



JBoss Enterprise SOA Platform 5

JUDDI レジストリガイド

JBoss システム管理者および開発者向け
エディション 5.2.0

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

JBoss システム管理者および開発者向け
エディション 5.2.0

編集者

David Le Sage
Red Hat Engineering Content Services

With contributions from

著作者: Tom Cunningham、Kurt Stam、Jeff Faath、および jUDDI オープンソースコミュニティ
(一部、jUDDI User Guide をベースとする)

法律上の通知

Copyright © 2011 Red Hat Inc..

This document is licensed by Red Hat under the [Creative Commons Attribution-ShareAlike 3.0 Unported License](#). If you distribute this document, or a modified version of it, you must provide attribution to Red Hat, Inc. and provide a link to the original. If the document is modified, all Red Hat trademarks must be removed.

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, 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 Software Collections 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

JBoss Enterprise SOA Platform の jUDDI Registry コンポーネントの活用方法について学習するには、本書を参照してください。

目次

第1章 レジストリとは？	3
第2章 レジストリの設定	7
第3章 REGISTRY 設定例	17
第4章 REGISTRY トラブルシューティング	25
第5章 ルートシードデータ	26
第6章 認証	30
第7章 UDDI アノテーション	34
第8章 API を使った簡単な公開方法	38
第9章 サブスクリプション	44
第10章 M BEAN のサポート	51
付録A UDDI REGISTRY	52
付録B JUDDI クライアントの使用	54
付録C © 2011	57
付録D 改訂履歴	58

第1章 レジストリとは？

1.1. はじめに

JBoss Enterprise SOA Platform のレジストリ実装に関する概要については、本章を参照してください。

レジストリは、アプリケーションやビジネスにの集約ポイントを提供して、ここにサービスの情報を格納できます。レジストリは、従来の「マーケットプレイス」と同じレベルの情報、同じ幅のサービスを提供するよう期待されています。理想としては、レジストリは、動的な環境を提供することで **e-commerce** トランザクションの自動検出、実行がスムーズに行われるようにするはずですが、そのため、レジストリは単なる **e-business** のディレクトリではなく、サービス指向アーキテクチャーの基本的なコンポーネントとなっています。

1.2. レジストリが必要な理由

手動、または特別な手法を使用して、簡単にビジネスパートナーを検出、管理して、規模が小さな状態でビジネスパートナーとつながります。しかし、この手法は、サービス数やインタラクションの頻度が増え、環境が分散していくと、うまく対応できません。レジストリがあれば、一般的かつユビキタスな方法でサービスを検出・公開し同意した基準に従いソリューションを提供することができます。

パートナーのサービスが社内の技術と互換性があるかどうかをクエリすることができるように集約管理します。

レジストリはサービス指向アーキテクチャーの集約部となる理由を説明していきます。実行時、サービスリクエストが実際の動作と相関可能なコンタクト地点として、レジストリは機能します。

レジストリは、実行時、設計時の両方で利用されるサービス指向アーキテクチャーのアーチファクトすべてに対して、メタデータエントリを保持します。

これらのアーチファクトには、様々なサービスやセキュリティデータ（証明書や監査追跡ログなど）を表すサービスやアイテムで使用するサービスポリシーの説明、**Extensible Mark-Up Language (XML)** スキーマといったアイテムが含まれます。

設計フェーズで、業務プロセスのアーキテクトは、様々なサービスへの呼び出しを連携するレジストリを使用する場合があります。こうすることで、ワークフローまたは業務プロセスを作成します。

ビジネスアナリストの観点からすると、レジストリはインターネット検索エンジンと似ています。ただし、インターネット検索は **Web** ページを、レジストリは業務プロセスを検索するよう設計されているのです。開発者の観点からすると、様々な基準にマッチするサービスを検出、公開する際に、レジストリを使用します。



注記

レジストリのパフォーマンスと信頼性を改善するには、**Red Hat** はレジストリの複製、連携を推奨しています。こうすることで、**SPOF** (単一障害点) とならないようにします。

1.3. レジストリ VS レポジトリ

レジストリのロールは、サービスの記録、メタデータの検出、事前定義済みカテゴリへのエンティティの分類を行うことです。レポジトリと違い、業務プロセスの定義、**WSDL**、または取引契約に必要なその他の文書を格納する機能はありません。レジストリは、アイテムのカタログであり、レポジトリはこのようなアイテムを含む格納領域です。

1.4. サービス指向アーキテクチャのコンポーネント

「サービス指向アーキテクチャは、エージェントが「サービス」となっている分散システムの一つです。」



注記

詳細の定義については、http://www.w3.org/TR/2003/WD-ws-arch-20030808/#service_oriented_architecture の W3C Working Draft を参照してください。

サービス指向アーキテクチャの主要コンポーネントは、以下の通りです。

1. 交換されるメッセージ
2. サービスリクエスターやプロバイダーとして機能するエージェント
3. メッセージのやり取りを可能にする共有トランスポートメカニズム

サービスとは、システムとユーザー間でやり取りを行うメッセージを指します。

サービス指向アーキテクチャでは、サービスプロバイダー、ブローカー、リクエスターという、重要なロールが3つあります。それぞれの定義を以下に示します。

サービスプロバイダー

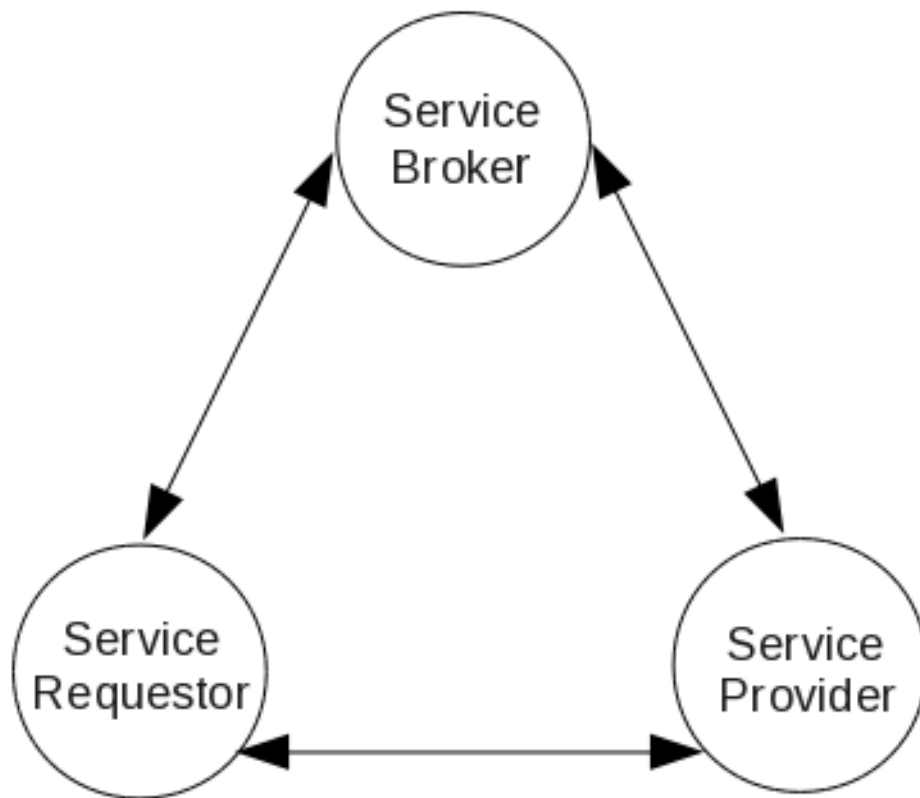
サービスプロバイダーは、サービスへのアクセス管理、サービスの説明の作成、サービスブローカーへの公開を行います。

サービスブローカー

サービスブローカーは、サービスの説明に関するレジストリをホストします。サービスリクエスターとサービスプロバイダーをつなぐ役割を果たします。

サービスリクエスター

サービスリクエスターは、サービスの検出を行います。サービスブローカーが提供するサービスの説明を検索することで、サービスを検出します。また、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。



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

Universal Description, Discovery and Integration Registry(UDDI) は、*web* サービスのディレクトリです。設計時、実行時にクエリを実行することで、UDDI を使用してサービスの検索を行います。

UDDI Registry 内では、情報はページに分類されます。

- グリーンページは、クライアントと提供のサービスをバインド可能な情報を提供します。
- イエローページは、所属する産業をベースにビジネスを分類する際に利用します。
- ホワイトページには、サービスを提供する企業の名前、住所、その他の連絡先など一般情報が含まれます。

UDDI は、サービスの説明を公開します。一般的な UDDI Registry には、*uniform resource locator*(URL) が含まれており、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を参照します。

1.6. REGISTRY と JBOSS SERVICE-ORIENTED ARCHITECTURE PLATFORM

Registry は、JBoss Enterprise SOA Platform の主要な部分です。サービスを SOA Platform の ESB にデプロイすると、サービスのエンドポイント参照が Registry に格納されます。

Registry を自動的 (サービスをまず起動) または 手動 (ユーザーによる介入) で更新することで設定します。

Registry はサービスを一覧表示しますが、ステータスを判断することはできません。つまり、**Registry** に表示されているエンドポイント参照が有効であるか保証はありません (例えば、問題があるか、または有効でないサービスを指している場合があります)。これは、**JBoss Enterprise SOA Platform** にはライフサイクルモニタリング機能がいないためです。そのため、管理者は手動で無効なエンドポイント参照を削除、更新する必要があります。

エンドポイント参照に関して **Registry** から警告やエラーメッセージを受け取った場合、サービスの担当者に通知するというのがベストプラクティスです。

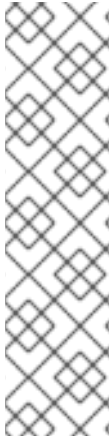


重要

ESB サービスは、自身のエンドポイント参照を自動で作成します。このようなエンドポイントは、内部実装であるため、変更しないようにしてください。変更した場合は、Red Hat のサポート対象外となります。

第2章 レジストリの設定

レジストリの設定方法については、本章を参照してください。



注記

バージョン 5.1 以降は、**JBoss Enterprise SOA Platform** に **JBoss Enterprise Data Services (EDS)** ソフトウェアが含まれています。EDS は JDBC ドライバーまたは web サービスとして公開されるため、ESB サービスは以下のいずれかの形式で消費することができます。

EDS Virtual Database (VDB) 向けの JDBC 接続の文字列は、標準の JDBC 接続の文字列と少し違います。VDB 向けの JDBC 接続の正しい形式は **jdbc:teiid:vdb_name@mm://localhost:31000** です。

詳細は、**EDS Developer Guide** を参照してください。

SOA Platform に含まれる jUDDI (<http://juddi.apache.org>) Registry は様々な設定が可能です。JBoss Enterprise Service Bus (ESB) は、Registry とのやり取りをすべてレジストリインターフェースを介して指示を出し、デフォルトバージョンが **Apache Scout** を使います。

Apache Scout

Apache Scout は JAXR 実装です。

レジストリインターフェース

これは、ESB が jUDDI Registry と通信する手段です。



注記

JBoss Enterprise SOA Platform は、他の UDDI レジストリ (**SOA Software SMS** や **HP Systinet**) もサポートしています。

手順2.1 レジストリ設定の定義

- **\$SOA_ROOT/server/\$PROFILE/deployers/esb.deployer/jbossesb-properties.xml** ファイルのレジストリのセクションは、以下のプロパティを定義しています。

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

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

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

```

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

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

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

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

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

```

以下に設定可能なプロパティを挙げています。

表2.1 Registry プロパティ

プロパティ	説明
org.jboss.soa.esb.registry.implementationClass	JBoss ESB Registry インターフェースを実装するクラスです。 JAXRRegistryImpl の実装 1 つが含まれています。 JAXRRegistry インターフェースを使用します。
org.jboss.soa.esb.registry.factoryClass	JAXR <i>ConnectionFactory</i> 実装のクラス名です。

プロパティ	説明
<code>org.jboss.soa.esb.registry.queryManagerURI</code>	サービスのクエリに JAXR で使用する URI です。
<code>org.jboss.soa.esb.registry.lifeCycleManagerURI</code>	JAXR が編集に使用する URI です。
<code>org.jboss.soa.esb.registry.user</code>	編集に使用するユーザー名です。
<code>org.jboss.soa.esb.registry.password</code>	指定のユーザーのパスワードです。
<code>org.jboss.soa.esb.scout.proxy.uddiVersion</code>	クエリの UDDI バージョンです。
<code>org.jboss.soa.esb.scout.proxy.uddiNamespace</code>	UDDI 名前空間です。
<code>org.jboss.soa.esb.scout.proxy.transportClass</code>	Apache Scout が UDDI Registry にアイテムを送信する際に利用するクラスです。
<code>org.jboss.soa.esb.registry.interceptors</code>	これは Registry に適用されるインターセプター一覧です。ESB には、インターセプターが 2 つあり、1 つは InVM 登録の処理に、もう 1 つは Registry にキャッシュを提供します。 デフォルトのインターセプターは、InVM インターセプター用にエントリ 1 つのみを含みます。
<code>org.jboss.soa.esb.registry.cache.maxSize</code>	これがキャッシュ内に含むことができる最大サーバーエントリ数です。この値を超えると、エントリが最も古い順番に削除されます。デフォルトは、 100 です。
<code>org.jboss.soa.esb.registry.cache.validityPeriod</code>	キャッシュインターセプターに設定された有効期限です。この値はミリ秒で指定し、デフォルトは 600000 (10 分) となっています。キャッシュが失効しないようにするには、この値を 0 に設定してください。
<code>org.jboss.soa.esb.registry.orgCategory</code>	ESB の <i>組織カテゴリ</i> 名です。デフォルトは org.jboss.soa.esb.:category です。

2.1. REGISTRY のコンポーネント

Registry の各部分についての詳細は本章を参照してください。

すべての Registry コンポーネントについての説明を以下に示しています。

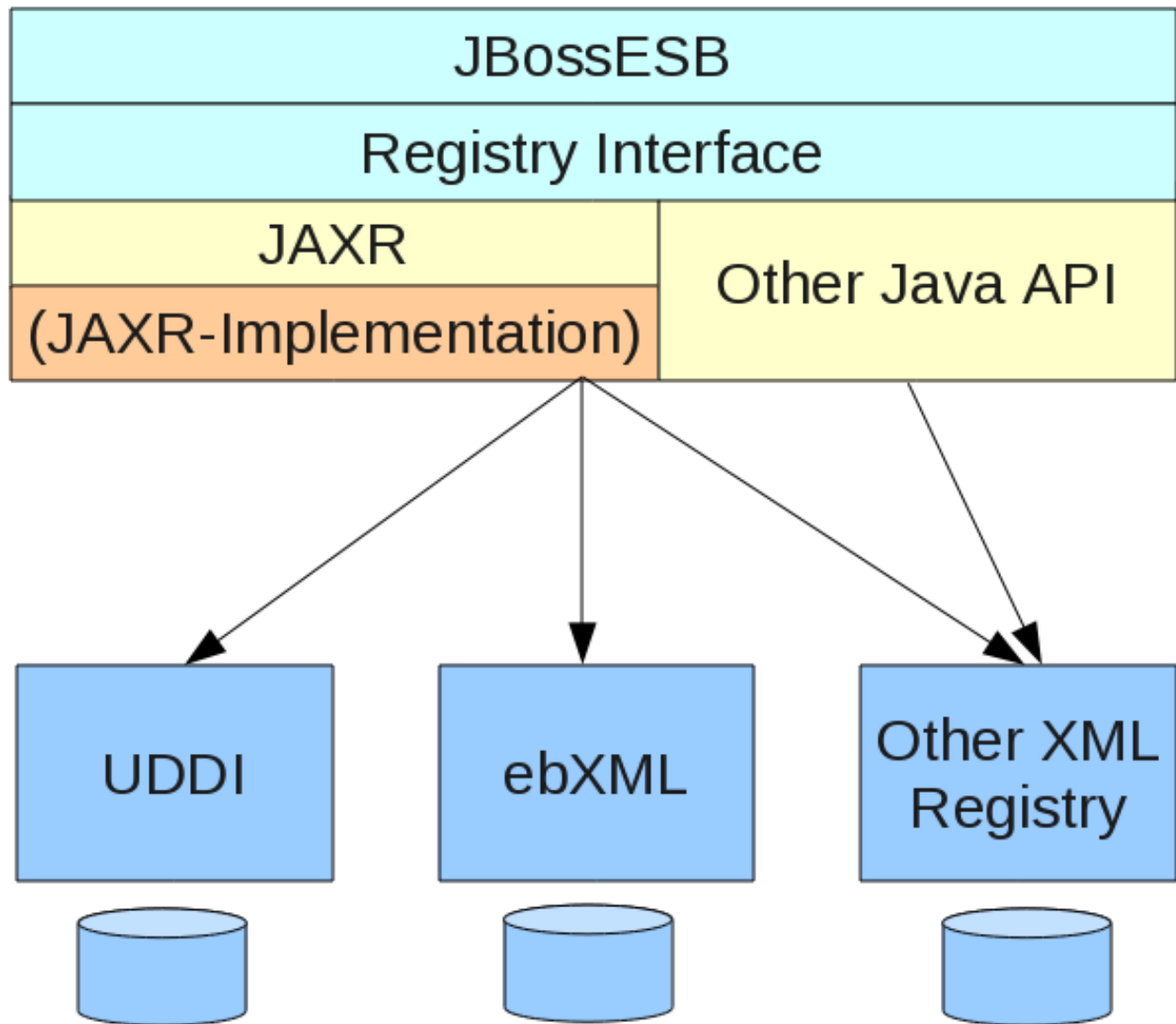
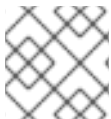


図2.1 Registry コンポーネントのアーキテクチャー

トップダウン図から、以下がわかります。

1. JBoss Enterprise Service Bus は、**registry interface** 経由で Registry とのやり取りをすべて送ります。
2. その後、このインターフェースの 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 (デフォルトは **Apache Scout**)。
4. 次に Apache Scout は jUDDI Registry を呼び出します。



注記

ここで挙げたものはデフォルトのみです。別の設定オプションは、多数存在します。



注記

デフォルトでは、このクラスは JAXR アプリケーションプログラミングインターフェースを使用します。**org.jboss.soa.esb.registry.implementationClass** API は、XML ベースのレジストリやレポジトリであればどれにでも接続できるため、便利です。しかし、別の API を使用することも可能です。

手順2.2 他の API の使用

- 新しい **SystinetRegistryImplementation** クラスを記述して、このプロパティ内から参照します。

2.2. JAXR 実装の設定

手順2.3 JAXR 実装の設定

1. 特定の JAXR 実装を選択します。
2. プロパティを設定してクラスを構成します (JBoss Enterprise SOA Platform は **Apache Scout** を使用して、デフォルトでこのプロパティは **org.apache.ws.scout.registry.ConnectionFactoryImpl** という Scout factory クラスに設定されています)。
3. Registry の場所を提供することで、JAXR 実装を設定します。
org.jboss.soa.esb.registry.queryManagerURI、
org.jboss.soa.esb.registry.lifeCycleManagerURI、
org.jboss.soa.esb.registry.securityManagerURI を編集します。

2.3. JUDDI トランスポートの使用

Apache Scout を使用する際は、特別な任意パラメーター (**org.jboss.soa.esb.scout.proxy.transportClass**) を設定することができます。これは、**Scout** と jUDDI Registry の間の通信を容易にする **transport** クラスです。

Scout を使用して jUDDI v. 3 と通信する場合、transport クラスを **LocalTransport** のままにして、jUDDI registry のトランスポート (InVM、RMI、WS) を使用するように **server/config/deploy/jbossesb.sar/META-INF/esb.juddi.client.xml** ファイルを設定します。このファイルは、Registry のノードを定義します。

ノード

ノードは、jUDDI Registry の場所です。

手順2.4 トランスポートの選択

1. ノード設定を使用してどのトランスポートを使用するか選択します。

```
<node>
  <!-- required 'default' node -->
  <name>default</name>
  <description>Main jUDDI node</description>
  <properties>
    <property name="serverName" value="localhost" />
    <property name="serverPort" value="8880" />
  </properties>
</node>
```

```

    </properties>
    <!-- JAX-WS Transport

<proxyTransport>org.apache.juddi.v3.client.transport.JAXWSTransport<
/proxyTransport>

<custodyTransferUrl>http://${serverName}:${serverPort}/juddiv3/servi
ces/custody-transfer?wsdl</custodyTransferUrl>

<inquiryUrl>http://${serverName}:${serverPort}/juddiv3/services/inqu
iry?wsdl</inquiryUrl>

<publishUrl>http://${serverName}:${serverPort}/juddiv3/services/publ
ish?wsdl</publishUrl>

<securityUrl>http://${serverName}:${serverPort}/juddiv3/services/sec
urity?wsdl</securityUrl>

<subscriptionUrl>http://${serverName}:${serverPort}/juddiv3/services
/subscription?wsdl</subscriptionUrl>

<subscriptionListenerUrl>http://${serverName}:${serverPort}/juddiv3/
services/subscription-listener?wsdl</subscriptionListenerUrl>

<juddiApiUrl>http://${serverName}:${serverPort}/juddiv3/services/jud
di-api?wsdl</juddiApiUrl>
-->
    <!-- In VM Transport Settings

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

<custodyTransferUrl>org.apache.juddi.api.impl.UDDICustodyTransferImp
l</custodyTransferUrl>

<inquiryUrl>org.apache.juddi.api.impl.UDDIInquiryImpl</inquiryUrl>

<publishUrl>org.apache.juddi.api.impl.UDDIPublicationImpl</publishUr
l>

<securityUrl>org.apache.juddi.api.impl.UDDISecurityImpl</securityUrl
>

<subscriptionUrl>org.apache.juddi.api.impl.UDDISubscriptionImpl</sub
scriptionUrl>

<subscriptionListenerUrl>org.apache.juddi.api.impl.UDDISubscriptionL
istenerImpl</subscriptionListenerUrl>

<juddiApiUrl>org.apache.juddi.api.impl.JUDDIApiImpl</juddiApiUrl>
-->
    <!-- RMI Transport Settings -->
<proxyTransport>org.apache.juddi.v3.client.transport.RMITransport</p
roxyTransport>
    <custodyTransferUrl>/juddiv3/UDDICustodyTransferService</custodyTra
nsferUrl>

```



```

<inquiryUrl>/juddiv3/UDDIInquiryService</inquiryUrl>
<publishUrl>/juddiv3/UDDIPublicationService</publishUrl>
<securityUrl>/juddiv3/UDDISecurityService</securityUrl>
<subscriptionUrl>/juddiv3/UDDISubscriptionService</subscriptionUrl>
<subscriptionListenerUrl>/juddiv3/UDDISubscriptionListenerService</s
ubscriptionListenerUrl>
<juddiApiUrl>/juddiv3/JUDDIApiService</juddiApiUrl>
<javaNamingFactoryInitial>org.jnp.interfaces.NamingContextFactory</j
avaNamingFactoryInitial>
<javaNamingFactoryUrlPkgs>org.jboss.naming</javaNamingFactoryUrlPkg
s>
<javaNamingProviderUrl>jnp://localhost:1099</javaNamingProviderUrl>
</node>

```

2. デフォルトでは *Remote Method Invocation* (RMI) の設定が有効になっています。トランスポートを切り替えるには、RMI の設定をコメントアウトして、使用したいものを有効にします。

transport を設定する場合、以下のアイテムを指定する必要があります。

- **proxyTransport**
- 対応の UDDI アプリケーションプログラミングインターフェースすべてに対する URL (**inquiry**、**publish**、**security**、**subscription**、**subscription-listener**、**custodytransfer**)
- jUDDI アプリケーションプログラミングインターフェースの URL
- RMI トランスポート (JNDI 設定を含みます)

2.4. APACHE SCOUT と JUDDI の使用

現在、**org.jboss.soa.esb.scout.proxy.transportClass** クラス実装が 4 つあり、SOAP、SAAJ、RMI、Embedded Java (Local) それぞれに対して 1 つずつ存在することになります。jUDDI Registry とやり取りを行う場合は、**transportClass** を **LocalTransport** に設定したままで残して、Registry のトランスポート (それぞれ、InVM、RMI、WS) を使用するように **uddi.xml** ファイルを設定します

また、以下の 2 つの要件が満たされているか確認してください。

1. jUDDI データベースにアクセスできなければなりません。アクセスできるかを確認するには、データベースにスキーマを作成して、**jbossesb publisher** を追加します (**product/install/jUDDI-registry** ディレクトリには、最も一般的なデータベース向けの **database-create** スクリプトが含まれています。)



注記

ユーザーにテーブル作成の権限が与えられている場合は、システムにより自動でデータベースを生成することができます。

jUDDI Registry は、関連の *Hibernate* 方言が存在するデータベースを作成することができます。

2. **esb.juddi.xml** と **esb.juddi.client.xml** は必須ファイルです。これらのファイルには、実際の jUDDI 設定が含まれています。



注記

ソフトウェアと別の UDDI レジストリの間でやり取りをさせたい場合は、**Apache Scout** の JAXR トランSPORTを使用します。このクラスには 4 つの実装があり、それぞれ SOAP、SAAJ、RMI、Embedded Java ベースとなっています。

トランSPORTを変更する場合、クエリとライフサイクルの **Uniform Resource Indicator** も変更するようにしてください。変更する方法を例示したサンプルコードを以下に提示しています。

SOAP

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

RMI

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

Local

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

2.5. REGISTRY インターセプター

Registry はリクエストをインターセプトできます。インターセプタースタックを使用して行います。このスタック内にある各インターセプターには、以下の機能があります。

- リクエストにサービスを提供
- リクエストに直接レスポンス
- 下層にあるインターセプターまたはレジストリ実装から受け取ったレスポンスを拡大

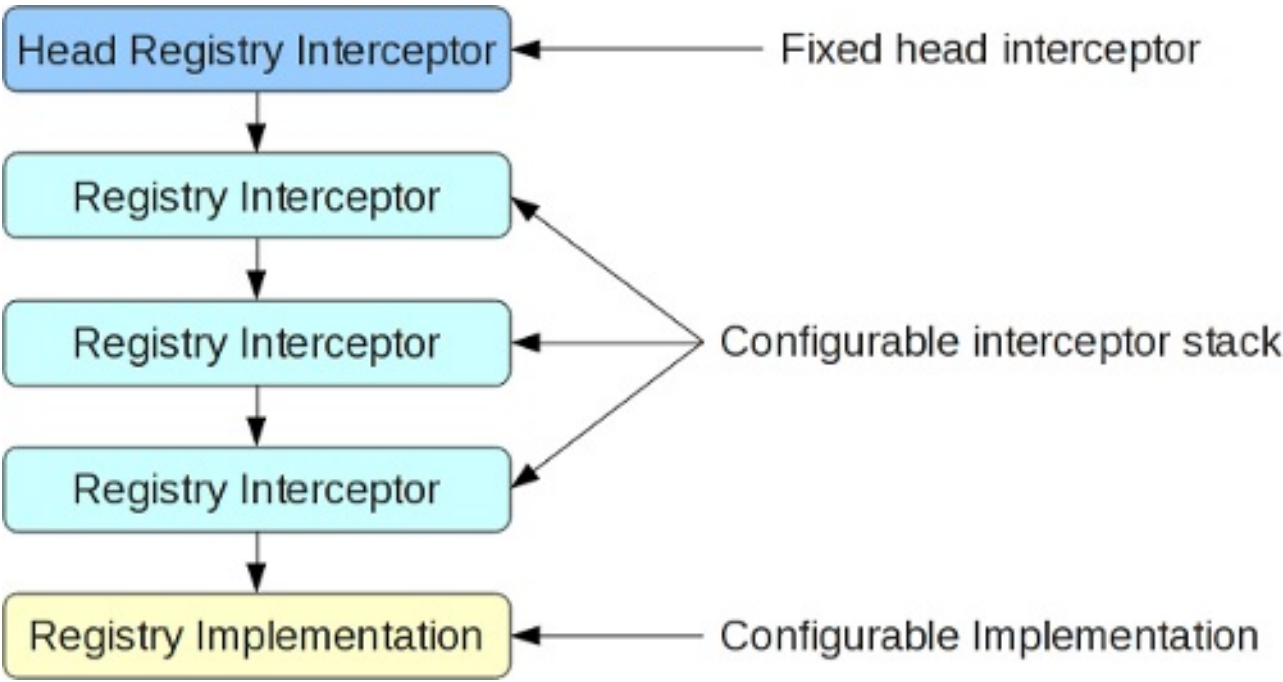


図2.2 Registry インターセプター

現在の実装では、インターセプターが2種、提供されています。

手順2.5 インターセプタースタックの設定

- `jbossesb-properties.xml` ファイルを開き `org.jboss.soa.esb.registry.interceptors` プロパティを変更します。

表2.2 Registry プロパティ

プロパティ	説明
<code>org.jboss.internal.soa.esb.services.registry.InVMRegistryInterceptor</code>	InVM レジストリインターセプターは、InVM エンドポイント参照の処理を担当します。同じサーバーインスタンス内で実行するサービスにより登録されます。InVM EPR や関連サービスの情報は、インターセプター内にキャッシュされ、後続のインターセプターに伝搬されず、後続のインターセプターやレジストリクエリからの決定を最適化して呼び出し元に返されます。

プロパティ	説明
<code>org.jboss.internal.soa.esb.services.registry.CachingRegistryInterceptor</code>	<p>キャッシュレジストリのインターセプターは、エンドポイント参照のキャッシュと関連のサービスを保持して、LRU ベースまたは、情報が失効した後にキャッシュから情報をエビクトします。</p> <p>インターセプターは、<code>jbossesb-properties.xml</code> 内で <code>org.jboss.soa.esb.registry.cache.maxSize</code> と <code>org.jboss.soa.esb.registry.cache.validityPeriod</code> プロパティを使用して設定することができます。</p>

第3章 REGISTRY 設定例

jUDDI Registry の様々な設定について、本章を参照してください。



注記

JBoss Enterprise SOA Platform には、jUDDI Registry を自動設定するツールが含まれています。このツールは、`${SOA_ROOT}/tools/schema/` サブディレクトリに入っています。使用方法については、**管理ガイド**の「データベースの切り替え」の章にあります。

3.1. JUDDI REGISTRY の埋め込み

サーバーコンポーネントが1つの jUDDI Registry を共有するようにしたい場合は、jUDDI Registry を埋め込んでください。この手法は、JBoss Enterprise SOA Platform 自体の複数のインスタンスが同じレジストリを共有できるようになります。このコンセプトについて例示しています。

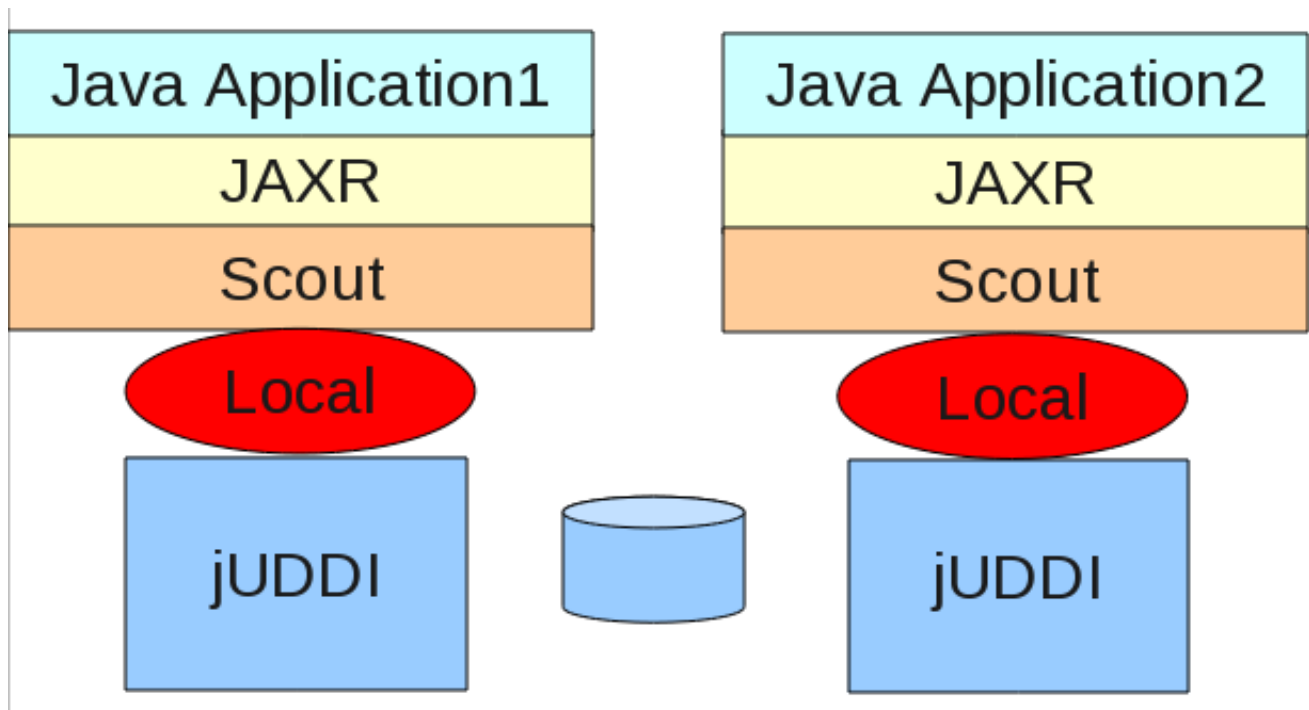


図3.1 埋め込み jUDDI

手順3.1 jUDDI Registry の埋め込み

- jUDDI Registry を埋め込むには、以下のプロパティを設定します。

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

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

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

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

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

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

</properties>
```

3.2. REMOTE METHOD INVOCATION を使用するための JUDDI REGISTRY の設定

2 つ目の設定方法では、Remote Method Invocation を使います。以下でこのコンセプトについて説明します。

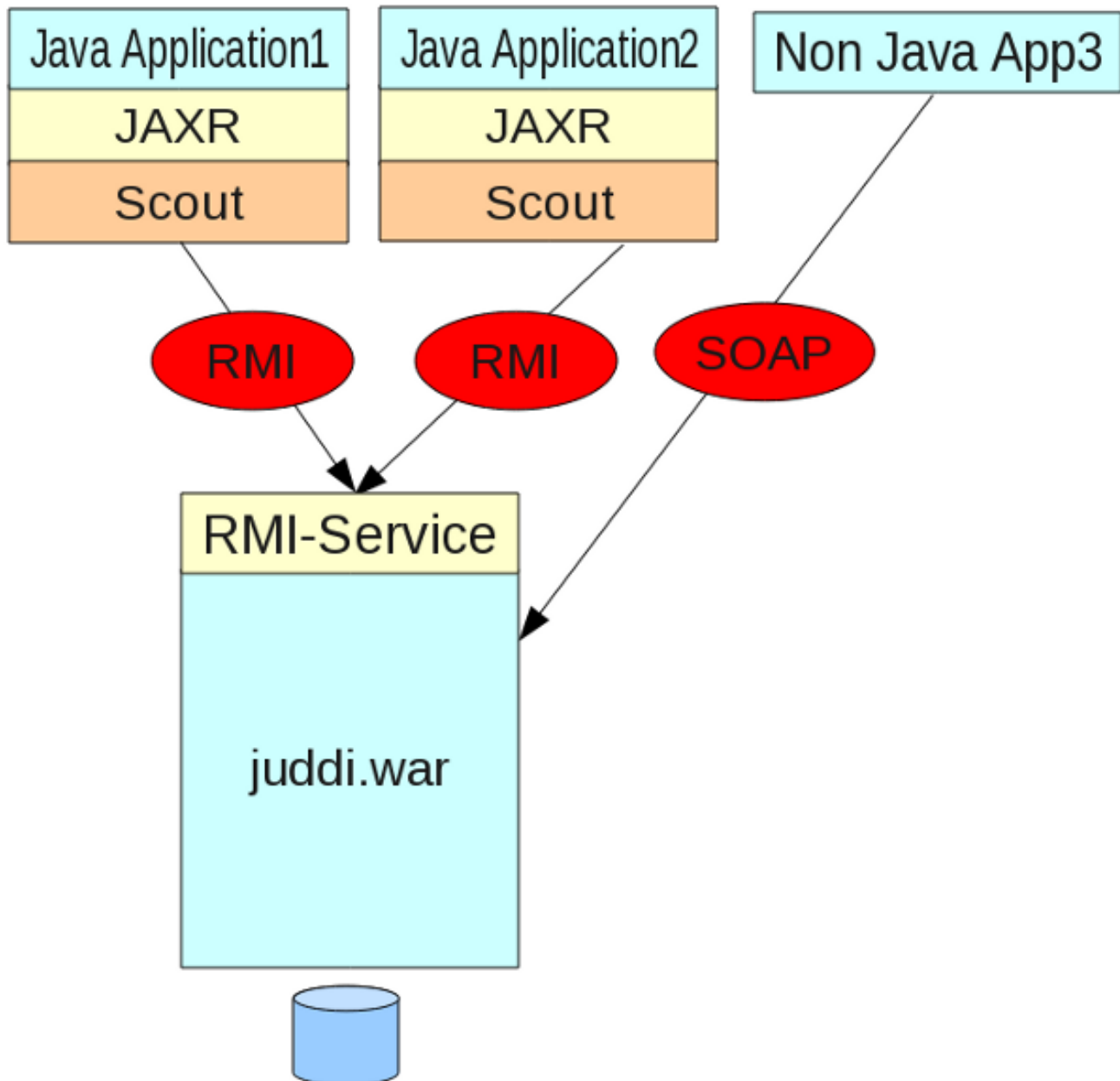


図3.2 Remote Method Invocation

手順3.2 jbossesb.sar ファイル経由で Remote Method Invocation を使用するための jUDDI Registry の設定

1. Remote Method Invocatoin サービスを呼び出す jUDDI Registry のバージョン 1 つをデプロイするだけです (JBoss Enterprise Service Bus は、デフォルトで Remote Invocation Service をデプロイします。ESB は **jbossesb.sar** アーカイブ内で Registry を起動して、Remote Method Invocation サービスを自動登録するアーカイブと同じです)。

設定プロパティを以下に示しています。

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

```

        value="org.apache.ws.scout.registry.ConnectionFactoryImpl"/>

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

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

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

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

```

juddi.war アーカイブは、**Remote Method Invocation** サービスを呼び出すために設定されます。**web.xml** ファイルに以下のような設定をします。

```

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

```

2. **juddi.properties** ファイルを編集して、以下の JNDI 設定を含めます。

```

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

```



重要

RMI クライアントのクラスパスに **scout-client.jar** を含めるのを忘れないでください。

3.3. RMI サービスの独自の JNDI 登録を使用した REMOTE METHOD INVOCATION

なんらかの理由で、**juddi.war** アーカイブをデプロイしたくない場合、JUDDI Registry と実行しているのと同じ Java 仮想マシンで、**Enterprise Service Bus** コンポーネントの1つを設定します。このコンセプトについて以下に例示しています。

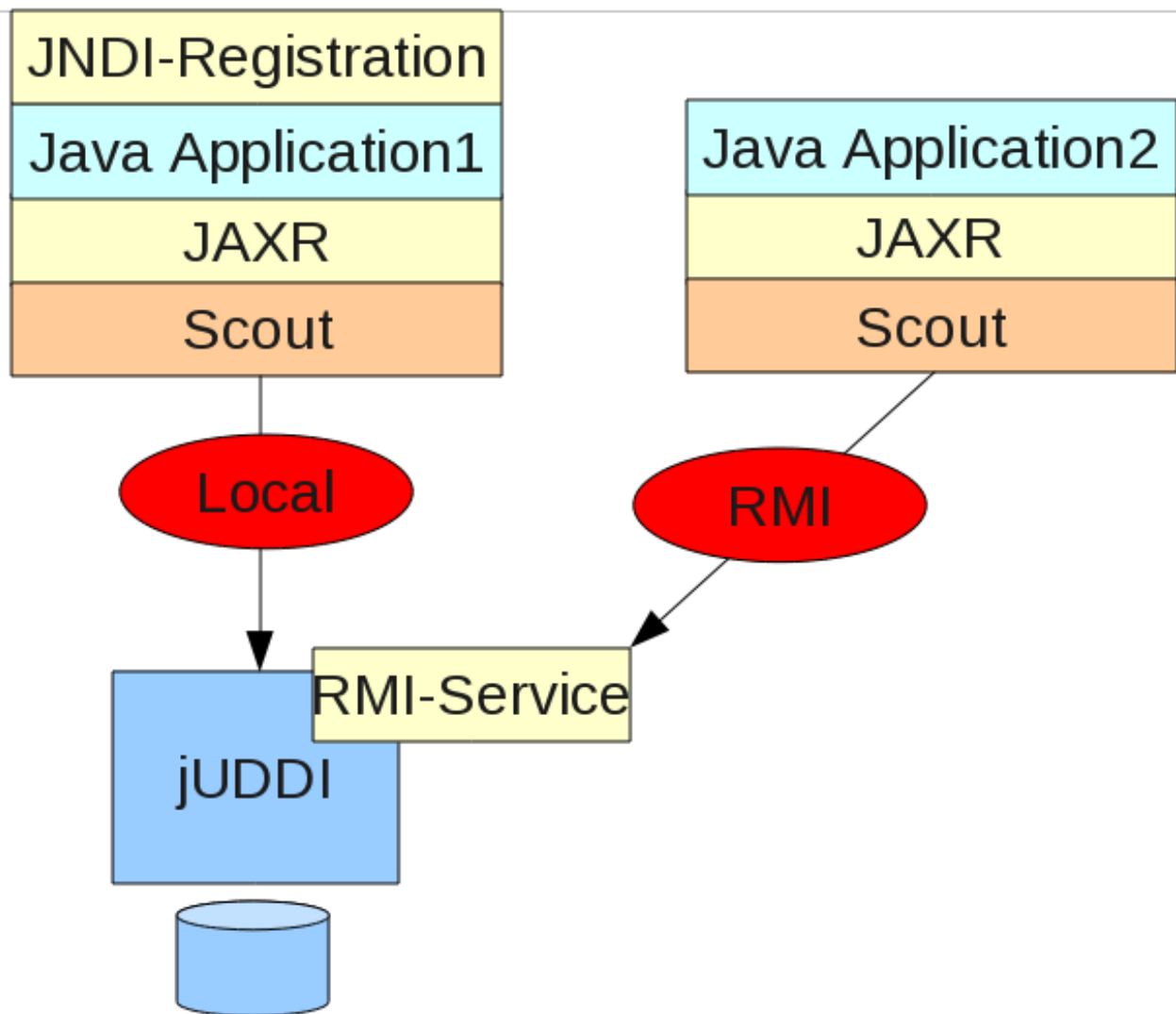


図3.3 独自の JNDI 登録を用いた RMI

手順3.3 独自の JNDI 登録を用いた RMI の設定

1. 1つ目のアプリケーションでは、以下のローカルの設定を行います。

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

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

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

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

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

  <property name="org.jboss.soa.esb.registry.user"
```

```

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

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

```

2. 2 つ目のアプリケーションでは、以下のリモートメソッド呼び出しの設定を行います。

```

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

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

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

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

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

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

```

3. **queryManagerURI** と **lifeCycleManagerURI** クラスのホスト名を jUDDI Registry を実行中のホスト (アプリケーションのレジストリも実行しているホスト) へ参照させます。明らかに、アプリケーションレジストリは、ネーミングサービスへのアクセスが必要です。

4. 以下の設定を使用してアプリケーションを登録します。

```

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

InitialContext context = new InitialContext(env);
Inquiry inquiry = new InquiryService();
log.info("Setting " + INQUIRY_SERVICE + ", " +
inquiry.getClass().getName());

```

```

mInquiry = inquiry;
context.bind(INQUIRY_SERVICE, inquiry);
Publish publish = new PublishService();
log.info("Setting " + PUBLISH_SERVICE + ", " +
publish.getClass().getName());
mPublish = publish;
context.bind(PUBLISH_SERVICE, publish);

```



重要

RMI クライアントのクラスパスに **scout-client.jar** ファイルを含めるのを忘れないでください。

3.4. SOAP

また、jUDDI Registry とのやり取りに SOAP (**Apache Scout** 経由) を使うという方法もあります。

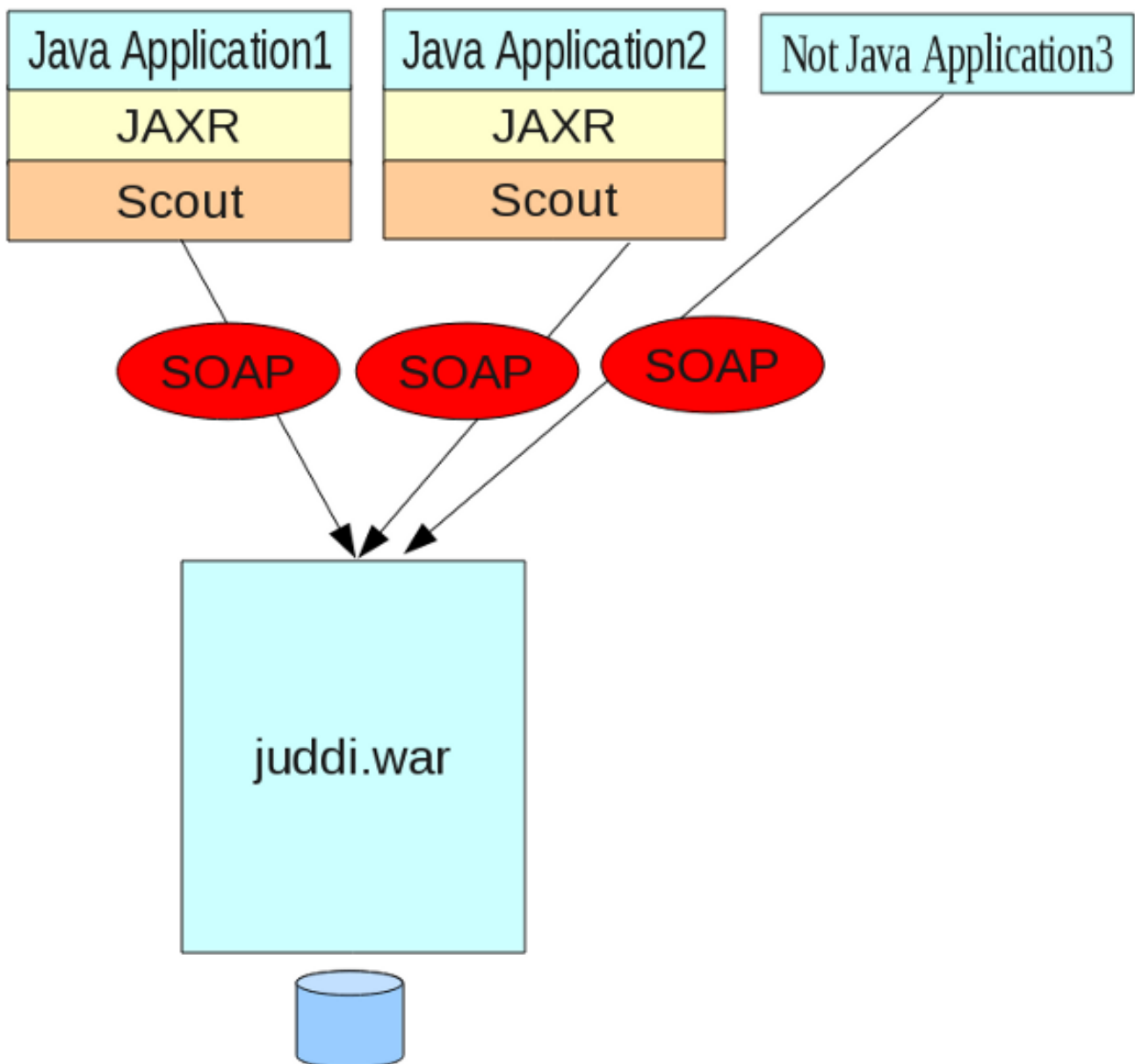


図3.4 SOAP ベースの通信

手順3.4 SOAP を使用するための Apache Scout 設定

- `juddi.war` アーカイブをデプロイして、データソースを設定します。



重要

`web.xml` ファイルの **RegisterServicesWithJNDI** サブレットをコメントアウトして、RMI サービスもシャットダウンするとよいでしょう。

サンプルプロパティを以下に示します。

```
<properties name="registry">
  <property name="org.jboss.soa.esb.registry.implementationClass"
    value="org.jboss.internal.soa.esb.services.registry.JAXRRegistryImp
1"/>

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

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

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

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

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

第4章 REGISTRY トラブルシューティング

4.1. SCOUT と JUDDI での注意点

- リモートのメソッド呼び出しアプローチを使用する場合、**juddi-client.jar** ファイルを取得するようにしてください(それぞれのプロファイルの場所にあります。**./server/default/deployers/esb.deployer/lib/juddi-client-3.1.0.jar**、**./server/all/deployers/esb.deployer/lib/juddi-client-3.1.0.jar**、または**./server/production/deployers/esb.deployer/lib/juddi-client-3.1.0.jar**。)
- **jbossesb-properties.xml** ファイルが **class-path** にあり、正しく読み込まれるようにしてください。そうでないと、Registry はクラスをインスタンス化するときの名前に "null" を使おうとします。
 - **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ます。
 - **persistence.xml** ファイルの設定が有効で、選択した **Hibernate** 方言がデータベースで使用するものと同じであることを確認してください。
- **esb.juddi.xml** ファイルが **class-path** にあるよう確認してください。これには、JUDDI Registry の設定の一部が含まれています。
- サービスが失敗するか、クリーンにシャットダウンされない場合、Registry に古いエントリが残る可能性があります。このようなエントリは手動で削除します。

4.2. 詳細情報

Registry のトラブルシューティングに関する詳細情報は、以下を確認してください。

- JBoss jUDDI Wiki: <http://www.jboss.org/community/docs/DOC-11217>
- JBoss ESB User Forum: <http://community.jboss.org/en/jbossesb?view=discussions>.

第5章 ルートシードデータ

5.1. はじめに

UDDI v3 では、各レジストリには *root publisher* が必要です。*root publisher* は、UDDI サービスのオーナー (*inquiry* や *publication* など) です。ノード毎に *root publisher* は1つのみとなっています。

jUDDI Registry には、*root* アカウント用にデフォルトのシードデータが同梱されています。*juddi_install_data/* の *juddi-core-3.x.jar* 配下にあります。

デフォルトでは、jUDDI Registry が *root* と *uddi* のパブリッシャーを 2 つインストールします。*Root* には *root* パーティションを所有し、*uddi* はその他の全シードデータを所有します (定義済みの *tModels* など)。

5.2. シードデータファイル

パブリッシャーごとに、シードデータファイルが 4 つあります。

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

root_Publisher.xml のコンテンツは以下のとおりです。

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

パブリッシャーごとに、独自のキー生成スキーマ (*key generator schema*) を持ち、カスタム生成されたキーと、他のパブリッシャーが生成したキーが同じにならないようにします。これを実装するには、パブリッシャーごとに、**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>
```

つまり、*root publisher* で使用されるキーは *uddi:juddi.apache.org:<text-of-choice>* の形式でなければなりません。別の形式を使用する場合、*illegal key* エラーが表示されます。

Root publisher は、**KeyGenerator** を1つしか所有できませんが、他のパブリッシャーは複数所有することができます。

できる限り、**KeyGenerators** は共有しないでください。複数の**KeyGenerator tModel**からパブリッシャーを確認したい場合、**<publisher>tModels.xml** ファイルを使用してください。

最後に **<publisher>BusinessEntity.xml** ファイルを使用して、**Business** と **Service** データを設定できます。**root_BusinessEntity.xml** で、サービスを指定します。

```
<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://${juddi.server.name}:${juddi.server.port}/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://${juddi.server.name}:${juddi.server.port}/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>
    <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>

```



警告

シードプロセスは、データベースに **publishers** がない場合のみ開始しますが、**juddi.seed.always** を **true** 設定すれば開始されません (ルートデータファイルの例外が全ファイルに再度適用されます)

このプロセスでは、再度シーディングされたエンティティに追加された情報は消去されてしまうため、データがなくなってしまう場合があります。これは、データのマージが行われないためです。

5.3. シードデータのトークン

お気づきの方もいらっしゃるかもしれませんが、**root_BusinessEntity.xml** ファイル (**\${juddi.server.baseurl}**) にトークンの値を設定するには、**juddiv3.properties** ファイルの該当セクションを変更します (値の代入は実行時に行われるため、必要であれば別のノードが独自の値を代入可能です)。

5.4. カスタマーシードデータ

jUDDI Registry で提供されているシードデータを使いたくない場合があります。最も簡単にカスタマイズする方法は、**juddiv3.war/WEB-INF/classes/** ディレクトリに **juddi_custom_install_data** を追加することです (こうすることで、**juddi-core-3.x.jar** アーカイブを変更する必要がなくなります)。

また、**root publisher** は **root** と呼ばれず、**juddiv3.properties** ファイルの **juddi.root.publisher** プロパティを下記のものの以外に設定する必要があります。

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

juddiv3.war アーカイブファイルには、サンプルデータのディレクトリが2つ含まれています。1つ目は、**Sales Affiliate** で、もう1つは **Marketing Affiliate** です。**Sales Seed Data** を使うには、**juddiv3.war/WEB-INF/classes/** のディレクトリの名前を変更します。以下のコマンドを実行してください。

```
mv RENAME4Sales_juddi_custom_install_data juddi_custom_install_data
```

初めて jUDDI Registry を開始する前に実行してください。次に、このデータを使用してデータベースを生成します。



注記

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て、**Tomcat** を再起動します (**juddiv3.properties** ファイルのトークンを忘れずに設定してください) **mv RENAME4Sales_juddi_custom_install_data juddi_custom_install_data**

第6章 認証

6.1. はじめに

jUDDI Registry への書き込み権限を適切なものにするため、この Registry へのリクエストは、有効な **authToken** がなければなりません。



重要

読み取り権限は制限がないため、レジストリへのクエリも制限がありません。

有効な **authToken** を取得するには、**getAuthToken()** リクエストを出す必要があります。リクエストを出すと **GetAuthToken** オブジェクトが渡されますので、このオブジェクトに、**userid** と **credential (password)** を設定してください。

```
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);
```

juddi.properties 設定ファイルの **juddi.auth** プロパティを使って、jUDDI Registry が **GetAuthToken** リクエストにより渡された認証情報をチェックするよう設定します (デフォルトでは **JUDDIAuthenticator** 実装を使用します)。

独自の認証実装や、下記のいずれかを使用することができます。認証実装には、**org.apache.juddi.auth.Authenticator** インターフェースを使用して、**juddi.auth** プロパティは実装クラスを参照するようにします。

認証は、**認証フェーズ**と**識別フェーズ**として知られる 2 フェーズプロセスとなっています。これらのフェーズは、**Authenticator** インターフェースのメソッドで表現されます。

認証フェーズは、**GetAuthToken** リクエストが出されると発生します。このフェーズの目的は、**userid** や認証情報を有効な **publisher id** に変換することです。**publisher id** (UDDI の用語では **認証名** と呼ばれる) は、UDDI 内でオーナーシップを割り当てる値です。新しいエンティティが作成されると、必ず **publisher** の認証名でオーナーシップがタグ付けされます。

publisher id の値は、jUDDI Registry とは関連がありません。唯一の要件としては、新しいエンティティに割り当てられるように **non-null** の **id** が存在しなければならない点です。**GetAuthToken** リクエストが完了すると、**authentication token** が呼び出し元に発行されます。

その後、UDDI API に認証を必要とする呼び出しを行うと、**GetAuthToken** リクエストから発行されたトークンを提供する必要があります。その後、識別フェーズに移動します。

識別フェーズが担当するのは、認証トークンを (またはトークンに関連付けられた **publisher id**) 有効な **UddiEntityPublisher** オブジェクトに変えることです。このオブジェクトには、UDDI エンティティのオーナーシップを処理する際に必要な全プロパティが含まれています。そのため、トークン (または **publisher id**) を使用して、**publisher** を識別します。

この 2 つのフェーズは、UDDI 認証構造に準拠して、独自の認証メカニズムを提供したいユーザーに柔軟性を提供します。

jUDDI Register 外で、認証と **publisher** プロパティをすべて処理できます。しかし、デフォルトでは、Registry が **UddiEntityPublisher** のサブクラスである **Publisher** エンティティを提供します。このサブクラスは、jUDDI Registry 内で **publisher** プロパティを永続化します。

6.2. JUDDI 認証

jUDDI Registry がデフォルトで提供する認証メカニズムは、**JUDDIAuthenticator** です。**JUDDIAuthenticator** の認証フェーズは、**Publisher** テーブルの記録とサブミットされた **user id** が一致するか確認するだけです。認証情報の確認は行いません。認証プロセスで、**Publisher** がないと判断されると、その場で追加されます。



警告

実稼働環境では jUDDI 認証プロセスは使用しないでください。提供された認証情報をすべて受け入れ、レジストリにアクセスする際にクライアントが認証を行う必要がなくなってしまうためです。

識別フェーズでは、**publisher id** を使用して、**Publisher** レコードをリトリーブし、返します。**Publisher** は **UddiEntityPublisher** から必要とされるプロパティをすべて継承します。

```
juddi.auth = org.apache.juddi.auth.JUDDIAuthentication
```

6.3. XMLDOCAUTHENTICATION

XMLDocAuthentication 実装を使用するには、XML ファイルをクラスパスに追加する必要があります。

デフォルトでは、このファイルの名前は **juddi-users.xml** で、内容は以下のようにになっています。

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<juddi-users>
  <user userid="anou_mana" password="password" />
  <user userid="bozo" password="clown" />
  <user userid="sviens" password="password" />
</juddi-users>
```

juddi.properties ファイルの中身は、以下ようになります。

```
juddi.auth = org.apache.juddi.auth.XMLDocAuthentication
juddi.usersfile = juddi-users.xml
```

認証フェーズは、ユーザー id とパスワードが XML ファイルの値と一致するか確認します。識別フェーズは、単に **user id** を使用して新しい **UddiEntityPublisher** を生成します。

6.4. CRYPTEDXMLDOCAUTHENTICATION

CryptedXMLDocAuthentication 実装は、**XMLDocAuthentication** 実装と似ていますが、パスワードが暗号化されています。

```
juddi.auth = org.apache.juddi.auth.CryptedXMLDocAuthentication
juddi.usersfile = juddi-users-encrypted.xml
juddi.cryptor = org.apache.juddi.cryptor.DefaultCryptor
```

ここでは、ユーザー認証ファイルは、**juddi-users-encrypted.xml** で、ファイルの内容は、以下のようになります。

```
<?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 実装は **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、この認証実装の使用方法を確認することができます。**org.apache.juddi.cryptor.Cryptor** インターフェースを実装して、この実装クラスを **juddi.cryptor** プロパティで参照し、独自の暗号化アルゴリズムをプラグインすることができます。

認証フェーズは、**user id** とパスワードが XML ファイルの値と一致するか確認します。識別フェーズは、単に **user id** を使用して新しい **UddiEntityPublisher** を生成します。

6.5. LDAP 認証

LdapSimpleAuthenticator を使用して、LDAP のシンプルな認証機能でユーザー認証を行います。このクラスは、**principal** と **juddi publisher id** が同じであれば、**LDAP** プリンシパルを元にユーザーを認証することができます。

このクラスを使用するには、以下を **juddi.properties** ファイルに追加する必要があります。

```
juddi.auth=org.apache.juddi.auth.LdapSimpleAuthenticator
juddi.auth.url=ldap://localhost:389
```

juddi.auth.url プロパティは、**LdapSimpleAuthenticator** クラスに LDAP サーバーが常駐する場所を伝えます。

6.6. JBOSS 認証

最後に、サードパーティの認証ストアとインターフェースするというオプションもあります。**AS** の認証コンポーネントとリンクすることができます。

docs/examples/auth ディレクトリに **JBossAuthenticator** クラスがあります。このクラスでは、**JBoss** に **JUDDI** をデプロイして、ユーザー認証を行うセキュリティドメインを使用できるようになります。

このクラスを使用するには、以下のプロパティを **juddi.properties** ファイルに追加する必要があります。

```
juddi.auth=org.apache.juddi.auth.JBossAuthenticator  
juddi.securityDomain=java:/jaas/other
```

juddi.auth プロパティは、**JBossAuthenticator** クラスと jUDDI Registry の **Authenticator** フレームワークを連携します。**juddi.secruity.domain** は、**JBossAuthenticator** にアプリケーションサーバーのセキュリティドメインがどこにあるか伝えます。このドメインを使用して認証を行います。

JBoss は、**\$JBOSS_HOME/server/default/conf/login-config.xml** ファイルにアプリケーションポリシー別のセキュリティドメインを1つ作成します。これらのドメインは、サーバー JNDI ツリーとその名前をバインドします **java:/jaas/<application-policy-name>** (ルックアップで、存在しないアプリケーションポリシーを参照した場合、デフォルトでは **other** という名前が使用されるようになっています)。

第7章 UDDI アノテーション

7.1. はじめに

従来、管理者が GUI を使って手動で、サービス (**BusinessService**) とエンドポイント (**BindingTemplates**) を **UDDI Registry** に登録していました。残念ながら、こうすると、**Registry** 内のデータが静的、つまり時間が経つと陳腐化してしまいます。

jUDDI Registry をより動的にするには、デプロイするたびに各エンドポイントを登録すると理にかなうはずですが、**UDDI アノテーション**は、これを行うためだけに設計されています。アプリケーションサーバーにデプロイされる際にサービスを登録します。

アノテーションには **UDDIService** と **UDDIServiceBinding** の 2 種類があります。エンドポイントを登録するには、上記の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 Registry** から削除されますが、サービス情報は残ります (エンドポイントがなくてもサービスがあるため、後に確認するよう指示を出すことができるため、**Registry** にサービスレベルの情報を残すのは理にかなっています)。



注記

サービス情報を削除したい場合、手動で削除する必要があります。



注記

アノテーションは **juddi-client** ライブラリを使用します。つまり、これらのアノテーションを使用して、**UDDI バージョン 3** レジストリに登録できるということです。

7.2. UDDISERVICE アノテーション

UDDIService アノテーションを使用して、**Registry** 内にすでにあるビジネスに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してください。

表7.1 UDDIService 属性

属性	説明	必須
serviceName	サービス名。デフォルトでは、クラスが WebService アノテーションで指定された名前を使用します。	いいえ
description	人間が解読できるようなサービスの説明	はい
serviceKey	サービスの UDDI v3 キー	はい
businessKey	サービスを所有すべきビジネスの UDDI v3 キー (ビジネスは、登録時に Registry に存在している必要があります)。	はい

属性	説明	必須
lang	これは、名前や説明で利用される言語ロケール(省略するとデフォルトの "en" に設定されます)。	いいえ
categoryBag	<i>CategoryBag</i> の定義	いいえ

7.3. UDDISERVICEBINDING アノテーション

UDDIServiceBinding アノテーションを使用して、UDDI Registry に BindingTemplate を登録します (このアノテーションのみでの使用はできません。UDDIService アノテーション内で使用する必要があります)。

表7.2 UDDIServiceBinding 属性

属性	説明	必須
bindingKey	ServiceBinding の UDDI v3 キー	はい
description	人間が解読できるようなサービスの説明	はい
accessPointType	UDDI v3 の AccessPointType (省略するとデフォルトの wsdlDeployment になります)	いいえ
accessPoint	エンドポイント参照	はい
lang	これは、名前や説明で利用される言語ロケール(省略するとデフォルトの "en" に設定されます)。	いいえ
tModelKeys	コンマ区切りの tModelKeys キー参照	いいえ
categoryBag	<i>CategoryBag</i> の定義	いいえ

7.4. WEBSERVICE の例

サービスを定義するクラスにアノテーションを使用することができます。ここでは、WebService に追加されています (POJO に JAX-WS 'WebService' アノテーションがついています)。

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

import javax.xml.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** をデプロイすると、**juddi-client** コードは、UDDI アノテーションに対してこのクラスをスキャンし、登録プロセスを担当します。

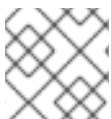
clerk のセクションでは、**org.apache.juddi.samples.HelloWorldImpl** サービスクラスに参照する必要があります。

```

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

```

このコードは、ボブが **default** という名前のノードの接続設定を使用しており、**sales publisher** (このパスワードは **sales**) を使用する予定であると記述しています。



注記

データソースの定義方法に似ています。

7.5. CATEGORYBAG 属性

CategoryBag 属性を使用して、**tModels** を参照します。以下に **CategoryBag** の例を示しています。

```

<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",
```

第8章 API を使った簡単な公開方法

jUDDI Registry を使用してサービスを公開する方法については、本章を参照してください。

8.1. UDDI データモデル

本章では、UDDI データモデルと、そのモデルに対する jUDDI 拡張について学習します。

アーチファクトの公開をする前に、データが UDDI モデルに適合する方法を把握する必要があります。本章では、プロセスの概要を把握しますが、詳細については UDDI 仕様を参照するようにしてください。

以下の図は、本モデルについてです (UDDI v3 仕様から抜粋)。

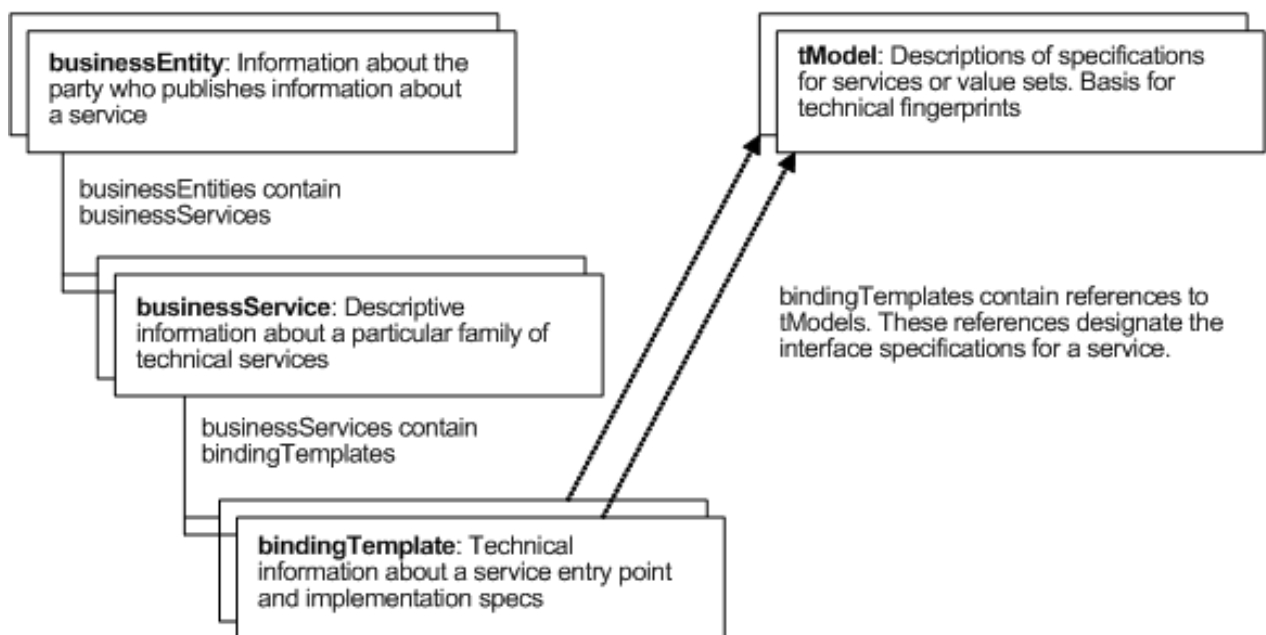


図8.1 UDDI コアデータ構造

図にあるように、データは階層別に整理されています。ビジネスエンティティがピラミッドの一番上に来っており、その中にビジネスサービスが含まれており、そのサービスの中にはバインディングテンプレートが含まれています。

TModels (またはテクニカルモデル) は、主なエンティティの1つに分類し、バインディングの技術詳細を説明 (プロトコルやトランスポート) し、キーパーティションを登録するなど、包括的な構造となっています。

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、サービスを公開する前にビジネスエンティティを構築し、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 (後ほど本章で例を使用して、このアプローチについて説明します)。

ビジネスエンティティは、サービスを提供する組織単位です。一般的に、ビジネスエンティティには説明と連絡先情報が含まれています。ビジネスエンティティの使用法は、各自の状況により異なります。小規模な企業であれば、エンティティ1つしか必要ないでしょう。より規模の大きい企業で複数の部署を持つ場合、部署ごとにエンティティを1つ持たせるかもしれません。

また、別のオプションとしては、包括的なエンティティを1つと、その配下に複数の部署を示す子エンティティを持たせることもできます。方法は、**Publisher** 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。

ビジネスサービスは、SOA 環境で中心的な存在であり、クライアントを消費する機能単位を表します。UDDI では、サービスには、名前、説明、カテゴリなど説明的な情報が主に含まれています。実際は、サービスに関する技術詳細は、バインディングテンプレートに含まれています。

バインディングテンプレートは、サービスの技術仕様が保存される場所です。仕様は、サービスのアクセスポイントやサービス WSDL の場所を提供するだけのシンプルなものから、WSDL の技術詳細の内訳などといった複雑なシナリオまで様々です (**tModels** と連携で使用)。

8.2. モデルへの JUDDI の追加

jUDDI Registry の実装は、仕様に記載されていない構造をデータモデルに提供します。パブリッシャーが主要なアイテムです。

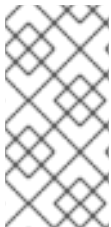
UDDI 仕様は、レジストリ内で公開されたエンティティのオーナーシップについて記述していますが、このオーナーシップを実装すべきであることについては記述がありません。個別の実装により変わってきます。

パブリッシャーは、jUDDI Registry 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。

Registry は認証トークンを取得できるようにしてパブリッシャーの認証を行ってからでないと、jUDDI にアセットの追加はできません。このトークンは後続の **publish** 呼び出しにアタッチされ、公開エンティティにオーナーシップを割り当てます。

カスタム API を用いて、jUDDI Registry にパブリッシャーを保存することができます。

パブリッシャーは、購入直後からアイデンティティ管理システムの実装をすぐにお使いいただけます。jUDDI により、独自のアイデンティティ管理システムに統合でき、パブリッシャーを完全に回避することも可能です。本書では、カスタマイズなしにすぐ利用できる、シンプルなパブリッシャーソリューションを利用します。



注記

オーナーシップは、本来 **authorizedName** フィールドから任意の登録エンティティに割り当てます。**operationalInfo** の構造を説明している仕様の章にて、このフィールドに関する説明があります。詳細は http://www.uddi.org/pubs/uddi_v3.htm#_Toc85908030 を参照してください。

8.3. UDDI および JUDDI API

UDDI データモデルについて理解されたと思いますので、次に **UDDI API** の構成や、jUDDI Registry がこの API を実装する方法について見ていきます。

UDDI API を使用するには、どのような構成になっているか理解する必要があります。複数の **sets** に分割されており、各 **set** は特定の機能エリアについて表しています。

- **Inquiry** は、エンティティの詳細に関する Registry へのクエリを処理します。details about entities.
- **Publication** は、エンティティの登録を処理します。
- **Security** は、認証処理を行う変更可能な仕様です。
- **Custody and Ownership Transfer** は、エンティティのオーナーシップの移行処理を行います。

- **Subscription** は、クライアントがサブスクリプションの形式を仕様してエンティティに関する情報をリトリブできるようにします。
- **Subscription Listener** は、サブスクリプションの結果を受け入れるクライアント API です。
- **Value Set (Validation and Caching)** は、キー付きの参照値を検証します (このセットは、jUDDI ではまだ利用できません)
- **Replication** は、レジストリノード間のデータを連携できるようにします (このセットは、jUDDI ではまだ利用できません)

最も頻繁に使用する **sets** は、**Inquiry**、**Publication**、**Security** です。**Registry** とやり取りを行う際に必要な標準機能を提供してくれます。

jUDDI サーバーは JAX-WS 準拠の Web サービスとして **sets** をそれぞれ実装します (セット内に定義されたメソッドは、該当の Web サービスにあるメソッドと同じです)。

jUDDI には、クライアントモジュールがあり、このモジュールはトランスポートクラスを使用してどのようにそれぞれを呼び出すのかを定義します (デフォルトのトランスポートは、JAX-WS を使用しますが、API への呼び出しには複数の方法があります)。

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、パブリッシャーやその他のメンテナンス機能 (多くが jUDDI のサブスクリプションモデル関連) を処理するメソッドが含まれています。この API セットは、jUDDI に特有で、UDDI 仕様には準拠していません。

8.4. JUDDI REGISTRY の初回利用

本章では、簡単な例をあげ、手順を説明していきます (この演習をはじめる前に jUDDI Registry を実行しておくようにしてください)。

8.4.1. 簡単なパブリッシャーの例

本章では、"simple-publish" の例についての手順を説明します。このサンプルでは、認証トークンをリトリブした後に、**Publisher**、**BusinessEntity**、**BusinessService** が jUDDI に登録されます。

このサンプルには、**SimplePublish** クラス 1 つのみで構成されています。以下がコンストラクターです。

```
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、デフォルトノードからトランスポートをリトリーブします。このサンプルでは、JAX-WS Web サービスで UDDI を呼び出しするように設計されたデフォルトクライアントのトランスポートクラスを単にリトリーブするだけです。

トランスポートがインスタンス化されると、このサンプルに必要な API セットを 3 つ取得してください。API セット 3 つは以下のとおりです。

- **Security API セット** (認証トークンを取得するため)
- **専用の jUDDI API セット** (パブリッシャーを保存するため)
- **Publication API セット** (jUDDI にエンティティを登録するため)

以下に **publish** メソッドの最初の数行の例を示しています。

```

// 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、単に **root** ユーザー用の認証トークンを取得するだけです。



注記

root ユーザーは自動的に、各 jUDDI インスタンスにインストールされ、jUDDI API 呼び出しの管理者として機能します。さらに、**root** ユーザーは jUDDI と合わせてインストールされた初期サービスすべてを所有するパブリッシャーとなります (初期サービスは、各 jUDDI ノードに Web サービスとして自動登録されるため、全 UDDI API セットから構成されています)。

root 認証を取得したので、パブリッシャーを追加していきます。

```

// 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 を使用して認証済みの名前 **my-publisher** でパブリッシャーを保存します。**root** ユーザーの認証トークンを使用している点に注意してください。

次に、この新規パブリッシャーの認証トークンを取得する必要があります。

```
// 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());
```

どちらの認証呼び出しにも、認証情報が設定されていません。これは、デフォルトの認証子を使っているので認証情報を渡す必要がないためです。

root 認証を取得したので、公開の準備が整いました。

```
// 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);
```

■

まとめると、この例では **BusinessEntity** を作成、保存してから、**BusinessService** を作成、保存しました。各エンティティに最小限のデータを追加しました (サービスには **BindingTemplates** は追加していません)。

実際には、特にサービス関連など、構造ごとにより多くの情報を追加するはずです。しかし、この例は便利なスタートポイントとなるでしょう。



注記

エンティティキーを使用する際に注意すべき重要なポイントがいくつかあります。仕様のバージョン 3 では、パブリッシャーは独自のキーを作成し、実装者にデフォルトのメソッドを持つように指示を出します。

Red Hat は、**save** 呼び出しで、各エンティティの **key** フィールドを空の状態にすることで、デフォルトの実装アプローチを選択しました。jUDDI のデフォルトキー生成は、ノードのパーティションを取り、**GUID** をそのパーティションに追加するだけです。

デフォルトのインストレーションでは、以下のようになります。

uddi:juddi.apache.org:<GUID>

(もちろん、上記をすべてカスタマイズすることも可能です)

2 つ目の重要なポイントは、**BusinessService** が保存されると、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リトリブ) が明示的に設定される点です。このように独立した呼び出しにサービスが保存される場合、この手順は必要となります。この手順を行わないと、jUDDI が親エンティティを見つけられないため、エラーが発生してしまいます。

第9章 サブスクリプション

9.1. はじめに

レジストリが複数ある環境では、サブスクリプションは便利です。企業内で、複数の UDDI Registry を実行するように決定したとします。例えば、独自の UDDI ノードに登録されたシステムへのみアクセスできるように、各部署ごとにレジストリを設定するように決めたとします。しかし、ある時点で、部署間でサービスの共有をする必要があると判明するかもしれません。そのような場合は、**subscription API** を使用してサービスを複数レジストリ間で登録して、親 UDDI のレジストリ情報が変更すると通知を送信することで、最新の情報に保ちます。

サブスクリプションには 2 種類あります。別の通知システムがそれぞれに存在します。

非同期

こちらは、サブスクリプションを保存して、設定したスケジュールで更新を受け取ることができます。

同期

こちらは、サブスクリプションを保存して、**get_Subscription** を呼び出すことで同期の返答を取得することができます。

非同期バージョンは、リスナーサービスを通知の送信先となるノードにインストールする必要があります。

9.2.2 ノードの例：SALES と MARKETING

この例では、**sales** と **marketing** のノードを設定します。これには、2 種のサービスに jUDDI Registry をデプロイして、システムが正しく設定されるように 2 つのプロシージャールに従う必要があります。

手順9.1 ノード 1 の設定: Sales

1. **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. **juddiv3.properties** を編集し、以下のプロパティ値を設定します。**sales** はサーバーの DNS 名となります。

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

3. サーバーを起動します。jUDDI が初めて起動される場合は、UDDI シードデータが読み込まれません。

```
bin/startup.sh
```

4. Web ブラウザーを開き、<http://sales:8080/juddiv3> のアドレスに移動します。ノードに関する情報を含むメッセージが表示されます。

手順9.2 ノード 2 の設定: Marketing

1. **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
```

2. **juddiv3.properties** を編集し、以下のプロパティ値を設定します。**marketing** はサーバーの DNS 名となります。

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

3. サーバーを起動します。JUDDI が初めて起動される場合は、UDDI シードデータが読み込まれます。

```
bin/startup.sh
```

4. Web ブラウザーを開き、<http://sales:8080/juddiv3> に移動します。また、問題なく実行された旨、またノード情報を含むメッセージが表示されます。



注記

営業部もマーケティング部も同社内に位置しているため、**root** パーティションは同じように保たれていますが、**Node Id** と **Name** で各ノードが別の部門に所属することがわかります。

次に **sales** 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** で置き換えます。

そして、以下のプロパティを設定し、**uddi-portlets.war/WEB-INF/classes/META-INF/uddi.xml** ファイルを編集します。

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

Web ブラウザーを起動して <http://sales:8080/pluto> のアドレスに移動します。

ユーザー名、パスワードともに **sales** でログインします。以下が画面に表示されます。

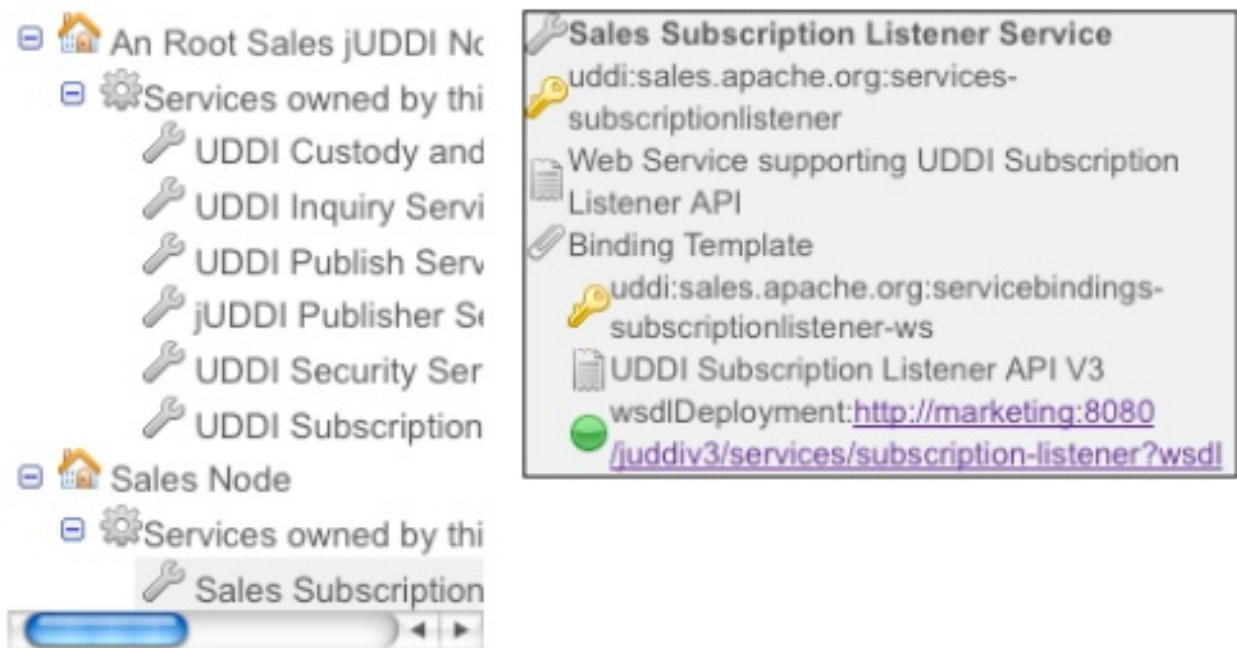


図9.1 Sales サービス

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` で置き換えます。

そして、以下のプロパティを設定し、`uddi-portlet.war/WEB-INF/classes/META-INF/uddi.xml` ファイルを編集します。

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

Web ブラウザーを起動して <http://marketing:8080/pluto> のアドレスに移動します。

ユーザー名、パスワードともに **sales** でログインします。以下が画面に表示されます。

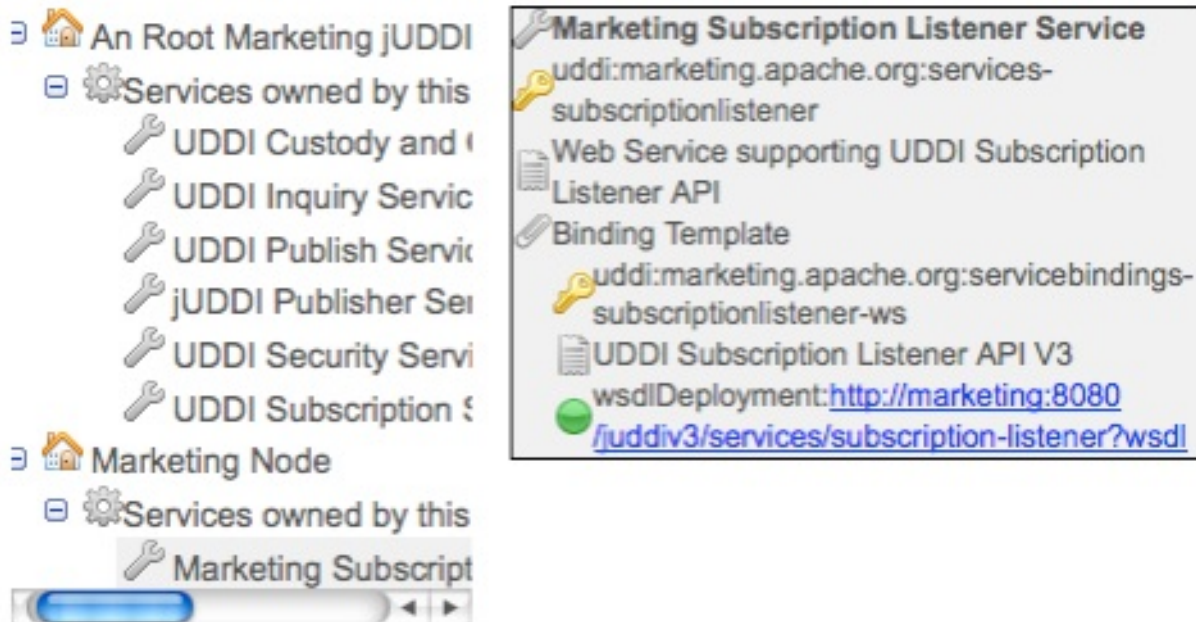


図9.2 Marketing サービス



注記

Root Marketing ノードではなく、Marketing ノードビジネスがsubscriptionlistener を所有します。Marketing Node Business は marketing publisher が管理します。

9.3. HELLOSALES サービスのデプロイ

次のサービス例は HelloSales と呼ばれます。HelloSales サービスは、**juddiv3-samples.war** アーカイブファイルにあり、アノテーションがついているため自動的に登録されます。

アーカイブをデプロイする前に、**sales** にプロパティ値を設定するように **juddiv3-samples.war/WEB-INF/classes/META-INF/uddi.xml** ファイルを編集します。以下のようになります。

```
<?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>

```

juddiv3-samples.war をビルド、デプロイすることで **sales** レジストリノードにデプロイします。

HelloWorld サービスは自動でデプロイされるはずです。

次に、Marketing UDDI ノードから Sales UDDI ノードの HelloWorld サービスにサブスクライブしてください。

9.4. ユーザーがサブスクリプションを作成ができるように許可

サブスクリプションを作成し保存するには、**sales** ノードと **marketing** ノードに対して有効な "publisher" ログイン権限がないといけません。また、**marketing publisher** が **marketing** ノードにレジストリオブジェクトを作成する場合、**marketing publisher** は、**sales keygenerator tModel** を所有する必要があります (この件についての詳細は、**Marketing** ノードと **sales** ノードの両方のルートシードデータで **marketing_*.xml** ファイルを確認してください。)

マーケティングレジストリの **marketing publisher** が以下の **tModels** を所有することを理解してください。

```

<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>
</save_tModel>

```

```

        </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>

```

marketing publisher を使用して sales レジストリの更新をサブスクライブする場合、このパブリッシャーに 2 つのクラスを提供する必要があります。方法は、**uddi-portlet.war** の **uddi.xml** ファイルに追加します。

```

<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>

```



```

    </xregister>
    -->
</clerks>

```

上記のコードでは、このパブリッシャに **MarketingCratchit** と **SalesCratchit** のクラークを 2 つ作成しました。こうすることで、パブリッシャーが所有する 2 つのシステム内でサブスクリプションをチェックします。

9.5. 同期通知

マーケティングポータルに **marketing publisher** でログインして、**UDDISubscription Portlet** を選択します。

両ノードが緑になると、(ツールバーにある) **new subscription** アイコンをクリックします。これは、同期サブスクリプションであるため、**Binding Key** と **Notification Interval** のみを残します。

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。

サブスクリプションキーが **marketing publisher** の **keyGenerator** ルールを使うようにしてください。 **sales-ws** UDDI ノードの下にオレンジのサブスクリプションアイコンが表示されるはずです。

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、対象期間を設定することができます。

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。ファイnderリクエストの例では、**HelloWorld** サービスに更新がないか **Sales** ノードを検索します。ロー XML レスポンスが **UDDISubscriptionNotification** ポートレットに掲示されます。

Subscription notifications:

```

<?xml version="1.0" encoding="UTF-8"
standalone="yes"?><subscriptionResultsList
xmlns:ns3="http://www.w3.org/2000/09/xmlsig#"
xmlns:ns2="urn:uddi-org:api_v3" xmlns="urn:uddi-
org:sub_v3"><coveragePeriod>
<startPoint>2008-01-01T00:00:00</startPoint>
<endPoint>2010-01-01T00:00:00</endPoint>
</coveragePeriod><subscription brief="false">
<subscriptionKey>uddi:marketing.apache.org:subscript
ion:key1</subscriptionKey><subscriptionFilter>
<ns2:find_service><ns2:findQualifiers>
<ns2:findQualifier>exactMatch</ns2:findQualifier>
</ns2:findQualifiers><ns2:name
xml:lang="en">HelloWorld</ns2:name>
</ns2:find_service></subscriptionFilter>
<maxEntities>1000</maxEntities>

```

図9.3 同期サブスクリプションリクエストへのロー XML レスポンス

レスポンスは **marketing** ノードで処理されます。このノードは、**HelloWorld** サブスクリプションの情報と **sales business** をインポートします。正常に同期されると、マーケティングノードのブラウザーポートレットに 3 つのビジネスが表示されます。

第10章 M BEAN のサポート

JXM コンソールで jUDI M Bean のクエリが可能です。これにより jUDDI スループットを観察することができます。以下に利用可能な MBean を示しています。

- `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 の各 UDDI オペレーションは、メソッドごとに以下の機能を提供します。

- 正常なクエリ
- 失敗したクエリ
- クエリ合計
- 処理時間
- API 別の合計／正常／失敗集計数

オペレーションは `resetCounts` の1つのみ。

付録A UDDI REGISTRY

A.1. はじめに



重要

SOA Platform のディストリビューションには、事前設定済みの jUDDI レジストリインストールが含まれています。カスタムクライアントでレジストリにアクセス可能にする API は、この jUDDI レジストリにより完全対応されています。独自に構築したカスタムのクライアントは、SOA Platform のサポート契約の対象範囲ではありません。jUDDI のサンプル、ドキュメント、API の全セットは、<http://juddi.apache.org/> から入手できます。

Universal Description, Discovery and Integration (UDDI) プロトコルは、正常な Web サービスに必要な主要ビルディングブロックの1つです。UDDI は、標準の相互運用可能なプラットフォームを作成して、企業やアプリケーションが Internet 経由で Web サービスを迅速、簡単、かつ動的に検索できるようにします。UDDI により、様々なコンテキストで様々な目的に対してオペレーションレジストリが管理できるようになっています。UDDI は、OASIS Standards Consortium 内の市場オペレーターや e-ビジネスリーダー、主なプラットフォームやソフトウェアプロバイダーが行う業界を超えた取り組みです。UDDI は、3 回改定され、最新版は 3.0.2 です。UDDI の追加情報は、<http://uddi.xml.org> から入手できます。

A.2. UDDI REGISTRY

UDDI Registry は、UDDI 仕様を実装します。UDDI は Web ベースの分散ディレクトリで、従来の Yellow page や White page といった電話帳とよく似ており、ビジネスがインターネット上で表示され、検索されるようになります。UDDI Registry とは、White page のようなビジネスディレクトリであり、技術仕様のライブラリでもあります。この Registry は、ビジネスやサービスの情報を格納し、詳細文書への参照を保持するように設計されています。

ビジネスは、サービスを UDDI レジストリに公開します。クライアントが、レジストリ内のサービスを検索して、サービスのバインディング情報を受け取ります。クライアントが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、相互運用性の面から SOAP ベースとなっています。UDDI v3 仕様では、9 つの API を定義しています。

1. **UDDI_Security_PortType**. セキュリティトークンを取得する API を定義します。有効なセキュリティトークンがあれば、パブリッシャーはレジストリを公開できます。全セッションを通して、セキュリティトークンを利用することができます。
2. **UDDI_Publication_PortType**. ビジネスやサービス情報を UDDI レジストリに公開する API を定義します。
3. **UDDI_Inquiry_PortType**. UDDI レジストリをクエリする API を定義します。通常、この API ではセキュリティトークンを必要としません。
4. **UDDI_CustodyTransfer_PortType**. この API を使用して、ビジネスの管理を UDDI ノードから別のノードに移行します。
5. **UDDI_Subscription_PortType**. 特定のビジネスサービスに更新を登録する API を定義します。
6. **UDDI_SubscriptionListener_PortType**. クライアントが UDDI ノードからサブスクリプション通知を受信するために実装する必要のある API を定義します。

7. **UDDI_Replication_PortType**。UDDI ノード間のレジストリデータを複製する API を定義します。
8. **UDDI_ValueSetValidation_PortType**。ノード別に外部プロバイダーが値の検証をできるようにします。また、Web サービスが **keyedReferences** または **keyedReferenceGroups** が有効かを評価します。
9. **UDDI_ValueSetCaching_PortType**。このような Web サービスから取得したキャッシュ値を使って、UDDI ノードは、パブリッシャー参照を検証する場合があります。

付録B JUDDI クライアントの使用

B.1. はじめに

jUDDI クライアントを使って Registry に接続します (使用するプロファイルにより、クライアントは、`./server/default/deployers/esb.deployer/lib/juddi-client-[version].jar`、`./server/all/deployers/esb.deployer/lib/juddi-client-[version].jar`、または `./server/production/deployers/esb.deployer/lib/juddi-client-[version].jar` にあります)。



注記

クライアントは、UDDI v3 API を使用するため、どの UDDI v3 準拠のレジストリでも接続できます。ただし、Red Hat は jUDDI v3 Registry のみに対応しています。

B.2. JAX-WS トランスポート

上記の `uddi.xml` ファイルの設定に基づき、クライアントは JAX-WS を用いて (リモートの) レジストリサーバーとやり取りします。つまり、クライアントは JAX-WS 準拠の Web サービススタック (CXF、Axis2、JBossWS) へアクセスする必要があります。

UDDI クライアントが WSDL ドキュメントを検索できるアドレスに、JAX-WS URL が参照しているか確認します。

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

B.3. RMI トランスポート

jUDDI をアプリケーションサーバーにデプロイする場合、UDDI サービスを *remote method invocation* (RMI) サービスとして登録できます。登録するには、サーバーの **deploy/jbossesb-registry.sar/esb.juddi.xml** ファイルを編集します。

これで、デプロイメント時に、RMI ベースの UDDI サービスがグローバルの JNDI 名前空間にバインドされています。

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

最後にクライアント側で、**uddi.properties** ファイルの JAXWS セクションをコメントアウトして、代わりに RMI Transport セクションを使用します。



注記

オプションで、**java.naming.*** プロパティも設定することができます。この例では、**JBoss Application Server** にデプロイした jUDDI v3 に接続する設定を指定しました。**jndi.properties** ファイルに **java.naming.*** プロパティを設定するか、システムパラメーターを設定することができます。

B.4. INVM トランスポート

In-Virtual Machine (InVM) トランスポートを使用することもできます。InVM はクライアントと同じ仮想マシンで jUDDI サーバーを実行することができます。**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** クラスにより起動されますが、コンテナ以外で実行している場合は、ご自身で **org.apache.juddi.Registry** サービスを開始、停止する必要があります。

手動で起動するには、他の呼び出しを行う前に、以下のメソッドを呼び出してください。

```
Registry.start()
```

Registry をシャットダウンするには、以下の呼び出しを使います。

```
Registry.stop()
```

これを行うことで、**Registry** が持っているリソースを開放できるようになります。

InVM トランスポートを使うには **uddi.properties** ファイルの該当セクションをアンコメントして、JAXWS と RMI Transport セクションをコメントアウトします。

B.5. 依存性

UDDI クライアントは、**uddi-ws-3.0.0.jar**、**commons-configuration-1.5.jar**、**commons-collection-3.2.1.jar**、**log4j-1.2.13.jar** に依存しています。

これに加え、お使いの設定によっては以下も必要な場合があります。

- JDK6 のライブラリ
- JAXWS クライアントライブラリ (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 (RMI トランスポートを使用している場合)

B.6. サンプルコード

jUDDI **uddi-client** モジュールにて UDDI クライアントの使用方法を説明している、サンプルコードがあります。

まず、**Registry** に呼び出しを行い、認証トークンを取得します (以下のコードは、このモジュールの単体テストから取得)。

```
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();
    }
}
```

この場合、パブリッシャー (今回は **root**) が **Registry** 内に既存のパブリッシャーで、正しいレスポンスを取得できるように適切な認証情報を提供しているか確認してください。

付録C © 2011

Apache License, Version 2.0 (以下、当該ライセンス) のライセンス。このライセンスに準拠する場合以外は本書の利用はできません。当該ライセンス文書は、<http://www.apache.org/licenses/LICENSE-2.0> から取得できます。準拠法で必要とされる、また書面で同意しない限り、当該ライセンス下で配信されたソフトウェアは、暗示または明示の保証や条件なしに現状渡しとします。当該ライセンスの許容事項、制限事項を管理する文言については当該ライセンスを参照してください。

付録D 改訂履歴

改訂 5.2.0-0.2.400 Rebuild with publican 4.0.0	2013-10-31	Rüdiger Landmann
改訂 5.2.0-0.2 日本語に翻訳	Wed Dec 19 2012	Credit
改訂 5.2.0-0.1 XML ソース 5.2.0-0 と翻訳ファイルを同期	Wed Dec 19 2012	Credit
改訂 5.2.0-0 初版	Wed Jul 6 2011	David Le Sage