|  |
| --- |
| DOM4J是dom4j.org出品的一个开源XML解析包，它的网站中这样定义：      **Dom4j是一个易用的、开源的库，用于XML，XPath和XSLT。它应用于Java平台，采用了Java集合框架并完全支持DOM，SAX和JAXP。**       DOM4J使用起来非常简单。只要你了解基本的 XML-DOM模型，就能使用。然而他自己带的指南只有短短一页（html），不过说的到挺全。国内的中文资料很少。因而俺写这个短小的教程方便大家使 用，这篇文章仅谈及基本的用法，如需深入的使用，请……自己摸索或查找别的资料。       之前看过IBM developer社区的文章（参见附录），提到一些XML解析包的性能比较，其中DOM4J的性能非常出色，在多项测试中名列前茅。（事实上DOM4J的官方文档中也引用了这个比较）所以这次的项目中我采用了DOM4J作为XML解析工具。        在国内比较流行的是使用JDOM作为解析器，两者各擅其长，但DOM4J最大的特色是使用大量的接口，这也是它被认为比JDOM灵活的主要原因。大师不是 说过么，“面向接口编程”。目前使用DOM4J的已经越来越多。如果你善于使用JDOM，不妨继续用下去，只看看本篇文章作为了解与比较，如果你正要采用 一种解析器，不如就用DOM4J吧。 |

它的主要接口都在org.dom4j这个包里定义：

|  |  |
| --- | --- |
| ***Attribute*** | Attribute定义了XML的属性 |
| ***Branch*** | Branch为能够包含子节点的节点如XML元素(Element)和文档(Docuemnts)定义了一个公共的行为， |
| ***CDATA*** | CDATA 定义了XML CDATA 区域 |
| [***Charact***](http://www.daima.com.cn/Info/55/Local%20Settings/Temporary%20Internet%20Files/MyWeb/myclasses/dom4j-1.4/doc/apidocs/org/dom4j/CharacterData.html)***e***[***rData***](http://www.daima.com.cn/Info/55/Local%20Settings/Temporary%20Internet%20Files/MyWeb/myclasses/dom4j-1.4/doc/apidocs/org/dom4j/CharacterData.html) | CharacterData是一个标识接口，标识基于字符的节点。如CDATA，Comment, Text. |
| ***Comment*** | Comment 定义了XML注释的行为 |
| ***Document*** | 定义了XML文档 |
| ***DocumentType*** |  |

|  |
| --- |
| DocumentType 定义XML DOCTYPE声明 |
| ***Element*** | Element定义XML 元素 |
| ***ElementHandler*** | ElementHandler定义了 Element 对象的处理器 |
| ***ElementPath*** | 被 ElementHandler 使用，用于取得当前正在处理的路径层次信息 |
| ***Entity*** | Entity定义 XML entity |
| ***Node*** | Node为所有的dom4j中XML节点定义了多态行为 |
| [***Node***](http://www.daima.com.cn/Info/55/Local%20Settings/Temporary%20Internet%20Files/MyWeb/myclasses/dom4j-1.4/doc/apidocs/org/dom4j/NodeFilter.html)***F***[***ilter***](http://www.daima.com.cn/Info/55/Local%20Settings/Temporary%20Internet%20Files/MyWeb/myclasses/dom4j-1.4/doc/apidocs/org/dom4j/NodeFilter.html) | NodeFilter 定义了在dom4j节点中产生的一个滤镜或谓词的行为（predicate） |
| ***ProcessingInstruction*** | ProcessingInstruction 定义 XML 处理指令. |
| ***Text*** | Text 定义XML 文本节点. |
| ***Visitor*** | Visitor 用于实现Visitor模式. |
| ***XPath*** | XPath 在分析一个字符串后会提供一个XPath 表达式 |

看名字大致就知道它们的涵义如何了。

要想弄懂这套接口，关键的是要明白接口的继承关系：

* interface java.lang.**Cloneable**
  + interface org.dom4j.**Node**
    - interface org.dom4j.**Attribute**
    - interface org.dom4j.**Branch**
      * interface org.dom4j.**Document**
      * interface org.dom4j.**Element**
    - interface org.dom4j.**CharacterData**
      * interface org.dom4j.**CDATA**
      * interface org.dom4j.**Comment**
      * interface org.dom4j.**Text**
    - interface org.dom4j.**DocumentType**
    - interface org.dom4j.**Entity**
    - interface org.dom4j.**ProcessingInstruction**

一目了然，很多事情都清楚了。大部分都是由Node继承来的。知道这些关系，将来写程序就不会出现ClassCastException了。

下面给出一些例子（部分摘自DOM4J自带的文档），简单说一下如何使用。

１． 读取并解析XML文档：

读写XML文档主要依赖于org.dom4j.io包，其中提供DOMReader和SAXReader两类不同方式，而调用方式是一样的。这就是依靠接口的好处。

|  |
| --- |
| // 从文件读取XML，输入文件名，返回XML文档  **public** Document read(String fileName) **throws** MalformedURLException, DocumentException {         SAXReader reader = **new** SAXReader();         Document document = reader.read(**new** File(fileName));  **return** document;      } |

    其中，reader的read方法是重载的，可以从InputStream, File, Url等多种不同的源来读取。得到的Document对象就代表了整个XML。根据本人自己的经验，读取的字符编码是按照XML文件头定义的编码来转换。 如果遇到乱码问题，注意要把各处的编码名称保持一致即可。

２．取得Root节点

读取后的第二步，就是得到Root节点。熟悉XML的人都知道，一切XML分析都是从Root元素开始的。

|  |
| --- |
| **public** Element getRootElement(Document doc){  **return** doc.getRootElement();      } |

３．遍历XML树

DOM4J提供至少3种遍历节点的方法：

1) 枚举(Iterator)

|  |
| --- |
| // 枚举所有子节点  **for** ( Iterator i = root.elementIterator(); i.hasNext(); ) {         Element element = (Element) i.next();         // do something      }      // 枚举名称为foo的节点  **for** ( Iterator i = root.elementIterator(foo); i.hasNext();) {         Element foo = (Element) i.next();         // do something      }      // 枚举属性  **for** ( Iterator i = root.attributeIterator(); i.hasNext(); ) {         Attribute attribute = (Attribute) i.next();         // do something      } |

2)递归

递归也可以采用Iterator作为枚举手段，但文档中提供了另外的做法

|  |
| --- |
| **public** **void** treeWalk() {         treeWalk(getRootElement());      }  **public** **void** treeWalk(Element element) {  **for** (**int** i = 0, size = element.nodeCount(); i < size; i++)     {             Node node = element.node(i);  **if** (node **instanceof** Element) {                treeWalk((Element) node);             } **else** { // do something....             }         }  } |

3) Visitor模式

最令人兴奋的是DOM4J对Visitor的支持，这样可以大大缩减代码量， 并且清楚易懂。了解设计模式的人都知道，Visitor是GOF设计模式之一。其主要原理就是两种类互相保有对方的引用，并且一种作为Visitor去访 问许多Visitable。我们来看DOM4J中的Visitor模式(快速文档中没有提供)

只需要自定一个类实现Visitor接口即可。

|  |
| --- |
| **public** **class** MyVisitor **extends** VisitorSupport {  **public** **void** visit(Element element){                 System.out.println(element.getName());             }  **public** **void** visit(Attribute attr){                 System.out.println(attr.getName());             }          }            调用：  root.accept(new MyVisitor()) |

    Visitor接口提供多种Visit()的重载，根据XML不同的对象，将采用不同的方式来访问。上面是给出的Element和Attribute的简 单实现，一般比较常用的就是这两个。VisitorSupport是DOM4J提供的默认适配器，Visitor接口的Default Adapter模式，这个模式给出了各种visit(\*)的空实现，以便简化代码。

    注意，这个Visitor是自动遍历所有子节点的。如果是root.accept(MyVisitor)，将遍历子节点。我第一次用的时候，认为是需要自己遍历，便在递归中调用Visitor，结果可想而知。

4. XPath支持

    DOM4J对XPath有良好的支持，如访问一个节点，可直接用XPath选择。

|  |
| --- |
| **public** **void** bar(Document document) {    List list = document.selectNodes( //foo/bar );  Node node = document.selectSingleNode(//foo/bar/author);    String name = node.valueOf( @name );    } |
|  |

    例如，如果你想查找XHTML文档中所有的超链接，下面的代码可以实现：

|  |
| --- |
| **public** **void** findLinks(Document document) **throws** DocumentException {          List list = document.selectNodes( //a/@href );  **for** (Iterator iter = list.iterator(); iter.hasNext(); ) {              Attribute attribute = (Attribute) iter.next();              String url = attribute.getValue();          }       } |

5. 字符串与XML的转换

有时候经常要用到字符串转换为XML或反之，

|  |
| --- |
| // XML转字符串  　 Document document = ...;    String text = document.asXML();  // 字符串转XML   String text = <person> <name>James</name> </person>;   Document document = DocumentHelper.parseText(text); |

6 用XSLT转换XML

|  |
| --- |
| **public** Document styleDocument(         Document document,         String stylesheet      ) **throws** Exception {      // load the transformer using JAXP      TransformerFactory factory = TransformerFactory.newInstance();      Transformer transformer = factory.newTransformer(  **new** StreamSource( stylesheet )      );      // now lets style the given document      DocumentSource source = **new** DocumentSource( document );      DocumentResult result = **new** DocumentResult();      transformer.transform( source, result );      // return the transformed document      Document transformedDoc = result.getDocument();  **return** transformedDoc;  } |

7. 创建XML

  一般创建XML是写文件前的工作，这就像StringBuffer一样容易。

|  |
| --- |
| **public** Document createDocument() {         Document document = DocumentHelper.createDocument();         Element root = document.addElement(root);         Element author1 =             root                .addElement(author)                .addAttribute(name, James)                .addAttribute(location, UK)                .addText(James Strachan);         Element author2 =             root                .addElement(author)                .addAttribute(name, Bob)                .addAttribute(location, US)                .addText(Bob McWhirter);  **return** document;      } |

8. 文件输出

    一个简单的输出方法是将一个Document或任何的Node通过write方法输出

|  |
| --- |
| FileWriter out = **new** FileWriter( foo.xml );      document.write(out); |

  如果你想改变输出的格式，比如美化输出或缩减格式，可以用XMLWriter类

|  |
| --- |
| **public** **void** write(Document document) **throws** IOException {         // 指定文件         XMLWriter writer = **new** XMLWriter(**new** FileWriter( output.xml ));         writer.write( document );         writer.close();         // 美化格式         OutputFormat format = OutputFormat.createPrettyPrint();         writer = **new** XMLWriter( System.out, format );         writer.write( document );         // 缩减格式         format = OutputFormat.createCompactFormat();         writer = **new** XMLWriter( System.out, format );         writer.write( document );      } |

如何，DOM4J够简单吧，当然，还有一些复杂的应用没有提到，如ElementHandler等。如果你动心了，那就一起来用DOM4J.

DOM4J官方网站：(我老连不上)

<http://www.dom4j.org/>