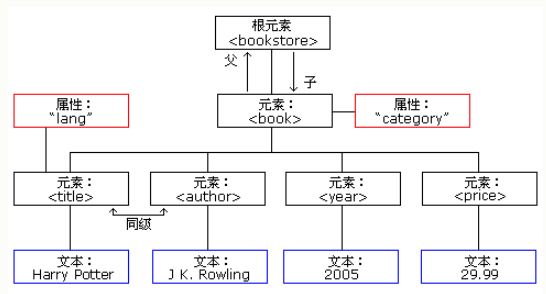
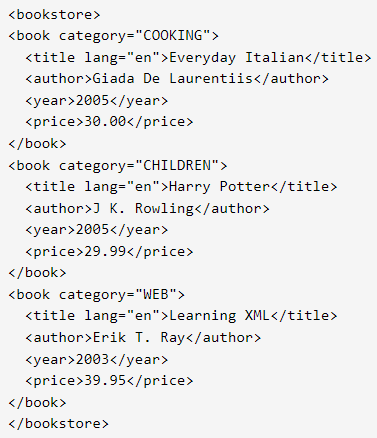
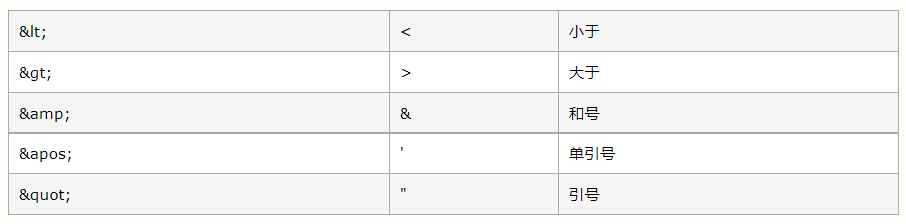
# XML

1. 基础知识
   1. 什么是XML
      1. eXtensible Markup Language
         1. 可扩展**标记语言**
      2. 设计宗旨
         1. **传输数据**，而非显示数据
      3. XML 标签**没有被预定义**。您需要自行定义标签
      4. 被设计为具有**自我描述性**
      5. 是 **W3C 的推荐标准**
      6. XML 仅仅是纯文本，是不作为的
         1. 单纯的XML 文档没有做任何事情。它仅仅是包装在 XML 标签中的纯粹的信息。我们需要编写软件或者程序，才能传送、接收和显示出这个文档
   2. XML 与 HTML 的主要差异
      1. XML **不是 HTML 的替代**
         1. XML 是对 HTML 的补充
         2. XML 是独立于软件和硬件的信息传输工具
      2. XML 和 HTML 为**不同的目的**而设计
         1. XML 被设计为**传输和存储数**据，其**焦点是数据的内容**。
         2. HTML 被设计用来**显示数据**，其**焦点是数据的外观**。
         3. HTML 旨在**显示信息**，而 XML 旨在**传输信息**。
   3. 用途
      1. XML 把数据从 HTML 分离
         1. 通过比如JavaScript，可以读取一个外部 XML 文件，然后更新 HTML 中的数据内容
      2. XML 简化数据共享
         1. 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
      3. XML 简化平台的变更
         1. XML 数据以文本格式存储。这使得 XML 在不损失数据的情况下，更容易扩展或升级到新的操作系统、新应用程序或新的浏览器
      4. XML 使数据更有用
         1. 不同的应用程序都能够访问您的数据，不仅仅在 HTML 页中，也可以从 XML 数据源中进行访问
         2. 通过 XML，数据可供各种阅读设备使用（手持的计算机、语音设备、新闻阅读器等），还可以供盲人或其他残障人士使用
      5. XML 用于创建新的 Internet 语言
         1. 很多新的 Internet 语言是通过 XML 创建的
            1. XHTML - 最新的 HTML 版本
            2. WSDL - 用于描述可用的 web service
            3. WAP 和 WML - 用于手持设备的标记语言
            4. RSS - 用于 RSS feed 的语言
            5. RDF 和 OWL - 用于描述资源和本体
            6. SMIL - 用于描述针针对 web 的多媒体
   4. 树结构
      1. XML 文档形成一种树结构
         1. XML 文档必须包含根元素。该元素是所有其他元素的父元素
         2. 父、子以及同胞等术语用于描述元素之间的关系。父元素拥有子元素。相同层级上的子元素成为同胞
         3. 所有元素均可拥有文本内容和属性
         4. 示例





* 1. 语法
     1. 所有 XML 元素都须有关闭标签
     2. XML 标签对大小写敏感
     3. XML 必须正确地嵌套
     4. XML 文档必须有根元素
     5. XML 的属性值须加引号
     6. 实体引用
        1. 五个预定义的实体引用



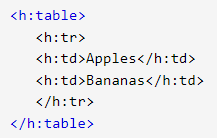
* + - 1. 注意
         1. 在 XML 中，只有字符 "<" 和 "&" 确实是非法的。大于号是合法的，但是用实体引用来代替它是一个好习惯
    1. 在 XML 中，空格会被保留
  1. 元素
     1. 什么是XML元素
        1. XML 元素指的是从（且包括）开始标签直到（且包括）结束标签的部分
        2. 元素可包含其他元素、文本或者两者的混合物。元素也可以拥有属性
     2. XML 命名规则
        1. 名称可以含字母、数字以及其他的字符
        2. 名称不能以数字或者标点符号开始
        3. 名称不能以字符 “xml”（或者 XML、Xml）开始
        4. 名称不能包含空格
        5. **最佳命名习惯**
           1. ***使名称具有描述性。使用下划线的名称也很不错。***
           2. ***名称应当比较简短，比如：<book\_title>，而不是：<the\_title\_of\_the\_book>。***
           3. ***避免 "-" 字符。如果您按照这样的方式进行命名："first-name"，一些软件会认为你需要提取第一个单词。***
           4. ***避免 "." 字符。如果您按照这样的方式进行命名："first.name"，一些软件会认为 "name" 是对象 "first" 的属性。***
           5. ***避免 ":" 字符。冒号会被转换为命名空间来使用***
           6. ***XML 文档经常有一个对应的数据库，其中的字段会对应 XML 文档中的元素。有一个实用的经验，即使用数据库的名称规则来命名 XML 文档中的元素***
     3. XML 元素是可扩展的
  2. 属性
     1. 属性提供的是元素的**额外信息**；属性通常提供**不属于数据组成部分的信息**
     2. 属性值必须加引号
     3. 针对元数据的 XML 属性
        1. 有时候会向元素分配 ID 引用。这些 ID 索引可用于标识 XML 元素，它起作用的方式与 HTML 中 ID 属性是一样的
        2. 注意
           1. 理念是：**元数据**（有关数据的数据）应当存储为属性，而**数据本身**应当存储为元素
     4. 注意
        1. 如果属性值本身包含双引号，那么有必要使用单引号包围它

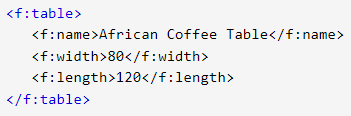


* + - 1. 可以使用实体引用



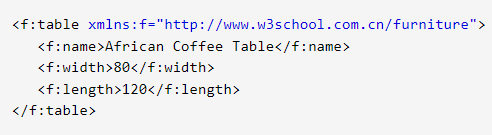
* + - 1. 属性用起来很便利，但是在 XML 中，应该尽量避免使用属性。如果信息感觉起来很像数据，那么使用子元素
         1. 属性无法包含多重的值（元素可以）
         2. 属性无法描述树结构（元素可以）
         3. 属性不易扩展（为未来的变化）
         4. 属性难以阅读和维护
  1. 验证
     1. 拥有正确语法的 XML 被称为“形式良好”的 XML
        1. XML 文档必须**有根元素**
        2. XML 文档必须**有关闭标签**
        3. XML 标签**对大小写敏感**
        4. XML 元素必须**被正确的嵌套**
        5. XML 属性**必须加引号**
     2. 通过 DTD / Schema 验证的 XML 是”有效“的 XML
     3. 合法的XML文档
        1. 格式良好的
        2. 有效的
  2. 命名空间
     1. XML 命名空间提供避免元素命名冲突的方法
        1. **什么是命名冲突**
           1. 在 XML 中，元素名称是由开发者定义的，当两个不同的文档使用相同的元素名时，就会发生命名冲突
     2. 如何解决命名冲突
        1. 使用前缀来避免命名冲突





* + - 1. 使用命名空间





* + - * 1. 注意

与仅仅使用前缀不同，为 <table> 标签添加了一个 **xmlns** 属性，这样就为前缀赋予了一个**与某个命名空间相关联的限定名称**

语法



当命名空间被定义在元素的开始标签中时，所有带有相同前缀的子元素都会与同一个命名空间相关联

注意

用于**标示命名空间的地址不会被解析器用于查找信息**。其惟一的作用是**赋予命名空间一个惟一的名称**。不过，很多公司**常常**会作为指针来使用命名空间**指向实际存在的网页**，**这个网页包含关于命名空间的信息**

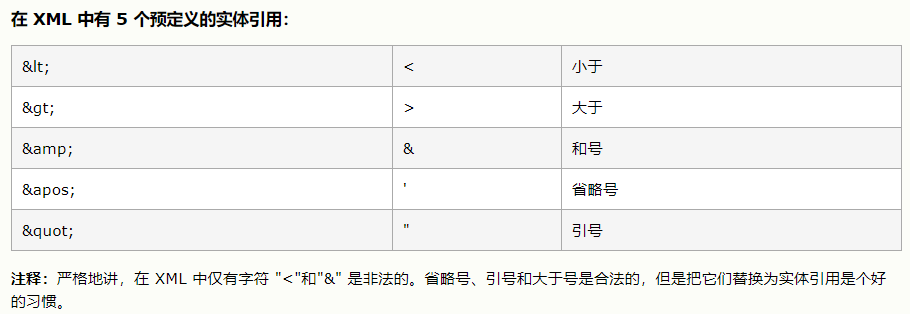
为元素定义**默认的命名空间**可以省去在所有的子元素中使用前缀的工作



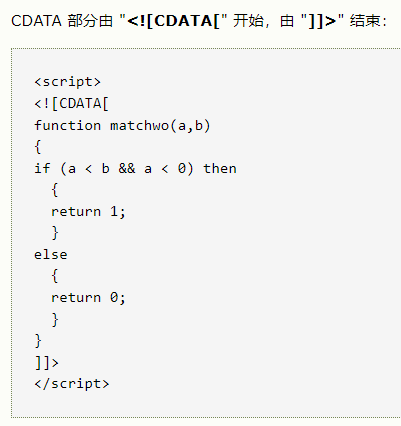
* 1. PCDATA和CDATA
     1. PCDATA
        1. 被解析的**字符数据**
           1. XML 解析器通常会解析 XML 文档中所有的文本
           2. 注意

解析器之所以这么做是因为 XML 元素可包含其他元素

* + 1. 转义字符
       1. 非法的 XML 字符必须被替换为实体引用



* + 1. CDATA
       1. 不应由 XML 解析器进行解析的文本数据
       2. 在XML元素中，有些元素是非法的，会产生错误，因为有解析器的存在，为了避免错误，可以将脚本代码定义为CDATA



* + - * 1. 注意

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

* 1. 编码
     1. XML 文档可以包含非 ASCII 字符，为了避免错误，需要规定 XML 编码，或者将 XML 文档存为 Unicode
     2. 语法
        1. <?xml version="1.0" encoding=“编码格式” ?>
     3. 结论：编码属性应当被指定为**文档被保存时所使用的编码**
        1. 使用支持编码的编辑器
        2. 确定编辑器使用的编码
        3. 在您的 XML 文档中使用相同的编码属性

1. 核心技术
   1. 文档描述、验证、约束技术
      1. DTD
         1. 什么是DTD
            1. 文档类型定义（DTD）可定义合法的XML文档构建模块。它使用一系列合法的元素来定义文档的结构
            2. DTD 可被成行地声明于 XML 文档中，也可作为一个外部引用
         2. 声明方式
            1. 内部声明法

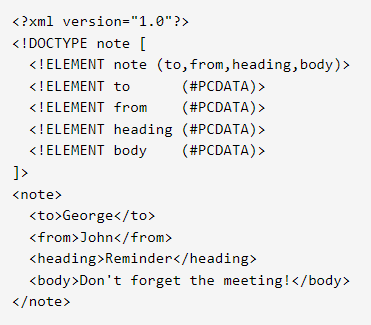
场景

DTD被包含在 XML 源文件中

语法

<!DOCTYPE 根元素 [元素声明]>

示例



* + - * 1. 外部声明法

场景

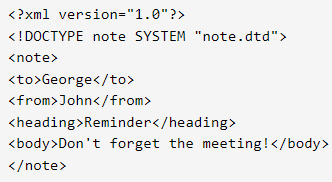
DTD位于 XML 源文件的外部

语法

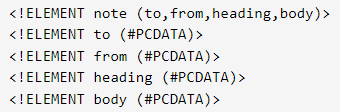
<!DOCTYPE 根元素 SYSTEM "文件名">

示例

xml文件



dtd文件



* + - 1. XML文档构建模块
         1. 元素

如何声明

<!ELEMENT 元素名称 类别>

<!ELEMENT 元素名称 (元素内容)>

空元素

<!ELEMENT 元素名称 EMPTY>

只有PCDATA的元素

<!ELEMENT 元素名称 (#PCDATA)>

带有任何内容的元素

<!ELEMENT 元素名称 ANY>

带有子元素（序列）的元素

<!ELEMENT 元素名称 (子元素名称 1)>

<!ELEMENT 元素名称 (子元素名称 1,子元素名称 2,.....)>

注意

当子元素按照由逗号分隔开的序列进行声明时，这些**子元素必须按照相同的顺序出现在文档中**。在一个完整的声明中，**子元素也必须被声明**，同时子元素也可拥有子元素

声明只出现一次的元素

<!ELEMENT 元素名称 (子元素名称)>

声明**最少出现一次（≥1）**的元素

<!ELEMENT 元素名称 (子元素名称**+**)>

声明出现**零次或多次（≥0）**的元素

<!ELEMENT 元素名称 (子元素名称**\***)>

出现**零次或一次（0或1）**的元素

<!ELEMENT 元素名称 (子元素名称**?**)>

**声明”非.../既...“类型的内容**

<!ELEMENT note (to,from,header,**(message|body)**)>

声明**混合型**的内容

<!ELEMENT note (#PCDATA|to|from|header|message)\*>

"note" 元素可包含出现零次或多次的 PCDATA、"to"、"from"、"header" 或者 "message"

* + - * 1. 属性

如何声明

<!ATTLIST 元素名称 属性名称 属性类型 默认值>



属性类型的选项

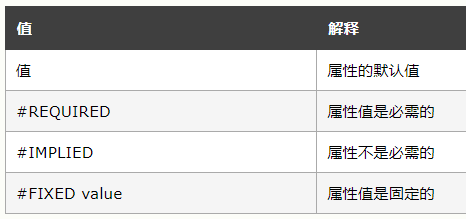


注意

NMTOKEN是CDATA的一个子集，表示属性值必须是由字母、数字、下划线和点组成

NMTOKENS是多个NMTOKEN的集合，由逗号隔开

默认值参数可使用的选项



规定一个默认值属性

<!ELEMENT square EMPTY><!ATTLIST square width CDATA "0">

列举属性值

如果您希望属性值为一系列固定的合法值之一，请使用列举属性值

<!ATTLIST payment type (check|cash) "cash">

IMPLIED

假如您不希望强制作者包含属性，并且您没有默认值选项的话，请使用关键词 #IMPLIED

<!ATTLIST contact fax CDATA #IMPLIED>

REQUIRED

假如您没有默认值选项，但是仍然希望强制作者提交属性的话，请使用关键词 #REQUIRED

<!ATTLIST person number CDATA #REQUIRED>

FIXED

如果您希望属性拥有固定的值，并不允许作者改变这个值，请使用 #FIXED 关键词。如果作者使用了不同的值，XML 解析器会返回错误。

<!ATTLIST sender company CDATA #FIXED "Microsoft">

* + - * 1. 实体

什么是实体

实体是用于**定义引用普通文本或特殊字符的快捷方式的变量**

实体引用是**对实体的引用**

实体可**在内部或外部进行声明**

内部声明

语法

<!ENTITY 实体名称 "实体的值">

示例



注意

一个实体由多部分构成: 一个和号 (&), 一个实体名称, 以及一个分号 (;)

外部声明

语法

<!ENTITY 实体名称 SYSTEM "URI/URL">

示例



* + - * 1. PCDATA
        2. CDATA
    1. Schema
       1. 什么是XML Schema
          1. 概念

XML Schema Definition

基于 XML 的 DTD 替代者

可描述 XML 文档的结构

* + - * 1. 作用

定义 XML 文档的合法构建模块

定义可出现在**文档中的元素**

定义可出现在**文档中的属性**

定义哪个元素是**子元素**

定义**子元素的次序**

定义**子元素的数目**

定义元素**是否为空，或者是否可包含文本**

定义**元素和属性的数据类型**

定义**元素和属性的默认值以及固定值**

* + - * 1. 为什么XML Schema是DTD的继承者

XML Schema 可针**对未来的需求进行扩展**

XML Schema **更完善，功能更强大**

XML Schema **基于 XML 编写**

XML Schema **支持数据类型**

XML Schema **支持命名空间**

* + - 1. 为何使用
         1. XML Schema 支持数据类型

可更容易地描述允许的文档内容

可更容易地验证数据的正确性

可更容易地与来自数据库的数据一并工作

可更容易地定义数据约束（data facets）

可更容易地定义数据模型（或称数据格式）

可更容易地在不同的数据类型间转换数据

* + - * 1. XML Schema 使用 XML 语法
        2. XML Schema 可保护数据通信
        3. XML Schema 可扩展

在其他 Schema 中重复使用您的 Schema

创建由标准类型衍生而来的您自己的数据类型

在相同的文档中引用多重的 Schema

* + - * 1. 形式良好是不够的

即使文档的形式良好，仍然不能保证它们不会包含错误，并且这些错误可能会产生严重的后果

* + - 1. 如何使用
         1. <schema>

每一个 XML Schema 的根元素

格式

<?xml version="1.0"?>

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

targetNamespace="(自己定义)"

xmlns="(自己定义)"

elementFormDefault="qualified"> ​...... ​</xs:schema>

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

显示 schema 中用到的元素和数据类型来自命名空间 "<http://www.w3.org/2001/XMLSchema>"。同时它还规定了来自命名空间 "<http://www.w3.org/2001/XMLSchema>" 的元素和数据类型应该使用前缀 xs：

targetNamespace="..."

显示被此 schema 定义的元素所来自的命名空间

xmlns="..."

指出默认的命名空间

elementFormDefault="qualified"

指出任何 XML 实例文档所使用的且在此 schema 中声明过的元素必须被命名空间限定

* + - * 1. 引用Schema

格式

<?xml version="1.0"?> <note xmlns="(自己定义)" xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>" xsi:schemaLocation="(自己定义) (xsd文件)"> ​... ​</note>

xmlns="..."

规定了默认命名空间的声明

xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>"

一旦拥有了可用的 XML Schema 命名空间将被实例化

xsi:schemaLocation="... ..."

此属性有两个值。第一个值是需要使用的命名空间。第二个值是供命名空间使用的 XML schema 的位置

* + - 1. 简单类型
         1. 什么是简单类型

指那些**仅包含文本的元素**。它**不会包含任何其他的元素或属性**

* + - * 1. 元素

定义简单属性

<xs:element name="..." type="..."/>

常用类型

xs:string

xs:decimal

xs:integer

xs:boolean

xs:date

xs:time

默认值和固定值

默认值的示例

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

固定值的示例

<xs:element name="color" type="xs:string" **fixed="red"**/>

* + - * 1. 属性

什么是属性

**简易元素无法拥有属性**。假如某个元素拥有属性，它就会被当作某种复合类型。但是**属性本身总是作为简易类型被声明的**。

定义属性

<xs:attribute name="..." type="..."/>

常用属性

xs:string

xs:decimal

xs:integer

xs:boolean

xs:date

xs:time

默认值和固定值

默认值的示例

<xs:attribute name="lang" type="xs:string" **default="EN"**/>

固定值的示例

<xs:attribute name="lang" type="xs:string" **fixed="EN"**/>

可选和必选的属性

在缺省的情况下，属性是可选的。如需规定属性为必选，请使用 "use" 属性

<xs:attribute name="lang" type="xs:string" **use="required"**/>

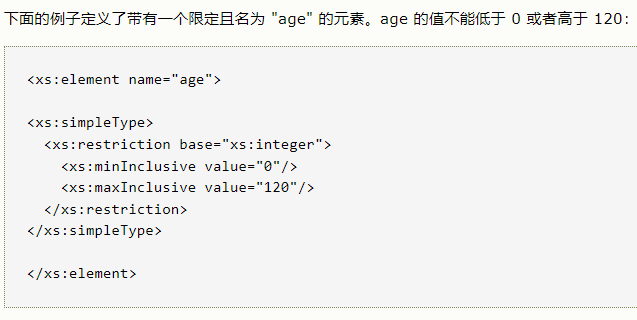
* + - * 1. 限定

什么是限定

限定用于为 XML 元素或者属性定义可接受的值。对 XML 元素的限定被称为 facet

对值的限定

示例



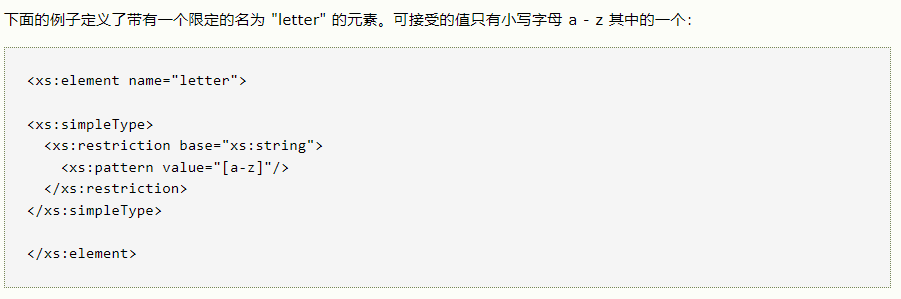
对一组值的限定

示例

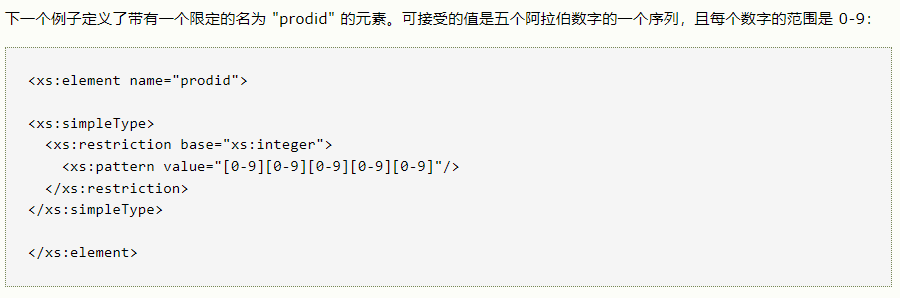


对一系列值的限定

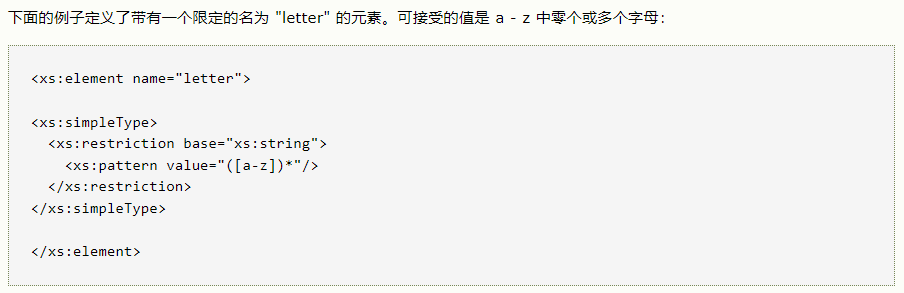
示例1



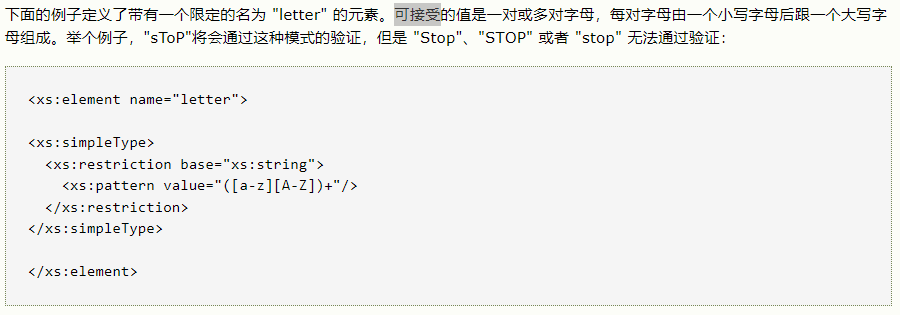
示例2



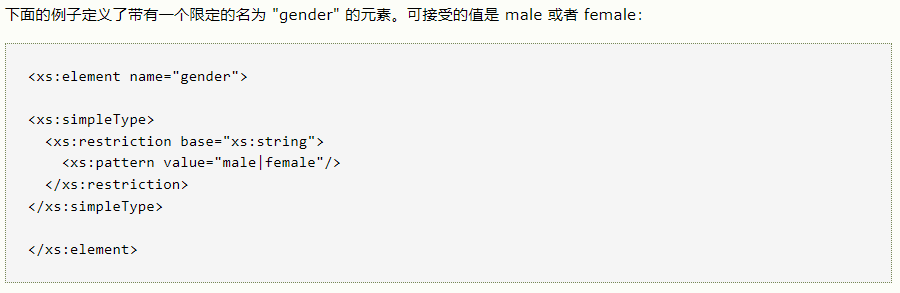
示例3



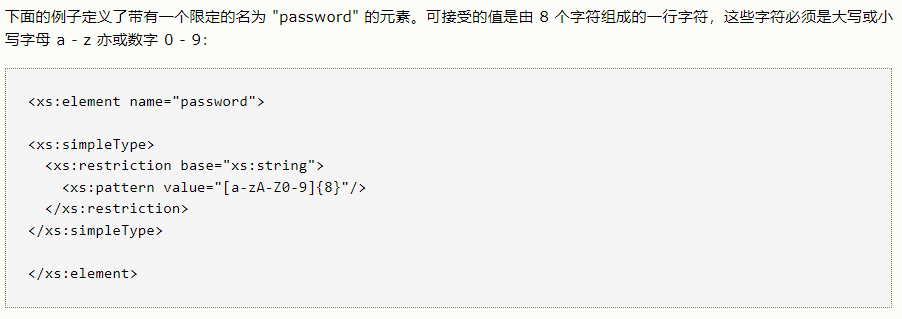
示例4



示例5

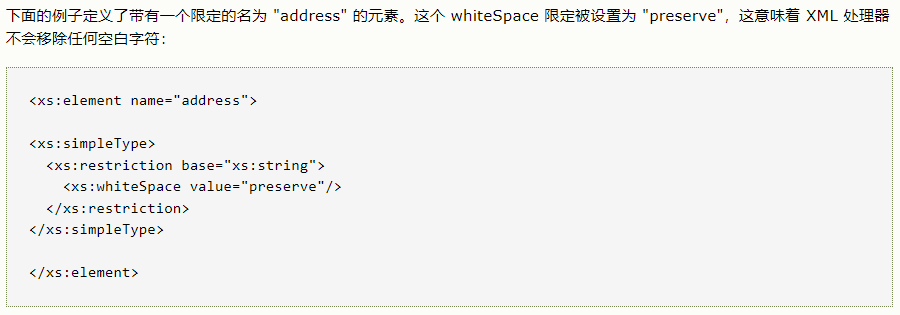


示例6

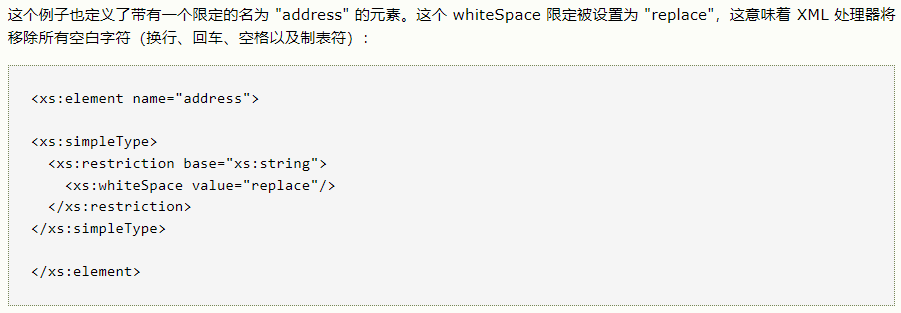


对空白字符的限定

preserve



replace

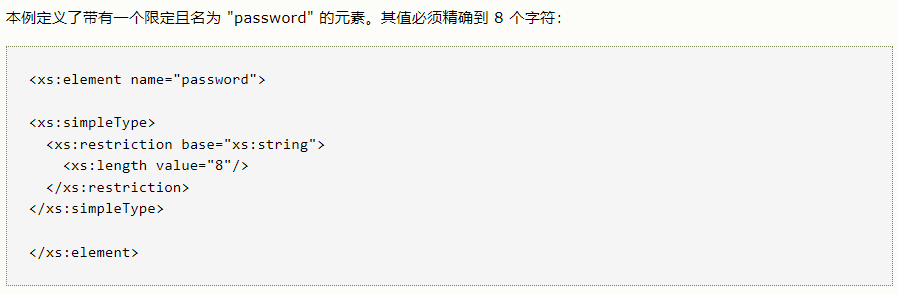


collapse

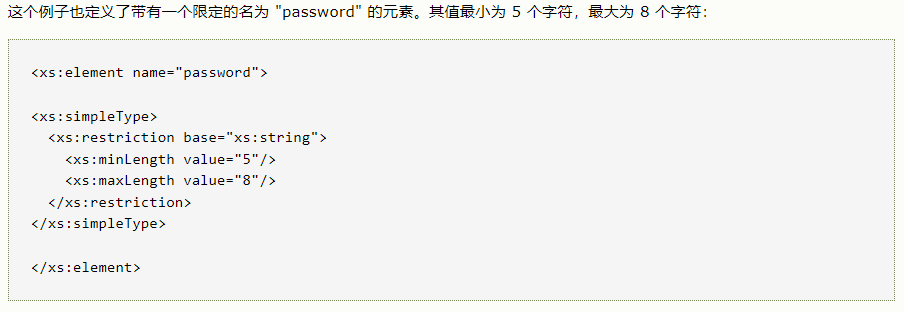


对长度的限定

示例1



示例2



数据类型的限定的汇总



* + - 1. 复杂类型
         1. 什么是复合元素

指包含其他元素及/或属性的 XML 元素

如何定义复合元素

不可复用

<xs:element name="元素名称"> <xs:complexType> （其他成分） </xs:complexType> ​</xs:element>

可复用

<xs:element name="元素名称" type="自定义类型名称"/> <xs:complexType name="自定义类型名称"> （其他成分） ​</xs:complexType>

可以在已有复合元素上添加一些元素

<xs:element name="元素名称" type="自定义类型1"/>

<xs:complexType name="自定义类型2">

（其他成分）

</xs:complexType>

<xs:complexType name="自定义类型1">

<xs:complexContent>

<xs:extension base="自定义类型2">

（其他成分）

</xs:extension>

</xs:complexContent> ​

</xs:complexType>

四种类型

空元素

什么是空元素

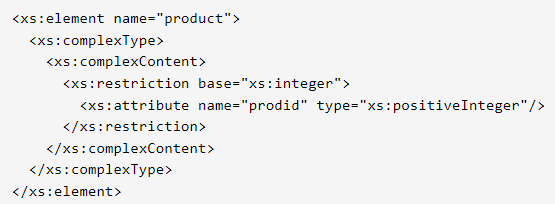
空的复合元素**不能包含内容，只能含有属性**

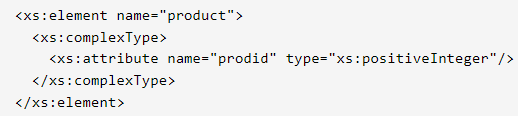
示例



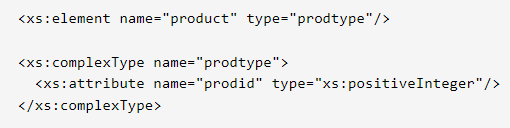
xsd代码

不可复用





可复用

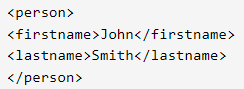


仅含元素

什么是仅含元素

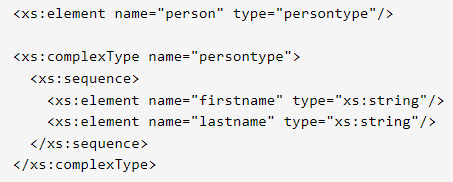
只能包含其他元素的元素

示例

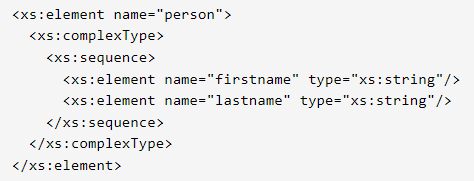


xsd代码

可复用



不可复用

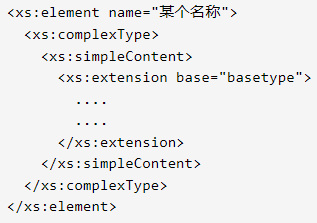


仅包含文本的元素

什么是仅包含文本的元素

仅包含简易的内容（文本和属性），要向此内容添加 simpleContent 元素。当使用简易内容时，就必须在 simpleContent 元素内**定义扩展或限定**

扩展



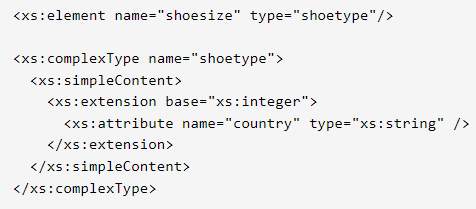
示例

xml代码



xsd代码

可复用



不可复用



限定



包含元素和文本的元素

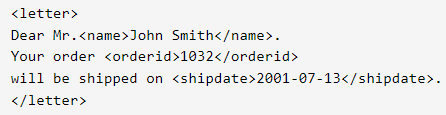
什么是混合内容

可包含属性、元素以及文本

带有混合内容的复合类型

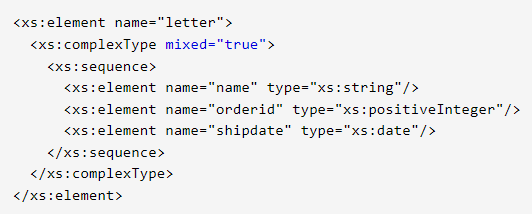
示例

xml代码

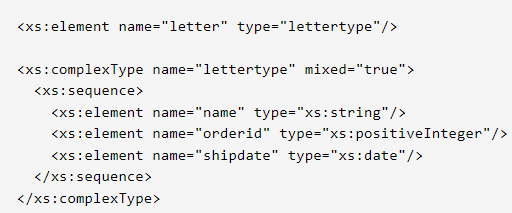


xsd代码

不可复用



可复用



注意

为了使字符数据可以出现在子元素之间，mixed 属性必须被设置为 "true"

指示器

什么是指示器

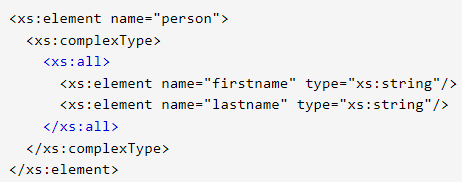
通过指示器，我们可以控制在文档中使用元素的方式

七种指示器

order指示器：用于定义元素的顺序

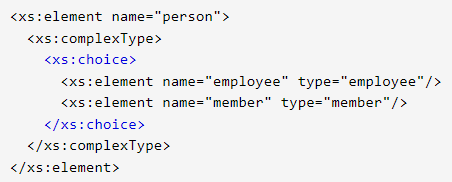
all

规定子元素可以**按照任意顺序出现**，且每个子元素**必须只出现一次**



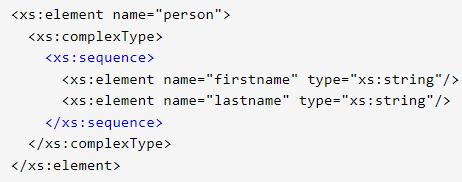
choice

规定可出现某个子元素或者可出现另外一个子元素**（非此即彼）**



sequence

规定子元素**必须按照特定的顺序出现**

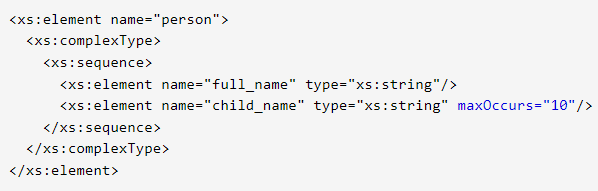


occurrence指示器：用于定义某个元素出现的频率

maxOccurs

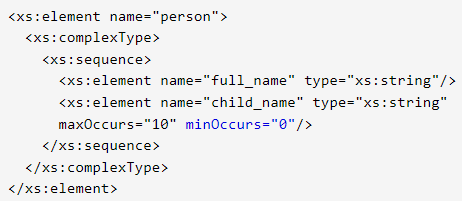
可规定某个元素可出现的**最大次数**，默认为1

如需使某个元素的**出现次数不受限制**，请使用 maxOccurs="unbounded"



minOccurs

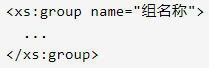
可规定某个元素能够出现的**最小次数**，默认为1



group指示器：用于定义相关的数批元素

元素组

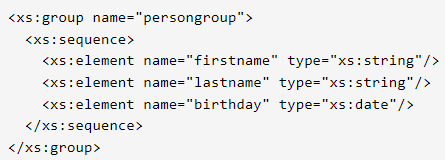
语法



注意

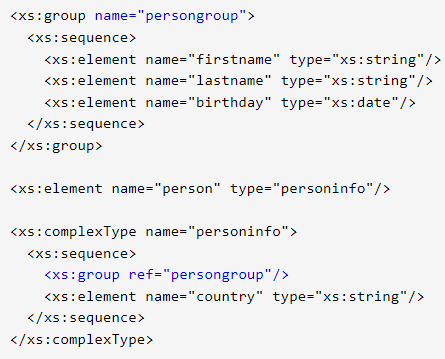
**必须**在 group 声明内部定义一个 all、choice 或者 sequence 元素

示例



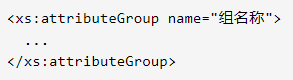
使用**ref**进行引用

示例

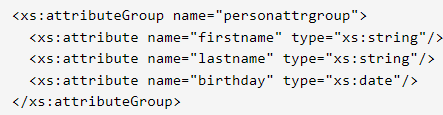


属性组

语法



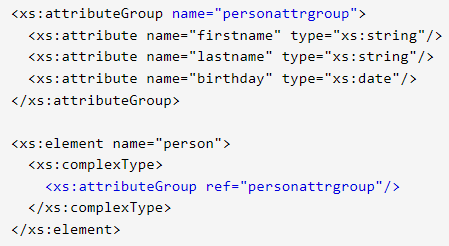
示例



注意

使用**ref**进行引用

示例



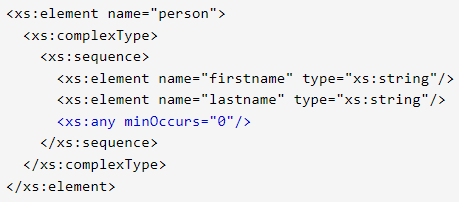
any

什么是any元素

使有能力通过未被 schema 规定的元素来拓展 XML 文档

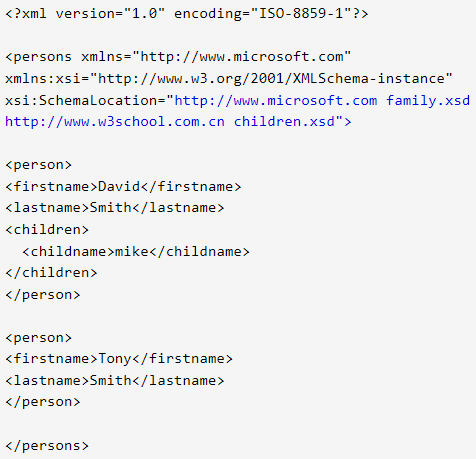
示例

xsd代码





xml代码



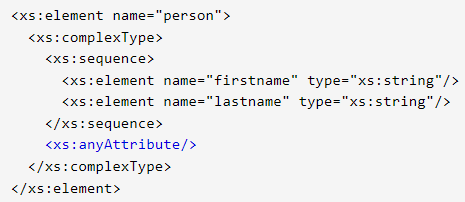
anyAttribute

什么是anyAttribute元素

使有能力通过未被 schema 规定的属性来扩展 XML 文档

示例

xsd代码





xml代码



* + - 1. 数据类型
         1. 字符串

字符串数据类型

示例

<xs:element name="customer" **type="xs:string"**/>

规格化字符串数据类型

解释

规格化字符串数据类型同样可包含字符，但是 XML 处理器**会移除折行，回车以及制表符**

示例

<xs:element name="customer" **type="xs:normalizedString"/**>

Token数据类型

解释

Token 数据类型同样可包含字符，但是 **XML 处理器会移除换行符、回车、制表符、开头和结尾的空格以及（连续的）空格**

示例

<xs:element name="customer" **type="xs:token"**/>

其他衍生于字符串数据类型的类型



对字符串数据类型的限定

enumeration

length

maxLength

minLength

pattern (NMTOKENS、IDREFS 以及 ENTITIES 无法使用此约束)

whiteSpace

* + - * 1. 日期

日期数据类型



时间数据类型



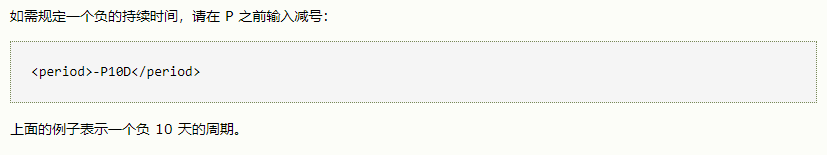
日期时间数据类型



持续时间数据类型



负的持续时间



汇总



对日期数据类型的限定

enumeration

maxExclusive

maxInclusive

minExclusive

minInclusive

pattern

whiteSpace

* + - * 1. 数值

十进制数据类型

示例

<xs:element name="prize" **type="xs:decimal"**/>

可规定的十进制数字的最大位数是 18 位

整数数据类型

示例

<xs:element name="prize" **type="xs:integer"**/>

数值数据类型汇总



对数值数据类型的限定

enumeration

fractionDigits

maxExclusive

maxInclusive

minExclusive

minInclusive

pattern

totalDigits

whiteSpace

* 1. 编程接口（python）
     1. 解释
        1. 常见的XML编程接口有DOM和SAX，这两种接口处理XML文件的方式不同，使用场合也不同
        2. python有三种解析XML
           1. DOM（Document Object Model）

一个 DOM 的解析器在解析一个 XML 文档时，一次性读取整个文档，把文档中所有元素保存在内存中的一个树结构里

使用场景

利用DOM 提供的不同的函数来**读取或修改文档的内容和结构**，也可以把**修改过的内容写入xml文件**

小案例

xml



DOM解析



* + - * 1. SAX（simple API for XML）

一个基于事件驱动的API。通过在解析XML的过程中触发一个个事件并调用用户定义的回调函数。利用SAX解析XML文档牵涉两个部分：**解析器和事件处理器**。**解析器负责读取XML文档，并向事件处理器发送事件。而事件处理器则负责对事件作出相应，对传递的XML数据进行处理**

使用场景

对大型文件进行处理

只需要文件的部分内容，或者只需要从文件中得到特定信息

想建立自己的对象模型

方法介绍

ContentHandler类方法

character(content)

调用时机

从行开始，遇到标签之前，存在字符，content 的值为这些字符串。

从一个标签，遇到下一个标签之前， 存在字符，content 的值为这些字符串。

从一个标签，遇到行结束符之前，存在字符，content 的值为这些字符串。

标签可以是开始标签，也可以是结束标签。

startDocument()

文档启动的时候调用

endDocument()

解析器到达文档结尾时调用

startElement(name, attrs)

遇到XML开始标签时调用，name是标签的名字，attrs是标签的属性值字典

endElement(name)

遇到XML结束标签时调用

make\_parse方法

创建一个新的解析器对象并返回

xml.sax.make\_parser( [parser\_list] )

parse方法

创建一个 SAX 解析器并解析xml文档

xml.sax.parse( xmlfile, contenthandler[, errorhandler])

parseString方法

parseString方法创建一个XML解析器并解析xml字符串

xml.sax.parseString(xmlstring, contenthandler[, errorhandler])

小案例

xml



SAX解析



* + - * 1. ElementTree
        2. DOM和SAX的比较

DOM

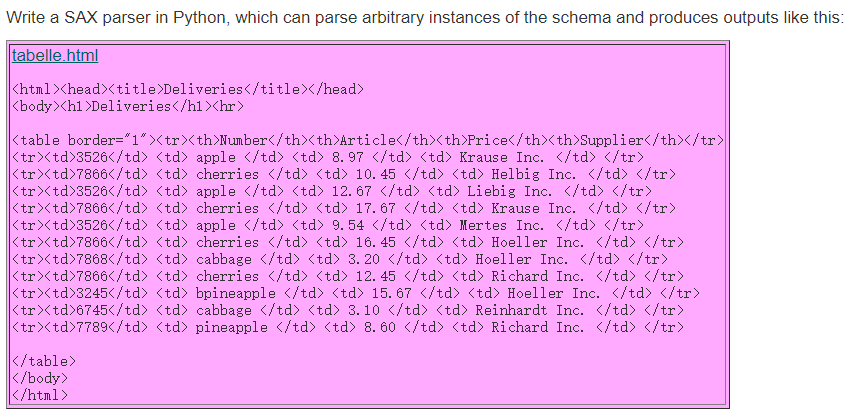
需要将XML数据映射到内存中的树，一是缓慢，二是比较消耗内存

SAX

流式读取XML文件，比较快，占用内存少，但是需要用户实现回调函数

* + - * 1. Learning Block 2

task1



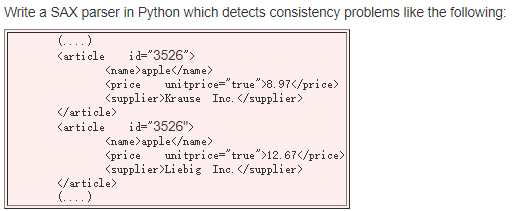
xsd和xml代码



SAX代码



task2



xsd和xml代码



SAX代码

版本1



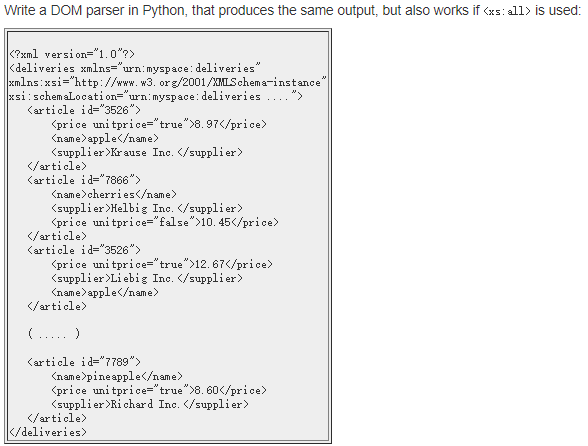
直接输出，即方法和输出混在了一起

版本2



通过数组和字典组合输出，即将方法和输出分开来思考

task3



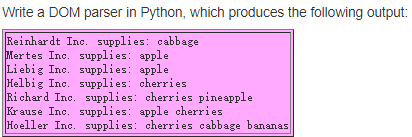
xml和xsd代码



DOM



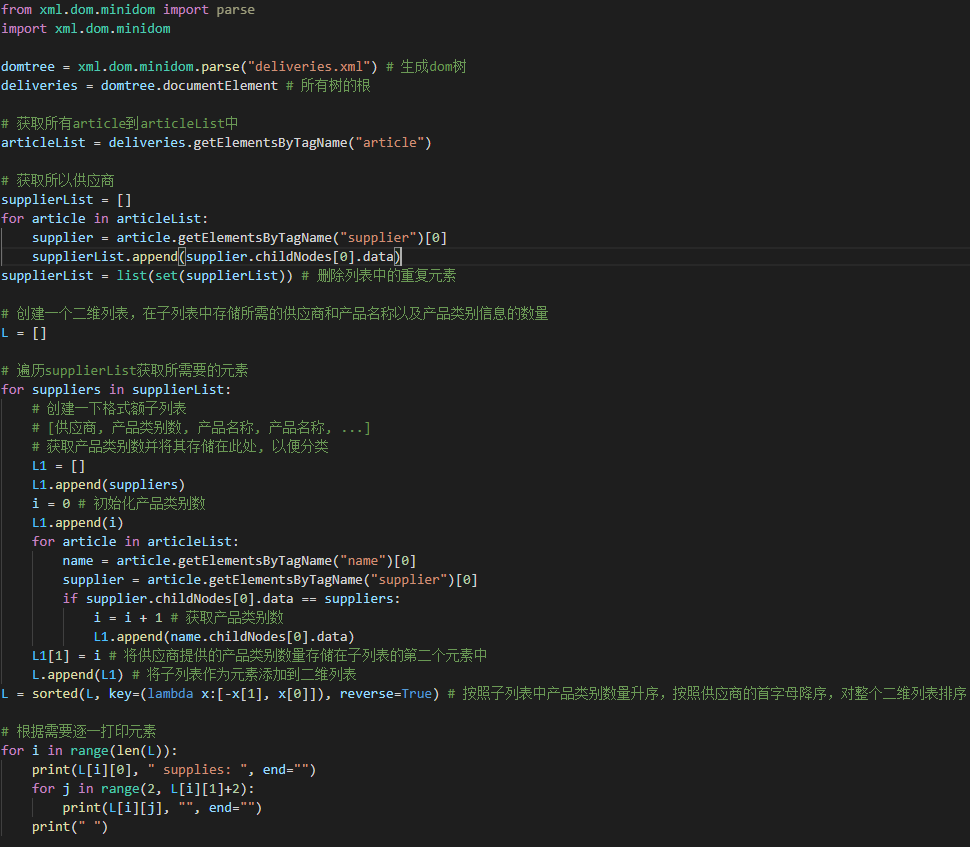
task4



xml和xsd代码



DOM代码



* 1. 文档转换技术
     1. 什么是XSL
        1. XSL 指扩展样式表语言（eXtensible Sylesheet Language）
        2. XSL = XML样式表
           1. XSL 可描述如何来显示 XML 文档
        3. XSL不仅仅是样式表语言
           1. XSLT

什么是XSLT

XSLT 指 XSL 转换（XSL Transformations）

XSLT 是 XSL 中最重要的**部分**

XSLT 可**将一种 XML 文档转换为另外一种 XML 文档**

XSLT 把 XML 源树转换为 XML 结果树

XSLT 是通过**把每个 XML 元素转换为 (X)HTML 元素**来完成工作的

通过 XSLT，可以**向或者从输出文件添加或移除元素和属性**。也**可重新排列元素**，**执行测试**并**决定隐藏**或**显示**哪个元素，等等

XSLT的使用

XSLT转换

把文档声明为 XSL 样式表的根元素是 **<xsl:stylesheet>** 或 **<xsl:transform>**

<xsl:stylesheet version="1.0" xmlns:xsl="<http://www.w3.org/1999/XSL/Transform>">

<xsl:transform version="1.0" xmlns:xsl="<http://www.w3.org/1999/XSL/Transform>">

注释

xmlns:xsl="<http://www.w3.org/1999/XSL/Transform>" 指向了官方的 W3C XSLT 命名空间

把 XSL 样式表链接到 XML 文档

向 XML 文档添加 XSL 样式表引用

<?xml-stylesheet type="text/xsl" href="xsl文件名"?>

XSLT元素

主要元素

**<xsl:apply-templates>和<xsl:template>**



<xsl:apply-templates>

可把一个模板应用于当前的元素或者当前元素的子节点

向 <xsl:apply-templates> 元素添加一个 select 属性，此元素就会仅仅素。我们可以使用 select 属性来规定子节点被处理的顺序

<xsl:template>

用于构建模板

match 属性用于关联 XML 元素和模板。match 属性也可用来为整个文档定义模板。match 属性的值是 XPath 表达式（举例，match="/" 定义整个文档）

**<xsl:value-of>**



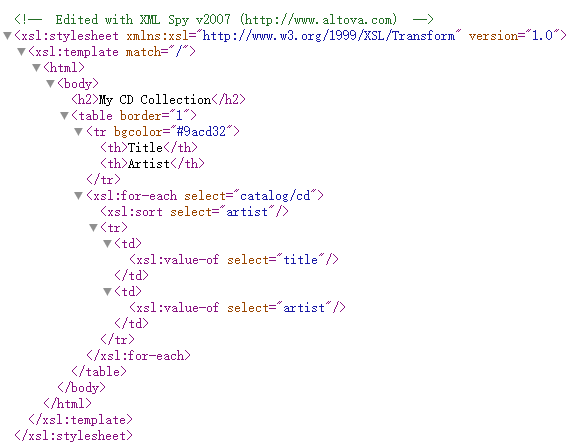
用于提取某个选定节点的值，并把值添加到转换的输出流中

**<xsl:for-each>**



用于选取指定的节点集中的每个 XML 元素

**<xsl:sort>**



如需对结果进行排序，只要简单地在 XSL 文件中的 <xsl:for-each> 元素内部添加一个 <xsl:sort> 元素

常用属性

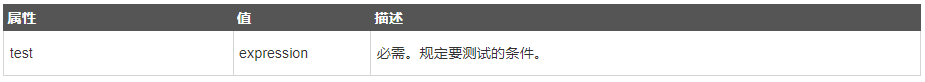


**<xsl:if>**



如需放置针对 XML 文件内容的条件测试，请向 XSL 文档添加 <xsl:if> 元素

常用属性



**<xsl:choose> 、<xsl:when> 和 <xsl:otherwise>**



<xsl:choose>

<xsl:choose> 元素与 <xsl:when> 以及 <xsl:otherwise> 协同使用，来表达多重条件测试。

如果没有 <xsl:when> 是 true，则处理 <xsl:otherwise> 的内容。

如果没有 <xsl:when> 是 true，且不存在 <xsl:otherwise> 元素，则不创建任何内容。

**注意：**对于简单的条件测试，请使用 <xsl:if> 元素取而代之。

<xsl:when>

<xsl:when> 元素用于为 <xsl:choose> 元素规定相关动作。 <xsl:when> 元素会计算一个表达式，如果返回 true，则执行规定的动作。

**注意：**<xsl:when> 元素与 <xsl:choose> 元素和 <xsl:otherwise> 元素协同使用，来表达多个条件测试

<xsl:otherwise>

<xsl:otherwise> 元素规定了 <xsl:choose> 元素的默认行为。在没有 <xsl:when> 的条件应用时，将发生该行为

其他元素

**<xsl:apply-imports>和<xsl:import>**



<xsl:apply-imports>

可应用来自导入样式表中的模版规则

导入样式表中的模版规则的优先级要比主样式表中的模版规则要低。如果希望使用导入样式表中的某条模版规则，而不是主样式表中的某条等价规则，就会用到 <xsl:apply-imports> 元素

<xsl:import>

顶层元素，用于把一个样式表中的内容倒入另一个样式表中

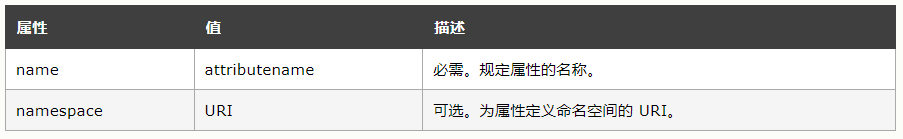
**<xsl:attribute>和<xsl:attribute-set>**



<xsl:attribute>

用于向元素添加属性和属性组

常用属性

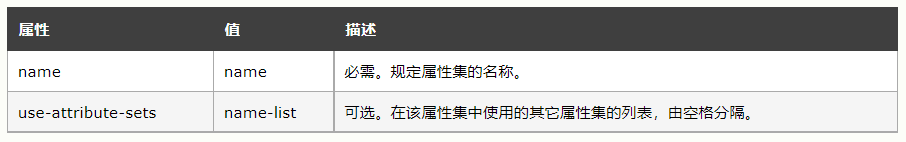


**注释：**<xsl:attribute> 元素会替换名称相同的已有属性

<xsl:attribute-set>

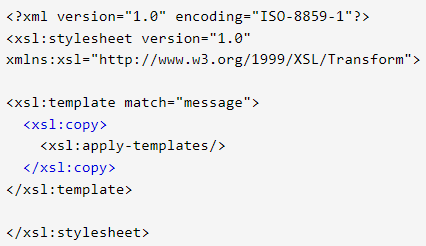
创建命名的属性集。该属性集（attribute-set）可作为整体应用到输出文档

常用属性



**注释：**必须是 <xsl:stylesheet> 或 <xsl:transform> 的子节点

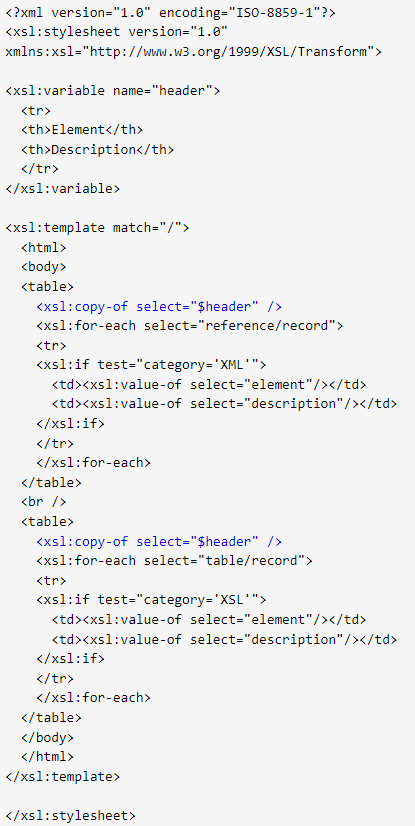
**<xsl:copy>**



可创建当前节点的一个副本（拷贝）

**注释：**当前节点的 Namespace 节点会被自动复制，但是当前节点的子节点和属性不会被自动复制！

**<xsl:copy-of>**



可创建当前节点的一个副本

**注释**

该元素可用于把相同节点的多个副本插入到输出的不同位置

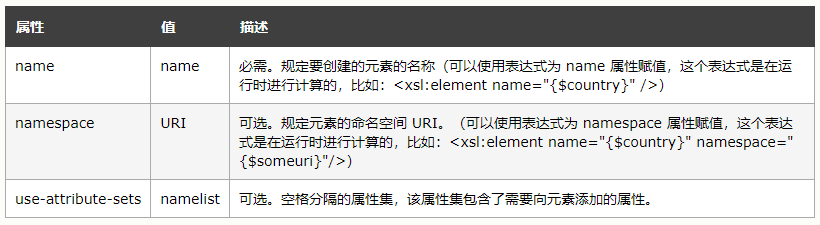
当前节点的 Namespace 节点、子节点以及属性都会被自动复制

**<xsl:element>**

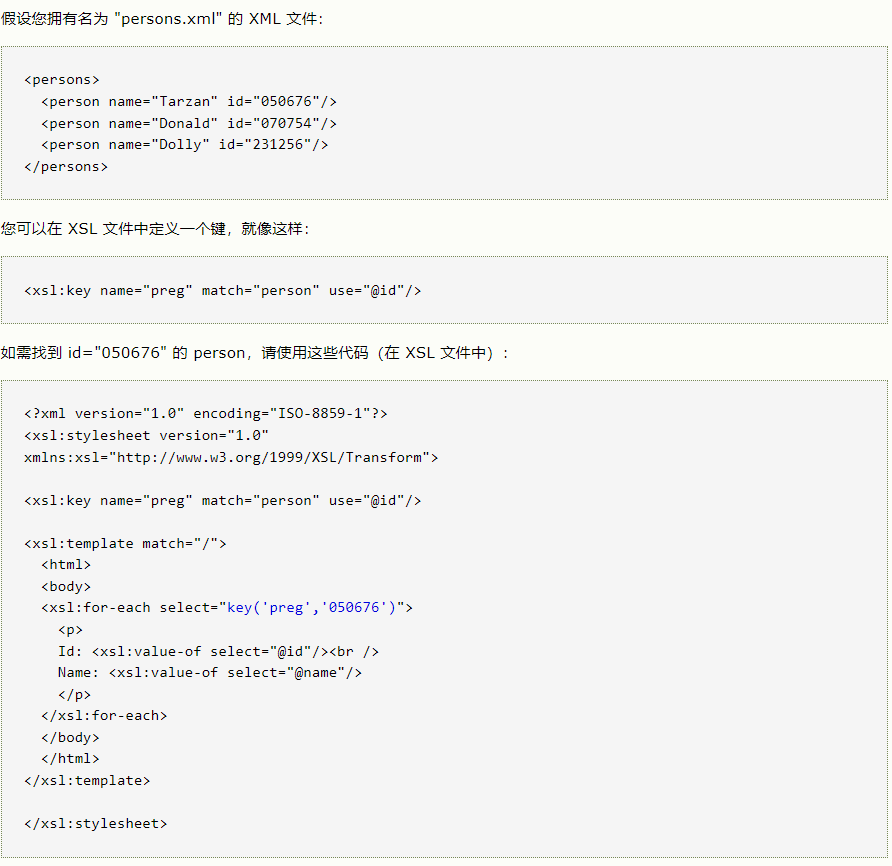


用于在输出文档中创建元素节点

常用属性



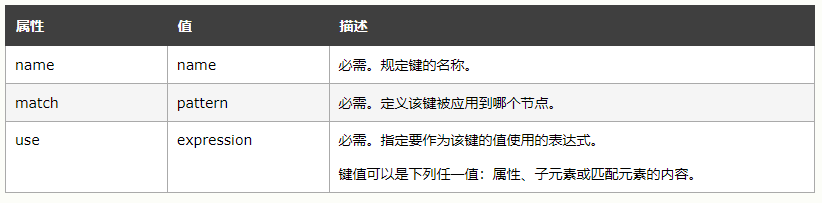
**<xsl:key>**



是顶层元素，它可声明一个命名的键（即为 XML 文档中指定的元素分配的名称和值对）

此键通过 key() 函数在样式表中使用，帮助有效地在复杂的 XML 文档中访问分配的元素

常用熟悉

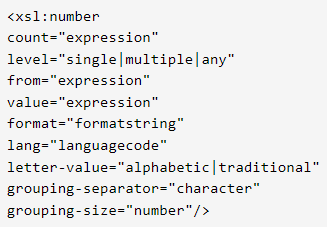


**<xsl:number>**



用于测定在源中当前节点的整数位置。它也用于将格式化的数字插入结果树

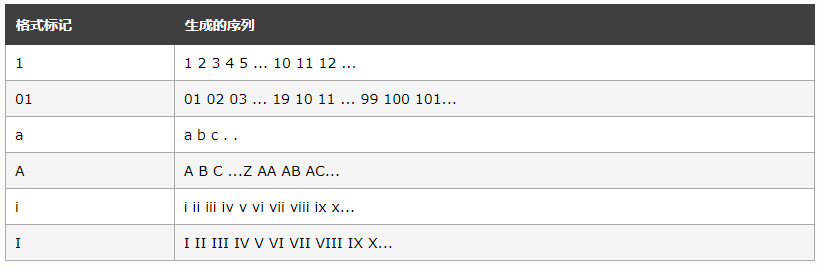
语法



常用属性



格式标记



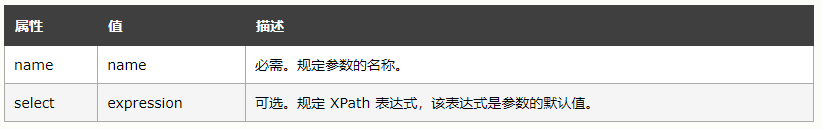
**<xsl:param>和<xsl:with-param>**



<xsl:param>

用于声明局部或全局参数

常用属性



**注释：**如果在模板内声明参数，就是局部参数，如果作为顶层元素来声明，就是全局参数

<xsl:with-param>

定义了传递给模板的参数的值

常用属性



**注释：**<xsl:with-param> 元素的 name 属性的值必须与 <xsl:param> 元素中的 name 相匹配，否则将忽略 <xsl:with-param> 元素

**<xsl:preserve-space>和<xsl:strip-space>**



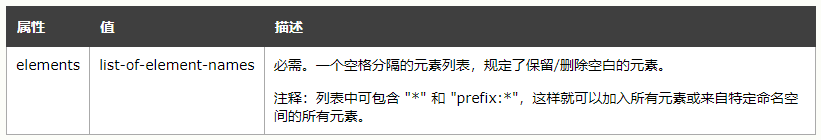
<xsl:preserve-space>

用于定义保留空白的元素

<xsl:strip-space>

用于定义删除空白的元素

常用属性



**注意：**注释：保留空白是默认的设置，所以只有当使用 <xsl:strip-space> 元素使才有必要使用 <xsl:preserve-space> 元素

**<xsl:variable>**





用于声明局部或全局的变量

常用属性



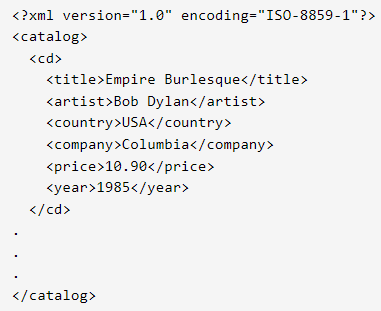
**注释**

如果被声明为顶层元素，则该变量是全局的，而如果在模板内声明，则变量是本地的

一旦设置了变量的值，就无法改变或修改该值

可以通过 <xsl:variable> 元素的内容或通过 select 属性，向变量添加值

公共文件XML



XSLT函数



* + - * 1. XPath (数据查询技术)

什么是XPath

XPath 使用路径表达式在 XML 文档中进行导航

XPath 包含一个标准函数库

XPath 是 XSLT 中的主要元素

XPath 是一个 W3C 标准

XPath路径表达式

XPath 使用路径表达式来选取 XML 文档中的节点或者节点集。这些路径表达式和我们在常规的电脑文件系统中看到的表达式非常相似

XPath节点

在 XPath 中，有七种类型的节点：**元素、属性、文本、命名空间、处理指令、注释以及文档（根）节点**。XML 文档是被作为节点树来对待的。树的根被称为文档节点或者根节点

基本值（或称原子值，Atomic value）

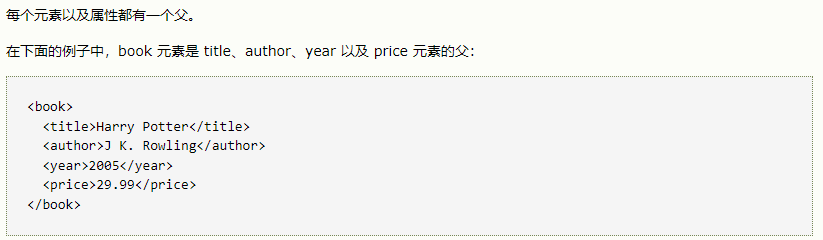
无父或无子的节点

项目

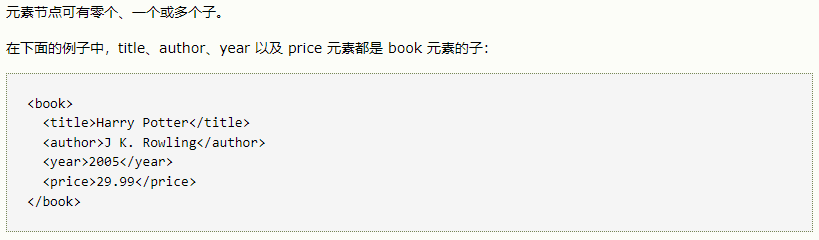
基本值或者节点

节点关系

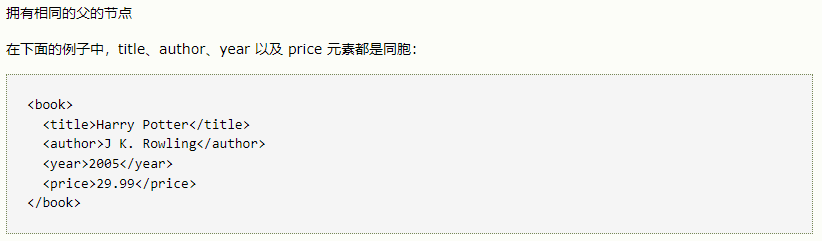
父



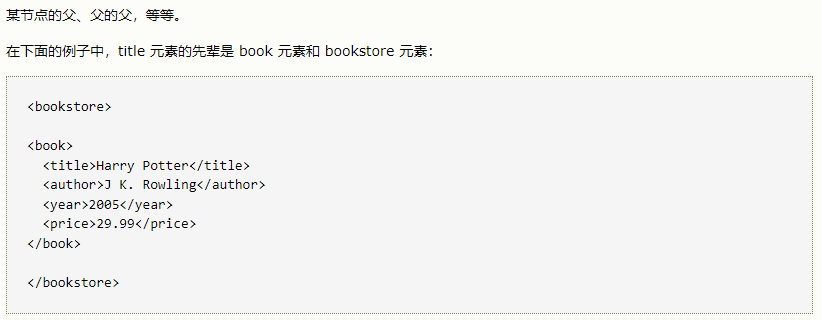
子



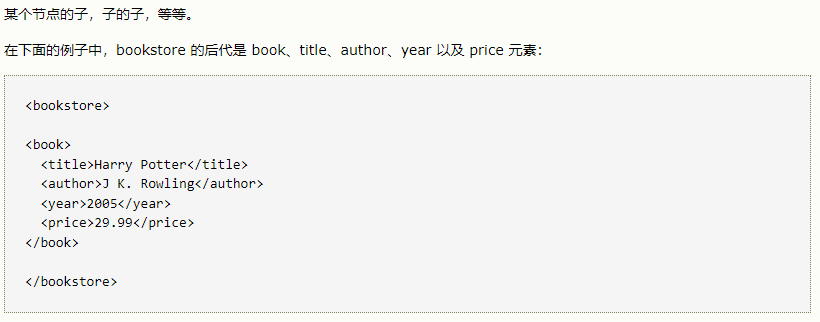
同胞



先辈

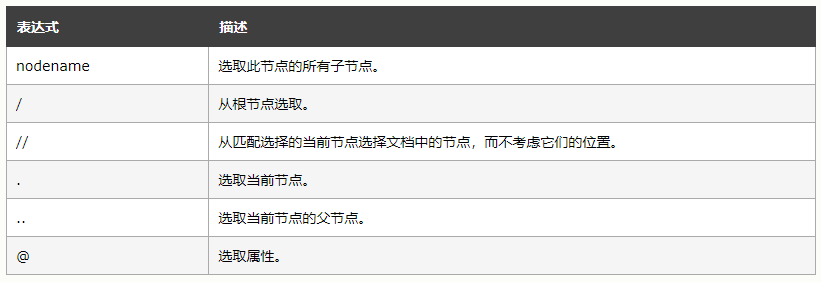


后代



XPath语法

选取节点



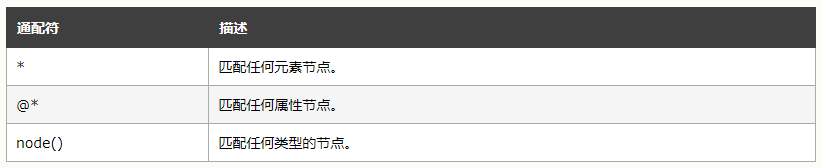


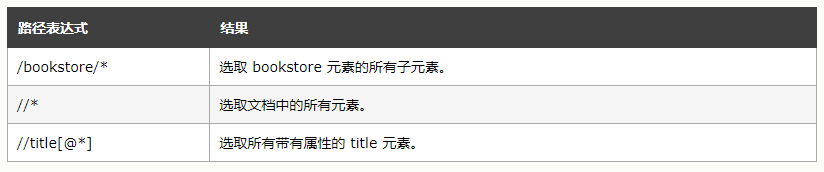
谓语

谓语用来查找某个特定的节点或者包含某个指定的值的节点；谓语被嵌在方括号中



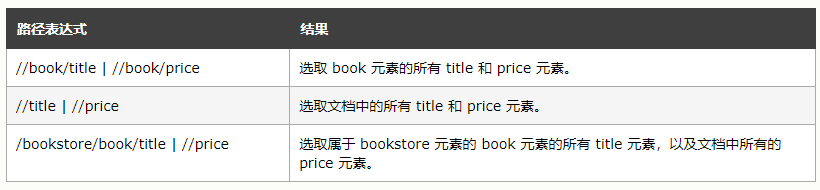
选取未知节点





选取若干路径

通过在路径表达式中使用“|”运算符，您可以选取若干个路径



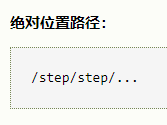
XPath轴

轴可定义相对于当前节点的节点集

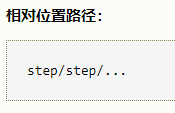


位置路径表达式

绝对路径



相对路径



步

每个步均根据当前节点集之中的节点来进行计算

三元素

轴

定义所选节点与当前节点之间的树关系

节点测试

识别某个轴内部的节点

零个或者更多谓语

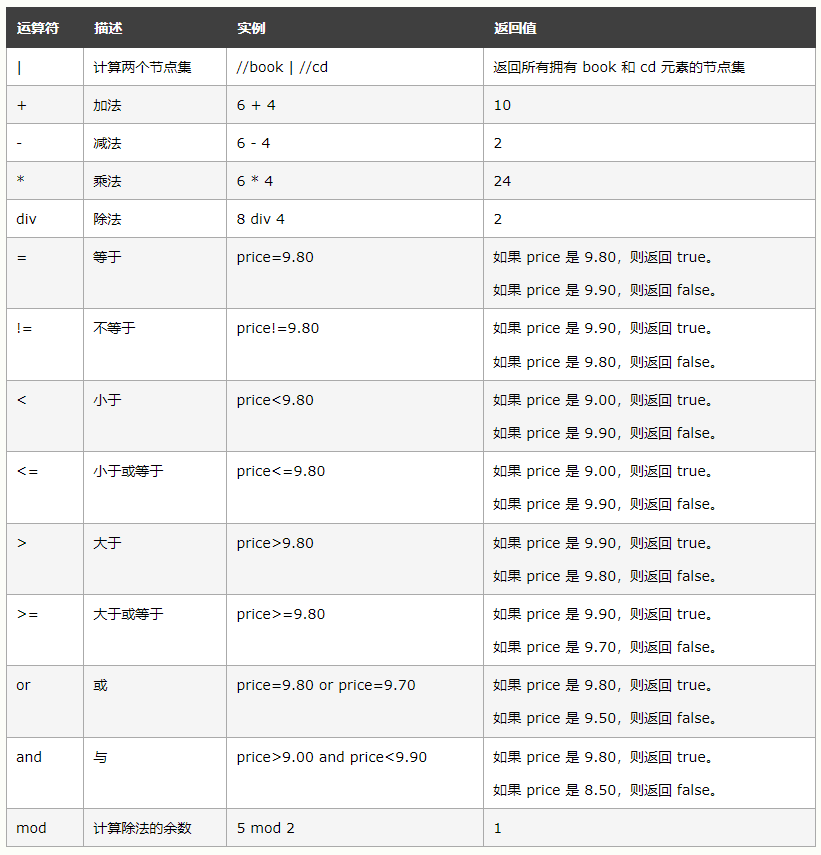
更深入地提炼所选的节点集

语法

轴名称::节点测试[谓语]

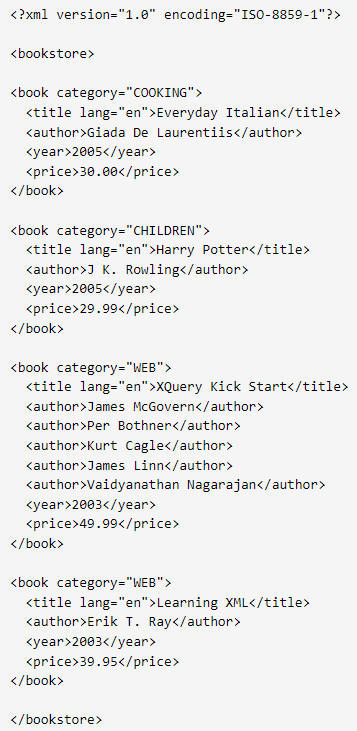


XPath运算符



实例

XML



选取所有 title

/bookstore/book/title

选取第一个 book 的 title

/bookstore/book[1]/title

选取所有价格

/bookstore/book/price/text()

选取价格高于 35 的 price 节点

/bookstore/book[price>35]/price

选取价格高于 35 的 title 节点

/bookstore/book[price>35]/title

* + - * 1. XSL-FO

什么是XSL-FO

XSL-FO 是用于格式化 XML 数据的语言

XSL-FO 指可扩展样式表语言格式化对象（Extensible Stylesheet Language Formatting Objects）

XSL-FO 是一个 W3C 推荐标准

XSL-FO 目前通常被称为 XSL

XSL-FO和格式化有关

XSL-FO 是一种基于 XML 的标记语言，用于描述向屏幕、纸或者其他媒介输出 XML 数据的格式化（信息）

XSL-FO 目前通常被称为 XSL

以前

样式化（Styling）是关于转换信息和格式化信息两方面的信息。在万维网联盟编写他们的首个 XSL 工作草案的时候，这个草案包括了有关转换和格式化 XML 文档的语言语法

目前

后来，XSL 工作组把这个原始的草案分为独立的标准

XSLT，用于转换 XML 文档的语言

XSL 和 XSL-FO，用于格式化 XML 文档的语言

XPath，是通过元素和属性在 XML 文档中进行导航的语言

* 1. 链接技术
     1. XLink
        1. 什么是XLink
           1. XLink 是 XML 链接语言（XML Linking Language）的缩写
           2. XLink 是用于在 XML 文档中创建超级链接的语言
           3. XLink 类似于 HTML 链接 - 但是更为强大
           4. XML 文档中的任何元素均可成为 XLink
           5. XLink 支持简易链接，也支持可将多重资源链接在一起的扩展链接
           6. 通过 XLink，链接可在被链接文件外进行定义
           7. XLink 是 W3C 推荐标准
        2. 常用属性



* + - 1. 语法
         1. 说明

在 XML 文档中，可以使用任何你需要的名称 - 因此对于浏览器来说是无法预知在 XML 文档中可调用何种超级链接元素

在 XML 文档中定义超级链接的方法是在元素上放置可用作超级链接的标记

* + - * 1. 示例



为了访问 XLink 的属性和特性，我们必须在文档的顶端声明 XLink 命名空间

<http://www.w3.org/1999/xlink>

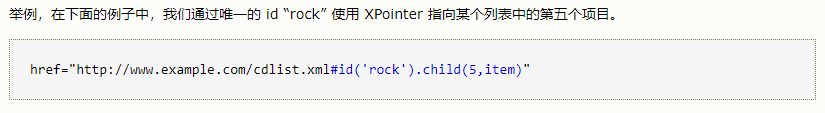
<homepage> 元素中的 xlink:type 和 xlink:href 属性定义了来自 XLink 命名空间的 type 和 href 属性

* + 1. XPointer
       1. 什么是XPointer
          1. XPointer 是 XML 指针文件（XML Pointer Language）的缩写
          2. XPointer 使超级链接可以指向 XML 文档中更多具体的部分（片断）
          3. XPointer 使用 XPath 表达式在 XML 文档中进行定位
          4. XPointer 是 W3C 推荐标准
       2. 语法
          1. 说明

可指向更多具体的内容会更有好处，通过 XPointer 是很容易做到的

假如超级链接指向某个 XML 文档，我们可以在 xlink:href 属性中把 XPointer 部分添加到 URL 后面，这样就可以导航（通过 XPath 表达式）到文档中某个具体的位置了

* + - * 1. 示例



* + 1. 两者的具体不区别和作用
       1. XLink 定义了一套在 XML 文档中创建超级链接的标准方法
       2. XPointer 使超级链接有能力指向 XML 文档中更多具体的部分（片断）
  1. 查询技术
     1. XQuery
        1. 什么是XQuery
           1. XQuery 是用于 XML 数据查询的语言
           2. XQuery 对 XML 的作用类似 SQL 对数据库的作用
           3. XQuery 建立在 XPath 表达式之上
           4. XQuery 被所有主要的数据库引擎支持（IBM、Oracle、Microsoft等等）
           5. XQuery 是 W3C 标准
        2. XQuery和XML
           1. XQuery 是用来从 XML 文档查找和提取元素及属性的语言
           2. 实例

从存储在名为 cd\_catalog.xml 的 XML 文档中的 CD 集那里选取所有价格低于 10 美元的 CD 记录

* + - 1. 语法
         1. 基本语法

规则

XQuery 对大小写敏感

XQuery 的元素、属性以及变量必须是合法的 XML 名称。

XQuery 字符串值可使用单引号或双引号。

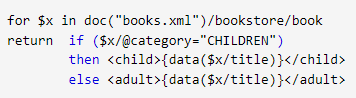
XQuery 变量由 “$” 并跟随一个名称来进行定义，举例，$bookstore

XQuery 注释被 (: 和 :) 分割，例如，(: XQuery 注释 :)

条件表达式

if-then-else

示例



比较

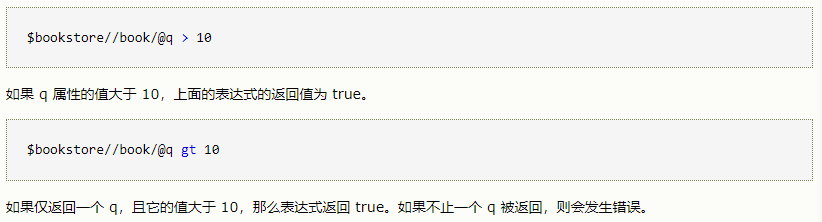
通用比较

=, !=, <, <=, >, >=

值的比较

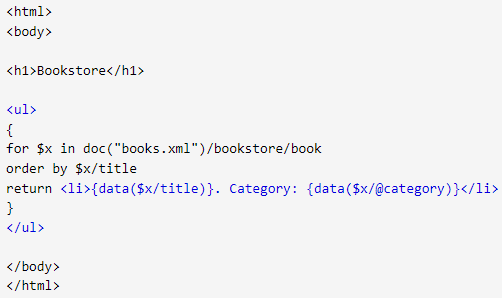
eq、ne、lt、le、gt、ge

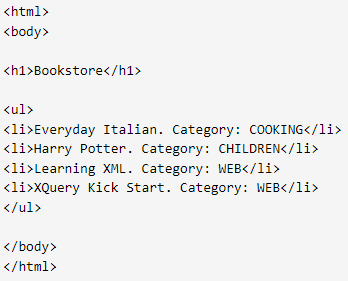
差异示例



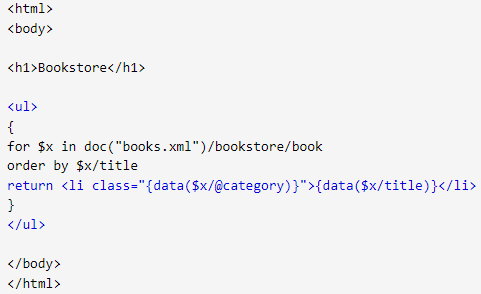
* + - * 1. 添加元素和属性

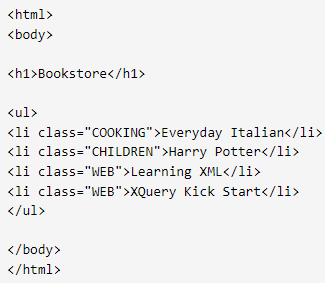
添加HTML元素和文本





向HTML元素添加属性





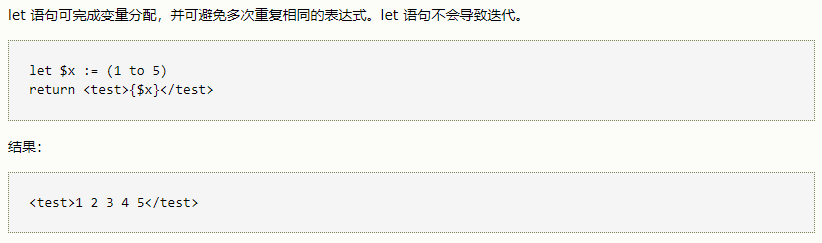
* + - * 1. 选择和过滤

FLWOR表达式

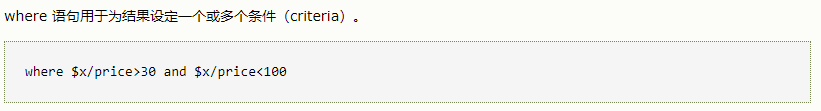
什么是FLWOR

*for* 语句利用循环提取XML文件中的元素。向每个由 in 表达式返回的项目捆绑一个变量

*let*可完成变量分配，并可避免多次重复相同的表达式。let 语句不会导致迭代



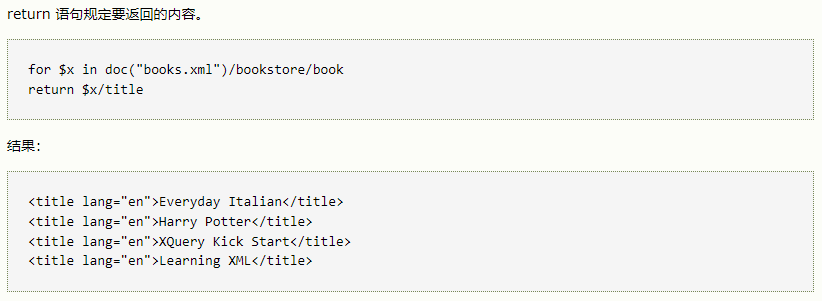
*where* 语句限定一个或多个提取元素的条件



*order by* 语句上设置了排序



*return* 语句规定返回什么内容

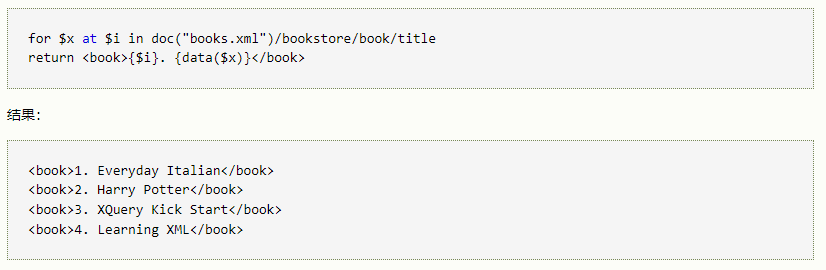


关键词

at

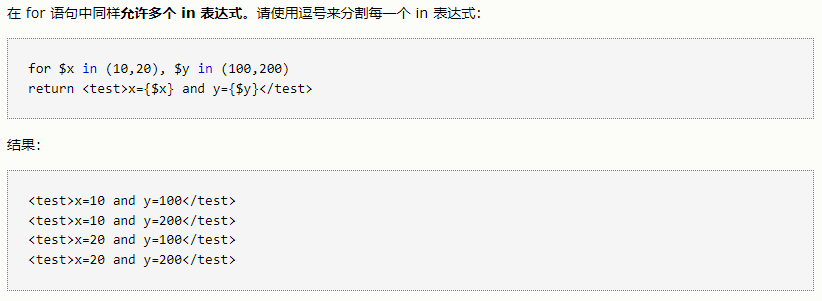
用于计算迭代

示例



in

用于限定范围



to

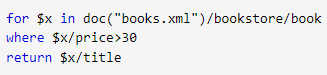
用于限定范围



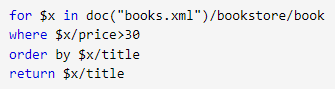
实例

可选取 bookstore 元素下的 book 元素下所有的 title 元素，并且其中的 price 元素的值必须大于 30

使用FLWOR



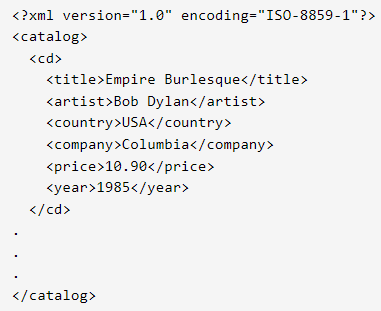
如果需要排序

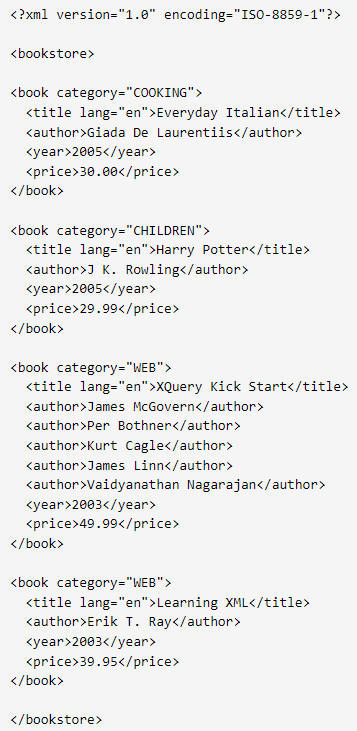


不使用FLWOR

doc("books.xml")/bookstore/book[price>30]/title

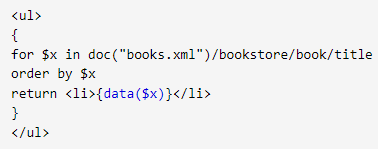
公共的XML文件



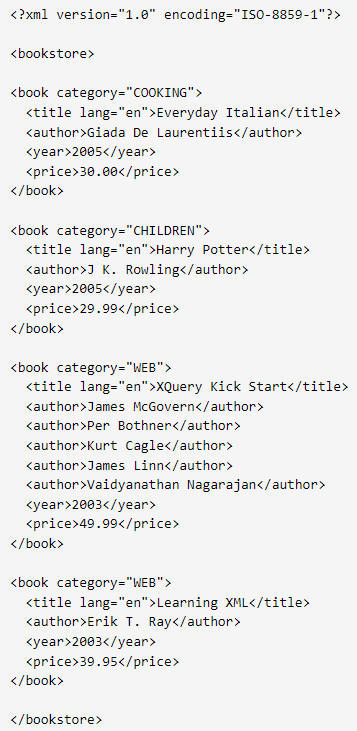


FLWOR + HTML

实例：使用 HTML 列表列出我们的书店中所有的书目



* + - * 1. 公共XML文件



* + - 1. 函数
         1. XQuery 含有超过 100 个内建的函数。这些函数可用于字符串值、数值、日期以及时间比较、节点和 QName 操作、序列操作、逻辑值等
         2. 内建函数

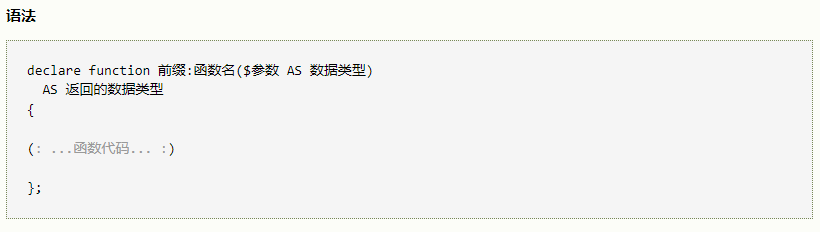
函数经常被通过 fn: 前缀进行调用，例如 fn:string()。不过，由于 fn: 是命名空间的默认前缀，所以函数名称不必在被调用时使用前缀

示例

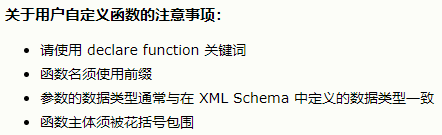


* + - * 1. 自定义函数

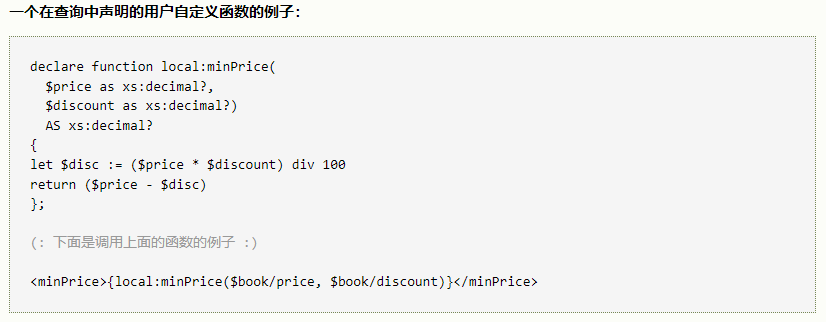
语法



注意事项



示例



* + - 1. XQuery的实际应用举例
         1. 提取信息以便在网络服务中使用
         2. 生成摘要报告
         3. 把 XML 数据转换为 XHTML
         4. 为获得相关信息而搜索网络文档