



JBoss Enterprise SOA Platform 5

JUDDI レジストリーガイド

これは、開発者にサービスレジストリーを使用するためのガイドです。

JBoss Enterprise SOA Platform 5 JUDDI レジストリーガイド

これは、開発者にサービスレジストリーを使用するためのガイドです。

Enter your first name here. Enter your surname here.

Enter your organisation's name here. Enter your organisational division here.

Enter your email address here.

法律上の通知

Copyright © 2023 | You need to change the HOLDER entity in the en-US/JUDDI_Registry_Guide.ent file |.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

XFS[®] is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js[®] is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack[®] Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

概要

このドキュメントでは、開発者にサービスレジストリーについて説明します。

目次

はじめに	5
1. ドキュメント規則	5
1.1. 表記規則	5
1.2. 引用規則	6
1.3. 注記および警告	7
2. ヘルプの利用とフィードバック提供	7
2.1. ヘルプが必要ですか？	7
2.2. ご意見をお寄せください	8
第1章 はじめに	9
1.1. ビジネス統合	9
1.2. サービス指向アーキテクチャーとは	9
1.3. サービス指向アーキテクチャーの重要なポイント	9
1.4. JBOSS ENTERPRISE SOA PLATFORM とは	10
1.5. SERVICE-ORIENTED ARCHITECTURE PARADIGM	10
1.6. コアおよびコンポーネント	10
1.7. JBOSS ENTERPRISE SOA PLATFORM のコンポーネント	11
1.8. JBOSS ENTERPRISE SOA PLATFORM の機能	11
1.9. JBOSS ENTERPRISE SOA PLATFORM の JBOSS ESB コンポーネントの機能	11
1.10. タスク管理	12
1.11. 統合のユースケース	12
1.12. ビジネス環境での JBOSS ENTERPRISE SOA PLATFORM の使用	13
第2章 JUDDI レジストリーの概要	14
2.1. 対象読者	14
2.2. 本書のねらい	14
2.3. データのバックアップ	14
2.4. JUDDI レジストリー	14
2.5. JUDDI および JBOSS ENTERPRISE SOA PLATFORM	14
2.6. サポートされるその他のサービスレジストリー	15
2.7. SERVICE REGISTRY	15
2.8. サービスプロバイダー	15
2.9. サービスブローカー	15
2.10. サービスリクエスター	16
2.11. WEB サービス	16
2.12. WEB サービスのエンドポイント	16
2.13. WEB SERVICES DESCRIPTION LANGUAGE (WSDL)	16
2.14. UNIVERSAL DESCRIPTION, DISCOVERY AND INTEGRATION (UDDI) レジストリー	16
2.15. UDDI アプリケーションプログラミングインターフェイス	17
2.16. UDDI ページタイプ	18
2.17. SERVICE REGISTRY および JBOSS ENTERPRISE SOA PLATFORM	18
2.18. JUDDI および ESB	18
2.19. レジストリーの仕組み	18
2.20. APACHE スカウト	19
2.21. JAVA API FOR XML REGISTRIES (JAXR)	19
2.22. レジストリーインターフェイス	19
2.23. 変数名 : SOA_ROOT ディレクトリー	19
2.24. 変数名: PROFILE	19
第3章 ルートシードデータの設定	20
3.1. サービスレジストリーパブリッシャー	20
3.2. シードデータ	20

3.3. SERVICE REGISTRY のシードデータファイル	20
3.4. ROOT_PUBLISHER.XML	20
3.5. キー生成スキーマ	21
3.6. TMODEL	21
3.7. ROOT_TMODELKEYGEN.XML	21
3.8. パブリッシャーのキー生成スキーマの作成	21
3.9. JUST ONE KEYGENERATOR TMODEL よりも多くのパブリッシャーを表示する	22
3.10. パブリッシャーを使用したビジネスおよびサービスデータの設定	22
3.11. ROOT_BUSINESSENTITY.XML ファイルでのトークンの設定	24
3.12. シードデータのカスタム化	25
第4章 API を介した単純な公開	26
4.1. UDDI データモデル	26
4.2. ビジネスエンティティー	26
4.3. ビジネスサービス	26
4.4. エンドポイント参照	26
4.5. UDDI および JUDDI API	27
4.6. UDDI モデルへの JUDDI の追加	27
4.7. チュートリアル: 初めてサービスレジストリーを使用する	28
第5章 サブスクリプション	32
5.1. サブスクリプション API	32
5.2. サブスクリプションタイプ	32
5.3. チュートリアル: サブスクリプション	32
5.4. チュートリアル: HELLOSALES サービスのデプロイ	34
5.5. ユーザーが独自のサブスクリプションを作成できるようにする	35
第6章 M-BEAN のサポート	38
6.1. M-BEAN	38
6.2. JUDDI M-BEANS	38
第7章 JUDDI クライアントの使用	39
7.1. JUDDI クライアント	39
7.2. JUDDI クライアント依存関係	39
7.3. JUDDI クライアントおよび JBOSS ENTERPRISE SOA PLATFORM	39
7.4. トランスポート設定	39
7.5. UDDI.XML	39
7.6. カスタム JUDDI クライアント設定のデプロイ	40
7.7. JUDDI クライアント設定ファイルの例	40
7.8. JAVA API FOR XML WEB SERVICES (JAX-WS)	43
7.9. JAX-WS トランスポートの設定	43
7.10. リモート呼び出しクラス	44
7.11. JUDDI とリモートメソッド呼び出し	44
7.12. JUDDI のリモートメソッド呼び出しの有効化	44
7.13. INVM トランスポート	45
7.14. INVM と JUDDI	46
7.15. JUDDI の INVM トランスポートの設定	46
7.16. INVM トランスポートを使用したサービスレジストリーの開始	46
7.17. INVM トランスポートを使用したサービスレジストリーの停止	47
7.18. UDDI アノテーション	47
7.19. UDDISERVICE アノテーションの使用	47
7.20. UDDISERVICE 属性	47
7.21. UDDISERVICEBINDING アノテーションの使用	48
7.22. UDDISERVICEBINDING 属性	48

7.23. UDDI アノテーションの使用例	49
7.24. CATEGORYBAG 属性	50
7.25. CATEGORYBAG アノテーションの使用例	50
7.26. アノテーションを使用した WEB サービスの開発	51
7.27. キーテンプレート	52
7.28. キーテンプレートのプロパティ	52
7.29. アプリケーションでの JUDDI クライアントコードの使用	53
7.30. WSDL および UDDI レジストリー	53
7.31. WSDL におけるサービスとバインディングの説明の仕様	53
7.32. JUDDI での WSDL プロセスの登録	54
7.33. サービスレジストリーからの WSDL バインディングの削除	54
7.34. JUDDI での BPEL プロセスの登録	55
7.35. URLLOCALIZER インターフェイスおよび JBOSS ENTERPRISE SOA PLATFORM	55
7.36. 動的 UDDI サービスルックアップ	56
7.37. SERVICE LOCATOR を使用したサービスバインディングの検索	56
7.38. チュートリアル: JUDDI クライアントの使用	56
 付録A 更新履歴	 58

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。

プロポーションアルボールド

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

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

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

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

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

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

Publican は *DocBook* 公開システムです。

1.2. 引用規則

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

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

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

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

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

    mutex_lock(&kvm->lock);

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

```

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

kvm_deassign_device(kvm, match);

kvm_free_assigned_device(kvm, match);

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

```

1.3. 注記および警告

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



注記

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



重要

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



警告

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

2. ヘルプの利用とフィードバック提供

2.1. ヘルプが必要ですか？

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

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

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

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

2.2. ご意見をお寄せください

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

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

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

第1章 はじめに

1.1. ビジネス統合

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

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

バグの報告

1.2. サービス指向アーキテクチャーとは

はじめに

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

すでに分かっているように、ハードウェアバスは、複数のシステムとサブシステムを結び付ける物理コネクタです。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 (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-oriented 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 5 準拠のアプリケーションサーバー (JBoss Enterprise Application Platform)
- エンタープライズサービスバス (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 JBOSS ESB 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ため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 JUDDI レジストリーの概要

2.1. 対象読者

本書は、JBoss Enterprise Service Platform のサービスレジストリーの使用方法を学習したい経験のある Java 開発者向けに書かれています。

[バグの報告](#)

2.2. 本書のねらい

jUDDI Service Registry の開発および操作方法については、本書をお読みください。本ガイドには、実稼働環境で使用する製品を最初に設定およびセキュリティー保護する方法が示されていません。詳細は、完全な [インストールおよび設定ガイド](#) を参照してください。

[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. JUDDI レジストリー

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

[バグの報告](#)

2.5. 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](#)

2.6. サポートされるその他のサービスレジストリー

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

- SOA ソフトウェア SMS
- HP Systinet

[バグの報告](#)

2.7. 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](#)

2.8. 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](#)

2.9. 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](#)

2.10. 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](#)

2.11. 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](#)

2.12. 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](#)

2.13. 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](#)

2.14. UNIVERSAL DESCRIPTION, DISCOVERY AND INTEGRATION (UDDI) レジストリー

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

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

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

バグの報告

2.15. UDDI アプリケーションプログラミングインターフェイス

UDDI v3 仕様は、9 つの API を定義します。

UDDI_Security_PortType

これは、セキュリティトークンを取得するための API を定義します。有効なセキュリティトークンでは、パブリッシャーはレジストリーに公開できます。セキュリティトークンは、セッション全体に使用できます。

UDDI_Publication_PortType

これにより、ビジネスおよびサービス情報を UDDI レジストリーに公開する API が定義されます。

UDDI_Inquiry_PortType

これは、UDDI レジストリーをクエリーする API を定義します。この API は通常、セキュリティトークンを必要としません。

UDDI_CustodyTransfer_PortType

この API は、ある UDDI ノードから別の UDDI ノードにビジネスのカストディを転送するために使用することができます。

UDDI_Subscription_PortType

これは、特定のサービスビジネスで更新を登録する API を定義します。

UDDI_SubscriptionListener_PortType

これは、UDDI ノードからサブスクリプション通知を受け取るためにクライアントが実装する必要のある API を定義します。

UDDI_Replication_PortType

これは、UDDI ノード間でレジストリーデータを複製する API を定義します。

UDDI_ValueSetValidation_PortType

これは、外部プロバイダーが検証を設定するのを許可するためにノードによって使用されます。keyedReferences または keyedReferenceGroups が有効かどうかを評価する Web サービス。

UDDI_ValueSetCaching_PortType

UDDI ノードは、このような Web サービスから取得したキャッシュされた値を使用して、パブリッシャー参照自体を検証することができます。

[バグの報告](#)

2.16. UDDI ページタイプ

緑色のページ

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

黄色のページ

黄色のページは、所属する業界に基づいて企業を分類するために使用されます。

ホワイトページ

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

[バグの報告](#)

2.17. SERVICE REGISTRY および JBOSS ENTERPRISE SOA PLATFORM

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

[バグの報告](#)

2.18. JUDDI および ESB

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

[バグの報告](#)

2.19. 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](#)

2.20. 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](#)

2.21. JAVA API FOR XML REGISTRIES (JAXR)

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

[バグの報告](#)

2.22. レジストリーインターフェイス

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

[バグの報告](#)

2.23. 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](#)

2.24. 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 ルートシードデータの設定

3.1. サービスレジストリーパブリッシャー

パブリッシャーは、Service Registry のアイデンティティ管理システムです。デフォルトでは、2つのパブリッシャーがインストールされます。

root

ルートパブリッシャーは、ルートパーティションおよび UDDI サービスの所有者です(**inquiry** や **publication** など)。各レジストリーには **ルートパブリッシャー**が必要です。ノードごとにルートパブリッシャーは1つだけ存在できます。jUDDI レジストリーは、juddi-core-3.x.jar ファイルの **juddi-core-3.x.jar** ファイルの **juddi_install_data/** ディレクトリーにあるルートアカウントのデフォルトのシードデータを提供します。

uddi

UDDI は、他のすべてのシードデータを所有します (事前定義された **tModels** など)。

[バグの報告](#)

3.2. シードデータ

シードデータは、システムにフィードする初期データです。テンプレートとして機能します。

[バグの報告](#)

3.3. SERVICE REGISTRY のシードデータファイル

パブリッシャーごとに、4つのシードデータファイルがあります。これらは、jUDDI を初めて起動するときに読み込まれます。

- <publisher>_Publisher.xml
- <publisher>_tModelKeyGen.xml
- <publisher>_BusinessEntity.xml
- <publisher>_tModels.xml

[バグの報告](#)

3.4. ROOT_PUBLISHER.XML

root_Publisher.xml データシードファイルの内容は以下のようになります。

```
<publisher xmlns="urn:juddi-apache-org:api_v3" authorizedName="root">
  <publisherName>root publisher</publisherName>
  <isAdmin>true</isAdmin>
</publisher>
```


[バグの報告](#)

3.5. キー生成スキーマ

名前が示すように、キージェネレータスキーマはキーを生成します。これを使用すると、他のパブリッシャーによって生成されたキーと同じキーを作成するリスクを回避できます。各パブリッシャーには、常に独自のキージェネレータスキーマが必要です。ルートパブリッシャーは KeyGenerator を 1 つだけ所有できますが、他のパブリッシャーは複数のキーを所有できます。

[バグの報告](#)

3.6. TMODEL

tModel（テクノロジープレビューモデル）は、レジストリーに登録されているサービスの 1 つを表す UDDI データ構造です。これらは、キーパーティションに登録するためのバインディングの技術詳細（プロトコルやトランスポートなど）を説明する、主要なエンティティのいずれかからあらゆることを実行できるキャッチオール構造です。

[バグの報告](#)

3.7. ROOT_TMODELKEYGEN.XML

tModel Key Generator は SOA_ROOT/jboss-as/server/PROFILE/deploy/jbossesb-registry.sar/juddi_custom_install_data/root_tModelKeyGen.xml ファイルで定義されます。

```
<tModel tModelKey="uddi:juddi.apache.org:keygenerator" xmlns="urn:uddi-org:api_v3">
  <name>uddi-org:keyGenerator</name>
  <description>Root domain key generator</description>
  <overviewDoc>
    <overviewURL useType="text">
      http://uddi.org/pubs/uddi_v3.htm#keyGen
    </overviewURL>
  </overviewDoc>
  <categoryBag>
    <keyedReference tModelKey="uddi:uddi.org:categorization:types"
      keyName="uddi-org:types:keyGenerator"
      keyValue="keyGenerator" />
  </categoryBag>
</tModel>
```

つまり、ルートパブリッシャーによって使用されるキーは **uddi:juddi.apache.org:<text-of-choice>** の形式で指定する必要があります。

[バグの報告](#)

3.8. パブリッシャーのキー生成スキーマの作成

手順3.1 タスク

1. 各パブリッシャーは、独自の KenGenerator tModel を定義する必要があります。tModel Key Generator は root_tModelKeyGen.xml ファイルで定義されます。

```
<tModel tModelKey="uddi:juddi.apache.org:keygenerator" xmlns="urn:uddi-org:api_v3">
  <name>uddi-org:keyGenerator</name>
  <description>Root domain key generator</description>
  <overviewDoc>
    <overviewURL useType="text">
      http://uddi.org/pubs/uddi_v3.htm#keyGen
    </overviewURL>
  </overviewDoc>
  <categoryBag>
    <keyedReference tModelKey="uddi:uddi.org:categorization:types"
      keyName="uddi-org:types:keyGenerator"
      keyValue="keyGenerator" />
  </categoryBag>
</tModel>
```

2. ルートパブリッシャーが使用するキーは、**uddi:juddi.apache.org:<text-of-choice>** 形式である必要があります。別の形式を使用する場合は、不正なキーエラーが表示されます。



警告

KeyGenerators を共有しないでください。

バグの報告

3.9. JUST ONE KEYGENERATOR TMODEL よりも多くのパブリッシャーを表示する

手順3.2 タスク

- **<publisher>_tModels.xml** ファイルを使用します。

バグの報告

3.10. パブリッシャーを使用したビジネスおよびサービスデータの設定

手順3.3 タスク

1. テキストエディターで **<publisher>_BusinessEntity.xml** ファイルを開きます。vi
SOA_ROOT/jboss-as/server/PROFILE/deploy/jbossesb-registry.sar/juddi_custom_install_data/root_BusinessEntity.xml
2. ファイルを編集し、サービスを指定します。以下に例を示します。

```

<businessEntity xmlns="urn:uddi-org:api_v3"
  xmlns:xml="http://www.w3.org/XML/1998/namespace"
  businessKey="uddi:juddi.apache.org:businesses-asf">
  <!-- Change the name field to represent the name of your registry -->
  <name xml:lang="en">An Apache jUDDI Node</name>
    <!-- Change the description field to provided
    a brief description of your registry -->
    <description xml:lang="en">
      This is a UDDI v3 registry node as implemented by Apache jUDDI.
    </description>
    <discoveryURLs>
      <!-- This discovery URL should point to the home installation URL of jUDDI -->
      <discoveryURL useType="home">
        http://${juddi.server.baseurl}/juddiv3
      </discoveryURL>
    </discoveryURLs>
    <categoryBag>
      <keyedReference tModelKey="uddi:uddi.org:categorization:nodes" keyValue="node" />
    </categoryBag>

    <businessServices>
      <!-- As mentioned above, you may want to provide user-defined keys for
      these (and the services/bindingTemplates below.  Services that you
      don't intend to support should be removed entirely -->
      <businessService serviceKey="uddi:juddi.apache.org:services-inquiry"
        businessKey="uddi:juddi.apache.org:businesses-asf">
        <name xml:lang="en">UDDI Inquiry Service</name>
        <description xml:lang="en">Web Service supporting UDDI Inquiry API</description>
        <bindingTemplates>
          <bindingTemplate bindingKey="uddi:juddi.apache.org:servicebindings-inquiry-ws"
            serviceKey="uddi:juddi.apache.org:services-inquiry">
            <description>UDDI Inquiry API V3</description>
            <!-- This should be changed to the WSDL URL of the inquiry API.
            An access point inside a bindingTemplate will be found for every service
            in this file.  They all must point to their API's WSDL URL -->
            <accessPoint useType="wsdlDeployment">
              http://${juddi.server.baseurl}/juddiv3/services/inquiry?wsdl
            </accessPoint>
            <tModelInstanceDetails>
              <tModelInstanceInfo tModelKey="uddi:uddi.org:v3_inquiry">
                <instanceDetails>
                  <instanceParms>
                    <![CDATA[
                      <?xml version="1.0" encoding="utf-8" ?>
                      <UDDIinstanceParmsContainer
                        xmlns="urn:uddi-org:policy_v3_instanceParms">
                        <defaultSortOrder>
                          uddi:uddi.org:sortorder:binarysort
                        </defaultSortOrder>
                      </UDDIinstanceParmsContainer>
                    ]]>
                  </instanceParms>
                </instanceDetails>
              </tModelInstanceInfo>
            </tModelInstanceDetails>
          </bindingTemplate>
        </bindingTemplates>
      </businessService>
    </businessServices>
  </businessEntity>
  <categoryBag>

```

```

        <keyedReference keyName="uddi-org:types:wsdl"
keyValue="wsdlDeployment"
        tModelKey="uddi:uddi.org:categorization:types"/>
    </categoryBag>
</bindingTemplate>
</bindingTemplates>
</businessService>
<businessService serviceKey="uddi:juddi.apache.org:services-publish"
    businessKey="uddi:juddi.apache.org:businesses-asf">
    <name xml:lang="en">UDDI Publish Service</name>
    .....
</businessService>
</businessServices>
</businessEntity>

```

3. ファイルを保存して終了します。



警告

juddi.seed.always を **true** に設定しない限り、シードプロセスは、データベースに **パブリッシャー** がない場合にのみ開始します。(ルートデータファイル以外のすべてのファイルを再適用します。)

これにより、再シードされたエンティティーに追加された情報は消去されるため、データが失われる可能性があります。これは、データがマージされないためです。

バグの報告

3.11. ROOT_BUSINESSENTITY.XML ファイルでのトークンの設定

手順3.4 タスク

1. root_BusinessEntity.xml ファイル (\${juddi.server.baseurl}) でトークンの値を設定するには、**SOA_ROOT/jboss-as/server/PROFILE/deploy/jbossesb-registry.sar/esb.juddi.xml** ファイルを変更します。テキストエディターで **<publisher>_BusinessEntity.xml** ファイルを開きます。vi **SOA_ROOT/jboss-as/server/PROFILE/deploy/jbossesb-registry.sar/juddi_custom_install_data/root_BusinessEntity.xml**
2. ここでは、キーおよびセキュリティトークンを設定できます。これらのセクションは、ニーズに合わせて変更します。
3. ファイルを保存して終了します。



注記

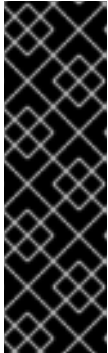
値の置き換えは実行時に行われるため、必要に応じて異なるノードを独自の値に置き換えることができます。

[バグの報告](#)

3.12. シードデータのカスタム化

手順3.5 タスク

•



重要

このシードデータはデータベースの設定に使用されるため、Service Registry を初めて起動する前にこれを実行する必要があります。

プロセスを再実行する場合は、作成したデータベースを削除します。

SOA_ROOT/jboss-as/server/PROFILE/deploy/jbossesb-registry.sar/esb.juddi.xml ファイルでトークンを設定するのを忘れないでください。

カスタムシードデータを **juddiv3.war/WEB-INF/classes/juddi_custom_install_data/** ディレクトリに追加します。



注記

Sales Seed Data を使用するには、次のコマンドを実行して **SOA_ROOT/jboss-as/server/PROFILE/deploy/jbossesb-registry.sar/WEB-INF/classes/** にあるディレクトリの名前を変更します: **mv RENAME4Sales_juddi_custom_install_data juddi_custom_install_data**



注記

ルートパブリッシャーが **root** と呼ばれていない場合は、**SOA_ROOT/jboss-as/server/PROFILE/deploy/jbossesb-registry.sar/esb.juddi.xml** ファイルの **juddi.root.publisher** プロパティを以下以外に設定する必要があります。

```
juddi.root.publisher=root
```

プロパティは以下のように設定されます。

```
<entry key="juddi.root.publisher">root</entry>
```

[バグの報告](#)

第4章 API を介した単純な公開

4.1. UDDI データ モデル

UDDI は、データを以下の階層に編成します。

ビジネスエンティティ

ビジネスエンティティは pyramid の上部にあります。ビジネスサービスが含まれます。

Business services

ビジネスサービスには、エンドポイントの参照が含まれます。

この階層と一貫性のある形式でデータをモデル化するには、まずサービスを公開する前にビジネスエンティティを作成する必要があります。また、バインディングテンプレートを公開する前にビジネスサービスを作成する必要があります。

[バグの報告](#)

4.2. ビジネスエンティティ

ビジネスエンティティは、サービスを担当する組織単位です。サービスを公開する当事者に関する情報(説明と連絡先情報など)が含まれます。ビジネスエンティティの使用法の選択方法は、状況によって異なります。小規模な会社の場合は、1つのエンティティのみが必要になる場合があります。複数の部門を持つ大規模な企業の場合は、部門ごとに1つのエンティティを用意することができます。

もう1つの可能性は、部署を表す複数の子エンティティを持つ1つのオーバーアーキテクチャーエンティティを持つことです。これを行うには、パブリッシャーアサーションを使用してエンティティ間の関係を作成します。

[バグの報告](#)

4.3. ビジネスサービス

ビジネスサービスは、クライアントが使用する機能の単位を表します。UDDI では、主に名前、説明、カテゴリなどの説明情報で設定されます。サービスに関する技術的な詳細は、実際にはそのバインディングテンプレートに含まれています。

[バグの報告](#)

4.4. 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

4.5. UDDI および JUDDI API

UDDI API を使用するには、その設定方法を理解する必要があります。これは複数の **セット** に分類されます。各 **セット** は、機能の特定エリアを表します。

- **quiry** は、エンティティの詳細についてレジストリーへのクエリーを処理します。
- **公開** はエンティティの登録を処理します。
- **セキュリティ** は、認証を処理するオープンな仕様です。
- **Custody および Ownership Transfer** はエンティティの所有権の移譲を処理します。
- **サブスクリプション** を使用すると、クライアントはサブスクリプション形式を使用してエンティティに関する情報を取得できます。
- **Subscription Listener** は、サブスクリプションの結果を受け入れるためのクライアント API です。
- **値 セット(Validation および Caching)** は、キー化された参照値を検証します。(このセットは jUDDI では利用できません。)
- **レプリケーション** を使用すると、レジストリーノード間でデータをフェデレーションできます。(このセットは jUDDI では利用できません。)

最も頻繁に使用される **セット** は、**Inquiry**、**公開**、および **セキュリティ** です。これらは、レジストリーとの対話に必要な標準的な機能を提供します。

jUDDI サーバーは、これらの各 **セット** を JAX-WS 準拠の Web サービスとして実装します。(セットで定義された各メソッドは、対応する Web サービスのメソッドです。)

jUDDI は、トランスポートクラスを使用して各呼び出しがどのように行われるかを定義するクライアントモジュールを提供します。(デフォルトのトランスポートは JAX-WS を使用しますが、API を呼び出す方法は複数あります。)

最後に、jUDDI は独自の API セットも定義します。この API セットには、ほとんどが jUDDI のサブスクリプションモデルに関連するパブリッシャーおよびその他のメンテナンス機能の処理を扱うメソッドが含まれています。この API セットは jUDDI に固有のもので、UDDI 仕様に準拠していません。

バグの報告

4.6. UDDI モデルへの JUDDI の追加

はじめに

jUDDI レジストリーの実装は、仕様に記載されている以外の追加の構造を持つデータモデルを提供します。これはパブリッシャーの概念です。

UDDI 仕様では、レジストリー内に公開されるエンティティの所有権について説明しますが、この所有権の実装方法は含まれません。これは個別の実装に留まります。

パブリッシャーは、jUDDI レジストリーにこの機能を提供するアイデンティティ管理システムです。

アセットを jUDDI に追加する前に、レジストリーは承認トークンを取得できるようにしてパブリッシャーを認証します。このトークンは後続の **publish** 呼び出しに割り当てられ、公開されたエンティティに所有権を割り当てます。

後者のカスタム API を使用して、パブリッシャーを jUDDI レジストリーに保存できます。

パブリッシャーは、すぐに使える ID 管理システムの実装です。jUDDI を使用すると、必要に応じてパブリッシャーを完全に回避して、独自の ID 管理システムに統合することもできます。本ガイドでは、すぐに使えるシンプルなパブリッシャーソリューションを使用します。

バグの報告

4.7. チュートリアル: 初めてサービスレジストリーを使用する

このチュートリアルでは、簡単な例を追って説明します。この例では、認証トークンの取得後に、Publisher、BusinessEntity、および BusinessService が jUDDI に登録されます。

手順4.1 タスク

1. **SimplePublish** クラスをインスタンス化します。コンストラクターは次のとおりです。

```
public SimplePublish()
{
    try
    {
        String clazz = UDDIClientContainer.getUDDIClerkManager(null).
            getClientConfig().getUDDINode("default").getProxyTransport();
        Class<?> transportClass = ClassUtil.forName(clazz, Transport.class);
        if (transportClass!=null)
        {
            Transport transport = (Transport) transportClass.
                getConstructor(String.class).newInstance("default");

            security = transport.getUDDISecurityService();
            juddiApi = transport.getJUDDIApiService();
            publish = transport.getUDDIPublishService();
        }
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
}
```

このコンストラクターは jUDDI クライアント API を使用して、デフォルトのノードからトランスポートを取得します。この例では、JAX-WS Web サービスを使用して UDDI 呼び出しを行うように設計されたデフォルトのクライアントトランスポートクラスを取得するだけです。

2. 例に必要な 3 つの API セットを取得します。

- 承認トークンを取得できるように **Security** API セット
- パブリッシャーを保存できるようにプロプライエタリーの **jUDDI** API セット
- エンティティーを jUDDI に登録できるように **公開** API が設定されている。

以下は、**publish** メソッドの最初の数行です。

```
// Setting up the values to get an authentication token for the 'root' user
// ('root' user has admin privileges and can save other publishers).
GetAuthToken getAuthTokenRoot = new GetAuthToken();
getAuthTokenRoot.setUserID("root");
getAuthTokenRoot.setCred("");

// Making API call that retrieves the authentication token for the 'root' user.
AuthToken rootAuthToken = security.getAuthToken(getAuthTokenRoot);
System.out.println ("root AUTHTOKEN = " + rootAuthToken.getAuthInfo());
```

このコードは、*root* ユーザーの承認トークンのみを取得します。

3. パブリッシャーを追加します。

```
// Creating a new publisher that we will use to publish our entities to.
Publisher p = new Publisher();
p.setAuthorizedName("my-publisher");
p.setPublisherName("My Publisher");

// Adding the publisher to the "save" structure, using the 'root' user authentication
// info and saving away.
SavePublisher sp = new SavePublisher();
sp.getPublisher().add(p);
sp.setAuthInfo(rootAuthToken.getAuthInfo());
juddiApi.savePublisher(sp);
```

上記のコードは、jUDDI API を使用して、作成者名 *my-publisher* を持つパブリッシャーを保存します。*root* ユーザーの承認トークンは使用されていることに注意してください。

4. この新しいパブリッシャーの承認トークンを取得します。

```
// Our publisher is now saved, so now we want to retrieve its authentication token
GetAuthToken getAuthTokenMyPub = new GetAuthToken();
getAuthTokenMyPub.setUserID("my-publisher");
getAuthTokenMyPub.setCred("");
AuthToken myPubAuthToken = security.getAuthToken(getAuthTokenMyPub);
System.out.println ("myPub AUTHTOKEN = " + myPubAuthToken.getAuthInfo());
```

認可呼び出しのいずれかに認証情報が設定されていないことに注意してください。これは、デフォルトのオーセンティケーターを使用することで何も指定する必要がないためです。

5. パブリッシュ:

```
// Creating the parent business entity that will contain our service.
BusinessEntity myBusEntity = new BusinessEntity();
Name myBusName = new Name();
myBusName.setValue("My Business");
```

```

myBusEntity.getName().add(myBusName);

// Adding the business entity to the "save" structure, using our publisher's
// authentication info and saving away.
SaveBusiness sb = new SaveBusiness();
sb.getBusinessEntity().add(myBusEntity);
sb.setAuthInfo(myPubAuthToken.getAuthInfo());
BusinessDetail bd = publish.saveBusiness(sb);
String myBusKey = bd.getBusinessEntity().get(0).getBusinessKey();
System.out.println("myBusiness key: " + myBusKey);

// Creating a service to save. Only adding the minimum data: the parent
// business key retrieved from saving the business above and a single name.
BusinessService myService = new BusinessService();
myService.setBusinessKey(myBusKey);
Name myServName = new Name();
myServName.setValue("My Service");
myService.getName().add(myServName);
// Add binding templates, etc...

// Adding the service to the "save" structure, using our publisher's
// authentication info and saving away.
SaveService ss = new SaveService();
ss.getBusinessService().add(myService);
ss.setAuthInfo(myPubAuthToken.getAuthInfo());
ServiceDetail sd = publish.saveService(ss);
String myServKey = sd.getBusinessService().get(0).getServiceKey();
System.out.println("myService key: " + myServKey);

```

注記

エンティティーキーの使用に関する重要な事項があります。バージョン 3 仕様では、パブリッシャーが独自のキーを作成できますが、実装者にデフォルトのメソッドがあることを指示します。

Red Hat は、**save** 呼び出しで各エンティティーの key フィールドを空白のままにして、デフォルトの実装アプローチを選択しました。jUDDI のデフォルトのキージェネレーターは、ノードのパーティションを取り、GUID を追加します。

デフォルトのインストールでは、以下のようになります。**uddi:juddi.apache.org:<GUID>**

(当然ながら、これをすべてカスタマイズすることができます。)

2 つ目の点は、BusinessService を保存すると、親ビジネスキー (ビジネスを保存する以前の呼び出しから取得) が明示的に設定されていることです。これは、サービスが独立した呼び出しに保存されている場合に必要手順です。これは、jUDDI が親エンティティーの場所を認識しないため、エラーが発生します。

結果

この例では、BusinessEntity を作成して保存し、BusinessService を作成および保存しました。最小限のデータを各エンティティーに追加している (実際は、BindingTemplates をサービスに追加していない)。

実際のシナリオでは、特にサービスに関して、各構造をより大きな情報で埋めたいと思うでしょう。ただし、これにより便利な開始点が証明されます。

バグの報告

第5章 サブスクリプション

5.1. サブスクリプション API

サブスクリプション API を使用すると、異なるサービスレジストリーでサービスを相互登録し、親レジストリーのレジストリー情報として通知を送信してそれらを最新の状態に保つことができます。これは、大規模または複雑な組織があり、部門ごとに異なるサービスレジストリーを実行し、それらの間で共有されるサービスがある場合に役立ちます。

バグの報告

5.2. サブスクリプションタイプ

サブスクリプションには 2 つのタイプがあります。それぞれの通知システムには、異なる通知システムがあります。

非同期

これにより、サブスクリプションを保存し、設定されたスケジュールで更新を受け取ることができます。通知を送信するノードにリスナーサービスをインストールする必要があります。

同期

これにより、サブスクリプションを保存し、**get_Subscription** を呼び出して同期応答を取得できます。

バグの報告

5.3. チュートリアル: サブスクリプション

前提条件

- この例では、**営業** および **マーケティング** 用にノードを設定します。これを実行するには、最初に Service Registry を 2 つの異なるサービスにデプロイする必要があります。

手順5.1 タスク

1. ノード 1 の設定: 営業

以下のコマンドを実行して **juddi_custom_install_data** を作成します。

```
cd juddiv3/WEB-INF/classes
```

```
mv RENAME4SALES_juddi_custom_install_data juddi_custom_install_data
```

2. テキストエディターで **SOA_ROOT/jboss-as/server/PROFILE/deploy/jbossesb-registry.sar/esb.juddi.xml** ファイルを開き、以下のプロパティー値を設定します (sales はサーバーの DNS 名に置き換えます)。

```
juddi.server.name=sales
juddi.server.port=8080
```

3. ファイルを保存して終了します。
4. サーバーを起動します。 **SOA_ROOT/jboss-as/bin/./run.sh**
5. Web ブラウザーを開き、このアドレス に移動します: <http://sales:8080/juddiv3> ノードについての情報を提供するメッセージが表示されるはずです。
6. **ノード 2 の設定: マーケティング**
以下のコマンドを実行して **juddi_custom_install_data** を作成します。

```
cd juddiv3/WEB-INF/classes
```

```
mv RENAME4MARKETING_juddi_custom_install_data juddi_custom_install_data
```

7. テキストエディターで **SOA_ROOT/jboss-as/server/PROFILE/deploy/jbossesb-registry.sar/esb.juddi.xml** ファイルを開き、以下のプロパティー値を設定します (これはサーバーの DNS 名です)。

```
juddi.server.name=marketing
juddi.server.port=8080
```

8. ファイルを保存して終了します。
9. サーバーを起動します。 **SOA_ROOT/jboss-as/bin/./run.sh**
10. Web ブラウザーを開き、このアドレス に移動します: <http://sales:8080/juddiv3> ここでも、ノードについての情報を提供するメッセージが表示されます。



注記

営業部門とマーケティング部門の両方が同じ会社にあるため、ルートパーティションは同じままでしたが、Node Id と Name は各ノードが異なる部門に属することを反映するように変更されている点に注意してください。

11. 次に、営業サーバーの **uddi-portlets.war/WEB-INF/classes/META-INF/uddi.xml** ファイルを **uddi-portlets.war/WEB-INF/classes/META-INF/uddi.xml.sales** に置き換えます。
12. テキストエディターで **uddi-portlets.war/WEB-INF/classes/META-INF/uddi.xml** ファイルを開き、以下のプロパティーを追加します。

```
<name>default</name>
<properties>
  <property name="serverName" value="sales"/>
  <property name="serverPort" value="8080"/>
  <property name="rmiPort" value="1099"/>
</properties>
```

13. ファイルを保存して終了します。
14. Web ブラウザーを起動し、このアドレスに移動します: <http://sales:8080/pluto>
15. ユーザー名とパスワードの両方で **sales** にログインし、画面に表示される内容を確認します。

16. マーケティングポータルにログインする前に、マーケティングの **uddi-portlet.war/WEB-INF/classes/META-INF/uddi.xml** ファイルを **udd-portlet.war/WEB-INF/classes/META-INF/uddi.xml.marketing** に置き換えます。
17. テキストエディターで **uddi-portlet.war/WEB-INF/classes/META-INF/uddi.xml** ファイルを開き、以下のプロパティーを追加します。

```
<name>default</name>
<properties>
  <property name="serverName" value="marketing"/>
  <property name="serverPort" value="8080"/>
  <property name="rmiPort" value="1099"/>
</properties>
```

18. ファイルを保存して終了します。
19. Web ブラウザーを起動し、このアドレス に移動します:<http://marketing:8080/pluto>
20. ユーザー名とパスワードの両方で **sales** にログインし、画面に表示される内容を確認します。



注記

subscriptionlistener は、Root Marketing ノードではなく、Marketing Node business によって所有されています。マーケティングノードビジネスはマーケティングパブリッシャーによって管理されます。

バグの報告

5.4. チュートリアル: HELLOSALES サービスのデプロイ

手順5.2 タスク

1. テキストエディターで **juddiv3-samples.war/WEB-INF/classes/META-INF/uddi.xml** ファイルを開き、以下のプロパティー値を sales に追加します。

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<uddi>
  <reloadDelay>5000</reloadDelay>
  <manager name="example-manager">
    <nodes>
      <node>
        <name>default</name>
        <description>Sales jUDDI node</description>
        <properties>
          <property name="serverName" value="sales"/>
          <property name="serverPort" value="8080"/>
          <property name="keyDomain" value="sales.apache.org"/>
          <property name="department" value="sales" />
        </properties>
        <proxyTransport>
          org.apache.juddi.v3.client.transport.InVMTransport
        </proxyTransport>
        <custodyTransferUrl>
```

```

        org.apache.juddi.api.impl.UDDICustodyTransferImpl
      </custodyTransferUrl>
      <inquiryUrl>org.apache.juddi.api.impl.UDDIInquiryImpl</inquiryUrl>
      <publishUrl>org.apache.juddi.api.impl.UDDIPublicationImpl</publishUrl>
      <securityUrl>org.apache.juddi.api.impl.UDDISecurityImpl</securityUrl>
      <subscriptionUrl>
        org.apache.juddi.api.impl.UDDISubscriptionImpl
      </subscriptionUrl>
      <subscriptionListenerUrl>
        org.apache.juddi.api.impl.UDDISubscriptionListenerImpl
      </subscriptionListenerUrl>
      <juddiApiUrl>org.apache.juddi.api.impl.JUDDIApiImpl</juddiApiUrl>
    </node>
  </nodes>
</manager>
</uddi>

```

2. ファイルを保存して終了します。
3. 次に、**juddiv3-samples.war** を営業レジストリーにデプロイします。

ant deploy

ant runtest



注記

HelloSales サービスは **juddiv3-samples.war** アーカイブファイルで提供され、アノテーションが付けられているため、自動的に登録されます。

4. Marketing UDDI ノード内から営業 UDDI ノードの HelloWord サービスにサブスクライブします。

バグの報告

5.5. ユーザーが独自のサブスクリプションを作成できるようにする

前提条件

- ユーザーがサブスクリプションを作成して保存するには、営業ノードとマーケティングノードの両方に有効なパブリッシュ済みのログインが必要です。
- マーケティングパブリッシャーがマーケティングノードでレジストリーオブジェクトを作成する場合、マーケティングパブリッシャーは **sales keygenerator tModel** を所有する必要があります。

マーケティングレジストリーのマーケティングパブリッシャーが以下の **tModels** を所有することを理解することが重要です。

```

<save_tModel xmlns="urn:uddi-org:api_v3">

  <tModel tModelKey="uddi:marketing.apache.org:keygenerator" xmlns="urn:uddi-
org:api_v3">

```

```

<name>marketing-apache-org:keyGenerator</name>
<description>Marketing domain key generator</description>
<overviewDoc>
  <overviewURL useType="text">
    http://uddi.org/pubs/uddi_v3.htm#keyGen
  </overviewURL>
</overviewDoc>
<categoryBag>
  <keyedReference tModelKey="uddi:uddi.org:categorization:types"
    keyName="uddi-org:types:keyGenerator"
    keyValue="keyGenerator" />
</categoryBag>
</tModel>

<tModel tModelKey="uddi:marketing.apache.org:subscription:keygenerator"
  xmlns="urn:uddi-org:api_v3">
  <name>marketing-apache-org:subscription:keyGenerator</name>
  <description>Marketing Subscriptions domain key generator</description>
  <overviewDoc>
    <overviewURL useType="text">
      http://uddi.org/pubs/uddi_v3.htm#keyGen
    </overviewURL>
  </overviewDoc>
  <categoryBag>
    <keyedReference tModelKey="uddi:uddi.org:categorization:types"
      keyName="uddi-org:types:keyGenerator"
      keyValue="keyGenerator" />
  </categoryBag>
</tModel>

<tModel tModelKey="uddi:sales.apache.org:keygenerator" xmlns="urn:uddi-org:api_v3">
  <name>sales-apache-org:keyGenerator</name>
  <description>Sales Root domain key generator</description>
  <overviewDoc>
    <overviewURL useType="text">
      http://uddi.org/pubs/uddi_v3.htm#keyGen
    </overviewURL>
  </overviewDoc>
  <categoryBag>
    <keyedReference tModelKey="uddi:uddi.org:categorization:types"
      keyName="uddi-org:types:keyGenerator"
      keyValue="keyGenerator" />
  </categoryBag>
</tModel>
</save_tModel>

```

手順5.3 タスク

1. マーケティングパブリッシャーを使用して営業レジストリーの更新をサブスクライブする場合は、このパブリッシャーに2つのクラスクを提供する必要があります。これを行うには、テキストエディターで **uddi-portlet.war/uddi.xml** ファイルを開き、以下の行を追加します。

```

<clerks registerOnStartup="false">
  <clerk name="MarketingCratchit" node="default"
    publisher="marketing" password="marketing"/>

```



```

<clerk name="SalesCratchit"      node="sales-ws"
       publisher="marketing"     password="marketing"/>
<!-- optional
<xregister>
  <servicebinding
    entityKey="uddi:marketing.apache.org:servicebindings-subscriptionlistener-ws"
    fromClerk="MarketingCratchit" toClerk="SalesCratchit"/>
  </xregister>
-->
</clerks>

```

上記のコードでは、このパブリッシャー (MarketingCratchit と SalesCratchit) の2つのクラスを作成しました。これにより、パブリッシャーは2つのシステムごとに所有しているサブスクリプションを確認できます。

2. ファイルを保存して終了します。
3. マーケティングポータルでマーケティングパブリッシャーとしてログインし、**UDDISubscription Portlet** を選択します。
4. 両方のノードが緑色に変わったら、new subscription アイコン (ツールバーに格納) をクリックします。これは同期サブスクリプションであるため、**Binding Key** および **Notification Interval** のみが残されます。
5. Save アイコンをクリックして、サブスクリプションを保存します。
6. サブスクリプションキーがマーケティングパブリッシャーの **keyGenerator** の規則を使用していることを確認してください。**sales-ws** UDDI ノードにオレンジ色のサブスクリプションアイコンが表示されるはずです。
7. 同期サブスクリプションを呼び出すには、Green Arrows アイコンをクリックします。これにより、カバレッジ期間を設定する機会が与えられます。
8. Green Arrows アイコンを再度クリックして、同期サブスクリプションリクエストを呼び出します。

finder リクエストの例では、sales ノードが **HelloWorld** サービスへの更新を検索します。その後、raw の XML 応答が **UDDISubscriptionNotification** ポートレットに送信されます。

9. レスポンスはマーケティングノードによって処理されます。次に、このノードは **HelloWorld** サブスクリプション情報と **sales business** をインポートします。正常に同期すると、3つの企業がマーケティングノードのブラウザーポートレットに表示されます。

バグの報告

第6章 M-BEAN のサポート

6.1. 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](#)

6.2. 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7章 JUDDI クライアントの使用

7.1. JUDDI クライアント

jUDDI クライアントはサービスレジストリーに接続します。クライアントは **SOA_ROOT/jboss-as/server/PROFILE/deployers/esb.deployer/lib/juddi-client-VERSION.jar** にあります。

[バグの報告](#)

7.2. JUDDI クライアント依存関係

jUDDI クライアントは、以下のファイルに依存します。

- **uddi-ws-3.0.0.jar**
- **commons-configuration-1.5.jar**
- **commons-collection-3.2.1.jar**
- **log4j-1.2.13.jar**

これに加えて、設定の選択に応じて以下が必要になる場合があります。

- JDK6 のライブラリー。
- JAXWS クライアントライブラリー (CXF などの JAXWS トランスポートを使用している場合)
- RMI および JNDI クライアントライブラリー (RMI トランスポートを使用している場合)。

[バグの報告](#)

7.3. JUDDI クライアントおよび JBOSS ENTERPRISE SOA PLATFORM



重要

jUDDI クライアントは UDDI v3 API を使用するので、UDDI v3 準拠のレジストリーに接続できるはずですが、ただし、Red Hat は jUDDI v3 レジストリーでのみサポートします。

[バグの報告](#)

7.4. トランスポート設定

トランスポート設定は、jUDDI クライアントがサーバーと通信するために使用する接続設定です。

[バグの報告](#)

7.5. UDDI.XML

META-INF/uddi.xml ファイルには、jUDDI クライアントの設定が含まれます。このファイルは、UDDI クライアントコードと対話するデプロイメントアーカイブに追加できます。

バグの報告

7.6. カスタム JUDDI クライアント設定のデプロイ

手順7.1 タスク

1. テキストエディターで META-INF/uddi.xml を開きます。
2. 設定を構成します。
3. 保存して終了します。
4. デプロイ時に uddi.xml ファイルをアーカイブに追加します。
5. このコードを呼び出します。

```
UDDIClerkManager clerkManager = new UDDIClerkManager("META/myuddi.xml");
clerkManager.start();
```

6. または、アプリケーションが WAR アーカイブとしてデプロイする場合は、クライアント設定を **yourwar/META-INF/myuddi.xml** に追加し、web.xml ファイルで以下のコンテキストパラメーター **uddi.client.manager.name** および **uddi.client.xml** を指定します。

この例では、両方のコンテキストパラメーターが設定され、デプロイ時に UDDIClerkServlet が設定を読み取ります。

```
<!-- required -->
<context-param>
  <param-name>uddi.client.manager.name</param-name>
  <param-value>example-manager</param-value>
</context-param>

<!-- optional override -->
<context-param>
  <param-name>uddi.client.xml</param-name>
  <param-value>META-INF/myuddi.xml</param-value>
</context-param>

<servlet>
  <servlet-name>UDDIClerkServlet</servlet-name>
  <display-name>Clerk Servlet</display-name>
  <servlet-class>org.apache.juddi.v3.client.config.UDDIClerkServlet</servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>
```

バグの報告

7.7. JUDDI クライアント設定ファイルの例

以下は、簡単な JUDDI クライアント設定です。

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<uddi>
  <reloadDelay>5000</reloadDelay>
  <manager name="example-manager">
    <nodes>
      <node isHomeJUDDI="true">
        <name>default</name>
        <description>jUDDI node</description>
        <properties>
          <property name="serverName" value="www.myuddiserver.com"/>
          <property name="serverPort" value="8080"/>
          <property name="keyDomain" value="mydepartment.mydomain.org"/>
          <property name="department" value="mydepartment" />
        </properties>
        <!-- InVM -->
        <proxyTransport>org.apache.juddi.v3.client.transport.InVMTransport</proxyTransport>
        <custodyTransferUrl>org.apache.juddi.api.impl.UDDICustodyTransferImpl</custodyTransferUrl>
        <inquiryUrl>org.apache.juddi.api.impl.UDDIInquiryImpl</inquiryUrl>
        <publishUrl>org.apache.juddi.api.impl.UDDIPublicationImpl</publishUrl>
        <securityUrl>org.apache.juddi.api.impl.UDDISecurityImpl</securityUrl>
        <subscriptionUrl>org.apache.juddi.api.impl.UDDISubscriptionImpl</subscriptionUrl>

        <subscriptionListenerUrl>org.apache.juddi.api.impl.UDDISubscriptionListenerImpl</subscriptionListen
erUrl>
        <juddiApiUrl>org.apache.juddi.api.impl.JUDDIApiImpl</juddiApiUrl>
        <!-- JAX-WS Transport
        <proxyTransport>org.apache.juddi.v3.client.transport.JAXWSTransport</proxyTransport>
        <custodyTransferUrl>http://${serverName}:${serverPort}/juddiv3/services/custody-
transfer</custodyTransferUrl>
        <inquiryUrl>http://${serverName}:${serverPort}/juddiv3/services/inquiry</inquiryUrl>
        <publishUrl>http://${serverName}:${serverPort}/juddiv3/services/publish</publishUrl>
        <securityUrl>http://${serverName}:${serverPort}/juddiv3/services/security</securityUrl>

        <subscriptionUrl>http://${serverName}:${serverPort}/juddiv3/services/subscription</subscriptionUrl>
        <subscriptionListenerUrl>http://${serverName}:${serverPort}/juddiv3/services/subscription-
listener</subscriptionListenerUrl>
        <juddiApiUrl>http://${serverName}:${serverPort}/juddiv3/services/juddi-api?wsdl</juddiApiUrl>
        -->
        <!-- RMI Transport Settings
        <proxyTransport>org.apache.juddi.v3.client.transport.RMITransport</proxyTransport>
        <custodyTransferUrl>/juddiv3/UDDICustodyTransferService</custodyTransferUrl>
        <inquiryUrl>/juddiv3/UDDIInquiryService</inquiryUrl>
        <publishUrl>/juddiv3/UDDIPublicationService</publishUrl>
        <securityUrl>/juddiv3/UDDISecurityService</securityUrl>
        <subscriptionUrl>/juddiv3/UDDISubscriptionService</subscriptionUrl>
        <subscriptionListenerUrl>/juddiv3/UDDISubscriptionListenerService</subscriptionListenerUrl>
        <juddiApiUrl>/juddiv3/JUDDIApiService</juddiApiUrl>
        <javaNamingFactoryInitial>org.jnp.interfaces.NamingContextFactory</javaNamingFactoryInitial>
        <javaNamingFactoryUrlPkgs>org.jboss.naming</javaNamingFactoryUrlPkgs>
        <javaNamingProviderUrl>jnp://${serverName}:1099</javaNamingProviderUrl>
        -->
      </node>
    </nodes>
    <clerks registerOnStartup="true">
```

```

<clerk name="BobCratchit" node="default" publisher="bob" password="bob">
  <class>org.apache.juddi.samples.HelloWorldImpl</class>
</clerk>
</clerks>
</manager>
</uddi>

```

manager 要素が必要です。example-manager name 属性は、デプロイメント環境に固有のものである必要があります。nodes 要素には、1つ以上のサブノード要素を含めることができます。通常、サブスクリプションを使用してエンティティの更新を別の UDDI レジストリーに転送する場合を除き、1つのノードのみが必要になります。ローカルレジストリーの場合は、**isHomeJUDDI="true"** を設定しますが、リモートレジストリーの場合は **isHomeJUDDI="false"** を設定します。

表7.1 要素

要素名	Description	必須？
name	ノードの名前	はい
description	ノードの説明	いいえ
properties	クラスに渡されるプロパティのコンテナ	いいえ
proxyTransport	UDDI サーバーに接続するためにクライアントが使用するトランスポートプロトコル	はい
custodyTransferUrl	カストディ転送の接続設定	いいえ
inquiryUrl	問い合わせの接続ロケーションの設定	はい
publishUrl	公開用の接続ロケーションの設定	はい
securityUrl	セキュリティトークンを取得するための接続ロケーションの設定	はい
subscriptionUrl	サブスクリプションリクエストを登録するための接続ロケーションの設定	いいえ
subscriptionListenerUrl	サブスクリプション通知を受信する接続ロケーションの設定	いいえ
juddiApiUrl	パブリッシャー管理など、JUDDI 固有の API の接続ロケーション設定	いいえ

最後に、manager 要素には、1つ以上のクラスを定義できるクラス要素を含めることができます。

表7.2 クラーク

属性名	Description	必須？
name	クラークの名前	はい
node	同じマネージャーで指定されたノードの1つへの名前参照	はい
publisher	既存のパブリッシャーの名前	はい
password	パブリッシャーのパスワード認証情報	はい

バグの報告

7.8. JAVA API FOR XML WEB SERVICES (JAX-WS)

Java API for XML Web Services (JAX-WS) は、Web サービスを作成できる Java API です。JAX-WS ハンドラーメカニズムは、メッセージ(または障害)が送信または受信されるたびに、ユーザー指定のクラスを呼び出すために Web サービスによって使用されます。したがって、ハンドラーはメッセージパイプラインにインストールされ、必要に応じてメッセージヘッダーまたは本文を操作するために使用されます。

バグの報告

7.9. JAX-WS トランスポートの設定

前提条件

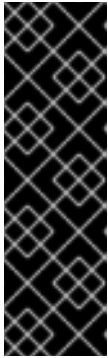
- **uddi.xml** ファイルの設定に基づいて、クライアントは JAX-WS を使用して(リモート)レジストリーサーバーと通信します。これは、クライアントが JAX-WS 準拠の Web サービススタック(CXF、Axis2、JBossWS など)にアクセスできる必要があることを意味します。

手順7.2 タスク

- JAX-WS URL は、UDDI クライアントが WSDL ドキュメントを見つけることができるアドレスを参照していることを確認します。

```
<!-- JAX-WS Transport -->
<proxyTransport>org.apache.juddi.v3.client.transport.JAXWSTransport</proxyTransport>
<custodyTransferUrl>http://${serverName}:${serverPort}/juddiv3/services/custody-
transfer</custodyTransferUrl>
<inquiryUrl>http://${serverName}:${serverPort}/juddiv3/services/inquiry</inquiryUrl>
<publishUrl>http://${serverName}:${serverPort}/juddiv3/services/publish</publishUrl>
<securityUrl>http://${serverName}:${serverPort}/juddiv3/services/security</securityUrl>
<subscriptionUrl>http://${serverName}:${serverPort}/juddiv3/services/subscription</subscripti
onUrl>
<subscriptionListenerUrl>http://${serverName}:${serverPort}/juddiv3/services/subscription-
```

```
listener</subscriptionListenerUrl>
<juddiApiUrl>http://${serverName}:${serverPort}/juddi3/services/juddi-api?
wsdl</juddiApiUrl>
```



重要

長所: これは UDDI 通信を実行する標準的な方法であり、すべての UDDIv3 サーバー実装で動作するはずです。

短所: サーバーが同じアプリケーションサーバーにデプロイメントされている場合、デプロイメント/デプロイメント解除の自動登録を使用すると、デプロイメント解除中に WS スタックが使用できなくなる可能性があるため、問題が発生する可能性があります。回避策は、別のサーバーで UDDI サーバーをホストすることです。

バグの報告

7.10. 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リスクにつながる可能性があります。

バグの報告

7.11. JUDDI とリモートメソッド呼び出し

jUDDI をアプリケーションサーバーにデプロイすると、UDDI サービスをリモートメソッド呼び出しサービスとして登録できます。

バグの報告

7.12. JUDDI のリモートメソッド呼び出しの有効化

手順7.3 タスク

1. jUDDI 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/deploy/jbossesb-registry.sar/esb.juddi.xml**。
2. 設定を編集し、プロパティ `<entry key="juddi.jndi.registration">true</entry>` を設定します。true に設定すると、RMI メソッドが jndi に登録され、検索して呼び出すことができます。デフォルトは true です。
3. ファイルを保存して終了します。
4. テキストエディターで UDDI 設定ファイルを開きます: **vi META-INF/uddi.xml**。
5. ファイルの JAX-WS セクションをコメントアウトし、代わりに RMI セクションのコメントを外します。

6. オプションの手順として、`java.naming.*` プロパティを設定することもできます。この例では、**JBoss Application Server** にデプロイされた jUDDI v3 に接続するための設定を指定しました。その代わりに、**jndi.properties** ファイルやシステムパラメータとして `java.naming.*` プロパティを設定することができます。

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

<juddiApiUrl>/juddiv3/JUDDIApiService</juddiApiUrl>
<javaNamingFactoryInitial>org.jnp.interfaces.NamingContextFactory</javaNamingFactoryInitial>
<javaNamingFactoryUrlPkgs>org.jboss.naming</javaNamingFactoryUrlPkgs>
<javaNamingProviderUrl>jnp://${serverName}:1099</javaNamingProviderUrl>
```



重要

長所: Web サービススタックを必要としないため、軽量で高速です。

短所: jUDDI v3 サーバー実装でのみ機能します。

7. ファイルを保存して終了します。

結果

これをデプロイすると、RMI ベースの UDDI サービスがグローバル JNDI 名前空間にバインドされます。

- juddiv3 (class: **org.jnp.interfaces.NamingContext**)
- UDDIPublicationService (class: **org.apache.juddi.rmi.UDDIPublicationService**)
- UDDICustodyTransferService (class: **org.apache.juddi.rmi.UDDICustodyTransferService**)
- UDDISubscriptionListenerService (class: **org.apache.juddi.rmi.UDDISubscriptionListenerService**)
- UDDISecurityService (class: **org.apache.juddi.rmi.UDDISecurityService**)
- UDDISubscriptionService (class: **org.apache.juddi.rmi.UDDISubscriptionService**)
- UDDIInquiryService (class: **org.apache.juddi.rmi.UDDIInquiryService**)

バグの報告

7.13. INVM トランスポート

InVM (仮想マシン内) トランスポートは、同じ JVM で実行されているサービス間の通信を提供します。

[バグの報告](#)

7.14. INVM と JUDDI

InVM トランスポートを使用するオプションがあります。これにより、クライアントと同じ仮想マシンで jUDDI サーバーを実行できます。**juddi.war** アーカイブにデプロイする場合、サーバーは **org.apache.juddi.RegistryServlet** クラスによって開始されますが、コンテナの外部で実行している場合は、**org.apache.juddi.Registry** サービスを自分で開始および停止する必要があります。

[バグの報告](#)

7.15. JUDDI の INVM トランスポートの設定

手順7.4 タスク

1. テキストエディターで UDDI 設定ファイルを開きます: **vi META-INF/uddi.xml**。
2. ファイルの JAX-WS および RMI トランスポートセクションをコメントアウトし、代わりに InVM トランスポートセクションのコメントを外します。

```
<!-- InVM -->
<proxyTransport>org.apache.juddi.v3.client.transport.InVMTransport</proxyTransport>
<custodyTransferUrl>org.apache.juddi.api.impl.UDDICustodyTransferImpl</custodyTransferUrl>
<inquiryUrl>org.apache.juddi.api.impl.UDDIInquiryImpl</inquiryUrl>
<publishUrl>org.apache.juddi.api.impl.UDDIPublicationImpl</publishUrl>
<securityUrl>org.apache.juddi.api.impl.UDDISecurityImpl</securityUrl>
<subscriptionUrl>org.apache.juddi.api.impl.UDDISubscriptionImpl</subscriptionUrl>
<subscriptionListenerUrl>org.apache.juddi.api.impl.UDDISubscriptionListenerImpl</subscriptionListenerUrl>
<juddiApiUrl>org.apache.juddi.api.impl.JUDDIApiImpl</juddiApiUrl>
```

重要

長所: 軽量であり、通信に最高のパフォーマンスを提供します。デプロイメントおよびデプロイメント解除時にサービスの自動登録を使用する場合、デプロイメント順序の問題はありません。

短所: jUDDI v3 サーバー実装でのみ機能します。通常、1つの共通データベースを共有するアプリケーションサーバーごとに jUDDI サーバーを使用します。

3. ファイルを保存して終了します。

[バグの報告](#)

7.16. INVM トランスポートを使用したサービスレジストリーの開始

- 他の呼び出しを行う前に、次のメソッドを呼び出します: **Registry.start()**

[バグの報告](#)

7.17. INVM トランスポートを使用したサービスレジストリーの停止

手順7.5 タスク

- 他の呼び出しを行う前に、次のメソッドを呼び出します: **Registry.stop()**

結果

レジストリーは、保持しているリソースをすべて解放します。

[バグの報告](#)

7.18. UDDI アノテーション

UDDI アノテーションは、デプロイ時にサービスを自動的に登録します。サービス登録を自動的に処理することで、各エンドポイント参照を手動で追加および削除する場合よりも、サービスレジストリー内のデータが古くなる可能性が低くなります。

2つのアノテーションがあります。つまり、UDDIService と UDDIServiceBinding です。エンドポイントを登録するには、両方を使用する必要があります。

サービスをアンデプロイすると、エンドポイントは UDDI レジストリーから削除されますが、サービス情報は残ります。（現時点ではエンドポイントがなくてもサービスが存在することを反映しているため、サービスレベル情報をレジストリーに残しておくことは理にかなっています。これは、後で確認するよう指示されます。



注記

サービス情報を削除する場合は、手動で行う必要があります。

[バグの報告](#)

7.19. UDDISERVICE アノテーションの使用

手順7.6 タスク

- UDDIService アノテーションを使用して、レジストリー内の既存のビジネスでサービスを登録します。アノテーションは、Java クラスのクラスレベルに追加する必要があります。

[バグの報告](#)

7.20. UDDISERVICE 属性

表7.3

属性	Description	必須？
serviceName	これはサービスの名前です。デフォルトでは、クラスは <code>WebService</code> アノテーションで指定された1つの名前を使用します。	いいえ
description	これは、人間が判読できるサービスの説明です。	はい
serviceKey	これは、サービスの UDDI v3 キーです。	はい
businessKey	これは、サービスを所有する必要があるビジネスの UDDI v3 キーです。(ビジネスは登録時にレジストリーに存在している必要があります。)	はい
lang	これは、名前と説明に使用される言語ロケールです。(省略した場合のデフォルトは en です。)	いいえ
categoryBag	これは、 <i>CategoryBag</i> の定義です。	いいえ

バグの報告

7.21. UDDISERVICEBINDING アノテーションの使用

手順7.7 タスク

- `UDDIServiceBinding` アノテーションを使用して、サービスレジストリーにエンドポイント参照を登録します。アノテーションは、Java クラスのクラスレベルに追加する必要があります。



注記

このアノテーションは単独で使用できません。 `UDDIService` アノテーション内で利用する必要があります。

バグの報告

7.22. UDDISERVICEBINDING 属性

表7.4 `UDDIServiceBinding` 属性

属性	Description	必須？
bindingKey	これは ServiceBinding の UDDI v3 キーです。	はい
description	これは、人間が判読できるサービスの説明です。	はい
accessPointType	これは UDDI v3 の AccessPointType です。(省略した場合、デフォルトでは wsdlDeployment です。)	いいえ
accessPoint	これは、エンドポイントの参照です。	はい
lang	これは、名前と説明に使用される言語ロケールです (省略した場合はデフォルトで en になります)。	いいえ
tModelKeys	これは、tModelKeys キー参照のコンマ区切りリストです。	いいえ
categoryBag	これは、CategoryBag の定義です。	いいえ

バグの報告

7.23. UDDI アノテーションの使用例

はじめに

サービスを定義する任意のクラスでアノテーションを使用できます。ここでは、それらが Web サービス (JAX-WS Web サービスアノテーションを持つ POJO) に追加されます。

```
package org.apache.juddi.samples;

import javax.jws.WebService;
import org.apache.juddi.v3.annotations.UDDIService;
import org.apache.juddi.v3.annotations.UDDIServiceBinding;

@UDDIService(
    businessKey="uddi:myBusinessKey",
    serviceKey="uddi:myServiceKey",
    description = "Hello World test service")
@UDDIServiceBinding(
    bindingKey="uddi:myServiceBindingKey",
    description="WSDL endpoint for the helloWorld Service. This service is used for "
    + "testing the jUDDI annotation functionality",
```

```

        accessPointType="wsdlDeployment",
        accessPoint="http://localhost:8080/juddiv3-samples/services/helloworld?wsdl")
@WebService(
    endpointInterface = "org.apache.juddi.samples.HelloWorld",
    serviceName = "HelloWorld")

public class HelloWorldImpl implements HelloWorld {
    public String sayHi(String text) {
        System.out.println("sayHi called");
        return "Hello " + text;
    }
}

```

この Web サービスをデプロイすると、**juddi-client** コードはこのクラスをスキャンして UDDI アノテーションを探し、登録プロセスを処理します。

clerk セクションでは、**org.apache.juddi.samples.HelloWorldImpl** サービスクラスを参照する必要があります。

```

<clerk name="BobCratchit" node="default" publisher="sales" password="sales">
    <class>org.apache.juddi.samples.HelloWorldImpl</class>
</clerk>

```

このコードは、Bob が **default** という名前のノードの接続設定を使用し、**sales** パブリッシャーを使用するように指定します (パスワードは **sales** です)。



注記

これは、data-sources の定義方法と似ています。

[バグの報告](#)

7.24. CATEGORYBAG 属性

CategoryBag 属性を使用して tModels を参照します。

[バグの報告](#)

7.25. CATEGORYBAG アノテーションの使用例

以下は、CategoryBag の例です。

```

<categoryBag>
    <keyedReference tModelKey="uddi:uddi.org:categorization:types"
        keyName="uddi-org:types:wsdl" keyValue="wsdlDeployment" />
    <keyedReference tModelKey="uddi:uddi.org:categorization:types"
        keyName="uddi-org:types:wsdl2" keyValue="wsdlDeployment2" />
</categoryBag>

```

以下のように追加できます。

```
categoryBag="keyedReference=keyName=uddi-org:types:wsdl;keyValue=wsdlDeployment;" +
    "tModelKey=uddi:uddi.org:categorization:types," +
    "keyedReference=keyName=uddi-org:types:wsdl2;keyValue=wsdlDeployment2;" +
    "tModelKey=uddi:uddi.org:categorization:types2",
```

バグの報告

7.26. アノテーションを使用した WEB サービスの開発

手順7.8 タスク

1. サービスを定義する任意のクラスにアノテーションを追加します。以下は、JAX-WS Webservice アノテーションを使用して Web サービス POJO に追加する例です。

```
package org.apache.juddi.samples;

import javax.ws.WebService;
import org.apache.juddi.v3.annotations.UDDIService;
import org.apache.juddi.v3.annotations.UDDIServiceBinding;

@UDDIService(businessKey="uddi:myBusinessKey", serviceKey="uddi:myServiceKey",
description = "Hello World test service")
@UDDIServiceBinding(bindingKey="uddi:myServiceBindingKey", description="WSDL
endpoint for the helloWorld Service. This service is used for "
    + "testing the jUDDI annotation functionality",
    accessPointType="wsdlDeployment", accessPoint="http://localhost:8080/juddiv3-
samples/services/helloworld?wsdl")
@WebService(endpointInterface = "org.apache.juddi.samples.HelloWorld", serviceName =
"HelloWorld")

public class HelloWorldImpl implements HelloWorld {
    public String sayHi(String text) {
        System.out.println("sayHi called");
        return "Hello " + text;
    }
}
```

この WebService のデプロイ時に、juddi-client コードはこのクラスをスキャンして UDDI アノテーションを探し、登録プロセスを処理します。

2. テキストエディターで uddi.xml ファイルを開きます。vi uddi.xml
3. クラークセクションまでスクロールして、**org.apache.juddi.samples.HelloWorldImpl** サービスクラスへの参照を追加します。

```
<clerk registerOnStartup="true">
  <clerk name="BobCratchit" node="default" publisher="bob" password="bob">
    <class>org.apache.juddi.samples.HelloWorldImpl</class>
  </clerk>
</clerk>
```

この例では、Bob は default という名前のノードの接続設定を使用しています。さらに、パスワードも bob である bob という名前のパブリッシャーを使用します。

4. ファイルを保存して終了します。

バグの報告

7.27. キーテンプレート

キーテンプレートは、ビジネス、サービス、またはバインドキーのアノテーション属性に設定できるテンプレートです。WSDL と BPEL の両方の登録コードは、キー形式 `o` を使用して UDDI v3 キーを作成します。キーの形式は、**uddi.xml** ファイルのプロパティセクションで定義されます。

キーの `serviceName` と `portName` の両方が `RegistrationInfo` から取得されます。`nodeName` は、環境から自動的に取得することも、**uddi.xml** ファイルで手動で設定することもできます。このテンプレートでプロパティに設定した値は、キーの登録時に使用されます。

バグの報告

7.28. キーテンプレートのプロパティ

表7.5 キーテンプレートのプロパティ

プロパティ	Description	必須？	デフォルト値
<code>lang</code>	登録で使用する言語設定	いいえ	<code>en</code>
<code>businessName</code>	登録が使用するビジネス名	はい	-
<code>keyDomain</code>	キードメインのキー部分 (キーフォーマットで使用)	はい	-
<code>businessKeyFormat</code>	ビジネスキーの作成に使用するキー形式	いいえ	<code>uddi:\${keyDomain}:business_\${businessName}</code>
<code>serviceKeyFormat</code>	BusinessService キーの作成に使用するキー形式	いいえ	<code>uddi:\${keyDomain}:service_\${serviceName}</code>
<code>bindingKeyFormat</code>	TemplateBinding キーの作成に使用されるキー形式	いいえ	<code>uddi:\${keyDomain}:binding_\${nodeName}_\${serviceName}_\${portName}</code>
<code>serviceDescription</code>	デフォルトの BusinessService の説明	いいえ	<code><wsdl:service></code> 要素内に <code><wsdl:document></code> 要素が定義されていない場合のデフォルトのサービスの説明

プロパティ	Description	必須？	デフォルト値
bindingDescription	デフォルトの BindingTemplate の説明	いいえ	<wsdl:binding> 要素内に <wsdl:document> 要素が定義されていない場合のデフォルトのバインディングの説明

バグの報告

7.29. アプリケーションでの JUDDI クライアントコードの使用

手順7.9 タスク

1. jUDDI クライアント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/deploy/jbossesb-registry.sar/esb.juddi.xml**。このファイルを独自の **uddi.xml** ファイルのテンプレートとして使用します。
2. カスタム uddi.xml がインスタンス化されたときに、クラークマネージャーがそのカスタム **uddi.xml** を指していることを確認してください。

```

UDDIClerkManager clerkManager = new UDDIClerkManager("META/myuddi.xml");
clerkManager.start();

UDDIClerk clerk = clerkManager.getClientConfig().getUDDIClerks().get(clerkName);

```



注記

UDDIClerk を使用すると、UDDI サーバーに対して認証済みのリクエストを行うことができます。

バグの報告

7.30. WSDL および UDDI レジストリー

jUDDI クライアントは、次の OASIS テクニカルノートで説明されているように、WSDL2UDDI マッピングの UDDI v3 バージョンを実装します。 <http://www.oasis-open.org/committees/uddi-spec/doc/tn/uddi-spec-tc-tn-wsdl-v2.htm> 登録プロセスは、各 WebService EndPoint の BindingTemplate を登録します。この BindingTemplate の BusinessService がまだ存在しない場合は、各 portType の WSDLPortType TModel および各バインディングの WSDLBinding TModel と共に BusinessService も登録します。

バグの報告

7.31. WSDL におけるサービスとバインディングの説明の仕様

UDDI 仕様では、BusinessService と TemplateBinding の両方に人間が判読できる説明を設定できます。これらの説明フィールドは、人間がレジストリーを参照する場合に重要です。デフォルトのグロー

バル説明は、**uddi.xml** ファイルにあります。ただし、各サービスやバインディングに固有の説明がある方がはるかに理にかなっているので、登録コードはこれらの説明を WSDL の `<wsdl:document>` タグから取得しようとしています。このタグは `<wsdl:service>` と `<wsdl:binding>` 要素の内部に子要素としてネストさせることが可能です。

バグの報告

7.32. JUDDI での WSDL プロセスの登録

前提条件

- `org.apache.juddi.v3.client.mapping`

手順7.10 タスク

1. **`org.apache.juddi.v3.client.mapping`** パッケージのコードを使用します。
2. 以下の呼び出しを行い、Web サービスのエンドポイントを非同期に登録します。

```
//Add the properties from the uddi.xml
properties.putAll(clerk.getUDDINode().getProperties());
RegistrationInfo registrationInfo = new RegistrationInfo();
registrationInfo.setServiceQName(serviceQName);
registrationInfo.setVersion(version);
registrationInfo.setPortName(portName);
registrationInfo.setServiceUrl(serviceUrl);
registrationInfo.setWsdUrl(wsdUrl);
registrationInfo.setWsdDefinition(wsdDefinition);
registrationInfo.setRegistrationType(RegistrationType.WSDL);
registration = new AsyncRegistration(clerk, urlLocalizer, properties, registrationInfo);
Thread thread = new Thread(registration);
thread.start();
```

注記

これは、WSDLDefinition に加えて、WSDL ファイルに URL を渡すことができることを前提としています。ほとんどの場合、デプロイメントに登録する WSDL ファイルをパッケージ化する必要があります。

WSDLDefinition を取得するには、以下のコードを使用します。

```
ReadWSDL readWSDL = new ReadWSDL();
Definition definition = readWSDL.readWSDL("wsdl/HelloWorld.wsdl");
```

クラスパスにある WSDL にパス情報を渡す必要があります。

バグの報告

7.33. サービスレジストリーからの WSDL バインディングの削除

手順7.12 ツェン

- サービスレジストリーから WSDL バインディングを削除するには、以下のコードを使用します。

```
WSDL2UDDI wsdl2UDDI = new WSDL2UDDI(clerk, urlLocalizer, properties);
String serviceKey = wsdl2UDDI.unregister(serviceName, portName, serviceURL);
```



注記

これが BusinessService の最後の BindingTemplate である場合、WSDLPortType および WSDLBinding TModel とともに BusinessService 自体も削除されます。ライフサイクルは、エンドポイントのデプロイ時に registration に設定され、エンドポイントの削除時に unregistration に設定されます。

バグの報告

7.34. JUDDI での BPEL プロセスの登録

前提条件

- org.apache.juddi.v3.client.mapping

手順7.12 タスク

1. **org.apache.juddi.v3.client.mapping** パッケージのコードを使用します。
2. 以下の呼び出しを行い、Web サービスのエンドポイントを非同期に登録します。

```
//Add the properties from the uddi.xml
properties.putAll(clerk.getUDDINode().getProperties());
RegistrationInfo registrationInfo = new RegistrationInfo();
registrationInfo.setServiceQName(serviceQName);
registrationInfo.setVersion(version);
registrationInfo.setPortName(portName);
registrationInfo.setServiceUrl(serviceUrl);
registrationInfo.setWsdUrl(wsdlURL);
registrationInfo.setWsdDefinition(wsdlDefinition);
registrationInfo.setRegistrationType(RegistrationType.BPEL);
registration = new AsyncRegistration(clerk, urlLocalizer, properties, registrationInfo);
Thread thread = new Thread(registration);
thread.start();
```

RegistrationInfo.RegistrationType を RegistrationType.BPEL に設定します。

バグの報告

7.35. URLLOCALIZER インターフェイスおよび JBOSS ENTERPRISE SOA PLATFORM

エンドポイント URL 設定は、WSDL <wsdl:port> の <soap:addressbinding> から取得されます。残念ながら

ら、この URL は静的であるため、動的にできると便利です。JBoss Enterprise SOA Platform に同梱される URLLocalizer のバージョンでは、ローカル WebService スタックから取得した設定が URL のプロトコルおよびホスト部分を上書きし、この問題を軽減します。

バグの報告

7.36. 動的 UDDI サービスルックアップ

動的 UDDI サービスルックアップとは、UDDI レジストリーに定期的に問い合わせ、新しいバインディング情報を取得するプロセスを指します。これにより、クライアント側に格納されている Web サービスのエンドポイントの実際のバインディング情報を単純に調べる場合よりも、システムがより動的になります。さらに、クライアントは、サービスデプロイメントトポロジで発生する変更を簡単に追跡できます。

バグの報告

7.37. SERVICE LOCATOR を使用したサービスバインディングの検索

前提条件

- サービスおよびポートの名前がすでに分かっている必要があります。

手順7.13 タスク

- サービスロケータを使用してサービスバインディングを検索するには、次のコードを使用します。

```
ServiceLocator serviceLocator = new ServiceLocator(clerk, urlLocalizer, properties);
String endPointURL = serviceLocator.lookupEndpoint(serviceQName, String portName);
```



注記

各サービスを呼び出す前にルックアップを行うことの欠点は、パフォーマンスに悪影響を与えることです。

バグの報告

7.38. チュートリアル: JUDDI クライアントの使用

前提条件

- パブリッシャー (この場合は **root**) がサービスレジストリーにすでに存在することを確認します。

手順7.14 タスク

- uddi-client** モジュールにあるサンプルコードを取得します。

2. レジストリーを呼び出して、認証トークンを取得します。(次のコードは、このモジュールの単体テストから取得したものです):

```
public void testAuthToken() {
    try {
        String clazz = ClientConfig.getConfiguration().getString(
            Property.UDDI_PROXY_TRANSPORT, Property.DEFAULT_UDDI_PROXY_TRANSPORT);
        Class<?> transportClass = Loader.loadClass(clazz);
        if (transportClass != null) {
            Transport transport = (Transport) transportClass.newInstance();
            UDDISecurityPortType securityService = transport.getSecurityService();
            GetAuthToken getAuthToken = new GetAuthToken();
            getAuthToken.setUserID("root");
            getAuthToken.setCred("");
            AuthToken authToken = securityService.getAuthToken(getAuthToken);
            System.out.println(authToken.getAuthInfo());
            Assert.assertNotNull(authToken);
        } else {
            Assert.fail();
        }
    } catch (Exception e) {
        e.printStackTrace();
        Assert.fail();
    }
}
```



注記

正常なレスポンスを取得するために、ルートパブリッシャーの正しい認証情報を提供していることを確認してください。

[バグの報告](#)

付録A 更新履歴

改訂 5.3.1-23.400 Rebuild with publican 4.0.0	2013-10-31	Rüdiger Landmann
改訂 5.3.1-23 Built from Content Specification: 6847, Revision: 365903 by dlesage	Thu Jan 10 2013	David Le Sage