BY JASON HUNTER

JDOM in the Real World, Part 3

This installment shows how to use JDOM inside your own Java programs.

In the first two parts of this series (*Oracle Magazine*, September/October 2002 and November/December 2002), I introduced JDOM, an open-source library for Java-optimized XML manipulation, and gave an overview of its design, features, and interfaces. In this article, I show two practical applications developed using JDOM to handle RDF site summary (RSS) news feed processing. Note that due to space considerations, some of the code referenced in this article appears only in the online version of this article, at otn.oracle.com/oramag/oracle/03-mar.

The first application shows how to build a server-side component to display RSS feeds on a Web site as a list that can be plugged into any page. RSS is one of the most widely used applications of XML, helpful for everything from announcing new Web log entries to sharing links to new magazine articles. With practical code examples, I'll show you how JDOM helps parse the RSS XML.

The second application is a client-side component that consumes RSS feeds and e-mails interested parties when new content appears. It's configured using an XML file, so I'll show you how to use JDOM at startup to read configuration information and to manipulate the incoming RSS.

RSS AND MEERKAT

RSS is a Web syndication format based on XML. It's a simple format allowing Web sites to publish a list of links with titles and descriptions that other sites or third-party clients can access and use. Using RSS, newspapers spread links, Web loggers (called bloggers, for short) announce their posts, and everyone keeps up with what's new. Interested in JDOM? With RSS, you can track the latest JDOM links at www.servlets.com/blog/index-jdom.xml. It's not a Web page; it's an XML file containing structured links. Listing 1 shows an RSS format example that covers JDOM.

The RSS name stands for RDF site summary, rich site summary, or really simple syndication, depending on whom you ask. There are also several dialects of RSS in active use, varying from simple and easy (the older forms) to extensible and complicated (the newer forms).

Meerkat, launched by the O'Reilly Network in 2000 as a free service, pulls together the disparate RSS feeds on the internet and makes them centrally available and searchable. Do you want to know about every article on servlets posted in the last 21 days? You can query the Meerkat Web interface or, once you learn the query string pattern, make a direct request.

Meerkat pulls RSS from syndicators and outputs to searchers in formats including RSS, XML, and HTML. For my example, I'll be consuming Meerkat content in its custom XML format, which is similar to RSS but adds extra category, channel, and time stamp information. The following is a sample Meerkat query:

```
http://www.oreillynet.com/meerkat/?t=21
DAY&c=5812&_fl=xml
```

The query string parameters request the last 21 days of channel 5812 (OTN) in XML format. Listing 2 shows the report generated by the sample query.

ON THE SERVER SIDE: EMBEDDING RSS IN A WEB PAGE

The following application pulls RSS data from either a local file or a remote URL and displays it as a formatted list in a section of an HTML page. The application must meet the following requirements: support both RSS 0.91 (a widely used form, which is much simpler than 1.0) and the Meerkat XML format, allow nonprogrammer configuration of the generated HTML, and cache the content with updates every 30 minutes. You can see the code in use on the front page of Servlets.com; the "What's New" section is driven by the http://www.servlets.com/blog/index-short.xml RSS data file. The data file itself is autogenerated by the MovableType blog tool.

For implementation tools, I'm going to use JDOM to read the XML, a servlet container or application server to serve the content, and the "Tea" framework to handle the display. Odds are, you haven't heard of Tea. Disney created the Tea framework for use on its high-traffic sites such as ESPN.com and ABCNews.com, which are constantly updated, and the company released it under an open-source license a couple of years ago. Tea supports an elegant development model

codeLISTING 1: RSS format example for JDOM content

```
<?xml version="1.0"?>
<rss version="0.91">
  <channel>
    <title>Servlets.com Weblog</title>
    <link>http://www.servlets.com/blog/</link>
    <description>Java, Open Source, XML, Web Services, and
                 (gasp) .NET</description>
    <language>en-us</language>

    <item>
      <!— 2002-08-31 22:54 —>
      <title>JDOM in Oracle Magazine</title>
      <description>The Sept/Oct issue of Oracle Magazine has an introductory article on JDOM
       covering Beta 8. The article is freely available online. It provides good coverage
       for newbies as well as some historical stories and advanced tricks.</description>
      <link>http://www.servlets.com/blog/archives/000031.html</link>
    </item>

    <item>
      <!— 2002-03-29 20:42 —>
      <title>XPath Class in JDOM</title>
      <description>JDOM now includes built-in XPath support with the
       org.jdom.xpath.XPath class. Here's how it works.</description>
      <link>http://www.servlets.com/blog/archives/000019.html</link>
    </item>

    <!— ... Many more items ... —>

  </channel>
</rss>
```

codeLISTING 2: Report from Meerkat query

```
<?xml version="1.0"?>
<!DOCTYPE meerkat_xml_flavour SYSTEM
"http://meerkat.oreillynet.com/dtd/meerkat_xml_flavour.dtd">

<meerkat>
  <title>Meerkat: An Open Wire Service</title>
  <link>http://meerkat.oreillynet.com</link>
  <language>en-us</language>
  <!— ... —>

<story id="1026581">
    <title>Logging and Debugging with Oracle9iAS Containers for J2EE</title>
    <link>http://otn.oracle.com/tech/java/oc4j/htdocs/oc4j-logging-debugging-
     technote.html</link>
    <description>This technical note describes the debugging and logging options available
     in OC4J and how to make use of them in both standalone and complete Oracle9iAS
     environment.</description>
    <category>Server: Database</category>
    <channel>Oracle Technology Network</channel>
    <timestamp>2002-10-17 10:49:00</timestamp>
  </story>

<story id="1026580">
    <title>Compete for the title of OTN TopCoder Wednesday at 6:30 pm PDT</title>
    <link>http://otn.oracle.com/topcoder</link>
    <description>The time has come! Compete against Java programmers in the worldwide OTN
     community for the title of OTN TopCoder in a new online programming competition. The
     coding starts promptly at 6:30 pm PDT Wednesday. All participants need to reserve their
     spots in the Competition Arena during the 3 hours prior to the start of the
     competition.</description>
    <category>Server: Database</category>
    <channel>Oracle Technology Network</channel>
    <timestamp>2002-10-16 21:08:00</timestamp>
</story>

<!— ... —>
</meerkat>
```

and an advanced integrated development environment (IDE) called Kettle and compiles Tea pages directly to Java bytecode (no intermediary servlet class is necessary). It works with any servlet container.

Listing 3 shows our first class, `Story`. It holds information for one story: id, title, link, description, category, channel, and time stamp.

Listing 4 (available with the online version of this article) contains the main logic class, `MeerkatContext`. It acts as a back end for Tea templates. Every public method in this class gets exposed to the Tea front end. The class has just one public method, `getStories (String)`, which takes a URL and returns an array of `Story` objects. The class supports URLs with either "http:" or "file:" protocols and supports content in either RSS or Meerkat XML format. The class caches each URL's content in the `storyCache` map, with the corresponding time stamps in the `timeCache` map. Only if the `storyCache` is empty or the time stamp is greater than "halfHour Ago" will new content be fetched. Notice how the class caches content internally and doesn't rely on external crontab entries. Crontabs, while heavily used, are external to the Web server and don't move easily between machines. Thus, they should be avoided whenever possible.

To read the content, the `Meerkat Context` class gets an `InputStream` with `getStoryStream()` and then in `getStory Document()` uses `SAXBuilder` to build a JDOM representation of the content. The `getStoryList()` method quickly walks the JDOM tree converting JDOM `Element` objects into `Story` objects. The `getChildText()` method is heavily used to directly read child text content.

There are two code paths, one for Meerkat files and one for RSS files. The RSS path loads less information because less is available, although you'll notice that the `getTimestampComment()` method supports reading a time stamp out of a leading XML comment. This method is a custom RSS extension that helps Servlets.com display a date at the bottom of entries.

The meerkat.tea template file appears in Listing 5. This file pulls on the `Meerkat` class back end to render customizable

HTML. The template accepts a URL to its "constructor" that gets passed using a request parameter (/meerkat.tea?url=xxx) or another template's direct call. If there's no URL, it sets the default URL to the ONJava feed. The template then customizes how dates and null values should be displayed. Next, it makes the call into the back end to retrieve the `Story` objects. If the array is null or empty, an error is printed in the page; otherwise the template loops over the array, printing HTML for each entry. Templates have access to the bean properties on all objects returned by functions using the syntax `<% bean.property %>`.

The following code concludes our application with a portion of an index.tea page that would call on meerkat.tea.

```
<!— page snippet —>
<b>What's New at OTN</b><br />
<% call meerkat(
   "http://otn.oracle.com/ws/otnrss.xml")
%>
```

Any Tea template page can add an automatically updating RSS or Meerkat data feed with one simple `<% call meerkat(url) %>` command. Behind the scenes, JDOM takes care of the XML parsing, Tea takes care of the templating, Meerkat handles the collection, and servlets glue it all together.

codeLISTING 3: Story class

```
import java.util.Date;

public class Story {
  private String id;
  private String title;
  private String link;
  private String description;
  private String category;
  private String channel;
  private Date timestamp;

  public Story(String id, String title, String link, String desc,
               String category, String channel, Date timestamp) {
    this.id = id;
    this.title = title;
    this.link = link;
    this.desc= desc;
    this.category = category;
    this.channel = channel;
    this.timestamp = timestamp;
  }

  public String getId() { return id; }
  public String getTitle() { return title; }
  public String getLink() { return link; }
  public String getDescription() { return description; }
  public String getCategory() { return category; }
  public String getChannel() { return channel; }
  public Date getTimestamp() { return timestamp; }
}
```

ON THE CLIENT SIDE: BLOG ALERT

My sample client-side application, which I call Blog Alert, consumes RSS and Meerkat feeds and, after detecting a change, sends e-mail notifications to interested parties. It supports instant notification or allows updates to be queued for daily or weekly digest mails. The e-mails can have customized subjects and footers with substitution rules allowing the insertion of title, category, and date information. Listing 6 shows what e-mail notifications look like.

Blog Alert runs off an external XML-based configuration file that provides the mail host information (for sending mails) and a set of mailing entries. Each mailing has a type (instant, daily, weekly), an RSS source, an e-mail sender address, an e-mail recipient address (such as a mailing list), a subject, and an optional footer. Subjects and footers support `%title`, `%category`, and `%date` substitutions. Listing 7 (available with the online version of this article) shows an example that gives instant notice of Servlets.com changes, weekly notices of JDOM changes, and daily notices of XMLHack changes.

The application code starts in Listing 8 (available with the online version of this article), which shows the `BlogAlert` class, holding the `main()` entry point and coordinating the processes. It starts by loading the configuration file (blogalert.xml) into a `Config` object.

Then the `handleMailing()` method follows a multistep algorithm. It gets the current list of stories from the RSS feed, loads the previous list of stories from disk, stores the current list to be the previous list for the next time `handleMailing()` is called, and then determines what new stories have appeared. Next, the logic determines what stories were pending (new but not sent out yet) and removes from the current list any in the pending list. Then `handleMailing()` creates and sets a new pending list and looks to see if it's time to send the e-mail. If so, it sorts the pending list and sends an e-mail.

Listing 9 (available with the online version of this article) shows the `Config` class, which is responsible for loading the blogalert.xml data file and returning the information with the `getMailHost()` and `getMailings()` methods. Almost the entire class consists of JDOM calls.

The `Mailing` class in Listing 10 (available with the online version of this article) represents each configured mailing item and makes available to the main logic the mailing's id, type, RSS location, sender, recipient, subject, footer, time stamp, current stories, previous stories, and pending stories. The class also has logic to determine if it's time to send and has a `send()` method that initiates the sending.

Notice that the previous and pending story lists are maintained as external XML files (mailingname.lastrss and mailingname.pending) in the `Mailing` class. The reading and writing of these lists is managed by the `Story` class—discussed later—an enhanced version of the class shown earlier running on the server side. For a time stamp, the `Mailing` class uses a mailingname.timestamp file storing a string representation of the time. It's separate and readable because, as an administrator, I find it sometimes useful to tweak the time—for example, to force-

codeLISTING 5: meerkat.tea template

```
<% template meerkat(String url)
storyURL = "http://www.oreillynet.com/meerkat/?t=21DAY&c=5136&_fl=xml";
if (url != null) { storyURL = url; }

// Specify how time/numbers are formatted
dateFormat("d MMMM yyyy")
nullFormat("")

stories = getStories(storyURL);
if (stories == null or stories.length == 0) {
  "<i>No stories currently available</i>"
}
else {
  foreach (story in stories) {
%>
    <font face="Arial, Helvetica, sans-serif" size="-1"> <b>
    <a href="<% story.link %>"
    <% if (findFirst(story.link, "servlets.com") == -1) {
         'target="meerkat"'
       }
    %>
    > <% story.title %>  </a></b><br /> <% story.description %>
    <i><font size="-2"><% story.timestamp %></font></i> <p /> </font>
<%
    }
  }
%>
```

codeLISTING 6: BlogAlert generated e-mail example

```
Subject: Servlets.com: Blog update for week of July 25th, 2002
   From: blogalert@servlets.com
     To: jhunter-blog@dorothy.denveronline.net

Open Java at JavaOne
http://www.servlets.com/blog/archives/000020.html

  The Apache-Sun agreement caused quite a buzz at JavaOne this year.
  Here's my behind-the-scenes look at how the public events unfolded.

More File Upload Improvements
http://www.servlets.com/blog/archives/000027.html

  There's a new update to the com.oreilly.servlet library available.
  This release polishes the pluggable renaming logic to provide...

–
To subscribe or unsubscribe, go to
http://www.servlets.com/lists/subscribe.html
```

send a weekly message. The `isTimeToSend()` method looks at the mailing type and how much time has elapsed and returns a boolean. The `send()` method uses the `MailSender` class to send e-mails and a `WrapFormat` class to handle consistent indenting. (These two classes are interesting but orthogonal to the point of this article; for their code, go to otn.oracle.com/oramag/oracle/03-mar.)

The last class I'll examine, in Listing 11 (available with the online version of this article), is a grown-up version of the `Story` class, based on the simple `Story` class I presented earlier. Now it's enhanced with methods to read and write story lists to and from files and includes a `Comparable` interface to support date-based sorting.

The `setStoryList(List, File)` method contains the bulk of the new `Story` class logic. This method constructs an in-memory, JDOM-modeled, RSS-style document from the passed-in Story list and uses JDOM's `XMLOutputter` to pretty-print the information for display to a local file. These files support the retrieval of previous and pending entries. I could have kept the original RSS or Meerkat feed around, but given how easy it is to use JDOM to create XML output, it's not worth the bother.

Run the `BlogAlert` class periodically and watch as `BlogAlert` pulls down RSS feeds, examines the feeds for new material, and fires notifying e-mails when appropriate. You'll see how JDOM helps read the configuration file, handles the incoming RSS feeds, and supports the outgoing RSS storage duties.

JDOM is a simple and straightforward way to handle XML files with Java. Written in and for Java, JDOM provides you with an intuitive way to read, write, and manipulate XML documents. Best of all, JDOM has been published under an open-source Apache-style license with a wide user and developer community that has developed the application-programming interface (API) to solve real-world problems. JDOM is also in the process of going through Sun's Java Community Process (JCP) as a Java Specification Request (JSR)—the first open-source project to become a JSR.

YOUR TURN

This article walks you through two practical uses of XML and JDOM for manipulating an RSS news feed. They're inspired by my real-life needs managing the Servlets.com site, and I hope the use of JDOM in these examples can help you with your real-life needs too. ■

__Jason Hunter__ (jasonhunter@servlets.com) is a consultant, a publisher of Servlets.com, and a vice president of the Apache Software Foundation. He also holds a seat on the Java Community Process (JCP) Executive Committee.

nextSTEPS

DOWNLOAD code examples
www.servlets.com

JDOM
jdom.org

Meerkat
www.oreillynet.com/meerkat

RSS
www.oreillynet.com/rss

Servlets
www.servlets.com

Tea
teatrove.sourceforge.net

MovableType
www.movabletype.org

READ online-only article content
otn.oracle.com/oraclemagazine