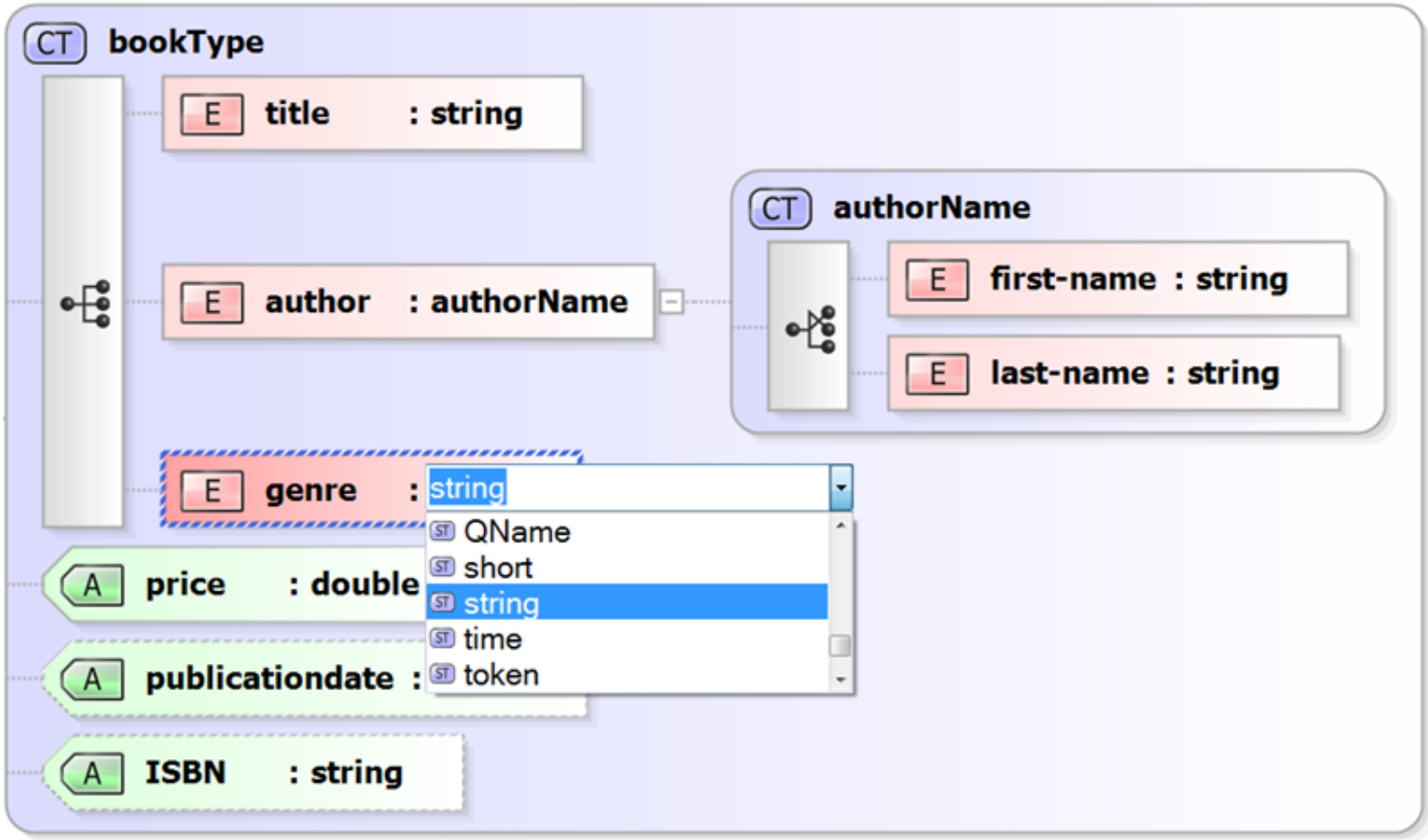
# XML 原理与应用

夏天

中国人民大学


CT bookType
E title : string
E author : authorName
CT authorName
E first-name : string
E last-name : string
E genre : string
QName
short
string
time
token
A price : double
A publicationdate :
A ISBN : string

# 本章学习目标

- 理解 XML Schema 的含义及用途
- 掌握 XML Schema 的元素、属性的作用及使用方式
- 掌握 XML Schema 的数据类型
- 理解 XML Schema 的命名空间的概念

# 目录

# 4.1 XML Schema 概述

- DTD 本身存在的不足
  - 具有自己独特的语法，需要特定的解析技术
  - 没有数据类型的概念
  - 不支持命名空间
  - 扩展机制复杂且较为脆弱
- XML Schema 是 DTD 的替代
  - DTD 较为简单，二者会长期并存

# 4.2 XML Schema 快速入门

一个 Schema 文档由元素、属性、命名空间和 XML 文档中的其他结点组成，并且至少要包含 Schema 根元素和 XML Schema 命名空间定义。

# 4.2.1 实例—XML 文档 I

```
1  <?xml version="1.0"?>
2  <book isbn="7535421016"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3       xsi:noNamespaceSchemaLocation="4-1.xsd">
4    <name>康熙大帝</name>
5    <author>二月河</author>
6    <price>86</price>
7    <pages>2012</pages>
8    <introduction>清顺治十八年，恶疾天花袭击了皇宫，皇帝爱妃
9  命丧黄泉，顺治痛不欲生，立意遁入空门。危急之际...</introduction>
10   <publish>
11     <publisher>长江文艺出版社</publisher>
12     <pubDate>2006-01-01</pubDate>
13   </publish>
14 </book>
```

# 4.2.1 实例—XML 文档 II

- 行 2 中的“xmlns:xsi”属性表示命名空间 xsi（XML Schema Instance）为一个 XML Schema 实例
- 通过 xsi:noNamespaceSchemaLocation 与指定的 XML Schema 文件 4-1.xsd 建立关联，实现有效性验证

# 实例—Schema 文档 I

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
3    <xsd:element name="book">
4      <xsd:complexType>
5        <xsd:sequence>
6          <xsd:element name="name" type="xsd:string"/>
7          <xsd:element name="author" type="xsd:string"/>
8          <xsd:element name="price" type="xsd:integer"/>
9          <xsd:element name="pages" type="xsd:integer"/>
10         <xsd:element name="introduction" type="xsd:string"/>
11         <xsd:element name="publish" minOccurs="0" maxOccurs="1">
12           <xsd:complexType>
13             <xsd:sequence>
14               <xsd:element name="publisher" type="xsd:string"/>
15               <xsd:element name="pubDate" type="xsd:date"/>
16             </xsd:sequence>
```

# 实例—Schema 文档 II

```
17              </xsd:complexType>
18            </xsd:element>
19          </xsd:sequence>
20          <xsd:attribute name="isbn" type="xsd:string"/>
21        </xsd:complexType>
22      </xsd:element>
23  </xsd:schema>
```

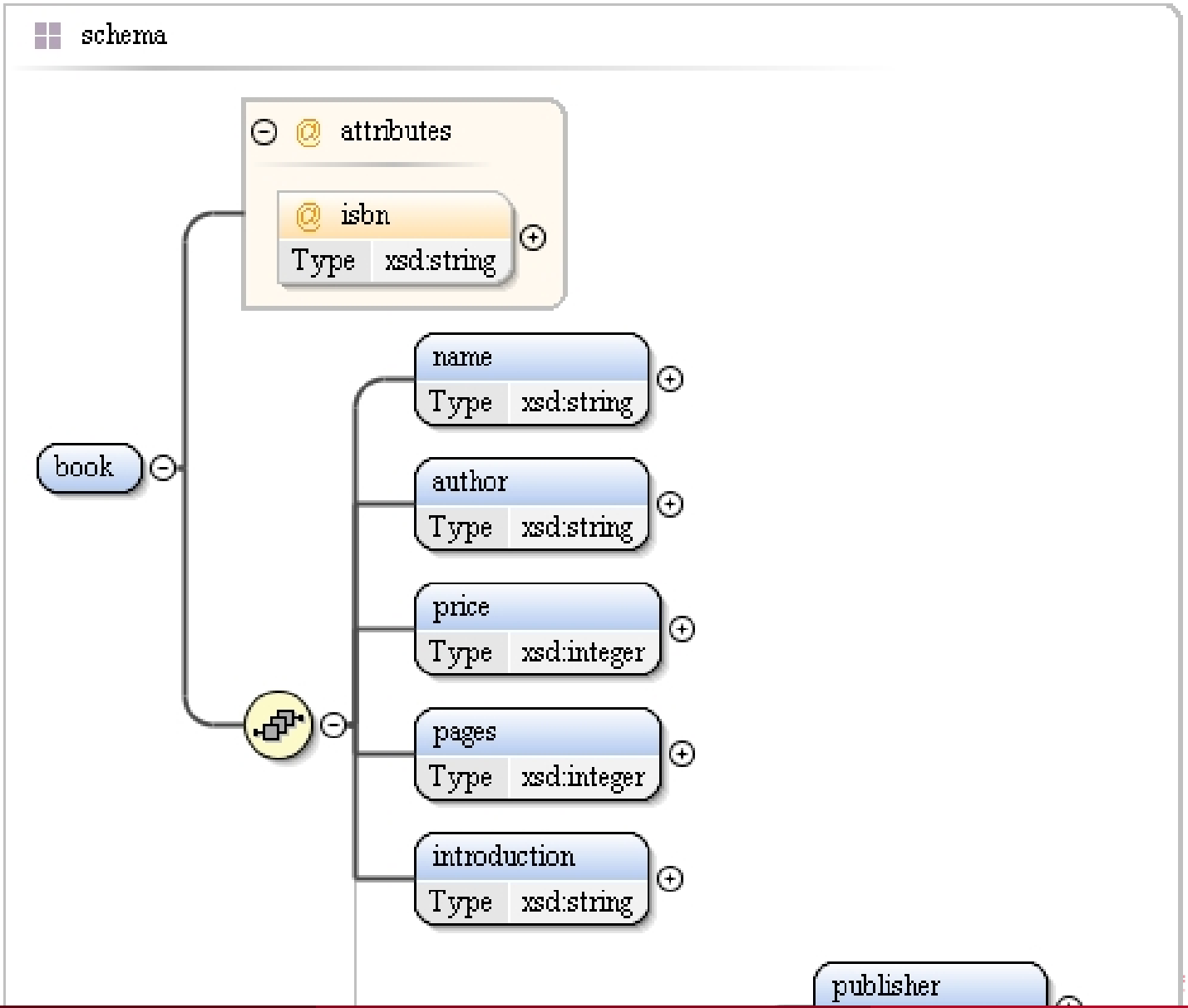
- 每一个 XML Schema 文档都以 schema 作为根元素，并位于命名空间“http://www.w3.org/2001/XMLSchema”之中
- 推荐使用的命名空间前缀有两种常见方式
  - xsd：XML Schema Document 的缩写表示
  - xs：XML Schema 的缩写表示

# 4.2.2 Schema 文档结构

- Schema 文档构成
  - 定义命名空间
  - 定义根元素的名字和类型
  - 定义子元素的名字和类型，并说明和根元素的关系
- 简单 Schema 文档

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
3   <!-- 详细声明内容 -->
4 </xsd:schema>
```

# Schema 文档可以树状方式查看



# 4.2.3 引用方式

- 最为常见的引用方式示例：

```
1 <book isbn="7535421016"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2   xsi:noNamespaceSchemaLocation="4-1.xsd">
3   <!-- XML文档内容..... -->
4 </book>
```

# 4.2.4 包含与导入

- 借助于包含（include）或导入（import）机制，一个主模式文档可以由多个模式文档组合而成
  - 当其他模式文档与主模式文档具有相同的目标命名空间时，可以使用包含方式
  - 当各自拥有不同的目标命名空间时，则使用导入方式
- 适用于复杂场景

# 4.3 XML Schema 的元素声明

XML Schema 通过 element 语法对元素进行声明，支持元素的类型名称、数据类型以及默认值、固定值等设置，元素可以是简单类型，也可以是复杂类型

# 4.3.1 schema 根元素

- Schema 文档都必须定义一个名称为 schema 的根元素
  - 该元素包含 8 个可选的属性，分别为 attributeFormDefault、blockDefault、elementFormDefault、finalDefault、id、targetNamespace、version、xml:lang
- 简单示例

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <xsd:schema id="mySchema"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">
3   <!-- XML Schema文档的详细定义 -->
4 </xsd:schema>
```

  - 可选的 id 属性为了方便用户的使用
  - 对应的名称空间为：http://www.w3.org/2001/XMLSchema

# 4.3.2 element 元素

- element 元素用于声明元素的属性，语法如下：

```
<xsd:element name="元素名称" type="元素类型"/>
```

  - name 是元素类型的名称
  - 必选的 type 属性用于说明元素的数据类型

- 例子

```
<xsd:element name="title" type="xsd:string"/>
```

# 4.3.3 elment 元素的默认值和固定值

- default 属性

```
<xsd:element name="gender" type="xsd:string" default="男"/>
```

  - 当对应元素为空时，填入默认值
  - “空”的定义与数据类型相关
    - 例如 string，本身就允许空值，因此其默认值不会被填充
    - integer 数据类型的空元素会被认为是空的，并将填入默认值
    - 如果元素的 xsi:nil 属性被设置为 true，也不会插入默认值
- fixed 属性
  - 元素值不能被改写，必须与 fixed 中所指定的值保持等价
  - 等价并不意味形式完全一样

# 元素默认值行为举例

- Schema 文档

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
3   <xsd:element name="name" type="xsd:string"/>
4   <xsd:element name="author" type="xsd:string" default="佚名"/>
5   <xsd:element name="price" type="xsd:integer" default="20"/>
6 </xsd:schema>
```

# 元素默认值行为举例

- 指定值: 前后不变
  - 扩充前: <author> 霍金 </author>
  - 扩充后: <author> 霍金 </author>
- 空元素（string）: 字符串的默认值不填充
  - 扩充前: <author></author>
  - 扩充后: <author></author>
- 空元素（integer）: 整数类型的默认值自动填充
  - 扩充前: <price></price>
  - 扩充后: <price>20</price>
- 元素为空: 保持不变
  - 扩充前: <price xsi:nil=”true”/>
  - 扩充后: <price xsi:nil=”true”/>

# 元素固定值行为举例

- Schema 文档

```
<xsd:element name="count" type="xsd:integer" fixed="10"/>
```

- 有效实例

```
1 <count>10</count>
2 <count>010</count>
3 <count>+10</count>
4 <count></count>
5 <count/>
```

# 4.3.4 元素的引用和替代

- 目的
  - 通过全局元素声明和元素引用，利用 ref 属性与已定义的元素进行关联
  - 避免重复定义
  - 如下例中的第 20 行和 27 行

# 示例文档 I

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="book">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="name" type="xsd:string"/>
        <xsd:element name="author" type="xsd:string"/>
        <xsd:element name="price" type="xsd:decimal"/>
        <xsd:element name="introduction" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="books">
    <xsd:complexType>
      <xsd:sequence>
```

# 示例文档 II

```
17          <xsd:element name="computer-books">
18            <xsd:complexType>
19              <xsd:sequence>
20                <xsd:element ref="book" maxOccurs="10"/>
21              </xsd:sequence>
22            </xsd:complexType>
23          </xsd:element>
24          <xsd:element name="math-books">
25            <xsd:complexType>
26              <xsd:sequence>
27                <xsd:element ref="book" maxOccurs="10"/>
28              </xsd:sequence>
29            </xsd:complexType>
30          </xsd:element>
31        </xsd:sequence>
32      </xsd:complexType>
```

# 示例文档 III

```
33     </xsd:element>
34 </xsd:schema>
```

# 4.4 XML Schema 的属性声明

- 通过 attribute 元素进行属性声明
- 属性声明与元素声明的区别
  - 属性的类型只能是简单类型
  - 属性不能包含子属性，而元素可以包含子元素
  - 属性之间没有顺序要求

# 4.4.1 属性声明

- 语法形式：

```
<xsd:attribute name="属性名称" type="属性类型"/>
```

- Example:

```
<xsd:attribute name="类别" type="xsd:string"/>
```

- 支持全局属性声明和属性引用，以提高复用程度

# use 属性

- use 属性用于指示所声明的属性是否在 XML 文档中必须出现
- 取值：
  - optional:
    - 可选属性，所声明的属性可以出现，也可以不出现，为 use 属性的默认值。
  - required
    - 该属性必须指定，不能缺省。
  - prohibited
    - 禁止在元素上使用该属性，等同于把该属性删除掉。

# 例子

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
3    <xsd:attribute name="code" type="xsd:string"/>
4
5    <xsd:element name="course">
6      <xsd:complexType>
7        <xsd:sequence>
8          <xsd:element name="title" type="xsd:string"/>
9          <xsd:element name="teacher" type="xsd:string"/>
10       </xsd:sequence>
11       <xsd:attribute ref="code" use="required"/>
12       <xsd:attribute name="location" type="xsd:string"
           use="prohibited"/>
13     </xsd:complexType>
14   </xsd:element>
15 </xsd:schema>
```

# 4.4.2 指派属性类型

- 属性不能包含子元素和属性，一定是简单类型，指定方式：
  - 在属性声明中通过 type 属性指定为简单类型
  - 通过 simpleType 以匿名类型的方式为属性指定类型
  - 不明确指定属性的类型，
    - 此时属性的类型为 anySimpleType，可以是任意合法的属性取值

# 例子 I

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
3    <xsd:attribute name="code" type="xsd:string"/>
4    <xsd:attribute name="classroom"/>
5    <xsd:attribute name="category">
6      <xsd:simpleType>
7        <xsd:restriction base="xsd:string">
8          <xsd:enumeration value="专业选修"/>
9          <xsd:enumeration value="专业必修"/>
10       </xsd:restriction>
11     </xsd:simpleType>
12   </xsd:attribute>
13 </xsd:schema>
```

# 例子 II

- 行 4 11 以 simpleType 匿名方式声明 category 属性
- 行 12 的 classroom 的属性类型取默认值 anySimpleType

# 4.4.3 属性的默认值和固定值

- 默认值和固定值分别通过 default 和 fixed 属性进行设置
  - 二者不能同时出现，定义和扩充的方式与 element 元素中的方式一致
  - 指定了默认值的属性在 XML 实例中没有出现，则该属性及其默认值会被填入
  - 指定了固定值的属性，如果在 XML 实例中出现，所指定的属性值应和 Schema 中定义的固定值相等，如未指定，则该属性及其固定值会被自动填入

# 例子 I

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
3    <xsd:element name="course">
4      <xsd:complexType>
5        <xsd:sequence>
6          <xsd:element name="title" type="xsd:string"/>
7        </xsd:sequence>
8        <xsd:attribute name="periods" type="xsd:integer" default="3"/>
9        <xsd:attribute name="location" type="xsd:string"
       fixed="ROOM-403"/>
10     </xsd:complexType>
11   </xsd:element>
12 </xsd:schema>
```

# 例子 II

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <course xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xsi:noNamespaceSchemaLocation="4-7.xsd" periods="4">
4   <title>XML原理与应用</title>
5 </course>
```

# 4.5 XML Schema 的数据类型

- 简单类型
  - 内置的简单数据类型
  - 用户通过 simpleType 自定义的简单数据类型
- 复杂类型
  - 通过 complexType 进行定义

# 4.5.1 简单数据类型：SimpleType

- 原子类型
  - 内置数据类型
    - 原始数据类型（Primitive）
    - 派生数据类型（Derived）
  - 自定义简单类型
- 列表类型
- 联合类型

# 4.5.1.1 内置简单类型

- 内置数据类型可以用来描述元素的内容和属性取值，或者组合生成其他自定义数据类型。
- 包括原始数据类型（Primitive）和派生数据类型（Derived）两类

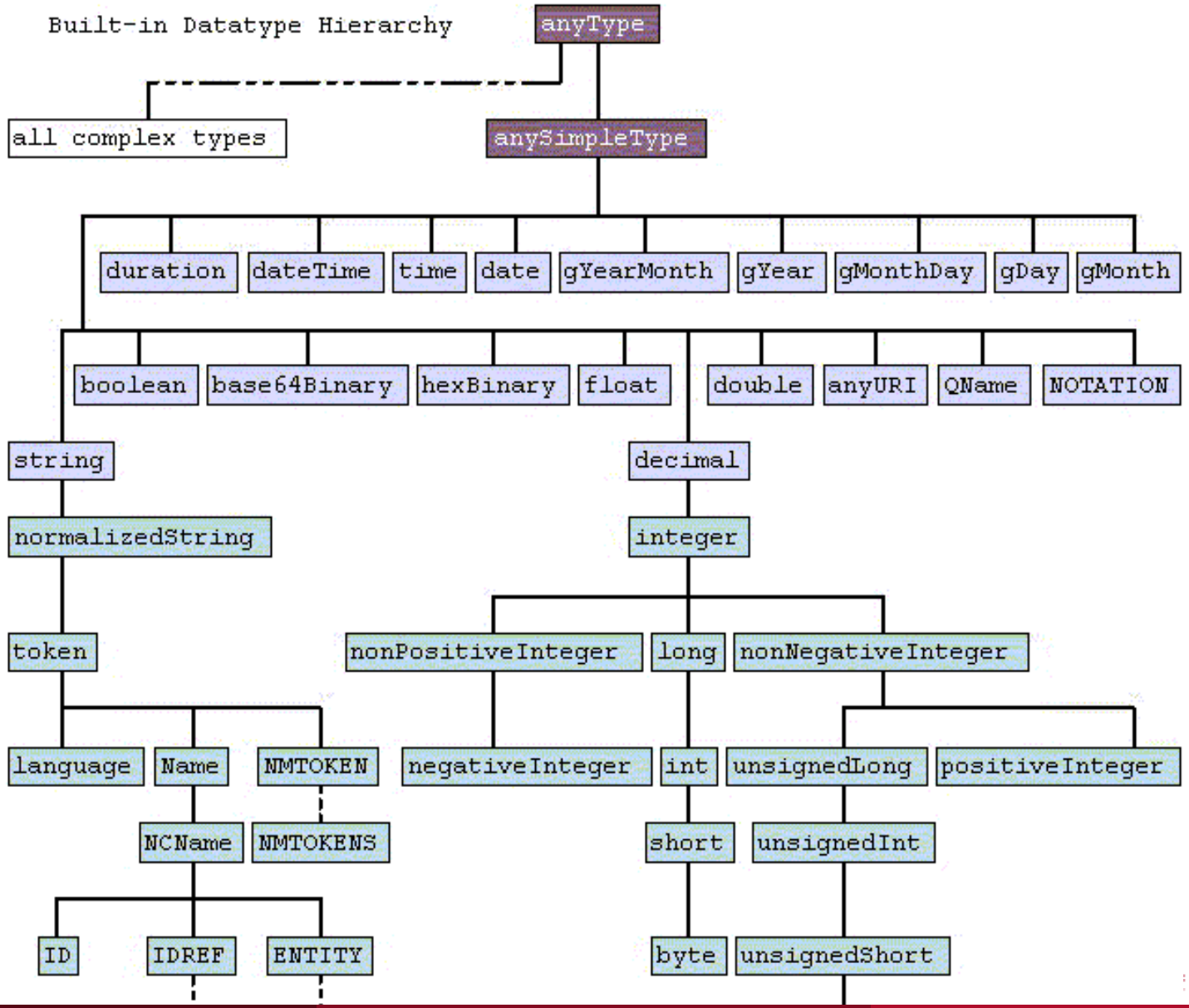
# 常用的原始数据类型

| 类型 | 描述 |
| --- | --- |
| string | 字符串 |
| boolean | 代表真假的布尔值 |
| decimal | 十进制数字 |
| float | 32 位单精度浮点数 |
| double | 64 位双精度浮点数 |
| date | 阳历日期 |
| time | 每天中任何一个时刻，如 18:36:16 |
| dateTime | 阳历日期和某一天时间组合而成的某个时刻 |

# 常用的派生数据类型

| 类型 | 描述 |
| --- | --- |
| integer | 任意长度的整数类型 |
| long | 64 位有符号整数 |
| int | 32 位有符号整数 |
| byte | 8 位有符号整数 |
| unsignedInt | 32 位无符号整数 |
| negativeInteger | 任意长度的负整数类型 |
| nonNegativeInteger | 大于等于零的整数 |
| normalizedString | 将回车、换行、制表符已转为空格 |

# XML Schema 内置数据类型的层次结构



# 4.5.1.2 自定义简单类型

- 根据已经存在的简单数据类型通过 simpleType 关键字进行定义
- 总是通过对一个已有简单类型进行约束（restriction）派生出来
- 用户自定义的简单数据类型语法：

```
<xsd:simpleType name="自定义数据类型的名称">
  <xsd:restriction base="所基于的简单数据类型的名称">
    <!-- 约束面 -->
  </xsd:restriction>
</xsd:simpleType>
```

  - 自定义简单类型通过 restriction 元素对现有类型进行约束
  - 约束面

# simpleType 的约束面

| 属性 | 描述 |
| --- | --- |
| pattern | 采用正则表达式方式限定数据的显示格式 |
| enumeration | 限定用户的取值为指定的数据集合 |
| length | 指定数据的长度 |
| maxExclusive | 指定数据的最大值（取值小于该指定值） |
| maxInclusive | 指定数据的最大值（取值小于或等于该指定值） |
| maxLength | 指定长度的最大值 |
| minExclusive | 指定数据的最小值（取值大于该指定值） |
| minInclusive | 指定数据的最小值（取值大于或等于该指定值） |
| minLength | 指定最小长度 |
| whiteSpace | 决定应用程序如何处理元素内容中的空白符 |
| totalDigits | 限定数字的最多位数 |
| fractionDigits | 限定最大的小数位，用于控制精度 |

# 示例

<mobile>13800138000</ mobile > 是描述一个手机号码的 XML 片段，此标记中的内容必须为数字，长度为固定的 11 位

```
1 <xsd:simpleType name="mobileType">
2   <xsd:restriction base="xsd:string">
3     <xsd:length value="11"/>
4     <xsd:pattern value="\d{11}"/>
5   </xsd:restriction>
6 </xsd:simpleType>
```

# 练习

- 定义一个整数，其取值范围为 [10, 100]
- 定义一个字符串，最少由 4 个字符组成，最多由 16 个字符组成
- 定义一个性别类型，其取值只能为“男”或“女”

# 4.5.1.3 列表类型与联合类型

- 列表类型所定义的元素或属性的值可以包含多个原子值，这些并列的原子值之间通过空格分隔。列表元素使用 list 元素进行定义
- 语法：

```
<xsd:simpleType name="列表类型名">
  <xsd:list itemType="某一原子类型"/>
</xsd:simpleType>
```

# 列表类型示例

```
1 <xsd:element name="years" type="yearListType"/>
2 <xsd:simpleType name="yearType">
3   <xsd:restriction base="xsd:string">
4     <xsd:pattern value="\d{4}"/>
5   </xsd:restriction>
6 </xsd:simpleType>
7 <xsd:simpleType name="yearListType">
8   <xsd:list itemType="yearType"/>
9 </xsd:simpleType>
```

```
<years>2010 2011 2012</years>
```

# 联合类型

- 语法：

```
<xsd:simpleType name="联合类型名">
<xsd:union memberTypes="简单类型1 简单类型2 ....."/>
</xsd:simpleType>
```

  - 联合类型可以包含多个简单类型，所包含的简单类型既可以是原子类型，也可以是列表类型
  - 取值可以是 memberTypes 属性值中所指明的某一个简单类型的实例

# 联合类型示例

```
<xsd:element name="whichYear" type="yearUnionType"/>
<xsd:simpleType name="yearUnionType">
  <xsd:union memberTypes="xsd:date yearListType"/>
</xsd:simpleType>
```

正确用例：

```
<whichYear>2008-02-09</whichYear>
<whichYear>2010</whichYear>
```

错误用例：

```
<whichYear>2008-02-09 2010</whichYear>
<whichYear>2010年</whichYear>
```

# 4.5.2 复杂数据类型：ComplexType

- 四种复杂数据类型
  - 只包含文本的简单内容类型
  - 只包含子元素的纯元素内容类型
  - 包含子元素和文本的混合内容类型
  - 空元素

# 复杂数据类型的基本语法形式

```
<xsd:complexType name="名称" id=ID mixed=BOOLEAN: false>
  <xsd:annotation | simpleContent | complexContent | group | all |
    choice | sequence | attribute | attributeGroup | anyAttribute>...
</xsd:complexType>
```

- name 复杂类型的名称
- id 该元素的 ID，可选项
- mixed，可选布尔类型，该复杂类型的子元素之中是否允许出现字符数据，默认值为 false

# 复杂类型的内容模型

- 复杂类型的内容模型指复杂类型子元素的顺序和结构称
  - 内容模型不依赖于属性，但允许有属性
  - complexType 下可以创建的内容模型有
    - simpleContent、complexContent、sequence、group、all、choice、annotation

# 4.5.2.1 simpleContent

- 用于从简单类型派生复杂类型
- 适用于元素包含字符内容和属性但不包含子元素的情况
- 简单内容模型的复杂类型可以包含属性，而简单类型不可
- 语法：

```
<xsd:simpleContent id=ID >
  <xsd:annotation | restriction | extension>...
</xsd:simpleContent>
```

# simpleContent 示例 I

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
3    <xsd:element name="book">
4      <xsd:complexType>
5        <xsd:simpleContent>
6          <xsd:extension base="xsd:string">
7            <xsd:attribute name="isbn" type="xsd:string"
       use="required"/>
8          </xsd:extension>
9        </xsd:simpleContent>
10     </xsd:complexType>
11   </xsd:element>
12 </xsd:schema>
```

# simpleContent 示例 II

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <book xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3     xsi:noNamespaceSchemaLocation="4-8.1.xsd"
4     isbn="6-666-66666-6">
5     XML原理与应用
6 </book>
```

# 4.5.2.2 complexContent

- 用于对复杂数据类型进行扩展或限制，从复杂类型派生新的复杂类型
- 语法：

```
<xsd:complexContent id=ID mixed=true|false>
  <xsd:annotation | restriction | extension>...
</xsd:simpleContent>
```

# complexContent 示例 I

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
3    <xsd:element name="student" type="studentType"/>
4
5    <xsd:complexType name="personType">
6      <xsd:sequence>
7        <xsd:element name="name" type="xsd:string"/>
8        <xsd:element name="age" type="xsd:integer"/>
9      </xsd:sequence>
10   </xsd:complexType>
11
12   <xsd:complexType name="studentType">
13     <xsd:complexContent>
14       <xsd:extension base="personType">
15         <xsd:sequence>
16           <xsd:element name="class" type="xsd:string"/>
```

# complexContent 示例 II

```
17              <xsd:element name="major" type="xsd:string"/>
18            </xsd:sequence>
19          </xsd:extension>
20        </xsd:complexContent>
21      </xsd:complexType>
22  </xsd:schema>
```

# 4.5.2.3 顺序声明：sequence

- 最为常用的内容模型定义方式
- 用于限定 sequence 内部的一组元素的出现顺序
- 带有 minOccurs 和 maxOccurs 两个属性，作用于整个顺序组合

# sequence 示例 I

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
3    <xsd:element name="person" type="personType"/>
4    <xsd:complexType name="personType">
5      <xsd:sequence minOccurs="1" maxOccurs="10">
6        <xsd:element name="name" type="xsd:string"/>
7        <xsd:element name="age" type="xsd:integer"/>
8      </xsd:sequence>
9    </xsd:complexType>
10 </xsd:schema>
```

# sequence 示例 II

以下代码片段能够通过有效性验证？

```
<name>刘备</name>
<age>35</age>
<name>张飞</name>
<age>35</age>
```

以下代码片段能够通过有效性验证？

```
<age>35</age>
<name>刘备</name>
```

# 4.5.2.4 选择声明：choice

- 多选一

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
3    <xsd:complexType name="personType">
4      <xsd:sequence minOccurs="1" maxOccurs="10">
5        <xsd:element name="name" type="xsd:string"/>
6        <xsd:element name="age" type="xsd:integer"/>
7        <xsd:choice>
8          <xsd:element name="wife" type="xsd:string"/>
9          <xsd:element name="husband" type="xsd:string"/>
10       </xsd:choice>
11     </xsd:sequence>
12   </xsd:complexType>
13 </xsd:schema>
```

# 4.5.2.5 分组声明：group

- 使用方式
  - group 声明
    - 用于指定一个模型组的定义，定义该分组内包含的内容，并以 schema 元素的子元素形式出现；
  - group 引用
    - 对已经定义分组进行引用，此时 group 出现在复杂类型定义或其他模型组定义的内部。

# group 示例 I

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
3    <xsd:group name="HeightAndWeightGroup">
4      <xsd:sequence>
5        <xsd:element name="height" type="xsd:integer"/>
6        <xsd:element name="weight" type="xsd:integer"/>
7      </xsd:sequence>
8    </xsd:group>
9
10   <xsd:complexType name="personType">
11     <xsd:sequence minOccurs="1" maxOccurs="10">
12       <xsd:element name="name" type="xsd:string"/>
13       <xsd:element name="age" type="xsd:integer"/>
14       <xsd:choice>
15         <xsd:element name="wife" type="xsd:string"/>
16         <xsd:element name="husband" type="xsd:string"/>
```

# group 示例 II

```
17          </xsd:choice>
18          <xsd:group ref="HeightAndWeightGroup"/>
19        </xsd:sequence>
20      </xsd:complexType>
21  </xsd:schema>
```

# 4.5.2.6 ALL 声明：all

- 元素可以在实例文档中以任意顺序出现
  - all 元素的任何子元素声明，其 minOccurs 属性只能取 0 或 1 的值，maxOccurs 属性只能取 1 值
- all 示例：

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
3   <xsd:complexType name="personType">
4     <xsd:all>
5       <xsd:element name="name" type="xsd:string"/>
6       <xsd:element name="age" type="xsd:integer"/>
7     </xsd:all>
8   </xsd:complexType>
9 </xsd:schema>
```

# all 示例 I

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
3    <xsd:group name="HeightAndWeightGroup">
4      <xsd:sequence>
5        <xsd:element name="height" type="xsd:integer"/>
6        <xsd:element name="weight" type="xsd:integer"/>
7      </xsd:sequence>
8    </xsd:group>
9
10   <xsd:complexType name="personType">
11     <xsd:sequence minOccurs="1" maxOccurs="10">
12       <xsd:element name="name" type="xsd:string"/>
13       <xsd:element name="age" type="xsd:integer"/>
14       <xsd:choice>
15         <xsd:element name="wife" type="xsd:string"/>
16         <xsd:element name="husband" type="xsd:string"/>
```

# all 示例 II

```
17          </xsd:choice>
18          <xsd:group ref="HeightAndWeightGroup"/>
19        </xsd:sequence>
20      </xsd:complexType>
21  </xsd:schema>
```

该定义方式与上一定义方式的约束效果相同。

# 4.6 XML Schema 与命名空间

- targetNamespace
- elementFormDefault
- attributeFormDefault

# 4.6.1 targetNamespace

- 每一个 Schema 文档都可有一个命名空间，称为模式文档的目标命名空间（Target Namespace）
- 每个被全局声明所定义的元素、属性、类型或分组，都与该目标命名空间有关。
- 通过 targetNamespace 指定所定义元素或属性所隶属的命名空间，
  - 默认情况下仅对全局声明的元素和属性起作用（未设置 elementFormDefault 和 attributeFormDefault）

# targetNamespace 示例:4-14.xsd

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
3          targetNamespace="http://www.example.org/test">
4    <xsd:element name="course">
5      <xsd:complexType>
6        <xsd:sequence>
7          <xsd:element name="name" type="xsd:string"/>
8          <xsd:element name="teacher" type="xsd:string"/>
9        </xsd:sequence>
10       <xsd:attribute name="code" type="xsd:string" use="required"/>
11     </xsd:complexType>
12   </xsd:element>
13 </xsd:schema>
```

# targetNamespace 示例:4-14.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <test:course xmlns:test=http://www.example.org/test
3     xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
4     xsi:schemaLocation="http://www.example.org/test 4-14.xsd"
        code="106">
5   <name>数据挖掘</name>
6   <teacher>夏天</teacher>
7 </test:course>
```

# 4.6.2 elementFormDefault 与 attributeFormDefault

- elementFormDefault
  - unqualified(默认值)
    - 内部元素不受目标命名空间的约束
  - qualified
    - 内部元素也将受到目标命名空间的约束
- attributeFormDefault
  - unqualified(默认值)
    - 内部自定义的属性不受目标命名空间的约束
  - qualified
    - 受目标命名空间约束

# 4.6.3 form 属性

- 如需要对特定元素或属性设置不同于全局的配置，可以借助于 element 或 attribute 元素的 form 属性
- form 属性的取值和作用与全局的 elementFormDefault 或 attributeFormDefault 相似，但只作用于当前声明的对象。

# 4.7 XML Schema 的注释与注解

- 注释：
  - 可以在模式文档中任意插入 XML 注释，只要符合 XML 的语法规定即可
- annotation
  - 增强模式文档的机器可读性
  - 不仅提供文档信息，还提供程序信息，允许模式文档使用读者可以理解和机器可以识别的信息来注释
  - 语法：

```
<xsd:annotation id=ID >
  <xsd:appinfo | documentation>...
</xsd:annotation>
```

# 注解应用示例 I

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
3    <xsd:element name="course" type="courseType"/>
4
5    <xsd:annotation>
6      <xsd:appinfo source="4-18.html"/>
7      <xsd:documentation
       xmlns:html="http://www.w3.org/1999/xhtml">
8
       <html:p>注解使用示例，下面所声明的<html:strong>courseType</htm
9      </xsd:documentation>
10   </xsd:annotation>
11   <xsd:complexType name="courseType">
12     <xsd:sequence>
13       <xsd:element name="name" type="xsd:string"/>
```

# 注解应用示例 II

```
14        <xsd:element name="teacher" type="xsd:string"
          form="unqualified"/>
15       </xsd:sequence>
16       <xsd:attribute name="code" type="xsd:string" form="qualified"
          use="required">
17         <xsd:annotation>
18
          <xsd:documentation>必须有课程代号属性code</xsd:documentation>
19         </xsd:annotation>
20       </xsd:attribute>
21     </xsd:complexType>
22  </xsd:schema>
```

# END