

# 解析xml的4种方法详解 - jzhf2012的专栏 - 博客频道 - CSDN

## 1. 介绍

### 1) DOM(JAXP Crimson解析器)

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。DOM以及广义的基于树的处理具有几个优点。首先,由于树在内存中是持久的,因此可以修改它以便应用程序能对数据和结构作出更改。它还可以在任何时候在树中上下导航,而不是像SAX那样是一次性的处理。DOM使用起来也要简单得多。

### 2) SAX

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。

### 3) JDOM <http://www.jdom.org/>

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代码。

### 4) DOM4J <http://dom4j.sourceforge.NET/>

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。

## 2. 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。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)。

## 3. 四种xml操作方式的基本使用方法

[java]view plaincopy

1. xml文件:

2. "1.0" encoding="utf-8" ?>

3.

4.

5. "2005">A1

6. GZ

7.

8.

9. "2004">A2

10. XG

11.

12.

### 13. 1) DOM

14. import java.io.\*;

15. import java.util.\*;

16. import org.w3c.dom.\*;

17. import javax.xml.parsers.\*;

18. public class MyXMLReader{

19. public static void main(String args[]){

20. long lasting = System.currentTimeMillis();

21. try{

22. File f=new File("data\_10k.xml");

23. DocumentBuilderFactory factory=DocumentBuilderFactory.newInstance();

24. DocumentBuilder builder=factory.newDocumentBuilder();

25. Document doc = builder.parse(f);

26. NodeList nl = doc.getElementsByTagName("VALUE");

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

28. System.out.print("车牌号码:" +

doc.getElementsByTagName("NO").item(i).getFirstChild().getNodeValue());

29. System.out.println("车主地址:" +

doc.getElementsByTagName("ADDR").item(i).getFirstChild().getNodeValue());

30. }

31. }catch (Exception e){

32. e.printStackTrace();

33. }

### 34. 2) SAX

35. import org.xml.sax.\*;

36. import org.xml.sax.helpers.\*;

37. import javax.xml.parsers.\*;

38. public class MyXMLReader extends DefaultHandler {

39. java.util.Stack tags = new java.util.Stack();

40. public MyXMLReader() {

41. super();

42. }

```

43. publicstaticvoid main(String args[]) {
44.     long lasting = System.currentTimeMillis();
45.     try {
46.         SAXParserFactory sf = SAXParserFactory.newInstance();
47.         SAXParser sp = sf.newSAXParser();
48.         MyXMLReader reader = new MyXMLReader();
49.         sp.parse(new InputSource("data_10k.xml"), reader);
50.     } catch (Exception e) {
51.         e.printStackTrace();
52.     }
53.     System.out.println("运行时间: " + (System.currentTimeMillis() - lasting) + "毫秒");}
54. publicvoid characters(char ch[], int start, int length) throws SAXException {
55.     String tag = (String) tags.peek();
56.     if (tag.equals("NO")) {
57.         System.out.print("车牌号码: " + new String(ch, start, length));
58.     }
59.     if (tag.equals("ADDR")) {
60.         System.out.println("地址:" + new String(ch, start, length));
61.     }
62. }
63. publicvoid startElement(String uri,String localName,String qName,Attributes attrs) {
64.     tags.push(qName);}
65. }
66. 3) JDOM
67. import java.io.*;
68. import java.util.*;
69. import org.jdom.*;
70. import org.jdom.input.*;
71. publicclass MyXMLReader {
72.     publicstaticvoid main(String arge[]) {
73.         long lasting = System.currentTimeMillis();
74.         try {
75.             SAXBuilder builder = new SAXBuilder();
76.             Document doc = builder.build(new File("data_10k.xml"));
77.             Element foo = doc.getRootElement();
78.             List allChildren = foo.getChildren();
79.             for(int i=0;i<allChildren.size();i++) {
80.                 System.out.print("车牌号码:" + ((Element)allChildren.get(i)).getChild("NO").getText());
81.                 System.out.println("车主地址:" +
((Element)allChildren.get(i)).getChild("ADDR").getText());
82.             }
83.         } catch (Exception e) {
84.             e.printStackTrace();
85.         }
86.     }
87. 4) DOM4J
88. import java.io.*;
89. import java.util.*;
90. import org.dom4j.*;
91. import org.dom4j.io.*;
92. publicclass MyXMLReader {
93.     publicstaticvoid main(String arge[]) {
94.         long lasting = System.currentTimeMillis();
95.         try {
96.             File f = new File("data_10k.xml");
97.             SAXReader reader = new SAXReader();

```

```
98.     Document doc = reader.read(f);
99.     Element root = doc.getRootElement();
100.     Element foo;
101. for (Iterator i = root.elementIterator("VALUE"); i.hasNext();) {
102.     foo = (Element) i.next();
103.     System.out.print("车牌号码:" + foo.elementText("NO"));
104.     System.out.println("车主地址:" + foo.elementText("ADDR"));
105. }
106. } catch (Exception e) {
107.     e.printStackTrace();
108. }
109. }
```