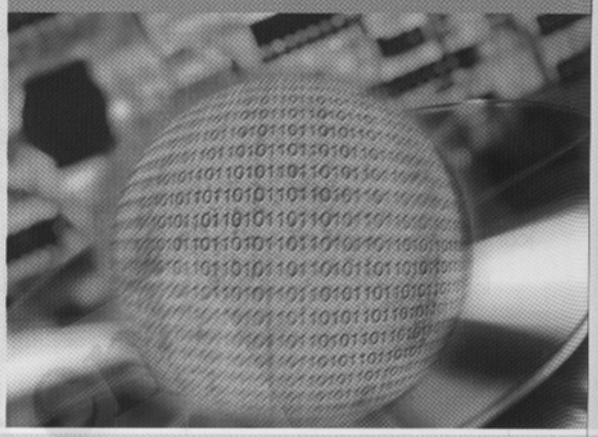
XML Schema 中的面向对象思想

Building XML Schemas Based on Object-Oriented Thinking

李浩 沈琦 （徐州中国矿业大学计算机科学与技术学院 221008）



摘 要：XML Schema 是一种用来描述信息结构的机制，可用来定义 XML 文档的结构、数据类型等内容。为了在编写 XML 实例文档时更具灵活性，可考虑应用 XMLSchema 中的面向对象思想。

关键词：面向对象 XML Schema 封装 继承 多态

1 引言

XML 模式（XMLSchema）提供了一组对 XML 文档的词汇表和语法进行约束和形式化的功能强大的工具。随着 XML 迅速地发展成为今后数据传输的格式，有一点很清楚：必须以有组织的方式来创建和存储 XML 的结构（由模式概述）。有面向对象设计经验的开发人员都知道，一个灵活的体系结构能在整个系统中确保一致性并能帮助适应增长与变化。故用面向对象的思想来构建XML模式是非常必要的，因为这有助于设计出可扩展的、灵活的和模块化的 XML 实例文档。下面对XMLSchema中的面向对象思想作详细分析。

2 封装

实际上，XML模式的本质作用就是封装XML元素。在面向对象思想中，封装指的是使对象成为黑箱的概念，这样当使用对象时就不知道它的内部工作原理。在模式方面，这一概念就变成了创建这样的类型：类型是预先定义的，并且只要引用该类型就可以很容易地在任何地方创建和交换数据。

现在创建一个 Book 类型，它指明一本书必须有 Author（作者）、Title（书名）和ISBN。因此，Book 类型封装了所有与书有关的信息。将该数据类型放入名为 DataTypes.xsd 的数据类型模式中。该模式包含可以被许多不同模式使用的所有一般数据类型。例如，Sales、Product 和 Accounting 模式都需要 Book 的定义。

DataTypes.xsd:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified"
attributeFormDefault="unqualified">
  <xs:complexType name="Book">
     <xs:sequence>
     <xs:element name="Title" type="xs:string"/>
      <xs:element name="Author" type="xs:string"/>
<xs:element name="ISBN" type="xs:string"/>
   </xs:sequence>
</xs:complexType>
</xs:schema>
```

文档1

在这个文档中，我们定义了一个名为 “Book”的组件。其实它只是一个复杂元素类型，每个复杂元素类型都描述一个组件的详细信息。为了成为可引用的元素，元素必须是全局元素，也就是说，是<schema> 的直接子元素。局部元素是嵌套在另一个组件内的元素声明；例如，Title 元素是全局元素 Book 内的局部元素。现有一个元素 Product，它包含许多产品，其中就有Book。当我们编写Product的XSD文档时，可以直接将<xs:element name="Book" type="Book"/>加入即可。这样便实现了面向对象思想中的封装性。

3 继承

软件重用是面向对象设计的另一个重要部分。可以通过继承实现软件重用。在编程语言中，通过子类提供这一能力。在 XML 模式中，可以使用抽象类型或仅使用指定基类型扩展或派生的标记来做到这一点。

3.1 通过定义抽象类型实现继承

抽象类型不能在实例文档中使用；它们只是为其派生类型提供占位符。在下面的例子中，将 Book定义为 抽象类型，同时定义另一个元素aBook,此元素不但有Title 、Author和ISBN，还有Price。很明显，aBook由Book派生而来，故可以继承Title、Author和ISBN三个元素而添加Price元素。为了将Book定义为抽象类，需添加 “abstract= “true”” 属性。<xs:complexType name="Book" abstract="true">。而aBook的继承是通过<extension base="Book">语句来实现的。

```
<xs:complexType name="aBook">
        <xs:complexContent>
            <xs:extension base="Book">
                <xs:sequence maxOccurs="unbounded">
                    <xs:element name="Price"
                     type="xs:string"/>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
```

文档2

3.2 通过扩展实现继承

另一个继承的示例是使用不带抽象类型的扩展。BookSales 类型包含有关书的信息，并且包含书的销售数量。可以通过使用 extension base 关键字扩展 Book 类型以创建 BookSales 类型。

```
<xs:complexType name="BookSales">
    <xs:complexContent>
        <xs:extension base="Book">
            <xs:sequence>
            <xs:element name="Number" type="xs:integer"/>
        </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

文档3

这意味着无论何时引用元素 BookSales，XML 实例文档都将在包含书名、作者和ISBN 的同时包含销量。

3.3 通过限制实现继承

通过限制实现的继承在希望创建基类型的子集的情况下有用。一个这样的示例是限制值的范围。在本例中，希望限制 Pamphlet（小册子）的定义，除了没有作者以外，它与 Book 完全相似。在创建小册子时，可以使用 restriction base="Book"语句。

```
〈xs:complexType name= "Pamphlet">
    <xs:complexContent>
        <xs:restriction base="Book">
            <xs:sequence>
                <xs:element name="Title" type="xs:string"/>
<xs:element name="ISBN" type="xs:string"/>
            </xs:sequence>
        </xs:restriction>
    </xs:complexContent>
</xs:complexType>
```

文档4

3.4 禁止继承

在编程时，可以将一些接口和类声明为 final，这样就不能用它们产生子类。通过使一些组件成为 final，可以在模式中实现同样的目标。例如，出版公司 有一个非常严格的描述自己的方法：名称、总部和 CEO。公司不希望任何人能够扩展这个定义。因此，它将组件 Company 声明为 final，如下所示。当使用关键字 #all 时，对组件既不能扩展也不能限制。在另外两种情况下，final 会禁止扩展或限制。

```
<xsd:complexType name=" Company " final="#all">
<xsd:complexType name=" Company " final="extension">
<xsd:complexType name=" Company " final="restriction">
```

文档5

4 多态性

多态性意味着在不同的上下文中对某对象赋予不同的意义或用法的能力，具体而言，就是允许对象有多种形式。在象 Java 技术这样的编程语言中，多态性指的是对输入的不同反应。更具体地说，它是子类对相同消息做出不同反应的能力。简单的继承允许两个子类各自添加不同的方法，而多态性则为了实现同一功能，但用的是不同的方法和不同的子类。因为 XML 不是行为语言，所以多态性出现在属性级别。让我们采用上面的继承示例。我们说过 Pamphlet 从 Book 继承 Title 和 ISBN 属性。然而，现在希望指定 Pamphlet 的 ISBN 具有其专有的特征，而且与 Book 的特征不同。换句话说，知道 Pamphlet 的 ISBN 一直有五位数字的限制。希望在模式中实施这一规则，以便在获得五位数以上时可验证出错误。因此需要构建一个名为 PamphletISBN

的新类型的组件。从类型 ISBN 派生该组件并限制其 ISBN 最大为五位。然后构造 Pamphlet 组件。但是，这一次将 ISBN 的类型设置为 PamphletISBN，这样就可保证虽然元素的名称将保持为 ISBN，但验证程序会检查更受限制的 PamphletISBN。改变子类型中派生组件的实现的能力演示了 XML 模式中多态性的一种形式。

```
<xs:complexType name="Pamphlet">
    <xs:complexContent>
        <xs:restriction base="Book">
            <xs:sequence>
                <xs:element name="Title"
                    type="xs:string"/>
<xs:element name="ISBN"
                    type="PamphletISBN"/>
            </xs:sequence>
        </xs:restriction>
    </xs:complexContent>
</xs:complexType>
<xs:simpleType name="ISBN">
    <xs:restriction base="xs:string"/>
</xs:simpleType>
<xs:simpleType name="PamphletISBN">
    <xs:restriction base="ISBN">
        <xs:maxLength value="5"/>
    </xs:restriction>
</xs:simpleType>
```

文档6

5 用于去耦合或内聚的设计模式

通过去耦合（decoupling）和内聚（cohesiveness）的设计模式，可以任意改变数据类型的粒度。目前有三种设计模式，它们代表着创建组件时的三种级别的粒度。

5.1 俄罗斯娃娃

指的是将所有相关组件都包含于一个组件之内。如类型 Book 由组件 Title、Author 和 ISBN 组成。这些组件都在 Book 组件内局部定义。俄罗斯娃娃式设计是最不易扩展的，因为四个类型中有三个是局部定义的，所以它们不能被任何其他组件访问。示例参见文档1。

5.2 意大利香肠片

通过引用不同的类型将组件组合在一起或聚合在一起。这样，Book、Title、Author 和 ISBN 每一个都是全局元素。Book 类型则引用其他三个元素作为其定义的部分。

```
<xs:element name="Title" type="string"/>
<xs:element name="Author" type="string"/>
<xs:element name="ISBN" type="integer"/>
<xs:element name="Book">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="Title"/>
            <xs:element ref="Author"/>
            <xs:element ref="ISBN"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

文档7

5.3 软百叶窗

将所有的元素和组件都定义为类型。这意味着当定义名为"Title"的组件时，即使它是字符串这一事实决定了它是一个简单类型，也需要通过类型"Title"引用它。这表明了将组件构造到其最细微阶段中的最高级别。

```
<xs:simpleType name="Title">
 <xs:restriction base="xs:string"/>
</xs:simpleType>
 <xs:simpleType name="Name">
    <xs:restriction base="xs:string">
        <xs:minLength value="1"/>
    </xs:restriction>
</xs:simpleType>
<xs:complexType name="Book">
    <xs:sequence>
        <xs:element name="Title" type="Title"/>
        <xs:element name="Author" type="Name"/>
    </xs:sequence>
</xs:complexType>
```

文档8

如何在这三种模式中进行选择取决于业务需求。如果模式大小是要考虑的问题，那么俄罗斯娃娃比较适合。然而，如果可扩展性是主要的关注对象，那么软百叶窗就是可应用的最佳设计模式。

6 通过使用名称空间模拟包

面向对象编程非常强调根据类的服务来封装它们。包结构对代码加以组织并推动模块化和维护。通过根据 XML 模式的功能来组织它

们，可以获得相似的好处。例如，会计部门创建自己的模式Accounting.xsd，而销售部门用Marketing.xsd指定其数据类型和定义。为了区别这些模式，除了将它们保存为不同的文件之外还必须做更多的工作 - 需要为它们分配单独的名称空间，因为两个部门定义的元素有可能同名，而类型并不相同。这样，当在不同的部门交换数据时我们就可以利用关键字include schemaLocation来指定数据类型来源。同时，如果公司想使用业界标准，但希望添加专有的信息以进行内部通信和存储，它们可使用前面描述的方法扩展业界标准。现在假设已定义了Fees和Sales模式，它们分别属于不同的命名空间namespace= "http://www.dist.com" 和targetNamespace= "http://www.bond.com"。将Fees 模式导入到Sales模式：

```
<xs:include schemaLocation="DataType.xsd"/>
  <xs:import namespace="http://www.Dist.com"
    schemaLocation="Fees.xsd"/>
  <xs:element name="Sales">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Books">
          <xs:complexType>
            <xs:sequence maxOccurs="unbounded">
              <xs:element name="BookSales" type="BookSales"/>
              <xs:element ref="dist:Fees"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
```

文档9

这样当我们生成XML实例文档时会用前缀 dist 限定所有由 Dist 名称空间定义的信息，即Sales.xml 中任何特定于 Dist 的元素都有这个前缀。因此，通过赋予元素或数据类型以不同的名称空间，可以模拟包的功能，并通过导入行为扩展包的定义。

7 结束语

由以上讨论可看出，在设计XML数据类型时可借鉴面向对象编程思想，这有助于使XML数据成为可扩展的、灵活的和模块化的。

参考文献

1 庞丽萍、印?F，软件开发基础-面向对象技术与操作系统虚拟机，高等教育出版社，1999。

2 郑小平，.NET精髓-web服务原理与开发，人民邮电出版社，2002.1。

3 景建萍、印旻，XML基础与应用教程，高等教育出版社，2001.7。