Red Hat JBoss Enterprise Application Platform 8.0

移行ガイド

Red Hat JBoss Enterprise Application Platform のメジャーバージョンへのアップグレード手順

Last Updated: 2024-03-28

Red Hat JBoss Enterprise Application Platform 8.0 移行ガイド

Red Hat JBoss Enterprise Application Platform のメジャーバージョンへのアップグレード手順

法律上の通知

概要

このガイドでは、Red Hat JBoss Enterprise Application Platform の以前のバージョンから移行する方法について説明します。

目次

JBOSS EAP ドキュメントへのフィードバック (英語のみ)

エラーを報告したり、ドキュメントを改善したりするには、Red Hat Jira アカウントにログインし、課題を送信してください。Red Hat Jira アカウントをお持ちでない場合は、アカウントを作成するように求められます。

手順

1. **このリンクをクリック** してチケットを作成します。
2. **Summary** に課題の簡単な説明を入力します。
3. **Description** に課題や機能拡張の詳細な説明を入力します。問題があるドキュメントのセクションへの URL を含めてください。
4. **Submit** をクリックすると、課題が作成され、適切なドキュメントチームに転送されます。

多様性を受け入れるオープンソースの強化

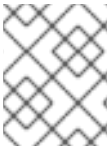
Red Hat では、コード、ドキュメント、Web プロパティーにおける配慮に欠ける用語の置き換えに取り組んでいます。まずは、マスター (master)、スレーブ (slave)、ブラックリスト (blacklist)、ホワイトリスト (whitelist) の 4 つの用語の置き換えから始めます。この取り組みは膨大な作業を要するため、今後の複数のリリースで段階的に用語の置き換えを実施して参ります。詳細は、Red Hat CTO である Chris Wright のメッセージ をご覧ください。

第1章 RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM 移行の概要

システム管理者は、Red Hat JBoss Enterprise Application Platform 7 から Red Hat JBoss Enterprise Application Platform 8.0 にアップグレードできます。このガイドでは、リリースで利用可能な新機能、非推奨およびサポート対象外となった機能、一貫した動作を維持するために必要なアプリケーションおよびサーバー設定の更新について説明します。

また、Java アプリケーションの移行を簡単にする Migration Toolkit for Runtimes や、サーバー設定を更新する JBoss Server Migration Tool など、移行に役立つツールに関する情報も提供します。

JBoss EAP 8.0 を正常にデプロイして実行した後、JBoss EAP 8.0 の新しい機能を使用するように個々のコンポーネントをアップグレードできます。



注記

JBoss EAP の古いリリースから移行する場合は、まず JBoss EAP 7.4 に移行する必要があります。詳細は、JBoss EAP 7.4 移行ガイド を参照してください。

1.1. 移行およびアップグレードについて

このセクションでは、メジャーアップグレード、マイナーアップデート、累積パッチなど、JBoss EAP のアップグレードおよびパッチ適用について説明します。

1.1.1. JBoss EAP のメジャーアップグレード

アプリケーションをあるメジャーリリースから別のメジャーリリース (JBoss EAP 7 から JBoss EAP 8.0 など) に移動した場合は、メジャーアップグレードまたは移行が必要です。Jakarta EE 8 仕様に準拠しているアプリケーション、独自のコードが含まれているアプリケーション、または非推奨の API を使用しているアプリケーションを JBoss EAP 8.0 で実行するには、アプリケーションコードの変更が必要になる場合があります。Jakarta EE 10 の導入には、Java パッケージ名やその他の側面の変更が含まれており、JBoss EAP 8.0 との互換性を確保するために Jakarta EE アプリケーションコードの調整が必要になります。さらに、JBoss EAP 8.0 ではサーバー設定が変更されたため、移行が必要です。本書ではこのような移行を取り上げます。

1.1.2. JBoss EAP のマイナーアップデート

JBoss EAP のマイナーアップデートは、バグ修正、セキュリティー修正、および新機能を提供するポイントリリースです。各ポイントリリースで行われた変更は、Red Hat JBoss Enterprise Application Platform 8.0 のリリースノート に記載されています。

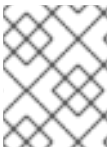
JBoss Server Migration Tool を使用して、あるポイントリリースから別のポイントリリースに自動的にアップグレードします (たとえば、JBoss EAP 7.0 から JBoss EAP 7.1 など)。詳細は、JBoss Server Migration Tool の使用 を参照してください。

1.1.3. JBoss EAP の累積パッチ

JBoss EAP は、バグ修正とセキュリティー修正を含む累積パッチを定期的に提供します。累積パッチのリリースでは、リリース番号の最後の数字が 1 ずつ増えます (例: バージョン 7.1.0 から 7.1.1 へのアップグレードなど)。

1.2. <EAP_HOME> 変数の使用

このドキュメントでは、**<EAP_HOME>** 変数は JBoss EAP インストールへのパスを示します。この変数は JBoss EAP インストールへの実際のパスに置き換えてください。

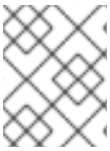


注記

<EAP_HOME> は環境変数ではありません。スクリプトで **JBOSS_HOME** 環境変数を使用します。

JBoss EAP のインストールの方法に応じて、以下のようにインストールディレクトリーまたはデフォルトのパスの場所を特定できます。

- アーカイブインストール方法を使用して JBoss EAP をインストールした場合、インストールディレクトリーはアーカイブを抽出した **jboss-eap-8.0** ディレクトリーになります。
- RPM インストール方法を使用して JBoss EAP をインストールした場合、インストールディレクトリーは **/opt/rh/eap8/root/usr/share/wildfly/** になります。
- インストーラーアプリケーションを使用して JBoss EAP をインストールした場合、**<EAP_HOME>** のデフォルトのパスは **${user.home}/EAP-8.0.0** です。
  - Red Hat Enterprise Linux および Oracle Solaris の場合: **/home/USER_NAME/EAP-8.0.0/**
  - Microsoft Windows の場合: **C:\Users\USER_NAME\EAP-8.0.0**
- Red Hat CodeReady Studio インストーラーアプリケーションを使用して JBoss EAP サーバーをインストールおよび設定した場合、**<EAP_HOME>** のデフォルトのパスは **${user.home}/devstudio/runtimes/jboss-eap** です。
  - Red Hat Enterprise Linux の場合、**/home/USER_NAME/devstudio/runtimes/jboss-eap/** になります。
  - Microsoft Windows の場合、**C:\Users\USER_NAME\devstudio\runtimes\jboss-eap** または **C:\Documents and Settings\USER_NAME\devstudio\runtimes\jboss-eap** になります。



注記

Red Hat CodeReady Studio で **ターゲットランタイム** を 8.0 以降のランタイムバージョンに設定すると、プロジェクトの Jakarta EE 10 仕様との互換性が確保されます。

第2章 JBOSS EAP 8.0 への移行の準備

システム管理者は、JBoss EAP 8.0 への移行を計画する必要があります。このアップグレードは、Java アプリケーションのパフォーマンスの向上、セキュリティーの強化、および安定性の向上に不可欠です。

JBoss EAP 8.0 は、JBoss EAP 7 アプリケーションに対して下位互換性を提供します。ただし、JBoss EAP 8.0 で非推奨または削除された機能をアプリケーションが使用している場合は、アプリケーションコードの変更が必要になる場合があります。

JBoss EAP 8.0 リリースでは、アプリケーションのデプロイメントに影響を与える可能性のあるいくつかの変更が導入されています。移行を確実に成功させるには、アプリケーションの移行を試みる前に調査と計画を行ってください。

移行プロセスを開始する前に、以下の最初の手順を実行します。

- Jakarta EE 10 の機能 を理解します。
- JBoss EAP 8.0 の機能 を確認します。
- JBoss EAP スタートガイド を確認します。
- データをバックアップし、サーバーの状態を確認する ことで、シームレスな移行プロセスを確保します。
- RPM インストールを使用して JBoss EAP を移行する ことで、インストールプロセスを合理化します。
- JBoss EAP をサービスとして移行する ことで、管理性と自動化を改善します。

機能の変更、開発ガイド、および移行作業を支援できるツールについて理解したら、アプリケーションとサーバー設定を評価して、JBoss EAP 8.0 で実行するために必要な変更を決定します。

2.1. JAKARTA EE 10 機能の確認

Jakarta EE 10 には、プライベートクラウドとパブリッククラウドの両方での機能豊富なアプリケーションの開発とデプロイメントを簡素化する多数の機能強化が導入されています。JBoss EE 7 は、HTML5、WebSocket、JSON、Batch、および Cocurrency Utilities などの新機能や最新の標準が導入されています。更新に含まれるのは、次のとおりです: Jakarta Persistence 3.1、Jakarta RESTful Web Services 3.1、Jakarta Servlet 6.0、Jakarta Expression Language 5.0、Java Message Service 3.1、Jakarta Server Faces 4.0、Jakarta Enterprise Beans 4.0、Contexts and Dependency Injection 2.0、および Jakarta Bean Validation 3.0。

関連情報

- Jakarta EE Platform 10

2.2. JBOSS EAP 8.0 の機能の確認

JBoss EAP 8.0 には、以前のリリースに対するアップグレードと改善が含まれています。JBoss EAP 8.0 で導入された新機能の完全なリストについては、Red Hat カスタマーポータルの **Red Hat JBoss Enterprise Application Platform 8.0 のリリースノート** で、新機能と機能拡張 を参照してください。

アプリケーションを JBoss EAP 8.0 に移行する前に、高いメンテナンスコスト、コミュニティーの関心の低さ、またはより優れた代替手段の可用性により、以前のリリースの一部の機能がサポートされなく

なったり、非推奨になったりする可能性があることに注意してください。JBoss EAP 8.0 の非推奨およびサポートされていない機能の完全なリストについては、Red Hat カスタマーポータルの **Red Hat JBoss Enterprise Application Platform 8.0 のリリースノート** で、サポートされない機能、非推奨の機能、削除された機能 を参照してください。

2.3. JBOSS EAP スタートガイドのドキュメントの確認

このセクションでは、JBoss EAP スタートガイドの主要コンポーネントについて説明し、JBoss EAP を使い始める上で役立つ重要な情報の簡潔な概要を提供します。

以下に関する重要な情報については、JBoss EAP スタートガイド を参照してください。

- JBoss EAP 8.0 のダウンロードとインストール、および効果的な環境セットアップ。
- 開発環境の改善を目的とした JBoss Tools のダウンロードとインストール。

重要

JBoss Tools はコミュニティープロジェクトであり、Red Hat ではサポートされていません。JBoss Tools のインスタンスのセットアップと実行については、コミュニティー Web サイト を参照してください。JBoss Tools をダウンロードするには、JBoss Tools Downloads を参照してください。

- 開発環境に合わせた Maven の設定とプロジェクトの依存関係の管理。
- 製品に付属する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と実行。

関連情報

- JBoss EAP を使用したアプリケーションの開発

2.4. データをバックアップし、サーバー状態を確認する

このセクションでは、アプリケーションを移行する前に、データをバックアップし、サーバーの状態を確認して、潜在的な問題に対処する必要があることを強調しています。デプロイメントを保護し、オープンなトランザクションを管理して、タイマーデータを評価することで、スムーズな移行を確実に実行できます。

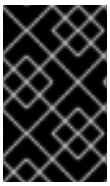
移行を開始する前に、次の潜在的な問題を考慮してください。

- 移行プロセスにより、一時フォルダーが削除される場合があります。移行する前に、**data/content/** ディレクトリー内のデプロイメントを必ずバックアップしてください。コンテンツの欠落によるサーバー障害を避けるために、完了後にデータを復元します。
- 移行前に、オープントランザクションを処理し、**data/tx-object-store/** transaction ディレクトリーを削除します。
- 移行を進める前に **data/timer-service-data** 内の永続タイマーデータを確認し、アップグレード後の適用性を判断します。アップグレードする前に、そのディレクトリー内の **deployment-*** ファイルをチェックして、引き続き使用されているタイマーを特定します。

移行を開始する前に、現在のサーバー設定とアプリケーションを必ずバックアップしてください。

2.5. RPM インストールを使用して JBOSS EAP を移行する

このガイドの移行に関するアドバイスは JBoss EAP の RPM インストールの移行にも当てはまりますが、アーカイブや **jboss-eap-installation-manager** インストールと比較した場合の、RPM インストールに合わせて JBoss EAP を起動する方法など、一部の手順を変更する必要があるかもしれません。



重要

単一の Red Hat Enterprise Linux サーバーでサポートされるのは、RPM でインストールされた JBoss EAP のインスタンス 1 つまでです。したがって、JBoss EAP 8.0 に移行する場合は、JBoss EAP インストールを新しいマシンに移行することを推奨します。

関連情報

- RPM インストール方法を使用した JBoss EAP のインストール

2.6. JBOSS EAP をサービスとして移行する

JBoss EAP 7 をサービスとして実行する場合は、Red Hat JBoss Enterprise Application Platform のインストール方法 で JBoss EAP 8.0 の更新された設定手順を確認してください。

2.7. クラスターを移行する

JBoss EAP クラスターを実行する場合は、JBoss EAP 7.4 **パッチ適用およびアップグレードガイド** の クラスターのアップグレード セクションに記載されている指示に従ってください。

第3章 効果的なツールを使用した JBOSS EAP 8.0 への移行の簡素化

システム管理者は、2 つの必須ツールを利用して JBoss EAP 8.0 への移行プロセスを簡素化できます。Migration Toolkit for Runtime (MTR) はアプリケーションを分析し、詳細な移行レポートを提供します。一方、JBoss Server Migration Tool はサーバー設定を更新して新しい機能と設定を含めます。

3.1. 移行前のアプリケーションの分析

Migration Toolkit for Runtimes MTR を使用して、JBoss EAP 6.4 および 7 アプリケーションを JBoss EAP 8.0 に移行する前に、そのコードとアーキテクチャーを分析できます。JBoss EAP 8.0 への移行用の MTR ルールセットは、JBoss EAP 8.0 への移行時に代替設定に置き換える必要がある XML 記述子、特定のアプリケーションコード、およびパラメーターに関するレポートを提供します。

MTR は、Java アプリケーションの移行を簡素化するのに役立つ、拡張可能でカスタマイズ可能なルールベースのツールセットです。MTR は、移行を計画しているアプリケーションで使用される API、テクノロジー、アーキテクチャーを分析し、各アプリケーションの詳細な移行レポートを提供します。レポートには以下の情報が含まれます。

- 必要な移行に関する変更の詳細な説明
- 変更が必須または任意であるかどうか
- 変更が複雑または簡単であるかどうか
- 移行変更が必要なコードへのリンク
- 必要な変更を行うためのヒントおよび情報へのリンク
- 見つかった各移行問題の推定作業量レベルおよびアプリケーションを移行するための推定合計作業量

関連情報

- Migration Toolkit for Runtimes

3.2. サーバー設定の移行の簡素化

JBoss Server Migration Tool は、既存の設定を維持しながらサーバー設定を更新して JBoss EAP 8.0 の新しい機能と設定を含める場合に推奨される方法です。JBoss Server Migration Tool は既存の JBoss EAP 6 サーバー設定ファイルを読み取り、新しいサブシステムの設定を追加します。 さらに、既存のサブシステム設定を新機能で更新し、古いサブシステム設定を削除します。

JBoss Server Migration Tool を使用して、スタンドアロンサーバーを移行し、ドメインを管理できます。

3.2.1. JBoss EAP 8.0 への移行

JBoss Server Migration Tool は、JBoss EAP バージョン 7 のすべてのリリースから JBoss EAP 8.0 への移行をサポートします。

注記

JBoss EAP 6.4 から移行する場合は、まずは最新の JBoss EAP 7.4 の累積パッチ (CP) バージョンに移行する必要があります。詳細は、JBoss EAP 7.4 移行ガイド を参照してください。その後、JBoss EAP 7.4 の CP バージョンから JBoss EAP 8.0 に移行できます。

前提条件

- JBoss EAP が実行されていない。

手順

1. JBoss EAP ダウンロードページ からツールをダウンロードします。
2. ダウンロードしたアーカイブをデプロイメントします。

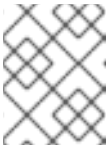
```
$ unzip <NAME_OF_THE_FILE>
```

3. **MIGRATION_TOOL_HOME/bin** ディレクトリーに移動します。
4. **jboss-server-migration** スクリプトを実行します。
   - Red Hat Enterprise Linux の場合は以下のコマンドを実行します。

```
$ ./jboss-server-migration.sh --source EAP_PREVIOUS_HOME --target
EAP_NEW_HOME
```

   - Microsoft Windows の場合

```
jboss-server-migration.bat --source EAP_PREVIOUS_HOME --target EAP_NEW_HOME
```



注記

EAP_PREVIOUS_HOME と **EAP_NEW_HOME** を、JBoss EAP の以前のインストールと新しいインストールへの実際のパスに置き換えます。

関連情報

- JBoss サーバー移行ツールの使用

第4章 RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM アプリケーションを JAKARTA EE 8 から 10 に移行する

JBoss EAP 8.0 は Jakarta EE 10 のサポートを提供します。Jakarta EE 10 は、JBoss EAP 7 でサポートされる Jakarta EE 8 仕様と比較して、Jakarta EE に大きな変更をもたらします。この章では、アプリケーション開発者がアプリケーションを JBoss EAP 7 から JBoss EAP 8.0 に移行する準備をする際に認識しておく必要がある、互換性に影響を与える Jakarta EE API の相違点について説明します。

注記

この章では、移行方法ではなく、アプリケーションを JBoss EAP 8.0 に移行するアプリケーション開発者が対処する必要があるかもしれない Jakarta EE 8 と Jakarta EE 10 の相違点に焦点を当てています。アプリケーションを JBoss EAP 7 から JBoss EAP 8.0 に移行する場合の詳細と、それを支援するために Red Hat が提供するツールについては、効果的なツールを使用した JBoss EAP 8.0 への移行の簡素化 および アプリケーションの移行に関する変更点 を参照してください。

4.1. JAVAX から JAKARTA パッケージ名前空間への変更

Jakarta EE 8 と EE 10 の互換性に影響を与える最大の違いは、EE API Java パッケージの名前が **javax.*** から **jakarta.*** に変更されたことです。

Java EE が Eclipse Foundation に移行し、Jakarta EE が確立された後、Eclipse と Oracle は、Jakarta EE コミュニティーが **javax.** パッケージ名前空間に変更を加えられない点で合意しました。したがって、引き続き EE API に変更を加えるために、Jakarta EE 9 以降、すべての EE API に使用されるパッケージが **javax.*** から **jakarta.*** に変更されました。この変更は、Java SE に含まれる javax パッケージには影響しません。

この名前空間の変更への適応は、アプリケーションを JBoss EAP 7 から JBoss EAP 8 に移行する際に伴う最大の変更です。Jakarta EE 10 に移行するアプリケーションは、以下を行う必要があります。

- EE API クラスの import ステートメントまたはその他のソースコードの使用を **javax** パッケージから **jakarta** に更新します。
- 名前が **javax.** で始まる EE 指定のシステムプロパティーまたはその他の設定プロパティーの名前を **jakarta.** で始まる名前に更新します。
- **java.util.ServiceLoader** メカニズムを使用してブートストラップされる EE インターフェイスまたは抽象クラスのアプリケーション提供の実装がある場合は、実装クラスを識別するリソースの名前を **META-INF/services/javax.[rest_of_name]** から **META-INF/services/jakarta.[rest_of_name]** に変更します。

注記

Red Hat Migration Toolkit を使用すると、アプリケーションソースコード内の名前空間の更新が容易になります。詳細は、How to use Red Hat Migration Toolkit for Auto-Migration of an Application to the Jakarta EE 10 Namespace を参照してください。ソースコードの移行がオプションではない場合、オープンソースの Eclipse Transformer プロジェクトは、既存の Java アーカイブを javax 名前空間から jakarta に変換するバイトコード変換ツールを提供します。

4.2. その他の変更点

パッケージの名前空間の変更に加えて、以前の EE バージョン用に作成されたアプリケーションは、Jakarta EE 10 に含まれる多くの仕様に加えられた変更に適応する必要がある場合があります。次のセクションでは、これらの変更について説明します。これらの変更の大部分は、長い間非推奨となっていた API 要素の削除です。

以下のセクションでは、**javax** 名前空間を使用する削除された API 要素のインスタンスについて、Jakarta EE 9 で使用される **jakarta** 名前空間で同等の削除が行われています。したがって、アプリケーションを更新して **javax** 名前空間を **jakarta** に置き換えた場合は、**javax** について言及している項目がアプリケーションに適用できると想定してください。

4.2.1. Jakarta Contexts and Dependency Injection Bean の検出

CDI 4.0 仕様変更ノート に従って、空の **beans.xml** ファイルを使用したデプロイメントで Contexts and Dependency Injection (CDI) Bean を検出するためのデフォルトの動作が、**all** から **annotated** に変更されました。これは、そのようなデプロイメントの場合、Bean 定義アノテーション を持つデプロイメントクラスのみが、CDI によって検出されることを意味します。Bean を使用するすべてのアプリケーションクラスにそのようなアノテーションがある場合、この CDI の変更は影響しません。そうでない場合は、CDI が特定の Bean を提供する型を見つけられず、アプリケーションのデプロイメントが失敗する可能性があります。

アプリケーションがこの変更の影響を受ける場合、いくつかのオプションがあります。

- **beans.xml** ファイルは空のままにしておきますが、Bean 定義アノテーションを必要とするすべてのクラスに追加します。
- クラスを変更しないままにしますが、**beans.xml** ファイルを空の状態から次の内容を含むファイルに変更します: **<beans bean-discovery-mode="all"></beans>**
- アプリケーションを変更しないままにします。ただし、サーバーの weld サブシステム設定を変更して、空の **beans.xml** ファイルの処理を JBoss EAP 7 の動作に戻します。この設定は、サーバー上のすべてのデプロイメントに影響します。たとえば、CLI の場合: **/subsystem=weld:write-attribute(name=legacy-empty-beans-xml-treatment,value=true)**

4.2.2. CDI API の変更

Jakarta Contexts and Dependency Injection 4.0 では、次の非推奨の API 要素が削除されました。

- **javax.enterprise.inject.spi.Bean.isNullable()** メソッドは削除されました。このメソッドは長年にわたって常に **false** を返してきたため、このメソッドを呼び出すアプリケーションは呼び出しを **false** に置き換えるか、ブランチロジックを削除して **false** ブランチの内容だけを保持することができます。
- **javax.enterprise.inject.spi.BeanManager.createInjectionTarget(AnnotatedType)** メソッドは削除されました。このメソッド呼び出しを **BeanManager.getInjectionTargetFactory(AnnotatedType)** に置き換え、返されたファクトリーを使用して注入ターゲットを作成します。詳細は、Jakarta Contexts and Dependency Injection 仕様の Obtaining an InjectionTarget for a class を参照してください。
- **javax.enterprise.inject.spi.BeanManager.fireEvent(Object, Annotation)** メソッドは削除されました。同様の API へのエントリーポイントとして **BeanManager.getEvent()** を使用します。詳細は、Jakarta Contexts and Dependency Injection 仕様の Firing an event を参照してください。
- **javax.enterprise.inject.spi.BeforeBeanDiscovery.addAnnotatedType(AnnotatedType)** メソッドは削除されました。アプリケーションがこのメソッドを呼び出している場合は、**BeforeBeanDiscovery.addAnnotatedType(AnnotatedType, (String) null)** の呼び出しに

置き換えることができます。

4.2.3. Jakarta Enterprise Beans

Java SE 14 では **java.security.Identity** クラスが削除されたため、その使用は Jakarta Enterprise Beans 4.0 API から削除されました。

- 非推奨の **javax.ejb.EJBContext.getCallerIdentity()** メソッドは削除されました。代わりに **EJBContext.getCallerPrincipal()** を使用すると、**java.security.Principal** が返されます。
- 非推奨の **javax.ejb.EJBContext.isCallerInRole(Identity role)** メソッドは削除されました。代わりに **EJBContext.isCallerInRole(String roleName)** を使用できます。
- Jakarta XML RPC 仕様は Jakarta EE 10 フルプラットフォームから削除されたため、**javax.xml.rpc.handler.MessageContext** を返す **javax.ejb.SessionContext.getMessageContext()** メソッドは削除されました。
- Jakarta XML RPC 仕様は Jakarta EE 8 ではオプションでしたが、Red Hat JBoss EAP 7 ではサポートされていません。この仕様を使用すると **IllegalStateException** がスローされるため、この EJB API の変更は JBoss EAP 7 で実行されている既存のアプリケーションに影響を与えることは想定されていません。
- 非推奨の **javax.ejb.EJBContext.getEnvironment()** メソッドは削除されました。JNDI ネーミングコンテキスト **java:comp/env** を使用して、エンタープライズ Bean の環境にアクセスします。

4.2.4. Jakarta Expression Language

スペルが間違っている **javax.el.MethodExpression.isParmetersProvided()** メソッドは削除されました。代わりに **MethodExpression.isParametersProvided()** を使用できます。

4.2.5. Jakarta JSON Binding

デフォルトでは、**jakarta.json.bind.annotation.JsonbCreator** アノテーションが付けられた型では、JSON コンテンツですべてのパラメーターを使用できる必要はありません。解析中の JSON にパラメーターのいずれかが欠落している場合は、デフォルト値が使用されます。JSON 内にすべてのパラメーターが存在することを必要とする EE 8 の動作は、**jakarta.json.bind.JsonbConfig().withCreatorParametersRequired(true)** を呼び出すことで有効化できます。

4.2.6. Jakarta Faces

次の非推奨の機能は Jakarta Faces 4.0 で削除されました。

4.2.6.1. Jakarta Faces と Java Server Pages

Jakarta Server Pages (JSP) のサポートは、Jakarta Faces 2.0 以降のバージョンでは非推奨になりました。JSP サポートは Jakarta Faces 4.0 で削除されました。優先するビュー定義言語 (VDL) として、Facelets がJSP を置き換えます。Faces ビューに JSP を使用するアプリケーションは、Facelets を使用して変更できます。**FacesServlet** を **web.xml** の ***.jsp** 接尾辞にマッピングすることで、アプリケーションを識別できます。

4.2.6.2. Faces マネージド Bean

非推奨となった Jakarta Faces 固有のマネージド Bean の概念は、Jakarta Contexts and Dependency Injection (CDI) Bean 用に Faces 4.0 で削除されました。Faces マネージド Bean を使用するアプリケーション (つまり、**javax.faces.bean.ManagingBean** でアノテーションが付けられたクラス、または **faces-config.xml** のマネージド Bean 要素で参照されるクラス) は、次の変更を行う必要がある場合があります。

- **javax.faces.bean.ManagingBean** のアノテーションが付けられたクラス、または **faces-config.xml** のマネージド Bean 要素で参照されるクラスには、代わりに **jakarta.inject.Named** のアノテーションが付けられ、**faces-config.xml** のマネージド Bean 要素はすべて削除される必要があります。

- **javax.faces.bean.ManagedProperty** アノテーションが付けられたメンバーは、**jakarta.inject.Inject** アノテーションとともに、代わりに **jakarta.faces.annotation.ManagingProperty** を使用する必要があります。古い **javax.faces.bean.ManagedBean(name="foo", eager=true)** に似た起動セマンティクスを取得するには、**public void xxx(@Observes jakarta.enterprise.event.Startup event)** メソッドまたは **public void xxx(@Observes @Initialized(ApplicationScoped.class) Object context)** メソッドを追加します。**jakarta.enterprise.event.Startup** オプションは CDI 4.0 では新しいオプションです。

- **javax.faces.bean.ApplicationScoped** アノテーションの使用は、**jakarta.enterprise.context.ApplicationScoped** に置き換える必要があります。

- **javax.faces.bean.CustomScoped** アノテーションの使用は、CDI カスタムスコープおよび **jakarta.enterprise.context.spi.Context** に置き換える必要があります。詳細は、CDI 4.0 仕様の Defining new scope types および The Context Interface を参照してください。

- **javax.faces.bean.NoneScoped** アノテーションの使用は、ほぼ同様のセマンティクスを持つ CDI 組み込みスコープである **jakarta.enterprise.context.Dependent** に置き換える必要があります。

- **javax.faces.bean.RequestScoped** アノテーションの使用は、**jakarta.enterprise.context.RequestScoped** に置き換える必要があります。

- **javax.faces.bean.SessionScoped** アノテーションの使用は、**jakarta.enterprise.context.SessionScoped** に置き換える必要があります。

4.2.6.3. その他の Faces API の変更

javax.faces.bean.ViewScoped アノテーションが削除されました。代わりに、**jakarta.faces.view.ViewScoped** を使用できます。

javax.faces.view.facelets.ResourceResolver アノテーションと **javax.faces.view.facelets.FaceletsResourceResolver** アノテーションが削除されました。アプリケーション内の ResourceResolver については、**jakarta.faces.application.ResourceHandler** インターフェイスを実装し、**faces-config.xml** の **application/resource-handler** 要素に実装の完全修飾クラス名を登録します。

4.2.7. Jakarta Servlet

Jakarta Servlet 6.0 では、Servlet 5.0 以前 (主に Servlet 2.x リリース) で非推奨となった多数の API クラスとメソッドが削除されています。

javax.servlet.SingleThreadModel マーカーインターフェイスは削除されたため、このインターフェイスを実装するサーブレットはインターフェイス宣言を削除し、サーブレットコードが状態および他のリソースアクセスを同時アクセスから適切に保護するようにする必要があります。たとえば、インスタン

ス変数の使用を回避したり、リソースにアクセスするコードのブロックを同期したりします。ただし、このような同期はパフォーマンスに悪影響を与えるため、開発者は **service** メソッド (またはサービスメソッドのディスパッチ先である **doGet** や **doPost** などのメソッド) を同期しないことが推奨されます。

javax.servlet.http.HttpSessionContext インターフェイスは、**javax.servlet.http.HttpSession.getSessionContext()** メソッドとともに削除されました。サーブレット 2.1 以降、このインターフェイスの実装では使用可能なデータを提供しないことが仕様で要求されていたため、このインターフェイスの使用例はありませんでした。

javax.servlet.http.HttpUtils ユーティリティークラスは削除されました。アプリケーションは、次のメソッドの代わりに **ServletRequest** インターフェイスと **HttpServletRequest** インターフェイスを使用する必要があります。

- **parseQueryString(String s)** および **parsePostData(int len, ServletInputStream in)** - **ServletRequest.getParameterMap()** を使用します。アプリケーションがクエリー文字列パラメーターとリクエスト本文パラメーターを区別する必要がある場合、アプリケーションはクエリー文字列自体を解析してそれを行うコードを実装する必要があります。

- **getRequestURL(HttpServletRequest req)**- **HttpServletRequest.getRequestURL()** を使用します。

また、次のさまざまなメソッドとコンストラクターが削除されました。

クラス/インターフェイス	削除済み	代替
javax.servlet.ServletContext	getServlet(String name)	代替なし
	getServlets()	代替なし
	getServletNames()	代替なし
	log(Exception exception, String msg)	log(String message, Throwable throwable)
javax.servlet.ServletRequest	getRealPath(String path)	ServletContext.getRealPath(String path)
javax.servlet.ServletRequestWrapper	getRealPath(String path)	ServletContext.getRealPath(String path)
javax.servlet.UnavailableException	getServlet()	代替なし

クラス/インターフェイス	削除済み	代替
	UnavailableException(Servlet servlet, String msg)	UnavailableException(String)
	UnavailableException(int seconds, Servlet servlet, String msg)	UnavailableException(String, int)
javax.servlet.http.HttpServletRequest	isRequestedSessionIdFromUrl()	isRequestedSessionIdFromURL()
javax.servlet.http.HttpServletRequestWrapper	isRequestedSessionIdFromUrl()	isRequestedSessionIdFromURL()
javax.servlet.http.HttpServletResponse	encodeUrl(String url)	encodeURL(String url)
	encodeRedirectUrl(String url)	encodeRedirectURL(String url)
	setStatus(int sc, String sm)	sendError(int, String)
javax.servlet.http.HttpServletResponseWrapper	encodeUrl(String url)	encodeURL(String url)
	encodeRedirectUrl(String url)	encodeRedirectURL(String url)
	setStatus(int sc, String sm)	sendError(int, String)
javax.servlet.http.HttpSession	getValue(String name)	getAttribute(String name)

クラス/インターフェイス	削除済み	代替
	getValueNames()	getAttributeNames()
	putValue(String name, Object value)	setAttribute(String name, Object value)
	removeValue(String name)	removeAttribute(String name)

4.2.8. 添付ファイル付きの Jakarta Soap

jaxm.properties ファイルによるプロバイダー検索のサポートは削除されました。

非推奨の **javax.xml.soap.SOAPElementFactory** クラスは削除されました。SOAPElements の作成には、**jakarta.xml.soap.SOAPFactory** を使用します。

SOAPElementFactory メソッド	SOAPFactory と同等のもの
newInstance()	newInstance()
create(Name)	createElement(Name)
create(String)	createElement(String)
create(String, String, String)	createElement(String, String, String)

4.2.9. Jakarta XML Binding

xml バインディングファイルで使用する必要がある XML 名前空間が変更されました。**http://java.sun.com/xml/ns/jaxb** 名前空間は **https://jakarta.ee/xml/ns/jaxb** に置き換える必要があります。

非推奨の **javax.xml.bind.Validator** インターフェイスは、**associated javax.xml.bind.JAXBContext.createValidator()** メソッドと同様に削除されました。マーシャリングおよびアンマーシャリング操作を検証するには、**javax.xml.validation.Schema** を **jakarta.xml.bind.Marshaller.setSchema(Schema)** に提供します。

JAXB 1.0 との互換性のサポートは削除されました。

JAXBContext 実装ルックアップアルゴリズムの非推奨のステップの一部が削除されました。**jaxb.properties** ファイル、**javax.xml.bind.context.factory** または **jakarta.xml.bind.JAXBContext** プロパティー、および **/META-**

INF/services/javax.xml.bind.JAXBContext リソースファイルを介した実装クラス名の検索は削除されました。現在の実装検出アルゴリズムの詳細は、Jakarta XML Binding 4.0 specification を参照してください。

javax.xml.bind.Marshaller インターフェイスの多くのメソッドの一般的な要件は、次のように変更されました。

Jakarta XML Binding 2.3 / 3.0	Jakarta XML Binding 4.0
<A extends XmlAdapter> void setAdapter(A adapter)	<A extends XmlAdapter<?, ?>> void setAdapter(A adapter)
<A extends XmlAdapter> void setAdapter(Class<A> type, A adapter)	<A extends XmlAdapter<?, ?>> void setAdapter(Class<A> type, A adapter)
<A extends XmlAdapter> A getAdapter(Class<A> type)	<A extends XmlAdapter<?, ?>> A getAdapter(Class<A> type)

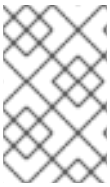
Jakarta XML Binding API の変更とは別に、実装ライブラリー EAP 8 のパッケージ名が大幅に変更されました。これは、実装ライブラリーに直接アクセスする一部のアプリケーションに影響を与える可能性があります。

- **com.sun.xml.bind** パッケージ内のクラスを使用する場合は、**org.glassfish.jaxb.runtime** パッケージ内のクラスに置き換える必要があります。**com.sun.xml.bind** のサブパッケージ内のクラスは、対応する **org.glassfish.jaxb.runtime** サブパッケージ内のクラスに置き換える必要があります。

- **jakarta.xml.bind.Marshaller** プロパティー設定の場合、プロパティー定数名を **com.sun.xml.bind.*** から **org.glassfish.jaxb.*** に変更します。たとえば、**marshaller.setProperty("com.sun.xml.bind.namespacePrefixMapper", mapper)** は **marshaller.setProperty("org.glassfish.jaxb.namespacePrefixMapper", mapper)** になります。

第5章 JBOSS EAP アプリケーションの MAVEN プロジェクトを JBOSS EAP 8.0 に移行する

アプリケーションの Maven プロジェクトを JBoss EAP BOM を使用して依存関係を管理する JBoss EAP 8.0 に移行する場合、JBoss EAP 8.0 BOM で導入された次の重要な変更のため、pom.xml ファイルを更新する必要があります。



注記

アプリケーションを変更せずに JBoss EAP 8.0 に移行すると、そのアプリケーションは誤った依存関係でビルドされ、JBoss EAP 8.0 へのデプロイに失敗する可能性があります。

5.1. JBOSS EAP JAKARTA EE 8 の名前変更

JBoss EAP **Jakarta EE 8** BOM の名前が **JBoss EAP EE** に変更され、その Maven 座標が **org.jboss.bom:jboss-eap-jakartaee8** から **org.jboss.bom:jboss-eap-ee** に変更されました。Maven プロジェクト (**pom.xml**) でのこの BOM の使用は、次の依存関係管理インポートで識別できます。

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.jboss.bom</groupId>
      <artifactId>jboss-eap-jakartaee8</artifactId>
      <version>${version.bom}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

Maven プロジェクト (**pom.xml**) は、代わりに新しい "JBoss EAP EE" BOM を依存関係管理にインポートする必要があります。

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.jboss.bom</groupId>
      <artifactId>jboss-eap-ee</artifactId>
      <version>${version.bom}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

5.2. ツールを使用した JBOSS EAP JAKARTA EE 8 の名前変更

JBoss EAP **Jakarta EE 8 With Tools** BOM の名前が **JBoss EAP EE with tools** に変更され、Maven コーディネートが **org.jboss.bom:jboss-eap-jakartaee8-with-tools** から **org.jboss.bom:jboss-eap-ee-with-tools** に変更されました。Maven プロジェクト (**pom.xml**) でのこの BOM の使用は、次の依存関係管理インポートで識別できます。

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.jboss.bom</groupId>
      <artifactId>jboss-eap-jakartaee8-with-tools</artifactId>
      <version>${version.bom}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

Maven プロジェクト (**pom.xml**) は、代わりに新しい "JBoss EAP EE With Tools" BOM を依存関係管理にインポートする必要があります。

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.jboss.bom</groupId>
      <artifactId>jboss-eap-ee-with-tools</artifactId>
      <version>${version.bom}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

5.3. JBOSS EAP JAKARTA EE 8 API の削除

JBoss EAP **Jakarta EE 8 APIs** BOM は JBoss EAP 8.0 で削除されたため、代わりに新しい **JBoss EAP EE** BOM を使用する必要があります。Maven プロジェクト (**pom.xml**) でのこの BOM の使用は、次の依存関係管理インポートで識別できます。

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.jboss.spec</groupId>
      <artifactId>jboss-jakartaee-8.0</artifactId>
      <version>${version.bom}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
    <dependency>
      <groupId>org.jboss.spec</groupId>
      <artifactId>jboss-jakartaee-web-8.0</artifactId>
      <version>${version.bom}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

Maven プロジェクト (**pom.xml**) は、新しい **JBoss EAP EE** BOM を依存関係管理にインポートする必要があります。

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.jboss.bom</groupId>
      <artifactId>jboss-eap-ee</artifactId>
      <version>${version.bom}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

5.4. JBOSS EAP RUNTIME BOM の削除

JBoss EAP **Runtime** BOM は JBoss EAP 8.0 では配布されなくなったため、代わりに新しい **JBoss EAP EE** BOM を使用する必要があります。Maven プロジェクト (**pom.xml**) でのこの BOM の使用は、次の依存関係管理インポートで識別できます。

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.jboss.bom</groupId>
      <artifactId>eap-runtime-artifacts</artifactId>
      <version>${version.bom}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

Maven プロジェクト (**pom.xml**) は、代わりに新しい **JBoss EAP EE** BOM を依存関係管理にインポートする必要があります。

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.jboss.bom</groupId>
      <artifactId>jboss-eap-ee</artifactId>
      <version>${version.bom}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

5.5. JAKARTA EE および JBOSS API の MAVEN コーディネートの変更

JBoss EAP BOM によって提供された以下の Jakarta EE および JBoss API アーティファクトは、Maven コーディネートが変更されたか、他のアーティファクトに置き換えられました。

- **com.sun.activation:jakarta.activation**
- **org.jboss.spec.javax.annotation:jboss-annotations-api_1.3_spec**

- **org.jboss.spec.javax.security.auth.message:jboss-jaspi-api_1.0_spec**
- **org.jboss.spec.javax.security.jacc:jboss-jacc-api_1.5_spec**
- **org.jboss.spec.javax.batch:jboss-batch-api_1.0_spec**
- **org.jboss.spec.javax.ejb:jboss-ejb-api_3.2_spec**
- **org.jboss.spec.javax.el:jboss-el-api_3.0_spec**
- **org.jboss.spec.javax.enterprise.concurrent:jboss-concurrency-api_1.0_spec**
- **org.jboss.spec.javax.faces:jboss-jsf-api_2.3_spec**
- **org.jboss.spec.javax.interceptor:jboss-interceptors-api_1.2_spec**
- **org.jboss.spec.javax.jms:jboss-jms-api_2.0_spec**
- **com.sun.mail:jakarta.mail**
- **org.jboss.spec.javax.resource:jboss-connector-api_1.7_spec**
- **org.jboss.spec.javax.servlet:jboss-servlet-api_4.0_spec**
- **org.jboss.spec.javax.servlet.jsp:jboss-jsp-api_2.3_spec**
- **org.apache.taglibs:taglibs-standard-spec**
- **org.jboss.spec.javax.transaction:jboss-transaction-api_1.3_spec**
- **org.jboss.spec.javax.xml.bind:jboss-jaxb-api_2.3_spec**
- **org.jboss.spec.javax.xml.ws:jboss-jaxws-api_2.3_spec**
- **javax.jws:jsr181-api**
- **org.jboss.spec.javax.websocket:jboss-websocket-api_1.1_spec**
- **org.jboss.spec.javax.ws.rs:jboss-jaxrs-api_2.1_spec**
- **org.jboss.spec.javax.xml.soap:jboss-saaj-api_1.4_spec**
- **org.hibernate:hibernate-core**
- **org.hibernate:hibernate-jpamodelgen**
- **org.jboss.narayana.xts:jbossxts**

Maven プロジェクト (**pom.xml**) は、アーティファクトが変更または置換された場合は依存関係の Maven コーディネートを更新し、アーティファクトがサポートされなくなった場合は依存関係を削除する必要があります。

```
<dependencies>
  <!-- replaces com.sun.activation:jakarta.activation -->
  <dependency>
    <groupId>jakarta.activation</groupId>
    <artifactId>jakarta.activation-api</artifactId>
  </dependency>
```

```
<!-- replaces org.jboss.spec.javax.annotation:jboss-annotations-api_1.3_spec -->
<dependency>
    <groupId>jakarta.annotation</groupId>
    <artifactId>jakarta.annotation-api</artifactId>
</dependency>
<!-- replaces org.jboss.spec.javax.security.auth.message:jboss-jaspi-api_1.0_spec -->
<dependency>
    <groupId>jakarta.authentication</groupId>
    <artifactId>jakarta.authentication-api</artifactId>
</dependency>
<!-- replaces org.jboss.spec.javax.security.jacc:jboss-jacc-api_1.5_spec -->
<dependency>
    <groupId>jakarta.authorization</groupId>
    <artifactId>jakarta.authorization-api</artifactId>
</dependency>
<!-- replaces org.jboss.spec.javax.batch:jboss-batch-api_1.0_spec -->
<dependency>
    <groupId>jakarta.batch</groupId>
    <artifactId>jakarta.batch-api</artifactId>
</dependency>
<!-- replaces org.jboss.spec.javax.ejb:jboss-ejb-api_3.2_spec -->
<dependency>
    <groupId>jakarta.ejb</groupId>
    <artifactId>jakarta.ejb-api</artifactId>
</dependency>
<!-- replaces org.jboss.spec.javax.el:jboss-el-api_3.0_spec -->
<dependency>
    <groupId>org.jboss.spec.jakarta.el</groupId>
    <artifactId>jboss-el-api_5.0_spec</artifactId>
</dependency>
<!-- replaces org.jboss.spec.javax.enterprise.concurrent:jboss-concurrency-api_1.0_spec -->
<dependency>
    <groupId>jakarta.enterprise.concurrent</groupId>
    <artifactId>jakarta.enterprise.concurrent-api</artifactId>
</dependency>
<!-- replaces org.jboss.spec.javax.faces:jboss-jsf-api_2.3_spec -->
<dependency>
    <groupId>jakarta.faces</groupId>
    <artifactId>jakarta.faces-api</artifactId>
</dependency>
<!-- replaces org.jboss.spec.javax.interceptor:jboss-interceptors-api_1.2_spec -->
<dependency>
    <groupId>jakarta.interceptor</groupId>
    <artifactId>jakarta.interceptor-api</artifactId>
</dependency>
<!-- replaces org.jboss.spec.javax.jms:jboss-jms-api_2.0_spec  -->
<dependency>
    <groupId>jakarta.jms</groupId>
    <artifactId>jakarta.jms-api</artifactId>
</dependency>
<!-- replaces com.sun.mail:jakarta.mail -->
<dependency>
    <groupId>jakarta.mail</groupId>
    <artifactId>jakarta.mail-api</artifactId>
</dependency>
<!-- replaces org.jboss.spec.javax.resource:jboss-connector-api_1.7_spec -->
```

```
<dependency>
    <groupId>jakarta.resource</groupId>
    <artifactId>jakarta.resource-api</artifactId>
</dependency>
<!-- replaces org.jboss.spec.javax.servlet:jboss-servlet-api_4.0_spec -->
<dependency>
    <groupId>jakarta.servlet</groupId>
    <artifactId>jakarta.servlet-api</artifactId>
</dependency>
<!-- replaces org.jboss.spec.javax.servlet.jsp:jboss-jsp-api_2.3_spec -->
<dependency>
    <groupId>jakarta.servlet.jsp</groupId>
    <artifactId>jakarta.servlet.jsp-api</artifactId>
</dependency>
<!-- replaces org.apache.taglibs:taglibs-standard-spec -->
<dependency>
    <groupId>jakarta.servlet.jsp.jstl</groupId>
    <artifactId>jakarta.servlet.jsp.jstl-api</artifactId>
</dependency>
<!-- replaces org.jboss.spec.javax.transaction:jboss-transaction-api_1.3_spec  -->
<dependency>
    <groupId>jakarta.transaction</groupId>
    <artifactId>jakarta.transaction-api</artifactId>
</dependency>
<!-- replaces org.jboss.spec.javax.xml.bind:jboss-jaxb-api_2.3_spec  -->
<dependency>
    <groupId>jakarta.xml.bind</groupId>
    <artifactId>jakarta.xml.bind-api</artifactId>
</dependency>
<!-- replaces org.jboss.spec.javax.xml.ws:jboss-jaxws-api_2.3_spec and javax.jws:jsr181-api -->
<dependency>
    <groupId>org.jboss.spec.jakarta.xml.ws</groupId>
    <artifactId>jboss-jakarta-xml-ws-api_4.0_spec</artifactId>
</dependency>
<!-- replaces org.jboss.spec.javax.websocket:jboss-websocket-api_1.1_spec -->
<dependency>
    <groupId>jakarta.websocket</groupId>
    <artifactId>jakarta.websocket-api</artifactId>
</dependency>
<!-- replaces org.jboss.spec.javax.ws.rs:jboss-jaxrs-api_2.1_spec -->
<dependency>
    <groupId>jakarta.ws.rs</groupId>
    <artifactId>jakarta.ws.rs-api</artifactId>
</dependency>
<!-- replaces org.jboss.spec.javax.xml.soap:jboss-saaj-api_1.4_spec -->
<dependency>
    <groupId>org.jboss.spec.jakarta.xml.soap</groupId>
    <artifactId>jboss-saaj-api_3.0_spec</artifactId>
</dependency>
<!-- replaces org.hibernate:hibernate-core -->
<dependency>
    <groupId>org.hibernate.orm</groupId>
    <artifactId>hibernate-core</artifactId>
</dependency>
<!-- replaces org.hibernate:hibernate-jpamodelgen -->
<dependency>
```

```
      <groupId>org.hibernate.orm</groupId>
      <artifactId>hibernate-jpamodelgen</artifactId>
    </dependency>
    <!-- replaces org.jboss.narayana.xts:jbossxts -->
    <dependency>
      <groupId>org.jboss.narayana.xts</groupId>
      <artifactId>jbossxts-jakarta</artifactId>
      <classifier>api</classifier>
    </dependency>
  </dependencies>
```

5.6. JBOSS EJB クライアントのレガシー BOM の削除

Maven **groupId:artifactId** コーディネート **org.jboss.eap:wildfly-ejb-client-legacy-bom** を含む JBoss EJB クライアントレガシー BOM は、JBoss EAP では提供されなくなりました。

次の例に示すように、Maven プロジェクト **pom.xml** 設定ファイル内の依存関係管理インポートまたは依存関係参照として使用されるこの BOM を識別できます。

依存関係管理インポートの例

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.jboss.bom</groupId>
      <artifactId>wildfly-ejb-client-legacy-bom</artifactId>
      <version>${version.bom}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

依存関係の参照例

```
<dependencies>
...
  <dependency>
    <groupId>org.jboss.bom</groupId>
    <artifactId>wildfly-ejb-client-legacy-bom</artifactId>
    <version>${version.bom}</version>
    <type>pom</type>
  </dependency>
...
</dependencies>
```

スタンドアロンクライアント Java アプリケーションは、削除された BOM の同じバージョン (バージョン **7.4.0.GA** など) を引き続き使用して、JBoss EAP 8 と連携できます。ただし、EJB クライアントレガシー API を置き換えることを推奨します。EJB クライアントの設定に関する詳細は、How configure an EJB client in EAP 7.1+ / 8.0+ を参照してください。JBoss EAP 7.4 での EJB クライアントレガシー API の非推奨に関する詳細は、Deprecated in Red Hat JBoss Enterprise Application Platform Platform (EAP) 7 を参照してください。

重要

JBoss EAP 7.4 クライアントと JBoss EAP 8.x サーバーは、異なる製品ライフサイクルに従います。詳細は、JBoss EAP の 製品ライフサイクル を参照してください。

第6章 サーバー移行の変更点

移行する前に、アプリケーションをサーバーにデプロイし、Red Hat JBoss Enterprise Application Platform 8.0 でアップグレードするために必要な移行の変更点を必ず理解してください。

6.1. WEB サーバー設定の変更

ルートコンテキストの動作に影響を与え、サーバー情報のセキュリティーを強化する、Red Hat JBoss Enterprise Application Platform 内の **mod_cluster** と Undertow の変更について説明します。

6.1.1. デフォルトの Web モジュールの動作の変更

JBoss EAP 7.0 では、Web アプリケーションのルートコンテキストは **mod_cluster** でデフォルトで無効になっていました。

JBoss EAP 7.1 よりこれが変更になりました。そのため、ルートコンテキストが無効になっていることを想定している場合は予期せぬ結果が生じる可能性があります。たとえば、リクエストが誤って不都合なノードにルーティングされたり、公開してはならないプライベートアプリケーションにパブリックプロキシーを介してアクセスされる可能性があります。Undertow の場所は、明示的に除外されない限り、**mod_cluster** ロードバランサーにも自動的に登録されるようになりました。

以下の管理 CLI コマンドを使用して、**modcluster** サブシステム設定から ROOT を除外します。

```
/subsystem=modcluster/mod-cluster-config=configuration:write-attribute(name=excluded-contexts,value=ROOT)
```

以下の管理 CLI コマンドを使用して、デフォルトのウェルカム web アプリケーションを無効にします。

```
/subsystem=undertow/server=default-server/host=default-host/location=\/:remove
/subsystem=undertow/configuration=handler/file=welcome-content:remove
reload
```

関連情報

- デフォルトの Welcome Web アプリケーションの設定

6.1.2. Undertow サブシステムのデフォルト設定の変更

Red Hat JBoss Enterprise Application Platform 7.2 より前のデフォルトの **undertow** サブシステム設定には、**default-host** によって各 HTTP 応答に追加される 2 つの応答ヘッダーフィルターが含まれていました。

- **Server** は、以前は **JBoss EAP/7** に設定されていました。
- **X-Powered-By** は、以前は **Undertow/1** に設定されていました。

使用中のサーバーに関する情報を無意識に公開しないようにするため、これらの応答ヘッダーフィルターはデフォルトの JBoss EAP 7.2 設定から削除されました。

以下の例は、JBoss EAP 7.1 **undertow** サブシステムのデフォルト設定を示しています。

```
<subsystem xmlns="urn:jboss:domain:undertow:4.0">
```

```
    <buffer-cache name="default"/>
    <server name="default-server">
        <http-listener name="default" socket-binding="http" redirect-socket="https"/>
        <https-listener name="https" socket-binding="https" security-realm="ApplicationRealm" enable-
http2="true"/>
        <host name="default-host" alias="localhost">
            <location name="/" handler="welcome-content"/>
            <filter-ref name="server-header"/>
            <filter-ref name="x-powered-by-header"/>
            <http-invoker security-realm="ApplicationRealm"/>
        </host>
    </server>
    <servlet-container name="default">
        <jsp-config/>
        <websockets/>
    </servlet-container>
    <handlers>
        <file name="welcome-content" path="${jboss.home.dir}/welcome-content"/>
    </handlers>
    <filters>
        <response-header name="server-header" header-name="Server" header-value="JBoss-EAP/7"/>
        <response-header name="x-powered-by-header" header-name="X-Powered-By" header-
value="Undertow/1"/>
    </filters>
</subsystem>
```

以下の例は、JBoss EAP 7.4 **undertow** サブシステムのデフォルト設定を示しています。

```
<subsystem xmlns="urn:jboss:domain:undertow:12.0" default-server="default-server" default-virtual-
host="default-host" default-servlet-container="default" default-security-domain="other">
    <buffer-cache name="default"/>
    <server name="default-server">
        <http-listener name="default" socket-binding="http" redirect-socket="https" enable-http2="true"/>
        <https-listener name="https" socket-binding="https" security-realm="ApplicationRealm" enable-
http2="true"/>
        <host name="default-host" alias="localhost">
            <location name="/" handler="welcome-content"/>
            <http-invoker security-realm="ApplicationRealm"/>
        </host>
    </server>
    <servlet-container name="default">
        <jsp-config/>
        <websockets/>
    </servlet-container>
    <handlers>
        <file name="welcome-content" path="${jboss.home.dir}/welcome-content"/>
    </handlers>
</subsystem>
```

以下は、JBoss EAP 8.0 のデフォルトの **undertow** サブシステム設定の例です。

```
<subsystem xmlns="urn:jboss:domain:undertow:14.0" default-virtual-host="default-host" default-
servlet-container="default" default-server="default-server" statistics-
enabled="${wildfly.undertow.statistics-enabled:${wildfly.statistics-enabled:false}}" default-security-
domain="other">
```

```
    <byte-buffer-pool name="default"/>
    <buffer-cache name="default"/>
    <server name="default-server">
        <http-listener name="default" socket-binding="http" redirect-socket="https" enable-http2="true"/>
        <https-listener name="https" socket-binding="https" ssl-context="applicationSSC" enable-
http2="true"/>
        <host name="default-host" alias="localhost">
            <location name="/" handler="welcome-content"/>
            <http-invoker http-authentication-factory="application-http-authentication"/>
        </host>
    </server>
    <servlet-container name="default">
        <jsp-config/>
        <websockets/>
    </servlet-container>
    <handlers>
        <file name="welcome-content" path="${jboss.home.dir}/welcome-content"/>
    </handlers>
    <application-security-domains>
        <application-security-domain name="other" security-domain="ApplicationDomain"/>
    </application-security-domains>
</subsystem>
```

6.2. INFINISPAN サーバー設定の変更

以下の点を考慮して、Red Hat JBoss Enterprise Application Platform 7.1 以降でパッシベーション用のカスタムステートフルセッション Bean (SFSB) キャッシュを設定します。

- idle-timeout 属性が非推奨となる
- レイジー (Lazy) パッシベーションの実装
- クラスター名の決定
- エビクションと有効期限の適切な設定
- パフォーマンスを向上させるためのキャッシュコンテナートランスポートプロトコルの変更

これらの考慮事項に従うことで、JBoss EAP 7.1 以降でのパッシベーションを改善するために SFSB キャッシュ設定を最適化できます。

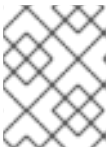
6.2.1. パッシベーション用のカスタムステートフルセッション Bean キャッシュの設定

JBoss EAP 7.1 以降のバージョンでは、パッシベーションが有効になっているカスタムステートフルセッション Bean (SFSB) キャッシュが変更されました。パッシベーションを使用して SFSB キャッシュを設定する場合は、次の重要な変更点を考慮してください。

- idle-timeout 属性が非推奨となる
- イーガー (Eager) パッシベーションからレイジー (Lazy) パッシベーションへの移行
- クラスター名の決定
- Jakarta Enterprise Beans キャッシュでのエビクションと有効期限の設定

JBoss EAP 7.1 以降のバージョンでパッシベーション用のカスタム SFSB キャッシュを設定する場合は、以下の制限を考慮してください。

- **ejb3** サブシステムの **infinispan passivation-store** で設定される **idle-timeout** 属性は、JBoss EAP 7.1 以上のリリースでは非推奨となりました。JBoss EAP 7.1 以降は、**max-size** のしきい値に達したときに発生するレイジー (Lazy) パッシベーションのみをサポートします。



注記

アイドルタイムアウトによるイーガー (Eager) パッシベーションは、これらのバージョンではサポートされなくなりました。

- JBoss EAP 7.1 以降では、Jakarta Enterprise Beans クライアントによって使用されるクラスター名は、**jgroups** サブシステムで設定されているチャネルの実際のクラスター名によって決定されます。

- JBoss EAP 7.1 以上のリリースでも、**max-size** 属性を設定してパッシベーションのしきい値を制御できます。

6.2.2. Infinispan キャッシュコンテナートランスポートの変更

JBoss EAP 7.0 とそれ以降のバージョン間の動作の変更には、バッチモードでキャッシュコンテナートランスポートプロトコルの更新を実行するか、特別なヘッダーを使用する必要があります。この変更は、JBoss EAP サーバーの管理に使用されるツールにも影響します。

以下は、JBoss EAP 7.0 でキャッシュコンテナートランスポートプロトコルの設定に使用された管理 CLI コマンドの例になります。

```
/subsystem=infinispan/cache-container=my:add()
/subsystem=infinispan/cache-container=my/transport=jgroups:add()
/subsystem=infinispan/cache-container=my/invalidation-cache=mycache:add(mode=SYNC)
```

以下は、JBoss EAP 7.1 で同じ設定を実行するために必要な管理 CLI コマンドの例になります。コマンドはバッチモードで実行されることに注意してください。

```
batch
/subsystem=infinispan/cache-container=my:add()
/subsystem=infinispan/cache-container=my/transport=jgroups:add()
/subsystem=infinispan/cache-container=my/invalidation-cache=mycache:add(mode=SYNC)
run-batch
```

バッチモードを使用したくない場合は、代わりにトランスポートの定義時に **allow-resource-service-restart=true** 操作ヘッダーを指定できます。

スクリプトを使用してキャッシュコンテナートランスポートプロトコルを更新する場合は、スクリプトを確認し、バッチモードを追加してください。

6.2.3. バージョン 8.0 以降からの EJB サブシステム設定の変更点

JBoss EAP 8.0 では、新しいサブシステムといくつかのリソースの更新を含む、分散可能ステートフルセッション Bean (SFSB) の Enterprise JavaBeans (EJB) サブシステム設定への変更が導入されています。JBoss EAP 6 および 7 で使用されるいくつかのリソースも非推奨になりました。これらの変更により、サーバー設定の移行が可能になり、アプリケーションが今後のメジャーリリースと確実に互換性を持つことができます。

JBoss EAP 8.0 は、JBoss EAP 6 および 7 で使用されていた非推奨のリソースを 2 つの新しいリソースと、SFSB キャッシングを分散的に設定するための **distributable-ejb** サブシステムに置き換えます。次の表は、非推奨のリソースと、それらに代わる新しいリソースの概要を示しています。

表6.1 SFSB キャッシュ設定の変更

非推奨のリソース	新しい非分散型 SFSB キャッシュ	新しい分散可能な SFSB キャッシュ
/subsystem=ejb3/cache	**/subsystem=ejb3/simple-cache**	**/subsystem=ejb3/distributable-cache**
/subsystem=ejb3/passivation-store	NA	**/subsystem=ejb3/distributable-cache=”name”/bean-management"=..**

非分散 SFSB キャッシュ **/subsystem=ejb3/simple-cache** は、パッシベーションストアが定義されていない JBoss EAP 7 で使用される SFSB キャッシュ **/subsystem=ejb3/cache** と同等になります。

分散可能 SFSB キャッシュ **/subsystem=ejb3/distributable-cache** には、**distributable-ejb** サブシステムからの対応するリソースを参照するオプションの **bean-management** 属性が含まれています。リソースを定義しない場合、**distributable-ejb** サブシステム内の **bean-management** リソースがデフォルトで使用されます。

サーバー設定を JBoss EAP 8.0 の更新されたアプローチに移行することを検討してください。現在のリリースは非推奨のリソースで引き続き機能しますが、将来のリリースでは、リソースが削除されると、機能しなくなる可能性があります。

JBoss EAP 7 と推奨される JBoss EAP 8.0 設定の比較例は次のとおりです。

JBoss EAP 7 設定:

```
/subsystem=ejb3/cache=example-simple-cache:add()
/subsystem=ejb3/passivation-store=infinispan:add(cache-container=ejb, bean-cache=default, max-size=1024)
/subsystem=ejb3/cache=example-distributed-cache:add(passivation-store=infinispan)
```

推奨される JBoss EAP 8.0 設定:

```
/subsystem=ejb3/simple-cache=example-simple-cache:add()
/subsystem=distributable-ejb=example-distributed-cache/infinispan-bean-management=example-bean-cache:add(cache-container=ejb, cache=default, max-active-beans=1024)
/subsystem=ejb3/distributable-cache=example-distributed-cache:add(bean-management=example-bean-cache)
```

推奨される JBoss EAP 8.0 設定を採用すると、サーバーが最新バージョンおよび今後のメジャーリリースと互換性を持つことが確実となります。また、分散可能な SFSB 用の改善されたリソースとサブシステムの恩恵を受けることもできます。

6.3. JAKARTA ENTERPRISE BEANS サーバー設定の変更

JBoss EAP 7 で **ejb3** サブシステムを設定している際に、エンタープライズ Bean アプリケーションのデプロイメント中にサーバーログに例外が表示される場合があります。

重要

JBoss Server Migration Tool を使用してサーバー設定を更新する場合は、**ejb3** サブシステムが適切に設定されており、Jakarta Enterprise Beans アプリケーションのデプロイ時に問題が発生しないことを確認してください。ツールの設定と実行については、JBoss Server Migration Tool の使用 を参照してください。

6.3.1. キャッシュ変更による DuplicateServiceException の解決

次の **DuplicateServiceException** エラーは、JBoss EAP 7 でのキャッシュ変更が原因で発生します。

サーバーログの DuplicateServiceException

```
ERROR [org.jboss.msc.service.fail] (MSC service thread 1-3) MSC000001: Failed to start service
jboss.deployment.unit."mdb-1.0-SNAPSHOT.jar".cache-dependencies-installer:
org.jboss.msc.service.StartException in service jboss.deployment.unit."mdb-1.0-
SNAPSHOT.jar".cache-dependencies-installer: Failed to start service
...
Caused by: org.jboss.msc.service.DuplicateServiceException: Service jboss.infinispan.ejb."mdb-1.0-
SNAPSHOT.jar".config is already registered
```

JBoss EAP 7 でのキャッシュの変更によって発生する **DuplicateServiceException** を解決するには、次のコマンドを実行して **ejb3** サブシステムでキャッシュを再設定します。

```
/subsystem=ejb3/file-passivation-store=file:remove
/subsystem=ejb3/cluster-passivation-store=infinispan:remove
/subsystem=ejb3/passivation-store=infinispan:add(cache-container=ejb, max-size=10000)

/subsystem=ejb3/cache=passivating:remove
/subsystem=ejb3/cache=clustered:remove
/subsystem=ejb3/cache=distributable:add(passivation-store=infinispan, aliases=[passivating,
clustered])
```

キャッシュを再設定することで、このエラーを解決し、**DuplicateServiceException** の発生を阻止することができます。

6.4. メッセージングサーバー設定の変更

設定と関連するメッセージングデータの両方を ActiveMQ Artemis に移行する方法について説明します。ActiveMQ Artemis は、Red Hat JBoss Enterprise Application Platform 8.0 で Jakarta Messaging サポートプロバイダーとして機能します。

6.4.1. メッセージングデータの移行

Red Hat JBoss Enterprise Application Platform でメッセージングデータを移行するために使用できるアプローチを確認します。

メッセージングデータを以前の JBoss EAP 7.x リリースから JBoss EAP 8.0 に移行するには、エクスポートおよびインポートのアプローチを使用してメッセージングデータを移行 できます。このアプローチでは、以前のリリースからメッセージングデータをエクスポートし、管理 CLI **import-journal** 操作を使用してそれを JBoss EAP 8.0 にインポートします。このアプローチは、特にファイルベースのメッセージングシステムに適用できることに注意してください。

バージョン 7 と同様に、JBoss EAP 8.0 は、Jakarta Messaging サポートプロバイダーとして ActiveMQ Artemis を引き続き使用します。これにより、移行プロセスがよりスムーズになります。

6.4.1.1. エクスポートおよびインポートのアプローチを使用してメッセージングデータを移行する

次のアプローチを使用して、メッセージングデータを以前のリリースから XML ファイルにエクスポートしてから、**import-journal** 操作を使用してそのファイルをインポートします。

1. JBoss EAP 7.x リリースからメッセージングデータをエクスポートします。
2. XML 形式のメッセージングデータをインポートする

重要

エクスポートおよびインポートメソッドは、JDBC ベースのジャーナルをストレージとして使用するシステム間でのメッセージングデータの移動には使用できません。

6.4.1.1.1. JBoss EAP 7.x リリースからメッセージングデータをエクスポートする

Red Hat JBoss Enterprise Application Platform 7.x リリースからメッセージングデータをエクスポートするには、以下に示す手順に従います。

前提条件

- JBoss EAP 7.x がシステムにインストールされている。
- ターミナルまたはコマンドラインインターフェイスにアクセスできる。
- ディレクトリーに移動し、コマンドを実行するために必要な権限を持っている。

手順

1. ターミナルを開き、JBoss EAP 7.x のインストールディレクトリーへ移動し、**admin-only** モードでサーバーを起動します。

```
$ EAP_HOME/bin/standalone.sh -c standalone-full.xml --start-mode=admin-only
```

2. 新しいターミナルを開き、JBoss EAP 7.x のインストールディレクトリーへ移動し、管理 CLI に接続します。

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

3. 以下の管理 CLI コマンドを使用して、メッセージングジャーナルデータをエクスポートします。

```
/subsystem=messaging-activemq/server=default:export-journal()
```

検証

- コマンド完了後に、ログにエラーや警告メッセージがないことを確認します。
- オペレーティングシステムと互換性のあるツールを使用して、生成された出力ファイル内の XML を検証します。

6.4.1.1.2. XML 形式のメッセージングデータをインポートする

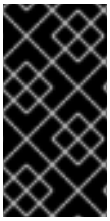
JBoss EAP 8.0 からメッセージングデータをエクスポートした後、**import-journal** 操作を使用して XML ファイルを JBoss EAP 8.0 以降にインポートする必要があります。

前提条件

- 管理 CLI 移行操作または JBoss Server Migration Tool のいずれかを使用して、JBoss EAP 8.0 の移行を完了します。
- Jakarta Messaging クライアントを接続せずに、JBoss EAP 8.0 サーバーを通常モードで起動します。

手順

XML ファイルを JBoss EAP 8.0 以降のバージョンにインポートするには、**import-journal** 操作を使用して、以下の手順に従います。



重要

ターゲットサーバーがすでにいくつかのメッセージングタスクを実行している場合は、インポートが失敗した場合のデータ損失を防ぐために、**import-journal** 操作を開始する前にメッセージングフォルダーを必ずバックアップしてください。詳細は、メッセージングフォルダーデータのバックアップ を参照してください。

1. Jakarta Messaging クライアントが **接続されていない状態** で、JBoss EAP 8.0 サーバーを通常モードで起動します。

重要

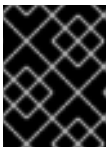
接続している Jakarta Messaging クライアントがない状態でサーバーを起動することが重要になります。これは、**import-journal** 操作が Jakarta Messaging プロデューサーのように動作するためです。操作の実行中、メッセージは即座に使用できます。インポート中にこの操作に失敗し、Jakarta Messaging クライアントが接続されている場合は、Jakarta Messaging クライアントがメッセージの一部を消費している可能性があるため、復元することができません。

2. 新しいターミナルを開き、JBoss EAP 8.0 インストールディレクトリーに移動し、管理 CLI に接続します。

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

3. 次の管理 CLI コマンドを使用して、メッセージングデータをインポートします。

```
/subsystem=messaging-activemq/server=default:import-
journal(file=OUTPUT_DIRECTORY/OldMessagingData.xml)
```



重要

このコマンドは 1 度だけ実行してください。2 回以上実行するとメッセージが複製されます。

6.4.1.1.3. メッセージングデータのインポート失敗からの回復

import-journal 操作が失敗した場合、メッセージングデータのインポート失敗から回復できます。

前提条件

- JBoss EAP 8.0 サーバーとその管理 CLI コマンドに精通している。
- メッセージングジャーナルフォルダーのディレクトリーの場所に関する知識がある。
- 可能な場合は、ターゲットサーバーのメッセージングデータを事前にバックアップしている。

手順

1. JBoss EAP 8.0 サーバーをシャットダウンします。
2. すべてのメッセージングジャーナルフォルダーを削除します。メッセージングジャーナルフォルダーの正しいディレクトリーの場所を確認するには、管理 CLI コマンドの メッセージングフォルダーデータのバックアップ を参照してください。
3. インポートの前にターゲットサーバーのメッセージングデータをバックアップしたら、メッセージングフォルダーをバックアップした場所から前の手順で決定したメッセージングジャーナルディレクトリーにコピーします。
4. XML 形式のメッセージングデータをインポートする の手順を繰り返します。

6.4.1.2. メッセージングブリッジを使用してメッセージングデータを移行する

Jakarta Messaging ブリッジは、ソース Jakarta Messaging キューまたはトピックからのメッセージを消費し、別のサーバー上にあるターゲット Jakarta Messaging キューまたはトピックに送信します。これにより、Jakarta Messaging 3.1 標準に準拠したメッセージングサーバー間のメッセージブリッジングが可能になります。Java Naming and Directory Interface を使用してソースおよび宛先の Jakarta Messaging リソースを検索し、Java Naming and Directory Interface ルックアップのクライアントクラスがモジュールにバンドルされていることを確認し、Jakarta Messaging ブリッジ設定でモジュール名を宣言します。

このセクションでは、サーバーを設定し、メッセージングデータを JBoss EAP 7 から JBoss EAP 8.0 に移動するためのメッセージングブリッジをデプロイする方法について説明します。これを達成するには、次の手順に進みます。

1. JBoss EAP 8.0 サーバーの設定
2. メッセージングデータの移行

6.4.1.2.1. JBoss EAP 8.0 サーバーの設定

モジュールの依存関係やキュー設定を含むメッセージングデータのシームレスな移行のために JBoss EAP 8.0 で Jakarta Messaging ブリッジを設定するには、以下に示す手順に従います。

前提条件

- JBoss EAP 8.0 サーバーがインストールされ、実行されている。

手順

1. JBoss EAP 8.0 サーバーの **message-activemq** サブシステムにデフォルトサーバー用の次の **jms-queue** 設定を作成します。

```
jms-queue add --queue-address=MigratedMessagesQueue --entries=
[jms/queue/MigratedMessagesQueue
java:jboss/exported/jms/queue/MigratedMessagesQueue]
```

2. **messaging-activemq** サブシステムの **default** サーバーに以下と似た **InVmConnectionFactory connection-factory** の設定が含まれるようにしてください。

```
<connection-factory name="InVmConnectionFactory" factory-type="XA_GENERIC"
entries="java:/ConnectionFactory" connectors="in-vm"/>
```

エントリーが含まれていない場合は、以下の管理 CLI コマンドを使用して作成します。

```
/subsystem=messaging-activemq/server=default/connection-
factory=InVmConnectionFactory:add(factory-type=XA_GENERIC, connectors=[in-vm],
entries=[java:/ConnectionFactory])
```

3. **InQueue** JMS キューからメッセージを読み取り、JBoss EAP 7.x サーバー上に設定されている **MigratedMessagesQueue** に転送する Jakarta Messaging ブリッジを作成してデプロイします。

```
/subsystem=messaging-activemq/jms-bridge=myBridge:add(add-messageID-in-
header=true,max-batch-time=100,max-batch-size=10,max-retries=-1,failure-retry-
interval=1000,quality-of-service=AT_MOST_ONCE,module=org.hornetq,source-
destination=jms/queue/InQueue,source-connection-
factory=jms/RemoteConnectionFactory,source-context=
[("java.naming.factory.initial"=>"org.wildfly.naming.client.WildFlyInitialContextFactory"),
("java.naming.provider.url"=>"http-remoting://legacy-host:8080")],target-
destination=jms/queue/MigratedMessagesQueue,target-connection-
factory=java:/ConnectionFactory)
```

これにより、JBoss EAP 8.0 サーバーの **message-activemq** サブシステムに次の **jms-bridge** 設定が作成されます。

```
<jms-bridge name="myBridge" add-messageID-in-header="true" max-batch-time="100" max-
batch-size="10" max-retries="-1" failure-retry-interval="1000" quality-of-
service="AT_MOST_ONCE">
    <source destination="jms/queue/InQueue" connection-
factory="jms/RemoteConnectionFactory">
        <source-context>
            <property name="java.naming.factory.initial"
value="org.wildfly.naming.client.WildFlyInitialContextFactory"/>
            <property name="java.naming.provider.url" value="http-remoting://legacy-host:8080"/>
        </source-context>
    </source>
    <target destination="jms/queue/MigratedMessagesQueue" connection-
factory="java:/ConnectionFactory"/>
</jms-bridge>
```

6.4.1.2.2. メッセージングデータの移行

メッセージングデータを Red Hat JBoss Enterprise Application Platform 8.0 から Red Hat JBoss Enterprise Application Platform 8.0 に移行するには、以下に示す手順に従います。

前提条件

- JBoss EAP 8.0 サーバーがインストールされ、実行されている。

手順

1. 以下の設定に提供する情報が正しいことを確認します。
   - キューおよびトピック名。
   - Java Naming and Directory Interface ルックアップの **java.naming.provider.url**。
2. ターゲット Jakarta Messaging 宛先が JBoss EAP 8.0 サーバーにデプロイされていることを確認してください。
3. 移行プロセスに関係する JBoss EAP 7 サーバーを含む、JBoss EAP 8.0 サーバーを起動します。

6.4.1.3. メッセージングフォルダーのデータをバックアップする

データの整合性を確保するために、サーバーがすでにメッセージを処理している場合は、変更を加える前にターゲットメッセージフォルダーをバックアップすることを推奨します。メッセージングフォルダーのデフォルトの場所は **EAP_HOME/standalone/data/activemq/** で確認できます。ただし、設定可能な場合もあります。メッセージングデータの場所が不明な場合は、次の管理 CLI コマンドを使用して確認できます。

手順

1. 次の管理 CLI コマンドを使用して、メッセージングデータの場所を確認します。

```
/subsystem=messaging-activemq/server=default/path=journal-directory:resolve-path
/subsystem=messaging-activemq/server=default/path=paging-directory:resolve-path
/subsystem=messaging-activemq/server=default/path=bindings-directory:resolve-path
/subsystem=messaging-activemq/server=default/path=large-messages-directory:resolve-path
```

注記

データをコピーする前に、必ずサーバーを停止してください。

2. それぞれの場所を特定した後、各メッセージングフォルダーを安全なバックアップの場所にコピーします。

6.4.2. Jakarta Messaging リソースアダプターの設定

サードパーティーの Jakarta Messaging プロバイダーで使用する汎用 Jakarta Messaging リソースアダプターを設定する方法が、Red Hat JBoss Enterprise Application Platform 8.0 で変更されました。詳細は、JBoss EAP 7.4 **メッセージングの設定** ガイドの 汎用 Java Message Service リソースアダプターのデプロイ を参照してください。

6.4.3. メッセージング設定の変更

Red Hat JBoss Enterprise Application Platform 7.0 では、**check-for-live-server** 属性を指定せずに **replication-primary** ポリシーを設定した場合、そのデフォルト値は **false** に設定されました。これは

JBoss EAP 7.1 で変更されました。**check-for-live-server** 属性のデフォルト値が **true** に設定されるようになりました。

以下は、**check-for-live-server** 属性を指定せずに **replication-primary** ポリシーを設定する管理 CLI コマンドの例です。

```
/subsystem=messaging-activemq/server=default/ha-policy=replication-primary:add(cluster-name=my-cluster,group-name=group1)
```

管理 CLI を使用してリソースを読み取ると、**check-for-live-server** 属性の値が **true** に設定されることに注意してください。

```
/subsystem=messaging-activemq/server=default/ha-policy=replication-primary:read-resource(recursive=true)
{
    "outcome" => "success",
    "result" => {
        "check-for-live-server" => true,
        "cluster-name" => "my-cluster",
        "group-name" => "group1",
        "initial-replication-sync-timeout" => 30000L
    },
    "response-headers" => {"process-state" => "reload-required"}
}
```

6.4.4. 組み込みブローカーメッセージング用の Galleon レイヤー

JBoss EAP 7 では、組み込みメッセージングブローカーはデフォルトのインストールの一部でした。JBoss EAP 8 では、この機能は、**embedded-activemq** と呼ばれる新しい Galleon レイヤーに追加されました。

この新しいレイヤーはデフォルト設定の一部ではないため、JBoss EAP にブローカーが組み込まれていることを信頼するユーザーは、それを明示的に設定に含める必要があります。

このレイヤーは、お客様が OpenShift 上の専用 AMQ クラスターを使用することが推奨されている場合でも、組み込みブローカーを備えた **messaging-activemq** サブシステムを提供します。また、このユースケースをサポートするために必要なソケットバインディングや必要な依存関係などの従属リソースもプロビジョニングします。

6.5. JBOSS EAP 8.0 のセキュリティー強化

JBoss EAP 8.0 以降では、レガシーセキュリティーサブシステムとレガシーセキュリティーレルムが使用できなくなったため、Elytron を使用する必要があります。Elytron のデフォルトは、JBoss Server Migration Tool を使用しなければ設定できません。そのため、レガシーセキュリティー設定は手動で移行する必要があります。

関連情報

- Elytron への移行

6.5.1. Vault の移行

Vault は JBoss EAP 8.0 から削除されました。気密性の高い文字列の保存には、**elytron** サブシステムによって提供される認証情報ストアを使用してください。

関連情報

- セキュアな vault およびプロパティーの移行
- Elytron のクレデンシャルとクレデンシャルストア

6.5.2. レガシーセキュリティーサブシステムとセキュリティーレルムの削除

レガシーセキュリティーサブシステムとレガシーセキュリティーレルムは、JBoss EAP 8.0 から削除されました。**elytron** サブシステムによって提供されるセキュリティーレルムを使用してください。

関連情報

- 認証設定の移行
- データベース認証設定の Elytron への移行
- 複合ストアの Elytron への移行
- キャッシングを使用するセキュリティードメインの Elytron への移行
- レガシープロパティーベースの設定の Elytron への移行
- LDAP 認証設定の Elytron への移行
- Kerberos 認証の Elytron への移行
- SSL 設定の移行
- アイデンティティーストアを使用したアプリケーションと管理インターフェイスの保護
- 複数のアイデンティティーストアを使用したアプリケーションと管理インターフェイスの保護

6.5.3. PicketLink サブシステムの削除

PicketLink サブシステムは JBoss EAP 8.0 から削除されました。PicketLink アイデンティティープロバイダーの代わりに Red Hat build of Keycloak を使用し、PicketLink サービスプロバイダーの代わりに Red Hat build of Keycloak SAML アダプターを使用してください。

関連情報

- PicketLink の削除
- Red Hat build of Keycloak

6.5.4. Red Hat build of Keycloak OIDC クライアントアダプターから JBoss EAP サブシステムへの移行

keycloak サブシステムは、JBoss EAP 8.0 ではサポートされず、**elytron-oidc-client** サブシステムに置き換えられます。移行は、デフォルトで JBoss Server Migration Tool が実行します。

関連情報

- **keycloak** サブシステムの移行

- JBoss EAP での OpenID Connect 設定

6.5.5. カスタムログインモジュールの移行

JBoss EAP 8.0 では、レガシーセキュリティーサブシステムが削除されました。**elytron** サブシステムでカスタムログインモジュールを引き続き使用するには、新しい Java Authentication and Authorization Service (JAAS) セキュリティーレルムと **jaas-realm** を使用します。

関連情報

- **elytron** サブシステムの JAAS レルム

6.5.6. FIPS モードの変更

JBoss EAP 7.1 以降では、開発目的で自己署名証明書の自動生成がデフォルトで有効になっています。FIPS モードで実行している場合は、自己署名証明書の自動作成を無効にするようにサーバーを設定します。設定しない場合は、サーバーの起動時に次のエラーが発生する可能性があります。

```
ERROR [org.xnio.listener] (default I/O-6) XNIO001007: A channel event listener threw an exception: java.lang.RuntimeException: WFLYDM0114: Failed to lazily initialize SSL context
...
Caused by: java.lang.RuntimeException: WFLYDM0112: Failed to generate self signed certificate
...
Caused by: java.security.KeyStoreException: Cannot get key bytes, not PKCS#8 encoded
```

関連情報

- 自動生成された自己署名証明書を使用したアプリケーションの SSL/TLS の有効化

6.6. MOD_CLUSTER 設定の変更

mod_cluster の静的プロキシーリストの設定は Red Hat JBoss Enterprise Application Platform 7.4 で変更されました。

JBoss EAP 7.4 以降、**proxy-list** 属性は非推奨になり、その後 JBoss EAP 8.0 で削除されました。

この属性は、アウトバウンドソケットバインディング名のリストである **proxies** 属性に置き換えられました。

この変更は、mod_cluster のアドバタイズを無効にするときなど、静的プロキシーリストを定義する方法に影響します。mod_cluster のアドバタイズを無効化する方法の詳細は、JBoss EAP 7.4 **設定ガイド** の mod_cluster のアドバタイズの無効化 を参照してください。

JBoss EAP 8.0 との互換性を確保するには、ユーザースクリプトとレガシーユーザー CLI スクリプトを次のように更新します。

- 非推奨の **ssl=configuration** を適切な **elytron-based** 設定に置き換えます。
- mod_cluster 設定パスを **/mod-cluster-config=CONFIGURATION** から **/proxy=default** に更新します。
- ユーザースクリプト内の動的ロードプロバイダーのパスを更新し、非推奨のパスを **provider=dynamic** に置き換えます。

- Undertow リスナーを参照する非推奨の **connector** 属性が削除されました。代わりに **listener** 属性を使用するようにユーザースクリプトを更新します。

mod_cluster 属性の詳細は、JBoss EAP 7.4 **設定ガイド** の ModCluster サブシステム属性 を参照してください。

6.7. 設定変更の表示

Red Hat JBoss Enterprise Application Platform 7 を使用すると、実行中のサーバーに加えられた設定変更を追跡できます。許可されたユーザーが行った設定変更の履歴を表示することもできます。

一方、JBoss EAP 7.0 では、**core-service** 管理 CLI コマンドを使用してオプションを設定し、最近の設定変更のリストを取得する必要がありました。

例: JBoss EAP 7.0 での設定変更の表示

```
/core-service=management/service=configuration-changes:add(max-history=10)
/core-service=management/service=configuration-changes:list-changes
```

JBoss EAP 7.1 には、稼働中のサーバーに追加された設定変更を追跡するよう設定できる新しい **core-management** サブシステムが導入されました。これは、JBoss EAP 7.1 以上で設定変更を設定および表示する推奨方法となります。

例: JBoss EAP 7.1 での設定変更の表示

```
/subsystem=core-management/service=configuration-changes:add(max-history=20)
/subsystem=core-management/service=configuration-changes:list-changes
```

JBoss EAP 7.1 で導入された新しい **core-management** サブシステムの使用に関する詳細は、JBoss EAP 7.4 **設定ガイド** の 設定変更の表示 を参照してください。

第7章 アプリケーションの移行における変更の理解

このセクションでは、アプリケーションを JBoss EAP 6.4 または 7.x から JBoss EAP 8.0 に移行するために必要な変更について説明します。

7.1. WEB サービスアプリケーションの変更

主に Apache CXF、Apache WSS4J、および Apache Santuario コンポーネントがアップグレードされ、JBossWS 5 は JBoss EAP 7 の Web サービスに新機能と改良されたパフォーマンスを提供します。JBoss EAP 8.0 は、JBossWS 7 を使用してその機能をサポートします。

7.1.1. JAX-RPC サポートの変更

XML ベースの RPC (JAX-RPC) は Java EE 6 で非推奨となり、Java EE 7 ではオプションとなりました。JBoss EAP 7 以降、サポートされなくなりました。JAX-RPC を使用するアプリケーションは、現在の Jakarta EE 標準 Web サービスフレームワークである Jakarta XML Web Services を使用するように移行する必要があります。

JAX-RPC web サービスの使用は、以下のいずれかの方法で特定できます。

- JAX-RPC マッピングファイルの存在。 これは、root 要素 **<java-wsdl-mapping>** のある XML ファイルです。
- **webservices.xml** XML 記述子ファイルの存在。このファイルには **<jaxrpc-mapping-file>** 子要素が含まれる **<webservice-description>** 要素があります。以下に JAX-RPC Web サービスを定義する **webservices.xml** 記述子ファイルの例を示します。

```
<webservices xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://www.ibm.com/webservices/xsd/j2ee_web_services_1_1.xsd" version="1.1">
 <webservice-description>
  <webservice-description-name>HelloService</webservice-description-name>
  <wsdl-file>WEB-INF/wsdl/HelloService.wsdl</wsdl-file>
  <jaxrpc-mapping-file>WEB-INF/mapping.xml</jaxrpc-mapping-file>
  <port-component>
   <port-component-name>Hello</port-component-name>
   <wsdl-port>HelloPort</wsdl-port>
   <service-endpoint-interface>org.jboss.chap12.hello.Hello</service-endpoint-interface>
   <service-impl-bean>
    <servlet-link>HelloWorldServlet</servlet-link>
   </service-impl-bean>
  </port-component>
 </webservice-description>
</webservices>
```

- **ejb-jar.xml** ファイルの存在。 これには、JAX-RPC マッピングファイルを参照する **<service-ref>** が含まれます。

7.1.2. Apache CXF Spring Web サービスの変更

以前のリリースの JBoss EAP では、エンドポイントデプロイメントアーカイブを **jbossws-cxf.xml** 設定ファイルに含め、JBossWS と Apache CXF の統合をカスタマイズすることができました。このユースケースの 1 つが、Apache CXF バスで Web サービスクライアントおよびサーバーエンドポイントのイ

ンターセプターチェインを設定することでした。この統合には、JBoss EAP サーバーに Spring をデプロイする必要がありました。

Spring 統合は JBoss EAP 8 ではサポートされなくなりました。**jbossws-cxf.xml** 記述子設定ファイルが含まれるアプリケーションを編集し、このファイルに定義されているカスタム設定を置き換える必要があります。JBoss EAP 7 でも Apache CXF API に直接アクセスすることはできますが、アプリケーションは移植できないことに注意してください。

可能な場合は、Spring のカスタム設定を新しい JBossWS 記述子設定オプションに置き換えることが推奨されます。この JBossWS 記述子ベースの方法では、クライアントエンドポイントコードを編集する必要がなく、同様の機能を提供できます。場合によっては、Spring を Context and Dependency Injection (コンテキストと依存性の注入、CDI) に置き換えることができます。

7.1.2.1. Apache CXF インターセプター

JBossWS 記述子は、クライアントエンドポイントコードを編集せずにインターセプターを宣言できる新しい設定オプションを提供します。**cxf.interceptors.in** および **cxf.interceptors.out** プロパティーのインターセプタークラス名のリストを指定して、事前定義されたクライアントおよびエンドポイント設定内でインターセプターを宣言します。

以下は、これらのプロパティーを使用してインターセプターを宣言する **jaxws-endpoint-config.xml** ファイルの例です。

```
<?xml version="1.0" encoding="UTF-8"?>
<jaxws-config xmlns="urn:jboss:jbossws-jaxws-config:5.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:jakartaee="https://jakarta.ee/xml/ns/jakartaee"
  xsi:schemaLocation="urn:jboss:jbossws-jaxws-config:5.0 schema/jbossws-jaxws-config_5_0.xsd">
  <endpoint-config>
    <config-name>org.jboss.test.ws.jaxws.cxf.interceptors.EndpointImpl</config-name>
    <property>
      <property-name>cxf.interceptors.in</property-name>
      <property-
value>org.jboss.test.ws.jaxws.cxf.interceptors.EndpointInterceptor,org.jboss.test.ws.jaxws.cxf.interceptos.FooInterceptor</property-value>
    </property>
    <property>
      <property-name>cxf.interceptors.out</property-name>
      <property-value>org.jboss.test.ws.jaxws.cxf.interceptors.EndpointCounterInterceptor</property-
value>
    </property>
  </endpoint-config>
</jaxws-config>
```

7.1.2.2. Apache CXF の機能

JBossWS 記述子を使用すると、**cxf.features** プロパティーの機能クラス名のリストを指定して、事前定義のクライアントおよびエンドポイント設定内で機能を宣言できます。

以下は、このプロパティーを使用して機能を宣言する **jaxws-endpoint-config.xml** ファイルの例です。

```
<?xml version="1.0" encoding="UTF-8"?>
<jaxws-config xmlns="urn:jboss:jbossws-jaxws-config:5.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:jakartaee="https://jakarta.ee/xml/ns/jakartaee"
```

```
  xsi:schemaLocation="urn:jboss:jbossws-jaxws-config:5.0 schema/jbossws-jaxws-config_5_0.xsd">
  <endpoint-config>
    <config-name>Custom FI Config</config-name>
    <property>
      <property-name>cxf.features</property-name>
      <property-value>org.apache.cxf.feature.FastInfosetFeature</property-value>
    </property>
  </endpoint-config>
</jaxws-config>
```

7.1.2.3. Apache CXF HTTP トランスポート

Apache CXF では、**org.apache.cxf.transport.http.HTTPConduit** オプションを指定すると HTTP トランスポートの設定を実行できます。JBossWS の統合により、以下のように Apache CXF API を使用して conduit がプログラム的に編集されます。

```
import org.apache.cxf.frontend.ClientProxy;
import org.apache.cxf.transport.http.HTTPConduit;
import org.apache.cxf.transports.http.configuration.HTTPClientPolicy;

// Set chunking threshold before using a JAX-WS port client
...
HTTPConduit conduit = (HTTPConduit)ClientProxy.getClient(port).getConduit();
HTTPClientPolicy client = conduit.getClient();

client.setChunkingThreshold(8192);
...
```

また、システムプロパティーを設定すると Apache CXF **HTTPConduit** のデフォルト値を制御および上書きできます。

プロパティー	型	説明
cxf.client.allowChunking	Boolean	チャンキングを使用してリクエストを送信するかどうかを指定します。
cxf.client.chunkingThreshold	Integer	非チャンキングからチャンキングモードに切り替えるしきい値を設定します。
cxf.client.connectionTimeout	Long	接続タイムアウトをミリ秒単位で設定します。
cxf.client.receiveTimeout	Long	受信タイムアウトをミリ秒単位で設定します。
cxf.client.connection	String	**Keep-Alive** または **close** 接続タイプを使用するかどうかを指定します。
cxf.tls-client.disableCNCheck	Boolean	CN ホスト名チェックを無効化するかどうかを指定します。

7.1.3. WS-Security の変更

このセクションでは、JBoss EAP 6.4 および JBoss EAP 7.0 でのアプリケーションの WS-Security のさまざまな変更について説明します。

- アプリケーションに、**org.apache.ws.security.WSPasswordCallback** クラスにアクセスするカスタムコールバックハンドラーが含まれている場合、このクラスは **org.apache.wss4j.common.ext** パッケージに移動されたため注意してください。
- **org.apache.ws.security.saml.ext** パッケージの SAML bean オブジェクトのほとんどは、**org.apache.wss4j.common.saml** パッケージに移動されました。
- RSA v1.5 キートランスポートおよびすべての関連アルゴリズムの使用は、デフォルトでは許可されていません。
- これまで、セキュリティートークンサービス (STS) は **onBehalfOf** トークンのみを検証しました。**ActAs** トークンも検証するようになりました。そのため、**ActAs** トークンに提供される **UsernameToken** に有効なユーザー名とパスワードを指定する必要があります。
- SAML Bearer トークンには内部署名が必要になりました。署名の検証を有効または無効にするため、**org.apache.wss4j.dom.validate.SamlAssertionValidator** クラスに **setRequireBearerSignature()** メソッドが含まれるようになりました。

7.1.4. JBoss モジュール構造の変更

cxf-api および **cxf-rt-core** JAR が 1 つの **cxf-core** JAR に統合されました。そのため、JBoss EAP の **org.apache.cxf** モジュールに **cxf-core** JAR が含まれるようになり、これまでのリリースよりも多いクラスが公開されました。

7.1.5. Bouncy Castle の要件および変更

XML/WS-Security での対称暗号化で Galois/Counter Mode (GCM) を用いた AES 暗号化を使用したい場合、BouncyCastle Security Provider が必要になります。

JBoss EAP 7 以降、これは **org.bouncycastle** モジュールに含まれており、JBossWS はそのクラスローダーを信頼して BouncyCastle セキュリティープロバイダーを取得して使用できるようになりました。そのため、現在の JVM に BouncyCastle を静的にインストールする必要がなくなりました。コンテナーの外部で実行しているアプリケーションの場合、BouncyCastle ライブラリーをクラスパスに追加すると JBossWS はこのセキュリティープロバイダーを使用できます。

この動作を無効にするには、**jaxws-endpoint-config.xml** デプロイメント記述子ファイル (サーバーの場合) または **jaxws-client-config.xml** 記述子ファイル (クライアントの場合) で **org.jboss.ws.cxf.noLocalBC** プロパティーの値を **true** に設定します。

JBoss EAP に同梱されるバージョンでないものを使用したい場合は、BouncyCastle を静的に JVM にインストールできます。この場合、静的にインストールされた BouncyCastle Security Provider はクラスパスに存在するプロバイダーよりも優先的に選択されます。問題を回避するには、BouncyCastle 1.72 以降を使用する必要があります。

7.1.6. Apache CXF バス選択ストラテジー

コンテナー内で実行されているクライアントのデフォルトのバス選択ストラテジーが **THREAD_BUS** から **TCCL_BUS** に変更されました。コンテナー外部で実行されているクライアントのデフォルトストラテジーは **THREAD_BUS** で、変更はありません。以下の方法の 1 つを使用すると、以前のリリースでの動作を復元できます。

- **org.jboss.ws.cxf.jaxws-client.bus.strategy** システムプロパティーの値を **THREAD_BUS** に設定して JBoss EAP サーバーを起動します。

- クライアントコードで選択ストラテジーを明示的に設定します。

7.1.7. WebServiceRef の Jakarta XML Web Services 2.2 要件

コンテナーは、コンストラクターで WebServiceFeature クラスが引数として含まれる Jakarta XML Web Services 2.2 スタイルのコンストラクターを使用して、web サービス参照に注入されるクライアントをビルドする必要があります。JBossWS 4 が同梱される JBoss EAP 6.4 はこの要件を隠します。JBossWS 5 を含む JBoss EAP 7 以降、この要件は隠蔽されていません。これは、コンテナーによって注入されるユーザー提供のサービスクラスは、1 つ以上の **WebServiceFeature** 引数を含む **jakarta.xml.ws.Service** コンストラクターを使用するように既存のコードを更新することにより、Jakarta XML Web Services 2.2 以降を実装する必要があることを示しています。

```
protected Service(URL wsdlDocumentLocation,
      QName serviceName,
      WebServiceFeature... features)
```

7.1.8. IgnoreHttpsHost CN チェックの変更

以前のリリースでは、システムプロパティー **org.jboss.security.ignoreHttpsHost** を **true** に設定すると、証明書にあるサービスの一般名 (CN) に対する HTTPS URL ホスト名チェックを無効にすることができました。このシステムプロパティー名は **cxf.tls-client.disableCNCheck** に置き換えられました。

7.1.9. サーバー側設定およびクラスローディング

サービスエンドポイントおよびサービスクライアントハンドラーへのインジェクションが有効になったため、**org.jboss.as.webservices.server.integration** JBoss モジュールから自動的にハンドラークラスをロードすることができなくなりました。アプリケーションが事前定義の設定に依存する場合、デプロイメントの新しいモジュール依存関係を明示的に定義する必要があることがあります。詳細は、明示的なモジュール依存関係の移行 を参照してください。

7.1.10. Java Endorsed Standards Override Mechanism が非推奨となる

Java Endorsed Standards Override Mechanism は JDK 1.8_40 では非推奨になり、JDK 9 では削除される予定です。これは、JAR を JRE 内の endorsed ディレクトリーに置くことで、デプロイされたアプリケーションすべてがライブラリーを利用できるメカニズムです。

JBoss EAP 7 リリース以降、アプリケーションが Apache CXF の JBossWS 実装を使用している場合は、必要な依存関係が正しい順序で確実に追加されるようになり、ユーザーはこの変更による影響を受けることはありません。アプリケーションが Apache CXF に直接アクセスする場合、アプリケーションデプロイメントの一部として JBossWS の依存関係の後に Apache CXF の依存関係を提供する必要があります。

7.1.11. EAR アーカイブでの記述子の仕様

以前のリリースの JBoss EAP では、Jakarta Enterprise Beans Web サービスデプロイメントの **jboss-webservices.xml** デプロイメント記述子ファイルを JAR アーカイブの **META-INF/** ディレクトリーまたは POJO Web サービスデプロイメントの **WEB-INF/** ディレクトリーと、WAR アーカイブにバンドル化された Jakarta Enterprise Beans Web サービスエンドポイントに設定することができました。

JBoss EAP 7 以降では、EAR アーカイブの **META-INF/** ディレクトリーに **jboss-webservices.xml** デプロイメント記述子ファイルを設定できるようになりました。**jboss-webservices.xml** ファイルが EAR アーカイブおよび JAR または WAR アーカイブで見つかった場合、JAR または WAR の **jboss-webservices.xml** ファイル内の設定データは、EAR 記述子ファイル内の対応するデータをオーバーライドします。

7.2. リモート URL コネクターとポートの更新

JBoss EAP 7 以降、デフォルトのコネクターが **remote** から **http-remoting** に変更され、デフォルトのリモート接続ポートが **4447** から **8080** に変更されました。デフォルト設定の JNDI プロバイダー URL は **remote://localhost:4447** から **http-remoting://localhost:8080** に変更になりました。

7.3. メッセージングアプリケーションの変更

このセクションでは、JBoss EAP 7 でのメッセージングアプリケーションのさまざまな変更について説明します。さらに、次の方法について詳しく学ぶことができます。

- Jakarta Messaging デプロイメント記述子の変更
- 外部 Jakarta Messaging クライアントの更新
- 非推奨のアドレス設定属性の置き換え
- 必要なメッセージングアプリケーションの変更の設定

7.3.1. Jakarta Messaging デプロイメント記述子の置き換えまたは更新

JBoss EAP 7 以降、命名パターン **-jms.xml** で識別される独自の HornetQ メッセージングリソースデプロイメント記述子ファイルは機能しません。以下は、JBoss EAP 6 の Java Message Service リソースデプロイメント記述子ファイルの例です。

```
<?xml version="1.0" encoding="UTF-8"?>
<messaging-deployment xmlns="urn:jboss:messaging-deployment:1.0">
  <hornetq-server>
    <jms-destinations>
      <jms-queue name="testQueue">
        <entry name="queue/test"/>
        <entry name="java:jboss/exported/jms/queue/test"/>
      </jms-queue>
      <jms-topic name="testTopic">
        <entry name="topic/test"/>
        <entry name="java:jboss/exported/jms/topic/test"/>
      </jms-topic>
    </jms-destinations>
  </hornetq-server>
</messaging-deployment>
```

以前のリリースのアプリケーションで **-jms.xml** Java Message Service デプロイメント記述子を使用した場合は、Jakarta EE platform の Resource Definition and Configuration セクションで指定されている標準デプロイメント記述子を使用するようにアプリケーションを変換するか、**messaging-activemq-deployment** スキーマを使用するようにデプロイメント記述子を更新できます。記述子の更新を選択した場合、以下の変更を加える必要があります。

- ネームスペースを "urn:jboss:messaging-deployment:1.0" から "urn:jboss:messaging-activemq-deployment:1.0" に変更します。
- **<hornetq-server>** 要素名を **<server>** に変更します。

編集後のファイルは以下の例のようになるはずです。

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

```
<messaging-deployment xmlns="urn:jboss:messaging-activemq-deployment:1.0">
  <server>
    <jms-destinations>
      <jms-queue name="testQueue">
        <entry name="queue/test"/>
        <entry name="java:jboss/exported/jms/queue/test"/>
      </jms-queue>
      <jms-topic name="testTopic">
        <entry name="topic/test"/>
        <entry name="java:jboss/exported/jms/topic/test"/>
      </jms-topic>
    </jms-destinations>
  </server>
</messaging-deployment>
```

7.3.2. HornetQ API の置換

JBoss EAP 6 には **org.hornetq** モジュールが含まれ、これによりアプリケーションのソースコードで HornetQ API を使用できました。

JBoss EAP 7 では、Apache ActiveMQ Artemis が HornetQ を置き換えるため、HornetQ API を使用したコードは Apache ActiveMQ Artemis API を使用するように移行する必要があります。この API のライブラリーは、**org.apache.activemq.artemis** モジュールに含まれています。

ActiveMQ Artemis は HornetQ の進化版なので、概念の多くは継続して適用されます。

7.3.3. 非推奨のアドレス設定属性の置き換え

auto-create-jms-queues、**auto-delete-jms-queues**、**auto-create-jms-topics**、および **auto-delete-jms-topics** 属性を使用した、トピックやキューの自動作成および自動削除機能は、JBoss EAP 7 では部分的にのみ実装され、完全に設定できません。これらの属性は非推奨となり、テクノロジープレビューとしてのみ提供されるためサポート対象外となります。

非推奨となったこれらの属性を以下の代替となる属性に置き換える必要があります。

注記

JBoss EAP 8.0 以降、非推奨の属性はこの機能を設定しなくなり、有効化されません。代替となる属性もサポートされません。代替となる属性は、ベストエフォートベースで、移行に対応するための方法としてのみ提供されます。

非推奨となった属性	代替となる属性
auto-create-jms-queues	auto-create-queues
auto-delete-jms-queues	auto-delete-queues
auto-create-jms-topics	auto-create-addresses
auto-delete-jms-topics	auto-delete-addresses

JBoss EAP 6 では、デフォルトのアドレス設定属性は **false** に設定されていました。JBoss EAP 7 以降、置換属性はデフォルトで **true** に設定されます。

JBoss EAP 6 の動作を保持する場合は、代替属性を **false** に設定する必要があります。

これらの代替属性の詳細は、JBoss EAP 7.4 **メッセージングの設定** ガイドの アドレス設定の属性 を参照してください。

7.3.4. JBoss EAP 7 に必要なメッセージングアプリケーションの変更

JBoss EAP 7.2 より、クライアントアプリケーションが直接 Artemis クライアント JAR (**artemis-jms-client**、**artemis-commons**、**artemis-core-client**、**artemis-selector** など) に依存する場合は **wildfly-client-properties** の **pom.xml** ファイルに以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.jboss.eap</groupId>
  <artifactId>wildfly-client-properties</artifactId>
</dependency>
```

これにより、**JBEAP-15889** に記載されているように、旧バージョンの JBoss EAP 7 クライアントから **message.getJMSReplyTo()** を呼び出すときに JMSRuntimeException が発生しないようにします。

7.4. JAKARTA RESTFUL WEB サービスと RESTEASY アプリケーションの変更

JBoss EAP 6 は、JAX-RS 1.x の実装であった RESTEasy 2 をバンドルしました。

JBoss EAP 7 および JBoss EAP 7.1 には、JSR 339: JAX-RS 2.0: The Java API for RESTful Web Services 仕様で定義されている JAX-RS 2.0 の実装である RESTEasy 3.0.x が含まれていました。

JBoss EAP 7.4 には、Jakarta RESTful Web Services 2.1 の実装である RESTEasy 3.15 が含まれています。このリリースでは、JDK 11 のサポートも追加されています。本リリースは RESTEasy 4 の主な機能の一部を提供しますが、ベースは RESTEasy 3.0 で、後方互換性を完全に維持します。そのため、RESTEasy 3.0 から RESTEasy 3.15 への移行の際は、問題がほとんど生しないはずです。Java API for RESTEasy 3.15 の詳細は、RESTEasy Jakarta RESTful Web Services 3.15.0.Final API を参照してください。

JBoss EAP 8.0 は、Jakarta RESTful Web Services 3.1 specification を実装する RESTEasy 6.2 のサポートを提供します。

JBoss EAP 6.4 から移行する場合、JBoss EAP に含まれる Jackson のバージョンが変更されたことに注意してください。JBoss EAP 6.4 には Jackson 1.9.9 が含まれていました。JBoss EAP 7 以上には Jackson 2.6.3 以上が含まれるようになりました。

このセクションでは、これらの変更が RESTEasy または Jakarta RESTful Web サービスを使用するアプリケーションにどのような影響を与える可能性があるかについて説明します。

7.4.1. RESTEasy の非推奨クラス

インターセプターおよび MessageBody クラス
JSR 311: JAX-RS: The Java™ API for RESTful Web Services にはインターセプターフレームワークが含まれなかったため、RESTEasy 2 によって提供されました。JSR 339: JAX-RS 2.0: The Java API for RESTful Web Services は、公式のインターセプターおよびフィルターフレームワークを導入したため、

RESTEasy 2 に含まれるインターセプターフレームワークは現在非推奨となり、RESTEasy 3.x の Jakarta REST 準拠インターセプター機能に置き換えられました。関連するインターフェイスは、**jakarta.ws.rs.api** モジュールの **jakarta.ws.rs.ext** パッケージで定義されています。

以下のプロバイダーは JBoss EAP 8.0 で削除されました。

- **org.jboss.resteasy:resteasy-jackson-provider**
- **org.jboss.resteasy:resteasy-jettison-provider**
- **org.jboss.resteasy:resteasy-yaml-provider**

以下は Jakarta RESTful Web Services に置き換えられたため、JBoss EAP 8.0 では削除されました。

- **@Suspend** と **org.jboss.resteasy.spi.AsynchronousResponse** は削除され、それぞれ **@Suspended** と **javax.ws.rs.container.AsyncResponse** に置き換えられました。
- **StringConverter** は **ParamConverter** に置き換えられます。
- **org.jboss.resteasy.plugins.providers.SerializableProvider** は非推奨となり、削除されました。
- RESTEasy 3.x で非推奨となった次のインターセプターインターフェイスは削除されました。
  - **org.jboss.resteasy.spi.interception.PreProcessInterceptor**
  - **org.jboss.resteasy.spi.interception.PostProcessInterceptor**
  - **org.jboss.resteasy.spi.interception.ClientExecutionInterceptor**
  - **org.jboss.resteasy.spi.interception.ClientExecutionContext**
  - **org.jboss.resteasy.spi.interception.AcceptedByMethod**
- **org.jboss.resteasy.spi.interception.PreProcessInterceptor** インターフェイスは、RESTEasy 3.x の **jakarta.ws.rs.container.ContainerRequestFilter** インターフェイスに置き換えられました。
- 以下のインターフェイスとクラスは、RESTEasy 3.x および JBoss EAP 8.0 から削除されました。
  - **org.jboss.resteasy.spi.interception.MessageBodyReaderInterceptor**
  - **org.jboss.resteasy.spi.interception.MessageBodyWriterInterceptor**
  - **org.jboss.resteasy.spi.interception.MessageBodyWriterContext**
  - **org.jboss.resteasy.spi.interception.MessageBodyReaderContext**
  - **org.jboss.resteasy.core.interception.InterceptorRegistry**
  - **org.jboss.resteasy.core.interception.InterceptorRegistryListener**
  - **org.jboss.resteasy.core.interception.ClientExecutionContextImpl**
- **org.jboss.resteasy.spi.interception.MessageBodyWriterInterceptor** インターフェイスは、**jakarta.ws.rs.ext.WriterInterceptor** インターフェイスに置き換えられました。

- さらに、**jakarta.ws.rs.ext.MessageBodyWriter** インターフェイスに対する一部の変更は、JAX-RS 1.x に関して下位互換性がない可能性があります。アプリケーションが JAX-RS 1.x を使用した場合はアプリケーションコードを確認し、エンドポイントに **@Produces** または **@Consumes** を定義するようにしてください。この定義を怠ると、以下のようなエラーが発生することがあります。

```
org.jboss.resteasy.core.NoMessageBodyWriterFoundFailure: Could not find
MessageBodyWriter for response object of type: <OBJECT> of media type:
```

以下に、このエラーの原因となる REST エンドポイントの例を示します。

```
@Path("dates")
public class DateService {

    @GET
    @Path("daysuntil/{targetdate}")
    public long showDaysUntil(@PathParam("targetdate") String targetDate) {
        DateLogger.LOGGER.logDaysUntilRequest(targetDate);
        final long days;

        try {
            final LocalDate date = LocalDate.parse(targetDate, DateTimeFormatter.ISO_DATE);
            days = ChronoUnit.DAYS.between(LocalDate.now(), date);
        } catch (DateTimeParseException ex) {
          // ** DISCLAIMER **. This example is contrived.
          throw new
WebApplicationException(Response.status(400).entity(ex.getLocalizedMessage()).type(Media
Type.TEXT_PLAIN)
                .build());
        }
        return days;
    }
}
```

この問題を解決するには、次のように、**jakarta.ws.rs.Produces** のインポートと **@Produces** アノテーションを追加します。

```
...
import jakarta.ws.rs.Produces;
...

@Path("dates")
public class DateService {

    @GET
    @Path("daysuntil/{targetdate}")
    @Produces(MediaType.TEXT_PLAIN)
    public long showDaysUntil(@PathParam("targetdate") String targetDate) {
        DateLogger.LOGGER.logDaysUntilRequest(targetDate);
        final long days;

        try {
            final LocalDate date = LocalDate.parse(targetDate, DateTimeFormatter.ISO_DATE);
            days = ChronoUnit.DAYS.between(LocalDate.now(), date);
        } catch (DateTimeParseException ex) {
```

```
            // ** DISCLAIMER **. This example is contrived.
            throw new
WebApplicationException(Response.status(400).entity(ex.getLocalizedMessage()).type(Media
Type.TEXT_PLAIN)
                    .build());
            }
            return days;
        }
}
```

注記

RESTEasy の以前のリリースのすべてのインターセプターは、新しい Jakarta REST フィルターおよびインターセプターインターフェイスと並行して実行できます。

関連情報

- RESTEasy インターセプター
- RESTEasy Jakarta RESTful Web Services 3.15.0.Final API

クライアント API
resteasy-jaxrs の RESTeasy クライアントフレームワークは、JBoss EAP 7.0 で JAX-RS 2.0 準拠の **resteasy-client** モジュールに置き換えられました。そのため、RESTEasy クライアント API クラスおよびメソッドの中には非推奨となっているものもあります。

- 以下のクラスは JBoss EAP 8.0 から削除されました。
  - **org.jboss.resteasy.client.ClientRequest**
  - **org.jboss.resteasy.client.ClientRequestFactory**
  - **org.jboss.resteasy.client.ClientResponse**
  - **org.jboss.resteasy.client.ProxyBuilder**
  - **org.jboss.resteasy.client.ProxyConfig**
  - **org.jboss.resteasy.client.ProxyFactory**
- **org.jboss.resteasy.client.ClientResponseFailure** 例外、**org.jboss.resteasy.client.ClientExecutor** インターフェイスおよび **org.jboss.resteasy.client.EntityTypeFactory** インターフェイスも非推奨になりました。
- **org.jboss.resteasy.client.ClientRequest** クラスおよび **org.jboss.resteasy.client.ClientResponse** クラスを、それぞれ **org.jboss.resteasy.client.jaxrs.ResteasyClient** および **jakarta.ws.rs.core.Response** に置き換える必要があります。
  以下は RESTEasy 2.3.x の RESTEasy クライアントでリンクヘッダーを送信する例です。

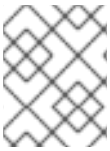
```
ClientRequest request = new ClientRequest(generateURL("/linkheader/str"));
request.addLink("previous chapter", "previous", "http://example.com/TheBook/chapter2",
null);
ClientResponse response = request.post();
LinkHeader header = response.getLinkHeader();
```

以下は RESTEasy 3 の RESTEasy クライアントで上記と同じタスクを実行する例です。

```
ResteasyClient client = new ResteasyClientBuilder().build();
Response response = client.target(generateURL("/linkheader/str")).request()
    .header("Link", "<http://example.com/TheBook/chapter2>; rel=\"previous\";
title=\"previous chapter\"").post(Entity.text(new String()));
jakarta.ws.rs.core.Link link = response.getLink("previous");
```

Jakarta REST Web サービスと対話する外部 Jakarta REST RESTEasy クライアントの例については、**resteasy-jaxrs-client** クイックスタートを参照してください。

- **org.jboss.resteasy.client.cache** パッケージのクラスおよびインターフェイスも非推奨になりました。これらは **org.jboss.resteasy.annotations.cache** パッケージ内の同等のクラスとインターフェイスに置き換えられました。



注記

org.jboss.resteasy.client.jaxrs API クラスの詳細は、RESTEasy Jakarta REST JavaDoc を参照してください。

StringConverter
org.jboss.resteasy.spi.StringConverter クラスは、RESTEasy 3.x および JBoss EAP 8.0 で非推奨になりました。この機能は、Jakarta REST jakarta.ws.rs.ext.ParamConverterProvider クラスを使用して置き換えることができます。

7.4.2. 削除または保護された RESTEasy クラス

ResteasyProviderFactory Add メソッド
RESTEasy 3.0 では、**org.jboss.resteasy.spi.ResteasyProviderFactory** **add()** メソッドのほとんどが削除または保護されています。たとえば、**addBuiltInMessageBodyReader()** および **addBuiltInMessageBodyWriter()** メソッドは削除され**addMessageBodyReader()** および **addMessageBodyWriter()** メソッドは保護されました。

現時点では、**registerProvider()** と **registerProviderInstance()** のメソッドを使用してください。

RESTEasy 3 から削除された他のクラス
Jakarta REST メソッドへの応答をサーバー上にキャッシュする必要があることを指定した **@org.jboss.resteasy.annotations.cache.ServerCached** アノテーションは RESTEasy 3 から削除されたため、アプリケーションコードから削除する必要があります。

7.4.3. 追加の RESTEasy 変更

このセクションでは、JBoss EAP の RESTEasy への追加的な変更点について説明します。

SignedInput および SignedOuput

- **resteasy-crypto** の **SignedInput** および **SignedOutput** では、**Content-Type** を **Request** または **Response** オブジェクトのいずれかで **multipart/signed** に設定する必要があります。 そうでない場合は、**@Consumes** または **@Produces** アノテーションを使用する必要があります。

- **SignedOutput** および **SignedInput** を使用すると、**@Produces** または **@Consumes** アノテーションにタイプを設定することで、**application/pkcs7-signature** MIME タイプ形式をバイナリー形式で返すことができます。

- **@Produces** または **@Consumes** が **text/plain** MIME タイプの場合、**SignedOutput** は base64 でエンコードされ、文字列として送信されます。

セキュリティーフィルター
@RolesAllowed、**@PermitAll**、および **@DenyAll** のセキュリティーフィルターは、"401 Unauthorized" ではなく "403 Forbidden" を返すようになりました。

クライアント側のフィルター
RESTEasy 3.0 より前のリリースから RESTEasy クライアント API を使用している場合は、JAX-RS 2.0 で導入されたクライアント側のフィルターはバインドされず、実行されません。

非同期 HTTP サポート
JAX-RS 2.0 仕様は、**@Suspended** アノテーションと **AsynResponse** インターフェイスを使用した非同期 HTTP サポートを追加したため、非同期 HTTP の RESTEasy プロプライエタリー API は非推奨となりました。今後の RESTEasy リリースで削除される可能性があります。非同期 Tomcat と非同期 JBoss Web モジュールもサーバーインストールから削除されています。 Servlet 3.0 コンテナーまたはそれ以降を使用していない場合、非同期 HTTP サーバー側の処理がシミュレートされ、同一リクエストスレッドで同期的に実行されます。

サーバー側のキャッシュ
サーバー側のキャッシュ設定が変更されました。詳細は、RESTEasy のドキュメント を参照してください。

YAML プロバイダーの設定変更
以前のリリースの JBoss EAP では、RESTEasy YAML プロバイダー設定はデフォルトで有効になっていました。これは JBoss EAP 7 で変更になりました。YAML プロバイダーがデフォルトで無効化されるようになりました。アンマーシャリングで RESTEasy によって使用される **SnakeYAML** ライブラリーにセキュリティー上の問題があるため、YAML プロバイダーの使用はサポートされず、アプリケーションで明示的に有効にする必要があります。アプリケーションで YAML プロバイダーを有効にし、Maven 依存関係を追加する方法は、JBoss EAP 7.4 **Web サービスアプリケーションの開発**の YAML プロバイダー を参照してください。

Content-Type ヘッダーのデフォルトの文字セット UTF-8
JBoss EAP 7.1 より、デフォルトで **resteasy.add.charset** パラメーターが **true** に設定されています。リソースメソッドが明示的な文字セットなしで **text/*** または **application/xml*** メディアタイプを返すときに、返された content-type ヘッダーに **charset=UTF-8** を追加したくない場合は、**resteasy.add.charset** パラメーターを **false** に設定できます。

テキストメディアタイプと文字セットの詳細は、JBoss EAP 7.4 **Web サービスアプリケーションの開発**の テキストメディアタイプおよび文字セット を参照してください。

SerializableProvider
信用できないソースから Java オブジェクトをデシリアライズすることは危険です。そのため、JBoss EAP 7 以降では **org.jboss.resteasy.plugins.providers.SerializableProvider** クラスがデフォルトで無効となり、このプロバイダーの使用は推奨されません。

リソースメソッドへのリクエストの一致
RESTEasy 3 では、JAX-RS 仕様の定義どおりに、一致ルールの実装に改善および修正が加えられました。特に、サブリソースメソッドおよびサブリソースロケーターのあいまいな URI の処理方法が変更されました。

RESTEasy 2 では、同じ URI を持つ別のサブリソースが存在していても、サブリソースロケーターが正常に実行される可能性がありました。仕様上ではこの挙動は適切ではありません。

RESTEasy 3 では、サブリソースおよびサブリソースロケーターのあいまいな URI が存在する場合、サブリソースの呼び出しには成功しますが、サブリソースロケーターの呼び出しは HTTP ステータス **405 Method Not Allowed** のエラーによって失敗します。

以下の例には、サブリソースメソッドおよびサブリソースロケーターのあいまいな **@Path** アノテーションが含まれています。エンドポイント **anotherResource** および **anotherResourceLocator** 両方の URI は同じであることに注目してください。この 2 つのエンドポイントの違いは、**anotherResource** メソッドは REST 動詞である **POST** に関連付けられていることです。**anotherResourceLocator** メソッドに関連付けられている REST 動詞はありません。仕様上では、REST 動詞を持つエンドポイント (この場合は **anotherResource** メソッド) が常に選択されます。

```
@Path("myResource")
public class ExampleSubResources {
    @POST
    @Path("items")
    @Produces("text/plain")
    public Response anotherResource(String text) {
        return Response.ok("ok").build();
    }

    @Path("items")
    @Produces("text/plain")
    public SubResource anotherResourceLocator() {
        return new SubResource();
    }
}
```

7.4.4. RESTEasy SPI の変更

RESTEasy SPI プロバイダーは JBoss EAP 8 で削除されました。

SPI 例外
すべての SPI 失敗例外は非推奨となり、内部的には使用されません。これらは、対応する Jakarta REST 例外に置き換えられました。

非推奨の例外	jaxrs-api モジュールでの代替の例外
org.jboss.resteasy.spi.ForbiddenException	jakarta.ws.rs.ForbiddenException
org.jboss.resteasy.spi.MethodNotAllowedException	jakarta.ws.rs.NotAllowedException
org.jboss.resteasy.spi.NotAcceptableException	jakarta.ws.rs.NotAcceptableException
org.jboss.resteasy.spi.NotFoundException	jakarta.ws.rs.NotFoundException
org.jboss.resteasy.spi.UnauthorizedException	jakarta.ws.rs.NotAuthorizedException
org.jboss.resteasy.spi.UnsupportedMediaTypeException	jakarta.ws.rs.NotSupportedException

InjectorFactory および Registry
InjectorFactory および **Registry** SPI が変更されました。ドキュメントに従ってサポートされるように RESTEasy を使用する場合は、問題はありません。

7.4.5. Jackson プロバイダーの変更

JBoss EAP 6.4 に含まれる Jackson のバージョンが変更されました。JBoss EAP 7 以降、Jackson プロバイダーは **resteasy-jackson-provider** から **resteasy-jackson2-provider** に変更されました。

resteasy-jackson2-provider へのアップグレードにはいくつかのパッケージ変更が必要になります。たとえば、Jackson アノテーションパッケージは **org.codehaus.jackson.annotate** から **com.fasterxml.jackson.annotation** に変更されました。

7.4.6. Spring RESTEasy 統合の変更

JBoss EAP 8.0 は RESTEasy 6.2 のサポートを提供します。JBoss EAP 8.0 で Spring 6.0 フレームワークを使用する予定の場合は、Java 17 を使用する必要があります。

Spring 4.0 フレームワークには、Java 8 のサポートが導入されました。Spring と RESTEasy 3.x 統合を使用する場合は、使用するデプロイメントで最小 Spring バージョンに 4.2.x を指定してください。 これは JBoss EAP 7 がサポートする安定性のある最も早期のバージョンです。

7.4.7. RESTEasy Jettison JSON プロバイダーの変更

RESTEasy Jettison JSON プロバイダーは JBoss EAP 7 以降非推奨となり、デフォルトでデプロイメントに追加されなくなりました。推奨される RESTEasy Jackson プロバイダーに切り替えるようにしてください。Jettison プロバイダーの使用継続を希望する場合は、以下の例で示すように **jboss-deployment-descriptor.xml** ファイルでその明示的な依存関係を定義する必要があります。

```
<?xml version="1.0" encoding="UTF-8"?>
<jboss-deployment-structure>
  <deployment>
    <exclusions>
      <module name="org.jboss.resteasy.resteasy-jackson2-provider"/>
      <module name="org.jboss.resteasy.resteasy-jackson-provider"/>
    </exclusions>
    <dependencies>
      <module name="org.jboss.resteasy.resteasy-jettison-provider" services="import"/>
    </dependencies>
  </deployment>
</jboss-deployment-structure>
```

明示的な依存関係を定義する方法の詳細は、JBoss EAP 7.4 **開発ガイド** の デプロイメントへの明示的なモジュール依存関係の追加 を参照してください。

7.4.8. JBoss EAP の MicroProfile

MicroProfile は、開発者がアプリケーションを変更したり再パッケージ化したりすることなく、複数の環境で実行できるようにアプリケーションやマイクロサービスを設定するために使用できる仕様の名前です。以前は、**MicroProfile** はテクノロジープレビューとして JBoss EAP 7.3 で利用可能でしたが、その後削除されました。**MicroProfile** は現在、JBoss EAP XP でのみ使用可能です。

関連情報

- MicroProfile について

7.5. CDI アプリケーションの変更

JBoss EAP 8.0 には CDI 4.0 のサポートが含まれています。その結果、古い CDI リリースを使用して作成されたアプリケーションは、JBoss EAP 8.0 に移行するときに動作が一部変更される可能性があります。このセクションでは、これらの変更点の一部のみを要約します。

Weld および CDI 4.0 の詳細は、以下を参照してください。

- Jakarta Context Dependency Injection 4.0
- Weld 5.1.1.Final - CDI Reference Implementation

7.5.1. Bean アーカイブ

有効な Bean の Bean クラスは、CDI によって検出され、Bean として処理されるように、Bean アーカイブにデプロイされる必要があります。

CDI 1.1 には **暗黙的** bean アーカイブが導入されました。これは bean を定義するアノテーションを持つ bean クラスが 1 つ以上またはセッション bean が 1 つ以上含まれるアーカイブです。暗黙的な bean アーカイブは CDI によってスキャンされ、型検索中に bean を定義するアノテーションを持つクラスのみが検出されます。詳細は、JSR 365: Contexts and Dependency Injection for Java ™ 2.0 の Type and Bean Discovery を参照してください。bean 定義アノテーションの Jakarta に相当するものは、Jakarta Context Dependency Injection 2.0 specification で定義されています。

CDI 4.0 の場合:

- アーカイブでは、beans.xml にバージョン番号の有無は区別されません。
- アーカイブには、ビルド互換エクステンションに加えて、beans.xml ファイルを含まないアーカイブも含まれます。ビルド互換エクステンションは Bean アーカイブではありません。
- 空の beans.xml ファイルを含むアーカイブのデフォルトの検出モードは、**all** ではなく **annotated** に設定されています。たとえば、**beans.xml** ファイルが空の場合、それは明示的な Bean アーカイブではなく、暗黙的な Bean アーカイブになります。
- どちらの場合も、Bean 検出要素は、**beans.xml** ファイルがあるアーカイブとないアーカイブの間で影響を受けません。

CDI 4.0 の詳細は、Jakarta Contexts and Dependency Injection 4.0 を参照してください。

bean アーカイブは **all**、**annotated**、または **none** の bean 検索モードを持ちます。空ではない beans.xml を含む Bean アーカイブでは、bean-discovery-mode 属性を指定する必要があります。この属性のデフォルト値は **annotated** です。

以下の場合、アーカイブは bean アーカイブではありません。

- **bean-discovery-mode** が **none** である **beans.xml** ファイルが含まれる場合。
- これにはポータブルエクステンションまたはビルド互換エクステンションが含まれていますが、**beans.xml** ファイルは含まれていません。

次の場合、アーカイブは **明示的な** Bean アーカイブです。

- アーカイブには、**bean-discovery-mode** が **all** の **beans.xml** ファイルが含まれています。

以下の場合、アーカイブは **暗黙的** bean アーカイブになります。

- アーカイブには空の beans.xml ファイルが含まれています。

- アーカイブに **beans.xml** ファイルが含まれなくても、bean を定義するアノテーションを持つ bean クラスが 1 つ以上含まれるか、セッション bean が 1 つ以上含まれる場合。

CDI 1.2 は bean を定義するアノテーション を以下に限定します。

- **@ApplicationScoped**、**@SessionScoped**、 **@ConversationScoped**、および **@RequestScoped** アノテーション
- その他すべての通常スコープタイプ
- **@Interceptor** および **@Decorator** アノテーション
- **@Stereotype** アノテーションが付けられた stereotype アノテーションすべて
- **@Dependent** スコープアノテーション

7.5.2. 会話解決の明確化

会話コンテキストのライフサイクルは、CDI Specification Issue CDI-411 で説明されているサーブレット仕様との競合を回避するために、CDI 1.2 で変更されました。会話スコープはすべてのサーブレットリクエストの間はアクティブで、他のサーブレットやサーブレットフィルターによるリクエストボディーや文字エンコードの設定を妨害してはなりません。詳細は、Conversation context lifecycle in Jakarta EE を参照してください。

7.5.3. オブザーバー解決

イベント解決は、CDI 1.2 で部分的に書き直されました。CDI 1.0 では、オブザーバーメソッドにすべてのイベント修飾子がある場合にイベントがオブザーバーメソッドに送信されます。CDI 1.2 では、オブザーバーメソッドにイベント修飾子がない場合やイベント修飾子のサブセットがある場合にイベントがオブザーバーメソッドに送信されます。詳細は、Observer resolution を参照してください。

7.6. HTTP セッション ID の変更

JBoss EAP 6.4 と JBoss EAP 7 の間で、HTTP セッションに割り当てられた一意な ID を取得するため **request.getSession().getId()** 呼び出しによって返される文字列が変更になりました。

JBoss EAP 6.4 では、セッション ID とインスタンス ID の両方が **session-id.instance-id** 形式で返されました。

JBoss EAP 7 および EAP 8 はセッション ID のみを返します。

この変更により、JBoss EAP 6 から JBoss EAP 8 への一部のアップグレードで、ルートのない cookie に問題が発生することがあります。アプリケーションがこのメソッド呼び出しからの戻り値を元にして JSESSIONID cookie を再作成する場合、適切に動作するようにアプリケーションコードを更新する必要がある場合があります。

7.7. 明示的なモジュール依存関係の移行

前リリースの JBoss EAP ではモジュラークラスローディングシステムと JBoss モジュールが導入され、アプリケーションが使用できるクラスを細かに制御することができました。この機能によって、アプリケーションの **MANIFEST.MF** ファイルまたは **jboss-deployment-structure.xml** デプロイメント記述子ファイルを使用して暗示的なモジュール依存関係を設定できました。

アプリケーションで明示的なモジュール依存関係を定義した場合、JBoss EAP 7 で変更になった以下の項目に注意してください。

利用可能な依存関係の確認
JBoss EAP に含まれるモジュールが変更されました。アプリケーションを JBoss EAP 7 に移行する場合は、**MANIFEST.MF** および **jboss-deployment-structure.xml** ファイルエントリーを調べて、製品のこのリリースで削除または非推奨になったモジュールを参照していないことを確認してください。

アノテーションのスキャンに必要な依存関係
以前のリリースの JBoss EAP では、EJB インターセプターの宣言時など、アノテーションのスキャン中に処理される必要があるアノテーションが依存関係に含まれていると、Jandex インデックスを生成して新しい JAR ファイルに含まれるようにし、**MANIFEST.MF** または **jboss-deployment-structure.xml** デプロイメント記述子ファイルにフラグを設定する必要がありました。

JBoss EAP 7 では、静的モジュールに対するアノテーションインデックスの自動ランタイム生成が提供されるため、手動で生成する必要がなくなりました。しかし、JBoss EAP 7 でも以下のように **annotations** フラグを **MANIFEST.MF** ファイルまたは **jboss-deployment-structure.xml** デプロイメント記述子ファイルに追加する必要があります。

例: MANIFEST.MF ファイルのアノテーションフラグ

```
Dependencies: com.company.my-ejb annotations, com.company.other
```

例: jboss-deployment-structure.xml ファイルのアノテーションフラグ

```
<jboss-deployment-structure>
  <deployment>
    <dependencies>
      <module name="com.company.my-ejb" annotations="true"/>
      <module name="com.company.other"/>
    </dependencies>
  </deployment>
</jboss-deployment-structure>
```

7.8. HIBERNATE の変更

JBoss EAP 8 には、Java プログラミング言語のオブジェクトリレーショナルマッピングツールである Hibernate ORM 6.2 のサポートが含まれています。Hibernate ORM 6.2 のドキュメントの詳細は、Hibernate ORM 6.2 を参照してください。

JBoss EAP 7.4 から JBoss EAP 8.0 に移行する場合は、使用中の Hibernate ORM バージョンの特定の Hibernate ORM 移行ドキュメントを参照してください。

- JBoss EAP 7.4 から JBoss EAP 8 に移行するには、以下の手順を完了する必要があります。
  - Hibernate ORM 5.3 から 5.4 への移行
  - Hibernate ORM 5.4 から 5.5 への移行
  - Hibernate ORM 5.5 から 5.6 への移行
  - Hibernate ORM 5.6 から 6.0 への移行
  - Hibernate ORM 6.0 から 6.1 への移行
  - Hibernate ORM 6.1 から 6.2 への移行
  - Hibernate ORM ダイアレクト

    - 非推奨の Hibernate ORM クラス
    - Hibernate ORM クラスのインキュベート
    - Hibernate ORM インターナル
- JBoss EAP および Hibernate の古いバージョンから移行するには、以下の手順を完了する必要があります。
    - Hibernate ORM 4.3 から Hibernate ORM 5.0 への移行
    - Hibernate ORM 5.0 から Hibernate ORM 5.1 への移行
    - Hibernate ORM 5.1 および Hibernate ORM 5.2 から Hibernate ORM 5.3 への移行

関連情報

- Hibernate ORM 6.2 のダイアレクト
- 非推奨の Hibernate ORM クラス
- Hibernate ORM クラスのインキュベート
- Hibernate ORM インターナル

7.8.1. Hibernate ORM 5.3 から 5.4 への移行

このセクションでは、Hibernate ORM バージョン 5.3 から 5.4 に移行する際に必要な変更点について説明します。Hibernate ORM 5.3 と Hibernate ORM 5.4 の間で実装された変更の詳細は、Hibernate ORM 5.4 Migration Guide を参照してください。

既知の変更点
以下では、Hibernate ORM バージョン 5.3 から 5.4 に移行する際の変更点の一部について説明します。

7.8.1.1. 遅延した ID 挿入動作のオーバーライド

- Hibernate 5.3 では、**FlushMode** 値または **FlushModeType** 値に基づいて影響を受ける **DelayedPostInsertIdentifier** の動作のサポートが提供されました。つまり、**Extended PersistenceContext** のサポートが強化されました。残念ながら、この変更にはいくつかの問題がありました。
- Hibernate 5.4 では、Hibernate 5.3 の動作を可能な限り保持し、選択されたユースケースに対して非常に限定された **DelayedPostInsertIdentifier の** 動作のみを復元することが決定されました。
- Hibernate 5.4 をより柔軟にするために、Hibernate 5.3 の動作を完全に無効にして Hibernate 5.2 以前に戻す一時的な解決策として使用する設定オプションが追加されました。

7.8.1.2. SQL Server JDBC ドライバーのバージョンを少なくとも 6.1.2 にアップグレードする

HHH-12973 を修正したため、JDBC ドライバーのバージョンを 6.1.2 にアップグレードする必要があります。この問題のため、古いバージョンの SQL Server JDBC ドライバーは、データベース接続を閉じずに **INFORMATION_SCHEMA.SEQUENCES** をイントロスペクトできません。

7.8.2. Hibernate ORM 5.4 から 5.5 への移行

このセクションでは、Hibernate ORM バージョン 5.4 から 5.5 に移行する際に必要な変更点について説明します。Hibernate ORM 5.4 と Hibernate ORM 5.5 の間で実装された変更の詳細は、Hibernate ORM 5.5 Migration Guide を参照してください。

既知の変更点
Hibernate ORM 5.5 バージョンは、5.4 メンテナンスリリースに適用されたすべてのバグ修正が含まれており、Jakarta Persistence API のサポートが導入されているため、Hibernate ORM 5.4 に似ています。

7.8.2.1. Dom4J ベースの XML マッピング

Hibernate による XML マッピング定義の解析の実装は、この依存関係を削除するための継続的な進展を確実にするために、DOM4J ではなく JAXB に基づいて完全に作り直されました。

7.8.2.2. "拡張プロキシー" を無効化する機能の削除

"拡張プロキシー" 機能は、Hibernate 5.3 のオプションのパフォーマンス向上機能として導入されました。この機能は永続的に有効化されました。

7.8.3. Hibernate ORM 5.5 から 5.6 への移行

このセクションでは、Hibernate ORM バージョン 5.5 から 5.6 に移行する際に必要な変更点について説明します。Hibernate ORM 5.5 と Hibernate ORM 5.6 の間で実装された変更の詳細は、Hibernate ORM 5.6 Migration Guide を参照してください。

非推奨の機能
Hibernate 5.6 バージョンは、以前の Hibernate 5.5 バージョンと非常に似ています。異なるのは、以前の Hibernate リリースで非推奨となった機能の一部が削除されている点です。

7.8.3.1. Javassist の削除

エンティティーのバイトコード拡張に使用する実装として **javassist** を選択できなくなりました。**Byte Buddy** がデフォルトであり、**javassist** はしばらくの間非推奨となり、現在は削除されています。これはアプリケーションに機能的な影響を与えません。唯一の例外は、**hibernate.bytecode.provider=javassist property** を設定することが無効化されたことです。この機能を使用している場合は、このプロパティーを削除できます。これにより、Hibenate ORM が依存関係の中に **javassist** をリストしていない場合に、問題が発生する可能性があります。

7.8.4. Hibernate ORM 5.6 から 6.0 への移行

このセクションでは、Hibernate ORM バージョン 5.6 から 6.0 に移行する際に必要な変更点について説明します。Hibernate ORM 5.6 と Hibernate ORM 6.0 の間で実装された変更の詳細は、Hibernate ORM 6.0 Migration Guide を参照してください。

Hibernate 6.0 リリースには次の変更が含まれています。

- Java 11 は、Hibernate 6.0 の互換性のある最小のベースラインバージョンです。
- Jakarta Persistence: Hibernate ORM 6.0 リリースのもう 1 つの重要な変更には、Java EE 仕様で定義された Java Persistence から Jakarta EE 仕様で定義された Jakarta Persistence への移行が含まれます。この変更による最も重要な影響には、Java Persistence クラス **javax.persistence.*** の代わりに Jakarta Persistence クラス **jakarta.persistence.*** が使用されることが含まれます。
- JDBC からの読み取り: Hibernate ORM 6.0 バージョンの開発のもう 1 つの理由は、JDBC

ResultSet からの結果の名前での読み取り (read-by-name) から、位置による結果の読み取り (read-by-position) に移行することでした。この変更は、スループットテストを実施することでスケーリングを改善するために行われました。

- 生成された SQL: この機能により、次の機能が強化されました。
  - 列のエイリアスが生成されなくなる
  - 列参照は "unique-d"
  - 結合の定義が改善され、不要な結合 (セカンダリーテーブル、継承テーブル) がより適切に判断される
- オブジェクトとしての識別子 - Hibernate の以前のバージョンでは、すべての識別子のタイプが **Serializable** を実装する必要がありましたが、Hibernate 6.0 では識別子を任意の **Object** にできるため、この制限がなくなりました。この変更は、**Serializable** を使用して以前に定義された多くの API および SPI メソッドに影響します。
- @IdGeneratorType: このリリースでは、**@IdGeneratorType** アノテーションを使用して、識別子生成用のカスタムジェネレーターを定義するタイプセーフな方法を向上させることができます。
- 暗黙的な識別子のシーケンスとテーブル名: Hibernate が識別子の生成に関連付けられたシーケンスとテーブルの暗黙的な名前を決定する方法が Hibernate 6.0 で変更されました。これはユーザーのアプリケーションの移行に影響を与える可能性があります。このリリースでは、Hibernate はデフォルトで単一のシーケンス **hibernate_sequence** ではなく、エンティティー階層ごとにシーケンスを作成します。
- 暗黙的シーケンスジェネレーターのデフォルト: 前述の **hibernate_sequence** のような暗黙的シーケンスは、JPA **@SequenceGenerator** アノテーションのデフォルト値に従うようになりました。これは、シーケンスの割り当てサイズが 50 であることを意味します。
- タイプシステム: Hibernate 6.0 はメジャーリリースであるため、もう 1 つの重要な変更は Hibernate のマッピングアノテーションを変更し、よりタイプセーフにすることでした。read-by-position の変更を実装するためにタイプ関連のコントラクトがすでに変更されていたため、この機能はこのリリースで提供されることが決定されました。
- クエリー: クエリーの機能に多くの変更が導入されました。HQL および Criteria クエリーをモデル化するための専用ツリー構造への移行、挿入、更新、削除などの一括 SQM DML ステートメントの実装の改善、**hibernate.criteria.copy_tree** プロパティーの動作の変更などのクエリー機能と、パススルートークンのインクルージョンがこのリリースで導入されました。
- **#onSave** メソッドのシグネチャーの変更: **#onSave** メソッドのシグネチャーは、**boolean onSave(Object entity, Serializable id, Object[] state, String[] propertyNames, Type[] types)** から **boolean onSave(Object entity, Object id, Object[] state, String[] propertyNames, Type[] types)** へ変更されました。これは、想定される ID タイプが **Serializable** から **Object** に変更されたことに対応するものです。
- フェッチの循環性の決定: Hibernate の以前のバージョンでは、深さ優先のアプローチを使用してフェッチを決定していましたが、これにより、奇妙な"循環性" が決定されることがありました。Hibernate 6.0 以降、幅優先アプローチを使用してフェッチの決定が実行されるようになりました。
- org.hibernate.loader の再構築: **loader.collection** パッケージの内容は、**loader.ast.spi** および **loader.ast.internal** に再構築され、SQM API にも適合されました。

- SQL パッケージの再構築: **sql.ordering** の内容は **metamodel.mapping.ordering.ast** に移動されました。
- hbm.xml マッピングの非推奨: レガシー **hbm.xml** マッピング形式は非推奨となり、6.x 以降はサポートされなくなります。
- レイジー (Lazy) 関連付けの遵守: Hibernate 6.0 より前では、**fetch="join" or @Fetch(FetchMode.JOIN)** を使用したレイジー関連付けは、クエリーの関連付けがレイジーとして扱われても、ID によって (**Session#get**/**EntityManager#find** を介して) ロードされると、イーガー (Eager) と見なされました。Hibernate 6.0 以降では、フェッチメカニズムに関係なく、このようなレイジー関連付けが適切に扱われるようになりました。下位互換性は、**lazy="false" or @ManyToOne(fetch = EAGER)**/**@OneToOne(fetch = EAGER)**/**@OneToMany(fetch = EAGER)**/**@ManyToMany(fetch = EAGER)** を指定することで実現できます。
- **hbm.xml <return-join/>** の動作の変更: Hibernate 6.0 に従って、**<return-join/>`** により、選択項目を追加するのではなく、関連付けがフェッチされます。

これらの機能の詳細は、Hibernate ORM 6.0 Migration Guide を参照してください。

以下に示すように、Hibernate 6.0 リリースでは、以前の Hibernate リリースの機能が多数削除されました。

- **hbm.xml** の複数の **<column/>** が許可されなくなる - 6.0 では、複数の列を使用した基本的なプロパティーマッピングのサポートが削除されました。コンポーネントクラス属性は、**CompositeUserType** クラスの適切な解釈をサポートするようになりました。
- レガシー Hibernate Criteria API - Hibernate 5.x で非推奨となったレガシー Hibernate Criteria API は、Hibernate 6.0 で削除されました。
- NativeQuery 経由で呼び出し可能 - **NativeQuery** を使用した SQL 関数およびプロシージャーの呼び出しは、サポートされなくなりました。代わりに、**org.hibernate.procedure.ProcedureCall** または **jakarta.persistence.StoredProcedureQuery** などのメソッドを使用してください。
- HQL fetch all properties 句 - fetch all properties 句が HQL 言語から削除されました。
- JMX 統合 - Hibernate は、JMX 環境と統合するための組み込みサポートを提供しなくなりました。
- JACC の統合 - Hibernate は、JACC 環境と統合するための組み込みサポートを提供しなくなりました。

Hibernate 6.0 で削除された機能の詳細は、Hibernate ORM 6.0 Migration Guide を参照してください。

7.8.5. Hibernate ORM 6.0 から 6.1 への移行

このセクションでは、Hibernate ORM バージョン 6.0 から 6.1 に移行する際に必要な変更点について説明します。Hibernate ORM 6.0 と Hibernate ORM 6.1 の間で実装された変更の詳細は、Hibernate ORM 6.1 Migration Guide を参照してください。

Hibernate 6.1 リリースには次の変更が含まれています。

- 基本アレイ: **byte[]**/**Byte[]** および **char[]**/**Character[]** 以外の基本アレイ、および基本コレクション (Collection のサブタイプのみ) は、デフォルトで型コード **SqlTypes.ARRAY** にマップされるようになり、これは、**org.hibernate.dialect.Dialect** の 新しいメソッド **getArrayTypeName**

および **supportsStandardArrays** によって決定される SQL 標準アレイタイプにマップされます。

- Enum マッピングの変更: Enum は、以前は **TINYINT** にマップされていましたが、現在はデフォルトで型コード **SqlType.SMALLINT** にマップされるようになりました。Java では実質的に最大 32K の enum エントリーが許可されていますが、**TINYINT** は 1 バイトのタイプにすぎないため、このマッピングは正しくありませんでした。

Hibernate 6.1 に含まれる機能の詳細は、Hibernate ORM 6.1 Migration Guide を参照してください。

7.8.6. Hibernate ORM 6.1 から 6.2 への移行

このセクションでは、Hibernate ORM バージョン 6.1 から 6.2 に移行する際に必要な変更点について説明します。Hibernate ORM 6.1 と Hibernate ORM 6.2 の間で実装された変更の詳細は、Hibernate ORM 6.2 Migration Guide を参照してください。

Hibernate 6.2 リリースには、次の機能強化が含まれています。

DDL 型の変更:

- OffsetTime マッピングの変更 - このリリースでは、**OffsetTime** は **@TimeZoneStorage** と **hibernate.timezone.default_storage** 設定に依存します。デフォルト設定は **TimeZoneStorageType.DEFAULT** であるため、そのような列に対する DDL の期待値が変更されたことを意味します。

- MariaDB での UUID マッピングの変更 - MariaDB では、**binary(16)** を使用していた以前と比較して、型コード **SqlTypes.UUID** はデフォルトで DDL 型 **uuid** を参照します。この変更により、既存のデータベースでスキーマ検証エラーが発生する可能性があります。

- SQL Server での UUID マッピングの変更 - SQL Server では、**binary(16)** を使用していた以前と比較して、型コード **SqlTypes.UUID** はデフォルトで DDL 型 **uniqueidentifier** を参照します。この変更により、既存のデータベースでスキーマ検証エラーが発生する可能性があります。

- Oracle での JSON マッピングの変更 - Oracle 12.1 以降では、**clob** を使用していた以前と比較して、型コード **SqlTypes.JSON** はデフォルトで DDL 型 blob を参照し、21 以降では json を参照します。この変更により、既存のデータベースでスキーマ検証エラーが発生する可能性があります。

- H2 での JSON マッピングの変更 - H2 1.4.200 以降では、**clob** を使用していた以前と比較して、型コード **SqlTypes.JSON** はデフォルトで DDL 型 JSON を参照します。この変更により、既存のデータベースでスキーマ検証エラーが発生する可能性があります。

- enum のデータ型 - Hibernate 6.2 以降、enum を格納するための暗黙的な SQL データ型の選択は、enum クラスで定義されたエントリーの数に影響されます。

- タイムゾーンとオフセットのストレージ - **hibernate.timezone.default_storage** のデフォルトが **DEFAULT** になりました。

- Byte[]/Character[] マッピングの変更 - Hibernate 6.2 では、ドメインモデル内の **Byte[]** のマッピングおよび **Character[]** マッピングの変更を処理するように設定できます。

- オプションの 1 対 1 マッピングの UNIQUE 制約 - Hibernate の以前のバージョンでは、**optional** とマークされた論理 1 対 1 関連付けに対する **UNIQUE** 制約がデータベースに作成されませんでした。Hibernate 6.2 以降では、これらの UNIQUE 制約が作成されるようになりました。

- Oracle でのネイティブ SQL クエリーにおける number (n,0) の列型推論 - Hibernate 6.0 以降、

Oracle でスケール 0 の number 型の列は、精度に応じて **boolean**、**tinyint**、**smallint**、**int**、または **bigint** と解釈されました。現在、スケール 0 の数値型の列は、精度に応じて **int** または **bigint** として解釈されます。

- レガシーデータベースバージョンのサポートの削除 - Hibernate 6.2 では、Hibernate でサポートされているほとんどのデータベースダイアレクトに対して、サポートされる最小データベースバージョンの概念が導入されています。

- CDI 処理の変更 - CDI が使用可能で設定されている場合、Hibernate は CDI **BeanManager** を使用してさまざまな Bean 参照を解決できます。Hibernate 6.2 以降では、**hibernate.cdi.extensions** が **true** に設定されている場合、これらのエクステンションは CDI **BeanManager** からのみ解決されます。

- 拡張デフォルトの変更と非推奨 - **enableLazyInitialization** および **enableDirtyTracking** 拡張ツールオプション、グローバルプロパティー **hibernate.bytecode.use_reflection_optimizer**、およびそれぞれの **hibernate.enhancer.enableLazyInitialization** および **hibernate.enhancer.enableDirtyTracking** 設定は、デフォルト値を **true** に切り替え、これらの設定は現在は非推奨となりました。

- **org.hibernate.cfg** および **org.hibernate.loader** のパッケージ更新: **org.hibernate.cfg** および **org.hibernate.loader** パッケージが更新され、API、SPI、および内部とみなされるコントラクトの違いが明確に表示されます。

- 統合コントラクト (SPI) の変更 - Hibernate ORM 6.2 の開発中に、**EntityPersister#lock**、**EntityPersister#multiLoad**、**Executable#afterDeserialize**、および **JdbcType#getJdbcRecommendedJavaTypeMapping()** の SPI が変更されました。

- クエリーパスの比較: Hibernate 6.2 に従って、パスの比較は早期に型チェックされます。

- バッチフェッチと LockMode - **LockMode** が **READ** より大きい場合、Hibernate はバッチフェッチを実行しないため、初期化されていない既存のプロキシーは初期化されません。これは、ロックモードがバッチフェッチキュー内のプロキシーの 1 つと異なるためです。

7.8.7. Hibernate ORM 4.3 から Hibernate ORM 5.0 への移行

JBoss EAP 7.0 には Hibernate ORM 5.0 が含まれていました。ここでは、Hibernate ORM をバージョン 4.3 からバージョン 5 に移行する際に必要な変更について説明します。Hibernate ORM 4 から Hibernate ORM 5 の間に実装された変更は、Hibernate ORM 5.0 Migration Guide を参照してください。

削除および非推奨となったクラス
以下のクラスは非推奨となり、Hibernate ORM 5 から削除されました。

- **org.hibernate.cfg.AnnotationConfiguration**
- **org.hibernate.id.TableGenerator**
- **org.hibernate.id.TableHiLoGenerator**
- **org.hibernate.id.SequenceGenerator**

クラスおよびパッケージのその他の変更

- ブートストラップの再設計に合わせて、**org.hibernate.integrator.spi.Integrator** インターフェイスが変更になりました。

- 新しいパッケージ **org.hibernate.engine.jdbc.env.spi** が作成されました。これには、**org.hibernate.engine.jdbc.spi.JdbcServices** インターフェイスからデプロイメントされた **org.hibernate.engine.jdbc.env.spi.JdbcEnvironment** インターフェイスが含まれます。

- **org.hibernate.id.PersistentIdentifierGenerator** 実装に影響する、新しい **org.hibernate.boot.model.relational.ExportableProducer** インターフェイスが導入されました。

- **org.hibernate.id.Configurable** の署名が変更になり、**org.hibernate.dialect.Dialect** のみでなく **org.hibernate.service.ServiceRegistry** を許可するようになりました。

- **org.hibernate.metamodel.spi.TypeContributor** インターフェイスは **org.hibernate.boot.model.TypeContributor** に移行されました。

- **org.hibernate.metamodel.spi.TypeContributions** インターフェイスは **org.hibernate.boot.model.TypeContributions** に移行されました。

タイプ処理

- 組み込みの **org.hibernate.type.descriptor.sql.SqlTypeDescriptor** 実装は、**org.hibernate.type.descriptor.sql.SqlTypeDescriptorRegistry** で自動登録しないようになりました。組み込みの実装を拡張し、その動作に依存するカスタム **SqlTypeDescriptor** 実装を使用するアプリケーションを更新し、**SqlTypeDescriptorRegistry.addDescriptor()** を呼び出すようにする必要があります。

- 生成された UUID として定義された ID では、**BINARY(16)** を生成して適切に比較が行われるようにするため、一部のデータベースでは **@Column(length=16)** を明示的に設定する必要があります。

- **hbm.xml** で定義された **EnumType** マッピングの場合、**javax.persistence.EnumType.STRING** **name-mapping** が必要な場合は、**useNamed(true)** 設定を使用するか、VARCHAR 値 を **12** に指定して、この設定を明示的に指定する必要があります。

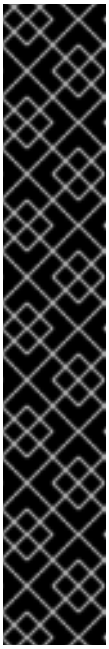
トランザクション管理

- Hibernate ORM 5 では、トランザクション SPI が大幅に再設計されました。Hibernate ORM 4.3 では、**org.hibernate.Transaction** API を使用して異なるバックエンドトランザクションストラテジーに直接アクセスしました。Hibernate ORM 5 には間接参照のレベルが導入されました。**org.hibernate.Transaction** 実装はバックエンドで、バックエンドストラテジーに応じて指定のセッションのトランザクションコンテキストを表す **org.hibernate.resource.transaction.TransactionCoordinator** と対話するようになりました。開発者への直接的な影響はありませんが、ブートストラップの設定に影響する可能性があります。これまで、アプリケーションは非推奨となった **hibernate.transaction.factory_class** プロパティーを指定し、**org.hibernate.engine.transaction.spi.TransactionFactory** FQN (完全修飾名) を参照しました。Hibernate ORM 5 では、**hibernate.transaction.coordinator_class** 設定を指定し、**org.hibernate.resource.transaction.TransactionCoordinatorBuilder** を参照します。詳細は、**org.hibernate.cfg.AvailableSettings.TRANSACTION_COORDINATOR_STRATEGY** を参照してください。

- 以下の短縮名が認識されるようになりました。

  - **jdbc**: JDBC **java.sql.Connection** を使用してトランザクションを管理します。これは、Jakarta Persistence 以外のトランザクションのデフォルトです。

  - **jta**: Jakarta Transactions を使用してトランザクションを管理します。



重要

Jakarta Persistence アプリケーションが **hibernate.transaction.coordinator_class** プロパティーの設定を提供しない場合、Hibernate は永続化ユニットのトランザクションタイプを基にして自動的に適切なトランザクションコーディネーターを構築します。

Jakarta 永続アプリケーションが **hibernate.transaction.coordinator_class** プロパティーの設定を提供しない場合、Hibernate はデフォルトで **jdbc** を使用してトランザクションを管理します。アプリケーションが実際に Jakarta Transactions を使用する場合は、このデフォルトによって問題が発生します。Jakarta Transaction を使用する JPA でないアプリケーションでは、**hibernate.transaction.coordinator_class** プロパティーの値を明示的に **jta** に設定するか、JTA ベースのトランザクションを適切に調整する **org.hibernate.resource.transaction.TransactionCoordinator** を構築するカスタムの **org.hibernate.resource.transaction.TransactionCoordinatorBuilder** を提供する必要があります。

Hibernate ORM 5 のその他の変更

- **cfg.xml** ファイルは完全に解析され、イベント、セキュリティー、およびその他の関数と統合されます。
- **EntityManagerFactory** を使用して **cfg.xml** からロードされたプロパティーは、これまで名前の前に **hibernate** が付きませんでした。これにより、一貫性が確立されました。
- 設定がシリアライズ不可能になりました。
- **org.hibernate.dialect.Dialect.getQuerySequencesString()** メソッドがカタログ、スキーマ、およびインクリメントの値を取得するようになりました。
- **AuditConfiguration** は **org.hibernate.envers.boot.internal.EnversService** から削除されました。
- **AuditStrategy** メソッドが変更され、廃止された **AuditConfiguration** を削除し、新しい **EnversService** を使用するようになりました。
- **org.hibernate.hql.spi** パッケージおよびサブパッケージの複数のクラスおよびインターフェイスが新しい **org.hibernate.hql.spi.id** に移動されました。これには **MultiTableBulkIdStrategy** クラスが含まれ、さらに **AbstractTableBasedBulkIdHandler**、**TableBasedDeleteHandlerImpl**、および **TableBasedUpdateHandlerImpl** インターフェイスとそれらのサブクラスが含まれます。
- プロパティーアクセスコントラクトが完全に再設計されました。
- 有効な **hibernate.cache.default_cache_concurrency_strategy** 設定の値は、**org.hibernate.cache.spi.access.AccessType** 列挙定数ではなく、**org.hibernate.cache.spi.access.AccessType.getExternalName()** メソッドを使用して定義されるようになりました。これにより、他の Hibernate 設定と統一されます。

7.8.8. Hibernate ORM 5.0 から Hibernate ORM 5.1 への移行

JBoss EAP 7.1 には Hibernate ORM 5.1 が含まれていました。ここでは、この 2 つの違いと、Hibernate ORM のバージョン 5.0 からバージョン 5.1 に移行する際に必要な変更について説明します。

Hibernate ORM 5.1 の機能
Hibernate のこのリリースには、パフォーマンスの向上とバグ修正が含まれています。詳細は、JBoss EAP **7.1.0 リリースノート** の Hibernate ORM 5.1 の機能 を参照してください。Hibernate ORM 5.0 から Hibernate ORM 5.1 の間に実装された変更に関する詳細は、Hibernate ORM 5.1 Migration Guide を参照してください。

スキーマ管理ツールの変更
JBoss EAP 7 でのスキーマ管理ツールの変更
Hibernate ORM 5.1 のスキーマ管理ツールの変更は、次の領域に重点を置いています。

- **hbm2ddl.auto** の処理と Hibernate の Jakarta Persistence **schema-generation** のサポートを統合します。
- NoSQL データストアの Jakarta Persistence サポートを提供する永続化エンジンである Hibernate OGM の置き換えを容易にするため、SPI から JDBC の懸念事項を削除。

スキーマ管理ツールの変更は、次のクラスを直接使用するアプリケーションにとってのみ移行の問題となります。

- **org.hibernate.tool.hbm2ddl.SchemaExport**
- **org.hibernate.tool.hbm2ddl.SchemaUpdate**
- **org.hibernate.tool.hbm2ddl.SchemaValidator**
- **org.hibernate.tool.schema.spi.SchemaManagementTool**、またはその委譲

JBoss EAP 7.1 でのスキーマ管理ツールの変更
JBoss EAP 7.1 に含まれる Hibernate ORM 5.1.10 では、**SchemaMigrator** および **SchemaValidator** のパフォーマンスを向上させるデータベーステーブルを取得するストラテジーが導入されました。このストラテジーは、単一の **java.sql.DatabaseMetaData#getTables(String, String, String, String[])** 呼び出しを実行して、各 **javax.persistence.Entity** がマップされたデータベーステーブルを持っているかどうかを判断します。これはデフォルトのストラテジーで、**hibernate.hbm2ddl.jdbc_metadata_extraction_strategy=grouped** プロパティー設定を使用します。このストラテジーには、**hibernate.default_schema** や **hibernate.default_catalog** を提供する必要があることがあります。

each javax.persistence.Entity に対して **java.sql.DatabaseMetaData#getTables(String, String, String, String[])** 呼び出しを実行する古いストラテジーを使用するには、**hibernate.hbm2ddl.jdbc_metadata_extraction_strategy=individually** プロパティー設定を使用します。

7.8.9. Hibernate ORM 5.1 および Hibernate ORM 5.2 から Hibernate ORM 5.3 への移行

JBoss EAP 7.4 には Hibernate ORM 5.3 が含まれています。このセクションでは、Hibernate ORM 5.1 から Hibernate ORM 5.2、さらに Hibernate ORM 5.3 に移行する際に必要な変更の一部を説明します。

Hibernate ORM 5.2 の機能
Hibernate ORM 5.2 は、Java 8 JDK を使用して構築され、起動時に Java 8 JRE が必要になります。以下は、本リリースに追加された変更の一部を示しています。

- The **hibernate-java8** モジュールは **hibernate-core** にマージされ、Java 8 date/time データ型はネイティブにサポートされるようになりました。

- **hibernate-entitymanager** モジュールは **hibernate-core** にマージされました。**HibernateEntityManager** および **HibernateEntityManagerFactory** は非推奨となりました。
- 非推奨のクラスを削除し、Jakarta Persistence Metamodel API と適合させるため、**Session**、**StatelessSession**、および **SessionFactory** クラスの階層がリファクタリングされました。
- **org.hibernate.persister** および **org.hibernate.tuple** パッケージの SPI が変更になりました。これらの SPI を使用したカスタムクラスをすべて確認および更新する必要があります。
- **LimitHandler** の変更によって、レガシー Hibernate 4.3 の制限ハンドラーの動作を有効にできる新しい **hibernate.legacy_limit_handler** 設定が追加されました。これはデフォルトで **false** に設定されています。これは一部のダイアレクトに影響します。
- **SchemaMigrator** および **SchemaValidator** のパフォーマンスを向上する、データベーステーブル取得の新しいストラテジーが導入されました。
- 本リリースでは、PostgreSQL81Dialect とそのサブクラスの使用時に、**@Lob** アノテーションが付けられた **String**、**character[]**、および **Character[]** 属性の **CLOB** 値を処理する方法が変更になりました。
- **@TableGenerator** および **@SequenceGenerator** 名の範囲がグローバルからローカルに変更になりました。

Hibernate 5.2 に実装された変更の完全なリストは、Hibernate ORM 5.2 Migration Guide を参照してください。

Hibernate ORM 5.3 の機能

Hibernate ORM 5.3 には、Jakarta Persistence 2.2 仕様のサポートが追加されました。本リリースにはこの仕様に準拠する変更と、その他の改善点が含まれています。以下にこれらの変更の一部を示します。

- 位置クエリーパラメーター処理の変更により、以下が変更になりました。
  - HQL/JPQL クエリーの JDBC スタイルパラメーター宣言のサポートが削除されました。
  - Jakarta Persistence 位置パラメーターは名前付きパラメーターのように動作します。
  - ネイティブクエリーの JDBC スタイルパラメーター宣言は、Jakarta Persistence との一貫性を保つため、ゼロベースではなく 1 ベースのパラメーターバインディングを使用します。ゼロベースのバインディングに戻すには、**hibernate.query.sql.jdbc_style_params_base** プロパティーを **true** に設定します。
- Jakarta Persistence に準拠するため、**@TableGenerator** 値によって保存されるシーケンス値は最後に生成された値になります。これまで Hibernate は、次のシーケンス値を保存しました。このレガシー動作を有効にするには、**hibernate.id.generator.stored_last_used** プロパティーを使用します。**@TableGenerator** を使用し、Hibernate 5.3 に移行する既存のアプリケーションは **hibernate.id.generator.stored_last_used configuration** プロパティーを **false** に設定する必要があります。
- **org.hibernate.query.QueryParameter** クラスの **getType()** メソッドの名前が **getHibernateType()** に変更されました。
- さまざまなキャッシングプロバイダーの要件により多く適合するため、Hibernate の 2 次レベルキャッシュ SPI が再設計されました。詳細は HHH-11356 を参照してください。

- HHH-11356 の変更により、コンシューマーの変更も必要になりました。 これは Hibernate Statistics システムに影響します。

- ネイティブアプリケーションの Hibernate ORM 5.1 から 5.3 への移行を容易にし、Hibernate 5.1 の改ページ調整動作を維持するため、一部のメソッドが一時的に **org.hibernate.Query** クラスに追加されましたが、これらのメソッドは非推奨となりました。今後のバージョンの Hibernate に移植できるようにするため、Jakarta Persistence メソッドを使用するようアプリケーションを更新する必要があります。

- Infinispan を Hibernate の 2 次キャッシュプロバイダーとして使用するためのサポートは、Infinispan プロジェクトに移されました。そのため、**hibernate-infinispan** モジュールは削除されました。

- **org.hibernate.tool.enhance.EnhancementTask** Ant タスクの API が変更になりました。**setBase()** および **setDir()** メソッドが推奨されるため、**addFileset()** メソッドは削除されました。詳細は HHH-11795 を参照してください。

- Hibernate 4.3 で見つかったバグにより、明示的にレイジー (Lazy) としてマップした場合でも、埋め込み可能なコレクション要素と複合 ID の多対 1 の関連が集中的に取得されました。Hibernate 5.3.2 ではこのバグが修正されました。そのため、このような関連はマッピングで指定されたとおりに取得されるようになりました。詳細は HHH-12687 を参照してください。

- 本リリースでは、Hibernate イベントリスナーの Jakarta Persistence およびネイティブ実装が統一されました。そのため、**JpaIntegrator** クラスは廃止されました。**org.hibernate.jpa.event.spi.JpaIntegrator** を拡張するクラスを変更し、**org.hibernate.integrator.spi.Integrator** インターフェイスを実装するように変更する必要があります。詳細は、HHH-11264 を参照してください。

- **org.hibernate.persister** パッケージの SPI が変更になりました。これらの SPI を使用したカスタムクラスをすべて確認および更新する必要があります。

Hibernate 5.3 に実装された変更の完全リストは、Hibernate ORM 5.3 Migration Guide を参照してください。

Hibernate 5.1 および Hibernate 5.3 間の例外処理の変更
Hibernate 5.2 と 5.3 では、Jakarta Persistence 仕様に準拠して、Hibernate のネイティブブートストラップを使用して構築される **SessionFactory** の例外処理は **HibernateException** をラッピングまたは変換しました。**Session.save()** や **Session.saveOrUpdate()** のように操作が Hibernate 固有の場合のみこの動作の例外となります。

Hibernate 5.3.3 では、**hibernate.native_exception_handling_51_compliance** プロパティーが追加されました。このプロパティーは、Hibernate のネイティブブートストラップを使用して構築された **SessionFactory** の例外処理が、Hibernate ORM 5.1 でのネイティブの例外処理と同じように動作すべきかどうかを示します。**true** に設定すると **HibernateException** は Jakarta Persistence 仕様のとおりにラップまたは変換されません。この設定は、Jakarta Persistence ブートストラップを使用して構築された **SessionFactory** では無視されます。

互換性トランスフォーマー
JBoss EAP 7.4 には、Hibernate ORM 5.1 との互換性がなくなった Hibernate ORM 5.3 API メソッドに対応する互換性トランスフォーマーが含まれています。トランスフォーマーは、Hibernate ORM 5.1 を使用して構築されたアプリケーションが JBoss EAP 7.4 の Hibernate 5.3 で同じ動作を実行できるようにするための一時的な措置です。これは一時的な対策であり、これらのメソッド呼び出しを推奨される Jakarta Persistence メソッド呼び出しに置き換える必要があります。

トランスフォーマーを有効にするには、以下の方法の 1 つを使用します。

- **Hibernate51CompatibilityTransformer** システムプロパティーを **true** に設定すると、トランスフォーマーをすべてのアプリケーションに対してグローバルに有効化できます。
- **jboss-deployment-structure.xml** ファイルを使用すると、アプリケーションレベルでトランスフォーマーを有効にできます。

```
<jboss-deployment-structure>
  <deployment>
    <transformers>
      <transformer class="org.jboss.as.hibernate.Hibernate51CompatibilityTransformer"/>
    </transformers>
  </deployment>
  <sub-deployment name="main.war">
    <transformers>
      <transformer class="org.jboss.as.hibernate.Hibernate51CompatibilityTransformer"/>
    </transformers>
  </sub-deployment>
</jboss-deployment-structure>
```

以下の表は、変換前の Hibernate 5.1 メソッドと、変換後の Hibernate 5.3 メソッドを表しています。

Hibernate 5.1 のリファレンスまたはメソッド	変換後の Hibernate 5.3 のリファレンスまたはメソッド
org.hibernate.BasicQueryContract.getFlushMode()	org.hibernate.BasicQueryContract.getHibernateFlushMode()
org.hibernate.Session.getFlushMode()	org.hibernate.Session.getHibernateFlushMode()
Enum org.hibernate.FlushMode.NEVER (0)	Enum org.hibernate.FlushMode.MANUAL (0)
org.hibernate.Query.getMaxResults()	org.hibernate.Query.getHibernateMaxResults()
org.hibernate.Query.setMaxResults(int)	org.hibernate.Query.setHibernateMaxResults(int)
org.hibernate.Query.getFirstResult(int)	org.hibernate.Query.getHibernateFirstResult()
org.hibernate.Query.setFirstResult(int)	org.hibernate.Query.setHibernateFirstResult(int)

7.9. HIBERNATE SEARCH の変更

JBoss EAP 7 に含まれる Hibernate Search のバージョンが変更されました。JBoss EAP の以前のリリースには Hibernate Search 4.6.x が含まれていました。JBoss EAP 7 には、Hibernate Search 5.5.x が含まれます。

Hibernate Search 5.5 は Apache Lucene 5.3.1 に構築されます。ネイティブ Lucene API を使用する場合は、必ずこのバージョンに合わせてください。Hibernate Search 5.5.8.Final では、バージョン 3 からバージョン 5 の間に加えられた複雑な Lucene API の変更の多くがラップされて隠されていますが、クラスの一部は非推奨となったり、名前変更または再パッケージされています。ここでは、これらの変更がアプリケーションコードに与える影響について説明します。

JBoss EAP 8.0 では 、Hibernate Search 5 API が削除され、Hibernate Search 6 API に置き換えられました。

関連情報

- Hibernate Search の変更

7.9.1. Hibernate Search 6 による Hibernate Search 5 API の置き換え

JBoss EAP 8.0 では 、Hibernate Search 5 API が削除され、Hibernate Search 6 API に置き換えられました。

削除された機能のリストを確認するには、Hibernate Search 5 APIs Deprecated in JBoss EAP 7.4 and removed in EAP 8.0 を参照してください。

注記

Hibernate Search 6 API には、Hibernate Search 5 API との **下位互換性がありません。**アプリケーションを Hibernate Search 6 に移行する必要があります。

JBoss EAP 8.0 に含まれる Hibernate Search 6 の最新バージョンは 6.2 です。Hibernate Search 5 から移行する場合は、バージョン 6.0、6.1、**および** 6.2 への移行を検討してください。

詳細は、次の移行ガイドを参照してください。

- アプリケーションを Hibernate Search 5 から移行するには、Hibernate Search 6.0 の移行ガイド を参照してください。
- アプリケーションを Hibernate Search 6.0 から 6.1 に移行するには、Hibernate Search 6.1 の移行ガイド を参照してください。
- アプリケーションを Hibernate Search 6.1 から 6.2 に移行するには、Hibernate Search 6.2 の移行ガイド を参照してください。

注記

Hibernate Search 6.2 は Hibernate ORM 6.2 と互換性があります。詳細は、Hibernate Search 6.2 リファレンスドキュメントの Hibernate ORM 6 セクションを参照してください。

7.9.2. Hibernate Search 6 による Elasticsearch のサポート

JBoss EAP 8.0 は、Hibernate Search 6 で Elasticsearch バックエンドを使用してリモート Elasticsearch クラスターまたは OpenSearch クラスターにデータのインデックスを作成するためのサポートも提供します。

使用可能な Hibernate Search アーキテクチャーとバックエンドのリストを確認するには、Hibernate Search 6.2 リファレンスドキュメントの Table 2. Comparison of architectures を参照してください。

Hibernate Search 6 の設定の詳細は、「WildFly Developer guide」の Using Hibernate Search を参照してください。

関連情報

- Hibernate の変更

7.10. エンティティー BEAN の JAKARTA PERSISTENCE への移行

Enterprise Java Beans エンティティー Bean のサポートは Java EE 8 ではオプションであり、JBoss EAP 7 以降ではサポートされません。

以前のリリースの JBoss EAP では、**javax.ejb.EntityBean** クラスを拡張し、必要なメソッドを実装してエンティティー bean がアプリケーションのソースコードに作成されました。作成後、エンティティー bean は **ejb-jar.xml** ファイルで設定されました。CMP エンティティー bean は、値が **Container** の **<persistence-type>** 子要素が含まれる **<entity>** 要素を使用して指定されました。BMP エンティティー bean は、値が **Bean** の **<persistence-type>** 子要素が含まれる **<entity>** 要素を使用して指定されました。

JBoss EAP 7 以降では、コードの CMP および BMP エンティティー bean を Jakarta Persistence エンティティーに置き換える必要があります。Jakarta Persistence エンティティーは jakarta.persistence.* クラスを使用して作成され、**persistence.xml** ファイルに定義されます。

以下は Jakarta Persistence エンティティークラスの例になります。

```
import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.Id;
import jakarta.persistence.Table;

@Entity
// User is a keyword in some SQL dialects!
@Table(name = "MyUsers")
public class MyUser {
    @Id
    @GeneratedValue
    private Long id;

    @Column(unique = true)
    private String username;
    private String firstName;
    private String lastName;

    public Long getId() {
        return id;
    }
    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
    public String getFirstName() {
        return firstName;
    }
    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }
    public String getLastName() {
        return lastName;
    }
```

```
    public void setLastName(String lastName) {
        this.lastName = lastName;
    }
```

以下は **persistence.xml** ファイルの例になります。

```
<persistence xmlns="https://jakarta.ee/xml/ns/persistence"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="https://jakarta.ee/xml/ns/persistence
          https://jakarta.ee/xml/ns/persistence/persistence_3_0.xsd"
        version="3.0">
  <persistence-unit name="my-unique-persistence-unit-name">
    <properties>
      // properties...
    </properties>
  </persistence-unit>
</persistence>
```

Jakarta Persistence エンティティーの実際の例については、JBoss EAP 8.0 に含まれる **cmt** クイックスタートを参照してください。

7.11. JAKARTA 永続プロパティーの変更

このセクションでは、JBoss EAP 7.0 および 7.1 で導入された Jakarta Persistence プロパティーの変更について説明します。

JBoss EAP 7.0 での Jakarta 永続プロパティーの変更
これまでの JBoss EAP リリースでの永続性動作と互換性を維持するため、新しい永続プロパティー **jboss.as.jpa.deferdetach** が追加されました。

jboss.as.jpa.deferdetach プロパティーは、非 Jakarta トランザクションスレッドで使用されるトランザクションスコープの永続コンテキストが各 **EntityManager** 呼び出しの後にロードされたエントリーをデタッチするかどうか、永続コンテキストが閉じられるまで待機するかどうか (セッション bean が終了するときなど) を制御します。このプロパティーのデフォルト値は **false** で、各 **EntityManager** 呼び出しの後にエンティティーがデタッチまたは消去されます。これは、Jakarta Persistence specification で定義されているデフォルトの正しい動作です。プロパティーの値が **true** に設定されると、永続コンテキストが閉じられるまでエンティティーはデタッチされません。

JBoss EAP 5 では、**jboss.as.jpa.deferdetach** プロパティーが **true** に設定された場合と同様に永続性が動作しました。JBoss EAP 5 から JBoss EAP 7 に移行するときにこの動作を保持するには、以下の例のように **persistence.xml** で **jboss.as.jpa.deferdetach** プロパティーの値を **true** 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="EAP5_COMPAT_PU">
   <jta-data-source>java:jboss/datasources/ExampleDS</jta-data-source>
   <properties>
       <property name="jboss.as.jpa.deferdetach" value="true" />
   </properties>
  </persistence-unit>
</persistence>
```

In JBoss EAP 6 では、**jboss.as.jpa.deferdetach** プロパティーが **false** に設定された場合と同様に永続性が動作しました。これは JBoss EAP 7 での動作と同じであるため、アプリケーションの移行時に変更を加える必要はありません。

JBoss EAP 7.1 での JPA 永続プロパティーの変更
JBoss EAP 7.0 では、同期されていない永続化コンテキストエラーチェックが以下の場合で厳格に行われませんでした。

- 同期されたコンテナー管理の永続コンテキストは、Jakarta トランザクションに関連付けられた同期されていない拡張永続コンテキストを使用することができました。この代わりに、同期されていない永続化コンテキストが使用されないよう **に** Illegal InstallMode を発生する必要があります。
- デプロイメント記述子に指定された同期されていない永続化コンテキストが同期された永続化コンテキストとして処理されました。

さらに、JBoss EAP 7.0 では **@PersistenceContext** でヒントする **PersistenceProperty** は誤って無視されました。

これらの問題は JBoss EAP 7.1 以上で修正されました。これらの更新によってアプリケーションの動作が不適切に変更されるため、後方互換性を提供して以前の動作を維持するために JBoss EAP 7.1 には 2 つの新しい永続化ユニットプロパティーが導入されました。

プロパティー	説明
wildfly.jpa.skipmixedsynctypechecking	このプロパティーはエラーのチェックを無効にします。JBoss EAP 7.0 で動作したアプリケーションが JBoss EAP 7.1 以上で動作しない場合に一時的な対応策としてのみ使用してください。このプロパティーは今後のリリースで非推奨となる可能性があるため、可能な限り早期にアプリケーションコードを修正することが推奨されます。
wildfly.jpa.allowjoinedunsync	このプロパティーは **wildfly.jpa.skipmixedsynctypechecking** の代わりになります。これは、Jakarta トランザクションに関連付けされた非同期の永続化コンテキストを同期された永続化コンテキストとして扱うことができます。

7.12. JAKARTA ENTERPRISE BEANS クライアントコードの移行

このセクションでは、JBoss EAP 7.0 の Jakarta Enterprise Beans クライアントの変更点について説明します。また、JBoss EAP 7.0 の新しいデフォルトのリモートポートとコネクターを使用するようにクライアントコードを変更する方法についても説明します。さらに、JBoss EAP 7.1 および JBoss EAP 7.0 で導入された JBoss EJB クライアントの必要な変更についても説明します。

注記

JBoss EAP 7.0 以降、エンタープライズエンティティー Bean はサポートされません。詳細は、エンティティー Bean を Jakarta Persistence に移行する を参照してください。

7.12.1. JBoss EAP 7 での Jakarta Enterprise Beans クライアントの変更

JBoss EAP 7 以降、デフォルトのリモートコネクターとポートが変更されました。この変更の詳細は、リモート URL コネクターおよびポートの更新 を参照してください。

JBoss Server Migration Tool を使用してサーバー設定を移行した場合、古い設定が保存されるため、ここで説明する変更を行う必要はありません。ただし、新しい JBoss EAP 8.0 のデフォルト設定を使用する場合は、次の変更を行う必要があります。

7.12.1.1. デフォルトのリモート接続ポートの更新

jboss-ejb-client.properties ファイルのリモート接続ポートの値を **4447** から **8080** に変更します。以下は、前リリースと本リリースの **jboss-ejb-client.properties** ファイルの例になります。

例: JBoss EAP 6 の jboss-ejb-client.properties ファイル

```
remote.connectionprovider.create.options.org.xnio.Options.SSL_ENABLED=false
remote.connections=default
remote.connection.default.host=localhost
remote.connection.default.port=4447
remote.connection.default.connect.options.org.xnio.Options.SASL_POLICY_NOANONYMOUS=false
```

例: JBoss EAP 8 jboss-ejb-client.properties ファイル

```
remote.connectionprovider.create.options.org.xnio.Options.SSL_ENABLED=false
remote.connections=default
remote.connection.default.host=localhost
remote.connection.default.port=8080
remote.connection.default.connect.options.org.xnio.Options.SASL_POLICY_NOANONYMOUS=false
```

7.12.1.2. デフォルトコネクターの更新

新しい JBoss EAP 7 設定を使用している場合、デフォルトのコネクターは **remote** から **http-remoting** に変更になりました。この変更は、JBoss EAP のリリースのライブラリーを使用して別のリリースのサーバーに接続するクライアントに影響を及ぼします。

- クライアントアプリケーションが JBoss EAP 6 の Jakarta Enterprise Beans クライアントライブラリーを使用し、JBoss EAP 7 サーバーへ接続する場合、**8080** 以外のポートで **remote** コネクターを公開するようサーバーを設定する必要があります。その後、そのクライアントは新しく設定されたコネクターを使用して接続する必要があります。

- JBoss EAP 7 の Jakarta Enterprise Beans クライアントライブラリーを使用し、JBoss EAP 6 サーバーへ接続するクライアントアプリケーションは、サーバーインスタンスによって **http-remoting** コネクターは使用されず、**remote** コネクターが使用されることを認識する必要があります。これは、新しいクライアント側接続プロパティーを定義することで実現されます。

  例: remote 接続プロパティー

  ```
  remote.connection.default.protocol=remote
  ```

7.12.2. リモートネーミングクライアントコードの移行

新しいデフォルトの JBoss EAP 7 設定で実行している場合、新しいデフォルトのリモートポートおよびコネクターを使用するようクライアントコードを変更する必要があります。

以下の例は、リモートネーミングプロパティーが JBoss EAP 6 のクライアントコードでどのように指定されていたかを示しています。

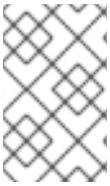
```
java.naming.factory.initial=org.jboss.naming.remote.client.InitialContextFactory
java.naming.provider.url=remote://localhost:4447
```

以下は、JBoss EAP 7 のクライアントコードでリモートネーミングプロパティーを指定する方法の例になります。

```
java.naming.factory.initial=org.wildfly.naming.client.WildFlyInitialContextFactory
java.naming.provider.url=http-remoting://localhost:8080
```

7.12.3. JBoss EAP 7.1 で導入された追加の JBoss EJB クライアントの変更

JBoss EAP 7.0 は JBoss Enterprise Java Beans クライアント 2.1.4 に含まれ、JBoss EAP 7.1 以降は JBoss Enterprise Java Beans クライアント 4.0.x に含まれていました。4.0.x には API に対する多くの変更が含まれています。



注記

JBoss EAP 7 以降、エンタープライズエンティティー Bean はサポートされません。エンティティー bean を Jakarta Persistence に移行する方法は、エンティティー Bean の Jakarta Persistence への移行 を参照してください。

- **org.ejb.client.EJBClientInvocationContext** クラスには、次の新しいメソッドが追加されます。

メソッド	型	説明
isBlockingCaller()	boolean	この呼び出しによって呼び出しているスレッドが現在ブロックされているかどうかを判断します。
isClientAsync()	boolean	メソッドがクライアント非同期としてマークされているか確認します。つまり、サーバー側のメソッドが非同期であるかに関係なく、呼び出しは非同期である必要があります。
isIdempotent()	boolean	メソッドが冪等 (idempotent) としてマークされているか判断します。つまり、マークされていると、メソッドは追加の効果なしで複数回呼び出されることを意味します。
setBlockingCaller(boolean)	void	この呼び出しによって呼び出しているスレッドが現在ブロックされているかどうかを確定します。
setLocator(EJBLocator <T>)	**<T> void**	呼び出し先のロケーターを設定します。

- **org.ejb.client.EJBLocator** クラスは以下の新しいメソッドに追加されました。

メソッド	型	説明
asStateful()	**StatefulEJBLocator<T>**	ある場合、このロケーターをステートフルロケーターとして返します。
asStateless()	**StatelessEJBLocator<T>**	ある場合、このロケーターをステートレスロケーターとして返します。
isEntity()	boolean	これがエンティティーロケーターであるか判断します。
isHome()	boolean	これがホームロケーターであるかどうかを判断します。
isStateful()	boolean	これがステートフルロケーターであるかどうかを判断します。
isStateless()	boolean	これがステートレスロケーターであるかどうかを判断します。
withNewAffinity(Affinity)	**abstract EJBLocator<T>**	このロケーターのコピーを新しいアフィニティーで作成します。

- java.security.Permission のサブクラスである新しい org.ejb.client.EJBClientPermission クラスは、特権のある Enterprise Java Beans 操作へのアクセスを制御するために導入されました。以下のコンストラクターを提供します。
  - **EJBClientPermission(String name)**
  - **EJBClientPermission(String name, String actions)**
- 以下のメソッドを提供します。

メソッド	型	説明
equals(EJBClientPermission obj)	boolean	2 つの **EJBClientPermission** オブジェクトが等値であるかをチェックします。
equals(Object obj)	boolean	2 つの **Permission** オブジェクトが等値であるかをチェックします。
equals(Permission obj)	boolean	2 つの **Permission** オブジェクトが等値であるかをチェックします。
getActions()	String	アクションを文字列として返します。
hashcode()	int	この **Permission** オブジェクトのハッシュコード値を返します。

メソッド	型	説明
implies(EJBClientPermission permission)	boolean	指定パーミッションのアクションが、この EJBClientPermission オブジェクトのアクションによって **暗示** されたかどうかをチェックします。
implies(Permission permission)	boolean	指定パーミッションのアクションが、この **Permission** オブジェクトのアクションによって **暗示** されたかどうかをチェックします。

- 特定の Enterprise Java Beans メソッドを検索するために、新しい **org.ejb.client.EJBMethodLocator** クラスが導入されました。以下のコンストラクターを提供します。
  - **EJBMethodLocator(String methodName, String… parameterTypeNames)**
- 以下のメソッドを提供します。

メソッド	型	説明
equals(EJBMethodLocator other)	boolean	このオブジェクトが別のオブジェクトと等しいかどうかを判断します。
equals(Object other)	boolean	このオブジェクトが別のオブジェクトと等しいかどうかを判断します。
forMethod(Method method)	**static EJBMethodLocator**	指定のリフレクションメソッドのメソッドロケーターを取得します。
getMethodName()	String	メソッド名を取得します。
getParameterCount()	int	パラメーターの数を取得します。
getParameterTypeName(int index)	String	指定インデックスのパラメーターの名前を取得します。
hashCode()	int	ハッシュコードを取得します。

- 失敗した場合のために、新しい **org.jboss.ejb.client.EJBReceiverInvocationContext.ResultProducer.Failed** クラスが導入されました。以下のコンストラクターを提供します。
  - **Failed(Exception cause)**
- 以下のメソッドを提供します。

メソッド	型	説明

メソッド	型	説明
discardResult()	void	結果を破棄し、使用されないことを示します。
getResult()	**オブジェクト**	結果を取得します。

- 即座に結果を得るために、新しい **org.jboss.ejb.client.EJBReceiverInvocationContext.ResultProducer.Immediate** クラスが導入されました。以下のコンストラクターを提供します。
  - **Failed(Exception cause)**
- 以下のメソッドを提供します。

メソッド	型	説明
discardResult()	void	結果を破棄し、使用されないことを示します。
getResult()	**オブジェクト**	結果を取得します。

- **org.jboss.ejb.client.Affinity** のサブクラスである新しい **org.jboss.ejb.client.URIAffinity** クラスが URI アフィニティー仕様に導入されました。これは、**Affinity.forUri(URI)** を使用して作成されます。
- 以下のメソッドを提供します。

メソッド	型	説明
equals(Affinity other)	boolean	別のオブジェクトがこのオブジェクトと等しいかどうかを示します。
equals(Object other)	boolean	別のオブジェクトがこのオブジェクトと等しいかどうかを示します。
equals(URIAffinity other)	boolean	別のオブジェクトがこのオブジェクトと等しいかどうかを示します。
getURI()	URI	関連する URI を取得します。
hashCode()	int	ハッシュコードを取得します。
toString()	String	オブジェクトの文字列表現を返します。

- **org.jboss.ejb.client.EJBMetaDataImpl** クラスは次のメソッドを非推奨とします。
  - **toAbstractEJBMetaData()**
  - **EJBMetaDataImpl(AbstractEJBMetaData<?,?>)**

7.13. WILDFLY 設定ファイルを使用するようクライアントを移行

JBoss EAP 7.1 のリリース前は、Enterprise Java Beans や命名などの JBoss EAP クライアントライブラリーは、異なる設定ストラテジーを使用していました。サーバー設定と似た方法で、すべてのクライアント設定を 1 つの設定ファイルに統合するため、JBoss EAP 7.1 には **wildfly-config.xml** ファイルが導入されました。

たとえば、JBoss EAP 7.1 より前では、**jboss-ejb-client.properties** ファイルを使用するか、**Properties** クラスを使用してプログラムでプロパティーを設定することによって、Enterprise Java Beans クライアントの新しい **InitialContext** を作成することがありました。

例: jboss-ejb-client.properties プロパティーファイル

```
remote.connectionprovider.create.options.org.xnio.Options.SSL_ENABLED=false
remote.connections=one
remote.connection.one.port=8080
remote.connection.one.host=127.0.0.1
remote.connection.one.username=quickuser
remote.connection.one.password=quick-123
```

JBoss EAP 7.1 以上では、クライアントアーカイブの **META-INF/** ディレクトリーに **wildfly-config.xml** ファイルを作成します。これは、**wildfly-config.xml** ファイルを使用した設定と同等です。

例: wildfly-config.xml ファイルを使用した同等の設定

```
<configuration>
    <authentication-client xmlns="urn:elytron:client:1.7">
        <authentication-rules>
            <rule use-configuration="ejb"/>
        </authentication-rules>
        <authentication-configurations>
            <configuration name="ejb">
                <set-user-name name="quickuser"/>
                <credentials>
                    <clear-password password="quick-123"/>
                </credentials>
            </configuration>
        </authentication-configurations>
    </authentication-client>
    <jboss-ejb-client xmlns="urn:jboss:wildfly-client-ejb:3.2">
        <connections>
            <connection uri="remote+http://127.0.0.1:8080" />
        </connections>
    </jboss-ejb-client>
</configuration>
```

関連情報

- Elytron クライアントによるクライアント認証の設定
- **wildfly-config.xml** ファイルを使用したクライアント設定

7.14. デプロイメント計画設定の移行

Java EE Application Deployment specification (JSR-88) は、複数のプロバイダーからのツールを有効にし、すべての Java EE プラットフォーム製品上にアプリケーションを設定およびデプロイするための標準のコントラクトを定義することが目的でした。このコントラクトでは、Tool Providers によってアクセスされる **DeploymentManager** および他の **javax.enterprise.deploy.spi** インターフェイスを Java EE Product Providors が実装する必要がありました。JBoss EAP 6 の場合、デプロイメントプランは、アーカイブまたは JAR アーカイブにバンドルされている **deployment-plan.xml** という名前の XML 記述子によって識別されます。

ほとんどのアプリケーションサーバー製品は、より機能が充実した独自のデプロイメントソリューションを提供するため、この仕様はほとんど採用されませんでした。そのため、JSR-88 のサポートは Java EE 7 で廃止されたため、JBoss EAP 7 でも廃止されました。

JSR-88 を使用してアプリケーションをデプロイした場合、別の方法でアプリケーションをデプロイする必要があります。JBoss EAP 管理 CLI の **deploy** コマンドを使用すると、標準的な方法でアーカイブをスタンドアロンサーバーまたはマネージドドメインのサーバーグループにデプロイできます。管理 CLI に関する詳細は、管理 CLI ガイド を参照してください。

7.15. カスタムアプリケーションバルブを移行する

カスタムバルブまたは **jboss-web.xml** XML ファイルで定義されたバルブは、手動で移行する必要があります。これには、**org.apache.catalina.valves.ValveBase** クラスを拡張して作成されたバルブや、**jboss-web.xml** 記述子ファイルの **<valve>** 要素で設定されたバルブが含まれます。

デプロイメントで設定されたバルブの移行

JBoss EAP 6 では、カスタムバルブを **jboss-web.xml** web アプリケーション記述子ファイルで設定するとアプリケーションレベルで定義することができました。JBoss EAP 7 以降、Undertow ハンドラーでもこれを行うことができます。

以下は、JBoss EAP 6 の **jboss-web.xml** ファイルに設定されたバルブの例になります。

```
<jboss-web>
  <valve>
    <class-name>org.jboss.examples.MyValve</class-name>
    <param>
      <param-name>myParam</param-name>
      <param-value>foobar</param-value>
    </param>
  </valve>
</jboss-web>
```

JBoss EAP でカスタムハンドラーを作成および設定する方法の詳細は、JBoss EAP 7.4 **開発ガイド** の カスタムハンドラーの作成 を参照してください。

カスタムオーセンティケーターバルブの移行

オーセンティケーターバルブを移行する方法については、オーセンティケーターバルブの移行 を参照してください。

7.16. セキュリティーアプリケーションの変更

JBoss Web を Undertow に置き換えるには、JBoss EAP 7 以降のセキュリティー設定を変更する必要があります。JBoss EAP 8.0 以降では、PicketBox が使用できなくなったため、レガシーセキュリティーとして Elytron を使用する必要があります。

7.16.1. オーセンティケーターバルブの移行

JBoss EAP 6.4 で **AuthenticatorBase** を拡張するカスタムオーセンティケーターバルブを作成した場合、JBoss EAP 7.1 では手作業でカスタム HTTP 認証実装に置き換える必要があります。HTTP 認証メカニズムは **elytron** サブシステムで作成され、**undertow** サブシステムで登録されます。カスタム HTTP 認証メカニズムの実装方法の詳細は、JBoss EAP 7.4 **開発ガイド** の カスタム HTTP メカニズムの開発 を参照してください。

7.16.2. PicketLink の削除

PicketLink は JBoss EAP 8.0 から削除されました。

PicketLink SP
PicketLink サービスプロバイダーの代わりに Keycloak SAML アダプターを使用します。

Keycloak SAML アダプターを設定して PicketLink から移行するには、次のタスクを実行します。

- Keycloak SAML クライアントを JBoss EAP 8.0 にインストールします。詳細は以下を参照してください。
  - jboss-eap-installation-manager を使用して JBoss EAP 8.0 をインストールする
  - SAML を使用してアプリケーションを保護するための Keycloak SAML アダプター機能パック
- 必要に応じて、PicketLink IdP の代わりに Keycloak SAML を設定します。Keycloak SAML を使用して SP アプリケーションを保護するには、SAML クライアントを作成する必要があります。Keycloak SAML クライアントの作成の詳細は、JBoss EAP 7.5 **サーバー管理ガイド** の SAML クライアントの作成 を参照してください。
- Keycloak SAML アダプターを使用するようにアプリケーションを更新します。アプリケーションの更新に関する詳細は、SAML を使用した Web アプリケーションの保護 を参照してください。

PicketLink IDP
JBoss EAP 8.0 以降では、PicketLink IDP を使用できません。代わりに Red Hat build of Keycloak を設定できます。詳細は、Red Hat build of Keycloak の設定 を参照してください。

PicketLink STS
以前のリリースでは、Apache CXF Security Token Service 実装の代替として PicketLink STS を設定できました。PicketLink STS 設定には、レガシーセキュリティードメインが含まれていました。STS アプリケーション内のレガシーセキュリティードメインおよび PicketLink への参照はすべて削除する必要があるため、代わりに Apache CXF STS を設定する必要があります。

Apache CXF STS の設定方法に関する詳細は、JBoss EAP 7.4 **Web サービスアプリケーションの開発** の セキュリテイートークンサービス (STS) を参照してください。

7.16.3. Vault の削除

Vault は JBoss EAP 8.0 から削除されました。アプリケーションでレガシー Vault 式を使用している場合は、Elytron 暗号化式を移行して使用する必要があります。

デプロイメントファイル内の **${VAULT::** のインスタンス (アノテーションやデプロイメント記述子にある可能性があります) を確認し、それらを対応する暗号化された式に置き換えます。

関連情報

- Elytron の暗号化式

7.16.4. OIDC クライアントの移行

Keycloak OIDC クライアントアダプターは JBoss EAP 8.0 ではサポートされておらず、同様の機能と設定を提供するネイティブ Elytron OIDC クライアントに置き換えられました。

Keycloak OIDC クライアントアダプターからネイティブ Elytron OIDC クライアントに移行するには、次の手順を実行します。

- アプリケーションの **web.xml** ファイル内の **<auth-method>KEYCLOAK</auth-method>** を確認し、デプロイメントの **web.xml** ファイル内の **<auth-method>OIDC</auth-method>** に置き換えます。
- **WEB-INF/keycloak.json** があることを確認し、その名前を **WEB-INF/oidc.json** に変更します。

関連情報

- JBoss EAP での OpenID Connect 設定
- **keycloak** サブシステムの移行

7.16.5. カスタムログインモジュールの移行

JBoss EAP 8.0 では、レガシーセキュリティーサブシステムが削除されました。**elytron** サブシステムでカスタムログインモジュールを引き続き使用するには、新しい Java Authentication and Authorization Service (JAAS) セキュリティーレルムと **jaas-realm** を使用します。

関連情報

- **elytron** サブシステムの JAAS レルム

7.16.6. その他のセキュリティーアプリケーションの変更

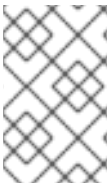
JBoss EAP 7.2 以降とそれ以前のバージョンの間には、いくつかの顕著な違いがあります。

- **NegotiationAuthenticator** バルブは **jboss-web.xml** には必要ありませんが、**web.xml** には **<security-constraint>** 要素と **<login-config>** 要素が定義されている必要があります。これらの要素は、セキュアなリソースを決定するのに使用されます。
- **<login-config>** 要素の **auth-method** 要素はコンマ区切りリストになりました。正確な値 **SPNEGO** が存在する必要があり、そのリストの最初に表示される必要があります。フォールバックとして **FORM** 認証が必要な場合、正確な値は **SPNEGO,FORM** になります。
- **jboss-deployment-structure.xml** ファイルは必要ありません。

7.17. JBOSS LOGGING の変更

JBoss EAP 7 以降、アプリケーションで JBoss Logging を使用する場合、**org.jboss.logging** パッケージのアノテーションが非推奨になることに注意してください。アノテーションは **org.jboss.logging.annotations** パッケージに移動されたため、ソースコードを更新して新しいパッケージをインポートする必要があります。

また、アノテーションは個別の Maven **groupId:artifactId:version** (GAV) ID に移動されたため、プロジェクトの **pom.xml** ファイルで **org.jboss.logging:jboss-logging-annotations** の新しいプロジェクト依存関係を追加する必要があります。



注記

ロギングアノテーションのみが移動されました。**org.jboss.logging.BasicLogger** および **org.jboss.logging.Logger** は、これまでどおり **org.jboss.logging** パッケージにあります。

非推奨となったアノテーションクラスと代替クラスを以下の表に示します。

表7.1 非推奨となったロギングアノテーションの代替

非推奨となったクラス	代替クラス
org.jboss.logging.Cause	org.jboss.logging.annotations.Cause
org.jboss.logging.Field	org.jboss.logging.annotations.Field
org.jboss.logging.FormatWith	org.jboss.logging.annotations.FormatWith
org.jboss.logging.LoggingClass	org.jboss.logging.annotations.LoggingClass
org.jboss.logging.LogMessage	org.jboss.logging.annotations.LogMessage
org.jboss.logging.Message	org.jboss.logging.annotations.Message
org.jboss.logging.MessageBundle	org.jboss.logging.annotations.MessageBundle
org.jboss.logging.MessageLogger	org.jboss.logging.annotations.MessageLogger
org.jboss.logging.Param	org.jboss.logging.annotations.Param
org.jboss.logging.Property	org.jboss.logging.annotations.Property

7.18. JAKARTA FACES コードの変更

このセクションでは、アプリケーションを JBoss EAP に移行する際の Jakarta Faces コード変更の影響について説明します。

4.0 より前の Jakarta Server Faces のサポート終了

注記

Jakarta Server Faces は、JavaServer Faces の新しい名前です。

JBoss EAP 6.4 では、**jboss-deployment-structure.xml** ファイルを作成しても、アプリケーションのデプロイメントと Jakarta Server Faces 1.2 を使用できます。JBoss EAP 7.4 には Jakarta Server Faces 2.3 が含まれ、Jakarta Server Faces 1.2 API はサポートされなくなりました。アプリケーションが Jakarta Server Faces 1.2 を使用する場合は、Jakarta Server Faces 2.3 を使用するようこれを書き直す必要があります。

JBoss EAP 8.0 は、4.0 より前のバージョンの JSF をサポートしなくなりました。

7.19. FACES の代替として MYFACES を統合する

Galleon フィーチャーパック **eap-myfaces-feature-pack** を導入することで、JBoss EAP 内のデフォルトの Mojarra Jakarta Faces 実装の代替として、別の Jakarta Faces 実装 **MyFaces** のインストールを簡素化できます。このフィーチャーパックを使用して、JBoss EAP 内で別の Jakarta Faces 実装をプロビジョニングできます。

eap-myfaces-feature-pack を使用すると、**MYFACES_VERSION** 環境変数を使用して **MyFaces** のバージョンを選択できます。この機能パックでは、**MyFaces** という名前の単一レイヤーが導入され、代替として **MyFaces** をインストールおよび選択するオプションが提供されます。詳細は、How to configure the Multi-JSF feature in EAP 8 を参照してください。

注記

JBoss EAP 8.0 との互換性はバージョン 4.x 以降に限定されます。

7.20. モジュールクラスのロードの変更

JBoss EAP 7 では、複数のモジュールに同じクラスまたはパッケージが含まれる場合のクラスローディングの動作が変更になりました。

お互いに依存し、一部同じパッケージが含まれる **MODULE_A** と **MODULE_B** の 2 つのモジュールがあるとします。JBoss EAP 6 では、依存関係からロードされたクラスまたはパッケージは **module.xml** ファイルの **resource-root** に指定されたものよりも優先されました。これは、**MODULE_A** は **MODULE_B** のパッケージを認識し、**MODULE_B** は **MODULE_A** のパッケージを認識したことを意味します。この動作は複雑で競合が発生することがありました。この動作は JBoss EAP 7 では変更になりました。**module.xml** ファイルの **resource-root** で指定されたクラスまたはパッケージは、依存関係で指定されたものよりも優先されるようになりました。これは **MODULE_A** が **MODULE_A** のパッケージを認識し、**MODULE_B** は **MODULE_B** のパッケージを参照します。これにより、競合を回避し、動作が安定します。

resource-root ライブラリーが含まれるカスタムモジュールまたは複製されたクラスがモジュール依存関係に含まれるパッケージを定義した場合、JBoss EAP 7 へ移行するときに **ClassCastException**、**LinkageError**、クラスローディングエラー、またはその他の動作の変更が発生することがあります。この問題を解決するには、**module.xml** ファイルを設定して 1 つのバージョンのクラスのみが使用されるようにする必要があります。これは、以下の方法の 1 つを使用して実現できます。

- モジュール依存関係のクラスを複製する **resource-root** を指定しないようにします。
- **imports** および **exports** 要素の **include** および **exclude** サブ要素を使用して **module.xml** ファイルでクラスローディングを制御できます。以下は、指定されたパッケージでクラスを除外する export 要素になります。

```
<exports>
  <exclude path="com/mycompany/duplicateclassespath/"/>
</exports>
```

既存の動作を保持するには、**filter** 要素を使用して **module.xml** ファイルの依存する **resource-root** から依存パッケージをフィルターする必要があります。これにより、JBoss EAP 6 で見られる odd loop なしで既存の動作を保持できます。以下は、指定のパッケージのクラスをフィルターする **root-resource** の例になります。

```
<resource-root path="mycompany.jar">
  <filter>
```

```
    <exclude path="com/mycompany/duplicateclassespath"/>
  </filter>
</resource-root>
```

モジュールおよびクラスローディングの詳細は、JBoss EAP 7.4 **開発ガイド** の クラスローディングとモジュール を参照してください。

7.21. アプリケーションクラスタリングの変更

このセクションでは、アプリケーションを JBoss EAP 6 から JBoss EAP 8 に移行するために必要なクラスタリングの変更について概説します。さらに、このセクションでは、クラスタリングの変更がアプリケーションの JBoss EAP 8.0 への移行にどのような影響を与える可能性があるかについて説明します。

7.21.1. 新しいクラスタリング機能の概要

以下のリストでは、アプリケーションを JBoss EAP 6 から JBoss EAP 8.0 に移行する場合に注意が必要な新しいクラスタリング機能の一部について説明します。

- JBoss EAP 7 には、シングルトンサービスのビルドを大幅に簡易化する、新しいパブリック API が導入されました。シングルトンサービスの詳細は JBoss EAP 7.4 **開発ガイド**の HA シングルトンサービス を参照してください。
- 一度にクラスターで単一のノードのみをデプロイおよび開始するよう、シングルトンデプロイメントを設定できます。詳細は、JBoss EAP 7.4 **開発ガイド** の HA シングルトンデプロイメント を参照してください。
- クラスター化されたシングルトン MDB を定義できるようになりました。詳細は、JBoss EAP 7.4 **Jakarta Enterprise Beans アプリケーションの開発** の クラスター化されたシングルトン MDB を参照してください。
- JBoss EAP 8.0 には Undertow mod_cluster 実装が含まれています。これは、http web サーバーを必要としない純粋な Java ロードバランシングソリューションを提供します。詳細は、JBoss EAP 7.4 **設定ガイド** の JBoss EAP をフロントエンドロードバランサーとして設定する を参照してください。

7.21.2. Web セッションクラスタリングの変更

JBoss EAP 7 では、新しい Web セッションクラスタリング実装が導入されました。これは、レガシーの JBoss Web サブシステムソースコードに密に結合された以前の実装に代わる実装です。

新しい Web セッションクラスタリング実装は、JBoss EAP プロプライエタリー Web アプリケーションの XML 記述子ファイルである **jboss-web.xml** にアプリケーションが設定される方法に影響します。以下は、このファイルのクラスタリング設定要素のみになります。

```
<jboss-web>
  ...
  <max-active-sessions>...</max-active-sessions>
  ...
  <replication-config>
    <replication-granularity>...</replication-granularity>
    <cache-name>...</cache-name>
  </replication-config>
  ...
</jboss-web>
```

distributable-web サブシステムは **jboss-web.xml** の **<replication-config>** 要素の使用を終了します。アドホックの分散可能 Web セッションプロファイルを生成して **<replication-config>** の使用を強化します。

セッション管理プロファイルを名前で参照するか、デプロイメント固有のセッション管理設定を指定して、デフォルトの分散可能なセッション管理動作をオーバーライドすることができます。詳細は、デフォルトの分散可能セッション管理動作のオーバーライド を参照してください。

以下の表は、**jboss-web.xml** ファイルの現在では廃止された要素と同様の動作を実現する方法を説明しています。

設定要素	変更の説明
<max-active-sessions/>	これまでの実装では、アクティブなセッションの数が **<max-active-sessions/>** によって指定された値を超えるとセッションの作成に失敗しました。 新しい実装では、**<max-active-sessions/>** を使用してセッションパッシベーションが有効化されます。セッションの作成によって、アクティブなセッションの数が **<max-active-sessions/>** を超える場合、新しいセッションが作成されるよう、セッションマネージャーが認識する最も古いセッションがパッシベートされます。
<passivation-config/>	JBoss EAP 7 以降、この設定要素とそのサブ要素は使用されなくなりました。
<use-session-passivation/>	以前の実装では、この属性を使用してパッシベーションが有効化されました。 新しい実装では、**<max-active-sessions/>** に負でない値を指定するとパッシベーションが有効化されます。
<passivation-min-idle-time/>	以前の実装では、パッシベーションの候補となる前にセッションは最小時間アクティブである必要がありました。そのため、パッシベーションが有効であってもセッションの作成に失敗することがありました。 新しい実装はこの論理をサポートしないため、サービス拒否 (DoS) 攻撃の発生を防ぎます。
<passivation-max-idle-time/>	以前の実装では、セッションは指定期間アイドル状態であった後にパッシベートされました。 新しい実装はレイジー (Lazy) パッシベーションのみをサポートします。イーガー (Eager) パッシベーションはサポートしません。セッションは、**<max-active-sessions/>** に準拠する必要があるときのみパッシベートされます。
<replication-config/>	**distributable-web** サブシステムは、この要素の使用を終了します。詳細は、JBoss EAP 7.4 **開発ガイド** の 分散可能 Web セッション設定の distributable-web サブシステム および デフォルトの分散可能セッション管理動作のオーバーライド を参照してください。

設定要素	変更の説明
<replication-trigger/>	以前の実装では、この要素を使用してセッションレプリケーションがトリガーされるタイミングを決定しました。新しい実装では、この設定は単一の堅牢なストラテジーに変更されました。詳細は、JBoss EAP 7.4 **開発ガイド** の 変更不能なセッション属性 を参照してください。
<use-jk/>	以前の実装では、**<use-jk/>** に指定された値に応じて、指定のリクエストを処理するノードの **instance-id** が **jsessionid** の末尾に追加され、mod_jk、mod_proxy_balancer、mod_cluster などのロードバランサーによって使用されました。 新しい実装では、**instance-id** が定義されている場合は常に **jsessionid** の末尾に追加されるようになりました。
<max-unreplicated-interval/>	以前の実装では、この設定オプションは変更されたセッション属性がない場合にセッションのタイムスタンプがレプリケートされないようにする最適化を目的としていました。しかし、セッション属性が変更されたかどうかに関係なく、セッションのアクセスにはキャッシュトランザクション RPC が必要であるため、実際は RPC の実行を防ぐことはできません。 新しい実装では、リクエストごとにセッションのタイムスタンプがレプリケートされるようになりました。これにより、フェイルオーバーの後にセッションメタデータが陳腐化されないようにします。
<snapshot-mode/>	これまでは、**<snapshot-mode/>** を **INSTANT** または **INTERVAL** として設定することができました。Infinispan の非同期レプリケーションにより、この設定オプションは廃止になりました。
<snapshot-interval/>	これは **<snapshot-mode>INTERVAL</snapshot-mode>** にのみ関係しました。**<snapshot-mode/>** は廃止されたため、このオプションも廃止されました。
<session-notification-policy/>	以前の実装では、この属性によって指定された値はセッションイベントをトリガーするポリシーを定義しました。 新しい実装ではこの動作は仕様に応じて判断されるようになり、設定不可能になりました。

新しいこの実装は、ライトスルーキャッシュストアとパッシベーションのみのキャッシュストアもサポートします。通常、ライトスルーキャッシュストアはインバリデーションキャッシュとともに使用されます。JBoss EAP 6 の web セッションクラスタリング実装は、インバリデーションキャッシュの使用時に適切に動作しませんでした。

7.21.3. デフォルトの分散可能セッション管理動作のオーバーライド

デフォルトの分散可能なセッション管理動作は、以下のいずれかの方法で上書きできます。

- 名前によるセッション管理プロファイルの参照
- デプロイメント固有のセッション管理設定の指定

既存のセッション管理プロファイルの参照

- 既存の分散可能なセッション管理プロファイルを使用するには、アプリケーションの **/WEB-INF** ディレクトリーにある分散可能な **distributable-web.xml** デプロイメント記述子を含めます。以下に例を示します。

/WEB-INF/distributable-web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<distributable-web xmlns="urn:jboss:distributable-web:1.0">
   <session-management name="foo"/>
</distributable-web>
```

- または、既存の **jboss-all.xml** デプロイメント記述子に、ターゲット分散セッション管理プロファイルを定義します。

/META-INF/jboss-all.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<jboss xmlns="urn:jboss:1.0">
   <distributable-web xmlns="urn:jboss:distributable-web:1.0">
      <session-management name="foo"/>
   </distributable-web>
</jboss>
```

デプロイメント固有のセッション管理プロファイルの使用
単一の Web アプリケーションのみがカスタムセッション管理設定を使用する場合は、デプロイメント記述子自体で設定を定義できます。アドホック設定は、**distributable-web** サブシステムで使用される設定と同じに見えます。

- デプロイメント記述子内にカスタムセッション管理設定を定義します。以下に例を示します。

/WEB-INF/distributable-web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<distributable-web xmlns="urn:jboss:distributable-web:1.0">
   <infinispan-session-management cache-container="foo" cache="bar" granularity="SESSION">
      <primary-owner-affinity/>
   </infinispan-session-management>
</distributable-web>
```

- または、既存の **jboss-all.xml** デプロイメント記述子内にセッション管理設定を定義します。

/META-INF/jboss-all.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<jboss xmlns="urn:jboss:1.0">
   <distributable-web xmlns="urn:jboss:distributable-web:1.0">
      <infinispan-session-management cache-container="foo" cache="bar" granularity="ATTRIBUTE">
         <local-affinity/>
      </infinispan-session-management>
   </distributable-web>
</jboss>
```

7.21.4. ステートフルセッション EJB クラスタリングの変更

JBoss EAP 6 では、以下の方法の 1 つでステートフルセッション Bean (SFSB) のクラスタリング動作を有効化する必要がありました。

- セッション Bean に **org.jboss.ejb3.annotation.Clustered** アノテーションを追加。

```
@Stateful
@Clustered
public class MyBean implements MySessionInt {

  public void myMethod() {
    //
  }
}
```

- **<clustered>** 要素を **jboss-ejb3.xml** ファイルに追加。

```
<c:clustering>
  <ejb-name>DDBasedClusteredSFSB</ejb-name>
  <c:clustered>true</c:clustered>
</c:clustering>
```

JBoss EAP 7 以降では、クラスタリング動作を有効にする必要がなくなりました。デフォルトでは、HA プロファイルを使用してサーバーが起動された場合は SFSB の状態が自動的にレプリケートされます。以下の方法の 1 つを使用すると、このデフォルト動作を無効にできます。

- Enterprise Java Beans 3.2 仕様の新機能である **@Stateful(passivationCapable=false)** を使用すると、単一のステートフルセッション Bean のデフォルト動作を無効にできます。
- サーバー設定の **ejb3** サブシステムの設定で、この動作をグローバルに無効化します。

注記

@Clustered アノテーションがアプリケーションから削除されると、このアノテーションは無視され、アプリケーションのデプロイメントには影響しません。

7.21.5. クラスタリングサービスの変更

JBoss EAP 6 では、クラスタリングサービスの API はプライベートモジュールでサポートされませんでした。

JBoss EAP 7 には、アプリケーションによって使用されるパブリッククラスタリングサービス API が導入されました。新しいサービスは、ライトウェイトで簡単にインジェクトできるよう設計されています。 外部の依存関係は必要ありません。

- 新しい **org.wildfly.clustering.group.Group** インターフェイスを使用すると、現在のクラスター状態へアクセスでき、クラスターメンバーシップの変更をリッスンできます。
- 新しい **org.wildfly.clustering.dispatcher.CommandDispatcher** インターフェイスを使用すると、すべてのノードまたはノードの選択されたサブセットのクラスターでコードを実行できます。

これらのサービスは、JBoss EAP 5 の **HAPartition**、JBoss EAP 6 の **GroupCommunicationService**、**GroupMembershipNotifier**、および **GroupRpcDispatcher** に代わるサービスです。 これらの API は以前のリリースで利用できました。

詳細は、JBoss EAP 7.4 **開発ガイド** の クラスタリングサービスのパブリック API を参照してください。

7.21.6. クラスタリング HA シングルトンの移行

JBoss EAP 6 では、クラスター全体の HA シングルトンサービスに使用できるパブリック API がありませんでした。プライベート **org.jboss.as.clustering.singleton.*** クラスを使用した場合は、アプリケーションを JBoss EAP 8 に移行するときに、新しいパブリック **org.wildfly.clustering.singleton.*** パッケージを使用するようにコードを変更する必要があります。

HA シングルトンに関する詳細は、JBoss EAP 7.4 **開発ガイド** の HA シングルトンサービス を参照してください。HA シングルトンのデプロイメントに関する詳細は、JBoss EAP 7.4 **開発ガイド** の HA シングルトンデプロイメント を参照してください。

7.22. コンテキストタイプを使用した CONTEXTSERVICE のカスタマイズ

Jakarta EE Concurrency 3.0 の一部として、コンテキストタイプを使用して **ContextService** プロパティーをカスタマイズできます。トランザクションコンテキストはそのようなタイプの 1 つです。トランザクションコンテキストは、**use-transaction-setup-provider** リソース定義属性の使用を置き換えます。**use-transaction-setup-provider** 属性が **true** に設定されている場合、トランザクションコンテキストはクリアされ、この属性が **false** に設定されている場合は、トランザクションコンテキストは変更されません。

Red Hat はベンダー固有の設定をサポートしなくなったため、そのような resource-definition 属性は非推奨になりました。JBoss EAP 7 では、デフォルト設定で **use-transaction-setup-provider** 属性が **false** と定義されていました。これは、コンテキストタスクがスレッド上で実行されたときにトランザクションコンテキストが変更されなかったことを意味します。デフォルトでは、JBoss EAP 8 では、デフォルトの **ContextService** プロパティーが Jakarta EE Concurrency 3.0 仕様に準拠しており、コンテキストタスクが実行される前にトランザクションコンテキストをクリアします。

別の **ContextService** を使用するには、**ContextServiceDefinition** アノテーションを使用するか、XML でこれを指定して、デプロイメントで定義する必要があります。

7.23. 非推奨の INITIALCONTEXT クラスの削除

org.jboss.naming.remote.client.InitialContextFactory クラスが JBoss EAP 8 で削除されました。JBoss EAP 7 では、**org.jboss.naming.remote.client.InitialContextFactory** クラスは非推奨となり、**org.wildfly.naming.client.WildFlyInitialContextFactory** クラスに置き換えられました。この変更を反映するには、ソースコードまたは設定ファイルを移行する必要があります。

命名設定の変更:

- ユーザーアプリケーションがシステムプロパティーまたは環境プロパティーを使用している場合、**java.naming.factory.initial** プロパティーを **java.naming.factory.initial=org.jboss.naming.remote.client.InitialContextFactory** から **java.naming.factory.initial=org.wildfly.naming.client.WildFlyInitialContextFactory** へ移行する必要があります。

- ユーザーアプリケーションが **<soapjms:jndiInitialContextFactory>** を含む WSDL コントラクトを使用している場合、その値を**<soapjms:jndiInitialContextFactory>org.jboss.naming.remote.client.InitialContextFactory<soapjms:jndiInitialContextFactory>** から **<soapjms:jndiInitialContextFactory>org.wildfly.naming.client.WildFlyInitialContextFactory<soapjms:jndiInitialContextFactory>** に移行する必要があります。

- ユーザーアプリケーションが Java コードを使用してリモートネーミングを設定している場合は、**Properties env = new Properties();env.put(Context.INITIAL_CONTEXT_FACTORY, org.jboss.naming.remote.client.InitialContextFactory.class.getName());** から **env.put(Context.INITIAL_CONTEXT_FACTORY, org.wildfly.naming.client.WildFlyInitialContextFactory.class.getName());** へ更新する必要があります。

以下にリストされている **org.wildfly.naming.client.ProviderEnvironment** クラスのメソッドは JBoss EAP 7 で非推奨となり、コード、ドキュメント、および Web プロパティーから問題のある用語を置き換える Red Hat の取り組みの一環として JBoss EAP 8 で削除されました。詳細は、Red Hat CTO である Chris Wright のメッセージ をご覧ください。

削除されたメソッドを含むコードは、対応する置換を使用してリファクタリングする必要があります。

- **getBlackList()** は **getBlockList()** に置き換えられる
- **updateBlacklist(URI)** は **updateBlockList(URI)** に置き換えられる
- **dropFromBlacklist(URI)** は、**dropFromBlocklist(URI)** に置き換えられる

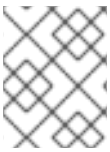
7.24. リソースアダプター

Jakarta Connectors Resource Adapter を使用すると、アプリケーションはどのメッセージングプロバイダーとも通信できます。MDB や他の Jakarta Enterprise Beans、さらには Servlet などの Jakarta JEE コンポーネントがメッセージを送受信する方法を設定します。

7.24.1. IBM MQ リソースアダプターのデプロイ

IBM MQ は、IBM のメッセージ指向ミドルウェア (MOM) 製品で、分散システム上のアプリケーションが互いに通信できるようにします。これは、メッセージとメッセージキューを使用して実行できます。IBM MQ は、メッセージキューにメッセージを配信し、メッセージチャネルを使用してデータを他のキューマネージャーへ転送します。IBM MQ の詳細は、IBM の製品 Web サイトの IBM MQ を参照してください。

summary
IBM MQ は、JBoss EAP 8.0 の外部 Java Message Service プロバイダーとして設定できます。このセクションでは、JBoss EAP で IBM MQ リソースアダプターをデプロイして設定する手順を説明します。このデプロイメントと設定は、管理 CLI ツールまたは Web ベースの管理コンソールを使用して実行できます。IBM MQ のサポートされる設定に関する最新情報は、JBoss EAP supported configurations を参照してください。



注記

設定の変更を有効にするには、IBM MQ リソースアダプターの設定後にシステムを再起動する必要があります。

JBoss EAP 8.0 は Jakarta EE 10 実装であるため、すべての EE API に使用されるパッケージは javax から jakarta に変更され、Jakarta EE 10 準拠のリソースアダプターが必要となります。JBoss EAP 7.x 以前で IBM MQ リソースアダプターを使用していた場合は、この jakarta 名前空間を使用する IBM MQ リソースアダプターである **wmq.jakarta.jmsra.rar** を使用する必要があります。

- **wmq.jmsra.rar** の以前のリソースアダプター設定を削除してデプロイ解除し、**wmq.jakarta.jmsra.rar** を使用します。

- **wmq.jakarta.jmsra.rar** をデプロイし、このセクションで説明されている手順に従って設定します。

前提条件

作業を開始する前に、IBM MQ リソースアダプターのバージョンを検証し、その設定プロパティーを理解する必要があります。

- IBM MQ リソースアダプターは、**wmq.jakarta.jmsra.rar** というリソースアーカイブ (RAR) ファイルとして提供されます。**wmq.jakarta.jmsra.rar** ファイルは **/opt/mqm/java/lib/jca/wmq.jakarta.jmsra.rar** から取得できます。JBoss EAP の各リリースでサポートされる特定のバージョンに関する詳細は、JBoss EAP でサポートされる構成 を参照してください。
- 以下の IBM MQ 設定値を知っている必要があります。これらの値については、IBM MQ 製品のドキュメントを参照してください。
  - **MQ_QUEUE_MANAGER**: IBM MQ キューマネージャーの名前
  - **MQ_HOST_NAME**: IBM MQ キューマネージャーへの接続に使用されるホスト名
  - **MQ_CHANNEL_NAME**: IBM MQ キューマネージャーへの接続に使用されるサーバーチャネル
  - **MQ_QUEUE_NAME**: 宛先キューの名前
  - **MQ_TOPIC_NAME**: 宛先トピックの名前
  - **MQ_PORT**: IBM MQ キューマネージャーへの接続に使用されるポート
  - **MQ_CLIENT**: トランスポートタイプ
- 送信接続の場合は、以下の設定値も理解している必要があります。
  - **MQ_CONNECTIONFACTORY_NAME**: リモートシステムへの接続を提供する接続ファクトリーインスタンスの名前

手順

以下は、IBM のデフォルト設定であり、変更される可能性があります。詳細は IBM MQ のドキュメントを参照してください。

1. まず、**wmq.jakarta.jmsra.rar** ファイルを **EAP_HOME/standalone/deployments/** ディレクトリーにコピーして、リソースアダプターを手動でデプロイします。
2. 次に、管理 CLI を使用してリソースアダプターを追加し、設定します。

```
/subsystem=resource-adapters/resource-adapter=wmq.jakarta.jmsra.rar:add(archive=wmq.jakarta.jmsra.rar, transaction-support=XATransaction)
```

transaction-support 要素が **XATransaction** に設定されたことに注意してください。トランザクションを使用する場合は、以下の例にあるように、XA リカバリーデータソースのセキュリティードメインを提供してください。

```
/subsystem=resource-adapters/resource-adapter=test/connection-definitions=test:write-attribute(name=recovery-security-domain,value=myDomain)
```

XA リカバリーに関する詳細は、JBoss EAP 7.4 **設定ガイド** の XA リカバリーの設定 を参照してください。

トランザクション以外のデプロイメントの場合は、**transaction-support** の値を **NoTransaction** に変更します。

```
/subsystem=resource-adapters/resource-adapter=wmq.jakarta.jmsra.rar:add(archive=wmq.jakarta.jmsra.rar, transaction-support=NoTransaction)
```

3. リソースアダプターが作成されたので、必要な設定要素を追加できます。

   a. キューの **admin-object** を追加し、そのプロパティーを設定します。

```
/subsystem=resource-adapters/resource-adapter=wmq.jakarta.jmsra.rar/admin-objects=queue-ao:add(class-name=com.ibm.mq.jakarta.connector.outbound.MQQueueProxy, jndi-name=java:jboss/MQ_QUEUE_NAME)

/subsystem=resource-adapters/resource-adapter=wmq.jakarta.jmsra.rar/admin-objects=queue-ao/config-properties=baseQueueName:add(value=MQ_QUEUE_NAME)

/subsystem=resource-adapters/resource-adapter=wmq.jakarta.jmsra.rar/admin-objects=queue-ao/config-properties=baseQueueManagerName:add(value=MQ_QUEUE_MANAGER)
```

   b. トピックの **admin-object** を追加し、そのプロパティーを設定します。

```
/subsystem=resource-adapters/resource-adapter=wmq.jakarta.jmsra.rar/admin-objects=topic-ao:add(class-name=com.ibm.mq.jakarta.connector.outbound.MQTopicProxy, jndi-name=java:jboss/MQ_TOPIC_NAME)

/subsystem=resource-adapters/resource-adapter=wmq.jakarta.jmsra.rar/admin-objects=topic-ao/config-properties=baseTopicName:add(value=MQ_TOPIC_NAME)

/subsystem=resource-adapters/resource-adapter=wmq.jakarta.jmsra.rar/admin-objects=topic-ao/config-properties=brokerPubQueueManager:add(value=MQ_QUEUE_MANAGER)
```

   c. 管理接続ファクトリーの接続定義を追加し、そのプロパティーを設定します。

```
/subsystem=resource-adapters/resource-adapter=wmq.jakarta.jmsra.rar/connection-definitions=mq-cd:add(class-name=com.ibm.mq.jakarta.connector.outbound.ManagedConnectionFactoryImpl, jndi-name=java:jboss/MQ_CONNECTIONFACTORY_NAME, tracking=false)

/subsystem=resource-adapters/resource-adapter=wmq.jakarta.jmsra.rar/connection-definitions=mq-cd/config-properties=hostName:add(value=MQ_HOST_NAME)

/subsystem=resource-adapters/resource-adapter=wmq.jakarta.jmsra.rar/connection-definitions=mq-cd/config-properties=port:add(value=MQ_PORT)

/subsystem=resource-adapters/resource-adapter=wmq.jakarta.jmsra.rar/connection-definitions=mq-cd/config-properties=channel:add(value=MQ_CHANNEL_NAME)
```

```
/subsystem=resource-adapters/resource-adapter=wmq.jakarta.jmsra.rar/connection-definitions=mq-cd/config-properties=transportType:add(value=MQ_CLIENT)

/subsystem=resource-adapters/resource-adapter=wmq.jakarta.jmsra.rar/connection-definitions=mq-cd/config-properties=queueManager:add(value=MQ_QUEUE_MANAGER)
```

4. JBoss EAP の EJB3 メッセージングシステムのデフォルトプロバイダーを JBoss EAP 8.0 メッセージングから IBM MQ に変更する場合は、管理 CLI を使用して **ejb3** サブシステムを次のように変更します。

```
/subsystem=ejb3:write-attribute(name=default-resource-adapter-name,value=wmq.jakarta.jmsra.rar)
```

5. 以下のように、MDB コードに **@ActivationConfigProperty** アノテーションと **@ResourceAdapter** アノテーションを設定します。

```
@MessageDriven(name="IbmMqMdb", activationConfig = {
    @ActivationConfigProperty(propertyName = "destinationType",propertyValue = "jakarta.jms.Queue"),
    @ActivationConfigProperty(propertyName = "useJNDI", propertyValue = "false"),
    @ActivationConfigProperty(propertyName = "hostName", propertyValue = "MQ_HOST_NAME"),
    @ActivationConfigProperty(propertyName = "port", propertyValue = "MQ_PORT"),
    @ActivationConfigProperty(propertyName = "channel", propertyValue = "MQ_CHANNEL_NAME"),
    @ActivationConfigProperty(propertyName = "queueManager", propertyValue = "MQ_QUEUE_MANAGER"),
    @ActivationConfigProperty(propertyName = "destination", propertyValue = "MQ_QUEUE_NAME"),
    @ActivationConfigProperty(propertyName = "transportType", propertyValue = "MQ_CLIENT")
})

@ResourceAdapter(value = "wmq.jakarta.jmsra-VERSION.rar")
@TransactionAttribute(TransactionAttributeType.NOT_SUPPORTED)
public class IbmMqMdb implements MessageListener {
}
```

@ResourceAdapter 値の **VERSION** は、RAR の名前に含まれる実際のバージョンに置き換えてください。

6. リソースアダプターをアクティベートします。

```
/subsystem=resource-adapters/resource-adapter=wmq.jakarta.jmsra.rar:activate()
```

7.24.1.1. IBM MQ リソースアダプターの制限と既知の問題

以下の表に、IBM MQ リソースアダプターの既知の問題をまとめています。バージョン列のチェックマーク (✔) は、そのバージョンのリソースアダプターに問題があることを示しています。

表7.2 IBM MQ リソースアダプターの既知の問題

JIRA	問題の説明	IBM MQ 9
JBEAP-503	IBM MQ リソースアダプターは、**Queue.toString()** メソッドと **QueueBrowser.getQueue().toString()** メソッドに異なる文字列の値を返します。**Queue** は **com.ibm.mq.connector.outbound.MQQueueProxy** クラスのインスタンスで、**QueueBrowser.htmlQueueBrowser.getQueue()** メソッドから返される **com.ibm.mq.jms.MQQueue** クラスとは異なります。これらのクラスには、**toString()** メソッドの異なる実装が含まれます。これらの **toString()** メソッドを使用して同じ値を返すことができないことに注意してください。	✔
JBEAP-511、JBEAP-550、JBEAP-3686	以下の制限は、IBM MQ のメッセージプロパティー名に適用されます。 ● デプロイメント記述子の **activation-config** セクションでは、_、**&**、| などの特殊文字を使用して **destinationName** プロパティーを設定しないでください。これらの文字を使用すると、MDB デプロイメントは **com.ibm.msg.client.jms.DetailedInvalidDestinationException** 例外で失敗します。 ● デプロイメント記述子の **activation-config** セクションで、**java:/** 接頭辞を使用して **destinationName** プロパティーを設定しないでください。この接頭辞を使用すると、MDB デプロイメントは **com.ibm.msg.client.jms.DetailedInvalidDestinationException** 例外で失敗します。 ● IBM MQ JMS クラスで使用するために予約されているため、プロパティーを JMS または usr.JMS で開始してはいけません。例外は、IBM Knowledge Center の Web サイトに記載されています。 メッセージプロパティー名の制限の完全なリストについては、IBM Knowledge Center Web サイトの Property name restrictions for IBM MQ, Version 9.0 を参照してください。	✔
JBEAP-549	**@ActivationConfigProperty** アノテーションを使用して MDB の **destination** プロパティー名の値を指定するときは、すべて大文字を使用する必要があります。例を以下に示します。 `@ActivationConfigProperty(propertyName = "destination", propertyValue = "QUEUE")`	✔

JIRA	問題の説明	IBM MQ 9

JBEAP-624	IBM MQ リソースアダプターを使用して **@JMSConnectionFactoryDefinition** アノテーションを使用して Jakarta EE デプロイメントに接続ファクトリーを作成する場合は、**resourceAdapter** プロパティーを指定する必要があります。指定しないと、デプロイメントは失敗します。
	
	`@JMSConnectionFactoryDefinition(`
	`  name = "java:/jms/WMQConnectionFactory",`
	`  interfaceName = "javax.jms.ConnectionFactory",`
	`  resourceAdapter = "wmq.jmsra",`
	`  properties = {`
	`  "channel=<channel>",`
	`  "hostName=<hostname_wmq_broker>",`
	`  "transportType=<transport_type>",`
	`  "queueManager=<queue_manager>"`
	`  }`
	`)`
		✔

JBEAP-2339	IBM MQ リソースアダプターは、接続開始前からキューとトピックからメッセージを読み取ることができます。これは、接続開始前にコンシューマーがメッセージを消費できることを意味します。この問題を回避するには、**external-context** を使用してリモート IBM MQ ブローカーが作成した接続ファクトリーを使用し、IBM MQ リソースアダプターが作成した接続ファクトリーは使用しないでください。	✔

JBEAP-3685	**<transaction-support>XATransaction</transaction-support>** が設定されると、インジェクションを使用して作成されたか、手動で作成されたかに関係なく、**JMSContext** は常に **JMSContext.SESSION_TRANSACTED** です。
	
	以下のコード例では、**@JMSSessionMode(JMSContext.DUPS_OK_ACKNOWLEDGE)** は無視され、**JMSContext** は **JMSContext.SESSION_TRANSACTED** のままになります。
	
	`@Inject`
	`@JMSConnectionFactory("jms/CF")`
	`@JMSPasswordCredential(userName="myusername",`
	`password="mypassword")`
	`@JMSSessionMode(JMSContext.DUPS_OK_ACKNOWLEDGE)`
	`transient JMSContext context3;`
		✔

JIRA	問題の説明	IBM MQ 9
JBEAP-14633	JMS 仕様によると、**QueueSession** インターフェイスを使用してパブリッシュ/サブスクライブドメインに固有のオブジェクトを作成することはできず、**Session** から継承する特定のメソッドは、**javax.jms.IllegalStateException** を出力する必要があります。このようなメソッドの 1 つが、**QueueSession.createTemporaryTopic()** です。**javax.jms.IllegalStateException** を出力する代わりに、IBM MQ リソースアダプターは **java.lang.NullPointerException** を出力します。	✔
JBEAP-14634	**MQTopicProxy.getTopicName()** は、IBM MQ ブローカーによって設定されていたものとは異なるトピック名を返します。たとえば、トピック名が **topic://MYTOPIC?XMSC_WMQ_BROKER_PUBQ_QMGR=QM** に設定された場合、**MQTopicProxy** は **topic://MYTOPIC** を返します。	✔
JBEAP-14636	**JMSContext** のデフォルトの **autoStart** 設定は **false** です。これは、**JMSContext** が使用するベースとなる接続が、コンシューマーの作成時に自動的に開始されないことを意味します。この設定はデフォルトの **true** にする必要があります。	✔
JBEAP-14640	無効な認証情報が使用されると、IBM MQ リソースアダプターは、**JMSSecurityException** ではなく **DetailedJMSException** を出力し、以下のエラーをサーバーコンソールに記録します。 WARN [org.jboss.jca.core.connectionmanager.pool.strategy.PoolByCri] (EJB default - 7) IJ000604: Throwable while attempting to get a new connection: null: com.ibm.mq.connector.DetailedResourceException: MQJCA1011: Failed to allocate a JMS connection., error code: MQJCA1011 An internal error caused an attempt to allocate a connection to fail. See the linked exception for details of the failure. 以下は、この問題を引き起こす可能性があるコードの例です。 QueueConnection qc = queueConnectionFactory.createQueueConnection("invalidUserName", "invalidPassword");	✔

JIRA	問題の説明	IBM MQ 9
JBEAP-14642	**MQMessageProducer.send(Destination destination, Message message)** メソッドと **MQMessageProducer.send(Destination destination, Message message, int deliveryMode, int priority, long timeToLive, CompletionListener completionListener)** メソッドのリソースアダプターによる無効なクラスキャスト変換により、IBM MQ リソースアダプターは **JMSException** を出力し、サーバーコンソールに以下のエラーメッセージをログに記録します。 > SVR-ERROR: Expected JMSException, received com.ibm.mq.connector.outbound.MQQueueProxy cannot be cast to com.ibm.mq.jms.MQDestination これは、キューまたはトピックルックアップで使用される JNDI 名が **com.ibm.mq.connector.outbound.MQQueueProxy/MQTopicProxy** であるためです。	✔
JBEAP-14643	**JMSProducer** インターフェイスの **setDeliveryDelay(expDeliveryDelay)** メソッドは設定を変更しません。このメソッドを呼び出しても、デフォルト設定の **0** のままです。	✔
JBEAP-14670	**UserTransaction.begin()** の前に作成される **QueueSession** で作業が行われる場合、その作業はトランザクションの一部とは見なされません。これは、このセッションを使用してキューに送信されたメッセージは **UserTransaction.commit()** によってコミットされず、**UserTransaction.rollback()** の後、メッセージがキューに残ることを意味します。	✔
JBEAP-14675	接続を閉じた直後に、同じ **clientID** で **JMSContext** を作成すると、IBM MQ リソースアダプターはサーバーコンソールに以下のエラーを断続的に記録します。 > ERROR [io.undertow.request] (default task-1) UT005023: Exception handling request to /jmsServlet-1.0-SNAPSHOT/: com.ibm.msg.client.jms.DetailedJMSRuntimeException: MQJCA0002: An exception occurred in the IBM MQ layer. See the linked exception for details. A call to IBM MQ classes for Java(tm) caused an exception to be thrown. この問題は、同じ **clientID** の接続が閉じられた後に新しい **JMSContext** の作成で遅延がある場合は発生しません。	✔

JIRA	問題の説明	IBM MQ 9
JBEAP-15535	コンテナー管理トランザクション (CMT) において、ステートフルセッション Bean がトピックにメッセージを送信しようとすると、メッセージ送信が失敗し、以下のメッセージが示されます。 > SVR-ERROR: com.ibm.msg.client.jms.DetailedJMSException: JMSWMQ2007: Failed to send a message to destination '**MDB_NAME TOPIC_NAME**' スタックトレースは、以下の例外によって発生したことを示しています。 > com.ibm.mq.MQException: JMSCMQ0001: IBM MQ call failed with compcode '2' ('MQCC_FAILED') reason '2072' ('MQRC_SYNCPOINT_NOT_AVAILABLE')	
JBEAP-25561	メッセージが **JMSReplyTo** ヘッダーを設定して宛先に送信されると、そのメッセージは IBM MQ 9 ブローカーに到達した後に変更されます。その結果、応答のリプライメッセージは、変更された **JMS_REPLY** ヘッダーで定義された宛先に送信されます。 たとえば、**JMSReplyTo** ヘッダーが、ヘッダー **message.setJMSReplyTo(queue)** を使用して **queue:///MYQUEUE** という名前でキューに設定されている場合、そして、**QM** という名前のキューマネージャーを備えた IBM MQ 9.3 ブローカーに送信されると、その名前は **queue://QM/MYQUEUE** に変更されます。	✔

7.24.2. Apache Log4j バージョン 1 API の削除

JBoss EAP 8 以降、Apache **Log4j** バージョン 1 API のサポートが停止されました。**log4j.jar** および **log4j** 設定をパッケージ化していないアプリケーションは更新する必要があります。

影響:

ログメッセージは、ロギングサブシステムに基づいてルーティングされなくなります。アプリケーションが **log4j.jar** をパッケージ化しておらず、次のステートメントのいずれかに該当する場合は、移行の変更が必要です。

- デプロイメントで **log4j** を使用し、**log4j** 設定ファイルを含めない場合は、新しいロギングファサードに移行するか、デプロイメントに **log4j** 設定を追加する必要があります。
- デプロイメントで **log4j.xml**、**log4j.properties**、または **jboss-log4j.xml** ファイルを使用し、アプリケーションで **log4j.jar** をパッケージ化していない場合。**jboss-log4j.xml** ファイルの場合は、ファイルの名前を **log4j.xml** に変更する必要があります。
- カスタムハンドラーの JBoss EAP Logging サブシステムで **log4j** v1 アペンダーを使用する場合、それはサポートされなくなります。
- アプリケーションクラスが **org.apache.log4j.Logger** などのクラスをインポートする場合。

- アプリケーションに **jboss-deployment-structure.xml** が含まれているか、**org.jboss.log4j.logmanager** へのモジュール依存関係を宣言する **MANIFEST.MF** で指定された**Dependencies:** がある場合、これらの依存関係を削除する必要があります。

移行:

- Apache **Log4jv2** クラスを使用するか、JBoss EAP 8 が提供する他の Logging API のいずれかを使用するようにアプリケーションクラスを更新します。
- **org.apache.log4j.Logger (log4j v1)** クラスを **org.apache.logging.log4j.Logger (log4j v2)** に変更します。
- アプリケーションが **log4j.properties**、**log4j.xml**、または **jboss-log4j.xml** をパッケージ化する場合、次のことを行う必要があります。
  - JBoss EAP 設定でロギングを設定します。
  - **log4jv2** 設定ファイルはアプリケーションでサポートされていないため、アプリケーションで **logging.properties** を設定します。

OR

- Logging API 用に JBoss EAP 8 に依存するのではなく、Apache **Log4j** バージョン 1 JAR をアプリケーションにパッケージ化します。ロギングサブシステムの **jboss-deployment-structure.xml exclude-subsystems** によって、アプリケーションから JBoss Logging API を除外することもできます。

さらなる詳細:

特定のデプロイメントに対する暗黙的なロギング依存関係の無効化

- アプリケーションの **jboss-deployment-structure.xml** で、次のように **exclude-subsystems** を設定してロギングサブシステムを除外します。

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-structure:1.2">
  <deployment>
    <exclude-subsystems>
      <subsystem name="logging"/>
    </exclude-subsystems>
  </deployment>
</jboss-deployment-structure>
```

- アプリケーションが EAR ファイルで、**example.war** という名前のサブデプロイメントがある場合、**jboss-deployment-structure.xml** ファイルは EAR ファイルの場所 / **META-INF/jboss-deployment-structure.xml** にあり、ロギングサブシステムは次のようなサブデプロイメントで宣言することで除外されます。

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-structure:1.2">
  <sub-deployment name="example.war">
    <exclude-subsystems>
      <subsystem name="logging"/>
    </exclude-subsystems>
  </sub-deployment>
</jboss-deployment-structure>
```

すべてのデプロイメントの暗黙的なロギング依存関係を無効にする

デプロイメントで Logging API をデフォルトで使用できないようにするには、次の CLI コマンドを使用して **add-logging-api-dependency** を **false** に設定します。

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

JBoss モジュールと Logging API を依存関係として設定するには、**jboss-deployment-structural.xml** または **MANIFEST.MF** 設定ファイルを変更します。

```
<subsystem xmlns="urn:jboss:domain:logging:8.0">
    <add-logging-api-dependencies value="false"/>
  ...
</subsystem>
```

注記

アプリケーションが Apache **Log4j v1** JAR と **log4j** 設定をアプリケーションにパッケージ化する場合: * アプリケーションのロギングは EAP によって管理されなくなり、アプリケーションによって管理されます。* ロギングフレームワークは、特定のログファイルに書き込むことが想定されていることから、予期しない結果が発生する可能性があるため、アプリケーションは **server.log** への書き込みを試みることはできません。

詳細は、Apache Log4j version 1 is no longer provided in JBoss EAP 8.0 を参照してください。

第8章 その他の変更点

このセクションでは、このリリースにおけるさまざまな変更について概説します。

8.1. JBOSS EAP NATIVES および APACHE HTTP SERVER の提供に関する変更

JBoss EAP 8.0 Native は、このリリースでは JBoss EAP 6 とは異なる方法で提供されます。一部のコンポーネントには、多くの Red Hat JBoss ミドルウェア製品に共通する補完ソフトウェアのセットである Red Hat JBoss Core Services 製品が含まれています。新製品では更新のより迅速な配布と一貫性のある更新が可能になります。JBoss Core Services 製品は、Red Hat カスタマーポータルの専用の場所からダウンロードできます。

- 以下の表では、リリースごとの提供方法の違いを示しています。

パッケージ	JBoss EAP 6	JBoss EAP 8.0
メッセージング用 AIO ネイティブ	別個の "Native Utilities" ダウンロードで製品とともに提供されます。	JBoss EAP ディストリビューションに含まれます。
Apache HTTP サーバー	別個の "Apache HTTP Server" ダウンロードで製品とともに提供。	新しい JBoss Core Services 製品とともに提供されます。
mod_cluster、mod_jk, isapi、および nsapi コネクター	別個の "Webserver コネクターネイティブ" ダウンロードで製品とともに提供。	新しい JBoss Core Services 製品とともに提供されます。
JSVC	別個の "Native Utilities" ダウンロードで製品とともに提供されます。	新しい JBoss Core Services 製品とともに提供されます。
OpenSSL	別個の "Native Utilities" ダウンロードで製品とともに提供されます。	新しい JBoss Core Services 製品とともに提供されます。
tcnatives	別個の "ネイティブコンポーネント" ダウンロードで製品とともに提供。	tcnatives のサポートは JBoss EAP 7 で削除されました。

JBoss EAP Natives および Apache HTTP Server の追加の変更

- 以下の変更点にも注意してください。
  - Red Hat Enterprise Linux RPM チャンネルからの Apache HTTP Server と使用される mod_cluster および mod_jk コネクターのサポートは廃止されました。Red Hat Enterprise Linux RPM チャネルから Apache HTTP Server を実行し、JBoss EAP 8.0 サーバーの負荷分散を設定する必要がある場合は、以下の 1 つを行います。
    - JBoss Core Services によって提供される Apache HTTP Server を使用します。
    - フロントエンドロードバランサーとして機能するように JBoss EAP 8.0 を設定できます。詳細は、JBoss EAP 7.4 **設定ガイド** の JBoss EAP をフロントエンドロードバランサーとして設定する を参照してください。

- 認証済みのサポートされるマシンに Apache HTTP Server をデプロイした後、そのマシンでロードバランサーを実行することができます。サポートされる設定のリストについては、JBoss EAP 7.4 **設定ガイド** の HTTP コネクターの概要 を参照してください。

- 詳細は、Apache HTTP Server インストールガイド の **JBoss Core Services** を参照してください。

8.2. AMAZON EC2 でのデプロイメントの変更

JBoss EAP 7 の Amazon Machine Images (AMI) にいくつかの変更が加えられました。ここでは、これらの変更の一部を簡単に説明します。

- Amazon EC2 で非クラスター化およびクラスター化された JBoss EAP インスタンスとドメインを開始する方法が、大幅に変更されました。

- JBoss EAP 6 では、設定は **User Data:** フィールドに依存していました。JBoss EAP 7 では、**User Data:** フィールドの設定を解析し、インスタンスの起動時にサーバーを自動的に起動する AMI スクリプトが削除されました。

- Red Hat JBoss Operations Network エージェントが JBoss EAP 6 にインストールされました。JBoss EAP 7.0 以降は、個別にインストールする必要があります。

Amazon EC2 での JBoss EAP 7 のデプロイに関する詳細は、**Deploying JBoss EAP on Amazon Web Services** を参照してください。

8.3. 共有モジュールを含むアプリケーションの削除

JBoss EAP 7.1 サーバーおよび Maven プラグインに導入された変更により、アプリケーションを削除しようとすると次の障害が発生する可能性があります。このエラーは、アプリケーションに相互に対話または依存するモジュールが含まれると発生します。

```
WFLYCTL0184:    New missing/unsatisfied dependencies
```

たとえば、アプリケーションに **application-A** と **application-B** の 2 つの Maven WAR プロジェクトモジュールが含まれ、これらのモジュールは **data-sharing** モジュールが管理するデータを共有するとします。

このアプリケーションをデプロイする場合、最初に共有された **data-sharing** モジュールをデプロイした後、そのモジュールに依存するモジュールをデプロイする必要があります。デプロイメントの順番は、親の **pom.xml** ファイルの **<modules>** 要素に指定されます。これは、JBoss EAP 6.4 から JBoss EAP 8.0 に当てはまります。

JBoss EAP 7.1 よりも前のリリースでは、以下のコマンドを使用すると、親プロジェクトのルートからそのアプリケーションのアーカイブをすべてアンデプロイできました。

```
$ mvn wildfly:undeploy
```

JBoss EAP 7.1 以上では、最初に共有されたモジュールを使用するアーカイブをアンデプロイした後、共有されたモジュールをアンデプロイする必要があります。プロジェクトの **pom.xml** ファイルを使用してアンデプロイの順序を指定できないため、手作業でモジュールをアンデプロイする必要があります。これには、親ディレクトリーのルートから以下のコマンドを実行します。

```
$ mvn wildfly:undeploy -pl application-A,application-B
$ mvn wildfly:undeploy -pl data-shared
```

この更新されたデプロイメント解除動作はより正確になり、不安定なデプロイメント状態に陥ることがなくなります。

8.4. ADD-USER スクリプトの変更

パスワードポリシーが変更になったため、JBoss EAP 7 では **add-user** のスクリプト動作が変更になりました。JBoss EAP 6 のパスワードポリシーは厳格でした。そのため、**add-user** スクリプトは最低要件に満たない弱いパスワードを拒否しました。JBoss EAP 7 以降では、弱いパスワードが許可され、警告が表示されます。詳細は、JBoss EAP 7.4 **設定ガイド** の Add-User ユーティリティーのパスワード制限の設定 を参照してください。

8.5. OSGI サポートの削除

JBoss EAP 6.0 GA の最初のリリースには、OSGi 仕様の実装である JBoss OSGi がテクノロジープレビューとして含まれていました。JBoss EAP 6.1.0 では、JBoss OSGi はテクノロジープレビューからサポート対象外に格下げされました。

JBoss EAP 6.1.0 では、**configadmin** および **osgi** 拡張モジュールとスタンドアロンサーバーのサブシステム設定が個別の **EAP_HOME/standalone/configuration/standalone-osgi.xml** 設定ファイルに移されました。サポート対象外であるこの設定ファイルを移行すべきではないため、JBoss OSGi サポートの廃止はスタンドアロンサーバー設定の移行には影響しないはずです。他のスタンドアロン設定ファイルを編集して **osgi** または **configadmin** を設定した場合は、その設定を削除する必要があります。

マネージドドメインでは、 JBoss EAP 6.1.0 リリースで **osgi** 拡張およびサブシステム設定が **EAP_HOME/domain/configuration/domain.xml** ファイルから削除されました。しかし、**configadmin** モジュール拡張およびサブシステム設定は **EAP_HOME/domain/configuration/domain.xml** ファイルから削除されていません。JBoss EAP 7 以降、この設定はサポートされなくなったため、削除する必要があります。

8.6. JAVA の SOAP WITH ATTACHEMENT の変更

JBoss EAP 8.0 への移行時に、SAAJ 3.0 仕様に準拠するようにユーザー定義の SOAP ハンドラーを更新します。

関連情報

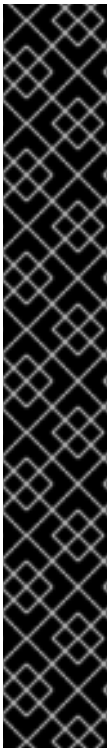
- 添付ファイル付きの Jakarta Soap

第9章 ELYTRON への移行

JBoss EAP 7.1 には Elytron が導入されました。Elytron はスタンドアロンサーバーとマネージドドメインの両方のアクセスを管理および設定できる単一の統合フレームワークです。JBoss EAP サーバーにデプロイされたアプリケーションのセキュリティーアクセスを設定するために使用することもできます。

JBoss EAP 8.0 以降では、レガシーセキュリティーサブシステムはアプリケーションの移行に使用できないため、Elytron を使用する必要があります。

9.1. ELYTRON の概要



重要

Elytron のアーキテクチャーと、PicketBox をベースとしたレガシー security サブシステムのアーキテクチャーは大変異なります。Elytron では、現在操作しているセキュリティー環境で操作できるようにソリューションを作成しようとしますが、PicketBox 設定オプションと同等の設定オプションがすべて Elytron に**あるわけではありません**。

レガシーセキュリティー実装の使用時に、Elytron を使用して同等の機能を実現するための情報がドキュメントで見つからない場合は、以下の方法で情報を見つけることができます。

- Red Hat 開発サブスクリプション をお持ちの場合は、Red Hat カスタマーポータルの サポートケース、ソリューション、および ナレッジ記事 にアクセスできます。以下のように 技術サポート でケースを作成したり、WildFly コミュニティーでヘルプを受けることもできます。
- Red Hat 開発サブスクリプションをお持ちでない場合でも、Red Hat カスタマーポータルの ナレッジ記事 にはアクセスできます。また、ユーザーフォーラムや ライブチャット に参加して、WildFly コミュニティーで質問することもできます。WildFly コミュニティーが提供する技術は、Elytron のエンジニアリングチームによって活発に管理されています。

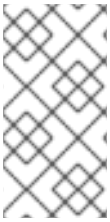
elytron サブシステムで使用できる新リソースの概要については、JBoss EAP 7.4 **セキュリティーアーキテクチャー** の Elytron サブシステムのリソース を参照してください。

9.2. セキュアな VAULT およびプロパティーの移行

Elytron を使用するには、安全な認証情報ストレージに安全な vault を移行し、レガシーセキュリティープロパティーを Elytron セキュリティープロパティーに移行する必要があります。

9.2.1. Secure Vault の Secure Credential Storage への移行

JBoss EAP 7.0 のレガシー **security** サブシステムにプレーンテキスト文字列暗号化を保存するために使用される安全な vault は、JBoss EAP 7.1 以降の Elytron と互換性がありません。JBoss EAP 7.1 以降は、クレデンシャルストアを使用して文字列を保存します。クレデンシャルストアは、JBoss EAP 設定ファイル以外のストレージファイル内の認証情報を暗号化します。Elytron によって提供される実装を使用するか、クレデンシャルストア API および SPI を使用して設定をカスタマイズすることができます。各 JBoss EAP サーバーに複数のクレデンシャルストアを含めることができます。



注記

以前に vault 式を使用して機密ではないデータをパラメーター化した場合は、そのデータを Elytron セキュリティープロパティーに置き換える必要があります。Elytron セキュリティープロパティーの詳細は、Elytron セキュリティープロパティー を参照してください。

クレデンシャルストアに関する詳細は、JBoss EAP 7.4 **サーバーセキュリティーの設定方法** の クレデンシャルストア を参照してください。

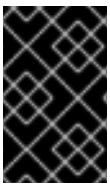
9.2.1.1. WildFly Elytron ツールを使用して vault データを移行する

JBoss EAP に同梱されている WildFly Elytron Tool は、vault の内容をクレデンシャルストアに移行するのに便利な **vault** コマンドを提供します。**vault** コマンドを実行するには、**EAP_HOME/bin** ディレクトリー内の **elytron-tool** スクリプトを実行します。

```
$ EAP_HOME/bin/elytron-tool.sh vault VAULT_ARGUMENTS
```

以下のコマンドを使用すると、使用できるすべての引数の説明を表示できます。

```
$ EAP_HOME/bin/elytron-tool.sh vault --help
```

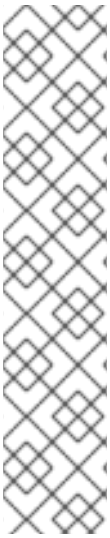


重要

クレデンシャルストアは、パスワードを保護するためにのみ使用されます。管理モデルで使用される vault 式機能はサポートされていません。詳細は、Elytron での暗号化式の作成 を参照してください。

以下の移行オプションの 1 つを選択します。

- 個別のセキュリティー vault のクレデンシャルストアへの移行
- 複数のセキュリティー vault を一括でクレデンシャルストアに移行する



注記

- WildFly Elytron Tool は、セキュリティー vault データファイルの最初のバージョンを処理できません。
- 以下の例のように **--keystore-password** 引数をマスクされた形式で入力し、単一の vault を移行します (またはクリアテキストで移行します)。
- **--salt** および **--iteration** 引数は、マスクされたパスワードを復号化する情報を提供したり、出力にマスクされたパスワードを生成するために提供されます。**--salt** および **--iteration** 引数を省略すると、デフォルトの値が使用されます。
- **--summary** 引数は、変換されたクレデンシャルストアを JBoss EAP 設定に追加するために使用できるフォーマットされた管理 CLI コマンドを生成します。プレーンテキストパスワードはサマリー出力でマスクされます。

9.2.1.1.1. 個別のセキュリティー vault のクレデンシャルストアへの移行

以下の例を使用して、個別のセキュリティー vault をクレデンシャルストアに移行します。

例: 単一のセキュリティー vault のクレデンシャルストアコマンドへの変換

```
$ EAP_HOME/bin/elytron-tool.sh vault --enc-dir vault_data/ --keystore vault-jceks.keystore --
keystore-password MASK-2hKo56F1a3jYGnJwhPmiF5 --iteration 34 --salt 12345678 --alias test --
location cs-v1.store --summary
```

このコマンドはセキュリティー vault をクレデンシャルストアに変換し、変換に使用された管理 CLI コマンドの概要を出力します。

```
Vault (enc-dir="vault_data/";keystore="vault-jceks.keystore") converted to credential store "cs-
v1.store"
Vault Conversion summary:
---------------------------------------
Vault Conversion Successful
CLI command to add new credential store:
/subsystem=elytron/credential-store=test:add(relative-
to=jboss.server.data.dir,create=true,modifiable=true,location="cs-v1.store",implementation-
properties={"keyStoreType"=>"JCEKS"},credential-reference={clear-text="MASK-
2hKo56F1a3jYGnJwhPmiF5;12345678;34"})
```

9.2.1.1.2. 複数のセキュリティー vault を一括でクレデンシャルストアに移行する

--bulk-convert 引数を使用して一括変換記述子ファイルを示し、複数の vault をクレデンシャルストアに変換することができます。

前提条件

ここで使用する例では、以下の一括変換記述子ファイルを使用します。

例: bulk-vault-conversion-descriptor.txt ファイル

```
keystore:vault-v1/vault-jceks.keystore
keystore-password:MASK-2hKo56F1a3jYGnJwhPmiF5
enc-dir:vault-v1/vault_data/
salt:12345678
iteration:34
location:v1-cs-1.store
alias:test

keystore:vault-v1/vault-jceks.keystore
keystore-password:secretsecret
enc-dir:vault-v1/vault_data/
location:v1-cs-2.store
alias:test

# different vault vault-v1-more
keystore:vault-v1-more/vault-jceks.keystore
keystore-password:MASK-2hKo56F1a3jYGnJwhPmiF5
enc-dir:vault-v1-more/vault_data/
salt:12345678
iteration:34
location:v1-cs-more.store
alias:test
```

新しい **keystore:** 行ごとに変換が新たに開始されます。**salt**、**iteration**、および **properties** 以外のオプションはすべて必須です。

手順

1. 一括変換を実行し、管理 CLI コマンドをフォーマットする出力を生成するには、以下のコマンドを実行します。

```
$ EAP_HOME/bin/elytron-tool.sh vault --bulk-convert path/to/bulk-vault-conversion-descriptor.txt --summary
```

このコマンドは、ファイルに指定されたすべてのセキュリティー vault をクレデンシャルストアに変換し、変換に使用された管理 CLI コマンドの概要を出力します。

```
Vault (enc-dir="vault-v1/vault_data/";keystore="vault-v1/vault-jceks.keystore") converted to credential store "v1-cs-1.store"
Vault Conversion summary:
---------------------------------------
Vault Conversion Successful
CLI command to add new credential store:
/subsystem=elytron/credential-store=test:add(relative-to=jboss.server.data.dir,create=true,modifiable=true,location="v1-cs-1.store",implementation-properties={"keyStoreType"=>"JCEKS"},credential-reference={clear-text="MASK-2hKo56F1a3jYGnJwhPmiF5;12345678;34"})
---------------------------------------

Vault (enc-dir="vault-v1/vault_data/";keystore="vault-v1/vault-jceks.keystore") converted to credential store "v1-cs-2.store"
Vault Conversion summary:
---------------------------------------
Vault Conversion Successful
CLI command to add new credential store:
/subsystem=elytron/credential-store=test:add(relative-to=jboss.server.data.dir,create=true,modifiable=true,location="v1-cs-2.store",implementation-properties={"keyStoreType"=>"JCEKS"},credential-reference={clear-text="secretsecret"})
---------------------------------------

Vault (enc-dir="vault-v1-more/vault_data/";keystore="vault-v1-more/vault-jceks.keystore") converted to credential store "v1-cs-more.store"
Vault Conversion summary:
---------------------------------------
Vault Conversion Successful
CLI command to add new credential store:
/subsystem=elytron/credential-store=test:add(relative-to=jboss.server.data.dir,create=true,modifiable=true,location="v1-cs-more.store",implementation-properties={"keyStoreType"=>"JCEKS"},credential-reference={clear-text="MASK-2hKo56F1a3jYGnJwhPmiF5;12345678;34"})
---------------------------------------
```

9.2.2. セキュリティープロパティーの Elytron への移行

以下の例では、**group.name** および **encoding.algorithm** セキュリティープロパティーは、レガシー **security** サブシステムで **security-properties** として定義されていることを前提としています。

security サブシステムで定義されたセキュリティープロパティーは、以下のとおりです。

```
<subsystem xmlns="urn:jboss:domain:security:2.0">
    ...
    <security-properties>
        <property name="group.name" value="engineering-group" />
        <property name="encoding.algorithm" value="BASE64" />
    </security-properties>
</subsystem>
```

elytron サブシステムでこれらのセキュリティープロパティーを定義するには、以下の管理 CLI コマンドを使用して **elytron** サブシステムの **security-properties** 属性を設定します。

```
/subsystem=elytron:write-attribute(name=security-properties, value={ group.name = "engineering-group", encoding.algorithm = "BASE64" })
```

これは、サーバー設定ファイルの **elytron** サブシステムの **security-properties** を定義します。

```
<subsystem xmlns="urn:wildfly:elytron:4.0" final-providers="combined-providers" disallowed-providers="OracleUcrypto">
    <security-properties>
        <security-property name="group.name" value="engineering-group"/>
        <security-property name="encoding.algorithm" value="BASE64"/>
    </security-properties>
    ...
</subsystem>
```

前のコマンドで使用した **write-attribute** 操作は、既存のプロパティーを上書きします。他のセキュリティープロパティーに影響を与えずにセキュリティープロパティーを追加または変更するには、管理 CLI コマンドで **map** 操作を使用します。

```
/subsystem=elytron:map-put(name=security-properties, key=group.name, value=technical-support)
```

同様に、**map-remove** 操作を使用すると特定のセキュリティープロパティーを削除できます。

```
/subsystem=elytron:map-remove(name=security-properties, key=group.name)
```

9.3. 認証設定の移行

このセクションでは、プロパティーベースの認証と認可の Elytron への移行に関する情報を提供します。さらに、Elytron へのキャッシュを使用する LDAP 認証、データベース認証設定、kerberos 認証、複合ストア、JACC セキュリティー、およびセキュリティードメインの移行に関する情報も提供します。

9.3.1. PicketBox プロパティーベースの設定を Elytron に移行

以下の例を使用して PicketBox プロパティーベースの認証を Elytron に移行します。

例: PicketBox プロパティーベースの設定のコマンド

```
/subsystem=security/security-domain=application-security:add
/subsystem=security/security-domain=application-security/authentication=classic:add(login-modules=
[{code=UsersRoles, flag=Required, module-options=
{usersProperties=file://${jboss.server.config.dir}/example-users.properties,
rolesProperties=file://${jboss.server.config.dir}/example-roles.properties}}])
```

-

これにより、サーバーが以下のように設定されます。

例: PicketBox プロパティーベースのセキュリティードメインの設定

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

9.3.1.1. プロパティーベースの認証の Elytron への移行

以下の手順に従って、PicketBox プロパティーベースの認証を Elytron に移行します。

前提条件

移行予定のデプロイされた Web アプリケーションは、フォームベースの認証を必要とするように設定する必要があります。アプリケーションは PicketBox セキュリティードメインを参照し、**UsersRolesLoginModule** を使用して **example-users.properties** および **example-roles.properties** ファイルからユーザー情報をロードします。この手順では、セキュリティードメインが、以下の管理 CLI コマンドを使用して、レガシー **security** サブシステムで定義されていることも前提としています。

PicketBox プロパティーベースの認証が設定された状態で開始していることを確認してください。

手順

1. PicketBox プロパティーファイルを参照する新しいレルムを **elytron** サブシステムに定義します。

```
/subsystem=elytron/properties-realm=application-properties:add(users-properties=
{path=example-users.properties, relative-to=jboss.server.config.dir, plain-text=true, digest-
realm-name="Application Security"}, groups-properties={path=example-roles.properties,
relative-to=jboss.server.config.dir}, groups-attribute=Roles)
```

2. **elytron** サブシステムでセキュリティードメインサブシステムを定義します。

```
/subsystem=elytron/security-domain=application-security:add(realms=[{realm=application-
properties}], default-realm=application-properties, permission-mapper=default-permission-
mapper)
```

これにより、サーバー設定ファイルの **elytron** サブシステム設定が以下のようになります。

```
<subsystem xmlns="urn:wildfly:elytron:4.0" final-providers="combined-providers" disallowed-
providers="OracleUcrypto">
  ...
  <security-domains>
    ...
```

```
    <security-domain name="application-security" default-realm="application-properties"
permission-mapper="default-permission-mapper">
      <realm name="application-properties"/>
    </security-domain>
  </security-domains>
  <security-realms>
    ...
    <properties-realm name="application-properties" groups-attribute="Roles">
      <users-properties path="example-users.properties" relative-to="jboss.server.config.dir"
digest-realm-name="Application Security" plain-text="true"/>
      <groups-properties path="example-roles.properties" relative-to="jboss.server.config.dir"/>
    </properties-realm>
  </security-realms>
  ...
</subsystem>
```

3. デプロイメントによって参照されるアプリケーションセキュリティードメインを、**undertow** サブシステムの新たに定義された HTTP 認証ファクトリーにマップします。

```
/subsystem=undertow/application-security-domain=application-security:add(security-
domain=application-security)
```

これにより、サーバー設定ファイルの **undertow** サブシステム設定が以下のようになります。

```
<subsystem xmlns="urn:jboss:domain:undertow:12.0">
  ...
  <application-security-domains>
    <application-security-domain name="application-security" security-domain="application-
security"/>
  </application-security-domains>
  ...
</subsystem>
```

4. 新しいアプリケーションセキュリティードメインマッピングを有効にするには、サーバーのリロードまたはアプリケーションの再デプロイが必要になります。

これで、認証が PicketBox 設定と同等になるよう設定されました。

9.3.2. レガシーセキュリティーレルムのプロパティーベースの設定を Elytron に移行する

このセクションでは、ユーザー、パスワード、およびグループ情報をプロパティーファイルから JBoss EAP 7.4 以前の Elytron にロードするレガシーセキュリティーレルムを移行する方法について説明します。通常このタイプのレガシーセキュリティーレルムは、管理インターフェイスまたはリモーティングコネクターのいずれかを保護するために使用されます。

JBoss EAP 8.0 では、filesystem-realm は properties-realm よりも優先されます。

前提条件

移行予定のデプロイされた Web アプリケーションは、フォームベースの認証を必要とするように設定する必要があります。アプリケーションは PicketBox セキュリティードメインを参照し、**UsersRolesLoginModule** を使用して **example-users.properties** および **example-**

roles.properties ファイルからユーザー情報をロードします。この手順では、セキュリティードメインが、以下の管理 CLI コマンドを使用して、レガシー **security** サブシステムで定義されていることも前提としています。

例: レガシーセキュリティーレルムコマンド

```
/core-service=management/security-realm=ApplicationSecurity:add
/core-service=management/security-realm=ApplicationSecurity/authentication=properties:add(relative-to=jboss.server.config.dir, path=example-users.properties, plain-text=true)
/core-service=management/security-realm=ApplicationSecurity/authorization=properties:add(relative-to=jboss.server.config.dir, path=example-roles.properties)
```

これにより、サーバーが以下のように設定されます。

例: レガシーセキュリティーレルム設定

```
<security-realm name="ApplicationSecurity">
  <authentication>
    <properties path="example-users.properties" relative-to="jboss.server.config.dir" plain-text="true"/>
  </authentication>
  <authorization>
    <properties path="example-roles.properties" relative-to="jboss.server.config.dir"/>
  </authorization>
</security-realm>
```

Elytron セキュリティーをアプリケーションサーバーに追加する理由の 1 つは、サーバー全体で一貫したセキュリティーソリューションを使用できるようにすることでした。プロパティーベースのレガシーセキュリティーレルムを Elytron に移行する最初のステップは、PicketBox プロパティーベースの認証を Elytron に移行する方法と似ています。以下の手順に従って、プロパティーベースのレガシーセキュリティーレルムを Elytron に移行します。

手順

1. プロパティーファイルを参照する新しいレルムを **elytron** サブシステムに定義します。

```
/subsystem=elytron/properties-realm=application-properties:add(users-properties={path=example-users.properties, relative-to=jboss.server.config.dir, plain-text=true, digest-realm-name="Application Security"}, groups-properties={path=example-roles.properties, relative-to=jboss.server.config.dir}, groups-attribute=Roles)
```

2. **elytron** サブシステムでセキュリティードメインサブシステムを定義します。

```
/subsystem=elytron/security-domain=application-security:add(realms=[{realm=application-properties}], default-realm=application-properties, permission-mapper=default-permission-mapper)
```

これにより、Elytron が以下のように設定されます。

```
<subsystem xmlns="urn:wildfly:elytron:4.0" final-providers="combined-providers" disallowed-providers="OracleUcrypto">
  ...
  <security-domains>
    ...
```

```
    <security-domain name="application-security" default-realm="application-properties"
permission-mapper="default-permission-mapper">
      <realm name="application-properties"/>
    </security-domain>
  </security-domains>
  <security-realms>
    ...
    <properties-realm name="application-properties" groups-attribute="Roles">
      <users-properties path="example-users.properties" relative-to="jboss.server.config.dir"
digest-realm-name="Application Security" plain-text="true"/>
      <groups-properties path="example-roles.properties" relative-to="jboss.server.config.dir"/>
    </properties-realm>
  </security-realms>
    ...
</subsystem>
```

3. レガシーセキュリティーレルムが SASL (Simple Authentication Security Layer) 認証にも使用されるように、**sasl-authentication-factory** を定義します。

```
/subsystem=elytron/sasl-authentication-factory=application-security-sasl:add(sasl-server-
factory=elytron, security-domain=application-security, mechanism-configurations=
[{mechanism-name=PLAIN}])
```

これにより、Elytron が以下のように設定されます。

```
<subsystem xmlns="urn:wildfly:elytron:4.0" final-providers="combined-providers" disallowed-
providers="OracleUcrypto">
  ...
  <sasl>
    ...
    <sasl-authentication-factory name="application-security-sasl" sasl-server-factory="elytron"
security-domain="application-security">
      <mechanism-configuration>
        <mechanism mechanism-name="PLAIN"/>
      </mechanism-configuration>
    </sasl-authentication-factory>
    ...
  </sasl>
</subsystem>
```

4. SASL 認証のリモーティングコネクターを設定し、レガシーセキュリティーレルムとの関連を削除します。

```
/subsystem=remoting/http-connector=http-remoting-connector:write-attribute(name=sasl-
authentication-factory, value=application-security-sasl)
```

これにより、サーバー設定ファイルの **remoting** サブシステムの設定が以下のようになります。

```
<subsystem xmlns="urn:jboss:domain:remoting:4.0">
  ...
  <http-connector name="http-remoting-connector" connector-ref="default" sasl-
authentication-factory="application-security-sasl"/>
</subsystem>
```

5. 2 つの認証ファクトリーを追加して、Elytron との **http-interface** を保護します。

```
/subsystem=elytron/http-authentication-factory=application-security-http:add(http-server-mechanism-factory=global, security-domain=application-security, mechanism-configurations=[{mechanism-name=BASIC}])
/core-service=management/management-interface=http-interface:write-attribute(name=http-upgrade.sasl-authentication-factory, value=application-security-sasl)
```

これにより、設定が以下のようになります。

```
<management-interfaces>
  <http-interface http-authentication-factory="application-security-http">
    <http-upgrade enabled="true" sasl-authentication-factory="application-security-sasl"/>
    <socket-binding http="management-http"/>
  </http-interface>
</management-interfaces>
```

注記

管理インターフェイスを保護する際は、これらの例で使用された名前に置き換える必要があります。

これで、レガシープロパティーベースの設定の Elytron への移行が完了しました。

9.3.3. filesystem-realm コマンドを使用したファイルシステムベースのセキュリティーレルムへの移行

elytron.sh ツールの **filesystem-realm** コマンドを使用して、レガシープロパティーベースのセキュリティーレルムを Elytron のファイルシステムベースのレルムに移行します。

ファイルシステムベースのレルムは、ユーザー ID を保存するための、Elytron で使用されるファイルシステムベースのアイデンティティーストアです。**filesystem-realm** コマンドは、**properties-realm** ファイルを **filesystem-realm** に変換します。また、このレリムとセキュリティードメインを **elytron** サブシステムに追加するためのコマンドも生成します。

手順

1. プロパティーファイルを移行します。
   一度に 1 つずつ user-properties ファイルを移行することも、プロパティーファイルを一括で移行することもできます。以下の例は、両方のタイプの移行の手順を示しています。

   - 単一のプロパティーファイルを移行するには、以下を行います。
     次の例は、関連付けられた roles-properties ファイルとともに単一の users-properties ファイルを **filesystem-realm** 変換します。この例では、レガシーセキュリティードメインに user-properties および role-properties ファイルがあることを前提としています。

     ```
     example-users.properties
     example-roles.properties
     ```

     例: 単一の user-property ファイルの移行

     ```
     $./bin/elytron-tool.sh filesystem-realm --users-file example-users.properties --roles-file example-roles.properties --output-location realms/example
     ```

これにより、filesystem-realm ファイルと管理 CLI コマンドを含むスクリプトが作成されます。このスクリプトは realms/example ディレクトリーに保存されます。

- 複数のプロパティーファイルを移行するには、以下を行います。
  次の例は、関連付けられた roles-properties ファイルとともに users-properties ファイルを一括で **filesystem-realm** に変換します。この例では、レガシーセキュリティードメインに以下のプロパティーファイルがあることを前提としています。

```
users-1.properties
users-2.properties
roles-1.properties
roles-2.properties
```

一括で users-roles ファイルを変換するには、**filesystem-realm** コマンドで使用する記述子ファイルを作成する必要があります。この例では、**/bin** ディレクトリーにある記述子ファイル **example-descriptor-file** が以下の内容で作成されます。

例: 記述子ファイル

```
users-file:/full/path/to/users-1.properties
roles-file:/full/path/to/roles-1.properties
output-location:./realms/bulk-1-example
filesystem-realm-name:exampleFileSystemRealm1
security-domain-name:exampleSecurityDomain1

users-file:/full/path/to/users-2.properties
roles-file:/full/path/to/roles-2.properties
output-location:./realms/bulk-2-example
filesystem-realm-name:exampleFileSystemRealm2
security-domain-name:exampleSecurityDomain2
```

記述子ファイルの空白行は、各 users-properties ファイルの操作を分けするために使用されます。

次の例では、記述子ファイルを使用して、割り当てられた roles-properties ファイルとともに、2 つのユーザープロパティーファイルを **filesystem-realm** に変換します。

例: 一括移行

```
$./bin/elytron-tool.sh filesystem-realm --bulk-convert example-descriptor-file
```

これにより、**filesystem-realm** ファイルおよび管理 CLI コマンドが含まれるスクリプトが作成されます。このスクリプトは、記述子ファイルの **output-location** 属性で指定されたディレクトリーに保存されます。

2. Elytron ツールによって生成された CLI コマンドを使用して、新しいセキュリティーレルムとセキュリティードメインを **elytron** サブシステムに追加します。

例: filesystem-realm の追加

```
/subsystem=elytron/filesystem-realm=converted-properties-filesystem-
realm:add(path=/full/path/to/realms/example)
```

```
/subsystem=elytron/security-domain=converted-properties-security-domain:add(realms=
[{realm=converted-properties-filesystem-realm}],default-realm=converted-properties-
filesystem-realm,permission-mapper=default-permission-mapper)
```

9.3.4. LDAP 認証設定の Elytron への移行

情報をアイデンティティー属性として管理できるように、レガシー LDAP 認証を Elytron に移行します。

前提条件

レガシー LDAP 認証を Elytron に移行する前に、プロパティーベースの認証と認可の Elytron への移行セクションの内容を読む必要があります。この内容はここにも当てはまります。セキュリティードメインと認証ファクトリーを定義する方法、およびそれらを認証に使用するようにマップする方法に重点を置く必要があります。このセクションではこれらの手順を繰り返しませんので、続行する前に必ずこのセクションを読んでください。

ここで使用する例は、グループまたはロール情報は直接 LDAP からロードされ、レガシー LDAP 認証は以下のとおり設定されることを仮定しています。

手順

1. LDAP サーバーには以下のユーザーおよびグループエントリーが含まれます。

   例: LDAP サーバーユーザーエントリー

   ```
   dn: uid=TestUserOne,ou=users,dc=group-to-principal,dc=wildfly,dc=org
   objectClass: top
   objectClass: inetOrgPerson
   objectClass: uidObject
   objectClass: person
   objectClass: organizationalPerson
   cn: Test User One
   sn: Test User One
   uid: TestUserOne
   userPassword: {SSHA}UG8ov2rnrnBKakcARVvraZHqTa7mFWJZlWt2HA==
   ```

   例: LDAP サーバーグループエントリー

   ```
   dn: uid=GroupOne,ou=groups,dc=group-to-principal,dc=wildfly,dc=org
   objectClass: top
   objectClass: groupOfUniqueNames
   objectClass: uidObject
   cn: Group One
   uid: GroupOne
   uniqueMember: uid=TestUserOne,ou=users,dc=group-to-principal,dc=wildfly,dc=org
   ```

   ユーザー名は認証の目的で **uid** 属性と照合され、グループ名はグループエントリーの **uid** 属性から取られます。

2. LDAP サーバーへの接続と関連するセキュリティーレルムは、以下の管理 CLI コマンドを使用して定義されます。

   例: LDAP セキュリティーレルム設定コマンド

   ```
   ```

```
batch
/core-service=management/ldap-
connection=MyLdapConnection:add(url="ldap://localhost:10389", search-
dn="uid=admin,ou=system", search-credential="secret")

/core-service=management/security-realm=LDAPRealm:add
/core-service=management/security-
realm=LDAPRealm/authentication=ldap:add(connection="MyLdapConnection", username-
attribute=uid, base-dn="ou=users,dc=group-to-principal,dc=wildfly,dc=org")

/core-service=management/security-
realm=LDAPRealm/authorization=ldap:add(connection=MyLdapConnection)
/core-service=management/security-realm=LDAPRealm/authorization=ldap/username-to-
dn=username-filter:add(attribute=uid, base-dn="ou=users,dc=group-to-
principal,dc=wildfly,dc=org")
/core-service=management/security-realm=LDAPRealm/authorization=ldap/group-
search=group-to-principal:add(base-dn="ou=groups,dc=group-to-
principal,dc=wildfly,dc=org", iterative=true, prefer-original-connection=true, principal-
attribute=uniqueMember, search-by=DISTINGUISHED_NAME, group-name=SIMPLE,
group-name-attribute=uid)
run-batch
```

これにより、サーバーが以下のように設定されます。

例: LDAP セキュリティーレルム設定

```
<management>
  <security-realms>
    ...
    <security-realm name="LDAPRealm">
      <authentication>
        <ldap connection="MyLdapConnection" base-dn="ou=users,dc=group-to-
principal,dc=wildfly,dc=org">
          <username-filter attribute="uid"/>
        </ldap>
      </authentication>
      <authorization>
        <ldap connection="MyLdapConnection">
          <username-to-dn>
            <username-filter base-dn="ou=users,dc=group-to-principal,dc=wildfly,dc=org"
attribute="uid"/>
          </username-to-dn>
          <group-search group-name="SIMPLE" iterative="true" group-name-attribute="uid">
            <group-to-principal search-by="DISTINGUISHED_NAME" base-
dn="ou=groups,dc=group-to-principal,dc=wildfly,dc=org" prefer-original-connection="true">
              <membership-filter principal-attribute="uniqueMember"/>
            </group-to-principal>
          </group-search>
        </ldap>
      </authorization>
    </security-realm>
  </security-realms>
  <outbound-connections>
    <ldap name="MyLdapConnection" url="ldap://localhost:10389" search-
dn="uid=admin,ou=system" search-credential="secret"/>
```

```
    </outbound-connections>
    ...
</management>
```

3. 以下の管理 CLI コマンドは、**LdapExtLoginModule** を使用してユーザー名とパスワードを検証する PicketBox セキュリティードメインの設定に使用されます。

例: セキュリティードメイン設定コマンド

```
/subsystem=security/security-domain=application-security:add
/subsystem=security/security-domain=application-security/authentication=classic:add(login-modules=[{code=LdapExtended, flag=Required, module-options={
java.naming.factory.initial=com.sun.jndi.ldap.LdapCtxFactory,
java.naming.provider.url=ldap://localhost:10389, java.naming.security.authentication=simple,
bindDN="uid=admin,ou=system", bindCredential=secret, baseCtxDN="ou=users,dc=group-to-principal,dc=wildfly,dc=org", baseFilter="(uid={0})", rolesCtxDN="ou=groups,dc=group-to-principal,dc=wildfly,dc=org", roleFilter="(uniqueMember={1})", roleAttributeID="uid" }}])
```

これにより、サーバーが以下のように設定されます。

例: セキュリティードメイン設定

```
<subsystem xmlns="urn:jboss:domain:security:2.0">
  ...
  <security-domains>
    ...
    <security-domain name="application-security">
      <authentication>
        <login-module code="LdapExtended" flag="required">
          <module-option name="java.naming.factory.initial"
value="com.sun.jndi.ldap.LdapCtxFactory"/>
          <module-option name="java.naming.provider.url" value="ldap://localhost:10389"/>
          <module-option name="java.naming.security.authentication" value="simple"/>
          <module-option name="bindDN" value="uid=admin,ou=system"/>
          <module-option name="bindCredential" value="secret"/>
          <module-option name="baseCtxDN" value="ou=users,dc=group-to-
principal,dc=wildfly,dc=org"/>
          <module-option name="baseFilter" value="(uid={0})"/>
          <module-option name="rolesCtxDN" value="ou=groups,dc=group-to-
principal,dc=wildfly,dc=org"/>
          <module-option name="roleFilter" value="(uniqueMember={1})"/>
          <module-option name="roleAttributeID" value="uid"/>
        </login-module>
      </authentication>
    </security-domain>
  </security-domains>
</subsystem>
```

9.3.4.1. レガシー LDAP 認証の Elytron への移行

以下の手順に従って、以前の LDAP 認証の設定例を JBoss EAP 7.4 以前の Elytron に移行します。この項は、レガシーセキュリティー LDAP レルム の移行と PicketBox LDAP セキュリティードメイン の移行が対象になります。

手順

1. LDAP への接続を **elytron** サブシステムに定義します。

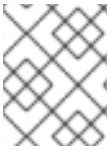
```
/subsystem=elytron/dir-context=ldap-connection:add(url=ldap://localhost:10389,
principal="uid=admin, ou=system", credential-reference={clear-text=secret})
```

2. セキュリティーレルムを作成し、LDAP の検索およびパスワードの検証を行います。

```
/subsystem=elytron/ldap-realm=ldap-realm:add(dir-context=ldap-connection, direct-
verification=true, identity-mapping={search-base-dn="ou=users, dc=group-to-principal,
dc=wildfly, dc=org", rdn-identifier="uid", attribute-mapping=[{filter-base-dn="ou=groups,
dc=group-to-principal, dc=wildfly, dc=org", filter="(uniqueMember={1})", from="uid",
to="Roles"}]})
```

これまでの手順によって、サーバー設定ファイルで **elytron** サブシステムが次のように設定されます。

```
<subsystem xmlns="urn:wildfly:elytron:4.0" final-providers="combined-providers" disallowed-
providers="OracleUcrypto">
  ...
  <security-realms>
    ...
    <ldap-realm name="ldap-realm" dir-context="ldap-connection" direct-verification="true">
      <identity-mapping rdn-identifier="uid" search-base-dn="ou=users,dc=group-to-
principal,dc=wildfly,dc=org">
        <attribute-mapping>
          <attribute from="uid" to="Roles" filter="(uniqueMember={1})" filter-base-
dn="ou=groups,dc=group-to-principal,dc=wildfly,dc=org"/>
        </attribute-mapping>
      </identity-mapping>
    </ldap-realm>
  </security-realms>
  ...
  <dir-contexts>
    <dir-context name="ldap-connection" url="ldap://localhost:10389"
principal="uid=admin,ou=system">
      <credential-reference clear-text="secret"/>
    </dir-context>
  </dir-contexts>
</subsystem>
```



注記

デフォルトでは、**security-domain** に対して定義された **role-decoder** がない場合は "Roles" アイデンティティー属性がアイデンティティーロールにマップされます。

これで、LDAP からロードされた情報を属性としてアイデンティティーに関連付けることができるようになりました。これらの属性をロールにマップすることができますが、別の目的でロードおよび使用することもできます。新たに作成されたセキュリティーレルムは、このガイドの プロパティーベースの認証と認可の Elytron への移行 に記載されている方法と同様にセキュリティードメインで使用できます。

9.3.5. データベース認証設定の Elytron への移行

JDBC データソースベースの PicketBox 認証を Elytron に移行します。セキュリティードメイン、認証ファクトリーの定義、それらを認証用にマッピングする手順については、プロパティーベースの認証および認可の Elytron への移行 を参照してください。

ここで使用する例は、以下の例に似た構文を使用して作成されたデータベーステーブルにユーザー認証情報が格納されることを仮定しています。

例: データベースユーザーテーブルを作成するための構文

```
CREATE TABLE User (
    id BIGINT NOT NULL,
    username VARCHAR(255),
    password VARCHAR(255),
    role ENUM('admin', 'manager', 'user'),
    PRIMARY KEY (id),
    UNIQUE (username)
)
```

認証の目的で、ユーザー名は **username** 列に格納されたデータと照合され、パスワードは 16 進エンコードされた MD5 ハッシュとして **password** 列に格納されることが想定されます。 さらに、承認目的のユーザーロールは **role** 列に格納されます。

PicketBox セキュリティードメインは、データベーステーブルからデータを取得するために JBDC データソースを使用します。PicketBox セキュリティードメインは以下の管理 CLI コマンドを使用して設定されることを仮定します。

例: PicketBox データベース LoginModule 設定コマンド

```
/subsystem=security/security-domain=application-security:add
/subsystem=security/security-domain=application-security/authentication=classic:add( login-
modules=[ { code=Database, flag=Required, module-options={
dsJndiName="java:jboss/datasources/ExampleDS", principalsQuery="SELECT password FROM
User WHERE username = ?", rolesQuery="SELECT role, 'Roles' FROM User WHERE username =
?", hashAlgorithm=MD5, hashEncoding=base64 } } ] )
```

これにより、レガシー **security** サブシステムの **login-module** 設定が次のようになります。

例: PicketBox LoginModule 設定

```
<subsystem xmlns="urn:jboss:domain:security:2.0">
  <security-domains>
    ...
    <security-domain name="application-security">
      <authentication>
        <login-module code="Database" flag="required">
          <module-option name="dsJndiName" value="java:jboss/datasources/ExampleDS"/>
          <module-option name="principalsQuery" value="SELECT password FROM User WHERE
username = ?"/>
          <module-option name="rolesQuery" value="SELECT role, 'Roles' FROM User WHERE
username = ?"/>
          <module-option name="hashAlgorithm" value="MD5"/>
          <module-option name="hashEncoding" value="base64"/>
        </login-module>
      </authentication>
```

```
    </security-domain>
  </security-domains>
</subsystem>
```

9.3.5.1. レガシーデータベース認証の Elytron への移行

JBoss EAP 7.4 以前のリリースでは、JDBC レルムを定義して Elytron による JDBC データソースアクセスを有効にし、以前のデータベース認証の設定例を Elytron に移行する必要があります。

手順

1. 以下の管理コマンドを使用して **jdbc-realm** を定義します。

```
/subsystem=elytron/jdbc-realm=jdbc-realm:add(principal-query=[ { data-source=ExampleDS,
sql="SELECT role, password FROM User WHERE username = ?", attribute-mapping=[{index=1,
to=Roles } ] simple-digest-mapper={algorithm=simple-digest-md5, password-index=2} } ] )
```

これにより、サーバー設定ファイルの **elytron** サブシステムの **jdbc-realm** 設定が以下のようになります。

```
<subsystem xmlns="urn:wildfly:elytron:4.0" final-providers="combined-providers" disallowed-
providers="OracleUcrypto">
  ...
  <security-realms>
    ...
    <jdbc-realm name="jdbc-realm">
      <principal-query sql="SELECT role, password FROM User WHERE username = ?" data-
source="ExampleDS">
        <attribute-mapping>
          <attribute to="Roles" index="1"/>
        </attribute-mapping>
        <simple-digest-mapper password-index="2"/>
      </principal-query>
    </jdbc-realm>
    ...
  </security-realms>
  ...
</subsystem>
```

これで、Elytron は JDBC レルム設定を使用してデータベース認証を管理するようになります。Elytron は 1 つの SQL クエリーを使用してユーザー属性とクレデンシャルをすべて取得し、SQL の結果からデータを抽出して認証に使用する属性のマッピングを作成するため、Elytron は PicketBox よりも効率がよくなります。

9.3.6. Kerberos 認証の Elytron への移行

Kerberos 設定を使用する場合、JBoss EAP サーバーは環境からの設定情報に依存でき、システムプロパティーを使用してキー設定を指定することも可能です。

これらのシステムプロパティーは、レガシー設定と移行された Elytron 設定の両方に適用されます。

例: Kerberos システムプロパティーの管理 CLI コマンド

```
# Enable debugging
```

```
/system-property=sun.security.krb5.debug:add(value=true)
# Identify the Kerberos realm to use
/system-property=java.security.krb5.realm:add(value=ELYTRON.ORG)
# Identify the address of the KDC
/system-property=java.security.krb5.kdc:add(value=kdc.elytron.org)
```

例: Kerberos システムプロパティーのサーバー設定

```
<system-properties>
  <property name="sun.security.krb5.debug" value="true"/>
  <property name="java.security.krb5.realm" value="ELYTRON.ORG"/>
  <property name="java.security.krb5.kdc" value="kdc.elytron.org"/>
</system-properties>
```

認証メカニズムに応じて、以下の移行オプションのいずれかを選択します。

- Kerberos HTTP 認証の移行
- Kerberos リモーティング SASL 認証

9.3.6.1. Kerberos HTTP 認証の移行

レガシーのセキュリティー設定では、セキュリティーレルムを定義して以下のように HTTP 管理インターフェイスの SPNEGO 認証を有効にできます。

前提条件

以下の例では、システムプロパティーを使用して Kerberos が設定されていることを前提としています。

例: HTTP 管理インターフェイスの SPNEGO 認証の有効化

```
/core-service=management/security-realm=Kerberos:add
/core-service=management/security-realm=Kerberos/server-identity=kerberos:add
/core-service=management/security-realm=Kerberos/server-identity=kerberos/keytab=HTTP\/test-server.elytron.org@ELYTRON.ORG:add(path=/path/to/test-server.keytab, debug=true)
/core-service=management/security-realm=Kerberos/authentication=kerberos:add(remove-realm=true)
```

例: Kerberos セキュリティーレルム設定

```
<security-realms>
  ...
  <security-realm name="Kerberos">
    <server-identities>
      <kerberos>
        <keytab principal="HTTP/test-server.elytron.org@ELYTRON.ORG" path="/path/to/test-server.keytab" debug="true"/>
      </kerberos>
    </server-identities>
    <authentication>
      <kerberos remove-realm="true"/>
    </authentication>
  </security-realm>
</security-realms>
```

また、2 つのレガシーセキュリティードメインを定義して、アプリケーションが Kerberos HTTP 認証を使用できるようにすることも可能です。

例: 複数のセキュリティードメインの定義

```
# Define the first security domain
/subsystem=security/security-domain=host:add
/subsystem=security/security-domain=host/authentication=classic:add
/subsystem=security/security-domain=host/authentication=classic/login-
module=1:add(code=Kerberos, flag=Required, module-options={storeKey=true, useKeyTab=true,
principal=HTTP/test-server.elytron.org@ELYTRON.ORG, keyTab=path/to/test-server.keytab,
debug=true}

# Define the second SPNEGO security domain
/subsystem=security/security-domain=SPNEGO:add
/subsystem=security/security-domain=SPNEGO/authentication=classic:add
/subsystem=security/security-domain=SPNEGO/authentication=classic/login-
module=1:add(code=SPNEGO, flag=requisite,  module-options={password-stacking=useFirstPass,
serverSecurityDomain=host})
/subsystem=security/security-domain=SPNEGO/authentication=classic/login-module=1:write-
attribute(name=module, value=org.jboss.security.negotiation)
/subsystem=security/security-domain=SPNEGO/authentication=classic/login-
module=2:add(code=UsersRoles, flag=required, module-options={password-stacking=useFirstPass,
usersProperties=  /path/to/kerberos/spnego-users.properties, rolesProperties=
/path/to/kerberos/spnego-roles.properties, defaultUsersProperties=  /path/to/kerberos/spnego-
users.properties, defaultRolesProperties=  /path/to/kerberos/spnego-roles.properties})
```

例: 2 つのセキュリティードメインを使用した設定

```
<subsystem xmlns="urn:jboss:domain:security:2.0">
  <security-domains>
    ...
    <security-domain name="host">
      <authentication>
        <login-module name="1" code="Kerberos" flag="required">
          <module-option name="storeKey" value="true"/>
          <module-option name="useKeyTab" value="true"/>
          <module-option name="principal" value="HTTP/test-server.elytron.org@ELYTRON.ORG"/>
          <module-option name="keyTab" value="/path/to/test-server.keytab"/>
          <module-option name="debug" value="true"/>
        </login-module>
      </authentication>
    </security-domain>
    <security-domain name="SPNEGO">
      <authentication>
        <login-module name="1" code="SPNEGO" flag="requisite"
module="org.jboss.security.negotiation">
          <module-option name="password-stacking" value="useFirstPass"/>
          <module-option name="serverSecurityDomain" value="host"/>
        </login-module>
        <login-module name="2" code="UsersRoles" flag="required">
          <module-option name="password-stacking" value="useFirstPass"/>
          <module-option name="usersProperties" value="path/to/kerberos/spnego-users.properties"/>
          <module-option name="rolesProperties" value="  /path/to/kerberos/spnego-roles.properties"/>
```

```
        <module-option name="defaultUsersProperties" value="  /path/to/kerberos/spnego-
users.properties"/>
        <module-option name="defaultRolesProperties" value="  /path/to/kerberos/spnego-
roles.properties"/>
      </login-module>
    </authentication>
  </security-domain>
 </security-domains>
</subsystem>
```

セキュリティーアプリケーションは、SPNEGO セキュリティードメインを参照してデプロイされ、SPNEGO によってセキュア化されます。

9.3.6.1.1. Kerberos HTTP 認証の Elytron への移行

セキュリティーレルムと Kerberos セキュリティーファクトリーを使用して、Elytron の管理インターフェイスおよびアプリケーションを保護します。

前提条件

以下の例では、システムプロパティーを使用して Kerberos が設定されていることを前提としています。

手順

1. アイデンティティー情報のロードに使用するセキュリティーレルムを定義します。

```
/subsystem=elytron/properties-realm=spnego-properties:add(users-properties=
{path=path/to/spnego-users.properties, plain-text=true, digest-realm-
name=ELYTRON.ORG}, groups-properties={path=path/to/spnego-roles.properties})
```

2. サーバーが独自の Kerberos アイデンティティーをロードできるようにする Kerberos セキュリティーファクトリーを定義します。

```
/subsystem=elytron/kerberos-security-factory=test-server:add(path=path/to/test-
server.keytab, principal=HTTP/test-server.elytron.org@ELYTRON.ORG, debug=true)
```

3. ポリシーと認証ポリシーの HTTP 認証ファクトリーを一緒にプルするためにセキュリティードメインを定義します。

```
/subsystem=elytron/security-domain=SPNEGODomain:add(default-realm=spnego-
properties, realms=[{realm=spnego-properties, role-decoder=groups-to-roles}], permission-
mapper=default-permission-mapper)
/subsystem=elytron/http-authentication-factory=spnego-http-authentication:add(security-
domain=SPNEGODomain, http-server-mechanism-factory=global,mechanism-
configurations=[{mechanism-name=SPNEGO, credential-security-factory=test-server}])
```

これにより、サーバー設定ファイルの **elytron** サブシステムの設定が以下のようになります。

例: 移行された Elytron 設定

```
<subsystem xmlns="urn:wildfly:elytron:4.0" final-providers="combined-providers" disallowed-
providers="OracleUcrypto">
  ...
```

```
  <security-domains>
  ...
    <security-domain name="SPNEGODomain" default-realm="spnego-properties"
permission-mapper="default-permission-mapper">
      <realm name="spnego-properties" role-decoder="groups-to-roles"/>
    </security-domain>
  </security-domains>
  <security-realms>
    ...
    <properties-realm name="spnego-properties">
      <users-properties path="path/to/spnego-users.properties" digest-realm-
name="ELYTRON.ORG" plain-text="true"/>
      <groups-properties path="path/to/spnego-roles.properties"/>
    </properties-realm>
  </security-realms>
  <credential-security-factories>
    <kerberos-security-factory name="test-server" principal="HTTP/test-
server.elytron.org@ELYTRON.ORG" path="path/to/test-server.keytab" debug="true"/>
  </credential-security-factories>
  ...
  <http>
    ...
    <http-authentication-factory name="spnego-http-authentication" http-server-mechanism-
factory="global" security-domain="SPNEGODomain">
      <mechanism-configuration>
        <mechanism mechanism-name="SPNEGO" credential-security-factory="test-server"/>
      </mechanism-configuration>
    </http-authentication-factory>
    ...
  </http>
  ...
</subsystem>
```

4. アプリケーションをセキュアにするには、**undertow** サブシステムのアプリケーションセキュリティードメインを定義し、セキュリティードメインをこの **http-authentication-factory** にマップします。この設定に定義された **http-authentication-factory** を参照するように、HTTP 管理インターフェイスを更新することができます。このプロセスの詳細は、プロパティーベースの認証と認可を Elytron に移行する に記載されています。

9.3.6.2. Kerberos リモーティング SASL 認証の移行

以下の情報を使用して、Kerberos リモーティング SASL 認証を移行します。

手順

1. ネイティブ管理インターフェイスなど、リモート認証に使用される Kerberos/GSSAPI SASL 認証のレガシーセキュリティーレルムを定義します。

例: リモーティング管理 CLI コマンドの Kerberos 認証

```
/core-service=management/security-realm=Kerberos:add
/core-service=management/security-realm=Kerberos/server-identity=kerberos:add
/core-service=management/security-realm=Kerberos/server-identity=kerberos/keytab=remote\/test-
```

```
server.elytron.org@ELYTRON.ORG:add(path=path/to/remote-test-server.keytab, debug=true)
/core-service=management/security-realm=Kerberos/authentication=kerberos:add(remove-
realm=true)
```

例: Kerberos リモーティングセキュリティーレルム設定

```
<management>
  <security-realms>
    ...
    <security-realm name="Kerberos">
      <server-identities>
        <kerberos>
          <keytab principal="remote/test-server.elytron.org@ELYTRON.ORG" path="path/to/remote-test-
server.keytab" debug="true"/>
        </kerberos>
      </server-identities>
      <authentication>
        <kerberos remove-realm="true"/>
      </authentication>
    </security-realm>
  </security-realms>
  ...
</management>
```

9.3.6.2.1. Kerberos リモーティング SASL 認証の Elytron への移行

以下の手順で、Kerberos リモーティング SASL 認証を Elytron に移行します。

手順

1. アイデンティティー情報のロードに使用するセキュリティーレルムを定義します。

```
/path=kerberos:add(relative-to=user.home, path=src/kerberos)
/subsystem=elytron/properties-realm=kerberos-properties:add(users-properties=
{path=kerberos-users.properties, relative-to=kerberos, digest-realm-name=ELYTRON.ORG},
groups-properties={path=kerberos-groups.properties, relative-to=kerberos})
```

2. サーバーのアイデンティティーの Kerberos セキュリティーファクトリーを定義します。

```
/subsystem=elytron/kerberos-security-factory=test-server:add(relative-to=kerberos,
path=remote-test-server.keytab, principal=remote/test-server.elytron.org@ELYTRON.ORG)
```

3. セキュリティードメインと SASL 認証ファクトリーを定義します。

```
/subsystem=elytron/security-domain=KerberosDomain:add(default-realm=kerberos-
properties, realms=[{realm=kerberos-properties, role-decoder=groups-to-roles}], permission-
mapper=default-permission-mapper)
/subsystem=elytron/sasl-authentication-factory=gssapi-authentication-factory:add(security-
domain=KerberosDomain, sasl-server-factory=elytron, mechanism-configurations=
[{mechanism-name=GSSAPI, credential-security-factory=test-server}])
```

これにより、サーバー設定ファイルの **elytron** サブシステムの設定が以下のようになります。

```
<subsystem xmlns="urn:wildfly:elytron:4.0" final-providers="combined-providers" disallowed-
providers="OracleUcrypto">
  ...
  <security-domains>
    ...
    <security-domain name="KerberosDomain" default-realm="kerberos-properties" permission-
mapper="default-permission-mapper">
      <realm name="kerberos-properties" role-decoder="groups-to-roles"/>
    </security-domain>
  </security-domains>
  <security-realms>
   ...
    <properties-realm name="kerberos-properties">
      <users-properties path="kerberos-users.properties" relative-to="kerberos" digest-realm-
name="ELYTRON.ORG"/>
      <groups-properties path="kerberos-groups.properties" relative-to="kerberos"/>
    </properties-realm>
  </security-realms>
  <credential-security-factories>
    <kerberos-security-factory name="test-server" principal="remote/test-
server.elytron.org@ELYTRON.ORG" path="remote-test-server.keytab" relative-to="kerberos"/>
  </credential-security-factories>
  ...
  <sasl>
    ...
    <sasl-authentication-factory name="gssapi-authentication-factory" sasl-server-factory="elytron"
security-domain="KerberosDomain">
      <mechanism-configuration>
        <mechanism mechanism-name="GSSAPI" credential-security-factory="test-server"/>
      </mechanism-configuration>
    </sasl-authentication-factory>
    ...
  </sasl>
 </subsystem>
```

検証

これで、管理インターフェイスまたはリモーティングコネクターが更新され、SASL 認証ファクトリーを参照するようになりました。

ここで定義した 2 つの Elytron サンプルを組み合わせて、共有セキュリティードメインとセキュリティーレルムを使用し、それぞれが独自の Kerberos セキュリティーファクトリーを参照する **protocol-specific authentication** ファクトリーを使用することもできます。

関連情報

同等の Elytron 設定を定義する手順は、Kerberos HTTP 認証の移行 の手順と非常に似ています。

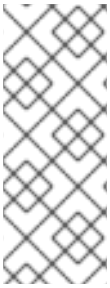
9.3.7. 複合ストアの Elytron への移行

このセクションでは、複数のアイデンティティーストアを使用する PicketBox または レガシーセキュリティーレルム 設定を Elytron Aggregate Security Realm 設定 に移行する方法について説明します。

PicketBox またはレガシーセキュリティーレルムを使用する場合、承認に使用される情報を 1 つのアイデンティティーストアからロードしながら別のアイデンティティーストアに対して認証を実行する設定

を定義することが可能です。Elytron に移行する場合、集約セキュリティーレルムを使用してこれを実現できます。

以下の例は、**example-users.properties** プロパティーファイルを使用してユーザー認証を実行し、LDAP をクエリーしてグループおよびロール情報をロードします。



注記

ここで提示されている設定は、追加の背景情報を提供する以下のセクションの例を基にしています。

- PicketBox プロパティーベースの設定を Elytron に移行
- LDAP 認証設定の Elytron への移行

9.3.7.1. PicketBox 複合ストア設定

この場合の PicketBox セキュリティードメインは、以下の管理 CLI コマンドを使用して設定されます。

例: PicketBox 設定コマンド

```
/subsystem=security/security-domain=application-security:add

/subsystem=security/security-domain=application-security/authentication=classic:add(login-modules=
[ {code=UsersRoles, flag=Required, module-options={ password-stacking=useFirstPass,
usersProperties=file://${jboss.server.config.dir}/example-users.properties}} {code=LdapExtended,
flag=Required, module-options={ password-stacking=useFirstPass,
java.naming.factory.initial=com.sun.jndi.ldap.LdapCtxFactory,
java.naming.provider.url=ldap://localhost:10389, java.naming.security.authentication=simple,
bindDN="uid=admin,ou=system", bindCredential=secret, baseCtxDN="ou=users,dc=group-to-
principal,dc=wildfly,dc=org", baseFilter="(uid={0})", rolesCtxDN="ou=groups,dc=group-to-
principal,dc=wildfly,dc=org",roleFilter="(uniqueMember={1})", roleAttributeID="uid" }}])
```

これにより、サーバーが以下のように設定されます。

例: PicketBox セキュリティードメイン設定

```
<security-domain name="application-security">
  <authentication>
    <login-module code="UsersRoles" flag="required">
      <module-option name="password-stacking" value="useFirstPass"/>
      <module-option name="usersProperties" value="file://${jboss.server.config.dir}/example-
users.properties"/>
    </login-module>
    <login-module code="LdapExtended" flag="required">
      <module-option name="password-stacking" value="useFirstPass"/>
      <module-option name="java.naming.factory.initial" value="com.sun.jndi.ldap.LdapCtxFactory"/>
      <module-option name="java.naming.provider.url" value="ldap://localhost:10389"/>
      <module-option name="java.naming.security.authentication" value="simple"/>
      <module-option name="bindDN" value="uid=admin,ou=system"/>
      <module-option name="bindCredential" value="secret"/>
      <module-option name="baseCtxDN" value="ou=users,dc=group-to-principal,dc=wildfly,dc=org"/>
      <module-option name="baseFilter" value="(uid={0})"/>
      <module-option name="rolesCtxDN" value="ou=groups,dc=group-to-principal,dc=wildfly,dc=org"/>
      <module-option name="roleFilter" value="(uniqueMember={1})"/>
```

```
      <module-option name="roleAttributeID" value="uid"/>
    </login-module>
  </authentication>
</security-domain>
```

詳細は、Elytron 集約セキュリティーレルム設定 で集約セキュリティーレルムの設定方法を参照してください。

9.3.7.2. レガシーセキュリティーレルム複合ストア設定

JBoss EAP 7.4 以前のリリースの場合、レガシーセキュリティーレルム設定は以下の管理 CLI コマンドを使用して設定されます。

注記

レガシーセキュリティーコマンドは JBoss EAP 8 では適用できないため、これらのコマンドは JBoss EAP 7.4 以前のリリースでのみ適用できます。

例: レガシーセキュリティーレルム設定コマンド

```
/core-service=management/ldap-connection=MyLdapConnection:add(url="ldap://localhost:10389",
search-dn="uid=admin,ou=system", search-credential="secret")

/core-service=management/security-realm=ApplicationSecurity:add
/core-service=management/security-
realm=ApplicationSecurity/authentication=properties:add(path=example-users.properties, relative-
to=jboss.server.config.dir, plain-text=true)

batch
/core-service=management/security-
realm=ApplicationSecurity/authorization=ldap:add(connection=MyLdapConnection)
/core-service=management/security-realm=ApplicationSecurity/authorization=ldap/username-to-
dn=username-filter:add(attribute=uid, base-dn="ou=users,dc=group-to-principal,dc=wildfly,dc=org")
/core-service=management/security-realm=ApplicationSecurity/authorization=ldap/group-
search=group-to-principal:add(base-dn="ou=groups,dc=group-to-principal,dc=wildfly,dc=org",
iterative=true, prefer-original-connection=true, principal-attribute=uniqueMember, search-
by=DISTINGUISHED_NAME, group-name=SIMPLE, group-name-attribute=uid)
run-batch
```

これにより、サーバーが以下のように設定されます。

例: レガシーセキュリティーレルム設定

```
<security-realms>
  ...
  <security-realm name="ApplicationSecurity">
    <authentication>
      <properties path="example-users.properties" relative-to="jboss.server.config.dir" plain-
text="true"/>
    </authentication>
    <authorization>
      <ldap connection="MyLdapConnection">
        <username-to-dn>
          <username-filter base-dn="ou=users,dc=group-to-principal,dc=wildfly,dc=org" attribute="uid"/>
```

```
        </username-to-dn>
        <group-search group-name="SIMPLE" iterative="true" group-name-attribute="uid">
          <group-to-principal search-by="DISTINGUISHED_NAME" base-dn="ou=groups,dc=group-to-
principal,dc=wildfly,dc=org" prefer-original-connection="true">
            <membership-filter principal-attribute="uniqueMember"/>
          </group-to-principal>
        </group-search>
      </ldap>
    </authorization>
  </security-realm>
</security-realms>
<outbound-connections>
  <ldap name="MyLdapConnection" url="ldap://localhost:10389" search-dn="uid=admin,ou=system"
search-credential="secret"/>
</outbound-connections>
```

elytron サブシステムで集約セキュリティーレルムを設定してこれを実現する方法については、**Elytron 集約セキュリティーレルム設定** を参照してください。

9.3.7.3. Elytron 集約セキュリティーレルム設定

この場合の同等の Elytron 設定は、以下の管理 CLI コマンドを使用して設定されます。

例: Elytron 設定コマンド

```
/subsystem=elytron/dir-context=ldap-connection:add(url=ldap://localhost:10389,
principal="uid=admin,ou=system", credential-reference={clear-text=secret})

/subsystem=elytron/ldap-realm=ldap-realm:add(dir-context=ldap-connection, direct-verification=true,
identity-mapping={search-base-dn="ou=users,dc=group-to-principal,dc=wildfly,dc=org", rdn-
identifier="uid", attribute-mapping=[{filter-base-dn="ou=groups,dc=group-to-
principal,dc=wildfly,dc=org",filter="(uniqueMember={1})",from="uid",to="Roles"}]})

/subsystem=elytron/properties-realm=application-properties:add(users-properties={path=example-
users.properties, relative-to=jboss.server.config.dir, plain-text=true, digest-realm-name="Application
Security"})

/subsystem=elytron/aggregate-realm=combined-realm:add(authentication-realm=application-
properties, authorization-realm=ldap-realm)

/subsystem=elytron/security-domain=application-security:add(realms=[{realm=combined-realm}],
default-realm=combined-realm, permission-mapper=default-permission-mapper)
/subsystem=elytron/http-authentication-factory=application-security-http:add(http-server-mechanism-
factory=global, security-domain=application-security, mechanism-configurations=[{mechanism-
name=BASIC}])
```

これにより、サーバーが以下のように設定されます。

例: Elytron 設定

```
<subsystem xmlns="urn:wildfly:elytron:4.0" final-providers="combined-providers" disallowed-
providers="OracleUcrypto">
  ...
  <security-domains>
    ...
```

```
    <security-domain name="application-security" default-realm="combined-realm" permission-
mapper="default-permission-mapper">
      <realm name="combined-realm"/>
    </security-domain>
  </security-domains>
  <security-realms>
    <aggregate-realm name="combined-realm" authentication-realm="application-properties"
authorization-realm="ldap-realm"/>
      ...
      <properties-realm name="application-properties">
        <users-properties path="example-users.properties" relative-to="jboss.server.config.dir" digest-
realm-name="Application Security" plain-text="true"/>
      </properties-realm>
      <ldap-realm name="ldap-realm" dir-context="ldap-connection" direct-verification="true">
        <identity-mapping rdn-identifier="uid" search-base-dn="ou=users,dc=group-to-
principal,dc=wildfly,dc=org">
          <attribute-mapping>
            <attribute from="uid" to="Roles" filter="(uniqueMember={1})" filter-base-
dn="ou=groups,dc=group-to-principal,dc=wildfly,dc=org"/>
          </attribute-mapping>
        </identity-mapping>
      </ldap-realm>
  </security-realms>
  ...
  <http>
    ...
    <http-authentication-factory name="application-security-http" http-server-mechanism-
factory="global" security-domain="application-security">
      <mechanism-configuration>
        <mechanism mechanism-name="BASIC"/>
      </mechanism-configuration>
    </http-authentication-factory>
    ...
  </http>
  ...
  <dir-contexts>
    <dir-context name="ldap-connection" url="ldap://localhost:10389"
principal="uid=admin,ou=system">
      <credential-reference clear-text="secret"/>
    </dir-context>
  </dir-contexts>
</subsystem>
```

elytron サブシステムでは、認証に使用するセキュリティーレルムと承認の決定に使用するセキュリティーレルムを指定するために **aggregate-realm** が定義されています。

9.3.8. キャッシングを使用するセキュリティードメインを Elytron に以降する

PicketBox を使用する場合、セキュリティードメインを定義し、アクセスのためにインメモリーキャッシングを有効にすることが可能でした。これにより、ユーザーはメモリー内のアイデンティティーデータにアクセスし、アイデンティティーストアへの追加の直接アクセスを回避できるようになりました。同様の設定を Elytron で実現することができます。このセクションでは、PicketBox 設定の例と、Elytron を使用する場合の同等のセキュリティードメインキャッシュ設定を示します。

9.3.8.1. PicketBox のキャッシュ済みセキュリティードメイン設定

以下のコマンドは、JBoss EAP 7.4 以前でキャッシングを有効にする PicketBox セキュリティードメイン設定を示しています。

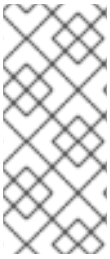
例: PicketBox のキャッシュ済みセキュリティードメインコマンド

```
/subsystem=security/security-domain=application-security:add(cache-type=default)
/subsystem=security/security-domain=application-security/authentication=classic:add(login-modules=
[{code=LdapExtended, flag=Required, module-options={
java.naming.factory.initial=com.sun.jndi.ldap.LdapCtxFactory,
java.naming.provider.url=ldap://localhost:10389, java.naming.security.authentication=simple,
bindDN="uid=admin,ou=system", bindCredential=secret, baseCtxDN="ou=users,dc=group-to-
principal,dc=wildfly,dc=org", baseFilter="(uid={0})", rolesCtxDN="ou=groups,dc=group-to-
principal,dc=wildfly,dc=org", roleFilter="(uniqueMember={1})", roleAttributeID="uid" }}])
```

これにより、サーバーが以下のように設定されます。

例: PicketBox のキャッシュ済みセキュリティードメイン設定

```
<subsystem xmlns="urn:jboss:domain:security:2.0">
  <security-domains>
    ...
    <security-domain name="application-security" cache-type="default">
      <authentication>
        <login-module code="LdapExtended" flag="required">
          <module-option name="java.naming.factory.initial" value="com.sun.jndi.ldap.LdapCtxFactory"/>
          <module-option name="java.naming.provider.url" value="ldap://localhost:10389"/>
          <module-option name="java.naming.security.authentication" value="simple"/>
          <module-option name="bindDN" value="uid=admin,ou=system"/>
          <module-option name="bindCredential" value="secret"/>
          <module-option name="baseCtxDN" value="ou=users,dc=group-to-
principal,dc=wildfly,dc=org"/>
          <module-option name="baseFilter" value="(uid={0})"/>
          <module-option name="rolesCtxDN" value="ou=groups,dc=group-to-
principal,dc=wildfly,dc=org"/>
          <module-option name="roleFilter" value="(uniqueMember={1})"/>
          <module-option name="roleAttributeID" value="uid"/>
        </login-module>
      </authentication>
    </security-domain>
  </security-domains>
</subsystem>
```



注記

このコマンドとこのコマンドによって得られた設定は、LDAP 認証設定の Elytron への移行 の例と似ていますが、ここでは **cache-type** 属性が **default** の値で定義されています。**default** キャッシュタイプはインメモリーキャッシュです。PicketBox を使用する場合、**infinispan** の **cache-type** を指定することもできますが、このタイプは Elytron ではサポートされません。

9.3.8.2. Elytron のキャッシュされたセキュリティードメインの設定

以下の手順に従って、Elytron の使用時にセキュリティードメインをキャッシュする同様の設定を作成します。

手順

1. セキュリティーレルムを定義し、キャッシングレルムでセキュリティーレルムをラップします。これにより、キャッシングレルムをセキュリティードメインで使用でき、引き続き認証ファクトリーで使用できます。

   例: Elytron セキュリティーレルム設定コマンド

   ```
   /subsystem=elytron/dir-context=ldap-connection:add(url=ldap://localhost:10389,
   principal="uid=admin,ou=system", credential-reference={clear-text=secret})
   /subsystem=elytron/ldap-realm=ldap-realm:add(dir-context=ldap-connection, direct-
   verification=true, identity-mapping={search-base-dn="ou=users,dc=group-to-
   principal,dc=wildfly,dc=org", rdn-identifier="uid", attribute-mapping=[{filter-base-
   dn="ou=groups,dc=group-to-principal,dc=wildfly,dc=org",filter="(uniqueMember=
   {1})",from="uid",to="Roles"}]})
   /subsystem=elytron/caching-realm=cached-ldap:add(realm=ldap-realm)
   ```

2. 前述のステップで定義された **cached-ldap** レルムを使用するセキュリティードメインと HTTP 認証ファクトリーを定義します。

   例: Elytron セキュリティードメインおよび認証ファクトリー設定コマンド

   ```
   /subsystem=elytron/security-domain=application-security:add(realms=[{realm=cached-ldap}],
   default-realm=cached-ldap, permission-mapper=default-permission-mapper)
   /subsystem=elytron/http-authentication-factory=application-security-http:add(http-server-
   mechanism-factory=global, security-domain=application-security, mechanism-
   configurations=[{mechanism-name=BASIC}])
   ```

   注記

   元のレルムではなく **caching-realm** を参照する必要があります。そうでないとキャッシングは迂回されます。

これらのコマンドにより、サーバー設定に以下が追加されます。

例: Elytron のキャッシュ済みセキュリティードメイン設定

```
<subsystem xmlns="urn:wildfly:elytron:4.0" final-providers="combined-providers" disallowed-
providers="OracleUcrypto">
  ...
  <security-domains>
    ...
    <security-domain name="application-security" default-realm="cached-ldap" permission-
mapper="default-permission-mapper">
      <realm name="cached-ldap"/>
    </security-domain>
  </security-domains>
  ...
  <security-realms>
    ....
  <ldap-realm name="ldap-realm" dir-context="ldap-connection" direct-verification="true">
      <identity-mapping rdn-identifier="uid" search-base-dn="ou=users,dc=group-to-
principal,dc=wildfly,dc=org">
        <attribute-mapping>
```

```
        <attribute from="uid" to="Roles" filter="(uniqueMember={1})" filter-base-
dn="ou=groups,dc=group-to-principal,dc=wildfly,dc=org"/>
       </attribute-mapping>
     </identity-mapping>
   </ldap-realm>
   <caching-realm name="cached-ldap" realm="ldap-realm"/>
  </security-realms>
  ...
  <http>
    ...
   <http-authentication-factory name="application-security-http" http-server-mechanism-
factory="global" security-domain="application-security">
     <mechanism-configuration>
       <mechanism mechanism-name="BASIC"/>
     </mechanism-configuration>
   </http-authentication-factory>
    ...
  </http>
   ...
  <dir-contexts>
   <dir-context name="ldap-connection" url="ldap://localhost:10389"
principal="uid=admin,ou=system">
     <credential-reference clear-text="secret"/>
   </dir-context>
  </dir-contexts>
  ...
```

9.3.9. Jakarta Authorization Security の Elytron への移行

デフォルトでは、JBoss EAP 7.4 以前では、レガシーセキュリティーサブシステムを使用して Jakarta Authorization ポリシープロバイダーおよびファクトリーを設定します。デフォルト設定は PicketBox から実装へマップします。

elytron サブシステムは、Java Authorization Contract for Containers (JACC) 仕様をベースとした組み込みポリシープロバイダーを提供します。

elytron サブシステムで Java Authorization Contract for Containers ポリシープロバイダーを有効にして定義する方法は、JBoss EAP 7.4 **開発ガイド** の Jakarta Authentication ポリシープロバイダーの定義 を参照してください。

9.4. アプリケーションクライアントの移行

このセクションでは、クライアントアプリケーションを Elytron に移行する方法について説明します。

ネーミングクライアント設定の Elytron への移行
ここでは、**org.jboss.naming.remote.client.InitialContextFactory** クラスによってバックされる **org.jboss.naming.remote.client.InitialContext** クラスを使用してリモート JNDI ルックアップを実行するクライアントアプリケーションを Elytron に移行する方法について説明します。

次の例では、**InitialContextFactory** クラスが、ユーザー認証情報と接続先のネーミングプロバイダーの URL のプロパティーを指定することによって作成されることを前提としています。

例: 以前のリリースで使用された InitialContext コード

```
Properties properties = new Properties();
```

```
properties.put(Context.INITIAL_CONTEXT_FACTORY,
"org.jboss.naming.remote.client.InitialContextFactory");
properties.put(Context.PROVIDER_URL,"http-remoting://127.0.0.1:8080");
properties.put(Context.SECURITY_PRINCIPAL, "bob");
properties.put(Context.SECURITY_CREDENTIALS, "secret");
InitialContext context = new InitialContext(properties);
Bar bar = (Bar) context.lookup("foo/bar");
...
```

以下の移行方法の 1 つを選択できます。

- 設定ファイルを使用したネーミングクライアントの移行
- プログラミングを使用したネーミングクライアントの移行

9.4.1. 設定ファイルアプローチを使用したネーミングクライアントの移行

設定アプローチを使用して、ネーミングクライアントを Elytron に移行します。

手順

1. クライアントアプリケーションの **META-INF/** ディレクトリーに **wildfly-config.xml** ファイルを作成します。このファイルには、ネーミングプロバイダーへの接続を確立するときに使用されるユーザークレデンシャルが含まれるようにします。

   例: wildfly-config.xml ファイル

   ```
   <configuration>
     <authentication-client xmlns="urn:elytron:client:1.7">
       <authentication-rules>
         <rule use-configuration="namingConfig">
           <match-host name="127.0.0.1"/>
         </rule>
       </authentication-rules>
       <authentication-configurations>
         <configuration name="namingConfig">
           <set-user-name name="bob"/>
           <credentials>
             <clear-password password="secret"/>
           </credentials>
         </configuration>
       </authentication-configurations>
     </authentication-client>
   </configuration>
   ```

2. 以下の例のように **InitialContext** を作成します。**InitialContext** は **org.wildfly.naming.client.WildFlyInitialContextFactory** クラスにバックされることに注意してください。

   例: InitialContext コード

   ```
   Properties properties = new Properties();
   properties.put(Context.INITIAL_CONTEXT_FACTORY,"org.wildfly.naming.client.WildFlyInitial
   ContextFactory");
   ```

```
properties.put(Context.PROVIDER_URL,"remote+http://127.0.0.1:8080");
InitialContext context = new InitialContext(properties);
Bar bar = (Bar) context.lookup("foo/bar");
...
```

9.4.2. プログラムによるアプローチを使用したネーミングクライアントの移行

この方法では、ネーミングプロバイダーへ接続を確立するために使用されるユーザークレデンシャルを直接アプリケーションコードに提供します。

例: プログラミングを使用したコード

```
// Create the authentication configuration
AuthenticationConfiguration namingConfig =
AuthenticationConfiguration.empty().useName("bob").usePassword("secret");

// Create the authentication context
AuthenticationContext context =
AuthenticationContext.empty().with(MatchRule.ALL.matchHost("127.0.0.1"), namingConfig);

// Create a callable that creates and uses an InitialContext
Callable<Void> callable = () -> {
    Properties properties = new Properties();

properties.put(Context.INITIAL_CONTEXT_FACTORY,"org.wildfly.naming.client.WildFlyInitialContextF
actory");
    properties.put(Context.PROVIDER_URL,"remote+http://127.0.0.1:8080");
    InitialContext context = new InitialContext(properties);
    Bar bar = (Bar) context.lookup("foo/bar");
    ...
    return null;
};

// Use the authentication context to run the callable
context.runCallable(callable);
```

9.4.3. Jakarta Enterprise Beans クライアントの Elytron への移行

この移行例は、**jboss-ejb-client.properties** ファイルを使用してリモートサーバーにデプロイされた Jakarta Enterprise Beans を呼び出すようクライアントアプリケーションが設定されていることを仮定します。クライアントアプリケーションの **META-INF/** ディレクトリーにあるこのファイルには、リモートサーバーへの接続に必要な以下の情報が含まれています。

例: jboss-ejb-client.properties ファイル

```
remote.connectionprovider.create.options.org.xnio.Options.SSL_ENABLED=false
remote.connections=default
remote.connection.default.host=127.0.0.1
remote.connection.default.port = 8080
remote.connection.default.username=bob
remote.connection.default.password=secret
```

クライアントは以下の例と似たコードを使用して Jakarta Enterprise Beans を検索し、メソッドの 1 つを呼び出します。

例: リモート Jakarta Enterprise Beans を呼び出すクライアントコード

```
// Create an InitialContext
Properties properties = new Properties();
properties.put(Context.URL_PKG_PREFIXES, "org.jboss.ejb.client.naming");
InitialContext context = new InitialContext(properties);

// Look up the Jakarta Enterprise Beans and invoke one of its methods
RemoteCalculator statelessRemoteCalculator = (RemoteCalculator) context.lookup(
    "ejb:/ejb-remote-server-side//CalculatorBean!" + RemoteCalculator.class.getName());
int sum = statelessRemoteCalculator.add(101, 202);
```

以下の移行方法の 1 つを選択できます。

- 設定ファイルを使用した Jakarta Enterprise Beans クライアントの移行
- Jakarta Enterprise Beans クライアントをプログラムで移行

9.4.3.1. 設定ファイルを使用した Jakarta Enterprise Beans クライアントの移行

以下の手順に従って、設定ファイルを使用してネーミングクライアントを Elytron に移行します。

手順

1. クライアントアプリケーションの **META-INF/** ディレクトリーで **wildfly-config.xml** ファイルを設定します。このファイルには、ネーミングプロバイダーへの接続を確立するときに使用されるユーザークレデンシャルが含まれるようにします。

   例: wildfly-config.xml ファイル

```
<configuration>
  <authentication-client xmlns="urn:elytron:client:1.7">
    <authentication-rules>
      <rule use-configuration="ejbConfig">
        <match-host name="127.0.0.1"/>
      </rule>
    </authentication-rules>
    <authentication-configurations>
      <configuration name="ejbConfig">
        <set-user-name name="bob"/>
        <credentials>
          <clear-password password="secret"/>
        </credentials>
      </configuration>
    </authentication-configurations>
  </authentication-client>
  <jboss-ejb-client xmlns="urn:jboss:wildfly-client-ejb:3.0">
    <connections>
      <connection uri="remote+http://127.0.0.1:8080" />
    </connections>
  </jboss-ejb-client>
</configuration>
```

2. 以下の例のように **InitialContext** を作成します。**InitialContext** は **org.wildfly.naming.client.WildFlyInitialContextFactory** クラスにバックされることに注意してください。

   例: InitialContext コード

```
// Create an InitialContext
Properties properties = new Properties();
properties.put(Context.INITIAL_CONTEXT_FACTORY,"org.wildfly.naming.client.WildFlyInitial
ContextFactory");
InitialContext context = new InitialContext(properties);

// Look up an Jakarta Enterprise Beans and invoke one of its methods
// Note that this code is the same as before
RemoteCalculator statelessRemoteCalculator = (RemoteCalculator) context.lookup(
    "ejb:/ejb-remote-server-side//CalculatorBean!" + RemoteCalculator.class.getName());
int sum = statelessRemoteCalculator.add(101, 202);----
```

3. 廃止された **jboss-ejb-client.properties** ファイルは必要がないため、削除できます。

9.4.3.2. Jakarta Enterprise Beans クライアントをプログラムで移行

以下の手順を使用して、Jakarta Enterprise Beans クライアントをプログラムで移行します。

手順

- リモートサーバーへの接続に必要な情報を直接アプリケーションコードに提供します。

例: プログラミングを使用したコード

```
// Create the authentication configuration
AuthenticationConfiguration ejbConfig =
AuthenticationConfiguration.empty().useName("bob").usePassword("secret");

// Create the authentication context
AuthenticationContext context =
AuthenticationContext.empty().with(MatchRule.ALL.matchHost("127.0.0.1"), ejbConfig);

// Create a callable that invokes the Jakarta Enterprise Beans
Callable<Void> callable = () -> {

    // Create an InitialContext
    Properties properties = new Properties();
    properties.put(Context.INITIAL_CONTEXT_FACTORY,
"org.wildfly.naming.client.WildFlyInitialContextFactory");
    properties.put(Context.PROVIDER_URL, "remote+http://127.0.0.1:8080");
    InitialContext context = new InitialContext(properties);

    // Look up the Jakarta Enterprise Beans and invoke one of its methods
    // Note that this code is the same as before
    RemoteCalculator statelessRemoteCalculator = (RemoteCalculator) context.lookup(
        "ejb:/ejb-remote-server-side//CalculatorBean!" + RemoteCalculator.class.getName());
    int sum = statelessRemoteCalculator.add(101, 202);
    ...
    return null;
```

```
};

// Use the authentication context to run the callable
context.runCallable(callable);
```

廃止された **jboss-ejb-client.properties** ファイルは必要がないため、削除できます。

9.5. SSL 設定の移行

以下の情報が含まれる Elytron を使用するには、アプリケーションの SSL 設定を移行します。

簡単な SSL 設定の Elytron への移行

セキュリティーレルムを使用して JBoss EAP サーバーへの HTTP 接続を保護した場合は、このセクションで提供される情報を使用して SSL 設定を Elytron に移行します。

前提条件

セキュリティーレルムを使用して JBoss EAP サーバーへの HTTP 接続を保護している。

ここで使用する例は、以下の **keystore** が **security-realm** に設定されていることを仮定します。

例: セキュリティーレルムキーストアを使用した SSL 設定

```
<security-realm name="ApplicationRealm">
  <server-identities>
    <ssl>
      <keystore path="server.keystore" relative-to="jboss.server.config.dir" keystore-
password="keystore_password" alias="server" key-password="key_password" />
    </ssl>
  </server-identities>
</security-realm>
```

Elytron を使用して同じ設定を実現するには、以下の手順を実行します。

手順

1. キーストアと暗号化されたパスワードの場所を指定する **key-store** を **elytron** サブシステムに設定します。このコマンドは、keytool コマンドを使用してキーストアが生成され、そのタイプが **JKS** であることを仮定しています。

   ```
   /subsystem=elytron/key-store=LocalhostKeyStore:add(path=server.keystore,relative-
   to=jboss.server.config.dir,credential-reference={clear-text="keystore_password"},type=JKS)
   ```

2. 前のステップで定義された **key-store**、エイリアス、およびキーのパスワードを指定する **key-manager** を **elytron** サブシステムに作成します。

   ```
   /subsystem=elytron/key-manager=LocalhostKeyManager:add(key-
   store=LocalhostKeyStore,alias-filter=server,credential-reference={clear-
   text="key_password"})
   ```

3. 前のステップで定義した **key-manager** を参照する **server-ssl-context** を **elytron** サブシステムに作成します。

```
/subsystem=elytron/server-ssl-context=LocalhostSslContext:add(key-
manager=LocalhostKeyManager)
```

4. **https-listener** をレガシー **security-realm** から新規作成された Elytron **ssl-context** に切り替えます。

```
batch
/subsystem=undertow/server=default-server/https-listener=https:undefine-
attribute(name=security-realm)
/subsystem=undertow/server=default-server/https-listener=https:write-attribute(name=ssl-
context,value=LocalhostSslContext)
run-batch
```

5. サーバーをリロードします。

```
reload
```

これにより、サーバー設定ファイルの **elytron** サブシステム設定が以下のようになります。

```
<subsystem xmlns="urn:wildfly:elytron:4.0" ...>
  ...
  <tls>
    <key-stores>
      <key-store name="LocalhostKeyStore">
        <credential-reference clear-text="keystore_password"/>
        <implementation type="JKS"/>
        <file path="server.keystore" relative-to="jboss.server.config.dir"/>
      </key-store>
    </key-stores>
    <key-managers>
      <key-manager name="LocalhostKeyManager" key-store="LocalhostKeyStore"  alias-
filter="server">
        <credential-reference clear-text="key_password"/>
      </key-manager>
    </key-managers>
    <server-ssl-contexts>
      <server-ssl-context name="LocalhostSslContext" key-manager="LocalhostKeyManager"/>
    </server-ssl-contexts>
  </tls>
</subsystem>
```

これにより、サーバー設定ファイルの **undertow** サブシステム設定が以下のようになります。

```
<https-listener name="https" socket-binding="https" ssl-context="LocalhostSslContext" enable-
http2="true"/>
```

詳細は、JBoss EAP 7.4 **サーバーセキュリティーの設定方法** の Elytron サブシステム および 管理インターフェイスの保護方法 を参照してください。

9.5.1. CLIENT-CERT SSL 認証の Elytron への移行

CLIENT-CERT SSL 認証を有効にするには、**truststore** 要素を **authentication** 要素に追加します。

```
<security-realm name="ManagementRealm">
  <server-identities>
    <ssl>
      <keystore path="server.keystore" relative-to="jboss.server.config.dir" keystore-
password="KEYSTORE_PASSWORD" alias="server" key-password="key_password" />
    </ssl>
  </server-identities>
  <authentication>
    <truststore path="server.truststore" relative-to="jboss.server.config.dir" keystore-
password="TRUSTSTORE_PASSWORD" />
    <local default-user="$local"/>
    <properties path="mgmt-users.properties" relative-to="jboss.server.config.dir"/>
  </authentication>
</security-realm>
```

注記

この設定では、**CLIENT-CERT** 認証が発生しない場合、クライアントをフォールバックし、ローカルメカニズムまたは **username/password** 認証メカニズムのいずれかを使用することができます。**CLIENT-CERT** ベースの認証を強制するには、**local** および **properties** 要素を削除します。

レガシー **truststore** の使用方法は 2 つあります。

- CA のみが含まれる レガシー **truststore**
- クライアントの証明書が含まれるレガシー **truststore**

9.5.1.1. CA のみが含まれるレガシー truststore

以下の手順に従って、有効な証明書とプライベートキーを持たないユーザーが Elytron を使用してサーバーにアクセスしないようにサーバーを設定します。

手順

1. キーストアと暗号化されたパスワードの場所を指定する **key-store** を **elytron** サブシステムに設定します。このコマンドは、keytool コマンドを使用してキーストアが生成され、そのタイプが **JKS** であることを仮定しています。

```
/subsystem=elytron/key-store=LocalhostKeyStore:add(path=server.keystore,relative-
to=jboss.server.config.dir,credential-reference={clear-text="keystore_password"},type=JKS)
```

2. トラストストアと暗号化されたパスワードの場所を指定する **key-store** を **elytron** サブシステムに設定します。このコマンドは、keytool コマンドを使用してキーストアが生成され、そのタイプが **JKS** であることを仮定しています。

```
/subsystem=elytron/key-store=TrustStore:add(path=server.truststore,relative-
to=jboss.server.config.dir,credential-reference={clear-text="truststore_password"},type=JKS)
```

3. 前のステップで定義された **LocalhostKeyStore** キーストア、エイリアス、およびキーのパスワードを指定する **key-manager** を **elytron** サブシステムに作成します。

```
/subsystem=elytron/key-manager=LocalhostKeyManager:add(key-
store=LocalhostKeyStore,alias-filter=server,credential-reference={clear-
text="key_password"})
```

4. 前のステップで作成されたトラストストアの **key-store** を指定する **trust-manager** を **elytron** サブシステムに作成します。

```
/subsystem=elytron/trust-manager=TrustManager:add(key-store=TrustStore)
```

5. 前のステップで定義した **key-manager** の参照、**trust-manager** 属性の設定、およびクライアント認証の有効化を行う **server-ssl-context** を **elytron** サブシステムに作成します。

```
/subsystem=elytron/server-ssl-context=LocalhostSslContext:add(key-
manager=LocalhostKeyManager,trust-manager=TrustManager,need-client-auth=true)
```

6. **https-listener** を、新しく作成した Elytron **ssl-context** に更新します。

```
/subsystem=undertow/server=default-server/https-listener=https:write-attribute(name=ssl-
context,value=LocalhostSslContext)
```

7. サーバーをリロードします。

```
reload
```

これにより、サーバー設定ファイルの **elytron** サブシステム設定が以下のようになります。

```
<subsystem xmlns="urn:wildfly:elytron:4.0"...>
  ...
  <tls>
    <key-stores>
      <key-store name="LocalhostKeyStore">
        <credential-reference clear-text="keystore_password"/>
        <implementation type="JKS"/>
        <file path="server.keystore" relative-to="jboss.server.config.dir"/>
      </key-store>
      <key-store name="TrustStore">
        <credential-reference clear-text="truststore_password"/>
        <implementation type="JKS"/>
        <file path="server.truststore" relative-to="jboss.server.config.dir"/>
      </key-store>
    </key-stores>
    <key-managers>
      <key-manager name="LocalhostKeyManager" key-store="LocalhostKeyStore" alias-
filter="server">
        <credential-reference clear-text="key_password"/>
      </key-manager>
    </key-managers>
    <trust-managers>
      <trust-manager name="TrustManager" key-store="TrustStore"/>
    </trust-managers>
    <server-ssl-contexts>
      <server-ssl-context name="LocalhostSslContext" need-client-auth="true" key-
manager="LocalhostKeyManager" trust-manager="TrustManager"/>
```

```
    </server-ssl-contexts>
  </tls>
</subsystem>
```

これにより、サーバー設定ファイルの **undertow** サブシステム設定が以下のようになります。

```
<subsystem xmlns="urn:jboss:domain:undertow:14.0">
...
<https-listener name="https" socket-binding="https" ssl-context="LocalhostSslContext" enable-http2="true"/>
...
</subsystem>
```

9.5.1.2. セキュリティーレルムおよびドメイン

セキュリティーレルムは 2 つの状況で使用されます。

- 証明書の認証に失敗したとき、セキュリティーレルムはパスワードのフォールバックで使用されます。
- パスワードと証明書に対する承認が完了したとき、レルムは各ユーザーのロールを提供します。

事前定義された Elytron **ManagementDomain** セキュリティードメインと **ManagementRealm** セキュリティーレルムを使用できるようにするため、ユーザーは標準のプロパティーファイルに格納されます。

```
<security-domains>
    <security-domain name="ManagementDomain" default-realm="ManagementRealm" permission-mapper="default-permission-mapper">
        <realm name="ManagementRealm" role-decoder="groups-to-roles"/>
        <realm name="local"/>
    </security-domain>
</security-domains>
<security-realms>
    <properties-realm name="ManagementRealm">
        <users-properties path="mgmt-users.properties" relative-to="jboss.server.config.dir" digest-realm-name="ManagementRealm"/>
        <groups-properties path="mgmt-groups.properties" relative-to="jboss.server.config.dir"/>
    </properties-realm>
</security-realms>
```

そのため、すべてのクライアント証明書に対してユーザーがセキュリティーレルムに存在する必要があります。

9.5.1.3. プリンシパルデコーダー

証明書認証が使用され、セキュリティーレルムがユーザー名を許可してアイデンティティーを解決する場合、クライアント証明書から **username** を取得する方法の定義が必要です。

この場合、証明書サブジェクトで **CN** 属性が使用されます。

```
/subsystem=elytron/x500-attribute-principal-decoder=x500-decoder:add(attribute-name=CN)
```

9.5.1.4. HTTP 認証ファクトリー

HTTP 接続では、以前定義したリソースを使用して HTTP 認証ファクトリーが定義されます。これは、**CLIENT_CERT** および **DIGEST** 認証をサポートするために設定されます。

プロパティーレルムはパスワードのみを検証し、クライアント証明書を検証できないため、最初に設定メカニズムのファクトリーを追加する必要があります。これは、セキュリティーレルムに対する証明書の検証を無効にします。

```
/subsystem=elytron/configurable-http-server-mechanism-factory=configured-cert:add(http-server-mechanism-factory=global, properties={org.wildfly.security.http.skip-certificate-verification=true})
```

HTTP 認証は次のように作成できます。

```
./subsystem=elytron/http-authentication-factory=client-cert-digest:add(http-server-mechanism-factory=configured-cert,security-domain=ManagementDomain,mechanism-configurations=[{mechanism-name=CLIENT_CERT,pre-realm-principal-transformer=x500-decoder},{mechanism-name=DIGEST, mechanism-realm-configurations=[{realm-name=ManagementRealm}]}])
```

上記のコマンドの結果は次のとおりです。

```
<subsystem xmlns="urn:wildfly:elytron:4.0" final-providers="combined-providers" disallowed-providers="OracleUcrypto">
  ...
  <http>
    ...
    <http-authentication-factory name="client-cert-digest" http-server-mechanism-factory="configured-cert" security-domain="ManagementDomain">
      <mechanism-configuration>
        <mechanism mechanism-name="CLIENT_CERT" pre-realm-principal-transformer="x500-decoder"/>
        <mechanism mechanism-name="DIGEST">
          <mechanism-realm realm-name="ManagementRealm"/>
        </mechanism>
      </mechanism-configuration>
    </http-authentication-factory>
    ...
    <configurable-http-server-mechanism-factory name="configured-cert" http-server-mechanism-factory="configured-cert">
        <properties>
            <property name="org.wildfly.security.http.skip-certificate-verification" value="true"/>
        </properties>
    </configurable-http-server-mechanism-factory>
    ...
  </http>
  ...
</subsystem>
```

9.6. LDAP でのレガシーセキュリティー動作の変更

Red Hat JBoss Enterprise Application Platform 8.0 では、LDAP に大幅な変更があります。これらの変更には、到達不能な LDAP レルムの HTTP ステータスの変更、DN からのロール解析のための LDAP セキュリティーレルムの有効化、LDAP サーバーへの JBoss EAP SSL 証明書の送信の変更が含まれます。

- **Elytron** サブシステムを利用する JBoss EAP 8.0 では、LDAP レルムが到達不能な場合、"500 Internal Error" が返され、到達不能なレルムに対する HTTP ステータスの変更が示されます。

- JBoss EAP 8.0 では、LDAP セキュリティーレルムを有効にして DN からのロールを解析できます。ID に関連付けられた属性として LDAP から情報をロードすると、後で attribute-mapping 要素を使用して、これらの属性をロールにマッピングできます。

  設定例

  ```
  <ldap-realm name="ldap-realm" dir-context="ldap-connection" direct-verification="true">
    <identity-mapping rdn-identifier="uid" search-base-dn="ou=users,dc=group-to-principal,dc=wildfly,dc=org">
      <attribute-mapping>
        <attribute from="dn" to="Roles" filter="(uniqueMember={1})" filter-base-dn="ou=groups,dc=group-to-principal,dc=wildfly,dc=org"/>
      </attribute-mapping>
    </identity-mapping>
  </ldap-realm>
  ```

- JBoss EAP 8.0 では、JBoss EAP SSL 証明書を LDAP サーバーに送信するために、双方向または相互 TLS を利用するようにアウトバウンド LDAP 接続を設定するオプションがあります。詳細は、管理インターフェイスとアプリケーションの双方向 SSL/TLS の有効化 を参照してください。

付録A 参考資料

JBoss EAP のユーザーは、異なるリリース間のシームレスな互換性と相互運用性を期待できます。さまざまなクライアントからサーバーへの接続がサポートされていますが、特定のケースでは追加の考慮事項が必要となります。

A.1. リリース間の互換性と相互運用性

このセクションでは、JBoss EAP 6、JBoss EAP 7、および JBoss EAP 8.0 リリース間での、クライアントおよびサーバーエンタープライズ Bean とメッセージングコンポーネントの互換性および相互運用性について説明します。

A.1.1. インターネット Inter-ORB プロトコルを介したエンタープライズ Bean のリモート処理

以下の設定はエラーなしで実行する必要があります。

- JBoss EAP 6 クライアントから JBoss EAP 8.0 サーバーへの接続
- JBoss EAP 7 クライアントから JBoss EAP 8.0 サーバーへの接続
- JBoss EAP 8.0 クライアントから JBoss EAP 7 サーバーへの接続
- JBoss EAP 8.0 クライアントから JBoss EAP 6 サーバーへの接続

A.1.2. Java Naming and Directory Interface を使用したエンタープライズ Bean のリモーティング

以下の設定はエラーなしで実行する必要があります。

- JBoss EAP 7 クライアントから JBoss EAP 8.0 サーバーへの接続
- JBoss EAP 8.0 クライアントから JBoss EAP 7 サーバーへの接続

JBoss EAP 6 は Enterprise Beans 3.1 仕様のサポートを提供し、標準化されたグローバル Java Naming and Directory Interface 名前空間の使用を導入しました。これらは JBoss EAP 8.0 でも引き続き使用されています。Java Naming and Directory Interface の名前空間名の変更によって、次の設定に非互換性が生じることはありません。

- JBoss EAP 6 クライアントから JBoss EAP 8.0 または JBoss EAP 7 サーバーへの接続
- JBoss EAP 8.0 または JBoss EAP 7 クライアントから JBoss EAP 6 サーバーへの接続

A.1.3. @WebService を使用したエンタープライズ Bean のリモート処理

以下の設定はエラーなしで実行する必要があります。

- JBoss EAP 6 クライアントから JBoss EAP 8.0 サーバーへの接続
- JBoss EAP 7 クライアントから JBoss EAP 8.0 サーバーへの接続
- JBoss EAP 8.0 クライアントから JBoss EAP 7 サーバーへの接続
- JBoss EAP 8.0 クライアントから JBoss EAP 6 サーバーへの接続

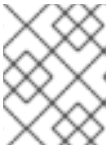
A.1.4. メッセージングスタンドアロンクライアント

以下の設定はエラーなしで実行する必要があります。

- JBoss EAP 7 クライアントから JBoss EAP 8.0 サーバーへの接続
- JBoss EAP 8.0 クライアントから JBoss EAP 7 サーバーへの接続

JBoss EAP 8.0 組み込みメッセージングは、プロトコル互換性の問題により、JBoss EAP 6 に同梱されている HornetQ 2.3.x に接続できません。このため、次の設定には互換性がありません。

- JBoss EAP 8.0 クライアントから JBoss EAP 6 サーバーへの接続
- JBoss EAP 6 クライアントから JBoss EAP 8.0 サーバーへの接続



注記

この接続を可能にするには、Java Naming and Directory Interface を通じてアクセスできるレガシー接続ファクトリーを作成する必要があります。

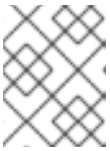
A.1.5. メッセージング MDB

以下の設定はエラーなしで実行する必要があります。

- JBoss EAP 7 クライアントから JBoss EAP 8.0 サーバーへの接続
- JBoss EAP 8.0 クライアントから JBoss EAP 7 サーバーへの接続

JBoss EAP 8.0 組み込みメッセージングは、プロトコル互換性の問題により、JBoss EAP 6 に同梱されている HornetQ 2.3.x に接続できません。このため、次の設定には互換性がありません。

- JBoss EAP 8.0 クライアントから JBoss EAP 6 サーバーへの接続
- JBoss EAP 6 クライアントから JBoss EAP 8.0 サーバーへの接続



注記

この接続を可能にするには、Java Naming and Directory Interface を通じてアクセスできるレガシー接続ファクトリーを作成する必要があります。

A.1.6. メッセージングブリッジ

以下の設定はエラーなしで実行する必要があります。

- JBoss EAP 6 クライアントから JBoss EAP 8.0 サーバーへの接続
- JBoss EAP 7 クライアントから JBoss EAP 8.0 サーバーへの接続
- JBoss EAP 8.0 クライアントから JBoss EAP 7 サーバーへの接続
- JBoss EAP 8.0 クライアントから JBoss EAP 6 サーバーへの接続