



JBoss Enterprise Application Platform 6.3

移行ガイド

Red Hat JBoss Enterprise Application Platform 6 向け

JBoss Enterprise Application Platform 6.3 移行ガイド

Red Hat JBoss Enterprise Application Platform 6 向け

Red Hat カスタマーコンテンツサービス JBoss EAP チーム

法律上の通知

Copyright © 2014 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.

概要

本書は、以前のバージョンの Red Hat JBoss Enterprise Application Platform からアプリケーションを移行するためのガイドです。

目次

第1章 はじめに	5
1.1. RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM 6	5
1.2. 移行ガイドの概要	5
第2章 移行の準備	6
2.1. 移行の準備	6
2.2. JBoss EAP 6 の新機能と変更内容	6
2.3. 廃止および未サポート機能リストの確認	8
第3章 アプリケーションの移行	9
3.1. ほとんどのアプリケーションで必要な変更	9
3.1.1. ほとんどのアプリケーションで必要な変更の確認	9
3.1.2. クラスローディングの変更	9
3.1.2.1. クラスローディングの変更によるアプリケーションの更新	9
3.1.2.2. モジュールの依存関係の理解	9
3.1.2.3. クラスローディングの変更によるアプリケーション依存関係の更新	10
3.1.3. 設定ファイルの変更	10
3.1.3.1. JBoss EAP 6 のクラスローディングを制御するファイルの作成または変更	10
3.1.3.2. jboss-deployment-structure.xml	14
3.1.3.3. 新しいモジュラークラスローディングシステムのパッケージリソース	14
3.1.3.4. ResourceBundle プロパティの場所変更	15
3.1.3.5. カスタムモジュールの作成	15
3.1.4. ロギングの変更	17
3.1.4.1. ロギング依存関係の編集	17
3.1.4.2. サードパーティーロギングフレームワークのアプリケーションコードの更新	17
3.1.4.3. 新しい JBoss ロギングフレームワークを使用したコードの変更	19
3.1.5. アプリケーションパッケージの変更	20
3.1.5.1. EAR および WAR パッケージの編集	20
3.1.6. データソースおよびリソースアダプター設定の変更	21
3.1.6.1. 設定変更によるアプリケーションの更新	21
3.1.6.2. DataSource 設定の更新	21
3.1.6.3. JDBC ドライバーのインストールと設定	22
3.1.6.4. Hibernate または JPA 用のデータソースの設定	27
3.1.6.5. リソースアダプター設定の更新	27
3.1.7. セキュリティーの変更	28
3.1.7.1. アプリケーションセキュリティの変更設定	28
3.1.7.2. PicketLink STS および Web サービスを使用するアプリケーションの更新	29
3.1.8. JNDI の変更	30
3.1.8.1. アプリケーションの JNDI 名前空間名の更新	30
3.1.8.2. 移植可能な EJB JNDI 名	31
3.1.8.3. JNDI 名前空間のルールの確認	31
3.1.8.4. JNDI 名前空間の新ルールに準拠するようアプリケーションを変更	32
3.1.8.5. 以前のリリースでの JNDI 名前空間の例、および JBoss EAP 6 での名前空間の指定方法	33
3.2. アプリケーションのアーキテクチャーやコンポーネントによって異なる変更	34
3.2.1. アプリケーションのアーキテクチャーやコンポーネントによって異なる変更の確認	34
3.2.2. Hibernate および JPA の変更	35
3.2.2.1. Hibernate や JPA を使用するアプリケーションの更新	35
3.2.2.2. Hibernate および JPA を使用するアプリケーションの変更設定	35
3.2.2.3. 永続ユニットプロパティ	36
3.2.2.4. Hibernate 4 を使用するよう Hibernate 3 のアプリケーションを更新する	38
3.2.2.5. Hibernate アイデンティティ自動生成値の既存動作の保持	39
3.2.2.6. Hibernate 3.3.x アプリケーションの Hibernate 4.x への移行	40

3.2.2.7. Hibernate 3.5.x アプリケーションの Hibernate 4.x への移行	40
3.2.2.8. クラスター環境で稼働する、移行された Seam および Hibernate アプリケーションの永続プロパティの変更	41
3.2.2.9. JPA 2.0 の仕様に準拠するようアプリケーションを更新する	42
3.2.2.10. Infinispan による JPA/Hibernate 2 次キャッシュの置き換え	42
3.2.2.11. Hibernate キャッシュプロパティ	44
3.2.2.12. Hibernate Validator 4 への移行	45
3.2.3. JSF の変更	46
3.2.3.1. アプリケーションが JSF の古いバージョンを使用できるようにする	46
3.2.4. Web サービスの変更	47
3.2.4.1. Web サービスの変更	47
3.2.5. JAX-RS および RESTEasy の変更	50
3.2.5.1. JAX-RS および RESTEasy の変更の設定	50
3.2.6. LDAP セキュリティーレルムの変更	51
3.2.6.1. LDAP セキュリティーレルムの変更設定	51
3.2.7. HornetQ の変更	52
3.2.7.1. HornetQ および NFS について	52
3.2.7.2. 既存の JMS メッセージを JBoss EAP 6 へ移行するために JMS ブリッジを設定する	53
3.2.7.3. JMS ブリッジの作成	53
3.2.7.4. HornetQ を JMS プロバイダーとして使用するためにアプリケーションを移行	58
3.2.7.5. HornetQ でのメッセージングの設定	59
3.2.8. クラスタリングの変更	59
3.2.8.1. クラスタリングに対するアプリケーションの変更	59
3.2.8.2. HA シングルトンの実装	64
3.2.9. サービススタイルデプロイメントの変更	70
3.2.9.1. サービススタイルデプロイメントを使用するアプリケーションの更新	70
3.2.10. リモート呼び出しの変更	70
3.2.10.1. JBoss EAP 5 にデプロイされ、JBoss EAP 6 へリモート呼び出しを行うアプリケーションの移行	70
3.2.10.2. JNDI を使用したリモートでのセッション Bean の呼び出し	72
3.2.10.3. EJB JNDI の名前に関する参考資料	75
3.2.11. EJB 2.x の変更	76
3.2.11.1. EJB 2.x を使用するアプリケーションの更新	76
3.2.12. JBoss AOP 変更	82
3.2.12.1. JBoss AOP を使用するアプリケーションの更新	82
3.2.13. Seam 2.2 アプリケーションの移行	83
3.2.13.1. Seam 2.2 アーカイブの JBoss EAP 6 への移行	83
3.2.13.2. Seam 2.2 アーカイブの移行の問題	87
3.2.14. Spring アプリケーションの移行	89
3.2.14.1. Spring アプリケーションの移行	89
3.2.15. 移行に影響するその他の変更	90
3.2.15.1. 移行に影響する可能性があるその他の変更について理解する	90
3.2.15.2. Maven プラグイン名の変更	90
3.2.15.3. クライアントアプリケーションの変更	90

第4章 ツールとヒント 91

4.1. 移行に役立つリソース	91
4.1.1. 移行に役立つリソース	91
4.1.2. 移行に便利なツールについて理解する	91
4.1.3. Tattletale を用いたアプリケーション依存関係の検索	91
4.1.4. Tattletale のダウンロードとインストール	92
4.1.5. Tattletale レポートの作成および確認	92
4.1.6. IronJacamar ツールを使用してデータソースとリソースアダプターの設定を移行する	93

4.1.7. IronJacamar 移行ツールのダウンロードとインストール	93
4.1.8. IronJacamar 移行ツールを使用したデータソース設定ファイルの変換	94
4.1.9. IronJacamar 移行ツールを使用したリソースアダプター設定ファイルの変換	96
4.2. 移行の問題のデバッグ	100
4.2.1. 移行の問題のデバッグと解決	100
4.2.2. ClassNotFoundExceptions および NoClassDefFoundErrors のデバッグと解決	101
4.2.3. JBoss モジュール依存関係の検索	101
4.2.4. 以前のインストールでの JAR の検索	102
4.2.5. ClassCastException のデバッグと解決	103
4.2.6. DuplicateServiceExceptions のデバッグと解決	103
4.2.7. JBoss Seam のデバッグページエラーのデバッグと解決	104
4.3. サンプルアプリケーションの移行の確認	105
4.3.1. サンプルアプリケーションの移行の確認	106
4.3.2. Seam 2.2 JPA サンプルの JBoss EAP 6 への移行	106
4.3.3. Seam 2.2 Booking サンプルの JBoss EAP 6 への移行	107
4.3.4. Seam 2.2 Booking アーカイブの JBoss EAP 6 への移行: 手順説明	111
4.3.5. JBoss EAP 5.X バージョンの Seam 2.2 Booking アプリケーションのビルドおよびデプロイ	112
4.3.6. Seam 2.2 Booking アーカイブのデプロイメントエラーや例外のデバッグおよび解決	113
4.3.7. Seam 2.2 Booking アーカイブのランタイムエラーや例外のデバッグおよび解決	121
4.3.8. Seam 2.2 Booking アプリケーションの移行時に加えられる変更概要の確認	125
付録A 改訂履歴	127

第1章 はじめに

1.1. RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM 6

Red Hat JBoss Enterprise Application Platform 6 (JBoss EAP 6) は、オープンな標準に基いて構築され、Java Enterprise Edition 6 の仕様に準拠するミドルウェアプラットフォームです。高可用性クラスターリング、メッセージング、分散キャッシングなどの技術が JBoss Application Server 7 と統合されます。

JBoss EAP 6 には、必要な場合にだけサービスを有効にできる新しいモジュール構造が含まれます (サービスの起動時間が短縮されます)。

管理コンソールと管理コマンドラインインターフェースにより、XML 設定ファイルの編集が不要になり、タスクをスクリプト化および自動化する機能が追加されました。

また、JBoss EAP 6 には、セキュアでスケーラブルな Java EE アプリケーションの迅速な開発を可能にする API と開発フレームワークが含まれます。

[バグを報告する](#)

1.2. 移行ガイドの概要

JBoss EAP 6 は Java Enterprise Edition 6 仕様の高速、軽量、および強力な実装です。アーキテクチャーはモジュラーサービスコンテナ上に構築され、アプリケーションが必要なときにオンデマンドでサービスを有効にできます。このような新しいアーキテクチャーが導入されたため、JBoss EAP 5 で実行するアプリケーションを JBoss EAP 6 で実行する場合に変更が必要になることがあります。

本ガイドは、JBoss EAP 5.1 のアプリケーションを JBoss EAP 6 で正常に実行し、デプロイするために必要な変更を記載することを目的としています。また、デプロイメントおよび実行時の問題を解決する方法や、アプリケーションの挙動が変更されないようにする方法を説明します。これは新しいプラットフォームへ移行する第一歩となります。アプリケーションが正常にデプロイされ、実行された後、JBoss EAP 6 の新機能を使用するための各コンポーネントのアップグレードを計画できます。

[バグを報告する](#)

第2章 移行の準備

2.1. 移行の準備

アプリケーションサーバーの構造が以前のバージョンと異なるため、移行について調査し、移行計画を立ててからアプリケーションを移行してください。

1. JBoss EAP 6 の新機能と変更内容

本リリースには、JBoss EAP 5 のアプリケーションのデプロイメントに影響する可能性がある変更が複数あります。これには、ファイルディレクトリー構造、スクリプト、デプロイメント設定、クラスローディング、JNDI ルックアップなどの変更が含まれます。詳細は、「[JBoss EAP 6 の新機能と変更内容](#)」を参照してください。

2. スタートガイドの確認

https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/にある JBoss EAP 6 向け『開発ガイド』の章「アプリケーションの開発」を必ず参照してください。これには以下に関する重要な情報が含まれます。

- Java EE 6
- 新しいモジュール形式クラスローディングシステム
- ファイル構造の変更
- JBoss EAP 6 のダウンロードおよびインストール方法
- JBoss Developer Studio のダウンロードおよびインストール方法
- 各開発環境に対する Maven の設定方法
- 製品に同梱されたクイックスタートサンプルアプリケーションのダウンロードおよび実行方法

3. Maven プロジェクトで JBoss EAP 6 の依存関係を使用する方法を理解する

https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/にある JBoss EAP 6 『開発ガイド』の「Maven ガイド」の章を確認してください。「プロジェクト依存関係の管理」の項には、プロジェクトが JBoss EAP の BOM (Bill of Material) アーティファクトを使用するよう設定する方法に関する重要な情報が含まれています。

4. アプリケーションの分析および理解

アプリケーションはそれぞれ異なるため、移行を開始する前に既存アプリケーションのコンポーネントやアーキテクチャーについて十分に理解する必要があります。



重要

アプリケーションを変更する前に、必ずバックアップコピーを作成するようにしてください。

[バグを報告する](#)

2.2. JBOSS EAP 6 の新機能と変更内容

はじめに

JBoss EAP 6 が以前のリリースと顕著に異なる点は次の通りです。

モジュールベースのクラスローディング

JBoss EAP 5 ではクラスローディングのアーキテクチャは階層的でした。JBoss EAP 6 ではクラスローディングが JBoss モジュールベースとなりました。これにより、正確にアプリケーションを分離できるようになったため、サーバー実装クラスを隠し、アプリケーションが必要なクラスのみをロードできるようになりました。より優れたパフォーマンスを実現するため、クラスローディングは平行して実行されます。JBoss EAP 5 向けに書かれたアプリケーションはモジュールの依存関係を指定するために変更する必要があります。場合によってはアーカイブを再パッケージ化する必要があることもあります。詳細について

は、https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/にある JBoss EAP 6 向け『開発ガイド』の章「クラスローディングとモジュール」の「クラスローディングとモジュールの概要」を参照してください。

ドメイン管理

JBoss EAP 6 ではサーバーをスタンドアロンサーバーとして実行したり、管理対象ドメインで実行することが可能です。管理対象ドメインではサーバーグループ全体を一度に設定できるため、サーバーのネットワーク全体で設定を同期化できます。これにより、以前のリリース向けに構築されたアプリケーションが影響を受けることはありませんが、複数サーバーへのデプロイメントの管理を簡素化できます。詳細について

は、https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/にある JBoss EAP 6 向け『管理および設定ガイド』の「管理対象ドメイン」を参照してください。

デプロイメント設定

スタンドアロンサーバーと管理対象ドメイン

JBoss EAP 5 ではプロファイルベースのデプロイメント設定を使用し、これらのプロファイルは **EAP_HOME/server/** ディレクトリーにありました。多くのアプリケーションにはセキュリティ、データベース、リソースアダプターなどの設定に対する複数の設定ファイルが含まれていました。JBoss EAP 6 では 1 つのファイルを使用してデプロイメントを設定できるようになりました。このファイルは、デプロイメントに使用されるすべてのサービスやサブシステムを設定するために使用されます。スタンドアロンサーバーは

EAP_HOME/standalone/configuration/standalone.xml ファイルを使用して設定されます。管理対象ドメインで実行されているサーバーでは、サーバーは

EAP_HOME/domain/configuration/domain.xml ファイルを使用して設定されます。JBoss EAP 5 の複数の設定ファイルに含まれる情報は、新しい単一の設定ファイルへ移行する必要があります。

デプロイメントの順序付け

JBoss EAP 6 は、デプロイメントに対して高速で平行した初期化を実行するため、パフォーマンスと効率性が向上します。ほとんどの場合でアプリケーションサーバーは自動的に依存関係を事前判断し、最も効率的なデプロイメントストラテジーを選択します。しかし、EAR としてデプロイされた複数のモジュールで構成され、CDI のインジェクションやリソース参照エントリーの代わりにレガシーの JNDI ルックアップを使用する JBoss EAP 5 のアプリケーションは、設定の変更が必要になります。

ディレクトリー構造とスクリプト

前述のとおり、JBoss EAP 6 はプロファイルベースのデプロイメント設定を使用しません。そのため、**EAP_HOME/server/** ディレクトリーは存在しません。スタンドアロンサーバーの設定ファイルは **EAP_HOME/standalone/configuration/** ディレクトリー、デプロイメントは **EAP_HOME/standalone/deployments/** ディレクトリーにあります。管理対象ドメインで実行されているサーバーの設定ファイルは **EAP_HOME/domain/configuration/** ディレクトリーにあります。

JBoss EAP 5 では、Linux スクリプト **EAP_HOME/bin/run.sh** または Windows スクリプト

EAP_HOME/bin/run.bat を使用してサーバーを起動しました。JBoss EAP 6 では、サーバーの起動方法によってサーバー起動スクリプトが異なります。スタンドアロンサーバーを使用する場合は、Linux スクリプト **EAP_HOME/bin/standalone.sh** または Windows スクリプト **EAP_HOME/bin/standalone.bat** を使用します。管理対象ドメインを起動する場合は、Linux スクリプト **EAP_HOME/bin/domain.sh** または Windows スクリプト **EAP_HOME/bin/domain.bat** を使用します。

JNDI ルックアップ

JBoss EAP 6 では、標準化された移植可能な JNDI 名前空間を使用するようになりました。JBoss EAP 5 向けに書かれた JNDI ルックアップを使用するアプリケーションは、新しい JNDI 名前空間の慣習に従って変更する必要があります。JNDI のネーミング構文についての詳細は「[移植可能な EJB JNDI 名](#)」を参照してください。

詳細については、https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/にある JBoss EAP 6 向け『開発ガイド』の「JBoss EAP 6 の新機能と変更内容」を参照してください。

[バグを報告する](#)

2.3. 廃止および未サポート機能リストの確認

アプリケーションを移行する前に、以前のリリースの JBoss EAP では使用可能で、本リリースでは廃止またはサポート対象外になった機能があることに注意する必要があります。このような機能の完全リストは、カスタマーポータル

https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/にある JBoss EAP 6 向け『リリースノート』の「サポートされない機能」を参照してください。

[バグを報告する](#)

第3章 アプリケーションの移行

3.1. ほとんどのアプリケーションで必要な変更

3.1.1. ほとんどのアプリケーションで必要な変更の確認

JBoss EAP 6 のクラスローディングと設定の変更はほとんどのアプリケーションに影響します。また、JBoss EAP 6 は新しい標準の移植可能な JNDI ネーミング構文を使用します。これらの変更はほとんどのアプリケーションに影響するため、アプリケーションを移行する際、最初に以下の情報を確認することが推奨されます。

1. [「クラスローディングの変更によるアプリケーションの更新」](#)
2. [「設定変更によるアプリケーションの更新」](#)
3. [「アプリケーションの JNDI 名前空間名の更新」](#)

[バグを報告する](#)

3.1.2. クラスローディングの変更

3.1.2.1. クラスローディングの変更によるアプリケーションの更新

モジュラークラスローディングは JBoss EAP 6 での重大な変更の 1 つであり、ほぼすべてのアプリケーションが影響を受けます。アプリケーションを移行する際、最初に次の情報を確認してください。

1. 最初に、アプリケーションのパッケージと依存関係を確認します。詳細は、[「クラスローディングの変更によるアプリケーション依存関係の更新」](#)を参照してください。
2. アプリケーションがロギングを行う場合、正しいモジュールの依存関係を指定する必要があります。詳細は、[「ロギング依存関係の編集」](#)を参照してください。
3. モジュラークラスローディングの変更により、EAR または WAR のパッケージ構造を変更する必要がある場合があります。詳細は、[「EAR および WAR パッケージの編集」](#)を参照してください。

[バグを報告する](#)

3.1.2.2. モジュールの依存関係の理解

概要

モジュールは独自のクラスと、明示的または暗黙的な依存関係を持つモジュールのクラスのみにアクセスすることが可能です。

手順3.1 モジュールの依存関係の理解

1. 暗黙的な依存関係を理解する

サーバー内のデプロイヤーは、`javax.api` や `sun.jdk` などの一般的に使用されるモジュール依存関係を暗黙的かつ自動的に追加します。これにより、ランタイム時にデプロイメントに対してクラスを可視化でき、開発者が依存関係を明示的に追加する作業が軽減されます。暗黙的な依存関係がいつどのように追加されるかについて

は、https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/にある JBoss EAP 6 向け『開発ガイド』の章「クラスローディングとモジュール」に記載されている「暗黙的なモジュール依存関係」の説明を参照してください。

2. 明示的な依存関係を理解する

その他のクラスに対してはモジュールを明示的に指定する必要があります。明示的に指定しないと、不明な依存関係が原因でデプロイメントエラーやランタイムエラーが発生します。依存関係が不明な場合は、サーバーログに **ClassNotFoundException** トレースまたは **NoClassDefFoundErrors** トレースが記録されます。複数のモジュールが同じ JAR をロードしたり、異なるモジュールによってロードされるクラスを拡張するクラスを 1 つのモジュールがロードしたりする場合は、サーバーログに **ClassCastException** トレースが記録されます。依存関係を明示的に指定するには、**MANIFEST.MF** を変更するか、JBoss 固有のデプロイメント記述子ファイル **jboss-deployment-structure.xml** を作成します。モジュール依存関係の詳細

は、https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/にある JBoss EAP 6 向け『開発ガイド』の章「クラスローディングとモジュール」に記載されている「クラスローディングとモジュールの概要」を参照してください。

バグを報告する

3.1.2.3. クラスローディングの変更によるアプリケーション依存関係の更新

概要

JBoss EAP 6 のクラスローディングは、以前のバージョンの JBoss EAP とは大きく異なっています。JBoss EAP 6 のクラスローディングは、JBoss モジュールプロジェクトが基盤となっています。各ライブラリーは、すべての JAR をフラットなクラスパスにロードする単一の階層的なクラスローダーではなく、依存するモジュールに対してのみリンクするモジュールとなります。また、JBoss EAP 6 のデプロイメントもモジュールであり、クラスの依存関係が明示的に定義されていないと、アプリケーションサーバーの JAR に定義されているクラスへアクセスできません。アプリケーションサーバーによって定義されるモジュール依存関係の一部は自動的に設定されます。たとえば、Java EE アプリケーションをデプロイする場合、Java EE API の依存関係は自動的にまたは暗黙的に追加されます。サーバーにより自動的に追加される依存関係の完全な一覧について

は、https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/にある JBoss EAP 6 用『開発ガイド』の章「クラスローディングとモジュール」に記載されている「暗黙的なモジュール依存関係」の説明を参照してください。

タスク

モジュラークラスローディングの変更に伴い、アプリケーションを JBoss EAP 6 に移行する際に以下のタスクを 1 つ以上実行する必要がある場合があります。

- 「モジュールの依存関係の理解」
- 「[Tattletale](#) を用いたアプリケーション依存関係の検索」
- 「JBoss EAP 6 のクラスローディングを制御するファイルの作成または変更」
- 「新しいモジュラークラスローディングシステムのパッケージリソース」

バグを報告する

3.1.3. 設定ファイルの変更

3.1.3.1. JBoss EAP 6 のクラスローディングを制御するファイルの作成または変更

概要

モジュールクラスローディングを使用する JBoss EAP 6 の変更に伴い、依存関係を追加したり、自動的に依存関係がロードされないようにしたりするために、1 つ以上のファイルを作成または変更する必要があります。クラスローディングとクラスローディングの優先度については、https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/ にある JBoss EAP 6 向け『開発ガイド』の章「クラスローディングとモジュール」を参照してください。

JBoss EAP 6 のクラスローディングを制御するために使用されるファイルは次のとおりです。

jboss-web.xml

jboss-web.xml ファイルに **<class-loading>** 要素が定義されている場合はこれを削除する必要があります。JBoss EAP 5 でこの要素によって引き起こされた動作は、JBoss EAP 6 ではクラスローディングのデフォルト動作となったため、この要素を定義する必要がなくなりました。この要素を削除しないと、サーバーログに `ParseError` と `XMLStreamException` が記録されます。

これは、コメントアウトされた **jboss-web.xml** ファイルの **<class-loading>** 要素の例になります。

```
<!DOCTYPE jboss-web PUBLIC
    "-//JBoss//DTD Web Application 4.2//EN"
    "http://www.jboss.org/j2ee/dtd/jboss-web_4_2.dtd">
<jboss-web>
<!--
    <class-loading java2ClassLoadingCompliance="false">
        <loader-repository>
            seam.jboss.org:loader=MyApplication
        </loader-repository>
    </class-loading>
-->
</jboss-web>
```

MANIFEST.MF

手作業による編集

アプリケーションが使用するコンポーネントやモジュールによって異なりますが、このファイルに 1 つ以上の依存関係を追加する必要があります。依存関係は **Dependencies** エントリーまたは **Class-Path** エントリーとして追加できます。

開発者によって編集された **MANIFEST.MF** の例は次のとおりです。

```
Manifest-Version: 1.0
Dependencies: org.jboss.logmanager
Class-Path: OrderManagerEJB.jar
```

このファイルを編集する場合、必ずファイルの最後に newline 文字が含まれるようにしてください。

Maven を使用した生成

Maven を使用する場合、**pom.xml** ファイルを編集して **MANIFEST.MF** ファイルの依存関係を生成する必要があります。アプリケーションによって EJB 3.0 が使用される場合、**pom.xml** ファイルに次のようなセクションが含まれることがあります。

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-ejb-plugin</artifactId>
  <configuration>
    <ejbVersion>3.0</ejbVersion>
  </configuration>
</plugin>
```

EJB 3.0 コードが **org.apache.commons.log** を使用する場合、**MANIFEST.MF** ファイルにこの依存関係が存在しなければなりません。この依存関係を生成するには、次のように **<plugin>** 要素を **pom.xml** ファイルに追加します。

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-ejb-plugin</artifactId>
  <configuration>
    <ejbVersion>3.0</ejbVersion>
    <archive>
      <manifestFile>src/main/resources/META-INF/MANIFEST.MF</manifestFile>
    </archive>
  </configuration>
</plugin>
```

上記の例では、次の依存関係エントリのみが **src/main/resources/META-INF/MANIFEST.MF** ファイルに含まれる必要があります。

```
Dependencies: org.apache.commons.logging
```

Maven は完全な **MANIFEST.MF** ファイルを生成します。

```
Manifest-Version: 1.0
Dependencies: org.apache.commons.logging
```

jboss-deployment-structure.xml

このファイルは、クラスローディングを細かく制御するために使用される JBoss 固有のデプロイメント記述子です。**MANIFEST.MF** と同様に、このファイルを使用して依存関係を追加することが可能です。また、自動的な依存関係が追加されないようにしたり、追加のモジュールを定義することが可能で、EAR デプロイメントの分離されたクラスローディング動作を変更したり、追加のリソースルートをもジュールへ追加したりすることもできます。

JSF 1.2 モジュールの依存関係を追加し、JSF 2.0 モジュールが自動的にローディングされないようにする **jboss-deployment-structure.xml** ファイルの例は次のとおりです。

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-structure:1.0">
  <deployment>
    <dependencies>
      <module name="javax.faces.api" slot="1.2" export="true"/>
    </dependencies>
  </deployment>
</jboss-deployment-structure>
```

```

        <module name="com.sun.jsf-impl" slot="1.2" export="true"/>
    </dependencies>
</deployment>
<sub-deployment name="jboss-seam-booking.war">
    <exclusions>
        <module name="javax.faces.api" slot="main"/>
        <module name="com.sun.jsf-impl" slot="main"/>
    </exclusions>
    <dependencies>
        <module name="javax.faces.api" slot="1.2"/>
        <module name="com.sun.jsf-impl" slot="1.2"/>
    </dependencies>
</sub-deployment>
</jboss-deployment-structure>

```

このファイルに関する詳細は、[「jboss-deployment-structure.xml」](#) を参照してください。

application.xml

以前のバージョンの JBoss EAP では、**jboss-app.xml** ファイルを使用して EAR 内でデプロイメントの順番を制御しました。本バージョンより、Java EE6 仕様によって **application.xml** に **<initialize-in-order>** 要素が提供されるようになり、EAR 内の Java EE モジュールがデプロイされる順番は、この要素によって制御されるようになりました。

ほとんどの場合、デプロイメントの順番を指定する必要はありません。依存関係インジェクションと、外部モジュールのコンポーネントを参照する resource-refs がアプリケーションによって使用される場合、アプリケーションサーバーは適切で最適なコンポーネントの順番を暗黙的に決定できるため、ほとんどの場合で **<initialize-in-order>** 要素は必要ありません。

myApp.ear 内にパッケージ化された **myBeans.jar** および **myApp.war** が含まれるアプリケーションがあるとしましょう。**myApp.war** のサーブレットは **@EJB** アノテーションを使用して **myBeans.jar** から Bean をインジェクトします。この場合、必ずサーブレットが起動する前に EJB コンポーネントを使用できるようにし、**<initialize-in-order>** 要素を使用する必要がないことをアプリケーションサーバーが適切に認識します。

しかし、次のようなレガシーの JNDI ルックアップスタイルのリモート参照を使用し、Bean へアクセスする場合はモジュールの順番を指定する必要がある場合があります。

```

init() {
    Context ctx = new InitialContext();
    ctx.lookup("TheBeanInMyBeansModule");
}

```

この場合、EJB コンポーネントが **myBeans.jar** にあることをサーバーが判断できないため、**myBeans.jar** のコンポーネントが **myApp.war** のコンポーネントの前に初期化され開始されるよう強制する必要があります。これには、**<initialize-in-order>** 要素を **true** に設定し、**myBeans.jar** モジュールと **myApp.war** モジュールの順番を **application.xml** ファイルに指定します。

以下は **<initialize-in-order>** 要素を使用してデプロイメントの順番を制御する例になります。**myBeans.jar** は **myApp.war** ファイルの前にデプロイされます。

```

<application xmlns="http://java.sun.com/xml/ns/javaee"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              version="6"

```

```

        xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
        http://java.sun.com/xml/ns/javaee/application_6.xsd">
<application-name>myApp</application-name>
<initialize-in-order>true</initialize-in-order>
<module>
    <ejb>myBeans.jar</ejb>
</module>
<module>
    <web>
        <web-uri>myApp.war</web-uri>
        <context-root>myApp</context-root>
    </web>
</module>
</application>

```

application.xml ファイルのスキーマは http://java.sun.com/xml/ns/javaee/application_6.xsd を参照してください。



注記

<initialize-in-order> 要素を **true** に設定するとデプロイメントの速度が遅くなることに注意してください。デプロイメントを最適化するときにはコンテナの柔軟性が向上するため、依存関係インジェクションや **resource-refs** を使用して適切な依存関係を定義する方法が推奨されます。

jboss-ejb3.xml

Java Enterprise Edition (EE) が定義する **ejb-jar.xml** デプロイメント記述子によって提供される機能を上書きしたり追加したりするために、**jboss.xml** は **jboss-ejb3.xml** デプロイメント記述子に置き換えられました。この新しいファイルは **jboss.xml** との互換性がないため、**jboss.xml** はデプロイメントで無視されます。

login-config.xml

login-config.xml ファイルはセキュリティ設定では使用されなくなりました。セキュリティはサーバー設定ファイルの **<security-domain>** 要素に設定されるようになりました。スタンドアロンサーバーでは **standalone/configuration/standalone.xml** ファイルを使用します。管理対象ドメインでサーバーを実行している場合は **domain/configuration/domain.xml** ファイルを使用します。

[バグを報告する](#)

3.1.3.2. jboss-deployment-structure.xml

jboss-deployment-structure.xml は JBoss EAP 6 の新しいオプションデプロイメント記述子です。このデプロイメント記述子を使用すると、デプロイメントのクラスローディングを制御できます。

このデプロイメント記述子の XML スキーマは、**EAP_HOME/docs/schema/jboss-deployment-structure-1_2.xsd** にあります。

[バグを報告する](#)

3.1.3.3. 新しいモジュラークラスローディングシステムのパッケージリソース

概要

以前のバージョンの JBoss EAP では、**WEB-INF/**ディレクトリー内のすべてのリソースが WAR クラスパスに追加されました。JBoss EAP 6 では、Web アプリケーションのアーティファクトは **WEB-INF/classes** および **WEB-INF/lib** ディレクトリーからのみロードされます。指定の場所でアプリケーションアーティファクトのパッケージ化に失敗した場合、**ClassNotFoundException** や **NoClassDefError** などのランタイムエラーが発生します。

これらのクラスローディングエラーを解決するには、アプリケーションアーカイブの構造を編集するか、カスタムモジュールを定義する必要があります。

リソースパッケージの編集

アプリケーションのみがリソースを使用できるようにするには、プロパティーファイル、JAR、およびその他のアーティファクトを **WEB-INF/classes/** または **WEB-INF/lib/** ディレクトリーへ移動し、WAR とバンドルします。この方法の詳細は「[ResourceBundle プロパティーの場所変更](#)」を参照してください。

カスタムモジュールの作成

JBoss EAP サーバー上で実行されているすべてのアプリケーションが、カスタムリソースを使用できるようにするには、カスタムモジュールを作成する必要があります。この方法の詳細は、「[カスタムモジュールの作成](#)」を参照してください。

バグを報告する

3.1.3.4. ResourceBundle プロパティーの場所変更

概要

以前のバージョンの JBoss EAP では、**EAP_HOME/server/SERVER_NAME/conf/**ディレクトリーはクラスパスに存在し、アプリケーションによる使用が可能でした。このプロパティーを JBoss EAP 6 のアプリケーションのクラスパスで使用できるようにするには、アプリケーション内でパッケージ化する必要があります。

手順3.2 ResourceBundle プロパティーの場所変更

1. WAR アーカイブをデプロイする場合、これらのプロパティーを WAR の **WEB-INF/classes/** フォルダーでパッケージ化する必要があります。
2. これらのプロパティーを EAR のすべてのコンポーネントに対してアクセス可能にするには、JAR のルートでパッケージ化し、EAR の **lib/** フォルダーに置く必要があります。

バグを報告する

3.1.3.5. カスタムモジュールの作成

次の手順では、JBoss EAP サーバー上で実行されているすべてのアプリケーションがプロパティーファイルやその他のリソースを使用できるようにするために、カスタムモジュールを作成する方法について説明します。

手順3.3 カスタムモジュールの作成

1. **module/** ディレクトリー構造を作成し、ファイルを追加します。

- a. **EAP_HOME/module** ディレクトリ下にディレクトリ構造を作成し、ファイルや JAR が含まれるようにします。例は次のとおりです。

```
$ cd EAP_HOME/modules/  
$ mkdir -p myorg-conf/main/properties
```

- b. 作成した **EAP_HOME/modules/myorg-conf/main/properties/** ディレクトリにプロパティファイルを移動します。
- c. 次の XML が含まれる **module.xml** ファイルを **EAP_HOME/modules/myorg-conf/main/** ディレクトリに作成します。

```
<module xmlns="urn:jboss:module:1.1" name="myorg-conf">  
  <resources>  
    <resource-root path="properties"/>  
  </resources>  
</module>
```

2. サーバー設定ファイルの **ee** サブシステムを編集します。JBoss CLI を使用するか、手作業でファイルを編集します。

- o 次の手順に従って JBoss CLI を使用し、サーバー設定ファイルを編集します。

- a. サーバーを起動し、管理 CLI へ接続します。

- Linux の場合は、コマンドラインで以下を入力します。

```
$ EAP_HOME/bin/jboss-cli.sh --connect
```

- Windows の場合は、コマンドラインで以下を入力します。

```
C:\>EAP_HOME\bin\jboss-cli.bat --connect
```

次の応答が表示されるはずです。

```
Connected to standalone controller at localhost:9999
```

- b. **ee** サブシステムに **myorg-conf** <global-modules> 要素を作成するには、コマンドラインで以下を入力します。

```
/subsystem=ee:write-attribute(name=global-modules, value=  
[{"name"=>"myorg-conf", "slot"=>"main"}])
```

次の結果が表示されるはずです。

```
{"outcome" => "success"}
```

- o サーバー設定ファイルを手作業で編集する場合は、次の手順に従ってください。

- a. サーバーを停止し、テキストエディターでサーバー設定ファイルを開きます。スタンドアロンサーバーを実行している場合は、**EAP_HOME/standalone/configuration/standalone.xml** ファイルになり

ます。管理対象ドメインを実行している場合は、**EAP_HOME/domain/configuration/domain.xml** ファイルになります。

- b. **ee** サブシステムを見つけ、**myorg-conf** のグローバルモジュールを追加します。以下は、**myorg-conf** 要素が含まれるように編集された **ee** サブシステム要素の例になります。

```
<subsystem xmlns="urn:jboss:domain:ee:1.0" >
  <global-modules>
    <module name="myorg-conf" slot="main" />
  </global-modules>
</subsystem>
```

3. **my.properties** という名前のファイルを正しいモジュールの場所にコピーした場合は、以下のようなコードを使用してプロパティファイルをロードできるようになります。

```
Thread.currentThread().getContextClassLoader().getResource("my.properties");
```

[バグを報告する](#)

3.1.4. ロギングの変更

3.1.4.1. ロギング依存関係の編集

概要

JBoss LogManager はすべてのロギングフレームワークのフロントエンドをサポートするため、現在のロギングコードの保持または新しい JBoss ロギングインフラストラクチャーへの移行が可能です。モジュールクラスローディングが変更されたため、いずれの場合でもアプリケーションを変更して必要な依存関係を追加する必要があるでしょう。

手順3.4 アプリケーションロギングコードの更新

1. 「サードパーティーロギングフレームワークのアプリケーションコードの更新」
2. 「新しい JBoss ロギングフレームワークを使用したコードの変更」

[バグを報告する](#)

3.1.4.2. サードパーティーロギングフレームワークのアプリケーションコードの更新

概要

JBoss EAP 6 では、Apache Commons Logging、Apache log4j、SLF4J、Java Logging などの一般的なサードパーティーフレームワークのロギング依存関係はデフォルトで追加されています。ほとんどの場合で、JBoss EAP コンテナによって提供されるロギングフレームワークを使用することが推奨されますが、サードパーティーのフレームワークによって提供される機能が必要な場合は、デプロイメントから対応する JBoss EAP モジュールを除外する必要があります。デプロイメントがサードパーティーのロギングフレームワークを使用する場合でも、サーバーログは継続して JBoss EAP ロギングサブシステムの設定を使用することに注意してください。

以下の手順は、JBoss EAP 6 **org.apache.log4j** モジュールをデプロイメントから除外する方法を示しています。最初の手順は、JBoss EAP 6 のすべてのリリースで動作します。2 つ目の手順は、JBoss EAP 6.3 およびそれ以降のリリースのみで使用できます。

手順3.5 log4j.properties または log4j.xml ファイルを使用するよう JBoss EAP 6 を設定する

この手順は、JBoss EAP 6 の全バージョンで動作します。



注記

この手順では log4j 設定ファイルが使用されるため、起動時に log4j ロギング設定を変更できなくなります。

1. 次の内容が含まれる **jboss-deployment-structure.xml** を作成します。

```
<jboss-deployment-structure>
  <deployment>
    <!-- Exclusions allow you to prevent the server from
    automatically adding some dependencies -->
    <exclusions>
      <module name="org.apache.log4j" />
    </exclusions>
  </deployment>
</jboss-deployment-structure>
```

2. **jboss-deployment-structure.xml** ファイルは、WAR をデプロイする場合は **META-INF/** または **WEB-INF/** ディレクトリーに置き、EAR をデプロイする場合は **META-INF/** ディレクトリーに置きます。デプロイメントに依存する子デプロイメントが含まれる場合は、各サブデプロイメントのモジュールも除外する必要があります。
3. **log4j.properties** または **log4j.xml** ファイルが、EAR の **lib/** ディレクトリーまたは WAR デプロイメントの **WEB-INF/classes/** ディレクトリーに含まれるようにします。このファイルを WAR の **lib/** ディレクトリーに置きたい場合は、**jboss-deployment-structure.xml** ファイルに **<resource-root>** パスを指定する必要があります。

```
<jboss-deployment-structure>
  <deployment>
    <!-- Exclusions allow you to prevent the server from
    automatically adding some dependencies -->
    <exclusions>
      <module name="org.apache.log4j" />
    </exclusions>
    <resources>
      <resource-root path="lib" />
    </resources>
  </deployment>
</jboss-deployment-structure>
```

4. 以下の引数を用いて JBoss EAP 6 サーバーを起動し、アプリケーションをデプロイするときに **ClassCastException** がコンソールに表示されないようにします。

```
-Dorg.jboss.as.logging.per-deployment=false
```

5. アプリケーションをデプロイします。

手順3.6 JBoss EAP 6.3 またはそれ以降のバージョンでのロギング依存関係の設定

JBoss EAP 6.3 およびそれ以降のバージョンでは、新しい **add-logging-api-dependencies** ログインシステム属性を使用してサードパーティーロギングフレームワークの依存関係を除外できます。以下の手順は、JBoss EAP スタンドアロンサーバーでこのロギング属性を変更する方法を示しています。

1. 以下の引数を用いて JBoss EAP 6 サーバーを起動し、アプリケーションをデプロイするときに **ClassCastException** がコンソールに表示されないようにします。

```
-Dorg.jboss.as.logging.per-deployment=false
```

2. ターミナルを開き、管理 CLI へ接続します。

- Linux の場合は、コマンドラインで以下を入力します。

```
$ EAP_HOME/bin/jboss-cli.sh --connect
```

- Windows の場合は、コマンドラインで以下を入力します。

```
C:\>EAP_HOME\bin\jboss-cli.bat --connect
```

3. ロギングサブシステムの **add-logging-api-dependencies** 属性を編集します。

この属性は、コンテナが暗黙的なロギング API の依存関係を追加するかどうかを制御します。

- デフォルトの **true** に設定すると、暗黙的なロギング API の依存関係がすべて追加されます。
- **false** に設定すると、依存関係はデプロイメントへ追加されません。

サードパーティーロギングフレームワークの依存関係を除外するには、以下のコマンドを使用してこの属性を **false** に設定する必要があります。

```
/subsystem=logging:write-attribute(name=add-logging-api-dependencies, value=false)
```

このコマンドは、**<add-logging-api-dependencies>** 要素を **standalone.xml** 設定ファイルの **logging** サブシステムに追加します。

```
<subsystem xmlns="urn:jboss:domain:logging:1.4">
  <add-logging-api-dependencies value="false"/>
  ....
</subsystem>
```

4. アプリケーションをデプロイします。

[バグを報告する](#)

3.1.4.3. 新しい JBoss ロギングフレームワークを使用したコードの変更

概要

新しいフレームワークを使用するには、次のようにインポートとコードを変更します。

手順3.7 JBoss ロギングフレームワークを使用するようコードおよび依存関係を変更する

1. インポートとロギングコードの変更

新しい JBoss ロギングフレームワークを使用するコードの例は以下のとおりです。

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

private static final Logger logger =
    Logger.getLogger(MyClass.class.toString());

if(logger.isTraceEnabled()) {
    logger.tracef("Starting...", subsystem);
}
```

2. ロギング依存関係の追加

JBoss ロギングクラスが含まれる JAR は **org.jboss.logging** という名前のモジュールにあります。**MANIFEST-MF** ファイルは次のようになるはずです。

```
Manifest-Version: 1.0
Dependencies: org.jboss.logging
```

モジュール依存関係の検索方法に関する詳細については、「[クラスローディングの変更によるアプリケーション依存関係の更新](#)」と「[移行の問題のデバッグと解決](#)」を参照してください。

[バグを報告する](#)

3.1.5. アプリケーションパッケージの変更

3.1.5.1. EAR および WAR パッケージの編集

概要

アプリケーションを移行する際、モジュラークラスローディングの変更に伴い、EAR または WAR のパッケージ構造を変更する必要がある場合があります。モジュール依存関係は次の順序でロードされます。

1. システム依存関係
2. ユーザー依存関係
3. ローカルリソース
4. デプロイメント間の依存性

手順3.8 アーカイブパッケージの編集

1. WAR のパッケージ化

WAR は単一のモジュールで、WAR のすべてのクラスは同じクラスローダーでロードされます。そのため **WEB-INF/lib/** ディレクトリーにパッケージされるクラスは、**WEB-INF/classes** ディレクトリーのクラスと同様に処理されます。

2. EAR のパッケージ化

EAR は複数のモジュールによって構成されます。**EAR/lib/** ディレクトリーは単一のモジュールで、EAR 内の各 WAR や EJB jar サブデプロイメントは個別のモジュールになります。依存関係が明示的に定義された場合を除き、クラスは EAR 内の他のモジュールにあるクラスヘアク

セスできません。サブデプロイメントは常に親モジュール上で自動的な依存関係を持ち、親モジュールは **EAR/lib/** ディレクトリーのクラスへのアクセスを許可します。しかし、サブデプロイメントはお互いのアクセスを許可するため常に自動的な依存関係を持っているわけではありません。この挙動は次のように **ee** サブシステム設定の **<ear-subdeployments-isolated>** 要素を設定すると制御できます。

```
<subsystem xmlns="urn:jboss:domain:ee:1.0" >
  <ear-subdeployments-isolated>false</ear-subdeployments-isolated>
</subsystem>
```

デフォルトでは **false** に設定され、EAR 内の他のサブデプロイメントに属するクラスがサブデプロイメントに対して可視化されます。

クラスローディングの詳細について

は、https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/ にある JBoss EAP 6 向け『開発ガイド』の章「クラスローディングとモジュール」を参照してください。

[バグを報告する](#)

3.1.6. データソースおよびリソースアダプター設定の変更

3.1.6.1. 設定変更によるアプリケーションの更新

JBoss EAP 5 では、サービスやサブシステムが多くのファイルに設定されていました。JBoss EAP 6 では、設定は主に 1 つのファイルで行われます。アプリケーションが以下のサービスやリソースを使用する場合は、設定の変更が必要になることがあります。

1. アプリケーションがデータソースを使用する場合は、[「DataSource 設定の更新」](#)を参照してください。
2. アプリケーションが JPA を使用し、現在 Hibernate JAR をバンドルする場合は、[「Hibernate または JPA 用のデータソースの設定」](#)を参照し、移行のオプションについて確認してください。
3. アプリケーションがリソースアダプターを使用する場合は、[「リソースアダプター設定の更新」](#)を参照してください。
4. [「アプリケーションセキュリティの変更設定」](#)を参照し、基本的なセキュリティの設定変更方法について確認してください。

[バグを報告する](#)

3.1.6.2. DataSource 設定の更新

概要

以前のバージョンの JBoss EAP では、ファイル名の最後に ***-ds.xml** が付くファイルに JCA データソースの設定が定義されていました。このファイルはサーバーの **deploy/** ディレクトリーにデプロイされるか、アプリケーションによってパッケージ化されました。JDBC ドライバーは **server/lib/** ディレクトリーにコピーされるか、アプリケーションの **WEB-INF/lib/** ディレクトリーにパッケージ化されました。この設定方法は開発環境では今でもサポートされていますが、JBoss の管理ツールではサポートされていないため実稼働環境では推奨されません。

JBoss EAP 6 では、データソースはサーバー設定ファイルに設定されています。JBoss EAP 6 インスタ

ンスが管理対象ドメインで実行されている場合、データソースは **domain/configuration/domain.xml** ファイルに設定されます。JBoss EAP 6 インスタンスがスタンドアロンサーバーとして実行されている場合、データソースは **standalone/configuration/standalone.xml** ファイルに設定されます。このように設定されたデータソースは、Web 管理コンソールやコマンドラインインターフェース (CLI) などが含まれる JBoss 管理インターフェースを使用して管理および制御されます。これらのツールはデプロイメントの管理や、管理対象ドメインで実行されている複数のサーバーの設定を容易にします。

次の項では、使用可能な管理ツールによって管理およびサポートされるよう、データソースの設定を変更する方法について説明します。

JBoss EAP 6 の管理可能なデータソース設定の移行

JDBC 4.0 対応のドライバーはデプロイメントまたはコアモジュールとしてインストールすることができます。JDBC 4.0 対応のドライバーには、ドライバークラス名を指定する **META-INF/services/java.sql.Driver** ファイルが含まれています。JDBC 4.0 対応でないドライバーには追加の設定が必要となります。ドライバーを JDBC 4.0 対応にする方法や、現在のデータソース設定を Web 管理コンソールや CLI によって管理可能な設定に更新する方法の詳細については、[「JDBC ドライバーのインストールと設定」](#)を参照してください。

アプリケーションが Hibernate や JPA を使用する場合、追加の変更が必要となる場合があります。詳細については、[「Hibernate または JPA 用のデータソースの設定」](#)を参照してください。

IronJacamar 移行ツールを使用した設定データの変換

IronJacamar ツールを使用するとデータソースおよびリソースアダプターの設定を移行できます。このツールは、***-ds.xml** 形式の設定ファイルを JBoss EAP 6 が想定する形式に変換します。詳細は、[「IronJacamar ツールを使用してデータソースとリソースアダプターの設定を移行する」](#)を参照してください。

[バグを報告する](#)

3.1.6.3. JDBC ドライバーのインストールと設定

概要

次の 2 つの方法の 1 つを用いて JDBC ドライバーをコンテナにインストールできます。

- デプロイメントとしてのインストール
- コアモジュールとしてのインストール

これらの方法の利点と欠点は次のとおりです。

JBoss EAP 6 では、データソースはサーバー設定ファイルに設定されています。JBoss EAP 6 インスタンスが管理対象ドメインで実行されている場合、データソースは **domain/configuration/domain.xml** ファイルに設定されます。JBoss EAP 6 インスタンスがスタンドアロンサーバーとして実行されている場合、データソースは **standalone/configuration/standalone.xml** ファイルに設定されます。両モードで共通のスキーマ参照情報は JBoss EAP 6 インストールの **doc/schema/** ディレクトリーにあります。ここでは説明上、サーバーがスタンドアロンサーバーとして稼働し、データソースが **standalone.xml** ファイルに設定されていると仮定します。

手順3.9 JDBC ドライバーのインストールと設定

1. JDBC ドライバーをインストールします。

a. **JDBC ドライバーをデプロイメントとしてインストールします。**

これはドライバーのインストールに推奨される方法です。JDBC ドライバーがデプロイメントとしてインストールされると、普通の JAR としてデプロイされます。JBoss EAP 6 インスタンスがスタンドアロンサーバーとして実行されている場合は、JDBC 4.0 対応の JAR を **EAP_HOME/standalone/deployments/** ディレクトリーへコピーします。管理対象ドメインの場合は、管理コンソールまたは管理 CLI を使用して JAR をサーバーグループにデプロイする必要があります。

スタンドアロンサーバーにデプロイメントとしてインストールされた MySQL JDBC ドライバーの例は次のとおりです。

```
$cp mysql-connector-java-5.1.15.jar
EAP_HOME/standalone/deployments/
```

JDBC 4.0 対応のドライバーは自動的に認識され、名前とバージョンによってシステムヘインストールされます。JDBC 4.0 対応の JAR にはドライバーのクラス名を指定する **META-INF/services/java.sql.Driver** という名前のテキストファイルが含まれてます。ドライバーが 4.0 対応でない場合は、次の方法の 1 つを用いてデプロイ可能にすることができます。

- **java.sql.Driver** ファイルを作成し、**META-INF/services/** パス下の JAR に追加します。このファイルには、次のようなドライバークラス名が含まれていなければなりません。

```
com.mysql.jdbc.Driver
```

- **java.sql.Driver** ファイルをデプロイメントディレクトリーに作成します。スタンドアロンサーバーとして実行されている JBoss EAP 6 のインスタンスの場合、このファイルを **EAP_HOME/standalone/deployments/META-INF/services/java.sql.Driver** に置く必要があります。サーバーが管理ドメインで実行されている場合は、管理コンソールまたは管理 CLI を使用してファイルをデプロイする必要があります。

この方法の利点は次のとおりです。

- モジュールを定義する必要がないため、これが最も簡単な方法です。
- サーバーが管理対象ドメインで実行されている場合は、この方法を使用するデプロイメントがドメイン内のすべてのサーバーに自動的に伝播されます。つまり、管理者はドライバー JAR を手動で配布する必要がありません。

この方法の欠点は次のとおりです。

- JDBC ドライバーが複数の JAR (たとえば、ドライバー JAR および依存ライセンス JAR またはローカリゼーション JAR) から構成される場合、ドライバーをデプロイメントとしてインストールすることはできません。JDBC ドライバーは、コアモジュールとしてインストールする必要があります。
- ドライバーが JDBC 4.0 対応でない場合は、ドライバークラス名を含むファイルを作成し、JAR にインポートするか、**deployments/** ディレクトリーにオーバーレイする必要があります。

b. **JDBC ドライバーをコアモジュールとしてインストールします。**

JDBC ドライバーをコアモジュールとしてインストールするには、**EAP_HOME/modules/** ディレクトリー下にファイルパス構造を作成する必要があります。この構造には、JDBC ドライバー JAR、任意の追加ベンダーライセンスまたはローカリゼーション JAR、および

モジュールを定義する **module.xml** ファイルが含まれます。

■ **MySQL JDBC ドライバーをコアモジュールとしてインストールします。**

- i. ディレクトリー構造 **EAP_HOME/modules/com/mysql/main/** を作成します。
- ii. **main/** サブディレクトリーで、MySQL JDBC ドライバーに対する以下のモジュール定義を含む **module.xml** ファイルを作成します。

```
<?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"/>
  </dependencies>
</module>
```

モジュール名 "com.mysql" はこのモジュールのディレクトリー構造と一致します。**<dependencies>** 要素は、このモジュールの他のモジュールへの依存関係を指定するために使用されます。この場合、全 JDBC データソースの場合と同様に、**javax.api** という名前の他のモジュールによって定義される Java JDBC API に依存します。このモジュールは **modules/system/layers/base/javax/api/main/** ディレクトリーに存在します。



注記

module.xml ファイルの最初に空白文字が存在しないようにしてください。空白文字が存在すると、このドライバーに対して「New missing/unsatisfied dependencies」エラーが発生します。

- iii. MySQL JDBC ドライバー JAR を **EAP_HOME/modules/com/mysql/main/** ディレクトリーへコピーします。

```
$ cp mysql-connector-java-5.1.15.jar
EAP_HOME/modules/com/mysql/main/
```

■ **IBM DB2 JDBC ドライバーとライセンス JAR をコアモジュールとしてインストールします。**

この例は、JDBC ドライバー JAR 以外に JAR が必要なドライバーをデプロイする方法を示すためにのみ提供されています。

- i. ディレクトリー構造 **EAP_HOME/modules/com/ibm/db2/main/** を作成します。
- ii. **main/** サブディレクトリーで、IBM DB2 JDBC ドライバーとライセンスに対する以下のモジュール定義を含む **module.xml** ファイルを作成します。

```
<?xml version="1.0" encoding="UTF-8"?>
<module xmlns="urn:jboss:module:1.1" name="com.ibm.db2">
  <resources>
    <resource-root path="db2jcc.jar"/>
  </resources>
</module>
```

```

    <resource-root path="db2jcc_license_cisuz.jar"/>
  </resources>
  <dependencies>
    <module name="javax.api"/>
    <module name="javax.transaction.api"/>
  </dependencies>
</module>

```



注記

module.xml ファイルの最初に空白文字が存在しないようにしてください。空白文字が存在すると、このドライバーに対して「New missing/unsatisfied dependencies」エラーが発生します。

iii. JDBC ドライバーおよびライセンス JAR を

EAP_HOME/modules/com/ibm/db2/main/ ディレクトリーにコピーします。

```

$ cp db2jcc.jar EAP_HOME/modules/com/ibm/db2/main/
$ cp db2jcc_license_cisuz.jar
  EAP_HOME/modules/com/ibm/db2/main/

```

この方法の利点は次のとおりです。

- これは、JDBC ドライバーが複数の JAR から構成される場合に有効な唯一の方法です。
- この方法では、JDBC 4.0 対応でないドライバーは、ドライバー JAR を変更したり、ファイルオーバーレイを作成したりせずにインストールできます。

この方法の欠点は次のとおりです。

- モジュールのセットアップはより困難になります。
- モジュールは、管理対象ドメインで実行されている各サーバーに手動でコピーする必要があります。

2. データソースを設定します。

a. データソースドライバーを追加します。

<driver> 要素を同じファイルの **<drivers>** 要素に追加します。以前 ***-ds.xml** ファイルに定義されたデータソース情報と同じ情報が一部含まれます。

最初にドライバー JAR が JDBC 4.0 対応であるか判断します。JDBC 4.0 対応の JAR にはドライバークラス名を指定する **META-INF/services/java.sql.Driver** ファイルが含まれています。サーバーはこのファイルを使用して JAR のドライバークラス名を探します。JDBC 4.0 対応のドライバーの JAR には既に **<driver-class>** 要素が指定されているため、この要素は必要ありません。JDBC 4.0 対応 MySQL ドライバーのドライバー要素の例は次のとおりです。

```

<driver name="mysql-connector-java-5.1.15.jar"
module="com.mysql"/>

```

JDBC 4.0 対応でないドライバーには、ドライバークラスを指定する **<driver-class>** 属性が必要です。これは、ドライバークラス名を指定する **META-INF/services/java.sql.Driver** ファイルが存在しないためです。JDBC 4.0 対応でな

いドライバーのドライバー要素の例は、次のとおりです。

```
<driver name="mysql-connector-java-5.1.15.jar"
module="com.mysql">
<driver-class>com.mysql.jdbc.Driver</driver-class></driver>
```

b. データソースを作成します。

standalone.xml ファイルの **<datasources>** セクションに **<datasource>** 要素を作成します。このファイルには、以前に ***-ds.xml** ファイルに定義されたデータソース情報とほとんど同じ情報が含まれています。



重要

サーバーの再起動後も変更が維持されるようにするには、サーバーを停止してからサーバー設定ファイルを編集する必要があります。

standalone.xml ファイルの MySQL データソース要素の例は次のとおりです。

```
<datasource jndi-name="java:/YourDatasourceName" pool-
name="YourDatasourceName">
  <connection-
url>jdbc:mysql://localhost:3306/YourApplicationURL</connection-
url>
  <driver>mysql-connector-java-5.1.15.jar</driver>
  <transaction-isolation>TRANSACTION_READ_COMMITTED</transaction-
isolation>
  <pool>
    <min-pool-size>100</min-pool-size>
    <max-pool-size>200</max-pool-size>
  </pool>
  <security>
    <user-name>USERID</user-name>
    <password>PASSWORD</password>
  </security>
  <statement>
    <prepared-statement-cache-size>100</prepared-statement-cache-
size>
    <share-prepared-statements/>
  </statement>
</datasource>
```

3. アプリケーションコードで JNDI 参照を更新します。

定義した新しい JNDI 標準データソース名を使用するには、アプリケーションソースコードの古い JNDI ルックアップ名を置き換える必要があります。詳細は、[「JNDI 名前空間の新ルールに準拠するようアプリケーションを変更」](#)を参照してください。

新しい JNDI 名を使用するには、データソースにアクセスする既存の **@Resource** アノテーションも置き換える必要があります。例は次のとおりです。

```
@Resource(name = "java:/YourDatasourceName").
```

[バグを報告する](#)

3.1.6.4. Hibernate または JPA 用のデータソースの設定

アプリケーションによって JPA が使用され、Hibernate JAR がバンドルされる場合、JBoss EAP 6 に含まれる Hibernate の使用が推奨されます。このバージョンの Hibernate を使用するには、アプリケーションから古いバージョンの Hibernate バンドルを削除する必要があります。

手順3.10 Hibernate バンドルの削除

1. アプリケーションライブラリーフォルダーより Hibernate JAR を削除します。
2. `persistence.xml` ファイルの `<hibernate.transaction.manager_lookup_class>` 要素は必要がないため、削除またはコメントアウトします。

[バグを報告する](#)

3.1.6.5. リソースアダプター設定の更新

概要

以前のバージョンのアプリケーションサーバーでは、リソースアダプター設定は、ファイル名の最後に `*-ds.xml` が付くファイルで定義されました。JBoss EAP 6 では、リソースアダプターはサーバー設定ファイルで設定されます。管理対象ドメインで実行されている場合、設定ファイルは `EAP_HOME/domain/configuration/domain.xml` ファイルになります。スタンドアロンサーバーとして実行されている場合、`EAP_HOME/standalone/configuration/standalone.xml` ファイルのリソースアダプターを設定します。両モードで共通であるスキーマ参照の情報は、IronJacamar の Web サイト <http://www.ironjacamar.org/documentation.html> の「Schemas」にあります。



重要

サーバーの再起動後に変更が維持されるようにするには、サーバーを停止してからサーバー設定ファイルを編集する必要があります。

リソースアダプターの定義

リソースアダプター記述子の情報は、サーバー設定ファイルの次のサブシステム要素下に定義されます。

```
<subsystem xmlns="urn:jboss:domain:resource-adapters:1.1"/>
```

以前にリソースアダプター `*-ds.xml` ファイルに定義された情報と同じものの一部を使用します。

以下に、サーバー設定ファイルのリソースアダプター要素の例を示します。

```
<resource-adapters>
  <resource-adapter>
    <archive>multiple-full.rar</archive>
    <config-property name="Name">ResourceAdapterValue</config-property>
    <transaction-support>NoTransaction</transaction-support>
    <connection-definitions>
      <connection-definition
        class-
name="org.jboss.jca.test.deployers.spec.rars.multiple.MultipleManagedConnectionFactory1"
        enabled="true" jndi-name="java:/eis/MultipleConnectionFactory1"
        pool-name="MultipleConnectionFactory1">
```

```

    <config-property name="Name">MultipleConnectionFactory1Value</config-
property>
    </connection-definition>
    <connection-definition
    class-
name="org.jboss.jca.test.deployers.spec.rars.multiple.MultipleManagedConne
ctionFactory2"
    enabled="true" jndi-name="java:/eis/MultipleConnectionFactory2"
    pool-name="MultipleConnectionFactory2">
    <config-property name="Name">MultipleConnectionFactory2Value</config-
property>
    </connection-definition>
    </connection-definitions>
    <admin-objects>
    <admin-object
    class-
name="org.jboss.jca.test.deployers.spec.rars.multiple.MultipleAdminObject1
Impl"
    jndi-name="java:/eis/MultipleAdminObject1">
    <config-property name="Name">MultipleAdminObject1Value</config-
property>
    </admin-object>
    <admin-object class-
name="org.jboss.jca.test.deployers.spec.rars.multiple.MultipleAdminObject2
Impl"
    jndi-name="java:/eis/MultipleAdminObject2">
    <config-property name="Name">MultipleAdminObject2Value</config-
property>
    </admin-object>
    </admin-objects>
    </resource-adapter>
</resource-adapters>

```

[バグを報告する](#)

3.1.7. セキュリティーの変更

3.1.7.1. アプリケーションセキュリティの変更設定

基本認証のセキュリティの設定

以前のバージョンの JBoss EAP では、**EAP_HOME/server/SERVER_NAME/conf/** ディレクトリーに置かれたプロパティーファイルはクラスパス上にあり、**UsersRolesLoginModule** によって簡単に見つけられました。JBoss EAP 6 ではディレクトリー構造が変更されたため、プロパティーファイルをアプリケーション内でパッケージ化し、クラスパスで使えるようにする必要があります。



重要

サーバーの再起動後に変更が維持されるようにするには、サーバーを停止してからサーバー設定ファイルを編集する必要があります。

基本認証のセキュリティを設定するには、**security-domains** 下の新しいセキュリティドメインを **standalone/configuration/standalone.xml** または **domain/configuration/domain.xml** サーバー設定ファイルに追加します。

```
<security-domain name="example">
  <authentication>
    <login-module code="UsersRoles" flag="required">
      <module-option name="usersProperties"
        value="${jboss.server.config.dir}/example-
users.properties"/>
      <module-option name="rolesProperties"
        value="${jboss.server.config.dir}/example-
roles.properties"/>
    </login-module>
  </authentication>
</security-domain>
```

JBoss EAP 6 インスタンスがスタンドアロンサーバーとして実行されている場合、`${jboss.server.config.dir}` は `EAP_HOME/standalone/configuration/` ディレクトリーを参照します。インスタンスが管理対象ドメインで実行されている場合、`${jboss.server.config.dir}` は `EAP_HOME/domain/configuration/` ディレクトリーを参照します。

セキュリティドメイン名の変更

JBoss EAP 6 では、セキュリティドメインの名前に接頭辞 `java:/jaas/` を使用しません。

- Web アプリケーションの場合は、`jboss-web.xml` のセキュリティドメイン設定からこの接頭辞を削除する必要があります。
- エンタープライズアプリケーションの場合は、`jboss-ejb3.xml` ファイルのセキュリティドメイン設定からこの接頭辞を削除する必要があります。JBoss EAP 6 では、`jboss.xml` はこのファイルに置き換えられました。

バグを報告する

3.1.7.2. PicketLink STS および Web サービスを使用するアプリケーションの更新

概要

JBoss EAP 6.1 のアプリケーションが PicketLink STS および Web サービスを使用する場合、JBoss EAP 6.2 またはそれ以降のバージョンへ移行するときに変更を加える必要があることがあります。CVE-2013-2133 に対処するために JBoss EAP に適用された修正により、EJB3 ベースの WS エンドポイントにアタッチする JAXWS ハンドラーを実行する前にコンテナによる承認チェックが実行されます。この結果、プロセスの後半で使用されるはずのセキュリティープリンシパルが PicketLink **SAML2Handler** によって確立されるため、PicketLink STS の機能が影響を受けます。**HandlerAuthInterceptor** が **SAML2Handler** にアクセスするときはプリンシパルが **NULL** であるため、サーバーログに **NullPointerException** が記録されることがあります。この問題を修正するには、このセキュリティーチェックを無効にする必要があります。

手順3.11 他の承認チェックの無効化

- 他の承認チェックを無効にし、既存の PicketLink デプロイメントを継続して使用するには、以下の方法の 1 つを使用します。
 - システム全体のプロパティーの設定
サーバーレベルで他の承認チェックを無効にするには、`org.jboss.ws.cxf.disableHandlerAuthChecks` システムプロパティーの値を `true` に設定します。この方法は、アプリケーションサーバーに対して作成されたすべての

デプロイメントに影響します。

システムプロパティの設定方法は、JBoss EAP 『管理および設定ガイド』の「管理 CLI を使用したシステムプロパティの設定」を参照してください。

○ デプロイメントの Web サービス記述子ファイルでのプロパティの作成

デプロイメントレベルで他の承認チェックを無効にするには、**jboss-webservices.xml** ファイルで **org.jboss.ws.cxf.disableHandlerAuthChecks** プロパティの値を **true** に設定します。この方法は、特定のデプロイメントのみに影響します。

- a. 他の承認チェックを無効にしたいデプロイメントの **META-INF/** ディレクトリで **jboss-webservices.xml** ファイルを作成します。
- b. 以下の内容を追加します。

```
<?xml version="1.1" encoding="UTF-8"?>
<webservices xmlns="http://www.jboss.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  version="1.2"
  xsi:schemaLocation="http://www.jboss.com/xml/ns/javaee">
  <property>
    <name>org.jboss.ws.cxf.disableHandlerAuthChecks</name>
    <value>true</value>
  </property>
</webservices>
```

注記

[CVE-2013-2133](#) に対して脆弱性を持つシステムをレンダリングする

org.jboss.ws.cxf.disableHandlerAuthChecks プロパティを有効にします。

EJB メソッド上で宣言された制限が適用されることをアプリケーションが想定し、JAX-WS ハンドラーに独立して制限を適用しない場合、プロパティを有効にしないでください。アプリケーションを破壊しないようにするため、後方互換性を維持するのに必要な場合のみこのプロパティを使用してください。

[バグを報告する](#)

3.1.8. JNDI の変更

3.1.8.1. アプリケーションの JNDI 名前空間名の更新

概要

EJB 3.1 には、標準化されたグローバル JNDI 名前空間や、Java EE アプリケーションのさまざまなスコープへマップする名前空間が導入されました。移植可能な EJB 名は、**java:global**、**java:module**、**java:app** の 3 つのみへバインドされます。EJB を用いるアプリケーションが JNDI を使用する場合、そのアプリケーションを変更し、新しい標準 JNDI 名前空間の慣例に従うようにする必要があります。

手順3.12 JNDI ルックアップの変更

1. 「移植可能な EJB JNDI 名」を確認する
2. 「JNDI 名前空間のルールの確認」

3. 「JNDI 名前空間の新ルールに準拠するようアプリケーションを変更」

JNDI マッピングの例

以前のリリースでの JNDI 名前空間の例や、JBoss EAP 6 で指定する方法については、「[以前のリリースでの JNDI 名前空間の例、および JBoss EAP 6 での名前空間の指定方法](#)」を参照してください。

[バグを報告する](#)

3.1.8.2. 移植可能な EJB JNDI 名

概要

Java EE 6 仕様には、独自のスコープを持つ 4 つの論理的な名前空間が定義されていますが、移植可能な EJB 名はそのうちの 3 つの名前空間へのみバインドされます。下表は、各名前空間の使用法と使用時の詳細を表しています。

表3.1 移植可能な JNDI 名前空間

JNDI 名前空間	説明
java:global	<p>この名前空間の名前は、アプリケーションサーバーインスタンスにデプロイされたすべてのアプリケーションで共有されます。同じサーバーヘデプロイされた EJB の外部アーカイブを検索するには、この名前空間の名前を使用します。</p> <p>Java:global 名前空間の例は、java:global/jboss-seam-bookings/jboss-seam-bookings-jar/HotelBookingAction です。</p>
java:module	<p>この名前空間の名前は、1 つの EJB モジュールにある全エンタープライズ Bean や Web モジュールにある全コンポーネントなど、モジュール内の全コンポーネントで共有されます。</p> <p>java:module 名前空間の例は、java:module/HotelBookingAction!org.jboss.seam.example.bookings.HotelBooking です。</p>
java:app	<p>この名前空間の名前は、1 つのアプリケーション内にある全モジュールのコンポーネントすべてで共有されます。例えば、同じ EAR ファイルにある WAR や EJB jar ファイルは java:app 名前空間のリソースにアクセスできます。</p> <p>java:app 名前空間の例は、java:app/jboss-seam-bookings-jar/HotelBookingAction です。</p>

JNDI ネーミングコンテキストの詳細は「JSR 316: Java Platform, Enterprise Edition 6 (Java EE 6) Specification」の EE.5.2.2 「Application Component Environment Namespaces」を参照してください。<http://jcp.org/en/jsr/detail?id=316> からこの仕様をダウンロードできます。

[バグを報告する](#)

3.1.8.3. JNDI 名前空間のルールの確認

概要

JBoss EAP 6 では JNDI 名前空間の名前が改良され、アプリケーションにバインドされた名前に対して予測可能で一貫性のあるルールを提供するだけでなく、将来的に互換性の問題が起こらないよう対処されます。そのため、現在の名前空間が新ルールに準拠しない場合、問題が発生することがあります。

名前空間は次のルールに準拠する必要があります。

1. **DefaultDS** や **jdbc/DefaultDS** などの不適切な相対名は、コンテキストに応じて **java:comp/env**、**java:module/env**、または **java:jboss/env** に対して相対的に修飾する必要があります。
2. **/jdbc/DefaultDS** などの不適切な **absolute** 名は、**java:jboss/root** 名に対して相対的に修飾する必要があります。
3. **java:/jdbc/DefaultDS** などの適切な **absolute** 名は、上記の不適切な **absolute** 名と同様に修飾する必要があります。
4. 特別な **java:jboss** 名前空間は、AS サーバーインスタンス全体で共有されます。
5. **java:** 接頭辞を持つ **relative** 名は、**comp**、**module**、**app**、**global**、または商用の **jboss** の 5 つの名前空間のいずれかに属する必要があります。xxx がこの 5 つのいずれにも一致しない **java:xxx** で始まる名前の場合は、無効な名前エラーが発生します。

[バグを報告する](#)

3.1.8.4. JNDI 名前空間の新ルールに準拠するようアプリケーションを変更

- JBoss EAP 5.1 の JNDI ルックアップの例は次のとおりです。通常、このコードは初期化メソッドに存在します。

```
private ProductManager productManager;
try {
    context = new InitialContext();
    productManager = (ProductManager)
context.lookup("OrderManagerApp/ProductManagerBean/local");
} catch (Exception lookupError) {
    throw new ServletException("Unable to find the ProductManager
bean", lookupError);
}
```

ルックアップ名は **OrderManagerApp/ProductManagerBean/local** になります。

- 以下は、ディペンデンスーインジェクション (依存性の注入) を使用して JBoss EAP 6 で同じルックアップがどのようにコード化されるかを示した例になります。

```
@EJB(lookup="java:app/OrderManagerEJB/ProductManagerBean!services.ej
b.ProductManager")
private ProductManager productManager;
```

ルックアップ値はメンバー変数として定義され、新しい移植可能な **java:app** JNDI 名前空間名である

java:app/OrderManagerEJB/ProductManagerBean!services.ejb.ProductManager が使用されます。

- ディペンデンシーインジェクション (依存性の注入)を使用したくない場合は、前述のとおり InitialContext を新規作成し、新しい JNDI 名前空間名を使用するようにルックアップを編集します。

```
private ProductManager productManager;
try {
    context = new InitialContext();
    productManager = (ProductManager)
context.lookup("java:app/OrderManagerEJB/ProductManagerBean!services
.ejb.ProductManager");
} catch(Exception lookupError) {
    throw new ServletException("Unable to find the ProductManager
bean", lookupError);
}
```

[バグを報告する](#)

3.1.8.5. 以前のリリースでの JNDI 名前空間の例、および JBoss EAP 6 での名前空間の指定方法

表3.2 JNDI 名前空間マッピングテーブル

JBoss EAP 5.x の名前空間	JBoss EAP 6 の名前空間	追加コメント
OrderManagerApp/ProductManagerBean/local	java:module/ProductManagerBean!services.ejb.ProductManager	Java EE 6 の標準バインディング。現在のモジュールヘスコープ指定され、同じモジュール内でのみアクセス可能。
OrderManagerApp/ProductManagerBean/local	java:app/OrderManagerEJB/ProductManagerBean!services.ejb.ProductManager	Java EE 6 の標準バインディング。現在のアプリケーションヘスコープ指定され、同じアプリケーション内でのみアクセス可能。
OrderManagerApp/ProductManagerBean/local	java:global/OrderManagerApp/OrderManagerEJB/ProductManagerBean!services.ejb.ProductManager	Java EE 6 の標準バインディング。アプリケーションサーバーヘスコープ指定され、グローバルにアクセス可能です。
java:comp/UserTransaction	java:comp/UserTransaction	名前空間は現在のコンポーネントヘスコープ指定されます。アプリケーションによって直接作成されるスレッドなど、Java EE 6 でないスレッドはアクセスできません。
java:comp/UserTransaction	java:jboss/UserTransaction	グローバルにアクセス可能です。java:comp/UserTransaction が使用できないときに使用します。
java:/TransactionManager	java:jboss/TransactionManager	

JBoss EAP 5.x の名前空間	JBoss EAP 6 の名前空間	追加コメント
java:/TransactionSynchronizationRegistry	java:jboss/TransactionSynchronizationRegistry	

[バグを報告する](#)

3.2. アプリケーションのアーキテクチャーやコンポーネントによって異なる変更

3.2.1. アプリケーションのアーキテクチャーやコンポーネントによって異なる変更の確認

アプリケーションが下記の技術やコンポーネントを使用する場合、JBoss EAP 6 への移行時にアプリケーションの変更が必要となることがあります。

Hibernate および JPA

アプリケーションが Hibernate または JPA を使用する場合は、アプリケーションを変更する必要があります。詳細は、[「Hibernate や JPA を使用するアプリケーションの更新」](#)を参照してください。

REST

アプリケーションが JAX-RS を使用する場合、JBoss EAP 6 は自動的に RESTEasy を設定するため、手作業で設定する必要がなくなりました。詳細は、[「JAX-RS および RESTEasy の変更の設定」](#)を参照してください。

LDAP

JBoss EAP 6 では LDAP セキュリティーレلمの設定が異なります。アプリケーションが LDAP を使用する場合は、[「LDAP セキュリティーレلمの変更設定」](#)で詳細を確認してください。

メッセージング

JBoss Messaging は JBoss EAP 6 から除外されました。アプリケーションがメッセージングプロバイダーとして JBoss Messaging を使用する場合は、JBoss Messaging コードを HornetQ に置き換える必要があります。詳細は、[「HornetQ を JMS プロバイダーとして使用するためにアプリケーションを移行」](#)を参照してください。

クラスタリング

JBoss EAP 6 では、クラスタリングを有効にする方法が変更になりました。詳細は、[「クラスタリングに対するアプリケーションの変更」](#)を参照してください。

サービススタイルのデプロイメント

JBoss EAP 6 では、サービススタイル記述子を使用しないようになりましたが、できる限り変更がない状態でコンテナはサービススタイルデプロイメントをサポートします。デプロイメントの情報については、[「サービススタイルデプロイメントを使用するアプリケーションの更新」](#)を参照してください。

リモート呼び出し

アプリケーションがリモート呼び出しを行う場合、JNDI を使用して Bean のプロキシをルックアッ

プし、返されたプロキシ上で呼び出しできます。必要な構文や名前空間の変更については、[「JBoss EAP 5 にデプロイされ、JBoss EAP 6 へリモート呼び出しを行うアプリケーションの移行」](#)を参照してください。

Seam 2.2

アプリケーションが Seam 2.2 を使用する場合は、[「Seam 2.2 アーカイブの JBoss EAP 6 への移行」](#)を参照して必要な変更について確認してください。

Spring

アプリケーションが Spring を使用する場合は、[「Spring アプリケーションの移行」](#)を参照してください。

移行に影響する可能性があるその他の変更

アプリケーションに影響する可能性がある JBoss EAP 6 のその他の変更については、[「移行に影響する可能性があるその他の変更について理解する」](#)を参照してください。

[バグを報告する](#)

3.2.2. Hibernate および JPA の変更

3.2.2.1. Hibernate や JPA を使用するアプリケーションの更新

概要

アプリケーションが Hibernate または JPA を使用する場合は、次の項を読んで JBoss EAP 6 への移行に必要な変更を行ってください。

- [「Hibernate および JPA を使用するアプリケーションの変更設定」](#)
- [「Hibernate 4 を使用するよう Hibernate 3 のアプリケーションを更新する」](#)
- [「JPA 2.0 の仕様に準拠するようアプリケーションを更新する」](#)
- [「Infinispan による JPA/Hibernate 2 次キャッシュの置き換え」](#)
- [「Hibernate Validator 4 への移行」](#)

[バグを報告する](#)

3.2.2.2. Hibernate および JPA を使用するアプリケーションの変更設定

概要

アプリケーションに `persistence.xml` ファイルが含まれていたり、コードが `@PersistenceContext` アノテーションや `@PersistenceUnit` アノテーションを使用する場合、JBoss EAP 6 はデプロイメント中にこれを検出し、アプリケーションによって JPA が使用されることを想定します。Hibernate 4 とその他の依存関係の一部を暗黙的にアプリケーションのクラスパスへ追加します。

現在、アプリケーションが Hibernate 3 ライブラリーを使用する場合、ほとんどの場合で Hibernate 4 へ切り替えることが可能で、Hibernate 4 を使用して正常に実行されるはずです。しかし、アプリケーションをデプロイするときに `ClassNotFoundException` が発生した場合、次の方法の 1 つを用いて問題を解決を図ることができます。



重要

Seam 2.2 で Hibernate を直接使用するアプリケーションは、アプリケーション内部にパッケージ化された Hibernate 3 のバージョンを使用することがあります。JBoss EAP 6 の org.hibernate モジュールを介して提供される Hibernate 4 は、Seam 2.2 によってサポートされません。この例の目的は、最初の手順として JBoss EAP 6 でアプリケーションを実行させることです。Hibernate 3 を Seam 2.2 アプリケーションでパッケージ化する設定はサポートされないことに注意してください。

手順3.13 アプリケーションの設定

1. 必要な Hibernate 3 の JAR をアプリケーションライブラリーへコピーする

見つからないクラスが含まれる特定の Hibernate 3 JAR をアプリケーションの **lib/** ディレクトリへコピーするか、他の方法を使用してクラスパスに追加すると問題を解決できることがあります。これにより、Hibernate のバージョンを複数使用することが原因で **ClassCastException** や他のクラスローディングの問題が発生することがあります。この問題が発生した場合、次の方法で対処する必要があります。

2. Hibernate 3 ライブラリーのみを使用するようサーバーへ指示する

JBoss EAP 6 では、Hibernate 3.5 (およびそれ以降のバージョン) の永続性プロバイダー jar をアプリケーションと共にパッケージ化することができます。サーバーで Hibernate 3 ライブラリーのみを使用し、Hibernate 4 ライブラリーを除外するには、次のように **jboss.as.jpa.providerModule** を **persistence.xml** の **hibernate3-bundled** に設定する必要があります。

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
version="1.0">
  <persistence-unit name="plannerdatasource_pu">
    <description>Hibernate 3 Persistence Unit.</description>
    <jta-data-source>java:jboss/datasources/PlannerDS</jta-data-
source>
    <properties>
      <property name="hibernate.show_sql" value="false" />
      <property name="jboss.as.jpa.providerModule"
value="hibernate3-bundled" />
    </properties>
  </persistence-unit>
</persistence>
```

Java Persistence API (JPA) デプロイヤーは、アプリケーションに永続性プロバイダーがあることを検出し、Hibernate 3 ライブラリーを使用します。JPA 永続性プロパティの詳細は、「[永続ユニットプロパティ](#)」を参照してください。

3. Hibernate の 2 次キャッシュの無効化

JBoss EAP 6 では、Hibernate 3 の 2 次キャッシュの挙動が以前のリリースと異なります。アプリケーションを用いて Hibernate の 2 次キャッシュを使用している場合、Hibernate 4 にアップグレードするまで 2 次キャッシュを無効にする必要があります。2 次キャッシュを無効にするには、**persistence.xml** ファイルの **<hibernate.cache.use_second_level_cache>** を **false** に設定します。

[バグを報告する](#)

3.2.2.3. 永続ユニットプロパティ

Hibernate 4.x 設定プロパティ

JBoss EAP 6 は、以下の Hibernate 4.x 設定プロパティを自動的に設定します。

表3.3 Hibernate 永続ユニットプロパティ

プロパティ名	デフォルト値	目的
hibernate.id.new_generator_mappings	true	<p>この設定は、@GeneratedValue(AUTO) を使用して新しいエンティティに対して一意のインデックスキーを生成する場合に有効です。新規のアプリケーションのデフォルト値は true になるはずですが、Hibernate 3.3.x を使用した既存のアプリケーションが継続してシーケンスオブジェクトやテーブルベースのジェネレーターを使用し、後方互換性を維持するにはデフォルト値を false に変更する必要がある場合があります。アプリケーションは persistence.xml ファイルにあるこの値を上書きすることが可能です。</p> <p>この動作の詳細は以下を参照してください。</p>
hibernate.transaction.jta.platform	org.hibernate.service.jpa.platform.spi.JtaPlatform インターフェースのインスタンス	このクラスはトランザクションマネージャー、ユーザーのトランザクション、およびトランザクション同期化レジストリーを Hibernate に渡します。
hibernate.ejb.resource_scanner	org.hibernate.ejb.packaging.Scanner インターフェースのインスタンス	このクラスは、JBoss EAP 6 のアノテーションインデクサーを使用して、より高速なデプロイメントを提供する方法を認識しています。
hibernate.transaction.manager_lookup_class		このプロパティは hibernate.transaction.jta.platform と競合することがあるため、persistence.xml に存在する場合は削除されます。
hibernate.session_factory_name	<i>QUALIFIED_PERSISTENCE_UNIT_NAME</i>	アプリケーション名 + 永続ユニット名に設定されます。アプリケーションは異なる値を指定することができますが、JBoss EAP 6 インスタンス上のすべてのアプリケーションデプロイメントで一意となる値でなければなりません。
hibernate.session_factory_name_is_jndi	false	アプリケーションが hibernate.session_factory_name の値を指定しなかった場合のみ設定されます。
hibernate.ejb.entitymanager_factory_name	<i>QUALIFIED_PERSISTENCE_UNIT_NAME</i>	アプリケーション名 + 永続ユニット名に設定されます。アプリケーションは異なる値を指定することができますが、JBoss EAP 6 インスタンス上のすべてのアプリケーションデプロイメントで一意となる値でなければなりません。

Hibernate 4.x では **new_generator_mappings** が **true** に設定されると以下が実行されます。

- `@GeneratedValue(AUTO)` が `org.hibernate.id.enhanced.SequenceStyleGenerator` ヘマッピングします。
- `@GeneratedValue(TABLE)` が `org.hibernate.id.enhanced.TableGenerator` ヘマッピングします。
- `@GeneratedValue(SEQUENCE)` が `org.hibernate.id.enhanced.SequenceStyleGenerator` ヘマッピングします。

Hibernate 4.x では `new_generator_mappings` が `false` に設定されると以下が実行されます。

- `@GeneratedValue(AUTO)` が Hibernate "native" ヘマッピングします。
- `@GeneratedValue(TABLE)` が `org.hibernate.id.MultipleHiLoPerTableGenerator` ヘマッピングします。
- `@GeneratedValue(SEQUENCE)` が Hibernate "seqhilo" ヘマッピングします。

これらのプロパティの詳細は、<http://www.hibernate.org/docs> および [Hibernate 4.1 Developer Guide](#) を参照してください。

JPA 永続プロパティ

以下の JPA プロパティが、`persistence.xml` ファイルの永続ユニット定義でサポートされます。

表3.4 JPA 永続ユニットプロパティ

プロパティ名	デフォルト値	目的
<code>jboss.as.jpa.providerModule</code>	<code>org.hibernate</code>	<p>永続プロバイダーモジュールの名前。</p> <p>Hibernate 3 JAR がアプリケーションアーカイブに含まれる場合、値は hibernate3-bundled である必要があります。</p> <p>永続プロバイダーがアプリケーションでパッケージ化された場合、この値は application である必要があります。</p>
<code>jboss.as.jpa.adapterModule</code>	<code>org.jboss.as.jpa.hibernate:4</code>	<p>JBoss EAP 6 が永続プロバイダーと動作するようにする統合クラスの名前。</p> <p>現在の有効値は以下のとおりです。</p> <ul style="list-style-type: none"> ● <code>org.jboss.as.jpa.hibernate:4</code>: これは、Hibernate 4 統合クラス用です。 ● <code>org.jboss.as.jpa.hibernate:3</code>: これは、Hibernate 3 統合クラス用です。

[バグを報告する](#)

3.2.2.4. Hibernate 4 を使用するよう Hibernate 3 のアプリケーションを更新する

概要

Hibernate 4 を使用するようアプリケーションを更新する場合、一般的な更新は、アプリケーションが現在使用する Hibernate のバージョンに関係なく適用されます。その他の更新についてはアプリケーションが現在使用するバージョンを判断する必要があります。

手順3.14 Hibernate 4 を使用するようアプリケーションを更新する

1. 自動インクリメントシーケンスジェネレーターのデフォルトの動作は JBoss EAP 6 で変更になりました。詳細は、「[Hibernate アイデンティティ自動生成値の既存動作の保持](#)」を参照してください。
2. アプリケーションによって現在使用されている Hibernate のバージョンを判断し、下記より適切な更新手順を選択します。
 - 「[Hibernate 3.3.x アプリケーションの Hibernate 4.x への移行](#)」
 - 「[Hibernate 3.5.x アプリケーションの Hibernate 4.x への移行](#)」
3. アプリケーションをクラスター化された環境で実行する場合は「[クラスター環境で稼働する、移行された Seam および Hibernate アプリケーションの永続プロパティの変更](#)」を参照してください。

バグを報告する

3.2.2.5. Hibernate アイデンティティ自動生成値の既存動作の保持

Hibernate 3.5 には、`@GeneratedValue` を使用する際にアイデンティティやシーケンスカラムの生成方法を指示する `hibernate.id.new_generator_mappings` というコアプロパティが導入されました。JBoss EAP 6 では、このプロパティのデフォルト値が次のように設定されています。

- ネイティブの Hibernate アプリケーションをデプロイする場合、デフォルト値は **false** になります。
- JPA アプリケーションをデプロイする場合、デフォルト値は **true** になります。

新規アプリケーションのガイドライン

`@GeneratedValue` アノテーションを使用する新しいアプリケーションでは `hibernate.id.new_generator_mappings` プロパティの値を **true** に設定するようにします。この設定は異なるデータベースにわたって移植性が高まるため推奨設定となります。ほとんどの場合で効率がよく、場合によっては JPA 2 仕様との互換性に対応します。

- 新しい JPA アプリケーションでは JBoss EAP 6 の `hibernate.id.new_generator_mappings` プロパティのデフォルトは **true** になります。この値は変更しないでください。
- 新しいネイティブの Hibernate アプリケーションでは JBoss EAP 6 の `hibernate.id.new_generator_mappings` プロパティのデフォルトは **false** になります。このプロパティを **true** に設定してください。

既存の JBoss EAP 5 アプリケーションに適用されるガイドライン

JBoss EAP 6 へ移行する際、新しいエンティティのプライマリキー値の作成に、`@GeneratedValue` アノテーションを使用する既存のアプリケーションと同じジェネレーターが使用されるようにする必要があります。

- 既存の JPA アプリケーションでは JBoss EAP 6 の **hibernate.id.new_generator_mappings** プロパティのデフォルトは **true** になります。 **persistence.xml** ファイルでこのプロパティを **false** に設定してください。
- 既存のネイティブ Hibernate アプリケーションでは JBoss EAP 6 の **hibernate.id.new_generator_mappings** プロパティのデフォルト値は **false** になります。この値は変更しないでください。

これらプロパティ設定の詳細は「[永続ユニットプロパティ](#)」を参照してください。

[バグを報告する](#)

3.2.2.6. Hibernate 3.3.x アプリケーションの Hibernate 4.x への移行

1. Hibernate の text タイプを JDBC LONGVARCHAR へマッピングする

バージョンが 3.5 以前の Hibernate では **text** 型は **JDBC CLOB** へマッピングされていました。Java **String** プロパティを **JDBC CLOB** へマッピングするため、新しい Hibernate タイプ **materialized_clob** が Hibernate 4 に追加されました。**JDBC CLOB** へのマッピングが目的で **type="text"** と設定されているプロパティがアプリケーションにある場合は、次の項目の 1 つを実行する必要があります。

- アプリケーションが hbm マッピングファイルを使用する場合、プロパティを **type="materialized_clob"** に変更します。
- アプリケーションがアノテーションを使用する場合、**@Type(type = "text")** を **@Lob** に置き換えます。

2. コードを確認し戻り値型の変更を探す

数値集約基準の射影 (projection) は HQL と同じ値型を返すようになりました。その結果、**org.hibernate.criterion** の以下の射影からの戻り型が変更されました。

- CountProjection**、**Projections.rowCount()**、**Projections.count(propertyName)**、および **Projections.countDistinct(propertyName)** の変更により、**count** および **count distinct** 射影は **Long** 値を返すようになりました。
- Projections.sum(propertyName)** の変更により、**sum** 射影はプロパティ型によって異なる値型を返すようになりました。



注記

アプリケーションコードを変更しないと、**java.lang.ClassCastException** が発生する原因となります。

- Long**、**Short**、**Integer**、またはプリミティブ型の整数としてマッピングされているプロパティは **Long** 値が返されます。
- Float**、**Double**、またはプリミティブ型の浮動小数としてマッピングされているプロパティは **Double** 値が返されます。

[バグを報告する](#)

3.2.2.7. Hibernate 3.5.x アプリケーションの Hibernate 4.x への移行

1. **AnnotationConfiguration** を設定へマージします。

AnnotationConfiguration はすでに廃止されていますが、移行に影響しないようにする必要があります。

hbm.xml ファイルを使用している場合、JBoss EAP 6 では 以前のリリースで使用された **org.hibernate.cfg.DefaultNamingStrategy** ではなく、**AnnotationConfiguration** の **org.hibernate.cfg.EJB3NamingStrategy** が使用されることに注意してください。そのため、ネーミングの不一致が発生する可能性があります。ネーミングストラテジーに依存してアソシエーション (多対多および要素のコレクション) テーブルの名前をデフォルトにする場合は、この問題が発生することがあります。この問題を解決するには、レガシーの **org.hibernate.cfg.DefaultNamingStrategy** を使用するように Hibernate に指示するため、**Configuration#setNamingStrategy** を呼び出して **org.hibernate.cfg.DefaultNamingStrategy#INSTANCE** に渡します。

2. 下表の新しい Hibernate DTD ファイル名に適合するよう名前空間を変更します。

表3.5 DTD 名前空間マッピングテーブル

以前の DTD 名前空間	新しい DTD 名前空間
<code>http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd</code>	<code>http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd</code>
<code>http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd</code>	<code>http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd</code>

3. 環境変数を編集します。
 - a. Oracle で **materialized_clob** または **materialized_blob** プロパティを使用している場合、グローバル環境変数 **hibernate.jdbc.use_streams_for_binary** を true に設定する必要があります。
 - b. PostgreSQL で **CLOB** または **BLOB** プロパティを使用している場合、グローバル環境変数 **hibernate.jdbc.use_streams_for_binary** を false に設定する必要があります。

バグを報告する

3.2.2.8. クラスター環境で稼働する、移行された Seam および Hibernate アプリケーションの永続プロパティの変更

JPA コンテナによって管理されるアプリケーションを移行する場合は、拡張された永続コンテキストのシリアル化に影響するプロパティが自動的にコンテナに渡されます。

ただし、Hibernate の変更により、移行した Seam または Hibernate アプリケーションをクラスター化された環境で実行すると、シリアル化の問題が発生する可能性があります。以下のようなエラーログが記録される場合があります。

```
javax.ejb.EJBTransactionRolledbackException: JBAS010361: Failed to
deserialize
....
Caused by: java.io.InvalidObjectException: could not resolve session
factory during session deserialization
[uuid=8aa29e74373ce3a301373ce3a44b0000, name=null]
```

このようなエラーを解決するには、設定ファイルのプロパティを変更する必要があります。ほとんどの場合、設定ファイルは **persistence.xml** ファイルになります。ネイティブの Hibernate API アプリケーションでは **hibernate.cfg.xml** ファイルになります。

手順3.15 クラスター化された環境で稼働するよう永続プロパティを設定

1. **hibernate.session_factory_name** 値を一意名に設定します。この名前は、JBoss EAP 6 インスタンス上のすべてのアプリケーションデプロイメントで一意となる必要があります。例は次のとおりです。

```
<property name="hibernate.session_factory_name" value="jboss-seam-booking.ear_session_factory"/>
```

2. **hibernate.ejb.entitymanager_factory_name** 値を一意名に設定します。この名前は、JBoss EAP 6 インスタンス上のすべてのアプリケーションデプロイメントで一意となる必要があります。例は次のとおりです。

```
<property name="hibernate.ejb.entitymanager_factory_name" value="seam-booking.ear_PersistenceUnitName"/>
```

Hibernate JPA 永続ユニットプロパティに関する詳細は、「[永続ユニットプロパティ](#)」を参照してください。

[バグを報告する](#)

3.2.2.9. JPA 2.0 の仕様に準拠するようアプリケーションを更新する

概要

JPA 2.0 の仕様では、永続コンテキストが JTA トランザクションの外部では伝播できないことが要件となっています。トランザクションスコープの永続コンテキストのみがアプリケーションによって使用される場合、JBoss EAP 6 での挙動は以前のバージョンと変わらないため、変更を加える必要はありません。アプリケーションが拡張永続コンテキスト (XPC) を使用してデータ変更のキューやバッチ処理を許可する場合は、アプリケーションを変更する必要があります。

永続コンテキストの伝播挙動

拡張永続コンテキストを使用するステートフルセッション Bean である **Bean1** がアプリケーションにあり、トランザクションスコープの永続コンテキストを使用するステートレスセッション Bean である **Bean2** を呼び出す場合、次のような挙動が想定されます。

- **Bean1** が JTA トランザクションを開始し、JTA トランザクションがアクティブな状態で **Bean2** メソッドを呼び出す場合、JBoss EAP 6 での挙動は以前のリリースと変わらないため、変更を加える必要はありません。
- **Bean1** が JTA トランザクションを開始せず、**Bean2** メソッドを呼び出す場合、JBoss EAP 6 は拡張永続コンテキストを **Bean2** へ伝播しません。この挙動は、拡張永続コンテキストを **Bean2** へ伝播した以前のリリースとは異なっています。拡張永続コンテキストがトランザクションエンティティマネージャーによって Bean へ伝播されることをアプリケーションが想定している場合、アクティブな JTA トランザクション内で呼び出しを行うようにアプリケーションを変更する必要があります。

[バグを報告する](#)

3.2.2.10. Infinispan による JPA/Hibernate 2 次キャッシュの置き換え

概要

2 次キャッシュ (2LC) では、JBoss Cache は Infinispan に置き換えられました。これにより、**persistence.xml** ファイルの変更が必要になります。使用する 2 次キャッシュが JPA または Hibernate であるかによって、構文は若干異なります。ここで取り上げる例は Hibernate の使用が前提となっています。

以下は、JBoss EAP 5.x の **persistence.xml** ファイルで 2 次キャッシュのプロパティを設定する例です。

```
<property name="hibernate.cache.region.factory_class"
value="org.hibernate.cache.jbc2.JndiMultiplexedJBossCacheRegionFactory"/>
<property name="hibernate.cache.region.jbc2.cachefactory"
value="java:CacheManager"/>
<property name="hibernate.cache.use_second_level_cache" value="true"/>
<property name="hibernate.cache.region.jbc2.cfg.entity" value="mvcc-
entity"/>
<property name="hibernate.cache.region_prefix" value="services"/>
```

以下の手順では、この例を使用して JBoss EAP 6 で Infinispan を設定します。

手順3.16 Infinispan を使用するよう persistence.xml ファイルを変更する

1. JBoss EAP 6 の JPA アプリケーション向けに Infinispan を設定する

JBoss EAP 6 で Infinispan を使用し、プロパティを指定して JPA アプリケーションに同じ設定を行う方法は次のとおりです。

```
<property name="hibernate.cache.use_second_level_cache"
value="true"/>
```

また、次のように、**shared-cache-mode** を **ENABLE_SELECTIVE** または **ALL** の値で指定する必要があります。

- デフォルト値は **ENABLE_SELECTIVE** で、これが推奨値となります。この場合、エンティティは明示的にキャッシュ可能であるとマークされない限りキャッシュされません。

```
<shared-cache-mode>ENABLE_SELECTIVE</shared-cache-mode>
```

- **ALL** では、エンティティはキャッシュ不可能としてマークした場合であっても常にキャッシュされます。

```
<shared-cache-mode>ALL</shared-cache-mode>
```

2. JBoss EAP 6 のネイティブ Hibernate アプリケーション向けに Infinispan を設定する

JBoss EAP 6 で Infinispan を使用し、ネイティブ Hibernate アプリケーションに同じ設定を指定する方法は次のとおりです。

```
<property name="hibernate.cache.region.factory_class"
value="org.jboss.as.jpa.hibernate4.infinispan.InfinispanRegionFactory"/>
<property name="hibernate.cache.infinispan.cachemanager"
value="java:jboss/infinispan/container/hibernate"/>
<property name="hibernate.transaction.manager_lookup_class"
```

```
value="org.hibernate.transaction.JBossTransactionManagerLookup"/>
<property name="hibernate.cache.use_second_level_cache"
value="true"/>
```

また、以下の依存関係を **MANIFEST.MF** ファイルに追加する必要があります。

```
Manifest-Version: 1.0
Dependencies: org.infinispan, org.hibernate
```

Hibernate キャッシュプロパティの詳細は、「[Hibernate キャッシュプロパティ](#)」を参照してください。

[バグを報告する](#)

3.2.2.11. Hibernate キャッシュプロパティ

表3.6 プロパティ

プロパティ名	説明
<code>hibernate.cache.region.factory_class</code>	カスタム CacheProvider のクラス名。
<code>hibernate.cache.use_minimal_puts</code>	ブール変数です。2 次キャッシュの操作を最適化し、読み取りを増やして書き込みを最小限にします。これはクラスター化されたキャッシュで最も便利な設定であり、Hibernate 3 ではクラスター化されたキャッシュの実装に対してデフォルトで有効になっています。
<code>hibernate.cache.use_query_cache</code>	ブール変数です。クエリーキャッシュを有効にします。各クエリーをキャッシュ可能に設定する必要があります。
<code>hibernate.cache.use_second_level_cache</code>	ブール変数です。 <cache> マッピングを指定するクラスに対してデフォルトで有効になっている 2 次キャッシュを完全に無効にするため使用されます。
<code>hibernate.cache.query_cache_factory</code>	カスタム QueryCache インターフェースのクラス名です。デフォルト値は組み込みの StandardQueryCache です。
<code>hibernate.cache.region_prefix</code>	2 次キャッシュのリージョン名に使用するプレフィックスです。
<code>hibernate.cache.use_structured_entries</code>	ブール変数です。人間が解読可能な形式でデータを 2 次キャッシュに保存するよう Hibernate を設定します。

プロパティ名	説明
<code>hibernate.cache.default_cache_concurrency_strategy</code>	<code>@Cacheable</code> または <code>@Cache</code> が使用される場合に使用するデフォルトの <code>org.hibernate.annotations.CacheConcurrencyStrategy</code> の名前を付与するため使用される設定です。 <code>@Cache(strategy="...")</code> を使用してこのデフォルト値が上書きされます。

[バグを報告する](#)

3.2.2.12. Hibernate Validator 4 への移行

概要

Hibernate Validator 4.x は、[JSR 303 - Bean Validation](#) を実装する完全に新しいコードベースです。Validator 3.x から 4.x への移行プロセスは非常に簡単ですが、アプリケーションの移行時にいくつかの変更を行う必要があります。

手順3.17 以下の 1 つまたは複数のタスクを実行する必要がある場合があります。

1. デフォルトの ValidatorFactory へのアクセス

JBoss EAP 6 は、デフォルトの ValidatorFactory を `java:comp/ValidatorFactory` 以下にある JNDI コンテキストにバインドします。

2. ライフサイクルでトリガーされた検証の理解

Hibernate Core 4 と組み合わせて使用する場合、ライフサイクルベースの検証は Hibernate Core により自動的に有効になります。

- a. 検証は、エンティティ **INSERT** 操作、**UPDATE** 操作、および **DELETE** 操作に対して行われます。
- b. 次のプロパティを使用してイベントタイプによってグループが検証されるよう設定することができます。

- `javax.persistence.validation.group.pre-persist`

- `javax.persistence.validation.group.pre-update`

- `javax.persistence.validation.group.pre-remove`

これらのプロパティの値は、バリデーションを行うグループのカンマ区切り完全修飾クラス名です。

バリデーショングループは、Bean Validation 仕様の新しい機能です。この新しい機能を使用しない場合は、Hibernate Validator 4 に移行するときに変更を必要としません。

- c. ライフサイクルベースのバリデーションを無効にするには、`javax.persistence.validation.mode` プロパティを `none` に設定します。このプロパティに有効な他の値は `auto` (デフォルト値)、`callback`、および `ddl` です。

3. アプリケーションが手動バリデーションを使用するよう設定する

- a. 手動でバリデーションを制御する場合は、次のいずれかの方法で Validator を作成できます。

- **getValidator()** メソッドを使用して、**ValidatorFactory** から **Validator** インスタンスを作成します。
 - **Validator** インスタンスを EJB、CDI Bean、または他の Java EE インジェクト可能リソースにインジェクトします。
- b. **Validator** インスタンスをカスタマイズするため
に、**ValidatorFactory.usingContext()** により返された **ValidatorContext** を使用できます。この API を使用して、カスタム **MessageInterpolator**、**TraversableResolver**、および **ConstraintValidatorFactory** を設定できます。これらのインターフェースは、Bean Validation 仕様で指定され、Hibernate Validator 4 で新しい機能です。
4. 新しい **Bean Validation** の制約を使用するようコードを変更する
Hibernate Validator 4 への移行時に、新しい Bean レベルのバリデーション制約では、コードの変更が必要です。
- a. Hibernate Validator 4 にアップグレードする場合は、次のパッケージの制約を使用する必要があります。
- **javax.validation.constraints**
 - **org.hibernate.validator.constraints**
- b. Hibernate Validator 3 に存在していたすべての制約は、Hibernate Validator 4 でも引き続き利用できます。これらを使用するには、指定されたクラスをインポートします。場合によっては、制約パラメーターの名前またはタイプを変更する必要があります。
5. カスタム制約の使用
Hibernate Validator 3 では、カスタム制約で **org.hibernate.validator.Validator** インターフェースを実装する必要がありました。Hibernate Validator 4 では、**javax.validation.ConstraintValidator** インターフェースを実装する必要があります。このインターフェースには、以前のインターフェースと同じ **initialize()** メソッドと **isValid()** メソッドが含まれますが、メソッドシグネチャーが変更されました。また、代替の **DDL** は Hibernate Validator 4 でサポートされなくなりました。

[バグを報告する](#)

3.2.3. JSF の変更

3.2.3.1. アプリケーションが JSF の古いバージョンを使用できるようにする

概要

アプリケーションが JSF の古いバージョンを使用する場合は、JSF 2.0 にアップグレードする必要はありません。代わりに、**jboss-deployment-structure.xml** ファイルを作成して、JBoss EAP 6 がアプリケーションデプロイメントで JSF 2.0 ではなく JSF 1.2 を使用するよう要求できます。この JBoss 固有のデプロイメント記述子は、クラスローディングを制御するために使用され、WAR の **META-INF/** または **WEB-INF/** ディレクトリー、あるいは EAR の **META-INF/** ディレクトリーに格納されます。

JSF 1.2 モジュールの依存関係を追加し、JSF 2.0 モジュールが自動的にローディングされないようにする **jboss-deployment-structure.xml** ファイルの例は次のとおりです。

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-structure:1.0">
```

```

<deployment>
  <dependencies>
    <module name="javax.faces.api" slot="1.2" export="true"/>
    <module name="com.sun.jsf-impl" slot="1.2" export="true"/>
  </dependencies>
</deployment>
<sub-deployment name="jboss-seam-booking.war">
  <exclusions>
    <module name="javax.faces.api" slot="main"/>
    <module name="com.sun.jsf-impl" slot="main"/>
  </exclusions>
  <dependencies>
    <module name="javax.faces.api" slot="1.2"/>
    <module name="com.sun.jsf-impl" slot="1.2"/>
  </dependencies>
</sub-deployment>
</jboss-deployment-structure>

```

バグを報告する

3.2.4. Web サービスの変更

3.2.4.1. Web サービスの変更

JBoss EAP 6 は、JAX-WS Web サービスエンドポイントのデプロイメントをサポートします。このサポートは JBossWS によって提供されます。Web サービスの詳細は、https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/ にある JBoss EAP 6 向け『開発ガイド』の章「JAX-WS Web サービス」を参照してください。

JBossWS 4 には、移行に影響する可能性がある以下の変更が含まれています。

JBossWS API プロジェクトの変更

JBossWS では、SPI および Common コンポーネントがリファクタリングされました。下表は、アプリケーションの移行に影響する可能性がある API およびパッケージの変更を表しています。

表3.7 サイズログハンドラプロパティ

従来の JAR	従来のパッケージ	新しい JAR	新しいパッケージ
JBossWS SPI	org.jboss.wsf.spi.annotation.*	JBossWS API	org.jboss.ws.api.annotation.*
JBossWS SPI	org.jboss.wsf.spi.binding.*	JBossWS API	org.jboss.ws.api.binding.*
JBossWS SPI	org.jboss.wsf.spi.management.recording.*	JBossWS API	org.jboss.ws.api.monitoring.*
JBossWS SPI	org.jboss.wsf.spi.tools.*	JBossWS API	org.jboss.ws.api.tools.*
JBossWS SPI	org.jboss.wsf.spi.tools.ant.*	JBossWS API	org.jboss.ws.tools.ant.*
JBossWS SPI	org.jboss.wsf.spi.tools.cmd.*	JBossWS API	org.jboss.ws.tools.cmd.*

従来の JAR	従来のパッケージ	新しい JAR	新しいパッケージ
JBossWS SPI	org.jboss.wsf.spi.util.ServiceLoader	JBossWS API	org.jboss.ws.api.util.ServiceLoader
JBossWS Common	org.jboss.wsf.common.*	JBossWS API	org.jboss.ws.common.*
JBossWS Common	org.jboss.wsf.common.handler.*	JBossWS API	org.jboss.ws.api.handler.*
JBossWS Common	org.jboss.wsf.common.addressing.*	JBossWS API	org.jboss.ws.api.addressing.*
JBossWS Common	org.jboss.wsf.common.DOMUtils	JBossWS API	org.jboss.ws.api.util.DOMUtils
JBossWS Native	org.jboss.ws.annotation.EndpointConfig	JBossWS API	org.jboss.ws.api.annotation.EndpointConfig

@WebContext アノテーション

JBossWS 3.4.x では、このアノテーションは JBossWS SPI プロジェクトの **org.jboss.wsf.spi.annotation.WebContext** としてパッケージ化されていました。

JBossWS 4.0 では、このアノテーションは JBossWS API プロジェクトの **org.jboss.ws.api.annotation.WebContext** に移動しました。アプリケーションに廃止された依存関係が含まれている場合は、アプリケーションのソースコードのインポートおよび依存関係を置き換え、新しい JBossWS API JAR に対してコンパイルする必要があります。

また、後方互換性への非対応が変更になりました。**String[] virtualHosts** 属性が **String virtualHost** へ変更になりました。JBoss EAP 6 では、デプロイメント毎に 1 つの仮想ホストのみを指定できます。複数の Web サービスが @WebContext アノテーションを使用する場合は、デプロイメントアーカイブに定義されるすべてのエンドポイントで virtualHost の値が同じになるようにする必要があります。

エンドポイントの設定

JBossWS 4.0 は、JBoss Web Services スタックと、ほとんどの Apache CXF プロジェクトモジュールとの統合を実現します。統合レイヤーにより、JAX-WS を含む標準の Web サービス API を使用できます。また、複雑な設定を必要とせずに、JBoss EAP 6 コンテナ上で Apache CXF の拡張機能を使用できます。

JBoss EAP 6 のドメイン設定にある **webservice** サブシステムには、事前定義されたエンドポイントの設定が含まれています。独自のエンドポイント設定を追加で設定することも可能です。指定のエンドポイント設定の参照には、**@org.jboss.ws.api.annotation.EndpointConfig** アノテーションを使用します。

JBoss サーバーで Web サービスエンドポイントを設定する方法の詳細は、https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/ にある JBoss EAP 6 向け『開発ガイド』の章「JAX-WS Web サービス」を参照してください。

jboss-webservices.xml デプロイメント記述子

JBossWS 4.0 には、Web サービスを設定する新しいデプロイメント記述子が導入されました。**jboss-webservices.xml** ファイルが指定のデプロイメントの追加情報を提供し、廃止された **jboss.xml** ファイルを部分的に置き換えます。

EJB Web サービスデプロイメントでは、**jboss-webservices.xml** 記述子ファイルの想定される場所は **META-INF/** ディレクトリー内になります。WAR ファイルでバンドルされた POJO および EJB Web サービスエンドポイントの想定される場所は、**WEB-INF/** ディレクトリーの **jboss-webservices.xml** ファイル内になります。

以下は、**jboss-webservices.xml** 記述子ファイルの例と、要素を説明する表になります。

```
<webservices>
  <context-root>foo</context-root>
  <config-name>Standard WSSecurity Endpoint</config-name>
  <config-file>META-INF/custom.xml</config-file>
  <property>
    <name>prop.name</name>
    <value>prop.value</value>
  </property>
  <port-component>
    <ejb-name>TestService</ejb-name>
    <port-component-name>TestServicePort</port-component-name>
    <port-component-uri>/*</port-component-uri>
    <auth-method>BASIC</auth-method>
    <transport-guarantee>NONE</transport-guarantee>
    <secure-wsdl-access>true</secure-wsdl-access>
  </port-component>
  <webservice-description>
    <webservice-description-name>TestService</webservice-
description-name>
    <wsdl-publish-location>file:///bar/foo.wsdl</wsdl-publish-
location>
  </webservice-description>
</webservices>
```

表3.8 jboss-webservice.xml ファイル要素の説明

要素名	説明
context-root	Web サービスデプロイメントのコンテキストルートのカスタマイズするために使用されます。
config-name config-file	指定のエンドポイント設定へエンドポイントデプロイメントを関連付けるために使用されます。エンドポイント設定は、参照された設定ファイルまたはドメイン設定の webservices サブシステムに指定されます。
property	Web サービススタックの挙動を設定するため、簡単なプロパティ名と値のペアを指定するために使用されます。
port-component	EJB エンドポイントのターゲット URI のカスタマイズまたはセキュリティ関連プロパティの設定に使用されます。
webservice-description	Web サービス WSDL がパブリッシュされる場所をカスタマイズまたは上書きするために使用されます。

[バグを報告する](#)

3.2.5. JAX-RS および RESTEasy の変更

3.2.5.1. JAX-RS および RESTEasy の変更の設定

JBoss EAP 6 は自動的に RESTEasy を設定するため、手作業で設定する必要はありません。そのため、**web.xml** ファイルから既存の RESTEasy の設定をすべて削除し、次の 3 つのオプションの 1 つに置き換える必要があります。

1. **javax.ws.rs.core.Application** をサブクラス化し、**@ApplicationPath** アノテーションを使用します。

これが最も簡単なオプションで、xml の設定が必要ありません。次のようにアプリケーションで **javax.ws.rs.core.Application** をサブクラス化し、JAX-RS クラスを使用可能にするパスを用いてアノテーションを付けます。

```
@ApplicationPath("/mypath")
public class MyApplication extends Application {
}
```

上記の例では、JAX-RS リソースはパス **/MY_WEB_APP_CONTEXT/mypath/** で使用できるようになります。



注記

パスは **/mypath/*** ではなく **/mypath** として指定する必要があることに注意してください。最後にフォワードスラッシュやアスタリスクがないようにしてください。

2. **javax.ws.rs.core.Application** をサブクラス化し、**web.xml** ファイルを使用して JAX-RS マッピングを設定します。

@ApplicationPath アノテーションを使用したくない場合でも **javax.ws.rs.core.Application** をサブクラス化する必要があります。サブクラス化した後に **web.xml** ファイルに JAX-RS マッピングを設定します。

```
public class MyApplication extends Application {
}

<servlet-mapping>
  <servlet-name>com.acme.MyApplication</servlet-name>
  <url-pattern>/hello/*</url-pattern>
</servlet-mapping>
```

上記の例では、JAX-RS リソースはパス `/MY_WEB_APP_CONTEXT/hello` で使用できるようになります。



注記

この方法を使用して `@ApplicationPath` アノテーションを使用して設定されたアプリケーションパスを上書きすることもできます。

3. web.xml ファイルを変更します。

Application をサブクラス化したくない場合、次のように **web.xml** ファイルで JAX-RS のマッピングを設定することが可能です。

```
<servlet-mapping>
  <servlet-name>javax.ws.rs.core.Application</servlet-name>
  <url-pattern>/hello/*</url-pattern>
</servlet-mapping>
```

上記の例では、JAX-RS リソースはパス `/MY_WEB_APP_CONTEXT/hello` で使用できるようになります。



注記

このオプションを選択した場合、マッピングの追加のみが必要となります。対応するサーブレットを追加する必要はありません。対応するサーブレットはサーバーによって自動的に追加されるはずです。

[バグを報告する](#)

3.2.6. LDAP セキュリティーレلمの変更

3.2.6.1. LDAP セキュリティーレلمの変更設定

JBoss EAP 5 では LDAP セキュリティーレلمは **login-config.xml** ファイルの **<application-policy>** 要素に設定されていました。JBoss EAP 6 では、LDAP セキュリティーレلمはサーバー設定ファイルの **<security-domain>** 要素に設定されています。サーバー設定ファイルはスタンドアロンサーバーでは **standalone/configuration/standalone.xml** ファイルになります。サーバーが管理対象ドメインで実行されている場合、**domain/configuration/domain.xml** ファイルがサーバー設定ファイルになります。

JBoss EAP 5 の **login-config.xml** ファイルにある LDAP セキュリティーレلم設定の例は次のとおりです。

```
<application-policy name="mcp_ldap_domain">
  <authentication>
    <login-module code="org.jboss.security.auth.spi.LdapExtLoginModule"
flag="required">
      <module-option
name="java.naming.factory.initial">com.sun.jndi.ldap.LdapCtxFactory</module-option>
      <module-option
name="java.naming.security.authentication">simple</module-option>
```

```

    . . . .
    </login-module>
  </authentication>
</application-policy>

```

JBoss EAP 6 のサーバーファイルにある LDAP 設定の例は次のとおりです。

```

<subsystem xmlns="urn:jboss:domain:security:1.2">
  <security-domains>
    <security-domain name="mcp_ldap_domain" cache-type="default">
      <authentication>
        <login-module code="org.jboss.security.auth.spi.LdapLoginModule"
flag="required">
          <module-option name="java.naming.factory.initial"
value="com.sun.jndi.ldap.LdapCtxFactory"/>
          <module-option name="java.naming.security.authentication"
value="simple"/>
          . . .
        </login-module>
      </authentication>
    </security-domain>
  </security-domains>
</subsystem>

```

注記

JBoss EAP 6 では XML パーサーが変更になりました。JBoss EAP 5 では、次のようにモジュールオプションを要素の内容として指定しました。

```

<module-option
name="java.naming.factory.initial">com.sun.jndi.ldap.LdapCtxFac
tory</module-option>

```

JBoss Enterprise Application Platform 6 では、次のようにモジュールオプションを「value=」で要素属性として指定する必要があります。

```

<module-option name="java.naming.factory.initial"
value="com.sun.jndi.ldap.LdapCtxFactory"/>

```

[バグを報告する](#)

3.2.7. HornetQ の変更

3.2.7.1. HornetQ および NFS について

NIO をジャーナルタイプとして使用する場合、NFS は HornetQ で使用する JMS データを保存するのに適した方法ではありません。これは、同期ロッキングメカニズムが動作する方法が要因です。ただし、NFS は特定の状況では Red Hat Enterprise Linux サーバーでのみ使用できます。これは、Red Hat Enterprise Linux によって使用される NFS 実装によって実現されます。

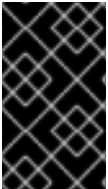
Red Hat Enterprise Linux の NFS 実装は、ダイレクト I/O (O_DIRECT フラグセットでファイルを開く) およびカーネルベースの非同期 I/O の両方をサポートします。これらの機能が両方あるため、厳格な設定ルール下で NFS を共有ストレージオプションとして使用可能です。

- Red Hat Enterprise Linux NFS クライアントのキャッシュは無効にする必要があります。



重要

サーバーログは、JBoss EAP 6 が起動された後にチェックし、ネイティブライブラリーが正常にロードされ、ASYNCIO ジャーナルタイプが使用されることを確認する必要があります。ネイティブライブラリーのロードに失敗した場合は、HornetQ が失敗して NIO ジャーナルタイプを使用し、これはサーバーログに示されます。



重要

非同期 I/O を実装するネイティブライブラリーを使用するには、JBoss EAP 6 が実行されている Red Hat Enterprise Linux システムに **libaio** がインストールされている必要があります。

[バグを報告する](#)

3.2.7.2. 既存の JMS メッセージを JBoss EAP 6 へ移行するために JMS ブリッジを設定する

JBoss EAP 6 では、デフォルトの JMS 実装が JBoss Messaging から HornetQ に変更になりました。JMS ブリッジを使用すると、最も簡単に JMS メッセージを別の環境に移行できます。JMS ブリッジはソースの JMS 宛先からメッセージを消費し、ターゲットの JMS 宛先へ送信します。JMS ブリッジを設定およびデプロイできるサーバーは、JBoss EAP 5.x サーバー、JBoss EAP 6.1 サーバー、およびそれ以降のサーバーです。

JMS メッセージを JBoss EAP 5.x から JBoss EAP 6.x へ移行する方法の詳細は、[「JMS ブリッジの作成」](#) を参照してください。

[バグを報告する](#)

3.2.7.3. JMS ブリッジの作成

概要

JMS ブリッジはソースの JMS キューまたはトピックからメッセージを消費し、通常異なるサーバーにあるターゲットの JMS キューまたはトピックへ送信します。JMS 1.1 に準拠する JMS サーバーの間でメッセージをブリッジングするために使用できます。送信元および宛先の JMS リソースは、JNDI を使用してルックアップされ、JNDI ルックアップのクライアントクラスはモジュールでバンドルされる必要があります。モジュール名は JMS ブリッジ設定で宣言されます。

手順3.18 JMS ブリッジの作成

この手順では、JMS ブリッジを設定して、メッセージを JBoss EAP 5.x サーバーから JBoss EAP 6 サーバーへ移行する方法を示します。

1. ソース JBoss EAP 5.x サーバー上のブリッジの設定

リリース間のクラスの競合を防ぐため、以下の手順に従って JBoss EAP 5.x 上の JMS ブリッジを設定する必要があります。SAR ディレクトリーおよびブリッジの名前は任意で、変更可能です。

- a. **EAP5_HOME/server/PROFILE_NAME/deploy/myBridge.sar/** のように、JBoss EAP 5 デプロイメントディレクトリーでサブディレクトリーを作成し、SAR が含まれるようにします。

- b. **EAP5_HOME/server/PROFILE_NAME/deploy/myBridge.sar/** に **META-INF** という名前のサブディレクトリーを作成します。
- c. **EAP5_HOME/server/PROFILE_NAME/deploy/myBridge.sar/META-INF/** ディレクトリーで **jboss-service.xml** ファイルを作成します。下例のような情報が含まれるようにしてください。

```
<server>
  <loader-repository>
    com.example:archive=unique-archive-name
    <loader-repository-
config>java2ParentDelegation=false</loader-repository-config>
  </loader-repository>

  <!-- JBoss EAP 6 JMS Provider -->
  <mbean code="org.jboss.jms.jndi.JMSProviderLoader"
name="jboss.messaging:service=JMSProviderLoader,name=EnterpriseAp
plicationPlatform6JMSProvider">
    <attribute
name="ProviderName">EnterpriseApplicationPlatform6JMSProvider</at
tribute>
    <attribute
name="ProviderAdapterClass">org.jboss.jms.jndi.JNDIProviderAdapte
r</attribute>
    <attribute
name="FactoryRef">jms/RemoteConnectionFactory</attribute>
    <attribute
name="QueueFactoryRef">jms/RemoteConnectionFactory</attribute>
    <attribute
name="TopicFactoryRef">jms/RemoteConnectionFactory</attribute>
    <attribute name="Properties">

java.naming.factory.initial=org.jboss.naming.remote.client.Initia
lContextFactory

java.naming.provider.url=remote://EnterpriseApplicationPlatform6h
ost:4447
        java.naming.security.principal=jbossuser
        java.naming.security.credentials=jbosspass
    </attribute>
  </mbean>

  <mbean code="org.jboss.jms.server.bridge.BridgeService"
name="jboss.jms:service=Bridge,name=MyBridgeName" xmbean-
dd="xmdesc/Bridge-xmbean.xml">
    <depends optional-attribute-
name="SourceProviderLoader">jboss.messaging:service=JMSProviderLo
ader,name=JMSProvider</depends>
    <depends optional-attribute-
name="TargetProviderLoader">jboss.messaging:service=JMSProviderLo
ader,name=EnterpriseApplicationPlatform6JMSProvider</depends>
    <attribute
name="SourceDestinationLookup">/queue/A</attribute>
    <attribute
name="TargetDestinationLookup">jms/queue/test</attribute>
    <attribute name="QualityOfServiceMode">1</attribute>
```

```

<attribute name="MaxBatchSize">1</attribute>
<attribute name="MaxBatchTime">-1</attribute>
<attribute name="FailureRetryInterval">60000</attribute>
<attribute name="MaxRetries">-1</attribute>
<attribute name="AddMessageIDInHeader">false</attribute>
<attribute name="TargetUsername">jbossuser</attribute>
<attribute name="TargetPassword">jbosspass</attribute>
</mbean>
</server>

```



注記

SAR に分離されたクラスローダーがあるようにするため **load-repository** 要素が存在します。また、JNDI ルックアップとブリッジの「ターゲット」には、ユーザーが「jbossuser」でパスワードが「jbosspass」のセキュリティークレデンシャルが含まれています。これは、JBoss EAP 6 はデフォルトでセキュア化されているからです。パスワードが「jbosspass」のユーザー「jbossuser」は、**EAP_HOME/bin/add_user.sh** スクリプトを使用して **ApplicationRealm** で作成され、**guest** ロールが割り当てられます。

- d. 次の JAR を **EAP_HOME/modules/system/layers/base/** ディレクトリーから **EAP5_HOME/server/PROFILE_NAME/deploy/myBridge.sar/** ディレクトリーへコピーします。 **VERSION_NUMBER** を JBoss EAP 6 ディストリビューションの実際のバージョン番号に置き換えてください。

- **org/hornetq/main/hornetq-core-VERSION_NUMBER.jar**
- **org/hornetq/main/hornetq-jms-VERSION_NUMBER.jar**
- **org/jboss/ejb-client/main/jboss-ejb-client-VERSION_NUMBER.jar**
- **org/jboss/logging/main/jboss-logging-VERSION_NUMBER.jar**
- **org/jboss/logmanager/main/jboss-logmanager-VERSION_NUMBER.jar**
- **org/jboss/marshalling/main/jboss-marshalling-VERSION_NUMBER.jar**
- **org/jboss/marshalling/river/main/jboss-marshalling-
river-VERSION_NUMBER.jar**
- **org/jboss/remote-naming/main/jboss-remote-
naming-VERSION_NUMBER.jar**
- **org/jboss/remoting3/main/jboss-remoting-VERSION_NUMBER.jar**
- **org/jboss/sasl/main/jboss-sasl-VERSION_NUMBER.jar**
- **org/jboss/netty/main/netty-VERSION_NUMBER.jar**
- **org/jboss/remoting3/remote-jmx/main/remoting-
jmx-VERSION_NUMBER.jar**

- `org/jboss/xnio/main/xnio-api-VERSION_NUMBER.jar`
- `org/jboss/xnio/nio/main.xnio-nio-VERSION_NUMBER.jar`



注記

javax API クラスは JBoss EAP 5.x のクラスと競合するため、そのまま **EAP_HOME/bin/client/jboss-client.jar** をコピーしないようにしてください。

2. 宛先 JBoss EAP 6 サーバー上のブリッジの設定

JBoss EAP 6.1 およびそれ以降のバージョンでは、JMS ブリッジを使用して JMS 1.1 に準拠するサーバーからメッセージをブリッジングできます。ソースおよびターゲットの JMS リソースは JNDI を使用してルックアップされるため、ソースメッセージングプロバイダーの JNDI ルックアップクラスまたはメッセージブローカーは JBoss モジュールでバンドルされる必要があります。次の手順では、例として架空の「MyCustomMQ」メッセージブローカーが使用されています。

- a. メッセージプロバイダーの JBoss モジュールを作成します。
 - i. 新しいモジュール向けに **EAP_HOME/modules/system/layers/base/** 下にディレクトリ構造を作成します。**main/** サブディレクトリには、クライアント JAR と **module.xml** ファイルを格納します。**EAP_HOME/modules/system/layers/base/org/mycustommq/main/** は MyCustomMQ メッセージングプロバイダー用に作成されたディレクトリ構造の例になります。
 - ii. **main/** サブディレクトリ内に、メッセージングプロバイダーのモジュール定義が含まれる **module.xml** ファイルを作成します。MyCustomMQ メッセージプロバイダー用に作成された **module.xml** の例は次のとおりです。

```
<?xml version="1.0" encoding="UTF-8"?>
<module xmlns="urn:jboss:module:1.1" name="org.mycustommq">
  <properties>
    <property name="jboss.api" value="private"/>
  </properties>

  <resources>
    <!-- Insert resources required to connect to the
    source or target -->
    <resource-root path="mycustommq-1.2.3.jar" />
    <resource-root path="mylogapi-0.0.1.jar" />
  </resources>

  <dependencies>
    <!-- Add the dependencies required by JMS Bridge code
    -->
    <module name="javax.api" />
    <module name="javax.jms.api" />
    <module name="javax.transaction.api"/>
    <!-- Add a dependency on the org.hornetq module since
    we send -->
    <!-- messages to the HornetQ server embedded in the
    local EAP instance -->
```

```

        <module name="org.hornetq" />
    </dependencies>
</module>

```

- iii. ソースリソースの JNDI ルックアップに必要なメッセージングプロバイダー JAR をモジュールの **main/** サブディレクトリへコピーします。MyCustomMQ モジュールのディレクトリ構造は次のようになるはずです。

```

modules/
  -- system
    -- layers
      -- base
        -- org
          -- mycustommq
            -- main
              -- mycustommq-1.2.3.jar
              -- mylogapi-0.0.1.jar
              -- module.xml

```

- b. JBoss EAP 6 サーバーの **messaging** サブシステムに JMS ブリッジを設定します。

- i. 設定を行う前に、サーバーを停止し、現在のサーバー設定ファイルをバックアップしてください。スタンドアロンサーバーを実行している場合は、**EAP_HOME/standalone/configuration/standalone-full-ha.xml** ファイルをバックアップします。管理対象ドメインを実行している場合は、**EAP_HOME/domain/configuration/domain.xml** ファイルおよび **EAP_HOME/domain/configuration/host.xml** ファイルを両方バックアップします。
- ii. **jms-bridge** 要素を、サーバー設定ファイルの **messaging** サブシステムへ追加します。**source** および **target** 要素は、JNDI ルックアップに使用される JMS リソースの名前を提供します。**user** および **password** クレデンシャルが指定されいると、JMS 接続の作成時に引数として渡されます。

MyCustomMQ メッセージングプロバイダー用に設定された **jms-bridge** 要素の例は次のとおりです。

```

<subsystem xmlns="urn:jboss:domain:messaging:1.3">
    ...
    <jms-bridge name="myBridge" module="org.mycustommq">
        <source>
            <connection-factory name="ConnectionFactory"/>
            <destination name="sourceQ"/>
            <user>user1</user>
            <password>pwd1</password>
            <context>
                <property key="java.naming.factory.initial"
value="org.mycustommq.jndi.MyCustomMQInitialContextFactory"/>
                <property key="java.naming.provider.url"
value="tcp://127.0.0.1:9292"/>
            </context>
        </source>
        <target>
            <connection-factory name="java:/ConnectionFactory"/>
            <destination name="/jms/targetQ"/>

```

```

</target>
<quality-of-service>DUPLICATES_OK</quality-of-service>
<failure-retry-interval>500</failure-retry-interval>
<max-retries>1</max-retries>
<max-batch-size>500</max-batch-size>
<max-batch-time>500</max-batch-time>
<add-messageID-in-header>true</add-messageID-in-header>
</jms-bridge>
</subsystem>

```

上記の例では、JNDI プロパティは **source** の **context** 要素に定義されています。上記の **target** の例のように、**context** 要素が省略されると、JMS リソースはローカルインスタンスでルックアップされます。

[バグを報告する](#)

3.2.7.4. HornetQ を JMS プロバイダーとして使用するためにアプリケーションを移行

JBoss Messaging は、JBoss EAP 6 に同梱されなくなりました。アプリケーションがメッセージングプロバイダーとして JBoss Messaging を使用する場合は、JBoss Messaging コードを HornetQ と置き換える必要があります。

手順3.19 開始する前に

1. クライアントとサーバーをシャットダウンします。
2. JBoss Messaging データのバックアップコピーを作成します。メッセージデータは、**JBM_** という接頭辞でデータベースのテーブルに格納されます。

手順3.20 HornetQ へのプロバイダーの変更

1. 設定の転送

既存の JBoss Messaging 設定を JBoss EAP 6 設定に転送します。以下の設定が、JBoss Messaging サーバーにあるデプロイメント記述子に存在します。

○ 接続ファクトリーサービス設定

この設定は、JBoss Messaging サーバーにデプロイされた JMS 接続ファクトリーを定義します。JBoss Messaging は、アプリケーションサーバーのデプロイメントディレクトリーにある **connection-factories-service.xml** という名前のファイルで接続ファクトリーを設定します。

○ 宛先設定

この設定は、JBoss Messaging サーバーでデプロイされた JMS キューおよびトピックを定義します。デフォルトでは、JBoss Messaging は、アプリケーションサーバーのデプロイメントディレクトリーにある **destinations-service.xml** という名前のファイルで宛先を設定します。

○ メッセージブリッジサービス設定

この設定には、JBoss Messaging サーバーでデプロイされたブリッジサービスが含まれます。デフォルトではブリッジがデプロイされないため、デプロイメントファイルの名前は、JBoss Messaging インストールによって異なります。

2. アプリケーションコードの変更

アプリケーションコードで標準的な JMS を使用する場合は、コードの変更が必要ありません。ただし、アプリケーションがクラスターに接続する場合は、クラスタリングセマンティクスについて HornetQ ドキュメンテーションを参照する必要があります。クラスタリングは、JMS 仕様の範囲外であり、クラスタリング機能の各実装において JBoss Messaging は非常に異なる方法を取ります。

アプリケーションが JBoss Messaging に固有な機能を使用する場合は、HornetQ で利用可能な同等の機能を使用するようコードを変更する必要があります。

HornetQ でメッセージングを設定する方法の詳細は、[「HornetQ でのメッセージングの設定」](#)を参照してください。

3. 既存のメッセージの移行

JMS ブリッジを使用して JBoss Messaging データベースのすべてのメッセージを HornetQ ジャーナルに移動します。JMS ブリッジの設定手順は、[「既存の JMS メッセージを JBoss EAP 6 へ移行するために JMS ブリッジを設定する」](#)を参照してください。

[バグを報告する](#)

3.2.7.5. HornetQ でのメッセージングの設定

JBoss EAP 6 でのメッセージングの設定では、管理コンソールまたは管理 CLI の使用が推奨されます。どちらの管理ツールでも、**standalone.xml** や **domain.xml** 設定ファイルを手作業で編集せずに永続的な変更を行うことができますが、デフォルト設定ファイルのメッセージングコンポーネントについて理解できると便利です。デフォルトの設定ファイルでは、管理ツールを使用するドキュメントサンプルによって参考用の設定ファイルスニペットが提供されます。

[バグを報告する](#)

3.2.8. クラスタリングの変更

3.2.8.1. クラスタリングに対するアプリケーションの変更

1. クラスタリングが有効な状態で JBoss EAP 6 を起動する

JBoss EAP 5.x でクラスタリングを有効にするには、次のように **all** プロファイル (またはその派生プロファイル) を使用してサーバーインスタンスを起動する必要がありました。

```
$ EAP5_HOME/bin/run.sh -c all
```

JBoss EAP 6 でクラスタリングを有効にする方法は、サーバーがスタンドアロンであるかまたは管理ドメインで実行されているかによって異なります。

a. 管理対象ドメインで実行されているサーバーに対してクラスタリングを有効にする

ドメインコントローラーを使用して起動したサーバーに対してクラスタリングを有効にするには、**domain.xml** を更新し、**ha** プロファイルと **ha-sockets** ソケットバインディンググループを使用するサーバーグループを指定します。例は次のとおりです。

```
<server-groups>
  <server-group name="main-server-group" profile="ha">
    <jvm name="default">
      <heap size="64m" max-size="512m"/>
    </jvm>
```

```
<socket-binding-group ref="ha-sockets"/>
</server-group>
</server-group>
```

- b. スタンドアロンサーバーに対してクラスタリングを有効にします。

スタンドアロンサーバーに対してクラスタリングを有効にするには、次のように適切な設定ファイルを使用してサーバーを起動します。

```
$ EAP_HOME/bin/standalone.sh --server-config=standalone-ha.xml -
Djboss.node.name=UNIQUE_NODE_NAME
```

2. バインドアドレスを指定する

通常、JBoss EAP 5.x では、以下のように **-b** コマンドライン引数を用いて、クラスタリングに使用するバインドアドレスを指定しました。

```
$ EAP5_HOME/bin/run.sh -c all -b 192.168.0.2
```

JBoss EAP 6 は、**standalone.xml**、**domain.xml**、および **host.xml** ファイルの **<interfaces>** 要素に含まれる IP アドレスおよびインターフェースヘソケットをバインドします。JBoss EAP に同梱される標準的な設定には、2 つのインターフェース設定が含まれています。

```
<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>
</interfaces>
```

これらのインターフェース設定は、**jboss.bind.address.management** および **jboss.bind.address** システムプロパティの値を使用します。これらのシステムプロパティが設定されていないと、デフォルトの **127.0.0.1** が各値に使用されます。

また、サーバーの起動時にバインドアドレスをコマンドライン引数として指定でき、JBoss EAP 6 サーバー設定ファイル内に明示的に定義することも可能です。

- JBoss EAP スタンドアロンサーバーの起動時に、コマンドラインでバインド引数を指定します。

以下の例は、スタンドアロンサーバーでコマンドラインにバインドアドレスを指定する方法を示しています。

```
EAP_HOME/bin/standalone.sh -Djboss.bind.address=127.0.0.1
```



注記

また、**-Djboss.bind.address=127.0.0.1** のショートカットである **-b** 引数を使用することもできます。

```
EAP_HOME/bin/standalone.sh -b=127.0.0.1
```

JBoss EAP 5 の構文形式もサポートされます。

```
EAP_HOME/bin/standalone.sh -b 127.0.0.1
```

-b 引数は **public** インターフェースのみを変更することに注意してください。**management** インターフェースは対象になりません。

- サーバー設定ファイルにバインドアドレスを指定します。

管理対象ドメインで稼働しているサーバーの場合

は、**domain/configuration/host.xml** ファイルでバインドアドレスを指定します。スタンドアロンサーバーの場合は、**standalone-ha.xml** ファイルでバインドアドレスを指定します。

以下の例では、**ha-sockets** ソケットバインディンググループ内のすべてのソケットに対するデフォルトインターフェースとして **public** インターフェースが指定されています。

```
<interfaces>
  <interface name="management">
    <inet-address value="192.168.0.2"/>
  </interface>
  <interface name="public">
    <inet-address value="192.168.0.2"/>
  </interface>
</interfaces>

<socket-binding-groups>
  <socket-binding-group name="ha-sockets" default-
interface="public">
    <!-- ... -->
  </socket-binding-group>
</socket-binding-groups>
```



注記

バインドアドレスを設定ファイルのシステムプロパティとしてではなく、ハードコードされた値として指定する場合は、コマンドライン引数でオーバーライドできません。

3. jvmRoute が mod_jk と mod_proxy をサポートするよう設定する

JBoss EAP 5 では、Web サーバー **jvmRoute** は **server.xml** ファイルのプロパティを使用して設定されていました。JBoss EAP 6 では、**jvmRoute** 属性は、以下のように **instance-id** 属性を使用してサーバー設定ファイルの Web サブシステムで設定されます。

```
<subsystem xmlns="urn:jboss:domain:web:1.1" default-virtual-server="default-host" native="false" instance-id="{JVM_ROUTE_SERVER}">
```

上記の `{JVM_ROUTE_SERVER}` は、`jvmRoute` サーバー ID で置き換える必要があります。

instance-id は、管理コンソールを使用して設定することもできます。

4. マルチキャストアドレスおよびポートを指定する

JBoss EAP 5.x では、以下のようにコマンドライン引数 **-u** を使用して、クラスター内の通信に使用されるマルチキャストアドレスを指定できました。同様に、引数 **-m** を使用してクラスター内の通信に使用されるポートを指定できました。

```
$ EAP5_HOME/bin/run.sh -c all -u 228.11.11.11 -m 45688
```

JBoss EAP 6 では、クラスター内の通信に使用されるマルチキャストアドレスとポートは、以下のように該当する JGroups プロトコルにより参照されたソケットバインディングにより定義されます。

```
<subsystem xmlns="urn:jboss:domain:jgroups:1.0" default-stack="udp">
  <stack name="udp">
    <transport type="UDP" socket-binding="jgroups-udp"/>
    <!-- ... -->
  </stack>
</subsystem>
```

```
<socket-binding-groups>
  <socket-binding-group name="ha-sockets" default-interface="public">
    <!-- ... -->
    <socket-binding name="jgroups-udp" port="55200" multicast-address="228.11.11.11" multicast-port="45688"/>
    <!-- ... -->
  </socket-binding-group>
</socket-binding-groups>
```

コマンドラインでマルチキャストアドレスおよびポートを指定する場合は、マルチキャストアドレスとポートをシステムプロパティーとして定義し、サーバーを起動するときにこれらのプロパティーをコマンドラインで使用できます。以下の例では、**jboss.mcast.addr** は、マルチキャストアドレスの変数名であり、**jboss.mcast.port** はポートの変数名です。

```
<socket-binding name="jgroups-udp" port="55200"
  multicast-address="${jboss.mcast.addr:230.0.0.4}" multicast-
  port="${jboss.mcast.port:45688}"/>
```

次のコマンドライン引数を使用してサーバーを起動できます。

```
$ EAP_HOME/bin/domain.sh -Djboss.mcast.addr=228.11.11.11 -
  Djboss.mcast.port=45688
```

5. 代替のプロトコルスタックを使用する

JBoss EAP 5.x では、**jboss.default.jgroups.stack** システムプロパティを使用してすべてのクラスタリングサービスに使用されるデフォルトのプロトコルスタックを操作できました。

```
$ EAP5_HOME/bin/run.sh -c all -Djboss.default.jgroups.stack=tcp
```

JBoss EAP 6 では、**domain.xml** または **standalone-ha.xml** 内の JGroups サブシステムによってデフォルトのプロトコルスタックが定義されます。

```
<subsystem xmlns="urn:jboss:domain:jgroups:1.0" default-stack="udp">
  <stack name="udp">
    <!-- ... -->
  </stack>
</subsystem>
```

6. バディレプリケーション

JBoss EAP 5.x は JBoss Cache のバディレプリケーションを使用して、クラスターのすべてのインスタンスへのデータレプリケーションを抑制しました。

JBoss EAP 6 ではバディレプリケーションは、Infinispan の分散キャッシュである **DIST** モードに置き換えられました。DIST (分散) モードは強力なクラスタリングモードで、サーバーがクラスターに追加されると Infinispan によって線形にスケールします。サーバーが DIST キャッシングモードを使用するよう設定する方法の例は次のとおりです。

- a. コマンドラインを開き、次のように HA または Full プロファイルのいずれかでサーバーを起動します。

```
EAP_HOME/bin/standalone.sh -c standalone-ha.xml
```

- b. 別のコマンドラインを開き、管理 CLI へ接続します。

- Linux の場合は、コマンドラインで以下を入力します。

```
$ EAP_HOME/bin/jboss-cli.sh --connect
```

- Windows の場合は、コマンドラインで以下を入力します。

```
C:\>EAP_HOME\bin\jboss-cli.bat --connect
```

次の応答が表示されるはずです。

```
Connected to standalone controller at localhost:9999
```

- c. 以下のコマンドを実行します。

```
/subsystem=infinispan/cache-container=web/:write-attribute(name=default-cache,value=dist)
/subsystem=infinispan/cache-container=web/distributed-cache=dist/:write-attribute(name=owners,value=3)
:reload
```

各コマンドの後に、以下の応答が表示されるはずです。

```
"outcome" => "success"
```

これらのコマンドは、次のように **standalone-ha.xml** ファイルの **infinispan** サブシステムにある **web <cache-container>** 設定の **dist <distributed-cache>** 要素を変更します。

```
<cache-container name="web" aliases="standard-session-cache"
  default-cache="dist"
  module="org.jboss.as.clustering.web.infinispan">
  <transport lock-timeout="60000"/>
  <replicated-cache name="repl" mode="ASYNC" batching="true">
    <file-store/>
  </replicated-cache>
  <replicated-cache name="sso" mode="SYNC" batching="true"/>
  <distributed-cache name="dist" owners="3" l1-lifespan="0"
    mode="ASYNC" batching="true">
    <file-store/>
  </distributed-cache>
</cache-container>
```

詳細

は、https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/にある JBoss EAP 6 向け『開発ガイド』の章「Web アプリケーションのクラスター化」を参照してください。

[バグを報告する](#)

3.2.8.2. HA シングルトンの実装

概要

以下の手順は、SingletonService デコレーターでラッピングされ、クラスター全体のシングルトンサービスとして使用されるサービスのデプロイ方法を示しています。このサービスは、クラスターで 1 度だけ開始されるスケジュール済みのタイマーをアクティベートします。

手順3.21 HA シングルトンサービスの実装

1. HA シングルトンサービスアプリケーションを作成します。

シングルトンサービスとしてデプロイされる **SingletonService** デコレーターでラッピングされた **Service** の簡単な例を以下に示します。完全な例は、Red Hat JBoss Enterprise Application Platform 6 に含まれる **cluster-ha-singleton** クイックスタートにあります。このクイックスタートには、アプリケーションをビルドおよびデプロイするための手順がすべて含まれています。

a. サービスを作成します。

以下の一覧はサービスの例になります。

```
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. **Service** をクラスター化されたシングルトンとしてインストールするアクティベーターを作成します。

以下は、**HATimerService** をクラスター化されたシングルトンサービスとしてインストールするサービスアクティベーターの例になります。

```

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()
    ;
}
}

```



注記

上記のコードサンプルは、JBoss EAP プライベート API の一部である **org.jboss.as.clustering.singleton.SingletonService** クラスを使用します。パブリック API は EAP 7 リリースで利用可能になる予定です。その後、プライベートクラスは廃止されますが、6.x のリリースサイクル中は維持され、利用可能です。

c. ServiceActivator ファイルを作成します。

アプリケーションの **resources/META-INF/services/** ディレクトリに **org.jboss.msc.service.ServiceActivator** という名前のファイルを作成します。前の手順で作成した ServiceActivator クラスの完全修飾名が含まれる行を追加します。

```
org.jboss.as.quickstarts.cluster.hasingleton.service.ejb.HATimerServiceActivator
```

d. 使用するタイマーをクラスター全体のシングルトンタイマーとして実装する Singleton Bean を作成します。

この Singleton Bean はリモートインターフェースを持たないようにし、すべてのアプリケーションの別の EJB からローカルインターフェースを参照しないようにする必要があります。これにより、クライアントや他のコンポーネントがルックアップできないようにし、SingletonService がシングルトンを完全に制御するようにします。

i. スケジューラーインターフェースを作成します。

```

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. クラスター全体のシングルトンタイマーを実装する **Singleton Bean** を作成します。

```

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. クラスタリングが有効な状態で各 JBoss EAP 6 インスタンスを起動します。

スタンドアロンサーバーに対してクラスタリングを有効にするには、各インスタンスの一意的なノード名とポートオフセットを使用して、各サーバーを **HA** プロファイルで起動する必要があります。

- Linux では、次のコマンド構文を使用してサーバーを起動します。

```

EAP_HOME/bin/standalone.sh --server-config=standalone-ha.xml -
Djboss.node.name=UNIQUE_NODE_NAME -Djboss.socket.binding.port-
offset=PORT_OFFSET

```

例3.1 Linux で複数のスタンドアロンサーバーを起動

```

$ 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

```

- Microsoft Windows では、次のコマンド構文を使用してサーバーを起動します。

```

EAP_HOME\bin\standalone.bat --server-config=standalone-ha.xml -
Djboss.node.name=UNIQUE_NODE_NAME -Djboss.socket.binding.port-
offset=PORT_OFFSET

```

例3.2 Microsoft Windows で複数のスタンドアロンサーバーを起動

```

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

```



注記

コマンドライン引数を使用したくない場合は、**standalone-ha.xml** ファイルを設定し、各サーバーインスタンスが個別のインターフェースでバインドするようにします。

3. アプリケーションをサーバーにデプロイします。

以下の Maven コマンドは、デフォルトのポートで稼働しているスタンドアロンサーバーへアプリケーションをデプロイします。

```
mvn clean install jboss-as:deploy
```

追加のサーバーをデプロイするには、サーバー名を渡します。別のホストにある場合は、コマンドラインでホスト名とポート番号を渡します。

```
mvn clean package jboss-as:deploy -Djboss-as.hostname=localhost -  
Djboss-as.port=10099
```

Maven の設定とデプロイメントの詳細は、JBoss EAP 6 に含まれる **cluster-ha-singleton** クイックスタートを参照してください。

[バグを報告する](#)

3.2.9. サービススタイルデプロイメントの変更

3.2.9.1. サービススタイルデプロイメントを使用するアプリケーションの更新

概要

JBoss EAP 6 はサービススタイル記述子を使用しないようになりましたが、できる限り変更がない状態でコンテナはサービススタイルデプロイメントをサポートします。そのため、JBoss EAP 5.x アプリケーションの **jboss-service.xml** または **jboss-beans.xml** デプロイメント記述子を使用した場合、JBoss EAP 6 へ変更をほとんどまたは全く加えなくても実行できるはずです。継続してファイルを EAR や SAR にパッケージ化することが可能ですが、ファイルを直接 deployments ディレクトリーに置くこともできます。スタンドアロンサーバーを実行している場合、deployments ディレクトリーは **EAP_HOME/standalone/deployments/** になります。管理ドメインを実行している場合は、コンソールまたは CLI を使用してアプリケーションをデプロイする必要があります。

[バグを報告する](#)

3.2.10. リモート呼び出しの変更

3.2.10.1. JBoss EAP 5 にデプロイされ、JBoss EAP 6 へリモート呼び出しを行うアプリケーションの移行

概要

JBoss EAP 5 では、EJB リモートインターフェースはデフォルトで JNDI にてバインドされ、ローカルインターフェースの場合は「ejbName/local」、リモートインターフェースの場合は「ejbName/remote」という名前でした。クライアントアプリケーションは「ejbName/remote」を使用して Bean をルックアップしました。

JBoss EAP 6 では、呼び出しの実行に新しい EJB クライアント API が導入されました。しかし、新しい API を使用するようコードを書き直したくない場合、以下の構文で既存のコードを編集すると、EJB へのリモートアクセスに **ejb:BEAN_REFERENCE** を使用できます。

以下は、ステートレス Bean 用の **ejb:BEAN_REFERENCE** の構文になります。

```
ejb:<app-name>/<module-name>/<distinct-name>/<bean-name>!<fully-qualified-  
classname-of-the-remote-interface>
```

以下は、ステートフル Bean 用の `ejb:BEAN_REFERENCE` の構文になります。

```
ejb:<app-name>/<module-name>/<distinct-name>/<bean-name>!<fully-qualified-
classname-of-the-remote-interface>?stateful
```

上記の構文で置き換える必要のある値は次のとおりです。

- **<app-name>** - デプロイされた EJB のアプリケーション名。通常、.ear 接尾辞を除いた ear 名になりますが、application.xml ファイルで名前が上書きされる場合があります。アプリケーションが .ear としてデプロイされていない場合、この値は空の文字列となります。この例は EAR としてデプロイされていないことを仮定します。
- **<module-name>** - サーバー上のデプロイされた EJB のモジュール名。通常、.jar サフィックスを除いた EJB デプロイメントの jar 名になりますが、ejb-jar.xml を使用して名前が上書きされる場合があります。この例では、EJB が jboss-ejb-remote-app.jar にデプロイされていることを仮定しているため、モジュール名は jboss-ejb-remote-app になります。
- **<distinct-name>** - EJB の任意の distinct name です。この例では distinct name は使用しないため、空の文字列を使用します。
- **<bean-name>** - デフォルトでは Bean 実装クラスの簡単なクラス名になります。
- **<fully-qualified-classname-of-the-remote-interface>** - リモートビューの完全修飾クラス名。

クライアントコードの更新

次のステートレス EJB を JBoss EAP 6 サーバーにデプロイしたことにします。これにより、Bean のリモートビューが公開されます。

```
@Stateless
@Remote(RemoteCalculator.class)
public class CalculatorBean implements RemoteCalculator {

    @Override
    public int add(int a, int b) {
        return a + b;
    }

    @Override
    public int subtract(int a, int b) {
        return a - b;
    }
}
```

JBoss EAP 5 では、クライアント EJB のルックアップと呼び出しが次のようにコード化されていました。

```
InitialContext ctx = new InitialContext();
RemoteCalculator calculator = (RemoteCalculator)
ctx.lookup("CalculatorBean/remote");
int a = 204;
int b = 340;
int sum = calculator.add(a, b);
```

JBoss EAP 6 では前述の情報を使用して、クライアントのルックアップと呼び出しが次のようにコード化されます。

```
final Hashtable jndiProperties = new Hashtable();
jndiProperties.put(Context.URL_PKG_PREFIXES,
"org.jboss.ejb.client.naming");
final Context context = new InitialContext(jndiProperties);
final String appName = "";
final String moduleName = "jboss-ejb-remote-app";
final String distinctName = "";
final String beanName = CalculatorBean.class.getSimpleName();
final String viewClassName = RemoteCalculator.class.getName();
final RemoteCalculator statelessRemoteCalculator = (RemoteCalculator)
context.lookup("ejb:" + appName + "/" + moduleName + "/" + distinctName +
"/" + beanName + "!" + viewClassName);

int a = 204;
int b = 340;
int sum = statelessRemoteCalculator.add(a, b);
```

クライアントがステートフル EJB にアクセスしている場合、次のようにコンテキストルックアップの最後に "?stateful" を追加する必要があります。

```
final RemoteCalculator statefulRemoteCalculator = (RemoteCalculator)
context.lookup("ejb:" + appName + "/" + moduleName + "/" + distinctName +
"/" + beanName + "!" + viewClassName + "?stateful")
```

サーバーおよびクライアントコードを含む完全な作業例はクイックスタートにあります。詳細については、https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/にある JBoss EAP 6 向け『開発ガイド』の章「アプリケーションの開発」に記載された「クイックスタートチュートリアルの確認」を参照してください。

JNDI を使用したリモート呼び出しの詳細については、「[JNDI を使用したリモートでのセッション Bean の呼び出し](#)」を参照してください。

[バグを報告する](#)

3.2.10.2. JNDI を使用したリモートでのセッション Bean の呼び出し

このタスクは、JNDI を使用してセッション Bean の呼び出すリモートクライアントへサポートを追加する方法を説明します。Maven を使用してプロジェクトがビルドされていることが前提となります。

ejb-remote クイックスタートには、この機能のデモを行う Maven プロジェクトが含まれています。このクイックスタートには、デプロイするセッション Bean のプロジェクトとリモートクライアントのプロジェクトの両方が含まれています。下記のコード例はリモートクライアントのプロジェクトから引用されています。

このタスクでは、セッション Bean に認証の必要がないことが前提となっています。

要件

始める前に、次の前提条件を満たしている必要があります。

- Maven プロジェクトが作成され、使用できる状態です。

- JBoss EAP 6 の Maven リポジトリがすでに追加されています。
- 呼び出しするセッション Bean がすでにデプロイされています。
- デプロイされたセッション Bean がリモートビジネスインターフェースを実装します。
- セッション Bean のリモートビジネスインターフェースは Maven 依存関係として使用できます。リモートビジネスインターフェースが JAR ファイルとしてのみ使用できる場合は、JAR をアーティファクトとして Maven リポジトリに追加することが推奨されます。手順については、<http://maven.apache.org/plugins/maven-install-plugin/usage.html> にある Maven ドキュメントの **install:install-file** ゴールを参照してください。
- セッション Bean をホストするサーバーのホスト名と JNDI ポートを覚えておく必要があります。

リモートクライアントよりセッション Bean を呼び出すには、最初にプロジェクトを適切に設定する必要があります。

手順3.22 セッション Bean のリモート呼び出しに対する Maven プロジェクト設定の追加

1. 必要なプロジェクト依存関係の追加

必要な依存関係が含まれるようにするため、プロジェクトの **pom.xml** を更新する必要があります。

2. **jboss-ejb-client.properties** ファイルの追加

JBoss EJB クライアント API は、JNDI サービスの接続情報が含まれる **jboss-ejb-client.properties** という名前のプロジェクトのルートにファイルがあることを想定します。このファイルを以下の内容と共にプロジェクトの **src/main/resources/** ディレクトリに追加します。

```
# In the following line, set SSL_ENABLED to true for SSL
remote.connectionprovider.create.options.org.xnio.Options.SSL_ENABLED=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_STARTTLS=true
remote.connection.default.host=localhost
remote.connection.default.port = 4447
remote.connection.default.connect.options.org.xnio.Options.SASL_POLICY_NOANONYMOUS=false
# Add any of the following SASL options if required
#
remote.connection.default.connect.options.org.xnio.Options.SASL_POLICY_NOANONYMOUS=false
#
remote.connection.default.connect.options.org.xnio.Options.SASL_POLICY_NOPLAINTEXT=false
#
remote.connection.default.connect.options.org.xnio.Options.SASL_DISALLOWED_MECHANISMS=JBOSS-LOCAL-USER
```

ホスト名とポートを変更してサーバーと一致するようにします。**4447** がデフォルトのポート番号です。安全な接続の場合、**SSL_ENABLED** 行を **true** に設定し、**SSL_STARTTLS** 行をアン

コメントします。コンテナ内のリモートシングインターフェースは同じポートを使用して安全な接続と安全でない接続をサポートします。

3. リモートビジネスインターフェースの依存関係の追加

セッション Bean のリモートビジネスインターフェースに対する `pom.xml` に Maven の依存関係を追加します。

```
<dependency>
  <groupId>org.jboss.as.quickstarts</groupId>
  <artifactId>jboss-ejb-remote-server-side</artifactId>
  <type>ejb-client</type>
  <version>${project.version}</version>
</dependency>
```

これでプロジェクトが適切に設定されたため、コードを追加してセッション Bean へのアクセスや呼び出しが可能になりました。

手順3.23 JNDI を使用した Bean プロキシの取得および Bean のメソッドの呼び出し

1. チェック例外の処理

次のコードに使用されるメソッドの2つ (`InitialContext()` および `lookup()`) は、タイプ `javax.naming.NamingException` のチェック済み例外を持っています。これらのメソッド呼び出しは、`NamingException` をキャッチする try/catch ブロックか、`NamingException` のスローが宣言されたメソッドでエンクローズされる必要があります。`ejb-remote` クイックスタートでは、2つ目の方法を使用します。

2. JNDI コンテキストの作成

JNDI コンテキストオブジェクトはサーバーよりリソースを要求するメカニズムを提供します。次のコードを使用して JNDI コンテキストを作成します。

```
final Hashtable jndiProperties = new Hashtable();
jndiProperties.put(Context.URL_PKG_PREFIXES,
  "org.jboss.ejb.client.naming");
final Context context = new InitialContext(jndiProperties);
```

JNDI サービスの接続プロパティは `jboss-ejb-client.properties` ファイルから読み取られます。

3. JNDI コンテキストの `lookup()` メソッドを使用した Bean プロキシの取得

Bean プロキシの `lookup()` メソッドを呼び出し、必要なセッション Bean の JNDI 名へ渡します。これにより、呼び出したいメソッドが含まれるリモートビジネスインターフェースのタイプへキャストされなければならないオブジェクトが返されます。

```
final RemoteCalculator statelessRemoteCalculator =
  (RemoteCalculator) context.lookup(
    "ejb:/jboss-ejb-remote-server-side//CalculatorBean!" +
    RemoteCalculator.class.getName());
```

セッション Bean の JNDI 名は特別な構文によって定義されます。詳細は、[「EJB JNDI の名前に関する参考資料」](#) を参照してください。

4. 呼び出しメソッド

プロキシ Bean オブジェクトを取得したため、リモートビジネスインターフェースに含まれるすべてのメソッドを呼び出しできます。

```
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);
```

メソッド呼び出し要求が実行されるサーバー上で、プロキシ Bean がメソッド呼び出し要求をセッション Bean へ渡します。結果はプロキシ Bean へ返され、プロキシ Bean によって結果が呼び出し側へ返されます。プロキシ Bean とリモートセッション Bean 間の通信は呼び出し側に透過的です。

これで、Maven プロジェクトを設定してリモートサーバー上で呼び出しを行うセッション Bean をサポートし、JNDI を使用してサーバーより読み出したプロキシ Bean を使用してセッション Bean メソッドを呼び出すコードを作成できるようになりました。

バグを報告する

3.2.10.3. EJB JNDI の名前に関する参考資料

セッション Bean の JNDI ルックアップ名の構文は次のとおりです。

```
ejb:<appName>/<moduleName>/<distinctName>/<beanName>!<viewClassName>?
stateful
```

<appName>

セッション Bean の JAR ファイルがエンタープライズアーカイブ (EAR) 内にデプロイされた場合、EAR の名前になります。デフォルトでは、ファイル名から **.ear** 接尾辞を除いたものが EAR の名前になります。また、アプリケーション名を **application.xml** ファイルで上書きすることも可能です。セッション Bean が EAR にデプロイされていない場合は空白のままにしておきます。

<moduleName>

モジュール名はセッション Bean がデプロイされた JAR ファイルの名前になります。デフォルトでは、ファイル名から **.jar** 接尾辞を除いたものが JAR ファイルの名前になります。また、モジュール名を JAR の **ejb-jar.xml** ファイルで上書きすることも可能です。

<distinctName>

JBoss EAP 6 では、各デプロイメントが任意の個別名を指定することができます。デプロイメントの個別名がない場合は空白のままにしておきます。

<beanName>

Bean 名は呼び出されるセッション Bean のクラス名です。

<viewClassName>

ビュークラス名はリモートインターフェースの完全修飾クラス名です。インターフェースのパッケージ名が含まれます。

?stateful

JNDI 名がステートフルセッション Bean を参照する時に **?stateful** 接尾辞が必要となります。他の Bean タイプでは含まれていません。

[バグを報告する](#)

3.2.11. EJB 2.x の変更

3.2.11.1. EJB 2.x を使用するアプリケーションの更新

JBoss EAP 6 はオープン標準で構築され、Java Enterprise Edition 6 仕様に準拠しています。アプリケーションサーバーは EJB 2.x のサポートを提供しますが、この仕様以降の機能をサポートしない可能性があります。Java EE 7 仕様では EJB 2.x はオプションとなっているため、アプリケーションコードを EJB 3.x 仕様向けに書き直すことが強く推奨されます。

EJB 2.x コードを移行する場合、JBoss EAP 6 を実行するために変更を加える必要があることがほとんどです。本トピックでは、JBoss EAP 6 上で EJB 2.x を実行するために必要となる変更の一部を取り上げます。

JBoss EAP 6 上で EJB 2.x を実行するために必要な設定変更

Full プロファイルでサーバーを起動

EJB 2.x の CMP (Container Managed Persistence) Bean には Java Enterprise Edition 6 Full Profile が必要です。このプロファイルには、CMP EJB を実行するために必要な設定要素が含まれています。

この設定プロファイルには **org.jboss.as.cmp** 拡張モジュールが含まれています。

```
<extensions>
...
<extension module="org.jboss.as.cmp"/>
...
</extensions>
```

さらに、**cmp** サブシステムも含まれています。

```
<profiles>
...
<subsystem xmlns="urn:jboss:domain:cmp:1.1"/>
...
</profiles>
```

Full プロファイルで JBoss EAP 6 スタンドアロンサーバーを起動するには、サーバーの起動時にコマンドラインで **-c standalone-full.xml** または **-c standalone-full-ha.xml** 引数を渡します。

コンテナ設定はサポートされません

以前のバージョンの JBoss EAP では、CMP エンティティおよび他の Bean の異なるコンテナを設定し、**jboss.xml** アプリケーションデプロイメント記述子ファイル内に参照を設定することが可能でした。たとえば、通常は SLSB とセッション Bean の設定は異なりました。

JBoss EAP 6.x では、標準のコンテナで EJB 2 エンティティ Bean を使用できますが、他のコンテナ設定はサポートされなくなりました。EJB2 のステートフルセッション Bean (SFSB)、ステートレスセッション Bean (SLSB)、およびメッセージ駆動型 Bean (MDB) を EJB 3 へ移行し、

CMP (Container-Managed Persistence) および BMP (Bean-Managed Persistence) エンティティ Bean が EJB 3 の仕様どおりに Java 永続 API (JPA) を使用することが推奨されます。

JBoss EAP 6 のデフォルトのコンテナ設定には、EJB 2 CMP Bean に対する変更が複数含まれています。

- 悲観的ロックはデフォルトで有効になっています。これにより、デッドロックが発生する可能性があります。
- JBoss EAP 5.x の CMP レイヤーに存在したデッドロック検出コードは JBoss EAP 6 には存在しません。

JBoss EAP 5.x では、キャッシング、プーリング、**commit-options**、およびインターセプタースタックをカスタマイズできましたが、JBoss EAP 6 ではカスタマイズできません。**commit-option C** を持つ **Instance Per Transaction** ポリシーと似た実装のみがあります。CMP2.x と互換性のある JDBC ベースの永続性マネージャーを使う **cmp2.x jdbc2 pm** エンティティ Bean コンテナ設定を使用するアプリケーションを移行する場合は、パフォーマンスに影響します。このコンテナはパフォーマンスに対して最適化されていました。アプリケーションを移行する前にこれらのエンティティを EJB 3 に移行することが推奨されます。

サーバー側インターセプター設定

JBoss EAP 6 は、**@Interceptors** および **@AroundInvoke** を使用して標準の Java EE **Interceptor** をサポートします。しかし、セキュリティ外部またはトランザクション外部の操作は許可されません。

以前のバージョンの JBoss EAP では、各 EJB 呼び出しに対してカスタムインターセプターを指定するためにインターセプタースタックを変更できました。これは通常、セキュリティチェック、トランザクションチェック、または作成の前にカスタマイズされたセキュリティまたはリトライメカニズムを実装するために使用されました。JBoss EAP 6.1 には、同様の機能を提供するコンテナインターセプターが導入されました。コンテナインターセプターの詳細は、JBoss EAP 『開発ガイド』の「コンテナインターセプター」の章を参照してください。

Java EE 仕様に準拠しながら、トランザクションのコミットフェーズ中および前後で制御を強化する別の方法には、Transaction Synchronization Registry (トランザクション同期レジストリー) を使用してリスナーを追加する方法があります。

以下の方法の 1 つを使用してリソースを読み出しできます。

- **InitialContext** の使用

```
TransactionSynchronizationRegistry tsr =
    (TransactionSynchronizationRegistry)
        new
    InitialContext().lookup("java:jboss/TransactionSynchronizationRegistry");
    tsr.registerInterposedSynchronization(new MyTxCallback());
```

- インジェクションの使用

```
@Resource(mappedName =
    "java:comp/TransactionSynchronizationRegistry")
TransactionSynchronizationRegistry tsr;
...
tsr.registerInterposedSynchronization(new MyTxCallback());
```

コールバックルーティングは、**javax.transaction.Synchronization** インターフェースを実装する必要があります。トランザクションがコミットまたはロールバックする前に、**beforeCompletion{}** メソッドを使用してチェックを実行します。このメソッドから **RuntimeException** が発生した場合、トランザクションがロールバックされ、クライアントには **EJBTransactionRolledbackException** が報告されます。XA トランザクションの場合、XA コントラクトにしたがってすべてのリソースがロールバックされます。また、**afterCompletion(int txStatus)** を使用して、有効なビジネスロジックがトランザクションの状態に依存するようにすることも可能です。このメソッドから **RuntimeException** が発生した場合、トランザクションはコミットまたはロールバックされた以前の状態を維持し、クライアントには報告されません。トランザクションマネージャーのみがサーバーログファイル内で警告を表示します。

クライアント側インターセプターのサーバー側設定

以前のバージョンの JBoss EAP では、サーバー設定内でクライアントインターセプターを設定し、クライアント API でクラスのみを提供することが可能でした。

しかし、JBoss EAP 6 ではこれが不可能になりました。クライアントプロキシがサーバー側で作成されなくなり、ルックアップ後にクライアントへ送信されなくなったためです。JBoss EAP 6 ではプロキシはクライアント側で作成されます。この最適化は、ルックアップのサーバー呼び出しや、クラスのアップロードが発生しないようにします。

エンティティ Bean プールの設定

JBoss EAP 6 では、エンティティ Bean プールの設定は推奨されません。この設定は、**<strict-max-pool>** 要素の設定に限定されるため、プールが小さすぎて結果セットのエントリをすべてロードできないと、デッドロックなどの問題が発生することがあります。エンティティ Bean は初期化中に大きなライフサイクルメソッドを持たないため、インスタンスおよび周囲のコンテナを作成する速度は、プールされたエンティティ Bean インスタンスを使用する場合と変わりません。

jboss.xml デプロイメント記述子ファイルの置き換え

jboss.xml デプロイメント記述子は **jboss-ejb3.xml** デプロイメント記述子に置き換えられました。このファイルは、Java Enterprise Edition (EE) によって定義された **ejb-jar.xml** デプロイメント記述子によって提供される機能を上書きおよび追加するために使用されます。**jboss-ejb3.xml** ファイルは **jboss.xml** との互換性がなく、**jboss.xml** はデプロイメントで無視されます。

たとえば、JBoss EAP の以前のリリースでは **ejb-jar.xml** ファイルで **<resource-ref>** を定義する場合に **jboss.xml** ファイルに JNDI 名の対応するリソース定義が必要でした。XDroplet が自動的にこれらのデプロイメント記述子ファイルを作成しました。JBoss EAP 6 では、JNDI マッピング情報が **jboss-ejb3.xml** ファイルで定義されるようになりました。以下のようにデータソースが Java ソースに定義されていることを仮定します。

```
DataSource ds1 = (DataSource) new
InitialContext().lookup("java:comp/env/jdbc/Resource1");
DataSource ds2 = (DataSource) new
InitialContext().lookup("java:comp/env/jdbc/Resource2");
```

ejb-jar.xml は以下のリソース参照を定義します。

```
<resource-ref >
  <res-ref-name>jdbc/Resource1</res-ref-name>
  <res-type>javax.sql.DataSource</res-type>
  <res-auth>Container</res-auth>
```

```

</resource-ref>
<resource-ref>
  <res-ref-name>java:comp/env/jdbc/Resource2</res-ref-name>
  <res-type>javax.sql.DataSource</res-type>
  <res-auth>Container</res-auth>
</resource-ref>

```

jboss-ejb3.xml ファイルは、以下の XML 構文を使用して JNDI 名を参照へマップします。

```

<resource-ref>
  <res-ref-name>jdbc/Resource1</res-ref-name>
  <jndi-name>java:jboss/datasources/ExampleDS</jndi-name>
</resource-ref>
<resource-ref>
  <res-ref-name>java:comp/env/jdbc/Resource2</res-ref-name>
  <jndi-name>java:jboss/datasources/ExampleDS</jndi-name>
</resource-ref>

```

JBoss EAP 6 には、JBoss EAP 5.x の **jboss.xml** ファイルで使用できた設定オプションの一部が実装されていません。以下のリストは、一般的な **jboss.xml** ファイルの属性と、JBoss EAP 6 で代替の方法があるかどうかを示しています。

- **method-attribute** 要素は個別のエンティティーおよびセッション Bean メソッドを設定するために使用されました。
 - **read-only** および **idempotent** 設定オプションは JBoss EAP 6 に移植されませんでした。
 - **transaction-timeout** オプションは **jboss-ejb3.xml** ファイルで設定されるようになりました。
- **missing-method-permission-exclude-mode** 属性は、セキュア化された Bean に明示的なセキュリティメタデータを実装せずにメソッドの挙動を変更しました。現在 JBoss EAP 6 では、**@RolesAllowed** アノテーションが存在しないと **@PermitAll** と同様に処理されます。

DataSource タイプマッピングの設定

以前のバージョンの JBoss EAP では、***-ds.xml** データソースデプロイメント設定ファイル内にデータソースタイプマッピングを設定できました。

JBoss EAP 6 では、この設定を **jbosscomp-jdbc.xml** デプロイメント記述子ファイルで行う必要があります。

```

<defaults>
  <datasource-mapping>mySQL</datasource-mapping>
  <create-table>true</create-table>
  ....
</defaults>

```

以前のバージョンの JBoss EAP では、カスタマイズされたマッピングは **standardjbosscomp-jdbc.xml** ファイルで行われました。このファイルは使用できなくなり、マッピングは **jbosscomp-jdbc.xml** デプロイメント記述子ファイルで行われるようになりました。

CMP (Container Managed Persistence) および CMR (Container Managed Relationship) に関するその他の変更

CMR (Container Managed Relationship) イテレーターおよびコレクションの変更

以前のリリースの JBoss EAP では、`cmp2.x jdbc2 pm` コンテナなどの一部のコンテナが CMR コレクションを繰り返すことが可能で、関係を削除または追加できました。JBoss EAP 6 ではコンテナ設定はサポートされていないため、これが不可能になりました。アプリケーションコードで同じ機能を実現する方法については、カスタマーポータルサポート - ナレッジ - ソリューションに記載されている [EJB2.1 Finder for CMP entities with relations \(CMR\) returns duplicates in EAP6](#) を参照してください。

ファインダーに対する CMR (Container Managed Relationship) の重複エントリー

以前のバージョンの JBoss EAP では、異なる永続性ストラテジーを使用する別の CMP コンテナを選択できませんでした。JBoss EAP 5.x の `cmp2.x jdbc2 pm` コンテナは最適化された **SQL-92** を使用して、ファインダーに対して最適化された LEFT OUTER JOIN 構文を生成しました。JBoss EAP 6.x は CMP および CMR の標準的なコンテナのみをサポートするため、実装にはこれらの最適化が含まれていません。結果セットのデカルト積を避けるため、ファインダーの **SELECT** ステートメントにキーワード **DISTINCT** が含まれるようにする必要があります。詳細は、カスタマーポータルサポート - ナレッジ - ソリューションに記載されている [EJB2.1 Finder for CMP entities with relations \(CMR\) returns duplicates in EAP6](#) を参照してください。

CMP エンティティー Bean に対するカスケード削除のデフォルト変更

カスケード削除のデフォルト値が **false** に変更されました。これにより、JBoss EAP 6 では削除に失敗することがあります。エンティティーの関係が **cascade-delete** とマークされた場合、`jbosscmp-jdbc.xml` ファイルで **batch-cascade-delete** を明示的に **true** に設定する必要があります。詳細は、カスタマーポータルサポート - ナレッジ - ソリューションに記載されている [cascade delete fail for EJB2 CMP Entities after migration to EAP6](#) を参照してください。

カスタムフィールドの CMP カスタムマッパー

JDBCParameterSetter、**JDBCResultSetReader**、**Mapper** などのカスタムマッパークラスを JBoss EAP 5.x アプリケーションで使用した場合、アプリケーションを JBoss EAP 6 にデプロイすると **java.lang.ClassNotFoundException** が発生することがあります。これは、インターフェースのパッケージ名が `org.jboss.ejb.plugins.cmp.jdbc.Mapper` から `org.jboss.as.cmp.jdbc.Mapper` に変更になったためです。詳細は、カスタマーポータルサポート - ナレッジ - ソリューションに記載されている [How to use Field mapping for custom classes in an EJB2 CMP application in EAP6](#) を参照してください。

エンティティーコマンドを使用した主キーの生成

Sequence や **Auto-increment** など、JBoss EAP 5 アプリケーションが **entity-commands** を使用して主キーを生成する場合、アプリケーションを JBoss EAP 6 に移行すると **JDBCOracleSequenceCreateCommand** クラスに対して **ClassNotFoundException** が発生することがあります。これは、クラスパッケージが `org.jboss.ejb.plugins.cmp.jdbc` から `org.jboss.as.cmp.jdbc.keygen` に変更になったためです。JBoss EAP 6 アプリケーションでこのクラスを使用する場合、`EAP_HOME/modules/system/layers/base/org/jboss/as/cmp` モジュール上に依存関係を追加する必要があります。

アプリケーションの変更

新しい JNDI 名前空間ルールを使用するためのコードの変更

EJB 3.0 と同様に、EJB 2.x でも完全な JNDI 接頭辞を使用する必要があります。新しい JNDI 名前空間ルールやコード例の詳細は、[「アプリケーションの JNDI 名前空間名の更新」](#)を参照してください。

以前のリリースから JNDI 名前空間を更新する方法を表す例は [「以前のリリースでの JNDI 名前空間の例、および JBoss EAP 6 での名前空間の指定方法」](#)にあります。

jboss-web.xml ファイル記述子の変更

各 `<ejb-ref>` に対する `<jndi-name>` を変更し、新しい JNDI 完全修飾ルックアップ形式を使用するようにします。

XDoclet を使用した内部ローカルインターフェースの JNDI 名へのマッピング

EJB 2 では、**Locator** パターンを使用した Bean のルックアップが一般的でした。アプリケーションコードを変更せずに、アプリケーションでこのパターンを使用する場合、[XDoclet](#) を使用して新しい JNDI 名のマップを生成できます。

通常、XDoclet アノテーションは以下のようになります。

```
@ejb.bean name="UserAttribute" display-name="UserAttribute" local-jndi-name="ejb21/UserAttributeEntity" view-type="local" type="CMP" cmp-version="2.x" primkey-field="id"
```

上例の JNDI 名 **ejb21/UserAttributeEntity** は、JBoss EAP 6 では無効です。サーバー設定の **naming** サブシステムと XDoclet のパッチを使用して、この名前を有効な JNDI 名へマッピングします。

前述の「カスタムフィールドの CMP カスタムマッパー」にしたがって、カスタマイズされたマッパーを作成できます。または、以下の手順どおりにコードを変更できます。

手順3.24 XDoclet で生成されたコードの変更と naming サブシステムの使用

1. **ejb-module.jar** にある XDoclet の **lookup.xdt** テンプレートを展開し、次のように **lookupHome** の **lookup()** を編集します。

```
private static Object lookupHome(java.util.Hashtable environment,
String jndiName, Class narrowTo) throws
javax.naming.NamingException {
    // Obtain initial context
    javax.naming.InitialContext initialContext = new
javax.naming.InitialContext(environment);
    try {
        // Replace the existing lookup
        // Object objRef = initialContext.lookup(jndiName);
        // This is the new mapped lookup
        Object objRef;
        try {
            // try JBoss EAP mapping
            objRef = initialContext.lookup("global/"+jndiName);
        } catch (java.lang.Exception e) {
            objRef = initialContext.lookup(jndiName);
        }
        // only narrow if necessary
        if (java.rmi.Remote.class.isAssignableFrom(narrowTo))
            return javax.rmi.PortableRemoteObject.narrow(objRef,
```

```

        narrowTo);
        else
            return objRef;
    } finally {
        initialContext.close();
    }
}

```

2. Ant を実行し、変更された **lookup.xdt** を **ejbdoclet** タスクに使用するようテンプレート属性を設定します。
3. サーバー設定ファイルの **naming** サブシステムを編集し、古い JNDI 名を新しく有効な JNDI 名へマッピングします。

```

<subsystem xmlns="urn:jboss:domain:naming:1.2">
    <bindings>
        <lookup name="java:global/ejb21/UserAttributeEntity"
lookup="java:global/ejb2CMP/ejb/UserAttribute!de.wfink.ejb21.cmp.c
mr.UserAttributeLocalHome"/>
    </bindings>
    <remote-naming/>
</subsystem>

```

廃止されたファイル

JBoss EAP 6 ではサポートされないファイルは次のとおりです。

jboss.xml

jboss.xml デプロイメント記述子ファイルはサポートされなくなり、デプロイされたアーカイブに含まれていると無視されます。

standardjbosscmp-jdbc.xml

standardjbosscmp-jdbc.xml 設定ファイルはサポートされません。この設定情報は、**org.jboss.as.cmp** モジュールに含まれるようになり、カスタマイズ不可能になりました。

standardjboss.xml

standardjboss.xml 設定ファイルはサポートされません。この設定情報は、スタンドアロンサーバーを実行する場合は **standalone.xml** ファイルに含まれ、管理対象ドメインで実行される場合は **domain.xml** に含まれるようになりました。

バグを報告する

3.2.12. JBoss AOP 変更

3.2.12.1. JBoss AOP を使用するアプリケーションの更新

JBoss AOP (Aspect Oriented Programming) は JBoss EAP 6 には含まれていません。以前のリリースでは、JBoss AOP は EJB コンテナによって使用されていましたが、JBoss EAP 6 では EJB コンテナは新しいメカニズムを使用します。アプリケーションが JBoss AOP を使用する場合、次のようにアプリ

ケーションコードを変更する必要があります。

アプリケーションのリファクタリング

- 以前、**ejb3-interceptors-aop.xml** ファイルで行われた標準的な EJB3 設定は、サーバー設定ファイルで設定されるようになりました。スタンドアロンサーバーの場合、このファイルは **standalone/configuration/standalone-full.xml** ファイルになります。サーバーが管理ドメインで実行されている場合は **domain/configuration/domain.xml** ファイルになります。
- サーバー側の AOP インターセプターが標準の Java EE **Interceptor** を使用するよう変更する必要があります。コンテナインターセプターの詳細や、アプリケーションでクライアント側インターセプターを使用する方法については、https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/ にある JBoss EAP 6 向け『開発ガイド』の章「コンテナインターセプター」を参照してください。

JBoss AOP ライブラリーの使用

- コードをリファクタリングできない場合は、JBoss AOP ライブラリーのコピーを取得し、そのコピーとアプリケーションをバンドルできます。AOP ライブラリーは JBoss EAP 6 で動作することがありますが、デプロイされません。手動でデプロイするには、サーバーの起動時にコマンドライン引数 **-Djboss.aop.path=PATH_TO_AOP_CONFIG** を使用します。



注記

JBoss AOP ライブラリーは JBoss EAP 6 で動作することがありますが、この設定はサポートされません。

[バグを報告する](#)

3.2.13. Seam 2.2 アプリケーションの移行

3.2.13.1. Seam 2.2 アーカイブの JBoss EAP 6 への移行

概要

Seam 2.2 アプリケーションを移行する際、データソースを設定し、モジュール依存関係を指定する必要があります。また、JBoss EAP 6 に同梱されないアーカイブにアプリケーションの依存関係があるかを判断し、依存している JAR をアプリケーションの **lib/** ディレクトリーにコピーする必要があります。



重要

Seam 2.2 で Hibernate を直接使用するアプリケーションは、アプリケーション内部にパッケージ化された Hibernate 3 のバージョンを使用することがあります。JBoss EAP 6 の org.hibernate モジュールを介して提供される Hibernate 4 は、Seam 2.2 によってサポートされません。この例の目的は、最初の手順として JBoss EAP 6 でアプリケーションを実行させることです。Hibernate 3 を Seam 2.2 アプリケーションでパッケージ化する設定はサポートされないことに注意してください。

手順3.25 Seam 2.2 アーカイブの移行

1. データソース設定を更新する

一部の Seam 2.2 の例は、**java:/ExampleDS** という名前のデフォルトの JDBC データソース

を使用します。このデフォルトデータソースは JBoss EAP 6 では **java:jboss/datasources/ExampleDS** に変更になりました。例のデータベースがアプリケーションによって使用される場合、以下の方法の 1 つ実行します。

- JBoss EAP 6 に同梱されるサンプルデータベースを使用する場合は、既存の **jta-data-source** 要素をサンプルデータベースのデータソース JNDI 名に置き換えるよう **META-INF/persistence.xml** ファイルを変更します。

```
<!-- <jta-data-source>java:/ExampleDS</jta-data-source> -->
<jta-data-source>java:jboss/datasources/ExampleDS</jta-data-source>
```

- 既存のデータベースを維持したい場合は、データソースの定義を **EAP_HOME/standalone/configuration/standalone.xml** ファイルに追加することができます。



重要

変更がサーバーの再起動後も維持されるようにするには、サーバー設定ファイルの編集前にサーバーを停止する必要があります。

JBoss EAP 6 で定義されたデフォルトの HSQL データソースのコピーは以下のとおりです。

```
<datasource name="ExampleDS" jndi-name="java:/ExampleDS"
enabled="true" jta="true" use-java-context="true" use-ccm="true">
  <connection-url>jdbc:h2:mem:test;DB_CLOSE_DELAY=-1</connection-url>
  <driver>h2</driver>
  <security>
    <user-name>sa</user-name>
    <password>sa</password>
  </security>
</datasource>
```

- 管理 CLI コマンドラインインターフェースを使用してデータソースの定義を追加することも可能です。データソースを追加する場合に使用しなければならない構文の例は次のとおりです。行の最後にある「\」はコマンドが次の行に続くことを表しています。

例3.3 データソース定義を追加する構文の例

```
$ EAP_HOME/bin/jboss-cli --connect
[standalone@localhost:9999 /] data-source add --name=ExampleDS
--jndi-name=java:/ExampleDS \
  --connection-url=jdbc:h2:mem:test;DB_CLOSE_DELAY=-1 --
driver-name=h2 \
  --user-name=sa --password=sa
```

データソースの設定方法の詳細は「[DataSource 設定の更新](#)」を参照してください。

2. 必要な依存関係を追加する

Seam 2.2 アプリケーションは JSF 1.2 を使用するため、JSF 1.2 モジュールの依存関係を追加し、JSF 2.0 モジュールを除外する必要があります。これを実行するには、以下のデータが格

納される EAR の **META-INF**/ ディレクトリーに **jboss-deployment-structure.xml** ファイルを作成する必要があります。

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-
structure:1.0">
  <deployment>
    <dependencies>
      <module name="javax.faces.api" slot="1.2" export="true"/>
      <module name="com.sun.jsf-impl" slot="1.2"
export="true"/>
    </dependencies>
  </deployment>
  <sub-deployment name="jboss-seam-booking.war">
    <exclusions>
      <module name="javax.faces.api" slot="main"/>
      <module name="com.sun.jsf-impl" slot="main"/>
    </exclusions>
    <dependencies>
      <module name="javax.faces.api" slot="1.2"/>
      <module name="com.sun.jsf-impl" slot="1.2"/>
    </dependencies>
  </sub-deployment>
</jboss-deployment-structure>
```

アプリケーションがサードパーティー製のログインフレームワークを使用する場合は、「[ログイン依存関係の編集](#)」で説明されているようにこれらの依存関係を追加する必要があります。

3. アプリケーションが Hibernate 3.x を使用する場合は、最初に Hibernate 4 ライブラリーを使用してアプリケーションを実行する

アプリケーションが Seam Managed Persistence Context、Hibernate 検索、バリデーション、または Hibernate 4 で変更された他の機能を使用しない場合は、Hibernate 4 ライブラリーで実行できることがあります。ただし、Hibernate クラスを参照する

ClassNotFoundException または **ClassCastException** がある場合や以下のようなエラーが発生した場合は、次の手順に従い、Hibernate 3.3 ライブラリーを使用するようアプリケーションを変更する必要があることがあります。

```
Caused by: java.lang.LinkageError: loader constraint
violation in interface itable initialization: when resolving method
"org.jboss.seam.persistence.HibernateSessionProxy.getSession(Lorg/hi
bernate/EntityMode;)Lorg/hibernate/Session;" the class loader
(instance of org/jboss/modules/ModuleClassLoader) of the current
class, org/jboss/seam/persistence/HibernateSessionProxy, and the
class loader (instance of org/jboss/modules/ModuleClassLoader) for
interface org/hibernate/Session have different Class objects for the
type org/hibernate/Session used in the signature
```

4. 外部フレームワークまたは他の場所より依存するアーカイブをコピーする

Hibernate 3.x を使用するアプリケーションが Hibernate 4 を正常に使用できない場合は、Hibernate 3.x JAR を **/lib** ディレクトリーにコピーし、以下のように **META-INF/jboss-deployment-structure.xml** のデプロイメントセクションで Hibernate モジュールを除外する必要があります。

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-
structure:1.0">
  <deployment>
```

```

    <exclusions>
      <module name="org.hibernate"/>
    </exclusions>
  </deployment>
</jboss-deployment-structure>

```

アプリケーションで Hibernate 3.x をバンドルする場合は、他にも実行する必要がある手順があります。詳細については、[「Hibernate および JPA を使用するアプリケーションの変更設定」](#)を参照してください。

5. Seam 2.2 JNDI エラーをデバッグおよび解決する

Seam 2.2 アプリケーションを移行するときに次のような

javax.naming.NameNotFoundException エラーがログに記録されることがあります。

```

javax.naming.NameNotFoundException: Name 'jboss-seam-booking' not
found in context ''

```

コード全体で JNDI ルックアップを変更しない場合は、以下のようにアプリケーションの **components.xml** ファイルを変更できます。

a. 既存の core-init 要素の置き換え

最初に、以下のように既存の core-init 要素を置き換える必要があります。

```

<!-- <core:init jndi-pattern="jboss-seam-booking/#
{ejbName}/local" debug="true" distributable="false"/> -->
<core:init debug="true" distributable="false"/>

```

b. サーバーログでの JNDI バインディング INFO メッセージの検索

次に、アプリケーションがデプロイされたときに、JNDI バインディング INFO メッセージがサーバーログに記録されていることを確認します。JNDI バインディングメッセージは以下のようなはずです。

```

INFO
org.jboss.as.ejb3.deployment.processors.EjbJndiBindingsDeployment
UnitProcessor (MSC service thread 1-1) JNDI bindings for session
bean
named AuthenticatorAction in deployment unit subdeployment
"jboss-seam-booking.jar" of deployment "jboss-seam-booking.ear"
are as follows:
    java:global/jboss-seam-booking/jboss-seam-
booking.jar/AuthenticatorAction!org.jboss.seam.example.booking.Au
thenticator
    java:app/jboss-seam-
booking.jar/AuthenticatorAction!org.jboss.seam.example.booking.Au
thenticator

    java:module/AuthenticatorAction!org.jboss.seam.example.booking.Au
thenticator
    java:global/jboss-seam-booking/jboss-seam-
booking.jar/AuthenticatorAction
    java:app/jboss-seam-booking.jar/AuthenticatorAction
    java:module/AuthenticatorAction

```

c. コンポーネント要素の追加

ログの各 JNDI バインディング INFO メッセージに対して、一致する **component** 要素を

`components.xml` ファイルに追加します。

```
<component
class="org.jboss.seam.example.booking.AuthenticatorAction" jndi-
name="java:app/jboss-seam-booking.jar/AuthenticatorAction" />
```

移行の問題に関するデバッグや解決方法の詳細は「[移行の問題のデバッグと解決](#)」を参照してください。

Seam 2 アーカイブでの既知の移行問題の一覧は、「[Seam 2.2 アーカイブの移行の問題](#)」を参照してください。

結果

Seam 2.2 アーカイブが JBoss EAP 6 上にデプロイされ、正常に実行されます。

バグを報告する

3.2.13.2. Seam 2.2 アーカイブの移行の問題

Seam 2.2 Drools と Java 7 の互換性がない

Seam 2.2 Drools と Java 7 は互換性がなく、エラー `org.drools.RuntimeDroolsException: value '1.7' is not a valid language level` が発生します。

Seam 2.2.5 の署名された `cglib.jar` によって Spring の例が動作しない

JBoss EAP 5 の Seam 2.2.5 に含まれる、署名された `cglib.jar` を使用して Spring サンプルが実行されると、次の原因で実行に失敗します。

```
java.lang.SecurityException: class
"org.jboss.seam.example.spring.UserService$$EnhancerByCGLIB$$7d6c3d12"'s
signer information does not match signer information of other classes in
the same package
```

この問題を回避するには、次のように `cglib.jar` を無署名にします。

```
zip -d $SEAM_DIR/lib/cglib.jar META-INF/JBOSSCOD\*
```

Seamby の例が `NotLoggedInException` によって失敗する

`SOAPRequestHandler` のメッセージを処理する際に SOAP メッセージのヘッダーが null であるため、conversation ID が設定されないことがこの問題の原因です。

この問題を回避するには、<https://issues.jboss.org/browse/JBPAPP-8376> の記述どおりに `org.jboss.seam.webservice.SOAPRequestHandler.handleOutbound` を上書きします。

Seamby の例が `UnsupportedOperationException: no transaction` によって失敗する

このバグは、JBoss EAP 6 で UserTransaction の JNDI 名が変更になったことが原因です。

この問題を回避するには、<https://issues.jboss.org/browse/JBPAPP-8322> の記述どおりに `org.jboss.seam.transaction.Transaction.getUserTransaction` を上書きします。

Tasks の例が `org.jboss.resteasy.spi.UnhandledException: Unable to unmarshall request body` をスローする

このバグの原因は JBoss EAP 5.1.2 に含まれる seam-resteasy-2.2.5 と、JBoss EAP 6 に含まれる RESTEasy 2.3.1.GA の間に互換性がないことです。

この問題を回避するには、<https://issues.jboss.org/browse/JBPAPP-8315> のとおりに **jboss-deployment-structure.xml** を使用してメインデプロイメントから `resteasy-jaxrs`、`resteasy-jettison-provider`、および `resteasy-jaxb-provider` を除外し、**jboss-seam-tasks.war** から `resteasy-jaxrs`、`resteasy-jettison-provider`、`resteasy-jaxb-provider`、および `resteasy-yaml-provider` を除外します。その後、EAR に Seam 2.2 とバンドルされる RESTEasy ライブラリーが含まれるようにする必要があります。

AJAX の要求中に `org.jboss.seam.core.SynchronizationInterceptor` とステートフルコンポーネントインスタンスの EJB ロックがデッドロックする

「Caused by javax.servlet.ServletException with message: "javax.el.ELException: /main.xhtml @36,71 value="#{hotelSearch.pageSize}": org.jboss.seam.core.LockTimeoutException: could not acquire lock on @Synchronized component: hotelSearch」が含まれるエラーページまたは同様のエラーメッセージが表示されます。

Seam 2 はステートフルセッション Bean (SFSB) ロックの外部で異なるスコープにて独自のロックを行うことが問題となります。そのため、同じトランザクションでスレッドが EJB へ 2 回アクセスすると、最初の呼び出しの後に seam ロックではなく SFSB ロックを取得します。その後、2 つ目のスレッドは seam ロックを取得でき、EJB ロックをヒットし待機します。最初のスレッドが 2 回目の呼び出しを実行しようとする、seam 2 インターセプター上でブロックし、デッドロックが発生します。Java EE 5 では平行アクセスが行われると即座に例外が発生しましたが、Java EE 6 ではこの挙動が変更されました。

この問題を回避するには EJB に `@AccessTimeout(0)` を追加します。これにより、この状態に陥ると即座に `ConcurrentAccessException` が発生します。

Dvdstore の例の注文作成が `javax.ejb.EJBTransactionRolledbackException` によって失敗する

dvdstore サンプルが以下のエラーを表示します。

```
JBAS011437: Found extended persistence context in SFSB invocation call stack but that cannot be used because the transaction already has a transactional context associated with it. This can be avoided by changing application code, either eliminate the extended persistence context or the transactional context. See JPA spec 2.0 section 7.6.3.1.
```

この問題は JPA 仕様の変更が原因です。

この問題を修正するには、`CheckoutAction` クラスと `ShowOrdersAction` クラスの永続コンテキストを `transactional` に変更し、エンティティマネージャーのマージ操作を `cancelOrder` および `detailOrder` メソッドで使います。

JBoss Cache の Seam キャッシュプロバイダーを JBoss EAP 6 で使用できない

JBoss Cache は JBoss EAP 6 ではサポートされていません。そのため、JBoss Cache の Seam キャッシュプロバイダーは、以下エラーによりアプリケーションサーバーの Seam アプリケーションで失敗します。

```
java.lang.NoClassDefFoundError: org/jboss/util/xml/JBossEntityResolver
```

JBoss EAP 6 での JPA エンティティの Hibernate 3.3.x 自動スキャンの問題

この問題を修正するには、すべてのエンティティークラスを手動で persistence.xml ファイルにリストします。例は次のとおりです。

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
version="1.0">
  <persistence-unit name="example_pu">
    <description>Hibernate 3 Persistence Unit.</description>
    <jta-data-source>java:jboss/datasources/ExampleDS</jta-data-
source>
    <properties>
      <property name="jboss.as.jpa.providerModule"
value="hibernate3-bundled" />
    </properties>
    <class>com.acme.Foo</class>
    <class>com.acme.Bar</class>
  </persistence-unit>
</persistence>
```

EJB でないスレッドから EJB Seam コンポーネントを呼び出すと javax.naming.NameNotFoundException が発生する

この問題は、新しいモジュラークラスローディングシステムを実装し、新たに標準化された JNDI 名前空間の慣例を採用する JBoss EAP 6 の変更が原因です。**java:app** 名前空間は、単一のアプリケーションですべてのコンポーネントによって共有される名前を対象としています。Quartz の非同期スレッドなどの EE でないスレッドは、アプリケーションサーバーインスタンスにデプロイされたすべてのアプリケーションによって共有される **java:global** 名前空間を使用する必要があります。

Quartz 非同期メソッドから EJB Seam コンポーネントの呼び出しを実行しようとしたときに **javax.naming.NameNotFoundException** を受信した場合、グローバル JNDI 名を使用するように **components.xml** ファイルを次のように変更する必要があります。

```
<component class="org.jboss.seam.example.quartz.MyBean" jndi-
name="java:global/seam-quartz/quartz-ejb/myBean"/>
```

JNDI の変更に関する詳細は、「[アプリケーションの JNDI 名前空間名の更新](#)」を参照してください。この問題の詳細は、カスタマーポータル [の Red Hat JBoss Web Framework Kit 向け『2.2.0 Release Notes』](#)に記載されている「[BZ#948215 - Seam 2.3 javax.naming.NameNotFoundException trying to call EJB Seam components from quartz asynchronous methods](#)」を参照してください。

[バグを報告する](#)

3.2.14. Spring アプリケーションの移行

3.2.14.1. Spring アプリケーションの移行

Spring アプリケーションの移行に関する情報は、カスタマーポータル <https://access.redhat.com/site/documentation/> に記載されている *Red Hat JBoss Web Framework Kit* ドキュメントを参照してください。**Red Hat JBoss Middleware** を選択し、*Red Hat JBoss Web Framework Kit* のリンクをクリックします。『Spring Installation Guide』および『Spring Developer Guide』を複数の形式で利用できます。

[バグを報告する](#)

3.2.15. 移行に影響するその他の変更

3.2.15.1. 移行に影響する可能性があるその他の変更について理解する

移行に影響を与える可能性がある JBoss EAP 6 のその他の変更内容は次の通りです。

- [「Maven プラグイン名の変更」](#)
- [「クライアントアプリケーションの変更」](#)

[バグを報告する](#)

3.2.15.2. Maven プラグイン名の変更

`jboss-maven-plugin` は更新されていないため、JBoss EAP 6 では動作しません。`org.jboss.as.plugins.jboss-as-maven-plugin` を使用して正しいディレクトリーをデプロイする必要があります。

[バグを報告する](#)

3.2.15.3. クライアントアプリケーションの変更

JBoss EAP 6 に接続するクライアントアプリケーションの移行を計画する場合、クライアントライブラリーをバンドルする JAR の名前と場所が変更になったことに注意してください。この JAR の名前は `jboss-client.jar` に変更され、`EAP_HOME/bin/client/` ディレクトリーにあります。この JAR は `EAP_HOME/client/jbossall-client.jar` に置き換わるもので、リモートクライアントから JBoss EAP 6 に接続するために必要なすべての依存関係が含まれています。

[バグを報告する](#)

第4章 ツールとヒント

4.1. 移行に役立つリソース

4.1.1. 移行に役立つリソース

アプリケーションを JBoss EAP 6 に移行する時に便利なリソースの一覧は次のとおりです。

ツール

設定変更の一部を自動化するのに役立つツールが複数あります。詳細は [「移行に便利なツールについて理解する」](#) を参照してください。

デバッグのヒント

アプリケーションの移行時に発生する問題やエラーの最も一般的な原因と解決法については [「移行の問題のデバッグと解決」](#) を参照してください。

移行の例

JBoss EAP 6 へ移行されたアプリケーションの例については、[「サンプルアプリケーションの移行の確認」](#) を参照してください。

[バグを報告する](#)

4.1.2. 移行に便利なツールについて理解する

概要

移行に便利なツールは複数あります。これらのツールとその説明の一覧は次のとおりです。

Tattletale

モジュラークラスローディングの変更に伴い、アプリケーション依存関係を検索し、修正する必要があります。Tattletale は依存するモジュールの名前を特定し、アプリケーションに対して設定 XML を生成する時に便利なツールです。

[「Tattletale を用いたアプリケーション依存関係の検索」](#)

IronJacamar 移行ツール

JBoss EAP 6 ではデータソースとリソースアダプターは個別のファイルに設定されていません。データソースとリソースアダプターはサーバー設定ファイルに定義され、新しいスキーマを使用します。IronJacamar 移行ツールは以前の設定を JBoss EAP 6 が想定する形式に変換するときに便利です。

[「IronJacamar ツールを使用してデータソースとリソースアダプターの設定を移行する」](#)

[バグを報告する](#)

4.1.3. Tattletale を用いたアプリケーション依存関係の検索

概要

JBoss EAP 6 のモジュラークラスローディングの変更にともない、アプリケーションを移行するときに **ClassNotFoundException** または **ClassCastException** トレースが JBoss のログに記録されるこ

とがあります。このエラーを解決するには、例外が指定するクラスが含まれる JAR を探す必要があります。

Tattletale はアプリケーションを再帰的にスキャンし、その内容の詳細レポートを提供する優れたサードパーティーツールです。Tattletale 1.2.0.Beta2 やそれ移行のバージョンには JBoss EAP 6 で使用される新しい JBoss Modules のクラスローディングに役立つ追加のサポートが含まれています。Tattletale の「JBoss AS 7」レポートを使用して、自動的に依存するモジュール名を特定および生成し、アプリケーションの **jboss-deployment-structure.xml** ファイルが含まれるようにすることが可能です。

手順4.1 Tattletale をインストールおよび実行してアプリケーションの依存関係を検索する

1. 「Tattletale のダウンロードとインストール」
2. 「Tattletale レポートの作成および確認」



注記

Tattletale は JBoss EAP 6 の一部としてはサポートされないサードパーティのツールです。Tattletale のインストール方法や使用方法に関する最新のドキュメントは、Tattletale の Web サイト <http://www.jboss.org/tattletale> をご覧ください。

[バグを報告する](#)

4.1.4. Tattletale のダウンロードとインストール

手順4.2 Tattletale のダウンロードとインストール

1. <http://sourceforge.net/projects/jboss/files/JBoss%20Tattletale> より Tattletale バージョン 1.2.0.Beta2 またはそれ以降のバージョンをダウンロードします。
2. 希望のディレクトリーにファイルを展開します。
3. 次のように **TATTLETALE_HOME/jboss-tattletale.properties** ファイルを変更します。
 - a. **ee6** と **as7** を **profiles** プロパティに追加します。

```
profiles=java5, java6, ee6, as7
```

- b. **scan** と **reports** プロパティをアンコメントします。

[バグを報告する](#)

4.1.5. Tattletale レポートの作成および確認

1. 次のコマンドを実行して Tattletale レポートを作成します: **java -jar TATTLETALE_HOME/tattletale.jar APPLICATION_ARCHIVE OUTPUT_DIRECTORY**

たとえば、**java -jar tattletale-1.2.0.Beta2/tattletale.jar ~/applications/jboss-seam-booking.ear ~/output-results/** となります。

2. ブラウザーで **OUTPUT_DIRECTORY/index.html** ファイルを開き、「Reports」セクション下の「JBoss AS 7」をクリックします。

- a. 左側の列にはアプリケーションによって使用されるアーカイブが一覧表示されます。`ARCHIVE_NAME` リンクをクリックし、場所やマニフェスト情報、含まれるクラスなどアーカイブの詳細を表示します。
- b. 右側の列にある `jboss-deployment-structure.xml` リンクは、左側の列に名前が表示されているアーカイブのモジュール依存関係を指定する方法を表示します。このリンクをクリックし、アーカイブのデプロイメント依存関係モジュール情報を定義する方法を確認します。

[バグを報告する](#)

4.1.6. IronJacamar ツールを使用してデータソースとリソースアダプターの設定を移行する

概要

以前のバージョンのアプリケーションサーバーでは、ファイル名が `*-ds.xml` で終わるファイルを使用してデータソースとリソースアダプターが設定されデプロイされました。IronJacamar 1.1 ディストリビューションには、これらの設定ファイルを JBoss EAP 6 が想定する形式に変換できるツールが含まれています。このツールは、以前のリリースよりソースの設定ファイルを解析し、XML 設定を作成してファイルを新しい形式で出力します。この XML を、JBoss EAP 6 のサーバー設定ファイルにある正しいサブシステム下にコピーおよび貼り付けできます。このツールは、できる限り以前の属性や要素を新しい形式に変換しますが、生成されたファイルに変更を追加する必要がある場合があります。

手順4.3 IronJacamar 移行ツールのインストールと実行

1. [「IronJacamar 移行ツールのダウンロードとインストール」](#)
2. [「IronJacamar 移行ツールを使用したデータソース設定ファイルの変換」](#)
3. [「IronJacamar 移行ツールを使用したリソースアダプター設定ファイルの変換」](#)



注記

IronJacamar 移行ツールは JBoss EAP 6 の一部としてはサポートされないサードパーティーのツールです。IronJacamar に関する情報は <http://www.ironjacamar.org/> を参照してください。このツールのインストール方法や使用方法についての最新のドキュメントは <http://www.ironjacamar.org/documentation.html> を参照してください。

[バグを報告する](#)

4.1.7. IronJacamar 移行ツールのダウンロードとインストール



注記

移行ツールは IronJacamar 1.1 とそれ以降のバージョンでのみ使用可能で、Java 7 またはそれ以降のバージョンが必要です。

1. <http://www.ironjacamar.org/download.html> から IronJacamar の最新ディストリビューションをダウンロードします。
2. 希望のディレクトリーにダウンロードしたファイルを展開します。
3. IronJacamar ディストリビューションのコンバータスクリプトを探します。

- Linux スクリプトは、**`IRONJACAMAR_HOME/doc/as/converter.sh`** にあります。
- Windows バッチファイルは、**`IRONJACAMAR_HOME/doc/as/converter.bat`** にあります。

[バグを報告する](#)

4.1.8. IronJacamar 移行ツールを使用したデータソース設定ファイルの変換



注記

IronJacamar コンバータースクリプトには Java 7 またはそれ以上のバージョンが必要です。

手順4.4 データソース設定ファイルの変換

1. コマンドラインを開き、**`IRONJACAMAR_HOME/doc/as/`** ディレクトリーへ移動します。
2. 以下のコマンドを入力して、コンバータースクリプトを実行します。

- Linux の場合: **`./converter.sh -ds SOURCE_FILE TARGET_FILE`**
- Microsoft Windows の場合: **`./converter.bat -ds SOURCE_FILE TARGET_FILE`**

`SOURCE_FILE` は以前のリリースのデータソース `-ds.xml` ファイルです。**`TARGET_FILE`** に新しい設定が含まれます。

たとえば、カレントディレクトリーにある **`jboss-seam-booking-ds.xml`** データソース設定ファイルを変換するには、以下のように入力します。

- Linux の場合: **`./converter.sh -ds jboss-seam-booking-ds.xml new-datasource-config.xml`**
- Microsoft Windows の場合: **`./converter.bat -ds jboss-seam-booking-ds.xml new-datasource-config.xml`**

データソース変換のパラメーターは **`-ds`** になります。

3. ターゲットファイルから **`<datasource>`** 要素をコピーし、**`<subsystem xmlns="urn:jboss:domain:datasources:1.1"><datasources>`** 要素下のサーバー設定ファイルに貼り付けます。



重要

サーバーの再起動後も変更が維持されるようにするには、サーバーを停止してからサーバー設定ファイルを編集する必要があります。

- 管理対象ドメインで実行している場合は、XML を **`EAP_HOME/domain/configuration/domain.xml`** ファイルにコピーします。
- スタンドアロンサーバーとして実行している場合は、XML を **`EAP_HOME/standalone/configuration/standalone.xml`** ファイルにコピーします。

4. 新しい設定ファイルに生成された XML を変更します。

以下に、JBoss EAP 5.x に同梱された Seam 2.2 Booking サンプルの **jboss-seam-booking-ds.xml** データソースの例を示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<datasources>
  <local-tx-datasource>
    <jndi-name>bookingDataSource</jndi-name>
    <connection-url>jdbc:hsqldb:./</connection-url>
    <driver-class>org.hsqldb.jdbcDriver</driver-class>
    <user-name>sa</user-name>
    <password></password>
  </local-tx-datasource>
</datasources>
```

以下はコンバータスクリプトを実行して生成された設定ファイルになります。生成されたファイルには **<driver-class>** 要素が含まれます。JBoss EAP 6 では、**<driver>** 要素を使用してドライバクラスを定義する方法が推奨されます。**<driver-class>** 要素がコメントアウトされ、対応する **<driver>** 要素が追加された JBoss EAP 6 の設定ファイルにある XML は次のようになります。

```
<subsystem xmlns="urn:jboss:domain:datasources:1.1">
  <datasources>
    <datasource enabled="true" jndi-
name="java:jboss/datasources/bookingDataSource" jta="true"
      pool-name="bookingDataSource" use-ccm="true" use-java-
context="true">
      <connection-url>jdbc:hsqldb:./</connection-url>
      <!-- Comment out the following driver-class element
      since it is not the preferred way to define this.
      <driver-class>org.hsqldb.jdbcDriver</driver-class>
      -->

      <!-- Specify the driver, which is defined later in the
      datasource -->
      <driver>h2</driver>
      <transaction-isolation>TRANSACTION_NONE</transaction-
isolation>
      <pool>
        <prefill>>false</prefill>
        <use-strict-min>>false</use-strict-min>
        <flush-strategy>FailingConnectionOnly</flush-strategy>
      </pool>
      <security>
        <user-name>sa</user-name>
        <password/>
      </security>
      <validation>
        <validate-on-match>>false</validate-on-match>
        <background-validation>>false</background-validation>
        <use-fast-fail>>false</use-fast-fail>
      </validation>
      <timeout/>
      <statement>
        <track-statements>>false</track-statements>
      </statement>
```

```

</datasource>
<drivers>
  <!-- The following driver element was not in the
  XML target file. It was created manually. -->
  <driver name="h2" module="com.h2database.h2">
    <xa-datasource-class>org.h2.jdbcx.JdbcDataSource</xa-
datasource-class>
  </driver>
</drivers>
</datasources>
</subsystem>

```

[バグを報告する](#)

4.1.9. IronJacamar 移行ツールを使用したリソースアダプター設定ファイルの変換



注記

IronJacamar コンバータースクリプトには Java 7 またはそれ以上のバージョンが必要です。

1. コマンドラインを開き、**IRONJACAMAR_HOME/docs/as/** ディレクトリーへ移動します。
2. 以下のコマンドを入力して、コンバータースクリプトを実行します。

- Linux の場合: **./converter.sh -ra SOURCE_FILE TARGET_FILE**
- Microsoft Windows の場合: **./converter.bat -ra SOURCE_FILE TARGET_FILE**

SOURCE_FILE は以前のリリースのリソースアダプター -ds.xml ファイルです。**TARGET_FILE** には、新しい設定が含まれます。

たとえば、カレントディレクトリーにある **mttestadapter-ds.xml** リソースアダプター設定ファイルを変換するには、以下のように入力します。

- Linux の場合: **./converter.sh -ra mttestadapter-ds.xml new-adapter-config.xml**
- Microsoft Windows の場合: **./converter.bat -ra mttestadapter-ds.xml new-adapter-config.xml**

リソースアダプター変換のパラメーターは **-ra** になります。

3. ターゲットファイルから **<resource-adapters>** 要素全体をコピーし、**<subsystem xmlns="urn:jboss:domain:resource-adapters:1.1">** 要素下のサーバー設定ファイルに貼り付けます。



重要

サーバーの再起動後も変更が維持されるようにするには、サーバーを停止してからサーバー設定ファイルを編集する必要があります。

- 管理対象ドメインで実行している場合は、XML を **EAP_HOME/domain/configuration/domain.xml** ファイルにコピーします。
 - スタンドアロンサーバーとして実行している場合は、XML を **EAP_HOME/standalone/configuration/standalone.xml** ファイルにコピーします。
4. 新しい設定ファイルに生成された XML を変更します。

以下に、JBoss EAP 5.x TestSuite の **mttestadapter-ds.xml** リソースアダプター設定ファイルの例を示します。

```
<?xml version="1.0" encoding="UTF-8"?>
  <!--
=====
-->
  <!-- ConnectionManager setup for jboss test adapter
-->
  <!-- Build jmx-api (build/build.sh all) and view for config
documentation -->
  <!--
=====
-->
<connection-factories>
  <tx-connection-factory>
    <jndi-name>JBossTestCF</jndi-name>
    <xa-transaction/>
    <rar-name>jbosstestadapter.rar</rar-name>
    <connection-
definition>javax.resource.cci.ConnectionFactory</connection-
definition>
    <config-property name="IntegerProperty"
type="java.lang.Integer">2</config-property>
    <config-property name="BooleanProperty"
type="java.lang.Boolean">>false</config-property>
    <config-property name="DoubleProperty"
type="java.lang.Double">5.5</config-property>
    <config-property name="UrlProperty"
type="java.net.URL">http://www.jboss.org</config-property>
    <config-property name="sleepInStart" type="long">200</config-
property>
    <config-property name="sleepInStop" type="long">200</config-
property>
  </tx-connection-factory>
  <tx-connection-factory>
    <jndi-name>JBossTestCF2</jndi-name>
    <xa-transaction/>
    <rar-name>jbosstestadapter.rar</rar-name>
    <connection-
definition>javax.resource.cci.ConnectionFactory</connection-
definition>
    <config-property name="IntegerProperty"
type="java.lang.Integer">2</config-property>
    <config-property name="BooleanProperty"
type="java.lang.Boolean">>false</config-property>
    <config-property name="DoubleProperty"
```

```

type="java.lang.Double">5.5</config-property>
  <config-property name="UrlProperty"
type="java.net.URL">http://www.jboss.org</config-property>
  <config-property name="sleepInStart" type="long">200</config-
property>
  <config-property name="sleepInStop" type="long">200</config-
property>
</tx-connection-factory>
<tx-connection-factory>
  <jndi-name>JBossTestCFByTx</jndi-name>
  <xa-transaction/>
  <track-connection-by-tx>true</track-connection-by-tx>
  <rar-name>jbosstestadapter.rar</rar-name>
  <connection-
definition>javax.resource.cci.ConnectionFactory</connection-
definition>
  <config-property name="IntegerProperty"
type="java.lang.Integer">2</config-property>
  <config-property name="BooleanProperty"
type="java.lang.Boolean">>false</config-property>
  <config-property name="DoubleProperty"
type="java.lang.Double">5.5</config-property>
  <config-property name="UrlProperty"
type="java.net.URL">http://www.jboss.org</config-property>
  <config-property name="sleepInStart" type="long">200</config-
property>
  <config-property name="sleepInStop" type="long">200</config-
property>
</tx-connection-factory>
</connection-factories>

```

以下はコンバータスクリプトを実行して生成された設定ファイルになります。生成された XML ファイルのクラス名属性値「FIXME_MCF_CLASS_NAME」を管理対象接続ファクトリー（この例では、「org.jboss.test.jca.adapter.TestManagedConnectionFactory」）の正しいクラス名に置き換えます。JBoss EAP 6 の設定ファイルにある **<class-name>** 要素値が変更された XML は次のようになります。

```

<subsystem xmlns="urn:jboss:domain:resource-adapters:1.1">
  <resource-adapters>
    <resource-adapter>
      <archive>jbosstestadapter.rar</archive>
      <transaction-support>XATransaction</transaction-support>
      <connection-definitions>
        <!-- Replace the "FIXME_MCF_CLASS_NAME" class-name value with the
correct class name
        <connection-definition class-name="FIXME_MCF_CLASS_NAME"
enabled="true"
          jndi-name="java:jboss/JBossTestCF" pool-name="JBossTestCF"
          use-ccm="true" use-java-context="true"> -->
      <connection-definition
        class-
name="org.jboss.test.jca.adapter.TestManagedConnectionFactory"
        enabled="true"
        jndi-name="java:jboss/JBossTestCF" pool-name="JBossTestCF"
        use-ccm="true" use-java-context="true">
      <config-property name="IntegerProperty">2</config-property>

```

```

<config-property name="sleepInStart">200</config-property>
<config-property name="sleepInStop">200</config-property>
<config-property name="BooleanProperty">false</config-property>
<config-property
name="UrlProperty">http://www.jboss.org</config-property>
<config-property name="DoubleProperty">5.5</config-property>
<pool>
  <prefill>false</prefill>
  <use-strict-min>false</use-strict-min>
  <flush-strategy>FailingConnectionOnly</flush-strategy>
</pool>
<security>
  <application/>
</security>
<timeout/>
<validation>
  <background-validation>false</background-validation>
  <use-fast-fail>false</use-fast-fail>
</validation>
</connection-definition>
  </connection-definitions>
</resource-adapter>
<resource-adapter>
  <archive>jbosstestadapter.rar</archive>
  <transaction-support>XATransaction</transaction-support>
  <connection-definitions>
    <!-- Replace the "FIXME_MCF_CLASS_NAME" class-name value with the
correct class name
    <connection-definition class-name="FIXME_MCF_CLASS_NAME"
enabled="true"
      jndi-name="java:jboss/JBossTestCF2" pool-name="JBossTestCF2"
      use-ccm="true" use-java-context="true"> -->
  <connection-definition
    class-
name="org.jboss.test.jca.adapter.TestManagedConnectionFactory"
    enabled="true"
    jndi-name="java:jboss/JBossTestCF2" pool-name="JBossTestCF2"
    use-ccm="true" use-java-context="true">
    <config-property name="IntegerProperty">2</config-property>
    <config-property name="sleepInStart">200</config-property>
    <config-property name="sleepInStop">200</config-property>
    <config-property name="BooleanProperty">false</config-property>
    <config-property
name="UrlProperty">http://www.jboss.org</config-property>
    <config-property name="DoubleProperty">5.5</config-property>
    <pool>
      <prefill>false</prefill>
      <use-strict-min>false</use-strict-min>
      <flush-strategy>FailingConnectionOnly</flush-strategy>
    </pool>
    <security>
      <application/>
    </security>
    <timeout/>
    <validation>
      <background-validation>false</background-validation>

```

```

        <use-fast-fail>false</use-fast-fail>
    </validation>
</connection-definition>
    </connection-definitions>
</resource-adapter>
<resource-adapter>
    <archive>jbosstestadapter.rar</archive>
    <transaction-support>XATransaction</transaction-support>
    <connection-definitions>
        <!-- Replace the "FIXME_MCF_CLASS_NAME" class-name value with the
correct class name
        <connection-definition class-name="FIXME_MCF_CLASS_NAME"
enabled="true"
            jndi-name="java:jboss/JBossTestCFByTx" pool-
name="JBossTestCFByTx"
            use-ccm="true" use-java-context="true"> -->
    <connection-definition
        class-
name="org.jboss.test.jca.adapter.TestManagedConnectionFactory"
        enabled="true"
        jndi-name="java:jboss/JBossTestCFByTx" pool-
name="JBossTestCFByTx"
        use-ccm="true" use-java-context="true">
        <config-property name="IntegerProperty">2</config-property>
        <config-property name="sleepInStart">200</config-property>
        <config-property name="sleepInStop">200</config-property>
        <config-property name="BooleanProperty">false</config-property>
        <config-property
name="UrlProperty">http://www.jboss.org</config-property>
        <config-property name="DoubleProperty">5.5</config-property>
    </pool>
        <prefill>false</prefill>
        <use-strict-min>false</use-strict-min>
        <flush-strategy>FailingConnectionOnly</flush-strategy>
    </pool>
    <security>
        <application/>
    </security>
    <timeout/>
    <validation>
        <background-validation>false</background-validation>
        <use-fast-fail>false</use-fast-fail>
    </validation>
</connection-definition>
    </connection-definitions>
</resource-adapter>
</resource-adapters>
</subsystem>

```

[バグを報告する](#)

4.2. 移行の問題のデバッグ

4.2.1. 移行の問題のデバッグと解決

クラスローディング、JNDI ネーミングルール、およびアプリケーションのその他の変更にもない、アプリケーションをそのままデプロイしようとする例外やその他のエラーが発生することがあります。発生する可能性のある一般的な例外や、エラーの一部を解決する方法については以下を参照してください。

- [「ClassNotFoundExceptions および NoClassDefFoundErrors のデバッグと解決」](#)
- [「ClassCastExceptions のデバッグと解決」](#)
- [「DuplicateServiceExceptions のデバッグと解決」](#)
- [「JBoss Seam のデバッグページエラーのデバッグと解決」](#)

[バグを報告する](#)

4.2.2. ClassNotFoundExceptions および NoClassDefFoundErrors のデバッグと解決

概要

通常、ClassNotFoundExceptions は未解決の依存関係が原因で発生します。そのため、他のモジュール上で依存関係を明示的に定義するか、外部ソースより JAR をコピーする必要があります。

1. 最初に、不明な依存関係を探します。詳細は、[「JBoss モジュール依存関係の検索」](#)を参照してください。
2. 不明なクラスのモジュールがない場合は、以前のインストールで JAR を探します。詳細は、[「以前のインストールでの JAR の検索」](#)を参照してください。

[バグを報告する](#)

4.2.3. JBoss モジュール依存関係の検索

依存関係を解決するには、最初に **EAP_HOME/modules/system/layers/base/** ディレクトリ内で **ClassNotFoundException** によって指定されたクラスが含まれるモジュールを探します。クラスのモジュールを見つけた場合は、マニフェストエントリに依存関係を追加する必要があります。

たとえば、ログに次の ClassNotFoundException トレースが記録されているとします。

```
Caused by: java.lang.ClassNotFoundException:
org.apache.commons.logging.Log
    from [Module "deployment.TopicIndex.war:main" from Service Module
Loader]
    at
org.jboss.modules.ModuleClassLoader.findClass(ModuleClassLoader.java:188)
```

この場合、次の手順を実行してこのクラスが含まれる JBoss モジュールを探します。

手順4.5 依存関係の特定

1. 最初にクラスの明白なモジュールがあるかを判断します。
 - a. **EAP_HOME/modules/system/layers/base/** ディレクトリへ移動し、**ClassNotFoundException** で指定されたクラスと一致するモジュールパスを探します。

モジュールパス **org/apache/commons/logging/** が見つかります。

- b. **EAP_HOME/modules/system/layers/base/org/apache/commons/logging/main/module.xml** ファイルを開き、モジュール名を探します。この例では "org.apache.commons.logging" になります。
- c. **MANIFEST.MF** ファイルの Dependencies にモジュール名を追加します。

```
Manifest-Version: 1.0
Dependencies: org.apache.commons.logging
```

2. クラスの明白なモジュールパスがない場合、依存関係を他の場所で探す必要があることがあります。
 - a. Tattletale レポートで **ClassNotFoundException** によって命名されたクラスを探します。
 - b. **EAP_HOME/modules** ディレクトリーで JAR が含まれているモジュールを探し、前の手順のとおりモジュール名を探します。

バグを報告する

4.2.4. 以前のインストールでの JAR の検索

サーバーによって定義されたモジュールにパッケージ化された JAR にクラスがない場合は、**EAP5_HOME** インストールまたは以前のサーバーの **lib/** ディレクトリーで JAR を探します。

たとえば、ログに次の **ClassNotFoundException** トレースが記録されているとします。

```
Caused by: java.lang.NoClassDefFoundError:
org/hibernate/validator/ClassValidator at
java.lang.Class.getDeclaredMethods0(Native Method)
```

この場合、次を実行してこのクラスが含まれる JAR を探します。

1. ターミナルを開き、**EAP5_HOME/** ディレクトリーに移動します。
2. コマンドを実行します。

```
grep 'org.hibernate.validator.ClassValidator' `find . \-name '*.jar'`
```

3. 複数の結果が表示されることもあります。その場合、必要な JAR は次のとおりです。

```
Binary file ./jboss-eap-5.1/seam/lib/hibernate-validator.jar matches
```

4. この JAR をアプリケーションの **lib/** ディレクトリーへコピーします。

大量の JAR が必要な場合は、クラスのモジュールを定義した方が簡単な場合があります。詳細について

は、https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/ にある JBoss EAP 6 向け『開発ガイド』の章「アプリケーションの開発」の「モジュール」を参照してください。

5. アプリケーションを再ビルドし、再デプロイします。

バグを報告する

4.2.5. ClassCastExceptions のデバッグと解決

ClassCastExceptions は、拡張するクラスではなく他のクラスによってクラスがロードされるときに発生することが多くあります。また、同じクラスが複数の JAR に存在することが原因である場合もあります。

1. **ClassCastException** によって名前付けされたクラスが含まれる JAR をすべて見つけるため、アプリケーションを検索します。クラスに対して定義されたモジュールがある場合、アプリケーションの WAR や EAR から重複する JAR を探し、削除します。
2. クラスが含まれる JBoss モジュールを探し、**MANIFEST.MF** ファイルまたは **jboss-deployment-structure.xml** ファイルに依存関係を明示的に定義します。詳細については、https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/ にある JBoss EAP 6 向け『開発ガイド』の章「クラスローディングとモジュール」の「クラスローディングとサブデプロイメント」を参照してください。
3. 上記の手順に従っても解決されない場合、クラスローダーの情報をログに出力すると問題の原因を判断できることがあります。たとえば、次の **ClassCastException** がログに記録されているとします。

```
java.lang.ClassCastException: com.example1.CustomClass1 cannot be
cast to com.example2.CustomClass2
```

- a. コードで **ClassCastException** によって命名されたクラスに対するクラスローダーの情報をログに出力します。例は次のとおりです。

```
logger.info("Class loader for CustomClass1: " +
com.example1.CustomClass1.getClass().getClassLoader().toString())
;
logger.info("Class loader for CustomClass2: " +
com.example2.CustomClass2.getClass().getClassLoader().toString())
;
```

- b. ログの情報には、どのモジュールがクラスをロードするかが記載されています。アプリケーションに基づき、競合する JAR を削除または移動する必要があります。

バグを報告する

4.2.6. DuplicateServiceExceptions のデバッグと解決

JAR のサブデプロイメントに対して DuplicateServiceException が発生したり、JBoss EAP 6 に EAR をデプロイするときに WAR アプリケーションがすでにインストールされているというメッセージが表示される場合、JBossWS によるデプロイメント処理の変更が原因である可能性があります。

JBossWS 3.3.0 リリースでは、TCK6 とのシームレスな互換性を実現するため、サーブレットベースのエンドポイントに対して新しいコンテキストルートマッピングアルゴリズムが導入されました。アプリケーション EAR アーカイブに同じ名前の WAR と JAR が含まれている場合、JBossWS が同じ名前の WAR コンテキストと Web コンテキストを作成することがありますが、Web コンテキストが WAR コンテキストと競合し、デプロイメントエラーが発生します。この場合、以下の方法の 1 つを用いてデプロイメントの問題を解決します。

- 生成される Web コンテキストと WAR コンテキストが一意になるよう、JAR ファイルの名前を WAR とは異なる名前に変更します。

- `<context-root>` 要素を `jboss-web.xml` ファイルに提供します。
- `<context-root>` 要素を `jboss-webservices.xml` ファイルに提供します。
- WAR の `<context-root>` 要素を `application.xml` ファイルでカスタマイズします。

[バグを報告する](#)

4.2.7. JBoss Seam のデバッグページエラーのデバッグと解決

アプリケーションを移行し、正常にデプロイした後、JBoss Seam デバッグページへリダイレクトされるランタイムエラーが発生することがあります。このページの URL は `http://localhost:8080/APPLICATION_CONTEXT/debug.seam` です。このページでは、現在のログインセッションに関連する Seam コンテキストで Seam コンポーネントを確認したり調査することができます。

JBoss Seam Debug Page

This page allows you to browse and inspect components in any of the Seam contexts associated with the current session. It also shows a list of active, long-running conversations. You can select a conversation to view its contents or destroy it.

Conversations

Conversation ID	Nested?	Activity	Description	View ID	Action
15	false	12:26:58 PM - 12:27:01 PM	Book hotel: Hilton Diagonal Mar	/book.xhtml	Select Destroy

+ Component

+ Conversation Context (None selected)

+ Business Process Context

+ Session Context

+ Application Context

図4.1 JBoss Seam デバッグページ

このページへリダイレクトされる原因は、アプリケーションコードで処理されなかった例外を Seam がキャッチしたためであることがほとんどです。通常、「JBoss Seam デバッグページ」上のリンクで例外の根本的な原因を見つけることができます。

1. このページの **Component** セクションを展開し、`org.jboss.seam.caughtException` コンポーネントを探します。
2. 原因とスタックトレースが欠落している依存関係が示されているはずです。

JBoss Seam Debug Page

This page allows you to browse and inspect components in any of the Seam contexts associated with the current session. It also shows a list of active, long-running conversations. You can select a conversation to view its contents or destroy it.

Conversations

Conversation ID	Nested?	Activity	Description	View ID	Action
15	false	12:26:58 PM - 12:27:01 PM	Book hotel: Hilton Diagonal Mar	/book.xhtml	Select Destroy

- Component

Select a component from one of the contexts below

[- Component \(org.jboss.seam.caughtException\)](#)

cause	java.lang.NoClassDefFoundError: org/slf4j/LoggerFactory
class	class javax.servlet.ServletException
localizedMessage	Servlet execution threw an exception
message	Servlet execution threw an exception
rootCause	java.lang.NoClassDefFoundError: org/slf4j/LoggerFactory
stackTrace	[org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:346), org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:248), org.jboss.seam.servlet.SeamFilter\$FilterChainImpl.doFilter(SeamFilter.java:83), org.jboss.seam.web.IdentityFilter.doFilter(IdentityFilter.java:40), [... rest of stacktrace omitted for display purposes]
toString()	javax.servlet.ServletException: Servlet execution threw an exception

+ Conversation Context (None selected)

+ Business Process Context

+ Session Context

+ Application Context

図4.2 コンポーネント `org.jboss.seam.caughtException` の情報

3. 「[ClassNotFountExceptions](#) および [NoClassDefFoundErrors](#) のデバッグと解決」の手法を使用してモジュール依存関係を解決します。

上記の例では、**org.slf4j** を **MANIFEST.MF** に追加するのが最も簡単な解決方法になります。

```
Manifest-Version: 1.0
Dependencies: org.slf4j
```

モジュールの依存関係を **jboss-deployment-structure.xml** ファイルに追加して解決する方法もあります。

```
<jboss-deployment-structure>
  <deployment>
    <dependencies>
      <module name="org.slf4j" />
    </dependencies>
  </deployment>
</jboss-deployment-structure>
```

[バグを報告する](#)

4.3. サンプルアプリケーションの移行の確認

4.3.1. サンプルアプリケーションの移行の確認

概要

以下は、JBoss EAP 6 へ移行された JBoss EAP 5.x サンプルアプリケーションの一覧になります。リンクをクリックすると、特定アプリケーションの変更内容の詳細を確認できます。

- [「Seam 2.2 JPA サンプルの JBoss EAP 6 への移行」](#)
- [「Seam 2.2 Booking サンプルの JBoss EAP 6 への移行」](#)
- [「Seam 2.2 Booking アーカイブの JBoss EAP 6 への移行: 手順説明」](#)

[バグを報告する](#)

4.3.2. Seam 2.2 JPA サンプルの JBoss EAP 6 への移行

概要

下記のタスクリストには、Seam 2.2 JPA のサンプルアプリケーションを JBoss EAP 6 へ正常に移行するために必要な変更の概要が記載されています。このサンプルアプリケーションは、最新の JBoss EAP 5 ディストリビューションの `EAP5.x_HOME/jboss-eap-5.x/seam/examples/jpa/` 下にあります。



重要

Seam 2.2 で Hibernate を直接使用するアプリケーションは、アプリケーション内部にパッケージ化された Hibernate 3 のバージョンを使用することがあります。JBoss EAP 6 の org.hibernate モジュールを介して提供される Hibernate 4 は、Seam 2.2 によってサポートされません。この例の目的は、最初の手順として JBoss EAP 6 でアプリケーションを実行させることです。Hibernate 3 を Seam 2.2 アプリケーションでパッケージ化する設定はサポートされないことに注意してください。

手順4.6 Seam 2.2 JPA サンプルの移行

1. **jboss-web.xml** ファイルを削除する
`jboss-seam-jpa.war/WEB-INF/` ディレクトリーより **jboss-web.xml** ファイルを削除します。**jboss-web.xml** に定義されるクラスローディングがデフォルトの挙動になります。
2. 以下のように `jboss-seam-jpa.jar/META-INF/persistence.xml` ファイルを変更する
 - a. `jboss-seam-jpa.war/WEB-INF/classes/META-INF/persistence.xml` ファイルの `hibernate.cache.provider_class` プロパティを削除またはコメントアウトします。

```
<!-- <property name="hibernate.cache.provider_class"
value="org.hibernate.cache.HashtableCacheProvider"/> -->
```
 - b. プロバイダーモジュールプロパティを `jboss-seam-booking.jar/META-INF/persistence.xml` ファイルに追加します。

```
<property name="jboss.as.jpa.providerModule" value="hibernate3-bundled" />
```

- c. **jta-data-source** プロパティを変更し、デフォルトの JDBC データソース JNDI 名を使用するようにします。

```
<jta-data-source>java:jboss/datasources/ExampleDS</jta-data-source>
```

3. Seam 2.2 依存関係を追加する

下記の JAR を Seam 2.2 ディストリビューションのライブラリーである **SEAM_HOME/lib/** から **jboss-seam-jpa.war/WEB-INF/lib/** ディレクトリーへコピーします。

- o antlr.jar
- o slf4j-api.jar
- o slf4j-log4j12.jar
- o hibernate-entitymanager.jar
- o hibernate-core.jar
- o hibernate-annotations.jar
- o hibernate-commons-annotations.jar
- o hibernate-validator.jar

4. 残りの依存関係を追加するため jboss-deployment-structure ファイルを作成する

以下のデータを含む **jboss-deployment-structure.xml** ファイルを **jboss-seam-jpa.war/WEB-INF/** フォルダーで作成します。

```
<jboss-deployment-structure>
  <deployment>
    <exclusions>
      <module name="javax.faces.api" slot="main"/>
      <module name="com.sun.jsf-impl" slot="main"/>
      <module name="org.hibernate" slot="main"/>
    </exclusions>
    <dependencies>
      <module name="org.apache.log4j" />
      <module name="org.dom4j" />
      <module name="org.apache.commons.logging" />
      <module name="org.apache.commons.collections" />
      <module name="javax.faces.api" slot="1.2"/>
      <module name="com.sun.jsf-impl" slot="1.2"/>
    </dependencies>
  </deployment>
</jboss-deployment-structure>
```

結果

Seam 2.2 JAP のサンプルアプリケーションが JBoss EAP 6 上にデプロイされ、正常に実行されます。

[バグを報告する](#)

4.3.3. Seam 2.2 Booking サンプルの JBoss EAP 6 への移行

概要

Seam 2.2 Booking EAR の移行は Seam 2.2 JPA WAR サンプルの移行よりも複雑です。Seam 2.2 JPA WAR サンプルの移行に関するドキュメントについては、[「Seam 2.2 JPA サンプルの JBoss EAP 6 への移行」](#)を参照してください。アプリケーションを移行するには、以下を実行する必要があります。

1. デフォルトの JSF 2 の代わりに JSF 1.2 を初期化します。
2. JBoss EAP 6 に同梱された Hibernate JAR を使用せずに、古いバージョンの Hibernate JAR をバンドルします。
3. 新しい Java EE 6 JNDI の移植可能な構文を使用するよう JNDI バインディングを変更します。

1. と 2. は、Seam 2.2 JPA WAR サンプルの移行で行われました。3. は新しい手順で、EAR には EJB が含まれるため必要になります。



重要

Seam 2.2 で Hibernate を直接使用するアプリケーションは、アプリケーション内部にパッケージ化された Hibernate 3 のバージョンを使用することがあります。JBoss EAP 6 の org.hibernate モジュールを介して提供される Hibernate 4 は、Seam 2.2 によってサポートされません。この例の目的は、最初の手順として JBoss EAP 6 でアプリケーションを実行させることです。Hibernate 3 を Seam 2.2 アプリケーションでパッケージ化する設定はサポートされないことに注意してください。

手順4.7 Seam 2.2 Booking サンプルの移行

1. **jboss-deployment-structure.xml** ファイルを作成する
jboss-deployment-structure.xml という名前の新しいファイルを **jboss-seam-booking.ear/META-INF/** で作成し、以下の内容を追加します。

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-structure:1.0">
  <deployment>
    <dependencies>
      <module name="javax.faces.api" slot="1.2" export="true"/>
      <module name="com.sun.jsf-impl" slot="1.2"
export="true"/>
      <module name="org.apache.log4j" export="true"/>
      <module name="org.dom4j" export="true"/>
      <module name="org.apache.commons.logging" export="true"/>
      <module name="org.apache.commons.collections"
export="true"/>
    </dependencies>
    <exclusions>
      <module name="org.hibernate" slot="main"/>
    </exclusions>
  </deployment>
  <sub-deployment name="jboss-seam-booking.war">
    <exclusions>
      <module name="javax.faces.api" slot="main"/>
      <module name="com.sun.jsf-impl" slot="main"/>
    </exclusions>
    <dependencies>
      <module name="javax.faces.api" slot="1.2"/>
    </dependencies>
  </sub-deployment>
</jboss-deployment-structure>
```

```

        <module name="com.sun.jsf-impl" slot="1.2"/>
    </dependencies>
</sub-deployment>
</jboss-deployment-structure>

```

2. 以下のように **jboss-seam-booking.jar/META-INF/persistence.xml** ファイルを変更する

- a. キャッシュプロバイダークラスの hibernate プロパティを削除またはコメントアウトします。

```

<!-- <property name="hibernate.cache.provider_class"
value="org.hibernate.cache.HashtableCacheProvider"/> -->

```

- b. プロバイダーモジュールプロパティを **jboss-seam-booking.jar/META-INF/persistence.xml** ファイルに追加します。

```

<property name="jboss.as.jpa.providerModule" value="hibernate3-
bundled" />

```

- c. **jta-data-source** プロパティを変更し、デフォルトの JDBC データソース JNDI 名を使用するようにします。

```

<jta-data-source>java:jboss/datasources/ExampleDS</jta-data-
source>

```

3. Seam 2.2 ディストリビューションより JAR をコピーする

以下の JAR を Seam 2.2 ディストリビューションの **EAP5.x_HOME/jboss-eap5.x/seam/lib/** から **jboss-seam-booking.ear/lib** ディレクトリにコピーします。

```

antlr.jar
slf4j-api.jar
slf4j-log4j12.jar
hibernate-core.jar
hibernate-entitymanager.jar
hibernate-validator.jar
hibernate-annotations.jar
hibernate-commons-annotations.jar

```

4. JNDI ルックアップ名を変更する

jboss-seam-booking.war/WEB-INF/components.xml ファイルの JNDI ルックアップ文字列を変更します。新しい移植可能な JNDI のルールが導入されたため、JBoss EAP 6 は移植可能な JNDI の構文ルールを使用して EJB をバインドします。JBoss EAP 5 で使用された単一の `jndiPattern` を使用することはできません。JBoss EAP 6 では、アプリケーションの EJB JNDI ルックアップ文字列を次のように変更する必要があります。

```

java:global/jboss-seam-booking/jboss-seam-
booking/HotelSearchingAction!org.jboss.seam.example.booking.HotelSea
rching
java:app/jboss-seam-
booking/HotelSearchingAction!org.jboss.seam.example.booking.HotelSea
rching
java:module/HotelSearchingAction!org.jboss.seam.example.booking.Hote

```

```

lSearching
java:global/jboss-seam-booking/jboss-seam-
booking/HotelSearchingAction
java:app/jboss-seam-booking/HotelSearchingAction
java:module/HotelSearchingAction

```

Seam 2.2 フレームワーク EJB に対する JNDI ルックアップ文字列は、以下のように変更する必要があります。

```

java:global/jboss-seam-booking/jboss-
seam/EjbSynchronizations!org.jboss.seam.transaction.LocalEjbSynchron
izations
java:app/jboss-
seam/EjbSynchronizations!org.jboss.seam.transaction.LocalEjbSynchron
izations
java:module/EjbSynchronizations!org.jboss.seam.transaction.LocalEjbS
ynchronizations
java:global/jboss-seam-booking/jboss-seam/EjbSynchronizations
java:app/jboss-seam/EjbSynchronizations
java:module/EjbSynchronizations

```

以下のどちらかの方法を使用できます。

a. コンポーネント要素を追加する

各 EJB に対する **jndi-name** を **WEB-INF/components.xml** に追加できます。

```

<component
class="org.jboss.seam.transaction.EjbSynchronizations" jndi-
name="java:app/jboss-seam/EjbSynchronizations"/>
<component
class="org.jboss.seam.async.TimerServiceDispatcher" jndi-
name="java:app/jboss-seam/TimerServiceDispatcher"/>
<component
class="org.jboss.seam.example.booking.AuthenticatorAction" jndi-
name="java:app/jboss-seam-booking/AuthenticatorAction" />
<component
class="org.jboss.seam.example.booking.BookingListAction" jndi-
name="java:app/jboss-seam-booking/BookingListAction" />
<component
class="org.jboss.seam.example.booking.RegisterAction" jndi-
name="java:app/jboss-seam-booking/RegisterAction" />
<component
class="org.jboss.seam.example.booking.HotelSearchingAction" jndi-
name="java:app/jboss-seam-booking/HotelSearchingAction" />
<component
class="org.jboss.seam.example.booking.HotelBookingAction" jndi-
name="java:app/jboss-seam-booking/HotelBookingAction" />
<component
class="org.jboss.seam.example.booking.ChangePasswordAction" jndi-
name="java:app/jboss-seam-booking/ChangePasswordAction" />

```

- b. JNDI パスを指定する **@JNDIName(value="")** アノテーションを追加してコードを変更することができます。変更されたステートレスセッション Bean のコード例は次のとおりです。この処理の詳細については、Seam 2.2 のリファレンスドキュメントを参照してください。

```

@Stateless
@Name("authenticator")
@JndiName(value="java:app/jboss-seam-
booking/AuthenticatorAction")
public class AuthenticatorAction
    implements Authenticator
{
    ...
}

```

結果

Seam 2.2 JAP の Booking アプリケーションが JBoss EAP 6 上にデプロイされ、正常に実行されます。

[バグを報告する](#)

4.3.4. Seam 2.2 Booking アーカイブの JBoss EAP 6 への移行: 手順説明

ここでは、Seam 2.2 Booking アプリケーションアーカイブを JBoss EAP 5.X から JBoss EAP 6 へ移植する方法をステップごとに説明します。アプリケーションの移行により適した方法が存在しますが、アプリケーションアーカイブをそのまま JBoss EAP 6 のサーバーへデプロイし、結果を見たい開発者も多くいるはずで、アプリケーションアーカイブをそのままデプロイすると発生する可能性がある問題や、そのデバッグ方法および解決方法について解説することを目的としています。

この例ではアプリケーション EAR は **EAP6_HOME/standalone/deployments** ディレクトリーにデプロイされ、変更はアーカイブの変更のみとなります。これにより、問題の発生や解決時にアーカイブ内にある XML ファイルの変更が容易になります。



重要

Seam 2.2 で Hibernate を直接使用するアプリケーションは、アプリケーション内部にパッケージ化された Hibernate 3 のバージョンを使用することがあります。JBoss EAP 6 の org.hibernate モジュールを介して提供される Hibernate 4 は、Seam 2.2 によってサポートされません。この例の目的は、最初の手順として JBoss EAP 6 でアプリケーションを実行させることです。Hibernate 3 を Seam 2.2 アプリケーションでパッケージ化する設定はサポートされないことに注意してください。

手順4.8 アプリケーションの移行

1. 「JBoss EAP 5.X バージョンの Seam 2.2 Booking アプリケーションのビルドおよびデプロイ」に従ってビルドおよびデプロイします。
2. 「Seam 2.2 Booking アーカイブのデプロイメントエラーや例外のデバッグおよび解決」に従ってデプロイメントエラーや例外のデバッグおよび解決を行います。
3. 「Seam 2.2 Booking アーカイブのランタイムエラーや例外のデバッグおよび解決」に従ってランタイムエラーや例外のデバッグおよび解決を行います。

この時点で URL <http://localhost:8080/seam-booking/> をブラウザで指定してアプリケーションにアクセスすることができます。demo/demo でログインすると Booking のウェルカムページが表示されます。

変更概要の確認

「Seam 2.2 Booking アプリケーションの移行時に加えられる変更概要の確認」に従って変更の概要を確認します。

[バグを報告する](#)

4.3.5. JBoss EAP 5.X バージョンの Seam 2.2 Booking アプリケーションのビルドおよびデプロイ

このアプリケーションを移行する前に、JBoss EAP 5.X の Seam 2.2 Booking アプリケーションをビルドし、アーカイブを抽出してから JBoss EAP 6 のデプロイメントフォルダーへコピーします。

手順4.9 EAR のビルドとデプロイ

1. EAR をビルドします。

```
$ cd /EAP5_HOME/jboss-eap5.x/seam/examples/booking
$ ANT_HOME/ant explode
```

jboss-eap5.x を移行元の JBoss EAP バージョンに置き換えます。

2. EAR を *EAP6_HOME* デプロイメントディレクトリーへコピーします。

```
$ cp -r EAP5_HOME/seam/examples/booking/exploded-archives/jboss-
seam-booking.ear EAP6_HOME/standalone/deployments/
$ cp -r EAP5_HOME/seam/examples/booking/exploded-archives/jboss-
seam-booking.war EAP6_HOME/standalone/deployments/jboss-seam.ear
$ cp -r EAP5_HOME/seam/examples/booking/exploded-archives/jboss-
seam-booking.jar EAP6_HOME/standalone/deployments/jboss-seam.ear
```

3. JBoss EAP 6 サーバーを起動し、ログをチェックします。ログには以下が記録されているはずです。

```
INFO [org.jboss.as.deployment] (DeploymentScanner-threads - 1) Found
jboss-seam-booking.ear in deployment directory.
    To trigger deployment create a file called jboss-seam-
booking.ear.dodeploy
```

4. **jboss-seam-booking.ear.dodeploy** という名前の空のファイルを作成し、**EAP6_HOME/standalone/deployments** ディレクトリーへコピーします。本アプリケーションの移行中に、このファイルを複数回デプロイメントディレクトリーへコピーする必要があるため、簡単に見つかる場所へ保存するようにしてください。デプロイ中であることを示す次のメッセージがログに記録されるはずです。

```
INFO [org.jboss.as.server.deployment] (MSC service thread 1-1)
Starting deployment of "jboss-seam-booking.ear"
INFO [org.jboss.as.server.deployment] (MSC service thread 1-3)
Starting deployment of "jboss-seam-booking.jar"
INFO [org.jboss.as.server.deployment] (MSC service thread 1-6)
Starting deployment of "jboss-seam.jar"
INFO [org.jboss.as.server.deployment] (MSC service thread 1-2)
Starting deployment of "jboss-seam-booking.war"
```

この時点で最初のデプロイメントエラーが発生します。次の手順で各問題を確認し、デバッグおよび解決方法について説明します。

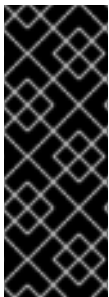
デプロイメントの問題のデバッグおよび解決方法については、「[Seam 2.2 Booking アーカイブのデプロイメントエラーや例外のデバッグおよび解決](#)」をクリックしてください。

以前のトピックに戻るには、「[Seam 2.2 Booking アーカイブの JBoss EAP 6 への移行: 手順説明](#)」をクリックしてください。

バグを報告する

4.3.6. Seam 2.2 Booking アーカイブのデプロイメントエラーや例外のデバッグおよび解決

前述の手順、「[JBoss EAP 5.X バージョンの Seam 2.2 Booking アプリケーションのビルドおよびデプロイ](#)」では JBoss EAP 5.X の Seam 2.2 Booking アプリケーションを構築し、JBoss EAP 6 のデプロイメントフォルダーにデプロイしました。この手順では発生したデプロイメントエラーをデバッグし解決します。



重要

Seam 2.2 で Hibernate を直接使用するアプリケーションは、アプリケーション内部にパッケージ化された Hibernate 3 のバージョンを使用することがあります。JBoss EAP 6 の org.hibernate モジュールを介して提供される Hibernate 4 は、Seam 2.2 によってサポートされません。この例の目的は、最初の手順として JBoss EAP 6 でアプリケーションを実行させることです。Hibernate 3 を Seam 2.2 アプリケーションでパッケージ化する設定はサポートされないことに注意してください。

手順4.10 デプロイメントエラーや例外のデバッグおよび解決

1. 問題 - java.lang.ClassNotFoundException: javax.faces.FacesException

アプリケーションをデプロイする場合に、ログに以下のエラーが含まれます。

```
ERROR \[org.jboss.msc.service.fail\] (MSC service thread 1-1)
MSC00001: Failed to start service jboss.deployment.subunit."jboss-seam-booking.ear"."jboss-seam-booking.war".POST_MODULE:
org.jboss.msc.service.StartException in service
jboss.deployment.subunit."jboss-seam-booking.ear"."jboss-seam-booking.war".POST_MODULE:
Failed to process phase POST_MODULE of subdeployment "jboss-seam-booking.war" of deployment "jboss-seam-booking.ear"
(.. additional logs removed ...)
Caused by: java.lang.ClassNotFoundException:
javax.faces.FacesException from \[Module "deployment.jboss-seam-booking.ear:main" from Service Module Loader\]
at
org.jboss.modules.ModuleClassLoader.findClass(ModuleClassLoader.java:191)
```

ログの解説

ClassNotFoundException は見つからない依存関係があることを示しています。この例では、クラス **javax.faces.FacesException** が見つからないため、依存関係を明示的に追加する必要があります。

解決方法

見つからないクラスと一致するパスを探し、**EAP6_HOME/modules/system/layers/base/**ディレクトリー内でそのクラスのモジュール名を見つけます。この例では、以下の2つのモジュールが見つかります。

```
javax/faces/api/main
javax/faces/api/1.2
```

両モジュールのモジュール名は同じ **javax.faces.api** ですが、メインディレクトリーにあるモジュールは JSF 2.0 向けで、1.2 ディレクトリーにあるものは JSF 1.2 向けです。一致するモジュールが1つのみの場合、**MANIFEST.MF** ファイルを作成し、モジュールの依存関係を追加します。この例では、メインディレクトリーにある 2.0 バージョンではなく JSF 1.2 バージョンを使用したいため、使用したい方を指定し、使用したくない方を除外します。これを行うには、EAR の **META-INF/** ディレクトリーに、次のデータが含まれる **jboss-deployment-structure.xml** ファイルを作成します。

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-
structure:1.0">
  <deployment>
    <dependencies>
      <module name="javax.faces.api" slot="1.2" export="true"/>
    </dependencies>
  </deployment>
  <sub-deployment name="jboss-seam-booking.war">
    <exclusions>
      <module name="javax.faces.api" slot="main"/>
    </exclusions>
    <dependencies>
      <module name="javax.faces.api" slot="1.2"/>
    </dependencies>
  </sub-deployment>
</jboss-deployment-structure>
```

deployment セクションで、JSF 1.2 モジュール用 **javax.faces.api** に対する依存関係を追加します。また、WAR のサブデプロイメントセクションで JSF 1.2 モジュール用依存関係を追加し、JSF 2.0 用モジュールを除外します。

EAP6_HOME/standalone/deployments/jboss-seam-booking.ear.failed ファイルを削除して同じディレクトリーに空の **jboss-seam-booking.ear.dodeploy** ファイルを作成し、アプリケーションを再デプロイします。

2. 問題 - java.lang.ClassNotFoundException: org.apache.commons.logging.Log

アプリケーションをデプロイする場合に、ログに以下のエラーが含まれます。

```
ERROR [org.jboss.msc.service.fail] (MSC service thread 1-8)
MSC00001: Failed to start service jboss.deployment.unit."jboss-seam-
booking.ear".INSTALL:
org.jboss.msc.service.StartException in service
jboss.deployment.unit."jboss-seam-booking.ear".INSTALL:
Failed to process phase INSTALL of deployment "jboss-seam-
booking.ear"
    (... additional logs removed ...)
Caused by: java.lang.ClassNotFoundException:
org.apache.commons.logging.Log from [Module "deployment.jboss-seam-
booking.ear.jboss-seam-booking.war:main" from Service Module Loader]
```

ログの解説

ClassNotFoundException は見つからない依存関係があることを示しています。この例では、クラス **org.apache.commons.logging.Log** が見つからないため、依存関係を明示的に追加する必要があります。

解決方法

見つからないクラスと一致するパスを探し、**EAP6_HOME/modules/system/layers/base/**ディレクトリ内でそのクラスのモジュール名を見つけます。この例では、パス **org/apache/commons/logging/** と一致するモジュールが1つあります。モジュール名は **"org.apache.commons.logging"**です。

モジュール依存関係をファイルのデプロイメントセクションに追加するよう **jboss-deployment-structure.xml** ファイルを変更します。

```
<module name="org.apache.commons.logging" export="true"/>
```

jboss-deployment-structure.xml の内容は以下のようにになります。

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-structure:1.0">
  <deployment>
    <dependencies>
      <module name="javax.faces.api" slot="1.2" export="true"/>
      <module name="org.apache.commons.logging" export="true"/>
    </dependencies>
  </deployment>
  <sub-deployment name="jboss-seam-booking.war">
    <exclusions>
      <module name="javax.faces.api" slot="main"/>
    </exclusions>
    <dependencies>
      <module name="javax.faces.api" slot="1.2"/>
    </dependencies>
  </sub-deployment>
</jboss-deployment-structure>
```

EAP6_HOME/standalone/deployments/jboss-seam-booking.ear.failed ファイルを削除して同じディレクトリに空の **jboss-seam-booking.ear.dodeploy** ファイルを作成し、アプリケーションを再デプロイします。

3. 問題 - java.lang.ClassNotFoundException: org.dom4j.DocumentException

アプリケーションをデプロイする場合に、ログに以下のエラーが含まれます。

```
ERROR [org.apache.catalina.core.ContainerBase.[jboss.web].[default-host].[/seam-booking]] (MSC service thread 1-3) Exception sending context initialized event to listener instance of class org.jboss.seam.servlet.SeamListener: java.lang.NoClassDefFoundError: org/dom4j/DocumentException
(... additional logs removed ...)
Caused by: java.lang.ClassNotFoundException: org.dom4j.DocumentException from [Module "deployment.jboss-seam-booking.ear.jboss-seam.jar:main" from Service Module Loader]
```

ログの解説

ClassNotFoundException は見つからない依存関係があることを示しています。この例では、クラス **org.dom4j.DocumentException** が見つかりません。

解決方法

org/dom4j/DocumentException を探し

て、**EAP6_HOME/modules/system/layers/base/**ディレクトリーを見つけます。モジュール名は、“org.dom4j”です。モジュール依存関係をファイルのデプロイメントセクションに追加するよう **jboss-deployment-structure.xml** ファイルを変更します。

```
<module name="org.dom4j" export="true"/>
```

jboss-deployment-structure.xml ファイルの内容は以下のようになります。

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-structure:1.0">
  <deployment>
    <dependencies>
      <module name="javax.faces.api" slot="1.2" export="true"/>
      <module name="org.apache.commons.logging" export="true"/>
      <module name="org.dom4j" export="true"/>
    </dependencies>
  </deployment>
  <sub-deployment name="jboss-seam-booking.war">
    <exclusions>
      <module name="javax.faces.api" slot="main"/>
    </exclusions>
    <dependencies>
      <module name="javax.faces.api" slot="1.2"/>
    </dependencies>
  </sub-deployment>
</jboss-deployment-structure>
```

EAP6_HOME/standalone/deployments/jboss-seam-booking.ear.failed ファイルを削除して同じディレクトリーに空の **jboss-seam-booking.ear.dodeploy** ファイルを作成し、アプリケーションを再デプロイします。

4. 問題 - java.lang.ClassNotFoundException: org.hibernate.validator.InvalidValue

アプリケーションをデプロイする場合に、ログに以下のエラーが含まれます。

```
ERROR [org.apache.catalina.core.ContainerBase.[jboss.web].[default-host].[/seam-booking]] (MSC service thread 1-6) Exception sending context initialized event to listener instance of class org.jboss.seam.servlet.SeamListener: java.lang.RuntimeException: Could not create Component: org.jboss.seam.international.statusMessages (... additional logs removed ...)
Caused by: java.lang.ClassNotFoundException: org.hibernate.validator.InvalidValue from [Module "deployment.jboss-seam-booking.ear.jboss-seam.jar:main" from Service Module Loader]
```

ログの解説

ClassNotFoundException は見つからない依存関係があることを示しています。この例では、クラス **org.hibernate.validator.InvalidValue** が見つかりません。

解決方法

モジュール「org.hibernate.validator」は存在しますが JAR に **org.hibernate.validator.InvalidValue** クラスが含まれていないため、モジュールの依存関係を追加してもこの問題は解決しません。この例では、クラスが含まれる JAR は JBoss EAP 5.X デプロイメントの一部になります。**EAP5_HOME/seam/lib/** ディレクトリーに不明なクラスが含まれている JAR を探します。これを実行するには、コンソールを開いて以下の内容を入力します。

```
$ cd EAP5_HOME/seam/lib
$ grep 'org.hibernate.validator.InvalidValue' `find . -name '*.jar'`
```

結果は以下のようになります。

```
$ Binary file ./hibernate-validator.jar matches
$ Binary file ./test/hibernate-all.jar matches
```

この場合は、**hibernate-validator.jar** を **jboss-seam-booking.ear/lib/** ディレクトリーにコピーします。

```
$ cp EAP5_HOME/seam/lib/hibernate-validator.jar jboss-seam-booking.ear/lib
```

EAP6_HOME/standalone/deployments/jboss-seam-booking.ear.failed ファイルを削除して同じディレクトリーに空の **jboss-seam-booking.ear.dodeploy** ファイルを作成し、アプリケーションを再デプロイします。

5. 問題 - java.lang.InstantiationException: org.jboss.seam.jsf.SeamApplicationFactory

アプリケーションをデプロイする場合に、ログに以下のエラーが含まれます。

```
INFO [javax.enterprise.resource.webcontainer.jsf.config] (MSC
service thread 1-7) Unsanitized stacktrace from failed start...:
com.sun.faces.config.ConfigurationException: Factory
'javax.faces.application.ApplicationFactory' was not configured
properly.
    at
com.sun.faces.config.processor.FactoryConfigProcessor.verifyFactorie
sExist(FactoryConfigProcessor.java:296) [jsf-impl-2.0.4-b09-
jbossorg-4.jar:2.0.4-b09-jbossorg-4]
    (... additional logs removed ...)
Caused by: javax.faces.FacesException:
org.jboss.seam.jsf.SeamApplicationFactory
    at
javax.faces.FactoryFinder.getImplGivenPreviousImpl(FactoryFinder.jav
a:606) [jsf-api-1.2_13.jar:1.2_13-b01-FCS]
    (... additional logs removed ...)
    at
com.sun.faces.config.processor.FactoryConfigProcessor.verifyFactorie
sExist(FactoryConfigProcessor.java:294) [jsf-impl-2.0.4-b09-
jbossorg-4.jar:2.0.4-b09-jbossorg-4]
    ... 11 more
```

```

Caused by: java.lang.InstantiationException:
org.jboss.seam.jsf.SeamApplicationFactory
  at java.lang.Class.newInstance0(Class.java:340) [:1.6.0_25]
  at java.lang.Class.newInstance(Class.java:308) [:1.6.0_25]
  at
javax.faces.FactoryFinder.getImplGivenPreviousImpl(FactoryFinder.java:604) [jsf-api-1.2_13.jar:1.2_13-b01-FCS]
... 16 more

```

ログの解説

com.sun.faces.config.ConfigurationException と **java.lang.InstantiationException** は依存関係の問題があることを示しています。この例では、原因は明らかではありません。

解決方法

com.sun.faces クラスが含まれるモジュールを探す必要があります。**com.sun.faces** モジュールは存在しませんが、**com.sun.jsf-impl** モジュールが2つあります。1.2 ディレクトリーの **jsf-impl-1.2_13.jar** を確認すると、**com.sun.faces** クラスが含まれていることがすぐに分かります。**javax.faces.FacesException ClassNotFoundException** の場合と同様、メインディレクトリーの JSF 2.0 バージョンではなく JFS 1.2 バージョンを使用したいため、使用したい方を指定し、使用したくない方を除外します。ファイルのデプロイメントセクションにモジュールの依存関係を追加するよう **jboss-deployment-structure.xml** を変更する必要があります。また、WAR のサブデプロイメントに追加し、JSF 2.0 モジュールを除外する必要もあります。ファイルの内容は次のようになるはずです。

```

<jboss-deployment-structure xmlns="urn:jboss:deployment-structure:1.0">
  <deployment>
    <dependencies>
      <module name="javax.faces.api" slot="1.2" export="true"/>
      <module name="com.sun.jsf-impl" slot="1.2"
export="true"/>
      <module name="org.apache.commons.logging" export="true"/>
      <module name="org.dom4j" export="true"/>
    </dependencies>
  </deployment>
  <sub-deployment name="jboss-seam-booking.war">
    <exclusions>
      <module name="javax.faces.api" slot="main"/>
      <module name="com.sun.jsf-impl" slot="main"/>
    </exclusions>
    <dependencies>
      <module name="javax.faces.api" slot="1.2"/>
      <module name="com.sun.jsf-impl" slot="1.2"/>
    </dependencies>
  </sub-deployment>
</jboss-deployment-structure>

```

EAP6_HOME/standalone/deployments/jboss-seam-booking.ear.failed ファイルを削除して同じディレクトリーに空の **jboss-seam-booking.ear.dodeploy** ファイルを作成し、アプリケーションを再デプロイします。

6. 問題 - java.lang.ClassNotFoundException: org.apache.commons.collections.ArrayStack

アプリケーションをデプロイする場合に、ログに以下のエラーが含まれます。

```
ERROR [org.apache.catalina.core.ContainerBase.[jboss.web].[default-
host].[/seam-booking]] (MSC service thread 1-1) Exception sending
context initialized event to listener instance of class
com.sun.faces.config.ConfigureListener: java.lang.RuntimeException:
com.sun.faces.config.ConfigurationException: CONFIGURATION FAILED!
org.apache.commons.collections.ArrayStack from [Module
"deployment.jboss-seam-booking.ear:main" from Service Module Loader]
(... additional logs removed ...)
Caused by: java.lang.ClassNotFoundException:
org.apache.commons.collections.ArrayStack from [Module
"deployment.jboss-seam-booking.ear:main" from Service Module Loader]
```

ログの解説

ClassNotFoundException は見つからない依存関係があることを示しています。この例では、クラス **org.apache.commons.collections.ArrayStack** が見つかりません。

解決方法

org/apache/commons/collections パスを探し

て、**EAP6_HOME/modules/system/layers/base/** ディレクトリーを見つけます。モジュール名は、"org.apache.commons.collections" です。モジュール依存関係をファイルのデプロイメントセクションに追加するよう **jboss-deployment-structure.xml** を変更します。

```
<module name="org.apache.commons.collections" export="true"/>
```

jboss-deployment-structure.xml ファイルの内容は以下のようになります。

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-
structure:1.0">
  <deployment>
    <dependencies>
      <module name="javax.faces.api" slot="1.2" export="true"/>
      <module name="com.sun.jsf-impl" slot="1.2"
export="true"/>
      <module name="org.apache.commons.logging" export="true"/>
      <module name="org.dom4j" export="true"/>
      <module name="org.apache.commons.collections"
export="true"/>
    </dependencies>
  </deployment>
  <sub-deployment name="jboss-seam-booking.war">
    <exclusions>
      <module name="javax.faces.api" slot="main"/>
      <module name="com.sun.jsf-impl" slot="main"/>
    </exclusions>
    <dependencies>
      <module name="javax.faces.api" slot="1.2"/>
      <module name="com.sun.jsf-impl" slot="1.2"/>
    </dependencies>
  </sub-deployment>
</jboss-deployment-structure>
```

EAP6_HOME/standalone/deployments/jboss-seam-booking.ear.failed ファイルを

削除して同じディレクトリーに空の **jboss-seam-booking.ear.dodeploy** ファイルを作成し、アプリケーションを再デプロイします。

7. 問題 - Services with missing/unavailable dependencies

アプリケーションをデプロイする場合に、ログに以下のエラーが含まれます。

```
ERROR [org.jboss.as.deployment] (DeploymentScanner-threads - 2)
{"Composite operation failed and was rolled back. Steps that
failed:" => {"Operation step-2" => {"Services with
missing/unavailable dependencies" =>
["jboss.deployment.subunit.\"jboss-seam-booking.ear\".\"jboss-seam-
booking.jar\".component.AuthenticatorAction.START missing [
jboss.naming.context.java.comp.jboss-seam-booking.\"jboss-seam-
booking.jar\".AuthenticatorAction.\"env/org.jboss.seam.example.booki
ng.AuthenticatorAction/em\" ]\",\"jboss.deployment.subunit.\"jboss-
seam-booking.ear\".\"jboss-seam-
booking.jar\".component.HotelSearchingAction.START missing [
jboss.naming.context.java.comp.jboss-seam-booking.\"jboss-seam-
booking.jar\".HotelSearchingAction.\"env/org.jboss.seam.example.book
ing.HotelSearchingAction/em\" ]\",\"
(... additional logs removed ...)
jboss.deployment.subunit.\"jboss-seam-booking.ear\".\"jboss-seam-
booking.jar\".component.BookingListAction.START missing [
jboss.naming.context.java.comp.jboss-seam-booking.\"jboss-seam-
booking.jar\".BookingListAction.\"env/org.jboss.seam.example.booking
.BookingListAction/em\" ]\",\"jboss.persistenceunit.\"jboss-seam-
booking.ear/jboss-seam-booking.jar#bookingDatabase\" missing [
jboss.naming.context.java.bookingDatasource ]"]}}}
```

ログの解説

「Services with missing/unavailable dependencies」(見つからない/使用できない依存関係を持つサービス)のエラーが発生したら、「missing」の後の括弧内にある文字を確認してください。この場合は、次のようになります。

```
missing [ jboss.naming.context.java.comp.jboss-seam-booking.\"jboss-
seam-
booking.jar\".AuthenticatorAction.\"env/org.jboss.seam.example.booki
ng.AuthenticatorAction/em\" ]
```

「/em」は、Entity Manager とデータソースの問題を示します。

解決方法

JBoss EAP 6 ではデータソースの設定が変更になったため、**EAP6_HOME/standalone/configuration/standalone.xml** ファイルに定義する必要があります。JBoss EAP 6 には、すでに **standalone.xml** ファイルに定義されているデータベースの例が含まれているため、このアプリケーションでサンプルデータベースを使用するよう **persistence.xml** ファイルを変更します。**standalone.xml** ファイルを見るとサンプルデータベースの **jndi-name** は **java:jboss/datasources/ExampleDS** であることが分かります。**jboss-seam-booking.jar/META-INF/persistence.xml** ファイルを変更して既存の **jta-data-source** 要素をコメントアウトし、以下のように置き換えます。

```
<!-- <jta-data-source>java:/bookingDatasource</jta-data-source> -->
<jta-data-source>java:jboss/datasources/ExampleDS</jta-data-source>
```

EAP6_HOME/standalone/deployments/jboss-seam-booking.ear.failed ファイルを削除して同じディレクトリーに空の **jboss-seam-booking.ear.dodeploy** ファイルを作成し、アプリケーションを再デプロイします。

- この時点で、アプリケーションはエラーを引き起こさずにデプロイされますが、ブラウザで URL <http://localhost:8080/seam-booking/> へアクセスし、アカウントへログインしようするとランタイムエラー「The page isn't redirecting properly」(ページが正しくリダイレクトしません)が発生します。次の手順でランタイムエラーのデバッグおよび解決方法について学びましょう。

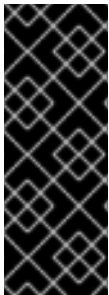
ランタイム問題のデバッグおよび解決方法については、「[Seam 2.2 Booking アーカイブのランタイムエラーや例外のデバッグおよび解決](#)」を参照してください。

以前のトピックに戻るには、「[Seam 2.2 Booking アーカイブの JBoss EAP 6 への移行: 手順説明](#)」をクリックしてください。

バグを報告する

4.3.7. Seam 2.2 Booking アーカイブのランタイムエラーや例外のデバッグおよび解決

前述の手順、「[Seam 2.2 Booking アーカイブのデプロイメントエラーや例外のデバッグおよび解決](#)」ではデプロイメントエラーのデバッグ方法について説明しました。この手順では発生したランタイムエラーをデバッグし解決します。



重要

Seam 2.2 で Hibernate を直接使用するアプリケーションは、アプリケーション内部にパッケージ化された Hibernate 3 のバージョンを使用することがあります。JBoss EAP 6 の org.hibernate モジュールを介して提供される Hibernate 4 は、Seam 2.2 によってサポートされません。この例の目的は、最初の手順として JBoss EAP 6 でアプリケーションを実行させることです。Hibernate 3 を Seam 2.2 アプリケーションでパッケージ化する設定はサポートされないことに注意してください。

手順4.11 ランタイムエラーや例外のデバッグおよび解決

この時点では、アプリケーションをデプロイしてもログにエラーは記録されていませんが、アプリケーションの URL にアクセスするとエラーがログに記録されます。

- 問題 - javax.naming.NameNotFoundException: Name 'jboss-seam-booking' not found in context "

ブラウザで URL <http://localhost:8080/seam-booking/> にアクセスすると、「The page isn't redirecting properly (ページが正常にリダイレクトされていません)」というメッセージが表示され、ログに以下のエラーが記録されます。

```
SEVERE [org.jboss.seam.jsf.SeamPhaseListener] (http--127.0.0.1-8080-1) swallowing exception: java.lang.IllegalStateException: Could not start transaction
    at
    org.jboss.seam.jsf.SeamPhaseListener.begin(SeamPhaseListener.java:598) [jboss-seam.jar:]
    (... log messages removed ...)
Caused by: org.jboss.seam.InstantiationException: Could not instantiate Seam component:
```

```
org.jboss.seam.transaction.synchronizations
    at org.jboss.seam.Component.newInstance(Component.java:2170)
[jboss-seam.jar:]
    (... log messages removed ...)
Caused by: javax.naming.NameNotFoundException: Name 'jboss-seam-
booking' not found in context ''
    at
org.jboss.as.naming.util.NamingUtils.nameNotFoundException(NamingUtil
s.java:109)
    (... log messages removed ...)
```

ログの解説

NameNotFoundException は JNDI の命名の問題であることを示しています。JBoss EAP 6 では JNDI の命名ルールが変更になったため、新しいルールに従ってルックアップ名を変更する必要があります。

解決方法

これをデバッグするには、使用された JNDI バインディングに対するサーバーログトレースを確認します。サーバーログを確認すると、内容は以下のようになります。

```
15:01:16,138 INFO
[org.jboss.as.ejb3.deployment.processors.EjbJndiBindingsDeploymentUn
itProcessor] (MSC service thread 1-1) JNDI bindings for session bean
named RegisterAction in deployment unit subdeployment "jboss-seam-
booking.jar" of deployment "jboss-seam-booking.ear" are as follows:
    java:global/jboss-seam-booking/jboss-seam-
booking.jar/RegisterAction!org.jboss.seam.example.booking.Register
    java:app/jboss-seam-
booking.jar/RegisterAction!org.jboss.seam.example.booking.Register
    java:module/RegisterAction!org.jboss.seam.example.booking.Register
    java:global/jboss-seam-booking/jboss-seam-
booking.jar/RegisterAction
    java:app/jboss-seam-booking.jar/RegisterAction
    java:module/RegisterAction
[JNDI bindings continue ...]
```

ログには、各セッション Bean に対して 1 つずつ合計で 8 つの INFO JNDI バインディング (RegisterAction、BookingListAction、HotelBookingAction、AuthenticatorAction、ChangePasswordAction、HotelSearchingAction、EjbSynchronizations、および TimerServiceDispatcher) がリストされます。新しい JNDI バインディングを使用するよう WAR の **lib/components.xml** ファイルを変更します。ログで、EJB JNDI バインディングがすべて "java:app/jboss-seam-booking.jar" で始まっていることに注意してください。以下のように **core:init** 要素を置き換えます。

```
<!--      <core:init jndi-pattern="jboss-seam-booking/#
{ejbName}/local" debug="true" distributable="false"/> -->
<core:init jndi-pattern="java:app/jboss-seam-booking.jar/#{ejbName}"
debug="true" distributable="false"/>
```

次に、EjbSynchronizations バインディングと TimerServiceDispatcher JNDI バインディングを追加する必要があります。ファイルに以下のコンポーネント要素を追加します。

```
<component class="org.jboss.seam.transaction.EjbSynchronizations"
jndi-name="java:app/jboss-seam/EjbSynchronizations"/>
```

```
<component class="org.jboss.seam.async.TimerServiceDispatcher" jndi-
name="java:app/jboss-seam/TimerServiceDispatcher"/>
```

components.xml ファイルは以下になるはずです。

```
<?xml version="1.0" encoding="UTF-8"?>
<components xmlns="http://jboss.com/products/seam/components"
  xmlns:core="http://jboss.com/products/seam/core"
  xmlns:security="http://jboss.com/products/seam/security"
  xmlns:transaction="http://jboss.com/products/seam/transaction"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation=
    "http://jboss.com/products/seam/core
http://jboss.com/products/seam/core-2.2.xsd
http://jboss.com/products/seam/transaction
http://jboss.com/products/seam/transaction-2.2.xsd
http://jboss.com/products/seam/security
http://jboss.com/products/seam/security-2.2.xsd
http://jboss.com/products/seam/components
http://jboss.com/products/seam/components-2.2.xsd">

  <!-- <core:init jndi-pattern="jboss-seam-booking/#
{ejbName}/local" debug="true" distributable="false"/> -->
  <core:init jndi-pattern="java:app/jboss-seam-booking.jar/#
{ejbName}" debug="true" distributable="false"/>
  <core:manager conversation-timeout="120000"
    concurrent-request-timeout="500"
    conversation-id-parameter="cid"/>
  <transaction:ejb-transaction/>
  <security:identity authenticate-method="#
{authenticator.authenticate}"/>
  <component
class="org.jboss.seam.transaction.EjbSynchronizations"
    jndi-name="java:app/jboss-seam/EjbSynchronizations"/>
  <component class="org.jboss.seam.async.TimerServiceDispatcher"
    jndi-name="java:app/jboss-
seam/TimerServiceDispatcher"/>
</components>
```

standalone/deployments/jboss-seam-booking.ear.failed ファイルを削除して同じディレクトリに空の **jboss-seam-booking.ear.dodeploy** ファイルを作成し、アプリケーションを再デプロイします。

2. 問題 - エラーが発生せずにアプリケーションがデプロイされ、実行されます。ブラウザで URL <http://localhost:8080/seam-booking/> にアクセスし、ログインしようとすると「Login failed. Transaction failed.」というメッセージが表示され、ログインに失敗します。サーバーログに次のような例外トレースが記録されるはずです。

```
13:36:04,631 WARN [org.jboss.modules] (http-/127.0.0.1:8080-1)
Failed to define class
org.jboss.seam.persistence.HibernateSessionProxy in Module
"deployment.jboss-seam-booking.ear.jboss-seam.jar:main" from Service
Module Loader: java.lang.LinkageError: Failed to link
org/jboss/seam/persistence/HibernateSessionProxy (Module
"deployment.jboss-seam-booking.ear.jboss-seam.jar:main" from Service
Module Loader)
```

```

....
Caused by: java.lang.LinkageError: Failed to link
org/jboss/seam/persistence/HibernateSessionProxy (Module
"deployment.jboss-seam-booking.ear.jboss-seam.jar:main" from Service
Module Loader)
...
Caused by: java.lang.NoClassDefFoundError:
org/hibernate/engine/SessionImplementor
  at java.lang.ClassLoader.defineClass1(Native Method)
[rt.jar:1.7.0_45]
...
Caused by: java.lang.ClassNotFoundException:
org.hibernate.engine.SessionImplementor from [Module
"deployment.jboss-seam-booking.ear.jboss-seam.jar:main" from Service
Module Loader]
...

```

ログの解説

ClassNotFoundException は Hibernate ライブラリーがないことを示しています。この場合、**hibernate-core.jar** が存在しません。

解決方法

hibernate-core.jar JAR を **EAP5_HOME/seam/lib/** ディレクトリーから **jboss-seam-booking.ear/lib** ディレクトリーへコピーします。

standalone/deployments/jboss-seam-booking.ear.failed ファイルを削除して同じディレクトリーに空の **jboss-seam-booking.ear.dodeploy** ファイルを作成し、アプリケーションを再デプロイします。

3. 問題 - エラーが発生せずにアプリケーションがデプロイされ、実行されます。ブラウザで URL <http://localhost:8080/seam-booking/> にアクセスすると正常にログインできますが、ホテルを予約しようとする、例外トレースが記録されます。

これをデバッグするには、真のエラーを隠している **jboss-seam-booking.ear/jboss-seam-booking.war/WEB-INF/lib/jboss-seam-debug.jar** を最初に削除します。この時点で、次のエラーが表示されるはずです。

```

java.lang.NoClassDefFoundError:
org/hibernate/annotations/common/reflection/ReflectionManager

```

ログの解説

NoClassDefFoundError は Hibernate ライブラリーがないことを示しています。

解決方法

hibernate-annotations.jar および **hibernate-commons-annotations.jar** JAR を **EAP5_HOME/seam/lib/** ディレクトリーから **jboss-seam-booking.ear/lib** ディレクトリーへコピーします。

standalone/deployments/jboss-seam-booking.ear.failed ファイルを削除して同じディレクトリーに空の **jboss-seam-booking.ear.dodeploy** ファイルを作成し、アプリケーションを再デプロイします。

4. ランタイムおよびアプリケーションエラーが解決されるはずです。

この時点で、エラーが発生せずにアプリケーションがデプロイおよび実行されます。

以前のトピックに戻るには、[「Seam 2.2 Booking アーカイブの JBoss EAP 6 への移行: 手順説明」](#) をクリックしてください。

[バグを報告する](#)

4.3.8. Seam 2.2 Booking アプリケーションの移行時に加えられる変更概要の確認

事前に依存関係を判断し、暗黙的な依存関係を一度に追加した方が効率的ですが、この例では問題がどのようにログに表示されるかを説明し、デバッグや解決方法に関する情報を提供します。JBoss EAP 6 へ移行するときにアプリケーションに加えられる変更の概要は次のとおりです。



重要

Seam 2.2 で Hibernate を直接使用するアプリケーションは、アプリケーション内部にパッケージ化された Hibernate 3 のバージョンを使用することがあります。JBoss EAP 6 の org.hibernate モジュールを介して提供される Hibernate 4 は、Seam 2.2 によってサポートされません。この例の目的は、最初の手順として JBoss EAP 6 でアプリケーションを実行させることです。Hibernate 3 を Seam 2.2 アプリケーションでパッケージ化する設定はサポートされないことに注意してください。

1. **jboss-deployment-structure.xml** ファイルを、EAR の **META-INF/** ディレクトリーに作成しました。**ClassNotFoundException** を解決するために、**<dependencies>** と **<exclusions>** を追加しました。このファイルには、以下のデータが含まれます。

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-structure:1.0">
  <deployment>
    <dependencies>
      <module name="javax.faces.api" slot="1.2" export="true"/>
      <module name="com.sun.jsf-impl" slot="1.2" export="true"/>
      <module name="org.apache.commons.logging" export="true"/>
      <module name="org.dom4j" export="true"/>
      <module name="org.apache.commons.collections"
export="true"/>
    </dependencies>
  </deployment>
  <sub-deployment name="jboss-seam-booking.war">
    <exclusions>
      <module name="javax.faces.api" slot="main"/>
      <module name="com.sun.jsf-impl" slot="main"/>
    </exclusions>
    <dependencies>
      <module name="javax.faces.api" slot="1.2"/>
      <module name="com.sun.jsf-impl" slot="1.2"/>
    </dependencies>
  </sub-deployment>
</jboss-deployment-structure>
```

2. **ClassNotFoundException** を解決するため、次の JAR を **EAP5_HOME/jboss-eap-5.X/seam/lib/** ディレクトリー (5.X は移行元の EAP 5 バージョンに置き換え) から **jboss-seam-booking.ear/lib/** ディレクトリーにコピーしました。

- hibernate-core.jar
 - hibernate-validator.jar
3. 次のように **jboss-seam-booking.jar/META-INF/persistence.xml** ファイルを変更しました。
1. JBoss EAP 6 に同梱されるサンプルデータベースを使用するよう **jta-data-source** 要素を変更しました。

```
<!-- <jta-data-source>java:/bookingDatasource</jta-data-source> -->
<jta-data-source>java:jboss/datasources/ExampleDS</jta-data-source>
```

2. `hibernate.cache.provider_class` プロパティをコメントアウトしました。

```
<!-- <property name="hibernate.cache.provider_class"
value="org.hibernate.cache.HashtableCacheProvider"/> -->
```

4. 新しい JNDI バインディングを使用するよう、WAR の **lib/components.xml** ファイルを変更しました。
1. 以下のように既存の **core:init** 要素を置き換えました。

```
<!-- <core:init jndi-pattern="jboss-seam-booking/#
{ejbName}/local" debug="true" distributable="false"/> -->
<core:init jndi-pattern="java:app/jboss-seam-booking.jar/#
{ejbName}" debug="true" distributable="false"/>
```

2. "EjbSynchronizations" バインディングおよび "TimerServiceDispatcher" JNDI バインディングのコンポーネント要素を追加しました。

```
<component class="org.jboss.seam.transaction.EjbSynchronizations"
jndi-name="java:app/jboss-seam/EjbSynchronizations"/>
<component class="org.jboss.seam.async.TimerServiceDispatcher"
jndi-name="java:app/jboss-seam/TimerServiceDispatcher"/>
```

[バグを報告する](#)

付録A 改訂履歴

改訂 6.3.0-24

Wednesday July 30 2014

Sande Gilda

Red Hat JBoss Enterprise Application Platform 6.3.0.GA