Red Hat JBoss Enterprise Application Platform 6.4

移行ガイド

Red Hat JBoss Enterprise Application Platform 6 向け

Last Updated: 2022-07-19

Red Hat JBoss Enterprise Application Platform 6.4 移行ガイド

Red Hat JBoss Enterprise Application Platform 6 向け

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

法律上の通知

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

1.1. RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM 6

Red Hat JBoss Enterprise Application Platform 6 (JBoss EAP 6) は、オープン標準に構築されたミドルウェアプラットフォームで、Java Enterprise Edition 6 仕様に準拠します。これは、JBoss Application Server 7 を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、およびその他のテクノロジーと統合します。

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必要になり、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。

バグの報告

1.2. 移行ガイド

JBoss EAP 6 は、Java Enterprise Edition 6 仕様の高速で強力な実装です。アーキテクチャーは Modular Service Container で構築され、アプリケーションが必要とする場合にオンデマンドでサービスを有効にします。この新しいアーキテクチャーにより、JBoss EAP 5 で実行されるアプリケーションは JBoss EAP 6 で実行するために変更が必要になる場合があります。

本ガイドの目的は、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文書化することです。デプロイメントおよびランタイムの問題を解決する方法や、アプリケーションの動作の変更を防ぐ方法に関する情報を提供します。これは、新しいプラットフォームに移行する最初のステップです。アプリケーションが正常にデプロイされ、実行されたら、各コンポーネントをアップグレードして JBoss EAP 6 の新機能を使用する計画を立てることができます。

バグの報告

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は以前のバージョンとは異なる構造のため、アプリケーションの移行を試みる前に調査および計画を行うことが推奨されます。

1. **JBoss EAP 6 の新機能と変更内容**
   本リリースには JBoss EAP 5 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複数の変更が含まれています。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、および JNDI ルックアップ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. **スタートガイドのドキュメント**
   必ずJBoss EAP 6 の『Development Guide』 の『Get Started Developing Applications』の章を確認してください (https://access.redhat.com/documentation/ja-jp/red_hat_jboss_enterprise_application_platform/?version=6.4)。これには、以下に関する重要な情報が含まれています。

   - Java EE 6
   - 新しいモジュラー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
   - 開発環境用に Maven を設定する方法
   - 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**
   必ずJBoss EAP 6 の『Development Guide』 の『Maven Guide』の章を確認してください (https://access.redhat.com/documentation/ja-jp/red_hat_jboss_enterprise_application_platform/?version=6.4)。プロジェクト 『Manage Project Dependencies』 セクションには、JBoss EAP の BOM (Bill of Material)アーティファクトを使用するようにプロジェクトを設定する方法に関する重要な情報が記載されています。

4. **アプリケーションの分析と理解**
   アプリケーションはそれぞれ異なるため、移行を始める前に既存のアプリケーションのコンポーネントおよびアーキテクチャーを十分に理解する必要があります。

重要

必ずバックアップコピーを作成してからアプリケーションに変更を加えるようにしてください。

バグの報告

2.2. JBOSS EAP 6 の新機能と変更内容

はじめに

以下は、以前のリリースから JBoss EAP 6 への主な変更点です。

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アーキテクチャーは階層的でした。JBoss EAP 6 では、クラスローディングは JBoss Modules をベースにしています。これにより、完全にアプリケーションが分離され、サーバー実装クラスが隠され、アプリケーションが必要とするクラスのみが読み込まれます。クラスローディングにより、同時にパフォーマンスが向上します。JBoss EAP 5 用に作成されたアプリケーションでは、モジュールの依存関係を指定し、場合によってはアーカイブを再パッケージ化するように変更する必要があります。詳細については、JBoss EAP 6 『Development Guide』の 『Class Loading and Modules』を参照してください (https://access.redhat.com/documentation/ja-jp/red_hat_jboss_enterprise_application_platform/?version=6.4)。

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または管理対象ドメインで実行できます。管理対象ドメインでは、サーバーのグループ全体を一度に設定し、サーバーのネットワーク全体で設定を同期できます。これは以前のリリース用にビルドされたアプリケーションに影響を与えることはありませんが、これにより複数のサーバーへのデプロイメントの管理が簡素化されます。詳細については、JBoss EAP 6 の 『Administration and Configuration Guide』の『About Managed Domains』を参照してください(https://access.redhat.com/documentation/ja-jp/red_hat_jboss_enterprise_application_platform/?version=6.4)。

デプロイメント設定

スタンドアロンサーバーおよび管理対象ドメイン

JBoss EAP 5 はプロファイルベースのデプロイメント設定を使用していました。これらのプロファイルは ***EAP_HOME*/server/** ディレクトリーににありました。アプリケーションには、多くの場合、セキュリティー、データベース、リソースアダプター、およびその他の設定用に複数の設定ファイルが含まれていました。JBoss EAP 6 では、デプロイメント設定は 1 つのファイルを使用して行われます。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および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稼働しているサーバーの場合は、***EAP_HOME*/domain/configuration/domain.xml** ファイルを使用して設定されます。複数の JBoss EAP 5 設定ファイルに含まれる情報を新しい単一の設定ファイルに移行する必要があります。

デプロイメントの順序

JBoss EAP 6 は、デプロイメントに高速な同時初期化を使用するため、パフォーマンスと効率性が向上します。ほとんどの場合、アプリケーションサーバーは、依存関係を事前に自動的に判断し、最も効率的なデプロイメント方式を選択できます。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る JBoss EAP 5 アプリケーション、または CDI 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 JNDI ルックアップを使用する JBoss EAP 5 アプリケーションには、設定の変更が必要になる場合があります。

ディレクトリー構造およびスクリプト

前述のように、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ないため、***EAP_HOME*/server/** ディレクトリーはありません。スタンドアロンサーバーの設定ファイルは ***EAP_HOME*/standalone/configuration/** ディレクトリーに配置され、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稼働しているサーバーの場合は、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** がサーバーの起動に使用されました。JBoss EAP 6 では、サーバー起動スクリプトはサーバーの実行方法によって異なります。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** を使用してスタンドアロンサーバーを起動します。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** は、管理対象ドメインの起動に使用されます。

JNDI ルックアップ

JBoss EAP 6 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。JNDI ルックアップを使用する JBoss EAP 5 用に書かれたアプリケーションは、新しい標準化された JNDI 名前空間規則に従うように変更する必要があります。JNDI 命名構文の詳細は、「移植可能な EJB JNDI 名」 を参照してください。

仮想ファイルシステム

JBoss EAP 6 では、VFS2 が VFS3 に置き換えられました。VFS2 を使用した JBoss EAP 5 で利用できた設定オプションの一部が JBoss EAP 6 では必要なくなりました。VFS2 で利用できたSystemプロパティー設定は、VFS3 では使用されず、利用できなくなりました。キャッシュシステムが VFS3 で置き換えられ、VFS2 に存在したキャッシュ問題を解決しました。VFS は JBoss EAP によって内部的に使用され、アプリケーションコード内で直接アクセスする必要はありません。

詳細は、JBoss EAP 6 の 『Development Guide』 の『New and Changed Features in JBoss EAP 6 』を参照してください (https://access.redhat.com/documentation/ja-jp/red_hat_jboss_enterprise_application_platform/?version=6.4)。

バグの報告

2.3. 非推奨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に移行する前に、以前のリリースの JBoss EAP で利用できた機能の一部が非推奨またはサポート対象外となった可能性があることに注意してください。包括的なリストは、カスタマーポータルの JBoss EAP 6 の 『Release Notes』 の 『Unsupported Features』 を参照してください (https://access.redhat.com/documentation/ja-jp/red_hat_jboss_enterprise_application_platform/?version=6.4)。

サポート対象外となった機能の一部を以下に示します。

サーバー起動の -b コマンドライン引数

以前のバージョンでは、JBoss EAP は IP アドレスに関係なく、**-b** 起動パラメーターで指定されたアドレスを自動的に使用していました。JBoss EAP 6 では、サーバー **<inet-address>** 設定は、一致する IP アドレスで設定されたネットワークインターフェースを検索します。IPアドレス **127.0.0.1** は機能しますが、**127.*.*.*** は機能しなくなりました。**127.*.*.*** IP アドレスにバインドする **-b** コマンドライン引数を使用して JBoss EAP 6 サーバーを起動した場合は、まずサーバー設定ファイルでインターフェースを **<inet-address>** から **<loopback-address>** に変更する必要があります。

管理 CLI を使用してサーバーを設定する方法は、カスタマーポータルの JBoss Enterprise Application Platform の 『Administration and Configuration Guide』 の 『Management CLI Operations』 を参照してください (https://access.redhat.com/documentation/ja-jp/red_hat_jboss_enterprise_application_platform/?version=6.4)。

EJB 依存関係

これまでのバージョンの JBoss EAP では、他の EJB を含むサービスの EJB 依存関係は、**jboss.xml** デプロイメント記述子の **<depends>** タグを使用して指定できました。以下に例を示します。

```
<depends>jboss.j2ee:jndiName=com/myorg/app/Foo,service=EJB</depends>
<depends>jboss.mq.destination:service=Queue,name=queue/HelloworldQueue</depends>
```

JBoss EAP 6 では、**@EJB** アノテーションを使用して EJB 参照を注入し、**@Resource** アノテーションを使用してデータソースやその他のリソースにアクセスする必要があります。以下に例を示します。

```
@EJB(lookup="java:global/MyApp/FooImpl!com.myorg.app.Foo")
@Resource(mappedName = "java:/queue/HelloworldQueue")
```

JNDI ルックアップも変更になりました。詳細は、本ガイドの 『JNDI の変更』 セクションを参照してください。

EJB 参照の詳細は、カスタマーポータルの JBoss Enterprise Application Platform の 『Development Guide』 の 『EJB Reference Resolution』 セクションを参照してください (https://access.redhat.com/documentation/ja-jp/red_hat_jboss_enterprise_application_platform/?version=6.4)。

HTTPInvoker

これまでのバージョンの JBoss EAP では、HTTPInvoker を使用して EJB、JNDI、または JMS を設定して HTTP プロトコルを使用することができました。これは JBoss EAP 6 では不可能になりました。

HA Singleton デプロイメントおよび BarrierController サービス

HA SingletonService は、クラスター内で実行されるサービスのインスタンスが 1 つだけであることを保証します。

JBoss EAP 5 は、HASingletonDeployer サービス、HASingletonController を使用した POJO デプロイメント、BarrierController サービスを使用した HASingleton デプロイメントなど、HA Singleton デプロイメントの複数の方式をサポートしていました。これらの方式は、クラスター内の異なるノードが起動および停止した際の通知を提供するのにすべて HAPartition に依存していたので、利用できなくなりました。

JBoss EAP 6 では、HA Singleton デプロイメントが完全に変更になりました。シングルトンデプロイヤーは Modular Service Container (MSC)サービスでのみ動作するようになりました。SingletonService を使用すると、ターゲットサービスはクラスター内のすべてのノードにインストールされますが、常に1つのノード上でのみ起動されます。この方法では、デプロイメント要件を単純化し、シングルトンマスターサービスをノード間で再配置するために必要な時間を最小限にします。ただし、同じ機能を実現するには、カスタムコードを作成する必要があります。HA Singleton デプロイメントの例は、製品に同梱される JBoss EAP クイックスタートサンプルアプリケーションに含まれています。HA Singleton に関する詳細は、「HA シングルトンの実装」 を参照してください。

JAAS Security Manager サービス

JBoss EAP 5 には JAAS Security Manager が同梱されており、データソースパスワード暗号化サービスを提供し、パスワードベースの暗号化(PBE)でアイデンティティーを設定していました。このサービスは JBoss EAP 6 には含まれていません。JBoss EAP 6 では、データソースパスワードの暗号化にパスワード vault を使用することを推奨しています。JAAS Security Manager サービスと JBoss EAP 5 を使用したパスワード暗号化アルゴリズムは、JBoss EAP 6 でパスワード vault を使用

してパスワードを暗号化するときに使用されるアルゴリズムと互換性がありません。管理者は、以前のリリースでパスワードベースの暗号化を使用して暗号化されたパスワードを再生成する必要があります。

以前のバージョンの JBoss EAP でクレデンシャルを動的にデプロイするのに DynamicLoginConfig サービスを使用した場合、JBoss EAP 6 ではクレデンシャルを動的にデプロイするための同様の方法はありません。広く知られた既知のマスク値以外の方法で vault パスワードをセキュアにする機能は、JBoss EAP 6.4 以降でのみ利用できます。

バグの報告

2.4. 移行およびアップグレード

メジャーアップグレード

JBoss EAP 5 から JBoss EAP 6 など、アプリケーションを他のメジャーリリースに移動する場合にメジャーアップグレードまたは移行が必要になります。これは、本ガイドで取り上げている移行のタイプです。アプリケーションが Java EE 仕様に準拠し、非推奨の API にアクセスせず、プロプライエタリーコードを含まない場合、変更なしにアプリケーションを JBoss EAP 6 で実行できる可能性があります。それ以外の場合は、アプリケーションコードの変更が必要になることがあります。JBoss EAP 6 でサーバー設定が変更になり、すべてのサーバー設定で移行が必要です。

マイナーアップデート

JBoss EAP では、定期的にポイントリリースが提供されます。JBoss EAP 6.3 から JBoss EAP 6.4 など、ある JBoss EAP ポイントリリースから別のポイントリリースに移行する予定の場合は、カスタマーポータルのJBoss EAP 6の『Installation Guide』 の『Patching JBoss EAP 6』の章を参照してください(https://access.redhat.com/documentation/ja-jp/red_hat_jboss_enterprise_application_platform/?version=6.4)。

累積パッチ

JBoss EAP では、バグおよびセキュリティーの修正が含まれる累積パッチも定期的に提供されます。累積パッチのリリースごとに、リリース番号の最後の数字が 1 ずつ増えます (例: 6.4.1 から 6.4.2)。パッチインストールの詳細は、カスタマーポータルの JBoss EAP 6 の『Installation Guide』 の『Patching JBoss EAP 6』を参照してください (https://access.redhat.com/documentation/ja-jp/red_hat_jboss_enterprise_application_platform/?version=6.4)。

バグの報告

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 でのクラスローディングと設定の変更は、ほぼすべてのアプリケーションに影響します。JBoss EAP 6 は、新しい標準の移植可能な JNDI 命名構文も使用します。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に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 JNDI 名前空間名の更新」

バグの報告

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がJBoss EAP 6 で大きく変更になり、ほとんどすべてのアプリケーションに影響します。アプリケーションを移行する際には、最初に以下の情報を確認してください。

1. まず、アプリケーションとその依存関係のパッケージ化を確認します。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する場合は、正しいモジュール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変更」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しなければならない場合があります。詳細は、 「EAR と WAR のパッケージ化の変更」を参照してください。

バグの報告

3.1.2.2. モジュール依存関係について

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がある任意のモジュールのクラスにのみアクセスできます。

暗黙的な依存関係

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ているモジュールの依存関係を暗黙的に自動的に追加します。これにより、クラスが実行時にデプロイメントからアクセス可能になり、依存関係を明示的に追加するタスクから開発者を解放します。これらの暗黙的な依存関係が追加される方法とタイミングに関する詳細は、JBoss EAP 6 の 『Development Guide』 の『Class Loading and Modules』の章の『Implicit Module Dependencies』を参照してください (https://access.redhat.com/documentation/ja-jp/red_hat_jboss_enterprise_application_platform/?version=6.4)。

明示的な依存関係

他のクラスでは、モジュールを明示的に指定しないと、依存関係がないため、デプロイメントまたはランタイムエラーが発生します。依存関係がない場合、サーバーログに **ClassNotFoundExceptions** 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 を読み込むか、またはモジュールが別のモジュールによって読み込まれたクラスを拡張するクラスを読み込む場合、サーバーログに **ClassCastExceptions** 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。モジュールの依存関係の詳細は、JBoss EAP 6 の『Development Guide』 の『Class Loading and Module』 の章の『Overview of Class Loading and Modules』を参照してください(https://access.redhat.com/documentation/ja-jp/red_hat_jboss_enterprise_application_platform/?version=6.4)。

バグの報告

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 でのクラスローディングは、これまでのバージョンの JBoss EAP とは大きく異なります。クラスローディングは JBoss Modules プロジェクトをベースにするようになりました。すべての JAR をフラットクラスパスに読み込む単一の階層構造クラスローダーではなく、各ライブラリーは、依存するモジュールにのみリンクするモジュールになります。JBoss EAP 6 のデプロイメントもモジュールであり、これらのクラスへの明示的な依存関係が定義されない限り、アプリケーションサーバーの JAR で定義されたクラスに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が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は、Java EE API の依存関係が自動的または暗黙的に追加されます。サーバーによって自動的に追加される依存関係の完全なリストについては、JBoss EAP 6 の 『Development Guide』 の『Class Loading and Modules』の章の『Implicit Module Dependencies』を参照してください (https://access.redhat.com/documentation/ja-jp/red_hat_jboss_enterprise_application_platform/?version=6.4)。

タスク

アプリケーションを JBoss EAP 6 に移行する場合は、モジュラークラスローディングの変更により、以下のタスクの 1 つまたは複数を実行しなければならない場合があります。

- 「モジュール依存関係について」
- 「Tattletale を使用したアプリケーション依存関係の検索」
- 「JBoss EAP 6 で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ティングシステムのパッケージリソース」

バグの報告

3.1.3. 設定ファイルの変更

3.1.3.1. JBoss EAP 6 で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

モジュラークラスローディングを使用する JBoss EAP 6 の変更に伴い、1 つまたは複数のファイルを作成または編集して依存関係を追加したり、自動的な依存関係がロードされないようにする必要がある場合があります。クラスローディングとクラスローディングの優先順位に関する詳細は、JBoss EAP 6 の

『Development Guide』 の『Class Loading and Modules』の章を参照してください (https://access.redhat.com/documentation/ja-jp/red_hat_jboss_enterprise_application_platform/?version=6.4)。

以下のファイルは、JBoss EAP 6 でのクラスローディングを制御するために使用されます。

jboss-web.xml

jboss-web.xml ファイルに **<class-loading>** 要素を定義した場合は、これを削除する必要があります。JBoss EAP 5 でこれが呼び出した動作が、JBoss EAP 6 ではデフォルトのクラスローディング動作であるため、これは不要になりました。この要素を削除しないと、サーバーログに ParseError および XMLStreamException が記録されます。

これは、コメントアウトされる **jboss-web.xml** ファイルの **<class-loading>** 要素の例です。

```
<!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-config>java2ParentDelegation=false</loader-repository-config>
      </loader-repository>
   </class-loading>
 -->
 </jboss-web>
```

MANIFEST.MF

手動での編集

アプリケーションが使用するコンポーネントまたはモジュールによっては、このファイルに依存関係を 1 つまたは複数追加する必要がある場合があります。 **Dependencies** または **Class-Path** エントリーとして追加できます。

以下は、開発者により編集される **MANIFEST.MF** の例です。

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

このファイルを変更する場合は、ファイルの最後に改行文字を含めるようにしてください。

Maven を使用した生成

Maven を使用する場合は、**pom.xml** ファイルを変更して **MANIFEST.MF** ファイルの依存関係を生成する必要があります。アプリケーションが EJB 3.0 を使用する場合は、**pom.xml** ファイルに以下のようなセクションがある可能性があります。

```
<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** を使用する場合は、**MANIFEST.MF** ファイルにその依存関係が必要になります。その依存関係を生成するには、以下のように **<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、**src/main/resources/META-INF/MANIFEST.MF** ファイルには依存関係エントリーのみを含める必要があります。

```
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、詳細な方法でクラスローディングを制御するために使用できる 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できます。また、自動依存関係が追加されなくなり、追加のモジュールを定義し、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、さらにリソースルートをモジュールに追加することもできます。

以下は、JSF 1.2 モジュールの依存関係を追加し、JSF 2.0 モジュールの自動読み込みを阻止する **jboss-deployment-structure.xml** ファイルの例になります。

```
<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"/>
  </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>
```

このファイルの詳細は、 「jboss-deployment-structure.xml」 を参照してください。

application.xml

これまでのバージョンの JBoss EAP では、**jboss-app.xml** ファイルを使用して EAR 内のデプロイメントの順序を制御していました。これは当てはまらなくなりました。Java EE6 仕様は、**application.xml** の **<initialize-in-order>** 要素を提供し、EAR 内の Java EE モジュールがデプロイされる順序を制御することができます。

多くの場合、デプロイメント順序を指定する必要はありません。アプリケーションが依存関係注入とリソース参照を使用して外部モジュールのコンポーネントを参照する場合、アプリケーションサーバーは正しく最適なコンポーネントの順序決定方法を暗黙的に決定できるため、多くの場合**<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** と **myApp.war** が含まれるアプリケーションがあるとします。**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 を注入します。この場合、アプリケーションサーバーは、サーブレットが起動される前に EJB コンポーネントが利用可能であることを把握し、**<initialize-in-order>** 要素を使用する必要はありません。

ただし、そのサーブレットが以下のようなレガシー JNDI ルックアップスタイルのリモート参照を使用して Bean にアクセスする場合は、モジュールの順序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、サーバーは EJB コンポーネントが **myBeans.jar** にあることを判断できず、**myBeans.jar** のコンポーネントが **myApp.war** のコンポーネントよりも先に初期化および起動されるように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、**application.xml** ファイルで **myBeans.jar** モジュールおよび **myApp.war** モジュールの順序を指定します。

以下は、**<initialize-in-order>** 要素を使用してデプロイメントの順序を制御する例です。**myBeans.jar** は **myApp.war** ファイルよりも先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については、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、デプロイメントが遅くなることに注意してください。デプロイメントの最適化に関してコンテナーの柔軟性を高めることができるため、依存関係の注入やリソース参照を使用して適切な依存関係を定義することが推奨されます。

jboss-ejb3.xml

jboss-ejb3.xml デプロイメント記述子は **jboss.xml** デプロイメント記述子に置き換わり、Java Enterprise Edition (EE)で定義された **ejb-jar.xml** デプロイメント記述子によって提供された機能をオーバーライドおよび追加します。新しいファイルは **jboss.xml** と互換性がなく、デプロイメントで **jboss.xml** は無視されるようになりました。

login-config.xml

login-config.xml ファイルはセキュリティー設定に使用されなくなりました。セキュリティー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の場合、これは **standalone/configuration/standalone.xml** ファイルになります。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** ファイルになります。

バグの報告

3.1.3.2. jboss-deployment-structure.xml

jboss-deployment-structure.xml は JBoss EAP 6 の新しい任意の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、デプロイメントで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。

バグの報告

3.1.3.3. 新しいモジュラークラスローテ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のみが **WEB-INF/classes** および **WEB-INF/lib** ディレクトリーから読み込まれます。指定された場所にアプリケーションのアーティファクトをパッケージ化できないと、**ClassNotFoundException**、**NoClassDefError**、またはその他のランタイムエラーが発生する可能性があります。

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変更するか、カスタムモジュールを定義する必要があります。

リソースのパッケージの変更

アプリケーションだけがリソースを利用できるようにするには、プロパティーファイル、JAR、または他のアーティファクトを WEB-INF/classes/ または WEB-INF/lib/ ディレクトリーに移動し、それらをWAR でバンドル化する必要があります。このアプローチの詳細は、 「ResourceBundle プロパティーの場所の変更」 を参照してください。

カスタムモジュールの作成

JBoss EAP 6 サーバーで実行しているすべてのアプリケーションで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。このアプローチの詳細は、 「カスタムモジュールの作成」 を参照してください。

バグの報告

3.1.3.4. ResourceBundle プロパティーの場所の変更

概要

これまでのバージョンの JBoss EAP では、 ***EAP_HOME***/server/***SERVER_NAME***/conf/ ディレクトリーがクラスパスにあり、アプリケーションで利用できていました。JBoss EAP 6 のアプリケーションのクラスパスでプロパティーを使用できるようにするには、アプリケーション内にプロパティーをパッケージ化する必要があります。

手順3.1 ResourceBundle プロパティーの場所の変更

1. WAR アーカイブをデプロイする場合は、これらのプロパティーを WAR の WEB-INF/classes/ フォルダーに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で利用できるようにしたい場合は、JAR のルートにパッケージ化してから JAR を EAR の lib/ フォルダーに配置する必要があります。

バグの報告

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3.2 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、反映させます。
   a. ***EAP_HOME***/module ディレクトリーにディレクトリー構造を作成し、ファイルと JAR が含まれるようにします。以下に例を示します。

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

   b. プロパティーファイルを、前のステップで作成した ***EAP_HOME***/modules/myorg-conf/main/properties/ ディレクトリーに移動します。

c. ***EAP_HOME*/modules/myorg-conf/main/** ディレクトリーに、以下の XML が含まれる **module.xml** ファイルを作成します。

```
<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** サブシステムを変更します。管理 CLI を使用するか、ファイルを手動で編集できます。

- 管理 CLI を使用してサーバー設定ファイルを変更するには、以下の手順に従います。

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** サブシステムで **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"}])
```

以下の結果が表示されるはずです。

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

- 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、サーバー設定ファイルをテキストエディターで開きます。スタンドアロンサーバーを実行している場合は、これは ***EAP_HOME*/standalone/configuration/standalone.xml** ファイルで、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変更された **ee** サブシステム要素の例になります。

例3.1 **myorg-conf** 要素

```
<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ーファイルをロードできるようになります。

例3.2 プロパティーファイルの読み込み

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

バグの報告

3.1.4. ロギングの変更

3.1.4.1. ロギング依存関係の変更

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を維持したり、新しい JBoss ロギングインフラストラクチャーに移行したりできます。意思決定に関係なく、モジュラークラスローディングの変更により、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ります。

手順3.3 アプリケーションの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用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するためのコード変更」

バグの報告

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

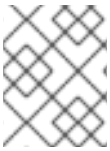
概要

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がデフォルトで追加されます。ほとんどの場合、JBoss EAP コンテナー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。ただし、サードパーティーフレームワークが提供する特定の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。デプロイメントでサードパーティーのロギングフレームワークが使用されますが、サーバーログは JBoss EAP ロギングサブシステム設定を引き続き使用します。

以下の手順は、デプロイメントから JBoss EAP 6 の **org.apache.log4j** モジュールを除外する方法を示しています。最初の手順は、JBoss EAP 6 の任意のリリースで機能します。2 つ目の手順は、JBoss EAP 6.3 以降にのみ適用されます。

手順3.4 log4j.properties または log4j.xml ファイルを使用する JBoss EAP 6 を設定

この手順は、すべてのバージョンの JBoss EAP 6 で機能します。



注記

このメソッドは log4j 設定ファイルを使用するため、実行時に log4j ロギング設定を変更できる必要はなくなりました。

1. **以下の内容で 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. **WAR をデプロイする場合は META-INF/ ディレクトリーまたは WEB-INF/ ディレクトリーに、EAR をデプロイする場合は META-INF/ ディレクトリーに、それぞれ jboss-deployment-structure.xml ファイルを配置します。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、それぞれのサブデプロイメントについてもモジュールを除外する必要があります。**

3. **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、それぞれ log4j.properties または log4j.xml ファイルを追加します。WAR の lib/ ディレクトリーにファイルを配置する場合は、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. **以下のランタイム引数を使用して JBoss EAP 6 サーバーを起動します。これにより、アプリケーションのデプロイメント時に ClassCastException がコンソールに表示されなくなります。**

```
-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3.5 JBoss EAP 6.3 以降のロギング依存関係の設定

JBoss EAP 6.3 以降では、新しい add-logging-api-dependencies ロギングシステムプロパティーを使用して、サードパーティーの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. 以下のランタイム引数を使用して JBoss EAP 6 サーバーを起動します。これにより、アプリケーションのデプロイメント時に ClassCastException がコンソールに表示されなくなります。

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

2. ターミナルを開き、管理 CLI に接続します。
   - 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. logging サブシステムの add-logging-api-dependencies 属性を変更します。

   この属性は、コンテナーが暗黙的なロギング API 依存関係をデプロイメントに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 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に追加されません。

   サードパーティーの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。

バグの報告

3.1.4.3. 新しい JBoss ロギングフレームワークを使用するためのコード変更

概要

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

手順3.6 JBoss ロギングフレームワークを使用するためのコードと依存関係の変更

1. インポートとロギングコードを変更します。

   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を以下に示します。

```
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. ロギング依存関係を追加します。

   JBoss Logging 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 ファイルは以下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を探す方法は、「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」 および 「移行の問題のデバッグおよび解決」 を参照してください。

バグの報告

3.1.5. アプリケーションのパッケージ化の変更

3.1.5.1. EAR と WAR のパッケージ化の変更

概要

アプリケーションを移行する場合、モジュラークラスローディングへの変更が原因で EAR または WAR のパッケージ構造を変更しなければならない場合があります。モジュール依存関係は、以下の特定の順序で読み込まれます。

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

手順3.7 アーカイブパッケージの変更

1. WAR をパッケージ化します。

   WAR は 1 つのモジュールで、WAR のすべてのクラスは、同じクラスローダーで読み込まれます。つまり、WEB-INF/lib ディレクトリーにパッケージ化されたクラスは WEB-INF/classes ディレクトリーにあるクラスと同じように処理されます。

2. EAR をパッケージ化します。

   EAR は複数のモジュールで構成されます。EAR/lib/ ディレクトリーは単一のモジュールで、EAR 内のすべての WAR または EJB jar サブデプロイメントは個別のモジュールです。明示的な依存関係が定義されない限り、クラスは EAR 内の他のモジュールのクラスにアクセスできません。サブデプロイメントは、EAR/lib/ ディレクトリーのクラスへのアクセスを提供する親モ

ジュールの自動的な依存関係を常に持ちます。ただし、サブデプロイメントには相互のアクセスを許可する自動依存関係が常にあるわけではありません。この動作は、以下のように ee サブシステム設定の <ear-subdeployments-isolated> 要素を設定することで制御されます。

```
<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、サブデプロイメントは EAR 内の他のサブデプロイメントに属するクラスを参照できます。

クラスローディングに関する詳細は、JBoss EAP 6 の 『Development Guide』 の『Class Loading and Modules』の章を参照してください (https://access.redhat.com/documentation/ja-jp/red_hat_jboss_enterprise_application_platform/?version=6.4)。

バグの報告

3.1.5.2. ルートコンテキストの優先順位の変更

JBoss EAP 5 では、jboss-web.xml WAR ファイルで定義された <context-root> 要素が application.xml EAR ファイルで定義された <context-root> 要素よりも優先されます。

JBoss EAP 6 では、逆です。application.xml EAR ファイルで定義された <context-root> 要素は、jboss-web.xml WAR ファイルで定義された <context-root> 要素よりも優先されます。WAR ファイルが EAR アーカイブ内にデプロイされている場合、application.xml ファイルに <context-root> 要素を定義します。

バグの報告

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、サービスおよびサブシステムは多くの異なるファイルで設定されていました。JBoss EAP 6 では、主に 1 つのファイルで設定が実行されるようになりました。アプリケーションが以下のリソースまたはサービスのいずれかを使用する場合、設定の変更が必要になる場合があります。

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 をバンドルしている場合は、移行オプションについて 「Hibernate または JPA のデータソースの設定」 を参照してください。
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. 基本的なセキュリティーの変更の設定方法に関する情報は、 「アプリケーションセキュリティーの変更の設定」 を参照してください。

バグの報告

3.1.6.2. DataSource設定の更新

概要

これまでのバージョンの JBoss EAP では、JCA DataSource 設定は ***-ds.xml** のサフィックスのファイルで定義されていました。その後、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、アプリケーションとともにパッケージ化されます。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す。DataSource のこの設定方法は開発用に引き続きサポートされますが、JBoss の管理ツールではサポートされないため、本番環境では推奨されません。

JBoss EAP 6 では、DataSource はサーバー設定ファイルで設定されます。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、DataSource は **domain/configuration/domain.xml** ファイルで設定されます。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、DataSource は **standalone/configuration/standalone.xml** ファイルで設定されます。このように設定されたDataSource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)などの JBoss 管理インターフェイスを使用して管理および制御できます。これらのツールにより、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が容易になります。

次のセクションでは、利用可能な管理ツールで管理しサポートできるように、DataSource 設定を変更する方法を説明します。

JBoss EAP 6 の管理可能な DataSource 設定への移行

JDBC 4.0 準拠のドライバーは、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できます。JDBC 4.0 に準拠するドライバーには、ドライバークラス名を指定する **META-INF/services/java.sql.Driver** ファイルが含まれます。JDBC 4.0 に準拠していないドライバーには、追加の手順が必要です。ドライバーを JDBC 4.0 準拠にする方法や、現在の DataSource 設定を Web 管理コンソールおよび CLI で管理できる設定に更新する方法は、「JDBC ドライバーのインストールおよび設定」 を参照してください。

アプリケーションが Hibernate または JPA を使用する場合は、追加の変更が必要になる場合があります。詳細は、「Hibernate または JPA 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 ツールを使用して DataSource および ResourceAdapter 設定を移行できます。このツールは、***-ds.xml** スタイルの設定ファイルを JBoss EAP 6 で想定される形式に変換します。詳細は、「IronJacamar Tool を使用したデータソースおよびリソースアダプター設定の移行」 を参照してください。

リモートDataSourceルックアップを実行するコードの移行

以前のバージョンの JBoss EAP では、DataSource オブジェクトの JNDI リモートルックアップを実行できますが、以下の理由により推奨されません。

- サーバーリソースのクライアント制御は信頼性がなく、クライアントがクラッシュしたり、サーバーへの接続を失うと接続が漏洩する可能性があります。
- すべてのデータベース操作が **MBean** を介してプロキシーされるため、パフォーマンスは非常に遅くなります。
- トランザクションの伝播はサポートされていません。

この機能は JBoss EAP 6 から削除されたため、アプリケーションの移行時に **NotSerializableException** が表示されることがあります。推奨される方法は、EJB を作成して DataSource にアクセスし、EJB をリモートで呼び出すことです。詳細は、本ガイドの「『リモート呼び出しの変更』」を参照してください。詳細は、JBoss EAP 6の 『Development Guide』 を参照してください (https://access.redhat.com/documentation/ja-jp/red_hat_jboss_enterprise_application_platform/?version=6.4)。

バグの報告

3.1.6.3. JDBC ドライバーのインストールおよび設定

概要

JDBC ドライバーは、以下の 2 つの方法のいずれかでコンテナーにインストールできます。

- デプロイメントとして
- コアモジュールとして

各アプローチの長所とデメリットを以下に示します。

JBoss EAP 6 では、データソースはサーバー設定ファイルで設定されます。JBoss EAP 6 インスタンスが管理対象ドメインで稼働している場合、データソースは **domain/configuration/domain.xml** ファイルで設定されます。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** ファイルで設定されます。両方のモードで同じであるスキーマ参照情報は、JBoss EAP 6 のインストールの **docs/schema/** ディレクトリーにあります。ここでは、サーバーがスタンドアロンサーバーとして実行され、データソースが **standalone.xml** ファイルで設定されることを前提とします。

手順3.8 JDBC ドライバーのインストールおよび設定

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。

      これが、ドライバーをインストールするの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、通常の 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、JDBC 4.0 準拠の JAR を ***EAP_HOME*/standalone/deployments/** ディレクトリーにコピーします。管理対象ドメインでは、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。

      以下は、デプロイメントとしてスタンドアロンサーバーにインストールされた MySQL JDBC ドライバーの例です。

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

      JDBC 4.0 準拠のドライバーは自動的に認識され、名前とバージョンでシステムにインストールされます。JDBC 4.0 準拠の JAR には、ドライバー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ます。ドライバーが JDBC 4.0 に準拠していない場合には、以下のいずれかの方法で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/** パスの下にある JAR に追加します。このファイルにはドライバークラス名が含まれている必要があります。以下に例を示します。

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

      - デプロイメントディレクトリーに **java.sql.Driver** ファイルを作成します。スタンドアロンサーバーとして実行されている JBoss EAP 6 インスタンスの場合は、ファイルを ***EAP_HOME*/standalone/deployments/META-INF/services/java.sql.Driver** に配置する必要があります。サーバーが管理対象ドメインにある場合は、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。

このアプローチの利点は以下のとおりです。

- 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。
- 管理対象ドメインでサーバーを実行している場合は、このアプローチを使用するデプロイメントは、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。これは、管理者がドライバー JAR を手動で配布する必要がないことを意味します。

このアプローチの短所は以下のとおりです。

- JDBC ドライバーが複数の JAR（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 など）で構成される場合、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 ドライバーを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 に準拠していない場合、ドライバークラス名が含まれるファイルを作成し、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、追加の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。

- コアモジュールとしてのMySQL JDBC ドライバーのインストール

  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 ドライバーの以下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。

```
<?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、このモジュールの依存関係を他のモジュールに指定するために使用されます。この場合、すべての JDBC データソースの場合と同様に、**javax.api** という名前の別のモジュールで定義される Java JDBC API に依存します。このモジュールは **modules/system/layers/base/javax/api/main/** ディレクトリーにあります。

注記

module.xml ファイルの先頭にスペースがないことを確認します。そうでないと、このドライバーの「New missing/unsatisfied dependencies」エラーが表示されます。

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

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

- IBM DB2 JDBC ドライバーおよびライセンス 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 に加えて JAR を必要とするドライバーをデプロイする方法を説明する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 ドライバーおよびライセンスの以下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。

```
<?xml version="1.0" encoding="UTF-8"?>
<module xmlns="urn:jboss:module:1.1" name="com.ibm.db2">
  <resources>
    <resource-root path="db2jcc.jar"/>
    <resource-root path="db2jcc_license_cisuz.jar"/>
  </resources>
  <dependencies>
    <module name="javax.api"/>
    <module name="javax.transaction.api"/>
  </dependencies>
</module>
```

注記

module.xml ファイルの先頭にスペースがないことを確認します。そうでないと、このドライバーの「New missing/unsatisfied dependencies」エラーが表示されます。

  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/
```

このアプローチの利点は以下のとおりです。

- これは、JDBC ドライバーが複数の JAR で構成される場合に機能する唯一の方法です。
- この方法では、JDBC 4.0 に準拠していないドライバーを、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。

このアプローチの短所は以下のとおりです。

- モジュールの設定がより困難です。
- モジュールは、管理対象ドメインで実行されているすべての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. データベースドライバーを追加します。

   同じファイルの <drivers> 要素に <driver> 要素を追加します。ここでも、これには *-ds.xml ファイルで以前に定義された同じデータソース情報が含まれます。

   まず、ドライバー JAR が JDBC 4.0 に準拠しているかどうかを判断します。JDBC 4.0 に準拠する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の名前を探します。JDBC 4.0 に準拠するドライバーは、JAR ですでに指定されているため、<driver-class> 要素は必要ありません。以下は、JDBC 4.0 準拠の MySQL ドライバーのドライバー要素の例です。

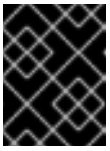
   ```
   <driver name="mysql-connector-java-5.1.15.jar" module="com.mysql"/>
   ```

   JDBC 4.0 に準拠していないドライバーでは、ドライバークラス名を指定する META-INF/services/java.sql.Driver ファイルがないため、ドライバークラスを識別する <driver-class> 属性が必要です。以下は、JDBC 4.0 に準拠していないドライバーのドライバー要素の例です。

   ```
   <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、*-ds.xml ファイルで以前に定義された同じデータソース情報の多くが含まれます。

   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。

   以下は、standalone.xml ファイルの MySQL データソース要素の例になります。

   ```
   <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の 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、アプリケーションの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、 「新しい JNDI 名前空間ルールに従うように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。以下に例を示します。

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

バグの報告

3.1.6.4. Hibernate または JPA のデータソースの設定

アプリケーションが JPA を使用し、現在 Hibernate JAR をバンドルしている場合、JBoss EAP 6 に含まれる Hibernate を使用できます。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 Hibernate バンドルを削除する必要があります。

手順3.9 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から Hibernate JAR を削除します。
2. persistence.xml ファイルの <hibernate.transaction.manager_lookup_class> 要素は必要ないため、この要素を削除またはコメントアウトします。

バグの報告

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の設定は、*-ds.xml のサフィックスのファイルで定義されていました。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している場合、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している場合は、*EAP_HOME*/standalone/configuration/standalone.xml ファイルでリソースアダプターを設定します。両方のモードで同じであるスキーマ参照情報は、IronJacamar Web サイト (http://www.ironjacamar.org/documentation.html)の 『Schemas』 セクションにあります。

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情報は、サーバー設定ファイルの以下のサブシステム要素で定義されます。

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

リソースアダプターの *-ds.xml ファイルで以前に定義された同じデータソース情報の一部を使用します。

以下は、サーバー設定ファイルのリソースアダプター要素の例です。

```
<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.MultipleManagedConnectionFactory2"
      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.MultipleAdminObject1Impl"
      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.MultipleAdminObject2Impl"
      jndi-name="java:/eis/MultipleAdminObject2">
    <config-property name="Name">MultipleAdminObject2Value</config-property>
      </admin-object>
      </admin-objects>
  </resource-adapter>
</resource-adapters>
```

バグの報告

3.1.6.6. リークしたデータソース接続の検出

概要

JBoss EAP 6 では、Cached Connection Manager (CCM)デバッグユーティリティーを使用して、リークされたデータソース接続を検出できます。このトピックでは、CCM ユーティリティーを有効にしてデバッグする方法を説明します。

手順3.10 Cached Connection Managerの有効化

1. **<use-ccm="true"** を設定して、サーバー設定ファイルの **datasources** サブシステムで CCM を有効にします。これはデフォルト値であり、明示的に設定する必要はありません。

```
<subsystem xmlns="urn:jboss:domain:datasources:1.2">
  <datasources>
```

```
    <datasource ... enabled="true" use-ccm="true">
      ...
    </datasource>
  </datasources>
</subsystem>
```

2. **<cached-connection-manager> がサーバー設定ファイルの jca サブシステムに存在することを確認します。debug 属性を true に設定します。**

```
<subsystem xmlns="urn:jboss:domain:jca:1.1">
  ...
  <cached-connection-manager debug="true" error="true"/>
  ...
</subsystem>
```

debug="true" を設定すると、以下が発生します。

- **"Closing a connection for you. Please close them yourself"というメッセージと共に、INFOメッセージがログに記録されます。**
- **リークした接続を開いたコード用にスタックトレースが生成されます。**
- **リークした接続が閉じられます。**

追加のプロパティー error="true" を使用すると、RuntimeException を発生させ、ログに ERROR メッセージを生成できます。

3. **デバッグを有効にすることは、パフォーマンスおよびログファイルのサイズに影響を及ぼすため、テスト時のみを使用することが推奨されます。リークが残っていないことを確認してから、アプリケーションを実稼働環境にデプロイする前に、debug="true" 設定を削除するか、<cached-connection-manager debug="false"/> を使用して、設定を元に戻します。**

手順3.11 Cached Connection Manager によって報告されないリークのデバッグ

1. **サーバー設定ファイルの datasource サブシステムに use-ccm="false"が設定されていないことを確認します。**
2. **サーバー設定ファイルの datasource サブシステムに jta="false"が設定されていないことを確認します。**
3. **org.jboss.jca では、最小ロギングレベルが INFO に設定されていることを確認します。**

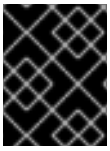
バグの報告

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ーの変更の設定

Basic 認証のセキュリティー設定

これまでのバージョンの 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 によって簡単に確認できました。JBoss EAP 6 では、ディレクトリー構造が変更になりました。プロパティーファイルは、クラスパスで利用できるようにするためにアプリケーション内にパッケージ化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。

Basic 認証のセキュリティーを設定するには、**standalone/configuration/standalone.xml** または **domain/configuration/domain.xml** サーバー設定ファイルの**security-domains** の下に、新しいセキュリティードメインを追加します。

```
<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には、**${jboss.server.config.dir}*EAP_HOME*/Standalone/configuration/** ディレクトリーを参照します。インスタンスが管理対象ドメインで実行されている場合には、**${jboss.server.config.dir}*EAP_HOME*/domain/configuration/** ディレクトリーを参照します。

セキュリティードメイン名の変更

JBoss EAP 6 では、セキュリティードメインの名前にプレフィックス **java:/jaas/** が使用されなくなりました。

- Web アプリケーションの場合、**jboss-web.xml** のセキュリティードメイン設定からこのプレフィックス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、**jboss-ejb3.xml** ファイルのセキュリティードメイン設定からこのプレフィックスを削除する必要があります。JBoss EAP 6 では、このファイルは **jboss.xml** に置き換えられています。

バグの報告

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 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 以降に移行するときに変更が必要になる場合があります。CVE-2013-2133 に対応するために 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された JAXWS ハンドラーを実行する前に、コンテナーによる承認チェックが実施されます。したがって、PicketLink **SAML2Handler** は後でプロセスで使用されるセキュリティープリンシパルを確立するため、PicketLink STS 機能の一部が影響を受ける可能性があります。**HandlerAuthInterceptor** が **SAML2Handler** にアクセスする際にプリンシパルが **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3.12 追加の承認チェックの無効化

- 以下の方法のいずれかを使用して、追加の承認チェックを無効にし、既存の PicketLink デプロイメントを引き続き使用できます。
  - システム全体のプロパティーを設定します。

    org.jboss.ws.cxf.disableHandlerAuthChecks システムプロパティーの値を **true** に設定すると、サーバーレベルで追加の承認チェックを無効にできます。この方法は、アプリケーションサーバーに加えら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ーの設定方法は、JBoss EAPの『Administration and Configuration Guide』の『Configure System Properties Using the Management CL』というトピックを参照してください。

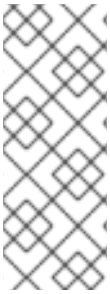
  - デプロイメントの Web サービス記述子ファイルにプロパティーを作成します。

    jboss-webservices.xml ファイルで **org.jboss.ws.cxf.disableHandlerAuthChecks** プロパティーの値を **true** に設定すると、デプロイメントレベルで追加の承認チェックを無効にできます。この方法は、特定のデプロイメントにのみ影響します。

    a. 追加の承認チェックを無効にするデプロイメントの **META-INF/** ディレクトリーに **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

org.jboss.ws.cxf.disableHandlerAuthChecks プロパティーを有効にすると、CVE-2013-2133 に対してシステムが脆弱になります。アプリケーションが EJB メソッドで宣言されたセキュリティー制限が適用されることを期待しているにも関わらず、JAX-WS ハンドラーに依存しないセキュリティー制限を適用しない場合、プロパティーを有効化しないでください。このプロパティーは、アプリケーションの破損を回避するために必要な場合、後方互換性の目的でのみ使用してください。

バグの報告

3.1.8. JNDI の変更

3.1.8.1. アプリケーション JNDI 名前空間名の更新

概要

EJB 3.1 では、標準化されたグローバル JNDI 名前空間と、Java EE アプリケーションのさまざまなスコープにマッピングする関連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**、および **java:app** の 3 つのみにバインドされます。JNDI を使用する EJB

のアプリケーションは、新しい標準化された JNDI 名前空間規則に従うように変更する必要があります。

手順3.13 JNDI ルックアップの変更

1. 以下について詳しく確認する 「移植可能な EJB JNDI 名」
2. 「JNDI 名前空間ルールの確認」
3. 「新しい JNDI 名前空間ルールに従うようにアプリケーションを変更」

JNDI マッピングの例

以前のリリースの JNDI 名前空間の例と JBoss EAP 6 でどのように指定されているかは、 「以前のリリースの JNDI 名前空間の例と JBoss EAP 6 での指定方法」を参照してください。

バグの報告

3.1.8.2. 移植可能な EJB JNDI 名

概要

Java EE 6 仕様は、それぞれ独自のスコープを持つ 4 つの論理名前空間を定義しますが、移植可能な EJB 名はその内の 3 つにのみバインドされます。以下の表は、各名前空間の使用方法と使用するタイミングの詳細を示しています。

表3.1 移植可能な JNDI 名前空間

JNDI 名前空間	説明
java:global	この名前空間内の名前は、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によって共有されます。この名前空間の名前を使用して、同じサーバーにデプロイされた EJB 外部アーカイブを検索します。 **java:global/jboss-seam-booking/jboss-seam-booking-jar/HotelBookingAction**は、java:global 名前空間の例です。
java:module	この名前空間内の名前は、モジュールのすべてのコンポーネント(例:単一の EJB モジュールのすべてのエンタープライズBeanや Web モジュールのすべてのコンポーネント)によって共有されます。 **java:module/HotelBookingAction!org.jboss.seam.example.booking.HotelBooking**は、java:module 名前空間の例です。
java:app	この名前空間内の名前は、単一アプリケーションのすべてのモジュールのすべてのコンポーネントで共有されます。たとえば、同じ EAR ファイルの WAR と EJB jar ファイルは java:app 名前空間内のリソースにアクセスできます。 **java:app/jboss-seam-booking-jar/HotelBookingAction**は、java:app 名前空間の例です。

JNDI の命名コンテキストに関する詳細は、「JSR 316: JavaTM Platform, Enterprise Edition (Java EE) Specification, v6」のセクション EE.5.2.2「Application Component Environment Namespaces」を参照してください。仕様は、http://jcp.org/en/jsr/detail?id=316からダウンロードできます。

バグの報告

3.1.8.3. JNDI 名前空間ルールの確認

概要

JBoss EAP 6 で JNDI 名前空間名が改善され、アプリケーションサーバーにバインドされるすべての名前に対して予測可能かつ一貫性のあるルールを提供するだけでなく、今後の互換性の問題を回避します。したがって、アプリケーションの現在の名前空間が新しいルールに準拠しない場合に、問題が発生する可能性があります。

名前空間は以下のルールに従う必要があります。

1. DefaultDS や jdbc/DefaultDS などの修飾されていない相対名は、コンテキストに応じて java:comp/env、java:module/env、または java:jboss/env との相対値として修飾する必要があります。
2. /jdbc/DefaultDS などの修飾されていない 絶対 名は、java:jboss/root 名との相対値として修飾する必要があります。
3. java:/jdbc/DefaultDS などの修飾された 絶対 名は、上記の非修飾 絶対 名と同じ方法で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: プレフィックスの 相対 名は、5つの名前空間comp、module、app、global、またはプロプライエタリーの jboss のいずれかになければなりません。java:xxx (ここで、*xxx* は上記の 5 つのいずれとも一致しない)で始まる名前の場合、無効な名前エラーが発生します。

バグの報告

3.1.8.4. 新しい JNDI 名前空間ルールに従うようにアプリケーションを変更

- 以下は、JBoss EAP 5.1 の JNDI ルックアップの例になります。このコードは、通常初期化メソッドにあります。

```
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 であることに注意してください。

- 以下は、依存性注入を使用して同じルックアップを JBoss EAP 6 でコーディングする方法の例になります。

```
@EJB(lookup="java:app/OrderManagerEJB/ProductManagerBean!services.ejb.Product
tManager")
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 java:app/OrderManagerEJB/ProductManagerBean!services.ejb.ProductManager を使用するようになりました。

- 依存性注入を使用しない場合は、上記のように新しい InitialContext を作成し、新しい JNDI 名前空間名を使用するようにルックアップを変更できます。

```
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);
}
```

バグの報告

3.1.8.5. 以前のリリースの JNDI 名前空間の例と JBoss EAP 6 での指定方法

表3.2 JNDI 名前空間マッピングテーブル

JBoss EAP 5.x での名前空間	JBoss EAP 6 での名前空間	追加のコメント
OrderManagerApp/ProductManagerBean/local	java:module/ProductManagerBean!services.ejb.ProductManager	Java EE 6 の標準バインディング。現在のモジュールがスコープで、同じモジュール内でのみアクセスできます。
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がスコープで、同じアプリケーション内でのみアクセスできます。
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がスコープで、グローバルにアクセスできます。
java:comp/UserTransaction	java:comp/UserTransaction	名前空間は現在のコンポーネントがスコープです。Java EE 6 ではないスレッド（例：アプリケーションが直接作成したスレッド）から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。java:comp/UserTransaction が利用できない場合に使用します。
java:/TransactionManager	java:jboss/TransactionManager	

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

バグの報告

3.2. アプリケーションアーキテクチャーおよびコンポーネントに応じた変更

3.2.1. アプリケーションアーキテクチャーおよびコンポーネントに応じた変更の確認

アプリケーションが以下のテクノロジーまたはコンポーネントのいずれかを使用する場合、JBoss EAP 6 への移行時にアプリケーションに変更を加える必要がある場合があります。

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、アプリケーションの変更が必要になる場合があります。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 を使用する場合は、JBoss EAP 6 が RESTEasy を自動的に設定するため、自分で設定する必要がないことに注意してください。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 セキュリティーレルムは異なる方法で設定されます。アプリケーションが LDAP を使用する場合は、「LDAP セキュリティーレルムの変更の設定」のトピックで詳細を参照してください。

Messaging

JBoss Messaging は JBoss EAP 6 に含まれなくなりました。アプリケーションが JBoss Messaging をメッセージングプロバイダーとして使用する場合は、JBoss Messaging コードを HornetQ に置き換える必要があります。「HornetQ を JMS プロバイダーとして使用するためのアプリケーションの移行」のトピックで、必要な操作について説明します。

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、「クラスタリング向けにアプリケーションに変更を加える。」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 はサービススタイルの記述子を使用しなくなりましたが、コンテナーは可能な限り変更なしにサービススタイルの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は、「サービススタイルの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 のプロキシーを検索し、返されたプロキシーで呼び出しを行うことができます。必要な構文および名前空間の変更に関する詳細は、「リモート呼び出しを JBoss EAP 6 に作成する JBoss EAP 5 のデプロイされた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、必要な変更について「Seam 2.2 アーカイブの JBoss EAP 6 への移行」のトピックを参照してください。

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。

移行に影響を与える可能性のあるその他の変更

アプリケーションに影響を与える可能性がある JBoss EAP 6 のその他の変更については、「移行の影響を受ける可能性がある他の変更点についてよく知る」を参照してください。

バグの報告

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、以下のセクションを読んで JBoss EAP 6 に移行するために必要な変更を加えてください。

- 「Hibernate および JPA を使用するアプリケーションの変更の設定」
- 「Hibernate 4 を使用するように Hibernate 3 アプリケーションを更新」
- 「JPA 2.0 仕様に準拠するようにアプリケーションを更新」
- 「JPA/Hibernate 二次キャッシュの Infinispan への置き換え」
- 「Hibernate Validator 4 への移行」

バグの報告

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の設定

概要

アプリケーションに **persistence.xml** ファイルが含まれる場合や、コードが **@PersistenceContext** または **@PersistenceUnit** アノテーションを使用する場合、JBoss EAP 6 はデプロイメント時にこれを検出し、アプリケーションが JPA を使用すると仮定します。暗黙的に Hibernate 4 と他の依存関係をアプリケーションクラスパスに追加します。

アプリケーションが現在 Hibernate 3 ライブラリーを使用している場合は、ほとんどの場合、Hibernate 4 の使用に切り替えて正常に実行できます。ただし、アプリケーションのデプロイ時に **ClassNotFoundExceptions** が表示される場合は、以下の方法の 1 つを使用して解決できます。

重要

Hibernate を Seam 2.2 で直接使用するアプリケーションは、アプリケーション内にパッケージ化された Hibernate 3 のバージョンを使用する場合があります。JBoss EAP 6 の org.hibernate モジュールで提供される Hibernate 4 は Seam 2.2 ではサポートされません。この例は、最初の手順としてアプリケーションを JBoss EAP 6 で実行できるようにするのに役立つことを目的としています。Seam 2.2 アプリケーションと共に Hibernate 3 をパッケージ化するオプションはサポートされないことに注意してください。

手順3.14 アプリケーションの設定

1. 必要な Hibernate 3 JAR をアプリケーションライブラリーにコピーします。

   不足している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/** ディレクトリーにコピーするか、他の方法を使用してクラスパスに追加して、問題を解決できる場合があります。これにより、Hibernate バージョンの混合使用が原因で **ClassCastExceptions** やその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。その場合には、次の方法を使用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にサーバーに指示します。

   JBoss EAP 6 では、アプリケーションで Hibernate 3.5（またはそれ以降）の永続プロバイダー jar をパッケージ化できます。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ようにサーバーに指示するには、以下のように **persistence.xml** の **jboss.as.jpa.providerModule** を **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、アプリケーション内に永続プロバイダーの存在を検出し、Hibernate 3 ライブラリーを使用します。JPA 永続プロパティーの詳細は、「永続ユニットプロパティー」 を参照してください。

3. Hibernate 二次キャッシュを無効にします。

   Hibernate 3 の 二次キャッシュは、JBoss EAP 6 では以前のリリースと同じようには動作しません。アプリケーションで Hibernate 二次キャッシュを使用している場合は、Hibernate 4 にアップグレードするまで無効にする必要があります。二次キャッシュを無効にするには、**persistence.xml** ファイルで **<hibernate.cache.use_second_level_cache>** を **false** に設定します。

4. **org.jboss.ejb3.entity.ExtendedEntityManager** への参照を置き換えます。

   JBoss EAP 5 では、拡張永続コンテキスト管理のために **org.jboss.ejb3.entity.ExtendedEntityManager** クラスが **javax.persistence.EntityManager** を拡張しました。JBoss EAP 6 では、このクラスは **org.jboss.as.jpa.container.ExtendedEntityManager** クラスに置き換えられました。ただし、

プロプライエタリークラスの代わりに標準の Java EE 6 JPA API javax.persistence.EntityManager クラスを使用するようコードを更新することが推奨されます。

バグの報告

3.2.2.3. 永続ユニットプロパティー

Hibernate 4.x 設定プロパティー

JBoss EAP 6 は以下の Hibernate 4.x 設定プロパティー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ー

プロパティー名	デフォルト値	目的
hibernate.id.new_generator_mappings	true	この設定は、**@GeneratedValue(AUTO)** を使用して新規エンティティーの一意のインデックスキーの値を生成する場合に該当します。新規アプリケーションはデフォルト値の **true** を維持する必要があります。Hibernate 3.3.x を使用した既存のアプリケーションは、シーケンスオブジェクトまたは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ために **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** ファイルのこの値をオーバーライドできます。 この動作の詳細については、以下を参照してください。
hibernate.transaction.jta.platform	**org.hibernate.service.jta.platform.spi.JtaPlatform** インターフェイスのインスタンス	このクラスは、トランザクションマネージャー、ユーザートランザクション、トランザクション同期レジストリーを Hibernate に渡します。
hibernate.ejb.resource_scanner	**org.hibernate.ejb.packaging.Scanner** インターフェイスのインスタンス	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デプロイメントを迅速化する方法を認識します。
hibernate.transaction.manager_lookup_class		このプロパティーは、**hibernate.transaction.jta.platform** と競合する可能性があるため、persistence.xmlにあれば削除されます。
hibernate.session_factory_name	*QUALIFIED_PERSISTENCE_UNIT_NAME*	これは、アプリケーション名と永続ユニット名の組み合わせに設定されます。アプリケーションは異なる値を指定できますが、JBoss EAP 6 インスタンスのすべてのアプリケーションデプロイメントで一意である必要があります。
hibernate.session_factory_name_is_jndi	false	これは、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にのみ設定されます。

プロパティー名	デフォルト値	目的
hibernate.ejb.entitymanager_factory_name	*QUALIFIED_PERSISTENCE_UNIT_NAME*	これは、アプリケーション名と永続ユニット名の組み合わせに設定されます。アプリケーションは異なる値を指定できますが、JBoss EAP 6 インスタンスのすべてのアプリケーションデプロイメントで一意である必要があります。

Hibernate 4.x では、**new_generator_mappings が true に設定されている場合：**

- **@GeneratedValue(AUTO)は org.hibernate.id.enhanced.SequenceStyleGenerator にマッピングします。**
- **@GeneratedValue(TABLE)は org.hibernate.id.enhanced.TableGenerator にマッピングします。**
- **@GeneratedValue(SEQUENCE)は org.hibernate.id.enhanced.SequenceStyleGenerator にマッピングします。**

Hibernate 4.x では、**new_generator_mappings が false に設定されている場合：**

- **@GeneratedValue(AUTO) は**Hibernate の「native」にマッピングします。
- **@GeneratedValue(TABLE)は org.hibernate.id.MultipleHiLoPerTableGenerator にマッピングします。**
- **@GeneratedValue(SEQUENCE) は**Hibernate の「seqhilo」にマッピングします。

これらのプロパティーの詳細は、http://www.hibernate.org/docs **にアクセスし、**Hibernate 4.1 Developer Guide**を参照してください。**

JPA 永続プロパティー

以下の JPA プロパティーは 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ー

プロパティー名	デフォルト値	目的
jboss.as.jpa.providerModule	**org.hibernate**	永続プロバイダーモジュールの名前。 Hibernate 3 JAR がアプリケーションアーカイブに ある場合は、値を **hibernate3-bundled** にする必要があります。 永続プロバイダーがアプリケーションとともにパッケージ化されている場合は、この値は **application** である必要があります。

プロパティー名	デフォルト値	目的
jboss.as.jpa.adapterModule	**org.jboss.as.jpa.hibernate:4**	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。 現在の有効な値は以下のとおりです。 • **org.jboss.as.jpa.hibernate:4**: これは Hibernate 4 インテグレーションクラス用です。 • **org.jboss.as.jpa.hibernate:3**: これは Hibernate 3 インテグレーションクラス用です。

バグの報告

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

概要

Hibernate 4 を使用するようにアプリケーションを更新すると、一部の更新は一般的で、アプリケーションによって現在使用されている Hibernate のバージョンに関係なく適用されます。その他の更新では、アプリケーションが現在使用しているバージョンを決定する必要があります。

手順3.15 Hibernate 4 を使用するようにアプリケーションを更新

1. **JBoss EAP 6 では、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動作が変更されました。詳細は、「Hibernate のアイデンティティー自動生成値に関する既存動作の維持」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 のバージョンを確認し、以下の正しい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. **クラスター環境でアプリケーションを実行する予定の場合には、「クラスター環境で実行する移行した Seam および Hibernate アプリケーションの永続プロパティーの変更」を参照してください。**

バグの報告

3.2.2.5. Hibernate のアイデンティティー自動生成値に関する既存動作の維持

Hibernate 3.5 では、@GeneratedValue の使用時にアイデンティティー列またはシーケンス列が生成される方法を示す hibernate.id.new_generator_mappings という名前のコアプロパティーが導入されました。JBoss EAP 6 では、このプロパティーのデフォルト値は以下のように設定されます。

- **ネイティブ 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 です。**
- **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 です。**

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規アプリケーション

は、hibernate.id.new_generator_mappings プロパティーの値を true に設定する必要があります。異なるデータベース間で移植できるので、この設定が推奨されます。多くの場合、効率的であり、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 は hibernate.id.new_generator_mappings プロパティーを true にデフォルト設定するため、変更しないでください。
- 新しいネイティブ Hibernate アプリケーションの場合、JBoss EAP 6 は hibernate.id.new_generator_mappings プロパティーを false にデフォルト設定します。このプロパティーを true に設定する必要があります。

既存の JBoss EAP 5 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既存のアプリケーションは、アプリケーションが JBoss EAP 6 に移行するときに、新しいエンティティーのプライマリーキー値を作成するために同じジェネレーターが使用されることを確認する必要があります。

- 既存の JPA アプリケーションの場合、JBoss EAP 6 は hibernate.id.new_generator_mappings プロパティーを true にデフォルト設定します。persistence.xml ファイルで、このプロパティーを false に設定する必要があります。
- 既存のネイティブ Hibernate アプリケーションの場合、JBoss EAP 6 は hibernate.id.new_generator_mappings を false にデフォルト設定するため、変更しないでください。

このプロパティー設定の詳細は、「永続ユニットプロパティー」 を参照してください。

バグの報告

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 テキスト タイプは JDBC LONGVARCHARにマッピングされるようになりました。

   3.5 よりも前のバージョンの Hibernate では、テキスト タイプは JDBC CLOB にマッピングされました。新しい Hibernate タイプ materialized_clob が Hibernate 4 に追加され、Java String プロパティーを JDBC CLOB にマッピングします。JDBC CLOB にマッピングすることが意図されたアプリケーションのプロパティーが type="text" として設定されている場合、以下のいずれかを実行する必要があります。

   a. 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は、プロパティーを type="materialized_clob" に変更します。

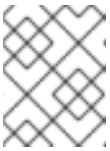
   b. アプリケーションがアノテーションを使用する場合は、@Type(type = "text") を @Lob に置き換える必要があります。

2. 戻り値タイプの変更を探すコードの確認

   数値集約基準のプロジェクションは、HQL のカウンターパートと同じ値タイプを返すようになりました。その結果、org.hibernate.criterion の以下のプロジェクションからの戻り値の型が変更されました。

   a. CountProjection、Projections.rowCount()、Projections.count(propertyName)、Projections.countDistinct(propertyName) の変更により、count および count distinct プロジェクションは Long 値を返すようになりました。

   b. Projections.sum(propertyName) の変更により、sum プロジェクションはプロパティータイプに依存する値タイプを返すようになりました。



注記

アプリケーションコードの変更に失敗する
と、java.lang.ClassCastException が発生することがあります。

i. Long、Short、Integer、またはプリミティブ整数タイプとしてマッピングされたプロパティーの場合、Long 値が返されます。

ii. Float、Double、またはプリミティブ浮動小数点タイプとしてマッピングされたプロパティーの場合は、Double 値が返されます。

3. nullSafeGet() および nullSafeSet() メソッドの UserType 署名を更新します。

UserType クラスの nullSafeGet() および nullSafeSet() メソッドの署名が Hibernate 4 で変更になりました。これらのメソッドを使用するアプリケーションコードは、新しい署名を使用するように更新する必要があります。

以下は、Hibernate 3.x のメソッド署名の例です。

```
public Object nullSafeGet(ResultSet rs, String[] names, Object owner)
    throws HibernateException, SQLException;

public void nullSafeSet(PreparedStatement st, Object value, int index)
    throws HibernateException, SQLException;
```

以下は、Hibernate 4 の新しいメソッド署名の例です。

```
public Object nullSafeGet(ResultSet rs, String[] names, SessionImplementor session,
Object owner)
    throws HibernateException, SQLException;

public void nullSafeSet(PreparedStatement st, Object value, int index,
SessionImplementor session)
    throws HibernateException, SQLException;
```

バグの報告

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 が Configurationにマージされました。

AnnotationConfiguration は非推奨になりましたが、移行には影響しません。

a. hbm.xml ファイルをまだ使用している場合は、JBoss EAP 6 では、以前のリリースで使用されていた org.hibernate.cfg.DefaultNamingStrategy ではなく、AnnotationConfiguration の org.hibernate.cfg.EJB3NamingStrategy が使用されるようになったことに注意してください。これにより、命名が一致しなくなる可能性があります。

b. 関連付け（多対多および要素のコレクション）表の名前のデフォルト設定を命名ストラテジーに依存している場合、この問題が発生する可能性があります。この問題を解決するには、Configuration#setNamingStrategy を呼び出して org.hibernate.cfg.DefaultNamingStrategy#INSTANCE に渡すことで、レガシー org.hibernate.cfg.DefaultNamingStrategy を使用するように Hibernate に指示できます。

2. 以下の表にあるように、新しい Hibernate DTD ファイル名に準拠するように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
http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd	http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd
http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd	http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd

3. 環境変数を変更します。

   a. Oracle を使用し、**materialized_clob** または **materialized_blob** プロパティーを使用している場合は、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 を使用し、**CLOB** または **BLOB** プロパティーを使用している場合は、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。

4. **nullSafeGet()** および **nullSafeSet()** メソッドの **UserType** 署名を更新します。

   UserType クラスの **nullSafeGet()** および **nullSafeSet()** メソッドの署名が Hibernate 4 で変更になりました。これらのメソッドを使用するアプリケーションコードは、新しい署名を使用するように更新する必要があります。詳細は、「Hibernate 3.3.x アプリケーションの Hibernate 4.x への移行」を参照してください。

バグの報告

3.2.2.8. クラスター環境で実行する移行した Seam および Hibernate アプリケーションの永続プロパティーの変更

JPA コンテナー管理のアプリケーションを移行する場合、拡張永続コンテキストのシリアライズに影響するプロパティーはコンテナーに自動的に渡されます。

ただし、Hibernate の変更により、移行された Seam または Hibernate アプリケーションをクラスター環境で実行すると、シリアライズの問題が発生することがあります。以下のようなエラーログメッセージが記録されるはずです。

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

これらのエラーを解決するには、設定ファイルのプロパティーを変更する必要があります。ほとんどの場合、これは **persistence.xml** ファイルです。ネイティブ Hibernate API アプリケーションの場合、これは **hibernate.cfg.xml** ファイルです。

手順3.16 クラスター環境で実行するための永続プロパティーの設定

1. **hibernate.session_factory_name** の値を一意の名前に設定します。その名前は JBoss EAP

1. hibernate.session_factory_name の値を一意の名前に設定します。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である必要があります。以下に例を示します。

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

2. hibernate.ejb.entitymanager_factory_name の値を一意の名前に設定します。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である必要があります。以下に例を示します。

```
<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ーの設定に関する詳細は、「永続ユニットプロパティー」を参照してください。

バグの報告

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

概要

JPA 2.0 仕様では、永続コンテキストを JTA トランザクションの外部に伝播できないことが求められます。アプリケーションがトランザクションスコープの永続コンテキストのみを使用する場合、JBoss EAP 6 での動作は以前のバージョンのアプリケーションサーバーと同じで、変更する必要はありません。ただし、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イングやバッチングを許可する場合は、アプリケーションに変更を加える必要がある場合があります。

永続コンテキストの伝播動作

アプリケーションに、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 **Bean1** があり、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 **Bean2** を呼び出す場合は、以下の動作が発生することが予想されます。

- **Bean1** が JTA トランザクションを開始し、JTA トランザクションを有効にして **Bean2** メソッドの呼び出しを行う場合、JBoss EAP 6 での動作は以前のリリースと同じであるため、変更す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** メソッド呼び出しを行う場合、JBoss EAP 6 は拡張永続コンテキストを **Bean2** に伝播しません。この動作は、拡張永続コンテキストを **Bean2** に伝播していた以前のリリースとは異なります。アプリケーションが、トランザクションエンティティーマネージャーにより拡張永続コンテキストが Bean に伝播されることを期待する場合は、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。

バグの報告

3.2.2.10. JPA/Hibernate 二次キャッシュの Infinispan への置き換え

概要

二次キャッシュ(2LC)については、JBoss キャッシュが Infinispan に置き換えられました。これには、**persistence.xml** ファイルを変更する必要があります。JPA または Hibernate 二次キャッシュを使用しているかどうかによって、構文は若干異なります。これらの例では、Hibernate を使用していることを前提としています。

これは、JBoss EAP 5.x の **persistence.xml** ファイルで二次キャッシュのプロパティーがどのように指定されるかの例になります。

```
<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3.17 Infinispan を使用するように **persistence.xml** ファイルを変更します。

1. JBoss EAP 6 での JPA アプリケーション用 Infinispan の設定

   以下は、JBoss EAP 6 で Infinispan を使用して JPA アプリケーション用に同じ設定を実現するためにプロパティーを指定する方法の例です。

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

   さらに、以下のように **ENABLE_SELECTIVE** または **ALL** の値で **shared-cache-mode** を指定する必要があります。

   - **ENABLE_SELECTIVE** はデフォルト値であり、推奨される値です。つまり、エンティティーを明示的にキャッシュ可能としてマークしない限り、エンティティーはキャッシュされません。

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

   - **ALL** は、エンティティーをキャッシュ可能ではないとマークしても常にキャッシュされることを意味します。

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

2. JBoss EAP 6 でのネイティブ Hibernate アプリケーション用 Infinispan の設定

   以下は、JBoss EAP 6 で Infinispan を使用してネイティブ Hibernate アプリケーション用に同じ設定を指定する方法の例です。

   ```
   <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.jta.platform"
       value="org.hibernate.service.jta.platform.internal.JBossAppServerJtaPlatform"/>
   <property name="hibernate.cache.use_second_level_cache" value="true"/>
   ```

   MANIFEST.MF ファイルに以下の依存関係も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ーの詳細は、「Hibernate キャッシュプロパティー」を参照してください。

バグの報告

3.2.2.11. Hibernate キャッシュプロパティー

表3.6 プロパティー

プロパティー名	説明
hibernate.cache.region.factory_class	カスタム **CacheProvider** のクラス名。
hibernate.cache.use_minimal_puts	ブール値。書き込みを最小限に抑えるために 2 次レベルのキャッシュ操作を最適化し、読み取りの頻度を上げます。この設定はクラスター化されたキャッシュで最も便利なもので、Hibernate3 ではクラスター化されたキャッシュ実装に対してデフォルトで有効になっています。
hibernate.cache.use_query_cache	ブール値。クエリーキャッシュを有効にします。個別のクエリーをキャッシュ可能に設定する必要があります。
hibernate.cache.use_second_level_cache	ブール値。**<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 次レベルのキャッシュを完全に無効にするために使用されます。
hibernate.cache.query_cache_factory	カスタム **QueryCache** インターフェイスのクラス名。デフォルト値はビルトインの **StandardQueryCache** です。
hibernate.cache.region_prefix	2 次キャッシュリージョン名に使用するプレフィックス。
hibernate.cache.use_structured_entries	ブール値。Hibernate が、よりヒューマンリーダブルな形式で、2 次レベルのキャッシュにデータを格納するように強制します。
hibernate.cache.default_cache_concurrency_strategy	**@Cacheable** または **@Cache** のいずれかが使用された場合に使用するデフォルトの **org.hibernate.annotations.CacheConcurrencyStrategy** の名前を付与するために使用される設定。**@Cache(strategy="..")** は、このデフォルトを上書きするのに使用されます。

バグの報告

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比較的直接的ですが、アプリケーションの移行時にいくつかの変更を加える必要があります。

手順3.18 以下のタスクの 1 つまたは複数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** という名前のセクションの JNDI コンテキストにバインドします。

2. ライフサイクルによりトリガーされる検証について

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

   a. 検証は、エンティティーの **INSERT**、**UPDATE**、**DELETE** 操作で実行されます。

   b. 以下のプロパティーを使用して、イベントタイプで検証するようにグループを設定できます。

      - **javax.persistence.validation.group.pre-persist**、
      - **javax.persistence.validation.group.pre-update**、および
      - **javax.persistence.validation.group.pre-remove**

      これらのプロパティーの値は、検証するグループの完全修飾クラス名のコンマ区切りリストです。

      検証グループは Bean Validation 仕様の新しい機能です。この新機能を活用しない場合は、Hibernate Validator 4 に移行するときに変更は必要ありません。

   c. **javax.persistence.validation.mode** プロパティーを **none** に設定すると、ライフサイクルベースの検証を無効にできます。このプロパティーの他の有効な値は、**auto**（デフォルト）、**callback**、および **ddl** です。

3. 手動検証を使用するようにアプリケーションを設定

   a. 検証を手動で制御するには、以下のいずれかの方法でValidatorを作成します。

      - **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。
      - EJB、CDI Bean、またはその他の Java EE の注入可能なリソースにValidatorインスタンスを注入します。

   b. **ValidatorFactory.usingContext()** によって返される **ValidatorContext** を使用してValidatorインスタンスをカスタマイズできます。この API を使用すると、カスタム **MessageInterpolator**、**TraverableResolver**、および **ConstraintValidatorFactory** を設定できます。これらのインターフェイスは Bean Validation 仕様で指定され、Hibernate Validator 4 の新機能です。

4. 新しい Bean Validation 制約を使用するためのコードの変更

   Hibernate Validator 4 に移行する際、新しい Bean レベルの検証制約に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には、以下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 に存在していた制約はすべて Hibernate Validator 4 で引き続き利用できます。これらを使用するには、指定のクラスをインポートする必要があり、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します。

5. **カスタム制約の使用**

   Hibernate Validator 3 では、**org.hibernate.validator.Validator** インターフェイスを実装するのにカスタム制約が必要でした。Hibernate Validator 4 では、**javax.validation.ConstraintValidator** インターフェイスを実装する必要があります。このインターフェイスには、以前のインターフェイスと同じ **initialize()** メソッドおよび **isValid()** メソッドが含まれますが、メソッドの署名が変更されました。さらに、Hibernate Validator 4 では DDL の変更に対応しなくなりました。

6. 以下の Hibernate 3.x **Validator** クラスは廃止されました。

   a. **org.hibernate.validator.NotNull** を **javax.validation.constraints.NotNull** に置き換えます。

   b. **org.hibernate.validator.ClassValidator** を **javax.validation.Validator.** に置き換えます。

   c. **org.hibernate.validator.InvalidValue** を **javax.validation.Validator.** に置き換えます。

   詳細は、『Hibernate Valitator Reference Guide』を参照してください。

7. **jboss-ejb3-ext-api-*VERSION*.jar** ファイルでは、多くのアノテーションエクステンション API が削除されました。JBoss EAP 6 では必要なくなったためです。たとえば、JBoss EAP 6 は相互互換性を自動的に処理するため、**org.jboss.ejb3.annotation.IgnoreDependency** クラスは利用できないか、または不要になりました。

バグの報告

3.2.3. JTS および JTA の変更

3.2.3.1. JBoss Transaction Service 設定の移行

概要

これまでのバージョンの JBoss EAP では、JBoss Transaction Service トランザクションマネージャーは以下の XML ファイルのいずれかで設定されていました。

表3.7 JBossTS 設定ファイル

JBoss EAP バージョン	設定ファイル名
4.2	jboss-eap-4.2.0/server/default/conf/jbossjta-properties.xml
4.3	jboss-eap-4.3.0/server/default/conf/jbossjta-properties.xml
5.2	jboss-eap-5.2.0/server/default/conf/jbossts-properties.xml

JBoss EAP 6 では、トランザクションサービスはサーバー設定ファイルで設定されます。

- JBoss EAP 6 には、ノード識別子のデフォルト値が含まれています。これは、単一のJBoss EAP サーバーインスタンスを実行する場合は問題ありませんが、複数のサーバーインスタンスを実行する場合は変更する必要があります。
- JBoss EAP 6 には、デフォルトで JTA トランザクションが有効になっています。JTS トランザクションを設定するには、追加の手順が必要です。

JTA トランザクションのノード識別子設定の移行

JBoss EAP 6 には、ノード識別子のデフォルト設定値が同梱されます。これは、単一のJBoss EAP サーバーインスタンスを実行する場合は問題ありませんが、ノード識別子はすべての JBoss EAP サーバーインスタンスについて一意でなければならないので、複数のサーバーインスタンスを実行する場合は値を変更する必要があります。

注記

以下の例では、置き換え可能な値 *UNIQUE_NODE_ID* および *UNIQUE_JACORB_ID* を使用して、一意のノードおよび JacORB ID を表します。必ず、JBoss EAP サーバーインスタンス全体で一意となる識別子の値に置き換えてください。

JBoss EAP 5 では、以下のプロパティーを使用してノード識別子が **jbossts-properties.xml** ファイルで設定されていました。

```
<property name="com.arjuna.ats.arjuna.xa.nodeIdentifier" value=UNIQUE_NODE_ID/>
```

JBoss EAP 6 では、ノード識別子は、管理 CLI コマンドを使用してサーバー設定ファイルの **transaction** サブシステムで設定されます。使用するコマンドは、管理対象ドメインとスタンドアロンサーバーのどちらを実行しているかによって異なります。

以下は、スタンドアロンサーバーのノード識別子を設定するコマンドの例になります。

```
/system-property=jboss.tx.node.id:add(value=UNIQUE_NODE_ID)
/subsystem=transactions:write-attribute(name=node-identifier,value="${jboss.tx.node.id}")
reload
```

以下は、管理対象ドメインのノード識別子を設定する方法の例になります。管理対象ドメインで実行している場合は、ドメインのすべてのホストおよびサーバーについて、一意の識別子で以下のコマンドを繰り返します。

```
/host=master/server-config=server-one/system-property=jboss.tx.node.id:add(boot-time=true,value=UNIQUE_NODE_ID)
/profile=PROFILE_NAME/subsystem=transactions:write-attribute(name=node-identifier,value="${jboss.tx.node.id}")
reload
```

JBoss EAP 6 で JTS トランザクションを有効にするための変更

JBoss EAP 5 では、***EAP5_HOME***/docs/examples/transactions ディレクトリーにある Ant スクリプトを実行して JTS トランザクションを有効にし、いくつかの手動の手順を実行していました。このスクリプトは、すべての JBoss EAP サーバー設定の **jbossts-properties.xml** および **jacorb.properties** ファイルが更新されました。

JBoss EAP 6 では、管理 CLI を使用して JBoss EAP インスタンスおよび JacORB インスタンスを一意に識別します。使用するコマンドは、管理対象ドメインとスタンドアロンサーバーのどちらを実行しているかによって異なります。

以下は、スタンドアロンサーバーの JBoss EAP および JacORB インスタンスを識別するスクリプトの例です。

```
batch

# Create a system property for the unique node identifier

/system-property=jboss.tx.node.id:add(value=UNIQUE_NODE_ID)
/system-property=jacorb.node.id:add(value=UNIQUE_JACORB_ID)

# JacORB properties must be unique for each JBoss server instance
# JacORB name must appear in JacORB context root i.e. ${jacorb.name}/Naming/root

/subsystem=jacorb:write-attribute(name=transactions,value="on")
/subsystem=jacorb:write-attribute(name="name",value="${jacorb.node.id}")
/subsystem=jacorb:write-attribute(name="root-
context",value="${jacorb.node.id}/Naming/root")

/subsystem=transactions:write-attribute(name=jts,value=true)
/subsystem=transactions:write-attribute(name=node-identifier,value="${jboss.tx.node.id}")

run-batch

reload
```

以下は、管理対象ドメインの JBoss EAP および JacORB インスタンスを識別するスクリプトの例です。管理対象ドメインで実行している場合は、設定のプロファイルごとに以下のスクリプトを実行する必要があります。

```
batch

#
# Define global system properties for the node identifier and JacORB implementation name.
#
/system-property=jboss.tx.node.id/:add(value="11",boot-time="true")
/system-property=jacorb.node.id:add(value="mars",boot-time="true")

/profile=PROFILE_NAME/subsystem=jacorb:write-attribute(name="security",value="on")
/profile=PROFILE_NAME/subsystem=jacorb:write-attribute(name="transactions",value="on")
/profile=PROFILE_NAME/subsystem=jacorb:write-
attribute(name="name",value="${jacorb.node.id}")
/profile=PROFILE_NAME/subsystem=jacorb:write-attribute(name="root-
context",value="${jacorb.node.id}/Naming/root")

/profile=PROFILE_NAME/subsystem=transactions:write-attribute(name="jts",value="true")
/profile=PROFILE_NAME/subsystem=transactions:write-attribute(name="node-
identifier",value="${jboss.tx.node.id:1}")

run-batch

reload --host=master
```

バグの報告

3.2.4. JSF の変更

3.2.4.1. 以前のバージョンの JSF を使用するアプリケーションの有効化

概要

アプリケーションが古いバージョンの JSF 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 を使用するように要求できます。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のに使用され、WAR の **META-INF/** または **WEB-INF/** ディレクトリー、または EAR の**META-INF/** ディレクトリーに配置されます。

以下は、JSF 1.2 モジュールの依存関係を追加し、JSF 2.0 モジュールの自動読み込みを除外または防止する **jboss-deployment-structure.xml** ファイルの例です。

```
<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>
```

バグの報告

3.2.5. キャッシュの変更

3.2.5.1. JBoss キャッシュの置き換え

これまでのバージョンの JBoss EAP では、JBoss Cache はサーバーの内部ニーズに対して基礎となる分散キャッシュのサポートを提供していました。また、JBoss EAP アプリケーションのキャッシュも提供しました。

JBoss EAP 6 では、サーバーによる内部使用のためだけに、JBoss Cache が Infinispan に置き換えられました。これらの内部キャッシュは、セッションレプリケーション、ステートフルセッション Bean レプリケーション、および JPA-Hibernate の二次キャッシュを制御し、サーバー設定ファイルの **infinispan** サブシステムで設定されます。これらがアプリケーションによって直接使用されることはなく、この目的ではサポートされません。

以前 JBoss Cache を使用していたアプリケーションは、Red Hat JBoss Data Grid を使用するようにコードを移行する必要があります。これはサポートされているソリューションであり、別のエンタイトルメントが必要です。JBoss Data Grid は、2 つの使用モードで利用できます。

リモートクライアント/サーバー

このモードは、管理、分散、およびクラスター化が可能なデータグリッドサーバーを提供します。アプリケーションは、Hot Rod、memcached または REST クライアント API を使用して、データグリッドサーバーにリモートでアクセスできます。

ライブラリー

このモードでは、ユーザーはカスタムランタイム環境をビルドおよびデプロイできます。ライブラリー使用モードは、アプリケーションプロセスの単一のデータグリッドノードをホストし、このノードは他の JVM でホストされるノードへのリモートアクセスが可能です。

JBoss Data Grid の詳細は、カスタマーポータルで『Red Hat JBoss Data Grid』の『Getting Started Guide』を参照してください (https://access.redhat.com/documentation/ja-JP/Red_Hat_JBoss_Data_Grid/)。

バグの報告

3.2.6. Web サービスの変更

3.2.6.1. Web サービスの変更

JBoss EAP 6 には JAX-WS Web Service エンドポイントデプロイのサポートが含まれています。このサポートは JBossWS によって提供されます。Web Service の詳細は、JBoss EAP 6の『Development Guide』の『JAX-WS Web Services』の章を参照してください (https://access.redhat.com/documentation/ja-jp/red_hat_jboss_enterprise_application_platform/?version=6.4)。

JBossWS 4 には、移行に影響を与える可能性のある以下の変更が含まれています。

JBossWS API の変更

SPI および Common コンポーネントが、JBossWS 4 でリファクタリングされました。以下の表は、アプリケーションの移行に影響を及ぼす可能性がある API およびパッケージングの変更の一覧です。

表3.8 JBossWS API の変更

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.*
JBossWS SPI	org.jboss.wsf.spi.util.ServiceLoader	JBossWS API	org.jboss.ws.api.util.ServiceLoader

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 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
JBossWS Framework	org.jboss.wsf.framework.invocation.RecordingServerHandler	JBossWS Common	org.jboss.ws.common.invocation.RecordingServerHandler

@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 JAR で **org.jboss.wsf.spi.annotation.WebContext** としてパッケージ化されました。JBossWS 4.0 では、このアノテーションは JBossWS API JAR 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る場合、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、後方互換性のない属性にも変更があります。**String[] virtualHosts** 属性が **String virtualHost** に変更になりました。JBoss EAP 6 では、デプロイメントごとに 1 つの仮想ホストだけを指定できます。複数の Web サービスで **@WebContext** アノテーションが使用される場合、virtualHost の値はデプロイメントアーカイブで定義されるすべてのエンドポイントで同一である必要があります。

エンドポイント設定

JBossWS 4.0 は、ほとんどの Apache CXF モジュールで JBoss Web Services スタックを統合します。インテグレーションレイヤーにより、JAX-WS を含む標準の webservices API を使用できます。また、複雑な設定や構成を必要とせずに JBoss EAP 6 コンテナー上で Apache CFX の高度な機能を使用することもできます。

JBoss EAP 6 のドメイン設定の **webservice** サブシステムには事前に定義されたエンドポイント設定が含まれます。また、独自の追加のエンドポイント設定を定義することもできます。**@org.jboss.ws.api.annotation.EndpointConfig** アノテーションは、指定のエンドポイント設定を参照するために使用されます。

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は、JBoss EAP 6 の『開発ガイド』の「『JAX-WS Web Services』」というタイトルの章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 では、Web サービスを設定するための新しいデプロイメント記述子が導入されました。**jboss-webservices.xml** ファイルは指定のデプロイメントに関する追加情報を提供し、廃止された **jboss.xml** ファイルを部分的に置き換えます。

EJB webservice デプロイメントの場合は、**jboss-webservices.xml** 記述子ファイルの場所として **META-INF/** ディレクトリーに配置されることが想定されます。POJO および EJB webservice エンドポイントが WAR ファイルにバンドルされている場合には、**jboss-webservices.xml** ファイルの場所として **WEB-INF/** ディレクトリーに配置されることが想定されます。

以下は、**jboss-webservices.xml** 記述子ファイルと要素を記述するテーブルの例になります。

```
<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3.9 jboss-webservice.xml ファイル要素の説明

要素名	説明
context-root	webservices デプロイメントのコンテキストルートをカスタマイズするために使用されます。
config-name config-file	エンドポイントデプロイメントと指定のエンドポイント設定の関連付けに使用されます。エンドポイント設定は、参照される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。
プロパティー	単純なプロパティー名の値ペアを設定するのに使用し、webservice スタックの動作を設定します。
port-component	EJB エンドポイントのターゲット URI をカスタマイズしたり、セキュリティー関連のプロパティーを設定したりするために使用されます。
webservice-description	webservice WSDL で公開された場所をカスタマイズしたり上書きしたりするために使用されます。

バグの報告

3.2.6.2. JBoss EAP 6 での Apache Axis の使用

概要

Web サービスアプリケーションの開発およびデプロイには JBoss EAP 6 にバンドルされた JBossWS Apache CXF を使用することを推奨します。JBoss EAP 6 に同梱される JBossWS Apache CXF はサポート対象の設定です。Axis を使用する場合は、このトピックで JBoss EAP 6 での使用方法を説明します。この実装はアプリケーションの一部とみなされており、サポート対象の設定ではない点に注意してください。

手順3.19 Axis を使用するための JBoss EAP の設定

1. 最初に、JBoss EAP 6 に同梱される web サービス実装を無効にする必要があります。詳細な手順は、「JBoss EAP 6 での JBossWS の無効化」を参照してください。

2. アプリケーションを再ビルドし、デプロイします。Axis WS API 実装が Java EE 6 API クラスに正しくアクセスされるはずです。

バグの報告

3.2.6.3. JBoss EAP 6 での JBossWS の無効化

概要

Java Enterprise Edition Specification (Java EE) 6 では、Web サービスアプリケーションの開発およびデプロイに、アプリケーションサーバーが統合された JAX-WS 実装を提供する必要があります。JAX-WS のサポートを提供しないサーブレットコンテナーなど、他のコンテナーから移行するアプリケーションは、独自の JAX-WS 実装とともにパッケージ化される場合があります。JBossWS CXF を使用して独自の JAX-WS 実装をパッケージ化する予定がない場合は、JBoss EAP 6 で **webservices** サブシステムを無効にすることができます。JBoss EAP 6 に同梱される JBossWS Apache CXF はサポート対象の設定であることに注意してください。その他の実装はアプリケーションの一部とみなされ、サポート対象の設定ではありません。

このトピックでは、単一のデプロイメントおよびサーバーへのすべてのデプロイメントで JBossWS を無効にする方法を説明します。

手順3.20 単一デプロイメントでの JBossWS の無効化

1. アプリケーション用の **jboss-deployment-structure.xml** ファイルを作成します。

   - アプリケーションが WAR としてパッケージ化されてデプロイされている場合は、以下の内容で WAR の **META-INF/** または **WEB-INF/** ディレクトリーに、**jboss-deployment-structure.xml** ファイルを作成します。

     ```
     <jboss-deployment-structure xmlns="urn:jboss:deployment-structure:1.2">
       <deployment>
         <!-- Exclude webservices subsystem so the implicit dependencies will not be added -->
         <exclude-subsystems>
           <subsystem name="webservices"/>
         </exclude-subsystems>
       </deployment>
     </jboss-deployment-structure>
     ```

   - アプリケーションが EAR でパッケージ化されてデプロイされている場合は、以下の例のような内容で EAR の **META-INF/** ディレクトリーに **jboss-deployment-structure.xml** ファイルを作成します。

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-structure:1.2">
  ...
  <sub-deployment name="example.war">
    <!-- Exclude webservices subsystem so the implicit dependencies will not be
added -->
     <exclude-subsystems>
       <subsystem name="webservices"/>
    </exclude-subsystems>
    <dependencies>
      ...
    </dependencies>
  </sub-deployment>
</jboss-deployment-structure>
```

2. 通常の方法でアプリケーションをデプロイします。JBossWS がアプリケーションで無効になりました。

手順3.21 サーバーに対する全デプロイメントでの JBossWS の無効化

1. JBoss EAP サーバーを停止します。
2. 編集するため、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の場合には、これは *EAP_HOME*/standalone/configuration/standalone.xml ファイルです。管理対象ドメインの場合には、これは *EAP_HOME*/domain/configuration/domain.xml ファイルです。
3. org.jboss.as.webservices 拡張を見つけ、コメントアウトまたは削除します。
4. webservices サブシステムプロファイルを見つけ、コメントアウトまたは削除します。
5. 以下は、standalone.xml ファイルに加えられた変更の例になります。

```
<?xml version='1.0' encoding='UTF-8'?>
<server xmlns="urn:jboss:domain:1.7">
  <extensions>
      ...
    <!-- <extension module="org.jboss.as.webservices"/> -->
      ...
  </extensions>
  ...
  <profile>
    <!--
    <subsystem xmlns="urn:jboss:domain:webservices:1.2">
      <modify-wsdl-address>true</modify-wsdl-address>
      <wsdl-host>${jboss.bind.address:127.0.0.1}</wsdl-host>
      <endpoint-config name="Standard-Endpoint-Config"/>
      <endpoint-config name="Recording-Endpoint-Config">
        <pre-handler-chain name="recording-handlers" protocol-
bindings="##SOAP11_HTTP ##SOAP11_HTTP_MTOM ##SOAP12_HTTP
##SOAP12_HTTP_MTOM">
          <handler name="RecordingHandler"
class="org.jboss.ws.common.invocation.RecordingServerHandler"/>
        </pre-handler-chain>
      </endpoint-config>
     <client-config name="Standard-Client-Config"/>
    </subsystem>
```

```
    -->
  ...
  </profile>
  ...
</server>
```

バグの報告

3.2.7. JAX-RS および RESTEasy の変更

3.2.7.1. JAX-RS および RESTEasy の変更の設定

JBoss EAP 6 では RESTEasy が自動的に設定されるので、独自に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 設定をすべて削除し、これを以下の 3 つのオプションのいずれかに置き換える必要があります。

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

   これは最も簡単なオプションで、xml 設定は必要ありません。アプリケーションで javax.ws.rs.core.Application サブクラスに JAX-RS クラスを利用可能にするパスでアノテーションを付けるだけです。以下に例を示します。

   ```
   @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/ で利用できます。

   注記

   パスは /mypath/* ではなく、/mypath として指定する必要があり ます。末尾に、スラッシュまたはアスタリスクは付けないでください。

2. サブクラス javax.ws.rs.core.Application。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ない場合は、javax.ws.rs.core.Application のサブクラスは依然として必要です。次に、web.xml ファイルで JAX-RS マッピングを設定します。以下に例を示します。

   ```
   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 で利用できます。

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。**

   サブクラス Application を使用しない場合は、以下のように web.xml ファイルに JAX-RS マッピングを設定できます。

```
<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 で利用できます。

注記

このオプションを選択した場合には、マッピングを追加するだけです。対応するサーブレットを追加する必要はありません。サーバーは対応するサーブレットを自動的に追加します。

バグの報告

3.2.8. LDAP セキュリティーレルムの変更

3.2.8.1. LDAP セキュリティーレルムの変更の設定

JBoss EAP 5 では、LDAP セキュリティーレルムは login-config.xml ファイルの <application-policy> 要素に設定されました。JBoss EAP 6 では、LDAP セキュリティーレルムはサーバー設定ファイルの <security-domain> 要素に設定されます。スタンドアロンサーバーの場合、これは standalone/configuration/standalone.xml ファイルになります。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 ファイルになります。

以下は、JBoss EAP 5 の login-config.xml ファイルの LDAP セキュリティーレルムの設定例です。

```
<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>
```

以下は、JBoss EAP 6 のサーバー設定ファイルの LDAP 設定例です。

```
<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、以下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.LdapCtxFactory</module-
option>
```

モジュールオプションは、以下のように "value=" の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"/>
```

バグの報告

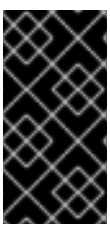
3.2.9. HornetQ の変更

3.2.9.1. HornetQ および NFS

多くの場合、同期ロックメカニズムの動作が原因で、NIO をジャーナルタイプとして使用する場合には、NFS は HornetQ で使用する JMS データの保存に適した方法ではありません。ただし、特定の状況では、NFS 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 で使用する NFS の実装が原因です。

Red Hat Enterprise Linux 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）とカーネルベースの非同期 I/O の両方をサポートします。これらの機能の両方が存在すると、厳格な設定ルールの下で NFS を共有ストレージオプションとして使用できます。

- 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

JBoss EAP 6 の起動後にサーバーログをチェックして、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てい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読み込みに失敗すると、HornetQ は NIO ジャーナルタイプに対して根本的な機能は保ちつつ失敗し、その点が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では、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。

バグの報告

3.2.9.2. 既存の JMS メッセージを JBoss EAP 6 に移行するような JMS ブリッジの設定

JBoss EAP 6 では、デフォルトの JMS 実装として JBoss Messaging が HornetQ に置き換えられました。JMS メッセージをある環境から別の環境に移行する最も簡単な方法は、JMS ブリッジを使用することです。JMS ブリッジの機能として、ソース JMS 宛先からメッセージを消費して、ターゲット JMS 宛先に送信することが挙げられます。JMS ブリッジは、JBoss EAP 5.x サーバーまたは JBoss EAP 6.1 以降のサーバーに設定およびデプロイできます。

JMS メッセージを JBoss EAP 5.x から JBoss EAP 6.x に移行する方法は、「JMS ブリッジの作成」を参照してください。

バグの報告

3.2.9.3. 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、通常は異なるサーバーにあるターゲット 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3.22 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 サーバーでのブリッジの設定

   リリース間のクラスで競合を回避するには、以下の手順にしたがって JBoss EAP 5.x で 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あり、必要に応じて変更できます。

   a. SAR を含む JBoss EAP 5 デプロイメントディレクトリーにサブディレクトリーを作成します（例： ***EAP5_HOME***/server/***PROFILE_NAME***/deploy/myBridge.sar/）。

   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/ ディレクトリーに **jboss-service.xml** ファイルを作成します。以下の例のような情報が含まれるはずです。

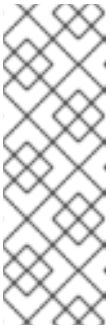
```
<server>
  <loader-repository>
    com.example:archive=unique-archive-name
    <loader-repository-config>java2ParentDelegation=false</loader-repository-
```

```
config>
  </loader-repository>

  <!-- JBoss EAP 6 JMS Provider -->
  <mbean code="org.jboss.jms.jndi.JMSProviderLoader"
name="jboss.messaging:service=JMSProviderLoader,name=EnterpriseApplication
Platform6JMSProvider">
    <attribute
name="ProviderName">EnterpriseApplicationPlatform6JMSProvider</attribute>
    <attribute
name="ProviderAdapterClass">org.jboss.jms.jndi.JNDIProviderAdapter</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.InitialContextFactory
       java.naming.provider.url=remote://EnterpriseApplicationPlatform6host: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=JMSProviderLoader,na
me=JMSProvider</depends>
    <depends optional-attribute-
name="TargetProviderLoader">jboss.messaging:service=JMSProviderLoader,nam
e=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>
```

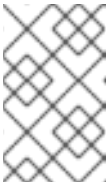


注記

SAR に分離されたクラスローダーが含まれるように、**load-repository** 要素が存在します。また、JNDI ルックアップとブリッジターゲットの両方に、パスワードが jbosspass のユーザー jbossuser のセキュリティークレデンシャルが含まれていることに注意してください。これは、JBoss EAP 6 はデフォルトでセキュアであるためです。パスワード「jbosspass」のユーザー「jbosspass」が、***EAP_HOME***/bin/add_user.sh スクリプトを使用して guest ロールで、ApplicationRealm に作成されました。

d. 以下の 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/ ディレクトリーにコピーします。各 *VERSION_NUMBER* は、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ます。

- 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、***EAP_HOME***/bin/client/jboss-client.jar は単純にコピーしないでください。

2. 宛先 JBoss EAP 6 サーバー上のブリッジの設定

JBoss EAP 6.1 以降では、JMS ブリッジを使用して JMS 1.1 準拠のサーバーからメッセージをブリッジできます。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またはメッセージブローカーの JNDI ルックアップクラスを JBoss モジュールにバンドルする必要があります。次の手順では、架空の MyCustomMQ メッセージブローカーを例として使用します。

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. 新しいモジュールの *EAP_HOME*/modules/system/layers/base/ の下にディレクトリー構造を作成します。main/ サブディレクトリーには、クライアント JAR および module.xml ファイルが含まれます。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ー構造の例:
EAP_HOME/modules/system/layers/base/org/mycustommq/main/

ii. main/ サブディレクトリーで、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 メッセージングプロバイダー用に作成された module.xml の例を以下に示します。

```
<?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 tothe 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/ サブディレクトリーにコピーします。MyCustomMQ モジュールのディレクトリー構造は以下のようになります。

```
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 サブシステムで JMS ブリッジを設定します。

i. 開始する前に、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ます。スタンドアロンサーバーを実行している場合は、これは

EAP_HOME/standalone/configuration/standalone-full-ha.xml ファイルです。管理対象ドメインを実行している場合は、*EAP_HOME*/domain/configuration/domain.xml ファイルと *EAP_HOME*/domain/configuration/host.xml ファイルの両方を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** サブシステムに追加します。**source** 要素および **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。ユーザー と パスワード の認証情報が指定されている場合は、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に設定された **jms-bridge** 要素の例を以下に示します。

```
<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 プロパティーは ソース の コンテキスト 要素で定義されています。上記の **target** の例のように、**context** 要素を省略すると、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。

バグの報告

3.2.9.4. HornetQ を JMS プロバイダーとして使用するためのアプリケーションの移行

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

前提条件

- クライアントとサーバーをシャットダウンしておく。

- JBoss Messaging データのバックアップコピーを作成しておく。メッセージデータは、**JBM_** で始まるテーブルのデータベースに保存されます。

手順3.23 プロバイダーを HornetQ に変更します。

1. 設定を転送します。

   既存の JBoss Messaging 設定を JBoss EAP 6 設定に転送します。以下の設定は、JBoss Messaging サーバー上のデプロイメント記述子にあります。

   - 接続ファクトリーサービスの設定

     この設定では、JBoss Messaging サーバーでデプロイされた JMS 接続ファクトリーを記述します。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に接続ファクトリーを設定します。

   - 宛先設定

     この設定では、JBoss Messaging サーバーでデプロイされた JMS キューとトピックについて記述します。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に宛先を設定します。

   - メッセージブリッジサービスの設定

     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は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 の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を変更します。

   アプリケーションコードで標準の JMS を使用する場合は、コードの変更は必要ありません。ただし、「移植可能な EJB JNDI 名」 セクションで説明されているように、新しい標準化された JNDI 名前空間を使用するため、アプリケーションを変更する必要があります。アプリケーションが JBoss Messaging 固有の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に、コードを変更する必要があります。

   以下は、**InitialContext** を作成し、JBoss EAP 6 でキューを検索する JMS クライアントの例です。

```
Hashtable<String, String> env = new Hashtable<String, String>();
String providerUrl = "remote:" + host + ":" + port;
env.put(Context.URL_PKG_PREFIXES, "org.jboss.ejb.client.naming");
env.put("java.naming.factory.initial",
"org.jboss.naming.remote.client.InitialContextFactory");
env.put("java.naming.provider.url", providerUrl);
env.put(Context.SECURITY_PRINCIPAL, user);
env.put(Context.SECURITY_CREDENTIALS, pass);
InitialContext context = new InitialContext(env);
Queue myTestQueue = (Queue) context.lookup("jms/queue/myTestQueue");
```

   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で HornetQ のドキュメントを慎重に確認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、HornetQ と JBoss Messaging はクラスタリング機能の実装ごとに大きく異なるアプローチをとっていました。

HornetQ でメッセージングを設定する方法は、以下を参照してください。 「HornetQ でのメッセージングの設定」

3. 既存のメッセージを移行します。

   JMS ブリッジを使用して、JBoss Messaging データベースのメッセージを 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 に移行するような JMS ブリッジの設定」 を参照してください。

リモートクライアントから JMS キューにアクセスする方法については、カスタマーポータル https://access.redhat.com/documentation/ja-jp/red_hat_jboss_enterprise_application_platform/?version=6.4 の JBoss Enterprise Application Platform 6.4 の 『『開発ガイド』』 の「リモート『JNDI ルックアップ』」の章を参照してください。

JBoss EAP 6 サーバーのインスタンスから別のインスタンスにメッセージを送信する方法は、カスタマーポータルの JBoss Enterprise Application Platform 6.4 の 『『Administration and Configuration Guide』』 の「『Configuring the HornetQ JCA Adapter for Remote Connection』」を参照 https://access.redhat.com/documentation/ja-jp/red_hat_jboss_enterprise_application_platform/?version=6.4 してください。

バグの報告

3.2.9.5. HornetQ でのメッセージングの設定

JBoss EAP 6 でメッセージングを設定する方法として、管理コンソールまたは管理 CLI が推奨されます。**standalone.xml** または **domain.xml** 設定ファイルを手動で編集しなくても、これらの管理ツールのいずれかで永続的な変更を行うことができます。ただし、デフォルトの設定ファイルのメッセージングコンポーネントを理解すると便利です。ここでは、管理ツールを使用したドキュメントの例では、参照用の設定ファイルスニペットが提供されます。

バグの報告

3.2.9.6. JMS 宛先の移行

以前のリリースの JBoss EAP では、JBoss Messaging JMS 宛先は **jbossmq-destinations-service.xml** または destination-service.**xml** ファイルの **<mbeans>** 要素に設定されていました。JBoss EAP 6 では JBoss Messaging が HornetQ に置き換えられたため、JMS 宛先はサーバー設定ファイルの **messaging** サブシステムに設定されるようになりました。

> **例3.3 JBoss EAP 4.2 宛先設定の例**
>
> 以下は、JBoss EAP 4.2 の **jbossmq-destinations-service.xml** ファイルに設定された JBoss MQ 宛先の例になります。**JNDIName** 属性が指定されていない場合、値はデフォルトで例に示されている名前に設定されます。
>
> ```
> <mbean code="org.jboss.mq.server.jmx.Queue"
>   name="jboss.mq.destination:service=Queue,name=DLQ">
>   <!-- The following JNDIName attribute shows the default JNDI binding name  -->
>   <attribute name="JNDIName">queue/DLQ</attribute>
>   <depends optional-attribute-name="DestinationManager">
>     jboss.mq:service=DestinationManager
>   </depends>
> </mbean>
> <mbean code="org.jboss.mq.server.jmx.Queue"
>   name="jboss.mq.destination:service=Queue,name=ExpiryQueue">
>   <!-- The following JNDIName attribute shows the default JNDI binding name  -->
> ```

```
  <attribute name="JNDIName">queue/ExpiryQueue</attribute>
  <depends optional-attribute-name="DestinationManager">
    jboss.mq:service=DestinationManager
  </depends>
</mbean>
```

例3.4 JBoss EAP 5 宛先設定の例

以下は、JBoss EAP 5 の destination -service.xml ファイルに設定された JBoss Messaging 宛先の例になります。JNDIName attribute が指定されていない場合、値はデフォルトで例に示されている名前に設定されます。

```
<mbean code="org.jboss.jms.server.destination.QueueService"
  name="jboss.messaging.destination:service=Queue,name=DLQ"
  xmbean-dd="xmdesc/Queue-xmbean.xml">
  <!-- The following JNDIName attribute shows the default JNDI binding name  -->
  <attribute name="JNDIName">queue/DLQ</attribute>
  <depends optional-attribute-name="ServerPeer">
    jboss.messaging:service=ServerPeer
  </depends>
  <depends>
    jboss.messaging:service=PostOffice
  </depends>
</mbean>
<mbean code="org.jboss.jms.server.destination.QueueService"
  name="jboss.messaging.destination:service=Queue,name=ExpiryQueue"
  xmbean-dd="xmdesc/Queue-xmbean.xml">
  <!-- The following JNDIName attribute shows the default JNDI binding name  -->
  <attribute name="JNDIName">queue/ExpiryQueue</attribute>
  <depends optional-attribute-name="ServerPeer">
    jboss.messaging:service=ServerPeer
  </depends>
  <depends>
    jboss.messaging:service=PostOffice
  </depends>
</mbean>
```

例3.5 JBoss EAP 6 宛先設定の例

以下は、JBoss EAP 6 のサーバー設定ファイルの messaging サブシステムで設定された JMS 宛先の例になります。JBoss EAP 6 では、entry 要素はキューを JNDI にバインドするために使用される名前を設定します。

```
<subsystem xmlns="urn:jboss:domain:messaging:1.4">
<hornetq-server>
  ...
    <jms-destinations>
      <jms-queue name="ExpiryQueue">
        <entry name="java:/jms/queue/ExpiryQueue"/>
      </jms-queue>
      <jms-queue name="DLQ">
        <entry name="java:/jms/queue/DLQ"/>
```

```
      </jms-queue>
    </jms-destinations>
  </hornetq-server>
</subsystem>
```

宛先の設定方法に関する詳細は、カスタマーポータルの JBoss Enterprise Application Platform の『『Administration and Configuration Guide』』の「『Configure the JMS Server』」を参照してください。 https://access.redhat.com/documentation/ja-jp/red_hat_jboss_enterprise_application_platform/?version=6.4

バグの報告

3.2.10. クラスタリングの変更

3.2.10.1. クラスタリング向けにアプリケーションに変更を加える。

1. **クラスタリングが有効な状態で JBoss EAP 6 を起動**
   JBoss EAP 5.x でクラスタリングを有効にするには、all プロファイルまたはその派生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す。

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

   JBoss EAP 6 では、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か、管理対象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稼働しているサーバーのクラスター化の有効化**
      ドメインコントローラーを使用して起動されたサーバーのクラスター化を有効にするには、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ようにサーバーグループを指定します。以下に例を示します。

      ```
      <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のクラスター化の有効化**
      スタンドアロンサーバーでクラスター化を有効にするには、以下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します。**
   JBoss EAP 5.x では、通常、以下のように -b コマンドライン引数を使用してクラスタリングに使用されるバインドアドレスを指定します。

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

JBoss EAP 6 は、**standalone.xml** ファイル、**domain.xml** ファイル、および **host.xml** ファイルの **<interfaces>** 要素に含まれる IP アドレスとインターフェースにソケットをバインドします。JBoss EAP に同梱される標準設定には、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>
```

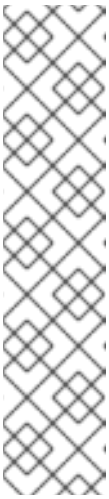
これらのインターフェース設定では、システムプロパティー **jboss.bind.address.management** および **jboss.bind.address** の値を使用します。これらのシステムプロパティーが設定されていない場合は、デフォルトの **127.0.0.1** が各値に使用されます。

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したり、JBoss EAP 6 サーバー設定ファイル内で明示的に定義したりできます。

- 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 bind 引数を指定します。

  以下は、スタンドアロンサーバーのコマンドラインでバインドアドレスを指定する方法の例になります。

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

  注記

  -b のショートカットである **-Djboss.bind.address=127.0.0.1** 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 構文の形式は引き続きサポートされます。

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

  -b 引数は、パブリック 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。管理 インターフェースには影響しません。

- 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。

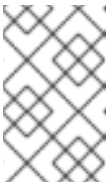
  管理対象ドメインで実行されているサーバーの場合は、**domain/configuration/host.xml** ファイルにバインドアドレスを指定します。スタンドアロンサーバーの場合は、**standalone-ha.xml** ファイルにバインドアドレスを指定します。

  以下の例では、**public** インターフェースは **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のデフォルトインターフェースとして指定されます。

  ```
  <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

設定ファイルのシステムプロパティーではなく、ハードコーディングされた値としてバインドアドレスを指定した場合、コマンドラインの引数で上書きすることはできません。

3. **mod_jk および mod_proxy をサポートする jvmRoute の設定**
   JBoss EAP 5 では、Web サーバー jvmRoute は server.xml ファイルのプロパティーを使用して設定されます。JBoss EAP 6 では、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 サブシステムに jvmRoute 属性が設定されます。

```
<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 に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の指定**
   JBoss EAP 5.x では、以下のようにコマンドライン引数 -u および -m を使用して、クラスター内通信に使用されるマルチキャストアドレスおよびポートを指定できます。

```
$ 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、以下のように関連する JGroups プロトコルスタックによって参照されるソケットバインディングによって定義されます。

```
<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と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、サーバーの起動時にコマンドラインでこれらのプロパティーを使用できます。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、 **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}"/>
```

次に、以下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. **別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ーを使用して、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す。

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

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

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

6. **バウンドレプリケーションの置き換え**
   JBoss EAP 5.x では JBoss Cache Buddy Replication 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のレプリケーションが抑制されました。

   JBoss EAP 6 では、Buddy Replication は Infinispan の分散キャッシュ（ **DIST** モードとも呼ばれる）に置き換えられました。Distribution（分散）は強力なクラスタリングモードで、より多くのサーバーがクラスターに追加されるため、Infinispan は直線的にスケーリングできます。以下は、DIST キャッシュモードを使用するようにサーバーを設定する方法の例になります。

   a. コマンドラインを開き、HA または Full Profile のいずれかでサーバーを起動します。以下に例を示します。

      ```
      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 に接続します。

      - 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てキャッシュを設定し、新しい設定をリロードするようにサーバーに指示を出します。

```
/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、以下のように standalone-ha.xml ファイルの infinispan サブシステムの dist <distributed-cache> 要素を変更します。 <cache-container>

```
<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、カスタマーポータル https://access.redhat.com/documentation/ja-jp/red_hat_jboss_enterprise_application_platform/?version=6.4 の JBoss EAP 6 の『『開発ガイド』』 の「 『Web アプリケーションでのクラスタリング』 」の章を参照してください。

バグの報告

3.2.10.2. 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プされ、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をデプロイする方法を示しています。サービスはスケジュールされたタ

イマーを有効にします。これはクラスター内で 1 回のみ起動します。

手順3.24 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記述します。

以下は、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プされるサービスの簡単な例です。完全な例は、Red Hat JBoss Enterprise Application Platform 6 に同梱される 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すべての手順が含まれます。

a. サービスを作成します。

サービスの例を以下に示します。

```
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.hasingleton.service.ejb.Sc
heduler")).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.hasingleton.service.ejb.Sc
heduler")).stop();
            } catch (NamingException e) {
                LOGGER.error("Could not stop timer", e);
            }
        }
    }
}
```

b.

サービス 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 activator を作成します。

以下のリストは、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 Service Activator の例です。

```
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("node1/singleton")));

        singleton.build(new DelegatingServiceContainer(context.getServiceTarget(),
context.getServiceRegistry()))
                .setInitialMode(ServiceController.Mode.ACTIVE)
                .install()
        ;
    }
}
```

注記

上記の例は、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 JBoss EAP 7 リリースで利用でき、プライベートクラスは非推奨になりましたが、これらのクラスは JBoss EAP 6.x リリースサイクルの間維持および利用可能になります。

c.

ServiceActivator ファイルの作成

アプリケーションの **resources/META-INF/services/** ディレクトリーに **org.jboss.msc.service.ServiceActivator** という名前のファイルを作成します。直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.

クラスター全体のシングルトンタイマーとして使用するタイマーを実装するシングルトン **Bean** を作成します。

このシングルトン **Bean** はリモートインターフェースを持たないため、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でください。これにより、クライアントまたは他のコンポーネントによるルックアップが阻止され、**SingletonService** がシングルトンを完全に制御できるようになります。

i.

Scheduler インターフェースの作成

```
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シングルトン 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. **クラスタリングを有効にして各 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でクラスター化を有効にするには、各インスタンスの一意のノード名とポートオフセットを使用して、HA プロファイルで各サーバーを起動する必要があります。

- **Linux の場合は、以下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***

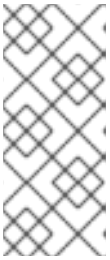
> **例3.6 Linux で複数のスタンドアロンサーバーを起動する**
>
> ```
> $ EAP_HOME/bin/standalone.sh --server-config=standalone-ha.xml -Djboss.node.name=node1
> $ EAP_HOME/bin/standalone.sh --server-config=standalone-ha.xml -Djboss.node.name=node2 -Djboss.socket.binding.port-offset=100
> ```

- **Microsoft Windows の場合は、以下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3.7 Microsoft Windows で複数のスタンドアロンサーバーを起動する

```
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ない場合は、各サーバーインスタンスの **standalone-ha.xml** ファイルを設定して、別のインターフェースにバインドすることができます。

3. **サーバーのアプリケーションのデプロイ**

以下の **Maven** コマンドは、デフォルトポートで実行しているスタンドアロンサーバーにアプリケーションをデプロイします。

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

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

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

Maven の設定やデプロイメントの詳細については、JBoss EAP 6 に同梱される cluster-ha-singleton クイックスタートを参照してください。

バグの報告

3.2.11. サービススタイルのデプロイメントの変更

3.2.11.1. サービススタイルの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

MBean は、以前のバージョンの Red Hat JBoss Enterprise Application Platform のコアアーキテクチャーの一部でした。JBoss 固有の jboss-service.xml および jboss-beans.xml サービススタイル記述子を使用する JBoss Service Archive(SAR)デプロイメントは、JBoss Beans に基づいて MBean を作成するためにアプリケーションサーバーによって使用されるものです。内部アーキテクチャーは JBoss EAP 6 で変更になり、MBean JMX アーキテクチャーを基にしなくなります。MBean はコアアーキテクチャーの一部ではなくなりました。管理 API のラッパーになりました。

アプリケーションがサービススタイルのデプロイメント記述子を使用する場合、アプリケーションが定義された MBean のみに依存し、JBoss Management API MBean ラッパーに依存しない限り、JBoss EAP 6 で引き続き動作します。JBoss EAP 6 では、SAR は別の SAR デプロイメントによって作成された MBean に対する MBean 依存関係のみを宣言できます。つまり、アプリケーションが作成した MBean（EJB またはメッセージングコンポーネントの MBean など）に依存していると動作しません。依存できる MBean は他の jboss-service.xml ファイルで定義された他の MBean のみです。

サービススタイルのデプロイメントを @Singleton に置き換えます。

以前のバージョンの JBoss EAP で使用されていた JBoss Service Archive(SAR)およびサービススタイル記述子は Java EE 6 仕様の一部ではないため、JBoss EAP 6 で使用することは推奨されません。アプリケーションを Java EE 6 仕様に変更することが推奨されます。MBean シングルトンの場合は、JBoss EAP プロプライエタリーの *-beans.xml ファイルおよび *-service.xml ファイルの代わりに移植可能な Java EE 6 @Singleton を使用するようにコードを変更する必要があります。MBean サービスの作成およびデプロイに関する詳細は、カスタマーポータル https://access.redhat.com/documentation/ja-jp/red_hat_jboss_enterprise_application_platform/?version=6.4 の 『JBoss EAP 6 の 『『開発ガイド』』 の「JBoss MBean」 サービス」の章を参照してください。

サービススタイルの記述子の変更

標準の Java EE 6 EJB @Singleton を使用するためにコードを変更しない場合、JBoss EAP 5 と JBoss EAP 6 の間で jboss-beans.xml ファイルおよび jboss-service.xml ファイルが変更されたこと

に注意してください。

jboss-beans.xml ファイルの XML 宣言は JBoss EAP 5 と JBoss EAP 6 との間で変更になりました。

以下は、JBoss EAP 5 の jboss-beans.xml ファイルの例になります。

```
<?xml version="1.0" encoding="UTF-8"?>
<deployment xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="urn:jboss:bean-deployer:2.0 bean-deployer_2_0.xsd"
    xmlns="urn:jboss:bean-deployer:2.0">
  <bean name="TestServantBean"
class="org.jboss.test.iiop.jbpapp6462.servant.TestServantBean"/>
</deployment>
```

以下は、JBoss EAP 6 の jboss-beans.xml ファイルの例になります。

```
<deployment xmlns="urn:jboss:pojo:7.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="urn:jboss:pojo:7.0 jboss-mc_7_0.xsd">
...
  <bean name="Bean">
    <alias>Alias-Bean</alias>
    <constructor factory-class="org.jboss.as.test.integration.pojo.support.TFactory" factory-
method="createBean">
      <parameter><value>test</value></parameter>
    </constructor>
    <property name="injectee"><inject bean="Injectee"/></property>
    <start>
      <parameter><value>start</value></parameter>
    </start>
    <stop>
      <parameter><value>stop</value></parameter>
    </stop>
    <install method="install" state="CREATE">
      <parameter><value>install</value></parameter>
    </install>
    <install bean="Alias-Injectee" method="sayHello">
      <parameter><value>from-other-bean</value></parameter>
    </install>
  </bean>
</deployment>
```

JBoss EAP 5 と JBoss EAP 6 の間で jboss-service.xml ファイルの DTD が変更になりました。

以下は、JBoss EAP 5 の jboss-service.xml ファイルの例になります。

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<!DOCTYPE server
   PUBLIC "-//JBoss//DTD MBean Service 5.0//EN"
   "http://www.jboss.org/j2ee/dtd/jboss-service_5_0.dtd">
<server>
...
</server>
```

以下は、JBoss EAP 6 の jboss-service.xml ファイルの例になります。

```
<server xmlns="urn:jboss:service:7.0"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="urn:jboss:service:7.0 jboss-service_7_0.xsd">
   <mbean name="jboss.test:service=testdeployments"
code="org.jboss.as.test.integration.mgmt.access.deployment.sar.Simple"/>

</server>
```

バグの報告

3.2.12. リモート呼び出しの変更

3.2.12.1. リモート呼び出しを JBoss EAP 6 に作成する JBoss EAP 5 のデプロイされた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 JNDI でバインドされ、デフォルトではローカルインターフェースの *EJB_NAME*/local でバインドされ、リモートインターフェースの *EJB_NAME*/remote がバインドされました。その後、クライアントアプリケーションは、*EJB_NAME*/remote を使用して Bean を検索しました。

JBoss EAP 6 では、リモート呼び出しに新しい EJB クライアント API が導入されました。ただし、新しい API を使用するためにコードを書き直さない場合は、以下の構文を使用して EJB へのリモートアクセスに ejb:*BEAN_REFERENCE* を使用するよう既存のコードを変更できます。

ステートレス Bean の場合、ejb:*BEAN_REFERENCE* 構文は以下のようになります。

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

ステートフル Bean の場合、ejb:*BEAN_REFERENCE* 構文は以下のようになります。

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

上記の構文で置換され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、これは .ear 接尾辞のない ear 名ですが、名前は application.xml ファイルで上書きできます。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になります。この例では EAR としてデプロイされていないことを仮定します。
- <module-name> - サーバーにデプロイされた EJB のモジュール名。これは通常、.jar サフィックスのない EJB デプロイメントの jar 名ですが、ejb-jar.xml を使用して上書きできます。この例では、EJB が jboss-ejb-remote-app.jar にデプロイされたことを仮定するため、モジュール名は jboss-ejb-remote-app になります。
- <distinct-name> - EJB の任意の一意の名前。この例では、別の名前を使用しないため、空の文字列を使用します。
- <bean-name> - デフォルトでは、Bean 実装クラスのシンプルなクラス名です。
- <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

以下のクライアントコードの例では、以下のステートレス EJB を JBoss EAP 6 サーバーにデプロイしていることを前提としています。@Remote アノテーションを使用して Bean のリモートビューを公開することに注意してください。

例3.8 ステートレスセッション Bean コードの例

```
@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 クライアントコードでは、JNDI 名を使用して InitialContext を作成し、EJB を検索しました。

例3.9 JBoss EAP 5 クライアントの例

```
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、上記のように Hashtable を作成し、Bean 参照のプロパティーを設定します。以下は、クライアントルックアップおよび呼び出しの例です。

例3.10 JBoss EAP 6 のステートレスクライアントの例

```
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は、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。

例3.11 JBoss EAP 6 のステートフルクライアントの例

```
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の両方を含む完全な作業例は、JBoss EAP 6 に同梱されるクイックスタートで確認できます。詳細は、JBoss EAP 6 『『開発ガイド』』 の「 『クイック 『スタートチュートリアルの確認』 』 」の章を参照して https://access.redhat.com/documentation/ja-jp/red_hat_jboss_enterprise_application_platform/?version=6.4 ください。

JNDI を使用したリモート呼び出しの詳細は、「JNDI を使用したリモートによるセッション Bean の呼び出し」 を参照してください。

バグの報告

3.2.12.2. JNDI を使用したリモートによるセッション Bean の呼び出し

このタスクでは、JNDI を使用してセッション Bean の呼び出しに対してリモートクライアントへのサポートを追加する方法を説明します。このタスクは、プロジェクトが Maven を使用してビルドされていることを前提としています。

ejb-remote クイックスタートには、この機能を実証する作業用の Maven プロジェクトが含まれています。クイックスタートには、デプロイするセッション Bean とリモートクライアントの両方用のプロジェクトが含まれます。以下のコードサンプルは、リモートクライアントプロジェクトから取得されます。

このタスクは、セッション Bean が認証を必要としないことを前提としています。

警告

Red Hat は、影響するすべてのパッケージで TLSv1.1 または TLSv1.2 を利用するために SSL を明示的に無効化することを推奨しています。

前提条件

開始する前に、以下の前提条件を満たす必要があります。

- **使用できる Maven プロジェクトがすでに作成されている必要があります。**
- **JBoss EAP 6 Maven リポジトリーの設定はすでに追加されています。**
- **呼び出すセッション Bean は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 は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 依存関係として利用できます。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利用できる場合は、JAR を Maven リポジトリーにアーティファクトとして追加することが推奨されます。方向については、install:install-file ゴールの Maven ドキュメントを参照してください。 http://maven.apache.org/plugins/maven-install-plugin/usage.html**
- **セッション Bean をホストするサーバーのホスト名および JNDI ポートを知っている必要があります。**

リモートクライアントからセッション Bean を呼び出すには、まずプロジェクトを正しく設定する必要があります。

手順3.25 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**

プロジェクトの pom.xml を更新して、必要な依存関係が含まれるように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でファイルを検索することを想定します。以下の内容を含む、このファイルをプロジェクトの src/main/resources/ ディレクトリーに追加します。

```
# 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
```

サーバーに合わせてホスト名とポートを変更します。4447 はデフォルトのポート番号です。セキュアな接続の場合は、SSL_ENABLED 行を true に設定し、SSL_STARTTLS 行のコメントを解除します。コンテナーの Remoting インターフェースは、同じポートを使用したセキュアな接続およびセキュア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の pom.xml に Maven 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>
```

プロジェクトが正しく設定されたので、コードを追加してセッション Bean にアクセスし、呼び出すことができます。

手順3.26 JNDI と Bean の Invoke メソッドを使用した Bean プロキシーの取得

1. チェックされた例外の処理

   以下のコード（InitialContext() および lookup()）で使用される 2 つのメソッドには、javax.naming.NamingException 型のチェック済み例外があります。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、または NamingException のスローに宣言されているメソッドに囲む必要があります。ejb-remote クイックスタートは 2 番目の手法を使用します。

2. JNDI コンテキストの作成

   JNDI Context オブジェクトは、サーバーからリソースを要求するメカニズムを提供します。以下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ーは jboss-ejb-client.properties ファイルから読み込まれます。

3. JNDI コンテキストの lookup（）メソッドを使用して Bean プロキシーを取得します。

   Bean プロキシーの lookup() メソッドを呼び出し、必要なセッション Bean の JNDI 名を渡します。これにより、呼び出す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にキャストする必要のある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、特別な構文を使用して定義されます。詳細は、「EJB JNDI 命名リファレンス」 を参照してください。

4. メソッドを呼び出す

プロキシー Bean オブジェクトがあると、リモートビジネスインターフェースに含まれるメソッドを呼び出すことができます。

```
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);
```

プロキシー Bean はメソッド呼び出しリクエストをサーバーのセッション Bean に渡します。結果はプロキシー Bean に返され、呼び出し元に返されます。プロキシー Bean とリモートセッション Bean 間の通信は、呼び出し元に対して透過的です。

これで、リモートサーバーでセッション Bean を呼び出し、JNDI を使用してサーバーから取得したプロキシー Bean を使用してコードを書き込むように Maven プロジェクトを設定できるようになりました。

バグの報告

3.2.12.3. EJB JNDI 命名リファレンス

セッション Bean の JNDI ルックアップ名の構文は以下のとおりです。

```
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、これはその EAR の名前になります。デフォルトでは、EAR の名前は .ear 接尾辞を含まないファイル名です。アプリケーション名は application.xml ファイルで上書きすることもできます。セッション Bean が EAR にデプロイされていない場合は、これを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 がデプロイされる JAR ファイルの名前です。デフォルトでは、JAR ファイルの名前は、.jar 接尾辞を含まないファイル名です。モジュール名は JAR の ejb-jar.xml ファイルで上書きすることもできます。

<distinctName>

JBoss EAP 6 では、各デプロイメントで任意の一意の名前を指定できます。デプロイメントに一意の名前がない場合は、これを空白のままにします。

<beanName>

Bean 名は、呼び出されるセッション Bean のクラス名です。

<viewClassName>

view 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。これには、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 を参照する場合は、?stateful サフィックスが必要です。他の Bean タイプには含まれません。

バグの報告

3.2.12.4. EJB 非同期メソッド呼び出しの移行

概要

EJB 3.1 仕様では、非同期メソッド呼び出しを行う機能が導入されました。この機能の導入前に、一部のアプリケーションサーバーはプロプライエタリーの非同期機能を提供していました。JBoss EAP 4.x は org.jboss.ejb3.asynchronous パッケージでクラスを提供し、JBoss EAP 5.x はこの目的のために org.jboss.ejb3.common.proxy.plugins.async.AsyncUtils クラスを提供していました。これらのクラスは JBoss EAP 6 ではサポートされず、置き換える必要があります。以下の例は、JBoss EAP 6 で非同期メソッド呼び出しを行う方法を示しています。

手順3.27 非同期セッション Bean とクライアントの作成

1. 非同期セッション Bean メソッドを作成します。

以下の例では、sayHelloAsync() メソッドは @Asynchronous アノテーションを使用して非同期としてマークされます。型 String で、必要な Future クラス実装を返します。

```
import javax.ejb.Remote;
import javax.ejb.SessionContext;
import javax.ejb.Stateless;
import javax.ejb.TransactionAttribute;
import javax.ejb.TransactionAttributeType;
import javax.interceptor.Interceptors;

import java.util.concurrent.Future;
import javax.ejb.AsyncResult;
import javax.ejb.Asynchronous;

@Stateless(name="Hello")
@Remote(Hello.class)
public class HelloBean implements Hello {

   @Asynchronous
   public Future<String> sayHelloAsync() {
     return new AsyncResult<String>("Hello");
   }
}
```

2. クライアントコードを作成します。

利用可能な Future.get（） メソッドの 1 つを使用して EJB を検索し、非同期メソッドを呼び出します。以下の例では、待機時間を timeout 引数および unit 引数で指定された値に制限します。

```
Future<String> future = ejbObject.sayHelloAsync();
String response = future.get(timeout, unit);
```

バグの報告

3.2.13. 「EJB Changes」

3.2.13.1. ステートフル EJB キャッシュ設定の移行

JBoss EAP 5 では、ステートフル EJB キャッシュは jboss.xml ファイルの container-cache-conf

要素を使用して設定されます。以下の例は、JBoss EAP 5 におけるステートフルキャッシュ設定を示しています。

```
<container-cache-conf>
  <cache-policy>org.jboss.ejb.plugins.LRUStatefulContextCachePolicy</cache-policy>
  <cache-policy-conf>
    <min-capacity>50</min-capacity>
    <max-capacity>200</max-capacity>
    <remover-period>1800</remover-period>
    <max-bean-life>1320</max-bean-life>
    <overager-period>300</overager-period>
    <max-bean-age>1260</max-bean-age>
  </cache-policy-conf>
 </container-cache-conf>
```

JBoss EAP 6 では、ステートフル EJB キャッシュはサーバー設定ファイルの ejb3 サブシステムに設定されます。詳細な手順は、「ステートフルセッション Bean キャッシュの設定」 を参照してください。この手順では、JBoss EAP 5 で指定された stateful-timeout に置き換わる <max-bean-age> の設定方法も説明します。

バグの報告

3.2.13.2. ステートフルセッション Bean キャッシュの設定

JBoss EAP 6 では、ステートフル EJB キャッシュはサーバー設定ファイルの ejb3 サブシステムに設定されます。以下の手順では、ステートフル EJB キャッシュおよびステートフルタイムアウトを設定する方法を説明します。

手順3.28 ステートフル EJB キャッシュの設定

1. サーバー設定ファイルの ejb3 サブシステムで <caches> 要素を見つけます。<cache> 要素を追加します。以下の例では、「my=cache」という名前のキャッシュを作成します。

```
<cache name="my-cache" passivation-store-ref="my-cache-file" aliases="my-custom-cache"/>
```

2. サーバー設定ファイルの **ejb3** サブシステムで **<passivation-stores>** 要素を見つけます。前のステップで定義したキャッシュ用に **<file-passivation-store>** を作成します。

```
<file-passivation-store name="my-cache-file" idle-timeout="1260" idle-timeout-
unit="SECONDS" max-size="200"/>
```

3. **ejb3** サブシステムの設定が以下の例のようになるはずです。

```
<subsystem xmlns="urn:jboss:domain:ejb3:1.4">
  ...
  <caches>
    <cache name="simple" aliases="NoPassivationCache"/>
    <cache name="passivating" passivation-store-ref="file" aliases="SimpleStatefulCache"/>
    <cache name="clustered" passivation-store-ref="infinispan" aliases="StatefulTreeCache"/>
    <cache name="my-cache" passivation-store-ref="my-cache-file" aliases="my-custom-
cache"/>
  </caches>
  <passivation-stores>
    <file-passivation-store name="file" idle-timeout="120" idle-timeout-unit="SECONDS" max-
size="500"/>
    <cluster-passivation-store name="infinispan" cache-container="ejb"/>
    <file-passivation-store name="my-cache-file" idle-timeout="1260" idle-timeout-
unit="SECONDS" max-size="200"/>
  </passivation-stores>
  ...
</subsystem>
```

パッシベーションキャッシュ **"my-cache"** は「**my-cache-file**」パッシベーションストアに設定されたようにステートフルセッション **Bean** をファイルシステムに渡します。これには、**idle-timeout**、**idle-timeout-unit**、および **max-size** オプションが含まれます。

4. **EJB JAR** の **META -INF/** ディレクトリーに **jboss-ejb3.xml** ファイルを作成します。以下の例は、前のステップで定義されたキャッシュを使用するように **EJB** を設定します。

```
<jboss:ejb-jar xmlns:jboss="http://www.jboss.com/xml/ns/javaee"
      xmlns="http://java.sun.com/xml/ns/javaee"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:c="urn:ejb-cache:1.0"
      xsi:schemaLocation="http://www.jboss.com/xml/ns/javaee
http://www.jboss.org/j2ee/schema/jboss-ejb3-2_0.xsd http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/ejb-jar_3_1.xsd"
      version="3.1"
      impl-version="2.0">
  <assembly-descriptor>
    <c:cache>
      <ejb-name>*</ejb-name>
      <c:cache-ref>my-cache</c:cache-ref>
```

```
    </c:cache>
  </assembly-descriptor>
</jboss:ejb-jar>
```

5. タイムアウト値を設定する方法は、EJB 2 または EJB 3 を実装するかどうかによって異なります。

   - EJB 3 ではアノテーションが導入されるため、以下のように EJB コードで javax.ejb.StatefulTimeout アノテーションを指定できます。

```
@StatefulTimeout(value = 1320, unit=java.util.concurrent.TimeUnit.SECONDS)
@Stateful
@Remote(MyStatefulEJBRemote.class)
public class MyStatefulEJB implements MyStatefulEJBRemote {
  ...
}
```

     @StatefulTimeout の値は以下のいずれかに設定できます。

     - 値が 0 の場合は、Bean がすぐに削除できます。
     - 0 より大きい値は、unit パラメーターで指定された単位のタイムアウト値を示します。デフォルトのタイムアウト単位は MINUTES です。パッシベーションキャッシュ設定を使用し、idle-timeout の値が StatefulTimeout 値よりも小さい場合、指定された idle-timeout 期間に対してアイドル状態になると、JBoss EAP は Bean をパッシベートします。その後、Bean は StatefulTimeout の期間後に削除することができます。
     - -1 を値と し て指定すると、タイムアウトにより bean が削除されません。パッシベーションキャッシュ設定を使用し、Bean が idle-timeout でアイドル状態である場合、JBoss EAP は Bean インスタンスを passivation-store に渡します。
     - -1 未満の値は有効ではありません。

- EJB 2 と EJB 3 の両方に対して、ejb-jar.xml ファイルでステートフルタイムアウトを設定できます。

```
<?xml version="1.0" encoding="UTF-8"?>
<ejb-jar xmlns="http://java.sun.com/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/ejb-jar_3_1.xsd"
         version="3.1">
  <enterprise-beans>
    <session>
      <ejb-name>HelloBean</ejb-name>
        <session-type>Stateful</session-type>
        <stateful-timeout>
          <timeout>1320</timeout>
          <unit>Seconds</unit>
        </stateful-timeout>
    </session>
  </enterprise-beans>
</ejb-jar>
```

- EJB 2 と EJB 3 の両方に対して、jboss-ejb3.xml ファイルでステートフルタイムアウトを 設定できます。

```
<jboss:ejb-jar xmlns:jboss="http://www.jboss.com/xml/ns/javaee"
      xmlns="http://java.sun.com/xml/ns/javaee"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:c="urn:ejb-cache:1.0"
      xsi:schemaLocation="http://www.jboss.com/xml/ns/javaee
http://www.jboss.org/j2ee/schema/jboss-ejb3-2_0.xsd http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/ejb-jar_3_1.xsd"
      version="3.1"
      impl-version="2.0">
  <enterprise-beans>
    <session>
      <ejb-name>HelloBean</ejb-name>
        <session-type>Stateful</session-type>
        <stateful-timeout>
          <timeout>1320</timeout>
          <unit>Seconds</unit>
```

```
        </stateful-timeout>
      </session>
    </enterprise-beans>
    <assembly-descriptor>
      <c:cache>
        <ejb-name>*</ejb-name>
        <c:cache-ref>my-cache</c:cache-ref>
      </c:cache>
    </assembly-descriptor>
  </jboss:ejb-jar>
```

追加情報

- ステートフルセッション Bean のパッシベーションを無効にするには、以下のいずれかを行います。
  - EJB 3 アノテーションを使用してステートフルセッション Bean を実装した場合、アノテーションのステートフルセッション Bean のパッシベーションを無効にできます。`@org.jboss.ejb3.annotation.Cache("NoPassivationCache")`
  - ステートフルセッション Bean が jboss-ejb3.xml ファイルで設定されている場合、<c:cache-ref> 要素の値を NoPassivationCache と同等である「simple」に設定します。

    ```
    <c:cache-ref>simple</c:cache-ref>
    ```

- JBoss EAP 6 では EJB キャッシュポリシー「LRUStatefulContextCachePolicy」が変更になったため、JBoss EAP 6 では 1 対 1 の設定マッピングを行うことができません。

- JBoss EAP 6 では、以下のキャッシュプロパティーを設定できます。
  - Bean のライフサイクル時間は、EJB 3.1 の @StatefulTimeout を使用して設定されます。
  - <file-passivation-store> 要素の idle-timeout 属性を使用して、サーバー設定ファイルの ejb3 サブシステムのディスクに Bean のパッシベーションを設定します。
  - <file-passivation-store> 要素の max-size 属性を使用して、サーバー設定ファイルの ejb3 サブシステムでパッシベーションストアの最大サイズを設定します。

- JBoss EAP 6 では、以下のキャッシュプロパティーを設定することはできません。
  - メモリーキャッシュの最小および最大数。
  - パッシベーションストアの最小番号。
  - キャッシュ操作の頻度を制御する *-period 設定。

バグの報告

3.2.13.3. ステートレスセッション Bean プールサイズの設定

概要

JBoss EAP 5 では、ステートレス EJB プールのデフォルトは

EAP_HOME/server/*PROFILE*/deploy/ejb3-interceptors-aop.xml ファイルで設定されていました。以下の例は、JBoss EAP 5 のプール設定を示しています。

```
<domain name="Stateless Bean" extends="Intercepted Bean" inheritBindings="true">
  ...
  <annotation expr="class(*) AND !class(@org.jboss.ejb3.annotation.Pool)">
   @org.jboss.ejb3.annotation.Pool (value="ThreadlocalPool", maxSize=30, timeout=10000)
  </annotation>
</domain>
```

このトピックでは、JBoss EAP 6 で EJB セッション bean のステートレスプールを設定する方法について説明します。

JBoss EAP 6 でのデフォルトのステートレスセッション Bean プールサイズの設定

JBoss EAP 6 では、ステートレスセッション Bean プールはサーバー設定ファイルの ejb3 サブ要素の <bean-instance-pools> セクションで設定されます。ステートレスセッション Bean のデフォルトプールは "slsb-strict-max-pool" です。<bean-instance-pools> で追加の設定を定義できます。以下の例は、JBoss EAP 6 で「my-stateless-bean-pool」という名前の新しい設定を定義します。

```
<subsystem xmlns="urn:jboss:domain:ejb3:1.3">
  <session-bean>
    <stateless>
      <bean-instance-pool-ref pool-name="slsb-strict-max-pool"/>
    </stateless>
    <stateful default-access-timeout="5000" cache-ref="simple"/>
      <singleton default-access-timeout="5000"/>
    </session-bean>
    <pools>
      <bean-instance-pools>
        <strict-max-pool name="slsb-strict-max-pool" max-pool-size="20" instance-acquisition-
timeout="5" instance-acquisition-timeout-unit="MINUTES"/>
        <strict-max-pool name="mdb-strict-max-pool" max-pool-size="20" instance-acquisition-
timeout="5" instance-acquisition-timeout-unit="MINUTES"/>
        <strict-max-pool name="my-stateless-bean-pool" max-pool-size="50" instance-acquisition-
timeout="20" instance-acquisition-timeout-unit="SECONDS"/>
      </bean-instance-pools>
    </pools>
  ...
</subsystem>
```

JBoss EAP 6 で Bean プールを使用するよう EJB を設定

以下の方法の 1 つを使用して、EJB をプールと関連付けることができます。

- アノテーションを使って EJB をプールに関連付けます。ここで、値はサーバー設定ファ

イルで定義されたプールの名前です。以下の例は、EJB を JBoss EAP 6 の「my-stateless-bean-pool」に関連付けます。

```
@Stateless
@org.jboss.ejb3.annotation.Pool(value="my-stateless-bean-pool")
public class MyBean....
```

- EJB JAR の META-INF/ ディレクトリーに jboss-ejb3.xml ファイルを設定して、EJB をプールに関連付けます。以下の例は、JBoss EAP 6 の jboss-ejb3.xml ファイルを使用して EJB をプールに関連付けます。

```
<jboss xmlns="http://java.sun.com/xml/ns/javaee" xmlns:p="urn:ejb-pool:1.0">
 ....
  <assembly-descriptor>
    <p:pool>
      <ejb-name>MyBean</ejb-name>
        <p:bean-instance-pool-ref>my-stateless-bean-pool</p:bean-instance-pool-ref>
      </p:pool>
    </assembly-descriptor>
</jboss>
```

JBoss EAP 6 でのステートレスセッション Bean プールの無効化

EJB インスタンスのメモリーが問題ではなく、EJB インスタンスが PostConstruct メソッドでコストのかかる初期化が行われていない場合は、デフォルトでプールを無効にすると便利です。プールが使用されていない場合、EJB でメソッドを呼び出すスレッドは単に EJB のインスタンスを作成し、その中にメソッドを呼び出し、EJB インスタンスを破棄します。

プーリングを無効にするには、サーバー設定ファイルの ejb3 サブシステムのステートレス EJB <bean-instance-pool-ref> 要素を削除します。

```
<subsystem xmlns="urn:jboss:domain:ejb3:1.3">
  <session-bean>
    <stateless>
      <!-- Remove the following line -->
      <bean-instance-pool-ref pool-name="slsb-strict-max-pool"/>
    </stateless>
    ...
```

バグの報告

3.2.13.4. jboss.xml ファイルの置き換え

概要

jboss.xml デプロイメント記述子ファイルは jboss-ejb3.xml デプロイメント記述子ファイルに置き換えられます。このファイルは、ejb-jar.xml デプロイメント記述子で定義された Java Enterprise Edition(EE)が提供する機能を上書きし、追加するために使用されます。新しいファイルは jboss.xml と互換性がなく、デプロイメントで jboss.xml は無視されるようになりました。

アプリケーションが EJB 2.x を使用する場合は、jboss.xml ファイルを jboss-ejb3.xml デプロイメント記述子ファイルに置き換える必要があります。アプリケーションが EJB 3.x を使用する場合は、jboss-ejb3.xml ファイルを使用するか、EJB3 アノテーションを使用して完全に削除することもできます。

jboss.xml ファイルを jboss-ejb3.xml ファイルに置き換えます。

以前のリリースの JBoss EAP では、ejb -jar.xml ファイルで <resource-ref > を定義した場合、jboss.xml ファイルに JNDI 名に対応するリソース定義が必要でした。XDoclet は、これらのデプロイメント記述子ファイルの両方を自動的に生成します。JBoss EAP 6 では、JNDI マッピング情報が jboss-ejb3.xml ファイルに定義されるようになりました。データソースが Java ソースコードに定義されていることを以下のように仮定します。

```
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.jxml ファイルは、以下の XML 構文を使用して JNDI 名を参照にマップします。

```
<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>
```

不足している jboss.xml 設定属性の処理

JBoss EAP 5.x の jboss.xml ファイルで使用できた一部の設定オプションは JBoss EAP 6 に実装されませんでした。以下のリストは、jboss.xml ファイルで一般的に使用される属性の一部と、JBoss EAP 6 でそれらを実行する代替方法があるかどうかを示します。

- 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す。
  - 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 オプションが jboss-ejb3.xml ファイルに設定されるようになりました。
- missing-method-permission-exclude-mode 属性は、セキュアな Bean に明示的なセキュリティーメタデータを実装しなくてもメソッドの動作を変更しました。JBoss EAP 6 では、@RolesAllowed アノテーションがないことは現在、以下と同様の方法で処理されます。@PermitAll

jboss-ejb3.xml ファイル設定のその他の例は、『MDB の jboss-ejb3.xml のリソースアダプターの指定』、コンテナー『インターセプターの設定』、および JBoss EAP 『開発ガイド』 の「 『jboss-

ejb3.xml デプロイメント記述子参照』 」のトピックを参照してください。『』

バグの報告

3.2.14. 「EJB 2.x and Earlier Changes」

3.2.14.1. EJB 1.x および EJB 2.x の非推奨の機能

JBoss EAP 6 はオープン標準として構築され、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 1.1 および EJB 2.x のサポートを提供しますが、仕様を超えた機能をサポートしない可能性があります。Java EE 6 仕様では以下の機能が非推奨となったことに注意してください。

- EJB 2.1 以前の CMP エンティティー Bean
- EJB 2.1 エンティティー Bean のクライアントビュー
- CMP エンティティー Bean クエリー用の EJB QL
- JAX-RPC ベースの Web サービスエンドポイントとクライアントビュー

EE 7 仕様はこれらの機能を任意にしたため、アプリケーションコードを書き直して EJB 3.x 仕様と JPA を使用することを強く推奨します。

バグの報告

3.2.14.2. EJB 1.x および EJB 2.x を使用するアプリケーションに必要な変更

3.2.14.2.1. EJB 2.x の実行に必要な設定変更

完全プロファイルでのサーバーの起動

EJB 2.x Container Managed Persistence(CMP)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 の実行に必要な設定要素が含まれます。

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ます。

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

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>
```

.

完全なプロファイルで 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* 引数または *-c standalone-full-ha.xml* 引数を渡します。

コンテナー設定の長期サポートなし

これまでのバージョンの JBoss EAP では、CMP エンティティーおよびその他の Bean に異なるコンテナーを設定し、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して使用することができました。たとえば、SLSB からセッション Bean への一般的な設定は異なります。

JBoss EAP 6.x では、標準コンテナーで EJB 2 エンティティー Bean を使用できます。ただし、異なるコンテナー設定に対応しなくなりました。EJB 3 仕様に従って、EJB2 Stateful Session Beans(SFSB)、ステートレスセッション Bean(SLSB)、メッセージ駆動型 Bean(MDB)、および Container-Managed Persistence(CMP)エンティティー Bean で、EJB 3 仕様に従って Java Persistence API(JPA)エンティティー Bean を移行する方法が推奨されます。

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

- デフォルトでは、pessimistic ロックがアクティブになっています。これにより、デッド

ロックが生じる可能性があります。

- JBoss EAP 5.x の CMP レイヤーに含まれていたデッドロック検出コードは JBoss EAP 6 ではなくなりました。

JBoss EAP 5.x では、キャッシング、プーリング、commit-options、インターセプタースタックをカスタマイズすることもできます。JBoss EAP 6 では、これはできなくなりました。commit-option C の Instance Per Transaction ポリシーと類似する実装は 1 つだけです。CMP2.x と互換性のある JDBC ベースの永続マネージャーを使用する cmp2.x jdbc2 pm エンティティー bean コンテナー設定を使用するアプリケーションを移行する場合、パフォーマンスに影響します。このコンテナーは、パフォーマンスに対して最適化されています。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ーを EJB 3 に移行することが推奨されます。

サーバー側のインターセプターの設定

JBoss EAP 6 は、@Interceptors および @AroundInvoke アノテーションを使用して標準の Java EE インターセプター をサポートします。ただし、セキュリティーやトランザクション以外の操作は許可されません。

これまでのバージョンの JBoss EAP では、インターセプタースタックを変更して各 EJB 呼び出しのカスタムインターセプターを指定できました。これは、セキュリティーチェックやトランザクションチェックまたは作成前に、カスタマイズされたセキュリティーまたは再試行メカニズムを実装するために使用されていました。JBoss EAP 6.1 では、同様の機能を提供するコンテナーインターセプターが導入されました。コンテナーインターセプターの詳細は、JBoss EAP 『『開発ガイド』』 の「コンテナー 『インターセプター』 」の章を参照してください。

Java EE 仕様を維持しつつ、トランザクションのコミットフェーズの前後に制御を強化するための別の方法は、Transaction Synchronization Registry を使用してリスナーを追加します。

リソースは、以下のいずれかの方法で取得できます。

- 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());
```

コールバックルーチンは javax.transaction.Synchronization インターフェースを実装する必要があります。beforeCompletion{} メソッドを使用して、トランザクションがコミットまたはロールバックされる前にチェックを実行します。このメソッドから RuntimeException が発生すると、トランザクションはロールバックされ、クライアントには gitops olledbackException が通知さ れます。XA-Transaction の場合、すべてのリソースは XA 契約に従ってロールバックされます。また、afterCompletion(int txStatus) メソッドを使用してトランザクションの状態に依存するようにビジネスロジックを有効にすることもできます。このメソッドから RuntimeException が発生すると、トランザクションは、コミットまたはロールバックのいずれかの以前の状態のままになり、クライアントには通知されません。サーバーログファイル内では、トランザクションマネージャーのみが警告を表示します。

クライアント側インターセプターのサーバー側設定

これまでのバージョンの JBoss EAP では、サーバー設定に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 のクラスだけを提供できました。

JBoss EAP 6 では、クライアントプロキシー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に送信されるため、これは不可能になりました。プロキシーがクライアント側で生成されるようになりました。この最適化により、ルックアップおよびクラスアップロードのサーバー呼び出しが回避されます。

エンティティー Bean プール設定

JBoss EAP 6 では、エンティティー bean プール設定は推奨されません。これは < 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すべてのエンティティーを読み込む場合に、デッドロックやその他の問題が発生する可能性があります。エンティティー bean には初期化中に大きなライフサイクルメソッドがないため、インスタンスの作成と周りのコンテナーの周りはプールされたエンティティー Bean インスタンスを使用する場合よりも遅くなります。

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ファイルは jboss-ejb3.xml デプロイメント記述子ファイルに置

き換えられます。このファイルは、ejb-jar.xml デプロイメント記述子で定義された Java Enterprise Edition(EE)が提供する機能を上書きし、追加するために使用されます。新しい jboss-ejb3.xml ファイルは jboss.xml ファイルと互換性がありません。このファイルは、デプロイメントで無視されるようになりました。詳細は、 「jboss.xml ファイルの置き換え」 を参照してください。

データソースタイプマッピングの設定

これまでのバージョンの JBoss EAP では、データソースの type-mapping を *-ds.xml データソースデプロイメント設定ファイル内に設定できました。

JBoss EAP 6 では、これを jbosscmp-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>
```

これまでのバージョンの JBoss EAP では、カスタマイズされたマッピングが standardjbosscmp-jdbc.xml ファイルで行われていました。このファイルは利用できなくなり、jbosscmp-jdbc.xml デプロイメント記述子ファイルでマッピングを実行できるようになりました。

バグの報告

3.2.14.2.2. EJB 2.x に必要なコンテナー管理による永続性およびコンテナー管理の変更

コンテナー管理(CMR)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、CMR コレクションを繰り返し処理し、関係を削除または追加するために、一部のコンテナー（ cmp2.x jdbc2 pm コンテナーなど）が発生しました。コンテナー設定はサポート対象外であるため、JBoss EAP 6 ではこれをできなくなりました。アプリケーションコードで同じ機能を実現する方法は、カスタマーポータルの「Support Knowledgebase Solutions」セクションの「 EJB2.1 Finder for CMP entities for CMP entity with CMP entities with relations(CMR)returns duplicates in EAP6 」を参照してください。

コンテナー管理(CMR)ファクサーの重複エントリー

これまでのバージョンの JBoss EAP では、異なる永続ストラテジーを使用する異なる CMP コンテナーを選択できました。JBoss EAP 5 .x の cmp2.x jdbc2 pm コンテナーは、最適化された SQL-92 を使用してファクサーに最適化された LEFT OUTER JOIN 構文を生成しました。JBoss EAP 6.x は CMP および CMR の標準コンテナーのみをサポートするため、実装にはこれらの最適化は含まれませ

ん。リファインダーには、結果セットにカテテス語の製品を回避するために、SELECT ステートメントにキーワード DISTINCT を含める必要があります。詳細は、カスタマーポータルの「Support Knowledgebase Solutions」セクションの「 EJB2.1 Finder for CMP entities with 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 のデフォルト変更の削除

カスケード削除のデフォルト値が false に変更になりました。これにより、JBoss EAP 6 で削除される可能性があります。エンティティー関係が cascade-delete とマークされている場合は、jbosscmp-jdbc.xml ファイルで batch-cascade-delete を true に明示的に設定する必要があります。詳細は、カスタマーポータルの「Support Knowledgebase Solutions」の「 cascade delete fail for EJB2 CMP entity 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

JBoss EAP 5.x アプリケーションで JDBCParameterSetter、JDBCResultSetReader、および Mapper などのカスタマーマッパークラスを使用している場合は、アプリケーションを JBoss EAP 6 にデプロイすると java.lang.ClassNotFoundException が表示されることがあります。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 に変更されているためです。詳細は、カスタマーポータルの「サポートナレッジベースソリューション」の「 How to use Field mapping for an EJB2 CMP application in an EJB2 CMP application in an EJB2 CMP application」を参照してください。

entity-commands を使用したプライマリーキーの生成

JBoss EAP 5 アプリケーションで entity-commands を使用してプライマリーキー（ Sequence や Auto-increment など）を生成すると、アプリケーションを JBoss EAP 6 に移行すると、JDBCOracleSequenceCreateCommand クラスの ClassNotFoundException が表示されることがあります。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 に変更されているためです。JBoss EAP 6 アプリケーションでこのクラスを使用する場合は、EAP_HOME/modules/system/layers/base/org/jboss/as/cmp モジュールに依存関係も追加する必要があります。

バグの報告

3.2.14.2.3. EJB 2.x の実行に必要なアプリケーションの変更

新しい JNDI ネームスペースルールを使用するようコードを変更します。

EJB 3.0 と同様に、EJB 2.x と完全な JNDI プレフィックスを使用する必要があります。新しい JNDI ネームスペースルールおよびコード例の詳細は、 「アプリケーション 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の例は、 「以前のリリースの JNDI 名前空間の例と JBoss EAP 6 での指定方法」 を参照してください。

jboss-web.xml ファイル記述子の変更

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

XDoclet を使用して内部ローカルインターフェースの JNDI 名をマッピングする

EJB 2 では、Locator パターンを使用して Bean を検索することは非常に一般的でした。アプリケーションコードを変更するのではなく、アプリケーションでこのパターンを使用した場合は、XDoclet を使用して新しい JNDI 名のマップを生成できます。

一般的な 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"
```

上記の例の JNDI 名 ejb21/UserAttributeEntity は JBoss EAP 6 では有効ではなくなりました。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 名にマップできます。

上記の段落で示さ 『れているように、カスタマイズされたマッパーを作成できます。カスタムフィールドのエンタイトルメントがある CMP』 カスタムマッパー、または以下の手順で説明しているようにコードを変更できます。

手順3.29 XDoclet 生成コードの変更および Naming サブシステムの使用

1. ejb-module.jar にある XDoclet lookup.xdt テンプレートを展開し、lookup() の lookupHome を以下のように変更します。

```
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、ejbdoclet タスクに変更した lookup.xdt を使用するように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 サブシステムを変更し、古い JNDI 名を新しい有効な JNDI 名にマッピングします。

```
<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.cmr.UserAttributeLocalHom
e"/>
    </bindings>
    <remote-naming/>
</subsystem>
```

バグの報告

3.2.14.2.4. EJB 2.x に関する既知の問題

JBoss EAP 6.3 以前のリリースで Message Driven Beans(MDB)を定義する EJB 2.0 ejb-jar.xml 記述子ファイルのデプロイメントには既知の問題があります。サーバーはデプロイメント時に解析エラーをスローし、サーバーログに以下の出力が表示されます。

```
ERROR [org.jboss.msc.service.fail] (MSC service thread 1-7) MSC000001: Failed to start
service jboss.deployment.unit."EJB_JAR_NAME.jar".PARSE:
org.jboss.msc.service.StartException in service
```

> **jboss.deployment.unit."*EJB_JAR_NAME*jar".PARSE: JBAS018733: Failed to process phase PARSE of deployment "*EJB_JAR_NAME*.jar"**

この問題は JBoss EAP 6.4 で修正されました。詳細は、Bugzilla 1057835 を参照してください。

バグの報告

3.2.14.2.5. EJB 2.x ファイルの廃止

JBoss EAP 6 では、以下のファイルに対応しなくなりました。

jboss.xml

jboss.xml アプリケーションデプロイメント記述子ファイルはサポートされず、デプロイされたアーカイブに含まれる場合は無視されます。このファイルは、jboss-ejb3.xml デプロイメント記述子ファイルに置き換えられました。

standardjbosscmp-jdbc.xml

standardjbosscmp-jdbc.xml サーバー設定ファイルに対応しなくなりました。この設定情報は org.jboss.as.cmp モジュールに含まれ、カスタマイズできなくなりました。

standardjboss.xml

standardjboss.xml サーバー設定ファイルはサポートされなくなりました。この設定情報は、管理対象ドメインで実行している場合、スタンドアロンサーバーまたは domain.xml ファイルを実行する際に standalone.xml ファイルに含まれるようになりました。

バグの報告

3.2.15. JBoss AOP の変更

3.2.15.1. JBoss AOP を使用するアプリケーションの更新

JBoss AOP（アスペクト指向プログラミング）は JBoss EAP 6 に含まれなくなりました。以前のリリースでは、JBoss AOP は EJB コンテナーによって使用されていました。JBoss EAP 6 では、EJB コンテナー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 を使用する場合は、以下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。

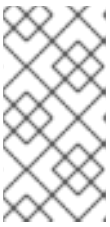
アプリケーションのリファクタリング

- ejb3-interceptors-aop.xml ファイルに以前に作成された 標準の EJB3 設定がサーバー設定ファイル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の場合、これは standalone/configuration/standalone-full.xml ファイルになります。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 ファイルになります。

- 標準の Java EE Interceptor を使用するように、サーバー側 AOP インターセプターを変更する必要があります。コンテナーインターセプターと、アプリケーションでクライアント側のインターセプターを使用する方法は、カスタマーポータル https://access.redhat.com/documentation/ja-jp/red_hat_jboss_enterprise_application_platform/?version=6.4 の JBoss EAP 6 の 『『開発ガイド』』 の「コンテナー 『インターセプター』 」の章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て、アプリケーションでバンドルできます。AOP ライブラリーは JBoss EAP 6 で動作しますが、デプロイされていません。サーバーの起動時に以下のコマンドライン引数を使用して手動でデプロイできます。-Djboss.aop.path=*PATH_TO_AOP_CONFIG*



注記

JBoss AOP ライブラリーは JBoss EAP 6 で動作しますが、この設定はサポートされていません。

バグの報告

3.2.16. jacorb の変更

3.2.16.1. jacorb 設定の変更

JBoss EAP 5 では、*EAP_HOME*/server/production/conf/jacorb.properties ファイルに指定されたプロパティーを使用して JacORB 設定が実行されました。以下は、そのファイルに設定された JacORB プロパティーの例になります。

```
jacorb.connection.client.pending_reply_timeout=600000
jacorb.connection.client.idle_timeout=120000
jacorb.connection.server.timeout=300000
jacorb.native_char_codeset=UTF8
jacorb.native_wchar_codeset=UTF16
```

JBoss EAP 6 では、管理 CLI を使用してサーバー設定ファイルに JacORB プロパティーが設定されます。管理 CLI を使用して JacORB プロパティーを設定する方法は、カスタマーポータル https://access.redhat.com/documentation/ja-jp/red_hat_jboss_enterprise_application_platform/?version=6.4 の JBoss Enterprise Application Platform の『『Administration and Configuration Guide』』のトピックを参照してください。『』

バグの報告

3.2.17. JBoss Web コンポーネントの変更

3.2.17.1. HTTP/HTTPS/AJP コネクター属性のマップ

以下の表は、以前のリリースから Red Hat JBoss Enterprise Application Platform 6 の新しい属性に HTTP、HTTPS、および AJP コネクター属性をマッピングする方法を示しています。

表3.10 マップコネクターの属性

以前のリリースの属性名	JBoss EAP 6 で同等のもの	Details
maxThreads	max-connections	この属性は、JBossWeb レベルのスレッドプールのサイズになります。これは Web サブシステムのコネクターに設定されます。デフォルトは CPU コアごとに 512 です。 `<connector name="http" protocol="HTTP/1.1" scheme="http" socket-binding="http" enabled="true" max-connections="200" />`

以前のリリースの属性名	JBoss EAP 6 で同等のもの	Details
minSpareThreads maxSpareThreads	該当なし	スレッドの回収にはほとんどないため、**minSpareThreads** または **maxSpareThreads** と同等のものはありません。デフォルトのワーカースレッドプールの代わりにコネクターにエクゼキューターを使用する場合、最も近いものは **core-threads** サイズおよび **keepalive-time** 属性になります。
proxyName proxyPort	proxy-name proxy-port	この属性は **Web** サブシステムの **コネクター** に設定されます。 `<connector name="http" protocol="HTTP/1.1" scheme="http" socket-binding="http" enabled="true" proxy-name="proxy.com" proxy-port="80"/>`
redirectPort	redirect-port	この属性は **Web** サブシステムの **コネクター** に設定されます。 `connector name="http" protocol="HTTP/1.1" scheme="https" secure="true" socket-binding="http" redirect-port="8443" proxy-name="loadbalancer.hostname.com" proxy-port="443"`
enableLookups	enable-lookups	この属性は **Web** サブシステムの **コネクター** に設定されます。 `<connector name="http" protocol="HTTP/1.1" scheme="http" socket-binding="http" enable-lookups="true"/>"`
MaxHttpHeaderSize	システムプロパティー	この属性はシステムプロパティーを使用して設定されます。デフォルト値は 8 KB です。 `<system-properties> <property name="org.apache.coyote.http11.Http11Protocol.MAX_HEADER_SIZE" value="8192"/> </system-properties>`
maxKeepAliveRequests	システムプロパティー	この属性はシステムプロパティーを使用して設定されます。 `<system-properties> <property name="org.apache.coyote.http11.Http11Protocol.MAX_KEEP_ALIVE_REQUESTS" value="1"/> </system-properties>`

以前のリリースの属性名	JBoss EAP 6 で同等のもの	Details

connectionTimeout	システムプロパティー	この属性はシステムプロパティーを使用して設定されます。以下の設定では、AJP connectionTimeout を 600000 ミリ秒（10 分）に設定し、HTTP connectionTimeout を 120000 ミリ秒（2 分）に設定します。
		
		`<system-properties>`
		`  <!-- connectionTimeout for AJP connector. Default value is "-1" (no timeout). -->`
		`  <property name="org.apache.coyote.ajp.DEFAULT_CONNECTION_TIMEOUT" value="600000"/>`
		`  <!-- connectionTimeout for HTTP connector. Default value is "60000". -->`
		`  <property name="org.apache.coyote.http11.DEFAULT_CONNECTION_TIMEOUT" value="120000"/>`
		`</system-properties>`

圧縮	システムプロパティー	この属性は圧縮を有効にします。コンテンツタイプを指定できます。デフォルトは **text/html,text/xml,text/plain** です。圧縮されるコンテンツの最小サイズを指定することもできます。デフォルトは **2048** バイトです。圧縮はシステムプロパティーを使用して設定されます。
		
		`<system-properties>`
		`  <property name="org.apache.coyote.http11.Http11Protocol.COMPRESSION" value="on"/>`
		`  <property name="org.apache.coyote.http11.Http11Protocol.COMPRESSION_MIN_SIZE" value="4096"/>`
		`  <property name="org.apache.coyote.http11.Http11Protocol.COMPRESSION_MIME_TYPES" value="text/javascript,text/css,text/html"/>`
		`</system-properties>`

URIencoding	システムプロパティー	この属性はシステムプロパティーを使用して設定されます。
		
		`<system-properties>`
		`  <property name="org.apache.catalina.connector.URI_ENCODING" value="UTF-8"/>`
		`</system-properties>`

以前のリリースの属性名	JBoss EAP 6 で同等のもの	Details
useBodyEncodingForURI	システムプロパティー	この属性はシステムプロパティーを使用して設定されます。 <system-properties> <property name="org.apache.catalina.connector.USE_BODY_ENCODING_FOR_QUERY_STRING" value="true"/> </system-properties>
server	システムプロパティー	この属性はシステムプロパティーを使用して設定されます。 <system-properties> <property name="org.apache.coyote.http11.Http11Protocol.SERVER" value="NewServerHeader"/> </system-properties>
allowTrace	システムプロパティー	この属性はシステムプロパティーを使用して設定されます。 <system-properties> <property name="org.apache.catalina.connector.ALLOW_TRACE " value="true"/> </system-properties>
xpoweredby	システムプロパティー	この属性はシステムプロパティーを使用して設定されます。 <system-properties> <property name="org.apache.catalina.connector.X_POWERED_BY " value="true"/> </system-properties>
keepAliveTimeout	該当なし	JBoss EAP 6.4 以前では、JBoss EAP 6 には同等のパラメーターがありませんでした。内部的には、これはデフォルトで **connectionTimeout** の値に設定されます。 JBoss EAP 6.4 では、新しいシステムプロパティー **org.apache.coyote.http11.DEFAULT_KEEP_ALIVE_TIMEOUT** が導入され、keepAliveTimeout を制御します。**-Dorg.apache.coyote.http11.DEFAULT_KEEP_ALIVE_TIMEOUT** 引数は、**-Dorg.apache.coyote.http11.DEFAULT_DISABLE_UPLOAD_TIMEOUT=true** 引数と共に使用する場合に影響を与えます。
disableUploadTimeout connectionUploadTimeout	該当なし	現在 JBoss EAP 6 には同等のパラメーターはありません。**disableUploadTimeout** はデフォルトで true で、**connectionUploadTimeout** は内部的に **connectionTimeout** の値を使用します。

以前のリリースの属性名	JBoss EAP 6 で同等のもの	Details

packetSize	システムプロパティー	この属性はシステムプロパティーを使用して設定されます。以下の設定では、AJP packetSize を **20000**に設定します。
		
		`<system-properties>`
		`<property name="org.apache.coyote.ajp.MAX_PACKET_SIZE" value="20000"/>`
		`</system-properties>`

maxPostSize maxSavePostSize	max-post-size max-save-post-size	値が **0** の場合は無制限を意味します。このパラメーターは、**Content-Type** が **application/x-www-form-urlencoded** の場合にのみデータサイズを制限することができることに注意してください。詳細は、カスタマーポータルの「How to limit data size of HTTP POST method from a client to JBoss」を参照してください。

tomcatAuthentication	システムプロパティー	JBoss EAP 6 のバージョンに応じて、この属性はシステムプロパティー **org.apache.coyote.ajp.AprProcessor.TOMCATAUTHENTICATION** または **org.apache.coyote.ajp.DEFAULT_TOMCAT_AUTHENTICATION** を使用して設定されます。すべてのバージョンのサーバーにパッチまたはアップグレードが必要になります。
		
		JBoss EAP 6.0.1 では、tomcatAuthentication は以下のプロパティーを使用して設定されます。
		
		`<system-properties>`
		`<property name="org.apache.coyote.ajp.AprProcessor.TOMCATAUTHENTICATION" value="false"/>`
		`</system-properties>`
		
		JBoss EAP 6.1 以降では、tomcatAuthentication は以下のように設定されます。
		
		`<system-properties>`
		`<property name="org.apache.coyote.ajp.DEFAULT_TOMCAT_AUTHENTICATION" value="false"/>`
		`</system-properties>`
		
		詳細は、カスタマーポータルの「How to configure tomcatAuthentication in JBoss EAP 6」を参照してください。

コネクター属性の詳細は、カスタマーポータルの「JBoss EAP 5.x と JBoss EAP 6.x との間の HTTP/HTTPS/AJP コネクター属性のマッピング」を参照してください。

バグの報告

3.2.17.2. 1-Way SSL の設定

JBoss EAP 6 では、管理インターフェースに 1 方向 SSL を設定するメソッドが変更になりました。このトピックでは、JBoss EAP 6 で一方向 SSL を設定する方法を説明します。

JBoss EAP 5 では、証明書ストアを生成しました。そのプロセスによって作成された keystore.jks ファイルは *EAP_HOME*/server/*PROFILE*/conf/ ディレクトリーに置かれました。その後、以下のように *EAP_HOME*/server/*PROFILE*/deploy/jbossweb.sar/server.xml ファイルで HTTP コネクターを設定します。

```
<Connector protocol="HTTP/1.1" SSLEnabled="true"
  port="8443" address="${jboss.bind.address}"
  scheme="https" secure="true" clientAuth="false"
  keystoreFile="${jboss.server.home.dir}/conf/keystore.jks"
  keystorePass="password" SSLProtocol = "TLS" />
```

JBoss EAP 6 で HTTP コネクターを設定するには、証明書ストアも生成します。スタンドアロンサーバーまたは管理対象ドメインのどちらで実行しているかに応じて、このプロセスによって作成された keystore.jks ファイルは *EAP_HOME*/standalone/configuration/ または *EAP_HOME*/domain/configuration/ ディレクトリーに配置されます。その後、以下のように HTTP コネクターは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。

```
<connector name="https" protocol="HTTP/1.1" scheme="https" socket-binding="https">
  <ssl name="ssl" key-alias="jboss" password="password"
    certificate-key-file="${jboss.server.config.dir}/keystore.jks"
    protocol="TLSv1" verify-client="false"/>
</connector>
```

JBoss EAP インストールでオペレーティングシステムのネイティブコンポーネントをインストールし、Apache Portability Runtime(APR)もインストールしている場合は、代わりに APR コネクターを使用して一方向 SSL を設定できます。HTTP コネクターと同様に、APR コネクターは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が、コネクタープロトコルは "org.apache.coyote.http11.Http11AprProtocol" に設定されます。

選択した方法に関係なく、SSL 設定の後にサーバーを再起動するようにしてください。

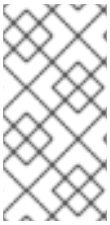
バグの報告

3.2.17.3. バルブ設定の移行

これまでのバージョンの JBoss EAP では、JAR を *EAP_HOME*/server/*PROFILE*/lib ディレクトリーに追加し、以下のファイルのいずれかを設定して Tomcat バルブを設定できます。

- JBoss EAP 4.x では、バルブは *EAP_HOME*/server/*PROFILE*deploy/jboss-web.deployer/server.xml または *EAP_HOME*/server/*PROFILE*deploy/jboss-web.deployer/context.xml ファイルのいずれかで設定されました。
- JBoss EAP 5.x では、バルブは *EAP_HOME*/server/*PROFILE*deploy/jbossweb.sar/server.xml ファイルに設定されました。

JBoss EAP 6 ではこれが変更になりました。バルブを有効にするには、まずモジュールを作成し、サーバー設定ファイルの web サブシステムでモジュールを設定します。バルブの設定方法に関する詳細は、カスタマーポータルの JBoss EAP『『Administration and Configuration Guide』』の「titled『Global Valves』」の章を参照してください。
https://access.redhat.com/documentation/ja-jp/red_hat_jboss_enterprise_application_platform/?version=6.4



注記

グローバルバルブは JBoss EAP 6.0.x ではサポートされません。JBoss EAP 6.1 以降にアップグレードすると、JBoss EAP 6.1 以上を使用する必要があります。

バグの報告

3.2.17.4. CertificatePrincipal クラスの設定

JBoss EAP 5 では、以下の例のように jbossweb-tomcat.sar/server.xml ファイルの certificatePrincipal の Realm 属性を設定して CertificatePrincipal クラスが JBossWeb コンテナーで設定されている。

```
<Realm className="org.jboss.web.tomcat.security.JBossWebRealm"
  certificatePrincipal="org.jboss.security.auth.certs.SubjectDNMapping"
```

```
allRolesMode="authOnly"/>
```

JBoss EAP 6 には、CertificatePrincipal インターフェースと複数の実装（ SubjectDNMapping クラスなど）が含まれます。JBoss EAP 6 でこれらの実装を使用するには、カスタムログインモジュールを作成します。カスタムログインモジュールを作成する方法は、にある Red Hat JBoss Enterprise Application Platform 6.4 の『『Login Module Reference』』を参照 https://access.redhat.com/documentation/ja-jp/red_hat_jboss_enterprise_application_platform/?version=6.4 してください。

バグの報告

3.2.18. Seam 2.2 アプリケーションの移行

3.2.18.1. 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場合は、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、アプリケーションに JBoss EAP 6 には同梱されないアーカイブの依存関係があるかどうかを確認し、依存する JAR をアプリケーションの lib/ ディレクトリーにコピーします。

重要

Hibernate を Seam 2.2 で直接使用するアプリケーションは、アプリケーション内にパッケージ化された Hibernate 3 のバージョンを使用する場合があります。JBoss EAP 6 の org.hibernate モジュールで提供される Hibernate 4 は Seam 2.2 ではサポートされません。この例は、最初の手順としてアプリケーションを JBoss EAP 6 で実行できるようにするのに役立つことを目的としています。Seam 2.2 アプリケーションと共に Hibernate 3 をパッケージ化するオプションはサポートされないことに注意してください。

手順3.30 Seam 2.2 アーカイブの移行

1. データソース設定を更新します。

Seam 2.2 の例では、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。このデフォルトのデータソースは JBoss EAP 6 で

java:jboss/datasources/ExampleDS に変更されています。アプリケーションがサンプルデータベースを使用する場合は、以下のいずれかを実行できます。

- 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、META-INF/persistence.xml ファイルを変更して、既存の jta-data-source 要素をデータベースのデータソースの JNDI 名に置き換えます。

```
<!-- <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保持する場合は、データソース定義を *EAP_HOME*/standalone/configuration/standalone.xml ファイルに追加できます。

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。

以下の定義は、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定義を追加することもできます。以下は、データソースの追加に使用する構文の例です。行の最後にある「\"」は、以下の行のコマンドの継続を示しています。

例3.12 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します。

Seam 2.2 アプリケーションは JSF 1.2 を使用しているため、JSF 1.2 モジュールの依存関係を追加して 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、以下のデータが含まれる 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のロギングフレームワークを使用する場合は、「ロギング依存関係の変更」 に記載されているように、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します。

アプリケーションが Seam Managed Persistence Context、Hibernate 検索、検証、また

はその他の機能を使用していない場合は、Hibernate 4 ライブラリーで実行することができます。ただし、Hibernate クラスを参照する ClassNotFoundExceptions または ClassCastExceptions が表示された場合や、以下のようなエラーが表示される場合には、次の手順の手順に従い、アプリケーションを Hibernate 3.3 ライブラリーを使用するように変更する必要がある場合があります。

```
Caused by: java.lang.LinkageError: loader constraint violation in interface itable initialization: when resolving method "org.jboss.seam.persistence.HibernateSessionProxy.getSession(Lorg/hibernate/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から依存アーカイブをコピーします。**

アプリケーションが Hibernate 3.x を使用し、アプリケーションで Hibernate 4 が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 deployments セクションの 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>
```

アプリケーションと Hibernate 3.x をバンドルする場合には、追加の手順を実施する必要があります。詳細は、「Hibernate および JPA を使用するアプリケーションの変更の設定」を参照してください。

5. **Seam 2.2 JNDI エラーをデバッグして解決します。**

Seam 2.2 アプリケーションを移行すると、以下のようなログに javax.naming.NameNotFoundException エラーが表示される可能性があります。

```
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たくない場合は、アプリケーションの components.xml ファイルを以下のように変更できます。

a.

既存の core-init 要素を置き換えます。

まず、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 JNDI バインディング INFO メッセージを検索します。

次に、アプリケーションのデプロイ時にサーバーログに出力される JNDI バインディング INFO メッセージを見つけます。JNDI バインディングメッセージは以下のようになります。

```
INFO
org.jboss.as.ejb3.deployment.processors.EjbJndiBindingsDeploymentUnitProces
sor (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.Authenticator
   java:app/jboss-seam-
booking.jar/AuthenticatorAction!org.jboss.seam.example.booking.Authenticator

java:module/AuthenticatorAction!org.jboss.seam.example.booking.Authenticator
   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を追加します。

ログの JNDI バインディング INFO メッセージごとに、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" />
```

移行の問題のデバッグおよび解決方法に関する詳細は、 「移行の問題のデバッグおよび解決」 を参照してください。

Seam 2 アーカイブの既知の問題の一覧は、 「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 アーカイブは JBoss EAP 6 で正常にデプロイされ、実行されます。

バグの報告

3.2.18.2. Seam 2.2 アーカイブの移行の問題

Seam 2.2 Drools と Java 7 は互換性がない

Seam 2.2 Drools と Java 7 は互換性がなく、org.drools.RuntimeDroolsException: 値 '1.7' は有効な言語レベルのエラーで失敗します。

Seam 2.2.5 署名 cglib.jar により、Spring の例が動作しない

JBoss EAP 5 の Seam 2.2.5 に同梱されている署名済み cglib.jar を使用して Spring の例を実行すると、以下の原因で失敗します。

```
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
```

この問題の回避策として、以下のように cglib.jar の署名を解除します。

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

Seambay の例が NotLoggedInExceptionで失敗する

この問題の原因は、SOAPRequestHandler でメッセージを処理すると SOAP メッセージヘッダーが null であるため、会話 ID は設定されていません。

この問題を回避するには、で説明されているように org.jboss.seam.webservice.SOAPRequestHandler.handleOutbound を上書きし https://issues.jboss.org/browse/JBPAPP-8376 ます。

Seambay の例が UnsupportedOperationException: no transaction で失敗する

このバグは、JBoss EAP 6 の UserTransaction の JNDI 名の変更によって生じます。

この問題を回避するには、の説明どおりに org.jboss.seam.transaction.Transaction.getUserTransaction を上書きし https://issues.jboss.org/browse/JBPAPP-8322 ます。

タスクの例は、org.jboss.resteasy.spi.UnhandledException: Unable to unmarshall request body をスローします。

このバグは、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 間の非互換性によって生じます。

この問題の回避策と https://issues.jboss.org/browse/JBPAPP-8315 して、jboss-deployment-structure.xml を使用して、メインのデプロイメントから resteasy-jaxrs、resteasy-jettison-provider、resteasy-jaxb-provider、resteasy-jaxrs、resteasy-jettison-provider、resteasy-jaxb-provider、resteasy-jaxb-provider、resteasy-yaml-provider を jboss-seam-tasks.war から除外することです。次に、EAR に Seam 2.2 とバンドルされた RESTEasy ライブラリーを含める必要があります。

AJAX 要求中の 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 ロック間のデッドロック

"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" or similar error message is could not acquire lock on @Synchronized component: hotelSearch" or similar error メッセージが表示されます。

この問題は、Seam 2 がステートフルセッション Bean(SFSB)ロック外の独自のロックと、異なるスコープである点です。つまり、スレッドが同じトランザクションで EJB に 2 回アクセスされた場合、最初の呼び出しの後に SFSB ロックがありますが、seam ロックは存在することを意味します。2 つ目のスレッドは、seam ロックの取得が可能です。次に、EJB ロックに到達して待機する

ことができます。最初のスレッドが 2 つ目の呼び出しを試みると、seam 2 インターセプターとデッドロックでブロックされます。Java EE 5 では、EJB は同時アクセスで即座に例外を発生させました。この動作は Java EE 6 で変更になりました。

この問題を回避するには、@AccessTimeout(0)を EJB に追加します。これにより、このような状況が発生すると 、すぐに Concurrent Warehouse をスローします。

dvdstore の例 create order が javax.ejb.gitopsolledbackExceptionで失敗する

dvdstore の例では、以下のエラーが表示されます。

> 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。

この問題の修正は、永続コンテキストを CheckoutAction および ShowOrdersAction クラスでトランザクション に変更し、cancelOrder メソッドおよび detailOrder メソッドでエンティティーマネージャーのマージ操作を使用することです。

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

JBoss EAP 6 では JBoss キャッシュはサポートされません。これにより、JBoss Cache Seam Cache プロバイダーは、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 における JPA エンティティーの問題に対する Hibernate 3.3.x 自動スキャン

この問題の修正は、persistence.xml ファイルのすべてのエンティティークラスを手動で一覧表示することです。以下に例を示します。

```
<?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 以外のスレッドから EJB Seam コンポーネントを呼び出すと、javax.naming.NameNotFoundException が発生します。

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 名前空間規則を導入するために JBoss EAP 6 の変更が原因です。java:app namespace は、1 つのアプリケーションのすべてのコンポーネントによって共有される名前について指定されます。Quartz 非同期スレッドなどのEE 以外のスレッドは、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 コンポーネントを呼び出しようとすると javax.naming.NameNotFoundException を受信した場合、以下のように components.xml ファイルを変更してグローバル JNDI 名を使用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 JNDI 名前空間名の更新」 のトピックを参照してください。この特定の問題に関する詳細は、Red Hat カスタマーポータルの 『Red Hat JBoss Web Framework Kit』 の 2.2 『. 『0 リリースノート』 の quartz 非同期メソッドから EJB Seam コンポーネントを呼び出しようとすると、BZ#948215 - Seam2.3 javax.naming.NameNotFoundException trying』 を参照してください。

バグの報告

3.2.19. Spring アプリケーションの移行

3.2.19.1. Spring アプリケーションの移行

Spring アプリケーションの移行方法は、*Red Hat JBoss Web Framework Kit* のドキュメントを参照してください。『Spring Installation Guide』 および 『Spring Developer Guide』 は、で複数の形式で利用でき https://access.redhat.com/documentation/ja-JP/Red_Hat_JBoss_Web_Framework_Kit/ ます。

バグの報告

3.2.20. 影響を受ける移行のその他の変更

3.2.20.1. 移行の影響を受ける可能性がある他の変更点についてよく知る

以下は、移行作業に影響する可能性がある JBoss EAP 6 におけるその他の変更のリストです。

- 「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」

バグの報告

3.2.20.2. Maven プラグイン名の変更

jboss-maven-plugin が更新されず、JBoss EAP 6 では機能しません。これで、org.jboss.as.plugins:jboss-as-maven-plugin を使用して正しいディレクトリーにデプロイする必要があります。

バグの報告

3.2.20.3. 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を移行する予定がある場合は、クライアントライブラリーをバンドルする JAR の名前および場所が変更されたことに注意してください。この JAR の名前は jboss-client.jar で、*EAP_HOME*/bin/client/ ディレクトリーにあります。*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ます。

バグの報告

第4章 ツールおよびヒント

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

以下の例は、アプリケーションを JBoss EAP 6 に移行する際に役立つ可能性のあるリソースの一覧です。

ツール

設定変更の一部を自動化するツールがいくつかあります。詳細は、 「移行に役立つツールに慣れる」 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表示される可能性のある問題およびエラーの最も一般的な原因と解決策の一覧は、 「移行の問題のデバッグおよび解決」 を参照してください。

移行の例

JBoss EAP 6 に移行したアプリケーションの例は、 「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。

バグの報告

4.1.2. 移行に役立つツールに慣れる

概要

移行作業に役立つツールがいくつかあります。以下は、これらのツールの一覧と、そのツールの動作を示しています。

Tattletale

モジュラークラスローディングの変更に伴い、アプリケーションの依存関係を見つけ、修正する必要があります。Tattletale は、依存するモジュール名を特定し、アプリケーションの設定 XML を生成するのに役立ちます。

「Tattletale を使用したアプリケーション依存関係の検索」

IronJacamar Migration Tool

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で設定されなくなりました。サーバー設定ファイルで定義され、新しいスキーマを使用するようになりました。IronJacamar Migration Tool は、以前の設定を JBoss EAP 6 によって想定される形式に変換できます。

「IronJacamar Tool を使用したデータソースおよびリソースアダプター設定の移行」

バグの報告

4.1.3. Tattletale を使用したアプリケーション依存関係の検索

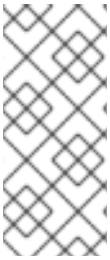
概要

JBoss EAP 6 のモジュラークラスローディングの変更により、アプリケーションの移行時に JBoss ログに ClassNotFoundException または ClassCastException トレースが表示されることがあります。これらのエラーを解決するには、例外によって指定されたクラスが含まれる JAR を見つける必要があります。

Tattletale は、アプリケーションを再帰的にスキャンし、そのコンテンツに関する詳細な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 以降には、JBoss EAP 6 で使用される新しい JBoss Modules 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、アプリケーションの jboss-deployment-structure.xml ファイルを組み込むため、依存するモジュール名を自動的に特定および生成することができます。

手順4.1 Tattletale をインストールして実行し、アプリケーションの依存関係を検索します。

1. 「Tattletale のダウンロードおよびインストール」
2. 「Tattletale レポートの作成およびレビュー」



注記

Tattletale はサードパーティーのツールで、JBoss EAP 6 の一部としてはサポートされません。Tattletale のインストールおよび使用に関する最新のドキュメントは、の Tattletale Web サイトを参照 http://tattletale.jboss.org/ してください。

バグの報告

4.1.4. Tattletale のダウンロードおよびインストール

手順4.2 Tattletale のダウンロードおよびインストール

1. Tattletale バージョン 1.2.0.Beta2 以降を から http://sourceforge.net/projects/jboss/files/JBoss%20Tattletale ダウンロードします。

2. 任意のディレクトリーに展開します。

3. 以下を実行して TATTLETALE_HOME/jboss-tattletale.properties ファイルを変更します。

   a. ee6 および as7 を profiles プロパティーに追加します。

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

   b. scan および reports プロパティーのコメントを解除します。

バグの報告

4.1.5. Tattletale レポートの作成およびレビュー

1.

以下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*

例： java -jar tattletale-1.2.0.Beta2/tattletale.jar ~/applications/jboss-seam-booking.ear ~/output-results/

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

a. 左側の列には、アプリケーションが使用す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て、その場所、マニフェスト情報、含まれるクラスなどのアーカイブの詳細を表示します。

b. 右側の列の jboss-deployment-structure.xml リンクは、左側の列で名前が付けられたアーカイブのモジュール依存関係を指定する方法を示しています。このリンクをクリックして、この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。

バグの報告

4.1.6. IronJacamar Tool を使用したデータソースおよびリソースアダプター設定の移行

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、*-ds.xml の接尾辞が付いた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、デプロイされていました。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 が期待する形式に変換するために使用できる移行ツールが含まれています。ツールは以前のリリースのソース設定ファイルを解析し、XML 設定を作成し、新しい形式で出力ファイルに書き込みます。この XML は、JBoss EAP 6 サーバー設定ファイルの正しいサブシステムの下にコピーおよび貼り付けることができます。このツールは、古い属性と要素を新しい形式に変換するベスト作業を行いますが、生成されたファイルに追加の変更が必要になる場合があります。

手順4.3 IronJacamar Migration ツールのインストールおよび実行

1. 「IronJacamar Migration Tool のダウンロードおよびインストール」

2. 「IronJacamar Migration Tool を使用したデータソース設定ファイルへの変換」

3. 「IronJacamar Migration Tool を使用したリソースアダプター設定ファイルへの変換」

注記

IronJacamar Migration ツールはサードパーティーのツールで、JBoss EAP 6 の一部としてはサポートされません。IronJacamar の詳細は、を http://www.ironjacamar.org/ 参照してください。このツールのインストールおよび使用に関する最新のドキュメントは、を http://www.ironjacamar.org/documentation.html 参照してください。

バグの報告

4.1.7. IronJacamar Migration Tool のダウンロードおよびインストール

注記

移行ツールは、IronJacamar 1.1 以降でのみ利用でき、Java 7 以降が必要になります。

1. IronJacamar の最新のディストリビューションを からダウンロードします。
http://www.ironjacamar.org/download.html

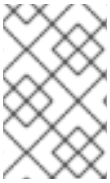
2. ダウンロードしたファイルを任意のディレクトリーに展開します。

3. IronJacamar ディストリビューションでコンバータースクリプトを見つけます。

- 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。

バグの報告

4.1.8. IronJacamar Migration Tool を使用したデータソース設定ファイルへの変換



注記

IronJacamar コンバータースクリプトには Java 7 以降が必要です。

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/ ディレクトリーに移動します。
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。
   - For Linux: ./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* には新しい設定が含まれます。

   たとえば、現在の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、以下を入力します。

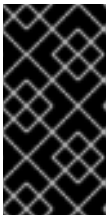
   -

For Linux: ./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 であることに注意してください。

3. ターゲットファイルから <datasource> 要素をコピーし、それを <subsystem xmlns="urn:jboss:domain:datasources:1.1"><datasources> 要素の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。

- 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で生成された XML を変更します。

以下は、JBoss EAP 5.x に同梱される Seam 2.2 Booking サンプルの jboss-seam-booking-ds.xml データソース設定ファイルの例です。

```
<?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>
```

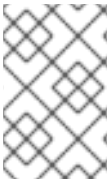
以下は、Converter スクリプトを実行して生成された設定ファイルです。生成されたファイルには <driver-class> 要素が含まれます。JBoss EAP 6 でドライバークラスを定義するのに推奨される方法は、<driver> 要素を使用することです。以下は、JBoss EAP 6 設定ファイルの結果として生成される XML で、<driver-class> 要素をコメントアウトし、対応する <driver> 要素を追加します。

```
<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>
```

バグの報告

4.1.9. IronJacamar Migration Tool を使用したリソースアダプター設定ファイルへの変換



注記

IronJacamar コンバータースクリプトには Java 7 以降が必要です。

1. コマンドラインを開き、*IRONJACAMAR_HOME*/docs/as/ ディレクトリーに移動します。
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。
   - For Linux: ./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。

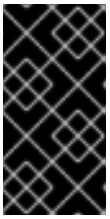
たとえば、現在のディレクトリーにある mttestadapter-ds.xml リソースアダプター設定ファイルを変換するには、以下を入力します。

- For Linux: ./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 であることに注意してください。

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



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。

- 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で生成された XML を変更します。

以下は、JBoss EAP 5.x TestSuite からの mttestadapter-ds.xml リソースアダプター設定ファイルの例です。

```
<?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、Converter スクリプトを実行して生成された設定ファイルです。生成された XML

の class-name 属性値 "FIXME_MCF_CLASS_NAME" を、管理接続ファクトリーの正しいクラス名に置き換えます（この場合は "org.jboss.test.jca.adapter.TestManagedConnectionFactory"）。以下は、JBoss EAP 6 設定ファイルの生成される XML で <class-name> 要素の値を変更します。

```
<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>
```

バグの報告

4.2. デバッグ移行の問題

4.2.1. 移行の問題のデバッグおよび解決

クラスローディング、JNDI 命名ルール、アプリケーションサーバーのその他の変更により、アプリケーションを「as-is」デプロイしようとすると例外やその他のエラーが発生する可能性があります。ここでは、より一般的な例外と発生する可能性のあるエラーの一部を解決する方法を説明します。

- 「debug および Resolve ClassNotFoundExceptions および NoClassDefFoundErrors」
- 「Debug and Resolve ClassCastExceptions」
- 「デバッグおよび解決された DuplicateServiceExceptions」
- 「JBoss Seam Debug Page エラーのデバッグと解決」

バグの報告

4.2.2. debug および Resolve ClassNotFoundExceptions および NoClassDefFoundErrors

概要

ClassNotFoundExceptions は通常、解決できない依存関係が原因で発生します。つまり、他のモジュールの依存関係を明示的に定義するか、外部ソースから JAR をコピーする必要があります。

1.

まず、不足している依存関係を確認してください。詳細は、以下を参照してください。「JBoss モジュール依存関係の検索」

2. 不足している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 を見つけます。詳細はを参照してください。 「以前のインストールで JAR を検索します。」

バグの報告

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/ ディレクトリーを参照して ClassNotFoundException によって指定されたクラスが含まれるモジュールを見つけてみてください。クラスのモジュールが見つかった場合は、依存関係をマニフェストエントリーに追加する必要があります。

たとえば、この ClassNotFoundException トレースがログに表示される場合は以下のようになります。

```
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)
```

以下を実行して、このクラスが含まれる JBoss モジュールを見つけます。

手順4.5 依存関係の検索

1. まず、クラスに明らかなモジュールが存在するかどうかを判断します。

   a. *EAP_HOME*/modules/system/layers/base/ ディレクトリーに移動し、ClassNotFoundException で名前が付けられたモジュールパス一致するクラスを探します。

      モジュールパス org/apache/commons/logging/ を見つけます。

   b. *EAP_HOME*/modules/system/layers/base/org/apache/commons/logging/main/module.xml ファイルを開き、モジュール名を見つけます。この場合、これは "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.
クラスに明らかなモジュールパスがない場合は、別の場所で依存関係を見つける必要がある場合があります。

a.
Tattletale レポートで ClassNotFoundException によって名前が付けられたクラスを検索します。

b.
EAP_HOME/modules ディレクトリーに JAR が含まれるモジュールを見つけ、前の手順でモジュール名を見つけます。

バグの報告

4.2.4. 以前のインストールで JAR を検索します。

クラスが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 にない場合、*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 を見つけます。

たとえば、この ClassNotFoundException トレースがログに表示される場合は以下のようになります。

```
Caused by: java.lang.NoClassDefFoundError: org/hibernate/validator/ClassValidator at
java.lang.Class.getDeclaredMethods0(Native Method)
```

以下のコマンドを実行して、このクラスが含まれる JAR を検索します。

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場合があります。この場合、以下の結果が必要です。

```
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/ ディレクトリーにコピーします。

多数の JAR が必要な場合は、クラスのモジュールの定義が簡単になります。詳細は、JBoss EAP 6『『開発ガイド』の「『スタートガイド』」の』章の「『モジュール』」を参照して https://access.redhat.com/documentation/ja-jp/red_hat_jboss_enterprise_application_platform/?version=6.4 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。

バグの報告

4.2.5. Debug and Resolve ClassCastExceptions

ClassCastExceptions は、拡張するクラスとは異なるクラスローダーによってクラスがロードされているため発生することがよくあります。また、複数の JAR に存在する同じクラスの結果になることもできます。

1. アプリケーションを検索し、ClassCastException によって名前が付けられたクラスが含まれるすべての JAR を検索します。クラスにモジュールが定義されている場合は、重複した JAR を見つけ、アプリケーションの WAR または EAR から削除します。

2. クラスが含まれる JBoss モジュールを見つけ、MANIFEST.MF ファイルまたは jboss-

deployment-structure.xml ファイルで依存関係を明示的に定義します。詳細は、JBoss EAP 6『の 『『開発ガイド』』 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 https://access.redhat.com/documentation/ja-jp/red_hat_jboss_enterprise_application_platform/?version=6.4 ください。

3. 上記の手順を使用して解決できない場合は、クラスローダー情報をログに出力して問題の原因を判断できます。たとえば、以下の ClassCastException がログに表示されます。

```
java.lang.ClassCastException: com.example1.CustomClass1 cannot be cast to
com.example2.CustomClass2
```

a. コードで、ClassCastException によって名前が付けられたクラスのクラスローダー情報をログに出力します。以下に例を示します。

```
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. ログの情報は、クラスをロードするモジュールを示し、アプリケーションに基づいて競合する JAR を削除または移動する必要があります。

バグの報告

4.2.6. デバッグおよび解決された DuplicateServiceExceptions

JBoss EAP 6 に EAR をデプロイするときに JAR のサブデプロイメントまたは WAR アプリケーションがすでにインストールされているメッセージに対して DuplicateServiceException が返されると、JBossWS がデプロイメントを処理する方法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、サーブレットベースのエンドポイントに新しい Context Root Mapping Algorithm が導入され、TCK6 とシームレスに互換性を持たせることができます。アプリケーション EAR アーカイブに同じ名前の WAR と JAR が含まれる場合、JBossWS は同じ名前の WAR コンテキストと Web コンテキストを作成できます。Web コンテキストは WAR コンテキストと競合し、これによりデプロイメントエラーが発生します。デプロイメントの問題を解決するには、以下のいずれかの方法で行います。

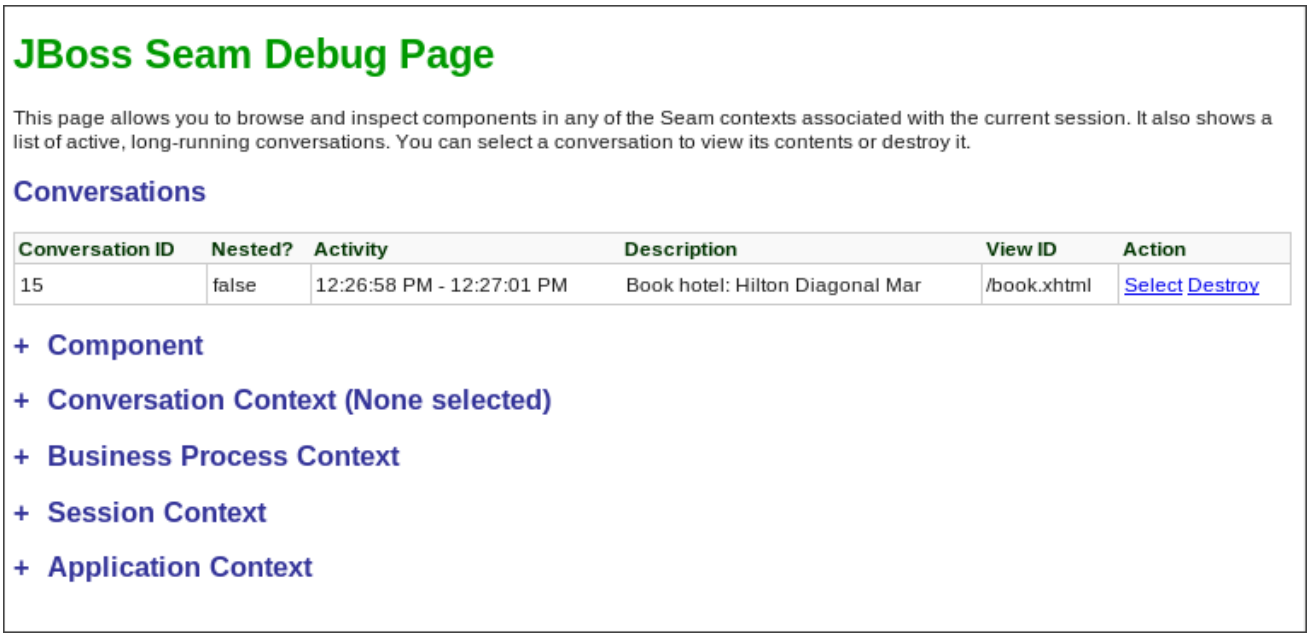
- JAR ファイルの名前を WAR 以外の名前に変更すると、生成された Web コンテキストと WAR コンテキストが一意となります。
- jboss-web.xml ファイルに <context-root> 要素を提供します。
- jboss-webservices.xml ファイルに <context-root> 要素を提供します。
- application.xml ファイルで WAR の <context-root> 要素をカスタマイズします。

バグの報告

4.2.7. JBoss Seam Debug Page エラーのデバッグと解決

アプリケーションを移行して正常にデプロイした後に、「JBoss Seam Debug」ページにリダイレクトするランタイムエラーが発生する可能性があります。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 コンポーネントを表示し、検査することができます。

図4.1 JBoss Seam Debug ページ



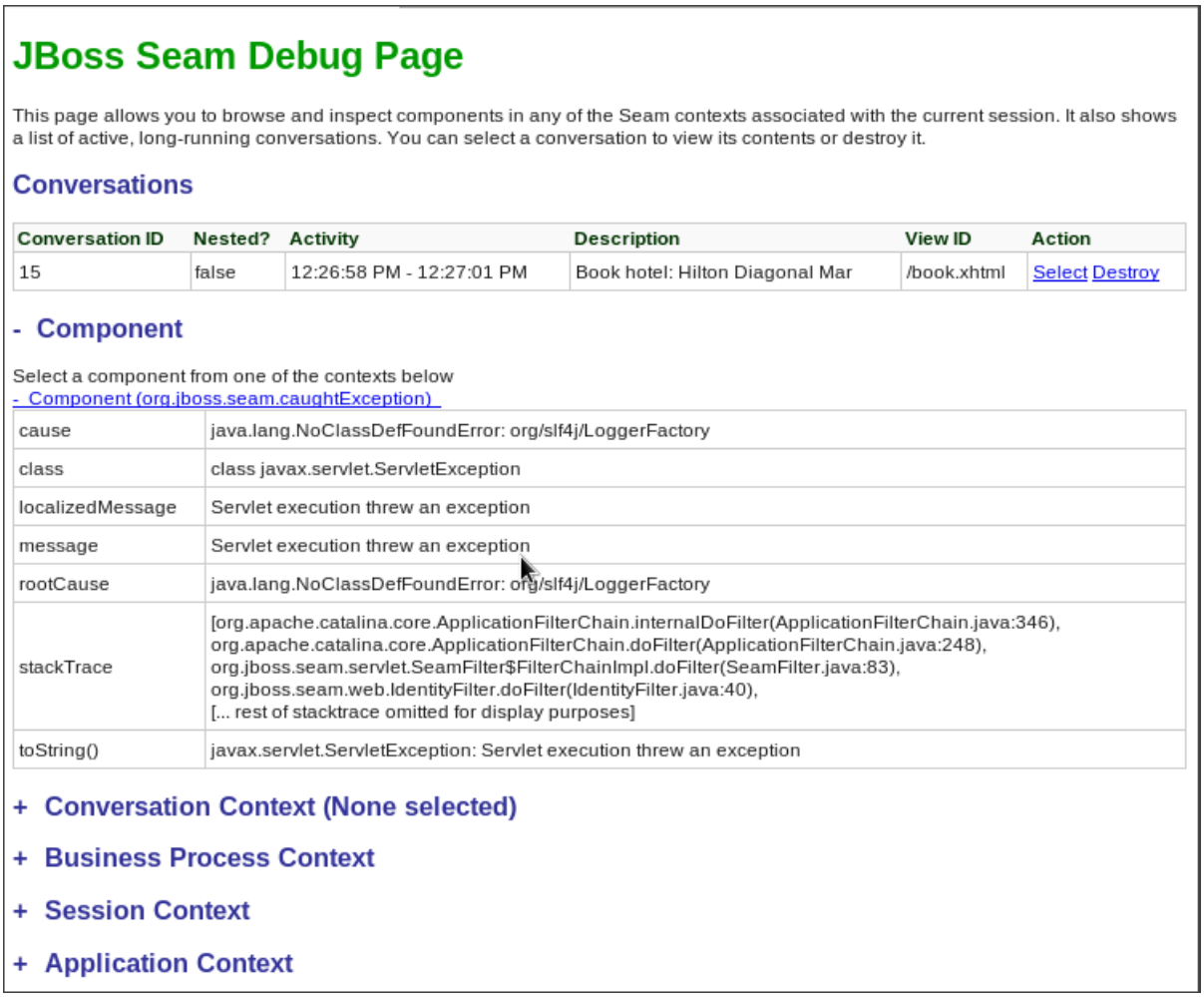
[D]

Seam はアプリケーションコードで処理されなかった例外をキャッチしたため、このページにリダイレクトされる可能性が最も高い理由です。例外の根本的な原因は、「JBoss Seam Debug Page」のリンクのいずれかにあります。

1. **ページの Component セクションを展開し、org.jboss.seam.caughtException コンポーネントを見つけます。**

2. **原因およびスタックトレースは、不足している依存関係を参照するはずです。**

図4.2 コンポーネントの org.jboss.seam.caughtException 情報



[D]

3. 「debug および Resolve ClassNotFoundExceptions および NoClassDefFoundErrors」 で説明されている技術を使用して、モジュール依存関係を解決します。

上記の例では、最もシンプルなソリューションは org.slf4j を MANIFEST.MFに追加することです。

```
Manifest-Version: 1.0
Dependencies: org.slf4j
```

または、モジュールの依存関係を jboss-deployment-structure.xml ファイルに追加します。

```
<jboss-deployment-structure>
  <deployment>
      <dependencies>
        <module name="org.slf4j" />
      </dependencies>
   </deployment>
</jboss-deployment-structure>
```

バグの報告

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 に移行した JBoss EAP 5.x サンプルアプリケーションのリストです。特定のアプリケーションで変更された内容の詳細を表示するには、以下のリンクをクリックします。

- 「Seam 2.2 JPA の例を JBoss EAP 6 に移行する」
- 「Seam 2.2 書籍の JBoss EAP 6 への移行」
- 「Seam 2.2 Booking アーカイブを JBoss EAP 6 に移行します： Step-By-Step Instructions」

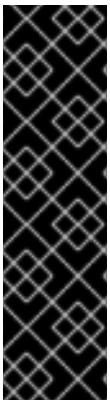
バグの報告

4.3.2. Seam 2.2 JPA の例を JBoss EAP 6 に移行する

概要

以下のタスク一覧は、jb 2.2 JPA サンプルアプリケーションを JBoss EAP 6 に正常に移行するた

めに必要な変更を要約しています。このサンプルアプリケーションは、EAP5. *x_HOME /jboss-eap-5.x/seam/examples/jpa/ の下にある最新の JBoss EAP*5 ディストリビューションにあります。



重要

Hibernate を Seam 2.2 で直接使用するアプリケーションは、アプリケーション内にパッケージ化された Hibernate 3 のバージョンを使用する場合があります。JBoss EAP 6 の org.hibernate モジュールで提供される Hibernate 4 は Seam 2.2 ではサポートされません。この例は、最初の手順としてアプリケーションを JBoss EAP 6 で実行できるようにするのに役立つことを目的としています。Seam 2.2 アプリケーションと共に Hibernate 3 をパッケージ化するオプション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 の例の移行

1. jboss-web.xml ファイルを削除します。

   jboss-seam-jpa.war/WEB-INF/ ディレクトリーから jboss-web.xml ファイルを削除します。jboss-web.xml で定義されたクラスローディングがデフォルトの動作になりました。

2. 以下のように jboss-seam-jpa.jar/META-INF/persistence.xml ファイルを変更します。

   a. jboss-seam-jpa.war/WEB-INF/classes/META-INF/persistence.xml ファイルの hibernate.cache.provider_class プロパティー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. provider モジュールプロパティーを 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 プロパティーを変更して、デフォルトの JDBC データソースの JNDI 名を使用します。

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

3. **Seam 2.2 の依存関係を追加します。**

以下の JAR を Seam 2.2 ディストリビューションライブラリー SEAM_HOME/lib/ から jboss-seam-jpa.war/WEB-INF/lib/ ディレクトリーにコピーします。

- **antlr.jar**
- **slf4j-api.jar**
- **slf4j-log4j12.jar**
- **hibernate-entitymanager.jar**
- **hibernate-core.jar**
- **hibernate-annotations.jar**
- **hibernate-commons-annotations.jar**
- **hibernate-validator.jar**

4. **jboss-deployment-structure ファイルを作成し、残りの依存関係を追加します。**

jboss-seam-jpa.war/WEB-INF/ フォルダーに以下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>
  <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 JPA サンプルアプリケーションは JBoss EAP 6 で正常にデプロイされ、実行されます。

バグの報告

4.3.3. Seam 2.2 書籍の JBoss EAP 6 への移行

概要

Seam 2.2 Book、EAR の移行は Seam 2.2 JPA WAR の例よりも複雑です。Seam 2.2 JPA WAR サンプル移行に関するドキュメントは、「Seam 2.2 JPA の例を JBoss EAP 6 に移行する」 を参照してください。アプリケーションを移行するには、以下を行う必要があります。

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 に同梱される旧バージョンを使用するのではなく、古いバージョンの Hibernate JAR をバンドルします。
3.

JNDI バインディングを変更して、新しい Java EE 6 の JNDI ポータブル構文を使用するようにします。

上記の最初の 2 つの手順は、jb 2.2 JPA WAR のサンプルで行われました。3 番目のステップは新しい手順であり、EAR には EJB が含まれるため必要になります。

重要

Hibernate を Seam 2.2 で直接使用するアプリケーションは、アプリケーション内にパッケージ化された Hibernate 3 のバージョンを使用する場合があります。JBoss EAP 6 の org.hibernate モジュールで提供される Hibernate 4 は Seam 2.2 ではサポートされません。この例は、最初の手順としてアプリケーションを JBoss EAP 6 で実行できるようにするのに役立つことを目的としています。Seam 2.2 アプリケーションと共に Hibernate 3 をパッケージ化するオプション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 の例の移行

1. jboss-deployment-structure.xml ファイルを作成します。

jboss-seam-booking.ear/META-INF/ に jboss-deployment-structure.xml という名前の新規ファイルを作成し、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"/>
       <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** ファイルを変更します。

   a. キャッシュプロバイダークラスの **hibernate** プロパティー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. **provider** モジュールプロパティーを **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** プロパティーを変更して、デフォルトの **JDBC** データソースの **JNDI** 名を使用します。

   ```
   <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から **JAR** をコピーします。

   Seam 2.2 ディストリビューション *EAP5.x_HOME /jboss-eap5.x*/seam/lib/ から **jboss-seam-booking.ear/lib** ディレクトリーに以下の **JAR** をコピーします。

   ```
   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します。

jboss-seam-booking.war/WEB-INF/components.xml ファイルで JNDI ルックアップ文字列を変更します。新しい JNDI ポータブルルールにより、JBoss EAP 6 は JNDI ポータブル構文ルールを使用して EJB をバインドし、JBoss EAP 5 で使用されていた単一の jndiPattern を使用できないようになりました。これは、アプリケーション EJB JNDI ルックアップ文字列を JBoss EAP 6 に変更する必要があるものです。

```
java:global/jboss-seam-booking/jboss-seam-
booking/HotelSearchingAction!org.jboss.seam.example.booking.HotelSearching
java:app/jboss-seam-
booking/HotelSearchingAction!org.jboss.seam.example.booking.HotelSearching
java:module/HotelSearchingAction!org.jboss.seam.example.booking.HotelSearching
java:global/jboss-seam-booking/jboss-seam-booking/HotelSearchingAction
java:app/jboss-seam-booking/HotelSearchingAction
java:module/HotelSearchingAction
```

Seam 2.2 フレームワークの JNDI ルックアップ文字列は、以下のように EJB を変更する必要があります。

```
java:global/jboss-seam-booking/jboss-
seam/EjbSynchronizations!org.jboss.seam.transaction.LocalEjbSynchronizations
java:app/jboss-
seam/EjbSynchronizations!org.jboss.seam.transaction.LocalEjbSynchronizations
java:module/EjbSynchronizations!org.jboss.seam.transaction.LocalEjbSynchronizatio
ns
java:global/jboss-seam-booking/jboss-seam/EjbSynchronizations
java:app/jboss-seam/EjbSynchronizations
java:module/EjbSynchronizations
```

以下の方法の 1 つを使用できます。

a. コンポーネント要素を追加します。

すべての EJB の 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.

コードを変更するには、JNDI パスを指定して @JNDIName(value="") アノテーションを追加します。変更されたステートレスセッション Bean コードの例を以下に示します。このプロセスの詳細な説明は、jb 2.2 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 Booking アプリケーションは JBoss EAP 6 で正常にデプロイされ、実行されます。

バグの報告

4.3.4. Seam 2.2 Booking アーカイブを JBoss EAP 6 に移行します： Step-By-Step Instructions

これは、JBoss EAP 5.X から JBoss EAP 6 への Seam 2.2 Bookるアプリケーションアーカイブの移植方法についての手順ごとに説明します。アプリケーションの移行には優れたアプローチがありますが、多くの開発者は、アプリケーションアーカイブを JBoss EAP 6 サーバーにデプロイして、何が起

こるかを確認することが望まれる可能性があります。本書の目的は、これらの問題のデバッグおよび解決方法により発生する可能性のある問題の種類を示しています。

この例では、アプリケーション EAR はアーカイブの抽出以外の変更なしで *EAP6_HOME*/standalone/deployments ディレクトリーにデプロイされます。これにより、問題が発生した場合にアーカイブに含まれる XML ファイルを簡単に変更し、解決することができます。

重要

Hibernate を Seam 2.2 で直接使用するアプリケーションは、アプリケーション内にパッケージ化された Hibernate 3 のバージョンを使用する場合があります。JBoss EAP 6 の org.hibernate モジュールで提供される Hibernate 4 は Seam 2.2 ではサポートされません。この例は、最初の手順としてアプリケーションを JBoss EAP 6 で実行できるようにするのに役立つことを目的としています。Seam 2.2 アプリケーションと共に Hibernate 3 をパッケージ化するオプション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とデプロイ」
2. 「アーカイブデプロイメントエラーと例外のデバッグおよび解決」
3. 「アーカイブランタイムエラーと例外をデバッグおよび解決」

この時点で、URL http://localhost:8080/seam-booking/ を使用してブラウザーでアプリケーションに正常にアクセスできます。demo/demo でログインすると、Booking welcome ページが表示されます。

変更の概要の確認

「Seam 2.2 Booking アプリケーションの移行時の変更の概要の確認」

バグの報告

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とデプロイ

このアプリケーションを移行する前に、**JBoss EAP 5.X Seam 2.2 Booking** アプリケーションをビルドし、アーカイブを展開して、**JBoss EAP 6** デプロイメントフォルダーにコピーする必要があります。

手順4.9 EAR のビルドおよびデプロイ

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、移行する JBoss EAP のバージョンに置き換えます。**

2. **EAR を *EAP6_HOME* deployments ディレクトリーにコピーします。**

```
$ 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、ログを確認します。以下が表示されます。**

```
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 という名前で空のファイルを作成し、*EAP6_HOME*/standalone/deployments ディレクトリーにコピーします。このアプリケーションの移行中にこのファイルを deployments ディレクトリーに複数回コピーする必要があります。これにより、簡単に見つけられる場所に維持する必要があります。ログに以下のメッセージが記録され、デプロイ中であることを示すメッセージが表示されます。**

```
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。次の手順では、各問題を実施し、デバッグして解決する方法を確認します。

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、こちらをクリックします。「アーカイブデプロイメントエラーと例外のデバッグおよび解決」

前のトピックに戻るには、ここをクリックしてください。「Seam 2.2 Booking アーカイブを JBoss EAP 6 に移行します： Step-By-Step Instructions」

バグの報告

4.3.6. アーカイブデプロイメントエラーと例外のデバッグおよび解決

前述の手順で、「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とデプロイ」 は JBoss EAP 5.X Seam 2.2 Booking アプリケーションをビルドし、JBoss EAP 6 デプロイメントフォルダーにデプロイします。このステップでは、発生した各デプロイメントエラーをデバッグし、解決します。

重要

Hibernate を Seam 2.2 で直接使用するアプリケーションは、アプリケーション内にパッケージ化された Hibernate 3 のバージョンを使用する場合があります。JBoss EAP 6 の org.hibernate モジュールで提供される Hibernate 4 は Seam 2.2 ではサポートされません。この例は、最初の手順としてアプリケーションを JBoss EAP 6 で実行できるようにするのに役立つことを目的としています。Seam 2.2 アプリケーションと共に Hibernate 3 をパッケージ化するオプション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と例外のデバッグおよび解決

1. Issue - java.lang.ClassNotFoundException: javax.faces.FacesException

アプリケーションをデプロイする場合、ログには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)
```

意味:

ClassNotFoundException は依存関係がないことを示します。この場合、javax.faces.FacesException クラス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:

足りないクラスに一致するパスを検索して、*EAP6_HOME*/modules/system/layers/base/ ディレクトリーでそのクラスのモジュール名を見つけます。この場合、以下に一致する 2 つのモジュールがあります。

```
javax/faces/api/main
javax/faces/api/1.2
```

両方のモジュールは同じモジュール名 javax.faces.api を持ちますが、メインディレクトリーの 1 つが JSF 2.0 用で、1.2 ディレクトリーにあるものは JSF 1.2 用です。利用可能なモジュールが 1 つしかない場合は、MANIFEST.MF ファイルを作成し、モジュール依存関係を追加するだけです。ただし、この場合は、main で 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場合は、1 つを指定してもう 1 つのバージョンを除外する必要があります。これには、以下のデータが含まれる 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 モジュールの javax.faces.api の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に JSF 1.2 モジュールの依存関係を追加

し、JSF 2.0 の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、空の jboss-seam-booking. ear.dodeploy ファイルを同じディレクトリーに作成してアプリケーションを再デプロイします。

2. Issue - 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、ログには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]
```

意味:

ClassNotFoundException は依存関係がないことを示します。この場合、org.apache.commons.logging.Log クラス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:

足りないクラスに一致するパスを検索して 、*EAP6_HOME*/modules/system/layers/base/ ディレクトリーでそのクラスのモジュール名を見つけます。この場合、org/apache/commons/logging/ パスに一致する 1 つのモジュールがあります。モジュール名は "org.apache.commons.logging" です。

jboss-deployment-structure.xml ファイルを変更して、モジュール依存関係をファイルの deployment セクションに追加します。

```
<module name="org.apache.commons.logging" export="true"/>
```

jboss-deployment-structure.xml 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"/>
     </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、空の jboss-seam-booking. ear.dodeploy ファイルを同じディレクトリーに作成してアプリケーションを再デプロイします。**

3. **issue - java.lang.ClassNotFoundException: org.dom4j.DocumentException**

アプリケーションをデプロイする場合、ログには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]
```

意味:

ClassNotFoundException は依存関係がないことを示します。この場合、org.dom4j.DocumentException クラスが見つかりません。

解決方法:

org/dom4j/DocumentException を参照して、*EAP6_HOME*/modules/system/layers/base/ディレクトリーでモジュール名を見つけます。モジュール名は "org.dom4j" です。jboss-deployment-structure.xml ファイルを変更して、モジュール依存関係をファイルの deployment セクションに追加します。

```
<module name="org.dom4j" export="true"/>
```

jboss-deployment-structure.xml ファイル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、空の jboss-seam-booking. ear.dodeploy ファイルを同じディレクトリーに作成してアプリケーションを再デプロイします。

4. Issue - java.lang.ClassNotFoundException: org.hibernate.validator.InvalidValue

アプリケーションをデプロイする場合、ログには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]
```

意味:

ClassNotFoundException は依存関係がないことを示します。この場合、org.hibernate.validator.InvalidValue クラスが見つかりません。

解決方法:

モジュール "org.hibernate.validator" がありますが、JAR には org.hibernate.validator.InvalidValue クラスが含まれないため、モジュール依存関係を追加してもこの問題は解決されません。この場合、クラスが含まれる JAR は JBoss EAP 5.X デプロイメントの一部でした。*EAP5_HOME/seam/lib/* ディレクトリーに不足しているクラスが含まれる JAR を検索します。これを実行するには、コンソールを開き、以下のコマンドを入力します。

```
$ 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、空の jboss-seam-booking. ear.dodeploy ファイルを同じディレクトリーに作成して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.でApplicationFactory**

アプリケーションをデプロイする場合、ログには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.verifyFactoriesExist(Factory
ConfigProcessor.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.java:606) [jsf-
api-1.2_13.jar:1.2_13-b01-FCS]
  (... additional logs removed ...)
  at
com.sun.faces.config.processor.FactoryConfigProcessor.verifyFactoriesExist(Factory
ConfigProcessor.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
```

意味:

com.sun.faces.config.ConfigurationException および java.lang.InstantiationException は、依存関係の問題を示します。この場合、原因は明確ではありません。

解決方法:

com.sun.faces クラスが含まれるモジュールを見つける必要があります。com.sun.faces モジュールがありませんが、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 を簡単にチェックすると、com.sun.faces クラスが含まれていることが分かります。javax.faces.FacesException ClassNotFoundException では、メインの JSF 2.0 バージョンではなく JSF 1.2 バージョンを使用する必要があるため、1 つを指定して別のバージョンを除外する必要があります。jboss-deployment-structure.xml を変更して、モジュールの依存関係をファイルの deployment セクションに追加します。また、WAR 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があります。ファイル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"/>
    </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、空の jboss-seam-booking. ear.dodeploy ファイルを同じディレクトリーに作成してアプリケーションを再デプロイします。

6.

Issue - 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、ログには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]
```

意味:

ClassNotFoundException は依存関係がないことを示します。この場合、org.apache.commons.collections.ArrayStack クラスが見つかりません。

解決方法:

org/apache/commons /collections パスを確認して、*EAP6_HOME*/modules/system/layers/base / ディレクトリーでモジュール名を見つけま

す。モジュール名は "org.apache.commons.collections" です。jboss-deployment-structure.xml を変更し、モジュール依存関係をファイルの deployment セクションに追加します。

```
<module name="org.apache.commons.collections" export="true"/>
```

jboss-deployment-structure.xml ファイル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、空の jboss-seam-booking. ear.dodeploy ファイルを同じディレクトリーに作成してアプリケーションを再デプロイします。**

7.

問題： 利用できない依存関係/利用できない依存関係のあるサービス

アプリケーションをデプロイする場合、ログには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.booking.Authenticato
```

```
rAction/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.booking.HotelSearc
hingAction/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.BookingListA
ction/em\" ]","jboss.persistenceunit.\"jboss-seam-booking.ear/jboss-seam-
booking.jar#bookingDatabase\" missing [
jboss.naming.context.java.bookingDatasource ]"]}}}
```

意味:

"Services with missing/unavailable dependencies" エラーが出たら、「missing」後の括弧内のテキストを確認してください。この場合は、以下を確認できます。

```
missing [ jboss.naming.context.java.comp.jboss-seam-booking.\"jboss-seam-
booking.jar\".AuthenticatorAction.\"env/org.jboss.seam.example.booking.AuthenticatorAction/em
\" ]
```

"/em" はエンティティーマネージャーおよびデータソースの問題を示します。

解決方法:

JBoss EAP 6 ではデータソース設定が変更になり、*EAP6_HOME*/standalone/configuration/standalone.xml ファイルで定義する必要があります。JBoss EAP 6 には standalone.xml ファイルにすでに定義されているデータベースのサンプルが含まれるため、persistence.xml ファイルを変更して、このアプリケーションでこのサンプルデータベースを使用します。standalone.xml ファイルを見ると、サンプルデータベースの jndi-name が java:jboss/datasources/ExampleDS であることを確認できます。jboss-seam-booking.jar/META-INF/persistence.xml ファイルを変更して、既存の jta-data-source 要素をコメント化し、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、空の jboss-seam-booking. ear.dodeploy ファイルを同じディレクトリーに作成してアプリケーションを再デプロイします。**

8.

この時点で、アプリケーションはエラーなしでデプロイされますが、ブラウザーで URL http://localhost:8080/seam-booking/ にアクセスし、「Account Login」を試みると、「The page doesn't redirecting properly」というランタイムエラーが表示されます。次の手順では、ランタイムエラーをデバッグし、解決する方法を説明します。

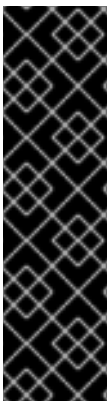
ランタイムの問題をデバッグおよび解決する方法については、こちらをクリックします。「アーカイブランタイムエラーと例外をデバッグおよび解決」

前のトピックに戻るには、ここをクリックしてください。「Seam 2.2 Booking アーカイブを JBoss EAP 6 に移行します： Step-By-Step Instructions」

バグの報告

4.3.7. アーカイブランタイムエラーと例外をデバッグおよび解決

前の手順で、「アーカイブデプロイメントエラーと例外のデバッグおよび解決」でデプロイメントエラーのデバッグ方法を説明しました。このステップでは、発生した各ランタイムエラーをデバッグして解決します。



重要

Hibernate を Seam 2.2 で直接使用するアプリケーションは、アプリケーション内にパッケージ化された Hibernate 3 のバージョンを使用する場合があります。JBoss EAP 6 の org.hibernate モジュールで提供される Hibernate 4 は Seam 2.2 ではサポートされません。この例は、最初の手順としてアプリケーションを JBoss EAP 6 で実行できるようにするのに役立つことを目的としています。Seam 2.2 アプリケーションと共に Hibernate 3 をパッケージ化するオプションはサポートされないことに注意してください。

手順4.11 ランタイムエラーと例外のデバッグおよび解決

この時点で、アプリケーションをデプロイするとログにエラーが表示されない。ただし、アプリケーション URL にアクセスすると、エラーがログに表示されます。

1.

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

ブラウザーで URL http://localhost:8080/seam-booking/ にアクセスすると、「The page doesn't redirecting correctly」が表示され、ログに以下のエラーが表示されます。

```
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(NamingUtils.java:109)
  (... log messages removed ...)
```

意味:

NameNotFoundException は JNDI 命名の問題を示します。JBoss EAP 6 では JNDI の命名規則が変更されたため、ルックアップ名を変更して新しいルールに従う必要があります。

解決方法:

これをデバッグするには、サーバーログトレースで以前使用された JNDI バインディングを確認します。以下のサーバーログを確認してください。

```
15:01:16,138 INFO
[org.jboss.as.ejb3.deployment.processors.EjbJndiBindingsDeploymentUnitProcessor] (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 ...]
```

ログには合計 8 INFO の JNDI バインディングがリストされています。各セッション Bean には、RegisterAction、BookingListAction、HotelBookingAction、AuthenticatorAction、ChangePasswordAction、HotelSearchingAction、EjbSynchronizations、および TimerServiceDispatcher があります。新しい JNDI バイン

ディングを使用するには、WAR の lib/components.xml ファイルを変更する必要があります。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 および TimerServiceDispatcher JNDI バインディングを追加する必要があります。以下のコンポーネント要素をファイルに追加します。

```
<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 ファイルは以下のようになります。

```
<?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、空の jboss-seam-booking.ear.dodeploy ファイルを同じディレクトリーに作成してアプリケーションを再デプロイします。

2. 問題： アプリケーションはエラーを出さずにデプロイおよび実行を行います。ブラウザーで 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」というメッセージが表示されて失敗します。トランザクションに失敗しました。" サーバーログに例外トレースが表示されるはずです。

```
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]
...
```

意味:

ClassNotFoundException は Hibernate ライブラリーがないことを示します。この場合、hibernate-core.jar になります。

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 ディレクトリーにコピーします。

standalone/deployments/jboss-seam-booking.ear.failed ファイルを削除し、空の jboss-seam-booking.ear.dodeploy ファイルを同じディレクトリーに作成してアプリケーションを再デプロイします。

3. 問題： アプリケーションはエラーを出さずにデプロイおよび実行を行います。ブラウザー

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。ただし、ホテルを予約しようとすると、例外トレースが表示されます。

これをデバッグするには、まず true エラーをマスクするために jboss-seam-booking.ear/jboss-seam-booking.war/WEB-INF/lib/jboss-seam-debug.jar を削除する必要があります。この時点で、以下のエラーが表示されます。

```
java.lang.NoClassDefFoundError:
org/hibernate/annotations/common/reflection/ReflectionManager
```

意味:

NoClassDefFoundError は Hibernate ライブラリーがないことを示します。

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 ディレクトリーにコピーします。

standalone/deployments/jboss-seam-booking.ear.failed ファイルを削除し、空の jboss-seam-booking.ear.dodeploy ファイルを同じディレクトリーに作成してアプリケーションを再デプロイします。

4. ランタイムエラーとアプリケーションのエラーは解決されるべき

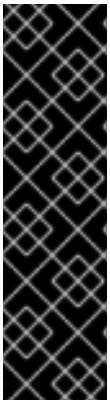
この時点で、アプリケーションはエラーなしでデプロイおよび実行を行います。

前のトピックに戻るには、ここをクリックしてください。 「Seam 2.2 Booking アーカイブを JBoss EAP 6 に移行します： Step-By-Step Instructions」

バグの報告

4.3.8. Seam 2.2 Booking アプリケーションの移行時の変更の概要の確認

依存関係を前もって判断し、1 つのステップで暗黙的な依存関係を追加する方がより効率的ですが、この演習では、問題がログにどのように表示されるかを示し、デバッグして解決する方法に関する情報を提供します。以下は、JBoss EAP 6 へ移行するときにアプリケーションに加えられた変更の概要です。



重要

Hibernate を Seam 2.2 で直接使用するアプリケーションは、アプリケーション内にパッケージ化された Hibernate 3 のバージョンを使用する場合があります。JBoss EAP 6 の org.hibernate モジュールで提供される Hibernate 4 は Seam 2.2 ではサポートされません。この例は、最初の手順としてアプリケーションを JBoss EAP 6 で実行できるようにするのに役立つことを目的としています。Seam 2.2 アプリケーションと共に Hibernate 3 をパッケージ化するオプションはサポートされないことに注意してください。

1. EAR の META-INF/ ディレクトリーに jboss-deployment-structure.xml ファイルが作成されている。<dependencies> を解決するために、<exclusions> および ClassNotFoundExceptions を追加している。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. 以下の JAR を *EAP5_HOME*/jboss-eap-5.X/seam/lib/ ディレクトリー（移行元の EAP 5

のバージョンに置き換えます）から jboss-seam-booking.ear/lib/ ディレクトリーにコピーされ、ClassNotFoundExceptions を解決します。

- hibernate-core.jar
- hibernate-validator.jar

3. jboss-seam-booking.jar/META-INF/persistence.xml ファイルを以下のように変更している。

   a. JBoss EAP 6 に同梱される Example データベースを使用するよう jta-data-source 要素を変更し、以下を行います。

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

   b. hibernate.cache.provider_class プロパティーをコメントアウトしている。

```
<!-- <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 ファイルを修正している。

   a. 以下のように core:init 既存の要素を置き換えている。

```
<!-- <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"/>
```

b.
「EjbSynchronizations」および「TimerServiceDispatcher」の 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"/>
```

バグの報告

付録A 改訂履歴

改訂 6.4.0-42 **Thursday November 16 2017** **Red Hat Customer Content Services**
Red Hat JBoss Enterprise Application Platform 6.4.0.GA Continuous Release