Java + XML = JDOM

by Jason Hunter and Brett McLaughlin
co-creators of JDOM

Mountain View Java User's Group
April 26, 2000
Introductions

Jason Hunter
jhunter@jdom.org
K&A Software
http://www.servlets.com

Author of
"Java Servlet Programming"
(O'Reilly)
Introductions

Brett McLaughlin
brett@jdom.org
Metro Information Services
http://www.newInstance.com

Author of upcoming
"Java and XML"
(O'Reilly)
What is JDOM?

- JDOM is the Java Document Object Model

- A way to represent an XML document for easy and efficient reading, manipulation, and writing
  - Straightforward API
  - Lightweight and fast
  - Java-optimized

- Despite the name similarity, it's not build on DOM or modeled after DOM
  - Although it integrates well with DOM and SAX
  - Name chosen for accuracy, not similarity to DOM

- An open source project with an Apache-style license
The JDOM Philosophy

• JDOM should be straightforward for Java programmers
  – Use the power of the language (Java 2)
  – Take advantage of method overloading, the Collections APIs, reflection, weak references
  – Provide conveniences like type conversions

• JDOM should hide the complexities of XML wherever possible
  – An Element has content, not a child Text node, which has content (ala DOM)
  – Exceptions should contain useful error messages
  – Give line numbers and specifics, use no SAX or DOM classes or constructs
More JDOM Philosophy

• JDOM should integrate with DOM and SAX
  – Support reading and writing DOM documents and SAX events
  – Support runtime plug-in of any DOM or SAX parser
  – Easy conversion from DOM/SAX to JDOM
  – Easy conversion from JDOM to DOM/SAX

• JDOM should stay current with the latest XML standards
  – DOM Level 2, SAX 2.0, XML Schema

• JDOM does not need to solve every problem
  – It should solve 80% of the problems with 20% of the effort
  – We think we got the ratios to 90% / 10%
The Historical Alternatives: DOM

- DOM is a large API designed for complex environments
  - Represents a document tree fully held in memory
  - Has to 100% accurately represent any XML document (well, it attempts to)
  - Has to have the same API on multiple languages
  - Reading and changing the document is non-intuitive
  - Fairly heavyweight to load and store in memory
The Historical Alternatives: SAX

• SAX is a lightweight API designed for fast reading
  – Callback mechanism reports when document elements are encountered
  – Lightweight since the document is never entirely in memory
  – Does not support modifying the document
  – Does not support random access to the document
  – Fairly steep learning curve to use correctly
Do you need JDOM?

- JDOM is a lightweight API
  - Benchmarks of "load and print" show performance on par with SAX
  - Manipulation and output are also lightning fast

- JDOM can represent a full document
  - Not all must be in memory at once

- JDOM supports document modification
  - And document creation from scratch, no "factory"

- JDOM is easy to learn
  - Optimized for Java programmers
  - Doesn't require in-depth XML knowledge
  - Allows easing into SAX and DOM, if needed
  - Simple support for namespaces, validation
The Document class

- Documents are represented by the `org.jdom.Document` class
  - A lightweight object holding a `DocType`, `ProcessingInstructions`, a root `Element`, and `Comments`

- It can be constructed from scratch:

  ```java
  Document doc =
  new Document(new Element("rootElement"));
  ```

- Or it can be constructed from a file, stream, or URL:

  ```java
  Builder builder = new SAXBuilder();
  Document doc = builder.build(url);
  ```
The Build Process

- A Document can be constructed using any build tool
  - The SAX build tool uses a SAX parser to create a JDOM document

- Current builders are SAXBuilder and DOMBuilder
  - `org.jdom.input.SAXBuilder` is fast and recommended
  - `org.jdom.input.DOMBuilder` is useful for reading an existing DOM tree
  - A builder can be written that lazily constructs the Document as needed
  - Other possible builders: LDAPBuilder, SQLBuilder
Builder Classes

- Builders have optional parameters to specify implementation classes and whether DTD-based validation should occur.

```java
SAXBuilder(String parserClass, boolean validate);
DOMBuilder(String adapterClass, boolean validate);
```

- Not all DOM parsers have the same API
  - Xerces, XML4J, Project X, Oracle (V1 and V2)
  - The DOMBuilder `adapterClass` implements `org.jdom.adapters.DOMAdapter`
  - Implements standard methods by passing through to an underlying parser
  - Adapters for all popular parsers are provided
  - Future parsers require just a small adapter class

- Once built, documents are not tied to their build tool
The Output Process

• A Document can be written using any output tool
  – `org.jdom.output.XMLOutputter` tool writes the document as XML
  – `org.jdom.output.SAXOutputter` tool generates SAX events
  – `org.jdom.output.DOMOutputter` tool creates a DOM document (coming soon)
  – Any custom output tool can be used

• To output a `Document` as XML:

```java
XMLOutputter outputter = new XMLOutputter();
outputter.output(doc, System.out);
```

• For machine-consumption, pass optional parameters
  – Zero-space indent, no new lines

```java
outputter = new XMLOutputter("", false);
outputter.output(doc, System.out);
```
import java.io.*;
import org.jdom.*;
import org.jdom.input.*;
import org.jdom.output.*;

public class PrettyPrinter {
    public static void main(String[] args) {
        // Assume filename argument
        String filename = args[0];
        try {
            // Build w/ SAX and Xerces, no validation
            Builder b = new SAXBuilder();
            // Create the document
            Document doc = b.build(new File(filename));

            // Output as XML to screen
            XMLOutputter outputter = new XMLOutputter();
            outputter.output(doc, System.out);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
The DocType class

- A Document may have a DocType

```xml
<!DOCTYPE html PUBLIC
"-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

- This specifies the DTD of the document
  - It's easy to read and write

```java
DocType docType = doc.getDocType();
System.out.println("Element: " +
        docType.getElementName());
System.out.println("Public ID: " +
        docType.getPublicID());
System.out.println("System ID: " +
        docType.getSystemID());

doc.setDocType(
    new DocType("html", "-//W3C...", "http://..."));
```
The Element class

- A Document has a root Element:

```xml
<web-app id="demo">
    <description>
        Gotta fit servlets in somewhere!
    </description>
    <distributable/>
</web-app>
```

- Get the root as an Element object:

```java
Element webapp = doc.getRootElement();
```

- An Element represents something like web-app
  - Has access to everything from the open `<web-app>` to the closing `</web-app>`
Playing with Children

- An element may contain child elements

```java
// Get a List of direct children as Elements
List allChildren = element.getChildren();
out.println("First kid: " +
    allChildren.get(0).getName());

// Get all direct children with a given name
List namedChildren = element.getChildren("name");

// Get the first kid with a given name
Element kid = element.getChild("name");

// Namespaces are supported
kid = element.getChild("nsprefix:name");
kid = element.getChild("nsprefix", "name");
```

- `getChild()` may throw `NoSuchElementException`
Playing with Grandchildren

```
<linux-config>
  <gui>
    <window-manager>
      <name>Enlightenment</name>
      <version>0.16.2</version>
    </window-manager>
    <!-- etc -->
  </gui>
</linux-config>
```

- Grandkids can be retrieved easily:

```
String manager =
  root.getChild("gui")
    .getChild("window-manager")
    .getChild("name")
    .getContent();
```

- Future JDOM versions are likely to support XPath
Managing the Population

- Children can be added and removed through `List` manipulation or convenience methods:

```java
List allChildren = element.getChildren();

// Remove the fourth child
allChildren.remove(3);

// Remove all children named "jack"
allChildren.removeAll(
    element.getChildren("jack"));

// Add a new child
allChildren.add(new Element("jane"));

// Add a new child in the second position
allChildren.add(1, new Element("second"));
```
Making Kids

• Elements are constructed directly, no factory method needed

```java
Element element = new Element("kid");
```

• Some prefer a nesting shortcut, possible since `addChild()` returns the `Element` on which the child was added:

```java
Document doc = new Document(
    new Element("family")
        .addChild(new Element("mom"))
        .addChild(new Element("dad")
            .addChild("kidOfDad")));
```

• A subclass of `Element` can be made, already containing child elements and content

```java
root.addChild(new FooterElement());
```
Making the linux-config Document

• This code constructs the `<linux-config>` seen previously:

```java
Document doc = new Document(
    new Element("linux-config")
    .addChild(new Element("gui")
        .addChild(new Element("window-manager")
            .addChild(new Element("name")
                .setContent("Enlightenment"))
            .addChild(new Element("version")
                .setContent("0.16.2")))
    );
```
Getting Element Attributes

- Elements often contain attributes:

  ```
  <table width="100%" border="0">  </table>
  ```

- Attributes can be retrieved several ways:

  ```
  String value =
      table.getAttribute("width").getValue();

  // Get "border" as an int, default of 2
  int value =
      table.getAttribute("border").getIntValue(2);

  // Get "border" as an int, no default
  try {
    value =
        table.getAttribute("border").getIntValue();
  } catch (DataConversionException e) {
  }
  ```

- `getAttribute()` may throw `NoSuchAttributeException`
Setting Element Attributes

- Element attributes can easily be added or removed

```java
// Add an attribute
table.addAttribute("vspace", "0");

// Add an attribute more formally
table.addAttribute(
    new Attribute("prefix", "name", "value");

// Remove an attribute
table.removeAttribute("border");

// Remove all attributes
table.getAttributes().clear();
```
Element Content

• Elements can contain text content:

```xml
<description>A cool demo</description>
```

• The content is directly available:

```java
String content = element.getContent();
```

• And can easily be changed:

```java
// This blows away all current content
element.setContent("A new description");
```
Mixed Content

- Sometimes an element may contain comments, text content, and children

```html
<table>
    <!-- Some comment -->
    Some text
    <tr>Some child</tr>
</table>
```

- Text and children can be retrieved as always:

```java
String text = table.getContent();
Element tr = table.getChild("tr");
```

- This keeps the standard uses simple
Reading Mixed Content

- To get all content within an `Element`, use `getMixedContent()`
  - Returns a `List` containing `Comment`, `String`, and `Element` objects

```java
List mixedContent = table.getMixedContent();
Iterator i = mixedContent.iterator();
while (i.hasNext()) {
    Object o = i.next();
    if (o instanceof Comment) {
        // Comment has a toString()
        out.println("Comment: " + o);
    }
    else if (o instanceof String) {
        out.println("String: " + o);
    }
    else if (o instanceof Element) {
        out.println("Element: " +
                    ((Element)o).getName());
    }
}
```
The ProcessingInstruction class

- Some documents have ProcessingInstructions
  
  ```xml
  <?cocoon-process type="xslt"?>
  ```

- PIs can be retrieved by name and their "attribute" values are directly available:
  
  ```java
  ProcessingInstruction cp = 
  doc.getProcessingInstruction(  
    "cocoon-process" );
  
  cp.getValue("type");
  ```

- All PIs can be retrieved as a List with
  
  ```java
  doc.getProcessingInstructions();
  ```
  - For simplicity JDOM respects PI order but not the actual placement

- `getProcessingInstruction()` may throw
  
  `NoSuchProcessingInstructionException`
Namespaces

- Namespaces are a DOM Level 2 addition
  - JDOM always supports even with DOM Level 1 parsers and even with validation on!

- Namespace prefix to URI mappings are held in the Document object
  - Element knows prefix and local name
  - Document knows prefix to URI mapping
  - Lets Elements easily move between Documents

- Retrieve and set a namespace URI for a prefix with:

```java
String uri = doc.getNamespaceURI("linux");
doc.addNamespaceMapping("linux", "http://www.linux.org");
```

- This mapping applies even for elements added previously
Using Namespaces

- Elements have "full names" with a prefix and local name
  - Can be specified as two strings
  - Can be specified as one "prefix:localname" string

```java
kid = elt.getChild("JavaXML", "Contents");
kid = elt.getChild("JavaXML:Contents");
kid = elt.getChild("Contents");
```

- Allows apps to ignore namespaces if they want.

- Element constructors work the same way.
List Details

• The current implementation uses LinkedList for speed
  – Speeds growing the List, modifying the List
  – Slows the relatively rare index-based access

• All List objects are mutable
  – Modifications affect the backing document
  – Other existing list views do not see the change
  – Same as SQL ResultSet, etc.
Exceptions

• **JDOMException** is the root exception
  – Thrown for build errors
  – Always includes a useful error message
  – May include a "root cause" exception

• Subclasses include:
  – NoSuchAttributeException
  – NoSuchElementException
  – NoSuchProcessingInstructionException
  – DataConversionException
Future

• There may be a new high-speed builder
  – Builds a skeleton but defers full analysis
  – Use of the List interface allows great flexibility

• There could be other implementations outside org.jdom
  – They should follow the specification
  – The current implementation is flexible
  – We don't expect alternate implementations to be necessary
Get Involved

• Download the software
  – http://jdom.org

• Read the specification
  – Coming soon

• Sign up for the mailing lists (see jdom.org)
  – jdom-announce
  – jdom-interest

• Watch for JavaWorld and IBM developerWorks articles
  – http://www.javaworld.com

• Help improve the software!