使用SAX或者DOM或者pull解析XML文件

在Android平台上可以使用Simple API for XML(SAX) 、 Document Object Model(DOM)和Android附带的pull解析器解析XML文件。 下面是本例子要解析的XML文件：

文件名称：itcast.xml

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

<persons>

<person id="23">

<name>李明</name>

<age>30</age>

</person>

<person id="20">

<name>李向梅</name>

<age>25</age>

</person>

</persons>

例子定义了一个javabean用于存放上面解析出来的xml内容， 这个javabean为Person，代码请见本页下面备注：

public class Person {

private Integer id;

private String name;

private Short age;

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public Short getAge() {

return age;

}

public void setAge(Short age) {

this.age = age;

}

}

使用SAX读取XML文件

SAX是一个解析速度快并且占用内存少的xml解析器，非常适合用于Android等移动设备。 SAX解析XML文件采用的是事件驱动，也就是说，它并不需要解析完整个文档，在按内容顺序解析文档的过程中，SAX会判断当前读到的字符是否合法XML语法中的某部分，如果符合就会触发事件。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。下面是一些ContentHandler接口常用的方法：

**startDocument()**

当遇到文档的开头的时候，调用这个方法，可以在其中做一些预处理的工作。

**endDocument()**

和上面的方法相对应，当文档结束的时候，调用这个方法，可以在其中做一些善后的工作。

**startElement(String namespaceURI, String localName, String qName, Attributes atts)**

当读到一个开始标签的时候，会触发这个方法。namespaceURI就是命名空间，localName是不带命名空间前缀的标签名，qName是带命名空间前缀的标签名。通过atts可以得到所有的属性名和相应的值。要注意的是SAX中一个重要的特点就是它的流式处理，当遇到一个标签的时候，它并不会纪录下以前所碰到的标签，也就是说，在startElement()方法中，所有你所知道的信息，就是标签的名字和属性，至于标签的嵌套结构，上层标签的名字，是否有子元属等等其它与结构相关的信息，都是不得而知的，都需要你的程序来完成。这使得SAX在编程处理上没有DOM来得那么方便。

**endElement(String uri, String localName, String name)**

这个方法和上面的方法相对应，在遇到结束标签的时候，调用这个方法。

**characters(char[] ch, int start, int length)**

这个方法用来处理在XML文件中读到的内容，第一个参数为文件的字符串内容，后面两个参数是读到的字符串在这个数组中的起始位置和长度，使用new String(ch,start,length)就可以获取内容。

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

<persons>

<person id="23">

<name>李明</name>

<age>30</age>

</person>

<person id="20">

<name>李向梅</name>

<age>25</age>

</person>

</persons>

解析itcast.xml触发的事件为：

读到的标签及内容 触发事件

{文档开始} startDocument()

<persons> startElement(, "persons", null, "{Attributes}")

"\n\t" characters("<persons>...</persons>", "12", "2")

<person> startElement(, "person", null, "{Attributes}")

"\n\t\t" characters("<persons>...</persons>", "31", "3")

<name> startElement(, "name", null, "{Attributes}")

"李明" characters("<persons>...</persons>", "40", "2")

</name> endElement("", "name", null)

"\n\t\t" characters("<persons>...</persons>", "50", "3")

<age> startElement(, "age", null, "{Attributes}")

"30" characters("<persons>...</persons>", "58", "2")

</age> endElement("", "age", null)

"\n\t" characters("<persons>...</persons>", "67", "2")

</person> endElement("", "person", null)

"\n\t" characters("<persons>...</persons>", "79", "2")

<person> startElement(, "person", null, "{Attributes}")

"\n\t\t" characters("<persons>...</persons>", "98", "3")

<name> startElement(, "name", null, "{Attributes}")

"李向梅" characters("<persons>...</persons>", "107", "3")

</name> endElement("", "name", null)

"\n\t\t" characters("<persons>...</persons>", "118", "3")

<age> startElement(, "age", null, "{Attributes}")

"25" characters("<persons>...</persons>", "126", "2")

</age> endElement("", "age", null)

"\n\t" characters("<persons>...</persons>", "135", "2")

</person> endElement("", "person", null)

"\n" characters("<persons>...</persons>", "147", "1")

</persons> endElement("", "persons", null)

{文档结束} endDocument()

使用SAX读取XML文件

只要为SAX提供实现ContentHandler接口的类，那么该类就可以得到通知事件（实际上就是SAX调用了该类中的回调方法）。因为ContentHandler是一个接口，在使用的时候可能会有些不方便，因此，SAX还为其制定了一个Helper类：DefaultHandler，它实现了ContentHandler接口，但是其所有的方法体都为空，在实现的时候，你只需要继承这个类，然后重写相应的方法即可。使用SAX解析itcast.xml的代码如下：

public static List<Person> readXML(InputStream inStream) {

try {

SAXParserFactory spf = SAXParserFactory.newInstance();

SAXParser saxParser = spf.newSAXParser(); //创建解析器

//设置解析器的相关特性，http://xml.org/sax/features/namespaces = true 表示开启命名空间特性

//saxParser.setProperty("http://xml.org/sax/features/namespaces",true);

XMLContentHandler handler = new XMLContentHandler();

saxParser.parse(inStream, handler);

inStream.close();

return handler.getPersons();

} catch (Exception e) {

e.printStackTrace();

}

return null;

}

SAX 支持已内置到JDK1.5中，你无需添加任何的jar文件。关于XMLContentHandler的代码实现请看本页下面备注。

import java.util.ArrayList;

import java.util.List;

import org.xml.sax.Attributes;

import org.xml.sax.SAXException;

import org.xml.sax.helpers.DefaultHandler;

import cn.itcast.xml.domain.Person;

public class XMLContentHandler extends DefaultHandler {

private List<Person> persons = null;

private Person currentPerson;

private String tagName = null;//当前解析的元素标签

public List<Person> getPersons() {

return persons;

}

/\*

\* 接收文档的开始的通知。

\*/

@Override public void startDocument() throws SAXException {

persons = new ArrayList<Person>();

}

/\*

\* 接收字符数据的通知。

\*/

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

if(tagName!=null){

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

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

this.currentPerson.setName(data);

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

this.currentPerson.setAge(Short.parseShort(data));

}

}

}

/\*

\* 接收元素开始的通知。

\* 参数意义如下：

\* namespaceURI：元素的命名空间

\* localName ：元素的本地名称（不带前缀）

\* qName ：元素的限定名（带前缀）

\* atts ：元素的属性集合

\*/

@Override public void startElement(String namespaceURI, String localName, String qName, Attributes atts) throws SAXException {

if(localName.equals("person")){

currentPerson = new Person();

currentPerson.setId(Integer.parseInt(atts.getValue("id")));

}

this.tagName = localName;

}

/\*

\* 接收文档的结尾的通知。

\* 参数意义如下：

\* uri ：元素的命名空间

\* localName ：元素的本地名称（不带前缀）

\* name ：元素的限定名（带前缀）

\*

\*/

@Override public void endElement(String uri, String localName, String name) throws SAXException {

if(localName.equals("person")){

persons.add(currentPerson);

currentPerson = null;

}

this.tagName = null;

}

}

使用DOM读取XML文件

除了使用 SAX可以解析XML文件，大家也可以使用熟悉的DOM来解析XML文件。 DOM解析XML文件时，会将XML文件的所有内容以文档树方式存放在内存中，然后允许您使用DOM API遍历XML树、检索所需的数据。使用DOM操作XML的代码看起来是比较直观的，并且在编码方面比基于SAX的实现更加简单。但是，因为DOM需要将XML文件的所有内容以文档树方式存放在内存中，所以内存的消耗比较大，特别对于运行Android的移动设备来说，因为设备的资源比较宝贵，所以建议还是采用SAX来解析XML文件，当然，如果XML文件的内容比较小采用DOM也是可行的。

代码请看本页下方备注

import java.io.InputStream;

import java.util.ArrayList;

import java.util.List;

import javax.xml.parsers.DocumentBuilder;

import javax.xml.parsers.DocumentBuilderFactory;

import org.w3c.dom.Document;

import org.w3c.dom.Element;

import org.w3c.dom.Node;

import org.w3c.dom.NodeList;

import cn.itcast.xml.domain.Person;

/\*\*

\* 使用Dom解析xml文件

\*

\*/

public class DomXMLReader {

public static List<Person> readXML(InputStream inStream) {

List<Person> persons = new ArrayList<Person>();

DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();

try {

DocumentBuilder builder = factory.newDocumentBuilder();

Document dom = builder.parse(inStream);

Element root = dom.getDocumentElement();

NodeList items = root.getElementsByTagName("person");//查找所有person节点

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

Person person = new Person();

//得到第一个person节点

Element personNode = (Element) items.item(i);

//获取person节点的id属性值

person.setId(new Integer(personNode.getAttribute("id")));

//获取person节点下的所有子节点(标签之间的空白节点和name/age元素)

NodeList childsNodes = personNode.getChildNodes();

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

Node node = (Node) childsNodes.item(j);  
 //判断是否为元素类型

if(node.getNodeType() == Node.ELEMENT\_NODE){ Element childNode = (Element) node;

//判断是否name元素

if ("name".equals(childNode.getNodeName())) {

//获取name元素下Text节点,然后从Text节点获取数据 person.setName(childNode.getFirstChild().getNodeValue());

} else if (“age”.equals(childNode.getNodeName())) {

person.setAge(new Short(childNode.getFirstChild().getNodeValue()));

}

}

}

persons.add(person);

}

inStream.close();

} catch (Exception e) {

e.printStackTrace();

}

return persons;

}

使用Pull解析器读取XML文件

除了可以使用 SAX和DOM解析XML文件，大家也可以使用Android内置的Pull解析器解析XML文件。 Pull解析器的运行方式与 SAX 解析器相似。它提供了类似的事件，如：开始元素和结束元素事件，使用parser.next()可以进入下一个元素并触发相应事件。事件将作为数值代码被发送，因此可以使用一个switch对感兴趣的事件进行处理。当元素开始解析时，调用parser.nextText()方法可以获取下一个Text类型节点的值。

使用Pull解析器读取itcast.xml的代码在本页下方备注

Pull解析器的源码及文档下载网址：http://www.xmlpull.org/

import org.xmlpull.v1.XmlPullParser;

import android.util.Xml;

import cn.itcast.xml.domain.Person;

public class PullXMLReader {

public static List<Person> readXML(InputStream inStream) {

XmlPullParser parser = Xml.newPullParser();

try {

parser.setInput(inStream, "UTF-8");

int eventType = parser.getEventType();

Person currentPerson = null;

List<Person> persons = null;

while (eventType != XmlPullParser.END\_DOCUMENT) {

switch (eventType) {

case XmlPullParser.START\_DOCUMENT://文档开始事件,可以进行数据初始化处理

persons = new ArrayList<Person>();

break;

case XmlPullParser.START\_TAG://开始元素事件

String name = parser.getName();

if (name.equalsIgnoreCase("person")) {

currentPerson = new Person();

currentPerson.setId(new Integer(parser.getAttributeValue(null, "id")));

} else if (currentPerson != null) {

if (name.equalsIgnoreCase("name")) {

currentPerson.setName(parser.nextText());// 如果后面是Text节点,即返回它的值

} else if (name.equalsIgnoreCase("age")) {

currentPerson.setAge(new Short(parser.nextText()));

}

}

break;

case XmlPullParser.END\_TAG://结束元素事件

if (parser.getName().equalsIgnoreCase("person") && currentPerson != null) {

persons.add(currentPerson);

currentPerson = null;

}

break;

}

eventType = parser.next();

}

inStream.close();

return persons;

} catch (Exception e) {

e.printStackTrace();

}

return null;

}

}

有些时候，我们需要生成一个XML文件，生成XML文件的方法有很多，如：可以只使用一个StringBuilder组拼XML内容，然后把内容写入到文件中；或者使用DOM API生成XML文件，或者也可以使用pull解析器生成XML文件，这里推荐大家使用Pull解析器。

使用Pull解析器生成一个与itcast.xml文件内容相同的myitcast.xml文件，代码在本页下方备注

使用代码如下（生成XML文件）：

File xmlFile = new File("myitcast.xml");

FileOutputStream outStream = new FileOutputStream(xmlFile);

OutputStreamWriter outStreamWriter = new OutputStreamWriter(outStream, "UTF-8");

BufferedWriter writer = new BufferedWriter(outStreamWriter);

writeXML(persons, writer);

writer.flush();

writer.close();

如果只想得到生成的xml字符串内容，可以使用StringWriter：

StringWriter writer = new StringWriter();

writeXML(persons, writer);

String content = writer.toString();

public static String writeXML(List<Person> persons, Writer writer){

XmlSerializer serializer = Xml.newSerializer();

try {

serializer.setOutput(writer);

serializer.startDocument("UTF-8", true);

//第一个参数为命名空间,如果不使用命名空间,可以设置为null

serializer.startTag("", "persons");

for (Person person : persons){

serializer.startTag("", "person");

serializer.attribute("", "id", person.getId().toString());

serializer.startTag("", "name");

serializer.text(person.getName());

serializer.endTag("", "name");

serializer.startTag("", "age");

serializer.text(person.getAge().toString());

serializer.endTag("", "age");

serializer.endTag("", "person");

}

serializer.endTag("", "persons");

serializer.endDocument();

return writer.toString();

} catch (Exception e) {

e.printStackTrace();

}

return null;

}