XML

## XML课程简介

1. Extensibale Markup Language(可扩展的标记语言)-XML
2. 为什么要学习XML？

进行数据交换、电子商务的基础、可扩展的开放的标记语言。。。

1. XML的知识体系

* XML的语法规则、
* XML核心技术

文档描述、验证、约束技术—DTD/Schema

* 文档转换技术—xsl
* 数据查询技术—Xpath
* 链接技术—Xlink/Xpointer
* 编程接口— DOM/SAX

1. 学习的主要内容

* XML的基本格式
* XML描述、验证、约束技术
* XML的数据转换
* DOM
* XML的应用

## 什么是XML

XML—Extensible Markup Language

传统的系统已经远远不足以来表达复杂的信息，简单的语言根本无法表达出一些细微的差别，需要更完整的语言来表达网络世界里日益繁杂的信息内涵。XML—可扩展标记语言由此应用而生了，它的出现一改HTML只追求美观的的效果，而不注重实际交流应用的现象，把语言的核心从装饰应用转变为商务应用。

它的出现把网络表达的语言集合推进了一大步，XML传递信息，表达知识，具有跨平台的特性，它作为数据交换和网络计算的基础，将无可非议的成为网络的通用语言，尤其是在电子商务的应用上出色的表现，现在已经没有人怀疑它给信息社会带来的革命性影响了。

## XML是元标记语言

元标记：通过标记就可以知道元素的含义，要表示的是什么意思。

* 关于XML要理解的第一件事就是，它不是像超文本标记语言(HTML)那样是格式化的程序，这些语言定义了一套固定的标记，用来描述一定数目的元素。如果标记语言中没有所需要的标记，用户也就没有办法了，这时只能等待标记语言的下一个版本了。希望在下一个版本中能够包含所需要的标记，这样就要依赖软件开发商的选择。
* XML是一种元标记语言，用户可以定义自己需要的标记。这些标记必须根据某些通用的原理来创建，但是在标记的意义上，也具有相当的灵活性。例如：我们处理学生相关的信息，那么就要描述学生的姓名、年龄、性别、家庭住址、学院等，这就需要创建应用每项的标记。新创建的标记可在文档类型定义(Document Type Definition，DTD)中描述，在后面将会学到DTD更多的内容，现在只需要将DTD看做是词汇表或文档的句法。
* 使用XML就意味着不必等待浏览器的开发商来满足用户的需求了。用户可以创建自己需要的标记。当需要时，告诉浏览器如何显示这些标记就可以了。
* 关于XML要理解的第二件事就是，XML描述的是结构和语义，而不是格式化。

第一个XML文档：

<?xml version=”1.0”?>

<students> —开始元素

<student>

<name>张三</name>

<gender>男</gender>

</student>

</students>—结束元素

注意：XML的元素必须成对出现，元素之间必须正确嵌套，先开始的必须先关闭，只能有一个根元素（唯一一个包含了其他所有元素的那个元素称为根元素）。students是父元素，student是子元素。

* 它不描述页面元素的格式化。可以用样式单为文档增加格式化信息。文档本身只说明文档包括什么标记，而不是说文档看起来是什么样的。
* 作为对照，HTML文档包括了格式化，结构和语义的标记。<b>就是格式化标记，它使其中的内容加粗。<strong>是一种语义标记，意味着其中的内容特别重要。<td>是结构标记，指明内容是表中的一个单元。有些标记可能具有以上三种意义，例如：<h1>同时表示了字体加粗又是第一级标题，又是页面标题。

XML描述的是文档的内容与语义，而不是文档如何去显示。

* 什么是标记语言：当我们需要通过标记将有用的信息告知一组用户时，我们必须有一个标准，用它来描述什么是有效标记。在HTML中所谓的标记就是标签:<tr>。还要有一个标准来描述每个标记的具体含义。

## XML的定义目标

* XML可以在互联网上直接使用
* XML应该支持各种不同的应用方式
* 处理XML文件的应用程序应该容易编写
* XML中的可选特性的数量应该减到最少，最好减至没有
* XML文档应该具有良好的可读性，并且比较清晰
* 用XML设计新的标记语言应该正式、简洁
* XML文件应该容易编制
* XML标记的简洁性并不重要
* 可扩充性
* 语义与表现形式的分离
* 在Internet上的可用性
* 简单性
* 国际化

## XML的优越性

1. XML是使信息自描述的新语言。自描述：自己定义元素
2. 信息共享，自定义数据格式，很容易使用工具读写

<student>

<name>zhang san</name>

<gender>male</gender>

</student>

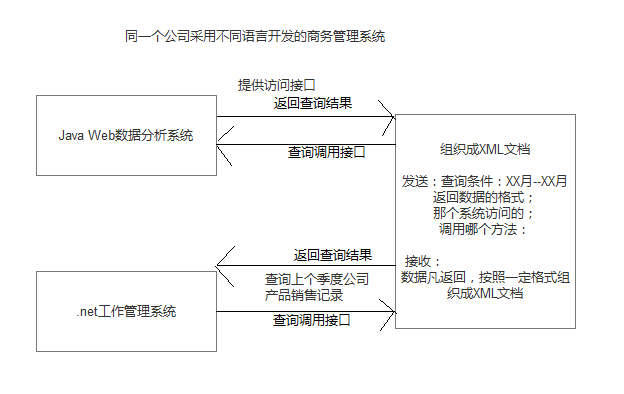
1. 数据传递，支持各种数据通道的传递
2. 数据重用
3. 分离数据和显示
4. 可扩展
5. 文档包含语义
6. 其他：容易阅读、工具、树形结构的面向对象编程
7. XML与HTML的比较

|  |  |  |
| --- | --- | --- |
| 比较内容 | HTML | XML |
| 可扩展性 | 不具有扩展性 | 是元标记语言，可用于定义新的标记语言 |
| 侧重点 | 侧重于如何表现信息 | 侧重于如何结构化的描述信息 |
| 语法要求 | 不要求标记的嵌套，配对等，不要求标记之间有一定的顺序 | 严格要求嵌套、配对和遵循DTD的树形 结构 |
| 可读性及可维护性 | 难以阅读、维护 | 结构清晰、便于阅读维护 |
| 数据和显示的关系 | 内容描述与显示方式整合为一体 | 内容描述与显示方式相分离 |
| 保值性 | 不具有保值性 | 具有保值性 |

## XML优点总结

1. 更为有意义的搜索：由于数据可以被XML唯一标识，所以搜索起来更为准确简便。XML可以不管你数据库的数据结构如何，直接按照你的需求进行分类，这样便于统一搜索的高速准确。
2. 开发灵活的Web应用软件：XML数据能够被结构化的发送到其他软件，对象或服务器上作进一步的处理。这对于开发灵活应用的web软件很有帮助。
3. 不同来源的数据集成：XML提供了将不同来源的结构化数据集成的强大功能，也能实现不兼容数据库的搜索。

举例：异构平台的相互交互



最基本的Web Service工作原理，通过web service可以实现异构语言、异构平台之间的交互。

1. 多种应用得到数据：XML的扩展性使得它可以使数据不需要内部描述就能被交换和处理，因为它的数据都是自我描述的。
2. 本地计算机和处理：XML格式的数据发送给客户后，允许客户使用XML文档对象DOM用脚本或其他编程语言来处理，数据不需返回服务器在本地就能进行计算。解放了一些只能在服务器上运行的软件。
3. 数据的多功能显示：XML支持的显示模式可以使数据与内容分开，显示可以根据用户的不同配置而有所不同，即个性化显示。
4. 颗粒状更新：当每一部分数据变化后，不需要重新改变整个结构，只要将变化的部分从服务器发给客户就可以了，有很强的可维护性。
5. 在Web上进行数据传送：XML不需要对现在的网路做任何的变化，就可以跟HTML一样发送数据。
6. 可适性：使用XML可以自己定义一套适合自己的标准，在合适的范围内使用。

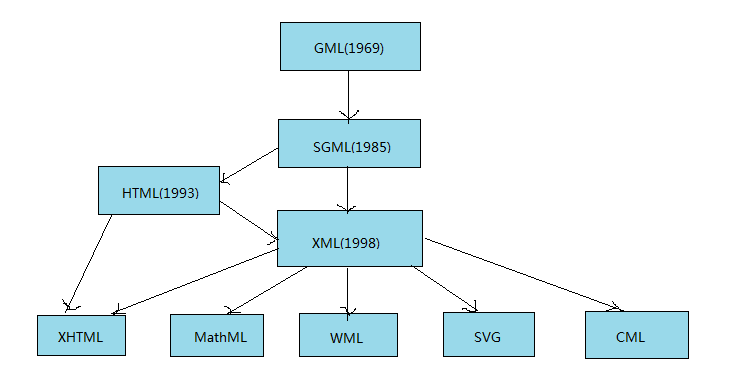
## 六、XML的发展

XML有两个先驱：SGML和HTML，这两个语言都是非常成功的标记语言，但是在某些方面存在着不足与缺陷。XML正是为解决他们的不足而诞生的。

SGML的全称是标准通用化标记语言，它从80年代初就开始使用。同XML一样，SGML也可以创建成千上万的标记语言。它为语法置标提供了强大的工具，同时具有极好的扩展性，因此在分类和索引数据中非常有用。SGML多用于科技文献和政府办公文件中。

HTML是免费、简单的，而且获得了广泛的支持。它是一个非常简单的SGML语言，可以方便普通人的使用。而正如设计之初所构想的那样，HTML现在在世界范围内得到了广泛的应用。不幸的是，HMTL有许多致命的缺陷。

XML定义的标记语言家族



自XML发明以来，又有一大批用XML定义的新的标记语言随之诞生，它们有的仍然处于草案阶段，还有一些已经正式被W3C推荐为正式标准，开始在各个领用发挥着他们，同时也是XML的巨大优势。

## 七、基本的XML语法

XML的文档结构；XML文档专用标记；元素和属性；字符和实体引用

### XML1.0语法

* XML是自描述数据
* 使用现有的国际标准
* 格式正规的XML
* 有效的XML

### XML解析器

什么是XML解析器？

读取XML文档提供对文档内容的访问的软件模块

类型：无验证的解析器和有验证的解析器

解析器：MSXML IE IBM XML4J Apache Xerces Oracle XML Parser

例如：IE浏览器能够打开并展示XML文档，正是因为IE浏览器中嵌入了一个XML的解析器，它能够正确的解析XML元素之间的嵌套关系，层次结构。能够将解析后的结果呈现出来，并提供可操作的“-”“+”操作符，来对整个XML文档进行层次的展现。

### 简单的XML文档

第一行是XML声明：<?xml version=”1.0” standalone=”yes”?>，这个是XML处理指令的例子，处理指令以“<?”开始，以“?>”结束。在“<?”后面的第一个单词是处理指令名，在本例中是xml。

standalone是指令中的涵义，表示xml文档是独立的，没有任何关联的文件，也就是说没有对它进行验证的文件关联，此文件时独立存在的。

Xml文档的处理指令<?一定要顶格写，不能有空格和其他操作符，指令紧跟着?号，中间同样不能有空格。

XML声明有version个standalone两个特性，特性是由等号分开的名称-数值对。位于等号左边的是特性名，右边的是特性的值。并用双引号或单引号扩起来。

在我们自定义的元素中，同样可以存在属性，属性的形式为：属性名=”属性值”，多个属性之间使用空格隔开。

<?xml version=”1.0” standalone=”yes”?>

<students>

<student id=”001” gender=”male”/>

<student>

<id>001</id>

<gender>male</gender>

</student>

</students>

编码格式:encoding，默认不写的话使用UTF-8，建议不写，因为UTF-8是使用最广的编码格式。

每一个XML文档都是以一个XML声明开始，用以指明所用的XML的版本号。在上面例子中version特性指明这个文档符合XML1.0的规范。XML声明还可以有standalone特性，这个告诉我们文档是否存在这一个文件里还是需要从外部导入文件。在本例中，所有的文档都在一个文件里完成，因此standalone的值设置为true。

上述例子中：<students>是开始标记，</students>是结束标记，<student>是开始标记,</student>是结束标记，<id>是开始标记,</id>是结束标记。Id之间的值是元素的内容。

### 为XML编写样式单

XML允许用户来创建任何所需要的标记，由于用户在创建标记上有完全的自由，所以浏览器无法预期用户的标记的意义，也无法为显示这些标记提供规则。因此用户需要自己编写样式单，来告知浏览器怎样显示特定的标记。用户创建的样式单可以由不同的文档不同的人来使用，还可以将自己写的样式单与他人写的样式单集成在一起使用。

举例：编写一个简单的CSS样式

<?xml version=”1.0” ?>

<student>张三</student>

样式文件：student.css

student{display:block;font-size:20pt;color:red}

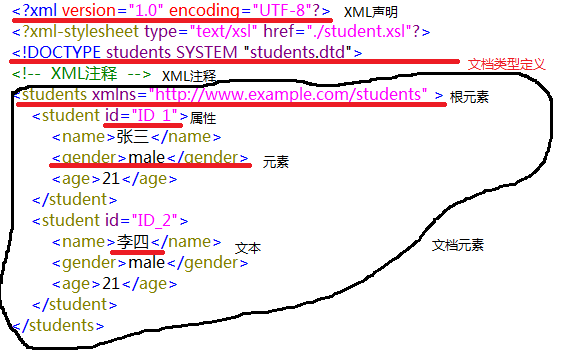
建立xml与css样式表的关联，通过xml指令建立关联

<?xml-stylesheet type=”text/css” href=”./student.css”?>

注意：理解内容与展现形式相分离的概念。

现在有两种样式单可以使用：一、使用级联样式单（CSS）Cascading Style Sheet，简写为CSS。CSS的优势在于它是W3C制定的标准，为编写HTML的许多人所熟悉。且被具有XML能力的浏览器所支持。二、另一种是可扩展的样式语言(Extensible Style Lanuage，XSL)，XSL是当前最强大的最灵活的样式语言，是特别为应用XML而设计的。但是XSL比CSS更为复杂。

### XML文档实例分析



注意：XML中注释是不允许嵌套的，原因是XML在开始标记后，会寻找他的结束标记，如果用注释去嵌套，会改变整个XML文档的结构，导致文档不能正确显示。

### XML文档—属性

<student id=”001”>

<name>zhang san</name>

</student>

属性值用双引号(“)或者单引号(‘)进行分隔，一个元素可以有多个属性，特定的属性名在一个元素中只允许出现一次，属性值不能包含<，>，&特殊字符。

注意：属性名在一个元素中不能重复出现。

### 实体

* 作用：避免重复输入，类似于java中定义的变量
* XML中预定义的实体：1、&lt; 等同于< 2、&gt;等同于> 3、&amp; 等同于& 4、&quot等同于 ” 5、&apos; 等同于 ‘

注意：实体都是以 &开始;结束。

* 自定义实体语法：

<!DOCTYPE 根元素[

<!ENTITY 实体名 “实体内容“>

]>

* 引用已定义的实体：&实体名;

举例：<?xml version="1.0"?>

<!DOCTYPE directory [

<!ENTITY dis "the frist">

]>

<directory>

&dis;

</directory>

### XML专用标记

#### 注释

<!-- this is commonts -->

注意：注释内容中不能出现--；不能把注释放在标记中间如：<name <!-- the name -->>

John</name>;注释不能嵌套；可以在标记以外的任何地方写注释；

#### 处理指令——PI

处理指令用于XML解析器传递信息到应用程序。1）XML解析器是读取并保存XML文档内容的软件模块。2）应用程序是从解析器获取文档内容并处理和显示这些内容的独立软件模块。3）IE同时提供XML解析器和应用部分。

* 语法<?目标 指令?>。1）PI必须以一个叫做目标的标识符开头，这个标识符遵从如同元素和属性一样的规则，目标是指令所指向的应用的名称。2）指令是传递给应用程序的信息。
* <?开始，?>结束
* 样式表处理指令。1）指令为xml-stylesheet，这个指令必须出现在序言部分，根元素之前。2）使用方式：<?xml-stylesheet type=”type” href=”uri”?>，type的值为text/css或者text/xsl。

#### CDATA节

* 用于把整段文本解释为纯字符数据而不是标记的情况。包含大量的<，>,&,”,’,字符，CDATA节中所有字符都会别当作元素字符数据的常量部分，而不是XML标记。
* 语法：<![CDATA[

………..

]]>

* 可以输入任意字符，除“]]>”外。
* 不能嵌套，并且不能写到根元素之外，只能在文档元素中写入。

举例：

<?xml version=”1.0”?>

<root>

<![CDATA

<hello>

<world>

]]>

<child>

</child>

<root>

#### DOCTYPE文档类型定义

* 文档类型声明，紧跟XML声明之后，包含所有实体声明。
* 语法<!DOCTYPE 根元素标记名[

实体声明：例如<!ENTITY 实体名 “实体的值”>

]>

举例：<!DOCTYPE student[

<!ENTITY name “张三”>

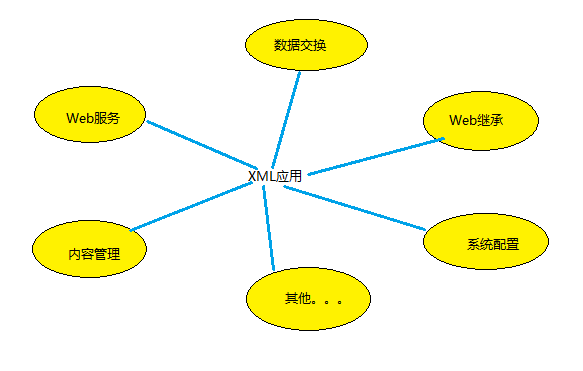
]>

引用：<name>&name;</name>

### 格式正规的XML文档

遵守如下规则的XML文档，称为格式正规的XML文档。1）必须有xml的声明，例如<?xml version=”1.0”?>。2）必须有且只有一个根元素。3）标记大小写敏感。4）属性值用引号，单引号双引号都可以，但必须一致。4）标记成对出现。5）空标记必须关闭，可以自关闭。例如：<book />。6）元素正确嵌套。7）名称中可以包含字母数字或其他字符。8）名称不能以数字开头。9）名称中不能含有空格。10）名称中不能包含：（冒号被保留，用作命名空间使用）-- well formed(格式正规的)

## 八、xml的应用范围



# DTD

Document Type Difinition (DTD)，用它来对XML文档进行格式校验，通过校验的XML文档才是格式有效的XML文档，首先XML文档必须是格式正规的文档，然后才能是格式有效的XML文档。（valid）

## 1、文档类型定义和命名空间

简介：理解文档类型定义，DTD的声明，元素的声明，属性的声明，实体的声明、命名空间。

## 2、什么是DTD，为什么使用DTD

DTD就是：文档类型定义Document Type Difinition，DTD用来描述XML文档结构，一个DTD文档包含：1）元素(ELEMENT)的定义规则。2）元素之间的关系规则。3）属性(ATTLIST)的定义规则。4）可以使用实体(ENTITY)或符号(NOTATION)的规则。

注意：XML文档符合这些规则，才能称得上是有效的XML文档。

* DTD文档与XML文档的实例关系。1）类与对象的关系2）数据库表结构与数据记录的关系。
* 使用DTD，每个XML文件可以携带一个自身格式的描述。
* 不同组织的人可以使用一个通用的DTD来交换数据。
* 应用程序可以使用DTD来校验从外部传入的XML是否有效。
* 可以使用DTD校验自己的XML数据。

## 3、生成DTD文档

DTD文档的声明及引用。

内部的DTD文档：<!DOCTYPE 根元素 [定义内容]>

外部的DTD文档：<!DOCTYPE 根元素 SYSTEM “DTD文档路径”>

内外部DTD文档结合：<!DOCTYPE 根元素 SYSTEM “DTD文档路径”[

<!-- 内部定义 -- >

]>

举例1：<?xml version=”1.0”?>

<!DOCTYPE email [

<!ELEMENT email (from,to,content)>

<!ELEMENT from (#PCDATA)>

<!ELEMENT to (#PCDATA)>

<!ELEMENT content (#PCDATA)>

]>

<email>

<from>zhang san</from>

<to>li si</to>

<content>别忘了还钱</content>

</email>

说明：#PCDATA的意思就是可解析的字符数据，简单理解就是字符串。

举例2：<?xml version=”1.0”?>

<!DOCTYPE email SYSTEM “./email.dtd”>

<email>

<from>zhang san</from>

<to>li si</to>

<content>别忘了还钱</content>

</email>

DTD文件：email.dtd

<!ELEMENT email (from,to,content)>

<!ELEMENT from (#PCDATA)>

<!ELEMENT to (#PCDATA)>

<!ELEMENT content (#PCDATA)>

举例3：<?xml version=”1.0”?>

<!DOCTYPE email SYSTEM “./email.dtd” [

<!ELEMENT to (#PCDATA)>

<!ELEMENT content (#PCDATA)>

]>

<email>

<from>zhang san</from>

<to>li si</to>

<content>别忘了还钱</content>

</email>

DTD文件：email.dtd

<!ELEMENT email (from,to,content)>

<!ELEMENT from (#PCDATA)>

## 4、元素的定义

语法：<!ELEMENT NAME TYPE>

ELEMENT:关键字 NAME：元素名 TYPE:元素类型

元素类型：

1）EMPTY：空元素，该元素不能包含子元素和文本，但是可以有属性

2）ANY：该元素可以包含任何在DTD中定义的元素内容。（很少使用）

3）#PCDATA：可以包含任何字符数据，但是不能在其中包含任何子元素。

4）纯元素类型：只包含子元素，并且这些子元素外没有文本。

5）混合类型：包含子元素和文本数据的混合体。

举例1：空元素

<!ELEMENT student EMPTY>

<students><student id=”001” name=”张三” gender=”male” /></students>

<student>张三</student>错误的

<student><name>张三</name></student>错误

举例2：ANY

<!ELEMENT student ANY >

<students><student id=”001” name=”张三” gender=”male” /></students>

<student>张三</student>正确

<student><name>张三</name></student>正确

注意：将根元素定义为ANY后，子元素出现的次数跟顺序将不受限制。

举例3：#PCDATA

<!ELEMENT student (#PCDATA) >

<student id=”001” name=”张三” **gender=”male” />错误**

<student>张三</student>正确

<student><name>张三</name></student>错误

举例4：组合类型

<!ELEMENT person (name,brother+,sister\*)>

<person>

<name>zhang san</name>

<brother>张一</brother>

<brother>张二</brother>

<sister>张美美</sister>

</person>

## 5、DTD中的修饰符

|  |  |  |  |
| --- | --- | --- | --- |
| 符号 | 用途 | 示例 | 示例说明 |
| () | 用来给元素分组 | (name|gender|age),(like|hate),address | 分三组 |
| | | 在列出的内容中选择一个 | (male|female) | Male或者female必须出现，两者选择一个 |
| + | 该对象至少出现一次，可以出现多次 | (成员+) | 表示成员必须出现，可以出现多次 |
| \* | 允许出现0次到多次 | (成员\*) | 成员可以出现0到多次 |
| ？ | 可以出现，但最多出现一次 | (成员?) | 成员可以出现0到1次 |
| ， | 对象必须按照指定的顺序出现 | (name,gender,age) | 表示name,gender,age必须出现，并且按指定的顺序出现。 |

举例：<!ELEMENT student ((name|gender|age),(like|hate),address)>

<student>

<name>zhangsan</name>

<like>money</like>

<address>XXX省XXX市</address>

</student>

## 6、属性的定义 （属性类型）

语法：<!ATTLIST 元素名称

属性名称 类型 属性特点

属性名称 类型 属性特点

属性名称 类型 属性特点

>

类型：CDATA、ID、IDREF/IDREFS、NMTOKEN/NMTOKENS、Enumerated、ENTITY/ENTITIES

属性特点：#REQUIRED(必须的)、#IMPLIED(可有可无的)、#FIXED、default

举例1：CDATA，属性值可以是任何字符。（包括数字和中文）

<!ATTLIST student id CDATA #REQUIRED>

<student id=”001”>

<!ATTLIST student id CDATA #IMPLIED>

<student/>

注意：CDATA与CDATA节的区别，CDATA与#PCDATA的区别

举例2：NMTOKEN/NMTOKENS

NMTOKEN是CDATA的一个子集，表示属性值必须是：英文字母、数字、句号、破折号、下划线、冒号，属性值不能含有空值。

NMTOKENS与NMTOKEN类似，包含多个由空格分隔的字符。

<!ATTLIST student name NMTOKENS #REQUIRED>

<student name=”zhang san”/>

注意：换成CDATA、NMTOKENS是可以的，NMTOKEN的属性值前后可以有空格。

举例3：ID表明该属性的值是唯一的

<!ATTLIST student

id ID #REQUIRED

name CDATA #REQUIRED

>

<student id=”s1” name=”zhangsan”/>

<student id=”s2” name=”lisi”/>

举例4：IDREF/IDREFS，IDREF属性的值指向文档中其它地方已经出现过的ID类型的值。IDREFS同IDREF，是可以具有由空格分开的多个引用

<!ELEMENT persons (person+)>

<!ELEMENT person EMPTY>

<!ATTLIST person

id ID #REQUEIRED

parend\_id IDREF #IMPLIED

name CDATA #REQUIRED

>

<persons>

<person id=”p\_1” name=”父亲”/>

<person id=”p\_2” name=”母亲” />

<person id=”p\_3” name=”孩子” parend \_id=”p\_1,p\_2” />

</persons>

举例5：Enumerated(不是属性类型值，而是一种描述)，事先定义好一些值，属性的值必须在所列出的值的范围内。

<!ATTLIST person

marryStuts (single|married|divorced|widowed) #IMPLIED

gender (male|female) #REQUIRED

>

<person marryStuts=”single” gender=”male” />

## 7、属性的定义（属性特点）

举例1：#REQUIRED,元素的所有实例都必须有该属性的值，不能为空(NOT NULL)

语法：<!ATTLIST 元素名 属性名 属性类型 #REQUIRED>

<!ATTLIST person name CDATA #REUQIRED>

<person name=”zhangsan” />

举例2：#IMPLIED，元素的所有实例可以忽略该属性的值，可以为空

语法：<!ATTLIST 元素名 属性名 属性类型 #IMPLIED>

<!ATTLIST person name CDATA #IMPLIED>

<person name=”zhangsan” />或者<person />

举例3：#FIXED value，元素实例中该属性的值必须为指定的固定值。

语法：<!ATTLIST 元素名 属性名 类型 #FIXED “value”>

<!ATTLIST student classNo CDATA #FIXED “201102”>

<student classNo=”201102” />

举例4：default(不是属性特点值，而是一种描述)，为属性提供一个默认值。

语法：<!ATTLIST 元素名 属性名 类型 “默认值”>

<!ATTLIST student classNo CDATA “201102”>

<student classNo=”201103”>

注意：提供的默认值，并非一定是该值。

## 8、定义实体

1）内部实体

语法：<!ENTITY 实体名 “实体值”>

示例：<!ENTITY name “张三”>

<person>

<name>&name;</name>

</person>

2）外部实体（仅限于了解）

语法：<!ENTITY 实体名 SYSTEM “URI/URL”>

示例：<!ENTITY name SYSTEM “http://www.w3schools.com/entities/entities.xml”>

<person>

<name>&name;</name>

</person>

3）实体类型：普通实体、外部实体、参数实体、外部参数实体

|  |  |  |  |
| --- | --- | --- | --- |
| 类型 | | 普通实体 | 参数实体 |
| 使用场合 | | 用在XML文档中 | 只用在DTD中元素和属性的声明中 |
| 声  明  方  式 | 内部 | <!ENTITY 实体名 “文本”> | <!ENTITY % 实体名“文本”> |
| 外部 | <!ENTITY 实体名 SYSTEM “URI/URL”> | <!ENTITY % 实体名 SYSTEM “URI/URL”> |
| 引用方式 | | &实体名; | % 实体名; |

举例1：普通实体

<!ATTLIST student

name CDATA #REQUIRED

>

<!ENTITY nameValue “张三”>

<student name=”&nameValue;” />

举例2：参数实体

<!ENTITY % arddress (城市,区县,街道)>

<!ELEMENT linkMan (name,tel,% arddress)>

等同于<!ELEMENT linkMan (name,tel,城市,区县,街道)>

## 9、分析DTD文档，写出XML实例

<!ELEMENT newspaper (article+)>

<!ELEMENT article (heading,byline,lead,body,notes)>

<!ELEMENT heading (#PCDATA)>

<!ELEMENT byline (#PCDATA)>

<!ELEMENT lead (#PCDATA)>

<!ELEMENT body (#PCDATA)>

<!ELEMENT notes (#PCDATA)>

<!ATTLIST article author CDATA #REQUIRED>

<!ATTLIST article editor CDATA #IMPLIED >

<!ATTLIST article date CDATA #IMPLIED >

<!ATTLIST article edition CDATA #IMPLIED >

## 10、分析XML，写出DTD文档

<?xml verison=”1.0”?>

<students>

<student id=”s1”>

<name>zhang san</name>

<age>21</age>

<gender>male</gender>

</student>

<student id=”s2”>

<name>lisi</name>

<age>22</age>

<gender>female</gender>

</student>

</student>

## 命名空间（namespace）

* 为什么使用命名空间？主要目的就是为了解决元素冲突问题。
* 什么是命名空间？
* 使用URL作为XML的命名空间
* 命名空间的语法定义：xmlns:[prefx]=”URL”
* 元素或属性都可以使用命名空间

1. 命名冲突

XML的元素名是不固定的，当两个不同的文档使用同样的名称描述两个不同类型的元素的时候，就会发生命名冲突。

例如：<table> <table>

<tr> <name>红木家具</name>

<td>C1单元格</td> <width>200</width>

<td>C2单元格</td> <length>100</length>

</tr> </table>

</table>

虽然命名冲突很少发生，但是即使是这样当我们把两个table放在一起时，也需要一种机制能够明确的将它们进行区分开来。

使用前缀来解决命名冲突问题

<h:table> <x:table>

<h:tr> <x:name>红木家具</x:name>

<h:td>C1单元格</h:td> <x:width>200</x:width>

<h:td>C2单元格</h:td> <x:length>100</x:length>

</h:tr> <x:/table>

</h:table>

讲解：在左边的table标签中，我们加入了h:这样一个标识，“:”前面的h表示命名空间前缀，表示table标签，tr标签，td是在h这个命名空间下的，同样，右面的table是在x命名空间下的。与java中的包名含义类似，可以理解为java包的概念。

那么命名空间h/x是怎样来的呢？

使用命名空间属性

<h:table xmlns:h=”http://www.w3.org /TR/html4/”>

<h:tr>

<h:td>C1单元格</h:td>

<h:td>C2单元格</h:td>

</h:tr>

</h:table>

<x:table xmlns:x=”http://www.w3schools.com/furniture”>

<x:name>红木家具</x:name>

<x:width>200</x:width>

<x:length>100</x:length>

<x:/table>

## 总结

* 一个有效的XML文档必然是一个格式正规的，结构正规的XML文档并不一定是有效的。
* DTD包含一套用来描述并限制XML文档结构的语法规则。

——元素定义的规则

——元素之间的关系规则

——属性的定义规则

——可使用的实体或符号规则

* 命名空间用于避免元素命名冲突
* 在XML中，使用全球唯一的URL作为命名空间

XML与DTD总结

1、XML（Extensible Markup Lanuage）可扩展的标记语言。

2、DTD（Document Type Definition）dtd文档类型定义。

3、xml文档中的标记都是成对出现的，有开始就有结束，先开始的标记要先结束。

4、xml文档都是有且只有一个根元素，所谓根元素就是唯一个包含了其他所有元素的元素。

5、xml描述的是文档的内容与语义，而不是文档应当如何显示。

6、xml可以实现异构语言、异构平台的交互。（例如：java平台与.net平台的交互）

7、xml的指令<?，指令必须顶格写，不允许出现空格。

8、xml的元素可以有属性 属性的方式是 名称+“=“+value的方式。例如：name=”张三” value必须要加上单引号或者双引号，多个属性要以空格分开。

9、在一个元素上，相同的属性只能出现一次。

10、格式正规(well formed)的XML文档。

11、格式有效(valid)的XML文档，首先是格式正规的然后能够通过DTD的格式校验才能成为格式有效的XML文档。

10、DTD包含一套用来描述并限制xml文档结构的语法规则，

——元素的定义规则

——元素之间的关系规则

——属性的定义规则

——可使用的实体与符号规则

# Schema详解

Schema与DTD要做的事情都是一样的，都是用来校验xml文档的有效性。但是他们本身所提供的功能性是差别相当大的。

如果把DTD比作数组的话，那么Schema就相当于集合。集合的功能要比数组的更加强大，同样Schema的功能要比DTD更加强大。

学好Schema只需要理解以下三点：

1. 理解Schema的数据类型
2. 理解Schema的元素类型
3. 理解验证与约束

## 什么是Schema

1. Xml Schema是用一套预先定义的xml元素和属性创建的，这些元素定义xml文档的机构和内容模式。所以从根本上说Schema就是标准的XML文档。
2. Xml Schema规定了XML文档实例的结构和每个元素或属性的数据类型。
3. 比较：<student>

<id>001</id><name>john</name> ——XML

</student>

<!ELEMENT student (id,name)>

<!ELEMENT id (#PCDATA)> ——DTD

<!ELEMENT name (#PCDATA)>

<xs:complexType name=”myStudent”>

<xs:sequence>

<xs:element name=”id” type=”xs:string”/>

<xs:element name=”name” type=”xs:string”/> ——Schema

</xs:sequence>

</xs:complexType>

<xs:element name=”student” type=”myStudent”/>

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## 为什么用Schema

DTD的局限性

——DTD不遵循XML的语法，它有自己的一套语法

——DTD数据类型有限，只有PCDATA（可看做字符串）

——DTD不可扩展

——DTD不支持命名空间

Schema的新特性

——Schema基于xml语法

——Schema可以用能处理XML文档的工具来处理

——Schema大大扩充了数据类型，可以自定义数据类型

——Schema支持元素的继承

——Schema支持属性组

（进行SchemaXML文档）

## Schema的文档结构

<?xml version=”1.0”?>

<xs:schema xmlns:xs=”<http://www.w3.org/2001/XMLSchema>”

targetNamespace=”http://mynamespace/myschema” >

</xs:schema>

所有的Schema文件都是以schema元素为根元素的

用于构造Schema的元素和数据类型都来自于<http://www.w3.org/2001.XMLSchema>命名

空间，此命名空间不能改变。

targetNamespace:本schema文档定义的元素和数据类型属于这个命名空间<http://mynamespace/myschema>，自己命名的空间可以改变。

## Schema的数据类型

1. 简单类型

—内置数据类型

基本数据类型 扩展的数据类型

—用户自定义数据类型（通过SimpleType来定义）

1. 复杂的数据类型（通过complexType）

基本数据类型

string 字符串

boolean 布尔型

decimal 代表特定精度的数字

float 代表单精度32位浮点数

double 代表双精度64位浮点数

duration 表示持续时间

datatime 表示特定的时间

time 表示特定的时间，但是是每天重复的

date 代表日期

hexBinary 代表十六进制

anyURI 代表一个URI，用来定位文件

NOTATION 代表NOTATION类型

扩展的数据类型

ID 用于唯一标识元素

IDREF 参考ID类型的元素或属性

ENTITY 实体类型

NMTOKEN NMTOKEN类型

NMTOKENS NMTOKENS类型

long 表示整型数，大小介于-9223372036854775808和

9223372036854775807之间

short 表示整型数，大小介于-32768和32767之间

byte 表示整型数，大小介于-128和127之间

数据类型的特性(类似属性)

enumeration 在指定的数据集中选择，限定用户取值

fractionDigits 限定最大的小数位，用于控制精度

length 用于指定数据的长度

maxExclusive 指定数据的最大值（小于）

maxInclusive 指定数据的最大值（小于等于）

minExclusive 指定数据的最小值（大于）

minInclusive 指定数据的最小值（大于等于）

minLength 指定最小长度

pattern 指定数据的显示规范

## Schema的元素的类型

schema element attribute group attributeGroup simpleType simpleContent

complexType choice list union sequence restriction

### schema元素

作用：包含已经定义的schema元素

用法：<xs:schema>

属性：xmlns targetNamespace(可以定义也可以不定义)

#### element元素

作用：声明一个元素

属性：name type ref minOccurs maxOccurs substitutionGroup fixed default

示例：

<xs:element name=”cat” type=”xs:string”/>

<xs:element name=”dog” type=”xs:string”/>

<xs:element name=”pets”>

<xs:complexType>

<xs:sequence minOccurs=”1” maxOccurs=”unbounded”>

<xs:element ref=”cat”/>

<xs:element ref=”dog”/>

</xs:sequence>

</xs:complexType>

</xs:element>

substitutionGroup：代替的意思，可以将当前元素出现的位置，替换成其它的指定的元素。（仅作了解）

实例：

<xs:element name=”dog” type=”xs:string”/>

<xs:element name=”redDog” type=”xs:string” substitutionGroup=”dog” />

Xml:

<dog>one</dog>

<redDog>two</redDog>或者<dog>two</dog>

#### group元素

作用：把一组元素声明在一起，以便他们能够一起被复杂数据类型使用。

属性：name ref

示例：<xs:element name=”thing1” type=”xs:string”/>

<xs:element name=”thing2” type=”xs:string”/>

<xs:attribute name=”myAttribute” type=”xs:decimal”/>

<xs:group name=”myGroupWithThing”>

<xs:sequence>

<xs:element ref=”thing1”/>

<xs:element ref=”thing2”/>

</xs:sequence>

</xs:group>

<xs:complexType name=”myCompleType”>

<xs:group ref=”myGroupWithThing”/>

<xs:attribute ref=”myAttribute”/>

</xs:complexType>

<xs:element name=”abs” type=” myCompleType”/>

#### Attribute元素

作用：声明一个属性

属性：name type ref use

示例：

<xs:complexType name=”myComplexType”>

<xs:attribute name=”length” type=”xs:integer” use=”required”/>

</xs:complexType?

#### attributeGroup元素

作用：把一组属性声明组合在一起，以便被复合类型使用

属性：name ref

示例：

<xs:attributeGroup name=”attributes”>

<xs:attribute name=”length” type=”xs:integer”/>

<xs:attribute name=”widht” type=”xs:integer”/>

<xs:attribute name=”color” type=”xs:string”/>

</xs:attributeGroup>

<xs:complexType name=”myComplexType”>

<xs:attributeGroup ref=” attributes”/>

<xs:complexType>

#### simpleType元素

作用：定义一个简单类型，它决定了元素和属性的值的约束和相关信息

属性：name

内容：应用已经存在的简单类型，三种方式1）、restriction，限定一个范围2）、list组合成列表3）、union包含一个值结合。

示例：1、restriction

<xs:simpleType name=”mySimpleType”>

<xs:restriction base=”xs:integer”>

<xs:minExClusive value=”0”/> 类似的minExClusive maxExClusive

<xs:maxInclusive value=”100”/>

</xs:restriction >

</xs:simpleType>

<xs:element name=”length” type=”mySimpleType”/>

示例：2、list

从一个特定数据类型的基础上创建一个集合类型的简单类型元素。

<xs:simpleType name=”mySimpleType”>

<xs:list itemType=”xs:date”/>

</xs:simpleType>

<xs:element name=”date” type=”mySimpleType”/>

<date>2010-12-12 2012-11-11</date>

示例：3、union

将几个简单数据类型元素进行组合生成组合类型，然后定义新的简单类型元素为组合类型，组合类型的值可以在被组合的简单数据类型的所有值中进行取值。

<xs:simpleType name=”mySimpleType”>

<xs:union memberTypes=”bike1 bike2”/>

</xs:simpleType>

<xs:simpleType name=”bike1”>

<xs:restriction base=”xs:integer”>

<xs:enumeration value=”35”/>

<xs:enumeration value=”40”/>

<xs:enumeration value=”45”/>

</xs:restriction>

</xs:simpleType>

<xs:simpleType name=”bike2”>

<xs:restriction base=”xs:string”>

<xs:enmeration value=”small”/>

<xs:enmeration value=”medium”/>

<xs:enmeration value=”large”/>

</xs:restriction>

</xs:simpleType>

<xs:element name=”like” type=”mySimpleType”/>

#### complexType

作用：定义了一个复合元素，它决定了一个组元素和属性值的约束和相关信息。

属性：name

示例：

complexType跟simpleType的区别

simpleType类型的元素中不能包含元素或者属性

当需要声明一个子元素或者属性时，需要使用complexType。

当需要基于内置的基本数据类型定义一个新的数据类型时，使用simpleType。

1. simpleContent

作用：应用于complexType，对它的内容进行扩展和约束，它是没有子元素，只有属性的

示例：

<xs:complexType name=”myComplex”>

<xs:simpleContent>

<xs:extension base=”xs:string”>

<xs:attribute name=”bgColor” use=”required”>

<xs:simpleType>

<xs:restriction base="xs:string">

<xs:enumeration value="#CCFF00"/>

<xs:enumeration value="#CCFF01"/>

<xs:enumeration value="#CCFF02"/>

</xs:restriction>

</xs:simpleType>

</xs:attribute>

</xs:extentsion>

</xs:simpleContent>

</xs:complexType>

<xs:element name=”length” type=” myComplex”/>

<length bgColor=” #CCFF01”>100</length>

<label color=”red”>标签</label> red blue green

#### choice元素

作用：允许唯一一个元素从一个组中被选中

属性：minOccurs/maxOccurs

示例：

<xs:element name="dog" type="xs:string"/>

<xs:element name="cat" type="xs:string"/>

<xs:complexType name="myChoice">

<xs:choice minOccurs="1" maxOccurs="3">

<xs:element ref="dog"/>

<xs:element ref="cat"/>

<xs:element name="mouse" type="xs:string"/>

<xs:element name="chicken" type="xs:string"/>

</xs:choice>

</xs:complexType>

<xs:element name=”dongwu” type=”myChoice”/>

< dongwu>

<dog>小狗</dog>

<dog>小狗</dog>

<cat>小猫</cat>

</dongwu>

#### **sequence元素**

作用：给一组元素一个特定的序列

属性：

示例：

<xs:complexType name=”myComplex”>

<xs:sequence minOccus=”1” maxOccus=”unbounded”>

<xs:element name=”dog” type=”xs:string”/>

<xs:element name=”cat” type=”xs:string”/>

</xs:sequence>

</xs:complexType>

<xs:element name=”pets” type=”myComplex”/>

注意：sequence中的minOccus和maxOccus是整体最少出现一次，或最多出现几次，choice中的minOccus和maxOccus是在这一组整体最少出现一次，或最多出现几次，但是在choice中选择其中一个记录为内容。

用schema的数据及元素类型来声明xml文档的元素和属性

—声明元素:<xs:element>

—声明属性:<xs:attribute>

—通过DOCTYPE可以明确指定文档的根元素，通过schema是无法明确指定目标xml文档的根元素的，只有用户可以指定。XMLSpy是通过根据哪个元素包含了其他的元素，通过包含最多的元素作为xml文档的根元素。

分析XML实例，书写schema

<newspaper>

<article author=”作者” editor=”” date=”” edition=””>

<headline>主标题</headline>

<byline>副标题</byline>

<lead>引文</lead>

<body>文章主体</body>

<notes>评论</notes>

</article>

<article author=”作者” editor=”” date=”” edition=””>

<headline>主标题</headline>

<byline>副标题</byline>

<lead>引文</lead>

<body>文章主体</body>

<notes>评论</notes>

</article>

<article author=”作者” editor=”” date=”” edition=””>

<headline>主标题</headline>

<byline>副标题</byline>

<lead>引文</lead>

<body>文章主体</body>

<notes>评论</notes>

</article>

</newspaper>

总结：schema是另一种文档类型定义，它符合xml文档语言规范。

Schema是可扩展的，它支持命名规范。

Schema支持更多的数据类型与元素类型。

Schema用element来声明元素，用attribute来声明属性。

Schema用simpleType来定义简单类型，用complexType来定义复杂类型。

# XMLSchema学习总结

## 简单类型

1. 简单元素：只能包含文本内容不能包含元素，也不能包含属性的元素

例子：

<xs:element name=”name” type=”xs:string”/>

<xs:element age=”age” type=”xs:integer”/>

1. 属性：所有的元素类型均被声明为简单类型，只有复杂类型的元素才能拥有属性

例子：

<xs:attribute name=”lang” type=”xs:string”/>

所有的属性都是默认可选的，我们可以使用use来指定属性是必须的还是可选的

例子：

<xs:attribute name=”lang” type=”xs:string” use=”optional”/>

<xs:attribute name=”lang” type=”xs:string” use=”requried”/>

我们可以通过使用default或fixed为简单类型（简单元素、属性）来指定默认值或固定值，例子：

<xs:element name=”color” type=”xs:string” default=”red”/>

<xs:attribute name=”lang” type=”xs:string” fixed=”CN”/>

对简单类型的约束

|  |  |
| --- | --- |
| 约束 | 含义 |
| enumeration | 定义允许值的枚举 |
| fractionDigits | 指定最多允许的小数位数（必须大于0或等于0） |
| length | 精确指定允许的最大的字符长度 |
| maxLength | 指定允许最大的字符长度（必须大于等于0） |
| minLength | 指定允许最小的字符长度 |
| maxExclusive | 指定允许的最大数值，必须小于该值 |
| maxInclusive | 指定允许的最大数值，必须小于或等于该值 |
| minExclusive | 指定允许的最小数值，必须大于该值 |
| minInclusive | 指定允许的最小数值，必须大于或等于该值 |
| pattern | 指定允许值的模式，类似正则表达式 |
| totalDigits | 精确指定数字个数 |
| whiteSpace | 处理空白（保留:perserve,替换:replace,合并:collapse） |

## 复杂类型

复杂类型指包含了其他元素或属性的元素类型

例子：

<message>

<to>A</to>

<from>B</from>

<content>hello,how are you</content>

</message>

在上面例子中，message就是一个复杂类型的元素，在schema中的描述为

<xs:element name=”message”>

<xs:complexType>

<xs:sequence>

<xs:element name=”to” type=”xs:string”/>

<xs:element name=”from” type=”xs:string”/>

<xs:element name=”content” type=”xs:string”/>

</xs:sequence>

</xs:complexType>

</xs:element>

注意：元素to from content包含在<xs:sequence></xs:sequence>中，表明这些元素必须按照定义的顺序出现在XML文件中。

Message元素也可以包含一个type属性，指定我们定义的复杂类型

<xs:element name=”message” type=”myMessage”/>

<xs:complexType name=”myMessage”>

<xs:sequence>

<xs:element name=”to” type=”xs:string”/>

<xs:element name=”from” type=”xs:string”/>

<xs:element name=”content” type=”xs:string”/>

</xs:sequence

</xs:complexType>

复杂类型和简单类型之间最根本的区别就是：复杂类型的内容可以包含其他元素也可以带有属性，简单元素类型不能包含子元素，也不能带有属性。

## Schema综述

1. 如何描述空元素，例如：<product prodId=”1212”/>

因为是空元素，不包含子元素，同时由于包含属性，用attribute属性定义

<xs:element name=”product”>

<xs:complexType>

<xs:attribute name=”prodId” type=”xs:positiveInteger”/>

</xs:complexType>

</xs:element>

也可以这样

<xs:element name=”product” type=”myProdcut”/>

<xs:complexType name=” myProdcut”>

<xs:attribute name=”prodId” type=”xs:positiveInteger”/>

</xs:complexType>

1. 如何描述只有含有简单内容（文本/属性）的元素，例如：

<message date=”2010-12-10”>Hi,My Girl</xs:message>

由于只包含简单内容，所以我们在元素内容定义的外面使用simpleContent指出，当描述简单内容的时候，我们需要在简单内容里面使用extension或者restriction来描述内容的数据类型。

<xs:element name=”message”>

<xs:complexType>

<xs:simpleContent>

<xs:extension base=”xs:string”/>

<xs:attribute name=”date” type=”xs:date”/>

</xs:simpleContent>

</xs:complexType>

</xs:element>

1. 如何定义混合类型，例如：

<message>

The message from<from>Alex</from>

</message>

message元素除了包含子元素from之外还包含文本信息the message from针对这种情况我们需要在complexType中使用mixed=”true”指出。

如：

<xs:element name=”message” mixed=”true”>

<xs:complexType>

<xs:element name=”from” type=”xs:string”/>

</xs:complexType>

</xs:element>

当然如果包含更多的子元素，我们需要使用<xs:sequence></xs:sequence>来限定元素的顺序。

## 在XML Schema中，共有3类7种指示器（Indicator）

1. 定义元素如何出现：all sequence choice
2. all，默认值。每个子元素必须出现且只能出现一次。例如：

<xs:element name=”person”>

<xs:complexType>

<xs:all>

<xs:element name=”firstName” type=”xs:string”/>

<xs:element name=”lastName” type=”xs:string”/>

</xs:all>

</xs:complexType>

</xs:element>

1. sequence：子元素按照定义的顺序出现。
2. choice：两个或多个子元素中，只能出现一个。例如：

<xs:element name=”gender”>

<xs:complexType>

<xs:choice>

<xs:element name=”male” type=”male”/>

<xs:element name=”womale” type=”womale”/>

<xs:choice>

</xs:complexType>

</xs:element>

1. 限定次数类：minOccurs,maxOccurs指定最少或最多出现次数，例如：

<xs:element name=”person”>

<xs:complexType>

<xs:sequence minOccurs=”1” maxOccurs=”3”>

<xs:element name=”people” type=”people”/>

</xs:sequence>

</xs:complexType>

</xs:element>

1. 组限定：group、attributeGroup用来定义一组相关的元素，比如：

<xs:group name=”personGroup”>

<xs:sequence>

<xs:element name=”name” type=”xs:string”/>

<xs:element name=”gender” type=”xs:string”/>

<xs:element name=”age” type=”xs:integer”/>

</xs:sequence>

</xs:group>

-----------------------------------------------------------------------------------

<xs:attributeGroup name=”myAttrGroup”>

<xs:sequence>

<xs:attribute name=”date” type=”xs:date”/>

<xs:attribute name=”id” type=”xs:string”/>

</xs:sequence>

</xs:attributeGroup>

# DOM方式解析XML

JavaScript解析XML文件

1. 实例讲解

function check(){

var docXml;

if (window.ActiveXObject) {

docXml = new ActiveXObject("Microsoft.XMLDOM");

}else {

docXml = document.implementation.createDocument("", "", null);

}

docXml.async = false;

docXml.load("./student.xml");

//docXml.async = false;

//获取标签名为student的节点，返回对象为数组

var student = docXml.getElementsByTagName("student")[0];

//获取属性节点

var studAttri = student.attributes;

var aaValue = studAttri[0].value;

//遍历孩子内容

for(var i=0;i<student.childNodes.length;i++){

var node = student.childNodes[i];

alert(node.tagName +"=="+node.text);

}

}

Document：JS的文档对象

Implementation属性：实现的接口

createDocument方法：创建一个xml的dom对象，其中三个参数，第一个xml对应的命名空间，第二个xml的根节点元素，第三个doc对象的类型，三个参数对我们解析xml来说没什么作用，所以一般情况下都是前两个传空字符串，最后一个传null。

Xml文件

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

<student aa="bb">

<name id="001">zhang san</name>

<age id="002">21</age>

</student>

第一步：获取根元素节点，有两种方式

1. 根据根元素节点名称

var rootElement =docXml.getElementsByTagName("student")[0];

返回一个Element元素数组对象。

1. 直接使用dom对象的documentElement属性。

var rootElement = docXml. documentElement;返回的是根元素节点对象

第二步：解析下面的元素

1. 遍历根元素

var length = rootElement. childNodes.length;

for(var i=0;i<length;i++){

var element = rootElement.childNodes.item(i); // rootElement.childNodes[i]

var id = element.getAttribute("id");

var tagName = element.tagName;

var text = element.text;

alert(id+"=="+tagName+"=="+text);

}

//判断当前节点是否为Element

Function isElement (element){

if(element.nodeType==1){

return true;

}else{

return false;

}

}

rootElement.childNodes返回所有子元素对象的一个数组

获取数组中元素使用item(index)方法。

获取当期元素的属性使用getAttribute(“attributeName”)

或者获取属性组element.attributes;

获取元素名称使用：element.tagName;

获取元素中的文本使用:element.text;

2）、解析任意类型XML

var docXml;

var xmlString = "";

function check(){

if (document.implementation.createDocument) {

docXml = document.implementation.createDocument("", "", null);

}else if (window.ActiveXObject) {

docXml = new ActiveXObject("Microsoft.XMLDOM");

}

docXml.load("./xmlFile/student.xml");

//获取标签名为student的节点，返回对象为数组

var rootElement = docXml.documentElement;

xmlString = "<?xml version=\"1.0\"?>";

dealElement(rootElement);

document.getElementById("xml").innerHTML=xmlString;

}

function dealElement(rootElement){

xmlString += "<";

var tagName = rootElement.tagName;

xmlString += tagName+" ";

var attributes = rootElement.attributes;

for(var i=0;i<attributes.length;i++){

var attr = attributes.item(i);

xmlString = xmlString+attr.nodeName+"=\""+attr.nodeValue+"\" ";

}

xmlString +=">";

for(var i=0;i<rootElement.childNodes.length;i++){

var node = rootElement.childNodes.item(i);

if(node.nodeType==1){

dealElement(node);

}else{

xmlString +=node.nodeValue;

}

}

xmlString +="</"+tagName+">";

}

## 简单工厂模式（与XML文档解析有关）

意图：简单工厂模式是类的创建模式，又叫做静态工厂方法(Static Factory Method)模式。简单工厂模式是由一个工厂对象来决定创建出哪一种产品的实例。通常它根据自变量的不同返回不同的实例。

实用性：简单工厂模式的实质是根据传入参数，动态的决定应该创建出哪种产品的实例。

简单工厂模式的构成：

工厂类（Creator）角色，担任这个角色的是工厂类模式的核心，含有与应用紧密相关的商业逻辑。工厂类在客户端的直接调用下创建产品对象，它往往由一个具体类来实现。

抽象产品（Product）角色，担任这个角色的类是简单工厂模式所创建的对象的父类，或他们共同拥有的接口。抽象产品角色可以由一个抽象类或者接口来实现。

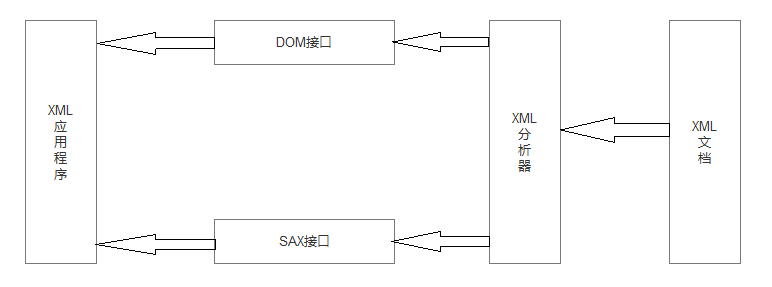
具体产品（Concrate）角色，简单工厂模式所创建的任何对象都是这个角色的实例，具体产品角色都由一个具体类实现。

具体示例：

## XML的解析方式有两种：DOM与SAX

W3C制定了一套书写XML分析器的标准接口规范—DOM。除此之外，XML DEV邮件列表中的成员根据应用的需求也自发的定义了一套对XML文档进行操作的接口规范—SAX。这两种规范各有侧重，互有长短，应用都比较广泛。

下面我们看下DOM和SAX在应用开发过程中所处地位的示意图。从图中可以看出，应用程序不是直接对XML文档进行操作的，而是由XML分析器首先对XML文档进行分析，然后应用程序通过XML分析器提供的DOM接口或者SAX接口对分析结果进行操作，从而间接的实现了对XML文档的访问。



## DOM

Dom的全称是Document Object Model，即文档对象模型。在应用程序中基于dom的XML分析器将一个XML文档转换成一个对象模型的集合，应用程序正是通过对这个对象模型的操作，来实现对XML文档数据的操作。通过DOM接口，应用程序可以任何时候访问XML文档中的任何一部分数据，因此这种利用DOM接口的机制也被称为随机访问机制。

Dom接口提供了一种通过分层对象模型来访问XML文档信息的方式，这些分层对象模型依据XML的文档结构形成了一个节点树。无论XML文档中所描述的是什么类型的信息，即便是制表数据、项目列表或一个文档，利用dom所生成的模型都是节点树的形式。也就是说dom强制使用树模型来访问xml文档中的信息。由于xml本质上就是一种分层结构，所以这种表述方法是相当有效的。

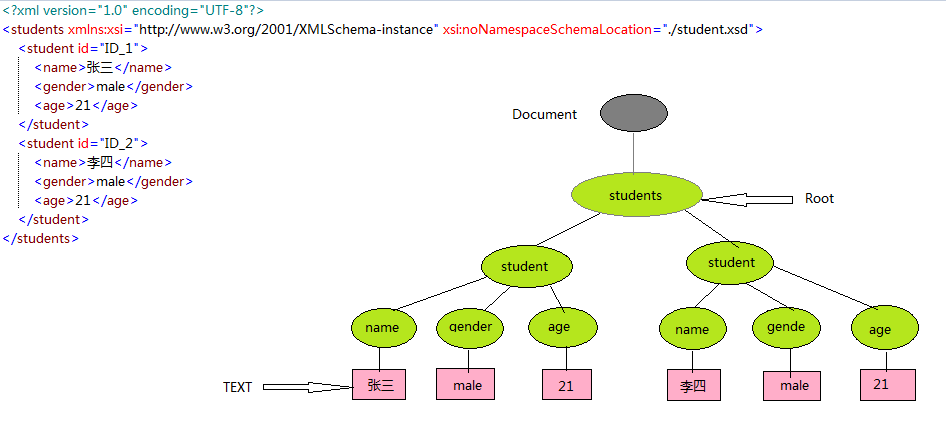
Dom树所提供的随机访问方式给应用程序开发带来了很大的灵活性，它可以任意的控制xml文档中的内容。然而，由于dom分析器把整个xml文档转换成dom树存放在内存中，因此当文档比较大结构比较复杂时，对内存的需求就比较高。而且对于机构复杂的树的遍历也是一种耗时的操作。所以dom分析器对机器性能的要求比较高。实现效率十分不理想。不过由于dom分析器所采用的树结构的思想与xml文档的结构想吻合，同时鉴于随访访问所带来的方便，因此dom分析器还是很有广泛的使用价值。

## Dom的组成

对于xml应用开发来说，dom就是一个对象化的xml数据接口，一个与语言无关，与平台无关的标准接口规范。它定义了HTML文档和XML文档的逻辑结构，给出了一种访问HTML文档和XML文档的方法。利用dom，程序开发人员可以动态的创建文档，遍历文档结构，添加、修改、删除文档的内容，改变文档的显示方式等等。可以这样说，文档代表的是数据，而dom则代表了如何去处理这些数据。无论是在浏览器里还是浏览器外，无论是在客户端还是服务器端，只要用到xml的地方就会碰到dom的应用。

作为W3C的标准接口规范，目前，dom由三部分组成，包括：核心（core）、HTML和XML。核心部分是结构化文档比较底层的对象的集合，这一部分所定义的对象已经完全可以表达出任何HTML和XML文档中的数据了。HTML接口和XML接口两部分则是专为操作具体的HTML文档和XML文档所提供的高级接口，使对这两类文档的操作更加方便。

## DOM树



Document是文档的根节点，students是文档的根元素节点，要严格区分根节点与根元素节点的区别，根节点代表的是xml文档本身是整个xml文档的入口，根元素节点则表示xml文档的根元素，它对应xml文档的root。

## 最常见的节点类型：

—元素：元素是xml的基本构件。典型的，元素可以有其他元素、文本节点或两者都有来作为其子节点。元素节点还是可以有属性的唯一类型的节点。

—属性：属性包含关于元素节点的信息，它不是元素的子节点。

—文本：它可以包含很多信息或者仅仅是空白。

—文档（根节点）：文档节点是文档中所有其他节点的父节点。

## 不常见的节点类型：

CDATA 、处理指令、注释

## DOM的四个基本接口

文档对象模型利用对象来把文档模型化，这些模型不仅描述了文档的结构，还定义了模型中对象的行为。换句话说，在上面给出的例子中，图中的节点不是数据节点，而是对象。对象中包含方法和属性，在DOM中对象模型要实现。

* 用来表示、操作文档的接口
* 接口的行为和属性
* 接口之间的关系和互操作。

在DOM接口规范中，有四个基本的接口：Document，Node，NodeList以及NamedNodeMap。在这四个接口中Document接口是对文档操作的入口，它是从Node接口继承过来的。Node接口是其他大多数接口的父类，像Document,Element,Attribute,

Text,Comment等接口都是从Node接口继承过来的。NodeList接口是一个节点的集合，它包含了某个节点中所有的子节点。NamedNodeMap接口也是一个节点的集合，通过该接口可以建立节点名与节点之间的一一映射关系，从而利用节点名可以直接访问特定的节点。

### Document接口

Document 接口表示整个 HTML 或 XML 文档。从概念上讲，它是文档树的根，并提供对文档数据的基本访问。

因为元素、文本节点、注释、处理指令等不能存在于 Document 的上下文之外，所以 Document 接口还包含所需的创建这些对象的工厂方法。所创建的 Node 对象具有 ownerDocument 属性，该属性将 Node 对象与创建这些对象时的上下文所属的 Document 关联起来。

### Node接口：

Node接口在整个DOM树中具有举足轻重的地位，DOM接口中有很大一部分接口是从Node接口中继承过来的，例如：Element，Attr，CDATASection等接口，都是从Node接口继承过来的。在DOM树中，Node接口代表了树中的一个节点。

### NodeList接口：

NodeList接口提供了对节点集合的抽象定义，它并不包含如何实现整个节点集的定义。NodeList表示有顺序关系的一组节点，比如某个节点的子节点序列。另外他还出现在一些方法的返回值中，如getElementsByTagName。

在DOM中，NodeList对象是“活”的，换句话说，对文档的改变会直接反映到相关的NodeList对象中。例如：如果通过DOM获得一个NodeList对象，该对象包含了某个Element节点的所有子节点集合，那么当通过DOM对Element节点进行操作（增加、删除、修改节点中的子节点）时，这些改变将会自动的反映到NodeList对象中去，而不需要DOM应用程序再做额外的其他操作。

NodeList中的每一个item都可以通过索引来访问，索引值从0开始。

### NamedNodeMap接口：

实现了NamedNodeMap接口的对象中包含了可以通过名字来访问的一组节点的集合。不过要注意的一点是，NamedNodeMap并不是从NodeList继承过来的，它所包含的节点集中的节点是无序的。尽管这些节点也可以通过索引来进行访问，但这只是提供了枚举

NamedNodeMap中所包含节点的一种简单的访问方法。并不表明在DOM规范中为NamedNodeMap中的节点规定了一种排列顺序。

NamedNodeMap表示的是一组节点和其唯一名字的一一对应关系，这个接口主要用在属性节点的表示上。

与NodeList相同的是，在DOM中NamedNodeMap对象是“活”的。

* XML解析器实际上就是一段代码，它读入一个XML文档并分析其结构。
* 分类：

—带校验的解析器

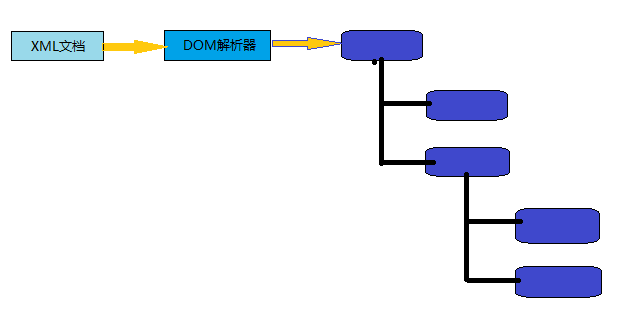
—不带校验的解析器（效率高）

—支持DOM的解析器（W3C的官方标准）

—支持SAX的解析器（事实上的工业标准）

* 文档对象模型
* 通过解析XML文档，为XML文档在逻辑上建立一个树模型，树的节点是一个个的对象

通过存取这些对象就能能够存取XML文档的内容



## Dom详解

DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();

我们使用DocumentBuilderFactory的目的是为了创建于具体的与解析器无关的程序，当DocumentBuilderFactory类的静态方法newInstance()被调用时，它根据系统变量来决定具体使用哪一个解析器。又因为所有的解析器都服从于JAXP(Java API for XML Parsing)所定义的接口，所以无论具体使用哪个解析器，代码都是一样的。所以当不同的解析器之间进行切换时，只需要改变系统变量的值，而不需要改变任何代码。这就是工厂所带来的好处。

DocumentBuilder db = dbf.newDocumentBuilder()，当获得一个工厂对象后，可以使用它的静态方法来获取一个DocumentBuilder对象，这个对象代表了具体的DOM解析器。但具体是哪一种解析器对于程序而言不重要。

获得解析器后，我们可以利用这个解析器对XML文档进行解析，Document doc = db.parse(new File(“./student.xml”))。DocumentBuilder的parse()方法接收一个XML文档名作为参数，返回一个Document对象，这个Document对象就代表了一个XML文档的树模型。以后所有的操作就与解析器无关了，直接在这个Document对象上进行操作。而对具体的Document对象

的操作就是有DOM所定义的了。

使用Document对象的getElementsByTagName()方法，我们可以得到一个NodeList对象，一个Node对象代表了一个XML文档中的标签元素，而NodeList对象代表的是一个Node对象的列表。

<students>

<student>zhang san</student>

<student>li si</student>

</students>

NodeList list = doc.getElementsByTagName(“student”);我们通过这行代码得到的是XML文档中所有<student>标签对应的Node对象的一个列表。然后，我们可以使用NodeList对象的item方法来得到列表中的每一个Node对象，Node node = list.item(i);

当一个Node对象被建立后，保存在XML文档中的数据就被提取出来，并封装在这个Node对象中了。提取<student>标签中的内容，我们一般使用Node对象的getNodeValue()或者getTextContent()方法来获取值。String content = node.getFirstChild().getNodeValue()或者node.getTextContent();

注意：在使用getNodeValue方法来获取值的时候，我们前面还使用了getFirstChild()这个方法，来获取student标签下的第一个Node对象。虽然在student标签下面已经没有子元素，只有文本内容了，但是我们还是使用了getFirstChild()的方法，这是与W3C对DOM的定义有关，W3C把标签内的文本也定义成了一个Node对象，所以先要获取文本对象的Node节点，才能获取里面的文本值，而getTextContent()则不需要了，因为它表示的就是获取这个元素下的所有文本内容。

DOM的基本对象有5个：Document、Node、NodeList、Element、Attr，Document对象代表了整个XML文档，所有其他的Node都以一定的顺序包含在Document对象内，排列成一个树形的结构。程序可以通过遍历这棵树来获取到XML文档的所有内容。这也是对XML文档操作的起点。我们总是先通过解析XML文档而得到一个Document对象，然后再开始后续的操作。此外Document对象还包含了创建其他节点的方法，如doc.createElement()来创建一个元素对象。它所包含的主要方法有：

createAttribute(String name)：用给定的参数名来创建一个Attr对象，并可以在其后面使用setAttributeNode(Attr)方法来放置在一个Element对象上面。

createElement(String name)：用给定的标签名来创建一个Element对象，代表XML文档中的一个标签，然后就可以在这个Element对象上添加属性或进行其他操作。

createTextNode(String)：用给定的字符串创建一个Text文本节点对象，Text对象代表了标签或者属性中所包含的纯文本字符串。如果在一个标签内没有其他标签，那么标签内的文本所代表的的Text对象就是这个Element对象的唯一子对象。

Node对象是DOM结构中最为基本的对象，代表了文档中的一个抽象节点。在实际使用的时候，很少会用到Node这个对象，而是使用Element、Text、Attr等Node的子对象来操作XML文档，Node对象为这些对象提供了一个抽象的公共的根。虽然在Node对象中定义了对其子节点进行存取的方法，但是有一些Node子对象，比如Text对象。它并存在子节点，这一点需要注意。

## 具体实例（解析）

//step 1：获得DOM解析器工厂，（工厂的作用是用于创建具体的解析器）

DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();

//step 2：获得具体的DOM解析器

DocumentBuilder builder = dbf.newDocumentBuilder();

//step 3：解析一个XML对象，获得Document对象（根节点对象）

Document doc = builder.parse(new File("./people.xml"));

NodeList list = doc.getElementsByTagName("PERSON");

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

Element element = (Element)list.item(i);

String content = element.getElementsByTagName("NAME").item(0).getFirstChild().

getNodeValue();

System.out.println("name:"+content);

content = element.getElementsByTagName("ADDRESS").item(0).getFirstChild().

getNodeValue();

System.out.println("address:"+content);

content = element.getElementsByTagName("TEL").item(0).getFirstChild().

getNodeValue();

System.out.println("address:"+content);

content = element.getElementsByTagName("FAX").item(0).getFirstChild().

getNodeValue();

System.out.println("address:"+content);

content = element.getElementsByTagName("EMAIL").item(0).getFirstChild().

getNodeValue();

System.out.println("address:"+content);

System.out.println("--------------------------------------------------");

}

}

解析XML文档，当数据是以字符串形式的方式存在是而不是以文件的方式存在的处理方式，因为在WEB应用中我们调用其他服务的接口返回时通常是以字符串的方式来接受信息，而不是将字符串先生成文件。xml分析器同样提供了此种情况的处理方式。

例如：

String xmlStr=”<?xml verison=’1.0’?><student id=’001’>

<name>zhang san</name><age>21</age></student>”;

StringReader rader = new StringReader(xmlStr);

InputSource source = new InputSource(reader);

Document doc = builder.parse(source);

解析任意格式的XML文件

String tagName = element.getTagName();

System.out.print("<"+tagName);

NamedNodeMap nnm = element.getAttributes();

if(null != nnm){

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

Attr attr = (Attr)nnm.item(i);

System.out.print(" "+attr.getName()+"='"+attr.getValue()+"' ");

}

}

System.out.print(">");

NodeList child = element.getChildNodes();

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

Node node = child.item(i);

short type = node.getNodeType();

if(type==Node.ELEMENT\_NODE){

parseElement((Element)node);

}

else if(type==Node.TEXT\_NODE){

System.out.print(node.getTextContent());

}else if(type==Node.COMMENT\_NODE){

Comment comment = (Comment)node;

System.out.println("<!--"+comment.getNodeValue()+"-->");

}

}

System.out.print("</"+tagName+">");

## 具体实例（创建）

**public** **static** **void** createDom() **throws** Exception{

DocumentBuilderFactory dbf = DocumentBuilderFactory.*newInstance*();

DocumentBuilder db = dbf.newDocumentBuilder();

Document dom = db.newDocument();

Element root = dom.createElement("students");

Element student = dom.createElement("student");

root.appendChild(student);

Element name = dom.createElement("name");

student.appendChild(name);

Element age = dom.createElement("age");

student.appendChild(age);

Node ageValue = dom.createTextNode("21");

age.appendChild(ageValue);

name.appendChild(dom.createTextNode("张三"));

dom.appendChild(root);

TransformerFactory tf = TransformerFactory.*newInstance*();

Transformer former = tf.newTransformer();

former.setOutputProperty(OutputKeys.*ENCODING*, "UTF-8");

former.setOutputProperty(OutputKeys.*INDENT*, "yes");

DOMSource source = **new** DOMSource(dom);

FileWriter writer = **new** FileWriter(**new** File("./test.xml"));

StreamResult result = **new** StreamResult(writer);

former.transform(source, result);

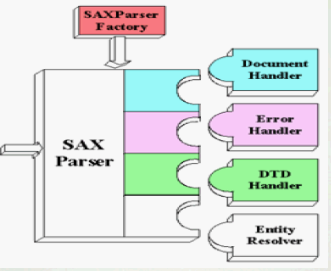
}

# SAX方式解析XML

SAX的全称是Simple APIs For XML，即XML简单应用程序接口。与Dom方式访问不同SAX提供的访问模式是一种顺序模式，这是一种快速读写XML数据的方式。当使用SAX分析器对XML文档进行分析时，会触发一系列事件，并激活相应的事件处理函数，应用程序通过这些事件处理函数实现XML文档的访问，因而SAX接口也被称为事件驱动接口。

SAX分析器在对XML文档进行解析时，触发了一系列的事件，由于事件触发本身是有时序性的，因此，SAX提供的是一种顺序访问机制，对于分析过的部分，不能再倒回去重新处理。SAX之所以被叫做“简单”的应用程序接口，是因为SAX分析器只做了一些简单的工作，大部分工作还是由应用程序自己去做。也就是说SAX分析器在实现时，它只是顺序地检查XML文档中的字节流，判断当前字节是XML语法中的哪一部分，是否符合XML语法，然后再触发相应的事件，而事件处理函数本身则由应用程序本身来实现。同DOM分析器相比，SAX分析器缺乏灵活性。然而，SAX分析器实现简单对内存要求比较低，因此实现效率比较高。对于那些只需要访问XML文档中的数据而不对文档进行更改的应用程序来说，SAX分析器更为合适。

## SAX分析器的大体构成框架



最上方SAXParserFactory用来生成一个分析器实例。XML文档是从左侧开始读入，当分析器对文档进行分析时，就会触发在DocumentHandler,ErrorHandler,DTDHandler以及EntityResolver接口中定义的回调方法。

* XML解析器实际上就是一段代码，它读入XML文档并分析其结构。
* 分类：

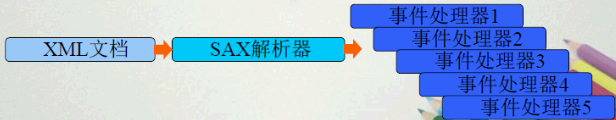
—带校验的分析器

—不带校验的分析器（效率高）

—支持DOM解析器（W3C官方标准）

—支持SAX的解析器（事实的工业标准）

* SAX是事件驱动的，读入XML的过程就是SAX的解析过程。
* 在读入的过程中，遇到不同的项目，解析器就会调用不同的处理方法。



## 常用的事件处理

org.xml.sax.helpers.DefaultHandler

类的方法

|  |  |
| --- | --- |
| 项目 | 处理方法 |
| 文档开始 | startDocument() |
| <PEOPLE> | startElement() |
| “Tony Blair” | Charactors() |
| </PEOPLE> | endElement() |
| 文档结束 | endDocument() |

## 事例：

//step 1：获得SAX解析器工厂实例

SAXParserFactory factory = SAXParserFactory.newInstance();

//step 2：获得SAX解析器实例

SAXParser saxParse = factory.newSAXParser();

//step 3：开始解析

saxParse.parse(new File("./people.xml"), new MyHandler());

class MyHandler extends DefaultHandler{

private Stack<String> stack = new Stack<String>();

private String name;

private String address;

private String tel;

@Override

public void startDocument() throws SAXException {

System.out.println("docuemnt begin");

}

@Override

public void endDocument() throws SAXException {

System.out.println("document end");

}

@Override

public void startElement(String uri, String localName, String qName,Attributes attributes) throws SAXException {

stack.push(qName); //指的当前遇到的元素名称

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

String attrName = attributes.getQName(i);//元素中属性的名称

String attrValue = attributes.getValue(i);//元素中属性的值

System.out.println(attrName + " : " +attrValue );

}

}

@Override

public void characters(char[] ch, int start, int length)throws SAXException {

String tagName = stack.peek();//弹出栈的首位的对象，并返回该对象

if("NAME".equals(tagName)){

name = new String(ch,start,length);

}else if("ADDRESS".equals(tagName)){

this.address = new String(ch,start,length);

}else if("TEL".equals(tagName)){

this.tel = new String(ch,start,length);

}

}

@Override

public void endElement(String uri, String localName, String qName)throws SAXException {

stack.pop();

if("NAME".equals(qName)){

System.out.println("name : "+this.name);

}else if("ADDRESS".equals(qName)){

System.out.println("addres : "+this.address);

}else if("TEL".equals(qName)){

System.out.println("tel : "+this.tel);

}

}

}

# JDOM框架对XML文档的处理

* JDOM是一种使用XML的独特Java工具包，用于快速开发XML应用程序。它的设计包含了Java语言的语法及语义。
* JAXP（用于XML语法分析的Java API）包含了三个软件

—org.w3c.dom，w3c推荐的用于XML标准规划文档对象模型的Java工具

—org.xml.sax，用于XML进行语法分析的事件驱动的简单API

—javax.xml.parsers，工厂化工具，允许应用程序开发人员获得并配置特殊的语法分析器工具。JDOM能够替换org.w3c.dom软件包来有计划的操作XML文档。

* JDOM是一个开源项目，它基于树形结构，利用纯java技术对XML文档进行解析、生成、序列化以及多种操作。(http://www.jdom.org)
* JDOM直接为java编程服务，它利用更为强有力的java语言的诸多特性（方法重载，集合概念等），把SAX跟DOM的功能有效的结合起来。
* JDOM是用Java语言读、写、操作XML的新API函数，在直接、简单有效的前提下，这些API函数被最大限度的优化。
* 在设计上尽量隐藏原来使用XML过程中的的复杂性。利用JDOM处理XML文档将是一件轻松、简单的事情。
* JDOM主要用来弥补DOM和SAX在实际应用中的不足。这些不足之处主要在于SAX对XML文档没有修改、随机访问以及输出功能，而对于DOM来说，java程序员使用起来总是不太方便。
* DOM的缺点主要是由于DOM是一个接口定义语言，它的任务是在不同语言实现中的一个最低的通用标准，并不是为Java特别设计的。
* 在JDOM中，XML元素就是Element的实例，XML属性就是Attribute的实例，XML文档本身即是Document的实例。
* 语言独立

—DOM并不是人们心目中的Java语言设计的，虽然这种方法保留了在不同语言中非常相似的API，它也使那些习惯了Java语言的程序员感到麻烦

—例如：Java语言内建了一种String类，而DOM则规范定义了自己的Text类。

* 严格的层次结构

—DOM API直接沿袭了XML规范，在XML中每个东西都是节点。因此在DOM中找到的每一个东西都是可以扩展的基于Node的的接口和返回Node的一些列方法。

—就多态性的观点来说，它是优秀的。但是鉴于以上解释，它在java语言中的应用是困难而且不方便的。其中从Node向子类型作显示向下类型转换时会导致代码的冗长跟难以理解。

* 接口驱动

—公共的DOM API仅有接口组成。W3C对实现并不感兴趣，它只对定义接口感兴趣。但也意味着作为java程序员在使用API创建XML对象时增加了负担。因此W3C标准大量使用工厂化的类和类似的灵活的但不直接的模式。

—在某些应用程序中，XML文档是仅由语法分析器建立的，而从不会由应用程序级代码建立，这是不相关的。但是，随着XML更广泛的使用，并不是所有问题都需要由语法分析器来驱动。应用程序的开发人员需要一个更为简单的方法有计划的构造XML对象。

* 对于程序员，这些约束以为着庞大（在内存占用和接口大小方面）的和难掌握的API，学习和使用都很难。
* JDOM是作为一种轻量级API被制定的，最主要的是它是以java为中心的，它在遵循DOM主要规则的基础上除去了上述确定。
* JDOM是java平台专用的。

—只要有可能，API都使用Java语言内建的String支持，因此文本值也适用于String。它还可以利用javaEE平台的类集，如List和Iterator，给程序员提供一个丰富的并且和java语言类似的环境。

* 没有层次性，在JDOM中XML元素就是Element的实例，XML属性就是Attribute的实例，XML文档本身就是Document的实例。由于在XML中所有这些都代表了不同的概念，因此它们总是作为自己的类型被引用，而不是作为一个模糊的的“节点”。
* 驱动类：JDOM对象就是Document、Element、Attribute这些类的直接实例，因此创建一个新的JDOM对象就如在java语言中new操作符一样简单，而不需要进行工厂化接口配置。
* JDOM是由以下几个包构成的

—org.jdom包含了所有xml文档要素的java类

—org.jdom.adapters包含了与dom适配器的java类

—org.jdom.filter包含了xml文档的过滤器类

—org.jdom.input包含了读取xml文档的java类

—org.jdom.output包含了输出xml文档的java类

—org.jdom.transform包含了将jdomxml文档接口转换为其他xml文档接口

—org.jdom.xpath包含了对xml文档xpath操作的类

* JDOM类的说明：Attribute、CDATA、Comment、DocType、Document、Element、EntityRef、NameSpace、ProscessingInstruction、Text
* JDOM的输入类：SAXBuilder、DOMBuilder、ResultSetBuilder

例子：生成XML文档

Document document = new Document();

Element root = new Element("root");

root.setAttribute("id", "001");

document.addContent(root);

Comment comment = new Comment("this is myComent");

root.addContent(comment);

Element e = new Element("hello");

e.addContent("hello word");

Attribute attr = new Attribute("color","red");

e.setAttribute(attr);

//方法链编程风格

root.addContent(e).setAttribute("aa","bb").setAttribute("xx","yy").setAttribute("zz","tt");

XMLOutputter out = new XMLOutputter();

out.setFormat(Format.getPrettyFormat().setIndent(" "));

out.output(document, new FileWriter("./jdom.xml"));

例子：解析XML文档

public static void main(String[] args) throws Exception {

SAXBuilder builder = new SAXBuilder();

Document doc = builder.build(new File("./jdom.xml"));

Element root = doc.getRootElement();

printElement(root);

}

public static void printElement(Element e){

System.out.println("元素："+e.getName());

List li = e.getAttributes();

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

Attribute attr = (Attribute)li.get(i);

System.out.print("属性："+attr.getName()+"=="+attr.getValue());

if(i==li.size()-1){

System.out.println(" ");

}

}

if(!"".equals(e.getText().trim())){

System.out.println("文本值："+e.getText().trim());

}

List<Element> list = e.getChildren();

if(list.size()>0){

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

printElement(list.get(i));

}

}else{

System.out.println(e.getName()+"==="+e.getValue());

}

}

方法链编程风格：方法调用后直接返回对象本身，可以继续调用

如：e.setAttribute(“aa”,”bb”).setAttribute(“cc”,”dd”);

# DOM4J框架对XML文档的处理

Dom4J（<http://www.dom4j.org/>）对xml文档的处理方式是以辅助方法集合的方式解析，创建的对象都不是以new操作符进行创建的，而是以dom4j中辅助类DocumentHelper进行创建。

例如：

1. 创建文档对象

Document doc = DocumentHelper.createDocument();

1. 创建元素

Element root = DocumentHelper.createElement(“root”);

例子：生成XML文档

Element root = DocumentHelper.createElement("student");

Document doc = DocumentHelper.createDocument(root);

root.addAttribute("aa", "bb");

root.addElement("name").addAttribute("id", "001").addText("zhang san");

root.addElement("age").addAttribute("id", "002").addText("21");

OutputFormat out = new OutputFormat(" ",true);

XMLWriter write = new XMLWriter(new FileWriter("./student.xml"),out);

write.write(doc);

write.flush();

DOM4J的读取方分为SAXReader、DOMReader两种方式

例子：解析XML文档

SAXReader reader = new SAXReader();

Document doc = reader.read(new File("./student.xml"));

Element root = doc.getRootElement();

System.out.println(root.getName()+"==="+root.getTextTrim()+"111==="+root.getStringValue());

List list = root.attributes();

System.out.println("========="+((Attribute)list.get(0)).getName()+"==="+((Attribute)list.get(0)).getValue());

List el = root.elements();

System.out.println("==="+el.size()+"=="+((Element)el.get(0)).getName()+"==="+((Element)el.get(0)).getText());