**XML资料**

XML技术

DTA/SCHEMA技术

JDOM/XQUERY

尚学堂科技 内部参考

**目 录**

[XML技术基础 3](#_Toc16003)

**[什么是XML?为什么需要XML?](#_Toc22147)** [3](#_Toc22147)

**[XML的特点](#_Toc29435)** [4](#_Toc29435)

[XML历史 5](#_Toc9177)

[我们的第一个XML程序 5](#_Toc9057)

**[XML基本语法](#_Toc16273)** [5](#_Toc16273)

[声明 6](#_Toc26391)

[XML元素 6](#_Toc22257)

[字符实体 7](#_Toc1042)

[CDATA 7](#_Toc14037)

[注释 8](#_Toc8076)

[Eclipse中编辑XML文件的插件/xmlBuddy 8](#_Toc5140)

[Eclipse下安装插件 8](#_Toc21859)

[DTD(Document Type Definition) 9](#_Toc29683)

[元素类型声明ETD 10](#_Toc29498)

[定义有效的元素属性 11](#_Toc7472)

[属性类型 12](#_Toc15833)

[SCHEMA(xsd)xml Schema Definition 12](#_Toc32369)

[Schema示例 13](#_Toc6753)

[命名空间 14](#_Toc7447)

[命名空间的定义方式 14](#_Toc592)

[XSTL显示XML文件 15](#_Toc30665)

[数据岛技术 17](#_Toc8292)

**[XML解析](#_Toc25136)** [18](#_Toc25136)

**[DOM方式解析](#_Toc9698)** [18](#_Toc9698)

[SAX(Simple APIs for XML ) 18](#_Toc19388)

**[使用Jdom解析](#_Toc8909)** [18](#_Toc8909)

**[使用JDOM开发的步骤 19](#_Toc12652)**

**[创建一个XML文档 19](#_Toc16650)**

**[修改XML文档 20](#_Toc15082)**

[上机作业 23](#_Toc25748)

# XML技术基础

## 什么是XML?为什么需要XML?

**XML(Extensible Markup Language)可扩展标记语言,一种描述数据的格式** 是的HTML语言也是一种标记语言。那么他们之间有什么联系吗？元标记语言，可以用来定义其他标记语言。首先，我们发现他也是一种标记语言。我们前面学过

HTML既描述了“显示什么”，又描述了“如何显示”。比如：<b>尚学堂科技</b>。这个描述了这样一个问题：

1. 显示的数据是：“尚学堂科技” *xml文件表示的”****有组织的数据****”*
2. 显示的方式是：粗体 *Css, XSL(Extensible Stylesheet Language), 数据岛技术*
3. 语法规定：<b> 而不是<b2><b3> ***DTD****(Document Type Definition), schema(xml文件)*

对于浏览器解析这些数据和显示数据，非常有用。所以，HTML一定程度上也可以描述数据。但是，我们并不能自己定义标记。比如，我要描述下面的数据：

“简历 姓名：朱元璋，原名：朱重八，性别：男 曾经从事职业：造反 职业：皇帝”

如何使用HTML来描述呢？我们做不到。因为，HTML不能进行标记的自定义。*(当然，话说回来，HTML本身跟XML没有任何关系。因为，他们都是标记语言。所以，我们为了便于学习，将他们列在一起比较。) 通过XML我们可以描述更加广义的数据：*

|  |
| --- |
| Xml描述数据  <resume >  <name>朱元璋</name>  <preName>朱重八</preName>  <gender>男</gender>  <preOccupation>造反</preOccupation>  <occupation>皇帝</occupation>  </resume>  Json描述数据  {  Resume:{  Name: ‘朱元璋’,  preName:’朱重八’,  Gender :’男’,  preOccupation:’造反’,  occupation:’皇帝’  }  } |
|  |
|  |

上面的文档中，所有标记都是我们自定义的。描述了朱元璋其人的简历内容。是一个非常典型的XML文档。

XML数据描述“显示什么”，这样的好处是数据和格式分离，结构清晰，便于阅读维护。

## Xml主要作用

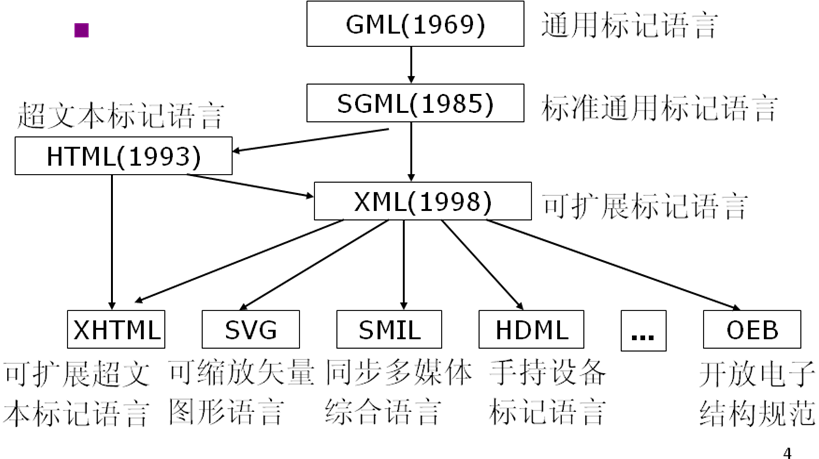
1）方便存储复杂数据在文件2）方便在网络传输复杂数据...

## XML的特点

* **XML是用来描述数据的，重点是“数据是什么”。**
* **优点：**
  + **数据与格式分离，共享数据，读取方便**
  + **自由的，可扩展的**
  + **面向对象的特性，XML的文件是以树状方式存储，有属性**
* **缺陷:**
  + **搜索的效率极高,但是插入和修改效率低**

**注：XML的本质就是有组织的数据。通过XML规范将数据进行合理的组织，便于人的阅读、便于机器解析，便于存储。**

## XML历史



## 我们的第一个XML程序

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <resume>  <name>朱元璋</name>2424  <preName>朱重八</preName>  <gender>男</gender>  <preOccupation>造反</preOccupation>  <occupation>皇帝</occupation>  </resume> |

然后可以通过普通浏览器打开查看。如果不是形势良好(Well-Formed)的XML文档，则会报错！形式良好指的是下面三点：

1. **严格要求元素嵌套**
2. **严格要求标记配对**
3. **只有一个根元素。**

**注意：XML区分大小写！**

## XML基本语法

XML文件的最基本构成

### 声明

**<?xml version=“1.0” encoding=“GB2312” standalone=“yes”?>**

* 三个属性
  + version属性
    - 在一个XML的处理指示中必须包括version属性指明所采用的XML的版本号，而且，它必须在属性列表中排在第一位。当前的XML最新版本是1.0.
  + encoding属性 （可选）
    - 所有的XML语法分析器都要支持8位和16位的编码标准。
    - 几个常见的编码：  简体中文码（GB2312,gbk）、繁体中文码（BIG5）、全球通用字符（UTF-8）
  + standalone属性 （可选）
    - 这个属性表明该XML文件是否和一个独立的标记声明文件配套使用。因此，如果该属性置为“yes”，说明不需要另外一个配套的DTD文件或者XSd文件。相反，如果这个属性置为“no”，则有可能有这样的文件。

### XML元素（标记，标签）

**语法：**

**<标记名 属性名1="值1" 属性名2="值2">**

**【元素的文本内容】或【子标记】**

**</标记名>**

### XML标记注意点

**1）描述数据必须通过标记**

**2）标记名，属性名等 都是严格区分大小写**

**3）标记必须有结束 如果标记没有【元素的文本内容】或【子标记】可以直接<标记名/>结束**

**例如：<person name="李四" age="18" />**

**4)属性的值必须用（单或双）引号包裹（推荐双引号）**

**5）xml文件描述数据有且只能有一个根标记**

**6）标记要正确嵌套**

### 数据1：张三，18，男，5000，四川省成都高新区布鲁明顿广场2-1001

### 数据2：张三，18，男，5000，四川省成都高新区布鲁明顿广场2-1001，两个手机（iphone6-4.7寸，note7，5.7寸）

### 字符实体

为了避免把字符数据和标记中需要用到的一些特殊符号相混淆，XML还提供了一些有用的实体引用。当在字符数据中需要使用这些特殊符号时，我们采用它的实体引用来代替。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 字符 | > | < | & | “ | ‘ |
| 实体引用 | &gt; | &lt; | &amp; | &quot; | &apos; |

在“示例”这个标记中出现文本:“<姓名>张三</姓名>”

〈示例〉&lt;姓名&gt;张三&lt;/姓名&gt;〈/示例〉

<STATEMENT VALUE = “She said, “Don’t go there!””>写为：

<STATEMENT VALUE = "She said, &quot;Don&apos;t go there!&quot;">

### CDATA

在一个特殊的标记CDATA下，所有的标记、实体引用都被忽略，而被XML处理程序一视同仁地当作字符数据看待。CDATA的形式如下：

**〈！[CDATA[ 文本内容 ]]〉**

注：CDATA的文本内容中是不能出现字符串“]]〉”

当你的文本数据中包含大量特殊符号时，你不得不通篇地使用实体引用，把本来很清晰的一段文字搞得乱七八糟。

|  |
| --- |
| <示例>          &lt;联系人&gt;           &lt;姓名&gt;张三&lt;/姓名&gt;           &lt;EMAIL&gt;zhang@aaa.com&lt;/EMAIL&gt;           &lt;/联系人&gt; </示例> |

为了避免这种不便，可以把这些字符数据放在一个CDATA数据块中，这样不管它看上去是一个标记还是一个实体引用，这些数据统统被当作字符照单全收。

|  |
| --- |
| <示例>         <![CDATA[              <联系人>              <姓名>张三</姓名>              <EMAIL>zhang@aaa.com</EMAIL>              </联系人>        ]]> </示例> |

### 注释

* 在HTML中，注释是用“〈！--”和“--〉”引起来的。在XML中，注释的方法完全相同。如：

<!-- <dd></dd>一个XML的例子 -->

* XML处理器对于注释中的一切内容都会视而不见，注释中出现的标记也一同被忽略。
* 在XML文件中使用注释时，同样要遵守几个规则
  + **在注释文本中不能出现字符串“**--**”**
  + 不要把注释文本放在标记之中，否则，它就不是一个“形式良好的”XML文件了。类似地，不要把注释文本放在实体声明中，也不要放在XML声明之前。
  + 注释不能被嵌套。

# Eclipse中编辑XML文件的插件/xmlBuddy

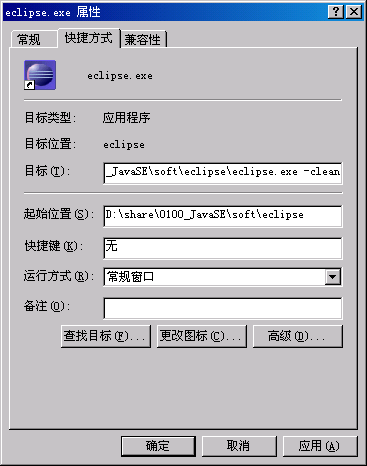
## Eclipse下安装插件

**Eclipse中插件的安装，使用link方式(以安装xmlbuddy为例)：**

1. 建立xmlbuddy目录，目录中必须包含：**eclipse/plugins**目录，然后将com.objfac.xmleditor\_2.0.72文件夹拷贝到%xmlbuddy%/eclipse/plugins目录下。
2. 在eclipse软件的子目录links中增加：xmlbuddy.link文件,文件内容如下：
   1. path=D:/eclipse3.3/xmlbuddy

**插件安装问题：**

1. 如果安装后看不到插件，把eclipse 目录下的configuration目录删除，重启即可。
2. 如果1中所述不起作用，你可以在你桌面eclipse的快捷方式上点右键->属性->快捷方式,在目标那一行的路径后加上"空格-clean"，然后重启eclipse即可。如图：



# DTD(Document Type Definition)

一个符合基本XML语法的文档，我们称之为“well-formed”，即形式良好的文档。但是不见得是有效的文档。比如说下面的XML文档：

<?xml version="1.0" encoding="UTF-8"?>

<resume>

<name>朱元璋</name>

<preName>朱重八</preName>

<gender>男</gender>

<preOccupation>造反</preOccupation>

<occupation>皇帝</occupation>

<compter>IBM</computer>

</resume>

上面的数据符合基本的XML规范，但是有一些不符合逻辑的数据。这个文档本来是用来描述朱元璋同志的简历的，结果出现了元素<compter>IBM</computer>，显然不符合基本的逻辑，不能算是有效的文档。那么是否有效由谁来定义? 这就需要我们的DTD(Document Type Definition)来告诉我们XML文档中可以有哪些元素。

DTD实际上是“元标记”这个概念的产物，它描述了一个标记语言的语法和词汇表，也就是定义了文件的整体结构以及文件的语法。简而言之，DTD规定了一个语法分析器为了解释一个“有效的”XML文件所需要知道的所有规则的细节。

为了说明特定的语法规则，DTD采用了一系列正则式，语法分析器将这些正则式与XML文件内部的数据模式相匹配，从而判别一个文件是否是有效的

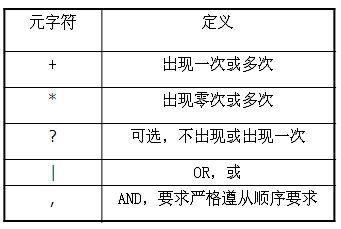
## 元素类型声明ETD

* ETD不但说明了每个文件中可能存在的元素，给出了元素的名字，而且给出了元素的具体类型。一个XML元素可以为空，也可以是一段纯文本，还可以有若干个子元素，而这些子元素同时又可以有它们的子元素。DTD正是通过元素之间的父子关系，描述了整个文件的结构关系。
* ETD应该采用如下的结构：

**<!ELEMENT 元素名 元素内容描述>**

|  |
| --- |
| <?xml version = "1.0“  encoding="GB2312"  standalone = “yes”?> <!DOCTYPE 联系人列表[     <!ELEMENT 联系人列表 **ANY**>     <!ELEMENT 联系人 (姓名)>     <!ELEMENT 姓名 (#PCDATA)>         ]>  <联系人列表>     <联系人>         <姓名>张三</姓名>     </联系人> </联系人列表> |

* 注意
  1. **除了根元素外，在定义其它元素时使用关键字ANY都是不好的习惯。**一个定义明确的DTD，实际上会使你在书写XML文件时有规可循，方便了你的工作和语法分析器的工作。
  2. 在定义元素时，ETD的顺序是无关紧要的。
  3. 不能对不同的元素使用相同的元素名，即便这些元素的内容、包含的子元素不同也不行，因为它只会引起文件各个元素的混淆，使文件的可读性大打折扣。
  4. 元素名的第一个字母必须是字母、或下划线（\_）、或冒号（：），后跟字母、数字、句号（.）、冒号、下划线、连结号（-）的组合，并且不能包含空白符，不能以“xml”开头。
  5. 尽管XML1.0标准允许使用任何长度的文件名，但是实际的XML处理器常常会限制标记名的长度。
* 元素定义是由它们的元素内容模型（ECM）来描述的，也就是说，是由紧跟元素后面的括号中的内容来定义的。
* ECM中的内容采取一组正则表达式的形式。



## 定义有效的元素属性

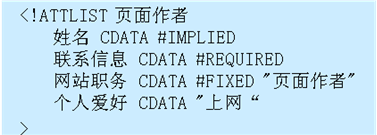
* <商品 类型 = "服装" 颜色 = "黄色">
* 在DTD中定义属性时，我们使用下面的格式：

**<!ATTLIST 元素名 （属性名 属性类型 缺省值）\*>**

* 元素名是属性所属的元素的名字；属性名是属性的命名；缺省值说明在XML文件中，如果没有特别说明属性的取值，语法分析器默认它具有的取值；属性类型则用来指定该属性是属于十个有效属性类型中的哪种类型。
* **注意：**由于ATTLIST是一个属性的列表，它可以包含很多属性，在实际应用中，**一个元素也经常有多个属性**。

<!ATTLIST 商品  
    类型 CDATA #REQUIRED  
    颜色 CDATA #IMPLIED  
>

* 必须赋值的属性 REQUIRED
  + 不用提供缺省值。
* 属性值可有可无的属性 IMPLIED
  + 文法解释器不再强行要求你在XML文件中给该属性赋值，而且也无须在DTD中为该属性提供缺省值。可以说，这是对属性值有无的最低要求，现实中经常用到。
* 固定取值的属性 FIXED
  + 提供一个缺省值，并且不希望XML文件的编写者把你的缺省值替代掉。
* 定义缺省值的属性
  + 不使用上面任何一种关键字的话，该种属性就是属于这种类型。对于这种属性，你需要在DTD中为它提供一个缺省值。**而在XML文件中可以为该属性给出新的属性值来覆盖事先定义的缺省值，也可以不另外给出属性值，后一种情况下它就默认为采用DTD中给出的缺省值**。



## 属性类型

* 一个元素属性可以为以下十种类型中的任意一种：
  + CDATA
  + Enumerated
  + ID
  + IDREF
  + IDREFS
  + ENTITY
  + ENTITIES
  + NMTOKEN
  + NMTOKENS
  + NOTATION

### SCHEMA(xsd)xml Schema Definition

* Schema也是XML的一种应用，它是将DTD重新使用XML语言规范来定义。
* 这个利用DTD把DTD改写成的Schema，它可具有许多“青出于蓝而胜于蓝”的优势。
  + 一致性
    - DTD的书写结构和XML文件的结构仿佛有着天壤之别，后者清晰直观，前者复杂晦涩。
    - Schema建立在XML之上，它的样子和一般的XML文件完全相同，使得XML达到了从内到外的完美统一。因为Schema本身也是一种XML，可以被现有的XML编辑制作工具所编辑、被XML语法分析器所解析、被XML应用系统所利用，既有投资得到了最大程度的保护。
  + 扩展性
    - DTD没有数据类型，无形之中大大增加了程序员的开发难度和工作量。
    - 虽然DTD中也定义了一些数据类型，但那都是针对属性类型而定义的，而且类型非常有限。
    - Schema对DTD进行了扩充，引入了数据类型，很好地解决了这一问题。
  + 易用性
    - XML Schema取代DTD的另一个原因要归结于DOM和SAX。作为一种XML API，DOM和SAX只是对XML实例文档有效，对于DTD则无能为力。
  + 规范性
    - 同DTD一样，Schema也提供了一套完整的机制以约束XML文档中标记的使用，但相比之下，后者基于XML，更具有规范性。
  + 互换性
    - 可根据需要设计适合自己应用的Schema。通过映射机制，还可以将不同的Schema进行转换，以实现更高层次的数据交换。

## Schema示例

|  |
| --- |
| <?xml version="1.0" encoding=“GB2312” ?> <Schema xmlns=“urn:schemas-microsoft-com:xml-data”                 xmlns:dt=“urn:schemas-microsoft-com:datatypes”>     <AttributeType name=“公司”/>     <ElementType name= “姓名”/>     <ElementType name= “ID”/>     <ElementType name= “公司”/>    <ElementType name= "EMAIL"/>     <ElementType name= "电话" dt:type="fixed.14.4"/>     <ElementType name= "街道"/>    <ElementType name= "城市"/>   <ElementType name= "省份"/>   <ElementType name= "地址" content="eltOnly">    <element type= “街道” />  <element type= "省份" />         <element type= "城市" />   </ElementType>   <ElementType name= "联系人" content="eltOnly">        <element type= "姓名" />        <element type= "ID" />        <element type= "公司" />        <element type= "EMAIL" />        <element type= "电话" />        <element type= "地址" />   </ElementType>   <ElementType name= "联系人列表" content="eltOnly">        <element type= "联系人" />        <attribute type="公司"/>   </ElementType> </Schema> |

# 命名空间

* XML较之HTML最根本的差别在于XML是定义标记语言的元语言，而HTML仅仅是由SGML元标记语言定义的一个实例语言。针对不同的应用方向，每设计一个XML的DTD，一种新的标记语言便随之诞生。在各种各样XML实例标记语言不断涌现的过程中，将会产生这样一种应用需求，**即在一个XML文档中，包含由多个DTD描述的元素。这个想法显然是达到“物尽其用”的一个好办法，它帮助我们最大程度地利用了现有的资源，正所谓“海纳百川，有容乃大”。**
* 如果不解决元素名称上的冲突问题，一个XML文档包含多个DTD中定义的元素这一天才构想就不能实现。为了解决这个问题，W3C的XML小组制定了被称为**命名空间（NameSpace）的标准**。W3C组织于1998年2月提出命名空间标准的第一个草案，到1999年1月14日正式发布为推荐标准。
* 命名空间标准提供了一个很聪明的方法，就是所谓**前缀标识法，即在元素名和属性名前面增加一个标识，以唯一区分当前元素或属性来自哪一个DTD。**

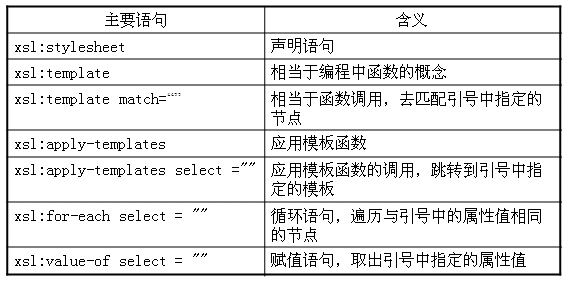
|  |
| --- |
| <联系人:联系人列表 **xmlns:联系人="联系人列表.dtd" xmlns:企业经理 ="企业经理.dtd">**   <联系人:联系人>     <联系人:姓名>张三</联系人:姓名>     <联系人:ID>001</联系人:ID>     <联系人:公司>A公司</联系人:公司>     <联系人:EMAIL>zhang@aaa.com</联系人:EMAIL>     <联系人:电话>(010)62345678</联系人:电话>     <联系人:地址>       <联系人:街道>五街1234号</联系人:街道>       <联系人:城市>北京市</联系人:城市>       <联系人:省份>北京</联系人:省份>     </联系人:地址>     <联系人:直接上级>       <企业经理:姓名>王五</企业经理:姓名>       <企业经理:电话>         <企业经理:秘书电话>(010)62345678</企业经理:秘书电话>         <企业经理:手机>13601234567</企业经理:手机>       </企业经理:电话>     </联系人:直接上级>   </联系人:联系人> </联系人:联系人列表> |

## 命名空间的定义方式

* 直接定义方式
  + **xmlns:<命名空间前缀> = <命名空间名>**
  + <联系人:联系人列表 xmlns:联系人 ="http://xml.net.cn/联系人列表.dtd" xmlns:企业经理 ="http://xml.net.cn/企业经理.dtd">
  + 命名空间声明的属性值部分是一个URI引用，其功能是区分不同的命名空间，因此，这个URI引用被称为“命名空间名”，它应该具有唯一性和持久性。
* 缺省定义方式
  + **xmlns = <命名空间名>**
  + <联系人:联系人列表 xmlns ="http://xml.net.cn/联系人列表.dtd" xmlns:企业经理 ="http://xml.net.cn/企业经理.dtd">

# XSTL显示XML文件

* 样式单(Style Sheet)是一种专门描述结构文档表现方式的文档。与HTML描述数据显示方式的传统方法相比，样式单有许多突出的优点：
  + 表达效果丰富
  + 文档体积小
  + 便于信息检索
  + 可读性好
* XML关于文档浏览的基本思想是将数据与数据的显示分别定义。
* 定义不同的样式表可以使相同的数据呈现出不同的显示外观，从而适合于不同应用，甚至能够在不同的显示设备上显示。这样，XML数据就可以得到最大程度上的重用性，满足不同的应用需求。
* 迄今为止，W3C已经给出了两种样式单语言的推荐标准
  + 层叠样式单CSS（Cascading Style Sheets）
  + 可扩展样式单语言**XSL（eXtensible Stylesheet Language）**。



|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?>  <xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">  <xsl:template match="/">  <HTML>  <HEAD>  <title>朱元璋个人简历</title>  </HEAD>  <BODY>  <xsl:apply-templates select="resume"/>  </BODY>  </HTML>  </xsl:template>  <xsl:template match="resume">  <table border="1">  <tr>  <td>姓名：</td>  <td><xsl:value-of select="name"/></td>  </tr>  <tr>  <td>曾用名：</td>  <td><xsl:value-of select="preName"/></td>  </tr>    <tr>  <td>第一份工作：</td>  <td><xsl:value-of select="preOccupation"/></td>  </tr>  <tr>  <td>第二份工作：</td>  <td><xsl:value-of select="occupation"/></td>  </tr>    </table>  <br/>  </xsl:template>  </xsl:stylesheet> |
| XML文档：  <?xml version="1.0" encoding ="utf-8" standalone = "no"?>  <?xml-stylesheet type="text/xsl" href="03.xslt"?>  <!DOCTYPE resume SYSTEM "03.dtd" >  <resume>  <name>朱元璋</name>  <preName>朱重八</preName>  <preOccupation>要饭</preOccupation>  <preOccupation>和尚</preOccupation>  <preOccupation>&lt;造反&gt;</preOccupation>  <occupation>皇帝</occupation>  </resume> |
| 显示效果： |

# 数据岛技术

* 什么是数据岛？(data Islands)
  + XML的数据源对象又称为DSO(data source object)或数据岛。
  + 数据岛是指存在于HTML页面中的XML代码。
* 数据岛的优势
  + 将XML数据与HTML页面显示完美结合起来
* 数据岛的缺陷
  + 只能在 IE 下运行，firefox等浏览器下无法使用。
  + 安全性：xml 数据岛可通过查看 html 的源代码获取
* DSO数据显示方式
  + 和SPAN标记捆绑
    - <span datasrc=”#数据源的名称” datafld=”某种标记或属性的名称” ></span>
  + 和DIV标记捆绑
    - <div datasrc=”#数据源的名称” datafld=”某种标记或属性的名称”></div>
  + 和INPUT标记捆绑
    - <input datasrc=”#数据源的名称” datafld=”某种标记或属性的名称”></input>
  + 和TEXTAREA标记捆绑
    - <textarea datasrc=”#数据源的名称” datafld=”某种标记或属性的名称”></textarea>
* **数据岛的使用方式**
  + 内嵌数据岛
    - <xml id=“xmlid”> … </xml>
  + 外联的数据岛
    - <xml id=“xmlid” src=“1.xml“ />

# XML解析

XML文档如何在程序中进行解析处理?一般有两种方式：DOM和SAX.

## DOM方式解析

* DOM 构建整个文档驻留内存的树。如果文档很大，就会要求有极大的内存。
* DOM 创建表示原始文档中每个东西的对象，包括元素、文本、属性和空格。如果您只需关注原始文档的一小部分，那么创建那些永远不被使用的对象是极其浪费的。
* DOM 解析器必须在您的代码取得控制权之前读取整个文档。对于非常大的文档，这会引起显著的延迟。
* 这些仅仅是由文档对象模型的设计引起的问题；撇开这些问题，DOM API 是解析 XML 文档非常有用的方法。

## 

## SAX(Simple APIs for XML )

* SAX 解析器向您的代码发送事件。当解析器发现元素开始、元素结束、文本、文档的开始或结束等时，它会告诉您。您可以决定什么事件对您重要，而且可以决定要创建什么类型的数据结构以保存来自这些事件的数据。如果您没有显式地保存来自某个事件的数据，它就被丢弃。
* SAX 解析器根本不创建任何对象，它只是将事件传递给您的应用程序。如果希望基于那些事件创建对象，这将由您来完成。
* SAX 解析器在解析开始的时候就开始发送事件。当解析器发现文档开始、元素开始和文本等时，代码会收到一个事件。您的应用程序可以立即开始生成结果；您不必一直等到整个文档被解析完毕。更妙的是，如果您只查找文档中某些内容，代码一旦找到所要找的东西就可以抛出一个异常。该异常会停止 SAX 解析器，然后代码用它找到的数据做它需要做的任何事。
* 这种处理的好处非常象流介质的好处；可以立即开始分析，而不是必须等待所有要处理的数据。同样，由于应用程序简单地检查经过其的数据，所以不需要将数据存储在内存里。当遇到大文档时，这是一个突出的优势。一般来讲，SAX 还比其替代物“文档对象模型”快。
* 另一方面，由于应用程序不以任何方式存储数据，所以，使用 SAX 时，不可能对数据进行更改，或者“返回”至数据流中前面的数据。

## 使用Jdom解析

实际开发中，我们往往借助一些开源软件实现，较典型的有：JDOM、DOM4J、XQUERY等。下面我们以JDOM为例，告诉大家如何的XML文档进行解析。其他开源类库，大家可以类比学习。我们在此不做累述。

* JDOM提供了以DOM思想操作XML数据文件的方式，但和标准的DOM接口并不兼容。但由于其简单性，在java开发领域有着广泛的应用。
* JDOM是为java优化的。
* JDOM**试图组合DOM和SAX的优点。它被设计成一个可以在小内存上快速执行轻量级api 。**

### 使用JDOM开发的步骤

第一步：需要加jdom.jar。

### 创建一个XML文档

**public** **static** **void** createXML() **throws** FileNotFoundException, IOException {

//构建元素节点和相关属性

Element root = **new** Element("resume"); //<resume></resume>

Element name = **new** Element("name"); //<name></name>

Element preName = **new** Element("preName");

Element occupation = **new** Element("occupation");

Attribute attr = **new** Attribute("dynasty","唐朝");

occupation.setAttribute(attr); // <ocucupation dynasty="唐朝" ></occupation>

Element preOccupation = **new** Element("preOccupation");

name.setText("李世民"); //<name>李世民</name> //setText只用于给元素增加文本。

preName.addContent("秦王"); //<preName>秦王</preName> //addContent：不光可以给元素增加文本，也可以增加元素，注释等等。

preOccupation.addContent("将军"); //<preOccupation>将军</preOccupation>

occupation.addContent("皇帝");

root.addContent(name);

// <resume>

// <name>李世民</name>

// <preName>秦王</preName>

// </resume>

root.addContent(preName);

root.addContent(occupation);

root.addContent(preOccupation);

Document doc = **new** Document(root);

// Format f = Format.getPrettyFormat();

Format f = Format.*getCompactFormat*();

f.setEncoding("gbk"); //设置生成的xml文档的编码方式：<?xml version="1.0" encoding="gbk"?>

XMLOutputter xmlOut = **new** XMLOutputter(f);

xmlOut.output(doc, **new** FileOutputStream("d:/2.xml"));

}

### 修改XML文档

**public** **static** **void** updateXML(String filePath) **throws** FileNotFoundException, JDOMException, IOException {

SAXBuilder sb = **new** SAXBuilder();

Document doc = sb.build(**new** FileInputStream(filePath)); //将某一个xml文件读入到doc中。

//对内存中的doc进行操作。

Element root = doc.getRootElement(); //一旦获得了根元素，就可以对所有XML数据进行处理。

// Element e = root.getChild("occupation");

// System.out.println(e.getText()); //返回元素的文本内容

// System.out.println(e.getName()); //返回元素的标记名

// System.out.println(e.getAttributeValue("dynasty")); //返回元素的属性值

// e.setText("lishimin"); //通过setText,getText直接设置元素的文本

// List li = root.getContent(); //所有内容：子元素、注释、文本等等

// List li1 = root.getChildren(); //只有标记内容

// System.out.println(li.size());

// System.out.println(li1.size());

// List list = root.getChildren();

// Element ele = (Element) list.get(0);

//// System.out.println(ele.getText());

// Attribute attr =new Attribute("国家","中国");

// ele.setAttribute(attr);

XMLOutputter xmlOut = **new** XMLOutputter(); //讲doc写入到某一个xml文件中，从而更新硬盘中的文件

xmlOut.output(doc, **new** FileOutputStream(filePath));

}

### 使用XPATH解析XML文档

（需加入jar包：saxpath.jar，jaxen-core.jar，jaxen-jdom.jar）

**public** **static** **void** testXPath(String filePath) **throws** FileNotFoundException, JDOMException, IOException {

// **TODO** Auto-generated method stub

SAXBuilder sb = **new** SAXBuilder();

Document doc = sb.build(**new** FileInputStream(filePath)); //将某一个xml文件读入到doc中。

//对内存中的doc进行操作。

Element root = doc.getRootElement(); //一旦获得了根标记，就可以对所有DOM中的数据进行处理。

XPath xpath = XPath.*newInstance*("/resume/\*"); //返回resume节点下面所有的子节点。

// XPath xpath = XPath.newInstance("//preOccupation"); //返回文档中所有preOccupation节点。

// XPath xpath = XPath.newInstance("/resume/wife//preOccupation"); //返回/resume/wife所有preOccupation节点。

// XPath xpath = XPath.newInstance("/resume/wife/preOccupation"); //返回/resume/wife/preOccupation这个节点。

// XPath xpath = XPath.newInstance("/resume/wife/preOccupation[@period]"); //返回名字为preOccupation有属性peroid的节点。

// XPath xpath = XPath.newInstance("//preOccupation[@period]"); //返回文档中所有preOccupation节点，并且有属性period

// XPath xpath = XPath.newInstance("/resume/wife/preOccupation[@period='5-18']"); //返回名字为preOccupation属性peroid='5-18'的节点。

// XPath xpath = XPath.newInstance("//preOccupation[text()='皇子']"); //返回文档中所有preOccupation节点，并且文本内容为皇子。

List list = xpath.selectNodes(root);

**for** (**int** i = 0; i < list.size(); i++) {

Element e = (Element)list.get(i);

System.*out*.println(e.getText());

}

// Element e2 =(Element) xpath.selectSingleNode(root);

// System.out.println(e2.getText());

}