读取XML文件的四种方式

众所周知，现在解析XML的方法越来越多，但主流的方法也就四种，即：DOM、SAX、JDOM和DOM4J

下面首先给出这四种方法的jar包下载地址

DOM：在现在的Java JDK里都**自带**了，在xml-apis.jar包里

SAX：[http://sourceforge.net/projects/sax/](http://sourceforge.net/projects/sax/" \t "_blank)

JDOM：[http://jdom.org/downloads/index.html](http://jdom.org/downloads/index.html" \t "_blank)

DOM4J：[http://sourceforge.net/projects/dom4j/](http://sourceforge.net/projects/dom4j/" \t "_blank)

**一、介绍及优缺点分析**

**1. DOM（Document Object Model)**

      DOM是用与平台和语言无关的方式表示XML文档的官方W3C标准。DOM是以层次结构组织的节点或信息片断的集合。这个层次结构允许开发人员在树中寻找特定信息。分析该结构通常需要加载整个文档和构造层次结构，然后才能做任何工作。由于它是基于信息层次的，因而DOM被认为是基于树或基于对象的。

【优点】  
      ①允许应用程序对数据和结构做出更改。  
      ②访问是双向的，可以在任何时候在树中上下导航，获取和操作任意部分的数据。  
【缺点】  
      ①通常需要加载整个XML文档来构造层次结构，消耗资源大。

**2. SAX（Simple API for XML)**

     SAX处理的优点非常类似于流媒体的优点。分析能够立即开始，而不是等待所有的数据被处理。而且，由于应用程序只是在读取数据时检查数据，因此不需要将数据[存储](http://www.storworld.com/)在内存中。这对于大型文档来说是个巨大的优点。事实上，应用程序甚至不必解析整个文档；它可以在某个条件得到满足时停止解析。一般来说，SAX还比它的替代者DOM快许多。

     选择DOM还是选择SAX？ 对于需要自己编写代码来处理XML文档的开发人员来说， 选择DOM还是SAX解析模型是一个非常重要的设计决策。 DOM采用建立树形结构的方式访问XML文档，而SAX采用的是事件模型。

     DOM解析器把XML文档转化为一个包含其内容的树，并可以对树进行遍历。用DOM解析模型的优点是编程容易，开发人员只需要调用建树的指令，然后利用navigation APIs访问所需的树节点来完成任务。可以很容易的添加和修改树中的元素。然而由于使用DOM解析器的时候需要处理整个XML文档，所以对性能和内存的要求比较高，尤其是遇到很大的XML文件的时候。由于它的遍历能力，DOM解析器常用于XML文档需要频繁的改变的服务中。

     SAX解析器采用了基于事件的模型，它在解析XML文档的时候可以触发一系列的事件，当发现给定的tag的时候，它可以激活一个回调方法，告诉该方法制定的标签已经找到。SAX对内存的要求通常会比较低，因为它让开发人员自己来决定所要处理的tag.特别是当开发人员只需要处理文档中所包含的部分数据时，SAX这种扩展能力得到了更好的体现。但用SAX解析器的时候编码工作会比较困难，而且很难同时访问同一个文档中的多处不同数据。

【优势】  
     ①不需要等待所有数据都被处理，分析就能立即开始。  
     ②只在读取数据时检查数据，不需要保存在内存中。  
     ③可以在某个条件得到满足时停止解析，不必解析整个文档。  
     ④效率和性能较高，能解析大于系统内存的文档。

【缺点】  
     ①需要应用程序自己负责TAG的处理逻辑（例如维护父/子关系等），文档越复杂程序就越复杂。  
     ②单向导航，无法定位文档层次，很难同时访问同一文档的不同部分数据，不支持XPath。

**3. JDOM(Java-based Document Object Model)**

     JDOM的目的是成为Java特定文档模型，它简化与XML的交互并且比使用DOM实现更快。由于是第一个Java特定模型，JDOM一直得到大力推广和促进。正在考虑通过“Java规范请求JSR-102”将它最终用作“Java标准扩展”。从2000年初就已经开始了JDOM开发。

     JDOM与DOM主要有两方面不同。首先，JDOM仅使用具体类而不使用接口。这在某些方面简化了API，但是也限制了灵活性。第二，API大量使用了Collections类，简化了那些已经熟悉这些类的Java开发者的使用。

     JDOM文档声明其目的是“使用20%（或更少）的精力解决80%（或更多）Java/XML问题”（根据学习曲线假定为20%）。JDOM对于大多数Java/XML应用程序来说当然是有用的，并且大多数开发者发现API比DOM容易理解得多。JDOM还包括对程序行为的相当广泛检查以防止用户做任何在XML中无意义的事。然而，它仍需要您充分理解XML以便做一些超出基本的工作（或者甚至理解某些情况下的错误）。这也许是比学习DOM或JDOM接口都更有意义的工作。

     JDOM自身不包含解析器。它通常使用SAX2解析器来解析和验证输入XML文档（尽管它还可以将以前构造的DOM表示作为输入）。它包含一些转换器以将JDOM表示输出成SAX2事件流、DOM模型或XML文本文档。JDOM是在Apache许可证变体下发布的开放源码。

【优点】  
     ①使用具体类而不是接口，简化了DOM的API。  
     ②大量使用了Java集合类，方便了Java开发人员。

【缺点】  
     ①没有较好的灵活性。  
     ②性能较差。

**4. DOM4J(Document Object Model for Java)**  
      虽然DOM4J代表了完全独立的开发结果，但最初，它是JDOM的一种智能分支。它合并了许多超出基本XML文档表示的功能，包括集成的XPath支持、XML Schema支持以及用于大文档或流化文档的基于事件的处理。它还提供了构建文档表示的选项，它通过DOM4J API和标准DOM接口具有并行访问功能。从2000下半年开始，它就一直处于开发之中。

      为支持所有这些功能，DOM4J使用接口和抽象基本类方法。DOM4J大量使用了API中的Collections类，但是在许多情况下，它还提供一些替代方法以允许更好的性能或更直接的编码方法。直接好处是，虽然DOM4J付出了更复杂的API的代价，但是它提供了比JDOM大得多的灵活性。

      在添加灵活性、XPath集成和对大文档处理的目标时，DOM4J的目标与JDOM是一样的：针对Java开发者的易用性和直观操作。它还致力于成为比JDOM更完整的解决方案，实现在本质上处理所有Java/XML问题的目标。在完成该目标时，它比JDOM更少强调防止不正确的应用程序行为。

      DOM4J是一个非常非常优秀的Java XML API，具有性能优异、功能强大和极端易用使用的特点，同时它也是一个开放源代码的软件。如今你可以看到越来越多的Java软件都在使用DOM4J来读写XML，特别值得一提的是连Sun的JAXM也在用DOM4J.

【优点】  
     ①大量使用了Java集合类，方便Java开发人员，同时提供一些提高性能的替代方法。  
     ②支持XPath。  
     ③有很好的性能。

【缺点】  
     ①大量使用了接口，API较为复杂。

**二、比较**

**1. DOM4J性能最好**，连Sun的JAXM也在用DOM4J。目前许多开源项目中大量采用DOM4J，例如大名鼎鼎的Hibernate也用DOM4J来读取XML配置文件。如果不考虑可移植性，那就采用DOM4J.

**2. JDOM和DOM在性能测试时表现不佳，在测试10M文档时内存溢出，但可移植**。在小文档情况下还值得考虑使用DOM和JDOM.虽然JDOM的开发者已经说明他们期望在正式发行版前专注性能问题，但是从性能观点来看，它确实没有值得推荐之处。另外，DOM仍是一个非常好的选择。DOM实现广泛应用于多种编程语言。它还是许多其它与XML相关的标准的基础，因为它正式获得W3C推荐（与基于非标准的Java模型相对），所以在某些类型的项目中可能也需要它（如在JavaScript中使用DOM）。

**3. SAX表现较好，**这要依赖于它特定的解析方式－事件驱动。一个SAX检测即将到来的XML流，但并没有载入到内存（当然当XML流被读入时，会有部分文档暂时隐藏在内存中）。

**我的看法：如果XML文档较大且不考虑移植性问题建议采用DOM4J；如果XML文档较小则建议采用JDOM；如果需要及时处理而不需要保存数据则考虑SAX。但无论如何，还是那句话：适合自己的才是最好的，如果时间允许，建议大家讲这四种方法都尝试一遍然后选择一种适合自己的即可。**

**三、示例**

为了节约篇幅，这里暂时不给出这四种建立XML文档的方法与差异，仅给出解析XML文档的代码，如果需要完整工程（建立XML文档+解析XML+测试比较），可去我的CSDN下载。

这里以下面的XML内容为例进行解析：

[复制代码](javascript:void(0);)

<?xml version="1.0" encoding="UTF-8"?>

<users>

<user id="0">

<name>Alexia</name>

<age>23</age>

<sex>Female</sex>

</user>

<user id="1">

<name>Edward</name>

<age>24</age>

<sex>Male</sex>

</user>

<user id="2">

<name>wjm</name>

<age>23</age>

<sex>Female</sex>

</user>

<user id="3">

<name>wh</name>

<age>24</age>

<sex>Male</sex>

</user>

</users>

[复制代码](javascript:void(0);)

首先定义XML文档解析的接口：

[复制代码](javascript:void(0);)

1 /\*\*

2 \* @author Alexia

3 \*

4 \* 定义XML文档解析的接口

5 \*/

6 public interface XmlDocument {

7

8 /\*\*

9 \* 解析XML文档

10 \*

11 \* @param fileName

12 \* 文件全路径名称

13 \*/

14 public void parserXml(String fileName);

15 }

[复制代码](javascript:void(0);)

1. DOM示例

[复制代码](javascript:void(0);)

package com.xml;

import java.io.FileNotFoundException;

import java.io.FileOutputStream;

import java.io.IOException;

import java.io.PrintWriter;

import javax.xml.parsers.DocumentBuilder;

import javax.xml.parsers.DocumentBuilderFactory;

import javax.xml.parsers.ParserConfigurationException;

import javax.xml.transform.OutputKeys;

import javax.xml.transform.Transformer;

import javax.xml.transform.TransformerConfigurationException;

import javax.xml.transform.TransformerException;

import javax.xml.transform.TransformerFactory;

import javax.xml.transform.dom.DOMSource;

import javax.xml.transform.stream.StreamResult;

import org.w3c.dom.Document;

import org.w3c.dom.Element;

import org.w3c.dom.Node;

import org.w3c.dom.NodeList;

import org.xml.sax.SAXException;

/\*\*

\* @author Alexia

\*

\* DOM 解析XML文档

\*/

public class DomDemo implements XmlDocument {

private Document document;

public void parserXml(String fileName) {

try {

DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();

DocumentBuilder db = dbf.newDocumentBuilder();

Document document = db.parse(fileName);

NodeList users = document.getChildNodes();

for (int i = 0; i < users.getLength(); i++) {

Node user = users.item(i);

NodeList userInfo = user.getChildNodes();

for (int j = 0; j < userInfo.getLength(); j++) {

Node node = userInfo.item(j);

NodeList userMeta = node.getChildNodes();

for (int k = 0; k < userMeta.getLength(); k++) {

if(userMeta.item(k).getNodeName() != "#text")

System.out.println(userMeta.item(k).getNodeName()

+ ":" + userMeta.item(k).getTextContent());

}

System.out.println();

}

}

} catch (FileNotFoundException e) {

e.printStackTrace();

} catch (ParserConfigurationException e) {

e.printStackTrace();

} catch (SAXException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

}

}

}

[复制代码](javascript:void(0);)

2. SAX示例

[复制代码](javascript:void(0);)

package com.xml;

import java.io.FileInputStream;

import java.io.FileNotFoundException;

import java.io.FileOutputStream;

import java.io.IOException;

import java.io.InputStream;

import java.io.OutputStream;

import java.io.StringWriter;

import javax.xml.parsers.ParserConfigurationException;

import javax.xml.parsers.SAXParser;

import javax.xml.parsers.SAXParserFactory;

import javax.xml.transform.OutputKeys;

import javax.xml.transform.Result;

import javax.xml.transform.Transformer;

import javax.xml.transform.TransformerConfigurationException;

import javax.xml.transform.sax.SAXTransformerFactory;

import javax.xml.transform.sax.TransformerHandler;

import javax.xml.transform.stream.StreamResult;

import org.xml.sax.Attributes;

import org.xml.sax.SAXException;

import org.xml.sax.helpers.AttributesImpl;

import org.xml.sax.helpers.DefaultHandler;

/\*\*

\* @author Alexia

\*

\* SAX 解析XML文档

\*/

public class SaxDemo implements XmlDocument {

public void parserXml(String fileName) {

SAXParserFactory saxfac = SAXParserFactory.newInstance();

try {

SAXParser saxparser = saxfac.newSAXParser();

InputStream is = new FileInputStream(fileName);

saxparser.parse(is, new MySAXHandler());

} catch (ParserConfigurationException e) {

e.printStackTrace();

} catch (SAXException e) {

e.printStackTrace();

} catch (FileNotFoundException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

}

}

}

class MySAXHandler extends DefaultHandler {

boolean hasAttribute = false;

Attributes attributes = null;

public void startDocument() throws SAXException {

// System.out.println("文档开始打印了");

}

public void endDocument() throws SAXException {

// System.out.println("文档打印结束了");

}

public void startElement(String uri, String localName, String qName,

Attributes attributes) throws SAXException {

if (qName.equals("users")) {

return;

}

if (qName.equals("user")) {

return;

}

if (attributes.getLength() > 0) {

this.attributes = attributes;

this.hasAttribute = true;

}

}

public void endElement(String uri, String localName, String qName)

throws SAXException {

if (hasAttribute && (attributes != null)) {

for (int i = 0; i < attributes.getLength(); i++) {

System.out.print(attributes.getQName(0) + ":"

+ attributes.getValue(0));

}

}

}

public void characters(char[] ch, int start, int length)

throws SAXException {

System.out.print(new String(ch, start, length));

}

}

[复制代码](javascript:void(0);)

3. JDOM示例

[复制代码](javascript:void(0);)

package com.xml;

import java.io.FileNotFoundException;

import java.io.FileOutputStream;

import java.io.IOException;

import java.util.List;

import org.jdom2.Document;

import org.jdom2.Element;

import org.jdom2.JDOMException;

import org.jdom2.input.SAXBuilder;

import org.jdom2.output.XMLOutputter;

/\*\*

\* @author Alexia

\*

\* JDOM 解析XML文档

\*

\*/

public class JDomDemo implements XmlDocument {

public void parserXml(String fileName) {

SAXBuilder builder = new SAXBuilder();

try {

Document document = builder.build(fileName);

Element users = document.getRootElement();

List userList = users.getChildren("user");

for (int i = 0; i < userList.size(); i++) {

Element user = (Element) userList.get(i);

List userInfo = user.getChildren();

for (int j = 0; j < userInfo.size(); j++) {

System.out.println(((Element) userInfo.get(j)).getName()

+ ":" + ((Element) userInfo.get(j)).getValue());

}

System.out.println();

}

} catch (JDOMException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

}

}

}

[复制代码](javascript:void(0);)

4. DOM4J示例

[复制代码](javascript:void(0);)

package com.xml;

import java.io.File;

import java.io.FileWriter;

import java.io.IOException;

import java.io.Writer;

import java.util.Iterator;

import org.dom4j.Document;

import org.dom4j.DocumentException;

import org.dom4j.DocumentHelper;

import org.dom4j.Element;

import org.dom4j.io.SAXReader;

import org.dom4j.io.XMLWriter;

/\*\*

\* @author Alexia

\*

\* Dom4j 解析XML文档

\*/

public class Dom4jDemo implements XmlDocument {

public void parserXml(String fileName) {

File inputXml = new File(fileName);

SAXReader saxReader = new SAXReader();

try {

Document document = saxReader.read(inputXml);

Element users = document.getRootElement();

for (Iterator i = users.elementIterator(); i.hasNext();) {

Element user = (Element) i.next();

for (Iterator j = user.elementIterator(); j.hasNext();) {

Element node = (Element) j.next();

System.out.println(node.getName() + ":" + node.getText());

}

System.out.println();

}

} catch (DocumentException e) {

System.out.println(e.getMessage());

}

}

}