JBoss Enterprise SOA Platform 5

インストールと設定ガイド

このガイドはインストールチーム向けです。

Last Updated: 2023-03-28

JBoss Enterprise SOA Platform 5 インストールと設定ガイド

このガイドはインストールチーム向けです。

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

法律上の通知

概要

本書では、JBoss Enterprise SOA Platform 製品のすべてのインストールおよび設定オプションについて説明します。

目次

はじめに

1. ドキュメント規則

本ガイドでは、いくつかの規則を使用して特定の単語やフレーズを強調表示し、特定の情報への注意を促しています。

1.1. 表記規則

特定の単語や句への注意を促すために 4 つの表記慣習を使用しています。これらの規則や、これらが適用される状況は以下のとおりです。

等幅ボールド

シェルコマンド、ファイル名、パスなど、システム入力を強調表示するために使用されます。キーとキーの組み合わせを強調表示するためにも使用されます。以下に例を示します。

> 現在の作業ディレクトリーのファイル **my_next_bestselling_novel** の内容を表示するには、シェルプロンプトで **cat my_next_bestselling_novel** コマンドを入力し、**Enter** を押してコマンドを実行します。

上記には、ファイル名、シェルコマンドおよびキーが含まれます。これはすべて等幅ボールドで表示され、コンテキストにより区別可能なものになります。

キーの組み合わせは、各部分をつなぐプラス記号によって、個別のキーと区別できます。以下に例を示します。

> **Enter** を押してコマンドを実行します。
>
> **Ctrl**+**Alt**+**F2** を押して、仮想ターミナルに切り替えます。

最初の例では、押す特定のキーを強調表示しています。2 つ目の例は、同時に押す 3 つのキーのセットというキーの組み合わせを強調表示しています。

ソースコードの場合、段落内で記述されるクラス名、メソッド、関数、変数名、および戻り値は、上記のように **等幅ボールド** で示されます。以下に例を示します。

> ファイル関連のクラスには、ファイルシステムの **filesystem**、ファイルの **file**、ディレクトリーの **dir** が含まれます。各クラスには、独自の関連付けられたパーミッションセットがあります。

プロポーショナルボールド

これは、アプリケーション名、ダイアログボックステキスト、ラベルが付いたボタン、チェックボックスおよびラジオボタン、メニュータイトルおよびサブメニュータイトルなど、システムで発生した単語またはフレーズを示します。以下に例を示します。

> メインメニューバーから System → Preferences → Mouse を選択し、Mouse Preferences を起動します。**Buttons** タブで、**Left-handed mouse** チェックボックスを選択し、**Close** をクリックしてメインのマウスボタンを左から右に切り替えます (マウスを左手で使い易くします)。
>
> 特殊文字を gedit ファイルに挿入するには、メインメニューバーから Applications → Accessories → Character Map を選択します。次に、Character Map メニューバーから Search → Find... を選択し、**Search** フィールドに文字の名前を入力して **Next** をクリックします。目的の文字が **Character Table** で強調表示されます。この強調表示し

た文字をダブルクリックして **Text to copy** フィールドに配置し、**Copy** ボタンをクリックします。ここでドキュメントに戻り、**gedit** メニューバーから **Edit → Paste** を選択します。

上記のテキストにはアプリケーション名、システム全体のメニュー名および項目、アプリケーション固有のメニュー名、GUI インターフェイス内のボタンおよびテキストなどがあります。すべては proportional bold で示され、コンテキストと区別できます。

等幅ボールドイタリック または ***プロポーショナルボールドイタリック***

等幅ボールドまたはプロポーショナルボールドのいずれでも、イタリックが追加されると、置換または変数テキストを意味します。イタリックは、状況に応じて変化するテキストや、文字を入力しないテキストを表します。以下に例を示します。

ssh を使用してリモートマシンに接続するには、シェルプロンプトで **ssh *username@domain.name*** を入力します。リモートマシンが **example.com** で、そのマシン上でのユーザー名が john の場合は、**ssh john@example.com** と入力します。

mount -o remount *file-system* コマンドにより、指定したファイルシステムが再マウントされます。たとえば、**/home** ファイルシステムを再マウントする場合、コマンドは **mount -o remount /home** となります。

現在インストールされているパッケージのバージョンを表示するには、**rpm -q *package*** コマンドを使用します。これにより、***package-version-release*** のような結果が返されます。

上記の太字のイタリック体の用語、username、domain.name、file-system、package、version、および release に注意してください。各単語はプレースホルダーで、コマンドの発行時に入力するテキストまたはシステムによって表示されるテキストのどちらかになります。

作業のタイトルを示す標準的な使用法のほかに、イタリックは新用語と重要な用語の最初の使用を示します。以下に例を示します。

Publican は *DocBook* 公開システムです。

1.2. 引用規則

端末の出力およびソースコードの一覧は、周りのテキストから視覚的に表示されます。

ターミナルに送信される出力は **mono-spaced roman** に設定され、以下のように表示されます。

```
books       Desktop   documentation  drafts  mss    photos   stuff  svn
books_tests  Desktop1  downloads      images  notes  scripts  svgs
```

ソースコードの一覧も **mono-spaced roman** に設定されますが、以下のように構文の強調表示が追加されます。

```
static int kvm_vm_ioctl_deassign_device(struct kvm *kvm,
                 struct kvm_assigned_pci_dev *assigned_dev)
{
        int r = 0;
        struct kvm_assigned_dev_kernel *match;

        mutex_lock(&kvm->lock);

        match = kvm_find_assigned_dev(&kvm->arch.assigned_dev_head,
```

```
                                assigned_dev->assigned_dev_id);
        if (!match) {
                printk(KERN_INFO "%s: device hasn't been assigned before, "
                  "so cannot be deassigned\n", __func__);
                r = -EINVAL;
                goto out;
        }

        kvm_deassign_device(kvm, match);

        kvm_free_assigned_device(kvm, match);

out:
        mutex_unlock(&kvm->lock);
        return r;
}
```

1.3. 注記および警告

最後に、3 つの視覚的スタイルを使用して、見落とす可能性のある情報に注意を促します。

注記

注記とは、タスクへのヒント、ショートカット、または代替アプローチです。注意を無視しても悪い結果を招くことはありませんが、便利なヒントを見逃してしまう可能性があります。

重要

見落としやすい詳細のある重要なボックス: 現行セッションにのみ適用される設定変更や、更新を適用する前に再起動が必要なサービスなどです。Important というラベルが付いたボックスを無視しても、データが失われることはありませんが、スムーズな操作が行えないことがあります。

警告

警告は無視すべきではありません。警告を無視すると、データが失われる可能性があります。

2. ヘルプの利用とフィードバック提供

2.1. ヘルプが必要ですか？

本ガイドで説明されている手順で問題が発生した場合は、Red Hat カスタマーポータル http://access.redhat.com にアクセスしてください。カスタマーポータルでは、以下を行うことができます。

- Red Hat 製品に関する技術サポート記事の知識ベースの検索または閲覧。

- Red Hat グローバルサポートサービス (GSS) へのサポートケースの送信。
- その他の製品ドキュメントへのアクセス。

Red Hat は、Red Hat のソフトウェアおよびテクノロジーについて、多くの電子メーリングリストも提供しています。一般に公開されているメーリングリストの一覧は、https://www.redhat.com/mailman/listinfoを参照してください。メーリングリストの名前をクリックして、その一覧をサブスクライブするか、またはメーリングリストのアーカイブにアクセスします。

2.2. ご意見をお寄せください

本ガイドで誤字脱字を発見されたり、このマニュアルを改善するための提案をお持ちの場合は、弊社までご連絡ください。JBoss Enterprise SOA Platform の製品に対して、http://bugzilla.redhat.com/ から Bugzilla レポートを送信してください。

バグレポートを送信する際には、マニュアル識別子(『Installation_and_Configuration_Guide』)を記載してください。

本ガイドを改善するためのご意見やご提案をお寄せいただく場合は、できるだけ具体的にご説明ください。エラーが見つかった場合は、簡単に確認できるように、セクション番号と前後のテキストを含めてください。

第1章 はじめに

1.1. ビジネス統合

動的かつアジャイルなビジネスインフラストラクチャーを提供するためには、異なるアプリケーションとデータソースが最小限のオーバーヘッドで相互に通信できるように、サービス指向のアーキテクチャーを用意することが重要です。

JBoss Enterprise SOA Platform は、ビジネスプロセスの変更に対応するために、継続的に再利用することなく、ビジネスサービスをオーケストレーションできるフレームワークです。JBoss Enterprise SOA Platform では、ビジネスルールとメッセージの変換およびルーティング機能を使用することで、複数のソースからビジネスデータを操作できます。

バグの報告

1.2. サービス指向アーキテクチャーとは

はじめに

SOA (*Service Oriented Architecture*) は単一のプログラムまたはテクノロジーではありません。ソフトウェア設計パラダイムと見なします。

すでに分かっているように、*ハードウェアバス* は、複数のシステムとサブシステムを結び付ける物理コネクターです。これを使用する場合は、システムのペア間で多数のポイントツーポイントコネクターを使用する代わりに、各システムを中央バスに接続するだけです。*エンタープライズサービスバス* (ESB) は、ソフトウェアとまったく同じことを行います。

アーキテクトは、メッセージングシステムの上記のアーキテクチャー層にあります。このメッセージングシステムは、このメッセージングシステムを使用することでサービス間の *非同期通信* を容易にします。実際、アーキテクトを使用している場合、すべてを概念的に、*サービス*（このコンテキストではアプリケーションソフトウェア）またはサービス間で送信される *メッセージ* のいずれかです。サービスは接続アドレスとして一覧表示されます (*エンドポイント参照* と呼ばれています)。

このコンテキストでは、サービスは必ずしも Web サービスであるとは限らないことに注意することが重要です。File Transfer Protocol や Java Message Service などのトランスポートを使用する他のタイプのアプリケーションもサービスになる可能性があります。

注記

この時点で、エンタープライズサービスバスがサービス指向のアーキテクチャーと同じ場合は、妨げられる場合があります。回答は Not exactly です。 アーキテクトは、サービス指向のアーキテクチャーを形成しません。代わりに、ツールを多数使用して構築できます。特に、SOA が必要とする *loose-coupling* および *非同期メッセージを渡すこと* が容易になります。SOA は常にソフトウェア以外のものと考えてください。これは一連の原則、パターン、およびベストプラクティスです。

バグの報告

1.3. サービス指向アーキテクチャーの重要なポイント

以下は、サービス指向のアーキテクチャーの主要なコンポーネントです。

1. 交換される *メッセージ*
2. サービスリクエスターおよびプロバイダーとして動作する *エージェント*
3. メッセージを送受信できるようにする *共有トランスポートメカニズム*。

バグの報告

1.4. JBOSS ENTERPRISE SOA PLATFORM とは

JBoss Enterprise SOA Platform は、エンタープライズアプリケーションインテグレーション (EAI) およびサービス指向アーキテクチャー (SOA) ソリューションを開発するためのフレームワークです。これは、エンタープライズサービスバス (JBoss Warehouse) およびビジネスプロセス自動化インフラストラクチャーで設定されます。これにより、ビジネスサービスの構築、デプロイ、統合、オーケストレーションを行うことができます。

バグの報告

1.5. SERVICE-ORIENTED ARCHITECTURE PARADIGM

SOA (Service-ient Architecture)は、リクエスター、プロバイダー、およびブローカーの 3 つのロールで設定されます。

サービスプロバイダー

サービスプロバイダーはサービスへのアクセスを許可し、サービスの説明を作成し、サービスブローカーに公開します。

サービスリクエスター

サービスリクエスターは、サービスブローカーが提供するサービスの説明を検索して、サービスを検出します。リクエスターはサービスプロバイダーが提供するサービスにバインドするロールも果たします。

サービスブローカー

サービスブローカーは、サービスの説明のレジストリーをホストします。リクエスターをサービスプロバイダーにリンクします。

バグの報告

1.6. コアおよびコンポーネント

JBoss Enterprise SOA Platform は、データ統合のニーズに対応する包括的なサーバーを提供します。基本的なレベルでは、Enterprise Service Bus を介してビジネスルールを更新し、メッセージをルーティングすることができます。

JBoss Enterprise SOA Platform の中心となるのは、Enterprise Service Bus です。JBoss (ESB) はメッセージの送受信を行う環境を作成します。メッセージにアクションを適用して変換し、サービス間でルーティングすることができます。

JBoss Enterprise SOA Platform を設定するコンポーネントは複数あります。ESB とともに、レジストリ (jUDDI)、変換エンジン (Smooks)、メッセージキュー (HornetQ)、BPEL エンジン (Riftsaw) が用意されています。

バグの報告

1.7. JBOSS ENTERPRISE SOA PLATFORM のコンポーネント

- 完全な Java EE 準拠のアプリケーションサーバー (JBoss Enterprise Application Platform)
- Enterprise Service Bus (JBoss ESB)
- ビジネスプロセス管理システム (jBPM)
- ビジネスルールエンジン (JBoss ルール)
- オプションの JBoss Enterprise Data Services (EDS) 製品のサポート。

バグの報告

1.8. JBOSS ENTERPRISE SOA PLATFORM の機能

JBoss Enterprise Service Bus (ESB)

ドメインはサービス間でメッセージを送信し、それらを変換して、異なるタイプのシステムで処理できるようにします。

Business Process Execution Language (BPEL)

Web サービスを使用して、この言語を使用してビジネスルールをオーケストレーションできます。これは、ビジネスプロセス命令を簡単に実行するために SOA に含まれています。

Java Universal Description、Discovery and Integration (jUDDI)

これは SOA のデフォルトサービスレジストリーです。ここで説明する内容は、インストラクター上のサービスに関する情報をすべて保管する場所です。

Smooks

この変換エンジンは SOA と組み合わせて使用してメッセージを処理できます。また、メッセージを分割して正しい宛先に送信するためにも使用できます。

JBoss ルール

これは、SOA にパッケージ化されたルールエンジンです。受信するメッセージからデータを推測して、実行する必要があるアクションを判別できます。

バグの報告

1.9. JBOSS ENTERPRISE SOA PLATFORM の JBOSSESB コンポーネントの機能

JBoss Enterprise SOA Platform の JBossESB コンポーネントは以下をサポートします。

- 複数のトランスポートおよびプロトコル
- リスナーアクションモデル (これにより、サービスを相互に選択可能)
- コンテンツベースのルーティング (JBoss Rules エンジン、XPath、Regex、および Smooks 経由)
- サービスオーケストレーション機能を提供するために JBoss Business Process Manager (jBPM) との統合
- ビジネスルールの開発機能を提供するために、JBoss ルールとの統合
- BPEL エンジンとの統合。

さらに、新しいデプロイメントにレガシーシステムを統合し、同期または非同期で通信できるようにします。

さらに、エンタープライズサービスバスは、以下を可能にするインフラストラクチャーおよびツールセットを提供します。

- さまざまなトランスポートメカニズム (電子メールや JMS など) と連携するよう設定されます。
- 汎用オブジェクトリポジトリーとして使用します。
- プラグ可能なデータ変換メカニズムを実装できるようにします。
- 対話のロギングをサポートします。

重要

ソースコードには、**org.jboss.internal.soa.esb** と **org.jboss.soa.esb** の 2 つのツリーがあります。**org.jboss.internal.soa.esb** パッケージの内容をそのまま使用します。これは、通知なしに変更される可能性があるためです。これとは対照的に、**org.jboss.soa.esb** パッケージ内のすべての内容は、Red Hat の非推奨ポリシーでカバーされます。

バグの報告

1.10. タスク管理

JBoss SOA は、影響を受けるすべてのシステムで汎用的に実行するタスクを指定することにより、タスクを簡素化します。つまり、ユーザーは各ターミナルで個別に実行するようにタスクを設定する必要はありません。ユーザーは、Web サービスを使用してシステムを簡単に接続できます。

JBoss SOA を使用して各マシンに複数回ではなく、ネットワーク全体でトランザクションを 1 回委譲することで、時間を節約できます。これにより、エラーが発生する可能性も低くなります。

バグの報告

1.11. 統合のユースケース

ACME Equity は、大規模な経理サービスです。会社には多くのデータベースやシステムがあります。旧式の COBOL ベースのレガシーシステムや、近年で小規模な企業で取得されるデータベースもあります。ビジネスルールが頻繁に変化するため、これらのデータベースを統合することは困難でコストがかかります。会社は、クライアント向け e-commerce の Web サイトを新たに開発することを希望していますが、現在使用している既存のシステムとは同期しない場合があります。

会社は、安価なソリューションを希望していますが、企業セクターの厳密な規制およびセキュリティー要件に準拠するソリューションを検討しています。会社が望ましくないことは、レガシーデータベースおよびシステムを接続するために glob コードを作成および維持する必要があることです。

JBoss Enterprise SOA Platform は、これらのレガシーシステムを新しいお客様の Web サイトに統合するためにミドルウェアレイヤーとして選択されました。フロントエンドシステムとバックエンドシステムの間のブリッジを提供します。JBoss Enterprise SOA Platform で実装されたビジネスルールは、すぐに簡単に更新できます。

その結果、SOA の統合方法により、古いシステムは新しいシステムと同期できるようになりました。1 カ月あたり数万のトランザクションであっても、ボトルネックはありません。XML、JMS、FTP などのさまざまな統合タイプは、システム間でデータを移動するために使用されます。数多くのエンタープライズ標準のメッセージングシステムの 1 つを JBoss Enterprise SOA Platform にプラグインすることで、柔軟性を高めることができます。

さらに利点は、既存のインフラストラクチャーにより多くのサーバーやデータベースが追加されると、システムを簡単にスケールアップできることです。

バグの報告

1.12. ビジネス環境での JBOSS ENTERPRISE SOA PLATFORM の使用

エラーメッセージが発生する可能性が低いサービスの実装により、コストを削減できます。生産性とソーシングオプションの強化により、継続的なコストを削減できます。

情報およびビジネスプロセスは、接続が増加するため、迅速に共有できます。これは Web サービスによって強化され、クライアントを簡単に接続するために使用できます。

レガシーシステムは Web サービスと組み合わせて使用して、異なるシステムが同じ言語にピークにすることができます。これにより、システムの同期に必要なアップグレードおよびカスタムコードの量が減ります。

バグの報告

パート I. はじめに

第2章 JBOSS ENTERPRISE SOA PLATFORM の紹介

2.1. 対象読者

本書は、JBoss Enterprise SOA Platform 製品をインストールするときに利用可能なすべてのオプションに関する包括的なガイドを必要とするインストールプロジェクトチーム向けに作成されています。

バグの報告

2.2. 本書のねらい

本書は、Red Hat Enterprise Linux または Microsoft Windows コンピューターに JBoss Enterprise SOA Platform をインストールして設定する方法を説明します。ここでは、利用可能なすべてのインストールオプションを説明します。また、テスト環境と実稼働環境の両方で製品の基本操作についても説明します。

バグの報告

2.3. データのバックアップ

警告

Red Hat は、本ガイドに記載されている設定タスクを行う前に、システム設定とデータのバックアップを行うことを推奨します。

バグの報告

2.4. SERVICE-ORIENTED ARCHITECTURE PARADIGM

SOA (Service-ient Architecture)は、リクエスター、プロバイダー、およびブローカーの 3 つのロールで設定されます。

サービスプロバイダー

サービスプロバイダーはサービスへのアクセスを許可し、サービスの説明を作成し、サービスブローカーに公開します。

サービスリクエスター

サービスリクエスターは、サービスブローカーが提供するサービスの説明を検索して、サービスを検出します。リクエスターはサービスプロバイダーが提供するサービスにバインドするロールも果たします。

サービスブローカー

サービスブローカーは、サービスの説明のレジストリーをホストします。リクエスターをサービスプロバイダーにリンクします。

バグの報告

2.5. 初期状態のアクション

追加設定なしのアクションは、JBoss Enterprise SOA Platform 製品に事前にパッケージ化されたアクションの汎用要素です。これらをサービスですぐに使用することも、ニーズに合わせてカスタマイズすることもできます。

バグの報告

2.6. JBOSS ENTERPRISE SOA PLATFORM の非ボックスアクション

SOA Platform に実装されている追加設定なしのアクションは、以下の機能グループに分類されます。

トランスフォーマーとコンバーター

トランスフォーマーとコンバーターアクションを使用して、メッセージデータをある形式から別の形式に変更します。

ビジネスプロセス管理

ソフトウェアを jBPM と統合する場合は、ビジネスプロセス管理アクションを使用します。

スクリプト

スクリプトアクションを使用して、サポートされているスクリプト言語で書かれたタスクを自動化します。

サービス

コードを Enterprise Java Beans と統合する際に、サービスアクションを使用します。

Routing

メッセージデータを宛先サービスに移動する際にルーティングアクションを使用します。

Notifier

データを、認識しない宛先に送信する場合は、通知アクションを使用します。

Web Services/SOAP

Web サービスをサポートする必要がある場合は、Web サービスアクションを使用します。

バグの報告

第3章 前提条件

3.1. JBOSS ENTERPRISE SOA PLATFORM をインストールするための前提条件

JBoss Enterprise SOA Platform をインストールして実行するには、コンピューターに以下のものが必要です。

- サポートされている Java 仮想マシン
- サポートされている Java Development Kit (クイックスタートの実行用)
- サポートされているデータベースサーバー (JBoss サーバーを実行するために必要)
- Apache Ant 1.7 以降 (データベーススキーマ設定ツールを実行し、JBoss ESB クイックスタートサンプルをデプロイするために必要)
- アーカイブツール (FileRoller、ark、tar など)。(圧縮ファイルの内容を解凍するために必要です)
- JBoss Developer Studio 5.0。(Red Hat Customer Portal https://access.redhat.com/home から入手してください)

Red Hat は、JBoss Enterprise SOA Platform を複数の異なるハードウェアプラットフォーム、Java 仮想マシン、オペレーティングシステム、およびデータベースに対してテストおよび認定しています。これは進行中のプロセスであり、サポートされる環境のリストは常に増え続けています。現在サポートされている環境のリストは、http://www.jboss.com/products/platforms/soa/testedconfigurations/ を参照してください。

バグの報告

3.2. RED HAT ENTERPRISE LINUX への OPEN JDK のインストール

手順3.1 Red Hat Enterprise Linux への Open JDK のインストール

1. **ベースチャンネルにサブスクライブする**
   RHN ベースチャンネルから OpenJDK を取得します。(Red Hat Enterprise Linux のインストールは、デフォルトでこのチャンネルにサブスクライブされています。)
2. **パッケージをインストールする**
   yum ユーティリティーを使用して OpenJDK をインストールします。**yum install java-1.7.0-openjdk-devel**
3. **OpenJDK がシステムのデフォルトになっていることを確認する**
   正しい JDK がシステムのデフォルトとして設定されていることを確認するには、root としてログインし、alternatives コマンド **/usr/sbin/alternatives --config java** を実行します。

   /usr/lib/jvm/jre-1.6.0-openjdk/bin/java を選択します。

   javac **/usr/sbin/alternatives --config javac** を設定します。

   /usr/lib/jvm/java-1.6.0-openjdk/bin/java を選択します。

バグの報告

3.3. APACHE ANT

Apache Ant ("Another Neat Tool") は、Java ベースのビルドツールです。このツールは、ソフトウェアのコンパイルのプロセスを自動化するように設計されています。開発者は、カスタムビルド命令を受け取る XML ベースのビルドファイルを提供する必要があります。詳細は、http://ant.apache.org/ を参照してください。

バグの報告

3.4. APACHE ANT のインストール

手順3.2 Red Hat Enterprise Linux での Apache Ant のインストール

1. **Apache Ant のダウンロード**
   ターミナルを開き、コマンド **sudo yum install ant-trax** を入力します。

2. **Apache Ant のインストール**
   インストーラーから入力を要求されたら、**Y** と入力します。

3. **ANT_HOME 環境変数の追加**

   a. **vi ~/.bash_profile**。

   b. 以下の行を追加します。

   ```
   export ANT_HOME=/FILEPATH/ant
   ```

   filepath は、Apache Ant がインストールされているディレクトリーです。vi を保存して終了します。

   例3.1

   ```
   export ANT_HOME=/opt/apache-ant-1.8.2
   ```

4. **Ant インストールの bin ディレクトリーを Path 環境変数に追加します。**

   a. **vi ~/.bash_profile**。

   b. 次の行を追加し、vi を保存して終了します。

   ```
   export PATH=$PATH:$ANT_HOME/bin
   ```

5. **インストールのテスト**
   ターミナルに戻り、**ant -version** を実行します。出力は以下のようになります。

   ```
   [localhost]$ ant -version
   Apache Ant(TM) version 1.8.2 compiled on July 6 2011
   ```

手順3.3 Microsoft Windows への Apache Ant のインストール

手順3.3 Microsoft Windows への Apache Ant のインストール

1. **Apache Ant のダウンロード**
   http://ant.apache.org/ から最新の Apache Ant バイナリーリリースをダウンロードします。

2. **Apache Ant の展開**
   ファイルを任意のインストール場所に展開します。以下に例を示します。

   - **c:\Program Files\Apache\Ant**

3. **ANT_HOME 環境変数の追加**

   a. **Start Menu** をクリックします。

   b. **Control Panel** を開きます。

   c. **System → Advanced → Environment Variables** を選択します。

   d. **ANT_HOME** という名前の新しい変数を作成します。

   e. Apache Ant ディレクトリーを指すように ANT_HOME 変数を設定します。

4. **Ant インストールの bin ディレクトリーを Path 環境変数に追加します。**

   a. **Start Menu** をクリックします。

   b. **Control Panel** を開きます。

   c. **System → Advanced → Environment Variables → System Variables** を選択します。

   d. **PATH** 変数を編集し、次のテキストを追加します。

   ```
   ;%ANT_HOME%\bin
   ```

5. **インストールのテスト**
   コマンドラインターミナルで **ant -version** を実行します。バージョン番号が表示されます。

バグの報告

パート II. 基本的なインストールおよび操作

第4章 製品のダウンロード

4.1. RED HAT カスタマーポータル

Red Hat カスタマーポータルは、https://access.redhat.com/home にある Web サイトです。サブスクリプションの管理および維持、Red Hat ナレッジベースへのアクセス、Red Hat およびパートナーとの連携を一箇所で行うことができます。

バグの報告

4.2. ダウンロード可能なパッケージ

表4.1 ダウンロード可能なパッケージ

パッケージ	Description
JBoss Enterprise SOA Platform パッケージ	SOA Platform パッケージは、完全な JBoss アプリケーションデプロイメント環境です。このインストールだけで、SOA アプリケーションをデプロイするための完全な環境が提供されます。これには、Seam、Hibernate、クラスタリング、およびトランザクションサービスが含まれます。
JBoss Enterprise SOA Platform スタンドアロン版パッケージ	SOA スタンドアロンパッケージは、コア SOA 機能のみが必要なデプロイメント向けの軽量ソリューションを提供します。クラスタリングには対応していません。
JBoss Enterprise SOA Platform ソースコードパッケージ	ソースコードパッケージには、JBoss Enterprise SOA Platform 製品の完全なソースコードが含まれています。
SOA Platform JavaDocs	JavaDocs パッケージには、JBoss Enterprise SOA Platform の API の完全な JavaDocs が含まれています。

バグの報告

4.3. JBOSS ENTERPRISE SOA PLATFORM のバージョン間の違い

表4.2 JBoss Enterprise SOA Platform のバージョン間の違い

	SOA Platform パッケージ	SOA スタンドアロンパッケージ
JBoss ESB	はい。	はい。

	SOA Platform パッケージ	SOA スタンドアロンパッケージ
JBoss Rules	はい。	はい。
JBoss JBPM	はい。	はい。
JBoss EAP	はい。	はい。
BPEL エンジン	はい。	はい。
EJB3	はい。	いいえ
JBoss RestEasy	はい。	いいえ
JBoss Seam	はい。	いいえ
JBoss Enterprise Data Services デプロイメントのサポート	はい。	いいえ

バグの報告

4.4. JAVADOCS

JavaDocs は、Java API の自動生成ドキュメントです。これらは、開発者がソースコードを記述するときに追加するコメントから作成されます。JavaDocs は、Java API を文書化するための事実上の標準的な方法になりました。

バグの報告

4.5. RED HAT カスタマーポータルからのファイルのダウンロード

タスクの前提条件

このタスクを開始する前に、カスタマーポータルのアカウントが必要です。https://access.redhat.com を参照し、右上隅にある **Register** リンクをクリックしてアカウントを作成します。

手順4.1 タスク:

1. https://access.redhat.com を参照し、右上隅にある **Log in** リンクをクリックします。認証情報を入力し、**Log In** をクリックします。

   結果:

   RHN にログインし、https://access.redhat.com のメイン Web ページに戻ります。

2. **Downloads ページに移動します。**
   次のいずれかの方法で **Downloads** ページに移動します。

- 上部のナビゲーションバーにある **Downloads** リンクをクリックします。
- https://access.redhat.com/downloads/ に直接移動します。

3. **ダウンロードする製品とバージョンを選択します。**
   以下の方法を使い、正しい製品とバージョンを選びダウンロードしてください。
   - ナビゲーションを使って 1 つずつ進めていきます。
   - 画面の右上端にある検索エリアを使い製品を検索します。
4. **お使いのオペレーティングシステムやインストール方法にあったファイルをダウンロードします。**
   選択した製品に応じて、特定のオペレーティングシステムおよびアーキテクチャー用の Zip アーカイブ、RPM、またはネイティブインストーラーを選択できます。ファイル名をクリックするか、ダウンロードするファイルの右側にある **Download** リンクをクリックします。

結果:

お使いのコンピューターにファイルをダウンロードします。

バグの報告

4.6. チェックサム検証

チェックサム検証は、ダウンロードしたファイルが破損していないことを確認するために使用されます。チェックサム検証では、デジタルデータの任意のブロックから固定サイズのデータ (チェックサム) を計算するアルゴリズムを採用しています。二者が同じアルゴリズムを使用して特定のファイルのチェックサムを計算すると、結果は同じになります。したがって、サプライヤーと同じアルゴリズムを使用してダウンロードしたファイルのチェックサムを計算する場合、チェックサムが一致すれば、ファイルの整合性が確認されます。不一致がある場合、ダウンロードプロセスでファイルが破損しています。

バグの報告

4.7. ダウンロードしたファイルの確認

手順4.2 ダウンロードしたファイルの確認

1. Red Hat カスタマーポータルからダウンロードしたファイルにエラーがないことを確認するには、ポータルサイトにまだある場合は、そのパッケージの **は** ページに移動します。ここには、ファイルの整合性をチェックするために使用する **MD5** および **SHA256** の "チェックサム" 値があります。
2. ターミナルウィンドウを開き、引数としてダウンロードした **ZIP** ファイル名を指定して、**md5sum** または **sha256sum** コマンドを実行します。プログラムはファイルのチェックサム値を出力します。
3. コマンドによって返されたチェックサム値を、ファイルの **Software Details** ページに表示されている対応する値と比較します。

注記

Microsoft Windows には、チェックサムツールが搭載されていません。Microsoft Windows ユーザーは、代わりにサードパーティー製品をダウンロードする必要があります。

結果

2 つのチェックサム値が同一の場合、ファイルは変更も破損もしていないため、安全に使用できます。

2 つのチェックサム値が同一でない場合は、ファイルを再度ダウンロードします。チェックサム値の違いは、ファイルがダウンロード中に破損したか、サーバーにアップロードされた後に変更されたことを意味します。何回かダウンロードしてもチェックサムが正常に検証されない場合は、Red Hat サポートにお問い合わせください。

バグの報告

4.8. RED HAT ドキュメントサイト

Red Hat の公式ドキュメントサイトは、https://access.redhat.com/knowledge/docs/ です。上記のサイトには、本書を含め、最新版の全ドキュメントが含まれています。

バグの報告

第5章 インストール

5.1. 変数名：SOA_ROOT ディレクトリー

SOA Root (SOA_ROOT として記述される) は、アプリケーションサーバーファイルが含まれるディレクトリーに指定された用語です。JBoss Enterprise SOA Platform パッケージの標準バージョンでは、SOA root は **jboss-soa-p-5** ディレクトリーです。ただし、スタンドアロン編集では、**jboss-soa-p-standalone-5** ディレクトリーになります。

ドキュメント全体で、このディレクトリーは頻繁に **SOA_ROOT** と呼ばれます。この名前がある場合は、必要に応じて **jboss-soa-p-5** または **jboss-soa-p-standalone-5** のいずれかを置き換えます。

バグの報告

5.2. 変数名: PROFILE

PROFILE は、JBoss Enterprise SOA Platform 製品に同梱されるサーバープロファイルの 1 つ (default、production、all、minimal、standard、または web) のいずれかになります。本書でファイルパスに PROFILE が表示されるたびに、使用しているこれらのいずれかを置き換えます。

バグの報告

5.3. RED HAT ENTERPRISE LINUX への JBOSS ENTERPRISE SOA PLATFORM のインストール

前提条件

- Java Development Kit (Red Hat は OpenJDK を推奨)
- データベースサーバー
- Apache Ant 1.7 以降
- ZIP ファイルを開くことができるアーカイブツール

警告

この手順では、テストシステムにのみ製品をインストールして設定する方法について説明します。これらのデフォルト設定では、実稼働システムに必要なセキュリティーレベルは得られません。この製品を実稼働環境で使用するためのインストールと設定については、完全なインストールガイドをお読みください。

Red Hat Customer Portal からスタンドアロンまたは完全な SOA パッケージをダウンロードし、その整合性を確認したら、以下の手順に従います。

手順5.1 インストール

1. **unzip soa-p-VERSION.zip** を実行して、インストールディレクトリーを展開します。
2. テキストエディターでユーザーアカウント設定ファイルを開きます (**vi SOA_ROOT/jboss-as/server/default/conf/props/soa-users.properties**)。

   このファイルの内容は、**username=password** という構文を使用しています。

   ```
   #admin=admin
   ```

   先頭のハッシュ文字が存在する場合は削除して、管理者アカウントのセキュリティーロールが有効になっていることを確認します。

   警告

   このアカウントは安全ではなく、パスワードは簡単に推測できるため、admin=admin はテスト目的でのみ使用してください。セキュリティーを損なう可能性があるため、**admin** を実稼働システムのパスワードとして使用しないでください。

3. ファイルを保存して vi を終了します。
4. テキストエディターでセキュリティーパーミッションファイルを開きます (**vi SOA_ROOT/jboss-as/server/default/conf/props/soa-roles.properties**)。

   ```
   #admin=JBossAdmin,HttpInvoker,user,admin
   ```

   先頭のハッシュ文字が存在する場合は削除して、管理者アカウントのセキュリティーロールが有効になっていることを確認します。

5. 変更をファイルに保存し、vi を終了します。

結果

JBoss Enterprise SOA Platform がインストールされ、基本的な使用のために設定されます。

バグの報告

5.4. MICROSOFT WINDOWS への JBOSS ENTERPRISE SOA PLATFORM のインストール

前提条件

- Java Development Kit
- データベースサーバー

- Apache Ant 1.7 以降
- ZIP ファイルを開くことができるアーカイブツール

警告

この手順では、テストシステムにのみ製品をインストールして設定する方法について説明します。これらのデフォルト設定では、実稼働システムに必要なセキュリティーレベルは得られません。この製品を実稼働環境で使用するためのインストールと設定については、完全なインストールガイドをお読みください。

Red Hat Customer Portal からスタンドアロンまたは完全な SOA パッケージをダウンロードし、その整合性を確認したら、以下の手順に従います。

手順5.2 インストール

1. 任意の ZIP ツールを使用して **soa-p-VERSION.zip** を展開します。
2. 先頭のハッシュ文字が存在する場合は削除して、管理者アカウントのセキュリティーロールが有効になっていることを確認します。

警告

このアカウントは安全ではなく、パスワードは簡単に推測できるため、admin=admin はテスト目的でのみ使用してください。セキュリティーを損なう可能性があるため、**admin** を実稼働システムのパスワードとして使用しないでください。

3. ファイルを保存し、メモ帳を終了します。
4. **SOA_ROOT\jboss-as\server\default\conf\props\soa-roles.properties** をメモ帳で開きます。

```
#admin=JBossAdmin,HttpInvoker,user,admin
```

ハッシュを削除して、管理者アカウントのセキュリティーアクセスパーミッションを有効にします。

5. ファイルを保存し、メモ帳を終了します。

結果

JBoss Enterprise SOA Platform がインストールされ、基本的な使用のために設定されます。

バグの報告

第6章 基本操作チュートリアル

6.1. JBOSS ENTERPRISE SOA PLATFORM の初回実行

はじめに

次のセクションでは、JBoss Enterprise SOA Platform を初めて起動して実行する方法を学習します。最も簡単な方法は、"Hello World" クイックスタートにあるデモコードを実行することです。

このクイックスタートを実行すると、その仕組みを把握できます。

バグの報告

6.2. JBOSS ENTERPRISE SOA PLATFORM の起動

前提条件

次のソフトウェアをインストールする必要があります。

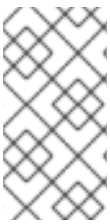
- JBoss Enterprise SOA Platform

手順6.1 JBoss Enterprise SOA Platform の起動

- ***サーバーウィンドウで*** **SOA サーバーを起動する**
  - Red Hat Enterprise Linux
    a. ターミナルを開き、コマンド **cd *SOA_ROOT*/jboss-as/bin** を入力して **bin** ディレクトリーに移動します。
    b. **./run.sh** と入力して SOA サーバーを起動します。(サーバープロファイルを指定していないため、デフォルトが使用されます。)
  - Microsoft Windows
    a. ターミナルを開き、コマンド **chdir *SOA_ROOT*\jboss-as\bin** を入力して **bin** ディレクトリーに移動します。
    b. **run.bat** と入力して SOA サーバーを起動します。(サーバープロファイルを指定していないため、デフォルトが使用されます。)

結果

サーバーが起動します。ハードウェアの速度にもよりますが、これには約 2 分かかります。



注記

エラーがないことを確認するには、サーバーログをチェックします (**less *SOA_ROOT*/jboss-as/server/*PROFILE*/log/server.log**)。別のチェックとして、Web ブラウザーを開いて、http://localhost:8080 に移動します。設定したユーザー名とパスワードで管理コンソールにログインできることを確認してください。

バグの報告

6.3. 起動プロセスのトラブルシューティング

server.log のエラーは、キーワード "ERROR" で示されます。ログにエラーが表示された場合は、次のリストを調べて原因を見つけてください。

1. "アドレスはすでに使用されています" - ポート 8080 で実行されているサーバーがすでに存在します。
2. "Java が見つかりません" - Java JRE がインストールされていない可能性があります。インストールされている場合は、Java ランタイムを見つけるように PATH 環境変数が設定されていません。
3. "クラスが見つかりません" - CLASSPATH 環境変数が正しく設定されていません。この変数は、サーバーの起動スクリプトによって設定されるため、設定する必要はありません。
4. これらのエラーのいずれかが表示された場合は、エラーメッセージの前後に表示される server.log メッセージを調べて、エラーの根本原因に関する追加情報を確認してください。

バグの報告

6.4. "HELLO WORLD" クイックスタートの実行

6.4.1. Quickstart

クイックスタートはサンプルプロジェクトです。それぞれが、サービスの構築を支援するために特定の機能を使用する方法を示しています。**SOA_ROOT/jboss-as/samples/quickstarts/** ディレクトリーには、数十のクイックスタートが含まれています。**Apache Ant** を使用して、すべてのクイックスタートをビルドしてデプロイします。

バグの報告

6.4.2. クイックスタートに関する重要事項

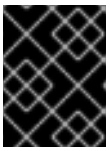
クイックスタートを実行する場合は、次の点に注意してください。

1. 各クイックスタートは、Apache Ant を使用してビルドおよびデプロイする必要があります。
2. 各クイックスタートは、**samples/quickstarts/conf/quickstarts.properties** ファイルを使用して、サーバーがインストールされたディレクトリーなどの環境固有の設定オプションを保存します。サーバーのインストールに一致する **quickstarts.properties** ファイルを作成する必要があります。プロパティーファイルの例 (**quickstarts.properties-example**) が含まれています。
3. クイックスタートごとに要件が異なります。これらは、個別の **readme.txt** ファイルに記載されています。
4. すべてのクイックスタートをすべてのサーバープロファイルで実行できるわけではありません。
5. jBPM クイックスタートには、有効な jBPM コンソールのユーザー名とパスワードが必要です。これらを **SOA_ROOT/jboss-as/samples/quickstarts/conf/quickstarts.properties** ファイルにプロパティーとして追加して提供します。

```
# jBPM console security credentials
jbpm.console.username=admin
jbpm.console.password=adminpassword
```

この要件の影響を受けるクイックスタートは、**bpm_orchestration1**、**bpm_orchestration2**、**bpm_orchestration3**、および **bpm_orchestration4** です。

6. サーバーが *ヘッドレス* モードで実行されていない場合は、一部のクイックスタート (**groovy_gateway** など) のみを実行できます。(JBoss Enterprise SOA Platform はデフォルトでヘッドレスモードで起動するように設定されています。)



重要

Red Hat は、実稼働サーバーをヘッドレスモードでのみ実行することをお勧めします。

バグの報告

6.4.3. テストサーバーへの "Hello World" クイックスタートのデプロイ

前提条件

- **SOA_ROOT/jboss-as/samples/quickstarts/conf/quickstarts.properties-example** の設定がサーバー設定 (テスト環境の **default**) と一致することを確認します。

手順6.2 "Hello World" クイックスタートのデプロイ

1. サーバーが完全に起動したことを確認します。
2. 2 番目のターミナルウィンドウを開き、クイックスタートを含むディレクトリーに移動します。**cd SOA_ROOT/jboss-as/samples/quickstarts/helloworld** (**chdir SOA_ROOT\jboss-as\samples\quickstarts\helloworld**)
3. **ant deploy** を実行して、クイックスタートをデプロイします。次のようなメッセージを探して、デプロイが成功したかどうかを確認します。

```
deploy-esb:
    [copy] Copying 1 file to
/jboss/local/53_DEV2/jboss-soa-p-5/jboss-as/server/default/deploy

deploy-exploded-esb:

quickstart-specific-deploys:
    [echo] No Quickstart specific deployments being made.

display-instructions:
    [echo]
    [echo] ******************
    [echo] Quickstart deployed to target JBoss ESB/App Server at
'/jboss/local/53_DEV2/jboss-soa-p-5/jboss-as/server/default/deploy'.
    [echo] 1.  Check your ESB Server console to make sure the deployment was
executed without errors.
```

```
    [echo] 2.  Run 'ant runtest' to run the Quickstart.
    [echo] 3.  Check your ESB Server console again.  The Quickstart should
have produced some output.
    [echo] ******************

deploy:

BUILD SUCCESSFUL
```

また、**SOA_ROOT/jboss-as/server/default/log/server.log** でこれを確認してください。

```
10:58:52,998 INFO  [NamingHelper] JNDI InitialContext properties:{}
11:00:58,154 INFO  [QueueService]
Queue[/queue/quickstart_helloworld_Request_esb] started, fullSize=200000,
pageSize=2000, downCacheSize=2000
11:00:58,186 INFO  [QueueService]
Queue[/queue/quickstart_helloworld_Request_gw] started, fullSize=200000,
pageSize=2000, downCacheSize=2000
11:00:58,427 INFO  [EsbDeployment] Starting ESB Deployment
'Quickstart_helloworld.esb'
```

4. コマンド **ant runtest** を発行して、クイックスタートを実行します。クイックスタートが実行されると、次のようなメッセージが **SOA_ROOT/jboss-as/server/default/log/server.log** に書き込まれます。

```
11:03:02,190 INFO  [STDOUT]
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
11:03:02,191 INFO  [STDOUT] Body: Hello World
11:03:02,191 INFO  [STDOUT]
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
11:03:02,192 INFO  [STDOUT] Message structure:
11:03:02,192 INFO  [STDOUT] [Hello World].
```

結果

サーバーターミナルに "Hello World" という言葉が表示されます。このメッセージは **SOA_ROOT/jboss-as/server/default/log/server.log** ファイルにも追加されます。

バグの報告

6.4.4. ant deploy

ant deploy は **build** ディレクトリーにあるクイックスタートのソースコードをコンパイルし、サーバープロファイルの **deploy** ディレクトリーに .ESB ファイル (**Quickstart_helloworld.esb** など) を生成します。(BPEL クイックスタート用の .JAR ファイルが生成されることに注意してください。) サーバーは新しい .esb アーカイブの存在を検出し、デプロイします。.ESB アーカイブには、**ant deploy** がキューを設定するために使用する deployment.xml ファイルがあります。

また、**ant deploy** は、XSL テンプレートを使用して、汎用 JMS キュー名をターゲットサーバーのメッセージングプロバイダーが必要とする特定の JMS キューに変換します。Ant は、メッセージングプロバイダーのデプロイメントのサーバーを調べて、正しいメッセージングプロバイダーを選択します。ビルドスクリプトによって検出されるのは、JBoss Messaging、JBoss MQ、および HornetQ のみです。

他のメッセージングプロバイダーは、クイックスタートではサポートされていません。次に、Ant は **deployment.xml** ファイルを **build/META-INF** ディレクトリーに配置してから、他のクイックスタートと同じ .ESB アーカイブに含めます。

バグの報告

6.4.5. ant runtest

ant runtest は、ESB 非対応の "Hello World" メッセージ (プレーンな String オブジェクト) を JMS キュー (**queue/quickstart_helloworld_Request_gw**) に送信します。このコマンドは、Java に送信側クラスを実行するように指示します ("Hello World" クイックスタートの場合、これは **org.jboss.soa.esb.samples.quickstart.helloworld.test.sendJMSMessage** と呼ばれます)。そうすることで、デプロイされたプロセスにメッセージを直接送信します。

バグの報告

6.4.6. ant sendesb

ant sendesb コマンドは、ESB メッセージを SOA サーバーに送信します。このコマンドは、ESB 対応メッセージを ESB リスナーに直接送信します。つまり、ゲートウェイを利用する必要はありません。

バグの報告

6.4.7. "Hello World" クイックスタートのアンデプロイ

手順6.3 タスク

1. クイックスタートのディレクトリーに移動します: **cd SOA_ROOT/jboss-as/samples/quickstarts/helloworld** (または、Microsoft Windows を実行している場合は **chdir SOA_ROOT\jboss-as\samples\quickstarts\helloworld**)。
2. **ant undeploy** コマンドを実行します。次のようなメッセージが表示されます。

```
undeploy:
[delete] Deleting:
/jboss/local/53_DEV2/jboss-soa-p-5/jboss-as/server/default/deploy/Quickstart_helloworld.esb

BUILD SUCCESSFUL
```

そして、次のようなメッセージが server.log に書き込まれます。

```
11:10:08,205 INFO  [EsbDeployment] Stopping 'Quickstart_helloworld.esb'
11:10:08,577 INFO  [EsbDeployment] Destroying 'Quickstart_helloworld.esb'
```

バグの報告

6.5. JBOSS ENTERPRISE SOA PLATFORM サーバーの停止

手順6.4 JBoss Enterprise SOA Platform サーバーの停止

- **SOA サーバーを停止する**
  server window (SOA サーバーが開始された端末ウィンドウ) で **ctrl-c** を押します。

結果

サーバーがシャットダウンします。このプロセスには数分かかることに注意してください。**server.log** ファイルで次の行を探して、サーバーが正常にシャットダウンしたことを確認します。

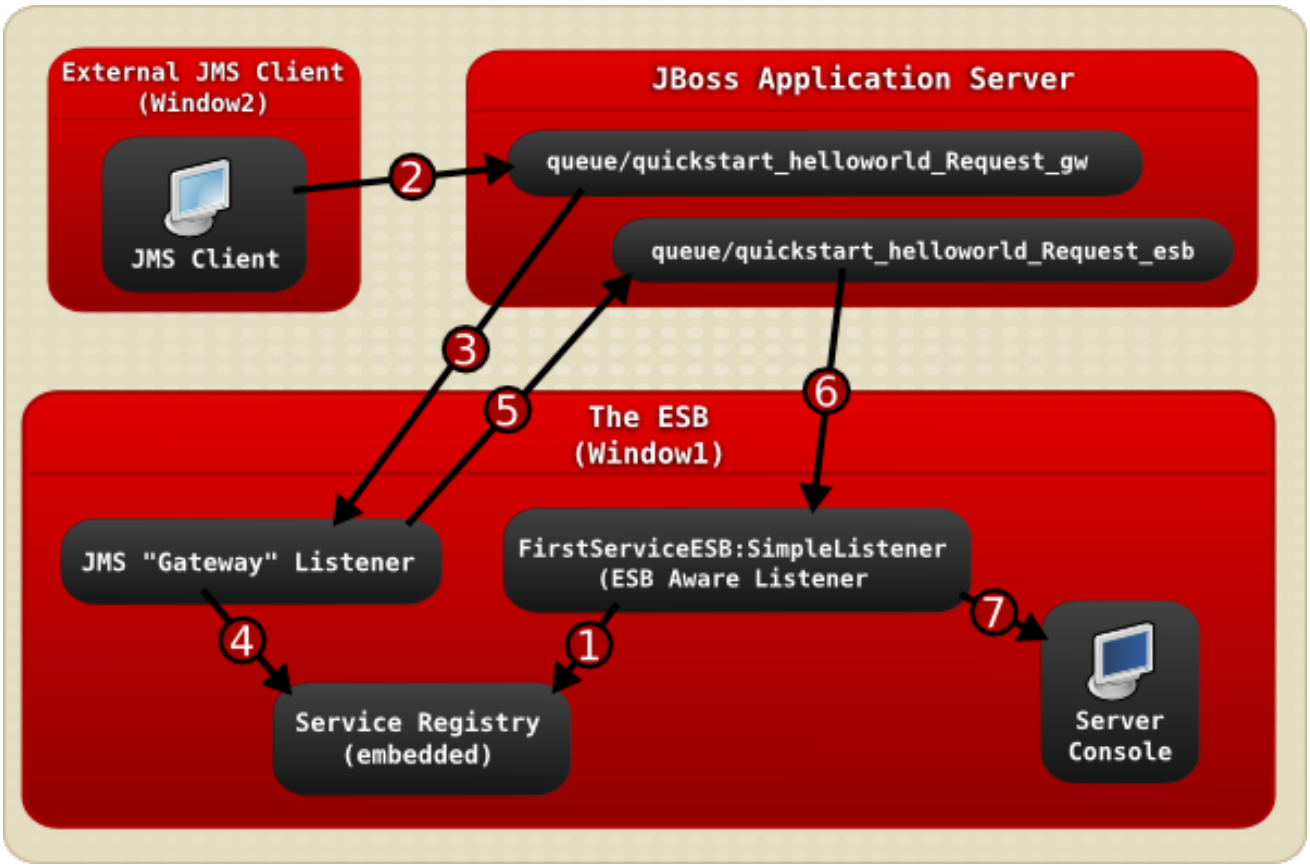
```
12:17:02,786 INFO  [ServerImpl] Shutdown complete
```

バグの報告

6.6. "HELLO WORLD" クイックスタートの検証

6.6.1. "Hello World" クイックスタートの仕組みの概要

図6.1イメージ



1. **JBoss Enterprise SOA Platform** サーバーが **Window1** で起動され、helloworld クイックスタートがデプロイされると **FirstServiceESB:SimpleListener** サービスが Service Registry サービスに追加されます。

2. JMS クライアントは、ESB 非対応の "Hello World" メッセージ (プレーンな **String** オブジェクト) を JMS キュー (**queue/quickstart_helloworld_Request_gw**) に送信します。

3. JMS ゲートウェイリスナーは、ESB 非認識メッセージを受信し、そこから ESB 認識エンドポイントで使用する ESB 認識メッセージを作成します。

4. **JMS ゲートウェイリスナー** は、**service registry** を使用して、**FirstServiceESB:SimpleListener** サービスの *end-point reference* (EPR) を見つけます。この場合、EPR は **queue/quickstart_helloworld_Request_esb** JMS キューです。
5. **JMS ゲートウェイリスナー** は、新しい ESB 対応メッセージを受け取り、それを **queue/quickstart_helloworld_Request_esb** JMS キューに送信します。
6. **FirstServiceESB:SimpleListener** サービスがメッセージを受信します。
7. **FirstServiceESB:SimpleListener** サービスはメッセージからペイロードを抽出し、コンソールに出力します。

バグの報告

6.6.2. ESB メッセージ

このメッセージは、 **org.jboss.soa.esb.message** インターフェイスによって定義された形式を取得するメッセージです。この標準化された形式は、ヘッダー、本文 (ペイロード)、および添付ファイルで設定されます。すべての ESB 対応のクライアントとサービスは、メッセージを使用して相互に通信します。

バグの報告

6.6.3. ESB メッセージのコンポーネント

ESB メッセージは、次のコンポーネントで設定されています。

Header

ヘッダーには、宛先エンドポイント参照、送信者エンドポイント参照、および応答先などの情報が含まれています。これはすべて、一般的なメッセージレベルの機能情報です。

コンテキスト

これは、メッセージをさらに説明する追加情報です。たとえば、トランザクションまたはセキュリティーデータ、最終的な受信者の ID、HTTP クッキー情報などです。

本文

メッセージの実際の内容。

異常

メッセージに関連付けられたエラー情報。

添付ファイル

メッセージに関連付けられた添付ファイル (追加ファイル)。

プロパティー

メッセージ固有のプロパティー (たとえば、 jbossesb.message.idプロパティーは、メッセージごとに一意の値を指定します)。

コード表現は次のとおりです。

```
    <xs:complexType name="Envelope">
  <xs:attribute ref="Header" use="required"/>
  <xs:attribute ref="Context" use="required"/>
  <xs:attribute ref="Body" use="required"/>
  <xs:attribute ref="Attachment" use="optional"/>
  <xs:attribute ref="Properties" use="optional"/>
  <xs:attribute ref="Fault" use="optional"/>
 </xs:complexType>
```

バグの報告

6.6.4. メッセージオブジェクトをキューに送信する方法

概要

JBoss Enterprise SOA Platform 製品は、ローカルサーバー上の JNDI の存在を識別するためにパラメーターが設定されたプロパティーオブジェクトを使用します。次に、これは、新しいネーミングコンテキストを作成するための呼び出しのパラメーターとして使用されます。このネーミングコンテキストを使用して、ConnectionFactory を取得します。次に、ConnectionFactory が QueueConnection を作成し、これが QueueSession を作成します。この QueueSession は、Queue の Sender オブジェクトを作成します。Sender オブジェクトは、送信者の ObjectMessage を作成し、キューの送信に使用されます。

バグの報告

6.6.5. Properties オブジェクト

Properties オブジェクトは、キーと値の汎用的なリストを格納するコンテナーです。Properties オブジェクトに、ローカルサーバーで JBoss JNDI を識別するために必要なパラメーターを入力し、それを新しい Naming Context を作成する呼び出しのパラメーターとして使用する必要があります。

バグの報告

6.6.6. Naming Context

Naming Context は、ディレクトリーに接続し、ユーザーが名前参照によってオブジェクトを取得できるようにするオブジェクトです。

バグの報告

6.6.7. ConnectionFactory

ConnectionFactory は、QueueConnection を作成するオブジェクト (**org.jboss.jms.client.JBossConnectionFactory**) です。Naming Contect は、JNDI から ConnectionFactory を取得します。

バグの報告

6.6.8. QueueConnection

QueueConnection は、ConnectionFactory によって作成されるオブジェクトです。QueueSession の作成と開始に使用されます。

バグの報告

6.6.9. QueueSession

QueueSession は QueueConnection によって作成されます。キューの Sender オブジェクトと、文字列を含む ObjectMessage を作成します。

バグの報告

6.6.10. SOA_ROOT/jboss-as/samples/quickstarts/helloworld/build.xml

build.xml ファイルには、**build** ディレクトリーにあるクイックスタートのソースコードをコンパイルするために **ant deploy** が使用する命令が含まれています。このファイルを編集して、独自のカスタム命令を追加できます。

バグの報告

6.6.11. SOA_ROOT/jboss-as/samples/quickstarts/helloworld/deployment.xml

deployment.xml ファイルは、メッセージングキューを作成および設定するために **ant runtest** によって使用されます。

ant deploy は、コンパイルプロセス中に **build/META-INF** ディレクトリーに **deployment.xml** ファイルを生成します。そして、ハードコーディングされた JMS キューのどれを使用する必要があるかを決定するときに、そのファイルを取り込みます。取り込まれると、ファイルは .ESB アーカイブの一部としてパッケージ化されます。(Ant は、XSL テンプレートを使用して、汎用 JMS キュー名を、ターゲットサーバーのメッセージングプロバイダーが必要とする特定の JMS キューに変換します。このテンプレートから **deployment.xml** ファイルが作成されます。)

バグの報告

6.6.12. メッセージングキュー

メッセージキューは、アプリケーションのデプロイ時に生成されるキューです。メッセージは、メッセージリスナーを待機するこれらのキューに送信されます。

バグの報告

6.6.13. メッセージリスナー

メッセージリスナーは、SB 対応メッセージの受信に必要な通信エンドポイントをカプセル化します。リスナーはサービスによって定義され、そのロールはキューを監視します。これらのキューに到着すると、メッセージを受信します。リスナーがメッセージを受信すると、ESB サーバーはアクション定義で

指定された適切なアクションクラスを呼び出します。このクラスのメソッドはメッセージを処理します。つまり、リスナーは受信ルーターとして機能し、メッセージをアクションパイプラインに送信します。パイプラインのアクションによってメッセージが変更されると、リスナーは結果を replyTo エンドポイントに送信します。

リスナーのさまざまなパラメーターを設定できます。たとえば、アクティブなワーカースレッドの数を設定できます。

リスナーには、ESB 対応リスナーとゲートウェイリスナーの 2 種類があります。ゲートウェイリスナーは、さまざまな形式のデータ (ファイル内のオブジェクト、SQL テーブル、JMS メッセージなど) を受け入れるという点で、ESB 対応のリスナーとは異なります。次に、これらの形式から ESB メッセージング形式に変換します。対照的に、ESB 対応のリスナーは、**org.jboss.soa.esb.message.Message** 形式のメッセージのみを受け入れることができます。各ゲートウェイリスナーには、対応する ESB リスナーが定義されている必要があります。

ESB 対応のリスナーでは、RuntimeExceptions がロールバックをトリガーできます。対照的に、ゲートウェイリスナーでは、トランザクションは単純にメッセージを JBoss ESB に送信します。その後、メッセージは非同期的に処理されます。このようにして、メッセージの失敗はメッセージの受信から分離されます。

バグの報告

6.6.14. ESB 対応

アプリケーションクライアントとサービスが ESB 対応である場合には、SOA プラットフォームの Enterprise Service Bus で使用されるメッセージ形式とトランスポートプロトコルを理解できます。

バグの報告

6.6.15. ゲートウェイリスナー

ゲートウェイリスナーは、ESB 認識の世界と ESB 非認識の世界を橋渡しするために使用されます。これは、外部 (ESB 非対応) エンドポイント経由で到着した ESB に対応していないメッセージのキューをリッスンするように設計された特殊なリスナープロセスです。メッセージがキューに到着すると、ゲートウェイリスナーがメッセージを受信します。ゲートウェイリスナーが受信データの到着を認識すると、そのデータ (ESB に対応していないメッセージ) を **org.jboss.soa.esb.message.Message** 形式に変換します。この変換は、ゲートウェイの種類に応じてさまざまな方法で行われます。変換が行われると、ゲートウェイリスナーはデータを正しい宛先にルーティングします。

バグの報告

6.6.16. 送信者

送信者は QueueSessions によって作成されます。キューごとに送信者があります。送信者の **send** メソッドは、**ant runtest** の実行時に QueueSession の ObjectMessage によって呼び出されます。これが発生すると、クライアントはメッセージをキューに送信します。

バグの報告

6.6.17. クイックスタートの詳細

特定のクイックスタートについて詳しく知るには:

手順6.5 タスク

1. クイックスタートの **readme.txt** ファイルを調べてください。
2. クイックスタートのディレクトリーで **ant help** コマンドを実行します。

バグの報告

6.7. "HELLO WORLD" クイックスタートのソースコード

6.7.1. SOA_ROOT/jboss-as/samples/quickstarts/helloworld/src

"Hello World" クイックスタートの **src** ディレクトリーには、コンパイルされていないプログラミング命令が含まれています。クラスは、サブディレクトリーにネストされたファイルにあります。ant deploy は、このソースコードをコンパイルします。

バグの報告

6.7.2. SOA_ROOT/jboss-as/samples/quickstarts/helloworld/lib

lib ディレクトリーには、(ソースコードに加えて) **ant deploy** がクイックスタートをコンパイルするために必要なクラスが含まれています。

バグの報告

6.7.3. SOA_ROOT/jboss-as/server/SERVER_PROFILE/deploy

ant deploy は、クイックスタートのコンパイル済みバージョン (.ESB アーカイブファイルの形式) を **build** ディレクトリーから **/jboss-as/server/default/deploy/** ディレクトリーに移動します。JBoss Enterprise SOA Server はこのディレクトリーをポーリングしており、新しい .ESB ファイルの存在を検出すると、それをデプロイします。

バグの報告

パート III. アカウントの管理

第7章 ユーザーアカウントの設定

7.1. ユーザーアカウント

ユーザーがログインして JBoss Enterprise SOA Platform のさまざまな Web ベースのコンソールを使用するには、アカウントが必要です。デフォルトのセキュリティーシステムは、プレーンテキストファイル (つまり、**soa-users.properties** および **soa-roles.properties**) を読み取って、ユーザーのパスワードをチェックし、アクセスレベルを決定します。SOA は、Java Authentication and Authorization Service (JAAS) を使用してユーザーアカウントを認証します。

警告

セキュリティーが損なわれるため、Red Hat はクリアテキストファイルのユーザーパスワードで設定された実稼働サーバーを実行することをお勧めしません。

バグの報告

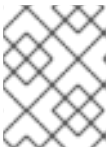
7.2. ユーザーアカウントの作成

手順7.1 新しいユーザーを追加

1. テキストエディターで **soa-users.properties** ファイルを開きます: **vi SOA_ROOT/jboss-as/server/PROFILE/conf/props/soa-users.properties**次の構文: **username=password** を使用して、ユーザーの名前とパスワードを新しい行に追加します。

   ログイン名が Harold のユーザーの例を次に示します。

   ```
   harold=@dm1nU53r
   ```

   注記

   このファイル内のハッシュ (#) で始まる行はすべて無視されます。(この規則を使用して、ユーザーアカウントを一時的に無効にすることができます。)

2. 変更をファイルに保存し、テキストエディターを終了します。

3. テキストエディターで **soa-roles.properties** ファイルを開きます: **vi SOA_ROOT/jboss-as/server/PROFILE/conf/props/soa-roles.properties**次の構文: **username=role1,role2,role3** を使用して、ユーザーとユーザーに割り当てるロールを新しい行に追加します。

   ```
   harold=JBossAdmin,HttpInvoker,user,admin
   ```

注記

ロールはいくつでも割り当てることができます。サーバーコンソールにログインできるようにするには、ユーザーに **JBossAdmin**、**HttpInvoker**、**user** および **admin** ロールを割り当てる必要があることに注意してください。

このファイル内のハッシュ (#) で始まる行はすべて無視されます。この規則を使用して、ユーザーロールを一時的に無効にすることができます。

4. 変更をファイルに保存し、テキストエディターを終了します。

結果

ユーザーは、http://localhost:8080 でサーバーコンソールにログインできるようになります。サーバーを再起動する必要はありません。

バグの報告

7.3. セキュリティーロール

表7.1 JBoss Enterprise SOA Platform コンソールユーザーのセキュリティー権限のリスト

Role	Description
JBossAdmin	SOA のさまざまな管理コンポーネントにログインするには、JBossAdmin ロールが必要です。これは主要なロールであるため、すべてのシステム管理者にこのロールを割り当てる必要があります。
HttpInvoker	HttpInvoker ロールは、遠隔地から JNDI および EJB にアクセスするために Http Invoker によって使用されます。
user	これは、SOA にデプロイされたサービスが JAAS セキュリティードメインを利用するように設定されている場合に、サービスへのユーザーアクセスを許可するために使用されます。jBPM コンソールは、この 1 つのロールのみに依存します。
admin	これは、SOA にデプロイされたサービスが JAAS セキュリティードメインを利用するように設定されている場合に、サービスへの管理アクセスを許可するために使用されます。

バグの報告

7.4. JAVA 認証および承認サービス (JAAS)

JAAS 1.0 API は、ユーザーの認証と承認用に設計された一連の Java パッケージで設定されています。この API は、標準の Pluggable Authentication Modules (PAM) フレームワークの Java バージョンを実

装し、Java 2 プラットフォームのアクセス制御アーキテクチャーを拡張して、ユーザーベースの承認をサポートします。

JAAS は、JDK 1.3 の拡張パッケージとして最初にリリースされ、JDK 1.6 にバンドルされています。

バグの報告

7.5. ユーザーのアカウントを無効にする

手順7.2 ユーザーのアカウントを無効にする

1. テキストエディターで **soa-users.properties** ファイルを開きます: **vi SOA_ROOT/jboss-as/server/PROFILE/conf/props/soa-users.properties**ユーザー名とパスワードを含む行全体を削除するか、その前にハッシュ (#) を付けてコメントアウトします。

   ログイン名が Harold のユーザーの例を次に示します。

   ```
   #harold=@dm1nU53r
   ```

2. 変更をファイルに保存し、テキストエディターを終了します。

結果

ユーザーはサーバーコンソールにログインできなくなります。サーバーを再起動する必要はありません。

バグの報告

パート IV. 高度な設定オプション

第8章 デフォルトデータベースの設定

8.1. HYPERSONIC データベース

Hypersonic は、JBoss Enterprise SOA Platform 製品に同梱され、追加設定なしで機能するデータベースです。これはデモデータベースのみで、実稼働システムでの使用には適していません。開発および評価の目的でのみ使用してください。Red Hat は、実稼働システムでの Hypersonic の使用をサポートしていません。

バグの報告

8.2. HYPERSONIC はサポートされていません

Hypersonic データベースはテスト目的のみに含まれており、サポートされていません。

Hypersonic Database の既知の問題は次のとおりです。

1. トランザクション分離スレッドとソケットリークはありません(connection.close （)がリソースをトレードアップしません）
2. 永続性の品質（通常、失敗後にログが破損し、自動リカバリーが阻止される）
3. 負荷がかかった状態でデータベースが破損する（データが多すぎるとデータベースプロセスが停止する）
4. クラスター環境では実行可能ではない

バグの報告

8.3. H2 DATABASE

H2 は、JBoss Enterprise SOA Platform 製品に同梱され、追加設定なしで機能するインメモリー参照データベースです。テスト環境でのみ使用してください。これは実稼働システムには適していませんが、Red Hat ではこのような環境でのその使用をサポートしていません。

バグの報告

8.4. データベース設定ツール

データベース設定ツール(**SOA_ROOT/jboss-as/tools/schema/** ディレクトリーにあります)は Apache Ant スクリプトです。JBoss Enterprise SOA Platform によって使用されるデータベースを設定します。サポートされるデータベースの一覧は、を参照して http://www.jboss.com/products/platforms/soa/supportedconfigurations/ ください。

バグの報告

8.5. 実稼働システムで使用するデータベースの設定

前提条件

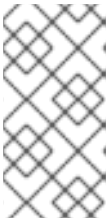
- Apache Ant
- 使用するデータベースがすでに存在している必要があります。
- そのデータベースに変更を加えるパーミッションを持つユーザーがすでに存在している必要があります。
- データベースの JDBC ドライバー JAR ファイルはサーバー設定の **lib/** ディレクトリーにある必要があります。

JBoss Enterprise SOA Platform は、データベースを使用してレジストリーサービスとメッセージストアを永続化します。デフォルトのデータベースは Hypersonic ですが、これは実稼働システムには適さないため、サポートされていません。実稼働環境で JBoss Enterprise SOA Platform を実行する前に、サポートされているデータベースに切り替える必要があります。

警告

Database Configuration Tool のみを使用して、データベース設定を 1 度変更できます。また、他の変更を加える前に実行する必要があります。すでに設定されているインストールで スクリプトを実行しようとすると、意図したとおりに機能しない可能性があります。

1. **サーバープロファイルのバックアップ**
   Database Configuration Tool が設定を変更する際にデータベースを設定する予定のサーバープロファイルのコピーを作成します。**cp -R SOA_ROOT/jboss-as/server/Profile /path/to/backup/folder**
2. **データベース設定ツールの実行**
   Database Configuration スクリプトを含むディレクトリーに移動します(**cd SOA_ROOT/jboss-as/tools/schema**)。
3. **Ant の実行**
   ant コマンドを実行してスクリプトを起動します。
4. **データを入力**
   プロンプトに従い、要求に応じて以下の情報を入力します。
   - 使用されているデータベースのタイプ。
   - データベースの名前
   - データベースのホスト名または IP アドレス
   - データベースに使用される TCP ポート
   - データベースへのアクセスに必要なユーザー名
   - このユーザーアカウントのパスワード

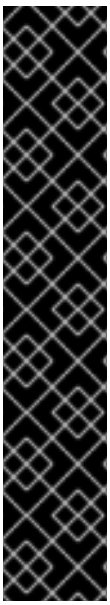


注記

スクリプトを実行する前に、これらの値を直接 **build.properties** ファイル（同じディレクトリーにある）に直接追加することもできます。Database Schema Tool がすでに ファイルに追加されている場合は、これらのプロパティーの入力を求めるプロンプトは表示されません。

結果

ツールは、関連する設定ファイルを更新し、終了します。



重要

このプロセスの実行時に jBPM 5 でデッドロックが生じる場合があります。これを回避するには、以下のコードを **to jbpm.esb/jbpm-ds.xml** に追加します。

```
<datasources>
   <local-tx-datasource>
      <jndi-name>BPELDB</jndi-name>
      ...
      <transaction-isolation>TRANSACTION_READ_COMMITTED</transaction-
isolation>
      <new-connection-sql>SET TRANSACTION ISOLATION LEVEL SNAPSHOT;
</new-connection-sql>
      ...
   </local-tx-datasource>
</datasources>
```

バグの報告

8.6. SOA_ROOT/JBOSS-AS/TOOLS/SCHEMA/BUILD.XML

SOA_ROOT/jboss-as/tools/schema/build.xml には、Database Schema Tool の設定が含まれています。これは、データベース設定オプションを介して実行するために **ant** コマンドによって使用されます。

バグの報告

8.7. SOA_ROOT/JBOSS-AS/TOOLS/SCHEMA/BUILD.PROPERTIES

これは、Database Schema Tool が設定を保存するファイルです。テキストエディターを使用して手動で編集することもできます。

バグの報告

第9章 SERVICE REGISTRY の設定

9.1. SERVICE REGISTRY

サービスレジストリーは、サービスに関する情報 (特にエンドポイントの参照) を格納する中央データベースです。JBoss Enterprise SOA Platform のデフォルトのサービスレジストリーは、jUDDI (Java Universal Description、Discovery、および Integration) です。ほとんどのサービスレジストリーは、Universal Description、Discovery and Integration (UDDI) 仕様に準拠するように設計されています。

ビジネスアナリストの観点からは、レジストリーはインターネット検索エンジンと似ていますが、Web ページではなく Web サービスを検索するように設計されています。開発者の観点からは、レジストリーはさまざまな条件に一致するサービスを検出し、公開するために使用されます。

多くの方法では、レジストリーサービスは JBoss Enterprise SOA Platform の最後とみなすことができます。サービスは、レジストリーがアクティブになったときにレジストリーへのエンドポイント参照をセルフパブリッシュし、サービスが不足したときにそれらを削除できます。コンシューマーはレジストリーを参照して、現在のサービスタスクにどのエンドポイントの参照が必要かを判断できます。

バグの報告

9.2. JUDDI レジストリー

JUDDI (Java Universal Description、Discovery and Integration) レジストリーは、JBoss Enterprise SOA Platform のコアコンポーネントです。製品のデフォルトサービスレジストリーであり、製品の一部として含まれています。これには、Enterprise Service Bus に接続されるすべてのサービスのアドレス (エンドポイント参照) が保存されます。JAXR に実装され、UDDI 仕様に準拠していました。

バグの報告

9.3. JUDDI および JBOSS ENTERPRISE SOA PLATFORM

juddi および JBoss Enterprise SOA Platform

JBoss Enterprise SOA Platform 製品には、jUDDI レジストリーの事前設定されたインストールが含まれています。特定の API を使用して、カスタムクライアントを介してこのレジストリーにアクセスできます。ただし、ビルドするカスタムクライアントは SOA Platform のサポート契約では対応されません。jUDDI の例、ドキュメント、および API の完全なセットには、次の場所からアクセスできます。http://juddi.apache.org/

バグの報告

9.4. サポートされるその他のサービスレジストリー

JBoss Enterprise SOA Platform は、以下の他の UDDI レジストリーもサポートします。

- SOA ソフトウェア SMS
- HP Systinet

バグの報告

9.5. UNIVERSAL DESCRIPTION、DISCOVERY AND INTEGRATION (UDDI) レジストリー

Universal Description、Discovery and Integration Registry (UDDI) は Web サービスのディレクトリーです。これを使用して、設計時または実行時にクエリーを実行してサービスを見つけます。UDDI レジストリー内では、情報はページで分類されます。UDDI は、企業やアプリケーションがインターネット上で Web サービスを動的に見つけて使用できるように、標準的な相互運用可能なプラットフォームを作成します。また、UDDI を使用すると、さまざまなコンテキストで運用レジストリーを目的別に維持することもできます。

UDDI により、プロバイダーはサービスの説明を公開することもできます。通常の UDDI レジストリーには、Web サービスの WSDL ドキュメントとサービスプロバイダーの連絡先情報の両方を参照する統一されたリソースロケーター (URL) が含まれます。

ビジネスはサービスを UDDI レジストリーに公開します。クライアントはレジストリーでサービスを検索し、サービスバインディング情報を受信します。その後、クライアントはバインディング情報を使用してサービスを呼び出します。UDDI API は、相互運用性のために SOAP ベースです。

バグの報告

9.6. UDDI ページタイプ

緑色のページ

緑のページでは、クライアントを提供されているサービスにバインドできる情報を提供します。

黄色のページ

黄色のページは、所属する業界に基づいて企業を分類するために使用されます。

ホワイトページ

ホワイトページには、サービスを提供する会社の名前、住所、その他の連絡先情報などの一般情報が含まれます。

バグの報告

9.7. SERVICE REGISTRY および JBOSS ENTERPRISE SOA PLATFORM

Service Registry は、JBoss Enterprise SOA Platform の主要な部分です。SOA Platform にサービスをデプロイする場合、それらのエンドポイントの参照はそれに保存されます。

バグの報告

9.8. JUDDI および ESB

JBoss Enterprise Service Bus は、レジストリーインターフェイス (Apache Scout を使用する のデフォルトバージョンの) を介してレジストリーとのすべての対話を指示します。

バグの報告

9.9. レジストリーの仕組み

1. JBoss Enterprise Service Bus は、レジストリーインターフェイスを介したレジストリーとのすべての対話をフェデレーションします。
2. その後、このインターフェイスの JAXR 実装を呼び出します。
3. JAXR API は JAXR 実装を使用する必要があります。(デフォルトでは Apache Scout です。)
4. Apache Scout は、次に レジストリー を呼び出します。

バグの報告

9.10. APACHE スカウト

Apache Scout は、Apache プロジェクトによって作成された JAXR のオープンソース実装です。

現在、**org.jboss.soa.esb.scout.proxy.transportClass** クラスにはそれぞれ SOAP、SAAJ、RMI、および Embedded Java (Local) 用の実装が 4 つあります。

バグの報告

9.11. JAVA API FOR XML REGISTRIES (JAXR)

Java API for XML Registries (JAXR) は、サービスレジストリー用に開発するための標準的な方法を提供する API です。

バグの報告

9.12. レジストリーインターフェイス

Registry Interface は、Mamelets が Service Registry と通信する手段です。

バグの報告

9.13. レジストリーの設定

はじめに

通常、データベース設定ツールの実行時に jUDDI レジストリーを自動的に設定します。

手動で設定する高度なオプションについては、本セクションをお読みください。

バグの報告

9.14. レジストリーの手動設定

手順9.1 タスク

1. グローバル設定ファイルをテキストエディターで開きます: **vi SOA_ROOT/jboss-as/server/PROFILE/deployers/esb.deployer/jbossesb-properties.xml**
2. registry セクションまでスクロールダウンし、必要に応じて設定を変更します。

```
<properties name="registry">
  <property name="org.jboss.soa.esb.registry.implementationClass"
    value="org.jboss.internal.soa.esb.services.registry.JAXRRegistryImpl"/>

  <property name="org.jboss.soa.esb.registry.factoryClass"
    value="org.apache.ws.scout.registry.ConnectionFactoryImpl"/>

  <property name="org.jboss.soa.esb.registry.queryManagerURI"
    value="org.apache.juddi.v3.client.transport.wrapper.UDDIInquiryService#inquire"/>

  <property name="org.jboss.soa.esb.registry.lifeCycleManagerURI"
    value="org.apache.juddi.v3.client.transport.wrapper.UDDIPublicationService#publish"/>
  <property name="org.jboss.soa.esb.registry.securityManagerURI"
    value="org.apache.juddi.v3.client.transport.wrapper.UDDISecurityService#secure"/>

  <property name="org.jboss.soa.esb.registry.user" value="root"/>
  <property name="org.jboss.soa.esb.registry.password" value="root"/>

  <property name="org.jboss.soa.esb.scout.proxy.uddiVersion" value="3.0"/>
  <property name="org.jboss.soa.esb.scout.proxy.uddiNameSpace" value="urn:uddi-
org:api_v3"/>

  <property name="org.jboss.soa.esb.scout.proxy.transportClass"
    value="org.apache.ws.scout.transport.LocalTransport"/>
  <!-- specify the interceptors, in order -->
  <property name="org.jboss.soa.esb.registry.interceptors"
  value="org.jboss.internal.soa.esb.services.registry.InVMRegistryInterceptor,
org.jboss.internal.soa.esb.services.registry.CachingRegistryInterceptor"/>
  <!-- The following properties modify the cache interceptor behaviour -->
  <property name="org.jboss.soa.esb.registry.cache.maxSize" value="100"/>
  <property name="org.jboss.soa.esb.registry.cache.validityPeriod" value="600000"/>

  <!-- Organization Category to be used by this deployment. -->
  <property name="org.jboss.soa.esb.registry.orgCategory"
  value="org.jboss.soa.esb.:category"/>
</properties>
```

3. ファイルを保存して終了します。

バグの報告

9.15. レジストリー設定オプションの表

表9.1 レジストリーのプロパティー

プロパティー	説明
org.jboss.soa.esb.registry.implementationClass	これは、JBossxhtml Registry インターフェイスを実装するクラスです。1 つの実装である **JAXRRegistryImpl** が含まれます。 **JAXRRegistry** インターフェイスを使用します。
org.jboss.soa.esb.registry.factoryClass	これは JAXR ConnectionFactory 実装のクラス名です。
org.jboss.soa.esb.registry.queryManagerURI	これは、JAXR がサービスをクエリーするために使用する URI です。
org.jboss.soa.esb.registry.lifeCycleManagerURI	これは JAXR が編集に使用する URI です。
org.jboss.soa.esb.registry.user	これは、編集に使用するユーザー名です。
org.jboss.soa.esb.registry.password	これは、指定したユーザーのパスワードです。
org.jboss.soa.esb.scout.proxy.uddiVersion	これは、クエリーの UDDI バージョンです。
org.jboss.soa.esb.scout.proxy.uddiNameSpace	これは UDDI 名前空間です。
org.jboss.soa.esb.scout.proxy.transportClass	これは、 **Apache Scout** がアイテムを UDDI レジストリーに送信するために使用するクラスです。
org.jboss.soa.esb.registry.interceptors	これは、レジストリーに適用される *インターセプター* の一覧です。アーキテクトは、InVM 登録を処理するためのインターセプターと、レジストリーにキャッシュを提供するインターセプターを 2 つ提供します。 デフォルトのインターセプターリストには、InVM インターセプターのエントリーが 1 つだけ含まれます。
org.jboss.soa.esb.registry.cache.maxSize	これは、キャッシュで許可されるサーバーエントリーの最大数です。この値を超えると、エントリーは最低限使用(Least Recently Used)ベースで削除されます。デフォルト値は **100** です。
org.jboss.soa.esb.registry.cache.validityPeriod	これは、キャッシュインターセプターに設定された有効期間です。値はミリ秒単位で指定され、デフォルトは **600000** (10 分)です。キャッシュの有効期限が切れない場合は、この値を **0** に設定します。

プロパティー	説明
org.jboss.soa.esb.registry.orgCategory	これは、Answan の *組織カテゴリー* の名前です。デフォルトは **org.jboss.soa.esb.:category** です。

バグの報告

9.16. レジストリー設定のユースケース

ユーザーがレジストリーを手動で設定する必要がある場合があります。

- queryManagerURI/lifeCycleManagerURI/securityManagerURI を、UDDI レジストリーの場所と一致するように変更します。
- レジストリーのユーザー/パスワードを変更するには、以下を実行します。
- カスタムの org.jboss.soa.esb.registry.orgCategory を提供する
- カスタムレジストリーインターセプターを指定します。

バグの報告

9.17. レジストリーの埋め込み

サーバーコンポーネントで単一の jUDDI レジストリーを共有する場合は、これを埋め込む必要があります。この方法では、JBoss Enterprise SOA Platform 自体の複数のインスタンスが同じレジストリーを共有できるようにします。

バグの報告

9.18. レジストリーの埋め込み

手順9.2 タスク

1. グローバル設定ファイルをテキストエディターで開きます: **vi SOA_ROOT/jboss-as/server/PROFILE/deployers/esb.deployer/jbossesb-properties.xml**
2. registry セクションまでスクロールダウンし、以下のように設定を変更します。

```
<properties name="registry">

    <property name="org.jboss.soa.esb.registry.implementationClass"
    value="org.jboss.internal.soa.esb.services.registry.JAXRRegistryImpl"/>

    <property name="org.jboss.soa.esb.registry.factoryClass"
    value="org.apache.ws.scout.registry.ConnectionFactoryImpl"/>
```

```
    <property name="org.jboss.soa.esb.registry.queryManagerURI"
    value="org.apache.juddi.registry.local.InquiryService#inquire"/>

    <property name="org.jboss.soa.esb.registry.lifeCycleManagerURI"
    value="org.apache.juddi.registry.local.PublishService#publish"/>

    <property name="org.jboss.soa.esb.registry.securityManagerURI"
    value="org.apache.juddi.registry.local.SecurityService#secure"/>

    <property name="org.jboss.soa.esb.registry.user" value="jbossesb"/>
    <property name="org.jboss.soa.esb.registry.password" value="password"/>

    <property name="org.jboss.soa.esb.scout.proxy.transportClass"
    value="org.apache.ws.scout.transport.LocalTransport"/>

</properties>
```

3. ファイルを保存して終了します。

バグの報告

9.19. リモートメソッド呼び出しを使用するようにレジストリーを設定する

レジストリーは RMI を使用するように設定できます。(JBoss Enterprise SOA Platform はデフォルトでリモートメソッド呼び出しサービスをデプロイします。これは、**jbossesb.sar** アーカイブ内でレジストリーを起動しますが、リモートメソッド呼び出しサービスも自動的に登録するアーカイブと同じです。)

バグの報告

9.20. リモートメソッド呼び出しを使用するようにレジストリーを設定する

手順9.3 タスク

1. グローバル設定ファイルをテキストエディターで開きます: **vi SOA_ROOT/jboss-as/server/PROFILE/deployers/esb.deployer/jbossesb-properties.xml**

2. registry セクションまでスクロールダウンし、以下のように設定を変更します。

```
<properties name="registry">
  <property name="org.jboss.soa.esb.registry.implementationClass"
    value="org.jboss.internal.soa.esb.services.registry.JAXRRegistryImpl"/>

  <property name="org.jboss.soa.esb.registry.factoryClass"
    value="org.apache.ws.scout.registry.ConnectionFactoryImpl"/>

  <property name="org.jboss.soa.esb.registry.queryManagerURI"
    value="jnp://localhost:1099/InquiryService?org.apache.juddi.registry.rmi.Inquiry#inquire"/>

  <property name="org.jboss.soa.esb.registry.lifeCycleManagerURI"
    value="jnp://localhost:1099/PublishService?
org.apache.juddi.registry.rmi.Publish#publish"/>
```

```
  <property name="org.jboss.soa.esb.registry.securityManagerURI"
    value="jnp://localhost:1099/PublishService?
org.apache.juddi.registry.rmi.Publish#publish"/>

  <property name="org.jboss.soa.esb.registry.user" value="jbossesb"/>
  <property name="org.jboss.soa.esb.registry.password" value="password"/>

  <property name="org.jboss.soa.esb.scout.proxy.transportClass"
    value="org.apache.ws.scout.transport.RMITransport"/>
</properties>
```

3. ファイルを保存して終了します。
4. テキストエディターで **web.xml** ファイルを開きます。
5. 以下のように設定します。

```
<!--  uncomment if you want to enable making calls in juddi with rmi  -->
 <servlet>
  <servlet-name>RegisterServicesWithJNDI</servlet-name>
  <servlet-class>org.apache.juddi.registry.rmi.RegistrationService</servlet-class>
  <load-on-startup>1</load-on-startup>
 </servlet>
```

6. ファイルを保存して終了します。
7. テキストエディターで jUDDI 設定ファイルを開きます(**vi SOA_ROOT/jboss-as/server/standard/deploy/juddi-service.sar/juddi.war/WEB-INF/juddi.properties**)。
8. 以下のように設定します。

```
# JNDI settings (used by RMITransport)
java.naming.factory.initial=org.jnp.interfaces.NamingContextFactory
java.naming.provider.url=jnp://localhost:1099
java.naming.factory.url.pkgs=org.jboss.naming
```

9. ファイルを保存して終了します。
10. **scout-client.jar** を RMI クライアントのクラスパスに追加します。

バグの報告

9.21. RMI サービスのカスタム JNDI 登録を使用したリモートメソッド呼び出し

何らかの理由で **juddi.war** アーカイブをデプロイしたくない場合は、jUDDI レジストリーと同じ Java 仮想マシンで実行している Enterprise Service Bus コンポーネントのいずれかを設定し、RMI サービスを登録することができます。

バグの報告

9.22. 独自の JNDI 登録を使用した RMI の設定

1. ローカルアプリケーションの場合は、テキストエディターでグローバル設定ファイル **vi SOA_ROOT/jboss-as/server/PROFILE/deployers/esb.deployer/jbossesb-properties.xml** を開きます。

2. registry セクションまでスクロールダウンし、以下のようにローカル設定を変更します。

```
<properties name="registry">
 <property name="org.jboss.soa.esb.registry.implementationClass"
value="org.jboss.internal.soa.esb.services.registry.JAXRRegistryImpl"/>

 <property name="org.jboss.soa.esb.registry.factoryClass"
value="org.apache.ws.scout.registry.ConnectionFactoryImpl"/>

    <property name="org.jboss.soa.esb.registry.queryManagerURI"
value="org.apache.juddi.registry.local.InquiryService#inquire"/>

    <property name="org.jboss.soa.esb.registry.lifeCycleManagerURI"
value="org.apache.juddi.registry.local.PublishService#publish"/>

    <property name="org.jboss.soa.esb.registry.securityManagerURI"
value="org.apache.juddi.registry.local.SecurityService#secure"/>

    <property name="org.jboss.soa.esb.registry.user" value="jbossesb"/>
    <property name="org.jboss.soa.esb.registry.password" value="password"/>

    <property name="org.jboss.soa.esb.scout.proxy.transportClass"
value="org.apache.ws.scout.transport.LocalTransport"/>
</properties>
```

3. ファイルを保存して終了します。

4. リモートアプリケーションの場合、テキストエディターでグローバル設定ファイル **vi SOA_ROOT/jboss-as/server/PROFILE/deployers/esb.deployer/jbossesb-properties.xml** を開きます。

5. registry セクションまでスクロールダウンし、以下のようにリモートメソッド呼び出し設定を変更します。

```
<properties name="registry">
 <property name="org.jboss.soa.esb.registry.implementationClass"
value="org.jboss.internal.soa.esb.services.registry.JAXRRegistryImpl"/>

 <property name="org.jboss.soa.esb.registry.factoryClass"
value="org.apache.ws.scout.registry.ConnectionFactoryImpl"/>

    <property name="org.jboss.soa.esb.registry.queryManagerURI"
value="jnp://localhost:1099/InquiryService?org.apache.juddi.registry.rmi.Inquiry#inquire"/>

    <property name="org.jboss.soa.esb.registry.lifeCycleManagerURI"
value="jnp://localhost:1099/PublishService?org.apache.juddi.registry.rmi.Publish#publish"/>

    <property name="org.jboss.soa.esb.registry.user" value="jbossesb"/>
    <property name="org.jboss.soa.esb.registry.password" value="password"/>
```

```
    <property name="org.jboss.soa.esb.scout.proxy.transportClass"
value="org.apache.ws.scout.transport.RMITransport"/>
</properties>
```

6. ファイルを保存して終了します。

7. **queryManagerURI** クラスおよび **lifeCycleManagerURI** クラスのホスト名を、jUDDI レジストリーが実行されているホストに指定します（これはローカルが実行されている場所でもあります）。ローカルアプリケーションはネーミングサービスにアクセスできる必要があることに注意してください。

8. これらの設定を使用して、アプリケーションを登録します。

```
//Getting the JNDI setting from the config
Properties env = new Properties();
env.setProperty(RegistryEngine.PROPNAME_JAVA_NAMING_FACTORY_INITIAL,factoryIniti
al);
env.setProperty(RegistryEngine.PROPNAME_JAVA_NAMING_PROVIDER_URL,
providerURL);
env.setProperty(RegistryEngine.PROPNAME_JAVA_NAMING_FACTORY_URL_PKGS,
  factoryURLPkgs);

InitialContext context = new InitialContext(env);
Inquiry inquiry = new InquiryService();
log.info("Setting " + INQUIRY_SERVICE + ", " + inquiry.getClass().getName());
mInquery = inquiry;
context.bind(INQUIRY_SERVICE, inquiry);
Publish publish = new PublishService();
log.info("Setting " + PUBLISH_SERVICE + ", " + publish.getClass().getName());
mPublish = publish;
context.bind(PUBLISH_SERVICE, publish);
```

9. RMI クライアントのクラスパスに **scout-client.jar** ファイルを追加します。

バグの報告

9.23. SOAP 経由で JUDDI レジストリーと通信します。

(Apache Scout を介して) SOAP を使用して、Enterprise Service Bus を jUDDI レジストリーと通信できます。

バグの報告

9.24. SIMPLE OBJECT ACCESS PROTOCOL (SOAP)

Simple Object Access Protocol (SOAP)は、ユーザーがメッセージの内容を定義し、受信者がそのメッセージを処理する方法をヒントを提供できるようにする軽量プロトコルです。SOAP は XML ベースの通信プロトコルです。

バグの報告

9.25. SOAP を使用するよう APACHE スカウトを設定

手順9.4 タスク

1. **web.xml** ファイルの RegisterServicesWithJNDI サーブレットをコメントアウトして RMI サービスをシャットダウンします。
2. **juddi.war** アーカイブをデプロイします。
3. データソースを設定します。

   以下は、プロパティーの例です。

```
<properties name="registry">
 <property name="org.jboss.soa.esb.registry.implementationClass"
 value="org.jboss.internal.soa.esb.services.registry.JAXRRegistryImpl"/>

 <property name="org.jboss.soa.esb.registry.factoryClass"
 value="org.apache.ws.scout.registry.ConnectionFactoryImpl"/>

   <property name="org.jboss.soa.esb.registry.queryManagerURI"
   value="http://localhost:8080/juddi/inquiry"/>

   <property name="org.jboss.soa.esb.registry.lifeCycleManagerURI"
   value="http://localhost:8080/juddi/publish"/>

   <property name="org.jboss.soa.esb.registry.user" value="jbossesb"/>
   <property name="org.jboss.soa.esb.registry.password" value="password"/>

   <property name="org.jboss.soa.esb.scout.proxy.transportClass"
   value="org.apache.ws.scout.transport.AxisTransport"/>
</properties>
```

バグの報告

9.26. JUDDI コンソール

jUDDI コンソールは、jUDDI レジストリーを設定するために使用する必要がある Web ベースのグラフィカルインターフェイスです。http://localhost:8080/uddi-console/ でアクセスできます。

バグの報告

9.27. JUDDI コンソールへのアクセスを許可する

前提条件

- user と admin のセキュリティーロールが割り当てられた root という名前のユーザー。

すべてのユーザーに管理権限を付与するには、root という名前の jUDDI パブリッシャーとしてログインする必要があります。ユーザーがこれらの管理権限を取得すると、その権限を他のユーザーに付与できます。

手順9.5 タスク

1. Web ブラウザーセッションを開き、http://localhost:8080/uddi-console/ で jUDDI コンソールに移動します。root でログインします。
2. 発行者をクリックします。
3. パブリッシャー ID リストから、ユーザー名をクリックします。
4. 管理者 チェックボックスを選択します。

結果

選択したユーザーに管理者権限が付与されました。

バグの報告

9.28. SOA 用の JBOSS DEVELOPER STUDIO プラグインのインストール

前提条件

- JBoss Developer Studio
- JBoss Tools コンポーネント

手順9.6 タスク

1. JBoss Developer Studio の起動
2. JBoss Central 画面に移動
3. SOA 開発作業に必要なプラグインをダウンロードします。

バグの報告

第10章 高度な SERVICE REGISTRY 設定オプション

10.1. 代替の JAXR 実装の設定

手順10.1 タスク

1. 特定の JAXR 実装を選択します。
2. プロパティーを設定してクラスを設定します。(**JBoss Enterprise SOA Platform** はデフォルトで **Apache Scout** を使用するため、このプロパティーは Scout ファクトリークラス(**org.apache.ws.scout.registry.ConnectionFactoryImpl**.)に設定されます。)
3. レジストリーの場所を指定して JAXR 実装を設定します。これを行うには、**org.jboss.soa.esb.registry.queryManagerURI**、**org.jboss.soa.esb.registry.lifeCycleManagerURI**、および **org.jboss.soa.esb.registry.securityManagerURI** を編集します。

バグの報告

10.2. JAXR への代替 API の設定

手順10.2 タスク

- 新しい **SystinetRegistryImplemtation** クラスを作成し、このプロパティー内からそのクラスへの参照を提供します。

バグの報告

10.3. トランスポートの使用

Apache Scout を使用する場合は、特別なオプションパラメーター **org.jboss.soa.esb.scout.proxy.transportClass** を設定できます。これは、 **Scout** レジストリーと jUDDI レジストリー間の通信を容易にするトランスポートクラスです。

Scout を使用して jUDDI と通信する場合は、トランスポートクラスを **LocalTransport** のままにし、jUDDI レジストリーのトランスポート(InVM、RMI、および WS)を利用するようにファイルを設定します。このファイルは、レジストリーのノードを定義します。

バグの報告

10.4. ノード

ノードはサービスレジストリーの場所です。

バグの報告

10.5. トランスポートの選択

手順10.3 タスク

1. ノード定義ファイル **vi SOA_ROOT/jboss-as/server/PROFILE/deploy/jbossesb.sar/esb.juddi.client.xml** を編集します。
2. ノード設定を使用して、使用するトランスポートを選択します。

```
<node>
    <!-- required 'default' node -->
    <name>default</name>
    <description>Main jUDDI node</description>
    <properties>
 <property name="serverName" value="localhost" />
  <property name="serverPort" value="8880" />
    </properties>
    <!-- JAX-WS Transport
    <proxyTransport>org.apache.juddi.v3.client.transport.JAXWSTransport</proxyTransport>
    <custodyTransferUrl>http://${serverName}:${serverPort}/juddiv3/services/custody-
transfer?wsdl</custodyTransferUrl>
   <inquiryUrl>http://${serverName}:${serverPort}/juddiv3/services/inquiry?wsdl</inquiryUrl>
   <publishUrl>http://${serverName}:${serverPort}/juddiv3/services/publish?
wsdl</publishUrl>
   <securityUrl>http://${serverName}:${serverPort}/juddiv3/services/security?
wsdl</securityUrl>
   <subscriptionUrl>http://${serverName}:${serverPort}/juddiv3/services/subscription?
wsdl</subscriptionUrl>
   <subscriptionListenerUrl>http://${serverName}:${serverPort}/juddiv3/services/subscription-
listener?wsdl</subscriptionListenerUrl>
   <juddiApiUrl>http://${serverName}:${serverPort}/juddiv3/services/juddi-api?
wsdl</juddiApiUrl>
    -->
   <!-- In VM Transport Settings

<proxyTransport>org.jboss.internal.soa.esb.registry.client.JuddiInVMTransport</proxyTransport
>

<custodyTransferUrl>org.apache.juddi.api.impl.UDDICustodyTransferImpl</custodyTransferUrl
>
    <inquiryUrl>org.apache.juddi.api.impl.UDDIInquiryImpl</inquiryUrl>
    <publishUrl>org.apache.juddi.api.impl.UDDIPublicationImpl</publishUrl>
   <securityUrl>org.apache.juddi.api.impl.UDDISecurityImpl</securityUrl>
   <subscriptionUrl>org.apache.juddi.api.impl.UDDISubscriptionImpl</subscriptionUrl>

<subscriptionListenerUrl>org.apache.juddi.api.impl.UDDISubscriptionListenerImpl</subscription
ListenerUrl>
   <juddiApiUrl>org.apache.juddi.api.impl.JUDDIApiImpl</juddiApiUrl>
  -->
 <!-- RMI Transport Settings -->
<proxyTransport>org.apache.juddi.v3.client.transport.RMITransport</proxyTransport>
 <custodyTransferUrl>/juddiv3/UDDICustodyTransferService</custodyTransferUrl>
 <inquiryUrl>/juddiv3/UDDIInquiryService</inquiryUrl>
 <publishUrl>/juddiv3/UDDIPublicationService</publishUrl>
<securityUrl>/juddiv3/UDDISecurityService</securityUrl>
<subscriptionUrl>/juddiv3/UDDISubscriptionService</subscriptionUrl>
<subscriptionListenerUrl>/juddiv3/UDDISubscriptionListenerService</subscriptionListenerUrl
>
```

```
<juddiApiUrl>/juddiv3/JUDDIApiService</juddiApiUrl>
<javaNamingFactoryInitial>org.jnp.interfaces.NamingContextFactory</javaNamingFactoryInitial>
 <javaNamingFactoryUrlPkgs>org.jboss.naming</javaNamingFactoryUrlPkgs>
 <javaNamingProviderUrl>jnp://localhost:1099</javaNamingProviderUrl>
 </node>
```

3. デフォルトでは、RMI (Remote Method Invocation)設定が有効になります。トランスポートを切り替えるには、それらのトランスポートをコメントアウトし、使用するものを有効にします。

4. ファイルを保存して終了します。

バグの報告

10.6. リモート呼び出しクラス

リモート呼び出しクラスは、その名前が示すように、リモートマシンから呼び出すことができるクラスです。これは開発者にとっては便利ですが、潜在的なセキュリティーリスクにつながる可能性もあります。

バグの報告

10.7. トランスポート設定

トランスポートを設定する場合は、以下の項目を指定する必要があります。

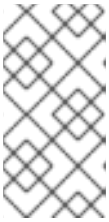
- a **proxyTransport**
- サポートされるすべての UDDI アプリケーションプログラミングインターフェイスの URL (、**セキュリティー** 、 サブスクリプション、 **サブスクリプション**、 **リスナー**、および **custodytransfer**が必要)
- jUDDI アプリケーションプログラミングインターフェイス URL。
- RMI トランスポート(*JNDI* 設定も含まれます)

バグの報告

10.8. APACHE スカウトの設定

手順10.4 タスク

1. **uddi.xml** ファイルで **transportClass** が LocalTransport に設定されていることを確認します。
2. レジストリーのトランスポート（それぞれ InVM、RMI、および WS）を使用するように **uddi.xml** ファイルを設定します。このファイルは **SOA_ROOT/jboss-as/server/all/deploy/jbossesb.sar/esb.juddi.client.xml**にあります。
3. **jbossesb パブリッシャー** を追加して、jUDDI レジストリーにスキーマを作成します。



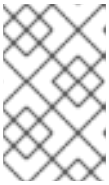
注記

product/install/jUDDI-registry ディレクトリーには、最も一般的なデータベースの **database-create** スクリプトが含まれています。これらのファイルは **SOA_ROOT/jboss-as/server/PROFILE/deploy/jbossesb-registry.sar/juddi-sql** にあります。

注記

ユーザーがテーブルを作成するパーミッションが付与されている場合、システムはデータベースを自動的に生成できます。（レジストリーは、関連付けられた *Hibernate* 方言が存在する任意のタイプのデータベースを作成できます。）

4. **SOA_ROOT/jboss-as/server/default/deploy/jbossesb-registry.sar/esb.juddi.xml** および **SOA_ROOT/jboss-as/server/all/deploy/jbossesb.sar/esb.juddi.client.xml** が存在することを確認します。これらのファイルには、レジストリーの設定が含まれます。



注記

ソフトウェアを別の UDDI レジストリーと通信させるには、Apache Scout の JAXR トランスポートを使用します。このクラスには 4 つの実装があり、SOAP、SAAJ、RMI、および組み込み Java に基づいています。

5. トランスポートを変更する場合には、クエリーとライフサイクルの統一されたリソースインジケーターも常に変更します。以下は、その方法を示すサンプルコードです。

```
SOAP
queryManagerURI http://localhost:8080/juddi/inquiry
lifeCycleManagerURI http://localhost:8080/juddi/publish
transportClass  org.apache.ws.scout.transport.AxisTransport

RMI
queryManagerURI  jnp://localhost:1099/InquiryService?
org.apache.juddi.registry.rmi.Inquiry#inquire
lifeCycleManagerURI  jnp://localhost:1099/PublishService?
org.apache.juddi.registry.rmi.Publish#publish
transportClass  org.apache.ws.scout.transport.RMITransport

Local
queryManagerURI org.apache.juddi.registry.local.InquiryService#inquire
lifeCycleManagerURI org.apache.juddi.registry.local.PublishService#publish
transportClass  org.apache.ws.scout.transport.LocalTransport
```

バグの報告

10.9. インターセプター

インターセプターは、要求をインターセプトするために Service Registry によって使用されます。これらはインターセプタースタックに保存されます。スタックの各インターセプターは、以下を実行できます。

- 要求のサービス

- 要求への直接応答を提供します。
- 下位インターセプターまたはレジストリー実装から受信した応答の拡張

現在の実装には、インターセプターが 2 つあります。

バグの報告

10.10. LOCALREGISTRYINTERCEPTOR

LocalRegistryInterceptor は、ローカルサービスを処理するクラスです。

バグの報告

10.11. インターセプタースタックの設定

1. グローバル設定ファイルをテキストエディターで開きます: **vi SOA_ROOT/jboss-as/server/PROFILE/deployers/esb.deployer/jbossesb-properties.xml**
2. org.jboss.soa.esb.registry.interceptors プロパティーを変更します。
3. ファイルを保存して終了します。

バグの報告

10.12. インターセプターの設定

表10.1 インターセプタープロパティー

プロパティー	Description
org.jboss.internal.soa.esb.services.registry.InVMRegistryInterceptor	InVM レジストリーインターセプターは、InVM エンドポイント参照を処理します。これらは、同じサーバーインスタンス内で実行されるサービスのいずれかで登録されます。InVM EPR とその関連サービスに関する情報はインターセプター内でキャッシュされ、後続のインターセプターまたはレジストリークエリーから結果を拡張することで呼び出し元に返されます。

プロパティー	Description
org.jboss.internal.soa.esb.services.registry.CachingRegistryInterceptor	キャッシュレジストリーインターセプターは、エンドポイント参照とその関連サービスのキャッシュを保持し、LRU ベースまたは情報の有効期限が切れた後に、キャッシュから情報をエビクトします。 インターセプターは、jbossesb-properties.xml 内の org.jboss.soa.esb.registry.cache.maxSize および org.jboss.soa.esb.registry.cache.validityPeriod プロパティーで設定できます。

バグの報告

第11章 BPEL エンジンとの SERVICE REGISTRY の統合

11.1. BPEL エンジン

BPEL エンジンは BPEL ビジネスプロセス命令を実行します。JBoss Enterprise SOA Platform 製品の一部として含まれる BPEL エンジンは、Apache ODE に基づいています。

注記

ブラウザーで BPEL コンソールウィンドウを 1 つだけ開くことを推奨します。そうしないと、ログイン時に空白のウィンドウが表示されたり、2 つ目のウィンドウからログインできなくなったりする可能性があります。詳細は、RIFTSAW-400 を参照してください。

バグの報告

11.2. BUSINESS PROCESS EXECUTION LANGUAGE (BPEL)

ビジネスプロセス実行言語 (BPEL) は、ビジネスルールオーケストレーション用の OASIS 標準言語です。詳細は、http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html を参照してください。

バグの報告

11.3. BPEL と SERVICE REGISTRY

BPEL は Service Registry と統合されているため、サービスはデプロイ時に自動的に登録できます。

この登録プロセスは、jUDDI クライアントライブラリーを使用します。サービスがデプロイされると、その BindingTemplate とその BindingTemplate （エンドポイント参照）の両方が登録され、partnerLinkChannel ごとに partnerLinkChannel が作成されます。同時に、WSDL エンドポイントは UDDI から取得されます。

デプロイを解除すると、BindingTemplate は **UDDI レジストリー から削除されます**。

バグの報告

11.4. BPEL-SERVICE レジストリー統合の有効化

手順11.1 タスク

- インテグレーションはデフォルトで有効になっています。これを確認するには、**vi SOA_ROOT/jboss-as/server/PROFILE/deploy/riftsaw.sar/bpel.properties.xml** を開き、が **bpel.uddi.registration=true** に設定されていることを確認します。

バグの報告

11.5. パートナーリンク

パートナーリンクは、BPEL プロセスとクライアント間の関係を確立するリンクです。

バグの報告

11.6. パートナーリンクチャンネル

パートナーリンクチャンネルは、BPEL プロセスに統合されているクライアントとサービスとの対話に使用される通信チャネルです。

バグの報告

11.7. ESB.JUDDI.CLIENT.XML

SOA_ROOT/jboss-as/server/PROFILE/deploy/jbossesb.sar/esb.juddi.client.xml ファイルは、jUDDI Service Registry のクライアント設定ファイルです。

バグの報告

11.8. BPEL.PROPERTIES 設定設定

表11.1 bpel.properties ファイルの UDDI 関連のプロパティー

attribute	タイプ（デフォルト）	description
bpel.uddi.registration	boolean (true)	'false' に設定すると、UDDI 統合はオフになります。RiftSaw インストールプロセスでは、jbossesb-registry.sar に jUDDI v3 レジストリーが含まれることが検出されると、この値を true に設定します。他のすべての場合は、自動的に false に設定されます。
bpel.webservice.secure	boolean (false)	UDDI 登録プロセスでは、登録している BPEL サービスの BindingTemplate に WSDL AccessPoint を登録します。BPEL サーバーは WS スタック上のサービス WSDL エンドポイントを公開します（現在、Red Hat は JBossWS および CXF をサポートします）。web サービススタックがセキュアなプロトコル(https など)を使用するように設定されている場合は、この設定を true に切り替える必要があります。（この設定は登録プロセス中にのみ使用されることに注意してください。）

attribute	タイプ（デフォルト）	description
bpel.uddi.client.impl	String (org.jboss.soa.bpel.uddi.UDDIRegistrationImpl)	これは、org.jboss.soa.bpel.runtime.engine.ode.UDDIRegistration インターフェイスを実装するクラスの名前です。
bpel.uddi.clerk.config	文字列（デフォルトでは使用されません）	これは、**bpel.uddi.client.xml** 設定ファイルへのパスを定義します。**riftsaw.sar/META-INF/riftsaw.uddi.xml** を使用する場合は、コメントアウトしたままにすることができます。この場合、**bpel.uddi.clerk.manager** を定義する必要があります。
bpel.uddi.clerk.manager	string (riftsaw-manager)	これは、riftsaw.uddi.xml がコメントアウトされている場合に使用される ClerkManager の名前を定義します。この値は、**esb.juddi.client.xml** のマネージャーの名前に対応している必要があります。bpel.uddi.clerk.config が定義されている場合、bpel.uddi.clerk.manager の設定は無視されます。
bpel.uddi.clerk	string (BPEL_clerk)	使用する clerk の名前を定義します。この値は、**riftsaw.uddi.xml** の clerk の名前に対応している必要があります。（デフォルトでは BPEL_clerk に設定されています。）

attribute	タイプ（デフォルト）	description
bpel.uddi.lookup	boolean (true)	これを true に設定すると、パートナーチャネルの作成プロセスは UDDI の serviceName によるルックアップを実行し、WSDL エンドポイントが取得されます。このプロセスにより、BPEL プロセスデプロイメントでパートナーリンク WSDL ファイルを更新することなく、サーバーファーム内のプロセスデプロイメントを簡単に移動できます。複数のエンドポイント (BindingTemplate)が見つかった場合、ServiceLocator が使用するデフォルトのポリシーは PolicyLocalFirst になります。各 partnerLink に初期パートナーリンク WSDL ファイルをデプロイする必要があることに注意してください。

注記

ClerkManager と Clerk の両方の名前は **bpel.properties** ファイルに指定されます。

バグの報告

11.9. CLERK

clerk (**org.apache.juddi.v3.client.config.UDDIClerk**)は、Service Registry にサービスのエンドポイントを登録します。

バグの報告

11.10. サービス登録時に CLERK が使用するプロパティーを設定します。

手順11.2 タスク

1. テキストエディターで **esb.juddi.client.xml** ファイルを開きます。**vi SOA_ROOT/jboss-as/server/PROFILE/deploy/jbossesb.sar/esb.juddi.client.xml**
2. 設定を構成します。以下に例を示します。

```
</nodes>
  <clerks registerOnStartup="false">
    <clerk name="SOAExample" node="default" publisher="root" password="root"/>
  </clerks>
  </manager>
</uddi>
```

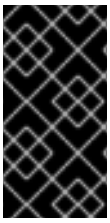
3. ファイルを保存して終了します。
4. ファイルの別のコピーをここに配置します（ファイルは常に対応する必要があります：**SOA_ROOT/jboss-as/server/PROFILE/deploy/jbossesb-registry.sar/juddi_custom_install_data/**）。
5. ファイルを保存して終了します。

バグの報告

11.11. SERVICE REGISTRY CLERK のデフォルト設定

表11.2 デフォルトの設定

プロパティー	値
keyDomain	esb.jboss.org
businessKey	redhat-jboss
serviceDescription	BPEL Service deployed by Riftsaw
bindingDescription	BPEL Endpoint deployed by Riftsaw



重要

SOA_ROOT/jboss-as/server/PROFILE/deploy/jbossesb-registry.sar/esb.juddi.xml ファイルには、 juddi.seed.always というプロパティーが含まれており、これは false に設定されます。つまり、サーバーの起動時に常に root シードデータを読み込もうとします。

バグの報告

11.12. UDDI 登録

BPEL プロセスをデプロイすると、BPEL4WS OASIS technotehttp://www.oasis-open.org/committees/uddi-spec/doc/tn/uddi-spec-tc-tn-bpel-20040725.htm（）に従ってプロセス情報が UDDI レジストリーに登録されます。

バグの報告

11.13. UDDI END-POINT LOOK-UP

BPEL サービスが別の BPEL サービス（または一般的な Web サービスのエンドポイント）を呼び出す場合、BPEL Engine は serviceQName および portName によるルックアップを実行します(WSDL から取得)。結果はクライアント側のサービスキャッシュに保存されるため、パフォーマンスが向上しま

す。クライアント側のキャッシュが Stale 情報を返しないようにするために、レジストリーは変更が発生するたびに、Subscription API を使用して UDDI レジストリーによりキャッシュを自動的に無効にします。

バグの報告

第12章 JAVA MESSAGE SERVICE PROVIDER の設定

12.1. JAVA MESSAGE SERVICE

Java Message Service (JMS) は、2 つのクライアント間でメッセージを送信するための Java API です。これにより、分散アプリケーションのさまざまなコンポーネントが相互に通信できるようになり、疎結合と非同期が可能になります。さまざまな Java Message Service プロバイダーが利用可能です。Red Hat は、HornetQ の使用を推奨しています。

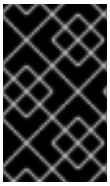
バグの報告

12.2. JAVA MESSAGE SERVICE PROVIDER の設定

はじめに

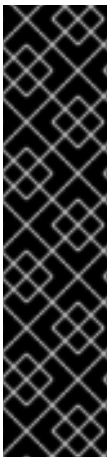
JBoss Enterprise SOA Platform と併用する多くの Java Message Service アプリケーションから選択できます。

サポートされる各 JMS アプリケーションの設定手順は、以下のセクションで説明します。



重要

本書の資料は、ベンダーが提供する Java Message Service ドキュメントを置き換えることは意図されていません。クラスターリングなどの高度な機能については、書籍を参照してください。



重要

以下のセクションでは、以下が前提となります。

- JMS プロバイダーが localhostで実行されている。
- connection-factory は **ConnectionFactory**です。
- destination-type は **キュー**です。
- destination-name は **キュー/A**です。

この資料の残りの部分を読む際には、これを念頭に置いてください。

バグの報告

12.3. サポートされる JAVA メッセージサービス

サポートされる JMS プロバイダーの完全なリストについては、サポートされる設定ページを参照し http://www.jboss.com/products/platforms/soa/supportedconfigurations てください。

Apache ActiveMQ や OracleAQ などの他の JSR-914 準拠の Java Message Service 実装http://jcp.org/en/jsr/detail?id=914も機能します。ただし、上記の Web サイトに記載されている JMS プロバイダーのみが完全にテストされています。

バグの報告

12.4. HORNETQ

HornetQ は Red Hat が開発したマルチプロトコルの非同期メッセージングシステムです。HornetQ は、サーバーに障害が発生した場合のメッセージの信頼性を保証するために、自動クライアントフェイルオーバーで高可用性(HA)を提供します。HornetQ は負荷分散されたメッセージで柔軟なクラスターリングソリューションもサポートします。

バグの報告

12.5. JAVA MESSAGE SERVICE PROVIDER として使用する HORNETQ の設定

警告

HornetQ インストーラーは、新規インストールでのみ実行されるように設計されており、元に戻すこともできず、複数回実行することもできます。

手順12.1 タスク

1. HornetQ ディレクトリー(**cd SOA_ROOT/jboss-as/extras/hornetq**)に移動します。
2. **デプロイ**
   ant を実行します。

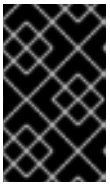
バグの報告

12.6. JAVA MESSAGE SERVICE PROVIDER として使用する JBOSS MESSAGING の設定

手順12.2 タスク

1. **JMS プロバイダーの設定**
   JBoss Messaging を設定するには、テキストエディターで設定ファイル **vi jboss-esb.xml** を開きます。
2. これらの設定を変更します。

```
jndi-URL="localhost”
 jndi-context-factory="org.jnp.interfaces.NamingContextFactory"
 connection-factory="ConnectionFactory"
 destination-type="queue"
 destination-name="queue/myqueue"
```



重要

常に class-path に **jboss-messaging-client.jar** ファイルを含めます。（このファイルは **SOA_ROOT/jboss-as/client/jboss-messaging-client.jar** アーカイブにあります。）

注記

負荷分散およびフェイルオーバー機能を使用する場合は、クラスターリング用に JBoss Messaging を設定する必要があります。この機能は常に進化しています。設定方法については、JBoss Messaging のドキュメントを参照してください。

3. JBoss Messaging の Service Binding Manager の設定が **SOA_ROOT/seam/bootstrap/deploy/remoting-service.xml** ファイルにあるものと一致することを確認します。

バグの報告

12.7. ACTIVEMQ JAVA MESSAGE SERVICE PROVIDER

重要

ActiveMQ は JBoss Enterprise SOA Platform 5 の認定またはサポート対象外のメッセージングプロバイダーではありません。

バグの報告

12.8. IBM WEBSPHERE MQ のインストールオプション

Websphere MQ JMS プロバイダーを設定するには、以下の 2 つの方法があります。

1. <jms-jca-provider> 設定を通じて Websphere MQ JCA アダプターを使用して JCA プロバイダーとして。

2. jms-provider 設定を使用した標準の JMS プロバイダー。

バグの報告

12.9. JAVA MESSAGE SERVICE PROVIDER として使用する IBM WEBSPHERE MQ の設定

手順12.3 タスク

1. IBM Websphere wmq.jmsra.rar ファイルを JBoss Enterprise SOA Platform (**cp wmq.jmsra.rar SOA_ROOT/jboss-as/server/PROFILE/deploy/**)にコピーします。

2. IBM Websphere com.ibm.mqetclient.jar を JBoss Enterprise SOA Platform (**cp com.ibm.mqetclient.jar SOA_ROOT/jboss-as/lib/**)にコピーします。

3. テキストエディターで run.conf を開きます(**vi SOA_ROOT/jboss-as/bin/run.conf**)。
4. **run.conf** の JAVA_OPTS を変更して、この設定を追加します。

```
JAVA_OPTS="-DtraceEnabled=true -DtraceDestination=wmq_jca.trc
   -DtraceLevel=10 -DlogWriterEnabled=false"
```

5. ファイルを保存して終了します。
6. テキストエディターで jboss-esb.xml ファイルを開きます(**vi SOA_ROOT/jboss-as/server/PROFILE/deploy/jbossesb.esb/META-INF/jboss-esb.xml**)。
7. 以下の設定を追加します。

```
<jms-jca-provider name="WMQ-JCA" connection-factory="MyAppXAConnectionFactory"
adapter="wmq.jmsra.rar"
      jndi-URL="10.12.10.110:1414/CH1"
      jndi-context-factory="com.ibm.mq.jms.context.WMQInitialContextFactory">
   <property name="max-xa-sessions-per-connection" value="1" />
<jms-bus busid="quickstartEsbChannel">
  <jms-message-filter dest-type="QUEUE" dest-name="Q1" transacted="true"/>
</jms-bus>
<activation-config>
    <property name="queueManager" value="QM1" />
    <property name="channel" value="CH1" />
    <property name="hostName" value="10.12.10.110" />
    <property name="port" value="1515" />
    <property name="transportType" value="CLIENT" />
    <property name="useJNDI" value="false" />
</activation-config>
</jms-jca-provider>
```

8. ファイルを保存して終了します。
9. **crtmqm -q QM1**を実行します。
10. Run **strmqm QM1**

```
define channel (CH.1) chltype (RCVR) trptype (TCP)
start channel (CH.1)
define qlocal (Q1)
define listener(QM1.LISTENER) trptype(TCP) port(30001) ipaddr(10.12.58.110)
start listener (QM1.LISTENER)
```

バグの報告

12.10. IBM WEBSPHERE MQ 設定チェックリスト

以下の項目がクラスパスに存在することを確認します。

- **com.ibm.mq.pcf.jar**
- **mqcontext.jar**

- **dhbcore.jar**
- **com.ibm.mq.jar**（クライアント **JAR**）
- **com.ibm.mqjms.jar**（クライアント **JAR**）

Websphere MQ v7.0 Client JAR ファイルを使用している場合は、以下をクラスパスに追加します。

- **com.ibm.mq.commonservices.jar**
- **com.ibm.mq.headers.jar**
- **com.ibm.mq.jmqi.jar**

注記

クライアント **JAR** ファイルは MQ 5.3 と MQ 6.0 によって異なります。ただし、6.0 の **JAR** ファイルは後方互換性を持つ必要があります。これらはオープンソースではないため、Red Hat は提供しません。WAS および MQ のインストールからそれらを取得する必要があります。

バグの報告

12.11. JAVA MESSAGE SERVICE および JNDI

Java が JMS を使用するための標準的な方法は、接続ファクトリーと JNDI で宛先を検索することです。これに使用される JNDI コンテキストファクトリーは com.ibm.mq.jms.context.WMQInitialContextFactory です。デフォルトの接続ファクトリーの名前はキューマネージャーと同じです。プレーン JMS 接続では、接続ファクトリーを JNDI に設定する必要があります。

バグの報告

12.12. JNDI を使用するように IBM WEBSPHERE MQ を設定する

前提条件

- IBM Websphere MQ がシステムにインストールされている。
- Java がクラスパス上にあることを確認します。

手順12.4 タスク

1. **/opt/mqm/java/bin/JMSAdmin -v**コマンドを実行します。

```
java -Dcom.ibm.msg.client.commonservices.log.outputName=$MQ_JAVA_DATA_PATH/log
-Dcom.ibm.msg.client.commonservices.trace.outputName=$MQ_JAVA_DATA_PATH/trace -
DMQJMS_INSTALL_PATH=$MQ_JAVA_INSTALL_PATH
com.ibm.mq.jms.admin.JMSAdmin $*
```

2. テキストエディターで JMSAdmin.config を開きます： **vi JMSAdmin.config**
3. このサンプルコードに従って ファイルを編集します。

```
INITIAL_CONTEXT_FACTORY=com.ibm.mq.jms.context.WMQInitialContextFactory

PROVIDER_URL=localhost:30002/SYSTEM.DEF.SVRCONN

SECURITY_AUTHENTICATION=none
```

注記

別のポートにリスナーがある場合は、代わりに使用します。（デフォルトは 1414 です）。

4. ファイルを保存して終了します。
5. 以下のコマンドを使用して、**InitCtx> def cf (ConnectionFactory) qmgr (QM1) tran (CLIENT) host (10.12.58.105) BROKERQMGR (QM1**)および **InitCtx> def qcf (QueueConnectionFactory) qmgr (QM1) tran (CLIENT) host (10.12.58.105)でオブジェクトを定義します。**
6. JMSAdmin に問題がある場合は、ログをトラブルシューティングしてください： **less /var/mqm/errors/AMQERR01.LOG** and **less /var/mqm/qmgrs/QUEUE MANAGER NAME/errors/AMQERR01.LOG**

注記

詳細は、を参照して http://www.ibm.com/developerworks/websphere/techjournal/0502_woolf/0502_w ください。

バグの報告

12.13. JMSADMIN で表示可能な JNDI オブジェクト

InitCtx> display qcf (QM1)

- ASYNCEXCEPTION(-1)
- CCSID(819)
- CHANNEL(SYSTEM.DEF.SVRCONN)
- CLIENTRECONNECTTIMEOUT(1800)
- COMPHDR(NONE)
- COMPMSG(NONE)
- CONNECTIONNAMELIST(10.12.58.105(1414))

- CONNOPT (STANDARD)
- FAILIFQUIESCE(YES)
- HOSTNAME(10.12.58.105)
- LOCALADDRESS()
- MAPNAMESTYLE (STANDARD)
- MSGBATCHSZ(10)
- MSGRETENTION (YES)
- POLLINGINT(5000)
- PORT(1414)
- PROVIDERVERSION (UNSPECIFIED)
- QMANAGER(QM1)
- RESCANINT(5000)
- SENDCHECKCOUNT(0)
- SHARECONVALLOWED (YES)
- SSLFIPSREQUIRED (NO)
- SSLRESETCOUNT(0)
- SYNCPOINTALLGETS (NO)
- TARGCLIENTMATCHING (YES)
- TEMPMODEL(SYSTEM.DEFAULT.MODEL.QUEUE)
- TEMPQPREFIX()
- TRANSPORT(CLIENT)
- USECONNPOOLING (YES)
- VERSION (7)
- WILDCARDFORMAT(TOPIC_ONLY)

InitCtx> display q (Q1)

- CCSID(1208)
- ENCODING (NATIVE)
- EXPIRY (APP)
- FAILIFQUIESCE(YES)

- MDMSGCTX(DEFAULT)
- MDREAD(NO)
- MDWRITE(NO)
- MSGBODY(UNSPECIFIED)
- PERSISTENCE (APP)
- PRIORITY (APP)
- PUTASYNCALLOWED(AS_DEST)
- QMANAGER(QM1)
- QUEUE(Q1)
- READAHEADALLOWED(AS_DEST)
- READAHEADCLOSEPOLICY(DELIVER_ALL)
- REPLYTOSTYLE （デフォルト）
- TARGCLIENT(JMS)
- VERSION (7)

InitCtx> display cf (ConnectionFactory)

- ASYNCEXCEPTION(-1)
- BROKERCCSUBQ(SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE)
- BROKERCONQ(SYSTEM.BROKER.CONTROL.QUEUE)
- BROKERPUBQ(SYSTEM.BROKER.DEFAULT.STREAM)
- BROKERQMGR(QM1)
- BROKERSUBQ(SYSTEM.JMS.ND.SUBSCRIBER.QUEUE)
- BROKERVER （未指定）
- CCSID(819)
- CHANNEL(SYSTEM.DEF.SVRCONN)
- CLEANUP (SAFE)
- CLEANUPINT(3600000)
- CLIENTRECONNECTTIMEOUT(1800)
- CLONESUPP （無効）
- COMPHDR(NONE)

- COMPMSG(NONE)
- CONNECTIONNAMELIST(10.12.58.105(1414))
- CONNOPT (STANDARD)
- FAILIFQUIESCE(YES)
- HOSTNAME(10.12.58.105)
- LOCALADDRESS()
- MAPNAMESTYLE (STANDARD)
- MSGBATCHSZ(10)
- MSGRETENTION (YES)
- MSGSELECTION(CLIENT)
- OPTIMISTICPUBLICATION (NO)
- OUTCOMENOTIFICATION (YES)
- POLLINGINT(5000)
- PORT(1414)
- PROCESSDURATION (UNKNOWN)
- PROVIDERVERSION (UNSPECIFIED)
- PUBACKINT(25)
- QMANAGER(QM1)
- RECEIVEISOLATION (COMMITTED)
- RESCANINT(5000)
- SENDCHECKCOUNT(0)
- SHARECONVALLOWED (YES)
- SPARSESUBS (NO)
- SSLFIPSREQUIRED (NO)
- SSLRESETCOUNT(0)
- STATREFRESHINT(60000)
- SUBSTORE (BROKER)
- SYNCPOINTALLGETS (NO)
- TARGCLIENTMATCHING (YES)

- TEMPMODEL(SYSTEM.DEFAULT.MODEL.QUEUE)
- TEMPQPREFIX()
- TEMPTOPICPREFIX()
- TRANSPORT(CLIENT)
- USECONNPOOLING (YES)
- VERSION (7)
- WILDCARDFORMAT(TOPIC_ONLY)

バグの報告

12.14. JMS-JCA-PROVIDER を使用した IBM WEBSPHERE MQ の設定

手順12.5 タスク

1. **アダプターの設定**
   アダプター名は **wmq.jmsra.rar** です。このファイルを JBoss Enterprise SOA Platform の **SOA_ROOT/jboss-as/server/PROFILE/deploy** ディレクトリーにコピーします。

2. **コンテキストファクトリーの確認**
   初期コンテキストファクトリーが **com.ibm.mq.jms.context.WMQInitialContextFactory** であることを確認します。

3. **QueueManager の設定**
   queueManager 設定が <activation-config> にあることを確認します。独自の Websphere MQ インストールでキューマネージャーを設定する必要があります。（すべての宛先キューがキューマネージャーで設定されていることに注意してください。）

4. **チャネルの設定**
   チャネル設定が <activation-config> プロパティーにあることを確認します。

   注記

   キューマネージャーは、さまざまな種類のチャネルを使用するように設定できます。クライアントはサーバー接続チャネルを介して接続します。Websphere MQ インストールの一部として、デフォルトのサーバー接続チャネル (SYSTEM.DEF.SVRCONN)が追加されますが、特に拡張(XA)トランザクションを使用している場合は、専用のチャンネルを作成することが推奨されます。

5. **transportType の設定**
   transportType を編集します。この設定は、<activation-config> プロパティーにあります。WebSphere MQ は、2 つのトランスポートタイプ（クライアントとバインディング）をサポートします。

6. **設定の編集**
   テキストエディターで設定ファイルを開き、編集します。

   以下は、<jms-jca-provider> 設定の例です。ゲートウェイプロバイダーの設定方法を示します。

   -

```
<jms-jca-provider adapter="wmq.jmsra.rar" connection-
factory="MyAppXAConnectionFactory" jndi-context-
factory="com.ibm.mq.jms.context.WMQInitialContextFactory" name="WMQ-JCA">
 <property name="max-xa-sessions-per-connection" value="1"/>
  <jms-bus busid="quickstartGwChannel">
  <jms-message-filter dest-name="QUEUE1_JMS" dest-type="QUEUE" transacted="true"/>
  </jms-bus>
   <activation-config>
    <property name="queueManager" value="TQM"/>
    <property name="channel" value="Q1CONN"/>
    <property name="hostName" value="localhost"/>
    <property name="port" value="1414"/>
    <property name="transportType" value="CLIENT"/>
   </activation-config>
</jms-jca-provider>
```

注記

これは実際には、リスナーのゲートウェイ/メッセージ対応特性を決定する <jms-listener> 設定です。ただし、Websphere MQ は、ゲートウェイリスナープロバイダーとしてのみ使用されます。つまり、この設定は Websphere MQ のメッセージ対応リスナーとしては機能しません。これは、プロバイダーに定義された宛先（バス）へのルーティングメッセージに対して、適切な JNDI プロバイダー URL (ServiceInvoker によって使用される)を指定していないためです。

7. **JCA アダプターの設定**
   <activation-config> は JCA アダプターを設定し、宛先からメッセージを取得するようにします。メッセージを宛先に配信する方法を決定しません。以下は、メッセージ対応リスナーのプロバイダーとして使用できる設定の例です。

```
<jms-jca-provider adapter="wmq.jmsra.rar" connection-
factory="MyAppXAConnectionFactory" jndi-URL="localhost:1414/CHANX" jndi-context-
factory="com.ibm.mq.jms.context.WMQInitialContextFactory" name="WMQ-JCA">
 <property name="max-xa-sessions-per-connection" value="1"/>
  <jms-bus busid="quickstartEsbChannel">
   <jms-message-filter dest-name="QUEUE2_JMS" dest-type="QUEUE" transacted="true"/>
  </jms-bus>
 <activation-config>
  <property name="queueManager" value="TQM"/>
  <property name="channel" value="Q2CONN"/>
  <property name="hostName" value="localhost"/>
  <property name="port" value="1414"/>
  <property name="transportType" value="CLIENT"/>
 </activation-config>
</jms-jca-provider>
```

8. **標準の JMS プロバイダーの作成**
   標準の JMS プロバイダー設定の作成設定は以下のようになります。

```
<jms-provider connection-factory="MyAppConnectionFactory" jndi-
URL="localhost:1414/CHAN1" jndi-context-
factory="com.ibm.mq.jms.context.WMQInitialContextFactory" name="JMS">
 <jms-bus busid="quickstartGwChannel2">
```

```
 <jms-message-filter dest-name="QUEUE3_JMS" dest-type="QUEUE"/>
 </jms-bus>
</jms-provider>
```

注記

標準プロバイダーでは、アダプターも <activation-config> 設定もありません。標準プロバイダーで定義された宛先からメッセージを受信するリスナーは、JCA Adapter Inflow を使用してメッセージを取得しません。代わりに、JNDI を使用して宛先を検索し、メッセージを取得する必要があります。つまり、Websphere MQ では、jndi-URL を常に指定する必要があります(JCA の場合は Message-Aware リスナーを提供する宛先にのみ必要であることに注意してください)。

9. **保存**
   ファイルを保存して終了します。

警告

Websphere MQ 6.0 を実行している場合は、この例外が発生する可能性があります。

```
Message: Unable to get a MQ series Queue Manager or Queue Connection.
Reason: failed to create connection javax.jms. JMSSecurityException:
MQJMS2013: invalid security authentication supplied for MQQueueManager
```

これはパーミッションの問題が原因で発生します。この問題を修正するには、JBoss Enterprise Service Bus を実行するユーザーを mqm グループに追加します。

バグの報告

12.15. JMS ルーターを使用するように IBM WEBSPHERE MQ を設定する

手順12.6 タスク

1. **JMSRouter 設定の編集**
   テキストエディターで JMSRouter の設定ファイルを開き、**jboss-esb.xml** の設定を変更します。以下は、メッセージを Websphere MQ にルーティングするための JMSRouter 設定の例です。

```
<action class="org.jboss.soa.esb.actions.routing.JMSRouter"
name="routeToORDERSQueue">
 <property name="jndi-context-factory"
value="com.ibm.mq.jms.context.WMQInitialContextFactory"/>
 <property name="jndi-URL" value="wmqserver:1414/CHANX"/>
 <property name="connection-factory" value="WMQConnectionFactory"/>
 <property name="jndiName" value="ORDERS"/>
</action>
```

2. **保存**
   ファイルを保存して終了します。

バグの報告

12.16. IBM WEBSPHERE MQ が拡張トランザクションクライアントを使用するように設定する

前提条件

- IBM からの Extended Transactional Client バンドル（デフォルトの Websphere MQ インストールの一部ではない）は、アプリケーションサーバーと外部クライアントアプリケーションのクラスパスにインストールする必要があります。
- XA 接続ファクトリーは WMQ JNDI 名前空間に設定する必要があります。を参照してください http://publib.boulder.ibm.com/infocenter/wasinfo/v6r0/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/umj_pjcfm.html。(XA 接続ファクトリーを設定したら、JNDI 接続プロパティーの connection-factory プロパティーで JNDI 名を使用して参照できます。)

手順12.7 タスク

1. **jms-jca-provider 設定の編集**
   テキストエディターで **jboss-esb.xml** ファイルを編集します。jms-jca-provider 設定がどのように拡張トランザクションクライアント設定を検索するかを示します。（この例では、WMQ JNDI 名前空間で設定された XA 接続ファクトリーは WMQXAConnectionFactory と呼ばれます。）

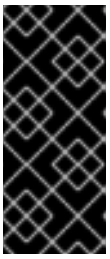
注記

インフロー関連の設定（アダプター、<activation-config> など）は、拡張クライアント設定とは関係がないため、意図的に省略されました。

```
<jms-jca-provider name="WMQ" connection-factory="WMQXAConnectionFactory"
 jndi-URL="wmqserver:1414/CHANXA_SEND"
 jndi-context-factory="com.ibm.mq.jms.context.WMQInitialContextFactory">
  <property name="max-xa-sessions-per-connection" value="1" />
   <jms-bus busid="ordersGwChannel">
   <jms-message-filter dest-type="QUEUE" dest-name="ORDERS" transacted="true"/>
   </jms-bus>
    <activation-config>
     <!--
     Used by inflow... not relevant to client
     See section on JMS and JCA.
     -->
  </activation-config>
</jms-jca-provider>
```

2. **JMSRouter で使用する設定**
   以下の例は、JMSRouter で同じ XA 接続ファクトリーを使用する方法を示しています。

```
<action name="routeToORDERSQueue"
class="org.jboss.soa.esb.actions.routing.JMSRouter">
<property name="jndi-context-factory"
value="com.ibm.mq.jms.context.WMQInitialContextFactory"/>
<property name="jndi-URL" value="wmqserver:1414/CHANXA_SEND"/>
<property name="connection-factory" value="WMQXAConnectionFactory"/>
<property name="jndiName" value="ORDERS"/>
<property name="max-xa-sessions-per-connection" value="1"/>
<!-- etc... -->
</action>
```



重要

Websphere MQ と XA に関する重要な点は、同じ Queue Manager Channel でメッセージの取得と配置の両方をサポートしていないことです。このため、XA トランザクションのコンテキストでメッセージを Websphere MQ 宛先に取得または配置するコンポーネントごとに専用チャネルを設定することが推奨されます。

3. **jboss-esb.xml** の max-xa-sessions-per-connection プロパティーを **1** に設定します。

4. **保存**
ファイルを保存して終了します。

5. Websphere MQ 宛先にメッセージを取得または送信するコンポーネントごとに、専用のキューマネージャーチャネルを設定します。

6. 各 Websphere MQ 宛先の GET および PUT プロパティーが **Inhibit** に設定されていないことを確認します。（これは、宛先が作成され、MROS がメッセージを取得または送信できない場合に発生します。）

バグの報告

12.17. PLAIN JMS 対話を使用するように IBM WEBSPHERE MQ を設定する

前提条件

- JBoss Enterprise SOA Platform の JMS プロバイダーとして事前設定された IBM Websphere MQ

手順12.8 タスク

1. **bin** ディレクトリーに移動します(**cd /opt/mqm/java/bin**)。

2. **JMSAdmin** を起動します。

3. このプログラムを使用して JNDI 設定を管理し、Plain JMS 対話のために ConnectionFactory を取得します。

注記

JMSAdmin の使用に
http://publib.boulder.ibm.com/infocenter/wmqv7/v7r0/index.jsp 関する詳細
は、を参照してください。

バグの報告

12.18. IBM WEBSPHERE MQ インストールの検証

前提条件

- JBoss Enterprise SOA Platform の JMS プロバイダーとして事前設定された IBM Websphere MQ

手順12.9 タスク

1. QM1 というキューマネージャーを作成します。**crtmqm -q QM1**
2. runmqsc: **runmqsc QM1**を起動します。
3. プロンプトが表示されたら、以下の設定を入力します。

```
define channel (CH.1) chltype (RCVR) trptype (TCP)
start channel (CH.1)
define qlocal (Q1)
define listener(QM1.LISTENER) trptype(TCP) port(30001) ipaddr(10.12.58.110)
start listener (QM1.LISTENER)
(end)
```

4. プログラムを終了します。
5. **com.ibm.mqetclient.jar** を Websphere MQ インストールからサーバーにコピーします： **cp com.ibm.mqetclient.jar SOA_ROOT/jboss-as/PROFILE/lib/**
6. Wmq.jmsra.ivt.ear を Websphere MQ インストールからサーバーにコピー： **cp wmq.jmsra.ivt.ear SOA_ROOT/jboss-as/server/PROFILE/deploy**
7. Wmq.jmsra.rar を Websphere MQ インストールからサーバーにコピーします： **cp wmq.jmsra.rar SOA_ROOT/jboss-as/server/PROFILE/deploy**
8. テキストエディターでデータソースファイルを作成します(IP アドレスを WSMQ サーバーのものに変更してください): **vi SOA_ROOT/jboss-as/server/PROFILE/deploy/wsmq-ds.xml**

   以下は、設定例です。

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



<connection-factories>
  <!-- connection factory definition -->
  <tx-connection-factory>
```

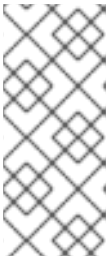
```
    <jndi-name>IVTCF</jndi-name>
    <xa-transaction />
    <rar-name>wmq.jmsra.rar</rar-name>


    <connection-definition>
      javax.jms.ConnectionFactory
    </connection-definition>


    <config-property name="channel"
type="java.lang.String">SYSTEM.DEF.SVRCONN</config-property>
    <config-property name="hostName" type="java.lang.String">
      10.12.58.110
    </config-property>
    <config-property name="username" type="java.lang.String">mqm</config-property>
    <config-property name="password" type="java.lang.String">mqm</config-property>
    <config-property name="port" type="java.lang.String">30001</config-property>
    <config-property name="queueManager" type="java.lang.String">QM1</config-property>
    <config-property name="transportType" type="java.lang.String">CLIENT</config-property>


    <security-domain-and-application>JmsXARealm</security-domain-and-application>
  </tx-connection-factory>


  <!-- admin object definition -->
  <mbean code="org.jboss.resource.deployment.AdminObject"
     name="jca.wmq:name=ivtqueue">


    <attribute name="JNDIName">
      IVTQueue
    </attribute>
    <depends optional-attribute-name="RARName">
      jboss.jca:service=RARDeployment,name='wmq.jmsra.rar'
    </depends>
    <attribute name="Type">javax.jms.Queue</attribute>


    <attribute name="Properties">
      baseQueueManagerName=QM1
      baseQueueName=Q1
    </attribute>
  </mbean>
</connection-factories>
```

9. ファイルを と して保存し、テキストエディターを終了します。

10. Web ブラウザーを起動し、にアクセスし http://localhost:8080/WMQ_IVT/ ます。

    これにより、IVT ("Install Verification Test")が起動します。

11. 通常のテストとトランザクションテストを実行します。



注記

詳細は、を参照して http://www.ibm.com/developerworks/websphere/library/techarticles/0710_ritchie/0710_ritchie.html ください。

バグの報告

12.19. IBM WEBSPHERE MQ JAVA MESSAGE SERVICE PROVIDER の診断トレース機能

IBM WebSphere MQ Java Message Service コンポーネントには、メッセージ追跡機能があります。これは、問題を診断しようとするときに役立ちます。

バグの報告

12.20. IBM WEBSPHERE MQ JCA アダプターの診断トレースを有効にする

診断トレースは、リソースアダプターのプロパティーとして、または Java 仮想マシンのシステムプロパティーで設定できます。JBoss ESB/Application では、Red Hat は JVM システムプロパティーアプローチを使用することを推奨します。

ただし、/run.sh シェルスクリプトを使用して起動されたシステムでは、次の方法を使用する必要があります。

手順12.10 タスク

1. **run.conf ファイルを開く**

   テキストエディターでファイルを開きます: **vi SOA_ROOT/jboss-as/bin/run.conf**

2. **run.conf ファイルを編集する**

   ファイルの末尾に次の行を追加します。

   ```
   # Settings to enable WebSphere MQ resource adapter trace
   JAVA_OPTS="$JAVA_OPTS -DtraceEnabled=true -DtraceDestination=wmq_jca.trc
   -DtraceLevel=10 -DlogWriterEnabled=false"
   ```

3. **クライアントログを有効にする**

   引き続きテキストエディターで、**MQJMS_TRACE_LEVEL** プロパティーを設定します。

   ```
   # Settings to enable WebSphere MQ resource adapter and client trace
   JAVA_OPTS="$JAVA_OPTS -DtraceEnabled=true -DtraceDestination=wmq_jca.trc
   -DtraceLevel=10 -DlogWriterEnabled=false -DMQJMS_TRACE_LEVEL=base"
   ```

4. **保存**

   ファイルを保存して終了します。

バグの報告

12.21. IBM WEBSPHERE MQ JAVA クライアントの診断トレースを有効にする

手順12.11 タスク

- **enableTrace 静的メソッドを呼び出す**

  com.ibm.mq.MQEnvironment の enableTrace 静的メソッドを呼び出します。

バグの報告

12.22. JAVA MESSAGE SERVICE PROVIDER として使用する RED HAT ENTERPRISE (MRG)メッセージングの設定

手順12.12 タスク

1. **設定の編集**

   テキストエディターで設定ファイル(vi SOA_ROOT/jboss-as/server/PROFILE/deploy/jbossesb.esb/META-INF/jboss-esb.xml)を開きます。

2. 以下のパラメーターを追加します。

```
<property name="jndi-prefixes" value="connectionFactory. , destination"/>
<property name="jndi-connection-factory"
value="org.apache.qpid.jndi.PropertiesFileInitialContextFactory"/>
<property name="connectionFactory.qpidConnectionFactory" value="amqp://
guest:guest@clientid/virtualHost?brokerlist='tcp://localhost:5672'"/>
<property name="destination.[queueName]Queue" value="direct://amq.direct//[queueName]?
routingkey=[routingkeyname]"/>
```

3. **保存**

   ファイルを保存して終了します。

4. **問い合わせファイルのクラスパスへの追加**

   これらの .JAR ファイルをクラスパスに追加します。

   - qpid-common-0.6.jar
   - qpid-client-0.6.jar

バグの報告

12.23. JAVA MESSAGE SERVICE PROVIDER として使用する TIBCO ENTERPRISE MESSAGE SERVICE の設定

手順12.13 タスク

1. **設定の編集**

   テキストエディターで設定ファイルを開き(vi SOA_ROOT/jboss-as/server/PROFILE/deploy/jbossesb.esb/META-INF/jboss-esb.xml)、以下のパラメーターを設定します。

```
jndi-URL="tcp://localhost:7222”
jndi-context-factory=”com.tibco.tibjms.naming.TibjmsInitialContextFactory"
connection-factory="QueueConnectionFactory"
destination-type="queue"
destination-name="myqueue"
```

2. **保存**

   ファイルを保存して終了します。

3. **問い合わせファイルのクラスパスへの追加**

   これらの .JAR ファイルをクラスパスに追加します(tibco/ems/clients/java ディレクトリーにあります)。

- jaxp.jar
- jndi.jar
- tibcrypt.jar
- tibjmsapps.jar
- tibrvjms.jar
- jms.jar
- jta-spec1_0_1.jar
- tibjmsadmin.jar
- tibjms.jar

バグの報告

12.24. SOA SERVICE MANAGER レジストリーを使用するための JBOSS の設定

1. 設定を設定するには、プロパティーファイル vi SOA_ROOT/jboss-as/server/PROFILE/deploy/jbossesb.sar/jbossesb-properties.xmlを開きます。

2. 以下の行を探します。

```
<property name="org.jboss.soa.esb.registry.queryManagerURI"
value="org.apache.juddi.registry.local.InquiryService#inquire"/>
<property name="org.jboss.soa.esb.registry.lifeCycleManagerURI"
value="org.apache.juddi.registry.local.PublishService#publish"/>
```

以下のように変更します。

```
<property name="org.jboss.soa.esb.registry.queryManagerURI" value="http://<server-name>:9901/uddi/inquiry_v2"/>
<property name="org.jboss.soa.esb.registry.lifeCycleManagerURI"
value="http://<server-name>:9901/uddi/publish_v2"/>
<property name="org.jboss.soa.esb.registry.uddi.maxRows" value="100"/>
```

3. SOA Workbench のインストールの URL に一致するように queryManagerURI および lifeCycleManagerURI を編集します。

```
<property name="org.jboss.soa.esb.registry.uddi.maxRows" value="100"/>
<property name="org.jboss.soa.esb.registry.queryManagerURI" value="http://<server-name>:9901/uddi/inquiry_v2"/>
<property name="org.jboss.soa.esb.registry.lifeCycleManagerURI"
value="http://<server-name>:9901/uddi/publish_v2"/>
```

4. 以下の行を見つけます。

```
<property name="org.jboss.soa.esb.scout.proxy.transportClass"
value="org.apache.ws.scout.transport.LocalTransport"/>
```

これを以下のように変更します。

```
<property name="org.jboss.soa.esb.scout.proxy.transportClass"
value="org.apache.ws.scout.transport.AxisTransport"/>
```

5. jbossesb-properties.xml 内で、使用する SOA Workbench のユーザー名とパスワードに一致するように変更します。

```
<property name="org.jboss.soa.esb.registry.user" value="administrator"/>
<property name="org.jboss.soa.esb.registry.password" value="password"/>
```

6. ファイルを保存して終了します。

バグの報告

12.25. 軸および COMMON-DISCOVERY ファイルの追加

1. axis-bin-1_4.tar.gz を から http://ws.apache.org/axis/ ダウンロードし、axis-1.4.jar および commons-discovery-0.2.jar （またはそれ以降）を展開します。

2. この 2 つのファイルを lib ディレクトリーにコピーします(cp *jar SOA_ROOT/jboss-as/server/PROFILE/lib)。

バグの報告

12.26. SOA WORKBENCH の設定

1. ゲストユーザーにゲストロールを適用して匿名アクセスを許可するには、組織ツリーに移動し、SOA Workbench 階層の最上位のレジストリーノードをクリックします。

注記

SOA Workbench は、サードパーティーの UDDI レジストリープロバイダーとして機能する SOA Service Manager Registry のセクションです。

2. 2 次レベルのセキュリティータブ(Workbench 階層内)をクリックします。

3. Role Memberships の下にある Guest の Role の管理をクリックします。

4. ユーザー guest を検索します。

5. guest の横にあるチェックボックスを選択し、Apply をクリックします。

バグの報告

12.27. ワークフローの設定

1. 作成時にサービスが公開されるようにワークフローを変更するには、設定 タブをクリックし、ワークフローを選択します。

2. ワークフローの定義を表示してローカルディスクに保存し、@create の function type 行を <function type="publish" /> に変更し、step 属性を "100" に変更します。

3. 保存します。

4. ワークフローを、保存したローカルコピーに更新します。

バグの報告

12.28. サービスの管理

- 検索オプションを使用してサービスを検索します。この機能により、説明、現在の状態、そのバージョン番号、生成されたアラートの数などの各サービスに関する情報も表示されます。サービスは SOA Service Workbench に登録されるため、グラフィカルユーザーインターフェイスから直接管理します。

バグの報告

12.29. 診断

- ワークフロー の設定および更新時にエラーが発生した場合は、（以下のように表示される）設定が元の設定から変更されたことを確認して、ワークフローが正しく変更されたことを確認できます。

```
<action id="1" name="@create">
 <results>
  <unconditional-result old-status="Created" status="Draft" step="100"
owner="${caller}"/>
 </results>
 <post-functions>
  <function type="setLifecycleStage">
   <arg name="stage">Design</arg>
   </function>
 </post-functions>
</action>
```

代わりに、以下のようになります。

```
<action id="1" name="@create">
 <results>
  <unconditional-result old-status="Created" status="Draft" step="100"
owner="${caller}"/>
  </results>
  <post-functions>
   <function type="publish">
  </function>
 </post-functions>
</action>
```

バグの報告

第13章 他の JAVA MESSAGE SERVICE PROVIDER 設定オプション

13.1. JAVA MESSAGE SERVICE リスナーおよびゲートウェイの設定

以下の手順に従って、JMS リスナーとゲートウェイがキューとトピックをリッスンするようにします。jboss-esb.xml 設定ファイル：

手順13.1 タスク

1. 設定ファイルを開きます。

   ```
   vi jboss-esb.xml
   ```

2. ファイルの編集

   以下のパラメーターを指定します。

   - jndi-URL
   - jndi-context-factory
   - jndi-pkg-prefix
   - connection-factory
   - destination-type
   - destination-name

   重要

   JMS プロバイダーのクライアントの JAR ファイルをクラスパスに含めるようにしてください。

3. 保存

ファイルを保存し、テキストエディターを編集します。

バグの報告

13.2. JMSCONNECTIONPOOL

名前が示すように、JmsConnectionPool は Java Message Service セッションをプールします。これは、リスナー、courier、およびルーターを含むすべての JMS ベースのコンポーネントによって使用されます。

一部の Java Message Service プロバイダーは、接続ごとに許可される JMS セッションの数を制限します。その結果、JBoss Enterprise Service Bus の JMS コンポーネントは、1 つの JmsConnectionPool インスタンスによって管理される各 JMS 接続から作成されるセッションの最大数を制御するメカニズムを利用します。

バグの報告

13.3. 接続ごとの最大セッション数の設定

手順13.2 タスク

1. 関連する設定ファイルを開きます。

   テキストエディターで問題の JNDI 設定ファイルのコンポーネントを開きます。

2. 関連するプロパティーを設定します。

   max-sessions-per-connection と max-xa-sessions-per-connection の 2 つのプロパティーを編集できます。

   JMS プロバイダー設定ファイルでパラメーターを汎用プロパティーとして設定する必要があります。このサンプルコードは、その方法を示しています。

```
<jms-provider ...>
 <property name="max-sessions-per-connection" value="5" />
 <property name="max-xa-sessions-per-connection" value="1" />
 <!-- And add providers.... -->
</jms-provider>
```

注記

これらのパラメーターのいずれかを設定しない場合、JmsConnectionPool は単一の JMS 接続を作成し、そこからすべてのセッションを取得します。

バグの報告

13.4. JNDI 設定ファイルのプロパティー

表13.1 JNDI 設定ファイルのプロパティー

プロパティー	Description
max-sessions-per-connection	これは、接続ごとに許可される最大セッション数です（拡張(XA)と非 XA セッションインスタンスの両方）。 デフォルト値は、JmsConnectionPool 全体で許可される JMS セッションの最大数と同じです(**jbossesb-properties.xml** ファイルの設定を変更して設定できます)。デフォルトは 20 です。）
max-xa-sessions-per-connection	これは、接続ごとに許可される XA セッションの最大数です。デフォルト値は、max-sessions-per-connection に設定されたものと同じです。

バグの報告

13.5. JMS-JCA-PROVIDER

jms-jca-provider は、JMS Java Connector Architecture (JCA)の専用設定です。jms-jca-provider は、JMS プロバイダーに接続する 2 つの方法で JMS プロバイダーに接続することで、JCA メッセージインフローを容易にします。これらはゲートウェイであり、各 JMS メッセージリスナーです。

バグの報告

13.6. JMS-JCA-PROVIDER の設定

手順13.3 タスク

1. 設定ファイルの編集

   テキストエディターで設定ファイルを開き、設定を変更します。以下に例を示します。

```
<jms-jca-provider connection-factory="XAConnectionFactory"
name="JBossMessaging">
 <jms-bus busid="ordersGwChannel">
  <jms-message-filter dest-name="queue/orders" dest-type="QUEUE"
transacted="true"/>
 </jms-bus>
  <activation-config>
   <property name="dLQMaxResent" value="5"/>
  </activation-config>
</jms-jca-provider>
```

2. 保存

   ファイルを保存して終了します。

バグの報告

13.7. JMS-JCA-PROVIDER 設定オプション

jms-jca-provider 設定は、システムの 2 つの異なる部分を設定します。これらは以下になります。

1. JCA インフロー（サーバー側にあります）

   これらのアイテムは、以下の項目に対して設定できます。

   - JCA アダプター名。これは、<jms-jca-provider>要素の属性として設定されます。
   - JCA プロバイダーアダプター JNDI。これは、<jms-jca-provider> 要素の属性として設定されます。
   - JCA エンドポイントクラス。これは、<jms-jca-provider> 要素の属性として設定されます。
   -

トランザクションフラグ。これは、<jms-jca-provider> 要素の属性として設定されます。

- メッセージタイプ。これは、<jms-jca-provider> 要素の属性として設定されます。
- JCA ブリッジ。これは、<jms-jca-provider> 要素の属性として設定されます。
- JCA アダプターのアクティベーション設定。これは、<jms-jca-provider> 内の <activation-config> 要素で設定されます。

注記

JCA アクティベーション設定は、<jms-jca-provider> 要素内にネストされている <jms-bus> および <jms-message-filter> から JCA Inflow の一部の設定プロパティーも展開します。これには、宛先タイプ、宛先名、およびメッセージセレクターが含まれます。

2. JMS 接続の詳細（クライアント側にある）は、メッセージを JMS JCA インフロー（ゲートウェイまたは アウェア対応リスナー）に配信するために使用されます。

これらの設定は、JMS JCA インフローに接続するクライアントのエンドポイント参照を生成するために使用されます。内容は以下の通りです。

- クライアントによって使用される JMS 接続ファクトリー。これは、<jms-jca-provider> 要素の属性として設定されます。
- JMS プロバイダーに接続するためにクライアントによって使用される JMS JNDI プロパティー。これらは、<jms-jca-provider> 要素(jndi-*)の属性として設定されます。
- JMS バスエンドポイントの宛先設定。これらは、<jms-jca-provider> 要素の <jms-bus> 内の <jms-message-filter> 要素で設定されます。

上記の JMS クライアント接続設定が JCA 管理リソース経由で接続するように設定されていないことを確認してください。JMS クライアント接続設定は、JCA 管理リソースではなく、JMS プロバイダーに直接接続する必要があります。この情報は、JBoss Enterprise SOA Platform の ServiceInvoker クラスおよびシステムの接続プール機能を使用するその他のコンポーネントで使用さ

れるエンドポイント参照に組み込まれるために重要になります。(JCA が管理する接続をプールするためにこの使用を避けることが重要です。)

これらは、接続プール機能を使用するコンポーネントです。

1. 静的ルーターやコンテンツベースルーターなどの ServiceInvoker クラスを使用するルーティングアクション。

2. メッセージを新しい対応 JMS リスナーによって提供されるアクションパイプラインに配信するゲートウェイリスナー。これにより、JCA JMS プロバイダーが使用されます。(これらのゲートウェイリスナーのほとんどは、ServiceInvoker クラスを使用してメッセージを配信します。)

3. JMS ルーター。

これらのコンポーネントは、ローカルの JCA 管理リソースを介してではなく、JMS プロバイダーに直接接続するようにしてください。

バグの報告

13.8. JNDI 拡張プロパティー

デフォルトでは、JNDI Java Message Service のリソース取得設定は、java.naming で始まるすべてのプロパティーを継承します。

一部の Java Message Service プロバイダーは、異なる命名接頭辞を指定します。これらのサポートをサポートするために、JBoss Enterprise SOA Platform では各プロバイダーのカスタムプロパティー接頭辞を指定できます。

バグの報告

13.9. JNDI 拡張プロパティーの設定

手順13.4 タスク

1. **適切な JMS プロバイダー要素の jndi-prefixes を追加します。**

   テキストエディターで設定ファイルを開き、使用する追加接頭辞のコンマ区切りリストを使用して、適切な JMS プロバイダー要素の jndi-prefixes を追加します。

```
<jms-provider name="JMS" connection-factory="ConnectionFactory">
<property name="jndi-prefixes" value="test.prefix." />
<property name="test.prefix.extension1" value="extension1" />
<property name="test.prefix.extension2" value="extension2" />
</jms-provider>
```

注記

同じ場所に extensions プロパティーを設定することもできます。

2. **保存**

   ファイルを保存して終了します。

バグの報告

第14章 JBPM の設定

14.1. JBPM

JBoss Business Process Manager (jBPM) は、ユーザーがビジネスプロセスと言語を制御できるようにするワークフロー管理ツールです。jBPM 3 がデフォルトとして使用されます。

バグの報告

14.2. JBPM 3 と JBOSS ENTERPRISE SOA PLATFORM の統合

JBoss Enterprise SOA Platform が jBPM 3 と統合するデフォルトの方法は、JMS/JCA-Inflow 経由です。

```
<service name="message" factory="org.jbpm.msg.db.DbMessageServiceFactory" />
<service name="scheduler" factory="org.jbpm.scheduler.db.DbSchedulerServiceFactory" />

<bean name="jbpm.job.executor" class="org.jbpm.job.executor.JobExecutor">
   ...
</bean>
```

JMS/JCA インフローは、JBoss Enterprise SOA Platform でサポートされる唯一の jBPM/ESB 統合設定です。jbpm.esb は、JBoss Messaging JMS プロバイダーのみをサポートします。

重要

jBPM エンタープライズモジュールはサポートされていません。（これは、JMS および EJB タイマーに基づくメッセージおよびスケジューラーサービスで設定されます。）

simulation モジュールもサポートされていません。（これはプロセス分析および最適化ツールです。）

バグの報告

14.3. JBPM 統合サービスの変更

手順14.1 タスク

1. jbpm.esb アーカイブの 設定 ディレクトリーに移動します。

2. 使用するサービスの設定ファイルを見つけます。

3. アクティブな設定を置き換えるには、jbpm.esb ディレクトリーにあるものを削除し、使用する設定ファイルに置き換えます（名前から .config 接尾辞を削除します）。

バグの報告

14.4. JBPM ジョブエグゼキューター

jBPM ジョブエグゼキューターは Web リスナーです。これは、Enterprise Service Bus が実行にデプロイされていない場合にのみサポートされます。

バグの報告

14.5. JBPM ジョブエグゼキューターの設定

手順14.2 タスク

1. Web アプリケーションで jBPM Job Executor を使用するには、テキストエディターでアプリケーションの web.xml ファイルを開き、以下のコードを追加します。

```
<!-- Job executor launcher -->
 <listener>
  <description>
   Starts the job executor on initialization and stops it on destruction.
  </description>
   <listener-class>org.jbpm.web.JobExecutorLauncher</listener-class>
 </listener>
<!-- Job executor launcher -->
```

2. ファイルを保存して終了します。

バグの報告

14.6. リモート JAVA メッセージサービスプロバイダーの使用

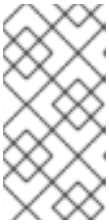
JMS ベースのメッセージとスケジューリングサービスがリモート JMS プロバイダーを参照する場合は、ローカル JNDI のプロバイダーアダプター設定を追加する必要があります。

注記

プロバイダーアダプターは、標準のアプリケーションサーバーの JCA Inflow 設定で使用される JMSProviderAdapter のインスタンスです。

手順14.3 タスク

1. テキストエディターで service.xml ファイルを開き、MBean を追加します。



注記

これは jBPM に固有にするか(jbpm-service.xml ファイルに追加)、他の JCA インフロー 設定と共有できます。

以下のコード例では、このアドレスで実行されるリモート JMS プロバイダーを参照するプロバイダーアダプターを作成し 192.168.1.1:1099 ます。

```
<mbean code="org.jboss.jms.jndi.JMSProviderLoader"
name="jboss.messaging:service=JMSProviderLoader,name=RemoteProviderLoader">
  <attribute name="ProviderName">RemoteProviderAdapter</attribute>
  <attribute
name="ProviderAdapterClass">org.jboss.jms.jndi.JNDIProviderAdapter</attribute>
  <attribute name="QueueFactoryRef">XAQueueConnectionFactory</attribute>
  <attribute name="TopicFactoryRef">XATopicConnectionFactory</attribute>
  <attribute name="Properties">
    java.naming.provider.url=192.168.1.1:1099
  </attribute>
</mbean>
```

注記

Properties 属性に設定された値を使用して、JNDI の InitialContext を作成します。

2. **JMSMessageServiceFactory ファイル内でプロバイダーアダプターへの参照を提供します。テキストエディターで vi JMSMessageServiceFactoryを開きます。**

3. **providerAdapterJNDI フィールドに JNDI の場所を追加します。**

```
<service name="message">
  <factory>
    <bean
class="org.jboss.soa.esb.services.jbpm.integration.msg.JmsMessageServiceFactory">
      <field name="providerAdapterJNDI"><string
value="RemoteProviderAdapter"/></field>
    </bean>
  </factory>
</service>
```

4. **ファイルを保存して終了します。**

5. **テキストエディターで JMSSchedulerServiceFactory ファイル内でプロバイダーアダプターへの参照を提供します(vi JMSSchedulerServiceFactory)。**

6. **providerAdapterJNDI フィールドに JNDI の場所を追加します。**

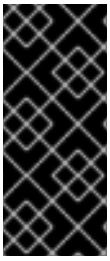
7. **ファイルを保存して終了します。**

バグの報告

第15章 高度なインストールオプション

15.1. JBOSS_HOME 環境変数

JBOSS_HOME 環境変数はオプションの .bash_profile 設定です。通常、この設定を設定する必要はありませんが、一部のスクリプトやサードパーティーソフトウェアにはこれが必要です。



重要

1 台のマシンに複数の JBoss サーバーをインストールしている場合、Red Hat は、設定が必要なスクリプト内でしか設定できない場合のみ、この設定の使用を回避することを推奨します。

バグの報告

15.2. JBOSS_HOME 環境変数の設定

手順15.1 タスク

1. Red Hat Enterprise Linux システムでの変数の追加

   vi ~./bash_profile コマンドで ファイルを開き、以下の行を追加します。

   ```
   export JBOSS_HOME=path
   ```

   (JBOSS_HOME=/path/to/ SOA_ROOT.)

2. 保存して終了します。

3. Microsoft Windows システムでの変数の追加

   Control Panel → System → Advanced → Environment Variables をクリックしてから、New をクリックします。変数名を JBOSS_HOME に、変数値を SOA_ROOT ディレクトリーに設定します。

バグの報告

15.3. ネイティブコンポーネントパッケージ

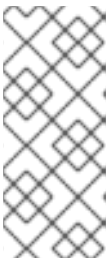
ネイティブコンポーネントパッケージは、Web サーバーのネイティブオペレーティングシステムのコンポーネントおよびコネクターを組み込む JBoss Enterprise Application Platform の任意のコンポーネントです。

バグの報告

15.4. ネイティブ JBOSS コンポーネントのインストール

手順15.2 タスク

- 手順は、JBoss Enterprise Application Platform の インストールガイド を参照してください。



注記

このガイドは、ファイルを jboss-eap-5.1 ディレクトリーにインストールするように指示します。対象のディレクトリーを参照するたびに SOA_ROOT を置き換えます。

バグの報告

15.5. MRG-M

2009-M は、Red Hat の Messaging/Realtime/Grid オファリングのメッセージングコンポーネントです。

バグの報告

15.6. チュークス-M のインストール

手順15.3 タスク

1. パッケージのダウンロード

administrators JCA Adapter を Red Hat カスタマーポータル からダウンロードします。SOA_ROOT/jboss-as/server/PROFILE/deploy ディレクトリーに保存します。

2. キューの作成

キューを作成するには、qpid-config add queue コマンドを使用します。

3. 設定ファイルの編集

設定ファイル vi qpid-jca-ds.xml を開きます。ここでは、3 つの JCA 管理オブジェクトがあります。受信(MRG-ESB_GW)キューの宛先アドレスと、アウトバウンドキューのルーティングキー(MRG-ESB_RESP)キューおよび routingKey を設定する必要があります。

4. 保存

ファイルを保存して終了します。

注記

QpidConnectionFactory は、メッセージの送信に使用される接続ファクトリーで、JMSRouter アクションで使用されます。他の 2 つの CF 管理オブジェクトは、この 2 つの CF 管理オブジェクトでは機能しないので、使用を試みないでください。

バグの報告

15.7. ファイル転送プロトコル

ファイル転送プロトコル(FTP)は、ネットワーク全体でファイルを送信するための標準プロトコルです。テキストおよびバイナリーファイルを送信できます。

バグの報告

15.8. ファイル転送プロトコルと JBOSS ENTERPRISE SOA PLATFORM

JBoss Enterprise SOA Platform は、FTP 経由で送信する前に同じファイル名を持つように、すべてのファイルの名前を変更します。（これにより、ソフトウェアが他の最後に名前が変更されるまで、個々のファイルが処理されないようにします。） ただし、一部の FTP サーバーは、書き込み後にファイルにロックを保持するため、名前変更プロセスが実行されなくなります。この問題が発生した場合、

ソフトウェアは事前定義済みの試行回数（デフォルトは 10）を作成して、ファイルの名前を変更します。最大試行回数を超えた後もこれを実行できない場合は、エラーメッセージが生成されます。

バグの報告

15.9. ファイル転送プロトコルの設定

手順15.4 タスク

1. **プロパティーの編集**

   テキストエディターでグローバル設定ファイル(vi SOA_ROOT/jboss-as/server/PROFILE/deployers/esb.deployer/jbossesb-properties.xml)を開きます。org.jboss.soa.esb.ftp.renameretry を検索し、この設定の値を変更します。以下に例を示します。

   ```
   <property name="org.jboss.soa.esb.ftp.renameretry" value="10"/>
   ```

2. **保存**

   ファイルを保存して終了します。

結果

これにより、再試行の回数が変更されます。

注記

このファイルの設定を変更すると、グローバルな効果があります。インスタンスごとにこれを行うには、代わりに特定のエンドポイント参照の設定を変更します。

バグの報告

パート V. セキュリティー

第16章 システムの保護

16.1. SECURITY ASSERTION MARKUP LANGUAGE (SAML)

Security Assertion Markup Language (SAML) は、XML でコンパイルされたフレームワークです。サービス間で情報を安全に受け渡すために使用されます。これは、認証と識別に最も一般的に使用されます。SAML を使用すると、ユーザーは 1 回ログインするだけで、SAML によって ID を検証できるため、認証情報を繰り返し再入力する必要がなくなります。

JBoss ESB は、JAAS ログインモジュールを介して PicketLink Project が提供する SAML を使用します。SAML セキュリティートークンを割り当てて検証する機能をユーザーに提供します。

バグの報告

16.2. SAML セキュリティートークンの発行

手順16.1 タスク

1. org.picketlink.identity.federation.core.wstrust.auth.STSIssuingLoginModule にあるログインモジュール (LM) を取得します。

2. LM の設定ファイルを開きます。

3. 次のコードを入力し、使用するサービスの名前を挿入します。

```
<application-policy name="saml-issue-token">
    <authentication>
        <login-module
code="org.picketlink.identity.federation.core.wstrust.auth.STSIssuingLoginModule"
flag="required">
            <module-option name="configFile">picketlink-sts-client.properties</module-
option>
            <module-option
name="endpointURI">http://security_saml/goodbyeworld</module-option>
        </login-module>
        <login-module
code="org.picketlink.identity.federation.core.wstrust.auth.STSValidatingLoginModule"
flag="required">
            <module-option name="configFile">picketlink-sts-client.properties</module-
option>
```

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

この設定では、スタック LM を使用します。最初の LM からのセキュリティートークンは、セキュリティートークンを検証する 2 番目の LM によって後で使用されます。セキュリティートークンの検証のみが必要な場合があるため、このために 2 つの個別の LM を使用すると便利です。

4. picketlink-sts-client プロパティーを指定します。

```
serviceName=PicketLinkSTS
portName=PicketLinkSTSPort
endpointAddress=http://localhost:8080/picketlink-sts/PicketLinkSTS
username=admin
password=admin
```

注記

このファイルのユーザー名とパスワードは、STSValidatingLoginModule でのみ使用されます。ユーザー名とパスワードは、スタックされるか、コールバックによって提供される場合もあります。

5. JBossESB でこの LM を使用するには、サーバーの login-config.xml を上記の application-policy で更新する必要があります。また、この LM を使用する場所を ESB サービスで指定する必要があります。

たとえば、これは jboss-esb.xml で設定する方法です:

```
<service category="SamlSecurityQuickstart" name="issueTokenService"
invmScope="GLOBAL"
   description="This service demonstrates how a service can be configured to issue
and validate a security token">



   <security moduleName="saml-issue-token"
callbackHandler="org.jboss.soa.esb.services.security.auth.login.JBossSTSIssueCallb
ackHandler">
      <!-- disable the security context timeout so that our security context is re-
evaluated -->
      <property name="org.jboss.soa.esb.services.security.contextTimeout"
value="0"/>
```

```
    </security>
    ...
</service>
```

指定されている callbackHandler は ESB に固有です。これは、セキュリティートークンを発行する必要があるユーザーのユーザー名とパスワードを取得するために、ESB 内の認証要求にアクセスする必要があるためです。

バグの報告

16.3. SAML セキュリティートークンの検証

手順16.2 タスク

1. org.picketlink.identity.federation.core.wstrust.auth.STSIssuingLoginModule からログインモジュール (LM) を開きます。

2. 次の例に示すように、プロパティーファイルを設定します。

```
      <application-policy name="saml-validate-token">
   <authentication>
      <login-module
code="org.picketlink.identity.federation.core.wstrust.auth.STSValidatingLoginModule"
flag="required">
         <module-option name="configFile">picketlink-sts-client.properties</module-
option>
      </login-module>
   </authentication>
</application-policy>



And in jboss-esb.xml:
<service category="SamlSecurityQuickstart" name="securedSamlService"
invmScope="GLOBAL"
   description="This service demonstrates that an ESB service can be configured to
only validate a security token.">

   <security moduleName="saml-validate-token"
callbackHandler="org.jboss.soa.esb.services.security.auth.login.JBossSTSTokenCall
backHandler">
      <!-- disable the security context timeout so that our security context is re-
evaluated -->
      <property name="org.jboss.soa.esb.services.security.contextTimeout"
value="0"/>
```

```
    </security>
    ...
</service>
```

注記

指定された callbackHandler は ESB に固有です。これは、検証対象の SAML トークンを取得するために、ESB 内の認証要求にアクセスする必要があるためです。

注記

JBossESB での SAML サポートの例は security_saml クイックスタートにあります。PicketLink が提供するログインモジュールの詳細は、http://www.jboss.org/community/wiki/STSLoginModules を参照してください。

バグの報告

16.4. PICKETLINK

PicketLink IDM は、セキュリティーシステムを相互にリンクするフレームワークです。これは、ID 管理を使用して行われます。これには、ユーザー ID、グループ、および権限に関する情報が含まれています。

バグの報告

16.5. SAML と PICKETLINK の統合

- クライアントはまず、トークンサービスに WS-Trust 要求を送信して、PicketLink STS から SAML アサーションを取得する必要があります。通常、このプロセスにはクライアントの認証が含まれます。
- STS から SAML アサーションを取得した後、クライアントは、Bean で操作を呼び出す前に、EJB 要求のセキュリティーコンテキストにアサーションを含めます。
-

呼び出しを受け取ると、EJB コンテナーはアサーションを抽出し、WS-Trust メッセージを STS に送信して検証します。アサーションが STS によって有効であると見なされた場合 (および必要に応じて所有証明トークンが検証された場合)、クライアントは認証されます。

- JBoss では、SAML アサーション検証プロセスはSAML2STSLoginModule で処理されます。設定可能なファイルからプロパティーを読み取ります (configFile オプション)、これらのプロパティーに基づいて STS との通信を確立します。

- アサーションが有効な場合、Principal は、アサーションサブジェクト名を使用して作成されます。ロールアサーションにロールが含まれている場合、これらのロールも抽出され、呼び出し元のSubject に関連付けられます。

バグの報告

16.6. JBOSS ENTERPRISE SOA PLATFORM インストールの保護

はじめに

JBoss Enterprise SOA Platform は、呼び出し元の認証が成功し、呼び出し元が正しいパーミッションを持っている場合にのみサービスが実行されるように製品を設定できるという意味で、安全にすることができます。デフォルトのセキュリティー実装は JAAS に基づいています。

サービスを呼び出す方法は 2 つあります。

1. ゲートウェイ経由

2. ServiceInvoker 経由で直接

ゲートウェイオプションを使用すると、呼び出し元を認証するために必要なセキュリティー情報を取得する責任があります。これは、トランスポートから必要な情報を抽出することによって行われます。それが完了すると、暗号化されて Enterprise Service Bus に渡される認証要求を作成します。

代わりに ServiceInvoker を使用する場合、サービスを呼び出す前に認証要求を行うのは、クライアントアプリケーションの責任になります。これには、SOAP ヘッダーのセキュリティー要素から UsernameToken または BinarySecurityToken のいずれかを抽出する必要があります。

バグの報告

16.7. JAVA 認証および承認サービス (JAAS)

JAAS 1.0 API は、ユーザーの認証と承認用に設計された一連の Java パッケージで設定されています。この API は、標準の Pluggable Authentication Modules (PAM) フレームワークの Java バージョンを実装し、Java 2 プラットフォームのアクセス制御アーキテクチャーを拡張して、ユーザーベースの承認をサポートします。

JAAS は、JDK 1.3 の拡張パッケージとして最初にリリースされ、JDK 1.6 にバンドルされています。

バグの報告

16.8. JAASSECURITYSERVICE

JaasSecurityService は、JBoss Enterprise SOA Platform で使用される JAAS のデフォルトの実装です。

バグの報告

16.9. システムを保護する

手順16.3 タスク

グローバル設定ファイルをテキストエディターで開きます: vi SOA_ROOT/jboss-as/server/PROFILE/deployers/esb.deployer/jbossesb-properties.xml

1. properties name="security" を含むセクションまで下にスクロールして、システムに合わせて設定を編集します。

```
<properties name="security">
<property name="org.jboss.soa.esb.services.security.implementationClass"
value="org.jboss.internal.soa.esb.services.security.JaasSecurityService"/>
```

```
<property name="org.jboss.soa.esb.services.security.callbackHandler"
value=
"org.jboss.internal.soa.esb.services.security.UserPassCallbackHandler"/>

<property name="org.jboss.soa.esb.services.security.sealAlgorithm"
value="TripleDES"/>

<property name="org.jboss.soa.esb.services.security.sealKeySize"
value="168"/>

<property name="org.jboss.soa.esb.services.security.contextTimeout"
value="30000"/>

<property name=
"org.jboss.soa.esb.services.security.contextPropagatorImplemtationClass"
value=
"org.jboss.internal.soa.esb.services.security.JBossASContextPropagator"/>

<property name="org.jboss.soa.esb.services.security.publicKeystore"
value="/publicKeyStore"/>


<property name="org.jboss.soa.esb.services.security.publicKeystorePassword"
value="testKeystorePassword"/>

<property name="org.jboss.soa.esb.services.security.publicKeyAlias"
value="testAlias"/>

<property name="org.jboss.soa.esb.services.security.publicKeyPassword"
value="testPassword"/>

<property name="org.jboss.soa.esb.services.security.publicKeyTransformation"
value="RSA/ECB/PKCS1Padding"/>

</properties>
```

2. **ファイルを保存して終了します。**

3. **テキストエディターでログイン設定ファイルを開きます。vi SOA_ROOT/server/PROFILE/conf/login-config.xml**

4. **このファイルの設定を編集して、JAAS ログインモジュールを設定します。(事前設定されたオプションを使用するか、独自のカスタムソリューションを作成できます。)**

5. **ファイルを保存して終了します。**

バグの報告

16.10. 暗号化されたパスワードファイルの作成

手順16.4 タスク

1. conf ディレクトリーに移動します: cd SOA_ROOT/jboss-as/server/PROFILE/conf

2. 次のコマンドを実行します: java -cp ../../../lib/jbosssx.jar org.jboss.security.plugins.FilePassword welcometojboss 13 testpass esb.password

結果

暗号化されたパスワードファイルが作成されます。

バグの報告

16.11. 暗号化オプション

表16.1 暗号化オプション

オプション	説明
Salt	これは、パスワードファイルの暗号化に使用されるソルトです。(上記の例では、**welcometojboss** 文字列)
Iteration	これは反復回数です。(上記の例では、これは数値 **13** です)
Password File Name	これは、暗号化されたパスワードが保存されるファイルの名前です。上記の例では、**esb.password** 文字列です。
testpass	これはテストパスワードです。

バグの報告

16.12. 平文パスワード

クリアテキストパスワードは、パスワードのプレーンテキストバージョンです。暗号化されていないか、復号化されたばかりです。クリアテキストのパスワードは安全ではありません。

バグの報告

16.13. パスワードマスク

パスワードマスクは、パスワードに使用できる文字を決定するテンプレートです。たとえば、一部のパスワードマスクではパスワードに英数字のみを使用できるように指示されていますが、他のパスワードマスクでは！および $ 記号。一般に、特殊文字を含むパスワードはより安全であると見なされています。

バグの報告

16.14. パスワードのマスキング

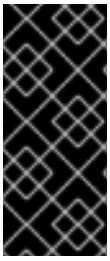
はじめに

パスワードは、リソースへのアクセスを許可された関係者のみに制限するために使用される秘密の認証トークンです。JBoss サービスがパスワードで保護されたリソースにアクセスするには、明らかにパスワードを使用できるようにする必要があります。

これは、起動時に JBoss Enterprise SOA Platform に渡されるコマンドライン引数によって実行できますが、これは本番環境では実用的ではありません。代わりに、パスワードは通常、設定ファイルに含めることで JBoss サービスで利用できるようになります。

すべての JBoss Enterprise SOA Platform 設定ファイルは安全なファイルシステムに保存し、プロセス所有者のみが読み取りできるようにする必要があります。

セキュリティーを強化するために、設定ファイルでパスワードをマスクすることもできます。このセクションでは、その方法について説明します。



重要

侵入できないセキュリティーなどありません。パスワードのマスキングも例外ではありません。侵入できないわけではありませんが、設定ファイルの簡単な検査が無効になり、パスワードを抽出するために必要な労力が増加します。

バグの報告

16.15. 平文パスワードをマスクする

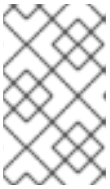
重要

この鍵ストアのパスワード暗号化手順は、1 回だけ実行してください。キーストアのパスワードの入力を間違えた場合、または後日キーストアを変更した場合は、jboss-keystore_pass.dat ファイルを削除して手順を繰り返す必要があります。キーストアを変更すると、以前に生成されたマスクされたパスワードは機能しなくなることに注意してください。

手順16.5 タスク

1. 次のコマンドを使用してキーペアを生成します: keytool -genkey -alias jboss -keyalg RSA -keysize 1024 -keystore password.keystore。次のプロンプトに従います。

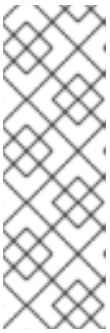
```
keytool -genkey -alias jboss -keyalg RSA -keysize 1024 -keystore
password.keystore
Enter keystore password:
Re-enter new password:
What is your first and last name?
  [Unknown]:  Bob Bobson
What is the name of your organizational unit?
  [Unknown]:  Corporate_IT
What is the name of your organization?
  [Unknown]:  XYZ
What is the name of your City or Locality?
  [Unknown]:  BRISBANE
What is the name of your State or Province?
  [Unknown]:  QLD
What is the two-letter country code for this unit?
  [Unknown]:  AU
Is CN=Bob Bobson, OU=Corporate_IT, O=XYZ, L=BRISBANE, ST=QLD, C=AU correct?
  [no]:  yes

Enter key password for jboss
 (RETURN if same as keystore password):
```



注記

キーストアとキーペアに同じパスワードを指定する必要があります。

2. chown を実行して所有権を JBoss Application Server プロセス所有者に変更し、chmod 600 password.keystore を実行してファイルの所有者だけがファイルを読み取れるようにします。



注記

プロセスの所有者は、コンソールログインアクセス権を持っていてはなりません。その場合、これらの操作は別のユーザーとして実行します。マスクされたパスワードを作成するには、キーストアへの読み取りアクセスが必要なため、キーストアファイルのアクセス許可を制限する前に、マスクされたパスワードの設定を完了しておくことをお勧めします。

3. jboss-as/bin ディレクトリーに移動します: cd SOA_ROOT/jboss-as/bin

4. Red Hat Enterprise Linux システムではコマンド ./password_tool.sh (Microsoft Windows ベースのシステムでは password_tool.bat) を使用して、パスワードツールを実行します。

5. 0 を押して 0: Encrypt Keystore Password を選択し、次に Enter を押します。

6. 上記で指定したキーストアのパスワードを入力します。

7. 暗号化の強度を高めるためにランダムな文字列を入力します。これが salt です。

8. 暗号化の強度を高めるために、イテレータカウントに整数を入力します。

9. 5: Exit を選択して終了します。

注記

パスワードツールは、次のメッセージで終了します。Keystore is null.Cannot store.これは正常なことです。

10. chown コマンドを使用して password/jboss_keystore_pass.dat ファイルの所有権をプロセス所有者に変更し、chmod 600 jboss-keystore_pass.dat を使用してその所有者のみがファイルを読み取れるようにします。

11. jboss-as/bin ディレクトリーに移動します: cd SOA_ROOT/jboss-as/bin

12. Red Hat Enterprise Linux システムではコマンド ./password_tool.sh (Microsoft Windows システムでは password_tool.bat) を使用して、パスワードツールを実行します。

13. 1 を押して 1: Specify KeyStore を選択し、Enter を押します。

14. 上で作成したキーストアへのパスを入力します。(絶対パス、または SOA_ROOT/jboss-as/bin への相対パスを指定できます。デフォルトを変更していない限り、これは SOA_ROOT/jboss-as/bin/password.keystore である必要があります。)

15. キー エイリアスを入力します。これは jboss である必要があります (高度なインストールを実行してデフォルトを変更した場合を除く)。

16. 2 を押して 2: Create Password を選択し、Enter を押します。セキュリティードメインの入力を求めるプロンプトが表示されます。画面のプロンプトに従います。

```
      /password_tool.sh
**********************************
****  JBoss Password Tool********
**********************************
Error while trying to load data:Encrypted password file not located
Maybe it does not exist and need to be created.
0: Encrypt Keystore Password 1:Specify KeyStore 2:Create Password  3: Remove a
domain 4:Enquire Domain 5:Exit
1
Enter Keystore location including the file name
password.keystore
Enter Keystore alias
jboss
0: Encrypt Keystore Password 1:Specify KeyStore 2:Create Password  3: Remove a
```

```
domain 4:Enquire Domain 5:Exit
2
Enter security domain:

default
Enter passwd:
passwordmask
Password created for domain:default
0: Encrypt Keystore Password 1:Specify KeyStore 2:Create Password  3: Remove a
domain 4:Enquire Domain 5:Exit
```

17. パスワードマスクの名前を入力します。これは、設定ファイルでパスワードマスクを識別するために使用する任意の一意の名前です。

18. マスクしたいパスワードを入力します。

19. パスワードマスクの作成プロセスを繰り返して、マスクするすべてのパスワードのマスクを作成します。

20. 5: Exit を選択してプログラムを終了します。

21. password ディレクトリーに移動します: cd SOA_ROOT/jboss-as/bin/password

バグの報告

16.16. 平文パスワードをそのパスワードマスクに置き換える

前提条件

- 既存のパスワードマスク

手順16.6 タスク

- テキストエディターを起動し、設定ファイル内のクリアテキストパスワードの各出現箇所を、そのマスクを参照する注釈に置き換えます。

これは、注釈の一般的な形式です。

```
<annotation>
@org.jboss.security.integration.password.Password(securityDomain=MASK_NAME,
methodName=setPROPERTY_NAME)
</annotation>
```

具体的な例として、JBoss Messaging パスワードはサーバープロファイルの deploy/messaging/messaging-jboss-beans.xml ファイルに保存されます。messaging という名前のパスワードマスクを作成すると、設定ファイルの前後のスニペットは次のようになります。

```
<property name="suckerPassword">
CHANGE ME!!
</property>
```

```
<annotation>
@org.jboss.security.integration.password.Password(securityDomain=messaging,
methodName=setSuckerPassword)
</annotation>
```

バグの報告

16.17. デフォルトのパスワードマスク設定の変更

デフォルトでは、サーバープロファイルはキーストア SOA_ROOT/jboss-as/bin/password/password.keystore と Key Aliasjboss を使用するように設定されています。パスワードマスキングに使用するキーペアを別の場所に保存するか、別のエイリアスで保存する場合は、サーバープロファイルを新しい場所または Key Alias で更新する必要があります。

手順16.7 タスク

1. テキストエディターでセキュリティー設定ファイルを開きます: vi SOA_ROOT/jboss-as/server/PROFILE/deploy/security/security-jboss-beans.xml。

2. キーストアの場所と Key Alias を編集します。設定例を次に示します。

```
<!-- Password Mask Management Bean-->
  <bean name="JBossSecurityPasswordMaskManagement"
        class="org.jboss.security.integration.password.PasswordMaskManagement" >
```

```
        <property name="keyStoreLocation">password/password.keystore</property>
        <property name="keyStoreAlias">jboss</property>
        <property
name="passwordEncryptedFileName">password/jboss_password_enc.dat</property>
        <property
name="keyStorePasswordEncryptedFileName">password/jboss_keystore_pass.dat</prop
erty>
  </bean>
```

3. ファイルを保存して終了します。

バグの報告

16.18. グローバル設定ファイルのセキュリティー設定

表16.2 jbossesb-properties.xml セキュリティー設定

プロパティー	Description	必須？
org.jboss.soa.esb.services.security.implementationClass	これは、使用すべき具体的な *SecurityService* 実装です。デフォルト設定は、**JaasSecurityService** です。	はい
org.jboss.soa.esb.services.security.callbackHandler	これはデフォルトの **CallbackHandler** 実装であり、JAAS ベースの **SecurityService** が採用されている場合に使用されます。 CallbackHandler の詳細は、セキュリティーのカスタマイズを参照してください。	いいえ
org.jboss.soa.esb.services.security.sealAlgorithm	これは、**SecurityContext** を封印するときに使用するアルゴリズムです。	いいえ
org.jboss.soa.esb.services.security.sealKeySize	これは、**SecurityContext** の暗号化/復号化に使用される秘密/対称キーのサイズです。	いいえ
org.jboss.soa.esb.services.security.contextTimeout	これは、セキュリティーコンテキストが有効な時間 (ミリ秒単位) です。グローバル設定。これはサービスごとに上書きされる場合があります。これを行うには、**jboss-esb.xml** ファイルの security 要素に存在する同じ名前のプロパティーを指定します。	いいえ

プロパティー	Description	必須？
org.jboss.soa.esb.services.security.contextPropagatorImplementationClass	これを使用して、グローバル **SecurityContextPropagator** を設定します。(**SecurityContextPropagator** の詳細については、高度なセキュリティーオプションのセクションを参照してください。)	いいえ
org.jboss.soa.esb.services.security.publicKeystore	これは、Enterprise Service Bus の外部にあるデータの暗号化と復号化に使用されるキーを保持するキー *ストア*です。Keystore は **AuthenticationRequest** を暗号化するために使用されます。	いいえ
org.jboss.soa.esb.services.security.publicKeystorePassword	これは、公開鍵ストアのパスワードです。	いいえ
org.jboss.soa.esb.services.security.publicKeyAlias	これは、公開鍵に使用するエイリアスです。	いいえ
org.jboss.soa.esb.services.security.publicKeyPassword	これは、作成時にエイリアスが指定された場合のエイリアスのパスワードです。	いいえ
org.jboss.soa.esb.services.security.publicKeyPassword	これは暗号変換です。フォーマット: **algorithm/mode/padding**これが指定されていない場合は、デフォルトでキーアルゴリズムが使用されます。	いいえ

バグの報告

16.19. キーペア

キーペアは、公開キーと秘密キーで設定される一連のセキュリティーツールです。公開鍵は暗号化に使用され、秘密鍵は復号化に使用されます。

バグの報告

16.20. キーストア

キーストアはセキュリティーメカニズムです。これには、多数のセキュリティー証明書とそれらに割り当てられたキーが含まれています。クライアント認証が必要な場合に使用されます。

JBoss Enterprise SOA Platform には、SOA_ROOT/jboss-as/samples/quickstarts/security_cert/keystore にあるキーストアの例が同梱されています。これを実稼動環境で使用しないでください。これは例としてのみ提供されています。

バグの報告

16.21. JBOSS RULES

JBoss Rules は、JBoss Enterprise SOA Platform 製品の一部として提供されるビジネスルールエンジンの名前です。

バグの報告

16.22. JBOSS RULES ENGINE を使用したコンテンツベースのルーティング

JBoss Rules は、コンテンツベースルーターのルールプロバイダーエンジンです。Enterprise Service Bus は、次の 3 つの異なるルーティング action classes を通じてこのエンジンと統合されます。

- JBoss Rules エンジンの DRL 言語で記述されたルーティングルールセット (あるいは、必要に応じて DSL 言語を使用することもできます)。
- メッセージの内容。これは、JBoss Rules エンジンに送られるデータです (XML 形式またはメッセージに埋め込まれたオブジェクトとして送られます)。
- 宛先 (エンジンから出力される結果情報から導出されます)。

注記

メッセージが コンテンツベースのルーター に送信されると、ルールセットがそのコンテンツを評価し、一連のサービス宛先を返します。

- org.jboss.soa.esb.actions.ContentBasedRouter: このアクションクラスは *コンテンツベースのルーティング* パターンを実装します。メッセージの内容と、その内容を評価するルールセットに基づいて、メッセージを 1 つ以上の宛先サービスにルーティングします。特定のルールセットまたはメッセージの組み合わせに一致する宛先がない場合には、コンテンツベースルーターは例外を出力します。このアクションはそれ以降のパイプライン処理を終了させるため、常にパイプラインの最後に配置してください。

- org.jboss.soa.esb.actions.ContentBasedWiretap: これは *WireTap* パターンを実装します。WireTap は、メッセージのコピーを制御チャネルに送信するエンタープライズ統合パターンです。WireTap の機能は標準のコンテンツベースのルーターと同じですが、パイプラインは終了されません。これは後者の特徴であり、wire-tap として使用するのに適した特徴であるため、その名前になります。詳細は、 を参照してください。

- org.jboss.soa.esb.actions.MessageFilter: *メッセージフィルター* パターンを実装します。メッセージフィルターパターンは、特定のコンテンツ要件を満たしていない場合に、メッセージを削除できる場合に使用されます。つまり、ルールセットが宛先と一致しない場合は例外を出力しないことを除いて、コンテンツベースのルーターと同じように機能します。詳細は、を参照してください http://www.eaipatterns.com/Filter.html。

バグの報告

16.23. ルールベース

ルールベースは、ルールのコレクションです。これらはイベントの処理に使用されます。ルールは、情報の保存方法と処理方法、許可されるアクション、およびメッセージの送信時に実行するアクションを決定するのに役立ちます。

バグの報告

16.24. シリアライズ

オブジェクトをシリアル化することは、オブジェクトをデータオブジェクトに変換することです。

バグの報告

16.25. 逆シリアル化

ファイルを逆シリアル化することは、ファイルをオブジェクトに変換することです。シリアル化の反対です。

バグの報告

16.26. JBOSS のルールとセキュリティー

デフォルトでは、JBoss Rules コンポーネントはルールパッケージまたは署名されていないルールベースをデシリアライズしません。

重要

設定が Red Hat によってサポートされるようにするには、このシリアライゼーションセキュリティー機能を有効にする必要があります。パッケージとそのルールベースをシリアライズするアプリケーション (以下、サーバーと呼びます) と、パッケージとそのルールベースをデシリアライズするアプリケーション (以下、クライアントと呼びます) の両方のシステムプロパティーを設定する必要があります。

バグの報告

16.27. サーバーでシリアル化を有効にする

手順16.8 タスク

1. SOA_ROOT ディレクトリーに移動します: cd SOA_ROOT。

2.

keytool コマンドを実行し、画面のプロンプトに従います。

```
keytool -genkey -alias droolsKey -keyalg RSA -keystore MyDroolsPrivateKeyStore.keystore
Enter keystore password:
Re-enter new password:
What is your first and last name?
  [Unknown]:  Test User
What is the name of your organizational unit?
  [Unknown]:  HR
What is the name of your organization?
  [Unknown]:  Test Org
What is the name of your City or Locality?
  [Unknown]:  Brisbane
What is the name of your State or Province?
  [Unknown]:  QLD
What is the two-letter country code for this unit?
  [Unknown]:  AU
Is CN=Test User, OU=HR, O=Test Org, L=Brisbane, ST=QLD, C=AU correct?
  [no]:  yes
Enter key password for droolsKey
    (RETURN if same as keystore password):
Re-enter new password:
```

すべての質問に回答すると、パスワードで保護された MyDroolsPrivateKeyStore.keystore という名前のファイルが作成されます。このキーストアファイルには、パスワード drools とともに droolsKey という秘密鍵があります。このファイルを環境内の安全な場所に保存します。これ以降、これを keystoredir と呼びます。

重要

上記のパスワードは単なる例であり、実稼動環境では使用しないでください。

3.

設定ファイルを開きます: vi jboss-as/server/default/deploy/properties-service.xml

4.

次のスニペットを properties-service.xml に追加して、JBoss Rules シリアライゼーション機能を使用するように JBoss Enterprise SOA Platform を設定します。

```
<mbean code="org.jboss.varia.property.SystemPropertiesService"
name="jboss:type=Service,name=SystemProperties">
   <attribute name="Properties">
    # Drools Security Serialization specific properties
    drools.serialization.sign=true

drools.serialization.private.keyStoreURL=file://$keystoredir/MyDroolsPrivateKeyStore.
```

```
keystore
    drools.serialization.private.keyStorePwd=drools
    drools.serialization.private.keyAlias=droolsKey
    drools.serialization.private.keyPwd=drools
  </attribute>
</mbean>
```

5. drools.serialization.sign プロパティーを true に設定します

```
drools.serialization.sign=true
```

- drools.serialization.private.keyStoreURL=<RL> は、秘密鍵ストアの場所の URL です。
- 上記の例で、keystoredir と MyDroolsKeyStore.keystore を、キーストアディレクトリーと、keytool で作成したキーストアの名前に置き換えます。
- drools.serialization.private.keyStorePwd=<password> は、プライベートキーストアにアクセスするためのパスワードです。
- drools.serialization.private.keyAlias=<key> は秘密鍵のキーエイリアス (識別子) です。
- drools.serialization.private.keyPwd=<password> は秘密鍵のパスワードです。

6. ファイルを保存して終了します。

7. サーバーインスタンスを再起動します。

警告

システムプロパティーが正しく設定されていない場合、ルールパッケージをビルドしようとすると、次のエラーが表示されます。

```
An error occurred building the package.

Error
signing object store: Key store with private key not configured. Please
configure it properly before using signed serialization
```

バグの報告

16.28. クライアントでシリアル化を有効にする

前提条件

- **サーバーのシリアル化がすでに有効になっている必要があります。**

手順16.9 タスク

1. **秘密鍵ストアから公開鍵証明書を作成します。(keytool -genkey -alias droolsKey -keyalg RSA -keystore を実行して、キーツールにアクセスできます。):**

```
keytool -export -alias droolsKey -file droolsKey.crt -keystore
```

```
MyDroolsPrivateKeyStore.keystore
Enter keystore password:
Certificate stored in file <droolsKey.crtU>
```

2. **公開鍵証明書を公開鍵ストアにインポートします。(これは、クライアントアプリケーションによって使用される場所です):**

```
keytool -import -alias droolsKey -file droolsKey.crt -keystore
```

```
MyPublicDroolsKeyStore.keystore
```

```
Enter keystore password:
Re-enter new password:
Owner: CN=Test User, OU=Dev, O=XYZ Corporation, L=Brisbane, ST=QLD, C=AU
Issuer: CN=Test User, OU=Dev, O=XYZ Corporation, L=Brisbane, ST=QLD, C=AU
Serial number: 4ca0021b
Valid from: Sun Sep 26 22:31:55 EDT 2010 until: Sat Dec 25 21:31:55 EST 2010
Certificate fingerprints:
     MD5:  31:1D:1B:98:59:CC:0E:3C:3F:57:01:C2:FE:F2:6D:C9
     SHA1: 4C:26:52:CA:0A:92:CC:7A:86:04:50:53:80:94:2A:4F:82:6F:53:AD
     Signature algorithm name: SHA1withRSA
     Version: 3
Trust this certificate? [no]:  yes
Certificate was added to keystore
```

3. **サーバー設定ファイルを開きます: vi grep drools jboss-as/server/default/deploy/properties-service.xml**

4. **keystoredir と MyPublicDroolsKeyStore.keystore をキーストアディレクトリー、以前に作成した公開キーストアの名前と置き換えます。**

```
# Drools Client Properties for Security Serialization
drools.serialization.public.keyStoreURL=file://$keystoredir/MyPublicDroolsKeyStore.keystore
drools.serialization.public.keyStorePwd=drools
```

5. **ファイルを保存して終了します。**

6. **JBoss Enterprise SOA Platform サーバーを再起動します。**

7. **Java クライアントアプリケーションの場合は、コードでシステムプロパティーを次のように設定します。**

```
// Set the client properties to deserialize the signed packages
URL clientKeyStoreURL = getClass().getResource(
"MyPublicDroolsKeyStore.keystore" );
System.setProperty( KeyStoreHelper.PROP_SIGN,
                    "true" );
System.setProperty( KeyStoreHelper.PROP_PUB_KS_URL,
                    clientKeyStoreURL.toExternalForm() );
System.setProperty( KeyStoreHelper.PROP_PUB_KS_PWD,
                    "drools" );
...
```

または、run.sh シェルスクリプト (vi SOA_ROOT/jboss-as/bin/run.sh) スクリプトを開

き、 JAVA_OPTS セクション:

```
# Serialization Security Settings
JAVA_OPTS="-Ddrools.serialization.sign=true $JAVA_OPTS"
JAVA_OPTS="-
Ddrools.serialization.private.keyStoreURL=file://$keystoredir/MyDroolsKeyStore.keyst
ore $JAVA_OPTS"
JAVA_OPTS="-Ddrools.serialization.private.keyStorePwd=drools $JAVA_OPTS"
JAVA_OPTS="-Ddrools.serialization.private.keyAlias=droolsKey $JAVA_OPTS"
JAVA_OPTS="-Ddrools.serialization.private.keyPwd=drools $JAVA_OPTS"
JAVA_OPTS="-
Ddrools.serialization.public.keyStoreURL=file://$keystoredir/MyPublicDroolsKeyStore.
keystore $JAVA_OPTS"
JAVA_OPTS="-Ddrools.serialization.public.keyStorePwd=drools $JAVA_OPTS"
```

上記の値を環境に固有の値に置き換えてから、サーバーインスタンスを再起動します。

バグの報告

16.29. シリアル化署名を無効にする

1. 設定ファイルを開きます: vi SOA_ROOT/jboss-as/server/PROFILE/deploy/properties-service.xml。

2. drools.serialization.sign プロパティーの値を削除します。

3. ファイルを保存して終了します。

このタスクを実行する別の方法は、run.sh シェルスクリプト (vi SOA_ROOT/jboss-as/bin/run.sh) を開いて、次のように編集することです。

```
JAVA_OPTS="-Ddrools.serialization.sign=false $JAVA_OPTS"
```

4. サーバーインスタンスを再起動します。

5. Java クライアントアプリケーションのサインオフをオフにするには、drools.serialization.signプロパティーを変更するか、次のスニペットを各アプリケーションの

コードに追加します。

```
System.setProperty( KeyStoreHelper.PROP_SIGN, "false" );
```

バグの報告

16.30. サービスごとにセキュリティーを設定する

1. グローバル設定ファイルをテキストエディターで開きます: vi SOA_ROOT/jboss-as/server/PROFILE/deployers/esb.deployer/jboss-esb.xml。

2. 設定するサービスまで下にスクロールします。

3. セキュリティー要素を追加します。この設定は、その方法を示しています。

```
<service category="Security" name="SimpleListenerSecured">
  <security moduleName="messaging" runAs="adminRole"
   rolesAllowed="adminRole, normalUsers"

callbackHandler="org.jboss.internal.soa.esb.services.security.UserPassCallbackHandler">
    <property name="property1" value="value1"/>
    <property name="property2" value="value2"/>
  </security>
  ...
</service>
```

4. ファイルを保存して終了します。

バグの報告

16.31. サービスごとのセキュリティープロパティー

表16.3 セキュリティープロパティー

プロパティー	Description	必須？
moduleName	これは **SOA_ROOT/jboss-as/server/PROFILE/conf/login-config.xml** ファイルに存在するモジュールです。	いいえ
runAs	これが runAs ロールです。	いいえ
rolesAllowed	これは、サービスを実行する権限が付与されたロールのコンマ区切りリストです。これは、発信者が実際に指定されたロールの1つに属していることを確認するために、発信者が認証された後に実行されるチェックとして使用されます。ロールは、基礎となるセキュリティーメカニズムによる認証が成功した後に割り当てられます。	いいえ
callbackHandler	これは **jbossesb-properties.xml** ファイルで定義されたものをオーバーライドする **CallbackHandler** です。	いいえ
property	これらはオプションのプロパティーで、一度定義すると **CallbackHandler** 実装で使用できるようになります。	いいえ

バグの報告

16.32. グローバルセキュリティー設定を上書きする

手順16.10 タスク

1. **グローバル設定ファイルをテキストエディターで開きます: vi SOA_ROOT/jboss-as/server/PROFILE/deployers/esb.deployer/jbossesb-properties.xml**

2. **問題の設定を設定します。以下に例を示します。**

```
<security moduleName="messaging"
  runAs="adminRole" rolesAllowed="adminRole">

<property
    name="org.jboss.soa.esb.services.security.contextTimeout"
```

```
    value="50000"/>

<property name=
"org.jboss.soa.esb.services.security.contextPropagatorImplementationClass"
    value="org.xyz.CustomSecurityContextPropagator" />

</security>
```

3. ファイルを保存して終了します。

バグの報告

16.33. セキュリティープロパティーのオーバーライド

表16.4 セキュリティープロパティーのオーバーライド:

プロパティー	Description	必須？
org.jboss.soa.esb.services.security.contextTimeout	このプロパティーにより、サービスは **jbossesb-properties.xml** ファイルで指定されたグローバルセキュリティーコンテキストタイムアウト (ミリ秒) をオーバーライドできます。	いいえ
org.jboss.soa.esb.services.security.contextPropagatorImplementationClass	このプロパティーにより、サービスは **jbossesb-properties.xml** ファイルで指定されたグローバルセキュリティーコンテキストプロパゲータークラスの実装をオーバーライドできます。	いいえ

バグの報告

16.34. セキュリティーコンテキスト

SecurityContext は、セキュリティー証明書が確認された後に作成されるオブジェクトです。作成後、証明書に関連するアクションを実行するたびに証明書を再認証する必要がないように設定されます。ESB は、メッセージに SecurityContext があることを検出すると、それがまだ有効であることを確認し、有効である場合は再認証を試みません。(SecurityContext は単一の Enterprise Service Bus ノードに対してのみ有効であることに注意してください。メッセージが別の ESB ノードにルーティングされる場合は、再認証する必要があります。)

バグの報告

16.35. 認証リクエスト

AuthenticationRequest は、ゲートウェイとサービス間、または 2 つのサービス間の認証に必要なセキュリティー情報を運びます。認証サービスを呼び出す前に、メッセージオブジェクトにこのクラスのインスタンスを設定する必要があります。クラスには、呼び出し元を認証するために必要な原則と認証情報が含まれている必要があります。このクラスは、コールバックハンドラーで使用できるようになります。

バグの報告

16.36. SECURITYCONFIG

SecurityConfig クラスは、jboss-esb.xml ファイルで指定されたセキュリティー設定へのアクセスを許可します。このクラスは、コールバックハンドラーで使用できるようになります。

バグの報告

16.37. 認証クラスをメッセージオブジェクトに追加する

手順16.11 タスク

- 次のコードを実行します。

```
byte[] encrypted = PublicCryptoUtil.INSTANCE.encrypt((Serializable)
 authRequest);
message.getContext.setContext(SecurityService.AUTH_REQUEST, encrypted);
```

結果

認証コンテキストは暗号化され、メッセージコンテキスト内に設定されます。(リクエストを認証できるように、後で Enterprise Service Bus によって復号化されます。)

バグの報告

16.38. SECURITY_BASIC クイックスタート

SOA_ROOT/jboss-as/samples/quickstarts/security_basic クイックスタートは、SecurityInvoker を使用する前にメッセージのセキュリティーを準備する方法を示しています。クイックスタートは、jbossesb-properties.xml グローバル設定ファイルをクライアントサービスで使用するように設定する方法も示します。

バグの報告

16.39. セキュリティーコンテキストの時間制限をグローバルに設定する

手順16.12 タスク

1. グローバル設定ファイルをテキストエディターで開きます: vi SOA_ROOT/jboss-as/server/PROFILE/deployers/esb.deployer/jbossesb-properties.xml
2. security.contextTimeout を含むセクションまで下にスクロールします。タイムアウト値を設定します (ミリ秒単位)。
3. ファイルを保存して終了します。

バグの報告

16.40. サービスごとにセキュリティーコンテキストの時間制限を設定する

手順16.13 タスク

1. サービスの設定ファイル vi jboss-esb.xml をテキストエディターで開きます。
2. Security Context を含むセクションまで下にスクロールします。タイムアウト値を設定します (ミリ秒単位)。

3. ファイルを保存して終了します。

バグの報告

第17章 高度なセキュリティーオプション

17.1. セキュリティーの伝播

セキュリティー伝播という用語は、セキュリティー情報を外部システムに渡すプロセスを指します。たとえば、Enterprise Service Bus と Enterprise Java Beans メソッドの両方を呼び出すために、同じ認証情報を使用したい場合があります。

バグの報告

17.2. SECURITYCONTEXTPROPAGATOR

SecurityContextPropagator クラスは、セキュリティーコンテキストを宛先環境に渡します。

バグの報告

17.3. SECURITYCONTEXTPROPAGATOR の実装

表17.1 SecurityContextPropagator の実装

Class	Description
パッケージ: org.jboss.internal.soa.esb.services.security Class: JBossASContextPropagator	このプロパゲーターは、セキュリティー認証情報を ESB に送信します。独自の実装を記述する必要がある場合は、**org.jboss.internal.soa.esb.services.security.SecurityContextPropagator** を実装するクラスを記述し、**jbossesb-properties.xml** または **jboss-esb.xml** でその実装を指定するだけです。

バグの報告

17.4. カスタムログインモジュールの追加

手順17.1 タスク

1. テキストエディターでログイン設定ファイルを開きます: **vi SOA_ROOT/jboss-**

as/server/PROFILE/conf/login-config.xml

2. カスタムログインモジュールの詳細を追加します。

3. ファイルを保存して終了します。

4. ログインモジュールが異なれば必要な情報も異なるため、使用する CallbackHandler 属性を指定する必要があります。そのサービスの特定のセキュリティー設定を開きます。

5. CallbackHandler が Esb CallbackHandler インターフェイスを実装するクラスの *完全修飾* クラス名を指定するようにしてください。このコードは、その方法を示しています。

```
public interface EsbCallbackHandler extends CallbackHandler
{
  void setAuthenticationRequest(final AuthenticationRequest authRequest);
  void setSecurityConfig(final SecurityConfig config);
}
```

6. 発信者を認証するために必要な原則と認証情報の両方を AuthenticationRequest クラスに追加します。

結果

JaasSecurityService は、カスタムセキュリティー実装に置き換えられます。

バグの報告

17.5. 証明書ログインモジュール

証明書ログインモジュールは、Enterprise Service Bus への呼び出しで渡された証明書を、ローカルキーストアに保持されている証明書と照合して認証を実行します。証明書の共通名は原則を作成します。

バグの報告

17.6. 証明書ログインモジュールのプロパティー

```
<security moduleName="CertLogin" rolesAllowed="worker"
  callbackHandler="org.jboss.soa.esb.services.security.auth.loginUserPass
CallbackHandler">
  <property name="alias" value="certtest"/>
</security>
```

表17.2 プロパティー

プロパティー	Description
moduleName	これは、使用する JAAS ログインモジュールを識別します。このモジュールは JBossAS login-config.xml で指定されます。
rolesAllow	これは、このサービスの実行が許可されているロールのコンマ区切りリストです。
alias	これは、ローカルキーストアを検索するために使用されるエイリアスであり、呼び出し元の証明書を検証するために使用されます。

バグの報告

17.7. 証明書ログインモジュールの設定ファイルのプロパティー

```
<application-policy name="CertLogin">
<authentication>
  <login-module
code="org.jboss.soa.esb.services.security.auth.login.CertificateLoginModule"
flag = "required" >
  <module-option name="keyStoreURL">
    file://pathToKeyStore
  </module-option>
  <module-option name="keyStorePassword">storepassword</module-option>
  <module-option name="rolesPropertiesFile">
    file://pathToRolesFile
  </module-option>
  </login-module>
</authentication>
</application-policy>
```

表17.3 証明書ログインモジュールの設定ファイルのプロパティー

プロパティー	Description

プロパティー	Description
keyStoreURL	これは、証明書の検証に使用されるキーストアへのパスです。これは、ローカルファイルシステムまたはクラスパス上のファイルにすることができます。
keyStorePassword	これは、上記のキーストアのパスワードです。
rolesPropertiesFile	これは任意です。これは、ロールのマッピングを含むファイルへのパスです。詳細については、Getting Started Guide の Role Mapping セクションを参照してください。

バグの報告

17.8. コールバックハンドラー

コールバックハンドラーは、バックエンド操作で使用されるライブラリーの一種です。アプリケーションがセキュリティーサービスを介して相互に対話できるようにし、認証データの確認に使用できます。

バグの報告

17.9. ロールマッピング

ロールマッピングは、セキュアなホスト間でデータを共有する方法です。信頼できるホストのリストを含むファイルが作成され、各ホストにはいくつかのロールのマッピングが割り当てられます。いずれかのホストからデータにアクセスすると、マッピングが発生します。送信者のロールは受信者のロールにマッピングされ、認証とデータ共有が可能になります。ロールのタイプには、ユーザーロール、アプリケーションロールなどがあります。これはオプション機能であり、デフォルトでは有効になっていません。

バグの報告

17.10. ロールマッピングを有効にする

手順17.2 タスク

1.

テキストエディターでログイン設定ファイルを開きます: vi SOA_ROOT/jboss-as/server/PROFILE/conf/login-config.xml

2. をセットする rolesPropertiesFile財産。(このプロパティーは、ローカルファイルシステムまたはクラスパスのいずれかにあるファイルを指すことができます)。

3. ユーザーをロールにマップします。このコード例は、その方法を示しています。

```
# user=role1,role2,...
guest=guest
esbuser=esbrole
# The current implementation will use the Common Name(CN) specified
# for the certificate as the user name.
# The unicode escape is needed only if your CN contains a space
Andy\u0020Anderson=esbrole,worker
```

4. ファイルを保存して終了します。

バグの報告

17.11. SECURITY_CERT クイックスタート

security_cert クイックスタートは、JBoss Enterprise SOA Platform のロールマッピング機能を示します。

バグの報告

17.12. セキュリティーサービス

SecurityService インターフェイスは、Enterprise Service Bus の中央のセキュリティーコンポーネントです。

バグの報告

17.13. セキュリティーサービスインターフェイスのカスタマイズ

手順17.3 タスク

1. SecurityService インターフェイスを実装します。

```
public interface SecurityService
{
   void configure() throws ConfigurationException;

   void authenticate(
      final SecurityConfig securityConfig,
      final SecurityContext securityContext,
      final AuthenticationRequest authRequest)
      throws SecurityServiceException;

   boolean checkRolesAllowed(
      final List<String> rolesAllowed,
      final SecurityContext securityContext);

   boolean isCallerInRole(
      final Subject subject,
      final Principle role);

   void logout(final SecurityConfig securityConfig);

   void refreshSecurityConfig();
}
```

2. グローバル設定ファイルをテキストエディターで開きます: vi SOA_ROOT/jboss-as/server/PROFILE/deployers/esb.deployer/jbossesb-properties.xml

3. カスタマイズされた SecurityServiceを使用するように ファイルを設定します。

4. ファイルを保存して終了します。

バグの報告

17.14. リモート呼び出しクラス

リモート呼び出しクラスは、その名前が示すように、リモートマシンから呼び出すことができるクラスです。これは開発者にとっては便利ですが、潜在的なセキュリティーリスクにつながる可能性もあ

ります。

バグの報告

17.15. ポート 8083 での非リモートメソッド呼び出しクラスの保護

デフォルトでは、クライアントアプリケーションは Remote Method Invocation を利用して、ポート 8083 経由で Enterprise Java Bean クラスをダウンロードできます。ただし、システムのリモートメソッド呼び出し設定を設定して、クライアントアプリケーションが必要なデプロイ済みリソースをダウンロードできるようにすることもできます。

手順17.4 タスク

1. **jboss-service.xml ファイルの設定の編集**

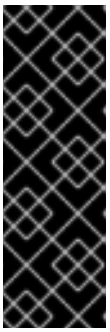
   テキストエディターでファイルを開きます: **vi SOA_ROOT/server/PROFILE/conf/jboss-service.xml**

2. **ファイルで設定を設定する**

   以下に例を示します。

```
<attribute name="DownloadServerClasses">false</attribute>
```

   クライアントアプリケーションがエンタープライズ Java Bean クラスのみをダウンロードできるようにするには、この値を false に設定します。



重要

デフォルトでは、この値は SOA プラットフォームの運用プロファイルで false に設定されています。SOA スタンドアロンバージョンのデフォルトプロファイルを含め、他のすべての場合、値は true に設定されます。これは安全な設定ではないため、開発環境でのみ使用する必要があることに注意してください。

バグの報告

第18章 サービスレジストリーの保護

18.1. サービスレジストリー認証

はじめに

ここでは、認証プロセスがどのように機能するかについての理論的な理解を示します。

認証は 2 フェーズのプロセスです。これらは、*認証フェーズ* および *識別フェーズ* として知られています。これらのフェーズはいずれも、 Authenticator インターフェイスのメソッドによって表されます。

認証フェーズは、 GetAuthToken リクエストが行われます。このフェーズの目標は、 user id と認証情報を有効な publisher id にかえることです。パブリッシャー ID (UDDI 用語では *許可された名前* と呼ばれます) は、UDDI 内で所有権を割り当てる値です。新しいエンティティーが作成されるたびに、発行者の承認された名前による所有権でタグ付けする必要があります。

publisher idの値は、jUDDI レジストリーには関係ありません。唯一の要件は、新しいエンティティーに割り当てるために存在するため、null でない必要があることです。 GetAuthToken 要求が完了すると、認証トークン が呼び出し元に発行されます。

認証を必要とする UDDI API への後続の呼び出しを行うときは、 GetAuthToken 要求に対する応答として発行されたトークンを提供する必要があります。これは、識別フェーズにつながります。

識別フェーズは、認証トークン (または publisher idそのトークンに関連付けられている) を有効な UddiEntityPublisher オブジェクトに変換します。このオブジェクトには、UDDI エンティティーの所有権を処理するために必要なすべてのプロパティーが含まれています。したがって、トークン (または publisher id) は、発行元を識別するために使用されます。

この 2 つのフェーズは、UDDI 認証構造に準拠し、独自の認証メカニズムを提供する場合に柔軟性を提供します。

クレデンシャルとパブリッシャープロパティーの処理は、jUDDI レジストリーの外部で完全に実行できます。ただし、デフォルトでは、レジストリーは UddiEntityPublisher のサブクラスである Publisher エンティティーを提供します。このサブクラスは、パブリッシャーのプロパティーを jUDDI レジストリー内で永続化します。

バグの報告

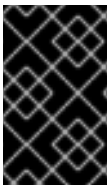
18.2. AUTHTOKEN

authToken は、パスワード認証情報を保持するセキュリティーコンテナーです。

バグの報告

18.3. AUTHTOKEN とサービスレジストリー

Service Registry への適切な書き込みアクセスを強制するには、サービスレジストリーに対する各リクエストに有効な authToken が必要です。



重要

読み取りアクセスはまったく制限されていないことに注意してください。

バグの報告

18.4. AUTHTOKEN を取得する

手順18.1 タスク

1. GetAuthToken() リクエストを行います。

2. GetAuthToken オブジェクトが返されます。 userid と credential (パスワード) をこのオブジェクトで設定します。

```
org.uddi.api_v3.GetAuthToken ga = new org.uddi.api_v3.GetAuthToken();
ga.setUserID(pubId);
ga.setCred("");

org.uddi.api_v3.AuthToken token = securityService.getAuthToken(ga);
```

3. SOA_ROOT/jboss-as/server/PROFILE/deploy/juddi-service.sar/juddi.war/WEB-INF で juddi.properties 設定ファイルを見つけます。テキストエディターで開きます。

4. を設定します juddi.authenticatorプロパティーを使用して、GetAuthToken リクエストによって渡された認証情報をサービスレジストリーがチェックする方法を指定します。(デフォルトでは、jUDDIAuthenticator 実装を使用します。)

5. ファイルを保存して終了します。

バグの報告

18.5. SERVICE REGISTRY で利用可能なセキュリティー認証の実装

jUDDI 認証

警告

この認証方法を本番環境で使用しないでください。提供されたすべての認証情報を受け入れ、クライアントがレジストリーにアクセスするときに認証する必要性を効果的に取り除きます。

Service Registry によって提供されるデフォルトの認証メカニズムは jUDDIAuthenticator です。jUDDIAuthenticator の認証フェーズは、user ID が Publisher テーブルのレコードに対して一致を送信したかどうかを確認します。資格証明のチェックは行われません。認証プロセス中に Publisher レコードが存在しないことが判明した場合は、オンザフライで追加されます。

識別フェーズでは、publisher ID Publisher レコードを取得して返すために使用されます。パブリッシャーは、必要なすべてのプロパティーを UddiEntityPublisher から継承します。

```
juddi.authenticator = org.apache.juddi.auth.JUDDIAuthentication
```

XMLDocAuthentication

認証フェーズでは、ユーザー ID とパスワードが XML ファイル内の値と一致することを確認し

ます。識別フェーズでは、user IDを使用して新しい UddiEntityPublisher を設定します。

CryptedXMLDocAuthentication

CryptedXMLDocAuthentication の実装は XMLDocAuthentication の実装に似ていますが、パスワードは暗号化されています。

```
juddi.authenticator = org.apache.juddi.auth.CryptedXMLDocAuthentication
juddi.usersfile = juddi-users-encrypted.xml
juddi.cryptor = org.apache.juddi.cryptor.DefaultCryptor
```

ここでは、ユーザー認証情報ファイルは juddi-users-encrypted.xml であり、ファイルの内容は次のようになります。

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<juddi-users>
<user userid="anou_mana" password="+j/kXkZJftwTFTBH6Cf6IQ=="/>
<user userid="bozo" password="Na2Ait+2aW0="/>
<user userid="sviens" password="+j/kXkZJftwTFTBH6Cf6IQ=="/>
</juddi-users>
```

DefaultCryptor の実装では、BEWithMD5AndDES と Base64 を使用してパスワードを暗号化します。

注記

AuthenticatorTest のコードを使用して、この Authenticator 実装の使用方法について詳しく知ることができます。org.apache.juddi.cryptor.Cryptor インターフェイスを実装し、juddi.cryptor プロパティーで実装クラスを参照することで、独自の暗号化アルゴリズムをプラグインできます。

認証フェーズでは、XML ファイル内の user ID とパスワード一致値をチェックします。識別フェーズでは、user ID を使用して新しい UddiEntityPublisher を設定します。

LDAP 認証

LdapSimpleAuthenticator を使用して、LDAP の簡易認証機能を介してユーザーを認証します。このクラスを使用すると、principle と jUDDI publisher ID が同じ場合に、*LDAP 原則* に基づいてユーザーを認証します。

JBoss 認証

最後の代替手段は、サードパーティーの認証情報ストアと連携することです。JBoss Application Server の認証コンポーネントにリンクできます。

docs/examples/auth ディレクトリーに JBossAuthenticator クラスが提供されています。このクラスは、JBoss 上の jUDDI デプロイメントがサーバーセキュリティードメインを使用してユーザーを認証できるようにします。

バグの報告

18.6. XMLDOCAUTHENTICATION の設定

手順18.2 タスク

1. juddi-users.xml というテキストファイルを作成し、jbossesb-registry.sar に保存します。

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<juddi-users>
   <user userid="sscholl" password="password" />
   <user userid="dsheppard" password="password" />
   <user userid="vbrittain" password="password" />
</juddi-users>
```

2. ファイルを保存して終了します。

3. ファイルをクラスパスに追加します。

4. テキストエディターで juddi.properties ファイルを開きます (SOA_ROOT/jboss-as/server/PROFILE/deploy/juddi-service.sar/juddi.war/WEB-INF)。

5. ファイルを次のように変更します。

```
juddi.authenticator = org.apache.juddi.auth.XMLDocAuthentication
juddi.usersfile = juddi-users.xml
```

6. ファイルを保存して終了します。

バグの報告

18.7. LIGHTWEIGHT DIRECTORY ACCESS PROTOCOL (LDAP)

Lightweight Directory Access Protocol (LDAP) は、インターネット経由で分散ディレクトリー情報にアクセスするためのプロトコルです。

バグの報告

18.8. LDAP 認証の設定

手順18.3 タスク

1. SOA_ROOT/jboss-as/server/PROFILE/deploy/juddi-service.sar/juddi.war/WEB-INF で juddi.properties ファイルを見つけます。テキストエディターで開きます。

2. 次の構成設定を追加してください。

```
juddi.authenticator=org.apache.juddi.auth.LdapSimpleAuthenticator
juddi.authenticator.url=ldap://localhost:389
```

juddi.authenticator.url プロパティーは、LDAP サーバーが存在する場所を LdapSimpleAuthenticator クラスに通知します。

3. ファイルを保存して終了します。

バグの報告

18.9. JBOSS 認証の設定

手順18.4 タスク

1. SOA_ROOT/jboss-as/server/PROFILE/deploy/juddi-service.sar/juddi.war/WEB-INF で juddi.properties ファイルを見つけます。テキストエディターで開きます。

2. ファイルに以下の行を追加します。

```
uddi.auth=org.apache.juddi.auth.JBossAuthenticator
juddi.securityDomain=java:/jaas/other
```

juddi.authenticator プロパティーは、JbossAuthenticator クラスを jUDDI レジストリーの Authenticator フレームワークに接続します。juddi.security.domain は、JBossAuthenticator に Application Server のセキュリティードメインを見つけることができる場所を伝えます。このドメインを使用して認証を実行します。

JBoss は SOA_ROOT/jboss-as/server/PROFILE/conf/login-config.xml ファイル内のアプリケーションポリシー要素ごとに 1 つのセキュリティードメインを作成することに注意してください。これらのドメインは、次の名前でサーバー JNDI ツリーにバインドされます。java:/jaas/<application-policy-name>(ルックアップが存在しないアプリケーションポリシーを参照する場合、other という名前のポリシーがデフォルトで使用されます。)

3. ファイルを保存して終了します。

バグの報告

パート VI. 操作

第19章 JBOSS ENTERPRISE SOA PLATFORM を本番環境で実行する

19.1. サーバープロファイル

表19.1 サーバープロファイル

プロファイル	Description
default	このプロファイルは、開発とテストに使用します。このプロファイルは、プロダクションプロファイルよりもメモリーの使用量が少なくなりますが、このモードではクラスターリングは有効になりません。さらに、このプロファイルは、"all" および "production" プロファイルよりも詳細なログを提供します。この詳細ログは追加情報を提供しますが、サーバーのパフォーマンスに悪影響を及ぼします。別のプロファイルを明示的に指定しない限り、サーバーの始動時にこのプロファイルが使用されます。
実稼働	このプロファイルは実動サーバーで使用してください。このプロファイルは、クラスターリングを提供し、より多くのメモリーを使用し、all または default プロファイルよりも詳細なログと画面コンソール出力を提供しないことで、パフォーマンスを最大化します。このモードでは、出力 (Hello World クイックスタートからのメッセージなど) がコンソール画面に表示されないことに注意してください。ログのみに書き込まれます。
最小	機能するシステムに必要な最小限の機能を有効にします。アーカイブはデプロイされません。ESB または SOA 機能は有効化されていません。BPEL エンジンは使用できません。
standard	これにより、テスト用の標準機能が提供されます。Web、ESB、または SOA 機能は有効になっていません。BPEL エンジンは使用できません。
web	このプロファイルが実行されると、jbossweb.sar アーカイブがデプロイされます。ESB または SOA 機能は有効化されていません。BPEL エンジンは使用できません。
all	このプロファイルを実行すると、事前にパッケージ化された ESB アーカイブがすべてデプロイされます。このプロファイルは、運用プロファイルよりもパフォーマンスとスケーラビリティが低くなりますが、実行に必要なメモリーは少なくなります。

バグの報告

19.2. RUN.SH オプションのスイッチ

表19.2 ./run.sh オプションのスイッチ

スイッチ	目的	使用例
-c	サーバーが特定のプロファイルを使用するようにします。何も指定されていない場合は、デフォルトが使用されます。	**./run.sh -c production**
-b	サーバーを特定の IP アドレスにバインドします。何も指定されていない場合は、デフォルト (127.0.0.1) が使用されます。	**./run.sh -b 10.34.5.2**

バグの報告

19.3. 本番環境で JBOSS ENTERPRISE SOA PLATFORM を起動する

手順19.1 本番環境で JBoss Enterprise SOA Platform を起動する

1. **bin ディレクトリーに移動します。**

   ターミナルを開き、次のコマンドを入力します: cd SOA_ROOT/jboss-as/bin (または Microsoft Windows では chdir SOA_Root\jboss-as\bin)。

   注記

   続行する前に、管理者のユーザー名とパスワードを設定する必要があります。

2. **Red Hat Enterprise Linux で JBoss Enterprise SOA サーバーを起動します。**

   製品を開始するには、次のコマンドを実行します: ./run.sh -c production

3. **Microsoft Windows で JBoss Enterprise SOA サーバーを起動します。**

   製品を開始するには、次のコマンドを実行します: run.bat -c production

結果

サーバーが起動します。ハードウェアの速度によっては、最大で約 2 分かかる場合があることに注意してください。

注記

エラーがないことを確認するには、サーバーログを確認します: less SOA_ROOT/jboss-as/server/PROFILE/log/server.log別のチェックとして、Web ブラウザーを開いて、http://localhost:8080 に移動します。設定したユーザー名とパスワードで管理コンソールにログインできることを確認してください。

バグの報告

19.4. サーバーのインストール

サーバーのインストールは、システムで JBoss Enterprise SOA Platform を設定する方法です。この方法でソフトウェアをインストールすると、ホストオペレーティングシステムを使用してソフトウェアを起動およびシャットダウンできます。これは、オペレーティングシステムの他のサービス (Linux/Unix 用語ではデーモン) と同じようにセットアップされます。

バグの報告

19.5. JBOSS ENTERPRISE SOA PLATFORM を RED HAT ENTERPRISE LINUX デーモンとして実行するように設定する

手順19.2 タスク

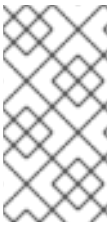
- JBoss Enterprise SOA Platform をバックグラウンドデーモン (サービス) として実行するには、独自のシェルスクリプトを作成する必要があります。Red Hat は、これを行うためのスクリプトを提供していません。

バグの報告

19.6. サーバーのインストールを開始する

前提条件

- JBoss Enterprise SOA Platform は、サービスとして実行するように事前設定する必要があります。



注記

この例では、サービスが jboss_soa という名前を使用してインストールされたと想定しています

手順19.3 タスク

- JBoss Enterprise SOA Platform をサービスとして起動するには、次のコマンドを発行します: service jboss_soa start

注記

JBoss ユーザーが (-R スイッチを使用して) システムアカウントとして作成された場合、警告メッセージが表示されます。これは無視しても問題ありません。

バグの報告

19.7. サーバーのインストールを停止する

この例では、サービスが jboss_soa という名前を使用してインストールされたと想定しています

手順19.4 タスク

- サービスとして実行中の JBoss Enterprise SOA Platform を停止するには、次のコマンドを発行します: service jboss_soa stop

バグの報告

パート VII. 削除

第20章 削除

20.1. システムからの JBOSS ENTERPRISE SOA PLATFORM の削除

手順20.1 システムからの JBoss Enterprise SOA Platform の削除

1. **Red Hat Enterprise Linux システムから JBoss Enterprise SOA Platform を削除する**

   サーバーがシャットダウンしていることを確認したら、SOA_ROOT ディレクトリーの上のレベルに移動し、rm -Rf SOA_ROOT コマンドを実行します。

2. **Microsoft Windows システムから JBoss Enterprise SOA Platform を削除する**

   サーバーがシャットダウンされていることを確認したら、Windows Explorer を開き、SOA_ROOT があるディレクトリーに移動し、SOA_ROOT を選択して削除します。

3. データベースを削除します。

バグの報告

付録A JBOSS ENTERPRISE SOA PLATFORM FOR CLOUD COMPUTING の設定

A.1. AMAZON ELASTIC COMPUTE CLOUD (EC2)

Amazon Elastic Compute Cloud (EC2)は、が提供 http://aws.amazon.com/ec2/ するサービスです。独自のソフトウェアを実行する仮想マシンを親できます。これは非常にスケーラビリティーの高いサービスであり、特定の処理のニーズを満たすために仮想マシンをリアルタイムで作成することができます。

バグの報告

A.2. EC2 クラウドで使用する JBOSS ENTERPRISE SOA PLATFORM の設定

前提条件

- 

  警告

  このプロセスでは、潜在的なセキュリティーリスクにつながるサーバーのファイアウォールをシャットダウンする必要があるため、この設定タスクは VPC サブネット内で実行され、ファイアウォールは外部インターフェイスで実行される必要があります。それ以外の場合、Red Hat はサポートしません。

手順A.1 タスク

1. EC2 でノードを作成します。それらは、制限のないセキュリティーグループに属している必要があります。

2. AWS 内で、データベースとして使用されるインスタンスも作成します。

3. AWS で新しい S3 バケットを作成します。

4. カスタマーポータル から JBoss Enterprise SOA Platform EC2 パッチをダウンロードします。

5. パッチを展開して、保存した場所から開きます。

6. ant を実行します。

これにより、実稼働 設定に基づいて cluster-ec2 という新しい設定が作成されます。

7. 新しい設定に必要な変更を加えます（デフォルトのデータベースの代わりに MySQL を使用するように変更するなど）。

8. すべてのノードでファイアウォール(iptables)をシャットダウンします。

9. 新たに作成した設定を実行し、 jboss.jgroups.s3_ping.access_key、jboss.jgroups.s3_ping.secret_access_key、および jboss.jgroups.s3_ping.bucket のパラメーターを渡します。(AWS コンソールからこれらを取得できます)

バグの報告

A.3. EC2 設定のトラブルシューティング

- パッチのバージョン番号が JBoss Enterprise SOA Platform ディストリビューションと一致することを確認してください。

- 多数のノードを起動した後も、クラスターにメンバーが 1 つしかない場合は、おそらくファイアウォールに関連する問題である可能性があります。

- JBoss Enterprise SOA Platform の起動時に、内部 VPN に接続されていないことを確認してください。

バグの報告

付録B JBPM 5 のインストール

B.1. JBPM 5 のインストール

前提条件

- BRMS 5.3 以降のサブスクリプション
- SOA Server にインストールされた BRMS deployable パッケージ

手順B.1 タスク

1. カスタマーポータルから soa-p-VERSION-jbpm.zip をダウンロードします。
2. テキストエディターで build.properties ファイルを開きます。必要な設定変更を行います。
3. build.properties ファイルを保存し、テキストエディターを終了します。
4. インストールスクリプトを実行します。

   このスクリプトは、soa-p-VERSION- jbpm.zip から SOA-P サーバーに jbpm5.esb ファイルをデプロイし、JBPM サービスリポジトリーを設定し、JBPM5 コンソール、ヒューマンタスク WAR ファイル、およびライブラリーを BRMS インストールから SOA サーバーにコピーします。

   または、スクリプトが機能しない場合は、最新の手順を参照 https://bugzilla.redhat.com/attachment.cgi?id=590710 してください。
5. JBoss Enterprise SOA Platform サーバーを起動します。
6. インストールをテストするには、jBPM 5 クイックスタートをインストールしたディレクトリー(cd SOA_ROOT/jboss-as/samples/quickstarts/bpm5processor)に移動します。

7. **クイックスタートのデプロイ： ant deploy**

8. **クイックスタートの実行： ant runtest**

バグの報告

付録C 便利な定義

C.1. ENTERPRISE SERVICE BUS

Enterprise Service Bus は、抽象的な SOA 設計構想の具体的な実装です。Enterprise Service Bus (ESB) には、メッセージルーティング機能を提供するロールと、サービスを登録できるようにするロールの 2 つがあります。JBoss Enterprise SOA Platform の中心にあるEnterprise Service Bus は、JBoss ESB と呼ばれます。

Enterprise Service Bus は、ビジネスロジックではなくインフラストラクチャーロジックを扱います (ビジネスロジックはより高いレベルに任せられます)。データは、Enterprise Service Bus を介して 2 つ以上のシステム間を移動します。メッセージキューが含まれる場合と含まれない場合があります。ESB は、データを宛先に渡す前に変換エンジンに渡すこともできます。

バグの報告

C.2. JAVA 仮想マシン

Java 仮想マシン (JVM) は、Java バイトコードを実行できるソフトウェアです。JVM は、中間バイトコードが実行される標準環境を作成します。基盤となるハードウェアとオペレーティングシステムの組み合わせに関係なく標準的な環境を作成することにより、プログラマーは一度 Java コードを記述すれば、どのシステムでも実行できるという安心感を持つことができます。Red Hat は、OpenJDK を使用することを推奨します。OpenJDK は、Red Hat Enterprise Linux システムで適切に動作する、サポートされているオープンソースの Java 仮想マシンです。Windows ユーザーは、Oracle JDK 1.6 をインストールする必要があります。

バグの報告

C.3. SOA-USERS.PROPERTIES

soa-users.properties ファイルは、SOA Web コンソールにアクセスするためのユーザーアカウントとパスワードが格納される場所です。管理者は、このファイルを編集してシステムへのアクセスを制御します。パスワードは平文で保存されるため、本番システムでは代わりにパスワード暗号化を使用する必要があることに注意してください。

バグの報告

C.4. SOA-ROLES.PROPERTIES

soa-roles.properties ファイルは、ユーザーアクセス権限が定義される場所です。このファイルは次の構文: username=role1,role2,role3 を使用します。ロールはいくつでも割り当てることができます。サーバーコンソールにログインできるようにするには、ユーザーに JBossAdmin、HttpInvoker、user、および admin ロールを割り当てる必要があることに注意してください。

バグの報告

C.5. RUN.SH

run.sh は、JBoss Enterprise SOA Platform を起動するためにユーザーが実行するシェルスクリプトです。Microsoft Windows では、run.bat がこれに相当します。スクリプトには、ユーザーがシェルで指定したプロファイルとポートバインドを使用してサーバーを起動するために必要なコマンドが含まれています。スクリプトは SOA_ROOT/jboss-as/bin ディレクトリーにあります。

バグの報告

C.6. RUN.CONF

SOA_ROOT/bin/run.conf は、デフォルトのサーバー設定ファイルです。run.conf.bat は Microsoft Windows と同等のものです。

バグの報告

C.7. ブートストラップモード

ソフトウェアをブートストラップモードにすると、何をいつロードするかが指示されます。

バグの報告

C.8. メッセージ再配信サービス

メッセージ再配信サービスは、エンドポイント参照が機能しない場合にメッセージの再配信を試みます。

バグの報告

C.9. アクションパイプライン

アクションパイプラインは、メッセージが処理されるアクションクラスのリストで構成されます。メッセージ処理時に実行するアクションを指定するために使用します。アクションは、メッセージを変換し、ビジネスロジックを適用できます。各アクションはメッセージをパイプラインの次のメッセージに渡すか、プロセスの最後に ReplyTo アドレスで指定されたエンドポイントリスナーに送信します。

アクションパイプラインは、通常の処理とそれに続く結果処理の 2 段階で機能します。最初のステージでは、パイプラインは、パイプラインの最後に到達するかエラーが発生するまで、各アクション (デフォルトではプロセスと呼ばれます) でプロセスメソッドを順番に呼び出します。この時点で、パイプラインは反転し (第 2 段階)、前の各アクションで結果メソッドを呼び出します (デフォルトでは、processException または processSuccess)。現在のアクション (成功した場合の最後のアクションまたは例外を発生させたアクション) から開始し、パイプラインの先頭に到達するまで逆方向に移動します。

バグの報告

C.10. クラスパス

クラスパスは、Java 仮想マシンに、ファイルシステム上でユーザーが作成したクラスとパッケージを見つける場所を伝える設定です。

バグの報告

C.11. ビジネスプロセス定義

ビジネスプロセス定義は、プロセスで使用されるランタイムインスタンスの共通要素を決定します。再利用可能です。

バグの報告

C.12. サーバープロファイル

サーバープロファイルは、JBoss Enterprise SOA Platform をさまざまな方法で実行するための事前定義された設定のセットです。次のプロファイルが製品に付属しています。all、default、minimal、production、standard、および web。これらは SOA_ROOT/jboss-as/server/ ディレクトリーにあります。ユーザーは、-c スイッチを使用して、ソフトウェアの起動時に実行するプロファイルを指定します。何も指定されていない場合は、"Default" プロファイルが使用されます。

バグの報告

C.13. データソース名。

データソース名 (DSN) は、特定のデータに付けられたタイトルです。たとえば、DSN はデータベースの名前を参照できます。

バグの報告

C.14. ディシジョンテーブル

デシジョンテーブルには、アクションのリストが含まれています。これらは、必要に応じてシステムによって実行されます。

バグの報告

C.15. サービス

サービスは、ESB メッセージを順次処理するアクションクラスのリストです。各サービス要素は、1 つ以上のリスナーと 1 つ以上のアクションで構成されます。これらは jboss-esb.xml 設定ファイル内で

設定されます。

バグの報告

C.16. ステートレスサービス

ステートレスサービスは、ユーザーからの指示を受け取る代わりに、タスクを独立して実行する自己完結型のサービスです。さらに、オブジェクトを識別するために膨大な量のデータを使い果たす必要はありません。

バグの報告

C.17. サービスバインディング

サービスバインディングを使用すると、クライアントとサービスをリンクすることで、それらの間でデータを転送できます。

バグの報告

C.18. ENTERPRISE JAVA BEAN

エンタープライズ Java Bean は、エンタープライズアプリケーション用に設計された Java コンポーネントアーキテクチャーです。これを使用して、これらのアプリケーションを作成し、サーバーにデプロイできます。

バグの報告

C.19. ルーズカップリング

疎結合とは、特定のタスクを実行するために 2 つのコンポーネントがリンクされているが、それ以外の時間はリンクされていない状態です。

バグの報告

C.20. 持続メカニズム

持続性メカニズムはフェイルオーバープロパティーです。オブジェクトを永続化します。つまり、シャットダウン後に自動的に再起動し、以前に実行していたタスクを再開できます。

バグの報告

C.21. リソースアダプター

リソースアダプターを使用すると、他のコンポーネントをプラグインできるようにアプリケーションを変更できます。これらのコンポーネントは、アダプターを使用して残りのシステムと通信できるようになります。

バグの報告

C.22. シェルスクリプト

シェルスクリプトは、Red Hat Enterprise Linux などの UNIX ベースのオペレーティングシステム用の一連のコマンドを含むテキストファイルです。実行時にシェル (ターミナル) を呼び出します。Microsoft Windows に相当するものはバッチファイルです。

バグの報告

C.23. WEB コンテナー

Web コンテナーは Java サーブレットと連携します。パフォーマンスを管理し、正しい情報を送受信していることを確認する責任があります。JBoss Enterprise Application Platform は Web コンテナーの一種です。

バグの報告

C.24. 初期コンテキストファクトリー

初期コンテキストファクトリーは、初期コンテキストオブジェクトが作成される場所です。これらのオブジェクトは、名前付けとディレクトリーのプロパティーを作成および表示するために使用されます。

バグの報告

C.25. USERNAMETOKEN

UsernameToken は、単一のセッションで複数回ログインする必要がないように、システム全体にユーザー名 (およびオプションでパスワード) を伝達するために使用されます。

バグの報告

C.26. スキーマの検証

これは、コードが機能することを確認するためにコードを検証またはチェックするプロセスです。スキーマ検証を実行することで、XML コードにエラーがないことを確認できます。

バグの報告

C.27. バイト配列

名前が示すように、これはメモリーなどのオブジェクトを設定するバイト配列です。メッセージの送信および処理パケットで使用するバイト配列を作成できます。

バグの報告

C.28. 拡張トランザクションクライアント

拡張トランザクションクライアントを使用すると、ローカルキューマネージャーからメッセージを送受信できます。また、外部キューマネージャーを表示および更新することもできます。

バグの報告

C.29. 接続プール

接続プーリングは、サーバーを複数のクライアントに接続するバックエンドの方法です。接続プールが作成されると、アプリケーションサーバーはそれを利用して、保存されたアクション (たとえば、データベースに特定のタスクを実行するよう要求する) を実行できます。ユーザーがアクションをデプロイすることを決定したときにアクションが接続プールに入る準備ができているため、一般的なタスクが簡素化されます。

バグの報告

C.30. プールされたデータベースマネージャー

名前が示すように、このマネージャーはプールされたデータベースで動作し、効率的にアクセス、管理、設定できるようにします。

バグの報告

C.31. 暗号変換

この変換を使用して、情報を復号化します。

バグの報告

C.32. 同時制御

この制御方法により、すべてのプロセスが正しく効率的に実行されていることを確認しながら、複数の操作を同時に実行できます。

バグの報告

C.33. 統一資源識別子

Uniform Resource Identifier (URI) は、一連の英数字を使用してシステム内のリソースを識別します。Web URL は URI の一種です。

バグの報告

C.34. プロバイダーアダプター

プロバイダーアダプターを使用すると、アプリケーションはリモートプロバイダーから情報を受け取ることができます。

バグの報告

C.35. 実装クラス

実装クラスは、特定のクラスに属するオブジェクトの実装方法を定義します。

バグの報告

C.36. インターセプタークラス

インターセプタークラスは、クラスで定義されている追加のアクションを実行させるために、オブジェクトに適用されます。

バグの報告

C.37. 取引済フラグ

true または false の値を持つトランザクションフラグを持つようにセッションを設定できます。特定のエンドポイントで true に設定して、それらをトランザクション対応にすることができます。これは、エンドポイントのすべてのアクションを、多数の小さなアクションではなく、1 つの単一のアクションにグループ化できることを意味します。

バグの報告

C.38. JAVA コネクターアーキテクチャー (JCA) トランスポート

Java コネクターアーキテクチャー (JCA) トランスポートは、サービスインテグレータとして機能する Java ベースのアーキテクチャーです。アプリケーションサーバーと企業情報システムをつなぐコネクターです。

バグの報告

C.39. JCA BRIDGE

JCA ブリッジは、接続を開いたり閉じたりできるディスパッチャーです。ユーザーが設定した接続を識別し、コネクターとゲートウェイを検出できます。

バグの報告

C.40. JCA アダプター

JCA アダプターは、アプリケーションサーバーとエンタープライズ情報システムをリンクする仲介者として機能します。

バグの報告

C.41. エンドポイントクラス

エンドポイントクラスを使用すると、ネットワークアドレスを提供することで、ネットワーク上のリソースとサービスを識別できます。

バグの報告

付録D グローバル設定ファイル

D.1. JBOSSESB-PROPERTIES.XML

jbossesb-properties.xml ファイルは、JBoss Enterprise SOA Platform のグローバル設定ファイルです。多くのタスクでは、このファイルを編集する必要があります。このファイルの場所は、システムのインストール方法によって異なります。サーバーデプロイメントをインストールした場合、このファイルは SOA_ROOT/jboss-as/server/PROFILE/deployers/esb.deployer/jbossesb-properties.xml に配置されますが、スタンドアロンクライアントはクラスパスを介して直接アクセスできます。

バグの報告

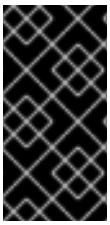
D.2. グローバル設定ファイルのリファレンス

グローバル設定ファイル (SOA_ROOT/jboss-as/server/PROFILE/deployers/esb.deployer/jbossesb-properties.xml) はセクションに分割され、それぞれが設定の特定の領域に関連しています。名前付きプロパティーセクションには、ESB の動作を設定するために使用される 1 つ以上のプロパティーが含まれます。1 つのプロパティーセクションは、別のセクションに依存することができます。依存関係は、PropertyManager によって最初にロードされるセクションを指定します。

コア

- org.jboss.soa.esb.jndi.server.context.factory: JNDI サーバーの初期コンテキストファクトリー。
- org.jboss.soa.esb.jndi.server.url: JNDI サーバーの URL。
- org.jboss.soa.esb.loadbalancer.policy: ESB ロードバランサーポリシー。
- org.jboss.soa.esb.mime.text.types: ペイロードをデコードできるかどうか、またはバイト配列のままにするかどうかを決定するために使用される MIME タイプのセミコロンで区切られたリスト。
- jboss.esb.invm.scope.default: ESB デプロイメントのデフォルトの InVM スコープ。
-

org.jboss.soa.esb.deployment.schema.validation: デプロイ時に JBoss ESB スキーマ検証を有効にする true/false フラグ。



重要

コア内の org.jboss.soa.esb.jndi.server.type および org.jboss.soa.esb.persistence.connection.factory プロパティーは廃止されました。

security

- org.jboss.soa.esb.services.security.implementationClass: 使用される具体的な SecurityService 実装。
- org.jboss.soa.esb.services.security.callbackHandler: デフォルトのコールバックハンドラーの実装。
- org.jboss.soa.esb.services.security.sealAlgorithm: SecurityContext をシールするときに使用されるアルゴリズム。
- org.jboss.soa.esb.services.security.sealKeySize: SecurityContext の暗号化/復号化に使用されるキーのサイズ。
- org.jboss.soa.esb.services.security.contextTimeout: SecurityContext が有効な時間。
- org.jboss.soa.esb.services.security.contextPropagatorImplementationClass: グローバル SecurityContextPropagator を設定するために使用されます。
- org.jboss.soa.esb.services.security.publicKeystore: ESB 外部のデータを暗号化および復号化するためのキーストア。
- org.jboss.soa.esb.services.security.publicKeystorePassword: キーストアのパスワード。
- org.jboss.soa.esb.services.security.publicKeyAlias: 使用する公開 Key Alias。

- **org.jboss.soa.esb.services.security.publicKeyPassword:** 使用する公開鍵パスワード。

- **org.jboss.soa.esb.services.security.publicKeyTransformation:** 使用する暗号変換。

レジストリー

- **org.jboss.soa.esb.registry.queryManagerURI:** サービスおよびバインディングに関する情報を取得するために使用されるレジストリークエリーマネージャー URI。

- **org.jboss.soa.esb.registry.lifeCycleManagerURI:** サービスとバインディングに関する情報を公開するために使用されるレジストリーライフサイクルマネージャーの URI。

- **org.jboss.soa.esb.registry.securityManagerURI:** レジストリーへのクエリーを認証するために使用されるレジストリーセキュリティーマネージャー URI。

- **org.jboss.soa.esb.registry.implementationClass:** JBoss ESB レジストリー実装クラス。ここでは、JAXR レジストリー実装が使用されます。

- **org.jboss.soa.esb.registry.factoryClass:** 使用する JAXR 実装を指定するレジストリーファクトリークラス。

- **org.jboss.soa.esb.registry.user:** レジストリーユーザー。

- **org.jboss.soa.esb.registry.password:** レジストリーのパスワード。

- **org.jboss.soa.esb.scout.proxy.transportClass** UDDI レジストリーとの通信に使用するトランスポートを定義する Scout トランスポートクラス。

- **org.jboss.soa.esb.scout.proxy.uddiVersion:** Scout UDDI バージョン。これは Apache Scout 固有の設定です。

- **org.jboss.soa.esb.scout.proxy.uddiNameSpace:** Scout UDDI 名前空間。これは Apache Scout 固有の設定です。

- org.jboss.soa.esb.registry.interceptors: レジストリーインターセプターのクラス名。
- org.jboss.soa.esb.registry.cache.maxSize: キャッシュレジストリーの最大キャッシュサイズ。
- org.jboss.soa.esb.registry.cache.validityPeriod: キャッシュレジストリーの有効期間。
- org.jboss.soa.esb.registry.orgCategory: 使用する UDDI 組織の値 - これは UDDI 固有の値であることに注意してください。

transports

- org.jboss.soa.esb.mail.smtp.host: SMTP サーバーのホスト名。
- org.jboss.soa.esb.mail.smtp.user: SMTP サーバーに使用するユーザー名。
- org.jboss.soa.esb.mail.smtp.password: SMTP サーバーで指定されたユーザーのパスワード。
- org.jboss.soa.esb.mail.smtp.port: SMTP サーバーのポート番号。
- org.jboss.soa.esb.mail.smtp.auth: AUTH コマンドを使用して SMTP サーバーに対してユーザーを認証するかどうかを指定するフラグ。
- org.jboss.soa.esb.ftp.localdir: FTP ローカルディレクトリー。
- org.jboss.soa.esb.ftp.remotedir: FTP リモートディレクトリー。
- org.jboss.soa.esb.ftp.timeout: ソケットを開くためのミリ秒単位の FTP タイムアウト。
- org.jboss.soa.esb.ftp.timeout.data: データ接続のミリ秒単位の FTP タイムアウト。

- org.jboss.soa.esb.ftp.timeout.so: 現在開いているソケットに使用されるミリ秒単位の FTP タイムアウト。
- org.jboss.soa.esb.ftp.timeout.default: デフォルトのタイムアウトを設定するミリ秒単位の FTP タイムアウト。
- org.jboss.soa.esb.jms.connectionPool: ESB JMS 接続プールのサイズ。
- org.jboss.soa.esb.jms.sessionSleep: JMS セッションを取得できない場合、ESB は取得を試み続けます。sessionSleep プロパティーは、ESB が試行する時間を決定します。
- org.jboss.soa.esb.invm.expiryTime: InVM 一時トランスポート内のメッセージの有効期限。
- org.jboss.soa.esb.invm.retry.limit: 再配信を再試行する最大回数。デフォルトは 5 です。
- org.jboss.soa.esb.ws.returnStackTrace: SOAP メッセージの障害時にスタックトレースを返すかどうかを決定する true/false フラグ。
- org.jboss.soa.esb.ws.timeout: RequestResponseBaseWebService 内で SOAP メッセージを配信するためのサービス呼び出しタイムアウト。
- org.jboss.soa.esb.aggregator.setOnProperties: コンテキストではなくメッセージのプロパティーで集計します。

jca

- org.jboss.soa.esb.jca.activation.mapper.jms-ra.rar: ActivationMapper をグローバルに指定します。
- org.jboss.soa.esb.jca.activation.mapper.wmq.jmsra.rar: ActivationMapper をグローバルに指定します。

dbstore

- org.jboss.soa.esb.persistence.db.conn.manager: Connection Manager 実装クラス名。
- org.jboss.soa.esb.persistence.db.datasource.name: J2EE 接続マネージャーを使用する場合にのみ使用されるデータソース名。
- org.jboss.soa.esb.persistence.db.connection.url: JDBC connection URL.
- org.jboss.soa.esb.persistence.db.jdbc.driver: JDBC ドライバークラス。
- org.jboss.soa.esb.persistence.db.user: データベースユーザー。
- org.jboss.soa.esb.persistence.db.pwd: データベースのパスワード。
- org.jboss.soa.esb.persistence.db.pool.initial.size: データベース接続の初期数。
- org.jboss.soa.esb.persistence.db.min.size: データベース接続の最小数。
- org.jboss.soa.esb.persistence.db.max.size: データベース接続の最大数。
- org.jboss.soa.esb.persistence.db.pool.test.table: データベース接続の有効性を照会するテーブル名。
- org.jboss.soa.esb.persistence.db.pool.timeout.millis: データベース接続のミリ秒単位のタイムアウト。

filters

- org.jboss.soa.esb.filter.1、org.jboss.soa.esb.filter.2、org.jboss.soa.esb.filter.3 など。

rules

-

org.jboss.soa.esb.services.rules.resource.scanner.interval: すべての KnowledgeAgent でグローバルに DRL 変更のポーリング間隔を定義します。

- **org.jboss.soa.esb.services.rules.continueState:** このプロパティーを設定すると、レガシー動作が有効になり、ステートフルルールの実行中にワーキングメモリーが破棄されなくなります。

重要

メッセージルーティングプロパティー (org.jboss.soa.esb.routing.cbrClass) は廃止されましたが、一部のファイルにはまだ存在している可能性があります。

バグの報告

付録E 更新履歴

改訂 5.3.1-103.400 **2013-10-31** **Rüdiger Landmann**
Rebuild with publican 4.0.0

改訂 5.3.1-103 **Tue Feb 05 2013** **David Le Sage**
Content Specification: 6483, Revision: 371693 by dlesage からビルド