Service Registry 3 2005Q4 開発者ガイド

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 819–4054
2005 年 10 月

051130@13215

目次

はじめに

Service Registry 3 2005Q4 開発者ガイドでは、Java™ API for XML Registries (JAXR) を使って Service Registry (Registry) に対してクエリーを実行したり、コンテンツを発行する方法について説明します。

対象読者

開発者ガイドは、レジストリの検索やレジストリへのコンテンツの発行を行う JAXR クライアントの開発を予定しているアプリケーションプログラマを対象にしています。このマニュアルは、次の事項に習熟している方を対象に記述されています。

- Java プログラミング言語
- ebXML レジストリおよびリポジトリの仕様の基本 Concept

お読みになる前に

次の仕様の基本 Concept に習熟していることが必要です。

- ebXML Registry Information Model Version 3.0
- ebXML Registry Services and Protocols Version 3.0

これらの仕様について公開されている中での最新バージョンを入手するには、OASIS の Web サイト (`http://www.oasis-open.org/`) にアクセスし、 ebXML RIM V3.0 と ebXML RS V3.0 へのリンクをクリックします。

コードを開発しながら、Service Registry ソフトウェアを備えた Web コンソールを使用して、コードが正しく機能することを確認できます。『Service Registry 3 2005Q4 ユーザーズガイド』を参照し、Web コンソールに慣れてください。

Service Registry は Java Web Services Developer Pack (`http://java.sun.com/webservices/jwsdp/`) の一部または Sun Java™ Enterprise System のコンポーネントとして提供されています。Sun Java™ Enterprise System は、ネットワーク環境やインターネット環境を通して分配されるエンタープライズアプリケーションをサポートするソフトウェアインフラストラクチャです。Java Enterprise System のコンポーネントとして Service Registry を購入した場合は、`http://docs.sun.com/coll/1286.1` のシステムドキュメントをよく読んでください。

内容の紹介

このマニュアルの内容は次のとおりです。

第 1 章では、JAXR の概要について説明します。

第 2 章では、Service Registry に対してクエリーや更新を実行できる JAXR クライアントを実装するための最初の手順について説明します。

第 3 章では、レジストリに対してクエリーを実行するために JAXR が提供しているインタフェースおよびメソッドについて説明します。

第 4 章では、レジストリにオブジェクトを発行する方法について説明します。

第 5 章では、オブジェクトの削除や状態変更など、レジストリ内のオブジェクトへの操作方法を説明します。

第 6 章では、UDDI クエリーを使用してレジストリを検索できるようにする Java クライアントプログラムの開発方法について説明します。

付録 A では、オブジェクトを一意の識別子で検索する場合に使用できる定数を一覧表示します。

Service Registry マニュアルセット

Service Registry のマニュアルセットは `http://docs.sun.com/app/docs/coll/1314.1` で参照できます。Service Registry の詳細については、次の表に示すマニュアルを参照してください。

表 **P–1** Service Registry マニュアル

マニュアルタイトル	内容
『Service Registry 3 2005Q4 リリースノート』	既知の問題など、Service Registry に関する最新の情報が記載されています。
『Service Registry 3 2005Q4 管理ガイド』	インストール後に Service Registry を設定する方法と、レジストリに付属している管理ツールの使用方法が記載されています。その他の管理タスクを実行する方法についても説明されています。
『Service Registry 3 2005Q4 ユーザーズガイド』	Service Registry Web コンソールを使用して Service Registry を検索し、データを発行する方法が記載されています。
『Service Registry 3 2005Q4 開発者ガイド』	Java API for XML Registries (JAXR) を使用して Service Registry を検索し、データを発行する方法が記載されています。

関連マニュアル

Service Registry をインストールすると、Sun Java System Application Server に配備されます。Application Server の管理については、『Sun Java System Application Server Enterprise Edition 8.1 2005Q2 Administration Guide』を参照してください。

Java ES マニュアルセットでは、配備計画とシステムインストールについて説明しています。システムドキュメントの URL は `http://docs.sun.com/coll/1286.1` です。Java ES の概要については、次の表に示すマニュアルをこの順序で参照してください。

表 **P–2** Java Enterprise System のマニュアル

マニュアルタイトル	内容
『Sun Java Enterprise System 2005Q4 Release Notes』	既知の問題など、Java ES に関する最新の情報が記載されています。また、各コンポーネント独自のリリースノートも含まれています。
『Sun Java Enterprise System 2005Q4 Documentation Roadmap』	Java ES のシステムおよび個々のコンポーネントに関連するすべてのマニュアルの概要が記載されています。
『Sun Java Enterprise System 2005Q4 Technical Overview』	Java ES の技術および Concept の基礎が紹介されています。コンポーネント、アーキテクチャー、プロセス、および機能について説明しています。

表 **P–2** Java Enterprise System のマニュアル　(続き)

マニュアルタイトル	内容
『Sun Java Enterprise System 2005Q4 Deployment Planning Guide』	Java ES に基づく企業向け配備ソリューションの計画と設計の概要が記載されています。配備の計画と設計に関する基本的な Concept と原則、ソリューションのライフサイクル、および Java ES に基づくソリューションを計画する際に使用する、高度な例と方針が示されています。
『Sun Java Enterprise System 2005Q4 Installation Planning Guide』	Java ES 配備のハードウェア、オペレーティングシステム、およびネットワークに関する実装仕様の開発に役立ちます。コンポーネントの依存性など、インストールと設定の計画で考慮する必要のある問題について説明しています。
『Sun Java Enterprise System 2005Q4 Installation Guide for UNIX』	Solaris オペレーティングシステムまたは Linux オペレーティングシステムに Java ES をインストールする手順が記載されています。インストール後にコンポーネントを設定し、正しく動作することを確認する方法も示されています。
『Sun Java Enterprise System 2005Q4 Installation Reference』	構成パラメータに関する追加情報、構成計画で使用するワークシート、およびデフォルトディレクトリやポート番号などの参考資料が記載されています。
『Sun Java Enterprise System 2005Q1 Deployment Example Series: Evaluation Scenario』	1 つのシステムに Java ES をインストールし、中核の共有ネットワークサービスのセットを確立し、確立したサービスにアクセスできるユーザーアカウントを設定する方法について説明しています。
『Sun Java Enterprise System 2005Q4 Upgrade Guide』	Solaris オペレーティングシステムまたは Linux オペレーティング環境で Java ES をアップグレードする手順が記載されています。
『Sun Java Enterprise System Glossary』	Java ES マニュアルで使用される用語を定義します。

Java ES とそのコンポーネントに関するすべてのマニュアルの URL は、`http://docs.sun.com/prod/entsys.05q4` です。

デフォルトのパスとファイル名

次の表は、このマニュアルで使用されているデフォルトのパス名とファイル名について説明したものです。

表 **P–3** デフォルトのパスとファイル名

プレースホルダ	説明	デフォルト値
ServiceRegistry-base	Service Registry のベースインストールディレクトリを表します。	Solaris システム: `/opt/SUNWsoar` Linux システム: `/opt/sun/SUNWsoar`
RegistryDomain-base	Service Registry の Application Server ドメインおよび Service Registry データベースが置かれているディレクトリを表します。	Solaris システム: `/var/opt/SUNWsoar` Linux システム: `/var/opt/sun/SUNWsoar`

表記上の規則

このマニュアルでは、次のような字体や記号を特別な意味を持つものとして使用します。

表 **P–4** 表記上の規則

字体または記号	意味	例
`AaBbCc123`	コマンド名、ファイル名、ディレクトリ名、画面上のコンピュータ出力、コード例を示します。	`.login` ファイルを編集します。 `ls -a` を使用してすべてのファイルを表示します。 `machine_name% you have mail.`
`AaBbCc123`	ユーザーが入力する文字を、画面上のコンピュータ出力と区別して示します。	`machine_name%` **`su`** `Password:`
aabbcc123	変数を示します。実際に使用する特定の名前または値で置き換えます。	ファイルを削除するには、`rm` *filename* と入力します。
『 』	参照する書名を示します。	『コードマネージャ・ユーザーズガイド』を参照してください。
「 」	参照する章、節、ボタンやメニュー名、強調する単語を示します。	第 5 章「衝突の回避」を参照してください。 この操作ができるのは、「スーパーユーザー」だけです。

表 **P–4** 表記上の規則　　(続き)

字体または記号	意味	例
\	枠で囲まれたコード例で、テキストがページ行幅を超える場合に、継続を示します。	sun% **grep '^#define ** **XV_VERSION_STRING'**

コード例は次のように表示されます。

- C シェル

  machine_name% **command y|n** [*filename*]

- C シェルのスーパーユーザー

  machine_name# **command y|n** [*filename*]

- Bourne シェルおよび Korn シェル

  $ **command y|n** [*filename*]

- Bourne シェルおよび Korn シェルのスーパーユーザー

  # **command y|n** [*filename*]

[ ] は省略可能な項目を示します。上記の例は、*filename* は省略してもよいことを示しています。

| は区切り文字 (セパレータ) です。この文字で分割されている引数のうち 1 つだけを指定します。

キーボードのキー名は英文で、頭文字を大文字で示します (例: Shift キーを押します)。ただし、キーボードによっては Enter キーが Return キーの動作をします。

ダッシュ (-) は 2 つのキーを同時に押すことを示します。たとえば、Ctrl-D は Control キーを押したまま D キーを押すことを意味します。

コマンド例のシェルプロンプト

次の表に、デフォルトのシステムプロンプトとスーパーユーザープロンプトを示します。

表 **P–5** シェルプロンプト

シェル	プロンプト
UNIX システムおよび Linux システムの C シェル	machine_name%

表 **P–5** シェルプロンプト　(続き)

シェル	プロンプト
UNIX システムおよび Linux システムの C シェルスーパーユーザー	`machine_name#`
UNIX システムおよび Linux システムの Bourne シェルおよび Korn シェル	`$`
UNIX システムおよび Linux システムの Bourne シェルおよび Korn シェルのスーパーユーザー	`#`
Microsoft Windows のコマンド行	`C:\`

記号の表記規則

次の表では、このマニュアルで使用されている記号について説明します。

表 **P–6** 記号の表記規則

記号	説明	例	意味
`[ ]`	省略可能な引数およびコマンドオプションが含まれます。	`ls [-l]`	`-l` オプションの指定は必須ではありません。
`{ | }`	必須のコマンドオプションの選択肢のセットが含まれます。	`-d {y|n}`	`-d` オプションには、`y` 引数または `n` 引数が必要です。
`${ }`	変数の参照を示します。	`${com.sun.javaRoot}`	`com.sun.javaRoot` 変数の値を参照します。
-	同時に押す複数のキーストロークを示します。	Control-A	コントロールキーを押しながら A キーを押します。
+	連続する複数のキーストロークを示します。	Ctrl + A + N	コントロールキーを押して離した後、以降のキーを押します。
→	グラフィカルユーザーインタフェースのメニューの選択を表示します。	「ファイル」→「新規」→「テンプレート」	「ファイル」メニューから「新規」を選択します。「新規」サブメニューから「テンプレート」を選択します。

マニュアル、サポート、およびトレーニング

Sun のサービス	**URL**	内容
マニュアル	`http://jp.sun.com/documentation/`	PDF 文書および HTML 文書をダウンロードできます。
サポートおよびトレーニング	`http://jp.sun.com/supportraining/`	技術サポート、パッチのダウンロード、および Sun のトレーニングコース情報を提供します。

第 1 章

JAXR の概要

この章では、Java™ API for XML Registries (JAXR) の概要について簡単に説明します。次のトピックについて説明します。

レジストリとリポジトリについて

XML「レジストリ」は、Web サービスを構築、配備、および検出するための基盤です。これは中立的なサードパーティーであり、疎結合された動的な B2B (Business-to-Business) 対話を支援します。レジストリは共有リソースとして複数の組織で使用でき、通常は Web ベースのサービスとして提供されます。

現在、さまざまな XML レジストリ仕様が存在しています。これらの仕様には次のものがあります。

- ebXML Registry and Repository 標準。この標準のスポンサは、OASIS (Organization for the Advancement of Structured Information Standards: 構造化情報標準促進協会) と U.N./CEFACT (United Nations Centre for the Facilitation of Procedures and Practices in Administration, Commerce and Transport: 行政、商業、運輸に関する実務及び手続簡易化センター。国連機関) です。*ebXML* は Electronic Business using eXtensible Markup Language (XML による電子商取引) の略です。
- UDDI (Universal Description, Discovery, and Integration) プロトコル。このプロトコルは、あるベンダーコンソーシアムによって開発されました。

「レジストリプロバイダ」とは、XML レジストリの特定の仕様に準拠したレジストリ実装のことです。

UDDI がビジネスと提供するサービスに関する情報を格納するのに対し、ebXML レジストリはより広い範囲をカバーします。それは、レジストリであると同時に「リポジトリ」でもあります。リポジトリには、任意のコンテンツとそのコンテンツに関する情報が格納されます。つまり、リポジトリにはデータとメタデータが格納されます。ebXML レジストリ標準は、相互運用可能な Web サービス用の ECM (Enterprise Content Management) API を規定しています。

Web における ebXML Registry and Repository は、エンタープライズアプリケーションにおけるリレーショナルデータベースに相当します。これは、Web サービスや Web アプリケーションがコンテンツとメタデータを格納および共有する手段を提供します。

ebXML レジストリは、「レジストリ連携」(関連するレジストリから成るグループ) に含めることができます。たとえば、欧州のある国の厚生省があるレジストリを運営し、さらにそのレジストリを、ほかの欧州諸国の厚生省のレジストリが含まれる連携に含める、といったことが可能です。

Service Registry は、ebXML Registry and Repository 仕様のバージョン 3.0 を実装したものです。この仕様は次の 2 つに分かれています。

- 『ebXML Registry Services and Protocols Specification』(「ebXML RS」) は、ebXML レジストリのサービスとプロトコルを規定しています。
- 『ebXML Registry Information Model Specification』(「ebXML RIM」) は、ebXML レジストリ内に格納できるメタデータとコンテンツの種類を規定しています。

これらの仕様の最新の公開版を入手するには、OASIS の Web サイト (`http://www.oasis-open.org/`) にアクセスし、ebXML RIM V3.0 と ebXML RS V3.0 へのリンクをクリックします。

JAXR について

JAXR を使用すれば、Java ソフトウェアプログラマは、1 つの簡便な抽象 API を通じてさまざまな XML レジストリにアクセスできます。JAXR の統一化された情報モデルは、XML レジストリ内のコンテンツとメタデータを記述します。

JAXR を使用すれば、開発者は、さまざまなターゲットレジストリに移植可能なレジストリクライアントプログラムを記述できます。さらに、JAXR は、背後のレジストリに備わる機能を超えた、付加価値の高い機能をも提供します。

JAXR 仕様の現行バージョンには、JAXR 情報モデルと ebXML レジストリ仕様間の詳細なバインディング情報が含まれています。JAXR 仕様の最新バージョンは、`http://java.sun.com/xml/downloads/jaxr.html` から入手できます。JAXR の API ドキュメントは、Java 2 Platform, Enterprise Edition (J2EE プラットフォーム) の API ドキュメント (`http://java.sun.com/j2ee/1.4/docs/api/index.html`) に含まれています。

Service Registry には、レベル 1 機能プロバイダを実装した JAXR プロバイダが含まれており、ebXML レジストリへのフルアクセスが可能となっています。ebXML 仕様と JAXR 仕様間の整合性は完全ではありません。これは、ebXML 仕様のほうが JAXR 仕様よりも改良が進んでいるためです。このため、Service Registry の JAXR プロバイダには、ebXML 仕様を実装する、追加の実装固有メソッドがいくつか含まれています。これらの追加メソッドは、JAXR 仕様の次のバージョンに組み込まれる予定です。

JAXR アーキテクチャー

JAXR の高レベルのアーキテクチャーは、次の要素から構成されます。

- *JAXR* クライアント: これは、JAXR API を使って JAXR プロバイダ経由でレジストリにアクセスするクライアントプログラムです。
- *JAXR* プロバイダ: これは JAXR API の実装であり、特定のレジストリプロバイダまたは共通の仕様に基づく一連のレジストリプロバイダへのアクセス機能を提供します。このマニュアルでは、JAXR プロバイダの実装方法については説明しません。

JAXR プロバイダが実装する主要パッケージは、次の 2 つです。

- `javax.xml.registry`。このパッケージは、API インタフェース、およびレジストリアクセスインタフェースを定義する クラス群から成ります。
- `javax.xml.registry.infomodel`。このパッケージは、JAXR の情報モデルを定義するインタフェース群から成ります。これらのインタフェースは、レジストリに格納されるオブジェクトのタイプと、それらの関係を定義します。このパッケージの基本インタフェースは `RegistryObject` です。

`javax.xml.registry` パッケージのもっとも基本的なインタフェースを次に示します。

- `Connection`。`Connection` インタフェースは、レジストリプロバイダとのクライアントセッションを表現したものです。クライアントは、レジストリを使用する前に、JAXR プロバイダとの接続を作成する必要があります。
- `RegistryService`。クライアントは、接続から `RegistryService` オブジェクトを取得します。この `RegistryService` オブジェクトにより、クライアントは、レジストリへのアクセスに使用するインタフェースを順に取得できます。

同じく `javax.xml.registry` パッケージ内に含まれるインタフェースのうち、主なものを次に示します。

- `QueryManager` と `BusinessQueryManager`。クライアントはこれらのインタフェースを使用して、`javax.xml.registry.infomodel` のインタフェース群に基づいてレジストリ内で情報を検索できます。省略可能なインタフェース `DeclarativeQueryManager` は、クライアントが SQL 構文を使ってクエリーを実行できるようにします。Service Registry の ebXML プロバイダは、

DeclarativeQueryManager を実装しています。

- LifeCycleManager と BusinessLifeCycleManager 。クライアントはこれらのインタフェースを使用して、情報の保存 (更新) または削除によってレジストリ内の情報を変更できます。

詳細、およびこれらのインタフェース間の関係を示す図については、javax.xml.registry パッケージの API ドキュメント (http://java.sun.com/j2ee/1.4/docs/api/javax/xml/registry/package-summary.html) を参照してください。

エラーが発生すると、AXR API のメソッドは JAXRException またはそのサブクラスの 1 つをスローします。

JAXR API の多くのメソッドでは、Collection オブジェクトが引数または戻り値として使用されています。Collection オブジェクトを使用すると、いくつかのレジストリオブジェクトに対する操作を一度に実行できます。

図 1–1 は、JAXR のアーキテクチャーを図示したものです。Service Registry の場合、JAXR クライアントは、JAXR API の機能レベル 0 と機能レベル 1 のインタフェースを使って JAXR プロバイダ (ebXML プロバイダ) にアクセスします。すると、JAXR プロバイダは、ebXML レジストリの一種である Service Registry にアクセスします。

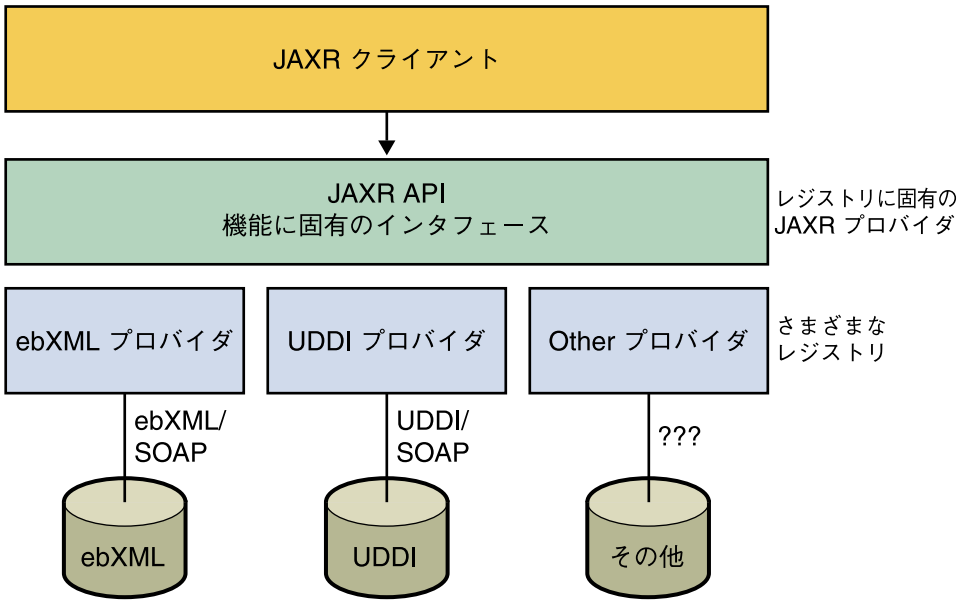

図 1–1 JAXR アーキテクチャー

サンプルについて

このマニュアルには、JAXR の機能を紹介するたくさんのサンプルクライアントプログラムを掲載しています。これらのサンプルを入手するには、次の URL にアクセスしてください。`http://www.sun.com/products/soa/registry/#faq/`。サンプルを収録した圧縮ファイルは、「リソース」セクションから入手できます。

この圧縮ファイルを、ファイルシステム上の適切な場所にダウンロードします。ファイルの解凍すると、<*INSTALL*>`/registry/samples` ディレクトリにサンプルのソースコードが格納されています。ここで、<*INSTALL*> はサンプルを解凍したディレクトリです。

サンプルごとまたはサンプルグループごとに `build.xml` ファイルが 1 つずつ用意されています。これらのファイルを使えば、`asant` ツール経由で各サンプルをコンパイルおよび実行できます。各 `build.xml` ファイルには、1 つの `compile` ターゲットと、1 つまたは複数のサンプルを実行する 1 つ以上のターゲットが含まれています。実行用ターゲットの中には、コマンド行引数を取るものもあります。

`asant` コマンドは Sun Java System Application Server の `bin` ディレクトリにあります。このディレクトリは通常、Solaris™ オペレーティングシステムでは `/opt/SUNWappserver/appserver/bin/`、Linux システム上では `/opt/sun/appserver/bin` になります。

サンプルを実行する前に、<*INSTALL*>`/registry/samples/common` ディレクトリにある 2 つのファイルを編集する必要があります。ファイル `build.properties` は、プログラムを実行する `asant` ターゲットが使用します。ファイル `JAXRExamples.properties` は、プログラム自身が使用するリソースバンドルです。

さらに、<*INSTALL*>`/registry/samples/common` ディレクトリにある `targets.xml` ファイルには、サンプルをコンパイルおよび実行するためのクラスパスが定義されています。また、各サンプルのコンパイル時に作成される `build` ディレクトリを削除する `clean` ターゲットも含まれています。このファイルを編集する必要はありません。

▼ `build.properties` ファイルを編集するには

手順 **1.** プロパティー **`container.home`** を、**Service Registry** が配備されているコンテナの場所に設定します。

これは、Sun Java System Application Server Enterprise Edition 8.1 がインストールされている場所です。この場所のデフォルトは、 Solaris システムでは `/opt/SUNWappserver`、Linux システムでは `/opt/sun/appserver` です。

2. プロパティー **registry.home** を、**Service Registry** のインストールディレクトリに設定します。

   このディレクトリは、Solaris システムでは /opt/SUNWsoar、Linux システムでは /opt/sun/SUNWsoar です。

3. プロパティー **registry.domain.home** を、**Service Registry** のドメインのインストールディレクトリに設定します。

   このディレクトリは、Solaris システムでは /var/opt/SUNWsoar/domains/registry、Linux システムでは /var/opt/sun/SUNWsoar/domains/registry です。

4. ファイアウォールの内側にいる場合、プロパティー **proxyHost** を、インターネットへのアクセス時に経由するシステムの名前に設定します。

   適切な値がわからない場合は、それらの情報を持つシステム管理者などに問い合わせてください。proxyPort の値は、一般的な値である 8080 に設定されていますが、必要があれば変更します。

▼ JAXRExamples.properties ファイルを編集するには

手順

1. プロパティー **query.url** と **publish.url** を編集して **Service Registry** の **URL** を指定します。

   このファイルでは、ホストとポートについてのデフォルト設定は localhost:6060 です。Service Registry がリモートサーバー上またはデフォルト以外のポートにインストールされている場合は、この設定を別のホストまたはポートに変更してください。

2. セキュリティー関連のプロパティーを編集し、**Service Registry** への発行に必要なプロパティーを指定します。これらの編集は、**Web** コンソールのユーザー登録ウィザードを使用した後で行なってください。詳細については、**21** ページの「**Service Registry** へのアクセス」を参照してください。

3. サンプルを試すときに、ファイルのほかの部分のデータを自由に変更してかまいません。

   クライアントサンプルを実行する asant ターゲットは、常にこのファイルの最新版を使用します。

第 2 章

JAXR クライアントの設定

この章では、Service Registry に対してクエリーや更新を実行できる JAXR クライアントを実装するための最初の手順について説明します。JAXR クライアントとは、JAXR API を使用してレジストリにアクセスするクライアントプログラムです。ここでは、次の項目について説明します。

Service Registry の起動

Service Registry を起動するには、Service Registry がインストールされているコンテナである Sun Java System Application Server を起動します。

Service Registry がまだ稼働していない場合は、Service Registry を起動します。手順については、『Service Registry 3 2005Q4 管理ガイド』の「レジストリ用 Application Server ドメインを停止および再起動する方法」を参照してください。

Service Registry へのアクセス

JAXR クライアントのすべてのユーザーは、Service Registry で、アクセス制御ポリシーで制限されていないオブジェクトに対してクエリーを実行できます。ただし、次のアクションを実行する場合、ユーザーは Service Registry からアクセス権を取得する必要があります。

- Service Registry へのデータの追加
- レジストリデータの更新
- 制限されているオブジェクトについてクエリーの実行

Service Registry では、ユーザーアクセスにクライアント証明書認証が使用されます。

Service Registry にデータを送信できるユーザーを作成するには、Web コンソールのユーザー登録ウィザードを使用します。Web コンソールは Service Registry ソフトウェアの一部です。ユーザーに Service Registry の使用を承認する証明書に加えて、ユーザー名とパスワードをこのウィザードで取得する方法については、『Service Registry 3 2005Q4 ユーザーズガイド』の「ユーザーアカウントの作成」を参照してください。認証局から取得した既存の証明書を使用することもできます。

Service Registry にデータを発行する前に、ダウンロードした `.p12` ファイルから JKS キーストアファイルに証明書を移動する必要があります。キーストアファイルの場所は、必ずホームディレクトリ内の次の位置にしてください。`$HOME/soar/3.0/jaxr-ebxml/security/keystore.jks` 。サンプルプログラムには、このタスクを実行する `asant` ターゲットが含まれています。詳細については、22 ページの「証明書のキーストアを作成するには」を参照してください。

ユーザーアカウントとキーストアの作成後、`JAXRExamples.properties` ファイルを編集します。詳細については、23 ページの「`JAXRExamples.properties` ファイルのセキュリティー設定を編集する」を参照してください。

▼ 証明書のキーストアを作成するには

証明書の JKS キーストアを作成するには、`asant` ターゲットの `move-keystore` を使用します。これは、`<INSTALL>/registry/samples/common/targets.xml` ファイルで定義されています。このターゲットファイルは、サンプルディレクトリ内のすべての `build.xml` ファイルで使用されます。

`move-keystore` ターゲットでは、*<INSTALL>*`/registry/samples/common/build.properties` ファイルで定義されている `keystoreFile` という名前のプロパティーが使用されます。このプロパティーの定義を変更しないでください。`move-keystore` ターゲットは、`ebxmlrr` のキーストアパスワードも指定します。この値は、`JAXRExamples.properties` ファイルの `security.storepass` プロパティーで使用されます。

手順

1. **`common` 以外のサンプルディレクトリのいずれかに移動します。**
   たとえば、次のコマンドを使用したとします。

   ```
   cd registry/samples/query-id
   ```

2. 次のコマンドを実行します。すべて **1** 行に入力してください。

   ```
   asant move-keystore -Dp12path=path_of_p12_file -Dalias=your_user_name
   -Dpassword=your_password
   ```

次のようなコマンドを使用します。

```
asant move-keystore -Dp12path=/home/myname/testuser.p12 -Dalias=testuser
-Dpassword=testuser
```

このターゲットの構文ヒントを表示するには、コマンド `asant -projecthelp` を使用します。

▼ JAXRExamples.properties ファイルのセキュリティー設定を編集する

手順

1. **<*INSTALL*>/registry/samples/common/JAXRExamples.properties** ファイルをテキストエディタで開きます。
2. 次の行を見つけます。

```
security.keystorePath=<home_dir>/soar/3.0/jaxr-ebxml/security/keystore.jks
security.storepass=ebxmlrr
security.alias=
security.keypass=
```

3. **security.keystorePath** プロパティーを指定するには、**<home_dir>** をホームディレクトリの絶対パス **(/home/myname** など**)** で置き換えます。
4. **security.alias** プロパティーの値には、ユーザー登録ウィザードで入力したユーザー名を指定します。
5. **security.keypass** プロパティーの値には、ユーザー登録ウィザードで入力したパスワードを指定します。
6. ファイルを保存して、閉じます。

Service Registry への接続の確立

JAXR クライアントで完了しなければならない最初のタスクは、レジストリへの接続の確立です。接続の確立には、次のタスクが必要です。

- 23 ページの「接続ファクトリの作成または検索」
- 24 ページの「接続の作成」

接続ファクトリの作成または検索

クライアントは、接続ファクトリから接続を作成します。ここでは、接続ファクトリを取得する 2 つの方法について説明します。

- スタンドアロンのクライアントプログラムで使用するために ConnectionFactory インスタンスを取得
- 配備された Java™ 2 Platform, Enterprise Edition (J2EE) アプリケーションで使用するために接続ファクトリを検索

ConnectionFactory インスタンスの取得

スタンドアロンのクライアントプログラムで JAXR を使用するには、抽象クラス ConnectionFactory のインスタンスを取得する必要があります。そのためには、JAXR プロバイダの JAXRUtility クラスの getConnectionFactory メソッドを呼び出します。

```
import org.freebxml.omar.client.xml.registry.util.JAXRUtility;
...
ConnectionFactory factory = JAXRUtility.getConnectionFactory();
```

接続ファクトリの検索

J2EE アプリケーションで使用するための 1 つ以上の事前設定済み接続ファクトリが、JAXR プロバイダによって提供される場合があります。これらのファクトリを取得するために、クライアントは JNDI (Java Naming and Directory Interface) API を使用してファクトリを検索します。

配備された J2EE アプリケーションで JAXR を使用するには、JAXR リソースアダプタ (RA) によって提供される接続ファクトリを使用します。そのような接続ファクトリにアクセスするには、JNDI 名が eis/MYSOAR であるコネクタリソースを使用する必要があります。このリソースは Service Registry の設定プロセスで作成されます。J2EE コンポーネントでこの接続ファクトリを検索するには、次のようなコードを使用します。

```
import javax.xml.registry.*;
import javax.naming.*;
...
    Context context = new InitialContext();
    ConnectionFactory connFactory = (ConnectionFactory)
         context.lookup("java:comp/env/eis/MYSOAR");
```

接続の作成

接続を作成するために、クライアントはまず、アクセスされる 1 つ以上のレジストリの URL を指定する一連のプロパティーを作成します。Service Registry がローカルシステムに配備されている場合、次のコードは Service Registry のクエリーサービスと発行サービスの URL を指定します。この文字列中には改行を入れないでください。

```
Properties props = new Properties();
props.setProperty("javax.xml.registry.queryManagerURL",
    "http://localhost:6060/soar/registry/soap");
```

```
props.setProperty("javax.xml.registry.lifeCycleManagerURL",
    "http://localhost:6060/soar/registry/soap");
```

次に、クライアントは、23 ページの「接続ファクトリの作成または検索」で説明されているように接続ファクトリを取得し、そのプロパティーを設定して、接続を作成します。次のコードは、これらのタスクを実行します。

```
ConnectionFactory factory =
     JAXRUtility.getConnectionFactory();
factory.setProperties(props);
Connection connection = factory.createConnection();
```

サンプルプログラムの `makeConnection` メソッドは、JAXR 接続の作成手順を示しています。

24 ページの「接続の作成」では、接続に対して設定可能な 2 つのプロパティーを表示して説明しています。これらのプロパティーは JAXR 仕様で定義されています。

表 2–1 標準 JAXR 接続プロパティー

プロパティー名と説明	データタイプ	デフォルト値
`javax.xml.registry.queryManagerURL` ターゲットレジストリプロバイダ内のクエリーマネージャーサービスの URL を指定します。	文字列	なし
`javax.xml.registry.lifeCycleManagerURL` ターゲットレジストリプロバイダ内のライフサイクルマネージャーサービスの URL を指定します (レジストリ更新用)。	文字列	`queryManagerURL` に指定された値と同じ

RegistryService オブジェクトの取得および使用

接続の作成後、クライアントはその接続を使用して `RegistryService` オブジェクトを取得してから、自らが使用する 1 つまたは複数のインタフェースを取得します。

```
RegistryService rs = connection.getRegistryService();
DeclarativeQueryManager bqm = rs.getDeclarativeQueryManager();
LifeCycleManager blcm =
     rs.getLifeCycleManager();
```

通常、クライアントは `RegistryService` オブジェクトから 2 つのオブジェクトを取得します。1 つはクエリーマネージャー、もう 1 つはライフサイクルマネージャーです。クエリーマネージャーは `DeclarativeQueryManager` オブジェクトまたは `BusinessQueryManager` オブジェクトです。ライフサイクルマネージャーは

`LifeCycleManager` オブジェクトまたは `BusinessLifeCycleManager` オブジェクトです。クライアントが単純なクエリーのみに Service Registry を使用しているのであれば、クライアントはクエリーマネージャーだけを取得すればよい場合があります。

第 3 章

レジストリの検索

この章では、JAXR で提供されているレジストリ検索用のインタフェースとメソッドについて説明します。ここでは、次のトピックについて説明します。

基本クエリーのメソッド

クライアントがレジストリを使用するもっとも単純な形態は、レジストリ内にあるオブジェクトやデータについての情報の検索です。`QueryManager`、`BusinessQueryManager`、および `RegistryObject` の各インタフェースでは、多くの検索メソッドおよび取得メソッドがサポートされています。これらのメソッドによって、クライアントは JAXR 情報モデルを使用してデータを検索できます。多くの検索メソッドは `BulkResponse` を返します。`BulkResponse` は、メソッドの引数に指定された一連の条件を満たすオブジェクトのコレクションです。これらのメソッドのうち、もっとも一般的なものは次のとおりです。

- `getRegistryObject` と `getRegistryObjects`。これらの `QueryManager` メソッドは、引数とともに使用すると、オブジェクト型または一意の識別子に基づいて 1 つまたは複数のオブジェクトを返します。引数を指定しない場合、`getRegistryObjects` メソッドは呼び出し元に所有されているオブジェクトを返します。一意の識別子については、32 ページの「一意の識別子によるオブジェクトの検索」を参照してください。
- `findObjects`。`BusinessQueryManager` クラスの実装固有のメソッド。指定された条件を満たし、指定された型を持つすべてのオブジェクトのリストを返します。

他の検索メソッドでは、JAXR 情報モデルでサポートされている特定の種類のオブジェクトを検索できます。UDDI レジストリでは、オブジェクトの特定の階層がサポートされます。すなわち、ユーザー、サービス、およびサービスバインディングを含む組織です。一方、ebXML レジストリでは、独立したさまざまな型のオブジェクトの格納を可能にし、いろいろな方法でオブジェクトどうしをリンクできます。ほかのオブジェクトは、独立したオブジェクトではなく、常に別のオブジェクトの属性になります。

`BusinessQueryManager` の検索メソッドは、主に UDDI レジストリの検索に役立ちます。より一般的な `findObjects` メソッドと `RegistryObject` の取得メソッドは、Service Registry に適しています。

クエリーを実行するためにレジストリにログインする必要はありません。デフォルトでは、未認証のユーザーには「Registry Guest」というユーザーアイデンティティーが付与されます。

JAXR 情報モデルのインタフェース

表 3–1 に、JAXR 情報モデルでサポートされている主要なインタフェースを示します。これらのインタフェースはすべて、`RegistryObject` インタフェースを拡張します。

より詳しい説明と、これらのインタフェース間の関係を示す図については、`javax.xml.registry.infomodel` パッケージの API ドキュメント (`http://java.sun.com/j2ee/1.4/docs/api/javax/xml/registry/infomodel/package-summary.html`) を参照してください。

表 **3–1** JAXR `RegistryObject` のサブインタフェース

インタフェース名	説明
`Association`	2 つのオブジェクトの関係を定義します。 取得メソッドおよび検索メソッド: `RegistryObject.getAssociations`、`BusinessQueryManager.findAssociations`、`BusinessQueryManager.findCallerAssociations`
`AuditableEvent`	オブジェクトに加えられた変更の履歴を表します。`AuditableEvent` オブジェクトのコレクションで、オブジェクトの監査証跡が構成されます。 取得メソッド: `RegistryObject.getAuditTrail`
`Classification`	`ClassificationScheme` を使用してオブジェクトを分類します。 取得メソッド: `RegistryObject.getClassifications`
`ClassificationScheme`	オブジェクトの分類に使用される分類方式を表します。内部 `ClassificationScheme` では、すべての分類方式の要素は `Concept` インスタンスとしてレジストリに定義されます。外部 `ClassificationScheme` では、値はレジストリで `Concept` インスタンスとして定義されるのではなく、その `String` 表現によって参照されます。 検索メソッド: `BusinessQueryManager.findClassificationSchemes`、`BusinessQueryManager.findClassificationSchemeByName`
`Concept`	分類方式の要素、および内部 `ClassificationScheme` のほかの要素との構造的関係を表します。ebXML 仕様では `ClassificationNode` と呼ばれます。 検索メソッド: `BusinessQueryManager.findConcepts`、`BusinessQueryManager.findConceptByPath`
`ExternalIdentifier`	識別スキーマ (外部 `ClassificationScheme`) の内部で、`String` 値を使用してオブジェクトに関する追加情報を提供します。識別スキーマの例には、DUNS 番号や社会保障番号などがあります。 取得メソッド: `RegistryObject.getExternalIdentifiers`
`ExternalLink`	レジストリの外部に存在するコンテンツの URI を表します。 取得メソッド: `RegistryObject.getExternalLinks`
`ExtrinsicObject`	レジストリに本来組み込まれていない型を持つコンテンツが送信された場合は、MIME タイプなどの追加の属性を使用して、そのコンテンツを記述する必要があります。このインタフェースは、そのようなメタデータを表します。 特定の取得メソッドや検索メソッドはありません。

表 **3–1** JAXR `RegistryObject` のサブインタフェース　(続き)

インタフェース名	説明
`Organization`	組織に関する情報を表します。1 つの親組織、および 1 つ以上の子組織を持つことができます。常に主担当者として User オブジェクトを持ち、Service オブジェクトを提供することもできます。 検索メソッド: `BusinessQueryManager.findOrganizations`
`RegistryPackage`	レジストリオブジェクトの論理的なグループ化を表します。`RegistryPackage` は任意の数の `RegistryObject` を持つことができます。 取得および検索メソッド: `RegistryObject.getRegistryPackages`、`BusinessQueryManager.findRegistryPackages`
`Service`	サービスに関する情報を表します。`ServiceBinding` オブジェクトの集合を持つことができます。 検索メソッド: `BusinessQueryManager.findServices`
`ServiceBinding`	`Service` へのアクセス方法に関する技術情報を表します。 取得および検索メソッド: `Service.getServiceBindings`、`BusinessQueryManager.findServiceBindings`
`Slot`	`RegistryObject` インスタンスに任意の属性を動的に追加する手段を提供します。 取得メソッド: `RegistryObject.getSlot`、`RegistryObject.getSlots`
`SpecificationLink`	`ServiceBinding` と技術仕様のリンクを表します。技術仕様には、`ServiceBinding` を用いてサービスを利用する方法が記述されています。 取得メソッド: `ServiceBinding.getSpecificationLinks`
`User`	レジストリ内の登録済みユーザーに関する情報を表します。`User` オブジェクトは `Organization` オブジェクトに関連付けられます。 取得メソッド: `Organization.getUsers`、`Organization.getPrimaryContact`

表 3–2 に、JAXR 情報モデルでサポートされているその他のインタフェースを示します。これらのインタフェースは、主要なレジストリオブジェクトの属性を表します。これらのインタフェースは `RegistryObject` インタフェースを拡張しません。

表 **3–2** 属性として使用される JAXR 情報モデルのインタフェース

インタフェース名	説明
`EmailAddress`	電子メールアドレスを表します。`User` は 1 つの `EmailAddress` を持つことができます。 取得メソッド: `User.getEmailAddresses`
`InternationalString`	複数のロケールに国際化できる `String` を表します。`LocalizedString` オブジェクトの `Collection` が含まれます。`RegistryObject` の名前と説明は `InternationalString` オブジェクトです。 取得メソッド: `RegistryObject.getName`、`RegistryObject.getDescription`
`Key`	`RegistryObject` を識別するオブジェクトです。一意の識別子の値を含みます。この値は DCE 128 UUID (Universal Unique IDentifier) である必要があります。 取得メソッド: `RegistryObject.getKey`
`LocalizedString`	`String` とその `Locale` を関連付ける、`InternationalString` のコンポーネントです。 取得メソッド: `InternationalString.getLocalizedStrings`
`PersonName`	個人の名前を表します。`User` は `PersonName` を 1 つ持ちます。 取得メソッド: `User.getPersonName`
`PostalAddress`	住所を表します。`Organization` または `User` は 1 つ以上の `PostalAddress` オブジェクトを持つことができます。 取得メソッド: `Organization.getPostalAddress`、`OrganizationImpl.getPostalAddresses` (実装に固有)、`User.getPostalAddresses`
`TelephoneNumber`	電話番号を表します。`Organization` または `User` は 1 つ以上の `TelephoneNumber` オブジェクトを持つことができます。 取得メソッド: `Organization.getTelephoneNumbers`、`User.getTelephoneNumbers`

一意の識別子によるオブジェクトの検索

Service Registry 内のすべてのオブジェクトは、一意の識別子 (Key とも呼ばれる) と論理識別子の 2 つの識別子を持ちます。多くの場合、一意の識別子と論理識別子は同じです。しかし、あるオブジェクトが複数のバージョンで存在する場合、一意の識別子はバージョンごとに異なりますが、論理識別子は同じになります。(52 ページの「オブジェクトのバージョンの取得」を参照)。

オブジェクトの一意の識別子の値がわかっている場合は、その String 値を引数として指定して QueryManager.getRegistryObject メソッドを呼び出すことで、オブジェクトを取得できます。たとえば、bqm という BusinessQueryManager インスタンスがあり、String 値が idString である場合は、次のコードでオブジェクトを取得できます。

```
RegistryObject obj = bqm.getRegistryObject(idString);
```

オブジェクトを取得したら、オブジェクトの型、名前、説明などの属性を取得できます。

一意の識別子によるオブジェクトの検索: 例

一意の識別子によってオブジェクトを検索する例については、*<INSTALL>*/registry/samples/search-id/src ディレクトリにある JAXRSearchById.java を参照してください。このサンプルは、指定された一意の識別子を持つオブジェクトを検索します。

▼ JAXRSearchById サンプルを実行するには

手順

1. **<*INSTALL*>/registry/samples/search-id** ディレクトリに移動します。
2. 次のコマンドを入力します。

   asant run -Did=*urn_value*

   たとえば、次の ID を指定すると、ObjectType 分類方式に関する情報を取得できます。

   ```
   urn:oasis:names:tc:ebxml-regrep:classificationScheme:ObjectType
   ```

名前によるオブジェクトの検索

オブジェクトを名前で検索するには、通常、検索修飾子と名前パターンの組み合わせを使用します。検索修飾子はソートとパターンマッチングに作用します。名前パターンには検索する文字列を指定します。

`BusinessQueryManagerImpl.findObjects` メソッドは、2 番目の引数として `FindQualifier` オブジェクトのコレクション、3 番目の引数として名前パターンのコレクションを取ります。メソッドシグニチャーは次のとおりです。

```
public BulkResponse findObjects(java.lang.String objectType,
        java.util.Collection findQualifiers,
        java.util.Collection namePatterns,
        java.util.Collection classifications,
        java.util.Collection specifications,
        java.util.Collection externalIdentifiers,
        java.util.Collection externalLinks)
    throws JAXRException
```

最初の引数にはオブジェクト型を指定しますが、通常は、`LifeCycleManager` インタフェースで定義されている一連の文字列定数の 1 つを指定します。

名前パターンにはワイルドカードを使用できます。パーセント記号 (`%`) を用いると、指定した検索文字列をオブジェクト名の先頭、中間、または末尾で検索するよう指定できます。次に例を示します。

- **`nor%`** と指定すると、`Nor` または `nor` で始まる文字列を検索できます。たとえば、`North` や `northern` などです。
- **`%off%`** と指定すると、文字列 `off` を含む文字列を検索できます。たとえば、`Coffee` などです。
- **`%ica`** と指定すると、`ica` で終わる文字列を検索できます。たとえば、`America` などです。

1 文字分に相当するワイルドカードとして、下線 (_) も使用できます。たとえば、検索文字列 `_us_` は、`Aus1` や `Bus3` などのオブジェクト名に一致します。

次のコードは、指定された文字列 `searchString` で始まる名前を持つすべての組織を Service Registry から検索し、見つかった組織名をアルファベット順にソートします。

```
// Define find qualifiers and name patterns
Collection findQualifiers = new ArrayList();
findQualifiers.add(FindQualifier.SORT_BY_NAME_ASC);
Collection namePatterns = new ArrayList();
namePatterns.add(searchString + "%");

// Find organizations with name that starts with searchString
BulkResponse response =
     bqm.findObjects("Organization", findQualifiers,
         namePatterns, null, null, null, null);
```

```
Collection orgs = response.getCollection();
```

FindQualifier.CASE_SENSITIVE_MATCH を指定しない限り、findObjects メソッドでは大文字と小文字が区別されません。上記のコードにおいて、最初の引数は "Organization" または "organization" のどちらでもよく、名前パターンは名前に一致します。大文字と小文字は区別されません。

次のコードは、文字列 searchString が名前に含まれているすべてのレジストリオブジェクトを検索し、アルファベット順にソートします。この検索では大文字と小文字を区別します。

```
Collection findQualifiers = new ArrayList();
findQualifiers.add(FindQualifier.CASE_SENSITIVE_MATCH);
findQualifiers.add(FindQualifier.SORT_BY_NAME_ASC);
Collection namePatterns = new ArrayList();
namePatterns.add("%" + searchString + "%");

// Find objects with name that contains searchString
BulkResponse response =
     bqm.findObjects("RegistryObject", findQualifiers,
         namePatterns, null, null, null, null);
Collection orgs = response.getCollection();
```

パーセント記号は任意の数の文字に一致します。1 文字に一致させるには、下線 (_) を使用します。たとえば、"Arg1" と "Org2" の両方に一致させるには、_rg_ という名前パターンを指定します。

名前によるオブジェクトの検索: 例

名前によってオブジェクトを検索する例については、*<INSTALL>*/registry/samples/search-name/src ディレクトリにある JAXRSearchByName.java を参照してください。

▼ JAXRSearchByName サンプルを実行するには

手順 **1.** **<*INSTALL*>/registry/samples/search-name** ディレクトリに移動します。

2. 次のコマンドを入力して **string** 値を指定します。

asant run -Dname=*string*

このプログラムは、大文字と小文字を区別せずに、指定された文字列が名前に含まれているすべてのオブジェクトを検索します。また、オブジェクトの分類、外部識別子、外部リンク、スロット、および監査証跡を表示します。

型によるオブジェクトの検索

指定された型を持つすべてのオブジェクトを検索するには、`BusinessQueryManagerImpl.findObjects` メソッドの最初の引数だけを指定し、必要に応じて `FindQualifier` オブジェクトのコレクションも指定します。たとえば、`typeString` が `"Service"` または `"service"` という値を持つ文字列である場合、次のコードは、Service Registry 内のすべてのサービスを検索し、アルファベット順にソートします。

```
Collection findQualifiers = new ArrayList();
findQualifiers.add(FindQualifier.SORT_BY_NAME_ASC);

BulkResponse response = bqm.findObjects(typeString,
     findQualifiers, null, null, null, null, null);
```

`findObjects` の最初の引数にはワイルドカードを使用できません。

型によるオブジェクトの検索: 例

型によってオブジェクトを検索する例については、*<INSTALL>*`/registry/samples/search-object-type/src` ディレクトリにある `JAXRSearchByObjectType.java` を参照してください。

▼ JAXRSearchByObjectType サンプルを実行するには

手順

1. ***<INSTALL>*`/registry/samples/search-object-type`** ディレクトリに移動します。
2. 次のコマンドを入力して **string** 値を指定します。

   `asant run -Dtype=`*`type_name`*

   このプログラムは、大文字と小文字を区別せずに、*type_name* で指定された型を持つすべてのオブジェクトを検索します。また、オブジェクトの名前、説明、および一意の識別子を表示します。次のコマンド行のように、ワイルドカードを使わずに型の名前を正確に指定してください。

   `asant run -Dtype=federation`

分類によるオブジェクトの検索

オブジェクトを分類によって検索するには、まず、特定の分類スキーマの内部での分類を確立します。次に、`BusinessQueryManagerImpl.findObjects` メソッドへの引数として分類を指定します。

特定の分類スキーマの内部で分類を確立するには、まず分類スキーマを検索します。次に、`findObjects` メソッドまたは別の検索メソッドへの引数として使用される `Classification` オブジェクトを作成します。

次のコードは、International Organization for Standardization (ISO) によって管理されている ISO 3166 国番号分類スキーマの内部の特定の分類に対応するすべての組織を検索します。詳細については、`http://www.iso.org/iso/en/prods-services/iso3166ma/index.html` を参照してください。この分類スキーマは、Service Registry に付属するサンプルデータベースで提供されています。

```
ClassificationScheme cScheme =
     bqm.findClassificationSchemeByName(null,
         "iso-ch:3166:1999");

Classification classification =
     blcm.createClassification(cScheme, "United States", "US");
Collection classifications = new ArrayList();
classifications.add(classification);
// perform search
BulkResponse response = bqm.findObjects("Organization", null,
    null, classifications, null, null, null);
Collection orgs = response.getCollection();
```

ebXML Registry Information Model Specification (レジストリ情報モデル仕様) では、一連の標準的な分類スキーマを ebXML レジストリに用意しておく必要があります。各スキーマには、ebXML 仕様で `ClassificationNode` オブジェクトと呼ばれる一連の必須 Concept もあります。標準的な分類スキーマの主な目的は、オブジェクトを分類することではなく、オブジェクトの属性のタイプを列挙することです。たとえば、`EmailType` 分類スキーマでは、`EmailAddress` オブジェクトの `type` 属性の値が列挙されています。

表 3–3 に、これらの標準的な分類スキーマの概要を示します。

表 3–3 標準的な分類スキーマ

分類スキーマ	説明
`AssociationType`	`RegistryObject` どうしの関連付けのタイプを定義します。
`ContentManagementService`	コンテンツ管理サービスのタイプを定義します。

表 3–3 標準的な分類スキーマ　(続き)

分類スキーマ	説明
DataType	仕様で定義されているクラスの属性のデータ型を定義します。
DeletionScopeType	RemoveObjectsRequest プロトコルメッセージの deletionScope 属性の値を定義します。
EmailType	電子メールアドレスのタイプを定義します。
ErrorHandlingModel	コンテンツ管理サービスのエラー処理モデルのタイプを定義します。
ErrorSeverityType	レジストリでプロトコルメッセージの処理中に発生するエラーの重要度のタイプを定義します。
EventType	レジストリで発生する可能性のあるイベントのタイプを定義します。
InvocationModel	レジストリでコンテンツ管理サービスを呼び出す方法を定義します。
NodeType	ClassificationScheme がその ClassificationNode の code 属性の値を割り当てる方法を定義します。
NotificationOptionType	クライアントがレジストリから Subscription 内部のイベントの通知を受け取る方法を定義します。
ObjectType	レジストリでサポートできる RegistryObject のタイプを定義します。
PhoneType	電話番号のタイプを定義します。
QueryLanguage	レジストリでサポートされるクエリー言語を定義します。
ResponseStatusType	RegistryResponse の状態のタイプを定義します。
StatusType	RegistryObject の状態のタイプを定義します。
SubjectGroup	アクセス制御の目的で User が所属できるグループを定義します。
SubjectRole	アクセス制御の目的で User に割り当てることのできるロールを定義します。

標準的な分類スキーマとその Concept を使用するオブジェクトを検索するために、パッケージ org.freebxml.common.CanonicalConstants で定義されている文字列定数を使用してオブジェクトを検索できます。定数の一覧については、92 ページの「分類スキーマに対する定数」を参照してください。

まず、一意の識別子の値を使用して分類スキーマを検索します。

```
String schemeId =
    CanonicalConstants.CANONICAL_CLASSIFICATION_SCHEME_ID_SubjectRole;
ClassificationScheme cScheme =
    (ClassificationScheme) bqm.getRegistryObject(schemeId);
String schemeName = getName(cScheme);
```

次に、同じ方法で Concept を検索し、それを使用して分類を作成します。

```
String concId =
    CanonicalConstants.CANONICAL_SUBJECT_ROLE_ID_RegistryAdministrator;
Concept concept = (Concept) bqm.getRegistryObject(concId);
Classification classification =
    blcm.createClassification(concept);
```

最後に、標準的な分類スキーマ以外の場合と同じ方法で、オブジェクトを検索します。

```
Collection classifications = new ArrayList();
classifications.add(classification);
BulkResponse response = bqm.findObjects("RegistryObject",
    null, null, classifications, null, null, null);
Collection objects = response.getCollection();
```

すべての標準的な分類スキーマとその Concept を表示するサンプルプログラムについては、 *<INSTALL>*/registry/samples/classification-schemes/src ディレクトリにある JAXRGetCanonicalSchemes.java を参照してください。

▼ JAXRGetCanonicalSchemes サンプルを実行するには

手順

1. **<*INSTALL*>/registry/samples/classification-schemes** ディレクトリに移動します。
2. 次のコマンドを入力します。

   ```
   asant get-schemes
   ```

分類によるオブジェクトの検索: 例

分類によってオブジェクトを検索する例については、*<INSTALL>*/registry/samples/search-classification/src ディレクトリにある JAXRSearchByClassification.java および JAXRSearchByCountryClassification.java を参照してください。最初のサンプルは、標準的な分類スキーマ SubjectRole を使用するオブジェクトを検索します。2 番目のサンプルは、地理的な分類を使用する組織を検索します。

▼ JAXRSearchByClassification サンプルおよび JAXRSearchByCountryClassification サンプルを実行するには

始める前に　JAXRSearchByCountryClassification サンプルで結果を取得するには、指定された分類を使用するオブジェクトを発行する必要があります。68 ページの「分類の追加: 例」または71 ページの「組織の作成: 例」のサンプルをまず実行してください。

手順

1. **<*INSTALL*>/registry/samples/search-classification** ディレクトリに移動します。
2. 次のいずれかのコマンドを入力します。

```
asant search-class
asant search-geo
```

通常、search-class ターゲットは 1 つの結果を返します。search-geo ターゲットは、68 ページの「分類の追加: 例」の run ターゲットまたは71 ページの「組織の作成: 例」の pub-org ターゲットが実行済みであれば、結果を返します。

外部識別子によるオブジェクトの検索

外部識別子によるオブジェクトの検索は、分類によるオブジェクトの検索に似ています。まず分類スキーマを検索し、次に BusinessQueryManagerImpl.findObjects メソッドなどの検索メソッドの引数として使用する ExternalIdentifier オブジェクトを作成します。

次のコードは、外部識別子として Sun Microsystems の株価表示記号を含んでいるすべてのレジストリオブジェクトを検索します。このサンプルを正しく動作させるには、NASDAQ という名前の外部分類スキーマを作成する必要があります。その方法の詳細は、68 ページの「オブジェクトへの外部識別子の追加」を参照してください。

外部識別子のコレクションは、findObjects メソッドの最後から 2 番目の引数として指定します。

```
ClassificationScheme cScheme = null;
cScheme =
     bqm.findClassificationSchemeByName(null, "NASDAQ");

ExternalIdentifier extId =
     blcm.createExternalIdentifier(cScheme, "%Sun%",
        "SUNW");
Collection extIds = new ArrayList();
```

```
extIds.add(extId);
// perform search
BulkResponse response = bqm.findObjects("RegistryObject",
     null, null, null, null, extIds, null);
Collection objects = response.getCollection();
```

外部識別子によるオブジェクトの検索: 例

外部識別子によってオブジェクトを検索する例については、*<INSTALL>*/registry/samples/search-external-identifier/src ディレクトリにある JAXRSearchByExternalIdentifier.java を参照してください。このサンプルは、NASDAQ 分類スキーマを使用するオブジェクトを検索します。

▼ JAXRSearchByExternalIdentifier サンプルを実行するには

始める前に　このサンプルで結果を取得するには、まず 68 ページの「分類の追加: 例」の publish-object サンプルを実行する必要があります。

手順

1. **<*INSTALL*>/registry/samples/search-external-identifier** ディレクトリに移動します。
2. 次のコマンドを入力します。

   asant run

外部リンクによるオブジェクトの検索

外部リンクによってオブジェクトを検索するには、分類スキーマを使用する必要はありませんが、有効な URI を指定する必要があります。createExternalLink メソッドの引数は、URI と説明です。

ファイアウォールの外側へのリンクを指定する場合は、JAXR で URI の有効性を確認できるように、プログラムの実行時にシステムプロパティー http.proxyHost および http.proxyPort も指定する必要があります。

次のコードは、指定された ExternalLink オブジェクトを持つすべての組織を検索します。

```
ExternalLink extLink =
     blcm.createExternalLink("http://java.sun.com/",
         "Sun Java site");
```

```
Collection extLinks = new ArrayList();
extLinks.add(extLink);
BulkResponse response = bqm.findObjects("Organization",
     null, null, null, null, null, extLinks);
Collection objects = response.getCollection();
```

外部リンクによるオブジェクトの検索: 例

外部リンクによってオブジェクトを検索する例については、*<INSTALL>*/registry/samples/search-external-link/src ディレクトリにある JAXRSearchByExternalLink.java を参照してください。このサンプルは、指定された外部リンクを持つオブジェクトを検索します。http.proxyHost プロパティーと http.proxyPort プロパティーは、build.xml ファイルの run ターゲットで指定されます。

▼ JAXRSearchByExternalLink サンプルを実行するには

始める前に　このサンプルで結果を取得するには、まず 68 ページの「分類の追加: 例」の publish-object サンプルを実行する必要があります。

手順

1. **<*INSTALL*>/registry/sample/s/search-external-link** ディレクトリに移動します。
2. 次のコマンドを入力します。

   asant run

発行したオブジェクトの検索

/Service Registry に発行したすべてのオブジェクトを取得できます。また、検索を絞り込み、発行したオブジェクトのうち特定のオブジェクト型のものだけを取得することもできます。発行したすべてのオブジェクトを取得するには、引数なしの QueryManager.getRegistryObjects メソッドを使用します。このメソッドの名前は誤解を招きやすいのですが、このメソッドで取得できるのは、すべてのレジストリオブジェクトではなく、発行したオブジェクトだけです。

たとえば、bqm という BusinessQueryManager インスタンスの場合は、次のコードを使用します。

```
BulkResponse response = bqm.getRegistryObjects();
```

発行したオブジェクトのうち特定の型のものだけを取得するには、QueryManager.getRegistryObjects に String 引数を指定します。

```
BulkResponse response = bqm.getRegistryObjects("Service");
```

このメソッドでは大文字と小文字が区別されるので、オブジェクト型は大文字で始める必要があります。

サンプルプログラム JAXRGetMyObjects および JAXRGetMyObjectsByType は、これらのメソッドの使用方法を示しています。

発行したオブジェクトの検索: 例

分類によってオブジェクトを検索する例については、<*INSTALL*>/registry/samples/get-objects/src ディレクトリにある JAXRGetMyObjects.java および JAXRGetMyObjectsByType.java を参照してください。最初のサンプル JAXRGetMyObjects.java は、発行したすべてのオブジェクトを取得します。2 番目のサンプル JAXRGetMyObjectsByType.java は、発行したオブジェクトのうち指定した型のものをすべて取得します。

▼ JAXRGetMyObjects サンプルおよび JAXRGetMyObjectsByType サンプルを実行するには

手順

1. **<*INSTALL*>/registry/sample/s/get-objects** ディレクトリに移動します。
2. 発行したすべてのオブジェクトを検索するには、次のコマンドを入力します。

   ```
   asant get-obj
   ```

3. 発行したオブジェクトのうち特定の型のものをすべて取得するには、次のコマンドを入力します。*type_name* では、大文字と小文字が区別されます。

   ```
   asant get-obj-type -Dtype=type_name
   ```

オブジェクトに関する情報の取得

目的のオブジェクトを取得した後は、そのオブジェクトの属性や、そのオブジェクトに属しているほかのオブジェクトも取得できます。

- 名前
- 説明
- タイプ
- 一意の識別子と論理識別子
- 分類
- 外部識別子

- 外部リンク
- スロット

組織については、次の属性も取得できます。

- 主担当者 (User オブジェクト)
- 住所
- 電話番号
- サービス

サービスの場合は、サービスバインディングを取得できます。

任意のオブジェクトについて、監査証跡も取得できます。監査証跡には、オブジェクトの状態を変更したイベントやバージョンが含まれています。オブジェクトのバージョン番号も取得できます。バージョン番号は、オブジェクトの属性のいずれかに変更が加えられるたびに更新されます。

ここでは、次の項目について説明します。

- 43 ページの「オブジェクトの ID 値の取得」
- 44 ページの「オブジェクトの名前または説明の取得」
- 44 ページの「オブジェクトの型の取得」
- 44 ページの「オブジェクトの分類の取得」
- 45 ページの「オブジェクトの外部識別子の取得」
- 46 ページの「オブジェクトの外部リンクの取得」
- 46 ページの「オブジェクトのスロットの取得」
- 47 ページの「組織またはユーザーの属性の取得」
- 49 ページの「組織のサービスおよびサービスバインディングの取得」
- 50 ページの「組織の階層の取得」
- 51 ページの「オブジェクトの監査証跡の取得」
- 52 ページの「オブジェクトのバージョンの取得」

オブジェクトの ID 値の取得

オブジェクトの一意の識別子は Key オブジェクトに格納されています。Key は、String 値である id 属性の形式で識別子を格納する構造体です。識別子を取得するには、RegistryObject.getKey().getId() メソッドを呼び出します。

JAXR プロバイダには、lid と呼ばれる論理識別子を取得するための、実装に固有のメソッドもあります。lid は RegistryObject のString 属性です。lid を取得するには、 RegistryObjectImpl.getLid を呼び出します。このメソッドのシグニチャーは次のとおりです。

```
public java.lang.String getLid()
    throws JAXRException
```

このメソッドの使用例については、<*INSTALL*>/registry/samples/organizations/src ディレクトリにある JAXRSearchOrg.java を参照してください。このサンプルの詳細については、48 ページの「組織の属性の取得: 例」を参照してください。

オブジェクトの名前または説明の取得

オブジェクトの名前と説明は、どちらも InternationalString オブジェクトです。InternationalString オブジェクトには LocalizedString オブジェクトの集合が格納されます。RegistryObject.getName メソッドおよび RegistryObject.getDescription メソッドは、デフォルトロケールの LocalizedString オブジェクトを返します。その後、LocalizedString オブジェクトの String 値を取得できます。これらのメソッドを使用するコードの例を次に示します。

```
String name = ro.getName().getValue();
String description = ro.getDescription().getValue();
```

特定のロケールの値を取得するには、Locale 引数を指定して getName メソッドまたは getDescription メソッドを呼び出します。

サンプルの多くには、オブジェクトの名前、説明、および一意の識別子を取得する private のユーティリティーメソッドが含まれています。たとえば、*<INSTALL>*ー/registry/samples/get-objects/src ディレクトリにある JAXRGetMyObjects.java を参照してください。

オブジェクトの型の取得

特定のオブジェクト型を指定せずに Service Registry を検索した場合、検索によって返されたオブジェクトの型を取得できます。Concept 値を返す RegistryObject.getObjectType メソッドを使用します。次に、Concept.getValue メソッドを使用して、オブジェクト型の String 値を取得します。これらのメソッドを使用するコードの例を次に示します。

```
Concept objType = object.getObjectType();
System.out.println("Object type is " + objType.getValue());
```

Concept は、標準的な分類スキーマ ObjectType の Concept のいずれかになります。このコードの例については、*<INSTALL>*/registry/samples/search-name/src ディレクトリにある JAXRSearchByName.java を参照してください。

オブジェクトの分類の取得

オブジェクトの分類の Collection を取得するには、RegistryObject.getClassifications メソッドを使用します。分類に関して重要な属性は、分類の値と、その分類が属している分類スキーマです。多くの場合、分類には名前や説明がありません。次のコードは、オブジェクトの分類を取得して表示します。

```
Collection classifications = object.getClassifications();
Iterator classIter = classifications.iterator();
while (classIter.hasNext()) {
```

```
    Classification classification =
         (Classification) classIter.next();
    String name = classification.getName().getValue();
    System.out.println("  Classification name is " + name);
    System.out.println("  Classification value is " +
        classification.getValue());
    ClassificationScheme scheme =
         classification.getClassificationScheme();
    System.out.println("  Classification scheme for " +
        name + " is " + scheme.getName().getValue());
}
```

一部のサンプルには、これに似たコードを使用する showClassifications メソッドがあります。たとえば、*<INSTALL>*/registry/samples/search-name/src ディレクトリにある JAXRSearchByName.java を参照してください。

オブジェクトの外部識別子の取得

オブジェクトの外部識別子の Collection を取得するには、RegistryObject.getExternalIdentifiers メソッドを使用します。各識別子について、その名前、値、およびそれが属している分類スキーマを取得できます。外部識別子の分類スキーマを取得するメソッドは getIdentificationScheme です。次のコードは、オブジェクトの外部識別子を取得して表示します。

```
Collection exIds = object.getExternalIdentifiers();
Iterator exIdIter = exIds.iterator();
while (exIdIter.hasNext()) {
    ExternalIdentifier exId =
         (ExternalIdentifier) exIdIter.next();
    String name = exId.getName().getValue();
    System.out.println("  External identifier name is " +
        name);
    String exIdValue = exId.getValue();
    System.out.println("  External identifier value is " +
         exIdValue);
    ClassificationScheme scheme =
         exId.getIdentificationScheme();
    System.out.println("  External identifier " +
         "classification scheme is " +
         scheme.getName().getValue());
}
```

一部のサンプルには、これに似たコードを使用する showExternalIdentifiers メソッドがあります。たとえば、*<INSTALL>*/registry/samples/search-name/src ディレクトリにある JAXRSearchByName.java を参照してください。

オブジェクトの外部リンクの取得

オブジェクトの外部リンクの Collection を取得するには、RegistryObject.getExternalLinks メソッドを使用します。各外部リンクについて、名前、説明、および値を取得できます。外部リンクについては、名前は省略可能です。次のコードは、オブジェクトの外部リンクを取得して表示します。

```
Collection exLinks = obj.getExternalLinks();
Iterator exLinkIter = exLinks.iterator();
while (exLinkIter.hasNext()) {
    ExternalLink exLink = (ExternalLink) exLinkIter.next();
    String name = exLink.getName().getValue();
    if (name != null) {
        System.out.println("  External link name is " + name);
    }
    String description = exLink.getDescription().getValue();
    System.out.println("  External link description is " +
        description);
    String externalURI = exLink.getExternalURI();
    System.out.println("  External link URI is " +
         externalURI);
}
```

一部のサンプルには、これに似たコードを使用する showExternalLinks メソッドがあります。たとえば、*<INSTALL>*/registry/samples/search-name/src ディレクトリにある JAXRSearchByName.java を参照してください。

オブジェクトのスロットの取得

スロットは、オブジェクトに対して作成できる任意の属性です。オブジェクトのスロットの Collection を取得するには、RegistryObject.getSlots メソッドを使用します。各スロットについて、名前、値、およびタイプを取得できます。Slot オブジェクトの名前は String で、InternationalString ではありません。スロットは値の Collection を持ちます。次のコードは、オブジェクトのスロットを取得して表示します。

```
Collection slots = object.getSlots();
Iterator slotIter = slots.iterator();
while (slotIter.hasNext()) {
    Slot slot = (Slot) slotIter.next();
    String name = slot.getName();
    System.out.println("  Slot name is " + name);
    Collection values = slot.getValues();
    Iterator valIter = values.iterator();
    int count = 1;
    while (valIter.hasNext()) {
        String value = (String) valIter.next();
        System.out.println("  Slot value " + count++ +
             ": " + value);
    }
    String type = slot.getSlotType();
```

```
    if (type != null) {
        System.out.println("  Slot type is " + type);
}
```

一部のサンプルには、このコードを使用する showSlots メソッドがあります。たとえば、*<INSTALL>*/registry/samples/search-name/src ディレクトリにある JAXRSearchByName.java を参照してください。

組織またはユーザーの属性の取得

各 Organization オブジェクトは、ほかのすべてのオブジェクトで使用可能な属性に加え、1 つの住所と複数の電話番号を持つことができます。各組織は主担当者として User オブジェクトも持ちます。組織には追加の User オブジェクトを関連付けることができます。

User オブジェクトの属性の 1 つに PersonName オブジェクトがありますが、その形式はオブジェクト名の形式とは異なっています。ユーザーは、複数の電話番号と同様に複数の住所を持つことができます。ユーザーは複数の電子メールアドレスも持つことができます。

組織の住所を取得するには、次のように Organization.getPostalAddress メソッドを呼び出します (org は組織)。

```
PostalAddress pAd = org.getPostalAddress();
```

住所を取得した後、次のようにして住所の属性を取得できます。

```
System.out.println(" Postal Address:\n  " +
    pAd.getStreetNumber() + " " + pAd.getStreet() +
    "\n  " + pAd.getCity() + ", " +
    pAd.getStateOrProvince() + " " +
    pAd.getPostalCode() + "\n  " + pAd.getCountry() +
    "(" + pAd.getType() + ")");
```

組織の主担当者を取得するには、次のように Organization.getPrimaryContact メソッドを呼び出します (org は組織)。

```
User pc = org.getPrimaryContact();
```

ユーザーの住所を取得するには、次のように User.getPostalAddresses メソッドを呼び出し、Collection の値を抽出します (pc は主担当者)。

```
Collection pcpAddrs = pc.getPostalAddresses();
Iterator pcaddIter = pcpAddrs.iterator();
while (pcaddIter.hasNext()) {
    PostalAddress pAd = (PostalAddress) pcaddIter.next();
    /* retrieve attributes */
}
```

組織またはユーザーの電話番号を取得するには、getTelephoneNumbers メソッドを呼び出します。次のコードで、org は組織を表しています。このコードは、国番号、市外局番、主番号、および電話番号のタイプを取得します。

```
Collection orgphNums = org.getTelephoneNumbers(null);
Iterator orgphIter = orgphNums.iterator();
while (orgphIter.hasNext()) {
    TelephoneNumber num = (TelephoneNumber) orgphIter.next();
    System.out.println(" Phone number: " +
         "+" + num.getCountryCode() + " " +
        "(" + num.getAreaCode() + ") " +
        num.getNumber() + " (" + num.getType() + ")");
}
```

TelephoneNumber には内線も含まれることがあり、これは getExtension メソッドで取得できます。電子的にダイアル可能な番号の場合は、url 属性も含まれることがあり、これは getUrl メソッドで取得できます。

ユーザーの名前を取得するには、User.getPersonName メソッドを呼び出します。PersonName には、ユーザーの名、ミドルネーム、および姓に対応する 3 つの属性があります。次のコードで、pc は主担当者を表しています。

```
PersonName pcName = pc.getPersonName();
System.out.println(" Contact name: " +
    pcName.getFirstName() + " " +
    pcName.getMiddleName() + " " +
    pcName.getLastName());
```

ユーザーの電子メールアドレスを取得するには、User.getEmailAddresses メソッドを呼び出します。EmailAddress には、アドレスとそのタイプの 2 つの属性があります。次のコードでは、pc は主担当者を表しています。

```
Collection eAddrs = pc.getEmailAddresses();
Iterator eaIter = eAddrs.iterator();
while (eaIter.hasNext()) {
    EmailAddress eAd = (EmailAddress) eaIter.next();
    System.out.println("  Email address: " +
        eAd.getAddress() + " (" + eAd.getType() + ")");
}
```

PostalAddress、TelephoneNumber、PersonName、および EmailAddress オブジェクトの属性はすべて String 値です。28 ページの「JAXR 情報モデルのインタフェース」で説明しているとおり、これらのオブジェクトは RegistryObject インタフェースを拡張しないため、ほかのレジストリオブジェクトの属性は持っていません。

組織の属性の取得: 例

組織、およびその主担当者である User の属性を取得するについては、*<INSTALL>*/registry/samples/organizations/src ディレクトリにある JAXRSearchOrg.java を参照してください。このサンプルは、指定された文字列が名前に含まれている組織についての情報を表示します。

▼ JAXRSearchOrg サンプルを実行するには

手順 **1.** **<*INSTALL*>/registry/samples/organizations** ディレクトリに移動します。

2. 次のコマンドを入力します。

```
asant search-org -Dorg=string
```

組織のサービスおよびサービスバインディングの取得

ほとんどの組織はサービスを提供します。JAXR には、組織のサービスおよびサービスバインディングを取得するメソッドがあります。

Service オブジェクトは、ほかのレジストリオブジェクトの属性をすべて持っています。さらに、通常は、サービスへのアクセス方法に関する情報を提供する「サービスバインディング」も持ちます。ServiceBinding は通常、ほかの属性に加えてアクセス URI と仕様リンクも持ちます。仕様リンクは、サービスバインディングと技術仕様をリンクします。技術仕様には、サービスバインディングを通してサービスを使用する方法が記述されています。仕様リンクには次の属性があります。

- 仕様オブジェクト。通常は ExtrinsicObject
- 使用法の説明。InternationalString オブジェクト
- 使用法パラメータの Collection。String 値

Service.getProvidingOrganization メソッドを使用して、サービスを提供している組織を取得できます。また、ServiceBinding.getService メソッドを使用して、サービスバインディングのサービスを取得できます。

次のコードは、組織 org のサービスを取得します。続いて、各サービスのサービスバインディングを取得し、各サービスバインディングについてそのアクセス URI と仕様リンクを取得します。

```
Collection services = org.getServices();
Iterator svcIter = services.iterator();
while (svcIter.hasNext()) {
    Service svc = (Service) svcIter.next();
    System.out.println(" Service name: " + getName(svc));
    System.out.println(" Service description: " +
        getDescription(svc));

    Collection serviceBindings = svc.getServiceBindings();
    Iterator sbIter = serviceBindings.iterator();
    while (sbIter.hasNext()) {
        ServiceBinding sb = (ServiceBinding) sbIter.next();
        System.out.println("  Binding name: " +
            getName(sb));
        System.out.println("  Binding description: " +
```

```
            getDescription(sb));
        System.out.println("  Access URI: " +
            sb.getAccessURI());

        Collection specLinks = sb.getSpecificationLinks();
        Iterator slIter = specLinks.iterator();
        while (slIter.hasNext()) {
            SpecificationLink sl =
                 (SpecificationLink) slIter.next();
            RegistryObject ro = sl.getSpecificationObject();
            System.out.println("Specification link " +
                "object of type " + ro.getObjectType());
            System.out.println("Usage description: " +
                sl.getUsageDescription().getValue());
            Collection ups = sl.getUsageParameters();
            Iterator upIter = ups.iterator();
            while (upIter.hasNext()) {
                String up = (String) upIter.next();
                System.out.println("Usage parameter: " +
                     up);
             }
        }
    }
}
```

48 ページの「組織の属性の取得: 例」のサンプルは、検索した組織のサービスおよびサービスバインディングも表示します。

サービスは組織から独立して存在することも多くあります。`BusinessQueryManagerImpl.findObjects` メソッドを使用すると、そのようなサービスを直接検索できます。

組織の階層の取得

JAXR では、組織をファミリーにグループ化することができます。組織はほかの組織をその子として持つことができます。子の組織もまた、子を持つことができます。したがって、組織は親、子、子孫を持つ場合があります。

`Organization.getParentOrganization` メソッドは、組織の親を取得します。次のコードで、`chorg` は子組織を表しています。

```
Organization porg = chorg.getParentOrganization();
```

`Organization.getChildOrganizations` メソッドは、組織の子の `Collection` を取得します。次のコードで、`org` は親組織を表しています。

```
Collection children = org.getChildOrganizations();
```

`Organization.getDescendantOrganizations` メソッドは、複数の世代の子孫を取得します。`Organization.getRootOrganization` メソッドは、任意の子孫の、親を持たない祖先を取得します。

組織の階層を取得する例については、72 ページの「組織階層の作成と取得: 例」を参照してください。

オブジェクトの監査証跡の取得

オブジェクトが Service Registry に発行されるたび、およびオブジェクトに何らかの変更が加えられるたびに、`AuditableEvent` と呼ばれる別のオブジェクトが JAXR プロバイダによって作成されます。JAXR プロバイダは、発行されたオブジェクトの監査証跡に `AuditableEvent` オブジェクトを追加します。監査証跡には、そのオブジェクトに関するすべてのイベントのリストが含まれています。監査証跡を取得するには、`RegistryObject.getAuditTrail` を呼び出します。監査証跡内の個別のイベントを取得し、そのイベントタイプを調べることもできます。JAXR では、51 ページの「オブジェクトの監査証跡の取得」に示されたイベントタイプがサポートされています。

表 3–4 `AuditableEvent` のタイプ

イベントタイプ	説明
`EVENT_TYPE_CREATED`	オブジェクトが作成され、レジストリに発行されました。
`EVENT_TYPE_DELETED`	`LifeCycleManager` または `BusinessLifeCycleManager` のいずれかの削除メソッドによってオブジェクトが削除されました。
`EVENT_TYPE_DEPRECATED`	`LifeCycleManager.deprecateObjects` メソッドによってオブジェクトが非推奨にされました。
`EVENT_TYPE_UNDEPRECATED`	`LifeCycleManager.unDeprecateObjects` メソッドによってオブジェクトが非推奨解除されました。
`EVENT_TYPE_VERSIONED`	オブジェクトの新しいバージョンが作成されました。このイベントは通常、オブジェクトのいずれかの属性が変更されたときに発生します。
`EVENT_TYPE_UPDATED`	オブジェクトが更新されました。
`EVENT_TYPE_APPROVED`	`LifeCycleManagerImpl.approveObjects` メソッドによってオブジェクトが承認されました (実装に固有)。
`EVENT_TYPE_DOWNLOADED`	オブジェクトがダウンロードされました (実装に固有)。
`EVENT_TYPE_RELOCATED`	オブジェクトが再配置されました (実装に固有)。

次のコードは、レジストリオブジェクトの監査証跡を取得し、各イベントのタイプとタイムスタンプを表示します。

```
Collection events = obj.getAuditTrail();
String objName = obj.getName().getValue();
Iterator eventIter = events.iterator();
while (eventIter.hasNext()) {
    AuditableEventImpl ae = (AuditableEventImpl) eventIter.next();
    int eType = ae.getEventType();
    if (eType == AuditableEvent.EVENT_TYPE_CREATED) {
        System.out.print(objName + " created ");
    } else if (eType == AuditableEvent.EVENT_TYPE_DELETED) {
        System.out.print(objName + " deleted ");
    } else if (eType == AuditableEvent.EVENT_TYPE_DEPRECATED) {
        System.out.print(objName + " deprecated ");
    } else if (eType == AuditableEvent.EVENT_TYPE_UNDEPRECATED) {
        System.out.print(objName + " undeprecated ");
    } else if (eType == AuditableEvent.EVENT_TYPE_UPDATED) {
        System.out.print(objName + " updated ");
    } else if (eType == AuditableEvent.EVENT_TYPE_VERSIONED) {
        System.out.print(objName + " versioned ");
    } else if (eType == AuditableEventImpl.EVENT_TYPE_APPROVED) {
        System.out.print(objName + " approved ");
    } else if (eType == AuditableEventImpl.EVENT_TYPE_DOWNLOADED) {
        System.out.print(objName + " downloaded ");
    } else if (eType == AuditableEventImpl.EVENT_TYPE_RELOCATED) {
        System.out.print(objName + " relocated ");
    } else {
        System.out.print("Unknown event for " + objName + " ");
    }System.out.println(ae.getTimestamp().toString());
}
```

一部のサンプルには、これに似たコードを使用する `showAuditTrail` メソッドがあります。たとえば、*<INSTALL>*`/registry/samples/search-name/src` ディレクトリにある `JAXRSearchByName.java` を参照してください。

レジストリオブジェクトの状態を変更する方法については、84 ページの「レジストリ内のオブジェクトの状態の変更」を参照してください。

オブジェクトのバージョンの取得

レジストリオブジェクトの属性を変更すると、Service Registry によってオブジェクトの新しいバージョンが作成されます。この仕組みの詳細については、84 ページの「レジストリ内のオブジェクトの状態の変更」を参照してください。オブジェクトを最初に作成したとき、そのオブジェクトのバージョンは 1.1 になります。

オブジェクトのバージョンを取得するには、レジストリオブジェクト用の実装に固有の `getVersionInfo` メソッドを使用します。このメソッドは、`VersionInfoType` オブジェクトを返します。このメソッドのシグニチャーは次のとおりです。

```
public VersionInfoType getVersionInfo()
    throws JAXRException
```

たとえば、組織 `org` のバージョン番号を取得するには、メソッドを呼び出すときに `org` を `RegistryObjectImpl` にキャストします。次に、`String` を返す `VersionInfoType.getVersionName` メソッドを呼び出します。

```
import org.oasis.ebxml.registry.bindings.rim.VersionInfoType;
...
VersionInfoType vInfo =
     ((RegistryObjectImpl)org).getVersionInfo();
if (vInfo != null) {
    System.out.println("Org version: " +
        vInfo.getVersionName());
}
```

一部のサンプルでは、このコードに似たコードが使用されています。たとえば、<*INSTALL* >/registry/samples/search-name/src ディレクトリにある JAXRSearchByName.java を参照してください。

宣言型クエリーの使用

BusinessQueryManager インタフェースの代わりに DeclarativeQueryManager インタフェースを使用して、Service Registry に対するクエリーを作成および実行できます。SQL に習熟している場合は、宣言型クエリーの方が使いやすいことがあります。DeclarativeQueryManager インタフェースは、Query という別のインタフェースに依存しています。

DeclarativeQueryManager インタフェースには、createQuery と executeQuery の 2 つのメソッドがあります。createQuery メソッドは、クエリータイプおよびクエリーを含む文字列を 2 つの引数として取ります。次のコードは、レジストリ内のすべての Service オブジェクトのリストを要求する SQL クエリーを作成します。ここで、rs は RegistryService オブジェクトです。

```
DeclarativeQueryManager qm = rs.getDeclarativeQueryManager();
String qString = "select s.* from Service s";
Query query = qm.createQuery(Query.QUERY_TYPE_SQL, qString);
```

クエリーを作成したら、次のようにクエリーを実行します。

```
BulkResponse response = qm.executeQuery(query);
Collection objects = response.getCollection();
```

次に、通常のクエリーの場合と同じ方法で、応答からオブジェクトを抽出します。

SQL クエリーの構文の詳細と例については、ebRS 3.0 仕様の第 6 章「Query Management Protocols」、特にセクション 6.6 を参照してください。

宣言型クエリーの使用: 例

宣言型クエリーの使用例については、
<*INSTALL*>/registry/samples/query-declarative/src ディレクトリにある JAXRQueryDeclarative.java および JAXRGetAllSchemes.java を参照してください。どちらのサンプルも、SQL クエリーを作成して実行します。クエリー文字列は JAXRExamples.properties ファイルで定義されています。

JAXRQueryDeclarative の SQL クエリー文字列は次のようになります。これは全体を 1 行に記述してください。

```
SELECT ro.* from RegistryObject ro, Name nm, Description d
WHERE upper(nm.value) LIKE upper('%free%') AND upper(d.value)
LIKE upper('%free%') AND (ro.id = nm.parent AND ro.id = d.parent)
```

このクエリーは、名前と説明の属性の両方に文字列 "free" を持つすべてのオブジェクトを検索します。

JAXRGetAllSchemes の SQL クエリー文字列は次のようになります。

```
SELECT * FROM ClassScheme s order by s.id
```

このクエリーは、レジストリ内のすべての分類スキーマを検索します。

▼ JAXRQueryDeclarative サンプルを実行するには

手順

1. **<*INSTALL*>/registry/samples/query-declarative** ディレクトリに移動します。
2. **JAXRQueryDeclarative** サンプルを実行するには、次のコマンドを入力します。

   ```
   asant get-free
   ```

3. **JAXRGetAllSchemes** サンプルを実行するには、次のコマンドを入力します。

   ```
   asant get-schemes
   ```

繰り返し型クエリーの使用

宣言型クエリーでは非常に大きな結果セットが返されると予測される場合には、実装に固有の繰り返し型クエリー機能を使用できます。
DeclarativeQueryManagerImpl.executeQuery メソッドには、一連のパラメータを指定する引数を 1 つ指定できます。このメソッドのシグニチャーは、次のとおりです。

```
public BulkResponse executeQuery(Query query,
        java.util.Map queryParams,
        IterativeQueryParams iterativeParams)
    throws JAXRException
```

結果セットのうち、各クエリーでそれぞれ異なるサブセットを要求するように、パラメータを指定できます。1 つのクエリーで結果セット全体を取得する代わりに、個々のクエリーで扱いやすい分量の結果セットを取得できます。

最大 100 個の結果を返すクエリー文字列があるとします。一連のパラメータを作成して、このクエリーで一度に 10 個の結果を返すようにすることができます。まず、`IterativeQueryParams` クラスのインスタンスを作成します。このクラスは、`org.freebxml.omar.common` パッケージで定義されています。このクラスの 2 つのフィールドは、配列の開始インデックスを表す `startIndex` と、返す結果の最大数を表す `maxResults` です。これらのフィールドの初期値はコンストラクタで指定します。

```
int maxResults = 10;
int startIndex = 0;
IterativeQueryParams iterativeQueryParams =
     new IterativeQueryParams(startIndex, maxResults);
```

`for` ループで各クエリを実行します。このループは、クエリーごとに `maxResults` の値だけ増加し、予測される最大結果数に達すると終了します。ループの繰り返しごとに `startIndex` フィールドを増分します。

```
for (int i = 0; i < 100; i += maxResults) {
    // Execute query with iterative query params
    Query query = dqm.createQuery(Query.QUERY_TYPE_SQL,
        queryStr);
     iterativeQueryParams.startIndex = i;
    BulkResponse br = dqm.executeQuery(query, null,
        iterativeQueryParams);
    Collection objects = br.getCollection();
    // retrieve individual objects ...
}
```

Service Registry では、クエリーの繰り返しの間にトランザクションの整合性や状態を維持する必要はありません。したがって、繰り返しの間に、完全な結果セットに対して新しいオブジェクトを追加することや既存のオブジェクトを削除することも可能です。そのため、繰り返しの間に結果セットの要素が抜けたり重複したりすることもあります。

繰り返し型クエリーの使用: 例

繰り返し型クエリーの使用例については、*<INSTALL>*`/registry/samples/query-iterative/src` ディレクトリにある `JAXRQueryIterative.java` を参照してください。このプログラムは、特定の文字列に一致する名前を持つすべてのレジストリオブジェクトを検索し、最初の 100 個を繰り返し処理します。

▼ JAXRQueryIterative サンプルを実行するには

手順 **1.** **<*INSTALL*>/registry/samples/query-iterative** ディレクトリに移動します。

2. 次のコマンドを入力して **string** 値を指定します。

```
asant run -Dname=string
```

ストアドクエリーの呼び出し

RegistryObjectImpl クラスを拡張した、実装に固有の AdhocQueryImpl クラスを使用すると、Service Registry に保存されているクエリーを呼び出すことができます。Service Registry には、呼び出し可能なデフォルトの AdhocQueryImpl オブジェクトがいくつかあります。最も役に立つのは FindAllMyObjects と GetCallersUser です。

- FindAllMyObjects は、41 ページの「発行したオブジェクトの検索」で説明されている QueryManager.getRegistryObjects() メソッドと同等の働きをします。
- GetCallersUser は、「私は誰ですか」という質問と同じです。このクエリーは、それを実行したクライアントに関連付けられている User オブジェクトを返します。呼び出し元が Service Registry にログインしていない場合、このクエリーは「Registry Guest」という名前のユーザーを返します。

ストアドクエリーを見つける最も簡単な方法は、一意の識別子でクエリーを検索することです。GetCallersUser クエリーには、このクエリー用に定義された標準的な定数がありますので98 ページの「ストアドクエリーに対する定数」を参照してください。標準的な定数を持たないクエリーは、一意の識別子の文字列値を使用して検索できます。

```
String queryId =
    CanonicalConstants.CANONICAL_QUERY_GetCallersUser;
AdhocQueryImpl aq =
    (AdhocQueryImpl) bqm.getRegistryObject(queryId);
```

次に、AdhocQuery に関連付けられているクエリー文字列を取得し、その文字列を使用してクエリーを作成して実行します。これには DeclarativeQueryManager のメソッドを使用します。

```
if (aq != null) {
        int qType = aq.getType();
    String qString = aq.toString();
    Query query = dqm.createQuery(qType, qString);

    BulkResponse br = dqm.executeQuery(query);
```

```
    Collection objects = br.getCollection();
    ...
```

ストアドクエリーの呼び出し: 例

ストアドクエリーの使用例については、<*INSTALL*>/registry/samples/query-stored/src ディレクトリにある JAXRQueryStored.java を参照してください。このサンプルは、ユーザーを認証し、そのユーザーのレジストリログイン名を返します。

▼ JAXRQueryStored サンプルを実行するには

手順

1. **<*INSTALL*>/registry/samples/query-stored** ディレクトリに移動します。
2. 次のコマンドを入力します。

```
asant run
```

レジストリ連携の検索

クエリーを実行している対象のレジストリが、1 つまたは複数のレジストリ連携 (15 ページの「レジストリとリポジトリについて」を参照) の一部である場合、そのレジストリメンバーとして含んでいるすべての連携内のすべてのレジストリに対して、あるいは 1 つの連携内のすべてのレジストリに対して、宣言型クエリーを実行できます。

レジストリをメンバーとして含んでいるすべての連携内のすべてのレジストリに対してクエリーを実行するには、実装に固有の setFederated メソッドを QueryImpl オブジェクトに対して呼び出します。このメソッドのシグニチャーは次のとおりです。

```
public void setFederated(boolean federated)
    throws JAXRException
```

このメソッドを次のように呼び出します。

```
QueryImpl query = (QueryImpl)
    dqm.createQuery(Query.QUERY_TYPE_SQL, qString);
query.setFederated(true);
```

レジストリが 1 つの連携だけのメンバーであることがわかっている場合、クエリーを実行する前に呼び出す必要があるのはこのメソッドだけです。

1 つの連携内のレジストリにクエリーを限定するには、実装に固有の setFederation メソッドも追加で呼び出す必要があります。このメソッドは、クエリーを実行する連携の一意識別子を引数として取ります。

```
public void setFederation(java.lang.String federationId)
    throws JAXRException
```

したがって、このメソッドを呼び出す前に、一意の識別子の値を取得する必要があります。そのためには、まず `BusinessQueryManagerImpl.findObjects` を呼び出して、連携を名前で特定します。このコードで、文字列 `"NameOfFederation"` を連携の実際の名前に置き換えます。

```
Collection namePatterns = new ArrayList();
namePatterns.add("NameOfFederation");

// Find objects with name NameOfFederation
BulkResponse response =
    bqm.findObjects("Federation", null, namePatterns,
         null, null, null, null);
```

次に、メンバーを 1 つだけ持っているコレクションを繰り返し処理して、キーの値を取得します。

```
String fedId = federation.getKey().getId();
```

最後に、クエリーを作成し、`setFederated` と `setFederation` を呼び出し、クエリーを実行します。

```
QueryImpl query = (QueryImpl)
    dqm.createQuery(Query.QUERY_TYPE_SQL, qString);
query.setFederated(true);
query.setFederation(fedId);
response = dqm.executeQuery(query);
```

連携クエリーの使用: 例

連携クエリーの使用例については、*<INSTALL>*`/registry/samples/query-federation/src` ディレクトリにある `JAXRQueryFederation.java` を参照してください。このサンプルは、見つかった各連携 (Service Registry に付属のデータベースに 1 つだけ含まれている) に対して、宣言型クエリーとストアドクエリーを実行します。

宣言型クエリーは、54 ページの「宣言型クエリーの使用: 例」で実行したものです。ストアドクエリーは `FindAllMyObjects` クエリーです。このサンプルでユーザーの認証は行われないので、クエリーを実行するユーザーは `RegistryGuest` となります。`RegistryGuest` ユーザーが所有するオブジェクトは、それ自体の 1 つだけです。したがって、`FindAllMyObjects` クエリーによって返される結果は `RegistryGuest` ユーザーの 1 つだけです。

▼ JAXRQueryFederation サンプルを実行するには

手順 **1.** *<INSTALL>***`/registry/samples/query-federation`** ディレクトリに移動します。

2. 次のコマンドを入力します。

```
asant run
```

第 4 章

Service Registry へのオブジェクトの発行

適切な権限を持っているクライアントは、Service Registry へのオブジェクトの送信、オブジェクトの変更、および削除ができます。クライアントは、`BusinessLifeCycleManager` インタフェースを使ってこれらのタスクを実行します。

通常、レジストリがクライアントにオブジェクトの変更または削除を許可するのは、そのクライアントのユーザーが、変更または削除しようとしているオブジェクトを最初に送信したユーザーと同じである場合だけです。アクセス制御ポリシーを使えば、オブジェクトを発行したり、それらのオブジェクトに対してアクションを実行したりする権限を、どのユーザーに対して与えるかを制御できます。

レジストリオブジェクトの発行には、次のタスクが含まれます。

- 62 ページの「Service Registry の認証」
- 63 ページの「オブジェクトの作成」
- 75 ページの「レジストリへのオブジェクトの保存」

オブジェクトの送信は複数の手順から成るタスクです。オブジェクトを作成し、それらの属性を設定したあとで、オブジェクトを保存します。オブジェクトを保存しないと、レジストリ内に表示されません。

分類や外部識別子などによってオブジェクトを検索する際に、検索で使用する分類やその他のオブジェクトの作成を思い出してください 。(たとえば、36 ページの「分類によるオブジェクトの検索」を参照)。ただし、このオブジェクトは保存されません。このオブジェクトは検索のためにのみ作成され、検索が終了すれば削除されます。オブジェクトを作成する際には Service Registry からの権限は必要ありませんが、オブジェクトを保存する際には権限が必要になります。

Service Registry の認証

Service Registry では証明書による認証が使用されるため、Service Registry にデータを送信するには証明書を持っている必要があります。また、レジストリにデータを送信できるユーザーを、Web コンソールのユーザー登録ウィザードを使って作成する必要もあります。詳細については、21 ページの「Service Registry へのアクセス」を参照してください。

データを送信するには、クライアントはまず、その証明書を一連の「資格」に追加して Service Registry に送信する必要があります。次のコードは、その実行方法を示したものです。資格を取得するには、次の必須の値を指定する必要があります。

- キーストアのパス。証明書の鍵の格納先となるファイル (通常は `keystore.jks`) へのフルパス
- キーストアのパスワード。通常は `ebxmlrr`
- ウィザードによる登録時に選択したユーザー名とパスワード

通常、これら 4 つの必須の値はリソースバンドルから取得し、メソッド内にコードの大部分をカプセル化します。

```
String keystorePath = "myKeystorePath";
String storepass = "myStorepass";
String alias = "myAlias";
String keypass = myKeypass");

Set credentials = new HashSet();
KeyStore keyStore = KeyStore.getInstance("JKS");
keyStore.load(new BufferedInputStream(
    new FileInputStream(keystorePath)),
     storepass.toCharArray());
X509Certificate cert = (X509Certificate)
     keyStore.getCertificate(alias);
PrivateKey privateKey =
     (PrivateKey) keyStore.getKey(alias, keypass.toCharArray());
credentials.add(new X500PrivateCredential(cert, privateKey,
    alias));
connection.setCredentials(credentials);
```

`setCredentials` メソッドが成功すると、ユーザーは Service Registry にログインした状態になり、オブジェクトを発行できるようになります。

Service Registry に対して認証するサンプルプログラムはすべて、このコードを含む `getCredentialsFromKeystore` という名前のメソッドを呼び出しています。このメソッドは、ファイル *<INSTALL>*`/registry/samples/common/src/RegistryCredentials.java` に定義されています。

オブジェクトの作成

クライアントは、オブジェクトを作成し、データを設定したあとで、そのオブジェクトを発行します。次のタイプの RegistryObject は、任意に作成および発行できます。

- AdhocQuery
- Association
- ClassificationScheme
- Concept
- ExternalLink
- ExtrinsicObject
- Federation
- Organization
- Person (実装に固有)
- RegistryPackage
- Service
- Subscription
- User

次のタイプの RegistryObject は、単独では発行できませんが、別のオブジェクトの一部として作成および保存することはできます。

- Classification (任意の RegistryObject)
- ExternalIdentifier (任意の RegistryObject)
- ServiceBinding (Service)
- Slot (任意の RegistryObject)
- SpecificationLink (ServiceBinding)

オブジェクトによっては、特殊なカテゴリに分類されるものもあります。それらを次に示します。

- AuditableEvent は、あるオブジェクトの状態が変化した場合に、Service Registry によって発行されます。
- Notification は、Subscription と AuditableEvent が一致した場合に、Service Registry によって発行されます。
- レジストリを発行できるのは、RegistryAdministrator ロールを持つユーザーだけです。

次の節では、最初に、すべてのレジストリオブジェクトの作成および保存に共通するタスクについて説明します。それから、特定のオブジェクト型に固有のタスクについて説明します。

- 64 ページの「オブジェクトの作成メソッドの使用」
- 64 ページの「オブジェクトへの名前と説明の追加」
- 65 ページの「オブジェクトの識別」
- 65 ページの「分類スキーマと Concept の作成と使用」

オブジェクトの作成メソッドの使用

LifeCycleManager インタフェースは、すべてのタイプの RegistryObject に対する作成メソッドをサポートしています。ただし、AuditableEvent と Notification は除きます。これらを作成できるのは Service Registry だけです。

さらに、LifeCycleManager.createObject ファクトリメソッドを使って特定のタイプのオブジェクトを作成することもできます。このメソッドは、LifeCycleManager インタフェースがサポートする static フィールドのいずれかを含む String 引数を取ります。次のコード内の blcm は、BusinessLifeCycleManager オブジェクトです。

```
Organization org = (Organization)
    blcm.createObject(blcm.ORGANIZATION);
```

オブジェクトに固有の作成メソッドは通常、オブジェクトのいくつかの属性を設定する 1 つまたは複数のパラメータを取ります。たとえば、createOrganization メソッドは組織名を設定します。

```
Organization org = blcm.createOrganization("MyOrgName");
```

これに対し、createExtrinsicObject メソッドは通常、付帯オブジェクトに対するリポジトリ項目を設定する DataHandler 引数を取ります。

オブジェクトへの名前と説明の追加

すべてのオブジェクトで、設定メソッドを呼び出すことで名前属性と説明属性を設定できます。これらの属性のタイプは、InternationalString です。InternationalString には一連の LocalizedString オブジェクトが含まれていますが、これらは、ユーザーが 1 つまたは複数のロケールで名前や説明を表示できるようにするためのものです。デフォルトで、InternationalString 値はデフォルトのロケールを使用します。

たとえば、次のコードでは、地域対応された 2 つの文字列を使用する説明を作成します。1 つの文字列はデフォルトロケールの言語です。もう一方の文字列はフランス語 (カナダ) です。

```
InternationalString is =
    blcm.createInternationalString("What We Do"));
Locale loc = new Locale("fr", "CA");
```

```
LocalizedString ls = blcm.createLocalizedString(loc,
     "ce que nous faisons");
is.addLocalizedString(ls);
org.setDescription(is);
```

オブジェクトの識別

32 ページの「一意の識別子によるオブジェクトの検索」で説明したように、Service Registry 内のすべてのオブジェクトには、一意の識別子と論理識別子の 2 つの識別子があります。オブジェクト作成時にこれらの識別子を設定しなかった場合は、レジストリによって一意の値が生成され、その値が一意の識別子と論理識別子の両方に割り当てられます。

新しいバージョンのオブジェクトが作成される場合は常に、論理識別子は元の識別子と同じですが、一意の識別子については、元の識別子にコロンとバージョン番号が付加された新しい一意の識別子が Service Registry によって生成されます。詳細については、52 ページの「オブジェクトのバージョンの取得」および77 ページの「オブジェクト間の関係の作成: 関連付け」を参照してください。

独自の識別スキーマを使用する場合、オブジェクト識別子を設定するための API メソッドを使用できます。

JAXR API では、一意の識別子は `Key` オブジェクトと呼ばれます。`LifeCycleManager.createKey` メソッドを使用して、`String` オブジェクトから一意の識別子を作成できます。その後、`RegistryObject.setKey` メソッドを使用してキーを設定できます。

論理識別子は `lid` と呼ばれます。Service Registry の JAXR プロバイダには、この識別子を設定するための、実装に固有の `RegistryObjectImpl.setLid`メソッドが用意されています。このメソッドも `String` 引数を取ります。このメソッドのシグニチャーは次のとおりです。

```
public void setLid(java.lang.String lid)
    throws JAXRException
```

指定する識別子はすべて、有効かつグローバルで一意な URN (Uniform Resource Name) である必要があります。JAXR API がオブジェクトのキーを生成するとき、そのキーの形式は DCE 128 UUID (Universal Unique IDentifier) になります。

分類スキーマと Concept の作成と使用

レジストリオブジェクトを分類するために、独自の分類スキーマと Concept の階層を作成できます。階層を作成するには、次の手順に従います。

1. `LifeCycleManager.createClassificationScheme` メソッドを使って分類スキーマを作成します。
2. `LifeCycleManager.createConcept` メソッドを使って Concept を作成します。

3. ClassificationScheme.addChildConcept メソッドを使って分類スキーマに Concept を追加します。
4. より深い階層を作成する場合は、Concept.addChildConcept メソッドを使って Concept に子 Concept を追加します。
5. 分類スキーマを保存します。

LifeCycleManager.createClassificationScheme メソッドにはいくつかの形式があります。名前と説明の 2 つの引数を、String 値または InternationalString 値として指定できます。たとえば、図書館における本の収納方法を記述する分類スキーマを作成する場合、次のようなコードを使用できます。

```
ClassificationScheme cs =
     blcm.createClassificationScheme("LibraryFloors",
         "Scheme for Shelving Books");
```

別の形式の createClassificationScheme メソッドは、1 つの引数 Concept を取り、Concept を ClassificationScheme に変換します。

createConcept メソッドは 3 つの引数を取ります。親、名前、および値です。親は、createClassificationScheme または別の Concept のいずれかです。値は指定するが名前は指定しない、ということも可能です。

次のコードでは、図書館のフロアの名前が格納された静的な String 配列を使用して、各フロアに対応する Concept を作成します。続いて、その Concept を分類スキーマに追加します。

```
for (int i = 0; i < floors.length; i++) {
    Concept con = blcm.createConcept(cs, floors[i], floors[i]);
    cs.addChildConcept(con);
    ...
```

各 Concept について、新しい Concept をさらに作成して Concept.addChildConcept を呼び出し、階層レベルを 1 つ増やすこともできます。分類スキーマを保存すると、その Concept 階層の全体も保存されます。

分類スキーマの作成と表示: 例

分類スキーマの作成のサンプルについては、*<INSTALL>*/registry/samples/classification-schemes/src ディレクトリにある JAXRPublishScheme.java を参照してください。このサンプルは、LibraryFloors という名前の分類スキーマを作成するとともに、図書館の各フロアとそこで見つけることのできる主題領域を含む Concept 階層を作成します。

Concept 階層を表示するには、同じディレクトリ内のプログラム JAXRSearchScheme.java を使用します。このサンプルは、ユーザーが指定した任意の分類スキーマに対する Concept 階層を表示します。

分類スキーマと Concept を削除するには、同じディレクトリ内のプログラム JAXRDeleteScheme.java を使用します。

▼ JAXRPublishScheme サンプルを実行するには

手順 1. **<*INSTALL*>/registry/samples/classification-schemes** ディレクトリに移動します。

2. 次のコマンドを入力します。

```
asant pub-scheme
```

▼ JAXRSearchScheme サンプルを実行するには

手順 1. **<*INSTALL*>/registry/samples/classification-schemes** ディレクトリに移動します。

2. 次のコマンドを入力します。

```
asant search-scheme -Dname=LibraryFloors
```

▼ JAXRDeleteScheme サンプルを実行するには

手順 1. **<*INSTALL*>/registry/samples/classification-schemes** ディレクトリに移動します。

2. 次のコマンドを入力します。

```
asant del-scheme -Dname=LibraryFloors
```

オブジェクトへの分類の追加

オブジェクトは、1 つまたは複数の分類スキーマ (分類方式) に基づく分類を、1 つまたは複数持つことができます。オブジェクトの分類を確立する場合、クライアントはまず、使用する分類方式を特定します。続いて、クライアントは、分類スキーマと分類スキーマ内の Concept (分類方式の要素) を使って分類を作成します。

Concept の階層を備えた新しい分類スキーマを作成する方法については、77 ページの「オブジェクト間の関係の作成: 関連付け」を参照してください。Concept 階層を持つ分類スキーマは「内部分類スキーマ」と呼ばれます。

既存の分類スキーマを使用する分類を追加するには、通常は `BusinessQueryManager.findClassificationSchemeByName` メソッドを呼び出します。このメソッドは 2 つの引数を取ります。1 つは `FindQualifier` オブジェクトの `Collection`、もう 1 つは名前パターンを指定する `String` です。このメソッドから複数の結果が返されるとエラーになるため、検索は的確に定義する必要があります。たとえば、次のコードでは、`AssociationType` という名前の分類スキーマを検索しています。

```
String schemeName = "AssociationType";
ClassificationScheme cScheme =
     bqm.findClassificationSchemeByName(null, schemeName);
```

分類スキーマを特定したら、3 つの引数を指定して `LifeCycleManager.createClassification` メソッドを呼び出します。引数は、分類スキーマ、Concept の名前、および Concept の値です。

```
Classification classification =
     blcm.createClassification(cScheme, "Extends", "Extends");
```

もう 1 つの方法として、`BusinessQueryManager.findConcepts` を呼び出すか、`Concept` 引数を指定して `BusinessQueryManagerImpl.findObjects` を呼び出し、使用する Concept を検索してから、Concept を唯一の引数として取る別の形式の `createClassification` を呼び出す、という方法があります。

```
Classification classification =
     blcm.createClassification(concept);
```

分類の作成が完了したら、`RegistryObject.addClassification` を呼び出して、分類をオブジェクトに追加します。

```
object.addClassification(classification);
```

複数の分類を追加するには、`Collection` を作成し、`Collection` に分類を追加したあと、`RegistryObject.addClassifications` を呼び出して `Collection` をオブジェクトに追加します。

分類の追加: 例

分類をオブジェクトに追加する例については、*<INSTALL>*`/registry/samples/publish-object/src` ディレクトリにある `JAXRPublishObject.java` を参照してください。このサンプルは、組織を 1 つ作成し、それにいくつかのオブジェクトを追加します。

▼ JAXRPublishObject サンプルを実行するには

手順

1. **<*INSTALL*>/registry/samples/publish-object** ディレクトリに移動します。
2. 次のコマンドを入力します。

   ```
   asant run
   ```

オブジェクトへの外部識別子の追加

オブジェクトに外部識別子を追加するには、次の手順に従います。

1. 使用する分類スキーマを検索または作成します。

2. その分類スキーマを使って外部識別子を作成します。

外部識別子を作成するには、Concept 階層を持たない分類スキーマである「外部分類スキーマ」を使用します。外部識別子に名前と値を指定します。

Service Registry に付属しているデータベースには、外部分類スキーマは含まれていません。外部分類スキーマを使用するには、次のようなコードを使って事前にそれらを作成する必要があります。

```
ClassificationScheme extScheme =
     blcm.createClassificationScheme("NASDAQ",
         "OTC Stock Exchange");
```

通常、既存の分類スキーマを検索するには、67 ページの「オブジェクトへの分類の追加」で説明したように `BusinessQueryManager.findClassificationSchemeByName` メソッドを呼び出します。

たとえば、次のコードでは、前の手順で作成した外部分類スキーマを検索しています。

```
ClassificationScheme extScheme =
     bqm.findClassificationSchemeByName(null,
         "NASDAQ");
```

外部識別子を追加するには、`LifeCycleManager.createExternalIdentifier` メソッドを呼び出します。このメソッドは分類スキーマ、外部識別子の名前、外部識別子の値の 3 つの引数を取ります。続いて、その外部識別子をオブジェクトに追加します。

```
ExternalIdentifier extId =
     blcm.createExternalIdentifier(extScheme, "Sun",
         "SUNW);
object.addExternalIdentifier(extId);
```

68 ページの「分類の追加: 例」で説明したサンプル `<INSTALL>/registry/samples/publish-object/src/JAXRPublishObject.java` は、オブジェクトへの外部識別子の追加も行います。

オブジェクトへの外部リンクの追加

オブジェクトに外部リンクを追加するには、`LifeCycleManager.createExternalLink` メソッドを呼び出します。このメソッドはリンクの URI とリンクの説明の 2 つの引数を取ります。続いて、その外部リンクをオブジェクトに追加します。

```
String eiURI = "http://java.sun.com/";
String eiDescription = "Java Technology";
ExternalLink extLink =
     blcm.createExternalLink(eiURI, eiDescription);
object.addExternalLink(extLink);
```

この URI は有効な URI でなければならず、JAXR プロバイダがその有効性をチェックします。ファイアウォールの外側へのリンクを指定する場合は、JAXR で URI の有効性を確認できるように、プログラムの実行時にシステムプロパティー `http.proxyHost` および `http.proxyPort` を指定する必要があります。

現在アクティブでないリンクを指定する場合などに URI の検証を無効にするには、リンクを作成する前に `ExternalLink.setValidateURI` メソッドを呼び出します。

```
extLink.setValidateURI(false);
```

68 ページの「分類の追加: 例」で説明したサンプル *<INSTALL>*`/registry/samples/publish-object/src/JAXRPublishObject.java` は、オブジェクトへの外部リンクの追加も行います。このサンプルの `build.xml` ファイルには、システムプロパティー `http.proxyHost` および `http.proxyPort` が指定されています。

オブジェクトへのスロットの追加

スロットは任意の属性であるため、API を使用すると、最大限の柔軟性をもってスロットを作成できます。ユーザーは、1 つの名前、1 つまたは複数の値、および 1 つの型を設定できます。名前と型は `String` オブジェクトです。これらの値は `String` オブジェクトの `Collection` として格納されますが、`LifeCycleManager.createSlot` メソッドには、単一の `String` 値を指定できる形式も用意されています。たとえば、次のコードでは、`String` 値を使ってスロットを作成したあとで、そのスロットをオブジェクトに追加しています。

```
String slotName = "Branch";
String slotValue = "Paris";
String slotType = "City";
Slot slot = blcm.createSlot(slotName, slotValue, slotType);
org.addSlot(slot);
```

68 ページの「分類の追加: 例」で説明したサンプル *<INSTALL>*`/registry/samples/publish-object/src/JAXRPublishObject.java` は、オブジェクトへのスロットの追加も行います。

組織の作成

`Organization` オブジェクトはおそらく、最も複雑なレジストリオブジェクトです。このオブジェクトには通常、すべてのオブジェクトに共通する属性のほかに、次の属性が含まれます。

- 1 つまたは複数の `PostalAddress` オブジェクト。
- 1 つまたは複数の `TelephoneNumber` オブジェクト。
- 1 つの `PrimaryContact` オブジェクト。これは `User` オブジェクトです。`User` オブジェクトには通常、1 つの `PersonName` オブジェクトが含まれるほか、`TelephoneNumber`、`EmailAddress`、および `PostalAddress` オブジェクトのコレクションも含まれます。

- 1 つまたは複数の Service オブジェクトとそれらに関連付けられた ServiceBinding オブジェクト。

組織は 1 つまたは複数の子組織を持つこともでき、それらの子組織もまた子を持つことができます。こうして、組織の階層が形成されます。

次のコードでは、組織を 1 つ作成し、その名前、説明、住所、および電話番号を指定しています。

```
// Create organization name and description
Organization org =
     blcm.createOrganization("The ebXML Coffee Break");
InternationalString is =
    blcm.createInternationalString("Purveyor of " +
        "the finest coffees. Established 1905");
org.setDescription(is);

// create postal address for organization
String streetNumber = "99";
String street = "Imaginary Ave. Suite 33";
String city = "Imaginary City";
String state = "NY");
String country = "USA");
String postalCode = "00000";
String type = "Type US";
PostalAddress postAddr =
     blcm.createPostalAddress(streetNumber, street, city, state,
        country, postalCode, type);
org.setPostalAddress(postAddr);

// create telephone number for organization
TelephoneNumber tNum = blcm.createTelephoneNumber();
tNum.setCountryCode("1");
tNum.setAreaCode("100");
tNum.setNumber("100-1000");
tNum.setType("OfficePhone");
Collection tNums = new ArrayList();
tNums.add(tNum);
org.setTelephoneNumbers(tNums);
```

電話番号のタイプは、PhoneType 分類スキーマに含まれる Concept の値です。"OfficePhone"、"MobilePhone" 、"HomePhone"、"FAX"、または "Beeper" のいずれかです。

組織の階層を作成するには、Organization.addChildOrganization メソッドを使ってある組織を別の組織に追加するか、Organization.addChildOrganizations メソッドを使って組織の Collection を別の組織に追加します。

組織の作成: 例

組織の作成方法のサンプルについては、*<INSTALL>*/registry/samples/organizations/src ディレクトリにある JAXRPublishOrg.java と JAXRPublishOrgNoPC.java を参照してください。

JAXRPublishOrg サンプルは、1 つの組織とその主担当者、および 1 つのサービスとそのサービスバインディングを作成します。サンプルでは組織、ユーザー、およびサービスに対する一意の識別子が表示され、ユーザーはあとでオブジェクトを削除する際にそれらの識別子を使うことができます。このサンプルは、組織の主担当者として架空の User を作成します。

もう 1 つのサンプル JAXRPublishOrgNoPC.java は、組織の主担当者を設定しません。この場合、主担当者はデフォルトで、プログラム実行時に認証された User になります。

▼ JAXRPublishOrg および JAXRPublishOrgNoPC サンプルを実行するには

手順 **1.** **<*INSTALL* >/registry/samples/organizations** ディレクトリに移動します。

2. 次のコマンドを入力します。

```
asant pub-org
asant pub-org-nopc
```

組織階層の作成と取得: 例

組織階層を発行および取得する方法のサンプルについては、<*INSTALL*>/registry/samples/organizations/src ディレクトリにある JAXRPublishOrgFamily.java および JAXRSearchOrgFamily.java を参照してください。

▼ JAXRPublishOrgFamily および JAXRSearchOrgFamily サンプルを実行するには

手順 **1.** **<*INSTALL*>/registry/samples/organizations** ディレクトリに移動します。

2. 組織を発行するには、次のコマンドを入力します。

```
asant pub-fam
```

3. 発行した組織を取得するには、次のコマンドを入力します。

```
asant search-fam
```

ユーザーの作成

主担当者を指定せずに組織を作成する場合、デフォルトの主担当者は、その組織を作成した User オブジェクト、つまり、Service Registry への接続確立時に設定した資格の所有者であるユーザーになります。もちろん、それとは異なるユーザーを主担当者として指定することもできます。User もまた、複雑なレジストリオブジェクトです。これには通常、すべてのオブジェクトに共通する属性のほかに、次の属性が含まれます。

- 1 つの PersonName オブジェクト
- 1 つまたは複数の PostalAddress オブジェクト
- 1 つまたは複数の TelephoneNumber オブジェクト
- 1 つまたは複数の EmailAddress オブジェクト
- ユーザーのホームページを表す 1 つまたは複数の URL オブジェクト

次のコードでは、User を作成し、その User を組織の主担当者として設定しています。この User には、電話番号と電子メールアドレスは設定されていますが、住所は設定されていません。

```
// Create primary contact, set name
User primaryContact = blcm.createUser();
String userId = primaryContact.getKey().getId();
System.out.println("User URN is " + userId);
PersonName pName =
     blcm.createPersonName("Jane", "M.", "Doe");
primaryContact.setPersonName(pName);

// Set primary contact phone number
TelephoneNumber pctNum = blcm.createTelephoneNumber();
pctNum.setCountryCode("1");
pctNum.setAreaCode("100");
pctNum.setNumber("100-1001");
pctNum.setType("MobilePhone");
Collection phoneNums = new ArrayList();
phoneNums.add(pctNum);
primaryContact.setTelephoneNumbers(phoneNums);

// Set primary contact email address
EmailAddress emailAddress =
blcm.createEmailAddress("jane.doe@TheCoffeeBreak.com");
emailAddress.setType("OfficeEmail"));
Collection emailAddresses = new ArrayList();
emailAddresses.add(emailAddress);
primaryContact.setEmailAddresses(emailAddresses);

URL pcUrl = new URL((bundle.getString("person.url"));
primaryContact.setUrl(pcUrl);

// Set primary contact for organization
org.setPrimaryContact(primaryContact);
```

主担当者の電話番号のタイプは、PhoneType 分類スキーマに含まれる Concept の値です。値は "OfficePhone"、"MobilePhone"、"HomePhone"、 "FAX"、または "Beeper" のいずれかです。主担当者の電子メールアドレスのタイプは、EmailType 分類スキーマに含まれる Concept の値です。値は "OfficeEmail" または "HomeEmail" です。

サービスとサービスバインディングの作成

ほとんどの組織は、サービスを提供するために自らをレジストリに発行します。このため、JAXR には、サービスやサービスバインディングを組織に追加する機能が用意されています。

また、どの組織にも関連付けられていないサービスを作成することもできます。

Service オブジェクトには Organization オブジェクトと同じく、名前、説明、およびサービス登録時にレジストリによって生成される一意のキーがあります。Service オブジェクトは分類を持つこともできます。

サービスには通常、すべてのオブジェクトに共通する属性のほかに、サービスへのアクセス方法に関する情報を提供する「サービスバインディング」があります。ServiceBinding オブジェクトには通常、説明、アクセス URI、および仕様リンクがあります。仕様リンクは、サービスバインディングと技術仕様をリンクします。技術仕様には、サービスバインディングを用いてサービスを使用する方法が記述されています。

次のコードは、サービスのコレクションを作成し、サービスにサービスバインディングを追加したあと、サービスを組織に追加する方法を示したものです。ここでは、アクセス URI は指定されていますが、仕様リンクは指定されていません。このアクセス URI は実際には存在しませんが、JAXR はデフォルトで、発行されたすべての URI の有効性をチェックします。このため、このバインディングでは validateURI 属性が false に設定されています。

```
// Create services and service
Collection services = new ArrayList();
Service service = blcm.createService("My Service Name");
InternationalString is =
     blcm.createInternationalString("My Service Description");
service.setDescription(is);

// Create service bindings
Collection serviceBindings = new ArrayList();
ServiceBinding binding =
     blcm.createServiceBinding();
is = blcm.createInternationalString("My Service Binding " +
     "Name"));
binding.setName(is);
is = blcm.createInternationalString("My Service Binding " +
    "Description");
binding.setDescription(is);
// allow us to publish a fictitious URI without an error
```

```
binding.setValidateURI(false);
binding.setAccessURI("http://TheCoffeeBreak.com:8080/sb/");
...
serviceBindings.add(binding);

// Add service bindings to service
service.addServiceBindings(serviceBindings);

// Add service to services, then add services to organization
services.add(service);
org.addServices(services);
```

サービスバインディングには通常、サービスへのアクセス方法を記述した技術仕様があります。そのような仕様の一例として、WSDL 文書が挙げられます。サービスの仕様が WSDL 文書である場合、その格納場所を発行するには、`ExtrinsicObject` を参照する `SpecificationLink` オブジェクトを作成します。詳細は、80 ページの「リポジトリへの項目の格納」を参照してください。

この機構は、仕様の場所を UDDI レジストリに発行する方法とは異なります。UDDI レジストリの場合、`Concept` オブジェクトを作成したあと、その `Concept` オブジェクトに、WSDL 文書の URL を `ExternalLink` オブジェクトとして追加します。

レジストリへのオブジェクトの保存

オブジェクトを作成し、その属性の設定が完了したら、そのオブジェクトを Service Registry に発行します。それには、`LifeCycleManager.saveObjects` メソッドを呼び出すか、`BusinessLifeCycleManager.saveOrganizations` や `BusinessLifeCycleManager.saveServices` などの、オブジェクトに固有の保存メソッドを呼び出します。単一のオブジェクトではなく、常にオブジェクトのコレクションを発行します。保存メソッドは、保存されたオブジェクトのキー (すなわち、一意の識別子) を含む `BulkResponse` オブジェクトを返します。次のコードでは、ある組織を保存したあと、そのキーを取得しています。

```
// Add organization and submit to registry
// Retrieve key if successful
Collection orgs = new ArrayList();
orgs.add(org);
BulkResponse response = blcm.saveOrganizations(orgs);
Collection exceptions = response.getExceptions();
if (exceptions == null) {
    System.out.println("Organization saved");

    Collection keys = response.getCollection();
    Iterator keyIter = keys.iterator();
    if (keyIter.hasNext()) {
        javax.xml.registry.infomodel.Key orgKey =
              (javax.xml.registry.infomodel.Key) keyIter.next();
        String id = orgKey.getId();
```

```
            System.out.println("Organization key is " + id);
        }
}
```

オブジェクトのいずれかがすでに存在しており、そのデータの一部が更新されている場合、保存メソッドはそのデータを更新して置換します。通常は、これによって、オブジェクトの新しいバージョンが作成されます (84 ページの「レジストリ内のオブジェクトの状態の変更」を参照)。

第 5 章

Service Registry 内のオブジェクトの管理

Service Registry へのオブジェクトの発行が完了すると、それらのオブジェクトに対して操作を実行できます。この章では、それらの操作について説明します。

オブジェクト間の関係の作成: 関連付け

`Association` オブジェクトを作成し、それを使って任意の 2 つのオブジェクト間の関係を指定することができます。ebXML 仕様で規定されている `AssociationType` 分類スキーマには、`Association` 作成時に使用可能な標準的な概念が、数多く含まれています。`AssociationType` 分類スキーマ内に独自の概念を作成することもできます。

標準的な関連付けタイプは、次のとおりです。

- `AccessControlPolicyFor`
- `AffiliatedWith` (サブ概念 `EmployeeOf` と `MemberOf` を持つ)
- `Contains`
- `ContentManagementServiceFor`
- `EquivalentTo`
- `Extends`
- `ExternallyLinks`
- `HasFederationMember`

- HasMember
- Implements
- InstanceOf
- InvocationControlFileFor (サブ概念 CatalogingControlFileFor と ValidationControlFileFor を持つ)
- OffersService
- OwnerOf
- RelatedTo
- Replaces
- ResponsibleFor
- SubmitterOf
- Supersedes
- Uses

レジストリは、これらの関連付けタイプの一部を自動的に使用します。たとえば、Service を Organization に追加する場合、レジストリは、その Organization をソースに、Service をターゲットに持つ OffersService 関連付けを作成します。

関連付けには方向があります。各 Association オブジェクトは 1 つのソースと 1 つのターゲットを持ちます。2 つのオブジェクト間の関連付けを確立する操作は、次の 3 つの手順から成ります。

1. 使用する AssociationType 概念を検索するか、作成します。
2. LifeCycleManager.createAssociation メソッドを使って関連付けを作成します。このメソッドは 2 つの引数を取ります。ターゲットオブジェクトと関係を識別する概念の 2 つです。
3. RegistryObject.addAssociation メソッドを使って関連付けをソースオブジェクトに追加します。

たとえば、2 つのオブジェクト obj1 と obj2 があり、それらの間に RelatedTo の関係を確立するとします。この関係の場合、どちらのオブジェクトをソースにし、どちらのオブジェクトをターゲットにするかは、自由に決めてかまいません。まず、RelatedTo 概念を検索します。

```
// Find RelatedTo concept for Association
String concString =
    CanonicalConstants.CANONICAL_ASSOCIATION_TYPE_ID_RelatedTo;
Concept relConcept = (Concept) bqm.getRegistryObject(concString);
```

obj2 をターゲットとして指定して関連付けを作成します。

```
Association relAssoc =
     blcm.createAssociation(obj2, relConcept);
```

その関連付けをソースオブジェクト obj1 に追加します。

```
obj1.addAssociation(relAssoc);
```

最後に、その関連付けを保存します。

```
Collection associations = new ArrayList();
associations.add(relAssoc1);
BulkResponse response = blcm.saveObjects(associations);
```

関連付けには区域内と区域外の 2 種類があります。ユーザーがソースオブジェクトとターゲットオブジェクトの両方を所有している場合、「区域内関連付け」が作成されます。これらのオブジェクトの少なくとも一方を所有していない場合は、「区域外関連付け」を作成します。オブジェクトの所有者は、アクセス制御ポリシーを使用して、そのオブジェクトをソースまたはターゲットに持つ区域外関連付けを作成する権限を制限できます。

関連付けの作成: 例

関連付けの作成の例については、*<INSTALL>*`/registry/samples/publish-association/src/` ディレクトリにある `JAXRPublishAssociation.java` を参照してください。このサンプルは、指定した一意の識別子を持つ任意の 2 つのオブジェクト間の `RelatedTo` 関連付けを作成します。たとえば、72 ページの「組織階層の作成と取得: 例」で作成された 2 つの子組織を指定することができます。

▼ JAXRPublishAssociation サンプルを実行するには

手順

1. **<*INSTALL*>/registry/samples/organizations** ディレクトリに移動します。
2. 次のコマンドを実行して組織階層を取得します。

   ```
   asant search-fam
   ```

   2 つの子組織のキーとなる ID 文字列に注意してください。
3. **<*INSTALL*>/registry/samples/publish-association** ディレクトリに移動します。
4. 次のコマンドを入力します。

   ```
   asant run -Did1=string1 -Did2=string2
   ```

   string1 と *string2* を 2 つの子組織の ID 文字列に置き換えてください。

   関連付けが区域内、区域外のいずれになるかは、2 つのオブジェクトの所有者によって決まります。この場合の関連付けは、区域内になります。

リポジトリへの項目の格納

15 ページの「レジストリとリポジトリについて」で説明したように、レジストリには、電子コンテンツの格納が可能なリポジトリが含まれています。リポジトリに格納するすべての項目について、まず ExtrinsicObject を作成する必要があります。ExtrinsicObject をレジストリに保存すると、関連付けられたリポジトリ項目も保存されます。

付帯オブジェクトの作成

ExtrinsicObject を作成するには、まず、リポジトリ項目に対して javax.activation.DataHandler オブジェクトを作成する必要があります。LifeCycleManager.createExtrinsicObject メソッドは DataHandler 引数を取ります。

注 – 引数を取らない、実装に固有の形式の createExtrinsicObject メソッドを使用することもできます。この形式を使用する場合、DataHandler オブジェクトをあとで作成し、ExtrinsicObject.setRepositoryItem メソッドを使ってそのリポジトリ項目を指定することができます。また、関連付けられたリポジトリ項目を持たない付帯オブジェクトを作成することもできます。

たとえば、リポジトリ内にファイルを格納するには、まず、java.io.File オブジェクトを作成します。その File オブジェクトから javax.activation.FileDataSource オブジェクトを作成し、それを使って DataHandler オブジェクトのインスタンスを生成します。

```
String filename = "./MyFile.xml";
File repositoryItemFile = new File(filename);
DataHandler repositoryItem =
     new DataHandler(new FileDataSource(repositoryItemFile));
```

次に、その DataHandler を引数に指定して createExtrinsicObject を呼び出します。

```
ExtrinsicObject eo =
     blcm.createExtrinsicObject(repositoryItem);
eo.setName("My Graphics File");
```

オブジェクトにアクセスできるように、オブジェクトの MIME タイプを指定します。デフォルトの MIME タイプは application/octet-stream です。ファイルが XML ファイルである場合、次のように設定します。

```
eo.setMimeType("text/xml");
```

最後に、ExtrinsicObject を Service Registry の適切な領域に格納するために、実装に固有の ExtrinsicObjectImpl.setObjectType メソッドを呼び出します。このメソッドのシグニチャーは、次のとおりです。

```
public void setObjectType(Concept objectType)
    throws JAXRException
```

特定のタイプのファイルに対する適切な概念を見つけるもっとも簡単な方法は、Web コンソールの探索機能を使用することです。ObjectType 分類スキーマの下で、さまざまなタイプの ExtrinsicObject 概念の中から探します。目的の概念の ID を getRegistryObject の引数として指定し、その概念を setObjectType の引数として指定します。

```
String conceptId =
"urn:oasis:names:tc:ebxml-regrep:ObjectType:RegistryObject:ExtrinsicObject:XML";
Concept objectTypeConcept =
     (Concept) bqm.getRegistryObject(conceptId);
((ExtrinsicObjectImpl)eo).setObjectType(objectTypeConcept);
```

最後に、ExtrinsicObject をレジストリに保存します。

```
Collection extobjs = new ArrayList();
extobjs.add(eo);
BulkResponse response = blcm.saveObjects(extobjs);
```

ExtrinsicObject にはメタデータが含まれており、ファイルのコピーがリポジトリ内に格納されます。

レジストリに格納するファイルの種類の概念が存在しない場合には、その概念を自分で作成し、保存できます。

付帯オブジェクトの作成: 例

付帯オブジェクトの作成方法の例については、<*INSTALL*>/registry/samples/publish-extrinsic/src ディレクトリにある JAXRPublishExtrinsicObject.java を参照してください。このサンプルは、1 つの XML ファイル (独自の build.xml ファイル) を Service Registry に発行します。

▼ JAXRPublishExtrinsicObject サンプルを実行するには

手順

1. **<*INSTALL*>/registry/samples/publish-extrinsic** ディレクトリに移動します。
2. 次のコマンドを入力します。

   ```
   asant run
   ```

仕様リンクでの付帯オブジェクトの使用

ExtrinsicObject は単独でも発行できますが、ServiceBinding オブジェクトに対する SpecificationLink の specificationObject 属性として使用する目的で ExtrinsicObject を作成することも、一般的に行われます。74 ページの「サービスとサービスバインディングの作成」を参照してください。ExtrinsicObject は通常、WSDL ファイルを参照します。

1. SpecificationLink オブジェクトを作成します。
2. WSDL 文書をリポジトリ内に格納し、それを参照する ExtrinsicObject を作成します。付帯オブジェクトのオブジェクト型を WSDL に、MIME タイプを text/xml に、それぞれ設定します。
3. この付帯オブジェクトを、SpecificationLink オブジェクトの specificationObject 属性として指定します。
4. この SpecificationLink オブジェクトを ServiceBinding オブジェクトに追加します。
5. この ServiceBinding オブジェクトを Service オブジェクトに追加します。
6. Service オブジェクトを保存します。

Service と ServiceBinding を作成してから、SpecificationLink を作成します。

```
SpecificationLink specLink = blcm.createSpecificationLink();
specLink.setName("Spec Link Name");
specLink.setDescription("Spec Link Description");
```

80 ページの「付帯オブジェクトの作成」で説明した方法で ExtrinsicObject を作成します。WSDL 概念の ID と text/xml MIME タイプを使用します。

```
String conceptId =
"urn:oasis:names:tc:ebxml-regrep:ObjectType:RegistryObject:ExtrinsicObject:WSDL";
Concept objectTypeConcept =
     (Concept) bqm.getRegistryObject(conceptId);
((ExtrinsicObjectImpl)eo).setObjectType(objectTypeConcept);
eo.setMimeType("text/xml");
```

ExtrinsicObject を、SpecificationLink の仕様オブジェクトとして設定します。

```
specLink.setSpecificationObject(eo);
```

SpecificationLink を ServiceBinding に追加してからオブジェクトをそのコレクションに追加し、サービスを保存します。

```
binding.addSpecificationLink(specLink);
serviceBindings.add(binding);
...
```

Service Registry からサービスを削除すると、そのサービスバインディングと仕様リンクも削除されます。ただし、仕様リンクに関連付けられていた付帯オブジェクトは削除されません。

仕様リンクで使用する付帯オブジェクトの作成: 例

仕様リンクで使用する付帯オブジェクトの作成方法の例については、 <*INSTALL*>/registry/samples/publish-service/src ディレクトリにある JAXRPublishService.java を参照してください。このサンプルは、レジストリに WSDL ファイルを発行します。

▼ JAXRPublishService サンプルを実行する方法

手順 **1.** **<*INSTALL*>/registry/samples/publish-service** ディレクトリに移動します。

2. 次のコマンドを入力します。

```
asant run
```

レジストリパッケージ内へのオブジェクトのグループ化

レジストリパッケージを使えば、論理的に関連付けられている多数のレジストリオブジェクトをグループ化できます。その際、個々のメンバーオブジェクトが異なる所有者に属していてもかまいません。RegistryPackage はファイルシステムのディレクトリまたはフォルダに相当し、その中のレジストリオブジェクトはディレクトリまたはフォルダ内のファイルに相当します。

RegistryPackage オブジェクトを作成するには、LifeCycleManager.createRegistryPackage メソッドを呼び出します。このメソッドは String 引数または InternationalString 引数を取ります。それから、RegistryPackage.addRegistryObject、RegistryPackage.addRegistryObjects のいずれかのメソッドを呼び出してオブジェクトをパッケージに追加します。

たとえば、"SunPackage" という名前の RegistryPackage オブジェクトを作成できます。

```
RegistryPackage pkg =
     blcm.createRegistryPackage("SunPackage");
```

それから、文字列 "Sun" を名前に含むすべてのオブジェクトを検索したあと、その結果に対して繰り返し処理を行い、各オブジェクトをパッケージに追加することができます。

```
pkg.addRegistryObject(object);
```

パッケージの一般的な用途は、一連の付帯オブジェクトをグループ化することです。レジストリ管理者は、Service Registry にファイルシステムをロードし、ディレクトリをレジストリパッケージとして、またファイルをパッケージの内容として、それぞれ格納できます。詳細については、『管理ガイド』を参照してください。

レジストリパッケージ内へのオブジェクトのグループ化: 例

レジストリパッケージの使用例については、*<INSTALL>*/registry/samples/packages/src ディレクトリにある JAXRPublishPackage.java および JAXRSearchPackage.java を参照してください。最初のサンプルは、文字列 "free" を名前に含むレジストリ内のすべてのオブジェクトが格納された RegistryPackage オブジェクトを発行します。2 つ目のサンプルは、このパッケージを検索し、その内容を表示します。

▼ JAXRPublishPackage サンプルと JAXRSearchPackage サンプルを実行するには

手順 1. **<*INSTALL*>/registry/samples/packages** ディレクトリに移動します。

2. 次のコマンドを入力します。

```
asant pub-pkg
```

3. 次のコマンドを入力します。

```
asant search-pkg
```

レジストリ内のオブジェクトの状態の変更

オブジェクトを Service Registry に発行したり、オブジェクトを何らかの形で変更したりするときは、オブジェクトの監査証跡に AuditableEvent オブジェクトを追加します。これらのイベントやこれらに関する情報の取得方法の詳細については、51 ページの「オブジェクトの監査証跡の取得」を参照してください。51 ページの「オブジェクトの監査証跡の取得」では、イベントやその作成方法について説明しています。

ほかのアクションの結果として多くのイベントが作成されます。

- オブジェクトをレジストリに保存すると、EVENT_TYPE_CREATED イベントが作成されます。
- 次のアクションを行うと、EVENT_TYPE_VERSIONED イベントが作成されます。
  - オブジェクトの名前または説明の変更
  - Classification、ExternalIdentifier、または Slot の追加、変更、または削除
  - Organization または User に対する、PostalAddress または TelephoneNumber の追加、変更、または削除

  オブジェクトのバージョン情報を取得できます。詳細については、52 ページの「オブジェクトのバージョンの取得」を参照してください。

注 – JAXR API を使って発行する場合、オブジェクトのバージョン管理はデフォルトで有効になります。Web コンソールを使って発行する場合、バージョン管理はデフォルトで無効になります。

オブジェクトの状態を明示的に変更することもできます。さまざまなバージョンのオブジェクトが混在しており、何らかのバージョン管理を行う必要がある本稼働環境においては、この機能が役立つ場合があります。たとえば、あるオブジェクトのバージョンを一般的に使用するものとして承認し、古いバージョンを非推奨にしてから削除する、といったことが可能です。オブジェクトを非推奨にしたあとで変更したくなったら、その非推奨を解除できます。登録ユーザーは、自分が所有するオブジェクトに対してのみ、このようなアクションを実行できます。

- LifeCycleManagerImpl.approveObjects メソッドを使ってオブジェクトを承認できます。この機能は実装に固有です。
- LifeCycleManager.deprecateObjects メソッドを使ってオブジェクトを非推奨にできます。
- LifeCycleManager.unDeprecateObjects メソッドを使ってオブジェクトの非推奨を解除できます。

LifeCycleManagerImpl.approveObjects メソッドのシグニチャーは次のとおりです。

```
public BulkResponse approveObjects(java.util.Collection keys)
    throws JAXRException
```

オブジェクトを非推奨にするコードは通常、次のようになります。

```
String id = id_string;
Key key = lcm.createKey(id);
Collection keys = new ArrayList();
keys.add(key);

// deprecate the object
lcm.deprecateObjects(keys);
```

これらのアクションへのアクセスを、レジストリ管理者などの特定のユーザー、ユーザーロール、およびユーザーグループに制限することが可能です。87 ページの「オブジェクトへのアクセスの制御」を参照してください。

RegistryObject の状態を変えないアクションに対しては、AuditableEvent は作成されません。たとえば、クエリーを実行しても AuditableEvent は生成されません。また、ある RegistryObject を RegistryPackage に追加したり、そのオブジェクトをソースまたはターゲットに持つ Association を作成したりしても、そのオブジェクトに対して AuditableEvent が生成されることはありません。

レジストリ内のオブジェクトの状態の変更: 例

オブジェクトの承認、非推奨、非推奨解除の例については、*<INSTALL>*/registry/samples/auditable-events/src にあるサンプル、JAXRApproveObject.java、JAXRDeprecateObject.java、および JAXRUndeprecateObject.java を参照してください。各サンプルは、指定された一意の識別子を持つオブジェクトに対してアクションを実行したあと、そのオブジェクトの監査証跡を表示します。このため、ユーザーはサンプルの実行結果を確認できます。

すべてのサンプルについて、ユーザーが指定するオブジェクトはそのユーザーが作成したものである必要があります。

▼ JAXRApproveObject、JAXRDeprecateObject、および JAXRUndeprecateObject サンプルを実行するには

手順

1. **<*INSTALL*>/registry/samples/auditable-events** ディレクトリに移動します。
2. 次のコマンドを入力します。

   asant approve-obj -Did=*id_string*
3. 次のコマンドを入力します。

   asant deprecate-obj -Did=*id_string*
4. 次のコマンドを入力します。

   asant undeprecate-obj -Did=*id_string*

オブジェクトへのアクセスの制御

レジストリ内のオブジェクトへのアクセスは、アクセス制御ポリシー (ACP) によって設定されます。デフォルトのアクセス制御ポリシーの指定内容は、次のとおりです。

- 定義済みユーザー `Registry Guest` は、任意のオブジェクトを読み取ることができます。Service Registry にログインしていないユーザーはすべて、このアイデンティティーを持ちます。
- すべての登録済みユーザーは、オブジェクトを作成できるほか、自分が所有するオブジェクトに対してアクションを実行できます。
- `RegistryAdministrator` として分類されるすべてのユーザーは、Service Registry 内のすべてのオブジェクトに対してアクションを実行できます。デフォルトでは、定義済みユーザー `Registry Operator` のみが、管理者として分類されています。管理者になる方法については、『Service Registry 3 2005Q4 管理ガイド』の「管理者の作成」を参照してください。

カスタム ACP を使えば、個々のオブジェクトに対して細粒度のアクセス制御を設定できます。ただし、ACP の記述は現時点では手動による処理であり、OASIS の XACML (eXtensible Access Control Markup Language) の知識が必要になります。詳細については、『ebXML RIM 3.0』の第 9 章「Access Control Information Model」、特に 9.7.6 節から 9.7.8 節にかけてのサンプルを参照してください。

Service Registry とリポジトリからのオブジェクトの削除

レジストリに送信したオブジェクトはどれでも、そのレジストリから削除することができます。対象オブジェクトの ID を、`LifeCycleManager.deleteObjects` メソッドの引数として使用します。

次のコードでは、指定されたキー文字列に対応するオブジェクトを削除したあと、正しいオブジェクトが削除されたことをユーザーが確認できるよう、そのキーを再度表示しています。

```
String id = key.getId();
Collection keys = new ArrayList();
keys.add(key);
BulkResponse response = blcm.deleteObjects(keys);
Collection exceptions = response.getException();
if (exceptions == null) {
    System.out.println("Objects deleted");
    Collection retKeys = response.getCollection();
```

```
    Iterator keyIter = retKeys.iterator();
    javax.xml.registry.infomodel.Key orgKey = null;
    if (keyIter.hasNext()) {
        orgKey =
             (javax.xml.registry.infomodel.Key) keyIter.next();
        id = orgKey.getId();
        System.out.println("Object key was " + id);
    }
}
```

Organization を削除しても、その Organization に属する Service オブジェクトと User オブジェクトは削除されません。それらのオブジェクトは個別に削除する必要があります。

Service オブジェクトを削除すると、それに属する ServiceBinding オブジェクトも削除され、さらにその ServiceBinding オブジェクトに属する SpecificationLink オブジェクトも削除されます。ただし、SpecificationLink オブジェクトを削除しても、それに関連付けられた ExtrinsicObject インスタンスとその関連リポジトリ項目は削除されません。付帯オブジェクトは個別に削除する必要があります。

AuditableEvent オブジェクトは、関連付けられたオブジェクトが削除されても削除されません。レジストリの使用を続けると、これらのオブジェクトが多数蓄積されていくのがわかります。

Service Registry からのオブジェクトの削除: 例

Service Registry からオブジェクトを削除する例については、*<INSTALL>*/registry/samples/delete-object/src ディレクトリにある JAXRDelete.java を参照してください。このサンプルは、指定された一意の識別子を持つオブジェクトを削除します。

▼ JAXRDelete サンプルを実行する方法

手順

1. **<*INSTALL*>/registry/samples/delete-object** ディレクトリに移動します。
2. 次のコマンドを入力します。

   asant run -Did=*id_string*

第 6 章

UDDI インタフェース用クライアントプログラムの開発

この章では、Service Registry への UDDI (Universal Description, Discovery and Integration) インタフェース用のクライアントプログラムの作成方法について説明します。

クライアントプログラムの作成

クライアントプログラムは、HTTP 上の SOAP 1.1 プロトコルを使って、Service Registry への UDDI インタフェースにアクセスできます。どのプログラミング言語で作成されたクライアントプログラムであっても、UDDI 3.0.2 Inquiry プロトコルを使って、Service Registry の UDDI インタフェースサービスエンドポイントにアクセスできます。UDDI Inquiry インタフェースのエンドポイントは、次のとおりです。

`http://`*host*`:`*port*`/soar/uddi/inquire`

Service Registry の UDDI インタフェースは、次の URL で定義されている UDDI 3.0.2 Inquiry API WSDL に準拠しています。

- UDDI API バインディング: `uddi_api_v3_binding.wsdl`: `http://uddi.org/wsdl/uddi_api_v3_binding.wsdl`
- UDDI API ポートタイプ: `uddi_api_v3_portType.wsdl`: `http://uddi.org/wsdl/uddi_api_v3_portType.wsdl`

UDDI インタフェース用の Java クライアントプログラムを JAX-RPC 1.1 を使って開発できます。それには、上記の UDDI 3.0.2 WSDL ファイルからクライアントスタブを生成します。詳細については、J2EE 1.4 Tutorial (`http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html`) の第 8 章「Building Web Services with JAX-RPC」を参照してください。

UDDI 3.0.2 WSDL とスキーマに追加や変更を行なって、JAX-RPC 1.1 仕様の要件どおりに Java クライアントが生成されるようにする方法については、次の URL にある『UDDI Spec TC Technical Note』で説明します。

`http://www.oasis-open.org/committees/uddi-spec/doc/tn/uddi-spec-tc-tn-jax-rpc-20050126.htm`。

これで、Java クライアントプログラムは、クライアントスタブによって公開されたメソッドを使って UDDI Inquiry インタフェース上のメソッドを呼び出せるようになります。

現行リリースの Service Registry では、UDDI インタフェースは UDDI 3.0.2 の Publication、Security、Custody Transfer、Subscription の各プロトコルをサポートしていません。次の各 UDDI 3.0.2 インタフェースは、すべてのメソッドに対して `E_unsupported` (10050) エラーコードを返すように実装されています。

`http://`*host*`:`*port*`/soar/uddi/custody`
`http://`*host*`:`*port*`/soar/uddi/publish`
`http://`*host*`:`*port*`/soar/uddi/security`
`http://`*host*`:`*port*`/soar/uddi/subscription`

Inquiry インタフェースの実装は、`-authInfo` 引数による承認や、`-listHead` 引数または `-maxRows` 引数による部分的な結果の要求をサポートしていません。

レジストリへの発行を行うクライアントプログラムでは、これまでに説明してきた JAXR API を使用する必要があります。

付録 A

標準的な定数

この付録では、ebXML Registry and Repository 仕様で定義されている一意の識別子に対する標準的な定数の一覧を記載します。これらの定数は、`org.freebxml.omar.common.CanonicalSchemes` を拡張したインタフェース `org.freebxml.omar.common.CanonicalConstants` 内に定義されています。

これらの定数は、既知のオブジェクトに対する一意の識別子の文字列を定義します。そのようなオブジェクトを識別子で検索する場合に、これらの定数を使用してください。

`org.freebxml.omar.common.CanonicalConstants` 内で定義されている Concept の標準的な定数には、各 Concept の論理識別子 (`lid`) に対する定数と、Concept のコードに対する定数も含まれています。後者は各 Concept の名前です。たとえば、`MemberOf` Concept には、次の 3 つの定数があります。

- `CANONICAL_ASSOCIATION_TYPE_ID_Uses`。`"urn:oasis:names:tc:ebxml-regrep:AssociationType:Uses"` として定義されている
- `CANONICAL_ASSOCIATION_TYPE_LID_Uses`。`"urn:oasis:names:tc:ebxml-regrep:AssociationType:Uses"` として定義されている
- `CANONICAL_ASSOCIATION_TYPE_CODE_Uses`。`"Uses"` として定義されている

分類スキーマには、一意の識別子と論理識別子に対する定数はありますが、コードの定数はありません。

この付録に記載しているのは一意の識別子に対する定数だけですが、適切な場合には `lid` とコードの定数も使用可能です。

分類スキーマに対する定数

標準的な分類スキーマの一意の識別子に対する定数は、次のとおりです。

- CANONICAL_CLASSIFICATION_SCHEME_ID_AssociationType
- CANONICAL_CLASSIFICATION_SCHEME_ID_ContentManagementService
- CANONICAL_CLASSIFICATION_SCHEME_ID_DataType
- CANONICAL_CLASSIFICATION_SCHEME_ID_DeletionScopeType
- CANONICAL_CLASSIFICATION_SCHEME_ID_EmailType
- CANONICAL_CLASSIFICATION_SCHEME_ID_ErrorHandlingModel
- CANONICAL_CLASSIFICATION_SCHEME_ID_ErrorSeverityType
- CANONICAL_CLASSIFICATION_SCHEME_ID_EventType
- CANONICAL_CLASSIFICATION_SCHEME_ID_InvocationModel
- CANONICAL_CLASSIFICATION_SCHEME_ID_NodeType
- CANONICAL_CLASSIFICATION_SCHEME_ID_NotificationOptionType
- CANONICAL_CLASSIFICATION_SCHEME_ID_ObjectType
- CANONICAL_CLASSIFICATION_SCHEME_ID_PhoneType
- CANONICAL_CLASSIFICATION_SCHEME_ID_QueryLanguage
- CANONICAL_CLASSIFICATION_SCHEME_ID_ResponseStatusType
- CANONICAL_CLASSIFICATION_SCHEME_ID_StabilityType
- CANONICAL_CLASSIFICATION_SCHEME_ID_StatusType
- CANONICAL_CLASSIFICATION_SCHEME_ID_SubjectGroup
- CANONICAL_CLASSIFICATION_SCHEME_ID_SubjectRole

関連付けタイプの Concept に対する定数

Association オブジェクトを識別する Concept の一意の識別子に対する定数は、次のとおりです。

- CANONICAL_ASSOCIATION_TYPE_ID_AccessControlPolicyFor
- CANONICAL_ASSOCIATION_TYPE_ID_AffiliatedWith
- CANONICAL_ASSOCIATION_TYPE_ID_CatalogingControlFileFor
- CANONICAL_ASSOCIATION_TYPE_ID_Contains
- CANONICAL_ASSOCIATION_TYPE_ID_ContentManagementServiceFor
- CANONICAL_ASSOCIATION_TYPE_ID_EmployeeOf
- CANONICAL_ASSOCIATION_TYPE_ID_EquivalentTo
- CANONICAL_ASSOCIATION_TYPE_ID_Extends
- CANONICAL_ASSOCIATION_TYPE_ID_ExternallyLinks
- CANONICAL_ASSOCIATION_TYPE_ID_HasFederationMember
- CANONICAL_ASSOCIATION_TYPE_ID_HasMember
- CANONICAL_ASSOCIATION_TYPE_ID_Implements
- CANONICAL_ASSOCIATION_TYPE_ID_InstanceOf

- CANONICAL_ASSOCIATION_TYPE_ID_InvocationControlFileFor
- CANONICAL_ASSOCIATION_TYPE_ID_MemberOf
- CANONICAL_ASSOCIATION_TYPE_ID_OffersService
- CANONICAL_ASSOCIATION_TYPE_ID_OwnerOf
- CANONICAL_ASSOCIATION_TYPE_ID_RelatedTo
- CANONICAL_ASSOCIATION_TYPE_ID_Replaces
- CANONICAL_ASSOCIATION_TYPE_ID_ResponsibleFor
- CANONICAL_ASSOCIATION_TYPE_ID_SubmitterOf
- CANONICAL_ASSOCIATION_TYPE_ID_Supersedes
- CANONICAL_ASSOCIATION_TYPE_ID_Uses
- CANONICAL_ASSOCIATION_TYPE_ID_ValidationControlFileFor

コンテンツ管理サービスの Concept に対する定数

コンテンツ管理サービスを識別する Concept の一意の識別子に対する定数は、次のとおりです。

- CANONICAL_CONTENT_MANAGEMENT_SERVICE_ID_ContentCatalogingService
- CANONICAL_CONTENT_MANAGEMENT_SERVICE_ID_ContentValidationService

データタイプの Concept に対する定数

データタイプを識別する Concept の一意の識別子に対する定数は、次のとおりです。

- CANONICAL_DATA_TYPE_ID_Boolean
- CANONICAL_DATA_TYPE_ID_Date
- CANONICAL_DATA_TYPE_ID_DateTime
- CANONICAL_DATA_TYPE_ID_Double
- CANONICAL_DATA_TYPE_ID_Duration
- CANONICAL_DATA_TYPE_ID_Float
- CANONICAL_DATA_TYPE_ID_Integer
- CANONICAL_DATA_TYPE_ID_ObjectRef
- CANONICAL_DATA_TYPE_ID_String
- CANONICAL_DATA_TYPE_ID_Time
- CANONICAL_DATA_TYPE_ID_URI

削除範囲タイプの Concept に対する定数

削除範囲タイプを識別する Concept の一意の識別子に対する定数は、次のとおりです。

- CANONICAL_DELETION_SCOPE_TYPE_ID_DeleteAll
- CANONICAL_DELETION_SCOPE_TYPE_ID_DeleteRepositoryItemOnly

電子メールタイプの Concept に対する定数

電子メールタイプを識別する Concept の一意の識別子に対する定数は、次のとおりです。

- CANONICAL_EMAIL_TYPE_ID_HomeEmail
- CANONICAL_EMAIL_TYPE_ID_OfficeEmail

エラー処理モデルの Concept に対する定数

エラー処理モデルを識別する Concept の一意の識別子に対する定数は、次のとおりです。

- CANONICAL_ERROR_HANDLING_MODEL_ID_FailOnError
- CANONICAL_ERROR_HANDLING_MODEL_ID_LogErrorAndContinue

エラー重大度タイプの Concept に対する定数

エラー重大度タイプを識別する Concept の一意の識別子に対する定数は、次のとおりです。

- CANONICAL_ERROR_SEVERITY_TYPE_ID_Error

- CANONICAL_ERROR_SEVERITY_TYPE_ID_Warning

イベントタイプの Concept に対する定数

イベントタイプを識別する Concept の一意の識別子に対する定数は、次のとおりです。

- CANONICAL_EVENT_TYPE_ID_Approved
- CANONICAL_EVENT_TYPE_ID_Created
- CANONICAL_EVENT_TYPE_ID_Deleted
- CANONICAL_EVENT_TYPE_ID_Deprecated
- CANONICAL_EVENT_TYPE_ID_Downloaded
- CANONICAL_EVENT_TYPE_ID_Relocated
- CANONICAL_EVENT_TYPE_ID_Undeprecated
- CANONICAL_EVENT_TYPE_ID_Updated
- CANONICAL_EVENT_TYPE_ID_Versioned

呼び出しモデルの Concept に対する定数

呼び出しモデルを識別する Concept の一意の識別子に対する定数は、次のとおりです。

- CANONICAL_INVOCATION_MODEL_ID_Decoupled
- CANONICAL_INVOCATION_MODEL_ID_Inline

ノードタイプの Concept に対する定数

ノードタイプを識別する Concept の一意の識別子に対する定数は、次のとおりです。

- CANONICAL_NODE_TYPE_ID_EmbeddedPath
- CANONICAL_NODE_TYPE_ID_NonUniqueCode
- CANONICAL_NODE_TYPE_ID_UniqueCode

通知オプションタイプの Concept に対する定数

通知オプションタイプを識別する Concept の一意の識別子に対する定数は、次のとおりです。

- CANONICAL_NOTIFICATION_OPTION_TYPE_ID_ObjectRefs
- CANONICAL_NOTIFICATION_OPTION_TYPE_ID_Objects

オブジェクト型の Concept に対する定数

オブジェクト型を識別する Concept の一意の識別子に対する定数は、次のとおりです。

- CANONICAL_OBJECT_TYPE_ID_AdhocQuery
- CANONICAL_OBJECT_TYPE_ID_Association
- CANONICAL_OBJECT_TYPE_ID_AuditableEvent
- CANONICAL_OBJECT_TYPE_ID_Classification
- CANONICAL_OBJECT_TYPE_ID_ClassificationNode
- CANONICAL_OBJECT_TYPE_ID_ClassificationScheme
- CANONICAL_OBJECT_TYPE_ID_ExternalIdentifier
- CANONICAL_OBJECT_TYPE_ID_ExternalLink
- CANONICAL_OBJECT_TYPE_ID_ExtrinsicObject
- CANONICAL_OBJECT_TYPE_ID_Federation
- CANONICAL_OBJECT_TYPE_ID_Notification
- CANONICAL_OBJECT_TYPE_ID_Organization
- CANONICAL_OBJECT_TYPE_ID_Person
- CANONICAL_OBJECT_TYPE_ID_Policy
- CANONICAL_OBJECT_TYPE_ID_PolicySet
- CANONICAL_OBJECT_TYPE_ID_Registry
- CANONICAL_OBJECT_TYPE_ID_RegistryObject
- CANONICAL_OBJECT_TYPE_ID_RegistryPackage
- CANONICAL_OBJECT_TYPE_ID_Service
- CANONICAL_OBJECT_TYPE_ID_ServiceBinding
- CANONICAL_OBJECT_TYPE_ID_SpecificationLink
- CANONICAL_OBJECT_TYPE_ID_Subscription
- CANONICAL_OBJECT_TYPE_ID_User
- CANONICAL_OBJECT_TYPE_ID_XACML
- CANONICAL_OBJECT_TYPE_ID_XForm
- CANONICAL_OBJECT_TYPE_ID_XHTML
- CANONICAL_OBJECT_TYPE_ID_XML

- CANONICAL_OBJECT_TYPE_ID_XMLSchema
- CANONICAL_OBJECT_TYPE_ID_XSLT

電話タイプの Concept に対する定数

電話タイプを識別する Concept の一意の識別子に対する定数は、次のとおりです。

- CANONICAL_PHONE_TYPE_ID_Beeper
- CANONICAL_PHONE_TYPE_ID_FAX
- CANONICAL_PHONE_TYPE_ID_HomePhone
- CANONICAL_PHONE_TYPE_ID_MobilePhone
- CANONICAL_PHONE_TYPE_ID_OfficePhone

クエリー言語の Concept に対する定数

クエリー言語を識別する Concept の一意の識別子に対する定数は、次のとおりです。

- CANONICAL_QUERY_LANGUAGE_ID_ebRSFilterQuery
- CANONICAL_QUERY_LANGUAGE_ID_SQL_92
- CANONICAL_QUERY_LANGUAGE_ID_XPath
- CANONICAL_QUERY_LANGUAGE_ID_XQuery

応答状態タイプの Concept に対する定数

応答状態タイプを識別する Concept の一意の識別子に対する定数は、次のとおりです。

- CANONICAL_RESPONSE_STATUS_TYPE_ID_Failure
- CANONICAL_RESPONSE_STATUS_TYPE_ID_Success
- CANONICAL_RESPONSE_STATUS_TYPE_ID_Unavailable

安定性タイプの Concept に対する定数

安定性タイプを識別する Concept の一意の識別子に対する定数は、次のとおりです。

- CANONICAL_STABILITY_TYPE_ID_Dynamic
- CANONICAL_STABILITY_TYPE_ID_DynamicCompatible
- CANONICAL_STABILITY_TYPE_ID_Static

状態タイプの Concept に対する定数

状態タイプを識別する Concept の一意の識別子に対する定数は、次のとおりです。

- CANONICAL_STATUS_TYPE_ID_Approved
- CANONICAL_STATUS_TYPE_ID_Deprecated
- CANONICAL_STATUS_TYPE_ID_Submitted
- CANONICAL_STATUS_TYPE_ID_Withdrawn

サブジェクトロールの Concept に対する定数

サブジェクトロールを識別する Concept の一意の識別子に対する定数は、次のとおりです。

- CANONICAL_SUBJECT_ROLE_ID_ContentOwner
- CANONICAL_SUBJECT_ROLE_ID_Intermediary
- CANONICAL_SUBJECT_ROLE_ID_RegistryAdministrator
- CANONICAL_SUBJECT_ROLE_ID_RegistryGuest

ストアドクエリーに対する定数

定義済みクエリーに対する定数が 1 つ提供されています。

- CANONICAL_QUERY_GetCallersUser

索引

D

E

F

G

I

J

K

L

N

O

P

Q

ろ

わ