SAX即是：Simple API for XML

SAX是基于事件驱动的。当然android的事件机制是基于回调函数的，在用SAX解析xml文档时候，在读取到文档开始和结束标签时候就会回调一个事件，在读取到其他节点与内容时候也会回调一个事件。

既然涉及到事件，就有事件源，事件处理器。在SAX接口中，事件源是org.xml.sax包中的XMLReader，它通过parser()方法来解析XML文档，并产生事件。事件处理器是org.xml.sax包中ContentHander、DTDHander、ErrorHandler，以及EntityResolver这4个接口

XMLReader通过相应事件处理器注册方法setXXXX()来完成的与ContentHander、DTDHander、ErrorHandler，以及EntityResolver这4个接口的连接，详细介绍请见下表：

但是我们无需都继承这4个接口，SDK为我们提供了DefaultHandler类来处理，DefaultHandler类的一些主要事件回调方法如下：

由以上可知，我们需要XmlReader 以及DefaultHandler来配合解析xml。

处理思路是：

1：创建SAXParserFactory对象

2: 根据SAXParserFactory.newSAXParser()方法返回一个SAXParser解析器

3：根据SAXParser解析器获取事件源对象XMLReader

4：实例化一个DefaultHandler对象

5：连接事件源对象XMLReader到事件处理类DefaultHandler中

6：调用XMLReader的parse方法从输入源中获取到的xml数据

7：通过DefaultHandler返回我们需要的数据集合。

代码如下：

public List<River> parse(String xmlPath){

List<River> rivers=null;

SAXParserFactory factory=SAXParserFactory.newInstance();

try {

SAXParser parser=factory.newSAXParser();

//获取事件源

XMLReader xmlReader=parser.getXMLReader();

//设置处理器

RiverHandler handler=new RiverHandler();

xmlReader.setContentHandler(handler);

//解析xml文档

//xmlReader.parse(new InputSource(new URL(xmlPath).openStream()));

xmlReader.parse(new InputSource(this.context.getAssets().open(xmlPath)));

rivers=handler.getRivers();

} catch (ParserConfigurationException e) {

// TODO Auto-generated catch block

e.printStackTrace();

} catch (SAXException e) {

// TODO Auto-generated catch block

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

}

return rivers;

}

重点在于DefaultHandler对象中对每一个元素节点，属性，文本内容，文档内容进行处理。

前面说过DefaultHandler是基于事件处理模型的，基本处理方式是：当SAX解析器导航到文档开始标签时回调startDocument方法，导航到文档结束标签时回调endDocument方法。当SAX解析器导航到元素开始标签时回调startElement方法，导航到其文本内容时回调characters方法，导航到标签结束时回调endElement方法。

根据以上的解释，我们可以得出以下处理xml文档逻辑：

1：当导航到文档开始标签时，在回调函数startDocument中，可以不做处理，当然你可以验证下UTF-8等等。

2：当导航到rivers开始标签时，在回调方法startElement中可以实例化一个集合用来存贮list，不过我们这里不用，因为在构造函数中已经实例化了。

3：导航到river开始标签时，就说明需要实例化River对象了，当然river标签中还有name ,length属性，因此实例化River后还必须取出属性值，attributes.getValue(NAME)，同时赋予river对象中，同时添加为导航到的river标签添加一个boolean为真的标识，用来说明导航到了river元素。

4:当然有river标签内还有子标签（节点），但是SAX解析器是不知道导航到什么标签的，它只懂得开始，结束而已。那么如何让它认得我们的各个标签呢？当然需要判断了，于是可以使用回调方法startElement中的参数String localName，把我们的标签字符串与这个参数比较下，就可以了。我们还必须让SAX知道，现在导航到的是某个标签，因此添加一个true属性让SAX解析器知道。因此

5：它还会导航到文本内标签，（就是<img></img>里面的内容），回调方法characters，我们一般在这个方法中取出就是<img></img>里面的内容，并保存。

6：当然它是一定会导航到结束标签</river> 或者</rivers>的，如果是</river>标签，记得把river对象添加进list中。如果是river中的子标签</introduction>，就把前面设置标记导航到这个标签的boolean标记设置为false.

按照以上实现思路，可以实现如下代码：

/\*\*导航到开始标签触发\*\*/

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

String tagName=localName.length()!=0?localName:qName;

tagName=tagName.toLowerCase().trim();

//如果读取的是river标签开始，则实例化River

if(tagName.equals(RIVER)){

isRiver=true;

river=new River();

/\*\*导航到river开始节点后\*\*/

river.setName(attributes.getValue(NAME));

river.setLength(Integer.parseInt(attributes.getValue(LENGTH)));

}

//然后读取其他节点

if(isRiver){

if(tagName.equals(INTRODUCTION)){

xintroduction=true;

}else if(tagName.equals(IMAGEURL)){

ximageurl=true;

}

}

}

/\*\*导航到结束标签触发\*\*/

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

String tagName=localName.length()!=0?localName:qName;

tagName=tagName.toLowerCase().trim();

//如果读取的是river标签结束，则把River添加进集合中

if(tagName.equals(RIVER)){

isRiver=true;

rivers.add(river);

}

//然后读取其他节点

if(isRiver){

if(tagName.equals(INTRODUCTION)){

xintroduction=false;

}else if(tagName.equals(IMAGEURL)){

ximageurl=false;

}

}

}

//这里是读取到节点内容时候回调

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

//设置属性值

if(xintroduction){

//解决null问题

river.setIntroduction(river.getIntroduction()==null?"":river.getIntroduction()+new String(ch,start,length));

}else if(ximageurl){

//解决null问题

river.setImageurl(river.getImageurl()==null?"":river.getImageurl()+new String(ch,start,length));

}

}