XML Schema

Author williamfan 2017-07-03

website pengfeifan.github.io

XML Schema 简介

`XML Schema` 是基于 XML 的 DTD 替代者。
`XML Schema` 可描述 XML 文档的结构。
`XML Schema` 语言也可作为 `XSD（XML Schema Definition）` 来引用。

应当具备的基础知识：

- HTML / XHTML
- XML 以及 XML 命名空间
- 对 DTD 的基本了解

什么是 XML Schema？

`XML Schema` 的作用是定义 XML 文档的合法构建模块，类似 `DTD`。
XML Schema:

- 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

XML Schema 是 DTD 的继任者

`XML Schema` 很快会在大部分网络应用程序中取代 `DTD`。
理由如下：

- 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

XML Schema 是 W3C 标准

XML Schema 在 *2001* 年 *5* 月 *2* 日成为 *W3C* 标准。

为什么使用 XML Schema?

XML Schema 支持数据类型

XML Schema 最重要的能力之一就是对数据类型的支持。
通过对数据类型的支持：

- 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
  注：`数据约束`，或称`facets`，是 XML Schema 原型中的一个术语，中文可译为`"面"`，用来约束数据类型的容许值。

XML Schema 使用 XML 语法

另一个关于 `XML Schema` 的重要特性是，它们由 `XML` 编写。
由 XML 编写 XML Schema 有很多好处：

- 不必学习新的语言
- 可使用 XML 编辑器来编辑 Schema 文件
- 可使用 XML 解析器来解析 Schema 文件
- 可通过 XML DOM 来处理 Schema
- 可通过 XSLT 来转换 Schema

XML Schema 可保护数据通信

当数据从发送方被发送到接受方时，其要点是双方应有关于内容的相同的”期望值”。
通过 XML Schema，发送方可以用一种接受方能够明白的方式来描述数据。
一种数据，比如`"03-11-2004"`，在某些国家被解释为11月3日，而在另一些国家为当作3月11日。
但是一个带有数据类型的 XML 元素，比如：`<date type="date">2004-03-11</date>`，可确保对**内容一致**的理解，这是因为 XML 的数据类型 `"date"` 要求的格式是`"YYYY-MM-DD"`。

XML Schema 可扩展

XML Schema 是可扩展的，因为它们由 XML 编写。
通过可扩展的 Schema 定义可以：

- 在其他 Schema 中重复使用的Schema

- 创建由标准类型衍生而来的数据类型
- 在相同的文档中引用多重的 Schema

XSD 如何使用?

`XML 文档`可对 `DTD`或 `XML Schema` 进行引用。
一个简单的 XML 文档：
`note.xml` 的 XML 文档：

```
<?xml version="1.0"?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

DTD 文件

`"note.dtd"`的 DTD 文件，它对上面那个 XML 文档`note.xml`的元素进行了定义：

```
<!ELEMENT note (to, from, heading, body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

第 1 行定义 note 元素有四个子元素：”to, from, heading, body”。
第 2-5 行定义了 to, from, heading, body 元素的类型是 “#PCDATA”。

XML Schema

`"note.xsd"`的`XML Schema` 文件，它定义了上面那个 XML 文档`note.xml`的元素：

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.w3schools.com"
xmlns="http://www.w3schools.com"
elementFormDefault="qualified">

<xs:element name="note">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="to" type="xs:string"/>
      <xs:element name="from" type="xs:string"/>
      <xs:element name="heading" type="xs:string"/>
      <xs:element name="body" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

</xs:schema>
```

note 元素是一个`复合类型`，因为它包含其他的子元素。其他元素 (to, from, heading, body) 是`简易类型`，因为它们没有包含其他元素。

对 DTD 的引用

此文件包含对 DTD 的引用：

```
<?xml version="1.0"?>

<!DOCTYPE note SYSTEM
"http://www.w3schools.com/dtd/note.dtd">

<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

对 XML Schema 的引用

此文件包含对 XML Schema 的引用：

```
<?xml version="1.0"?>

<note
xmlns="http://www.w3schools.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.w3schools.com note.xsd">
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

XSD - <schema> 元素

<schema> 元素是每一个 XML Schema 的根元素。

<schema> 元素

<schema>元素是每一个 **XML Schema 的根元素**：

```
<?xml version="1.0"?>

<xs:schema>
...
...
</xs:schema>
```

<schema>元素可包含 属性。一个 schema 声明往往看上去类似这样：

```
<?xml version="1.0"?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.w3c.com"
xmlns="http://www.w3c.com"
elementFormDefault="qualified">
...
...
</xs:schema>
```

以下代码片段:

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`：

这个片断：

```
targetNamespace="http://www.w3c.com"
```

显示被此 schema 定义的元素 (note, to, from, heading, body) 来自命名空间：`"http://www.w3c.com"`。

这个片断：

```
xmlns="http://www.w3c.com"
```

指出默认的命名空间是 `"http://www.w3c.com"`。

这个片断：

```
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`。

在 XML 文档中引用 Schema

此 XML 文档含有对 XML Schema 的引用：

```
<?xml version="1.0"?>

<note xmlns="http://www.w3c.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.w3c.com note.xsd">

<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

下面的代码片断：

```
xmlns="http://www.w3c.com"
```

规定了 `默认命名空间的声明`。此声明会告知 `schema 验证器`，在此 XML 文档中使用的所有元素都被声明于 `"http://www.w3c.com"` 这个命名空间。
一旦有了可用的 `XML Schema实例命名空间` ：

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

就可以使用 `schemaLocation 属性` 了。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` ：

```
xsi:schemaLocation="http://www.w3c.com note.xsd"
```

XSD 简易元素

`XML Schema` 可定义 XML 文件的 `元素` 。
`简易元素` 指那些只包含文本的元素。它不会包含任何其他的元素或属性。

什么是简易元素？

简易元素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。
不过，”仅包含文本”这个限定却很容易造成误解。文本有很多类型。它可以是 XML Schema 定义中包括的类型中的一种（布尔、字符串、数据等等），或者它也可以是自行定义的定制类型。
也可向 `数据类型` 添加限定（即 `facets` ），以此来限制它的内容，或者可以要求 `数据匹配某种特定的模式` 。

定义简易元素

定义简易元素的语法：

```
<xs:element name="xxx" type="yyy"/>
```

此处 `xxx` 指元素的名称，`yyy` 指元素的数据类型。 `XML Schema` 拥有很多内建的 `数据类型` 。
最常用的类型是：

- xs:string
- xs:decimal
- xs:integer
- xs:boolean
- xs:date
- xs:time

实例

这是一些 XML 元素：

```
<lastname>Refsnes</lastname>
<age>36</age>
<dateborn>1970-03-27</dateborn>
```

这是相应的简易元素定义：

```
<xs:element name="lastname" type="xs:string"/>
<xs:element name="age" type="xs:integer"/>
<xs:element name="dateborn" type="xs:date"/>
```

简易元素的默认值和固定值

简易元素可拥有指定的`默认值`或`固定值`。

当没有其他的值被规定时，`默认值`就会自动分配给元素。
在下面的例子中，缺省值是`"red"`：

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

`固定值`同样会自动分配给元素，并且`无法规定另外一个值`。
在下面的例子中，固定值是`"red"`：

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

XSD 属性

所有的`属性`均作为`简易类型`来声明。

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。

如何声明属性？

定义属性的语法是

```
<xs:attribute name="xxx" type="yyy"/>
```

在此处，`xxx`指属性名称，`yyy`则规定属性的数据类型。`XML Schema`拥有很多内建的`数据类型`。
最常用的类型是：

- xs:string

- xs:decimal
- xs:integer
- xs:boolean
- xs:date
- xs:time

实例

这是带有属性的 XML 元素：

```
<lastname lang="EN">Smith</lastname>
```

这是对应的属性定义：

```
<xs:attribute name="lang" type="xs:string"/>
```

属性的默认值和固定值

`属性` 可拥有指定 `的默认值` 或 `固定值`。
当没有其他的值被规定时，`默认值` 就会自动分配给元素。
在下面的例子中，缺省值是 `"EN"` ：

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

`固定值` 同样会自动分配给元素，并且您无法规定另外的值。
在下面的例子中，固定值是 `"EN"` ：

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

可选的和必需的属性

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"/>
```

对内容的限定

当 XML `元素` 或 `属性` 拥有被定义的 `数据类型` 时，就会向元素或属性的内容添加 `限定`。
假如 XML 元素的类型是 `"xs:date"` ，而其包含的内容是类似 “Hello World” 的字符串，元素将不会（通过）验证。
通过 `XML schema` ，也可向XML 元素及属性添加自己的限定。这些限定被称为 `facet`。

XSD 限定 / Facets

`限定（restriction）` 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

下面的例子定义了带有一个限定且名为 `"age"` 的元素。age 的值不能低于 `0` 或者高于 `120` ：

```
<xs:element name="age">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="120"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

对一组值的限定

如需把 XML 元素的内容限制为一组可接受的值，要使用 `枚举约束（enumeration constraint）` 。
下面的例子定义了带有一个限定的名为 `"car"` 的元素。可接受的值只有： `Audi, Golf, BMW` ：

```
<xs:element name="car">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="Audi"/>
      <xs:enumeration value="Golf"/>
      <xs:enumeration value="BMW"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

上面的例子也可以被写为：

```
<xs:element name="car" type="carType"/>

<xs:simpleType name="carType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="Audi"/>
    <xs:enumeration value="Golf"/>
    <xs:enumeration value="BMW"/>
  </xs:restriction>
</xs:simpleType>
```

注意： 在这种情况下，类型 "carType" 可被其他元素使用，因为它不是 "car" 元素 的组成部分。

对一系列值的限定

如需把 XML 元素的内容限制定义为一系列可使用的 数字 或 字母 ，要使用 模式约束（pattern constraint）。
下面的例子定义了带有一个限定的名为 "letter" 的元素。可接受的值只有 小写字母 a - z 其中的一个：

```
<xs:element name="letter">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[a-z]"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

下一个例子定义了带有一个限定的名为 "initials" 的元素。可接受的值是 大写字母 A - Z 其中的三个：

```
<xs:element name="initials">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[A-Z][A-Z][A-Z]"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

下一个例子也定义了带有一个限定的名为 "initials" 的元素。可接受的值是 大写或小写字母 a - z 其中的三个：

```
<xs:element name="initials">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[a-zA-Z][a-zA-Z][a-zA-Z]"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

下一个例子定义了带有一个限定的名为 "choice 的元素。可接受的值是 字母 x, y 或 z 中的一个：

```
<xs:element name="choice">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[xyz]"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

下一个例子定义了带有一个限定的名为 `"prodid"` 的元素。可接受的值是 `五个阿拉伯数字的一个序列`，且每个数字的范围是 `0-9`：

```
<xs:element name="prodid">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:pattern value="[0-9][0-9][0-9][0-9][0-9]"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

对一系列值的其他限定

下面的例子定义了带有一个限定的名为 `"letter"` 的元素。可接受的值是 `a - z 中零个或多个字母`：

```
<xs:element name="letter">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="([a-z])*"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

下面的例子定义了带有一个限定的名为 `"letter"` 的元素。可接受的值是一对或多对字母，每对字母由一个小写字母后跟一个大写字母组成。举个例子，”sToP”将会通过这种模式的验证，但是 “Stop”、”STOP” 或者 “stop” 无法通过验证：

```
<xs:element name="letter">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="([a-z][A-Z])+"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

下面的例子定义了带有一个限定的名为 "gender" 的元素。可接受的值是 male 或者 female ：

```
<xs:element name="gender">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="male|female"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

下面的例子定义了带有一个限定的名为 "password" 的元素。可接受的值是由 8 个字符组成的一行字符，这些字符必须是大写或小写字母 a - z 亦或数字 0 - 9 ：

```
<xs:element name="password">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[a-zA-Z0-9]{8}"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

对空白字符的限定

如需规定对 空白字符（whitespace characters） 的处理方式，需要使用 whiteSpace 限定。
下面的例子定义了带有一个限定的名为 "address" 的元素。这个 whiteSpace 限定被设置为 "preserve" ，这意味着 XML 处理器不会移除任何空白字符：

```
<xs:element name="address">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:whiteSpace value="preserve"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

这个例子也定义了带有一个限定的名为 "address" 的元素。这个 whiteSpace 限定被设置为 "replace" ，这意味着 XML 处理器将移除所有空白字符（换行、回车、空格以及制表符）：

```
<xs:element name="address">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:whiteSpace value="replace"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

这个例子也定义了带有一个限定的名为 `"address"` 的元素。这个 `whiteSpace` 限定被设置为 `"collapse"`，这意味着 XML 处理器将移除所有空白字符（换行、回车、空格以及制表符会被替换为空格，开头和结尾的空格会被移除，而多个连续的空格会被缩减为一个单一的空格）：

```
<xs:element name="address">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:whiteSpace value="collapse"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

对长度的限定

如需限制元素中值的长度，需要使用 `length`、`maxLength` 以及 `minLength` 限定。本例定义了带有一个限定且名为 `"password"` 的元素。其值必须精确到 `8 个字符`：

```
<xs:element name="password">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:length value="8"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

这个例子也定义了带有一个限定的名为 `"password"` 的元素。其值最小为 5 个字符，最大为 8 个字符：

```
<xs:element name="password">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:minLength value="5"/>
      <xs:maxLength value="8"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

数据类型的限定

限定	描述
enumeration	定义可接受值的一个列表
fractionDigits	定义所允许的最大的小数位数。必须大于等于0。
length	定义所允许的字符或者列表项目的精确数目。必须大于或等于0。
maxExclusive	定义数值的上限。所允许的值必须小于此值。
maxInclusive	定义数值的上限。所允许的值必须小于或等于此值。
maxLength	定义所允许的字符或者列表项目的最大数目。必须大于或等于0。
minExclusive	定义数值的下限。所允许的值必需大于此值。
minInclusive	定义数值的下限。所允许的值必需大于或等于此值。
minLength	定义所允许的字符或者列表项目的最小数目。必须大于或等于0。
pattern	定义可接受的字符的精确序列。
totalDigits	定义所允许的阿拉伯数字的精确位数。必须大于0。
whiteSpace	定义空白字符（换行、回车、空格以及制表符）的处理方式。

XSD 复合元素

`复合元素` 包含了 `其他元素` 及 `/` 或 `属性`。

什么是复合元素？

复合元素指包含 `其他元素` 及 `/` 或 `属性` 的 `XML 元素`。
有四种类型的复合元素：

- 空元素

- 包含其他元素的元素
- 仅包含文本的元素
- 包含元素和文本的元素

注意： 上述元素均可包含属性！

复合元素的例子

复合元素，`"product"`，是空的：

`<product pid="1345"/>`

复合元素，`"employee"`，仅包含其他元素：

```
<employee>
  <firstname>John</firstname>
  <lastname>Smith</lastname>
</employee>
```

复合 XML 元素，`"food"`，仅包含文本：

`<food type="dessert">Ice cream</food>`

复合XML元素，`"description"`包含元素和文本：

```
<description>
It happened on <date lang="norwegian">03.03.99</date> ....
</description>
```

如何定义复合元素？

请看这个复合 XML 元素，`"employee"`，仅包含其他元素：

```
<employee>
  <firstname>John</firstname>
  <lastname>Smith</lastname>
</employee>
```

在`XML Schema`中，有两种方式来定义复合元素：

- 1.通过命名此元素，可直接对`"employee"`元素进行声明，就像这样：

```
<xs:element name="employee">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

假如使用上面所描述的方法，那么仅有 `"employee"` 可使用所规定的复合类型。请注意其子元素，`"firstname"` 以及 `"lastname"`，被包围在指示器 `<sequence>` 中。这意味着子元素必须以它们被声明的次序出现。

- 2. `"employee"` 元素可以使用 `type 属性`，这个属性的作用是引用要使用的复合类型的名称：

```
<xs:element name="employee" type="personinfo"/>

<xs:complexType name="personinfo">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

如果使用了上面所描述的方法，那么若干元素均可以使用相同的复合类型，比如这样：

```
<xs:element name="employee" type="personinfo"/>
<xs:element name="student" type="personinfo"/>
<xs:element name="member" type="personinfo"/>

<xs:complexType name="personinfo">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

也可以在已有的复合元素之上以某个复合元素为基础，然后添加一些元素，就像这样：

```
<xs:element name="employee" type="fullpersoninfo"/>

<xs:complexType name="personinfo">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="fullpersoninfo">
  <xs:complexContent>
    <xs:extension base="personinfo">
      <xs:sequence>
        <xs:element name="address" type="xs:string"/>
        <xs:element name="city" type="xs:string"/>
        <xs:element name="country" type="xs:string"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

XSD 空元素

`空的复合元素`不能包含内容，只能含有`属性`。

复合空元素：

一个空的 XML 元素：

```
<product prodid="1345" />
```

上面的`"product"`元素根本没有内容。为了定义`无内容的类型`，就必须声明一个在其内容中`只能包含元素`的类型，但是实际上并不会声明任何元素，比如这样：

```
<xs:element name="product">
  <xs:complexType>
    <xs:complexContent>
      <xs:restriction base="xs:integer">
        <xs:attribute name="prodid" type="xs:positiveInteger"/>
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
```

在上面的例子中，定义了一个带有复合内容的复合类型。`complexContent`元素给出的信号是，打算限定或者拓展某个复合类型的内容模型，而 `integer`限定则声明了一个属性但不会引入任何的元素内容。
但是，也可以更加紧凑地声明此`"product"`元素：

```
<xs:element name="product">
  <xs:complexType>
    <xs:attribute name="prodid" type="xs:positiveInteger"/>
  </xs:complexType>
</xs:element>
```

或者可以为一个 `complexType`元素起一个名字，然后为`"product"`元素设置一个 `type`属性并引用这个`complexType`名称（通过使用此方法，若干个元素均可引用相同的复合类型）：

```
<xs:element name="product" type="prodtype"/>

<xs:complexType name="prodtype">
  <xs:attribute name="prodid" type="xs:positiveInteger"/>
</xs:complexType>
```

XSD 仅含元素

`仅含元素`的复合类型元素是只能包含其他元素的元素。

复合类型仅包含元素

XML 元素，`"person"`，仅包含其他的元素：

```
<person>
  <firstname>John</firstname>
  <lastname>Smith</lastname>
</person>
```

可在`schema` 中这样定义 `"person"`元素：

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

请留意这个 。它意味着被定义的元素必须按上面的次序出现在`"person"`元素中。
或者可以为`complexType`元素设定一个名称，并让`"person"`元素的`type`属性来引用此名称（如使用此方法，若干元素均可引用相同的复合类型）：

```
<xs:element name="person" type="persontype"/>

<xs:complexType name="persontype">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

XSD 仅含文本

仅含文本的复合元素可包含`文本`和`属性`。

仅含文本的复合元素

此类型仅包含简易的内容（`文本和属性`），因此要向此内容添加`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，就像这样：

```
<xs:element name="somename">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="basetype">
        ....
        ....
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

或者：

```
<xs:element name="somename">
  <xs:complexType>
    <xs:simpleContent>
      <xs:restriction base="basetype">
        ....
        ....
      </xs:restriction>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

提示： 请使用 `extension` 或 `restriction` 元素来扩展或限制元素的 `基本简易类型`。 这里有一个 XML 元素的例子，`"shoesize"`，其中仅包含文本：

`<shoesize country="france">35</shoesize>`

下面这个例子声明了一个复合类型，其内容被定义为整数值，并且 `"shoesize"` 元素含有名为 `"country"` 的属性：

```
<xs:element name="shoesize">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:integer">
        <xs:attribute name="country" type="xs:string" />
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

也可为 `complexType` 元素设定一个名称，并让 `"shoesize"` 元素的 `type` 属性来引用此名称（通过使用此方法，若干元素均可引用相同的复合类型）：

```
<xs:element name="shoesize" type="shoetype"/>

<xs:complexType name="shoetype">
  <xs:simpleContent>
    <xs:extension base="xs:integer">
      <xs:attribute name="country" type="xs:string" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

XSD 混合内容

混合的复合类型可包含`属性`、`元素`以及`文本`。

带有混合内容的复合类型

XML 元素，`"letter"`，含有文本以及其他元素：

```
<letter>
  Dear Mr.<name>John Smith</name>.
  Your order <orderid>1032</orderid>
  will be shipped on <shipdate>2001-07-13</shipdate>.
</letter>
```

下面这个`schema`声明了这个`"letter"`元素：

```
<xs:element name="letter">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="orderid" type="xs:positiveInteger"/>
      <xs:element name="shipdate" type="xs:date"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

注意：为了使字符数据可以出现在`"letter"`的子元素之间，`mixed`属性必须被设置为`"true"`。`<xs:sequence>`标签（`name`、`orderid`以及`shipdate`）意味着被定义的元素必须依次出现在`"letter"`元素内部。
也可以为`complexType`元素起一个名字，并让`"letter"`元素的`type`属性引用`complexType`的这个名称（通过这个方法，若干元素均可引用同一个复合类型）：

```
<xs:element name="letter" type="lettertype"/>

<xs:complexType name="lettertype" mixed="true">
  <xs:sequence>
    <xs:element name="name" type="xs:string"/>
    <xs:element name="orderid" type="xs:positiveInteger"/>
    <xs:element name="shipdate" type="xs:date"/>
  </xs:sequence>
</xs:complexType>
```

XSD 指示器

通过 `指示器` ，可以控制在文档中使用元素的方式。

指示器

有七种指示器：

- `Order` 指示器：
  - All
  - Choice
  - Sequence
- `Occurrence` 指示器：
  - maxOccurs
  - minOccurs
- `Group` 指示器：
  - Group name
  - attributeGroup name

Order 指示器

Order 指示器用于定义 `元素的顺序` 。

All 指示器

`<all>` 指示器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：

```
<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>
```

注意： 当使用 `<all>` 指示器时，你可以把 `<minOccurs>` 设置为 0 或者 1，而只能把 `<maxOccurs>` 指示器设置为 1。

Choice 指示器

`<choice>` 指示器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）：

```
<xs:element name="person">
  <xs:complexType>
    <xs:choice>
      <xs:element name="employee" type="employee"/>
      <xs:element name="member" type="member"/>
    </xs:choice>
  </xs:complexType>
</xs:element>
```

Sequence 指示器

`<sequence>` 规定子元素必须按照特定的顺序出现：

```
<xs:element name="person">
   <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Occurrence 指示器

`Occurrence` 指示器用于定义某个元素出现的频率。

注意：对于所有的 `"Order"` 和 `"Group"` 指示器（`any、all、choice、sequence、group name 以及 group reference`），其中的 `maxOccurs` 以及 `minOccurs` 的默认值均为 `1`。

maxOccurs 指示器

`<maxOccurs>` 指示器可规定某个元素可出现的最大次数：

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="full_name" type="xs:string"/>
      <xs:element name="child_name" type="xs:string" maxOccurs="1
0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

上面的例子表明，子元素 `"child_name"` 可在 `"person"` 元素中最少出现一次（其中 `minOccurs` 的默认值是 1），最多出现 10 次。

minOccurs 指示器

`<minOccurs>` 指示器可规定某个元素能够出现的最小次数：

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="full_name" type="xs:string"/>
      <xs:element name="child_name" type="xs:string"
      maxOccurs="10" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

上面的例子表明，子元素 `"child_name"` 可在 `"person"` 元素中出现最少 0 次，最多出现 10 次。
提示：如需使某个元素的出现次数不受限制，请使用 `maxOccurs="unbounded"` 这个声明。

一个实际的例子：

名为 `"Myfamily.xml"` 的 XML 文件：

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

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

<person>
  <full_name>Hege Refsnes</full_name>
  <child_name>Cecilie</child_name>
</person>

<person>
  <full_name>Tove Refsnes</full_name>
  <child_name>Hege</child_name>
  <child_name>Stale</child_name>
  <child_name>Jim</child_name>
  <child_name>Borge</child_name>
</person>

<person>
  <full_name>Stale Refsnes</full_name>
</person>

</persons>
```

上面这个 XML 文件含有一个名为 `"persons"` 的根元素。在这个根元素内部，定义了三个 `"person"` 元素。每个 `"person"` 元素必须含有一个 `"full_name"` 元素，同时它可以包含多至 5 个 `"child_name"` 元素。
这是schema文件 `"family.xsd"` ：

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">

<xs:element name="persons">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="person" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="full_name" type="xs:string"/>
            <xs:element name="child_name" type="xs:string"
            minOccurs="0" maxOccurs="5"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>

</xs:schema>
```

Group 指示器

`Group` 指示器用于定义相关的数批元素。

元素组

元素组通过 `group` 声明进行定义：

```
<xs:group name="groupname">
...
</xs:group>
```

必须在 `group` 声明内部定义一个 `all`、`choice` 或者 `sequence` 元素。下面这个例子定义了名为 `"persongroup"` 的 `group`，它定义了必须按照精确的顺序出现的一组元素：

```
<xs:group name="persongroup">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
    <xs:element name="birthday" type="xs:date"/>
  </xs:sequence>
</xs:group>
```

在把 `group` 定义完毕以后，就可以在另一个定义中引用它了：

```
<xs:group name="persongroup">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
    <xs:element name="birthday" type="xs:date"/>
  </xs:sequence>
</xs:group>

<xs:element name="person" type="personinfo"/>

<xs:complexType name="personinfo">
  <xs:sequence>
    <xs:group ref="persongroup"/>
    <xs:element name="country" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

属性组

属性组通过 `attributeGroup` 声明来进行定义：

```
<xs:attributeGroup name="groupname">
...
</xs:attributeGroup>
```

下面这个例子定义了名为 `"personattrgroup"` 的一个属性组：

```
<xs:attributeGroup name="personattrgroup">
  <xs:attribute name="firstname" type="xs:string"/>
  <xs:attribute name="lastname" type="xs:string"/>
  <xs:attribute name="birthday" type="xs:date"/>
</xs:attributeGroup>
```

在已定义完毕属性组之后，就可以在另一个定义中引用它了，就像这样：

```
<xs:attributeGroup name="personattrgroup">
  <xs:attribute name="firstname" type="xs:string"/>
  <xs:attribute name="lastname" type="xs:string"/>
  <xs:attribute name="birthday" type="xs:date"/>
</xs:attributeGroup>

<xs:element name="person">
  <xs:complexType>
    <xs:attributeGroup ref="personattrgroup"/>
  </xs:complexType>
</xs:element>
```

XSD `<any>` 元素

`<any>`元素可以通过未被 schema 规定的元素来拓展 XML 文档！

`<any>` 元素

`<any>` 元素可以通过未被 schema 规定的元素来拓展 XML 文档！
下面这个例子是从名为 `"family.xsd"`的 `XML schema` 中引用的片段。它展示了一个针对 `"person"` 元素的声明。通过使用 `<any>`元素，可以通过任何元素（在 `<lastname>`之后）扩展 `"person"`的内容：

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
      <xs:any minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

现在，希望使用 `"children"`元素来扩展`"person"`元素。这此种情况下就可以这么做，即使以上这个 schema 的作者没有声明任何 “children” 元素。
请看这个 schema 文件，名为`"children.xsd"` ：

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.w3schools.com"
xmlns="http://www.w3schools.com"
elementFormDefault="qualified">

<xs:element name="children">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="childname" type="xs:string"
      maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

</xs:schema>
```

下面这个 XML 文件（名为 `"Myfamily.xml`“），使用了来自两个不同的 schema 中的成分，`"family.xsd"` 和 `"children.xsd"`：

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

<persons xmlns="http://www.microsoft.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.microsoft.com family.xsd
http://www.w3schools.com children.xsd">

<person>
  <firstname>Hege</firstname>
  <lastname>Refsnes</lastname>
  <children>
    <childname>Cecilie</childname>
  </children>
</person>

<person>
  <firstname>Stale</firstname>
  <lastname>Refsnes</lastname>
</person>

</persons>
```

上面这个 XML 文件是有效的，这是由于 `schema "family.xsd"` 允许通过在 `"lastname"` 元素后的可选元素来扩展 `"person"` 元素。
`<any>` 和 `<anyAttribute>` 均可用于制作可扩展的文档！它们使文档有能力包含未在主 `XML schema` 中声明过的附加元素。

XSD `<anyAttribute>` 元素

`<anyAttribute>` 元素可以通过未被 schema 规定的属性来扩展 XML 文档！

下面的例子是来自名为 `"family.xsd"` 的 `XML schema` 的一个片段。展示了针对 `"person"` 元素的一个声明。通过使用 `<anyAttribute>` 元素，就可以向 `"person"` 元素添加任意数量的属性：

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
    </xs:sequence>
    <xs:anyAttribute/>
  </xs:complexType>
</xs:element>
```

通过 `"gender"` 属性来扩展 `"person"` 元素。在这种情况下就可以这样做，即使这个 `schema` 的从未声明过任何 `"gender"` 属性。
请看这个 schema 文件，名为 `"attribute.xsd"` ：

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.w3schools.com"
xmlns="http://www.w3schools.com"
elementFormDefault="qualified">

<xs:attribute name="gender">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="male|female"/>
    </xs:restriction>
  </xs:simpleType>
</xs:attribute>

</xs:schema>
```

下面这个 XML（名为 `"Myfamily.xml"`），使用了来自不同 schema 的成分，`"family.xsd"` 和 `"attribute.xsd"`：

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

<persons xmlns="http://www.microsoft.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:SchemaLocation="http://www.microsoft.com family.xsd
http://www.w3schools.com attribute.xsd">

<person gender="female">
  <firstname>Hege</firstname>
  <lastname>Refsnes</lastname>
</person>

<person gender="male">
  <firstname>Stale</firstname>
  <lastname>Refsnes</lastname>
</person>

</persons>
```

上面这个 XML 文件是有效的，这是因为 schema `"family.xsd"` 允许向 `"person"` 元素添加属性。
`<any>` 和 `<anyAttribute>` 均可用于制作可扩展的文档！它们使文档有能力包含未在主 XML schema 中声明过的附加元素。

XSD 元素替换(Element Substitution)

通过 `XML Schema`，一个元素可对另一个元素进行替换。

元素替换

举例说明：用户来自英国和挪威。希望有能力让用户选择在 XML 文档中使用挪威语的元素名称还是英语的元素名称。
为了解决这个问题，可以在 `XML schema` 中定义一个 `substitutionGroup`。首先，声明主元素，然后会声明次元素，这些次元素可声明它们能够替换主元素。

```
<xs:element name="name" type="xs:string"/>
<xs:element name="navn" substitutionGroup="name"/>
```

在上面的例子中，`"name"` 元素是主元素，而 `"navn"` 元素可替代 `"name"` 元素。
请看一个 `XML schema` 的片段：

```
<xs:element name="name" type="xs:string"/>
<xs:element name="navn" substitutionGroup="name"/>

<xs:complexType name="custinfo">
  <xs:sequence>
    <xs:element ref="name"/>
  </xs:sequence>
</xs:complexType>

<xs:element name="customer" type="custinfo"/>
<xs:element name="kunde" substitutionGroup="customer"/>
```

有效的 XML 文档类似这样（根据上面的 schema）：

```
<customer>
  <name>John Smith</name>
</customer>
```

或类似这样：

```
<kunde>
  <navn>John Smith</navn>
</kunde>
```

阻止元素替换

为防止其他的元素替换某个指定的元素，请使用 `block` 属性：

`<xs:element name="name" type="xs:string" block="substitution"/>`

请看某个 `XML schema` 的片段：

```
<xs:element name="name" type="xs:string" block="substitution"/>
<xs:element name="navn" substitutionGroup="name"/>

<xs:complexType name="custinfo">
  <xs:sequence>
    <xs:element ref="name"/>
  </xs:sequence>
</xs:complexType>

<xs:element name="customer" type="custinfo" block="substitution"/>
<xs:element name="kunde" substitutionGroup="customer"/>
```

合法的 XML 文档应该类似这样（根据上面的 schema）：

```
<customer>
  <name>John Smith</name>
</customer>
```

但是下面的文档不再合法：

```
<kunde>
  <navn>John Smith</navn>
</kunde>
```

使用 substitutionGroup

可替换元素的类型必须和主元素相同，或者从主元素衍生而来。假如可替换元素的类型与主元素的类型相同，那么就不必规定可替换元素的类型了。
请注意，`substitutionGroup` 中的所有元素（主元素和可替换元素）必须被声明为`全局元素`，否则就无法工作！

什么是全局元素（Global Elements）？

全局元素指`"schema"` 元素的直接子元素！本地元素（`Local elements`）指嵌套在其他元素中的元素。

XSD 实例

演示如何编写一个`XML Schema`。

XML 文档

`"shiporder.xml"`的 XML 文档：

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<shiporder orderid="889923"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="shiporder.xsd">
  <orderperson>John Smith</orderperson>
  <shipto>
    <name>Ola Nordmann</name>
    <address>Langgt 23</address>
    <city>4000 Stavanger</city>
    <country>Norway</country>
  </shipto>
  <item>
    <title>Empire Burlesque</title>
    <note>Special Edition</note>
    <quantity>1</quantity>
    <price>10.90</price>
  </item>
  <item>
    <title>Hide your heart</title>
    <quantity>1</quantity>
    <price>9.90</price>
  </item>
</shiporder>
```

上面的XML文档包括根元素 `"shiporder"`，其中包含必须名为 `"orderid"` 的属性。
“shiporder” 元素包含三个不同的子元素：`"orderperson"`、`"shipto"` 以及 `"item"`。
“item” 元素出现了两次，它含有一个 `"title"`、一个可选 `"note"` 元素、一个 `"quantity"` 以及一个 `"price"` 元素。
上面这一行
`xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"`，告知XML解析器根据某个 schema 来验证此文档。
这一行：
`xsi:noNamespaceSchemaLocation="shiporder.xsd"` 规定了 schema 的位置（在这里，它与 `"shiporder.xml"` 处于相同的文件夹）。

创建一个 XML Schema

为上面这个 XML 文档创建一个 schema。
文件命名为 `"shiporder.xsd"`。要创建 `schema`，仅仅需要简单地遵循 XML 文档中的结构，定义所发现的每个元素。首先开始定义一个标准的 XML 声明：

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
...
</xs:schema>
```

在上面的 `schema` 中，使用了 `标准的命名空间 (xs)`，与此命名空间相关联的 `URI` 是 `Schema 的语言定义（Schema language definition）`，其标准值是 `http://www.w3.org/2001/XMLSchema`。
接下来，定义 `"shiporder"` 元素。此元素拥有一个属性，其中包含其他的元素，因此将它认定为 `复合类型`。 `"shiporder"` 元素的子元素被 `xs:sequence` 元素包围，定义了子元素的次序：

```
<xs:element name="shiporder">
  <xs:complexType>
    <xs:sequence>
      ...
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

然后把 `"orderperson"` 元素定义为 `简易类型`（这是因为它不包含任何属性或者其他的元素）。 `类型 (xs:string)` 的前缀是由命名空间的前缀规定的，此命名空间与指示预定义的 `schema 数据类型` 的 `XML schema` 相关联：

```
<xs:element name="orderperson" type="xs:string"/>
```

接下来，把两个元素定义为 `复合类型`： `"shipto"` 和 `"item"`。从定义 “shipto” 元素开始：

```
<xs:element name="shipto">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="address" type="xs:string"/>
      <xs:element name="city" type="xs:string"/>
      <xs:element name="country" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

通过 schema，可使用 `maxOccurs` 和 `minOccurs` 属性来定义某个元素可能出现的次数。`maxOccurs` 定义某元素出现次数的最大值，而 `minOccurs` 则定义某元素出现次数的最小值。`maxOccurs` 和 `minOccurs` 的默认值都是 1！
定义 `"item"` 元素。这个元素可在 `"shiporder"` 元素内部出现多次。这是通过把 `"item"` 元素的 `maxOccurs` 属性的值设定为 `"unbounded"` 来实现的，这样 `"item"` 元素就可出现任意多次。请注意，`"note"` 元素是可选元素。把此元素的 `minOccurs` 属性设定为 0 了：

```
<xs:element name="item" maxOccurs="unbounded">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="title" type="xs:string"/>
      <xs:element name="note" type="xs:string" minOccurs="0"/>
      <xs:element name="quantity" type="xs:positiveInteger"/>
      <xs:element name="price" type="xs:decimal"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

声明 `"shiporder"` 元素的属性了。由于这是一个必选属性，规定 `use="required"`。
注意：此属性的声明必须被置于最后：

```
<xs:attribute name="orderid" type="xs:string" use="required"/>
```

这是这个名为 `"shiporder.xsd"` 的 schema 文件的文档清单：

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="shiporder">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="orderperson" type="xs:string"/>
      <xs:element name="shipto">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="name" type="xs:string"/>
            <xs:element name="address" type="xs:string"/>
            <xs:element name="city" type="xs:string"/>
            <xs:element name="country" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="item" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="title" type="xs:string"/>
            <xs:element name="note" type="xs:string" minOccur
s="0"/>
            <xs:element name="quantity" type="xs:positiveInteger"/>
            <xs:element name="price" type="xs:decimal"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="orderid" type="xs:string" use="required"/>
  </xs:complexType>
</xs:element>

</xs:schema>
```

分割 Schema

前面的设计方法非常容易，但当文档很复杂时却难以阅读和维护。
接下来介绍的设计方法基于首先对所有元素和属性的定义，然后再使用 `ref 属性` 来引用它们。
这是用新方法设计的 `schema 文件("shiporder.xsd")` ：

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<!-- definition of simple elements -->
<xs:element name="orderperson" type="xs:string"/>
<xs:element name="name" type="xs:string"/>
<xs:element name="address" type="xs:string"/>
<xs:element name="city" type="xs:string"/>
<xs:element name="country" type="xs:string"/>
<xs:element name="title" type="xs:string"/>
<xs:element name="note" type="xs:string"/>
<xs:element name="quantity" type="xs:positiveInteger"/>
<xs:element name="price" type="xs:decimal"/>

<!-- definition of attributes -->
<xs:attribute name="orderid" type="xs:string"/>

<!-- definition of complex elements -->
<xs:element name="shipto">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="name"/>
      <xs:element ref="address"/>
      <xs:element ref="city"/>
      <xs:element ref="country"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="item">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="title"/>
      <xs:element ref="note" minOccurs="0"/>
      <xs:element ref="quantity"/>
      <xs:element ref="price"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="shiporder">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="orderperson"/>
      <xs:element ref="shipto"/>
      <xs:element ref="item" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute ref="orderid" use="required"/>
```

```
    </xs:complexType>
  </xs:element>

  </xs:schema>
```

使用指定的类型（Named Types）

第三种设计方法定义了类或者类型，这样可以重复使用元素的定义。具体的方式是：首先对简易元素和复合元素进行命名，然后通过元素的 `type 属性` 来指向它们
这是利用第三种方法设计的 `schema 文件 ("shiporder.xsd")` ：

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:simpleType name="stringtype">
  <xs:restriction base="xs:string"/>
</xs:simpleType>

<xs:simpleType name="inttype">
  <xs:restriction base="xs:positiveInteger"/>
</xs:simpleType>

<xs:simpleType name="dectype">
  <xs:restriction base="xs:decimal"/>
</xs:simpleType>

<xs:simpleType name="orderidtype">
  <xs:restriction base="xs:string">
    <xs:pattern value="[0-9]{6}"/>
  </xs:restriction>
</xs:simpleType>

<xs:complexType name="shiptotype">
  <xs:sequence>
    <xs:element name="name" type="stringtype"/>
    <xs:element name="address" type="stringtype"/>
    <xs:element name="city" type="stringtype"/>
    <xs:element name="country" type="stringtype"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="itemtype">
  <xs:sequence>
    <xs:element name="title" type="stringtype"/>
    <xs:element name="note" type="stringtype" minOccurs="0"/>
    <xs:element name="quantity" type="inttype"/>
    <xs:element name="price" type="dectype"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="shipordertype">
  <xs:sequence>
    <xs:element name="orderperson" type="stringtype"/>
    <xs:element name="shipto" type="shiptotype"/>
    <xs:element name="item" maxOccurs="unbounded" type="itemtype"/>
  </xs:sequence>
  <xs:attribute name="orderid" type="orderidtype" use="required"/>
</xs:complexType>
```

```
  <xs:element name="shiporder" type="shipordertype"/>

  </xs:schema>
```

`restriction` 元素显示出数据类型源自于 `W3C XML Schema` 命名空间的数据类型。因此，下面的片段也就意味着元素或属性的值必须是字符串类型的值：

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

`restriction 元素` 常被用于向元素施加限制。请看下面这些来自以上 schema 的片段：

```
<xs:simpleType name="orderidtype">
  <xs:restriction base="xs:string">
    <xs:pattern value="[0-9]{6}"/>
  </xs:restriction>
</xs:simpleType>
```

这段代码指示出，元素或属性的值必须为 `字符串` ，并且必须是 `连续的六个字符` ，同时这些字符必须是 `0-9 的数字` 。

XSD 字符串 数据类型

字符串数据类型用于可包含字符串的值。

字符串数据类型（String Data Type）

`字符串数据类型` 可包含 `字符、换行、回车以及制表符` 。
下面是一个关于某个 scheme 中字符串声明的例子：

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

文档中的元素看上去应该类似这样：

`<customer>John Smith</customer>`

或者类似这样：

`<customer> John Smith </customer>`

注意：如果使用字符串数据类型，XML 处理器就不会更改其中的值。

规格化字符串数据类型（NormalizedString Data Type）

规格化字符串数据类型源自于字符串数据类型。
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。
下面是一个关于在某个 schema 中规格化字符串数据类型的例子：

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

文档中的元素看上去应该类似这样：

```
<customer>John Smith</customer>
```

或者类似这样：

```
<customer> John Smith </customer>
```

注意：在上面的例子中，XML 处理器会使用空格替换所有的制表符。

Token 数据类型（Token Data Type）

`Token 数据类型` 同样源自于字符串数据类型。
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。
下面是在 schema 中一个有关 token 声明的例子：

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

文档中的元素看上去应该类似这样：

```
<customer>John Smith</customer>
```

或者类似这样：

```
<customer> John Smith </customer>
```

注意：在上面这个例子中，XML 解析器会移除制表符。

字符串数据类型

请注意，所有以下的数据类型均衍生于字符串数据类型（除了字符串数据类型本身）！

名称	描述
ENTITIES	
ENTITY	
ID	在 XML 中提交 ID 属性的字符串 (仅与 schema 属性一同使用)
IDREF	在 XML 中提交 IDREF 属性的字符串(仅与 schema 属性一同使用)

IDREFS language	包含合法的语言 id 的字符串
Name	包含合法 XML 名称的字符串
NCName	
NMTOKEN	在 XML 中提交 NMTOKEN 属性的字符串 (仅与 schema 属性一同使用)
NMTOKENS	
normalizedString	不包含换行符、回车或制表符的字符串
QName	
string	字符串
token	不包含换行符、回车或制表符、开头或结尾空格或者多个连续空格的字符串

对字符串数据类型的限定（Restriction）

可与字符串数据类型一同使用的限定：

- enumeration
- length
- maxLength
- minLength
- pattern (NMTOKENS、IDREFS 以及 ENTITIES 无法使用此约束)
- whiteSpace

XSD 日期和时间数据类型

`日期及时间数据类型` 用于包含 `日期` 和 `时间` 的值。

日期数据类型（Date Data Type）

日期数据类型用于定义日期。
日期使用此格式进行定义：`"YYYY-MM-DD"`，其中：

- YYYY 表示年份
- MM 表示月份
- DD 表示天数

注意：所有的成分都是必需的
下面是一个有关 `schema` 中日期声明的例子：

```
<xs:element name="start" type="xs:date"/>
```

文档中的元素看上去应该类似这样：

```
<start>2002-09-24</start>
```

时区

如需规定一个时区,可以通过在日期后加一个 `"Z"` 的方式，使用世界调整时间（UTC time）来输入一个日期 - 比如这样：

```
<start>2002-09-24Z</start>
```

或者也可以通过在日期后添加一个正的或负时间的方法，来规定以世界调整时间为准的 `偏移量` - 比如这样：

```
<start>2002-09-24-06:00</start>
```

或者

```
<start>2002-09-24+06:00</start>
```

时间数据类型（Time Data Type）

`时间数据` 类型用于定义 `时间`。
时间使用下面的格式来定义：`"hh:mm:ss"`，其中

- hh 表示小时
- mm 表示分钟
- ss 表示秒

注意： 所有的成分都是必需的！
下面是一个有关 schema 中时间声明的例子：

```
<xs:element name="start" type="xs:time"/>
```

文档中的元素看上去应该类似这样：

```
<start>09:00:00</start>
```

或者类似这样：

```
<start>09:30:10.5</start>
```

时区

如需规定一个时区，可以通过在时间后加一个 `"Z"` 的方式，使用世界调整时间（UTC time）来输入一个时间 - 比如这样：

```
<start>09:30:10Z</start>
```

或者也可以通过在时间后添加一个正的或负时间的方法，来规定以世界调整时间为准的 `偏移量` - 比如这样：

```
<start>09:30:10-06:00</start>
```

or

```
<start>09:30:10+06:00</start>
```

日期时间数据类型（DateTime Data Type）

日期时间数据类型用于定义日期和时间。
日期时间使用下面的格式进行定义：`"YYYY-MM-DDThh:mm:ss"`，其中：

- YYYY 表示年份
- MM 表示月份
- DD 表示日
- T 表示必需的时间部分的起始
- hh 表示小时
- mm 表示分钟
- ss 表示秒

注意： 所有的成分都是必需的！
下面是一个有关 schema 中日期时间声明的例子：

```
<xs:element name="startdate" type="xs:dateTime"/>
```

文档中的元素看上去应该类似这样：

```
<startdate>2002-05-30T09:00:00</startdate>
```

或者类似这样：

```
<startdate>2002-05-30T09:30:10.5</startdate>
```

时区

如需规定一个时区，您也可以通过在日期时间后加一个 "Z" 的方式，使用世界调整时间（UTC time）来输入一个日期时间 - 比如这样：

```
<startdate>2002-05-30T09:30:10Z</startdate>
```

或者也可以通过在时间后添加一个正的或负时间的方法，来规定以世界调整时间为准的偏移量 - 比如这样：

```
<startdate>2002-05-30T09:30:10-06:00</startdate>
```

或者

```
<startdate>2002-05-30T09:30:10+06:00</startdate>
```

持续时间数据类型（Duration Data Type）

持续时间数据类型 用于 规定时间间隔 。
时间间隔使用下面的格式来规定："PnYnMnDTnHnMnS"，其中：

- P 表示周期(必需)
- nY 表示年数
- nM 表示月数
- nD 表示天数
- T 表示时间部分的起始（如果您打算规定小时、分钟和秒，则此选项为必需）
- nH 表示小时数
- nM 表示分钟数
- nS 表示秒数
  下面是一个有关 schema 中持续时间声明的例子：

  ```
  <xs:element name="period" type="xs:duration"/>
  ```

文档中的元素看上去应该类似这样：

```
<period>P5Y</period>
```

上面的例子表示一个 5 年的周期。
或者类似这样：

```
<period>P5Y2M10D</period>
```

上面的例子表示一个 5 年、2 个月及 10 天的周期。
或者类似这样：

```
<period>P5Y2M10DT15H</period>
```

上面的例子表示一个 5 年、2 个月、10 天及 15 小时的周期。
或者类似这样：

```
<period>PT15H</period>
```

上面的例子表示一个 15 小时的周期。

负的持续时间

如需规定一个负的持续时间，请在 `P` 之前输入 `减号` ：

```
<period>-P10D</period>
```

上面的例子表示一个负 10 天的周期。

日期和时间数据类型

名称	描述
date	定义一个日期值
dateTime	定义一个日期和时间值
duration	定义一个时间间隔
gDay	定义日期的一个部分 - 天 (DD)
gMonth	定义日期的一个部分 - 月 (MM)
gMonthDay	定义日期的一个部分 - 月和天 (MM-DD)
gYear	定义日期的一个部分 - 年 (YYYY)
gYearMonth	定义日期的一个部分 - 年和月 (YYYY-MM)
time	定义一个时间值

对日期数据类型的限定（Restriction）

可与日期数据类型一同使用的限定：

- enumeration
- maxExclusive
- maxInclusive
- minExclusive
- minInclusive
- pattern

- whiteSpace

XSD 数值数据类型

十进制数据类型

十进制数据类型用于规定一个数值。
下面是一个关于某个 scheme 中十进制数声明的例子。

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

文档中的元素看上去应该类似这样：

```
<prize>999.50</prize>
```

或者类似这样：

```
<prize>+999.5450</prize>
```

或者类似这样：

```
<prize>-999.5230</prize>
```

或者类似这样：

```
<prize>0</prize>
```

或者类似这样：

```
<prize>14</prize>
```

注意： 可规定的十进制数字的最大位数是 `18 位`。

整数数据类型

整数数据类型用于规定无小数成分的数值。
下面是一个关于某个 scheme 中整数声明的例子。

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

文档中的元素看上去应该类似这样：

```
<prize>999</prize>
```

或者类似这样：

```
<prize>+999</prize>
```

或者类似这样：

```
<prize>-999</prize>
```

或者类似这样：

```
<prize>0</prize>
```

数值数据类型

请注意，下面所有的数据类型均源自于十进制数据类型（除 decimal 本身以外）！

名字	秒数
byte	有正负的 8 位整数
decimal	十进制数
int	有正负的 32 位整数
integer	整数值
long	有正负的 64 位整数
negativeInteger	仅包含负值的整数 (.., -2, -1.)
nonNegativeInteger	仅包含非负值的整数 (0, 1, 2, ..)
nonPositiveInteger	仅包含非正值的整数 (.., -2, -1, 0)
positiveInteger	仅包含正值的整数 (1, 2, ..)
short	有正负的 16 位整数
unsignedLong	无正负的 64 位整数
unsignedInt	无正负的 32 位整数
unsignedShort	无正负的 16 位整数
unsignedByte	无正负的 8 位整数

对数值数据类型的限定（Restriction）

可与数值数据类型一同使用的限定：

- enumeration
- fractionDigits
- maxExclusive
- maxInclusive
- minExclusive
- minInclusive
- pattern
- totalDigits
- whiteSpace

XSD 杂项 数据类型

其他杂项数据类型包括`布尔`、`base64Binary`、`十六进制`、`浮点`、`双精度`、`anyURI`以及`NOTATION`。

布尔数据类型（Boolean Data Type）

布尔数据性用于规定 true 或 false 值。
下面是一个关于某个 scheme 中逻辑声明的例子：

```
<xs:attribute name="disabled" type="xs:boolean"/>
```

文档中的元素看上去应该类似这样：

```
<prize disabled="true">999</prize>
```

注意： 合法的布尔值是`true、false、1（表示 true） 以及 0（表示 false）`。

二进制数据类型（Binary Data Types）

二进制数据类型用于表达二进制形式的数据。
可使用两种二进制数据类型：

- base64Binary (Base64 编码的二进制数据)
- hexBinary (十六进制编码的二进制数据)

下面是一个关于某个 scheme 中 hexBinary 声明的例子：

```
<xs:element name="blobsrc" type="xs:hexBinary"/>
```

AnyURI 数据类型（AnyURI Data Type）

anyURI 数据类型用于规定 URI。
下面是一个关于某个 scheme 中 anyURI 声明的例子：

```
<xs:attribute name="src" type="xs:anyURI"/>
```

文档中的元素看上去应该类似这样：

```
<pic src="http://www.w3schools.com/images/smiley.gif" />
```

注意： 如果某个 URI 含有空格，请用`%20`替换它们。

杂项数据类型

名称	描述
anyURI	
base64Binary	
boolean	
double	
float	
hexBinary	
NOTATION	
QName	

对杂项数据类型的限定（Restriction）

可与杂项数据类型一同使用的限定：

- enumeration (布尔数据类型无法使用此约束)
- length (布尔数据类型无法使用此约束)
- maxLength (布尔数据类型无法使用此约束)
- minLength (布尔数据类型无法使用此约束)
- pattern
- whiteSpace
  约束指 *constraint*

XML Schema 参考手册

XML Schema 支持数据类型（data type）和命名空间（namespace）。

XSD 元素

元素	解释
all	规定子元素能够以任意顺序出现，每个子元素可出现零次或一次。
annotation	annotation 元素是一个顶层元素，规定 schema 的注释。
any	使创作者可以通过未被 schema 规定的元素来扩展 XML 文档。
anyAttribute	使创作者可以通过未被 schema 规定的属性来扩展 XML 文档。
appInfo	规定 annotation 元素中应用程序要使用的信息。
attribute	定义一个属性。
attributeGroup	定义在复杂类型定义中使用的属性组。
choice	仅允许在 `<choice>` 声明中包含一个元素出现在包含元素中。
complexContent	定义对复杂类型（包含混合内容或仅包含元素）的扩展或限制。
complexType	定义复杂类型。
documentation	定义 schema 中的文本注释。
element	定义元素。
extension	扩展已有的 simpleType 或 complexType 元素。
field	规定 XPath 表达式，该表达式规定用于定义标识约束的值。
group	定义在复杂类型定义中使用的元素组。
import	向一个文档添加带有不同目标命名空间的多个 schema。
include	向一个文档添加带有相同目标命名空间的多个 schema。
key	指定属性或元素值（或一组值）必须是指定范围内的键。
keyref	规定属性或元素值（或一组值）对应指定的 key 或 unique 元素的值。
list	把简单类型定义为指定数据类型的值的一个列表。
notation	描述 XML 文档中非 XML 数据的格式。

redefine	重新定义从外部架构文件中获取的简单和复杂类型、组和属性组。
restriction	定义对 simpleType、simpleContent 或 complexContent 的约束。
schema	定义 schema 的根元素。
selector	指定 XPath 表达式，该表达式为标识约束选择一组元素。
sequence	要求子元素必须按顺序出现。每个子元素可出现 0 到任意次数。
simpleContent	包含对 complexType 元素的扩展或限制且不包含任何元素。
simpleType	定义一个简单类型，规定约束以及关于属性或仅含文本的元素的值的信息。
union	定义多个 simpleType 定义的集合。
unique	指定属性或元素值（或者属性或元素值的组合）在指定范围内必须是唯一的。

ref

XSD 限定/Facets

限定	**描述**
enumeration	定义可接受值的一个列表
fractionDigits	定义所允许的最大的小数位数。必须大于等于0。
length	定义所允许的字符或者列表项目的精确数目。必须大于或等于0。
maxExclusive	定义数值的上限。所允许的值必须小于此值。
maxInclusive	定义数值的上限。所允许的值必须小于或等于此值。
maxLength	定义所允许的字符或者列表项目的最大数目。必须大于或等于0。
minExclusive	定义数值的下限。所允许的值必需大于此值。
minInclusive	定义数值的下限。所允许的值必需大于或等于此值。
minLength	定义所允许的字符或者列表项目的最小数目。必须大于或等于0。
pattern	定义可接受的字符的精确序列。
totalDigits	定义所允许的阿拉伯数字的精确位数。必须大于0。
whiteSpace	定义空白字符（换行、回车、空格以及制表符）的处理方式。