JDOMでXMLプログラミングを単純化する

このオープン・ソースAPIにより、Java開発者にとってXML文書の操作が簡単になります

Wes Biggs (wes@tralfamadore.com)
上級開発者
T.H.I.

Harry Evans (harry@tralfamadore.com)
上級開発者
T.H.I.

2001年 5月 01日

JDOMは、XMLを扱うためのユニークなJavaツールキットであり、XMLアプリケーションを短期間で開発できるようにします。JDOMの設計は、構文からセマンティクスまでJava言語を取り入れています。しかし、JDOMは既存の (そして、より標準的な) XML APIよりも優れているのでしょうか。いくつかの例を紹介し、この人気のあるオープン・ソース・プロジェクト (最近、Javaの仕様要求として正式に承認されました) の設計目標を浮き彫りにしますので、読者自身で判断してください。

開発者である読者は、おそらく、80対20のルールについて聞いたことがあると思います。Paretoの法則としても知られるもので、「プロセスまたは方法論は、考えられる状況の80%には適用できるが、残りの20%については個別に対応する必要がある」というものです。ソフトウェア開発にこれを当てはめてみれば、関連する事象の80%についてのみ適用できるテクノロジーを開発することは非常に容易であることを意味します。

もちろん、ソフトウェア・プロダクトおよび標準が必ずしも常に80対20ルールに従って発展するわけではありません。特に、細かく割れ砕けたようなJava XMLツールの世界は、このルールの例外です。Javaプログラミングの世界には、特定のXMLタスクに洗練されたソリューションを提供する、数多くのAPIがあります (いくつかは、小規模な手作りのもので、いくつかは大手企業のマーケティング力に支えられたものです)。XMLの普遍性を示す証拠として、すべての新しいタスクごとに新しいテクノロジーが存在します。しかし、それらを結び付けるものは何なのでしょうか。また、何度も行う必要のある80%の内容 (Java言語への直感的なマッピングを用いて行う基本的なXMLツリー操作) に適したツールを見つけるには、どうしたらよいのでしょうか。JDOMは、まさにそうした疑問を念頭に入れて構築されたXML APIです。

鬼さんこちら: JavaとXML

多くの点で、Java言語はXMLに適したプログラム言語となっています。Apache Software FoundationおよびIBM alphaWorksによる草分け的な仕事により、XML文書の作成、操作、変換、および構文解析を行うための完全なツール連鎖が出来上がっています。

しかし、多くのJava開発者が日常的にXMLを使用している一方で、Sunは、XMLをJavaプラットフォームに組み込むという点で業界に後れをとっています。XMLが企業間 (B2B) 統合からWebサイト・コンテンツのストリーミングまで、あらゆるものに必要なキー・テクノロジーとしての地位を築く前に、Java 2プラットフォームは成功を収めていたので、JSRプロセスを使用していたSunは、広く受け入れられている既存のXML APIについてもそれを特例扱いしているのです。現在までのところ、最も注意すべき追加事項はJAXP (Java API for XML Parsing) の組み込みです。これには、次の3つのパッケージが含まれます。

- `org.w3c.dom`。標準でプログラム可能なXMLのDocument Object Modelに関するW3C勧告のJavaでの実装。
- `org.xml.sax`。イベント・ドリブン方式 XML構文解析のための簡単なAPI。
- `javax.xml.parsers`。アプリケーション開発者が特定のパーサー実装を構成および入手できるようにする、ファクトリーの実装。

これらのパッケージを組み込んだことはJava開発者にとってはありがたいことなのですが、これは既存のAPI標準を正式に認めたことにすぎず、Java-XMLのインターオペラビリティーを提供するための大きな飛躍ではありません。コアJavaプラットフォームに欠けていたものは、XML文書をJavaプラットフォームとして操作するための直感的なインターフェースです。

本題のJDOMに入りましょう。JDOMは、2人の有名なJava開発者兼著述家Brett McLaughlinとと Jason Hunter が考案したもので、2000年の初めにApacheに類似したライセンスのもとでオープン・ソース・プロジェクトとして発足しました。JDOMは、コントリビューションや企業からのフィードバック、および広範なJava開発者によるバグ修正を受け入れて成長して、JavaコードからXMLデータのアクセス、操作、および出力を行うための、完全なJavaプラットフォーム・ベースのソリューションを構築することを目指しています。

APIに決まっているさ: JDOMの適用対象

JDOMは、プログラムに基づいてXML文書の操作を行う場合、`org.w3c.dom` パッケージの代わりに使用することができます。これはドロップイン式の置き換えではありません。実際に、JDOMとDOMはうまく共存することができます。さらに、JDOMはテキスト入力されたXMLを構文解析するようにはなっていませんが、パーサーの実装を構成および実行するための多くの作業を引き受けるラッパー・クラスを備えています。JDOMは、プロジェクトのホーム・ページが提唱している、「よりよいマウス・トラップ」を構築するための既存のAPIの機能を基に作られています。

なぜ代替APIが必要なのかを理解するためには、W3C DOMの設計上の制約を思い起こしてください。

- **言語独立である。** DOMはJava言語を念頭に入れて設計されたものではありません。このようなアプローチの結果、異なる言語間でAPIが非常に類似したものになりますが、その一方で、Java言語のイディオムに慣れたプログラマーにとってはAPIが扱いにくくなることも事実

です。たとえば、Java言語では`String`クラスが言語に組み込まれていますが、DOM仕様では独自の`Text`クラスが定義されている、といった具合です。
- **厳密な階層がある。** DOMのAPIは、XML仕様に直接従っています。XML内では、あらゆるものがノードであるため、`Node`ベース・インターフェースが、DOMのほとんどの要素がそれを拡張するというかたちで使われたり、また、`Node`を戻すメソッドのホスト中で使われます。これは多様性の観点から見れば洗練された方法ですが、上に説明したように、Java言語では、`Node`から末端のタイプへのダウン・キャストを行うと、入り組んでいて理解しにくいコードが生成されるため、この方法で作業するのは困難で厄介です。
- **インターフェース主導型である。** パブリックDOM APIはいつくかのインターフェースだけから構成されています (もちろん、`Exception`クラスだけは例外です)。W3Cは実装を提供することには関心を示さず、インターフェースの定義だけに関心を払っています。確かに一理あります。しかしこのことは、Javaプログラマーとして APIを使用すると、XMLオブジェクトの作成時にかなりの程度の分離作業を押し付けられることも意味します。W3C標準は、汎用ファクトリー・クラス、および、同じように柔軟でしかもあまり直接的でないパターンを頻繁に使用するからです。XML文書がパーサーだけで作成され、アプリケーション・レベルのコードで作成されることが決してないような特定の用途では、これが問題となることはありません。しかし、XMLの用途が広がるにつれ、取り扱う対象のすべてがパーサー主導型のままであるとは限らなくなります。このため、アプリケーション開発者にとっては、XMLオブジェクトをプログラムに基づいて作成するための便利な方法が必要になります。

プログラマーにとって、これらの制約は、APIが、(メモリー使用とインターフェース・サイズの両方の点で) 重くて扱いにくく、習得が困難で使い勝手の悪いものになることを意味します。これと対照的に、JDOMは、一にも二にもJava中心の、軽量のAPIとして考案されています。JDOMではDOMの原則をすっかり方向転換し、上記の扱いにくさを解消しています。

- **JDOMはJavaプラットフォーム固有のものである。** このAPIは、可能な場合にはJava言語の組み込み`String`サポートを使用しますので、テキスト値が常に`String`として使用可能になります。また、`List`や`Iterator`などのようなJava 2プラットフォームのコレクション・クラスを使用して、Java言語に慣れたプログラマーのために豊富な環境を提供します。
- **階層がない。** JDOMでは、XMLエレメントは`Element`のインスタンスであり、XML属性は`Attribute`のインスタンスであり、XML文書自体は`Document`のインスタンスです。これらすべては、XMLにおいて異なる概念を表していますので、漠然とした「ノード」として参照されることは決してなく、常に独自のタイプとして参照されます。
- **クラス主導型である。** JDOMオブジェクトは`Document`、`Element`、および`Attribute`などのクラスの直接のインスタンスであるため、オブジェクトの作成は、Java言語における`new`演算子の使用と同じように、簡単に行うことができます。このことはまた、ファクトリー・インターフェースを構成する必要がないことも意味します。JDOMは、とりだしてすぐに使用することができます。

見てよ、ママ`Node`がないよ: JDOM文書の作成と操作

JDOMはJavaの標準コーディング・パターンを使用します。そして、可能な場合には、複雑なファクトリー・パターンの代わりにJavaの`new`演算子を使用しますので、初心者にもオブジェクトの操作が簡単に行えます。例として、JDOMを使用して簡単なXML文書を新しく作成する方法を示します。ここで作ろうとしている構造はリスト1に示すとおりです (この記事に関連する完全なコードは、参考文献からダウンロードできます。)

リスト1. 作成しようとしているサンプルXML文書

```
<?xml version="1.0" encoding="UTF-8"?>
<car vin="123fhg5869705iop90">
  <!--Description of a car-->
  <make>Toyota</make>
  <model>Celica</model>
  <year>1997</year>
  <color>green</color>
  <license state="CA">1ABC234</license>
</car>
```

注: 私たちが作る サンプル文書 は、下記のリスト2から7までで詳しく説明されています。

最初に、ルート・エレメントを作成して、それを文書に追加しましょう。

リスト2.`Document` の作成

```
Element carElement = new Element("car");
Document myDocument = new Document(carElement);
```

このステップにより新しい`org.jdom.Element` が作成され、それが`org.jdom.Document` のルート・エレメントである`myDocument` になります。(参考文献で提供されるサンプル・コードを使用する場合には、必ず`org.jdom.*` をインポートしてください。)XML文書は必ず単一のルート・エレメントを持っていなければなりませんので、`Document` はそのコンストラクターに`Element` を取り込みます。

次に、`vin` 属性を追加します。

リスト3.`Attribute` の追加

```
carElement.addAttribute(new Attribute("vin", "123fhg5869705iop90"));
```

エレメントの追加も非常に単純です。ここでは`make` エレメントを追加します。

リスト4. エレメントとサブエレメント

```
Element make = new Element("make");
make.addContent("Toyota");
carElement.addContent(make);
```

`Element` の`addContent` メソッドは`Element` を戻しますので、これを次のように書くこともできます。

リスト5. 簡潔な形式によるエレメントの追加

```
carElement.addContent(new Element("make").addContent("Toyota"));
```

どちらのステートメントも同じことを行います。最初の例のほうが読みやすいと言う人もいるでしょうが、多数のエレメントを同時に構成する場合には、2番目の例のほうが読みやすくなります。文書の構成を終えるために、次のようにします。

リスト6. 残りのエレメントの追加

```
carElement.addContent(new Element("model").addContent("Celica"));
carElement.addContent(new Element("year").addContent("1997"));
carElement.addContent(new Element("color").addContent("green"));
carElement.addContent(new Element("license")
    .addContent("1ABC234").addAttribute("state", "CA"));
```

`license` エレメントについては、エレメントの内容を追加するだけでなく属性も追加して、ライセンス発行時の状態を指定しています。これが可能なのは、`Element` に対する`addContent` メソッドが、ボイド宣言を行う代わりに、常に`Element` 自体を戻すからです。

コメント・セクションまたはその他の標準XMLタイプの追加も、同じように行います。

リスト7. コメントの追加

```
carElement.addContent(new Comment("Description of a car"));
```

文書の操作も、似たような方法で行われます。たとえば、`year` エレメントへの参照を獲得するには、`Element` の`getChild` メソッドを使用します。

リスト8. 子エレメントのアクセス

```
Element yearElement = carElement.getChild("year");
```

このステートメントは、実際には、`year` というエレメント名の最初の子`Element` を戻します。`year` エレメントがない場合には、呼び出しはヌルで戻されます。DOM`Node` インターフェースなどの、どのようなものからの戻り値も、アップ・キャストする必要はありません。つまり、`Element` の子は`Element` のままです。同じような方法で、`year` エレメントを文書から除去することができます。

リスト9. 子エレメントの除去

```
boolean removed = carElement.removeChild("year");
```

この呼び出しは`year` エレメントだけを除去します。文書のそれ以外の部分はそのまま残ります。

ここまでは、文書の作成方法と操作方法を説明してきました。完成した文書をコンソールに出力するには、JDOMの`XMLOutputter` クラスを使用することができます。

リスト10. JDOMからXMLテキストへの変換

```
try {
    XMLOutputter outputter = new XMLOutputter("  ", true);
    outputter.output(myDocument, System.out);
} catch (java.io.IOException e) {
    e.printStackTrace();
}
```

`XMLOutputter` には、少数のフォーマット・オプションがあります。ここでは、子エレメントを親エレメントから2桁字下げするように指定し、またエレメントとエレメントの間で改行を行うように指定しました。`XMLOutputter` は`Writer` に出力することも`OutputStream` に出力することもできます。ファイルに出力したい場合には、出力行を次のように変更することができます。

リスト11.`FileWriter` によるXMLの出力

```
FileWriter writer = new FileWriter("/some/directory/myFile.xml");
outputter.output(myDocument, writer);
writer.close();
```

ほかの子とも仲良く: 既存のXMLツールとの相互操作

JDOMには、他のAPIとのインターオペラビリティーという興味深い機能があります。JDOMを使用すると、文書を`Stream` や`Reader` に出力するだけでなく、SAX`Event Stream` やDOM`Document` として出力することもできます。このような柔軟性があるため、JDOMを異機種混合の環境で使用したり、すでに別の方法を使用してXMLを処理しているシステムにJDOMを追加したりすることができます。後の例で見るように、JDOMは、まだJDOMデータ構造をネイティブで認識していない他のXMLツールを使用することもできます。

JDOMのもう1つの用途として、すでに存在するXMLデータを読み取ったり操作したりする機能があります。フォーマットの整ったXMLファイルの読み取りは、`org.jdom.input` 内のクラスの1つを使用して行われます。この例では、`SAXBuilder` を使用します。

リスト12.`SAXBuilder` によるXMLファイルの構文解析

```
try {
  SAXBuilder builder = new SAXBuilder();
  Document anotherDocument = builder.build(new File("/some/directory/sample.xml"));
} catch(JDOMException e) {
  e.printStackTrace();
} catch(NullPointerException e) {
  e.printStackTrace();
}
```

このプロセスで作成された文書の操作は、前のリスト2から7までで示した方法によって行うことができます。

JDOMのもう1つの実用的な利用方法として、JDOMをApacheの製品Xalanと結合することができます (参考文献を参照)。上記の自動車の例を使用して、特定の自動車の詳細を表示する内容の、オンライン自動車ディーラーのWebページを作成することができます。最初に、上で作成した文書が、私たちがユーザーに提示しようとする自動車に関する情報を表示するものと仮定します。次に、このJDOM`Document` をXSLスタイルシートと組み合わせ、HTML形式の結果をサーブレットの`OutputStream` に出力して、ユーザーのブラウザーで表示できるようにします。

この例では、`car.xsl` というXSLスタイルシートを使用することにします。

リスト13. 自動車のレコードをHTMLに変換するXSL文書

```
<?xml version="1.0" encoding="UTF-8"?>


<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/car">
    <html>
        <head>
          <title><xsl:value-of select="make"/> <xsl:value-of select="model"/>
        </head>
```

```
        <body>
          <h1><xsl:value-of select="make"/></h1><br />
          <h2><xsl:value-of select="model"/></h2><br />
          <table border="0">
          <tr><td>VIN:</td><td><xsl:value-of select="@vin"/></td></tr>
          <tr><td>Year:</td><td><xsl:value-of select="year"/></td></tr>
          <tr><td>Color:</td><td><xsl:value-of select="color"/></td></tr>
          </table>
        </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

次に、`org.jdom.Document` をDOM`Document` に変換し、私たちのXSLを表すファイルと、ここでのアプリケーション・サーバー (このサーバーは、たまたまサーブレットを使用しているものとします) から得られた`OutputStream` とともに、それをXalanに送ります (リスト14を参照)。

リスト14. JDOMおよびXalanによるHTML文書の作成

```
TransformerFactory tFactory = TransformerFactory.newInstance();
// Make the input sources for the XML and XSLT documents
org.jdom.output.DOMOutputter outputter = new org.jdom.output.DOMOutputter();
org.w3c.dom.Document domDocument = outputter.output(myDocument);
javax.xml.transform.Source xmlSource = new javax.xml.transform.dom.DOMSource(domDocument);
StreamSource xsltSource = new StreamSource(new FileInputStream("/some/directory/car.xsl"));
// Make the output result for the finished document using
// the HTTPResponse OutputStream
StreamResult xmlResult = new StreamResult(response.getOutputStream());
// Get a XSLT transformer
Transformer transformer = tFactory.newTransformer(xsltSource);
// Do the transform
transformer.transform(xmlSource, xmlResult);
```

この例では、出力はJavaサーブレットの`HTTPResponseOutputStream` を介して流されます。しかし、このストリームは、前に`XMLOutputter` を使用する例で示したように、簡単にファイル・ストリームにすることもできます。XalanのためのXMLソースを生成するためには、`DOMOutputter` を使用しました。しかし、同じ入力は、`XMLOutputter` でXML文書を`String` として出力し、それを`StreamSource` にすることもできます。柔軟性については、次のようなことが言えます。すなわち、JDOMはその構造を`String`、SAX`Event Stream`、またはDOM`Document` のいずれとしても出力することができます。これにより、JDOMは、これらのどれかのモデルを入力として使用できるようなツールとインターフェースすることができます。(追加機能については、JDOM Webサイトで`contrib` パッケージを調べてください。JDBC ResultSetベースのビルダーやXPATHの実装などのツールを提供するJDOMベースのユーティリティーが含まれる、貴重なライブラリーが見つかります。)

JDOMは、ごくわずかな行数のコードで広範な機能を可能にします。私たちは、XMLを用いて、XML文書を構文解析、またプログラムとして作成し、それらの文書を操作し、またそれらを使用してXML主導型のWebページを作成しました。

JDOMの成長: 将来への一瞥

SunとJDOM: パッケージ名について

JDOMの公式の1.0リリースは、Java Community Processの間断のない進化と同時に行われる可能性があります。JSR-102として提出されたJDOMは、Sunによる「JDOMは以前のAPIよりも

かなり使いやすいようであるため、弊社は、これのプラットフォームへの追加は大変効果的であると信じています」というコメントにより、Javaのコア・プラットフォームに実質的に組み込まれることが承認されています。JSRによると、1.0のリリース時にはJDOMのパッケージングが "org.jdom" から "javax.xml.tree" に変更される可能性があります。未来は確かに肯定的に見えますが、開発者たちは、最終的には、自分たちのコードを変更して、新規バージョンで使用できるようにする必要が生じるかもしれません。

この記事を書いている時点では、JDOMプロジェクトは ベータ6バージョンをリリースしています。JDOMはベータ状態ではありますが、数多くの実用に対応できる安定した実装であることがすでに明らかになっています。API自体は、すでに大部分が安定した内容になっていますが、多くの分野で、まだ作業が続けられていて、既存のインタフェースに影響を与える可能性があります。したがって、現時点で進行中の開発プロジェクトでは、バグだらけの実装であることを恐れてJDOMを避ける必要はありませんが、最終的なリリースが行われてJavaのコアAPIに適用される前に、いつかのメソッド・シグネチャやある部分の意味が変更される可能性がある、という事実は考慮すべきです。(パッケージ名についてを参照してください。)

JDOMの当面の予定は、APIの安定化と、実装の各部分のパフォーマンスの評価に焦点が当てられています。現在進行中のその他の項目で、一部のアプリケーション開発者の仕事の邪魔になりそうなものとして、DTDエンティティーおよび使用頻度の低い構成のサポートがあります。さらにその先には、XPATH (XSLTなどのアプリケーションにとってネイティブなXMLパス言語) のコア・サポート、およびXMLデータ・ソースとのより直接的な統合があります。

結論として、JDOMは既存のXML APIよりも優れているのでしょうか。読者が夢の中でもJava言語を使うほどになっているのであれば、答えはおそらくイエスです。JDOMは、読者のお気に入りのパーサーやXML対応データベースに取って代わるものではありませんが、その設計原則は、XMLの世界に入ろうとする、あるいはその世界にすでに熟達しているJava開発者にとっては、特に短時間で習得できるものです。

著者について

Wes Biggs

Wes Biggsは、Los Angeles TimesやUSWeb and Elite Information Systemsなどの会社のためにインターネット・アプリケーションを開発してきました。彼はオープン・ソースJavaプロジェクトに頻繁に貢献し、Free Software Foundationのgnu.regexp 正規表現パッケージを保守しています。Wesの連絡先は wes@tralfamadore.com です。

Harry Evans

Harry Evansは、いくつかのWebベースおよびインターネット対応製品の設計 (特に起動環境) を含め、ソフトウェア設計とアプリケーション・エンジニアリングの分野でさまざまな経験を積んでいます。彼は、Rapid Application Developmentから既存製品の統合まで、製品のライフ・サイクルのあらゆる段階で仕事をしてきました。Harryの連絡先は harry@tralfamadore.com です。