XML解析

什么是XML：

* XML 于 1998 年 2 月 10 日成为 W3C 的推荐标准。
* XML 指可扩展标记语言（EXtensible Markup Language）。
* XML 是一种很像HTML的标记语言。
* XML 的设计宗旨是传输数据，而不是显示数据。
* XML 标签没有被预定义。您需要自行定义标签。
* XML 被设计为具有自我描述性。
* XML 是 W3C 的推荐标准。

XML语法规则：

<root>

<child>

<subchild>.....</subchild>

</child>

</root>

示例：

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

<!-- This is a comment -->

<note>

<to>Tove</to>

<from>Jani</from>

<heading>Reminder</heading>

<body>Don't forget me this weekend!</body>

</note>

命名规则：

* 名称可以包含字母、数字以及其他的字符
* 名称不能以数字或者标点符号开始
* 名称不能以字母 xml（或者 XML、Xml 等等）开始
* 名称不能包含空格

特点：

* 标签必须正确嵌套
* 标签必须关闭
* 区分大小写
* 节点中属性的值必须加引号

实体引用：

在 XML 中，一些字符拥有特殊的意义。

|  |  |  |
| --- | --- | --- |
| &lt; | < | 小于 |
| &gt: | > | 大于 |
| &amp: | & | 与 |
| &apos; | ' | 单引号 |
| &quot; | " | 引号 |

XML CDATA：

XML 文档中的所有文本均会被解析器解析。只有 CDATA 区段中的文本会被解析器忽略。

CDATA 部分由 "**<![CDATA[**" 开始，由 "**]]>**" 结束：

**<！[CDATA[** 2>3 </font> **]]>**

DOM 解析：

是将xml文件全部载入，组装成一颗dom树，然后通过节点以及节点之间的关系来解析xml文件或将xml以树结构的方式一次性加载进内存，易用，修改方便，但是更占内存。不适应于大文件，一般情况下用它即可应付大多数应用。 使用DOM创建XML:

/\*\*

\* DOM方式创建XML

\*

\* @throws ParserConfigurationException

\* @throws TransformerFactoryConfigurationError

\* @throws FileNotFoundException

\* @throws TransformerException

\*/

static void fun1() throws ParserConfigurationException, TransformerFactoryConfigurationError, FileNotFoundException,

TransformerException {

// 文档解析器工厂

DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();

// 文档解析器

DocumentBuilder builder = factory.newDocumentBuilder();

// xml文档对象

Document document = builder.newDocument();

// xml文件路径

String path = "src/com/kaige/xml/newdata.xml";

// 根节点

Element classinfo = document.createElement("classinfo");

// 创建5个学生节点

for (int i = 0; i < 10; i++) {

Element student = document.createElement("Student");

student.setAttribute("id", "stu" + i); // 添加属性

Element name = document.createElement("name");

name.setAttribute("nameid", "name" + i);// 添加属性

name.appendChild(document.createTextNode("姓名" + i));// 追加子节点

Element age = document.createElement("age");

age.appendChild(document.createTextNode(18 + i + ""));

// 将age和name加入到student

student.appendChild(name);

student.appendChild(age);

// 将student加入到root

classinfo.appendChild(student);

}

// 将classinfo加入到document

document.appendChild(classinfo);

// 将Document对象转换为DOMSource对象

DOMSource source = new DOMSource(document);

//变压器

Transformer tf = TransformerFactory.newInstance().newTransformer();

tf.setOutputProperty(OutputKeys.ENCODING, "UTF-8");// 设置编码格式

tf.setOutputProperty(OutputKeys.INDENT, "yes"); // 设置缩进

// 结果流

StreamResult sr=new StreamResult(filename);

//写入 数据

tf.transform(source, sr);

System.out.println("创建xml成功");

}

使用DOM读取XML:

/\*\*

\* DOM方式读取XML

\*

\* @throws ParserConfigurationException

\* @throws IOException

\* @throws SAXException

\*/

static void fun2() throws ParserConfigurationException, SAXException, IOException {

// 文档解析器工厂

DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();

// 文档解析器

DocumentBuilder builder = factory.newDocumentBuilder();

// 文档对象

Document document = builder.parse("src/com/kaige/xml/newdata.xml");

NodeList nodeList= document.getChildNodes();

//System.out.println("班级数量："+nodeList.getLength());

//根节点

Node classinfo=nodeList.item(0);

//获取学生节点

NodeList studentlist= classinfo.getChildNodes();

//遍历学生节点

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

Node node=studentlist.item(i);

if(node instanceof Text)continue;

System.out.println("学号："+node.getAttributes().item(0).getNodeValue());

NodeList fieldlist= node.getChildNodes();

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

Node fieldNode=fieldlist.item(j);

if(fieldNode instanceof Text)continue;

System.out.println(fieldNode.getNodeName()+":"+

fieldNode.getTextContent());

}

}

System.out.println("---------------------------");

}

}

SAX 解析：

SAX的工作方式简单地说就是对文档进行顺序扫描，当扫描到文档（document）开始与结束,元素（Element）开始与结束、文档（document）结束等地方时通知事件处理函数，由事件处理函数做相应的动作，然后继续同样的扫描，直至文档结束。

优点：消耗资源比较少；适合大文件解析；

缺点：只能读取不能修改；开发复杂；

sax.xml

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

<classinfo>

<student id="s001">

<name>张三</name>

<age>20</age>

</student>

<student id="s002">

<name>李四</name>

<age>25</age>

</student>

</classinfo>

模型类：

class Student{

private String id;

private String name;

private int age;

public String getId() {

return id;

}

public void setId(String id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public int getAge() {

return age;

}

public void setAge(int age) {

this.age = age;

}

@Override

public String toString() {

return "Student [id=" + id + ", name=" + name + ", age=" + age + "]";

}

}

handler:

class SAX2 extends DefaultHandler{

//定义字段

List<Student> list=null; //学生集合

Student student=null; //学生对象

String tag=null; //当前标签

public List<Student> getData(){

return list;

}

@Override

public void startDocument() throws SAXException {

// TODO Auto-generated method stub

//super.startDocument();

System.out.println("开始读取信息");

list=new ArrayList<>();

}

@Override

public void endDocument() throws SAXException {

// TODO Auto-generated method stub

//super.endDocument();

System.out.println("信息读取完毕");

}

@Override

public void startElement(String uri, String localName, String qName, Attributes attributes) throws SAXException {

// TODO Auto-generated method stub

//super.startElement(uri, localName, qName, attributes);

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

student=new Student();

student.setId(attributes.getValue(0));

}

tag=qName;

}

@Override

public void endElement(String uri, String localName, String qName) throws SAXException {

// TODO Auto-generated method stub

//super.endElement(uri, localName, qName);

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

list.add(student);

student=null;

}

tag=null;

}

@Override

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

// TODO Auto-generated method stub

//super.characters(ch, start, length);

if(tag!=null){

String content=new String(ch,start,length);

if("name".equals(tag)){

student.setName(content);

}else if("age".equals(tag)){

student.setAge(Integer.parseInt(content));

}

}

}

}

测试类：

public static void main(String[] args) throws Exception {

SAXParserFactory factory=SAXParserFactory.newInstance();

SAXParser parser=factory.newSAXParser();

SAX2 sax2=new SAX2();

parser.parse("src/com/kaige/xml/sax.xml", sax2);

for (Student stu:sax2.getData()) {

System.out.println(stu);

}

}

Dom 4j 解析：

dom4j-2.1.1.jar

DOM4J是 dom4j.org 出品的一个开源 XML 解析包。DOM4J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。

    Dom：把整个文档作为一个对象。

  DOM4J 最大的特色是使用大量的接口。它的主要接口都在org.dom4j里面定义：

|  |  |
| --- | --- |
| Attribute | 定义了 XML 的属性。 |
| Branch | 指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 | 定义了 XML CDATA 区域 |
| CharacterData | 是一个标识接口，标识基于字符的节点。如CDATA，Comment, Text. |
| Comment | 定义了 XML 注释的行为 |
| Document | 定义了XML 文档 |
| DocumentType | 定义 XML DOCTYPE 声明 |
| Element | 定义XML 元素 |
| 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 | 定义 XML entity |
| Node | 为dom4j中所有的XML节点定义了多态行为 |
| NodeFilter | 定义了在dom4j 节点中产生的一个滤镜或谓词的行为（predicate） |
| ProcessingInstruction | 定义 XML 处理指令 |
| Text | 定义 XML 文本节点 |
| Visitor | 用于实现 Visitor模式 |
| XPath | 在分析一个字符串后会提供一个 XPath 表达式 |

接口之间的继承关系：

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

常用方法：

访问根节点

Element root = document.getRootElement();

访问所有子节点

List<Element> list= root.elements();

访问指定名称的节点

Element name= student.element("name");

获取指定名称/索引的属性

Attribute sid= student.attribute("sid")

Attribute sid= student.attribute(0)

获取某节点的全部属性

List<Attribute> list = student.attributes();

删除某节点

Element student=root.elements().get(2);

root.remove(student);

删除指定名称的属性

Element student= root.elements().get(2);

student.remove(student.attribute("sid"));

修改节点名称和节点值

Element skill = root.elements().get(2);

student.setName("teacher");

student.setText("老陈");

修改属性值（属性名称无法修改）

Attribute attr = root.elements().get(2).attribute("sid");

attr.setText("stu118");

创建xml:

/\*\*

\* 创建xml

\* @throws Exception

\*/

static void fun1() throws Exception {

// 创建文档对象

Document document = DocumentHelper.createDocument();

// 创建根节点

Element root = document.addElement("classinfo");

root.addComment("班级信息");

for (int i = 0; i < 10; i++) {

Element student = root.addElement("student");

student.addAttribute("sid", "stu" + i);

Element name = student.addElement("name");

name.addAttribute("nameid", i + "");

name.setText("姓名" + i);

Element age = student.addElement("age");

age.setText(18 + i + "");

}

// 格式化输出器

OutputFormat format = OutputFormat.createPrettyPrint();

format.setEncoding("utf-8");

// 文件输出流

FileOutputStream fos = new FileOutputStream("src/com/kaige/xml/dom4j.xml");

// xml写入器

XMLWriter writer = new XMLWriter(fos, format);

// 写数据

writer.write(document);

// 关闭

writer.close();

System.out.println("创建成功");

}

读取xml:

/\*\*

\* 读取xml

\*

\* @throws DocumentException

\*/

static void fun2() throws DocumentException {

String filename = "src/com/kaige/xml/dom4j.xml";

// sax读取器

SAXReader saxReader = new SAXReader();

// 文档对象

Document document = saxReader.read(filename);

// 获取根元素

Element root = document.getRootElement();

// 获取子节点

List<Element> list = root.elements();

// System.out.println(list.size());

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

Element student = list.get(i);

Attribute sid = student.attribute("sid");

System.out.println(sid.getName() + ":" + sid.getValue());

System.out.println("姓名：" + student.element("name").getStringValue());

System.out.println("年龄：" + student.element("age").getStringValue());

System.out.println("------------------------------------------");

}

}

修改xml:

/\*\*

\* 修改xml

\*

\* @throws DocumentException

\* @throws IOException

\*/

static void fun3() throws DocumentException, IOException {

String filename = "src/com/kaige/xml/dom4j.xml";

// sax读取器

SAXReader saxReader = new SAXReader();

// 文档对象

Document document = saxReader.read(filename);

// 获取根元素

Element root = document.getRootElement();

// 修改第3个学生的姓名为大力神

root.elements().get(2).elements().get(0).setText("大力神");

// 格式化输出器

OutputFormat format = OutputFormat.createPrettyPrint();

format.setEncoding("utf-8");

// 文件输出流

FileOutputStream fos = new FileOutputStream("src/com/kaige/xml/dom4j.xml");

// xml写入器

XMLWriter writer = new XMLWriter(fos, format);

// 写数据

writer.write(document);

// 关闭

writer.close();

System.out.println("修改完成");

}