# 1. XML概要

XML文件是纯文本文件，用来描述数据，常用做项目的配置文件。XML是可扩展标记语言（Extensible Markup Language），XML文件使用xml作为文件扩展名。下面介绍XML中的元素。

## 1.1 XML文档声明

所有XML文件内容的第一行是XML的文档声明，一般文档声明格式如下（一定是这种格式，并且version和encoding的顺序不能颠倒）：

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?> |

其中version="1.0"表示XML的版本号，目前一般就使用1.0的版本；encoding="utf-8"表示该文件以UTF-8方式编码，这是XML最常用的编码。同时要注意保存文件时，应确保保存文件的编码和encoding中设置的编码一致，一般IDE会自动帮我们保存为相一致的编码。

注意：文档声明必须从文档的第一行、第一列开始！

## 1.2 标签和属性

XML通过标签和属性存储数据。XML中的标签和属性是自己定义的。比如现在用student.xml来保存学生的数据，内容如下：

|  |
| --- |
| <?xml version="1.0" encoding="utf-8" ?>  <students>  <student age="12">张三</student>  <student age="13">李四</student>  </students> |

这个xml中就存储了两个学生，保存了每个学生的姓名和年龄。students和student就是标签名，标签用尖括号括起来，每个标签都有开始标签和结束标签。结束标签就是在标签名前面加斜杠。

标签里面可以直接有文本内容，也可以再嵌套标签，这样就形成了有层次的数据结构。如果一个标签没有内容，则是空标签，比如：<student></student>。对于空标签，可简写成：<student />。

XML是用来存储数据的，能够解析空白和换行符，例如：

|  |
| --- |
| <student age="12">张三</student> |

和：

|  |
| --- |
| <student age="12">  张三  </student> |

是不同的，一个表示文本内容就是“张三”，而另一个表示“换行符 + 空格 + 张三 + 换行符”在内的所有内容。下面我们用Java解析这两种格式的数据时，得到的结果都是不同的。因此建议不要为了使XML“好看”，而在标签文本中加上这些控制字符和空格。当然，不同标签之间进行换行和缩进这些格式化操作是不存在这些问题的。

XML文件是区分的，并且标签名不能以数字开头。需要注意的是，一个XML文档中只能有一个根标签。在例子中，根标签是students，不能再写一个根标签。

在上述例子中，写在标签尖括号中的age就是一个属性。标签中可定义多个属性，而属性值必须放在引号里面。单引号和双引号都可以，但是不能单双引号混用。

## 1.3 注释、转义字符和CDATA块

XML中可以使用注释，语法是：<!-- 注释内容 -->。

XML中，像大于号>、小于号<这样的字符已经被标签使用了，如果把这些字符作为数据存储的话，就需要用到转义字符，否则会产生歧义。

|  |  |
| --- | --- |
| 需转义的字符 | 对应的转义字符 |
| < （小于号） | &lt; |
| > （大于号） | &gt; |
| " （引号） | &quot; |
| & | &amp; |
| 空格 | &nbsp; |

若一段数据中，有大量的特殊字符要转义，那么可以使用CDATA块（Character Data），将内容包含在CDATA块中。对于CDATA块中的内容，XML只会将它解析成普通的文本。CDATA块的语法是：

<![CDATA[内容]]，比如：

|  |
| --- |
| *<?***xml version="1.0" encoding="utf-8"** *?>* <**students**>  <**student age="12"**>张三</**student**>  <**student age="13"**>李四</**student**>  <![CDATA[<table>我只是文本</table>]]> </**students**> |

## 1.4 处理指令

处理指令是用来指示XML解析引擎如何去解析XML文档内容的。处理指令以“<?>”开头，以“?>”结尾，XML声明语句就是最常见的处理指令。

除此之外，XML还能还有样式，也是通过处理指令完成的。例如下面的示例。

（1）在test.xml中引入test.css样式文件，其中句用到了处理指令：

|  |
| --- |
| *<?***xml version="1.0" encoding="utf-8"***?> <?***xml-stylesheet type="text/css" href="test.css"***?>* <**students**>  <**student**>张三</**student**>  <**student**>李四</**student**> </**students**> |

（2）编写的test.css样式文件为：

|  |
| --- |
| **student** {  **color**: **red**; } |

这时我们在浏览器中打开test.xml文件，就能显示应用的样式了。详细的就不再介绍了，实际中并不常用。

# 2. 解析XML

XML中存储了数据，那么在程序中要想使用这些数据，必须用程序对XML进行解析来获得数据。用程序读取XML文件内容并获得数据就叫做XML解析。

Java中，常用的XML解析方式是SAX解析和DOM解析。SAX我们使用官方提供的API，DOM我们使用DOM4J第三方库。

## 2.1 SAX解析

SAX解析工具有关的包在org.xml.sax和javax.xml.parsers中。解析的步骤如下：

（1）首先通过SAXParserFactory工厂类创建一个SAXParser解析器对象。

（2）调用解析器的parse方法进行解析。方法原型是：

public void parse(String uri, DefaultHandler dh)。

参数1：表示XML文件的地址，可以是网络资源，也可以是本地资源地址。此函数的其他重载中可以传递File类型参数和InputStream类型参数。

参数2：要一个DefaultHandler的子类对象。SAX是基于事件处理XML的，DefaultHandler中定义了SAX的所有事件方法，比如开始读到文件的方法、读到一个标签的方法、读到一个文本的方法等。我们只需要在子类中实现具体的事件处理方法即可。

例子：简单使用下SAX。

（1）主类

|  |
| --- |
| **package** com.zhang.test;  **import** javax.xml.parsers.SAXParser; **import** javax.xml.parsers.SAXParserFactory;  **public class** Demo {  **public static void** main(String[] args) **throws** Exception {  *// SAXParserFactory是单例，通过静态方法拿到工厂对象再创建SAXParser解析器* SAXParser parser = SAXParserFactory.*newInstance*().newSAXParser();  *// 读取示例的student.xml文件。创建MyXMLHandler对象。MyXMLHandler类下面创建。* parser.parse(**"src/student.xml"**, **new** MyXMLHandler());  } } |

（2）MyXMLHandler类继承DefaultHandler类：

|  |
| --- |
| **package** com.zhang.test;  **import** org.xml.sax.Attributes; **import** org.xml.sax.SAXException; **import** org.xml.sax.helpers.DefaultHandler;  **public class** MyXMLHandler **extends** DefaultHandler {  *// 接收文档开始的通知* **public void** startDocument() **throws** SAXException {  System.***out***.println(**"事件————开始读取文档"**);  }   *// 接收元素（即标签）开始的通知* **public void** startElement(String uri, String localName, String qName, Attributes attributes) **throws** SAXException {  System.***out***.println(**"事件————读到标签："** + qName);  }   *//接收元素中字符数据的通知* **public void** characters(**char**[] ch, **int** start, **int** length) **throws** SAXException {  System.***out***.println(**"事件————读到文本："** + **new** String(ch, start, length));  }   *// 接收元素（标签）结束的通知* **public void** endElement(String uri, String localName, String qName) **throws** SAXException {  System.***out***.println(**"事件————读完标签："** + qName);  }   *// 接收文档结束的通知* **public void** endDocument() **throws** SAXException {  System.***out***.println(**"事件————读完文档"**);  } } |

以上重写的就是主要事件。其中，startDocument()和endDocument()是文档开始和结束的事件，一般情况下不需要有具体处理，可不重写。除非需要在读取文档前需要初始化资源等操作。

重要的数据处理集中在以下三个方法中：

（1）public void startElement(String uri, String localName, String qName, Attributes attributes)：每读取到一个开始标签就会调用此方法。uri是名称空间，localName是本地名称。这两个学完名称空间之后就会知道。现在只需要用到qName，这就是该标签的名称。参数attributes中就存放了该标签的所有的属性和值。该节点通常表示读到了某条数据。

（2）public void endElement(String uri, String localName, String qName)：每读到一个结束标签就会调用此方法。参数的含义和方法1中相同。

（3）public void characters(char[] ch, int start, int length)：每读到文本数据时调用此方法。ch参数表示字符数组，start和length表示开始位置和长度。因此用String data = new String(ch, start, length)；来得到文本数据。

上述代码的运行结果是：

|  |
| --- |
| 事件————开始读取文档  事件————读到标签：students  事件————读到文本：    事件————读到标签：student  事件————读到文本：张三  事件————读完标签：student  事件————读到文本：    事件————读到标签：student  事件————读到文本：李四  事件————读完标签：student  事件————读到文本：    事件————读到文本：<table>我只是文本</table>  事件————读到文本：  事件————读完标签：students  事件————读完文档 |

解释：上述的运行结果为什么出现很多空白的文本呢？因为为了方便阅读，XML内容中，标签间含有大量空白。如果把内容改成如下紧缩的格式就不会有空白文本，并且能节省文件存储空间，但不利于阅读：

|  |
| --- |
| *<?***xml version="1.0" encoding="utf-8"** *?>* <**students**><**student age="12"**>张三</**student**><**student age="13"**>李四</**student**><![CDATA[<table>我只是文本</table>]]></**students**> |

### 2.1.1 SAX解析案例

在此案例中，还是以阅读性高的XML文件解析为例（即格式化后的XML文件），将XML中描述的学生信息存储到程序的List<Student>对象中，顺便讲如何只读取标签里面的文本内容，而忽略标签间的空白。

处理的方法：实际上解析器类只需要重写startElement、character和endElement方法。

（1）当startElement读到students标签时，说明将要读到各个学生的信息了，这时可创建List<Student>对象了（为了能在下面的其他方法中使用此List对象，应将此变量提升为类的成员变量）。当读到student标签时，说明有了新的一个学生数据，说明可创建一个Student对象，并且读取此标签中的年龄属性，给该学生对象年龄赋值。为了让该对象能在下面的方法中使用，也应该由类维护一个Student成员变量，暂存每次的Student数据。

（2）当进入到character方法时，有可能读到了该学生的姓名，也可能读到空白。如何解决呢？办法是在类中维护一个成员变量暂存当前读到的标签名称。即在startElement方法中给此成员赋值。那么此方法中，可判断当前标签是否是student标签，若是student标签，说明读到的是标签中的姓名文本，此时拿到类维护的Student成员变量并赋值，若不是，则不是姓名文本，不进行处理。

（3）当进入endElement方法时，若此标签是student，那么说明该Student数据读取完毕，可将此学生对象加入到List集合中。注意此时还要将暂存标签名称的变量设置为null，否则执行完此结束标签，就会读到空白换行字符，此时标签名称还为student，那么就会执行character方法，将学生姓名设置为换行和空白。至此完毕。

示例代码如下：

（1）Student类：

|  |
| --- |
| **package** com.zhang.test;  **public class** Student {  **private** String **name**;  **private int age**;   **public** String getName() {  **return name**;  }   **public void** setName(String name) {  **this**.**name** = name;  }   **public int** getAge() {  **return age**;  }   **public void** setAge(**int** age) {  **this**.**age** = age;  }   @Override  **public** String toString() {  **return "姓名："** + **this**.getName() + **"，年龄："** + **this**.getAge();  } } |

（2）解析器类：

|  |
| --- |
| **package** com.zhang.test;  **import** org.xml.sax.Attributes; **import** org.xml.sax.SAXException; **import** org.xml.sax.helpers.DefaultHandler;  **import** java.util.ArrayList; **import** java.util.List;  **public class** MyXMLHandler **extends** DefaultHandler {  *// 需要维护的成员：List对象、Student对象、当前标签名称* **private** List<Student> **stuList** = **null**;  **private** Student **student** = **null**;  **private** String **currentQName** = **null**;   **public void** startElement(String uri, String localName, String qName, Attributes attributes) **throws** SAXException {  *// 先给currentQName赋值* **currentQName** = qName;  *// 若是students标签，则创建List集合，若是student标签，则创建Student对象* **if**(**"students"**.equals(qName)) {  **stuList** = **new** ArrayList<>();  }  **if**(**"student"**.equals(qName)) {  **student** = **new** Student();  *// 并给年龄属性赋值* String age = attributes.getValue(**"age"**); *// 通过Attribute的getValue(String qName)方法得到某个属性的值。由于是文本，拿到的都是String类型* **student**.setAge(Integer.*parseInt*(age)); *// 先转换成int类型再赋值* }  }   **public void** characters(**char**[] ch, **int** start, **int** length) **throws** SAXException {  *// 若当前标签是student，就给姓名赋值* **if**(**"student"**.equals(**currentQName**)) {  **student**.setName(**new** String(ch, start, length));  }  }   *// 接收元素（标签）结束的通知* **public void** endElement(String uri, String localName, String qName) **throws** SAXException {  **currentQName** = **null**;  **if**(**"student"**.equals(qName)) {  **stuList**.add(**student**);  }  }   *// 此方法返回集合数据* **public** List<Student> getStuData() {  **return stuList**;  } } |

此类中使用equals()方法时，把字符串常量写在左边，这是防止字符串变量为null的情况。若变量写在左边，调用变量.equals()方法时，可能会出现空指针异常。而且例子中也有主动设置字符串变量为null的情况，这就更为重要。这是编程的技巧。也推荐将常量写在左边。

（3）Demo类：

|  |
| --- |
| **package** com.zhang.test;  **import** javax.xml.parsers.SAXParser; **import** javax.xml.parsers.SAXParserFactory; **import** java.util.List;  **public class** Demo {  **public static void** main(String[] args) **throws** Exception {  SAXParser parser = SAXParserFactory.*newInstance*().newSAXParser();  *// 创建MyXMLHandler，不要直接用parser.parse("src/student.xml", new MyXMLHandler());  // 因为等会还要调用MyXMLHandler中自定义方法getStuData()获得处理好的数据。  // 如果每次都是new MyXMLHandler来创建对象，根本拿不到数据，因为new创建了一个新对象，里面数据是独立的。* MyXMLHandler mh = **new** MyXMLHandler();  parser.parse(**"src/student.xml"**, mh);  *// 调用自定义的getStuData方法获得数据* List<Student> students = mh.getStuData(); *// 一定先使用上面的parse才能调用此方法* System.***out***.println(students);  } } |

## 2.2 DOM

DOM解析的原理是：XML解析器先把整个XML文档内容加载进内存，然后在内存中构建一颗Document对象树。因为XML文档就是树形结构，即根节点下有很多个节点，通过节点访问或者操作XML文档中的内容。这个Document树对象就是文档对象模型DOM（Document Object Model）。

常用DOM4J工具包来进行DOM解析。需要在项目中使用第三方jar包。关于如何在Java SE项目中导入jar包，请见《附录2：在Java SE项目中导入JAR包——Eclipse与IDEA》。

可在网址https://github.com/dom4j/dom4j/releases上下载DOM4J的官方资料，引入其中的JAR包：dom4j-2.0.0.jar。至此，我们可进行编码，来获得指定XML文件的DOM对象。代码如下：

|  |
| --- |
| **package** com.zhang.test;  **import** org.dom4j.Document; **import** org.dom4j.DocumentException; **import** org.dom4j.io.SAXReader;  **import** java.io.File;  **public class** Demo {  **public static void** main(String[] args) **throws** DocumentException {  *// 创建SAXReader解析器对象* SAXReader reader = **new** SAXReader();  *// 读取XML文档，并返回DOM对象* Document document = reader.read(**new** File(**"src/student.xml"**));  System.***out***.println(document);  } } |

### 2.2.1 DOM4J解析XML

获得了DOM对象后，可通过DOM解析数据。主要可通过DOM获得Element对象，Element对象就是一个标签对象，进而通过Element对象获得标签的Attribute属性对象和文本信息。主要用到的方法如下：

（1）调用Document对象的getRootElement()可获得DOM树的根节点对象，返回Element对象。

（2）Element对象提供了以下主要方法：

Element element(String name)：获得此节点（标签）下名称为name的第一个子标签对象。

List<Element> elements()：获得此节点下所有的子标签。

List<Element> elements(String name)：获得此节点下所有的指定名称的子标签。

List<Attribute> attributes()：获得该节点所有属性的集合对象。

Attribute attribute(String name)：获得指定名称的属性对象。此方法的重载还能传递int类型参数，即获得第几个属性对象。

String attributeValue(String name)：可通过Element直接获得该节点中指定属性的属性值。

（3）Attribute对象提供了以下主要方法：

调用getName()方法得到属性名，调用getValue()方法可得到属性值。

（4）另外，Element对象提供了getText()方法得到标签中的本文内容。还可直接用Element提供的elementText(String name)得到当前标签的指定名称的子标签的文本内容。适用于只有该一个子标签。

案例：将上述student.xml用DOM解析，将学生数据存储在List<Student>集合中。

|  |
| --- |
| **package** com.zhang.test;  **import** org.dom4j.Document; **import** org.dom4j.DocumentException; **import** org.dom4j.Element; **import** org.dom4j.io.SAXReader;  **import** java.io.File; **import** java.util.ArrayList; **import** java.util.List;  **public class** Demo {  **public static void** main(String[] args) **throws** DocumentException {  SAXReader reader = **new** SAXReader();  Document dom = reader.read(**new** File(**"src/student.xml"**));  *// 集合和当前学生对象* List<Student> stuList = **null**;  Student student = **null**;  *// 获得根元素* Element root = dom.getRootElement();  **if**(**"students"**.equals(root.getName())) {  stuList = **new** ArrayList<>();  }  *// 获得根下所有节点* List<Element> es = root.elements();  **for**(Element e : es) {  student = **new** Student();  *// 获得标签中文本，并给Student赋值* student.setName(e.getText());  *// 获得标签中age属性，并给Student赋值* student.setAge(Integer.*parseInt*(e.attributeValue(**"age"**)));  *// 向集合添加元素* stuList.add(student);  }  *// 输出List* System.***out***.println(stuList);  } } |

### 2.2.2 DOM4J修改XML文档

可以通过DOM4J构建一个树对象，然后将此DOM对象转化成一个XML文件存储在硬盘上。常用的方法是：

（1）DocumentHelper.createDocument()：用此方法新建一个文档对象，即DOM，返回Document对象。

（2）Document和Element提供了addElement(String name)方法，用于在此节点上再创建一个Element节点，并返回此Element节点对象。

（3）Element对象提供了addAttribute(String name, String value)方法来为此Element加上属性，方法还是返回Element。

下面代码就是创建了DOM对象，通过IO流把DOM输出成硬盘上的XML文件。

|  |
| --- |
| **package** com.zhang.test;  **import** org.dom4j.\*; **import** org.dom4j.io.XMLWriter;  **import** java.io.\*;  **public class** Demo {  **public static void** main(String[] args) **throws** DocumentException, IOException {  *// 创建DOM对象* Document dom = DocumentHelper.*createDocument*();  *// 创建根节点* Element rootElem = dom.addElement(**"students"**);  *// 为根节点添加两个学生信息* **for**(**int** i = 0; i < 2; i++) {  *// 添加节点* Element stu = rootElem.addElement(**"student"**);  *// 设置文本* stu.setText(**"张三"** + i);  *// 添加属性* stu.addAttribute(**"age"**, 20 + i + **""**);  }  *// 创建文件输出流* OutputStream os = **new** FileOutputStream(**"src/stu.xml"**);  *// 用XMLWriter输出DOM* XMLWriter xmlWriter = **new** XMLWriter(os);  xmlWriter.write(dom);  *// 关闭资源* os.close();  xmlWriter.close();  } } |

上述用到了setText()方法设置标签的文本。很多的方法可以通过查看官方提供的API文档来了解。需要注意的是，导入包时，应该导入的是org.dom4j中相关的包，要注意分辨，特别是有相同的类名时。

上述代码生成的xml文档是标准的，具有XML的文件声明，但是文件内容是紧缩的。我们可通过OutputFormat对象来设置具有格式化的XML内容。代码如下：

|  |
| --- |
| **package** com.zhang.test;  **import** org.dom4j.\*; **import** org.dom4j.io.OutputFormat; **import** org.dom4j.io.XMLWriter;  **import** java.io.\*;  **public class** Demo {  **public static void** main(String[] args) **throws** DocumentException, IOException {  Document dom = DocumentHelper.*createDocument*();  Element rootElem = dom.addElement(**"students"**);  *// 添加学生信息* **for**(**int** i = 0; i < 2; i++) {  Element stu = rootElem.addElement(**"student"**);  stu.setText(**"张三"** + i);  stu.addAttribute(**"age"**, 20 + i + **""**);  }  OutputStream os = **new** FileOutputStream(**"src/stu.xml"**);  *// 创建OutputFormat对象。有两种方式。  // OutputFormat.createPrettyPrint()就是使用格式化的形式；  // OutputFormat.createCompactFormat()就是使用紧缩的形式。* OutputFormat format = OutputFormat.*createPrettyPrint*();  *// 可设置编码等* format.setEncoding(**"UTF-8"**);  *// 这时要将format传递进XMLWriter构造中* XMLWriter xmlWriter = **new** XMLWriter(os, format);  xmlWriter.write(dom);  *// 关闭资源* os.close();  xmlWriter.close();  } } |

那么既然这样，修改XML就很方便了。先获得要修改的XML文件的DOM对象，然后调用方法对DOM进行修改等操作，最后通过XMLWriter对象将DOM对象再写回文件。关注以下方法：

（1）Attribute对象提供了setValue(值)方法可更改属性值；

（2）Element和Attribute对象提供了detach()方法用于删除此元素或者属性，并返回。

## 2.3 SAX和DOM解析比较

（1）SAX解析是加载一点就处理一点，而DOM解析首先就要把整个文档内容构建成DOM树存放在内存中。这导致SAX对内存要求较低，适合读取数据量大的文档；而DOM解析占用内存较高（可能导致内存溢出），不适合读取内容较多的XML文件。

（2）DOM解析采用的是面向对象的方法，而SAX解析是基于事件的编程方法，开发方式有所不同，SAX可能稍微复杂一点。

（3）由于SAX是基于事件的，只能顺序读取，而DOM方式可随时读取任意位置内容。

（4）SAX只能用于解析XML，而采用DOM可以方便的增删改查，并且DOM能结合下面讲的XPath技术方便的找到元素节点。

# 3. XPath技术

XPath即为XML路径语言，它用于快速的找到指定的元素节点。对于层次较深的DOM树，使用XPath非常好用。是一种用来确定XML文档中某部分位置的语言。XPath基于XML的树状结构，提供在数据结构树中找寻节点的能力。

## 3.1 XPath语法

XPath的用法教程见所提供的资料：“XPath Tutorial.chm”。里面提供了中文的翻译。看其中的实例即能掌握XPath语法。



## 3.2 在DOM4J中使用XPath

DOM4J提供了对XPath的支持，想要用XPath，还要导入一个名叫jaxen的JAR包（此包用于实现XPath）。资料里提供了jaxen-1.1.6.jar包，只要再引入即可。

在DOM4J中使用XPath语法找到元素只需用Node接口的以下两个方法：

（1）public List<Node> selectNodes(String xpathExp)：根据xpath语法查询并返回多个节点对象。

（2）public Node selectSingleNode(String xpathExp)：根据xpath语法查询并返回一个节点对象。

实际上在DOM4J中，Document、Element、Attribute和Text都是接口，并且他们都继承自接口Node，因此上述方法返回的是Node对象。一般XPath语法查找到的都是标签，因此可将Node强制转换成Element对象再调用其中的方法。

例子：

|  |
| --- |
| **package** com.zhang.test;  **import** org.dom4j.Document; **import** org.dom4j.DocumentException; **import** org.dom4j.Element; **import** org.dom4j.Node; **import** org.dom4j.io.OutputFormat; **import** org.dom4j.io.SAXReader; **import** org.dom4j.io.XMLWriter;  **import** java.io.\*;  **public class** Demo {  **public static void** main(String[] args) **throws** DocumentException, IOException {  SAXReader reader = **new** SAXReader();  Document dom = reader.read(**new** File(**"src/student.xml"**));  *//找到年龄为13的李四所在的节点。* Node node = dom.selectSingleNode(**"//student[@age=13]"**);  *// 强转成Element进行操作* Element elem = (Element) node;  *// 删除此节点* elem.detach();  *// 然后将此DOM树写回此XML文件实现修改* XMLWriter xmlWriter = **new** XMLWriter(**new** FileOutputStream(**"src/student.xml"**), OutputFormat.*createPrettyPrint*());  xmlWriter.write(dom);  xmlWriter.close();  } } |

下面是XPath使用的一些例子：

|  |
| --- |
| **package** com.zhang.test;  **import** org.dom4j.Document; **import** org.dom4j.DocumentException; **import** org.dom4j.Node; **import** org.dom4j.io.SAXReader;  **import** java.io.\*; **import** java.util.List;  **public class** Demo {  **public static void** main(String[] args) **throws** DocumentException, IOException {  Document dom = **new** SAXReader().read(**new** File(**"src/student.xml"**));  String xpath = **""**;  *// 1. / 表示从根元素开始或者子元素开始（即一个层次结构）* xpath = **"/students"**;  xpath = **"/students/student"**;  *// 2. // 表示不区分层次结构，从任意位置选择到元素* xpath = **"//student"**;  *// 3. \* 表示通配符* xpath = **"/students/\*"**; *// students的子标签* xpath = **"/students//\*"**; *// 这表示students中所有的标签，不分层次  // 4. [] 表示带有条件* xpath = **"//student[@age]"**; *// 带有age属性的student节点* xpath = **"//student[2]"**; *// 找到第二个student标签* xpath = **"//student[last()]"**; *// 选择最后一个student标签  // 5. @ 表示选择属性节点* xpath = **"//@age"**; *// 找到所有age属性节点，可强转成Attribute对象* xpath = **"//student[not(@age)]"**; *// 选择不包含age属性的student标签* xpath = **"//student[@age='12']"**; *// 选择age属性值为12的student标签* xpath = **"//student[@age='12' and @gender='男']"**; *// 选择age属性值为12且gender值为男的student标签  // 6. text()表示选择标签中的文本内容* xpath = **"//student/text()"**; *// 选择student标签中的文本，可强转成Text对象* xpath = **"//student[text()='张三']"**; *// 选择文本内容为张三的student标签* List<Node> list = dom.selectNodes(xpath);  **for**(Node node : list) {  System.***out***.println(node.getText());  }  } } |

# 4. XML约束

前面已经讲过XML的语法，XML中可自定义标签，存储各种数据。但有时，我们希望对XML文件加上一些约束。即在某XML配置文件中，只能出现特定的标签，并且还要限制他们的顺序。例如：规定<student>标签必须出现在<students>标签中；<student>标签必须要有age属性等。

实现上述功能，需要用到XML约束。有以下方式来定义XML约束：

（1）DTD约束：语法和功能简单，学习成本低；

（2）Schema约束：语法相对复杂，功能也强大，学习成本高。越来越多的框架使Schema作为XML约束。

如果一个XML文件使用了某个约束，那么此XML文件应该遵循约束文件来编写，否则IDE中会给出违反约束的信息。

## 4.1 DTD约束

DTD的含义是文档类型定义（Document Type Definition），DTD约束可以直接写在需要被约束的XML文件中。例如：

|  |
| --- |
| *<?***xml version="1.0" encoding="utf-8"** *?> <!-- 在XML开头定义DTD约束 -->* **<!DOCTYPE student** [  <!ELEMENT student (name, age, gender)>  <!ELEMENT name (#PCDATA)>  <!ELEMENT age (#PCDATA)>  <!ELEMENT gender (#PCDATA)> ]***>*** *<!-- 以下是XML真正的内容 -->* <**student**>  <**name**>张三</**name**>  <**age**>12</**age**>  <**gender**>男</**gender**> </**student**> |

以上是对标签进行了约束。知识点如下：

（1）约束应该定义在以下内容中：

|  |
| --- |
| <!DOCTYPE 根元素 [  约束的内容……  ]> |

例子中约束了根元素是student。

（2）<!ELEMENT ...>用于约束标签，语法是：

|  |
| --- |
| <!ELEMENT 标签名 此标签中允许的内容M> |

M可以为：

* EMPTY：表示此标签中不能有文本内容（即空元素）；
* ANY：表示此标签中可包含任意内容，包括子标签。
* (#PCDATA)：表示此标签中内容是普通字符串，不能含有子标签；
* (子标签1，子标签2，...)：规定此标签中可以出现的子标签，这些子标签也必须在约束中用<!ELEMENT 标签名 M>来定义。这样定义好后，XML内容中必须括号中标签的顺序来书写，比如例子中，必须先写<name>，再写<age>，最后写<gender>。另外，还能对这些标签限定出现的次数（默认必须出现一次）：
  + +：至少出现一次；
  + \*：出现0次或者n次；
  + ?：出1现0次或1次。

（3）同样，<!ATTLIST >用来约束属性，语法是：

|  |
| --- |
| <!ATTLIST 元素名称 属性名称 属性值类型 默认值> |

属性值类型可以是：

* CDATA：普通字符串；
* ID：表示该属性值用于唯一标识此标签，因此此属性值必须在XML中是唯一的，并且不能以数字开头。
* (属性值1，属性值2，...)：表示属性值只能在给定的属性值中选用。

默认值可以是：

* #REQUIRED：表示此属性是必须的
* #IMPLIED：表示此属性不是必须的
* #FIXED value：表示此属性不是必须的，但是必须为value值。

（4）对于DTD中没有约束的东西，是不能出现的。

例子中的约束是直接写在XML中的，但是一般都采取约束和XML内容分离的方式，即将DTD约束单独保存成一个文件，扩展名为dtd，然后当某XML需要使用此约束时，再在XML中导入此约束。导入的语法是：

|  |
| --- |
| <!DOCTYPE 根元素 导入类型 DTD约束文件地址> |

导入的类型可以使用SYSTEM或者PUBLIC。当使用SYSTEM时，表示导入的是本地约束文件，此时地址可用相对地址。当使用PUBLIC时，表示导入外部约束文件，此时地址就写网络上的DTD文件。一般使用后者，因为一些框架的约束是组织提供的，我们只要导入外部约束即可，IDE会自动下载相关联的约束文件帮助我们编写XML配置文件。以后使用框架将会体验更深。

综合案例：

DTD约束文件：student.dtd

|  |
| --- |
| **<!DOCTYPE students** [  **<!ELEMENT students** (**student**) >  **<!ELEMENT student** (**name**, **age**?, **email**\*) >  **<!ATTLIST student id ID** >  **<!ELEMENT name** (#PCDATA) >  **<!ELEMENT age** (#PCDATA) >  **<!ELEMENT email** ANY >  ]***>*** |

## 4.2 Schema约束

Schema约束文件的扩展名是xsd，即XML Schemas Definition，XML结构定义。Schame约束文件本身也是XML文件。

例子：

（1）XSD文件book.xsd：

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"** *?>* <**xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" targetNamespace="http://www.demo.com" elementFormDefault="qualified"**>  <**xs:element name='bookshelf'**>  <**xs:complexType**>  <**xs:sequence maxOccurs='unbounded'**>  <**xs:element name='book'**>  <**xs:complexType**>  <**xs:sequence**>  <**xs:element name='name' type='xs:string'**/>  <**xs:element name='author' type='xs:string'**/>  <**xs:element name='price' type='xs:integer'**/>  </**xs:sequence**>  </**xs:complexType**>  </**xs:element**>  </**xs:sequence**>  </**xs:complexType**>  </**xs:element**> </**xs:schema**> |

（2）book.xml文件：

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"***?>* <**zhang:bookshelf xmlns:zhang="http://www.demo.com"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://www.demo.com book.xsd"**>  <**zhang:book**>  <**zhang:name**>JavaScript网页开发</**zhang:name**>  <**zhang:author**>Mr. NB</**zhang:author**>  <**zhang:price**>128</**zhang:price**>  </**zhang:book**> </**zhang:bookshelf**> |

XSD中使用了名称空间。XSD定义了名称空间是“http://www.demo.com”（即targetNamespace定义的），XSD本身也受到XSD的约束，引入了名称空间“http://www.w3.org/2001/XMLSchema”，这是W3C指定的。并且起了别名为xs方便使用（这是xmlns的作用，用来指明引用的名称空间和别名，xmlns即XML Namespace）。xmlns在根标签中使用，并且根标签也可用此别名。

可以看出XSD约束很强大，可约束类型为string、int等，但是语法较复杂，这里不必深究了。

XML文档引用了book.xsd，也首先用xmlns来指明了引用的名称空间就是上述的“http://www.demo.com”，并且起别名为zhang，那么就可以直接使用zhang这个名称了。当然，使用xmlns时，可直接引入名称空间而不写别名，这样就能在XML中直接写标签了。

xmlns还定义了一个xsi的别名，这个是引用默认的约束，主要用来约束XML的语法。这是W3C提供的。

最后，xsi:schemaLocation用于指明引入的XSD约束的地址。对于默认的约束不需要指明地址。对于book.xsd的引入，直接使用了相对地址book.xsd。一个XML中可以引入多个约束文件，所以每个约束的地址前面都指明了它的名称空间。比如：

|  |
| --- |
| **xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4\_0\_0.xsd"** |

若有多个约束，那么就是这样：

|  |
| --- |
| xsi:schemaLocation=**"名称空间1 XSD地址1 名称空间2 XSD地址2 ..."** |

多个约束可能约束同一个标签或者属性，这时该标签/属性到底是哪个约束的呢？这时就是名称空间在起作用了。比如：

|  |
| --- |
| <c:id c:age=**"12"**>张三</c:id> |

其中的id和age就是“c”命令空间对应的xsd进行约束的。有时在某个XML中，大部分使用的都是一个xsd约束，只有少部分会用到其他的名称空间约束，为了简便，常用的xsd约束可以不加名称空间，作为默认的名称空间，直接使用标签即可，其他名称空间的标签需要加上名称空间。

对于XML约束，能大概理解并看懂即可，以后书写XML按照约束来进行即可，尤其是以后学习Java EE的框架，需要使用大量的XML配置文件时。