Introduction of parsing XML data using JDOM

Sungsoo Kim

April 6, 2009
Abstract

This document describes what XML is and how to receive information using XML feedback. Examples are shown using XML feedback sent by weather.com. The first section of this application note introduces what is XML. The second section describes how to retrieve XML data and how to parse the information using JDOM. This application note assumes the reader is familiar with Java.

Keywords

JAVA, XML, Parser, JDOM, weather.com

Objectives

● Introduce XML

● Teach how to retrieve the data from XML document using JDOM
1 Introduction

XML is widely used for exchanging information on the World Wide Web. It stands for eXtensible Markup Language. Extensible means that the user can define its own structure according to its purpose. Markup Language is a system of how the document is to be described or logically structured. The two most famous Markup Languages are XML and HTML. While HTML is used for displaying data on the web browser, XML is used for carrying data on the web.

Compatibility is the greatest strength of XML. Data written in XML format can be shared between any different kinds of computers, different kinds of operating systems or different kinds of applications. Applications can define their own data structure in XML format so that different applications can communicate data and interpret for its own purpose using different XML Parser. For example, Safari Web browser running on Macbook can interpret XML data written in Microsoft Windows running on Intel i7 processor. XML is based on international standards and supports Unicode that it supports languages other than English. Its compatibility also makes it easier to upgrade or expand to new applications or new browsers without losing old data.

To begin, it is necessary to overview the XML structure and parsing in order to understand how it is used.
2 Structure of XML

XML syntax is quite strict. Only “Well-Formed” or “Validated” XML data can be used by other XML parsers. In this section, a structure of “Well-Formed” XML is discussed. Figure 1 shows actual data written in XML format.

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!-- This document is intended only for use by authorized licensees of The -->
<!-- Weather Channel. Unauthorized use is prohibited. Copyright 1995-2009, -->
<!-- The Weather Channel Interactive, Inc. All Rights Reserved. -->
<weather ver="2.0">
  <head>
    <locale>en_US</locale>
    <form>MEDIUM</form>
    <ut>F</ut>
    <mi></mi>
    <mph></mph>
    <in></in>
    <in></in>
  </head>
  <doc id="48823">
    <dnm>East Lansing, MI (48823)</dnm>
    <tm>2:15 PM</tm>
    <lat>42.78</lat>
    <lon>-84.44</lon>
    <sunr>6:13 AM</sunr>
    <sum>8:09 PM</sum>
    <zone>-4</zone>
  </loc>
  <lnks type="prmo">
    <cc>
      <lsup>4/5/09 1:53 PM EDT</lsup>
      <obst>Lansing, MI</obst>
      <tmp>42</tmp>
      <flk>37</flk>
      <cloudy></cloudy>
      <icon>26</icon>
      <bar>
        <r>29.84</r>
        <d>failing</d>
      </bar>
      <wind>
        <hmid>53</hmid>
        <vis>10.0</vis>
        <uv>
          <i>1</i>
          <t>Low</t>
        </uv>
        <dewp>26</dewp>
      </moon>
    </cc>
  </lnks>
</weather>
```

Figure 1. XML data sent by weather.com
2.1 Declaration

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
```

**Figure 2. XML Declaration**

XML document declaration includes the information of the XML version and how it is encoded. XML declaration is always placed in the top line. There are two versions of XML, XML1.0 and XML 1.1. To avoid confliction, the version information is declared in the XML declaration. Since XML supports Unicode, data can be written in many languages other than English. Encoding information must be included to inform the receiver of data which language this XML data written. Any instructions declared between `<?` and `?>` is called as processing instruction. `<?xml version="1.0"?>` is a processing instruction, however XML calls it as declaration.

2.2 Comments

```xml
<!-- This document is intended only for use by authorized licensees of The -->
<!-- Weather Channel. Unauthorized use is prohibited. Copyright 1995-2009, -->
<!-- The Weather Channel Interactive, Inc. All Rights Reserved. -->
```

**Figure3. Comments in XML**

Comments are used to inform the user of XML data or help the user understand the data structure. All XML comments begin with `<!--` and close with `-->`. XML comments can be placed anywhere.
2.3 Elements and Attribute

Elements contain the actual data of XML document. Elements must have a start-tag and an end-tag which contain the element’s name. Element content sits between these two tags. Elements have a tree-based data structure. In the example in Figure 4, `dnam`, `tm` and other elements but `loc` have exact values and are indented, which mean these elements are children of the `loc` element.

Figure 4. Elements in XML

Figure 5. Illustrates Figure 4 as a tree-based data structure
There is only one root element per each XML document, which contains all of the data’s information. In figure 1, the root element is weather.

In the start-tag of the loc element, there is one piece of information, id. The information included in the start-tag is called Attribute. Attribute makes XML documents easy to understand and clarify data. The value of attribute must be quoted, using either single quotes or double quotes.

```
<element_name attribute_name="attribute_value">Element Content</element_name>
```

**Figure 6. Syntax for using Element and Attribute**

XML also provides ways to represent empty element contents. By adding a slash after the element name in start-tag, “Empty element” can be represented. Empty element can have attribute name and values.

### 2.4 Entity

Entity is often used in XML to represent single characters which are not easy to typed on the keyboard, for example, “&”, “<”, and “>”. Entity reference consists of an ampersand, the entity’s name, and a semicolon. Some special characters’ entity values are predefined. For example, the ampersand is defined as “&amp;”. Undefined special
characters can be defined by the user in the document’s *Document Type Definition* (DTD).

### 3. Parse XML data

After receiving XML data from the World Wide Web, there must be some methods to handle XML data. A parser for XML is used to handle data in XML format. In computer science, a parser is one of the components in a compiler, which checks for correct syntax and builds a data structure. There are several types of parsers, but SAX (Simple API(Application Programming Interface) for XML) and DOM (Document Object Model for XML) are the two most commonly used parsers. However, SAX uses event-driven API so that previously parsed data cannot be re-read. DOM has a complicated API and is slow compared to SAX. To compliment the disadvantages of these two parsers, JDOM was developed by Jason Hunter and Brett McLaughlin in March 2000. JDOM integrates DOM and SAX and it is designed for JAVA platform. The following covers parsing XML documents using JDOM.

#### 3.1 Creating XML Document

To create an XML document, the first thing to do is declare the Document object.

```java
SAXBuilder saxBuilder = new SAXBuilder();
Document document = saxBuilder.build(URL);
```

**Figure 7. Example of the declaration of Document object**
In the first line, the user defines which builder, either SAXBuilder or DOMBuilder, is used. In this application note, SAXBuilder is used. Declare SAXBuilder and use Build function in SAXBuilder to create document object like Figure 7. Build function is used to retrieve the XML data from either a file in local drive or the server on the web. Appendix A shows the usage of Build function. Now, all XML data is in the Document object.

3.2 Parse XML data

After creating an XML document, the user can parse any data from the document object. JDOM provides several functions to parse the XML documents and return XML document’s Attributes value, Entities, and Elements. The getRootElement function, the getChild function and the getText function are the most used functions to parse the XML documents. Declare the getRootElement function like Figure 8.

```java
Element root = document.getRootElement();
```

Figure 8. Example of using the getRootElement function

The getRootElement function returns the root element of the XML document. After retrieving the root element, the user can get the data from its child node. Use the getChild function to retrieve the specified child element of the root element. Figure 9 is an example of retrieving dnam Element from its parent node, loc Element.
Figure 9. Example of using the getChild function

Use the getText function to retrieve the data after retrieving Element containing the data user wants.

```java
Element e = loc.getChild("dnam");

String Display_Name = e.getText();
```

Figure 10. Example of using the getText function

Figure 10 shows how to retrieving the data from the Element. The data of dnam element, which contains city name and zip code, is stored in Display_Name string. After storing the XML data in string, the user can print out the value to the screen or return to main function for the further usage.

4. Conclusion

XML is widely used because of its compatibility and extensibility. XML has a tree-based data structure so that every element but root element has both parent node and child node. XML is made of elements, entities, comments and processing instructions. JDOM integrates advantages of both SAX and DOM. To parse XML document, declare which Builder to use and create Document object. JDOM provides a large selection of functions to parse the XML documents and retrieve the data for the user.
## Appendix

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>build(java.io.File file)</code></td>
<td><code>Document</code></td>
<td>This builds a document from the supplied filename.</td>
</tr>
<tr>
<td><code>build(org.xml.sax.InputSource in)</code></td>
<td><code>protected Document</code></td>
<td>This builds a document from the supplied input source.</td>
</tr>
<tr>
<td><code>build(java.io.InputStream in)</code></td>
<td><code>Document</code></td>
<td>This builds a document from the supplied input stream.</td>
</tr>
<tr>
<td><code>build(java.io.InputStream in, java.lang.String systemId)</code></td>
<td><code>Document</code></td>
<td>This builds a document from the supplied input stream.</td>
</tr>
<tr>
<td><code>build(java.io.Reader characterStream)</code></td>
<td><code>Document</code></td>
<td>This builds a document from the supplied Reader.</td>
</tr>
<tr>
<td><code>build(java.io.Reader characterStream, java.lang.String SystemId)</code></td>
<td><code>Document</code></td>
<td>This builds a document from the supplied Reader.</td>
</tr>
<tr>
<td><code>build(java.lang.String systemId)</code></td>
<td><code>Document</code></td>
<td>This builds a document from the supplied URL.</td>
</tr>
<tr>
<td><code>build(java.net.URL url)</code></td>
<td><code>Document</code></td>
<td>This builds a document from the supplied URL.</td>
</tr>
</tbody>
</table>

### Appendix A. Table for the Build function in SAXBuilder class
Reference

Reference:


