Red Hat JBoss Enterprise Application Platform 7.2

移行ガイド

Red Hat JBoss Enterprise Application Platform 7.2 向け

Last Updated: 2020-01-08

Red Hat JBoss Enterprise Application Platform 7.2 移行ガイド

Red Hat JBoss Enterprise Application Platform 7.2 向け

法律上の通知

概要

本ガイドでは、以前のバージョンの Red Hat JBoss Enterprise Application Platform からアプリケーションを移行する方法を説明します。

目次

第1章 はじめに

本書には、Red Hat JBoss Enterprise Application Platform 6 のアプリケーションを Red Hat JBoss Enterprise Application Platform 7 で正常に実行し、デプロイするために必要な変更内容が記載されています。本リリースで利用できる新機能、非推奨機能、およびサポート対象外となった機能に関する情報を提供し、アプリケーションの挙動が変わらないように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も取り上げます。

また、本書では Java アプリケーションの移行を簡単にする Red Hat Application Migration Toolkit や、サーバー設定を更新する JBoss Server Migration Tool などの、移行に役立つツールについても説明します。

アプリケーションが正常にデプロイされ、実行されたら、各コンポーネントをアップグレードして JBoss EAP 7 の新機能を使用する計画を立てることができます。

JBoss EAP 5 のアプリケーションを直接 JBoss EAP 7 に移行する場合は、「JBoss EAP の旧リリースからの移行」 を参照してください。

1.1. 移行およびアップグレード

メジャーアップグレード

JBoss EAP 6.4 から JBoss EAP 7.0 など、アプリケーションを他のメジャーリリースに移動する場合にメジャーアップグレードまたは移行が必要になります。アプリケーションが Java EE 仕様に準拠し、非推奨の API にアクセスせず、プロプライエタリーコードを含まない場合、アプリケーションコードを変更せずにアプリケーションを JBoss EAP 7 で実行できる可能性があります。しかし、JBoss EAP 7 ではサーバー設定が変更になったため移行が必要になります。本書ではこのような移行を取り上げます。

マイナーアップデート

JBoss EAP では、定期的にポイントリリースが提供されます。これは、バグ修正、セキュリティーの修正、および新機能が含まれるマイナーアップデートです。 ポイントリリースで追加された変更に関する情報は、本書と『7.2.0 リリースノート』に記載されています。

JBoss Server Migration Tool を使用すると、ポイントリリースを自動的に別のポイントリリースにアップグレードできます (JBoss EAP 7.0 から JBoss EAP 7.1 など)。このツールの設定および実行方法に関する詳細は、『Using the JBoss Server Migration Tool』を参照してください。

サーバー設定を手作業でアップグレードすることもできます。手作業でアップグレードする方法と手順については、JBoss EAP『パッチおよびアップグレードガイド』の「JBoss EAP のアップグレード」を参照してください。

累積パッチ

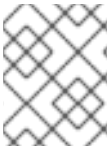
JBoss EAP では、バグおよびセキュリティーの修正が含まれる累積パッチも定期的に提供されます。累積パッチのリリースごとに、リリース番号の最後の数字が 1 ずつ増えます (例: 7.1.0 から 7.1.1)。パッチインストールの詳細は、JBoss EAP の『パッチおよびアップグレードガイド』に記載されています。

1.2. 本書における EAP_HOME の使用

本書では、変数 **EAP_HOME** を使用して JBoss EAP へのパスを示しています。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。

- ZIP インストール方法で 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、ZIP アーカイブを抽出した **jboss-eap-7.2** ディレクトリーとなります。

- RPM インストール方法で 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/eap7/root/usr/share/wildfly/** になります。
- インストーラーを使用して 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 **${user.home}/EAP-7.2.0** になります。
  - Red Hat Enterprise Linux および Solaris では、**/home/USER_NAME/EAP-7.2.0/** になります。
  - Microsoft Windows の場合、**C:\Users\USER_NAME\EAP-7.2.0** になります。
- Red Hat CodeReady Studio インストーラーを使用して JBoss EAP サーバーをインストールおよび設定した場合、**EAP_HOME** のデフォルトのパスは **${user.home}/devstudio/runtimes/jboss-eap** になります。
  - 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

EAP_HOME は環境変数ではありません。**JBOSS_HOME** がスクリプトで使用される環境変数です。

第2章 移行の準備

2.1. 準備の概要

JBoss EAP 7 では、JBoss EAP 6 のアプリケーションへの後方互換性確立が取り組まれています。しかし、JBoss EAP 7 で非推奨なった機能や削除された機能がアプリケーションによって使用される場合は、アプリケーションコードの変更が必要になることがあります。

さらに、本リリースには JBoss EAP 7 アプリケーションのデプロイメントに影響する可能性がある複数の変更が含まれています。移行について調査し、計画を立ててからアプリケーションを移行することが推奨されます。

- Java EE 8 の機能 を理解するようにしてください。
- JBoss EAP 6.4 から移行する場合は、Java EE 7 の機能 も理解するようにしてください。
- JBoss EAP 7 の新機能を確認してください。
- 非推奨の機能およびサポートされない機能のリストを確認してください。
- JBoss EAP 7 の『スタートガイド』をお読みください。
- 移行に便利なツールを確認してください。

機能の変更、開発の資料、および移行に便利な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、JBoss EAP 7 で実行するために必要な変更について判断します。

2.2. JAVA EE 8 機能の確認

Java EE 8 は、プライベートおよびパブリッククラウドでの機能が充実したアプリケーションの開発や実行を容易にする多くの改良点が含まれていた Java EE 7 上に構築されています。Java EE 7 には、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した。アップデートには JPA 2.1、JAX-RS 2.0、Servlet 3.1、Expression Language 3.0、JMS 2.0、JSF 2.2、EJB 3.2、CDI 1.2、および Bean Validation 1.1 が含まれていました。

Java EE 8 には、新しい移植可能なセキュリティー API、HTTP/2 サポートによる Java Servlet 4.0 のサポート、JPA 2.2、JAX-RS 2.1、JSF 2.3、CDI 2.0、強化された JSON サポートと新しい JSON バインディング API、非同期 CDI イベントのサポートなど、多くの強化機能が追加されました。

チュートリアルなどの Java EE 8 に関する詳細は、オラクルの Web サイトにある「Java EE at a Glance」を参照してください。

2.3. JAVA EE 7 機能の確認

JBoss EAP 6.4 から移行する場合、Java EE 7 には、プライベートおよびパブリッククラウドでの機能が充実したアプリケーションの開発や実行を容易にする多くの改良点が含まれています。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。アップデートには JPA 2.1、JAX-RS 2.0、Servlet 3.1、Expression Language 3.0、JMS 2.0、JSF 2.2、EJB 3.2、CDI 1.2、および Bean Validation 1.1 が含まれていました。

チュートリアルなどの Java EE 7 に関する詳細は、オラクルの Web サイトにある「Java™ EE Documentation」を参照してください。

2.4. JBOSS EAP 7 の新機能

JBoss EAP 7 には、以前のリリースからのアップグレードや改良点が含まれています。ここでは、JBoss EAP 7 のポイントリリースで導入された新機能および改良された機能の一部を取り上げます。

JBoss EAP 7.0 の新機能および改良された機能

Java EE 7

JBoss EAP 7 は Java EE 7 の認定実装で、 Web プロファイルおよびフルプラットフォーム仕様の両方に準拠しています。また、CDI 1.2 および Web Sockets 1.1 の最新のイテレーションもサポートします。

Undertow

Undertow は JBoss EAP 7 に含まれる、軽量で柔軟性のあるパフォーマンスに優れた新しい Web サーバーです。JBoss Web は Undertow に置き換えられました。Undertow は Java で書かれ、スループットとスケーラビリティーを最大にするよう設計されています。 新しい HTTP/2 標準などの最新の Web 技術をサポートします。

Apache ActiveMQ Artemis

Apache ActiveMQ Artemis は JBoss EAP 7 の新しいビルトインメッセージングプロバイダーです。この Apache サブプロジェクトは HornetQ から寄贈されたコードをベースにし、証明された非ブロッキングアーキテクチャーを基に優れたパフォーマンスを実現します。

IronJacamar 1.2

最新の Iron Jacamar は、安定性が高く、機能が充実したサポートを JCA および DataSources に提供します。

JBossWS 5

JBossWS 5 はこれまでのバージョンから大きく飛躍し、新機能や改良されたパフォーマンスを JBoss EAP 7 の web サービスに提供します。

RESTEasy 3

JBoss EAP 7 には最新の RESTEasy が含まれています。JSON Web Encryption、Jackson、JSON-P、Jettison などの便利な拡張を提供し、標準の Java EE REST API (JAX-RS 2.0) を越えた機能性を実現します。

OpenJDK ORB

JBoss EAP 7 では、JacORB IIOP 実装が OpenJDK ORB のダウンストリームブランチに置き換えられ、JVM ORB と Java EE RI との相互運用性が向上されました。

機能が充実したクラスタリング

JBoss EAP 7 ではクラスタリングのサポートが大幅にリファクタリングされ、アプリケーションのアクセスを可能にするパブリック API が複数含まれています。

ポートの削減

JBoss EAP 7 では HTTP のアップグレードを利用し、ほぼすべてのプロトコルが管理ポート (9990) とアプリケーションポート (8080) の 2 つの HTTP ポート上で多重化されます。

ロギングの強化

管理 API が、サーバー上で利用可能なログファイルをリストおよび表示する機能をサポートするようになりました。また、デフォルトのパターンフォーマッター以外のカスタムフォーマッターを定義する機能もサポートするようになりました。さらに、デプロイメントのロギング設定も大幅に向上されました。

JBoss EAP 7.0 に導入された新機能の完全リストは、JBoss EAP『7.0.0 リリースノート』の「新機能および改良された機能」を参照してください。

JBoss EAP 7.1 の新機能および改良された機能

Elytron

WildFly Elytron プロジェクトをベースとする Elytron は、JBoss EAP 7.1 の新しいセキュリティーフレームワークです。Elytron は、アプリケーションサーバー全体でセキュリティーを統一します。

管理コンソール

管理コンソールが改良され、より多くのサブシステムを設定できるようになりました。 強化された **transaction** サブシステムおよびトランザクションリソースメトリックスを提供し、多くの追加設定を管理します。

管理 CLI

管理 CLI では、**echo-command** 引数を使用する応答、添付ファイル、モジュール設定、およびデバッグのサポートが向上されました。

JBoss EAP 7.1 に導入された新機能の完全リストは、Red Hat カスタマーポータルで『7.1.0 リリースノート』の「新機能および改良された機能」を参照してください。

JBoss EAP 7.2 の新機能および改良された機能

Java EE 8

JBoss EAP 7.2 は Java EE 8 の認定実装です。Java Servlet 4.0、Java Persistence 2.2、CDI 2.0、JSF 2.3、JSON-B 1.0、JSON-P 1.1、JAX-RS 2.1 などのサポートが含まれています。Java Enterprise Edition (Java EE) 8 プラットフォームでサポートされる技術に関する詳細は、「Java™ EE 8 Technologies」を参照してください。

アプリケーションの開発に使用可能な BOM

Java EE 8 の JBoss EAP ランタイムの依存関係を提供する新しい BOM が利用できます。Java EE 7 の BOM 名には **javaee7** が含まれましたが、本リリースの BOM では名前に **javaee8** が含まれます。新しい BOM の詳細は、JBoss EAP『開発ガイド』の「プロジェクト依存関係の管理」を参照してください。

JBoss EAP 7.1 に導入された新機能の完全リストは、Red Hat カスタマーポータルで『7.1.0 リリースノート』の「新機能および改良された機能」を参照してください。

2.5. 非推奨および未サポート機能リストの確認

アプリケーションを JBoss EAP 7.2 に移行する前に、以前のリリースの JBoss EAP で利用できた機能の一部が非推奨またはサポート対象外となった可能性があることに注意してください。維持費の高さ、コミュニティーの関心の低さ、より優れた代替のソリューションなどが理由で、一部の技術のサポートが廃止されました。

非推奨およびサポート対象外となった機能の一部を以下に示します。

EJB エンティティー Bean

EJB エンティティー Bean はサポート対象外になりました。アプリケーションが EJB エンティティー Bean を使用する場合は、パフォーマンスや柔軟性が高い API を提供する JPA を使用するようコードを移行してください。

JAX-RPC

JAX-WS はより正確で完全なソリューションを提供するため、JAX-RPC 用に書かれたコードは JAX-WS を使用するよう移行する必要があります。

JSR-88

すべての Java EE プラットフォーム製品でアプリケーションを設定およびデプロイするために複数のプロバイダーからツールを有効化するコントラクトを定義する、Java EE Application Deployment API 仕様 (JSR-88) は広く採用されませんでした。管理コンソール、管理 CLI、デプロイメントスキャナー、Maven など、アプリケーションのデプロイメントでは JBoss EAP がサポートする他のオプションを使用する必要があります。

汎用 **JMS** リソースアダプター

汎用 JMS リソースアダプターを設定して JMS プロバイダーへ接続する機能はサポート対象外になりました。

IO サブシステム

IO バッファープールは非推奨となりました。これは Undertow バイトバッファープールに置き換えられました。

キャッシュストア

remote キャッシュストアは非推奨になりました。 代わりに **hotrod** キャッシュストアの使用が推奨されます。

プラットフォームおよび機能

JBoss EAP 7.2 では、これまでのリリースで使用できた複数のプラットフォームやデータベースが非推奨となりました。

JBoss EAP 7.0 で非推奨になった機能とサポート対象外になった機能の完全リストは、Red Hat カスタマーポータルで JBoss EAP『7.0.0 リリースノート』の「サポートされない機能および非推奨の機能」を参照してください。

JBoss EAP 7.1 で非推奨になった機能とサポート対象外になった機能の完全リストは、Red Hat カスタマーポータルで JBoss EAP『7.1.0 リリースノート』の「サポートされない機能および非推奨の機能」を参照してください。

JBoss EAP 7.2 で非推奨になった機能とサポート対象外になった機能の完全リストは、Red Hat カスタマーポータルで JBoss EAP『7.2.0 リリースノート』の「サポートされない機能および非推奨の機能」を参照してください。

2.6. JBOSS EAP 7 スタートガイドの確認

JBoss EAP の『スタートガイド』を必ず確認してください。このガイドには、以下の重要な情報が含まれています。

- JBoss EAP 7 のダウンロードおよびインストール方法
- Red Hat CodeReady Studio のダウンロードおよびインストール方法
- 開発環境に応じた Maven の設定方法、プロジェクト依存関係の管理方法、および JBoss EAP の Bill of Material (BOM) アーティファクトを使用するようプロジェクトを設定する方法。
- 製品に同梱されたクイックスタートサンプルアプリケーションのダウンロードおよび実行方法

2.7. 移行の分析および計画

アプリケーションとサーバー設定はそれぞれ異なるため、移行を始める前に既存のアプリケーションとサーバープラットフォームのコンポーネントおよびアーキテクチャーを十分に理解する必要があります。移行計画にはテストの詳細なロードマップが含まれる必要があり、以下の情報を考慮して実稼働に展開する必要があります。

移行責任者の特定

ステークホルダー、プロジェクトマネージャー、開発者、管理者、およびその他の移行責任者を特定します。

アプリケーションサーバープラットフォーム設定とハードウェアの確認

既存のアプリケーションサーバーとプラットフォーム設定を検証し、JBoss EAP 7 の今後の変更がどのように影響するかを判断します。以下の項目が含まれる必要があります。

- オペレーティングシステムおよびバージョン
- アプリケーションによって使用されるデータベース
- Web サーバー
- セキュリティーアーキテクチャー
- プロセッサーの数およびタイプ
- メモリーの容量
- 物理ディスクストレージの容量
- データベースまたはメッセージングデータの移行
- 移行の影響を受ける可能性があるその他のコンポーネント

現在の本番環境の確認

移行プロセスのテストおよびステージングでは、できるだけ本番環境に近い状態を再現するように計画を立てる必要があります。

- クラスタリング設定を考慮します。クラスターの移行方法に関する詳細は、JBoss EAP『パッチ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」を参照してください。
- 現在、大型の管理対象ドメインを実行している場合は、段階的な移行方法を考慮してください。
- データベースまたはメッセージングデータの移行が必要であるかどうかを判断します。

既存アプリケーションの検証および理解

既存の JBoss EAP 6 アプリケーションを完全に検証します。以下を含むアーキテクチャー、関数、機能、およびコンポーネントについて完全に理解してください。

- JVM バージョン
- 他の Red Hat アプリケーションサーバーミドルウェアコンポーネントとの統合
- プロプライエタリーサードパーティーソフトウェアとの統合
- 代替機能が必要となる非推奨機能の使用
- デプロイメント記述子、JNDI、永続性、JDBC 設定およびプーリング、JMS トピックおよびキュー、ロギングを含むアプリケーション設定

JBoss EAP 7 への移行中に変更が必要な互換性のないコードまたは設定を特定

詳細テストプランの作成

- 計画には、回帰テストと受け入れ基準の要件が含まれる必要があります。
- パフォーマンステストが含まれる必要もあります。
- 本番環境へロールアウトする前に移行をテストするため、できるだけ本番環境に近くなるようステージング環境を設定します。

- 必ず、バックアップおよびバックアウト計画を作成してください。

移行プロセスに使用できるリソースの確認

- 開発チームのスキルを評価し、トレーニングまたは追加のコンサルティングを計画します。
- 移行プロセス中は完了まで、ステージングやテストで追加のハードウェアやその他のリソースが必要になることにも注意してください。
- 正式なトレーニングが必要であるかどうかを判断します。必要な場合はスケジュールに追加します。

計画の遂行

必要なリソースを確保し、移行計画を実行します。

重要

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

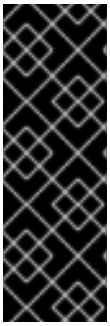
2.8. 重要データのバックアップおよびサーバー状態の確認

アプリケーションを移行する前に、以下の問題が発生する可能性があることを認識しておく必要があります。

- 移行によって一時フォルダーが削除される可能性があります。移行前に **data/content/** ディレクトリーに保存されたすべてのデプロイメントをバックアップし、移行後に復元する必要があります。この作業を怠ると、削除された内容が原因でサーバーが起動しないことがあります。
- 移行前に、開かれたトランザクションをすべて処理し、**data/tx-object-store/** トランザクションディレクトリーを削除します。
- **data/timer-service-data** にある永続タイマーデータをチェックし、アップグレード後も適用可能であるかを判断します。移行前に、このディレクトリーにある **deployment-*** ファイルをチェックし、使用されているタイマーを確認します。

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もバックアップするようにしてください。

2.9. RPM インストールの移行



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。そのため、JBoss EAP 7 に移行の際は、使用中の JBoss EAP インストールを新規マシンに移行することが推奨されます。

JBoss EAP RPM インストールを JBoss EAP 6 から JBoss EAP 7 に移行する際は、JBoss EAP 7 のインストール先となるマシンに既存の JBoss EAP RPM インストールがないことを確認してください。

RPM を使用して JBoss EAP 7 をインストールする場合は、JBoss EAP の『インストールガイド』を参照してください。

本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適用されますが、ZIP やインストーラーによるインストールの場合と比べると、一部の手順を変更して (JBoss EAP の開始方法など)、RPM インストールに合わせる必要があることがあります。

2.10. サービスとして実行するよう JBOSS EAP を移行

JBoss EAP 6 をサービスとして実行する場合は、必ず JBoss EAP『インストールガイド』の「サービスとして実行するよう **JBoss EAP** を設定」を読んで、JBoss EAP 7 で更新された設定手順を確認してください。

2.11. OPENSHIFT JDK 11 イメージへの移行

JBoss EAP for OpenShift の JDK 8 イメージから OpenShift JDK 11 イメージへの移行は、『JBOSS EAP FOR OPENSHIFT CONTAINER PLATFORM のスタートガイド』のJBOSS EAP FOR OPENSHIFT の JDK 11 イメージへの移行」を参照してください。

第3章 移行に役立つツール

3.1. RED HAT APPLICATION MIGRATION TOOLKIT を使用した移行のアプリケーションの分析

Red Hat Application Migration Toolkit (RHAMT) は、拡張およびカスタマイズ可能なルールベースのツールセットで、Java アプリケーションの移行を容易にします。RHAMT は移行予定のアプリケーションによって使用される API、技術、および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。

- 必要な移行変更の詳細
- 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

RHAMT を使用すると、JBoss EAP 6 アプリケーションを JBoss EAP 7 へ移行する前にこれらのアプリケーションのコードやアークテクチャーを分析できます。JBoss EAP 6 から JBoss EAP 7 への移行に対する RHAMT のルールセットは XML 記述子についてレポートし、JBoss EAP 7 への移行時に代替の設定に置き換える必要がある特定のアプリケーションコードおよびパラメーターについてもレポートします。

Red Hat Application Migration Toolkit を使用して JBoss EAP 6 アプリケーションを分析する方法の詳細は、『スタートガイド』を参照してください。

3.2. JBOSS SERVER MIGRATION TOOL を使用したサーバー設定の移行

JBoss Server Migration Tool を使用して、既存の設定を保持しながら JBoss EAP 7 の新機能や新設定が含まれるようにサーバー設定を更新することが推奨されます。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 を使用するとスタンドアロンサーバーおよび管理対象ドメインを以下の設定に移行できます。

JBoss EAP 7.2 への移行

JBoss Server Migration Tool は JBoss EAP 7.2 と同梱されるため、別にダウンロードやインストールを行う必要はありません。このツールは、JBoss EAP 6.4 以上から JBoss EAP 7.2 への移行をサポートします。このツールを実行するには、**EAP_HOME/bin** ディレクトリーにある **jboss-server-migration** スクリプトを実行します。ツールの設定および実行方法に関する詳細は、『Using the JBoss Server Migration Tool』を参照してください。
このバージョンの JBoss Server Migration Tool は サポート されるため、このバージョンを使用してサーバー設定を JBoss EAP 7.2 に移行することが推奨されます。

WildFly から JBoss EAP への移行

WildFly サーバーを JBoss EAP に移行するには、JBoss Server Migration Tool GitHub リポジトリー

から JBoss Server Migration Tool の最新のバイナリーディストリビューションをダウンロードする必要があります。これはオープンソースのスタンドアロンバージョンのツールで、複数のバージョンの WildFly サーバーから JBoss EAP への移行をサポートします。このツールのインストールおよび実行方法に関する詳細は、JBoss Server Migration Tool の『User Guide』を参照してください。

重要

JBoss Server Migration Tool のバイナリーディストリビューションはサポートされていません。以前のリリースの JBoss EAP から移行する場合は、サポートされるバージョンのツールを使用して、サーバー設定を JBoss EAP 7.2 に移行することが推奨されます。

第4章 サーバー設定の変更

4.1. RPM インストールの変更

JBoss EAP 6 では、RPM インストールのデフォルトのパスは **/usr/share/jbossas/** ディレクトリーでした。

JBoss EAP 7 は、Software Collections Library 慣例に従って構築されました。Software Collectionsとベースシステムインストールの競合を避けるため、Software Collections のルートディレクトリーは通常 **/opt/** にあります。**/opt/** ディレクトリーの使用は、Filesystem Hierarchy Standard (FHS) によって推奨されています。そのため、RPM インストールのデフォルトパスは JBoss EAP 7 では **/opt/rh/eap7/root/usr/share/wildfly/** に変更されました。

4.2. サーバー設定移行オプション

サーバー設定を JBoss EAP 6 から JBoss EAP 7 に移行するには、JBoss Server Migration Tool を使用するか、管理 CLI の **migrate** 操作 を使用して手作業で移行を行います。

JBoss Server Migration Tool

既存の設定を保持しながら、設定を更新して JBoss EAP 7 の新機能および設定を追加する場合は、JBoss Server Migration Tool を使用することが推奨されます。このツールの設定および実行方法に関する詳細は、『Using the JBoss Server Migration Tool』を参照してください。

管理 CLI の migrate 操作

管理 CLI の **migrate** 操作を使用して JBoss EAP 6 設定ファイルの **jacorb**、**messaging**、および **web** サブシステムを更新すると、新しいリリースで実行できるようにすることができますが、完全な JBoss EAP 7 の設定にはならないことに注意してください。例を以下に示します。

- この操作は、元の **remote** プロトコルおよびポート設定を JBoss EAP 7 で使用される新しい **http-remoting** およびポート設定に更新しません。
- 設定には、新しい JBoss EAP サブシステム、クラスター化されたシングルトンデプロイメントなどの機能、正常シャットダウンが含まれません。
- 設定には、バッチ処理などの新しい Java EE 7 の機能が含まれません。
- **migrate** 操作は **ejb3** サブシステムの設定を移行しません。起こりうる EJB の移行問題に関する詳細は、EJB サーバー設定の変更を参照してください。

migrate 操作を使用したサーバー設定の移行に関する詳細は、「管理 CLI の移行操作」を参照してください。

4.3. 管理 CLI の移行操作

管理 CLI を使用して、JBoss EAP 7 上で実行するよう JBoss EAP 6 のサーバー設定ファイルを更新することができます。管理 CLI では、**jacorb**、**messaging**、および **web** サブシステムを以前のリリースから新しい設定へ自動的に更新する **migrate** 操作を実行できます。また、 **jacorb**、**messaging**、および **web** サブシステムに **describe-migration** 操作を実行すると、移行を行う前に提案された移行設定の変更を確認することもできます。**cmp**、**jaxr**、および **threads** サブシステムの代替はないため、これらのサブシステムをサーバー設定から削除する必要があります。

重要

「サーバー設定移行オプション」で **migrate** 操作の制限を確認してください。既存の設定を保持しながら、設定を更新して JBoss EAP 7 の新機能および設定を追加する場合は、JBoss Server Migration Tool を使用することが推奨されます。このツールの設定および実行方法に関する詳細は、『Using the JBoss Server Migration Tool』を参照してください。

表4.1 サブシステムの移行および管理 CLI 操作

JBoss EAP 6 サブシステム	JBoss EAP 7 サブシステム	管理 CLI 操作
cmp	代替なし	remove
jacorb	iiop-openjdk	migrate
jaxr	代替なし	remove
messaging	messaging-activemq	migrate
threads	代替なし	remove
web	undertow	migrate

サーバーおよび管理 CLI の起動

以下の手順に従って、JBoss EAP 7 上で実行されるよう JBoss EAP 6 のサーバー設定を更新します。

1. 移行を始める前に、「重要データのバックアップおよびサーバー状態の確認」の内容を見直してください。ここには、サーバーを良好な状態にし、適切なファイルをバックアップするための重要な情報が記載されています。
2. JBoss EAP 6 の設定で JBoss EAP 7 のサーバーを起動します。
   a. JBoss EAP 7 サーバー設定ファイルをバックアップします。
   b. 以前のリリースの設定ファイルを JBoss EAP 7 ディレクトリーにコピーします。

```
$ cp EAP6_HOME/standalone/configuration/standalone-full.xml
EAP7_HOME/standalone/configuration
```

   c. JBoss EAP 7 のインストールディレクトリーへ移動し、**--start-mode=admin-only** 引数を使用してサーバーを起動します。

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

注記

サーバーを起動すると、以下の **org.jboss.as.controller.management-operation** エラーがサーバーログに記録されます。これらのエラーは予期されたエラーで、レガシーサブシステムの設定を削除するか JBoss EAP 7 に移行する必要があることを示しています。

- WFLYCTL0402: Subsystems [cmp] provided by legacy extension 'org.jboss.as.cmp' are not supported on servers running this version.Both the subsystem and the extension must be removed or migrated before the server will function.
- WFLYCTL0402: Subsystems [jacorb] provided by legacy extension 'org.jboss.as.jacorb' are not supported on servers running this version.Both the subsystem and the extension must be removed or migrated before the server will function.
- WFLYCTL0402: Subsystems [jaxr] provided by legacy extension 'org.jboss.as.jaxr' are not supported on servers running this version.Both the subsystem and the extension must be removed or migrated before the server will function.
- WFLYCTL0402: Subsystems [messaging] provided by legacy extension 'org.jboss.as.messaging' are not supported on servers running this version.Both the subsystem and the extension must be removed or migrated before the server will function.
- WFLYCTL0402: Subsystems [threads] provided by legacy extension 'org.jboss.as.threads' are not supported on servers running this version.Both the subsystem and the extension must be removed or migrated before the server will function.
- WFLYCTL0402: Subsystems [web] provided by legacy extension 'org.jboss.as.web' are not supported on servers running this version.Both the subsystem and the extension must be removed or migrated before the server will function.

3. 新しいターミナルを開いて、JBoss EAP 7 のインストールディレクトリーへ移動し、**--controller=remote://localhost:9990** 引数を使用して管理 CLI を開始します。

```
$ bin/jboss-cli.sh --connect --controller=remote://localhost:9990
```

JacORB、Messaging、および Web サブシステムの移行

1. 移行を行う前にサブシステムに追加した設定変更を確認するには、**describe-migration** 操作を実行します。
describe-migration 操作では以下の構文を使用します。

```
/subsystem=SUBSYSTEM_NAME:describe-migration
```

以下の例は、JBoss EAP 7 への移行時に JBoss EAP 6.4 の **standalone-full.xml** 設定ファイルに加えられる設定の変更を示しています。見やすくするため、エントリーは出力から削除されています。

例: describe-migration 操作

```
-
```

```
/subsystem=messaging:describe-migration
{
    "outcome" => "success",
    "result" => {
        "migration-warnings" => [],
        "migration-operations" => [
            {
                "operation" => "add",
                "address" => [("extension" => "org.wildfly.extension.messaging-activemq")],
                "module" => "org.wildfly.extension.messaging-activemq"
            },
            {
                "operation" => "add",
                "address" => [("subsystem" => "messaging-activemq")]
            },
            <!-- *** Entries removed for readability *** -->
            {
                "operation" => "remove",
                "address" => [("subsystem" => "messaging")]
            },
            {
                "operation" => "remove",
                "address" => [("extension" => "org.jboss.as.messaging")]
            }
        ]
    }
}
```

2. **migrate** 操作を実行し、サブシステムの設定を JBoss EAP 7 の代替サブシステムに移行します。この操作では以下の構文を使用します。

```
/subsystem=SUBSYSTEM_NAME:migrate
```

注記

messaging サブシステムの **describe-migration** および **migrate** 操作を使用すると、引数を渡してレガシークライアントによるアクセスを設定することができます。コマンド構文の詳細は、「Messaging サブシステムの移行および前方互換性」を参照してください。

3. このコマンドの結果を確認します。必ず、操作が正常に完了し、"migration-warning" エントリーがないことを確認してください。これは、サブシステムの移行設定が完了したことを意味します。

例: 警告のない成功した migrate 操作

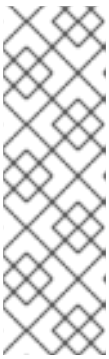
```
/subsystem=messaging:migrate
{
    "outcome" => "success",
    "result" => {"migration-warnings" => []}
}
```

サーバー設定の移行が正常に完了したにも関わらず、すべての要素と属性を移行できなかった場合、ログに "migration-warnings" エントリーが表示されます。"migration-warnings" が示す提

案に従い、追加の管理 CLI コマンドを実行してこれらの設定を編集する必要があります。"migration-warnings" を返す **migrate** 操作の例を以下に示します。

例: 警告のある migrate 操作

```
/subsystem=messaging:migrate
{
    "outcome" => "success",
    "result" => {"migration-warnings" => [
        "WFLYMSG0080: Could not migrate attribute group-address from resource [
    (\"subsystem\" => \"messaging-activemq\"),
    (\"server\" => \"default\"),
    (\"broadcast-group\" => \"groupB\")
]. Use instead the socket-binding attribute to configure this broadcast-group.",
        "WFLYMSG0080: Could not migrate attribute group-port from resource [
    (\"subsystem\" => \"messaging-activemq\"),
    (\"server\" => \"default\"),
    (\"broadcast-group\" => \"groupB\")
]. Use instead the socket-binding attribute to configure this broadcast-group.",
        "WFLYMSG0080: Could not migrate attribute local-bind-address from resource [
    (\"subsystem\" => \"messaging-activemq\"),
    (\"server\" => \"default\"),
    (\"broadcast-group\" => \"groupA\")
]. Use instead the socket-binding attribute to configure this broadcast-group.",
        "WFLYMSG0080: Could not migrate attribute local-bind-port from resource [
    (\"subsystem\" => \"messaging-activemq\"),
    (\"server\" => \"default\"),
    (\"broadcast-group\" => \"groupA\")
]. Use instead the socket-binding attribute to configure this broadcast-group.",
        "WFLYMSG0080: Could not migrate attribute group-address from resource [
    (\"subsystem\" => \"messaging-activemq\"),
    (\"server\" => \"default\"),
    (\"broadcast-group\" => \"groupA\")
]. Use instead the socket-binding attribute to configure this broadcast-group.",
        "WFLYMSG0080: Could not migrate attribute group-port from resource [
    (\"subsystem\" => \"messaging-activemq\"),
    (\"server\" => \"default\"),
    (\"broadcast-group\" => \"groupA\")
]. Use instead the socket-binding attribute to configure this broadcast-group."
    ]}
}
```

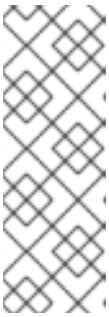


注記

各サブシステムの **migrate** および **describe-migration** 警告のリストは、本ガイドの最後にあるリファレンス資料 に記載されています。

- jacorb サブシステム移行操作の警告
- Messaging サブシステム移行操作の警告
- Web サブシステム移行操作の警告

4. サーバー設定ファイルを確認し、拡張、サブシステム、および名前空間が更新され、既存のサブシステム設定が JBoss EAP 7 に移行されたことを確認します。



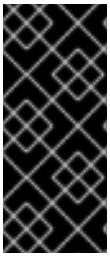
注記

この手順は、以下のコマンドを使用して **jacorb**、**messaging**、および **web** の各サブシステムに対して繰り返す必要があります。

```
/subsystem=jacorb:migrate
/subsystem=messaging:migrate
/subsystem=web:migrate
```

5. **cmp**、**jaxr**、および **threads** サブシステムおよび拡張をサーバー設定から削除します。
管理 CLI プロンプトで以下のコマンドを実行し、廃止された **cmp**、**jaxr**、および **threads** サブシステムを削除します。

```
/subsystem=cmp:remove
/extension=org.jboss.as.cmp:remove
/subsystem=jaxr:remove
/extension=org.jboss.as.jaxr:remove
/subsystem=threads:remove
/extension=org.jboss.as.threads:remove
```



重要

サーバーを再起動して通常操作を行う前に **messaging**、**jacorb**、および **web** サブシステムを移行し、**cmp**、**jaxr**、および **threads** 拡張およびサブシステムを削除する必要があります。この作業の完了前にサーバーを再起動する必要がある場合は、サーバー起動のコマンドラインで **--start-mode=admin-only** 引数を使用するようにしてください。これにより、サーバーの変更を継続できます。

4.4. ロギングの変更

4.4.1. ロギングメッセージの接頭辞の変更

ログメッセージには、メッセージを報告するサブシステムのプロジェクトコードが接頭辞として付けられます。JBoss EAP 7 では、すべてのログメッセージの接頭辞が変更になりました。

JBoss EAP 7 で使用されるログメッセージの新しいプロジェクトコード接頭辞の完全リストは、JBoss EAP『開発ガイド』の「**JBoss EAP** で使用されるプロジェクトコード」を参照してください。

4.4.2. ルートロガーコンソールハンドラーの変更

JBoss EAP 7.0 のルートロガーには、すべてのドメインサーバープロファイルと **standalone-full-ha** プロファイル以外のデフォルトのスタンドアロンプロファイル用のコンソールログハンドラーが含まれていました。JBoss EAP 7.1 より、ルートロガーには管理対象ドメインプロファイルのコンソールログハンドラーは含まれていません。ホストコントローラーとプロセスコントローラーはデフォルトでコンソールにログが記録されます。JBoss EAP 7.0 で提供された機能を実現するには、JBoss EAP 『設定ガイド』の「**Console** ログハンドラーの設定」を参照してください。

4.5. WEB サーバー設定の変更

4.5.1. Undertow による Web サブシステムの置換

JBoss EAP 7 の web サーバーは、JBoss Web から Undertow に変更になりました。そのため、**web** サブシステム設定を新しい JBoss EAP 7 **undertow** サブシステム設定に移行する必要があります。

- サーバー設定ファイルの **urn:jboss:domain:web:2.2** サブシステム設定ネームスペースは **urn:jboss:domain:undertow:7.0** ネームスペースに置き換えられました。
- **EAP_HOME/modules/system/layers/base/** にあった **org.jboss.as.web** 拡張モジュールは、**org.wildfly.extension.undertow** 拡張モジュールに置き換えられました。

管理 CLI **migrate** 操作を使うと、 **web** サブシステムをサーバー設定ファイル内の **undertow** に移行することができます。ただし、この操作では JBoss Web サブシステムのすべての設定が移行できるわけではないことに注意してください。"migration-warning" エントリーが表示される場合は、追加の管理 CLI コマンドを実行して設定を Undertow に移行する必要があります。管理 CLI **migrate** 操作についての詳細情報は、「管理 CLI の移行操作」を参照してください。

以下の例は、JBoss EAP 6.4 のデフォルトの **web** サブシステム設定を示しています。

```
<subsystem xmlns="urn:jboss:domain:web:2.2" default-virtual-server="default-host" native="false">
    <connector name="http" protocol="HTTP/1.1" scheme="http" socket-binding="http"/>
    <virtual-server name="default-host" enable-welcome-root="true">
        <alias name="localhost"/>
        <alias name="example.com"/>
    </virtual-server>
</subsystem>
```

以下の例は、JBoss EAP 7.2 のデフォルトの **undertow** サブシステム設定を示しています。

```
<subsystem xmlns="urn:jboss:domain:undertow:7.0" 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>
    ...
</subsystem>
```

4.5.2. JBoss Web リライト条件の移行

管理 CLI の **migrate** 操作はリライト条件を自動的に移行できません。"migration-warnings" として報告され、手作業で移行する必要があります。Undertow の述語属性およびハンドラーを使用すると (「Undertow Predicates Attributes and Handlers 」を参照)、JBoss EAP 7 で同等の設定を作成できます。

以下の例は、**rewrite** 設定を含む JBoss EAP 6 の **web** サブシステム設定を示しています。

```
<subsystem xmlns="urn:jboss:domain:web:2.2" default-virtual-server="default" native="false">
    <virtual-server name="default" enable-welcome-root="true">
        <alias name="localhost"/>
        <rewrite name="test" pattern="(.*)/toberewritten/(.*)" substitution="$1/rewritten/$2" flags="NC"/>
```

```
        <rewrite name="test2" pattern="(.*)" substitution="-" flags="F">
            <condition name="get" test="%{REQUEST_METHOD}" pattern="GET"/>
            <condition name="andCond" test="%{REQUEST_URI}" pattern=".*index.html" flags="NC"/>
        </rewrite>
    </virtual-server>
</subsystem>
```

「管理 CLI の移行操作」にある手順にしたがってサーバーと管理 CLI を開始し、以下のコマンドを使用して **web** サブシステム設定ファイルを移行します。

```
/subsystem=web:migrate
```

上記の設定で **migrate** 操作を実行すると、以下の "migration-warnings" がレポートされます。

```
/subsystem=web:migrate
{
    "outcome" => "success",
    "result" => {"migration-warnings" => [
        "WFLYWEB0002: Could not migrate resource {
    \"pattern\" => \"(.*)\",
    \"substitution\" => \"-\",
    \"flags\" => \"F\",
    \"operation\" => \"add\",
    \"address\" => [
        (\"subsystem\" => \"web\"),
        (\"virtual-server\" => \"default-host\"),
        (\"rewrite\" => \"test2\")
    ]
}",
        "WFLYWEB0002: Could not migrate resource {
    \"test\" => \"%{REQUEST_METHOD}\",
    \"pattern\" => \"GET\",
    \"flags\" => undefined,
    \"operation\" => \"add\",
    \"address\" => [
        (\"subsystem\" => \"web\"),
        (\"virtual-server\" => \"default-host\"),
        (\"rewrite\" => \"test2\"),
        (\"condition\" => \"get\")
    ]
}",
        "WFLYWEB0002: Could not migrate resource {
    \"test\" => \"%{REQUEST_URI}\",
    \"pattern\" => \".*index.html\",
    \"flags\" => \"NC\",
    \"operation\" => \"add\",
    \"address\" => [
        (\"subsystem\" => \"web\"),
        (\"virtual-server\" => \"default-host\"),
        (\"rewrite\" => \"test2\"),
        (\"condition\" => \"andCond\")
    ]
}"
    ]}
}
```

サーバー設定ファイルを確認すると、**undertow** サブシステムが以下の設定になっています。

注記

リライト設定は削除されています。

```
<subsystem xmlns="urn:jboss:domain:undertow:7.0" 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="http" socket-binding="http"/>
        <https-listener name="https" socket-binding="https" security-realm="ApplicationRealm" enable-http2="true"/>
        <host name="default-host" alias="localhost, example.com">
            <location name="/" handler="welcome-content"/>
        </host>
    </server>
    <servlet-container name="default">
        <jsp-config/>
    </servlet-container>
    <handlers>
        <file name="welcome-content" path="${jboss.home.dir}/welcome-content"/>
    </handlers>
</subsystem>
```

管理 CLI を使用してフィルターを作成し、**undertow** サブシステムのリライト設定を置き換えます。各コマンドで "{"outcome" ⇒ "success"}" が表示されるはずです。

```
# Create the filters
/subsystem=undertow/configuration=filter/expression-filter="test1":add(expression="path('(.*)/toberewritten/(.*)') -> rewrite('$1/rewritten/$2')")
/subsystem=undertow/configuration=filter/expression-filter="test2":add(expression="method('GET') and path('.*index.html') -> response-code(403)")

# Add the filters to the default server
/subsystem=undertow/server=default-server/host=default-host/filter-ref="test1":add
/subsystem=undertow/server=default-server/host=default-host/filter-ref="test2":add
```

更新されたサーバー設定ファイルを確認します。JBoss Web サブシステムは完全に移行され、**undertow** サブシステムに設定されます。

```
<subsystem xmlns="urn:jboss:domain:undertow:7.0" 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="http" socket-binding="http"/>
        <https-listener name="https" socket-binding="https" security-realm="ApplicationRealm" enable-http2="true"/>
        <host name="default-host" alias="localhost, example.com">
            <location name="/" handler="welcome-content"/>
            <filter-ref name="test1"/>
            <filter-ref name="test2"/>
        </host>
    </server>
```

```
    <servlet-container name="default">
        <jsp-config/>
    </servlet-container>
    <handlers>
        <file name="welcome-content" path="${jboss.home.dir}/welcome-content"/>
    </handlers>
    <filters>
        <expression-filter name="test1" expression="path('(.*)/toberewritten/(.*)') ->
rewrite('$1/rewritten/$2')"/>
        <expression-filter name="test2" expression="method('GET') and path('.*index.html') -> response-
code(403)"/>
    </filters>
</subsystem>
```

管理 CLI を使用してフィルターとハンドラーを設定する方法については、JBoss EAP 7 『設定ガイド』の「**Web** サーバーの設定」を参照してください。

4.5.3. JBoss Web システムプロパティーの移行

以前のリリースの JBoss EAP では、システムプロパティーを使用して JBoss Web のデフォルトの動作を変更することが可能でした。Undertow で同じ動作を設定する方法は、「JBoss Web システムプロパティー移行のリファレンス」を参照してください。

4.5.4. アクセスログヘッダーパターンの更新

JBoss EAP 6.4 から JBoss EAP 7 に移行するときに、アクセスログに予期される "Referer" および "User-agent" 値が書き込まれないことに気付くかもしれません。これは、JBoss EAP 6.4 に含まれていた JBoss Web は **access-log** で **%{headername}i** パターンを使用して受信ヘッダーをログに記録したためです。

例: JBoss EAP 6.4 でのアクセスログ形式

```
<access-log pattern="%h %l %u %t &quot;%T sec&quot; &quot;%r&quot; %s %b &quot;%
{Referer}i&quot; &quot;%{User-agent}i&quot;"/>
```

JBoss EAP 7 では Undertow が使用されるようになったため、受信ヘッダーのパターンは **%{i,headername}** に変更になりました。

例: JBoss EAP 7 でのアクセス形式ヘッダー

```
<access-log pattern="%h %l %u %t &quot;%T sec&quot; &quot;%r&quot; %s %b &quot;%
{i,Referer}&quot; &quot;%{i,User-Agent}&quot;"/>
```

4.5.5. グローバルバルブの移行

以前のリリースの JBoss EAP ではバルブがサポートされました。バルブとは、リクエストの変更や追加処理を実行するために、サーブレットフィルターの前にアプリケーションのリクエスト処理パイプラインへ挿入されるカスタムクラスのことです。

- グローバルバルブは、デプロイされたすべてのアプリケーションのリクエスト処理パイプラインへ挿入され、サーバー設定ファイルで設定されます。
- オーセンティケーターバルブはリクエストのクレデンシャルを認証します。

- カスタムアプリケーションバルブは、**org.apache.catalina.valves.ValveBase** クラスを拡張して作成され、**jboss-web.xml** 記述子ファイルの **<valve>** 要素で設定されます。これらのバルブは手作業で移行する必要があります。

この項では、グローバルバルブの移行方法について説明します。カスタムおよびオーセンティケーターバルブの移行については、本ガイドの「カスタムアプリケーションバルブの移行」を参照してください。

JBoss EAP 7 で JBoss Web の代わりに導入された Undertow はグローバルバルブをサポートしませんが、Undertow ハンドラーを使用すると同様の機能を実現できます。Undertow には共通の機能を提供する複数のビルトインハンドラーが含まれています。また、カスタムハンドラーを作成する機能も含まれ、カスタムバルブの機能を置き換えるために使用することができます。

アプリケーションがバルブを使用する場合、JBoss EAP 7 へ移行するときに適切な Undertow ハンドラーコードに置き換え、同様の機能を実現する必要があります。

ハンドラーの設定方法に関する詳細は、JBoss EAP 7『設定ガイド』の「ハンドラーの設定」を参照してください。

フィルターの設定方法に関する詳細は、JBoss EAP 7『設定ガイド』の「フィルターの設定」を参照してください。

JBoss Web バルブの移行
以下の表では、JBoss EAP の前リリースで JBoss Web が提供していたバルブとそれに対応する Undertow ビルトインハンドラーを示しています。JBoss Web バルブは、**org.apache.catalina.valves** パッケージにあります。

表4.2 バルブとハンドラーのマッピング

バルブ	ハンドラー
AccessLogValve	io.undertow.server.handlers.accesslog.AccessLogHandler
CrawlerSessionManagerValve	io.undertow.servlet.handlers.CrawlerSessionManagerHandler
ExtendedAccessLogValve	io.undertow.server.handlers.accesslog.AccessLogHandler
JDBCAccessLogValve	手順については **JDBCAccessLogValve** の手動移行の手順を参照してください。
RemoteAddrValve	io.undertow.server.handlers.IPAddressAccessControlHandler
RemoteHostValve	io.undertow.server.handlers.AccessControlListHandler
RemoteIpValve	io.undertow.server.handlers.ProxyPeerAddressHandler
RequestDumperValve	io.undertow.server.handlers.RequestDumpingHandler
RewriteValve	これらのバルブを手作業で移行する手順は、「JBoss Web リライト条件の移行」を参照してください。
StuckThreadDetectionValve	io.undertow.server.handlers.StuckThreadDetectionHandler

管理 CLI の **migrate** 操作を使用すると、以下の基準を満たすグローバルバルブを自動的に移行できます。

- 前述の表にリストされている、手動処理の必要がないバルブに限定されます。
- サーバー設定ファイルの **web** サブシステムに定義されている必要があります。

管理 CLI **migrate** 操作についての詳細情報は、「管理 CLI の移行操作」を参照してください。

JDBCAccessLogValve の手動移行の手順
org.apache.catalina.valves.JDBCAccessLogValve バルブはルールの例外で、**io.undertow.server.handlers.JDBCLogHandler** へ自動的に移行することができません。以下の手順に従って、次のバルブ例を移行します。

```
<valve name="jdbc" module="org.jboss.as.web" class-
name="org.apache.catalina.valves.JDBCAccessLogValve">
   <param param-name="driverName" param-value="com.mysql.jdbc.Driver" />
   <param param-name="connectionName" param-value="root" />
   <param param-name="connectionPassword" param-value="password" />
   <param param-name="connectionURL" param-value="jdbc:mysql://localhost:3306/wildfly?
zeroDateTimeBehavior=convertToNull" />
   <param param-name="format" param-value="combined" />
</valve>
```

1. ログエントリーを保存するデータベースのドライバーモジュールを作成します。
2. データベースのデータソースを設定し、ドライバーを **datasources** サブシステムの利用可能なドライバーのリストに追加します。

```
<datasources>
   <datasource jndi-name="java:jboss/datasources/accessLogDS" pool-
name="accessLogDS" enabled="true" use-java-context="true">
      <connection-url>jdbc:mysql://localhost:3306/wildfly?
zeroDateTimeBehavior=convertToNull</connection-url>
      <driver>mysql</driver>
      <security>
         <user-name>root</user-name>
         <password>Password1!</password>
      </security>
   </datasource>
   ...
   <drivers>
      <driver name="mysql" module="com.mysql">
         <driver-class>com.mysql.jdbc.Driver</driver-class>
      </driver>
   ...
   </drivers>
</datasources>
```

3. 式 **jdbc-access-log(datasource=DATASOURCE_JNDI_NAME)** を使用して、**undertow** サブシステムの **expression-filter** を設定します。

```
<filters>
   <expression-filter name="jdbc-access" expression="jdbc-access-
log(datasource='java:jboss/datasources/accessLogDS')" />
```

```
    ...
</filters>
```

4.5.6. Set-Cookie の動作変更

RFC2109 や RFC2965 など、**Set-Cookie** HTTP 応答ヘッダー構文の以前の仕様では、クッキー値が引用符で囲まれている場合はクッキー値に空白やその他の区切り文字を使用することができました。JBoss EAP 6.4 の JBoss Web は以前の仕様に準拠し、区切り文字が含まれるクッキー値を自動的に引用符で囲みました。

Set-Cookie HTTP 応答ヘッダー構文の **RFC6265** 仕様には、**Set-Cookie** 応答ヘッダーのクッキー値は特定の文法制約に準拠しなければならないと記載されています。たとえば、US-ASCII 文字でなければならず、CTRL (コントロール)、空白、2 重引用符、コンマ、セミコロン、またはバックスラッシュ文字を含んではいけません。

JBoss EAP 7.0 の累積パッチ Red Hat JBoss Enterprise Application Platform 7.0 Update 08 以前では、Undertow はこれらの無効な文字を制限せず、除外された文字が含まれたクッキーを引用符で囲みません。この累積パッチまたはこれ以降の累積パッチを適用し、**io.undertow.cookie.DEFAULT_ENABLE_RFC6265_COOKIE_VALIDATION** システムプロパティーを **true** に設定すると、RFC6265 に準拠するクッキーの検証を有効にすることができます。

JBoss EAP 7.1 より、Undertow はデフォルトで RFC6265 に準拠するクッキーの検証を有効化しないようになりました。除外された文字が含まれたクッキーを引用符で囲みます。JBoss EAP 7.1 より、**io.undertow.cookie.DEFAULT_ENABLE_RFC6265_COOKIE_VALIDATION** システムプロパティーを使用して RFC6265 に準拠するクッキーの検証を有効にすることはできません。代わりに、**rfc6265-cookie-validation** リスナー属性を **true** に設定して HTTP、HTTPS、または AJP リスナーの RFC6265 に準拠するクッキーの検証を有効にします。この属性のデフォルト値は **false** です。以下の例は、HTTP リスナーの RFC6265 に準拠するクッキーの検証を有効にします。

```
/subsystem=undertow/server=default-server/http-listener=default:write-attribute(name=rfc6265-cookie-validation,value=true)
```

4.5.7. HTTP メソッド呼び出しの挙動変更

JBoss Web が web サーバーとして含まれていた JBoss EAP 6.4 では、デフォルトで HTTP **TRACE** メソッド呼び出しが許可されていました。

JBoss EAP 7 で JBoss Web の代わりに web サーバーとして導入された Undertow は、デフォルトでは HTTP **TRACE** メソッド呼び出しを許可しません。この設定は、**undertow** サブシステムで **http-listener** 要素の **disallowed-methods** 属性を使用して設定されます。この設定を確認するには、以下の **read-resource** コマンドの出力を確認します。**disallowed-methods** 属性の値が **["TRACE"]** であることに注目してください。

```
/subsystem=undertow/server=default-server/http-listener=default:read-resource
{
    "outcome" => "success",
    "result" => {
        "allow-encoded-slash" => false,
        "allow-equals-in-cookie-value" => false,
        "allow-unescaped-characters-in-url" => false,
        "always-set-keep-alive" => true,
        "buffer-pipelined-data" => false,
        "buffer-pool" => "default",
        "certificate-forwarding" => false,
```

```
        "decode-url" => true,
        "disallowed-methods" => ["TRACE"],
         ...
    }
}
```

JBoss EAP 7 およびそれ以降のバージョンで HTTP **TRACE** メソッド呼び出しを有効にするには、以下のコマンドを実行して **disallowed-methods** 属性リストから "TRACE" エントリーを削除する必要があります。

```
/subsystem=undertow/server=default-server/http-listener=default:list-remove(name=disallowed-methods,value="TRACE")
```

read-resource コマンドを再度実行すると、**TRACE** メソッド呼び出しが許可されないメソッドのリストから削除されたことが確認できます。

```
/subsystem=undertow/server=default-server/http-listener=default:read-resource
{
    "outcome" => "success",
    "result" => {
        "allow-encoded-slash" => false,
        "allow-equals-in-cookie-value" => false,
        "allow-unescaped-characters-in-url" => false,
        "always-set-keep-alive" => true,
        "buffer-pipelined-data" => false,
        "buffer-pool" => "default",
        "certificate-forwarding" => false,
        "decode-url" => true,
        "disallowed-methods" => [],
         ...
    }
}
```

HTTP メソッドのデフォルト動作に関する詳細は、JBoss EAP『設定ガイド』の「**HTTP** メソッドのデフォルトの動作」を参照してください。

4.5.8. デフォルトの Web モジュール動作の変更

JBoss EAP 7.0 では、mod_cluster の web アプリケーションのルートコンテキストは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 の場所も明示的に除外しない限り、mod_cluster ロードバランサーに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
```

デフォルトのウェルカム Web アプリケーションを設定する方法の詳細は、JBoss EAP『開発ガイド』の「デフォルトの**Welcome Web** アプリケーションの設定」を参照してください。

4.5.9. Undertow サブシステムのデフォルト設定の変更

JBoss EAP 7.2 よりも前のリリースでは、**undertow** サブシステムのデフォルト設定に 2 つの応答ヘッダーフィルターが含まれ、これらのフィルターは **default-host** によって各 HTTP 応答に追加されていました。

- **JBoss-EAP/7** に設定されていた **Server**
- **Undertow/1** に設定されていた **X-Powered-By**

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 7.2 **undertow** サブシステムの新しいデフォルト設定を示しています。

```
<subsystem xmlns="urn:jboss:domain:undertow:7.0" 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>
```

4.6. JGROUPS サーバー設定の変更

4.6.1. JGroups はデフォルトでプライベートネットワークインターフェースを使用

JBoss EAP 6 のデフォルト設定では、JGroups はサーバー設定ファイルの **<interfaces>** セクションに定義された **public** インターフェースを使用しました。

専用のネットワークインターフェースを使用することが推奨されるため、JBoss EAP 7 では JGroups はデフォルトでサーバー設定ファイルの **<interfaces>** セクションに定義された新しい **private** インターフェースを使用します。

4.6.2. JGroups チャネルの変更

JGroups は JGroups チャネルで HA サービスのグループ通信サポートを提供します。JBoss EAP 7 では、サーバー設定ファイルの **jgroups** サブシステムに **<channel>** 要素が導入されました。管理 CLI を使用して JGroups チャネル設定を追加、削除、および変更できます。

JGroups の設定方法の詳細は、JBoss EAP『設定ガイド』の「**JGroups** を用いたクラスター通信」を参照してください。

4.7. INFINISPAN サーバー設定の変更

4.7.1. Infinispan のデフォルトキャッシュ設定の変更

JBoss EAP 6 では、Web セッションレプリケーションおよび EJB レプリケーションのデフォルトのクラスター化キャッシュはレプリケートされた **ASYNC** キャッシュでしたが、これが JBoss EAP 7 では変更になりました。これは JBoss EAP 7 で変更になりました。デフォルトのクラスター化されたキャッシュは **ASYNC** キャッシュを分散するようになりました。レプリケートされたキャッシュは、デフォルトでは設定されなくなりました。レプリケートされたキャッシュを追加して、デフォルトに設定する方法は、JBoss EAP 『設定ガイド』 の「キャッシュモードの設定」を参照してください。

これは、新しい JBoss EAP 7 のデフォルト設定を使用する場合のみ影響します。JBoss EAP 6 から設定を移行する場合は、**infinispan** サブシステムの設定は保持されます。

4.7.2. Infinispan のキャッシュストラテジーの変更

ASYNC キャッシュストラテジーの動作が JBoss EAP 7 では変更になりました。

JBoss EAP 6 では、**ASYNC** キャッシュの読み取りにロックはありませんでした。ブロックは発生しませんでしたが、フェイルオーバーなどで陳腐データのダーティーリードが発生する傾向にありました。これは、リクエストの完了前に同じユーザーの次のリクエストを開始できたためです。これは、クラスタートポロジーの変更がセッションアフィニティーに影響し、容易にデータが陳腐化することがあるため、分散モードを使用するときは許可されません。

JBoss EAP 7 では、**ASYNC** キャッシュの読み取りにロックが必要になりました。レプリケーションが終了するまで同じユーザーの新しいリクエストをブロックするようになったため、ダーティーリードが発生しないようになりました。

4.7.3. パッシベーションに対するカスタムステートフルセッション bean の設定

JBoss EAP 7.1 以上のリリースで、パッシベーションに対してカスタムステートフルセッション bean (SFSB) を設定する場合は、以下の制限に注意してください。

- **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 6.4 ではイーガー (eager) パッシベーションがサポートされ、**idle-timeout** の値を基にパッシベーションが実行されました。JBoss EAP 7.1 以上のリリースではレイジー (lazy) パッシベーションがサポートされ、**max-size** しきい値に達するとパッシベーションが行われます。

- JBoss EAP 7.1 以上のリリースでは、EJB 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るチャネルの実際のクラスター名によって判断されます。

- 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。

- EJB キャッシュ設定でエビクションまたはエクスパレーションを設定しないでください。

  - エビクションは、**ejb3** サブシステムの **passivation-store** の **max-size** 属性を使用して設定してください。

  - エクスパレーションは、SFSB Java ソースコードの **@StatefulTimeout** アノテーションを使用するか、**ejb-jar.xml** ファイルに **stateful-timeout** 値を指定して設定してください。

4.7.4. 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 の動作が JBoss EAP 7.1 以上のバージョンで変更になったため、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はバッチモードで実行するか、特別なヘッダーを使用して実行する必要があります。この動作の変更は、JBoss EAP サーバーの管理に使用されたすべてのツールにも影響があります。

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。

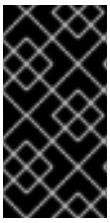
```
batch
/subsystem=infinispan/cache-container=my:add()
/subsystem=infinispan/cache-container=my/transport=jgroups:add()
/subsystem=infinispan/cache-container=my/invalidation-cache=mycache:add(mode=SYNC)
run-batch
```

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。これによりサービスが再起動され、操作が適用されますが、このサービスが再起動するまで一部のサービスが停止する可能性があります。

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。

4.8. EJB サーバー設定の変更

ejb3 サブシステムの **migrate** 操作はないため、管理 CLI の **migrate** 操作を使用して他の既存の JBoss EAP 6.4 設定をアップグレードする場合は、**ejb3** サブシステムの設定は移行されないことに注意してください。JBoss EAP 7 での **ejb3** サブシステムの設定は、JBoss EAP 6.4 とは若干異なるため、EJB アプリケーションをデプロイしたときにサーバーログに例外が記録されることがあります。



重要

JBoss Server Migration Tool を使用してサーバー設定を更新すると、**ejb3** サブシステムは適切に設定され、EJBが アプリケーションのデプロイ時に問題は発生しないはずです。このツールの設定および実行方法に関する詳細は、『Using the JBoss Server Migration Tool』を参照してください。

DuplicateServiceException

以下の **DuplicateServiceException** は、JBoss EAP 7.1 のキャッシングの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
```

キャッシュを再設定してこのエラーを解決する必要があります。

1. 「サーバーおよび管理 CLI の起動」の手順に従います。
2. 以下のコマンドを実行して、**ejb3** サブシステムのキャッシングを再設定します。

```
/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])
```

-

4.9. メッセージングサーバー設定の変更

JBoss EAP 7 では、JMS サポートプロバイダーが HornetQ から ActiveMQ Artemis に変更になりました。ここでは、設定と関連するメッセージングデータの移行方法を説明します。

4.9.1. メッセージングサブシステムサーバー設定の変更

EAP_HOME/modules/system/layers/base/ にあった **org.jboss.as.messaging** モジュール拡張は、**org.wildfly.extension.messaging-activemq** 拡張モジュールに置き換えられました。

urn:jboss:domain:messaging:3.0 サブシステム設定ネームスペースは **urn:jboss:domain:messaging-activemq:4.0** ネームスペースに置き換えられました。

管理モデル
ほとんどの場合で、要素名と属性名は以前のリリースとできる限り同じになるよう努力しています。以下の表は変更の一部を表しています。

表4.3 メッセージング属性のマッピング

HornetQ での名前	ActiveMQ での名前
hornetq-server	server
hornetq-serverType	serverType
connectors	connector
discovery-group-name	discovery-group

新しい **messaging-activemq** サブシステム上で呼び出される管理操作は、**/subsystem=messaging/hornetq-server=** から **/subsystem=messaging-activemq/server=** に変更になりました。

migrate 操作を呼び出すと、既存の JBoss EAP 6 の **messaging** サブシステム設定を JBoss EAP 7 の **messaging-activemq** サブシステムに移行できます。

```
/subsystem=messaging:migrate
```

migrate 操作を実行する前に **describe-migration** 操作を呼び出して、既存の JBoss EAP 6 の **messaging** サブシステム設定から JBoss EAP 7 サーバーの **messaging-activemq** サブシステムへの移行を実行する管理操作のリストを確認することができます。

```
/subsystem=messaging:describe-migration
```

migrate および **describe-migration** 操作は、自動的に移行されないリソースや属性の **migration-warnings** のリストも表示します。

Messaging サブシステムの移行および前方互換性
messaging サブシステムの **describe-migration** および**migrate** 操作は、追加の設定引数を提供します。メッセージングを設定してレガシー JBoss EAP 6 クライアントが JBoss EAP 7 サーバーに接続できるようにするには、以下のようにブール値の **add-legacy-entries** 引数を **describe-migration** または

migrate 操作に追加します。

```
/subsystem=messaging:describe-migration(add-legacy-entries=true)
/subsystem=messaging:migrate(add-legacy-entries=true)
```

ブール値の引数 **add-legacy-entries** が **true** に設定されていると、**messaging-activemq** サブシステムが **legacy-connection-factory** リソースを作成し、**legacy-entries** を **jms-queue** および **jms-topic** リソースに追加します。

ブール値の引数 **add-legacy-entries** が **false** に設定されていると、**messaging-activemq** サブシステムにはレガシーリソースが作成されず、レガシー JMS クライアントは JBoss EAP 7 サーバーと通信できません。これがデフォルト値になります。

前方互換性および後方互換性に関する詳細は、JBoss EAP『Configuring Messaging』の「**Backward and Forward Compatibility**」を参照してください。

管理 CLI の **migrate** および **describe-migration** 操作の詳細は、「管理 CLI の移行操作」を参照してください。

forward-when-no-consumers 属性の動作変更
JBoss EAP 7 では、**forward-when-no-consumers** 属性の動作が変更になりました。

JBoss EAP 6 では、**forward-when-no-consumers** が **false** に設定され、クラスターにコンシューマーがない場合に、メッセージはクラスターのすべてのノードへ再分散されました。

この挙動は JBoss EAP 7 では変更になりました。**forward-when-no-consumers** が **false** に設定され、クラスターにコンシューマーがない場合は、メッセージは再分散されません。代わりに、それらが送信される元のノードに保管されます。

デフォルトのクラスター負荷分散ポリシーの変更
JBoss EAP 7 では、デフォルトのクラスター負荷分散ポリシーが変更になりました。

JBoss EAP 6 ではデフォルトの負荷分散ポリシーは **STRICT** と似ており、レガシーの **forward-when-no-consumers** パラメーターを **true** に設定した場合と同様でした。JBoss EAP 7 では、デフォルトは **ON_DEMAND** になり、レガシーの **forward-when-no-consumers** パラメーターを **false** に設定した場合と同様になります。これらの設定の詳細は、JBoss EAP『Configuring Messaging』の「**Cluster Connection Attributes**」を参照してください。

Messaging サブシステムの XML 設定
新しい **messaging-activemq** サブシステムにより、XML の設定が大幅に変更になり、他の JBoss EAP サブシステムとより一貫性のある XML スキームが提供されるようになりました。

新しい **messaging-activemq** サブシステムに準拠するために JBoss EAP **messaging** サブシステムの XML 設定を変更しないでください。この代わりに、レガシーサブシステムの **migrate** 操作を呼び出してください。この操作は、実行の一部として新しい **messaging-activemq** の XML 設定を書き込みます。

4.9.2. メッセージングデータの移行

以下の方法のいずれかを使用してメッセージングデータを以前のリリースの JBoss EAP から現在のリリースに移行します。

- ファイルベースのメッセージングシステムでは、エクスポートおよびインポートメソッドを使用してメッセージングデータを JBoss EAP 6.4 および以前の JBoss EAP 7.x リリースから JBoss EAP 7.2 に移行できます。この方法では、これまで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 の **import-journal** 操作を使用してインポートします。これは、ファイルベースのメッセージングシステムでのみ使用できることに注意してください。

- JMS ブリッジを設定して、メッセージングデータを JBoss EAP 6.4 から JBoss EAP 7.2 に移行できます。この方法は、ファイルベースのメッセージングシステムと JDBC メッセージングシステムの両方に使用できます。

JMS サポートプロバイダーが HornetQ から ActiveMQ Artemis に変更になったため、JBoss EAP 7.0 以上のリリースではメッセージングデータの形式と場所が変更になりました。6.4 から 7.x リリースで変更になったメッセージングデータフォルダー名と場所の詳細は、「メッセージングフォルダー名のマッピング」を参照してください。

4.9.2.1. エクスポートおよびインポートを使用したメッセージングデータの移行

この方法では、以前のリリースのメッセージングデータを XML ファイルへエクスポートし、**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. メッセージングデータの XML ファイルへのエクスポート
   - メッセージングデータの JBoss EAP 6.4 からのエクスポート
   - メッセージングデータの JBoss EAP 7.x からのエクスポート
2. XML 形式のメッセージングデータのインポート

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。

メッセージングデータの **JBoss EAP 6.4** からのエクスポート
JMS サポートプロバイダーが HornetQ から ActiveMQ Artemis に変更になったため、JBoss EAP 7.0 以上のリリースではメッセージングデータの形式と場所が変更になりました。

JBoss EAP 6.4 からメッセージングデータをエクスポートする場合は、HornetQ の **exporter** ユーティリティーを使用する必要があります。HornetQ **exporter** ユーティリティーは、メッセージングデータを生成し、JBoss EAP 6.4 から XML 形式のファイルへエクスポートします。このコマンドでは、JBoss EAP 6.4 に同梱されている必須の HornetQ JAR へのパスを指定し、以前のリリースからの **messagingbindings/**、**messagingjournal/**、**messagingpaging/**、および **messaginglargemessages/** フォルダーへのパスを引数として渡し、エクスポートされる XML データを書き込む出力ファイルを指定する必要があります。

以下は HornetQ **exporter** ユーティリティーで必要となる構文です。

```
$ java -jar -mp MODULE_PATH org.hornetq.exporter MESSAGING_BINDINGS_DIRECTORY
MESSAGING_JOURNAL_DIRECTORY MESSAGING_PAGING_DIRECTORY
MESSAGING_LARGE_MESSAGES_DIRECTORY > OUTPUT_DATA.xml
```

カスタムモジュールを作成し、パッチやアップグレードでインストールされた JAR を含む HornetQ JAR の正しいバージョンがロードされ、**exporter** ユーティリティーで利用可能になるようにします。好みのエディターを使用して、**EAP6_HOME/modules/org/hornetq/exporter/main/** ディレクトリーに新しい **module.xml** ファイルを作成し、以下のコンテンツをコピーします。

```
<?xml version="1.0" encoding="UTF-8"?>
<module xmlns="urn:jboss:module:1.1" name="org.hornetq.exporter">
```

```
    <main-class name="org.hornetq.jms.persistence.impl.journal.XmlDataExporter"/>
    <properties>
        <property name="jboss.api" value="deprecated"/>
    </properties>
    <dependencies>
        <module name="org.hornetq"/>
    </dependencies>
</module>
```

注記

カスタムモジュールは **modules/system/layers/base/** ディレクトリーではなく、**modules/** ディレクトリー内に作成します。

以下の手順に従い、データをエクスポートします。

1. JBoss EAP 6.4 サーバーを停止します。
2. 上記の説明にあるようにカスタムモジュールを作成します。
3. 以下のコマンドを実行してデータをエクスポートします。

```
$ java -jar jboss-modules.jar -mp modules/ org.hornetq.exporter
standalone/data/messagingbindings/ standalone/data/messagingjournal/
standalone/data/messagingpaging standalone/data/messaginglargemessages/ >
OUTPUT_DIRECTORY/OldMessagingData.xml
```

4. 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。
5. 使用中のオペレーティングシステムで利用可能なツールを使用して、生成された出力ファイルの XML を検証します。

メッセージングデータの **JBoss EAP 7.x** からのエクスポート
以下の手順に従って、JBoss EAP 7.x からメッセージングデータをエクスポートします。

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()
```

4. 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。
5. 使用中のオペレーティングシステムで利用可能なツールを使用して、生成された出力ファイルの XML を検証します。

XML 形式のメッセージングデータのインポート
以下のように、 **import-journal** 操作を使用して、XML ファイルを JBoss EAP 7.0 およびそれ以降のリリースにインポートします。

重要

ターゲットサーバーによって一部のメッセージングタスクがすでに実行済みである場合は、インポートに失敗した場合にデータを損失しないようにするため、**import-journal** 操作の実行前にメッセージングフォルダーを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. JBoss EAP 6.4 サーバーを 7.2 に移行する場合、サーバー設定の移行が完了してから管理 CLI の migrate 操作の使用または JBoss Server Migration Tool の実行を開始してください。このツールの設定および実行方法に関する詳細は、『Using the JBoss Server Migration Tool』を参照してください。

2. JMS クライアントが未接続の状態で、JBoss EAP 7.x サーバーを通常モードで起動します。

重要

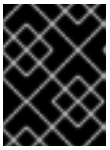
接続している JMS クライアントがない状態でサーバーを起動することが重要になります。これは、**import-journal** 操作が JMS プロデューサーのように動作するからです。操作の実行中、メッセージは即座に使用できます。インポート中にこの操作に失敗し、JMS クライアントが接続状態である場合、JMS クライアントがメッセージの一部を消費済みである可能性があるため、復元することができません。

3. 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
```

4. 以下の管理 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。

警告

JBoss EAP 7.0 を使用している場合、Red Hat JBoss Enterprise Application Platform 7.0 Update 05 またはこれ以降の累積パッチを JBoss EAP インストールに適用し、大型メッセージの読み取り時に既知の問題が発生しないようにする必要があります。詳細は、JBEAP-4407 - Consumer crashes with IndexOutOfBoundsException when reading large messages from imported journal を参照してください。

この問題は JBoss EAP 7.1 以降には影響しません。

メッセージングデータのインポート失敗からの復元
import-journal 操作に失敗した場合、以下の手順に従って復元を行います。

1. JBoss EAP 7.x 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ディレクトリーの場所を正しく判断する管理 CLI コマンドについては、「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。
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. 手順を繰り返して、XML でフォーマットされたメッセージングデータをインポートします。

4.9.2.2. JMS ブリッジを使用したメッセージングデータの移行

この方法では、JMS ブリッジを JBoss EAP 7.x サーバーに設定およびデプロイします。JMS ブリッジはメッセージを JBoss EAP 6.4 HornetQ のキューから JBoss EAP 7.x ActiveMQ Artemis のキューに移動します。

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

ここでは、メッセージングデータを JBoss EAP 6.4 から JBoss EAP 7.x に移動するためにサーバーを設定し、JMS ブリッジをデプロイする方法を説明します。

1. ソース JBoss EAP 6.4 サーバーを設定します。
2. ターゲット JBoss EAP 7.x サーバーを設定します。
3. メッセージングデータを移行します。

ソース **JBoss EAP 6.4** サーバーの設定

1. JBoss EAP 6.4 サーバーを停止します。
2. HornetQ のジャーナルおよび設定ファイルをバックアップします。

- デフォルトでは、HornetQ ジャーナルは **EAP6_HOME/standalone/data/** ディレクトリーにあります。
- 各リリースにおけるメッセージングフォルダーのデフォルトの場所は、「メッセージングフォルダー名のマッピング」を参照してください。

3. JMS メッセージが含まれる **InQueue** JMS キューが JBoss EAP 6.4 サーバーで定義されていることを確認してください。
4. **messaging** サブシステムの設定に以下と似た **RemoteConnectionFactory** のエントリーが含まれていることを確認してください。

```
<connection-factory name="RemoteConnectionFactory">
   <entries>
      <entry name="java:jboss/exported/jms/RemoteConnectionFactory"/>
   </entries>
   ...
</connection-factory>
```

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/hornetq-server=default/connection-
factory=RemoteConnectionFactory:add(factory-type=XA_GENERIC, connector=[netty],
entries=[java:jboss/exported/jms/RemoteConnectionFactory],ha=true,block-on-
acknowledge=true,retry-interval=1000,retry-interval-multiplier=1.0,reconnect-attempts=-1)
```

ターゲット **JBoss EAP 7.x** サーバーの設定

1. 以前のリリースでは、JMS ブリッジ設定が HornetQ サーバーに接続するには、**org.hornetq** モジュールが必要です。このモジュールと直接の依存関係は JBoss EAP 7.x では存在しないため、以前のリリースから以下のモジュールをコピーする必要があります。
   - **org.hornetq** モジュールを JBoss EAP 7.x の **EAP_HOME/modules/org/** ディレクトリーにコピーします。
     - このモジュールにパッチを適用しなかった場合は、 JBoss EAP 6.4 サーバーから **EAP6_HOME/modules/system/layers/base/org/hornetq/** をコピーします。
     - このモジュールにパッチを適用した場合は、JBoss EAP 6.4 サーバーから **EAP6_HOME/modules/system/layers/base/.overlays/layer-base-jboss-eap-6.4.x.CP/org/hornetq/** をコピーします。
   - JBoss EAP 7.x の **EAP_HOME/modules/org/hornetq/main/module.xml** ファイルから HornetQ **lib** の **<resource-root>** を削除します。
     - JBoss EAP 6.4 の **org.hornetq** モジュールにパッチを適用しなかった場合、ファイルから以下の行を削除します。

       ```
       <resource-root path="lib"/>
       ```

     - JBoss EAP 6.4 の **org.hornetq** モジュールにパッチを適用した場合、ファイルから以下の行を削除します。

       ```
       <resource-root path="lib"/>
       <resource-root path="../../../../../org/hornetq/main/lib"/>
       ```

警告

HornetQ **lib** パス **resource-root** の削除に失敗すると、ブリッジに障害が発生し、以下のエラーがログファイルに記録されます。

```
2016-07-15 09:32:25,660 ERROR
[org.jboss.as.controller.management-operation] (management-
handler-thread - 2) WFLYCTL0013: Operation ("add") failed -
address: ([
    ("subsystem" => "messaging-activemq"),
    ("jms-bridge" => "myBridge")
]) - failure description: "WFLYMSGAMQ0086: Unable to load
module org.hornetq"
```

- **org.jboss.netty** モジュールを JBoss EAP 7.x **EAP_HOME/modules/org/jboss/** ディレクトリーにコピーします。
  - このモジュールにパッチを適用しなかった場合は、 JBoss EAP 6.4 サーバーから **EAP6_HOME/modules/system/layers/base/org/jboss/netty/**フォルダーをコピーします。
  - このモジュールにパッチを適用した場合は、 JBoss EAP 6.4 サーバーから **EAP6_HOME/modules/system/layers/base/.overlays/layer-base-jboss-eap-6.4.x.CP/org/jboss/netty** フォルダーをコピーします。

2. JBoss EAP 6.4 サーバーから受信したメッセージを格納するために JSM キューを作成します。以下は、**MigratedMessagesQueue** JMS キューを作成してメッセージを受信する管理 CLI コマンドの例になります。

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

このコマンドにより、JBoss EAP 7 .x サーバーの **messaging-activemq** サブシステムに以下のデフォルトサーバー用の **jms-queue** 設定が作成されます。

```
<jms-queue name="MigratedMessagesQueue" entries="jms/queue/MigratedMessagesQueue
java:jboss/exported/jms/queue/MigratedMessagesQueue"/>
```

3. **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])
```

4. JBoss EAP 6 .4 サーバーで設定された **InQueue** JMS キューからメッセージを読み取る JMS ブリッジを作成およびデプロイし、JBoss EAP 7.x サーバーで設定された **MigratedMessagesQueue** に転送します。

```
/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"=>"remote://127.0.0.1:4447")],target-destination=jms/queue/MigratedMessagesQueue,target-connection-factory=java:/ConnectionFactory)
```

これにより、以下の **jms-bridge** 設定が JBoss EAP 7.x サーバーの **messaging-activemq** サブシステムに作成されます。

```
<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" module="org.hornetq">
    <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="remote://127.0.0.1:4447"/>
        </source-context>
    </source>
    <target destination="jms/queue/MigratedMessagesQueue" connection-factory="java:/ConnectionFactory"/>
</jms-bridge>
```

5. セキュリティーが JBoss EAP 6.4 に設定されている場合、接続の作成時に JNDI ルックアップで使用される正しいユーザー名およびパスワードを指定する **source-context** が含まれるよう、JMS ブリッジ設定 **<source>** 要素を設定する必要もあります。

メッセージングデータの移行

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。
   - JNDI ルックアップの **java.naming.provider.url**。
2. ターゲット JMS 宛先が JBoss EAP 7.x サーバーにデプロイされているようにしてください。
3. JBoss EAP 6.4 サーバーと JBoss EAP 7.x サーバーを両方起動します。

4.9.2.3. メッセージングフォルダー名のマッピング

以下の表では、JBoss EAP の以前および現在のリリースで対応するメッセージングディレクトリーの名前を示しています。ディレクトリーは **jboss.server.data.dir** ディレクトリーに相対的で、指定がない場合はデフォルトで **EAP_HOME/standalone/data/** になります。

JBoss EAP 6.4 のディレクトリー名	JBoss EAP 7.x のディレクトリー名
messagingbindings/	**activemq/bindings/**
messagingjournal/	**activemq/journal/**
messaginglargemessages/	**activemq/largemessages/**
messagingpaging/	**activemq/paging/**

注記

大型のメッセージがない場合や、ページングが無効になっている場合は、**messaginglargemessages/** および **messagingpaging/** ディレクトリーが存在しないことがあります。

4.9.2.4. メッセージングフォルダーデータのバックアップ

ターゲットサーバーによってメッセージがすでに処理されている場合、ターゲットメッセージングフォルダーをバックアップしてから作業を開始した方がよいでしょう。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/** ですが、場所を設定できます。メッセージングデータの場所が分からない場合は、以下の管理 CLI コマンドを使用してメッセージングフォルダーの場所を探すことができます。

```
/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
```

フォルダーの場所を特定したら、各フォルダーを安全にバックアップできる場所にコピーします。

4.9.3. JMS 宛先の移行

JBoss EAP 6 では、JMS 宛先キューは **messaging** サブシステムの **<hornetq-server>** 要素下にある **<jms-destinations>** 要素に設定されました。

```
<hornetq-server>
  ...
  <jms-destinations>
    <jms-queue name="testQueue">
      <entry name="queue/test"/>
       <entry name="java:jboss/exported/jms/queue/test"/>
    </jms-queue>
  </jms-destinations>
  ...
</hornetq-server>
```

JBoss EAP 7 では、JMS 宛先キューは **messaging-activemq** サブシステムのデフォルトの **<server>** 要素に設定されます。

```
<server name="default">
```

```
    ...
    <jms-queue name="testQueue" entries="queue/test java:jboss/exported/jms/queue/test"/>
    ...
</server>
```

4.9.4. メッセージングインターセプターの移行

JBoss EAP 7 では、JMS メッセージングプロバイダーが HornetQ から ActiveMQ Artemis に変更されたため、メッセージングインターセプターが大幅に変更されました。

以前の JBoss EAP リリースに含まれていた HornetQ **messaging** サブシステムでは、HornetQ インターセプターを JAR に追加し、HornetQ **module.xml** ファイルを修正するというインストール方法が必要でした。

JBoss EAP 7 に含まれる **messaging-activemq** サブシステムは、**module.xml** ファイルの修正を必要としません。Apache ActiveMQ Artemis Interceptor インターフェースを実装するユーザーインターセプタークラスは、どのサーバーモジュールからでもロードできるようになりました。インターセプターのロード先となるモジュールは、サーバー設定ファイルの **messaging-activemq** サブシステムで指定します。

例: インターセプター設定

```
<subsystem xmlns="urn:jboss:domain:messaging-activemq:4.0">
  <server name="default">
    ...
    <incoming-interceptors>
      <class name="com.mycompany.incoming.myInterceptor" module="com.mycompany" />
      <class name="com.othercompany.incoming.myOtherInterceptor" module="com.othercompany" />
    </incoming-interceptors>
    <outgoing-interceptors>
      <class name="com.mycompany.outgoing.myInterceptor" module="com.mycompany" />
      <class name="com.othercompany.outgoing.myOtherInterceptor" module="com.othercompany" />
    </outgoing-interceptors>
  </server>
</subsystem>
```

4.9.5. Netty サーブレット設定の変更

JBoss EAP 6 では、Netty サーブレットトランスポートと動作するようサーブレットエンジンを設定することができました。JBoss EAP 7 では、ビルトインメッセージングプロバイダーが HornetQ から ActiveMQ Artemis に変更になったため、この設定は使用できなくなりました。新しいビルトインメッセージング HTTP コネクターおよび HTTP アクセプターを使用するよう、サーブレット設定を変更する必要があります。

4.9.6. 汎用 JMS リソースアダプターの設定

サードパーティー JMS プロバイダーと使用するために汎用 JMS リソースアダプターを設定する方法は JBoss EAP 7 で変更になりました。詳細は、JBoss EAP『Configuring Messaging』の「**Deploying a Generic JMS Resource Adapter**」を参照してください。

4.9.7. メッセージング設定の変更

JBoss EAP 7.0 では、**check-for-live-server** 属性を指定せずに **replication-master** ポリシーを設定した場合のデフォルト値が **false** でした。これは JBoss EAP 7.1 で変更になりました。 **check-for-live-server** 属性のデフォルト値は **true** です。

以下は、**check-for-live-server** 属性を指定せずに **replication-master** を設定する管理 CLI コマンドの例になります。

```
/subsystem=messaging-activemq/server=default/ha-policy=replication-master: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-master: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"}
}
```

4.9.8. リリース間の JMS シリアル化動作の変更

JMS 1.1 と JMS 2.0.0 では、**javax.jms.JMSException** の **serialVersionUID** が変更になりました。そのため、JMS 1.1 を使用して **JMSException** のインスタンスまたは **JMSException** のサブクラスをシリアライズした場合、JMS 2.0.0 を使用してデシリアライズすることはできません。その逆も同様です。JMS 2.0.0 を使用して **JMSException** のインスタンスをシリアライズした場合、JMS 1.1 を使用してデシリアライズすることはできません。両方の場合で以下のような例外が発生します。

```
javax.jms.JMSException: javax.jms.JMSException; local class incompatible: stream classdesc serialVersionUID = 8951994251593378324, local class serialVersionUID = 2368476267211489441
```

この問題は、JMS 2.0.1 メンテナンスリリースで修正されました。

以下の表は、各 JBoss EAP リリースの JMS 実装の詳細を表しています。

表4.4 各 JBoss EAP リリースの JMS 実装

JBoss EAP バージョン	JMS 実装	JMS バージョン
6.4	HornetQ	JMS 1.1
7.0	Apache ActiveMQ Artemis	JMS 2.0.0
7.1 以上	Apache ActiveMQ Artemis	JMS 2.0.1 以上

serialVersionUID の互換性が維持されないと、以下の状況で移行時に問題が発生することがあります。

- JBoss EAP 6.4 クライアントを使用して **JMSException** が含まれるメッセージを送信し、メッセージングデータを JBoss EAP 7.0 に移行した後、JBoss EAP 7.0 クライアントを使用してそのメッセージをデシリアライズしようとするとデシリアライズに失敗し、例外が発生します。これは、JMS 1.1 の **serialVersionUID** は JMS 2.0.0 の serialVersionUID と互換性が ないからです。

- JBoss EAP 7.0 クライアントを使用して **JMSException** が含まれるメッセージを送信し、メッセージングデータを JBoss EAP 7.1 以上のリリースに移行した後、JBoss EAP 7.1 以上の クライアントを使用してそのメッセージをデシリアライズしようとするとデシリアライズに失敗し、例外が発生します。これは、JMS 2.0.0 の **serialVersionUID** は JMS 2.0.1 以上の serialVersionUID と互換性がないからです。

JBoss EAP 6.4 クライアントを使用して **JMSException** が含まれるメッセージを送信し、メッセージングデータを JBoss EAP 7.1 以上のリリースに移行した後、JBoss EAP 7.1 以上のクライアントを使用してそのメッセージをデシリアライズした場合、JMS 1.1 の **serialVersionUID** は JMS 2.0.1 以上の serialVersionUID と互換性があるため、デシリアライズに成功します。

重要

以下を実行してからメッセージングデータを移行することが推奨されます。

- JMSExceptions が含まれる JMS 1.1 のメッセージをすべて消費してからメッセージングデータを JBoss EAP 6.4 から JBoss EAP 7.0 へ移行します。

- JMSExceptions が含まれる JMS 2.0.0 のメッセージをすべて消費してからメッセージングデータを JBoss EAP 7.0 から JBoss EAP 7.1 以上のリリースへ移行します。

4.10. JMX 管理の変更

JBoss EAP 6 の HornetQ コンポーネントは独自の JMX 管理を提供しましたが、それは推奨されず、今回非推奨となったため、サポート対象外になりました。JBoss EAP 6 でこの機能に依存していた場合、EAP 7 で提供される JBoss EAP 管理 CLI または JMX 管理のいずれかを使用するよう、管理ツールを移行する必要があります。

また、クライアントライブラリーをアップグレードして、JBoss EAP 7 に同梱される **jboss-client.jar** を使用する必要もあります。

以下は、JBoss EAP 6 で使用された HornetQ JMX 管理コードの例になります。

```
JMXConnector connector = null;
try {
    HashMap environment = new HashMap();
    String[] credentials = new String[]{"admin", "Password123!"};
    environment.put(JMXConnector.CREDENTIALS, credentials);

    // HornetQ used the protocol "remoting-jmx" and port "9999"
    JMXServiceURL beanServerUrl = new JMXServiceURL("service:jmx:remoting-
jmx://127.0.0.1:9990");

    connector = JMXConnectorFactory.connect(beanServerUrl, environment);
    MBeanServerConnection mbeanServer = connector.getMBeanServerConnection();

    // The JMX object name pointed to the HornetQ JMX management
    ObjectName objectName = new ObjectName("org.hornetq:type=Server,module=JMS");
```

```
    // The invoked method name was "listConnectionIDs"
    String[] connections = (String[]) mbeanServer.invoke(objectName, "listConnectionIDs", new
Object[]{}, new String[]{});
    for (String connection : connections) {
        System.out.println(connection);
    }
} finally {
    if (connector != null) {
      connector.close();
    }
}
```

以下は、JBoss EAP 7 の ActiveMQ Artemis に必要な同等のコード例になります。

```
JMXConnector connector = null;
try {
    HashMap environment = new HashMap();
    String[] credentials = new String[]{"admin", "Password123!"};
    environment.put(JMXConnector.CREDENTIALS, credentials);

    // ActiveMQ Artemis uses the protocol "remote+http" and port "9990"
    JMXServiceURL beanServerUrl = new
JMXServiceURL("service:jmx:remote+http://127.0.0.1:9990");

    connector = JMXConnectorFactory.connect(beanServerUrl, environment);
    MBeanServerConnection mbeanServer = connector.getMBeanServerConnection();

    // The JMX object name points to the new JMX management in the `messaging-activemq`
subsystem
    ObjectName objectName = new ObjectName("jboss.as:subsystem=messaging-
activemq,server=default");

    // The invoked method name is now "listConnectionIds"
    String[] connections = (String[]) mbeanServer.invoke(objectName, "listConnectionIds", new Object[]
{}, new String[]{});
    for (String connection : connections) {
        System.out.println(connection);
    }
} finally {
    if (connector != null) {
      connector.close();
    }
}
```

新しい実装ではメソッド名とパラメーターが変更されたことに注意してください。以下の手順に従うと、JConsole で新しいメソッド名を検索できます。

1. 以下のコマンドを使用して JConsole に接続します。

```
$ EAP_HOME/bin/jconsole.sh
```

2. JBoss EAP のローカルプロセスに接続します。「jboss-modules.jar」で始まることに注意してください。

3. MBeans タブで jboss.as → messaging-activemq → default → Operations の順に選択し、メソッド名と属性のリストを表示します。

4.11. ORB サーバー設定の変更

JBoss EAP 7 では、JacORB 実装が OpenJDK ORB のダウンストリームブランチに変更になりました。

EAP_HOME/modules/system/layers/base/ にあった **org.jboss.as.jacorb** 拡張モジュールは、**org.wildfly.iiop-openjdk** 拡張モジュールに置き換えられました。

サーバー設定ファイルの **urn:jboss:domain:jacorb:1.4** サブシステム設定ネームスペースは **urn:jboss:domain:iiop-openjdk:2.1** ネームスペースに置き換えられました。

以下の例は、JBoss EAP 6 のデフォルトの **jacorb** システム設定を示しています。

```
<subsystem xmlns="urn:jboss:domain:jacorb:1.4">
    <orb socket-binding="jacorb" ssl-socket-binding="jacorb-ssl">
        <initializers security="identity" transactions="spec"/>
    </orb>
</subsystem>
```

以下の例は、JBoss EAP 7 のデフォルトの **iiop-openjdk** サブシステム設定を示しています。

```
<subsystem xmlns="urn:jboss:domain:iiop-openjdk:2.1">
    <orb socket-binding="jacorb" ssl-socket-binding="jacorb-ssl" />
    <initializers security="identity" transactions="spec" />
</subsystem>
```

新しい **iiop-openjdk** サブシステム設定は、レガシー要素および属性のサブセットのみを受け入れます。以下は、有効な要素および属性がすべて含まれる、前リリースの JBoss EAP の **jacorb**サブシステム設定例になります。

```
<subsystem xmlns="urn:jboss:domain:jacorb:1.4">
  <orb name="JBoss" print-version="off" use-imr="off" use-bom="off"  cache-typecodes="off"
      cache-poa-names="off" giop-minor-version="2" socket-binding="jacorb" ssl-socket-
binding="jacorb-ssl">
      <connection retries="5" retry-interval="500" client-timeout="0" server-timeout="0"
          max-server-connections="500" max-managed-buf-size="24" outbuf-size="2048"
          outbuf-cache-timeout="-1"/>
      <initializers security="off" transactions="spec"/>
  </orb>
  <poa monitoring="off" queue-wait="on" queue-min="10" queue-max="100">
      <request-processors pool-size="10" max-threads="32"/>
  </poa>
  <naming root-context="JBoss/Naming/root" export-corbaloc="on"/>
  <interop sun="on" comet="off" iona="off" chunk-custom-rmi-valuetypes="on"
      lax-boolean-encoding="off" indirection-encoding-disable="off" strict-check-on-tc-creation="off"/>
  <security support-ssl="off" add-component-via-interceptor="on" client-supports="MutualAuth"
      client-requires="None" server-supports="MutualAuth" server-requires="None"/>
  <properties>
      <property name="some_property" value="some_value"/>
  </properties>
</subsystem>
```

以下の要素属性はサポート対象外になったため、削除する必要があります。

表4.5 削除する属性

要素	サポートされない属性
<orb>	• client-timeout • max-managed-buf-size • max-server-connections • outbuf-cache-timeout • outbuf-size • connection retries • retry-interval • name • server-timeout
<poa>	• queue-min • queue-max • pool-size • max-threads

以下の **on/off** 属性はサポート対象外となり、管理 CLI の **migrate** 操作を実行しても移行されません。これらの属性が **on** に設定されていると移行の警告が表示されます。**<security support-ssl="on|off">** などの、この表に記載されていない **on/off** 属性のサポートは継続され、正常に移行されます。唯一の違いは、値が **on/off** から **true/false** に変更されることです。

表4.6 off に設定するまたは削除する属性

要素	off に設定する属性
<orb>	• cache-poa-names • cache-typecodes • print-version • use-bom • use-imr

要素	off に設定する属性
<interop>	(**sun** 以外すべて) • comet • iona • chunk-custom-rmi-valuetypes • indirection-encoding-disable • lax-boolean-encoding • strict-check-on-tc-creation
<poa>	• monitoring • queue-wait

4.12. THREADS サブシステム設定の移行

JBoss EAP 6 のサーバー設定には、サーバーの異なるサブシステムにまたがってスレッドプールを管理するために使用される **threads** が含まれていました。

threads サブシステムは JBoss EAP 7 では利用できないようになりました。この代わりに、各サブシステムが独自のスレッドプールを管理します。

infinispan サブシステムのスレッドプールを設定する方法は、JBoss EAP『設定ガイド』の「**Infinispan** スレッドプールの設定」を参照してください。

jgroups サブシステムのスレッドプールを設定する方法は、JBoss EAP『設定ガイド』の「**JGroups** スレッドプールの設定」を参照してください。

JBoss EAP 6 では、**threads** サブシステムに定義された **executor**を参照して **web** サブシステムのコネクターおよびリスナーのスレッドプールを設定しました。JBoss EAP 7 では、**io** サブシステムに定義された **worker** を参照して **undertow** サブシステムのスレッドプールを設定します。詳細は、JBoss EAP『設定ガイド』の「**IO** サブシステムの設定」を参照してください。

remoting サブシステムのスレッドプール設定の変更に関する詳細は、本ガイドの「Remoting サブシステム設定の移行」と、JBoss EAP『設定ガイド』の「エンドポイントの設定」を参照してください。

4.13. REMOTING サブシステム設定の移行

JBoss EAP 6 では、複数の **worker-*** 属性を設定して **remoting** サブシステムのスレッドプールを設定しました。JBoss EAP 7 では **remoting** サブシステムでワーカースレッドプールを設定しないようになりました。 既存の設定を変更しようとすると、以下のメッセージが表示されます。

> WFLYRMT0022: Worker configuration is no longer used, please use endpoint worker configuration

JBoss EAP 7 では、ワーカースレッドプールは、**io** サブシステムで定義される **worker** を参照するエンドポイント設定に置き換えられました。

エンドポイントの設定方法については、JBoss EAP『設定ガイド』の「エンドポイントの設定」を参照してください。

4.14. WEBSOCKET サーバー設定の変更

JBoss EAP 6 で WebSockets を使用するには、以下と同様のコマンドを使用して JBoss EAP サーバー設定ファイルの **web** サブシステムにある **http** コネクターに対して非ブロッキング Java NIO2 コネクターを有効にする必要がありました。

```
/subsystem=web/connector=http/:write-
attribute(name=protocol,value=org.apache.coyote.http11.Http11NioProtocol)
```

アプリケーションで WebSockets を使用するには、アプリケーションの **WEB-INF/jboss-web.xml** ファイル内で **<enable-websockets>** 要素を作成し、これを **true** に設定する必要もありました。

JBoss EAP 7 では、デフォルトの WebSocket サポートに対してサーバーを設定したり、アプリケーションがそれを使用するように設定したりする必要がなくなりました。WebSocket は Java EE 7 では必須で、必要なプロトコルはデフォルトで設定されています。さらに複雑な WebSocket の設定は、JBoss EAP サーバー設定ファイルの **undertow** サブシステムにある **servlet-container** で行います。以下のコマンドを実行すると、使用できる設定を表示できます。

```
/subsystem=undertow/servlet-container=default/setting=websockets:read-resource(recursive=true)
{
    "outcome" => "success",
    "result" => {
        "buffer-pool" => "default",
        "dispatch-to-worker" => true,
        "worker" => "default"
    }
}
```

WebSocket の開発に関する詳細は、JBoss EAP『開発ガイド』の「**WebSocket** アプリケーションの作成」を参照してください。

また、WebSocket のコードサンプルは JBoss EAP に同梱されるクイックスタートに含まれています。

4.15. シングルサインオンサーバーの変更

JBoss EAP 7 でも、**infinispan** サブシステムによって HA サービスの分散キャッシングサポートが Infinispan キャッシュの形式で提供されます。 しかし、認証情報のキャッシングおよび分散の処理方法はこれまでのリリースとは異なります。

JBoss EAP 6 では、シングルサインオン (SSO) が Infinispan キャッシュに提供されないと、キャッシュが分散されませんでした。

JBoss EAP 7 では、HA プロファイルを選択すると SSO が自動的に分散されるようになりました。HA プロファイルを実行している場合、各ホストは web キャッシュコンテナーのデフォルトキャッシュを基にした独自の Infinispan キャッシュを持ちます。このキャッシュは関連するセッションとホストの SSO クッキーの情報を格納します。JBoss EAP は、各キャッシュ情報をすべてのホストに伝搬する処理を行います。JBoss EAP 7 では、明確に Infinispan キャッシュを SSO に割り当てる方法はありません。

JBoss EAP 7 では、SSO はサーバー設定ファイルの **undertow** サブシステムで設定されます。

JBoss EAP 7 に移行する際、アプリケーションコードを変更する必要はありません。

4.16. データソース設定の変更

4.16.1. JDBC データソースドライバー名

以前のリリースの JBoss EAP でデータソースを設定した場合、ドライバー名に指定される値は JDBC ドライバー JAR に含まれる **META-INF/services/java.sql.Driver** ファイルにリストされたクラスの数によって決まりました。

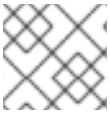
単一クラスを含むドライバー
META-INF/services/java.sql.Driver ファイルで指定されたクラスが 1 つのみであった場合、ドライバー名は単に JDBC ドライバー JAR の名前でした。これは JBoss EAP 7 でも変更ありません。

複数クラスを含むドライバー
JBoss EAP 6 では、**META-INF/services/java.sql.Driver** ファイルに複数のクラスが記載されている場合、以下の形式で JAR 名にドライバークラスにするクラスの名前とメジャーおよびマイナーバージョンを追加してクラスを指定していました。

> **JAR_NAME** + **DRIVER_CLASS_NAME** + "_" + **MAJOR_VERSION** + "_" + **MINOR_VERSION**

これは JBoss EAP 7 で変更になりました。以下の形式でドライバー名を指定できるようになりました。

> **JAR_NAME** + "_" + **DRIVER_CLASS_NAME** + "_" + **MAJOR_VERSION** + "_" + **MINOR_VERSION**



注記

JAR_NAME と DRIVER_CLASS_NAME の間にアンダースコアが加えられています。

2 つのクラスが含まれるドライバーの例としては、MySQL 5.1.31 JDBC ドライバーが挙げられます。このドライバークラス名は **com.mysql.jdbc.Driver** となります。以下の 2 つの例では、JBoss EAP の以前のリリースと現行リリースにおけるドライバー名の指定方法の違いが示されています。

例: JBoss EAP 6 ドライバー名

```
mysql-connector-java-5.1.31-bin.jarcom.mysql.jdbc.Driver_5_1
```

例: JBoss EAP 7 ドライバー名

```
mysql-connector-java-5.1.31-bin.jar_com.mysql.jdbc.Driver_5_1
```

4.17. セキュリティーサーバー設定の変更

JBoss EAP 7 に移行し、Java Security Manager を有効にして実行する計画がある場合、ポリシーを定義する方法に変更があり、追加の設定変更が必要になる可能性があります。また、JBoss EAP 7 ではカスタムセキュリティーマネージャーはサポートされないため注意してください。

Java Security Manager のサーバー設定の変更に関する詳細は、JBoss EAP『How to Configure Server Security』の「**Considerations Moving from Previous Versions**」を参照してください。

4.17.1. JBoss EAP 7.0 と JBoss EAP 7.1 間のレガシーセキュリティー動作の変更

4.17.1.1. 到達不可能な LDAP レルムの HTTP ステータスの変更

JBoss EAP 7.0 では、サーバーが到達できる LDAP レルムがない場合は **security** サブシステムが HTTP ステータスコード「401 Unauthorized」を返しました。

JBoss EAP 7.1 以上のレガシー **security** サブシステムは、予期せぬ状況が発生したためにサーバーがリクエストを処理できなかったことをより正確に示す HTTP ステータスコード「500 Internal Error」を返します。

4.17.1.2. LDAP セキュリティーレルムの有効化による DN のロールの解析

JBoss EAP 7.0 では、DN からのロールを解析するために **org.jboss.as.domain.management.security.parseGroupNameFromLdapDN** システムプロパティーを使用して LDAP セキュリティーを有効にしました。このプロパティーが **true** に設定されると、ロールは DN から解析されました。それ以外の場合は、ロールの検索に通常の LDAP 検索が使用されました。

JBoss EAP 7.1 以上では、このシステムプロパティーは非推奨になりました。このオプションを設定するには、以下の管理 CLI コマンドを使用して、コアサービスパスで新規導入された **parse-group-name-from-dn** 属性を **true** に設定します。

```
/core-service=management/security-realm=REALM_NAME/authorization=ldap/group-search=principal-to-group:add(parse-group-name-from-dn=true)
```

4.17.1.3. JBoss EAP SSL 証明書の LDAP サーバーへの送信に関する変更

JBoss EAP 7.0 では、**ldapSSL** セキュリティーレルムを使用するよう管理インターフェースが設定されていると、サーバーと LDAP 間の相互認証に失敗し、管理インターフェースの認証に失敗することがあります。これは、それぞれ異なるスレッドによって 2 つの LDAP 接続が確立され、これらの接続が SSL セッションを共有しないためです。

JBoss EAP 7.1 の LDAP **outbound-connection** には、新しいブール値 **always-send-client-cert** が導入されました。このオプションを使用すると、アウトバウンド LDAP 接続の設定によって、常にクライアント証明書が必要であると設定された LDAP サーバーをサポートすることができます。

LDAP 認証は 2 つの手順で行われます。

1. アカウントを検索します。
2. クレデンシャルを検証します。

デフォルトでは、**always-send-client-cert** 属性は **false** に設定されるため、クライアント SSL 証明書は最初のアカウント検索リクエストとのみ送信されます。この属性が **true** に設定されると、JBoss EAP LDAP クライアントは検索および検証リクエストの両方とともにクライアント証明書を LDAP サーバーに送信します。

以下の管理 CLI コマンドを使用してこの属性を **true** に設定できます。

```
/core-service=management/ldap-connection=my-ldap-connection:write-attribute(name=always-send-client-cert,value=true)
```

これにより、サーバー設定ファイルに以下の LDAP アウトバウンド接続が追加されます。

```
<management>
  ....
  <outbound-connections>
    <ldap name="my-ldap-connection" url="ldap://127.0.0.1:389" search-
```

```
dn="cn=search,dc=myCompany,dc=com" search-credential="myPass" always-send-client-
cert="true"/>
  </outbound-connections>
  ....
</management>
```

4.17.2. FIPS モードの変更

FIPS モードで実行している場合、JBoss EAP 7.0 のデフォルト動作が JBoss EAP 7.1 では変更になったことに注意してください。

レガシーセキュリティーレルムを使用している場合、JBoss EAP 7.1 以上では開発の目的で自己署名証明書が自動的に生成されます。この機能は JBoss EAP 7.0 にはなく、デフォルトで有効になっています。そのため、FIPS モードで実行している場合、サーバーを設定して自己署名証明書の自動作成を無効にする必要があります。無効にしないと、サーバーの起動時に以下のエラーが発生することがあります。

```
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
```

自己署名証明書の自動作成に関する詳細は、JBoss EAP『How to Configure Server Security』の「**Automatic Self-signed Certificate Creation for Applications**」を参照してください。

4.18. TRANSACTIONS サブシステムの変更

JBoss EAP 6 の **transactions** サブシステムで使用できた Transaction Manager 設定属性の一部が JBoss EAP 7 で変更になりました。

transactions サブシステムの削除された属性

以下の表は、JBoss EAP 7 の **transactions** サブシステムから削除された JBoss EAP 6 の属性と、それらの代替となる同等の属性を示しています。

JBoss EAP 6 の属性	JBoss EAP 7 で代替となる属性
path	object-store-path
relative-to	object-store-relative-to

非推奨となったトランザクションサブシステム属性

JBoss EAP 6 の **transactions** サブシステムで使用できた以下の属性は、JBoss EAP 7 では非推奨となりました。非推奨となった属性は将来のリリースで削除される可能性があります。以下の表はそれらの代替となる同等の属性を示しています。

JBoss EAP 6 の属性	JBoss EAP 7 で代替となる属性
use-hornetq-store	use-journal-store

JBoss EAP 6 の属性	JBoss EAP 7 で代替となる属性
hornetq-store-enable-async-io	journal-store-enable-async-io
enable-statistics	statistics-enabled

4.19. MOD_CLUSTER 設定の変更

JBoss EAP 7 では、mod_cluster の静的プロキシーの設定が変更になりました。

JBoss EAP 6 では、**hostname:port** の形式で指定された httpd プロキシーアドレスのカンマ区切りリストである **proxy-list** 属性を設定しました。

proxy-list 属性は JBoss EAP 7 では非推奨になりました。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『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。

mod_cluster 属性の詳細は、JBoss EAP『設定ガイド』の「**ModCluster** サブシステムの属性」を参照してください。

4.20. 設定変更の確認

JBoss EAP 7 には、稼働中のサーバーに加えられた設定変更を追跡する機能があります。この機能を使用すると、管理者は他の許可されたユーザーが追加した設定変更の履歴を確認することができます。

JBoss EAP 7.0 では、オプションの設定と最近の設定変更の一覧表示に**core-service** 管理 CLI コマンドを使用する必要があります。

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 以上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 に導入された新しい **core-management** サブシステムの使用に関する詳細は、JBoss EAP『設定ガイド』の「設定変更の確認」を参照してください。

第5章 アプリケーション移行の変更

5.1. 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。

5.1.1. 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。JAX-RPC は JBoss EAP 7 では使用できず、サポートされません。JAX-RPC を使用するアプリケーションは、現在の Java EE 標準の Web サービスフレームワークである JAX-WS 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。

5.1.2. 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。

JBoss EAP 7 では、Spring の統合がサポートされないようになりました。**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。

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:4.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:javaee="http://java.sun.com/xml/ns/javaee"
  xsi:schemaLocation="urn:jboss:jbossws-jaxws-config:4.0 schema/jbossws-jaxws-config_4_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>
```

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:4.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:javaee="http://java.sun.com/xml/ns/javaee"
  xsi:schemaLocation="urn:jboss:jbossws-jaxws-config:4.0 schema/jbossws-jaxws-config_4_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>
```

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	タイプ	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。

5.1.3. WS-Security の変更

- 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。
- ほとんどの SAML Bean オブジェクトは **org.apache.ws.security.saml.ext** パッケージから **org.apache.wss4j.common.saml package** に移動されました。

- RSA v1.5キートランスポートおよび関連するすべてのアルゴリズムの使用はデフォルトで禁止されています。
- 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。

5.1.4. 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。

5.1.5. Bouncy Castle の要件の変更

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 には **org.bouncycastle** モジュールが同梱されているため、JBossWS はそのクラスローダーに依存して BouncyCastle Security Provider を入手および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。この問題を回避するには、BouncyCastle 1.49、1.51、またはそれ以降のバージョンを使用する必要があります。

5.1.6. 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。

5.1.7. WebServiceRef の JAX-WS 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 JAX-WS 2.2 スタイルのコンストラクターを使用して、web サービス参照にインジェクトされるクライアントを構築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 が同梱される JBoss EAP 7 はこの要件を隠さないようになりました。そのため、コンテナーによってイン

ジェクトされたユーザー提供のサービスクラスが、WebServiceFeature 引数が 1 つ以上含まれる **javax.xml.ws.Service** コンストラクターを使用するよう既存コードを更新し、JAX-WS 2.2 以上を実装する必要があります。

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

5.1.8. 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。

5.1.9. 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。

5.1.10. Java Endorsed Standards Override Mechanism の非推奨

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。

アプリケーションが Apache CXF の JBossWS 実装を使用する場合、JBoss EAP 7 では必要な依存関係が正しい順序で追加されるため、この変更による影響はないはずです。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。

5.1.11. 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、EJB 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 EJB 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 では、**jboss-webservices.xml** デプロイメント記述子ファイルを EAR アーカイブの **META-INF/** ディレクトリーで設定できるようになりました。**jboss-webservices.xml** ファイルが EAR アーカイブと JAR (または WAR) アーカイブの両方で見つかった場合、JAR または WAR の **jboss-webservices.xml** ファイルにある設定データによって EAR 記述子ファイルの対応するデータが上書きされます。

5.2. リモート URL コネクターおよびポートの更新

JBoss EAP 7 では、デフォルトのコネクターが **remote** から **http-remoting** に変更になり、デフォルトのリモート接続ポートが **4447** から **8080** に変更になりました。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。

JBoss EAP 7 の **migrate** 操作を使用して設定を更新すると、移行操作によってサブシステム設定の JBoss EAP 6 リモーティングコネクターおよび **4447** ポート設定が保持されるため、リモートコネク

ター、リモートポート、または JNDI プロバイダー URL を変更する必要がありません。**migrate** 操作の詳細は、「管理 CLI の移行操作」を参照してください。

migrate 操作を使用せずに、新しい JBoss EAP 7 のデフォルト設定を使用して実行する場合は、新しい設定を使用するようにリモートコネクター、リモートポート、および JNDI プロバイダー URL を変更する必要があります。

5.3. メッセージングアプリケーションの変更

5.3.1. JMS デプロイメント記述子の置き換えおよび更新

ネーミングパターン **-jms.xml** によって識別されたプロプライエタリーの HornetQ JMS リソースデプロイメント記述子ファイルは JBoss EAP 7 では動作しません。以下は、JBoss EAP 6 での JMSリソースデプロイメント記述子ファイルの例になります。

```
<?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>
```

以前のリリースで **-jms.xml** JMS デプロイメント記述子をアプリケーションで使用した場合、 Java EE 7 仕様の EE.5.18 で指定されたとおりに標準の Java EE デプロイメント記述子を使用するよう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デプロイメント記述子を更新してください。

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>
```

メッセージングに関するサーバー設定の変更については、「メッセージングサーバー設定の変更」を参照してください。

5.3.2. 外部 JMS クライアントの更新

JBoss EAP 7 は JMS 1.1 API をサポートするため、コードを変更する必要はありません。

JBoss EAP 7 ではデフォルトのリモートコネクターおよびポートが変更になりました。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。

migrate 操作を使用してサーバー設定を移行する場合、これまでの設定は保持され、**PROVIDER_URL** を更新する必要はありません。しかし、新しい JBoss EAP 7 のデフォルト設定を使用して実行する場合は、新しい **http-remoting://localhost:8080** 設定を使用するよう **PROVIDER_URL** を変更する必要があります。詳細は「リモートネーミングクライアントコードの移行」を参照してください。

JMS 2.0 API を使用するためにコードを移行する計画がある場合は、作業例を **helloworld-jms** クイックスタートで確認してください。

5.3.3. 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ソースコードで HornetQ API が使用できました。

JBoss EAP 7 では HornetQ が Apache ActiveMQ Artemis に置き換えられたため、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。

5.3.4. 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

非推奨となった属性の機能は JBoss EAP 7.2 では設定できず、反映されません。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

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

代替の属性の詳細は、『Configuring Messaging』の「**Address Setting Attributes**」を参照してください。

5.3.5. JBoss EAP 7.2 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。

5.4. JAX-RS および 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 7.0 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。RESTful Web サービスの Java API に関する詳細情報は、 JAX-RS 2.0 API Specification を参照してください。

JBoss EAP 7.2 には RESTEasy 3.6.1 が含まれています。これは、JSR 370: Java(TM)API for RESTful Web Services (JAX-RS 2.1) Specification で定義されている JAX-RS 2.1 の実装です。本リリースでは、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.x から 3.6.1 への移行中に問題はほとんど発生しないはずです。RESTEasy 3.6.1 の Java API に関する詳細は、「RESTEasy JAX-RS 3.6.1.Final API」を参照してください。

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。

本セクションでは、これらの変更が RESTEasy または JAX-RS を使用するアプリケーションに及ぼす可能性のある影響について説明します。

5.4.1. 非推奨の RESTEasy 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 によって正式なインターセプターおよびフィルターフレームワークが導入されたため、RESTEasy 2 に含まれたインターセプターフレームワークは非推奨になり、RESTEasy 3.x の JAX-RS 対応インターセプターファシリティーに置き換えられました。関係するインターフェースは、**jaxrs-api** モジュールの **javax.ws.rs.ext** パッケージに定義されます。

- RESTEasy 3.x では、以下のインターセプターインターフェースが非推奨になりました。
  - **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** インターフェースは RESTEasy 3.x の **javax.ws.rs.container.ContainerRequestFilter** インターフェースに置き換えられました。
- RESTEasy 3.x では、以下のインターフェースとクラスも非推奨になりました。
  - **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** インターフェースは **javax.ws.rs.ext.WriterInterceptor** インターフェースに置き換えられました。
- さらに、**javax.ws.rs.ext.MessageBodyWriter** インターフェースの変更の一部は JAX-RS 1.x の後方互換性を維持し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;
    }
}
```

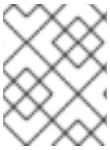
この問題を修正するには、次のように **javax.ws.rs.Produces** のインポートと **@Produces** アノテーションを追加します。

```
...
import javax.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 の以前のリリースからのインターセプターはすべて、新規の JAX-RS フィルターおよびインターセプターインターフェースと並行して実行することが可能です。

インターセプターの詳細情報は、JBoss EAP『Developing Web Services Applications』の「**RESTEasy Interceptors**」を参照してください。

新しい代替の API は、RESTEasy JAX-RS 3.6.1.Final API を参照してください。

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。

- 以下のクラスは非推奨になりました。
  - **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** インターフェースおよび **org.jboss.resteasy.client.EntityTypeFactory** インターフェースも非推奨になりました。
- **org.jboss.resteasy.client.ClientRequest** 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 **javax.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()));
javax.ws.rs.core.Link link = response.getLink("previous");
```

  JAX-RS Web サービスと対話する外部 JAX-RS RESTEasy クライアントの例は、**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ースも非推奨になりました。これらのクラスとインターフェースは、**org.jboss.resteasy.client.jaxrs.cache** パッケージの同等のクラスおよびインターフェースに置き換えられました。

注記

org.jboss.resteasy.client.jaxrs API クラスの詳細については、RESTEasy JAX-RS JavaDoc を参照してください。

StringConverter
org.jboss.resteasy.spi.StringConverter クラスは RESTEasy 3.x では非推奨になりました。この機能は JAX-RS の jax.ws.rs.ext.ParamConverterProvider クラスを使用して置き換えできます。

5.4.2. 削除または保護されている RESTEasy クラス

ResteasyProviderFactory Add メソッド
org.jboss.resteasy.spi.ResteasyProviderFactory **add()** メソッドのほとんどは、RESTEasy 3.0 で削除または保護されました。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
@org.jboss.resteasy.annotations.cache.ServerCached アノテーションは、JAX-RS メソッドへの応答をサーバーでキャッシュすることを指定するものでしたが、これは RESTEasy 3 から削除されたので、アプリケーションコードから削除する必要があります。

5.4.3. 他の RESTEasy 変更点

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** アノテーションで **application/pkcs7-signature** MIME タイプを設定して、そのタイプの形式をバイナリ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** インターフェー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 Documentation」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『Developing Web Services Applications』の「**YAML Provider**」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『Developing Web Services Applications』の「**Text Media Types and Character Sets**」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 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();
    }
}
```

リソースメソッドアルゴリズムの切り替え
3.0.25.Final より前の RESTEasy 3.0.x バージョンで使用されたリソースメソッド一致アルゴリズムでバグが見つかりました。このバグにより、リクエストの応答時に RESTEasy は余分なリソースメソッドを返しました。

一致アルゴリズムには 3 つの手順があります。

1. リクエストパスを使用して可能なリソースクラスを選択する。
2. リクエストパスを使用して可能なリソースメソッドを選択する。
3. 送受信で HTTP 動詞とメディアタイプを使用して最終的なリソースメソッドを選択する。

JAX-RS 2.0 仕様によると、潜在的なリソースメソッドセットのソート後に最大要素のみを手順 3 に渡す必要があります。しかし、RESTEasy 3.0.25 より前の RESTEasy 3.0.x 実装はすべてのメソッドを手順 3 に渡しました。JBoss EAP 7.1.0 に含まれる RESTEasy 3.0.24 は、この不適切な動作を実行します。

JBoss EAP 7.1.1 に含まれる RESTEasy 3.0.25 は、JAX-RS 2.0 仕様に準拠するために手順 3 に渡すメソッドを制限する修正を提供します。厳格でない動作が望ましい場合もあるため、RESTEasy 3.0.25 にはこれまでの動作を有効にできる **context-param** 設定オプションの **resteasy.loose.step2.request.matching** も導入されました。 これはデフォルトで **false** に設定されています。

JBoss EAP サーバーを 7.1.0 から 7.1.1 に更新し、これまでの動作を維持して潜在的なリソースメソッドをすべて手順 3 に渡す場合は、**resteasy.loose.step2.request.matching** オプションを **true** に設定します。

一致アルゴリズムは、一致したすべてのリソースメソッドを手順 3 に渡すよう JAX-RS 2.1 で変更になりました。JBoss EAP 7.2 に含まれている RESTEasy 3.6.1 は、JAX-RS 2.0 仕様で定義されている厳格な動作を維持する **jaxrs.2.0.request.matching** オプションを提供します。

アプリケーションを JBoss EAP 7.1.0 から 7.2.x に移行した場合、リソースメソッドの一致アルゴリズムの動作は変更されません。アプリケーションを JBoss EAP 7.1.1 から 7.2.x に移行し、JAX-RS 2.0 仕様に定義されている厳格な動作を維持したい場合は、**jaxrs.2.0.request.matching** オプションを **true** に変更します。

5.4.4. RESTEasy SPI の変更点

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 JAX-RS 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	javax.ws.rs.ForbiddenException
org.jboss.resteasy.spi.MethodNotAllowedException	javax.ws.rs.NotAllowedException

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

5.4.5. Jackson プロバイダーの変更

JBoss EAP に含まれる Jackson のバージョンは変更されました。JBoss EAP の以前のリリースに含まれていた Jackson は 1.9.9 でした。JBoss EAP 7 には Jackson 2.6.3 またはそれ以降が含まれています。このため、Jackson プロバイダーが **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。

JBoss EAP の以前のリリースに含まれていたデフォルトのプロバイダーを使用するようにアプリケーションを切り替えるには、JBoss EAP『Developing Web Services Applications』の「**Switching the Default Jackson Provider**」を参照してください。

5.4.6. Spring と RESTEasy の統合の変更

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。

5.4.7. 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 では非推奨となり、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>
```

明示的な依存関係の定義方法については、JBoss EAP『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。

5.5. CDI アプリケーションの変更

JBoss EAP 7.2 には CDI 2.0 のサポートが含まれています。このため、CDI 1.0 または CDI 1.2 を使って作成されたアプリケーションを JBoss EAP 7.2 に移行すると、一部の動作が変更になる可能性があります。ここでは、CDI 1.2 および CDI 2.0 の変更内容のいくつかを簡単に取り上げます。

Weld および CDI 2.0 についての追加情報は、以下の参照先で確認できます。

- JSR 365: Contexts and Dependency Injection for Java 2.0
- CDI 2.0 Javadoc
- Weld 3.0.5.Final - CDI Reference Implementation

Bean アーカイブ
CDI によってスキャンされ、bean クラスを検索および処理するようにするため、有効な bean の bean クラスを bean アーカイブにデプロイする必要があります。

CDI 1.0 では、アプリケーションクライアント、EJB、またはライブラリー JAR の **META-INF**/ ディレクトリーに **beans.xml** ファイルが含まれた場合や、WAR の **WEB-INF**/ ディレクトリーに **beans.xml** ファイルが含まれた場合には、アーカイブが 明示的 bean アーカイブとして定義されました。

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 アーカイブは **all**、**annotated**、または **none** の bean 検索モードを持ちます。バージョンのない **beans.xml** ファイルが含まれる bean アーカイブは、デフォルトの bean 検索モード **all** を持ちます。バージョンが **1.1** 以上の **beans.xml** ファイルが含まれる bean アーカイブは **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。
- **beans.xml** ファイルがない CDI 拡張が含まれる場合。

以下の場合、アーカイブは 明示的 bean アーカイブになります。

- バージョン番号が 1.1 以上で、**bean-discovery-mode** が **all** の **beans.xml** ファイルがアーカイブに含まれる場合。
- アーカイブにバージョン番号のない **beans.xml** ファイルが含まれる場合。
- アーカイブに空の **beans.xml** ファイルが含まれる場合。

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 **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。
- アーカイブに **bean-discovery-mode** が **annotated beans.xml** ファイルが含まれる場合。

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

bean アーカイブの詳細は、『JSR 365: Contexts and Dependency Injection for Java 2.0 』の「**Bean Archives**」を参照してください。

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、『JSR 365: Contexts and Dependency Injection for Java 2.0』の「**Conversation context lifecycle in Java EE**」を参照してください。

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、『JSR 365: Contexts and Dependency Injection for Java 2.0』の「**Observer resolution**」を参照してください。

5.6. 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 ではセッション ID のみが返されます。

この変更により、JBoss EAP 6 から JBoss EAP 7 への一部のアップグレードでルートのないクッキー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 クッキーを再作成する場合、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。

5.7. 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、**MANIFEST.MF** および **jboss-deployment-structure.xml** ファイルエントリーを確認し、これらのエントリーが本リリースで削除されたモジュールを参照しないようにしてください。

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>
```

5.8. HIBERNATE および JPA の移行の変更

5.8.1. Hibernate ORM 3.0

JBoss EAP 6.4 の Hibernate ORM 3 を簡単に使用できるようにする統合クラスは JBoss EAP 7 から削除されました。膨大な作業を行わないと JBoss EAP で Hibernate ORM 3 を実行できなくなったため、アプリケーションが Hibernate ORM 3 ライブラリーを使用する場合は、アプリケーションを移行して Hibernate ORM 5 を使用するようにすることが強く推奨されます。Hibernate ORM 5 へ移行できない場合は、Hibernate ORM 3 JAR のカスタム JBoss モジュールを定義し、アプリケーションから Hibernate ORM 5 のクラスを除外する必要があります。

5.8.2. Hibernate ORM 4.0 - 4.3

アプリケーションに対して 2 次キャッシュを有効にする必要がある場合、nfinispan 8.x が Hibernate ORM 5.0 と統合されたことに注意してください。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、Hibernate ORM 5.3 の Infinispan プロジェクトに移されました。 そのため、**hibernate-infinispan** モジュールは本リリースより削除されました。

Hibernate ORM 4.x で作成されたアプリケーションは Hibernate ORM 4.x を引き続き使用できます。

Hibernate ORM 4.x JAR のカスタム JBoss モジュールを定義し、アプリケーションから Hibernate ORM 5 クラスを排除する必要があります。しかし、Hibernate ORM 5 を使用するようにアプリケーションコードを書き直すことが強く推奨されます。Hibernate ORM 5 への移行に関する情報は、「Hibernate ORM 5 への移行」を参照してください。

5.8.3. Hibernate ORM 5 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** インターフェー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** インターフェースから展開された **org.hibernate.engine.jdbc.env.spi.JdbcEnvironment** インターフェースが含まれます。
- **org.hibernate.id.PersistentIdentifierGenerator** 実装に影響する、新しい **org.hibernate.boot.model.relational.ExportableProducer** インターフェー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** インターフェースは **org.hibernate.boot.model.TypeContributor** に移行されました。
- **org.hibernate.metamodel.spi.TypeContributions** インターフェースは **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。

- **javax.persistence.EnumType.STRING** name-mapping を希望する、**hbm.xml** に定義された **EnumType** マッピングでは、 **useNamed(true)** 設定を使用するか、VARCHAR の値を **12** に指定して、設定を明示的に示す必要があります。

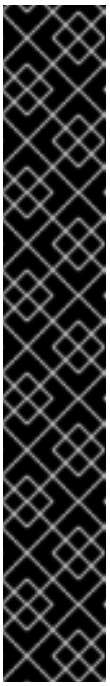
トランザクション管理

- 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、JPA 以外のトランザクションのデフォルトです。

  - **jta**: JTA を使用してトランザクションを管理します。



重要

JPA 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。

JPA でない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 JTA ベースのトランザクションを使用する場合は、このデフォルトによって問題が発生します。JTA ベースのトランザクションを使用する 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ースが新しい 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** インターフェー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。

5.8.4. 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
このリリースの Hibernate には、JBoss EAP 7.1.0 リリースノート の **Hibernate ORM 5.1 の機能** に記載されているパフォーマンスの改善やバグ修正が含まれます。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は、主に以下の項目を中心に変更されています。

- **hbm2ddl.auto** と Hibernate の JPA **schema-generation** サポートの処理を統合。
- NoSQL データストアの Java Persistance (JPA) 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。

移行時にスキーマ管理ツールの変更に注意する必要があるのは、以下のクラスを直接使用するアプリケーションのみです。

- **org.hibernate.tool.hbm2ddl.SchemaExport**
- **org.hibernate.tool.hbm2ddl.SchemaUpdate**
- **org.hibernate.tool.hbm2ddl.SchemaValidator**
- **org.hibernate.tool.schema.spi.SchemaManagementTool** またはこの委譲

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 には、**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。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 **java.sql.DatabaseMetaData#getTables(String, String, String, String[])** 呼び出しを実行する旧式の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。

5.8.5. Hibernate ORM 5.1 から Hibernate ORM 5.3 への移行

JBoss EAP 7.2 には Hibernate ORM 5.3 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 5.1からバージョン 5.3 に移行する際に必要な変更について説明します。

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、JPA 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 JPA 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

- 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。
- JPA 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、JPA 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。

- JPA 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-11264」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、JPA メソッドを使用するようアプリケーションをアップデート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-11264**」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-11264」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 JPA と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** インターフェー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。

5.8.5.1. 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、JPA 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 JPA 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、JPA 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。

5.8.5.2. 互換性トランスフォーマー

JBoss EAP 7.2 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 7.2 の Hibernate 5.3 で同じ動作を実行できるようにする一時的な措置です。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 JPA 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.Session.getHibernateFlushMode()
Enum org.hibernate.FlushMode.NEVER (0)	Enum org.hibernate.FlushMode.MANUAL (0)

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.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)

5.9. HIBERNATE SEARCH の変更

JBoss EAP 7 に同梱される Hibernate Search のバージョンが変更になりました。以前のリリースの JBoss EAP には Hibernate Search 4.6.x が同梱されていました。JBoss EAP 7 には Hibernate Search 5.5.x が同梱されます。

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 API 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、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。

Hibernate Search マッピングの変更
埋め込み関係の **ID** フィールドのインデックス化
@IndexedEmbedded アノテーションを使用して関連エントリーからのフィールドを含む場合、関連エントリーの **id** が含まれなくなりました。**id** が含まれるようにするには、**@IndexedEmbedded** アノテーションの **includeEmbeddedObjectId** 属性を使用します。

例: @IndexedEmbedded アノテーション

```
@IndexedEmbedded(includeEmbeddedObjectId=true)
```

番号および日付のインデックス形式の変更
番号や日付はデフォルトで数字フィールドとしてインデックス化されるようになりました。タイプ **int**、**long**、**float**、**double** のプロパティーおよびこれらのラッパークラスは文字列としてインデックス化されないようになりました。代わりに、これらは Lucene の適切な数字エンコーディングを使用してインデックス化されるようになりました。**id** フィールドはこのルールの例外で、数字タイプによって表されても、デフォルトで文字列キーワードとしてインデックス化されます。**@NumericField** の使用は、数字エンコーディングのカスタム精度を指定する場合以外は廃止されました。数値エンコーディングのカスタム精度を指定しない限り、**@NumericField** が廃止されるようになりました。文字列エンコーディングフィールドブリッジを明示的に指定すると、これまでの文字列ベースのインデックス形式を保持できます。整数の場合、これは org.hibernate.search.bridge.builtin.IntegerBridge です。その他の公開されているフィールドブリッジは、org.hibernate.search.bridge.builtin パッケージを確認してください。

Date および **Calendar** は文字列としてインデックス化されないようになりました。代わりに、インスタンスは 1970 年 1 月 1 日グリニッジ標準時 00:00:00 からの期間 (ミリ秒) を表す長整数としてエンコードされます。新しい EncodingType 列挙を使用するとインデックス形式を切り替えできます。例を以下に示します。

例: @DateBridge および @CalendarBridge アノテーション

```
@DateBridge(encoding=EncodingType.STRING)
@CalendarBridge(encoding=EncodingType.STRING)
```

数字と日付のエンコーディングの変更は重要で、アプリケーションの動作に大きく影響する可能性があります。以前は文字列にエンコードされ、現在は数字にエンコードされるフィールドをターゲットとするクエリーがある場合、クエリーを更新する必要があります。**NumericRangeQuery** で数字フィールドを検索する必要があります。また、ファセッティングがターゲットとするすべてのフィールドが文字列でエンコードされるようにする必要があります。Search クエリー DSL を使用する場合、適切なクエリーが自動的に作成されるはずです。

その他の Hibernate Search の変更

- ソートオプションが改良され、ソートオプションに対してフィールドエンコーディングが誤って指定されるとランタイム例外が発生するようになりました。また、あらかじめソートに使用されるフィールドが分かる場合、Lucene によってより高性能なソート機能が提供されます。Hibernate Search 5.5 は新しい **@SortableField** および **@SortableFields** アノテーションを提供します。詳細は、『Migration Guide from Hibernate Search 5.4 to 5.5 』を参照してください。

- Lucene の **SortField** API には、以下のアプリケーションコードの変更を適用する必要があります。
  以前のリリースの JBoss EAP では、以下のようにクエリーでソートフィールドのタイプを設定しました。

  ```
  fulltextQuery.setSort(new Sort(new SortField("title", SortField.STRING)));
  ```

  JBoss EAP 7 での設定例は次のとおりです。

  ```
  fulltextQuery.setSort(new Sort(new SortField("title", SortField.Type.STRING)));
  ```

- **SearchFactory** は ORM の統合でのみ使用される必要があるため、 **hibernate-search-engine** モジュールから **hibernate-search-orm** モジュールに移動されました。他のインタグレーターは **SearchIntegrator** のみに依存する必要があります。これは、非推奨となった **SearchFactoryIntegrator** の代わりに使用されます。

- 列挙値 **SpatialMode.GRID** の名前が **SpatialMode.HASH** に変更になりました。

- **FullTextIndexEventListener** が最終クラスになりました。現在このクラスを拡張する場合、同じ機能を実現できる他の方法を見つける必要があります。

- **hibernate-search-analyzers** モジュールが削除されました。**org.apache.lucene:lucene-analyzers-common** などの適切な Lucene アーティファクトを直接使用する方法が推奨されます。

- JMS コントロール API が変更になりました。他の ORM 環境で使用できるようにするため、Hibernate ORM の JMS バックエンド依存関係が削除されました。そのため、**org.hibernate.search.backend.impl.jms.AbstractJMSHibernateSearchController** のインプリメンターを新しい署名に応じて調整する必要があります。このクラスは内部クラスで、拡張せずに例として使用することが推奨されます。

- **org.hibernate.search.spi.ServiceProvider** SPI がリファクターされました。古いサービスコントラクトで統合している場合は、ServiceManager、 **Service**、 **Startable**、および **Stoppable** の『**Hibernate Search 5.5 Javadoc**』で新しいコントラクトの詳細を確認してください。

- Lucene 3.x によって生成されたインデックスを保持し、これらのインデックスを Hibernate

Search 5.0 以上で再構築していない場合、**IndexFormatTooOldException** が発生します。マスインデクサーでインデックスを再構築することが推奨されます。再構築できない場合は、ILucene の **IndexUpgrader** を使用してください。デフォルトの動作が変更になった場合があるため、注意して Hibernate Search のマッピングを更新する必要があります。詳細は『Apache Lucene Migration Guide』を参照してください。

- JBoss EAP 7 では、Apache Lucene が 3.6 から 5.3 にアップグレードされました。ご使用のコードが 直接 Lucene のコードをインポートする場合は、『Apache Lucene Migration Guide』で変更の詳細を確認してください。Lucene Change Log にも追加の情報が記載されています。

- **@Field(indexNullAs=)** を使用してインデックスの null マーカー値をエンコードする場合、同じフィールドでインデックス化されるその他の値すべてに適合するマーカーのタイプである必要があります。たとえば、これまでは文字列「null」を使用して数字フィールドの null 値 をエンコードすることが可能でした。しかし、本リリースではこれができないようになりました。代わりに、**-1** などの **null** 値を表す数字を選択する必要があります。

- ファセッティングエンジンに大幅な改良が加えられました。多くの変更は API には影響しません。ファセッティングに使用する予定のフィールドすべてに **@Facet** または **@Facets** アノテーションを付ける必要があります。

Hibernate Search の名前変更および再パッケージされたクラス

以下は、名前変更または再パッケージされた Hibernate Search クラスのリストになります。

以前のパッケージおよびクラス	新しいパッケージおよびクラス
org.hibernate.search.Environment	org.hibernate.search.cfg.Environment
org.hibernate.search.FullTextFilter	org.hibernate.search.filter.FullTextFilter
org.hibernate.search.ProjectionConstants	org.hibernate.search.engine.ProjectionConstants
org.hibernate.search.SearchException	org.hibernate.search.exception.SearchException
org.hibernate.search.Version	org.hibernate.search.engine.Version

Lucene - 名前変更および再パッケージされたクラス

クエリーパーサーが新しいモジュールに移動されたため、パッケージが **org.apache.lucene.queryParser.QueryParser** から **org.apache.lucene.queryparser.classic.QueryParser** に変更になりました。

Lucene アナライザーの多くがリファクターされたため、パッケージが変更になりました。「Apache Lucene Documentation」を参照してください。

TokenizerFactory や **TokenFilterFactory** などの Apache Solr ユーティリティークラスの一部が Apache Lucene に移動されました。これらのユーティリティーやカスタムアナライザーがアプリケーションによって使用される場合は、Apache Lucene で新パッケージ名を探す必要があります。

詳細は『Apache Lucene Migration Guide』を参照してください。

Hibernate Search で非推奨となった API

Hibernate Search で非推奨となったインターフェース、クラス、列挙、アノテーションタイプ、メソッド、コンストラクター、および列挙定数の完全リストは、「Hibernate Search Deprecated API」を参照してください。

Hibernate Search で非推奨となったインターフェース

インターフェース	説明
org.hibernate.search.store.IndexShardingStrategy	Hibernate Search 4.4 で非推奨となりました。Hibernate Search 5 で削除される可能性があります。代わりに **ShardIdentifierProvider** を使用してください。
org.hibernate.search.store.Workspace	このインターフェースは移動され、パブリックでない API として考慮すべきです。詳細は、HSEARCH-1915 を参照してください。

Hibernate Search で非推奨となったクラス

クラス	説明
org.hibernate.search.filter.FilterKey	カスタムフィルターキーは非推奨となり、Hibernate Search 6 で削除される予定です。Hibernate Search 5.1 では、Lucene フィルターをキャッシュするキーは指定のフィルターパラメーターを基に自動的に算出されます。
org.hibernate.search.filter.StandardFilterKey	カスタムフィルターキーは非推奨となり、Hibernate Search 6 で削除される予定です。Hibernate Search 5.1 では、Lucene フィルターをキャッシュするキーは指定のフィルターパラメーターを基に自動的に算出されます。

Hibernate Search で非推奨となった列挙

列挙	説明
org.hibernate.search.annotations.FieldCacheType	非推奨となったため **CacheFromIndex** アノテーションを削除します。「Hibernate Search で非推奨となったアノテーション」を参照してください。

Hibernate Search で非推奨となったアノテーション

アノテーション	説明
org.hibernate.search.annotations.CacheFromIndex	アノテーションを削除してください。代替は必要ありません。
org.hibernate.search.annotations.Key	カスタムフィルターキャッシュキーは非推奨機能となり、Hibernate Search 6 で削除される予定です。Hibernate Search 5.1 では、フィルターキャッシュキーはフィルターパラメーターを基に自動的に判断されるため、キーオブジェクトを提供する必要がなくなりました。

Hibernate Search で非推奨となったメソッド

方法	説明
org.hibernate.search.FullTextSharedSessionBuilder.autoClose()	代替はありません。
org.hibernate.search.FullTextSharedSessionBuilder.autoClose(boolean)	代替はありません。
org.hibernate.search.cfg.IndexedMapping.cacheFromIndex(FieldCacheType...)	今後削除される予定で、代替はありません。
org.hibernate.search.cfg.EntityDescriptor.getCacheInMemory()	今後削除される予定で、代替はありません。
org.hibernate.search.cfg.ContainedInMapping.numericField()	代わりに、**field().numericField()** を呼び出します。
org.hibernate.search.cfg.EntityDescriptor.setCacheInMemory(Map<String, Object>)	今後削除される予定で、代替はありません。
org.hibernate.search.MassIndexer.threadsForSubsequentFetching(int)	このメソッドは削除される予定です。
org.hibernate.search.query.dsl.FuzzyContext.withThreshold(float)	**FuzzyContext.withEditDistanceUpTo(int)** を使用します。

Hibernate Search で非推奨となったコンストラクター

コンストラクター	説明
org.hibernate.search.cfg.NumericFieldMapping(PropertyDescriptor, EntityDescriptor, SearchMapping)	代わりに **NumericFieldMapping.NumericFieldMapping(String, PropertyDescriptor, EntityDescriptor, SearchMapping)** を使用します。

上級インテグレーターに影響する変更

ここでは、パブリック API の一部ではない変更について説明します。これらのアーティファクトは、Hibernate Search フレームワークを拡張するインテグレーターのみがアクセスできる必要があるため、通常の開発者には影響はないはずです。

- **IndexWriterSetting.MAX_THREAD_STATES** および **IndexWriterSetting.TERM_INDEX_INTERVAL** 列挙定数は非推奨になりました。これらは設定から読み取るプロパティーに影響します。実際にこれらの列挙定数がないと、**hibernate.search.Animals.2.indexwriter.term_index_interval = default** などの設定プロパティーが無視されます。これによる影響は、プロパティーが適用されないことのみです。

- **SearchFactoryIntegrator** インターフェースが非推奨になりました。**SearchIntegrator** を使用するよう、即座にすべてのコードを移行する必要があります。

- **SearchFactoryBuilder** クラスが非推奨になりました。代わりに **SearchIntegrationBuilder** を使用してください。
- **HSQuery.getExtendedSearchIntegrator()** メソッドが非推奨になりました。**SearchIntegrator** を使用できる可能性がありますが、このメソッドを削除することが推奨されます。
- **DocumentBuilderIndexedEntity.getFieldCacheOption()** メソッドが非推奨になりました。代替メソッドはありません。
- **BuildContext.getIndexingStrategy()** メソッドが非推奨になりました。代わりに **BuildContext.getIndexingMode()** を使用してください。
- **DirectoryHelper.getVerifiedIndexDir(String, Properties, boolean)** メソッドが非推奨になりました。代わりに **DirectoryHelper.getVerifiedIndexPath(java.lang.String, java.util.Properties, boolean)** を使用してください。
- 以下は、名前変更または再パッケージされた Hibernate Search クラスのリストになります。

以前のパッケージおよびクラス	新しいパッケージおよびクラス
org.hibernate.search.engine.impl.SearchMappingBuilder	org.hibernate.search.engine.spi.SearchMappingHelper
org.hibernate.search.indexes.impl.DirectoryBasedIndexManager	org.hibernate.search.indexes.spi.DirectoryBasedIndexManager
org.hibernate.search.spi.MassIndexerFactory	org.hibernate.search.batchindexing.spi.MassIndexerFactory
org.hibernate.search.spi.SearchFactoryBuilder	org.hibernate.search.spi.SearchIntegratorBuilder
org.hibernate.search.spi.SearchFactoryIntegrator	org.hibernate.search.spi.SearchIntegrator

5.10. エンティティー BEAN の JPA への移行

Java EE 7 では EJB エンティティー bean のサポートが任意となり、JBoss EAP 7 ではサポート対象外になりました。そのため、JBoss EAP 7 への移行時に CMP (Container-Managed Persistence) および BMP (Bean-Managed Persistence) エンティティー bean を書き直し、Java Persistence API (JPA) エンティティーを使用するようにする必要があります。

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 では、コードの CMP および BMP エンティティー bean を Java Persistence API (JPA) エンティティーに置き換える必要があります。JPA エンティティーは javax.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 JPA エンティティークラスの例になります。

```
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.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 version="2.1"
      xmlns="http://xmlns.jcp.org/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
      xsi:schemaLocation="
        http://xmlns.jcp.org/xml/ns/persistence
        http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
  <persistence-unit name="my-unique-persistence-unit-name">
    <properties>
      // properties...
    </properties>
  </persistence-unit>
</persistence>
```

JAP エンティティーの実施例は、JBoss EAP 7 に同梱される **bmt**、**cmt**、および **hibernate5** クイックスタートを参照してください。

5.11. JPA 永続プロパティーの変更

JBoss EAP 7.0 での JPA 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 JTA 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、または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、JPA 仕様 で定義されている正しいデフォルト動作です。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、JTA 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、JTA 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。

5.12. EJB クライアントコードの移行

5.12.1. JBoss EAP 7 での EJB クライアントの変更

JBoss EAP 7 ではデフォルトのリモートコネクターおよびポートが変更になりました。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。

migrate 操作を使用してサーバー設定を移行した場合は、旧設定は保持されるため以下の変更を行う必要はありません。しかし、新しい JBoss EAP 7 のデフォルト設定で実行する場合は以下の変更を行う必要があります。

5.12.1.1. 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 7 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=8080
remote.connection.default.connect.options.org.xnio.Options.SASL_POLICY_NOANONYMOUS=false
```

5.12.1.2. デフォルトコネクターの更新

新しい JBoss EAP 7 設定で実行している場合、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、使用するライブラリーと接続するサーバーの JBoss EAP リリースが異なるクライアントに影響します。

- クライアントアプリケーションが JBoss EAP 6 の EJB 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 EJB 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
  ```

5.12.2. 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
```

5.12.3. JBoss EAP 7.1 に導入された EJB Client のその他の変更

JBoss EAP 7.0 には JBoss EJB Client 2.1.4 が同梱されていましたが、JBoss EAP 7.1 以上には JBoss EJB Client 4.0.x が同梱され、API に複数の変更が追加されました。

- 以下の新しいメソッドに **org.ejb.client.EJBClientInvocationContext** クラスが追加されました。

方法	タイプ	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。

方法	タイプ	説明
isClientAsync()	boolean	メソッドがクライアント非同期 (client-asynchronus) としてマークされているかどうかを判断します。 マークされていると、サーバー側のメソッドが非同期であるかどうかに関わらず、呼び出しは非同期になります。
isIdempotent()	boolean	メソッドが冪等 (idempotent) としてマークされているかどうかを判断します。 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があります。
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。

方法	タイプ	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どうかを判断します。
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。

- 特権のある EJB 操作へのアクセスを制御するために、**java.security.Permission** のサブクラスである新しい **org.ejb.client.EJBClientPermission** 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。

    - **EJBClientPermission(String name)**
    - **EJBClientPermission(String name, String actions)**
  - 以下のメソッドを提供します。

方法	タイプ	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。
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 EJB メソッドを見つけるために、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。

方法	タイプ	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。

方法	タイプ	説明
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。

- 失敗したケースに対して、**org.jboss.ejb.client.EJBReceiverInvocationContext.ResultProducer.Failed** クラスが新たに導入されました。
  - 以下のコンストラクターを提供します。
    - **Failed(Exception cause)**
  - 以下のメソッドを提供します。

方法	タイプ	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、**org.jboss.ejb.client.EJBReceiverInvocationContext.ResultProducer.Immediate** クラスが新たに導入されました。
  - 以下のコンストラクターを提供します。
    - **Failed(Exception cause)**
  - 以下のメソッドを提供します。

方法	タイプ	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。

- URIアフィニティーを指定するために、**org.jboss.ejb.client.Affinity** のサブクラスである **org.jboss.ejb.client.URIAffinity** クラスが新たに導入されました。
  - 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。

方法	タイプ	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** クラスによって以下のメソッドが非推奨になりました。
  - **toAbstractEJBMetaData()**
  - **EJBMetaDataImpl(AbstractEJBMetaData<?,?>)**

5.12.4. JBoss EAP 7.2 で必要な EJB クライアントの変更

JBoss EAP 7.2 では **org.apache.santuario.xmlsec** モジュールが 2.0.8 から 2.1.1 にアップグレードされ、**PicketLinkSTS** のリモーティングで回帰が発生します。この問題により以下のランタイム例外が発生します。

```
java.lang.IllegalArgumentException: ELY05131: Invalid ASCII control "0xA"
```

これは、アサーションの生成された **SignatureValue** のフォーマットの変更が原因です。これまでのリリースでは、生成された値は以下の例と似ていました。

```
<dsig:SignatureValue
xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">cUNpFJIZlLYrBDZtQSTDrq2K6PbnAHyg2qbx/D5F
uB4XMjdQ5oxQjkMejLyelnA7s4GFusoLhahlqlTOT8UrOyxrR4yYAmJ/e5s+f4gys926+tbiraT/3/wG8wM/L
vcjvk5Ap69zODuRYpypsWfA4jrI7TTBXVPGy8g4KUdnFviUiTuFTc2Ghgxp53AmUuLis/THyP28jE7+28//
q8bi/bQrFwHC6tWX67+NK1duFCOcQ6IPIKeVrePZz55Ivgl+WWdkF6uYCz5IdMzurhzmeQ3K8DAMIxz/
MG67VWJlOnuGNWF7nmdye5zd9AFcRsr1XadvZJCbGNfuc89AL5inCg==</dsig:SignatureValue>
```

JBoss EAP 7.2 では、生成された値に無効な非表示の ASCII **0xD** キャリッジリターンおよび **0xD** ラインフィード制御文字のインスタンスが含まれます。

```
<dsig:SignatureValue
xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">cUNpFJIZlLYrBDZtQSTDrq2K6PbnAHyg2qbx/D5F
uB4XMjdQ5oxQjkMejLyelnA7s4GFusoLhahl&#13;
qlTOT8UrOyxrR4yYAmJ/e5s+f4gys926+tbiraT/3/wG8wM/Lvcjvk5Ap69zODuRYpypsWfA4jrI&#13;
7TTBXVPGy8g4KUdnFviUiTuFTc2Ghgxp53AmUuLis/THyP28jE7+28//q8bi/bQrFwHC6tWX67+N&#13;
```

```
K1duFCOcQ6IPIKeVrePZz55Ivgl+WWdkF6uYCz5IdMzurhzmeQ3K8DAMIxz/MG67VWJIOnuGNWF7&
#13;
nmdye5zd9AFcRsr1XadvZJCbGNfuc89AL5inCg==</dsig:SignatureValue>
```

上記のランタイム例外が発生した場合はクライアントコードを更新し、発生した非表示の ASCII 文字を返されたアサーション文字列から削除する必要があります。

たとえば、現在のコードが以下の例と同様であるとします。

```
WSTrustClient client = new WSTrustClient("PicketLinkSTS", "PicketLinkSTSPort",
        "http://localhost:8080/picketlink-sts/PicketLinkSTS", new
WSTrustClient.SecurityInfo(username, password));
Element assertion = client.issueToken(SAMLUtil.SAML2_TOKEN_TYPE);

// Return the assertion as a string
String assertionString = DocumentUtil.getNodeAsString(assertion);
...
properties.put("remote.connection.main.password", assertionString);
```

コードを 1 行追加して、以下のように無効な非表示の ASCII **0xD** キャリッジリターンおよび **0xA** ラインフィード文字を削除する必要があります。

```
WSTrustClient client = new WSTrustClient("PicketLinkSTS", "PicketLinkSTSPort",
        "http://localhost:8080/picketlink-sts/PicketLinkSTS", new
WSTrustClient.SecurityInfo(username, password));
Element assertion = client.issueToken(SAMLUtil.SAML2_TOKEN_TYPE);

// Return the assertion as a string, stripping the invalid hidden ASCII characters
String assertionString = DocumentUtil.getNodeAsString(assertion).replace(String.valueOf((char)
0xA), "").replace(String.valueOf((char) 0xD), "");
...
properties.put("remote.connection.main.password", assertionString);
```

5.13. WILDFLY 設定ファイルを使用するようクライアントを移行

EJB や naming などの JBoss EAP クライアントライブラリーは、リリース 7.1 まで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 以前では、Java EJB クライアントの新しい **InitialContext** の作成には、**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ミングでプロパティーファイルを設定しました。

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:1.2">
        <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:3.0">
        <connections>
            <connection uri="remote+http://127.0.0.1:8080" />
        </connections>
    </jboss-ejb-client>
</configuration>
```

wildfly-config.xml ファイルを使用して Elytron クライアントに対してクライアント認証を設定する方法については、JBoss EAP『How to Configure Identity Management』の「**Configure Client Authentication with Elytron Client**」を参照してください。

wildfly-config.xml ファイルを使用して実行できるクライアント設定の種類に関する詳細は、JBoss EAP『開発ガイド』の「**wildfly-config.xml** ファイルを使用したクライアント設定」参照してください。

5.14. 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** インターフェースを Java EE Product Providors が実装する必要がありました。JBoss EAP 6 の場合、ZIP または JAR アーカイブでバンドルされる **deployment-plan.xml** という名前の XML 記述子によってデプロイメント計画が識別されます。

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管理対象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。

5.15. カスタムアプリケーションバルブの移行

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。

重要

jboss-web.xml ファイルに定義されたカスタムバルブおよびバルブは、対応する Undertow のビルトインハンドラーで書き直すか、置き換える必要があります。Undertow ハンドラーへのバルブのマッピングに関する詳細は「JBoss Web バルブの移行」を参照してください。

認証バルブは、Undertow ビルトイン認証メカニズムを使用して手動で置き換える必要があります。

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 では、Undertow ハンドラーを使用して定義することもできます。

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『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。

5.16. セキュリティーアプリケーションの変更

JBoss Web を Undertow で置き換えるには、JBoss EAP 7 のセキュリティー設定における変更が必要になります。

5.16.1. 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 『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。

5.16.2. PicketLink の変更

SSO の SAML v2 設定で必要となる変更の情報は、JBoss EAP『How To Set Up SSO with SAML v2』の「**Changes from Previous Versions of JBoss EAP**」を参照してください。

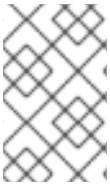
5.16.3. その他のセキュリティーアプリケーションの変更

Kerberos による SSO 設定についての相違点は、JBoss EAP『How to Set Up SSO with Kerberos』の「**Differences from Configuring Previous Versions JBoss EAP**」を参照してください。

5.17. JBOSS LOGGING の変更

アプリケーションが 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は JBoss EAP 7 では非推奨になった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5.1 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

5.18. JAVASERVER FACES (JSF) のコード変更

JSF 1.2 のサポート停止

JBoss EAP 6.4 では、**jboss-deployment-structure.xml** ファイルを作成して、アプリケーションデプロイメントで JSF 1.2 の使用を継続することができました。

JBoss EAP 7.2 には JSF 2.3 が含まれ、JSF 1.2 API はサポート対象外になりました。アプリケーションが JSF 1.2 を使用する場合、JSF 2.3 を使用するようアプリケーションを書き直す必要があります。

5.19. モジュールクラスローティングの変更

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『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。

5.20. アプリケーションクラスタリングの変更

5.20.1. 新しいクラスタリング機能の概要

以下のリストは、アプリケーションを JBoss EAP 6 から JBoss EAP 7 へ移行するときに注意する必要がある新しいクラスタリング機能の一部を示しています。

- 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『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『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『Developing EJB Applications 』で「**Clustered Singleton MDBs**」を参照してください。
- JBoss EAP 7 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 『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」を参照してください。

本セクションの残りの部分では、クラスタリングの変更がアプリケーションの JBoss EAP 7 への移行に及ぼす可能性のある影響について説明します。

5.20.2. 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>
```

以下の表は、**jboss-web.xml** ファイルの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

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/>	この設定要素とサブ要素は JBoss EAP 7 では使用されないようになりました。
<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/>	新しい実装では複数のサブ要素が非推奨になりました。
<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『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。

設定要素	変更の説明
<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。

5.20.3. 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 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。

- EJB 3.2 仕様に新たに導入された **@Stateful(passivationCapable=false)** を使用して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化します。
- 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。

5.20.4. 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** インターフェースを使用すると、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** インターフェースを使用すると、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『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。

5.20.5. 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。プライベートの **org.jboss.as.clustering.singleton.*** クラスを使用した場合、アプリケーションを JBoss EAP 7 に移行するときに新しいパブリックの **org.wildfly.clustering.singleton.*** パッケージを使用するよう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『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『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。

第6章 その他の変更

6.1. JBOSS EAP ネイティブおよび APACHE HTTP SERVER の提供に関する変更

JBoss EAP 7 ネイティブは、これまでのリリースとは異なる方法で提供されます。一部のネイティブは、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セットである、新規の Red Hat JBoss Core Services 製品に同梱されるようになりました。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別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 7
メッセージング用 AIO ネイティブ	別個の "ネイティブユーティリティー" ダウンロードで製品とともに提供。	JBoss EAP ディストリビューションに含まれます。他のダウンロードは必要ありません。
Apache HTTP Server	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 "ネイティブユーティリティー" 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。
OpenSSL	別個の "ネイティブユーティリティー" 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。
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。	これは JBoss EAP 7 では廃止されました

- 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 チャンネルから Apache HTTP Server を実行し、JBoss EAP 7 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動作するよう JBoss EAP 7 を設定できます。詳細は、JBoss EAP 『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」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。サポートされる構成の一覧は、JBoss EAP 『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 Installation Guide』の「**JBoss Core Services**」を参照してください。

6.2. AMAZON EC2 でのデプロイメントの変更

JBoss EAP 7 では、Amazon Machine Images (AMI) に複数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。

- 非クラスターおよびクラスターの JBoss EAP インスタンスおよびドメインを Amazon EC2 で起動する方法が大幅に変更されました。
- JBoss EAP 6 では JBoss EAP の設定に **User Data:** フィールドが使用されました。JBoss EAP 7 では、**User Data:** フィールドの設定を分析した AMI スクリプトと、インスタンスの起動時に自動的にサーバーを起動した AMI スクリプトが削除されました。
- Red Hat JBoss Operations Network エージェントは、以前のリリースの JBoss EAP にインストールされていました。JBoss EAP 7 では、これを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。

6.3. 共有モジュールが含まれるアプリケーションのアンデプロイ

JBoss EAP 7.1 サーバーと Maven プラグインが変更されたため、アプリケーションをアンデプロイしようとすると以下のエラー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 7.2 まで該当します。

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
```

このアンデプロイの動作はより適切で、デプロイメントが不安定な状態にならないようにします。

6.4. JBOSS EAP 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 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、『Add-User ユーティリティーのパスワード制限の設定』の「設定ガイド」を参照してください。

6.5. OSGI サポートの廃止

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。

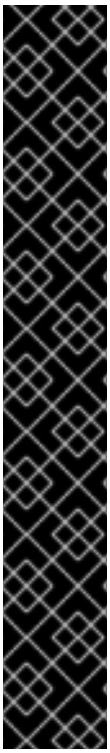
JBoss EAP 6.1.0 では、**configadmin** および **osgi** 拡張モジュールとスタンドアロンサーバーのサブシステム設定が個別の **EAP_HOME/standalone/configuration/standalone-osgi.xml** 設定ファイルに移されました。サポート対象外であるこの設定ファイルを移行すべきではないため、JBoss OSGi サポートの廃止はスタンドアロンサーバー設定の移行には影響しないはずです。他のスタンドアロン設定ファイルを編集して **osgi** または **configadmin** を設定した場合は、その設定を削除する必要があります。

管理対象ドメインでは、 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。この設定は JBoss EAP 7 ではサポートされないため、削除する必要があります。

第7章 ELYTRON への移行

7.1. ELYTRON の概要

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管理対象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。



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。また、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 にはアクセスすることができます。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。

PicketBox ベースのレガシー **security** サブシステムを使用する JBoss EAP 7.0 サーバー設定およびデプロイメントは、JBoss EAP 7.1 以上のリリースを変更せずに実行できるはずです。PicketBox は継続してセキュリティードメインをサポートするため、アプリケーションは既存のログインモジュールを継続して使用できます。セキュリティーのために管理レイヤーによって使用されるセキュリティーレルムも Elytron に引き継がれ、エミュレートされます。これにより、**elytron** サブシステムとレガシーの **security** サブシステムの両方で認証を定義でき、両方のサブシステムを並行して使用できます。アプリケーションが Elytron およびレガシーセキュリティーを使用するよう設定する方法は、『How to Configure Identity Management』の「**Configure Web Applications to Use Elytron or Legacy Security for Authentication**」を参照してください。

PicketBox 認証のサポートは継続されますが、アプリケーションを移行する準備ができたら Elytron に移行することが推奨されます。Elytron セキュリティーを使用する利点の 1 つは、サーバーとアプリケーション全体で一貫したセキュリティーソリューションを提供することです。Elytron を使用するよう PicketBox の認証および承認を移行する方法については、本ガイドの「認証設定の移行」を参照してください。

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 『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。

重要

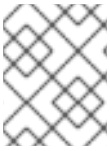
デプロイメントでレガシー **security** サブシステムと Elytron の両方を使用する場合、異なるセキュリティーアーキテクチャーを使用したデプロイメント間での呼び出しはサポートされないため注意してください。

これらのサブシステムを並行して使用する方法は、『How to Configure Identity Management』の「**Using Elytron and Legacy Security Subsystems in Parallel**」を参照してください。

7.2. セキュアな VAULT およびプロパティーの移行

7.2.1. クレデンシャルストレージをセキュア化する vault の移行

JBoss EAP 7.0 のレガシー **security** サブシステムでプレーンテキストの文字列暗号を保存するために使用された vault は、新たに設計されたクレデンシャルストアを使用して文字列を保存する JBoss EAP 7.1 以上の Elytron と互換性がありません。クレデンシャルストアは、JBoss EAP 設定ファイル外のストレージファイルでクレデンシャルを安全に暗号化します。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 vault 式を使用して機密でないデータをパラメーター化した場合、そのデータを Elytron セキュリティープロパティー に置き換えることが推奨されます。

レガシー **security** サブシステムの使用を継続する場合、vault データを編集または更新する必要はないはずです。しかし、アプリケーションを移行して Elytron を使用する計画がある場合、既存の vault をクレデンシャルストアに変換し、**elytron** サブシステムが処理できるようにする必要があります。クレデンシャルストアに関する詳細は、『How to Configure Server Security』の「**Credential Stores**」を参照してください。

WildFly Elytron Tool を使用した vault データの移行
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。**EAP_HOME/bin** ディレクトリーにある**elytron-tool** スクリプトを実行してツールを実行します。

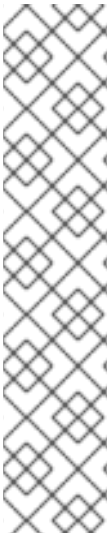
```
$ EAP_HOME/bin/elytron-tool.sh vault VAULT_ARGUMENTS
```

java -jar コマンドを実行してツールを実行することもできます。

```
$ java -jar EAP_HOME/bin/wildfly-elytron-tool.jar 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
```



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)。
- **--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。

重要

クレデンシャルストアはパスワードのセキュア化のみに使用できることに注意してください。管理モデルで使用できる vault 式機能をサポートしません。

以下の移行オプションの 1 つを選択します。

- 単一のセキュリティー vault をクレデンシャルストアに移行
- 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

単一のセキュリティー **vault** をクレデンシャルストアに移行
以下は、単一のセキュリティー vault をクレデンシャルストアに変換するために使用されるコマンドの例になります。

```
$ 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"})
```

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
--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。

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。

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"})
-----------------------------------
```

7.2.2. セキュリティープロパティーの Elytron への移行

ここの例では、**group.name** および **encoding.algorithm** セキュリティープロパティーは以下のようにレガシー **security** サブシステムで **security-properties** として定義されていることを仮定します。

例: security サブシステムに定義されたセキュリティープロパティー

```
<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>
```

同じセキュリティープロパティーを **elytron** サブシステムで定義するには、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で **elytron** サブシステムの以下の **security-properties** を設定します。

```
<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)
```

7.3. 認証設定の移行

7.3.1. プロパティーベースの認証および承認の Elytron への移行

7.3.1.1. PicketBox プロパティーベースの設定を Elytron に移行

ここでは、PicketBox プロパティーベースの認証を Elytron に移行する方法を説明します。PicketBox セキュリティードメインのみを Elytron に公開してプロパティーベースの認証を 部分的に移行 するか、または Elytron を使用するようプロパティーベースの認証設定を 完全に移行 するかを選択できます。

以下の手順は、移行を行うデプロイされた web アプリケーションが、フォームベースの認証を必要とするよう設定されていると仮定します。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。これらの例では、以下の管理 CLI コマンドを使用してセキュリティードメインがレガシー **security** サブシステムで定義されたことも仮定します。

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>
```

以下の移行オプションの 1 つを選択します。

- PicketBox セキュリティードメインを Elytron に公開して部分的に移行
- プロパティーベースの認証を Elytron に完全移行

PicketBox セキュリティードメインを Elytron に公開して部分的に移行
PicketBox セキュリティードメインを Elytron セキュリティーレルムとして公開し、Elytron 設定に結合

するすることができますが、これによりレガシー **security** サブシステムとの依存関係が作成されます。プロパティーベースの認証のみを移行する場合は、アプリケーションを Elytron に完全移行 してレガシー **security** サブシステムとの不必要な依存関係が作成されないようにすることが推奨されます。しかし、Elytron を使用するようアプリケーションを完全移行できない場合は、部分的な移行で部分的に対処できます。

この手順に従って、既存の PicketBox セキュリティーレルム設定を Elytron セキュリティーレルムとして追加します。

1. Elytron セキュリティーレルムへのマッピングをレガシー **security** サブシステム内に追加します。

```
/subsystem=security/elytron-realm=application-security:add(legacy-jaas-config=application-security)
```

これにより、以下の Elytron セキュリティーレルムがサーバー設定ファイルの **security** サブシステムに設定されます。

```
<subsystem xmlns="urn:jboss:domain:security:2.0">
  ...
  <elytron-integration>
    <security-realms>
      <elytron-realm name="application-security" legacy-jaas-config="application-security"/>
    </security-realms>
  </elytron-integration>
  ...
</subsystem>
```

2. エクスポートされたセキュリティーレルムを参照する **elytron** サブシステムのセキュリティードメインを定義します。

```
/subsystem=elytron/security-domain=application-security:add(realms=[{realm=application-security}], default-realm=application-security, 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。

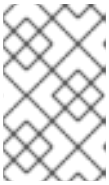
```
<subsystem xmlns="urn:wildfly:elytron:4.0" final-providers="combined-providers" disallowed-providers="OracleUcrypto">
  ...
  <security-domains>
    ...
    <security-domain name="application-security" default-realm="application-security" permission-mapper="default-permission-mapper">
      <realm name="application-security"/>
    </security-domain>
  </security-domains>
  ...
</subsystem>
```

3. **undertow** サブシステムにて、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新たに定義されたセキュリティードメインにマップします。

```
/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:wildfly:elytron:4.0">
  ...
  <application-security-domains>
    <application-security-domain name="application-security" security-domain="application-security"/>
  </application-security-domains>
  ...
</subsystem>
```



注記

この設定の指定前にアプリケーションがデプロイされた場合、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。

4. 以下の管理 CLI コマンドを使用して、マッピングがデプロイメントに適用されたことを確認します。この例で使用されるデプロイメントは **HelloWorld.war** です。このコマンドの出力は、このデプロイメントが Elytron マッピングを参照していることを表しています。

```
/subsystem=undertow/application-security-domain=application-security:read-resource(include-runtime=true)

{
"outcome" => "success",
    "result" => {
        "enable-jacc" => false,
        "http-authentication-factory" => undefined,
        "override-deployment-config" => false,
        "referencing-deployments" => ["HelloWorld.war"],
        "security-domain" => "application-security",
        "setting" => undefined
    }
}
```

この段階では、この前に定義されたセキュリティードメインは **LoginModule** 設定に使用されますが、認証を引き継ぐ Elytron コンポーネントによってラップされます。

プロパティーベースの認証を **Elytron** に完全移行
以下の手順に従って、PicketBox プロパティーベースの認証を Elytron に完全移行します。この手順は、本セクションの始めで説明したレガシー設定を使用し、前述の部分的な移行を行っていないことを仮定します。この手順の完了後も、レガシー security サブシステムに存在するセキュリティードメイン定義はすべて Elytron 設定から完全に独立した状態を維持します。

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:7.0">
  ...
  <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すが、認証には Elytron コンポーネントのみが使用されます。

7.3.1.2. レガシープロパティーベースの設定を Elytron に移行

ここでは、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 Elytron にロードするレガシーセキュリティーレルムの移行方法を説明します。通常このタイプのレガシーセキュリティーレルムは、管理インターフェースまたはリモーティングコネクターのいずれかをセキュアするために使用されます。

これらの例では、レガシー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定義されることを仮定します。

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 つが、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す。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。

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)
/subsystem=remoting/http-connector=http-remoting-connector:undefine-
attribute(name=security-realm)
```

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レガシーセキュリティーレルムの参照を削除し、 **http-interface** を Elytron でセキュアにします。

```
/core-service=management/management-interface=http-interface:write-attribute(name=http-
authentication-factory, value=application-security-http)
/core-service=management/management-interface=http-interface:write-attribute(name=http-
upgrade.sasl-authentication-factory, value=application-security-sasl)
/core-service=management/management-interface=http-interface:undefine-
attribute(name=security-realm)
```

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

管理インターフェースをセキュア化するとき、例で使用されている名前よりも適切な名前を選択してください。

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。

7.3.2. LDAP 認証設定の Elytron への移行

ここでは、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するために、レガシー LDAP 認証を移行する方法について説明します。特に、セキュリティードメインおよび認証ファクトリーの定義方法と認証のためにこれらをマップする方法に関しては、「プロパティーベースの認証および承認の Elytron への移行」に記載されている多くの情報がここでも適用されます。ここでは、同じ手順は再度説明しないため、『プロパティーベースの認証および承認の Elytron への移行』を読んでから作業を続行してください。

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。

- 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。

- 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>
```

- 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>
```

7.3.2.1. 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 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。

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、本ガイドの「プロパティーベースの認証および承認の 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。

7.3.3. データベース認証設定の Elytron への移行

ここでは、JDBC データソースベースの PicketBox 認証を Elytron に移行する方法について説明します。特に、セキュリティードメインおよび認証ファクトリーの定義方法と認証のためにこれらをマップする方法に関しては、「プロパティーベースの認証および承認の Elytron への移行」に記載されている多くの情報がここでも適用されます。ここでは、同じ手順は再度説明しないため、『プロパティーベースの認証および承認の Elytron への移行』を読んでから作業を続行してください。

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。それを使用してユーザー名とパスワードを検証し、ロールを割り当てます。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>
```

7.3.3.1. レガシーデータベース認証の Elytron への移行

前述のデータベース認証例の設定を Elytron に移行するには、JDBC レルムを定義して Elytron による JDBC データソース へのアクセスを有効にする必要があります。

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。

7.3.4. 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。ここでは、Kerberos HTTP および Kerberos SASL 認証の移行方法を説明します。

ここで使用する例では、以下のシステムプロパティーを使用して Kerberos が設定されたことを仮定します。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>
```

以下の移行オプションの 1 つを選択します。

- Kerberos HTTP 認証の移行
- Kerberos リモーティング SASL 認証の移行

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 管理インターフェースの SPNEGO 認証を有効にできます。

例: HTTP 管理インターフェースの 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。

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すると、管理インターフェースとアプリケーションの両方をセキュアにすることができます。

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 管理インターフェースを更新することができます。この手順については、本書の「プロパティーベースの認証および承認の Elytron への移行」に記載されています。

Kerberos リモーティング SASL 認証
ネイティブ管理インターフェースなど、リモーティング認証に使用される Kerberos / GSSAPI SASL 認証のレガシーセキュリティーレルムを定義することが可能です。

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>
```

Kerberos リモーティング SASL 認証の Elytronへの移行
同等の Elytron 設定を定義する手順は、「Kerberos HTTP 認証の移行」の手順と大変似ています。

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>
```

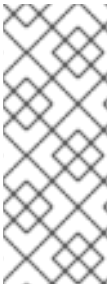
これで、管理インターフェースまたはリモーティングコネクターが更新され、SASL 認証ファクトリーを参照するようになりました。

ここで定義された 2 つの Elytron の例は、共有されたセキュリティードメインおよびセキュリティーレルムを使用し、各自が独自の Kerberosセキュリティーファクトリーを参照するプロトコル固有の認証ファクトリーを使用するよう組み合わせることも可能です。

7.3.5. 複合ストアの Elytron への移行

ここでは、PicketBox または 複数のアイデンティティーストアを使用する レガシーセキュリティーレルム設定を Elytron 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。

- プロパティーベースの認証および承認の Elytron への移行
- LDAP 認証設定の Elytron への移行

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>
```

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。

レガシーセキュリティーレルム複合ストア設定
この場合の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。

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。

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。

7.3.6. 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

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す。これにより、ユーザーはメモリーのアイデンティティーデータにアクセスでき、それ以外ではアイデンティティーストアへ直接アクセスできないようにします。同様の設定を Elytronで実現することができます。ここでは、Elytron 使用時のセキュリティードメインキャッシングの設定方法について説明します。

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、キャッシングを有効にする PicketBox セキュリティードメインの設定方法を表しています。

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。

Elytron のキャッシュ済みセキュリティードメイン設定
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。

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

このステップでは、元のレルムではなく**caching-realm** を参照することが重要になります。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>
...
```

7.3.7. JACC セキュリティーの Elytron への移行

JBoss EAP はデフォルトではレガシー **security** サブシステムを使用して、JACC (Java Authorization Contract for Containers) 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、JACC 仕様を基に組み込みのポリシープロバイダーを提供します。サーバーを設定して Elytron が JACC 設定およびその他のポリシーを管理できるようにする前に、以下の管理 CLIコマンドを使用して最初にレガシー **security** サブシステムの JACC を無効にする必要があります。

```
/subsystem=security:write-attribute(name=initialize-jacc, value=false)
```

この作業を怠ると、次のエラーメッセージがサーバーログに出力されます: **MSC000004: Failure during stop of service org.wildfly.security.policy: java.lang.StackOverflowError**

JACC を有効にする方法や、**elytron** サブシステムで JACC ポリシープロバイダーを定義する方法の詳細は、JBoss EAP『開発ガイド』の「**elytron** サブシステムを使用した **JACC** の有効化」を参照してください。

7.4. アプリケーションクライアントの移行

7.4.1. 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。

以下の例は、ユーザークレデンシャルのプロパティーおよび接続するネーミングプロバイダーの URL のプロパティーを指定して **InitialContextFactory** クラスが作成されることを仮定します。

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

7.4.1.1. 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

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。

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:1.2">
    <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");
...
```

7.4.1.2. 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

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);
```

7.4.2. EJB 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 EJB 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 EJB を検索し、メソッドの 1 つを呼び出します。

例: リモート EJB 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 EJB 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 EJB クライアントの移行
- プログラミングを使用した EJB クライアントの移行

7.4.2.1. 設定ファイルを使用した EJB 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。

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:1.2">
    <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 EJB 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。

7.4.2.2. プログラミングを使用した EJB クライアントの移行

この方法では、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 EJB
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 EJB 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。

7.5. SSL 設定の移行

7.5.1. 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 接続をセキュアにした場合、本セクションの情報を使用してその設定を Elytron に移行できます。

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>
```

以下の手順に従って、Elytron を使用して同じ設定を実現します。

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。

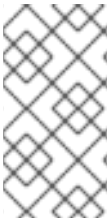
```
<https-listener name="https" socket-binding="https" ssl-context="LocalhostSslContext" enable-
http2="true"/>
```

詳細は、『How to Configure Server Security』の 「Elytron Subsystem」および「**How to Secure the Management Interfaces**」を参照してください。

7.5.2. 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**

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。

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
```

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:7.0">
...
<https-listener name="https" socket-binding="https" ssl-context="LocalhostSslContext" enable-
http2="true"/>
...
</subsystem>
```

レルムおよびドメイン
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>
```

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。

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。

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)
```

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>
```

第8章 JBOSS EAP の旧リリースからの移行

8.1. JBOSS EAP 5 から JBOSS EAP 7 への移行

本ガイドは、JBoss EAP 6 のアプリケーションを JBoss EAP 7 で正常に実行するために必要な変更に重点を置いています。アプリケーションを直接 JBoss EAP 5 から JBoss EAP 7 に移行する計画がある場合、移行の計画や実行に役立つリソースが複数あります。以下を行うことが推奨されます。

1. 本ガイドの「各リリースに追加された変更の概要」で、JBoss EAP の各リリースに追加された変更の概要を確認します。
2. JBoss EAP 6 の『移行ガイド』と本ガイドを読み、両方の内容を把握します。
3. 特定のコンポーネントや機能に関する移行情報のリファレンスとして、JBoss EAP 5 コンポーネントのアップグレードリファレンス を使用します。
4. ルールベースの Red Hat Application Migration Toolkit は、直接 JBoss EAP 5 から JBoss EAP 7 に移行するのに役立つツールを継続して追加します。これらのツールを使用してアプリケーションを分析し、JBoss EAP 7 への移行に必要な変更に関する詳細なレポートを生成します。詳細は、「Red Hat Application Migration Toolkit を使用した移行のアプリケーションの分析 」を参照してください。
5. カスタマーポータルのナレッジベース では、現在 JBoss EAP 5 から JBoss EAP 6 への移行に役立つ記事やソリューションを利用できます。JBoss EAP 5 から JBoss EAP 7 への移行に関するコンテンツを徐々に追加する計画があります。

8.2. 各リリースに追加された変更の概要

移行を計画する前に、JBoss EAP 6 と JBoss EAP 7 で追加された変更について認識できるようにしてください。

『JBoss EAP 6 移行ガイド』は、JBoss EAP 5 と JBoss EAP 6 との間で追加された変更を取り上げています。以下に、JBoss EAP 6 に追加された最も重要な変更の一覧を示します。

- モジュラーサービスコンテナーに構築される新しいアーキテクチャーを実装しました。
- Java Enterprise Edition 6 仕様の認定実装でした。
- ドメイン管理、新しいデプロイメント設定、および新しいファイルディレクトリー構造とスクリプトが導入されました。
- 新しい移植可能な JNDI ネームスペースを標準化しました。

JBoss EAP 6 に加えられた変更の詳細なリストは、JBoss EAP 6『移行ガイド』の「**JBoss EAP 6** の新機能と変更内容」を参照してください。

JBoss EAP 7 は、JBoss EAP 6 と同じモジュラー構造に構築され、同じドメイン管理、デプロイメント設定、ファイルディレクトリー構造、およびスクリプトが含まれます。また、同じ標準化された JNDI ネームスペースも使用します。しかし、JBoss EAP 7 には以下の変更が追加されています。

- Java Enterprise Edition 7 仕様のサポートが追加されています。
- Web サーバーが Undertow に置き換えられました。
- JacORB IIOP 実装が OpenJDK ORB ダウンストリームブランチに置き換えられました。

- Apache ActiveMQ Artemis が新しいメッセージングプロバイダーとして含まれています。
- **cmp**、**jaxr**、および **threads** サブシステムが削除されました。
- DJB エンティティー bean のサポートが削除されました。

変更の完全リストは、JBoss EAP 7 の新機能 を参照してください。

8.3. MIGRATION GUIDE (移行ガイド) の内容確認

各リリースの Migration Guide (移行ガイド) の内容をすべて確認し、追加された機能や非推奨となった機能について認識し、そのリリースの既存アプリケーションの実行に必要なサーバーの設定およびアプリケーションの変更について理解するようにしてください。

JBoss EAP 6 と JBoss EAP 7 の間では基盤のアーキテクチャーは変更されていないため、JBoss EAP 6 の『移行ガイド』に記載されている多くの変更内容は JBoss EAP 7 にも該当します。たとえば、「ほとんどのアプリケーションで必要な変更」に記載されている変更は、JBoss EAP 6 で追加された基盤のアーキテクチャーの変更に関連し、本リリースにも該当します。新しいモジュラークラスローティングシステムへの変更は重要で、ほぼすべての JBoss EAP 5 アプリケーションのパッケージ化や依存関係に影響します。「アプリケーションのアーキテクチャーやコンポーネントによって異なる変更」に記載されている変更の多くは、JBoss EAP 7 にも該当します。しかし、JBoss EAP 7 では web サーバー、ORB、およびメッセージングプロバイダーが置き換えられ、**cmp**、**threads**、および **jaxr** サブシステムが削除され、さらに EJB エンティティー bean のサポートが削除されたため、これらのコンポーネントに関する変更については本書を参考にする必要があります。移行を始める前に、本ガイドの「サーバー設定の変更」および「アプリケーション移行の変更」に注意してください。

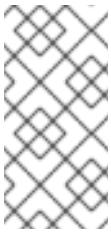
8.4. JBOSS EAP 5 コンポーネントのアップグレードリファレンス

以下の表を使用して、特定の機能またはコンポーネントを JBoss EAP 5 から JBoss EAP 7.2 に移行する方法を検索してください。

JBoss EAP 5の機能またはコンポーネント	変更の概要および移行情報の場所
アプリケーションのパッケージ化およびクラスローティング	JBoss EAP 6 では、以前の階層的なクラスローディング構造が JBoss Modules を基にしたモジュラーアーキテクチャーに置き換えられました。新しいモジュラークラスローディング構造の導入に伴い、アプリケーションのパッケージ化も変更になりました。このアーキテクチャーは JBoss EAP 7 でも使用されています。新しいモジュラーアーキテクチャーに関する情報は、JBoss EAP 7.2『**開発ガイド**』の以下の章を参照にしてください。 - クラスローディングとモジュール 新しいモジュラーアーキテクチャーのアプリケーションの更新および再パッケージ方法に関する情報は、JBoss EAP 6『移行ガイド』の以下の項を参照してください。 - クラスローディングの変更

JBoss EAP 5の機能またはコンポーネント	変更の概要および移行情報の場所
アプリケーション設定ファイル	モジュラークラスローディングを使用する JBoss EAP 6 の変更に伴い、1つ以上のアプリケーション設定ファイルを作成または編集して依存関係を追加したり、自動的な依存関係がロードされないようにする必要がある場合があります。これは JBoss EAP 7 でも変更ありません。詳細は、JBoss EAP 6『**移行ガイド**』の以下の項を参照してください。 ● 設定ファイルの変更
キャッシュおよびInfinispan	JBoss EAP 6 では、サーバーによる内部使用のみで JBoss Cache が Infinispan に置き換えられました。アプリケーションコードで JBoss Cache を置き換える方法については、JBoss EAP 6『**移行ガイド**』の以下の項を参照してください。 ● キャッシュの変更 JBoss EAP 7 で適用された Infinispan キャッシングストラテジーと設定の変更は、本ガイドの以下の項を参照してください。 ● Infinispan サーバー設定の変更
データソースおよびリソースアダプター	JBoss EAP 6 ではデータソースとリソースアダプターの設定が主に1つのファイルに集約されましたが、これは JBoss EAP 7 でも同様です。詳細は、JBoss EAP 6『**移行ガイド**』の以下の項を参照してください。 ● データソースおよびリソースアダプター設定の変更
ディレクトリー構造、スクリプト、およびデプロイメント設定	JBoss EAP 6 では、ディレクトリー構造、スクリプト、およびデプロイメント設定が変更になりました。これらの変更は JBoss EAP 7 にも該当します。詳細は、JBoss EAP 6『**移行ガイド**』の以下の項を参照してください。 ● JBoss EAP 6 の新機能と変更内容

JBoss EAP 5の機能またはコンポーネント	変更の概要および移行情報の場所
EJB	Java EE 7 仕様によって、EJB 2.x およびそれ以前の機能が任意となったため、アプリケーションを書き直して EJB 3.x 仕様と JPA を使用するようにすることが強く推奨されます。非推奨となった機能と EJB 2.x の実行に必要な変更については、JBoss EAP 6 『**移行ガイド**』の以下の項を参照してください。 ● 「EJB 2.x and Earlier Changes」 JBoss EAP 6 では、ステートフル EJB キャッシュおよびステートレスセッション bean プールサイズはサーバー設定ファイルの **ejb3** サブシステムに設定されています。**jboss.xml** デプロイメント記述子ファイルは **jboss-ejb3.xml** デプロイメント記述子ファイルに置き換えられます。変更についての詳細は、JBoss EAP 6 『**移行ガイド**』の以下の項を参照してください。 ● 「EJB Changes」 JBoss EAP 7 ではデフォルトのリモートコネクターおよびポートが変更になりました。この詳細とサーバー設定の変更については、本ガイドの以下の項を参照してください。 ● EJB サーバー設定の変更 ● EJB クライアントコードの移行 JBoss EAP 7 では、EJB エンティティー bean がサポートされません。エンティティー bean を JPA に移行する方法は、本ガイドの以下の項を参照してください。 ● エンティティー Bean の JPA への移行

JBoss EAP 5の機能またはコンポーネント	変更の概要および移行情報の場所
Hibernate および JPA	JBoss EAP 6 では、Hibernate がバージョン 3 から 4 に更新されました。この JBoss EAP バージョンは JPA 2.0 仕様も実装し、JPA 永続プロパティーに変更が加えられました。これらの変更に合わせてアプリケーションを編集する方法については、JBoss EAP 6『**移行ガイド**』を参照してください。 ● Hibernate および JPA の変更 JBoss EAP 7.2 は JPA 2.2 を実装し、Hibernate 5.3 が含まれます。また、Hibernate Search のバージョン 5.10 も含まれます。その他の変更には、EJB エンティティーbean のサポートの削除や JPA 永続プロパティーの追加更新が含まれます。これらの変更によるアプリケーションへの影響に関する情報は、本ガイドの以下の項を参照してください。 ● Hibernate および JPA の移行の変更 ● Hibernate Search の変更 ● エンティティー Bean の JPA への移行 ● JPA 永続プロパティーの変更  **注記** JBoss EAP に同梱されるものとは異なるバージョンの Hibernate を使用することはできません。JBoss EAP に同梱されるバージョンは、テストされた Hibernate の唯一のバージョンであり、不具合への対処としてパッチが提供される唯一のバージョンとなります。
JAX-RS および RESTEasy	JBoss EAP 6 は、自動的に RESTEasy とアプリケーション設定の必要な変更を設定した RESTEasy 2 をバンドルしました。詳細は、JBoss EAP 6『移行ガイド』の以下の項を参照してください。 ● JAX-RS および RESTEasy の変更 JBoss EAP 7 には RESTEasy 3 が含まれ、多くのクラスが非推奨になりました。Jackson のバージョンが 1.9.9 から 2.6.3 以上に変更されました。これらの変更の詳細は、本ガイドの以下の項を参照してください。 ● JAX-RS および RESTEasy アプリケーションの変更
JBoss AOP	JBoss AOP (アスペクト指向プログラミング) は JBoss EAP 6 では削除されました。JBoss AOP を使用するアプリケーションをリファクタリングする方法については、JBoss EAP 6『移行ガイド』の以下の項を参照してください。 ● JBoss AOP 変更

JBoss EAP 5の機能またはコンポーネント	変更の概要および移行情報の場所
JGroups およびクラスタリング	JBoss EAP 6 では、クラスタリングを有効にし、バインドアドレスを指定する方法が変更になりました。詳細は、JBoss EAP 6『**移行ガイド**』の以下の項を参照してください。 ● クラスタリングの変更 JBoss EAP 7 では、JGroups はデフォルトでパブリックではなくプライベートネットワークインターフェースを使用するようになり、さらに **<channel>** 要素が **jgroups** サブシステムに導入されました。JBoss EAP 7 には Undertow mod_cluster 実装が含まれ、シングルトンサービスの構築に新しい API が導入され、その他にも新しいクラスタリング機能が導入されました。これらの変更は本ガイドの以下の項に記載されています。 ● JGroups サーバー設定の変更 ● アプリケーションクラスタリングの変更
JNDI	JBoss EAP 6 は新しい標準化されたグローバル JNDI ネームスペースを実装し、Java EE アプリケーションのさまざまなスコープへマップする関連するネームスペースを実装していました。新しい JNDI ネームスペースルールを使用するために必要なアプリケーションの変更については、JBoss EAP 6『**移行ガイド**』の以下の項を参照してください。 ● JNDI の変更
JSF	JBoss EAP 6.4 には JSF 1.2 と JSF 2.1 の両方が含まれ、アプリケーションを設定して旧バージョンを使用することができました。JBoss EAP 7.2 には JSF 2.3 が含まれ、旧バージョンを使用することができなくなりました。詳細は、本ガイドの以下の項を参照してください。 ● JavaServer Faces (JSF) のコード変更

JBoss EAP 5の機能またはコンポーネント	変更の概要および移行情報の場所
Logging	JBoss EAP 6 には、新しい JBoss Logging フレームワークが導入され、これは JBoss EAP 7 でも使用されています。サードパーティーロギングフレームワークを使用するアプリケーションは、モジュラークラスローディングの変更による影響を受ける場合があります。変更の詳細については、JBoss EAP 6『**移行ガイド**』の以下の項を参照してください。 ● ロギングの変更 JBoss EAP 7 では、**org.jboss.logging** パッケージのアノテーションが非推奨となったため、ソースコードおよび Maven GAV (groupId:artifactId:version) に影響します。すべてのログメッセージの接頭辞も変更になりました。変更の詳細は、本ガイドの以下の項を参照してください。 ● JBoss Logging の変更 ● ロギングメッセージの接頭辞の変更
メッセージングおよび JMS	JBoss EAP 6 では、デフォルトの JMS 実装として JBoss Messaging が HornetQ に置き換えられました。JBoss EAP 7 では、ビルトインメッセージングプロバイダーとして HornetQ が ActiveMQ Artemis に置き換えられました。 メッセージング設定を移行する場合、JBoss EAP 7 のデフォルトサーバー設定から始め、以下のガイドを使用して現在のメッセージング設定の変更を適用することが推奨されます。 ● JBoss EAP 7.2 の『**Configuring Messaging**』 JBoss Messaging から HornetQ に移行するために必要な変更を理解するには、JBoss EAP 6『**移行ガイド**』の以下の項を参照してください。 ● HornetQ の変更 その後、HornetQ 設定と関連するメッセージングデータの移行方法に関し、本ガイドの以下の情報を確認します。 ● メッセージングサーバー設定の変更 ● メッセージングアプリケーションの変更
ORB	JBoss EAP 6 では、JacORB 設定は **EAP_HOME/server/production/conf/jacorb.properties** ファイルからサーバー設定ファイルに移されました。JBoss EAP 7 では、JacORB IIOP 実装は OpenJDK ORB のダウンストリームブランチに置き換えられました。 ORB 設定を移行する場合、JBoss EAP 7 のデフォルトサーバー設定から始め、JBoss EAP 7.2 **設定ガイド** の以下の項を参照して現在の ORB 設定の変更を適用することが推奨されます。 ● ORB 設定

JBoss EAP 5の機能またはコンポーネント	変更の概要および移行情報の場所
リモート呼び出し	JBoss EAP 6 では、リモート呼び出しのために新しい EJB クライアント API が導入されましたが、新しい API を使用するためにアプリケーションコードを書き直したくない場合は、既存のコードを編集して EJB へのリモートアクセスに **ejb:BEAN_REFERENCE** を使用することができました。詳細は、JBoss EAP 6『**移行ガイド**』の以下の項を参照してください。 ● リモート呼び出しの変更 JBoss EAP 7 では、デフォルトのコネクターとデフォルトのリモート接続ポートが変更になりました。詳細は、本ガイドの以下の項を参照してください。 ● リモート URL コネクターおよびポートの更新 ● 外部クライアントの更新 ● EJB クライアントコードの移行
Seam 2.x	JBoss EAP 6 では、Seam 2.2 アプリケーションの正式サポートがなくなりましたが、JSF 1.2 と Hibernate 3 の依存関係を設定して Seam 2.2 アプリケーションを実行することが可能でした。JSF 2.3 と Hibernate 5.3.1 が含まれる JBoss EAP 7.2 では、Red Hat JBoss Web Framework Kit のライフサイクル終了に伴い Seam 2.2 または Seam 2.3 はサポートされません。Weld CDI bean を使用して Seam コンポーネントを書き直すことが推奨されます。
Security	JBoss EAP 6 のセキュリティー更新には、セキュリティードメイン名の変更および Basic 認証のセキュリティー設定方法の変更が含まれていました。LDAP セキュリティーレルム設定はサーバー設定ファイルに移されました。詳細は、JBoss EAP 6『**移行ガイド**』の以下の項を参照してください。 ● セキュリティーの変更 ● LDAP セキュリティーレルムの変更 JBoss EAP 7 のセキュリティーに影響する更新には、サーバー設定の変更やアプリケーションの変更が含まれます。詳細は、本ガイドの以下の項を参照してください。 ● セキュリティーサーバー設定の変更 ● セキュリティーアプリケーションの変更
Spring アプリケーション	Spring 4.2.x は、JBoss EAP 7 によってサポートされる最も初期の安定したバージョンです。Apache CXF Spring web サービスおよび Spring と RESTEasy の統合の変更に関する詳細は、本ガイドの以下の項を参照してください。 ● Apache CXF Spring Web サービスの変更 ● Spring と RESTEasy の統合の変更

JBoss EAP 5の機能またはコンポーネント	変更の概要および移行情報の場所
Transactions	JBoss EAP 6 では、トランザクション設定が集約され、サーバー設定ファイルに移されました。その他の更新には、JTA ノード識別子の設定や JTS の有効方法の変更が含まれます。詳細は、JBoss EAP 6『**移行ガイド**』の以下の項を参照してください。 ● JTS および JTA の変更 JBoss EAP 6 の **transactions** サブシステムで使用できた Transaction Manager 設定属性の一部が JBoss EAP 7 で変更になりました。詳細は、本ガイドの以下の項を参照してください。 ● transactions サブシステムの変更
バルブ	JBoss EAP 7 では JBoss Web が Undertow に置き換えられ、バルブはサポート対象外となりました。本ガイドの以下の項を参照してください。 ● グローバルバルブの移行 ● カスタムアプリケーションバルブの移行 ● オーセンティケーターバルブの移行
Web サービス	JBoss EAP 6 には JBossWS 4 が含まれていました。バージョン更新に必要な変更に関する詳細は、JBoss EAP 6『**移行ガイド**』の以下の項を参照してください。 ● Web サービスの変更 JBoss EAP 7 には JBossWS 5 が導入されました。必要な更新については、本ガイドの以下の項を参照してください。 ● Web サービスアプリケーションの変更

付録A リファレンス資料

A.1. JACORB サブシステム移行操作の警告

migrate 操作はすべてのリソースや属性を処理することはできません。**jacorb** サブシステムの **migrate** または **describe-migration** 操作を実行すると表示される可能性がある警告の一部を以下の表に示します。

注記

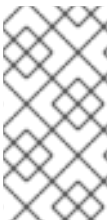
migrate 操作の出力に「Could not migrate」または「Can not migrate」エントリーが記録された場合、サーバー設定の移行は正常に完了したにも関わらず、すべての要素および属性を自動的に移行できなかったことを表します。「migration-warnings」の提案に従ってこれらの設定を変更する必要があります。

警告メッセージ	メッセージの意味 / 修正方法
The **iiop** migration can be performed when the server is in **admin-only** mode (iiop の移行はサーバーが admin-only モードであるときに実行できます)	**migrate** 操作を実行するには、**--start-mode=admin-only** をサーバー起動コマンドに追加し、サーバーを **admin-only** モードで起動する必要があります。 $ **EAP_HOME**/bin/standalone.sh --start-mode=admin-only
Properties **X** cannot be emulated using OpenJDK ORB and are not supported (プロパティー X は OpenJDK ORB を使用してエミュレートできず、サポートされていません)	指定プロパティーの設定はサポートされず、新しい **iiop-openjdk** サブシステム設定には含まれていません。以前のリリースの JBoss EAP でこのプロパティーによって示された動作は移行されないため、管理者は JBoss EAP 7 の新しい **iiop-openjdk** サブシステムがこの動作を示さずに正しく操作できることを検証する必要があります。 サポートされないプロパティーには以下が含まれます: **cache-poa-names**、 **cache-typecodes**、 **chunk-custom-rmi-valuetypes**、 **client-timeout**、 **comet**、 **indirection-encoding-disable**、 **iona**、 **lax-boolean-encoding**、 **max-managed-buf-size**、 **max-server-connections**、 **max-threads**、 **outbuf-cache-timeout**、 **outbuf-size**、 **queue-max**、 **queue-min**、 **poa-monitoring**、 **print-version**、 **retries**、 **retry-interval**、 **queue-wait**、 **server-timeout**、 **strict-check-on-tc-creation**、 **use-bom**、 **use-imr**。

警告メッセージ	メッセージの意味 / 修正方法
The properties **X** use expressions.(プロパティー X は式を使用します。Configuration properties that are used to resolve those expressions should be transformed manually to the new **iiop-openjdk** subsystem format. (これらの式を解決するために使用される設定プロパティーは新しい iiop-openjdk サブシステム形式に手動で変換する必要があります)	式を使用するプロパティーは管理者が手作業で設定する必要があります。 たとえば、JBoss EAP 6 の **jacorb** サブシステムは **giop-minor-version** を定義しました。JBoss EAP 7 の **iiop-openjdk** サブシステムは **giop-version** プロパティーを定義します。**jacorb** サブシステムのマイナーバージョン属性が **${iiop-giop-minor-version}** に設定され、システムプロパティーが **-Diiop-giop-minor-version=1** として **standalone.conf** ファイルに設定されているとします。**migrate** 操作の後に、新しいサブシステムが適切に設定されるように、adminstrator がシステムプロパティー値を **1.1** に変更する必要があります。
Can not migrate: the new **iiop-openjdk** subsystem is already defined (移行できません: 新しい iiop-openjdk サブシステムはすでに定義されています)	メッセージには説明が含まれています。

A.2. MESSAGING サブシステム移行操作の警告

migrate 操作はすべてのリソースや属性を処理することはできません。**messaging** サブシステムの **migrate** または **describe-migration** 操作を実行すると表示される可能性がある警告の一部を以下の表に示します。



注記

migrate 操作の出力に「Could not migrate」または「Can not migrate」エントリーが記録された場合、サーバー設定の移行は正常に完了したにも関わらず、すべての要素および属性を自動的に移行できなかったことを表します。「migration-warnings」の提案に従ってこれらの設定を変更する必要があります。

警告メッセージ	メッセージの意味 / 修正方法
The **migrate** operation can not be performed: the server must be in **admin-only** mode (migrate 操作は実行できません。サーバーは admin-only モードである必要があります)	**migrate** 操作を実行するには、**--start-mode=admin-only** をサーバー起動コマンドに追加し、サーバーを **admin-only** モードで起動する必要があります。 `$ EAP_HOME/bin/standalone.sh --start-mode=admin-only`
Can not migrate attribute **local-bind-address** from resource **X**. (リソース X から属性 local-bind-address を移行できません。Use instead the **socket-binding** attribute to configure this **broadcast-group**.(代わりに socket-binding 属性を使用してこの broadcast-group を設定してください。)	メッセージには説明が含まれています。

警告メッセージ	メッセージの意味 / 修正方法
Can not migrate attribute **local-bind-port** from resource **X**.(リソース X から属性 local-bind-port を移行できません。Use instead the **socket-binding** attribute to configure this **broadcast-group**.(代わりに socket-binding 属性を使用してこの broadcast-group を設定してください。)	メッセージには説明が含まれています。
Can not migrate attribute **group-address** from resource **X**.(リソース X から属性 group-address を移行できません。Use instead the **socket-binding** attribute to configure this **broadcast-group**.(代わりに socket-binding 属性を使用してこの broadcast-group を設定してください。)	メッセージには説明が含まれています。
Can not migrate attribute **group-port** from resource **X**.(リソース X から属性 group-port を移行できません。Use instead the **socket-binding** attribute to configure this **broadcast-group**.(代わりに socket-binding 属性を使用してこの broadcast-group を設定してください。)	**broadcast-group** リソースは **local-bind-address**、**local-bind-port**、**group-address**、または **group-port** 属性を許可しないようになりました。このリソースは **socket-binding** 属性のみを許可します。この警告は、リソース **X** にサポートされない属性があることを通知します。リソースの **socket-binding** 属性を手作業で設定し、定義された **socket-binding** リソースに対応するようにする必要があります。
Classes providing the **X** are discarded during the migration.（移行中に X を提供するクラスは破棄されます)To use them in the new **messaging-activemq** subsystem, you will have to extend the Artemis-based **Interceptor**.(新しい messaging-activemq サブシステムでこれらを使用するには、Artemis ベースのインターセプターを拡張する必要があります)	JBoss EAP 7 のメッセージングインターセプターのサポートは大幅に異なります。以前のバージョンのサブシステムに設定されたインターセプターはすべて移行中に破棄されます。詳細は メッセージングインターセプターの移行を参照してください。
Can not migrate the HA configuration of **X**.(X の HA 設定は移行できません)Its **shared-store** and **backup** attributes holds expressions and it is not possible to determine unambiguously how to create the corresponding **ha-policy** for the messaging-activemq's server.(この shared-store および backup 属性は式を保持し、messaging-activemq のサーバーの対応する ha-policy を作成する方法を明確に決定することができません)	This means the **hornetq-server** X's **shared-store** or **backup** attributes contained an expression, such as ${xxx}, and the migration operation was not able to resolve it to a concrete expression.(これは、hornetq-server X の shared-store または backup 属性に ${xxx} などの式が含まれ、移行操作が具体的な式に解決できなかったことを意味します)値は破棄され、**messaging-activemq** の **ha-policy** を手作業で更新する必要があります。
Can not migrate attribute **local-bind-address** from resource **X**. (リソース X から属性 local-bind-address を移行できません。Use instead the **socket-binding** attribute to configure this **discovery-group**.(代わりに socket-binding 属性を使用してこの discovery-group を設定してください)	メッセージには説明が含まれています。

警告メッセージ	メッセージの意味 / 修正方法
Can not migrate attribute **local-bind-port** from resource **X**.(リソース X から属性 local-bind-port を移行できません。Use instead the **socket-binding** attribute to configure this **discovery-group**.(代わりに socket-binding 属性を使用してこの discovery-group を設定してください)	メッセージには説明が含まれています。
Can not migrate attribute **group-address** from resource **X**.(リソース X から属性 group-address を移行できません。Use instead the **socket-binding** attribute to configure this **discovery-group**.(代わりに socket-binding 属性を使用してこの discovery-group を設定してください)	メッセージには説明が含まれています。
Can not migrate attribute **group-port** from resource **X**.(リソース X から属性 group-port を移行できません。Use instead the **socket-binding** attribute to configure this **discovery-group**.(代わりに socket-binding 属性を使用してこの discovery-group を設定してください)	**discovery-group** リソースは **local-bind-address**、**local-bind-port**、**group-address**、または **group-port** 属性を許可しないようになりました。**socket-binding** のみを許可します。この警告は、リソース **X** にサポートされない属性があることを通知します。リソースの **socket-binding** 属性を手作業で設定し、定義された **socket-binding** リソースに対応するようにする必要があります。
Can not create a **legacy-connection-factory** based on **connection-factory X**.(connection-factory X を基にして legacy-connection-factory を作成できません)It uses a HornetQ **in-vm** connector that is not compatible with Artemis **in-vm** connector (これは Artemis in-vm コネクターと互換性のない HornetQ in-vm コネクターを使用します)	JBoss EAP 6 クライアントが JBoss EAP 7 に接続できるようにするため、レガシーの HornetQ リモート **connection-factory** リソースは **legacy-connection-factory** リソースに移行されました。しかし、**legacy-connection-factory** リソースは **connection-factory** がリモートコネクターを使用しているときのみ作成されます。**in-vm** クライアントは JBoss EAP 6 ではなく JBoss EAP 7 をベースにしているため、**in-vm** を使用する **connection-factory** は移行されません。この警告は、**in-vm connection-factory** が移行されなかったことを通知します。

警告メッセージ	メッセージの意味 / 修正方法
Can not migrate attribute **X** from resource **Y**.(リソース Y から属性 X を移行できません。属性はシステムプロパティーに応じて異なって解決される式を使用します。After migration, this attribute must be added back with an actual value instead of the expression.(移行後、式の代わりに実際の値を用いてこの属性を戻す必要があります)	この警告は、移行処理中に属性 **X** を具体的な値に解決できないときに表示されます。値は破棄され、属性を手作業で移行する必要があります。これは以下の場合に発生します。 ● **cluster-connection forward-when-no-consumers**: このブール値属性は、列挙で **OFF**、**STRICT**、または **ON_DEMAND** を値として持つ **message-load-balancing-type** 属性に置き換えられました。 ● **broadcast-group** および **discovery-group** の **jgroups-stack** および **jgroups-channel** 属性 これらの属性は他のリソースを参照し、JBoss EAP 7 ではこれらの式が許可されないようになりました。
Can not migrate attribute **X** from resource **Y**.(リソース Y から属性 X を移行できません。This attribute is not supported by the new **messaging-activemq** subsystem.(この属性は新しい messaging-activemq サブシステムによってサポートされません)	一部の属性は新しい **messaging-activemq** サブシステムではサポート対象外となり、破棄されます。 ● **hornetq-server** の **failback-delay** ● **http-connector** の **use-nio** 属性 ● **http-acceptor** の **use-nio** 属性 ● **remote-connector** の **use-nio** 属性 ● **remote-acceptor** の **use-nio** 属性
Can not migrate attribute **failback-delay** from resource **X**.(リソース X から 属性 failback-delay を移行できません)Artemis detects failback deterministically and it no longer requires to specify a delay for failback to occur.(フェイルバックは Artemis によって確定的に検出され、フェイルバックの遅延を指定する必要がなくなりました)	メッセージには説明が含まれています。

非推奨の broadcast-group または discovery-group 属性の置き換え

非推奨の **broadcast-group** または **discovery-group** 属性を **socket-binding** 属性に置き換えるよう通知された場合は、管理 CLI を使用して、この新しい属性を追加できます。

以下の例では、**messaging** サブシステムに以下の **discovery-group** 設定が含まれるスタンドアロンサーバーを移行することを前提とします。

```
<discovery-groups>
   <discovery-group name="my-discovery-group">
      <group-address>224.0.1.105</group-address>
      <group-port>56789</group-port>
   </discovery-group>
</discovery-groups>
```

messaging サブシステムに対して **migrate** 操作を実行すると、以下の出力および警告が表示されます。

```
/subsystem=messaging:migrate
{
    "outcome" => "success",
    "result" => {"migration-warnings" => [
        "WFLYMSG0084: Can not migrate attribute group-address from resource [
    (\"subsystem\" => \"messaging-activemq\"),
    (\"server\" => \"default\"),
    (\"discovery-group\" => \"my-discovery-group\")
]. Use instead the socket-binding attribute to configure this discovery-group.",
        "WFLYMSG0084: Can not migrate attribute group-port from resource [
    (\"subsystem\" => \"messaging-activemq\"),
    (\"server\" => \"default\"),
    (\"discovery-group\" => \"my-discovery-group\")
]. Use instead the socket-binding attribute to configure this discovery-group."
    ]}
}
```

migrate 操作によって、"my-discovery-group" という名前の **discovery-group** が新しい **messaging-activemq** サブシステムに作成され、以下のように設定されます。

```
<discovery-group name="my-discovery-group"/>
```

ここで、以下の管理 CLI コマンドを使用して、"my-discovery-group-socket-binding" という名前のサーバー設定ファイルに **socket-binding** 要素を作成する必要があります。

```
/socket-binding-group=standard-sockets/socket-binding=my-discovery-group-socket-binding:add(multicast-address=224.0.1.105, multicast-port=56789)
```

次に、以下の管理 CLI コマンドを使用して、新たに作成された **socket-binding** をサーバー設定ファイルにある **messaging-activemq** サブシステムの "my-discovery-group" という名前の **discovery-group** に追加します。

```
/subsystem=messaging-activemq/server=default/discovery-group=my-discovery-group:write-attribute(name=socket-binding,value=my-discovery-group-socket-binding)
```

これらのコマンドによって、サーバー設定ファイルに以下の XML が作成されます。

```
<subsystem xmlns="urn:jboss:domain:messaging-activemq:4.0">
    <server name="default">
        ...
        <discovery-group name="my-discovery-group" socket-binding="my-discovery-group-socket-binding"/>
        ...
    </server>
</subsystem>
...
<socket-binding-group name="standard-sockets" default-interface="public" port-offset="${jboss.socket.binding.port-offset:0}">
    ...
    <socket-binding name="my-discovery-group-socket-binding" multicast-address="224.0.1.105"
```

```
multicast-port="56789"/>
    ...
</socket-binding-group>
```

A.3. WEB サブシステム移行操作の警告

migrate 操作はすべてのリソースや属性を処理することはできません。**web** サブシステムの **migrate** または **describe-migration** 操作を実行すると表示される可能性がある警告の一部を以下の表に示します。

注記

migrate 操作の出力に「Could not migrate」または「Can not migrate」エントリーが記録された場合、サーバー設定の移行は正常に完了したにも関わらず、すべての要素および属性を自動的に移行できなかったことを表します。「migration-warnings」の提案に従ってこれらの設定を変更する必要があります。

警告メッセージ	メッセージの意味 / 修正方法
Migrate operation only allowed in admin only mode (移行操作は管理専用モードのみで許可されます)	**migrate** 操作を実行するには、**--admin-only** パラメーターをサーバー起動コマンドに追加し、サーバーを **admin-only** モードで起動する必要があります。 `$ EAP_HOME/bin/standalone.sh --admin-only`
Could not migrate resource **X** (リソース X を移行できませんでした)	以前のリリースの JBoss EAP でこのリソースによって示された動作は移行されませんでした。管理者は JBoss EAP 7 の新しい **undertow** サブシステムがこの動作を示さずに正しく操作できるかどうか、またはこの動作を手作業で移行する必要があるかどうかを検証する必要があります。
Could not migrate attribute **X** from resource **Y**.(リソース Y から属性 X を移行できません)	以前のリリースの JBoss EAP でこのリソース属性によって示された動作は移行されませんでした。管理者は JBoss EAP 7 の新しい **undertow** サブシステムがこの動作を示さずに正しく操作できるかどうか、またはこの動作を手作業で移行する必要があるかどうかを検証する必要があります。 移行されない属性の一覧は、web サブシステム移行操作属性の警告 を参照してください。
Could not migrate SSL connector as no SSL config is defined (定義された SSL 設定がないため SSL コネクターを移行できませんでした)	メッセージには説明が含まれています。
Could not migrate **verify-client** attribute **X** to the Undertow equivalent (verify-client 属性 X を Undertow の同等の属性に移行できませんでした)	メッセージには説明が含まれています。

警告メッセージ	メッセージの意味 / 修正方法
Could not migrate **verify-client** expression **X** (verify-client 式 X を移行できませんでした)	メッセージには説明が含まれています。
Could not migrate valve **X** (バルブ X を移行できませんでした)	以前のリリースの JBoss EAP でこのバルブによって示された動作は移行されませんでした。管理者は JBoss EAP 7 の新しい **undertow** サブシステムがこの動作を示さずに正しく操作できるかどうか、またはこの動作を手作業で移行する必要があるかどうかを検証する必要があります。 この警告は以下のバルブに対して発生する可能性があります。 ● **org.apache.catalina.valves.RemoteAddrValve** 許可または拒否される値が 1 つ以上必要です。 ● **org.apache.catalina.valves.RemoteHostValve** 許可または拒否される値が 1 つ以上必要です。 ● **org.apache.catalina.authenticator.BasicAuthenticator** ● **org.apache.catalina.authenticator.DigestAuthenticator** ● **org.apache.catalina.authenticator.FormAuthenticator** ● **org.apache.catalina.authenticator.SSLAuthenticator** ● **org.apache.catalina.authenticator.SpnegoAuthenticator** ● カスタムバルブ

警告メッセージ	メッセージの意味 / 修正方法
Could not migrate attribute **X** from valve **Y** (バルブ Y から属性 X を移行できませんでした)	以前のリリースの JBoss EAP でこのバルブ属性によって示された動作は移行されませんでした。管理者は JBoss EAP 7 の新しい **undertow** サブシステムがこの動作を示さずに正しく操作できるかどうか、またはこの動作を手作業で移行する必要があるかどうかを検証する必要があります。この警告は、以下のバルブ属性に対して発生する可能性があります。 ● **org.apache.catalina.valves.AccessLogValve** ○ **resolveHosts** ○ **fileDateFormat** ○ **renameOnRotate** ○ **encoding** ○ **locale** ○ **requestAttributesEnabled** ○ **buffered** ● **org.apache.catalina.valves.ExtendedAccessLogValve** ○ **resolveHosts** ○ **fileDateFormat** ○ **renameOnRotate** ○ **encoding** ○ **locale** ○ **requestAttributesEnabled** ○ **buffered** ● **org.apache.catalina.valves.RemoteIpValve** ○ **httpServerPort** ○ **httpsServerPort** ○ **remoteIpHeader** 定義されていても "x-forwarded-for" に設定されていない場合 ○ **protocolHeader** 定義されていても "x-forwarded-proto" に設定されていない場合

web サブシステム移行操作属性の警告

migrate 操作はすべてのリソースや属性を処理することはできません。処理されなかった属性を手作業で移行する方法については、以下の表を参照してください。

Web SSL コネクター属性
以下の属性は、SSL コネクターを設定するために JBoss EAP 6 で使用されました。OpenSSL ネイティブライブラリーは JBoss EAP 7 ではサポートされないため、同等の設定はありません。

属性	説明	Undertow での同等の属性
ca-revocation-url	呼び出しリストが含まれるファイルまたは URL。	Undertow では同等の属性がありません。
certificate-file	OpenSSL の暗号化を使用する場合、サーバー証明書が含まれるファイルへのパスになります。	Undertow では同等の属性がありません。
ssl-protocol	SSL プロトコルの文字列。	Undertow では同等の属性がありません。
verify-depth	クライアントが有効な証明を持たないと判断するまでにチェックされる中間証明書発行者の最大数。	Undertow では同等の属性がありません。

web 静的リソース属性
以下の **static-resources** 要素は、静的リソースが **DefaultServlet** または **WebdavServlet** によってどのように処理されるかを記述するために使用されました。WebDAV は Undertow によってサポートされないため、これらの属性と同等のものはありません。詳細は、https://issues.jboss.org/browse/JBEAP-1036 を参照してください。

属性	説明	Undertow での同等の属性
disabled	デフォルトのサーブレットマッピングを有効にします。	Undertow には同等の設定がありません。
file-encoding	静的ファイルの読み取り時に使用されるファイルエンコーディング。	Undertow には同等の設定がありません。
max-depth	**PROPFIND** の最大再帰数。	これは WebDAV の設定で、WebDAV は Undertow によってサポートされません。
read-only	HTTP メソッドの記述を許可します (PUT、DELETE)。	これは WebDAV の設定で、WebDAV は Undertow によってサポートされません。
secret	WebDAV ロッキング操作のシークレット。	これは WebDAV の設定で、WebDAV は Undertow によってサポートされません。

属性	説明	Undertow での同等の属性
sendfile	指定のバイトサイズよりも大きいファイルに対し、可能であればsendfile を有効にします。	Undertow では妥当なデフォルト値に設定され、設定不可能です。
webdav	WebDAV の機能を有効にします。	WebDAV は Undertow によってサポートされません。

web SSO リソース属性
SSO の処理はこれまでのリリースとは異なり、JBoss EAP 7 には同等の属性設定がありません。

JBoss Web 属性	説明	Undertow での同等の属性
cache-container	クラスター化された SSO に使用するキャッシュコンテナーの名前。	この設定は Undertow では必要ありません。これは、分散されたクラスター化環境全体でデフォルトで動作します。
cache-name	クラスター化された SSO で使用するキャッシュの名前。	この設定は Undertow では必要ありません。これは、分散されたクラスター化環境全体でデフォルトで動作します。
reauthenticate	各リクエストによって再認証が行われるかどうか。	挙動が JBoss EAP 6 の **reauthenticate=true** 設定と似ている設定は Undertow ではありません。**reauthenticate=false** はパフォーマンスを向上できる可能性がありましたが、セキュリティーの問題が発生する可能性もありました。

web アクセスログ属性

JBoss Web 属性	説明	Undertow での同等の属性
resolve-hosts	アクセスログのホストの解決を有効にするかどうか。	コネクターの設定を使用して同じ挙動を実現します。

web コネクター属性

JBoss Web 属性	説明	Undertow での同等の属性
executor	このコネクターのスレッドを処理するために使用されるべきエクゼキューターの名前。	**io** サブシステムに定義されたワーカーを参照するようになります。 詳細は、threads サブシステム設定の移行を参照してください。

JBoss Web 属性	説明	Undertow での同等の属性
proxy-binding	リダイレクトの送信時に使用されるホストおよびポートを定義するソケットバインディング。	直接的に同等な設定はありません。 使用可能な設定オプションについては、JBoss EAP 設定ガイド の **https-listener 属性** を参照してください。
redirect-port	セキュアなコネクターへリダイレクトするためのポート。	この属性は JBoss EAP 6 で非推奨となり、**redirect-binding** に置き換えられました。Undertow は、**http-listener** 要素上で **redirect-socket** 属性を提供し、これは **redirect-binding** の代わりとなります。 詳細は、JBoss EAP 設定ガイド の **https-listener 属性** を参照してください。

A.4. JBOSS WEB システムプロパティーのリファレンス

このリファレンスでは、JBoss Web 設定で以前使用されたシステムプロパティーを JBoss EAP 7 の Undertow で同等の設定にマップする方法を説明します。

- サーブレットコンテナーおよびコネクターシステムプロパティーのマップ
- EL システムプロパティーのマップ
- JSP システムプロパティー
- セキュリティーシステムプロパティーのマップ

表A.1 サーブレットコンテナーおよびコネクターシステムプロパティーのマップ

JBoss EAP 6 システムプロパティー	説明 JBoss EAP 7 での同等設定
jvmRoute	**jvmRoute** 属性のデフォルトの値を提供します。**standalone-ha.xml** 設定ファイルを使用する場合は自動生成された値をオーバーライドしません。 **reload** をサポートします。 管理 CLI コマンド: `/subsystem=undertow:write-attribute(name=instance-id,value=VALUE)`

org.apache.tomcat.util.buf.StringCache.byte.enabled	**true** を指定した場合、String キャッシュは **ByteChunk** に対して有効になります。値の指定がない場合は、デフォルト値の **false** が使用されます。
	同等の設定はありません。
org.apache.tomcat.util.buf.StringCache.char.enabled	**true** を指定した場合、String キャッシュは **CharChunk** に対して有効になります。値の指定がない場合は、デフォルト値の **false** が使用されます。
	同等の設定はありません。
org.apache.tomcat.util.buf.StringCache.cacheSize	String キャッシュのサイズ。値の指定がない場合は、デフォルト値の **5000** が使用されます。
	同等の設定はありません。
org.apache.tomcat.util.buf.StringCache.maxStringSize	キャッシュされる String の最大長。値の指定がない場合は、デフォルト値の **128** が使用されます。
	同等の設定はありません。
org.apache.tomcat.util.http.FastHttpDateFormat.CACHE_SIZE	解析およびフォーマットされた日付値を使用するキャッシュのサイズ。値の指定がない場合は、デフォルト値の **1000** が使用されます。
	同等の設定はありません。
org.apache.catalina.core.StandardService.DELAY_CONNECTOR_STARTUP	**true** を使用した場合、コネクターは自動的に起動されません。これは埋め込みモードで便利です。
	同等の設定はありません。
org.apache.catalina.connector.Request.SESSION_ID_CHECK	**true** に指定された場合、指定のセッション ID でセッションを作成する前に、サーブレットコンテナーはそのセッション ID のコンテキストにセッションが存在することを確認します。
	同等の設定はありません。
org.apache.coyote.USE_CUSTOM_STATUS_MSG_IN_HEADER	**true** を指定した場合は、HTTP ヘッダー内でカスタムの HTTP ステータスメッセージが使用されます。XSS の脆弱性から保護するため、特にメッセージにユーザーによる入力が含まれる場合は、このようなメッセージをすべて **ISO-8859-1** でエンコードする必要があります。指定の値がない場合は、デフォルト値の **false** が使用されます。

	カスタムの **io.undertow.servlet.ServletExtension** を実装して、プログラミングで有効にする必要があります。その後、拡張を使用して **io.undertow.servlet.api.DeploymentInfo** 構造インスタンス上で **setSendCustomReasonPhraseOnError(true)** を呼び出します。
org.apache.tomcat.util.http.Parameters.MAX_COUNT	ポストのボディーで解析できるパラメーターの最大数。パラメーターがこの数を超えると、**IllegalStateException** によって解析に失敗します。デフォルト値は **512** パラメーターです。

	管理 CLI コマンド:
	`/subsystem=undertow/server=default-server/http-listener=default:write-attribute(name=max-parameters,value=VALUE)`
	`/subsystem=undertow/server=default-server/https-listener=default:write-attribute(name=max-parameters,value=VALUE)`
	`/subsystem=undertow/server=default-server/ajp-listener=default:write-attribute(name=max-parameters,value=VALUE)`

org.apache.tomcat.util.http.MimeHeaders.MAX_COUNT	HTTP リクエストで送信できるヘッダーの最大数。ヘッダーの数がこの値を超えると、**IllegalStateException** によって解析に失敗します。デフォルト値は **128** ヘッダーです。

	管理 CLI コマンド:
	`/subsystem=undertow/server=default-server/http-listener=default:write-attribute(name=max-headers,value=VALUE)`
	`/subsystem=undertow/server=default-server/https-listener=default:write-attribute(name=max-headers,value=VALUE)`
	`/subsystem=undertow/server=default-server/ajp-listener=default:write-attribute(name=max-headers,value=VALUE)`

org.apache.tomcat.util.net.MAX_THREADS	コネクターがリクエストの処理に使用するスレッドの最大数。デフォルト値は **32** x **512** です。(**JIO** コネクター の場合は **512** x **Runtime.getRuntime**（）**.availableProcessors**（））
	管理 CLI コマンド: `/subsystem=io/worker=default:write-attribute(name=task-max-threads, value=VALUE)`

org.apache.coyote.http11.Http11Protocol.MAX_HEADER_SIZE	HTTP ヘッダーのバイト単位の最大サイズ。この値を超えると、**ArrayOutOfBoundsException** によって解析に失敗します。デフォルト値は **8192** バイトです。
	管理 CLI コマンド: /subsystem=undertow/server=default-server/http-listener=default:write-attribute(name=max-header-size,value=VALUE) /subsystem=undertow/server=default-server/https-listener=default:write-attribute(name=max-header-size,value=VALUE) /subsystem=undertow/server=default-server/ajp-listener=default:write-attribute(name=max-header-size,value=VALUE)
org.apache.coyote.http11.Http11Protocol.COMPRESSION	HTTP コネクターでの簡単な圧縮の使用を許可します。デフォルト値は **off** です。 **on** を使用すると圧縮を有効にすることができ、条件に応じて有効するか、常に強制的に有効にすることができます。
	管理 CLI を使用したフィルターの設定: # Create a filter /subsystem=undertow/configuration=filter/gzip=gzipfilter:add() /subsystem=undertow/server=default-server/host=default-host/filter-ref=gzipfilter:add()
org.apache.coyote.http11.Http11Protocol.COMPRESSION_RESTRICTED_UA	圧縮されたコンテンツを受け取らないユーザーエージェント regexps。デフォルト値は空です。
	管理 CLI を使用したフィルターにおける述語の設定 # Use a predicate in a filter /subsystem=undertow/configuration=filter/gzip=gzipfilter:add() /subsystem=undertow/server=default-server/host=default-host/filter-ref=gzipfilter:add(predicate="regex[pattern='AppleWebKit',value=%{i,User-Agent}]")
org.apache.coyote.http11.Http11Protocol.COMPRESSION_MIME_TYPES	圧縮可能なコンテンツのコンテンツタイプ接頭辞。デフォルト値は **text/html,text/xml,text/plain** です。

	管理 CLI を使用したフィルターにおける述語の設定 ``` # Use a predicate in a filter /subsystem=undertow/configuration=filter/gzip=gzipfilter:add() /subsystem=undertow/server=default-server/host=default-host/filter-ref=gzipfilter:add(predicate="regex[pattern='text/html',value=%{o,Content-Type}]") ```
org.apache.coyote.http11.Http11Protocol.COMPRESSION_MIN_SIZE	圧縮されるコンテンツの最小サイズ。デフォルト値は **2048** バイトです。
	管理 CLI を使用したフィルターにおける述語の設定 ``` # Use a predicate in a filter /subsystem=undertow/configuration=filter/gzip=gzipfilter:add() /subsystem=undertow/server=default-server/host=default-host/filter-ref=gzipfilter:add(predicate="max-content-size[value=MIN_SIZE]") ```
org.apache.coyote.http11.DEFAULT_CONNECTION_TIMEOUT	デフォルトのソケットタイムアウト。デフォルト値は **60000** にミリ秒です。
	管理 CLI コマンド: ``` /subsystem=undertow/server=default-server/http-listener=default:write-attribute(name=no-request-timeout,value=VALUE) /subsystem=undertow/server=default-server/https-listener=default:write-attribute(name=no-request-timeout,value=VALUE) /subsystem=undertow/server=default-server/ajp-listener=default:write-attribute(name=no-request-timeout,value=VALUE) ```
org.jboss.as.web.deployment.DELETE_WORK_DIR_ONCONTEXTDESTROY	このプロパティーを使用して **.java** および **.class** ファイルを削除し、JSP リソースが確実に再コンパイルされるようにします。デフォルト値は **false** です。**keep-alive** のデフォルトのソケットタイムアウトです。デフォルト値は **-1** ミリ秒で、デフォルトのソケットタイムアウトを使用します。
	同等の設定はありません。
org.apache.tomcat.util.buf.StringCache.trainThreshold	**toString()** が指定の回数呼び出された後にキャッシュがアクティベートされます。デフォルト値は **100000** です。
	同等の設定はありません。

表A.2 EL システムプロパティーのマップ

JBoss EAP 6 システムプロパティー	説明
	JBoss EAP 7 での同等設定
org.apache.el.parser.COERCE_TO_ZERO	**true** を指定した場合、強制的に式を数字に変換すると、仕様の要件どおりに空の文字列 ("") とnull は強制的にゼロに変換されます。値の指定がない場合は、デフォルト値の **true** が使用されます。
	システムプロパティーは有効で、EL によって処理されます。

表A.3 JSP システムプロパティー

JBoss EAP 6 システムプロパティー	説明
	JBoss EAP 7 での同等設定
org.apache.jasper.compiler.Generator.VAR_EXPRESSIONFACTORY	式言語の式ファクトリーに使用される変数の名前。値の指定がない場合は、デフォルト値の **_el_expressionfactory** が使用されます。
	システムプロパティーの変更はありません。
org.apache.jasper.compiler.Generator.VAR_INSTANCEMANAGER	インスタンスマネージャーファクトリーに使用する変数の名前。値の指定がない場合は、デフォルト値の **_jsp_instancemanager** が使用されます。
	システムプロパティーの変更はありません。
org.apache.jasper.compiler.Parser.STRICT_QUOTE_ESCAPING	**false** を指定した場合、JSP 属性の引用符をエスケープ処理する要件が緩和され、必要な引用符がなくてもエラーが発生しません。値の指定がない場合は、仕様に準拠するデフォルトの **true** が使用されます。
	システムプロパティーの変更はありません。
org.apache.jasper.Constants.DEFAULT_TAG_BUFFER_SIZE	**org.apache.jasper.Constants.DEFAULT_TAG_BUFFER_SIZE** を超えたタグバッファーは破棄され、デフォルトサイズの新しいバッファーが作成されます。値の指定がない場合は、デフォルト値の **512** が使用されます。
	システムプロパティーの変更はありません。
org.apache.jasper.runtime.JspFactoryImpl.USE_POOL	**true** の場合、ThreadLocal PageContext プールが使用されます。値の指定がない場合は、デフォルト値の **true** が使用されます。

	システムプロパティーの変更はありません。
org.apache.jasper.runtime.JspFactoryImpl.POOL_SIZE	ThreadLocal PageContext のサイズ。値の指定がない場合は、デフォルト値の **8** が使用されます。
	システムプロパティーの変更はありません。
org.apache.jasper.Constants.JSP_SERVLET_BASE	JSP から生成されたサーブレットのベースクラス。値の指定がない場合は、デフォルト値の **org.apache.jasper.runtime.HttpJspBase** が使用されます。
	システムプロパティーの変更はありません。
org.apache.jasper.Constants.SERVICE_METHOD_NAME	ベースクラスによって呼び出されるサービスメソッドの名前。値の指定がない場合は、デフォルト値の **_jspService** が使用されます。
	システムプロパティーの変更はありません。
org.apache.jasper.Constants.SERVLET_CLASSPATH	JSP のクラスパスが提供される ServletContext 属性の名前。値の指定がない場合は、デフォルト値の **org.apache.catalina.jsp_classpath** が使用されます。
	システムプロパティーの変更はありません。
org.apache.jasper.Constants.JSP_FILE	サーブレット定義の **<jsp-file>** 要素に対するリクエスト属性の名前。リクエストに存在する場合は、**request.getServletPath()** によって返された値をオーバーライドし、実行される JSP ページを選択します。値の指定がない場合は、デフォルト値の **org.apache.catalina.jsp_file** が使用されます。
	システムプロパティーの変更はありません。
org.apache.jasper.Constants.PRECOMPILE	JSP エンジンがサーブレットを事前生成し、呼び出しは行わないようにするクエリーパラメーターの名前。値の指定がない場合は、デフォルト値の **org.apache.catalina.jsp_precompile** が使用されます。
	システムプロパティーの変更はありません。
org.apache.jasper.Constants.JSP_PACKAGE_NAME	コンパイルされた JSP ページのデフォルトのパッケージ名。値の指定がない場合は、デフォルト値の **org.apache.jsp** が使用されます。
	システムプロパティーの変更はありません。

org.apache.jasper.Constants.TAG_FILE_PACKAGE_NAME	タグファイルから生成されたタグハンドラーのデフォルトのパッケージ名。値の指定がない場合は、デフォルトの **org.apache.jsp.tag** が使用されます。
	システムプロパティーの変更はありません。
org.apache.jasper.Constants.TEMP_VARIABLE_NAME_PREFIX	生成された一時的な変数名に使用する接頭辞。値の指定がない場合は、デフォルト値の **_jspx_temp** が使用されます。
	システムプロパティーの変更はありません。
org.apache.jasper.Constants.USE_INSTANCE_MANAGER_FOR_TAGS	**true** を指定した場合、タグハンドラーインスタンスの取得にインスタンスマネージャーが使用されます。値の指定がない場合は **true** が使用されます。
	システムプロパティーの変更はありません。
org.apache.jasper.Constants.INJECT_TAGS	**true** の場合、タグに指定されたアノテーションは処理およびインジェクトされます。簡単なタグを使用する場合やタグプーリングが無効になっている場合はパフォーマンスに影響することがあります。値の指定がない場合は **false** が使用されます。
	システムプロパティーの変更はありません。

表A.4 セキュリティーシステムプロパティーのマップ

JBoss EAP 6 システムプロパティー	**説明**
	JBoss EAP 7 での同等設定
org.apache.catalina.connector.RECYCLE_FACADES	**true** に指定された場合や、セキュリティーマネージャーが使用中の場合は、各リクエストに新しいファサードオブジェクトが作成されます。値の指定がない場合は、デフォルト値の **false** が使用されます。
	同等の設定はありません。
org.apache.catalina.connector.CoyoteAdapter.ALLOW_BACKSLASH	**true** を指定した場合、「\」文字はパス区切り文字として使用できます。値の指定がない場合は、デフォルト値の **false** が使用されます。
	同等の設定はありません。
org.apache.tomcat.util.buf.UDecoder.ALLOW_ENCODED_SLASH	**true** を指定した場合、「%2F」および「%5C」はパス区切り文字として使用できます。値の指定がない場合は、デフォルト値の **false** が使用されます。

	管理 CLI コマンド:
	
	`/subsystem=undertow/server=default-server/http-listener=default:write-attribute(name=allow-encoded-slash,value=VALUE)`
	`/subsystem=undertow/server=default-server/https-listener=default:write-attribute(name=allow-encoded-slash,value=VALUE)`
	`/subsystem=undertow/server=default-server/ajp-listener=default:write-attribute(name=allow-encoded-slash,value=VALUE)`

org.apache.catalina.STRICT_SERVLET_COMPLIANCE	値の指定がない場合は **true** が使用されます。**true** の場合は次のアクションが発生します。元のリクエストまたは応答がラップされるようにするため、アプリケーションディスパッチャーに渡されるラップされたリクエストまたは応答オブジェクトはすべてチェックされます。(SRV.15.2.22.1) 文字エンコーディングの指定がない場合に **Response.getWriter()** を呼び出すと、後続の **Response.getCharacterEncoding()** が **ISO-8859-1** を返し、**Content-Type** 応答ヘッダーに **charset=ISO-8859-1** コンポーネントが含まれます。リクエストが明示的にセッションにアクセスしたかどうかに関わらず、セッションに関連する各リクエストによって、セッションの最後のアクセス時間が更新されます。(SRV.7.6)
	デフォルトで対応。
org.apache.catalina.core.StandardWrapperValve.SERVLET_STATS	**true** を指定した場合または org.apache.catalina.STRICT_SERVLET_COMPLIANCE が **true** の場合、ラッパーは各サーブレットのJSR-77 統計を収集します。値の指定がない場合は、デフォルト値の **false** が使用されます。
	同等の設定はありません。
org.apache.catalina.session.StandardSession.ACTIVITY_CHECK	**true** を指定した場合または **org.apache.catalina.STRICT_SERVLET_COMPLIANCE** が **true** の場合、Tomcat は各セッションのアクティブなリクエスト数を追跡します。セッションが有効であるかを判断するとき、アクティブなセッションが 1 つ以上あるセッションは常に有効であると見なされます。値の指定がない場合は、デフォルト値の **false** が使用されます。
	同等の設定はありません。

A.5. リリース間の互換性および相互運用性

ここでは、JBoss EAP 5、JBoss EAP 6、および JBoss EAP 7 リリース間での、クライアントおよびサーバー EJB 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。

IIOP 上の EJB リモーティング

以下の設定では問題が発生しません。

- JBoss EAP 5 クライアントから JBoss EAP 7 サーバーへの接続
- JBoss EAP 6 クライアントから JBoss EAP 7 サーバーへの接続
- JBoss EAP 7 クライアントから JBoss EAP 6 サーバーへの接続
- JBoss EAP 7 クライアントから JBoss EAP 5 サーバーへの接続

JNDI を使用した EJB リモーティング
以下の設定では問題が発生しません。

- JBoss EAP 6 クライアントから JBoss EAP 7 サーバーへの接続
- JBoss EAP 7 クライアントから JBoss EAP 6 サーバーへの接続

JBoss EAP 6 では、EJB 3.1 仕様のサポートが提供され、標準化されたグローバル JNDI ネームスペースが導入されました。JBoss EAP 7 でも標準化されたグローバル JNDI ネームスペースは使用されますが、JNDI ネームスペースの名前が変更になったため、以下の設定は互換性がありません。

- JBoss EAP 5 クライアントから JBoss EAP 7 または JBoss EAP 6 サーバーへの接続
- JBoss EAP 7 または JBoss EAP 6 クライアントから JBoss EAP 5 サーバーへの接続

標準化された JNDI ネームスペースの変更に関する詳細は、JBoss EAP 6 移行ガイドの **JNDI** の変更を参照してください。

@WebService を使用した EJB リモーティング
以下の設定では問題が発生しません。

- JBoss EAP 5 クライアントから JBoss EAP 7 サーバーへの接続
- JBoss EAP 6 クライアントから JBoss EAP 7 サーバーへの接続
- JBoss EAP 7 クライアントから JBoss EAP 6 サーバーへの接続
- JBoss EAP 7 クライアントから JBoss EAP 5 サーバーへの接続

メッセージングスタンドアロンクライアント
以下の設定では問題が発生しません。

- JBoss EAP 6 クライアントから JBoss EAP 7 サーバーへの接続
- JBoss EAP 7 クライアントから JBoss EAP 6 サーバーへの接続

以下の設定では、クライアントが汎用 JMS API ではなくメッセージングブローカー専用の HornetQ API を使用すれば接続が可能です。しかし、JBoss EAP 7 に同梱される JBoss EAP レガシー JNDI ネーミング拡張を使用して JNDI ルックアップに対応する必要があります。

- JBoss EAP 5 クライアントから JBoss EAP 7 サーバーへの接続

プロトコル互換性の問題があるため、JBoss EAP 7 のビルトインメッセージングは JBoss EAP 5 に同梱された HornetQ 2.2.x へは接続できません。そのため、以下の設定は互換性がありません。

- JBoss EAP 7 クライアントから JBoss EAP 5 サーバーへの接続

メッセージング MDB

以下の設定では問題が発生しません。

- JBoss EAP 6 クライアントから JBoss EAP 7 サーバーへの接続
- JBoss EAP 7 クライアントから JBoss EAP 6 サーバーへの接続

以下の設定では、クライアントが汎用 JMS API ではなくメッセージングブローカー専用の HornetQ API を使用すれば接続が可能です。しかし、JBoss EAP 7 に同梱される JBoss EAP レガシー JNDI ネーミング拡張を使用して JNDI ルックアップに対応する必要があります。

- JBoss EAP 5 クライアントから JBoss EAP 7 サーバーへの接続

プロトコル互換性の問題があるため、JBoss EAP 7 のビルトインメッセージングは JBoss EAP 5 に同梱された HornetQ 2.2.x へは接続できません。そのため、以下の設定は互換性がありません。

- JBoss EAP 7 クライアントから JBoss EAP 5 サーバーへの接続

JMS ブリッジ

以下の設定では問題が発生しません。

- JBoss EAP 5 クライアントから JBoss EAP 7 サーバーへの接続
- JBoss EAP 6 クライアントから JBoss EAP 7 サーバーへの接続
- JBoss EAP 7 クライアントから JBoss EAP 6 サーバーへの接続
- JBoss EAP 7 クライアントから JBoss EAP 5 サーバーへの接続

Revised on 2019-11-27 13:00:21 CET