1. Xml整体概述

一、xml简介

1、侧重于数据的存储与传输，不再侧重于数据如何表现，因此xml如今已经成为了一种跨平台的数据交换格式，一种轻量级的持久化方案。

2、使用xml可以在不同平台、不同系统之间进行数据交换。

3、xml广泛的应用于JAVAEE开发的各个方面，xml还是webservice技术的重要基础。

二、xml竞争对手

1、annotation

2、JSON:是一种轻量级数据交换格式，最早是JavaScript的数据格式，后来慢慢发展成了一种与平台无关的数据交换格式。

三、xml文档分类

1、格式不良好的xml文档：没有完全遵守xml文档定义规则；

2、格式良好但无效的xml文档：完全遵守xml文档定义规则，但是没有使用DTD或schema来定义语义约束；

3、有效的xml文档：完全遵守xml文档定义规则，并且使用了DTD或schema指定了详细的语义约束，内容遵守Schema或DTD所指定的约束。

四、xml文档整体结构

1、有且仅有一个根元素

2、元素必须合理结束

3、元素之间必须合理嵌套

4、元素的属性必须有值

五、xml声明

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

|  |  |  |  |
| --- | --- | --- | --- |
| 简体中文 | 繁体中文 | 西欧字符 | 通用的国际编码 |
| GBK,GB2312 | BIG5 | ISO8859-1 | Unicode、utf-8 |

endcoding属性指定的是读取文件时所采用的字符编码格式，简体中文windows平台保存文件默认的是GBK;

六、xml元素基本规则

1、合法的标签名:标签名遵守的规则：

①、标签名可以由字母、数字、下划线、中划线、冒号和点号组成，但不能以数字、中划线、点号开头；

②、标签名不能包含<、>、，、$等符号；

③、标签名中尽量不要出现冒号":",除非是在使用名字空间；

④、标签名不能以字符“xml”开始；

⑤、标签名不能包含空格。

2、嵌套子元素

3、空元素

Eg、<book />或者<book></book>(一种是自闭合形式，一种是半闭合形式)

空元素只是不能包含子元素，也不可接受字符串内容，但完全可以接受属性，而且可以接受任意多个属性。

4、字符数据：处理特殊字符的两种方式

①、使用实体引用：&lt;(表示小于符号) &gt;(表示大于符号) &amp;(表示and符号) &apos;

(表示英文单引号) &quot;（表示英文双引号）

②、使用CDATA标记：将要改变的文档定义成字符串

语法格式：<![CDATA[文本内容]]>

5、注释：<!—XML注释内容 -->

6、处理指令

一个常用的处理指令xml-stylesheet,用于为xml文件导入样式单。

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

<!--导入css样式单，为xml指定样式 -->

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

<bookList>

<itbook>

<bookName>xml权威指南</bookName>

<bookVersion>第三版</bookVersion>

<author>李刚</author>

</itbook>

</bookList>

7、属性的使用

①、XML元素必须有属性值，属性值必须用引号引起来；

②、同一个xml元素里不能有多个同名属性；

③、多个属性之间没有先后顺序。

注意：数据本身应该存储为元素，使用属性来提供数据无关的额外信息。

1. DTD详解

一、xml引入DTD的方式：

1、内部DTD

<!DOCTYPE 根元素名[

元素描述(<!ELEMENT 元素名 (元素定义)>)

]>

例、

<!DOCTYPE lolPlayer[

<!ELEMENT lolPlayer (area)\*>

<!ELEMENT area (aoly|jsf|cjzd|hsmg|sjzh|ylzz)+>

<!ELEMENT aoly (totalCount,legendary,pentaKill)>

<!ELEMENT totalCount (#PCDATA)>

<!ELEMENT legendary (#PCDATA)>

<!ELEMENT pentaKill (#PCDATA)>

]>

2、外部DTD

<!DOCTYPE 根元素名 SYSTEM "外部DTD的URL">

例、

<!DOCTYPE 根元素名

SYSTEM "http://java.sun.com/xml/ns/javaee">

3、公用DTD

<!DOCTYPE 根元素名 PUBLIC " DTD标识名" "外部DTD的URL">

<!DOCTYPE struts PUBLIC

"-//Apache Software//DTD Struts Configuration 2.3//EN"

"http://struts.apache.org/dtds/struts-2.3.dtd">

二、DTD文档结构

第一行是DTD文档声明

0到多个注释部分

0到多个<!ELEMENT...>定义，每个<ELEMENT> 定义一个xml元素；

0到多个<!ATTLIST...>定义，每个<!ATTLIST...>为xml元素定义一个属性；

0到多个<!ENTITY...>定义，每个<!ENTITY...>定义一个实体；

0到多个<!NOTATION...>定义，每个<!NOTATION...>定义一个符号；

三、定义元素

五种元素类型：

1、任意类型：这种元素既可以是字符串，也可以包含其他子元素，还可以是空元素

eg. <!ELEMENT 元素名 ANY>

注意：应尽量避免使用ANY规则来定义xml元素，因为ANY规则对xml元素几乎没有任何约束；

2、字符串值：这种元素只能是字符串，不能包含其他子元素，也不可以是空元素；

eg. <!ELEMENT 元素名 (#PCDATA)>

3、空元素：这种元素只能是空元素，既不可以包含子元素，也不可以包含字符串值；

eg. <!ELEMENT 元素名 EMPTY>

4、包含子元素：包含子元素的元素比较复杂，因为需要详细定义子元素之间的顺序和子元素出现的次数；

①、有序的子元素

使用英文逗号(,)作为子元素之间的分隔符，则子元素之间必须遵守所定义的顺序。

eg. <!ELEMENT computerBook(name,author,price,introduction)>

②、互斥的子元素

互斥子元素使用竖线（|）分隔，以竖线分隔的多个子元素只能出现其中的一个。

eg. <!ELEMENT computerBook(name|author|price|introduction)>

③、子元素出现的频率

+：表明子元素可以出现1次或多次

\*：表明子元素可以出现0次或多次

?：表明子元素可以出现0次或1次

④、组合子元素

使用圆括号将多个子元素括起来形成元素组，元素组内的各个子元素之间可用英文逗号分隔，也个用竖线分隔。

eg. <!ELEMENT computerBook((name,author)+,price,introduction)>

⑤无序的子元素

eg. <!ELEMENT computerBook(name|author|price|introduction)+>

5、混合类型：指定xml的值只能是几个确定的类型，混合类型是比任意类型更强的约束，但又能提供和任意类型大致相当的功能。

eg. <!ELEMENT 父元素名 (#PCDATA|子元素1|子元素2|子元素3|...)\*> 该DTD语法指定了父元素的内容既可以是普通字符串也可以是各子元素名所指定的子 元素，值得指出的是这里的子元素1、子元素2和子元素3之间的竖线（|）并不表示互斥， 而只是表示这些子元素能无序的重复出现，出现次数不受限制。

四、定义元素属性

语法格式：

<!ATTLIST 属性所属的元素 属性名 属性类型[元素对属性的约束] [默认值]>

“元素对属性的约束”和默认值两个部分是可选的,对于他们的使用有如下几种情况：

①、在没有指定“元素对属性的约束”时，不能为该属性指定“默认值”；

②、当“元素对属性的约束”是#REQUIRED时，不能为该属性指定“默认值”；

③、当“元素对属性的约束”是#IMPLIED时，不能为该属性指定“默认值”；

④、当“元素对属性的约束”是#FIXED时，必须为该属性指定“默认值”；

1、对属性的约束规则

#REQUIRED：必需的属性

#IMPLIED：该属性时可有可无的

#FIXED：该属性的值是固定的，定义是必须指定固定的值

2、定义属性类型

|  |  |
| --- | --- |
| 类型 | 说明 |
| CDATA | 该属性值只能是字符串数据（character data） |
| (en1|en2|en3) | 该属性值是一系列枚举值之一 |
| ID | 该属性值必须是有效的标识符，且该属性值可用于标识该元素，因而该属性值在此xml文档中必须唯一 |
| IDREF | 该属性值必须引用自另一个已有的ID属性值 |
| IDREFS | 该属性值必须引用自多个已有的ID属性值，多个ID属性值之间用空格隔开 |
| NMTOKEN | 该属性值必须是一个合法的xml名称，它也指定了该属性值是字符串数据，但比CDATA具有更强的约束，它表明属性值只能由字符、数字、英文下画线（\_）、英文中画线（-）、英文点号（.）和英文冒号（:）等组成 |
| NMTOKENS | 该属性值可以实多个NMTOKEN，多个NMTOKEN之间用空格隔开 |
| ENTITY | 该属性值是一个外部实体，例如图片文件 |
| ENTITIES | 该属性值是多个外部实体，多个实体之间以空格隔开 |
| NOTATION | 该属性值是在DTD中声明过的符号（NOTATION），这是个将要过期的规范，应尽量避免使用 |
| xml: | 该属性值是一个预定义的xml值 |

五、定义实体

所谓实体引用就是用一个字符串代替另一个字符串；

1、定义普通实体

定义语法格式：<!ENTITY 实体名 *"实体值"*>

使用语法格式：&实体名；

注意：在XML文档中引用实体时，必须以&开始，以英文分号（;）结束。

2、定义参数实体

定义语法格式：<!ENTITY % 实体名 *"实体值"*>

使用语法格式：%实体名；

注意：1、参数实体只能在DTD中使用;

2、在DTD文档中使用参数实体时，必须以英文百分号（%）开始，以英文分号（;）结束。与定义普通实体相比，定义参数实体需要多使用一个百分号（%）。

3、DTD中使用的参数实体必须前向定义，也就是说必须先定义参数实体，然后才能使用该参数实体。

3、定义外部实体

定义语法格式：<!ENTITY 实体名 SYSTEM *"实体值所在文件的URI"*>或

<!ENTITY 实体名 PUBLIC *"公用实体标识名"* *"实体值所在文件的URI"*>

使用语法格式：&实体名；

注意：外部实体对应的文件可以拥有一个文本声明，它类似于XML声明，但可以省略version属性，且不需要指定standalone属性。外部对应文件可以为txt文本。

该外部文件是一个文本文件。 该外部文件是满足XML要求的结构化文档。

4、定义外部参数实体

定义语法格式：

<!ENTITY % 实体名 SYSTEM|PUBLIC *["公用实体标识名"]* *"文件URI"*>

六、定义符号

Xml文档无法处理复杂的数据类型，比如视频、图片等，因此XML文档将不负责处理这些数据，而是通过定义一个符号来标识这些数据。

1、定义普通符号的语法格式：<!NOTATION notation SYSTEM *"value"*>

2、定义公共符号的语法格式：<!NOTATION notation PUBLIC *"name"* *"value"*>

3、符号值的两种类型

①、MIME（多用途Internet邮件扩展）类型：一般说来，通用MIME类型的文件总是由相应的程序负责处理。

②、外部程序所在路径：这种方式直接指定某个外部程序负责处理XML文档中的外部数据。

4、符号的两种用途

①、用来定义未解析实体。

②、作为NOTATION类型的属性的值。

5、未解析实体

①、不满足：外部文件是一个文本文件；外部文件是满足XML要求的结构化文档；两个要求的外部文件不能被xml解析，只能讲该外部文件声明为未解析实体。

②、由于XML解析器不能处理未解析实体所对应的数据，因此XML文档不能像引用普通实体一样直接引用未解析实体，而需要通过ENTITY、ENTITIES类型的属性来调用。

③、定义语法格式如下：

<!ENTITY % 实体名 SYSTEM|PUBLIC [*"标识名"*] *"文件URI"* NDATA notation>

注意：外部实体对应的数据无法直接“插入”XML文档，通常只能作为ENTITY或者ENTITIES类型属性的属性值。

6、ENTITY和ENTITIES类型的属性

定义ENTITY或ENTITIES类型的属性与定义其他类型的属性没有太大的区别，只是在XML文档中使用这些属性时，其值只能是未解析实体。

注意：

一旦声明某个属性的类型是ENTITY或ENTITIES，就意味着该属性值只能是一个或多个未解析实体，而不能是已经解析的实体。对于未解析实体而言，不能通过普通实体引用的方式去引用，它们只能作为ENTITY或ENTITIES类型的属性值。

7、NOTATION类型的属性

NOTATION类型的属性的属性值只能是符号名，而且定义NOTATION类型的属性时必须采用枚举方法列出该属性所支持的值，其语法格式如下：

<!ATTLIST 属性所属的元素 属性名 NOTATION (值1|值2) 约束 默认值>

1. Schema详解

一、schema概述

XML Schema采用XML文档来定义语义约束，比DTD稍微复杂一点，但功能要强大得多。XML Schema支持丰富的数据类型，而且允许开发者自定义数据类型，因此完全可以处理更复杂的语义约束场景。

二、schema入门

1、XML schema根元素

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

<xs:schema xmlns:xs=*"http://www.w3.org/2001/XMLSchema"* targetNamespace=*"http://www.example.org/NewXMLSchema"*

xmlns:tns=*"http://www.example.org/NewXMLSchema"*

elementFormDefault=*"qualified"*

attributeFormDefault =*"qualified"*>

</xs:schema>

①、上述Schema中指定了xmlns:xs=*"http://www.w3.org/2001/XMLSchema"*，这是因为XML Schema本身也是XML文档，也需要语义约束。

②、xmlns[:xxx]属性用于为指定XML文档引入语义约束。该属性的属性值就是该文本所使用的语义约束对应的命名空间。xxx是一个任意的标识名，该标识名将作为该语义约束所定义的所有元素的前缀（前缀和标签名之间以英文冒号隔开），用于代表该语义约束所对应的命名空间。

提示：

不同语义约束（Schema或DTD）下可能包含同名的元素，为了在同一份XML文档中使用不同语义约束下的同名元素，需要在这些元素前增加特定的前缀，不同的前缀实质上代表了不同的语义约束。

③、targetNamespace指定该Schema对应的目标命名空间

④、不管是XML文档本身，还是XML Schema文档，它们的根元素都可以接受任意多个xmlns[:xxx]属性，其中[:xxx]部分是可选的，xxx可以是任意合法的标识名。xmlns[:xxx]=*"schemaNamespace"*指定使用*schemaNamespace*命名空间下的Schema组件时，应该使用xxx前缀作为限定。如果指定该属性时省略了[:xxx]部分，那就是说使用*schemaNamespace*命名空间下的Schema组件时，无须使用任何前缀作为限定。

⑤、elementFormDefault：该属性值可以是*qualified*或*unqualified*，用于指定XML文档使用该Schema中定义的局部元素时是否必须用命名空间限定。

⑥、attributeFormDefault：该属性值可以是*qualified*或*unqualified*，用于指定XML文档使用该Schema中定义的局部属性时是否必须用命名空间限定。

2、在XML中引用无命名空间的Schema

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

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

xsi:noNamespaceSchemaLocation=*"noNamespaceBook.xsd"*>

<computerBook>疯狂Java 讲义</computerBook>

</booklist>

xsi:noNamespaceSchemaLocation：该属性值用于指定XML Schema文件的URI，既 可是绝对的URL地址，也可是位于磁盘上的相对路径。

3、在XML中引用有命名空间的Schema

<booklist

xmlns=*"http://www.leach/schema/booklist"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xmlns:crazy=*"http://www.leach/chema/crazy"*

xsi:SchemaLocation=*"http://www.leach/schema/booklist testSchema/booklist.xsd*

*http://www.leach/chema/crazy testSchema/crazy.xsd"*>

<computerBook>疯狂Java 讲义</computerBook>

</booklist>

①、每引入一个有命名空间的XML Schema就为XML根元素增加一个xmlns[:xxx]属性，其中[:xxx]是可选的，但最多只能有一个xmlns属性，其他的都必须是形如xmlns:xxx的属性，而且xxx应该互不相同。表明使用各命名空间对应的XMLSchema所定义的Schema组件时应该使用xxx前缀作为限定。

②、如果XML根元素中已有xsi:schemaLocation属性（xsi前缀可变），则在该属性值后为该XMLSchema追加一项，追加项要保持schemaNamespace schemaURI的格式。如果XML根元素中还没有xsi:schemaLocation属性，则为其增加该属性，并设置属性值为schemaNamespace schemaURI。

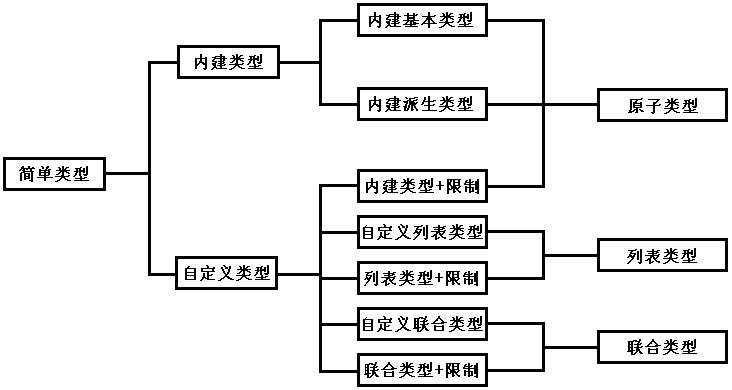
提示：

noNamespaceSchemaLocation和SchemaLocation属性都是 http://www.w3.org/2001/XMLSchema-instance命名空间对应的XML Schema所定义的属性，因 此使用该属性时需要增加xsi前缀作为限定*。*

三、理解schema数据类型

1、从宏观上来看，Schema支持的数据类型分为两大类：

|  |  |
| --- | --- |
| 简单类型： | 既适合作为XML元素的类型，也适合作为XML属性的类型。 |
| 复杂类型： | 只能作为XML元素的类型。 |

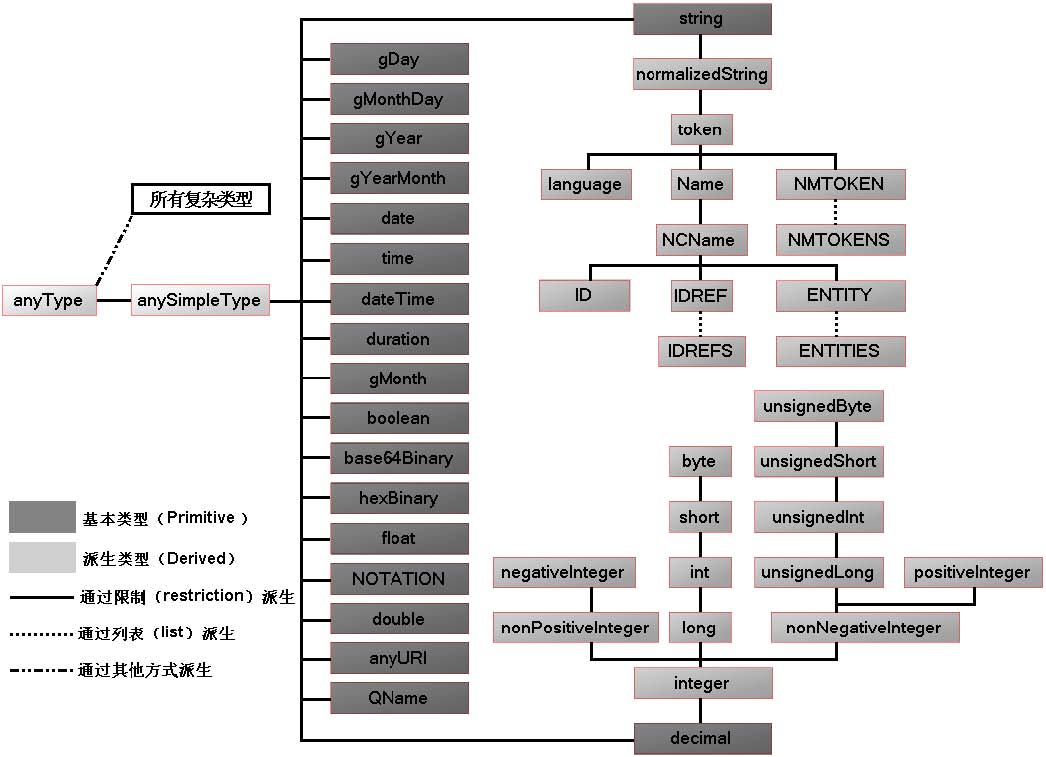


2、Schema支持的派生方式大致上有如下3种：

|  |  |
| --- | --- |
| 限制： | 使用<restriction.../>元素为原有类型增加一个或多个额外的约束。 |
| 列表： | 使用<list.../>元素定义。通过这种方式产生的类型也称为列表类型。 |
| 联合： | 使用<union.../>元素定义，用于将多个已有的数据类型联合起来。 |

注意：其实内建派生类型中有3种是列表类型：IDREFS、ENTITIES和NMTOKENS三者都是采用列表方式派生的新类型，这一点需要读者注意。内建派生类型中没有包含联合类型。

3、schema内置类型



4、数值类型

|  |  |  |
| --- | --- | --- |
| 浮点数 | *float* | 代表32位的单精度浮点数,无须添加f或F后缀 |
| *double* | 代表64位的双精度浮点数 |
| 精确小数 | *decimal* | 支持18位有效的小数, 不支持使用科学计数法, 不支持-INF、INF和NaN等特殊值 |
| 整数 | *integer* | 代表一个任意大的整数,由decimal派生而来 |
| *int* | 代表32位的有符号整数，由interger派生而来 |

5、字符串及相关类型

|  |  |
| --- | --- |
| *string* | 它会原封不动地保留字符串内容前面、后面及中间的所有空白。 |
| *normalizedString* | 它会将字符串内容中包含的换行、制表符和回车符都替换成空白。 |
| *token* | 它会将字符串内容中包含的换行、制表符和回车符都替换成空白，并自动删除字符串前后的空白，如果字符串中间包含多个连续的空白，多个连续的空白会被压缩成单个空白。 |
| *Name* | 要求字符串内容是一个合法的XML标签名，即该类型的值可以由字母、数字、下画线（\_）、中画线（-）、冒号（:）和点号（.）组成，且不能以数字、中画线（-）或点号（.）开头。 |
| *NCName* | 要求字符串内容是一个不带命名空间前缀的XML标签名。因此它与Name类型的区别在于，它不能包含冒号（:）。 |
| *QName* | 要求字符串内容是一个带命名空间前缀的XML标签名，但它允许省略命名空间前缀——省略命名空间前缀后，不能以冒号开头。一旦在QName类型的值中使用了命名空间前缀，则该命名空间前缀必须有对应的命名空间。 |
| ID、IDREF、IDREFS、ENTITY、ENTITIES、NMTOKEN和NMTOKENS等类型，与前面DTD中对应的类型完全相同。 | |

6、日期、时间类型

|  |  |  |
| --- | --- | --- |
| 数据类型 | 格式 | 说明 |
| *date* | YYYY-MM-DD | 日期 |
| *time* | hh:mm:ss.sss | 时间，sss表示毫秒数 |
| *dateTime* | YYYY-MM-DDT hh:mm:ss.sss | 日期时间，T是必需的，是日期时间的分隔符 |
| *gYear* | YYYY | 年份 |
| *gYearMonth* | YYYY-MM | 年月 |
| *gMonth* | --MM | 月份，- -是必需的 |
| *gMonthDay* | --MM-DD | 月日，- -是必需的 |
| *gDay* | ---DD | 日，---是必需的 |
| *duration* | PnYnMnDTnHnMnS | P是固定的，YMDHMS—年月日时分秒，T为分隔符，YMDHM前的n必须是整数，S前的n可以是小数 |
| 注意：  上面列出的前八个数据类型的值后面添加Z，表示UTC时间；  上面列出的前5个数据类型的值前面可添加负号，表示公元前。 | | |

7、其他数据类型

|  |  |  |
| --- | --- | --- |
| 布尔类型 | *boolean* | 只能接受true、false（大小写不能改变）、0和1等4个值 |
| anyURI类型 | *uri* | 值必须是一个合法URI |
| 二进制数据 | *hexBinary* | 以十六进制数字保存的二进制数据，因此该类型的值只能由0～9、a～f和A～F（代表10～15）这几种字符组成。 |
| *Base64Binary* | 以Base64编码保存的任意二进制数据。由a～z、A～Z、0～9和加号（+）等字符组成,长度必须是64的倍数 |

四、使用限制派生新类型

1、概念及相关约束

这种派生方式以某个现有类型为基础，通过使用<restriction.../>元素来添加一个或多个约束，从而派生出新类型。

XML Schema规范共推荐了12种约束，用于对原有数据类型添加额外的限制：

|  |  |  |
| --- | --- | --- |
| 范围约束 | minInclusive | 定义元素或属性的下界值，元素或属性的值大于等于该值 |
| maxInclusive | 定义元素或属性的上界值，元素或属性的值小于等于该值 |
| minExclusive | 定义元素或属性的下界值，元素或属性的值必须大于该值 |
| maxExclusive | 定义元素或属性的上界值，元素或属性的值必须小于该值 |
| 长度约束 | length | 定义元素或属性的值的字符长度 |
| minLength | 字符串的最小长度 |
| maxLength | 字符串的最大长度 |
| 精度约束 | fractionDigits | 对于任意精度的十进制数起作用，用于定义小数点后的最大位数。 |
| totalDigits | 指定decimal及其派生类型的数值最大能有几位数（包括小数和整数部分）。 |
| 枚举约束 | enumeration | 指定元素或属性的值是枚举值，只能是一系列值其中之一。 |
| 正则表达式约束 | patten | 数据类型的值必须匹配的指定的正则表达式，pattern值必须是一个正则表达式。 |
| 空白处理 | whiteSpace | 对字符串中的空白的处理方式，其值可以为：preserve，保留字符串中的空白。 |

2、指定基类型的方式

程序清单：assignBase.xsd

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

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

elementFormDefault=*"qualified"*

attributeFormDefault =*"qualified"*>

<xs:simpleType name=*"age\_Type"*>

<xs:restriction base=*"xs:int"*>

<xs:maxInclusive value=*"100"*/>

<xs:minInclusive value=*"0"*/>

</xs:restriction>

</xs:simpleType>

<xs:element name=*"age"* type=*"age\_Type"*/>

</xs:schema>

程序清单：assignBase.xml

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

<age xmlns:xs=*"http://www.leach/schema/booklist"*

xs:noNamespaceSchemaLocation =*"assignBase.xsd"*>34</age>

新类型的定义需要通过<simpleType>和<complexType>元素来指定，通过<restriction>元素来指定基类型，通过约束元素来加以具体的约束条件。

|  |  |  |
| --- | --- | --- |
| <restriction> | id | 指定该<restriction.../>元素的唯一标识 |
| base | 表明该<restriction.../>元素以哪种已有的类型作为基类型 |
| <simpleType>/<complexType> | id | 指定元素的唯一标识 |
| name | 为新的数据类型指定数据类型名 |

3、指定元素或属性类型的两种方式

1. 使用type属性指定元素或属性的数据类型；
2. 使用<simpleType.../>或<complexType.../>子元素指定元素或属性的数据类型；

五、使用<List…/>派生列表类型

1、概念、语法及示例

Schema可以由任何类型派生出对应的列表类型，而且这种列表类型不仅可用于指定属性类型，也可用于指定元素的类型。

Schema内置类型中的IDREFS是IDREF的列表类型，ENTITIES是ENTITY的列表类型，NMTOKENS是NMTOKEN的列表类型。

派生类表类型的两种方式 ：

1. 、为<list.../>元素的itemType属性指定列表元素的数据类型。
2. 、为<list.../>元素增加<simpleType.../>子元素来指定列表元素的数据类型。

程序清单：assignBase.xsd

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

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

<xsi:simpleType name=*"age\_List\_Type"*>

<xsi:list>

<xsi:simpleType>

<xsi:restriction base=*"xsi:int"*>

<xsi:maxInclusive value=*"100"*/>

<xsi:minInclusive value=*"0"*/> </xsi:restriction>

</xsi:simpleType>

</xsi:list>

</xsi:simpleType>

<xsi:element name=*"agelist"* type=*"age\_List\_Type"*/>

</xsi:schema>

或者：<xsi:simpleType name=*"age\_Type"*>

<xsi:restriction base=*"xsi:int"*>

<xsi:maxInclusive value=*"100"*/>

<xsi:minInclusive value=*"0"*/> </xsi:restriction>

</xsi:simpleType>

<xsi:simpleType name=*"age\_List\_Type"*>

<xsi:list itemType=*"age\_ Type"*></xsi:list>

</xsi:simpleType>

程序清单：list.xml

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

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

xsi:noNamespaceSchemaLocation=*"list.xsd"*>

57 12 23 9 68

</agelist>

2、限制列表类型

Schema允许为列表类型增加如下几类约束：

•长度约束：length、maxLength和minLength。

•枚举约束：enumeration。

•正则表达式约束：pattern。

•空白处理：whiteSpace。

六、使用<union../>派生联合类型

1、概念、语法及示例

Schema允许使用<union.../>元素将多个简单类型联合成新的类型，使用<union.../>元素创建联合类型时，需要指定一个到多个简单类型，组成联合类型的数据类型称为联合类型的成员类型，任意数量的原子类型或列表类型都可作为联合类型的成员类型，联合类型的值可以是任意一种成员类型的值。派生联合类型的两种方式：

①、为<union.../>元素的memberTypes属性指定一个到多个简单类型，多个简单类型之间以空格隔开。

②、为<union.../>元素增加一个到多个<simple-Type.../>子元素，每个<simpleType.../>子元素指定一个简单类型。

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

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

<xsi:simpleType name=*"price"*>

<xsi:restriction base=*"xsi:decimal"*>

<xsi:maxExclusive value=*"100"*/>

<xsi:minExclusive value=*"0"*/>

<xsi:totalDigits value=*"2"*/>

</xsi:restriction>

</xsi:simpleType>

<xsi:simpleType name=*"punish"*>

<xsi:restriction base=*"xsi:date"*>

<xsi:maxExclusive value=*"2017-07-05"*/>

<xsi:minExclusive value=*"2012-01-01"*/>

</xsi:restriction>

</xsi:simpleType>

<!-- 派生联合类型 -->

<xsi:simpleType name=*"pirce\_punish\_type"*>

<xsi:union memberTypes=*"price punish"*></xsi:union>

</xsi:simpleType>

<xsi:element name=*"agelist"* type=*"pirce\_punish\_type"*/>

</xsi:schema>

2、限制联合类型

Schema允许为联合类型增加如下两类约束：

•枚举约束：enumeration。 •正则表达式约束：pattern。

注意：

对于列表类型应用enumeration和pattern约束时，是对列表内容整体（多个列表项及括空格组成的整体）起作用，而不是对单个的列表元素起作用。同样为联合类型添加的约束是对整个联合类型的值起作用。

七、列表和联合结合使用

<union.../>元素可以把多个已有的类型联合成联合类型，因此它的成员类型既可以是原子类型，也可以是列表类型，还可以是联合类型，功能非常丰富。

<list.../>元素可以由已有的数据类型派生出对应的列表类型，列表类型的列表元素的类型既可以是原子类型，也可以是联合类型，但不可以是列表类型，也不可以是其成员类型包括列表类型的混合类型。

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

<xsi:schema xmlns:xsi=*"http://www.w3.org/2001/XMLSchema"*

elementFormDefault=*"qualified"* attributeFormDefault =*"unqualified"*>

<!-- 从int 类型派生的ageType -->

<xsi:simpleType name=*"ageType"*>

<xsi:restriction base=*"xsi:int"*>

<xsi:maxExclusive value=*"100"*/>

<xsi:minExclusive value=*"0"*/>

</xsi:restriction>

</xsi:simpleType>

<!-- 从string 类型派生的nameType -->

<xsi:simpleType name=*"nameType"*>

<xsi:restriction base=*"xsi:string"*>

<xsi:maxLength value=*"20"*/>

<xsi:minLength value=*"4"*/>

</xsi:restriction>

</xsi:simpleType>

<!-- 以ageType 和nameType 联合成联合类型 -->

<xsi:simpleType name=*"ageType\_nameType"*>

<xsi:union memberTypes=*"ageType nameType"*></xsi:union>

</xsi:simpleType>

<!-- 由ageType 派生出对应的列表类型 -->

<xsi:simpleType name=*"ageListType"*>

<xsi:list itemType=*"ageType"*></xsi:list>

</xsi:simpleType>

<!-- 由nameType 派生出对应的列表类型 -->

<xsi:simpleType name=*"nameListType"*>

<xsi:list itemType=*"nameType"*></xsi:list>

</xsi:simpleType>

<!-- 将两个列表类型联合成一个联合类型 -->

<xsi:simpleType name=*"ageList\_nameList"*>

<xsi:union memberTypes=*"ageListType nameListType"*></xsi:union>

</xsi:simpleType>

<!-- 将两个联合类型联合成一个新的联合类型 -->

<xsi:simpleType name=*"further\_Type"*>

<xsi:union memberTypes=*"ageType\_nameType ageList\_nameList"*></xsi:union>

</xsi:simpleType>

<!-- 将一个列表类型、一个联合类型联合成一个新的联合类型 -->

<xsi:simpleType name=*"further\_mix\_Type"*>

<xsi:union memberTypes=*"nameListType ageList\_nameList"*></xsi:union>

</xsi:simpleType>

<!-- 由ageType\_nameType 派生出对应的列表类型 -->

<xsi:simpleType name=*"ageType\_nameType\_list"*>

<xsi:list itemType=*"ageType\_nameType"*></xsi:list>

</xsi:simpleType>

</xsi:schema>

11、阻止派生新的简单类型

Schema允许将自定义类型作为“最终类型”，从而不允许从该类型再次派生新的类型。

1、使用final属性

Schema允许为<simpleType.../>元素指定final属性来限制派生新的类型，该属性可以接受如下几个值：

•*#all*：限制该类型以任何形式派生新的类型。它指定该<simpleType.../>元素定义的类型既不可通过增加约束派生，也不可派生出对应的列表类型，也不可作为联合类型的成员类型。

•*restriction*、*list*和*union*的组合：可以是这三个中的任意一个、两个或者三个（与#all相同），分别用于限制对应的派生方式。

•*""*：默认方式。

2、为约束指点fixed属性

Schema还允许为任何约束指定fixed属性，该属性的值只能是true或false。这个属性用于阻止相应约束的值被修改——在由一个已有的类型派生新的类型时，如果新类型添加了和原类型相同的约束，而且约束名相同，则新的约束值会覆盖原有的约束值，但如果为原类型的对应约束指定了fixed="true"，则可阻止这种覆盖。

1. schema高级知识

一、使用anyType 定义任意类型

anyType是所有简单类型和复杂类型的基类型，它通常用于派生新的类型，而不是直接用来定义元素。当某个元素的类型定义为anyType时，该元素的内容完全不受限制，它既可包含子元素，也可包含字符串内容，还可添加任何属性（只要Schema中定义过该属性）。

当指定某个元素的类型是anyType时，就意味着该元素的内容不受任何限制，这显然与定义语义约束的初衷不符，因此应尽量避免将某个元素的类型定义为anyType。

二、定义复杂类型

复杂类型专门为元素而准备——只有元素的类型才能是复杂类型。属性的类型只能是简单类型，而元素的类型则既可以是简单类型，也可以是复杂类型。

|  |  |
| --- | --- |
| 复杂类型的元素 | 1、元素内容是简单类型值，但元素包含属性——称为包含简单内容的复杂类型。 |
| 2、包含子元素的元素，空元素或混合内容的元素，不管它们是否包含属性，都是复杂类型的——称为包含复杂内容的复杂类型。 |
| 简单类型的元素 | 1、元素内容是简单类型值。 |
| 2、元素不能包含属性。 |

定义复杂类型使用<complexType.../>元素，有两种方式：

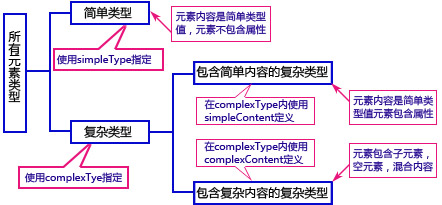
①、定义全局命名复杂类型：将<complexType.../>元素作为<schema.../>、<redefine.../>元素的子元素使用，需要指定name属性，其值就是该复杂类型的类型名。

②、定义局部匿名复杂类型：将<complexType.../>元素作为<element.../>元素的子元素使用，无须指定name属性，仅用于指定其父元素<element.../>所定义的元素的类型。

为了具体地定义复杂类型所包含的内容，<complexType.../>元素可以接受如下两个子元素：

|  |  |
| --- | --- |
| <simpleContent.../> | 如果元素内容是简单类型的值，但元素包含属性，则使用该元素定义复杂类型。该元素用于定义包含简单内容的复杂类型。 |
| <complexContent.../> | 对于包含子元素的元素，空元素或混合内容的元素，不管它们是否包含属性，其类型都必须使用该元素定义。该元素用于定义包含复杂内容的复杂类型。 |

元素类型的整体分类，以及各类型定义所需何种元素：



1、定义复杂类型的方式

定义包含简单内容的复杂类型使用<simpleContent.../>，<simpleContent.../> 有且只能有一个如下的子元素（二者不能共存）：

|  |  |
| --- | --- |
| <restriction.../> | 对已有类型进行限制从而派生新的复杂类型。 |
| <extension.../> | 对已有类型进行扩展从而派生新的复杂类型。 |

在使用<simpleContent.../>元素定义元素类型时，不管是使用<restriction.../>还是<extension.../>子元素来派生新类型，都必须指定一个基类型，这个基类型既不能是包含子元素的类型，也不能是混合内容类型，也不能是空元素类型。换句话说，这个基类型要么是简单类型，要么是使用<simpleContent.../>元素定义的复杂类型。

2、扩展简单类型（使成为包含简单内容的复杂类型）

使用<extension.../>元素能以扩展简单类型的方式来派生复杂类型，这种扩展包括添加属性或属性组等方式。

注意：

为简单类型增加限制后派生的新类型依然是简单类型，而不是复杂类型。对于简单类型而言，只能通过扩展来派生复杂类型。

3、包含属性的两种方式

Schema支持如下两种语法来指定属性属于某个元素：

①、局部属性：直接把属性放在<complexType.../>内定义，该局部属性只能属于当前复杂类型。

②、全局属性：把属性作为<schema.../>的子元素定义，该全局属性可属于多个复杂类型。通过为<attribute.../>元素指定ref属性建立全局属性和复杂类型之间的关联。

使用<attribute.../>元素定义属性时可以指定如下两个常用属性：

①、fixed：为该属性指定一个固定值。

②、default：为该属性指定一个默认值。

注意：不能在同一个<attribute.../>元素中同时使用这两个属性。

<attribute.../>元素可以使用use属性，需满足如下条件：

①、<attribute.../>元素里指定了ref属性。

②、采用<attribute.../>元素定义局部属性时。

use属性用于指定该复杂类型对该属性的要求，支持如下几个属性值：

|  |  |
| --- | --- |
| *optional* | 指定该属性是可选的，即它是一个可有可无的属性。这是该属性的默认值。 |
| *prohibited* | 指定该属性是被禁止的，表明该属性不能出现 |
| *required* | 指定该属性是必需的 |

4、扩展包含简单内容的复杂类型

5、限制包含简单内容的复杂类型

限制包含简单内容的复杂类型主要可从3个方面入手：

①、为元素内容增加进一步的约束；

②、为元素的属性类型增加进一步的约束；

③、删除某些属性。

6、限制anyType派生新类型

Schema为定义子元素提供了如下3个元素：

|  |  |
| --- | --- |
| <sequence.../> | 该元素包含的所有子元素必须按定义的顺序出现； |
| <choice.../> | 该元素包含的所有子元素只能出现其中之一； |
| <all.../> | 该元素包含的子元素能以任意顺序出现。 |

Schema约定：如果某个复杂类型是由限制anyType派生出来的，那么在定义该复杂类型时可以省略<complexContent.../>和<restriction.../>元素，而直接在<complexType.../>元素内使用<sequence.../>、<choice.../>、<all.../>和<attribute.../>来定义元素和属性。

使用<all.../>元素包含子元素有如下几个限制：

①、<all.../>元素内包含的子元素不能重复出现，最多只能出现一次；

②、<all.../>元素不能与<sequence.../>和<choice.../>元素同时出现，而且不能放在<sequence.../>和<choice.../>元素内部使用，通常只能作为<complexContent.../>或<complexType.../>元素的顶级子元素使用；

③、<choice.../>和<sequence.../>两个元素可以相互嵌套使用，<choice.../>和<sequence.../>元素包含的<element.../>元素可指定maxOccurs属性，用于指定该元素允许出现的次数，<all.../>元素也可指定maxOccurs属性，但其属性值只能是0或1。

7、包含子元素的两种方式

在Schema中定义元素有两种方式：

①、全局元素：将<element.../>元素放在<schema.../>元素的根元素下定义；

②、局部元素：将<element.../>元素放在<se-quence.../>、<choice.../>或<all.../>等3个元素里定义。

使用<element.../>定义元素时可以指定如下3个常用属性：

|  |  |
| --- | --- |
| fixed | 为该元素指定一个固定值 |
| default | 为该元素指定一个默认值 |
| nillable | 该属性值只能是true或false。用于指定是否可以将显式的零值分配给该元素，该属性只对元素内容有效，而对元素属性无效。默认值为false。 |

Schema还允许为<element.../>元素添加ref属性，以引用一个已有的全局元素，即可将被引用的全局元素变成当前元素的子元素。

8、空元素类型

空元素类型用于定义元素内容为空或空字符串的元素，但该元素可以接受属性。

定义空元素类型有两种方式：

①、扩展长度为0的字符串：如果该元素不需要包含属性，那么直接使用长度为0的字符串类型定义该元素即可。

②、限制anyType：限制anyType时不定义任何子元素，只定义所需属性即可。

实际开发中多采用第二种方式。

9、混合内容类型

Schema中定义混合内容类型只需为<complexType…/>元素增加mixed="true"即可。

三、限制和扩展复杂类型

1、限制空元素类型

从如下两方面入手：

•对指定属性增加进一步的约束。

•删除某个属性。

2、扩展空元素类型

扩展空元素类型可以从如下3个方面入手：

•为原有类型增加属性：派生出来的新类型依然是空元素类型。

•为原有类型增加子元素：派生出来的新类型将是包含子元素的类型。

•为原有类型增加mixed="true"：派生出来的新类型将是混合内容类型。

3、限制包含子元素的类型

限制包含子元素的类型可以从如下几个方面入手：

•可以对指定属性的类型增加进一步约束。

•可以对指定子元素的类型增加进一步约束。

•可以删除指定属性。

•可以删除指定元素。

注意：

派生类型中为子元素和属性的类型增加进一步约束时，新增的约束必须在基类型的约束范围之内，不可以超出基类型里的约束。例如基类型里name属性的类型是token，则派生类型中name属性的类型只能是token类型及其增加限制后派生的新类型。对于删除子元素和属性而言，删除子元素的实质是将maxOccurs和minOccurs同时设为0，而删除属性的实质是将use属性设为prohibited。因此如果希望删除某个子元素，则基类型中定义该子元素时必须指定minOccurs="0"；而如果希望删除某个属性，则基类型中定义该属性时一定不可以指定use="required"。

4、扩展包含子元素的类型

扩展包含子元素的类型可从如下两个方面入手：

•为基类型增加新的子元素。

•为基类型增加新的属性。

5、限制混合内容类型

限制混合内容类型与限制包含子元素的类型的方式基本相同。不过在这种情况下，由于基类型是混合内容类型，因此派生类型也可以是混合内容类型，当然，也可以不再是混合内容类型——如果派生类型没有指定mixed="true"，则派生类型将不再是混合内容类型。

6、扩展混合内容类型

扩展混合内容类型与扩展包含子元素的类型的方式基本相同，只是通过扩展混合内容类型派生出来的新类型必须是混合内容类型，也就是说必须保留mixed="true"。

四、阻止派生新的类型

为了阻止派生类型的再次派生，Schema允许在定义<complexType.../>元素时指定final属性，属性可以是如下属性值：

|  |  |
| --- | --- |
| *#all* | 阻止以任何方式来以该类型为基类型进行派生 |
| *extension* | 阻止以扩展方式来以该类型为基类型进行派生 |
| *restriction* | 阻止以限制方式来以该类型为基类型进行派生 |
| *extension*和*restriction*的组合 | 阻止以限制和扩展方式来以该类型为基类型进行派生。该属性值等同于#all。 |

五、通配符

在某些情况下，如果无法确定指定元素还需要包含哪些子元素、哪些属性，可以使用通配符。Schema支持使用<any.../>元素作为元素通配符，使用<anyAttribute.../>元素作为属性通配符。

1、元素通配符

Schema中使用<any.../>元素作为元素通配符，<any.../>元素可以代表任何元素。也就是说，<any.../>元素出现的位置可以使用任何元素代替。<any.../>元素可接受如下几个属性：

|  |  |  |  |
| --- | --- | --- | --- |
| id | 指定该元素的唯一标识，通常无须指定。 | | |
| maxOccurs | 指定该通配符所代表的元素最多可以出现的次数，该属性值可以是任何非负整数或"unbounded"字符串。 | | |
| minOccurs | 指定该通配符所代表的元素最少可以出现的次数，该属性值可以是任何非负整数。 | | |
| namespace | 指定可代替该元素通配符的元素必须来自哪个空间，该属性可接受如下属性值： | | |
| ##any | | 来自任何命名空间的元素都可以代替该元素通配符 |
| ##other | | 来自该元素的父元素的目标命名空间之外的其他命名空间的元素才可代替该元素通配符。 |
| ##local | | 无命名空间限定的元素可代替该元素通配符。 |
| ##targetNamespce | | 来自该元素的父元素的目标命名空间的元素才可代替该元素通配符。 |
| 命名空间URI | | 只有来自该命名空间的元素才可代替该元素通配符。 |
| 上面多个值的列表 | | 值列表中的任意一个所对应命名空间的元素都可。 |
| processContents | 指定应用程序或XML处理器如何对替换元素进行验证，默认为strict。可接受如下属性值： | | |
| strict | XML处理器必须获得所需命名空间（由namespace指定）对应的Schema，并验证来自该命名空间的所有元素 | |
| lax | XML处理器试图获得所需命名空间对应的Schema，如果获得了该Schema，则验证来自该命名空间的所有元素；即使没有获得该Schema，也不会发生任何错误。 | |
| skip | XML处理器不会去获取所需命名空间，也不会进行任何验证。当然，替换元素依然必须来自namespace属性所指定的命名空间。 | |

2、属性通配符

属性通配符与元素通配符的用法基本相似，只是定义属性通配符使用<anyAttribute.../>元素，而且由于同一个元素下不能包含同名属性，因此<anyAttribute.../>元素不能指定minOccurs和maxOccurs两个属性。<anyAttribute.../>元素中的id、namespace和processContents属性与<any.../>元素中的相应属性意义完全相同。

六、元素替换

1、概念及定义

Schema还提供了一种机制，允许使用一个元素替换另一个元素。如果想定义某个元素可替换另一个元素，可为该元素增加substitutionGroup属性，其值就是该元素想替换的元素的名字。

使用元素替换要注意如下几点：

①、替换元素和被替换元素都必须以全局元素的形式来声明。

②、替换元素与被替换元素要么有相同的数据类型，要么替换元素类型是被替换元素类型的派生类型。

2、阻止自己被替换

如果想在定义某个全局元素时阻止自己被替换，可以为<element.../>元素增加final属性，其值可以是：

|  |  |
| --- | --- |
| *#all* | 该元素不会被其他元素替换 |
| *extension* | 阻止以扩展该元素类型的方式派生出来的新类型的元素替换自己 |
| *restriction* | 阻止以限制该元素类型的方式派生出来的新类型的元素替换自己 |
| *extension*和*restriction*的组合 | 阻止以限制和扩展该元素类型的方式派生出来的新类型的元素替换自己 |

Schema允许将<element.../>元素里的final属性替换成block，block属性能接受的属性值与final属性基本一样，区别是通过block属性阻止自己被替换只对XML文档有效，对Schema则不会有任何影响。

3、阻止指定派生类型的替换

使用block属性指定，该属性能接受的属性值与<ele-ment.../>元素里final属性能接受的属性值大致相同。

七、抽象元素与抽象类型

1、概念及定义

抽象元素和抽象类型都不可以直接在XML文档中使用，在XML文档中必须使用替换元素来替换抽象元素，使用派生类型来替换抽象类型。定义元素或者类型时指定abstract="true"，时该元素或类型成为抽象元素或抽象类型。

2、抽象元素

如果将某个元素定义为抽象元素，通常应该为该元素定义替换元素，否则在XML文档中将无法使用该抽象元素。

示例：

3、抽象类型

希望在XML文档中使用抽象类型的元素，必须为该元素增加xsi:type属性，以指定它是抽象类型的某个派生类型的元素。

示例：

八、一致性约束

1、概念及定义

一致性约束只能在<element.../>元素内定义，且只能在<element.../>元素的最后面定义。

|  |  |
| --- | --- |
| 3个约束元素 | |
| key | 相当于数据库里的主键约束，要求指定内容必须存在而且唯一 |
| keyref | 相当于数据库里的外键约束，要求指定内容的值必须引用另一个key约束或unique约束。keyref约束用于建立两个数据项之间的引用关系。 |
| unique | 相当于数据库里的唯一约束，要求指定内容必须唯一，但可以不存在。 |
| 2个子元素指定该约束将对哪些部分起作用 | |
| selector | 该元素需要指定一个xpath属性，其值是一个XPath表达式，用来确定一个元素范围。在一次约束定义中，<selector.../>元素必须出现一次，也只能出现一次。 |
| field | 该元素需要指定一个xpath属性，其值也是一个XPath表达式。在一次约束定义中，<field.../>元素至少要出现一次，也可以出现多次。 |

2、key约束

示例：

3、unique约束

示例：

keyref约束

示例：

九、元素组和属性组

|  |  |
| --- | --- |
| 定义元素组：<group.../> | |
| 有顺序的元素关系 | 使用<sequence.../>元素定义。 |
| 互斥的元素关系 | 使用<choice.../>元素定义。 |
| 无顺序的元素关系 | 使用<all.../>元素定义。 |
| <xs:group ref=*"*分组名*"* minOccurs=*"*最少出现次数*"* maxOccurs=*"*最多出现次数*"*/> | |
| 定义属性组：<attributeGroup.../> | |
| 语法格式：<xs:attributeGroup ref=*"*属性组名*"*/> | |

元素组示例：

属性组实例：

十、定义符号

Schema使用<notation.../>元素来定义符号，该元素可以接受如下属性：

|  |  |
| --- | --- |
| id | 指定该符号的唯一标识，通常无须指定 |
| name | 指定该符号的名称。这是一个必填属性，而且该名称在整个Schema内必须是唯一的。 |
| public | 指定该符号所标识数据的外部格式或对应处理程序。这也是个必填属性，相当于<!NOTATION...>中PUBLIC的作用。 |
| system | 指定该符号所标识数据的外部格式或对应处理程序。这是个可选属性，相当于<! NOTATION...>中SYSTEM的作用。 |

示例：