Java コンテンツ・リポジトリー API の紹介

JSR-170 が CMA のビルドを簡単にする方法について

Titus Barik (titus@barik.net)
Senior Programmer Analyst
SoloFX Enterprises, LLC

2006年 6月 27日
(初版 2005年 8月 23日)

コンテンツ・マネージメント・アプリケーションの人気が高まるなか、コンテンツ・リポジトリーに共通の標準 API が必要であることは明らかです。Content Repository for JavaTM Technology API (JSR-170) は、そのようなインターフェースを提供することを目的としたものです。この記事では、JSR-170 のオープン・ソース Apache Jackrabbit インプリメンテーションを用いてウィキペディアのような百科事典バックエンドを設計しながら、この有望なフレームワークの機能を探っていきます。

コンテンツ・マネージメント・アプリケーションを開発しようとしたことがあるならば、コンテンツ・システムのインプリメンテーションにつきものの難しさについては十分おわかりでしょう。数多くのベンダーがそれぞれに固有のリポジトリー・エンジンを提供しているため、全体が断片化されてしまうのです。このような問題のために、コンテンツ・システムはより複雑で保守しにくいものとなり、またベンダーにも一層束縛され、エンタープライズ市場では長期に渡ってレガシー・サポートを受ける必要がますますでてきます。企業の Web ログと電子企業文書管理が普及するにつれ、標準化されたコンテンツ・リポジトリー・インターフェースの必要性は今まで以上に明らかになってきています。

JCP (Java Commnunity Process) のもとで JSR-170 として開発された Content Repository for Java Technology の仕様は、こういった業界のニーズを満たすことを目的としています。この仕様は名前空間 javax.jcr の配下に統一 API を提供するもので、この API に従えば、ベンダー・ニュートラルな方法 (ベンダーに依存しない方法) で、仕様に準拠するすべてのリポジトリー・インプリメンテーションにアクセスすることができます。

JCR (Java Content Repository) が提供する機能は、API の標準化だけではありません。JSR-170 の主な利点は、特定の基本アーキテクチャーにしばられないことです。JSR-170 インプリメンテーションのバックエンド・データ・ストレージは、例えばファイル・システムであってもよいし、WebDAV リポジトリー、XML 支援システム、あるいは SQL データベースであってもかまいません。さらに、JSR-170 のエクスポートとインポート機能によって、インテグレーターはコンテンツ・バックエンドと JCR インプリメンテーションをシームレスに切り替えることができます。また、JCR は、多種多様な既存のコンテンツ・リポジトリーの最上位層と直接のインターフェースを

もつことができると同時に、バージョン管理、アクセス制御、検索などの複雑な機能をまとめて標準化します。

JCR を説明するにはいくつかの方法がありますが、この記事では開発者の立場から JSR-170 仕様が提供する機能について検討します。ここで焦点とするのは、使用可能な API と、プログラマーがコンテンツ・アプリケーションを設計する際に、効率的に JSR-170 リポジトリーを使えるようにするためのインターフェースです。ここでは例として、JCRWiki という、ウィキペディアのような百科事典システム用の単純なバックエンドをインプリメントします。このバックエンドは、バイナリー・コンテンツ、バージョン管理、バックアップ、検索をサポートします。このアプリケーションの開発には、JSR-170 のオープン・ソース・インプリメンテーションである Apache Jackrabbit を使用します。

リポジトリー・モデル

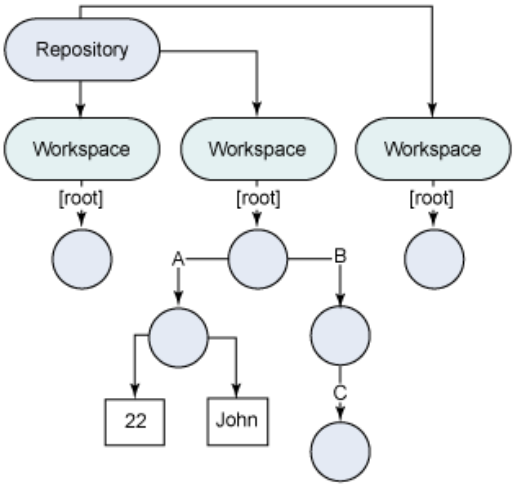
まず、JCR を知っていただくためにリポジトリー・モデルの概要からお話しましょう。リポジトリー・モデルは単純な階層になっており、n分木のようなものです。この階層は単一のコンテンツ・リポジトリーで構成され、コンテンツ・リポジトリーには1 つ以上のワークスペースがあります。(この記事では、ワークスペースを 1 つに絞って説明します。) 各ワークスペースには項目のツリーが含まれ、それぞれの項目はノードまたはプロパティーのどちらかになります。ノードはゼロまたは複数の子と、ゼロまたは複数の関連プロパティーを持つことができ、実際のコンテンツが格納されます。

いずれのノードも、その 1 次ノード型は 1 つしかありません。1 次ノード型は、プロパティーや子ノードのような、ノードが持つことのできる特性を定義します。ノードには1 次ノード型の他、1 つ以上の mixin型がある場合もあります。 mixin 型は修飾子のように機能するもので、ノードの特性をさらに追加します。JCR インプリメンテーションは、事前定義された mixin 型を提供できます。具体的には、以下の3 つです。

- `mix:versionable`: ノードがバージョン管理をサポートするようにします。
- `mix:lockable`: ノードのロック機能を使用可能にします。
- `mix:referenceable`: 参照可能な固有の識別子をノードに付与する `jcr:uuid` プロパティーを自動作成します。

この構造を図 1 に示します。丸はノードを表し、長方形はプロパティーを表しています。ここで対象とするのは、単一のルート・ノードの下に続くノード A、B、C です。 ノード A には、ストリング「John」と整数 22 という 2 つのプロパティーがあります。

図 1. 複数のワークスペースを持つリポジトリー・モデル



事前定義されたノード型

すべてのリポジトリーは、1 次ノード型である `nt:base` をサポートする必要があります。 リポジトリーがサポートできる共通ノード型は、他にもたくさんあります。

- `nt:unstructured` は最もフレキシブルなノード型で、任意の名前の子ノードまたはプロパティーをいくつでも持つことができます。このノード型は、JCRWiki エントリーを表します。
- `nt:file` はファイルを表し、 `jcr:content` という名前の単一の子ノードを必要とします。このノード型は、JCRWiki エントリーのイメージとその他のバイナリー・コンテンツを表します。
- `nt:folder` ノード型は、従来のファイル・システムに含まれるようなフォルダーを表します。
- `nt:resource` は一般に、ファイルの実際のコンテンツを表します。
- `nt:version` は、バージョン管理をサポートするリポジトリーに必要なノード型です。

ノード型の階層全体は、JSR-170 仕様のセクション 6.7.22.1 に記載されています (リンクについては、参考文献を参照してください)。

名前空間

名前空間のサポートは、リポジトリー・モデルの中の便利でありながら、見逃されることの多い機能です。名前空間は、さまざまなソースやアプリケーション・ドメインの項目およびノード型の名前の衝突を防ぎます。名前空間は、単一のコロン(:) で区切られた接頭部で定義されます。この記事で前述した `jcr` 内部プロパティーのjcr、mixin 型の `mix`、ノード型の `nt` が、名前空間です。JCRWiki では、すべてのデータの名前空間として`wiki` を使用します。

JCR をインストールする

Apache Foundation の JSR-170 のオープン・ソース・インプリメンテーションである Apache Jackrabbit は、この記事を作成した時点では、バージョン 1.0 です。コンパイル済みバイトコードの JAR ファイルは、Jackrabbit の Web サイトから直接ダウンロードできます (参考文献を参

照)。Jackrabbit は、SVN を使ってソースからコンパイルすることもできますが、Jackrabbit ライブラリーは夜間のビルドがもはや必要ないところまで安定してきています。このセクションでは、できるだけ短時間で JCR インプリメンテーションを設定して実行する手順を段階的に説明します。

必要なライブラリー

この記事の例を実践するには、クラスパスで以下のライブラリーを使用できるようにします。

- `jackrabbit-core`: Apache 提供の JSR-170 用 Jackrabbit コンテンツ・リポジトリーのコア・インプリメンテーションおよび共通ユーティリティー・コード。
- `commons-collections`commons-collections: Java アプリケーションの開発を早く行えるための強力なデータ構造を含むフレームワーク。
- `concurrent`: Java 並列プログラミングに共通のユーティリティー・クラスを標準化した効率的なバージョンを提供するライブラリー。
- `derby`: Apache DB サブプロジェクト。Java 言語で一貫してインプリメントされるリレーショナル・データベースを提供します。
- `jcr`: JSR-170 仕様に準拠したインターフェースのセット。
- `log4j`: ランタイム・ロギング・ライブラリー。
- `lucene`: 充実した機能を備えた、高性能のテキスト検索エンジン・ライブラリー。
- `slf4j` (Simple Logging Facade for Java): 各種ロギング API 用の単純なファサードとして機能するためのもので、ユーザーがデプロイ時に目的のインプリメンテーションに接続できるようにします。
- `xerces`: 拡張 XML パーサー。SAX バージョン 1、DOM レベル 1、および SAX バージョン 1 の API をサポートします。

上記の JAR ファイルはすべて、Jackrabbit のビルド・プロセスでダウンロードされ、SVN から Jackrabbit をビルドしている場合は Maven キャッシュ・ディレクトリーで使用可能になります。Linux では、これらの JAR はホーム・ディレクトリーの .maven の下にあります。バイナリー・ビルドを使っている場合は、ライブラリーをそれぞれの Web サイトからダウンロードするか、これらのリソースすべてに直接リンクできる Jackrabbit の Web サイトでチュートリアル「First Hops with Jackrabbit」をブラウズしてください。jcr-1.0.jar は、Java コミュニティー・プロセスの Web サイトにある JSR-170 仕様のダウンロードによっても入手できます。

手動構成

JSR-170 では、最初どのようにして `Repository` オブジェクトを取得するかについては、明確に指定していません。これは、リポジトリー・ベンダーごとのインプリメンテーションで詳細を決められます。ただし、JSR-170 インプリメンテーションを Jackrabbit に完全に依存させないようにするアプリケーションでは、コンテナー環境の JNDI やその他の構成メカニズムを使用することをお勧めします。 この方法では初期構成時に複雑さが増しますが、さまざまな JSR-170 インプリメンテーション間での移植性が向上します。これより移植性は劣りますが、単純な構成方法としては、この記事で後述する自動構成があります。

手動構成では、プログラムによってロードされる repository.xml という構成ファイルと組み合わせて JNDI を使うことで、リポジトリーを取得することができます。

リポジトリーの構成

最も簡単な最初のステップは、Jackrabbit 用の repository.xml ファイルを作成することです。この構成ファイルは、多くの重要なタスクを実行します。なかでもとりわけ重要なのは、外部記憶装置、アクセス制御メカニズム、使用可能なワークスペース、バージョン管理システム、そして検索サブシステムを指定するというタスクです。 リスト 1 に、一例を示します。

リスト 1. repository.xml 構成ファイルの例

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Repository>
    <FileSystem
       class="org.apache.jackrabbit.core.fs.local.LocalFileSystem">
        <param name="path" value="${rep.home}/repository"/>
    </FileSystem>

    <Security appName="Jackrabbit">
        <AccessManager
           class="org.apache.jackrabbit.core.security.
              SimpleAccessManager"/>
    </Security>
    <Workspaces
       rootPath="${rep.home}/workspaces"
          defaultWorkspace="default" />
    <Workspace name="${wsp.name}">
        <FileSystem
           class="org.apache.jackrabbit.core.fs.local.
              LocalFileSystem">
            <param name="path" value="${wsp.home}"/>
        </FileSystem>
        <PersistenceManager
            class="org.apache.jackrabbit.core.state.xml.
               XMLPersistenceManager" />
        <SearchIndex
           class="org.apache.jackrabbit.core.query.lucene.
              SearchIndex">
            <param name="path" value="${wsp.home}/index" />
        </SearchIndex>
    </Workspace>
    <Versioning rootPath="${rep.home}/versions">
        <FileSystem
           class="org.apache.jackrabbit.core.fs.local.
              LocalFileSystem">
            <param name="path" value="${rep.home}/versions"/>
        </FileSystem>
        <PersistenceManager
            class="org.apache.jackrabbit.core.state.xml.
               XMLPersistenceManager" />
    </Versioning>
</Repository>
```

上記の構成では、リポジトリー・データを格納するのにローカル・ファイル・システムを使い、アクセス制御に `SimpleAccessManager` を使っています。このファイルの残りの大部分は読んで字のごとくで、そのままリポジトリー・ディレクトリーにコピーすることができます。

セキュリティー構成

プロジェクトのルート・ディレクトリーにある JAAS 構成ファイル jaas.config を使って、簡単なセキュリティーの設定ができます。 リスト 2 に、一例を示します。

リスト 2. JAAS 構成の例

```
Jackrabbit {
org.apache.jackrabbit.core.security.SimpleLoginModule required
   anonymousId="anonymous";
};
```

リポジトリー初期化コード

リスト 3 に示すパッケージは、リポジトリーの初期化に使われます。

リスト 3. 手動構成の場合の初期設定の import 文

```
import org.apache.jackrabbit.core.jndi.RegistryHelper;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;

import javax.jcr.*;
import javax.jcr.query.*;
import javax.jcr.version.*;

import java.util.Hashtable;
import java.util.Calendar;
import java.io.*;

import sun.net.www.MimeTable;
```

Repository オブジェクトを取得するには、`configFile` 変数が repository.xml ファイルを指し、`repHomeDir` 変数がリポジトリーの常駐場所となるローカル・ファイル・システム・ディレクトリーを指すように設定します。JNDI と `RegistryHelper` を併用すると、リスト 4 に示すように簡単にリポジトリーを取得できます。

リスト 4. JNDI を使ったリポジトリー・オブジェクトの取得

```
String configFile = "repository.xml";
String repHomeDir = "repository";

Hashtable env = new Hashtable();
   env.put(Context.INITIAL_CONTEXT_FACTORY,
      "org.apache.jackrabbit.core.jndi" +
      ".provider.DummyInitialContextFactory");

env.put(Context.PROVIDER_URL, "localhost");

InitialContext ctx = new InitialContext(env);

RegistryHelper.registerRepository(ctx,
   "repo",
   configFile,
   repHomeDir,
   true);

Repository r = (Repository) ctx.lookup("repo");
```

次に、`SimpleCredentials` を使用して `Session` オブジェクトを取得します。このインプリメンテーションでは、`SimpleCredentials` はあらゆるユーザー名を受け入れます。別の JCR インプリメ

ンテーションでは、資格情報を提供してくれる LDAP サーバーや外部データベースに接続するような、より複雑な認証メカニズムを提供することもできます。(認証およびアクセス制御の完全な機能は、この記事では扱っていません。詳しくは、JSR-170 仕様のセクション 6.9 を参照してください。)

`Session` オブジェクトは、従来のオブジェクト・リレーショナル・マッピング・ツールに見られるような一時ストレージ・レイヤーをプログラマーに提供するだけでなく、特定のワークスペースへのリンクとして機能します。クライアントはこのオブジェクトを使って、そのセッションに関係するノードまたはプロパティーにアクセスできます。セッションからワークスペースを取得し、そのワークスペースからルート・ノードを取得することができます。これらのステップはすべて、リスト 5 に示す簡単なコードで実行できます。

リスト 5. ワークスペースとルート・ノードの取得

```
SimpleCredentials cred = new SimpleCredentials("userid",
   ".toCharArray());

Session session = r.login(cred, null);
Workspace ws = session.getWorkspace();
Node rn = session.getRootNode();
```

これで、セッション、ワークスペース、ルート・ノード参照を使って、さまざまなアブストラクション・レベルでリポジトリー機能にアクセスできるようになりました。最後に、リポジトリーが正常に初期化されていることを確認するため、rn.getPrimaryNodeType().getName() を呼び出してルート・ノードの名前を印刷します。次のような出力になるはずです。

```
rep:root
```

JAAS を使用しているため、Java VM パラメーターの 1 つに、`-Djava.security.auth.login.config==jaas.config.` を含めることを忘れないでください。

JCRWiki 名前空間

この演習では、JCRWiki コンテンツはいずれも `wiki` 名前空間に分類されます。 リポジトリーにこの名前空間を認識させるには、初期化の際に名前空間を登録して、以下のようにする必要があります。

```
ws.getNamespaceRegistry().registerNamespace
   ("wiki", "http://www.barik.net/wiki/1.0");
```

おめでとうございます。 これでリポジトリーの手動構成が完了しました。

自動構成

Jackrabbit インプリメンテーションは、コア API から `TransientRepository` クラスも提供します。このクラスは、最初のセッションが開始されるとコンテンツ・リポジトリーを自動的に初期化し、最後のセッションが終了されるとリポジトリーをシャットダウンします。単純なスタンドアロン・アプリケーションの場合、`TransientRepository` を使用すると、JSR-170 の移植性が犠牲になりますが、リポジトリーの構成が極めて簡単になります。

`TransientRepository` は、repository.xml とリポジトリー・フォルダーを自動的に作成します。また、内部的には認証およびセキュリティーを扱う `SimpleAccessManager` を提供します。

リスト 6 に示すように、自動構成には初期設定の import 文が必要です。手動構成とは対照的に、すべての JNDI 参照が除去されます。 `RegistryHelper` に代わって、`TransientRepository` が使用されます。

リスト 6. 自動構成の場合の初期設定の import 文

```
import org.apache.jackrabbit.core.jndi.RegistryHelper;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;

import javax.jcr.*;
import javax.jcr.query.*;
import javax.jcr.version.*;

import java.util.Hashtable;
import java.util.Calendar;
import java.io.*;

import sun.net.www.MimeTable;
```

`TransientRepository` は自動的に初期化を行うため、リスト 7 に示すようにリポジトリーを非常に簡単に取得できます。

リスト 7. TransientRepository によるリポジトリー、ワークスペース、およびルート・ノードの取得

```
Repository r = new TransientRepository();
Session session = r.login(new SimpleCredentials("userid", ".toCharArray()));

Workspace ws = session.getWorkspace();
Node rn = session.getRootNode();
```

手動構成の場合と同様に、JCRWiki コンテンツはいずれも `wiki` 名前空間の下に配置されます。

```
ws.getNamespaceRegistry().registerNamespace
   ("wiki", "http://www.barik.net/wiki/1.0");
```

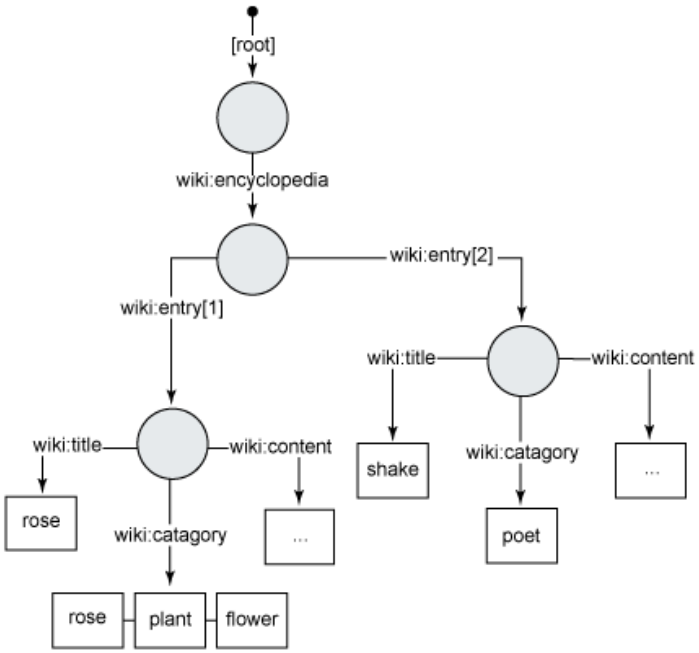
おめでとうございます。 これでリポジトリーの自動構成がこれで完了しました。

JCRWiki の設計方法

ここで、JCRWiki リポジトリー・コンテンツ階層の全体を見てみましょう。 この例では、「rose」と「Shakespeare」という 2 つのエントリーを作成します。型はどちらも `nt:unstructured` です。設計上の規約として、各百科事典エントリーのプロパティーは 3 つとします。つまり、エントリーのタイトル、エントリーのコンテンツ、そして多値カテゴリー・プロパティー (エントリーに複数のカテゴリーがある場合) または単一値カテゴリー・プロパティー (エントリーのカテゴリーが単一の場合) のいずれかです。多値プロパティーは、プログラムで配列として振る舞います。

図 2 に、JCRWiki 設計方法の概略を示します。

図 2. JCRWiki トポロジーの概略図



JCRWiki の機能

コンテンツのないリポジトリーは、あまり役には立ちません。 このセクションでは、JSR-170 が提供する基本的なコンテンツ操作機能を実演し、より高度なオプションのリポジトリー機能 (バージョン管理や XML コンテンツのインポートおよびエクスポートなど) について説明します。

コンテンツを追加する

リスト 8 ではまず最初に、コンテンツ・ノードをリポジトリーに追加します。これによって、図 2 の JCRWiki トポロジーのようになります。

リスト 8. JCR リポジトリーへのコンテンツの追加

```
Node encyclopedia = rn.addNode("wiki:encyclopedia");

Node p = encyclopedia.addNode("wiki:entry");
p.setProperty("wiki:title", new StringValue("rose"));
p.setProperty("wiki:content", new
   StringValue("A rose is a flowering shrub."));
p.setProperty("wiki:category",
   new Value[]{
      new StringValue("flower"),
      new StringValue("plant"),
      new StringValue("rose")});

Node n = encyclopedia.addNode("wiki:entry");
n.setProperty("wiki:title", new StringValue("Shakespeare"));
n.setProperty("wiki:content", new
   StringValue("A famous poet who likes roses."));
n.setProperty("wiki:category", new StringValue("poet"));

session.save();
```

デフォルトでは、Jackrabbit のノードは `nt:unstructured` に設定されます。 「rose」のカテゴリー・プロパティーは多値であることに注意してください。 コードの最後の行で、セッションを保管します。 ノードとノード・プロパティーを追加して設定しても、一時セッション・ストレージ・レイヤーしか変更されないことを思い出してください。 リポジトリーに対するこれらの変更を維持するには、`session.save()` でセッションを保管する必要があります。 ノードを除去するには、対象のノードで `Node.remove()` を呼び出します。

コンテンツにアクセスする

JSR-170 にはノードにアクセスする 2 つの方法、すなわち全探索アクセスと直接アクセスがあります。 全探索アクセスでは、相対パスを使ってコンテンツ・ツリーを探索します。一方、直接アクセスでは、絶対パスまたはUUID (ノードが参照可能な場合) のいずれかを使って直接ノードにジャンプできます。2 つのアクセス方法は似ているため、この記事では全探索アクセスのみをとりあげます。

全探索アクセスは、任意の `Node` オブジェクトやそのメソッドである `Node.getNode()` および `Node.getProperty()` から使用できます。 JCRWiki トポロジーを使用して、以下のコードでルート・ノードから百科事典ノードを取得できます。

```
Node encyclopedia = rn.addNode("wiki:encyclopedia");
```

さらにプロパティーまで全探索することができます。 例えば、JCRWiki トポロジーがあらかじめ分かっている場合、ルート・ノードの「rose」百科事典エントリーでは、以下のようにプロパティーまで全探索できます。

```
String roseTitle = rn.getProperty
  ("wiki:encyclopedia/wiki:entry[1]/wiki:title").getString()
```

`wiki:entry[1]` の下の階層まで全探索することに注意してください。 同じ名前を持つ複数の子がある場合は、インデックス表記を使用して子を明確にすることができます。 JCR では、同じ名前の兄弟には 0 ではなく 1 からインデックスを付けます。さらに、繰り返しの中で `Node.getNodes()` によってノードを取得し、返された順番に基づいて、インデックスは付けられます。

そしてリスト 9 に示すように、特定ノードの子を返す `NodeIterator` を取得することによって、すべての JCRWiki エントリーをブラウズできます。

リスト 9. コンテンツ・リポジトリーのブラウズ

```
Node encyclopedia = rn.getNode("wiki:encyclopedia");
NodeIterator entries = encyclopedia.getNodes("wiki:entry");

while (entries.hasNext()) {

    Node entry = entries.nextNode();

    System.out.println(entry.getName());
    System.out.println(entry.getProperty("wiki:title").getString());
    System.out.println(entry.getProperty("wiki:content").getString());
    System.out.println(entry.getPath());
```

```
    Property category = entry.getProperty("wiki:category");

    try {
      String c = category.getValue().getString();
      System.out.println("Category: " + c);
    } catch (ValueFormatException e) {

      Value[] categories = category.getValues();

      for (Value c : categories) {
        System.out.println("Category: " + c.getString());
      }
    }
}
```

カテゴリー・プロパティーは多値または単一値のどちらかであるため、`try-catch` 文を使って確認します。 多値プロパティーで `getValue()` を呼び出すと、ValueFormatException が投げられます。 通常は、直接アクセスと全探索アクセスのどちらの場合も、内部のノード構造が分かっていなければなりません。 そこで次は、検索を使ってより明示的にノードにアクセスする方法を見てみましょう。

XPath でコンテンツを検索する

すでにおわかりのように、全探索アクセスや直接アクセスをする場合、アクセス対象の位置に関する知識が必要となります。 特定のエントリーを取得するには、JCR のXPath 検索機能を使った方法がより適しています。 ワークスペース・モデルはツリー構造を持つ XML 文書と非常によく似ているため、XPath はノードを見つけるには理想的な構文です。 XPath 照会は、`QueryManager` オブジェクトを使って行われます。 このプロセスは、リスト 10 でわかるように、JDBC によるレコードのアクセスに似ています。

リスト 10. XPath によるコンテンツの検索

```
QueryManager qm = ws.getQueryManager();
Query q = qm.createQuery
("//wiki:encyclopedia/wiki:entry[@wiki:title = 'rose']",
            Query.XPATH);

QueryResult result = q.execute();
NodeIterator it = result.getNodes();

while (it.hasNext()) {
   Node n = it.nextNode();

   System.out.println(n.getName());
   System.out.println(n.getProperty("wiki:title").getString());
   System.out.println(n.getProperty("wiki:content").getString());
}
```

`createQuery()` の 2 番目のパラメーターで、照会に使う言語を指定します。 JCR インプリメンテーションでは、さらに SQL 構文の `Query.SQL` をサポートするように選択することもできます。 また、より複雑な照会を実行することもできます。 例えば以下のようにして、「rose」という言葉が含まれるすべてのエントリーを照会できます。

```
Query q = qm.createQuery
   ("//wiki:encyclopedia/" +
   "wiki:entry[jcr:contains(@wiki:content, 'rose')]",
   Query.XPATH);
```

XML を使ってコンテンツをインポートおよびエクスポートする

JSR-170 では、JCR インプリメンテーション間での移植性を確実にするよう努力が払われています。 この移植性を向上させる方法の 1 つは、標準化された XML インポートおよびエクスポート機能を使うことです。 これらの機能を使うと、標準準拠のベンダー・リポジトリーのコンテンツを別の標準準拠のベンダー・リポジトリーに簡単に転送できます。シリアライズに XML を使うもう一つの利点は、エクスポートされたリポジトリーを従来の XML 構文解析ツールで操作できることです。 リスト 11 に示すように、たった 3 行のコードでエクスポートを実行できます。

リスト 11. データのエクスポート

```
File outputFile = new File("systemview.xml");
FileOutputStream out = new FileOutputStream(outputFile);
session.exportSystemView("/wiki:encyclopedia", out, false, false);
```

エクスポートした XML ファイルは、リスト 12 に示すように、別の新しいリポジトリーに転送できます。

リスト 12. データの転送

```
File inputFile = new File("systemview.xml");
FileInputStream in = new FileInputStream(inputFile);
session.importXML
   ("/", in, ImportUUIDBehavior.IMPORT_UUID_CREATE_NEW);
session.save();
```

バイナリー・コンテンツを追加する

ここまでは、ノードのプロパティーには `StringValue` を使ってきました。 その他にも、JCR はブール値型、日付型、長整数型などの型をサポートします。 リスト 13 に、JCR で使用可能なストリーム型を使って、ノードのバイナリー・イメージを保管するコードを示します。 このリストでは、rose.gif ファイルをメタデータとして `nt:file` node ノードに追加します。 ファイル・データ自体は `nt:resource` サブ子として保管されます。

リスト 13. バイナリー・コンテンツの追加

```
File file = new File("rose.gif");
MimeTable mt = MimeTable.getDefaultTable();
String mimeType = mt.getContentTypeFor(file.getName());
if (mimeType == null) mimeType = "application/octet-stream";

Node fileNode = roseMode.addNode(file.getName(), "nt:file");
Node resNode = fileNode.addNode("jcr:content", "nt:resource");
resNode.setProperty("jcr:mimeType", mimeType);
resNode.setProperty("jcr:encoding", "");
resNode.setProperty("jcr:data", new FileInputStream(file));
Calendar lastModified = Calendar.getInstance();
lastModified.setTimeInMillis(file.lastModified());
resNode.setProperty("jcr:lastModified", lastModified);
```

MimeTable クラスを使ってコンテンツ・タイプを判別した後、`FileInputStream` を使ってファイルをロードします。 あとは正しく名前を付けたプロパティーを、実際のファイル・データを保持する `nt:resource` ノード型に追加するだけです。

バージョン管理

JSR-170 は、アクセス制御、トランザクション、ロック機能、バージョン管理など、多くのオプション機能をサポートします。 これらの機能はそれ自体がトピックとして十分成立するため、ここでは最もよく使われるバージョン管理についてだけ簡単に説明します。 最も単純なケースでは、バージョン管理を実行するために、任意のノードに `mix:versionable` 型を追加します。 そのノードでは、リスト 14 に概略を示すように、CVS の動作に似たメソッドを使ってバージョン管理を行うことができます。

リスト 14. バージョン管理メソッド

```
n.checkout();
n.setProperty("wiki:content", "Updated content for the entry.");
n.save();
n.checkin();
```

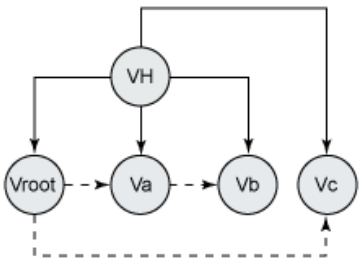
JCR で使用できるその他の動作には、更新、マージ、前のバージョンの復旧などがあります。 特定ノードのバージョン履歴全体をブラウズするには、リスト 15 のステップを実行します。

リスト 15. バージョン履歴のブラウズ

```
VersionHistory vh = n.getVersionHistory();
VersionIterator vi = vh.getAllVersions();

vi.skip(1);
while (vi.hasNext()) {
   Version v = vi.nextVersion();
   NodeIterator ni = v.getNodes();

   while (ni.hasNext()) {
    Node nv = ni.nextNode();
    System.out.println("Version: " +
    v.getCreated().getTime());

    System.out.println(nv.getProperty("wiki:title").getString());
    System.out.println(nv.getProperty("wiki:content").getString());
   }

}
```

`vi.skip(1)` を使うのは、最初は変な感じがするかもしれませんが、バージョン履歴がどのように内部で保管されるかが分かると、その用途は明らかになります。図 3 に、その保管方法を示します。

図 3. ノードのバージョン履歴の構造モデル



ノードのバージョン履歴では、各バージョンがバージョン履歴の子として、後継バージョンを示す参照情報と一緒に保管されます。 図 3 では、バージョン Vb はバージョン Va の後継で、バージョン Vc と Va はともに、バージョン・グラフの開始である Vroot の後継となります。 Vroot は自動作成されたサブノードで、ここがバージョン・グラフの開始点となります。このサブノードには状態情報は含まれません。 そのため、アプリケーションがバージョン履歴で繰り返されると、Vroot はスキップされます。

まとめ

この記事では、JSR-170 仕様が提供するさまざまな機能を広範に渡って紹介しました。 2005 年 6 月 17 日に承認された最終的な仕様リリースによって、Day Software の CRX、そしてObinary の Magnolia Power Pack という 2 つの商用インプリメンテーションがすでに誕生しています。 JSR-170 の登場はまた、企業オープン・ソース・ポータルと、Magnolia および eXo プラットフォームなどのコンテンツ・マネージメント・システムを生み出しました。 何よりも重要なことは、JSR-170 は、SAP AG、Macromedia、そして IBM などの業界のリーダーから強力な支持を得ており、エンタープライズ全体でそ使われるようになり、重要性を確固たるものにしていることです。 オブジェクト関連マッピング・フレームワークがデータベース・プログラミングの形を変えたのと同様に、JCR API はコンテンツ・アプリケーションに対する私たちの考え方とその開発方法を大幅に変える可能性を持っています。

著者について

Titus Barik

Titus Barik は、オープン・ソース・エンタープライズ・ソリューションを専門とするコンテンツ・アプリケーション開発者です。 企業や非営利環境での Magnolia やその他の JSR-170 テクノロジーをデプロイした実績があります。 彼個人の Web ログが barik.net で公開されています。コメント、ご意見をお寄せください。