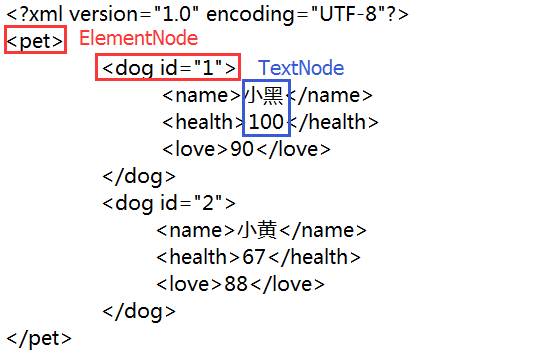
XML 中的节点分为ElementNode、TextNode两种，分别如下所示：

其中，**空白也属于TextNode.**



**DOM解析XML:**

DOM为Document Object Model的缩写，即文档对象模型，DOM将XML组织成一棵树，为其在逻辑上建立一个树模型，树的节点是一个个对象，通过存取这些对象就能够存取XML文档的内容。

1、获得解析器工厂对象：

DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();

2、获得解析器对象：

DocumentBuilder builder = factory.newDocumentBuilder();

3、获得解析XML文件得到的Document对象：

Document document = builder .parse(“d:\\pet.xml”);

这个Document对象代表了一个XML文档的树模型，所有对XML文档的操作，都与解析器无关，直接在这个Document对象上进行操作。

4、得到NodeList对象，Node对象代表了XML文档中的一个标签元素，NodeList对象则代表Node对象列表：

NodeList nodeList = document.getElementsByTagName(“dog”);

通过上述语句得到的是XML文档中所有<dog>标签对应的Node对象的列表，可以使用下述方法来得到列表中的每一个Node对象：

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

Node node = nodeList.item(i);

}

此时，保存在XML某个节点中的数据就被提取出来封装到这个Node中，使用下述方法来获取<dog>标签中的内容：

String message = node.getFirstChild().getNodeValue();

DOM基本对象：

DOM基本对象有5个：Document、Node、NodeList、Element、Attr.

**Document：整个XML文档**

**Node：DOM树中的节点**

**NodeList：Node列表**

**Element：标签**

**Attr ：标签中的属性**

**其中Element、Attr都继承了Node.**

**Document**代表**整个XML文档**，Document还包含了创建其他对象的方法，如：

createAttribute(String): 创建一个Attr对象，可以使用setAttributeNode()放置在某一个Element对象上面

createElement(String): 创建一个Element对象，代表XML文档中的一个标签

createTextNode(String): 创建一个Text对象，代表标签或属性中所包含的纯文本字符串

getElementsByTagName(String)：返回一个NodeList对象，包含了所有指定名字的Node

getDocumentElement()：返回代表这个DOM树根节点的Element对象，即代表XML文档根元素的对象

**Node**代表DOM树中的**一个节点**，是DOM结构中最基本的对象。在实际使用时很少使用到Node对象，而是使用Element, Attr, Text等Node对象的子对象来操作文档。Node对象为这些对象提供了一个抽象的、公共的根。Node所包含的主要方法有：

appendChild(Node)：为当前节点添加一个子节点，放置在所有节点的最后，如果子节点已经存在，将其删掉再添加进去

getFirstChild():如果该节点存在子节点，则返回第一个子节点

getLastChild():返回最后一个子节点

**getNextSibling():返回DOM树中该节点的下一个兄弟节点**

**getPreviousSibling():返回DOM树中该节点的前一个兄弟节点**

getNodeName() : 返回节点的名称

getNodeType() : 返回节点的类型

getNodeValue() : 返回节点的值

hasChildNodes():判断是否存在子节点

hasAttributes():判断该节点是否存在属性

getOwnerDocument():返回节点所处的Document对象

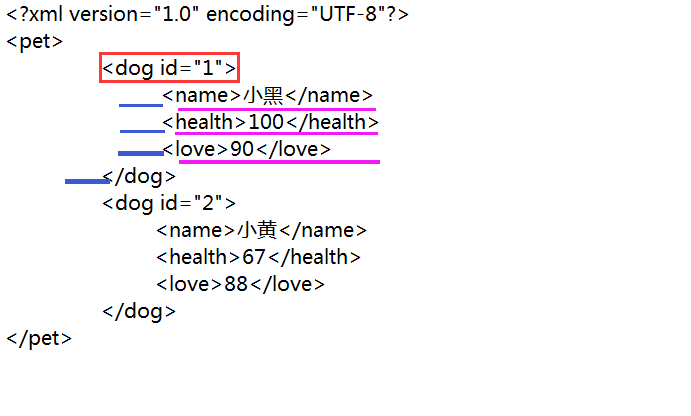
insertBefore(Node new, Node ref): 在给定的节点前再插入一个节点

removeChild(Node):删除指定的子节点

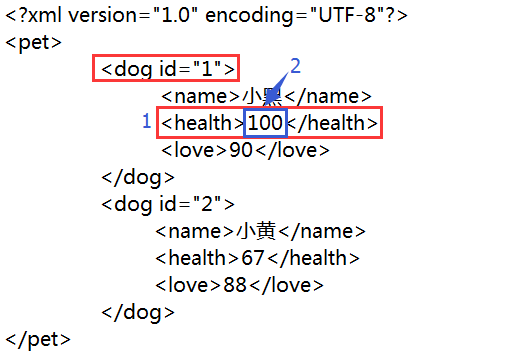
replaceChild(Node new, Node old): 用Node对象new替换old

对于下图中的节点<dog id=”1”>， 一共有7个子节点，其中4个空白，3个直接子节点。在解析的时候，不能忽略**空白**。

即：一个节点，若直接子节点个数为n, 则加上空白，其子节点个数为 2n+1



对于下图中的节点<dog id=”1”>，节点1为其子节点，节点2为孙子节点。



NodeList对象代表包含Node对象的列表，方法为：

**getLength() : 得到列表的长度**

**item(int) : 得到指定位置的Node对象**

**Element**代表XML文档中的**标签**，继承于Node, 是Node的最主要子元素。在标签中可以包含有属性Attr，因此，Element对象中有存取属性的方法：

getTagName():返回标签名字

getAttribute(String): 返回标签中给定属性名称的属性值

getAttributeNode(String):返回代表给定属性的Attr对象

**Attr** 代表**标签中的属性**，继承了Node 。Attr是包含在Element中的，Attr是包含它的Element的一部分，不作为DOM树中单独的一个节点出现，所以Node中的getParentNode()， getPreviousSibling(), getNextSibliing()都返回null。

DOM优点：

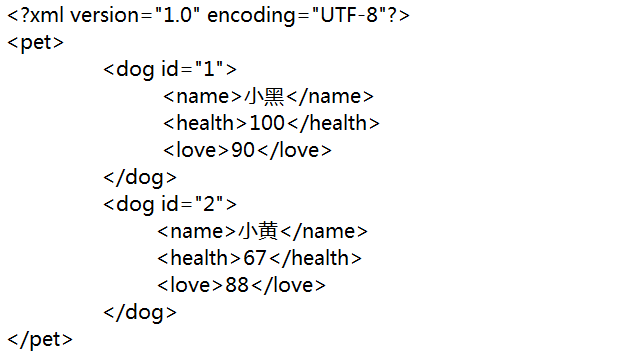
整个文档树(Document对象)存在于内存中，支持实时删除，修改，排列

DOM缺点：

将整个文档调入内存（包括无用的节点），浪费时间和空间

适用于解析了文档还需多次访问这些数据的场合。

被解析的XML文档：





NodeList nodeList = document.getElementsByTagName("dog");

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

Node pet = nodeList.item(i);

for(Node node = pet.getFirstChild(); node != null; node = node.getNextSibling()){ //内层循环一共会调用 (2n+1) 次，n为节点 个数

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

String nodeName = node.getNodeName();

String nodeValue = node.getFirstChild().getNodeValue();

System.out.println( nodeName + " : " + value + "\t");

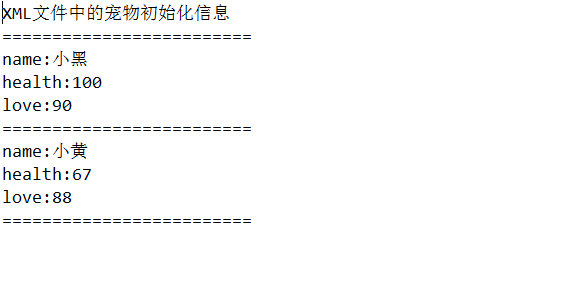
}

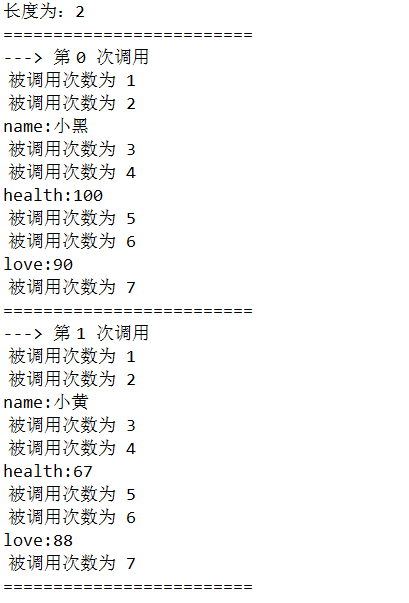
}

}

**其中空白节点(TextNode)不会进入if(node.getNodeType() == Node.ELEMENT\_NODE)块**

结果：

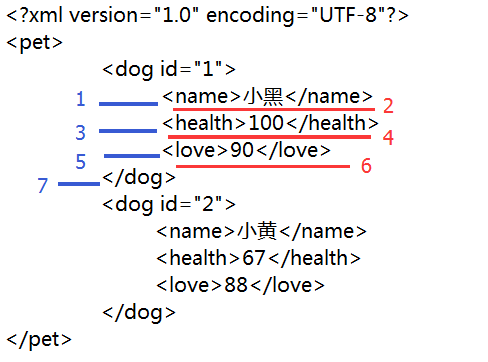




**SAX解析XML:**

基于事件的简单XML API ( Simple API for XML )称为SAX，SAX基于事件驱动。

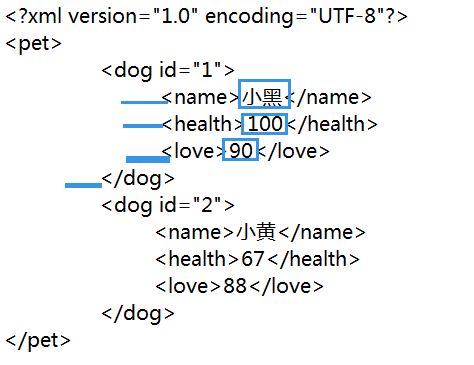
SAX在解析XML时可以触发一系列的事件，当发现给定tag时，会激活一个回调方法，开发者需要做的是重写回调方法即可。SAX对内存要求低，因为它让开发人员自己来决定要处理的tag, 特别是当开发人员只需要处理XML文档中的部分数据时。



void **startDocument()**: 开始解析XML时调用，即当读到**<?xml.....>**时调用

void **startElement**(String uri, String **localName**, String qName, Attributes attribute): 读到**开始标签**时调用，即读到<pet><dog>等时调用。**localName**是节点名称，有可能为空，**通常使用第3个参数qName**，attribute是这个节点的属性。当读到<dog id="1">这个节点时，只有一个属性id，可以通过attribute.getValue(0)得到。

void **endElement**(String uri, String localName, String qName): 读到**结束标签**时调用，即读到</dog>、</pet>等标签时调用



**characters**(char[] ch, int start, int length)：读到上述**TextNode**时被调用，注意SAX解析器也将上述中的空白视为一个TextNode。

即：**总TextNode节点个数为 = 2n+1**;

SAXParser类中常用方法：

getParser()：返回解析器

getProperty()：返回指定属性的值

setProperty()：设定指定属性的值

getXMLReader()：返回XMLReader对象

isNamespaceAware()：指明是否配置解析器以了解命名空间

isValidating()：指明是否配置解析器以验证XML文档

SAX解析类：

SAX解析类需要继承DefaultHandler类，并且在解析过程中需要使用一个String对象记录当前正在解析的节点的名称，避免在characters(char[] ch, int start, int length)方法中重复进入if块。

SAXParserFactory factory = SAXParserFactory.newInstance();

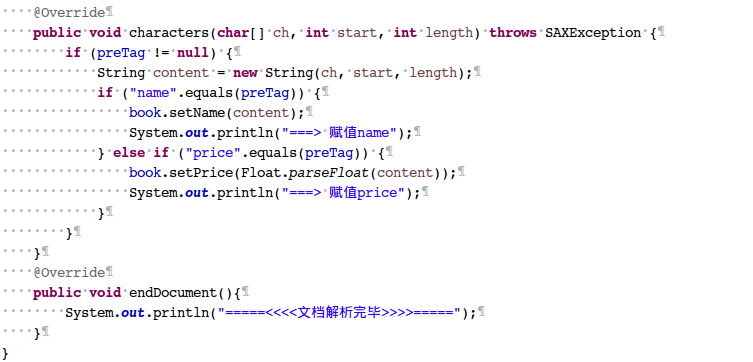
SAXParser parser = factory.newSAXParser();

parser.parse(fileURL, this);

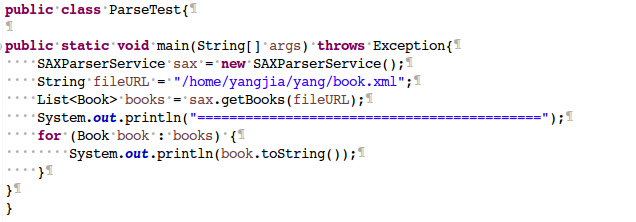








测试类：





**PULL解析XML：**

Android系统内置了Pull解析器来解析XML文档，SharedPreference就是使用内置pull解析器解析配置文件的，使用方式和SAX相似，都是采用**事件驱动**。PULL的编码比较简单，只需要处理开始与结束事件(下述4种事件)，通常使用**switch语句**，根据事件的不同类型，匹配不同的处理方式，一共有5种事件：

**START\_DOCUMENT** ：开始解析XML文档时产生

**START\_TAG** ：读取到开始标签产生

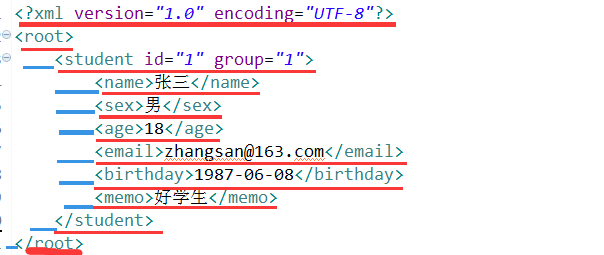
**TEXT** ：读取到**文本**产生(在switch结构中没有此case)

**END\_TAG** : 读取到结束标签产生

**END\_DOCUMENT** ：结束处理XML文档时产生

pull提供了开始标签和结束标签，当读取到**开始标签**时，调用parser.nextText()取出其后被该开始标签和其结束标签包裹的数据。

XML文件：



注意：

文档根元素**开始标签<root>**前面没有TextNode，**结束标签</root>**前面则有

Pull解析类：

XmlPullParserFactory factory = XmlPullParserFactory.newInstance();

XmlPullParser parser = factory.newPullParser();

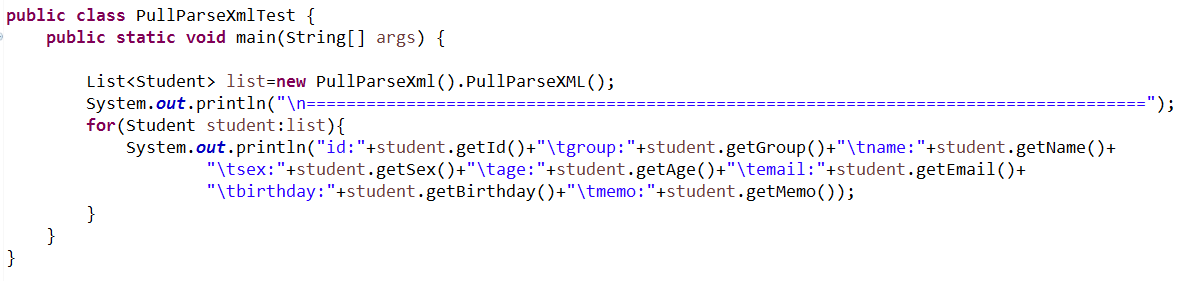
parser.setInput(Thread.currentThread().getContextClassLoader().getResourceAsStream(“studend.xml”), “UTF-8”);



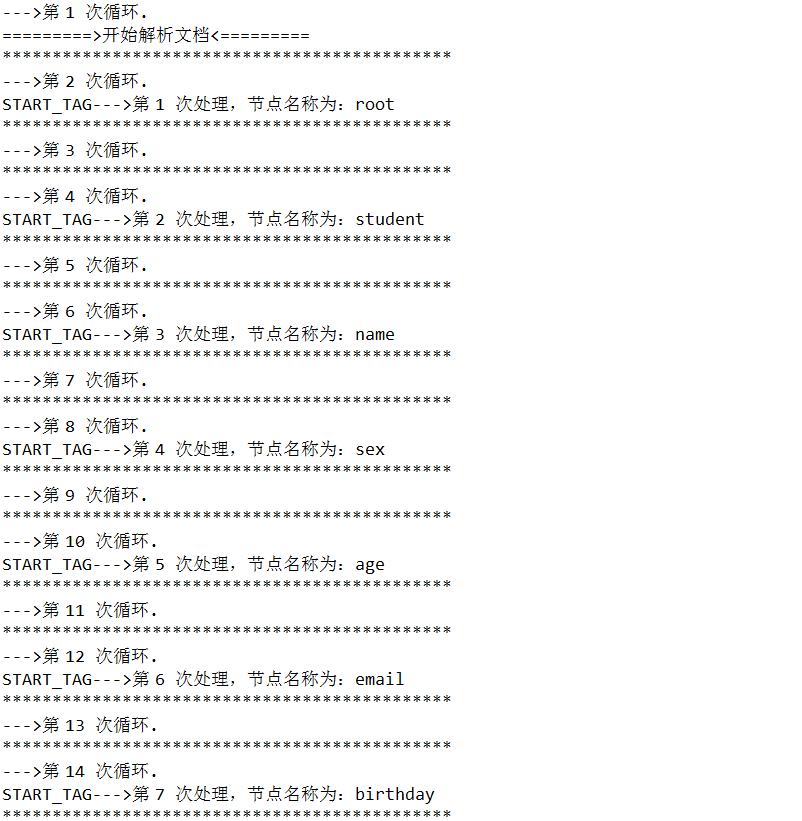




测试类：



结果：

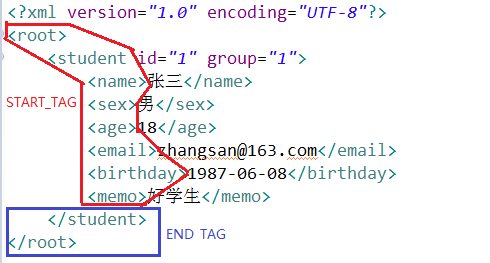




从结果可以看出：

遇到**所有的开始标签**时都会产生**START\_TAG**事件；

只有遇到下列蓝色框内的嵌套其他节点的节点的**结束标签**时才会产生**END\_TAG**事件。



SAX与PULL区别：

1. 继承类不同

SAX: 需要继承DefaultHandler

PULL：不需要继承类

2、处理Node的地方不同

SAX: 在不同的回调方法中进行处理

PULL：在一个switch结构中进行处理，该switch结构放在循环中

3、获取节点属性的方法不同

SAX：attributes.getValue(int index)

PULL: parser.getAttributeValue(int index)

4、TextNode处理方式不同

SAX: 是在characters(char[], int, int)方法中专门处理

PULL: 是在START\_TAG块中处理，没有TEXT这个case块

5、开始和结束ElementNode的触发次数不同

SAX: 开始ElementNode和结束ElementNode的触发次数是相同的

PULL: 所有的开始ElementNode都会触发START\_TAG，但只有复合节点的结束 ElementNode才会触发END\_TAG.

DOM与PULL对比：

1. 获取非空TextNode值的方式不同

DOM: node.getFirstChild().getNodeValue()

PULL: parser.nextText()

PULL可以在解析中途根据判断主动停止解析，DOM和SAX则不能。

如果在一个XML文档中只需要前面一部分数据，但是**使用SAX或DOM会对整个文档进行解析**，尽管XML文档中后面的大部分数据都不需要，因此这样实际上就浪费了处理资源。使用PULL方式正合适。

Pull解析器和SAX解析器虽有区别但也有相似性。他们的区别为：SAX解析器的工作方式是自动将事件推入注册的事件处理器进行处理，SAX不能主动结束事件的处理；而Pull解析器的工作方式为允许你的应用程序代码主动从解析器中获取事件，正因为是主动获取事件，因此PULL可以在满足了需要的条件后不再获取事件，主动结束解析。这是他们主要的区别。

而他们的相似性在运行方式上，Pull解析器也提供了类似SAX的事件，开始文档START\_DOCUMENT和结束文档END\_DOCUMENT，开始元素START\_TAG和结束元素END\_TAG，遇到元素内容TEXT等，但需要调用next() 方法提取它们（主动提取事件）。

Android系统中和Pull方式相关的包为org.xmlpull.v1，在这个包中提供了Pull解析器的工厂类XmlPullParserFactory和Pull解析器XmlPullParser，XmlPullParserFactory实例调用newPullParser方法创建XmlPullParser解析器实例，接着XmlPullParser实例就可以调用getEventType()和next()等方法依次主动提取事件，并根据提取的事件类型进行相应的逻辑处理。

//创建XmlPullParser,有两种方式

//**方式一:使用工厂类XmlPullParserFactory**

XmlPullParserFactory pullFactory = XmlPullParserFactory.newInstance();

XmlPullParser xmlPullParser = pullFactory.newPullParser();

//**方式二:使用Android提供的实用工具类android.util.Xml**

XmlPullParser xmlPullParser = **Xml.newPullParser()**;

XmlPullParser.setInput(InputStream inputStream, String inputEncoding) 解析对应的输入文件

XmlPullParser.getEventType() 得到元素类型

PULL方式比较简单，而且可以根据判断停止解析（DOM和SAX则不能）。

步骤：

**DOM解析：**

1.得到解析器工厂对象

2.得到解析器对象

3.得到Document对象

4.得到指定标签的NodeList对象

5.使用for循环开始解析

DocumentBuilderFactory factory = DocumentBuilderFactory**.newInstance()**;

**DocumentBuilder** builder = factory.new**DocumentBuilder**();

Document document = builder.parse("d:\\pet.xml");

NodeList nodeList = document.getElementsByTagName("dog");

**SAX解析：**

1.得到解析器工厂对象

2.得到解析器对象

3.调用解析器对象的parse()方法解析

SAXParserFactory factory = SAXParserFactory**.newInstance()**;

**SAXParser** parser = factory.new**SAXParser**();

parser.parse(fileURL, this);

**PULL解析：**

1.得到解析器工厂对象

2.得到解析器对象

3.设置输入流XML文件解析

XmlPullParserFactory factory = XmlPullParserFactory**.newInstance()**;

Xml**PullParser** parser = factory.new**PullParser**();

parser.setInput(Thread.currentThread().getContextClassLoader().getResourceAsStream("student.xml"), "UTF-8");