JBoss Enterprise SOA Platform 5

管理ガイド

このガイドは、企業システム管理者を対象にしています。

Last Updated: 2023-03-28

JBoss Enterprise SOA Platform 5 管理ガイド

このガイドは、企業システム管理者を対象にしています。

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 の日常的な管理タスク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、Bugzilla: http://bugzilla.redhat.com/ 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 (『Administration_Guide』) を指定してください。

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。

パート I. 基本

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

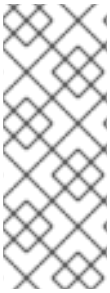
はじめに

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

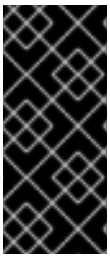
JBoss Enterprise SOA Platform の JBossESB コンポーネントは以下をサポートします。

- 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

第2章 JBOSS ENTERPRISE SOA PLATFORM の紹介

2.1. 対象読者

本書は、JBoss Enterprise SOA Platform を使用して日常の管理、設定、および監視タスクを行う企業システム管理者向けに書かれています。

バグの報告

2.2. 本書のねらい

企業の JBoss Enterprise SOA Platform のインストールを維持するために無数の日常タスクを実行する方法を学ぶために、この本をお読みください。このガイドでは、ユーザー管理、ログ記録、監査、監視、セキュリティー、およびトラブルシューティングのすべてについて説明します。**インストールおよび設定ガイド** の指示に従って、システムがすでにインストールされ、正しく設定されていることを前提としています。

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. 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

2.5. 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

2.6. 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

第3章 テスト環境で JBOSS ENTERPRISE SOA PLATFORM を実行する

3.1. 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3.1 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

3.2. 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3.2 "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

3.3. "HELLO WORLD" クイックスタートのアンデプロイ

手順3.3 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

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

手順3.4 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

第4章 クイックスタート

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

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

4.3. 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:

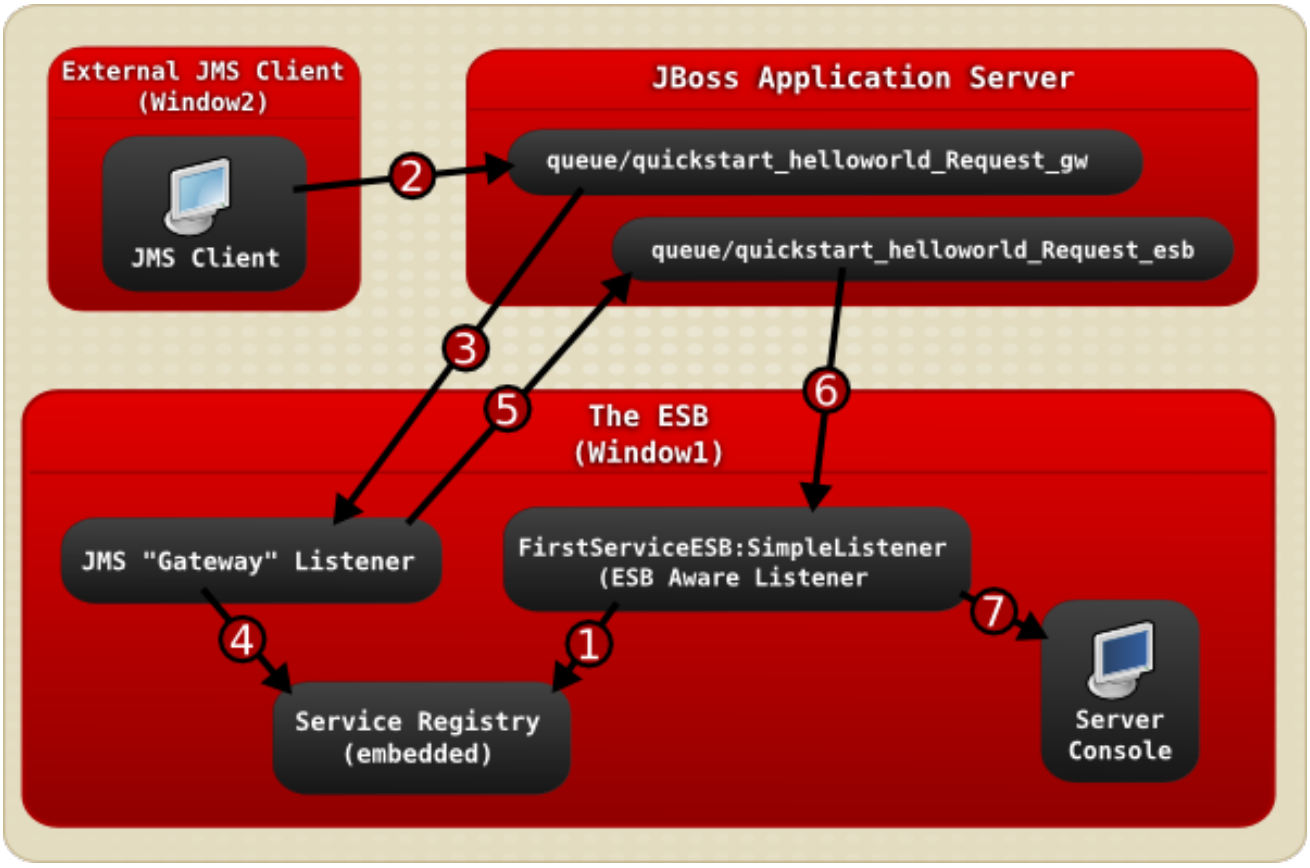
手順4.1 タスク

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

4.4. "HELLO WORLD" クイックスタートの仕組みの概要

図4.1 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

第5章 JBOSS ENTERPRISE SOA PLATFORM を本番環境で実行する

5.1. サーバープロファイル

表5.1 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

5.2. RUN.SH オプションのスイッチ

表5.2 ./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

5.3. 本番環境で JBOSS ENTERPRISE SOA PLATFORM を起動する

手順5.1 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

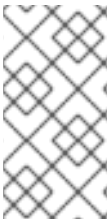
   続行する前に、管理者のユーザー名とパスワードを設定する必要があります。

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

5.4. 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

5.5. 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5.2 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

5.6. 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5.3 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

5.7. 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5.4 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 II. セキュリティー

第6章 ユーザーアカウントの管理

6.1. 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

6.2. ユーザーアカウントの作成

手順6.1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

6.3. 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

6.4. 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

6.5. セキュリティーロール

表6.1 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。

Role	Description
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

6.6. ユーザーのアカウントを無効にする

手順6.2 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

6.7. 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

6.8. SAML セキュリティートークンの発行

手順6.3 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.JBossSTSIssueCallbackHandle
">
        <!-- 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

6.9. SAML セキュリティートークンの検証

手順6.4 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.JBossSTSTokenCallbackHandl
r">
        <!-- 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

6.10. 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

6.11. 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

第7章 システムの保護

7.1. 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

7.2. 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

7.3. 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

7.4. システムを保護する

手順7.1 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

7.5. 暗号化されたパスワードファイルの作成

手順7.2 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

7.6. 暗号化オプション

表7.1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

7.7. 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

7.8. 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

7.9. パスワードのマスキング

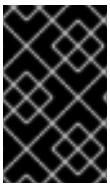
はじめに

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。

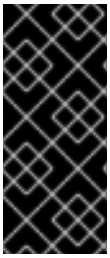


重要

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

7.10. 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。

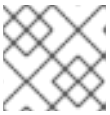
手順7.3 タスク

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

7.11. 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7.4 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

7.12. 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7.5 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</property>
  </bean>
```

3. ファイルを保存して終了します。

バグの報告

7.13. グローバル設定ファイルのセキュリティー設定

表7.2 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
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

プロパティー	Description	必須？
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

7.14. 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

7.15. 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

7.16. 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

7.17. サーバーでシリアル化を有効にする

手順7.6 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

7.18. 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7.7 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.keystore
$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

7.19. 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

7.20. 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

7.21. サービスごとのセキュリティープロパティー

表7.3 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

プロパティー	Description	必須？
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

7.22. グローバルセキュリティー設定を上書きする

手順7.8 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

7.23. セキュリティープロパティーのオーバーライド

表7.4 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？

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

7.24. 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

7.25. 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

7.26. 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

7.27. 認証クラスをメッセージオブジェクトに追加する

手順7.9 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

7.28. 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

7.29. 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7.10 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. 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

7.30. 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7.11 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8章 高度なセキュリティーオプション

8.1. 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

8.2. 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

8.3. SECURITYCONTEXTPROPAGATOR の実装

表8.1 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

8.4. カスタムログインモジュールの追加

手順8.1 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

8.5. 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

8.6. 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8.2 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

8.7. 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8.3 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
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

8.8. 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

8.9. 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

8.10. ロールマッピングを有効にする

手順8.2 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

8.11. 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

8.12. 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

8.13. 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8.3 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

8.14. 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

8.15. 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。

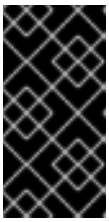
手順8.4 タスク

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9章 サービスレジストリーの保護

9.1. 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

9.2. 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

9.3. 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

9.4. 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

9.5. AUTHTOKEN を取得する

手順9.1 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

9.6. 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

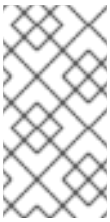
CryptedXMLDocAuthentication の実装は **XMLDocAuthentication** の実装に似ていますが、パスワードは暗号化されています。

```
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

9.7. XMLDOCAUTHENTICATION の設定

手順9.2 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

9.8. 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

9.9. LDAP 認証の設定

手順9.3 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

9.10. JBOSS 認証の設定

手順9.4 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 III. WEB コンソール

第10章 管理者 WEB コンソールでシステムを監視する

10.1. 管理コンソール

JBoss SOA Platform (SOA-P) 管理コンソールは、システム管理者が実行中の SOA-P インスタンス内にデプロイされたサービスを監視および設定できる Web ベースのツールです。

バグの報告

10.2. 管理コンソールの実行

前提条件

- JBoss Enterprise SOA Platform がインストールされ、実行されている必要があります。
- ユーザーの詳細は、次の場所で正しく設定する必要があります。
  - ***SOA_ROOT*/jboss-as/server/*PROFILE*/conf/props/soa-users.properties** および
  - ***SOA_ROOT*/jboss-as/server/*PROFILE*/conf/props/soa-roles.properties**

手順10.1 管理コンソールの実行

1. **Web ブラウザーでコンソールを起動する**
   Web ブラウザーで http://localhost:8080/admin-console を開きます。
2. **コンソールへの認証**
   ***SOA_ROOT*/jboss-as/server/*PROFILE*/conf/props/soa-users.properties** に設定されている **Username** 名と **Password** を入力します。

バグの報告

10.3. 管理コンソールでキューを表示する

手順10.2 タスク

1. Web ブラウザーを起動して、localhost:8080/admin-console にアクセスします。
2. ユーザー名とパスワードとして **admin** を入力します。
3. キューを表示するには、**Resources**、**JBoss Messaging**、**Queues** の順にクリックします。

結果

サーバーにデプロイされた JMS キューのリストが表示されます。

バグの報告

第11章 SERVICE LIST CONSOLE を使用したシステムの監視

11.1. サービスリストコンソール

サービスリストコンソールを使用すると、システム管理者は JBoss Enterprise SOA Platform のサービスレジストリーにリストされているサービスに関する情報を表示できます。http://localhost:8080/contract/ からアクセスします。

バグの報告

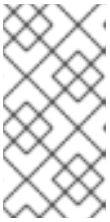
11.2. サービスリストコンソールの機能

サービスリストコンソールには、次の情報が表示されます。

- 処理時間
- 失敗したメッセージの数
- 転送されたバイト数
- 最後に成功したメッセージと失敗したメッセージの日時スタンプ

また、ESB サービスごとに次の情報も表示されます。

- アクションあたりの処理時間
- アクションごとの処理数
- アクションごとの失敗数
- 全体のメッセージ数 (サービスごと)
- コンソールは、メッセージカウンターを介して Enterprise Service Bus を通過したメッセージの数も記録します。(このカウンターは、正常に処理されたメッセージと失敗したメッセージの数も追跡し、処理されたバイト数と各メッセージのタイムスタンプを記録します。)
- また、配信不能メッセージを処理するデッドレターサービスを監視することもできます。



注記

ただし、基礎となるトランスポートがネイティブにサポートされている場合、デッドレターサービスは使用されません。(これは Java Messaging Service の場合です。) このような状況では、配信不能サービスとトランスポート固有の同等のサービスの両方を検査する必要があります。

- コンソールは、Smooks 変換 (およびそれらの実行にかかった時間) を含む、アクションパイプラインで実行されたイベントも追跡します。

バグの報告

第12章 JMX コンソールでシステムを監視する

12.1. JMX コンソール

JMX コンソールは、Java メッセージトランザクションを監視するための Web コンソールです。さまざまなパフォーマンス統計を収集するさまざまな M-Bean をデプロイできます。
http://localhost:8080/jmx-console で JMX コンソールにアクセスします。これにより、アプリケーションサーバーのマイクロカーネルの内部を表示して、登録されているすべてのサービス (M-Bean の形式をとる) を確認できます。

バグの報告

12.2. M-BEAN

M-Bean (Managed Bean) は、サービスやアプリケーションなどの管理可能なリソースを表す Java オブジェクトです。アプリケーションサーバーのマイクロカーネルに登録されているサービスはすべて M-Beans として表されます。

バグの報告

12.3. モニタリングと管理の M-BEAN

jboss.esb> ドメインには、次の M-Bean があります。

deployment=<ESB package name>

Deployments M-Bean を使用して、デプロイされたすべての ESB パッケージとそれに関連付けられた XML 設定のステータスを確認します。

listener-name=<Listener name>

この M-Bean は、デプロイされたすべてのリスナーを一覧表示します。XML 設定、開始時間、maxThreads、および状態に関する情報が表示されます。

リスナーに明示的に管理されたスレッドプールがある場合、その現在の最小および最大スレッドプール数もこの MBean を通じて公開されます。

スレッドプール内のアクティブなスレッドの数は、サービスの負荷に応じて、この最小値 (最初は 1 に設定されています) と定義された最大値の間で動的に変化します。管理者には、システムの実行中にこれらの値を変更するオプションがありますが、サーバーまたは ESB アーティファクトが再起動された場合は元の値に戻ります。

ここから、それらを初期化、開始、停止、および破棄することもできます。

category=MessageCounter

メッセージカウンターには、特定のリスナーに対してデプロイされたすべてのサービス、これらの各サービスのアクション、処理されたメッセージの数、および各メッセージの処理にかかった時間が表示されます。

service-name=<Service name>

この M-Bean は、メッセージ数、状態、平均サイズ、処理時間など、各サービスのさまざまな統計を表示します。メッセージ数のリセットやサービスの開始停止もここから行えます。

注記

上記の M-Beans に加えて、Java Message Service ドメインはいくつかの追加の M-Beans を提供します。これらは、メッセージキューの統計を示します。この情報は、システムをデバッグしたり、そのパフォーマンスを分析したりするときに役立つ場合があります。

バグの報告

第13章 JON FOR SOA WEB コンソールでシステムを監視する

13.1. JBOSS オペレーションネットワーク (JON)

JBoss Operations Network (JON) プログラムは、JBoss Enterprise SOA Platform サーバーの管理に使用できるツールです。

これにより、インベントリー、監視、デプロイメント、およびソフトウェア更新の管理タスクを実行できます。JON はカスタマイズ可能な Web ポータルインターフェイスを利用し、システムを管理できる中心的な場所を提供します。

バグの報告

13.2. SOA の JON

JON for SOA は、JBoss Enterprise SOA Platform 用に特別に設計された機能を含む JON の特別にパッケージ化されたバージョンです。

バグの報告

13.3. JBOSS ENTERPRISE SOA PLATFORM ENTERPRISE SERVICE BUS 統計の分析

手順13.1 タスク

1. **JBoss ESB Statistics** (**Resources** メニューの上) をクリックして、さまざまなレベルの統計をドリルダウンします。

   最初のレベルでは、表示される数値は ESB インスタンス全体の要約です。

2. **JBoss ESB Deployment** 項目をクリックして、サーバーにデプロイされたすべての Enterprise Service Bus パッケージのリストを表示します。このレベルでは統計は表示されませんが、ここからデプロイメントを選択し、ドリルダウンして表示できます。

3. さらにドリルダウンして、そのデプロイメントの設定サービスとアクションの詳細を表示します。

バグの報告

13.4. JON FOR SOA を介して利用可能なメトリクス

選択したレベルに応じて、異なるメトリックが表示されます。次のリストは、利用可能なすべての人をまとめたものです。

ESB レベルで利用可能な統計:

- メッセージ数 (成功)
- メッセージ数 (合計)

- メッセージ数 (失敗)
- 処理されたバイト
- 最後に失敗したメッセージの日付
- 最後に成功したメッセージの日付

サービスレベルで利用可能な統計:

- メッセージ数
- 1分あたりのメッセージ数 (平均)
- 全体のバイト
- 失敗したバイト全体
- 処理された全体のバイト
- 処理された全体のサービス時間

アクションレベルで利用可能な統計:

- メッセージ数
- 1分あたりのメッセージ数 (平均)
- メッセージ失敗
- 1分あたりの失敗したメッセージ (平均)
- 正常に処理されたメッセージ
- 1分あたりの正常に処理されたメッセージ (平均)
- 全体のバイト
- 失敗したバイト全体
- 処理された全体のバイト
- 処理時間

リスナーレベルで利用可能な統計:

- ライフサイクル状態
- 最大スレッド数
- MEP
- サービスカテゴリー
- サービス説明
- サービス名

- Start Date

これらの統計を使用して、JBoss Enterprise SOA Platform デプロイメント、サービス、またはアクションに対して標準の JBoss Operations Network 機能 (アラートなど) を設定します。

バグの報告

13.5. JON FOR SOA を使用してアーカイブをデプロイする

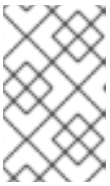
手順13.2 タスク

1. Web ブラウザーを開き、JON for SOA コンソールにログインします。
2. **JBoss ESB Statistics** 画面に移動します。
3. **INVENTORY** タブをクリックします。
4. **Child Resources** に移動します。

注記

過去のデプロイリクエストもここで表示できます。

5. **Create New** メニューに移動し、**JBoss ESB Deployment** を選択します。
6. **Create New Resource** ページで、デプロイするアーカイブを選択し、送信先を選択します (通常、これは **deploy** ディレクトリーになります)。



注記

アップロードできるのは圧縮ファイルのみであることに注意してください。ZIP 形式で **Deploy Zipped** オプションを使用して、圧縮またはデプロイメントされたアーカイブとしてデプロイメントするかどうかを決定します。

バグの報告

13.6. JON FOR SOA を使用してアーカイブを削除する

手順13.3 タスク

1. Web ブラウザーを開き、JON for SOA コンソールにログインします。
2. **JBoss ESB Statistics** 画面に移動します。
3. **INVENTORY** タブをクリックします。
4. **Child Resources** リストに移動します。
5. 削除するエントリーにチェックを入れます。
6. **DELETE** をクリックします。

注記

過去の削除リクエストもここで表示できます。

結果

アーカイブが削除されます。

バグの報告

13.7. 自動サービス検出

JON エージェントは、JON とは無関係にデプロイまたは削除された ESB アーカイブを自動的に検出できます。これは、自動サービス検出機能として知られています。JON エージェントは、このチェックを 24 時間ごとに 1 回実行します。

バグの報告

13.8. 自動サービス検出機能のポーリングレートを変更する

手順13.4 タスク

1. JON エージェントと共にインストールされた設定ファイルをテキストエディターで開きます: **vi rhq-agent/conf/agent-configuration.xml**

   このサンプルコードに従ってファイルを編集します。

   ```
   <entry key="rhq.agent.plugins.service-discovery.period-secs" value="86400"/>
   ```

2. ファイルを保存して終了します。

3. JBoss Enterprise SOA Platform を再起動します。

   JBoss Enterprise SOA Platform コンソールとは対照的に、JON Web コンソールに強制的に新しいデータをオンデマンドで収集させる方法はありません。**Get Current Values** (**Metric Data** タブの下にあります) などのボタンをクリックすると、最近収集されたデータを反映するように表示が更新されるだけです。すぐに更新が必要な場合は、収集期間を非常に短い値 (30 秒など) にリセットします。(後で間隔を前の数値に戻すことを忘れないでください。)

重要

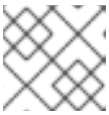
値を低く設定しすぎると、パフォーマンスが低下します。

バグの報告

13.9. 自動サービス検出機能のポーリングレートを変更する (代替方法)

手順13.5 タスク

1. Web ブラウザーを開き、JON for SOA コンソールにログインします。
2. サーバーのリソースのインベントリーに JON エージェントを追加します。
3. **CONFIGURE** をクリックします。
4. **Service Discovery Period** の値を変更します。



注記

変更を有効にするためにエージェントを再起動する必要はありません。

バグの報告

第14章 JUDDI WEB コンソールを使用したサービスレジストリーの管理

14.1. 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

14.2. 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

14.3. 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

14.4. 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14.1 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

14.5. JUDDI M-BEANS

JMX コンソールで jUDDI M.-Beans をクエリーできます。これにより、サービスレジストリー操作を確認できます。以下は、利用可能な M.-Beans です。

- org.apache.juddi.api.impl.UDDIServiceCounter
- org.apache.juddi.api.impl.UDDICustodyTransferCounter
- org.apache.juddi.api.impl.UDDIInquiryCounter
- org.apache.juddi.api.impl.UDDIPublicationCounter
- org.apache.juddi.api.impl.UDDISecurityCounter
- org.apache.juddi.api.impl.UDDISubscriptionCounter

API 下の各 UDDI 操作は、メソッドごとに以下の機能を提供します。

- 正常なクエリー
- 失敗したクエリー
- クエリーの合計
- 処理時間
- API ごとの合計/成功/失敗の集約数

使用できる操作は 1 つだけです (resetCounts)。

バグの報告

第15章 JBPM WEB コンソールでシステムを管理する

15.1. 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

15.2. JBPM WEB コンソール

jBPM コンソールは、JBoss Business Process Manager を管理するための Web ベースのインターフェイスです。http://localhost:8080/jbpm-console/ で入手できます。

バグの報告

第16章 BPEL WEB コンソールを使用したシステムの管理

16.1. BPEL WEB コンソール

BPEL Web コンソールは、以下を表示できるツールです。

- *BPEL エンジン* にデプロイしたすべてのプロセス定義
- BPEL エンジンで実行されているプロセスインスタンス
- プロセスの実行履歴
- 実行履歴に関するクエリー

http://localhost:8080/bpel-console でコンソールを見つけます。

重要

ブラウザーで BPEL コンソールのウィンドウを同時に開くのは 1 つだけにすることをお勧めします。複数開いてしまうと、ログインに失敗したり、ログインしても空白のウィンドウが表示されることがあります。

バグの報告

16.2. 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

16.3. ビジネスルールのオーケストレーション

ビジネスルールオーケストレーションとは、Web サービスを介してビジネスプロセス内のアクションを指定する行為を指します。

バグの報告

16.4. プロセス定義

BPEL プロセス定義は、プロセスのテンプレートとして機能する XML ファイルです。公開されると、プロセス定義はそれ自体が Web サービスであるプロセスを作成します。

バグの報告

16.5. プロセスインスタンス

プロセスインスタンスは、プロセス定義の 1 つの実行です。

バグの報告

16.6. BPEL WEB コンソールを使用してデプロイされたプロセスを表示する

手順16.1 タスク

1. Web ブラウザーを起動して、http://localhost:8080/bpel-console に移動します。
2. ユーザー名とパスワードを入力します。
3. **インスタンスの管理** タブをクリックして、現在デプロイされている BPEL プロセスを確認します。これらの各プロセスのバージョン情報も表示されます。
4. プロセス定義を選択して開きます。下部のパネルに、その特定の定義に対してアクティブなプロセスインスタンスのリストが表示されます。

注記

一度にアクティブにできるプロセスのバージョンは 1 つだけです。プロセス定義を開くと、アクティブなバージョンが自動的に選択されます。

5. 廃止されたバージョンを管理する必要がある場合があります (たとえば、実行中のインスタンスを終了するため)。このような場合は、**More** - **Change Version** をクリックして、目的のバージョンを選択します。

注記

特定のプロセスアーカイブ (**Quickstart_bpel_simple_invoke.jar** など) のバージョンがない場合は、バージョン 0 として扱われます。(この場合、**Quickstart_bpel_simple_invoke-1.jar** が次にデプロイされるバージョンになります。)

バグの報告

16.7. ビジネスプロセス分析フォーマット (BPAF)

Business Process Analytics Format (BPAF) は、組織プロセスの効率と有効性に関する情報を提供するように設計されています。

バグの報告

16.8. BPEL WEB コンソールで BPAF データを表示する

手順16.2 タスク

1. Web ブラウザーを起動して、http://localhost:8080/bpel-console に移動します。
2. ユーザー名とパスワードを入力します。

3. **インスタンスの管理** タブをクリックして、現在デプロイされている BPEL プロセスを確認します。これらの各プロセスのバージョン情報も表示されます。

4. プロセス定義を選択して開きます。下部のパネルに、その特定の定義に対してアクティブなプロセスインスタンスのリストが表示されます。

5. **実行履歴** を使用してグラフを作成します。ここでは、確認する特定の期間を指定し、失敗したインスタンスと終了したインスタンスをグラフに含めるかどうかを選択できます。

バグの報告

16.9. 実行履歴チャートをナビゲートするときに使用するショートカットキーのリスト

表16.1 ナビゲーションキーストローク

キーボードまたはマウスコマンド	結果
上矢印	ズームイン
下矢印	ズームアウト
左矢印	Half-Page Left
右矢印	Half-Page Right
Page-Up	Page Left
Page-Down	Page Right
TAB	Next Focus
Shift-TAB	Previous Focus
HOME	Max Zoom Out
ENTER	Max Zoom In to Focus
マウスドラッグ	スクロールチャート
Shift マウスドラッグ	ドラッグ選択/ズーム
マウスホイールアップ/Z	ズームイン
マウスホイールダウン/X	ズームアウト
バックスペース/戻るボタン	戻る
マウスの右クリック	コンテキストメニュー

キーボードまたはマウスコマンド	結果
左クリック	フォーカスを設定
ダブルクリック	フォーカスへのズームインを最大化

バグの報告

16.10. BPEL WEB コンソールのロギング機能を有効にする

手順16.3 タスク

1. テキストエディターで **deploy.xml** ファイルを開きます (bpel_helloworld クイックスタートの場合、これは **vi SOA_ROOT/jboss-as/samples/quickstarts/bpel_hello_world/bpelContent/deploy.xml になります)。**
2. 以下のようにファイルを編集します。

```
<deploy xmlns="http://www.apache.org/ode/schemas/dd/2007/03"
 xmlns:bpl="http://www.jboss.org/bpel/examples"
 xmlns:intf="http://www.jboss.org/bpel/examples/wsdl">

 <process name="bpl:HelloGoodbye">
  <active>true</active>
       <process-events generate="all"/>
  <provide partnerLink="helloGoodbyePartnerLink">
   <service name="intf:HelloGoodbyeService" port="HelloGoodbyePort"/>
  </provide>
 </process>
</deploy>
```

3. ファイルを保存して終了します。
4. **bpel.properties** 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/deploy/riftsaw.sar/bpel.properties**
5. 記録したい特定のプロセスの process-events で切り替え、org.jboss.soa.bpel.console.bpaf.BPAFLogAdapter が有効になっていることを確認します。
6. ファイルを保存して終了します。

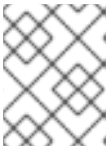
バグの報告

16.11. BPEL WEB コンソールでインスタンスデータを表示する

手順16.4 タスク

1. Web ブラウザーを起動して、http://localhost:8080/bpel-console に移動します。

2. ユーザー名とパスワードを入力します。
3. **インスタンスの管理** タブをクリックして、現在デプロイされている BPEL プロセスを確認します。これらの各プロセスのバージョン情報も表示されます。
4. プロセス定義を選択して開きます。下部のパネルに、その特定の定義に対してアクティブなプロセスインスタンスのリストが表示されます。



注記

一度にアクティブにできるプロセスのバージョンは 1 つだけです。プロセス定義を開くと、アクティブなバージョンが自動的に選択されます。

5. **インスタンスデータ** ボタンをクリックします。
6. **View** タブにはインスタンスの実行グラフが表示され、その下の **Source** タブにはすべてのアクティビティーイベントが表示されます。

バグの報告

16.12. インスタンス実行グラフ

インスタンス実行グラフは、プロセスの実行中のインスタンスを視覚的に表現したものです。これは、プロセスインスタンスの経時的なパフォーマンスをユーザーに伝えます。

バグの報告

16.13. BPEL WEB コンソールでインスタンス実行グラフを表示する

手順16.5 タスク

1. Web ブラウザーを起動して、http://localhost:8080/bpel-console に移動します。
2. ユーザー名とパスワードを入力します。
3. **インスタンスの管理** タブをクリックして、現在デプロイされている BPEL プロセスを確認します。これらの各プロセスのバージョン情報も表示されます。
4. プロセス定義を選択して開きます。下部のパネルに、その特定の定義に対してアクティブなプロセスインスタンスのリストが表示されます。

注記

一度にアクティブにできるプロセスのバージョンは 1 つだけです。プロセス定義を開くと、アクティブなバージョンが自動的に選択されます。

5. **実行パス** ボタンをクリックして、プロセスのインスタンス実行グラフを表示します。

バグの報告

16.14. 履歴インスタンスクエリーの表示

前提条件

- 履歴ログを有効にする必要があります。

手順16.6 タスク

1. BPEL Web コンソールにログインします。
2. リストボックスからプロセス定義とプロセスステータスを選択します。

   オプションで、相関キー、開始時刻、および終了時刻を検索条件として入力することもできます。
3. History Instances List に移動し、行をダブルクリックします。そのプロセスの実行時に発生したすべての実行イベントを示すウィンドウがポップアップ表示されます。

バグの報告

16.15. アクティブなプロセス定義

BPEL プロセス定義の最初のバージョンをデプロイすると、それが自動的にアクティブな定義になります。この定義が後で変更されて再デプロイされると、そのバージョンは廃止され、新しいバージョンが自動的にアクティブになります。

バグの報告

16.16. 廃止されたプロセス定義

アクティブなプロセス定義が変更されて再デプロイされると、古いバージョンは廃止されます。新しいバージョンが自動的にアクティブになります。アクティブなバージョンと廃止されたバージョンの唯一の違いは、廃止されたバージョンでは新しいプロセスインスタンスを作成できないことです。ただし、廃止されたプロセスバージョンに関連付けられたアクティブなプロセスインスタンスがある場合、これらは引き続き実行されます。

バグの報告

16.17. アクティブなプロセス定義を手動で廃止する

手順16.7 タスク

1. Web ブラウザーを起動して、http://localhost:8080/bpel-console に移動します。
2. ユーザー名とパスワードを入力します。
3. **Runtime** タブをクリックします。
4. **Deployment** オプションを選択します。

各プロセス定義のバージョン情報と現在のステータス (アクティブまたは廃止) を確認できるようになりました。

5. 廃止するプロセス定義の特定のバージョンを選択し、**Retire** ボタンを押します。

注記

プロセスをアンデプロイすると、そのエンドポイントは、そのプロセスの以前のバージョンが存在しない場合にのみ非アクティブ化されます。

バグの報告

16.18. エンドポイント参照

エンドポイント参照 (EPR) には、サービスのアドレス情報と技術仕様が含まれています。実際には、すべてのサービスに対応するサービスは、エンドポイント参照を使用して識別されます。これらの参照を通じて、サービスが問い合わせられます。これらはレジストリーに保存されます。サービスは、起動時にレジストリーにエンドポイント参照を追加し、終了時に自動的にそれらを削除する必要があります。サービスには、複数のエンドポイント参照が含まれる場合があります。エンドポイントの参照は、バインディングテンプレートとも呼ばれます。

エンドポイントの参照には、特定のサービスのインターフェイス仕様を指定する tModels へのリンクを含めることができます。

バグの報告

16.19. 廃止されたプロセス定義を手動で再アクティブ化する

手順16.8 タスク

1. Web ブラウザーを起動して、http://localhost:8080/bpel-console に移動します。
2. ユーザー名とパスワードを入力します。
3. **Runtime** タブをクリックします。
4. **Deployment** オプションを選択します。

   各プロセス定義のバージョン情報と現在のステータス (アクティブまたは廃止) を確認できるようになりました。

5. 再アクティブ化する廃止されたバージョンを選択し、**Active** ボタン (画面の右下にあります) を押します。

バグの報告

16.20. プロセスまたは外部 WEB サービスの UTF-8 サポートを有効にする

手順16.9 タスク

1. データベースを調べて、デフォルトで UTF-8 エンコーディングが使用されていることを確認してください。
2. テキストエディターを起動し、データベースの設定ファイルを開きます。
3. 次の設定をファイルに追加します。

```
hibernate.connection.useUnicode=true
hibernate.connection.characterEncoding=UTF-8
```

4. ファイルを保存して終了します。

バグの報告

パート IV. 複数のサーバー設定の管理

第17章 複数の JBOSS ENTERPRISE SOA PLATFORM インスタンスを並べて実行する

17.1. アプリケーションサーバーを並べて実行する

はじめに

JBoss Enterprise SOA Platform は、JBoss Enterprise Application Platform などの別の JBoss 製品と一緒に実行できます。これを実現するには、次の 2 つの方法があります。

- マルチホーミングを使用
- サービスバインディングマネージャーを使用

警告

それぞれが同じデータベース設定を持つ複数のサーバーを同時に起動すると、エラーが発生する場合があります。サーバーは、初めて起動したときに新しいデータベースを初期化します。複数のサーバーが同時にこれを行おうとすると、問題が発生する可能性があります。この問題を回避するには、1 つのインスタンスを起動し、データベースの初期化が完了するまで待ってから、他のインスタンスを起動します。これは、サーバーを初めて起動するときにのみ実行する必要があります。

バグの報告

17.2. マルチホーミングを使用してアプリケーションサーバーを並べて実行する

手順17.1 タスク

1. 複数の IP アドレスが割り当てられるように、オペレーティングシステムのネットワークインターフェイスを設定します。(これを行う手順については、オペレーティングシステムのドキュメントを参照してください)。
2. **-b** スイッチを使用して各サーバーインスタンスを起動して、単一の IP アドレスへにすべてをバインドします。以下に例を示します: **SOA_ROOT/jboss-as/bin/./run.sh -b 10.34.5.2**

バグの報告

第18章 クラスターの管理

18.1. CLUSTER

クラスターとは、疎に接続されたコンピューターのグループであり、単一のシステムとして表示できるように連携します。クラスターのコンポーネント (またはノード) は通常、高速なローカルエリアネットワークを介して相互に接続され、各ノードは独自のオペレーティングシステムを実行します。クラスターリングミドルウェアは、ノード間のコラボレーションを調整し、ユーザーとアプリケーションがクラスターを単一のプロセッサーとして扱うことを可能にします。クラスターは、必要に応じてノードを追加することでパフォーマンスが向上するため、スケーラブルなエンタープライズアプリケーションに効果的です。さらに、1 つのノードに障害が発生した場合はいつでも、他のノードがその負荷を引き継ぐことができます。

バグの報告

18.2. ステートレスサービスフェイルオーバー

JBoss Enterprise SOA Platform は、ステートレスサービスにフェイルオーバー機能を提供します。1 つのノードに障害が発生した場合、サービスは別のノードで再開されます。ServiceInvoker は、フェイルオーバーの複雑さの多くをユーザーから隠しますが、ネイティブの Enterprise Service Bus メッセージでのみ機能し、さらに、すべてのゲートウェイがそれを利用するようにプログラムされているわけではありません。したがって、これらのゲートウェイ実装に送信された非 ESB 認識メッセージは、サービスフェイルオーバーを使用できない場合があります。

バグの報告

18.3. SERVICEINVOKER

ServiceInvoker (**org.jboss.soa.esb.client.ServiceInvoker**) は、指定されたサービスへのメッセージの配信を管理します。また、エンドポイント参照の読み込みと courier の選択も管理するため、メッセージ配信用に一元的なインターフェイスを提供します。

ServiceInvoker は、下位レベルの詳細の多くを非表示にして、ステートレスサービスのフェイルオーバーメカニズムと背後で連携できるので、開発作業を簡素化するために導入されました。そのため、ServiceInvoker は JBoss Enterprise SOA Platform 内でのサービスの使用に推奨されるクライアント側インターフェイスです。

クライアントが対話するサービスごとに ServiceInvoker のインスタンスを作成できます。インスタンスが作成されると、レジストリーを調べてプライマリーエンドポイント参照を特定し、フェイルオーバーの場合は代替エンドポイント参照を特定します。

バグの報告

18.4. ロードバランシング

ロードバランシングは、複数のコンピューターまたはコンピュータークラスター、ネットワークリンク、中央処理装置、ディスクドライブ、またはその他のリソースにワークロードを分散して、最適なリソース使用率を達成し、スループットを最大化し、応答時間を最小化し、過負荷を回避するためのコンピューターネットワーキング手法です。単一のコンポーネントではなく、複数のコンポーネントを負荷分散とともに使用すると、冗長性によって信頼性も向上します。

バグの報告

18.5. 負荷分散ポリシーの設定

手順18.1 タスク

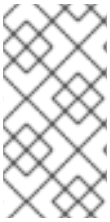
1. グローバル設定ファイルをテキストエディターで開きます: **vi SOA_ROOT/jboss-as/server/PROFILE/deployers/esb.deployers/jbossesb-properties.xml**
2. org.jboss.soa.esb.loadbalancer.policy プロパティーまでスクロールダウンします。使用するポリシーで設定します。
3. ファイルを保存して終了します。

バグの報告

18.6. 負荷分散ポリシー

表18.1 利用可能な負荷分散ポリシー

ポリシー名	Description
最初に入手可能	正常なサービスバインディングが見つかった場合は、それが終了するまで使用されます。リスト内の次のエンドポイント参照が使用されます。このポリシーでは、2 つのサービスインスタンス間の負荷分散は行われません。
ラウンドロビン	各エンドポイント参照がリスト順に使用される標準の負荷分散ポリシー。
ランダムロビン	これはラウンドロビンに似ていますが、選択はランダム化されます。



注記

ポリシーで使用されるエンドポイント参照リストは、デッド EPR が削除されるにつれて、時間の経過とともに小さくなる可能性があります。リストが使い果たされたとき、またはリストキャッシュの time-to-live が上限を超えると、ServiceInvoker はレジストリーから EPR の新しいリストを取得します。

バグの報告

18.7. レジストリーのキャッシュの寿命を変更する

手順18.2 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.deployers/jbossesb-properties.xml**

2. property name="org.jboss.soa.esb.registry.cache.validityPeriod を含むセクションまで下にスクロールします。このプロパティー (タイムアウト値) を必要な値に設定します (デフォルトは 60 秒)。

```
<properties name="core">
<property name="org.jboss.soa.esb.registry.cache.life" value="60000"/>
<!-- 60 seconds is the default -->
</properties>
```

3. ファイルを保存して終了します。

結果

この時間の値を超えると、ServiceInvoker はレジストリーからエンドポイント参照の新しいリストを取得します。

バグの報告

18.8. クラスター内の複数のノードで同じサービスを実行する

手順18.3 タスク

- クラスター内の複数のノードで同じサービスを実行するには、レジストリーのキャッシュが再検証されるまで待ちます。

バグの報告

18.9. 失敗したエンドポイント参照をレジストリーから削除する

手順18.4 タスク

1. **jbossesb-properties.xml** をテキストエディターで開きます: **vi SOA_ROOT/jboss-as/server/PROFILE/deployers/esb.deployers/jbossesb-properties.xml**

2. org.jboss.soa.esb.failure.detect.removeDeadEPR を含むセクションまで下にスクロールします。このプロパティーを true に設定します。

3. ファイルを保存して終了します。

警告

この機能は細心の注意を払って使用する必要があるため、デフォルト設定は false であることに注意してください。これを使用すると、単純に過負荷になっているため応答が遅いサービスのエンドポイント参照が、誤って削除される可能性があります。これらの孤立したサービスとのやり取りはなくなり、再起動が必要になる場合があります。

バグの報告

18.10. BPEL エンジンでのクラスターリングのサポート

クラスター化された環境で BPEL エンジンを正しく動作させるには、環境のすべてのノード間で共有されるデータベースを利用するように設定する必要があります。(これは、BPEL がデータベース内のすべてのプロセス状態を永続化するためです。)

SOAP リクエストをすべてのノードに正しくディスパッチするには、*ロードバランサー*も必要です。

バグの報告

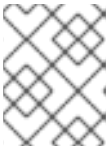
18.11. BPEL クラスターリングの設定

手順18.5 タスク

1. テキストエディターで **jboss-beans.xml** サンプルファイルを開きます。
2. @database@ プロパティーを以下のいずれかに設定します。
   - mysql
   - postgre
   - db2
   - sqlserver
   - oracle
   - sybase
3. ファイルを保存して終了します。
4. **jboss-beans.xml** ファイルを **SOA_ROOT/jboss-as/server/PROFILE/deploy/riftsaw.sar/META-INF/** にコピーします。
5. **SOA_ROOT/jboss-as/server/PROFILE/deploy/cluster/jboss-cache-manager.sar/META-INF/jboss-cache-manager-jboss-beans.xml** を **riftsaw-cache-manager-jboss-beans.xml** に置き換えます。

警告

別の BPEL Engine デプロイメントをインストールしようとすると、統合が壊れる可能性があります。



注記

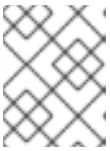
クラスターにデプロイしたサービスを使用する場合は、WSDL ファイルで SOAP アドレスの代わりにロードバランサーの URL を指定します。

バグの報告

18.12. クラスターに BPEL プロセスをデプロイする

手順18.6 タスク

- BPEL アーティファクトをファームディレクトリーにコピーします: **cp FILENAME.jar SOA_ROOT/jboss-as/server/PROFILE/farm**



注記

クラスターリングは、"production" および "all" プロファイルでのみ利用できることに注意してください。

注記

BPEL サービスを呼び出すときに、(WSDL で指定された SOAP アドレスではなく) ロードバランサーの URL を指定します。その後、ロードバランサーは、クラスターのどのサーバーを使用するかを決定します。

バグの報告

パート V. サービスの管理

第19章 出版契約

19.1. サービスリストアプリケーション

重要

Red Hat は、現時点ではサービスリスト機能を **テクニカルプレビュー** としてのみ提供しています。今後のリリースでは、別のテクノロジーに置き換えられる予定です。

Service List Application は、ユーザーがエンドポイントに関する情報を表示できるようにするツールです。(SOAPProcessor アクションによって公開された Web サービスエンドポイントを利用している場合、この情報が必要になることがよくあります)。ツールは http://localhost:8080/contract/ です。このツールは、関連付けられているサービスの下にエンドポイントをグループ化します。

バグの報告

19.2. エンドポイントコントラクト

エンドポイントコントラクトは、エンドポイントが他のサービスと通信する内容を指定します。

バグの報告

19.3. JBOSS ENTERPRISE SOA PLATFORM がエンドポイントコントラクトを検出する方法

JBoss Enterprise SOA Platform は、コントラクト情報を公開できる最初のアクションのアクションパイプラインを調べることで、エンドポイントコントラクトを検出します。

どのアクションも実行できない場合、Service List Application は次のメッセージを表示します。

```
Unavailable on Contract
```

バグの報告

19.4. 契約書を発行する

手順19.1 タスク

1. コントラクト情報を公開するには、次の **org.jboss.internal.soa.esb.publish.Publish** アノテーションをアクションに与える必要があります。(この例では説明目的で SOAPProcessor を使用しています):

```
@Publish(JBossWSWebserviceContractPublisher.class)
public class SOAPProcessor extends AbstractActionPipelineProcessor {
//TODO: implement
}
```

2. **org.jboss.soa.esb.actions.soap.ContractPublisher** インターフェイスを実装します（実装する必要があるのは1つのメソッドだけです）。

```
public ContractInfo getContractInfo(EPR epr);
```

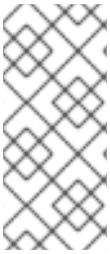
バグの報告

第20章 アーカイブファイルのデプロイメント

20.1. ホットデプロイメント

ホットデプロイメントとは、デプロイされた ESB アーカイブがサーバープロファイルのデプロイディレクトリーに入るとすぐにそれを検出する SOA Platform サーバーの機能を指します。SOA Platform サーバーソフトウェアは、そのディレクトリーを常に監視し、ディレクトリーへの変更を自動的に検出します。また、既存のアーカイブの状態の変化を検出して、それらを自動的に再デプロイメントすることもできます。

これらのアクションには、*ライフサイクルサポート* があります。これは、ホット再デプロイ時に、アクティブなリクエストを終了することによって正常に終了することを意味します。それらは、再起動されるまで、それ以上着信メッセージを受け入れません。(これはすべて自動的に行われます。)



注記

JBoss Enterprise SOA Platform がスタンドアロンモードで実行されている場合、アーカイブをデプロイできません。

ノードはこのファイルのタイムスタンプを監視し、変更が発生した場合はその内容を再度読み取ります。

バグの報告

20.2. ホットデプロイメントと JBOSSESB.SAR

jbossesb.sar アーカイブはライブでデプロイできます。次の場合にデプロイされます。

- タイムスタンプが変更されます (アーカイブが圧縮されている場合)。
- **META-INF/jboss-service.xml** ファイルのタイムスタンプが変更されます (アーカイブがデプロイメント形式の場合)。

バグの報告

20.3. ホットデプロイメントと ESB アーカイブ

***.esb** アーカイブは、次の場合に自動的に再デプロイメントされます。

- アーカイブのタイムスタンプが変更されます (アーカイブが圧縮されている場合)。
- **META-INF/jboss-esb.xml** ファイルのタイムスタンプが変更されます (アーカイブがデプロイメント形式の場合)。

バグの報告

20.4. ルールファイルの再デプロイ

手順20.1 タスク

1. .DRL または .DSL ファイルを再デプロイするには、jbrules.esb ディレクトリーを **SOA_ROOT/jboss-as/server/PROFILE/deploy** にコピーして再デプロイします。

2. または、Action Configuration の ruleReload 機能を有効にすることもできます。この機能を有効にした後、ルールファイルが変更されると、自動的に再ロードされます。

バグの報告

20.5. 変換ファイルの再デプロイ

手順20.2 タスク

1. .ESB アーカイブを **SOA_ROOT/jboss-as/server/PROFILE/deploy** ディレクトリーにコピーして再デプロイします。

2. または、Web ブラウザーを起動して、http://localhost:8080/admin-console Monitoring and Management Console にログインします。

3. Java Message Service を介して通知メッセージを送信します。Smooks はこのイベントを処理し、自動的にリロードします。

バグの報告

20.6. ビジネスプロセス定義の再デプロイ

前提条件

- JBoss Developer Studio

手順20.3 タスク

- jBPM Eclipse プラグインを使用して、新しいバージョンのビジネスプロセス定義を jBPM データベースにデプロイします。

注記

新しいプロセスインスタンスのみがこの新しいバージョンを使用することに注意してください。既存のプロセスライフサイクルでは、以前の定義が引き続き使用されます。

注記

この手順はスタンドアロンモードでも機能することに注意してください。

バグの報告

20.7. スタンドアロンモードでの実行中にルールをリロードする

手順20.4 タスク

- **ruleReload** を実行します。

バグの報告

第21章 外部 WEB サービスと JBOSS ENTERPRISE SOA PLATFORM の統合

21.1. WEB サービス

Web サービスは、2 つのアプリケーションが Web 経由で通信できるようにする方法です。Web サービスは、この目的を達成するためのツールセットで設定されます。Web サービスには、REST 準拠のサービス (Web リソースの XML 表現を操作する目的) と任意の Web サービス (サービスが任意の操作を公開できる) の 2 つのタイプがあります。

バグの報告

21.2. WEB サービスのエンドポイント

Web サービスのエンドポイントとは、Web サービスを実装するソフトウェアです。これらは、サービス指向のアーキテクチャー環境で Web サービス間のメッセージベースの通信を実装するために使用されます。

これらのレジストリーエントリーポイントの外部アプリケーションには、.NET プログラム、その他の外部 Java ベースのアプリケーションサーバー、および LAMP ソフトウェアバンドルを含めることができます。

バグの報告

21.3. WEB SERVICES DESCRIPTION LANGUAGE (WSDL)

Web Services Description Language (WSDL)は、Web サービスインターフェイスの定義に使用される XML ベースの言語です。Web サービスを使用するアプリケーションは、サービスの WSDL ドキュメントを解析し、以下を検出します。

- サービスの場所
- サービスがサポートする操作
- サービスがサポートするプロトコルバインディング (SOAP、HTTP など)
- アクセス手順

各操作について、WSDL はクライアントが準拠する必要のあるインターフェイス形式を記述します。

バグの報告

21.4. REST

REST (Representational State Transfer) は、クライアントとサーバーで設定されるソフトウェアアーキテクチャーの一種です。これは、インターネットを含む分散ネットワーク用に特別に設計されました。これの中心にあるのは、複数のリソース (データソース) の概念であり、それぞれが一意のグローバルアドレスによって識別されます。これらのリソースを操作するために、クライアントとサーバーは標準化されたインターフェイスを介して通信し、データを交換します。

バグの報告

21.5. SOAPPROCESSOR

SOAPProcessor は、JBossESB がホストする任意のリスナーを介して JBossWS がホストする Web サービスエンドポイントを呼び出すことを可能にするアクションです。このアクションを介して、エンタープライズサービスバスは、まだ公開されていないサービスの Web サービスエンドポイントを公開できます。その後、HTTP、FTP、JMS など、エンタープライズサービスバスでサポートされている任意のトランスポートチャネルを介してサービスを呼び出すことができます。

SOAPProcessor は JBossWS-Native と JBossWS-CXF スタックの両方をサポートします。

バグの報告

21.6. SOAPPROXY

SOAPProxy は、外部 Web サービスエンドポイントを消費するアクションです。また、Enterprise Service Bus を介して Web サービスエンドポイントを再公開することもできます。外部サービスと ESB の間に位置するこの仲介者の目的は、次の機能を提供する抽象化レイヤーを提供することです。

- これにより、クライアントとサービスの間の疎結合が促進されます (両者はお互いをまったく認識していないため)。
- これは、クライアントがリモートサービスのホスト名/IP アドレスに直接接続できなくなったことを意味します。
- クライアントは、インバウンド/アウトバウンドパラメーターを変更する変更された WSDL を確認します。少なくとも、現在プロキシーされている元のエンドポイントではなく、ESB の公開されたエンドポイントをクライアントが指すように WSDL を微調整する必要があります。
- インバウンド要求とアウトバウンド応答の両方について、アクションパイプラインを介して SOAP エンベロープ/ボディの変換を導入できます。
- クライアントはエンタープライズサービスバス上の 2 つ以上のプロキシーエンドポイントに接続でき、それぞれに独自の WSDL や変換、ルーティング要件があり、ESB が適切なメッセージを適切なエンドポイントに送信し、究極の対応。
- ContentBasedRouter を介した複雑なコンテキストベースのルーティングが可能になります。

バグの報告

21.7. WEB サービスを ENTERPRISE SERVICE BUS と統合する利点

JBoss Enterprise SOA Platform のエンタープライズサービスバスは、リソースのプロデューサーと最終的にそれを消費するクライアントアプリケーションの間に位置します。ESB の目的は、抽象化レイヤーを提供することです。これにより、次の利点が得られます。

- クライアントとサービスは、互いの存在をまったく認識しないため、疎結合できます。
- クライアントは、リモートサービスのホスト名または IP アドレスに直接接続されなくなります。
- クライアントは、インバウンド/アウトバウンドパラメーターを変更して、変更された WSDL

を確認できます。(少なくとも、クライアントが元のエンドポイントではなく、Enterprise Service Bus によって公開されたエンドポイントを指すように WSDL を変更する必要があることに注意してください。)

- インバウンド要求とアウトバウンド応答の両方に適用されるアクションパイプラインを介して、SOAP エンベロープ/ボディ変換を導入できます。
- クライアントは 2 つ以上のプロキシーエンドポイントに接続できるため、サービスのバージョン管理を実装できます。それぞれに独自の WSDL や変換、ルーティング要件があります。Enterprise Service Bus は適切なメッセージを正しいエンドポイントに送信し、応答を返します。
- **ContentBasedRouter** クラスを使用して、高度なルーティング機能を導入できます。

バグの報告

21.8. WEB サービス統合の設定

手順21.1 タスク

- QE/SME が情報を提供します。

バグの報告

21.9. SOAPPROXY アクションを使用して WEB サービスを再公開する

手順21.2 タスク

- QE/SME が情報を提供します。

バグの報告

21.10. CONTENT-BASED ROUTER

コンテンツベースのルーターは、宛先アドレスを持たないメッセージを正しいエンドポイントに送信します。コンテンツベースのルーティングは、一連のルール (XPath または Drools 表記で定義可能) をメッセージの本文に適用することによって機能します。これらのルールは、どの当事者がメッセージに関心を持っているかを確認します。これは、送信アプリケーションが宛先アドレスを提供する必要がないことを意味します。

一般的な使用例は、優先度の高いキューで優先度の高いメッセージを処理することです。ここでの利点は、そのように設定されている場合、サービスの実行中にルーティングルールをオンザフライで変更できることです。(ただし、これには重大なパフォーマンス上の欠点があります。)

コンテンツベースのルーターが役立つ可能性があるその他の状況には、元の宛先が存在しなくなった場合、サービスが移動した場合、またはアプリケーションが時刻などの要素のコンテンツに基づいてメッセージの送信先をより詳細に制御したい場合などがあります。

バグの報告

21.11. 静的ベースのルーター

静的ベースのルーターは、ネットワーク全体で情報を調整するのに役立ちます。サーバー間で情報を転送し、メッセージの送信先を伝えます。デフォルトが非効率的であると感じた場合は、特定のルートを取るようにプログラムできます。他のサービスへのルーティングにのみ使用できます。

バグの報告

21.12. ルーティングキー

ルーティングキーは、メッセージとそれらがバインドされているアイテムを照合するために使用されます。JBoss Enterprise SOA Platform のコンテキストでは、ルーティングキーをメッセージに適用して、同じルーティングキーが与えられたキューに送信できます。

バグの報告

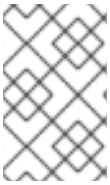
パート VI. システムの監査とトラブルシューティング

第22章 システム監査

22.1. メッセージストア

メッセージストアは、監査追跡を可能にするように設計されたデータベース永続化メカニズムです。メッセージストアは、要求に応じてメッセージの読み取りと書き込みを行います。各メッセージには一意の識別番号が必要です。他の ESB サービスと同様に、メッセージストアはプラグイン可能です。つまり、必要に応じて独自の永続化メカニズムをプラグインできますが、デフォルトのデータベース永続化メカニズムが提供されています。

システム障害が発生した場合、メッセージストアは再配信が必要なメッセージの保持場所としても使用されます。



注記

ファイルの永続化メカニズムなど、データベース以外の何かが必要な場合は、独自のサービスを作成してから、設定を変更してデフォルトの動作をオーバーライドできます。

バグの報告

22.2. サービスルートフィルター

ServiceRouteFilter (**org.jboss.internal.soa.esb.message.filter.ServiceRouteFilter**) は、さまざまなサービスを通じてメッセージのパスを追跡できるようにする監査メカニズムです。他のフィルターと同様に、**jbossesb-properties.xml** ファイル内から有効にする必要があります。

バグの報告

22.3. メッセージストア内のデータを監査する

手順22.1 タスク

1. テキストエディターで **jbossesb-properties.xml** を開きます: **vi jbossesb-properties.xml**
2. フィルターと呼ばれるセクションに移動し、次のコードサンプルに従って編集します。

```
        @lt;properties name="filters"@gt;
        @lt;property name="org.jboss.soa.esb.filter.1"
value="org.jboss.internal.soa.esb.message.filter.MetaDataFilter"/@gt;
        @lt;property name="org.jboss.soa.esb.filter.2"
value="org.jboss.internal.soa.esb.message.filter.GatewayFilter"/@gt;
        @lt;property name="org.jboss.soa.esb.filter.3"
value="org.jboss.internal.soa.esb.message.filter.ServiceRouteFilter"/@gt;
    @lt;/properties@gt;
```

3. ファイルを保存して終了します。

メッセージとサービスをチェックして、サービスルート情報を記録する必要があるかどうかを確認します。サービスごとまたはメッセージごとのレベルで、フィルターにルート情報をコンテキストに追加するように指示できます。

4. サービスレベルで設定するには、サービス定義に recordRoute="true" を追加します。

```
<service
    category="FirstServiceESB"
    name="SimpleListener"
    description="Hello World"
    recordRoute="true">
```

5. メッセージレベルで設定するには、service-record-route プロパティーをメッセージプロパティーに追加して、**true** に設定します。

バグの報告

22.4. TRACEFILTER

TraceFilter (**org.jboss.internal.soa.esb.message.filter.TraceFilter**) は JBoss Enterprise SOA Platform のメタデータフィルターです。そのロールは、メッセージがコンポーネントと対話するたびにログにエントリーを記録することです。イベントを追跡し、それに関する情報を返すことができます。たとえば、特定の種類のメッセージを追跡し、その動きを表示して監視しやすくするように設定できます。

バグの報告

22.5. ログメッセージ

デフォルトでは、コンポーネントとメッセージ間のすべてのやり取りがログに記録されます。ログメッセージには、メインメッセージにリンクするヘッダー情報が含まれています。したがって、複数の SOA インスタンス間で送信されたメッセージを相互に関連付けることができます。

バグの報告

22.6. ログメッセージの特定

手順22.2 タスク

- **メッセージがログメッセージかどうかの判断**
  ログメッセージを識別するには、メッセージを開いて、次の形式に従っているかどうかを確認します。

```
header: [ To: EPR: PortReference < <wsa:Address ftp://foo.bar/> >,
From: null, ReplyTo: EPR: PortReference < <wsa:Address http://bar.
foo/> >, FaultTo: null, Action: urn:dowork, MessageID: urn:foo/bar
/1234, RelatesTo: null ]
```

バグの報告

22.7. ログメッセージのフィルター

手順22.3 タスク

1. **jbossesb-properties.xml ファイルを開く**
   jbossesb-properties.xml をテキストエディターで開きます: **vi SOA_ROOT/jboss-as/server/PROFILE/deployers/esb.deployers/jbossesb-properties.xml**

2. ファイルのフィルターセクションまでスクロールします。

3. org.jboss.soa.esb.messagetrace プロパティーを on に設定します。有効になったので、それを通過するすべてのメッセージがログに記録されます。

4. ログに記録するメッセージと無視するメッセージをより正確に制御するには、org.jboss.soa.esb.permessagetrace プロパティーも on に設定します。これにより、フィルターは org.jboss.soa.esb.message.unloggable プロパティーが yes に設定されているメッセージを無視します。

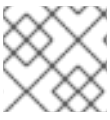
5. **保存**
   ファイルを保存して終了します。

結果

TraceFilter がオンになります。メッセージがこのフィルターを通過するたびに、情報レベルで次のように表示されます。

```
TraceFilter.onOutput ( header: [ To: EPR: PortReference < <wsa:Add
ress ftp://foo.bar/> >, From: null, ReplyTo: EPR: PortReference <
<wsa:Address http://bar.foo/> >, FaultTo: null, Action: urn:dowork
, MessageID: urn:foo/bar/1234, RelatesTo: null ] )
```

```
 TraceFilter.onInput ( header: [ To: EPR: PortReference < <wsa:Addr
ess ftp://foo.bar/> >, From: null, ReplyTo: EPR: PortReference < <
wsa:Address http://bar.foo/> >, FaultTo: null, Action: urn:dowork,
MessageID: urn:foo/bar/1234, RelatesTo: null ] )
```



注記

このフィルターは、入力または出力メッセージには影響しません。

バグの報告

第23章 トラブルシューティング

23.1. JBOSS ENTERPRISE SOA PLATFORM インストールのトラブルシューティング

ここでは、ユーザーが最もよく遭遇する問題のいくつかに対する解決策を示します。

JBOSS_HOME が正しく設定されていない

オプションの環境変数 JBOSS_HOME が設定されている場合、正しいディレクトリーを指している必要があります。複数のインストールがある場合は、実行しようとしているものを指していることを確認してください。

警告

特に必要がない限り、この変数を設定しないでください。

Java が正しくインストールされていない

Java 環境が正しくインストールまたは設定されていない場合、JBoss Enterprise SOA Platform は機能しません。

VM が十分なメモリーを割り当てられない

このエラーは、JBoss Enterprise SOA Platform の要件を満たすのに十分な空きメモリーがシステムで利用できない場合に発生します。アプリケーションを終了する、より多くの仮想メモリーを割り当てる、またはシステムにインストールされている RAM の量を物理的に増やすという 3 つの方法のいずれかで、使用可能な容量を増やすことができます。

バグの報告

23.2. 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

23.3. エンドポイント参照

エンドポイント参照 (EPR) には、サービスのアドレス情報と技術仕様が含まれています。実際には、すべてのサービスに対応するサービスは、エンドポイント参照を使用して識別されます。これらの参照を通じて、サービスが問い合わせられます。これらはレジストリーに保存されます。サービスは、起動時にレジストリーにエンドポイント参照を追加し、終了時に自動的にそれらを削除する必要があります。サービスには、複数のエンドポイント参照が含まれる場合があります。エンドポイントの参照は、バインディングテンプレートとも呼ばれます。

エンドポイントの参照には、特定のサービスのインターフェイス仕様を指定する tModels へのリンクを含めることができます。

バグの報告

23.4. レジストリーサービスのトラブルシューティング

レジストリーはサービスを一覧表示しますが、それらのステータスを判断することはできません。つまり、レジストリーにリストされているエンドポイント参照が有効であるという保証はありません (たとえば、形式が正しくないか、アクティブでなくなったサービスを表している可能性があります)。 これは、JBoss Enterprise SOA Platform が現在ライフサイクル監視機能を備えていないためです。したがって、管理者は無効なエンドポイント参照を手動で更新または削除する必要があります。

重要

ESB サービスは、独自のエンドポイント参照を自動的に作成します。これらのエンドポイントは内部実装であるため、決して変更しないでください。その場合、Red Hat はサポートしません。

バグの報告

23.5. レジストリーからエンドポイント参照を削除する

前提条件

- システムが非アクティブ状態であることを確認する

手順23.1 タスク

1. テキストエディターでエンドポイント参照ファイルを開きます。
2. エンドポイントリファレンスの remove-old-service タグ値を true に設定します。

```
<jms-listener name="JMS-ESBListener"  busidref="quickstartEsbChannel">
      <property name="remove-old-service" value="true"/>
</jms-listener>
```

警告

すべてのエンドポイント参照を含むサービス全体が削除されるため、このオプションは常に注意して使用してください。

3. ファイルを保存して終了します。

バグの報告

23.6. 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

23.7. SERVICE REGISTRY と APACHE SCOUT のトラブルシューティングチェックリスト

- リモートメソッド呼び出しを使用する場合は、必ず **juddi-client.jar** ファイル (**SOA_ROOT/jboss-as./server/PROFILE/deployers/esb.deployer/lib/juddi-client-VERSION.jar**) を取得してください。
- **jbossesb-properties.xml** ファイルがクラスパスにあり、正しく読み取られていることを確認してください。そうでない場合、レジストリーはクラスをインスタンス化するための名前として null を使用しようとします。
- **META-INF/esb.juddi.client.xml** ファイルが有効なトランスポートを指定していることを確認してください。
- **persistence.xml** ファイルの設定が有効であること、および選択した Hibernate ダイアレクトがデータベースで使用されているものと一致していることを確認してください。
- **esb.juddi.xml** ファイルがクラスパスにあることを確認してください。これには、レジストリーの設定設定の一部が含まれています。
- サービスが失敗したり、正常にシャットダウンされなかったりすると、レジストリーに古いエントリーが残ることがあります。これらを手動で削除します。

バグの報告

23.8. その他の SERVICE REGISTRY トラブルシューティングリソース

レジストリーのトラブルシューティングの詳細については、次の場所にアクセスしてください。

- JBoss jUDDI Wiki: http://www.jboss.org/community/docs/DOC-11217
- JBoss ESB ユーザーフォーラム: http://community.jboss.org/en/jbossesb?view=discussions

バグの報告

23.9. 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

23.10. 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

23.11. 診断トレース

診断トレースは、ファイルと設定をトレースして、生産プロセスでエラーが発生したかどうかを確認できるトラブルシューティングの方法です。

バグの報告

23.12. 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23.2 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

23.13. 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23.3 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

パート VII. パフォーマンスチューニング

第24章 パフォーマンスチューニング

24.1. パフォーマンスチューニング

パフォーマンスチューニングは、システムのパフォーマンスを微調整して改善するプロセスです。これは、より多くのデータを処理するように設定を設定したり、パフォーマンスを高速化するために耐荷重機能を実行したりすることで実現できます。

バグの報告

24.2. JBOSS ENTERPRISE SOA PLATFORM を高パフォーマンスに調整する

手順24.1 タスク

- **製品の調整方法を学ぶ**
  パフォーマンスチューニングについては、http://community.jboss.org/wiki/JBossESBPerformanceTuning Web サイトにアクセスしてください。

バグの報告

24.3. レジストリーのパフォーマンス

レジストリーのパフォーマンスと信頼性を向上させるために、Red Hat はレジストリーを複製または統合することをお勧めします。このようにして、単一障害点になることを防ぎます。

バグの報告

24.4. JMS メッセージの優先順位付け

JMS メッセージの優先順位付けは、エンタープライズサービスバスによって処理されるときにどのメッセージが優先されるかを指定できる機能です。ゲートウェイで優先度を設定します。その後、ESB サービスに渡され、その後呼び出されます。ESB サービスとの通信に使用されるすべての JMS トランスポートに優先度を設定する必要があります。

メッセージフローの優先順位は、メッセージコンテキストの一部として、ESB 呼び出しを通じて、呼び出されたサービスに渡されます。ただし、サービスの実行中はメッセージ内に存在しません。代わりに、優先度はスレッドローカルコンテキスト内に保持されるため、後続の ServiceInvoker の呼び出しを通じて優先度を伝播できます。

バグの報告

24.5. JMS メッセージの優先度を設定する

手順24.2 タスク

1. テキストエディターでゲートウェイの設定ファイルを開きます。
2. ファイルのリスナー、バス、またはプロバイダーのいずれかの領域に次のコードを追加します。

```
<property name="messageFlowPriority" value="X"/>
```

x の値は、**0** に **9** 包括的、どこで **0** は優先度が最も低く、**9** は最高です。

3. ファイルを保存して終了します。

バグの報告

24.6. 優先度を設定できるゲートウェイ

次のゲートウェイで JMS メッセージの優先度を設定できます。

- スケジュール済み (ファイルなどを含む)
- Groovy
- JMS (このトランスポートが JMS Courier を介して呼び出される場合、優先順位は MessageProducer の設定にも使用されます。)
- SQL
- JCA 流入
- Camel
- Hibernate
- HTTP
- JBoss Remoting
- UDP
- EBWS (EBWS の場合、プロパティーは設定ファイルの "service" 要素内で指定する必要があります。)

バグの報告

24.7. MESSAGEAWARELISTENER スレッドプールの動的設定

MessageAwareListener スレッドプール内のメッセージスレッドの数は動的に変化します。これは、スレッドの最小数と最大数が必要に応じて変更されることを意味します。この機能は、サービス M-Bean を通じて公開されます。

バグの報告

付録A 便利な定義

A.1. 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

A.2. 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

A.3. 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

A.4. 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

A.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

A.6. 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

A.7. 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

A.8. 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

A.9. 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

A.10. 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

A.11. 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

A.12. 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

A.13. 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

A.14. 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

A.15. 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

A.16. 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

A.17. 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

A.18. 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

A.19. 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

A.20. 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

A.21. 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

A.22. 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

A.23. 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

A.24. 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

A.25. 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

A.26. 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

A.27. 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

A.28. 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

A.29. 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

A.30. 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

A.31. 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

A.32. 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

A.33. 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

A.34. 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

A.35. 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

A.36. 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

A.37. 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、その説明を作成し、サービスブローカーにパブリッシュします。

バグの報告

A.38. 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。サービス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

A.39. サービスリクエスター

サービスリクエスターは、サービスの検出を行います。これは、サービスブローカーによって提供されたサービスの説明を検索して行います。リクエスターは、サービスプロバイダーから取得したサービスをバインドします。

バグの報告

A.40. 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

A.41. 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

A.42. 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

A.43. 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

A.44. 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

A.45. 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

A.46. 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

A.47. 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

A.48. 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

付録B グローバル設定ファイル

B.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。

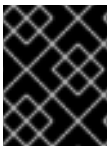
バグの報告

B.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。

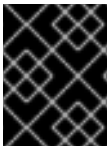
- org.jboss.soa.esb.persistence.db.datasource.name: J2EE 接続マネージャーを使用する場合にのみ使用されるデータソース名。
- org.jboss.soa.esb.persistence.db.connection.url: JDBC connection URL.
- org.jboss.soa.esb.persistence.db.jdbc.driver: JDBC ドライバークラス。
- org.jboss.soa.esb.persistence.db.user: データベースユーザー。
- org.jboss.soa.esb.persistence.db.pwd: 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C ESB アーカイブ

C.1. JAVA アーカイブの種類

JBoss Enterprise Application Platform は、いくつかの異なるタイプのアーカイブファイルを認識します。アーカイブファイルは、デプロイメント可能なサービスとアプリケーションをパッケージ化するために使用されます。

通常、アーカイブファイルは、特定のファイル拡張子と特定のディレクトリー構造を持つ Zip アーカイブです。Zip アーカイブがアプリケーションサーバーにデプロイされる前に抽出された場合、デプロイメントされたアーカイブと呼ばれます。その場合、ディレクトリー名には引き続きファイル拡張子が含まれ、ディレクトリー構造の要件は引き続き適用されます。

表C.1

アーカイブの種類	範囲	目的	ディレクトリー構造の要件
Java アーカイブ	.jar	Java クラスライブラリーが含まれています。	**META-INF/MANIFEST.MF** ファイル (オプション)。どのクラスが **メイン** クラスであるかなどの情報を指定します。
Web アーカイブ	.war	Java クラスとライブラリーに加えて、Java Server Pages (JSP) ファイル、サーブレット、および XML ファイルが含まれます。Web アーカイブのコンテンツは、Web アプリケーションとも呼ばれます。	**WEB-INF/web.xml** ファイル。Web アプリケーションの構造に関する情報が含まれています。他のファイルも **WEB-INF/** に存在する場合があります。
リソースアダプターアーカイブ。	.rar	ディレクトリー構造は JCA 仕様で指定されています。	Java Connector Architecture (JCA) リソースアダプターが含まれています。コネクターとも呼ばれます。
エンタープライズアーカイブ	.ear	1 つまたは複数のモジュールを 1 つのアーカイブにパッケージ化するために Java Enterprise Edition (EE) で使用されます。これにより、モジュールをアプリケーションサーバーに同時にデプロイメントできます。Maven と Ant は、EAR アーカイブの構築に使用される最も一般的なツールです。	1 つ以上の XML デプロイメント記述子ファイルを含む **META-INF/** ディレクトリー。

アーカイブの種類	範囲	目的	ディレクトリー構造の要件
			次のいずれかのタイプのモジュール。 ● Web アーカイブ (WAR)。 ● Plain Old Java Object (POJO) を含む 1 つ以上の Java アーカイブ (JAR)。 ● 独自の **META-INF/** ディレクトリーを含む、1 つ以上の Enterprise JavaBean (EJB) モジュール。このディレクトリーには、デプロイされた永続クラスの記述子が含まれています。 ● 1 つ以上のリソースアーカイブ (RAR)。
サービスアーカイブ	.sar	エンタープライズアーカイブに似ていますが、JBoss Enterprise Application Platform に固有です。	**jboss-service.xml** または **jboss-beans.xml** ファイルを含む **META-INF/** ディレクトリー。

バグの報告

C.2. ESB アーカイブ

ESB アーカイブは特殊なタイプの JAR ファイルです。デプロイ可能な ESB プロジェクトを設定するすべてのファイルが含まれています。

バグの報告

C.3. アーカイブをデプロイする

手順C.1 タスク

- アーカイブをサーバーにデプロイするには、**デプロイ** ディレクトリーにコピーします: **cp FILENAME.esb SOA_ROOT/jboss-as/server/PROFILE/deploy**

結果

ディレクトリーはサーバーによってポーリングされているため、アーカイブはすぐに見つかります。*.war ファイルをアーカイブ形式または非圧縮形式でデプロイすることもできます。

バグの報告

C.4. ESB アーカイブの構造

ESB アーカイブファイルは、次のような多数のファイルで設定されます。

アーカイブのルートレベルには、次のファイルがあります。

***-ds.xml (たとえば、message-store-ds.xml または quickstart-ds.xml)**
これらはデータベーススクリプトです。

***-service.xml (例: jbm-queue-service.xml)**
キューの管理オブジェクトを含むサービス、および上記のスクリプトを使用してデータベースを初期化するサービス

hsqldb
データベースの例。その下には、データベースを作成する **create.sql** ファイルがあります。

アーカイブの **META_INF** ディレクトリーの下に、次のファイルがあります。

deployment.xml
これは、.esb に必要な依存関係をリストします。

jboss-esb.xml
これは、この .esb のデプロイメント記述子です

MANIFEST.MF
マニフェストファイル。

最後に、org/jboss/soa/esb ディレクトリーの下にネストされたカスタムアクションとデータベーススクリプトがあります。

バグの報告

付録D 更新履歴

改訂 5.3.1-69.400 **2013-10-31** **Rüdiger Landmann**
Rebuild with publican 4.0.0

改訂 5.3.1-69 **Tue Feb 05 2013** **David Le Sage**
Content Specification: 6585、 Revision: 371683 by dlesage から構築