# 一 认识XML

XML 指可扩展标记语言（eXtensible Markup Language）。

XML 是一种标记语言，很类似 HTML

XML 的设计宗旨是传输数据，而非显示数据

XML 标签没有被预定义。需要自行定义标签。

XML 与 HTML 的主要差异:

XML 不是 HTML 的替代。

XML 和 HTML 为不同的目的而设计：

XML 被设计为传输和存储数据，其焦点是数据的内容。

HTML 被设计用来显示数据，其焦点是数据的外观。

HTML 旨在显示信息，而 XML 旨在传输信息。

# 二 XML语法

## 1 声明

一般放在第一行，指定xml的版本和使用的编码

## 2 元素

也可称为节点

XML 文档必须有一个根元素

Xml中元素包含开始标签和结束标签（必须有结束标签）

标签名区分大小写

元素中可以包含其他元素、文本

根元素：最外层的元素

子元素：元素中包含的其他元素

## 3 属性

元素的开始标签中可以包含属性

属性值必须使用单引号或双引号括起来

## 4 注释

<!-- 注释内容 -->

# 三 XML解析

## 1 DOM解析

将整个文件读到内存中，然后根据节点的关系，解析相关内容

本例使用dom4j进行解析

需要导入dom4j-1.6.1.jar

dom4j是一套开源的类库；集成的 XPath(路径查询语言)支持、XML Schema(约束模式文档)支持

1）主要涉及对象：

Document：用于描述XML文档。

Element：用于描述XML文档中的元素。

Attribute：用于描述XML文档中的元素的属性

2）常用方法

获取xml文档对象Document。

File xmlFile = new File(“test.xml”);

SAXReader reader = new SAXReader();

Document xmlDoc = reader.read(xmlFile);

获取根元素对象

Element root = xmlDoc.getRootElement();

获取根元素名称

String elementName = root.getName();

通过元素名称获取直接子元素对象

Element element = root.element(“元素名称");

获取所有直接子元素对象集合

List elementList = root.elements();

获取element元素的内部文本

String text = element.getText();

获取element子元素内部文本

String text = element.elementText(“子元素名称”);

通过属性名获取元素属性对象

Attribute attribute = element.attribute(“属性名称");

通过索引获取属性对象

Attribute attribute = element.attribute(0);

获取所有属性对象集合

List attributeList = root.attributes();

通过属性对象获取属性值

String value = attribute.getValue();

通过元素对象获取属性值

String value = element.attributeValue(“属性名称”);

**public** **void** parse2() **throws** Exception{

SAXReader reader = **new** SAXReader();

Document doc = reader.read(**new** File(Demo1.**class**.getResource("/contact.xml").getPath()));

//获取文档的根节点

Element root = doc.getRootElement();

//根据节点名称获取节点对象

List<Element> eles = root.elements("contact");

**for** (Element contact : eles) {

//根据属性名得到属性对象

Attribute attr = contact.attribute("id");

System.***out***.println(attr.getValue());

// List<Element> cEles = contact.elements();

// for (Element c : cEles) {

// System.out.println(c.getText());

// }

//根据子节点名称获取子节点对象

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

System.***out***.println(name.getText());

System.***out***.println(contact.elementText("age"));

System.***out***.println(contact.elementText("phone"));

System.***out***.println(contact.elementText("email"));

System.***out***.println(contact.elementText("qq"));

}

}

3）DOM方式解析存在的问题：

DOM 构建整个文档驻留内存的树。如果文档很大，就会要求有极大的内存。

DOM 创建表示原始文档中每个东西的对象，包括元素、文本、属性和空格。如果只需关注原始文档的一小部分，那么创建那些永远不被使用的对象是极其浪费的。

DOM 解析器必须在代码取得控制权之前读取整个文档。对于非常大的文档，这会引起显著的延迟

## 2 SAX方式

SAX(Simple API for XML)：采用从上而下的基于事件驱动的方式，解析速度快并且占用内存少。SAX解析器自动触发事件，并回调相应的方法，实现对xml文档流的解析

所谓事件，其实就是一些回调（callback）方法，这些方法(事件)定义在ContentHandler接口中。因此，sax解析最重要的步骤就是定义一个Handler处理类，让其继承DefaultHandler，从而重写其回调方法

回调方法：

**startDocument()：**只会在文档开始解析的时候被调用，每次解析只会调用一次。

**startElement()：**每次开始解析一个元素，即遇到元素标签开始的时候都会调用。

**characters()：**每次解析到元素标签携带的内容时都会调用，即使该元素标签的内容为空或换行。可以使用new String(ch,off,len)获取内容。

**endElement()：**每次结束解析一个元素，即遇到元素标签结束的时候都会调用。

**endDocument()：**只会在文档解析结束的时候被调用，每次解析只会调用一次。

**public** **static** **void** main(String[] args) **throws** ParserConfigurationException, SAXException, IOException {

// **TODO** Auto-generated method stub

//创建sax解析器的工厂对象

SAXParserFactory sf = SAXParserFactory.*newInstance*();

//通过工厂对象创建sax解析器

SAXParser sp = sf.newSAXParser();

SaxService saxService= **new** SaxService();

//String path = Demo.class.getResource("UserInfo.xml").getPath();

InputStream in = Demo.**class**.getResourceAsStream("UserInfo.xml");

//解析

sp.parse(in, saxService);

}

//自定义解析类继承DefaultHandler

**public** **class** SaxService **extends** DefaultHandler{

List<User> list = **null**;

User user = **null**;

//表示当前解析的节点

String tagName = **null**;

//开始解析文档

@Override

**public** **void** startDocument() **throws** SAXException {

// **TODO** Auto-generated method stub

//super.startDocument();

list = **new** ArrayList<>();

}

//解析文档结束

@Override

**public** **void** endDocument() **throws** SAXException {

// **TODO** Auto-generated method stub

//super.endDocument();

System.***out***.println(list);

}

//开始解析某个节点

@Override

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

// **TODO** Auto-generated method stub

System.***out***.println(qName);

// System.out.println(uri);

// System.out.println(localName);

//如果是user节点创建对象

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

user = **new** User();

user.setId(Integer.*parseInt*(attributes.getValue("id")));

}

tagName = qName;

}

//节点解析结束

@Override

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

// **TODO** Auto-generated method stub

//解析节点结束，将对象放入集合

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

list.add(user);

}

tagName = **null**;

}

//获取节点的内容

@Override

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

// **TODO** Auto-generated method stub

**if**(tagName == **null**){

**return**;

}

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

**switch**(tagName){

**case** "name":

user.setName(content);

**break**;

**case** "age":

user.setAge(Integer.*parseInt*(content));

**break**;

**case** "hobby":

user.setHobby(content);

**break**;

}

}

}

# 四 XPath语法

XPath 是在 XML 文档中查找信息的语言

XPath 是通过元素和属性进行查找

XPath简化了Dom4j查找节点的过程

使用XPath是为了更快更好地选取我们想要选取的XML元素。

需要导入jaxen-1.1-beta-6.jar

主要方法：

获取所有符合条件的节点

selectNodes(String xpathExpression) 返回List集合

获取符合条件的单个节点

selectSingleNode(String xpathExpression) 返回一个Node对象。

如果符合条件的节点有多个，那么返回第一个。

SAXReader reader = **new** SAXReader();

Document doc = reader.read(**new** File("./bin/Book.xml"));

//查询到指定路径的所有节点

List<Element> selectNodes = doc.selectNodes("/root/books/book/name");

**for** (Element ele : selectNodes) {

//asXML 将Element转换为xml格式的字符串

// System.out.println(ele.asXML());

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

}

//查询到所有的name节点，而不管他们的层次关系

selectNodes = doc.selectNodes("//name");

**for** (Element ele : selectNodes) {

//asXML 将Element转换为xml格式的字符串

System.***out***.println(ele.asXML());

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

}

//查询book节点下，name子节点内容为呐喊的book节点

Element ele = (Element)doc.selectSingleNode("/root/books/book[name='呐喊']");

System.***out***.println(ele.asXML());

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

//根据节点的属性值查询节点，属性名前使用@

ele = (Element)doc.selectSingleNode("/root/books/book[@id='1']");

System.***out***.println(ele.asXML());

}