Broadview
WWW.BROADVIEW.COM.CN

国际技术期刊中文精华合集

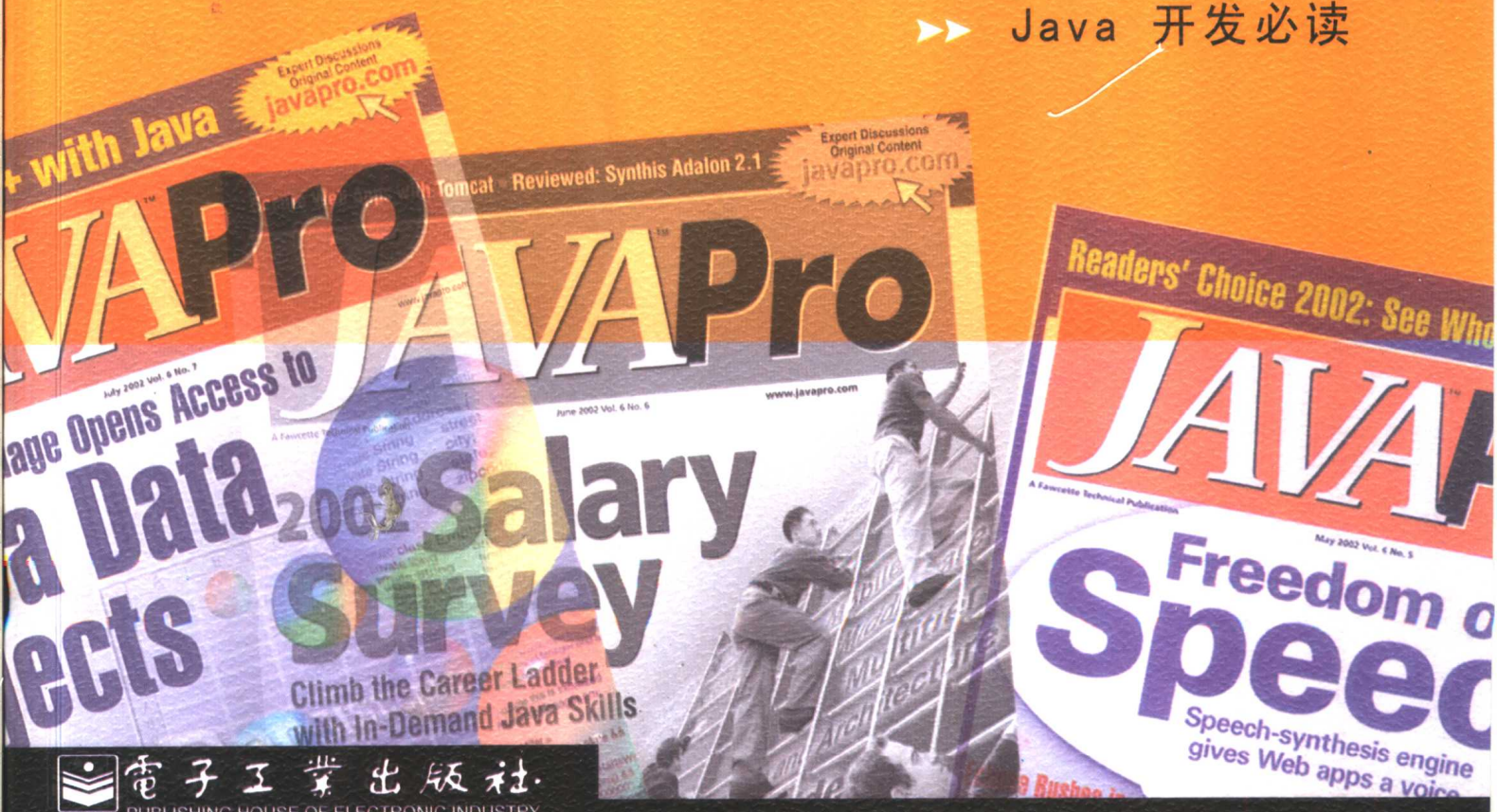
JavaPro

2002-2003 中文精华合集 [第1辑]

Fawcette Technical Publications 独家授权

国际技术期刊合集编委会 编译

- Java 名刊精华
- Java 技术荟萃
- Java 开发必读



電子工業出版社
PUBLISHING HOUSE OF ELECTRONIC INDUSTRY
http://www.phei.com.cn

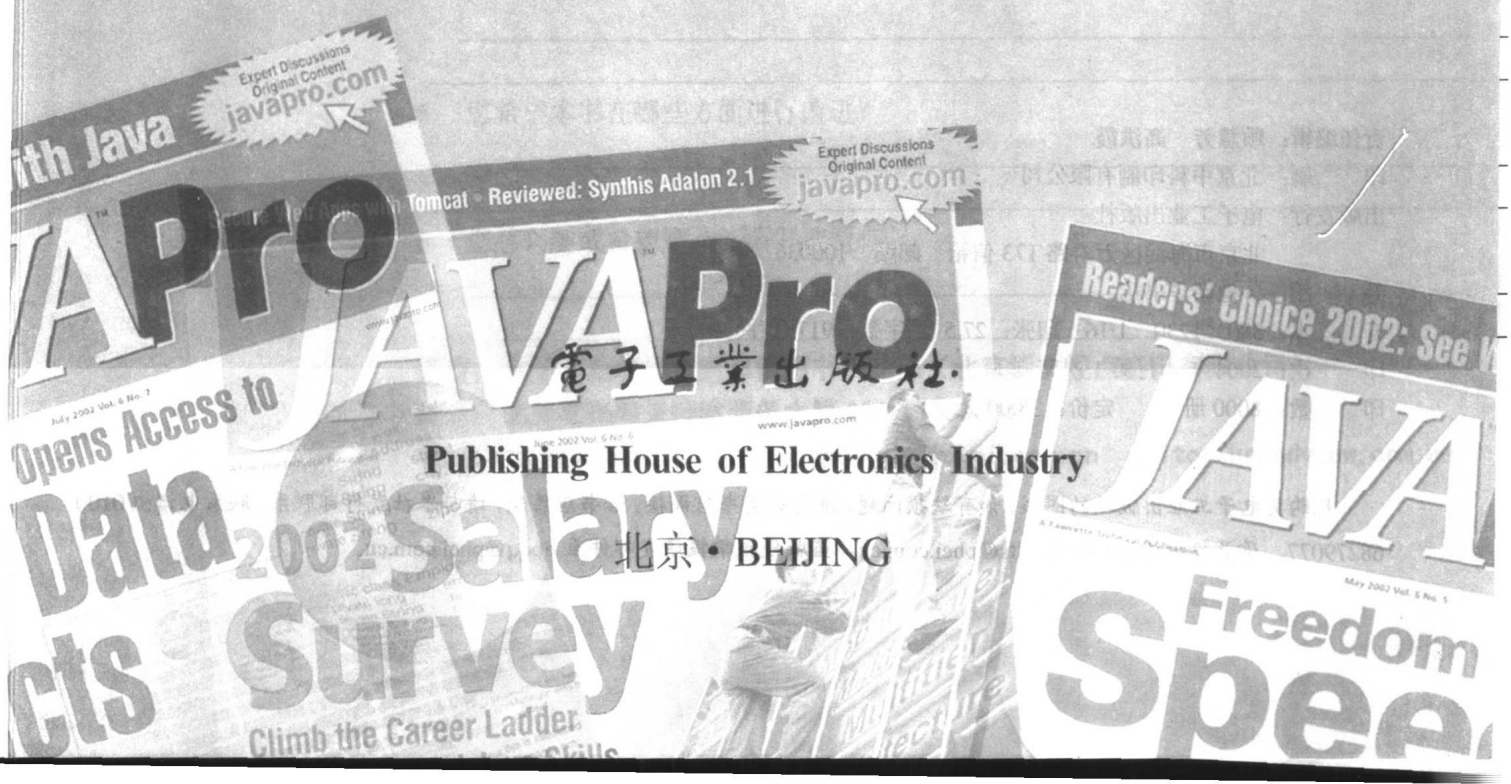
国际技术期刊中文精华合集

JavaPro

2002—2003 中文精华合集[第1辑]

Fawcette Technical Publications 独家授权

国际技术期刊合集编委会 编译



電子工業出版社
Publishing House of Electronics Industry
北京·BEIJING

内 容 简 介

本书汇集了国际著名技术媒体 Fawcette Technical Publications 旗下技术期刊 Java Pro 2002—2003 年度精华文章数十篇。主要围绕 Java 编程语言，涉及 J2ME，J2SE 和 J2EE 等应用平台，由各自领域内的一流专家撰写，其内容包括了从编程技术到项目组织管理，从工具使用技巧到新技术剖析的各个方面，技术含量丰富，观点权威，涵盖面广。

本书不仅适合专业软件开发者阅读学习和参考，同时也适合广大技术爱好者、在校学生和教师阅读学习。

版权贸易合同登记号　图字：01-2003-7711

图书在版编目（CIP）数据

Java Pro 2002—2003 中文精华合集. 第 1 辑 / 美国发赛特技术出版集团著；国际技术期刊合集编委会编译. —北京：电子工业出版社，2004.4
（国际技术期刊中文精华合集）
书名原文：Java Pro
ISBN 7-5053-9691-9

Ⅰ.J...　Ⅱ.①美...②国...　Ⅲ.Java 语言—程序设计—文集　Ⅳ.TP312-53

中国版本图书馆 CIP 数据核字（2004）第 013277 号

责任编辑：顾慧芳　高洪霞
印　　刷：北京中科印刷有限公司
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　邮编　100036
经　　销：各地新华书店
开　　本：880×1230　1/16　印张：27.5　字数：917 千字
印　　次：2004 年 4 月第 1 次印刷
印　　数：8000 册　　定价：38.00 元

凡购买电子工业出版社的图书，如有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系。联系电话：（010）68279077。质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

前　言

这是一本开发技术文集，其中收录了国际著名开发技术期刊 Java Pro 杂志 2002 年至 2003 年发表的一批精华文章。

程序员作为一个职业群体，既需要系统的学习基础和经典的理论，不断的在实际项目中磨炼自己，也需要充分交流，保持对新技术和新思想的敏感，向同行中的佼佼者取经。在这个方面，技术期刊能够起到别的形式无法替代的作用，而国外的高水平技术期刊在这方面更是有着辉煌的传统。技术期刊中的文章，一方面能紧跟最新的技术趋势，另一方面来自名家之作，高屋建瓴，能够反映比较高的技术水平和崭新的视角。在 IT 产业发展的历史上，技术期刊起到了重要的推动作用。很多重要的新技术和新思想，最初都是发表在期刊上，进而对整个产业界产生深刻的影响。因此，作为一名不断追求卓越的程序员，应当经常阅读高水平的国际技术期刊。

Fawcette Technical Publications 是美国著名的 IT 技术媒体和出版集团，旗下拥有著名的 VSLive!技术大会和 Visual Studio Magazine，Java Pro 杂志等技术期刊。FTP 集团的技术期刊拥有一大批世界知名的技术作家，如 VB 之父 Alan Cooper,.NET 专家 Dino Esposito，Java 专家 James Cooper 等，一向以文章质量高、观点权威新颖、技术含量丰富著称。

自从 Sun 公司 1995 年向世界发布 Java 技术以来，在短短几年时间内，Java 迅速被工业界、教育界和学术界所接受，成为最主流的软件开发、技术研究和企业应用平台之一。目前，Java 在国内已经获得了广泛的应用，广大 Java 开发者急需通过阅读高水平、国际化的 Java 技术刊物迅速提升自己的技术水平。

Java Pro 杂志自 1996 年创刊，一路发展壮大，在国际 Java 技术界享有盛誉。此次我们本着为广大专业开发人员和学习者服务的宗旨，引进 Java Pro 杂志，将其精华内容集结成书，翻译出版。在选编时，我们尽可能注意了技术期刊内容的特点，既选择了一些最新、最前沿的技术文章，也选择了一些立意深远、具有长期意义的文章。希望能够使这本精华合集既能够满足开发者平时学习工作的需要，也能够具有一定的保存价值。

全书共 13 篇，分别收录了 Java Pro 杂志 2002 年 3 月-2003 年 2 月的各期内容，以及 2003 年 3-9 月的文章精选。

本书由“国际技术期刊合集编委会”主持，韩伟、王亮、李原野选编翻译。

由于时间仓促及水平有限，失误和疏漏之处在所难免，恳请读者的不吝赐教。

国际技术期刊合集编委会

2004 年 2 月

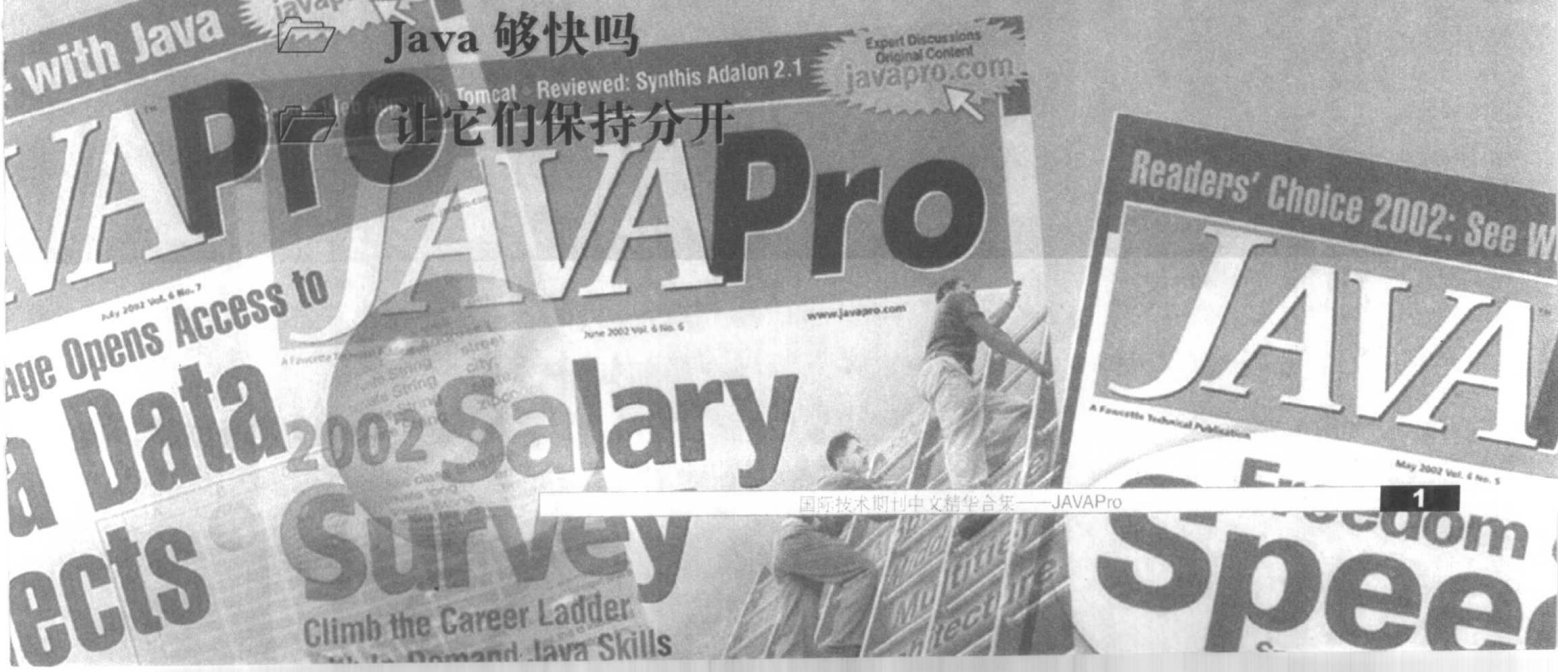
目 录

Java Pro

第一篇

2002年3月

- 另一个视角看 XML
- 分割的文档
- 获取 JMS 消息
- 监测你的 Java 对象
- Java 够快吗
- 让它们保持分开



另一个视角看 XML

Visual XBeans 提供了一个构建基于 XML 的图形用户接口的模块化方法

● John B. O'Donahue

Visual XBeans 设计用于显示 XML 文档的 JavaBeans。JavaBeans 将 XBeans 底层概念引入 GUI 环境，利用 Swing 的组件，将 XML 文档显示为 JTree，JTable 或其他 Swing 组件。与在我的文章《JavaBeans 的新面孔》(Java Pro，2001 年 10 月刊)中提及的基础 XBeans 相似，可视化 XBeans 也是互相连接的，它们通过 Java 代理事件模型来接收、处理并传递标准的文档对象模型（DOM）文档。获得有关 XBeans 的资料、源代码及相关描述的一个很好的网站是 xbeans.org，在那里你还能读到 Bruce Martin 关于这个方面的原创性权威文章《用 XBeans 生成分布式应用程序》。

本文中我们将使用 Xerces 分析器来处理 DOM 文档。Xerces可以从http://xml.apache.org/xerces-j下载。只要对代码进行少量修改，你也可以使用 JDOM（可以从 http://www.jdom.org/ 处下载）。虽然 JDOM 是特别针对 Java 设计的，使用起来可能更直观，但 Xerces 是 DOM 的一个完全实现版本，所以仍是目前公认的标准。

XBeans的一个最重要的特点是它们之间可以互相通信。XBeans 通过 Java 事件模型通信，将 DOM 文档作为事件对象传输。通过实现 DOMSource 和 DOMListener 两个接口中的一个或两个来处理通信。

DOMSource 接口定义了两个方法：setDOMListener 和 getDOMListener，分别用来设置和获得 DOMListener，XBeans 将接收其输出：

```
public interface DOMSource {

    public void setDOMListener(
            DOMListener DOMListener);

    public DOMListener
            getDOMListener();
}
```

DOMListener 接口定义了一个单一的方法，documentReady(DOMEvent evt)。该方法由源XBeans调用，将 XML 文档作为事件对象传输：

```
public interface DOMListener
      extends EventListener {

      public void documentReady(
            DOMEvent evt) throws
            XbeansException;
}
```

DocumentReady()方法通常将 XML 文档传递到 processDocument()方法来为 XBeans 执行实际的文档处理。仅仅实现DOMSource接口的XBeans通常采用一个无参数的 processDocument()方法，从一些数据源，如数据库或文件等获得数据，生成 XML 文档。processDocument()在实例化方法中被调用。

XBeans通常由一个中间对象操纵。该对象实例化XBeans，执行必要的初始化，用setListener()方法建立互连，并通过调用源 Bean 的 processDocument()方法来启动传递链：

```
String source = "SourceBean";
String listen = "ListenerBean";
DOMSource srcBean =
      (DOMSource)beans.instantiate(
      source);
DOMListener listenerBean =
      (DOMListener)beans.instantiate(
      listen);
srcBean.setFName("data.xml");
srcBean.setListener(
      listenerBean);
srcBean.processDocument();
```

processDocument()方法将 data.xml 文件分解成一个 DOM 文档，然后调用监听器的 documentReady()方法来触

发DOMEvent事件，从而将文档传递到监听器。监听器的documentReady()方法反过来调用自己的processDocument()方法，执行必要的本地处理，然后再将处理后的文档传递到链中的下一个Bean。

因此，processDocument()方法既可以在外部调用，就像在本例子中一样，由中间对象调用从而启动处理链；或者在内部调用，这时，DOMListener的documentReady()方法由DOMSource Xbean调用（参见图1）。

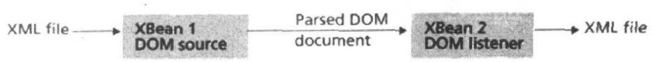


图1 数据活动图

通常，我们在基础XBeans类中隐藏其实现，然后从基类派生出新类，以提供所需的功能（见程序清单1）。

每当我们使用XBeans时，我们都可能使用一个DOMParserBean对象，或者用一个与其功能等价的对象来让信息流动起来。DOMParserBean是一个简单但却很有用的DOMSource XBean，我们可以用它从一个URL处获得XML文档，将XML文档分解为DOM文档，然后发送到一个DOMListener对象（见程序清单2）。在我们的第一个例子中，我们使用方法setUrlName(String UrlName)将DOMParserBean指向moreover.com网站的最顶层新闻特写页面，那里是XML文档的大宝库。然后我们生成一个可视化的XBean对象作为DOMListener。

可视化XBeans很简单，它接收一个XML文档，用该文档作为图形用户接口的一部分。正如我们从一个基础Xbeans类派生出DOMParserBean类一样，我们可从一个基础XMLGUIBean类派生出相当多的可视化XBeans类。

XMLGUIBean是我们从JPanel派生的最重要的XBeans基础类（见程序清单3）。它实现了DOMListener接口，所以能够接收它要显示的XML文档。为了处理像鼠标点击等图形用户接口事件，它也实现了DOMSource接口，所以它能将一个XML文档发送到监听器以报告鼠标点击事件。这个监听器通常是装载这些Bean的中间对象类。

XMLGUIBean一个显而易见的例子是XMLTreeBean（见程序清单4），这个类接收XML文档并在一个JTree中显示该文档内的树形结构。这样，只要简单地拖动树节点，然后在希望的地方放下来，就对这个JTree实现了编辑。再加几行代码就能传播这些改动，从而编辑JTree下面的XML文档。

像大多数其他的可视化XBeans一样，XMLTreeBean也是依据JPanel设计的，所以它可以作为一个完整图形用户接口的一部分放置在一个JFrame上。

我们只要用一个简单的中间对象，便可将DOMParserBean连接到XMLTreeBean，就能在JTree中显示文档：

```
DOMParserBean parser =
      (DOMParserBean)Beans.instantiate
      (null,
      "xbeans.XBeans.DOMParserBean");

XMLTreeBean tree =
      (XMLTreeBean)Beans.instantiate
      (null,
      "xbeans.GUIBeans.XMLTreeBean");

parser.setUrlString(args[0]);
parser.setDOMListener(tree);

parser.processDocument();

frame.getContentPane().add(
      tree,BorderLayout.WEST);
```

正如你所看到的那样，XMLTreeBean运转起来了（见图2），不过要是我们只依靠默认的TreeCellRenderer，运行结果看起来并不怎么样。如果为每篇文章显示其大字标题而不是标签名字“article”，看起来就会好得多。我们用一个定制的TreeCellRenderer来实现这一点（见程序清单5）。

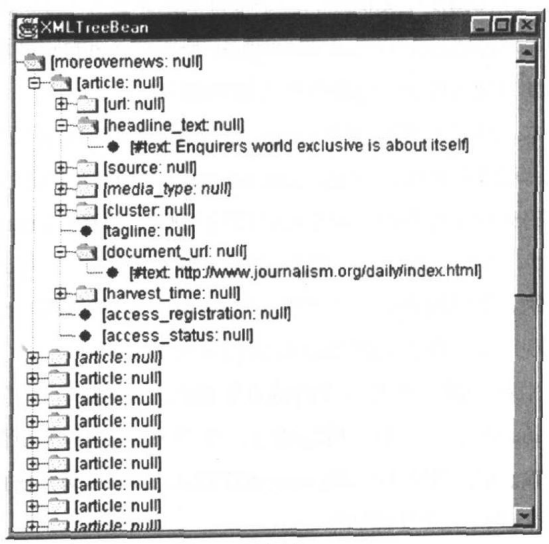


图2 树中的Bean

定制一个TreeCellRenderer最简单的方法是从DefaultTreeCellRenderer派生。TreeCellRenderer的任务是返回一个适于渲染给定树节点的JLabel。为了定制DefaultTreeCellRenderer，我们只要用自己的方法来重载getTreeCellRendererComponent()方法，该方法将创建一个更好的JLabel。

传递给getTreeCellRendererComponent()的一个参数是正在被渲染的树节点的UserObject。当然，在我们的例子里，

UserObject是一个XML节点。我们只要将其传给我们自己建的NodeLabel()方法，该方法便会检查这是一个Element还是一个TextNode。如果是一个TextNode，就用getNodeValue()方法生成一个JLabel；如果是一个Element，就使用默认的getTagName()方法。

最后一步是看看：tagName是不是“article”。如果是，就用getElementsByTagName()方法找到元素“headline_text”，然后在JLabel中使用该元素的textNode。最终结果如图3所示。

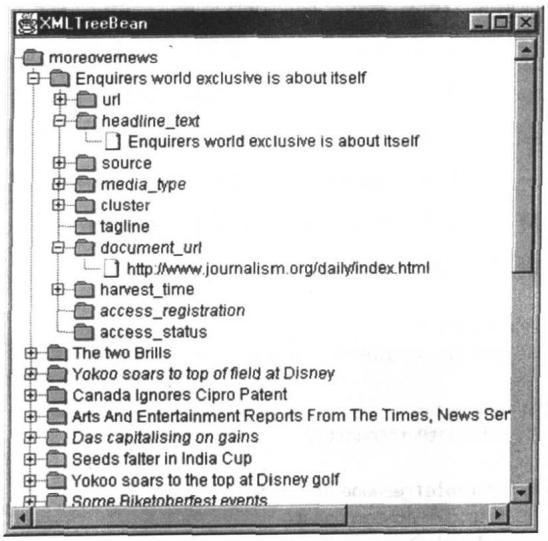


图3 使用的窍门

现在，加上一个鼠标事件处理监听器(MouseListener)如何？这样当用户单击一个TreeNode时就可以做些更有趣的事情。我们从MouseAdapter派生出这个MouseListener，并实现DOMSource接口(见程序清单6)。换句话说，我们生成了一个一般化的XMLMouseListenerBean。我们的mouseClicked()方法获得了与被点击的TreeNode关联的XML节点，并将其传到sendXmlMsg()方法。

在sendXmlMsg()里，我们为这个消息生成一个新的DOM文档。然后生成一个标签名为“XMLTREE_MSG”的新根节点，为DOMListener指明消息来源。我们也为根节点加上了一个鼠标按键属性和点击次数属性。下一步，导入被点击的XML节点，并附在根节点上，生成一个包含了被点击节点所有已知信息的文档。

这些鼠标点击生成的文档非常简单。我们要的只是一个包含嵌套在根节点下的被点击节点的文档，以告诉中间对象该文档的来源(为了清晰，这个例子做了删减。moreover.com上的原文有更多的信息)：

```
<?xml>
<XMLTREE_MSG CLICKS="2"
        RIGHTCLICK="true">
        <article id="_24637971">
                <url>
                        http://c.moreover.com/?x1234
                </url>
                <headline_text>
                        Israel strikes at Gaza
                </headline_text>
                <source>BBC</source>
                <harvest_time>
                        Sep 15 2001 10:46AM
                </harvest_time>
        </article>
</XMLTREEMSG>
```

对一个鼠标点击产生一个XML消息的主要原因是，这样我们就有一个对应用程序完全透明的事件处理器，我们可以在别的任何地方再使用它。这就是使用XBeans最大的一个好处：如果编码正确，你无需改动，或只要派生新的子类，就可以在很多地方使用它。

为了与Bean容器的中间对象的角色保持一致，我们让中间对象成为一个DOMListener对象，让它通过分解来自MouseListenerBean的XML消息和装载选定的文档来处理鼠标点击事件(中间对象类在这里可用)。

正如我们所看到的那样，为有效使用XBeans，中间对象起了非常重要的作用。在这个例子中，我们已经看到中间对象是怎么起作用的：

- 实例化一个DOMParserBean。
- 实例化一个XMLTreeBean，它装载了我们定制的XMLTreeCellRenderer。
- 添加一个XMLMouseListenerBean到XMLTreeBean。
- 将这个XMLTreeBean作为XMLMouseListenerBean的DOMListener。

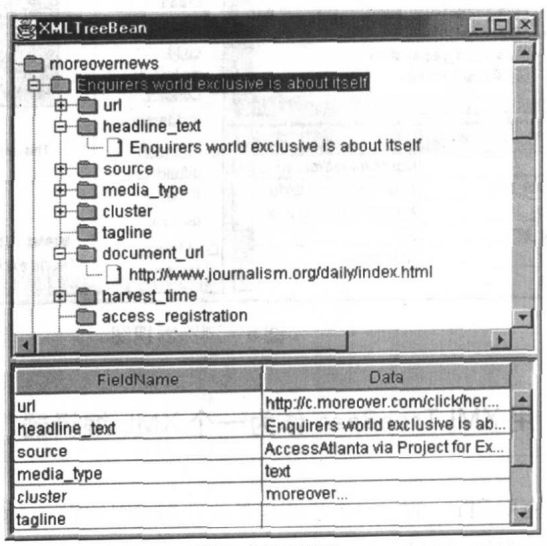


图4 展示XML

现在修改中间对象来分解鼠标点击文档，并产生相应的显示。因为我们有一个点击次数属性，故我们用这个属性决定是在表格里显示文档，还是请求它所涉及的HTML文档并在浏览器中显示。

我们用鼠标单击来生成一个表格，用双击运行浏览器，装载一个XMLTableBean或JEditorPane对象。熟悉IDE包的用户会很了解这样两种选择的显示(见图4)。

我们的XMLTableBean(见程序清单7)比XMLTreeBean简单得多，因为我们所要做的就是把文档分解存放在一个或多个Vector以代表一行数据。使用基于Vector的JTable是因为在装载一个Vector前不需要知道其大小，这样我们可以略过空的XML节点，而不用先在一个循环中计算有内容的节点的数目，再在另一个循环里装载节点树组。

一个更复杂的XMLTableBean会在每行排列多篇文章，并用元素标签名字作为列表头。

显然，下一步是修改中间对象来使用文档的内容，从服务器请求新闻特写，并作为一个HTML文档在JEditorPane里显示。为了做到这一点，我们简单地获得URL元素的NodeValue属性，传给JEditorPane的setPage()方法。

我们现在有了一个简单的图形用户接口，便可让我们从moreover.com获得一系列新闻特写，并选择其中一个在浏览器中显示（见图5）。

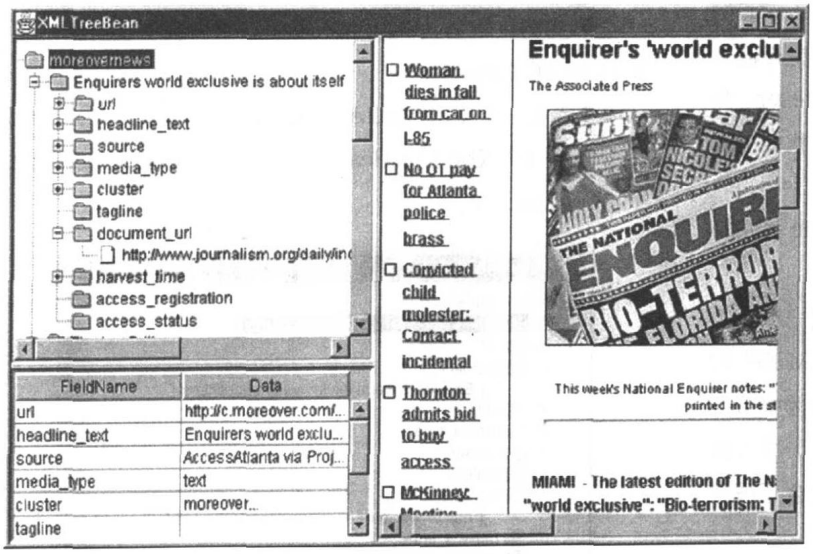


图5 浏览器视图

用XMLTreeBean作为一个XML编辑器

使用JTree的另一个小花样是使它们可以编辑。正如我前面提到的，用MutableTreeNodes作为XMLTreeBean的基类的主要原因是支持对树的拖放编辑。这样，添加几行代码，我们就能将改动传播到下层的XML。

首先得澄清，这不是真正的拖放。java.awt.dnd包实现了比我们所需要的更多功能，不过也需要更多的开销。我们准备做的是用一个MouseMotionListener捕获图标的拖动事件，改变光标使之看起来像个图标，从而获得视觉上的拖动效果。然后，当鼠标被释放时，触发DropEvent事件，注册的DropListeners就会处理对树所做的改动。而真正意义上的拖放是为更复杂的操作而设计的。

运用我们简化的拖放技术的第一步是添加一个MouseMotionListener来识别拖动动作，再添加一个MouseListener来识别表示放下事件的mouseReleased事件。

一旦接收到mouseDragged事件，我们先寻找距离点击位置最近的路径，然后在这条路径上找到被点击的节点，从而识别正在被拖动的节点：

```
TreePath path =
      getClosestPathForLocation(x,y);

int nodeIndex =
      path.getPathCount();

DefaultMutableTreeNode dragNode =
      path.getPathComponent(
      nodeIndex);
```

现在为了达到一个拖动的效果，我们根据节点图标生成一个定制的光标：

```
Toolkit toolkit =
       tree.getToolkit();
ImageIcon icon =
       renderer.getNodeIcon((
       Node)dragNode.getUserObject());

Cursor cursor =
       toolkit.createCustomCursor(
       icon.getImage(), new Point(4,4),
       "dragCursor");

tree.setCursor(cursor);
```

最后，我们移走正在被拖动的节点：

```
DefaultTreeModel model = (
       DefaultTreeModel)tree.getModel();
model.removeNodeFromParent(
       dragNode);
```

这样看起来就像是我们把节点拖离了树。拖动操作就这么多了。现在在新的位置放下节点并把它加到树上。同样，先寻找离放下位置最近的路径，然后在这条路径上找到节点：

```
TreePath path =
      getClosestPathForLocation(x,y);
int nodeIndex =
      path.getPathCount();
DefaultMutableTreeNode dropNode =
      path.getPathComponent(
      nodeIndex);
```

现在我们获得TreeModel，然后用它在放下位置插入被拖动的节点：

```
int insertIndex =
      dropNode.getChildCount();
DefaultTreeModel model = (
      DefaultTreeModel)tree.getModel();
model.insertNodeInto(
      dragNode,dropNode,insertIndex);
```

在对JTree做改动时使用TreeModel很重要，因为TreeModel会触发正确的事件以确保所有东西都得到更新。如果你用其他任何方式将节点添加到JTree中，JTree将不会正确显示。

为了确保我们把拖动的节点放置在JTree合适的位置上，我们在mouseDragged()里添加几行代码，对鼠标正在掠过的节点添加下划线：

```
int row =
      tree.getRowForLocation(x,y);
if(row>=0){
      int rowHeight =
            tree.getRowHeight();
      Graphics g = tree.getGraphics();
      Rectangle rowBounds =
            tree.getRowBounds(row);

      Dimension d = tree.getSize();
      g.setColor(Color.red);
      g.drawRect(rowBounds.x,
            rowBounds.y+rowBounds.height-1,
            rowBounds.width, 1);
}
```

这样我们就不会把它们放在错误的位置以至于丢失节点(当我在Windows里移动文件时，总是发生这样的事)。

最后，恢复原先的鼠标：

```
tree.setCursor(
      Cursor.getDefaultCursor());
tree.repaint();
```

现在，添加代码处理XML编辑。因为我们知道树节点参与了拖动操作，在mouseDragged()方法里我们所要做的就是获得它们的用户对象，然后调用父节点的removeChild()方法将被拖动的节点从XML树上移走：

```
Node xmlParent =
      (Node)parent.getUserObject();

Node xmlChild =
      (Node)dragNode.getUserObject();

xmlParent.removeChild(xmlChild);
```

现在为了把XML节点添加到它的新父节点下，我们将在mouseReleased()方法实现这个过程：

```
Node xmlParent =
      (Node)dropNode.getUserObject();
Node xmlChild =
      (Node)dragNode.getUserObject();
xmlParent.appendChild(xmlChild);
```

要想看看做完这些会发生什么，需添加一些代码，使我们改变树时先触发一个DOMEvent事件：

```
public void treeNodesChanged(
      TreeModelEvent e){
      DOMEvent evt = new
            DOMEvent(treeBean,
            treeBean.XmlDoc);
      treeBean.fireDOMEvent(evt);
}
```

总的效果就是创建一个很基本的拖放编辑器，可以将节点从JTree拖动，并在我们放下时附在路径的最末端(参看XMLEditableTreeBean和TreeEditListener的程序清单)。

现在让我们生成一个XMLTextAreaBean对象(见程序清单8)来查看结果：我们对JTree的编辑同时反映在XML文档上。因为XML不一定有格式，故我们添加了简单的打印功能，从而获得有良好格式的显示。你也可以使用Xerces的XMLSerializer和OutputFormat类来处理格式化。然而，要

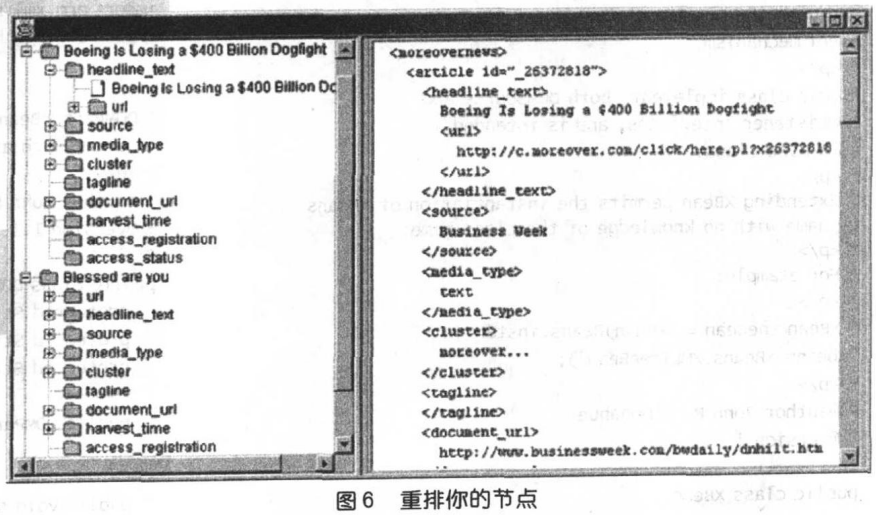


图6　重排你的节点

做到这一点，需要从OutputStream派生新类来对JTextArea进行写操作，还得用OutputFormat类做实验以获得期望的结果。

XMLTextAreaBean继承了XMLGUIBean，递归分解XML文档，添加尖括号和适当的缩进对其进行格式化，并附在一个JTextArea上。在分解文档前先清空JTextArea，分解后将文档插入到JTextArea最顶部，以达到清楚的效果（见图6）。

这篇文章仅仅涉及了使用可视化XBeans能很容易完成的工作。我曾编过一个我觉得很有用的应用程序。那是一个基于XBeans的可视化IDE。在这个开发环境中生成可视化XBeans应用程序只需要从XMLToolBarBean拖动XBeans并放在XMLTreeBean上，就可以生成XML定义的基于XBeans的可视化GUI组件。我希望你会觉得可视化XBeans是很有用的。

关于作者：

John B. O'Donahue是一位有20多年经验的软件开发人员，曾用过C，C++和Java。他现在是J-Machines公司（一家致力于开发多层文档管理系统和基于XML的图形用户接口开发的咨询公司）的首席技术官。可以发邮件到john@j-machines.com与作者联系。

程序清单1：Xbeans基类隐藏了DOMSource和DOMListener接口。

```
package xbeans.XBeans;

import xbeans.*;
import java.util.Vector;
import org.w3c.dom.Document;
/**
* XBeans are java beans which process xml documents,
and pass the processed documents via the
event mechanism.
* <p/>
* This class implements both DOMSource and
DOMListener interfaces, and is intended
as a base class.
* <p/>
* Extending XBean permits the instantiation of XBeans
by name with no knowledge of the class name:
* <p/>
* For example:
* <p/>
* XBean theBean = (XBean)Beans.instantiate(
"xbeans.XBeans.XMLTreeBean");
* <p/>
* @author John B. O'Donahue
* @version 1.1.
*/
public class XBean
    implements xbeans.DOMListener,xbeans.DOMSource{
    protected Document processedXmlDoc = null;
    protected DOMListener listener = null;

    public XBean(){
     }

    public void setDOMListener(
       DOMListener listener) {
       this.listener=listener;
     }

    public void documentReady(DOMEvent evt)
       throws XbeansException {
       processedXmlDoc = processDocument(
          evt.getDocument());
       DOMEvent domEvt = new DOMEvent
          this,processedXmlDoc);
       fireDOMEvent(domEvt);
     }

    public void fireDOMEvent(DOMEvent domEvt)
       throws XbeansException {
       if(listener!=null){
          listener.documentReady(domEvt);
        }
     }

    public Document processDocument(
       Document doc)throws XbeansException {
       return doc;
     }
}
```

程序清单2：DOMParserBean是一个分析字符串，文件，或者URL中XML的通用source bean。

```
package xbeans.XBeans;

import java.io.*;
import java.net.*;

import xbeans.*;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.xml.sax.InputSource;
import org.apache.xerces.parsers.DOMParser;

/**
* DOMParserBean extends XBean to parse xml rom
 sources such as Strings, files and URLs.
* <p/>
* @author John B. O'Donahue
* @version 1.1.
*/
public class DOMParserBean extends XBean{
   protected String UrlString = null;
   protected String XmlString = null;
   protected String XmlFileName = null;

   public DOMParserBean(){
    }

   public void setXmlString(String XmlString){
```

```
        this.XmlString = XmlString;
    }
    public void setUrlString(String UrlString){
        this.UrlString = UrlString;
        String buffer = "";
        try {
            URL fileURL = new URL(UrlString);
            InputStream inputStream =
                fileURL.openStream();

            int byteCount;
            byte[] xml = new byte[4096];
            while((byteCount =
                inputStream.read(xml))&gt;0){
                buffer += new String(xml,0,byteCount);
            }
            XmlString = buffer;
        }catch( Exception e ){
            e.printStackTrace();
        }
    }
    public void setXmlFileName(String XmlFileName){
        this.XmlFileName = XmlFileName;
        if(XmlFileName!=null){
            File f = new File (XmlFileName);
            byte[] xml = new byte[(int)f.length()];
            try {
                FileInputStream is =
                    new FileInputStream(f);
                is.read(xml,0,(int)f.length());
            } catch (Exception e) {
                System.out.println(e);
                e.printStackTrace();
            }
            XmlString = new String(xml);
        }
    }
    public void processDocument(){
        try{
            byte[] xml = getXml();
            if(xml.length > 0){
                ByteArrayInputStream x =
                    new ByteArrayInputStream(xml);
                DOMParser p = new DOMParser();
                InputSource s = new InputSource(x);
                p.parse(s);
                Document doc = p.getDocument();
                DOMEvent e = new DOMEvent(this,doc);
                fireDOMEvent(e);
            }
        }
        catch (Exception e){
            e.printStackTrace();
        }
    }
    private byte[] getXml() {
        byte[] xml = null;
        if(XmlString!=null) xml = XmlString.getBytes();
        return xml;
    }
}
```

程序清单3：XMLGUIBean是大多数XMLGUI组件的基类。

```
package xbeans.GUIBeans;

import java.awt.*;
import java.awt.dnd.*;
import java.awt.datatransfer.*;
import java.awt.event.*;
import java.io.*;
import java.util.*;
import javax.swing.*;
import javax.swing.border.*;
import javax.swing.tree.*;
import java.beans.Beans;
import xbeans.*;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;

import xbeans.XBeans.*;

/**
* XMLGUIBean extends JPanel and implements DOMSource
and DOMListener to provide a base class for GUIBeans.
*
* @author John B. O'Donahue
* @version 1.1.
*/
public class XMLGUIBean extends JPanel implements
DOMSource,DOMListener{
    protected Document processedXmlDoc = null;
    protected DOMListener listener = null;

    public XMLGUIBean(){
    }

    public void setDOMListener(
        DOMListener listener) {
        this.listener=listener;
    }

    public void documentReady(DOMEvent evt)
        throws XbeansException {
        processedXmlDoc =
            processDocument(evt.getDocument());
        DOMEvent domEvt =
            new DOMEvent(this,processedXmlDoc);
        fireDOMEvent(domEvt);
    }

    public void fireDOMEvent(DOMEvent domEvt)
        throws XbeansException {
        if(listener!=null){
            listener.documentReady(domEvt);
        }
    }

    public Document processDocument(Document doc)
        throws XbeansException {
        return doc;
    }
}
```

程序清单4：XMLTreeBean是表示XML文档的一条有效途径。

```
packagepackage xbeans.GUIBeans;
import java.awt.*;
```

```
import java.awt.event.*;
import java.io.*;
import java.util.*;
import javax.swing.*;
import javax.swing.border.*;
import javax.swing.event.*;
import javax.swing.tree.*;

import java.beans.Beans;
import xbeans.*;
import org.w3c.dom.Document;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.apache.xerces.parsers.DOMParser;

/**
* MutableXMLTreeBean displays xml in a JTree.
* <p/>
* @author John B. O'Donahue
* @version 1.1.
*/
public class XMLTreeBean extends XMLGUIBean{
  protected DOMListener listener = null;
  //protected Document processedXmlDoc = null;

  protected JPanel JTreePanel = new JPanel();
  protected JScrollPane JTreeScroller =
     new JScrollPane();
  protected Dimension size =
     new Dimension(426,266);

  protected int mouseOverRow = -1;

  protected JTree XMLTree =
     new javax.swing.JTree();

  public XMLTreeBean(){
     setLayout(new BorderLayout(0,0));
     JTreePanel.setLayout(new BorderLayout(0,0));
     add(BorderLayout.CENTER,JTreePanel);
     JTreeScroller.setOpaque(true);
     JTreePanel.add(BorderLayout.CENTER,
        JTreeScroller);
     XMLTree.setBorder(new EmptyBorder(5,5,5,5));
     XMLTree.setCellRenderer(
        new XMLTreeCellRenderer());
  }

  public Document processDocument(
     Document doc)throws XbeansException{
     DefaultTreeModel treeModel =
        createXMLTreeModel(doc);
     XMLTree.setModel(treeModel);

     XMLTree.addTreeWillExpandListener(
        new TreeExpansionListener());
     JTreeScroller.getViewport().add(XMLTree);
     XMLTree.setRootVisible(true);

     revalidate();
     return doc;
  }

  public void setMouseClickListener(
     MouseListener mouseListener){
     XMLTree.addMouseListener(mouseListener);
  }
     class TreeExpansionListener
     implements TreeWillExpandListener{
     public void treeWillCollapse(
        TreeExpansionEvent event)
        throws ExpandVetoException{
     }
     public void treeWillExpand(
        TreeExpansionEvent event)
        throws ExpandVetoException{
           revalidate();
     }
  }
  /**
  * Create a TreeModel using DefaultMutableTreeNodes
  */
  protected DefaultTreeModel createXMLTreeModel(
     Document doc)
     throws XbeansException{
     Node root = doc.getDocumentElement();
     DefaultMutableTreeNode treeRoot =
        createTreeNode(root);
     return new DefaultTreeModel(treeRoot);
  }
  /**
  * Recursively parse an xml document to create a
  * JTree of DefaultMutableTreeNodes
  */
  protected DefaultMutableTreeNode
     createTreeNode(Node node){
     DefaultMutableTreeNode treeNode =
        new DefaultMutableTreeNode(node);

     NodeList children = node.getChildNodes();
     for(int i=0;i<children.getLength();i++){
        Node child = (Node)children.item(i);

        // if node is an empty text node, do not add
        // to tree
        if(!(child.getNodeType()==Node.TEXT_NODE &&
           child.getNodeValue().trim().length()==0)){
           DefaultMutableTreeNode childNode =
              createTreeNode(child);
           treeNode.add(childNode);
        }
     }
     return treeNode;
  }
}
```

程序清单5：XMLTreeCellRenderer定制JTree.的外观。

```
package xbeans.GUIBeans;

import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.util.*;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.border.*;
import javax.swing.tree.*;
```