title: Android基础入门教程——7.2.1 Android XML数据解析
categories: Android
[基础入门，教程]

Android基础入门教程——7.2.1 Android XML数据解析

本节引言：

> 前面两节我们对Android内置的Http请求方式：HttpURLConnection和HttpClient，本来以为OkHttp
> 已经集成进来了，然后想讲解下Okhttp的基本用法，后来发现还是要导第三方，算了，放到进阶部分
> 吧，而本节我们来学习下Android为我们提供的三种解析XML数据的方案！他们分别是：
> SAX,DOM,PULL三种解析方式，下面我们就来对他们进行学习！

1.XML数据要点介绍

首先我们来看看XML数据的一些要求以及概念：

XML格式数据的简单理解

全名叫做可扩展标记语言,类似于HTML(超文本标记),这里不深究他的定义,只介绍他是用来做什么的!其实我们更多的时候是把xml用来存储数据,可以看作一个微型的数据库,比如我们前面学的SharedPreference就是使用xml文件来保存配置信息的,而我们的SQLite其实底层也是一个xml文件;而在网络应用方面通常作为信息的载体,我们通常把数据包装成xml来传递

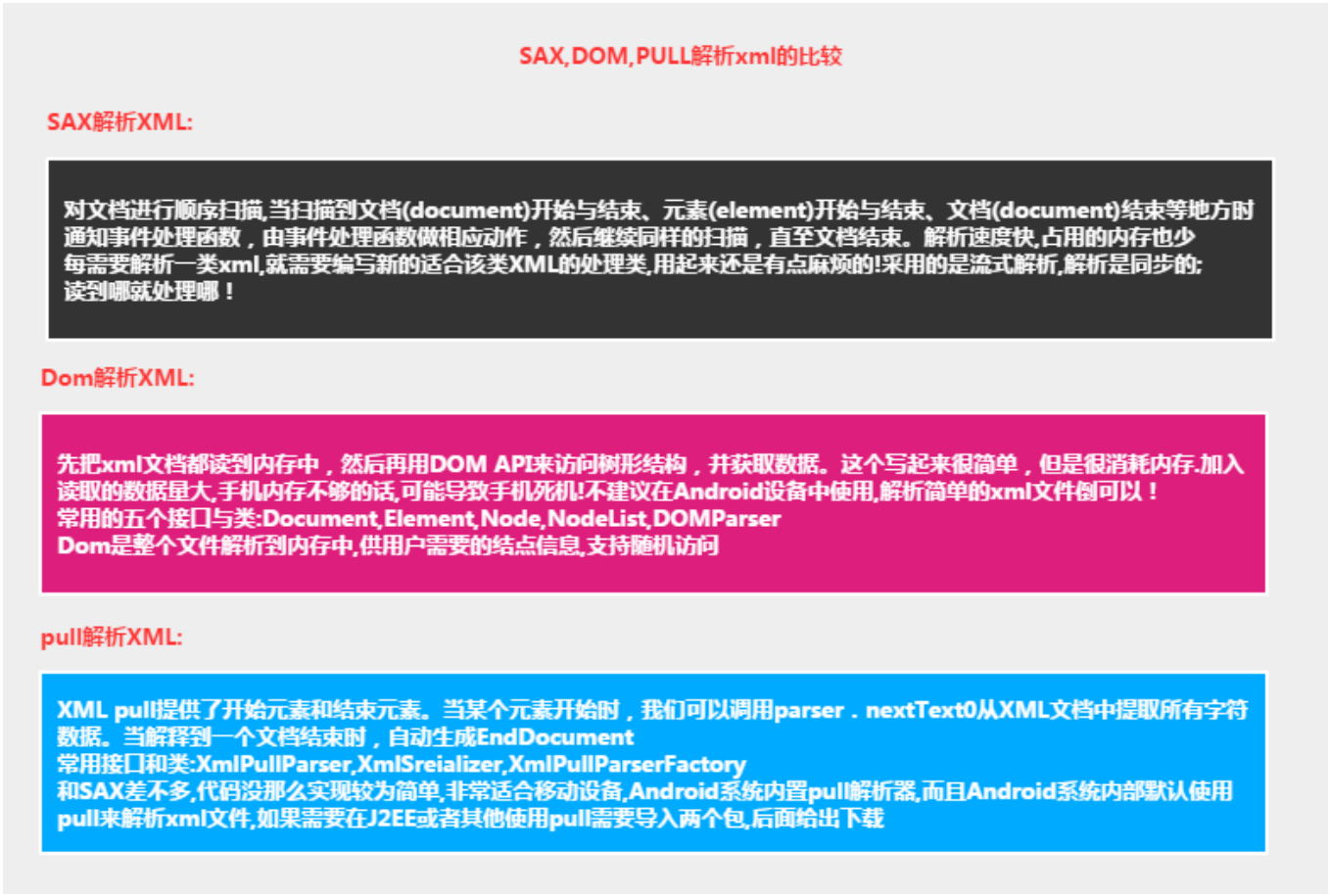
```
<?xml version="1.0" encoding="UTF-8"?>
<persons>
    <person id = "11">
        <name>Coder-pig</name>
        <age>20</age>
    </person>
    <person id = "13">
        <name>Jay</name>
        <age>20</age>
    </person>
</persons>
```

对应结构

文档开始
开始元素(persons)
文本结点(空白文本) 开始元素(person) 属性
文本结点(空白文本) 开始元素(name) 文本结点 结束元素
文本结点(空白文本) 开始元素(age) 文本结点 结束元素
文本结点(空白文本) 结束元素
...
结束元素(persons)
文档结束

ps:上面就是简单的定义一个存储person对象的xml文件的编码;要注意一点哦,外面的空白区域也是文本结点哦!

2.三种解析XML方法的比较



3.SAX解析XML数据

SAX解析xml文件

SAX是一个解析速度快且占用内存少的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;也就是不需要解析整个文档,而是在解析文档的过程中,判断读到的字符是否符合xml语法的某部分(文件开头,文档结束,或者标签开头与标签结束时)符合的话就会触发事件(回调方法)而这些方法都定义在ContentHandler接口中,而ContentHanlder是一个接口,使用起来有些不方便,而Android很贴心地为我们提供了一个帮助类:DefaultHandler,只需要继承这个类,重写对应的方法即可

可供重写的方法如下：

startDocument():	当读取到文档开始标志时触发,通常在这里完成一些初始化操作
endDocument():	文档结束部分,在这里完成一些善后工作
startElement(namespaceURI,localName,qName,atts):	参数依次为命名空间;不带命名空间的前缀标签名;带命名控件前缀名的标签,通过atts可以得到所有的属性名与相应的值；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(uri,localName,name)	在遇到结束标签的时候，调用这个方法
characters(ch,start,length)	这个方法用来处理在XML文件中读到的内容，第一个参数用于存放文件的内容，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。

核心代码：

SAX解析类：**SaxHelper.java**：

```
1.  /**
2.   * Created by Jay on 2015/9/8 0008.
3.   */
4.  public class SaxHelper extends DefaultHandler {
5.      private Person person;
6.      private ArrayList<Person> persons;
7.      //当前解析的元素标签
8.      private String tagName = null;
9.
10.     /**
11.      * 当读取到文档开始标志是触发，通常在这里完成一些初始化操作
12.      */
13.     @Override
14.     public void startDocument() throws SAXException {
15.         this.persons = new ArrayList<Person>();
16.         Log.i("SAX", "读取到文档头,开始解析xml");
```

```
17.         }
18.
19.
20.         /**
21.          * 读到一个开始标签时调用,第二个参数为标签名,最后一个参数为属性数组
22.          */
23.         @Override
24.         public void startElement(String uri, String localName, String qName
    ,
25.                                  Attributes attributes) throws SAXException
    {
26.             if (localName.equals("person")) {
27.                 person = new Person();
28.                 person.setId(Integer.parseInt(attributes.getValue("id")));
29.                 Log.i("SAX", "开始处理person元素~");
30.             }
31.             this.tagName = localName;
32.         }
33.
34.
35.         /**
36.          * 读到到内容,第一个参数为字符串内容,后面依次为起始位置与长度
37.          */
38.
39.         @Override
40.         public void characters(char[] ch, int start, int length)
41.                 throws SAXException {
42.             //判断当前标签是否有效
43.             if (this.tagName != null) {
44.                 String data = new String(ch, start, length);
45.                 //读取标签中的内容
46.                 if (this.tagName.equals("name")) {
47.                     this.person.setName(data);
48.                     Log.i("SAX", "处理name元素内容");
49.                 } else if (this.tagName.equals("age")) {
50.                     this.person.setAge(Integer.parseInt(data));
51.                     Log.i("SAX", "处理age元素内容");
52.                 }
53.
54.             }
55.
56.         }
57.
58.         /**
59.          * 处理元素结束时触发,这里将对象添加到结合中
```

```
         */
        @Override
        public void endElement(String uri, String localName, String qName)
                throws SAXException {
            if (localName.equals("person")) {
                this.persons.add(person);
                person = null;
                Log.i("SAX", "处理person元素结束~");
            }
            this.tagName = null;
        }

        /**
         * 读取到文档结尾时触发,
         */
        @Override
        public void endDocument() throws SAXException {
            super.endDocument();
            Log.i("SAX", "读取到文档尾,xml解析结束");
        }

        //获取persons集合
        public ArrayList<Person> getPersons() {
            return persons;
        }

    }
```

然后我们在MainActivity.java中写上写上这样一个方法，然后要解析XML的时候调用下就好了~

```
        private ArrayList<Person> readxmlForSAX() throws Exception {
            //获取文件资源建立输入流对象
            InputStream is = getAssets().open("person1.xml");
            //①创建XML解析处理器
            SaxHelper ss = new SaxHelper();
            //②得到SAX解析工厂
            SAXParserFactory factory = SAXParserFactory.newInstance();
            //③创建SAX解析器
            SAXParser parser = factory.newSAXParser();
            //④将xml解析处理器分配给解析器,对文档进行解析,将事件发送给处理器
            parser.parse(is, ss);
            is.close();
            return ss.getPersons();
```

```
14.         }
```

一些其他的话：

嗯，对了，忘记给大家说下我们是定义下面这样一个person1.xml文件，然后放到assets目录下的！
文件内容如下：**person1.xml**

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <persons>
3.      <person id = "11">
4.          <name>SAX解析</name>
5.          <age>18</age>
6.      </person>
7.      <person id = "13">
8.          <name>XML1</name>
9.          <age>43</age>
10.     </person>
11. </persons>
```

我们是把三种解析方式都糅合到一个demo中，所以最后才贴全部的效果图，这里的话，贴下打印的Log，
相信大家会对SAX解析XML流程更加明了：

```
I/SAX：  读取到文档头,开始解析xml
I/SAX：  开始处理person元素~
I/SAX：  处理name元素内容
I/SAX：  处理age元素内容
I/SAX：  处理person元素结束~
I/SAX：  开始处理person元素~
I/SAX：  处理name元素内容
I/SAX：  处理age元素内容
I/SAX：  处理person元素结束~
I/SAX：  读取到文档尾,xml解析结束
```

另外，外面的空白文本也是文本节点哦！解析的时候也会走这些节点！

4.DOM解析XML数据

Dom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的形式存放到内存中,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。

先了解一下Dom的一些api

DocumentBuilderFactory (解析器工厂类)	创建方法: DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
DocumentBuilder (解析器类)	创建方法: 通过解析器工厂类来获得 DocumentBuilder dbBuilder = dbFactory.newDocumentBuilder();
Document (文档树模型)	将要解析的xml文件读入Dom解析器 Document doc = dbBuilder.parse(context.getAssets().open("persons2.xml"));

ps:Document对象代表了一个xml文档的模型树,所有的其他Node都以一定的顺序包含在Document对象之内，排列成一个树状结构，以后对XML文档的所有操作都与解析器无关，

NodeList (结点列表类)	代表一个包含一个或者多个Node的列表,可看作数组,有以下两个方法: item(index):返回集合的第index个Node项　getLength():列表的节点数
Node (结点类)	Dom中最基本的对象,代表文档树中的抽象结点,很少会直接使用的;通常是调用他的子对象Element,Attr,Text等
Element (元素类)	Node最主要的自对象,再元素中可以包含属性,因此有取属性的方法: getAttribute()获得属性值;getTagName():元素的名称;
Attr (属性类)	代表某个元素的属性，虽然Attr继承自Node接口，但因为Attr是包含在Element中的，但并不能将其看做是Element的子对象，因为Attr并不是DOM树的一部分

核心代码：

DomHelper.java

```
/**
 * Created by Jay on 2015/9/8 0008.
 */
public class DomHelper {
    public static ArrayList<Person> queryXML(Context context)
    {
```

```
7.          ArrayList<Person> Persons = new ArrayList<Person>();
8.          try {
9.              //①获得DOM解析器的工厂示例：
10.             DocumentBuilderFactory dbFactory = DocumentBuilderFactory.n
    ewInstance();
11.             //②从Dom工厂中获得dom解析器
12.             DocumentBuilder dbBuilder = dbFactory.newDocumentBuilder();
13.             //③把要解析的xml文件读入Dom解析器
14.             Document doc = dbBuilder.parse(context.getAssets().open("pe
    rson2.xml"));
15.             System.out.println("处理该文档的DomImplemention对象=" + doc.ge
    tImplementation());
16.             //④得到文档中名称为person的元素的结点列表
17.             NodeList nList = doc.getElementsByTagName("person");
18.             //⑤遍历该集合,显示集合中的元素以及子元素的名字
19.             for(int i = 0;i < nList.getLength();i++)
20.             {
21.                 //先从Person元素开始解析
22.                 Element personElement = (Element) nList.item(i);
23.                 Person p = new Person();
24.                 p.setId(Integer.valueOf(personElement.getAttribute("id"
    )));
25.
26.                 //获取person下的name和age的Note集合
27.                 NodeList childNoList = personElement.getChildNodes();
28.                 for(int j = 0;j < childNoList.getLength();j++)
29.                 {
30.                     Node childNode = childNoList.item(j);
31.                     //判断子note类型是否为元素Note
32.                     if(childNode.getNodeType() == Node.ELEMENT_NODE)
33.                     {
34.                         Element childElement = (Element) childNode;
35.                         if("name".equals(childElement.getNodeName()))
36.
    p.setName(childElement.getFirstChild().getNodeValue());
37.                         else
    if("age".equals(childElement.getNodeName()))
38.                             p.setAge(Integer.valueOf(childElement.getFir
    stChild().getNodeValue()));
39.                     }
40.                 }
41.                 Persons.add(p);
42.             }
43.         } catch (Exception e) {e.printStackTrace();}
44.         return Persons;
```

```
45.         }
46.     }
```

代码分析：

> 从代码我们就可以看出DOM解析XML的流程，先整个文件读入Dom解析器，然后形成一棵树，
> 然后我们可以遍历节点列表获取我们需要的数据!

5.PULL解析XML数据

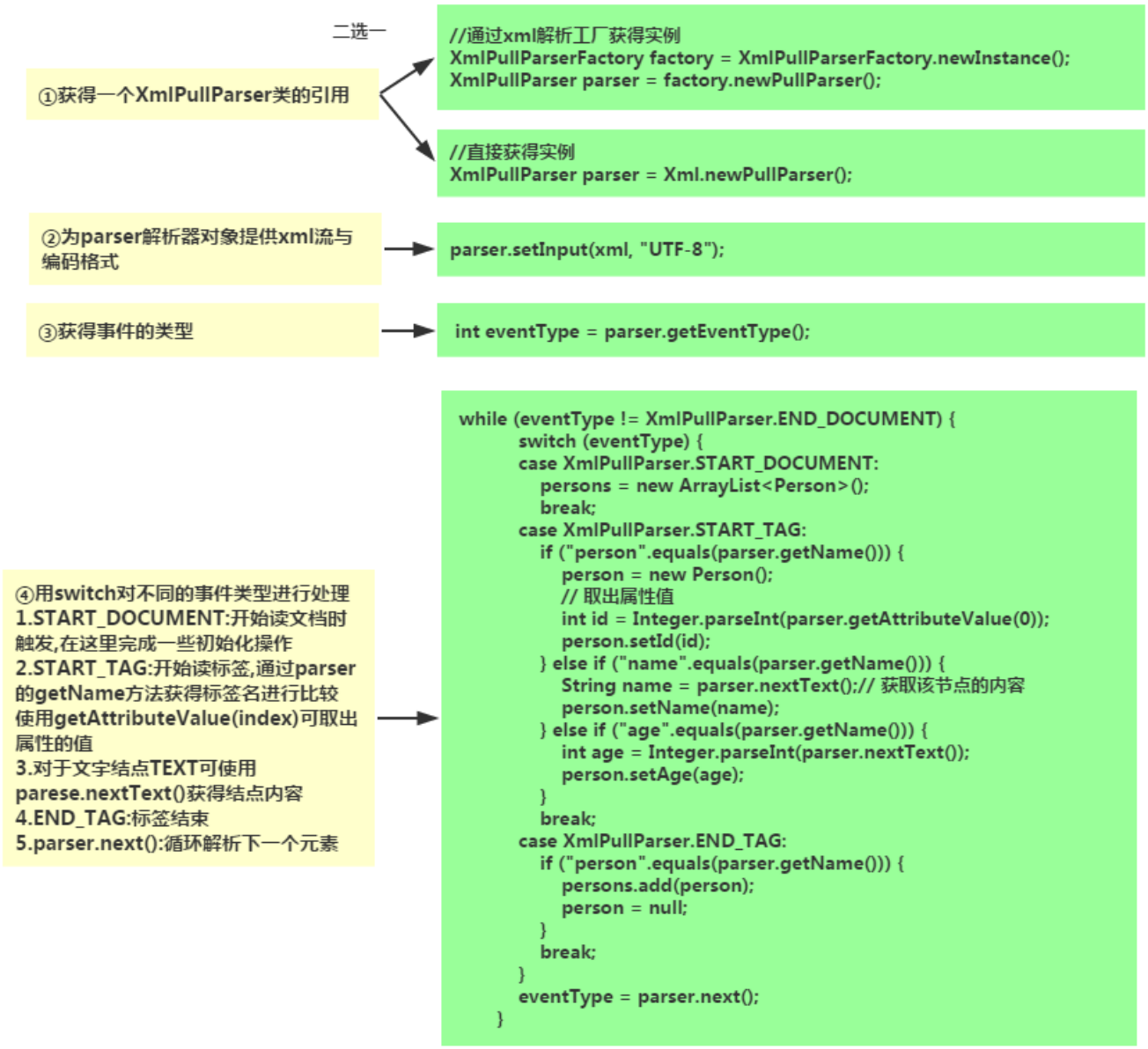
使用Pull解析xml

简单介绍

除了前面的SAX和DOM解析XML文件外,Android系统其实内置了Pull解析器用来解析xml文件,比如我们前面学到的SharedPreference就是使用的内置pull解析配置文件的！他的使用和SAX相似,都是采用事件驱动来完成xml的解析的;
而pull的编码较为简单,只需处理开始与结束事件,通常是使用switch语句,根据事件的不同类型,匹配不同的处理方式,
有五种事件:START_DOCUMENT;START_TAG;TEXT;END_TAG;END_DOCUMENT
XML pull提供了开始元素和结束元素。当某个元素开始时，可以调用parser.nextText从XML文档中提取所有字符数据。
当解析到一个文档结束时，自动生成EndDocument事件。在PULL解析过程中返回的是数字，且我们需要自己获取产生的事件然后做相应的操作，而不像SAX那样由处理器触发一种事件的方法，执行我们的代码：
读取到xml的声明返回 START_DOCUMENT； 结束返回 END_DOCUMENT ； 开始标签返回 START_TAG；
结束标签返回 END_TAG； 文本返回 TEXT。

使用PULL解析XML数据的流程：

使用Pull解析xml文件流程:



核心代码：

```
1.  public static ArrayList<Person> getPersons(InputStream xml)throws Excep
    tion
2.      {
3.          //XmlPullParserFactory pullPaser =
    XmlPullParserFactory.newInstance();
4.          ArrayList<Person> persons = null;
5.          Person person = null;
6.          // 创建一个xml解析的工厂
7.          XmlPullParserFactory factory = XmlPullParserFactory.newInstance
```

```
();
            // 获得xml解析类的引用
            XmlPullParser parser = factory.newPullParser();
            parser.setInput(xml, "UTF-8");
            // 获得事件的类型
            int eventType = parser.getEventType();
            while (eventType != XmlPullParser.END_DOCUMENT) {
                switch (eventType) {
                case XmlPullParser.START_DOCUMENT:
                    persons = new ArrayList<Person>();
                    break;
                case XmlPullParser.START_TAG:
                    if ("person".equals(parser.getName())) {
                        person = new Person();
                        // 取出属性值
                        int id = Integer.parseInt(parser.getAttributeValue(
0));
                        person.setId(id);
                    } else if ("name".equals(parser.getName())) {
                        String name = parser.nextText();// 获取该节点的内容
                        person.setName(name);
                    } else if ("age".equals(parser.getName())) {
                        int age = Integer.parseInt(parser.nextText());
                        person.setAge(age);
                    }
                    break;
                case XmlPullParser.END_TAG:
                    if ("person".equals(parser.getName())) {
                        persons.add(person);
                        person = null;
                    }
                    break;
                }
                eventType = parser.next();
            }
            return persons;
        }
```

使用Pull生成xml数据的流程:

使用Pull生成xml文件流程:

核心代码：

```
public static void save(List<Person> persons, OutputStream out) throws Exception {
        XmlSerializer serializer = Xml.newSerializer();
        serializer.setOutput(out, "UTF-8");
        serializer.startDocument("UTF-8", true);
        serializer.startTag(null, "persons");
        for (Person p : persons) {
            serializer.startTag(null, "person");
            serializer.attribute(null, "id", p.getId() + "");
            serializer.startTag(null, "name");
            serializer.text(p.getName());
            serializer.endTag(null, "name");
            serializer.startTag(null, "age");
            serializer.text(p.getAge() + "");
            serializer.endTag(null, "age");
```

```
            serializer.endTag(null, "person");
        }

        serializer.endTag(null, "persons");
        serializer.endDocument();
        out.flush();
        out.close();
    }
```

6.代码示例下载：

运行效果图：

代码下载：
XMLParseDemo.zip：http://www.runoob.com/try/download/XMLParseDemo.zip

---

本节小结：

> 本节介绍了Android中三种常用的XML解析方式，DOM，SAX和PULL，移动端我们建议用后面这
> 两种，而PULL用起来更加简单，这里就不多说了，代码是最好的老师~本节就到这里，下节我们
> 来学习Android为我们提供的扣脚Json解析方式！谢谢~