JBoss Enterprise Application Platform 6.3

Guide de développement

À utiliser dans Red Hat JBoss Enterprise Application Platform 6

Last Updated: 2017-10-27

JBoss Enterprise Application Platform 6.3 Guide de développement

À utiliser dans Red Hat JBoss Enterprise Application Platform 6

Notice légale

Copyright © 2015 Red Hat, Inc..

This document is licensed by Red Hat under the Creative Commons Attribution-ShareAlike 3.0 Unported License. If you distribute this document, or a modified version of it, you must provide attribution to Red Hat, Inc. and provide a link to the original. If the document is modified, all Red Hat trademarks must be removed.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat Software Collections is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Résumé

Cet ouvrage procure des références et des exemples pour les développeurs Java EE 6 qui utilisent Red Hat JBoss Enterprise Application Platform 6, et inclut des correctifs publiés.

Table des matières

CHAPITRE 1. INTRODUCTION AU DÉVELOPPEMENT D'APPLICATIONS 13
1.1. INTRODUCTION 13
1.1.1. Red Hat JBoss Enterprise Application Platform 6 13
1.2. CONDITIONS PRÉALABLES 13
1.2.1. Familiarisez vous avec Java Enterprise Edition 6 13
1.2.1.1. Les Profils EE 6 13
1.2.1.2. Web Profil de Java Enterprise Edition 6 13
1.2.1.3. Java Enterprise Edition 6 Full Profile 14
1.2.2. Les modules et le système de chargement de la nouvelle classe modulaire de JBoss EAP 6. 15
1.2.2.1. Modules 15
1.3. INSTALLER L'ENVIRONNEMENT DE DÉVELOPPEMENT 16
1.3.1. Télécharger et installer Red Hat JBoss Developer Studio (JDBS) 16
1.3.1.1. Installer Red Hat JBoss Developer Studio (JDBS) 16
1.3.1.2. Téléchargez Red Hat JBoss Developer Studio 7.1 16
1.3.1.3. Installer Red Hat JBoss Developer Studio 7.1 17
1.3.1.4. Démarrer Red Hat JBoss Developer Studio 18
1.3.1.5. Ajouter le serveur de JBoss EAP en utilisant Define New Server 18
1.4. EXÉCUTER VOTRE PREMIÈRE APPLICATION 23
1.4.1. Télécharger les exemples de codes Quickstart 23
1.4.1.1. Accès aux Quickstarts 23
1.4.2. Exécuter les Quickstarts 24
1.4.2.1. Exécuter les quickstarts (démarrages rapides) dans Red Hat JBoss Developer Studio 24
1.4.2.2. Exécuter les quickstarts par la Ligne de commande 29
1.4.3. Revoir les tutoriels Quickstart 29
1.4.3.1. Découvrir le Quickstart HelloWorld 29
1.4.3.2. Découvrir le Quickstart numberguess 32
1.4.4. Remplacer l'application web Welcome par défaut 39

CHAPITRE 2. GUIDE MAVEN 40
2.1. POUR EN SAVOIR PLUS SUR MAVEN 40
2.1.1. Le référentiel Maven 40
2.1.2. Le fichier POM Maven 40
2.1.3. Conditions minimum pour un fichier POM Maven 41
2.1.4. Le fichier des configurations de Maven 41
2.2. INSTALLER LE RÉFÉRENTIEL JBOSS MAVEN ET MAVEN 42
2.2.1. Télécharger et installer Maven 42
2.2.2. Installer le référentiel Maven de JBoss EAP 6 43
2.2.3. Installer le référentiel Maven de JBoss EAP 6 localement 43
2.2.4. Installer le référentiel Maven de JBoss EAP 6 à utiliser avec Appache httpd 44
2.2.5. Installer le référentiel Maven de JBoss EAP 6 en utilisant le gestionnaire de référentiels Nexus Maven 45
2.2.6. Gestionnaires de référentiels Maven 46
2.3. UTILISER LE RÉFÉRENTIEL MAVEN 46
2.3.1. Configurer le référentiel Maven 6 de JBoss EAP 46
2.3.2. Configurer le référentiel JBoss EAP 6 Platform Maven Repository par les paramètres de configuration de Maven 47
2.3.3. Configurer Maven pour utilisation dans Red Hat JBoss Developer Studio 53
2.3.4. Configurer le référentiel JBoss EAP 6 Platform Maven Repository par le Projet POM 57
2.3.5. Gestion des dépendances du projet 59
Artéfacts Maven pris en charge 59
Gestion des dépendances 60

JBoss JavaEE Specs Bom de nomenclature 60
Les nomenclatures JBoss EAP BOM et les Quickstarts 61
JBoss Client BOMs 62
2.4. MISE À NIVEAU DU RÉFÉRENTIEL MAVEN 63
2.4.1. Appliquer un correctif dans le répertoire Maven local 63

CHAPITRE 3. CHARGEMENT DE CLASSES MODULAIRE ET MODULES 65
3.1. INTRODUCTION 65
3.1.1. Chargement des classes de chargement et de modules 65
3.1.2. Chargement des classes 65
3.1.3. Modules 65
3.1.4. Les dépendances de modules 66
3.1.5. Chargement des classes dans les déploiements 67
3.1.6. Précédence pour le chargement des classes 68
3.1.7. Nommage de modules dynamiques 69
3.1.8. jboss-deployment-structure.xml 69
3.2. AJOUTER UNE DÉPENDANCE DE MODULE EXPLICITE À UN DÉPLOIEMENT 69
3.3. GÉNÉRER DES ENTRÉES MANIFEST.MF EN UTILISANT MAVEN 71
3.4. EMPÊCHER QU'UN MODULE SOIT CHARGÉ IMPLICITEMENT 73
3.5. EXCLURE UN SOUS-SYSTÈME D'UN DÉPLOIEMENT 74
3.6. UTILISER LE CHARGEUR DE CLASSES PAR PROGRAMMATION DANS UN DÉVELOPPEMENT 75
3.6.1. Chargement des classes et des ressources par programmation dans le déploiement. 75
3.6.2. Itérer les ressources par programmation dans un déploiement. 77
3.7. CHARGEMENT DE CLASSE ET SOUS-DÉPLOIEMENTS 80
3.7.1. Chargement de classes et de modules dans les EAR (Archives Enterprise) 80
3.7.2. Isolement du chargeur de classes d'un sous-déploiement 81
3.7.3. Désactiver l'isolement du chargeur de classes d'un sous-déploiement dans un EAR 81
3.8. RÉFÉRENCE 82
3.8.1. Dépendances de modules implicites 82
3.8.2. Les modules inclus 85
3.8.3. Référence de descripteur de déploiement de structure de déploiement de JBoss 86

CHAPITRE 4. VALVES 87
4.1. VALVES 87
4.2. VALVES GLOBALES 87
4.3. LES VALVES D'AUTHENTIFICATION 87
4.4. CONFIGURER UNE APPLICATION WEB POUR UTILISER UNE VALVE. 87
4.5. CONFIGURER UNE APPLICATION WEB POUR QU'ELLE UTILISE UNE VALVE D'AUTHENTIFICATION 89
4.6. CRÉER UNE VALVE PERSONNALISÉE 90

CHAPITRE 5. LA JOURNALISATION POUR LES DÉVELOPPEURS 93
5.1. INTRODUCTION 93
5.1.1. Journalisation 93
5.1.2. Frameworks de journalisations d'applications pris en charge par JBoss LogManager 93
5.1.3. Niveaux de journalisation 93
5.1.4. Niveaux de journalisation pris en charge 94
5.1.5. Emplacements de fichiers de journalisation par défaut 95
5.2. JOURNALISATION DANS LE JBOSS LOGGING FRAMEWORK 95
5.2.1. JBoss Logging 95
5.2.2. Fonctionnalités de JBoss Logging 95
5.2.3. Ajouter une journalisation à une application par JBoss Logging 96
5.3. LA JOURNALISATION PAR DÉPLOIEMENT 98
5.3.1. La journalisation par déploiement 98

5.3.2. La journalisation par déploiement vers une application 98
5.3.3. Exemple de fichier logging.properties 99
5.4. PROFILS DE JOURNALISATION 99
5.4.1. Profils de journalisation 99
5.4.2. Spécifier un profil de journalisation dans une application 100

CHAPITRE 6. INTERNATIONALISATION ET LOCALISATION 102
6.1. INTRODUCTION 102
6.1.1. Internationalisation 102
6.1.2. Localisation 102
6.2. OUTILS JBOSS LOGGING 102
6.2.1. Aperçu 102
6.2.1.1. JBoss Logging Tools - Internationalisation et Localisation 102
6.2.1.2. JBoss Logging Tools Quickstart 103
6.2.1.3. Message Logger 103
6.2.1.4. Lot de messages 103
6.2.1.5. Messages de journalisation internationalisés 103
6.2.1.6. Exceptions internationalisées 103
6.2.1.7. Messages internationalisés 103
6.2.1.8. Fichiers de propriétés de traduction 104
6.2.1.9. Codes de projets de JBoss Logging Tools 104
6.2.1.10. Ids de messages de JBoss Logging Tools 104
6.2.2. Création de loggers, de messages ou d'exceptions internationalisés 104
6.2.2.1. Créer des messages log internationalisés 104
6.2.2.2. Créer et utiliser des messages internationalisés 106
6.2.2.3. Créer des exceptions personnalisées 107
6.2.3. Localisation de loggers, messages ou exceptions internationalisés 108
6.2.3.1. Générer des nouveaux fichiers de propriétés de traduction avec Maven 108
6.2.3.2. Traduire un logger, une exception ou un message internationalisé 110
6.2.4. Personnalisation des messages de journalisation internationalisés 111
6.2.4.1. Ajouter les ids de messages et les codes de projets aux messages de journalisation 111
6.2.4.2. Indiquer le niveau de journalisation d'un message 112
6.2.4.3. Personnaliser les messages de journalisation par des paramètres 112
6.2.4.4. Indiquer une exception comme cause d'un message de journalisation 113
6.2.5. Personnalisation des exceptions internationalisées 114
6.2.5.1. Ajouter les ids et les codes de projets aux messages d'exceptions 114
6.2.5.2. Personnaliser les messages d'exception avec des paramètres 116
6.2.5.3. Indiquer une exception comme cause d'une autre exception 117
6.2.6. Référence 118
6.2.6.1. Configuration Maven JBoss Logging Tools 118
6.2.6.2. Format de fichier de propriétés de traduction 119
6.2.6.3. Références aux annotations de JBoss Logging Tools 120

CHAPITRE 7. ENTERPRISE JAVABEANS 122
7.1. INTRODUCTION 122
7.1.1. Entreprise JavaBeans 122
7.1.2. Groupe de fonctionnalités EJB 3.1 122
7.1.3. EJB 3.1 Lite 123
7.1.4. Fonctionnalités EJB 3.1 Lite 123
7.1.5. Beans Enterprise 123
7.1.6. Écriture des beans Enterprise 124
7.1.7. Interfaces métier de Session Bean 124
7.1.7.1. Interfaces de métier Bean Enterprise 124

7.1.7.2. Interfaces de métier locales EJB 125
7.1.7.3. Interfaces métier à distance EJB 125
7.1.7.4. Beans EJB no-interface 125
7.2. CRÉER DES PROJETS ENTERPRISE BEAN 125
7.2.1. Créer un projet d'archives EJB avec le Red Hat Studio JBoss Developer 125
7.2.2. Créer un projet EJB Archive dans Maven. 129
7.2.3. Créer un projet EAR contenant un projet EJB 131
7.2.4. Ajouter un descripteur de déploiement à un projet EJB 134
7.3. SESSION BEANS 135
7.3.1. Session Beans 135
7.3.2. Stateless Session Beans 135
7.3.3. Stateful Session Beans 136
7.3.4. Singleton Session Beans 136
7.3.5. Ajouter des Session Beans à un Projet dans Red Hat JBoss Developer Studio 136
7.4. MESSAGE-DRIVEN BEANS 139
7.4.1. Message-Driven Beans 139
7.4.2. Adaptateurs de ressources 139
7.4.3. Créer un Message-Driven Bean basé JMS dans Red Hat JBoss Developer Studio 139
7.4.4. Activer la substitution de propriété des EJB et MDB dans une application 141
7.5. INVOQUER LES SESSION BEANS 145
7.5.1. Invoquer un session bean à distance avec JNDI 145
7.5.2. Contextes Client EJB 148
7.5.3. Considérations lors de l'utilisation d'un Contexte EJB Unique 148
7.5.4. Utiliser des Contextes Client EJB scoped 150
7.5.5. Configurer les EJB en utilisant un contexte client EJB scoped 151
7.5.6. Propriétés Client EJB 152
7.6. INTERCEPTEURS DE CONTENEURS 156
7.6.1. Intercepteurs de conteneurs 156
7.6.2. Créer une classe d'intercepteur de conteneur 157
7.6.3. Configurer un intercepteur de conteneur 157
7.6.4. Modifier l'identité du contexte de sécurité 159
7.6.5. Sécurité supplémentaire pour l'authentification EJB 163
7.6.6. Utiliser un intercepteur côté client dans une application 169
7.7. JAVABEANS ENTERPRISE CLUSTERISÉS 170
7.7.1. JavaBeans clusterisées (EJB) 170
7.7.2. Configuration de client EJB In-server ou Autonome 170
7.7.3. Mettre en place une politique d'équilibrage des charges presonnalisée pour les appels EJB 172
7.7.4. Comportement de transaction des invocations EJB 176
7.8. RÉFÉRENCE 179
7.8.1. Référence de nommage EJB JNDI 179
7.8.2. Résolution de référence EJB 180
7.8.3. Dépendances de projet pour les clients EJB distants 181
7.8.4. Référence de descripteur de déploiement jboss-ejb3.xml 182

CHAPITRE 8. LES SERVICES JBOSS MBEAN 185
8.1. RÉDIGER DES SERVICES JBOSS MBEAN 185
8.2. EXEMPLE DE MBEAN STANDARD 185
8.3. DÉPLOYER JBOSS MBEAN SERVICES 187

CHAPITRE 9. CLUSTERING DANS LES APPLICATIONS WEB 189
9.1. RÉPLIQUE DE SESSION 189
9.1.1. La réplique de session HTTP 189
9.1.2. Cache de session web 189

9.1.3. Configurer le cache de session web 189
9.1.4. Activer la copie de session pour votre application 190
9.2. PASSIVATION ET ACTIVATION HTTPSESSION 193
9.2.1. La passivation et l'activation de session HTTP 193
9.2.2. Configurer la passivation HttpSession dans votre application 194
9.3. DOMAINE DU COOKIE 196
9.3.1. Cookie Domain 196
9.3.2. Configurer Cookie Domain 196
9.4. IMPLÉMENTER UN SINGLETON HA 196
9.5. APPLICATION APACHE MOD_CLUSTER-MANAGER 202
9.5.1. L'application mod_cluster-manager 202
9.5.2. L'application mod_cluster-manager 202

CHAPITRE 10. CDI 205
10.1. CDI 205
10.1.1. CDI 205
10.1.2. CDI (Contexts and Dependency Injection) 205
10.1.3. Avantages de CDI 205
10.1.4. Injection de dépendance de Type-safe 206
10.1.5. Relation entre Weld, Seam 2, Seam 3, et JavaServer Faces 206
10.2. UTILISER CDI 206
10.2.1. Premières étapes 206
10.2.1.1. Activer CDI 206
10.2.2. Utiliser CDI pour développer une application 207
10.2.2.1. Utiliser le CDI pour développer une application 207
10.2.2.2. CDI avec le code existant 208
10.2.2.3. Exclure les beans du processus de balayage 208
10.2.2.4. Utiliser une injection pour étendre une implémentation 209
10.2.3. Dépendances ambigues ou non satisfaites 210
10.2.3.1. Dépendances ambigües et non-satisfaites 210
10.2.3.2. Qualificateurs 210
10.2.3.3. Utiliser un qualificateur pour résoudre une injection ambiguë 211
10.2.4. Beans gérés 212
10.2.4.1. Les beans gérés 212
10.2.4.2. Types de Classes qui ne sont pas des Beans 213
10.2.4.3. Utiliser le CDI pour injecter un objet dans un bean 213
10.2.5. Contextes, Scopes et Dépendances 215
10.2.5.1. Contextes et scopes 215
10.2.5.2. Contextes disponibles 215
10.2.6. Cycle de vie d'un bean 216
10.2.6.1. Gestion du cycle de vie d'un bean 216
10.2.6.2. Utilisation d'une méthode Producer 217
10.2.7. Beans nommés et beans alternatifs 218
10.2.7.1. Named Beans 218
10.2.7.2. Utilisation des Named Beans 218
10.2.7.3. Beans Alternative 219
10.2.7.4. Remplacer une injection par une alternative 220
10.2.8. Stérétypes 220
10.2.8.1. Stéréotypes 220
10.2.8.2. Utilisation des stéréotypes 221
10.2.9. Méthodes Observer 222
10.2.9.1. Méthodes Observer 222
10.2.9.2. Appliquer et observer des événements 222

10.2.10. Intercepteurs 223
10.2.10.1. Les intercepteurs 223
10.2.10.2. Utiliser les intercepteurs avec le CDI 224
10.2.11. Les décorateurs 225
10.2.12. Extensions portables 226
10.2.13. Proxies Beans 226
10.2.13.1. Proxys Bean 226
10.2.13.2. Utiliser un Proxy ou une Injection 227

CHAPITRE 11. JAVA TRANSACTION API (JTA) 228
11.1. APERÇU 228
11.1.1. Java Transactions API (JTA) 228
11.2. CONCEPTS DE TRANSACTIONS 228
11.2.1. Transactions 228
11.2.2. Les propriétés ACID de transactions 228
11.2.3. Coordinateur de transactions ou Gestionnaire de transactions 229
11.2.4. Participants à une transaction 229
11.2.5. JTA (Java Transactions API) 229
11.2.6. JTS (Java Transaction Service) 230
11.2.7. Sources de données XA et transactions XA 230
11.2.8. Recouvrement XA 230
11.2.9. Le protocole de validation en 2-Phases 231
11.2.10. Les délais d'attente des transactions 231
11.2.11. Les transactions distribuées 232
11.2.12. API de portabilité ORB 232
11.2.13. Transactions imbriquées 232
11.3. OPTIMISATIONS DES TRANSACTIONS 233
11.3.1. Optimisations de transactions 233
11.3.2. Optimisation LRCO pour une validation en 1 phase (1 PC) 233
11.3.2.1. Commit Markable Resource (CRM) 233
11.3.3. Optimisation Presumed-Abort 236
11.3.4. Optimisation Lecture-seule 236
11.4. RÉSULTATS DE TRANSACTIONS 236
11.4.1. Résultats de transactions 236
11.4.2. Transactions de validation 237
11.4.3. Transactions Roll-Back 237
11.4.4. Résultats heuristiques 237
11.4.5. Erreurs et exceptions pour les transactions JBoss 238
11.5. APERÇU SUR LES TRANSACTIONS JTA 238
11.5.1. JTA (Java Transactions API) 238
11.5.2. Cycle de vie d'une transaction JTA 238
11.6. CONFIGURATION DE SOUS-SYSTÈME DE TRANSACTION 239
11.6.1. Configuration des transactions 239
11.6.2. Configuration de source de données transactionnelle 239
11.6.2.1. Configurez votre base de données pour pouvoir utiliser les transactions JTA 239
11.6.2.2. Configuration d'une source de données XA 241
11.6.2.3. Se conncecter à la console de gestion 241
11.6.2.4. Créer une source de données non-XA par les interfaces de gestion 242
11.6.2.5. Paramètres de source de données 243
11.6.3. Journalisation des transactions 250
11.6.3.1. Messages de journalisation de transactions 250
11.6.3.2. Configurer la journalisation des sous-systèmes de transactions 252
11.6.3.3. Naviguer et gérer les transactions 253

11.7. UTILISER LES TRANSACTIONS JTA 257
11.7.1. Transactions JTA 257
11.7.2. Transactions de contrôle 257
11.7.3. Démarrer une transaction 258
11.7.4. Transactions imbriquées 259
11.7.5. Valider une transaction 259
11.7.6. Annuler une transaction 260
11.7.7. Gérer un résultat heuristique dans une transaction 262
11.7.8. Temps d'expiration des transactions 263
11.7.8.1. Les délais d'attente des transactions 263
11.7.8.2. Configurer le Transaction Manager (TM) (ou gestionnaire de transactions) 263
11.7.9. Gestion des erreurs de transactions JTA 268
11.7.9.1. Erreurs de transactions 268
11.8. CONFIGURATION ORB 269
11.8.1. CORBA (Common Object Request Broker Architecture) 269
11.8.2. Configurer l'ORB pour les transactions JTS 269
11.9. RÉFÉRENCES DE TRANSACTIONS 271
11.9.1. Erreurs et exceptions pour les transactions JBoss 271
11.9.2. Limitations sur les transactions JTA 271
11.9.3. Exemple de transaction JTA 271
11.9.4. Documentation API pour JBoss Transactions JTA 273
11.9.5. Limitations dans le processus de recouvrement XA 274

CHAPITRE 12. HIBERNATE 276
12.1. HIBERNATE CORE 276
12.2. JPA (JAVA PERSISTENCE API) 276
12.2.1. JPA 276
12.2.2. Hibernate EntityManager 276
12.2.3. Guide de départ 276
12.2.3.1. Créer un projet JPA dans le Red Hat JBoss Developer Studio 276
12.2.3.2. Créer le fichier de paramètres de persistance dans Red Hat JBoss Developer Studio 280
12.2.3.3. Exemple de fichier de configuration de persistance 282
12.2.3.4. Créer le fichier de configuration Hibernate dans Red Hat JBoss Developer Studio 282
12.2.3.5. Exemple de fichier de configuration Hibernate 283
12.2.4. Configuration 284
12.2.4.1. Propriétés de configuration de Hibernate 284
12.2.4.2. Hibernate JDBC et Propriétés de Connexion 286
12.2.4.3. Propriétés d'Hibernate Cache 287
12.2.4.4. Propriétés de transaction Hibernate 288
12.2.4.5. Propriétés Hibernate diverses 289
12.2.4.6. Dialectes SQL Hibernate 290
12.2.5. Caches de second niveau 292
12.2.5.1. Les caches de second niveau 292
12.2.5.2. Configurer un cache de second niveau pour Hibernate 293
12.3. ANNOTATIONS HIBERNATE 293
12.3.1. Annotations Hibernate 294
12.4. LANGAGE DE RECHERCHE HIBERNATE 298
12.4.1. Langage de recherche d'Hibernate 298
12.4.2. Déclarations HQL 299
12.4.3. La déclaration INSERT 299
12.4.4. La clause FROM 301
12.4.5. La clause WITH 301
12.4.6. Mises à jour, insertions et suppressions en bloc 301

12.4.7. Références de collection 303
12.4.8. Expressions de chemins qualifiées 304
12.4.9. Fonctions scalaires 305
12.4.10. Fonctions standardisées HQL 305
12.4.11. Opération de concaténation 306
12.4.12. Instanciation dynamique 306
12.4.13. Prédicats HQL 307
12.4.14. Comparaisons relationnelles 309
12.4.15. Prédicat IN 310
12.4.16. SQL Ordering 312
12.5. SERVICES HIBERNATE 312
12.5.1. Les services Hibernate 313
12.5.2. Contrats de service 313
12.5.3. Types de dépendances de service 313
12.5.4. Le ServiceRegistry 314
12.5.4.1. Le ServiceRegistry 314
12.5.5. Services personnalisés 314
12.5.5.1. Services personnalisés 314
12.5.6. Le registre Bootstrap 315
12.5.6.1. Le registre Boot-strap 316
12.5.6.2. Utilisation de BootstrapServiceRegistryBuilder 316
12.5.6.3. Les services BootstrapRegistry 316
12.5.7. Le registre SessionFactory 317
12.5.7.1. Le registre SessionFactory 317
12.5.7.2. Services SessionFactory 317
12.5.8. Intégrateurs 318
12.5.8.1. Intégrateurs 318
12.5.8.2. Cas d'utilisation d'Integrator 318
12.6. VALIDATION D'UN BEAN 319
12.6.1. Bean Validation 319
12.6.2. Hibernate Validator 319
12.6.3. Contraintes de validation 320
12.6.3.1. Les contraintes de validation 320
12.6.3.2. Créer une annotation de contrainte dans Red Hat JBoss Developer Studio 320
12.6.3.3. Créer une nouvelle classe Java dans Red Hat JBoss Developer Studio 322
12.6.3.4. Contraintes d'Hibernate Validator 322
12.6.4. Configuration 325
12.6.4.1. Exemple de fichier de configuration de validation 325
12.7. ENVERS 325
12.7.1. Hibernate Envers 325
12.7.2. Audit de classes persistantes 326
12.7.3. Stratégies d'auditing 326
12.7.3.1. Stratégies d'auditing 326
12.7.3.2. Définir la stratégie d'auditing 327
12.7.4. Introduction à l'auditing d'entités 327
12.7.4.1. Ajouter une support d'auditing à une entité JPA 327
12.7.5. Configuration 329
12.7.5.1. Configurer les paramètres Envers 329
12.7.5.2. Activer ou désactiver l'auditing en cours d'exécution 330
12.7.5.3. Configurer l'auditing conditionnel. 330
12.7.5.4. Propriétés de configuration Envers 331
12.7.6. Recherches 333
12.7.6.1. Suppression de l'information d'auditing 333

12.8. RÉGLAGE DE LA PERFORMANCE 337
12.8.1. Algorithmes de chargement de lots alternatifs 337
12.8.2. Mise en cache de second niveau d'objets de référence pour les données non mutables 338

CHAPITRE 13. SERVICES WEB JAX-RS 340
13.1. JAX-RS 340
13.2. RESTEASY 340
13.3. SERVICES WEB RESTFUL 340
13.4. ANNOTATIONS DÉFINIES RESTEASY 340
13.5. CONFIGURATION RESTEASY 343
13.5.1. Paramètres de configuration RESTEasy 343
13.6. JAX-RS WEB SERVICE SECURITY 344
13.6.1. Activez la sécurité basée-rôle pour RESTEasy JAX-RS Web Service 344
13.6.2. Sécuriser un service JAX-RS Web par des annotations 346
13.7. RESTEASY LOGGING 347
13.7.1. JAX-RS Web Service Logging 347
13.7.2. Catégories de journalisation définies dans RESTEasy 347
13.8. GESTION DES EXCEPTIONS 347
13.8.1. Créer un mappeur d'exceptions 347
13.8.2. Exceptions RESTEasy lancées en interne 348
13.9. INTERCEPTEURS RESTEASY 350
13.9.1. Interception des invocations JAX-RS 350
13.9.2. Lier un intercepteur à une méthode JAX-RS 352
13.9.3. Enregistrer un intercepteur 353
13.9.4. Familles de précédence d'intercepteur 353
13.9.4.1. Familles de précédence d'intercepteur 353
13.9.4.2. Définissez une famille de précédence d'intercepteurs personnalisée 354
13.10. ANNOTATIONS BASÉES CHAÎNE 355
13.10.1. Conversion des annotations basées @*Param en objects 355
13.11. CONFIGURATION DES EXTENSIONS DE FICHIERS 359
13.11.1. Mapper les extensions de fichiers avec les types de media dans le fichier web.xml 359
13.11.2. Mapper les extensions de fichiers vers les langues dans le fichier web.xml 359
13.11.3. Types de media supportés par RESTEasy 360
13.12. API JAVASCRIPT RESTEASY 361
13.12.1. API JavaScript RESTEasy 361
13.12.2. Activation du Servlet API JavaScript RESTEasy 361
13.12.3. Paramètres de l'API JavaScript RESTEasy 362
13.12.4. Créer des requêtes AJAX par l'API JavaScript 362
13.12.5. Membres de la classe REST.Request 363
13.13. SERVICE DE JOB ASYNCHRONE RESTEASY 364
13.13.1. Service Job Asynchrone RESTEasy 364
13.13.2. Activer le service de jobs asynchrones 365
13.13.3. Configurer les jobs asynchrones avec RESTEasy 365
13.13.4. Paramètres de configuration de Service Job Asynchrone 367
13.14. RESTEASY JAXB 368
13.14.1. Créer un décorateur JAXB 368
13.15. RESTEASY ATOM SUPPORT 369
13.15.1. Fournissseur et API Atom 369
13.16. RESTEASY/SPRING INTEGRATION 370
13.16.1. Intégration RESTEasy/Spring 370

CHAPITRE 14. SERVICES WEB JAX-WS 372
14.1. SERVICES WEB JAX-WS 372

14.2. CONFIGURER LE SOUS-SYSTÈME DES SERVICES WEB 373
14.3. PONTS DE TERMINAISON DE SERVICES WEB JAX-WS 376
14.3.1. Les points de terminaison de Services Web JAX-WS 376
14.3.2. Écrire et déployer un point de terminaison de Service Web JAX-WS 379
14.4. CLIENTS DU SERVICE JAX-WS WEB 381
14.4.1. Consommer et accéder à un Service Web JAX-WS 381
14.4.2. Développer une application client JAX-WS 385
14.5. RÉFÉRENCE DE DÉVELOPPEMENT JAX-WS 391
14.5.1. Activation de WS-Addressing (Web Services Addressing) 391
14.5.2. Référence API Commun JAX-WS 393

CHAPITRE 15. WEBSOCKETS 397
15.1. WEBSOCKETS 397
15.2. CRÉER UNE APPLICATION WEBSOCKET 397

CHAPITRE 16. IDENTITÉ AU SEIN D'APPLICATIONS 404
16.1. CONCEPTS DE BASE 404
16.1.1. Cryptage 404
16.1.2. Les domaines de sécurité 404
16.1.3. Cryptage SSL 404
16.1.4. Sécurité déclarative 405
16.2. SÉCURITÉ BASÉE-RÔLE POUR LES APPLICATIONS 405
16.2.1. La sécurité des applications 405
16.2.2. Authentification 406
16.2.3. L'autorisation 406
16.2.4. Security Auditing 406
16.2.5. Security Mapping 406
16.2.6. Java Authentication et Authorization Service (JAAS) 407
16.2.7. Java Authentication et Authorization Service (JAAS) 407
16.2.8. Utiliser un domaine de sécurité dans votre application 408
16.2.9. Utilisation de la sécurité basée-rôle dans les Servlets 411
16.2.10. Utilisation d'une authentification système de tierce partie pour votre application 413
16.3. DOMAINES DE SÉCURITÉ 420
16.3.1. Domaines de sécurité 420
16.3.2. Ajout d'un domaine de sécurité 420
16.3.3. Ajout d'un utilisateur à un domaine de sécurité 421
16.4. SÉCURITÉ DES APPLICATIONS EJB 422
16.4.1. Identité Sécurité 422
16.4.1.1. L'identité de sécurité EJB 422
16.4.1.2. Définir l'identité de sécurité d'un EJB 422
16.4.2. Permissions de méthodes EJB 423
16.4.2.1. Permissions de méthodes EJB 423
16.4.2.2. Utilisation des permissions de méthodes EJB 424
16.4.3. Annotations de sécurité EJB 427
16.4.3.1. Les annotations de sécurité EJB 427
16.4.3.2. Utilisation des annotations de sécurité EJB 427
16.4.4. Accès à distance aux EJB 428
16.4.4.1. Remote Method Access 428
16.4.4.2. Remoting Callbacks 429
16.4.4.3. Remoting Server Detection 430
16.4.4.4. Configurer le sous-système de JBoss Remoting 430
16.4.4.5. Utilisation des domaines de sécurité avec les clients EJB distants 438
16.4.4.6. Ajout d'un domaine de sécurité 439

16.4.4.7. Ajout d'un utilisateur à un domaine de sécurité 440
16.4.4.8. Accès EJB à distance utilisant le cryptage SSL 440
16.5. SÉCURITÉ DES APPLICATIONS JAX-RS 440
16.5.1. Activez la sécurité basée-rôle pour RESTEasy JAX-RS Web Service 440
16.5.2. Sécuriser un service JAX-RS Web par des annotations 442
16.6. PROTOCOLE MOTS DE PASSES DISTANTS SÉCURISÉS 443
16.6.1. Protocole pour mots de passes distants sécurisés (SRP pour Secure Remote Password) 443
16.6.2. Configuration du protocole SRP (Secure Remote Password) 443
16.7. L'ARCHIVAGE SÉCURISÉ DES MOTS DE PASSE POUR LES STRINGS DE NATURE CONFIDENTIELLE 445
16.7.1. Sécurisation des chaînes confidentielles de fichiers en texte clair 445
16.7.2. Créer un keystore Java pour stocker des strings sensibles 446
16.7.3. Masquer le mot de passe du keystore et initialiser le mot de passe de l'archivage de sécurité 449
16.7.4. Configurer JBoss EAP pour qu'il utilise l'archivage sécurisé des mots de passe 450
16.7.5. Configurer JBoss EAP pour qu'il utilise une implémentation d'archivage sécurisé personnalisée 452
16.7.6. Stocker et résoudre les strings sensibles cryptés du keystore Java. 452
16.7.7. Stocker et résoudre des strings sensibles de vos applications 455
16.8. JAVA AUTHORIZATION CONTRACT FOR CONTAINERS (JACC) 457
16.8.1. Java Authorization Contract for Containers (JACC) 457
16.8.2. Configurer la sécurité JACC (Java Authorization Contract for Containers) 458
16.9. JASPI (JAVA AUTHENTICATION SPI FOR CONTAINERS) 459
16.9.1. Sécurité Java Authentication SPI pour Conteneurs (JASPI) 459
16.9.2. Configuration de la sécurité Java Authentication SPI pour conteneurs (JASPI) 459

CHAPITRE 17. SSO (SINGLE SIGN ON) 461
17.1. SSO (SINGLE SIGN ON) POUR LES APPLICATIONS WEB 461
17.2. SSO (SINGLE SIGN ON) CLUSTERISÉES POUR LES APPLICATIONS WEB 461
17.3. CHOISIR L'IMPLÉMENTATION SSO QUI VOUS CONVIENT 462
17.4. UTILISATION DE SSO (SINGLE SIGN ON) POUR LES APPLICATIONS WEB 462
17.5. KERBEROS 464
17.6. SPNEGO 465
17.7. MICROSOFT ACTIVE DIRECTORY 465
17.8. CONFIGURATION DE KERBEROS OU MICROSOFT ACTIVE DIRECTORY DESKTOP SSO POUR LES APPLICATIONS WEB 465
17.9. CONFIGURER SPNEGO AVEC UN RENVOI À L'AUTHENTIFICATION FORM 469

CHAPITRE 18. RÉFÉRENCES DE SÉCURITÉ POUR LE DÉVELOPPEMENT 471
18.1. RÉFÉRENCE DE CONFIGURATION JBOSS-WEB.XML 471
18.2. RÉFÉRENCE DE PARAMÈTRE DE SÉCURITÉ EJB 474

CHAPITRE 19. RÉFÉRENCES SUPPLÉMENTAIRES 476
19.1. TYPES D'ARCHIVES JAVA 476

ANNEXE A. HISTORIQUE DE RÉVISION 478

CHAPITRE 1. INTRODUCTION AU DÉVELOPPEMENT D'APPLICATIONS

1.1. INTRODUCTION

1.1.1. Red Hat JBoss Enterprise Application Platform 6

Red Hat JBoss Enterprise Application Platform 6 (JBoss EAP 6) est une plate-forme middleware construite sur la base de standards ouverts et compatibles avec Java Enterprise Edition 6. Elle intègre JBoss Application Server 7 avec un clustering de haute disponibilité, une messagerie, une mise en cache distribuée et autres technologies.

JBoss EAP 6 comprend une nouvelle structure modulaire qui permet aux services d'être activés seulement si nécessaire, améliorant ainsi la vitesse de démarrage.

La console de gestion et l'interface CLI rendent la modification des fichiers de configuration XML inutile et rajoutent la capacité d'encoder et d'automatiser des tâches.

En plus, JBoss EAP 6 comprend des frameworks de développement et des API pour développer rapidement des applications de Java EE sécurisées et évolutives.

Rapporter un bogue

1.2. CONDITIONS PRÉALABLES

1.2.1. Familiarisez vous avec Java Enterprise Edition 6

1.2.1.1. Les Profils EE 6

Java Enterprise Edition 6 (EE 6) inclut un support pour des profils multiples, ou des sous-ensembles d'API. Les deux seuls profils que la spécification EE6 définit sont *Full Profile* et *Web Profile*.

EE 6 Full Profile inclut tous les API et toutes les spécifications incluses dans la spécification EE 6. EE 6 Web Profile inclut un sous-ensemble des APIs qui sont utiles aux développeurs web.

JBoss EAP 6 est une implémentation certifiée de Java Enterprise Edition 6 Full Profile et des spécifications Web Profile.

- Section 1.2.1.2, « Web Profil de Java Enterprise Edition 6 »
- Section 1.2.1.3, « Java Enterprise Edition 6 Full Profile »

Rapporter un bogue

1.2.1.2. Web Profil de Java Enterprise Edition 6

Le Web Profil est l'un des deux profils définis dans la spécification Java Enterprise Edition 6. Il a été conçu pour le développement d'une application web. L'autre profil défini par la spécification Java Enterprise Edition 6 est le Full Profile. Pour plus d'informations, voir Section 1.2.1.3, « Java Enterprise Edition 6 Full Profile ».

Conditions préalables de Java EE 6 Web Profile

- Java Platform, Enterprise Edition 6

- **Technologies Java Web**
  - Servlet 3.0 (JSR 315)
  - JSP 2.2 et Expression Language (EL) 1.2
  - JavaServer Faces (JSF) 2.1 (JSR 314)
  - Java Standard Tag Library (JSTL) for JSP 1.2
  - Débogage du support pour les autres langages 1.0 (JSR 45)

- **Enterprise Application Technologies**
  - Contexts and Dependency Injection (CDI) (JSR 299)
  - Injection de dépendance dans Java (JSR 330)
  - Enterprise JavaBeans 3.1 Lite (JSR 318)
  - Java Persistence API 2.0 (JSR 317)
  - Annotations communes de la Plateforme Java 1.1 (JSR 250)
  - Java Transaction API (JTA) 1.1 (JSR 907)
  - Bean Validation (JSR 303)

Rapporter un bogue

1.2.1.3. Java Enterprise Edition 6 Full Profile

La spécification Java Enterprise Edition 6 (EE 6) définit un concept de profil, et en définit deux d'entre eux dans le cadre de la spécification. En plus des items supportés dans le Java Enterprise Edition 6 Web Profile (Section 1.2.1.2, « Web Profil de Java Enterprise Edition 6 »), le Full Profile supporte les API suivants. JBoss Enterprise Edition 6 supporte le Full Profile.

Items inclus dans EE 6 Full Profile

- EJB 3.1 (not Lite) (JSR 318)
- Java EE Connector Architecture 1.6 (JSR 322)
- Java Message Service (JMS) API 1.1 (JSR 914)
- JavaMail 1.4 (JSR 919)

- **Technologies Service Web**
  - Jax-RS RESTful Web Services 1.1 (JSR 311)
  - Implémentation d'Enterprise Web Services 1.3 (JSR 109)
  - JAX-WS Java API for XML-Based Web Services 2.2 (JSR 224)

- Java Architecture pour XML Binding (JAXB) 2.2 (JSR 222)
- Web Services Metadata pour Java Platform (JSR 181)
- Les API Java pour XML-based RPC 1.1 (JSR 101)
- Les API Java pour XML Messaging 1.3 (JSR 67)
- Java API pour les Registres XML (JAXR) 1.0 (JSR 93)

- **Technologies de Gestion et de Sécurité**
  - Interface Fournisseur Service Java Authentification pour Conteneurs 1.0 (JSR 196)
  - Contrat Java Authentication Contract pour les Conteneurs 1.3 (JSR 115)
  - Déploiement Application Java EE 1.2 (JSR 88)
  - J2EE Management 1.1 (JSR 77)

Rapporter un bogue

1.2.2. Les modules et le système de chargement de la nouvelle classe modulaire de JBoss EAP 6.

1.2.2.1. Modules

Un module est un regroupement logique des classes utilisées pour le chargement de classes et pour la gestion des dépendances. JBoss EAP 6 identifie deux types de modules, parfois appelés modules statiques et dynamiques. Cependant, la seule différence entre les deux est la façon dont ils sont empaquetés. Tous les modules offrent les mêmes caractéristiques.

Modules statiques

Les modules statiques sont prédéfinis dans le répertoire ***EAP_HOME*/modules/** du serveur d'applications. Chaque sous-répertoire représente un module et contient un fichier de configuration (**module.xml**) et tout fichier JAR requis. Le nom du module est défini dans le fichier **module.xml**. Tous les API fournis par le serveur de l'application sont des modules statiques, y compris les API Java EE, et les autres API comme JBoss Logging.

Exemple 1.1. Exemple de fichier module.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<module xmlns="urn:jboss:module:1.0" name="com.mysql">
  <resources>
    <resource-root path="mysql-connector-java-5.1.15.jar"/>
  </resources>
  <dependencies>
    <module name="javax.api"/>
    <module name="javax.transaction.api"/>
  </dependencies>
</module>
```

Le nom du module, **com.mysql**, doit correspondre à la structure du répertoire du module.

Les modules fournis dans les distributions JBoss EAP se trouvent dans un répertoire **system** se trouvant lui-même dans le répertoire ***JBOSS_HOME*/modules**. Cela les rend séparés de tout module fourni par une tierce partie.

Tout produit mis en couche de Red Hat, se superposant sur JBoss EAP 6.1 ou version supérieure installera également leurs modules dans le répertoire **system**.

La création de modules statiques personnalisés peut être utile si plusieurs applications sont déployées sur un même serveur utilisant les mêmes bibliothèques de tierce partie. Au lieu d'un regroupement de ces bibliothèques pour chaque application, un module contenant ces bibliothèques peut être créé et installé par l'administrateur JBoss. Les applications peuvent ensuite déclarer une dépendance explicite sur les modules statiques personnalisés.

Les utilisateurs doivent s'assurer que les modules personnalisés soient installés dans le répertoire ***JBOSS_HOME*/modules**, en utilisant un répertoire par couche de modules. Cela garantit que les versions personnalisées de modules qui existent déjà dans le répertoire **system** soient bien chargées à la place des versions fournies. Ainsi, les modules utilisateur auront la priorité sur les modules fournis par le système.

Si vous utilisez la variable d'environnement **JBOSS_MODULE_PATH** pour changer les emplacements où JBoss EAP cherche les modules, le produit ira chercher dans une structure de sous-répertoire **system** dans un des emplacements spécifiés. Une structure de sous-répertoire **system** doit exister quelquepart dans les emplacements spécifiés dans **JBOSS_MODULE_PATH**.

Modules dynamiques

Les modules dynamiques sont créés et chargés par le serveur d'application pour chaque déploiement JAR ou WAR (ou sous-déploiement d'un EAR). Le nom d'un module dynamique est dérivé du nom de l'archive déployée. Comme les déploiements sont chargés sous forme de modules, ils peuvent configurer des dépendances et peuvent être utilisés comme dépendances par d'autres déploiements.

Les modules ne sont chargés qu'en fonction des besoins. Cela a généralement lieu quand une application est déployée avec des dépendances implicites ou explicites.

Rapporter un bogue

1.3. INSTALLER L'ENVIRONNEMENT DE DÉVELOPPEMENT

1.3.1. Télécharger et installer Red Hat JBoss Developer Studio (JDBS)

1.3.1.1. Installer Red Hat JBoss Developer Studio (JDBS)

1. Section 1.3.1.2, « Téléchargez Red Hat JBoss Developer Studio 7.1 »
2. Section 1.3.1.3, « Installer Red Hat JBoss Developer Studio 7.1 »
3. Section 1.3.1.4, « Démarrer Red Hat JBoss Developer Studio »
4. Section 1.3.1.5, « Ajouter le serveur de JBoss EAP en utilisant Define New Server »

Rapporter un bogue

1.3.1.2. Téléchargez Red Hat JBoss Developer Studio 7.1

1. Visitez https://access.redhat.com/.
2. Sélectionnez **Téléchargements** du menu en haut de la page.
3. Cherchez **`Red Hat JBoss Developer Studio`** dans la liste et cliquez dessus.
4. Sélectionnez la version appropriée et cliquez sur **`Télécharger`**.

Rapporter un bogue

1.3.1.3. Installer Red Hat JBoss Developer Studio 7.1

Conditions préalables :

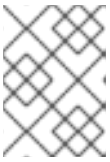
Section 1.3.1.2, « Téléchargez Red Hat JBoss Developer Studio 7.1 »

Procédure 1.1. Installer Red Hat JBoss Developer Studio 7.1

1. Ouvrir un terminal.
2. Aller dans le répertoire qui contient le fichier téléchargé **`.jar`**.
3. Exécuter la commande suivante pour lancer le GUI d'installation.

```
java -jar jbdevstudio-build_version.jar
```

4. Cliquer sur **`Suivant`** pour commencer le processus d'installation.
5. Sélectionner **`I accept the terms of this license agreement`** (J'accepte les conditions de licence) et cliquer sur **`Suivant`**.
6. Ajuster le chemin d'accès de l'installation et cliquer sur **`Next`**.



NOTE

Si le dossier de chemin d'installation n'existe pas, vous verrez une invite. Cliquer alors sur **`Ok`** pour créer le dossier.

7. Choisir une JVM, ou bien conserver la JVM sélectionnée par défaut, et cliquer sur **`Next`**.
8. Ajouter une plateforme d'applications disponible, et cliquer sur **`Next`**.
9. Vérifier les informations d'installation et cliquer sur **`Next`**.
10. Cliquer sur **`Next`** une fois le processus d'installation terminé.
11. Configurer les raccourcis bureau pour Red Hat JBoss Developer Studio et cliquer sur **`Next`** (suivant).
12. Cliquer sur le bouton **`Done`**.

Rapporter un bogue

1.3.1.4. Démarrer Red Hat JBoss Developer Studio

Conditions préalables :

Section 1.3.1.3, « Installer Red Hat JBoss Developer Studio 7.1 »

Procédure 1.2. Commande de démarrage de Red Hat JBoss Developer Studio

1. Ouvrir un terminal.
2. Aller dans le répertoire d'installation.
3. Lancer la commande suivante pour démarrer JBoss Developer Studio :

```
[localhost]$ ./jbdevstudio
```

Rapporter un bogue

1.3.1.5. Ajouter le serveur de JBoss EAP en utilisant Define New Server

Ces instructions assument qu'il s'agit d'une première introduction à Red Hat JBoss Developer Studio 7.x et que vous n'avez pas encore ajouté de serveurs JBoss EAP. La procédure ci-dessous comprend l'ajout d'un serveur par l'assistant Define New Server.

Procédure 1.3. Ajouter le serveur

1. Ouvrir l'onglet **`Servers`**. S'il n'y a pas d'onglet **`Servers`**, l'ajouter au panneau comme suit :
   a. Cliquer sur **Window** → **Show View** → **Other...**.
   b. Sélectionner **`Serveurs`** à partir du dossier **`Server`** et cliquer sur **`OK`**.
2. Cliquer sur le lien **`create a new server`** ou bien cliquer à droite dans le panneau vide Serveur, et sélectionner **New** → **Server**.

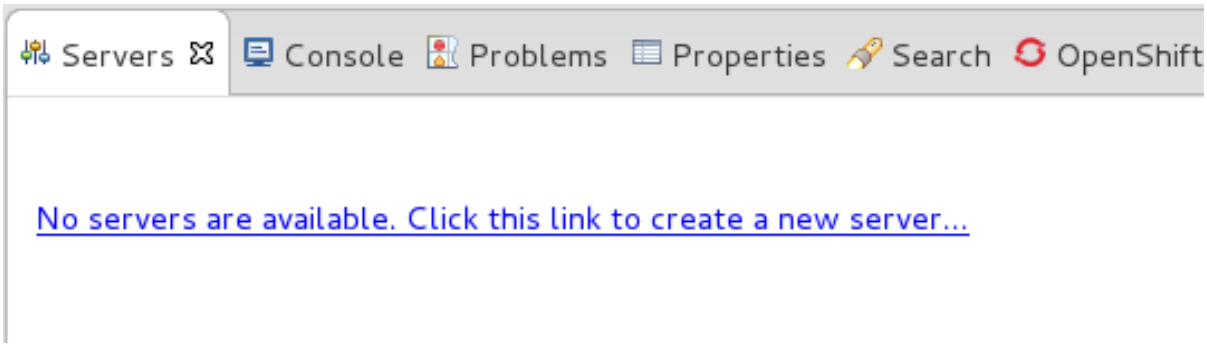


Figure 1.1. Ajouter un nouveau serveur - Aucun serveur disponible

3. Étendre **`JBoss Enterprise Middleware`** et choisir **JBoss Enterprise Application Platform 6.1+**. Puis, cliquer sur **`Next`** pour créer le JBoss Runtime et définir le serveur. La prochaine fois que vous définirez un nouveau serveur, ce dialogue affichera un menu de sélection **`Server runtime environment`** avec la nouvelle définition de runtime.

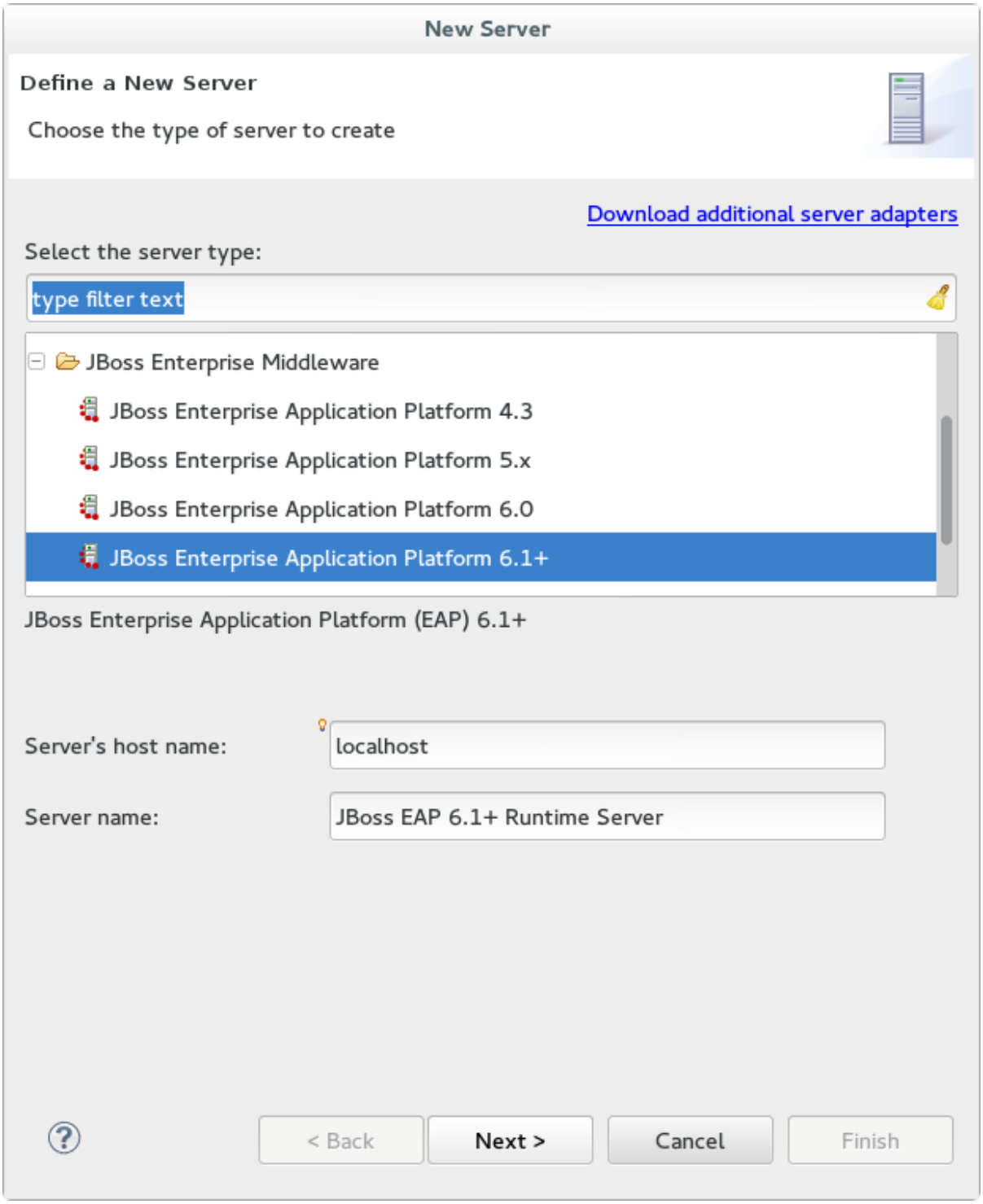


Figure 1.2. Définir un nouveau serveur

4. Saisir un nom comme "JBoss EAP 6.3 Runtime". Sous **`Home Directory`**, cliquer sur **`Browse`** puis, naviguer vers l'emplacement de l'installation de JBoss EAP. Puis, cliquer sur **`Next`**.

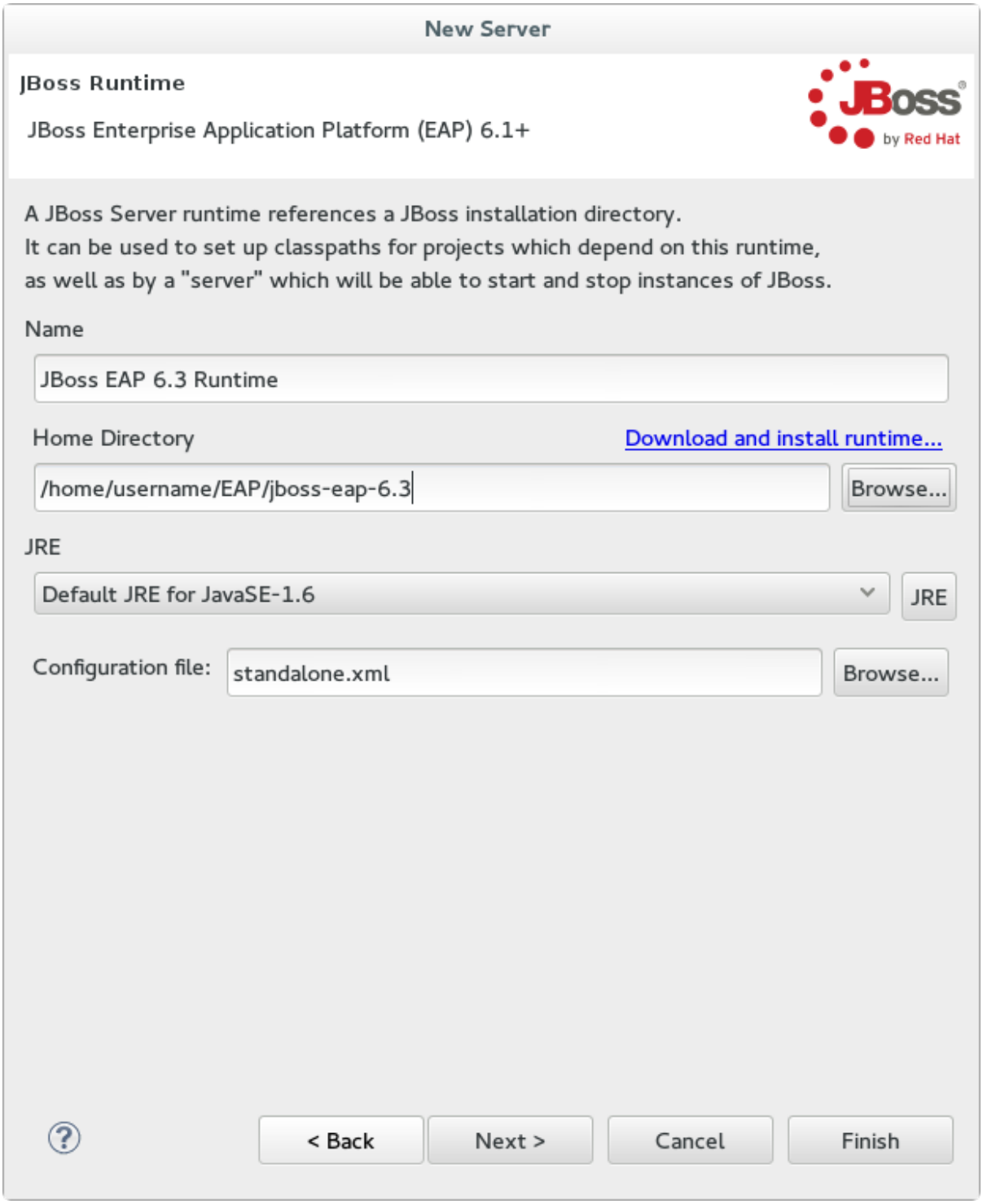


Figure 1.3. Ajouter un environnement de runtime du nouveau serveur

5. Sur cet écran, vous définissez le comportement de serveur. Vous pouvez démarrer le serveur manuellement ou laisser Red Hat JBoss Developer Studio le gérer pour vous. Vous pouvez également définir un serveur distant pour le déploiement et déterminer si vous souhaitez exposer le port de gestion pour ce serveur, par exemple, si vous avez besoin de le connecter en utilisant JMX. Dans cet exemple, supposons que le serveur est local et que vous souhaitiez que Red Hat JBoss Developer Studio gère votre serveur sans que vous ayiez besoin de vérifier quoi que ce soit. Cliquer sur **`Next`**.

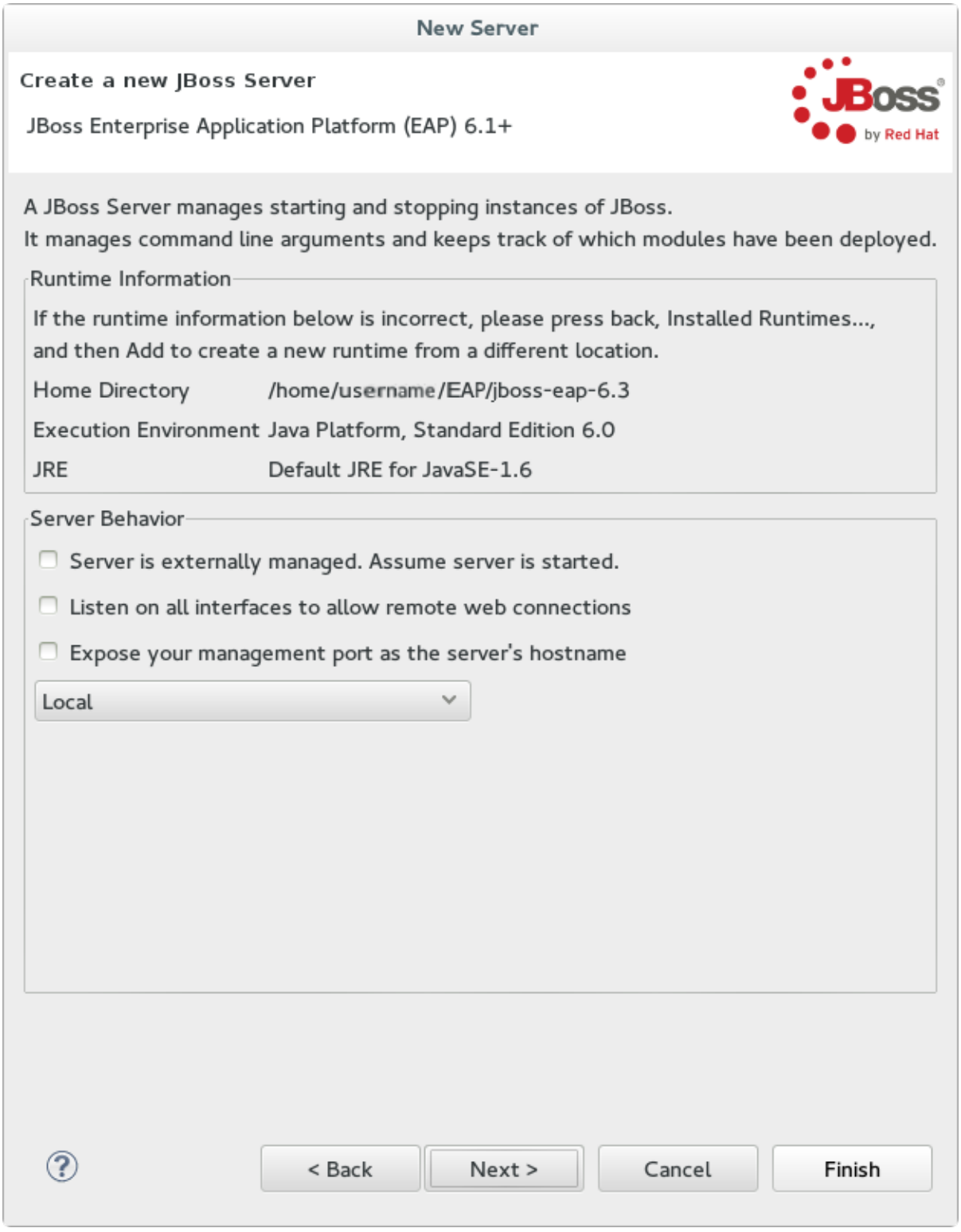


Figure 1.4. Définir le nouveau comportement du serveur de JBoss

6. Cet écran vous permet de configurer les projets existants pour le nouveau serveur. Puisque vous n'avez pas de projets à ce stade, cliquez sur **`Terminé`**.

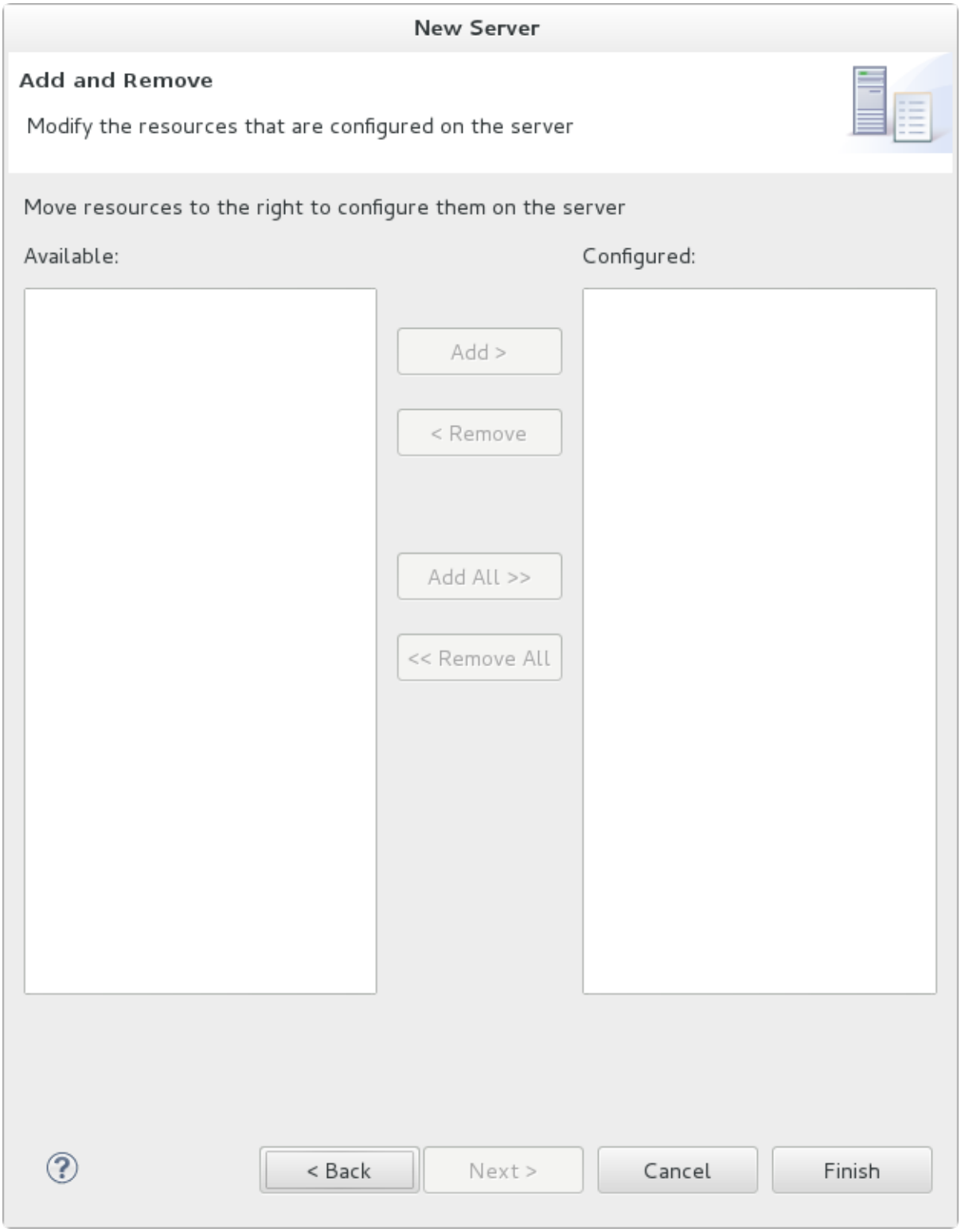


Figure 1.5. Modifier les ressources dans le nouveau serveur de JBoss

Résultat

Le serveur de JBoss EAP est listé dans l'onglet **`Servers`**.

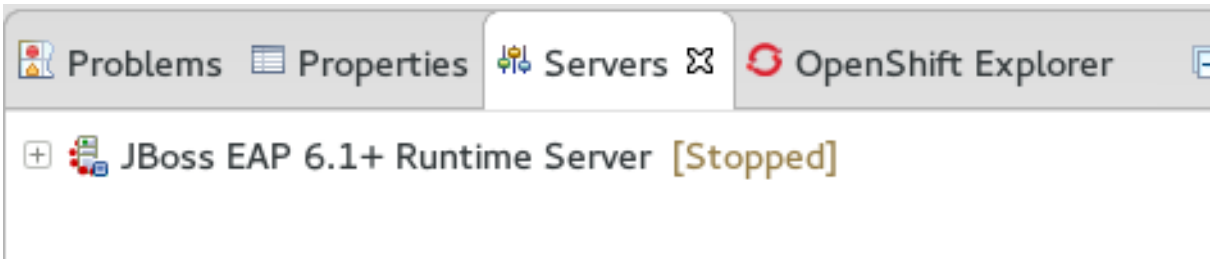


Figure 1.6. Le server apparaît sur la liste de serveurs

Rapporter un bogue

1.4. EXÉCUTER VOTRE PREMIÈRE APPLICATION

1.4.1. Télécharger les exemples de codes Quickstart

1.4.1.1. Accès aux Quickstarts

Résumé

JBoss EAP 6 contient une série d'exemples quickstart conçus pour aider les utilisateurs à commencer à rédiger des applications en utilisant les technologies Java EE 6.

Conditions préalables

- Maven 3.0.0 ou versions plus récentes. Pour en savoir plus sur l'installation de Maven, veuillez consulter http://maven.apache.org/download.html.
- Section 2.1.1, « Le référentiel Maven »
- Le référentiel JBoss EAP 6.3 Maven est disponible en ligne, donc il n'est pas utile de le télécharger ou de l'installer localement. Si vous envisagez de l'utiliser en ligne, passer à l'étape suivante. Si vous préférez l'installer dans un référentiel local, consulter : Section 2.2.3, « Installer le référentiel Maven de JBoss EAP 6 localement ».
- Section 2.3.2, « Configurer le référentiel JBoss EAP 6 Platform Maven Repository par les paramètres de configuration de Maven »

Procédure 1.4. Télécharger les Quickstarts

1. Ouvrir un navigateur web, et accéder à cet URL: https://access.redhat.com/jbossnetwork/restricted/listSoftware.html?product=appplatform.
2. Trouver "Quickstarts" dans la liste.
3. Cliquer sur le bouton **`Downloads`** pour télécharger un fichier ZIP contenant les exemples.
4. Décompresser l'archive dans un répertoire de votre choix.

Résultat

Les exemples de Java EE Quickstart ont été téléchargés et décompressés. Veuillez consulter le fichier **`README.md`** dans le répertoire supérieur des archives Quickstart pour des instructions concernant le déploiement de chaque quickstart.

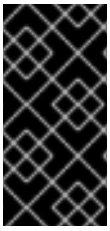
Rapporter un bogue

1.4.2. Exécuter les Quickstarts

1.4.2.1. Exécuter les quickstarts (démarrages rapides) dans Red Hat JBoss Developer Studio

Procédure 1.5. Importer les quickstarts dans Red Hat JBoss Developer Studio

Chaque quickstart est fourni avec un fichier POM (Project Object Model) qui contient des informations de projet et de configuration de quickstart. À l'aide de ce fichier POM, vous pouvez facilement importer le quickstart dans Red Hat JBoss Developer Studio.



IMPORTANT

Si votre dossier de projet quickstart se situe dans l'espace de travail IDE, lorsque vous l'importez dans Red Hat JBoss Developer Studio, l'IDE génère un nom de projet non valide et un nom d'archive WAR. N'oubliez pas que votre dossier de projet quickstart doit être situé en dehors de l'espace de travail IDE avant de commencer !

1. Si vous ne l'avez pas encore fait, Section 2.3.2, « Configurer le référentiel JBoss EAP 6 Platform Maven Repository par les paramètres de configuration de Maven ».
2. Démarrer Red Hat JBoss Developer Studio.
3. À partir du menu, sélectionner **Fichier** → **Importer**.
4. Dans la liste sélectionnée, choisir **Maven** → **Projets Maven existants**, puis cliquer sur **`Suivant`**.

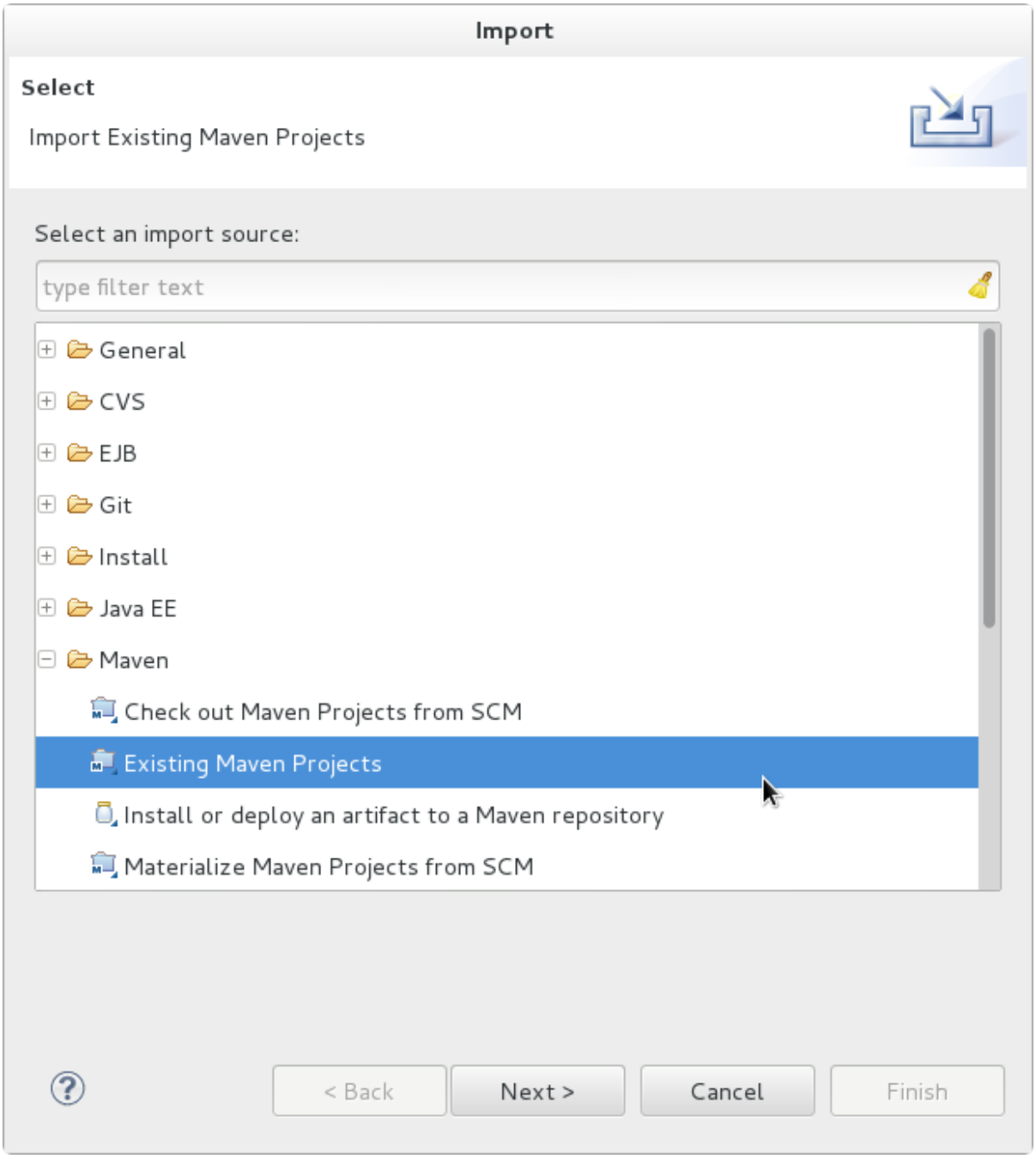


Figure 1.7. Importer les projets Maven existants

5. Naviguer vers le répertoire du quickstart que vous souhaitez tester, comme **`helloworld`**, et cliquer sur **`OK`**. La zone de liste **`Projects`** verra apparaître le fichier **`pom.xml`** du projet quickstart sélectionné.

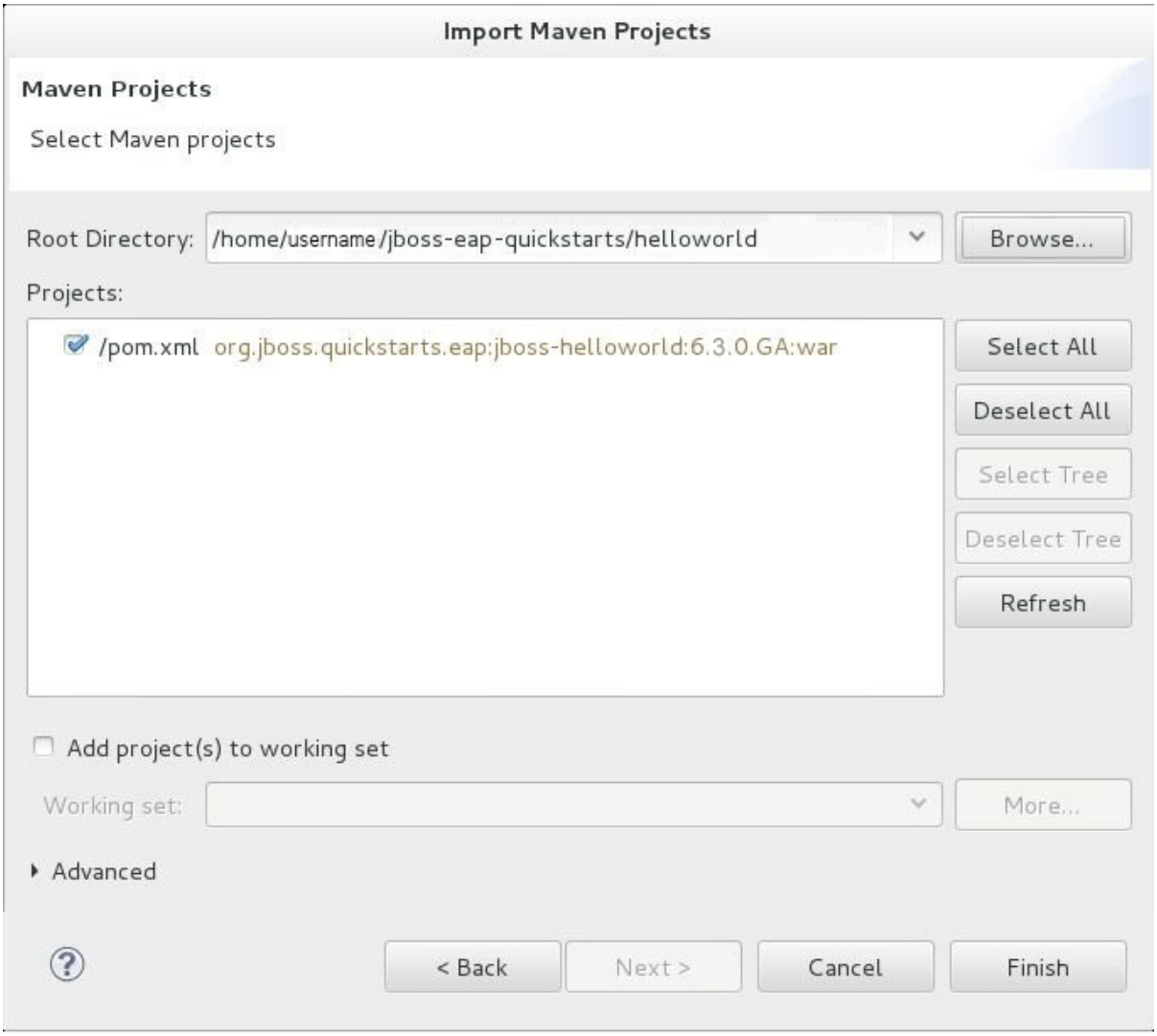


Figure 1.8. Sélectionner les projets Maven

6. Cliquer sur **Terminé**.

Procédure 1.6. Générer et déployer le Quickstart `helloworld`

Le Quickstart **helloworld** est un des quickstarts les plus simples et représente une bonne façon de vérifier que le serveur JBoss est configuré et exécute correctement.

1. Si vous ne voyez pas un onglet **Servers**, ajoutez le au panneau comme suit :
   a. Cliquer sur **Window** → **Show View** → **Other...**.
   b. Sélectionner **Servers** à partir du dossier **Server** et cliquer sur **OK**.
2. Cliquer à droite sur **jboss-helloworld** sur l'onglet **Project Explorer**, puis sélectionner **Run As**. On vous présentera une liste de choix. Sélectionner **Run on Server**.

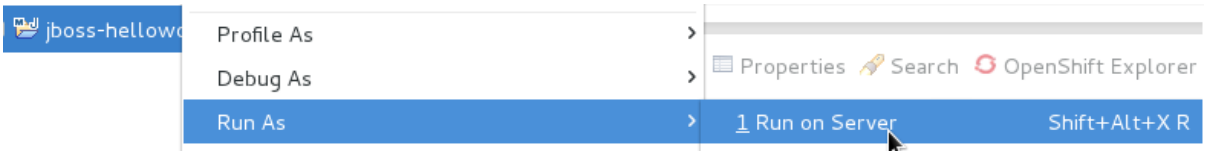


Figure 1.9. Run As - Run on Server

3. Sélectionner **`jboss-eap-6.3`** de la liste de serveurs, et cliquer sur **`Next`**.

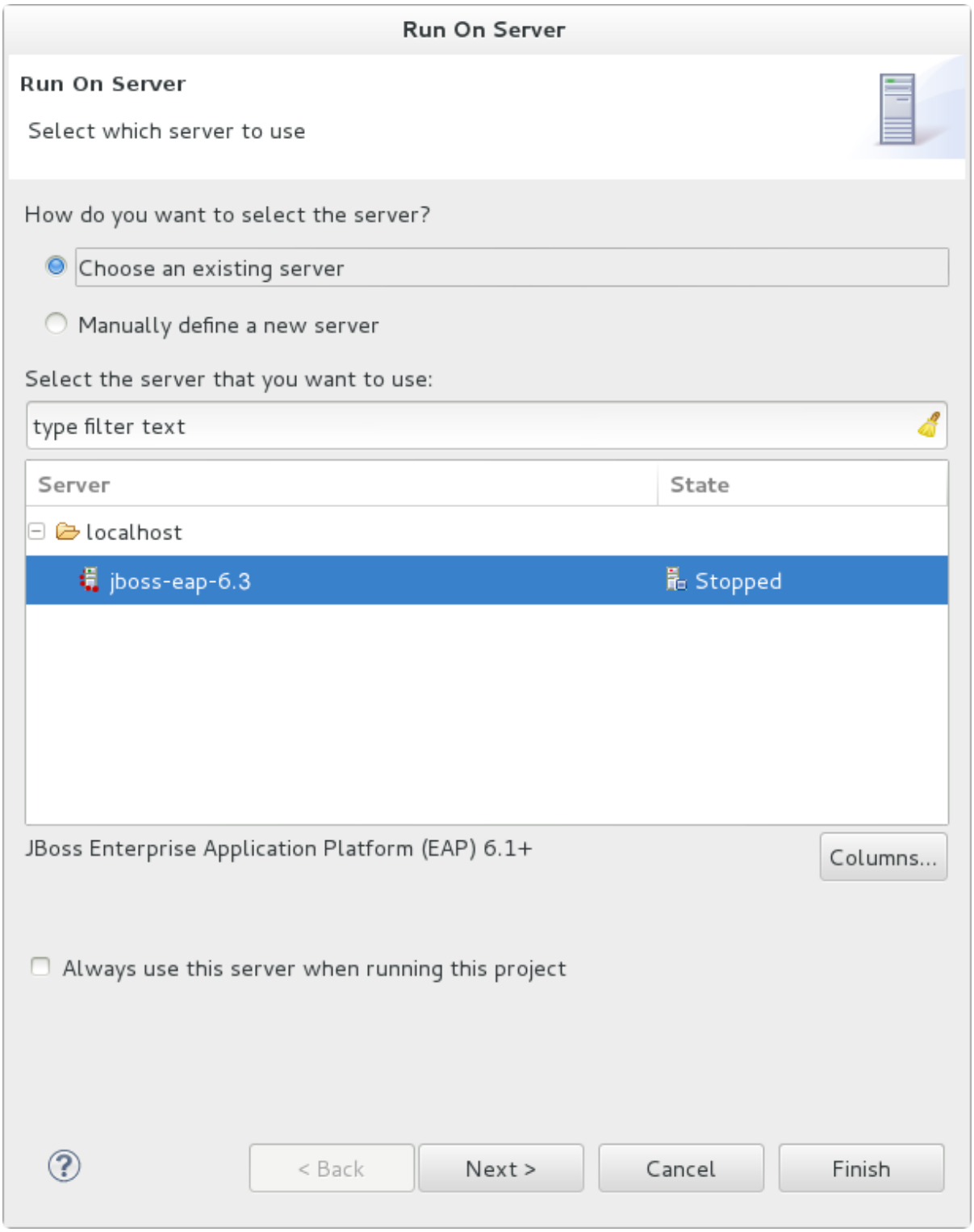


Figure 1.10. Exécuter sur le serveur

4. L'écran suivant affiche les ressources qui sont configurées sur le serveur. Le quickstart de **`jboss-helloworld`** est configuré pour vous. Cliquer sur **`Finish`** pour déployer le quickstart.

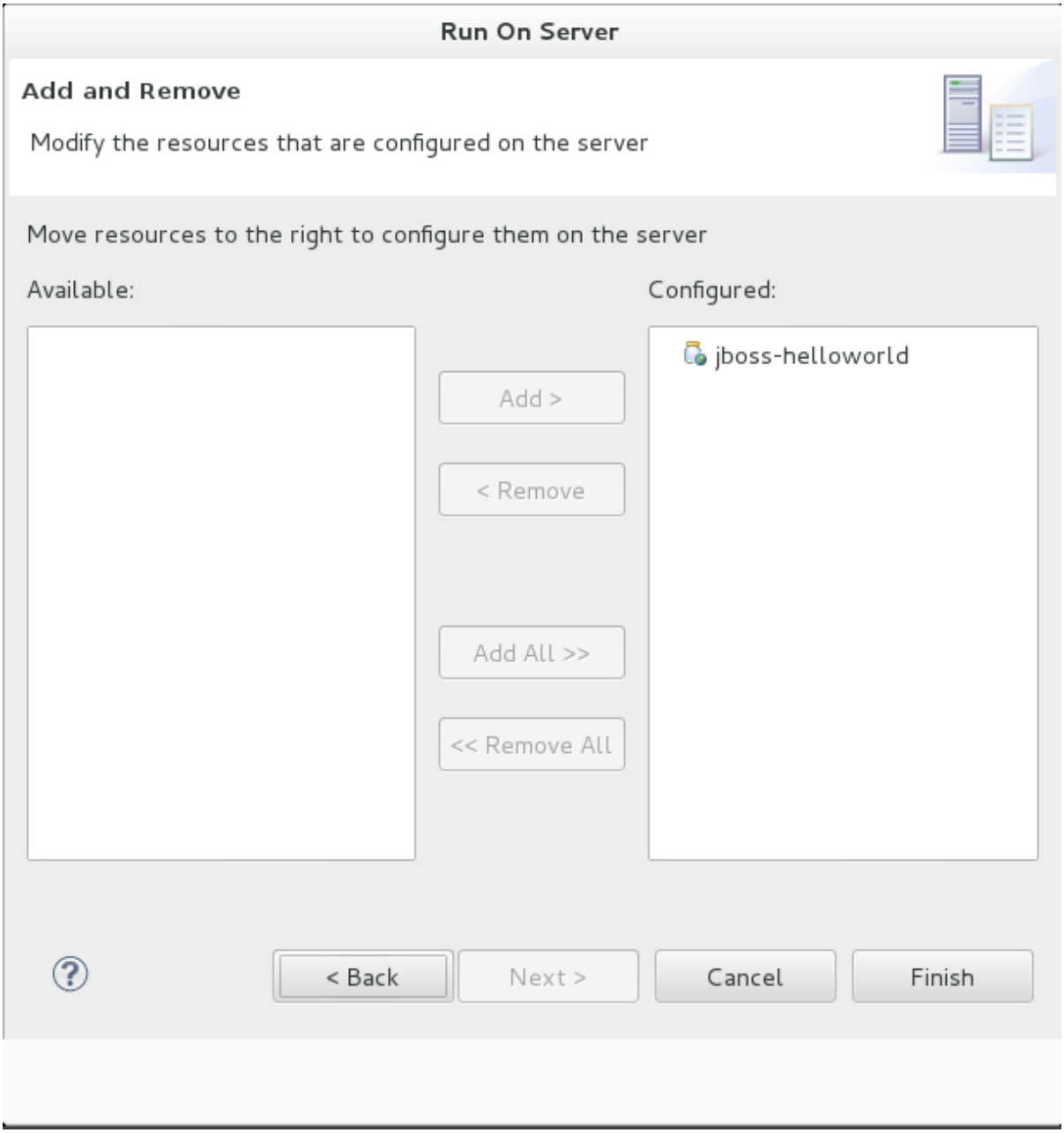


Figure 1.11. Modifier les ressources configurées sur le serveur

5. Vérifier les résultats.
   - Dans l'onglet **`Server`**, le statut de JBoss EAP 6.3 Runtime Server passe à **`[Started, Republish]`** .
   - L'onglet **`Console`** du serveur affiche des messages détaillant le démarrage du serveur JBoss EAP 6.3 et le déploiement du quickstart helloworld.
   - Un onglet **`helloworld`** apparaîtra avec l'URL http://localhost:8080/jboss-helloworld/HelloWorld et le texte "Hello World!".
   - Les messages suivants de la **`Console`** confirment le déploiement du fichier **`jboss-helloworld.war`** :

     ```
     JBAS018210: Register web context: /jboss-helloworld
     JBAS018559: Deployed "jboss-helloworld.war" (runtime-name :
     "jboss-helloworld.war")
     ```

Le contexte web enregistré se rajoute à **`http://localhost:8080`** pour fournir l'URL utilisé pour accéder à l'application qui est déployée.

6. Pour vérifier que le Quickstart **`helloworld`** a été déployé correctement dans le serveur JBoss, ouvrir le navigateur web, et accéder à l'application dans l'URL : http://localhost:8080/jboss-helloworld

Rapporter un bogue

1.4.2.2. Exécuter les quickstarts par la Ligne de commande

Procédure 1.7. Générer et déployer les quickstarts par la Ligne de commande

Vous pouvez facilement générer ou déployer les démarrages rapides (quickstart) par la ligne de commande. Sachez que, lorsque vous utilisez une ligne de commande, vous devrez démarrer le serveur JBoss si nécessaire.

1. Si vous ne l'avez pas encore fait, Section 2.3.2, « Configurer le référentiel JBoss EAP 6 Platform Maven Repository par les paramètres de configuration de Maven ».

2. Vérifier le fichier **`README.html`** qui se trouve dans le répertoire racine des quickstarts.

   Ce fichier contient des informations générales sur les conditions préalables de systèmes, sur la façon de configurer Maven, comment ajouter des utilisateurs, et comment exécuter les quickstarts. Lisez attentivement avant de commencer.

   Il contient également un tableau qui répertorie les quickstarts disponibles. Le tableau répertorie chaque nom de quickstart et chaque technologie dont il s'agit. Il y a une brève description pour chaque quickstart et une indication du niveau d'expérience requis pour l'installer. Pour des informations plus détaillées, cliquez sur le nom du quickstart.

   Certains quickstarts sont conçus pour améliorer ou étendre d'autres quickstarts. Ils sont indiqués dans la colonne **`Conditions préalables`**. Si un quickstart est associé à une liste de conditions préalables, vous devrez les installer avant d'utiliser le quickstart.

   Certains quickstart ont besoin de l'installation et de la configuraiton des composants optionnels. Ne pas installer ces composants à moins que le quickstart ne l'exige.

3. Exécuter le quickstart **`helloworld`**

   Le quickstart **`helloworld`** est l'un des quickstarts les plus simples, et représente un bon moyen de vérifier si le serveur JBoss est configuré et exécute correctement. Ouvrir le fichier **`README.html`** dans la racine du quickstart **`helloworld`**. Il contient des instructions détaillés sur la façon de construire et de déployer le quickstart et accéder à l'application en cours.

4. Exécuter d'autres quickstarts.

   Suivre les instructions dans le fichier **`README.html`** que se trouve dans le dossier racine de chaque quickstart pour exécuter l'exemple.

Rapporter un bogue

1.4.3. Revoir les tutoriels Quickstart

1.4.3.1. Découvrir le Quickstart HelloWorld

Résumé

Le Quickstart **helloworld** vous montre comment déployer un simple Servlet dans la plateforme JBoss EAP 6. La logique commerciale comprend un service fourni sous forme de bean CDI (Contexts and Dependency Injection) qui est injecté dans le Servlet. Ce Quickstart est très simple. Tout ce qu'il se contente de faire est d'imprimer "Hello World" dans une page web. C'est une bonne façon de savoir si vous avez bien configuré et démarré votre serveur.

Des instructions détaillées sur la façon de construire et de déployer ce Quickstart par la ligne de commande se trouvent dans le fichier README.html qui se trouve dans le répertoire de Quickstart **helloworld**. Nous vous montrons ici comment utiliser Red Hat JBoss Developer Studio pour exécuter le Quickstart. Dans cette section, nous assumons que vous avez installé le Red Hat JBoss Developer Studio, configuré Maven, importé et exécuté le Quickstart **helloworld**.

Conditions préalables

- Installer Red Hat JBoss Developer Studio en suivant la procédure qui suit : Section 1.3.1.3, « Installer Red Hat JBoss Developer Studio 7.1 ».
- Configurer Maven pour qu'il puisse être utilisé avec Red Hat JBoss Developer Studio en suivant la procédure qui suit : Section 2.3.3, « Configurer Maven pour utilisation dans Red Hat JBoss Developer Studio ».
- Suivre les procédures suivantes pour importer, créer et déployer le Quickstart **helloworld** dans Red Hat JBoss Developer Studio : Section 1.4.2.1, « Exécuter les quickstarts (démarrages rapides) dans Red Hat JBoss Developer Studio »
- Vérifier que le Quickstart **helloworld** est déployé correctement dans JBoss EAP en ouvrant un navigateur web, et en accédant à l'application dans cet URL : http://localhost:8080/jboss-helloworld

Procédure 1.8. Observer la structure du répertoire

Le code du Quickstart **helloworld** se trouve dans le répertoire ***QUICKSTART_HOME*/helloworld**. Le Quickstart **helloworld** comprend un Servlet et un bean CDI. Il comprend également un fichier beans.xml qui indique à JBoss EAP 6 comment trouver des beans pour cette application et comment activer le CDI.

1. Le fichier **beans.xml** se trouve dans le dossier **WEB-INF/** qui se trouve dans le répertoire **src/main/webapp/** du Quickstart.
2. Le répertoire **src/main/webapp/** inclut également un fichier **index.html** qui utilise une simple réactualisation meta pour rediriger le navigateur de l'utilisateur vers le Servlet, qui se trouve à http://localhost:8080/jboss-helloworld/HelloWorld.
3. Tous les fichiers de configuration de cet exemple se trouvent dans **WEB-INF/**, qui se trouve dans le répertoire **src/main/webapp/** de l'exemple.
4. Notez que le Quickstart n'a pas même besoin d'un fichier **web.xml**!

Procédure 1.9. Examiner le code

La déclaration de paquet et les importations ont été exclues de ces listes. La liste complète est disponible dans le code source du Quickstart.

1. **Vérifier le code HelloWorldServlet**

Le fichier **HelloWorldServlet.java** se trouve dans le répertoire **src/main/java/org/jboss/as/quickstarts/helloworld/**. Le Servlet envoie les informations dans le navigateur.

```
42.  @SuppressWarnings("serial")
43.  @WebServlet("/HelloWorld")
44.  public class HelloWorldServlet extends HttpServlet {
45.
46.      static String PAGE_HEADER = "<html><head>
<title>helloworld</title></head><body>";
47.
48.      static String PAGE_FOOTER = "</body></html>";
49.
50.      @Inject
51.      HelloService helloService;
52.
53.      @Override
54.      protected void doGet(HttpServletRequest req,
HttpServletResponse resp) throws ServletException, IOException {
55.          resp.setContentType("text/html");
56.          PrintWriter writer = resp.getWriter();
57.          writer.println(PAGE_HEADER);
58.          writer.println("<h1>" +
helloService.createHelloMessage("World") + "</h1>");
59.          writer.println(PAGE_FOOTER);
60.          writer.close();
61.      }
62.
63.  }
```

Tableau 1.1. Infos HelloWorldServlet

Ligne	Note
43	Avant Java EE 6, on utilisait un fichier XML pour enregistrer les Servlets. C'est bien plus clean. Tout ce qu'il vous reste à faire est d'ajouter l'annotation **@WebServlet** et de fournir un mappage vers un URL qui est utilisé pour accéder au serveur.
46-48	Chaque page web a besoin d'HTML formé correctement. Ce Quickstart utilise les Strings statiques pour écrire les sorties minimum de l'en-tête et du pied de page.
50-51	Ces lignes injectent le bean CDI HelloService, qui génère le message réel. Tant que nous ne changeons les API de HelloService, cette approche nous permet de modifier l'implémentation de HelloService à une date ultérieure sans changer l'affichage.
58	Cette ligne appelle le service pour générer le message "Hello World", et l'écrire dans la requête HTTP.

2. **Vérifier le code HelloService**
   Le fichier **HelloService.java** se trouve dans le répertoire **src/main/java/org/jboss/as/quickstarts/helloworld/**. Ce service est très simple. Il renvoie un message. Nul besoin d'enregistrement d'annotation ou d'XML.

```
public class HelloService {

    String createHelloMessage(String name) {
        return "Hello " + name + "!";
    }
}
```

Rapporter un bogue

1.4.3.2. Découvrir le Quickstart numberguess

Résumé

Ce Quickstart vous montre comment créer et déployer une simple application de JBoss EAP 6. Cette application ne persiste pas n'importe quelle information. Les informations sont affichées par JSF, et la logique métier est encapsulée dans deux beans CDI. Dans le Quickstart **numberguess**, vous obtenez 10 tentatives pour deviner un nombre entre 1 et 100. Après chaque tentative, on vous indique si votre estimation est trop élevée ou trop faible.

Le code du Quickstart **numberguess** se trouve dans le répertoire ***QUICKSTART_HOME*/numberguess**. Le Quickstart **hnumberguess** comprend un certain nombre de beans, des fichiers de configuration et des affichages Facelets (JSF), tout cela dans un package de module WAR.

Des instructions détaillées sur la façon de construire et de déployer ce Quickstart par la ligne de commande se trouvent dans le fichier README.html du répertoire de Quickstart **numberguess**. Nous vous montrons ici comment utiliser Red Hat JBoss Developer Studio pour exécuter le Quickstart. Dans cette section, nous assumons que vous avez installé le Red Hat JBoss Developer Studio, configuré Maven, importé et exécuté le Quickstart **numberguess**.

Conditions préalables

- Installer Red Hat JBoss Developer Studio en suivant la procédure qui suit : Section 1.3.1.3, « Installer Red Hat JBoss Developer Studio 7.1 ».
- Configurer Maven pour qu'il puisse être utilisé avec Red Hat JBoss Developer Studio en suivant la procédure qui suit : Section 2.3.3, « Configurer Maven pour utilisation dans Red Hat JBoss Developer Studio ».
- Suivre les procédures suivantes pour importer, créer et déployer le Quickstart **numberguess** dans Red Hat JBoss Developer Studio : Section 1.4.2.1, « Exécuter les quickstarts (démarrages rapides) dans Red Hat JBoss Developer Studio »
- Vérifier que le Quickstart **numberguess** est déployé correctement dans JBoss EAP en ouvrant un navigateur web, et en accédant à l'application dans cet URL : http://localhost:8080/jboss-numberguess

Procédure 1.10. Examiner les fichiers de configuration

Tous les fichiers de configuration de cet exemple se trouvent dans le répertoire **WEB-INF/**, qui se trouve dans le répertoire **src/main/webapp/** du Quickstart.

1. Examiner le fichier **faces-config.xml**.

   Ce Quickstart utilise la version de JSF 2.0 de **faces-config.xml**. Une version normalisée de Facelets correspond au gestionnaire d'affichage par défaut de JSF 2.0, donc il n'y a pas vraiment grand chose à configurer. JBoss EAP 6 va au-delà de Java EE ici. Il va

automatiquement configurer JSF pour vous, si vous incluez ce fichier de configuration. En conséquence, la configuration ne consistera uniquement en l'élément racine :

```
19. <faces-config version="2.0"
20.    xmlns="http://java.sun.com/xml/ns/javaee"
21.    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
22.    xsi:schemaLocation="
23.       http://java.sun.com/xml/ns/javaee>
24.       http://java.sun.com/xml/ns/javaee/web-
facesconfig_2_0.xsd">
25.
26. </faces-config>
```

2. Examiner le fichier **`beans.xml`**.

   Il y a également un fichier **`beans.xml`** qui indique à JBoss EAP 6 de chercher des beans dans cette application et d'activer le CDI.

3. Il n'y a pas de fichier **`web.xml`**

   Notez que le Quickstart n'a pas même besoin d'un fichier **`web.xml`**!

Procédure 1.11. Examiner le code JSF

JSF utilise l'extension de fichier **`.xhtml`** pour les fichiers source, mais s'occupe des vues rendues par l'extension **`.jsf`**.

- Examiner le code **`home.xhtml`**.

  Le fichier **`home.xhtml`** se trouve dans le répertoire **`src/main/webapp/`**.

```
19. <html xmlns="http://www.w3.org/1999/xhtml"
20.    xmlns:ui="http://java.sun.com/jsf/facelets"
21.    xmlns:h="http://java.sun.com/jsf/html"
22.    xmlns:f="http://java.sun.com/jsf/core">
23.
24. <head>
25. <meta http-equiv="Content-Type" content="text/html; charset=iso-
8859-1" />
26. <title>Numberguess</title>
27. </head>
28.
29. <body>
30.    <div id="content">
31.       <h1>Guess a number...</h1>
32.       <h:form id="numberGuess">
33.
34.          <!-- Feedback for the user on their guess -->
35.          <div style="color: red">
36.             <h:messages id="messages" globalOnly="false" />
37.             <h:outputText id="Higher" value="Higher!"
38.                rendered="#{game.number gt game.guess and
game.guess ne 0}" />
39.             <h:outputText id="Lower" value="Lower!"
40.                rendered="#{game.number lt game.guess and
```

```
game.guess ne 0}" />
41.          </div>
42.
43.          <!-- Instructions for the user -->
44.          <div>
45.             I'm thinking of a number between <span
46.                id="numberGuess:smallest">#
{game.smallest}</span> and <span
47.                id="numberGuess:biggest">#{game.biggest}</span>.
You have
48.             #{game.remainingGuesses} guesses remaining.
49.          </div>
50.
51.          <!-- Input box for the users guess, plus a button to
submit, and reset -->
52.          <!-- These are bound using EL to our CDI beans -->
53.          <div>
54.             Your guess:
55.             <h:inputText id="inputGuess" value="#{game.guess}"
56.                required="true" size="3"
57.                disabled="#{game.number eq game.guess}"
58.                validator="#{game.validateNumberRange}" />
59.             <h:commandButton id="guessButton" value="Guess"
60.                action="#{game.check}"
61.                disabled="#{game.number eq game.guess}" />
62.          </div>
63.          <div>
64.             <h:commandButton id="restartButton" value="Reset"
65.                action="#{game.reset}" immediate="true" />
66.          </div>
67.       </h:form>
68.
69.    </div>
70.
71.    <br style="clear: both" />
72.
73. </body>
74. </html>
```

Tableau 1.2. Infos JSF

Ligne	Note
36-40	Voici les messages qui peuvent être envoyés par l'utilisateur : "Higher!" et "Lower!"
45-48	Au fur et à mesure que l'utilisateur devine, l'étendue des nombres qu'ils devinent se rétrécit. Cette phrase change pour s'assurer qu'ils connaissent la portée d'une tentative valide.
55-58	Ce champ d'entrée est lié à une propriété de bean qui utilise une expression de valeur.
58	On utilise une liaison de validateur pour s'assurer que l'utilisateur ne mette pas le nombre qu'il devine en dehors de la limite. Si le validateur n'était pas présent, l'utilisateur peut deviner un nombre en dehors de la limite.

Ligne	Note
59-61	Il doit y avoir un moyen pour que l'utilisateur envoie le nombre qu'il devine au serveur. Ici, on associe une méthode d'action sur le bean.

Procédure 1.12. Examiner les fichiers de classe

Tous les fichiers source du Quickstart **numberguess** se trouvent dans le répertoire **src/main/java/org/jboss/as/quickstarts/numberguess/**. La déclaration de paquet et les importations ont été exclues de ces listings. La liste complète se trouve dans le code source du Quickstart.

1. Vérifier le code du qualificateur **Random.java**.

   Un qualificateur est utilisé pour supprimer l'ambiguïté entre deux beans, qui sont tous deux admissibles pour l'injection selon leur type. Pour plus d'informations sur les qualificateurs, se référer à Section 10.2.3.3, « Utiliser un qualificateur pour résoudre une injection ambiguë »

   Le qualificateur **@Random** est utilisé pour injecter un nombre au hasard.

   ```
   @Target({ TYPE, METHOD, PARAMETER, FIELD })
   @Retention(RUNTIME)
   @Documented
   @Qualifier
   public @interface Random {

   }
   ```

2. Vérifier le code du qualificateur **MaxNumber.java**.

   Le **qualificateur@MaxNumber** est utilisé pour injecter le nombre maximum autorisé.

   ```
   @Target({ TYPE, METHOD, PARAMETER, FIELD })
   @Retention(RUNTIME)
   @Documented
   @Qualifier
   public @interface MaxNumber {

   }
   ```

3. Vérifier le code **Generator.java**.

   La classe **Generator** est chargé de créer le nombre aléatoire par une méthode de producteur. Elle indique également le nombre maximum possible par une méthode de producteur. Cette classe reste dans la portée de l'application (scoped), donc, vous n'obtenez pas un nombre différent au hasard à chaque fois.

   ```
   @SuppressWarnings("serial")
   @ApplicationScoped
   public class Generator implements Serializable {

       private java.util.Random random = new
   java.util.Random(System.currentTimeMillis());
   ```

```
    private int maxNumber = 100;

    java.util.Random getRandom() {
        return random;
    }

    @Produces
    @Random
    int next() {
        // a number between 1 and 100
        return getRandom().nextInt(maxNumber - 1) + 1;
    }

    @Produces
    @MaxNumber
    int getMaxNumber() {
        return maxNumber;
    }
}
```

4. Vérifier le code **`Game.java`**.

   La classe session scoped **`Game`** est le point d'entrée principal de l'application. Elle est là pour initialiser et réinitialiser le jeu, pour capturer et valider la tentative de numéro de l'utilisateur, et afin de fournir un commentaire de retour à l'utilisateur avec un **`FacesMessage`**. Elle utilise une méthode de cycle de vie de post-construction pour initialiser le jeu, en tirant un nombre au hasard du bean **`@Random Instance<Integer>`**.

   Notez l'annotation @Named dans la classe. Cette annotation est uniquement requise quand on veut rendre le bean accessible à une vue JSF via Expression Language (EL), et dans ce cas **`#{game}`**.

```
@SuppressWarnings("serial")
@Named
@SessionScoped
public class Game implements Serializable {

    /**
     * The number that the user needs to guess
     */
    private int number;

    /**
     * The users latest guess
     */
    private int guess;

    /**
     * The smallest number guessed so far (so we can track the valid
guess range).
     */
    private int smallest;

    /**
     * The largest number guessed so far
```

```
     */
    private int biggest;

    /**
     * The number of guesses remaining
     */
    private int remainingGuesses;

    /**
     * The maximum number we should ask them to guess
     */
    @Inject
    @MaxNumber
    private int maxNumber;

    /**
     * The random number to guess
     */
    @Inject
    @Random
    Instance<Integer> randomNumber;

    public Game() {
    }

    public int getNumber() {
        return number;
    }

    public int getGuess() {
        return guess;
    }

    public void setGuess(int guess) {
        this.guess = guess;
    }

    public int getSmallest() {
        return smallest;
    }

    public int getBiggest() {
        return biggest;
    }

    public int getRemainingGuesses() {
        return remainingGuesses;
    }

    /**
     * Check whether the current guess is correct, and update the
biggest/smallest guesses as needed. Give feedback to the user
     * if they are correct.
     */
    public void check() {
        if (guess > number) {
```

```
            biggest = guess - 1;
        } else if (guess < number) {
            smallest = guess + 1;
        } else if (guess == number) {
            FacesContext.getCurrentInstance().addMessage(null, new
FacesMessage("Correct!"));
        }
        remainingGuesses--;
    }

    /**
     * Reset the game, by putting all values back to their defaults,
and getting a new random number. We also call this method
     * when the user starts playing for the first time using
{@linkplain PostConstruct @PostConstruct} to set the initial
     * values.
     */
    @PostConstruct
    public void reset() {
        this.smallest = 0;
        this.guess = 0;
        this.remainingGuesses = 10;
        this.biggest = maxNumber;
        this.number = randomNumber.get();
    }

    /**
     * A JSF validation method which checks whether the guess is
valid. It might not be valid because there are no guesses left,
     * or because the guess is not in range.
     *
     */
    public void validateNumberRange(FacesContext context,
UIComponent toValidate, Object value) {
        if (remainingGuesses <= 0) {
            FacesMessage message = new FacesMessage("No guesses
left!");
            context.addMessage(toValidate.getClientId(context),
message);
            ((UIInput) toValidate).setValid(false);
            return;
        }
        int input = (Integer) value;

        if (input < smallest || input > biggest) {
            ((UIInput) toValidate).setValid(false);

            FacesMessage message = new FacesMessage("Invalid
guess");
            context.addMessage(toValidate.getClientId(context),
message);
        }
    }
}
```

Rapporter un bogue

1.4.4. Remplacer l'application web Welcome par défaut

JBoss EAP 6 inclut l'application Welcome, qui s'affiche quand on ouvre l'URL du serveur sur le port 8080. Vous pouvez remplacer cette application par votre propre application web, en suivant la procédure suivante.

Procédure 1.13. Remplacer l'application web Welcome par défaut par votre propre application web

1. **Désactiver l'application Welcome**
   Utiliser le script d'interface CLI ***EAP_HOME*/bin/jboss-cli.sh** pour exécuter la commande suivante. Vous aurez sans doute besoin de modifier un profil de domaine géré, ou retirer une portion de la commande **/profile=default** du serveur autonome.

   ```
   /profile=default/subsystem=web/virtual-server=default-host:write-
   attribute(name=enable-welcome-root,value=false)
   ```

2. **Configurer votre application web par le contexte root.**
   Afin de configurer votre application web, pour qu'elle utilise (/) comme adresse URL, modifier son fichier **jboss-web.xml**, qui se trouve dans le répertoire **META-INF/** ou **WEB-INF/**. Remplacer sa directive **<context-root>** par une autre directive qui ressemble à ce qui suit.

   ```
   <jboss-web>
       <context-root>/</context-root>
   </jboss-web>
   ```

3. **Déployer votre application.**
   Déployer votre application pour le groupe de serveurs ou le serveur que vous avez modifié lors de la première étape. L'application est maintenant disponible sur **http://*SERVER_URL*:*PORT*/**.

Rapporter un bogue

CHAPITRE 2. GUIDE MAVEN

2.1. POUR EN SAVOIR PLUS SUR MAVEN

2.1.1. Le référentiel Maven

Apache Maven est un outil d'automation de builds distribué, utilisé en développement Java pour créer, gérer et générer des projets informatiques. Maven utilise des fichiers de configuration nommés Project Object Model, ou POM, pour définir des projets et gérer le processus de construction. Les POM décrivent le module et les dépendance du composant, et les sorties sont sous forme de fichier XML. Cela assure que le projet soit construit de façon correcte et uniforme.

Maven y parvient en utilisant un référentiel. Un référentiel Maven stocke les bibliothèques, plug-ins et autres artefacts de la build. Le référentiel public par défaut est le Maven 2 Central Repository, mais les référentiels peuvent être privés et internes à une entreprise dans le but de partager des artefacts communs au sein d'équipes de développeurs. Les référentiels sont également disponibles auprès de tierces parties. JBoss EAP 6 comprend un référentiel Maven contenant un grand nombre de prérequis que les développeurs de Java EE utilisent habituellement pour développer des applications sous JBoss EAP 6. Pour configurer votre projet pour utiliser ce référentiel, veuillez consulter Section 2.3.1, « Configurer le référentiel Maven 6 de JBoss EAP ».

On ne peut accéder aux référentiels qu'à l'aide de protocoles tels qu'**`http://`** pour les référentiels qui se trouvent sur un serveur HHTP ou **`file://`** pour les référentiels qui se trouvent dans un serveur de fichiers.

Pour davantage d'informations sur Maven, voir Welcome to Apache Maven.

Pour davantage d'informations sur les référentiels Maven, voir Apache Maven Project - Introduction to Repositories.

Pour davantage d'informations sur les fichiers POM de Maven, voir Apache Maven Project POM Reference et Section 2.1.2, « Le fichier POM Maven ».

Rapporter un bogue

2.1.2. Le fichier POM Maven

Un fichier POM (Project Object Model) est un fichier de configuration utilisé par Maven pour compiler des projets. Il s'agit d'un fichier XML contenant des informations sur un projet et sur comment le compiler, y compris l'emplacement des répertoires source, test et cible, les dépendances des projets, les réferentiels de plug-ins et les objectifs qu'il peut exécuter. Il peut également comprendre des détails supplémentaires sur le projet, comme, en autre, sa version, sa description, ses développeurs, sa liste de diffusion et sa licence. Un fichier **`pom.xml`** requiert certaines options de configuration définissant la valeur par défaut. Voir Section 2.1.3, « Conditions minimum pour un fichier POM Maven » pour plus de détails.

Le schéma pour le fichier **`pom.xml`** est disponible sur le lien http://maven.apache.org/maven-v4_0_0.xsd.

Pour plus d'informations concernant les fichiers POM, veuillez consulter Apache Maven Project POM Reference.

Rapporter un bogue

2.1.3. Conditions minimum pour un fichier POM Maven

Conditions minimum

Les conditions minimum pour un fichier **pom.xml** sont les suivantes :

- project root
- modelVersion
- groupId - l'id du groupe du projet
- artifactId - l'id de l'artifact (projet)
- version - la version de l'artifact dans le groupe spécifié

Échantillon de fichier pom.xml

Un fichier **pom.xml** de base devrait resssembler à ceci :

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.jboss.app</groupId>
  <artifactId>my-app</artifactId>
  <version>1</version>
</project>
```

Rapporter un bogue

2.1.4. Le fichier des configurations de Maven

Le fichier Maven **settings.xml** contient des informations de configuration spécifiques à l'utilisateur pour Maven. Il contient des informations qui ne devraient pas être distribuées dans le fichier **pom.xml**, comme l'identité du développeur, l'information sur le proxy, l'emplacement du référentiel local, et autres paramètres propres à un utilisateur.

Il y a deux endroits où on peut trouver le fichier **settings.xml**.

Dans l'installation Maven

Le fichier de configuration se trouve dans le répertoire ***M2_HOME*/conf/**. On les appelle les paramètres **globaux**. Le fichier de configuration Maven par défaut est un modèle qui peut être copié et utilisé comme point de départ pour le fichier des paramètres utilisateur.

Dans l'installation de l'utilisateur

Le fichier de configuration se trouve dans le répertoire ***USER_HOME*/.m2/**. Si les fichiers **settings.xml** et Maven existent tous les deux, leurs contenus seront fusionnés. S'il y a une intersection, le fichier utilisateur **settings.xml** aura priorité.

Voici un exemple d'un fichier **settings.xml** Maven :

```
<?xml version="1.0" encoding="UTF-8"?>
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
```

```
http://maven.apache.org/xsd/settings-1.0.0.xsd">
  <profiles>
    <!-- Configure the JBoss EAP Maven repository -->
    <profile>
      <id>jboss-eap-maven-repository</id>
      <repositories>
        <repository>
          <id>jboss-eap</id>
          <url>file:///path/to/repo/jboss-eap-6.3-maven-repository</url>
          <releases>
            <enabled>true</enabled>
          </releases>
          <snapshots>
            <enabled>false</enabled>
          </snapshots>
        </repository>
      </repositories>
      <pluginRepositories>
        <pluginRepository>
          <id>jboss-eap-maven-plugin-repository</id>
          <url>file:///path/to/repo/jboss-eap-6.3-maven-repository</url>
          <releases>
            <enabled>true</enabled>
          </releases>
          <snapshots>
            <enabled>false</enabled>
          </snapshots>
        </pluginRepository>
      </pluginRepositories>
    </profile>
  </profiles>
  <activeProfiles>
    <!-- Optionally, make the repository active by default -->
    <activeProfile>jboss-eap-maven-repository</activeProfile>
  </activeProfiles>
</settings>
```

On peut trouver le schéma du fichier **`settings.xml`** à l'adresse suivante http://maven.apache.org/xsd/settings-1.0.0.xsd.

Rapporter un bogue

2.2. INSTALLER LE RÉFÉRENTIEL JBOSS MAVEN ET MAVEN

2.2.1. Télécharger et installer Maven

Si vous envisagez d'utiliser la ligne de commande Maven pour créer et déployer vos applications JBoss EAP, vous devez télécharger et installer Maven. Si vous envisagez d'utiliser Red Hat JBoss Developer Studio pour créer et déployer vos applications, vous pouvez ignorer cette procédure car Maven est distribué avec Red Hat JBoss Developer Studio.

1. Aller dans Apache Maven Project - Download Maven et télécharger la dernière distribution de votre système d'exploitation.

2. Voir la documentation Maven pour les informations sur le façon de télécharger et d'installer Apache Maven sur votre système d'exploitation.

Rapporter un bogue

2.2.2. Installer le référentiel Maven de JBoss EAP 6

Il y a trois moyens d'installer le référentiel; sur votre système de fichiers local, sur le serveur web Apache, ou par un gestionnaire de référentiel Maven.

- Section 2.2.3, « Installer le référentiel Maven de JBoss EAP 6 localement »
- Section 2.2.4, « Installer le référentiel Maven de JBoss EAP 6 à utiliser avec Appache httpd »
- Section 2.2.5, « Installer le référentiel Maven de JBoss EAP 6 en utilisant le gestionnaire de référentiels Nexus Maven »

Rapporter un bogue

2.2.3. Installer le référentiel Maven de JBoss EAP 6 localement

Résumé

Le référentiel de JBoss EAP 6.3 Maven est disponible en ligne. Il n'est pas donc nécessaire de le télécharger et de l'installer localement. Toutefois, si vous préférez installer le référentiel JBoss EAP Maven localement, il y a trois façons de procéder : sur votre système de fichiers local, sur le serveur Web Apache, ou avec un gestionnaire de référentiel Maven. Cet exemple couvre les étapes pour télécharger le référentiel JBoss EAP 6 Maven sur le système de fichiers local. Cette option est facile à configurer et vous permet d'obtenir d'être opérationnel rapidement sur votre ordinateur local. Il peut vous aider à vous familiariser à l'utilisation de Maven pour le développement, mais n'est pas recommandé pour les environnements de production en équipes.

Procédure 2.1. Télécharger et installer le référentiel Maven de JBoss EAP 6 sur le système de fichier local.

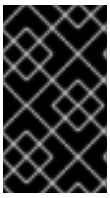
1. Ouvrir un navigateur web, et accéder à cet URL: https://access.redhat.com/jbossnetwork/restricted/listSoftware.html?product=appplatform.
2. Chercher "Red Hat JBoss Enterprise Application Platform 6.3.0 Maven Repository" dans la liste.
3. Cliquer sur le bouton **`Télécharger`** pour télécharger un fichier **`.zip`** contenant le référentiel.
4. Décompresser le fichier dans le même répertoire sur votre système de fichiers local dans un répertoire de votre choix.
5. Section 2.3.2, « Configurer le référentiel JBoss EAP 6 Platform Maven Repository par les paramètres de configuration de Maven ».

Résultat

Ceci crée un répertoire de référentiel Maven appelé **`jboss-eap-6.3.0.GA-maven-repository`**.

IMPORTANT

Si vous souhaitez continuer à utiliser un référentiel local moins récent, vous devez le configurer séparemment dans le fichier de configuration **`settings.xml`** de Maven. Chaque référentiel local doit être configuré dans sa propre balise **`<repository>`**.



IMPORTANT

Quand vous téléchargez un nouveau référentiel Maven, supprimer le sous-répertoire **`repository/`** cache, qui se situe sous le répertoire **`.m2/`** avant de tenter d'utiliser le nouveau répertoire Maven.

Rapporter un bogue

2.2.4. Installer le référentiel Maven de JBoss EAP 6 à utiliser avec Appache httpd

Il y a trois moyens d'installer le référentiel; sur votre système de fichiers local, sur le serveur web Apache, ou par un gestionnaire de référentiel Maven. Cet exemple porte sur les étapes de téléchargement du référentiel Maven de JBoss EAP 6 à utiliser avec Apache httpd. Cette option se prête bien aux environnements de développement inter-équipes ou multi-utilisateurs, car n'importe quel développeur qui puisse accéder à un serveur web, peut également accéder au référentiel Maven.

Conditions préalables

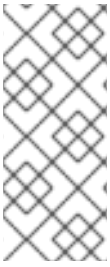
Il vous faut configurer Apache httpd. Voir la documentation Apache HTTP Server Project pour obtenir des instructions.

Procédure 2.2. Télécharger l'archive ZIP du référentiel Maven de JBoss EAP 6

1. Ouvrir un navigateur web, et accéder à cet URL : https://access.redhat.com/jbossnetwork/restricted/listSoftware.html?product=appplatform.
2. Chercher "Red Hat JBoss Enterprise Application Platform 6.3.0 Maven Repository" dans la liste.
3. Cliquer sur le bouton **`Télécharger`** pour télécharger un fichier **`.zip`** contenant le référentiel.
4. Décompresser les fichiers dans un répertoire qui soit accessible au serveur Apache.
5. Configurer Apache pour lui donner la permission écriture et pour permettre la navigation dans le répertoire créé.
6. Section 2.3.2, « Configurer le référentiel JBoss EAP 6 Platform Maven Repository par les paramètres de configuration de Maven ».

Résultat

Cela permet à un environnement multi-utilisateurs d'accéder à un référentiel Maven sur Apache httpd.



NOTE

Si vous passez à une version supérieure du référentiel, veuillez noter que les artéfacts de référentiel Maven de JBoss peuvent simplement être extraits dans un référentiel Maven de produits JBoss (comme JBoss EAP 6.1.0). Une fois que l'archive de référentiel a été extraite, les artefacts peuvent être utilisés avec les paramètres existants de Maven dans ce référentiel.

Rapporter un bogue

2.2.5. Installer le référentiel Maven de JBoss EAP 6 en utilisant le gestionnaire de référentiels Nexus Maven

Il y a trois façons d'installer un référentiel; sur votre système de fichiers local, sur le serveur web Apache, ou à l'aide d'un gestionnaire de référentiels Maven. Cette option sera plus adaptée si vous avez déjà une licence et utilisez un gestionnaire de référentiels car vous pourrez héberger le référentiel de JBoss avec vos référentiels déjà existants. Pour obtenir plus d'informations sur les gestionnaires de référentiels Maven, voir Section 2.2.6, « Gestionnaires de référentiels Maven ».

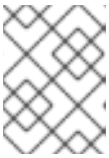
Cet exemple couvrira les étapes d'installation du référentiel Maven de JBoss EAP 6 en utilisant le référentiel Sonatype Nexus Maven. Pour obtenir davantage d'informations, voir Sonatype Nexus: Manage Artifacts.

Procédure 2.3. Télécharger l'archive ZIP du référentiel Maven de JBoss EAP 6

1. Ouvrir un navigateur web, et accéder à cet URL : https://access.redhat.com/jbossnetwork/restricted/listSoftware.html?product=appplatform.
2. Chercher "Red Hat JBoss Enterprise Application Platform 6.3.0 Maven Repository" dans la liste.
3. Cliquer sur le bouton **`Télécharger`** pour télécharger un fichier **`.zip`** contenant le référentiel.
4. Décompresser les fichiers dans un répertoire de votre choix sur le serveur qui héberge Nexus.

Procédure 2.4. Ajouter le référentiel Maven de JBoss EAP 6 en utilisant le gestionnaire de référentiels Nexus Maven

1. Connectez-vous à Nexus en tant qu'administrateur.
2. Sélectionner la section **`Repositories`** à partir du menu **Views** → **Repositories** qui se trouve à gauche du gestionnaire de référentiels.
3. Cliquer sur le menu déroulant **`Add...`**, puis sélectionner **`Hosted Repository`**.
4. Donner un nom et une ID au nouveau référentiel.
5. Saisir le chemin du disque qui mène au référentiel décompressé dans le champ d'emplacement **`Override Local Storage`**.
6. Continuer si vous souhaitez que l'artéfact soit disponible dans un groupe de référentiels. Ne pas continuer avec cette procédure si ce n'est pas ce que vous souhaitez.
7. Sélectionner le groupe de référentiels.
8. Cliquer sur l'onglet **`Configure`**.
9. Déplacez le nouveau référentiel JBoss Maven de la liste **`Available Repositories`** vers la liste **`Ordered Group Repositories`** sur la gauche.



NOTE

Notez que l'ordre de cette liste détermine la priorité de recherche d'artefacts Maven.

10. Section 2.3.2, « Configurer le référentiel JBoss EAP 6 Platform Maven Repository par les paramètres de configuration de Maven ».

Résultat

Le référentiel est configuré par le gestionnaire de référentiels Maven Nexus.

Rapporter un bogue

2.2.6. Gestionnaires de référentiels Maven

Un gestionnaire de référentiel est un outil qui vous permet de gérer facilement les référentiels Maven. Les gestionnaires de référentiels sont utiles pour différentes raisons :

- Ils donnent la possibilité de configurer des proxys entre votre organisation et des référentiels Maven à distance, ce qui offre un grand nombre d'avantages, parmi lesquels des déploiements plus rapides et plus efficaces et un meilleur niveau de contrôle sur ce qui est téléchargé par Maven.
- Ils fournissent des destinations de déploiement pour les artefacts que vous aurez vous-même générés, permettant ainsi une collaboration entre différentes équipes de développement au sein d'une organisation.

Pour de plus amples d'informations concernant les gestionnaires de référentiels Maven, veuillez consulter la liste des gestionnaires de référentiels du projet Apache Maven Apache Maven Project - The List of Repository Managers.

Gestionnaires de référentiels Maven les plus communément utilisés

Sonatype Nexus

Voir Sonatype Nexus: Manage Artifacts pour obtenir plus d'informations sur Nexus.

Artifactory

Voir Artifactory Open Source pour obtenir plus d'informations sur Artifactory.

Apache Archiva

Voir Apache Archiva: The Build Artifact Repository Manager pour obtenir plus d'informations sur Apache Archiva.

Rapporter un bogue

2.3. UTILISER LE RÉFÉRENTIEL MAVEN

2.3.1. Configurer le référentiel Maven 6 de JBoss EAP

Aperçu

Il existe deux approches pour amener Maven à utiliser le référentiel Maven de JBoss EAP 6 dans votre projet :

- Vous pouvez configurer les référentiels dans les paramètres globaux ou d'utilisateur de Maven.
- Vous pouvez configurer les référentiels dans le fichier POM du projet.

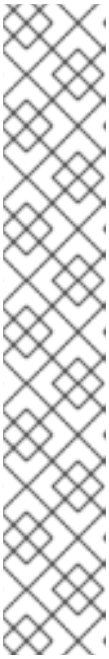
Procédure 2.5. Configurer les paramètres Maven pour qu'ils utilisent le référentiel Maven de JBoss EAP 6

1. **Configurer le référentiel Maven par les paramètres Maven**
   Nous vous conseillons cette dernière approche. Les paramètres Maven utilisés avec un gestionnaire de référentiel ou un référentiel sur un serveur partagé offrent un meilleur contrôle et une meilleure gestion des projets. Les paramètres offrent également la capacité d'utiliser un miroir alternatif pour rediriger toutes les demandes de recherche pour un référentiel spécifique vers un gestionnaire de référentiel sans changer les fichiers du projet. Pour plus d'informations concernant les miroirs, veuillez consulter http://maven.apache.org/guides/mini/guide-mirror-settings.html.

   Cette méthode de configuration s'applique à tous les projets Maven, tant que le fichier POM du projet ne contient pas de configuration de référentiel.

   Section 2.3.2, « Configurer le référentiel JBoss EAP 6 Platform Maven Repository par les paramètres de configuration de Maven ».

2. **Configurer le référentiel Maven pour qu'il utilise le fichier POM du projet.**
   Cette méthode de configuration est généralement déconseillée. Si vous décidez de configurer des référentiels dans le fichier POM de votre projet, planifiez de manière prudente et souvenez-vous que cela pourrait ralentir votre build et que vous pourriez vous retrouver avec des artefacts ne provenant pas du référentiel attendu.

   NOTE

   Dans le cas d'une entreprise, où un gestionnaire de référentiel est généralement utilisé, Maven doit interroger tous les artefacts pour tous les projets en utilisant ce gestionnaire. Maven utilise tous les référentiels déclarés pour trouver les artéfacts manquants. Par conséquent, s'il ne peut trouver ce qu'il cherche, il pourra le chercher dans la centrale de référentiels (définie dans le fichier POM parent intégré). Pour remplacer cet emplacement central, vous pouvez ajouter une définition avec `central` de manière à ce que la centrale de référentiels devienne également votre gestionnaire de référentiel. Cela fonctionne bien avec les projets établis, mais pose un problème pour les projets vierges ou récents puisque cela crée une dépendance cyclique.

   Les fichiers POM inclus de manière transitive représentent également un problème avec ce type de configuration. Maven doit interroger ces référentiels externes pour trouver les artefacts manquants. Non seulement cela ralentira votre build, mais vous fera également perdre le contrôle sur la provenance de vos artefacts et risquera d'endommager votre build.

   Cette méthode de configuration remplace les configuration utilisateur et globales pour le projet configuré.

   Section 2.3.4, « Configurer le référentiel JBoss EAP 6 Platform Maven Repository par le Projet POM ».

Rapporter un bogue

2.3.2. Configurer le référentiel JBoss EAP 6 Platform Maven Repository par les paramètres de configuration de Maven

Il existe deux approches pour amener Maven à utiliser le référentiel Maven de JBoss EAP 6 dans votre projet :

- Vous pouvez modifier les paramètres Maven. Cela amène Maven à utiliser la configuration à travers tous les projets.
- Vous pouvez configurer le fichier POM du projet. Cela limite la configuration au projet particulier.

Cette section vous montre comment amener Maven à utiliser le référentiel JBoss EAP 6 Maven à travers tous les projets grâce aux paramètres de configuration de Maven. C'est l'approche conseillée.

Vous pouvez configurer Maven pour utiliser un référentiel de JBoss EAP 6 en ligne ou installé localement. Si vous choisissez d'utiliser le référentiel en ligne, vous pouvez utiliser un fichier de paramètres préconfigurés ou ajouter les profils de JBoss EAP 6 Maven pour le fichier de paramètres existants. Pour utiliser un référentiel local, vous devez télécharger le référentiel et configurer les paramètres pour pointer vers votre référentiel installé localement. Les procédures suivantes décrivent comment configurer Maven dans JBoss EAP 6.

NOTE

L'URL du référentiel dépend si le référentiel se trouve sur le système de fichiers ou sur le serveur web. Pour plus d'informations sur l'installation du référentiel, veuillez consulter : Section 2.2.2, « Installer le référentiel Maven de JBoss EAP 6 ». Voici quelques exemples pour chacune des options d'installation :

Système de fichiers

file:///path/to/repo/jboss-eap-6.x-maven-repository

Apache Web Server

http://intranet.acme.com/jboss-eap-6.x-maven-repository/

Gestionnaire de référentiel Nexus

https://intranet.acme.com/nexus/content/repositories/jboss-eap-6.x-maven-repository

Vous pouvez configurer Maven soit par les paramètres d'install globaux Maven, soit par l'install utilisateur. Ces instructions configurent les paramètres d'installation utilisateur car il s'agit de l'installation la plus commune.

Procédure 2.6. Configurer Maven par les paramètres fournis dans les exemples Quickstart

Les quickstarts de Red Hat JBoss EAP 6 contiennent un fichier **`settings.xml`** configuré pour utiliser le référentiel JBoss EAP 6 Maven en ligne. Il s'agit de l'approche la plus simple.

1. Cette procédure remplace le fichier de configuration Maven existant, donc vous devrez sauvegarder le fichier **`settings.xml`** Maven existant.

   a. Chercher le répertoire d'installation Maven qui corresponde à votre système d'exploitation. Il se trouve normalement dans le répertoire ***`USER_HOME`*`/.m2/`**.

      - Dans Linux ou Mac, c'est: **`~/.m2/`**
      - Pour Windows, c'est : **`\Documents and Settings\`*`USER_NAME`*`\.m2\`** ou **`\Users\`*`USER_NAME`*`\.m2\`**

   b. Si vous avez un fichier ***`USER_HOME`*`/.m2/settings.xml`** existant, renommer le et faire une copie de sauvegarde afin de pouvoir le restaurer plus tard.

2. Télécharger et décompresser les exemples quickstart fournis dans JBoss EAP 6. Pour plus d'informations, voir Section 1.4.1.1, « Accès aux Quickstarts »

3. Copier le fichier ***QUICKSTART_HOME*/settings.xml** dans le répertoire ***USER_HOME*/.m2/** .

4. Si vous modifiez le fichier **settings.xml** tandis que Red Hat JBoss Developer Studio est en cours d'exécution, suivre la procédure ci-dessous intitulée *Refresh the JBoss Developer Studio User Settings*.

Procédure 2.7. Modifier et configurer les paramètres Maven manuellement pour qu'ils utilisent le référentiel Maven de JBoss EAP 6 en ligne

Vous pouvez ajouter manuellement les profils JBoss EAP 6 à un fichier de paramètres Maven existant.

1. Chercher le répertoire d'installation Maven qui corresponde à votre système d'exploitation. Il se trouve normalement dans le répertoire ***USER_HOME*/.m2/**.
   - Dans Linux ou Mac, c'est **~/.m2/**
   - Pour Windows, c'est **\Documents and Settings*USER_NAME*\.m2** ou **\Users*USER_NAME*\.m2**

2. Si vous ne trouvez pas de fichier **settings.xml**, copier le fichier **settings.xml** du répertoire ***USER_HOME*/.m2/conf/** dans le répertoire ***USER_HOME*/.m2/**.

3. Copier l'XML suivant dans l'élément **<profiles>** du fichier.

```
<!-- Configure the JBoss GA Maven repository -->
<profile>
  <id>jboss-ga-repository</id>
  <repositories>
    <repository>
      <id>jboss-ga-repository</id>
      <url>http://maven.repository.redhat.com/techpreview/all</url>
      <releases>
        <enabled>true</enabled>
      </releases>
      <snapshots>
        <enabled>false</enabled>
      </snapshots>
    </repository>
  </repositories>
  <pluginRepositories>
    <pluginRepository>
      <id>jboss-ga-plugin-repository</id>
      <url>http://maven.repository.redhat.com/techpreview/all</url>
      <releases>
        <enabled>true</enabled>
      </releases>
      <snapshots>
        <enabled>false</enabled>
      </snapshots>
    </pluginRepository>
  </pluginRepositories>
</profile>
<!-- Configure the JBoss Early Access Maven repository -->
```

```
<profile>
  <id>jboss-earlyaccess-repository</id>
  <repositories>
    <repository>
      <id>jboss-earlyaccess-repository</id>
      <url>http://maven.repository.redhat.com/earlyaccess/all/</url>
      <releases>
        <enabled>true</enabled>
      </releases>
      <snapshots>
        <enabled>false</enabled>
      </snapshots>
    </repository>
  </repositories>
  <pluginRepositories>
    <pluginRepository>
      <id>jboss-earlyaccess-plugin-repository</id>
      <url>http://maven.repository.redhat.com/earlyaccess/all/</url>
      <releases>
        <enabled>true</enabled>
      </releases>
      <snapshots>
        <enabled>false</enabled>
      </snapshots>
    </pluginRepository>
  </pluginRepositories>
</profile>
```

Copier l'XML suivant dans l'élément **<activeProfiles>** de **settings.xml**.

```
<activeProfile>jboss-ga-repository</activeProfile>
<activeProfile>jboss-earlyaccess-repository</activeProfile>
```

4. Si vous modifiez le fichier **settings.xml** tandis que Red Hat JBoss Developer Studio est en cours d'exécution, suivre la procédure ci-dessous intitulée *Refresh the JBoss Developer Studio User Settings*.

Procédure 2.8. Configurer les paramètres pour pouvoir utiliser un référentiel JBoss EAP installé localement

Vous pouvez modifier les paramètres pour utiliser le référentiel JBoss EAP 6 qui est installé dans le système de fichiers local.

1. Chercher le répertoire d'installation Maven qui corresponde à votre système d'exploitation. Il se trouve normalement dans le répertoire ***USER_HOME*/.m2/**.
   - Dans Linux ou Mac, c'est **~/.m2/**
   - Pour Windows, c'est **\Documents and Settings*USER_NAME*\.m2** ou **\Users*USER_NAME*\.m2**
2. Si vous ne trouvez pas de fichier **settings.xml**, copier le fichier **settings.xml** du répertoire ***USER_HOME*/.m2/conf/** dans le répertoire ***USER_HOME*/.m2/**.

3. Copier l'XML suivant dans l'élément **<profiles>** du fichier **settings.xml**. Veillez bien à changer l'**<url>** à l'emplacement de référentiel lui-même.

```
<profile>
  <id>jboss-eap-repository</id>
  <repositories>
    <repository>
      <id>jboss-eap-repository</id>
      <name>JBoss EAP Maven Repository</name>
      <url>file:///path/to/repo/jboss-eap-6.x-maven-repository</url>
      <layout>default</layout>
      <releases>
        <enabled>true</enabled>
        <updatePolicy>never</updatePolicy>
      </releases>
      <snapshots>
        <enabled>false</enabled>
        <updatePolicy>never</updatePolicy>
      </snapshots>
    </repository>
  </repositories>
  <pluginRepositories>
    <pluginRepository>
      <id>jboss-eap-repository-group</id>
      <name>JBoss EAP Maven Repository</name>
      <url>
      file:///path/to/repo/jboss-eap-6.x-maven-repository
      </url>
      <layout>default</layout>
      <releases>
        <enabled>true</enabled>
        <updatePolicy>never</updatePolicy>
      </releases>
      <snapshots>
        <enabled>false</enabled>
        <updatePolicy>never</updatePolicy>
      </snapshots>
    </pluginRepository>
  </pluginRepositories>
</profile>
```

Copier l'XML suivant dans l'élément **<activeProfiles>** de **settings.xml**.

```
<activeProfile>jboss-eap-repository</activeProfile>
```

4. Si vous modifiez le fichier **settings.xml** tandis que Red Hat JBoss Developer Studio est en cours d'exécution, suivre la procédure ci-dessous intitulée *Refresh the JBoss Developer Studio User Settings*.

Procédure 2.9. Réactualiser les paramètres de configuration d'utilisateur de Red Hat JBoss Developer Studio

Si vous modifiez le fichier **settings.xml** tandis que Red Hat JBoss Developer Studio est en cours d'exécution, vous devrez réactualiser les paramètres de configuration d'utilisateur.

1. À partir du menu, sélectionner **Window** → **Preferences**.
2. Dans la fenêtre Window **`Preferences`**, étendre **`Maven`** et sélectionner **`User Settings`**.
3. Cliquer sur le bouton **`Update Settings`** (Mise à jour Configuration) pour réactualiser les configurations utilisateur de Maven dans Red Hat JBoss Developer Studio.

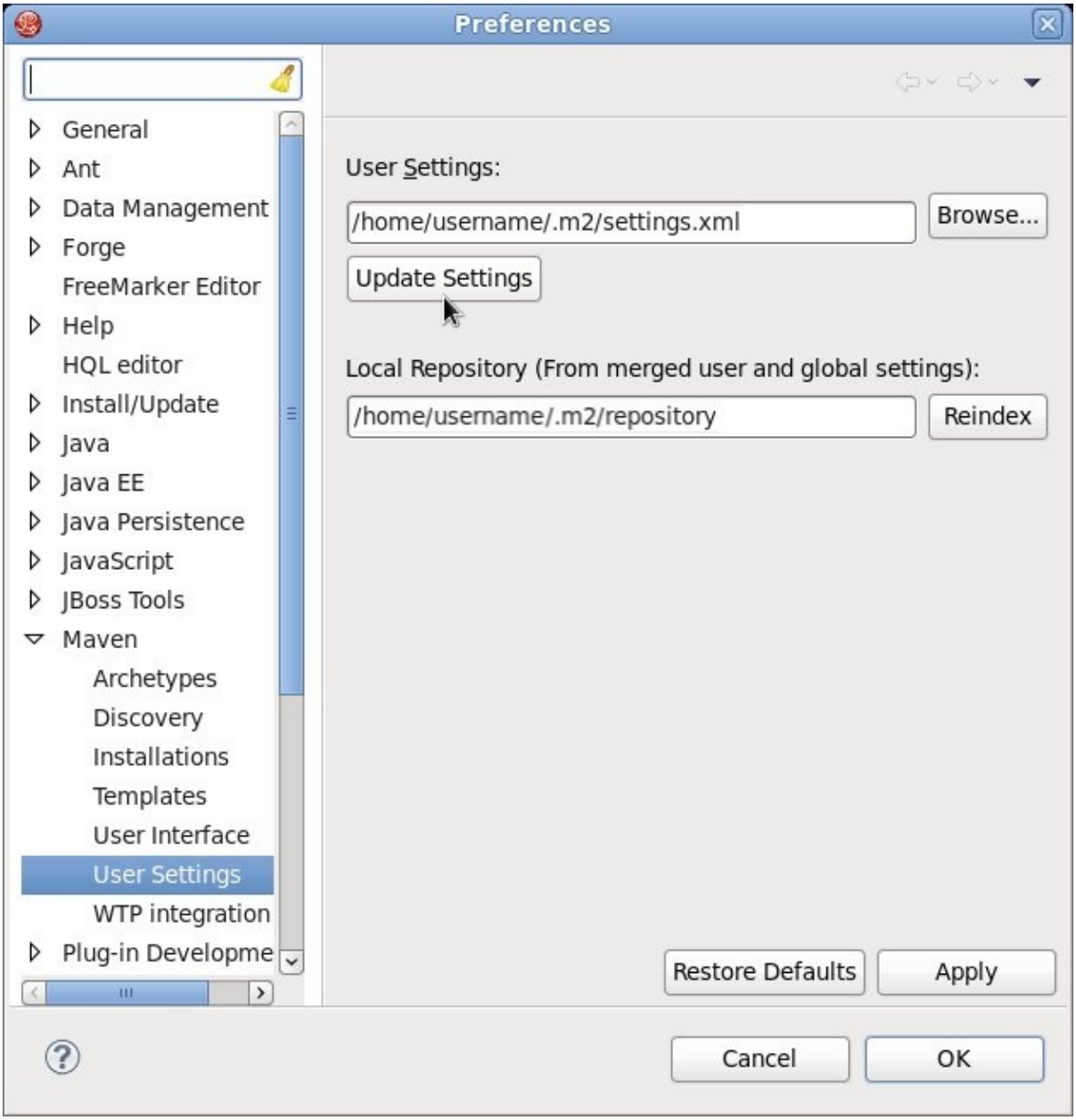
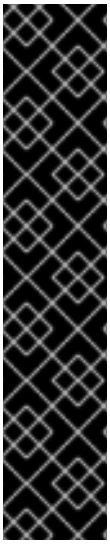


Figure 2.1. Mise à jour des paramètres de configuration de l'utilisateur Maven



IMPORTANT

Si votre référentiel Maven contient des artéfacts obsolètes, vous risquez de rencontrer un des messages d'erreur Maven suivants quand vous créez ou déployez votre projet :

- Missing artifact *ARTIFACT_NAME*
- [ERROR] Failed to execute goal on project *PROJECT_NAME*; Could not resolve dependencies for *PROJECT_NAME*

Pour résoudre ce problème, suppprimer la version mise en cache de votre référentiel local pour forcer un téléchargement des derniers artefacts de Maven. Le référentiel mis en cache e situe dans votre sous-répertoire **`~/.m2/repository/`** dans Linux, et dans le sous-répertoire ***`%SystemDrive%`*`\Users\`*`USERNAME`*`\.m2\repository\`** dans Windows.

Rapporter un bogue

2.3.3. Configurer Maven pour utilisation dans Red Hat JBoss Developer Studio

Les artefacts et les dépendances nécessaires pour construire et déployer des applications dans Red Hat JBoss Enterprise Application Platform sont hébergés sur un référentiel public. Vous devez indiquer à Maven d'utiliser le référentiel quand vous créez vos applications. Cette section couvre les étapes de configuration de Maven si vous souhaitez créer et déployer des application avec Red Hat JBoss Developer Studio.

Maven est distribué dans Red Hat JBoss Developer Studio, donc il n'est pas nécessaire de l'installer séparément. Toutefois, vous devez configurer Maven pour qu'il puisse utiliser l'assistant Java EE Web Project pour les déploiements dans JBoss EAP. La procédure suivante montre comment configurer Maven pour utilisation avec JBoss EAP en éditant le fichier de configuration Maven au sein de Red Hat JBoss Developer Studio.

Procédure 2.10. Configurer Maven dans Red Hat JBoss Developer Studio

1. Cliquer sur **Window**→**Preferences**, puis **`JBoss Tools`** et sélectionner **`JBoss Maven Integration`**.

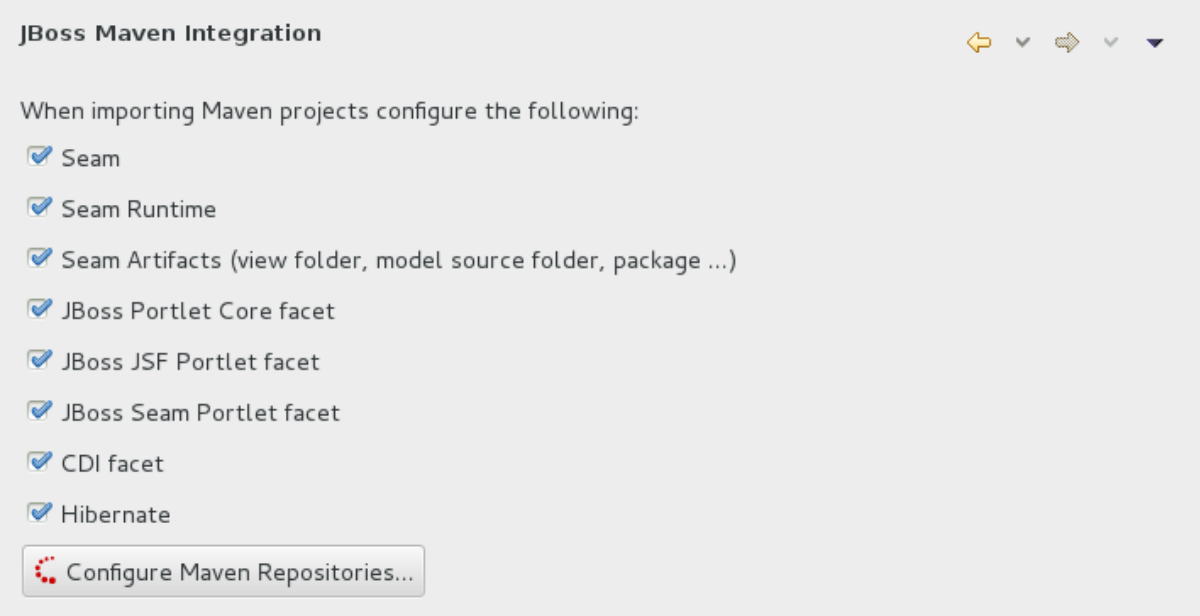


Figure 2.2. Panneau d'intégration de JBoss Maven dans la fenêtre Préférences

2. Cliquer sur **`Configure Maven Repositories`**.
3. Cliquer sur **`Add Repository`** pour configurer le référentiel de JBoss GA Tech Preview Maven. Remplir les champs de **`Add Maven Repository`** comme suit :
   a. Définir les valeurs de **`Profile ID`**, **`Repository ID`**, et **`Repository Name`** à **`jboss-ga-repository`**.
   b. Définir la valeur de **`Repository URL`** à **`http://maven.repository.redhat.com/techpreview/all`**.
   c. Cliquer sur la case **`Active by default`** pour activer le référentiel Maven.
   d. Cliquer sur **`OK`**

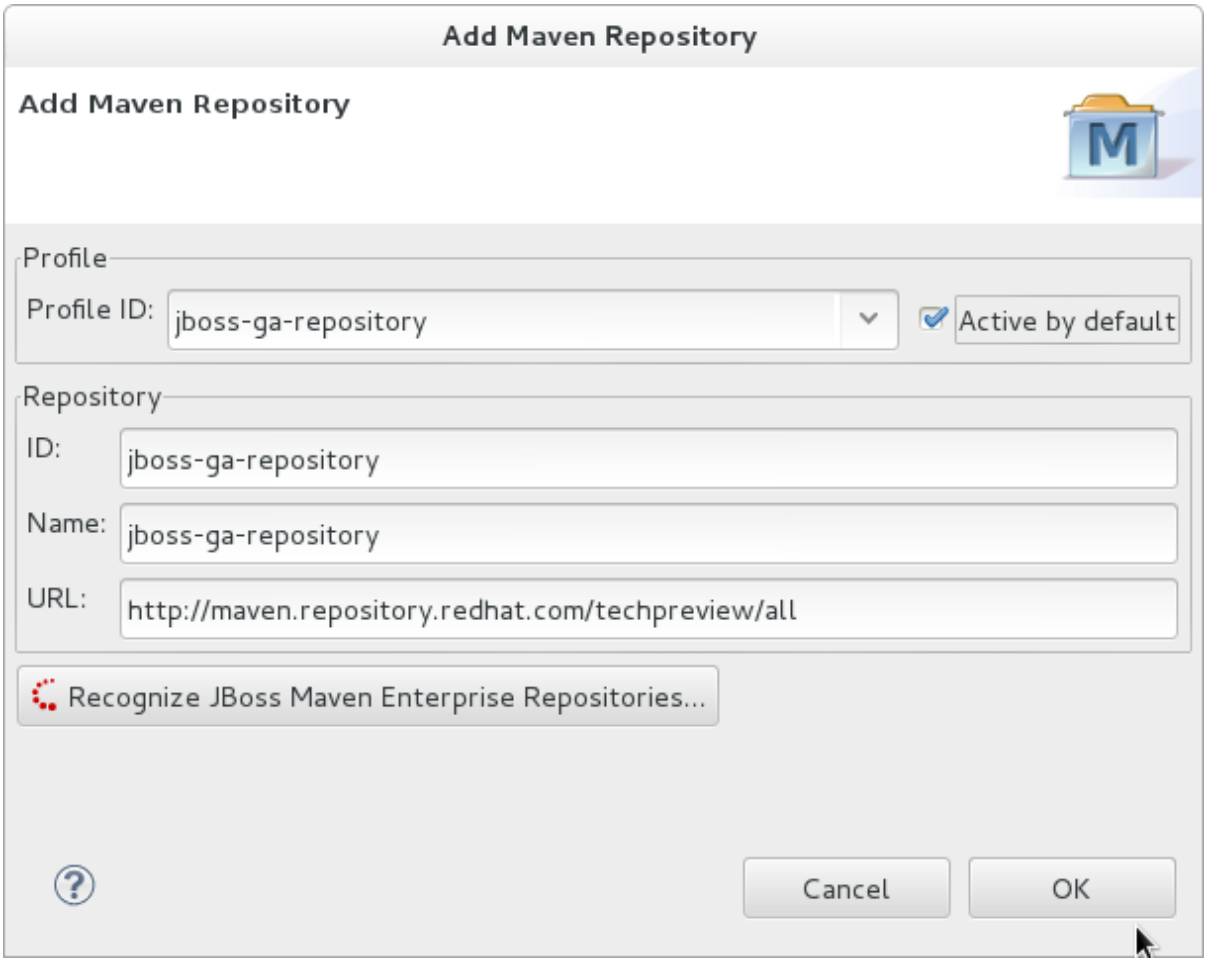


Figure 2.3. Ajouter le référentiel Maven - JBoss Tech Preview

4. Cliquer sur **`Add Repository`** pour configurer le référentiel JBoss Early Access Maven. Remplir les champs de **`Add Maven Repository`** comme suit :

   a. Définir les valeurs de **`Profile ID`**, **`Repository ID`**, et **`Repository Name`** à **`jboss-earlyaccess-repository`**.

   b. Définir la valeur de **`Repository URL`** à **`http://maven.repository.redhat.com/techpreview/all`**.

   c. Cliquer sur la case **`Active by default`** pour activer le référentiel Maven.

   d. Cliquer sur **`OK`**

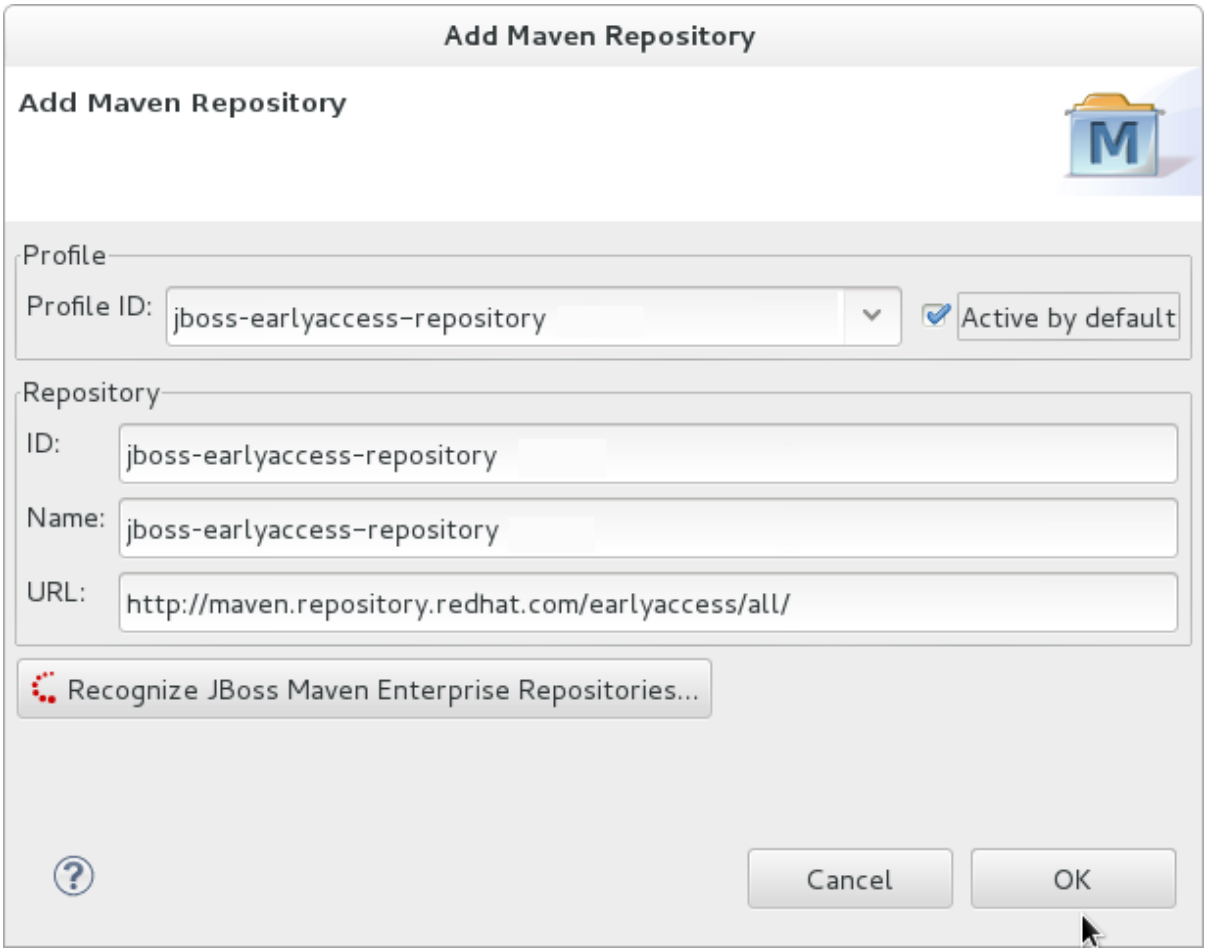


Figure 2.4. Ajouter le référentiel Maven - JBoss Early Access

5. Vérifier les référentiels et cliquer sur **`Finish`**.

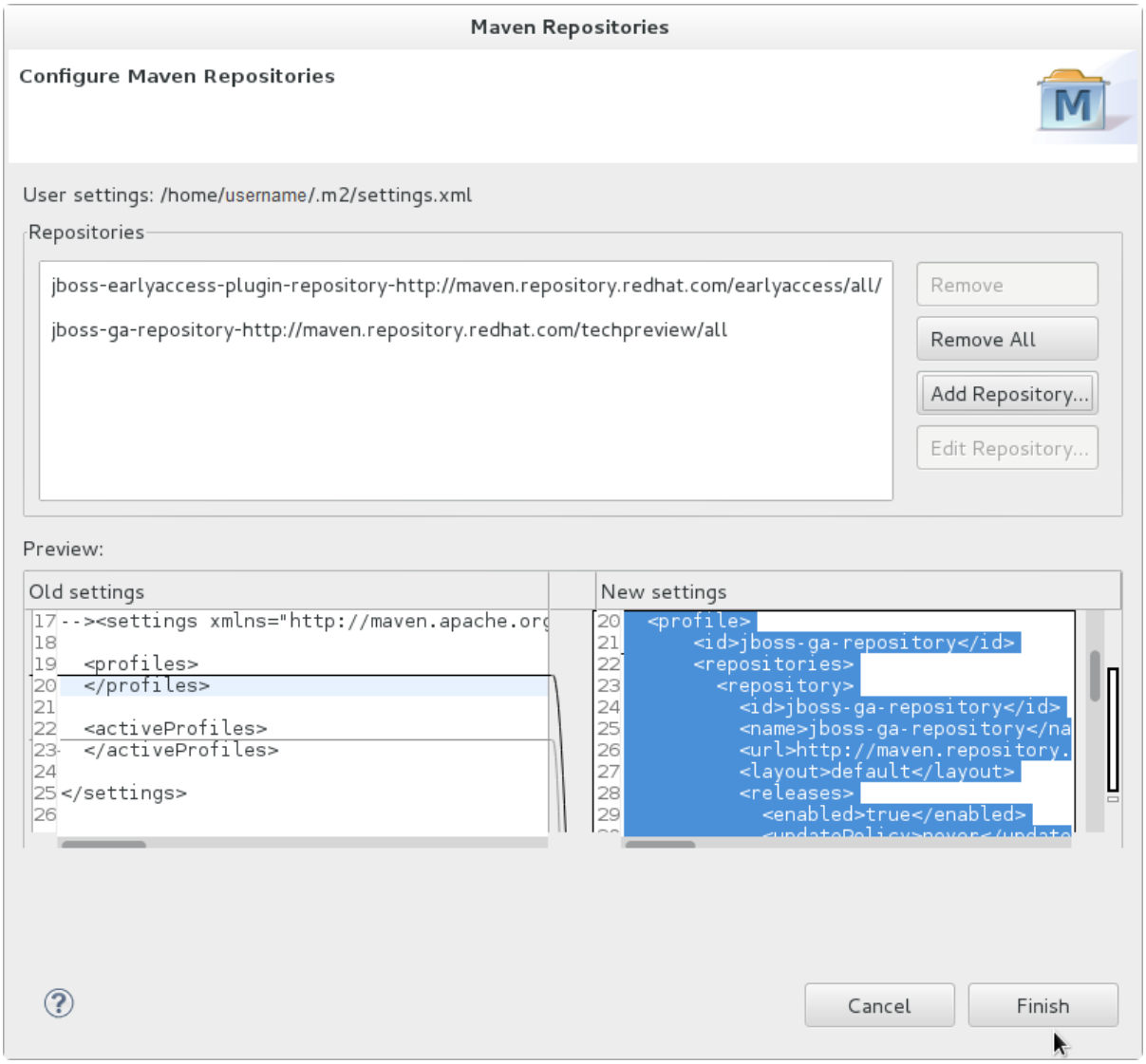


Figure 2.5. Vérifier les référentiels Maven

6. Le message suivant appraîtra "Are you sure you want to update the file 'MAVEN_HOME/settings.xml'?". Cliquer sur **Yes** pour mettre les paramètres de configuration à jour. Cliquer sur **OK** pour fermer la boîte de dialogue.

   Le référentiel Maven est maintenant configuré pour utilisation dans Red Hat JBoss Developer Studio

Rapporter un bogue

2.3.4. Configurer le référentiel JBoss EAP 6 Platform Maven Repository par le Projet POM

Il existe deux approches pour amener Maven à utiliser le référentiel Maven de JBoss EAP 6 dans votre projet :

- Vous pouvez modifier les paramètres Maven.
- Vous pouvez configurer le fichier POM du projet.

Cette tâche vous montre comment configurer un projet particulier pour utiliser le référentiel JBoss EAP 6 Maven Repository en ajoutant des informations de référentiel au projet **`pom.xml`**. Cette méthode de configuration remplace les configurations Utilisateur et Globales.

Cette méthode de configuration est généralement déconseillée. Si vous décidez de configurer des référentiels dans le fichier POM de votre projet, planifiez de manière prudente et souvenez-vous que cela pourrait ralentir votre build et que vous pourriez vous retrouver avec des artefacts ne provenant pas du référentiel attendu.

NOTE

Dans le cas d'une entreprise, où un gestionnaire de référentiel est généralement utilisé, Maven doit interroger tous les artefacts pour tous les projets en utilisant ce gestionnaire. Maven utilise tous les référentiels déclarés pour trouver les artéfacts manquants. Par conséquent, s'il ne peut trouver ce qu'il cherche, il pourra le chercher dans la centrale de référentiels (définie dans le fichier POM parent intégré). Pour remplacer cet emplacement central, vous pouvez ajouter une définition avec **`central`** de manière à ce que la centrale de référentiels devienne également votre gestionnaire de référentiel. Cela fonctionne bien avec les projets établis, mais pose un problème pour les projets vierges ou récents puisque cela crée une dépendance cyclique.

Les fichiers POM inclus de manière transitive représentent également un problème avec ce type de configuration. Maven doit interroger ces référentiels externes pour trouver les artefacts manquants. Non seulement cela ralentira votre build, mais cela fera également perdre le contrôle sur la provenance de vos artefacts et risquera d'endommager votre build.

NOTE

L'URL du référentiel dépend de savoir si le référentiel se trouve sur le système de fichiers ou sur le serveur web. Pour plus d'informations sur l'installation du référentiel, veuillez consulter : Section 2.2.2, « Installer le référentiel Maven de JBoss EAP 6 ». Voici quelques exemples pour chacune des options d'installation :

Système de fichiers

file:///path/to/repo/jboss-eap-6.x-maven-repository

Apache Web Server

http://intranet.acme.com/jboss-eap-6.x-maven-repository/

Gestionnaire de référentiel Nexus

https://intranet.acme.com/nexus/content/repositories/jboss-eap-6.x-maven-repository

1. Ouvrir le fichier **`pom.xml`** de votre projet dans un éditeur de texte.
2. Ajouter la configuration du référentiel suivant. S'il y a déjà une configuration **`<repositories>`** qui se trouve dans le fichier, lui ajouter l'élément **`<repository>`**. Veillez bien à modifier l'**`<url>`** pour qu'il corresponde bien à l'emplacement du référentiel.

```
<repositories>
   <repository>
      <id>jboss-eap-repository-group</id>
      <name>JBoss EAP Maven Repository</name>
      <url>file:///path/to/repo/jboss-eap-6.3.0-maven-
repository/</url>
      <layout>default</layout>
      <releases>
         <enabled>true</enabled>
         <updatePolicy>never</updatePolicy>
```

```
      </releases>
      <snapshots>
         <enabled>true</enabled>
         <updatePolicy>never</updatePolicy>
      </snapshots>
   </repository>
</repositories>
```

3. Ajouter la configuration de référentiel de plug-in suivante. S'il existe déjà une configuration **<pluginRepositories>** dans le fichier, lui ajouter l'élément **<pluginRepository>**.

```
<pluginRepositories>
   <pluginRepository>
      <id>jboss-eap-repository-group</id>
      <name>JBoss EAP Maven Repository</name>
      <url>file:///path/to/repo/jboss-eap-6.3.0-maven-
repository/</url>
      <releases>
         <enabled>true</enabled>
      </releases>
      <snapshots>
         <enabled>true</enabled>
      </snapshots>
   </pluginRepository>
</pluginRepositories>
```

Rapporter un bogue

2.3.5. Gestion des dépendances du projet

Cette section décrit les fichiers POM de nomenclature (NOMENCL ou BOM de l'anglais Bills of Material) de Red Hat JBoss Enterprise Application Platform 6.

Un fichier BOM est un fichier Maven **pom.xml** (POM) qui indique les versions de toutes les dépendances d'exécution d'un module donné. Les dépendances de versions sont listées dans la section de gestion des dépendances du fichier.

Un projet utilise un NOMENCL en ajoutant son **groupId:artifactId:version** (GAV) à la section de gestion des dépendances du fichier **pom.xml** du projet et en spécifiant **<scope>import</scope>** et les valeurs de l'élément **<type>pom</type>**.

NOTE

Dans bien des cas, les dépendances des fichiers POM du projet donné utilisent le scope **fourni**. C'est parce que ces classes sont fournies par le serveur d'applications en cours d'exécution et qu'il n'est pas nécessaire de les empaqueter avec l'application utilisateur.

Artéfacts Maven pris en charge

Dans le cadre du processus de génération du produit, tous les composants d'exécution de JBoss EAP sont construits à partir de la source dans un environnement contrôlé. Cela permet d'assurer que les objets binaires ne contiennent pas de code malveillant, et qu'ils puissent être soutenus pour la durée de vie du produit. Ces objets peuvent être facilement identifiés par le qualificateur de version **-redhat**, par exemple **1.0.0-redhat-1**.

L'ajout d'un artefact pris en charge pour le fichier **pom.xml** de configuration de build garantit que le build utilise l'artefact binaire correct pour la génération de builds locaux et pour le testing. Notez qu'un artefact avec une version **-redhat** ne fait pas nécessairement partie de l'API public prise en charge et peut changer à l'avenir au cours des révisions. Pour plus d'informations sur l'API public pris en charge, voir la documentation JavaDoc incluse dans la sortie.

Ainsi, pour pouvoir utiliser la version d'Hibernate prise en charge, ajouter quelque chose de similaire à votre configuration de build.

```
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-core</artifact>
  <version>4.2.6.Final-redhat-1</version>
  <scope>provided</scope>
</dependency>
```

Notez que l'exemple ci-dessus contient une valeur pour le champ **<version/>**. Cependant, il est recommandé d'utiliser la gestion des dépendances Maven pour configurer les versions de dépendances.

Gestion des dépendances

Maven inclut un mécanisme de gestion des versions de dépendances directes et transitives par le build. Pour obtenir des informations générales sur la gestion des dépendances, voir le projet Apache Maven Project Introduction to the Dependency Mechanism.

Utiliser une ou plusieurs dépendances JBoss prises en charge directement dans votre build ne garantit pas que toutes les dépendances transitives du build seront des artéfacts JBoss totalement pris en charge. Il est commun pour les builds Maven d'utiliser un mélange de sources d'artéfacts en provenance du référentiel central Maven, du référentiel Maven JBoss.org, ou autres référentiels Maven.

Dans le référentiel JBoss EAP Maven, il y a une NOMENCL de dépendance de gestion, qui spécifie tous les artefacts binaires de JBoss EAP pris en charge. Cette nomenclature peut être utilisée dans un build pour s'assurer que Maven donnera la priorité à des dépendances JBoss EAP prises en charge pour toutes les dépendances directes et transitives du build. En d'autres termes, les dépendances transitives seront gérées en fonction de la version de dépendance qui conviendra suivant le cas. La version de BOM correspond à celle de la version de JBoss EAP.

```
<dependencyManagement>
  <dependencies>
    ...
    <dependency>
      <groupId>org.jboss.bom</groupId>
      <artifactId>eap6-supported-artifacts</artifactId>
      <version>6.3.0.GA</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
    ...
  </dependencies>
</dependencyManagement>
```

JBoss JavaEE Specs Bom de nomenclature

Le code BOM **jboss-javaee-6.0** contient les JAR de spécification JAVA EE utilisés dans JBoss EAP.

Pour utiliser cette nomenclature BOM dans un projet, ajouter une dépendance au GAV qui contient la version du JSP et des JARS de l'API du serveur qui doivent être créés et déployer l'application.

L'exemple suivant utilise la version **`3.0.2.Final-redhat-x`** de la NOMENCL **`jboss-javaee-6.0`**.

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.jboss.spec</groupId>
      <artifactId>jboss-javaee-6.0</artifactId>
      <version>3.0.2.Final-redhat-x</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
    ...
  </dependencies>
</dependencyManagement>

<dependencies>
  <dependency>
    <groupId>org.jboss.spec.javax.servlet</groupId>
    <artifactId>jboss-servlet-api_3.0_spec</artifactId>
    <scope>provided</scope>
  </dependency>
  <dependency>
    <groupId>org.jboss.spec.javax.servlet.jsp</groupId>
    <artifactId>jboss-jsp-api_2.2_spec</artifactId>
    <scope>provided</scope>
  </dependency>
  ...
</dependencies>
```

Les nomenclatures JBoss EAP BOM et les Quickstarts

Les fichiers JBoss EAP BOM du projet jboss-bom se trouvent à l'adresse suivante https://github.com/jboss-developer/jboss-eap-boms.

Les quickstarts fournissent des exemples de cas d'utilisation primaires du référentiel Maven. Le tableau suivant dresse une liste des NOMENCL (BOM) utilisées par les quickstarts.

Tableau 2.1. Nomenclatures JBoss BOM utilisées par les Quickstarts

Id Artéfact Maven	Description
jboss-javaee-6.0-with-hibernate	Cette nomenclature BOM s'appuie sur le profil BOM complet de Java EE, ajoutant des projets communautaires Hibernate ORM, Hibernate Search et Hibernate Validator. Elle fournit également des projets d'outil tel que Hibernate JPA Model Gen et le processeur Hibernate Validator Annotation.
jboss-javaee-6.0-with-hibernate3	Cette nomenclature BOM s'appuie sur le profil BOM complet de Java EE, ajoutant des projets communautaires Hibernate 3 ORM, Hibernate Entity Manager (JPA 1.0) et Hibernate Validator.

Id Artéfact Maven	Description
jboss-javaee-6.0-with-logging	Cette nomenclature BOM s'appuie sur le profil BOM complet de Java EE, ajoutant JBoss Logging Tools et Log4j framework.
jboss-javaee-6.0-with-osgi	Cette nomenclature BOM s'appuie sur le profil BOM complet de Java EE, ajoutant OSGI.
jboss-javaee-6.0-with-resteasy	Cette nomenclature BOM s'appuie sur le profil BOM complet de Java EE, ajoutant RESTEasy
jboss-javaee-6.0-with-security	Cette nomenclature BOM s'appuie sur le profil BOM complet de Java EE, ajoutant Picketlink.
jboss-javaee-6.0-with-tools	Cette nomenclature BOM s'appuie sur le profil BOM complet de Java EE, ajoutant Arquillian à l'ensemble. Elle procure également une version de JUnit et de TestNG conseillés pour l'utilisation avec Arquillian.
jboss-javaee-6.0-with-transactions	Cette nomenclature BOM inclut un gestionnaire de transactions de classe mondiale. Utiliser les API JBossTS pour pouvoir profiter de toutes ses fonctionnalités.

L'exemple suivant utilise la version **`6.3.0.GA`** de la nomemclature BOM **`jboss-javaee-6.0-with-hibernate`**.

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.jboss.bom.eap</groupId>
      <artifactId>jboss-javaee-6.0-with-hibernate</artifactId>
      <version>6.3.0.GA</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
    ...
  </dependencies>
</dependencyManagement>

<dependencies>
  <dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <scope>provided</scope>
  </dependency>
  ...
</dependencies>
```

JBoss Client BOMs

Les JBoss EAP Builds comprennent deux nomenclatures BOM : **`jboss-as-ejb-client-bom`** et **`jboss-as-jms-client-bom`**.

Les NOMENCL de clients ne créent pas de section de gestion des dépendances, ni ne définissent les dépendances. Au lieu de cela, ces nomenclatures se rajoutent à d'autres BOM et sont utilisées pour empaqueter un ensemble de dépendances nécessaires pour les cas d'utilisation de clients distants.

L'exemple suivant utilise la version **`7.4.0.Final-redhat-x`** du client **`jboss-as-ejb-client-bom`** BOM.

```
<dependencies>
  <dependency>
    <groupId>org.jboss.as</groupId>
    <artifactId>jboss-as-ejb-client-bom</artifactId>
    <version>7.4.0.Final-redhat-x</version>
    <type>pom</type>
  </dependency>
  ...l
</dependencies>
```

Cet exemple utilise la version **`7.4.0.Final-redhat-x`** du client BOM **`jboss-as-jms-client-bom`**.

```
<dependencies>
  <dependency>
    <groupId>org.jboss.as</groupId>
    <artifactId>jboss-as-jms-client-bom</artifactId>
    <version>7.4.0.Final-redhat-x</version>
    <type>pom</type>
  </dependency>
  ...
</dependencies>
```

Pour plus d'informations sur les dépendances Maven et sur les fichiers BOM POM, consulter Apache Maven Project - Introduction to the Dependency Mechanism.

Rapporter un bogue

2.4. MISE À NIVEAU DU RÉFÉRENTIEL MAVEN

2.4.1. Appliquer un correctif dans le répertoire Maven local

Résumé

Un référentiel Maven stocke des bibliothèques Java, des plug-ins et autres objets nécessaires pour compiler et déployer des applications JBoss EAP. Le référentiel de JBoss EAP est disponible en ligne ou en fichier ZIP téléchargé. Si vous utilisez le référentiel hébergé publiquement, les mises à jour seront appliquées automatiquement. Toutefois, si vous téléchargez et installez le repository Maven localement, vous serez chargé d'appliquer les mises à jour vous-même. Chaque fois qu'un correctif sera disponible dans JBoss EAP, un correctif correspondant sera fourni pour le référentiel JBoss EAP Maven. Ce correctif est disponible sous la forme d'un fichier ZIP incrémentiel qui est décompressé dans le référentiel local existant. Le fichier ZIP contient des nouveaux fichiers JAR et POM. Il n'écrase pas les JAR existants, ni ne les supprime, donc il n'y a aucune exigence de restauration.

Pour plus d'informations sur le Management CLI, voir le chapitre intitulé *Patching and Upgrading JBoss EAP 6* de *Installation Guide* concernant JBoss Enterprise Application Platform 6 qui se situe sur le Portail clients à l'adresse suivante :
https://access.redhat.com/site/documentation/JBoss_Enterprise_Apnplication_Platform/.

Cette tâche décrit comment appliquer les mises à jour de Maven dans votre référentiel Maven en utilisant la commande **`unzip`**.

Conditions préalables

- Accès valide et abonnement au portail clients de Red Hat.
- Le fichier Red Hat JBoss Enterprise Application Platform 6.3.0 Maven Repository ZIP, téléchargé et installé localement.

Procédure 2.11. Mise à jour du référentiel Maven

1. Ouvrir un navigateur et connectez-vous dans https://access.redhat.com.
2. Sélectionner **Téléchargements** du menu en haut de la page.
3. Cherchez **`Red Hat JBoss Enterprise Application Platform`** dans la liste et cliquez dessus.
4. Sélectionner la version de JBoss EAP qui convient à partir du menu déroulant **`Version`** qui apparaît sur l'écran, puis cliquer sur **`Patches`**.
5. Chercher **`Red Hat JBoss Enterprise Application Platform 6.3 CP`*x*** **`Incremental Maven Repository`** dans la liste et cliquer sur **`Download`**.
6. Vous êtes invités à sauvegarder le fichier ZIP dans un répertoire de votre choix. Sélectionner un répertoire et sauvegardez le fichier.
7. Chercher le chemin d'accès vers le répertoire JBoss EAP dont il s'agit dans les commandes ci-dessous, sous *EAP_MAVEN_REPOSITORY_PATH*, correspondant à votre système d'exploitation. Pour obtenir plus d'informations sur la façon d'installer le référentiel Maven sur le système de fichiers local, voir Section 2.2.3, « Installer le référentiel Maven de JBoss EAP 6 localement ».
8. Décompresser le fichier de correctifs Maven dans le répertoire d'installation de JBoss EAP 6.3.*x*.
   - Dans Linux, ouvrir un terminal et saisir la commande suivante :

     ```
     [standalone@localhost:9999 /] unzip -o jboss-eap-6.3.x-
     incremental-maven-repository.zip -d EAP_MAVEN_REPOSITORY_PATH
     ```

   - Dans Windows, utiliser l'utilitaire pour extraire le fichier ZIP pour le mettre dans la racine du répertoire***`EAP_MAVEN_REPOSITORY_PATH`*** .

Résultat

Le référentiel Maven installé localement est mis à jour au dernier correctif.

Rapporter un bogue

CHAPITRE 3. CHARGEMENT DE CLASSES MODULAIRE ET MODULES

3.1. INTRODUCTION

3.1.1. Chargement des classes de chargement et de modules

JBoss EAP 6 utilise un nouveau système de chargement de classes modulaire pour contrôler les chemins de classe d'applications déployées. Ce système offre plus de souplesse et de contrôle que le système traditionnel des chargeurs de classes hiérarchiques. Les développeurs ont un contrôle précis des classes disponibles à leurs applications et peuvent configurer un déploiement pour qu'il puisse ignorer les classes fournies par le serveur d'application en faveur de leurs propres classes.

Le nouveau chargeur de classes modulaire sépare toutes les classes Java en des groupes logiques appelés modules. Chaque module peut définir des dépendances sur d'autres modules de manière à ajouter les classes de ces modules dans leur propre chemin de classe. Le fait que chaque fichier JAR et WAR déployé soit traité comme un module permet aux développeurs de contrôler le contenu du chemin de classe de leur application en ajoutant une configuration de module à leur application.

Rapporter un bogue

3.1.2. Chargement des classes

Le chargement des classes est un mécanisme par lequel les classes et les ressources Java sont chargées dans l'environnement d'exécution de Java (Java Runtime Environment).

Rapporter un bogue

3.1.3. Modules

Un module est un regroupement logique des classes utilisées pour le chargement de classes et pour la gestion des dépendances. JBoss EAP 6 identifie deux types de modules, parfois appelés modules statiques et dynamiques. Cependant, la seule différence entre les deux est la façon dont ils sont empaquetés. Tous les modules offrent les mêmes caractéristiques.

Modules statiques

Les modules statiques sont prédéfinis dans le répertoire ***EAP_HOME*/modules/** du serveur d'applications. Chaque sous-répertoire représente un module et contient un fichier de configuration (**module.xml**) et tout fichier JAR requis. Le nom du module est défini dans le fichier **module.xml**. Tous les API fournis par le serveur de l'application sont des modules statiques, y compris les API Java EE, et les autres API comme JBoss Logging.

Exemple 3.1. Exemple de fichier module.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<module xmlns="urn:jboss:module:1.0" name="com.mysql">
  <resources>
    <resource-root path="mysql-connector-java-5.1.15.jar"/>
  </resources>
  <dependencies>
    <module name="javax.api"/>
```

```
    <module name="javax.transaction.api"/>
  </dependencies>
</module>
```

Le nom du module, **com.mysql**, doit correspondre à la structure du répertoire du module.

Les modules fournis dans les distributions JBoss EAP se trouvent dans un répertoire **system** se trouvant lui-même dans le répertoire ***JBOSS_HOME*/modules**. Cela les rend séparés de tout module fourni par une tierce partie.

Tout produit mis en couche de Red Hat, se superposant sur JBoss EAP 6.1 ou version supérieure installera également leurs modules dans le répertoire **system**.

La création de modules statiques personnalisés peut être utile si plusieurs applications sont déployées sur un même serveur utilisant les mêmes bibliothèques de tierce partie. Au lieu d'un regroupement de ces bibliothèques pour chaque application, un module contenant ces bibliothèques peut être créé et installé par l'administrateur JBoss. Les applications peuvent ensuite déclarer une dépendance explicite sur les modules statiques personnalisés.

Les utilisateurs doivent s'assurer que les modules personnalisés soient installés dans le répertoire ***JBOSS_HOME*/modules**, en utilisant un répertoire par couche de modules. Cela garantit que les versions personnalisées de modules qui existent déjà dans le répertoire **system** soient bien chargées à la place des versions fournies. Ainsi, les modules utilisateur auront la priorité sur les modules fournis par le système.

Si vous utilisez la variable d'environnement **JBOSS_MODULE_PATH** pour changer les emplacements où JBoss EAP cherche les modules, le produit ira chercher dans une structure de sous-répertoire **system** dans un des emplacements spécifiés. Une structure de sous-répertoire **system** doit exister quelquepart dans les emplacements spécifiés dans **JBOSS_MODULE_PATH**.

Modules dynamiques

Les modules dynamiques sont créés et chargés par le serveur d'application pour chaque déploiement JAR ou WAR (ou sous-déploiement d'un EAR). Le nom d'un module dynamique est dérivé du nom de l'archive déployée. Comme les déploiements sont chargés sous forme de modules, ils peuvent configurer des dépendances et peuvent être utilisés comme dépendances par d'autres déploiements.

Les modules ne sont chargés qu'en fonction des besoins. Cela a généralement lieu quand une application est déployée avec des dépendances implicites ou explicites.

Rapporter un bogue

3.1.4. Les dépendances de modules

Une dépendance de module est une déclaration qui indique qu'un module a besoin des classes d'un autre module pour pouvoir fonctionner. Les modules peuvent déclarer leurs dépendances sur un certain nombre d'autres modules. Quand le serveur d'applications charge un module, le chargeur de classes de module traite les dépendances de ce module et ajoute les classes de chaque dépendance à son chemin de classe. Si une dépendance particulière est introuvable, le module ne pourra pas charger.

Les applications déployées (JAR et WAR) sont chargées sous forme de modules dynamiques et utilisent des dépendances pour accéder aux API fournies par JBoss EAP 6.

Il y a deux types de dépendances : explicite et implicite.

Les dépendances explicites sont déclarées dans la configuration par le développeur. Les modules statiques peuvent déclarer des dépendances dans le fichier modules.xml. Les modules dynamiques peuvent avoir des dépendances déclarées dans les descripteurs de déploiement MANIFEST.MF ou jboss-deployment-structure.xml.

Les dépendances explicites peuvent être spécifiées comme étant optionnelles. Une erreur de chargement de dépendance optionnelle n'entraînera pas l'annulation d'un chargement de module. Cependant, si la dépendance est rendue disponible par la suite, elle ne sera PAS ajoutée au chemin de classe du module. Les dépendances doivent être rendues disponibles quand le module est chargé.

Des dépendances implicites sont ajoutées automatiquement par le serveur d'applications quand certaines conditions ou meta-données se trouvent dans un déploiement. Les API Java EE 6 fournies avec JBoss EAP 6 sont des exemples de modules ajoutés par détection de dépendances implicites dans les déploiements.

Les déploiements peuvent également être configurés de façon à exclure des dépendances implicites particulières. Il vous faut pour cela le fichier de déploiement jboss-deployment-structure.xml. C'est normalement le cas quand une application empaquète une version spécifique de bibliothèque que le serveur d'applications tentera d'ajouter comme dépendance implicite.

Un chemin de classe de module ne contient que ses propres classes et celles de ses dépendances immédiates. Un module n'est pas en mesure d'accéder aux classes des dépendances. Cependant, un module peut indiquer quand une dépendance explicite est exportée. Une dépendance exportée est fournie à tout module qui dépend du module qui l'exporte.

> **Exemple 3.2. Les dépendances de module**
>
> Le Module A dépend du Module B et le Module B dépend su Module C. Le Module A peut accéder aux classes du Module B, et le Module B peut accéder aux classes du Module C. Le Module C ne peut pas accéder aux classes du Module C à moins que :
>
> - Le Module A déclare une dépendance explicite sur le Module C, ou bien
> - Le Module B exporte ses dépendances sur le Module C.

Rapporter un bogue

3.1.5. Chargement des classes dans les déploiements

Dans le cadre du chargement de classes, tous les déploiements sont traités sous forme de modules par JBoss EAP 6. On les appelle des modules dynamiques. Les comportements de chargement de classes varient selon le type de déploiement.

Déploiement WAR

Un déploiement WAR est un simple module. Les classes du répertoire **`WEB-INF/lib`** sont considérées de la même manière que celles du répertoire **`WEB-INF/classes`**. Toutes les classes empaquetées dans le war seront téléchargées par le même chargeur de classes.

Déploiement EAR

Les déploiements EAR sont faits de plus d'un module. La définition de ces modules suit ces règles :

1. Le répertoire **`lib/`** du EAR est un simple module nommé le module parent.

2. Chaque déploiement WAR du EAR est un simple module.

3. Chaque déploiement EJB JAR du EAR est un simple module.

Les modules de sous-déploiement (les déploiements WAR et JAR dans EAR) ont une dépendance automatique sur le module parent. Cependant ils n'ont pas de dépendance automatique l'un sur l'autre. Ceci est appelé l'isolement du sous-déploiement et peut être désactivé par déploiement ou pour le serveur de toute l'application dans son ensemble.

Les dépendances explicites entre les modules de sous-déploiement peuvent être ajoutées par les mêmes moyens, tout comme pour tout autre module.

Rapporter un bogue

3.1.6. Précédence pour le chargement des classes

Le chargeur de classes modulaires JBoss EAP 6 utilise un système de priorité pour empêcher le chargement des conflits de classes.

Pendant le déploiement, la liste complète des packages et des classes est créée pour chaque déploiement et chacune de ses dépendances. La liste est ordonnée selon les règles de priorité de chargement de classes. Lors du chargement de classes au moment de l'exécution, le chargeur de classes recherche dans cette liste et charge la première correspondance. Cela empêche plusieurs copies des mêmes classes et packages dans le chemin d'accès de classe des déploiements d'entrer en conflit l'un avec l'autre.

Le chargeur de classe charge les classes dans l'ordre suivant, du plus élevé au plus bas :

1. Dépendances implicites.

   Voici les dépendances qui sont ajoutées automatiquement par JBoss EAP 6, comme les API JAVA EE. Ces dépendances ont la plus haute précédence de chargeur de classe car ils contiennent des API et des fonctionnalités communes qui sont fournies par JBoss EAP 6.

   Voir Section 3.8.1, « Dépendances de modules implicites » pour obtenir des détails sur chaque dépendance implicite.

2. Dépendances explicites

   Il s'agit de dépendances qui sont ajoutées manuellement dans la configuration de l'application. Cela peut être fait à l'aide du fichier d'application **`MANIFEST.MF`** ou du nouveau fichier du descripteur de déploiement en option de JBoss **`jboss-deployment-structure.xml`**.

   Voir Section 3.2, « Ajouter une dépendance de module explicite à un déploiement » pour apprendre comment ajouter des dépendances explicites.

3. Ressources locales.

   Fichiers de classe empaquetées à l'intérieur du déploiement lui-même, par ex. les répertoires **`WEB-INF/classes`** ou **`WEB-INF/lib`** d'un fichier WAR.

4. Dépendances inter-déploiement.

   Ce sont des dépendances sur les autres déploiements d'un déploiement EAR. Cela peut inclure des classes du répertoire **`lib`** du EAR ou des classes des autres jars EJB.

Rapporter un bogue

3.1.7. Nommage de modules dynamiques

Tous les déploiements sont chargés en tant que modules par JBoss EAP 6 et sont nommés en fonction des conventions suivantes :

1. Les déploiements des fichiers WAR et JAR sont nommés selon le format suivant :

   ```
   deployment.DEPLOYMENT_NAME
   ```

   Par exemple, **`inventory.war`** et **`store.jar`** auront les mêmes noms de module que **`deployment.inventory.war`** et **`deployment.store.jar`** respectivement.

2. Les sous-déploiements des archives Enterprise sont nommés selon le format suivant :

   ```
   deployment.EAR_NAME.SUBDEPLOYMENT_NAME
   ```

   Ainsi, le sous-déploiement **`reports.war`**, qui se trouve dans l'archive Enterprise **`accounts.ear`**, aura le nom de module du **`deployment.accounts.ear.reports.war`**.

Rapporter un bogue

3.1.8. jboss-deployment-structure.xml

`jboss-deployment-structure.xml` est un descripteur de déploiement optionnel de JBoss EAP 6. Ce descripteur de déploiement fournit un contrôle pour le chargement de classes dans le déploiement.

Le schéma XML de ce descripteur de déploiement se trouve dans ***`EAP_HOME`*`/docs/schema/jboss-deployment-structure-1_2.xsd`**

Rapporter un bogue

3.2. AJOUTER UNE DÉPENDANCE DE MODULE EXPLICITE À UN DÉPLOIEMENT

Cette tâche montre comment ajouter une dépendance explicite à une application. On peut ajouter des dépendances explicites de module aux applications pour ajouter les classes de ces modules au chemin de classe de l'application lors du déploiement.

Certaines dépendances sont automatiquement ajoutées aux déploiements par JBoss EAP 6. Voir Section 3.8.1, « Dépendances de modules implicites » pour plus d'informations.

Conditions préalables

1. Vous devez déjà avoir un projet de logiciel qui fonctionne, et auquel vous souhaitez ajouter une dépendance de module.

2. Vous devez connaître le nom du module qui est ajouté comme dépendance. Voir Section 3.8.2, « Les modules inclus » pour obtenir la liste des modules statiques inclus dans JBoss EAP 6. Si le module correspond à un autre déploiement, voir Section 3.1.7, « Nommage de modules dynamiques » pour déterminer le nom du module.

Les dépendances peuvent être configurées par deux méthodes différentes :

1. Par l'ajout d'entrées dans le fichier **MANIFEST.MF** du déploiement.

2. Par l'ajout d'entrées dans le descripteur de déploiement **jboss-deployment-structure.xml**.

Procédure 3.1. Par l'ajout d'une configuration de dépendance à MANIFEST.MF

Les projets Maven peuvent être configurés pour créer les entrées de dépendance requises par le fichier **MANIFEST.MF**. Voir Section 3.3, « Générer des entrées MANIFEST.MF en utilisant Maven ».

1. **Ajouter le fichier MANIFEST.MF**
   Si le projet ne possède pas de fichier **MANIFEST.MF**, créer un fichier nommé **MANIFEST.MF**. Pour une application web (WAR), ajouter ce fichier au répertoire **META-INF**. Pour une archive EJB (JAR), l'ajouter au répertoire **META-INF**.

2. **Ajouter une entrée de dépendance**
   Ajouter une entrée de dépendance au fichier **MANIFEST.MF** avec une liste de noms de modules de dépendance séparés par des virgules.

   ```
   Dépendances : org.javassist, org.apache.velocity
   ```

3. **Option : rendre une dépendance optionnelle**
   On peut rendre une dépendance optionnelle an ajoutant **optional** au nom du module de l'entrée de dépendance.

   ```
   Dépendances : org.javassist optional, org.apache.velocity
   ```

4. **Option : export d'une dépendance**
   On peut exporter une dépendance en ajoutant **export** au nom du module de l'entrée de dépendance.

   ```
   Dépendences : org.javassist, org.apache.velocity export
   ```

Procédure 3.2. Ajouter une configuration de dépendance à jboss-deployment-structure.xml

1. **Ajouter jboss-deployment-structure.xml**
   Si l'application n'a pas de fichier **jboss-deployment-structure.xml**, créer un nouveau fichier nommé **jboss-deployment-structure.xml** et l'ajouter au projet. Ce fichier est un fichier XML ayant l'élément racine de **<jboss-deployment-structure>**.

   ```
   <jboss-deployment-structure>

   </jboss-deployment-structure>
   ```

   Pour une application web (WAR), ajouter ce fichier au répertoire **WEB-INF**. Pour une archive EJB (JAR), l'ajouter au répertoire **META-INF**.

2. **Ajouter une section de dépendances**
   Créer un élément **<deployment>** à l'intérieur de la racine du document et un élément **<dependencies>** également.

3. **Ajouter les éléments du module**

Dans le nœud de dépendances, ajouter un élément de module pour chaque dépendance du module. Définir l'attribut **name** au nom du module.

```
<module name="org.javassist" />
```

4. **Option : rendre une dépendance optionnelle**
   On peut rendre une dépendance optionnelle en ajoutant l'attribut **optional** à l'entrée du module, avec la valeur **TRUE**. La valeur par défaut de cet attribut est **FALSE**.

```
<module name="org.javassist" optional="TRUE" />
```

5. **Option : export d'une dépendance**
   On peut exporter une dépendance en ajoutant l'attribut **optional** à l'entrée du module, avec la valeur **TRUE**. La valeur par défaut de cet attribut est **FALSE**.

```
<module name="org.javassist" export="TRUE" />
```

Exemple 3.3. jboss-deployment-structure.xml avec deux dépendances

```
<jboss-deployment-structure>

   <deployment>

      <dependencies>
         <module name="org.javassist" />
         <module name="org.apache.velocity" export="TRUE" />
      </dependencies>

   </deployment>

</jboss-deployment-structure>
```

JBoss EAP 6 ajoutera les classes à partir des modules spécifiés dans le chemin de classes de l'application quand elle sera déployée.

Rapporter un bogue

3.3. GÉNÉRER DES ENTRÉES MANIFEST.MF EN UTILISANT MAVEN

Les projets Maven qui utilisent les plug-ins JAR, EJB ou WAR peuvent générer un fichier **MANIFEST.MF** avec une entrée **Dependencies**. Ce procédé ne génère pas automatiquement la liste de dépendances, il crée seulement le fichier **MANIFEST.MF** avec les informations spécifiées dans le fichier **pom.xml**.

Conditions préalables

1. Vous devez déjà posséder un projet Maven en cours.
2. Le projet Maven doit utiliser l'un des plug-ins JAR, EJB, ou WAR (**maven-jar-plugin**, **maven-ejb-plugin**, **maven-war-plugin**).
3. Vous devez connaître le nom des dépendances de module du projet. Voir Section 3.8.2, « Les

modules inclus » pour obtenir la liste des modules statiques inclus dans JBoss EAP 6. Si le module est dans un autre déploiement, voir Section 3.1.7, « Nommage de modules dynamiques » pour déterminer le nom du module.

Procédure 3.3. Générer un fichier MANIFEST.MF contenant des dépendances de module.

1. **Ajouter une configuration**
   Ajouter la configuration suivante à la configuration du plug-in de l'empaquetage dans le fichier **`pom.xml`** du projet.

```
<configuration>
   <archive>
      <manifestEntries>
         <Dependencies></Dependencies>
      </manifestEntries>
   </archive>
</configuration>
```

2. **Liste de dépendances**
   Ajouter la liste des dépendances de module dans l'élement <Dependencies>. Utiliser le même format utilisé lors de l'ajout des dépendances dans le **`MANIFEST.MF`**. Veuillez consulter Section 3.2, « Ajouter une dépendance de module explicite à un déploiement » pour plus d'informations sur ce format.

```
<Dependencies>org.javassist, org.apache.velocity</Dependencies>
```

3. **Build de projet**
   Procédez au build de projet en utilisant l'objectif d'assemblage de Maven.

```
[Localhost ]$ mvn assembly:assembly
```

Quand un build de projet est réalisé en utilisant l'objectif d'assemblage, l'archive finale comprend un fichier **`MANIFEST.MF`** avec les dépendances de module spécifiées.

Exemple 3.4. Dépendances de module configurées dans pom.xml

L'exemple ci-dessous montre le plug-in WAR mais il fonctionne également avec les plug-ins JAR et EJB (maven-jar-plugin et maven-ejb-plugin).

```
<plugins>
   <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-war-plugin</artifactId>
      <configuration>
         <archive>
            <manifestEntries>
               <Dependencies>org.javassist,
org.apache.velocity</Dependencies>
            </manifestEntries>
         </archive>
      </configuration>
   </plugin>
</plugins>
```

Rapporter un bogue

3.4. EMPÊCHER QU'UN MODULE SOIT CHARGÉ IMPLICITEMENT

Cette tâche décrit comment configurer votre application pour en exclure une liste de dépendances de module.

Vous pouvez configurer une application déployable pour empêcher le chargement des dépendances implicites. Cela est commun quand il s'agit d'une version différente de bibliothèque ou d'un framework autre que celui qui sera fourni par le serveur d'applications comme dépendance implicite.

Conditions préalables

1. Vous devez déjà avoir un projet informatique qui fonctionne et dont vous souhaitez exclure une dépendance.
2. Vous devez connaître le nom du module à exclure. Voir Section 3.8.1, « Dépendances de modules implicites » pour obtenir une liste des dépendances implicites et de leurs conditions.

Procédure 3.4. Ajouter une configuration d'exclusion de dépendances de jboss-deployment-structure.xml

1. Si l'application n'a pas de fichier **`jboss-deployment-structure.xml`**, créer un nouveau fichier intitulé **`jboss-deployment-structure.xml`** et ajoutez-le au projet. Ce fichier est un fichier XML ayant comme élément root **`<jboss-deployment-structure>`**.

   ```
   <jboss-deployment-structure>

   </jboss-deployment-structure>
   ```

   Pour une application web (WAR), ajouter ce fichier au répertoire **`WEB-INF`**. Pour une archive EJB (JAR), l'ajouter au répertoire **`META-INF`**.

2. Créer un élément **`<deployment>`** dans la racine de l'élément et un **`<exclusions>`** à l'intérieur.

   ```
   <deployment>
      <exclusions>

      </exclusions>
   </deployment>
   ```

3. Dans l'élément d'exclusion, ajouter un élément **`<module>`** pour chaque module à exclure. Définir l'attribut **`name`** au nom du module.

   ```
   <module name="org.javassist" />
   ```

Exemple 3.5. Exclusion de deux modules

```
<jboss-deployment-structure>
   <deployment>
      <exclusions>
```

```
            <module name="org.javassist" />
            <module name="org.dom4j" />
        </exclusions>
    </deployment>
</jboss-deployment-structure>
```

Rapporter un bogue

3.5. EXCLURE UN SOUS-SYSTÈME D'UN DÉPLOIEMENT

Résumé

Ce sujet traite des étapes nécessaires pour exclure un sous-système d'un déploiement. Pour ce faire, il convient de modifier le fichier de configuration **`jboss-deployment-structure.xml`**. Le fait d'exclure un sous-système crée le même effet que de le retirer, sauf qu'il ne s'applique qu'à un seul déploiement.

Procédure 3.5. Exclure un sous-système

1. Ouvrir le fichier **`jboss-deployment-structure.xml`** dans un éditeur de texte.
2. Ajouter le XML suivant dans les balises de <deployment> :

```
<exclude-subsystems>
  <subsystem name="SUBSYSTEM_NAME" />
</exclude-subsystems>
```

3. Enregistrer le fichier **`jboss-deployment-structure.xml`**

Résultat

Le sous-système a bien été exclus. Les processeurs d'unité de déploiement du sous-système n'exécuteront plus sur le déploiement.

Exemple 3.6. Exemple d'un fichier `jboss-deployment-structure.xml`.

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-structure:1.2">
  <ear-subdeployments-isolated>true</ear-subdeployments-isolated>
  <deployment>
    <exclude-subsystems>
      <subsystem name="resteasy" />
    </exclude-subsystems>
    <exclusions>
      <module name="org.javassist" />
    </exclusions>
    <dependencies>
      <module name="deployment.javassist.proxy" />
      <module name="deployment.myjavassist" />
      <module name="myservicemodule" services="import"/>
    </dependencies>
    <resources>
      <resource-root path="my-library.jar" />
    </resources>
  </deployment>
  <sub-deployment name="myapp.war">
```

```
    <dependencies>
      <module name="deployment.myear.ear.myejbjar.jar" />
    </dependencies>
    <local-last value="true" />
  </sub-deployment>
  <module name="deployment.myjavassist" >
    <resources>
     <resource-root path="javassist.jar" >
       <filter>
         <exclude path="javassist/util/proxy" />
       </filter>
     </resource-root>
    </resources>
  </module>
  <module name="deployment.javassist.proxy" >
    <dependencies>
      <module name="org.javassist" >
        <imports>
          <include path="javassist/util/proxy" />
          <exclude path="/**" />
        </imports>
      </module>
    </dependencies>
  </module>
</jboss-deployment-structure>
```

Rapporter un bogue

3.6. UTILISER LE CHARGEUR DE CLASSES PAR PROGRAMMATION DANS UN DÉVELOPPEMENT

3.6.1. Chargement des classes et des ressources par programmation dans le déploiement.

Vous pouvez trouver ou charger des classes ou des ressources par programmation dans votre code d'application. La méthode choisie dépendra d'un certain nombre de facteurs. Cette section décrit les méthodes disponibles et vous donne des instructions sur la façon de les utiliser.

Charger une classe en utilisant la méthode Class.forName()

Vous pouvez utiliser la méthode **`Class.forName()`** pour charger ou initialiser les classes par programmation. Cette méthode comprend deux signatures.

Class.forName(String className)

Cette signature ne prend en compte qu'un seul paramètre, le nom de la classe que vous devez charger. Avec cette méthode de signature, la classe est chargée par le chargeur de classes et initialise la classe nouvellement chargée par défaut.

Class.forName(String className, boolean initialize, ClassLoader loader)

Cette signature s'attend à recevoir trois paramètres : le nom de la classe, une valeur booléenne qui indique si on doit initialiser la classe, et le ClassLoader qui doit charger la classe.

La signature à trois arguments est la méthode recommandée pour charger une classe par programmation. Cette signature vous permet de contrôler si vous souhaitez que la classe cible soit initialisée pendant le chargement. Il est également plus efficace d'obtenir et de fournir le chargeur de classe parce que la JVM n'a pas besoin d'examiner la pile des appels pour déterminer quel chargeur de classe utiliser. En supposant que la classe contenant le code se nomme **`CurrentClass`**, vous pourrez obtenir le chargeur de classes de la classe par la méthode **`CurrentClass.class.getClassLoader()`**.

L'exemple suivant donne un exemple de chargeur de classes qui puisse charger et initialiser la classe **`TargetClass`** :

Exemple 3.7. Fournir un chargeur de classes pour télécharger et initialiser les TargetClass.

```
Class<?> targetClass = Class.forName("com.myorg.util.TargetClass",
true, CurrentClass.class.getClassLoader());
```

Chercher toutes les ressources avec un Prénom

Si vous connaissez le nom ou le chemin de la ressource, la meilleure façon de la charger directement serait d'utiliser la classe JDK ou l'API Classloader.

Charger une seule ressource

Pour charger une seule ressource qui se trouve dans le même répertoire que votre classe ou dans une autre classe de votre déploiement, vous pouvez utiliser la méthode **`Class.getResourceAsStream()`**.

Exemple 3.8. Charger une seule ressource dans votre déploiement.

```
InputStream inputStream =
CurrentClass.class.getResourceAsStream("targetResourceName");
```

Charger toutes les instances en une ressource unique

Pour charger toutes les instances d'une même ressource qui sont visibles par le chargeur de classe de votre déploiement, utilisez la méthode **`Class.getClassLoader retourne ().getResources(String resourceName)`**, où **`resourceName`** est le chemin d'accès qualifié complet de la ressource. Cette méthode retourne une énumération de tous les objets **`URL`** pour les ressources accessibles par le chargeur de classe du prénom donné. Vous pouvez ensuite parcourir le tableau des URL pour ouvrir chaque flux à l'aide de la méthode **`openStream()`**.

Exemple 3.9. Charger toutes les instances d'une ressource et itérer le résultat.

```
Enumeration<URL> urls =
CurrentClass.class.getClassLoader().getResources("full/path/to/reso
urce");
while (urls.hasMoreElements()) {
    URL url = urls.nextElement();
    InputStream inputStream = null;
    try {
        inputStream = url.openStream();
        // Process the inputStream
        ...
    } catch(IOException ioException) {
        // Handle the error
```

```
        } finally {
            if (inputStream != null) {
                try {
                    inputStream.close();
                } catch (Exception e) {
                    // ignore
                }
            }
        }
    }
```

NOTE

Comme les instances URL sont chargées à partir du stockage local, il n'est pas utile d'utiliser **`openConnection()`** ou des méthodes semblables. Les flux sont plus faciles à utiliser et minimisent la complexité du code.

Charger un fichier de classe à partir d'un chargeur de classes

Si une classe a déjà été chargée, vous pouvez charger le fichier de classe qui correspond à cette classe en utilisant la syntaxe suivante :

Exemple 3.10. Charger un fichier de classe déjà chargée.

```
InputStream inputStream =
CurrentClass.class.getResourceAsStream(TargetClass.class.getSimpleNam
e() + ".class");
```

Si la classe n'a pas encore été chargée, vous devrez utiliser le chargeur de classes et en interpréter le chemin d'accès :

Exemple 3.11. Charger un fichier de classe d'une classe qui n'a pas été chargée.

```
String className = "com.myorg.util.TargetClass"
InputStream inputStream =
CurrentClass.class.getClassLoader().getResourceAsStream(className.rep
lace('.', '/') + ".class");
```

Rapporter un bogue

3.6.2. Itérer les ressources par programmation dans un déploiement.

La bibliothèque JBoss Modules fournit plusieurs API pour itérer toutes les ressources de déploiement. La JavaDoc de l'API de JBoss Modules se trouve à cet endroit : http://docs.jboss.org/jbossmodules/1.3.0.Final/api/. Pour utiliser des API, vous devez ajouter la dépendance suivante à **`MANIFEST.MF`** :

```
Dépendences: org.jboss.modules
```

Il est important de noter que malgré que ces API fournissent une flexibilité grandissante, ils exécuteront bien plus lentement qu'une recherche de chemin d'accès directe.

Cette section décrit un nombre de façons dont vous pouvez itérer les ressources de votre code d'application par programmation.

Lister les ressources du déploiement et de toutes les importations.

Il y a des moments où il n'est pas possible de rechercher des ressources par le chemin exact. Par exemple, le chemin d'accès exact n'est peut être pas connu ou vous devrez peut-être examiner plus d'un fichier dans un chemin d'accès donné. Dans ce cas, la bibliothèque de modules de JBoss fournit plusieurs API pour une itération de toutes les ressources de déploiement. Vous pouvez parcourir les ressources dans un déploiement en utilisant une des deux méthodes.

Itérer toutes les ressources qui se trouvent dans un seul module

La méthode **`ModuleClassLoader.iterateResources()`** itère toutes les ressources au sein de ce chargeur de classes de module. Cette méthode accepte deux arguments : le nom du répertoire de départ à rechercher et une valeur booléenne qui spécifie s'il doit agir récursivement dans les sous-répertoires.

L'exemple suivant montre comment on obtient le ModuleClassLoader ou l'itérateur pour les ressources qui se trouvent dans le répertoire **`bin/`**, récursivement dans les sous-répertoires.

Exemple 3.12. Chercher les ressources du répertoire "bin" récursivement dans les sous-répertoires.

```
ModuleClassLoader moduleClassLoader = (ModuleClassLoader)
TargetClass.class.getClassLoader();
Iterator<Resource> mclResources =
moduleClassLoader.iterateResources("bin",true);
```

L'itérateur résultant peut être utilisé pour examiner chaque ressource correspondante et pour chercher son nom et sa taille (si disponible), pour ouvrir une transmission à lire, ou pour obtenir l'URL d'une ressource.

Itérer toutes les ressources qui se trouvent dans un seul module et dans les ressources importées

La méthode **`Module.iterateResources()`** itère toutes les ressources au sein de ce chargeur de classes de module, y compris les ressources qui sont importées dans le module. Cette méthode renvoie un ensemble beaucoup plus vaste que la méthode précédente. Cette méthode requiert un argument: un filtre qui rétrécit le résultat à un modèle spécifique. Alternativement, les PathFilters.acceptAll() peuvent être fournis pour retourner l'ensemble.

Exemple 3.13. Chercher toutes les ressources de ce module, y compris les importations.

```
ModuleClassLoader moduleClassLoader = (ModuleClassLoader)
TargetClass.class.getClassLoader();
Module module = moduleClassLoader.getModule();
Iterator<Resource> moduleResources =
module.iterateResources(PathFilters.acceptAll());
```

Chercher toutes les ressources qui correspondent à un modèle particulier

Si vous avez besoin de trouver des ressources spécifiques dans votre déploiement ou dans son importation complète, vous devrez filtrer l'itération de la ressource. Les API de filtrage de JBoss Modules peuvent vous donner plusieurs outils pour y parvenir.

Examiner toutes les dépendances

Si vous devez examiner l'ensemble des dépendances, vous pouvez utiliser le paramètre **`PathFilter`** de la méthode **`Module.iterateResources()`** pour vérifier le nom de chaque ressource et trouver une correspondance.

Examiner les dépendances de déploiement

Si vous devez chercher uniquement dans le déploiement, utilisez la méthode **`ModuleClassLoader.iterateResources()`**. Toutefois, vous devez utiliser des méthodes supplémentaires pour filtrer l'itérateur qui en résulte. La méthode **`PathFilters.filtered()`** peut fournir une vue filtrée d'un itérateur de ressources dans ce cas. La classe **`PathFilters`** comprend plusieurs méthodes statiques pour créer et composer des filtres qui remplissent des fonctions diverses, y compris la recherche de chemins dépendants ou des correspondances exactes, ou encore correspondant à un motif "glob" Ant-style.

Exemples de codes supplémentaires pour filtrer les ressources

Les exemples suivants démontrent comment filtrer les ressources sur la base de critères différents.

Exemple 3.14. Trouver tous les noms qui comprennent "messages.properties" dans votre déploiement.

```
ModuleClassLoader moduleClassLoader = (ModuleClassLoader)
TargetClass.class.getClassLoader();
Iterator<Resource> mclResources =
PathFilters.filtered(PathFilters.match("**/messages.properties"),
moduleClassLoader.iterateResources("", true));
```

Exemple 3.15. Trouver tous les noms qui comprennent "messages.properties" dans votre déploiement et dans vos importations.

```
ModuleClassLoader moduleClassLoader = (ModuleClassLoader)
TargetClass.class.getClassLoader();
Module module = moduleClassLoader.getModule();
Iterator<Resource> moduleResources =
module.iterateResources(PathFilters.match("**/message.properties));
```

Exemple 3.16. Trouver tous les fichiers qui se trouvent dans tout répertoire nommé "my-resources" de votre déploiement.

```
ModuleClassLoader moduleClassLoader = (ModuleClassLoader)
TargetClass.class.getClassLoader();
Iterator<Resource> mclResources =
PathFilters.filtered(PathFilters.match("**/my-resources/**"),
moduleClassLoader.iterateResources("", true));
```

Exemple 3.17. Trouver tous les fichiers nommés "messages.properties" dans votre déploiement et dans vos importations.

```
ModuleClassLoader moduleClassLoader = (ModuleClassLoader)
TargetClass.class.getClassLoader();
Module module = moduleClassLoader.getModule();
Iterator<Resource> moduleResources =
module.iterateResources(PathFilters.any(PathFilters.match("**/messages
"), PathFilters.match("**/errors"));
```

Exemple 3.18. Trouver tous les fichiers qui se trouvent dans un package spécifique de votre déploiement.

```
ModuleClassLoader moduleClassLoader = (ModuleClassLoader)
TargetClass.class.getClassLoader();
Iterator<Resource> mclResources =
moduleClassLoader.iterateResources("path/form/of/packagename",
false);
```

Rapporter un bogue

3.7. CHARGEMENT DE CLASSE ET SOUS-DÉPLOIEMENTS

3.7.1. Chargement de classes et de modules dans les EAR (Archives Enterprise)

Les EAR (Archives Enterprise) ne sont pas chargées sous forme d'un simple module à la manière des déploiements JAR ou WAR. Ils sont chargés sous forme d'une multitude de modules uniques.

Les règles suivantes déterminent quels modules existent dans un EAR.

- Chaque sous-déploiement WAR ou EJB JAR est un module.
- Le contenu du répertoire **`lib/`** de la racine de l'archive EAR est un module. On l'appelle un module parent.

Ces modules possèdent le même comportement que tout autre module ayant les dépendances implicites suivantes :

- Les sous-déploiements WAR possédent des dépendances implicites sur le module parent et sur tout sous-déploiement JAR EJB.
- Les sous-déploiements EJB JAR possédent des dépendances implicites sur le module parent et sur tout sous-déploiement JAR EJB.

IMPORTANT

Aucun sous-déploiement ne gagne une dépendance implicite sur un sous-déploiement WAR. Tout sous-déploiement peut être configuré avec des dépendances explicites sur un autre sous-déploiement comme dans n'importe quel autre module.

Les dépendances implicites décrites ci-dessus ont lieu car JBoss EAP 6 a l'isolement de chargeur de classe de sous-déploiement désactivé par défaut.

Tout isolement de chargeur de classe de sous-déploiement peut être activé si une stricte compatibilité est nécessaire. Ceci peut être activé pour un simple déploiement EAR ou pour tous les déploiements EAR. La spécification Java EE 6 recommande que les applications portables ne reposent pas sur des sous-déploiements pouvant accéder les uns aux autres, à moins que les dépendances soient explicitement déclarées comme entrées **`Class-Path`** dans le fichier **`MANIFEST.MF`** de chaque sous-déploiement.

Rapporter un bogue

3.7.2. Isolement du chargeur de classes d'un sous-déploiement

Chaque sous-déploiement d'EAR (Archive Enterprise) est un module dynamique possédant son propre chargeur de classe. Par défaut, un sous-déploiement peut accéder aux ressources d'autres sous-déploiements.

Si un sous-déploiement ne doit pas accéder aux ressources d'autres sous-déploiements (une isolation de sous-déploiement est alors requise), alors cela pourra être activé.

Rapporter un bogue

3.7.3. Désactiver l'isolement du chargeur de classes d'un sous-déploiement dans un EAR

Cette tâche vous montre comment faire pour désactiver l'isolement du chargeur de classes de sous-déploiement dans un déploiement EAR, à l'aide d'un descripteur de déploiement spécial EAR. Cela ne nécessite des modifications au serveur d'applications et n'affecte pas les autres déploiements.

IMPORTANT

Même quand l'isolement d'un chargeur de classes de sous-déploiement est désactivé, il est possible d'ajouter un déploiement WAR comme dépendance.

1. **Ajouter le fichier du descripteur de déploiement**
   Ajouter le fichier de descripteur de déploiement **`jboss-deployment-structure.xml`** au répertoire **`META-INF`** du EAR s'il n'existe pas déjà et ajouter le contenu suivant :

   ```
   <jboss-deployment-structure>

   </jboss-deployment-structure>
   ```

2. **Ajouter l'élément `<ear-subdeployments-isolated>`**
   Ajouter l'élément **`<ear-subdeployments-isolated>`** au fichier **`jboss-deployment-structure.xml`** s'il n'existe pas déjà dans le contenu de **`false`**.

   ```
   <ear-subdeployments-isolated>false</ear-subdeployments-isolated>
   ```

Résultat :

L'isolement de chargeur de classe de sous-déploiement sera maintenant désactivé pour le déploiement de cet EAR. Cela signifie que les sous-déploiements du EAR auront des dépendances automatiques sur chacun des sous-déploiements non-WAR.

Rapporter un bogue

3.8. RÉFÉRENCE

3.8.1. Dépendances de modules implicites

Le tableau suivant liste les modules qui sont ajoutés automatiquement au déploiement en tant que dépendances et les conditions qui déclenchent la dépendance.

Tableau 3.1. Dépendances de modules implicites

Sous-système responsable de l'ajout de la dépendance	Dépendances qui sont toujours ajoutées	Dépendances qui sont ajoutées conditionnellement	Conditions qui déclenchent l'ajout de dépendances
Serveur principal	• javax.api • sun.jdk • org.jboss.vfs		
EE subsystem	• javaee.api		
EJB 3 subsystem		• javaee.api	La présence d'un ficher **`ejb-jar.xml`** à des emplacements valides dans le déploiement, tel que mentionné dans les spécifications Java EE 6. La présence d'EJB basés-annotations, comme : **`@Stateless`**, **`@Stateful`**, **`@MessageDriven`**

Sous-système responsable de l'ajout de la dépendance	Dépendances qui sont toujours ajoutées	Dépendances qui sont ajoutées conditionnellement	Conditions qui déclenchent l'ajout de dépendances
Sous-système JAX-RS (RESTEasy)	• javax.xml.bind.api	• org.jboss.resteasy.resteasy-atom-provider • org.jboss.resteasy.resteasy-cdi • org.jboss.resteasy.resteasy-jaxrs • org.jboss.resteasy.resteasy-jaxb-provider • org.jboss.resteasy.resteasy-jackson-provider • org.jboss.resteasy.resteasy-jsapi • org.jboss.resteasy.resteasy-multipart-provider • org.jboss.resteasy.async-http-servlet-30	La présence d'annotations JAX-RS dans le déploiement.
Sous-système JCA	• javax.resource.api	• javax.jms.api • javax.validation.api • org.jboss.logging • org.jboss.ironjacamar.api • org.jboss.ironjacamar.impl • org.hibernate.validator	Le déploiement d'une archive d'adaptateur de ressources (RAR).

Sous-système responsable de l'ajout de la dépendance	Dépendances qui sont toujours ajoutées	Dépendances qui sont ajoutées conditionnellement	Conditions qui déclenchent l'ajout de dépendances
Sous-système JPA (Hibernate)	• javax.persistence.api	• javaee.api • org.jboss.as.jpa • org.hibernate	La présence de l'annotation **`@PersistenceUnit`** ou **`@PersistenceContext`**, **`<persistence-unit-ref>`** ou encore d'un élément **`<persistence-context-ref>`** dans un descripteur de déploiement. JBoss EAP 6 mappe les noms du fournisseur de persistance aux noms du module. Si vous nommez un fournisseur particulier dans le fichier **`persistence.xml`**, une dépendance sera ajoutée au module qui convient. Si ce comportement ne convient pas, vous pourrez l'exclure par l'intermdiaire d'un fichier **`jboss-deployment-structure.xml`**.
Sous-système de journalisation	• org.jboss.logging • org.apache.log4j • org.apache.commons.logging • org.slf4j • org.jboss.logging.jul-to-slf4j-stub		Ces dépendances sont toujours ajoutées à moins que l'attribut **`add-logging-api-dependencies`** soit défini à false.
Sous-système SAR		• org.jboss.logging • org.jboss.modules	Le déploiement d'une archive SAR.
Sous-système de sécurité	• org.picketbox		

Sous-système responsable de l'ajout de la dépendance	Dépendances qui sont toujours ajoutées	Dépendances qui sont ajoutées conditionnellement	Conditions qui déclenchent l'ajout de dépendances
Sous-système Web		• javaee.api • com.sun.jsf-impl • org.hibernate.validator • org.jboss.as.web • org.jboss.logging	Le déploiement d'une archive WAR. JavaServer Faces (JSF) est ajouté uniquement si utilisé.
Sous-système de Services Web	• org.jboss.ws.api • org.jboss.ws.spi		
Sous-système Weld (CDI)		• javax.persistence.api • javaee.api • org.javassist • org.jboss.interceptor • org.jboss.as.weld • org.jboss.logging • org.jboss.weld.core • org.jboss.weld.api • org.jboss.weld.spi	La présence d'un fichier **`beans.xml`** dans le déploiement.

Rapporter un bogue

3.8.2. Les modules inclus

Un tableau énumérant les modules JBoss EAP 6 inclus et indiquant s'ils sont pris en charge se trouve dans le portail clients à https://access.redhat.com/articles/1122333.

Rapporter un bogue

3.8.3. Référence de descripteur de déploiement de structure de déploiement de JBoss

Les tâches principales qui peuvent être effectuées par ce descripteur de déploiement sont les suivantes :

- Définition des dépendances de modules explicites.
- Empêcher les dépendances implicites spécifiques de charger.
- Définir des modules supplémentaires à partir des ressources de ce déploiement.
- Modifier le comportement de l'isolation du sous-déploiement dans ce déploiement EAR.
- Ajouter des racines de ressources supplémentaires à un EAR.

Rapporter un bogue

CHAPITRE 4. VALVES

4.1. VALVES

Une valve est une classe Java insérée dans le pipeline de traitement de demande d'une application. Elle est insérée dans le pipeline avant les filtres servlet. Les valves peuvent apporter des modifications à la demande avant de les passer ou d'effectuer tout autre traitement comme l'authentification ou annuler la demande.

Les valves peuvent être configurées au niveau du serveur ou au niveau de l'application. La seule différence est la façon dont elles sont configurées ou empaquetées.

- Les valves globales sont configurées au niveau serveur et s'appliquent à toutes les applications déployées dans le serveur. Les instructions sur la façon de configurer les valves globales se trouvent dans le guide *Administration and Configuration Guide* de JBoss EAP.
- Les valves configurées au niveau de l'application sont empaquetées dans le déploiement de l'application et n'affectent que l'application en question. Des instructions sur la façon de configurer des valves au niveau de l'application se trouvent dans le guide *Development Guide* de JBoss EAP.

Les versions 6.1.0 et supérieures prennent en charge les valves globales.

Rapporter un bogue

4.2. VALVES GLOBALES

Une valve globale est une valve insérée dans le pipeline de traitement de requête de toutes les applications déployées. Une valve est rendue globale lorsqu'elle est mise en paquetage et qu'elle est installée comme module static dans JBoss EAP 6. Les valves globales sont configurées dans le sous-système web.

Seules les versions 6.1.0 et supérieures prennent en charge les valves globales.

Pour obtenir des instructions sur la façon de configurer les valves globales, voir le chapitre qui s'intitule *Global Valves* dans le guide *Administration and Configuration Guide for JBoss EAP*.

Rapporter un bogue

4.3. LES VALVES D'AUTHENTIFICATION

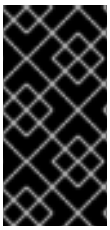
Une valve d'authentification est une valve qui authentifie les informations d'identification d'une requête. Cette valve est une sous-classe de **`org.apache.catalina.authenticator.AuthenticatorBase`** et elle remplace la méthode **`authenticate(Request request, Response response, LoginConfig config)`**.

Elle peut être utilisée pour implémenter des schémas d'authentification supplémentaires.

Rapporter un bogue

4.4. CONFIGURER UNE APPLICATION WEB POUR UTILISER UNE VALVE.

Les valves qui ne sont pas installées en tant que valves globales doivent être incluses avec votre application et configurées dans le descripteur de déploiement **jboss-web.xml**.



IMPORTANT

Les valves globales sont appliquées automatiquement à toutes les applications qui sont déployées. Pour obtenir des instructions sur la façon de configurer les valves globales, voir le chapitre qui s'intitule *Global Valves* dans le guide *Administration and Configuration Guide for JBoss EAP*.

Conditions préalables

- La valve doit être créée et incluse dans le chemin de classe de l'application. Pour ce faire, il convient de l'inclure dans le fichier WAR de l'application ou dans tout autre module ajouté comme dépendance. Les exemples de tels modules comprennent un module statique installé sur le serveur ou un fichier JAR dans le répertoire **lib/** d'une archive EAR si le WAR est déployé dans un EAR.
- L'application doit inclure un descripteur de déploiement de **jboss-web.xml**.

Procédure 4.1. Configurer une application pour une valve locale.

1. **Configuration d'une valve**
   Créer un élément **valve** contenant l'élément enfant **class-name** au fichier **jboss-web.xml** de l'application. Le **class-name** est le nom de la classe de valve.

```
<valve>
      <class-name>VALVE_CLASS_NAME</class-name>
</valve>
```

Exemple 4.1. Exemple d'élément de valve configuré dans le fichier jboss-web.xml

```
<valve>
      <class-
name>org.jboss.security.negotiation.NegotiationAuthenticator</clas
s-name>
</valve>
```

2. **Configurer une valve personnalisée**
   Si la valve possède des paramètres configurables, ajouter un élément enfant **param** à l'élément **valve** pour chaque paramètre, en spécifiant leur nom et leur valeur**param-name** et **param-value**.

Exemple 4.2. Exemple d'élément de valve personnalisé configuré dans le fichier jboss-web.xml

```
<valve>
    <class-
name>org.jboss.web.tomcat.security.GenericHeaderAuthenticator</cla
ss-name>
    <param>
        <param-name>httpHeaderForSSOAuth</param-name>
```

```
        <param-value>sm_ssoid,ct-remote-
user,HTTP_OBLIX_UID</param-value>
    </param>
    <param>
        <param-name>sessionCookieForSSOAuth</param-name>
        <param-value>SMSESSION,CTSESSION,ObSSOCookie</param-
value>
    </param>
</valve>
```

Une fois l'application déployée, la valve sera autorisée pour cette application avec la configuration spécifiée.

Exemple 4.3. Configuration valve jboss-web.xml

```
<valve>
    <class-
name>org.jboss.samplevalves.RestrictedUserAgentsValve</class-name>
    <param>
        <param-name>restrictedUserAgents</param-name>
        <param-value>^.*MS Web Services Client Protocol.*$</param-
value>
    </param>
 </valve>
```

Rapporter un bogue

4.5. CONFIGURER UNE APPLICATION WEB POUR QU'ELLE UTILISE UNE VALVE D'AUTHENTIFICATION

Afin de configurer une application pour utiliser une valve d'authentification, il convient d'installer et de configurer la valve (locale à l'application ou en tant que valve globale) et de configurer le descripteur de déploiement **`web.xml`** de l'application. Dans le cas le plus simple, la configuration **`web.xml`** revient à utiliser une authentification **`BASIC`** sauf que l'élément enfant **`auth-method`** de **`login-config`** est défini sous le nom de la valve qui effectue la configuration.

Conditions préalables

- La valve d'authentification devrait déjà être créée.
- Si la valve d'authentification est une valve globale, cette dernière devrait déjà être installée et configurée, et il convient de connaître le nom sous lequel elle a été configurée.
- Informez-vous du nom de domaine de la zone de sécurité que l'application utilise.

Si vous ne savez pas quel nom de domaine de la zone de sécurité ou la valve à utiliser, informez-vous auprès de l'administrateur de votre serveur.

Procédure 4.2. Configurer une application pour utiliser une valve d'authentification

1. **Configurer la valve**

L'utilisation d'une valve locale nécessite une configuration dans le descripteur de déploiement **`jboss-web.xml`** des applications. Voir Section 4.4, « Configurer une application web pour utiliser une valve. ».

Cela n'est pas nécessaire pour l'utilisation d'une valve globale.

2. **Ajouter une configuration de sécurité à web.xml.**
Ajouter la configuration de sécurité au fichier web.xml pour votre application en utilisant les éléments standards tels que la contrainte de sécurité, la configuration de l'identifiant et le rôle de la sécurité. Dans l'élément de configuration de l'identifiant, définir la valeur de la méthode d'identification en tant que nom de la valve d'identification. L'élément de nom de la zone doit également être défini en tant que nom de la zone de sécurité de JBoss utilisée par l'application.

```
<login-config>
   <auth-method>VALVE_NAME</auth-method>
   <realm-name>REALM_NAME</realm-name>
</login-config>
```

Quand l'application est déployée, l'authentification des requêtes est gérée par la valve d'authentification configurée.

Rapporter un bogue

4.6. CRÉER UNE VALVE PERSONNALISÉE

Une Valve est une classe Java qui est insérée dans le pipeline de traitement de requête d'une application avant les filtres de servlet de l'application. Elle peut être utilisée pour modifier la demande ou pour un autre comportement. Cette tâche vous montre les étapes de base requises pour créer une valve.

Procédure 4.3. Créer une valve personnalisée

1. **Créer la classe Valve**
Créer une sous-classe de **`org.apache.catalina.valves.ValveBase`**.

```
package org.jboss.samplevalves;

import org.apache.catalina.valves.ValveBase;
import org.apache.catalina.connector.Request;
import org.apache.catalina.connector.Response;

public class RestrictedUserAgentsValve extends ValveBase {

}
```

2. **Mettre en place la méthode d'invocation**
La méthode **`invoke()`** est utilisée lorsque la valve est exécutée dans la pipeline. Les objets de requête et de réponse sont considérés comme des paramètres. Effectuer tout procédé et modification de requête et réponse ici.

```
public void invoke(Request request, Response response)
{

}
```

3. **Invoquer la prochaine étape de pipeline**
   La dernière étape que la méthode d'invocation doit effectuer est d'invoquer l'étape suivante de la pipeline et de transmettre les objets de requête et de réponse modifiés, en utilisant la méthode **`getNext().invoke()`**.

```
getNext().invoke(request, response);
```

4. **Option : indiquer les paramètres**
   Si la valve doit être configurable, activer cette option en ajoutant un paramètre. Pour ce faire, il convient d'ajouter une variable d'instance et une méthode setter pour chaque paramètre.

```
private String restrictedUserAgents = null;

public void setRestricteduserAgents(String mystring)
{
   this.restrictedUserAgents = mystring;
}
```

Exemple 4.4. Échantillon de valve personnalisée

```
package org.jboss.samplevalves;

import java.io.IOException;
import java.util.regex.Pattern;

import javax.servlet.ServletException;
import org.apache.catalina.valves.ValveBase;
import org.apache.catalina.connector.Request;
import org.apache.catalina.connector.Response;

public class RestrictedUserAgentsValve extends ValveBase
{
    private String restrictedUserAgents = null;

    public void setRestrictedUserAgents(String mystring)
    {
        this.restrictedUserAgents = mystring;
    }

    public void invoke(Request request, Response response) throws
IOException, ServletException
    {
      String agent = request.getHeader("User-Agent");
      System.out.println("user-agent: " + agent + " : " +
restrictedUserAgents);
      if (Pattern.matches(restrictedUserAgents, agent))
      {
         System.out.println("user-agent: " + agent + " matches: " +
restrictedUserAgents);
         response.addHeader("Connection", "close");
      }
      getNext().invoke(request, response);
    }
}
```

Rapporter un bogue

CHAPITRE 5. LA JOURNALISATION POUR LES DÉVELOPPEURS

5.1. INTRODUCTION

5.1.1. Journalisation

La journalisation consiste à l'enregistrement d'une série de messages d'une application dans un fichier-journal des activités de l'application.

Les messages de journalisation fournissent des informations importantes pour les développeurs lors du débogage d'une application et pour les administrateurs système quand au maintien des applications de production.

La plupart des frameworks de journalisation de Java incluent également des informations comme l'heure et l'origine du message.

Rapporter un bogue

5.1.2. Frameworks de journalisations d'applications pris en charge par JBoss LogManager

JBoss LogManager prend en charge les frameworks de journalisation suivants :

- JBoss Logging - inclus avec JBoss EAP 6
- Apache Commons Logging - http://commons.apache.org/logging/
- Simple Logging Facade Java (SLF4J) - http://www.slf4j.org/
- Apache log4j - http://logging.apache.org/log4j/1.2/
- Java SE Logging (java.util.logging) - http://download.oracle.com/javase/6/docs/api/java/util/logging/package-summary.html

Rapporter un bogue

5.1.3. Niveaux de journalisation

Les niveaux de journalisation sont des ensembles ordonnés de valeurs énumérées qui indiquent la nature et la sévérité d'un message de journalisation. Le niveau d'un message de journalisation donné est indiqué par le développeur par des méthodes qui conviennent dans un framework de journalisation particulier pour envoyer le message.

JBoss EAP 6 accepte tous les niveaux de journalisation utilisés par les frameworks de journalisation de l'application prise en charge. Les six niveaux de journalisation les plus utilisés sont (dans l'ordre croissant) : **`TRACE`**, **`DEBOG`**, **`INFO`**, **`ATTENTION`**, **`ERREUR`** et **`FATAL`**.

Les niveaux de journalisation sont utilisés par des catégories et des gestionnaires de journalisation pour limiter les messages dont ils sont responsables. Chaque niveau de journalisation possède une valeur numérique qui indique son ordre par rapport à d'autres niveaux de journalisation. Les catégories et gestionnaires de journalisation correspondent à un certain niveau de journalisation, et ils traitent les

messages de journalisation du même niveau ou d'un niveau supérieur uniquement. Par exemple, un gestionnaire de journalisation du niveau **ATTENTION** enregistrera uniquement les messages des niveaux **ATTENTION**, **ERREUR** et **FATAL**.

Rapporter un bogue

5.1.4. Niveaux de journalisation pris en charge

Tableau 5.1. Niveaux de journalisation pris en charge

Niveau de journalisation	Valeur	Description
FINESSE MAX	300	-
PLUS FIN	400	-
TRACE	400	Utilisé pour des messages qui fournissent des informations détaillées sur l'état d'exécution d'une application. Les messages de journalisation **TRACE** sont habituellement capturés uniquement lors du débogage d'une application.
DEBOG	500	Utilisé pour des messages qui indiquent des demandes individuelles de progrès ou des activités d'une application. Les messages de journalisation **DEBUG** ne sont habituellement capturés que lors du débogage d'une application.
FINESSE	500	-
CONFIG	700	-
INFO	800	Utilisé pour des messages qui indiquent la progression globale de l'application. Souvent utilisé pour le démarrage de l'application, la fermeture et autres événements majeurs de cycle de vie.
AVERTISSEMENT	900	Utilisé pour indiquer une situation qui n'est pas en erreur, mais n'est pas considérée comme idéale. Peut indiquer des circonstances qui peuvent entraîner des erreurs dans le futur.
ATTENTION	900	-
ERREUR	1000	Utiliser pour indiquer une erreur qui s'est produite et qui puisse empêcher l'activité actuelle ou la demande de se remplir, mais qui n'empêchera pas l'application d'exécuter.
SÉVÈRE	1000	-
FATALE	1100	Utiliser pour indiquer les événements qui pourraient entraîner des défaillances de services critiques ou la fermeture de l'application, ou qui pourraient entraîner la fermeture de la plateforme JBoss EAP 6.

Rapporter un bogue

5.1.5. Emplacements de fichiers de journalisation par défaut

Il s'agit des fichiers de journalisation qui ont été créés pour les configurations de journalisation par défaut. La configuration par défaut écrit des fichiers de journalisation du serveur à l'aide de gestionnaires de journaux périodiques.

Tableau 5.2. Fichier de journalisation par défaut d'un serveur autonome

Fichier journal	Description
***EAP_HOME*/standalone/log/server.log**	Journal du serveur. Contient les messages de journalisation de serveur, dont les messages de démarrage de serveur.
***EAP_HOME*/standalone/log/gc.log**	Journalisation de Garbage collection. Contient des informations sur tous les nettoyagess de mémoire.

Tableau 5.3. Fichiers de journalisation par défaut d'un domaine géré

Fichier journal	Description
***EAP_HOME*/domain/log/host-controller.log**	Journal d'amorçage du contrôleur hôte. Contient les messages de journalisation liés au démarrage du contrôleur hôte.
***EAP_HOME*/domain/log/process-controller.log**	Journal d'amorçage du contrôleur de processus. Contient les messages de journalisation liés au démarrage du contrôleur de processus.
***EAP_HOME*/domain/servers/*SERVERNAME*/log/server.log**	Le journal du serveur pour le serveur nommé. Contient les messages de journalisation de ce serveur, dont les messages de démarrage de serveur.

Rapporter un bogue

5.2. JOURNALISATION DANS LE JBOSS LOGGING FRAMEWORK

5.2.1. JBoss Logging

JBoss Logging est le framework de journalisation d'applications inclus dans JBoss EAP 6

JBoss Logging procure un moyen facile d'ajouter une journalisation à une application. Vous ajoutez du code à votre application qui utilise le framework pour envoyer des messages de journalisation dans un format défini. Lorsque l'application est déployée sur un serveur d'applications, ces messages peuvent être capturés par le serveur et affichées ou écrits dans les fichiers selon la configuration du serveur.

Rapporter un bogue

5.2.2. Fonctionnalités de JBoss Logging

- Fournit un enregistreur d'événements de saisie ("typed") facile d'utilisation.

- Support total pour l'internationalisation et la localisation. Les traducteurs travaillent sur des lots de messages dans des fichiers de propriétés, tandis que les développeurs peuvent travailler sur des interfaces et des annotations.
- Outils « buid-time » pour générer les enregistreurs d'événements en saisie pour la production, et la génération runtime d'enregistreurs d'événements pour le développement.

Rapporter un bogue

5.2.3. Ajouter une journalisation à une application par JBoss Logging

Pour journaliser des messages à partir de votre application, créer un objet de créateur de journal (**`org.jboss.logging.Logger`**) et utiliser la méthode appropriée de cet objet. Cette tâche décrit les étapes nécessaires pour ajouter un support pour ceci à votre application.

Conditions préalables

Vous devez remplir les conditions suivantes avant de continuer cette tâche :

- Si vous utilisez Maven en tant que système de build, il convient que le projet soit déjà configuré pour inclure le référentiel JBoss Maven. Voir Section 2.3.2, « Configurer le référentiel JBoss EAP 6 Platform Maven Repository par les paramètres de configuration de Maven »
- Les fichiers JAR de JBoss Logging doivent se trouver dans le chemin de build d'application. La façon dont vous procédrez dépendra de votre décision de générer votre application avec Red Hat JBoss Developer Studio ou avec Maven.
  - Quand vous générez avec Red Hat JBoss Developer Studio vous pouvez procéder en sélectionnant Project -> Properties à partir du menu Red Hat JBoss Developer Studio, en sélectionnant Targeted Runtimes et en veillant bien à vérifier le temps d'exécution de JBoss EAP 6.
  - Quand vous générez avec Maven, vous pouvez procéder en ajoutant la configuration de dépendance suivante au fichier **`pom.xml`** de votre projet.

    ```
    <dependency>
       <groupId>org.jboss.logging</groupId>
       <artifactId>jboss-logging</artifactId>
       <version>3.1.2.GA-redhat-1</version>
       <scope>provided</scope>
    </dependency>
    ```

    Vous n'avez pas besoin d'inclure les JAR de l'application générée car JBoss EAP 6 les fournit aux applications qui sont déployées.

Une fois que votre projet sera correctement installé, vous devrez suivre les étapes suivantes pour chaque classe à laquelle vous souhaitez ajouter une journalisation.

1. **Ajouter Imports**
   Ajouter les déclarations d'importation des espace-noms de classe JBoss Logging que vous allez utiliser. Vous devrez importer au minimum **`import org.jboss.logging.Logger`**.

   ```
   import org.jboss.logging.Logger;
   ```

2. **Créer un objet Logger**

Créer une instance de **`org.jboss.logging.Logger`** et l'initialiser en utilisant la méthode statique **`Logger.getLogger(Class)`**. Red Hat vous recommande de la créer en tant que variable à instance unique pour chaque classe.

```
private static final Logger LOGGER =
Logger.getLogger(HelloWorld.class);
```

3. **Ajouter les messages de journalisation**
   Ajouter des appels aux méthodes de l'objet **`Logger`** à votre code, là où vous souhaitez qu'il envoie des messages. L'objet **`Logger`** comprend un certain nombre de méthodes contenant des paramètres différents suivant les types de messages. Les plus faciles à utiliser sont les suivants :

`debog(message objet)`
`info(message objet)`
`erreur(message objet)`
`trace(message objet)`
`fatal(message objet)`

Ces méthodes envoient un message de journalisation avec un niveau de journalisation correspondant et le paramètre **`message`** comme string.

```
LOGGER.error("Configuration file not found.");
```

Pour obtenir une liste complète des méthodes JBoss Logging, consulter le package **`org.jboss.logging`** dans JBoss EAP 6 API Documentation.

Exemple 5.1. Utilisation de JBoss Logging quand on ouvre un fichier de propriétés

Cet exemple vous montre un exemple de code de classe qui charge une configuration personnalisée pour une application à partir d'un fichier de propriétés. Si le fichier spécifié n'est pas trouvé, un message de journalisation sera enregistré au niveau ERROR.

```
import org.jboss.logging.Logger;
public class LocalSystemConfig
{
   private static final Logger LOGGER =
Logger.getLogger(LocalSystemConfig.class);

   public Properties openCustomProperties(String configname) throws
CustomConfigFileNotFoundException
   {
      Properties props = new Properties();
      try
      {
         LOGGER.info("Loading custom configuration from "+configname);
         props.load(new FileInputStream(configname));
```

```
        }
        catch(IOException e) //catch exception in case properties file
does not exist
        {
           LOGGER.error("Custom configuration file ("+configname+") not
found. Using defaults.");
           throw new CustomConfigFileNotFoundException(configname);
        }

        return props;
    }
```

Rapporter un bogue

5.3. LA JOURNALISATION PAR DÉPLOIEMENT

5.3.1. La journalisation par déploiement

La journalisation par déploiement permet à un développeur de configurer à l'avance la configuration de journalisation de ses applications. Lorsque l'application est déployée, la journalisation commence selon la configuration définie. Les fichiers journaux créés par le biais de cette configuration contiennent uniquement des informations sur le comportement de l'application.

Cette approche a des avantages et des inconvénients pour l'utilisation de la journalisation dans l'ensemble du système. Un des avantages est que l'administrateur de l'instance JBoss EAP n'a pas besoin de configurer la journalisation. Un inconvénient est que la configuration de journalisation par déploiement est en lecture seule au démarrage et ne peut donc pas être modifiée pendant l'exécution.

Rapporter un bogue

5.3.2. La journalisation par déploiement vers une application

Pour configurer la journalisation par dépliement, ajouter le fichier de configuration de journalisation **`logging.properties`** au déploiement. Ce fichier de configuration est conseillé car il peut être utilisé avec n'importe quelle façade de journalisation dans la mesure où JBoss Log Manager est le gestionnaire de journalisation sous-jacent utilisé.

Si vous utilisez **`Simple Logging Facade for Java (SLF4J)`** ou **`Apache log4j`**, le fichier de configuration **`logging.properties`** est approprié. Si vous utilisez appenders de Apache log4j, alors la configuration fichier **`log4j.properties`** sera requise. Le fichier de configuration **`jboss-logging.properties`** est pris en charge uniquement pour les déploiements existants hérités.

Procédure 5.1. Ajouter le fichier de configuration à l'application

- **Le répertoire auquel le fichier de configuration est ajouté dépend de la méthode de déploiement : `EAR`, `WAR` ou `JAR`.**
  - **Déploiement `EAR`**
    Copier le fichier de configuration de journalisation dans le répertoire **`META-INF`**.
  - **Déploiement `WAR` ou `JAR`**
    Copier le fichier de configuration de journalisation dans le répertoire **`META-INF`** ou **`WEB-INF/classes`**.

Rapporter un bogue

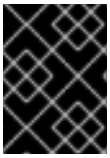
5.3.3. Exemple de fichier logging.properties

```
# Additional loggers to configure (the root logger is always configured)
loggers=
# Root logger configuration
logger.level=INFO
logger.handlers=FILE

# A handler configuration
handler.FILE=org.jboss.logmanager.handlers.FileHandler
handler.FILE.level=ALL
handler.FILE.formatter=PATTERN
handler.FILE.properties=append,autoFlush,enabled,suffix,fileName
handler.FILE.constructorProperties=fileName,append
handler.FILE.append=true
handler.FILE.autoFlush=true
handler.FILE.enabled=true
handler.FILE.fileName=${jboss.server.log.dir}/app.log

# The formatter to use
formatter.PATTERN=org.jboss.logmanager.formatters.PatternFormatter
formatter.PATTERN.properties=pattern
formatter.PATTERN.constructorProperties=pattern
formatter.PATTERN.pattern=%d %-5p %c: %m%n
```

Rapporter un bogue

5.4. PROFILS DE JOURNALISATION

5.4.1. Profils de journalisation



IMPORTANT

Les profils de journalisation ne sont disponibles que dans les versions 6.1.0 ou versions supérieures. Ils ne peuvent pas être configurés par la console de gestion.

Les profils de journalisation sont des ensembles de configuration de journalisation indépendants qui peuvent être assignés à des applications déployées. Un profil de journalisation peut définir les handlers, les catégories et un root logger à la manière du sous-système de journalisation standard, mais ne peut pas recommander la configuration à d'autres profils ou au sous-système de journalisation principal. La conception des profils de journalisation émule le sous-système de journalisation au niveau de l'aisance de configuration.

Les profils de journalisation permettent aux administrateurs de créer des configurations de journalisation spécifiques à une ou à plusieurs applications sans affecter une autre configuration de journalisation. Comme chaque profil est défini dans une configuration serveur, cela implique que la configuration de journalisation peut être modifiée sans exiger que les applications affectées ne soient redéployées.

Chaque profil de journalisation peut avoir la configuration suivante :

- Un nom unique. Ceci est requis.

- N'importe quel nombre de log handlers.
- N'importe quelle catégorie log.
- Un root logger maximum.

Une application peut spécifier un profil de journalisation à utiliser dans son fichier **`MANIFEST.MF`**, en utilisant l'attribut ***`logging-profile`***.

Rapporter un bogue

5.4.2. Spécifier un profil de journalisation dans une application

Une application spécifie le profil de journalisation à utiliser dans son fichier **`MANIFEST.MF`**.

Conditions préalables :

1. Vous devez connaître le nom du profil de journalisation qui a été défini sur le serveur pour cette application. Demandez à votre administrateur de systèmes le nom du profil à utiliser.

Procédure 5.2. Ajouter une configuration de profil de journalisation à une application

- **Modifier `MANIFEST.MF`**
  Si votre application ne possède pas de fichier **`MANIFEST.MF`** : le créer avec le contenu suivant, en remplaçant *NAME* par le nom de profil qui convient.

  ```
  Manifest-Version: 1.0
  Logging-Profile: NAME
  ```

  Si votre application contient déjà un fichier **`MANIFEST.MF`** : ajouter la ligne suivante, en remplaçant *NAME* par le nom du profil qui convient.

  ```
  Logging-Profile: NAME
  ```

NOTE

Si vous utilisez Maven et **`maven-war-plugin`**, vous pourrez mettre votre fichier MANIFEST.MF dans **`src/main/resources/META-INF/`** et ajouter la configuration suivante à votre fichier **`pom.xml`**.

```
<plugin>
  <artifactId>maven-war-plugin</artifactId>
  <configuration>
    <archive>
      <manifestFile>src/main/resources/META-
INF/MANIFEST.MF</manifestFile>
    </archive>
  </configuration>
</plugin>
```

Quand l'application sera déployée, elle utilisera la configuration qui se trouve dans le profil de journalisation spécifié pour ses messages de journalisation.

Rapporter un bogue

CHAPITRE 6. INTERNATIONALISATION ET LOCALISATION

6.1. INTRODUCTION

6.1.1. Internationalisation

Le mot internationalisation fait référence au processus de design des logiciels, pour qu'ils puissent être adaptés aux différentes langues et régions sans changements d'ingénierie.

Rapporter un bogue

6.1.2. Localisation

Le mot localisation fait référence au processus d'adaptation des logiciels internationaux à une région spécifique ou à une langue spécifique, impliquant le rajout de composants particuliers à la locale et à la traduction de textes.

Rapporter un bogue

6.2. OUTILS JBOSS LOGGING

6.2.1. Aperçu

6.2.1.1. JBoss Logging Tools - Internationalisation et Localisation

JBoss Logging Tools est une API Java qui apporte un soutien pour l'internationalisation et la localisation des messages de journalisation, des messages d'exception et chaînes génériques. En plus de fournir un mécanisme de traduction, les outils JBoss Logging fournissent également un support d'identificateur unique pour chaque message de journalisation.

Les messages et les exceptions internationalisés sont créés en tant que définitions de méthode à l'intérieur des interfaces annotées à l'aide des annotations **`org.jboss.logging`**. Il n'est pas nécessaire d'implémenter les interfaces, JBoss Logging Tools le fait au moment de la compilation. Une fois définies, vous pouvez utiliser ces méthodes pour enregistrer des messages ou obtenir des objets d'exception dans votre code.

Les interfaces d'exception et de journalisation internationalisées créées par JBoss Logging Tools peuvent être localisées en créant un fichier de propriétés pour chaque lot contenant les traductions pour un langage et une région spécifiques. JBoss Logging Tools peut générer des fichiers de propriétés de modèle pour chaque lot qui peut ensuite être édité par un traducteur.

JBoss Logging Tools crée une implémentation de chaque lot pour chaque fichier de propriétés de traductions correspondantes dans votre projet. Tous que vous devez faire est d'utiliser les méthodes définies dans les lots et JBoss Logging Tools veillera à ce que l'implémentation qui convient soit invoquée pour vos paramètres régionaux en cours.

Les ids de messages et des codes de projets sont des identificateurs uniques qui sont ajoutés à chaque message du journal. Ces identificateurs uniques peuvent servir de documentation pour faciliter la recherche d'informations sur les messages de journalisation. Avec une documentation suffisante, la signification d'un message de journalisation peut être déterminée par les identificateurs, quelle que soit la langue dans laquelle le message a été rédigé.

Rapporter un bogue

6.2.1.2. JBoss Logging Tools Quickstart

JBoss Logging Tools Quickstart, **`logging-tools`**, contient un simple projet Maven qui démontre les fonctionnalités de JBoss Logging Tools. A été utilisé extensivement dans cette documentation pour les exemples de code.

Voir ce Quickstart pour obtenir une démonstration complète de toutes les fonctionnalités décrites dans cette documentation.

Rapporter un bogue

6.2.1.3. Message Logger

Un Message Logger est une interface utilisée pour définir les messages de journalisation internationalisés. Une interface de Message Logger est annotée par **`@org.jboss.logging.MessageLogger`**.

Rapporter un bogue

6.2.1.4. Lot de messages

Un lot de messages est une interface qui peut être utilisée pour définir des messages génériques à traduire et des objets d'Exception avec des messages internationalisés. Un lot de messages n'est pas utilisé pour créer des messages de journalisation.

Une interface de lot de messages est annotée par **`@org.jboss.logging.MessageBundle`**.

Rapporter un bogue

6.2.1.5. Messages de journalisation internationalisés

Les messages de journalisation internationalisés sont des messages de journalisation créés en définissant une méthode dans un Message Logger. La méthode doit contenir les annotations **`@LogMessage`** et **`@Message`** et spécifier le message de journalisation en utilisant l'attribut de la valeur de **`@Message`**. Les messages de journalisation internationalisés sont localisés en fournissant des traductions dans un fichier de propriétés.

JBoss Logging Tools génère les classes de journalisation requises pour chaque traduction au moment de la compilation et invoque les méthodes qui conviennent à la locale en cours d'exécution.

Rapporter un bogue

6.2.1.6. Exceptions internationalisées

Une exception internationalisée est un objet d'exception retourné par une méthode définie dans un lot de messages. Les méthodes de lots de messages qui retournent des objets d'Exception Java peuvent être annotés pour définir un message d'exception par défaut. Le message par défaut est remplacé par une traduction, si elle existe dans dans un fichier de propriétés correspondant pour le paramètres régionaux en cours. Les exceptions internationalisées peuvent aussi avoir des codes de projet et des ID de messages assignés.

Rapporter un bogue

6.2.1.7. Messages internationalisés

Un message internationalisé est un string retourné par une méthode définie dans un lot de messages. Les méthodes de lots de messages qui retournent des objets String Java peuvent être annotées afin de définir le contenu par défaut de ce string, connu comme message. Le message par défaut est remplacé par une traduction, si une traduction se trouve dans un fichier de propriétés correspondant, pour les paramètres régionaux en cours.

Rapporter un bogue

6.2.1.8. Fichiers de propriétés de traduction

Les fichiers de propriétés de traductions sont des fichiers de propriétés Java qui contiennent les traductions des messages à partir d'une interface pour une variante, locale et pays. Les fichiers de propriétés de traductions sont utilisés par JBoss Logging Tools pour créer des classes qui renvoient les messages.

Rapporter un bogue

6.2.1.9. Codes de projets de JBoss Logging Tools

Les codes de projets sont des chaînes de caractères qui identifient des groupes de messages. Ils se trouvent au début de chaque message de journalisation, et contiennent l'id du message. Les codes de projets sont définis dans l'attribut **`projectCode`** de l'annotation **`@MessageLogger`**.

Rapporter un bogue

6.2.1.10. Ids de messages de JBoss Logging Tools

Les Ids de messages sont des nombres, qui lorsque combinés à un code de projet, identifient de façon unique un message de journalisation. Les Ids de messages sont affichées au début de chaque message du journal, ajoutés au code de projet du message. Les Ids de messages sont définis par l'attribut **`id`** de l'annotation **`@Message`**.

Rapporter un bogue

6.2.2. Création de loggers, de messages ou d'exceptions internationalisés

6.2.2.1. Créer des messages log internationalisés

Cette tâche vous montre comment utiliser JBoss Logging Tools pour créer des messages de journalisation internationalisés en créant des interfaces MessageLogger. Elle ne traite pas de toutes les caractéristiques optionnelles ou de la localisation des messages de journalisation.

Voir le guide de démarrage **`logging-tools`** pour trouver un exemple complet.

Prérequis :

1. Vous devez déjà posséder un projet Maven en cours. Voir Section 6.2.6.1, « Configuration Maven JBoss Logging Tools ».
2. Le projet doit avoir la configuration Maven qui convient pour JBoss Logging Tools.

Procédure 6.1. Créer un lot de messages log internationalisés

1. **Créer une interface de Message Logger**

Ajouter une interface Java à votre projet pour contenir les définitions de messages log. Définir l'interface de façon descriptive pour les messages log qui seront définis à l'intérieur.

L'interface de messages log ont les prérequis suivants :

- Doit être annotée par **`@org.jboss.logging.MessageLogger`**.
- Doit étendre **`org.jboss.logging.BasicLogger`**.
- L'interface doit définir un champ qui est un Logger typed quiimpléemnte cette interface. Procédez par la méthode **`getMessageLogger()`** de **`org.jboss.logging.Logger`**.

```
package com.company.accounts.loggers;

import org.jboss.logging.BasicLogger;
import org.jboss.logging.Logger;
import org.jboss.logging.MessageLogger;

@MessageLogger(projectCode="")
interface AccountsLogger extends BasicLogger
{
   AccountsLogger LOGGER = Logger.getMessageLogger(
         AccountsLogger.class,
         AccountsLogger.class.getPackage().getName() );
}
```

2. **Ajouter les définitions de méthode**
   Ajouter une définition de méthode à l'interface de chaque message log. Nommez chaque méthode descriptivement par rapport au message log qu'elle représente.

   Chaque méthode a les prérequis suivants :

   - La méthode doit renvoyer **`void`**.
   - La méthode doit être annotée par **`@org.jboss.logging.LogMessage`**.
   - La méthode doit être annotée par **`@org.jboss.logging.Message`**.
   - L'attribut de **`@org.jboss.logging.Message`** contient le message log par défaut. Il s'agit du message qui est utilisé s'il n'y a pas de traduction.

   ```
   @LogMessage
   @Message(value = "Customer query failed, Database not available.")
   void customerQueryFailDBClosed();
   ```

   Le niveau de journalisation par défaut est **`INFO`**.

3. **Invoquer les méthodes**
   Ajouter les appels aux méthodes d'interface dans votre code là où les messages doivent être journalisés. Il n'est pas utile de créer des implémentations des interfaces, le processeur d'annotations le fait pour vous quand le projet est compilé.

   ```
   AccountsLogger.LOGGER.customerQueryFailDBClosed();
   ```

Les loggers personnalisés sont des sous-classes de BasicLogger, donc les méthodes de journalisation de **`BasicLogger`** (**`debug()`**, **`error()`** etc) peuvent également être utilisées. Il n'est pas utile de créer d'autres loggers pour enregistrer les messages non internationalisés.

```
AccountsLogger.LOGGER.error("Invalid query syntax.");
```

RÉSULTAT: le projet supporte maintenant un ou plusieurs loggers internationalisés qui peuvent maintenant être localisés.

Rapporter un bogue

6.2.2.2. Créer et utiliser des messages internationalisés

Cette tâche vous montre comment créer des messages internationalisés et comment les utiliser. Cette tâche ne couvre pas toutes les fonctionnalités en option ou la localisation de ces messages de journalisation.

Voir le guide de démarrage **`logging-tools`** pour trouver un exemple complet.

Pré-requis

1. Vous avez un projet Maven utilisant le référentiel JBoss EAP 6. Voir Section 2.3.2, « Configurer le référentiel JBoss EAP 6 Platform Maven Repository par les paramètres de configuration de Maven ».
2. La configuration Maven requise pour JBoss Logging Tools a été ajoutée. Voir Section 6.2.6.1, « Configuration Maven JBoss Logging Tools ».

Procédure 6.2. Créer et utiliser des messages internationalisés

1. **Créer une interface pour les exceptions.**
   JBoss Logging Tools définit les messages internationalisés dans les interfaces. Nommer chaque interface de manière descriptive pour les messages qui seront définis à l'intérieur.

   L'interface possède les prérequis suivants :

   - Elle doit être déclarée publique
   - Elle doit être annotée par **`@org.jboss.logging.MessageBundle`**.
   - L'interface doit définir un champ qui corresponde à un lot de messages du même type que l'interface.

   ```
   @MessageBundle(projectCode="")
   public interface GreetingMessageBundle
   {
      GreetingMessageBundle MESSAGES =
   Messages.getBundle(GreetingMessageBundle.class);
   }
   ```

2. **Ajouter les définitions de méthode**
   Ajouter une définition de méthode dans l'interface de chaque message.

   Chaque méthode a les prérequis suivants :

- Elle doit renvoyer un objet de type **`String`**.
- La méthode doit être annotée par **`@org.jboss.logging.Message`**.
- L'attribut**`@org.jboss.logging.Message`** doit être défini à la valeur par défaut du message. Il s'agit du message qui est utilisé si aucune traduction n'est disponible.

```
@Message(value = "Hello world.")
   String helloworldString();
```

3. **Méthodes d'invocation**
   Invoquer les méthodes d'interface dans votre application quand vous aurez besoin d'obtenir le message.

```
System.console.out.println(helloworldString());
```

RÉSULTAT : le projet prend maintenant en charge des chaînes de messages internationalisés qui peuvent être localisés.

Rapporter un bogue

6.2.2.3. Créer des exceptions personnalisées

Cette tâche vous montre comment créer des exceptions personnalisées et comment les utiliser. Cette tâche ne couvre pas toutes les fonctionnalités en option ou le processus de localisation de ces exceptions.

Voir le guide de démarrage **`logging-tools`** pour trouver un exemple complet.

Pour cette tâche, on assume que vous possédez déjà un projet informatique créé dans le Red Hat JBoss Developer Studio ou Maven, et sur lequel vous souhaitez ajouter des exceptions internationalisées.

Procédure 6.3. Créer et utiliser des exceptions internationalisées

1. **Ajouter une configuration JBoss Logging Tools**
   Ajouter la configuration de projet qui convient pour prendre en charge JBoss Logging Tools. Voir Section 6.2.6.1, « Configuration Maven JBoss Logging Tools »
2. **Créer une interface pour les exceptions.**
   JBoss Logging Tools définit les exceptions internationalisées dans les interfaces. Nommer chaque interface de façon descriptive pour les exceptions qui seront décrites à l'intérieur.

   L'interface possède les prérequis suivants :

   - Elle doit être déclarée **`public`**.
   - Elle doit être annotée par **`@org.jboss.logging.MessageBundle`**.
   - L'interface doit définir un champ qui corresponde à un lot de messages du même type que l'interface.

```
@MessageBundle(projectCode="")
public interface ExceptionBundle
{
```

```
   ExceptionBundle EXCEPTIONS =
Messages.getBundle(ExceptionBundle.class);
}
```

3. **Ajouter les définitions de méthode**
   Ajouter une définition de méthode à l'interface pour chaque exception. Nommez chaque méthode de façon descriptive pour chaque exception qu'elle représente.

   Chaque méthode a les prérequis suivants :

   - Elle doit renvoyer un objet du type **`Exception`** ou bien un sous-type d' **`Exception`**.
   - La méthode doit être annotée par **`@org.jboss.logging.Message`**.
   - L'attribut de **`@org.jboss.logging.Message`** doit être défini à la valeur du message d'exception par défaut. Il s'agit du message qui est utilisé s'il n'y a pas de traduction.
   - Si l'exception est retournée à un constructeur qui nécessite des paramètres en plus d'une chaîne de message, alors ces paramètres doivent être fournis dans la définition de méthode à l'aide de l'annotation **`@param`**. Les paramètres doivent être du même type et du même ordre que le constructeur.

```
@Message(value = "The config file could not be opened.")
IOException configFileAccessError();

@Message(id = 13230, value = "Date string '%s' was invalid.")
ParseException dateWasInvalid(String dateString, @Param int
errorOffset);
```

4. **Méthodes d'invocation**
   Invoquer les méthodes d'interface de votre code quand vous devrez obtenir une des exceptions. Les méthodes ne lancent pas d'exceptions, elles envoient l'objet d'exception que vous pourrez alors envoyer vous-même.

```
try
{
   propsInFile=new File(configname);
   props.load(new FileInputStream(propsInFile));
}
catch(IOException ioex) //in case props file does not exist
{
   throw ExceptionBundle.EXCEPTIONS.configFileAccessError();
}
```

RÉSULTAT : le projet prend maintenant en charge les exceptions internationalisées qui peuvent être localisées.

Rapporter un bogue

6.2.3. Localisation de loggers, messages ou exceptions internationalisés

6.2.3.1. Générer des nouveaux fichiers de propriétés de traduction avec Maven

Des projets qui sont construits avec Maven peuvent générer des fichiers de propriétés de traduction vides pour chaque Message Logger et Lots de messages qu'ils contiennent. Ensuite, ces fichiers peuvent être utilisés comme nouveaux fichiers de propriétés de traduction.

La procédure suivante montre comment configurer un projet Maven pour créer des nouveaux fichiers de propriétés de traduction.

Voir le guide de démarrage **`logging-tools`** pour trouver un exemple complet.

Conditions préalables :

1. Vous devez déjà posséder un projet Maven en cours.
2. Le projet devra avoir été configuré pour JBoss Logging Tools.
3. Le projet doit contenir une ou plusieurs interfaces qui définissent des exceptions ou des messages de journalisation internationalisés

Procédure 6.4. Générer des nouveaux fichiers de propriétés de traduction avec Maven

1. **Ajouter la configuration Maven**
   Ajouter l'argument de compilateur **`-AgenereatedTranslationFilePath`** à la configuration du plug-in du compilateur Maven et lui assigner le chemin d'accès où les nouveaux fichiers devront être créés.

```
<plugin>
   <groupId>org.apache.maven.plugins</groupId>
   <artifactId>maven-compiler-plugin</artifactId>
   <version>2.3.2</version>
   <configuration>
      <source>1.6</source>
      <target>1.6</target>
      <compilerArgument>
      -
AgeneratedTranslationFilesPath=${project.basedir}/target/generated-
translation-files
      </compilerArgument>
      <showDeprecation>true</showDeprecation>
   </configuration>
</plugin>
```

   La configuration ci-dessus va créer des nouveaux fichiers dans le répertoire **`target/generated-translation-files`** de votre projet Maven.

2. **Créer votre projet**
   Créer votre projet en utilisant Maven.

```
[Localhost]$ mvn compile
```

Un fichier de propriétés est créé par interface annotée par **`@MessageBundle`** ou **`@MessageLogger`**. Les nouveaux fichiers sont créés dans un sous-répertoire qui correspond au package Java dans lequel chaque interface est déclarée.

Chaque nouveau fichier est nommé par la syntaxe suivante, avec **`InterfaceName`** comme nom d'interface pour laquelle ce fichier a été créé : ***`InterfaceName`*`.i18n_locale_COUNTRY_VARIANT.properties`**.

Ces fichiers peuvent alors être copiés dans votre projet pour servir de base à de nouvelles traductions.

Rapporter un bogue

6.2.3.2. Traduire un logger, une exception ou un message internationalisé

Les messages d'exception ou de journalisation définis dans les interfaces qui utilisent JBoss Logging Tools peuvent avoir des traductions dans les fichiers de propriétés.

La procédure suivante vous montre comment créer et utiliser un fichier de propriétés de traduction. On assume que vous possédez déjà un projet avec une ou plusieurs interfaces définies pour les exceptions internationalisées ou pour les messages de journalisation.

Voir le guide de démarrage **`logging-tools`** pour trouver un exemple complet.

Pré-requis

1. Vous devez déjà posséder un projet Maven en cours.
2. Le projet devra avoir été configuré pour JBoss Logging Tools.
3. Le projet devra contenir une ou plusieurs interfaces qui définissent les exceptions ou message log d'internationalisation.
4. Le projet devra être configuré pour générer des fichiers de propriétés modèles.

Procédure 6.5. Traduire un message, une exception ou un message internationalisé

1. **Créer des fichiers de propriétés modèles**
   Exécuter la commande **`mvn compile`** pour créer les modèles de fichiers de propriétés de traduction.
2. **Ajouter le fichier modèle à votre projet.**
   Copier le modèle des interfaces que vous souhaitez traduire depuis le répertoire où elles ont été créées dans le répertoire **`src/main/resources`** de votre projet. Les fichiers de propriétés doivent figurer dans le même package des interfaces qu'ils traduisent.
3. **Renommer le fichier modèle copié**
   Renommer la copie du fichier modèle selon la traduction qu'il contient. Par exemple, **`GreeterLogger.i18n_fr_FR.properties`**.
4. **Traduire le contenu du modèle**
   Modifier le nouveau fichier de propriétés de traduction de manière à contenir la traduction appropriée.

   ```
   # Level: Logger.Level.INFO
   # Message: Hello message sent.
   logHelloMessageSent=Bonjour message envoyé.
   ```

   Répéter les étapes deux, trois et quatre pour chaque traduction de chaque lot exécuté.

RÉSULTAT : le projet contient désormais des traductions pour un ou plusieurs lots de messages ou de loggers. La construction du projet génèrera les classes appropriées aux messages de journalisation avec les traductions fournies. Il n'est pas nécessaire d'invoquer des méthodes explicitement ou de fournir des paramètres pour des langues spécifiques, JBoss Logging Tools choisit automatiquement la bonne classe pour la locale actuelle du serveur d'application.

Le code source des classes générées peuvent être consultées dans **`target/generated-sources/annotations/`**.

Rapporter un bogue

6.2.4. Personnalisation des messages de journalisation internationalisés

6.2.4.1. Ajouter les ids de messages et les codes de projets aux messages de journalisation

Cette tâche montre comment ajouter les ids de messages et les codes de projets aux messages de journalisation internationalisés créés en utilisant JBoss Logging Tools. Un message de journalisation doit contenir à la fois un code de projet et un id de message afin qu'ils puissent s'afficher dans le journal. Si un message ne contient pas à la fois un code de projet et un id de message, alors aucun ne s'affichera.

Voir le guide de démarrage **`logging-tools`** pour trouver un exemple complet.

Conditions préalables

1. Vous devez déjà posséder un projet avec des messages de journalisation internationalisés. Voir Section 6.2.2.1, « Créer des messages log internationalisés ».

2. Vous devrez connaître le code de projet que vous utilisez. Vous pouvez utiliser un code de projet simple, ou en définir un différent pour chaque interface.

Procédure 6.6. Ajouter les ids et les codes de projets aux messages de journalisation

1. **Indiquer le code de projet de l'interface.**
   Indiquer le code du projet par l'attribut projectCode de l'annotation **`@MessageLogger`** qui est attachée à une interface de logger personnalisée. Tous les messages définis dans l'interface utiliseront ce code de projet.

   ```
   @MessageLogger(projectCode="ACCNTS")
   interface AccountsLogger extends BasicLogger
   {

   }
   ```

2. **Indiquer les ids de messages**
   Indiquer un id de message pour chaque message qui utilise un attribut **`id`** de l'annotation **`@Message`** liée à la méthode qui définit le message.

   ```
   @LogMessage
   @Message(id=43, value = "Customer query failed, Database not
   available.")  void customerQueryFailDBClosed();
   ```

Les messages de journalisation qui ont à la fois un id de message et un code de projet associés ajouteront ces informations au message enregistré.

```
10:55:50,638 INFO  [com.company.accounts.ejb] (MSC service thread 1-4)
ACCNTS000043: Customer query failed, Database not available.
```

Rapporter un bogue

6.2.4.2. Indiquer le niveau de journalisation d'un message

Le niveau de journalisation par défaut d'un message défini par une interface par JBoss Logging Tools est **`INFO`**. Un niveau de journalisation différent peut être indiqué avec l'attribut **`level`** de l'annotation **`@LogMessage`** jointe à la méthode de journalisation.

Procédure 6.7. Indiquer le niveau de journalisation d'un message

1. **Indiquer l'attribut du niveau**
   Ajouter l'attribut **`level`** à l'annotation de **`@LogMessage`** de la définition de la méthode de message de journalisation.

2. **Allouer un niveau de journalisation**
   Allouer l'attribut **`level`** à la valeur du niveau de journalisation de ce message. Les valeurs valides de **`level`** sont les six constantes énumérées définies dans **`org.jboss.logging.Logger.Level`** : **`DEBUG`**, **`ERROR`**, **`FATAL`**, **`INFO`**, **`TRACE`**, et **`WARN`**.

   ```
   Import org.jboss.logging.Logger.Level;

   @LogMessage(level=Level.ERROR)
   @Message(value = "Customer query failed, Database not available.")
   void customerQueryFailDBClosed();
   ```

Invoquer une méthode de journalisation dans l'échantillon ci-dessus produira un message de journalisation du niveau **`ERROR`**.

```
10:55:50,638 ERROR  [com.company.app.Main] (MSC service thread 1-4)
 Customer query failed, Database not available.
```

Rapporter un bogue

6.2.4.3. Personnaliser les messages de journalisation par des paramètres

Les méthodes d'enregistrement personnalisé peuvent définir les paramètres. Ces paramètres sont utilisés pour passer des informations supplémentaires à afficher dans le message journalisé. L'endroit où les paramètres apparaissent dans le message de journal est spécifié dans le message lui-même à l'aide d'une indexation explicite ou ordinaire.

Procédure 6.8. Personnaliser les messages de journalisation avec des paramètres

1. **Ajouter des paramètres à la définition de la méthode**
   Les paramètres d'un type donné peuvent être ajoutés à la définition de la méthode. Quel que soit le type, la représentation String du paramètre est ce qui est affiché dans le message.

2. **Ajouter les références de paramètre dans le message de journalisation**

Les références peuvent utiliser des indexations explicites ou ordinaires.

- Pour utiliser des indexations ordinaires, insérer les caractères **%s** dans la chaîne de message où vous souhaitez voir apparaître chaque paramètre. Le premier **%s** insèrera le premier paramètre, le deuxième insèrera le deuxième paramètre, et ainsi de suite.
- Pour les indexations explicites, insérer les caractères **%{#}** dans le message où *#* correspond au numéro du paramètre que vous souhaitez apercevoir.

IMPORTANT

L'utilisation des indexations explicites permet aux références de paramètres du message d'être dans un ordre différent que celui défini dans la méthode. C'est important pour les messages traduits qui peuvent nécessiter un ordonnancement des paramètres différent.

Le nombre de paramètres doit correspondre au nombre de références aux paramètres dans le message spécifié ou le code ne compilera pas. Un paramètre marqué avec l'annotation **@Cause** n'est pas inclus dans le nombre de paramètres.

Exemple 6.1. Paramètres de messages qui utilisent des indexations ordinaires.

```
@LogMessage(level=Logger.Level.DEBUG)
@Message(id=2, value="Customer query failed, customerid:%s, user:%s")
void customerLookupFailed(Long customerid, String username);
```

Exemple 6.2. Paramètres de messages qui utilisent des indexations explicites

```
@LogMessage(level=Logger.Level.DEBUG)
@Message(id=2, value="Customer query failed, customerid:%{1}, user:%
{2}")
void customerLookupFailed(Long customerid, String username);
```

Rapporter un bogue

6.2.4.4. Indiquer une exception comme cause d'un message de journalisation

JBoss Logging Tools permet à un paramètre d'une méthode de journalisation personnalisée d'être défini comme cause du message. Ce paramètre doit être du type **Throwable** ou d'une de ses sous-classes et doit être marqué par l'annotation **@Cause** . Ce paramètre ne peut pas être référencé dans le message de journal comme les autres paramètres et il s'affiche à la suite du message de journalisation.

La procédure suivante montre comment mettre à jour une méthode de journalisation en utilisant le paramètre **@Cause** pour indiquer l'exception "qui cause". On suppose que vous avez déjà créé des messages de journalisation internationalisés auxquels vous souhaitez ajouter cette fonctionnalité.

Procédure 6.9. Indiquer une exception comme cause d'un message de journalisation

1. **Ajouter le paramètre**
   Ajouter un paramètre **Throwable** ou une sous-classe à la méthode.

```
@LogMessage
@Message(id=404, value="Loading configuration failed. Config
file:%s")
void loadConfigFailed(Exception ex, File file);
```

2. **Ajouter l'annotation**
   Ajouter l'annotation **@Cause** au paramètre.

```
import org.jboss.logging.Cause

@LogMessage
@Message(value = "Loading configuration failed. Config file: %s")
void loadConfigFailed(@Cause Exception ex, File file);
```

3. **Invoquer la méthode**
   Quand la méthode est invoquée dans votre code, un objet du type qui convient devra être passé et sera affiché à la suite du message de journalisation.

```
try
{
   confFile=new File(filename);
   props.load(new FileInputStream(confFile));
}
catch(Exception ex) //in case properties file cannot be read
{
     ConfigLogger.LOGGER.loadConfigFailed(ex, filename);
}
```

   Vous trouverez ci-dessous la sortie des extraits de code ci-dessus, si le code a lancé une exception de type **FileNotFoundException**.

```
10:50:14,675 INFO [com.company.app.Main] (MSC service thread 1-3)
Loading configuration failed. Config file: customised.properties
java.io.FileNotFoundException: customised.properties (No such file
or directory)
   at java.io.FileInputStream.open(Native Method)
   at java.io.FileInputStream.<init>(FileInputStream.java:120)
   at com.company.app.demo.Main.openCustomProperties(Main.java:70)
   at com.company.app.Main.go(Main.java:53)
   at com.company.app.Main.main(Main.java:43)
```

Rapporter un bogue

6.2.5. Personnalisation des exceptions internationalisées

6.2.5.1. Ajouter les ids et les codes de projets aux messages d'exceptions

La procédure suivante montre les étapes requises pour ajouter des ID de message et des codes de projets aux messages d'exception internationalisés créés en utilisant les JBoss Logging Tools.

Les ID de messages et les codes de projet sont des identificateurs uniques qui sont ajoutés à chaque message affiché par les exceptions internationalisées. Ces codes d'identification permettent de créer une référence pour tous les messages d'exception d'une application pour qu'une personne puisse

rechercher le sens d'un message d'exception écrit dans une langue qu'elle ne comprent pas.

Conidtions préalables

1. Vous devrez déjà avoir un projet avec des exceptions internationalisées. Voir Section 6.2.2.3, « Créer des exceptions personnalisées ».

2. Vous devrez connaître le code de projet que vous utilisez. Vous pouvez utiliser un code de projet simple, ou en définir un différent pour chaque interface.

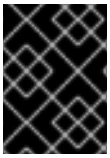
Procédure 6.10. Ajouter les ids et les codes de projets aux messages d'exceptions

1. **Indiquer un code de projet**
   Indiquer le code de projet en utilisant l'attribut **`projectCode`** de l'annotation **`@MessageBundle`** attachée à une interface de lot d'exceptions. Tous les messages qui sont définis dans l'interface utiliseront ce code de projet.

```
@MessageBundle(projectCode="ACCTS")
interface ExceptionBundle
{
   ExceptionBundle EXCEPTIONS =
Messages.getBundle(ExceptionBundle.class);
}
```

2. **Indiquer les ID de messages**
   Indiquer l'id de message pour chaque exception en utilisant l'attribut **`id`** de l'annotation **`@Message`** associée à la méthode qui définit l'exception.

```
@Message(id=143, value = "The config file could not be opened.")
IOException configFileAccessError();
```



IMPORTANT

Un message qui comprend à la fois un code de projet et un ID de message les affiche à la suite du message. Si un message ne les possède pas, ils ne seront pas affichés.

Exemple 6.3. Créer des exceptions internationalisées

Cette interface de lot d'exceptions possède le code de projet d'ACCTS, avec une simple méthode d'exception ayant comme Id 143.

```
@MessageBundle(projectCode="ACCTS")
interface ExceptionBundle
{
    ExceptionBundle EXCEPTIONS =
Messages.getBundle(ExceptionBundle.class);

    @Message(id=143, value = "The config file could not be opened.")
    IOException configFileAccessError();
}
```

L'objet d'exception peut être obtenu et lancé grâce au code suivant.

```
throw ExceptionBundle.EXCEPTIONS.configFileAccessError();
```

Cela aurait pour effet d'afficher un message d'exception qui ressemble à ce qui suit :

```
Exception in thread "main" java.io.IOException: ACCTS000143: The config
file could not be opened.
at com.company.accounts.Main.openCustomProperties(Main.java:78)
at com.company.accounts.Main.go(Main.java:53)
at com.company.accounts.Main.main(Main.java:43)
```

Rapporter un bogue

6.2.5.2. Personnaliser les messages d'exception avec des paramètres

Les méthodes de lots d'exceptions qui définissent des exceptions peuvent définir les paramètres. Ces paramètres sont utilisés pour passer des informations supplémentaires à afficher dans le message d'exception. L'endroit où les paramètres apparaissent dans le message d'exception est spécifié dans le message lui-même à l'aide d'une indexation explicite ou ordinaire.

La procédure suivante vous montre les étapes requises pour utiliser les paramètres de méthodes pour personnaliser des exceptions de méthodes.

Procédure 6.11. Personnaliser les messages d'exception avec les paramètres

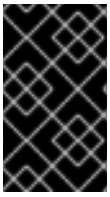
1. **Ajouter des paramètres à la définition de la méthode**
   Les paramètres d'un type donné peuvent être ajoutés à la définition de la méthode. Quel que soit le type, la représentation **`String`** du paramètre est ce qui est affiché dans le message.

2. **Ajouter les références de paramètre au message d'exception**
   Les références peuvent utiliser des indexations explicites ou ordinaires.

   - Pour utiliser des indexations ordinaires, insérer les caractères **`%s`** dans la chaîne de message où vous souhaitez voir apparaître chaque paramètre. Le premier **`%s`** insèrera le premier paramètre, le deuxième insèrera le deuxième paramètre, et ainsi de suite.

   - Pour les indexations explicites, insérer les caractères **`%{#}`** dans le message où *#* correspond au numéro du paramètres que vous souhaitez apercevoir.

   L'utilisation des indexations explicites permet aux références de paramètres du message d'être dans un ordre différent que celui défini dans la méthode. C'est important pour les messages traduits qui peuvent nécessiter un ordonnancement des paramètres différent.



IMPORTANT

Le nombre de paramètres doit correspondre au nombre de références aux paramètres dans le message spécifié ou le code ne compilera pas. Un paramètre marqué avec l'annotation **`@Cause`** n'est pas inclus dans le nombre de paramètres.

Exemple 6.4. Utilisation des indexations ordinaires

```
@Message(id=143, value = "The config file %s could not be opened.")
IOException configFileAccessError(File config);
```

Exemple 6.5. Utilisation des indexations explicites

```
@Message(id=143, value = "The config file %{1} could not be opened.")
IOException configFileAccessError(File config);
```

Rapporter un bogue

6.2.5.3. Indiquer une exception comme cause d'une autre exception

Les exceptions renvoyées par les méthodes de lots d'exceptions peuvent avoir une autre exception spécifiée comme cause sous-jacente. Cela se fait par l'ajout d'un paramètre à la méthode et en annotant le paramètre **@Cause**. Ce paramètre est utilisé pour passer l'exception de «cause». Ce paramètre ne peut pas être référencé dans le message d'exception.

La procédure suivante montre comment mettre à jour une méthode de lot d'exception en utilisant le paramètre **@Cause** pour indiquer l'exception de « cause ». On suppose que vous avez déjà créé des messages de journalisation internationalisés auxquels vous souhaitez ajouter cette fonctionnalité.

Procédure 6.12. Indiquer une exception comme « cause » d'une autre exception

1. **Ajouter le paramètre**
   Ajouter un paramètre **Throwable** ou une sous-classe à la méthode.

   ```
   @Message(id=328, value = "Error calculating: %s.")
   ArithmeticException calculationError(Throwable cause, String msg);
   ```

2. **Ajouter l'annotation**
   Ajouter l'annotation **@Cause** au paramètre.

   ```
   import org.jboss.logging.Cause

   @Message(id=328, value = "Error calculating: %s.")
   ArithmeticException calculationError(@Cause Throwable cause, String
   msg);
   ```

3. **Invoquer la méthode**
   Invoquer la méthode d'interface pour obtenir un objet d'exception. Le cas d'utilisation le plus commun est de lancer une exception à partir d'un catch block en utilisant l'exception récupérée comme « cause ».

   ```
   try
   {
      ...
   }
   catch(Exception ex)
   {
      throw ExceptionBundle.EXCEPTIONS.calculationError(
                                        ex, "calculating payment due
   per day");
   }
   ```

-

Exemple 6.6. Indiquer une exception comme « cause » d'une autre exception

Ce lot d'exceptions définit une méthode simple qui renvoie une exception du type ArithmeticException.

```
@MessageBundle(projectCode = "TPS")
interface CalcExceptionBundle
{
 CalcExceptionBundle EXCEPTIONS =
Messages.getBundle(CalcExceptionBundle.class);

    @Message(id=328, value = "Error calculating: %s.")
    ArithmeticException calcError(@Cause Throwable cause, String value);

}
```

Cet extrait de code exécute une opération qui lève une exception parce qu'il tente de diviser un entier par zéro. L'exception est interceptée et une nouvelle exception est créée en utilisant la première comme « cause ».

```
int totalDue = 5;
int daysToPay = 0;
int amountPerDay;

try
{
   amountPerDay = totalDue/daysToPay;
}
catch (Exception ex)
{
   throw CalcExceptionBundle.EXCEPTIONS.calcError(ex, "payments per
day");
}
```

Voici à quoi ressemble le message :

```
Exception in thread "main" java.lang.ArithmeticException: TPS000328:
Error calculating: payments per day.
 at com.company.accounts.Main.go(Main.java:58)
 at com.company.accounts.Main.main(Main.java:43)
Caused by: java.lang.ArithmeticException: / by zero
 at com.company.accounts.Main.go(Main.java:54)
 ... 1 more
```

Rapporter un bogue

6.2.6. Référence

6.2.6.1. Configuration Maven JBoss Logging Tools

Pour construire un projet Maven qui utilise JBoss Logging Tools pour l'internationalisation, vous devrez appliquer les changements suivants à la configuration du projet dans le fichier **pom.xml** :

Veuillez vous référer au quick start **logging-tools** pour obtenir un exemple de fichier **pom.xml** complet.

1. Le référentiel JBoss Maven Repository doit être activé pour le projet. Voir Section 2.3.2, « Configurer le référentiel JBoss EAP 6 Platform Maven Repository par les paramètres de configuration de Maven ».

2. Les dépendances Maven de **jboss-logging** et **jboss-logging-processor** doivent être ajoutées. Les deux dépendances sont disponibles sur JBoss EAP 6, ainsi l'élément scope de chaque dépendance peut être définie comme **provided** tel qu'il a été montré.

```
<dependency>
   <groupId>org.jboss.logging</groupId>
   <artifactId>jboss-logging-processor</artifactId>
   <version>1.0.0.Final</version>
   <scope>provided</scope>
</dependency>

<dependency>
   <groupId>org.jboss.logging</groupId>
   <artifactId>jboss-logging</artifactId>
   <version>3.1.0.GA</version>
   <scope>provided</scope>
</dependency>
```

3. Le **maven-compiler-plugin** doit utiliser au minimum la version **2.2** et être configuré pour la cible et les sources générées de **1.6**.

```
<plugin>
   <groupId>org.apache.maven.plugins</groupId>
   <artifactId>maven-compiler-plugin</artifactId>
   <version>2.3.2</version>
   <configuration>
      <source>1.6</source>
      <target>1.6</target>
   </configuration>
</plugin>
```

Rapporter un bogue

6.2.6.2. Format de fichier de propriétés de traduction

Les fichiers de propriété utilisés pour les traductions de messages dans JBoss Logging Tools sont des fichiers de propriétés Java standards. Le format du fichier est le format **key=value** simple, orienté-ligne, décrit dans la documentation pour la classe **java.util.Properties**, http://docs.oracle.com/javase/6/docs/api/java/util/Properties.html.

Le format du nom du fichier a le format suivant :

```
InterfaceName.i18n_locale_COUNTRY_VARIANT.properties
```

- **`InterfaceName`** est le nom de l'interface où les traductions s'appliquent.
- **`locale`**, **`COUNTRY`**, et **`VARIANT`** identifient les paramètres régionaux correspondant à la traduction.
- **`locale`** et **`COUNTRY`** indiquent la langue et le pays en utilisant respectivement les codes Langues et Pays ISO-639 et ISO-3166. **`COUNTRY`** est optionnel.
- **`VARIANT`** est un identifiant optionnel pouvant être utilisé pour identifier les traductions qui ne s'appliquent qu'à un système d'exploitation ou à un navigateur en particulier.

Les propriétés contenues dans le fichier de traduction sont les noms des méthodes de l'interface en cours de traduction. La valeur assignée de la propriété est la traduction. Si une méthode est surchargée, c'est indiqué en ajoutant un point, puis le nombre de paramètres pour le nom. Les méthodes de traduction peuvent seulement être surchargées en fournissant un nombre différent de paramètres.

Exemple 6.7. Échantillon de fichier de propriétés de traductions

Nom du fichier : **`GreeterService.i18n_fr_FR_POSIX.properties`**.

```
# Level: Logger.Level.INFO
# Message: Hello message sent.
logHelloMessageSent=Bonjour message envoyé.
```

Rapporter un bogue

6.2.6.3. Références aux annotations de JBoss Logging Tools

Les annotations suivantes sont définies dans JBoss Logging pour pouvoir être utilisées pour l'internationalisation et la localisation des messages de journalisation, strings et exceptions.

Tableau 6.1. Annotations de JBoss Logging Tools

Annotation	Cible	Description	Attributs
`@MessageBundle`	Interface	Définit l'interface en tant que Lot de Messages	**`projectCode`**
`@MessageLogger`	Interface	Définit l'interface en tant que Message Logger	**`projectCode`**
`@Message`	Méthode	Peut être utilisé dans les Lots de Messages et Message Loggers. Dans un Message Logger, définit une méthode comme étant un logger localisé. Dans un Lot de Messages, définit la méthode comme étant une méthode qui renvoie une Chaîne localisées ou un objet d'Exception.	**`value`**, **`id`**

Annotation	Cible	Description	Attributs
`@LogMessage`	Méthode	Définit une méthode dans un Message Logger comme étant une méthode de logging.	**`level`** (default **`INFO`**)
`@Cause`	Paramètre	Définit un paramètre comme étant un paramètre faisant passer une exception en tant que cause d'un Message Log ou autre exception.	-
`@Param`	Paramètre	Définit un paramètre comme étant un paramètre passé au constructeur d'une Exception.	-

Rapporter un bogue

CHAPITRE 7. ENTERPRISE JAVABEANS

7.1. INTRODUCTION

7.1.1. Entreprise JavaBeans

Enterprise JavaBeans (EJB) 3.1 est une API pour développer des applications de Java EE distribuées, transactionnelles, sécurisées et portables grâce à l'utilisation des composants côté serveur appelés Enterprise Beans. Enterprise Beans implémente la logique métier d'une application, de manière découplée, qui encourage la réutilisation. Enterprise JavaBeans 3.1 est documenté dans la spécification Java EE JSR-318.

JBoss EAP 6 prend en charge la génération d'applications qui utilisent la spécification Enterprise JavaBeans 3.1.

Rapporter un bogue

7.1.2. Groupe de fonctionnalités EJB 3.1

Les fonctionnalités suivantes sont prises en charge dans EJB 3.1

- Session Beans
- Message Driven Beans
- Vues non-interfaces
- Interfaces locales
- Interfaces distantes
- Services web JAX-WS
- Services web JAX-RS
- Service de minuterie
- Appels asynchrones
- Intercepteurs
- Interopérabilité RMI/IIOP
- Support de transaction
- Sécurité
- API intégrable

Les fonctionnalités suivantes sont prises en charge dans EJB 3.1 mais sont susceptibles d'être supprimées. Cela signifie qu'elles seront certainement optionnelles dans Java EE 7.

- Entity Beans (persistance gérée-bean et conteneur))
- Vues client Entity Beans EJB 2.1

- EJB Query Language (EJB QL)
- Services Web basés JAX-RPC (points de terminaison et vues client)

Rapporter un bogue

7.1.3. EJB 3.1 Lite

EJB Lite est un sous-ensemble de la spécification EJB 3.1. Il fournit une version simplifiée de la spécification EJB 3.1 complète dans le cadre du profil web Java EE 6.

EJB Lite simplifie l'implémentation de la logique métier dans les applications web avec les beans entreprise en :

1. ne supportant que les fonctionnalités qui sont utiles pour les applications-web, et
2. en permettant aux EJB d'être déployés dans le même fichier WAR sous forme d'application web.

Rapporter un bogue

7.1.4. Fonctionnalités EJB 3.1 Lite

EJB Lite inclut les fonctionnalités suivantes :

- Session Beans sans état, avec état, et singleton
- Interfaces commerciales locales et beans "sans interface"
- Intercepteurs
- Transactions gérées-conteneur et gérées-bean
- Sécurité déclarative et programmatique
- API intégrable

Les fonctionnalités suivantes d'EJB 3.1 ne sont par incluses spécifiquement :

- Interfaces éloignées
- Intéropérabilité RMI-IIOP
- Ponts de terminaison de services web JAX-WS
- Service de minuterie EJB
- Invocations de session beans asynchrones
- Message-driven beans

Rapporter un bogue

7.1.5. Beans Enterprise

Les beans enterprise sont des composants d'applications côté serveur, définis dans la spécification Enterprise JavaBeans (EJB) 3.1, JSR-318. Les beans enterprise sont conçus pour l'implémentation d'une logique commerciale d'application d'une manière découplée, pour encourager sa réutilisation.

Les beans enterprise sont écrits comme des classes Java et sont annotés avec les annotations EJB appropriées. Ils peuvent être déployés sur le serveur d'applications dans leurs propres archives (un fichier JAR) ou être déployés dans le cadre d'une application Java EE. Le serveur d'applications gère le cycle de vie de chaque bean entreprise et leur fournit des services comme la sécurité, les transactions et la gestion de concurrence.

Un bean enterprise peut également définir un nombre d'interfaces de métier. Les interfaces de métier vous proposent un plus grand contrôle sur les méthodes bean disponibles aux clients et peut également vous donner accès aux clients qui exécutent dans les JVM à distance.

Il existe trois types de beans enterprise : les session beans, les message-driven beans et les entity beans.

IMPORTANT

Les entity beans sont maintenant obsolètes dans EJB 3.1 et Red Hat recommande d'utiliser des entités JPA à la place. Red Hat ne recommande d'utiliser des entity beans que pour les compatibilités rétroactives avec les systèmes hérités.

Rapporter un bogue

7.1.6. Écriture des beans Enterprise

Les beans Enterprise sont des composants côté serveur conçus pour encapsuler la logique métier d'une manière découplée d'une application client spécifique. En implémentant votre logique de métier dans des beans Enterprise, vous serez en mesure de réutiliser ces beans dans plusieurs applications.

Les beans Enterprise sont écrits sous forme de classes Java annotées et n'ont pas besoin d'implémenter des interfaces EJB particulières, ni de représenter des sous-classes de classes EJB Super pour pouvoir être considérés bean Enterprise.

Les Bean Enterprise EJB 3.1 sont empaquetés et déployés dans des fichiers JAR (Archives Java). Un fichier JAR Bean Enterprise peut être déployé dans votre serveur d'applications, ou être inclus dans un fichier EAR et déployé dans cette application. Il est également possible de déployer des beans Enterprise dans un fichier WAR en parallèle à une application web.

Rapporter un bogue

7.1.7. Interfaces métier de Session Bean

7.1.7.1. Interfaces de métier Bean Enterprise

Une interface de métier EJB est une interface Java écrite par le développeur de bean qui fournit les déclarations de méthodes publiques d'un bean de session disponible aux clients. Les beans de session peuvent implémenter un certain nombre d'interface ou pas du tout (bean "no-interface").

Les interfaces de métier peuvent être déclarées comme interfaces locales ou à distance, mais pas les deux à la fois.

Rapporter un bogue

7.1.7.2. Interfaces de métier locales EJB

Une interface de métier locale EJB déclare les méthodes qui sont disponibles quand le bean et le client sont dans la même JVM. Quand un bean de session implémente une interface de métier locale, seules les méthodes déclarées dans cette interface seront disponibles aux clients.

Rapporter un bogue

7.1.7.3. Interfaces métier à distance EJB

Une interface de métier à distance EJB déclare les méthodes qui sont disponibles pour les clients à distance. L'accès à distance à un bean de session qui implémente une interface à distance est fourni automatiquement par un conteneur EJB.

Un client à distance est un client qui exécute sur une JVM différente et peut inclure des applications de bureau ainsi que des applications web, des services et bean Enterprise déployés dans un serveur d'applications différent.

Les clients locaux peuvent accéder aux méthodes exposées par une interface de métier à distance. Cela est effectué par les mêmes méthodes que pour les clients à distance et comprend les mêmes protocoles de départ que pour une demande à distance.

Rapporter un bogue

7.1.7.4. Beans EJB no-interface

Un bean de session qui n'implémente pas d'interface de métier s'appelle un bean « no-interface ». Toutes les méthodes publiques de beans no-interface sont accessibles aux clients locaux.

Un bean de session qui implémente une interface de métier peut également être écrit pour exposer un affichage « no-interface ».

Rapporter un bogue

7.2. CRÉER DES PROJETS ENTERPRISE BEAN

7.2.1. Créer un projet d'archives EJB avec le Red Hat Studio JBoss Developer

Cette tâche décrit comment créer un EJB (Enterprise JavaBeans) avec le Red Hat Studio JBoss Developer

Conditions préalables

- Un serveur et un temps d'exécution du serveur ont été établis pour JBoss EAP 6.

Procédure 7.1. Créer un projet EJB dans le Red Hat JBoss Developer Studio

1. **Créer un nouveau projet**
   Pour ouvrir l'Assistant New EJB Project, naviguer jusqu'au menu **Fichier**, sélectionner **Nouveau** puis **Projet EJB**.

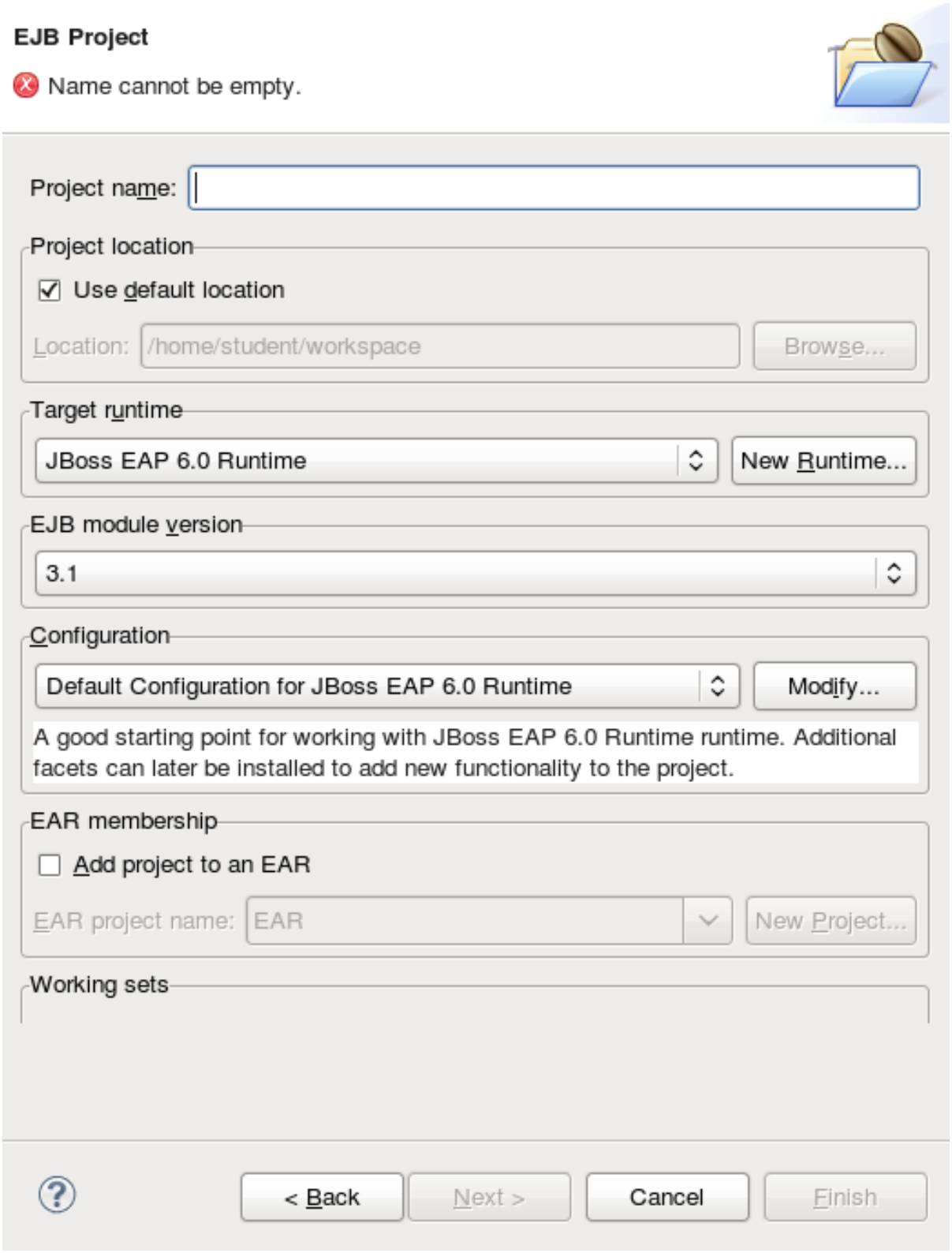


Figure 7.1. Assistant New EJB Project

2. **Détails**
   Fournir les informations suivantes :

   - Nom du projet.

     Il s'agira du nom du projet qui apparaît dans Red Hat JBoss Developer Studio, et également le nom du fichier par défaut du fichier JAR déployé.

   - Emplacement du projet

Le répertoire où les fichiers du projet seront sauvegardés. La valeur par défaut est un répertoire qui se trouve dans l'espace de travail en cours.

- Runtime cible.

  Il s'agit du runtime de serveur utilisé pour le projet. Il devra être défini au même runtime JBoss EAP 6, qui est utilisé par le serveur dans lequel vous souhaitez déployer.

- Version de module EJB. Il s'agit de la version de la spécification EJB avec laquelle vos beans Enterprise devront se conformer. Red Hat recommande que vous utilisiez **`3.1`**.

- Configuration. Cela vous permet d'ajuster les fonctionnalités prises en charge par votre projet. Utiliser la configuration par défaut pour le runtime que vous avez sélectionné.

Cliquer sur **`Suivant`** pour continuer.

3. **Java Build Configuration**
   Cet écran vous permet de personnaliser les répertoires qui contiennent les fichiers source de Java et le répertoire où les résultats de la génération se trouvent.

   Conservez cette configuration sans y apporter aucun changement et cliquer sur le bouton **`Suivant`**.

4. **Paramétrage du Module EJB**
   Cocher la case **`Generate ejb-jar.xml deployment descriptor`** (Générer descripteur de déploiement ejb-jar.xml) si un descripteur de déploiement est requis. Le descripteur de déploiement est une option dans EJB 3.1, et peut être rajouté plus tard, si nécessaire.

   Cliquer sur le bouton **`Terminé`** et le projet sera créé. Il s'affichera dans Project Explorer.

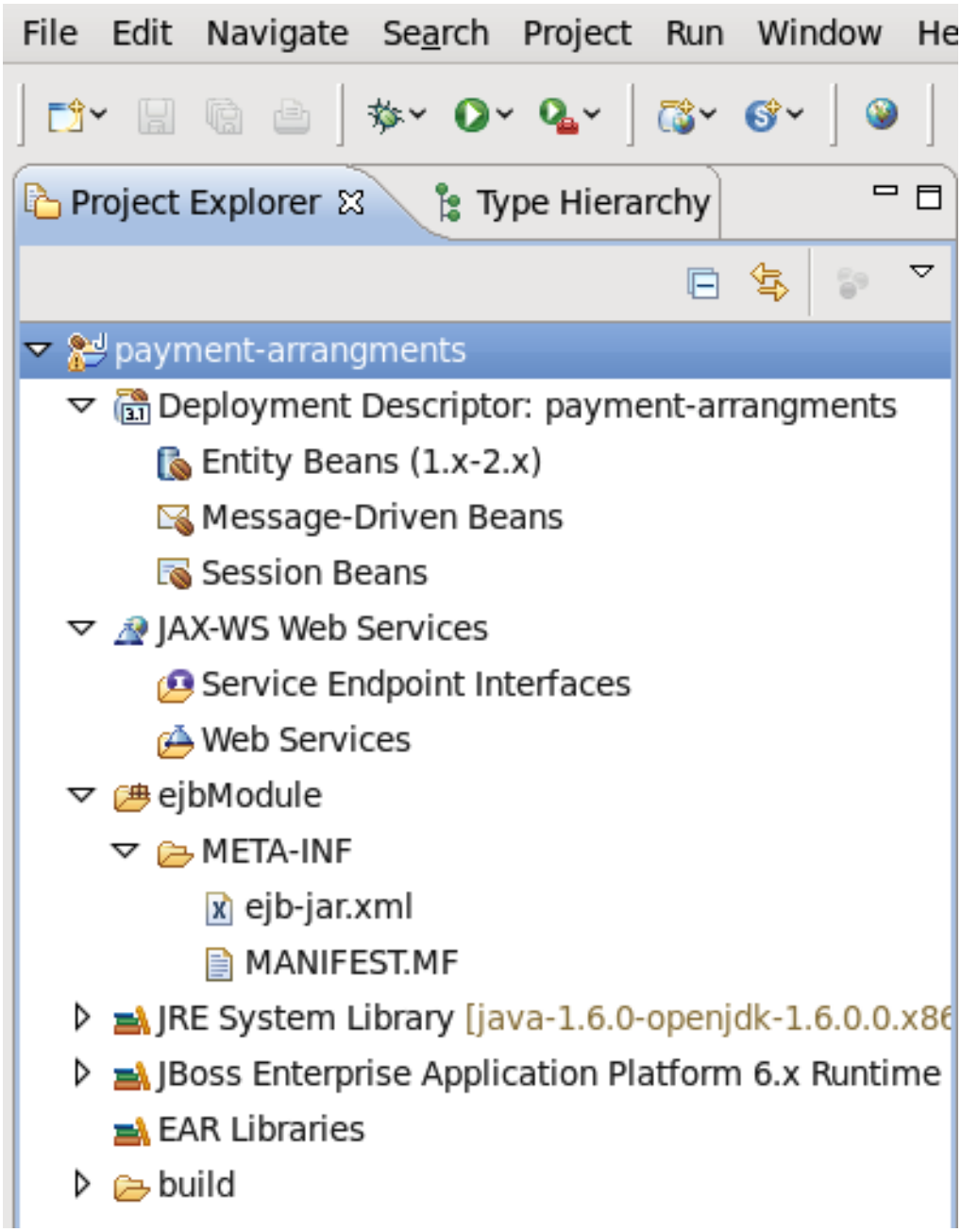


Figure 7.2. Nouveau projet EJB créé dans Project Explorer

5. **Ajouter l'artifact généré dans le serveur en vue du déploiement**
   Ouvrir le dialogue **`Add and Remove`** en cliquant à droite sur le serveur dans lequel vous souhaitez générer l'artefact dans l'onglet serveur, et sélectionner 'Ajouter et Supprimer'.

   Sélectionner la ressource à déployer à partir de la colonne **`Available`** et cliquer sur le bouton **`Add`**. La ressource sera déplacée dans la colonne **`Configured`**. Cliquer sur **`Finish`** pour fermer le dialogue.

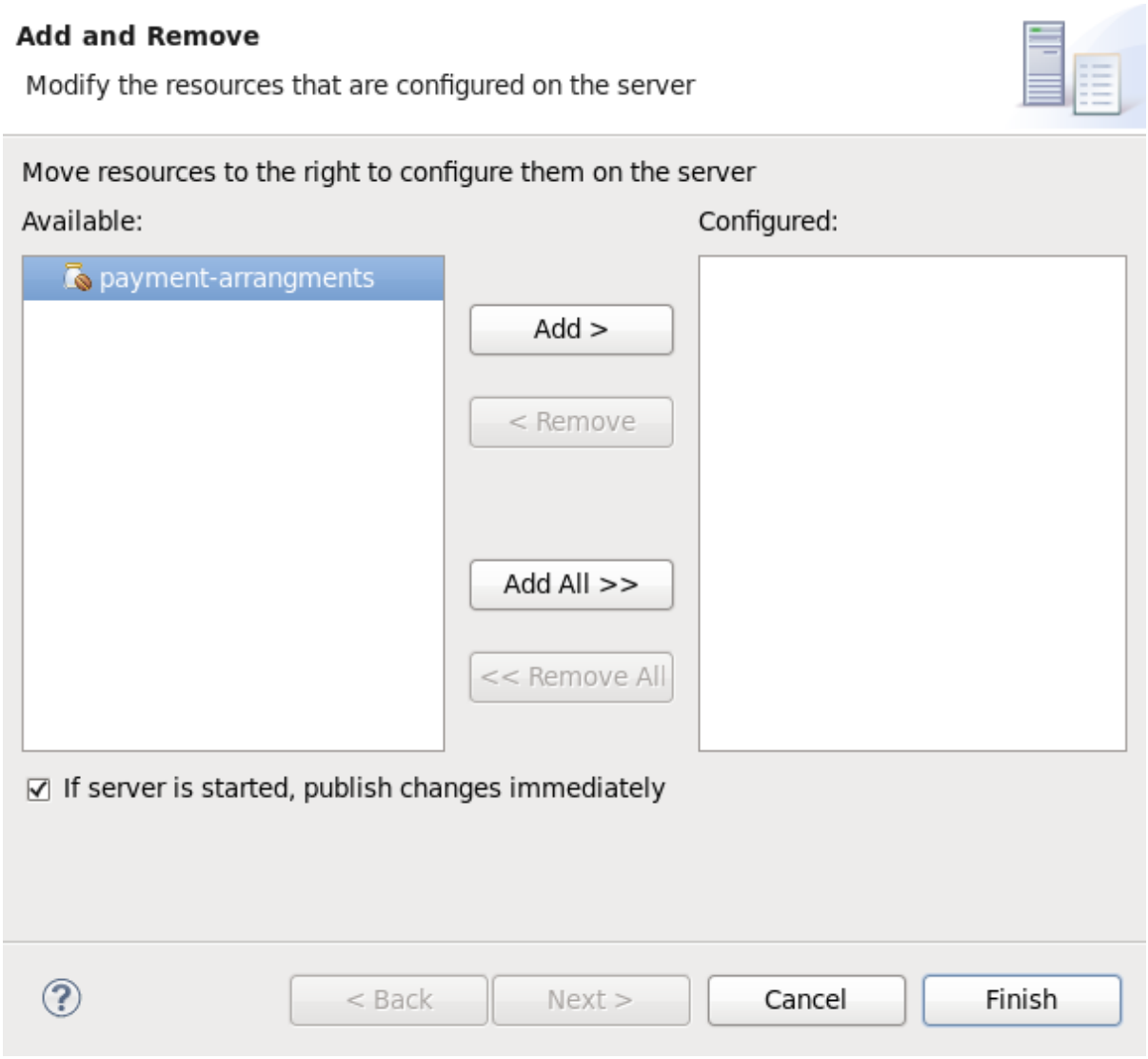


Figure 7.3. Ajouter et Supprimer le dialogue

Résultat

Vous avez maintenant un Projet EJB dans Red Hat JBoss Developer Studio pouvant être construit et déployé pour un serveur spécifique.

Si aucun bean Enterprise n'est ajouté au projet, alors Red Hat JBoss Developer Studio affichera l'avertissement « un module EJB doit contenir un ou plusieurs beans Enterprise. » Cet avertissement disparaîtra une fois qu'un ou plusieurs beans Enterprise beans auront été ajoutés au projet.

Rapporter un bogue

7.2.2. Créer un projet EJB Archive dans Maven.

Cette tâche montre comment créer, dans Maven, un projet contenant un ou plusieurs Beans Enterprise empaquetés dans un fichier JAR.

Prérequis :

- Maven est déjà installé.
- Vous comprenez les bases d'utilisation de Maven.

Procédure 7.2. Créer un projet EJB Archive dans Maven

1. **Créer le projet Maven**
   Un projet EJB peut être créé en utilisant le système archétype de Maven et l'archétype **ejb-javaee6**. Pour ce faire, exécuter la commande **mvn** avec les paramètres suivants :

   ```
    mvn archetype:generate -
   DarchetypeGroupId=org.codehaus.mojo.archetypes -
   DarchetypeArtifactId=ejb-javaee6
   ```

   Maven vous demandera le **groupId**, le **artifactId**, la **version** et le **package** de votre projet.

   ```
   [localhost]$ mvn archetype:generate -
   DarchetypeGroupId=org.codehaus.mojo.archetypes -
   DarchetypeArtifactId=ejb-javaee6
   [INFO] Scanning for projects...
   [INFO]
   [INFO] ------------------------------------------------------------
   --------------
   [INFO] Building Maven Stub Project (No POM) 1
   [INFO] ------------------------------------------------------------
   --------------
   [INFO]
   [INFO] >>> maven-archetype-plugin:2.0:generate (default-cli) @
   standalone-pom >>>
   [INFO]
   [INFO] <<< maven-archetype-plugin:2.0:generate (default-cli) @
   standalone-pom <<<
   [INFO]
   [INFO] --- maven-archetype-plugin:2.0:generate (default-cli) @
   standalone-pom ---
   [INFO] Generating project in Interactive mode
   [INFO] Archetype [org.codehaus.mojo.archetypes:ejb-javaee6:1.5]
   found in catalog remote
   Define value for property 'groupId': : com.shinysparkly
   Define value for property 'artifactId': : payment-arrangments
   Define value for property 'version':  1.0-SNAPSHOT: :
   Define value for property 'package':  com.shinysparkly: :
   Confirm properties configuration:
   groupId: com.company
   artifactId: payment-arrangments
   version: 1.0-SNAPSHOT
   package: com.company.collections
   Y: :
   [INFO] ------------------------------------------------------------
   --------------
   [INFO] BUILD SUCCESS
   [INFO] ------------------------------------------------------------
   --------------
   [INFO] Total time: 32.440s
   [INFO] Finished at: Mon Oct 31 10:11:12 EST 2011
   [INFO] Final Memory: 7M/81M
   [INFO] ------------------------------------------------------------
   --------------
   [localhost]$
   ```

2. **Ajouter vos beans enterprise**
   Ecrire vos beans enterprise et les ajouter au projet sous le répertoire **`src/main/java`** dans le sous-répertoire approprié pour le paquetage du bean.

3. **Créer votre projet**
   Pour construire ce projet, exécuter la commande **`mvn package`** dans le même répertoire que le fichier **`pom.xml`**. Cela compilera les classes Java et empaquettera le fichier JAR. Le fichier JAR construit se nomme ***`artifactId-version`*`.jar`** et se trouve dans le répertoire **`target/`**.

RÉSULTAT : Vous avez maintenant un projet Maven qui construit et empaquette un fichier JAR. Ce projet peut contenir des beans enterprise et le fichier JAR peut être déployé vers un serveur d'application.

Rapporter un bogue

7.2.3. Créer un projet EAR contenant un projet EJB

Cette tâche décrit comment créer dans Red Hat JBoss Developer Studio un nouveau projet Enterprise Archive (EAR) contenant un projet EJB.

Conditions préalables

- Un serveur et un runtime de serveur pour JBoss EAP 6 a été installé. Veuillez vous référer à Section 1.3.1.5, « Ajouter le serveur de JBoss EAP en utilisant Define New Server ».

Procédure 7.3. Créer un projet EAR contenant un projet EJB

1. **Ouvrir le nouvel Assistant de projet de l'application EAR.**
   Naviguer vers le menu **File**, sélectionner **New**, puis **Project** et l'assistant **`New Projet`** apparaîtra. Sélectionner **Java EE**/**Enterprise Application Project** et cliquer sur le bouton **`Next`**.

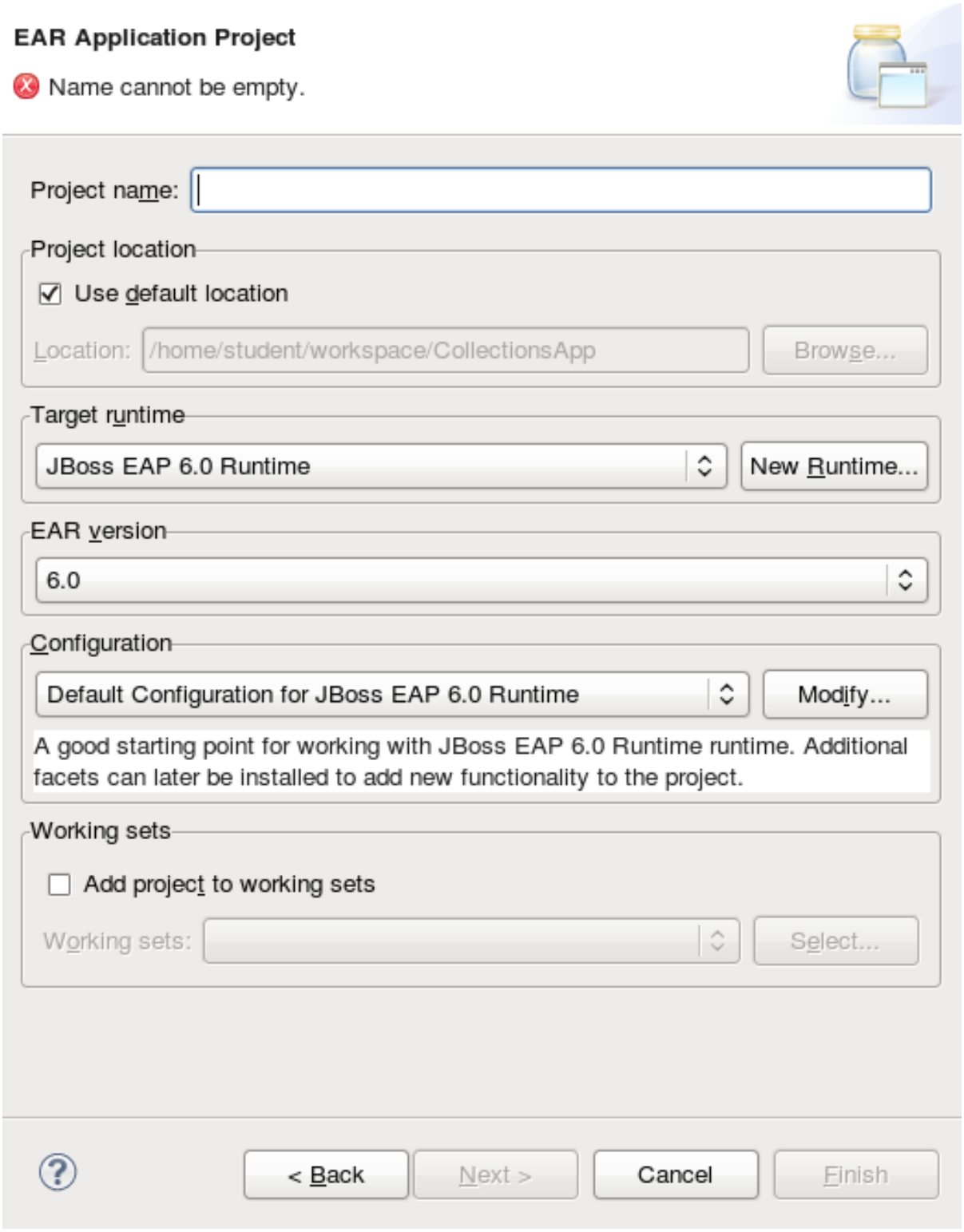


Figure 7.4. Nouvel Assistant de Projet de l'Application EAR.

2. **Fournir des Informations**
   Fournir les informations suivantes :

   - Nom du projet.

     Il s'agira du nom du projet qui apparaît dans Red Hat JBoss Developer Studio, et également le nom du fichier par défaut du fichier EAR déployé.

   - Emplacement du projet

Le répertoire où les fichiers du projet seront sauvegardés. La valeur par défaut est un répertoire qui se trouve dans l'espace de travail en cours.

- Runtime cible.

  Il s'agit du runtime de serveur utilisé pour le projet. Il devra être défini au même runtime JBoss EAP 6 qui est utilisé par le serveur dans lequel vous souhaitez déployer.

- Version EAR.

  C'est la version des spécifications de Java Enterprise Edition auquelles votre projet devra se conformer. Red Hat recommande d'utiliser la version **6**.

- Configuration. Cela vous permet d'ajuster les fonctionnalités prises en charge par votre projet. Utiliser la configuration par défaut pour le runtime que vous avez sélectionné.

Cliquer sur **Suivant** pour continuer.

3. **Ajouter un nouveau module EJB.**
   Les nouveaux modules peuvent être ajoutés à partir de la page **Enterprise Application** de l'assistant. Pour ajouter un nouveau projet EJB en tant que module, veuillez suivre les étapes suivantes :

   a. **Ajouter un nouveau module EJB.**
   Cliquer sur **New Module**, décocher la case **Create Default Modules**, sélectionner **Enterprise Java Bean** et cliquer sur **Next**. L'assistant **New EJB Project** apparaîtra.

   b. **Créer un projet EJB.**
   L'assistant **New EJB Project** est le même que l'assistant utilisé pour créer de nouveaux projets EJB autonomes et est décrit dans Section 7.2.1, « Créer un projet d'archives EJB avec le Red Hat Studio JBoss Developer ».

   Les détails minimales requis pour créer le project sont les suivants :

   - Nom du Projet
   - Runtime de la cible
   - Version Module EJB
   - Configuration

   Toutes les autres étapes de l'assistant sont optionnelles. Cliquer sur **Finish** Terminé) pour finir de créer le projet EJB.

   Le projet EJB tout juste créé est listé dans les dépendances du module Java EE et la case est cochée.

4. **Optionnel : ajouter un descripteur de déploiement d'application.xml**
   Cocher la case **Generate application.xml deployment descriptor** Générer descripteur de déploiement d'application.xmlle cas échéant.

5. **Cliquer sur Finish**
   Deux nouveaux projets vont apparaître, le projet EJB et le projet EAR.

6. **Ajouter l'artifact généré dans le serveur en vue du déploiement**
   Ouvrir la boîte de dialogue **Add and Remove** (Ajouter et Supprimer) en cliquant avec le bouton

droit de la souris, dans l'onglet **Servers** (Serveurs), sur le serveur dans lequel vous souhaitez déployer l'artéfact de la build dans l'onglet serveur, et sélectionner **Add and Remove**.

Sélectionner la ressource EAR à déployer depuis la colonne **Available** (disponible) et cliquer sur le bouton **Add**. La ressource sera déplacée vers la colonne **Configured**. Cliquer sur **Finish** pour fermer la boîte de dialogue.

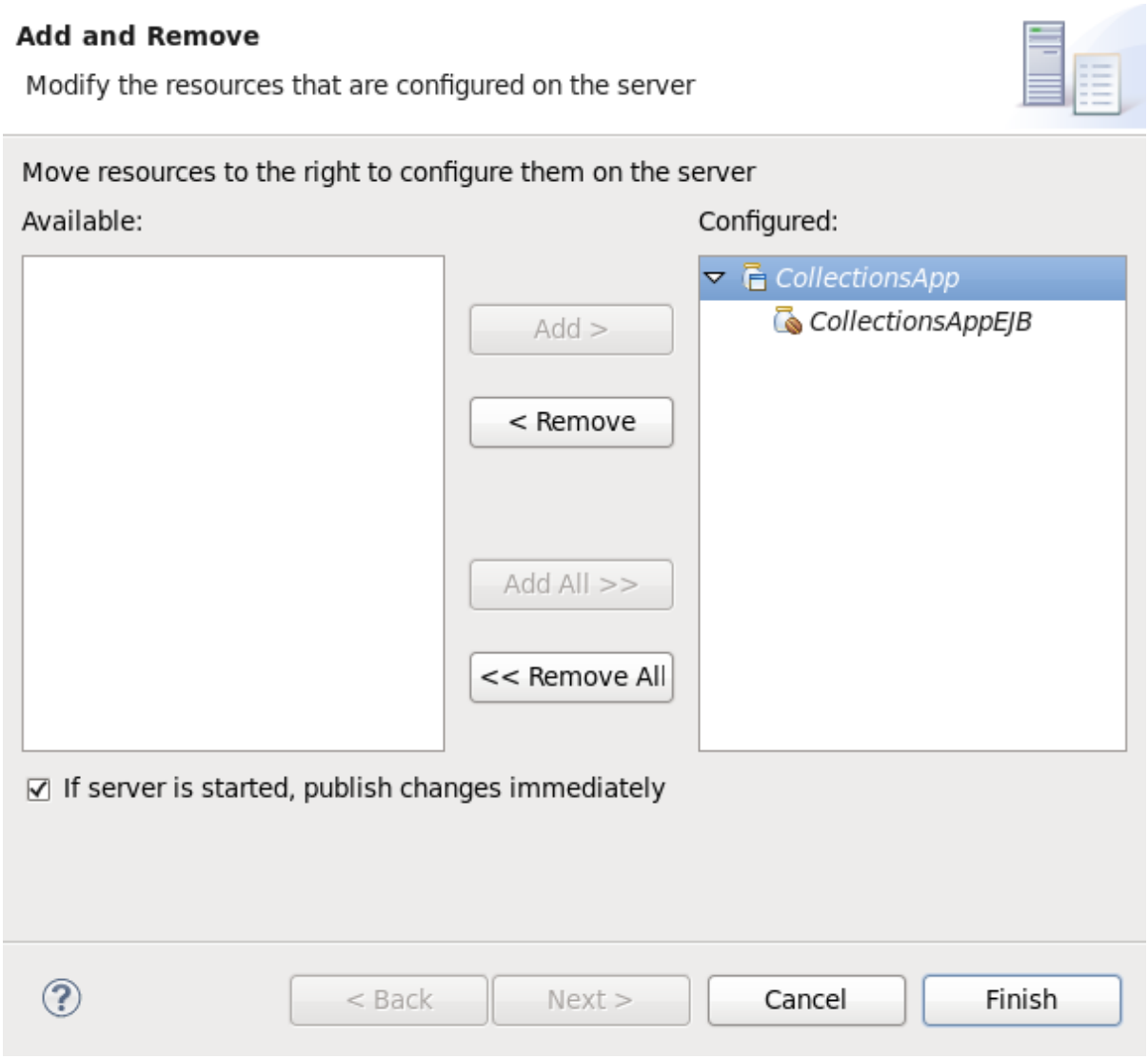


Figure 7.5. Ajouter et Supprimer le dialogue

Résultat

Vous avez maintenant un Projet d'application enterprise avec un Projet EJB membre. Celui-ci se créera et se déploiera vers le serveur indiqué en tant que déploiement EAR unique contenant un sous-déploiement EJB.

Rapporter un bogue

7.2.4. Ajouter un descripteur de déploiement à un projet EJB

On peut ajouter un descripteur de déploiement à un projet EJB qui a été créé sans descripteur. Pour cela, suivre la procédure ci-dessous.

Conditions préalables :

- Vous avez un projet EJB dans Red Hat JBoss Developer Studio auquel vous souhaitez ajouter un descripteur de déploiement EJB.

Procédure 7.4. Ajouter un descripteur de déploiement à un projet EJB.

1. **Ouvrir le projet**
   Ouvrir un projet dans Red Hat JBoss Developer Studio.

2. **Ajouter un descipteur de déploiement**
   Cliquer à droite sur le dossier Descripteur de déploiement dans la vue du projet et sélectionner **Generate Deployment Descriptor Stub** (Générer Descripteur de déploiement).

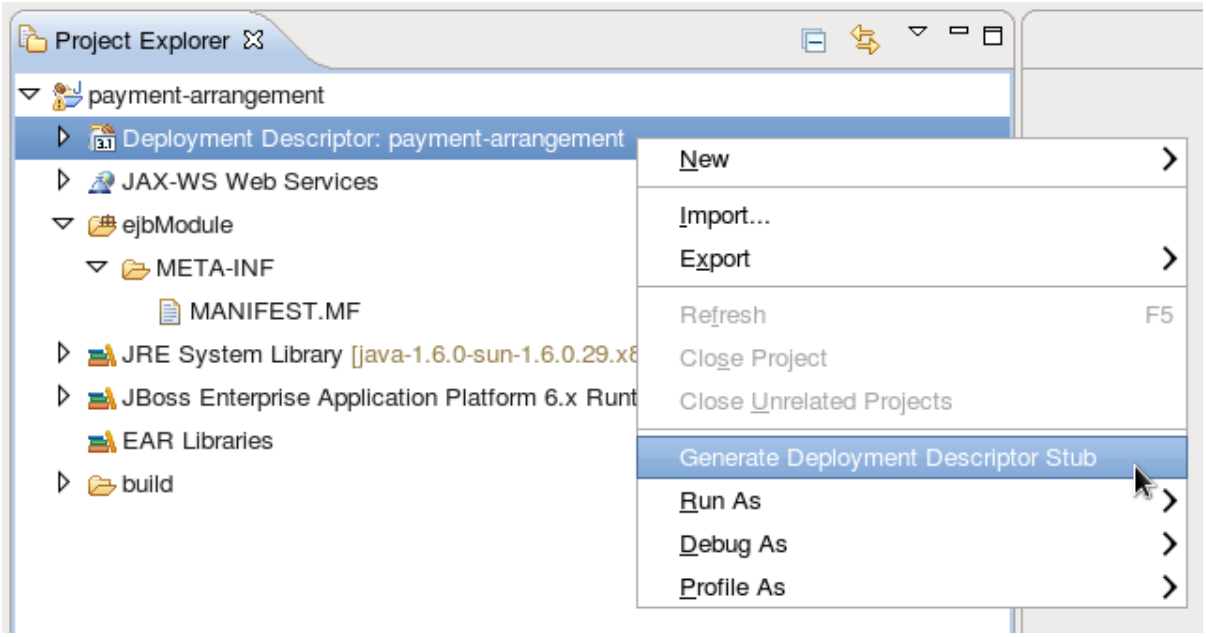


Figure 7.6. Ajouter un descripteur de déploiement

Le nouveau fichier, **`ejb-jar.xml`**, sera créé dans **`ejbModule/META-INF/`**. Cliquer deux fois dans le dossier Descripteur de déploiement dans la vue du projet, ce qui permettra d'ouvrir ce fichier également.

Rapporter un bogue

7.3. SESSION BEANS

7.3.1. Session Beans

Les session beans sont des beans Enterprise qui encapsulent un ensemble de processus métier connexes ou des tâches, et qui sont injectés dans les classes qui en ont fait la demande. Il existe trois types de session beans : sans état, avec état et singleton.

Rapporter un bogue

7.3.2. Stateless Session Beans

Les Stateless Session Beans sont à la fois les types de bean de session les plus simples et les plus largement utilisés. Ils fournissent des méthodes commerciales à des applications client, mais ne maintiennent pas d'état entre les appels de méthode. Chaque méthode est une tâche complète, qui ne

s'appuie pas sur un état partagé au sein de ce bean de session. Parce qu'il n'y a aucun état, le serveur d'application n'est pas tenu de s'assurer que chaque appel de méthode soit effectué sur la même instance. Cela rend les beans de session stateless très efficaces et évolutifs.

Rapporter un bogue

7.3.3. Stateful Session Beans

Les Stateful Session Beans sont des beans Enterprise qui fournissent des modèles commerciaux aux applications client et qui maintiennent un état conversationnel avec le client. Ils doivent être utilisés pour des tâches en plusieurs étapes - appels de méthode - chacun répondant à l'état de l'étape précédente ayant été maintenue. Le serveur d'applications veille à ce que chaque client reçoive la même instance d'un stateful session bean pour chaque appel de méthode.

Rapporter un bogue

7.3.4. Singleton Session Beans

Les Singleton Session Beans sont des beans de session qui sont instanciés une fois par application, et chaque requête de client de Singleton Bean va dans la même instance. Les Singleton Beans représentent une implémentation du Singleton Design Pattern décrit dans l'ouvrage *Design Patterns: Elements of Reusable Object-Oriented Software* d'Erich Gamma, Richard Helm, Ralph Johnson et John Vlissides; publié par Addison-Wesley en 1994.

Les beans Singleton fournissent la plus petite empreinte parmi tous les types de beans de session, mais ils doivent être thread-safe. EJB 3.1 fournit la CMC (de l'anglais Container Managed Concurrency) pour permettre aux développeurs d'implémenter des beans Singleton facilement. Cependant, les beans Singleton peuvent également être écrits grâce à un code multi-thread BMC (de l'anglais Bean Managed Concurrency) si CMC n'offre pas suffisamment de flexibilité.

Rapporter un bogue

7.3.5. Ajouter des Session Beans à un Projet dans Red Hat JBoss Developer Studio

Red Hat JBoss Developer Studio possède plusieurs assistants qui peuvent être utilisés pour créer rapidement des classes bean Enterprise. La procédure suivante vous montre comment utiliser les assistants du Red Hat JBoss Developer Studio pour ajouter un session bean à un projet.

Conditions préalables :

- Vous avec un EJB ou Dynamic Web Project dans Red Hat JBoss Developer Studio auquel vous souhaitez ajouter un ou plusieurs session beans.

Procédure 7.5. Ajouter des Session Beans à un Projet dans Red Hat JBoss Developer Studio

1. **Ouvrir le projet**
   Ouvrir un projet dans Red Hat JBoss Developer Studio.

2. **Ouvrir l'assistant "Create EJB 3.x Session Bean" (Créer EJB 3.x Session Bean)**
   Pour ouvrir l'assistant **`Créer EJB 3.x Session Bean`**, naviguer dans le menu **File**, et sélectionner **New**, puis **Session Bean (EJB 3.x)**.

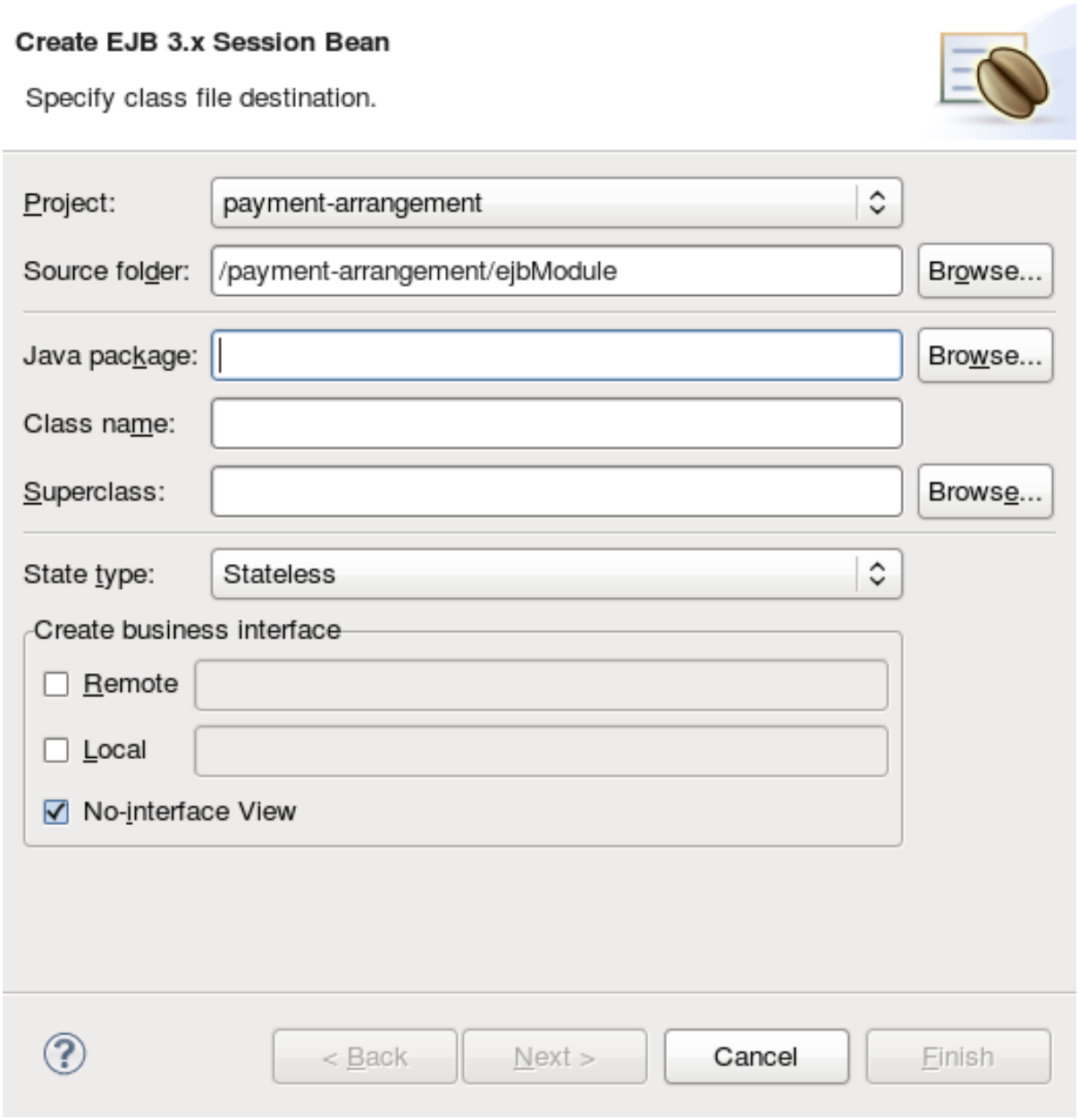


Figure 7.7. Créer un assistant EJB 3.x Session Bean

3. **Indiquer les informations de classe**
   Fournir les informations suivantes :

   - Projet

     Verifier que le projet qui convient a bien été sélectionné.

   - Dossier source

     Il s'agit du dossier dans lequel les fichiers source de Java seront créés. Cela n'a pas normalement besoin d'être changé.

   - Package

     Indiquer le package auquel cette classe appartient.

   - Nom de la classe

     Indiquer le nom de la classe qui représentera le session bean.

- Superclass

  La classe de session bean peut hériter d'une Superclass. Indiquer ici si votre session a une Superclass.

- Type d'état

  Indiquer le type d'état du session bean: stateless, stateful, ou singleton.

- Interfaces commerciales

  Par défaut, la case No-interface est cochée, donc aucune interface ne sera créée. Cocher les cases des interfaces que vous souhaitez définir et ajuster les noms si nécessaire.

  Nous vous rappelons que les beans Enterprise d'un WAR (Web Archive) ne supportent qu'EJB 3.1 Lite, et que cela n'inclut pas les interfaces commerciales distantes.

Cliquer sur **`Next`**.

4. **Informations spécifiques aux Session Bean**
   Vous pouvez saisir des informations supplémentaires ici pour personnaliser encore plus le session bean. Vous n'avez pas besoin de changer quoi que ce soit ici.

   Les items que vous pouvez changer sont :

   - Nom de bean.
   - Nom mappé.
   - Type de transaction (géré conteneur ou géré bean).
   - On peut fournir des interfaces supplémentaires que le bean doit implémenter.
   - Vous pouvez également spécifier des interfaces EJB 2.x Home et Component si nécessaire.

5. **Terminer**
   Cliquer sur le bouton **`Finish`** et le nouveau session bean sera créé, et ajouté au projet. Les fichiers de toute nouvelle interface commerciale seront également créés, s'ils ont été spécifiés.

RÉSULTAT: un nouveau session bean sera ajouté au projet.

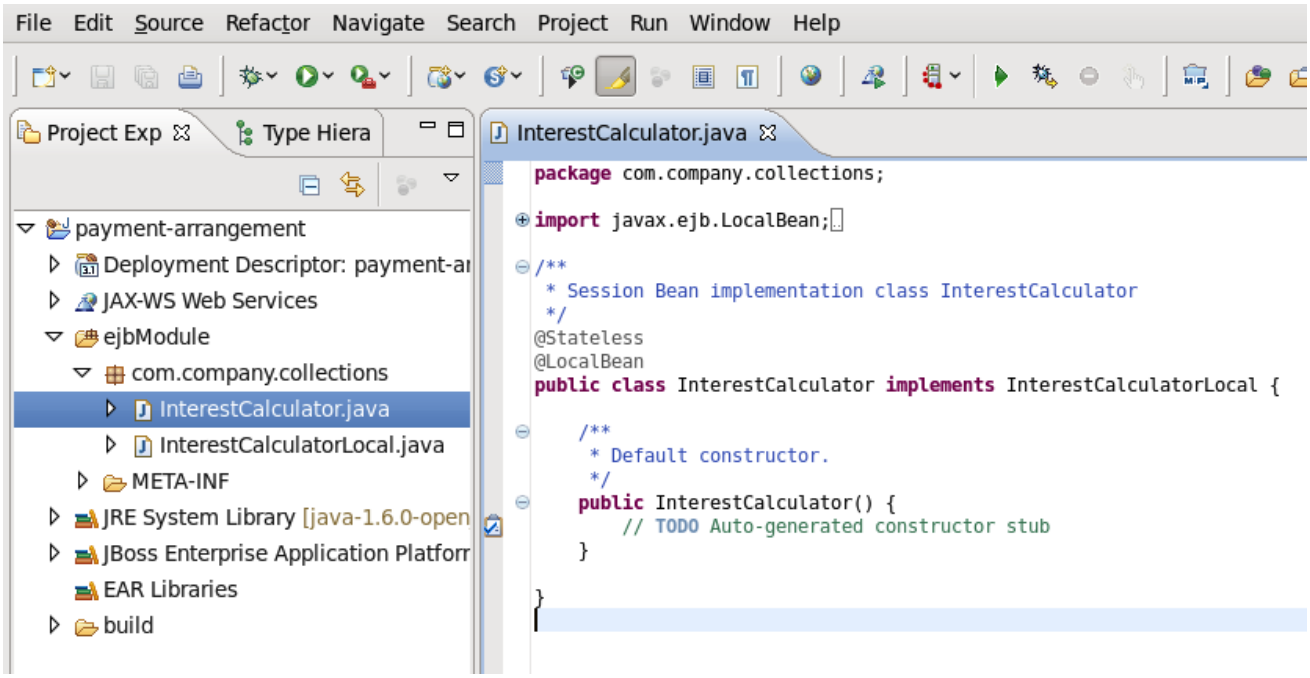


Figure 7.8. Nouveau Session Bean dans Red Hat JBoss Developer Studio

Rapporter un bogue

7.4. MESSAGE-DRIVEN BEANS

7.4.1. Message-Driven Beans

Les Message-driven Beans (MDB) fournissent un modèle basé-événement pour le développement de l'application. Les méthodes des MDB ne sont pas injectées ou invoquées du code client mais sont déclenchées par la réception de messages d'un service de messagerie comme le serveur Java Messaging Service (JMS). La spécification Java EE 6 exige que JMS soit pris en charge, mais les autres systèmes de messagerie peuvent être supportés également.

Rapporter un bogue

7.4.2. Adaptateurs de ressources

Un adaptateur de ressources est un composant Java EE déployable qui permet la communication entre une application Java EE et une entreprise d'informations système (EIE) à l'aide de la spécification Java Connector Architecture (JCA). Un adaptateur de ressources est souvent fourni par les fournisseurs de l'EIS pour permettre une intégration facile de leurs produits aux applications Java EE.

Un système d'information Enterprise peut être n'importe quel autre système de logiciel au sein d'une organisation. Les exemples incluent les systèmes ERP (Enterprise Resource Planning), les systèmes de base de données, les serveurs d'e-mails et les systèmes de messagerie propriétaires.

Un adaptateur de ressources est empaqueté dans un fichier de Ressources Adaptateur Archive (RAR) qui peut être déployé dans JBoss EAP 6. Un fichier RAR peut également être inclus dans un déploiement Enterprise Archive (EAR).

Rapporter un bogue

7.4.3. Créer un Message-Driven Bean basé JMS dans Red Hat JBoss Developer Studio

Cette procédure montre comment ajouter un Message-Driven Bean basé JMS dans Red Hat JBoss Developer Studio. Cette procédure crée un Message-Driven Bean EJB 3.x qui utilise des annotations.

Conditions préalables :

1. Vous devrez avoir un projet existant ouvert dans Red Hat JBoss Developer Studio.
2. Vous devrez connaître le nom et le type de la destination JMS que le bean écoutera.
3. Le support pour JMS (Java Messaging Service) doit être autorisé dans la configuration JBoss EAP 6 dans laquelle ce bean sera déployé.

Procédure 7.6. Ajouter un Message-Driven Bean basé JMS dans Red Hat JBoss Developer Studio

1. **Ouvrir l'assistant `Create EJB 3.x Message-Driven Bean`**
   Aller à **File** → **New** → **Other**. Sélectionner **`EJB/Message-Driven Bean (EJB 3.x)`** et cliquer sur le bouton **`Next`**.

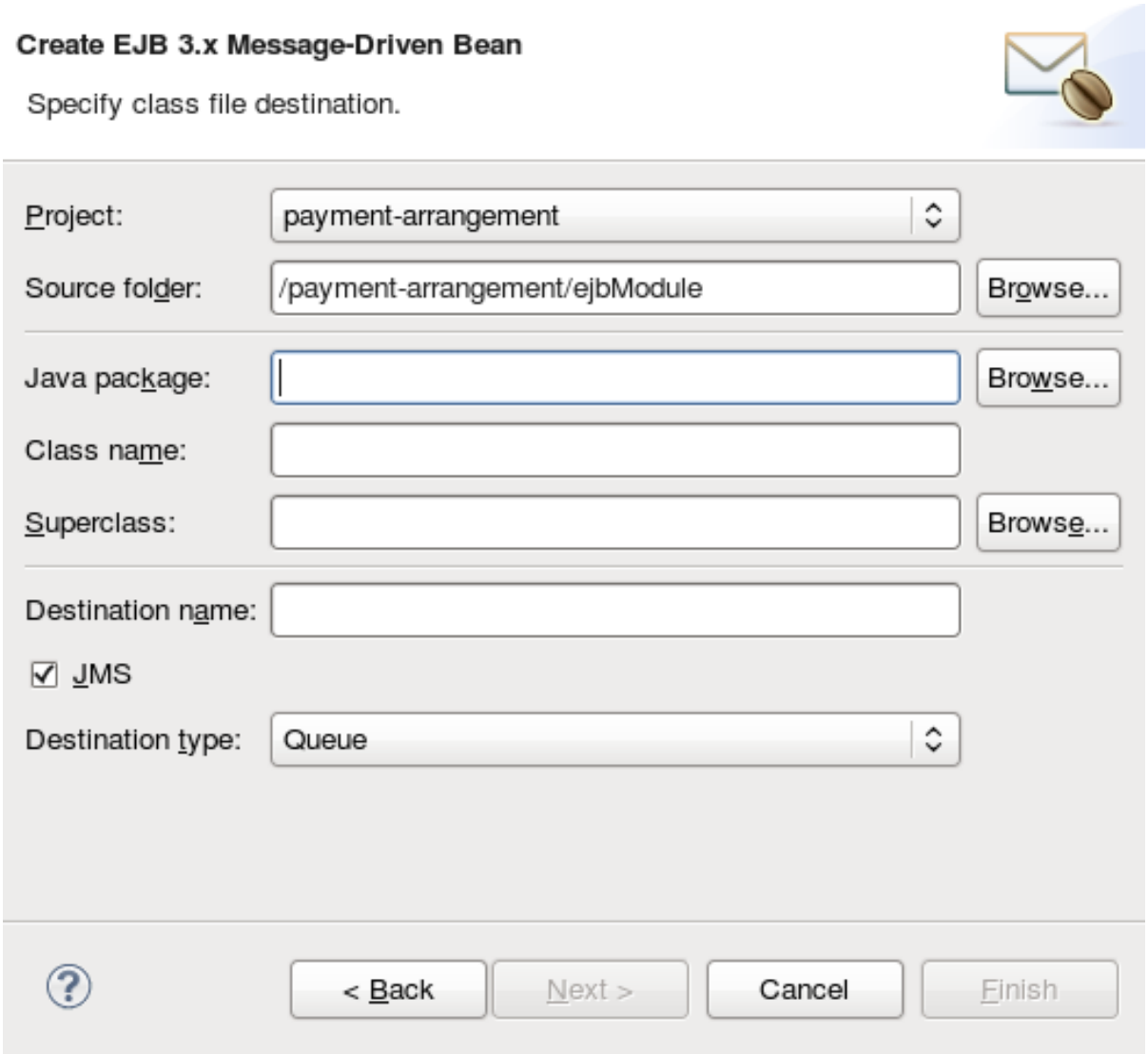


Figure 7.9. Créer l'assistant Message-Driven Bean EJB 3.x

2. **Indiquer les détails de la destination de fichier de classe**
   Il existe trois ensembles de détails à indiquer pour la classe de bean : le projet, la classe Java et le message de destination.

Projet

- Si plusieurs projets existent dans **`Workspace`**, veillez à ce que le bon projet soit sélectionné dans le menu **Project**.
- Le dossier où le fichier source du nouveau bean sera créé est **`ejbModule`** sous le répertoire du projet sélectionné. Ne peut être changé que pour répondre à des cas particuliers.

Classe Java

- Les champs obligatoires sont les suivants : **`Java Package`** et **`class name`**.
- Il n'est pas nécessaire de fournir une **`Superclass`** à moins que la logique commerciale de votre application ne l'exige.

Destination du message

Voici les informations à fournir pour un Message-Driven Bean basé JMS :

- **`Destination name`**. C'est la file d'attente ou le nom du sujet qui contient les messages auxquels le bean répondra.
- La case **`JMS`** est sélectionnée par défaut. Veuillez ne pas la changer.
- Définir **`Destination type`** comme **`Queue`** ou **`Topic`** selon ce que l'on vous demande.

Cliquer sur le bouton **`Next`**.

3. **Saisir les informations spécifiques aux Message-Driven Bean**
   Les valeurs par défaut sont appropriées pour un Message-Driven Bean basé JMS utilisant des transactions gérées conteneur.
   - Changer le type de transaction à Bean si le Bean doit utiliser des transactions Bean-managed (gérées bean).
   - Changer le nom du Bean s'il est demandé un nom de bean différent du nom de classe.
   - L'interface JMS Message Listener sera déjà listée. Vous n'aurez pas besoin d'ajouter ou de supprimer des interfaces, à moins qu'elles soient spécifiques à votre logique commerciale d'applications.
   - Laisser les cases décochées pour créer les méthodes stub sélectionnées.

   Cliquer sur le bouton **`Finish`**.

Résultat: le Bean Message-Driven sera créé avec des méthodes stub pour le contructeur par défaut et la méthode **`onMessage()`**. Une fenêtre d'édition de Red Hat JBoss Developer Studio est apparue avec le fichier correspondant.

Rapporter un bogue

7.4.4. Activer la substitution de propriété des EJB et MDB dans une application

Red Hat JBoss EAP vous permet maintenant d'activer la substitution de propriétés dans les EJB et les MDB à l'aide des annotations **`@ActivationConfigProperty`** et **`@Resource`**. La substitution de propriétés nécessite la configuration et les changements de code suivants :

- Vous devez activer la substitution de propriété dans le fichier de configuration du serveur JBoss EAP.
- Vous devez définir les propriétés système dans le fichier de configuration du serveur ou les passer comme arguments quand vous démarrez le serveur JBoss EAP.
- Vous devez modifier le code pour utiliser les variables de substitution.

Procédure 7.7. Implémenter la substitution de propriétés dans une application MDB

Les exemples de code suivants sont basés sur le guide de démarrage rapide **`helloworld-mdb`** fourni dans JBoss EAP 6.3 ou version supérieure. Cette section vous montre comment modifier le guide de démarrage rapide pour activer la substitution de propriété.

1. **Configurer le serveur JBoss EAP pour activer la substituion de propriété.**
   Le serveur JBoss EAP doit être configuré pour activer la substitution de propriété. Pour cela, définir l'attribut **`<annotation-property-replacement>`** qui se trouve dans le sous-système **`ee`** du fichier de configuration du serveur à **`true`**.

   a. Sauvegarder le fichier de configuration du serveur. L'exemple de guide de démarrage rapide **`helloworld-mdb`** nécessite un profil complet pour les serveurs autonomes, donc il s'agit du **`standalone/configuration/standalone-full.xml`**. Si vous exécutez votre serveur dans un domaine géré, il s'agira du fichier **`domain/configuration/domain.xml`**.

   b. Démarrer le serveur JBos EAP avec le profil complet.

      Dans Linux :

      ```
      EAP_HOME/bin/standalone.sh -c standalone-full.xml
      ```

      Dans Windows :

      ```
      EAP_HOMEbin\standalone.bat -c standalone-full.xml
      ```

   c. Lancer l'interface CLI par la commande pour votre système d'exploitation.

      Dans Linux :

      ```
      EAP_HOME/bin/jboss-cli.sh --connect
      ```

      Dans Windows :

      ```
      EAP_HOME\bin\jboss-cli.bat --connect
      ```

   d. Saisir la commande suivante pour activer la substitution de propriété d'annotation.

      ```
      /subsystem=ee:write-attribute(name=annotation-property-
      replacement,value=true)
      ```

e. Vous devriez voir apparaître le résultat suivant :

```
{"outcome" => "success"}
```

f. Réviser les changements apportés au fichier de configuration du serveur JBoss EAP. Le sous-système **ee** doit maintenant contenir l'XML suivant.

```
<subsystem xmlns="urn:jboss:domain:ee:1.2">
    <spec-descriptor-property-replacement>false</spec-descriptor-
property-replacement>
    <jboss-descriptor-property-replacement>true</jboss-
descriptor-property-replacement>
    <annotation-property-replacement>true</annotation-property-
replacement>
</subsystem>
```

2. **Définir les propriétés système.**
Vous pouvez spécifier les propriétés du système dans le fichier de configuration du serveur ou vous pouvez les passer comme arguments de ligne de commande lorsque vous démarrez le serveur JBoss EAP. Les propriétés système définies dans le fichier de configuration de serveur ont la priorité sur celles passées sur la ligne de commande lorsque vous démarrez le serveur.

   - Vous devez définir les propriétés système dans le fichier de configuration.

     a. Démarrez le serveur JBoss EAP et le Management API comme indiqué dans les étapes précédentes.

     b. Utiliser la syntaxe de commande suivante pour configurer une propriété système dans le serveur JBoss EAP :

     ```
     /system-property=PROPERTY_NAME:add(value=PROPERTY_VALUE)
     ```

     Pour le guide de démarrage rapide **`helloworld-mdb`**, nous configurons les propriétés système suivantes :

     ```
     /system-
     property=property.helloworldmdb.queue:add(value=java:/queue/HE
     LLOWORLDMDBPropQueue)
     /system-
     property=property.helloworldmdb.topic:add(value=java:/topic/HE
     LLOWORLDMDBPropTopic)
     /system-
     property=property.connection.factory:add(value=java:/Connectio
     nFactory)
     ```

     c. Réviser les changements apportés au fichier de configuration du serveur JBoss EAP. Les propriétés système suivantes doivent maintenant apparaître après les **`<extensions>`**.

     ```
     <system-properties>
         <property name="property.helloworldmdb.queue"
     value="java:/queue/HELLOWORLDMDBPropQueue"/>
         <property name="property.helloworldmdb.topic"
     value="java:/topic/HELLOWORLDMDBPropTopic"/>
     ```

```
    <property name="property.connection.factory"
value="java:/ConnectionFactory"/>
</system-properties>
```

- Passez les propriétés système comme arguments de ligne de commande quand vous démarrez le serveur JBoss EAP sous la forme d'un **-D*PROPERTY_NAME=PROPERTY_VALUE***. Ce qui suit est un exemple de la façon dont on passe les arguments dans les propriétés système définies dans l'étape précédente.

```
EAP_HOME/bin/standalone.sh -c standalone-full.xml -
Dproperty.helloworldmdb.queuejava:/queue/HELLOWORLDMDBPropQueue -
Dproperty.helloworldmdb.topic=java:/topic/HELLOWORLDMDBPropTopic
-Dproperty.connection.factory=java:/ConnectionFactory
```

3. **Modifier le code pour utiliser les substitutions de propriétés système.**
Remplacer les valeurs d'annotation **@ActivationConfigProperty** et **@Resource** codées en dur par des substitutions pour les propriétés système nouvellement définies. Voici des exemples sur la façon de changer le guide de démarrage rapide **helloworld-mdb** pour qu'il utilise les substitutions de propriétés système nouvellements définies pour les annotations qui se trouvent dans le code source.

   a. Modifier la valeur de propriété **@ActivationConfigProperty destination** qui se trouve dans la classe **HelloWorldQueueMDB** afin d'utiliser la substitution pour la propriété système. L'annotation **@MessageDriven** devra ressembler à ceci :

```
@MessageDriven(name = "HelloWorldQueueMDB", activationConfig = {
    @ActivationConfigProperty(propertyName = "destinationType",
propertyValue = "javax.jms.Queue"),
    @ActivationConfigProperty(propertyName = "destination",
propertyValue = "${property.helloworldmdb.queue}"),
    @ActivationConfigProperty(propertyName = "acknowledgeMode",
propertyValue = "Auto-acknowledge") })
```

   b. Modifier la valeur de propriété **@ActivationConfigProperty destination** qui se trouve dans la classe **HelloWorldTopicMDB** afin d'utiliser la substitution pour la propriété système. L'annotation **@MessageDriven** devra ressembler à ceci :

```
@MessageDriven(name = "HelloWorldQTopicMDB", activationConfig = {
    @ActivationConfigProperty(propertyName = "destinationType",
propertyValue = "javax.jms.Topic"),
    @ActivationConfigProperty(propertyName = "destination",
propertyValue = "${property.helloworldmdb.topic}"),
    @ActivationConfigProperty(propertyName = "acknowledgeMode",
propertyValue = "Auto-acknowledge") })
```

   c. Modifier les annotations **@Resource** qui se trouvent dans la classe **HelloWorldMDBServletClient** afin d'utiliser les substitutions de propriétés système. Le code devra ressembler à ceci :

```
@Resource(mappedName = "${property.connection.factory}")
private ConnectionFactory connectionFactory;

@Resource(mappedName = "${property.helloworldmdb.queue}")
private Queue queue;
```

```
@Resource(mappedName = "${property.helloworldmdb.topic}")
private Topic topic;
```

d. Modifier le fichier **`hornetq-jms.xml`** pour pouvoir utiliser les valeurs de substitution de propriétés système.

```
<?xml version="1.0" encoding="UTF-8"?>
<messaging-deployment xmlns="urn:jboss:messaging-deployment:1.0">
    <hornetq-server>
        <jms-destinations>
            <jms-queue name="HELLOWORLDMDBQueue">
                <entry name="${property.helloworldmdb.queue}"/>
            </jms-queue>
            <jms-topic name="HELLOWORLDMDBTopic">
                <entry name="${property.helloworldmdb.topic}"/>
            </jms-topic>
        </jms-destinations>
    </hornetq-server>
</messaging-deployment>
```

4. Déployer l'application. L'application utilisera maintenant les valeurs indiquées par les propriétés système comme valeurs de propriété pour **`@Resource`** et **`@ActivationConfigProperty`**.

Rapporter un bogue

7.5. INVOQUER LES SESSION BEANS

7.5.1. Invoquer un session bean à distance avec JNDI

Cette tâche décrit comment ajouter un support à un client distant pour l'invocation de session beans avec JNDI. La tâche assume que le projet est construit avec Maven.

Le Quickstart **`ejb-remote`** contient des projets Maven qui démontrent cette fonctionnalité. Le Quickstart contient des projets à la fois pour le déploiement des session beans et pour le client distant. Les exemples de code ci-dessous sont extraits du projet du client distant.

Cette tâche assume que les session beans n'ont pas besoin d'authentification.

Conditions préalables

Les conditions suivantes doivent être respectées avant de commencer :

- Vous devez déjà avoir un projet Maven créé, prêt à l'utilisation.
- La configuration du référentiel JBoss EAP 6 Maven a déjà été ajoutée.
- Les session beans que vous souhaitez invoquer sont déjà déployés.
- Les session beans déployés implémentent les interfaces commerciales éloignées.
- Les interfaces commerciales éloignées des beans de session sont disponibles sous forme de dépendance de Maven. Si les interfaces commerciales éloignées ne sont disponibles que sous forme de fichier JAR, alors il est recommandé d'ajouter le JAR à votre référentiel Maven comme

un artefact. Reportez-vous à la documentation de Maven pour obtenir des directives **`install:install-file`**, http://maven.apache.org/plugins/maven-install-plugin/usage.html

- Vous aurez besoin de connaître le nom d'hôte et le port JNDI du serveur qui hébergent les session beans.

Pour invoquer un session bean d'un client distant, vous devez tout d'abord configurer le projet correctement.

Procédure 7.8. Ajouter une configuration de projet Maven pour l'invocation à distance des session beans

1. **Ajouter les dépendances de projet utiles**
   Le **`pom.xml`** du projet doit être mis à jour pour pouvoir inclure les dépendances nécessaires.

2. **Ajouter le fichier `jboss-ejb-client.properties`**
   L'API du client JBoss EJB s'attend à trouver un fichier dans le root du projet nommé **`jboss-ejb-client.properties`** qui contient les informations de connexion aux services JNDI. Ajouter ce fichier au répertoire **`src/main/resources/`** de votre projet avec le contenu suivant.

   ```
   # In the following line, set SSL_ENABLED to true for SSL
   remote.connectionprovider.create.options.org.xnio.Options.SSL_ENABLE
   D=false
   remote.connections=default
   # Uncomment the following line to set SSL_STARTTLS to true for SSL
   #
   remote.connection.default.connect.options.org.xnio.Options.SSL_START
   TLS=true
   remote.connection.default.host=localhost
   remote.connection.default.port = 4447
   remote.connection.default.connect.options.org.xnio.Options.SASL_POLI
   CY_NOANONYMOUS=false
   # Add any of the following SASL options if required
   #
   remote.connection.default.connect.options.org.xnio.Options.SASL_POLI
   CY_NOANONYMOUS=false
   #
   remote.connection.default.connect.options.org.xnio.Options.SASL_POLI
   CY_NOPLAINTEXT=false
   #
   remote.connection.default.connect.options.org.xnio.Options.SASL_DISA
   LLOWED_MECHANISMS=JBOSS-LOCAL-USER
   ```

   Changer le nom d'hôte et le port pour qu'ils correspondent au serveur. **`4447`** est le numéro de port par défaut. Pour une connexion sécurisée, définir la ligne **`SSL_ENABLED`** à **`true`** et décommenter la ligne **`SSL_STARTTLS`**. L'interface éloignée du conteneur supporte les connexions sécurisées et non sécurisées en utilisant le même port.

3. **Ajouter des dépendances aux interfaces commerciales à distance.**
   Ajouter les dépendances Maven au **`pom.xml`** aux interfaces commerciales éloignées des session beans.

   ```
   <dependency>
      <groupId>org.jboss.as.quickstarts</groupId>
      <artifactId>jboss-ejb-remote-server-side</artifactId>
   ```

```
    <type>ejb-client</type>
    <version>${project.version}</version>
</dependency>
```

Maintenant que le projet a été configuré correctement, vous pouvez ajouter le code pour accéder et invoquer les session beans.

Procédure 7.9. Obtenez un Bean Proxy par JNDI et invoquez les méthodes du Bean

1. **Exceptions vérifiées par Handle**
   Deux des méthodes utilisées dans le code suivant (**`InitialContext()`** et **`lookup()`**) ont une exception vérifiée du type **`javax.naming.NamingException`**. Ces appels de méthode doivent soit être contenus dans un bloc try/catch qui intercepte **`NamingException`** ou dans une méthode déclarée pour lancer **`NamingException`**. Le Quickstart **`ejb-remote`** utilise la seconde technique.

2. **Créer un contexte JNDI**
   Un objet de contexte JNDI fournit le mécanisme pour demander les ressources dans le serveur. Créer un contexte JNDI avec le code suivant :

   ```
   final Hashtable jndiProperties = new Hashtable();
   jndiProperties.put(Context.URL_PKG_PREFIXES,
   "org.jboss.ejb.client.naming");
   final Context context = new InitialContext(jndiProperties);
   ```

   Les propriétés de connexion du service JNDI sont lues à partir du fichier **`jboss-ejb-client.properties`**.

3. **Utiliser la méthode JNDI Context's lookup() pour obtenir un Bean Proxy**
   Invoquer le méthode **`lookup()`** du bean proxy et lui faire passer le nom JNDI du session bean dont vous avez besoin. Un objet sera renvoyé et il devra correspondre au type de méthode d'interface commerciale qui contient les méthodes que vous souhaitez invoquer.

   ```
   final RemoteCalculator statelessRemoteCalculator =
   (RemoteCalculator) context.lookup(
       "ejb:/jboss-ejb-remote-server-side//CalculatorBean!" +
       RemoteCalculator.class.getName());
   ```

   Les noms de session bean JNDI sont définis par une syntaxe particulière. Pour plus d'informations, consulter Section 7.8.1, « Référence de nommage EJB JNDI » .

4. **Méthodes d'invocation**
   Maintenant que vous avez un objet bean proxy, vous pouvez invoquer n'importe quelle méthode qui contient l'interface commerciale distante.

   ```
   int a = 204;
   int b = 340;
   System.out.println("Adding " + a + " and " + b + " via the remote
   stateless calculator deployed on the server");
   int sum = statelessRemoteCalculator.add(a, b);
   System.out.println("Remote calculator returned sum = " + sum);
   ```

   Le proxy bean transmet la demande d'invocation de méthode au bean de session sur le serveur,

où elle est exécutée. Le résultat est retourné au proxy bean qui, ensuite, le retourne à l'appelant. La communication entre le proxy bean et le bean de session distant est transparente à l'appelant.

Vous devriez maintenant être en mesure de configurer un projet Maven pour soutenir les sessions beans invoquant de session sur un serveur distant et écrire le code d'invocation des méthodes de sessions beans à l'aide d'un proxy bean récupéré du serveur par JNDI.

Rapporter un bogue

7.5.2. Contextes Client EJB

JBoss EAP 6 a introduit l'API client EJB pour gérer les invocations EJB à distance. L'API client EJB JBoss utilise le Contexte Client EJB, qui peut être associé à, et être utilisé par un ou plusieurs threads simultanément. Cela signifie qu'un Contexte Client EJB pourrait contenir n'importe quel nombre de récepteurs EJB. Un récepteur EJB est un composant qui sait communiquer avec un serveur capable de gérer l'invocation EJB.

- Un client à distance, exécuté comme application Java autonome.
- Un client à distance, exécuté dans une autre instance de JBoss EAP 6.

Selon le type de client à distance, d'un point de vue API Client EJB, il pourrait y avoir plus d'un Context Client EJB au sein de la JVM.

Bien que les applications autonomes possèdent généralement un Contexte Client EJB unique pouvant être sauvegardé par n'importe quel nombre de récepteurs EJB, cela n'est pas obligatoire. Si une application autonome possède plus d'un Contexte Client EJB, un sélecteur de Contexte Client EJB sera chargé de retourner le contexte approprié.

Dans le cas où des clients à distance s'exécutent dans une autre instance de JBoss EAP 6, chaque application déployée aura un contexte client EJB correspondant. A chaque fois que cette application invoque un autre EJB, le contexte client EJB correspondant est utilisé pour trouver le récepteur EJB correct, qui gère ensuite l'invocation.

Rapporter un bogue

7.5.3. Considérations lors de l'utilisation d'un Contexte EJB Unique

Résumé

Vous devez considérer les conditions préalables de votre application lors de l'utilisation d'un contexte client EJB unique avec des clients à distance autonomes. Pour plus d'informations sur les différents types de clients à distance, veuillez consulter : Section 7.5.2, « Contextes Client EJB » .

Procédure typique pour un Client Autonome à distance avec un Contexte Client EJB unique.

Un client autonome à distance possède généralement un seul contexte client EJB sauvegardé par n'importe quel nombre de récepteurs EJB. Voici un exemple d'application client à distance autonome :

```
public class MyApplication {
    public static void main(String args[]) {
        final javax.naming.Context ctxOne = new
javax.naming.InitialContext();
        final MyBeanInterface beanOne =
ctxOne.lookup("ejb:app/module/distinct/bean!interface");
        beanOne.doSomething();
```

```
        ...
    }
}
```

Les recherches JNDI de client à distance sont généralement sauvegardées par un fichier **`jboss-ejb-client.properties`** utilisé pour installer le contexte client EJB et les récepteurs EJB. Cette configuration comprend également les informations de sécurité, qui sont ensuite utilisées pour créer le récepteur EJB qui relie le serveur JBoss EAP 6. Quand le code ci-dessus est invoqué, l'API client EJB recherche le contexte client EJB, ensuite utilisé pour sélectionner le récepteur EJB qui recevra et traitera la requête d'invocation EJB. Dans ce cas, il n'y a que le contexte client EJB unique, donc ce contexte sera utilisé par le code ci-dessus pour invoquer le bean. La procédure pour invoquer un bean de session à distance en utilisant JNDI est décrite plus en détails ici : Section 7.5.1, « Invoquer un session bean à distance avec JNDI » .

Client autonome à distance nécessitant différentes informations

Une application d'utilisateur pourrait vouloir invoquer un bean plus d'une fois, mais se connecter au serveur JBoss EAp 6 en utilisant différentes informations de sécurité. Voici un exemple d'une application client à distance autonome qui invoque deux fois le même bean :

```
public class MyApplication {
    public static void main(String args[]) {
        // Use the "foo" security credential connect to the server and
invoke this bean instance
        final javax.naming.Context ctxOne = new
javax.naming.InitialContext();
        final MyBeanInterface beanOne =
ctxOne.lookup("ejb:app/module/distinct/bean!interface");
        beanOne.doSomething();
        ...

        // Use the "bar" security credential to connect to the server and
invoke this bean instance
        final javax.naming.Context ctxTwo = new
javax.naming.InitialContext();
        final MyBeanInterface beanTwo =
ctxTwo.lookup("ejb:app/module/distinct/bean!interface");
        beanTwo.doSomething();
        ...
    }
}
```

Dans ce cas, l'application veut se connecter à la même instance de serveur pour invoquer l'EJB hébergé sur ce serveur, mais veut utiliser deux informations différentes pour se connecter au serveur. Puisque l'application du client possède un contexte de client EJB unique, qui peut avoir seulement un récepteur EJB pour chaque instance de serveur, cela signifie que le code ci-dessus n'utilise qu'une seule information pour se connecter au serveur et que le code ne s'exécute pas comme il le devrait.

Solution

Les contextes client EJB scoped offrent une solution à ce problème. Ils offrent un plus grand contrôle sur les contextes client EJB et leurs contextes JNDI associés, qui sont généralement utilisés pour les invocations EJB. Pour plus d'informations sur les contextes client EJB scoped, veuillez consulter Section 7.5.4, « Utiliser des Contextes Client EJB scoped » et Section 7.5.5, « Configurer les EJB en utilisant un contexte client EJB scoped » .

Rapporter un bogue

7.5.4. Utiliser des Contextes Client EJB scoped

Résumé

Pour invoquer un EJB dans les versions antérieures de JBoss EAP 6, il faut généralement créer un contexte JNDI et lui passer le PROVIDER_URL, qui indique le serveur de cible. Toute invocation faite sur les proxys EJB recherchées en utilisant ce contexte JNDI finiraient sur ce serveur. Dans les contextes client EJB scoped, les applications utilisateur contrôlent quel récepteur EJB est utilisé pour une invocation spécifique.

Utiliser le Contexte Client EJB scoped dans un client autonome à distance

Avant l'introduction des contextes client EJB étendus, le contexte était généralement étendu à l'application client. Les contextes client étendus autorisent maintenant les contextes client EJB à être étendus avec les contextes JNDI. Voici un exemple d'une application client autonome à distance qui invoque le même bean deux fois en utilisant un contexte client EJB scoped.

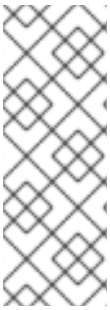
```
public class MyApplication {
    public static void main(String args[]) {

        // Use the "foo" security credential connect to the server and
invoke this bean instance
        final Properties ejbClientContextPropsOne =
getPropsForEJBClientContextOne():
        final javax.naming.Context ctxOne = new
javax.naming.InitialContext(ejbClientContextPropsOne);
        final MyBeanInterface beanOne =
ctxOne.lookup("ejb:app/module/distinct/bean!interface");
        beanOne.doSomething();
        ...
        ctxOne.close();

        // Use the "bar" security credential to connect to the server and
invoke this bean instance
        final Properties ejbClientContextPropsTwo =
getPropsForEJBClientContextTwo():
        final javax.naming.Context ctxTwo = new
javax.naming.InitialContext(ejbClientContextPropsTwo);
        final MyBeanInterface beanTwo =
ctxTwo.lookup("ejb:app/module/distinct/bean!interface");
        beanTwo.doSomething();
        ...
        ctxTwo.close();
    }
}
```

Pour utiliser le contexte client EJB scoped, configurer les propriétés client EJB par programmation et passer les propriétés sur la création de contexte. Les propriétés correspondent au même ensemble de propriétés que celles qui sont utilisées dans le fichier standard **`jboss-ejb-client.properties`**. Pour étendre le contexte client EJB au contexte JNDI, il faut également indiquer la propriété **`org.jboss.ejb.client.scoped.context`** et définir sa valeur sur **`true`**. Cette propriété notifie à l'API client EJB qu'il doit créer un contexte client EJB, sauvegardé par des récepteurs EJB, et que le contexte créé doit ensuite être étendu ou visible uniquement au contexte JNDI qui l'a créé. Tout proxy

EJB recherché ou invoqué en utilisant ce contexte JNDI ne connaîtra que le contexte client EJB associé à ce contexte JNDI. Les autres contextes JNDI utilisés par l'application pour rechercher ou invoquer des EJB ne reconnaîtra pas les autres contextes client EJB scoped.

Les contextes JNDI qui ne passent pas les propriétés **`org.jboss.ejb.client.scoped.context`** et ne sont pas étendus vers un contexte client EJB, utiliseront le comportement par défaut, qui utilise le contexte client EJB existant normalement attaché à l'application dans sa totalité.

Les contextes client EJB scoped donnent aux applications d'utilisateur la même souplesse qui caractérisait les invocations JNDI basées JNP dans les versions précédentes de JBoss EAP 6. Ils donnent aux applications utilisateurs un plus grand contrôle sur le contexte JNDI communique avec un serveur et sur comment il se connecte à ce serveur.



NOTE

Dans le contexte scoped, les ressources sous-jacentes ne sont plus gérées par le conteneur ou l'API, donc vous devez fermer l'**`InitialContext`** lorsqu'il n'est plus utile. Lorsque l'**`InitialContext`** est fermé, les ressources sont libérées immédiatement. Les proxys qui y sont liées ne sont plus valides et tout invocation soulèvera une exception. Une mauvaise fermeture du **`InitialContext`** peut entraîner les problèmes de ressource ou de performance.

Rapporter un bogue

7.5.5. Configurer les EJB en utilisant un contexte client EJB scoped

Résumé

Les EJB peuvent être configurés en utilisant un contexte scoped basé mappe. Cela est possible en complétant une mappe **`Properties`** par programmation en utilisant les propriétés standards trouvées dans **`jboss-ejb-client.properties`**, en indiquant **`true`** pour la propriété **`org.jboss.ejb.client.scoped.context`**, et en passant les propriétés sur la création **`InitialContext`**.

L'avantage d'utiliser un contexte scoped est qu'il vous permet de configurer l'accès sans référencer directement l'EJB ou importer les classes JBoss. Il permet également de configurer et de répartir la charge d'un hôte dans un environnement multithreadé lors du runtime.

Procédure 7.10. Configurer un EJB en utilisant un contexte scoped basé mappage

1. **Configurer les propriétés**
   Configurer les propriétés client EJB par programmation, en indiquant le même ensemble de propriétés utilisées dans le fichier **`jboss-ejb-client.properties`** standard. Pour activer le contexte scoped, vous devez indiquer la propriété **`org.jboss.ejb.client.scoped.context`** et configurer sa valeur sur **`true`**. Voici un exemple de configuration des propriétés par programmation :

   ```
   // Configure  EJB Client properties for the InitialContext
   Properties ejbClientContextProps = new Properties();
   ejbClientContextProps.put(“remote.connections”,”name1”);
   ejbClientContextProps.put(“remote.connection.name1.host”,”localhost”
   );
   ejbClientContextProps.put(“remote.connection.name1.port”,”4447”);
   // Property to enable scoped EJB client context which will be tied
   ```

```
to the JNDI context
ejbClientContextProps.put("org.jboss.ejb.client.scoped.context",
“true”);
```

2. **Passer les propriétés en création de contexte**

```
// Create the context using the configured properties
InitialContext ic = new InitialContext(ejbClientContextProps);
MySLSB bean = ic.lookup("ejb:myapp/ejb//MySLSBBean!" +
MySLSB.class.getName());
```

Informations supplémentaires

- Les contextes générés par les proxys EJB de recherche sont liés par ce contexte étendu et n'utilisent que les paramètres de connexion appropriés. Cela permet de créer différents contextes pour accéder aux données au sein d'une application client ou pour accéder aux serveurs de manière indépendante en utilisant différents logins.
- Dans le client, le **`InitialContext`** étendu et le proxy étendu sont tous les deux transférés vers les threads, permettant à chaque thread de fonctionner avec le contexte donné. Il est également possible de transférer le proxy vers des threads multiples qui peuvent l'utiliser simultanément.
- Le contexte étendu proxy EJB est sérialisé sur l'appel distant, puis désérialisé sur le serveur. Lorsqu'il est désérialisé, les informations de contexte étendu sont supprimées et il retourne à son état par défaut. Si le proxy désérialisé est utilisé sur le serveur distant, c-a-d s'il n'a plus le contexte étendu utilisé lors de sa création, cela peut provoquer une erreur **`EJBCLIENT000025`** ou éventuellement appeler une cible indésirable en utilisant le nom de l'EJB.

Rapporter un bogue

7.5.6. Propriétés Client EJB

Résumé

Les tableaux suivants énumèrent les propriétés pouvant être configurées par programmation ou dans le fichier **`jboss-ejb-client.properties`**.

Propriétés Globales de Client EJB

Le tableau suivant énumère les propriétés valides pour la bibliothèque entière au sein de la même étendue.

Tableau 7.1. Propriétés globales

Nom de propriété	Description
`endpoint.name`	Nom du point de terminaison du client. Si pas configuré, sa valeur par défaut sera **`client-endpoint`**. Cela peut s'avérer utile pour distinguer différentes configurations de point de terminaison puisque le nom de thread comprend cette propriété.

Nom de propriété	Description
remote.connectionprovider.create.options.org.xnio.Options.SSL_ENABLED	Valeur booléenne qui indique si le protocole SSL est activé pour toutes les connexions.
deployment.node.selector	Nom complet de l'implémentation de **org.jboss.ejb.client.DeploymentNodeSelector**. Ceci est utilisé pour équilibrer les charges de l'invocation des EJB.
invocation.timeout	Délai pour la liaison EJB ou le cycle requête/réponse de l'invocation de la méthode. Valeur exprimée en millisecondes. L'invocation de toute méthode lève une **java.util.concurrent.TimeoutException** si l'exécution prend plus de temps que la période de délai. L'exécution se termine et le serveur n'est pas interrompu.
reconnect.tasks.timeout	Délai des tâches de reconnexion de l'arrière-plan. Valeur exprimée en millisecondes. Si des connexions sont éteintes, la prochaine invocation EJB de client utilisera un algorithme pour décider si une reconnexion est nécessaire pour trouver le bon nœud.
org.jboss.ejb.client.scoped.context	Valeur booléenne indiquant s'il faut activer le contexte client EJB scoped. La valeur par défaut est **false**. Si défini comme **true**, le client EJB utilisera le contexte étendu (scoped) lié au contexte JNDI. Sinon, le contexte de client EJB utilisera la sélecteur global dans le JVM pour déterminer les propriétés utilisées pour appeler l'hôte et l'EJB à distance.

Propriétés de connexion de client EJB

Les propriétés de connexion commencent par le préfixe **remote.connection.*CONNECTION_NAME*** où le *CONNECTION_NAME* est un identifiant local utilisé uniquement pour identifier la connexion de manière exclusive.

Tableau 7.2. Propriétés de connexion

Nom de propriété	Description
remote.connections	Liste de **connection-names** actifs séparés par des virgules. Chaque connexion est configurée sous ce nom.
remote.connection.*CONNECTION_NAME*.host	Nom d'hôte ou IP de cette connexion

Nom de propriété	Description
remote.connection.*CONNECTION_NAME*.port	Port de la connexion. La valeur par défaut est 4447.
remote.connection.*CONNECTION_NAME*.username	Nom d'utilisateur utilisé pour authentifier la sécurité de connexion.
remote.connection.*CONNECTION_NAME*.password	Mot de passe utilisé pour authentifier l'utilisateur.
remote.connection.*CONNECTION_NAME*.connect.timeout	Période de délai de la connexion initiale. Après cela, la tâche de reconnexion vérifiera périodiquement si la connexion peut être établie. Valeur exprimée en millisecondes.
remote.connection.*CONNECTION_NAME*.callback.handler.class	Nom complet de la classe **CallbackHandler**. Sera utilisé pour établir la connexion et ne peut être changé tant que la connexion sera ouverte.
remote.connection.*CONNECTION_NAME*. **channel.options.org.jboss.remoting3.RemotingOptions.MAX_OUTBOUND_MESSAGES**	Valeur entière indiquant le nombre maximale de requêtes sortantes. La valeur par défaut est 80. Il n'y a qu'une connexion entre le client (JVM) et le serveur pour gérer les invocations.
remote.connection.*CONNECTION_NAME*. **connect.options.org.xnio.Options.SASL_POLICY_NOANONYMOUS**	Valeur booléenne déterminant si les identifiants doivent être fournis par le client pour réussir la connexion. La valeur par défaut est **true**. Si défini comme **true**, le client doit fournir les identifiants. Si défini comme **false**, l'invocation est autorisée tant que le connecteur à distance ne demande pas un domaine de sécurité.
remote.connection.*CONNECTION_NAME*. **connect.options.org.xnio.Options.SASL_DISALLOWED_MECHANISMS**	Désactive certains mécanismes SASL utilisés pour l'authentification pendant la création de la connexion. **JBOSS_LOCAL_USER** signifie que le mécanisme d'authentification silencieuse, utilisé lorsque le client et le serveur sont sur la même machine, est désactivé.

Nom de propriété	Description
remote.connection.*CONNECTION_NAME*.connect.options.org.xnio.Options.SASL_POLICY_NOPLAINTEXT	Valeur booléenne qui active ou désactive l'utilisation de messages de texte brut pendant l'authentification. Si JAAS est utilisé, il doit être défini comme « false » pour autoriser un mot de passe de texte brut.
remote.connection.*CONNECTION_NAME*.connect.options.org.xnio.Options.SSL_ENABLED	Valeur booléenne indiquant si le protocole SSL est activé pour cette connexion.
remote.connection.*CONNECTION_NAME*.connect.options.org.jboss.remoting3.RemotingOptions.HEARTBEAT_INTERVAL	Intervalle pour envoyer une pulsation entre le client et le serveur afin de prévenir les fermetures automatiques, dans le cas d'un pare-feu par exemple. Valeur exprimée en millisecondes.

Propriétés de cluster de client EJB

Si la connexion initiale se connecte à un environnement clusterisé, la topologie du cluster est reçue de manière automatique et asynchrone. Ces propriétés sont utilisées pour se connecter à chaque membre reçu. Chaque propriété commence avec le préfixe **remote.cluster.*CLUSTER_NAME*** où le *CLUSTER_NAME* se réfère à la configuration du sous-système Infinispan du serveur associé.

Tableau 7.3. Propriétés de Cluster

Nom de propriété	Description
remote.cluster.*CLUSTER_NAME*.clusternode.selector	Le nom complet de l'implémentation de **org.jboss.ejb.client.ClusterNodeSelector**. Cette classe, plutôt que **org.jboss.ejb.clientDeploymentNodeSelector**, est utilisée pour équilibrer la charge des invocations EJB dans un environnement clusterisé. Si le cluster est complètement arrêté, l'invocation échouera avec comme message **No ejb receiver available**.

Nom de propriété	Description

`remote.cluster.CLUSTER_NAME.`
 	
`channel.options.org.jboss.remoting3.RemotingOptions.MAX_OUTBOUND_MESSAGES`
	Valeur entière indiquant le nombre maximale de requêtes sortantes pouvant être faites au cluster entier.

`remote.cluster.CLUSTER_NAME.`
 	
`node.NODE_NAME.channel.options.org.jboss.remoting3.RemotingOptions.MAX_OUTBOUND_MESSAGES`
	Valeur entière indiquant le nombre maximale de requêtes sortantes pouvant être faites à ce nœud de cluster spécifique.

Rapporter un bogue

7.6. INTERCEPTEURS DE CONTENEURS

7.6.1. Intercepteurs de conteneurs

Les intercepteurs standard Java EE, définis dans la spécification JSR 318, Enterprise JavaBeans 3.1 sont sensés exécuter une fois que le conteneur a complété la propagation de contexte de sécurité, la gestion des transactions, et que d'autres conteneurs ont fourni le traitement de l'invocation. C'est un problème si l'application doit intercepter un appel avant qu'un intercepteur spécifique de conteneur soit exécuté.

Les versions antérieures de JBoss EAP 6.0 fournissent un moyen de connecter des intercepteurs côté serveur dans le flux de l'invocation, donc vous pourriez exécuter la logique de l'application spécifique avant que le conteneur ne termine le traitement de l'invocation. Cette fonctionnalité existait déjà dans JBoss EAP 6.1. Cela permet aux intercepteurs standards de Java EE d'être utilisés comme des intercepteurs de conteneur, ce qui signifie qu'ils utilisent les mêmes éléments XSD que ceux autorisés dans le fichier `ejb-jar.xml` pour la version 3.1 du descripteur de déploiement ejb-jar.

Positionnement de l'intercepteur du conteneur dans la chaîne d'intercepteur

Les intercepteurs de conteneur configurés pour un EJB sont certains d'être exécutés avant que le JBoss EAP ne fournisse des intercepteurs de sécurité, des intercepteurs de gestion de transaction ou autres intercepteurs fournis par le serveur. Cela permet aux intercepteurs de conteneurs spécifiques à l'application de traiter ou de configurer des données de contexte pertinentes avant l'invocation.

Différences entre l'intercepteur de conteneur et l'API Java EE Interceptor

Bien que les intercepteurs de conteneur soient modélisés pour pouvoir ressembler aux intercepteurs de Java EE, il y a quelques différences dans la sémantique de l'API. Par exemple, il est illégal pour les intercepteurs de conteneur d'invoquer la méthode `javax.interceptor.InvocationContext.getTarget()` parce que ces intercepteurs sont invoqués bien avant que les composants EJB ne soient configurés ou instanciés.

Rapporter un bogue

7.6.2. Créer une classe d'intercepteur de conteneur

Résumé

Les classes d'intercepteur de conteneur sont de simples POJO (Plain Old Java Objects). Ils utilisent **`@javax.annotation.AroundInvoke`** pour marquer la méthode qui est invoquée lors de l'invocation du bean.

Voici un exemple de classe d'intercepteur de conteneur qui marque la méthode **`iAmAround`** de l'invocation :

Exemple 7.1. Exemple de classe d'intercepteur de conteneur

```
public class ClassLevelContainerInterceptor {
    @AroundInvoke
    private Object iAmAround(final InvocationContext invocationContext)
throws Exception {
        return this.getClass().getName() + " " +
invocationContext.proceed();
    }
}
```

Pour obtenir un exemple de fichier de descripteur d'intercepteur de conteneur configuré pour utiliser cette classe, voir le fichier **`jboss-ejb3.xml`** décrit ici : Section 7.6.3, « Configurer un intercepteur de conteneur ».

Rapporter un bogue

7.6.3. Configurer un intercepteur de conteneur

Résumé

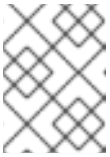
Les intercepteurs de conteneurs utilisent les bibliothèques d'intercepteur J2EE standard, ce qui signifie qu'ils utilisent les mêmes éléments XSD que ceux qui sont autorisés dans le fichier **`ejb-jar.xml`** pour la version 3.1 du descripteur de déploiement ejb-jar. Comme ils se reposent sur les bibliothèques standard d'interceptor Jave EE, les intercepteurs de conteneur peuvent uniquement être configurés à l'aide de descripteurs de déploiement. Cela a été conçu pour que les applications n'exigent pas une annotation spécifique de JBoss ou autre dépendance de bibliothèque. Pour plus d'informations sur les intercepteurs de conteneur, voir : Section 7.6.1, « Intercepteurs de conteneurs ».

Procédure 7.11. Créer le fichier de descripteur pour configurer l'intercepteur de conteneur

1. Créer un fichier **`jboss-ejb3.xml`** dans le répertoire **`META-INF`** du déploiement EJB.
2. Configurer les éléments de l'intercepteur du conteneur dans le fichier du descripteur.
    a. Utiliser l'espace-nom **`urn:container-interceptors:1.0`** pour indiquer la configuration des éléments de l'intercepteur du conteneur.
    b. Utiliser l'élément **`<container-interceptors>`** pour indiquer les intercepteurs du conteneur.

c. Utiliser les éléments **<interceptor-binding>** pour relier l'intercepteur du conteneur aux EJB. Les intercepteurs peuvent être reliés d'une des manières suivantes :

- Relier l'intercepteur à tous les EJB du déploiement par le caractère générique *****.
- Relier l'intercepteur au niveau bean individuel par le nom spécifique de l'EJB.
- Relier l'intercepteur au niveau de méthode spécifique des EJB.



NOTE

Ces éléments sont configurés par EJB 3.1 XSD de la même façon que pour les intercepteurs Java EE.

3. Vérifier les exemples d'éléments ci-dessus dans le fichier de descripteur suivant.

Exemple 7.2. jboss-ejb3.xml

```
<jboss xmlns="http://www.jboss.com/xml/ns/javaee"
       xmlns:jee="http://java.sun.com/xml/ns/javaee"
       xmlns:ci ="urn:container-interceptors:1.0">

    <jee:assembly-descriptor>
        <ci:container-interceptors>
            <!-- Default interceptor -->
            <jee:interceptor-binding>
                <ejb-name>*</ejb-name>
                <interceptor-
class>org.jboss.as.test.integration.ejb.container.interceptor.Cont
ainerInterceptorOne</interceptor-class>
            </jee:interceptor-binding>
            <!-- Class level container-interceptor -->
            <jee:interceptor-binding>
                <ejb-name>AnotherFlowTrackingBean</ejb-name>
                <interceptor-
class>org.jboss.as.test.integration.ejb.container.interceptor.Clas
sLevelContainerInterceptor</interceptor-class>
            </jee:interceptor-binding>
            <!-- Method specific container-interceptor -->
            <jee:interceptor-binding>
                <ejb-name>AnotherFlowTrackingBean</ejb-name>
                <interceptor-
class>org.jboss.as.test.integration.ejb.container.interceptor.Meth
odSpecificContainerInterceptor</interceptor-class>
                <method>
                    <method-
name>echoWithMethodSpecificContainerInterceptor</method-name>
                </method>
            </jee:interceptor-binding>
            <!-- container interceptors in a specific order -->
            <jee:interceptor-binding>
                <ejb-name>AnotherFlowTrackingBean</ejb-name>
                <interceptor-order>
                    <interceptor-
class>org.jboss.as.test.integration.ejb.container.interceptor.Clas
```

```
sLevelContainerInterceptor</interceptor-class>
                    <interceptor-
class>org.jboss.as.test.integration.ejb.container.interceptor.Meth
odSpecificContainerInterceptor</interceptor-class>
                    <interceptor-
class>org.jboss.as.test.integration.ejb.container.interceptor.Cont
ainerInterceptorOne</interceptor-class>
                </interceptor-order>
                <method>
                    <method-
name>echoInSpecificOrderOfContainerInterceptors</method-name>
                </method>
            </jee:interceptor-binding>
        </ci:container-interceptors>
    </jee:assembly-descriptor>
</jboss>
```

L'XSD de l'espace-nom **`urn:container-interceptors:1.0`** se trouve dans ***`EAP_HOME`*`/docs/schema/jboss-ejb-container-interceptors_1_0.xsd`**.

Rapporter un bogue

7.6.4. Modifier l'identité du contexte de sécurité

Résumé

Par défaut, lorsque vous effectuez un appel distant à un EJB déployé sur le serveur d'applications, la connexion au serveur est authentifiée et toute demande reçue via cette connexion sera exécutée en tant qu'identité ayant authentifié la connexion. Cela est vrai pour les appels client-serveur et serveur-serveur. Si vous avez besoin d'utiliser différentes identités en provenance d'un même client, vous devrez normalement ouvrir des connexions multiples au serveur afin que chacune d'entre elles soient authentifiée en tant qu'identité différente. Plutôt que d'ouvrir plusieurs connexions de client, vous pouvez donner l'autorisation à l'utilisateur authentifié d'exécuter une requête sous un nom différent.

Cette section décrit comment changer d'identité sur une connexion client existante. Les exemples de code suivants sont des versions abrégées du code du Quickstart. Voir le Quickstart **`ejb-security-interceptors`** pour obtenir un exemple complet.

Procédure 7.12. Modifier l'identité du contexte de sécurité

Pour modifier l'identité d'une connexion sécurisée, vous devez créer les 3 composants suivants.

1. **Créer l'intercepteur côté client**
   Cet intercepteur côté client doit implémenter l'interface **`org.jboss.ejb.client.EJBClientInterceptor`**. L'intercepteur doit passer l'identité requise par le mappage de données contextuelles, par l'intermédiaire d'un appel **`EJBClientInvocationContext.getContextData()`**. Voici un exemple de code d'intercepteur côté client :

```
public class ClientSecurityInterceptor implements
EJBClientInterceptor {

    public void handleInvocation(EJBClientInvocationContext context)
```

```
throws Exception {
        Principal currentPrincipal =
SecurityActions.securityContextGetPrincipal();

        if (currentPrincipal != null) {
            Map<String, Object> contextData =
context.getContextData();

contextData.put(ServerSecurityInterceptor.DELEGATED_USER_KEY,
currentPrincipal.getName());
        }
        context.sendRequest();
    }

    public Object handleInvocationResult(EJBClientInvocationContext
context) throws Exception {
        return context.getResult();
    }
}
```

Les applications utilisateur peuvent insérer l'intercepteur dans la chaîne d'intercepteur dans le **`EJBClientContext`** par l'un des moyens suivants :

- **Par programmation**
  Par cette approche, vous appelez la méthode **`org.jboss.ejb.client.EJBClientContext.registerInterceptor(int order, EJBClientInterceptor interceptor)`** et passez l'instance **`order`** et l'instance **`interceptor`**. L'instance **`order`** est utilisé pour déterminer où exactement cet intercepteur est placé dans la chaîne d'intercepteur.

- **Mécanisme ServiceLoader**
  Cette approche nécessite la création d'un fichier **`META-INF/services/org.jboss.ejb.client.EJBClientInterceptor`** et de le placer ou le mettre dans le chemin de classe de l'application client. Les règles du fichier sont dictées par le Java ServiceLoader Mechanism. Ce fichier doit contenir, dans chaque ligne distincte, le nom de classe qualifié complet de l'implémentation d'intercepteur de client EJB. Les classes d'intercepteur de client EJB doivent être disponibles dans le chemin de classe. Les intercepteurs de client EJB ajoutés en utilisant le mécanisme de **`ServiceLoader`** sont ajoutés à la fin de la chaîne d'intercepteur de client, dans l'ordre dans lequel ils se trouvent dans le chemin de classe (classpath). Le démarrage rapide ou quickstart **`ejb-security-interceptors`** utilise cette approche.

2. **Créer et configurer l'intercepteur du conteneur côté serveur**
   Les classes d'intercepteur de conteneur sont de simples Plain Old Java Objects (POJOs). Elles utilisent **`@javax.annotation.AroundInvoke`** pour marquer la méthode qui sera invoquée lors de l'invocation sur le bean. Pour plus d'informations sur les intercepteurs de conteneur, consulter : Section 7.6.1, « Intercepteurs de conteneurs ».

   a. **Créer l'intercepteur de conteneur**
      Cet intercepteur reçoit le **`InvocationContext`** en même temps que l'identité et demande le changement d'identité nouvelle. Ce qui suit est une version modifiée de l'exemple de code :

      ```
      public class ServerSecurityInterceptor {
      ```

```
    private static final Logger logger =
Logger.getLogger(ServerSecurityInterceptor.class);

    static final String DELEGATED_USER_KEY =
ServerSecurityInterceptor.class.getName() + ".DelegationUser";

    @AroundInvoke
    public Object aroundInvoke(final InvocationContext
invocationContext) throws Exception {
        Principal desiredUser = null;
        UserPrincipal connectionUser = null;

        Map<String, Object> contextData =
invocationContext.getContextData();
        if (contextData.containsKey(DELEGATED_USER_KEY)) {
            desiredUser = new SimplePrincipal((String)
contextData.get(DELEGATED_USER_KEY));

            Collection<Principal> connectionPrincipals =
SecurityActions.getConnectionPrincipals();

            if (connectionPrincipals != null) {
                for (Principal current : connectionPrincipals) {
                    if (current instanceof UserPrincipal) {
                        connectionUser = (UserPrincipal)
current;
                        break;
                    }
                }

            } else {
                throw new IllegalStateException("Delegation user
requested but no user on connection found.");
            }
        }


        ContextStateCache stateCache = null;
        try {
            if (desiredUser != null && connectionUser != null
                &&
(desiredUser.getName().equals(connectionUser.getName()) ==
false)) {
                // The final part of this check is to verify
that the change does actually indicate a change in user.
                try {
                    // We have been requested to use an
authentication token
                    // so now we attempt the switch.
                    stateCache =
SecurityActions.pushIdentity(desiredUser, new
OuterUserCredential(connectionUser));
                } catch (Exception e) {
                    logger.error("Failed to switch security
context for user", e);
                    // Don't propagate the exception stacktrace
```

```
            back to the client for security reasons
                            throw new EJBAccessException("Unable to
        attempt switching of user.");
                        }
                    }

                    return invocationContext.proceed();
                } finally {
                    // switch back to original context
                    if (stateCache != null) {
                        SecurityActions.popIdentity(stateCache);;
                    }
                }
            }
```

b. **Configurer l'intercepteur du conteneur**
Pour plus d'informations sur la façon de configurer les intercepteurs de conteneurs côté serveur, consulter : Section 7.6.3, « Configurer un intercepteur de conteneur ».

3. **Créer le JAAS LoginModule**
Ce composant doit vérifier que l'utilisateur est en mesure d'exécuter les requêtes selon l'identité de la requête. Les exemples de code abrégés suivants indiquent les méthodes de login et de validation :

```
    @SuppressWarnings("unchecked")
    @Override
    public boolean login() throws LoginException {
        if (super.login() == true) {
            log.debug("super.login()==true");
            return true;
        }

        // Time to see if this is a delegation request.
        NameCallback ncb = new NameCallback("Username:");
        ObjectCallback ocb = new ObjectCallback("Password:");

        try {
            callbackHandler.handle(new Callback[] { ncb, ocb });
        } catch (Exception e) {
            if (e instanceof RuntimeException) {
                throw (RuntimeException) e;
            }
            return false; // If the CallbackHandler can not handle
        the required callbacks then no chance.
        }

        String name = ncb.getName();
        Object credential = ocb.getCredential();

        if (credential instanceof OuterUserCredential) {
            // This credential type will only be seen for a
        delegation request, if not seen then the request is not for us.

            if (delegationAcceptable(name, (OuterUserCredential)
        credential)) {
```

```
                identity = new SimplePrincipal(name);
                if (getUseFirstPass()) {
                    String userName = identity.getName();
                    if (log.isDebugEnabled())
                        log.debug("Storing username '" + userName +
"' and empty password");
                    // Add the username and an empty password to
the shared state map

sharedState.put("javax.security.auth.login.name", identity);

sharedState.put("javax.security.auth.login.password", "");
                }
                loginOk = true;
                return true;
            }
        }

        return false; // Attempted login but not successful.
    }

    protected boolean delegationAcceptable(String requestedUser,
OuterUserCredential connectionUser) {
    if (delegationMappings == null) {
        return false;
    }

    String[] allowedMappings =
loadPropertyValue(connectionUser.getName(),
connectionUser.getRealm());
    if (allowedMappings.length == 1 &&
"*".equals(allowedMappings[1])) {
        // A wild card mapping was found.
        return true;
    }
    for (String current : allowedMappings) {
        if (requestedUser.equals(current)) {
            return true;
        }
    }
    return false;
}
```

Voir le fichier Quickstart **`ejb-security-interceptors` `README`** pour obtenir des instructions complètes et des informations détaillées sur le code.

Rapporter un bogue

7.6.5. Sécurité supplémentaire pour l'authentification EJB

Résumé

Par défaut, lorsque vous effectuez un appel distant à un EJB déployé sur le serveur d'applications, la connexion au serveur est authentifiée et toute demande reçue via cette connexion est exécutée en utilisant les informations d'identification qui ont authentifié la connexion. L'authentification au niveau de la connexion dépend des capacités des mécanismes sous-jacents SASL (Simple Authentication and

Security Layer). Plutôt que d'écrire des mécanismes SASL personnalisés, vous pouvez ouvrir et authentifier une connexion au serveur, puis ajouter les jetons de sécurité supplémentaires avant d'appeler un EJB. Cette rubrique décrit comment passer des informations supplémentaires sur la connexion existante du client pour l'authentification de l'EJB.

Les exemples de code ci-dessous sont uniquement à des fins de démonstration. Ils ne présentent qu'une approche possible et doivent être personnalisés pour répondre aux besoins précis de l'application. Le mot de passe est échangé par le mécanisme SASL. Si l'authentification DIGEST-MD5 SASL est utilisée, le mot de passe est toujours haché avec difficulté et non clair. Toutefois, les jetons restants, eux, sont clairs. Si ces jetons contiennent des informations sensibles, vous pouvez activer le cryptage pour la connexion.

Procédure 7.13. Information de sécurité pour l'authentification EJB

Pour un token de sécurité supplémentaire de connexion authentifiée, vous devrez créer les 3 composants suivants.

1. **Créer l'intercepteur côté client**
   Cet intercepteur doit implémenter **`org.jboss.ejb.client.EJBClientInterceptor`**. L'intercepteur doit passer le token de sécurité supplémentaire par le mappage de données contextuelles, par l'intermédiaire d'un appel **`EJBClientInvocationContext.getContextData()`**. Voici un exemple de code d'intercepteur côté client qui crée un token de sécurité supplémentaire :

   ```
   public class ClientSecurityInterceptor implements
   EJBClientInterceptor {

       public void handleInvocation(EJBClientInvocationContext context)
   throws Exception {
           Principal currentPrincipal =
   SecurityActions.securityContextGetPrincipal();

           if (currentPrincipal != null) {
               Map<String, Object> contextData =
   context.getContextData();

   contextData.put(ServerSecurityInterceptor.DELEGATED_USER_KEY,
   currentPrincipal.getName());
           }

           context.sendRequest();
       }

       public Object handleInvocationResult(EJBClientInvocationContext
   context) throws Exception {
           return context.getResult();
       }

   }
   ```

   Pour obtenir des informations sur la façon de connecter l'intercepteur client dans une application, voir Section 7.6.6, « Utiliser un intercepteur côté client dans une application ».

2. **Créer et configurer l'intercepteur du conteneur côté serveur**
   Les classes d'intercepteur de conteneur sont de simples Plain Old Java Objects (POJOs). Elles

utilisent **`@javax.annotation.AroundInvoke`** pour marquer la méthode qui est invoquée lors de l'invocation sur le bean. Pour plus d'informations sur les intercepteurs de conteneur, consulter : Section 7.6.1, « Intercepteurs de conteneurs ».

a. **Créer l'intercepteur de conteneur**
Cet intercepteur récupère le jeton d'authentification de sécurité du contexte et le passe au domaine JAAS (Java Authentication and Autorisation Service) pour vérification. Voici un exemple de code d'intercepteur de conteneur :

```
public class ServerSecurityInterceptor {

    private static final Logger logger =
Logger.getLogger(ServerSecurityInterceptor.class);

    static final String DELEGATED_USER_KEY =
ServerSecurityInterceptor.class.getName() + ".DelegationUser";

    @AroundInvoke
    public Object aroundInvoke(final InvocationContext
invocationContext) throws Exception {
        Principal desiredUser = null;
        UserPrincipal connectionUser = null;

        Map<String, Object> contextData =
invocationContext.getContextData();
        if (contextData.containsKey(DELEGATED_USER_KEY)) {
            desiredUser = new SimplePrincipal((String)
contextData.get(DELEGATED_USER_KEY));

            Collection<Principal> connectionPrincipals =
SecurityActions.getConnectionPrincipals();

            if (connectionPrincipals != null) {
                for (Principal current : connectionPrincipals) {
                    if (current instanceof UserPrincipal) {
                        connectionUser = (UserPrincipal)
current;
                        break;
                    }
                }

            } else {
                throw new IllegalStateException("Delegation user
requested but no user on connection found.");
            }
        }


        ContextStateCache stateCache = null;
        try {
            if (desiredUser != null && connectionUser != null
                &&
(desiredUser.getName().equals(connectionUser.getName()) ==
false)) {
                // The final part of this check is to verify
that the change does actually indicate a change in user.
```

```
                try {
                    // We have been requested to use an
authentication token
                    // so now we attempt the switch.
                    stateCache =
SecurityActions.pushIdentity(desiredUser, new
OuterUserCredential(connectionUser));
                } catch (Exception e) {
                    logger.error("Failed to switch security
context for user", e);
                    // Don't propagate the exception stacktrace
back to the client for security reasons
                    throw new EJBAccessException("Unable to
attempt switching of user.");
                }
            }

            return invocationContext.proceed();
        } finally {
            // switch back to original context
            if (stateCache != null) {
                SecurityActions.popIdentity(stateCache);;
            }
        }
    }
```

b. **Configurer l'intercepteur du conteneur**
Pour plus d'informations sur la façon de configurer les intercepteurs de conteneurs côté serveur, consulter : Section 7.6.3, « Configurer un intercepteur de conteneur ».

3. **Créer le JAAS LoginModule**
Ce module personnalisé exécute l'authentification à l'aide de l'information de la connexion authentifiée existante ainsi qu'à l'aide qu'un jeton de sécurité supplémentaire. Voici un exemple de code réduit qui utilise le jeton de sécurité supplémentaire et qui exécute l'authentification. On peut trouver l'exemple de code complet dans le quickstart **`ejb-security-interceptors`** fourni dans JBoss EAP 6.3 ou version supérieure.

```
public class DelegationLoginModule extends AbstractServerLoginModule
{

    private static final String DELEGATION_PROPERTIES =
"delegationProperties";

    private static final String DEFAULT_DELEGATION_PROPERTIES =
"delegation-mapping.properties";

    private Properties delegationMappings;

    private Principal identity;

    @Override
    public void initialize(Subject subject, CallbackHandler
callbackHandler, Map<String, ?> sharedState, Map<String, ?> options)
{
        addValidOptions(new String[] { DELEGATION_PROPERTIES });
```

```
        super.initialize(subject, callbackHandler, sharedState,
options);

        String propertiesName;
        if (options.containsKey(DELEGATION_PROPERTIES)) {
            propertiesName = (String)
options.get(DELEGATION_PROPERTIES);
        } else {
            propertiesName = DEFAULT_DELEGATION_PROPERTIES;
        }
        try {
            delegationMappings = loadProperties(propertiesName);
        } catch (IOException e) {
            throw new
IllegalArgumentException(String.format("Unable to load properties
'%s'", propertiesName), e);
        }
    }

    @SuppressWarnings("unchecked")
    @Override
    public boolean login() throws LoginException {
        if (super.login() == true) {
            log.debug("super.login()==true");
            return true;
        }

        // Time to see if this is a delegation request.
        NameCallback ncb = new NameCallback("Username:");
        ObjectCallback ocb = new ObjectCallback("Password:");

        try {
            callbackHandler.handle(new Callback[] { ncb, ocb });
        } catch (Exception e) {
            if (e instanceof RuntimeException) {
                throw (RuntimeException) e;
            }
            return false; // If the CallbackHandler can not handle
the required callbacks then no chance.
        }

        String name = ncb.getName();
        Object credential = ocb.getCredential();

        if (credential instanceof OuterUserCredential) {
            // This credential type will only be seen for a
delegation request, if not seen then the request is not for us.

            if (delegationAcceptable(name, (OuterUserCredential)
credential)) {

                identity = new SimplePrincipal(name);
                if (getUseFirstPass()) {
                    String userName = identity.getName();
                    if (log.isDebugEnabled())
                        log.debug("Storing username '" + userName +
```

```
"' and empty password");
                    // Add the username and an empty password to
the shared state map

sharedState.put("javax.security.auth.login.name", identity);

sharedState.put("javax.security.auth.login.password", "");
                }
                loginOk = true;
                return true;
            }
        }

        return false; // Attempted login but not successful.
    }
```

4. **Ajouter le LoginModule personnalisé à la chaîne**
   Vous devez ajouter le nouveau LoginModule personnalisé à l'endroit qui convient dans la chaîne pour qu'il soit invoqué dans le bon ordre. Dans cet exemple, le **`SaslPlusLoginModule`** doit être mis dans la chaîne avant le LoginModule qui charge les rôles par l'option **`password-stacking`** .

   - **Configurer l'ordonnancement du LoginModule par le Management CLI**
     Ce qui suit est un exemple de commandes de Management CLI qui enchaînent le **`SaslPlusLoginModule`** personnalisé avant que le LoginModule **`RealmDirect`** définisse l'option **`password-stacking`**.

     ```
     /subsystem=security/security-domain=quickstart-domain:add(cache-
     type=default)
     /subsystem=security/security-domain=quickstart-
     domain/authentication=classic:add
     /subsystem=security/security-domain=quickstart-
     domain/authentication=classic/login-
     module=DelegationLoginModule:add(code=org.jboss.as.quickstarts.ej
     b_security_plus.SaslPlusLoginModule,flag=optional,module-options=
     {password-stacking=useFirstPass})
     /subsystem=security/security-domain=quickstart-
     domain/authentication=classic/login-
     module=RealmDirect:add(code=RealmDirect,flag=required,module-
     options={password-stacking=useFirstPass})
     ```

     Pour plus d'informations sur le Management CLI, voir le chapitre intitulé *Management Interfaces* du guide *Administration and Configuration Guide* de JBoss EAP 6 qui se trouve sur le portail clients https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/

   - **Configurer l'ordonnancement du LoginModule manuellement**
     Ce qui suit est un exemple de XML qui configure l'ordonnancement du LoginModule dans le sous-système de **`security`** du fichier de configuration du serveur. Le **`SaslPlusLoginModule`** doit précéder le LoginModule **`RealmDirect`** pour qu'il puisse vérifier l'utilisateur distant avant que les rôles utilisateurs ne soient chargés et l'option **`password-stacking`** définie.

```
<security-domain name="quickstart-domain" cache-type="default">
    <authentication>
        <login-module
code="org.jboss.as.quickstarts.ejb_security_plus.SaslPlusLoginMod
ule" flag="required">
            <module-option name="password-stacking"
value="useFirstPass"/>
        </login-module>
        <login-module code="RealmDirect" flag="required">
            <module-option name="password-stacking"
value="useFirstPass"/>
        </login-module>
    </authentication>
</security-domain>
```

5. **Créer le client distant**
   Dans l'exemple de code suivant, on assume que le fichier **`additional-secret.properties`** auquel accède le JAAS LoginModule ci-dessus contient la propriété suivante :

   ```
   quickstartUser=7f5cc521-5061-4a5b-b814-bdc37f021acc
   ```

   Le code suivant montre comment créer un token de sécurité et comment le définir avant l'appel EJB. Le token secret est codé en dur dans des buts de démonstration uniquement. Ce client se contente d'afficher les résultats sur la console.

   ```
   import static
   org.jboss.as.quickstarts.ejb_security_plus.EJBUtil.lookupSecuredEJB;

   public class RemoteClient {

       /**
        * @param args
        */
       public static void main(String[] args) throws Exception {
           SimplePrincipal principal = new
   SimplePrincipal("quickstartUser");
           Object credential = new
   PasswordPlusCredential("quickstartPwd1!".toCharArray(), "7f5cc521-
   5061-4a5b-b814-bdc37f021acc");


   SecurityActions.securityContextSetPrincipalCredential(principal,
   credential);
           SecuredEJBRemote secured = lookupSecuredEJB();

           System.out.println(secured.getPrincipalInformation());
       }
   }
   ```

Rapporter un bogue

7.6.6. Utiliser un intercepteur côté client dans une application

Résumé

Vous pouvez connecter un intercepteur côté client dans une application par programmation ou en utilisant un mécanisme ServiceLoader. La procédure suivante décrit les deux méthodes.

Procédure 7.14. Connecter l'intercepteur

- **Par programmation**
  Par cette approche, vous appelez l'API **`org.jboss.ejb.client.EJBClientContext.registerInterceptor(int order, EJBClientInterceptor interceptor)`** et passez l'instance **`order`** et l'instance **`interceptor`**. L'instance **`order`** est utilisée pour déterminer où exactement cet **`interceptor`** est placé dans la chaîne d'intercepteur du client.

- **Mécanisme ServiceLoader**
  Cette approche nécessite la création d'un fichier **`META-INF/services/org.jboss.ejb.client.EJBClientInterceptor`** et le placer ou le mettre dans le chemin de classe de l'application client. Les règles du fichier sont dictées par le Java ServiceLoader Mechanism. Ce fichier devrait contenir, dans chaque ligne distincte, le nom de classe qualifié complet de l'implémentation d'intercepteur de client EJB. Les classes d'intercepteur de client EJB doivent être disponibles dans le chemin de classe. Les intercepteurs de client EJB ajoutés en utilisant le mécanisme de **`ServiceLoader`** sont ajoutés à la fin de la chaîne d'intercepteur de client, dans l'ordre dans lequel ils se trouvent dans le chemin de classe (classpath). Le démarrage rapide ou quickstart **`ejb-security-interceptors`** utilise cette approche.

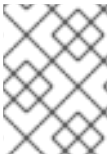
Rapporter un bogue

7.7. JAVABEANS ENTERPRISE CLUSTERISÉS

7.7.1. JavaBeans clusterisées (EJB)

Les composants EJB peuvent être clusterisés dans les scénarios de haute disponibilité. Ils utilisent différents protocoles HTTP, donc ils doivent être clusterisés différemment. Les beans avec ou sans état EJB 2 et 3 peuvent être clusterisés.

Pour plus d'informations sur les singletons, veuillez consulter : Section 9.4, « Implémenter un Singleton HA ».



NOTE

Les bean d'entité EJB 2 ne peuvent pas être clusterisés dans EAP 6 etc. C'est un problème de migration.

Rapporter un bogue

7.7.2. Configuration de client EJB In-server ou Autonome

Pour connecter un client EJB à une application EJB clusterisée, vous devez étendre la configuration existante dans le client EJB autonome ou dans l'EJB In-server afin d'inclure la configuration de connexion de cluster. Le fichier **`jboss-ejb-client.properties`** pour un client EJB autonome, ou même le fichier **`jboss-ejb-client.xml`** pour une application côté serveur doit être étendue pour inclure une configuration de cluster.

L'exemple suivant montre la configuration de connexion d'un client EJB autonome. L'exemple suivant ne montre que la configuration de cluster supplémentaire. L'exemple suivant (fichier de configuration **`jboss-ejb-client.xml`**) montre la configuration complète :

Exemple 7.3. Client autonome avec configuration `jboss-ejb-client.properties`

```
remote.clusters=ejb
remote.cluster.ejb.connect.options.org.xnio.Options.SASL_POLICY_NOANONYM
OUS=false
remote.cluster.ejb.connect.options.org.xnio.Options.SSL_ENABLED=false
remote.cluster.ejb.username=test
remote.cluster.ejb.password=password
```

Si une application utilise une remote-outbound-connection, vous devez configurer le fichier **`jboss-ejb-client.xml`** et ajouter la configuration de cluster comme illustré dans l'exemple suivant :

Exemple 7.4. Application client déployée dans une autre instance EAP 6 (Configurer le fichier jboss-ejb-client.xml)

```
<jboss-ejb-client xmlns:xsi="urn:jboss:ejb-client:1.2"
xsi:noNamespaceSchemaLocation="jboss-ejb-client_1_2.xsd">
   <client-context>
      <ejb-receivers>
            <!-- this is the connection to access the app-one -->
        <remoting-ejb-receiver outbound-connection-ref="remote-ejb-
connection-1" />
            <!-- this is the connection to access the app-two -->
        <remoting-ejb-receiver outbound-connection-ref="remote-ejb-
connection-2" />
      </ejb-receivers>

<!-- if an outbound connection connects to a cluster; a list of members
is provided after successful connection.
To connect to this node this cluster element must be defined. -->

    <clusters>
       <!-- cluster of remote-ejb-connection-1 -->
       <cluster name="ejb" security-realm="ejb-security-realm-1"
username="quickuser1">
          <connection-creation-options>
            <property name="org.xnio.Options.SSL_ENABLED" value="false"
/>
            <property name="org.xnio.Options.SASL_POLICY_NOANONYMOUS"
value="false" />
          </connection-creation-options>
       </cluster>
    </clusters>
   </client-context>
</jboss-ejb-client>
```

NOTE

Pour avoir une connexion sécurisée, vous devez ajouter les identifiants à la configuration du cluster pour éviter une exception d'authentification.

Rapporter un bogue

7.7.3. Mettre en place une politique d'équilibrage des charges presonnalisée pour les appels EJB

Il est possible de mettre en place une politique d'équilibrage des charges personnalisée/alternative de façon à ce que les serveurs de l'application n'aient pas à gérer le même nombre d'appels EJB en général ou pour une période donnée.

Vous pouvez implémenter **`AllClusterNodeSelector`** pour les appels EJB. Le comportement de sélection de noeud de **`AllClusterNodeSelector`** ressemble à celui du sélecteur par défaut, sauf que **`AllClusterNodeSelector`** utilise tous les noeuds de cluster disponibles, même pour les clusters de grande taille (nombre de noeuds>20). Si un noeud de cluster non connecté est renvoyé, il s'ouvre automatiquement. L'exemple suivant montre une implémentation de **`AllClusterNodeSelector`** :

```
package org.jboss.as.quickstarts.ejb.clients.selector;

import java.util.Arrays;
import java.util.Random;
import java.util.logging.Level;
import java.util.logging.Logger;

import org.jboss.ejb.client.ClusterNodeSelector;
public class AllClusterNodeSelector implements ClusterNodeSelector {
  private static final Logger LOGGER =
Logger.getLogger(AllClusterNodeSelector.class.getName());

  @Override
  public String selectNode(final String clusterName, final String[]
connectedNodes, final String[] availableNodes) {
    if(LOGGER.isLoggable(Level.FINER)) {
      LOGGER.finer("INSTANCE "+this+ " : cluster:"+clusterName+"
connected:"+Arrays.deepToString(connectedNodes)+"
available:"+Arrays.deepToString(availableNodes));
    }

    if (availableNodes.length == 1) {
        return availableNodes[0];
    }
    final Random random = new Random();
    final int randomSelection = random.nextInt(availableNodes.length);
    return availableNodes[randomSelection];
  }

}
```

Vous pouvez également implémenter le **`SimpleLoadFactorNodeSelector`** pour les appels EJB. L'équilibrage des charges dans **`SimpleLoadFactorNodeSelector`** a lieu sur la base d'un facteur de charge. Le facteur de charge (2/3/4) est calculé sur la base des noms de noeuds (A/B/C)

irrespectivement de la charge qui pèse sur chaque noeud. L'exemple suivant vous montre une implémentation de **`SimpleLoadFactorNodeSelector`** :

```
package org.jboss.as.quickstarts.ejb.clients.selector;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collection;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.logging.Level;
import java.util.logging.Logger;

import org.jboss.ejb.client.DeploymentNodeSelector;
public class SimpleLoadFactorNodeSelector implements
DeploymentNodeSelector {
  private static final Logger LOGGER =
Logger.getLogger(SimpleLoadFactorNodeSelector.class.getName());
  private final Map<String, List<String>[]> nodes = new HashMap<String,
List<String>[]>();
  private final Map<String, Integer> cursor = new HashMap<String, Integer>
();

  private ArrayList<String> calculateNodes(Collection<String>
eligibleNodes) {
    ArrayList<String> nodeList = new ArrayList<String>();

    for (String string : eligibleNodes) {
      if(string.contains("A") || string.contains("2")) {
        nodeList.add(string);
        nodeList.add(string);
      } else if(string.contains("B") || string.contains("3")) {
        nodeList.add(string);
        nodeList.add(string);
        nodeList.add(string);
      } else if(string.contains("C") || string.contains("4")) {
        nodeList.add(string);
        nodeList.add(string);
        nodeList.add(string);
        nodeList.add(string);
      }
    }
    return nodeList;
  }

  @SuppressWarnings("unchecked")
  private void checkNodeNames(String[] eligibleNodes, String key) {
    if(!nodes.containsKey(key) || nodes.get(key)[0].size() !=
eligibleNodes.length || !nodes.get(key)
[0].containsAll(Arrays.asList(eligibleNodes))) {
      // must be synchronized as the client might call it concurrent
      synchronized (nodes) {
        if(!nodes.containsKey(key) || nodes.get(key)[0].size() !=
eligibleNodes.length || !nodes.get(key)
[0].containsAll(Arrays.asList(eligibleNodes))) {
```

```
          ArrayList<String> nodeList = new ArrayList<String>();
          nodeList.addAll(Arrays.asList(eligibleNodes));

          nodes.put(key, new List[] { nodeList, calculateNodes(nodeList)
});
        }
      }
    }
  }
   private synchronized String nextNode(String key) {
    Integer c = cursor.get(key);
    List<String> nodeList = nodes.get(key)[1];

    if(c == null || c >= nodeList.size()) {
      c = Integer.valueOf(0);
    }

    String node = nodeList.get(c);
    cursor.put(key, Integer.valueOf(c + 1));

    return node;
  }

  @Override
  public String selectNode(String[] eligibleNodes, String appName, String
moduleName, String distinctName) {
    if (LOGGER.isLoggable(Level.FINER)) {
      LOGGER.finer("INSTANCE " + this + " : nodes:" +
Arrays.deepToString(eligibleNodes) + " appName:" + appName + "
moduleName:" + moduleName
          + " distinctName:" + distinctName);
    }

    // if there is only one there is no sense to choice
    if (eligibleNodes.length == 1) {
      return eligibleNodes[0];
    }
    final String key = appName + "|" + moduleName + "|" + distinctName;

    checkNodeNames(eligibleNodes, key);
    return nextNode(key);
  }
}
```

Configuration avec `jboss-ejb-client.properties`

Vous devrez ajouter la propriété **`remote.cluster.ejb.clusternode.selector`** au nom de votre classe d'implémentation (**`AllClusterNodeSelector`** ou **`SimpleLoadFactorNodeSelector`**). Le sélecteur vérifiera les serveurs configurés disponibles au moment de l'invocation. l'exemple suivant utilise **`AllClusterNodeSelector`** comme sélecteur de noeud de déploiement :

```
remote.clusters=ejb
remote.cluster.ejb.clusternode.selector=org.jboss.as.quickstarts.ejb.clien
ts.selector.AllClusterNodeSelector
remote.cluster.ejb.connect.options.org.xnio.Options.SASL_POLICY_NOANONYMOU
S=false
```

```
remote.cluster.ejb.connect.options.org.xnio.Options.SSL_ENABLED=false
remote.cluster.ejb.username=test
remote.cluster.ejb.password=password

remote.connectionprovider.create.options.org.xnio.Options.SSL_ENABLED=fals
e
remote.connections=one,two
remote.connection.one.host=localhost
remote.connection.one.port = 4447
remote.connection.one.connect.options.org.xnio.Options.SASL_POLICY_NOANONY
MOUS=false
remote.connection.one.username=user
remote.connection.one.password=user123
remote.connection.two.host=localhost
remote.connection.two.port = 4547
remote.connection.two.connect.options.org.xnio.Options.SASL_POLICY_NOANONY
MOUS=false
```

Utiliser l'API JBoss ejb-client

Vous devrez ajouter la propriété **`remote.cluster.ejb.clusternode.selector`** à la liste de constructeur **`PropertiesBasedEJBClientConfiguration`**. L'exemple suivant utilise **`AllClusterNodeSelector`** comme sélecteur de noeud de déploiement :

```
Properties p = new Properties();
p.put("remote.clusters", "ejb");
p.put("remote.cluster.ejb.clusternode.selector",
"org.jboss.as.quickstarts.ejb.clients.selector.AllClusterNodeSelector");
p.put("remote.cluster.ejb.connect.options.org.xnio.Options.SASL_POLICY_NOA
NONYMOUS", "false");
p.put("remote.cluster.ejb.connect.options.org.xnio.Options.SSL_ENABLED",
"false");
p.put("remote.cluster.ejb.username", "test");
p.put("remote.cluster.ejb.password", "password");

p.put("remote.connectionprovider.create.options.org.xnio.Options.SSL_ENABL
ED", "false");
p.put("remote.connections", "one,two");
p.put("remote.connection.one.port", "4447");
p.put("remote.connection.one.host", "localhost");
p.put("remote.connection.two.port", "4547");
p.put("remote.connection.two.host", "localhost");

EJBClientConfiguration cc = new PropertiesBasedEJBClientConfiguration(p);
ContextSelector<EJBClientContext> selector = new
ConfigBasedEJBClientContextSelector(cc);
EJBClientContext.setSelector(selector);

p = new Properties();
p.put(Context.URL_PKG_PREFIXES, "org.jboss.ejb.client.naming");
InitialContext context = new InitialContext(p);
```

Configuration côté application de serveur avec `jboss-ejb-client.xml`

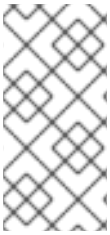
Pour pouvoir utiliser la politique d'équilibrage des charges pour la communication d'un serveur à un autre, empaquetez la classe avec l'application et configurez-là dans les paramètres de **`jboss-ejb-`**

client.xml (qui se trouvent dans le dossier **META-INF**). L'exemple suivant utilise **AllClusterNodeSelector** comme sélecteur de noeud de déploiement :

```
<jboss-ejb-client xmlns:xsi="urn:jboss:ejb-client:1.2"
xsi:noNamespaceSchemaLocation="jboss-ejb-client_1_2.xsd">
  <client-context>
    <ejb-receivers>
      <!-- this is the connection to access the app -->
      <remoting-ejb-receiver outbound-connection-ref="remote-ejb-
connection-1" />
    </ejb-receivers>

  <!-- if an outbound connection connect to a cluster a list of members is
provided after successful connection.
To connect to this node this cluster element must be defined.
-->
   <clusters>
     <!-- cluster of remote-ejb-connection-1 -->
     <cluster name="ejb" security-realm="ejb-security-realm-1"
username="test" cluster-node-
selector="org.jboss.as.quickstarts.ejb.clients.selector.AllClusterNodeSele
ctor">
        <connection-creation-options>
           <property name="org.xnio.Options.SSL_ENABLED" value="false" />
           <property name="org.xnio.Options.SASL_POLICY_NOANONYMOUS"
value="false" />
        </connection-creation-options>
      </cluster>
   </clusters>
  </client-context>
</jboss-ejb-client>
```

Pour pouvoir utiliser la configuration ci-dessus en toute sécurité, il vous faudra ajouter **ejb-security-realm-1** à la configuration côté serveur. L'exemple suivant vous montre les commandes CLI pour ajouter un domaine de sécurité (**ejb-security-realm-1**). La valeur correspond au mot de passe base64 codifié pour l'utilisateur "test" :

```
core-service=management/security-realm=ejb-security-realm-1:add()
core-service=management/security-realm=ejb-security-realm-1/server-
identity=secret:add(value=cXVpY2sxMjMr)
```



NOTE

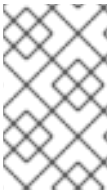
Si vous êtes en mode autonome, utiliser l'option de démarrage **-Djboss.node.name=** ou le fichier de configuration du serveur **standalone.xml** pour configurer le nom du serveur (server name=""). Veillez à ce que le nom de serveur soit unique. En mode de domaine, le contrôleur valide automatiquement le fait que les noms soient uniques.

Rapporter un bogue

7.7.4. Comportement de transaction des invocations EJB

Invocations de serveur à serveur

Les attributs de transaction d'applications distribuées EAP doivent être gérés comme si l'application était appelée sur le même serveur. Pour mettre fin à une transaction, la méthode de destination doit comporter la mention **`REQUIRES_NEW`**, à l'aide d'interfaces différentes.



NOTE

EAP 6 n'a pas besoin de JTS (de l'anglais Java Transaction Services) pour la propagation de la transaction sur les invocations EJB d'un serveur à l'autre si les deux serveurs sont EAP 6. La bibliothèque d'API cliente de JBoss EJB la gère elle-même.

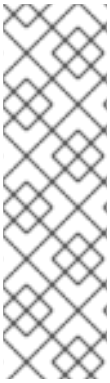
Les invocations côté client

Pour invoquer les beans de session EJB avec un client EAP 6 autonome, le client doit avoir une référence à l'objet **`InitialContext`** tandis que les proxies EJB ou **`UserTransaction`** sont utilisés. Il est également important de garder l'objet **`InitialContext`** ouvert tant que les proxies EJB ou **`UserTransaction`** sont utilisés. Le contrôle des connexions se fera à l'intérieur des classes créées par **`InitialContext`** avec les propriétés.

L'exemple suivant vous montre l'API du client EJB qui contient la référence à l'objet **`InitialContext`** :

```
package org.jboss.as.quickstarts.ejb.multi.server;

import java.util.Date;
import java.util.Properties;
import java.util.logging.Level;
import java.util.logging.Logger;

import javax.naming.Context;
import javax.naming.InitialContext;

import org.jboss.as.quickstarts.ejb.multi.server.app.MainApp;
import org.jboss.ejb.client.ContextSelector;
import org.jboss.ejb.client.EJBClientConfiguration;
import org.jboss.ejb.client.EJBClientContext;
import org.jboss.ejb.client.PropertiesBasedEJBClientConfiguration;
import org.jboss.ejb.client.remoting.ConfigBasedEJBClientContextSelector;

public class Client {

/**
* @param args no args needed
* @throws Exception
*/
    public static void main(String[] args) throws Exception {
        // suppress output of client messages
        Logger.getLogger("org.jboss").setLevel(Level.OFF);
        Logger.getLogger("org.xnio").setLevel(Level.OFF);

        Properties p = new Properties();

p.put("remote.connectionprovider.create.options.org.xnio.Options.SSL_ENABL
ED", "false");
        p.put("remote.connections", "one");
        p.put("remote.connection.one.port", "4447");
```

```
        p.put("remote.connection.one.host", "localhost");
        p.put("remote.connection.one.username", "quickuser");
        p.put("remote.connection.one.password", "quick-123");

        EJBClientConfiguration cc = new
PropertiesBasedEJBClientConfiguration(p);
        ContextSelector<EJBClientContext> selector = new
ConfigBasedEJBClientContextSelector(cc);
        EJBClientContext.setSelector(selector);

        Properties props = new Properties();
        props.put(Context.URL_PKG_PREFIXES,
"org.jboss.ejb.client.naming");
        InitialContext context = new InitialContext(props);


        final String rcal = "ejb:jboss-ejb-multi-server-app-main/ejb//" +
("MainAppBean") + "!" + MainApp.class.getName();
        final MainApp remote = (MainApp) context.lookup(rcal);
        final String result = remote.invokeAll("Client call at "+new
Date());

        System.out.println("InvokeAll succeed: "+result);
    }

}
```



NOTE

L'obtention d'une réféence **`UserTransaction`** sur un client n'est pas prise en charge pour les scénarios de contextes client EJB scoped et pour les invocations qui utilisent le protocole **`remote-naming`**. C'est parce que dans ces scénarios, **`InitialContext`** encapsule sa propre instance de contexte client, à laquelle on ne peut pas accéder par les méthodes statiques de la classe **`EJBClient`**. Quand la méthode **`EJBClient.getUserTransaction()`** est invoquée, elle renvoie une transaction en provenance du contexte client EJB (global) par défaut (pas forcément initialisé) et non pas en provenance du contexte souhaité.

Référence UserTransaction côté client

L'exemple suivant vous montre comment obtenir une référence **`UserTransaction`** sur un client autonome :

```
import org.jboss.ejb.client.EJBClient;
import javax.transaction.UserTransaction;
.
.
    Context context=null;
    UserTransaction tx=null;
    try {
      Properties props = new Properties();
      // REMEMBER: there must be a jboss-ejb-client.properties with the
connection parameter
      //           in the clients classpath
      props.put(Context.URL_PKG_PREFIXES, "org.jboss.ejb.client.naming");
```

```
      context = new InitialContext(props);
      System.out.println("\n\tGot initial Context: "+context);
      tx=EJBClient.getUserTransaction("yourServerName");
      System.out.println("UserTransaction = "+tx.getStatus());
      tx.begin();
      // do some work
      ...
    }catch (Exception e) {
      e.printStackTrace();
      tx.rollback();
    }finally{
      if(context != null) {
        context.close();
      }
    }
```

NOTE

Pour obtenir une référence **`UserTransaction`** côté client, démarrer votre serveur avec la propriété système suivante **`-Djboss.node.name=yourServerName`** et utilisez-là ensuite côté client comme suit :

```
tx=EJBClient.getUserTransaction("yourServerName");
```

Remplacer "yourServerName" par le nom de votre serveur. Si une transaction utilisateur démarre sur un nœud, toutes les invocations seront sticky sur le nœud et le nœud devra avoir tous les EJB nécessaires. Il n'est pas possible d'utiliser **`UserTransaction`** avec un protocole de nommage à distance ou avec un scoped-context.

Rapporter un bogue

7.8. RÉFÉRENCE

7.8.1. Référence de nommage EJB JNDI

Le nom de recherche JNDI d'une session bean a la syntaxe suivante :

```
 ejb:<appName>/<moduleName>/<distinctName>/<beanName>!<viewClassName>?
stateful
```

`<appName>`

Si le fichier JAR du session bean a été déployé dans un EAR, alors ce se sera le nom de l'EAR. Par défaut, le nom d'un EAR correspond à son nom de fichier sans le suffixe **`.ear`**. Le nom de l'application peut également être remplacé dans son fichier **`application.xml`**. Si le bean de session n'est pas déployé dans un EAR, laissez le vide.

`<moduleName>`

Le nom du module correspond au nom du fichier dans lequel le session bean est déployé. Par défaut, le nom du fichier JAR correspond à son nom de fichier sans le suffixe **`.jar`**. Le nom du module peut également être remplacé par le **`ejb-jar.xml`** du JAR.

<distinctName>

JBoss EAP 6 permet à chaque déploiement de spécifier un nom distinct en option. Si le déploiement n'a pas de nom distinct, laisser le vide.

<beanName>

Le nom du bean est le nom de classe de la session bean à invoquer.

<viewClassName>

Le nom de classe de vue est le nom de classe complet de l'interface distante de l'interface distante. Inclut le nom du package de l'interface.

?stateful

Le suffixe **?stateful** est requis quand le nom JNDI se réfère à un bean de session stateful. Non inclus dans d'autres types de bean.

Rapporter un bogue

7.8.2. Résolution de référence EJB

Cette section explique comment JBoss implémente **@EJB** et **@Resource**. Veuillez noter qu'XML remplace toujours les annotations, mais que les mêmes règles s'appliquent.

Règles pour les annotations @EJB

- L'annotation **@EJB** possède également un attribut **mappedName()**. La spécification le considère comme une métadonnée spécifique au fournisseur tandis que JBoss reconnait **mappedName()** en tant que nom JNDI global du EJB que vous référencez. Si vous avez indiqué un **mappedName()**, alors tous les autres attributs seront ignorés et ce nom JNDI global sera utilisé comme liaison.
- Si vous indiquez **@EJB** sans attributs définis :

  ```
  @EJB
  ProcessPayment myEjbref;
  ```

  Alors les règles suivantes vont s'appliquer :

  - Le jar EJB du bean de référencement est analysé pour trouver un EJB avec l'interface utilisée dans l'injection **@EJB**. S'il existe plus d'un EJB qui publie la même interface commerciale, alors une exception apparaîtra. S'il n'y a qu'un seul bean dans cette interface, alors celui-là sera utilisé.
  - Chercher l'EAR des EJB qui publient cette interface. S'il y a des doubles, alors il y aura une exception. Sinon le bean correspondant sera renvoyé.
  - Chercher globalement dans le runtime de JBoss un EJB de cette interface. Une fois de plus, s'il y a des doubles, il y aura une exception.
- **@EJB.beanName()** correspond à **<ejb-link>**. Si le **beanName()** est défini, il vous faudra utiliser le même algorithme que **@EJB** sans attribut défini mis à part **beanName()** comme une clé de la recherche. Il existe une exception à cette règle si vous utilisez la syntaxe ejb-link '#'. La

syntaxe '#' vous permettra de mettre un chemin d'accès relatif dans un jar de l'EJB qui contient l'EJB de référencement. Reportez-vous à la spécification EJB 3.1 pour plus de détails.

Rapporter un bogue

7.8.3. Dépendances de projet pour les clients EJB distants

Les projets Maven qui incluent l'invocation de session beans de clients éloignés ont besoin des dépendances suivantes du référentiel JBoss EAP 6 Maven.

Tableau 7.4. Dépendances Maven pour les clients EJB

GroupID	ArtifactID
org.jboss.spec	jboss-javaee-6.0
org.jboss.as	jboss-as-ejb-client-bom
org.jboss.spec.javax.transaction	jboss-transaction-api_1.1_spec
org.jboss.spec.javax.ejb	jboss-ejb-api_3.1_spec
org.jboss	jboss-ejb-client
org.jboss.xnio	xnio-api
org.jboss.xnio	xnio-nio
org.jboss.remoting3	jboss-remoting
org.jboss.sasl	jboss-sasl
org.jboss.marshalling	jboss-marshalling-river

A l'exception de **`jboss-javaee-6.0`** et **`jboss-as-ejb-client-bom`**, ces dépendances doivent être ajoutées à la section **`<dependencies>`** du fichier **`pom.xml`**.

Les dépendances **`jboss-javaee-6.0`** et **`jboss-as-ejb-client-bom`** doivent être ajoutées à la section **`<dependencyManagement>`** du fichier **`pom.xml`** avec le scope d' **`import`**.

NOTE

Les versions de **`artifactID`** peuvent être modifiées. Veuillez consulter le référentiel Maven pour connaître la version la plus récente.

```
<dependencyManagement>
   <dependencies>
      <dependency>
         <groupId>org.jboss.spec</groupId>
         <artifactId>jboss-javaee-6.0</artifactId>
```

```
            <version>3.0.0.Final-redhat-1</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>

        <dependency>
            <groupId>org.jboss.as</groupId>
            <artifactId>jboss-as-ejb-client-bom</artifactId>
            <version>7.1.1.Final-redhat-1</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
```

Voir **ejb-remote/client/pom.xml** dans les fichiers quickstart pour obtenir un exemple complet de configuration de dépendance pour une invocation de session bean éloignée.

Rapporter un bogue

7.8.4. Référence de descripteur de déploiement jboss-ejb3.xml

jboss-ejb3.xml est un descripteur de déploiement personnalisé pouvant être utilisé dans un JAR EJB ou des archives WAR. Dans une archive JAR EJB, il doit être situé dans le répertoire **META-INF/**. Dans une archive WAR, il doit être situé dans le répertoire **WEB-INF/**.

Le format ressemble à **ejb-jar.xml**, qui utilise les mêmes espace-noms et qui fournit des espace-noms supplémentaires. Les contenus de **jboss-ejb3.xml** sont mergés avec les contenus de **ejb-jar.xml**, et les items **jboss-ejb3.xml** ont la priorité.

Ce document ne couvre que les espace-noms supplémentaires non-standard utilisés par **jboss-ejb3.xml**. Voir http://java.sun.com/xml/ns/javaee/ pour la documentation sur les espace-noms standard.

L'espace-nom root est **http://www.jboss.com/xml/ns/javaee**.

Espace-nom de descripteur d'assemblage

Les espace-noms suivants peuvent être utilisés dans l'élément **<assembly-descriptor>**. Ils peuvent être utilisés pour appliquer leur configuration à un simple bean, ou à tous les beans du déploiement en utilisant ***** comme **ejb-name**.

Espace-nom de clustering : urn:clustering:1.0

```
xmlns:c="urn:clustering:1.0"
```

Cela vous permet de marquer les EJB comme étant clusterisés. Il s'agit du descripteur de déploiement équivalent au **@org.jboss.ejb3.annotation.Clustered**.

```
<c:clustering>
   <ejb-name>DDBasedClusteredSFSB</ejb-name>
   <c:clustered>true</c:clustered>
</c:clustering>
```

L'espace-nom de sécurité (`urn:security`)

```
xmlns:s="urn:security"
```

Cela vous permet de définir le domaine de sécurité et le principal prétendant pour l'EJB.

```
<s:security>
  <ejb-name>*</ejb-name>
  <s:security-domain>myDomain</s:security-domain>
  <s:run-as-principal>myPrincipal</s:run-as-principal>
</s:security>
```

L'espace-nom d'adaptateur de ressource : `urn:resource-adapter-binding`

```
xmlns:r="urn:resource-adapter-binding"
```

Cela vous permet de définir l'adaptateur de ressource d'un Message-Driven Bean.

```
<r:resource-adapter-binding>
  <ejb-name>*</ejb-name>
  <r:resource-adapter-name>myResourceAdapter</r:resource-adapter-name>
</r:resource-adapter-binding>
```

L'espace-nom IIOP : `urn:iiop`

```
xmlns:u="urn:iiop"
```

L'espace-nom IIOP est l'endroit où les paramètres IIOP sont configurés.

L'espace-nom du pool : `urn:ejb-pool:1.0`

```
xmlns:p="urn:ejb-pool:1.0"
```

Cela vous permet de sélectionner le pool qui est utilisé par les stateless session beans ou par les Message-Driven Beans. Les pools sont définis dans la configuration du serveur.

```
<p:pool>
   <ejb-name>*</ejb-name>
   <p:bean-instance-pool-ref>my-pool</p:bean-instance-pool-ref>
</p:pool>
```

L'espace-nom du cache : `urn:ejb-cache:1.0`

```
xmlns:c="urn:ejb-cache:1.0"
```

Cela vous permet de sélectionner le cache qui est utilisé par les stateful session beans inclus. Les caches sont définis dans la configuration du serveur.

```
<c:cache>
   <ejb-name>*</ejb-name>
   <c:cache-ref>my-cache</c:cache-ref>
</c:cache>
```

Exemple 7.5. Fichier exemple de jboss-ejb3.xml

```
<?xml version="1.1" encoding="UTF-8"?>
   <jboss:ejb-jar xmlns:jboss="http://www.jboss.com/xml/ns/javaee"
                  xmlns="http://java.sun.com/xml/ns/javaee"
                  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                  xmlns:c="urn:clustering:1.0"

xsi:schemaLocation="http://www.jboss.com/xml/ns/javaee
http://www.jboss.org/j2ee/schema/jboss-ejb3-2_0.xsd
http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/ejb-
jar_3_1.xsd"
                  version="3.1"
                  impl-version="2.0">
      <enterprise-beans>
         <message-driven>
            <ejb-name>ReplyingMDB</ejb-name>
            <ejb-
class>org.jboss.as.test.integration.ejb.mdb.messagedestination.ReplyingM
DB</ejb-class>
            <activation-config>
               <activation-config-property>
                  <activation-config-property-
name>destination</activation-config-property-name>
                  <activation-config-property-
value>java:jboss/mdbtest/messageDestinationQueue
                  </activation-config-property-value>
               </activation-config-property>
            </activation-config>
         </message-driven>
      </enterprise-beans>
      <assembly-descriptor>
         <c:clustering>
            <ejb-name>DDBasedClusteredSFSB</ejb-name>
            <c:clustered>true</c:clustered>
         </c:clustering>
      </assembly-descriptor>
   </jboss:ejb-jar>
```

Rapporter un bogue

CHAPITRE 8. LES SERVICES JBOSS MBEAN

8.1. RÉDIGER DES SERVICES JBOSS MBEAN

Rédiger un service MBean personnalisé qui s'appuie sur un service JBoss nécessite un modèle de méthode d'interface de service. Le modèle de méthode d'interface de service de JBoss MBean se compose d'un ensemble d'opérations de cycle de vie qui informent un service MBean quand il se **crée**, quand il **démarre**, **s'arrête** ou se **détruit** lui-même.

Vous pouvez gérer l'état de la dépendance par une des approches suivantes :

- Si vous souhaitez que des méthodes spécifiques soient invoquées sur votre MBean, il vous faudra déclarer ces méthodes dans votre interface MBean. Cette approche permet à votre implementation MBean d'éviter des dépendances sur les classes JBoss.
- Si vous ne vous inquiétez pas des dépendances sur les classes spécifiques à JBoss, alors vous souhaiterez sans doute que votre interface MBean prolonge l'interface **ServiceMBean** et la classe **ServiceMBeanSupport**. La classe **ServiceMBeanSupport** fournit des implémentations des méthodes de cycle de vie de services comme **create**, **start** et **stop**. Pour un événement spécifique comme **start()**, vous devrez surcharger la méthode **startService()** donnée par la classe ServiceMBeanSupport.

Rapporter un bogue

8.2. EXEMPLE DE MBEAN STANDARD

Cette section développe deux exemples de services MBean qui font partie d'un même package d'archive de service (**.sar**).

L'interface **ConfigServiceMBean** déclare des méthodes spécifiques telles que les méthodes **start**, **getTimeout** et **stop** pour qu'elles **démarrent**, **mettent en attente** et **stoppent** le MBean correctement sans utiliser de classes spécifiques à JBoss. La classe **ConfigService** implémente l'interface **ConfigServiceMBean** et de ce fait implémente les méthodes utilisées dans cette interface.

La classe **PlainThread** étend la classe **ServiceMBeanSupport** et implémente l'interface **PlainThreadMBean**. **PlainThread** démarre un thread et utilise la méthode **ConfigServiceMBean.getTimeout()** pour déterminer la durée pendant laquelle le thread doit dormir.

Exemple 8.1. Exemples de services MBean

```
package org.jboss.example.mbean.support;

public interface ConfigServiceMBean {

    int getTimeout();

    void start();

    void stop();

}

package org.jboss.example.mbean.support;
```

```
public class ConfigService implements ConfigServiceMBean {
    int timeout;


    @Override
    public int getTimeout() {
        return timeout;
    }

    @Override
    public void start() {
        //Create a random number between 3000 and 6000 milliseconds
        timeout = (int)Math.round(Math.random() * 3000) + 3000;
        System.out.println("Random timeout set to " + timeout + "
seconds");
    }

    @Override
    public void stop() {
        timeout = 0;
    }

}

package org.jboss.example.mbean.support;

import org.jboss.system.ServiceMBean;

public interface PlainThreadMBean extends ServiceMBean {
    void setConfigService(ConfigServiceMBean configServiceMBean);
}

package org.jboss.example.mbean.support;

import org.jboss.system.ServiceMBeanSupport;

public class PlainThread extends ServiceMBeanSupport implements
PlainThreadMBean {

    private ConfigServiceMBean configService;
    private Thread thread;
    private volatile boolean done;

    @Override
    public void setConfigService(ConfigServiceMBean configService) {
        this.configService = configService;
    }

    @Override
    protected void startService() throws Exception {
        System.out.println("Starting Plain Thread MBean");
        done = false;
        thread = new Thread(new Runnable() {
            @Override
            public void run() {
```

```
                try {
                    while (!done) {
                        System.out.println("Sleeping....");
                        Thread.sleep(configService.getTimeout());
                        System.out.println("Slept!");
                    }
                } catch (InterruptedException e) {
                    Thread.currentThread().interrupt();
                }
            }
        });
        thread.start();
    }

    @Override
    protected void stopService() throws Exception {
        System.out.println("Stopping Plain Thread MBean");
        done = true;
    }


}
```

Le descripteur **`jboss-service.xml`** montre comment la classe **`ConfigService`** est injectée dans la classe **`PlainThread`** en utilisant la balise **`inject`**. La balise **`inject`** établit une dépendance entre **`PlainThreadMBean`** et **`ConfigServiceMBean`** permettant ainsi à **`PlainThreadMBean`** d'utiliser **`ConfigServiceMBean`** aisément.

Exemple 8.2. Décripteur de service JBoss-service.xml

```
<server xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="urn:jboss:service:7.0 jboss-
service_7_0.xsd"
        xmlns="urn:jboss:service:7.0">
 <mbean code="org.jboss.example.mbean.support.ConfigService"
name="jboss.support:name=ConfigBean"/>
 <mbean code="org.jboss.example.mbean.support.PlainThread"
name="jboss.support:name=ThreadBean">
  <attribute name="configService">
   <inject bean="jboss.support:name=ConfigBean"/>
  </attribute>
 </mbean>
</server>
```

Aprés avoir rédigé les exemples de MBeans, vous pouvez empaqueter les classes et le descripteur **`jboss-service.xml`** dans le dossier **`META-INF`** d'une archive de service (**`.sar`**).

Rapporter un bogue

8.3. DÉPLOYER JBOSS MBEAN SERVICES

Pour créer et déployer l'exemple de MBeans (**`ServiceMBeanTest.sar`**) en mode **`Domain`**, utliser les commandes suivantes :

```
[domain@localhost:9999 /] deploy ~/Desktop/ServiceMBeanTest.sar
```

```
[domain@localhost:9999 /] deploy ~/Desktop/ServiceMBeanTest.sar --all-server-groups
```

Pour créer et déployer l'exemple de MBeans (**`ServiceMBeanTest.sar`**) en mode **`Standalone`**, utliser la commande suivante :

```
[standalone@localhost:9999 /] deploy ~/Desktop/ServiceMBeanTest.sar
```

Pour annuler le déploiement de l'exemple MBeans, utiliser la commande suivante :

```
[standalone@localhost:9999 /] undeploy ServiceMBeanTest.sar
```

Rapporter un bogue

CHAPITRE 9. CLUSTERING DANS LES APPLICATIONS WEB

9.1. RÉPLIQUE DE SESSION

9.1.1. La réplique de session HTTP

La réplique de session veille à ce que les sessions client d'applications distribuables ne soient pas interrompues par des défaillances de noeuds dans le système. Chaque noeud du cluster partage des informations sur les sessions en cours, et peut les récupérer si le noeud d'origine disparaît.

La réplique de session est un mécanisme par lequel les clusters mod_cluster, mod_jk, mod_proxy, ISAPI, et NSAPI procurent une haute disponibilité.

Rapporter un bogue

9.1.2. Cache de session web

Le cache de session web peut être configuré quand vous utilisez l'un des profils HA, dont le profil **`standalone-ha.xml`**, ou les profils de domaine gérés **`ha`** ou **`full-ha`**. Les éléments les plus couramment configurés sont le mode cache et le nombre de propriétaires de cache pour un cache distribué.

Mode Cache

Le mode cache peut être **`REPL`** (par défaut) ou **`DIST`**.

REPL

Le mode **`REPL`** reproduit tout le cache à un nœud sur deux dans le cluster. Cela constitue l'option la plus sûre, mais elle introduit aussi plus de surcharge de temps.

DIST

Le mode **`DIST`** est semblable au *buddy mode* fourni dans les exécutions précédentes. Il réduit la surcharge de temps en répartissant le cache entre le nombre de nœuds indiqués dans le paramètre **`owners`**. Ce nombre de propriétaires est défini par défaut à **`2`**.

Propriétaires

Le paramètre **`owners`** contrôle combien de nœuds de clusters détiennent des duplicatas de la session. La valeur par défaut est **`2`**.

Rapporter un bogue

9.1.3. Configurer le cache de session web

Par défaut, le cache de session web est **`REPL`**. Si vous souhaitez utiliser le mode **`DIST`**, exécutez les deux commandes suivantes dans 'interface CLI. Si vous utilisez un profil différent, changez le nom de profil dans les commandes. Si vous utilisez un serveur autonome, retirez la partie **`/profile=ha`** des commandes.

Procédure 9.1. Configurer le cache de session web

1. **Modifier le mode cache par défaut en `DIST`.**
   -

```
/profile=ha/subsystem=infinispan/cache-container=web/:write-
attribute(name=default-cache,value=dist)
```

2. **Définir le nombre de propriétaires de cache distribué.**
   La commande suivante définit **5** propriétaires. La valeur par défaut est **2**.

```
/profile=ha/subsystem=infinispan/cache-container=web/distributed-
cache=dist/:write-attribute(name=owners,value=5)
```

3. **Modifier le mode cache par défaut en REPL.**

```
/profile=ha/subsystem=infinispan/cache-container=web/:write-
attribute(name=default-cache,value=repl)
```

4. **Relancer le serveur**
   Après avoir modifié le mode cache du web, vous devez relancer le serveur.

Résultat

Votre serveur est configuré pour une copie de session. Pour utiliser une copie de session dans vos propres applications, veuillez vous référer au sujet suivant : Section 9.1.4, « Activer la copie de session pour votre application ».

Rapporter un bogue

9.1.4. Activer la copie de session pour votre application

Résumé

Pour profiter des fonctionnalités HA (High Availability) de JBoss EAP 6, configurer votre application de manière à ce qu'elle soit distribuable. Cette procédure vous montrera comment y parvenir puis vous expliquera certaines options de configuration avancées que vous pourrez utiliser.

Procédure 9.2. Rendez votre Application Distribuable

1. **Prérequis : indiquer que votre application est distribuable**
   Si votre application n'apparaît pas comme distribuable, ses sessions ne seront jamais distribuées. Ajouter l'élément **`<distributable />`** dans la balise **`<web-app>`** du fichier descriptif **`web.xml`** de votre application. Voici un exemple :

Exemple 9.1. Configuration minimum pour une application distribuable

```
<?xml version="1.0"?>
<web-app  xmlns="http://java.sun.com/xml/ns/j2ee"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee

http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
          version="2.4">

      <distributable/>

</web-app>
```

2. **Modifier le comportement de réplication par défaut si vous le souhaitez.**
Si vous souhaitez changer une des valeurs qui affectent la réplique de session, vous pouvez les changer dans l'élément **<replication-config>** qui est un dépendant de l'élément enfant de l'élément **<jboss-web>** du fichier **jboss-web.xml** de votre application. Pour un exemple donné, ne l'inclure que si vous souhaitez remplacer les valeurs par défaut. L'exemple suivant énumère tous les paramètres par défaut, et est suivi d'un tableau qui explique les options les plus fréquemment modifées.

Exemple 9.2. Valeurs <replication-config> par défaut

```
<!DOCTYPE jboss-web PUBLIC
    "-//JBoss//DTD Web Application 5.0//EN"
    "http://www.jboss.org/j2ee/dtd/jboss-web_5_0.dtd">

<jboss-web>

   <replication-config>
      <cache-name>custom-session-cache</cache-name>
      <replication-trigger>SET</replication-trigger>
      <replication-granularity>ATTRIBUTE</replication-
granularity>
      <use-jk>false</use-jk>
      <max-unreplicated-interval>30</max-unreplicated-interval>
      <snapshot-mode>INSTANT</snapshot-mode>
      <snapshot-interval>1000</snapshot-interval>
      <session-notification-
policy>com.example.CustomSessionNotificationPolicy</session-
notification-policy>
  </replication-config>

</jboss-web>
```

Tableau 9.1. Options communes pour la réplique de session

Option	Description

Option	Description
`<replication-trigger>`	Contrôle quelles conditions doivent déclencher la réplication de données de session dans l'ensemble du cluster. Cette option est utile, quand un objet mutable stocké comme attribut de session est accessible à partir de la session, le conteneur n'a aucun moyen évident de savoir si l'objet a été modifié et s'il doit être reproduit, sauf si la méthode **`setAttribute()`** est appelée directement. **Valeurs valides pour `<replication-trigger>`** **SET_AND_GET** Il s'agit de l'option la plus sécurisée, mais à moindre performance. Les données de session sont toujours répliquées, même si leur contenu a été accédé, sans avoir été modifié. Cette configuration est préservée à but d'héritage uniquement. Éviter cette configuration et définir **`<max-unreplicated-interval>`** à 0, pour avoir le même comportement et une meilleure performance. **SET_AND_NON_PRIMITIVE_GET** La valeur par défaut. Les données de session ne sont répliquées que si on a accédé à un objet de type non primitif. Cela signifie que l'objet n'est pas d'un type Java connu comme **`Integer`**, **`Long`**, ou **`String`**. **SET** Cette option présume que l'application va explicitement appeler **`setAttribute`** sur la session quand les données auront besoin d'être répliquées. Cela empêche la réplication inutile et peut bénéficier à la performance dans son ensemble, mais il y a un problème de sécurité. Quelle que soit la configuration, vous pourrez toujours déclencher la réplication de session en appelant **`setAttribute()`**.
`<replication-granularity>`	Détermine la granularité des données répliquées. La valeur par défaut est **`SESSION`**, mais peut être définie à **`ATTRIBUTE`** à la place, pour augmenter la performance dans les sessions où la plupart des attributs restent inchangés.

Les options suivantes ont rarement besoin d'être modifiées.

Tableau 9.2. Options les moins communément modifiées pour les réplications de session

Option	Description
`<useJK>`	Présumons qu'un équilibreur de charge tel que **`mod_cluster`**, **`mod_jk`**, ou **`mod_proxy`** soit utilisé. La valeur par défaut est **`false`**. Si définie sur **`true`**, le conteneur examine l'ID de session associé à chaque requête et remplace la partie **`jvmRoute`** de l'ID de session en cas de basculement.

Option	Description
<max-unreplicated-interval>	Le plus grand intervalle (en secondes) à attendre après une session avant de déclencher une réplication d'un horodateur de session, même s'il est considéré comme inchangé. Cela permet aux nœuds de cluster d'être informés de l'horodateur de chaque session et aux sessions non répliquées de ne pas expirer de manière incorrecte pendant un basculement. Cela vous permet également de vous fier à une valeur correcte pour les appels à méthode **HttpSession.getLastAccessedTime()** pendant un basculement. Par défaut, aucune valeur n'est indiquée. Cela signifie que la configuration **jvmRoute** du conteneur détermine si le basculement JK est utilisé. Une valeur **0** entraîne une réplication de l'horodateur à chaque accès à la session. Une valeur **-1** n'entraîne une réplication de l'horodateur que si une autre activité entraîne une réplication pendant la requête. Une valeur positive supérieure à **HttpSession.getMaxInactiveInterval()** est traitée comme une erreur de configuration et sera convertie à **0**.
<snapshot-mode>	Indique quand les sessions sont répliquées vers d'autres nœuds. La valeur par défaut est **INSTANT** et l'autre valeur possible est **INTERVAL**. En mode **INSTANT**, les modifications sont répliquées à la fin d'une requête, au moyen du thread de traitement des requêtes. L'option **<snapshot-interval>** est ignorée. En mode **INTERVAL**, une tâche d'arrière-plan s'exécute à l'intervalle indiqué par **<snapshot-interval>**, et réplique les sessions modifiées.
<snapshot-interval>	L'intervalle, en millisecondes, pendant lequel les sessions modifiées doivent être répliquées si elles utilisent **INTERVAL** pour la valeur de **<snapshot-mode>**.
<session-notification-policy>	Nom complet de la classe de l'implémentation de l'interface **ClusteredSessionNotificationPolicy** qui contrôle si les notifications de spécification servlet sont émises vers un **HttpSessionListener**, **HttpSessionAttributeListener**, ou **HttpSessionBindingListener** enregistré.

Rapporter un bogue

9.2. PASSIVATION ET ACTIVATION HTTPSESSION

9.2.1. La passivation et l'activation de session HTTP

La *Passivation* est le processus de contrôle de l'utilisation de la mémoire, qui consiste au retrait de sessions relativement inutiles de la mémoire quand on les stocke en stockage persistant.

L'*Activation* signifie que les données sont extraites d'un stockage persistant et sont renvoyées dans la mémoire.

La passivation a lieu à trois reprises au cours du cycle de vie d'une session HTTP :

- Quand le conteneur réclame la création d'une nouvelle session, si le nombre de sessions actives en cours dépasse une limite configurable, le serveur tentera de passiver certaines sessions pour faire place à une nouvelle session.
- De façon périodique, à un intervalle précis configuré, une tâche d'arrière-plan vérifiera si les sessions doivent être passivées.
- Quand une application web est déployée et qu'une copie de sauvegarde de sessions actives sur d'autres serveurs est acquise par le gestionnaire de session de l'application web récemment déployée, les sessions peuvent être passivées.

Une session est passivée si elle remplit les conditions suivantes :

- La session n'a pas été utilisée pendant une durée plus longue qu'une période d'inactivité configurable maximale.
- Le nombre de sessions actives dépasse un maximum configurable et la session n'a pas été utilisée pendant une durée plus longue qu'une période d'inactivité configurable maximale.

Les sessions sont toujours passivées en utilisant un algorithme LRU (Least Recently Used).

Rapporter un bogue

9.2.2. Configurer la passivation HttpSession dans votre application

Aperçu

La passivation HttpSession est configurée dans le fichier **`WEB_INF/jboss-web.xml`** ou **`META_INF/jboss-web.xml`** de votre application.

Exemple 9.3. Exemple de fichier `jboss-web.xml`

```
<!DOCTYPE jboss-web PUBLIC
    "-//JBoss//DTD Web Application 5.0//EN"
    "http://www.jboss.org/j2ee/dtd/jboss-web_5_0.dtd">

<jboss-web version="6.0"
           xmlns="http://www.jboss.com/xml/ns/javaee"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://www.jboss.com/xml/ns/javaee
http://www.jboss.org/j2ee/schema/jboss-web_6_0.xsd">

   <max-active-sessions>20</max-active-sessions>
   <passivation-config>
      <use-session-passivation>true</use-session-passivation>
      <passivation-min-idle-time>60</passivation-min-idle-time>
      <passivation-max-idle-time>600</passivation-max-idle-time>
   </passivation-config>


</jboss-web>
```

Éléments de configuration de passivation

`<max-active-sessions>`

Nombre maximal de sessions actives autorisé. Si le nombre de sessions gérées par le gestionnaire de sessions dépasse cette valeur et que la passivation est activée, ce dépassement sera passivé sur la base du **`<passivation-min-idle-time>`** configuré. Ensuite, si le nombre de sessions actives dépasse toujours cette limite, les tentatives de création de nouvelles sessions échoueront. La valeur par défaut de **`-1`** ne définit aucune limite quant au nombre maximal de sessions actives.

`<passivation-config>`

Cet élément contient le reste des paramètres de configuration de passivation, en tant qu'éléments enfants.

Eléments enfants `<passivation-config>`

`<use-session-passivation>`

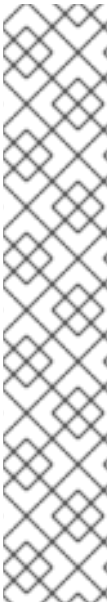
Indique si oui ou non la passivation de session doit être utilisée. La valeur par défaut est **`false`**.

`<passivation-min-idle-time>`

La durée minimum, en secondes, pendant laquelle une session doit être inactive avant que le conteneur n'envisage de la passiver de manière à réduire le nombre de sessions actives pour être conforme à la valeur définie par les max-active-sessions. La valeur par défaut **`-1`** empêche les sessions d'être passivées avant que le temps de **`<passivation-max-idle-time>`** ne se soit écoulé. Les valeurs -1 ou supérieures sont déconseillées si **`<max-active-sessions>`** est configuré.

`<passivation-max-idle-time>`

La durée maximum, en secondes, pendant laquelle une session peut être inactive avant que le conteneur ne tente de la passiver pour économiser de la mémoire. La passivation de ces sessions aura lieu, que le nombre de sessions actives excède **`<max-active-sessions>`** ou pas. Cette valeur doit être inférieure à la configuration **`<session-timeout>`** dans le fichier **`web.xml`**. La valeur par défaut **`-1`** désactive la passivation selon l'inactivité maximum.



NOTE

Le nombre total de sessions en mémoire comprend les sessions répliquées à partir d'autres nœuds de cluster dont l'accès ne se trouve pas sur ce nœud. Tenez-en compte lors de la configuration de **`<max-active-sessions>`**. Le nombre de sessions répliquées à partir d'autres nœuds dépend aussi de savoir si le mode cache **`REPL`** ou **`DIST`** est désactivé. Dans le mode cache **`REPL`**, chaque session est répliquée vers chaque nœud. Dans le mode cache **`DIST`**, chaque session n'est répliquée que vers le nombre de nœuds indiqué par le paramètre **`owner`**. Veuillez vous référer à Section 9.1.2, « Cache de session web » et Section 9.1.3, « Configurer le cache de session web » pour plus de renseignements sur la configuration des modes cache de session.

Prenons comme exemple un cluster à huit nœuds, dont chaque nœud gère les requêtes de 100 utilisateurs. Avec le mode cache **`REPL`**, chaque nœud peut stocker 800 sessions en mémoire. Avec le mode cache **`DIST`** désactivé, et la configuration **`owners`** définie par défaut sur **`2`**, chaque nœud stocke 200 sessions en mémoire.

Rapporter un bogue

9.3. DOMAINE DU COOKIE

9.3.1. Cookie Domain

Le *cookie domain* fait référence à l'ensemble des hôtes capables de lire un cookie à partir du navigateur client qui accède votre application. C'est un mécanisme de configuration pour réduire le risque de voir des tierce-parties accéder à des informations que votre application stocke dans les cookies de navigateur.

La valeur par défaut du domaine de cookie est `/`. Cela signifie que seul l'hôte d'origine peut lire le contenu d'un cookie. Le fait de configurer un domaine de cookie particulier rend le contenu du cookie disponible à un plus grand nombre d'hôtes. Pour configurer le domaine de cookie, veuillez vous référer à Section 9.3.2, « Configurer Cookie Domain ».

Rapporter un bogue

9.3.2. Configurer Cookie Domain

Pour autoriser votre valve SSO à partager un contexte SSO, configurer le domaine de cookie dans la configuration de valve. La configuration suivante autoriserait les applications sur **`http://app1.xyz.com`** et **`http://app2.xyz.com`** à partager un contexte SSO, même si ces applications fonctionnent sur des serveurs différents dans un cluster ou si l'hôte virtuel avec lequel ils sont associés possède plusieurs alias.

Exemple 9.4. Exemple de configuration de domaine de cookie

```
<Valve
className="org.jboss.web.tomcat.service.sso.ClusteredSingleSignOn"
                 cookieDomain="xyz.com" />
```

Rapporter un bogue

9.4. IMPLÉMENTER UN SINGLETON HA

Résumé

La procédure suivante illustre comment déployer un Service qui se trouve dans un decorator de SingletonService et qui est utilisé comme service singleton dans l'ensemble du cluster. Le service active une minuterie programmée, qui n'est lancée qu'à une seule reprise dans le cluster.

Procédure 9.3. Implémenter un Service HA Singleton

1. **Rédiger une application de Service HA Singleton**
   Voici un exemple simple de **`Service`** se trouvant dans le decorator **`SingletonService`** devant être déployé comme service singleton. On trouvera un exemple complet dans le quickstart de **`cluster-ha-singleton`** qui est livré avec Red Hat JBoss Enterprise Application Platform 6. Ce quickstart contient toutes les instructions pour générer et déployer l'application.

   a. **Créer un service.**
      Voici un exemple de service :

```
package org.jboss.as.quickstarts.cluster.hasingleton.service.ejb;

import java.util.Date;
import java.util.concurrent.atomic.AtomicBoolean;

import javax.naming.InitialContext;
import javax.naming.NamingException;

import org.jboss.logging.Logger;
import org.jboss.msc.service.Service;
import org.jboss.msc.service.ServiceName;
import org.jboss.msc.service.StartContext;
import org.jboss.msc.service.StartException;
import org.jboss.msc.service.StopContext;


/**
 * @author <a href="mailto:wfink@redhat.com">Wolf-Dieter Fink</a>
 */
public class HATimerService implements Service<String> {
    private static final Logger LOGGER =
Logger.getLogger(HATimerService.class);
    public static final ServiceName SINGLETON_SERVICE_NAME =
ServiceName.JBOSS.append("quickstart", "ha", "singleton",
"timer");

    /**
     * A flag whether the service is started.
     */
    private final AtomicBoolean started = new
AtomicBoolean(false);

    /**
     * @return the name of the server node
     */
    public String getValue() throws IllegalStateException,
IllegalArgumentException {
        LOGGER.infof("%s is %s at %s",
HATimerService.class.getSimpleName(), (started.get() ? "started"
: "not started"), System.getProperty("jboss.node.name"));
        return "";
    }

    public void start(StartContext arg0) throws StartException {
        if (!started.compareAndSet(false, true)) {
            throw new StartException("The service is still
started!");
        }
        LOGGER.info("Start HASingleton timer service '" +
this.getClass().getName() + "'");

        final String node =
System.getProperty("jboss.node.name");
        try {
            InitialContext ic = new InitialContext();
            ((Scheduler) ic.lookup("global/jboss-cluster-ha-
```

```
singleton-
service/SchedulerBean!org.jboss.as.quickstarts.cluster.hasingleto
n.service.ejb.Scheduler")).initialize("HASingleton timer @" +
node + " " + new Date());
        } catch (NamingException e) {
            throw new StartException("Could not initialize
timer", e);
        }
    }

    public void stop(StopContext arg0) {
        if (!started.compareAndSet(true, false)) {
            LOGGER.warn("The service '" +
this.getClass().getName() + "' is not active!");
        } else {
            LOGGER.info("Stop HASingleton timer service '" +
this.getClass().getName() + "'");
            try {
                InitialContext ic = new InitialContext();
                ((Scheduler) ic.lookup("global/jboss-cluster-ha-
singleton-
service/SchedulerBean!org.jboss.as.quickstarts.cluster.hasingleto
n.service.ejb.Scheduler")).stop();
            } catch (NamingException e) {
                LOGGER.error("Could not stop timer", e);
            }
        }
    }
}
```

b. **Créer un activator qui installe le `Service` en tant que singleton clusterisé.**
La liste suivante est un exemple d'activateurr de service qui installe le **`HATimerService`** comme service singleton clusterisé :

```
package org.jboss.as.quickstarts.cluster.hasingleton.service.ejb;

import org.jboss.as.clustering.singleton.SingletonService;
import org.jboss.logging.Logger;
import org.jboss.msc.service.DelegatingServiceContainer;
import org.jboss.msc.service.ServiceActivator;
import org.jboss.msc.service.ServiceActivatorContext;
import org.jboss.msc.service.ServiceController;


/**
 * Service activator that installs the HATimerService as a
clustered singleton service
 * during deployment.
 *
 * @author Paul Ferraro
 */
public class HATimerServiceActivator implements ServiceActivator
{
    private final Logger log = Logger.getLogger(this.getClass());

    @Override
```

```
    public void activate(ServiceActivatorContext context) {
        log.info("HATimerService will be installed!");

        HATimerService service = new HATimerService();
        SingletonService<String> singleton = new
SingletonService<String>(service,
HATimerService.SINGLETON_SERVICE_NAME);
        /*
         * To pass a chain of election policies to the singleton,
for example,
         * to tell JGroups to prefer running the singleton on a
node with a
         * particular name, uncomment the following line:
         */
        // singleton.setElectionPolicy(new
PreferredSingletonElectionPolicy(new
SimpleSingletonElectionPolicy(), new
NamePreference("node2/cluster")));

        singleton.build(new
DelegatingServiceContainer(context.getServiceTarget(),
context.getServiceRegistry()))
                .setInitialMode(ServiceController.Mode.ACTIVE)
                .install()
        ;
    }
}
```

NOTE

L'exemple de code ci-dessus utilise une classe, **`org.jboss.as.clustering.singleton.SingletonService`**, qui fait partie de l'API privée de JBoss EAP. Une API publique deviendra disponible dans la version EAP 7 et les classes privées seront obsolètes, mais ces classes seront entretenues et rendues disponible pendant toute la durée du cycle de version EAP 6.x.

c. **Créer un fichier ServiceActivator**
Créer un fichier nommé **`org.jboss.msc.service.ServiceActivator`** dans le répertoire **`resources/META-INF/services/`** de l'application. Ajouter une ligne contenant le nom complet de la classe de Service Activator créée dans l'étape précédente.

```
org.jboss.as.quickstarts.cluster.hasingleton.service.ejb.HATimerS
erviceActivator
```

d. **Créer un bean singleton qui implémente une minuterie de singleton à utiliser dans tout le cluster.**
Ce bean Singleton ne doit pas avoir d'interface distante et ne doit pas référencer son interface locale à partir d'un autre EJB d'application. Cela évite une recherche client ou de la part d'un autre composant et assure un parfait contrôle du Singleton par le SingletonService.

i. **Créer une interface de planification.**

```
package
org.jboss.as.quickstarts.cluster.hasingleton.service.ejb;

/**
 * @author <a href="mailto:wfink@redhat.com">Wolf-Dieter
Fink</a>
 */
public interface Scheduler {

    void initialize(String info);

    void stop();

}
```

ii. **Créer le bean Singleton qui implémente la minuterie de singleton à utiliser dans tout le cluster.**

```
package
org.jboss.as.quickstarts.cluster.hasingleton.service.ejb;

import javax.annotation.Resource;
import javax.ejb.ScheduleExpression;
import javax.ejb.Singleton;
import javax.ejb.Timeout;
import javax.ejb.Timer;
import javax.ejb.TimerConfig;
import javax.ejb.TimerService;

import org.jboss.logging.Logger;


/**
 * A simple example to demonstrate a implementation of a
cluster-wide singleton timer.
 *
 * @author <a href="mailto:wfink@redhat.com">Wolf-Dieter
Fink</a>
 */
@Singleton
public class SchedulerBean implements Scheduler {
    private static Logger LOGGER =
Logger.getLogger(SchedulerBean.class);
    @Resource
    private TimerService timerService;

    @Timeout
    public void scheduler(Timer timer) {
        LOGGER.info("HASingletonTimer: Info=" +
timer.getInfo());
    }

    @Override
    public void initialize(String info) {
        ScheduleExpression sexpr = new ScheduleExpression();
        // set schedule to every 10 seconds for demonstration
```

```
        sexpr.hour("*").minute("*").second("0/10");
        // persistent must be false because the timer is
started by the HASingleton service
        timerService.createCalendarTimer(sexpr, new
TimerConfig(info, false));
    }

    @Override
    public void stop() {
        LOGGER.info("Stop all existing HASingleton timers");
        for (Timer timer : timerService.getTimers()) {
            LOGGER.trace("Stop HASingleton timer: " +
timer.getInfo());
            timer.cancel();
        }
    }
}
```

2. **Démarrez chaque instance de JBoss EAP 6 avec le clustering activé.**

   Pour activer le clustering dans les serveurs autonomes, démarrer chaque serveur par le profil **HA**, en utilisant un nom de code unique et un offset (décallage) de port pour chaque instance comme suit :

   - Dans Linux, on utilise la commande suivante pour démarrer les serveurs :

     ```
     EAP_HOME/bin/standalone.sh --server-config=standalone-ha.xml -
     Djboss.node.name=UNIQUE_NODE_NAME -Djboss.socket.binding.port-
     offset=PORT_OFFSET
     ```

     Exemple 9.5. Démarrage de multiples serveurs autonomes sur Linux

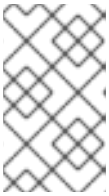
     ```
     $ EAP_HOME/bin/standalone.sh --server-config=standalone-ha.xml
     -Djboss.node.name=node1
     $ EAP_HOME/bin/standalone.sh --server-config=standalone-ha.xml
     -Djboss.node.name=node2 -Djboss.socket.binding.port-offset=100
     ```

   - Dans Microsoft Windows, on utilise la commande suivante pour démarrer les serveurs :

     ```
     EAP_HOME\bin\standalone.bat --server-config=standalone-ha.xml -
     Djboss.node.name=UNIQUE_NODE_NAME -Djboss.socket.binding.port-
     offset=PORT_OFFSET
     ```

     Exemple 9.6. Démarrage de multiples serveurs autonomes dans Microsoft Windows

     ```
     C:> EAP_HOME\bin\standalone.bat --server-config=standalone-
     ha.xml -Djboss.node.name=node1
     C:> EAP_HOME\bin\standalone.bat --server-config=standalone-
     ha.xml -Djboss.node.name=node2 -Djboss.socket.binding.port-
     offset=100
     ```



NOTE

Si vous ne souhaitez pas utiliser des arguments en ligne de commande, vous pouvez configurer le fichier **`standalone-ha.xml`** pour que chaque instance de serveur puisse se lier à une interface séparée.

3. **Déployer l'application dans les serveurs**
   Si vous utilisez la commande Maven suivante pour déployer votre application dans un serveur autonome qui exécute sur les ports par défaut.

   ```
   mvn clean install jboss-as:deploy
   ```

   Pour déployer des serveurs supplémentaires, indiquer le nom du serveur. S'il est sur un hôte différent, passez le nom d'hôte et le numéro de port dans la ligne de commande :

   ```
   mvn clean package jboss-as:deploy -Djboss-as.hostname=localhost -
   Djboss-as.port=10099
   ```

   Voir le quickstart **`cluster-ha-singleton`** fourni dans JBoss EAP 6 pour la configuration et les détails de déploiement de Maven.

Rapporter un bogue

9.5. APPLICATION APACHE MOD_CLUSTER-MANAGER

9.5.1. L'application mod_cluster-manager

L'application mod_cluster-manager est une page web d'administration qui est disponible sur le serveur HTTP Apache. Elle est utilisée pour surveiller les nœuds de worker connectés et pour effectuer diverses tâches d'administration comme l'activation/la désactivation de contextes et pour configurer les propriétés d'équilibrage de charge de nœuds de worker dans un cluster.

Rapporter un bogue

9.5.2. L'application mod_cluster-manager

L'application mod_cluster-manager peut être utilisée pour effectuer un certain nombre de tâches administratives sur les noeuds de worker.

Le schéma ci-dessous montre la page web de l'application mod_cluster-manager contenant des annotations qui mettent en relief les composants importants et les options d'administration sur cette page.

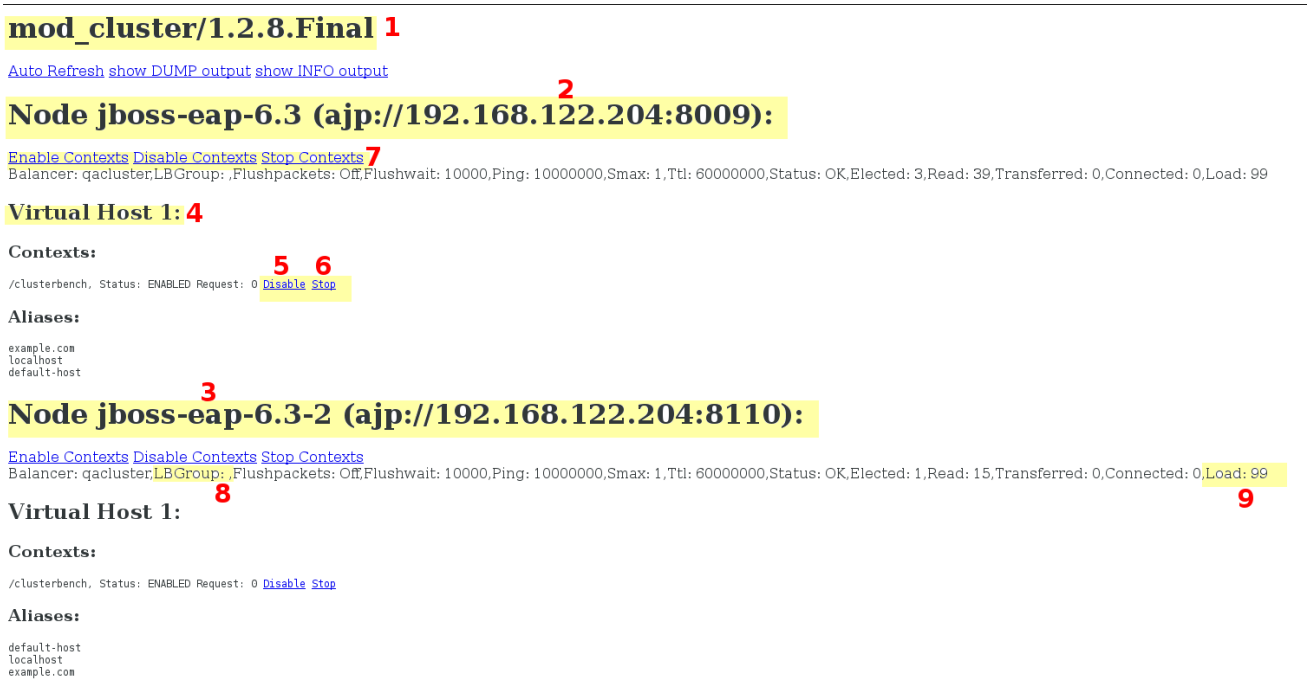
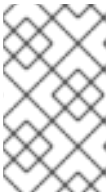


Figure 9.1. La page web d'administration de mod_cluster

Les annotations sont expliquées ci-dessous :

- **`[1] mod_cluster/1.2.8.Final`** : indique la version de la bibliothèque de mod_cluster native
- **`[2] ajp://192.168.122.204:8099`**: indique le protocole utilisé (un parmi AJP, HTTP, HTTPS), le nom d'hôte ou l'adresse IP d'un noeud de worker et le port
- **`[3] jboss-eap-6.3-2`**: indique la JVMRoute du noeud de worker.
- **`[4] Virtual Host 1`**: indique le(s) hôte(s) virtuel(s) configuré(s) sur le noeud de worker
- **`[5] Disable`** : il s'agit d'une option administrative qui peut être utilisée pour désactiver la création de nouvelles sessions dans un contexte particulier. Cependant, les sessions courantes ne sont pas désactivées et demeurent intactes.
- **`[6] Stop`** : il s'agit d'une option administrative qui peut être utilisée pour stoper la redirection de demandes de sessions au contexte. Les sessions restantes seront reprises par défaut par un autre noeud à moins que la propriété **`sticky-session-force`** soit définie sur "true"
- **`[7] Enable Contexts Disable Contexts Stop Contexts`** (Activer les contextes Désativer les contextes Stopper les contextes) : opérations qui peuvent être effectuées sur tout le noeud. En sélectionnant une de ces options, vous affectez tous les contextes de noeuds dans tous ses hôtes virtuels.
- **`[8] Load balancing group (LBGroup)`**: la propriété **`load-balancing-group`** est définie dans le sous-système mod_cluster dans la configuration EAP pour regrouper tous les noeuds de worker en groupes d'équilibrage de charges personnalisés. Le groupe d'équilibrage de charge (LBGroup) est un champ informel qui donne des informations sur tous les groupes d'équilibrage de charge définis. Si ce champ n'est pas défini, alors tous les noeuds de worker sont regroupés en un seul groupe d'équilibrage de charges par défaut.



NOTE

Il s'agit d'un champ informationnel qui ne peut donc pas être utilisé pour définir la propriété **`load-balancing-group`**. Cette propriété doit être définie dans le sous-système mod_cluster dans la configuration EAP.

- **`[9] Load (value)`** : indique la charge sur le noeud de worker. Les « load factors » ou facteurs de charge sont évalués ainsi :

```
-load > 0  : un facteur de charge de 1 indique une surcharge du
noeud de worker, 100 indique qu'il est libre de toute charge.
-load = 0  : un facteur de charge de 0 indique que le noeud de
worker est en attente, c-a-d qu'aucune demande de session ne sera
redirigée vers ce noeud sauf si les autres noeuds de worker ne sont
pas disponibles
-load = -1 : un facteur de charge de -1 indique qu'il y a une erreur
dans l'état du noeud de worker.
-load = -2 : un facteur de charge de -2 indique que le noeud de
worker est en cours de CPing/CPong et se trouve dans un état de
transition
```

Rapporter un bogue

CHAPITRE 10. CDI

10.1. CDI

10.1.1. CDI

- Section 10.1.2, « CDI (Contexts and Dependency Injection) »
- Section 10.1.5, « Relation entre Weld, Seam 2, Seam 3, et JavaServer Faces »
- Section 10.1.3, « Avantages de CDI »

Rapporter un bogue

10.1.2. CDI (Contexts and Dependency Injection)

DCI (Context and Dependency Injection) est une spécification conçue pour permettre aux composants EJB 3.0 "d'être utilisés comme beans JSF (java Server Faces), réunissant les deux modèles de composants et permettant la simplification du modèle de programmation des applications basées-web dans Java". La citation suivante est extraite de la spécification JSR-299, que l'on peut trouver à http://www.jcp.org/en/jsr/detail?id=299.

JBoss EAP 6 inclut Weld, qui est l'implementation de référence de JSR-299. Pour obtenir plus d'informations sur l'injection de dépendance type-safe, voir Section 10.1.4, « Injection de dépendance de Type-safe ».

Rapporter un bogue

10.1.3. Avantages de CDI

- CDI simplifie et rétrécit votre base code en remplaçant des gros morceaux de code par des annotations.
- CDI est flexible, ce qui vous permet de désactiver ou d'activer des injections et des événements, d'utiliser des autres beans, ou d'injecter des objets non-CDI facilement.
- Il est facile d'utiliser votre ancien code avec CDI. Vous n'avez qu'à y inclure un **`beans.xml`** dans votre répertoire **`META-INF/`** ou **`WEB-INF/`**. Le fichier peut être vide.
- CDI simplifie le packaging et les déploiements, et réduit le montant d'XML que vous aurez besoin d'ajouter à vos déploiements.
- CDI propose une gestion du cycle de vie par contexte. Vous pouvez lier des injections aux requêtes, sessions, conversations, ou contextes personnalisés.
- CDI propose une injection de dépendance type-safe, qui est plus sécurisée et facile à débogger qu'une injection basée string.
- CDI découple les intercepteurs des beans.
- CDI fournit une notification d'événements complexe.

Rapporter un bogue

10.1.4. Injection de dépendance de Type-safe

Avant JSR-299 et DCI, la seule façon d'injecter des dépendances dans Java était par chaînes. Cela portait à erreur. DCI a introduit la possibilité d'injecter des dépendances d'une manière plus sûre.

Pour plus d'information sur CDI, voir Section 10.1.2, « CDI (Contexts and Dependency Injection) ».

Rapporter un bogue

10.1.5. Relation entre Weld, Seam 2, Seam 3, et JavaServer Faces

Le but de *Seam 2* était d'unifier les beans gérés Enterprise Java Beans (EJBs) et JavaServer Faces (JSF).

JavaServer Faces (JSF) implémente JSR-314. Il s'agit d'un API qui construit les interfaces utilisateur côté serveur. *JBoss Web Framework Kit* inclut *RichFaces*, qui est une implémentation de JavaServer Faces et d'AJAX.

Weld est l'implémentation de référence de *Contexts and Dependency Injection (CDI)*, définie dans JSR-299. Weld est inspiré de Seam 2 et d'autres frameworks d'injection de dépendances. Weld est inclus dans JBoss EAP 6.

Rapporter un bogue

10.2. UTILISER CDI

10.2.1. Premières étapes

10.2.1.1. Activer CDI

Résumé

Le CDI (Contexts and Dependency Injection) est l'une des technologies de base de la JBoss EAP 6, et est activé par défaut. Si, pour une raison ou une autre, il était désactivé et que vous souhaitiez l'activer, veuillez suivre le procédé suivant :

Procédure 10.1. Activer le CDI dans JBoss EAP 6

1. **Vérifier si les informations du sous-système sont dé-commentées dans le fichier de configuration.**
   On peut désactiver un sous-système en dé-commentant la section qui convient dans les fichiers de configuration **`domain.xml`** ou **`standalone.xml`**, ou en supprimant la section qui convient.

   Pour trouver le sous-système dans ***`EAP_HOME`*`/domain/configuration/domain.xml`** ou ***`EAP_HOME`*`/standalone/configuration/standalone.xml`**, le chercher dans les strings suivants. S'il existe, il se trouve dans la section <extensions>.

   ```
   <extension module="org.jboss.as.weld"/>
   ```

   La ligne suivante doit se trouver dans le profil que vous utilisez. Les profils se trouvent dans des éléments de <profile> dans la section <profiles>.

   ```
   <subsystem xmlns="urn:jboss:domain:weld:1.0"/>
   ```

2. **Avant de modifier un fichier, arrêter JBoss EAP 6.**
   JBoss EAP 6 modifie les fichiers de configuration pendant le temps d'exécution, vous devez donc stopper le serveur avant de modifier directement les fichiers de configuration.

3. **Modifer le fichier de configuration pour restaurer le sous-système CDI.**
   Si le sous-système CDI est dé-commenté, supprimer les commentaires.

   S'il était supprimé totalement, le restaurer en ajoutant cette ligne au fichier dans une nouvelle ligne directement au dessus de la balise </extensions> :

   ```
   <extension module="org.jboss.as.weld"/>
   ```

4. Vous devez également ajouter la ligne suivante dans le profil concerné qui se trouve dans la section <profiles>.

   ```
   <subsystem xmlns="urn:jboss:domain:weld:1.0"/>
   ```

5. **Démarrer à nouveau JBoss EAP 6.**
   Démarrer JBoss EAP 6 avec votre configuration mise à jour.

Résultat

JBoss EAP 6 démarre avec le sous-système CDI activé.

Rapporter un bogue

10.2.2. Utiliser CDI pour développer une application

10.2.2.1. Utiliser le CDI pour développer une application

Introduction

Le CDI (Contexts and Dependency Injection) vous offre une très grande flexibilité pour développer des applications, réutiliser des codes, adapter votre code au déploiement ou à l'exécution et tester des unités. JBoss EAP 6 comprend Weld, l'implémentation de référence de CDI. Ces tâches vous montrent comment utiliser CDI dans vos applications Enterprise.

- Section 10.2.1.1, « Activer CDI »
- Section 10.2.2.2, « CDI avec le code existant »
- Section 10.2.2.3, « Exclure les beans du processus de balayage »
- Section 10.2.2.4, « Utiliser une injection pour étendre une implémentation »
- Section 10.2.3.3, « Utiliser un qualificateur pour résoudre une injection ambiguë »
- Section 10.2.7.4, « Remplacer une injection par une alternative »
- Section 10.2.7.2, « Utilisation des Named Beans »
- Section 10.2.6.1, « Gestion du cycle de vie d'un bean »
- Section 10.2.6.2, « Utilisation d'une méthode Producer »
- Section 10.2.10.2, « Utiliser les intercepteurs avec le CDI »

- Section 10.2.8.2, « Utilisation des stéréotypes »
- Section 10.2.9.2, « Appliquer et observer des événements »

Rapporter un bogue

10.2.2.2. CDI avec le code existant

Presque chaque classe concrète de Java ayant un constructeur sans paramètre, ou un constructeur désigné avec l'annotation @Inject, est un bean. La seule chose que vous devez faire avant de commencer à injecter des beans est de créer un fichier appelé **`beans.xml`** dans le répertoire **`META-INF/`** ou **`WEB-INF/`** de votre archive. Ce fichier peut rester vide.

Procédure 10.2. Utiliser les anciens beans dans les applications CDI

1. **Groupez vos beans dans une archive.**
   Groupez vos beans dans une archive JAR ou WAR.

2. **Inclure un fichier `beans.xml` dans votre archive.**
   Mettez un fichier **`beans.xml`** dans votre répertoire d'archive JAR **`META-INF/`** ou d'archive WAR **`WEB-INF/`**. Ce fichier peut rester vide.

Résultat :

Vous pouvez utiliser ces beans avec CDI. Le conteneur peut créer et détruire des instances de vos beans et les associer à un contexte désigné, les injecter dans les autres beans, les utiliser dans des expressions EL, les spécialiser par des annotations de qualificateur et y ajouter des intercepteurs et des décorateurs, sans aucune modification de votre code existant. Dans certains cas, vous devrez peut-être ajouter des annotations.

Rapporter un bogue

10.2.2.3. Exclure les beans du processus de balayage

Résumé

Une des caractéristiques de Weld, l'implémentation de CDI de JBoss EAP 6, est la possibilité d'exclure des classes de balayage de votre archive, d'avoir des événements de cycle de vie conteneur émis, et déployés comme beans. Cela ne fait pas partie de la spécification JSR-299.

> **Exemple 10.1. Exclure des packages de votre bean.**
>
> L'exemple suivant possède plusieurs balises <weld:exclude>.
>
> 1. Le premier exclut toutes les classes Swing.
> 2. Le second exclut les classes Google Web Toolkit si le Google Web Toolkit n'est pas installé.
> 3. Le troisième exclut les classes qui finissent dans la chaîne **`Blether`** (en utilisant une expression régulière), si la propriété de système verbosity est définie comme **`low`**.
> 4. Le quatrième exclut les classes JSF (Java Server Faces) si les classes Wicket sont présentes et la propriété de système viewlayer n'est pas définie.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://java.sun.com/xml/ns/javaee"
```

```
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:weld="http://jboss.org/schema/weld/beans"
       xsi:schemaLocation="
          http://java.sun.com/xml/ns/javaee
http://docs.jboss.org/cdi/beans_1_0.xsd
          http://jboss.org/schema/weld/beans
http://jboss.org/schema/weld/beans_1_1.xsd">

    <weld:scan>

        <!-- Don't deploy the classes for the swing app! -->
        <weld:exclude name="com.acme.swing.**" />

        <!-- Don't include GWT support if GWT is not installed -->
        <weld:exclude name="com.acme.gwt.**">
            <weld:if-class-available name="!com.google.GWT"/>
        </weld:exclude>

        <!--
            Exclude classes which end in Blether if the system property
verbosity is set to low
            i.e.
              java ... -Dverbosity=low
        -->
        <weld:exclude pattern="^(.*)Blether$">
            <weld:if-system-property name="verbosity" value="low"/>
        </weld:exclude>

       <!--
             Don't include JSF support if Wicket classes are present,
and the viewlayer system
             property is not set
        -->
        <weld:exclude name="com.acme.jsf.**">
            <weld:if-class-available name="org.apache.wicket.Wicket"/>
            <weld:if-system-property name="!viewlayer"/>
        </weld:exclude>
    </weld:scan>
</beans>
```

La spécification officielle des options de configuration spéciale Weld se trouve dans http://jboss.org/schema/weld/beans_1_1.xsd.

Rapporter un bogue

10.2.2.4. Utiliser une injection pour étendre une implémentation

Résumé

Vous pouvez utiliser une injection pour ajouter ou modifier une caractéristique de votre code existant. Cet exemple vous montre comment ajouter une capacité de traduction à une classe existante. La traduction est une caractéristique hypothétique et la façon dont elle est implémentée dans cet exemple est par pseudo-code, et elle est fournie à titre illustratif uniquement.

L'exemple part du principe que vous avez déjà une classe Welcome, qui possède une méthode

buildPhrase. La méthode **buildPhrase** utilise le nom d'une ville comme paramètre et crée une formule telle que "Welcome to Boston". Votre objectif est de créer une version de cette classe **Welcome** pouvant traduire cette formule d'accueil dans une autre langue.

Exemple 10.2. Injecter un Bean Translator dans la classe Welcome.

Le pseudo-code suivant injecte un objet **Translator** hypothétique dans la classe **Welcome**. L'objet **Translator** peut être un bean sans état EJB ou un autre type de bean, pouvant traduire des phrases d'une langue à une autre. Ainsi, le **Translator** est utilisé pour traduire la formule d'accueil complète, sans avoir à modifier la classe **Welcome** d'origine. Le **Translator** est injecté avant que la méthode **buildPhrase** ne soit implémentée.

L'échantillon de code ci-dessous est un exemple de classe Welcome de Traduction.

```
public class TranslatingWelcome extends Welcome {

    @Inject Translator translator;

    public String buildPhrase(String city) {
        return translator.translate("Welcome to " + city + "!");
    }
    ...
}
```

Rapporter un bogue

10.2.3. Dépendances ambigues ou non satisfaites

10.2.3.1. Dépendances ambigües et non-satisfaites

On parle de dépendances ambigües quand le conteneur ne peut pas résoudre une injection dans un bean précisément.

On parle de dépendances non-satisfaites quand le conteneur ne peut pas résoudre une injection dans aucun bean.

Le conteneur suit les étapes suivantes pour essayer de résoudre les dépendances :

1. Il résout les annotations de qualificateur sur tous les beans qui implémentent le bean d'un point d'injection.
2. Il filre les beans désactivés. Les beans désactivés sont des beans @Alternative qui ne sont pas explicitement activés.

Dans le cas d'une dépendance ambigüe ou non satisfaite, le conteneur stoppe le déploiement et envoie une exception.

Pour fixer une dépendance ambigüe, voir Section 10.2.3.3, « Utiliser un qualificateur pour résoudre une injection ambiguë ».

Rapporter un bogue

10.2.3.2. Qualificateurs

Un qualificateur est une annotation qui relie un bean à un autre. Cela vous permet d'indiquer avec précision quel bean vous souhaitez injecter. Les qualificateurs possèdent une rétention et une cible, définies comme dans l'exemple ci-dessous.

Exemple 10.3. Définit les qualificateurs `@Synchronous` et `@Asynchronous`

```
@Qualifier
@Retention(RUNTIME)
@Target({TYPE, METHOD, FIELD, PARAMETER})
public @interface Synchronous {}
```

```
@Qualifier
@Retention(RUNTIME)
@Target({TYPE, METHOD, FIELD, PARAMETER})
public @interface Asynchronous {}
```

Exemple 10.4. Utiliser les qualificateurs `@Synchronous` et `@Asynchronous`

```
@Synchronous
public class SynchronousPaymentProcessor implements PaymentProcessor {

   public void process(Payment payment) { ... }

}
```

```
@Asynchronous
public class AsynchronousPaymentProcessor implements PaymentProcessor {

   public void process(Payment payment) { ... }
}
```

Rapporter un bogue

10.2.3.3. Utiliser un qualificateur pour résoudre une injection ambiguë

Résumé

Cette tâche indique une injection ambiguë et la retire par un qualificateur. Pour en savoir plus sur les injections ambiguës, voir Section 10.2.3.1, « Dépendances ambigües et non-satisfaites ».

Exemple 10.5. Injections ambiguës

Il y a deux implémentations de la classe **`Welcome`**, une qui traduit, et l'autre non. Dans cette situation, l'injection ci-dessous est ambiguë et a besoin d'être spécifiée pour qu'on puisse utiliser la classe de traduction **`Welcome`**.

```
public class Greeter {
  private Welcome welcome;

  @Inject
  void init(Welcome welcome) {
```

```
    this.welcome = welcome;
  }
  ...
}
```

Procédure 10.3. Résoudre une injection ambiguë avec un qualificateur

1. **Créer une annotation de qualificateur nommée `@Translating`.**

```
@Qualifier
@Retention(RUNTIME)
@Target({TYPE,METHOD,FIELD,PARAMETERS})
public @interface Translating{}
```

2. **Annoter votre classe `Welcome` par l'annotation `@Translating`.**

```
@Translating
public class TranslatingWelcome extends Welcome {
    @Inject Translator translator;
    public String buildPhrase(String city) {
        return translator.translate("Welcome to " + city + "!");
    }
    ...
}
```

3. **Demander la classe de traduction `Welcome` dans votre injection.**
   Vous devez demander une implémentation explicitement qualifiée, qui ressemble au modèle de la méthode de fabrique. L'ambiguité se résout au point d'injection.

```
public class Greeter {
  private Welcome welcome;
  @Inject
  void init(@Translating Welcome welcome) {
    this.welcome = welcome;
  }
  public void welcomeVisitors() {
    System.out.println(welcome.buildPhrase("San Francisco"));
  }
}
```

Résultat

`TranslatingWelcome` est utilisé, et il n'y a pas d'ambiguité.

Rapporter un bogue

10.2.4. Beans gérés

10.2.4.1. Les beans gérés

Avant Java EE 6, il y n'avait aucun définition précise du terme *bean* dans la plate-forme Java EE. Il y avait plusieurs concepts que l'on nommait « bean » dans les spécifications de Java EE, comme les EJB

beans et les JSF managed beans. Les frameworks de tierce partie comme Spring et Seam ont commencé à nous donner leur avis sur ce que l'on entend par **bean**.

Java EE 6 a établi une définition commune dans la spécification de Managed Beans. Les Managed Beans sont définis comme des objets gérés par conteneur avec des restrictions minimales de programmation, connus aussi sous le sigle POJO (Plain Old Java Object). Ils supportent un petit ensemble de services de base, tels que l'injection de ressources, les rappels de cycle de vie et les intercepteurs. Les spécifications de leurs accompagnateurs, comme les EJB et CDI, s'appuient sur ce modèle de base.

À part quelques exceptions, chaque classe concrète de Java qui a un constructeur sans paramètre (ou un constructeur désigné avec l'annotation **@Inject**) est un bean. Cela inclut chaque JavaBean et chaque bean de session EJB. La seule exigence pour activer les services mentionnés dans les beans est qu'ils résident dans une archive (un JAR ou un module Java EE comme un WAR ou JAR EJB) qui contienne un fichier spécial marqueur : **META-INF/beans.xml**.

Rapporter un bogue

10.2.4.2. Types de Classes qui ne sont pas des Beans

Un bean géré est une classe Java. Le cycle de base et la sémantique d'un bean géré est définie par la spécification de beans gérés. Vous pouvez explicitement déclarer un bean géré en annotant la classe bean **@ManagedBean**, mais en CDI cela n'est pas nécessaire. Conformément à la spécification, le conteneur CDI traite n'importe quelle classe qui remplit les conditions suivantes en tant que bean géré :

- Il s'agit d'une classe interne non-statique
- Il ne s'agit pas d'une classe concrète, ou bien elle est annotée par **@Decorator**
- Elle n'est pas annotée par une annotation définie par un composant EJB, ni déclarée comme classe bean EJB dans **ejb-jar.xml**.
- Elle n'implémente pas l'interface **javax.enterprise.inject.spi.Extension**.
- Elle possède soit un constructeur sans paramètre soit un constructeur annoté avec **@Inject**.

L'ensemble de types de beans non-restraints d'un bean géré contient la classe de bean, toutes les superclasses et interfaces qu'il implémente directement ou indirectement.

Si un bean géré contient un champ public, il devra avoir comme scope par défaut @Dependent.

Rapporter un bogue

10.2.4.3. Utiliser le CDI pour injecter un objet dans un bean

Quand votre archive de déploiement inclut un fichier **META-INF/beans.xml** ou **WEB-INF/beans.xml**, chaque objet de votre déploiement peut être injecté par le CDI.

Cette procédure vous explique les moyens les plus courants d'injecter des objets dans d'autres objets.

1. **Injecter un objet dans n'importe quelle partie de bean par l'annotation @Inject.**
   Pour obtenir une instance de classe, dans votre bean, annotez ce champ avec **@Inject**.

   > **Exemple 10.6. Injecter une instance TextTranslator dans un TranslateController**

```
public class TranslateController {

   @Inject TextTranslator textTranslator;
   ...
```

2. **Utiliser les méthodes de l'objet injecté**
   Vous pouvez directement utiliser les méthodes de vos objets injectés. Partez du principe que **`TextTranslator`** possède une méthode **`translate`**.

   Exemple 10.7. Utiliser les méthodes de l'objet injecté

```
// in TranslateController class

public void translate() {

   translation = textTranslator.translate(inputText);

}
```

3. **Utiliser l'injection dans le contructeur d'un bean**
   Vous pouvez injecter des objets dans le constructeur d'un bean, ou bien, vous pouvez utiliser une fabrique ou un localisateur de service pour les créer.

   Exemple 10.8. Utiliser l'injection pour la construction d'un bean

```
public class TextTranslator {

   private SentenceParser sentenceParser;

   private Translator sentenceTranslator;



   @Inject

   TextTranslator(SentenceParser sentenceParser, Translator
sentenceTranslator) {

      this.sentenceParser = sentenceParser;

      this.sentenceTranslator = sentenceTranslator;

   }

   // Methods of the TextTranslator class
   ...
}
```

4. **Utiliser l'interface `Instance(<T>)` pour obtenir les instances de façon programmatique.**

L'interface **`Instance`** peut renvoyer une instance de TextTranslator si paramétrée par le type de bean.

Exemple 10.9. Obtenir une instance par programmation

```
@Inject Instance<TextTranslator> textTranslatorInstance;

...

public void translate() {

   textTranslatorInstance.get().translate(inputText);

}
```

Résultat :

Lorsque vous injectez un objet dans un bean, toutes les méthodes et les propriétés de l'objet sont disponibles pour votre bean. Si vous injectez dans le constructeur de votre bean, les instances des objets injectés sont créés quand le constructeur de votre bean est appelé, à moins que l'injection ne fasse référence à une instance qui existe déjà. Par exemple, une nouvelle instance ne serait pas créée si vous injectiez un bean basé-session pendant la durée de la session.

Rapporter un bogue

10.2.5. Contextes, Scopes et Dépendances

10.2.5.1. Contextes et scopes

Un contexte, en terme de CDI, est une zone de stockage qui contient des instances de beans associées à un scope particulier.

Un scope est un lien entre un bean et un contexte. Un scope/context peut avoir un cycle de vie particulier. Il existe plusieurs scopes pré-définis, et vous pouvez créer vos propres scopes. Voici des exemples de scopes pré-définis **`@RequestScoped`**, **`@SessionScoped`**, et **`@ConversationScope`**.

Rapporter un bogue

10.2.5.2. Contextes disponibles

Tableau 10.1. Contextes disponibles

Contexte	Description
@Dependent	Le bean est lié au cycle de vie du bean qui contient la référence.
@ApplicationScoped	Lié au cycle de vie de l'application.
@RequestScoped	Lié au cycle de vie de la requête.

Contexte	Description
@SessionScoped	Lié au cycle de vie de la session.
@ConversationScoped	Lié au cycle de vie de la conversation. Le scope de la conversation correspond aux longueurs de la requête et de la session, et est contrôlé par l'application.
Scopes personnalisés	Si les contextes ne correspondent pas à vos besoins, vous pourrez définir des scopes personnalisés.

Rapporter un bogue

10.2.6. Cycle de vie d'un bean

10.2.6.1. Gestion du cycle de vie d'un bean

Résumé

Cette tâche vous montre comment sauvegarder un bean pendant la durée de vie d'une requête. Il existe plusieurs autres scopes, et vous pouvez définir vos propres scopes.

Le scope par défaut d'un bean injecté est **`@Dependent`**. Cela veut dire que le cycle de vie du bean dépend du cycle de vie du bean qui contient la référence. Pour plus d'informations, voir Section 10.2.5.1, « Contextes et scopes ».

Procédure 10.4. Gestion du cycle de vie d'un bean

1. **Annoter le bean par le scope qui correspond au scope désiré.**

```
@RequestScoped
@Named("greeter")
public class GreeterBean {
  private Welcome welcome;
  private String city; // getter & setter not shown
  @Inject   void init(Welcome welcome) {
    this.welcome = welcome;
  }
  public void welcomeVisitors() {
    System.out.println(welcome.buildPhrase(city));
  }
}
```

2. **Quand votre bean est utilisé dans la vue JSF, il contient un état.**

```
<h:form>
  <h:inputText value="#{greeter.city}"/>
  <h:commandButton value="Welcome visitors" action="#
{greeter.welcomeVisitors}"/>
</h:form>
```

Résultat :

Votre bean est sauvegardé dans le contexte lié au scope que vous spécifiez, et dure aussi longtemps que le scope est applicable.

- Section 10.2.13.1, « Proxys Bean »
- Section 10.2.13.2, « Utiliser un Proxy ou une Injection »

Rapporter un bogue

10.2.6.2. Utilisation d'une méthode Producer

Résumé

Cette tâche vous montre comment utiliser les méthodes Producer pour créer un certain nombre d'objets qui ne sont pas des beans, pour l'injection.

Exemple 10.10. Utiliser une méthode Producer au lieu d'une méthode Alternative, pour permettre le polymorphisme suite au déploiement.

L'annotation **`@Preferred`** dans l'exemple donné est une annotation qualifiante. Pour plus d'informations sur les qualificateurs, veuillez vous référer à : Section 10.2.3.2, « Qualificateurs ».

```
@SessionScoped
public class Preferences implements Serializable {
   private PaymentStrategyType paymentStrategy;
   ...
   @Produces @Preferred
   public PaymentStrategy getPaymentStrategy() {
       switch (paymentStrategy) {
           case CREDIT_CARD: return new CreditCardPaymentStrategy();
           case CHECK: return new CheckPaymentStrategy();
           default: return null;
       }
   }
}
```

Le point d'injection suivant a les mêmes type et qualificateur d'annotations que la méthode de Producer, donc il se résout à la méthode Producer en utilisant les règles usuelles d'injection CDI. La méthode Producer est appelée par le conteneur pour obtenir une instance au service de ce point de l'injection.

```
@Inject @Preferred PaymentStrategy paymentStrategy;
```

Exemple 10.11. Assigner un scope à une méthode Producer

L'étendue par défaut d'une méthode Producer est **`@Dependent`**. Si vous attribuez un scope à un bean, celle-ci est destinée au contexte approprié. La méthode Producer dans cet exemple n'est appelée qu'une fois par session.

```
@Produces @Preferred @SessionScoped
public PaymentStrategy getPaymentStrategy() {
   ...
```

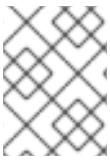
```
}
```

Exemple 10.12. Utiliser une injection à l'intérieur d'une méthode Producer

Les objets instanciés directement par une application ne peuvent pas profiter de l'injection de dépendance, et n'ont pas d'intercepteurs. Cependant, vous pouvez utiliser l'injection de dépendance dans la méthode Producer pour obtenir des instances de bean.

```
@Produces @Preferred @SessionScoped
public PaymentStrategy getPaymentStrategy(CreditCardPaymentStrategy
ccps,
                                          CheckPaymentStrategy cps )
{
   switch (paymentStrategy) {
      case CREDIT_CARD: return ccps;
      case CHEQUE: return cps;
      default: return null;
   }
}
```

Si vous injectez un bean de demande (request-scoped) dans un Producer de session (session-scoped), la méthode Producer favorise l'instance actuelle de demande (request-scoped) dans le scope de la session. Ce n'est certainement pas le comportement souhaité, alors soyez prudent lorsque vous utilisez une méthode Producer de cette façon.



NOTE

Le scope de la méthode Producer ne provient pas du bean qui déclare la méthode producer.

Résultat

Les méthodes Producer vous permettent d'injecter des objets non-bean et de changer votre code de façon dynamique.

Rapporter un bogue

10.2.7. Beans nommés et beans alternatifs

10.2.7.1. Named Beans

Un Named Bean est un bean nommé qui utilise l'annotation **`@Named`**. Nommer un bean vous permet de l'utiliser directement dans JSF (Java Server Faces).

L'annotation **`@Named`** prend un paramètre optionnel, qui correspond au nom du bean. Si le paramètre est omis, le nom du bean en minuscules sera utilisé.

Rapporter un bogue

10.2.7.2. Utilisation des Named Beans

1. **Utiliser l'annotation `@Named` pour assigner un nom à un bean.**

```
@Named("greeter")
public class GreeterBean {
  private Welcome welcome;

  @Inject
  void init (Welcome welcome) {
    this.welcome = welcome;
  }

  public void welcomeVisitors() {
    System.out.println(welcome.buildPhrase("San Francisco"));
  }
}
```

   Le nom du bean en soi est une option. Si vous l'oubliez, le bean sera nommé par rapport à son nom de classe, avec la première lettre non capitalisée. Dans l'exemple ci-dessus, le nom par défaut serait donc **`greeterBean`**.

2. **Utiliser le bean nommé dans une vue JSF**

```
<h:form>
  <h:commandButton value="Welcome visitors" action="#
{greeter.welcomeVisitors}"/>
</h:form>
```

Résultat :

Votre bean nommé correspond à une action dans le contrôle de votre vue JSF, et à un minimum de code.

Rapporter un bogue

10.2.7.3. Beans Alternative

Les beans Alternative sont des beans qui possèdent une implémentation spécifique à un modèle de client particulier ou à un scénario de déploiement particulier.

Exemple 10.13. Définition d'alternative

Cet alternative se rapporte à une implémentation factice d'à la fois un @Synchronous PaymentProcessor et d'un @Asynchronous PaymentProcessor, en un :

```
@Alternative @Synchronous @Asynchronous

public class MockPaymentProcessor implements PaymentProcessor {

   public void process(Payment payment) { ... }

}
```

Par défaut, les beans @Alternative sont désactivés. Ils sont activés pour une archive bean particulière grâce à la modification du fichier **`beans.xml`**.

Rapporter un bogue

10.2.7.4. Remplacer une injection par une alternative

Résumé

Les beans alternatifs vous permettent de remplacer des beans existants. On peut les concevoir comme un moyen d'ajouter une classe qui remplit le même rôle, mais qui fonctionne différemment. Ils sont désactivés par défaut. Cette tâche vous montre comment indiquer et activer une alternative.

Procédure 10.5. Remplacer une injection

Cette tâche part du principe que vous possédez déjà une classe **`TranslatingWelcome`** dans votre projet, mais que vous souhaitez la remplacer par une "fausse" classe TranslatingWelcome. Cela serait le cas pour un déploiement test, où le vrai bean Translator ne peut être utilisé.

1. **Définir l'alternative.**

```
@Alternative
@Translating
public class MockTranslatingWelcome extends Welcome {
  public String buildPhrase(string city) {
    return "Bienvenue Ã  " + city + "!");
  }
}
```

2. **Substituer l'alternative.**
   Pour activer l'implémentation de substitution, ajouter le nom complet de la classe à votre fichier **`META-INF/beans.xml`** ou **`WEB-INF/beans.xml`**.

```
<beans>
  <alternatives>
    <class>com.acme.MockTranslatingWelcome</class>
  </alternatives>
</beans>
```

Résultat

L'implémentation alternative est maintenant utilisée à la place de l'implémentation d'origine.

Rapporter un bogue

10.2.8. Stérétypes

10.2.8.1. Stéréotypes

Dans de nombreux systèmes, l'utilisation de modèles architecturaux produit un ensemble de rôles de beans récurrents. Un stéréotype vous permet d'identifier un tel rôle et de déclarer certaines métadonnées communes pour les beans avec ce rôle dans un emplacement centralisé.

Un stéréotype englobe toute combinaison de :

- scope par défaut
- un groupe de liaisons d'intercepteur

Un stéréotype peut également indiquer un des deux scénarios :

- tous les beans avec un stéréotype ont des noms EL de bean par défaut
- tous les beans avec un stéréotype sont des alternatifs

Un bean peut déclarer zéro, un ou plusieurs stéréotypes. Les annotations de stéréotype peuvent être appliquées à une classe de bean, à une méthode Producer ou à un champ.

Un stéréotype est une annotation, annotée @Stereotype, qui regroupe plusieurs autres annotations.

Une classe qui hérite d'un scope d'un stéréotype peut remplacer ce stéréotype et spécifier un scope directement sur le bean.

De plus, si un stéréotype possède une annotation **`@Named`**, tout bean sur lequel elle se trouve a un nom de bean par défaut. Le bean peut remplacer ce nom si l'annotation @Named est directement spécifiée sur le bean. Pour plus d'informations sur les beans nommés, voir Section 10.2.7.1, « Named Beans ».

Rapporter un bogue

10.2.8.2. Utilisation des stéréotypes

Résumé

Sans les stéréotypes, les annotations peuvent devenir encombrées. Cette tâche vous montre comment utiliser les stéréotypes pour réduire l'encombrement et simplifier votre code. Pour plus d'informations sur ce que sont les stéréotypes, voir Section 10.2.8.1, « Stéréotypes ».

Exemple 10.14. Encombrement d'annotation

```
@Secure
@Transactional
@RequestScoped
@Named
public class AccountManager {
  public boolean transfer(Account a, Account b) {
    ...
  }
}
```

Procédure 10.6. Définition et utilisation des stéréotypes

1. **Définir le stéréotype**

```
@Secure
@Transactional
@RequestScoped
@Named
@Stereotype
@Retention(RUNTIME)
@Target(TYPE)
public @interface BusinessComponent {
 ...
}
```

2. **Utilisation d'un stéréotype.**

```
@BusinessComponent
public class AccountManager {
  public boolean transfer(Account a, Account b) {
    ...
  }
}
```

Résultat :

Les stéréotypes rationalisent et simplifient votre code.

Rapporter un bogue

10.2.9. Méthodes Observer

10.2.9.1. Méthodes Observer

Les *méthodes Observer* reçoivent des notifications quand un événement a lieu.

CDI fournit des *méthodes Observer de transactions*, qui reçoivent des notifications d'événements *avant* ou *après* la phase de complétion de la transaction dans laquelle l'événement a été dirigé.

Rapporter un bogue

10.2.9.2. Appliquer et observer des événements

Exemple 10.15. Appliquer un événement

Ce code vous montre un événement injecté et utilisé dans une méthode.

```
public class AccountManager {
  @Inject Event<Withdrawal> event;

  public boolean transfer(Account a, Account b) {
    ...
    event.fire(new Withdrawal(a));
  }
}
```

Exemple 10.16. Appliquer un événement par un qualificateur

Vous pouvez annoter votre injection d'événement par un qualificateur, pour le rendre plus précis. Pour plus d'informations sur les qualificateurs, voir Section 10.2.3.2, « Qualificateurs ».

```
public class AccountManager {
  @Inject @Suspicious Event <Withdrawal> event;

  public boolean transfer(Account a, Account b) {
    ...
```

```
      event.fire(new Withdrawal(a));
    }
  }
```

Exemple 10.17. Observer un événement

Pour observer un événement, utiliser l'annotation **`@Observes`**.

```
public class AccountObserver {
  void checkTran(@Observes Withdrawal w) {
    ...
  }
}
```

Exemple 10.18. Oberver un événement qualifié

Vous pouvez utiliser des qualificateurs pour observer uniquement certains types d'événements. Pour plus d'informations, voir Section 10.2.3.2, « Qualificateurs »

```
public class AccountObserver {
  void checkTran(@Observes @Suspicious Withdrawal w) {
    ...
  }
}
```

Rapporter un bogue

10.2.10. Intercepteurs

10.2.10.1. Les intercepteurs

Les intercepteurs font partie de la spécification Enterprise JavaBeans specification, qui se trouve http://jcp.org/aboutJava/communityprocess/final/jsr318/. Les intercepteurs vous permettent d'ajouter des fonctionnalités aux méthodes commerciales d'un bean sans modifier la méthode de bean directement. L'intercepteur est exécuté avant toute autre méthode commerciale du bean.

CDI améliore cette fonctionnalité en vous permettant d'utiliser des annotations pour lier les intercepteurs aux beans.

Points d'interception

Interception de méthodes commerciales

Un intercepteur de méthode commeciale s'applique aux invocations de méthodes du bean par les clients du bean.

Interception de lifecycle callback

Un intercepteur de lifecycle callback s'applique aux invocations de lifecycle callback par le conteneur.

Interception de méthode de timeout

Un intercepteur de méthode de timeout s'applique aux invocations de méthodes d'EJB timeout par le conteneur.

Rapporter un bogue

10.2.10.2. Utiliser les intercepteurs avec le CDI

Exemple 10.19. Intercepteurs sans le CDI

Sans le CDI, les intercepteurs ont deux problèmes.

- Le bean doit indiquer l'implémentation de l'intercepteur directement.
- Chaque bean de l'application doit indiquer l'ensemble des intercepteurs dans l'ordre qui convient. Cela fait que l'ajout ou le retrait d'intercepteurs sur la base d'une application prend beaucoup de temps et comporte un risque d'erreur non négligeable.

```
@Interceptors({
  SecurityInterceptor.class,
  TransactionInterceptor.class,
  LoggingInterceptor.class
})
@Stateful public class BusinessComponent {
  ...
}
```

Procédure 10.7. Utiliser les intercepteurs avec le CDI

1. **Définir le type de liaison d'intercepteur.**

```
@InterceptorBinding
@Retention(RUNTIME)
@Target({TYPE, METHOD})
public @interface Secure {}
```

2. **Marquer l'implémentation de l'intercepteur.**

```
@Secure
@Interceptor
public class SecurityInterceptor {
  @AroundInvoke
  public Object aroundInvoke(InvocationContext ctx) throws Exception
{
    // enforce security ...
    return ctx.proceed();
    }
}
```

3. **Utiliser l'intercepteur dans votre code commercial.**

```
@Secure
public class AccountManager {
  public boolean transfer(Account a, Account b) {
    ...
  }
}
```

4. **Activer l'intercepteur dans votre déploiement, en l'ajoutant à votre fichier `META-INF/beans.xml` ou `WEB-INF/beans.xml`.**

```
<beans>
  <interceptors>
    <class>com.acme.SecurityInterceptor</class>
    <class>com.acme.TransactionInterceptor</class>
  </interceptors>
</beans>
```

   Les intercepteurs sont appliqués dans l'ordre indiqué.

Résultat :

Le CDI simplifie votre code d'intercepteur et simplifie l'application de votre code commercial.

Rapporter un bogue

10.2.11. Les décorateurs

Un décorateur intercepte les invocations d'une interface Java particulière. Il est conscient de toutes les sémantiques de cette interface. Les décorateurs sont utiles pour des modèles commerciaux particuliers, mais ne possèdent pas la généralité des intercepteurs. C'est un bean, voire même une classe abstraite, qui implémente le type qu'il décore, et est annoté avec **`@Decorator`**.

Exemple 10.20. Exemple de décorateur

```
@Decorator

public abstract class LargeTransactionDecorator

      implements Account {

   @Inject @Delegate @Any Account account;

   @PersistenceContext EntityManager em;

   public void withdraw(BigDecimal amount) {

      ...

   }

   public void deposit(BigDecimal amount);

      ...
```

```
    }
}
```

Rapporter un bogue

10.2.12. Extensions portables

CDI doit être une fondation de frameworks, d'extensions et d'intégration à d'autres technologies. De ce fait, CDI expose un ensemble de SPI à utiliser par les développeurs d'extensions portables aux CDI. Les extensions fournissent ces types de fonctionnalités :

- intégration par les moteurs de Business Process Management
- intégration aux frameworks de tierce partie comme Spring, Seam, GWT ou Wicket
- nouvelle technologie basée sur le modèle de programmation CDI

D'après la spécification JSR-299, une extension portable peut s'intégrer avec le conteneur d'une des manières suivantes :

- En fournissant ainsi ses propres beans, intercepteurs et décorateurs au conteneur
- En injectant des dépendances dans ses propres objets en utilisant le service d'injections de dépendances
- En fournissant une implémentation de contexte pour un scope personnalisé
- En augmentant ou en remplaçant les métadonnées basées annotation par les métadonnées d'autres sources

Rapporter un bogue

10.2.13. Proxies Beans

10.2.13.1. Proxys Bean

Un proxy est une sous-classe de bean, qui est générée lors de l'exécution. Il est injecté au moment de la création du bean et les beans scoped dépendants peuvent être injectés, parce que les cycles de vie des beans dépendants sont liés au proxy. Les proxys sont utilisées en tant que substitut pour l'injection de dépendance et résolvent deux problèmes différents.

Problèmes d'injection de dépendance, qui sont résolus en utilisant les proxys.

- Performance - les proxys sont bien plus rapides que l'injection de dépendance, donc vous pouvez les utiliser dans des beans qui requièrent une haute performance.
- Thread safety - les proxys envoient des requêtes vers l'instance de bean qui convient, même quand plusieurs threads accèdent à un bean en même temps.

Classes qui ne peuvent pas être mises en proxy

- Types Primitives ou Tableaux

- Classes **`final`** ou qui possèdent une méthode **`final`**
- Classes qui ont un constructeur non-privé par défaut

Rapporter un bogue

10.2.13.2. Utiliser un Proxy ou une Injection

Aperçu

Un proxy est utilisé pour l'injection quand les cycles de vie des beans sont différents les uns des autres. Le proxy est une sous-classe du bean qui est créée au moment de l'exécution et qui remplace toutes les méthodes non privées de la classe de bean. Le serveur proxy transmet l'invocation sur l'instance réelle du bean.

Dans cet exemple, l'instance **`PaymentProcessor`** n'est pas injectée directement dans **Shop**. À la place, un proxy est injecté, et quand la méthode **`processPayment()`** est appelée, le proxy cherche l'instance de bean courante **`PaymentProcessor`** et appelle la méthode **`processPayment()`** dessus.

Exemple 10.21. Injection de proxy

```
@ConversationScoped
class PaymentProcessor
{
  public void processPayment(int amount)
  {
    System.out.println("I'm taking $" + amount);
  }
}

@ApplicationScoped
public class Shop
{

  @Inject
  PaymentProcessor paymentProcessor;

  public void buyStuff()
  {
    paymentProcessor.processPayment(100);
  }
}
```

Pour plus d'informations sur les proxys, y compris les types de classes qui peuvent être mises en proxy, voir Section 10.2.13.1, « Proxys Bean ».

Rapporter un bogue

CHAPITRE 11. JAVA TRANSACTION API (JTA)

11.1. APERÇU

11.1.1. Java Transactions API (JTA)

Introduction

Ces topics vous donnent des explications de base sur l'API Java Transactions (JTA).

- Section 11.2.5, « JTA (Java Transactions API) »
- Section 11.5.2, « Cycle de vie d'une transaction JTA »
- Section 11.9.3, « Exemple de transaction JTA »

Rapporter un bogue

11.2. CONCEPTS DE TRANSACTIONS

11.2.1. Transactions

Une transaction se compose de deux ou plusieurs actions qui doivent toutes réussir ou échouer. Le succès est une validation et un échec est un roll-back. Dans un roll-back, l'état de chaque membre retourne à son état d'origine, avant que la transaction ait été commise.

Le standard d'une transaction bien conçue est qu'elle doit être *Atomique, Consistante, Isolée, et Durable (ACID)*.

Rapporter un bogue

11.2.2. Les propriétés ACID de transactions

ACID est un acronyme pour **`Atomicity`**, **`Consistency`**, **`Isolation`**, et **`Durability`**. Cette terminologie est normalement utilisée dans le contexte de bases de données ou d'opérations transactionnelles.

Définitions ACID

Atomicité

Pour qu'une transaction soit atomique, tous les membres de la transaction doivent prendre la même décision. Ils doivent soit tous valider, soit tous se retirer. Si l'atomicité est cassée, ce qui en résultera s'appelle un *heuristic outcome.*.

Homogénéité

Homogénéité signifie que les données écrites dans la base de données doivent être valides, en terme de schéma de base de données. La base de donnée et les autres sources de données doivent toujours être dans un état consistant. Par exemple, un état inconsistant pourrait correspondre à un champ dans lequel la moitié des données sont écrites avant qu'une opération n'échoue. Dans le cas d'un état consistant, toutes les données sont écrites, et l'écriture est retirée si incomplète.

Isolation

Isolation signifie que les données ajoutées à l'opération doivent être verrouillées avant toute modification, pour ne pas que les processus modifient les données au delà de la portée de la transaction en question.

Durabilité

Durabilité signifie qu'en cas d'échec externe, après que les membres d'une transaction aient été instruits de valider, tous les membres seront disponibles pour continuer la validation de la transaction quand l'échec sera résolu. Cette défaillance est sans doute liée au matériel, logiciels, réseau, ou tout autre système impliqué.

Rapporter un bogue

11.2.3. Coordinateur de transactions ou Gestionnaire de transactions

Les termes *Coordinateur de transactions* et *Gestionnaire de transactions* sont généralement interchangeables en terme de transaction dans JBoss EAP 6. Le terme Coordinateur de transactions est généralement utilisé dans un contexte de transactions distribuées.

Dans les transactions JTA, le *Gestionnaire de transactions* s'exécute dans JBoss EAP 6 et communique avec les participants de la transaction pendant le protocole *two-phase commit*.

Le gestionnaire de transactions ordonne aux participants de commettre ou de retirer leurs données, selon le résultat des autres participants. De cette façon, il s'assure que les transactions respectent la norme ACID.

Pour les transactions JTS, le Coordinateur de transactions gère les interactions entre les gestionnaires de transactions sur des serveurs différents.

- Section 11.2.4, « Participants à une transaction »
- Section 11.2.2, « Les propriétés ACID de transactions »
- Section 11.2.9, « Le protocole de validation en 2-Phases »

Rapporter un bogue

11.2.4. Participants à une transaction

Un participant est un processus qui se trouve dans une transaction, ayant la possibilité de valider ou de retirer l'état. Il peut s'agir d'une base de données ou d'une toute autre application. Chaque participant à une transaction décide par lui-même de valider, ou de revenir en arrière sur son état, et c'est seulement si tous les autres participants peuvent valider que la transaction dans son ensemble réussit. Sinon, chacun d'entre eux sera retiré et la transaction n'aura pas lieu. Le gestionnaire de transactions coordonne les opérations de validation ou de renvoi en arrière et il détermine le résultat de la transaction.

- Section 11.2.1, « Transactions »
- Section 11.2.3, « Coordinateur de transactions ou Gestionnaire de transactions »

Rapporter un bogue

11.2.5. JTA (Java Transactions API)

Java Transactions API (JTA) est une spécification pour les transactions à utiliser dans les applications JBoss Enterprise Edition. Définie dans JSR-907.

Les transactions JTA ne sont pas distribuées à travers les serveurs d'applications multiples, et ne peuvent pas être imbriquées.

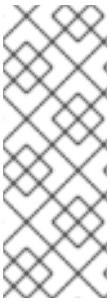
Les transactions JTA sont contrôlées par le conteneur EJB. Les annotations représentent une méthode pour créer et contrôler des transactions dans votre code.

Rapporter un bogue

11.2.6. JTS (Java Transaction Service)

Java Transaction Service (JTS) est un mécanisme servant à soutenir les transactions Java Transaction API (JTA) quand les participants aux transactions se trouvent dans de multiples conteneurs Java Enterprise Edition (serveurs d'application). Tout comme dans les transactions JTA locales, chaque conteneur exécute un procédé appelé *Transaction Manager (TM)*. Les TM communiquent entre eux grâce à un procédé appelé *Object Request Broker (ORB)*, et une communication standard appelée *Common Object Request Broker Architecture (CORBA)*.

Du point de vue d'une application, une transaction JTS fonctionne de la même façon qu'une transaction JTA. La différence est que les participants à la transaction et les sources de données se trouvent dans des conteneurs différents.



NOTE

L'implémentation de JTS compris dans JBoss EAP 6 prend en charge les *distributed JTA transactions*. La différence entre les transactions JTA distribuées et les transactions JTS entièrement compatibles est l'interopérabilité avec les ORB de tierce-parties externes. Cette fonctionnalité n'est pas prise en charge dans JBoss EAP 6. Les configurations prises en charge distribuent des transactions à travers des conteneurs JBoss EAP 6 multiples uniquement.

- Section 11.2.11, « Les transactions distribuées »
- Section 11.2.3, « Coordinateur de transactions ou Gestionnaire de transactions »

Rapporter un bogue

11.2.7. Sources de données XA et transactions XA

Une source de données XA correspond à une source de données qui peut participer à une transaction globale XA.

Une transaction XA est une transaction qui peut s'étendre sur plusieurs ressources. Comprend un gestionnaire de coordination des transactions, avec une ou plusieurs bases de données ou d'autres ressources transactionnelles, toutes impliquées dans une transaction globale unique.

Rapporter un bogue

11.2.8. Recouvrement XA

L'API JTA (Java Transaction API) autorise les transactions distribuées sur de multiples *X/Open XA resources*. XA signifie *Extended Architecture* développée par le groupe X/Ouvert pour définir une transaction qui utilise plus d'une banque de données principale. La norme XA décrit l'interface entre un

Transaction Manager (TM) global et un manager de ressources local. XA autorise de multiples ressources, telles que les serveurs d'application, les bases de données, les caches et les queues de message, à participer à la même transaction, tout en préservant l'atomicité de la transaction. Atomicité signifie que si l'un des participants ne parvient pas à appliquer ses changements, les autres participants metteront fin à la transaction, et restaureront leur état au même statut qu'ils possédaient avant que la transaction n'ait été effectuée.

XA Recovery est le procédé visant à garantir que toutes les ressources affectées par une transaction soient mises à jour ou annulées, même si certaines de ces ressources sont un crash de participants à la transaction ou deviennent indisponibles. Dans le scope de JBoss EAP 6, le sous-système de transaction fournit les mécanismes pour une récupération XA à toute ressource XA ou à tout sous-systèmes qui les utilise, comme les sources de données XA, les queues de messages JMS et les adaptateurs de ressources JCA.

Le recouvrement XA se produit sans l'intervention de l'utilisateur. Dans le cas d'un échec de recouvrement XA, les erreurs sont enregistrées dans la sortie du journal. Veuillez contacter le Service d'assistance globale de Red Hat pour toute demande d'assistance.

Rapporter un bogue

11.2.9. Le protocole de validation en 2-Phases

Le protocole de validation en 2 phases (2PC) correspond à un algorithme qui détermine le résultat d'une transaction.

Phase 1

Au cours de la première phase, les participants à la transaction indiquent au coordinateur de transaction s'ils sont en mesure de valider la transaction ou bien, s'ils doivent la retirer.

Phase 2

Dans la deuxième phase, le coordinateur de transactions prend la décision de savoir si l'opération globale doit être validée ou bien, retirée. Si l'un des participants ne peut pas valider, la transaction sera retirée. Sinon, l'opération peut être validée. Le coordinateur dirige les opérations et en notifie le coordinateur quand les décisions sont prises. À ce moment-là, la transaction est terminée.

Rapporter un bogue

11.2.10. Les délais d'attente des transactions

Afin de préserver l'atomicité et de respecter la norme ACID des transactions, certaines parties d'une transaction peuvent être longues. Les participants à la transaction doivent verrouiller des pièces de sources de données lorsqu'ils commettent, et le gestionnaire de transactions doit attendre d'avoir des nouvelles de chaque participant à une transaction avant de pouvoir tous les diriger à commettre ou à annuler. Les ressources peuvent être verrouillés indéfiniment pour cause de panne de matériel ou de réseau.

Les délais d'attente de transactions peuvent être associés à des transactions afin de contrôler leur cycle de vie. Si un seuil de délai d'attente passe avant que la transaction s'engage ou soit annulée, le délai d'attente provoque l'annulation de l'opération automatiquement.

Vous pouvez configurer les valeurs de délai d'attente par défaut pour tout le sous-système de la transaction, ou bien, vous pouvez désactiver les valeurs de délai d'attente par défaut et spécifier les délais d'attente sur une base d'une transaction.

Rapporter un bogue

11.2.11. Les transactions distribuées

Une *transaction distribuée*, ou *transaction distribuée Java Transaction API (JTA)* est une transaction avec des participants sur de multiples serveurs JBoss EAP 6. Les transactions distribuées diffèrent des *transactions Java Transaction Service (JTS)* du fait que les spécifications JTS requièrent que les transactions puissent être distribuées à travers les serveurs d'application de différents fournisseurs. Le JBoss EAP prend en charge les transactions JTA distribuées.

Rapporter un bogue

11.2.12. API de portabilité ORB

L'Object Request Broker (ORB) est un processus qui envoie et reçoit des messages aux participants de la transaction, aux coordinateurs, aux ressources et aux autres services distribués sur plusieurs serveurs d'applications. Un ORB utilise un IDL (Interface Description Language) pour communiquer et interpréter des messages. *Common Object Request Broker Architecture (CORBA)* est l'IDL utilisé par l'ORB dans JBoss EAP 6.

Le principal type de service qui utilise un ORB est un système de Transactions Java distribuées qui utilise le protocole Java Transaction Service (JTS). Les autres systèmes, notamment les systèmes hérités, peuvent choisir d'utiliser un ORB pour la communication, plutôt que d'autres mécanismes tels que Enterprise JavaBeans à distance ou les services JAX-WS ou JAX-RS Web.

L'API de portabilité ORB fournit des mécanismes pour interagir avec un ORB. Cet API fournit des méthodes pour obtenir une référence à l'ORB, ainsi que pour placer une application dans un mode d'écoute des connexions entrantes de l'ORB. Certaines des méthodes de l'API ne sont pas supportées par tous les ORB. Dans ces cas, une exception sera levée.

Un API comprend plusieurs classes :

Classes API de la portabilité ORB

- `com.arjuna.orbportability.orb`
- `com.arjuna.orbportability.oa`

Voir les documents JBoss EAP 6 Javadocs du portail Red Hat Customer pour obtenir des détails sur les méthodes et les propriétés incluses dans l'API de portabilité ORB.

Rapporter un bogue

11.2.13. Transactions imbriquées

Les transactions imbriquées sont des transactions dans lesquelles les participants sont également des transactions.

Les avantages des transactions imbriquées

Isolation des erreurs

Si une sous-transaction s'annule, par exemple parce qu'un objet utilisé échoue, la transaction englobante ne nécessite pas d'être annulée.

Modularité

Si une transaction est déjà associée à un appel quand une nouvelle transaction commence, la

nouvelle transaction sera imbriquée à l'intérieur. Dès lors, si vous savez qu'un objet requiert des transactions, vous pouvez les créer dans l'objet. Si les méthodes de l'objet sont invoquées sans une transaction client, alors les transactions de l'objet sont de niveau supérieur. Sinon, elles sont imbriquées dans l'étendue des transactions du client. De la même manière, un client n'a pas besoin de savoir si un objet est transactionnel. Il peut commencer sa propre transaction.

Les transactions imbriquées sont seulement prises en charge dans le cadre de l'API JTS (Java Transaction Service) et non pas dans le cadre du JTA (Java Transaction API). Le fait de tenter d'imbriquer des transactions JTA (non distribuées) entraîne une exception.

Rapporter un bogue

11.3. OPTIMISATIONS DES TRANSACTIONS

11.3.1. Optimisations de transactions

Introduction

Le sous-système de transactions de JBoss EAP 6 inclut plusieurs optimisations dont vous pouvez tirer avantage dans vos applications.

- Section 11.3.2, « Optimisation LRCO pour une validation en 1 phase (1 PC) »
- Section 11.3.3, « Optimisation Presumed-Abort »
- Section 11.3.4, « Optimisation Lecture-seule »

Rapporter un bogue

11.3.2. Optimisation LRCO pour une validation en 1 phase (1 PC)

Bien que le protocole de validation en deux phases (2PC) soit plus commun, certaines situations ne peuvent pas accommoder les deux phases. Dans ces cas, vous pouvez utiliser le protocole *single phase commit (1PC)*. Cela a lieu par exemple quand une source de données non-XA-aware doit participer à la transaction.

Dans ces situations, une optimisation appelée *Last Resource Commit Optimization (LRCO)* est employée. La ressource monophasée est traitée en dernier dans la phase de préparation de la transaction, et il y a une tentative de validation. Si la validation réussit, le journal des transactions est écrit et les ressources restantes passent par le protocole 2PC. Si la dernière ressource ne parvient pas à être validée, la transaction est annulée.

Malgré que ce protocole permet le traitement de la plupart des opérations, certains types d'erreurs peuvent aboutir à des résultats de transactions inconsistants. Il est donc recommandé d'utiliser cette approche en dernier ressort.

Quand on utilise une source de données locale TX dans une transaction, l'optimisation LRCO est automatiquement appliquée.

- Section 11.2.9, « Le protocole de validation en 2-Phases »

Rapporter un bogue

11.3.2.1. Commit Markable Resource (CRM)

Résumé

La configuration de l'accès à un gestionnaire de ressources via l'interface CRM de validation de ressources marquables (Commit Markable Resource) garantit que le gestionnaire de ressources 1PC puisse être inscrit dans une transaction 2PC. C'est une implémentation de l'algorithme LRCO, qui rend les ressources non-XA entièrement récupérables.

Jusqu'à maintenant, l'ajout de ressources 1PC à une transaction 2PC était réalisée par la méthode LRCO, mais il y a place à erreur avec LRCO. Suivre la procédure ci-dessous pour ajouter des ressources 1PC à une transaction 2PC via la méthode LRCO :

1. Préparer 2PC
2. Valider LRCO
3. Write tx log
4. Commit 2PC

Si la procédure échoue entre les étapes 2 et 3, vous ne pourrez pas valider le 2PC. CMR élimine cette restriction et permet à 1PC d'être bien inscrit dans une transaction 2PC.

Restrictions

Une transaction peut ne contenir qu'une ressource CMR.

Conditions préalables

Vous devez avoir un tableau créé pour lequel le SQL suivant fonctionne :

```
SELECT xid,actionuid FROM _tableName_ WHERE transactionManagerID IN
(String[])
DELETE FROM _tableName_ WHERE xid IN (byte[[]])
INSERT INTO _tableName_ (xid, transactionManagerID, actionuid) VALUES
(byte[],String,byte[])
```

Exemple 11.1. Voici quelques exemples de requêtes SQL

Sybase :

```
CREATE TABLE xids (xid varbinary(144), transactionManagerID varchar(64),
actionuid varbinary(28))
```

Oracle :

```
CREATE TABLE xids (xid RAW(144), transactionManagerID varchar(64),
actionuid RAW(28))
CREATE UNIQUE INDEX index_xid ON xids (xid)
```

IBM :

```
CREATE TABLE xids (xid VARCHAR(255), transactionManagerID varchar(64),
actionuid VARCHAR(255))
CREATE UNIQUE INDEX index_xid ON xids (xid)
```

SQL Server :

```
CREATE TABLE xids (xid varbinary(144), transactionManagerID varchar(64),
actionuid varbinary(28))
CREATE UNIQUE INDEX index_xid ON xids (xid)
```

Postgres :

```
CREATE TABLE xids (xid bytea, transactionManagerID varchar(64),
actionuid bytea)
CREATE UNIQUE INDEX index_xid ON xids (xid)
```

Activer un gestionnaire de ressources en tant que CMR

Par défaut, la fonctionnalité CMR est désactivée pour les sources de données. Pour l'activer, vous devez créer ou modifier la configuration de source de données et vous assurer que l'attribut connectable est défini sur true. Voici un exemple d'entrée de configuration de la section sources de données d'un fichier de configuration xml serveur :

```
<datasource enabled="true" jndi-
name="java:jboss/datasources/ConnectableDS" pool-name="ConnectableDS"
jta="true" use-java-context="true" spy="false" use-ccm="true"
connectable="true"/>
```

NOTE

Cette fonctionnalité ne s'applique pas aux sources de données XA.

Vous pouvez également activer un gestionnaire de ressources en tant que CMR par le CLI de la façon suivante :

```
/subsystem=datasources/data-source=ConnectableDS:add(enabled="true", jndi-
name="java:jboss/datasources/ConnectableDS", jta="true", use-java-
context="true", spy="false", use-ccm="true", connectable="true",
connection-url="validConnectionURL", driver-name="h2")
```

Mettre à jour une ressource existante pour qu'elle puisse utiliser la nouvelle fonctionnalité CMR

Si vous ne devez mettre à jour qu'une seule ressource existante pour utiliser la nouvelle fonctionnalité CMR, il vous suffira de modifier l'attribut connectable :

```
/subsystem=datasources/data-source=ConnectableDS:write-
attribute(name=connectable,value=true)
```

Identification des sources de données pouvant utiliser la fonctionnalité CMR

Le sous-système de transactions identifie les sources de données qui peuvent utiliser la fonctionnalité CMR par l'intermédiaire d'une entrée dans la section de config du sous-système de transaction comme dans l'exemple ci-dessous :

```
<subsystem xmlns="urn:jboss:domain:transactions:3.0">
    ...
    <commit-markable-resources>
        <commit-markable-resource jndi-
name="java:jboss/datasources/ConnectableDS">
```

```
            <xid-location name="xids" batch-size="100" immediate-
cleanup="false"/>
        </commit-markable-resource>
        ...
    </commit-markable-resources>
</subsystem>
```

NOTE

Vous devez démarrer le serveur à nouveau pour ajouter le CMR.

Rapporter un bogue

11.3.3. Optimisation Presumed-Abort

Quand une transaction est sur le point d'être supprimée, elle peut enregistrer ces informations localement et informer tous les participants inscrits. Cette notification est une courtoisie seulement et n'a aucun effet sur le résultat de la transaction. Après que tous les participants aient été contactés, les informations sur la transaction peuvent être supprimés.

Si une demande subséquente au sujet de l'état de la transaction se produit, il y n'aura aucun information disponible. Dans ce cas, le demandeur suppose que la transaction a avorté et a été restaurée. Cette optimisation *presumed-abort* indique qu'aucune information sur les participants ne doit être rendue persistante tant que la transaction n'a pas été engagée, car tout échec avant ce point sera considéré comme un abandon de la transaction.

Rapporter un bogue

11.3.4. Optimisation Lecture-seule

Lorsqu'un participant est invité à se préparer, il peut indiquer au coordonnateur qu'il n'a pas modifié de données durant la transaction. Une tel participant n'a pas besoin d'être informé de l'issue de l'opération, puisque le sort du participant n'a aucun effet sur la transaction. Ce participant *read-only* peut être omis de la deuxième phase du protocole de validation.

Rapporter un bogue

11.4. RÉSULTATS DE TRANSACTIONS

11.4.1. Résultats de transactions

Il y a trois résultats possibles suite à une transaction.

Roll-back

Si un participant à une transaction ne peut pas effectuer une validation, ou si le coordinateur de transactions ne peut pas indiquer aux participants de valider, la transaction sera annulée. Voir Section 11.4.3, « Transactions Roll-Back » pour plus d'informations.

Valider

Si chaque participant à une transaction peut valider, le coordinateur de transactions leur indique comment procéder, Voir Section 11.4.2, « Transactions de validation » pour plus d'informations.

Résultat heuristique

Si certains participants à une transaction valident, et que d'autres annulent, on parle de résultat heuristique. Les résultats heuristiques dérivent d'interventions humaines. Voir Section 11.4.4, « Résultats heuristiques » pour plus d'informations.

Rapporter un bogue

11.4.2. Transactions de validation

Lorsqu'un participant à une transaction fait une validation, il rend son nouvel état durable. Le nouvel état est créé par le participant qui effectue le travail dans la transaction. L'exemple le plus courant est lorsqu'un membre de la transaction écrit des enregistrements à une base de données.

Suite à la validation, les informations sur la transaction sont retirées du coordinateur de transactions, et l'état nouvellement inscrit est maintenant l'état durable.

Rapporter un bogue

11.4.3. Transactions Roll-Back

Un participant à une transaction est considéré «roll-back» quand son état est restauré pour refléter son état qui précède le début de la transaction. Après un roll-back, l'état est le même que si la transaction n'avait jamais eu lieu.

Rapporter un bogue

11.4.4. Résultats heuristiques

Un résultat heuristique, ou résultat non atomique, est une anomalie de la transaction. Il se réfère à une situation où certains participants à une transaction ont validé leur état, et d'autres l'ont annulé. Un résultat heuristique entraîne une incompatibilité entre les états.

Les résultats heuristiques se produisent généralement au cours de la deuxième phase du protocole de validation à 2 phases (2PC). Ils sont souvent causées par des échecs au niveau du matériel physique sous-jacent ou des sous-systèmes de communication des serveurs sous-jacents.

Il y a quatre types de résultats heuristiques différents.

Rollback heuristique

L'opération de validation a échoué car certains ou tous les participants ont abandonné la transaction.

Validation heuristique

Une opération de tentative de restauration a échoué, car tous les participants ont validé unilatéralement. Cela peut se produire si, par exemple, le coordinateur est en mesure de préparer correctement la transaction, mais décide alors de l'annuler en raison d'une défaillance de son côté, comme un oubli de mettre à jour son journal. Dans l'intervalle, les participants peuvent décider de valider.

Mixed heuristiques

Certains participants ont validé et d'autres ont abandonné

Danger heuristique

Le résultat de certaines mises à jour est inconnu. Pour les connues, ils ont soit tous été validés ou été annulés.

Les résultats heuristiques peuvent entraîner des pertes d'intégrité dans le système, et nécessitent généralement une intervention humaine pour être résolus. Ne créez pas de code qui en dépende.

- Section 11.2.9, « Le protocole de validation en 2-Phases »

Rapporter un bogue

11.4.5. Erreurs et exceptions pour les transactions JBoss

Pour obtenir des informations lancées par les méthodes de la classe **`UserTransaction`**, voir la spécification *UserTransaction API* dans http://download.oracle.com/javaee/1.3/api/javax/transaction/UserTransaction.html.

Rapporter un bogue

11.5. APERÇU SUR LES TRANSACTIONS JTA

11.5.1. JTA (Java Transactions API)

Java Transactions API (JTA) est une spécification pour les transactions à utiliser dans les applications JBoss Enterprise Edition. Définie dans JSR-907.

Les transactions JTA ne sont pas distribuées à travers les serveurs d'applications multiples, et ne peuvent pas être imbriquées.

Les transactions JTA sont contrôlées par le conteneur EJB. Les annotations représentent une méthode pour créer et contrôler des transactions dans votre code.

Rapporter un bogue

11.5.2. Cycle de vie d'une transaction JTA

Quand une ressource demande à participer à une transaction, une chaîne d'événements est démarrée. Le *Transaction Manager* ou Gestionnaire de transactions est un processus qui réside dans le serveur de l'application et qui gère les transactions. Les *participants à la transaction* sont des objets qui participent à une transaction. Les *Ressources* sont des sources de données, des usines de connexion JMS, ou autres connexions JCA.

1. **Votre application démarre une nouvelle transaction**
   Pour démarrer une transaction, votre application doit obtenir une instance de la classe **`UserTransaction`** du JNDI ou bien, s'il s'agit d'un EJB, en provenance d'une annotation. L'interface **`UserTransaction`** comprend des méthodes pour commencer, valider, ou annuler des transactions au plus haut niveau. Les transactions nouvellement créées sont associées automatiquement à leur thread invoquant. Les transactions imbriquées ne sont pas supportées dans JTA, donc toutes les transactions sont des transactions au plus haut niveau.

   En appelant **`UserTransaction.begin()`**, on commence une nouvelle transaction. Toute ressource qui est utilisée après cela sera associée à la transaction. S'il y a plus plus d'une seule ressource listée, votre transaction devient une transaction XA, et participera au protocole de validation en deux temps.

2. **Votre application modifie son état.**
   Dans l'étape suivante, votre transaction effectue son travail et précède aux changements d'état.

3. **Votre application décide de valider ou de s'annuler**
   Quand votre application a fini de changer d'état, elle décide si elle doit valider ou s'annuler. Elle appelle la méthode qui convient. Elle appelle **`UserTransaction`**. **`commit()`** ou **`UserTransaction`**.**`rollback()`**. C'est quand le protocole en deux phases (2PC) a lieu aprés que vous ayiez listé plus d'une ressource. Section 11.2.9, « Le protocole de validation en 2-Phases »

4. **Le gestionnaire de transactions supprime la transaction des archives.**
   Après la validation ou l'annulation, le gestionnaire de transactions nettoie ses archives et supprime les informations sur votre transaction.

Recouvrement d'échec

Le recouvrement d'échec a lieu automatiquement. Si une ressource, un participant à une transaction, ou un serveur d'applications est disponible, le gestionnaire de transactions s'occupe du recouvrement quand l'échec sous-jacent est résolu.

- Section 11.2.1, « Transactions »
- Section 11.2.3, « Coordinateur de transactions ou Gestionnaire de transactions »
- Section 11.2.4, « Participants à une transaction »
- Section 11.2.9, « Le protocole de validation en 2-Phases »
- Section 11.2.7, « Sources de données XA et transactions XA »

Rapporter un bogue

11.6. CONFIGURATION DE SOUS-SYSTÈME DE TRANSACTION

11.6.1. Configuration des transactions

Introduction

Les procédures suivantes vous montrent comment configurer le sous-système de transactions de JBoss EAP 6.

- Section 11.6.2.1, « Configurez votre base de données pour pouvoir utiliser les transactions JTA »
- Section 11.6.2.2, « Configuration d'une source de données XA »
- Section 11.7.8.2, « Configurer le Transaction Manager (TM) (ou gestionnaire de transactions) »
- Section 11.6.3.2, « Configurer la journalisation des sous-systèmes de transactions »

Rapporter un bogue

11.6.2. Configuration de source de données transactionnelle

11.6.2.1. Configurez votre base de données pour pouvoir utiliser les transactions JTA

Résumé

Cette tâche vous montre comment activer JTA (Java Transactions API) sur votre source de données.

Conditions préalables

Vous devez remplir les conditions suivantes avant de continuer cette tâche :

- Votre base de données ou une autre ressource devra supporter JTA. Dans le doute, veuillez consulter la documentation.
- Créer une source de données. Veuillez vous référer à Section 11.6.2.4, « Créer une source de données non-XA par les interfaces de gestion ».
- Stopper le serveur JBoss EAP 6
- Obtenir un accès pour pouvoir éditer les fichiers de configuration directement dans un éditeur de texte.

Procédure 11.1. Configurer la source de données pour utiliser les transactions JTA.

1. **Ouvrir le fichier de configuration dans l'éditeur de texte.**
   Selon que vous exécutiez JBoss EAP 6 sur un domaine géré ou un serveur autonome, votre fichier de configuration ne se trouvera pas au même endroit.
   - **Domaine géré**
     Le fichier de configuration par défaut d'un domaine géré se trouve dans ***EAP_HOME*/domain/configuration/domain.xml** pour Red Hat Enterprise Linux, et ***EAP_HOME*\domain\configuration\domain.xml** pour Microsoft Windows Server.
   - **Serveur autonome**
     Le fichier de configuration par défaut d'un serveur autonome se trouve dans ***EAP_HOME*/standalone/configuration/domain.xml** pour Red Hat Enterprise Linux, et ***EAP_HOME*\standalone\configuration\domain.xml** pour Microsoft Windows Server.
2. **Chercher la balise `<datasource>` qui correspond à votre source de données.**
   La source de données aura un attribut **`jndi-name`** correspondant à celui que vous aviez indiqué quand vous l'avez créée. Par exemple, la source de données ExampleDS ressemble à ceci :

   ```
   <datasource jndi-name="java:jboss/datasources/ExampleDS" pool-
   name="H2DS" enabled="true" jta="true" use-java-context="true" use-
   ccm="true">
   ```

3. **Définir l'attribut `jta` à `true`.**
   Ajouter l'élément suivant au contenu de votre balise **`<datasource>`**, tel qu'il apparaît à l'étape précédente : **`jta="true"`**
4. **Sauvegarder le fichier de configuration.**
   Sauvegarder le fichier de configuration et sortir de l'éditeur de texte.
5. **Démarrer JBoss EAP 6.**
   Relancer le serveur JBoss EAP 6.

Résultat :

JBoss EAP 6 démarre, et votre source de données est configurée pour utiliser les transactions JTA.

Rapporter un bogue

11.6.2.2. Configuration d'une source de données XA

Conditions préalables

Pour pouvoir ajouter une source de données XA, vous devrez vous connecter à la console de gestion. Voir Section 11.6.2.3, « Se conncecter à la console de gestion » pour plus d'informations.

1. **Ajouter une nouvelle source de données.**
   Ajouter une nouvelle source de données à la plateforme JBoss EAP 6. Suivre les instructions qui se trouvent dans Section 11.6.2.4, « Créer une source de données non-XA par les interfaces de gestion », puis, cliquer sur l'onglet `XA Datasource` en haut.

2. **Configurer les propriétés supplémentaires suivant les besoins.**
   Tous les paramètres de la source de données se trouvent dans Section 11.6.2.5, « Paramètres de source de données ».

Résultat

Votre source de données XA est configurée et prête à l'utilisation.

Rapporter un bogue

11.6.2.3. Se conncecter à la console de gestion

Dans la page d'accueil de la console de gestion de Red Hat JBoss Enterprise Application Platform, il y a un point d'entrée unique dans l'application pour les serveurs autonomes, ainsi que pour les serveurs exécutant dans un domaine géré. Cette page donne également des liens pour des ressources générales, opérationnelles ou à but de développement.

Conditions préalables

- Vous devez créer un utilisateur administratif. Pour obtenir des instructions complètes, voir *Add the User for the Management Interfaces* qui se trouve dans le guide *Administration and Configuration Guide* de JBoss Enterprise Application Platform.

  JBoss EAP doit être en cours d'exécution.

Procédure 11.2. Se conncecter à la console de gestion

1. **Naviguer vers la page de démarrage de la console de gestion**
   Naviguer dans la console de gestion avec votre navigateur web. L'emplacement par défaut est http://localhost:9990/console/App.html, où le port 9990 est prédéfini comme liaison de socket de la console de gestion.

2. **Se conncecter à la console de gestion**
   Saisir le nom d'utilisateur et le mot de passe du compte que vous avez déjà créés pour vous connecter à l'écran de connexion de la console de gestion.

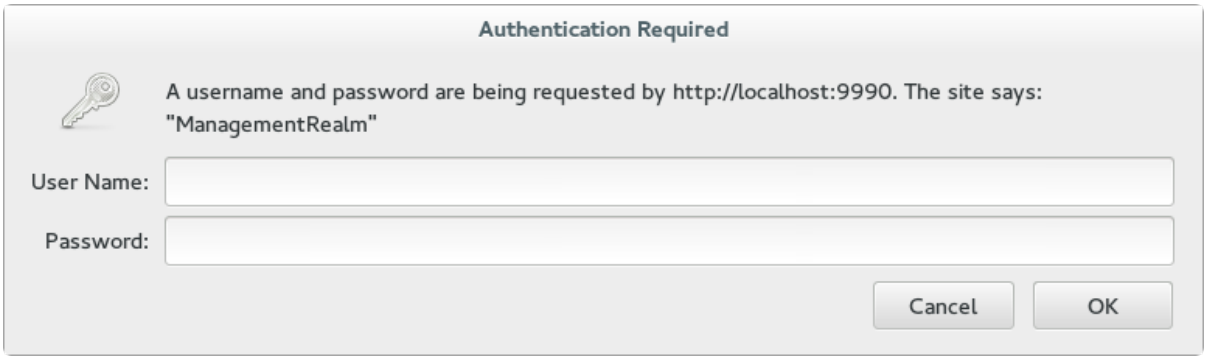


Figure 11.1. Écran de connexion de la console de gestion

Résultat

Une fois connecté, la page d'accueil de la console de gestion apparaîtra : http://localhost:9990/console/App.html#home

Rapporter un bogue

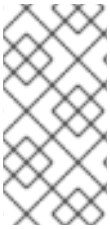
11.6.2.4. Créer une source de données non-XA par les interfaces de gestion

Résumé

Cette section explique les étapes à suivre pour créer une source de données non-XA, en utilisant la console de gestion ou l'interface CLI.

Conditions préalables

- Le serveur JBoss EAP 6 doit être en cours d'exécution.



NOTE

Avant la version 10.2 de la source de données Oracle, le paramètre <no-tx-separate-pools/> était requis, car le mélange de connexions transactionnelles et non-transactionnelles aurait créé une erreur. Ce paramètre n'est plus requis pour certaines applications.

Procédure 11.3. Créer une source de données en utilisant l'interface CLI ou la console de gestion

- **L'interface CLI**
  a. Lancer le CLI et connectez-vous à votre serveur.
  b. Exécuter la commande suivante pour créer une source de données non-XA, et configurer les variables comme il se doit :

```
data-source add --name=DATASOURCE_NAME --jndi-name=JNDI_NAME --driver-name=DRIVER_NAME  --connection-url=CONNECTION_URL
```

  c. Activer la source de données :

```
data-source enable --name=DATASOURCE_NAME
```

- **Console de gestion**

a. Connectez-vous à la console de gestion.

b. **Naviguez dans le panneau `Datasources` qui se trouve dans la console de gestion**

   i. Sélectionner **`Configuration`** qui se trouve en haut de la console.

   ii. En mode de domaine uniquement, sélectionner un profil à partir du menu déroulant qui se trouve en haut et à gauche.

   iii. Étendre le menu **Subsystems** qui se trouve à gauche de la console, puis étendre le menu **Connector**.

   iv. Sélectionner **Datasources** à partir du menu à gauche de la console.

c. **Créer une nouvelle source de données**

   i. Cliquer sur **`Add`** qui se trouve en haut du panneau **`Datasources`**.

   ii. Saisir les attributs de la nouvelle source de données de l'assistant **`Create Datasource`** et appuyer sur **`Next`**.

   iii. Saisir les informations sur le pilote JDBC dans l'assistant **`Create Datasource`** et cliquer sur **`Next`** pour continuer.

   iv. Saisir les paramètres de connexion dans l'assistant **`Create Datasource`**.

   v. Cliquer sur le bouton **`Test Connection`** pour tester la connexion à la ressource de données et vérifier que les paramètres de configuration soient corrects.

   vi. Cliquer sur **`Done`** pour terminer.

Résultat

La source de données non-Xa a été ajoutée au serveur. Elle est maintenant visible dans le fichier **`standalone.xml`** ou le fichier **`domain.xml`**, ainsi que dans les interfaces de gestion.

Rapporter un bogue

11.6.2.5. Paramètres de source de données

Tableau 11.1. Les paramètres de source de données communs aux sources XA ou non-XA

Paramètre	Description
jndi-name	Le nom JNDI unique pour la source de données.
pool-name	Le nom du pool de gestion de la source de données.
enabled	Indique si la source de données est activée.
use-java-context	Indique si on doit relier la source de données au JNDI global.

Paramètre	Description
spy	Activer la fonctionnalité **spy** sur la couche JDBC. Cela journalisera tout le trafic JDBC dans la source de données. Notez que la catégorie de journalisation **jboss.jdbc.spy** doit également être définie au niveau **DEBUG** dans le sous-système de journalisation.
use-ccm	Activer le gestionnaire de connexion cache.
new-connection-sql	Une déclaration SQL qui exécute quand la connexion est ajoutée au pool de connexion.
transaction-isolation	Une déclaration parmi celles-ci : • TRANSACTION_READ_UNCOMMITTED • TRANSACTION_READ_COMMITTED • TRANSACTION_REPEATABLE_READ • TRANSACTION_SERIALIZABLE • TRANSACTION_NONE
url-delimiter	Le délimiteur d'URLs d'une connexion url pour les bases de données clusterisées HA (Haute disponibilité).
url-selector-strategy-class-name	Une classe qui implémente l'interface **org.jboss.jca.adapters.jdbc.URLSelectorStrategy**.
sécurité	Contient des éléments dépendants en tant que paramètres de sécurité. Voir Tableau 11.6, « Paramètres de sécurité ».
validation	Contient des éléments dépendants en tant que paramètres de validation. Voir Tableau 11.7, « Paramètres de validation ».
timeout	Contient des éléments dépendants en tant que paramètres de timeout. Voir Tableau 11.8, « Paramètres de timeout ».
énoncé	Contient des éléments dépendants en tant que paramètres d'énoncés. Voir Tableau 11.9, « Paramètres d'instruction ».

Tableau 11.2. Paramètres de source de données non-xa

Paramètre	Description
jta	Active l'intégration JTA pour les sources de données non-XA. Ne s'applique pas aux sources de données XA.
connection-url	L'URL de connexion du pilote JDBC.
driver-class	Le nom complet de la classe de pilote JDBC.
connection-property	Propriétés de connexion arbitraires passées à la méthode **`Driver.connect(url,props)`**. Chaque connection-property indique une paire name/value. Le nom de la propriété provient du nom, et la valeur provient du contenu de l'élément.
pool	Contient des éléments dépendants en tant que paramètres de pooling. Voir Tableau 11.4, « Les paramètres de pool communs aux sources XA ou non-XA ».

Tableau 11.3. Paramètres de source de données XA

Paramètre	Description
xa-datasource-property	Une propriété pour assigner la classe d'implémentation **`XADataSource`**. Spécifié par *name=value*. Si une méthode setter existe, dans le format **`setName`**, la propriété sera définie en appelant une méthode setter sous le format **`setName(value)`**.
xa-datasource-class	Le nom complet de la classe d'implémentation de **`javax.sql.XADataSource`**.
pilote	Unique référence au module de chargeur de classe qui contient le pilote JDBC. Le format accepté est *driverName#majorVersion.minorVersion*.
xa-pool	Contient des éléments dépendants en tant que paramètres de pooling. Voir Tableau 11.4, « Les paramètres de pool communs aux sources XA ou non-XA » et Tableau 11.5, « Paramètres du pool XA »
recouvrement	Contient des éléments dépendants en tant que paramètres de recouvrement. Voir Tableau 11.10, « Paramètres de recouvrement ».

Tableau 11.4. Les paramètres de pool communs aux sources XA ou non-XA

Paramètre	Description
min-pool-size	Le nombre minimum de connexions contenues par un pool.
max-pool-size	Le nombre maximum de connexions qu'un pool peut contenir
Pré-remplissage	Indique si l'on doit essayer de pré-remplir un pool de connexions. Un élément vide indique une valeur **`true`**. La valeur par défaut est **`false`**.
use-strict-min	Indique si la taille du pool est stricte. **`false`** par défaut.
flush-strategy	Indique si le pool est vidé en cas d'erreur. Les valeurs acceptées sont : • FailingConnectionOnly • IdleConnections • EntirePool La valeur par défaut est **`FailingConnectionOnly`**.
allow-multiple-users	Indique si plusieurs utilisateurs pourront avoir accès à la source de données par la méthode getConnection(user, password), et si les types de pools internes ont une influence sur ce comportement.

Tableau 11.5. Paramètres du pool XA

Paramètre	Description
is-same-rm-override	Indique si la classe **`javax.transaction.xa.XAResource.isSameRM(XAResource)`** retourne **`true`** ou **`false`**.
entrelacement	Indique si on doit activer l'entrelacement pour les fabriques de connexion XA.
no-tx-separate-pools	Indique si on doit créer des sous-répertoires distincts pour chaque contexte. Cela est nécessaire pour les sources de données Oracle, qui ne permettent pas aux connexions XA d'être utilisées à la fois à l'intérieur et à l'extérieur d'une transaction de JTA Utiliser cette option entraînera la multiplication par deux de la taille du pool **`max-pool-size`**, car en fait, deux pools seront créés.

Paramètre	Description
pad-xid	Indique si on doit remplir le Xid.
wrap-xa-resource	Indique si on doit inclure XAResource dans une instance **`org.jboss.tm.XAResourceWrapper`**.

Tableau 11.6. Paramètres de sécurité

Paramètre	Description
user-name	Le nom d'utilisation pour créer une nouvelle connexion.
password	Le mot de passe à utiliser pour créer une nouvelle connexion
security-domain	Contient le nom d'un gestionnaire de sécurité JAAS, qui gère l'authentification. Ce nom correspond à l'attribut application-policy/name de la configuration de connexion JAAS.
reauth-plugin	Définit un plugin d'authentification à nouveau pour la réauthentification de connexions physiques.

Tableau 11.7. Paramètres de validation

Paramètre	Description
valid-connection-checker	Une mise en œuvre d'interface **`org.jboss.jca.adaptors.jdbc.ValidConnectionChecker`** qui fournit une méthode **`SQLException.isValidConnection(Connection e)`** pour valider une connexion. Une exception signifie que la connexion est détruite. Cela remplace le paramètre **`check-valid-connection-sql`** s'il est présent.
check-valid-connection-sql	Déclaration SQL pour vérifier la validité d'un pool de connexion. Peut être appelé quand une connexion gérée est tirée d'un pool.

Paramètre	Description
validate-on-match	Indique si la validation des niveaux connexion est exécutée lorsqu'une fabrique de connexions essaie de correspondre à une connexion gérée pour un ensemble donné. Indiquer "true" pour **`validate-on-match`** n'est pas normalement fait en conjonction avec "true" pour **`background-validation`**. **`Validate-on-match`** est utile quand un client doit avoir une connexion vallidée avant utilisation. Ce paramètre est à false par défaut.
background-validation	Indique que les connexions sont validées sur un thread d'arrière-plan. La validation de l'arrière-plan (background validation) est une optimisation de performances lorsque non utilisé avec **`validate-on-match`**. Si **`Validate-on-match`** est sur true, l'utilisation de **`background-validation`** pourrait entraîner des contrôles redondants. La validation de l'arrière-plan pourrait provoquer une mauvaise connexion (une connexion qui irait mal entre le moment de l'analyse de validation et avant d'être donnée au client), l'application cliente doit par conséquent tenir compte de cette possibilité.
background-validation-millis	La durée, en millisecondes, pendant laquelle la validation d'arrière-plan exécute.
use-fast-fail	Si défini sur true, échoue une allocation de connexion lors de la première tentative, si la connexion est non valide. La valeur par défaut est **`false`**.
stale-connection-checker	Une instance de **`org.jboss.jca.adapters.jdbc.StaleConnectionChecker`** qui produit une méthode booléenne **`isStaleConnection(SQLException e)`**. Si cette méthode renvoie un **`true`**, l'exception sera contenue dans **`org.jboss.jca.adapters.jdbc.StaleConnectionException`**, qui correspond à une sous-classe de **`SQLException`**.
exception-sorter	Une instance de **`org.jboss.jca.adapters.jdbc.ExceptionSorter`** qui fournit une méthode booléenne **`isExceptionFatal(SQLException e)`**. Cette méthode valide le fait qu'une exception soit envoyée à toutes les instances d'un **`javax.resource.spi.ConnectionEventListener`** en tant que message **`connectionErrorOccurred`**.

Tableau 11.8. Paramètres de timeout

Paramètre	Description
use-try-lock	Utiliser **`tryLock()`** au lieu de **`lock()`**. Vous essayerez ainsi d'obtenir un verrou pour le nombre de secondes configurées, avant le timeout, au lieu d'échouer immédiatement quand le verrou n'est pas disponible. La valeur par défaut est de **`60`** secondes. Par exemple, pour définir un timeout de 5 minutes, définir **`<use-try-lock>`**300**`</use-try-lock>`**.
blocking-timeout-millis	La durée maximale, en millisecondes, de blocage lorsque vous attendez une connexion. Après ce délai, une exception sera levée. Cela bloque uniquement pendant que vous attendez un permis de connexion et ne lève pas d'exception si la création d'une nouvelle connexion prend beaucoup de temps. Par défaut, à 30000, ce qui correspond à 30 secondes.
idle-timeout-minutes	La durée maximale, en minutes, avant qu'une connexion inactive soit fermée. La durée maximale réelle dépend de la durée d'analyse de l'idleRemover, qui correspond à la moitié du plus petit **`idle-timeout-minutes`** de n'importe quel pool.
set-tx-query-timeout	Indique si on doit définir le timeout d'interrogation par rapport au temps qui reste avant le timeout de transaction. Si aucune transaction n'existe, on utilisera le timeout de recherche configuré. La valeur par défaut est **`false`**.
query-timeout	Timeout pour les recherches, en secondes. La valeur par défaut est « no timeout ».
allocation-retry	Le nombre de tentatives de connexions avant d'envoyer une connexion. La valeur par défaut est **`0`**, pour qu'une exception puisse être envoyée à la première défaillance.
allocation-retry-wait-millis	Le temps, en millisecondes, qu'il faut attendre avant de retenter d'allouer une connexion. La valeur par défaut est 5000, soit 5 secondes.
xa-resource-timeout	Si la valeur est non nulle, elle passe à la méthode **`XAResource.setTransactionTimeout`**.

Tableau 11.9. Paramètres d'instruction

Paramètre	Description

Paramètre	Description
track-statements	Indique si l'on doit vérifier les déclarations non fermés lorsqu'une connexion est renvoyée à un pool ou qu'une déclaration est renvoyée dans le cache de la déclaration préparée. Si défini à false, les déclarations ne seront pas suivies. **Valeurs valides** • **`true`** : les déclarations et les ensembles de résultats sont suivis, et un avertissement sera émis s'ils ne sont pas fermés. • **`false`** : ni les déclarations, ni les ensembles de résultats ne seront suivis. • **`nowarn`** : les déclarations seront suivies, mais il n'y a aucun avertissement. Valeur par défaut.
prepared-statement-cache-size	Le nombre de déclarations préparése par connexion, dans le cache LRU (Least Recently Used / Utilisé le moins souvent récemment).
share-prepared-statements	Indique si le fait de demander une même déclaration deux fois sans la fermer utilise la même déclaration préparé sous-jacente. La valeur par défaut est **`false`**.

Tableau 11.10. Paramètres de recouvrement

Paramètre	Description
recover-credential	Une paire nom d'utilisateur/mot de passe ou domaine de sécurité pour le recouvrement.
recover-plugin	Une mise en œuvre de la classe **`org.jboss.jca.core.spi.recoveryRecoveryPlugin`** à utiliser pour le recouvrement.

Rapporter un bogue

11.6.3. Journalisation des transactions

11.6.3.1. Messages de journalisation de transactions

Pour suivre le statut de la transaction tout en gardant les fichiers de journalisation lisibles, utiliser le niveau de journalisation **DEBUG** pour le logger de transaction. Pour un débogage détaillé, utiliser le niveau de journalisation **TRACE**. Veuillez consulter Section 11.6.3.2, « Configurer la journalisation des sous-systèmes de transactions » pour plus d'informations sur la configuration du logger de transaction.

Le gestionnaire de transaction peut générer beaucoup d'informations de journalisation si configuré pour se connecter au niveau de journalisation **TRACE**. Vous trouverez ci-dessous quelques-uns des messages les plus courants. Cette liste n'est pas exhaustive, il se peut que vous rencontriez d'autres messages.

Tableau 11.11. Changement d'état de transaction

Début de transaction	Quand une transaction commence, le code suivant s'exécute :
	`com.arjuna.ats.arjuna.coordinator.BasicAction::Begin:1342`
	`tsLogger.logger.trace("BasicAction::Begin() for action-id "+ get_uid());`

Validation de transaction	Quand une transaction est validée, le code suivant s'exécute :
	`com.arjuna.ats.arjuna.coordinator.BasicAction::End:1342`
	`tsLogger.logger.trace("BasicAction::End() for action-id "+ get_uid());`

Restauration de transaction	Quand une transaction est restaurée, le code suivant s'exécute :
	`com.arjuna.ats.arjuna.coordinator.BasicAction::Abort:1575`
	`tsLogger.logger.trace("BasicAction::Abort() for action-id "+ get_uid());`

Délai d'expiration de transaction	Quand une transaction expire, le code suivant s'exécute :
	`com.arjuna.ats.arjuna.coordinator.TransactionReaper::doCancellations:349`
	`tsLogger.logger.trace("Reaper Worker " + Thread.currentThread() + " attempting to cancel " + e._control.get_uid());`
	Vous verrez ensuite le même thread restaurer la transaction comme montré ci-dessus.

Rapporter un bogue

11.6.3.2. Configurer la journalisation des sous-systèmes de transactions

Résumé

Utiliser cette procédure pour contrôler la quantité d'informations enregistrées sur les transactions, indépendamment des autres paramètres de journalisation dans JBoss EAP 6. La procédure montre comment procéder dans la console de gestion sur le web. La commande de gestion CLI est donnée par la suite.

Procédure 11.4. Configurer l'enregistreur de transactions (Transaction Logger) par la console de gestion

1. **Naviguer vers la zone de configuration de la journalisation**
   Dans la console de gestion, cliquer sur l'onglet **`Configuration`**. Si vous utilisez un domaine géré, fermer le profil du serveur que vous souhaitez configurer, à partir de la case de sélection **`Profile`** qui se trouve en haut et à gauche.

   Étendre le menu **`Core`**, et sélectionner **`Logging`**.

2. **Modifier les attributs de `com.arjuna`.**
   Sélectionner l'onglet **`Log Categories`**. Sélectionner **`com.arjuna`**, puis **`Edit`** dans **`Details`**. Vous pourrez ajouter ici les informations de journalisation spécifiques à la classe. La classe **`com.arjuna`** est déjà présente. Vous pourrez modifier le niveau de journalisation et décider si vous souhaitez utiliser les handlers parents.

   Niveau de journalisation

   Le niveau de journalisation est **`WARN`** par défaut. Comme les transactions peuvent produire une grande quantité de messages de journalisation, la signification des niveaux de journalisation standard est légèrement différente pour l'enregistreur de transactions. En général, les messages avec des niveaux de gravité moins élevés que le niveau choisi seront ignorés.

   Niveaux de journalisation des transactions, du plus au moins détaillé.

   - TRACE
   - DEBOG
   - INFO
   - AVERTISSEMENT
   - ERREUR
   - FAILURE

   Utiliser les gestionnaires parents

   Indique si l'enregistreur d'événements doit envoyer ses sorties vers l'enregistreur d'événements parent. Le comportement par défaut est **`true`**.

3. Les changements prennent effet immédiatement.

Rapporter un bogue

11.6.3.3. Naviguer et gérer les transactions

Le CLI prend en charge la capacité de naviguer et de manipuler les enregistrements des transactions. Cette fonctionnalité est fournie par l'interaction entre le gestionnaire de transactions et l'API de gestion de JBoss EAP 6.

Le gestionnaire de transactions stocke des informations sur chaque transaction en attente et les participants impliqués dans la transaction, dans un stockage persistant appelé *object store*. L'API de gestion expose le store objet sous forme de ressource appelée **`log-store`**. Une opération API nommée **`probe`** lit les journaux de transactions et crée un noeud pour chaque journal. Vous pouvez invoquer la commande **`probe`** manuellement, quand vous souhaitez réactualiser le **`log-store`**. Il est normal pour les journaux de transaction d'apparaitre ou de disparaitre rapidement.

Exemple 11.2. Réactualiser le log store

Cette commande réactualise le log store des groupes de serveurs qui utilisent le profil par défaut **`default`** dans un domaine géré. Dans le cas d'un serveur autonome, supprimer **`profile=default`** de la commande.

```
/profile=default/subsystem=transactions/log-store=log-store/:probe
```

Exemple 11.3. Voir toutes les transactions préparées

Pour voir toutes les transactions préparées, commencer par réactualiser le log store (voir Exemple 11.2, « Réactualiser le log store »), puis exécuter la commande suivante, qui fonctionne de la même manière qu'une commande **`ls`** de système de fichiers.

```
ls /profile=default/subsystem=transactions/log-store=log-
store/transactions
```

Chaque transaction est visible, ainsi que son identifiant unique. Les opérations individuelles peuvent être exécutées pour une transaction individuelle (voir Gérer une transaction).

Gérer une transaction

Voir des attributs de transaction.

Pour voir des informations sur une transaction, comme son nom JNDI, son nom de produit EIS ou sa version, ou encore son statut, utiliser la commande CLI **`:read-resource`**.

```
/profile=default/subsystem=transactions/log-store=log-
store/transactions=0\:ffff7f000001\:-b66efc2\:4f9e6f8f\:9:read-resource
```

Voir tous les participants à une transaction.

Chaque journal de transaction contient un élément enfant nommé **`participants`**. Utiliser la commande CLI **`read-resource`** sur cet élément pour voir les participants des transactions. Les participants sont identifiés par leur nom JNDI.

```
/profile=default/subsystem=transactions/log-store=log-
```

```
store/transactions=0\:ffff7f000001\:-
b66efc2\:4f9e6f8f\:9/participants=java\:\/JmsXA:read-resource
```

Le résultat doit ressembler à ceci :

```
{
   "outcome" => "success",
   "result" => {
       "eis-product-name" => "HornetQ",
       "eis-product-version" => "2.0",
       "jndi-name" => "java:/JmsXA",
       "status" => "HEURISTIC",
       "type" => "/StateManager/AbstractRecord/XAResourceRecord"
   }
}
```

Le statut du résultat affiché ici est dans un état **HEURISTIC** et est susceptible d'être recouvré. Voir Recouvrement d'une transaction. pour plus d'informations.

Supprimer une transaction.

Chaque journal de transaction supporte une opération **:delete** pour effacer l'enregistrement qui représente la transaction.

```
/profile=default/subsystem=transactions/log-store=log-
store/transactions=0\:ffff7f000001\:-b66efc2\:4f9e6f8f\:9:delete
```

Recouvrement d'une transaction.

Chaque journal de transaction supporte le recouvrement par la commande CLI **:recover**.

Recouvrement des transactions heuristiques et des participants

- Si le statut de la transaction est **HEURISTIC**, l'opération de recouvrement change l'état en **PREPARE** et déclenche un recouvrement.
- Si l'un des participants de la transaction est heuristique, l'opération de recouvrement tente de reprendre l'opération **commit** (validation) à nouveau. En cas de succès, le participant est retiré du journal des transactions. Vous pouvez vérifier cela en exécutant à nouveau l'opération **:probe** sur le **log-store** et en vérifiant que le participant n'est plus inscrit. Si c'est le dernier participant, la transaction sera également supprimée.

Réactualiser le statut de la transaction qui a besoin d'être recouvrée.

Si une transaction a besoin d'être recouvrée, vous pourrez utiliser la commande CLI **:refresh** pour vous assurer qu'elle a toujours besoin d'être recouvrée, avant de tenter le recouvrement.

```
/profile=default/subsystem=transactions/log-store=log-
store/transactions=0\:ffff7f000001\:-b66efc2\:4f9e6f8f\:9:refresh
```

Voir les statistiques de transaction

Si les statistiques de Transaction manager sont activées, vous pouvez consulter les statistiques à propos du gestionnaire de transactions et du sous-système de transaction. Veuillez consulter Section 11.7.8.2, « Configurer le Transaction Manager (TM) (ou gestionnaire de transactions) » pour plus d'informations

sur l'activation des statistiques de Transaction manager.

Vous pouvez consulter les statistiques soit par la console de gestion basée-web, soit par l'interface CLI. de gestion Dans la console de gestion basée-web, les statistiques de transaction seront disponibles via **Runtime → Status → Subsystems → Transactions**. Les statistiques de transaction sont disponibles pour chaque serveur dans un domaine géré, également. Pour voir le statut d'un autre serveur, sélectionner **`Change Server`** situé en haut à gauche du menu, et sélectionner un serveur de la liste.

Le tableau suivant affiche chaque statistique disponible, sa description et la commande CLI de gestion pour afficher le statistique.

Tableau 11.12. Les statistiques de sous-système de transaction

Statistique	Description	Commande CLI
Total	Le nombre total de transactions exécutées par le gestionnaire de transactions sur ce serveur.	`/host=master/server=server-one/subsystem=transactions/:read-attribute(name=number-of-transactions,include-defaults=true)`
Validé	Le nombre de transactions validées exécutées par le gestionnaire de transactions sur ce serveur.	`/host=master/server=server-one/subsystem=transactions/:read-attribute(name=number-of-committed-transactions,include-defaults=true)`
Abandonné	Le nombre de transactions interrompues exécutées par le gestionnaire de transactions sur ce serveur.	`/host=master/server=server-one/subsystem=transactions/:read-attribute(name=number-of-aborted-transactions,include-defaults=true)`

Statistique	Description	Commande CLI
Délai expiré	Le nombre de transactions expirées exécutées par le gestionnaire de transactions sur ce serveur.	`/host=master/server=server-one/subsystem=transactions/:read-attribute(name=number-of-timed-out-transactions,include-defaults=true)`
Heuristiques	Pas disponible dans la console de gestion. Nombre de transactions dans un état heuristique.	`/host=master/server=server-one/subsystem=transactions/:read-attribute(name=number-of-heuristics,include-defaults=true)`
Transactions In-Flight	Pas disponible dans la console de gestion. Nombre de transactions commencées non achevées.	`/host=master/server=server-one/subsystem=transactions/:read-attribute(name=number-of-inflight-transactions,include-defaults=true)`
Origine de l'échec - Applications	Le nombre de transactions échouées dont l'origine de l'échec était une application.	`/host=master/server=server-one/subsystem=transactions/:read-attribute(name=number-of-application-rollbacks,include-defaults=true)`

Statistique	Description	Commande CLI
Origine de l'échec - Ressources	Le nombre de transactions échouées dont l'origine de l'échec était une ressource.	`/host=master/server=server-one/subsystem=transactions/:read-attribute(name=number-of-resource-rollbacks,include-defaults=true)`

Rapporter un bogue

11.7. UTILISER LES TRANSACTIONS JTA

11.7.1. Transactions JTA

Introduction

Les procédures suivantes sont utiles quand vous aurez besoin d'utiliser des transactions dans votre application.

- Section 11.7.2, « Transactions de contrôle »
- Section 11.7.3, « Démarrer une transaction »
- Section 11.7.5, « Valider une transaction »
- Section 11.7.6, « Annuler une transaction »
- Section 11.7.7, « Gérer un résultat heuristique dans une transaction »
- Section 11.7.8.2, « Configurer le Transaction Manager (TM) (ou gestionnaire de transactions) »
- Section 11.7.9.1, « Erreurs de transactions »

Rapporter un bogue

11.7.2. Transactions de contrôle

Introduction

Cette liste de procédures expose les différentes façons de contrôler les transactions dans vos applications utilisant les API JTA ou JTS.

- Section 11.7.3, « Démarrer une transaction »
- Section 11.7.5, « Valider une transaction »
- Section 11.7.6, « Annuler une transaction »
- Section 11.7.7, « Gérer un résultat heuristique dans une transaction »

Rapporter un bogue

11.7.3. Démarrer une transaction

Cette procédure montre comment démarrer une nouvelle transaction JTA, ou comment participer à une transaction distribuée en utilisant le protocole JTS (Java Transaction Service).

Transactions distribuées

Dans une transaction distribuée, les participants à la transaction se trouvent dans des applications séparées sur de multiples serveurs. Si un participant rejoint une transaction déjà existante, plutôt que de créer un nouveau contexte de transaction, les deux (ou plus) participants qui partagent le contexte participent à une transaction distribuée. Pour utiliser des transactions distribuées, vous devez configurer l'ORB. Veuillez consulter la section *ORB Configuration* du *Administration and Configuration Guide* pour plus d'informations sur la configuration ORB.

1. **Obtenez une instance de `UserTransaction`.**
   Vous pouvez obtenir une instance par JNDI, injection, ou un contexte d'EJB, si l'EJB utilise des transactions gérées-bean, par le biais de l'annotation **`@TransactionManagement(TransactionManagementType.BEAN)`**.

   - **JNDI**

     ```
     new InitialContext().lookup("java:comp/UserTransaction")
     ```

   - **Injection**

     ```
     @Resource UserTransaction userTransaction;
     ```

   - **Contexte**

     - Pour un bean stateless/stateful :

       ```
       @Resource SessionContext ctx;
       ctx.getUserTransaction();
       ```

     - Pour un bean message-driven :

       ```
       @Resource MessageDrivenContext ctx;
       ctx.getUserTransaction()
       ```

2. **Appeler `UserTransaction.begin()` une fois connecté à votre source de données.**

   ```
   ...
   try {
       System.out.println("\nCreating connection to database: "+url);
       stmt = conn.createStatement();  // non-tx statement
       try {
           System.out.println("Starting top-level transaction.");
           userTransaction.begin();
           stmtx = conn.createStatement(); // will be a tx-statement
           ...
       }
   }
   ```

Participer à une transaction en utilisant l'API JTS.

L'un des avantages des EJB est que le conteneur gère toutes les transactions. Si vous avez configuré l'ORB, le conteneur gérera les transactions distribuées pour vous.

Résultat :

La transaction est lancée. Toute utilisation de votre source de données sera transactionnelle jusqu'à ce que vous validiez ou annuliez la transaction.

NOTE

Pour voir un exemple complet Section 11.9.3, « Exemple de transaction JTA ».

Rapporter un bogue

11.7.4. Transactions imbriquées

Les transactions imbriquées ne sont prises en charge que lorsque vous utilisez des transactions distribuées avec les API JTS. De plus, beaucoup de fournisseurs de base de données ne prennent pas en charge les transactions imbriquées, renseignez-vous donc auprès de votre fournisseur de base de données avant d'ajouter des transactions imbriquées à votre application.

Les spécifications OTS permettent un type limité de transactions imbriquées, où le protocole de validation de sous-transaction est le même que pour les transactions de niveau supérieur. Il existe deux phases : une phase **`prepare`** et une phase **`commit`** ou **`abort`**. Ce type de transaction imbriquée peut mener à des résultats incohérents, comme un coordinateur de sous-transaction en cours de validation qui découvre qu'une ressource ne peut être validée. Il se peut que le coordinateur ne puisse être capable de dire aux ressources validées de s'interrompre, causant ainsi un résultat heuristique. Cette transaction stricte imbriquée OTS est disponible via l'interface **`CosTransactions::SubtransactionAwareResource`**.

La mise en œuvre de JTS de JBoss EAP 6 prend en charge ce type de transaction imbriquée. Elle prend également en charge un type de transaction imbriquée avec un protocole de validation multiphase, évitant ainsi les problèmes que l'on peut rencontrer avec le modèle OTS strict. Ce type de transaction imbriquée est disponible via le **`ArjunaOTS::ArjunaSubtranAwareResource`**. Il est poussé par un protocole de validation à deux phases à chaque fois qu'une transaction imbriquée est validée.

Pour créer une transaction imbriquée, créer une nouvelle transaction à l'intérieur d'une transaction parente. Veuillez consulter Section 11.7.3, « Démarrer une transaction » pour plus d'informations sur la création d'une transaction.

L'effet d'une transaction imbriquée dépend de la validation/rollback de ses transactions intégrées. Ces effets sont annulés si la transaction intégrée s'interrompt, même si la transaction imbriquée a été validée.

Rapporter un bogue

11.7.5. Valider une transaction

Cette procédure montre comment valider une transaction en utilisant JTA (Java Transaction Api). Cet API est utilisé à la fois pour les transactions locales et les transactions distribuées. Les transactions distribuées sont gérées par le JTS (Java Transaction Server) et nécessitent la configuration d'un ORB (Object Request Broker). Pour plus d'informations sur la configuration d'un ORB, veuillez consulter la section *ORB Configuration* du *Administration and Configuration Guide*.

Conditions préalables

Vous devez démarrer une transaction avant de pouvoir la valider. Pour plus d'informations sur le démarrage d'une transaction, veuillez consulter Section 11.7.3, « Démarrer une transaction ».

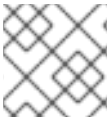
1. **Appeler la méthode `commit()` sur `UserTransaction`.**
   Lorsque vous appelez la méthode **`commit()`** sur **`UserTransaction`**, le gestionnaire de transactions tente de valider la transaction.

```
@Inject
private UserTransaction userTransaction;

public void updateTable(String key, String value)
    EntityManager entityManager =
entityManagerFactory.createEntityManager();
    try {
        userTransaction.begin():
        <!-- Perform some data manipulation using entityManager -->
        ...
        // Commit the transaction
        userTransaction.commit();
    } catch (Exception ex) {
        <!-- Log message or notify Web page -->
        ...
        try {
            userTransaction.rollback();
        } catch (SystemException se) {
            throw new RuntimeException(se);
        }
        throw new RuntimeException(e);
    } finally {
        entityManager.close();
    }
}
```

2. **Si vous utilisez des CMT (Container Managed Transactions), vous n'avez pas besoin de valider manuellement.**
   Si vous configurez votre bean pour qu'il puisse utiliser des transactions gérées conteneur, le conteneur devra gérer le cyle de vie des transactions pour vous sur la base d'annotations que vous aurez configurées dans le code.

Résultat

Votre source de données valide et votre transaction se termine, ou une exception est lancée.



NOTE

Pour voir un exemple complet Section 11.9.3, « Exemple de transaction JTA ».

Rapporter un bogue

11.7.6. Annuler une transaction

Cette procédure montre comment annuler une transaction en utilisant JTA (Java Transaction Api). Cet API est utilisé à la fois pour les transactions locales et les transactions distribuées. Les transactions

distribuées sont gérées par le JTS (Java Transaction Server) et nécessitent la configuration d'un ORB (Object Request Broker). Pour plus d'informations sur la configuration d'un ORB, veuillez consulter la section *ORB Configuration* du *Administration and Configuration Guide*.

Conditions préalables

Vous devez démarrer une transaction avant de pouvoir la supprimer. Pour plus d'informations sur le démarrage d'une transaction, veuillez consulter Section 11.7.3, « Démarrer une transaction ».

1. **Appeler la méthode `rollback()` sur `UserTransaction`.**
   Lorsque vous appelez la méthode **`rollback()`** sur **`UserTransaction`**, le gestionnaire de transactions tentera de valider la transaction.

```
@Inject
private UserTransaction userTransaction;

public void updateTable(String key, String value)
    EntityManager entityManager =
entityManagerFactory.createEntityManager();
    try {
        userTransaction.begin():
        <!-- Perform some data manipulation using entityManager -->
          ...
          // Commit the transaction
        userTransaction.commit();
    } catch (Exception ex) {
        <!-- Log message or notify Web page -->
        ...
        try {
            userTransaction.rollback();
        } catch (SystemException se) {
            throw new RuntimeException(se);
        }
        throw new RuntimeException(e);
    } finally {
        entityManager.close();
    }
}
```

2. **Si vous utilisez des CMT (Container Managed Transactions), vous n'avez pas besoin de supprimer la transaction manuellement.**
   Si vous configurez votre bean pour qu'il puisse utiliser des transactions gérées conteneur, le conteneur devra gérer le cyle de vie des transactions pour vous sur la base d'annotations que vous aurez configurées dans le code.

Résultat

Votre transaction est annulée par le gestionnaire de transactions.

NOTE

Voici un exemple complet Section 11.9.3, « Exemple de transaction JTA ».

Rapporter un bogue

11.7.7. Gérer un résultat heuristique dans une transaction

Cette procédure vous montre comment gérer un résultat heuristique dans une transaction JTA, qu'elle soit locale ou distribuée, en utilisant JTS (Java Transaction Service). Pour utiliser des transactions distribuées, il vous faut configurer l'ORB. Veuillez consulter la section *ORB Configuration* du *Administration and Configuration Guide* pour plus d'informations sur la configuration ORB.

Les résultats de transaction heuristique sont rares et ont généralement des causes exceptionnelles. Le mot *heuristique* signifie «manuellement», et c'est ainsi que ces résultats doivent généralement être traités. Voir Section 11.4.4, « Résultats heuristiques » pour plus d'informations sur les résultats de l'opération heuristique.

Procédure 11.5. Gérer un résultat heuristique dans une transaction

1. **Déterminer la cause**
   En général, la cause d'un résultat heuristique pour une transaction est qu'un gestionnaire de ressources a promis qu'il pourrait commettre ou annuler, et il n'a pas pu remplir sa promesse. Cela pouvait être dû à un problème avec un composant de tierce partie, la couche d'intégration entre le composant tiers et la plate-forme EAP 6 ou bien la plate-forme JBoss EAP 6 elle-même.

   De loin, les deux causes les plus communes d'erreurs heuristiques sont des erreurs temporaires situées dans l'environnement ou des erreurs de code liées aux gestionnaires de ressources.

2. **Solution aux échecs temporaires de l'environnement**
   Normalement, s'il y a un échec temporaire dans votre environnement, vous le saurez avant d'être au courant de l'erreur heuristique. Il peut s'agir d'une défaillance de réseau ou de matériel, d'une erreur de base de données, d'une panne de courant, d'un hôte ou de tout autre cause.

   En cas de résultat heuristique ayant lieu dans un environnement de test, au cours de test de stress, cela vous fournit des informations sur les points faibles de votre environnement.

   AVERTISSEMENT

   La plate-forme JBoss EAP 6 restaurera automatiquement les transactions qui ne sont pas dans un état heuristique au moment de la défaillance, mais elle ne tentera pas de restaurer les transactions heuristiques.

3. **Contacter les fournisseurs de gestionnaires de ressources**
   Si vous n'expérimentez aucun échec évident dans votre environnement, ou si le résultat heuristique est facilement reproductible, c'est probablement une erreur de codage. Contacter les fournisseurs de tierce partie pour savoir si une solution est disponible. Si vous pensez que le problème est dans le gestionnaire de transactions de la plate-forme JBoss EAP 6 elle-même, veuillez prendre contact avec Red Hat Global Support Services.

4. **Dans un environnement de test, supprimer les logs et démarrez à nouveau la plate-forme JBoss EAP 6.**
   Dans un environnement de test, ou bien dans les cas où vous ne vous souciez pas de l'intégrité des données, le fait de supprimer les journaux des transactions et de redémarrer la plate-forme JBoss EAP 6 se débarrasse du résultat heuristique. Les journaux des transactions sont situés dans `EAP_HOME/standalone/data/tx-object-store/` pour les serveurs autonomes, ou

dans ***EAP_HOME*/domain/servers/*SERVER_NAME*/data/tx-object-store** pour les domaines gérés, par défaut. Dans le cas d'un domaine géré, *SERVER_NAME* se rapporte au nom du serveur individuel qui participe à un groupe de serveurs.

NOTE

L'emplacement du journal de transactions dépend aussi du store d'objets en usage et des valeurs définies des paramètres ***oject-store-relative-to*** et ***object-store-path***. Pour les journaux de système de fichier (journaux en mémoire fantôme standard - or standard shadow - et les journaux HornetQ), la localisation de direction par défaut est utilisée, mais lorsque vous utilisez un magasin d'objets JDBC, les journaux des transactions sont stockés dans une base de données.

5. **Résoudre le résultat à la main**
   Le processus de résolution de résultat à la main dépend des circonstances particulières de l'incident. Normalement, vous aurez besoin de procéder aux étapes suivantes, en les appliquant à votre situation.

   a. Identifier les gestionnaires de ressources impliqués.

   b. Examiner l'état dans le gestionnaire de transactions et dans les gestionnaires de ressources.

   c. Forcer manuellement le nettoyage des journaux et la reconciliation des données dans un ou plusieurs des composants impliqués.

   La façon dont procéder à ces étapes va delà de la portée de ce document.

Rapporter un bogue

11.7.8. Temps d'expiration des transactions

11.7.8.1. Les délais d'attente des transactions

Afin de préserver l'atomicité et de respecter la norme ACID des transactions, certaines parties d'une transaction peuvent être longues. Les participants à la transaction doivent verrouiller des pièces de sources de données lorsqu'ils commettent, et le gestionnaire de transactions doit attendre d'avoir des nouvelles de chaque participant à une transaction avant de pouvoir tous les diriger à commettre ou à annuler. Les ressources peuvent être verrouillés indéfiniment pour cause de panne de matériel ou de réseau.

Les délais d'attente de transactions peuvent être associés à des transactions afin de contrôler leur cycle de vie. Si un seuil de délai d'attente passe avant que la transaction s'engage ou soit annulée, le délai d'attente provoque l'annulation de l'opération automatiquement.

Vous pouvez configurer les valeurs de délai d'attente par défaut pour tout le sous-système de la transaction, ou bien, vous pouvez désactiver les valeurs de délai d'attente par défaut et spécifier les délais d'attente sur une base d'une transaction.

Rapporter un bogue

11.7.8.2. Configurer le Transaction Manager (TM) (ou gestionnaire de transactions)

Vous pouvez configurer le Transaction Manager (TM) à l'aide de la console de gestion sur le web ou en ligne de commande. Pour chaque commande ou option donnée, on assume que vous exécutez JBoss EAP 6 comme un domaine géré. Si vous utilisez un serveur autonome ou que vous souhaitez modifier un profil différent de la valeur par défaut **`default`**, il se peut que vous ayez à modifier les étapes et les commandes de la manière suivante.

Notes sur les commandes d'exemple

- Pour la console de gestion, le profil par défaut **`default`** est celui qui sera sélectionné quand vous vous connectez. Si vous souhaitez modifier la configuration du gestionnaire de transactions (TM) dans un autre profile, sélectionnez votre profile à la place, et non pas **`default`**, pour chaque instruction.

  De même, substituez votre profil à la place du profil par défaut **`default`** pour les commandes CLI de l'exemple.

- Si vous utilisez un serveur autonome, un seul profil existe. Ignorer toute instruction pour choisir un profil spécifique. Dans les commandes CLI, retirer la partie **`/profile=default`** des commandes d'échantillon.

NOTE

Pour que les options du TM soient visibles dans la console de gestion ou dans l'interface CLI, le sous-système **`transactions`** doit être activé. Il est activé par défaut, et il faut pour cela qu'un certain nombre d'autres sous-systèmes fonctionnent correctement, donc il est improbable qu'il soit désactivé.

Configurer le TM par la console de gestion

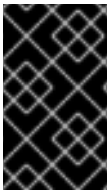
Pour configurer le gestionnaire de transactions (TM) à l'aide de la console de gestion sur le web, sélectionnez l'onglet **`Configuration`** en haut de l'écran. Si vous utilisez un domaine géré, vous avez le choix de plusieurs profils. Choisir le bon **`Profile`** de la boîte de sélection dans la partie supérieure gauche de l'écran. Étendez le menu **`Container`**, et sélectionnez **`Transactions`**.

Vous verrez la plupart des options dans la page de configuration du gestionnaire de transactions. Les options **`Recovery`** sont cachées par défaut. Cliquer sur l'onglet **`Recovery`** pour voir les options de recouvrement. Cliquer sur **`Edit`** pour éditer une des options. Les changements prendront place immédiatement.

Cliquer sur l'étiquette **`Need Help?`** pour afficher le texte d'aide en ligne.

Configurer le TM par l'interface CLI

Dans l'interface CLI, vous pouvez configurer le TM en utilisant une série de commandes. Les commandes commencent toutes par **`/profile=default/subsystem=transactions/`** pour un domaine géré avec **`default`** de profil, ou par **`/subsystem=transactions`** pour un serveur autonome.



IMPORTANT

HornetQ n'autorise pas plusieurs instances à partager un message log store. Si vous configurez plusieurs instances d'HornetQ, chaque instance devra avoir son propre message log store.

Tableau 11.13. Options de configuration du TM

Option	Description	Commande CLI
Activer les statistiques	Indique s'il faut activer les statistiques de transaction. Ces statistiques se trouvent dans la console de gestion dans la section **`Subsystem Metrics`** de l'onglet **`Runtime`**.	**`/profile=default/subsystem=transactions/:write-attribute(name=enable-statistics,value=true)`**
Délai d'attente par défaut	Délai d'attente de transaction par défaut. La valeur par défaut est de **`300`** secondes. Vous pouvez la remplacer par programmation, sur la base d'une transaction.	**`/profile=default/subsystem=transactions/:write-attribute(name=default-timeout,value=300)`**
Chemin	Le chemin d'accès relatif ou absolu du système de fichiers dans lequel le cœur du gestionnaire de transactions stocke les données. Par défaut, la valeur est un chemin d'accès relatif à la valeur de l'attribut **`relative-to`**. Le chemin est utilisé pour sauvegarder l'ID du processus quand la balise ***`process-id`*** (ou le ***`process-id-uuid CLI attribute`***) sont utilisés.	**`/profile=default/subsystem=transactions/:write-attribute(name=path,value=var)`**
Relatif à	Fait référence une configuration de chemin global dans le modèle du domaine. La valeur par défaut correspond au répertoire de données de JBoss EAP 6, qui correspond à la valeur de la propriété **`jboss.server.data.dir`**, dont la valeur par défaut est ***`EAP_HOME`*`/domain/data/`** pour un domaine géré, ou ***`EAP_HOME`*`/standalone/data/`** pour une instance de serveur autonome. La valeur de l'attribut **`path`** du gestionnaire de transactions est relative à ce chemin.	**`/profile=default/subsystem=transactions/:write-attribute(name=relative-to,value=jboss.server.data.dir)`**
Chemin de Store Objet	Un chemin de système de fichiers relatif ou absolu où le store objet TM stocke des données. Relatif, par défaut, à la valeur du paramètre **`object-store-relative-to`**.	**`/profile=default/subsystem=transactions/:write-attribute(name=object-store-path,value=tx-object-store)`**

Option	Description	Commande CLI
Chemin de Store Objet Relatif à	Fait référence une configuration de chemin global dans le modèle du domaine. La valeur par défaut correspond au répertoire de données de JBoss EAP 6, qui correspond à la valeur de la propriété **jboss.server.data.dir**, ayant comme valeur par défaut ***EAP_HOME*/domain/data/** pour un domaine géré, ou ***EAP_HOME*/standalone/data/** pour une instance de serveur autonome. La valeur de l'attribut **path** de l'attribut du gestinnaire de transactions est relative à ce chemin.	**/profile=default/subsystem=transactions/:write-attribute(name=object-store-relative-to,value=jboss.server.data.dir)**
Liaisons de sockets	Indique le nom de la liaison du socket utilisé par le gestionnaire de transactions pour la récupération et la création des identificateurs de transactions, lorsque le mécanisme du socket est utilisé. Se référer à **processus-id-socket-max-ports** pour plus d'informations sur la génération de l'identificateur unique. Les liaisons de socket sont spécifiées par le groupe de serveurs dans l'onglet **Server** de la console de gestion.	**/profile=default/subsystem=transactions/:write-attribute(name=socket-binding,value=txn-recovery-environment)**
Listener de recouvrement	Indique si oui ou non le processus de recouvrement de transaction doit écouter au socket de réseau. La valeur par défaut est **false**.	**/profile=default/subsystem=transactions/:write-attribute(name=recovery-listener,value=false)**

Les options suivantes sont pour une utilisation avancée et ne peuvent être modifiées que par ligne de commande. Soyez prudent lors de leur modification à partir de la configuration par défaut. Communiquer avec Red Hat Global Support Services pour plus d'informations.

Tableau 11.14. Options de configuration du TM avancées

Option	Description	Commande CLI
jts	Indique si l'on doit utiliser les transactions Java Transaction Service (JTS). La valeur par défaut est **false**, qui utilise des transactions JTA uniquement.	**/profile=default/subsystem=transactions/:write-attribute(name=jts,value=false)**

Option	Description	Commande CLI
Identifiant de nœud	L'identificateur de noeud de gestionnaire de transactions. Cette option est requise dans les situations suivantes : • Pour les communications de JTS à JTS • Quand deux gestionnaires de transactions accèdent à des gestionnaires de ressources partagées • Quand deux gestionnaires de transactions accèdent à des object stores partagés Le **`node-identifier`** doit être unique pour chaque gestionnaire de transaction car il doit assurer l'intégrité des données pendant le recouvrement. Le **`node-identifier`** doit également être unique à JTA car plusieurs noeuds peuvent entrer en interraction avec le même gestionnaire de ressources ou partager un object store de transations.	**`/profile=default/subsystem=transactions/:write-attribute(name=node-identifier,value=1)`**
process-id-socket-max-ports	Le gestionnaire de transactions crée un identifiant unique pour chaque journal des transactions. Deux mécanismes différents sont fournis pour générer des identificateurs uniques : un mécanisme basé sur le socket et un mécanisme fondé sur l'identificateur de processus du processus. Dans le cas de l'identifiant basé-socket, le socket est ouvert et son numéro de port est utilisé pour l'identifiant. Si le port est déjà utilisé, on cherchera le port suivant jusqu'à ce qu'un port libre soit trouvé. Les **`processus-id-socket-max-ports`** représentent le nombre maximal de sockets que le TM va essayer avant d'abandonner. La valeur par défaut est **`10`**.	**`/profile=default/subsystem=transactions/:write-attribute(name=process-id-socket-max-ports,value=10)`**

Option	Description	Commande CLI
process-id-uuid	Définir à **`true`** avec un identifiant de processus pour créer un identifiant unique pour chaque transaction. Sinon, le mécanisme basé socket sera utilisé. La valeur par défaut est **`true`**. Se référer à **`process-id-socket-max-ports`** pour obtenir davantage d'informations.	**`/profile=default/subsystem=transactions/:write-attribute(name=process-id-uuid,value=true)`**
use-hornetq-store	Utiliser les mécanismes de stockage journalisés de HornetQ au lieu du stockage basé sur des fichiers, pour les journaux de transactions. Ceci est désactivé par défaut, mais peut améliorer les performances I/O. Il n'est pas recommandé pour les transactions JTS sur les gestionnaires de transactions séparés. Quand vous changez cette option, le serveur doit démarrer à nouveau à l'aide de la commande **`shutdown`** pour que le changement puisse prendre effet.	**`/profile=default/subsystem=transactions/:write-attribute(name=use-hornetq-store,value=false)`**

Rapporter un bogue

11.7.9. Gestion des erreurs de transactions JTA

11.7.9.1. Erreurs de transactions

Les erreurs de transactions sont difficiles à résoudre car souvent dépendantes du timing. Voici quelques erreurs courantes et quelques idées pour les résoudre.

NOTE

Ces recommandations ne s'appliquent pas aux erreurs heuristiques. Si vous rencontrez des erreurs heuristiques, voir Section 11.7.7, « Gérer un résultat heuristique dans une transaction » et contacter Red Hat Global Support Services pour obtenir une assistance.

La transaction a expiré mais le thread de logique commerciale ne s'en est pas aperçu.

Ce type d'erreur se manifeste souvent quand Hibernate n'est pas en mesure d'obtenir une connexion de base de données pour le chargement différé. Si cela survient fréquemment, vous pourrez augmenter la valeur du temps d'expiration. Voir Section 11.7.8.2, « Configurer le Transaction Manager (TM) (ou gestionnaire de transactions) ».

Si cela n'est pas possible, vous pourrez sans doute régler votre environnement extérieur pour aller plus vite, ou restructurer votre code pour être plus efficace. Contacter Red Hat Global Services si vous avez des problèmes avec le temps d'expiration.

La transaction exécute déjà sur un thread, ou reçoit une exception `NotSupportedException`

L'exception **`NotSupportedException`** indique habituellement que vous avez tenté d'insérer une transaction JTA, et que ce n'est pas pris en charge. Si vous n'étiez pas en train d'essayer d'insérer une transaction, il est probable qu'une autre transaction avait commencé dans une tâche de pool de threads, mais a terminé la tâche sans suspendre, ni mettre fin à la transaction.

Les application utilisent normalement **`UserTransaction`** , qui s'en occupe automatiquement. Si tel est le cas, il y a peut-être un problème dans le framework.

Si votre code utilise les méthodes **`TransactionManager`** ou **`Transactions`** directement, considérez le comportement suivant lors de la validation ou de l'annulation d'une transaction. Si votre code utilise les méthodes du **`TransactionManager`** pour contrôler vos transactions, la validation ou l'annulation d'une transaction dissociera la transaction du thread actuel. Toutefois, si votre code utilise les méthodes **`Transaction`** , la transaction peut ne pas être associée au thread en cours d'exécution, et il faudra la dissocier de son thread manuellement, avant de la retourner au pool de threads.

Vous n'êtes pas en mesure d'enlister une seconde ressource locale

Cette erreur a lieu si vous essayez d'enlister une seconde ressource non-XA dans une transaction. Si vous avez besoin de ressources multiples pour une transaction, elles doivent être XA.

Rapporter un bogue

11.8. CONFIGURATION ORB

11.8.1. CORBA (Common Object Request Broker Architecture)

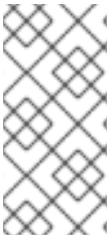
Common Object Request Broker Architecture (CORBA) est une norme qui autorise les applications et services à travailler ensemble même lorsqu'ils sont écrits dans de multiples langages normalement incompatibles ou lorsqu'ils sont hébergés sur des plateformes différentes. Les requêtes CORBA sont émises par un composant côte serveur appelé *Object Request Broker (ORB)*. JBoss EAP 6 fournit une instance ORB au moyen du composant JacORB.

L'ORB est utilisé en interne pour les transactions *Java Transaction Service (JTS)*, et peut également être utilisé par votre propre application.

Rapporter un bogue

11.8.2. Configurer l'ORB pour les transactions JTS

Dans une installation JBoss EnAP 6 par défaut, l'ORB est désactivé. Vous pouvez activer l'ORB en utilisant l'interface CLI de ligne de commande.



NOTE

Dans un domaine géré, le sous-système JacORB est disponible dans les profils **`full`** et **`full-ha`** uniquement. Dans un serveur autonome, il est disponible uniquement quand vous utilisez les configurations **`standalone-full.xml`** ou **`standalone-full-ha.xml`**.

Procédure 11.6. Configurer l'ORB par la console de gestion

1. **Voir les paramètres de configuration du profil.**

Sélectionner **Configuration** dans la partie supérieure de la console de gestion. Si vous utilisez un domaine géré, sélectionner soit le profil **full** ou **full-ha** à partir de la boîte de dialogue de sélection en haut à gauche.

2. **Modifier les paramètres Initializers**
   Étendre sur le menu **Subsystems**. Étendre le menu **Container**, et sélectionner **JacORB**.

   Sur le formulaire qui apparaît sur l'écran principal, sélectionner l'onglet **Initializers**, et cliquer sur le bouton **Edit**.

   Activer les intercepteurs de sécurité en configurant la valeur de **Security** à **active**.

   Pour activer ORB sur JTS, définir la valeur des **Transaction Interceptorss** à **active**, au lieu de la valeur par défaut **spec**.

   Voir le lien **Need Help?** sur le formulaire pour accéder à des explications sur ces valeurs. Cliquer sur **Save** quand vous aurez fini de modifier les valeurs.

3. **Configuration ORB avancée**
   Voir les autres sections du formulaire pour les options de configuration avancées. Chaque section inclut un lien **Need Help?** avec des informations détaillées sur les paramètres.

Configurer l'ORB par l'interface CLI

Vous pouvez configurer chaque aspect de l'ORB à l'aide de l'interface CLI. Les commandes suivantes configurent les initialisateurs aux mêmes valeurs que celles de la procédure ci-dessus, pour la console de gestion. Il s'agit de la configuration minimale pour l'ORB, si utilisé avec JTS.

Ces commandes sont configurées pour un domaine de sécurité utilisant le profil **full**. Si nécessaire, modifier le profil pour qu'il convienne mieux à celui que vous aurez besoin de configurer. Si vous utilisez un serveur autonome, n'utilisez pas la portion **/profile=full** des commandes.

Exemple 11.4. Activer les intercepteurs de sécurité

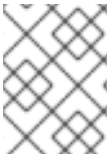
```
/profile=full/subsystem=jacorb/:write-attribute(name=security,value=on)
```

Exemple 11.5. Activer les transactions dans le sous-système JacORB

```
/profile=full/subsystem=jacorb/:write-
attribute(name=transactions,value=on)
```

Exemple 11.6. Activer JTS dans le sous-système de transactions

```
/profile=full/subsystem=transactions:write-
attribute(name=jts,value=true)
```



NOTE

Pour l'activation JTS, le serveur doit être démarré à nouveau car le rechargement ne suffira pas.

Rapporter un bogue

11.9. RÉFÉRENCES DE TRANSACTIONS

11.9.1. Erreurs et exceptions pour les transactions JBoss

Pour obtenir des informations lancées par les méthodes de la classe **`UserTransaction`**, voir la spécification *UserTransaction API* dans http://download.oracle.com/javaee/1.3/api/javax/transaction/UserTransaction.html.

Rapporter un bogue

11.9.2. Limitations sur les transactions JTA

Les transactions JTA ne peuvent pas être au courant de la distribution à travers les instances multiples de JBoss EAP 6. Pour ce comportement, utiliser les transactions JTS.

Pour pouvoir utiliser les transactions JTS, vous devrez configurer l'ORB, qui inclut l'activation des transactions dans le sous-système JacORB, puis configurez le sous-système JTS.

- Section 11.8.2, « Configurer l'ORB pour les transactions JTS »

Rapporter un bogue

11.9.3. Exemple de transaction JTA

Cet exemple illustre comment démarrer, valider et annuler une transaction JTA. Vous devrez ajuster les paramètres de connexion et de source de données pour accommoder votre environnement, et mettre deux tableaux de tests dans votre base de données.

Exemple 11.7. Exemple de transaction JTA

```
public class JDBCExample {
    public static void main (String[] args) {
        Context ctx = new InitialContext();
        // Change these two lines to suit your environment.
        DataSource ds = (DataSource)ctx.lookup("jdbc/ExampleDS");
        Connection conn = ds.getConnection("testuser", "testpwd");
        Statement stmt = null; // Non-transactional statement
        Statement stmtx = null; // Transactional statement
        Properties dbProperties = new Properties();

        // Get a UserTransaction
        UserTransaction txn = new
InitialContext().lookup("java:comp/UserTransaction");

        try {
            stmt = conn.createStatement();  // non-tx statement
```

```
            // Check the database connection.
            try {
                stmt.executeUpdate("DROP TABLE test_table");
                stmt.executeUpdate("DROP TABLE test_table2");
            }
            catch (Exception e) {
                // assume not in database.
            }

            try {
                stmt.executeUpdate("CREATE TABLE test_table (a
INTEGER,b INTEGER)");
                stmt.executeUpdate("CREATE TABLE test_table2 (a
INTEGER,b INTEGER)");
            }
            catch (Exception e) {
            }

            try {
                System.out.println("Starting top-level transaction.");

                txn.begin();

                stmtx = conn.createStatement(); // will be a tx-
statement

                // First, we try to roll back changes

                System.out.println("\nAdding entries to table 1.");

                stmtx.executeUpdate("INSERT INTO test_table (a, b)
VALUES (1,2)");

                ResultSet res1 = null;

                System.out.println("\nInspecting table 1.");

                res1 = stmtx.executeQuery("SELECT * FROM test_table");

                while (res1.next()) {
                    System.out.println("Column 1: "+res1.getInt(1));
                    System.out.println("Column 2: "+res1.getInt(2));
                }
                System.out.println("\nAdding entries to table 2.");

                stmtx.executeUpdate("INSERT INTO test_table2 (a, b)
VALUES (3,4)");
                res1 = stmtx.executeQuery("SELECT * FROM test_table2");

                System.out.println("\nInspecting table 2.");

                while (res1.next()) {
                    System.out.println("Column 1: "+res1.getInt(1));
                    System.out.println("Column 2: "+res1.getInt(2));
                }
```

```
                System.out.print("\nNow attempting to rollback
changes.");

                txn.rollback();

                // Next, we try to commit changes
                txn.begin();
                stmtx = conn.createStatement();
                ResultSet res2 = null;

                System.out.println("\nNow checking state of table 1.");

                res2 = stmtx.executeQuery("SELECT * FROM test_table");

                while (res2.next()) {
                    System.out.println("Column 1: "+res2.getInt(1));
                    System.out.println("Column 2: "+res2.getInt(2));
                }

                System.out.println("\nNow checking state of table 2.");

                stmtx = conn.createStatement();

                res2 = stmtx.executeQuery("SELECT * FROM test_table2");

                while (res2.next()) {
                    System.out.println("Column 1: "+res2.getInt(1));
                    System.out.println("Column 2: "+res2.getInt(2));
                }

                txn.commit();
            }
            catch (Exception ex) {
                ex.printStackTrace();
                System.exit(0);
            }
        }
        catch (Exception sysEx) {
            sysEx.printStackTrace();
            System.exit(0);
        }
    }
}
```

Rapporter un bogue

11.9.4. Documentation API pour JBoss Transactions JTA

La documentation API pour le sous-système de transactions de JBoss EAP 6 est disponible à l'endroit suivant :

- UserTransaction - http://download.oracle.com/javaee/1.3/api/javax/transaction/UserTransaction.html

Si vous utilisez Red Hat JBoss Developer Studio pour développer vos applications, la documentation API est incluse dans le menu **`Help`**.

Rapporter un bogue

11.9.5. Limitations dans le processus de recouvrement XA

Le recouvrement XA a les limitations suivantes :

Le journal de transactions

Si le serveur JBoss EAP échoue après qu'une méthode de validation **`XAResource`** réussisse et validifie la transaction, mais avant que le coordinateur puisse mettre le journal à jour, vous verrez sans doute le message suivant apparaître dans le journal quand vous démarrerez le serveur à nouveau :

```
ARJUNA016037: Could not find new XAResource to use for recovering non-
serializable XAResource XAResourceRecord
```

C'est parce que lors de la restauration, le gestionnaire de transactions de JBoss (Jboss Transaction Manager) voit les participants à la transaction dans le journal et tente la validation à nouveau. Ensuite, le gestionnaire de transactions de JBoss suppose que les ressources sont validées et ne tente plus de valider à nouveau. Dans ce cas, il est possible d'ignorer cet avertissement car la transaction est validée et il n'y a aucune perte de données.

Pour éviter cet avertissement, définir la valeur de la propriété com.arjuna.ats.jta.xaAssumeRecoveryComplete à **`true`** . Cette propriété est vérifiée à chaque fois qu'une nouvelle instance de **`XAResource`** ne peut pas être trouvée parmi les **`XAResourceRecovery`** enregistrées. Si défini sur **`true`**, le recouvrement assume que les tentatives précédentes ont réussi, et l'instance peut être supprimée du journal sans besoin de nouvelle tentative. Cette propriété doit être utilisée avec soin car elle est globale et si elle est utilisée par de façon erronée, on se retrouve avec des instances **`XAResource`** restantes non validées.

La restauration n'est pas automatique pour les transactions JTS qui suivent un plantage de serveur à la fin de XAResource.prepare().

Si le serveur JBoss EAP se plante suite à un appel de méthode **`XAResource prepare()`**, tous les participants à XAResources sont verrouillés dans l'état « prepared » et y demeurent lors du redémarrage du serveur. La transaction n'est pas restaurée et les ressources restent verrouillées jusqu'à ce que l'opération arrive à expiration ou qu'un DBA restaure les ressources et efface les transactions du journal manuellement.

Le recouvrement périodique peut avoir lieu sur des transactions déjà validées.

Quand le serveur est sous charge intensive, le journal du serveur peut contenir les messages d'avertissement suivants, suivis d'un stackstace :

```
ARJUNA016027: Local XARecoveryModule.xaRecovery got XA exception
XAException.XAER_NOTA: javax.transaction.xa.XAException
```

Sous forte charge, le temps de traitement pris par une transaction peut chevaucher le calendrier d'activité de la procédure de restauration périodique. Le processus de restauration périodique détecte la transaction en cours d'exécution et essaie d'initier une restauration, mais en fait, la transaction se poursuit jusqu'à la fin. Au moment où le processus de restauration périodique tente mais échoue à la

restauration, il enregistre l'échec de restauration dans le journal de serveur. La cause sous-jacente de cette question sera abordée dans une prochaine version, mais en attendant, une solution de contournement est disponible.

Augmenter l'intervalle entre les deux phases du processus de restauration en définissant la propriété com.arjuna.ats.jta.orphanSafetyInterval sur une valeur supérieure à la valeur par défaut de 10 000 millisecondes. Une valeur de 40 000 millisecondes est recommandée. Veuillez noter que cela ne résout pas le problème, au contraire, cela diminue la probabilité qu'il se produira et que le message d'avertissement s'affichera dans le journal.

Rapporter un bogue

CHAPITRE 12. HIBERNATE

12.1. HIBERNATE CORE

Hibernate Core est une bibliothèque de mappages objet/relationnel. Elle fournit le framework de mappage des tables de la base de données, permettant aux applications d'éviter une interaction directe avec la base de données.

Pour plus d'information, consulter Section 12.2.2, « Hibernate EntityManager » et Section 12.2.1, « JPA ».

Rapporter un bogue

12.2. JPA (JAVA PERSISTENCE API)

12.2.1. JPA

L'API Java Persistence (JPA) est standard pour l'utilisation de la persistance dans les projets Java. Les applications Java EE 6 utilisent la spécification Java Persistence 2.0, documentée ici : http://www.jcp.org/en/jsr/detail?id=317.

Hibernate EntityManager implémente les interfaces de programmation et les règles de cycle de vie définies par la spécification. Il fournit une solution Java Persistence complète pour la plate-forme JBoss EAP 6.

JBoss EAP 6 est conforme à 100% à la spécification Java Persistence 2.0. Hibernate fournit également quelques fonctionnalités supplémentaires à la spécification.

Pour commencer avec JPA et JBoss EAP 6, voir les quickstarts **`bean-validation`**, **`greeter`**, et **`kitchensink`** : Section 1.4.1.1, « Accès aux Quickstarts ».

Rapporter un bogue

12.2.2. Hibernate EntityManager

Hibernate EntityManager implémente les interfaces de programmation et les règles de cycles de vie définies dans JPA 2.0 specification. Il procure une solution Java Persistence complète pour la plateforme JBoss EAP 6.

Pour plus d'informations, voir Section 12.2.1, « JPA » et Section 12.1, « Hibernate Core ».

Rapporter un bogue

12.2.3. Guide de départ

12.2.3.1. Créer un projet JPA dans le Red Hat JBoss Developer Studio

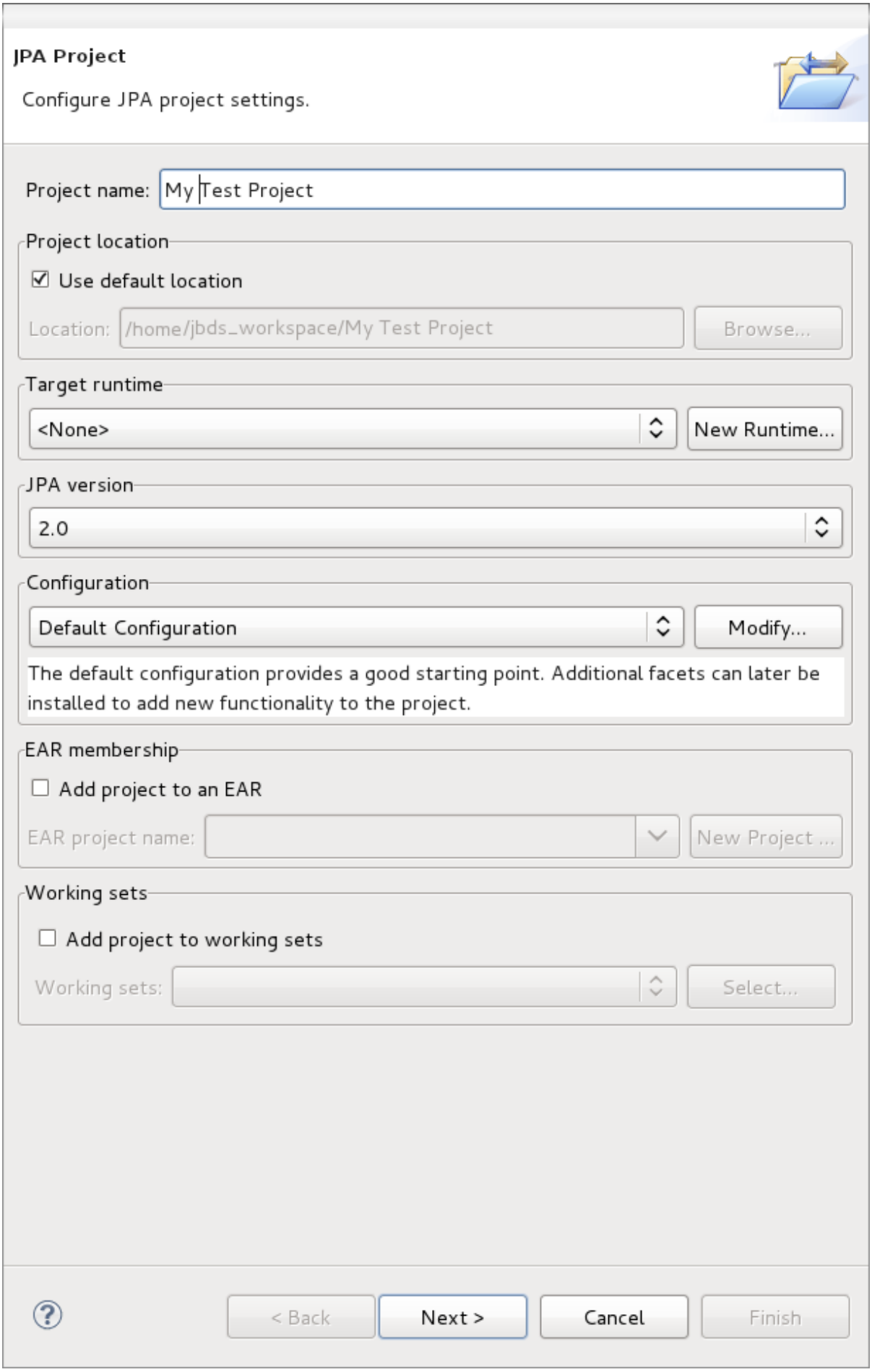
Résumé

Cet exemple indique les étapes à effectuer pour créer un projet JPA dans Red Hat JBoss Developer Studio.

Procédure 12.1. Créer un projet JPA dans le Red Hat JBoss Developer Studio

1. Dans la fenêtre Red Hat JBoss Developer Studio, cliquer sur **Fichier** → **Nouveau** → **Projet JPA**.

2. Dans la boîte de dialogue du projet, taper le nom du projet.

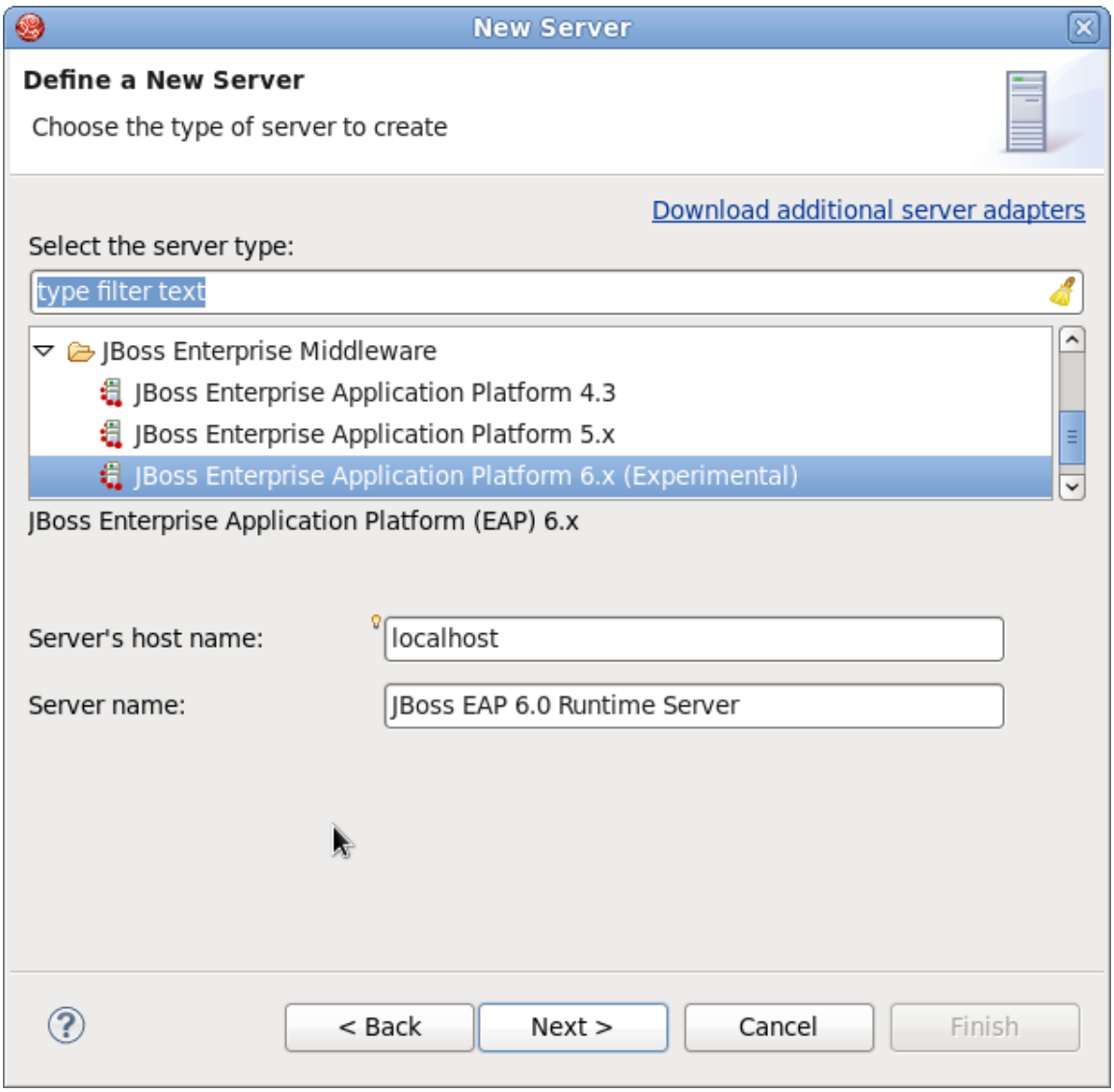


3. Sélectionner un runtime cible depuis le menu déroulant.

4. a. Si aucun runtime cible n'est disponible, cliquer sur **`New Runtime`**.

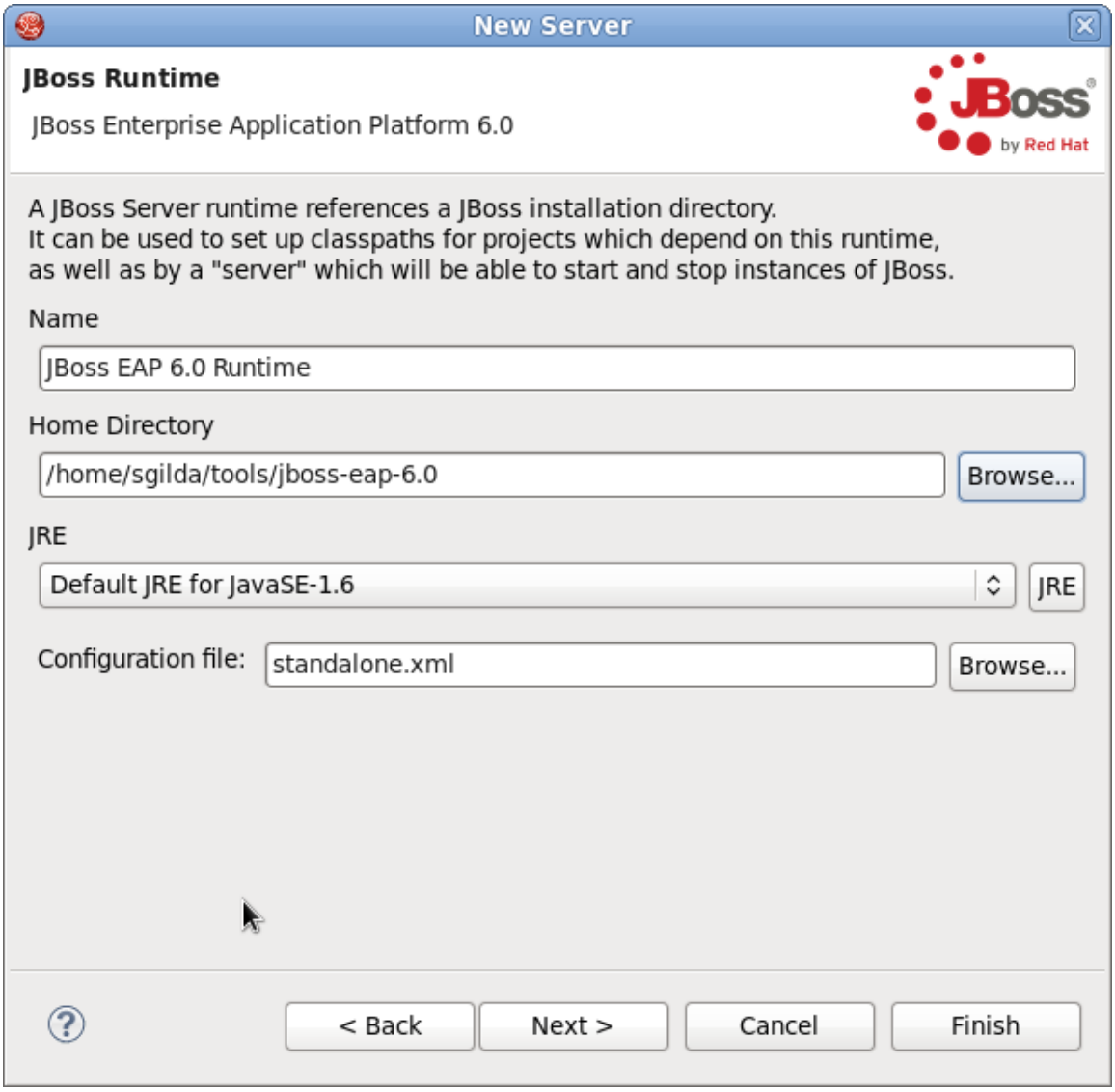
   b. Trouver le Dossier JBoss Community dans la liste.

   c. Sélectionner JBoss Enterprise Application Platform 6.x Runtime



   d. Cliquer sur **`Suivant`**.

   e. Dans le champ Répertoire Personnel, cliquer sur **`Naviguer`** pour définir le dossier source JBoss EAP comme Répertoire Personnel.



f. Cliquer sur **`Terminé`**.

5. Cliquer sur **`Suivant`**.
6. Laisser les dossiers source sur la fenêtre de chemin de build en mode défaut, et cliquer sur **`Suivant`**.
7. Dans le menu déroulant de la Plateforme, vous assurer que Hibernate (JPA 2.x) est sélectionné.
8. Cliquer sur **`Terminé`**.
9. Si cela vous est demandé, choisissez si vous souhaitez ouvrir la fenêtre de perspective JPA.

Rapporter un bogue

12.2.3.2. Créer le fichier de paramètres de persistance dans Red Hat JBoss Developer Studio

Résumé

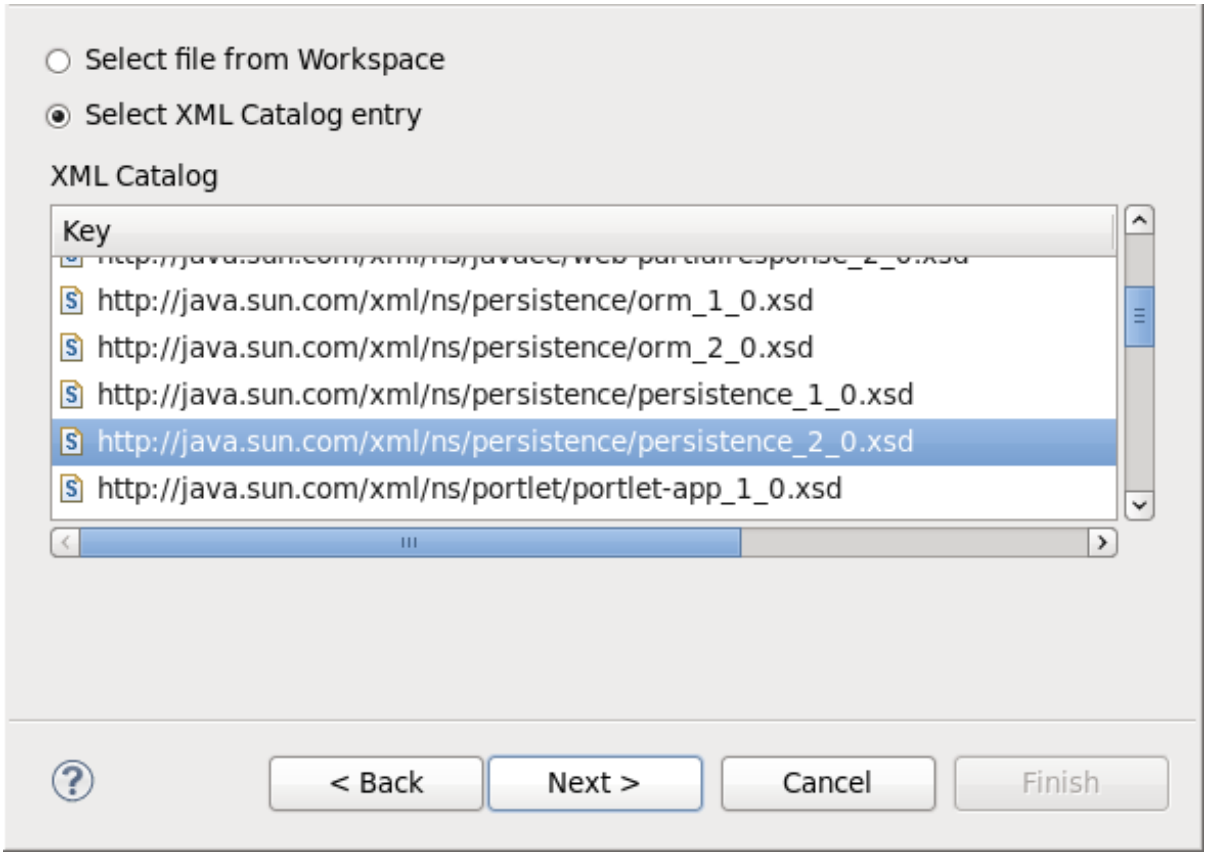
Ce sujet couvre le processus de création du fichier **`persistence.xml`** dans un projet Java qui utilise Red Hat JBoss Developer Studio.

Conditions préalables

- Section 1.3.1.4, « Démarrer Red Hat JBoss Developer Studio »

Procédure 12.2. Créer et configurer un nouveau fichier de configuration de persistance

1. Ouvrir un projet EJB 3.x dans Red Hat JBoss Developer Studio.
2. Cliquer à droite sur le répertoire racine du projet dans le panneau **`Project Explorer`**
3. Sélectionner **Nouveau** → **Autre...**.
4. Sélectionner **`XML File`** à partir du dossier **`XML`** et cliquer sur **`Next`**.
5. Sélectionner le dossier **`ejbModule/META-INF`** comme répertoire parent.
6. Nommer le fichier **`persistence.xml`** et cliquer sur **`Suivant`**.
7. Sélectionner **`Create XML file from an XML schema file`** (créer fichier à partir d'un fichier xml) et cliquer sur **`Next`**.
8. Sélectionner **`http://java.sun.com/xml/ns/persistence/persistence_2.0.xsd`** dans la liste **`Select XML Catalog entry`** et cliquer sur **`Next`**.



9. Cliquer sur **`Terminer`** pour créer le fichier.

Résultat :

Le fichier **`persistence.xml`** a été créé dans le dossier **`META-INF/`** et est prêt pour la configuration. Un exemple de fichier se trouve ici : Section 12.2.3.3, « Exemple de fichier de configuration de persistance »

Rapporter un bogue

12.2.3.3. Exemple de fichier de configuration de persistance

Exemple 12.1. persistence.xml

```
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
   version="2.0">
   <persistence-unit name="example" transaction-type="JTA">
      <provider>org.hibernate.ejb.HibernatePersistence</provider>
      <jta-data-source>java:jboss/datasources/ExampleDS</jta-data-
source>
      <mapping-file>ormap.xml</mapping-file>
      <jar-file>TestApp.jar</jar-file>
      <class>org.test.Test</class>
      <shared-cache-mode>NONE</shared-cache-mode>
      <validation-mode>CALLBACK</validation-mode>
      <properties>
         <property name="hibernate.dialect"
value="org.hibernate.dialect.H2Dialect"/>
         <property name="hibernate.hbm2ddl.auto" value="create-drop"/>
      </properties>
   </persistence-unit>
</persistence>
```

Rapporter un bogue

12.2.3.4. Créer le fichier de configuration Hibernate dans Red Hat JBoss Developer Studio

Conditions préalables

- Section 1.3.1.4, « Démarrer Red Hat JBoss Developer Studio »

Résumé

Ce sujet couvre le processus de création du fichier **`hibernate.cfg.xml`** dans un projet Java qui utilise Red Hat JBoss Developer Studio.

Procédure 12.3. Créer un nouveau fichier de configuration Hibernate

1. Ouvrir un projet Java dans Red Hat JBoss Developer Studio
2. Cliquer à droite sur le répertoire racine du projet dans le panneau **`Project Explorer`**
3. Sélectionner **New** → **Other...**.
4. Sélectionner **`Hibernate Configuration File`** depuis le dossier **`Hibernate`** et cliquer sur **`Next`**.

5. Sélectionner le répertoire **src/** et cliquer sur **Next**.
6. Configurer :
   - Nom de la fabrique à sessions
   - Dialecte de base de données
   - Classe pilote
   - URL de connexion
   - Nom d'utilisateur
   - Mot de passe
7. Cliquer sur **Finich** pour créer le fichier.

Résultat :

Le fichier **hibernate.cfg.xml** a été créé dans le dossier **src/**. Un exemple de fichier est disponible ici : Section 12.2.3.5, « Exemple de fichier de configuration Hibernate ».

Rapporter un bogue

12.2.3.5. Exemple de fichier de configuration Hibernate

Exemple 12.2. hibernate.cfg.xml

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

    <session-factory>

        <!-- Datasource Name -->
        <property name="connection.datasource">ExampleDS</property>

        <!-- SQL dialect -->
        <property
name="dialect">org.hibernate.dialect.H2Dialect</property>

        <!-- Enable Hibernate's automatic session context management --
>
        <property
name="current_session_context_class">thread</property>

        <!-- Disable the second-level cache  -->
        <property
name="cache.region.factory_class">org.hibernate.cache.NoCacheProvider</p
roperty>

        <!-- Echo all executed SQL to stdout -->
```

```
        <property name="show_sql">true</property>

        <!-- Update the database schema on startup -->
        <property name="hbm2ddl.auto">update</property>

        <mapping
resource="org/hibernate/tutorial/domain/Event.hbm.xml"/>

    </session-factory>

</hibernate-configuration>
```

Rapporter un bogue

12.2.4. Configuration

12.2.4.1. Propriétés de configuration de Hibernate

Tableau 12.1. Propriétés

Nom de propriété	Description
hibernate.dialect	Le nom de classe d'un **`org.hibernate.dialect.Dialect`** Hibernate. Permet à Hibernate de générer SQL optimisé pour une base de données relative particulière. Dans la plupart des cas, Hibernate pourra choisir la mise en œuvre **`org.hibernate.dialect.Dialect`** correcte, selon les **`JDBC metadata`** renvoyées par le pilote JDBC.
hibernate.show_sql	Booléen. Écrit toutes les déclarations SQL à la console. C'est une alternative pour définir la catégorie du journal **`org.hibernate.SQL`** sur **`debug`**.
hibernate.format_sql	Booléen. Mettre en forme le SQL dans le journal et la console.
hibernate.default_schema	Qualifie les noms de tableaux non qualifiés avec le schéma/espace tableau donné dans le SQL généré.
hibernate.default_catalog	Qualifie les noms de tableaux non qualifiés avec le catalogue donné dans le SQL généré.
hibernate.session_factory_name	Le **`org.hibernate.SessionFactory`** sera automatiquement lié à ce nom dans JNDI après avoir été créé. Par exemple, **`jndi/composite/name`**.

Nom de propriété	Description
hibernate.max_fetch_depth	Définit une "profondeur" maximum pour l'arborescence de récupération de la jointure externe pour les associations à fin unique (un-à-un, plusieurs-à-un). Un **0** désactive la récupération de jointure externe par défaut. La valeur recommandée se situe entre **0** et **3**.
hibernate.default_batch_fetch_size	Définit une taille par défaut pour la récupération en lot d'associations de Hibernate. Les valeurs recommandées sont **4**, **8**, et **16**.
hibernate.default_entity_mode	Définit un mode par défaut pour la réprésentation d'entité pour toutes les sessions ouvertes à partir de **SessionFactory**. Les valeurs comprennent : **dynamic-map**, **dom4j**, **pojo**.
hibernate.order_updates	Booléen. Oblige Hibernate à ordonner les mises à jour SQL par la valeur clé principale des éléments mis à jour. Cela se traduira en un nombre inférieur de blocages de transactions dans des systèmes hautement simultanés.
hibernate.generate_statistics	Booléen. Si autorisé, Hibernate va collecter les statistiques utiles pour le réglage de performance.
hibernate.use_identifier_rollback	Booléen. Si autorisé, les propriétés d'identifiant générées seront restaurées à leur valeur par défaut lorsque les objets sont supprimés.
hibernate.use_sql_comments	Booléen. Si activé, Hibernate génèrera des commentaires à l'intérieur de SQL, pour faciliter de débogage. La valeur par défaut est **false**.
hibernate.id.new_generator_mappings	Booléen. Cette propriété est appropriée lorsque **@GeneratedValue** est utilisée. Elle indique si les nouvelles implémentations **IdentifierGenerator** sont utilisées pour **javax.persistence.GenerationType.AUTO**, **javax.persistence.GenerationType.TABLE** et **javax.persistence.GenerationType.SEQUENCE**. La valeur par défaut est **true**.

IMPORTANT

Dans **hibernate.id.new_generator_mappings**, les nouvelles applications devront garder la valeur par défaut **true**. Les applications existantes qui utilisent Hibernate 3.3.x devront sans doute modifier cette valeur à **false** pour continuer à utiliser un objet de séquence ou un générateur basé-table et pour maintenir la compatibilité rétro-active.

Rapporter un bogue

12.2.4.2. Hibernate JDBC et Propriétés de Connexion

Tableau 12.2. Propriétés

Nom de propriété	Description
hibernate.jdbc.fetch_size	Valeur non nulle qui détermine la taille de récupération JDBC (appelle **`Statement.setFetchSize()`**).
hibernate.jdbc.batch_size	Une valeur non nulle active l'utilisation des mises à jour de lots de JDBC2 par Hibernate. Les valeurs recommandées se situent entre **`5`** et **`30`**.
hibernate.jdbc.batch_versioned_data	Booléen. Définir cette propriété sur **`true`** si le pilote JDBC renvoie des nombres de lignes corrects avec **`executeBatch()`**. Hibernate utilisera ensuite un DML en lot pour les données versionnées automatiquement. La valeur par défaut est **`false`**.
hibernate.jdbc.factory_class	Sélectionner un **`org.hibernate.jdbc.Batcher`** personnalisé. La plupart des applications n'auront pas besoin de cette propriété de configuration.
hibernate.jdbc.use_scrollable_resultset	Booléen. Active l'utilisation des ensembles de résultats déroulants de JDBC2 par Hibernate. Cette propriété n'est nécessaire que lorsque l'on utilise des connexions JDBC fournies par l'utilisateur. Sans quoi, Hibernate utilise des métadonnées de connexion.
hibernate.jdbc.use_streams_for_binary	Booléen. Ceci est une propriété de niveau système. Utiliser les flux lors de l'écriture/la lecture de types **`binary`** ou **`serializable`** depuis/vers JDBC.
hibernate.jdbc.use_get_generated_keys	Booléen. Active l'utilisation de JDBC3 **`PreparedStatement.getGeneratedKeys()`** pour récupérer les cléfs générées en mode natif après insertion. Requiert le pilote JDBC3+ et JRE1.4+. Défini sur false si le pilote JDBC rencontre des problèmes avec les générateurs d'identifiants Hibernate. Par défaut, il tente de déterminer les capacités du pilote en utilisant les métadonnées de connexion.
hibernate.connection.provider_class	Le nom de classe d'un **`org.hibernate.connection.ConnectionProvider`** personnalisé qui fournit des connexions JDBC à Hibernate.
hibernate.connection.isolation	Définit le niveau d'isolation des transactions JDBC. Vérifier **`java.sql.Connection`** pour des valeurs compréhensibles, mais veuillez noter que la plupart des bases de données ne prennent pas en charge tous les niveaux d'isolation et que quelques unes définissent des isolations additionnelles, non-standard. Les valeurs standard sont **`1, 2, 4, 8`**.

Nom de propriété	Description
hibernate.connection.autocommit	Booléen. Cette propriété n'est pas recommandée à l'utilisation. Active l'autovalidation pour les connexions « en commun » de JDBC.
hibernate.connection.release_mode	Indique à quel moment Hibernate devra diffuser les connexions JDBC. Par défaut, une connexion JDBC est maintenue jusqu'à ce que la session soit explicitement fermée ou déconnectée. La valeur par défaut **`auto`** choisira **`after_statement`** pour les stratégies de transaction JTA et CMT, et **`after_transaction`** pour la stratégie de transaction JDBC. Les valeurs disponibles sont **`auto`** (par défaut) | **`on_close`** | **`after_transaction`** | **`after_statement`**. Ce paramètre n'affecte que les **`Session`**s renvoyées depuis **`SessionFactory.openSession`**. Pour les **`Session`**s obtenues par **`SessionFactory.getCurrentSession`**, l'implémentation **`CurrentSessionContext`** configurée pour l'utilisation contrôle le mode de publication de connexion pour ces **`Session`**s.
hibernate.connection.*<propertyName>*	Passer la propriété JDBC *<propertyName>* à **`DriverManager.getConnection()`**.
hibernate.jndi.*<propertyName>*	Passer la proriété *<propertyName>* à **`InitialContextFactory`** de JNDI.

Rapporter un bogue

12.2.4.3. Propriétés d'Hibernate Cache

Tableau 12.3. Propriétés

Nom de propriété	Description
`hibernate.cache.region.factory_class`	Le nom de classe d'un **`CacheProvider`** personnalisé.
`hibernate.cache.use_minimal_puts`	Booléen. Optimise l'opération cache de second niveau pour minimiser les écritures, au détriment de lectures plus fréquentes. Cette configuration est surtout utile pour les caches clusterisés et, dans Hibernate 3, est activée par défaut pour les implémentations cache clusterisées.
`hibernate.cache.use_query_cache`	Booléen. Active le cache de recherche. Les recherches individuelles doivent toujours être définies comme cachables.

Nom de propriété	Description
hibernate.cache.use_second_level_cache	Booléen. Utilisé pour désactiver totalement le cache de second niveau, qui est activé par défaut pour les classes qui spécifient un mappage **<cache>**.
hibernate.cache.query_cache_factory	Le nom de classe d'une interface **QueryCache** personnalisée. La valeur par défaut est le **StandardQueryCache** intégré.
hibernate.cache.region_prefix	Un préfixe à utiliser pour les noms régionaux de cache de second niveau.
hibernate.cache.use_structured_entries	Booléen. Force Hibernate à stocker des données dans un cache de second niveau dans un format plus amical pour l'utilisateur.
hibernate.cache.default_cache_concurrency_strategy	Configuration utilisée pour donner le nom de la statégie qu'il faut **org.hibernate.annotations.CacheConcurrencyStrategy** quand **@Cacheable** ou **@Cache** sont utilisés. **@Cache(strategy="..")** est utilisé pour remplacer cette valeur par défaut.

Rapporter un bogue

12.2.4.4. Propriétés de transaction Hibernate

Tableau 12.4. Propriétés

Nom de propriété	Description
hibernate.transaction.factory_class	Le nom de classe d'une **TransactionFactory** à utiliser avec l'API de **Transaction** de Hibernate. Valeur par défaut : **JDBCTransactionFactory**.
jta.UserTransaction	Un nom JNDI utilisé par **JTATransactionFactory** pour obtenir la **UserTransaction** JTA depuis le serveur d'applications.
hibernate.transaction.manager_lookup_class	Le nom de classe d'une **TransactionManagerLookup**. Requis lorsque la mise en cache de niveau JVM est activée ou lorsque le générateur hilo est utilisé dans un environnement JTA.
hibernate.transaction.flush_before_completion	Booléen. Si activé, la session sera automatiquement vidée avant la phase d'achèvement de la transaction. Une gestion de contexte de session automatique et intégrée est préférée.

Nom de propriété	Description
hibernate.transaction.auto_close_session	Booléen. Si activé, la session sera automatiquement fermée après la phase d'achèvement de la transaction. Une gestion de contexte de session automatique et intégrée est préférable.

Rapporter un bogue

12.2.4.5. Propriétés Hibernate diverses

Tableau 12.5. Propriétés

Nom de propriété	Description
hibernate.current_session_context_class	Fournit une stratégie personnalisée pour l'étendue de la **Session** « actuelle ». Les valeurs comprennent **jta** | **thread** | **managed** | **custom.Class**.
hibernate.query.factory_class	Choisir l'implémentation de l'analyseur de HQL : **org.hibernate.hql.internal.ast.ASTQueryTranslatorFactory** ou **org.hibernate.hql.internal.classic.ClassicQueryTranslatorFactory**.
hibernate.query.substitutions	Utilisé pour mapper à partir de jetons dans les requêtes Hibernate aux jetons SQL (les jetons peuvent avoir un nom fonctionnel ou littéral). Par exemple, **hqlLiteral=SQL_LITERAL, hqlFunction=SQLFUNC**.
hibernate.hbm2ddl.auto	Valide ou exporte automatiquement le schéma DDL vers la base de données quand la **SessionFactory** est créée. Avec **create-drop**, le schéma de base de données sera déposé lorsque la **SessionFactory** est fermée explicitement. Les options de valeur de propriété sont **validate** | **update** | **create** | **create-drop**
hibernate.hbm2ddl.import_files	Les noms séparés par des virgules des fichiers optionnels contenant des déclarations SQL DML exécutées pendant la création de **SessionFactory**. Cela est utile pour tester ou démontrer. Par exemple, en ajoutant des déclarations INSERT, la base de données peut être remplie avec un lot minimum de données quand elle est déployée. Exemple de valeur : **/humans.sql,/dogs.sql**. L'ordre de fichier compte, puisque les déclarations d'un fichier donné sont exécutées avant la déclaration du fichier suivant. Ces déclarations ne sont exécutées que si le schéma est créé (c'est-à-dire si **hibernate.hbm2ddl.auto** est défini comme **create** ou **create-drop**).

Nom de propriété	Description
`hibernate.hbm2ddl.import_files_sql_extractor`	Le nom de classe d'un **`ImportSqlCommandExtractor`** personnalisé. Valeur par défaut de **`SingleLineSqlCommandExtractor`** intégré. Cela est utile pour implémenter un analyseur dédié qui extrait une déclaration unique de SQL depuis chaque fichier d'importation. Hibernate fournit également **`MultipleLinesSqlCommandExtractor`**, qui prend en charge des instructions/commentaires et des chaînes citées réparties sur de multiples lignes (point-virgule obligatoire à la fin de chaque déclaration).
`hibernate.bytecode.use_reflection_optimizer`	Booléen. Ceci est une propriété de niveau système, qui ne peut pas être définie dans le fichier **`hibernate.cfg.xml`**. Elle permet l'utilisation de la manipulation bytecode au lieu de la réflexion d'exécution. La réflexion peut parfois être utile lors de la résolution de problèmes. Hibernate requiert toujours soit CGLIB ou javassist même si l'optimizer est éteint.
`hibernate.bytecode.provider`	Javassist et cglib peuvent tous les deux être utilisés comme moteurs de manipulation de byte. La valeur par défaut est **`javassist`**. La valeur de propriété est soit **`javassist`** soit **`cglib`**.

Rapporter un bogue

12.2.4.6. Dialectes SQL Hibernate

IMPORTANT

La propriété **`hibernate.dialect`** devrait être définie sur la sous-classe **`org.hibernate.dialect.Dialect`** correcte pour la base de données de l'application. Si un dialecte est indiqué, Hibernate utilisera des valeurs par défaut raisonnables pour quelques unes des autres propriétés. Cela signifie qu'elles n'ont pas besoin d'être spécifiées manuellement.

Tableau 12.6. Dialectes SQL (`hibernate.dialect`)

RDBMS	Dialecte
DB2	**`org.hibernate.dialect.DB2Dialect`**
DB2 AS/400	**`org.hibernate.dialect.DB2400Dialect`**
DB2 OS390	**`org.hibernate.dialect.DB2390Dialect`**

RDBMS	Dialecte
Firebird	`org.hibernate.dialect.FirebirdDialect`
FrontBase	`org.hibernate.dialect.FrontbaseDialect`
H2 Database	`org.hibernate.dialect.H2Dialect`
HypersonicSQL	`org.hibernate.dialect.HSQLDialect`
Informix	`org.hibernate.dialect.InformixDialect`
Ingres	`org.hibernate.dialect.IngresDialect`
Interbase	`org.hibernate.dialect.InterbaseDialect`
Mckoi SQL	`org.hibernate.dialect.MckoiDialect`
Microsoft SQL Server 2000	`org.hibernate.dialect.SQLServerDialect`
Microsoft SQL Server 2005	`org.hibernate.dialect.SQLServer2005Dialect`
Microsoft SQL Server 2008	`org.hibernate.dialect.SQLServer2008Dialect`
Microsoft SQL Server 2012	`org.hibernate.dialect.SQLServer2008Dialect`
MySQL5	`org.hibernate.dialect.MySQL5Dialect`
MySQL5 avec InnoDB	`org.hibernate.dialect.MySQL5InnoDBDialect`
MySQL avec MyISAM	`org.hibernate.dialect.MySQLMyISAMDialect`
Oracle (toute version)	`org.hibernate.dialect.OracleDialect`
Oracle 9i	`org.hibernate.dialect.Oracle9iDialect`

RDBMS	Dialecte
Oracle 10g	`org.hibernate.dialect.Oracle10gDialect`
Oracle 11g	`org.hibernate.dialect.Oracle10gDialect`
Pointbase	`org.hibernate.dialect.PointbaseDialect`
PostgreSQL	`org.hibernate.dialect.PostgreSQLDialect`
PostgreSQL 9.2	`org.hibernate.dialect.PostgreSQL82Dialect`
Postgres Plus Advanced Server	`org.hibernate.dialect.PostgresPlusDialect`
Progress	`org.hibernate.dialect.ProgressDialect`
SAP DB	`org.hibernate.dialect.SAPDBDialect`
Sybase	`org.hibernate.dialect.SybaseASE15Dialect`
Sybase 15.7	`org.hibernate.dialect.SybaseASE157Dialect`
Sybase Anywhere	`org.hibernate.dialect.SybaseAnywhereDialect`

Rapporter un bogue

12.2.5. Caches de second niveau

12.2.5.1. Les caches de second niveau

Un cache de second niveau est un store de données locales qui contient des informations persistées en dehors de la session d'application. Le cache est géré par le fournisseur de persistances, qui améliore le temps d'exécution en gardant les données séparées de l'application.

JBoss EAP 6 prend en charge la mise en cache dans les cas suivants :

- Web Session Clustering
- Stateful Session Bean Clustering

- SSO Clustering
- Cache de second niveau d'Hibernate

Chaque conteneur cache définit un "repl" et un "dist". Ces caches ne doivent pas être utilisés directement par les applications utilisateur.

Rapporter un bogue

12.2.5.2. Configurer un cache de second niveau pour Hibernate

Ce sujet couvre les besoins de configuration pour permettre à Infinispan d'agir en tant que second niveau de cache pour Hibernate.

Procédure 12.4. Créer et modifier le fichier `hibernate.cfg.xml`

1. **Créer le fichier hibernate.cfg.xml**
   Créer le fichier **`hibernate.cfg.xml`** dans le chemin de classe du déploiement. Pour plus de détails, voir Section 12.2.3.4, « Créer le fichier de configuration Hibernate dans Red Hat JBoss Developer Studio » .

2. Ajouter ces lignes d'XML au fichier **`hibernate.cfg.xml`** de votre application. Le XML doit être à l'intérieur des balises <session-factory>:

   ```
   <property
   name="hibernate.cache.use_second_level_cache">true</property>
   <property name="hibernate.cache.use_query_cache">true</property>
   ```

3. Ajouter un élément ci-dessous à la section <session-factory> de votre fichier **`hibernate.cfg.xml`** :

   - **Si le CacheManager Infinispan est lié au JNDI :**

     ```
     <property name="hibernate.cache.region.factory_class">
         org.hibernate.cache.infinispan.JndiInfinispanRegionFactory
     </property>
     <property name="hibernate.cache.infinispan.cachemanager">
         java:CacheManager
     </property>
     ```

   - **Si le CacheManager Infinispan est autonome :**

     ```
     <property name="hibernate.cache.region.factory_class">
         org.hibernate.cache.infinispan.InfinispanRegionFactory
     </property>
     ```

Résultat

Infinispan est configuré au second niveau de cache dans Hibernate.

Rapporter un bogue

12.3. ANNOTATIONS HIBERNATE

12.3.1. Annotations Hibernate

Tableau 12.7. Annotations définies d'Hibernate

Annotation	Description
AccessType	Type d'accès de propriété
Tous	Définit une association ToOne pointant vers plusieurs types d'entités. L'association du type d'entité correspondant se fait à travers une colonne discriminatoire de métadonnées. Ce type de mappage devrait seulement être marginal.
AnyMetaDef	Définit les métadonnées @Any et @manyToAny.
AnyMedaDefs	Définit les ensembles de métadonnées @Any et @ManyToAny. Peut être défini au niveau d'entité ou de paquetage.
BatchSize	Taille de lot pour chargement SQL.
Cache	Ajouter une stratégie de mise en cache à une entité root ou une collection.
Cascade	Appliquer une stratégie de cascade sur une association
Check	Contraintes de vérification SQL arbitraires qui peuvent être définies aux niveaux de classe, de propriété ou de collection.
Columns	Prend en charge une gamme de colonnes. Utile pour les mappages de type utilisateur de composant.
ColumnTransformer	Expression SQL personnalisée utilisée pour lire la valeur depuis et écrire une valeur vers une colonne. Utiliser pour le chargement/enregistrement d'objet direct ainsi que les requêtes. L'expression de lecture doit contenir exactement un "?" pour la valeur.
ColumnTransformers	Annotation plurielle pour @ColumnTransformer. Utile lorsque plus d'une colonne utilise ce comportement.
DiscriminatorFormula	Formule discriminatoire à placer à l'entité de root.
DiscriminatorOptions	Annotation optionnelle pour exprimer les propriétés discriminatoires spécifiques à Hibernate.
Entity	Prolonge l'entité avec les caractéristiques Hibernate.

Annotation	Description
Fetch	Définit la stratégie de récupération utilisée pour l'association donnée.
FetchProfile	Définit le profil de stratégie de récupération
FetchProfiles	Annotation plurielle pour @FetchProfile.
Filter	Ajoute des filtres à une entité ou une entité de cible d'une collection.
FilterDef	Définition de Filtre.
FilterDefs	Gamme de définitions de filtre.
FilterJoinTable	Ajoute des filtres pour une collection rejoindre une table
FilterJoinTables	Ajoute de multiples @FilterJoinTable à une collection
Filters	Ajoute de multiples @Filters.
Formula	A utiliser comme remplacement pour @Column dans la plupart des emplacements. La formule doit être un fragment SQL valide.
Generated	Cette propriété annotée est générée par la base de données.
GenericGenerator	Annotation de générateur décrivant toute sorte de générateur Hibernate d'une manière non-typée.
GenericGenerators	Gamme de définitions de générateur génériques.
Immutable	Marque une Entité ou une Collection comme immuable. Si aucune annotation, l'élément est mutable. Une entité immuable peut ne pas être mise à jour par l'application. Les mises à jour vers une entité immuable seront ignorées, mais aucune exception ne sera levée. Placer @Immutable sur une collection rend la collection immuable, ce qui signifie que les additions et suppressions vers et depuis la collection ne sont pas autorisées. Une HibernateException est levée dans ce cas.
Index	Définit un indice de base de données.

Annotation	Description
JoinFormula	A utiliser comme remplacement pour @JoinColumn dans la plupart des emplacements. La formule doit être un fragment SQL valable.
LazyCollection	Définit le statut lazy d'une collection
LazyToOne	Définit le statut lazy d'une association ToOne (c'est-à-dire OneToOne or ManyToOne).
Loader	Ecrase la méthode FIND par défaut de Hibernate.
ManyToAny	Définit une association ToMany en indiquant différent types d'entité. L'association du type d'entité correspondant se fait à travers une colonne discriminatoire de métadonnées. Ce type de mappage devrait seulement être marginal.
MapKeyType	Définit le type de clé d'une mappe persistante.
MetaValue	Représente une valeur discriminatoire associée à un type d'entité donné.
NamedNativeQueries	Prolonge les NamedNativeQueries pour maintenir les objets NamedNativeQuery de Hibernate.
NamedNativeQuery	Prolonge la NamedNativeQuery avec les caractéristiques Hibernate.
NamedQueries	Etend les NamedQueries pour maintenir les objets NamedQuery de Hibernate.
NamedQuery	Prolonge NamedQuery avec les caractéristiques Hibernate.
IdNaturel	Indique que la propriété fait partie de l'identifiant naturel de l'entité.
NotFound	Action à accomplir lorsqu'un élément est introuvable dans une association.
OnDelete	Stratégie à utiliser pour la suppression de collections, de tableaux ou de sous-classes jointes. OnDelete des tableaux secondaires n'est pas actuellement pris en charge.

Annotation	Description
OptimisticLock	Indique si une modification de la propriété annotée déclenchera une incrémentation de version d'entité. Si l'annotation n'est pas présente, la propriété est impliquée dans la stratégie de verrouillage optimiste (par défaut).
OptimisticLocking	Utilisé pour définir le style de verrouillage optimiste à appliquer à une entité. Dans une hiérarchie, seulement valide sur l'entité de root.
OrderBy	Ordonner une collection en utilisant l'ordre SQL (pas l'ordre HSL).
ParamDef	Une définition de paramètre.
Paramètre	Modèle clé/valeur
Parent	Renvoie la référence de la propriété en tant que pointeur vers le propriétaire (en général l'entité propriétaire).
Persister	Indique un persister personnalisé
Polymorphisme	Utilisé pour définir le type de polymorphisme qu'Hibernate appliquera aux hiérarchies d'entité.
Proxy	Configuration proxy et lazy d'une classe particulière.
RowId	Prend en charge les caractéristiques de mappage ROWID de Hibernate
Tri	Tri de collection (tri de niveau Java).
Source	Annotation optionnelle conjointement avec les propriétés de Version et de version Timestamp. La valeur d'annotation décide de l'endroit où timestamp est généré.
SQLDelete	Ecrase la méthode DELETE par défaut de Hibernate.
SQLDeleteAll	Ecrase la méthode DELETE ALL par défaut de Hibernate.
SQLInsert	Remplace la méthode INSERT INTO par défaut de Hibernate.

Annotation	Description
SQLUpdate	Remplace la méthode UPDATE par défaut de Hibernate.
Subselect	Mappe une entité immuable et en lecture seule vers une expression sous-sélection SQL donnée.
Synchronize	S'assure qu'auto-flush fonctionne correctement et que les requêtes concernant l'entité dérivée ne renvoient pas des données obsolètes. Principalement utilisé avec Subselect.
Table	Information complémentaire d'une table primaire ou secondaire
Tables	Annotation plurielle de Table
Cible	Définit une source explicite et évite la résolution de réflexion et de génériques.
Tuplizer	Définit un tuplizer pour une entité ou un composant.
Tuplizers	Définit un ensemble de tuplizers pour une entité ou un composant.
Type	Type Hibernate.
TypeDef	Définition d'un type Hibernate
TypeDefs	Gamme de définition de Type Hibernate
Where	Clause Where à ajouter à l'élément Entité ou entité de cible d'une collection. La clause est écrite en SQL.
WhereJoinTable	Clause Where à ajouter à la collection joindre une table. La clause est écrite en SQL.

NOTE

L'annotation « Entity » est obsolète et doit être supprimée dans les versions à venir.

Rapporter un bogue

12.4. LANGAGE DE RECHERCHE HIBERNATE

12.4.1. Langage de recherche d'Hibernate

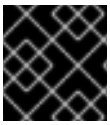
Les langages HQL (Hibernate Query Language) et JPQL (Java Persistence Query Language) sont tous les deux des langages de recherche qui se concentrent sur un modèle objet et sont similaires à SQL de par leur nature. HQL est un sur-ensemble de JPQL. Une requête HQL n'est pas toujours une requête JPQL valide, mais une requête JPQL est toujours une requête HQL valide.

HQL et JPQL sont des manières non sécurisées d'effectuer des opérations de recherche. Les requêtes de critère offrent une approche sécurisée pour les requêtes.

Rapporter un bogue

12.4.2. Déclarations HQL

HQL autorise les déclarations **SELECT**, **UPDATE**, **DELETE**, et **INSERT**. La déclaration **INSERT** de HQL n'a pas d'équivalent dans JPQL.



IMPORTANT

Veuillez être attentif au moment où une déclaration **UPDATE** ou **DELETE** est exécutée.

Tableau 12.8. Déclarations HQL

Déclaration	Description

SELECT	Le BNF pour les déclarations **SELECT** dans HQL est :
	`select_statement :: = [select_clause] from_clause [where_clause] [groupby_clause] [having_clause] [orderby_clause]`
	La déclaration **SELECT** de HQL la plus simple qui soit est sous la forme :
	`de com.acme.Cat`

UDPATE	Le BNF pour la déclaration UPDATE dans HQL est le même que dans JPQL.
DELETE	Le BNF pour les déclarations DELETE dans HQL est le même que dans JPQL.

Rapporter un bogue

12.4.3. La déclaration INSERT

HQL ajoute la possibilité de définir les déclarations **INSERT**. Il n'existe pas d'équivalent JPQL. Le BNF d'une déclaration **INSERT** HQL est :

-

```
insert_statement ::= insert_clause select_statement

insert_clause ::= INSERT INTO entity_name (attribute_list)

attribute_list ::= state_field[, state_field ]*
```

La **`attribute_list`** est analogue à la **`column specification`** dans la déclaration **`INSERT`** de SQL. Pour les entités impliquées dans un héritage mappé, seuls les attributs définis directement sur l'entité nommée peuvent être utilisés dans la **`attribute_list`**. Les propriétés de sur-classe ne sont pas autorisées et les propriétés de sous-classe sont inutiles. En d'autres termes, les déclarations **`INSERT`** sont, par nature, non-polymorphes.

AVERTISSEMENT

`select_statement` peut être n'importe quelle requête sélectionnée HQL valide, avec l'avertissement que les types de renvoi doivent correspondre aux types attendus par l'insertion. Actuellement, cela est vérifié lors de la compilation de requêtes plutôt que de permettre une vérification reléguée à la base de données. Cela pourrait causer des problèmes entre les types Hibernate qui sont *equivalent* et non pas *equal*. Par exemple, cela pourrait entraîner des problèmes avec des incompatibilités entre un attribut mappé en tant que **`org.hibernate.type.DateType`** et un attribut défini en tant que **`org.hibernate.type.TimestampType`**, bien que la base de données puisse ne pas faire la distinction ou puisse être capable de traiter la conversion.

Pour l'attribut de l'identifiant, la déclaration d'insertion vous propose deux options. Vous pouvez soit indiquer de manière spécifique la propriété de l'identifiant dans la **`attribute_list`**, dans quel cas sa valeur sera prise à partir de l'expression sélectionnée correspondante, ou l'omettre à partir de la **`attribute_list`**, dans quel cas une valeur générée sera utilisée. Cette dernière option n'est disponible que lors de l'utilisation de générateurs d'identifiants qui opèrent « dans la base de données ». Tenter d'utiliser cette option avec n'importe quel générateur de type « en mémoire » causera une exception en cours de traitement.

Pour les attributs de verrouillages optimistes, la déclaration d'insertion vous propose à nouveau deux options : vous pouvez soit indiquer l'attribut dans la **`attribute_list`**, dans quel cas sa valeur sera prise à partir des expressions sélectionnées correspondantes, ou l'omettre à partir de la **`attribute_list`** dans quel cas la **`seed value`** définie par le **`org.hibernate.type.VersionType`** correspondant sera utilisé.

Exemple 12.3. Exemple d'énoncé de requêtes INSERT

```
String hqlInsert = "insert into DelinquentAccount (id, name) select
c.id, c.name from Customer c where ...";
int createdEntities = s.createQuery( hqlInsert ).executeUpdate();
```

Rapporter un bogue

12.4.4. La clause FROM

La clause **FROM** est chargée de définir l'étendue des types de modèles objets disponibles au reste de la requête. Elle est également chargée de définir toutes les « variables d'identification » disponibles au reste de la requête.

Rapporter un bogue

12.4.5. La clause WITH

HQL définit une clause **WITH** pour qualifier les conditions de jointure. Cela est spécifique à HQL. JPQL ne définit pas cette fonctionnalité.

Exemple 12.4. Exemple de jointure de with-clause

```
select distinct c
from Customer c
    left join c.orders o
        with o.value > 5000.00
```

La grande différence est que, dans le SQL généré, les conditions de **with clause** font partie de **on clause** dans le SQL généré contrairement aux autres requêtes dans cette section où les conditions HQL/JPQL font partie de la **where clause** dans le SQL généré. La différence dans cet exemple en particulier n'est sûrement pas très importante. La **with clause** est quelques fois nécessaire pour des requêtes plus compliquées.

Les jointures explicites peuvent faire référence à des attributs d'association ou de composant/intégrés. Dans le cas d'attributs de composant/intégrés, la jointure est tout simplement logique et n'est pas en corrélation avec une jointure (SQL) physique.

Rapporter un bogue

12.4.6. Mises à jour, insertions et suppressions en bloc

Hibernate autorise l'utilisation de DML (Data Manipulation Language) pour les données d'insertions, de mises à jour et de suppressions en bloc directement dans la base de données mappée à travers par Hibernate Query Language.

AVERTISSEMENT

L'utilisation de DML pourrait enfreindre le mappage objet/relationnel et affecter l'état objet. L'état objet reste en mémoire et, en utilisant DML, l'état d'un objet en mémoire ne sera pas affecté selon l'opération effectuée sur la base de données sous-jacente. Les données en mémoire doivent être utilisées avec soin si DML est utilisé.

La pseudo-syntaxe pour les déclarations UPDATE et DELETE est : **(UPDATE | DELETE) FROM? EntityName (WHERE where_conditions)?**.

NOTE

Le mot clé **FROM** et le **WHERE Clause** sont optionnels.

Le résultat d'exécution d'une déclaration UPDATE ou DELETE est le nombre de lignes réellement affectées (mises à jour ou supprimées).

Exemple 12.5. Exemple de déclaration de mise à jour en bloc

```
Session session = sessionFactory.openSession();
Transaction tx = session.beginTransaction();

String hqlUpdate = "update Company set name = :newName where name =
:oldName";
int updatedEntities = s.createQuery( hqlUpdate )
        .setString( "newName", newName )
        .setString( "oldName", oldName )
        .executeUpdate();
tx.commit();
session.close();
```

Exemple 12.6. Exemple de déclaration de suppression en bloc

```
Session session = sessionFactory.openSession();
Transaction tx = session.beginTransaction();

String hqlDelete = "delete Company where name = :oldName";
int deletedEntities = s.createQuery( hqlDelete )
        .setString( "oldName", oldName )
        .executeUpdate();
tx.commit();
session.close();
```

La valeur **int** renvoyée par la méthode **Query.executeUpdate()** indique le nombre d'entités qui se trouvent dans la base de données et qui sont affectées par l'opération.

En interne, la base de données peut utiliser plusieurs déclarations SQL pour exécuter l'opération en réponse à une mise à jour DML ou à une demande de suppression. Cela est sans doute en raison des relations existantes entre les tableaux et les tableaux de liaisons qui ont besoin d'être mis à jour ou effacés.

Ainsi et par exemple, si on émet une déclaration de suppression (exemple ci-dessus), on risque d'avoir une délétion non seulement des **Company** (entreprises) qui contiennent **oldName** dans le tableau, mais aussi dans les tableaux de liaisons. Donc, un tableau d'entreprises (Companies) qui se trouve dans une relation BiDirectionelle ManyToMany avec un tableau d'employés (Employee) perdrait des lignes dans le tableau de liaison correspondant **Company_Employee** suite à l'exécution de l'exemple précédent.

La valeur **int deletedEntries** ci-dessus contiendra le nombre de lignes affectées par cette opération, y compris les lignes qui se trouvent dans les tableaux de liaisons.

La pseudo-syntaxe pour les déclarations INSERT est : **`INSERT INTO EntityName properties_list select_statement`**.

NOTE

Seule la forme INSERT INTO ... SELECT ... est prise en charge; et non pas la forme INSERT INTO ... VALUES ...

Exemple 12.7. Exemple de déclaration d'insertion en bloc

```
Session session = sessionFactory.openSession();
Transaction tx = session.beginTransaction();

String hqlInsert = "insert into Account (id, name) select c.id, c.name
from Customer c where ...";
int createdEntities = s.createQuery( hqlInsert )
        .executeUpdate();
tx.commit();
session.close();
```

Si vous ne fournissez pas la valeur de l'attribut **`id`** via SELECT, un ID sera généré, tant que la base de données sous-jacente prend en charge les clés générées automatiquement. La valeur de retour de cette opération d'insertion en bloc est le nombre d'entrées créées dans la base de données.

Rapporter un bogue

12.4.7. Références de collection

Les références aux associations collection-valued font en fait référence aux *valeurs* de cette collection.

Exemple 12.8. Exemple de références de collection

```
select c
from Customer c
    join c.orders o
    join o.lineItems l
    join l.product p
where o.status = 'pending'
  and p.status = 'backorder'

// alternate syntax
select c
from Customer c,
    in(c.orders) o,
    in(o.lineItems) l
    join l.product p
where o.status = 'pending'
  and p.status = 'backorder'
```

Dans cet exemple, la variable d'identification **`o`** fait référence au type de modèle de variable **`Order`** qui correspond au type d'éléments de l'association **`Customer#orders`**.

L'exemple montre une autre syntaxe d'association de collection avec **`IN`**. Les deux formes sont équivalentes. La forme choisie par une application est tout simplement une question de goût.

Rapporter un bogue

12.4.8. Expressions de chemins qualifiées

Dans le passé, il était indiqué que les associations collection-valued faisaient référence aux *valeurs* de la dite collection. Pour ce type de collection, il existe également un ensemble d'expressions de qualification explicites.

Tableau 12.9. Expressions de chemins

Expression	Description
`VALUE`	Correspond à la valeur de la collection. C'est équivalent à ne pas spécifier de qualificateur. Utile pour démontrer une intention. Valide pour n'importe quel type de référence collection-valued.
`INDEX`	Selon les règles HQL, ceci est valable pour les mappages et pour les listes qui spécifient une annotation **`javax.persistence.OrderColumn`** pour désigner la clé de Mappage ou la position de la Liste (aka la valeur OrderColumn). JPQL toutefois réserve ceci pour une utilisation dans la case Liste et ajoute la **`clé`** dans la case Mappage. Les applications intéressées par la transférabilité des fournisseurs JPA doivent tenir compte de cette distinction.
`KEY`	Uniquement valide pour le mappage. Se réfère à la clé de mappage. Si la clé elle-même est une entité, elle peut naviguer davantage.
`ENTRY`	Uniquement valide pour les mappages. Voir le tuple (combinaison d'une clé et de sa valeur) logique du Mappagev **`java.util.Map.Entry`**. **`ENTRY`** n'est valide qu'en tant que chemin final et uniquement pour la clause sélectionnée.

Exemple 12.9. Exemple de références de collection qualifiée

```
// Product.images is a Map<String,String> : key = a name, value = file
path

// select all the image file paths (the map value) for Product#123
select i
from Product p
    join p.images i
where p.id = 123

// same as above
select value(i)
from Product p
    join p.images i
```

```
where p.id = 123

// select all the image names (the map key) for Product#123
select key(i)
from Product p
    join p.images i
where p.id = 123

// select all the image names and file paths (the 'Map.Entry') for
Product#123
select entry(i)
from Product p
    join p.images i
where p.id = 123

// total the value of the initial line items for all orders for a
customer
select sum( li.amount )
from Customer c
        join c.orders o
        join o.lineItems li
where c.id = 123
  and index(li) = 1
```

Rapporter un bogue

12.4.9. Fonctions scalaires

HQL définit des fonctions standard disponibles quelle que soit la base de données sous-jacente utilisée. HQL peut également comprendre des fonctions supplémentaires définies par le Dialect ou l'application.

Rapporter un bogue

12.4.10. Fonctions standardisées HQL

Les fonctions suivantes sont disponibles en HQL quelle que soit la base de données utilisée sous-jacente.

Tableau 12.10. Fonctions standardisées HQL

Fonction	Description
BIT_LENGTH	Renvoie la longueur des données binaires.
CAST	Effectue une transcription SQL. La cible de transcription doit désigner le type de mappage Hibernate à utiliser. Voir le chapitre sur les types de données pour obtenir plus d'informations.
EXTRACT	Effectue une extraction SQL sur les valeurs Heure/Date. L'extraction soutire une partie de la valeur (l'année, par exemple). Voir les formes d'abréviation ci-dessous.

Fonction	Description
SECOND	Abréviation pour extraire une Seconde.
MINUTE	Forme d'abréviation pour extraire une Minute.
HOUR	Abréviation pour extraire une Heure.
DAY	Abréviation pour extraire un Jour.
MONTH	Abréviation pour extraire un Mois.
YEAR	Abréviation pour extraire une Année.
STR	Abréviation pour la conversion d'une valeur en caractère.

Les développeurs d'applications peuvent également fournir leur propre ensemble de fonctions. Habituellement, cela représente des fonctions SQL personnalisées ou des alias pour les extraits de code SQL. Ces déclarations de fonction sont effectuées au moyen de la méthode **`addSqlFunction`** de **`org.hibernate.cfg.Configuration`**

Rapporter un bogue

12.4.11. Opération de concaténation

HQL définit un opérateur de concaténation en plus de sa prise en charge de la fonction (**`CONCAT`**). N'est pas défini par JPQL, donc les applications portables doivent éviter de l'utiliser. L'opérateur de concaténation vient d'un opérateur de concaténation SQL - **`||`**.

Exemple 12.10. Exemple d'opération de concaténation

```
select 'Mr. ' || c.name.first || ' ' || c.name.last
from Customer c
where c.gender = Gender.MALE
```

Rapporter un bogue

12.4.12. Instanciation dynamique

Il y a un type d'expression particulier qui n'est valide que dans la clause Select. Hibernate nomme cela une "dynamic instanciation". JPQL supporte cette fonctionnalité et nomme cela une "expression de constructeur".

Exemple 12.11. Exemple d'instanciation dynamique - Constructeur

```
select new Family( mother, mate, offspr )
from DomesticCat as mother
    join mother.mate as mate
```

```
    left join mother.kittens as offspr
```

Au lieu de s'affairer avec l'Object[], nous englobons ici les valeurs dans un objet java de type-safe, qui sera retourné en tant que résultat de la demande. La référence de la classe doit être totalement qualifiée et doit avoir un constructeur correspondant.

La classe doit ici être mappée. Si elle représente une entité, les instances résultantes devront être renvoyées dans un état NEW (non géré!).

C'est la partie que JPQL prend en charge également. HQL prend en charge des fonctionnalités en "dynamic instantiation" supplémentaires. Tout d'abord, la recherche peut spécifier de renvoyer une Liste à la place d'un Object[] pour les résultats scalaires :

Exemple 12.12. Exemple d'instanciation dynamique - Liste

```
select new list(mother, offspr, mate.name)
from DomesticCat as mother
    inner join mother.mate as mate
    left outer join mother.kittens as offspr
```

Les résultats de cette recherche correspondront à List<List> et non pas à List<Object[]>

HQL supporte également l'absorption de résultats scalaires dans une Mappe.

Exemple 12.13. Exemple d'instanciation dynamique - Mappe

```
select new map( mother as mother, offspr as offspr, mate as mate )
from DomesticCat as mother
    inner join mother.mate as mate
    left outer join mother.kittens as offspr

select new map( max(c.bodyWeight) as max, min(c.bodyWeight) as min,
count(*) as n )
from Cat cxt"/>
```

Les résultats de cette recherche correspondront à List<Map<String,Object>> et non pas à List<Object[]>. Les clés de la mappe sont définies par les alias donnés aux expressions Select.

Rapporter un bogue

12.4.13. Prédicats HQL

Les prédicats sont à la base des expressions de recherche, la clause «where», la clause «having». Ce sont des expressions qui résolvent une valeur de vérité, comme **TRUE** ou **FALSE**, bien que les comparaison booléènnes comportant des NULL résolvent les **UNKNOWN**.

Prédicats HQL

Prédicat Nullness

Cherchez une valeur nulle et sans effet. Peut s'appliquer à des références d'attributs de base, à des références d'entité et à des paramètres. De plus, HQL permet une application à des types de composants/intégrables.

Exemple 12.14. Exemples de vérification de Nullness

```
// select everyone with an associated address
select p
from Person p
where p.address is not null

// select everyone without an associated address
select p
from Person p
where p.address is null
```

Prédicat Like

Procède à une comparaison d'équivalence entre des valeurs de chaîne. La syntaxe est la suivante :

```
like_expression ::=
   string_expression
   [NOT] LIKE pattern_value
   [ESCAPE escape_character]
```

Les sémantiques suivent l'expression Like SQL. La **`pattern_value`** est un modèle qui tente de correspondre à l'expression de chaîne **`string_expression`**. Tout comme pour l'expresion «like» SQL, le modèle ou **`pattern_value`** peut utiliser "_" et "%" comme caractères génériques. Les significations sont les mêmes. "_" correspond à n'importe quel caractère unique. "%" correspond à un certain nombre de caractères.

Le caractère **`escape`** optionnel est utilisé pour spécifier un caractère d'échappement en combinaison à ce qui est impliqué par "_" ou "%" dans le modèle ou **`pattern_value`**. Ceci est utile quand on doit chercher des modèles contenant des "_" ou des "%".

Exemple 12.15. Exemples de prédicats «Like»

```
select p
from Person p
where p.name like '%Schmidt'

select p
from Person p
where p.name not like 'Jingleheimmer%'

// find any with name starting with "sp_"
select sp
from StoredProcedureMetadata sp
where sp.name like 'sp|_%' escape '|'
```

Prédicat Between

Ressemble à l'expression SQL **BETWEEN**. Détermine qu'une valeur est entre 2 autres valeurs. Tous les opérandes doivent avoir des types comparables.

Exemple 12.16. Exemples de prédicats «Between»

```
select p
from Customer c
    join c.paymentHistory p
where c.id = 123
  and index(p) between 0 and 9

select c
from Customer c
where c.president.dateOfBirth
        between {d '1945-01-01'}
            and {d '1965-01-01'}

select o
from Order o
where o.total between 500 and 5000

select p
from Person p
where p.name between 'A' and 'E'
```

Rapporter un bogue

12.4.14. Comparaisons relationnelles

Les comparaison impliquent un des opérateurs de comparaison - =, >, >=, <, <=, <>]>. HQL also defines <![CDATA[!= comme opérateur de comparaison synonyme de <>. Les opérandes doivent être du même type.

Exemple 12.17. Exemples de comparaisons relationnelles

```
// numeric comparison
select c
from Customer c
where c.chiefExecutive.age < 30

// string comparison
select c
from Customer c
where c.name = 'Acme'

// datetime comparison
select c
from Customer c
where c.inceptionDate < {d '2000-01-01'}

// enum comparison
select c
from Customer c
```

```
where c.chiefExecutive.gender = com.acme.Gender.MALE

// boolean comparison
select c
from Customer c
where c.sendEmail = true

// entity type comparison
select p
from Payment p
where type(p) = WireTransferPayment

// entity value comparison
select c
from Customer c
where c.chiefExecutive = c.chiefTechnologist
```

Les comparaison peuvent également impliquer des qualificateurs de sous-requêtes - **ALL**, **ANY**, **SOME**. **SOME** et **ANY** sont synonymes.

Le qualificateur **ALL** passe à true si la comparaison est true pour toutes les valeurs qui se trouvent dans le résultat de la sous-requête. Passe à false si le résultat de la sous-requête est vide.

Exemple 12.18. Exemple de qualificateur de comparaison de sous-requête ALL

```
// select all players that scored at least 3 points
// in every game.
select p
from Player p
where 3 > all (
   select spg.points
   from StatsPerGame spg
   where spg.player = p
)
```

Le qualificateur **ANY**/**SOME** passe à true si la comparaison est true pour toutes les valeurs qui se trouvent dans le résultat de la sous-requête. Passe à false si le résultat de la sous-requête est vide.

Rapporter un bogue

12.4.15. Prédicat IN

Le prédicat **IN** effectue une vérification pour savoir si une valeur donnée fait partie de la liste des valeurs. Sa syntaxe est la suivante :

```
in_expression ::= single_valued_expression
            [NOT] IN single_valued_list

single_valued_list ::= constructor_expression |
            (subquery) |
```

```
            collection_valued_input_parameter

constructor_expression ::= (expression[, expression]*)
```

Les types d'expression **`single_valued_expression`** et des valeurs individuelles qui se trouvent dans la liste **`single_valued_list`** doivent être consistantes. JPQL limite les types valides ici à des chaînes, des valeurs numériques, date, heure, heure/date, et enum. Dans JPQL, **`single_valued_expression`** ne peut se référer qu'à des :

- "state fields", son terme pour attributs simples. Cela exclut en particulier les attributs composants/intégrés et associations.
- expression de type d'entité.

Dans HQL, **`single_valued_expression`** peut désigner un bien plus large éventail de types d'expression. Les association de valeurs simples sont autorisées, ainsi que les attributs composant/intégré, bien que cette fonctionnalité dépend du niveau de soutien pour le tuple ou pour la «syntaxe de constructeur de valeur de ligne» dans la base de données sous-jacente. En outre, les HQL ne limitent le type de valeur en aucune façon, bien que les développeurs d'applications doivent être conscients que les différents types peuvent encourir un soutien limité basé sur le fournisseur de base de données sous-jacent. C'est en grande partie la raison des limitations JPQL

La liste des valeurs peut venir d'un certain nombre de sources. Dans **`constructor_expression`** et **`collection_valued_input_parameter`**, la liste des valeurs ne doit pas rester vide; elle doit contenir une valeur au moins.

Exemple 12.19. Exemples de prédicat

```
select p
from Payment p
where type(p) in (CreditCardPayment, WireTransferPayment)

select c
from Customer c
where c.hqAddress.state in ('TX', 'OK', 'LA', 'NM')

select c
from Customer c
where c.hqAddress.state in ?

select c
from Customer c
where c.hqAddress.state in (
    select dm.state
    from DeliveryMetadata dm
    where dm.salesTax is not null
)

// Not JPQL compliant!
select c
from Customer c
where c.name in (
    ('John','Doe'),
    ('Jane','Doe')
)
```

```
// Not JPQL compliant!
select c
from Customer c
where c.chiefExecutive in (
    select p
    from Person p
    where ...
)
```

Rapporter un bogue

12.4.16. SQL Ordering

Les résultats de la demande peuvent également être ordonnancés. La clause **ORDER BY** est utilisée pour spécifier les valeurs sélectionnées à utiliser pour ordonnancer le résultat. Les types d'expressions considérées comme valides pour la clause order-by sont les suivantes :

- state fields
- attributs composant/intégré
- expressions scalaires comme les opérations arithmétiques, les fonctions, etc.
- variable d'identification déclarée dans la clause Select pour n'importe quel type d'expression antérieure.

HQL ne prescrit pas que toutes les valeurs référencées dans la clause orde-by doivent être nommées dans la clause Select, mais ceci est requis dans JPQL. Les applications désireuses de portabilité de base de données doivent noter que toutes les bases de données ne prennent pas en charge les valeurs de références dans la clause order-by qui ne sont pas référencées dans la clause select

Les expressions individuelles d'order-by peuvent être soit qualifiées par **ASC** (ascendant) ou **DESC** (descendant) pour indiquer la direction d'ordonnancement désirée.

Exemple 12.20. Exemple Order-by

```
// legal because p.name is implicitly part of p
select p
from Person p
order by p.name

select c.id, sum( o.total ) as t
from Order o
    inner join o.customer c
group by c.id
order by t
```

Rapporter un bogue

12.5. SERVICES HIBERNATE

12.5.1. Les services Hibernate

Les services sont des classes qui procurent à Hibernate des implémentations connectables de divers types de fonctionnalité. Plus précisément, ce sont des implémentations de certaines interfaces de contrats de service. L'interface est connue comme rôle de service; la classe d'implémentation est connue comme implémentation du service. De manière générale, les utilisateurs peuvent se raccorder sur d'autres implémentations de rôles de service standard (par substitution); ils peuvent également définir des services supplémentaires au-delà de l'ensemble de base des rôles de service (extension).

Rapporter un bogue

12.5.2. Contrats de service

La nécessité de base d'un service est d'implémenter l'interface du marqueur **`org.hibernate.service.Service`**. Hibernate l'utilise en interne pour des raisons de sécurité de base.

En option, le service peut également implémenter les interfaces **`org.hibernate.service.spi.Startable`** et **`org.hibernate.service.spi.Stoppable`** pour recevoir des notifications de démarrage ou d'arrêt. Il existe un autre service **`org.hibernate.service.spi.Manageable`** qui indique que le service est gérable dans le JMX dans la mesure où l'intégration JMX est activée.

Rapporter un bogue

12.5.3. Types de dépendances de service

Les services peuvent déclarer des dépendances sur d'autres services par l'une des approches suivantes :

`@org.hibernate.service.spi.InjectService`

Toute méthode de classe d'implémentation de service qui accepte un seul paramètre et qui est annotée par **`@InjectService`** est considérée comme réclamant une injection de la part d'un autre service.

Par défaut, le type du paramètre de méthode doit correspondre au rôle de service à injecter. Si le type de paramètre est différent du rôle de service, l'attribut **`serviceRole`** du **`InjectService`** devra être utilisé pour nommer explicitement le rôle.

Les services injectés sont requis par défaut, c'est à dire que le démarrage échouera si un service dépendant nommé est manquant. Si le service injecté est optionnel, l'attribut **`requis`** d'**`InjectService`** devra être déclaré **`false`** (la valeur par défaut est **`true`**).

`org.hibernate.service.spi.ServiceRegistryAwareService`

La seconde approche est une approche pull pour laquelle le service implémente l'interface de service optionnelle **`org.hibernate.service.spi.ServiceRegistryAwareService`** qui déclare une méthode **`injectServices`** unique.

En cours de démarrage, Hibernate injectera le **`org.hibernate.service.ServiceRegistry`** lui-même dans les services qui implémentent cette interface. Le service pourra alors utiliser la référence **`ServiceRegistry`** pour trouver les services additionnels dont il pourrait avoir besoin.

Rapporter un bogue

12.5.4. Le ServiceRegistry

12.5.4.1. Le ServiceRegistry

L'API du service central, mis à part les services eux-même, correspond à l'interface **`org.hibernate.service.ServiceRegistry`**. Le but principal d'un registre de service est de contenir, de gérer et de donner accès aux services.

Les registres de services sont hiérarchiques. Les services d'un registre peuvent dépendre ou utiliser des services de ce même registre, ou bien de registres parents.

Utiliser **`org.hibernate.service.ServiceRegistryBuilder`** pour créer une instance **`org.hibernate.service.ServiceRegistry`**.

Exemple 12.21. Utiliser ServiceRegistryBuilder pour créer un ServiceRegistry

```
ServiceRegistryBuilder registryBuilder = new ServiceRegistryBuilder(
bootstrapServiceRegistry );
    ServiceRegistry serviceRegistry =
registryBuilder.buildServiceRegistry();
```

Rapporter un bogue

12.5.5. Services personnalisés

12.5.5.1. Services personnalisés

Une fois qu'un **`org.hibernate.service.ServiceRegistry`** est créé, il est considéré comme immuable. Les services eux-mêmes pourraient accepter une reconfiguration, mais ici, l'immuabilité signifie un ajout/remplacement des services. Et donc, un autre rôle assuré par **`org.hibernate.service.ServiceRegistryBuilder`** est de permettre de peaufiner des services qui figureront dans le **`org.hibernate.service.ServiceRegistry`** qu'il génère.

Il y a 2 façons d'informer le **`org.hibernate.service.ServiceRegistryBuilder`** à propos de services personnalisés.

- Implémenter une classe **`org.hibernate.service.spi.BasicServiceInitiator`** pour contrôler la construction sur demande de la classe du service et y ajouter le **`org.hibernate.service.ServiceRegistryBuilder`** via sa méthode **`addInitiator`**.
- Il vous suffit d'instancier la classe de service et de l'ajouter au **`org.hibernate.service.ServiceRegistryBuilder`** via sa méthode **`addService`**.

L'ajout d'une approche de service ou l'ajout d'une approche d'initiateur sont valables pour l'extension d'un registre (ajout de nouveaux rôles de service) et pour la substitution de services (remplacer les implémentations de service).

Exemple 12.22. Utiliser ServiceRegistryBuilder pour remplacer un service existant par un service personnalisé

```
   ServiceRegistryBuilder registryBuilder = new ServiceRegistryBuilder(
bootstrapServiceRegistry );
```

```
   registryBuilder.addService( JdbcServices.class, new FakeJdbcService()
);
   ServiceRegistry serviceRegistry =
registryBuilder.buildServiceRegistry();


   public class FakeJdbcService implements JdbcServices{

       @Override
       public ConnectionProvider getConnectionProvider() {
           return null;
       }

       @Override
       public Dialect getDialect() {
           return null;
       }

       @Override
       public SqlStatementLogger getSqlStatementLogger() {
           return null;
       }

       @Override
       public SqlExceptionHelper getSqlExceptionHelper() {
           return null;
       }

       @Override
       public ExtractedDatabaseMetaData getExtractedMetaDataSupport() {
           return null;
       }

       @Override
       public LobCreator getLobCreator(LobCreationContext
lobCreationContext) {
           return null;
       }

       @Override
       public ResultSetWrapper getResultSetWrapper() {
           return null;
       }

       @Override
       public JdbcEnvironment getJdbcEnvironment() {
           return null;
       }
   }
```

Rapporter un bogue

12.5.6. Le registre Bootstrap

12.5.6.1. Le registre Boot-strap

Le registre de bootstrap contient les services qui doivent absolument être disponibles pour que tout fonctionne. Le service principal ici est le **`ClassLoaderService`** qui en est un parfait exemple. Même la résolution des fichiers de configuration doit avoir accès aux services de chargement de classes (pour la recherche de ressources). C'est le registre racine (sans parent) en conditions normales d'utilisation.

Les instances de boot-strap sont créées par la classe **`org.hibernate.service.BootstrapServiceRegistryBuilder`**.

Rapporter un bogue

12.5.6.2. Utilisation de BootstrapServiceRegistryBuilder

Exemple 12.23. Utilisation de BootstrapServiceRegistryBuilder

```
BootstrapServiceRegistry bootstrapServiceRegistry = new
BootstrapServiceRegistryBuilder()
        // pass in org.hibernate.integrator.spi.Integrator instances
which are not
        // auto-discovered (for whatever reason) but which should be
included
        .with( anExplicitIntegrator )
        // pass in a class-loader Hibernate should use to load
application classes
        .with( anExplicitClassLoaderForApplicationClasses )
        // pass in a class-loader Hibernate should use to load
resources
        .with( anExplicitClassLoaderForResources )
        // see BootstrapServiceRegistryBuilder for rest of available
methods
        ...
        // finally, build the bootstrap registry with all the above
options
        .build();
```

Rapporter un bogue

12.5.6.3. Les services BootstrapRegistry

`org.hibernate.service.classloading.spi.ClassLoaderService`

Hibernate a besoin d'interagir avec les chargeurs de classes. Cependant, la manière dont Hibernate (ou n'importe quelle bibliothèque) doit interagir avec les chargeurs de classe parents varie selon l'environnement d'exécution qui héberge l'application. Les serveurs d'applications, les conteneurs OSGi et autres systèmes de chargement de classes modulaires ont des exigences très spécifiques de chargement de classes. Ce service fournit à Hibernate une abstraction de cette complexité environnementale. Et tout aussi important, il le fait par l'intermédiaire d'un composant simple et interchangeable.

En terme d'interaction avec ClassLoader, Hibernate a besoin des fonctionnalités suivantes :

- la capacité de localiser des classes d'application

- la capacité de localiser des classes d'intégration
- la capacité de localiser des ressources (fichiers de propriétés, fichiers xml, etc.)
- la capacité à charger **`java.util.ServiceLoader`**

NOTE

Actuellement, la capacité à charger des classes d'application et la capacité à charger des classes d'intégration sont combinées en une même capacité de "chargement de classe" pour le service. Cela risque de changer dans les versions à venir.

`org.hibernate.integrator.spi.IntegratorService`

Les applications, modules et autres avaient tous besoin de s'intégrer dans Hibernate moyennant une contrepartie, généralement l'application, pour coordonner l'inscription les pièces de chaque intégration nécessaire au nom de chaque intégrateur. Ce service vise à permettre à ces intégrateurs d'être découverts et à les faire s'intégrer dans Hibernate.

Ce service se focalise sur l'aspect discovery. Il utiliser la capacité Java standard **`java.util.ServiceLoader`** fournie par **`org.hibernate.service.classloading.spi.ClassLoaderService`** pour découvrir les implémentations du contrat **`org.hibernate.integrator.spi.Integrator`**.

Les intégrateurs se contentaient de définir un fichier nommé **`/META-INF/services/org.hibernate.integrator.spi.Integrator`** et de le rendre disponible sur le chemin de classe. **`java.util.ServiceLoader`** couvre le format de ce fichier en détails, et surtout dresse une liste des classes par FQN qui implémentent le **`org.hibernate.integrator.spi.Integrator`** un par ligne.

Rapporter un bogue

12.5.7. Le registre SessionFactory

12.5.7.1. Le registre SessionFactory

Malgré qu'il soit conseillé de traiter les instances de tous les types de registres comme ciblant une **`org.hibernate.SessionFactory`** donnée, les instances de service de ce groupe appartiennent explicitement à une **`org.hibernate.SessionFactory`** unique.

La différence est une question de savoir quand elles doivent être initiées. En général, elles doivent avoir accès à la **`org.hibernate.SessionFactory`** pour pouvoir être initiées. Ce registre spécial est **`org.hibernate.service.spi.SessionFactoryServiceRegistry`**

Rapporter un bogue

12.5.7.2. Services SessionFactory

`org.hibernate.event.service.spi.EventListenerRegistry`

Description

Service pour gérer les listeners d'événements

Initiateur

`org.hibernate.event.service.internal.EventListenerServiceInitiator`

Implémentations

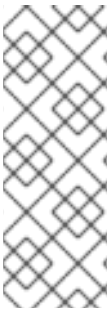
`org.hibernate.event.service.internal.EventListenerRegistryImpl`

Rapporter un bogue

12.5.8. Intégrateurs

12.5.8.1. Intégrateurs

Le **`org.hibernate.integrator.spi.Integrator`** a pour but de procurer un simple moyen de permettre aux développeurs de raccrocher un processus de création de SessionFactory en état de fonctionnement. L'interface **`org.hibernate.integrator.spi.Integrator`** définit 2 méthodes intéressantes : **`integrate`** permet de se joindre à un processus de création; **`disintegrate`** nous permet de raccrocher la fermeture de la SessionFactory.



NOTE

Il y a une 3ème méthode définie dans **`org.hibernate.integrator.spi.Integrator`**, une forme surchargée de **`integrate`** qui accepte un **`org.hibernate.metamodel.source.MetadataImplementor`** à la place d'une **`org.hibernate.cfg.Configuration`**. Cette forme a pour dessein d'être utilisée avec le nouveau code de méta modèle qui devra est opérationnel dans 5.0.

En plus de l'approche discovery fournie dans IntegratorService, les applications peuvent enregistrer des implémentations d'integrator manuellement quand elles créent le BootstrapServiceRegistry.

Rapporter un bogue

12.5.8.2. Cas d'utilisation d'Integrator

Les cas principaux d'intégration d'un **`org.hibernate.integrator.spi.Integrator`** actuellement consistent à enregistrer des listeners d'événements et de fournir des services (voir **`org.hibernate.integrator.spi.ServiceContributingIntegrator`**). Dans 5.0, nous espérons étendre ceci afin de permettre de modifier les méta modèles qui décrivent le mappage entre les modèles relationnels et les objets.

Exemple 12.24. Enregistrement des listeners d'événements

```
public class MyIntegrator implements
org.hibernate.integrator.spi.Integrator {

    public void integrate(
            Configuration configuration,
            SessionFactoryImplementor sessionFactory,
            SessionFactoryServiceRegistry serviceRegistry) {
        // As you might expect, an EventListenerRegistry is the thing
with which event listeners are registered  It is a
```

```
        // service so we look it up using the service registry
        final EventListenerRegistry eventListenerRegistry =
serviceRegistry.getService( EventListenerRegistry.class );

        // If you wish to have custom determination and handling of
"duplicate" listeners, you would have to add an
        // implementation of the
org.hibernate.event.service.spi.DuplicationStrategy contract like this
        eventListenerRegistry.addDuplicationStrategy(
myDuplicationStrategy );

        // EventListenerRegistry defines 3 ways to register listeners:
        //     1) This form overrides any existing registrations with
        eventListenerRegistry.setListeners( EventType.AUTO_FLUSH,
myCompleteSetOfListeners );
        //     2) This form adds the specified listener(s) to the
beginning of the listener chain
        eventListenerRegistry.prependListeners( EventType.AUTO_FLUSH,
myListenersToBeCalledFirst );
  //     3) This form adds the specified listener(s) to the end of the
listener chain
        eventListenerRegistry.appendListeners( EventType.AUTO_FLUSH,
myListenersToBeCalledLast );
    }
}
```

Rapporter un bogue

12.6. VALIDATION D'UN BEAN

12.6.1. Bean Validation

Bean Validation, ou JavaBeans Validation, est un modèle de validation des données dans des objets Java. Le modèle utilise les contraintes d'annotations prédéfinies et personnalisées afin d'assurer l'intégrité des données d'application. La spécification est documentée ici http://jcp.org/en/jsr/detail?id=303.

Hibernate Validator est l'implémentation JBoss EAP 6 de Bean Validation. C'est également l'implémentation de référence du JSR.

JBoss EAP 6 est conforme à 100% avec JSR 303 - Bean Validation. Hibernate Validator fournit également quelques fonctionnalités supplémentaires à la spécification.

Pour débuter avec Bean Validation, consulter **`bean-validation`** quickstart example : Section 1.4.1.1, « Accès aux Quickstarts ».

Rapporter un bogue

12.6.2. Hibernate Validator

Hibernate Validator est l'implémentation de référence de JSR 303 - Bean Validation.

Bean Validation donne aux utilisateurs un modèle de validation des données d'objets Java. Pour plus d'informations, voir Section 12.6.1, « Bean Validation » et Section 12.6.3.1, « Les contraintes de validation ».

Rapporter un bogue

12.6.3. Contraintes de validation

12.6.3.1. Les contraintes de validation

Les contraintes de validation sont des règles qui s'appliquent à un élément Java, comme un champ, une propriété ou un bean. Une contrainte aura sans doute un groupe d'attributs utilisés pour définir ses limites. Il existe des contraintes prédéterminées, et des contraintes personnalisées peuvent être créées. Chaque contrainte est exprimée sous forme d'annotation.

Les contraintes de validation intégrées d'Hibernate Validator sont listées ici : Section 12.6.3.4, « Contraintes d'Hibernate Validator »

Pour plus d'informations, voir Section 12.6.2, « Hibernate Validator » et Section 12.6.1, « Bean Validation ».

Rapporter un bogue

12.6.3.2. Créer une annotation de contrainte dans Red Hat JBoss Developer Studio

Résumé

Cette tâche couvre le processus de création d'une annotation de contrainte dans Red Hat JBoss Developer Studio, à utiliser dans une application Java.

Conditions préalables

- Section 1.3.1.4, « Démarrer Red Hat JBoss Developer Studio »

Procédure 12.5. Créer une annotation de contrainte

1. Ouvrir un projet Java dans Red Hat JBoss Developer Studio
2. **Créer un ensemble de données**
   Une annotation de contrainte requiert un ensemble de données qui définissent les valeurs qui conviennent.
   a. Cliquer à droite sur le dossier racine du projet dans la section **`Project Explorer`**.
   b. Sélectionner **New** → **Enum**.
   c. Configurer les éléments suivants :
      - **`Package:`**
      - **`Name:`**
   d. Cliquer sur le bouton **`Add...`** pour ajouter des interfaces selon les besoins.
   e. Cliquer sur **`Finish`** pour créer le fichier.

f. Ajouter un ensemble de valeurs à l'ensemble des données et cliquer sur **Save**.

Exemple 12.25. Exemple d'ensemble de données

```
package com.example;

public enum CaseMode {
    UPPER,
    LOWER;
}
```

3. **Créer le fichier Annotation**
   Créer une nouvelle classe Java. Pour les détails, consulter Section 12.6.3.3, « Créer une nouvelle classe Java dans Red Hat JBoss Developer Studio ».

4. Configurer l'annotation de la contrainte et cliquer sur **Save**.

Exemple 12.26. Exemple de fichier d'annotation de contrainte

```
package com.mycompany;

import static java.lang.annotation.ElementType.*;
import static java.lang.annotation.RetentionPolicy.*;

import java.lang.annotation.Documented;
import java.lang.annotation.Retention;
import java.lang.annotation.Target;

import javax.validation.Constraint;
import javax.validation.Payload;

@Target( { METHOD, FIELD, ANNOTATION_TYPE })
@Retention(RUNTIME)
@Constraint(validatedBy = CheckCaseValidator.class)
@Documented
public @interface CheckCase {

    String message() default "
{com.mycompany.constraints.checkcase}";

    Class<?>[] groups() default {};

    Class<? extends Payload>[] payload() default {};

    CaseMode value();

}
```

Résultat

Une annotation de contrainte personnalisée possédant un ensemble de valeurs possibles a été créée, prête à l'utilisation dans le projet Java.

Rapporter un bogue

12.6.3.3. Créer une nouvelle classe Java dans Red Hat JBoss Developer Studio

Conditions préalables

- Section 1.3.1.4, « Démarrer Red Hat JBoss Developer Studio »

Résumé

Ce sujet couvre le processus de création d'une classe Java dans un projet Java qui utilise Red Hat JBoss Developer Studio.

Procédure 12.6. Créer une nouvelle classe Java

1. Cliquer à droite sur le dossier racine du projet dans le panneau **`Project Explorer`**.
2. Sélectionner **Nouveau** → **Classe**.
3. Configurer les éléments suivants :
   - **`Package:`**
   - **`Name:`**
4. **Option : ajouter une interface**
   a. Cliquer sur **`Ajouter...`**
   b. Rechercher le nom de l'interface
   c. Sélectionner l'interface qui convient.
   d. Répéter les étapes 2 et 3 pour chaque interface requise.
   e. Cliquer sur **`Ajouter`**.
5. Cliquer sur **`Terminer`** pour créer le fichier.

Résultat

Une nouvelle Classe Java a maintenant été créée dans le projet, prête à la configuration.

Rapporter un bogue

12.6.3.4. Contraintes d'Hibernate Validator

Tableau 12.11. Contraintes intégrées

Annotation	+	Vérification du runtime	Impact des métadonnées Hibernate

Annotation	+	Vérification du runtime	Impact des métadonnées Hibernate
@Length(min=, max=)	propriété (String)	Vérifier si la longueur de la chaîne de caractères correspond à la plage.	La colonne sera définie à la longueur maximum.
@Max(value=)	propriété (numérique ou représentation par chaîne d'un numérique)	Vérifiez si la valeur est inférieure ou égale à la valeur maximum.	Ajouter un contrôle sur la colonne.
@Min(value=)	propriété (numérique ou représentation par chaîne d'un numérique)	Vérifier si la valeur est supérieure ou égale à la valeur minimale.	Ajouter un contrôle sur la colonne.
@NotNull	propriété	Vérifier si la valeur est non nulle.	Colonne(s) non nulles.
@NotEmpty	propriété	Vérifier si la chaîne est non nulle ou vide. Vérifier si la connexion est non nulle ou vide.	Colonne(s) non nulles (pour Chaîne).
@Past	propriété (date ou calendrier)	Vérifier si la date est dans le passé.	Ajouter un contrôle sur la colonne.
@Future	propriété (date ou calendrier)	Vérifier si la date est dans le futur.	Aucun(e).
@Pattern(regex="regexp", flag=) or @Patterns({@Pattern(...)})	propriété (String)	Vérifier si la propriété correspond à une expression standard avec un indicateur de correspondance (voir **`java.util.regex.Pattern`**).	Aucun(e).
@Range(min=, max=)	propriété (numérique ou représentation par chaîne d'un numérique)	Vérifier si la valeur est comprise entre la valeur minimum et la valeur maximum (comprise).	Ajouter un contrôle sur la colonne.
@Size(min=, max=)	propriété (tableau, collection, mappe)	Vérifier si la taille de l'élément est comprise entre la valeur minimum et la valeur maximum (comprise).	Aucun(e).

Annotation	+	Vérification du runtime	Impact des métadonnées Hibernate
@AssertFalse	propriété	Vérifier si la méthode évalue à false (utile pour les contraintes en code et non pas sous forme d'annotations).	Aucun(e).
@AssertTrue	propriété	Vérifier si la méthode évalue à true (utile pour les contraintes en code et non pas sous forme d'annotations).	Aucun(e).
@Valid	propriété (objet)	Procède à une validation récursive de l'objet associé. Si l'objet correspond à une collection ou à un tableau, les éléments seront validés de façon récursive. Si l'objet est une mappe, les éléments correspondant à la valeur seront validés de façon récursive.	Aucun(e).
@Email	propriété (String)	Vérifier si la chaîne de caractères est bien conforme à la spécification de l'adresse email.	Aucun(e).
@CreditCardNumber	propriété (String)	Vérifier si la chaîne de caractères correspond à un numéro de carte de crédit formaté (dérivatif de l'algorithme Luhn).	Aucun(e).
@Digits(integerDigits=1)	propriété (numérique ou représentation par chaîne d'un numérique)	Vérifier si la propriété est un numéro qui comprend un maximum de **`integerDigits`** chiffres et **`fractionalDigits`** chiffres sous forme de fraction.	Définit la précision et l'échelle de la colonne.

Annotation	+	Vérification du runtime	Impact des métadonnées Hibernate
@EAN	propriété (String)	Vérifie si la chaîne correspond à un code UPC-A ou EAN formaté comme il faut.	Aucun(e).

Rapporter un bogue

12.6.4. Configuration

12.6.4.1. Exemple de fichier de configuration de validation

Exemple 12.27. validation.xml

```
<validation-config
xmlns="http://jboss.org/xml/ns/javax/validation/configuration"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://jboss.org/xml/ns/javax/validation/configurati
on">

    <default-provider>
        org.hibernate.validator.HibernateValidator
    </default-provider>
    <message-interpolator>

org.hibernate.validator.messageinterpolation.ResourceBundleMessageInterp
olator
    </message-interpolator>
    <constraint-validator-factory>
        org.hibernate.validator.engine.ConstraintValidatorFactoryImpl
    </constraint-validator-factory>

    <constraint-mapping>
        /constraints-example.xml
    </constraint-mapping>

    <property name="prop1">value1</property>
    <property name="prop2">value2</property>
</validation-config>
```

Rapporter un bogue

12.7. ENVERS

12.7.1. Hibernate Envers

Hibernate Envers est un système d'audit et de gestion des versions, offrant un moyen de suivre les changements historiques à des classes persistantes de JBoss EAP 6. Des tableaux de vérification sont créées pour les entités annotées avec **`@Audited`**, et ils stockent l'historique des modifications apportées à l'entité. Les données peuvent ensuite être récupérées et interrogées.

Envers permet aux développeurs de :

- vérifier tous les mappages définis par la spécification JPA,
- vérifier tous les mappages qui étendent la spécification JPA,
- vérifier toules les entités mappées ou qui utilisent l'API Hibernate natif
- journaliser des données pour chaque révision par l'intermédiaire d'une entité de révision, et
- interroger les données historiques.

Rapporter un bogue

12.7.2. Audit de classes persistantes

L'auditing de classes persistantes est fait par l'intermédiaire d'Hibernate Envers dans JBoss EAP 6 et de l'annotation **`@Audited`** . Quand l'annotation s'applique à une classe, un tableau est créé, et ce tableau stocke l'historique de révision de l'entité.

À chaque fois qu'il y a un changement de classe, une entrée est ajoutée au tableau d'audit. L'entrée contient les changements apportés à la classe, et reçoit un numéro de révision. Cela signifie que les changements peuvent être annulés, ou que les révisions précédentes peuvent être consultées.

Rapporter un bogue

12.7.3. Stratégies d'auditing

12.7.3.1. Stratégies d'auditing

Les stratégies d'auditing définissent comment l'information d'audit est persistée, interrogée et stockée. Il existe actuellement deux stratégies d'audit disponibles dans Hibernate Envers :

Stratégie d'audit par défaut

Cette stratégie persiste les données d'audit avec la révision de départ. Pour chaque ligne insérée, mise à jour ou effacée dans un tableau audité, une ou plusieurs lignes seront insérées dans les tableaux d'audit, avec la révision de départ de sa validité.

Les lignes des tableaux d'audit ne sont jamais mises à jour suite à une insertion. Les demandes d'informations d'audit utilisent des sous-requêtes pour sélectionner les lignes qui s'appliquent aux tableaux d'audit, et qui sont lentes et difficiles à indexer.

Stratégie d'audit de validité

Cette stratégie stocke la révision de départ, ainsi que la révision de fin de l'information d'audit. Pour chaque rangée insérée, mise à jour ou effacée dans un tableau audité, une ou plusieurs rangées seront insérées dans les tableaux d'audit, avec la révision de départ de sa validité.

En même temps, le champ de révision de fin des lignes d'audit précédentes (si disponible) est défini pour cette révision. Les demandes d'informations d'audit peuvent ensuite utiliser *entre révision de départ ou de fin*, à la place des sous-requêtes. Cela signifie que persister les informations d'audit est

un peu plus lent à cause des mises à jour supplémentaires, mais la récupération d'informations d'audit est beaucoup plus rapide.

Cela peut également être amélioré en ajoutant des indices supplémentaires.

Pour obtenir plus d'informations sur l'auditing, consulter Section 12.7.2, « Audit de classes persistantes ». Pour définir la stratégie d'auditing de l'application, voir : Section 12.7.3.2, « Définir la stratégie d'auditing ».

Rapporter un bogue

12.7.3.2. Définir la stratégie d'auditing

Résumé

Il y a deux stratégies d'auditing prises en charge par JBoss EAP 6 : la stratégie d'audit de validité ou la stratégie d'audit par défaut. Cette tâche couvre les étapes nécessaires requises pour définir la stratégie d'auditing d'une application.

Procédure 12.7. Définir une stratégie d'auditing

- Configurer la propriété **`org.hibernate.envers.audit_strategy`** qui se trouve dans le ficher **`persistence.xml`** de l'application. Si la propriété est définie dans le fichier **`persistence.xml`**, alors la stratégie d'audit par défaut sera utilisée.

  Exemple 12.28. Définir la stratégie d'auditing par défaut

  ```
  <property name="org.hibernate.envers.audit_strategy"
  value="org.hibernate.envers.strategy.DefaultAuditStrategy"/>
  ```

  Exemple 12.29. Définir la stratégie d'auditing de validité

  ```
  <property name="org.hibernate.envers.audit_strategy"
  value="org.hibernate.envers.strategy.ValidityAuditStrategy"/>
  ```

Rapporter un bogue

12.7.4. Introduction à l'auditing d'entités

12.7.4.1. Ajouter une support d'auditing à une entité JPA

JBoss EAP 6 utilise l'auditing d'entités, par l'intermédiaire de Section 12.7.1, « Hibernate Envers », pour suivre les changements historiques d'une classe persistante. Cette rubrique couvre l'ajout d'un support d'auditing pour une entité JPA.

Procédure 12.8. Ajouter une support d'auditing à une entité JPA

1. Configurer les paramètres d'auditing disponible qui conviennent au déploiement : Section 12.7.5.1, « Configurer les paramètres Envers ».

2. Ouvrir l'entité JPA à auditer.

3. Importer l'interface **`org.hibernate.envers.Audited`**.

4. Appliquer l'annotation **`@Audited`** à chaque champ ou propriété à auditer, ou l'appliquer une seule fois à toute la classe.

Exemple 12.30. Audit de deux champs

```
import org.hibernate.envers.Audited;

import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.GeneratedValue;
import javax.persistence.Column;

@Entity
public class Person {
    @Id
    @GeneratedValue
    private int id;

    @Audited
    private String name;

    private String surname;

    @ManyToOne
    @Audited
    private Address address;

    // add getters, setters, constructors, equals and hashCode
here
}
```

Exemple 12.31. Audit d'une classe complète

```
import org.hibernate.envers.Audited;

import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.GeneratedValue;
import javax.persistence.Column;

@Entity
@Audited
public class Person {
    @Id
    @GeneratedValue
    private int id;

    private String name;

    private String surname;
```

```
    @ManyToOne
    private Address address;

    // add getters, setters, constructors, equals and hashCode
here
}
```

Résultat

L'entité JPA a été configurée pour l'auditing. Un tableau intitulé ***`Entity`*_AUD** sera créé pour stocker les changements historiques.

Rapporter un bogue

12.7.5. Configuration

12.7.5.1. Configurer les paramètres Envers

JBoss EAP 6 utilise l'auditing d'entités, par l'intermédiaire d'Hibernate Envers, pour suivre les changements historiques d'une classe persistante. Cette rubrique couvre la configuration des paramètres Envers disponibles.

Procédure 12.9. Configurer les paramètres Envers

1. Ouvrir le fichier **`persistence.xml`** de l'application.
2. Ajouter, supprimer ou configurer les propriétés Envers selon les besoins. Pour obtenir une liste complète des propriétés disponibles, consulter Section 12.7.5.4, « Propriétés de configuration Envers ».

Exemple 12.32. Exemples de paramètres Envers

```
<persistence-unit name="mypc">
  <description>Persistence Unit.</description>
  <jta-data-source>java:jboss/datasources/ExampleDS</jta-data-source>
  <shared-cache-mode>ENABLE_SELECTIVE</shared-cache-mode>
  <properties>
    <property name="hibernate.hbm2ddl.auto" value="create-drop" />
    <property name="hibernate.show_sql" value="true" />
    <property name="hibernate.cache.use_second_level_cache" value="true"
/>
    <property name="hibernate.cache.use_query_cache" value="true" />
    <property name="hibernate.generate_statistics" value="true" />
    <property name="org.hibernate.envers.versionsTableSuffix" value="_V"
/>
    <property name="org.hibernate.envers.revisionFieldName"
value="ver_rev" />
  </properties>
</persistence-unit>
```

Résultat

L'auditing a été configuré pour toutes les entités JPA de l'application.

Rapporter un bogue

12.7.5.2. Activer ou désactiver l'auditing en cours d'exécution

Résumé

Cette tâche couvre les étapes de configuration requises pour activer/désactiver l'auditing de version d'entité en cours d'exécution.

Procédure 12.10. Activer/désactiver l'auditing

1. Sous-classe la classe **`AuditEventListener`**.
2. Remplace les méthodes suivantes appelées sur les événements Hibernate :
   - onPostInsert
   - onPostUpdate
   - onPostDelete
   - onPreUpdateCollection
   - onPreRemoveCollection
   - onPostRecreateCollection
3. Spécifie la sous-classe comme listener des événements.
4. Détermine si le changement doit être vérifié.
5. Passe l'appel à la sur-classe si le changement doit être vérifié.

Rapporter un bogue

12.7.5.3. Configurer l'auditing conditionnel.

Résumé

Hibernate Envers persiste les données d'auditing en réaction à certains événements, en utilisant une série de listeners d'événements. Ces listeners sont enregistrés automatiquement si le jar Envers est sur le chemin de classe. Cette tâche couvre les étapes requises pour mettre en place l'auditing conditionnel, en remplaçant certains listeners d'événements Envers.

Procédure 12.11. Implémentation de l'Auditing conditionnel

1. Définir la propriété Hibernate **`hibernate.listeners.envers.autoRegister`** à false dans le fichier **`persistence.xml`**.
2. Sous-classer chaque listener d'événement à remplacer. Mettez la logique d'auditing conditionnel dans la sous-classe, et appeler la super méthode si on doit procéder à l'auditing.
3. Créer une implémentation personnalisée de **`org.hibernate.integrator.spi.Integrator`**, semblable à

`org.hibernate.envers.event.EnversIntegrator`. Utiliser les sous-classes de listeners d'événements créées à la deuxième étape, à la place des classes par défaut.

4. Ajouter un fichier `META-INF/services/org.hibernate.integrator.spi.Integrator` au jar. Ce fichier doit contenir le nom complet de la classe qui implémente d'interface.

Résultat

L'auditing conditionnel alors est configuré, remplaçant ainsi les listeners d'événements Envers par défaut.

Rapporter un bogue

12.7.5.4. Propriétés de configuration Envers

Tableau 12.12. Paramètres de configuration du Versioning de données d'entité

Nom de propriété	Valeur par défaut	Description
org.hibernate.envers.audit_table_prefix		String attaché au nom d'une entité auditée, pour créer le nom de l'entité qui contiendra l'audit en formation.
org.hibernate.envers.audit_table_suffix	_AUD	String attaché au nom d'une entité auditée, pour créer le nom de l'entité qui contiendra l'audit en formation. Par exemple, si une entité ayant comme nom de tableau `Person` est auditée, Envers créera un tableau nommé `Person_AUD` pour stocker les données historiques.
org.hibernate.envers.revision_field_name	REV	Le nom du champ qui contient le numéro de révision dans l'entité d'audit.
org.hibernate.envers.revision_type_field_name	REVTYPE	Le nom du champ de l'entité d'audit qui contient le type de révision. Les types actuels de révision possibles sont : `add`, `mod` et `del`.
org.hibernate.envers.revision_on_collection_change	true	Cette propriété indique si une révision doit être créée si un champ relationnel qui n'appartient à personne change. Il peut s'agir d'une collection d'une relation one-to-many, ou bien du champ qui utilise l'attribut `mappedBy` dans une relation one-to-one.
org.hibernate.envers.do_not_audit_optimistic_locking_field	true	Si true, les propriétés utilisées pour le verrouillage optimiste (annoté par `@Version`) sera exclus automatiquement de l'audit.

Nom de propriété	Valeur par défaut	Description
org.hibernate.envers.store_data_at_delete	false	Cette propriété définit si oui ou non les données d'entité doivent être stockées dans la révision lorsque l'entité est supprimée, au lieu de l'ID uniquement, avec toutes les autres propriétés marquées comme null. Ce n'est pas généralement nécessaire, car les données sont présentes dans l'avant dernière révision. Parfois, il est plus facile et plus efficace d'y accéder dans la dernière révision. Cependant, cela signifie que les données de l'entité contenues avant suppression apparaissent deux fois.
org.hibernate.envers.default_schema	null (comme dans les tableaux normaux)	Le nom de schéma par défaut utilisé pour les tableaux d'audit. Peut être substitué à l'aide de l'annotation **`@AuditTable(schema="...")`**. Si absent, le schéma sera le même que le schéma des tableaux normaux.
org.hibernate.envers.default_catalog	null (comme dans les tableaux normaux)	Le nom de catalogue par défaut qui doit être utilisé dans les tableaux d'audit. Peut être substitué par l'annotation **`@AuditTable(catalog="...")`**. Si non présent, le catalogue sera le même que le catalogue des tableaux normaux.
org.hibernate.envers.audit_strategy	org.hibernate.envers.strategy.DefaultAuditStrategy	Cette propriété définit la stratégie d'auditing qui doit être utilisée lors de la persistance des données d'audit. Par défaut, seule la révision où une entité a été modifiée est stockée. Alternativement, **`org.hibernate.envers.strategy.ValidityAuditStrategy`** stocke la révision du départ et révision de la fin. Ceux-ci définissent ensemble quand une ligne d'auditing est valide.
org.hibernate.envers.audit_strategy_validity_end_rev_field_name	REVEND	Le nom de la colonne qui contiendra le dernier numéro de révision dans les entités d'auditing. Cette propriété n'est valide que si la stratégie d'auditing de validité est utilisée.

Nom de propriété	Valeur par défaut	Description
org.hibernate.envers.audit_strategy_validity_store_revend_timestamp	false	Cette propriété définit si l'horodatage du dernier numéro de révision des dernières données valides doit être conservé en plus de la dernière révision elle-même. Ceci est utile pour pouvoir purger les archives d'audit sur une base de données relationnelle en utilisant le partitionnement de la table. Le partitionnement exige une colonne qui existe dans la table. Cette propriété n'est évaluée que si la **`ValidityAuditStrategy`** est utilisée
org.hibernate.envers.audit_strategy_validity_revend_timestamp_field_name	REVEND_TSTMP	Nom de la colonne d'horodatage de la dernière révision au moment où les données étaient encore valides. Utilisé uniquement si **`ValidityAuditStrategy`** est utilisé, et si **`org.hibernate.envers.audit_strategy_validity_store_revend_timestamp`** évalue sur true.

Rapporter un bogue

12.7.6. Recherches

12.7.6.1. Suppression de l'information d'auditing

Résumé

Hibernate Envers fournit une fonctionnalité qui permet d'extraire des informations d'audit par des requêtes. Cette section vous donne des exemples de telles requêtes.

NOTE

Les requêtes sur données auditées seront, dans la plupart des cas, bien plus lentes que les requêtes correspondantes sur données **`live`**, car elles impliquent des sous-sélections en corrélation.

Exemple 12.33. Chercher des entités d'une classe pour une révision donnée

Le point d'entrée de ce type de recherche est :

```
AuditQuery query = getAuditReader()
    .createQuery()
    .forEntitiesAtRevision(MyEntity.class, revisionNumber);
```

Des contraintes peuvent alors être spécifiées par la classe de fabrique **`AuditEntity`**. La recherche ci-dessous ne sélectionne que des entités dont la propriété **`nom`** est égale à **`John`** :

```
query.add(AuditEntity.property("name").eq("John"));
```

Les demandes ci-dessous ne sélectionnent que des entités liées à une entité donnée :

```
query.add(AuditEntity.property("address").eq(relatedEntityInstance));
// or
query.add(AuditEntity.relatedId("address").eq(relatedEntityId));
```

Les résultats peuvent alors être classés, limités, et avoir des agrégations ou des projections (sauf en groupings) définis. L'exemple ci-dessous correspond à une recherche complète.

```
List personsAtAddress = getAuditReader().createQuery()
    .forEntitiesAtRevision(Person.class, 12)
    .addOrder(AuditEntity.property("surname").desc())
    .add(AuditEntity.relatedId("address").eq(addressId))
    .setFirstResult(4)
    .setMaxResults(2)
    .getResultList();
```

Exemple 12.34. Chercher les révisions de demandes pour lesquelles les entités d'une classe donnée ont changé

Le point d'entrée de ce type de recherche est :

```
AuditQuery query = getAuditReader().createQuery()
    .forRevisionsOfEntity(MyEntity.class, false, true);
```

Des contraintes peuvent être ajoutées à cette recherche de la même façon que dans l'exemple précédent. Il y a d'autres possibilités pour cette recherche :

`AuditEntity.revisionNumber()`

Spécifie les contraintes, les projections et l'ordre en considération desquels l'entité auditée a été modifiée dans le numéro de révision.

`AuditEntity.revisionProperty(propertyName)`

Spécifier les contraintes, les projections et l'ordre de l'entité de révision sur une propriété, en correspondance à la révision dans laquelle l'entité auditée a été modifiée.

`AuditEntity.revisionType()`

Fournit des accès au type de révision (ADD, MOD, DEL).

Les résultats de la recherche peuvent alors être ajustés selon les besoins. La recherche ci-dessous sélectionne le plus petit nombre de révisions au cours desquelles l'entité de la classe de **`MyEntity`**, ayant pour ID **`entityId`** a changé, après le numéro de révision 42 :

```
Number revision = (Number) getAuditReader().createQuery()
    .forRevisionsOfEntity(MyEntity.class, false, true)
```

```
    .setProjection(AuditEntity.revisionNumber().min())
    .add(AuditEntity.id().eq(entityId))
    .add(AuditEntity.revisionNumber().gt(42))
    .getSingleResult();
```

Les demandes de révision peuvent aussi minimiser/maximiser une propriété. La recherche ci-dessous sélectionne la révision au cours de laquelle la valeur de **`actualDate`** d'une entité donnée est supérieure à une valeur donnée, mais aussi petite que possible :

```
Number revision = (Number) getAuditReader().createQuery()
    .forRevisionsOfEntity(MyEntity.class, false, true)
    // We are only interested in the first revision
    .setProjection(AuditEntity.revisionNumber().min())
    .add(AuditEntity.property("actualDate").minimize()
        .add(AuditEntity.property("actualDate").ge(givenDate))
        .add(AuditEntity.id().eq(givenEntityId)))
    .getSingleResult();
```

Les méthodes **`minimize()`** et **`maximize()`** renvoient un critère auquel des contraintes peuvent être ajoutées, et qui doit être respecté par les entités avec des propriétés maximisées/minimisées.

Il y a deux paramètres booléens qui ont été passés quand on crée une recherche.

`selectEntitiesOnly`

Ce paramètre n'est valide que lorsqu'une projection explicite n'est pas définie.

Si true, le résultat de la recherche correspondra à une liste d'entités qui ont changé lors de révisions satisfaisant les contraintes spécifiées.

Si false, le résultat correspondra à une liste de trois tableaux d'éléments. Le premier élément correspondra à l'instance d'entité modifiée. Le second correspondra à une entité qui contient les données de révision. Si aucune entité personnalisée n'est utilisée, alors on aura une instance de **`DefaultRevisionEntity`**. Le troisième tableau d'éléments correspondra au type de révision (ADD, MOD, DEL).

`selectDeletedEntities`

Ce paramètre indique si les révisions dans lesquelles les entités ont été supprimées doivent être incluses dans les résultats. Si true, les entités auront le type de révision **`DEL`**, et tous les champs, à part l'id, auront pour valeur **`null`**.

Exemple 12.35. Chercher les révisions d'une entité qui ont pu modifier une propriété donnée

La recherche ci-dessous renverra toutes les révisions de **`MyEntity`** pour une id donnée, quand la propriété **`actualDate`** a été modifiée.

```
AuditQuery query = getAuditReader().createQuery()
  .forRevisionsOfEntity(MyEntity.class, false, true)
  .add(AuditEntity.id().eq(id));
  .add(AuditEntity.property("actualDate").hasChanged())
```

La condition **`hasChanged`** peut être combinée à des critères supplémentaires. La recherche ci-dessous renvoie une coupe horizontale de **`MyEntity`** au moment où le *revisionNumber* est créé. Ce

sera limité aux révisions qui ont modifié **prop1**, et non pas **prop2**.

```
AuditQuery query = getAuditReader().createQuery()
  .forEntitiesAtRevision(MyEntity.class, revisionNumber)
  .add(AuditEntity.property("prop1").hasChanged())
  .add(AuditEntity.property("prop2").hasNotChanged());
```

Le jeu de résultats contiendra également des révisions avec des nombres inférieurs au *revisionNumber*. Cela signifie que cette recherche ne peut être interprétée comme « Renvoyer toutes les **MyEntities** modifiées dans le *revisionNumber* avec **prop1** modifié et **prop2** intact. »

Le recherche ci-dessous montre comment ce résultat peut être retourné, par l'intermédiaire de la recherche **forEntitiesModifiedAtRevision** :

```
AuditQuery query = getAuditReader().createQuery()
  .forEntitiesModifiedAtRevision(MyEntity.class, revisionNumber)
  .add(AuditEntity.property("prop1").hasChanged())
  .add(AuditEntity.property("prop2").hasNotChanged());
```

Exemple 12.36. Recherche d'entités modifiées dans une révision donnée

L'exemple ci-dessous montre la recherche de base d'entités modifiées pour une révision donnée. Cela permet aux noms d'entités et à leurs classes Java correspondantes d'être modifiées dans une révision spéciale que l'on souhaite extraire :

```
Set<Pair<String, Class>> modifiedEntityTypes = getAuditReader()

.getCrossTypeRevisionChangesReader().findEntityTypes(revisionNumber);
```

Il y a un certain nombre de recherches accessibles de **org.hibernate.envers.CrossTypeRevisionChangesReader**:

List<Object> findEntities(Number)

Retourne des instantanés de toutes les entités d'auditing changées (ajoutées, mises à jour et supprimées) dans une révision donnée. Exécute des requêtes SQL **n + 1**, où **n** est un nombre de classes d'entités différentes modifiées dans la révision spécifiée.

List<Object> findEntities(Number, RevisionType)

Retourne des instantanés de toutes les entités d'auditing changées (ajoutées, mises à jour et supprimées) dans une révision donnée filtrée par type de modification. Exécute des requêtes SQL **n + 1**, où **n** est un nombre de classes d'entités différentes modifiées dans la révision spécifiée.

Map<RevisionType, List<Object>> findEntitiesGroupByRevisionType(Number)

Retourne une carte contenant les listes des instantanés d'entité regroupés par opération de modification (ajout, mise à jour et suppression). Exécute des requêtes SQL **3n + 1**, où **n** est un nombre de classes d'entités différentes modifiées au cours de la révision spécifiée.

Rapporter un bogue

12.8. RÉGLAGE DE LA PERFORMANCE

12.8.1. Algorithmes de chargement de lots alternatifs

Hibernate permet de charger les données d'associations à l'aide de l'une des quatre stratégies de récupération : joindre (join), sélectionnez (select), sous-sélectionner (subselect) et par lots (batch). En dehors de ces quatre stratégies, le chargement par lots permet des plus grands gains de performance car c'est une stratégie d'optimisation pour l'extraction sélective. Dans cette stratégie, Hibernate récupère un lot d'instances d'entités ou de collections dans une instruction SELECT unique en spécifiant une liste de clés primaires ou étrangères. L'extraction de lot est une optimisation de la stratégie de récupération sélective différée.

Il existe deux manières de configurer l'extraction par batch: Niveau Par classe ou Niveau Par collection.

- Niveau Par classe

  Lors du chargement de données sur un niveau par classe, Hibernate, requiert la taille de lot de l'association à pré-charger lorsqu'il est interrogé. Par exemple, imaginons que lors de l'exécution, vous ayez 30 instances d'un objet **`voiture`** chargées dans la session. Chaque objet **`voiture`** appartient à un objet **`propriétaire`**. Si vous deviez parcourir tous les objets **`voiture`** et appeler leurs propriétaires, par un chargement **`différé`** (lazy loading), Hibernate émettra 30 instructions select - une pour chaque propriétaire. Il s'agit d'un goulot d'étranglement de performance.

  À la place, vous pouvez dire à Hibernate de charger à l'avance le prochain lot de propriétaires avant qu'ils aient été interrogés par une requête. Quand un objet **`owner`** (propriétaire) est interrogé, Hibernate interrogera ces objets à plusieurs reprises dans le même énoncé SELECT.

  Le nombre d'objets **`owner`** (propriétaire) à interroger à l'avance dépend du paramètre ***`batch-size`*** spécifié en cours de configuration :

  ```
  <class name="owner" batch-size="10"></class>
  ```

  Cela indique à Hibernate d'interroger au moins 10 objets **`owner`** au cas où ils seraient nécessaires dans un avenir proche. Lorsqu'un utilisateur interroge le propriétaire **`owner`** de **`car A`**, le propriétaire **`owner`** de **`car B`** peut déjà avoir été chargé dans un lot de chargement. Lorsque l'utilisateur a besoin du propriétaire **`owner`** de **`car B`**, au lieu d'aller à la base de données (et émettre une instruction SELECT), la valeur peut être récupérée de la session en cours.

  En plus du paramètre **`batch-size`**, Hibernate 4.2.0 a introduit un nouvel élément de configuration pour améliorer la performance de chargement de lots. La configuration s'appelle **`Batch Fetch Style`** et est spécifiée par le paramètre ***`hibernate.batch_fetch_style`***.

  Il existe trois styles d'extraction de lots pris en charge : LEGACY, PADDED et DYNAMIC. Pour spécifier le style, utiliser **`org.hibernate.cfg.AvailableSettings#BATCH_FETCH_STYLE`**.

  - LEGACY : dans le style hérité de chargement, une série de tailles de lots pré-construits basées **`ArrayHelper.getBatchSizes(int)`** sont utilisées. Les lots sont chargés à l'aide de la taille de lot de pré-construction suivante à partir du nombre d'identificateurs existants pouvant être mis en lot.

    En continuant avec l'exemple suivant, ayant une taille **`batch-size`** de 30, les tailles de lot pré-construits sont [30, 15, 10, 9, 8, 7, .., 1]. Si l'on tente de mettre en lots 29 identifiants, on

obtiendra des lots de 15, 10 et 4. Il y aura 3 requêtes SQL correspondantes, chacune chargeant 15, 10 et 4 propriétaires de la base de données.

- PADDED - similaire au style de chargement de classes LEGACY. Utilise toujours des tailles de lots pré-construits, mais utilise la taille de lot supérieure et remplit les espaces réservés de l'identifiant supplémentaire.

  Tout comme dans l'exemple ci-dessus, si 30 objets propriétaires doivent être initialisés, il ne pourra y avoir qu'une demande exécutée dans la base de données.

  Cependant, si 29 objets propriétaires doivent être initialisés, Hibernate n'exécutera qu'un seul énoncé select SQL de taille de lot 30, avec l'espace supplémentaire rempli par un identifiant répété.

- Dynamic - bien que toujours conforme à la limitation de la taille des lots, ce style de chargement dynamique de lots construit son énoncé SQL SELECT à l'aide du nombre réel d'objets à charger.

  Par exemple, pour 30 objets propriétaires, et une taille de lot maximale de 30, un appel pour récupérer 30 objets propriétaires entraînera une instruction SQL SELECT. Un appel pour récupérer 35 se traduira en deux énoncés SQL, ayant pour taille de lot 30 et 5, respectivement. Hibernate modifiera dynamiquement le second énoncé SQL pour maintenir à 5, le nombre requis, tout en restant sous la restriction de 30 pour la taille des lots. C'est différent de la version PADDED, car le deuxième SQL ne sera pas PADDED et à la différence du style LEGACY, il n'y a pas de taille fixe pour le second énoncé SQL - le second SQL est créé dynamiquement.

  Pour une requête inférieure à 30 identifiants, ce style ne chargera de façon dynamique que le nombre d'identifiants requis.

- Niveau Par collection

  Hibernate peut aussi mettre en lots des collections de chargement, en respectant la taille ou les styles des lots d'extraction suivant la liste de la section par-classe ci-dessus.

  Pour inverser l'exemple utilisé dans la section précédente, considérons que vous deviez charger tous les objets **`car`** (voiture) appartenant à chaque objet **`owner`** (propriétaire). Si 10 objets propriétaire **`owner`** sont chargés dans la session en cours, itérer tous les propriétaires va générer 10 énoncés SELECT, un pour chaque appel de méthode à **`getCars()`**. Si vous activez l'extraction de lot pour la collection de voitures dans le mappage du propriétaire, Hibernate pourra chercher ces collections à l'avance, comme illustré ci-dessous.

  ```
  <class name="Owner"><set name="cars" batch-size="5"></set></class>
  ```

  Donc, avec une taille de lot **`batch-size`** égale à 5 et en utilisant le style de lot hérité, Hibernate chargera 5 collections dans deux énoncés SELECT.

Rapporter un bogue

12.8.2. Mise en cache de second niveau d'objets de référence pour les données non mutables

Hibernate met en cache automatiquement les données dans la mémoire pour des performances améliorées. Ceci est accompli par un cache en mémoire, ce qui réduit le nombre de fois où les recherches de base de données sont nécessaires, surtout pour les données qui changent rarement.

Hibernate administre deux types de caches. Le cache principal (également appelé le cache de premier niveau) est obligatoire. Ce cache est associé à la session en cours et toutes les demandes doivent passer à travers. Le cache secondaire (également appelé le cache de second niveau) est facultatif et est consulté seulement après consultation préalable du cache primaire.

Les données sont stockées dans le cache de second niveau en commençant par un désassemblement dans un tableau d'états. Ce tableau est copié complètement, et cette copie complète est mise en cache. L'inverse est fait pour la lecture du cache. Cela fonctionne bien pour les données qui ne changent pas (données mutables), mais est inefficace pour des données immuables.

La copie complète des données est une opération coûteuse en termes de vitesse de traitement et d'utilisation de la mémoire. Pour les grands ensembles de données, la vitesse de traitement et la mémoire deviennent un facteur de limitation de la performance. Hibernate vous permet de spécifier le référencement des données immuables au lieu de les copier. Au lieu de copier des ensembles de données complets, Hibernate peut maintenant stocker la référence aux données dans le cache.

Cela peut être effectué en modifiant la valeur du paramètre de configuration de **`hibernate.cache.use_reference_entries`** à **`true`**. Par défaut, **`hibernate.cache.use_reference_entries`** est défini à **`false`**.

Lorsque **`hibernate.cache.use_reference_entries`** a pour valeur **`true`**, un objet de données immuables qui n'a pas d'associations n'est pas copié dans le cache de second niveau, et seulement sa référence y sera stockée.

AVERTISSEMENT

Lorsque **`hibernate.cache.use_reference_entries`** a pour valeur **`true`**, un objet de données immuables qui possède des associations sera copié dans le cache de second niveau.

Rapporter un bogue

CHAPITRE 13. SERVICES WEB JAX-RS

13.1. JAX-RS

JAX-RS est l'API Java des services web RESTful. Il fournit un support pour construire des services web avec REST, à l'aide d'annotations. Ces annotations simplifient le processus de mappage d'objets Java dans des ressources web. La spécification est définie dans http://www.jcp.org/en/jsr/detail?id=311.

RESTEasy est l'implémentation JBoss EAP 6 de JAX-RS. Il procure également des fonctionnalités supplémentaires de spécification.

JBoss EAP 6 est compatible avec JSR 311 - JAX-RS.

Pour démarrer avec JAX-RS et JBoss EAP 6, voir **`helloworld-rs`**, **`jax-rs-client`**, et **`kitchensink`** quickstart: Section 1.4.1.1, « Accès aux Quickstarts ».

Rapporter un bogue

13.2. RESTEASY

RESTEasy est une implémentation portable de l'API Java JAX-RS. Fournit également des fonctionnalités supplémentaires, comme un framework côté client (RESTEasy JAX-RS Client Framework) pour mapper les demandes sortantes vers les serveurs distants, ce qui permet à JAX-RS d'opérer en tant que client ou spécification côté serveur.

Rapporter un bogue

13.3. SERVICES WEB RESTFUL

Les services web RESTful sont conçus pour exposer les API sur le web. Ils sont destinés à améliorer la performance, l'adaptabilité, et la flexibilité par rapport aux services web traditionnels, tout en permettant aux clients d'accéder à des données et à des ressources par des URL prédictibles.

La spécification Java Enterprise Edition 6 des services RESTful est JAX-RS. Pour obtenir davantage d'informations sur JAX-RS, voir Section 13.1, « JAX-RS » et Section 13.2, « RESTEasy ».

Rapporter un bogue

13.4. ANNOTATIONS DÉFINIES RESTEASY

Tableau 13.1. Annotations JAX-RS/RESTEasy

Annotation	Utilisation
`ClientResponseType`	Il s'agit d'une annotation que vous pouvez ajouter à une interface de client RESTEasy qui a un type de renvoi Réponse.
`ContentEncoding`	Méta annotation qui indique un Contenet-Encoding à appliquer par l'annotation annotée.

Annotation	Utilisation
`DecorateTypes`	Doit être mis sur une classe DecoratorProcessor pour spécifier les types pris en charge.
`Decorator`	Méta annotation à mettre dans une autre annotation qui déclenche décoration.
`Form`	Peut être utilisé comme objet de valeur pour les requêtes/réponses entrantes/sortantes.
`StringParameterUnmarshallerBinder`	Méta annotation à mettre dans une autre annotation qui déclenche un StringParameterUnmarshaller à appliquer à un injecteur d'annotation basé chaîne.
`Cache`	Définit l'en-tête Cache-Control de réponse automatiquement.
`NoCache`	Définit la réponse Cache-Control à "nocache".
`ServerCached`	Indique que la réponse à cette méthode jax-rs devrait être mise en cache sur le serveur.
`ClientInterceptor`	Identifie un intercepteur en tant qu'intercepteur côté client.
`DecoderPrecedence`	Cet intercepteur est un décodeur Content_Encoding.
`EncoderPrecedence`	Cet intercepteur est un encodeur Content_Encoding.
`HeaderDecoratorPrecedence`	Les intercepteurs HeaderDecoratorPrecedence doivent toujours venir en premier car ils décorent une réponse (sur le serveur), ou une réquête sortante (sur le client) avec des en-têtes spéciaux définis utilisateur.
`RedirectPrecedence`	Doivent être mis dans un PreProcessInterceptor.
`SecurityPrecedence`	Doivent être mis dans un PreProcessInterceptor.
`ServerInterceptor`	Identifie un intercepteur en tant qu'intercepteur côté client.
`NoJackson`	Mis sur une classe, un paramètre, un champ ou une métode quand vous ne souhaitez pas déclencher un fournisseur Jackson.
`ImageWriterParams`	Annotation qu'une classe de ressource peut utiliser pour passer des paramètres au IIOImageProvider.

Annotation	Utilisation
`DoNotUseJAXBProvider`	Mettre ceci sur une classe ou sur un paramètre quand vous ne souhaitez pas utiliser le MessageBodyReader/Writer JAXB, mais que vous avez à la place un fournisseur plus spécifique que vous souhaitez utliser pour marshaler le type.
`Formatted`	Formate la sortie XML par des indentations et des nouvelles lignes. Il s'agit d'un Decorateur JAXB.
`IgnoreMediaTypes`	Mis sur un type, une méthode, un paramètre ou un champ pour indiquer au JAXRS de ne pas utiliser de fournisseur JAXB pour une certain type de media.
`Stylesheet`	Indique un en-tête de feuille de style XML.
`Wrapped`	Mettre ceci sur une méthode ou un paramètre quand vous souhaitez marshaller ou démarshaller une collection ou un tableau d'objets JAXB.
`WrappedMap`	Mettre ceci sur une méthode ou un paramètre quand vous souhaitez marshaller ou démarshaller une mappe d'objets JAXB.
`XmlHeader`	Définit un en-tête XML de document renvoyé.
`BadgerFish`	Un JSONConfig.
`Mapped`	Un JSONConfig.
`XmlNsMap`	Un JSONToXml.
`MultipartForm`	Peut être utilisé comme objet de valeur pour les requêtes/réponses entrantes/sortantes d'un type de mime multipart/form-data.
`PartType`	Doit être utilisé en conjonction avec des fournisseurs Multipart quand on écrit une liste ou un mappage de type multipart/*.
`XopWithMultipartRelated`	Cette annotation peut être utilisée pour traiter/produire des messages XOP entrants/sortants (packagés comme multipart/related) vers/en provenance d'objets annotés JAXB.
`After`	Utilisée pour ajouter un attribut d'expiration quand on signe ou comme contrôle.

Annotation	Utilisation
`Signed`	Annotation pratique qui déclenche la signature d'une requête ou d'une réponse utilisant la spécification DOSETA.
`Verify`	Vérification d'une signature entrante spécifiée dans un en-tête de signature.

Rapporter un bogue

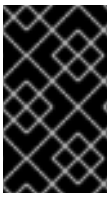
13.5. CONFIGURATION RESTEASY

13.5.1. Paramètres de configuration RESTEasy

Tableau 13.2. Éléments

Nom d'option	Valeur par défaut	Description
resteasy.servlet.mapping.prefix	Pas de valeur par défaut	Si le modèle d'url du mappage de servlet Resteasy n'est pas /*.
resteasy.scan	false	Scanner automatiquement les jars WEB-INF/lib et le répertoire WEB-INF/classes pour @Provider et les classes de ressource JAX-RS (@Path, @GET, @POST etc..) à la fois et les enregistrer.
resteasy.scan.providers	false	Scanner les classes @Provider et les enregistrer.
resteasy.scan.resources	false	Scanner les classes de ressources JAX-RS.
resteasy.providers	pas de valeur par défaut	Une liste de noms de classes @Provider complètes que vous souhaitez enregistrer délimitée par des virgules.
resteasy.use.builtin.providers	true	Indique si on doit ou non enregistrer les classes @Provider intégrées, par défaut.
resteasy.resources	Pas de valeur par défaut	Une liste de noms de classes de ressources JAX-RS complètes que vous souhaitez enregistrer délimitée par des virgules.

Nom d'option	Valeur par défaut	Description
resteasy.jndi.resources	Pas de valeur par défaut	Une liste délimitée par des virgules de noms JNDI qui référencent des objets que vous souhaitez enregistrer comme ressources JAX-RS.
javax.ws.rs.Application	Pas de valeur par défaut	Nom qualifié de classe d'application à amorcer en spec portable.
resteasy.media.type.mappings	Pas de valeur par défaut	Remplace le besoin d'un en-tête Accept par mappage des extensions de noms de fichier (comme .xml ou .txt) avec un type de media. Utilisé quand le client n'est pas en mesure d'utiliser un en-tête Accept pour choisir une représentation (par ex. un navigateur).
resteasy.language.mappings	Pas de valeur par défaut	Remplace le besoin d'un en-tête Accept-Language par mappage des extensions de noms de fichier (comme .en ou .fr) avec une langue. Utilisé quand le client n'est pas en mesure d'utiliser un en-tête Accept-Language pour choisir une représentation (par ex. un navigateur).
resteasy.document.expand.entity.references	false	Indique si on doit étendre les entitiés externes ou si on doit les remplacer par une chaine vide. Dans JBoss EAP 6.3, ce paramètre a comme valeur par défaut **`false`**, donc elles sont remplacées par une chaîne vide.



IMPORTANT

Dans un conteneur Servlet 3.0, les configurations **`resteasy.scan.*`** du fichier **`web.xml`** seront ignorées, et tous les composants annotés JAX-RS seront scannés automatiquement.

Rapporter un bogue

13.6. JAX-RS WEB SERVICE SECURITY

13.6.1. Activez la sécurité basée-rôle pour RESTEasy JAX-RS Web Service

Résumé

RESTEasy supporte les annotations @RolesAllowed, @PermitAll, et @DenyAll sur les méthodes JAX-RS. Cependant, il ne reconnaît pas ces annotations par défaut. Suivre les étapes suivantes pour configurer le fichier **`web.xml`** et pour activer la sécurité basée-rôle.

AVERTISSEMENT

Ne pas activer la sécurité basée-rôle si l'application utilise les EJB. Le conteneur EJB procurera la fonctionnalité à la place de RESTEasy.

Procédure 13.1. Activez la sécurité basée-rôle pour RESTEasy JAX-RS Web Service

1. Ouvrir le fichier **web.xml** de l'application dans un éditeur de texte.
2. Ajoutez le <context-param> suivant au fichier, dans les balises **web-app** :

```
<context-param>
    <param-name>resteasy.role.based.security</param-name>
    <param-value>true</param-value>
</context-param>
```

3. Déclarez tous les rôles utilisés dans le fichier RESTEasy JAX-RS WAR file, en utilisant les balises de <security-role> :

```
<security-role>
    <role-name>ROLE_NAME</role-name>
</security-role>
<security-role>
    <role-name>ROLE_NAME</role-name>
</security-role>
```

4. Autorisez l'accès à tous les URL gérés par le runtime JAX-RS pour tous les rôles :

```
<security-constraint>
    <web-resource-collection>
 <web-resource-name>Resteasy</web-resource-name>
 <url-pattern>/PATH</url-pattern>
    </web-resource-collection>
    <auth-constraint>
 <role-name>ROLE_NAME</role-name>
 <role-name>ROLE_NAME</role-name>
    </auth-constraint>
</security-constraint>
```

Résultat

La sécurité basée rôle à été activée dans l'application, avec un certain nombre de rôles définis.

Exemple 13.1. Exemple de configuration de sécurité basée rôles

```
<web-app>

    <context-param>
 <param-name>resteasy.role.based.security</param-name>
 <param-value>true</param-value>
```

```
    </context-param>

    <servlet-mapping>
<servlet-name>Resteasy</servlet-name>
<url-pattern>/*</url-pattern>
    </servlet-mapping>

    <security-constraint>
<web-resource-collection>
     <web-resource-name>Resteasy</web-resource-name>
     <url-pattern>/security</url-pattern>
</web-resource-collection>
<auth-constraint>
     <role-name>admin</role-name>
     <role-name>user</role-name>
</auth-constraint>
    </security-constraint>

    <security-role>
<role-name>admin</role-name>
    </security-role>
    <security-role>
<role-name>user</role-name>
    </security-role>

</web-app>
```

Rapporter un bogue

13.6.2. Sécuriser un service JAX-RS Web par des annotations

Résumé

Cette rubrique couvre les étapes à parcourir pour sécuriser un service JAX-RS Web par les annotations de sécurité supportées.

Procédure 13.2. Sécuriser un service JAX-RS Web par des annotations de sécurité supportées.

1. Activez la sécurité basée-rôle. Pour plus d'informations, voir Section 13.6.1, « Activez la sécurité basée-rôle pour RESTEasy JAX-RS Web Service ».

2. Ajoutez des annotations de sécurité au service JAX-RS Web. RESTEasy supporte les annotations suivantes :

   @RolesAllowed

   Définit les rôles qui peuvent accéder à la méthode. Tous les rôles doivent être définis dans le fichier **`web.xml`**.

   @PermitAll

   Autorise tous les rôles définis dans le fichier **`web.xml`** à accéder à la méthode.

   @DenyAll

   Refuse tout accès à la méthode.

Rapporter un bogue

13.7. RESTEASY LOGGING

13.7.1. JAX-RS Web Service Logging

RESTEasy supporte la journalisation via java.util.logging, log4j, et slf4j. Le framework est choisi avec l'algorithme suivant :

1. Si log4j est dans le chemin de classe de l'application, log4j sera utilisé.
2. Si slf4j est dans le chemin de classe de l'application, slf4j sera utilisé.
3. java.util.logging est la valeur par défaut si ni log4j ni slf4j sont dans le chemin de classe.
4. Si le param de contexte de servlet resteasy.logger.type est défini à java.util.logging, log4j, ou slf4j remplaceront le comportement par défaut.

Rapporter un bogue

13.7.2. Catégories de journalisation définies dans RESTEasy

Tableau 13.3. Catégories

Catégorie	Fonction
`org.jboss.resteasy.core`	Journalise toutes les activités par l'implémentation RESEasy principale.
`org.jboss.resteasy.plugins.providers`	Journalise toutes les activités par les fournisseurs d'entité RESTEasy.
`org.jboss.resteasy.plugins.server`	Journalise tous les activités par l'implémentation RESEasy.
`org.jboss.resteasy.specimpl`	Journalise toutes les activités par les classes d'implémentation JAX-RS.
`org.jboss.resteasy.mock`	Journalise les activités par le framework factice RESTEasy.

Rapporter un bogue

13.8. GESTION DES EXCEPTIONS

13.8.1. Créer un mappeur d'exceptions

Résumé

Les mappeurs d'exceptions sont des composants fournis par des applications, personnalisés qui interceptent les exceptions lancées et qui écrivent des réponses HTTP spécifiques.

Exemple 13.2. Mappeur d'exceptions

Un mappeur d'exceptions est une classe annotée par @Provider qui implémente l'interface **`ExceptionMapper`** .

Voici un exemple de mappeur d'exceptions ci-dessous.

```
@Provider
public class EJBExceptionMapper implements
ExceptionMapper<javax.ejb.EJBException>
    {
    Response toResponse(EJBException exception) {
    return Response.status(500).build();
    }
}
```

Pour enregistrer un mappeur d'exceptions, listez-le dans le fichier **`web.xml`** sous le paramètre de contexte **`resteasy.providers`**, ou bien renvoyez-le par programmation via la classe **`ResteasyProviderFactory`**.

Rapporter un bogue

13.8.2. Exceptions RESTEasy lancées en interne

Tableau 13.4. Liste des exceptions

Exception	Code HTTP	Description
BadRequestException	400	Échec de la requête. La requête n'a pas été formatée correctement, ou il y a eu un problème lors de son traitement.
UnauthorizedException	401	Non autorisé. Exception de sécurité envoyée si vous utilisez la sécurité basée rôle annotée RESTEasy.
InternalServerErrorException	500	Erreur interne de serveur.
MethodNotAllowedException	405	Il n'y a pas de méthode JAX-RS pour la ressource qui peut gérer l'opération HTTP invoquée.
NotAcceptableException	406	Il n'y a pas de méthode JAX-RS qui puisse produire les types de media listés dans l'en-tête Accept
NotFoundException	404	Il n'y a pas de méthode JAX-RS qui serve le chemin/la ressource de la requête.

Exception	Code HTTP	Description
ReaderException	400	Toutes les exceptions envoyées, à partir des **`MessageBodyWriters`** sont incluses dans cette exception. S'il n'y a pas de **`ExceptionMapper`** dans l'exception, ou si l'exception n'est pas une **`WebApplicationException`**, alors RESTEasy renverra un code 400 par défaut.
WriterException	500	Toutes les exceptions envoyées, à partir des **`MessageBodyWriters`** sont incluses dans cette exception. S'il n'y a pas de **`ExceptionMapper`** dans l'exception, ou si l'exception n'est pas une **`WebApplicationException`**, alors RESTEasy renverra un code 400 par défaut.
JAXBUnmarshalException	400	Les fournisseurs JAXB (XML et Jettison) envoient cette exception sur les lectures. Peuvent correspondre à des wrapping JAXBExceptions. Cette classe s'étend à **`ReaderException`**.
JAXBMarshalException	500	Les fournisseurs JAXB (XML et Jettison) envoient cette exception sur les lectures. Peuvent correspondre à des wrapping JAXBExceptions. Cette classe s'étend à **`WriterException`**.
ApplicationException	S/O	Encapsule toutes les exceptions levées à partir du code d'application. Fonctionne de la même façon que **`InvocationTargetException`**. S'il y a un ExceptionMapper pour l'exception encapsulée, elle sera utilisée pour traiter la demande
Échec	S/O	Erreur RESTEasy interne. Non journalisée.

Exception	Code HTTP	Description
LoggableFailure	S/O	Erreur RESTEasy interne. Journalisée.
DefaultOptionsMethodException	S/O	Si l'utilisateur invoque **`HTTP OPTIONS`** et qu'il n'y a pas de méthode JAX-RS, RESTEasy fournira un comportement par défaut en lançant cette exception.

Rapporter un bogue

13.9. INTERCEPTEURS RESTEASY

13.9.1. Interception des invocations JAX-RS

Résumé

RESTEasy peut intercepter les invocations JAX-RS et les re-router vers des objets style listener que l'on appelle des intercepteurs. Cette section couvre les descriptions de quatre types d'intercepteurs.

Exemple 13.3. Intercepteurs MessageBodyReader/Writer

Les MessageBodyReaderInterceptors et les MessageBodyWriterInterceptors peuvent être utilisés soit côté cleint, soit côté serveur. Ils sont annotés par **`@Provider`**, ainsi que par **`@ServerInterceptor`** ou **`@ClientInterceptor`**, de façon à ce que RESTEasy sache s'il doit les ajouter ou non à la liste d'intercepteurs.

Ces intercepteurs encapsulent l'invocation de **`MessageBodyReader.readFrom()`** ou de **`MessageBodyWriter.writeTo()`**. Ils peuvent être utilisés pour encapsuler les flux d'entrée ou de sortie.

Le Support RESTEasy GZIP possède des intercepteurs qui créent ou remplacent les flux d'entrée ou de sortie par défaut par un GzipOutputStream ou un GzipInputStream afin que l'encodage gzip puisse opérer. Ils peuvent également être utilisés pour ajouter les en-têtes à la réponse, ou la requête sortante côté client.

```
public interface MessageBodyReaderInterceptor
  {
     Object read(MessageBodyReaderContext context) throws IOException,
WebApplicationException;

  }

public interface MessageBodyWriterInterceptor
  {
     void write(MessageBodyWriterContext context) throws IOException,
WebApplicationException;

  }
```

Les intercepteurs et le MessageBodyReader ou Writer sont invoqués dans une seule pile d'appels de Java. **`MessageBodyReaderContext.proceed()`** ou **`MessageBodyWriterContext.proceed()`** est appelé pour aller vers le prochain intercepteur, ou, s'il n'y a plus d'intercepteur à invoquer, la méthode **`readFrom()`** ou **`writeTo()`** du MessageBodyReader ou du MessageBodyWriter. Cette encapsulation permet aux objets d'être modifiés avant d'aller vers le Reader ou Writer, puis d'être nettoyés, après **`proceed()`**.

L'exemple ci-dessous montre un serveur côté intercepteur, qui ajoute une valeur d'en-tête à la réponse.

```
@Provider
@ServerInterceptor
public class MyHeaderDecorator implements MessageBodyWriterInterceptor {

    public void write(MessageBodyWriterContext context) throws
IOException, WebApplicationException
    {
       context.getHeaders().add("My-Header", "custom");
       context.proceed();
    }
}
```

Exemple 13.4. PreProcessInterceptor

Les PreProcessInterceptors sont exécutés après qu'une méthode de ressource JAX-RS ait été trouvée pour l'invocation, mais avant l'invocation réelle. Ils sont annotés avec @ServerInterceptor et exécutent en séquence.

Ces interfaces ne sont pas utilisables sur le serveur. Elles peuvent être utilisées pour implémenter des fonctionnalités de sécurité, ou afin de préempter la demande Java. La mise en œuvre de la sécurité RESTEasy utilise ce type d'intercepteur pour abandonner les demandes avant qu'elles aient lieu si l'utilisateur ne passe pas d'autorisation. Le framework de mise en cache de RESTEasy utilise ceci également pour renvoyer des réponses mises en cache afin d'éviter une fois de plus l'invocation de méthodes.

```
public interface PreProcessInterceptor
    {
       ServerResponse preProcess(HttpRequest request, ResourceMethod
method) throws Failure, WebApplicationException;
    }
```

Si la méthode **`preProcess()`** renvoie une ServerResponse, alors la méthode JAX-RS sous-jacente ne sera pas invoquée, et le runtime traitera la réponse et retournera au client. Si la méthode **`preProcess()`** ne renvoie pas une ServerResponse, la méthode JAX-RS sous-jacente sera alors invoquée.

Exemple 13.5. PostProcessInterceptors

Les PreProcessInterceptors sont exécutés après qu'une méthode de ressource JAX-RS ait été invoquée, mais avant que MessageBodyWriters ait été invoqué. Ils sont utilisés si un en-tête de réponse a besoin d'être défini quand un MessageBodyWriter n'est pas invoqué.

Ils ne peuvent être utilisés que du côté du serveur. Ils ne peuvent pas encapsuler quoi que ce soit, et sont invoqués séquentiellement.

```
public interface PostProcessInterceptor
  {
    void postProcess(ServerResponse response);
  }
```

Exemple 13.6. ClientExecutionInterceptors

Les ClientExecutionInterceptors ne sont utilisables que du côté client. Ils sont encapsulés dans l'invocation HTTP qui va vers le serveur. Ils doivent être annotés par **`@ClientInterceptor`** et **`@Provider`**. Ces intercepteurs sont exécutés après que le MessageBodyWriter, et que le ClientRequest aient été créés côté client.

Le support RESTEasy GZIP utilise des ClientExecutionInterceptors pour que l'en-tête Accept puisse contenir «gzip, deflate» avant que la demande soit envoyée. Le cache du client RESTEasy l'utilise pour vérifier si son cache contient la ressource avant qu'elle soit envoyée.

```
public interface ClientExecutionInterceptor
{
  ClientResponse execute(ClientExecutionContext ctx) throws Exception;
}

public interface ClientExecutionContext
{
  ClientRequest getRequest();

  ClientResponse proceed() throws Exception;
}
```

Rapporter un bogue

13.9.2. Lier un intercepteur à une méthode JAX-RS

Résumé

Par défaut, tous les intercepteurs sont invoqués pour chaque requête. L'interface **`AcceptedByMethod`** peut être implémentée pour préciser ce comportement.

Exemple 13.7. Exemple de liaison d'intercepteurs

RESTEasy appelera la méthode **`accept()`** pour les intercepteurs qui implémentent l'interface **`AcceptedByMethod`**. Si la méthode renvoie true, l'intercepteur sera ajouté à la chaîne d'appel de la méthode JAX-RS; sinon, il sera ignoré avec cette méthode.

Dans l'exemple ci-dessous, **`accept()`** détermine si l'annotation @GET est présente dans la méthode JAX-RS. Si tel est le cas, l'intercepteur sera appliqué à la chaîne d'appel de la méthode.

```
@Provider
@ServerInterceptor
public class MyHeaderDecorator implements MessageBodyWriterInterceptor,
```

```
AcceptedByMethod {

    public boolean accept(Class declaring, Method method) {
       return method.isAnnotationPresent(GET.class);
    }

   public void write(MessageBodyWriterContext context) throws
IOException, WebApplicationException
   {
      context.getHeaders().add("My-Header", "custom");
      context.proceed();
   }
}
```

Rapporter un bogue

13.9.3. Enregistrer un intercepteur

Résumé

Cette section couvre la façon dont on doit enregistrer un intercepteur RESTEasy JAX-RS dans une application.

Procédure 13.3. Enregistrer un intercepteur

- Pour enregistrer un intercepteur, listez-le dans le fichier **`web.xml`** sous le paramètre de contexte **`resteasy.providers`**, ou bien renvoyez-le sous forme de classe ou d'objet dans **`Application.getClasses()`** ou **`Application.getSingletons()`**

Rapporter un bogue

13.9.4. Familles de précédence d'intercepteur

13.9.4.1. Familles de précédence d'intercepteur

Résumé

Les intercepteurs peuvent être sensibles à l'ordre dans lequel ils sont invoqués. Mettre les intercepteurs de groupes RESTEasy en familles pour faciliter l'ordonnancement. Cette section couvre les familles de précédence d'intercepteurs intégrés et les intercepteurs associés à chaque famille.

Il existe cinq familles prédéfinies. Elles sont invoquées dans l'ordre suivant :

SECURITY

Les intercepteurs SECURITY sont normalement des PreProcessInterceptors. Ils sont invoqués en premier car on ne doit pas faire grand chose avant que l'invocation soit autorisée.

HEADER_DECORATOR

Les intercepteurs HEADER_DECORATOR ajoutent des en-têtes à une réponse ou à une requête sortante. Ils suivent les intercepteurs de sécurité car les en-têtes ajoutés peuvent affecter le comportement des autres familles d'intercepteurs.

ENCODER

Les intercepteurs ENCODER changent l'OutputStream. Par exemple, l'intercepteur GZIP crée un GZIPOutputStream pour envelopper l'outputStream réel pour la compression.

REDIRECT

Les intercepteurs REDIRECT sont normalement utilisés dans PreProcessInterceptors, car ils peuvent router à nouveau la demande et éviter ainsi totalement la méthode JAX-RS.

DECODER

Les intercepteurs DECODER englobent l'InputStream. Par exemple, le décodeur d'intercepteur GZIP englobe le GZIPInputStream.

Les intercepteurs qui ne sont pas associés à une famille de précédence sont invoqués après les autres. Pour assigner un intercepteur à une famille de précédence, utiliser l'annotation @Precedence, et voir Section 13.4, « Annotations définies RESTEasy ».

Rapporter un bogue

13.9.4.2. Définissez une famille de précédence d'intercepteurs personnalisée

Résumé

Les familles de précédence personnalisées peuvent être créées ou enregistrées dans le fichier **`web.xml`**. Ce sujet couvre des exemples des paramètres de contexte disponibles pour définir les familles de précédence d'intercepteurs.

Il existe trois paramètres de contexte qui puissent être utilisés pour définir une nouvelle famille de précédence.

Exemple 13.8. resteasy.append.interceptor.precedence

Le paramètre de contexte **`resteasy.append.interceptor.precedence`** rajoute la nouvelle famille de précédence à la liste de famille de précédent par défaut.

```
<context-param>
    <param-name>resteasy.append.interceptor.precedence</param-name>
    <param-value>CUSTOM_PRECEDENCE_FAMILY</param-value>
</context-param>
```

Exemple 13.9. resteasy.interceptor.before.precedence

Le paramètre de contexte **`resteasy.interceptor.before.precedence`** définit la famille de précédence par défaut exécutée avant la famille personnalisée. La valeur du paramètre est sous la forme *DEFAULT_PRECEDENCE_FAMILY*/*CUSTOM_PRECEDENCE_FAMILY*, délimitée par un ':'.

```
<context-param>
    <param-name>resteasy.interceptor.before.precedence</param-name>
    <param-value>DEFAULT_PRECEDENCE_FAMILY :
CUSTOM_PRECEDENCE_FAMILY</param-value>
</context-param>
```

Exemple 13.10. resteasy.interceptor.after.precedence

Le paramètre de contexte **`resteasy.interceptor.after.precedence`** définit la famille de précédence par défaut exécutée après la famille personnalisée. La valeur du paramètre est sous la forme *DEFAULT_PRECEDENCE_FAMILY*/*CUSTOM_PRECEDENCE_FAMILY*, délimitée par un **`:`**.

```
<context-param>
    <param-name>resteasy.interceptor.after.precedence</param-name>
    <param-value>DEFAULT_PRECEDENCE_FAMILY :
CUSTOM_PRECEDENCE_FAMILY</param-value>
</context-param>
```

Les familles de précédence sont appliquées aux intercepteurs à l'aide de l'annotation @Precedence. Pour la liste de famille de précédence par défaut, reportez-vous à : Section 13.9.4.1, « Familles de précédence d'intercepteur ».

Rapporter un bogue

13.10. ANNOTATIONS BASÉES CHAÎNE

13.10.1. Conversion des annotations basées @*Param en objects

Les annotations **`@*Param`** JAX-RS,@PathParam et @FormParam incluses, sont représentées par des chaînes dans une requête HTTP brut. Ces types de paramètres injectés peuvent être convertis en objets si ces objets ont une méthode valueOf(String) statique ou un constructeur qui prend un paramètre String.

RESTEasy fournit deux interfaces **`@Provider`** commerciales pour effectuer cette conversion pour les classes qui n'ont ni de méthode **`valueOf(String)`** statique, ni de constructeur de string.

Exemple 13.11. StringConverter

L'interface StringConverter est mise en place afin de fournir un marshalling de chaîne personnalisée. Elle est inscrite dans resteasy.providers context-param du fichier web.xml. Elle peut également être enregistrée manuellement en appelant la méthode ResteasyProviderFactory.addStringConverter().

L'exemple suivant est un simple exemple de StringConverter.

```
import org.jboss.resteasy.client.ProxyFactory;
import org.jboss.resteasy.spi.StringConverter;
import org.jboss.resteasy.test.BaseResourceTest;
import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;

import javax.ws.rs.HeaderParam;
import javax.ws.rs.MatrixParam;
import javax.ws.rs.PUT;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.QueryParam;
import javax.ws.rs.ext.Provider;

public class StringConverterTest extends BaseResourceTest
```

```
{
  public static class POJO
  {
     private String name;

     public String getName()
     {
 return name;
     }

     public void setName(String name)
     {
 this.name = name;
     }
  }

  @Provider
  public static class POJOConverter implements StringConverter<POJO>
  {
     public POJO fromString(String str)
     {
 System.out.println("FROM STRNG: " + str);
 POJO pojo = new POJO();
 pojo.setName(str);
 return pojo;
     }

     public String toString(POJO value)
     {
 return value.getName();
     }
  }

  @Path("/")
  public static class MyResource
  {
     @Path("{pojo}")
     @PUT
     public void put(@QueryParam("pojo")POJO q, @PathParam("pojo")POJO
pp,
       @MatrixParam("pojo")POJO mp, @HeaderParam("pojo")POJO hp)
     {
 Assert.assertEquals(q.getName(), "pojo");
 Assert.assertEquals(pp.getName(), "pojo");
 Assert.assertEquals(mp.getName(), "pojo");
 Assert.assertEquals(hp.getName(), "pojo");
     }
  }

  @Before
  public void setUp() throws Exception
  {

dispatcher.getProviderFactory().addStringConverter(POJOConverter.class);
     dispatcher.getRegistry().addPerRequestResource(MyResource.class);
  }
```

```
  @Path("/")
  public static interface MyClient
  {
     @Path("{pojo}")
     @PUT
     void put(@QueryParam("pojo")POJO q, @PathParam("pojo")POJO pp,
       @MatrixParam("pojo")POJO mp, @HeaderParam("pojo")POJO hp);
  }

  @Test
  public void testIt() throws Exception
  {
     MyClient client = ProxyFactory.create(MyClient.class,
"http://localhost:8081");
     POJO pojo = new POJO();
     pojo.setName("pojo");
     client.put(pojo, pojo, pojo, pojo);
  }
}
```

Exemple 13.12. StringParameterUnmarshaller

L'interface **`StringParameterUnmarshaller`** est sensible aux annotations placées sur le paramètre ou le champ sur lesquels vous injectez. Il est créé par injecteur. La méthode setAnnotations() est appelée par resteasy pour initialiser l'unmarshaller.

Cette interface peut être ajoutée en créant et en enregistrant un fournisseur qui implémente l'interface. Il peut être également lié à une meta-annotations appelée **`org.jboss.resteasy.annotations.StringsParameterUnmarshallerBinder`**.

L'exemple ci-dessous formate un **`java.util.Date`** basé @PathParam.

```
public class StringParamUnmarshallerTest extends BaseResourceTest
{
   @Retention(RetentionPolicy.RUNTIME)
   @StringParameterUnmarshallerBinder(DateFormatter.class)
   public @interface DateFormat
   {
      String value();
   }

   public static class DateFormatter implements
StringParameterUnmarshaller<Date>
   {
      private SimpleDateFormat formatter;

      public void setAnnotations(Annotation[] annotations)
      {
         DateFormat format = FindAnnotation.findAnnotation(annotations,
DateFormat.class);
         formatter = new SimpleDateFormat(format.value());
      }
```

```
      public Date fromString(String str)
      {
         try
         {
            return formatter.parse(str);
         }
         catch (ParseException e)
         {
            throw new RuntimeException(e);
         }
      }
   }

   @Path("/datetest")
   public static class Service
   {
      @GET
      @Produces("text/plain")
      @Path("/{date}")
      public String get(@PathParam("date") @DateFormat("MM-dd-yyyy")
Date date)
      {
         System.out.println(date);
         Calendar c = Calendar.getInstance();
         c.setTime(date);
         Assert.assertEquals(3, c.get(Calendar.MONTH));
         Assert.assertEquals(23, c.get(Calendar.DAY_OF_MONTH));
         Assert.assertEquals(1977, c.get(Calendar.YEAR));
         return date.toString();
      }
   }

   @BeforeClass
   public static void setup() throws Exception
   {
      addPerRequestResource(Service.class);
   }

   @Test
   public void testMe() throws Exception
   {
      ClientRequest request = new
ClientRequest(generateURL("/datetest/04-23-1977"));
      System.out.println(request.getTarget(String.class));
   }
}
```

Définit une nouvelle annotation appelée @DateFormat. L'annotation est annotée avec la meta-annotation StringParameterUnmarshallerBinder avec une référence aux classes DateFormater.

La méthode Service.get() possède un paramètre @PathParam qui est aussi annoté avec @DateFormat. L'application @DateFormat déclenche la liaison du DateFormatter. Le DateFormatter sera désormais exécuté pour supprimer le marshalling du paramètre de chemin d'accès dans le paramètre date de la méthode get().

Rapporter un bogue

13.11. CONFIGURATION DES EXTENSIONS DE FICHIERS

13.11.1. Mapper les extensions de fichiers avec les types de media dans le fichier web.xml

Résumé

Certains clients, comme les navigateurs, ne peuvent pas utiliser les en-têtes Accept et Accept-Language pour négocier le type de média de la représentation ou la langue. RESTEasy peut mapper des suffixes de noms de fichier aux langues et types de médias pour régler ce problème. Procédez comme suit pour mapper les types de media avec les extensions de fichier, dans le fichier **`web.xml`**

Procédure 13.4. Mapper des types de media avec les extensions de fichiers

1. Ouvrir le fichier **`web.xml`** de l'application dans un éditeur de texte.
2. Ajouter le paramètre de contexte **`resteasy.media.type.mappings`** au fichier, dans les balises **`web-app`** :

```
<context-param>
    <param-name>resteasy.media.type.mappings</param-name>
</context-param>
```

3. Configurer les valeurs des paramètres. Les mappages forment une liste délimitée par des virgules. Chaque mappage est délimité par un **`:`**:

Exemple 13.13. Exemple de mappage

```
<context-param>
    <param-name>resteasy.media.type.mappings</param-name>
    <param-value>html : text/html, json : application/json, xml :
application/xml</param-value>
</context-param>
```

Rapporter un bogue

13.11.2. Mapper les extensions de fichiers vers les langues dans le fichier web.xml

Résumé

Certains clients, comme les navigateurs, ne peuvent pas utiliser les en-têtes Accept et Accept-Language pour négocier le type de média de la représentation ou la langue. RESTEasy peut mapper des suffixes de noms de fichiers aux langues et aux types de médias pour régler ce problème. Procédez comme suit pour mapper les langues avec les extensions de fichier, dans le fichier **`web.xml`**

Procédure 13.5. Mapper les extensions de fichiers vers les langues dans le fichier web.xml

1. Ouvrir le fichier **`web.xml`** de l'application dans un éditeur de texte.

2. Ajouter le paramètre de contexte **`resteasy.language.mappings`** au fichier, dans les balises **`web-app`** :

```
<context-param>
    <param-name>resteasy.language.mappings</param-name>
</context-param>
```

3. Configurer les valeurs des paramètres. Les mappages forment une liste délimitée par des virgules. Chaque mappage est délimité par un `:` :

Exemple 13.14. Exemple de mappage

```
<context-param>
    <param-name>resteasy.language.mappings</param-name>
    <param-value> en : en-US, es : es, fr : fr</param-name>
</context-param>
```

Rapporter un bogue

13.11.3. Types de media supportés par RESTEasy

Tableau 13.5. Types de media

Type de media	Type Java
application/*+xml, text/*+xml, application/*+json, application/*+fastinfoset, application/atom+*	Classes annotées JaxB
application/*+xml, text/*+xml	org.w3c.dom.Document
/	java.lang.String
/	java.io.InputStream
text/brut	primitives, java.lang.String, ou tout type qui possède un contructeur de String, ou une méthode valueOf(String) pour les entrées et, toString() pour les sorties
/	javax.activation.DataSource
/	byte[]
/	java.io.File
application/x-www-form-urlencoded	javax.ws.rs.core.MultivaluedMap

Rapporter un bogue

13.12. API JAVASCRIPT RESTEASY

13.12.1. API JavaScript RESTEasy

RESTEasy peut générer un API JavaScript qui utilise les appels AJAX pour invoquer les opérations JAX-RS. Chaque classe de ressource JAX-RS va générer un objet JavaScript du même nom que la classe ou l'interface qui déclare. L'objet JavaScript contient chaque méthode JAX-RS sous forme de propriété.

Exemple 13.15. Simple exemple d'API JavaScript JAX-RS

```
@Path("/")
public interface X{
 @GET
 public String Y();
 @PUT
 public void Z(String entity);
}
```

L'interface ci-dessus définit les méthodes Y et Z, qui deviennent des propriétés dans l'API JavaScript, comme ci-dessous :

```
var X = {
 Y : function(params){...},
 Z : function(params){...}
};
```

Chaque méthode d'API JavaScript prend un objet optionnel comme simple paramètre avec chaque propriété comme cookie, en-tête, chemin d'accès, requête ou paramètre de formulaire identifiés par un nom, ou les propriétés de paramètre de l'API. Les propriétés disponibles sont les suivantes : Section 13.12.3, « Paramètres de l'API JavaScript RESTEasy ».

Rapporter un bogue

13.12.2. Activation du Servlet API JavaScript RESTEasy

Résumé

L'API JavaScript RESTEasy n'est pas activé par défaut. Suivre les étapes suivantes en utilisant le fichier **`web.xml`**.

Procédure 13.6. Modifier web.xml pour activer l'API RESTEasy JavaScript

1. Ouvrir le fichier **`web.xml`** de l'application dans l'éditeur de textes.
2. Ajouter la configuration suivante dans le fichier, dans les balises **`web-app`** :

```
<servlet>
    <servlet-name>RESTEasy JSAPI</servlet-name>
    <servlet-class>org.jboss.resteasy.jsapi.JSAPIServlet</servlet-
class>
</servlet>

<servlet-mapping>
```

```
    <servlet-name>RESTEasy JSAPI</servlet-name>
    <url-pattern>/URL</url-pattern>
</servlet-mapping>
```

Rapporter un bogue

13.12.3. Paramètres de l'API JavaScript RESTEasy

Tableau 13.6. Propriétés des paramètres

Property	Valeur par défaut	Description
$entity		L'entité pour envoyer une requête PUT ou POST.
$contentType		Type MIME de l'entité de contenu envoyée comme en-tête de type de contenu ou Content-Type. Déterminé par l'annotation @Consumes.
$accepts	*/*	Types MIME de l'entité envoyés comme en-tête de Accept. Déterminé par l'annotation @Provides.
$callback		Définir à une fonction (httpCode, xmlHttpRequest, value) pour un appel asynchrone. Si non présent, l'appel sera synchronisé et renverra une valeur.
$apiURL		Définit la base de l'URI du point de terminaison JAX-RS, sans la dernière barre oblique.
$username		Si le nom d'utilisateur ou le mot de passe sont définis, ils seront utilisés comme informations d'authentification pour la demande.
$password		Si le nom d'utilisateur ou le mot de passe sont définis, ils seront utilisés comme informations d'authentification pour la demande.

Rapporter un bogue

13.12.4. Créer des requêtes AJAX par l'API JavaScript

Résumé

L'API JavaScript RESTEasy peut être utilisé pour créer des requêtes manuellement. Cette section couvre des exemples de ce comportement.

Exemple 13.16. L'Objet REST

L'objet REST peut être utilisé pour remplacer le comportement du client API JavaScript RESTEasy :

```
// Change the base URL used by the API:
REST.apiURL = "http://api.service.com";

// log everything in a div element
REST.log = function(text){
 jQuery("#log-div").append(text);
};
```

L'objet RESTEasy contient les propriétés lecture-écriture suivantes :

apiURL

Défini par défaut à l'URL root JAX-RS. Utilisé par chaque fonction d'API Client JavaScript lors de la création de requêtes.

log

Définir à une fonction(string) pour pouvoir recevoir les logs d'API Client RESTEasy. Cela est utile si vous souhaitez déboguer votre API Client ou mettre les logs là où vous pouvez les voir.

Exemple 13.17. La classe REST.Request

La classe REST.Request peut être utilisée pour créer des requêtes personnalisées :

```
var r = new REST.Request();
r.setURI("http://api.service.com/orders/23/json");
r.setMethod("PUT");
r.setContentType("application/json");
r.setEntity({id: "23"});
r.addMatrixParameter("JSESSIONID", "12309812378123");
r.execute(function(status, request, entity){
 log("Response is "+status);
});
```

Rapporter un bogue

13.12.5. Membres de la classe REST.Request

Tableau 13.7. Classe REST.Request

Membre	Description

Membre	Description
execute(callback)	Exécute la requête avec toutes les informations définies pour l'objet en cours. La valeur est passée à la fonction de rappel de l'argument optionnel, et n'est pas renvoyée.
setAccepts(acceptHeader)	Définit l'en-tête de la requête Accept. Par défaut */*.
setCredentials(username, password)	Définit les informations d'identification de la requête.
setEntity(entity)	Définit les informations d'identification de l'entité.
setContentType(contentTypeHeader)	Définit l'en-tête de la requête Type de contenu
setURI(uri)	Définit l'URI de la requête. Doit correspondre à un URI absolu.
setMethod(method)	Définit la méthode de requête. GET par défaut.
setAsync(async)	Contrôle si la requête doit être asynchrone. True par défaut.
addCookie(name, value)	Définit le cookie donnée dans le document en cours quand on exécute la requête. Sera persistant dans le navigateur.
addQueryParameter(name, value)	Ajoute un paramètre d'interrogation à la partie de requête URI.
addMatrixParameter(name, value)	Ajoute un paramètre de matrix (paramètre de chemin d'accès) au dernier segment de l'URI de la requête.
addHeader(nom, valeur)	Ajoute un en-tête de requête.

Rapporter un bogue

13.13. SERVICE DE JOB ASYNCHRONE RESTEASY

13.13.1. Service Job Asynchrone RESTEasy

Le Service Job Asynchrone RESTEasy est conçu pour ajouter un comportement asynchrone au protocole HTTP. Alors que HTTP est un protocole synchrone, il possède une faible idée des appels asynchrones. Le code réponse 202 HTTP 1.1 « Accepted » signifie que le serveur a reçu et accepté la réponse pour le traitement, mais que le traitement n'est pas encore terminé. Le Service Job Asynchrone est construit autour de cela.

Pour activer le service, consulter : Section 13.13.2, « Activer le service de jobs asynchrones ». Pour obtenir des informations sur la façon dont le service fonctionne, voir Section 13.13.3, « Configurer les jobs asynchrones avec RESTEasy ».

Rapporter un bogue

13.13.2. Activer le service de jobs asynchrones

Procédure 13.7. Modifier le fichier web.xml

- Activer le service de jobs asynchrones dans le fichier **`web.xml`** :

```
<context-param>
    <param-name>resteasy.async.job.service.enabled</param-name>
    <param-value>true</param-value>
</context-param>
```

Résultat

Le service de jobs asynchrones a été activé. Pour les options de configuration, consulter : Section 13.13.4, « Paramètres de configuration de Service Job Asynchrone ».

Rapporter un bogue

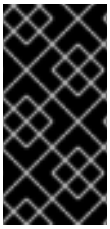
13.13.3. Configurer les jobs asynchrones avec RESTEasy

Résumé

Cette section couvre des exemples de paramètres de recherche de jobs asynchrones avec RESTEasy.

AVERTISSEMENT

La sécurité basée rôle ne fonctionne pas dans le Service de jobs asynchrones car il ne fonctionne pas de manière portable. Si le Service de jobs asynchrones est utilisé, la sécurité d'application doit être effectuée via des déclarations XML par le fichier **`web.xml`** à la place.



IMPORTANT

Bien que les méthodes GET, DELETE, et PUT soient invoquées de manière asynchrones, cela rompt le contrat HTTP 1.1 de ces méthodes. Bien que ces invocations ne changent sans doute pas l'état de la ressource en cas d'invocations multiples, elles changent d'état du serveur en nouvelles entrées de Job pour chaque invocation.

Exemple 13.18. Le paramètre Asynch

Le paramètre de requête **asynch** est utilisé pour exécuter des invocations en arrière-plan. Une réponse 202 Accepted est retournée, ainsi que d'un en-tête d'emplacement avec un URL pointant vers la réponse de la méthode de base.

```
POST http://example.com/myservice?asynch=true
```

L'exemple ci-dessus retournera une réponse Accepted 202. Il retournera également un en-tête d'emplacement avec un URL pointant vers la réponse de la méthode de base. Voici un exemple de l'en-tête d'emplacement :

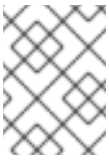
```
HTTP/1.1 202 Accepted
Location: http://example.com/asynch/jobs/3332334
```

L'URI prendra la forme suivante :

```
/asynch/jobs/{job-id}?wait={millisconds}|nowait=true
```

Les opérations GET, POST et DELETE peuvent être effectuées sur cet URL.

- GET renvoie la méthode de ressources JAX-RS invoquée comme réponse si le job est complété. Si le job n'a pas été complété, ce GET renverra un code de réponse 202 Accepted. L'invocation de GET ne supprime pas le job; peut être appelé à plusieurs reprises.
- POST procède à une lecture de la réponse du job et supprime le job s'il est terminé.
- DELETE est appelé pour nettoyer manuellement la file d'attente du job.



NOTE

Quand la file d'attente du job est pleine, il expulsera job le plus ancien de la mémoire automatiquement, sans avoir besoin d'appeler DELETE.

Exemple 13.19. Wait / Nowait

Les opérations GET et POST permettent de définir une durée d'attente maximum, par les paramètres de demande **`wait`** et **`nowait`**. Si le paramètre **`wait`** n'est pas spécifié, l'opération aura par défaut **`nowait=true`**, et n'attendra pas si le job n'est pas terminé. Le paramètre **`wait`** est défini en millisecondes.

```
POST http://example.com/asynch/jobs/122?wait=3000
```

Exemple 13.20. Le Paramètre Oneway

RESTEasy supporte «fire and forget jobs» (abandonner les jobs) avec le paramètre de requête **`oneway`**.

```
POST http://example.com/myservice?oneway=true
```

L'exemple ci-dessus retournera la réponse 202 Accepted, mais aucun job ne sera créé.

Rapporter un bogue

13.13.4. Paramètres de configuration de Service Job Asynchrone

Résumé

Le tableau suivant donne des informations sur la paramètres contextuels configurables du Service Job Asynchrone. Ces paramètres sont configurés dans le fichier **`web.xml`**.

Tableau 13.8. Paramètres de configuration

Paramètre	Description
resteasy.async.job.service.max.job.results	Nombre de résultats de jobs qui peuvent être conservés en toute harmonie à un moment donné. La valeur par défaut est 100.
resteasy.async.job.service.max.wait	Temps d'attente maximum quand un client interroge un job. La valeur par défaut est 300000.
resteasy.async.job.service.thread.pool.size	Taille de thread pool des threads d'arrière-plan qui exécutent le job. La valeur par défaut est 100.
resteasy.async.job.service.base.path	Définit le chemin de base des URI de job. La valeur par défaut est /asynch/jobs

Exemple 13.21. Exemple de configuration de jobs asynchrones

```
<web-app>
    <context-param>
        <param-name>resteasy.async.job.service.enabled</param-name>
        <param-value>true</param-value>
    </context-param>

    <context-param>
        <param-name>resteasy.async.job.service.max.job.results</param-
name>
        <param-value>100</param-value>
    </context-param>
    <context-param>
        <param-name>resteasy.async.job.service.max.wait</param-name>
        <param-value>300000</param-value>
    </context-param>
    <context-param>
        <param-name>resteasy.async.job.service.thread.pool.size</param-
name>
        <param-value>100</param-value>
    </context-param>
    <context-param>
        <param-name>resteasy.async.job.service.base.path</param-name>
        <param-value>/asynch/jobs</param-value>
    </context-param>

    <listener>
        <listener-class>
            org.jboss.resteasy.plugins.server.servlet.ResteasyBootstrap
```

```
        </listener-class>
    </listener>

    <servlet>
        <servlet-name>Resteasy</servlet-name>
        <servlet-class>

org.jboss.resteasy.plugins.server.servlet.HttpServletDispatcher
        </servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>Resteasy</servlet-name>
        <url-pattern>/*</url-pattern>
    </servlet-mapping>

</web-app>
```

Rapporter un bogue

13.14. RESTEASY JAXB

13.14.1. Créer un décorateur JAXB

Résumé

Les fournisseurs JAXB de RESTEasy ont une manière enfichable de décorer des instances Marshaller ou Unmarshaller. Il y a une annotation qui est créée soit par un Marshaller, soit par un Unmarshaller. Cette section couvre toutes les étapes pour créer un décorateur JAXB dans RESTEasy.

Procédure 13.8. Créer un décorateur JAXB dans RESTEasy

1. **Créer la classe de processeur**

   a. Créer une classe qui implémente DecoratorProcessor<Target, Annotation>. La cible sera soit la classe de Marshaller ou Unmarshaller JAXB. L'annotation est créée dans la seconde étape.

   b. Annoter la classe par @DecorateTypes, et déclarer les types MIME que le décorateur doit décorer.

   c. Définir les propriétés ou les valeurs au sein de la fonction **`decorate`**.

Exemple 13.22. Exemple de classe de processeur

```
import org.jboss.resteasy.core.interception.DecoratorProcessor;
import org.jboss.resteasy.annotations.DecorateTypes;

import javax.xml.bind.Marshaller;
import javax.xml.bind.PropertyException;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.Produces;
import java.lang.annotation.Annotation;
```

```
@DecorateTypes({"text/*+xml", "application/*+xml"})
public class PrettyProcessor implements
DecoratorProcessor<Marshaller, Pretty>
{
    public Marshaller decorate(Marshaller target, Pretty
annotation,
   Class type, Annotation[] annotations, MediaType mediaType)
    {
 target.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT,
Boolean.TRUE);
    }
}
```

2. **Créer une annotation**

   a. Créer une interface personnalisée annotée par @Decorator.

   b. Déclarer le processeur ou la cible de l'annotation @Decorator. Le processeur est créé dans la première étape. La cible sera soit la classe Marshaller JAXB ou la classe Unmarshaller JAXB.

Exemple 13.23. Exemple d'annotation

```
import org.jboss.resteasy.annotations.Decorator;

@Target({ElementType.TYPE, ElementType.METHOD,
ElementType.PARAMETER, ElementType.FIELD})
@Retention(RetentionPolicy.RUNTIME)
@Decorator(processor = PrettyProcessor.class, target =
Marshaller.class)
public @interface Pretty {}
```

3. Ajouter l'annotation créée dans une fonction dans la seconde étape de façon à ce que l'entrée ou la sortie soit décorée quand elle sera mise en ordre.

Résultat

Le décorateur JAXB a été créé et sera appliqué au sein du service web JAX-RS.

Rapporter un bogue

13.15. RESTEASY ATOM SUPPORT

13.15.1. Fournissseur et API Atom

Le Fournisseur et l'API Atom RESTEasy est un modèle d'objet simple que RESTEasy définit pour représenter Atom. Les classes principales de l'API se trouvent dans le package **`org.jboss.resteasy.plugins.providers.atom`**. RESTEasy utilise JAXB pour marshaller et démarshaller l'API. Le fournisseur est basé JAXB, et ne se limite pas à envoyer des objets Atom par XML. Tous les fournisseurs JAXB que RESTEasy contient peuvent être réutilisés par le fournisseur et l'API Atom, y compris JSON. Voir Javadocs pour obtenir plus d'informations sur l'API.

Rapporter un bogue

13.16. RESTEASY/SPRING INTEGRATION

13.16.1. Intégration RESTEasy/Spring

Conditions préalables

- Votre application doit posséder un service JAX-WS et une configuration client.

Procédure 13.9. Activer la fonctionnalité d'intégration RESTEasy/Spring

- RESTEasy s'intègre dans Spring 3.0.x.

  Les utilisateurs Maven doivent utiliser l'artefact resteasy-spring. Sinon, le jar sera disponible en tant que module dans JBoss EAP 6.

  RESTEasy est livré avec son propre Spring ContextLoaderListener qui enregistre un BeanPostProcessor spécifique à RESTEasy qui traite les annotations JAX-RS lorsqu'un bean est créé par une BeanFactory. Cela veut dire que RESTEasy recherchera automatiquement les annotations de ressource JAX-RS et @Provider dans votre classe bean et les enregistrera en tant que ressources de JAX-RS.

  Exemple 13.24. Modifier web.xml

  Ajouter ce qui suit au fichier web.xml pour activer la fonctionnalité d'intégration RESTEasy/Spring :

```
<web-app>
 <display-name>
   Archetype Created Web Application
 </display-name>
 <listener>
    <listener-class>
     org.jboss.resteasy.plugins.server.servlet.ResteasyBootstrap
   </listener-class>
 </listener>

 <listener>
   <listener-class>

org.jboss.resteasy.plugins.spring.SpringContextLoaderListener
    </listener-class>
 </listener>

 <servlet>
   <servlet-name>Resteasy
   </servlet-name>
    <servlet-class>

org.jboss.resteasy.plugins.server.servlet.HttpServletDispatcher
    </servlet-class>
 </servlet>
```

```
 <servlet-mapping>
   <servlet-name>
    Resteasy
   </servlet-name>
   <url-pattern>/*</url-pattern>
 </servlet-mapping>
</web-app>
```

Le SpringContextLoaderListener doit être déclaré après ResteasyBootstrap car il utilise les attributs de ServletContext initialisés par celui-ci.

Pour plus d'informations concernant RestEasy et Spring Integration, consulter http://docs.jboss.org/resteasy/docs/2.3.7.Final/userguide/html_single/

Rapporter un bogue

CHAPITRE 14. SERVICES WEB JAX-WS

14.1. SERVICES WEB JAX-WS

API Java pour les Services Web XML (JAX-WS) est une API incluse dans la plate-forme Enterprise Edition (J2EE) et est utilisé pour créer des Services Web. Les Services Web sont des applications conçues pour communiquer entre elles via un réseau, typiquement échangeant des informations en XML ou autres formats de texte structuré. Les Services Web sont indépendants de la plate-forme. Une application typique de JAX-WS utilise un modèle client/serveur. Le composant de serveur s'appelle un *Point de terminaison de Service Web*.

JAX-WS a un équivalent pour les plus petits et plus simples Services Web qui utilisent un protocole appelé JAX-RS. JAX-RS est un protocole de *Representational State Transfer*, ou REST. Les applications JAX-RS sont généralement légères et ne comptent que sur le protocole HTTP proprement dit pour la communication. JAX-WS rend plus facile de soutenir divers protocoles de services web, tels que **`WS-Notification`**, **`WS-Addressing`**, **`WS-Policy`**, **`WS-Security`** et **`WS-Trust`**. Ils communiquent à l'aide d'un XML spécialisé appelé *Simple Object Access Protocol (SOAP)*, qui définit une architecture de message et le message de formats.

Un Service Web JAX-WS inclut également une description lisible par machine des opérations fournies, écrite en *Web Services Description Language (WSDL)*, qui est un type spécialisé de document XML.

Un point de terminaison de service web consiste en une classe qui implémente les interfaces **`WebService`** et **`WebMethod`**.

Un Client de Service Web est un client qui dépend de plusieurs classes appelées des *stubs* générées à partir de la définition WSDL. JBoss EAP 6 comprend les outils pour générer les classes à partir de WSDL.

Dans un Service Web JAX-WS, un contrat formel est créé pour décrire l'interface qui offre le Service Web. Le contrat est généralement écrit en WSDL, mais peut-être être écrit dans les messages SOAP. L'architecture du Service Web aborde généralement les besoins opérationnels, comme les transactions, la sécurité, la messagerie et la coordination. JBoss EAP 6 fournit des mécanismes pour traiter ces besoins commerciaux.

Web Services Description Language (WSDL) est un langage basé XML utilisé pour décrire des Services Web et comment y avoir accès. Le Service Web lui-même est écrit en Java ou autre langage de programmation. La définition WSDL est composée des références à l'interface, des définitions de port et des instructions sur la façon dont les autres Services Web doit interagir avec lui sur un réseau. Les Services Web communiquent entre eux à l'aide de *Simple Object Access Protocol (SOAP)*. Ce type de Service Web contraste avec les *Services Web RESTful* conçus à l'aide du principe *Representative State Transfer (REST)*. Ces Services Web RESTful n'exigent pas l'utilisation de WSDL ou de SOAP, mais reposent sur la structure du protocole HTTP lui-même pour définir comment les autres services interagissent avec eux.

JBoss EAP 6 inclut un support pour le déploiement des points de terminaison du service JAX-WS Web Service. Ce support est donné par JBossWS. La configuration du sous-système de services web, comme la configuration des points de terminaison, les chaînes de gestionnaires, les gestionnaires, est fournie via le sous-système **`webservices`**.

Exemples

Les JBoss EAP Quickstarts incluent plusieurs applications JAX-WS Web Services qui fonctionnent correctement. Exemples :

- wsat-simple

- wsba-coordinator-completion-simple
- wsba-participant-completion-simple

Rapporter un bogue

14.2. CONFIGURER LE SOUS-SYSTÈME DES SERVICES WEB

Il existe de nombreuses options de configuration pour le sous-système de **webservices**, qui contrôle le comportement des Services Web déployés dans JBoss EAP 6. La commande pour modifier chaque élément dans le script de gestion CLI (***EAP_HOME* /bin/jboss-cli.sh** ou ***EAP_HOME* /bin/jboss-cli.bat**) est fournie. Supprimer la partie de la commande **/profile=default** pour les serveurs autonomes, ou remplacer **default** par le nom de profil à configurer.

Adresse de point de terminaison publiée

Vous pouvez écrire à nouveau l'élément **<soap:address>** dans les contrats WSDL endpoint-published. Cette solution peut être utilisée pour contrôler l'adresse du serveur qui est publiée aux clients pour chaque point de terminaison. Chacun des éléments suivants en option peuvent être modifiés pour satisfaire vos besoins. La modification de ces éléments exige un redémarrage du serveur.

Tableau 14.1. Éléments de configuration pour les adresses de points de terminaison publiés.

Élément	Description	Commande CLI
modify-wsdl-address	Indique s'il faut toujours modifier l'adresse WSDL. Si true, le contenu de **< adresse:soap >** sera toujours remplacé. Si false, le contenu de **< adresse:soap >** sera remplacé seulement si ce n'est pas une URL valide. Les valeurs utilisées seront **wsdl-host**, **wsdl-port**, et **wsdl-secure-port** décrit ci-dessous.	**/profile=default/subsystem=webservices/:write-attribute(name=modify-wsdl-address,value=true)**
wsdl-host	Le nom d'hôte / adresse IP à utiliser pour écrire à nouveau **<soap:address>**. Si **wsdl-host** est défini au string **jbossws.undefined.host**, l'hôte du demandeur sera utilisé quand on écrit à nouveau une **<soap:address>**.	**/profile=default/subsystem=webservices/:write-attribute(name=wsdl-host,value=10.1.1.1)**
wsdl-port	Entier relatif qui définit explicitement le port HTTP qui sera utilisé pour écrire à nouveau l'adresse SOAP. Si non défini, le port HTTP peut être identifié en cherchant la liste de connecteurs HTTP installés.	**/profile=default/subsystem=webservices/:write-attribute(name=wsdl-port,value=8080)**

Élément	Description	Commande CLI
wsdl-secure-port	Entier relatif qui définit explicitement le port HTTPS qui sera utilisé pour écrire à nouveau l'adresse SOAP. Si non défini, le port HTTPS peut être identifié en cherchant la liste de connecteurs HTTPS installés.	**`/profile=default/subsystem=webservices/:write-attribute(name=wsdl-secure-port,value=8443)`**

Configurations de point de terminaison prédéfinis

Vous pouvez définir des configurations de points de terminaison qui peuvent être référencées par les implémentations de points de terminaison. Une des façons dont cela puisse être utilisé consiste à ajouter un gestionnaire donné à n'importe quel point de terminaison WS, pour lequel il est indiqué une configuration de point de terminaison donnée avec l'annotation **`@org.jboss.ws.api.annotation.EndpointConfig`**.

JBoss EAP 6 inclut une **`Standard-Endpoint-Config`** par défaut. Il existe également une configuration **`Recording-Endpoint-Config`** personnalisée, également incluse. Cela vous donne un exemple de gestionnaire d'enregistrement. La **`Standard-Endpoint-Config`** est utilisée automatiquement pour tout point de terminaison non associé à une autre configuration.

Pour lire la **`Standard-Endpoint-Config`** par l'interface CLI, il suffit d'utiliser la commande suivante :

```
/profile=default/subsystem=webservices/endpoint-config=Standard-Endpoint-Config/:read-resource(recursive=true,proxies=false,include-runtime=false,include-defaults=true)
```

Configurations des points de terminaison

La configuration d'un point de terminaison, à laquelle on fait référence ainsi **`endpoint-config`** dans l'API de gestion, inclut **`pre-handler-chain`**, **`post-handler-chain`** et quelques propriétés qui sont appliquées à un point de terminaison particulier. Les commandes suivantes lisent et ajoutent un point de config.

Exemple 14.1. Lecture d'une config de point de terminaison

```
/profile=default/subsystem=webservices/endpoint-config=Recording-Endpoint-Config:read-resource
```

Exemple 14.2. Ajout d'une config de point de terminaison

```
/profile=default/subsystem=webservices/endpoint-config=My-Endpoint-Config:add
```

Chaînes de gestionnaires

Chaque config de point de terminaison peut être associée à des chaînes de gestionnaires **PRE** ou **POST**. Chaque chaîne de gestionnaire peut inclure des gestionnaires conformes à JAXWS. Pour les messages sortants, les gestionnaires de chaînes de gestionnaires PRE sont exécutés avant tout gestionnaire

attaché aux points de terminaison, à l'aide de moyens JAXWS standards, comme avec l'annotation **@HandlerChain**. Les gestionnaires de chaînes de POST handler sont exécutés après les gestionnaires de points de terminaison habituels. Pour les messages entrants, c'est l'opposé. JAX-WS est une API standard pour les services basés XML, et est documentée à l'adresse suivante http://jcp.org/en/jsr/detail?id=224.

Une chaîne de gestionnaire peut aussi inclure un attribut **protocol-bindings**, qui définit les protocoles qui déclenchent le démarrage de la chaîne.

Exemple 14.3. Lecture d'une chaîne de gestionnaire

```
/profile=default/subsystem=webservices/endpoint-config=Recording-
Endpoint-Config/pre-handler-chain=recording-handlers:read-resource
```

Exemple 14.4. Ajouter un chaîne de gestionnaire

```
/profile=default/subsystem=webservices/endpoint-config=My-Endpoint-
Config/post-handler-chain=my-handlers:add(protocol-
bindings="##SOAP11_HTTP")
```

Gestionnaires

Un gestionnaire JAXWS est un **handler** (gestionnaire) d'élément dépendant, qui se trouve à l'intérieur d'une chaîne de gestionnaire. Le gestionnaire prend un attribut de **classe**, qui est le nom de classe complet de la classe du gestionnaire. Quand le point de terminaison est déployé, une instance de cette classe sera créée pour chaque déploiement référençant. Le chargeur de classes de déploiement ou le chargeur de classes du module **org.jboss.as.webservices.server.integration** doivent pouvoir charger la classe de gestionnaire.

Exemple 14.5. Lecture d'un gestionnaire

```
/profile=default/subsystem=webservices/endpoint-config=Recording-
Endpoint-Config/pre-handler-chain=recording-
handlers/handler=RecordingHandler:read-resource
```

Exemple 14.6. Ajout d'un gestionnaire

```
/profile=default/subsystem=webservices/endpoint-config=My-Endpoint-
Config/post-handler-chain=my-handlers/handler=foo-
handler:add(class="org.jboss.ws.common.invocation.RecordingServerHandler
")
```

Informations de runtime des services web

Vous pouvez afficher des informations d'exécution sur les Services Web, tels que le contexte web et l'URL WSDL, en interrogeant les points de terminaison eux-mêmes. Vous pouvez utiliser le caractère * pour interroger tous les points de terminaison à la fois. Les deux exemples suivants montrent la commande à la fois pour un serveur dans un domaine géré, et pour un serveur autonome.

Exemple 14.7. Voir les informations de runtime sur tous les points de terminaison d'un serveur web dans un domaine géré.

Cette commande affiche les informations sur tous les points de terminaison sur le serveur nommé **server-one** qui se trouve sur l'hôte physique **master** et qui exécute sur un domaine géré.

```
/host=master/server=server-
one/deployment="*"/subsystem=webservices/endpoint="*":read-resource
```

Exemple 14.8. Voir les informations de runtime sur tous les points de terminaison d'un serveur web dans un domaine autonome.

Cette commande affiche des informations sur tous les points de terminaison de services web sur un serveur autonome.

```
/deployment="*"/subsystem=webservices/endpoint="*":read-resource
```

Exemple 14.9. Exemple d'information de point de terminaison

L'exemple suivant est une sortie hypothétique.

```
{
   "outcome" => "success",
   "result" => [{
       "address" => [
           ("deployment" => "jaxws-samples-handlerchain.war"),
           ("subsystem" => "webservices"),
           ("endpoint" => "jaxws-samples-handlerchain:TestService")
       ],
       "outcome" => "success",
       "result" => {
           "class" =>
"org.jboss.test.ws.jaxws.samples.handlerchain.EndpointImpl",
           "context" => "jaxws-samples-handlerchain",
           "name" => "TestService",
           "type" => "JAXWS_JSE",
           "wsdl-url" => "http://localhost:8080/jaxws-samples-
handlerchain?wsdl"
       }
   }]
}
```

Rapporter un bogue

14.3. PONTS DE TERMINAISON DE SERVICES WEB JAX-WS

14.3.1. Les points de terminaison de Services Web JAX-WS

Cette section est une vue d'ensemble des points de terminaison de service JAX-WS web et des

concepts qui l'accompagnent. Un point de terminaison de Service Web JAX-WS est le composant de serveur d'un Service Web. Les clients et les autres Services Web le communiquent via le protocole HTTP à l'aide d'un langage XML appelé *Simple Object Access Protocol (SOAP)*. Le point de terminaison lui-même est déployé dans le conteneur JBoss EAP 6.

Les descripteurs WSDL peuvent être créés de plusieurs façons :

1. Vous pouvez écrire les descripteurs WSDL manuellement.
2. Si vous utilisez les annotations de JAX-WS qui créent les descripteurs WSDL automatiquement pour vous. Il s'agit de la méthode la plus commune pour créer des descripteurs WSDL.

Un bean d'implémentation de point de terminaison est annoté d'annotations JAX-WS et est déployé sur le serveur. Le serveur génère et publie automatiquement le contrat abstrait en format WSDL pour la consommation client. Tous les marshalling et unmarshalling sont délégués au service *Architecture Java pour XML Binding (JAXB)*.

Le point de terminaison peut être un POJO (Plain Old Java Object) ou une Application Web Java EE. Vous pouvez également exposer des points de terminaison à l'aide d'un bean de session stateless EJB3. Il est empaqueté dans un fichier d'Archive Web (WAR). La spécification d'emballage de point de terminaison, appelée *Java Service Endpoint (JSE)* est définie dans JSR-181, qui se trouve dans http://jcp.org/aboutJava/communityprocess/mrel/jsr181/index2.html.

Conditions préalables de développement

Un service web doit se conformer aux conditions préalables de l'API JAXWS et à la spécification des métadonnées des services web qui se trouvent dans http://www.jcp.org/en/jsr/summary?id=181. Une implémentation valide devra remplir les critères suivants :

- Contenir une annotation **`javax.jws.WebService`**.
- Tous les paramètres de méthode et les types de renvoi doivent être compatibles avec la spécification JAXB 2.0, JSR-222. Voir http://www.jcp.org/en/jsr/summary?id=222 pour plus d'informations.

Exemple 14.10. Exemple de point de terminaison POJO

```
@WebService
@SOAPBinding(style = SOAPBinding.Style.RPC)
public class JSEBean01
{
   @WebMethod
   public String echo(String input)
   {
      ...
   }
}
```

Exemple 14.11. Exemple de point de terminaison de Services Web

```
<web-app ...>
  <servlet>
    <servlet-name>TestService</servlet-name>
    <servlet-
class>org.jboss.test.ws.jaxws.samples.jsr181pojo.JSEBean01</servlet-
```

```
class>
  </servlet>
  <servlet-mapping>
    <servlet-name>TestService</servlet-name>
    <url-pattern>/*</url-pattern>
  </servlet-mapping>
</web-app>
```

Exemple 14.12. Exposer un point de terminaison dans un EJB

Le bean de session stateless EJB3 expose la même méthode sur l'interface éloignée et en tant qu'opération de point de terminaison.

```
@Stateless
@Remote(EJB3RemoteInterface.class)
@RemoteBinding(jndiBinding = "/ejb3/EJB3EndpointInterface")

@WebService
@SOAPBinding(style = SOAPBinding.Style.RPC)
public class EJB3Bean01 implements EJB3RemoteInterface
{
   @WebMethod
   public String echo(String input)
   {
      ...
   }
}
```

Fournisseurs de points de terminaison

Les services JAX-WS implémentent généralement une interface de point de terminaison de service Java (SEI), qui peut être mappée à partir d'un type de port WSDL, soit directement ou à l'aide d'annotations. Ce SEI fournit une abstraction de haut niveau qui masque les détails entre les objets Java et leurs représentations XML. Toutefois, dans certains cas, les services doivent pouvoir opérer au niveau du message XML. L'interface de point de terminaison **`Provider`** (fournisseur) fournit cette fonctionnalité aux Services Web qui l'implémentent.

Consommer et accéder au point de terminaison.

Une fois que vous aurez déployé votre Service Web, vous pourrez consommer le WSDL pour créer les stubs de composants qui seront à la base de votre application. Votre application pourra alors accéder au point de terminaison et faire son travail.

Exemples

Les JBoss EAP Quickstarts incluent plusieurs applications JAX-WS Web Services qui fonctionnent correctement. Exemples :

- wsat-simple
- wsba-coordinator-completion-simple

- wsba-participant-completion-simple

Rapporter un bogue

14.3.2. Écrire et déployer un point de terminaison de Service Web JAX-WS

Introduction

Cette rubrique décrit le développement d'un simple point de terminaison du service JAX-WS, le composant côté serveur qui répond aux demandes des clients JAX-WS et publie la définition WSDL pour lui-même. Pour plus d'informations sur les points de terminaison de service JAX-WS, consulter Section 14.5.2, « Référence API Commun JAX-WS » et la documentation de l'API en format Javadoc distribuée dans JBoss EAP 6.

Pré-requis de développement

Un service web doit se conformer aux pré-requis de l'API JAXWS et à la spécification des métadonnées des services web qui se trouvent dans http://www.jcp.org/en/jsr/summary?id=181. Une implémentation valide devra remplir les critères suivants :

- Contenir une annotation **`javax.jws.WebService`**.
- Tous les paramètres de méthode et les types de renvoi doivent être compatibles avec la spécification JAXB 2.0, JSR-222. Voir http://www.jcp.org/en/jsr/summary?id=222 pour plus d'informations.

Exemple 14.13. Exemple d'implémentation de service

```
package org.jboss.test.ws.jaxws.samples.retail.profile;

import javax.ejb.Stateless;
import javax.jws.WebService;
import javax.jws.WebMethod;
import javax.jws.soap.SOAPBinding;

@Stateless
@WebService(
   name="ProfileMgmt",
   targetNamespace = "http://org.jboss.ws/samples/retail/profile",
   serviceName = "ProfileMgmtService")
@SOAPBinding(parameterStyle = SOAPBinding.ParameterStyle.BARE)
public class ProfileMgmtBean {

   @WebMethod
   public DiscountResponse getCustomerDiscount(DiscountRequest request)
{
      return new DiscountResponse(request.getCustomer(), 10.00);
   }
}
```

Exemple 14.14. Exemple de charge XML

Ce qui suit est un exemple de classe **DiscountRequest** utilisée par le bean **ProfileMgmtBean** dans l'exemple précédent. Les annotations sont incluses par souci de verbosité. Normalement, les valeurs par défaut JAXB sont raisonnables et n'ont pas besoin d'être spécifiées.

```
package org.jboss.test.ws.jaxws.samples.retail.profile;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlType;

import org.jboss.test.ws.jaxws.samples.retail.Customer;

@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(                                                    (1)
  name = "discountRequest",
  namespace="http://org.jboss.ws/samples/retail/profile",
  propOrder = { "customer" }
)
public class DiscountRequest {

   protected Customer customer;

   public DiscountRequest() {
   }

   public DiscountRequest(Customer customer) {
      this.customer = customer;
   }

   public Customer getCustomer() {
      return customer;
   }

   public void setCustomer(Customer value) {
      this.customer = value;
   }

}
```

Pour les mappages plus complexe possibles, voir la spécification de l'API JAXB à l'adresse suivante https://jaxb.java.net/ pour plus d'information.

Packager votre déploiement

La classe d'implémentation est encapsulée dans un déploiement de **JAR**. Toutes les métadonnées requises pour le déploiement proviennent des annotations qui se trouvent sur la classe d'implémentation et l'interface de point de terminaison de service. Déployer le JAR à l'aide du Management CLI ou de l'interface de gestion, et le point de terminaison HTTP sera créé automatiquement.

Le listing suivant vous montre un exemple de structure qui convient pour un développement JAR de Service Web EJB.

Exemple 14.15. Exemple de structure JAR pour un déploiement de service web

```
[user@host ~]$ jar -tf jaxws-samples-retail.jar
org/jboss/test/ws/jaxws/samples/retail/profile/DiscountRequest.class
org/jboss/test/ws/jaxws/samples/retail/profile/DiscountResponse.class
org/jboss/test/ws/jaxws/samples/retail/profile/ObjectFactory.class
org/jboss/test/ws/jaxws/samples/retail/profile/ProfileMgmt.class
org/jboss/test/ws/jaxws/samples/retail/profile/ProfileMgmtBean.class
org/jboss/test/ws/jaxws/samples/retail/profile/ProfileMgmtService.class
org/jboss/test/ws/jaxws/samples/retail/profile/package-info.class
```

Rapporter un bogue

14.4. CLIENTS DU SERVICE JAX-WS WEB

14.4.1. Consommer et accéder à un Service Web JAX-WS

Après avoir créé un point de terminaison de Service Web, soit manuellement ou à l'aide d'annotations de JAX-WS, vous pouvez accéder à son WSDL, qui peut être utilisé pour créer l'application client de base qui communiquera avec le Service Web. Le processus de génération de code Java de la publication WSDL est appelé consommer le service Web. Cela se produit en deux phases :

1. Créer les artefacts client.
2. Construire un service stub.
3. Accéder au point de terminaison.

Créer les artefacts client

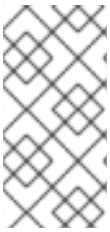
Avant de pouvoir créer des artefacts client, vous devrez créer votre contrat WSDL. Le contrat WSDL suivant est utilisé dans les exemples présentés dans le reste de cette section.

Exemple 14.16. Exemple de contrat WSDL

```
<definitions
    name='ProfileMgmtService'
    targetNamespace='http://org.jboss.ws/samples/retail/profile'
    xmlns='http://schemas.xmlsoap.org/wsdl/'
    xmlns:ns1='http://org.jboss.ws/samples/retail'
    xmlns:soap='http://schemas.xmlsoap.org/wsdl/soap/'
    xmlns:tns='http://org.jboss.ws/samples/retail/profile'
    xmlns:xsd='http://www.w3.org/2001/XMLSchema'>

   <types>

      <xs:schema targetNamespace='http://org.jboss.ws/samples/retail'
                 version='1.0'
xmlns:xs='http://www.w3.org/2001/XMLSchema'>
         <xs:complexType name='customer'>
            <xs:sequence>
               <xs:element minOccurs='0' name='creditCardDetails'
type='xs:string'/>
               <xs:element minOccurs='0' name='firstName'
type='xs:string'/>
```

```
               <xs:element minOccurs='0' name='lastName'
type='xs:string'/>
            </xs:sequence>
         </xs:complexType>
      </xs:schema>

      <xs:schema
          targetNamespace='http://org.jboss.ws/samples/retail/profile'
          version='1.0'
          xmlns:ns1='http://org.jboss.ws/samples/retail'
          xmlns:tns='http://org.jboss.ws/samples/retail/profile'
          xmlns:xs='http://www.w3.org/2001/XMLSchema'>

         <xs:import namespace='http://org.jboss.ws/samples/retail'/>
         <xs:element name='getCustomerDiscount'
                     nillable='true' type='tns:discountRequest'/>
         <xs:element name='getCustomerDiscountResponse'
                     nillable='true' type='tns:discountResponse'/>
         <xs:complexType name='discountRequest'>
            <xs:sequence>
               <xs:element minOccurs='0' name='customer'
type='ns1:customer'/>

            </xs:sequence>
         </xs:complexType>
         <xs:complexType name='discountResponse'>
            <xs:sequence>
               <xs:element minOccurs='0' name='customer'
type='ns1:customer'/>
               <xs:element name='discount' type='xs:double'/>
            </xs:sequence>
         </xs:complexType>
      </xs:schema>

   </types>

   <message name='ProfileMgmt_getCustomerDiscount'>
      <part element='tns:getCustomerDiscount'
name='getCustomerDiscount'/>
   </message>
   <message name='ProfileMgmt_getCustomerDiscountResponse'>
      <part element='tns:getCustomerDiscountResponse'
            name='getCustomerDiscountResponse'/>
   </message>
   <portType name='ProfileMgmt'>
      <operation name='getCustomerDiscount'
                 parameterOrder='getCustomerDiscount'>

         <input message='tns:ProfileMgmt_getCustomerDiscount'/>
         <output
message='tns:ProfileMgmt_getCustomerDiscountResponse'/>
      </operation>
   </portType>
   <binding name='ProfileMgmtBinding' type='tns:ProfileMgmt'>
      <soap:binding style='document'
                    transport='http://schemas.xmlsoap.org/soap/http'/>
```

```
      <operation name='getCustomerDiscount'>
         <soap:operation soapAction=''/>
         <input>

            <soap:body use='literal'/>
         </input>
         <output>
            <soap:body use='literal'/>
         </output>
      </operation>
   </binding>
   <service name='ProfileMgmtService'>
      <port binding='tns:ProfileMgmtBinding' name='ProfileMgmtPort'>

         <soap:address
             location='SERVER:PORT/jaxws-samples-
retail/ProfileMgmtBean'/>
      </port>
   </service>
</definitions>
```



NOTE

Si vous utilisez les annotations de JAX-WS pour créer votre point de terminaison de Service Web, le contrat WSDL sera généré automatiquement, et vous n'aurez besoin que de son URL. Vous pouvez obtenir cet URL dans la section **`Webservices`** de **`Runtime`** de la console de gestion sur le web, une fois le point de terminaison déployé.

L'outil **`wsconsume.sh`** ou **`wsconsume.bat`** est utilisé pour consommer le contrat abstrait (WSDL) et produire les classes Java annotées et les sources optionnelles qui le définissent. La commande se trouve dans le répertoire ***`EAP_HOME`*`/bin/`** de l'installation JBoss EAP 6.

Exemple 14.17. Syntaxe de la commande `wsconsume.sh`

```
[user@host bin]$ ./wsconsume.sh --help
WSConsumeTask is a cmd line tool that generates portable JAX-WS
artifacts from a WSDL file.

usage: org.jboss.ws.tools.cmd.WSConsume [options] <wsdl-url>

options:
    -h, --help                  Show this help message
    -b, --binding=<file>        One or more JAX-WS or JAXB binding
files
    -k, --keep                  Keep/Generate Java source
    -c  --catalog=<file>        Oasis XML Catalog file for entity
resolution
    -p  --package=<name>        The target package for generated source
    -w  --wsdlLocation=<loc>    Value to use for
@WebService.wsdlLocation
    -o, --output=<directory>    The directory to put generated artifacts
    -s, --source=<directory>    The directory to put Java source
```

```
    -t, --target=<2.0|2.1|2.2>  The JAX-WS specification target
    -q, --quiet                 Be somewhat more quiet
    -v, --verbose               Show full exception stack traces
    -l, --load-consumer         Load the consumer and exit (debug
utility)
    -e, --extension             Enable SOAP 1.2 binding extension
    -a, --additionalHeaders     Enable processing of implicit SOAP
headers
    -n, --nocompile             Do not compile generated sources
```

La commande suivante génère les fichiers source **.java** listés dans la sortie, à partir du fichier **ProfileMgmtService.wsdl**. Les sources utilisent la structure de répertoire du package qui est spécifié par le commutateur **-p**.

```
[user@host bin]$  wsconsume.sh -k -p
org.jboss.test.ws.jaxws.samples.retail.profile ProfileMgmtService.wsdl
output/org/jboss/test/ws/jaxws/samples/retail/profile/Customer.java
output/org/jboss/test/ws/jaxws/samples/retail/profile/DiscountRequest.java
output/org/jboss/test/ws/jaxws/samples/retail/profile/DiscountResponse.jav
a
output/org/jboss/test/ws/jaxws/samples/retail/profile/ObjectFactory.java
output/org/jboss/test/ws/jaxws/samples/retail/profile/ProfileMgmt.java
output/org/jboss/test/ws/jaxws/samples/retail/profile/ProfileMgmtService.j
ava
output/org/jboss/test/ws/jaxws/samples/retail/profile/package-info.java
output/org/jboss/test/ws/jaxws/samples/retail/profile/Customer.class
output/org/jboss/test/ws/jaxws/samples/retail/profile/DiscountRequest.clas
s
output/org/jboss/test/ws/jaxws/samples/retail/profile/DiscountResponse.cla
ss
output/org/jboss/test/ws/jaxws/samples/retail/profile/ObjectFactory.class
output/org/jboss/test/ws/jaxws/samples/retail/profile/ProfileMgmt.class
output/org/jboss/test/ws/jaxws/samples/retail/profile/ProfileMgmtService.c
lass
output/org/jboss/test/ws/jaxws/samples/retail/profile/package-info.class
```

Les fichiers source **.java** et les fichiers **.class** compilés sont générés dans le répertoire **output/** qui se trouve dans le répertoire où vous exécutez la commande.

Tableau 14.2. Descriptions d'artefacts créés par `wsconsume.sh`

Fichier	Description
ProfileMgmt.java	Interface de point de terminaison de service.
Customer.java	Personnaliser le type de données.
Discount*.java	Personnaliser les types de données.
ObjectFactory.java	Référentiel JAXB XML.

Fichier	Description
`package-info.java`	Annotations de package JAXB.
`ProfileMgmtService.java`	Usine de connexions.

La commande **`wsconsume.sh`** génère tous les types de données personnalisés (classes JAXB annotées), l'interface de point de terminaison de service et une classe de fabrique de services. Ces artefacts sont utilisés pour créer les implémentations web service clients.

Créer un Service Stub

Les clients de service Web permet d'abstraire les détails d'un appel de service web distant stubs de service. Pour une application cliente, une invocation WS ressemble à une invocation de tout autre composant de l'entreprise. Dans ce cas, l'interface de point de terminaison de service agit comme l'interface de l'entreprise, et une classe de fabrique de service n'est pas utilisée pour sa construction comme une ébauche de service.

Exemple 14.18. Créer un Service Stub et accéder à un point de terminaison

L'exemple suivant crée d'abord une fabrique de services à l'aide de l'emplacement WSDL et le nom du service. Il utilise ensuite l'interface de point de terminaison de service créée par la commande **`wsconsume.sh`** pour construire le stub de service. Enfin, le stub peut être utilisé comme le serait toute autre interface commerciale.

Vous pouvez trouver l'URL du WSDL de votre point de terminaison dans la console de gestion sur le web. Choisissez l'élément de menu **`Runtime`** en haut à gauche, puis l'élément de menu **`Déploiements`** en bas à gauche. Cliquez sur les **`Services Web`** et sélectionnez votre déploiement pour afficher ses détails.

```
import javax.xml.ws.Service;
[...]
Service service = Service.create(
new URL("http://example.org/service?wsdl"),
new QName("MyService")
);
ProfileMgmt profileMgmt = service.getPort(ProfileMgmt.class);

// Use the service stub in your application
```

Rapporter un bogue

14.4.2. Développer une application client JAX-WS

Cette section traite des clients de Services Web JAX-WS en général. Le client communique avec et réclame une tâche du point de terminaison de JAX-WS, qui est déployé dans le conteneur Java Enterprise Edition 6. Pour obtenir des informations détaillées sur les classes, méthodes et autres détails d'implémentation mentionnés ci-dessous, reportez-vous à Section 14.5.2, « Référence API Commun JAX-WS » et aux sections pertinentes du package Javadocs inclus dans JBoss EAP 6.

Service

Aperçu

Un **`Service`** est une abstraction qui représente un service WSDL. Un service WSDL est une collection de ports liés les uns aux autres, dont chacun comprend un type de port lié à un protocole particulier et à une adresse de point de terminaison particulière.

Généralement, le service est créé quand le reste des stubs du composant sont générés à partir d'un contrat WSDL existant. Le contrat WSDL est disponible via l'URL WSDL du point de terminaison déployé, ou peut être créé à partir de la source de point de terminaison à l'aide de la commande **`wsprovide.sh`** dans le répertoire **`/bin/ EAP_HOME`**.

On appelle cela une utilisation *statique*. Dans un tel cas, vous créez des instances de la classe **`Service`** , créée elle-même comme un des composants stubs.

Vous pouvez également créer le service manuellement, par la méthode **`Service.create`**. On appelle cela une utilisation *dynamique*.

Utilisation

Cas d'utilisation static

Le cas d'utilisation *statique* pour un client JAX-WS suppose que vous ayez déjà un contrat WSDL. Ceci peut être généré par un outil externe ou généré à l'aide des annotations JAX-WS qui conviennent lorsque vous créez votre point de terminaison de JAX-WS.

Pour générer vos stubs de composants, vous pouvez utiliser le script **`wsconsume.sh`** ou **`wsconsume.bat`** inclus dans ***`EAP_HOME`*`/bin/`**. Ce script prend le WSDL URL ou fichier en tant que paramètre, et génère de nombreux fichiers structurés dans une arborescence de répertoires. La source et les fichiers de classe représentant votre **`Service`** s'appellent ***`CLASSNAME`*`_Service.java`** et ***`CLASSNAME`*`_Service.class`**, respectivement.

La classe d'implémentation générée possède deux constructeurs publics, un sans argument et un avec deux arguments. Les deux arguments représentent respectivement l'emplacement WSDL (un **`java.net.URL`**) et le nom du service (un **`javax.xml.namespace.QName`**).

Le constructeur sans argument est celui qui est le plus souvent utilisé. Dans ce cas, l'emplacement WSDL et le nom du service sont ceux que l'on trouve dans le fichier WSDL. Ceux-ci sont définis implicitement à partir de l'annotation **`@WebServiceClient`** qui décore la classe générée.

Exemple 14.19. Exemple de classe de service générée

```
@WebServiceClient(name="StockQuoteService",
targetNamespace="http://example.com/stocks",
wsdlLocation="http://example.com/stocks.wsdl")
public class StockQuoteService extends javax.xml.ws.Service
{
   public StockQuoteService()
   {
      super(new URL("http://example.com/stocks.wsdl"), new
QName("http://example.com/stocks", "StockQuoteService"));
   }

   public StockQuoteService(String wsdlLocation, QName serviceName)
   {
      super(wsdlLocation, serviceName);
   }
```

```
    ...
}
```

Cas d'utilisation dynamique

Dans un cas d'utilisation dynamique, aucun stub n'est généré automatiquement. Au lieu de cela, le client de service web utilise la méthode **`Service.create`** pour créer des instances de **`Service`**. Le fragment de code suivant illustre ce processus.

Exemple 14.20. Création de service manuellement

```
URL wsdlLocation = new URL("http://example.org/my.wsdl");
QName serviceName = new QName("http://example.org/sample",
"MyService");
Service service = Service.create(wsdlLocation, serviceName);
```

Handler Resolver

JAX-WS fournit un framework plug-in flexible pour des modules de traitement des messages, appelés *handlers*. Ces gestionnaires étendent les fonctionnalités d'un système de runtime JAX-WS. Une instance de **`Service`** donnera accès à un **`HandlerResolver`** grâce à une paire de méthodes **`getHandlerResolver`** et **`setHandlerResolver`** qui peuvent configurer un ensemble de gestionnaires sur la base de service, de port ou de protocole.

Lorsqu'une instance de **`Service`** crée un proxy ou une instance **`Dispatch`**, le Handler Resolver enregistré dans le service crée la chaîne de gestionnaires requise. Les changements ultérieurs apportés au Handler Resolver qui sont configurés pour une instance de **`Service`** n'affectent pas les gestionnaires sur proxies préalablement créés ou les instances de **`Dispatch`**.

Exécuteur

Les instances de **`Service`** peuvent être configurées par un **`java.util.concurrent.Executor`**. **`Executor`** invoque les callbacks asynchrones demandés par l'application. Les méthodes **`setExecutor`** et **`getExecutor`** de **`Service`** peuvent modifier et extraire l' **`Executor`** configuré pour un service.

Proxy dynamique

Un *dynamic proxy* est une instance de proxy de client utilisant une des méthodes **`getPort`** fournie par le **`Service`**. Le **`portName`** indique le nom du port WSDL que le service utilise. La **`serviceEndpointInterface`** indique l'interface du point de terminaison du service prise en charge par l'instance du proxy dynamique créé.

Exemple 14.21. Méthodes `getPort`

```
public <T> T getPort(QName portName, Class<T> serviceEndpointInterface)
public <T> T getPort(Class<T> serviceEndpointInterface)
```

L' *interface du point de terminaison du service* ou Service Endpoint Interface (SEI) est normalement créée par la commande **wsconsume.sh** qui interprète le WSDL et qui, à partir de cela, crée des classes Java.

Il y a aussi une méthode typée retournant un port qui est fournie. Ces méthodes renvoient également des proxys dynamiques qui implémentent la SEI. Voir les exemples suivants.

Exemple 14.22. Revoie le Port d'un Service

```
@WebServiceClient(name = "TestEndpointService", targetNamespace =
"http://org.jboss.ws/wsref",
   wsdlLocation = "http://localhost.localdomain:8080/jaxws-samples-
webserviceref?wsdl")

public class TestEndpointService extends Service
{
    ...

    public TestEndpointService(URL wsdlLocation, QName serviceName) {
        super(wsdlLocation, serviceName);
    }

    @WebEndpoint(name = "TestEndpointPort")
    public TestEndpoint getTestEndpointPort()
    {
        return (TestEndpoint)super.getPort(TESTENDPOINTPORT,
TestEndpoint.class);
    }
}
```

@WebServiceRef

L'annotation **@WebServiceRef** déclare une référence à un Service Web. Elle suit un modèle de ressource indiqué par l'annotation **javax.annotation.Resource** définie dans http://www.jcp.org/en/jsr/summary?id=250.

Cas d'utilisateur @WebServiceRef

- Vous pouvez l'utiliser pour définir une référence dont le type est une classe de **Service** générée. Dans ce cas, l'élément de valeur et le type font tous deux référence au type de classe de **Service** généré. En outre, si le type de référence peut être déduit grâce au champ ou à la déclaration de méthode à laquelle l'annotation est s'applique, les éléments de type et de valeur peuvent (mais n'y sont pas tenus) avoir la valeur par défaut de **Object.class**. Si le type ne peut pas être déduit, alors l'élément au moins doit être présent, avec une valeur par défaut.
- Vous pouvez l'utiliser pour définir une référence de type SEI. Dans ce cas, l'élément de type peut (mais pas forcément) se présenter avec sa valeur par défaut si le type de référence peut être déduit de la déclaration de champ ou de méthode annotée. Toutefois, l'élément de valeur doit toujours être présent et faire référence à un type de classe de service généré, qui est un sous-type de **javax.xml.ws.Service**. L'élément **wsdlLocation**, s'il est présent, substitue les informations d'emplacement WSDL spécifiées dans l'annotation **@WebService** de la classe de service générée référencée.

Exemple 14.23. Exemples de `@WebServiceRef`

```
public class EJB3Client implements EJB3Remote
{
   @WebServiceRef
   public TestEndpointService service4;

   @WebServiceRef
   public TestEndpoint port3;
```

Dispatch

Les Services Web XML utilisent des messages XML pour la communication entre le point de terminaison, qui est déployé dans le conteneur Java EE, et tous les clients. Les messages XML utilisent un langage XML appelé *Simple Object Access Protocol (SOAP)*. L'API JAX-WS fournit les mécanismes pour que le point de terminaison et les clients puissent chacun être en mesure d'envoyer et de recevoir des messages SOAP et de convertir les messages SOAP en Java et vice versa. Cela s'appelle **`marshalling`** et **`unmarshalling`**.

Dans certains cas, vous devez accéder aux messages SOAP bruts eux-mêmes, plutôt qu'au résultat de la conversion. La classe de **`Dispatch`** fournit cette fonctionnalité. **`Dispatch`** opère dans un des deux modes d'utilisation, qui sont identifiés par l'une des constantes suivantes.

- **`javax.xml.ws.Service.Mode.MESSAGE`** - ce mode ordonne aux applications clientes de travailler directement avec les structures de message qui sont spécifiques au protocole. Si utilisé avec une liaison de protocole SOAP, une application cliente fonctionne directement avec un message SOAP.
- **`javax.xml.ws.Service.Mode.PAYLOAD`** - ce mode amène le client à travailler avec la charge elle-même. Par exemple, s'il est utilisé avec une liaison de protocole SOAP, une application cliente travaillera alors avec le contenu SOAP plutôt qu'avec l'intégralité du message SOAP.

`Dispatch` est une API de bas niveau qui exige des clients de structurer les messages ou les charges en XML, avec une adhérence stricte aux normes du protocole individuel et une connaissance approfondie de la structure de message ou de la charge. **`Dispatch`** est une classe générique qui prend en charge l'entrée et la sortie des messages ou des charges de message de n'importe quel type.

Exemple 14.24. Utilisation de `Dispatch`

```
Service service = Service.create(wsdlURL, serviceName);
Dispatch dispatch = service.createDispatch(portName, StreamSource.class,
Mode.PAYLOAD);

String payload = "<ns1:ping
xmlns:ns1='http://oneway.samples.jaxws.ws.test.jboss.org/'/>";
dispatch.invokeOneWay(new StreamSource(new StringReader(payload)));

payload = "<ns1:feedback
xmlns:ns1='http://oneway.samples.jaxws.ws.test.jboss.org/'/>";
Source retObj = (Source)dispatch.invoke(new StreamSource(new
StringReader(payload)));
```

Invocations asynchrones

L'interface **`BindingProvider`** représente un composant qui fournit une liaison de protocole que les clients peuvent utiliser. Elle est implémentée et étendue par l'interface **`Dispatch`**.

Les instances de **`BindingProvider`** peuvent fournir des possibilités d'opérations asynchrones. Comme les opérations asynchrones, les invocations sont découplées de l'instance **`BindingProvider`** au moment de l'invocation. Le contexte de réponse n'est pas mis à jour lorsque l'opération est terminée. Au lieu de cela, un contexte de réponse distinct est mis à disposition à l'aide de l'interface de **`réponse`**.

Exemple 14.25. Invocation asynchrone

```
public void testInvokeAsync() throws Exception
{
   URL wsdlURL = new URL("http://" + getServerHost() + ":8080/jaxws-
samples-asynchronous?wsdl");
   QName serviceName = new QName(targetNS, "TestEndpointService");
   Service service = Service.create(wsdlURL, serviceName);
   TestEndpoint port = service.getPort(TestEndpoint.class);
   Response response = port.echoAsync("Async");
   // access future
   String retStr = (String) response.get();
   assertEquals("Async", retStr);
}
```

Invocations `@Oneway`

L'annotation **`@Oneway`** indique que la méthode web donnée ne prend un message d'entrée mais ne renvoie aucun message de sortie. Habituellement, une méthode **`@Oneway`** renvoie le thread de contrôle à l'application appelante avant l'exécution de la méthode commerciale.

Exemple 14.26. Exemple d'invocation `@Oneway`

```
@WebService (name="PingEndpoint")
@SOAPBinding(style = SOAPBinding.Style.RPC)
public class PingEndpointImpl
{
   private static String feedback;

   @WebMethod
   @Oneway
   public void ping()
   {
      log.info("ping");
      feedback = "ok";
   }

   @WebMethod
   public String feedback()
   {
      log.info("feedback");
      return feedback;
   }
}
```

Configuration du timeout

Il y a deux propriétés qui contrôlent le comportement du délai d'expiration de la connexion HTTP et le délai d'attente d'un client qui attend de recevoir un message. Le premier est **`javax.xml.ws.client.connectionTimeout`** et le second est **`javax.xml.ws.client.receiveTimeout`**. Chacun est exprimé en millisecondes, et la syntaxe correcte est indiquée ci-dessous.

Exemple 14.27. Configuration de timeout JAX-WS

```
public void testConfigureTimeout() throws Exception
{
   //Set timeout until a connection is established

((BindingProvider)port).getRequestContext().put("javax.xml.ws.client.con
nectionTimeout", "6000");

   //Set timeout until the response is received
   ((BindingProvider)
port).getRequestContext().put("javax.xml.ws.client.receiveTimeout",
"1000");

   port.echo("testTimeout");
}
```

Rapporter un bogue

14.5. RÉFÉRENCE DE DÉVELOPPEMENT JAX-WS

14.5.1. Activation de WS-Addressing (Web Services Addressing)

Conditions préalables

- Votre application doit posséder un service JAX-WS et une configuration client.

Procédure 14.1. Annoter et mettre à jour le code client

1. **Annoter le point de terminaison du service**
   Ajouter l'annotation **`@Addressing`** au code du point de terminaison de l'application.

   Exemple 14.28. annotation `@Addressing`

   Cet exemple montre un point de terminaison normal avec l'ajout de l'annotation **`@Addressing`**.

   ```
   package org.jboss.test.ws.jaxws.samples.wsa;

   import javax.jws.WebService;
   import javax.xml.ws.soap.Addressing;
   ```

```
@WebService
(
   portName = "AddressingServicePort",
   serviceName = "AddressingService",
   wsdlLocation = "WEB-INF/wsdl/AddressingService.wsdl",
   targetNamespace = "http://www.jboss.org/jbossws/ws-
extensions/wsaddressing",
   endpointInterface =
"org.jboss.test.ws.jaxws.samples.wsa.ServiceIface"
)
@Addressing(enabled=true, required=true)
public class ServiceImpl implements ServiceIface
{
   public String sayHello()
   {
      return "Hello World!";
   }
}
```

2. **Mise à jour du code client**
   Mettez à jour le code client dans l'application pour configurer WS-Addressing.

Exemple 14.29. La configuration client pour WS-Addressing

Cet exemple montre un client JAX-WS mis à jour pour configurer WS-Addressing.

```
package org.jboss.test.ws.jaxws.samples.wsa;

import java.net.URL;
import javax.xml.namespace.QName;
import javax.xml.ws.Service;
import javax.xml.ws.soap.AddressingFeature;

public final class AddressingTestCase
{
   private final String serviceURL =
       "http://localhost:8080/jaxws-samples-
wsa/AddressingService";

   public static void main(String[] args) throws Exception
   {
      // construct proxy
      QName serviceName =
          new QName("http://www.jboss.org/jbossws/ws-
extensions/wsaddressing",
                    "AddressingService");
      URL wsdlURL = new URL(serviceURL + "?wsdl");
      Service service = Service.create(wsdlURL, serviceName);
      ServiceIface proxy =
          (ServiceIface)service.getPort(ServiceIface.class,
                                        new
AddressingFeature());
      // invoke method
```

```
            proxy.sayHello();
        }
    }
```

Résultat

Le client et le point de terminaison communiquent maintenant avec WS-Addressing.

Rapporter un bogue

14.5.2. Référence API Commun JAX-WS

Plusieurs concepts de développement de JAX-WS sont partagés entre les clients et les points de terminaison de Service Web. Incluent le framework de gestionnaires, le contexte du message et la gestion des erreurs.

Framework de gestionnaires

Le framework de gestionnaires est implémenté par une liaison de protocole de JAX-WS dans le runtime du client et le point de terminaison, qui est le composant de serveur. Les proxies et les instances **`Dispatch`**, connus collectivement en tant que *fournisseurs de liaison*, chacun utilisant des liaisons de protocole pour lier leur fonctionnalité abstraite à des protocoles spécifiques.

Les gestionnaires côté client et côté serveur sont organisés en une liste ordonnée, connue sous le nom de *chaîne de gestionnaire*. Les gestionnaires au sein d'une chaîne de gestionnaires sont appelés à chaque fois qu'un message est envoyé ou reçu. Les messages entrants sont traités par des gestionnaires avant que le fournisseur de liaison les traite. Les messages sortants sont traités par les gestionnaires après que le fournisseur de liaison les aient traités.

Les gestionnaires sont invoqués avec un contexte de message qui fournit des méthodes pour accéder et modifier les messages entrants et sortants et gérer un ensemble de propriétés. Les propriétés de contexte de message facilitent la communication entre des gestionnaires individuels, ainsi qu'entre les gestionnaires, les clients et les implémentations de services. Différents types de gestionnaires sont invoqués avec différents types de contextes de messages.

Types de gestionnaires de messages

Gestionnaire logique

Les *gestionnaires logiques* n'opèrent que sur des propriétés de contexte de messages et sur des charges utiles de messages. Les gestionnaires logiques sont libérés des protocoles et ne peuvent pas affecter les parties d'un message qui soient spécifiques à un protocole. Les gestionnaires logiques implémentent l'interface **`javax.xml.ws.handler.LogicalHandler`**.

Gestionnaire de protocoles

Les *gestionnaires de protocoles* fonctionnent sur des propriétés de contexte de message et des messages de protocole spécifique. Les gestionnaires de protocoles sont spécifiques à un protocole particulier et peuvent accéder et modifier les aspects spécifiques d'un protocole de message. Les gestionnaires de protocoles implémentent une interface dérivée de **`javax.xml.ws.handler.Handler except javax.xml.ws.handler.LogicalHandler`**.

Gestionnaire de points de terminaison

Sur un point de terminaison de service, les gestionnaires sont définis à l'aide de l'annotation **`@HandlerChain`**. L'emplacement du fichier de chaîne de gestionnaires peut être soit un

java.net.URL absolu dans **externalForm** ou un chemin d'accès relatif du fichier source ou du fichier de classe.

Exemple 14.30. Exemple de gestionnaire de points de terminaison

```
@WebService
@HandlerChain(file = "jaxws-server-source-handlers.xml")
public class SOAPEndpointSourceImpl
{
   ...
}
```

Service Client Handler

Sur un client JAX-WS, les gestionnaires sont définis soit en utilisant l'annotation **@HandlerChain**, comme dans les points de terminaison de service, soit de façon dynamique, à l'aide de l'API JAX-WS.

Exemple 14.31. Définir un Service Client Handler par une API

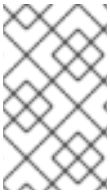
```
Service service = Service.create(wsdlURL, serviceName);
Endpoint port = (Endpoint)service.getPort(Endpoint.class);

BindingProvider bindingProvider = (BindingProvider)port;
List<Handler> handlerChain = new ArrayList<Handler>();
handlerChain.add(new LogHandler());
handlerChain.add(new AuthorizationHandler());
handlerChain.add(new RoutingHandler());
bindingProvider.getBinding().setHandlerChain(handlerChain);
```

L'appel à la méthode **setHandlerChain** est requis.

Contexte du message

L'interface **MessageContext** est une super interface pour tous les contextes de message JAX-WS. Elle étend **Map <String,Object>** de méthodes et des constantes supplémentaires pour gérer un ensemble de propriétés qui permettent aux gestionnaires d'une chaîne de gestionnaires de partager l'état connexe de traitement. Par exemple, un gestionnaire peut utiliser la méthode **put** pour insérer une propriété dans le contexte du message. Un ou plusieurs autres gestionnaires de la chaîne de gestionnaires peuvent obtenir par la suite le message via la méthode **get**.

Les propriétés sont catégorisées soient comme **APPLICATION** ou **GESTIONNAIRE**. Toutes les propriétés sont disponibles pour tous les gestionnaires d'une instance d'un *modèle d'échange de messages* ou MEP (de l'anglais Message Exchange Pattern) d'un point de terminaison particulier. Par exemple, si un gestionnaire logique met une propriété dans le contexte du message, cette propriété sera également disponible à tous les gestionnaires de protocole dans la chaîne au cours de l'exécution d'une instance MEP.



NOTE

Un modèle d'échanges de messages asynchrone (MEP) permet d'envoyer et de recevoir des messages de façon asynchrone au niveau de la connexion HTTP. Vous pouvez l'activer en définissant des propriétés supplémentaires dans le contexte de la demande.

Les propriétés catégorisées en tant qu' **`APPLICATION`** sont également rendues disponibles en tant qu'applications clients et implémentations de point de terminaison de services. Le **`defaultscope`** d'une propriété est **`HANDLER`** par défaut.

Les messages logiques et SOAP utilisent des contextes différents.

Contexte de message logique

Lorsque les gestionnaires logiques sont invoqués, ils reçoivent un contexte de message de type **`LogicalMessageContext`**. Le **`LogicalMessageContext`** étend **`MessageContext`** par des méthodes qui récupèrent et modifient la charge du message. Il ne donne pas accès aux aspects spécifiques au protocole d'un message. Une liaison de protocole définit quels composants d'un message sont disponibles via un contexte logique message. Un gestionnaire logique déployé dans une liaison SOAP peut accéder au contenu du corps SOAP, mais pas aux en-têtes SOAP. En revanche, la liaison XML/HTTP définit qu'un gestionnaire de logique accède à l'entière charge XML d'un message.

Contexte de message SOAP

Quand les gestionnaires SOAP sont invoqués, ils reçoivent un **`SOAPMessageContext`**. Le **`SOAPMessageContext`** étend **`MessageContext`** par des méthodes qui obtiennent et modifient la charge du message SOAP.

Gestion des fautes

Une application peut lever une **`SOAPFaultException`** ou une exception spécifique à l'application utilisateur. Dans ce dernier cas, les wrapper beans de faute requis seront générés en cours d'exécution s'ils ne font pas déjà partie du déploiement.

Exemple 14.32. Exemple de gestion des fautes

```
public void throwSoapFaultException()
{
   SOAPFactory factory = SOAPFactory.newInstance();
   SOAPFault fault = factory.createFault("this is a fault string!", new
QName("http://foo", "FooCode"));
   fault.setFaultActor("mr.actor");
   fault.addDetail().addChildElement("test");
   throw new SOAPFaultException(fault);
}
```

```
public void throwApplicationException() throws UserException
{
   throw new UserException("validation", 123, "Some validation error");
}
```

Annotations JAX-WS

Les annotations disponibles via JAX-WS API sont définies dans JSR-224, que l'on peut trouver à l'adresse suivante http://www.jcp.org/en/jsr/detail?id=224. Ces annotations sont dans le package **`javax.xml.ws`**.

Les annotations disponibles via JWS API sont définies dans JSR-181, que l'on peut trouver à l'adresse suivante http://www.jcp.org/en/jsr/detail?id=181. Ces annotations sont dans le package **`javax.jws`**.

Rapporter un bogue

CHAPITRE 15. WEBSOCKETS

15.1. WEBSOCKETS

Le protocole WebSocket fournit la communication dans les deux sens entre serveur et clients web. Les communications entre les clients et le serveur sont basées sur des événements, ce qui permet un traitement plus rapide et une plus petite bandwidth par rapport aux méthodes basées sur des interrogations. WebSocket peut s'utiliser en application web par l'intermédiaire d'un API JavaScript.

Une connexion est d'abord établie entre le client et le serveur sous forme de connexion HTTP. Le client demande alors une connexion WebSocket en utilisant l'en-tête **`Upgrade`**. Toutes les communications sont alors en full-duplex sur la même connexion TCP/IP, avec une charge minimale de données. Comme chaque message n'inclut pas de contenu d'en-tête HTTP inutile, les communications Websocket nécessitent moins de bandwidth. Le résultat est une voie de communication de faible latence, adaptée aux applications nécessitant une réactivité en temps réel.

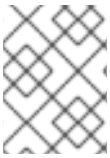
L'application JBoss EAP 6.3 WebSocket fournit un support complet d'injection de dépendance des points de terminaison de serveur; cependant, elle ne fournit pas de services CDI pour les clients de points de terminaison. Le support CDI est limité à celui qui est exigé par la plate-forme Java EE 6, et de ce fait, les fonctionnalités Java EE 7, telles les intercepteurs sur les points de terminaison, ne sont pas pris en charge.

Rapporter un bogue

15.2. CRÉER UNE APPLICATION WEBSOCKET

Une application WebSocket nécessite les composants et les changements de configuration suivants :

- Un client Java ou un client WebSocket : http://caniuse.com/websockets
- Une classe de point de terminaison de serveur WebSocket.
- Un fichier **`jboss-web.xml`** configuré pour activer WebSockets.
- Dépendances de projet configurées pour déclarer une dépendance sur l'API WebSocket.
- Activer le connecteur **`NIO2`** dans le sous-système **`web`** du fichier de configuration du serveur JBoss EAP. Si vous installez les composants natifs de votre système d'exploitation dans votre installation JBoss EAP et que vous avez déjà installé l'APR (Apache Portability Runtime), vous pouvez choisir d'activer le connecteur **`APR`** à la place.



NOTE

Les applications WebSocket requièrent un environnement Java Runtime de version 7 ou supérieure. Sinon, WebSocket ne sera pas activé.

Procédure 15.1. Créer l'application WebSocket

Voici un exemple simple d'application WebSocket. Vous trouverez des boutons pour ouvrir une connexion, envoyer un message, et fermer une connexion. Cet exemple n'implémente pas d'autres fonctions, ni n'inclut de gestion d'erreurs dont on aurait besoin dans une application réelle.

1. **Créer un client HTML JavaScript.**
   Voici un exemple de client WebSocket. Il contient les fonctions JavaScript suivantes :

- **`connect()`** : cette fonction crée la connexion webSocket qui donne l'URI de WebSocket. L'emplacement de la ressource correspond à la ressource définie dans la classe de point de terminaison du serveur. Cette fonction intercepte et gère les WebSockets **onopen**, **`onmessage`**, **`onerror`**, et **`onclose`**.
- **`sendMessage()`** : cette fonction a le même nom que celui qui se trouve sur le formulaire, crée un message, et l'envoie par la commande WebSocket.send().
- **`disconnect()`** : cette fonction génère la commande WebSocket.close().
- **`displayMessage()`** : cette fonction détermine le message affiché sur la page à la valeur donnée par la méthode de point de terminaison de WebSocket.
- **`displayStatus()`** : cette fonction afficher le statut de la connexion de WebSocket.

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
    <head>
        <title>WebSocket: Say Hello</title>
        <link rel="stylesheet" type="text/css"
href="resources/css/hello.css" />
        <script type="text/javascript">
            var websocket = null;

            function connect() {
                var wsURI = 'ws://' + window.location.host +
'/jboss-websocket-hello/websocket/helloName';
                websocket = new WebSocket(wsURI);

                websocket.onopen = function() {
                    displayStatus('Open');
                    document.getElementById('sayHello').disabled =
false;
                    displayMessage('Connection is now open. Type a
name and click Say Hello to send a message.');
                };
                websocket.onmessage = function(event) {
                    // log the event
                    displayMessage('The response was received! ' +
event.data, 'success');
                };
                websocket.onerror = function(event) {
                    // log the event
                    displayMessage('Error! ' + event.data,
'error');
                };
                websocket.onclose = function() {
                    displayStatus('Closed');
                    displayMessage('The connection was closed or
timed out. Please click the Open Connection button to reconnect.');
                    document.getElementById('sayHello').disabled =
true;
                };
            }
```

```
            function disconnect() {
                if (websocket !== null) {
                    websocket.close();
                    websocket = null;
                }
                message.setAttribute("class", "message");
                message.value = 'WebSocket closed.';
                // log the event
            }

            function sendMessage() {
                if (websocket !== null) {
                    var content =
document.getElementById('name').value;
                    websocket.send(content);
                } else {
                    displayMessage('WebSocket connection is not
established. Please click the Open Connection button.', 'error');
                }
            }

            function displayMessage(data, style) {
                var message =
document.getElementById('hellomessage');
                message.setAttribute("class", style);
                message.value = data;
            }

            function displayStatus(status) {
                var currentStatus =
document.getElementById('currentstatus');
                currentStatus.value = status;
            }

        </script>
    </head>
    <body>

        <div>
            <h1>Welcome to JBoss!</h1>
            <div>This is a simple example of a WebSocket
implementation.</div>
            <div id="connect-container">
                <div>
                    <fieldset>
                        <legend>Connect or disconnect using
WebSocket :</legend>
                        <input type="button" id="connect"
onclick="connect();" value="Open Connection" />
                        <input type="button" id="disconnect"
onclick="disconnect();" value="Close Connection" />
                    </fieldset>
                </div>
                <div>
                    <fieldset>
                        <legend>Type your name below. then click
```

```
the `Say Hello` button :</legend>
                        <input id="name" type="text" size="40"
style="width: 40%"/>
                        <input type="button" id="sayHello"
onclick="sendMessage();" value="Say Hello" disabled="disabled"/>
                    </fieldset>
                </div>
                <div>Current WebSocket Connection Status: <output
id="currentstatus" class="message">Closed</output></div>
                <div>
                    <output id="hellomessage" />
                </div>
            </div>
        </div>
    </body>
</html>
```

2. **Créer le point de terminaison du serveur de WebSocket.**
   Vous pouvez créer un point de terminaison de serveur WebSocket par l'une des méthodes suivantes.

   - **`Programmatic Endpoint`** : le point de terminaison prolonge la classe de point de terminaison.
   - **`Annotated Endpoint`** : la classe de point de terminaison utilise des annotations pour interagir avec les événements WebSocket. Plus facile à codifier qu'un point de terminaison programmatique.

   L'exemple de code ci-dessous utilise une approche de point de terminaison annoté et gère les événements suivants.

   - L'annotation **`@ServerEndpoint`** identifie cette classe comme point de terminaison de serveur WebSocket et indique le chemin d'accès.
   - L'annotation **`@OnOpen`** s'active quand la connexion de WebSocket est ouverte.
   - L'annotation **`@OnMessage`** s'active quand un message est envoyé vers une connexion de WebSocket.
   - L'annotation **`@OnClose`** s'active quand la connexion de WebSocket est fermée.

```
package org.jboss.as.quickstarts.websocket_hello;

import javax.websocket.CloseReason;
import javax.websocket.OnClose;
import javax.websocket.OnMessage;
import javax.websocket.OnOpen;
import javax.websocket.Session;
import javax.websocket.server.ServerEndpoint;

@ServerEndpoint("/websocket/helloName")
public class HelloName {

    @OnMessage
    public String sayHello(String name) {
        System.out.println("Say hello to '" + name + "'");
```

```
        return ("Hello" + name);
    }

    @OnOpen
    public void helloOnOpen(Session session) {
        System.out.println("WebSocket opened: " + session.getId());
    }

    @OnClose
    public void helloOnClose(CloseReason reason) {
        System.out.println("Closing a WebSocket due to " +
reason.getReasonPhrase());
    }
}
```

3. **Configurer le fichier jboss-web.xml.**
   Vous devez créer l'élément **<enable-websockets>** dans l'application **WEB-INF/jboss-web.xml** et la définir à **true**.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Enable WebSockets -->
<jboss-web>
   <enable-websockets>true</enable-websockets>
</jboss-web>
```

4. **Déclarer la dépendance de l'API de Websocket dans votre fichier POM de projet.**
   Si vous utilisez Maven, vous devrez ajouter la dépendance suivante au fichier **pom.xml** du projet.

```
<dependency>
  <groupId>org.jboss.spec.javax.websocket</groupId>
  <artifactId>jboss-websocket-api_1.0_spec</artifactId>
  <version>1.0.0.Final</version>
  <scope>provided</scope>
</dependency>
```

5. **Configurer le serveur JBoss EAP.**
   Configurer le **<connector> http** dans le sous-système **web** du fichier de configuration du serveur pour qu'il puisse utliser le protocole **NIO2**.

   a. Démarrer le serveur JBoss EAP.

   b. Lancer l'interface CLI par la commande pour votre système d'exploitation.

      Dans Linux :

```
EAP_HOME/bin/jboss-cli.sh --connect
```

      Dans Windows :

```
EAP_HOME\bin\jboss-cli.bat --connect
```

   c. Activer le connecteur **NIO2** ou **APR** dans le sous-système **web** du fichier de configuration du serveur JBoss EAP.

- Saisir la commande suivante pour pouvoir utiliser le protocole de connecteur **NIO2** Java non bloquant :

```
/subsystem=web/connector=http/:write-
attribute(name=protocol,value=org.apache.coyote.http11.Http11N
ioProtocol)
```

- Si vous avez installé l'APR (Apache Portability Runtime), vous pourrez saisir les commandes suivantes pour utiliser le protocole de connecteur **APR** natif :

```
/subsystem=web:write-attribute(name=native,value=true)
/subsystem=web/connector=http/:write-
attribute(name=protocol,value=org.apache.coyote.http11.Http11A
prProtocol)
```

Quelle que soit la commande dont il s'agit, vous devrez voir le résultat suivant apparaître :

```
{
    "outcome" => "success",
    "response-headers" => {
        "operation-requires-reload" => true,
        "process-state" => "reload-required"
    }
}
```

d. Indiquer au serveur de charger la configuration à nouveau.

```
:reload
```

Vous devriez voir apparaître le résultat suivant :

```
{
    "outcome" => "success",
    "result" => undefined
}
```

e. Réviser les changements apportés au fichier de configuration du serveur JBoss EAP. Le sous-système **web** doit maintenant contenir l'XML suivant pour le **<connector> http**.

Pour la configuration de connecteur **NIO2** :

```
<subsystem xmlns="urn:jboss:domain:web:2.1" default-virtual-
server="default-host" native="false">
    <connector name="http"
protocol="org.apache.coyote.http11.Http11NioProtocol"
scheme="http" socket-binding="http"/>
    <virtual-server name="default-host" enable-welcome-
root="true">
    <alias name="localhost"/>
        <alias name="example.com"/>
    </virtual-server>
</subsystem>
```

Pour la configuration de connecteur **APR** :

```
<subsystem xmlns="urn:jboss:domain:web:2.1" default-virtual-
server="default-host" native="true">
    <connector name="http"
protocol="org.apache.coyote.http11.Http11AprProtocol"
scheme="http" socket-binding="http"/>
    <virtual-server name="default-host" enable-welcome-
root="true">
        <alias name="localhost"/>
        <alias name="example.com"/>
    </virtual-server>
</subsystem>
```

Rapporter un bogue

CHAPITRE 16. IDENTITÉ AU SEIN D'APPLICATIONS

16.1. CONCEPTS DE BASE

16.1.1. Cryptage

Le *cryptage* désigne le brouillage des informations sensibles en appliquant des algorithmes mathématiques. Le cryptage est une des fondations de la sécurisation de votre infrastructure pour les violations de données, les pannes de système et autres risques.

Le cryptage peut être appliqué aux données de chaînes simples, comme les mots de passe. Il peut également être appliqué aux flux de communication de données. Le protocole HTTPS, par exemple, crypte toutes les données avant de les transférer d'une partie à une autre. Si vous vous connectez d'un serveur à un autre en utilisant le protocole Secure Shell (SSH), l'ensemble de votre communication sera envoyé vers un *tunnel* crypté.

Rapporter un bogue

16.1.2. Les domaines de sécurité

Les domaines de sécurité font partie du sous-système de sécurité JBoss EAP. La configuration de la sécurité est désormais gérée de façon centralisée, ou par le contrôleur de domaines d'un domaine géré ou par le serveur autonome.

Un domaine de sécurité se compose de configurations d'authentification, d'autorisation, de mappage de sécurité et d'audit. Il met en place la sécurité déclarative *Java Authentication and Authorization Service (JAAS)*.

L'authentification est impliquée dans la vérification de l'identité d'un utilisateur. Dans la terminologie de la sécurité, cet utilisateur est appelé un *principal*. Bien que l'authentification et l'autorisation soient différentes, de nombreux modules d'authentification intégrés gèrent également l'autorisation.

Une *authorization* est un process par lequel le serveur détermine si un utilisateur authentifié a la permission d'accéder à des privilèges ou à des ressources particulières dans le système ou l'opération.

Security mapping se rapporte à la possibilité d'ajouter, de modifier ou de supprimer des informations d'un principal, rôle ou attribut avant de passer les informations à votre application.

L'Auditing Manager vous permet de configurer les *provider modules* pour contrôler la façon dont les événements de sécurité sont rapportés.

Si vous utilisez des domaines de sécurité, vous pouvez supprimer toutes les configurations de sécurité spécifiques à votre application proprement dite. Cela permet de modifier les paramètres de sécurité de façon centralisée. Un scénario courant qui bénéficie de ce type de structure de configuration est le processus de déplacement des applications entre les environnements de test et de production.

Rapporter un bogue

16.1.3. Cryptage SSL

SSL (Secure Sockets Layer) crypte le trafic réseau entre deux systèmes. Le trafic entre les deux systèmes est crypté à l'aide d'une clé bidirectionnelle, générée au cours de la phase de connexion de protocole ou *handshake* et qui n'est connue que par ces deux systèmes.

Pour les échanges bidirectionnels, SSL utilise PKI (de l'anglais Public Key Infrastructure), une méthode d'encodage qui utilise une *key pair* (paire de clés). Une paire de clés est composée de deux clés cryptographiques distinctes, mais correspondantes - une clé publique et une clé privée. La clé publique est partagée avec les autres et est utilisée pour crypter les données et la clé privée est tenue secrète et est utilisée pour déchiffrer des données qui ont été chiffrées à l'aide de la clé publique.

Lorsqu'un client fait une demande de connexion sécurisée, une phase de protocole de transfert prend place avant que la communication sécurisée puisse commencer. Pendant la négociation SSL, le serveur transmet sa clé publique au client sous la forme d'un certificat. Le certificat contient l'identité du serveur (son URL), la clé publique du serveur et une signature numérique valide le certificat. Ensuite, le client valide le certificat et prend une décision quant à savoir si le certificat doit être approuvé ou non. Si le certificat est approuvé, le client génère la clé de cryptage bidirectionnel pour la connexion SSL, la chiffre à l'aide de la clé publique du serveur et la renvoie sur le serveur. Le serveur décrypte la clé de cryptage bidirectionnel à l'aide de sa clé privée, et ensuite, la communication entre les deux machines est cryptée via cette connexion par la clé de cryptage bidirectionnel.

Rapporter un bogue

16.1.4. Sécurité déclarative

La *sécurité déclarative* est une méthode de séparation des problèmes de sécurité du code d'application, qui utilise le conteneur pour gérer la sécurité. Le conteneur procure un système d'autorisation basée soit sur les permissions de fichiers ou basée utilisateur, groupe ou rôle. Cette approche est normalement supérieure à la sécurité *programmatique*, qui donne à l'application toutes les responsabilités pour la sécurité.

La plate-forme JBoss EAP 6 fournit une sécurité déclarative par les domaines de sécurité.

Rapporter un bogue

16.2. SÉCURITÉ BASÉE-RÔLE POUR LES APPLICATIONS

16.2.1. La sécurité des applications

Pour chaque développeur d'applications, sécuriser vos applications est une tâche importante couvrant des aspects multiples. JBoss EAP 6 vous donne tous les outils dont vous aurez besoin pour rédiger des applications sécurisées, y compris les possibilités suivantes :

- Section 16.2.2, « Authentification »
- Section 16.2.3, « L'autorisation »
- Section 16.2.4, « Security Auditing »
- Section 16.2.5, « Security Mapping »
- Section 16.1.4, « Sécurité déclarative »
- Section 16.4.2.1, « Permissions de méthodes EJB »
- Section 16.4.3.1, « Les annotations de sécurité EJB »

Voir également Section 16.2.8, « Utiliser un domaine de sécurité dans votre application ».

Rapporter un bogue

16.2.2. Authentification

L'authentification consiste à identifier un sujet et à vérifier l'authenticité de l'identification. Le mécanisme d'authentification le plus commun est une combinaison de nom d'utilisateur et de mot de passe. D'autres mécanismes d'authentification communs utilisent des clés partagées, des smart cards (cartes à puce) ou des empreintes digitales. Le résultat d'une authentification réussie s'appelle un Principal, en termes de sécurité déclarative Java Enterprise Edition.

JBoss EAP utilise un système pouvant se connecter à des modules d'authentification en vue de flexibilité et d'intégration avec les systèmes d'authentification que vous utilisez dans votre organisation. Chaque domaine de sécurité contient un ou plusieurs modules d'authentification configurés. Chaque module comprend des paramètres de configuration supplémentaires pour personnaliser son comportement. Le plus simple consiste à configurer le sous-système d'authentification dans la console de gestion web.

L'authentification n'est pas la même chose que l'autorisation, même si elles sont souvent liées. Bon nombre des modules d'authentification intégrés peuvent également gérer des autorisations.

Rapporter un bogue

16.2.3. L'autorisation

L'autorisation est un mécanisme d'octroi ou de refus de permission d'accéder à une ressource, basé sur l'identité. Ce mécanisme est implémenté sous forme de rôles de sécurité déclarative, qui peuvent être donnés à des principaux.

JBoss EAP utilise un système modulaire pour configurer l'autorisation. Chaque domaine de sécurité peut contenir une ou plusieurs stratégies d'autorisation. Chaque stratégie possède un module de base qui définit son comportement. Il est configuré via les attributs et indicateurs spécifiques. La façon la plus simple consiste à configurer le sous-système de l'autorisation à l'aide de la console de gestion basée web.

L'authentification est différente de l'autorisation, et a lieu, en général, après l'authentification. La plupart des modules d'authentification gèrent également l'autorisation.

Rapporter un bogue

16.2.4. Security Auditing

Security Auditing se réfère au déclenchement d'événements, comme écrire un blog en réponse à un événement qui a lieu dans le sous-système de sécurité. Les mécanismes de sécurité sont configurés dans le cadre du domaine de sécurité, avec les informations d'authentification, d'autorisation ou de mappage de sécurité.

Auditing utilise *des modules de fournisseur*. Vous pouvez en utiliser un existant ou bien créer le vôtre.

Rapporter un bogue

16.2.5. Security Mapping

Le mappage de sécurité vous permet de combiner l'authentification et l'autorisation d'informations après que l'authentification ou l'autorisation aient eu lieu, mais avant que l'information ait été passée à votre application. Un exemple qui l'illustre est l'utilisation d'un certificat X509 pour l'authentification, puis la conversion du principal du certificat en nom logique que votre application puisse afficher.

Vous pouvez mapper vos principaux (authentification), rôles (autorisation), ou identifiants (attributs qui ne sont ni principaux, ni rôles).

Le mappage de rôles est utilisé pour ajouter, remplacer ou supprimer des rôles du sujet après l'authentification.

Le mappage du principal est utilisé pour modifier un principal après l'authentification.

Le mappage d'attributs est utilisé pour convertir des attributs d'un système externe à utiliser par votre application, et vice versa.

Rapporter un bogue

16.2.6. Java Authentication et Authorization Service (JAAS)

Java Authentication and Authorization Service (JAAS) est une API de sécurité qui consiste en un ensemble de packages Java conçus pour l'autorisation et l'authentification des utilisateurs. L'API est une implémentation Java du framework standard PAM (de l'anglais Pluggable Authentication Modules). Il étend l'architecture de contrôle d'accès de Java Enterprise Edition pour approuver l'autorisation basée utilisateur.

Dans JBoss EAP 6, JAAS ne fournit qu'une sécurité déclarative basée rôle. Pour plus d'informations sur le sécurité déclarative, voir Section 16.1.4, « Sécurité déclarative ».

JAAS est indépendant de toute technologie d'authentification sous-jacente, comme Kerberos ou LDAP. Vous pouvez modifier votre structure de sécurité sous-jacente sans modifier votre application. Vous devez uniquement modifier la configuration JAAS.

Rapporter un bogue

16.2.7. Java Authentication et Authorization Service (JAAS)

L'architecture de sécurité de JBoss EAP 6 comprend le sous-système de configuration de sécurité, des configurations de sécurité propres aux applications qui sont incluses dans plusieurs fichiers de configuration de l'application.

Domaine, Groupe serveur et Configuration spécifique au serveur

Les groupes de serveurs (dans un domaine géré) et les serveurs (dans un serveur autonome) comprennent la configuration des domaines de la sécurité. Un domaine de sécurité comprend des informations sur une combinaison d'authentification, autorisation, mapping et modules, avec détails de configuration. Une application spécifie quel domaine de sécurité est exigé, par son nom, dans son fichier **`jboss-web.xml`**.

Configuration spécifique à une application

Une configuration spécifique à une application a lieu dans un ou plusieurs fichiers.

Tableau 16.1. Fichiers de configuration spécifique à une application

Fichier	Description

Fichier	Description
ejb-jar.xml	Le descripteur de déploiement pour une application Enterprise JavaBeans (EJB), situé dans le répertoire **`META-INF`** de l'archive. Utiliser le fichier **`ejb-jar.xml`** pour préciser les rôles et les mapper aux principaux, au niveau de l'application. Vous pouvez également limiter certaines méthodes et classes spécifiques à certains rôles. Également utilisé pour les autres configurations EJB spécifiques non liées à la sécurité.
web.xml	Le descripteur de déploiement d'une application web en Java Enterprise Edition (EE). Utilisez le fichier **`web.xml`** pour déclarer le domaine de sécurité que l'application utilise pour l'authentification et l'autorisation, ainsi que les contraintes de transport et ressources, comme limiter les types de demandes HTTP autorisées. Vous pouvez également configurer l'authentification basée-web dans ce fichier. Utilisé également pour d'autre configurations spécifiques à l'application non liées à la sécurité.
jboss-ejb3.xml	Contient des extensions au descripteur **`ejb-jar.xml`** spécifiques à JBoss.
jboss-web.xml	Contient des extensions au descripteur **`web.xml`** spécifiques à JBoss.

NOTE

Les fichiers **`ejb-jar.xml`** et **`web.xml`** sont définis dans la spécification Java Enterprise Edition (Java EE). Le fichier **`jboss-ejb3.xml`** fournit des extensions spécifiques à JBoss pour le fichier **`ejb-jar.xml`**, et le fichier **`jboss-web.xml`** fournit des extensions spécifiques à JBoss pour le fichier **`web.xml`**.

Rapporter un bogue

16.2.8. Utiliser un domaine de sécurité dans votre application

Aperçu

Pour utiliser un domaine de sécurité dans votre application, vous devez tout d'abord définir le domaine dans le fichier de configuration du serveur, puis vous devez l'activer pour une application dans le descripteur de déploiement de l'application. Ensuite, vous devez ajouter les annotations requises à l'EJB qui les utilise. Cette rubrique décrit les étapes requises pour utiliser un domaine de sécurité dans votre application.

AVERTISSEMENT

Si une application fait partie d'un domaine de sécurité qui utilise un cache d'authentification, les authentifications utilisateur de cette application seront rendues disponibles à d'autres applications dans ce domaine de sécurité.

Procédure 16.1. Configurez votre application pour qu'elle puisse utiliser un domaine de sécurité

1. **Définir le domaine de sécurité**
   Vous devez définir le domaine de sécurité dans le fichier de configuration du serveur, puis l'activer pour une application dans le fichier du descripteur de l'application.

   a. **Configurez le domaine de sécurité dans le fichier de configuration du serveur**
   Le domaine de sécurité est configuré dans le sous-système de **`sécurité`** du fichier de configuration du serveur. Si l'instance de JBoss EAP 6 s'exécute dans un domaine géré, il s'agira du fichier **`domain/configuration/domain.xml`**. Si l'instance de JBoss EAP 6 s'exécute comme un serveur autonome, ce sera le fichier **`standalone/configuration/standalone.xml`**.

   Les domaines de sécurité **`other`**, **`jboss-web-policy`**, et **`jboss-ejb-policy`** sont fournis par défaut dans JBoss EAP 6. L'exemple XML suivant a été copié à partir du sous-système de **`sécurité`** dans le fichier de configuration du serveur.

   L'attribut **`cach-type`** d'un domaine de sécurité spécifie un cache pour pouvoir effectuer des contrôles d'authentification plus rapides. Les valeur autorisées sont les valeurs par **`défaut`** en cas de simple mappe comme cache ou **`infinispan`** pour un cache Infinispan.

```
<subsystem xmlns="urn:jboss:domain:security:1.2">
    <security-domains>
        <security-domain name="other" cache-type="default">
            <authentication>
                <login-module code="Remoting" flag="optional">
                    <module-option name="password-stacking"
value="useFirstPass"/>
                </login-module>
                <login-module code="RealmDirect"
flag="required">
                    <module-option name="password-stacking"
value="useFirstPass"/>
                </login-module>
            </authentication>
        </security-domain>
        <security-domain name="jboss-web-policy" cache-
type="default">
            <authorization>
                <policy-module code="Delegating"
flag="required"/>
            </authorization>
        </security-domain>
        <security-domain name="jboss-ejb-policy" cache-
type="default">
```

```
            <authorization>
                <policy-module code="Delegating"
flag="required"/>
            </authorization>
        </security-domain>
    </security-domains>
</subsystem>
```

Vous pouvez configurer des domaines de sécurité supplémentaires selon les besoins par la console de gestion ou par l'interface CLI.

b. **Activer le domaine de sécurité dans le fichier de descripteur de l'application.**
Le domaine de sécurité est spécifié dans l'élément enfant **`<security-domain>`** de l'élément **`<jboss-web>`** du fichier **`WEB-INF/jboss-web.xml`** de l'application. L'exemple suivant configure un domaine de sécurité nommé **`my-domain`**.

```
<jboss-web>
    <security-domain>my-domain</security-domain>
</jboss-web>
```

Il s'agit d'une des configurations que vous pouvez indiquer dans le descripteur **`WEB-INF/jboss-web.xml`**.

2. **Ajoutez l'annotation requise à l'EJB.**
Vous pouvez configurer la sécurité dans EJB par les annotations **`@SecurityDomain`** et **`@RolesAllowed`**. L'exemple de code EJB suivant limite l'accès au domaine de sécurité **`other`** aux utilisateurs ayant pour rôle **`guest`** (invité).

```
package example.ejb3;

import java.security.Principal;

import javax.annotation.Resource;
import javax.annotation.security.RolesAllowed;
import javax.ejb.SessionContext;
import javax.ejb.Stateless;

import org.jboss.ejb3.annotation.SecurityDomain;

/**
 * Simple secured EJB using EJB security annotations
 * Allow access to "other" security domain by users in a "guest"
role.
 */
@Stateless
@RolesAllowed({ "guest" })
@SecurityDomain("other")
public class SecuredEJB {

   // Inject the Session Context
   @Resource
   private SessionContext ctx;

   /**
    * Secured EJB method using security annotations
```

```
     */
    public String getSecurityInfo() {
       // Session context injected using the resource annotation
       Principal principal = ctx.getCallerPrincipal();
       return principal.toString();
    }
}
```

Pour obtenir des exemples de code supplémentaires, voir le quickstart **`ejb-security`** dans le package JBoss EAP 6 Quickstarts disponible à partir du portail clients de Red Hat.

Rapporter un bogue

16.2.9. Utilisation de la sécurité basée-rôle dans les Servlets

Pour ajouter la sécurité à un servlet, vous devez mapper chaque servlet à un type d'URL et créer des contraintes de sécurité sur les types d'URL qui doivent être sécurisés. Les contraintes de sécurité limitent l'accès des URL aux rôles. L'authentification et l'autorisation sont gérées par le domaine de sécurité spécifié dans **`jboss-web.xml`** du WAR.

Conditions préalables

Avant d'utiliser la sécurité basée-rôles dans un servlet, le domaine de sécurité utilisé pour authentifier et autoriser l'accès doit être configuré sur la plateforme JBoss EAP 6.

Procédure 16.2. Ajout de la sécurité basée-rôle dans les servlets

1. **Ajout de mappages entre les types d'URL et les servlets.**
   Utiliser les éléments **`<servlet-mapping>`** du fichier **`web.xml`** pour mapper les servlets individuels à des types d'URL. L'exemple suivant mappe le serveur nommé **`DisplayOpResult`** au type d'URL **`/DisplayOpResult`**.

   ```
   <servlet-mapping>
       <servlet-name>DisplayOpResult</servlet-name>
       <url-pattern>/DisplayOpResult</url-pattern>
   </servlet-mapping>
   ```

2. **Ajout des contraintes de sécurité aux types d'URL.**
   Pour mapper le type d'URL avec une contrainte de sécurité, utilisez un **`<security-constraint>`**. L'exemple suivant limite l'accès d'un type d'URL **`/DisplayOpResult`** afin qu'il soit accessible aux principaux ayant pour rôle **`eap_admin`**. Le rôle doit être présent dans le domaine de sécurité.

   ```
   <security-constraint>
    <display-name>Restrict access to role eap_admin</display-name>
    <web-resource-collection>
     <web-resource-name>Restrict access to role eap_admin</web-
   resource-name>
     <url-pattern>/DisplayOpResult/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
     <role-name>eap_admin</role-name>
    </auth-constraint>
   </security-constraint>
   ```

```
<security-role>
  <role-name>eap_admin</role-name>
</security-role>


<login-config>
    <auth-method>BASIC</auth-method>
</login-config>
```

Vous aurez besoin d'indiquer la méthode d'authentification, qui peut être une des méthodes suivantes : **`BASIC, FORM, DIGEST, CLIENT-CERT, SPNEGO.`**. Cet exemple utilise l'authentification **`BASIC`**.

3. **Indiquez le domaine de sécurité et le fichier `jboss-web.xml` du WAR.**
   Ajoutez le domaine de sécurité au fichier **`jboss-Web.xml`** du WAR afin de connecter les servlets au domaine de la sécurité configuré sachant comment authentifier et autoriser les principaux selon les contraintes de sécurité. L'exemple suivant utilise le domaine de sécurité appelé **`acme_domain`**.

```
<jboss-web>
 ...
 <security-domain>acme_domain</security-domain>
 ...
</jboss-web>
```

Exemple 16.1. Exemple `web.xml` avec la sécurité basée rôle configurée.

```
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
         version="3.0">

<display-name>Use Role-Based Security In Servlets</display-name>

<welcome-file-list>
  <welcome-file>/index.jsp</welcome-file>
</welcome-file-list>

<servlet-mapping>
    <servlet-name>DisplayOpResult</servlet-name>
    <url-pattern>/DisplayOpResult</url-pattern>
</servlet-mapping>

<security-constraint>
  <display-name>Restrict access to role eap_admin</display-name>
    <web-resource-collection>
      <web-resource-name>Restrict access to role eap_admin</web-
resource-name>
      <url-pattern>/DisplayOpResult/*</url-pattern>
      </web-resource-collection>
      <auth-constraint>
```

```
        <role-name>eap_admin</role-name>
      </auth-constraint>
    </security-constraint>

    <security-role>
      <role-name>eap_admin</role-name>
    </security-role>

    <login-config>
        <auth-method>BASIC</auth-method>
    </login-config>

</web-app>
```

Rapporter un bogue

16.2.10. Utilisation d'une authentification système de tierce partie pour votre application

Vous pouvez intégrer des systèmes de sécurité de tierce partie avec JBoss EAP 6. Ces types de systèmes sont habituellement basés sur des tokens. Le système externe effectue l'authentification et passe un token à l'application web via les en-têtes de demande. Ceci est souvent dénommé *authentification de périmètre*. Pour configurer l'authentification de périmètre dans votre application, ajoutez une valve d'authentification personnalisée. Si vous avez une valve de fournisseur tiers, veillez à ce qu'elle soit sur votre chemin de classe et suivez les exemples ci-dessous, ainsi que la documentation pour votre module d'authentification tiers.

NOTE

L'emplacement pour la configuration des valves a changé dans JBoss EAP 6. Il n'y a plus de descripteur de déploiement **`context.xml`**. Les valves sont configurées directement dans le descripteur **`jboss-web.xml`** à la place. Le fichier **`context.xml`** peut maintenant être ignoré.

Exemple 16.2. Valve d'authentification de base

```
<jboss-web>
  <valve>
    <class-
name>org.jboss.security.negotiation.NegotiationAuthenticator</class-
name>
  </valve>
</jboss-web>
```

Cette valve est utilisée pour les SSO basés-Kerberos. Montre également les modèles les plus simples d'indication d'authentificateurs de tierce partie pour votre application web.

Exemple 16.3. Personnaliser une valve avec des attributs d'en-tête

```
<jboss-web>
  <valve>
```

```
    <class-
name>org.jboss.web.tomcat.security.GenericHeaderAuthenticator</class-
name>
    <param>
      <param-name>httpHeaderForSSOAuth</param-name>
      <param-value>sm_ssoid,ct-remote-user,HTTP_OBLIX_UID</param-value>
    </param>
    <param>
      <param-name>sessionCookieForSSOAuth</param-name>
      <param-value>SMSESSION,CTSESSION,ObSSOCookie</param-value>
    </param>
  </valve>
</jboss-web>
```

Cet exemple montre comment définir des attributs personnalisés sur votre valve. L'authentificateur vérifie la présence de l'en-tête ID et de la clé de session, puis les passe au framework JAAS qui actionne la couche de sécurité, comme le nom d'utilisateur et le mot de passe. Vous avez besoin d'un module de connexion JAAS personnalisé qui puisse traiter le nom d'utilisateur et le mot de passe tout en remplissant le sujet par les rôles qui conviennent. Si aucune valeur d'en-tête ne correspond aux valeurs configurées, les sémantiques de d'authentification basée sur les formulaires réguliers s'appliqueront.

Rédaction d'un authentificateur personnalisé

Rédiger vous-même votre authentificateur est en dehors de la portée de ce document. Cependant, le code Java suivant est fourni comme exemple.

Exemple 16.4. GenericHeaderAuthenticator.java

```
/*
 * JBoss, Home of Professional Open Source.
 * Copyright 2006, Red Hat Middleware LLC, and individual contributors
 * as indicated by the @author tags. See the copyright.txt file in the
 * distribution for a full listing of individual contributors.
 *
 * This is free software; you can redistribute it and/or modify it
 * under the terms of the GNU Lesser General Public License as
 * published by the Free Software Foundation; either version 2.1 of
 * the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
 * Lesser General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public
 * License along with this software; if not, write to the Free
 * Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA
 * 02110-1301 USA, or see the FSF site: http://www.fsf.org.
 */

package org.jboss.web.tomcat.security;

import java.io.IOException;
import java.security.Principal;
```

```
import java.util.StringTokenizer;

import javax.management.JMException;
import javax.management.ObjectName;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.catalina.Realm;
import org.apache.catalina.Session;
import org.apache.catalina.authenticator.Constants;
import org.apache.catalina.connector.Request;
import org.apache.catalina.connector.Response;
import org.apache.catalina.deploy.LoginConfig;
import org.jboss.logging.Logger;

import org.jboss.as.web.security.ExtendedFormAuthenticator;

/**
 * JBAS-2283: Provide custom header based authentication support
 *
 * Header Authenticator that deals with userid from the request header
Requires
 * two attributes configured on the Tomcat Service - one for the http
header
 * denoting the authenticated identity and the other is the SESSION
cookie
 *
 * @author <a href="mailto:Anil.Saldhana@jboss.org">Anil Saldhana</a>
 * @author <a href="mailto:sguilhen@redhat.com">Stefan Guilhen</a>
 * @version $Revision$
 * @since Sep 11, 2006
 */
public class GenericHeaderAuthenticator extends
ExtendedFormAuthenticator {
  protected static Logger log = Logger
      .getLogger(GenericHeaderAuthenticator.class);

  protected boolean trace = log.isTraceEnabled();

  // JBAS-4804: GenericHeaderAuthenticator injection of ssoid and
  // sessioncookie name.
  private String httpHeaderForSSOAuth = null;

  private String sessionCookieForSSOAuth = null;

  /**
   * <p>
   * Obtain the value of the <code>httpHeaderForSSOAuth</code>
attribute. This
   * attribute is used to indicate the request header ids that have to
be
   * checked in order to retrieve the SSO identity set by a third party
   * security system.
   * </p>
   *
```

```
    * @return a <code>String</code> containing the value of the
    *         <code>httpHeaderForSSOAuth</code> attribute.
    */
  public String getHttpHeaderForSSOAuth() {
    return httpHeaderForSSOAuth;
  }

  /**
    * <p>
    * Set the value of the <code>httpHeaderForSSOAuth</code> attribute.
This
    * attribute is used to indicate the request header ids that have to
be
    * checked in order to retrieve the SSO identity set by a third party
    * security system.
    * </p>
    *
    * @param httpHeaderForSSOAuth
    *            a <code>String</code> containing the value of the
    *            <code>httpHeaderForSSOAuth</code> attribute.
    */
  public void setHttpHeaderForSSOAuth(String httpHeaderForSSOAuth) {
    this.httpHeaderForSSOAuth = httpHeaderForSSOAuth;
  }

  /**
    * <p>
    * Obtain the value of the <code>sessionCookieForSSOAuth</code>
attribute.
    * This attribute is used to indicate the names of the SSO cookies
that may
    * be present in the request object.
    * </p>
    *
    * @return a <code>String</code> containing the names (separated by a
    *         <code>','</code>) of the SSO cookies that may have been
set by a
    *         third party security system in the request.
    */
  public String getSessionCookieForSSOAuth() {
    return sessionCookieForSSOAuth;
  }

  /**
    * <p>
    * Set the value of the <code>sessionCookieForSSOAuth</code>
attribute. This
    * attribute is used to indicate the names of the SSO cookies that may
be
    * present in the request object.
    * </p>
    *
    * @param sessionCookieForSSOAuth
    *            a <code>String</code> containing the names (separated
by a
    *            <code>','</code>) of the SSO cookies that may have been
```

```
set by
   *            a third party security system in the request.
   */
  public void setSessionCookieForSSOAuth(String sessionCookieForSSOAuth)
{
    this.sessionCookieForSSOAuth = sessionCookieForSSOAuth;
  }

  /**
   * <p>
   * Creates an instance of <code>GenericHeaderAuthenticator</code>.
   * </p>
   */
  public GenericHeaderAuthenticator() {
    super();
  }

  public boolean authenticate(Request request, HttpServletResponse
response,
      LoginConfig config) throws IOException {
    log.trace("Authenticating user");

    Principal principal = request.getUserPrincipal();
    if (principal != null) {
      if (trace)
        log.trace("Already authenticated '" + principal.getName() +
"'");
      return true;
    }

    Realm realm = context.getRealm();
    Session session = request.getSessionInternal(true);

    String username = getUserId(request);
    String password = getSessionCookie(request);

    // Check if there is sso id as well as sessionkey
    if (username == null || password == null) {
      log.trace("Username is null or password(sessionkey) is
null:fallback to form auth");
      return super.authenticate(request, response, config);
    }
    principal = realm.authenticate(username, password);

    if (principal == null) {
      forwardToErrorPage(request, response, config);
      return false;
    }

    session.setNote(Constants.SESS_USERNAME_NOTE, username);
    session.setNote(Constants.SESS_PASSWORD_NOTE, password);
    request.setUserPrincipal(principal);

    register(request, response, principal, HttpServletRequest.FORM_AUTH,
        username, password);
    return true;
```

```
    }

    /**
     * Get the username from the request header
     *
     * @param request
     * @return
     */
    protected String getUserId(Request request) {
      String ssoid = null;
      // We can have a comma-separated ids
      String ids = "";
      try {
        ids = this.getIdentityHeaderId();
      } catch (JMException e) {
        if (trace)
          log.trace("getUserId exception", e);
      }
      if (ids == null || ids.length() == 0)
        throw new IllegalStateException(
            "Http headers configuration in tomcat service missing");

      StringTokenizer st = new StringTokenizer(ids, ",");
      while (st.hasMoreTokens()) {
        ssoid = request.getHeader(st.nextToken());
        if (ssoid != null)
          break;
      }
      if (trace)
        log.trace("SSOID-" + ssoid);
      return ssoid;
    }

    /**
     * Obtain the session cookie from the request
     *
     * @param request
     * @return
     */
    protected String getSessionCookie(Request request) {
      Cookie[] cookies = request.getCookies();
      log.trace("Cookies:" + cookies);
      int numCookies = cookies != null ? cookies.length : 0;

      // We can have comma-separated ids
      String ids = "";
      try {
        ids = this.getSessionCookieId();
        log.trace("Session Cookie Ids=" + ids);
      } catch (JMException e) {
        if (trace)
          log.trace("checkSessionCookie exception", e);
      }
      if (ids == null || ids.length() == 0)
        throw new IllegalStateException(
            "Session cookies configuration in tomcat service missing");
```

```
    StringTokenizer st = new StringTokenizer(ids, ",");
    while (st.hasMoreTokens()) {
      String cookieToken = st.nextToken();
      String val = getCookieValue(cookies, numCookies, cookieToken);
      if (val != null)
        return val;
    }
    if (trace)
      log.trace("Session Cookie not found");
    return null;
  }

  /**
   * Get the configured header identity id in the tomcat service
   *
   * @return
   * @throws JMException
   */
  protected String getIdentityHeaderId() throws JMException {
    if (this.httpHeaderForSSOAuth != null)
      return this.httpHeaderForSSOAuth;
    return (String) mserver.getAttribute(new ObjectName(
        "jboss.web:service=WebServer"), "HttpHeaderForSSOAuth");
  }

  /**
   * Get the configured session cookie id in the tomcat service
   *
   * @return
   * @throws JMException
   */
  protected String getSessionCookieId() throws JMException {
    if (this.sessionCookieForSSOAuth != null)
      return this.sessionCookieForSSOAuth;
    return (String) mserver.getAttribute(new ObjectName(
        "jboss.web:service=WebServer"), "SessionCookieForSSOAuth");
  }

  /**
   * Get the value of a cookie if the name matches the token
   *
   * @param cookies
   *            array of cookies
   * @param numCookies
   *            number of cookies in the array
   * @param token
   *            Key
   * @return value of cookie
   */
  protected String getCookieValue(Cookie[] cookies, int numCookies,
      String token) {
    for (int i = 0; i < numCookies; i++) {
      Cookie cookie = cookies[i];
      log.trace("Matching cookieToken:" + token + " with cookie name="
          + cookie.getName());
```

```
        if (token.equals(cookie.getName())) {
          if (trace)
            log.trace("Cookie-" + token + " value=" + cookie.getValue());
          return cookie.getValue();
        }
      }
      return null;
    }
  }
```

Rapporter un bogue

16.3. DOMAINES DE SÉCURITÉ

16.3.1. Domaines de sécurité

Un *domaine de sécurité* est une série de mappages entre les utilisateurs et les mots de passe, les utilisateurs et les rôles. Les domaines de sécurité représentent un mécanisme permettant d'ajouter l'authentification et l'autorisation à vos applications Web et EJB. JBoss EAP 6 fournit deux domaines de sécurité par défaut :

- **`ManagementRealm`** stocke les informations d'authentification pour l'API de gestion, qui fournit les fonctionnalités pour le Management CLI et la Console de gestion sur le web. Il fournit un système d'authentification pour gérer JBoss EAP 6 . Vous pouvez également utiliser le **`ManagementRealm`** si votre application a besoin des mêmes règles commerciales que vous utilisez pour l'API de gestion, lors de son authentification.
- **`ApplicationRealm`** stocke l'utilisateur, le mot de passe et les informations de rôle pour les applications Web et les EJB.

Chaque domaine est stocké dans deux fichiers du système de fichiers :

- ***`REALM`*`-users.properties`** stocke les mots de passe et les mots de passe hachés.
- ***`REALM`*`-roles.properties`** stocke les mappages user-to-role.
- **`mgmt-groups.properties`** stocke les fichiers de mappage user-to-role de **`ManagementRealm`**.

Les fichiers de propriété sont stockés dans les répertoires **`domain/configuration/`** et **`standalone/configuration/`**. Les fichiers sont inscrits simultanément par la commande **`add-user.sh`** ou **`add-user.bat`**. Quand vous exécutez la commande, la première décision est de décider dans quel domaine ajouter votre premier utilisateur.

Rapporter un bogue

16.3.2. Ajout d'un domaine de sécurité

1. **Exécuter l'interface CLI**
   Démarrez par la commande **`jboss-cli.sh`** ou **`jboss-cli.bat`** et connectez-vous au serveur.

2. **Créer le nouveau domaine de sécurité lui-même.**

Exécutez la commande suivante pour créer un nouveau domaine de sécurité nommé **`MyDomainRealm`** sur un contrôleur de domaine ou sur un serveur autonome.

Pour une instance de domaine, utiliser la commande suivante :

```
/host=master/core-service=management/security-
realm=MyDomainRealm:add()
```

Pour une instance autonome, utiliser la commande suivante :

```
/core-service=management/security-realm=MyDomainRealm:add()
```

3. **Créer les références du fichier de propriétés qui stocke les informations sur le nouveau rôle.**
Exécutez la commande suivante pour créer un pointeur au fichier nommé **`myfile.properties`**, qui contiendra les propriétés attachées au nouveau rôle.

NOTE

Le fichier de propriétés nouvellement créées n'est pas géré par les scripts **`add-user.sh`** et **`add-user.bat`** inclus. Il devra être administré en externe.

Pour une instance de domaine, utiliser la commande suivante :

```
/host=master/core-service=management/security-
realm=MyDomainRealm/authentication=properties:add(path=myfile.proper
ties)
```

Pour une instance autonome, utiliser la commande suivante :

```
/core-service=management/security-
realm=MyDomainRealm/authentication=properties:add(path=myfile.proper
ties)
```

Résultat

Votre nouveau domaine de sécurité sera créé. Lorsque vous ajoutez des utilisateurs et des rôles à ce nouveau domaine, l'information va être stockée dans un fichier séparé des domaines de sécurité par défaut. Vous pouvez gérer ce nouveau fichier à l'aide de vos propres applications ou procédures.

Rapporter un bogue

16.3.3. Ajout d'un utilisateur à un domaine de sécurité

1. **Éxécuter la commande `add-user.sh` ou `add-user.bat`.**
Ouvrez un terminal (shell) et changez de répertoire ***`EAP_HOME`*`/bin/`**. Si vous exécutez Red Hat Enterprise Linux ou un autre système d'exploitation style UNIX, exécutez **`add-user.sh`**. Si vous exécutez sur un serveur Microsoft Windows, exécutez **`add-user.bat`**.

2. **Choisir d'ajouter un utilisateur de gestion ou un utilisateur d'application.**
Pour cette procédure, saisir **b** pour ajouter un utilisateur d'application.

3. **Choisir le domaine dans lequel l'utilisateur sera ajouté.**

Par défaut, le seul domaine disponible est **`ApplicationRealm`**. Si vous avez ajouté un domaine personnalisé, vous pouvez saisir son nom à la place.

4. **Saisir le nom d'utilisateur, le mot de passe et les rôles lorsque vous y serez invité.** Saisir le nom d'utilisateur, le mot de passe et les rôles lorsque vous y serez invité. Vérifiez votre choix en tapant **`yes`**, ou **`no`** pour annuler les changements. Les changements sont inscrits dans les fichiers de propriétés du domaine de sécurité.

Rapporter un bogue

16.4. SÉCURITÉ DES APPLICATIONS EJB

16.4.1. Identité Sécurité

16.4.1.1. L'identité de sécurité EJB

L'*identité de sécurité*, connue également sous le nom *identité d'invocation*, se réfère à la balise **`<security-identity>`** qui se trouve dans la configuration de la sécurité. Il s'agit d'une référence à l'identité qu'un autre EJB doit utiliser quand il invoque des méthodes ou des composants.

L'identité d'invocation peut soit correspondre à l'appelant actuel, ou correspondre à un rôle spécifique. Dans le premier cas, la balise **`<use-caller-identity>`** sera présente, et dans le second cas, la balise **`<run-as>`** sera utilisée.

Pour plus d'information sur la configuration de l'identité de sécurité sur un EJB, voir Section 16.4.1.2, « Définir l'identité de sécurité d'un EJB ».

Rapporter un bogue

16.4.1.2. Définir l'identité de sécurité d'un EJB

Exemple 16.5. Définir l'identité de sécurité d'un EJB pour que ce soit la même que celle de l'appelant

Cet exemple définit l'identité de sécurité pour les invocations de méthode faîtes par un EJB de façon à ce qu'elle soit la même identité que celle de l'appelant actuel. Ce comportement correspond au comportement par défaut si vous ne spécifiez pas de déclaration d'élément **`<security-identity>`**.

```
<ejb-jar>
  <enterprise-beans>
  <session>
  <ejb-name>ASessionBean</ejb-name>
  <!-- ... -->
  <security-identity>
    <use-caller-identity/>
  </security-identity>
  </session>
  <!-- ... -->
  </enterprise-beans>
</ejb-jar>
```

Exemple 16.6. Définir l'idendité de sécurité d'un EJB à un rôle spécifique

Pour définir l'id de sécurité à un rôle spécifique, utiliser **<run-as>** et les balises **<role-name>** dans la balise **<security-identity>**.

```
<ejb-jar>
  <enterprise-beans>
  <session>
  <ejb-name>RunAsBean</ejb-name>
  <!-- ... -->
  <security-identity>
    <run-as>
    <description>A private internal role</description>
    <role-name>InternalRole</role-name>
    </run-as>
  </security-identity>
  </session>
  </enterprise-beans>
  <!-- ... -->
</ejb-jar>
```

Par défaut, quand vous utilisez **<run-as>**, un principal nommé **anonymous** est assigné aux appels sortants. Pour assigner un autre principal, utiliser **<run-as-principal>**.

```
<session>
    <ejb-name>RunAsBean</ejb-name>
    <security-identity>
        <run-as-principal>internal</run-as-principal>
    </security-identity>
</session>
```

NOTE

Vous pouvez utiliser les éléments **<run-as>** et **<run-as-principal>** à l'intérieur d'un élément de servlet.

Voir également :

- Section 16.4.1.1, « L'identité de sécurité EJB »
- Section 18.2, « Référence de paramètre de sécurité EJB »

Rapporter un bogue

16.4.2. Permissions de méthodes EJB

16.4.2.1. Permissions de méthodes EJB

EJB fournit une déclaration d'élément **<method-permisison>**. Cette déclaration définit les rôles qui sont autorisés à appeler des méthodes de l'interface EJB. Vous pouvez définir des permissions pour les combinaisons suivantes :

- Toutes les méthodes d'interface de composant ou d'accueil de l'EJB nommé
- Une méthode spécifiée d'interface de composant ou d'accueil de l'EJB nommé
- Une méthode spécifiée à l'intérieur d'un ensemble de méthodes avec un nom surchargé

Pour obtenir des exemples, voir Section 16.4.2.2, « Utilisation des permissions de méthodes EJB ».

Rapporter un bogue

16.4.2.2. Utilisation des permissions de méthodes EJB

Aperçu

L'élément **<method-permission>** définit les roles logiques qui peuvent accéder aux méthodes EJB définies par les éléments **<method>**. Un certain nombre d'exemples expliquent la syntaxe XML. Plusieurs énoncés de method-permission peuvent être présents, et avoir un effet cumulatif. L'élément **<method-permission>** est un dépendant de l'élément **<assembly-descriptor>** du descripteur **<ejb-jar>**.

La syntaxe XML est une alternative aux annotations pour les permissions de méthode EJB.

Exemple 16.7. Permet aux rôles d'accéder à toutes les méthodes d'un EJB

```
<method-permission>
  <description>The employee and temp-employee roles may access any
method
  of the EmployeeService bean </description>
  <role-name>employee</role-name>
  <role-name>temp-employee</role-name>
  <method>
    <ejb-name>EmployeeService</ejb-name>
    <method-name>*</method-name>
  </method>
</method-permission>
```

Exemple 16.8. Permet aux rôles d'accéder uniquement à des méthodes spécifiques d'un EJB, et de déterminer quels paramètres de méthode peuvent être passés.

```
<method-permission>
  <description>The employee role may access the findByPrimaryKey,
  getEmployeeInfo, and the updateEmployeeInfo(String) method of
  the AcmePayroll bean </description>
  <role-name>employee</role-name>
  <method>
 <ejb-name>AcmePayroll</ejb-name>
 <method-name>findByPrimaryKey</method-name>
  </method>
  <method>
 <ejb-name>AcmePayroll</ejb-name>
 <method-name>getEmployeeInfo</method-name>
  </method>
  <method>
```

```
 <ejb-name>AcmePayroll</ejb-name>
 <method-name>updateEmployeeInfo</method-name>
 <method-params>
   <method-param>java.lang.String</method-param>
 </method-params>
  </method>
</method-permission>
```

Exemple 16.9. Permet à n'importe quel utilisateur authentifié d'accéder aux méthodes des EJB

Utiliser l'élément **<unchecked/>** permet à un utilisateur authentifié d'utiliser les méthodes spécifiées.

```
<method-permission>
  <description>Any authenticated user may access any method of the
  EmployeeServiceHelp bean</description>
  <unchecked/>
  <method>
 <ejb-name>EmployeeServiceHelp</ejb-name>
 <method-name>*</method-name>
  </method>
</method-permission>
```

Exemple 16.10. Exclut totalement l'utilisation de certaines méthodes EJB

```
<exclude-list>
  <description>No fireTheCTO methods of the EmployeeFiring bean may be
  used in this deployment</description>
  <method>
 <ejb-name>EmployeeFiring</ejb-name>
 <method-name>fireTheCTO</method-name>
  </method>
</exclude-list>
```

Exemple 16.11. Un `<assembly-descriptor>` complet contenant plusieurs blocs de `<method-permission>`

```
<ejb-jar>
    <assembly-descriptor>
        <method-permission>
            <description>The employee and temp-employee roles may
access any
                method of the EmployeeService bean </description>
            <role-name>employee</role-name>
            <role-name>temp-employee</role-name>
            <method>
                <ejb-name>EmployeeService</ejb-name>
                <method-name>*</method-name>
            </method>
        </method-permission>
```

```
        <method-permission>
            <description>The employee role may access the
findByPrimaryKey,
                getEmployeeInfo, and the updateEmployeeInfo(String)
method of
                the AcmePayroll bean </description>
            <role-name>employee</role-name>
            <method>
                <ejb-name>AcmePayroll</ejb-name>
                <method-name>findByPrimaryKey</method-name>
            </method>
            <method>
                <ejb-name>AcmePayroll</ejb-name>
                <method-name>getEmployeeInfo</method-name>
            </method>
            <method>
                <ejb-name>AcmePayroll</ejb-name>
                <method-name>updateEmployeeInfo</method-name>
                <method-params>
                    <method-param>java.lang.String</method-param>
                </method-params>
            </method>
        </method-permission>
        <method-permission>
            <description>The admin role may access any method of the
                EmployeeServiceAdmin bean </description>
            <role-name>admin</role-name>
            <method>
                <ejb-name>EmployeeServiceAdmin</ejb-name>
                <method-name>*</method-name>
            </method>
        </method-permission>
        <method-permission>
            <description>Any authenticated user may access any method
of the
                EmployeeServiceHelp bean</description>
            <unchecked/>
            <method>
                <ejb-name>EmployeeServiceHelp</ejb-name>
                <method-name>*</method-name>
            </method>
        </method-permission>
        <exclude-list>
            <description>No fireTheCTO methods of the EmployeeFiring
bean may be
                used in this deployment</description>
            <method>
                <ejb-name>EmployeeFiring</ejb-name>
                <method-name>fireTheCTO</method-name>
            </method>
        </exclude-list>
    </assembly-descriptor>
</ejb-jar>
```

Rapporter un bogue

16.4.3. Annotations de sécurité EJB

16.4.3.1. Les annotations de sécurité EJB

Les EJB utilisent des annotations de sécurité pour faire passer des informations de sécurité au déployeur. Elles comprennent :

@DeclareRoles

Déclare quels rôles sont disponibles.

@RunAs

Configure l'identification de sécurité propagée d'un composant.

Pour plus d'information, voir Section 16.4.3.2, « Utilisation des annotations de sécurité EJB ».

Rapporter un bogue

16.4.3.2. Utilisation des annotations de sécurité EJB

Aperçu

Vous pouvez utiliser deux descripteurs XML ou des annotations pour contrôler quels rôles de sécurité sont en mesure d'appeler des méthodes dans votre Enterprise JavaBeans (EJB). Pour plus d'informations sur l'utilisation des descripteurs XML, reportez-vous à Section 16.4.2.2, « Utilisation des permissions de méthodes EJB ».

Annotations pour contrôler les permissions de sécurité des EJB

@DeclareRoles

Utiliser @DeclareRoles pour déterminer sur quels rôles de sécurité vérifier les permissions. Si aucun @DeclareRoles n'est présent, la liste sera édifié automatiquement à partir de l'annotation @RolesAllowed.

@RolesAllowed, @PermitAll, @DenyAll

Utiliser @RolesAllowed pour lister les rôles autorisés pour accéder à une méthode ou à des méthodes. Utiliser @PermitAll ou @DenyAll pour permettre ou refuser à tous les roles d'utiliser une méthode ou des méthodes.

@RunAs

Utiliser @RunAs pour spécifier un role utilisé par une méthode lors d'appels en provenance d'une méthode annotée.

Exemple 16.12. Exemple d'annotations de sécurité

```
@Stateless
@RolesAllowed({"admin"})
@SecurityDomain("other")
public class WelcomeEJB implements Welcome {
 @PermitAll
 public String WelcomeEveryone(String msg) {
  return "Welcome to " + msg;
```

```
    }
    @RunAs("tempemployee")
    public String GoodBye(String msg) {
        return "Goodbye, " + msg;
    }
    public String GoodbyeAdmin(String msg) {
     return "See you later, " + msg;
    }
}
```

Avec ce code, tous les rôles peuvent accéder à la méthode **`WelcomeEveryone`**. La méthode **`GoodBye`** exécute sous la forme du rôle **`tempemployee`** pour les appels. Seul le rôle **`admin`** peut accéder à la méthode **`GoodbyeAdmin`**, ou à toute autre méthode sans annotation de sécurité.

Rapporter un bogue

16.4.4. Accès à distance aux EJB

16.4.4.1. Remote Method Access

JBoss Remoting est le framework qui fournit un accès à distance aux EJBs, JMX MBeans, et autres services similaires. Fonctionne avec les types de transport suivants, avec ou sans SSL :

Types de services pris en charge

- Socket / Secure Socket
- RMI / RMI sur SSL
- HTTP / HTTPS
- Servlet / Secure Servlet
- Bisocket / Secure Bisocket

JBoss Remoting fournit aussi automatic discovery via Multicast ou JNDI.

Il est utilisé par bon nombre des sous-systèmes au sein de la plateforme JBoss EAP 6 et permet également de concevoir, d'implémenter et de déployer des services pouvant être appelés à distance par les clients sur plusieurs mécanismes de transport différents. Il vous permet également d'accéder aux services existants dans JBoss EAP 6.

Data Marshalling

Le système Remoting d'accès distant fournit également des services de marshalling et unmarshalling de données. Le marshaling de données désigne la capacité de déplacer en toute sécurité des données au-delà des limites de réseau et de la plate-forme, afin qu'un système séparé puisse effectuer des tâches dessus. Le travail est ensuite renvoyé vers le système d'origine et se comporte comme s'il avait eu lieu localement.

Aperçu de l'architecture

Lorsque vous créez une application cliente qui utilise Remoting, vous indiquez à votre application de communiquer avec le serveur en la configurant pour qu'elle utilise un type spécial de localisateur de ressources appelé un **`InvokerLocator`**, qui est une chaîne simple avec un format de type URL. Le

serveur écoute les requêtes des ressources distantes sur un **`connecteur`**, qui est configuré comme faisant partie du sous-système **`remoting`**. Le **`connecteur`** transmet la demande à un **`ServerInvocationHandler`** configuré. Chaque **`ServerInvocationHandler`** implémente une méthode **`invoke(InvocationRequest)`** qui sait comment gérer la demande.

Le framework JBoss Remoting contient trois couches qui se miroitent les unes par rapport aux autres côté client et côté serveur.

Couches de framework JBoss Remoting

- L'utilisateur interagit avec la couche externe. Côté client, la couche externe est la classe **`Client`**, qui envoie des requêtes d'invocation. Côté serveur, c'est InvocationHandler, mis en œuvre par l'utilisateur, qui reçoit des demandes d'invocation.
- Le transport est contrôlé par la couche d'invocateur.
- La couche inférieure contient le marshaller et le unmarshaller, qui convertit les formats de données en formats de transmission.

Rapporter un bogue

16.4.4.2. Remoting Callbacks

Lorsqu'un client Remoting (à distance) demande des informations du serveur, il peut y avoir un blocage et il faut alors attendre que le serveur réponde, mais ce n'est pas un comportement idéal. Pour permettre au client d'être à l'écoute des événements asynchrones sur le serveur et pour pouvoir continuer à faire d'autre tâches en attendant que le serveur complète la requête, votre application peut demander au serveur d'envoyer une notification quand il aura fini. C'est ce que l'on appelle un callback. Un client peut s'ajouter comme un écouteur d'événements asynchrones générés au nom d'un autre client, aussi bien. Il y a deux choix différents pour recevoir des callbacks : «pull callbacks» ou «push callbacks». Les clients vérifient les «pull callbacks» de façon synchrone, mais écoutent passivement les «push callbacks».

En substance, un callback fonctionne de cette façon: le serveur envoie une **`InvocationRequest`** au client. Votre code côté serveur fonctionne de la même façon que le rappel soit synchrone ou asynchrone. Seul le client a besoin de connaître la différence. InvocationRequest du serveur envoie un **`responseObject`** au client. Il s'agit de la charge que le client a demandé. C'est peut-être une réponse directe à une demande ou à une notification d'événement.

Votre serveur suit aussi les listeners à l'aide d'un objet **`m_listeners`**. Il contient une liste de tous les listeners qui ont été ajoutés à votre server handler. L'interface du **`ServerInvocationHandler`** inclut des méthodes qui vous permettent de gérer cette liste.

Le client gère les «pull callbacks» et les «push callbacks» de différentes manières. Dans les deux cas, il doit implémenter un callback handler. Un callback handler est une implémentation de l'interface **`org.jboss.remoting.InvokerCallbackHandler`**, qui traite les données de callback. Après l'implémentation du callback handler, soit vous vous ajoutez vous-même, en tant que listener de «pull callback», ou bien, vous installez un serveur de callbacks pour un «push callback».

Pull Callbacks

Pour un «pull callback», votre client s'ajoute à la liste du serveur des listeners à l'aide de la méthode **`Client.addListener()`**. Ensuite, il interroge le serveur périodiquement au sujet de l'exécution synchrone des données de callback. Ce sondage est effectué à l'aide de **`Client.getCallbacks()`**.

Push Callback

Un «push callback» requiert que votre application cliente exécute elle-même son propre InvocationHandler. Pour ce faire, vous devez exécuter un service Remoting sur le client lui-même. Ceci s'appelle un *callback server*. Le callback server accepte les requêtes entrantes de façon asynchrone et les traite pour l'auteur de la demande (dans ce cas, le serveur). Pour inscrire le callback server de votre client avec le serveur principal, passez l'argument **`InvokerLocator`** du callback server comme deuxième argument à la méthode **`addListener`**.

Rapporter un bogue

16.4.4.3. Remoting Server Detection

Les clients et les serveurs d'accès distant peuvent se détecter les uns les autres automatiquement à l'aide de JNDI ou Multicast. Un détecteur Remoting Detector est ajouté aux client et serveur, et un NetworkRegistry est ajoutée au client.

Le détecteur côté serveur scanne périodiquement InvokerRegistry et extrait tous les invocateurs de serveur qu'il a créés. Il utilise ces informations pour publier un message de détection, qui contient le localisateur et les sous-systèmes pris en charge par chaque invocateur de serveur. Il publie ce message via une multidiffusion ou une liaison vers un serveur JNDI.

Côté client, le détecteur reçoit le message de multidiffusion ou interroge périodiquement le serveur JNDI pour récupérer les messages de détection. Si le détecteur remarque qu'un message de détection est pour un serveur d'accès distant nouvellement détecté, il l'inscrit dans NetworkRegistry. Le détecteur met également à jour NetworkRegistry s'il détecte qu'un serveur n'est plus disponible

Rapporter un bogue

16.4.4.4. Configurer le sous-système de JBoss Remoting

Aperçu

JBoss Remoting a trois éléments configurables de niveau supérieur : le pool de worker threads, un ou plusieurs connecteurs et une série de liens URI locaux et distants. Cette rubrique propose une explication pour chaque élément configurable, des exemples de commandes CLI pour savoir comment configurer chaque élément et un exemple XML d'un sous-système entièrement configuré. Cette configuration s'applique uniquement au serveur. La plupart des gens n'auront pas à configurer le sous-système de communication à distance, sauf s'ils utilisent des connecteurs personnalisés pour leurs propres applications. Les applications qui agissent comme des clients Remoting, comme les EJB, nécessitent une configuration distincte pour se connecter à un connecteur spécifique.

NOTE

La configuration du sous-système Remoting n'est pas exposée à la console de gestion sur web, mais est entièrement configurable de l'interface CLI en ligne de commande. Il n'est pas recommandé de modifier le code XML à la main.

Adaptation des commandes CLI

Les commandes CLI sont formulées pour un domaine géré, lorsque vous configurez le profil **`par défaut`**. Pour configurer un profil différent, changez-en le nom. Pour un serveur autonome, omettre la section **`/profile=default`** de la commande.

Configuration en dehors du sous-système Remoting

Il y a un certain nombre d'aspects de configuration qui sont en dehors du sous-système **`remoting`** :

Network Interface

L'interface de réseau qui est utilisée par le sous-système **`remoting`** est l'interface **`unsecure`** définie dans **`domain/configuration/domain.xml`** ou dans **`standalone/configuration/standalone.xml`**.

```
<interfaces>
   <interface name="management"/>
   <interface name="public"/>
   <interface name="unsecure"/>
</interfaces>
```

La définition par-hôte de l'interface **`unsecure`** est définie dans **`host.xml`** dans le même répertoire que **`domain.xml`** ou **`standalone.xml`**. Cette interface est également utilisée par plusieurs autres sous-systèmes. Soyez vigilants quand vous la modifierez.

```
<interfaces>
   <interface name="management">
      <inet-address
value="${jboss.bind.address.management:127.0.0.1}"/>
   </interface>
   <interface name="public">
      <inet-address value="${jboss.bind.address:127.0.0.1}"/>
   </interface>
   <interface name="unsecure">
      <!-- Used for IIOP sockets in the standard configuration.
         To secure JacORB you need to setup SSL -->
      <inet-address value="${jboss.bind.address.unsecure:127.0.0.1}"/>
   </interface>
</interfaces>
```

socket-binding

La liaison-socket par défaut utilisée par le sous-système **`remoting`** se lie au port TCP 4777. Reportez-vous à la documentation sur les liaisons de socket et de groupes de liaisons de socket pour plus d'informations si vous avez besoin de procéder à une modification.

Remoting Connector Reference pour EJB

Le sous-système EJB contient une référence vers le connecteur à distance pour les invocations de méthodes à distance. Voici la configuration par défaut :

```
<remote connector-ref="remoting-connector" thread-pool-name="default"/>
```

Configuration du transport sécurisé

Les transports à distance (Remoting) utilisent StartTLS pour avoir une connexion sécurisée (HTTPS, Secure Servlet, etc.) si le client le demande. La même liaison de socket (port réseau) est utilisée pour les connexions sécurisées et non sécurisées, donc aucune configuration côté serveur supplémentaire n'est nécessaire. Le client demande le transport sécurisé ou non sécurisé, tel que le dictent ses besoins. Les composants JBoss EAP 6 qui utilisent Remoting, tels que les fournisseur JMS, EJB et ORB exigent des interfaces sécurisées par défaut.

AVERTISSEMENT

StartTLS fonctionne en activant une connexion sécurisée si le client le demande au lieu, par défaut, d'une connexion non sécurisée. StartTLS est, par nature, sensible à un exploit de style *Man in the Middle* dans lequel un attaquant intercepte la demande du client et la modifie pour demander une connexion non sécurisée. Les clients doivent avoir reçu des écritures pour échouer correctement s'ils ne reçoivent pas une connexion sécurisée, sauf si une connexion non sécurisée constitue un fall-back approprié.

Worker Thread Pool

Un Worker Thread Pool est un ensemble de threads qui peuvent traiter les tâches qui arrivent par les connecteurs Remoting. Il s'agit d'un seul élément **`<worker-thread-pool>`**, et nécessite un certain nombre d'attributs. Régler ces attributs si vous avez des timeouts de réseau, ou si vous devez limiter l'utilisation de la mémoire. Les conseils varient suivant la situation dans laquelle vous vous trouvez. Contacter Red Hat Global Support Services pour obtenir davantage d'informations.

Tableau 16.2. Attributs de Worker Thread Pool

Attribut	Description	Commande CLI
read-threads	Le nombre de read-threads à créer pour le worker à distance. La valeur par défaut est **`1`**.	**`/profile=default/subsystem=remoting/:write-attribute(name=worker-read-threads,value=1)`**
write-threads	Le nombre de write-threads à créer pour le worker à distance. La valeur par défaut est **`1`**.	**`/profile=default/subsystem=remoting/:write-attribute(name=worker-write-threads,value=1)`**
task-keepalive	Le nombre de millisecondes pour conserver les threads de tâche de workers à distance non-core vivants. La valeur par défaut est **`60`**.	**`/profile=default/subsystem=remoting/:write-attribute(name=worker-task-keepalive,value=60)`**
task-max-threads	Le nombre de maximum de threads pour le Worker Task Thread Pool distant. La valeur par défaut est **`60`**.	**`/profile=default/subsystem=remoting/:write-attribute(name=worker-task-max-threads,value=16)`**
task-core-threads	Le nombre de threads principaux pour le Worker Task Thread Pool distant. La valeur par défaut est **`4`**.	**`/profile=default/subsystem=remoting/:write-attribute(name=worker-task-core-threads,value=4)`**

Attribut	Description	Commande CLI
task-limit	Le nombre de maximum de tâches de worker distantes. La valeur par défaut est **16384**.	**/profile=default/subsystem=remoting/:write-attribute(name=worker-task-limit,value=16384)**

Connecteur

Le connecteur est le principal élément de configuration de Remoting (d'accès à distance). Les connecteurs multiples sont autorisés. Chacun se compose d'un élément **<connector>** avec plusieurs sous-éléments, ainsi que quelques attributs possibles. Le connecteur par défaut est utilisé par plusieurs sous-systèmes de JBoss EAP 6. Des paramètres spécifiques pour les éléments et les attributs de vos connecteurs personnalisés dépendent de vos applications, donc contactez Red Hat Global Support Services pour plus d'informations.

Tableau 16.3. Attributs de connecteur

Attribut	Description	Commande CLI
socket-binding	Le nom de la liaison de socket à utiliser pour ce connecteur.	**/profile=default/subsystem=remoting/connector=remoting-connector/:write-attribute(name=socket-binding,value=remoting)**
authentication-provider	Le module JASPIC (de l'anglais Java Authentication Service Provider Interface for Containers) à utiliser avec ce connecteur. Le module doit être dans le chemin de classes.	**/profile=default/subsystem=remoting/connector=remoting-connector/:write-attribute(name=authentication-provider,value=myProvider)**
security-realm	En option. Le domaine de la sécurité qui contient les utilisateurs, mots de passe et les rôles de votre application. Un EJB ou une Application Web peut authentifier sur un domaine de sécurité. **ApplicationRealm** est disponible dans une installation de JBoss EAP 6 par défaut.	**/profile=default/subsystem=remoting/connector=remoting-connector/:write-attribute(name=security-realm,value=ApplicationRealm)**

Tableau 16.4. Éléments de connecteur

Attribut	Description	Commande CLI

Attribut	Description	Commande CLI
sasl	Élément englobant des mécanismes d'authentification SASL (Simple Authentication and Security Layer)	**N/A**
propriétés	Contient un ou plusieurs éléments **<property>**, contenant chacun un attribut **name** et un attribut **value** optionnel.	**/profile=default/subsystem=remoting/connector=remoting-connector/property=myProp/:add(value=myPropValue)**

Les connexions de sortie

Vous pouvez spécifier trois types différents d'attribut de connexion sortante :

- Connexion sortante vers un URI.
- Connexion sortante locale – se connectant à une ressource locale, comme un socket.
- Connexion sortante à distance – se connectant à une ressource à distance et s'authentifiant par l'intermédiaire d'un domaine de sécurité.

Toutes les connexions sortantes sont enfermées dans un élément **<outbound-connections>**. Chacun de ces types de connexion prend un attribut **outbound-socket-binding-ref**. La connexion sortante prend un attribut **uri**. La connexion sortante prend les attributs facultatifs **username** (nom d'utilisateur) et **security-realm** (domaine de sécurité) pour l'autorisation.

Tableau 16.5. Éléments de connexion sortante

Attribut	Description	Commande CLI
outbound-connection	Connexion sortante standard	**/profile=default/subsystem=remoting/outbound-connection=my-connection/:add(uri=http://my-connection)**
local-outbound-connection	Connexion sortante en schéma implicite local:// URI.	**/profile=default/subsystem=remoting/local-outbound-connection=my-connection/:add(outbound-socket-binding-ref=remoting2)**
remote-outbound-connection	Connexions sortantes en schéma remote:// URI, utilisant l'authentification de base/digest avec un domaine de sécurité.	**/profile=default/subsystem=remoting/remote-outbound-connection=my-connection/:add(outbound-socket-binding-ref=remoting,username=myUser,security-realm=ApplicationRealm)**

Éléments SASL

Avant de définir des éléments enfants SASL, vous devez créer l'élément SASL initial. Utiliser la commande suivante :

```
/profile=default/subsystem=remoting/connector=remoting-
connector/security=sasl:add
```

Les éléments enfants de l'élément SASL sont décrits dans le tableau ci-dessous.

Attribut	Description	Commande CLI
include-mechanisms	Contient un attribut **value**, qui correspond à une liste de mécanismes SASL séparés par des espaces.	`/profile=default/subsystem=remoting/connector=remoting-connector/security=sasl:write-attribute(name=include-mechanisms,value=["DIGEST","PLAIN","GSSAPI"])`
qop	Contient un attribut **value**, qui correspond à une liste de valeurs de protection SASL séparées par des espaces, en ordre décroissant de préférence.	`/profile=default/subsystem=remoting/connector=remoting-connector/security=sasl:write-attribute(name=qop,value=["auth"])`
puissance	Contient un attribut **value**, qui correspond à une liste de valeurs de puissance cipher SASL séparées par des espaces, en ordre décroissant de préférence.	`/profile=default/subsystem=remoting/connector=remoting-connector/security=sasl:write-attribute(name=strength,value=["medium"])`
reuse-session	Contient un attribut **value**, qui correspond à une valeur booléenne. Si sur true, tente de réutiliser les sessions.	`/profile=default/subsystem=remoting/connector=remoting-connector/security=sasl:write-attribute(name=reuse-session,value=false)`

Attribut	Description	Commande CLI
server-auth	Contient un attribut **`value`**, qui correspond à une valeur booléenne. Si sur true, le serveur s'authentifie auprès du client.	`/profile=default/subsystem=remoting/connector=remoting-connector/security=sasl:write-attribute(name=server-auth,value=false)`

politique	Un élément clôturé qui contient zéro ou plusieurs des éléments suivants, prenant chacun une seule **`valeur`**.
	• forward-secrecy – indique si on a besoin de mécanismes pour implémenter la forward-secrecy (l'infiltration dans une session ne va pas forcément donner des informations sur la façon de s'infiltrer dans des sessions futures)
	• no-active – indique si les mécanismes sensibles aux attaques hors dictionnaire sont permis. Une valeur **`false`** permet, et **`true`** non.
	• no-anonymous – indique si les mécanismes qui acceptent la connexion anonyme sont permis. Une valeur de **`false`** le permet, et **`true`** non.
	• no-active – indique si les mécanismes sensibles aux attaques dictionnaire passives sont permis. Une valeur **`false`** le permet, et **`true`** non.
	• no-plain-text – indique si les mécanismes sensibles aux simples attaques dictionnaire passives sont permis. Une valeur **`false`** le permet, et **`true`** non.
	• pass-credentials – indique si les mécanismes qui font
		`/profile=default/subsystem=remoting/connector=remoting-connector/security=sasl/sasl-policy=policy:add`
		`/profile=default/subsystem=remoting/connector=remoting-connector/security=sasl/sasl-policy=policy:write-attribute(name=forward-secrecy,value=true)`
		`/profile=default/subsystem=remoting/connector=remoting-connector/security=sasl/sasl-policy=policy:write-attribute(name=no-active,value=false)`
		`/profile=default/subsystem=remoting/connector=remoting-connector/security=sasl/sasl-policy=policy:write-attribute(name=no-anonymous,value=false)`
		`/profile=default/subsystem=remoting/connector=remoting-connector/security=sasl/sasl-`

Attribut	Description	Commande CLI

	passer les identifications clients sont permis.	`policy=policy:write-attribute(name=no-dictionary,value=true)`
		
		`/profile=default/subsystem=remoting/connector=remoting-connector/security=sasl/sasl-policy=policy:write-attribute(name=no-plain-text,value=false)`
		
		`/profile=default/subsystem=remoting/connector=remoting-connector/security=sasl/sasl-policy=policy:write-attribute(name=pass-credentials,value=true)`

propriétés	Contient un ou plusieurs éléments **`<property>`**, contenant chacun un attribut **`name`** et un attribut **`value`** optionnel.	`/profile=default/subsystem=remoting/connector=remoting-connector/security=sasl/property=myprop:add(value=1)`
		
		`/profile=default/subsystem=remoting/connector=remoting-connector/security=sasl/property=myprop2:add(value=2)`

Exemple 16.13. Exemples de configurations

Cet exemple montre le sous-système à distance par défaut fourni dans JBoss EAP 6.

```
<subsystem xmlns="urn:jboss:domain:remoting:1.1">
    <connector name="remoting-connector" socket-binding="remoting"
security-realm="ApplicationRealm"/>
</subsystem>
```

Cet exemple contient de nombreuses valeurs hypothétiques, et est présenté de façon à ce que l'on puisse mettre les éléments et les attributs dont on a discutés plus haut en contexte.

```
<subsystem xmlns="urn:jboss:domain:remoting:1.1">
    <worker-thread-pool read-threads="1" task-keepalive="60" task-max-
threads="16" task-core-thread="4" task-limit="16384" write-threads="1"
/>
    <connector name="remoting-connector" socket-binding="remoting"
security-realm="ApplicationRealm">
        <sasl>
            <include-mechanisms value="GSSAPI PLAIN DIGEST-MD5" />
            <qop value="auth" />
            <strength value="medium" />
            <reuse-session value="false" />
            <server-auth value="false" />
            <policy>
                <forward-secrecy value="true" />
                <no-active value="false" />
                <no-anonymous value="false" />
                <no-dictionary value="true" />
                <no-plain-text value="false" />
                <pass-credentials value="true" />
            </policy>
            <properties>
                <property name="myprop1" value="1" />
                <property name="myprop2" value="2" />
            </properties>
        </sasl>
        <authentication-provider name="myprovider" />
        <properties>
            <property name="myprop3" value="propValue" />
        </properties>
    </connector>
    <outbound-connections>
        <outbound-connection name="my-outbound-connection"
uri="http://myhost:7777/"/>
        <remote-outbound-connection name="my-remote-connection"
outbound-socket-binding-ref="my-remote-socket" username="myUser"
security-realm="ApplicationRealm"/>
        <local-outbound-connection name="myLocalConnection" outbound-
socket-binding-ref="my-outbound-socket"/>
    </outbound-connections>
</subsystem>
```

Aspects de la configuration non encore documentés

- JNDI et Détection automatique multi diffusion

Rapporter un bogue

16.4.4.5. Utilisation des domaines de sécurité avec les clients EJB distants

Une façon d'ajouter la sécurité aux clients qui font appel aux EJB à distance consiste à utiliser des

domaines de sécurité. Un domaine de sécurité est une simple base de données de paires nom d'utilisateur/mot de passe et nom d'utilisateur/rôle. La terminologie est également utilisée dans le cadre de conteneurs web, avec un sens légèrement différent.

Pour authentifier un EJB par une paire nom d'utilisateur et mot de passe particulière existant déjà dans un domaine de sécurité, suivre les étapes suivantes :

- Ajoutez un nouveau domaine de sécurité au contrôleur de domaine ou au serveur autonome.
- Ajoutez les paramètres suivants au fichier **`jboss-ejb-client.properties`**, qui est dans le chemin de classes de l'application. Cet exemple suppose que la connexion est appelée **`par défaut`** par les autres paramètres qui se trouvent dans le fichier.

```
remote.connection.default.username=appuser
remote.connection.default.password=apppassword
```

- Créez un connecteur Remoting personnalisé sur le domaine ou sur le serveur autonome qui utilise le nouveau domaine de sécurité.
- Déployez votre EJB dans le groupe de serveur configuré pour utiliser le profil avec le connecteur Remoting personnalisé, ou dans le serveur autonome si vous n'utilisez pas de domaine géré.

Rapporter un bogue

16.4.4.6. Ajout d'un domaine de sécurité

1. **Exécuter l'interface CLI**
   Démarrez par la commande **`jboss-cli.sh`** ou **`jboss-cli.bat`** et connectez-vous au serveur.

2. **Créer le nouveau domaine de sécurité lui-même.**
   Exécutez la commande suivante pour créer un nouveau domaine de sécurité nommé **`MyDomainRealm`** sur un contrôleur de domaine ou sur un serveur autonome.

   Pour une instance de domaine, utiliser la commande suivante :

```
/host=master/core-service=management/security-
realm=MyDomainRealm:add()
```

   Pour une instance autonome, utiliser la commande suivante :

```
/core-service=management/security-realm=MyDomainRealm:add()
```

3. **Créer les références du fichier de propriétés qui stocke les informations sur le nouveau rôle.**
   Exécutez la commande suivante pour créer un pointeur au fichier nommé **`myfile.properties`**, qui contiendra les propriétés attachées au nouveau rôle.

NOTE

Le fichier de propriétés nouvellement créées n'est pas géré par les scripts **`add-user.sh`** et **`add-user.bat`** inclus. Il devra être administré en externe.

Pour une instance de domaine, utiliser la commande suivante :

```
/host=master/core-service=management/security-
realm=MyDomainRealm/authentication=properties:add(path=myfile.proper
ties)
```

Pour une instance autonome, utiliser la commande suivante :

```
/core-service=management/security-
realm=MyDomainRealm/authentication=properties:add(path=myfile.proper
ties)
```

Résultat

Votre nouveau domaine de sécurité sera créé. Lorsque vous ajoutez des utilisateurs et des rôles à ce nouveau domaine, l'information va être stockée dans un fichier séparé des domaines de sécurité par défaut. Vous pouvez gérer ce nouveau fichier à l'aide de vos propres applications ou procédures.

Rapporter un bogue

16.4.4.7. Ajout d'un utilisateur à un domaine de sécurité

1. **Éxécuter la commande `add-user.sh` ou `add-user.bat`.**
   Ouvrez un terminal (shell) et changez de répertoire ***`EAP_HOME`*`/bin/`**. Si vous exécutez Red Hat Enterprise Linux ou un autre système d'exploitation style UNIX, exécutez **`add-user.sh`**. Si vous exécutez sur un serveur Microsoft Windows, exécutez **`add-user.bat`**.

2. **Choisir d'ajouter un utilisateur de gestion ou un utilisateur d'application.**
   Pour cette procédure, saisir **b** pour ajouter un utilisateur d'application.

3. **Choisir le domaine dans lequel l'utilisateur sera ajouté.**
   Par défaut, le seul domaine disponible est **`ApplicationRealm`**. Si vous avez ajouté un domaine personnalisé, vous pouvez saisir son nom à la place.

4. **Saisir le nom d'utilisateur, le mot de passe et les rôles lorsque vous y serez invité.**
   Saisir le nom d'utilisateur, le mot de passe et les rôles lorsque vous y serez invité. Vérifiez votre choix en tapant **`yes`**, ou **no** pour annuler les changements. Les changements sont inscrits dans les fichiers de propriétés du domaine de sécurité.

Rapporter un bogue

16.4.4.8. Accès EJB à distance utilisant le cryptage SSL

Par défaut, le trafic réseau pour les RMI (Remote Method Invocation) des Beans EJB2 et EJB3 n'est pas crypté. Dans le cas où le cryptage est requis, SSL (Secure Sockets Layer) peut être utilisé afin que la connexion entre le client et le serveur soit cryptée. L'utilisation SSL a l'avantage supplémentaire de permettre au trafic réseau de traverser les pare-feu qui bloquent le port RMI.

Rapporter un bogue

16.5. SÉCURITÉ DES APPLICATIONS JAX-RS

16.5.1. Activez la sécurité basée-rôle pour RESTEasy JAX-RS Web Service

Résumé

RESTEasy supporte les annotations @RolesAllowed, @PermitAll, et @DenyAll sur les méthodes JAX-RS. Cependant, il ne reconnaît pas ces annotations par défaut. Suivre les étapes suivantes pour configurer le fichier **web.xml** et pour activer la sécurité basée-rôle.

AVERTISSEMENT

Ne pas activer la sécurité basée-rôle si l'application utilise les EJB. Le conteneur EJB procurera la fonctionnalité à la place de RESTEasy.

Procédure 16.3. Activez la sécurité basée-rôle pour RESTEasy JAX-RS Web Service

1. Ouvrir le fichier **web.xml** de l'application dans un éditeur de texte.
2. Ajoutez le <context-param> suivant au fichier, dans les balises **web-app** :

```
<context-param>
    <param-name>resteasy.role.based.security</param-name>
    <param-value>true</param-value>
</context-param>
```

3. Déclarez tous les rôles utilisés dans le fichier RESTEasy JAX-RS WAR file, en utilisant les balises de <security-role> :

```
<security-role>
    <role-name>ROLE_NAME</role-name>
</security-role>
<security-role>
    <role-name>ROLE_NAME</role-name>
</security-role>
```

4. Autorisez l'accès à tous les URL gérés par le runtime JAX-RS pour tous les rôles :

```
<security-constraint>
    <web-resource-collection>
 <web-resource-name>Resteasy</web-resource-name>
 <url-pattern>/PATH</url-pattern>
    </web-resource-collection>
    <auth-constraint>
 <role-name>ROLE_NAME</role-name>
 <role-name>ROLE_NAME</role-name>
    </auth-constraint>
</security-constraint>
```

Résultat

La sécurité basée rôle à été activée dans l'application, avec un certain nombre de rôles définis.

Exemple 16.14. Exemple de configuration de sécurité basée rôles

```
<web-app>

   <context-param>
<param-name>resteasy.role.based.security</param-name>
<param-value>true</param-value>
   </context-param>

   <servlet-mapping>
<servlet-name>Resteasy</servlet-name>
<url-pattern>/*</url-pattern>
   </servlet-mapping>

   <security-constraint>
<web-resource-collection>
    <web-resource-name>Resteasy</web-resource-name>
    <url-pattern>/security</url-pattern>
</web-resource-collection>
<auth-constraint>
    <role-name>admin</role-name>
    <role-name>user</role-name>
</auth-constraint>
   </security-constraint>

   <security-role>
<role-name>admin</role-name>
   </security-role>
   <security-role>
<role-name>user</role-name>
   </security-role>

</web-app>
```

Rapporter un bogue

16.5.2. Sécuriser un service JAX-RS Web par des annotations

Résumé

Cette rubrique couvre les étapes à parcourir pour sécuriser un service JAX-RS Web par les annotations de sécurité supportées.

Procédure 16.4. Sécuriser un service JAX-RS Web par des annotations de sécurité supportées.

1. Activez la sécurité basée-rôle. Pour plus d'informations, voir Section 16.5.1, « Activez la sécurité basée-rôle pour RESTEasy JAX-RS Web Service ».

2. Ajoutez des annotations de sécurité au service JAX-RS Web. RESTEasy supporte les annotations suivantes :

   @RolesAllowed

   Définit les rôles qui peuvent accéder à la méthode. Tous les rôles doivent être définis dans le fichier **`web.xml`**.

   @PermitAll

Autorise tous les rôles définis dans le fichier **`web.xml`** à accéder à la méthode.

@DenyAll

Refuse tout accès à la méthode.

Rapporter un bogue

16.6. PROTOCOLE MOTS DE PASSES DISTANTS SÉCURISÉS

16.6.1. Protocole pour mots de passes distants sécurisés (SRP pour Secure Remote Password)

Le protocole SRP est l'implémentation d'une poignée de main d'échange de clés publiques décrite dans Internet Standards Working Group Request For Comments 2945 (RFC2945). L'extrait de RFC2945 énonce ce qui suit :

> Ce document décrit un mécanisme d'authentification de réseau à cryptage puissant connu comme le nom de protocole SRP (Secure Remote Password). Ce mécanisme est approprié pour négocier des connexions sécurisées avec un mot de passe fourni par l'utilisateur, tout en éliminant les problèmes de sécurité traditionnellement associés à des mots de passe réutilisables. Ce système effectue également un échange de clés sécurisée dans le processus d'authentification, ce qui permet à ce que des couches de sécurité (protection de la vie privée et/ou de l'intégrité) soient activées lors de la session. Il n'est nul besoin de serveurs de clés de confiance ou d'infrastructures de certificats, et les clients ne sont pas tenus de stocker ou de gérer des clés à long terme. SRP offre des avantages tant au niveau de la sécurité qu'au niveau déploiement sur les techniques existantes de stimulation / réponse, ce qui en fait un remplacement idéal pour les besoins d'authentification de mots de passe sécurisés.

On trouvera la description complète de la spécification RFC2945 à http://www.rfc-editor.org/rfc.html. Vous trouverez plus d'informations sur l'algorithme SRP et son historique à http://srp.stanford.edu/.

Les algorithmes Diffie-Hellman et RSA sont connus comme algorithmes d'échange de clés publiques. Le concept d'algorithmes à clé publique est d'avoir deux clés, une publique accessible à tous et une privée et uniquement connue par vous-même. Quand des personnes veulent vous envoyer des informations cryptées, ils cryptent les informations à l'aide de votre clé publique. Vous seul êtes capable de déchiffrer les informations à l'aide de votre clé privée. Cela contraste avec les systèmes de déchiffrage de mots de passe partagés plus traditionnels qui exigent que l'émetteur et le récepteur partagent le mot de passe. Les algorithmes à clé publiques éliminent le besoin de partager des mots de passe.

Rapporter un bogue

16.6.2. Configuration du protocole SRP (Secure Remote Password)

Afin d'utiliser le protocole SRP (Secure Remote Password) dans votre application, vous devrez tout d'abord créer un MBean qui implémente l'interface **`SRPVerifierStore`**. Vous trouverez des informations sur l'implémentation à Implémentation SRPVerifierStore.

Procédure 16.5. Intégrer le store de mots de passe existant

1. **Créer un store d'informations de mots de passe hachés.**
   Si vos mots de passe sont déjà stockés sous une forme hachée irréversible, vous aurez besoin de faire cela sur la base utilisateur.

Vous pouvez implémenter un **`setUserVerifier(String, VerifierInfo)`** en tant que méthode noOp, ou d'une méthode qui envoie une exception signifiant que le store est en lecture-seule.

2. **Créer l'interface SRPVerifierStore.**
   Créer une implémentation d'interface **`SRPVerifierStore`** personnalisée qui puisse obtenir **`VerifierInfo`** du store que vous avez créé.

   Le **`verifyUserChallenge(String, Object)`** peut servir à intégrer des schémas basés token de matériel existant comme SafeWord ou Radius dans l'algorithme SRP. Cette méthode d'interface est appelée uniquement lorsque la configuration SRPLoginModule du client spécifie l'option hasAuxChallenge.

3. **Créer le MBean JNDI.**
   Créer un MBean qui expose l'interface **`SRPVerifierStore`** disponible à JNDI, et expose tous les paramètres configurables requis.

   Le service **`org.jboss.security.srp.SRPVerifierStoreService`** par défaut vous permet d'implémenter ceci. Vous pouvez également implémenter le MBean par une implémentation de fichier de propriétés Java du **`SRPVerifierStore`**.

Implémentation SRPVerifierStore

L'implémentation par défaut de l'interface **`SRPVerifierStore`** n'est pas recommandée pour les systèmes de production, car elle exige que toutes les informations de hachage de mot de passe soient disponibles sous forme de fichier d'objets sérialisés.

L'implémentation du **`SRPVerifierStore`** fournit un accès à l'objet **`SRPVerifierStore.VerifierInfo`** pour un nom d'utilisateur donné. La méthode **`getUserVerifier(String)`** est appelée par le SRPService au départ d'une session SRP utilisateur pour obtenir les paramètres requis par l'algorithme SRP.

Éléments d'un Objet VerifierInfo

nom d'utilisateur

Le nom d'utilisateur ou l'ID utilisateur pour s'authentifier

verifier

Un hachage unidirectionnel du mot de passe que l'utilisateur saisit comme preuve d'identité. La classe **`org.jboss.security.Util`** inclut une méthode **`calculateVerifier`** qui exécute l'algorithme de hachage de mot de passe. Le mot de passe de sortie prend la forme **`H(salt | H(username | password))`**, où **`H`** est la fonction de hachage sécurisée SHA, telle que définie par RFC2945. Le nom d'utilisateur est converti d'une chaîne en un byte [] en utilisant le codage UTF-8.

salt

Un nombre aléatoire utilisé pour augmenter la difficulté d'une attaque de force brute de dictionnaire sur la base de données de mot de passe de vérification au cas où la base de données soit compromise. La valeur doit être générée à partir d'un algorithme de cryptage fort composé de nombres aléatoires lorsque le mot de passe en texte clair existant de l'utilisateur est haché.

g

Le générateur primitif d'algorithme SRP. Cela peut être un paramètre fixe bien connu plutôt qu'un paramètre par utilisateur. La classe d'utilitaire de **`org.jboss.security.srp.SRPConf`** fournit

plusieurs paramètres pour **g**, avec une valeur par défaut appropriée obtenue par l'intermédiaire de **`SRPConf.getDefaultParams().g()`**.

N

SRP algorithm safe-prime modulus. Cela peut être un paramètre fixe bien connu, plutôt qu'un paramètre par utilisateur. La classe d'utilitaire de **`org.jboss.security.srp.SRPConf`** fournit plusieurs paramètres pour «N», y compris un bon nombre de valeurs par défaut obtenues via **`SRPConf.getDefaultParams().N()`**.

Exemple 16.15. L'interface SRPVerifierStore

```
package org.jboss.security.srp;

import java.io.IOException;
import java.io.Serializable;
import java.security.KeyException;

public interface SRPVerifierStore
{
    public static class VerifierInfo implements Serializable
    {

        public String username;


        public byte[] salt;
        public byte[] g;
        public byte[] N;
    }


    public VerifierInfo getUserVerifier(String username)
        throws KeyException, IOException;

    public void setUserVerifier(String username, VerifierInfo info)
        throws IOException;


     public void verifyUserChallenge(String username, Object
auxChallenge)
         throws SecurityException;
}
```

Rapporter un bogue

16.7. L'ARCHIVAGE SÉCURISÉ DES MOTS DE PASSE POUR LES STRINGS DE NATURE CONFIDENTIELLE

16.7.1. Sécurisation des chaînes confidentielles de fichiers en texte clair

Les applications web et autres déploiements incluent souvent des fichiers en texte clair, comme des

descripteurs de déploiement XML, qui comprennent des informations sensibles telles que les mots de passe et autres strings sensibles. JBoss EAP 6 inclut un mécanisme d'archivage sécurisé de mots de passe qui vous permet de crypter les strings sensibles et de les stocker dans un keystore chiffré. Le mécanisme d'archivage sécurisé parvient à décrypter les strings à utiliser dans les domaines de sécurité, ou autres systèmes de vérification. Ceci fournit une couche supplémentaire de sécurité. Le mécanisme s'appuie sur les outils qui sont inclus dans toutes les implémentations de Java Development Kit (JDK) prises en charge.

AVERTISSEMENT

Des problèmes ont été décelés lors de l'utilisation de la fonctionnalité de sécurité d'archivage sécurisé avec JBoss EAP 6. Il a été constaté que le **`vault.keystore`** généré par le **`keytool`** n'est pas un fichier de clés valide lorsqu'il est utilisé avec un JDK IBM. Cela est dû au fait que les implémentations de keystore JCEKS diffèrent selon les vendeurs de Java.

Le problème se présente lorsqu'un fichier de clés généré par Oracle Java est utilisé dans une instance de JBoss EAP sur une installation Java d'IBM. Dans ces cas, le serveur ne démarre pas et lève l'exception suivante :

```
java.io.IOException:
com.sun.crypto.provider.SealedObjectForKeyProtector
```

À l'heure actuelle, la seule solution qui existe consiste à éviter toute tentative d'utilisation d'un fichier de clés généré avec un keytool Oracle dans un environnement utilisant une implémentation de Java d'IBM.

Rapporter un bogue

16.7.2. Créer un keystore Java pour stocker des strings sensibles

Conditions préalables

- La commande **`keytool`** doit être disponible. Elle est fournie par le Java Runtime Environment (JRE). Chercher le chemin du fichier. Se trouve à l'emplacement suivant **`/usr/bin/keytool`** dans Red Hat Enterprise Linux.

Procédure 16.6. Installation du Java Keystore

1. **Créer un répertoire pour stocker votre keystore et d'autres informations cryptées.**
   Créez un répertoire qui contiendra votre keystore et d'autres informations pertinentes. Le reste de cette procédure assume que le répertoire est ***`EAP_HOME`*`/vault/`**. Comme le répertoire devra contenir des informations sensibles, il devra être accessible à un nombre restraint d'utilisateurs. Le compte d'utilisateur sous lequel JBoss EAP exécute requiert au minimum un accès en lecture-écriture.

2. **Déterminer les paramètres à utiliser avec `keytool`.**
   Déterminer les paramètres suivants :

alias

L'alias est un identificateur unique pour l'archivage sécurisé ou autres données stockées dans le keystore. L'alias dans l'exemple de commande à la fin de cette procédure est **`vault`** (archivage sécurisé). Les alias sont insensibles à la casse.

keyalg

L'algorithme à utiliser pour le cryptage. Dans cette procédure, l'exemple utilise **RSA**. Consultez la documentation de votre JRE et de votre système d'exploitation pour étudier vos possibilités.

keysize

La taille d'une clé de cryptage impacte sur la difficulté de décrypter au seul moyen de la force brutale. Dans cette procédure, l'exemple utilise **`1024`**. Pour plus d'informations sur les valeurs appropriées, voir la documentation distribuée avec **`keytool`**.

keystore

Le keystore est une base de données qui contient des informations chiffrées et des informations sur la façon de déchiffrer. Si vous ne spécifiez pas de keystore, le keystore par défaut à utiliser est un fichier appelé **`.keystore`** dans votre répertoire personnel. Quand vous ajoutez des données dans un keystore pour la première fois, il sera créé. L'exemple de cette procédure utilise le keystore **`vault.keystore`**.

La commande du **`keytool`** a plusieurs options. Consultez la documentation de votre JRE ou de votre système d'exploitation pour obtenir plus d'informations.

3. **Déterminer les réponses aux questions que la commande `keystore` vous demandera.**
Le **`keystore`** a besoin des informations suivantes pour remplir l'entrée du keytore :

Mot de passe du keystore

Lorsque vous créez un keystore, vous devez définir un mot de passe. Pour pouvoir travailler dans keystore dans le futur, vous devez fournir le mot de passe. Créer un mot de passe dont vous vous souviendrez. Le keystore est sécurisé par son mot de passe et par la sécurité du système d'exploitation et du système de fichiers où il se trouve.

Mot de passe clé (en option)

En plus du mot de passe du keystore, vous pouvez indiquer un mot de passe pour chaque clé contenue. Pour utiliser une clé, le mot de passe doit être donné à chaque utilisation. Normalement, cette fonction n'est pas utilisée.

Prénom et nom de famille

Cela, ainsi que le reste de l'information dans la liste, aide à identifier la clé de façon unique et à la placer dans une hiérarchie par rapport aux autres clés. Il ne doit pas nécessairement correspondre à un nom, mais doit être composé de deux mots et doit être unique à une clé. L'exemple dans cette procédure utilise **`Admninistrateur Comptabilité`**. En terme de répertoires cela devient le *nom commun* du certificat.

Unité organisationnelle

Il s'agit d'un mot unique d'identification qui utilise le certificat. Il se peut que ce soit l'application ou l'unité commerciale. L'exemple de cette procédure utilise **`AccountingServices`**. Normalement, tous les keystores utilisés par un groupe ou une

application utilisent la même unité organisationnelle.

Organisation

Il s'agit normalement d'une représentation de votre nom d'organisation en un seul mot. Demeure constant à travers tous les certificats qui sont utilisés par une organisation. Cet exemple utilise **`MyOrganization`**.

Ville ou municipalité

Votre ville.

État ou province

Votre état ou province, ou l'équivalent pour votre localité.

Pays

Le code pays en deux lettres.

Ces informations vont créer ensemble une hiérarchie de vos keystores et certificats, qui garantira qu'ils utilisent une structure de nommage consistante, et unique.

4. **Exécuter la commande `keytool`, en fournissant les informations que vous avez collectées.**

Exemple 16.16. Exemple d'entrée et de sortie de la commande `keystore`

```
$ keytool -genseckey -alias vault -storetype jceks -keyalg AES -
keysize 128 -storepass vault22 -keypass vault22 -validity 730 -
keystore EAP_HOME/vault/vault.keystore
Enter keystore password: vault22
Re-enter new password:vault22
What is your first and last name?
  [Unknown]:  Accounting Administrator
What is the name of your organizational unit?
  [Unknown]:  AccountingServices
What is the name of your organization?
  [Unknown]:  MyOrganization
What is the name of your City or Locality?
  [Unknown]:  Raleigh
What is the name of your State or Province?
  [Unknown]:  NC
What is the two-letter country code for this unit?
  [Unknown]:  US
Is CN=Accounting Administrator, OU=AccountingServices,
O=MyOrganization, L=Raleigh, ST=NC, C=US correct?
  [no]:  yes

Enter key password for <vault>
        (RETURN if same as keystore password):
```

Résultat

Un fichier nommé **vault.keystore** est créé dans le répertoire ***EAP_HOME*/vault/**. Il stocke une clé simple, nommée **vault**, qui sera utilisée pour stocker des strings cryptés, comme des mots de passe, pour la plateforme JBoss EAP 6.

Rapporter un bogue

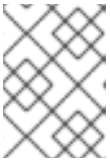
16.7.3. Masquer le mot de passe du keystore et initialiser le mot de passe de l'archivage de sécurité

Conditions préalables

- Section 16.7.2, « Créer un keystore Java pour stocker des strings sensibles »

1. **Exécuter la commande `vault.sh`.**
   Exécuter ***EAP_HOME*/bin/vault.sh**. Démarrer une nouvelle session interactive en tapant **0**.

2. **Saisir le nom du répertoire où les fichiers cryptés seront stockés.**
   Ce répertoire doit être accessible à des utilisateurs limités uniquement. Au minimum, le compte d'utilisateur sous lequel JBoss EAP exécute requiert un accès lecture-écriture. Si vous avez suivi Section 16.7.2, « Créer un keystore Java pour stocker des strings sensibles », votre keystore se trouvera dans le répertoire nommé ***EAP_HOME*/vault/**.

   NOTE

   N'oubliez pas d'inclure la barre oblique finale dans le nom du répertoire. Soit `/` ou `\`, selon votre système d'exploitation.

3. **Saisir le nom de votre keystore.**
   Saisir le nom complet vers le fichier de keystore. Cet exemple utilise ***EAP_HOME*/vault/vault.keystore**.

4. **Crypter le mot de passe du keystore.**
   Les étapes suivantes vous servent à crypter le mot de passe du keystore, afin que vous puissiez l'utiliser dans les applications et les fichiers de configuration en toute sécurité.

   a. **Saisir le mot de passe du keystore.**
      Quand vous y serez invité, saisir le mot de passe du keystore.

   b. **Saisir une valeur salt.**
      Entrez une valeur salt de 8 caractères. La valeur salt, ainsi que le nombre d'itérations (ci-dessous), sont utilisés pour créer la valeur de hachage

   c. **Saisir le nombre d'itérations.**
      Saisir un nombre pour le nombre d'itérations.

   d. **Notez les informations de mot de passe masqué.**
      Le mot de passe masqué, salt, et le nombre d'itérations sont imprimés en sortie standard. Prenez-en note dans un endroit sûr. Un attaquant pourrait les utiliser pour déchiffrer le mot de passe.

   e. **Saisir un alias pour l'archivage de sécurité.**
      Quand on vous y invite, saisir un alias pour l'archivage de sécurité. Si vous suivez Section 16.7.2, « Créer un keystore Java pour stocker des strings sensibles » pour créer votre archivage de sécurité, l'alias sera **vault**.

5. **Sortir de la console interactive.**
   Saisir **2** pour sortir de la console interactive.

Résultat

Votre mot de passe de keystore est masqué afin de pouvoir être utilisé dans les fichiers de configuration et de déploiement. De plus, votre archivage de sécurité est complètement configuré et prêt à l'utilisation.

Rapporter un bogue

16.7.4. Configurer JBoss EAP pour qu'il utilise l'archivage sécurisé des mots de passe

Aperçu

Avant de masquer les mots de passe et d'autres attributs sensibles dans les fichiers de configuration, vous devez sensibiliser JBoss EAP 6 à l'archivage sécurisé des mots de passe qui les stocke et les déchiffre. Suivez cette procédure pour activer cette fonctionnalité.

Conditions préalables

- Section 16.7.2, « Créer un keystore Java pour stocker des strings sensibles »
- Section 16.7.3, « Masquer le mot de passe du keystore et initialiser le mot de passe de l'archivage de sécurité »

Procédure 16.7. Assigner un mot de passe d'archivage sécurisé.

1. **Déterminer les valeurs qui conviennent pour la commande.**
   Déterminer les valeurs pour les paramètres suivants, qui sont déterminés par les commandes utilisées pour créer le keystore lui-même. Pour obtenir des informations sur la façon de créer un keystore, voir les sujets suivants : Section 16.7.2, « Créer un keystore Java pour stocker des strings sensibles » et Section 16.7.3, « Masquer le mot de passe du keystore et initialiser le mot de passe de l'archivage de sécurité ».

Paramètre	Description
KEYSTORE_URL	Le chemin d'accès ou URI du fichier keystore, qui s'appelle normalement **`vault.keystore`**
KEYSTORE_PASSWORD	Le mot de passe utilisé pour accéder au keystore. Cette valeur devrait être masquée.
KEYSTORE_ALIAS	Le nom du keystore.
SALT	Le salt utilisé pour crypter et décrypter les valeurs de keystore.
ITERATION_COUNT	Le nombre de fois que l'algorithme de chiffrement est exécuté.

Paramètre	Description
ENC_FILE_DIR	Le chemin d'accès au répertoire à partir duquel les commandes de keystore sont exécutées. Normalement, le répertoire contient les mots de passe sécurisés.
hôte (domaine géré uniquement)	Le nom de l'hôte que vous configurez

2. **Utilisez l'interface CLI pour activer les mots de passe sécurisés.**
Exécutez une des commandes suivantes, selon que vous utilisez un domaine géré ou une configuration de serveur autonome. Substituez les valeurs de la commande par celles de la première étape de cette procédure.

NOTE

Si vous utilisez Microsoft Windows Server, remplacez chaque caractère `\` du chemin d'accès du répertoire par un caractère `\` supplémentaire dans la commande CLI. Par exemple, **`C:\\data\\vault\\vault.keystore`**. C'est parce qu'un simple caractère `\` est utilisé comme caractère d'échappement.

- **Domaine géré**

```
/host=YOUR_HOST/core-service=vault:add(vault-options=
[("KEYSTORE_URL" => "PATH_TO_KEYSTORE"), ("KEYSTORE_PASSWORD" =>
"MASKED_PASSWORD"), ("KEYSTORE_ALIAS" => "ALIAS"), ("SALT" =>
"SALT"),("ITERATION_COUNT" => "ITERATION_COUNT"), ("ENC_FILE_DIR"
=> "ENC_FILE_DIR")])
```

- **Serveur autonome**

```
/core-service=vault:add(vault-options=[("KEYSTORE_URL" =>
"PATH_TO_KEYSTORE"), ("KEYSTORE_PASSWORD" => "MASKED_PASSWORD"),
("KEYSTORE_ALIAS" => "ALIAS"), ("SALT" => "SALT"),
("ITERATION_COUNT" => "ITERATION_COUNT"), ("ENC_FILE_DIR" =>
"ENC_FILE_DIR")])
```

Ce qui suit est un exemple de la commande avec des valeurs hypothétiques :

```
/core-service=vault:add(vault-options=[("KEYSTORE_URL" =>
"/home/user/vault/vault.keystore"), ("KEYSTORE_PASSWORD" => "MASK-
3y28rCZlcKR"), ("KEYSTORE_ALIAS" => "vault"), ("SALT" =>
"12438567"),("ITERATION_COUNT" => "50"), ("ENC_FILE_DIR" =>
"/home/user/vault/")])
```

Résultat

JBoss EAP 6 est configuré pour décrypter les strings masqués par l'intermédiaire de l'archivage sécurisé de mots de passe. Pour ajouter des strings à l'archivage sécurisé et les utiliser dans votre configuration, voir la section suivante : Section 16.7.6, « Stocker et résoudre les strings sensibles cryptés du keystore Java. ».

Rapporter un bogue

16.7.5. Configurer JBoss EAP pour qu'il utilise une implémentation d'archivage sécurisé personnalisée

Résumé

Vous pouvez utiliser votre propre implémentation de **`SecurityVault`** pour masquer les mots de passe, et les autres attributs sensibles dans les fichiers de configuration.

Procédure 16.8. Utilisez une implémentation sécurisée de l'archivage des mots de passe

1. Créez une classe qui implémente l'interface **`SecurityVault`**.
2. Créez un module contenant la classe de l'étape précédente, et spécifiez une dépendance sur **`org.picketbox`** où l'interface est **`SecurityVault`**.
3. Activez l'archivage de mots de passe sécurisée dans la configuration du serveur JBoss EAP en ajoutant l'élément d'archivage à l'aide des attributs suivants :

   code

   La nom complet de la classe qui implémente **`SecurityVault`**.

   module

   Le nom du module qui contient la classe personnalisée.

   Optionnellement, vous pouvez utiliser les paramètres **`vault-options`** pour initialiser la classe personnalisée d'un archivage de mots de passe. Par exemple :

```
/core-
service=vault:add(code="custom.vault.implementation.CustomSecurityVa
ult", module="custom.vault.module", vault-options=[("KEYSTORE_URL"
=> "PATH_TO_KEYSTORE"), ("KEYSTORE_PASSWORD" => "MASKED_PASSWORD"),
("KEYSTORE_ALIAS" => "ALIAS"), ("SALT" => "SALT"),("ITERATION_COUNT"
=> "ITERATION_COUNT"), ("ENC_FILE_DIR" => "ENC_FILE_DIR")])
```

Résultat

JBoss EAP 6 est configuré pour décrypter les chaînes masquées par l'intermédiaire d'une implémentation personnalisée de l'archivage des mots de passe.

Rapporter un bogue

16.7.6. Stocker et résoudre les strings sensibles cryptés du keystore Java.

Résumé

En comptant les mots de passe et les autres strings sensibles, les fichiers de configuration en texte brut ne sont pas sécurisés. JBoss EAP 6 inclut la capacité à stocker et à utiliser les valeurs masquées dans les fichiers de configuration, et à utiliser ces valeurs masquées dans les fichiers de configuration.

Conditions préalables

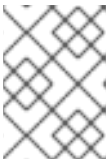
- Section 16.7.2, « Créer un keystore Java pour stocker des strings sensibles »

- Section 16.7.3, « Masquer le mot de passe du keystore et initialiser le mot de passe de l'archivage de sécurité »
- Section 16.7.4, « Configurer JBoss EAP pour qu'il utilise l'archivage sécurisé des mots de passe »
- L'application ***EAP_HOME*/bin/vault.sh** doit pouvoir être accessible via l'interface de ligne de commande.

Procédure 16.9. Installation du Java keystore

1. **Exécutez la commande `vault.sh`.**
   Exécutez ***EAP_HOME*/bin/vault.sh**. Démarrez une nouvelle session interactive en tapant **0**.

2. **Saisir le nom du répertoire où les fichiers cryptés seront stockés.**
   Si vous suivez Section 16.7.2, « Créer un keystore Java pour stocker des strings sensibles », votre keystore sera dans un répertoire nommé **vault/** de votre répertoire de base. Dans la plupart des cas, il est logique de stocker toutes vos informations cryptées au même endroit dans le keystore. Cet exemple utilise le répertoire **/home/*USER*/vault/**.

   NOTE

   N'oubliez pas d'inclure la barre oblique finale dans le nom du répertoire. Soit `/` ou `\` selon votre système d'exploitation.

3. **Saisissez le nom de votre keystore.**
   Saisissez le nom complet vers le fichier de keystore. Cet exemple utilise **/home/*USER*/vault/vault.keystore**.

4. **Saisissez le mot de passe du keystore, le nom de l'archivage sécurisé, salt, et le nombre d'itérations.**
   Quand vous y êtes invité, saisissez le mot de passe du keystore, le nom de l'archivage sécurisé, salt, et le nombre d'itérations.

5. **Sélectionnez l'option de stockage d'un mot de passe.**
   Sélectionnez l'option **0** de stockage d'un mot de passe ou autre string sensible.

6. **Saisissez la valeur.**
   Une fois que vous y êtes invité, saisissez la valeur à deux reprises. Si les valeurs ne correspondent pas, vous serez invité à essayer à nouveau.

7. **Saisissez le bloc d'archivage sécurisé.**
   Saisissez le bloc d'archivage sécurisé, qui correspond à un conteneur pour les attributs ayant trait à la même ressource. Un exemple de nom d'attribut pourrait être **ds_ExampleDS**. Cela fera partie de la référence à la chaîne cryptée, dans votre source de données ou autre définition de service.

8. **Saissez le nom de l'attribut.**
   Saisissez le nom de l'attribut que vous stockez. Exemple de nom d'attribut **password**.

   Résultat

   Un message comme celui qui suit montre que l'attribut a été sauvegardé.

   ```
   La valeur d'attribut sécurisé est stockée dans l'archivage sécurisé.
   ```

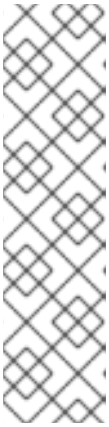
9. **Notez les informations pour ce string crypté.**
   Un message s'affiche sur la sortie standard, montrant le bloc d'archivage sécurisé, le nom de l'attribut, la clé partagée et des conseils sur l'utilisation du string dans votre configuration. Prendre note de ces informations dans un emplacement sécurisé. Voici un exemple de sortie.

```
*********************************************
Vault Block:ds_ExampleDS
Attribute Name:password
Configuration should be done as follows:
VAULT::ds_ExampleDS::password::1
*********************************************
```

10. **Utilisez le string crypté dans votre configuration.**
    Utilisez le string de l'étape de configuration précédente, à la place du string en texte brut. Une source de données utilisant le mot de passe crypté ci-dessus, est indiquée ci-dessous.

```
...
  <subsystem xmlns="urn:jboss:domain:datasources:1.0">
    <datasources>
      <datasource jndi-name="java:jboss/datasources/ExampleDS"
enabled="true" use-java-context="true" pool-name="H2DS">
        <connection-url>jdbc:h2:mem:test;DB_CLOSE_DELAY=-
1</connection-url>
        <driver>h2</driver>
        <pool></pool>
        <security>
          <user-name>sa</user-name>
          <password>${VAULT::ds_ExampleDS::password::1}</password>
        </security>
      </datasource>
      <drivers>
         <driver name="h2" module="com.h2database.h2">
            <xa-datasource-class>org.h2.jdbcx.JdbcDataSource</xa-
datasource-class>
         </driver>
      </drivers>
    </datasources>
  </subsystem>
...
```

    Vous pouvez utiliser un string crypté n'importe où dans votre fichier de configuration autonome ou de domaine pour lequel les expressions sont autorisées.



NOTE

Pour vérifier si les expressions sont utilisées dans un sous-système particulier, exécutez la commande CLI suivante sur ce sous-système :

```
/host=master/core-service=management/security-
realm=TestRealm:read-resource-description(recursive=true)
```

À partir du résultat de cette commande, cherchez la valeur du paramètre **`expressions-allowed`**. Si 'true', vous pourrez utiliser des expressions dans la configuration de ce sous-système particulier.

Une fois que vous aurez mis votre string dans le keystore, utilisez la syntaxe suivante pour remplacer tout string en texte claire par un texte crypté.

```
${VAULT::VAULT_BLOCK::ATTRIBUTE_NAME::ENCRYPTED_VALUE}
```

Voici un exemple de valeur réelle, où le bloc d'archivage sécurisé est **`ds_ExampleDS`** et l'attribut est **`password`**.

```
<password>${VAULT::ds_ExampleDS::password::1}</password>
```

Rapporter un bogue

16.7.7. Stocker et résoudre des strings sensibles de vos applications

Aperçu

Les éléments de configuration de la plate-forme JBoss EAP 6 prennent en charge la capacité à régler les chaînes cryptées en fonction des valeurs stockées dans Java Keystore, via le mécanisme d'archivage sécurisé. Vous pouvez ajouter le support de cette fonctionnalité à vos propres applications.

Tout d'abord, ajoutez le mot de passe dans votre archivage sécurisé. En second lieu, remplacez le mot de passe de texte clair par celui qui est stocké dans l'archivage sécurisé. Vous pouvez utiliser cette méthode pour obscurcir les strings sensibles de votre application.

Pré-requis

Avant d'effectuer cette procédure, assurez-vous que le répertoire utilisé pour stocker vos fichiers dans l'archivage sécurisé existe bien. Qu'importe où vous les placez, tant que l'utilisateur qui exécute JBoss EAP 6 dispose de l'autorisation de lire et écrire des fichiers. Cet exemple situe le répertoire **`vault/`** dans le répertoire **`/home/USER/vault/`**. L'archivage sécurisé lui-même correspond à un fichier nommé **`vault.keystore`** qui se trouve dans le répertoire **`vault/`**.

Exemple 16.17. Ajout de la chaîne de mot de passe dans l'archivage sécurisé

Ajoutez la chaîne de mot de passe dans l'archivage sécurisé par la commande **`EAP_HOME/bin/vault.sh`**. La série de commandes et de réponses est incluse dans la session suivante. Les valeurs saisies par l'utilisateur apparaîtront clairement. Certaines sorties seront supprimées pour le formatage. Dans Microsoft Windows, le nom de la commande est **`vault.bat`**. Notez que dans Microsoft Windows, les chemins d'accès au fichier utilisent le caractère `\` comme séparateur de répertoire, et non pas le caractère `/`.

```
[user@host bin]$ ./vault.sh
```

```
*********************************
****  JBoss Vault ********
*********************************
Please enter a Digit::   0: Start Interactive Session  1: Remove
Interactive Session  2: Exit
0
Starting an interactive session
Enter directory to store encrypted files:/home/user/vault/
Enter Keystore URL:/home/user/vault/vault.keystore
Enter Keystore password: ...
Enter Keystore password again: ...
Values match
Enter 8 character salt:12345678
Enter iteration count as a number (Eg: 44):25

Enter Keystore Alias:vault
Vault is initialized and ready for use
Handshake with Vault complete
Please enter a Digit::   0: Store a password  1: Check whether password
exists  2: Exit
0
Task:  Store a password
Please enter attribute value: sa
Please enter attribute value again: sa
Values match
Enter Vault Block:DS
Enter Attribute Name:thePass
Secured attribute value has been stored in vault.

Please make note of the following:
********************************************
Vault Block:DS
Attribute Name:thePass
Configuration should be done as follows:
VAULT::DS::thePass::1
********************************************

Please enter a Digit::   0: Store a password  1: Check whether password
exists  2: Exit
2
```

La chaîne qui sera ajoutée au code Java est la dernière valeur de sortie, la ligne commençant par **VAULT**.

Le servlet suivant utilise la chaîne voûtée au lieu d'un mot de passe de texte clair. La version en texte clair est commentée afin que vous puissiez voir la différence.

Exemple 16.18. Servlet qui utilise un mot de passe d'archivage sécurisé.

```
package vaulterror.web;

import java.io.IOException;
import java.io.Writer;

import javax.annotation.Resource;
```

```
import javax.annotation.sql.DataSourceDefinition;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.sql.DataSource;


/*@DataSourceDefinition(
        name = "java:jboss/datasources/LoginDS",
        user = "sa",
        password = "sa",
        className = "org.h2.jdbcx.JdbcDataSource",
        url = "jdbc:h2:tcp://localhost/mem:test"
)*/
@DataSourceDefinition(
        name = "java:jboss/datasources/LoginDS",
        user = "sa",
        password = "VAULT::DS::thePass::1",
        className = "org.h2.jdbcx.JdbcDataSource",
        url = "jdbc:h2:tcp://localhost/mem:test"
)
@WebServlet(name = "MyTestServlet", urlPatterns = { "/my/" },
loadOnStartup = 1)
public class MyTestServlet  extends HttpServlet {

    private static final long serialVersionUID = 1L;


    @Resource(lookup = "java:jboss/datasources/LoginDS")
    private DataSource ds;

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse
resp) throws ServletException, IOException {
        Writer writer = resp.getWriter();
        writer.write((ds != null) + "");
    }
}
```

Votre servlet est maintenant capable de résoudre la chaîne d'archivage sécurisé.

Rapporter un bogue

16.8. JAVA AUTHORIZATION CONTRACT FOR CONTAINERS (JACC)

16.8.1. Java Authorization Contract for Containers (JACC)

Java Authorization Contract for Containers (JACC) est une norme qui définit un contrat entre les conteneurs et les fournisseurs de services d'autorisation, et qui se traduit par l'implémentation de fournisseurs à utiliser par des conteneurs. Il a été défini dans JSR-115, qui se trouve sur le site Web de

Java Community Process http://jcp.org/en/jsr/detail?id=115. A fait partie de la spécification Java Enterprise Edition (Java EE) depuis Java EE version 1.3.

JBoss EAP 6 implémente un support pour JACC dans la fonctionnalité de sécurité du sous-système de sécurité.

Rapporter un bogue

16.8.2. Configurer la sécurité JACC (Java Authorization Contract for Containers)

Pour configurer JACC (Java Authorization Contract for Containers), il convient de configurer votre domaine de sécurité avec le module qui convient, puis de modifier votre fichier **`jboss-web.xml`** pour y inclure les paramètres qu'il faut.

Ajouter JACC Support au domaine de sécurité

Pour ajouter JACC au domaine de sécurité, ajouter la police d'autorisation **`JACC`** à la pile d'autorisations du domaine de sécurité, avec l'indicateur **`requis`**. Voici un exemple de domaine de sécurité avec support JACC. Cependant, le domaine de sécurité est configuré dans la console de gestion ou dans l'interface CLI, plutôt que directement dans le code XML.

```
<security-domain name="jacc" cache-type="default">
    <authentication>
        <login-module code="UsersRoles" flag="required">
        </login-module>
    </authentication>
    <authorization>
        <policy-module code="JACC" flag="required"/>
    </authorization>
</security-domain>
```

Configurer une application web qui utilise JACC

Le fichier **`jboss-web.xml`** se trouve dans **`META-INF/`** ou dans le répertoire **`WEB-INF/`** de votre déploiement, et contient des ajouts ou remplacements de configuration spécifique JBoss pour le conteneur web. Pour utiliser votre domaine de sécurité activé-JACC, vous devrez inclure l'élément **`<security-domain>`**, et aussi définir l'élément **`<use-jboss-authorization>`** sur **`true`**. L'application suivante est configurée correctement pour pouvoir utiliser le domaine de sécurité JACC ci-dessus.

```
<jboss-web>
    <security-domain>jacc</security-domain>
    <use-jboss-authorization>true</use-jboss-authorization>
</jboss-web>
```

Configurer une application EJB pour utiliser JACC

La façon de configurer les EJB à utiliser un domaine de sécurité et JACC diffère selon des application Web. Pour un EJB, vous pouvez déclarer des *method permissions* sur une méthode ou sur un groupe de méthodes, dans le descripteur **`ejb-jar.xml`**. Dans l'élément **`<ejb-jar>`**, chaque élément **`<method-permission>`** dépendant contient des informations sur les roles JACC. Voir l'exemple de configuration pour plus d'informations. La classe **`EJBMethodPermission`** fait partie de Java Enterprise Edition 6 API, et est documentée dans http://docs.oracle.com/javaee/6/api/javax/security/jacc/EJBMethodPermission.html.

Exemple 16.19. Exemple de permissions de méthode JACC dans un EJB

```
<ejb-jar>
  <assembly-descriptor>
    <method-permission>
      <description>The employee and temp-employee roles may access any
method of the EmployeeService bean </description>
      <role-name>employee</role-name>
      <role-name>temp-employee</role-name>
      <method>
        <ejb-name>EmployeeService</ejb-name>
        <method-name>*</method-name>
      </method>
    </method-permission>
  </assembly-descriptor>
</ejb-jar>
```

Vous pouvez également contraindre les mécanismes d'authentification et d'autorisation d'un EJB à l'aide d'un domaine de sécurité, comme vous pouvez le faire pour une application web. Les domaines de sécurité sont déclarés dans le descripteur **`jboss-ejb3.xml`** qui se trouve dans l'élément enfant **`<security>`**. En plus du domaine de sécurité, vous pouvez également spécifier le*run-as principal*, qui change le principal que l'EJB exécute.

Exemple 16.20. Exemple de déclaration de domaine de sécurité dans un EJB

```
<security>
  <ejb-name>*</ejb-name>
  <security-domain>myDomain</security-domain>
  <run-as-principal>myPrincipal</run-as-principal>
</security>
```

Rapporter un bogue

16.9. JASPI (JAVA AUTHENTICATION SPI FOR CONTAINERS)

16.9.1. Sécurité Java Authentication SPI pour Conteneurs (JASPI)

Java Application SPI pour Conteneurs (JASPI or JASPIC) est une interface enfichable pour applications JSR-196 du Java Community Process. Consulter http://www.jcp.org/en/jsr/detail?id=196 pour obtenir des informations sur la spécification.

Rapporter un bogue

16.9.2. Configuration de la sécurité Java Authentication SPI pour conteneurs (JASPI)

Pour s'authentifier auprès d'un fournisseur JASPI, ajouter un élément **`<authentication-jaspi>`** à votre domaine de sécurité. La configuration est similaire à celle d'un module d'authentification standard, mais les éléments de module de login sont inclus dans l'élément **`<login-module-stack>`**. La structure de configuration est la suivante :

Exemple 16.21. Structure de l'élément `authentication-jaspi`

```
<authentication-jaspi>
 <login-module-stack name="...">
   <login-module code="..." flag="...">
     <module-option name="..." value="..."/>
   </login-module>
 </login-module-stack>
 <auth-module code="..." login-module-stack-ref="...">
   <module-option name="..." value="..."/>
 </auth-module>
</authentication-jaspi>
```

Le module de connexion est lui-même configuré de la même façon que le module d'authentification standard.

Comme la console de gestion basée web n'expose pas la configuration des modules d'authentification JASPI, vous devez stopper la plateforme JBoss EAP 6 complètement avant d'ajouter la configuration directement dans le fichier ***EAP_HOME*/domain/configuration/domain.xml** ou dans le fichier ***EAP_HOME*/standalone/configuration/standalone.xml**.

Rapporter un bogue

CHAPITRE 17. SSO (SINGLE SIGN ON)

17.1. SSO (SINGLE SIGN ON) POUR LES APPLICATIONS WEB

Aperçu

Single Sign On (SSO) autorise l'authentification à une ressource, en vue d'autoriser implicitement l'accès à d'autres ressources.

SSO clusterisées ou non-clusterisées

SSO non-clusterisée limite le partage des informations d'autorisation pour les applications d'un même hôte virtuel. En outre, il n'y a aucun mécanisme de résilience en cas de défaillance de l'hôte. Les données d'authentification SSO clusterisées peuvent être partagées entre les applications de plusieurs hôtes virtuels et sont résistantes au basculement. De plus, SSO clusterisée est capable de recevoir des demandes d'un équilibreur de charges.

La façon dont SSO fonctionne

Si une ressource n'est pas protégée, un utilisateur n'a pas besoin de s'authentifier. Si un utilisateur accède à une ressource protégée, l'utilisateur devra s'authentifier.

Si l'authentification réussit, les rôles associés à l'utilisateur seront stockés et utilisés pour l'autorisation de toutes les ressources associées.

Si l'utilisateur se déconnecte d'une application, ou bien si l'application produit la session par programmation, toutes les données d'autorisation persistées seront retirées, et le processus recommencera.

Un timeout de session ne rend pas la session SSO valide si d'autres sessions sont encore valides.

Rapporter un bogue

17.2. SSO (SINGLE SIGN ON) CLUSTERISÉES POUR LES APPLICATIONS WEB

SSO (Single Sign On) est la possibilité pour les utilisateurs de s'authentifier à une application web unique au moyen d'une authentification réussie, afin d'obtenir l'autorisation de plusieurs autres applications. SSO clusterisée stocke les informations d'authentification et d'autorisation dans un cache clusterisé. Cela permet aux applications de serveurs différents de partager l'information et rend également l'information résistante à une défaillance de l'un des hôtes.

Une configuration SSO s'appelle une valve. Une valve est connectée à un domaine de sécurité, qui est configuré au niveau du serveur ou du groupe de serveurs. Chaque application qui doit partager les mêmes informations d'authentification mises en cache est configurée pour utiliser la même valve. Cette configuration s'effectue dans le fichier **`jboss-Web.xml`** de l'application.

Voici quelques valves SSO standards prises en charge par le sous-système de JBoss EAP 6 :

- Apache Tomcat ClusteredSingleSignOn
- Apache Tomcat IDPWebBrowserSSOValve
- SSO basée-SPNEGO de PicketLink

Suivant le type particulier de valve, vous aurez sans doute besoin d'ajouter des configurations supplémentaires au domaine de sécurité, pour que votre valve puisse fonctionner normalement.

Rapporter un bogue

17.3. CHOISIR L'IMPLÉMENTATION SSO QUI VOUS CONVIENT

JBoss EAP 6 exécute des applications de JBoss Enterprise Edition (EE), qui peuvent être des applications web, des applications EJB, des services web ou autres. SSO (Single Sign On) permet de propager les informations de contexte et d'identité de sécurité entre ces applications. Il existe plusieurs solutions SSO mais vous devrez choisir en fonction des besoins de votre organisation.

Notez qu'il y a une nette différence entre une application web en cluster et une SSO clusterisée. Une application web cluster est une application distribuée à travers les nœuds d'un cluster pour répartir la charge d'accueil de cette appplication. Si elle est indiquée come étant distribuable, toutes les nouvelles sessions et les changements aux sessions existantes seront répliqués à travers les autres membres du cluster. Une application est marquée comme pouvant être distribuée à travers l'ensemble de nœuds du cluster avec la balise <distributable/> dans le descripteur de déploiement **`web.xml`**. Une SSO clusterisée permet la réplication des informations d'identité et de contexte de sécurité, indépendamment du fait de savoir si les applications sont elles-mêmes clusterisées. Bien que ces technologies peuvent être utilisées conjointement, elles repésentent des concepts distincts.

SSO Desktop basé-Kerberos

Si votre organisation utilise déjà une authentification basée Kerberos et un système d'autorisation comme Microsoft Active Directory, vos pourrez utiliser les mêmes systèmes pour vous authentifier de façon transparente à vos applications Enterprise en cours d'exécution sur JBoss EAP 6.

SSO d'application web non clusterisée

Si vous exécutez un certain nombre d'applications sur une seule instance et que vous avez besoin d'activer une réplication de session SSO pour ces applications, la SSO non clusterisée sera la solution indiquée.

SSO Application Web et clusterisée

Si vous exécutez une ou plusieurs applications dans un cluster et que vous avez besoin d'activer une réplication de session SSO pour toutes ces applications, la SSO clusterisée sera la solution indiquée.

Rapporter un bogue

17.4. UTILISATION DE SSO (SINGLE SIGN ON) POUR LES APPLICATIONS WEB

Aperçu

Les fonctionnalités SSO (Single Sign On) sont fournies par les sous-systèmes web et Infinispan. Utiliser cette procédure pour configurer SSO dans les applications web.

Conditions préalables

- Vous devez avoir un domaine de sécurité configuré qui gère les authentifications et autorisations.
- Le sous-système **`infinispan`** doit être présent. Il se trouve dans le profil **`complet-ha`** pour un domaine géré, ou en utilisant la configuration **`standalone-full-ha.xml`** dans un serveur autonome.

- Le **web cache-conteneur** et le cache-conteneur SSO doivent chacun être présents. Les exemples de fichiers de configurations contiennent déjà le cache-conteneur **web**, et certaines des configurations contiennent déjà le cache-conteneur SSO également. Utilisez les commandes suivantes pour vérifier et activer le cache conteneur SSO. Notez que ces commandes modifient le profil **ha** d'un domaine géré. Vous pouvez modifier les commandes pour utiliser un profil différent, ou supprimer la portion **/profile=ha** de la commande, pour un serveur autonome.

Exemple 17.1. Vérifier le cache-conteneur web

Les profils et les configurations mentionnées ci-dessus incluent le cache-conteneur **web** par défaut. Utilisez la commande suivante pour vérifier sa présence. Si vous utilisez un profil différent, remplacez son nom à la place de **ha**.

```
/profile=ha/subsystem=infinispan/cache-container=web/:read-
resource(recursive=false,proxies=false,include-
runtime=false,include-defaults=true)
```

Si le résultat affiche un **success**, le sous-système sera présent. Sinon, vous devrez l'ajouter.

Exemple 17.2. Ajouter le cache-conteneur web

Utiliser les trois commandes suivantes pour activer le cache-conteneur **web** à votre configuration. Modifier le nom du profil selon le cas, ainsi que les autres paramètres. Ici, les paramètres sont ceux qui sont utilisés dans une configuration par défaut.

```
/profile=ha/subsystem=infinispan/cache-container=web:add(aliases=
["standard-session-cache"],default-
cache="repl",module="org.jboss.as.clustering.web.infinispan")
```

```
/profile=ha/subsystem=infinispan/cache-
container=web/transport=TRANSPORT:add(lock-timeout=60000)
```

```
/profile=ha/subsystem=infinispan/cache-container=web/replicated-
cache=repl:add(mode="ASYNC",batching=true)
```

Exemple 17.3. Vérifier le cache-conteneur SS0

Éxécuter la commande de l'interface CLI suivante :

```
/profile=ha/subsystem=infinispan/cache-container=web/:read-
resource(recursive=true,proxies=false,include-
runtime=false,include-defaults=true)
```

Chercher une sortie qui ressemble à ceci : **"sso" => {**

Si vous ne la trouvez pas, le cache-conteneur ne sera pas présent dans votre configuration.

Exemple 17.4. Ajouter le cache-conteneur SS0

```
/profile=ha/subsystem=infinispan/cache-container=web/replicated-cache=sso:add(mode="SYNC", batching=true)
```

- Le sous-système **web** doit être configuré pour utiliser SSO. La commande suivante active SSO sur le serveur virtuel appelé **default-host** et le domaine de cookies **domaine.com**. Le nom de cache est **sso**, et une nouvelle authentification est désactivée.

```
/profile=ha/subsystem=web/virtual-server=default-host/sso=configuration:add(cache-container="web",cache-name="sso",reauthenticate="false", domain="domain.com")
```

- Chaque application qui partage les informations SSO doit être configurée pour utiliser le même <security-domain> dans son descripteur de déploiement de **jboss-web.xml** et le même Realm dans son fichier de configuration **web.xml**.

Configurer une SSO clusterisée ou non-clusterisée

Configurer **sso** sous le sous-système web dans le profil utilisateur. La version **ClusteredSingleSignOn** est utilisée quand l'attribut **cache-container** est présent, sinon la classe **SingleSignOn** sera utilisée.

Exemple 17.5. Exemple de configuration SSO clusterisée

```
/subsystem=web/virtual-server=default-host/sso=configuration:add(cache-container="web",cache-name="sso",reauthenticate="false",domain="domain.com")
```

Exemple 17.6. Exemple de configuration SSO non clusterisée

```
/subsystem=web/virtual-server=default-host/sso=configuration:add(reauthenticate="false")
```

Rendre une session non valide

Une application peut rendre invalide une session en invoquant la méthode **javax.servlet.http.HttpSession.invalidate()**.

Rapporter un bogue

17.5. KERBEROS

Kerberos est un protocole d'authentification réseau pour les applications client/serveur. Il permet l'authentification sur un réseau non sécurisé en toute sécurité, à l'aide de la clé secrète de cryptographie symétrique.

Kerberos utilise les tokens de sécurité appelés «tickets». Pour utiliser un service sécurisé, vous devez obtenir un ticket auprès du Ticket Granting Service (TGS), qui est un service de délivrance de tickets exécuté sur un serveur sur votre réseau. Après avoir obtenu le ticket, vous pourrez demander un Ticket de Service (ST ou Service Ticket) d'un Service d'authentification (AS), qui est un autre service

s'exécutant sur votre réseau. Utilisez ensuite ST pour vous authentifier auprès du service que vous souhaitez utiliser. Les TGS et AS tous exécutent tous deux à l'intérieur d'un service intégré appelé KDC (Key Distribution Center).

Kerberos est conçu pour être utilisé dans un environnement client-serveur et est rarement utilisé dans les applications Web ou des environnements clients légers. Cependant, de nombreuses organisations utilisent déjà un système Kerberos pour l'authentification desktop et préfèrent réutiliser leur système existant plutôt que d'en créer un second pour leurs applications web. Kerberos est partie intégrante de Microsoft Active Directory et est également utilisé dans de nombreux environnements Red Hat Enterprise Linux.

Rapporter un bogue

17.6. SPNEGO

Simple and Protected GSS_API Negotiation Mechanism (SPNEGO) propose un mécanisme d'expension d'environnement SSO (Single Sign On) pour les applications web.

Lorsqu'une application d'ordinateur client, comme un navigateur web, tente d'accéder à une page de protection sur le serveur web, le serveur répond qu'une autorisation est requise. Ensuite, l'application demande un ticket de service du Centre de distribution des clés Kerberos (KDC). Après l'obtention du ticket, l'application le renvoie dans une requête formatée SPNEGO et l'envoie à l'application web, par l'intermédiaire du navigateur. Le conteneur web exécutant l'application web déployée ouvre la requête et authentifie le billet. Après une authentification réussie, l'accès est accordé.

SPNECO fonctionne avec tous les types de fournisseurs Kerberos, y compris le service Kerberos inclus dans Red Hat Enterprise Linux et le serveur Kerberos, qui fait partie intégrale du Microsoft Active Directory.

Rapporter un bogue

17.7. MICROSOFT ACTIVE DIRECTORY

Microsoft Active Directory est un service de répertoires développé par Microsoft pour authentifier les utilisateurs et les ordinateurs d'un domaine Microsoft Windows. Il est inclus dans Microsoft Windows Server. L'ordinateur dans Microsoft Windows Server est nommé le Contrôleur de domaine. Les serveurs de Red Hat Enterprise Linux exécutant le service Samba peuvent aussi agir comme Contrôleur de domaine dans ce type de réseau.

Active Directory repose sur trois technologies principales dépendantes les unes des autres :

- LDAP (Lightweight Directory Access Protocol) stocke les informations utilisateurs, ordinateurs, mots de passe et autres ressources.
- Kerberos procure une authentification sécurisée sur le réseau.
- DNS (Domain Service Name) fournit des mappages entre les adresses IP, les noms d'hôtes et les ordinateurs ou autres périphériques du réseau.

Rapporter un bogue

17.8. CONFIGURATION DE KERBEROS OU MICROSOFT ACTIVE DIRECTORY DESKTOP SSO POUR LES APPLICATIONS WEB

Introduction

Pour authentifier vos applications web ou EJB à l'aide de l'authentification basée Kerberos existante de votre organisation et de votre infrastructure d'autorisation, comme Microsoft Active Directory, vous pouvez utiliser les fonctionnalités de JBoss Negotiation intégrées dans JBoss EAP 6. Si vous configurez votre application web correctement, une connexion réussie en desktop ou réseau sera suffisante pour vous authentifier en toute transparence dans votre application web, donc aucune invite de connexion supplémentaire ne sera nécessaire.

Différence par rapport aux versions précédentes de la plateforme

Il existe des différences notables entre la plateforme JBoss EAP 6 et les versions précédentes :

- Les domaines de sécurité sont configurés centralement, pour chaque profil d'un domaine géré, ou pour chaque serveur autonome. Ils ne font pas partie du déploiement lui-même. Le domaine de sécurité qu'un déploiement doit utiliser est nommé dans le fichier de déploiement **`jboss-web.xml`** ou **`jboss-ejb3.xml`**.
- Les propriétés de sécurité sont configurées dans le cadre du domaine de sécurité, faisant partie de sa configuration centrale. Elles ne font pas partie du déploiement.
- Vous pouvez ne plus remplacer les authentificateurs dans votre déploiement. Toutefois, vous pouvez ajouter une valve NegotiationAuthenticator à votre descripteur **`jboss-web.xml`** afin d'obtenir le même effet. La valve nécessite toujours les éléments **`<security-constraint>`** et **`<login-config>`** à définir dans le fichier **`web.xml`**. Ils servent à décider quelles ressources sont sécurisées. Cependant, l'auth-method choisie sera substituée par la valve NegotiationAuthenticator du fichier **`jboss-web.xml`**.
- Les attributs **`CODE`** des domaines de sécurité utilisent maintenant un nom simple au lieu d'un nom de classe complet. Le tableau suivant montre les mappages entre les classes utilisées pour JBoss Negotiation, et leurs classes.

Tableau 17.1. Codes de modules de connexion et Noms de classes

Nom simple	Nom de classe	But
Kerberos	com.sun.security.auth.module.Krb5LoginModule com.ibm.security.auth.module.Krb5LoginModule	Module de connexion Kerberos quand on utilise le JDK d'Oracle. Module de connexion Kerberos quand on utilise le JDK d'IBM.
SPNEGO	org.jboss.security.negotiation.spnego.SPNEGOLoginModule	Le mécanisme qui permet aux applications web de s'authentifier à votre serveur d'authentification Kerberos.
AdvancedLdap	org.jboss.security.negotiation.AdvancedLdapLoginModule	Utilisé avec les serveurs LDAP autres que Microsoft Active Directory.
AdvancedAdLdap	org.jboss.security.negotiation.AdvancedADLoginModule	Utilisé avec les serveurs Microsoft Active Directory LDAP.

JBoss Negotiation Toolkit

`JBoss Negotiation Toolkit` est un outil de débogage que vous pouvez télécharger à partir de https://community.jboss.org/servlet/JiveServlet/download/16876-2-34629/jboss-negotiation-toolkit.war. Il est fourni comme un outil supplémentaire pour vous aider à déboguer et tester les mécanismes

d'authentification avant d'introduire votre application en production. C'est un outil non pris en charge, mais considéré comme très utile. Comme SPNEGO, cet outil peut être difficile à configurer pour les applications web.

Procédure 17.1. Installation de l'authentification SSO (Single Sign On) pour les Applications Web ou EJB

1. **Configurer un domaine de sécurité qui représente l'identité de votre serveur. Définir les propriétés système si nécessaire.**
   Le premier domaine de sécurité authentifie le conteneur lui-même au service de répertoires. Il a besoin d'utiliser un module de connexion qui accepte un type de mécanisme de connexion statique, car un utilisateur réel n'est pas impliqué. Cet exemple utilise une entité de sécurité statique et fait référence à un fichier keytab qui contient les informations d'identification.

   Le code XML est donné ici dans un but de clarification, mais vous devez utiliser la console de gestion ou le Management CLI pour configurer vos domaines de sécurité.

   ```
   <security-domain name="host" cache-type="default">
      <authentication>
         <login-module code="Kerberos" flag="required">
            <module-option name="storeKey" value="true"/>
            <module-option name="useKeyTab" value="true"/>
            <module-option name="principal"
   value="host/testserver@MY_REALM"/>
            <module-option name="keyTab"
   value="/home/username/service.keytab"/>
            <module-option name="doNotPrompt" value="true"/>
            <module-option name="debug" value="false"/>
         </login-module>
      </authentication>
   </security-domain>
   ```

2. **Configurer un second domaine de sécurité pour sécuriser l'application web ou les applications. Définir les propriétés système si nécessaire.**
   Le deuxième domaine de sécurité est utilisé pour authentifier l'utilisateur particulier au serveur d'authentification Kerberos ou SPNEGO. Vous avez besoin d'au moins un module de connexion pour authentifier l'utilisateur et un autre à la recherche de rôles à appliquer à l'utilisateur. Le code XML suivant montre un exemple de domaine de sécurité SPNEGO. Il inclut un module d'autorisation pour mapper les rôles aux utilisateurs individuels. Vous pouvez également utiliser un module de recherche pour les rôles sur le serveur d'authentification lui-même.

   ```
   <security-domain name="SPNEGO" cache-type="default">
      <authentication>
         <!-- Check the username and password -->
         <login-module code="SPNEGO"  flag="requisite">
            <module-option name="password-stacking"
   value="useFirstPass"/>
            <module-option name="serverSecurityDomain" value="host"/>
         </login-module>
         <!-- Search for roles -->
         <login-module code="UsersRoles" flag="required">
            <module-option name="password-stacking"
   value="useFirstPass" />
            <module-option name="usersProperties" value="spnego-
   users.properties" />
   ```

```
         <module-option name="rolesProperties" value="spnego-
roles.properties" />
      </login-module>
   </authentication>
</security-domain>
```

3. **Spécifier la security-constraint et la login-config dans le fichier `web.xml`**
   Le descripteur **`web.xml`** contient des informations sur les contraintes de sécurité et sur la configuration de la connexion. En voici des exemples :

```
<security-constraint>
   <display-name>Security Constraint on Conversation</display-name>
   <web-resource-collection>
      <web-resource-name>examplesWebApp</web-resource-name>
      <url-pattern>/*</url-pattern>
   </web-resource-collection>
   <auth-constraint>
   <role-name>RequiredRole</role-name>
   </auth-constraint>
</security-constraint>

<login-config>
   <auth-method>SPNEGO</auth-method>
   <realm-name>SPNEGO</realm-name>
</login-config>

<security-role>
   <description> role required to log in to the
Application</description>
   <role-name>RequiredRole</role-name>
</security-role>
```

4. **Indiquer le domaine de sécurité et les autres paramètres dans le descripteur `jboss-web.xml`.**
   Spécifiez le nom du domaine de la sécurité côté client (l'autre dans cet exemple) dans le descripteur **`jboss-Web.xml`** de votre déploiement, pour obliger votre application à utiliser ce domaine de sécurité.

   Vous ne pouvez plus remplacer les authentificateurs directement. À la place, vous devez ajouter NegotiationAuthenticator comme valve au descripteur **`jboss-web.xml`** si nécessaire. **`<jacc-star-role-allow>`** vous permet d'utiliser un astérisque (*) pour faire correspondre plusieurs noms de rôles, et est optionnel.

```
<jboss-web>
   <security-domain>java:/jaas/SPNEGO</security-domain>
   <valve>
      <class-
name>org.jboss.security.negotiation.NegotiationAuthenticator</class-
name>
   </valve>
   <jacc-star-role-allow>true</jacc-star-role-allow>
</jboss-web>
```

5. **Ajouter une dépendance au `MANIFEST.MF` de votre application, pour trouver l'emplacement des classes Negotiation.**
   L'application web a besoin d'une dépendance sur la classe **`org.jboss.security.negotiation`** à ajouter au manifeste **`META-INF/MANIFEST.MF`** du déploiement pour pouvoir localiser les classes JBoss Negotiation. Ce qui suit vous montre une entrée formatée correctement.

   ```
   Manifest-Version: 1.0
   Build-Jdk: 1.6.0_24
   Dependencies: org.jboss.security.negotiation
   ```

   - Sinon, ajouter une dépendance à votre application en modifiant le fichier **`META-INF/jboss-deployment-structure.xml`** :

     ```
     <?xml version="1.0" encoding="UTF-8"?>
     <jboss-deployment-structure>
       <deployment>
         <dependencies>
      <module name='org.jboss.security.negotiation'/>
         </dependencies>
       </deployment>
     </jboss-deployment-structure>
     ```

Résultat

Votre application web accepte et authentifie les informations d'identification avec Kerberos, Active Directory de Microsoft ou autre service de répertoire compatible SPNEGO. Si l'utilisateur exécute l'application d'un système qui est déjà enregistré dans le service de répertoires, et où les rôles sont déjà appliqués à l'utilisateur, l'application web n'aura pas besoin d'authentification, et les fonctionnalités SSO seront opérationnelles.

Rapporter un bogue

17.9. CONFIGURER SPNEGO AVEC UN RENVOI À L'AUTHENTIFICATION FORM

Suivre la procédure ci-dessous pour configurer un SPNEGO avec un renvoi à l'authentification Form.

Procédure 17.2. Sécurité SPNEGO avec renvoi à l'authentification Form

1. **Configurer SPNEGO**
   Voir la procédure décrite dans Section 17.8, « Configuration de Kerberos ou Microsoft Active Directory Desktop SSO pour les applications web »

2. **Modifier le fichier `web.xml`**
   Ajouter un élément **`login-config`** à votre application et configurer les pages de connexion et d'erreurs dans le fichier web.xml :

   ```
   <login-config>
       <auth-method>SPNEGO</auth-method>
       <realm-name>SPNEGO</realm-name>
           <form-login-config>
               <form-login-page>/login.jsp</form-login-page>
   ```

```
            <form-error-page>/error.jsp</form-error-page>
        </form-login-config>
    </login-config>
```

3. **Ajouter un contenu web**
   Ajouter des references de **`login.html`** et de **`error.html`** à **`web.xml`**. Ces fichiers sont ajoutés à l'archive d'application web à l'emplacement spécifié dans la configuration **`form-login-config`**. Pour obtenir des informations, voir la section *Enable Form-based Authentication* du *Security Guide* de JBoss EAP 6. Un fichier **`login.html`** typique ressemble à ceci :

```
<html>
    <head>
        <title>Vault Form Authentication</title>
    </head>
    <body>
        <h1>Vault Login Page</h1>
        <p>
        <form method="post" action="j_security_check">
        <table>
            <tr>
                <td>Username</td><td>-</td>
                <td><input type="text" name="j_username"></td>
            </tr>
            <tr>
                <td>Password</td><td>-</td>
                <td><input type="password" name="j_password"></td>
            </tr>
            <tr>
                <td colspan="2"><input type="submit"></td>
            </tr>
        </table>
        </form>
        </p>
        <hr>
    </body>
</html>
```

NOTE

Le renvoi à la logique FORM n'est disponible que dans le cas où il n'y a pas de SPNEGO (ni de NTLM). De ce fait, un formulaire de connexion n'est pas présenté au navigateur si le navigateur envoie un token NTLM.

Rapporter un bogue

CHAPITRE 18. RÉFÉRENCES DE SÉCURITÉ POUR LE DÉVELOPPEMENT

18.1. RÉFÉRENCE DE CONFIGURATION JBOSS-WEB.XML

Introduction

Le fichier **`jboss-web.xml`** se trouve dans **`WEB-INF`** ou dans le répertoire **`META-INF`** de votre déploiement. Il contient des informations de configuration sur des fonctionnalités que le conteneur JBoss Web ajoute au Servlet 3.0. Les paramètres de configuration spécifiques à la spécification de Servlet 3.0 se trouvent dans **`web.xml`**, dans le même répertoire.

L'élément qui se trouve tout en haut dans le fichier **`jboss-web.xml`** est l'élément **`<jboss-web>`**.

Mappage des ressources globales aux prérequis WAR

La plupart des paramètres de configuration font correspondre les prérequis définis dans le fichier **`web.xml`** de l'application à des ressources locales. Pour plus d'explications sur les paramètres de configuration de **`web.xml`** consulter http://docs.oracle.com/cd/E13222_01/wls/docs81/webapp/web_xml.html.

Ainsi, si **`web.xml`** requiert **`jdbc/MyDataSource`**, alors **`jboss-web.xml`** fera sans doute correspondre la source de données globale **`java:/DefaultDS`** pour remplir ce besoin. Le WAR utilise la source de données globale pour faire correspondre **`jdbc/MyDataSource`**.

Tableau 18.1. Attributs de haut niveau communs

Attribut	Description
env-entry	Un mappage à **`env-entry`** requis par **`web.xml`**.
ejb-ref	Un mappage à **`ejb-ref`** requis par **`web.xml`**.
ejb-local-ref	Un mappage à **`ejb-local-ref`** requis par **`web.xml`**.
service-ref	Un mappage à **`service-ref`** requis par **`web.xml`**.
resource-ref	Un mappage à **`resource-ref`** requis par **`web.xml`**.
resource-env-ref	Un mappage à **`resource-env-ref`** requis par **`web.xml`**.
message-destination-ref	Un mappage à **`message-destination-ref`** requis par **`web.xml`**.
persistence-context-ref	Un mappage à **`persistence-context-ref`** requis par **`web.xml`**.

Attribut	Description
persistence-unit-ref	Un mappage à **`persistence-unit-ref`** requis par **`web.xml`**.
post-construct	Un mappage à **`post-context`** requis par **`web.xml`**.
pre-destroy	Un mappage à **`pre-destroy`** requis par **`web.xml`**.
data-source	Un mappage à **`data-source`** requis par **`web.xml`**.
context-root	Le contexte root de l'application. La valeur par défaut est le nom du déploiement sans le suffixe **`.war`**.
virtual-host	Le nom de l'hôte virtuel HTTP à partir duquel l'application accepte les requêtes. Se réfère au contenu de l'en-tête de l'**`Host`** HTTP.
annotation	Décrit une annotation utilisée par l'application. Voir <annotation> pour plus d'information.
listener	Décrit un listener utilisée par l'application. Voir <listener> pour plus d'information.
session-config	Cet élément remplit la même fonction que l'élément **`<session-config>`** du fichier **`web.xml`** et est inclus dans un but de compatibilité seulement.
valve	Décrit une valve utilisée par l'application. Voir <valve> pour plus d'information.
overlay	Le nom d'une couche supplémentaire à ajouter à l'application.
security-domain	Le nom du domaine de sécurité utilisé par l'application. Le domaine de sécurité lui-même est configuré à partir de la console de gestion basée web ou le Management CLI.
security-role	Cet élément remplit la même fonction que l'élément **`<session-role>`** du fichier **`web.xml`** et est inclus dans un but de compatibilité seulement.

Attribut	Description
use-jboss-authorization	Si cet élément est présent et contient la valeur non sensible à la casse "true", la pile d'autorisation web JBoss sera utilisée. S'il n'est pas présent ou contient une valeur non "true", alors seuls les mécanismes d'autorisation indiqués dans les spécifications Java Enterprise Edition seront utilisés. Cet élément est nouveau dans JBoss EAP 6.
disable-audit	Définir cet élément de booléen à **`false`** pour activer et à **`true`** pour désactiver l'auditing web. L'auditing de sécurité web ne fait pas partie de la spécification Java EE. Cet élément est nouveau dans JBoss EAP 6.
disable-cross-context	Si sur **`false`**, l'application sera en mesure d'appeler un autre contexte d'application. La valeur par défaut est **`true`**.

Les éléments suivants ont chacun des éléments enfants.

<annotation>

Décrit une annotation utilisée par l'application. Le tableau suivant liste les éléments enfants d'une **`<annotation>`**.

Tableau 18.2. Éléments de configuration d'une annotation

Attribut	Description
class-name	Nom de la classe de l'annotation
servlet-security	L'élément, comme **`@ServletSecurity`**, qui représente la sécurité du servlet.
run-as	L'élément, comme **`@RunAs`**, qui représente l'information run-as
multi-part	L'élément, comme **`@MultiPart`**, qui représente l'information multi-part.

<listener>

Décrit un listener. Le tableau suivant liste les éléments enfants d'un **`<listener>`**.

Tableau 18.3. Éléments de configuration d'un listener

Attribut	Description
class-name	Nom de la classe du listener

Attribut	Description
listener-type	Liste les éléments de **`condition`** qui indiquent quel sorte de listener ajouter au contexte de l'application. Les choix valides sont les suivants : **CONTAINER** Ajoute un ContainerListener au contexte. **LIFECYCLE** Ajoute un LifecycleListener au contexte. **SERVLET_INSTANCE** Ajoute un InstanceListener au contexte **SERVLET_CONTAINER** Ajoute un WrapperListener au contexte. **SERVLET_LIFECYCLE** Ajoute un WrapperLifecycle au contexte.
module	Le nom du module qui contient la classe du listener.
param	Un paramètre. Contient deux éléments enfants, **`<param-name>`** et **`<param-value>`**.

<valve>

Décrit une valve de l'application. Contient les mêmes éléments de configuration que <listener>.

Rapporter un bogue

18.2. RÉFÉRENCE DE PARAMÈTRE DE SÉCURITÉ EJB

Tableau 18.4. Éléments de paramètres de sécurité EJB

Élément	Description
`<security-identity>`	Contient des éléments enfants relatifs à l'identité de sécurité d'un EJB.
`<use-caller-identity />`	Indique que l'EJB utilise la même identité de sécurité que l'appelant.
`<run-as>`	Contient un élément **`<role-name>`**.
`<run-as-principal>`	Si présent, indique le principal assigné aux appels sortants. Si non présent, les appels sortants sont assignés à un principal nommé **`anonymous`**.

Élément	Description
<role-name>	Spécifie le role d'exécution de l'EJB.
<description>	Décrit le role nommé dans **<role-name>** .

Exemple 18.1. Exemples d'identité de sécurité

Cet exemple décrit chaque balise décrite dans Tableau 18.4, « Éléments de paramètres de sécurité EJB ». Peuvent également être mis dans un **<session>**.

```
<ejb-jar>
    <enterprise-beans>
        <session>
            <ejb-name>ASessionBean</ejb-name>
            <security-identity>
                <use-caller-identity/>
            </security-identity>
        </session>
        <session>
            <ejb-name>RunAsBean</ejb-name>
            <security-identity>
                <run-as>
                    <description>A private internal role</description>
                    <role-name>InternalRole</role-name>
                </run-as>
            </security-identity>
        </session>
    <session>
    <ejb-name>RunAsBean</ejb-name>
    <security-identity>
    <run-as-principal>internal</run-as-principal>
    </security-identity>
    </session>
    </enterprise-beans>
</ejb-jar>
```

Rapporter un bogue

CHAPITRE 19. RÉFÉRENCES SUPPLÉMENTAIRES

19.1. TYPES D'ARCHIVES JAVA

JBoss EAP 6 reconnaît différents types de fichiers d'archives. Les fichiers d'archives sont utilisés pour empaqueter des services et des applications déployables.

En général, les fichiers d'archives sont des archives Zip, avec des extensions de fichiers spécifiques et des structures de répertoires spécifiques. Si l'archive Zip est extraite avant d'être déployée sur le serveur d'application, on l'appelle archive explosée. Dans ce cas, le nom du répertoire contient toujours l'extension de fichier, et les conditions de structure de répertoire continuent de s'appliquer.

Tableau 19.1.

Type d'archive	Extension	But	Exigences pour la structure de répertoire
Archive Java	.jar	Contient les bibliothèques de classes Java.	Le fichier **`META-INF/MANIFEST.MF`** (optionnel), spécifiant des informations comme quelle classe est la classe **`principale`** .
Archive Web	.war	Contient des fichiers Java Server Pages (JSP), des servlets, et des fichiers XML, en plus des classes et des bibliothèques. Le contenu Archive Web s'appelle aussi Web Application.	Le fichier **`WEB-INF/web.xml`**, qui contient des informations sur la structure de l'application web. Il y a également d'autres fichiers présents dans **`WEB-INF/`**.
Resource Adapter Archive	.rar	La structure du répertoire est dans la spécification JCA.	Contient un adaptateur de ressources Java Connector Architecture (JCA). S'appelle également un connecteur.
Enterprise Archive	.ear	Utilisé par Java Enterprise Edition (EE) pour empaqueter un ou plusieurs modules dans une simple archive, pour que les modules puissent être déployés dans le serveur d'applications simultanément. Maven et Ant sont les outils les plus communément utilisés pour générer les archives EAR.	Le répertoire **`META-INF/`** qui contient un ou plusieurs fichiers de descripteurs de déploiement XML.

Type d'archive	Extension	But	Exigences pour la structure de répertoire
			N'importe quel type de module. • WAR (Web Archive) • Une ou plusieurs JAR (Java Archives) contenant des POJO (Plain Old Java Objects) • Un ou plusieurs modules Enterprise JavaBean (EJB), contenant son propre répertoire **`META-INF/`**. Ce répertoire inclut des descipteurs pour les classes de persistence qui sont déployées. • Une ou plusieurs Archives de ressources (RAR).
Service Archive	.sar	Ressemble à une archive Enterprise, spécifique à Enterprise Application Platform.	Répertoire **`META-INF/`** contenant un fichier **`jboss-service.xml`** ou **`jboss-beans.xml`**.

Rapporter un bogue

ANNEXE A. HISTORIQUE DE RÉVISION

Version 6.3.0-28 **Friday July 25 2014** **Lucas Costi**

Red Hat JBoss Enterprise Application Platform 6.3.0.GA