# XML

## 什么是XML？

XML:extensiable markup language 被称作可扩展标记语言

**XML是可以描述很复杂的数据关系的**

**XML被设计为“什么都不做”，XML数据或XML文档只用于组织、存储数据，除此之外的数据生成、读取、传送、存取等等操作都与XML本身无关！**

### **为什么我们需要使用XML呢**？

* ①我们没有XML这种语言之前，我们使用的是String作为两个程序之间的通讯！现在问题就来了，如果我们传输的是带有关系型结构的数据
* **HTML语言本身就有缺陷：**
  + - 标记都是固定的，不能自定义。HTML语言中有什么标记就只能用什么标记
    - HTML标签本身就缺少含义（tr标签里面什么内容都能放进去，不规范！）
    - HTML没有实现真正的国际化
* XML 把数据从 HTML 分离: 通过 XML，数据能够存储在独立的 XML 文件中。这样你就可以专于使用 HTML 进行布局和显示，并确保修改底层数据不再需要对 HTML 进行任何的改变。通过使用几行 JavaScript，你就可以读取一个外部 XML 文件，然后更新 HTML 中的数据内容。
* 由于 XML 独立于硬件、软件以及应用程序，XML 使您的数据更可用，也更有用。
* 通过 XML，可以在不兼容的系统之间轻松地交换数据。
* XML 数据**以纯文本格式进行存储**，因此提供了一种独立于软件和硬件的数据存储方法。

XML文件就解决了以上的问题了，如果使用XML描述上述图片的关系，是非常简单的！

<?xml version="1.0" encoding="UTF-8" ?>  
 <中国>  
 <北京>  
 <海淀></海淀>  
 <丰台></丰台>  
 </北京>  
 <湖南>  
 <长沙></长沙>  
 <岳阳></岳阳>  
 </湖南>  
 <湖北>  
 <武汉></武汉>  
 <荆州></荆州>  
 </湖北>  
 </中国>

### XML的用途

* ①：配置文件(例子：Tomcat的web.xml,server.xml......)，XML能够非常清晰描述出程序之间的关系
* ②：程序间数据的传输，XML的格式是通用的，能够减少交换数据时的复杂性！
* ③：充当小型数据库，如果我们的数据有时候需要人工配置的，那么XML充当小型的数据库是个不错的选择，程序直接读取XML文件显然要比读取数据库要快呢！

## XML 语法规则

* **XML元素中的出现的空格和换行都会被当做元素内容进行处理**
* **所有 XML 元素都须有关闭标签**: <p>This is a paragraph</p>
* **XML 标签对大小写敏感**: <Message>这是错误的。</message>
* **XML 必须正确地嵌套:** <b><i>This text is bold and italic</b></i>错误
* **XML 的属性值须加引号**
* **XML 文档必须有根元素：**

<root>

<child>

<subchild>.....</subchild>

</child>

</root>

* **实体引用**

在 XML 中，一些字符拥有特殊的意义。

如果你把字符 "<" 放在 **XML 元素中**，会发生错误，**这是因为解析器会把它当作新元素的开始。**

这样会产生 XML 错误：

<message>if salary < 1000 then</message>

为了避免这个错误，请用**实体引用**来代替 "<" 字符：

<message>if salary &lt; 1000 then</message>

在 XML 中，有 5 个预定义的实体引用：

|  |  |  |
| --- | --- | --- |
| &lt; | < | 小于 |
| &gt; | > | 大于 |
| &amp; | & | 和号 |
| &apos; | ' | 单引号 |
| &quot; | " | 引号 |

**注释：**在 XML 中，只有字符 "<" 和 "&" 确实是非法的。大于号是合法的，但是用实体引用来代替它是一个好习惯。

* **XML 中的注释**

在 XML 中编写注释的语法与 HTML 的语法很相似：

<!-- This is a comment -->

* **在 XML 中，空格会被保留**

HTML 会把多个连续的空格字符裁减（合并）为一个：

HTML: Hello my name is David.

输出: Hello my name is David.

在 XML 中，文档中的空格不会被删节。

### 文档声明：

* **XML声明放在XML的第一行**
* **version----版本**
* **encoding--编码**
* **standalone--独立使用--默认是no。standalone表示该xml是不是独立的，如果是yes，则表示这个XML文档时独立的，不能引用外部的DTD规范文件；如果是no，则该XML文档不是独立的，表示可以引用外部的DTD规范文档。**
* **正确的文档声明格式，属性的位置不能改变！**

<?xml version="1.0" encoding="utf-8" standalone="no"?>

### 什么是 XML 元素？

**XML 元素**指的是从（且包括）开始标签直到（且包括）结束标签的部分。

元素可包含其他元素、文本或者两者的混合物。元素也可以拥有属性。

<bookstore>

<book category="CHILDREN">

<title>Harry Potter</title>

<author>J K. Rowling</author>

<year>2005</year>

<price>29.99</price>

</book>

<book category="WEB">

<title>Learning XML</title>

<author>Erik T. Ray</author>

<year>2003</year>

<price>39.95</price>

</book>

</bookstore>

在上例中，<bookstore> 和 <book> 都拥有**元素内容**，因为它们包含了其他元素。<author> 只有**文本内容**，因为它仅包含文本。

### 属性

在上例中，只有 <book> 元素拥有**属性** (category="CHILDREN")。Book的额外信息

<file type="gif">computer.gif</file>：文件类型与数据无关

**XML 元素 vs. 属性**

请看这些例子：

<person sex="female">

<firstname>Anna</firstname>

<lastname>Smith</lastname>

</person>

<person>

<sex>female</sex>

<firstname>Anna</firstname>

<lastname>Smith</lastname>

</person>

在第一个例子中，sex 是一个属性。在第二个例子中，sex 则是一个子元素。两个例子均可提供相同的信息。

没有什么规矩可以告诉我们什么时候该使用属性，而什么时候该使用子元素。我的经验是在 HTML 中，属性用起来很便利，但是在 XML 中，您应该尽量避免使用属性。**如果信息感觉起来很像数据，那么请使用子元素吧。**

### PCDATA

PCDATA 指的是**被解析的字符数据**（Parsed Character Data）。

XML 解析器通常会解析 XML 文档中所有的文本。

当某个 XML 元素被解析时，其标签之间的文本也会被解析：

<message>此文本也会被解析</message>

解析器之所以这么做是因为 XML 元素可包含其他元素，就像这个例子中，其中的 <name> 元素包含着另外的两个元素(first 和 last)：

<name><first>Bill</first><last>Gates</last></name>

而解析器会把它分解为像这样的子元素：

<name>

<first>Bill</first>

<last>Gates</last>

</name>

### CDATA

术语 CDATA 指的是不应由 XML 解析器进行解析的文本数据（Unparsed Character Data）。

在 XML 元素中，"<" 和 "&" 是非法的。

"<" 会产生错误，因为解析器会把该字符解释为新元素的开始。

"&" 也会产生错误，因为解析器会把该字符解释为字符实体的开始。

某些文本，**比如 JavaScript 代码，包含大量 "<" 或 "&" 字符。为了避免错误，可以将脚本代码定义为 CDATA。**

**在编写XML文件时，有些内容可能不想让解析引擎解析执行，而是当作原始内容处理。遇到此种情况，可以把这些内容放在CDATA区里，对于CDATA区域内的内容，XML解析程序不会处理，而是直接原封不动的输出**

语法：<![CDATA[ ...内容 ]]>

**CDATA 部分中的所有内容都会被解析器忽略。**

CDATA 部分由 "**<![CDATA[**" 开始，由 "**]]>**" 结束：

<script>

<![CDATA[

function matchwo(a,b)

{

if (a < b && a < 0) then

{

return 1;

}

else

{

return 0;

}

}

]]>

</script>

在上面的例子中，解析器会忽略 CDATA 部分中的所有内容。

**关于 CDATA 部分的注释：**

CDATA 部分不能包含字符串 "]]>"。也不允许嵌套的 CDATA 部分。

标记 CDATA 部分结尾的 "]]>" 不能包含空格或折行。

### 处理指令

**处理指令，简称PI （processing instruction）。处理指令用来指挥解析引擎如何解析XML文档内容。**

例如：

**在XML文档中可以使用xml-stylesheet指令，通知XML解析引擎，应用css文件显示xml文档内容。**

<?xml-stylesheet type="text/css" href="1.css"?>

* **XML代码：**

<?xml version="1.0" encoding="UTF-8" ?>  
 <?xml-stylesheet type="text/css" href="1.css"?>  
 **<china>**  
 **<guangzhou>**  
 广州  
 **</guangzhou>**  
 **<shenzhen>**  
 深圳  
 **</shenzhen>**  
 **</china>**

* **CSS代码：**  
   guangzhou{  
   font-size: 40px;  
   }
* 效果



**注释：**使用 CSS 格式化 XML 不是常用的方法，更不能代表 XML 文档样式化的未来。**W3C 推荐使用 XSLT。**

## 验证 XML 文档（DTD、schema模式）

### 验证

合法的 XML 文档是“形式良好”的 XML 文档，同样遵守**文档类型定义 (DTD) 的语法规则**：

<?xml version="1.0" ?>

<!DOCTYPE 书架 SYSTEM "book.dtd">

<书架>

<书>

<书名>Java就业培训教程</书名>

<作者>张孝祥</作者>

<售价>39.00元</售价>

</书>

<书>

<书名>JavaScript网页开发</书名>

<作者>张孝祥</作者>

<售价>28.00元</售价>

</书>

</书架>

**DOCTYPE 声明是对外部 DTD 文件的引用。**

**本质区别：schema本身是xml的，可以被XML解析器解析(这也是从DTD上发展schema的根本目的)**

**1.schema 是内容开放模型，可扩展，功能性强，而DTD可扩展性差。**

**2.shema 支持丰富的数据类型，而 DTD不支持元素的数据类型，对属性的类型定义也很有限。**

**3.schema 支持命名空间机制，而DTD不支持。**

**4.schema 可针对不同情况对整个XML 文档或文档局部进行验证；而 DTD缺乏这种灵活性。**

**5.schema 完全遵循XML规范，符合XML语法，可以和DOM结合使用，功能强大；而DTD 语法本身有自身的语法和要求，难以学习。**

### XML解析方式

**所有现代浏览器都内建了供读取和操作 XML 的 XML 解析器。**

**解析器把 XML 转换为 XML DOM 对象 - 可通过 JavaScript 操作的对象。**

应用程序不是直接对XML文档进行操作的，而是由XML解析器对XML文档进行分析，然后应用程序通过XML解析器所提供的**DOM接口或者SAX接口对**分析结果进行操作，从而间接地实现了对XML文档的访问！

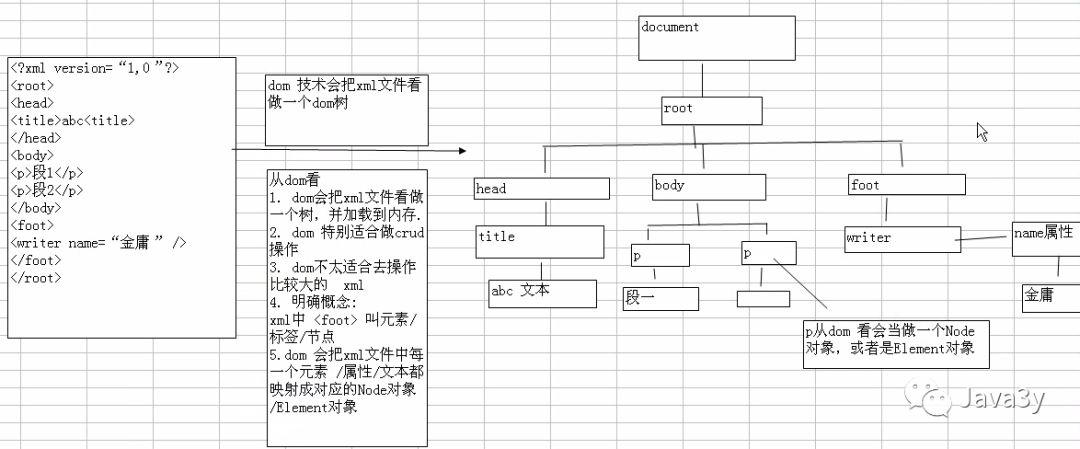
**为什么有3种开发包？**

* jaxp开发包是JDK自带的，不需要导入开发包。
* 由于sun公司的jaxp不够完善，于是就被研发了Jdom。XML解析如果使用Jdom，需要导入开发包
* dom4j是由于Jdom的开发人员出现了分歧，dom4j由Jdom的一批开发人员所研发。XML解析如果使用Jdom，需要导入开发包【现在用dom4j是最多的！】

## jaxp

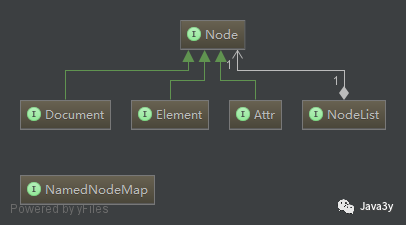
### DOM解析操作

DOM解析会把XML文档加载到内存中，生成DOM树的元素都是以对象的形式存在的！我们操作这些对象就能够操作XML文档了！很多父子关系

****

在DOM解析中有几个**核心的操作接口**：

* **Document【代表整个XML文档，通过Document节点可以访问XML文件中所有的元素内容！】**
* **Node【Node节点几乎在XML操作接口中几乎相当于普通Java类的Object，很多核心接口都实现了它，在下面的关系图可以看出！】**
* **NodeList【代表着一个节点的集合，通常是一个节点中子节点的集合！】**
* NameNodeMap【表示一组节点和其唯一名称对应的一一对应关系，主要用于属性节点的表示（书上说是核心的操作接口，但我好像没用到！呃呃呃，等我用到了，我再来填坑！）】

****

**好的，不跟你们多bb，我们来使用一下Dom的方式解析XML文档吧！**

* **XML文档代码**

<?xml version="1.0" encoding="UTF-8" ?>  
 <china>  
 <guangzhou >广州</guangzhou>  
 <shenzhen>深圳</shenzhen>  
 <beijing>北京</beijing>  
 <shanghai>上海</shanghai>  
 </china>

根据XML解析的流程图，**我们先要获取到解析器对象！**

public class DomParse {  
 public static void main(String[] args) throws ParserConfigurationException, IOException, SAXException {  
 //API规范：需要用一个工厂来造解析器对象，于是我先造了一个工厂！  
 DocumentBuilderFactory documentBuilderFactory = **DocumentBuilderFactory.**newInstance()  
 //获取解析器对象  
 DocumentBuilder documentBuilder = documentBuilderFactory.**newDocumentBuilder();**  
 //获取到解析XML文档的流对象  
 InputStream inputStream = DomParse.class.getClassLoader()**.getResourceAsStream**("city.xml")；  
 //解析XML文档，得到了代表XML文档的Document对象！  
 Document document = documentBuilder.parse(inputStream);  
 }  
 }

解析XML文档的内容用来干嘛？无非就是增删改查遍历，**只要我们会对XML进行增删改查，那就说明我们是会使用DOM解析的**！

### DOM和SAX解析的区别：

**DOM解析读取整个XML文档，在内存中形成DOM树，很方便地对XML文档的内容进行增删改。但如果XML文档的内容过大，那么就会导致内存溢出！**

**SAX解析采用部分读取的方式，可以处理大型文件，但只能对文件按顺序从头到尾解析一遍，不支持文件的增删改操作**

DOM和SAX解析有着明显的差别，什么时候使用DOM或者SAX就非常明了了。

### dom4j的解析器

为什么需要有dom4j

* dom缺点:比较耗费内存
* sax缺点:只能对xml文件进行读取,不能修改,添加,删除
* dom4j:既可以提高效率,同时也可以进行crud操作

## XPATH

为什么我们需要用到XPATH

上面我们使用dom4j的时候，**要获取某个节点，都是通过根节点开始，一层一层地往下寻找，这就有些麻烦了**！

如果我们用到了XPATH这门语言，要获取得到XML的节点，就非常地方便了！