1 部分

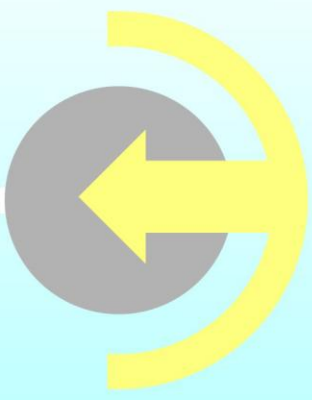
XML概述与基本语法

第一次课 XML概述

第二次课 XML基本语法

欢迎大家来到XML学习的天地。在学习XML之前，建议您最好具备HTML基本的知识，如果曾动手写过HTML则更好。

第一部分主要介绍了XML概述与XML基本语法,在XML概述中主要介绍了为什么要学习XML，什么是XML，XML的发展历程， XML的体系结构，XML在行业方面的各种应用。XML的基本语法中,主要讲解了XML文档的结构和基础语法，在XML中如何定义元素，如何定义属性，处理指令概念，文本内容的定义，在XML中注释的编写等。通过这部分的学习，大家在了解XML的基础上可以根据需要熟练编写一个结构良好的XML文档。



第一次课　XML 概述

学习内容	难度
※为什么要学习 XML	★★
※什么是 XML	★★
XML 历史	★★★
XML 的应用	★★
XML 技术架构概要介绍	★★

注：带※为重点学习内容

笔记区

1.1 为什么要学 XML

随着 Internet 的迅速发展和广泛普及，大量的信息通过互联网进行发布，以实现信息的共享和交流。人们通过 Internet 每天都可以从世界各地获得大量的信息。在基于 B/S 这种架构上，HTML 对整个互联网这几年来的发展，可以说是第一功臣，HTML 在互联网上无处不在。尽管 HTML 在人机界面交互这方面很拿手，但是却非常不利于机器之间互相交流、传递信息。我们来看个网络书店的实例,下面这段 HTML 码,将书籍信息以表格(table)方式排列.

Code 1-1

```
<h1>最热卖书籍</h1>
<table style="width:100%;" bgcolor="#3333CC" border=0 cellspacing=1 >
        <tr bgcolor="White">
            <td class="style5">书籍编号</td>
            <td class="style6">书称</td>
            <td class="style7">单价</td>
            <td class="style8">作者</td>
            <td class="style9">出版社</td>
        </tr>
        <tr bgcolor="White">
            <td class="style1">YH001</td>
            <td class="style2">企业级应用数据传输技术</td>
            <td class="style3">48</td>
            <td class="style4">漆美云，肖发辉，李灯登</td>
            <td>电子出版社</td>
        </tr>
        <tr bgcolor="White">
            <td class="style1">YH002</td>
            <td class="style2">SQL2005数据库开发</td>
            <td class="style3">60</td>
            <td class="style4">漆美云，王勤</td>
            <td>电子出版社</td>
```


```
        </tr>
        <tr bgcolor="White">
            <td class="style1">YH003</td>
            <td class="style2">JavaScript&amp;amp;Ajax开发</td>
            <td class="style3">68</td>
            <td class="style4">漆美云，沈慧芳，何翠云</td>
            <td>电子出版社</td>
        </tr>
        </table>
```

在浏览器中呈现的效果如下:

最热卖书籍

书籍编号	书称	单价	作者	出版社
YH001	企业级应用数据传技术	48	漆美云，肖发辉，李灯登	电子出版社
YH002	SQL2005 数据库开发	60	漆美云，王勤	电子出版社
YH003	JavaScript&Ajax 开发	68	漆美云，沈慧芳，何翠云	电子出版社

图 1-1

但 HTML 的问题正出在,HTML 的标签大多是设计来呈现文章的布局和外观的,譬如像上例中<table><tr><td>还有像
<font>等比比皆是。

今天如果我要利用机器爬虫程序到网络上各电子商站去自动把最新的价目抓回来,让我们的客户可以藉此比价，从此再也不需要辛苦地一家家网站亲自去看。

这时问题出现了，书店甲可能使用 HTML 表格来呈现，但书店乙可能是 … 的的条列方式，而书店丙可能用的又是另一套。更严重的是 HTML 中可能有一大长串长得非常类似的标签。到底哪些才是商品名称和价目的标签？要如何将它们可靠地取出来。这就有点麻烦了。我们的爬虫程序如果不是特别聪明，那根本别想从各种各样的信息中找出关键信息，达成任务。如果这些网页信息只是纯供人类阅读，消化，那没有问题，但是在信息爆炸的网络世界，人类需要逐渐依赖机器来替我们多分忧解劳，以 HTML 码来标注的信息，对机器和编程的人来说，不但太残忍了点，处理起来更是极度缺乏效率。

我们一起来仔细思考一下上面这个例子,其实线上商店里的信息，在还没有制作成 HTML 网页这前，可能都是一笔笔按照栏位存储在数据库或者购物车中，如上图 1-1 所示

笔记区

但一旦从数据库中取出来，转成 HTML 格式后，这些重要的栏位、项目名称全变成了毫无意义的<h1>,<h3>或者 <th><td>……,换句话说,原本建在数据库中非常宝贵的信息架构，在商品信息转换为 HTML 提供给客户的同时，荡然无存。试想：今天我们如果能将数据库中的资料保留原本完整的资料架购，一五一十地在自己和客户的电脑之间互传，必定大大有利双方把资料快速地建入自己的数据库内，进而大幅提升效率，这如果用 HTML 来做，恐怕很困难。

我们尝试来用 XML 来存储线上商品的信息.在 XML 里,我们可以自由定义标签,XML 针对会使用 HTML 的人轻松上手.因为例子的写法和 HTML 类似,把上面的例子改成 XML 如下:

Code 1-2

```
<书籍 编号=”YH001”>
<名称>企业级应用数据传输技术</名称>
<作者>漆美云,肖发辉,李灯登</作者>
<单价>48</单价>
</书籍>
<书籍 编号=”YH002”>
<名称>SQL2005 数据库开发</名称>
<作者>漆美云,王勤</作者>
<单价>60</单价>
</书籍>
```

Year,竟然连中文都是中文的,是的,这也是 XML 的一大特性.XML 在设计之初就考虑到国际化的问题,因此打从一开始便构建在 Unicode 统一码之上,对于身为中文使用者的我们来说,真的是非常不错的考虑.

有了 XML 以后,前述的自动化比价程序,突然间变得很容易实现了,我们的机器爬虫程序只要到线上书店的 XML 网页中去找<书籍>区块中的<名称>和<单价>就可以了,将各家书店的名称和单价一一抓回来进行比较就可以顺序完成任务了.

当今的网络世界,电子商务日渐兴盛,异质电脑系统之间的互动日益频繁(商务信息传递),自动化程序也越来越重要（譬如前述的爬虫程序），我们需要一个比目前 HTML 更好的方法,否则未来网络的发展将会越来越慢。欢迎来到 XML 的天地,XML 可以来帮大家解决这些问题。

笔记区

1.2 什么是 XML?

在了解什么是 XML 之前，先要了解 XML 和 HTML 的区别。

HTML 主要是用来显示数据的

XML 是用来存放数据的

XML 不是 HTML 的替代品和升级品，XML 和 HTML 是两种不同用途的语言。

XML 是被设计用来描述数据的，重点是：什么是数据，如何存放数据。

HTML 是被设计用来显示数据的，重点是：显示数据以及如何显示数据更好上面。

HTML 是与显示信息相关的, XML 则是与描述信息相关的。

XML（eXtensible Markup Language）即可扩展标记语言，它与 HTML 一样，都是 SGML(Standard Generalized Markup Language,标准通用标记语言)。XML 是 Internet 环境中跨平台的，依赖于内容的技术，是当前处理结构化文档信息的有力工具。扩展标记语言 XML 是一种简单的数据存储语言，使用一系列简单的标记描述数据，而这些标记可以用方便的方式建立，虽然 XML 占用的空间比二进制数据要占用更多的空间，但 XML 极其简单易于掌握和使用。

XML 与 Access,Oracle 和 SQL Server 等数据库不同，这些数据库提供了更强有力的数据存储和分析能力，例如：数据索引、排序、查找、相关一致性等，XML 仅仅是存储数据。事实上 XML 与其他数据表现形式最大的不同是：他极其简单。我们经常把数据集说成是临时的数据库。如果我们把这个数据集比喻是仓库，那么 XML 就是火车、轮船上的集装箱，数据集负责临时存储数据，XML 负责装运数据时的容器，相比数据集来讲 XML 甚至还要更重要一些。原因很简单，联系世界的纽带中少不了它，它的作用是直接的、广域的、世界性的。XML 主要用来作为系统与系统之间传输数据时的载体。

XML 的简单使其易于在任何应用程序中读写数据，这使 XML 很快成为数据交换的唯一公共语言，虽然不同的应用软件也支持其它的数据交换格式，但不久之后他们都将支持 XML，那就意味着程序可以更容易的与 Windows、Mac OS, Linux 以及其他平台下产生的信息结合，然后可以很容易加载 XML 数据到程序中并分析他，并以 XML 格式输出结果。

为了使得 SGML 显得用户友好，XML 重新定义了 SGML 的一些内部值和参数，去掉了大量的很少用到的功能，这些繁杂的功能使得 SGML 在设计网站时显得复杂化。XML 保留了 SGML 的结构化功能，这样就使得网站设计者可以定义自己的文档类型，XML 同时也推出一种新型文档类型，使得开发者也可以不必定义文档类型。

笔记区

因为 XML 是 W3C 制定的，XML 的标准化工作由 W3C 的 XML 工作组负责，该小组成员由来自各个地方和行业的专家组成，他们通过 email 交流对 XML 标准的意见，并提出自己的看法 (www.w3.org/TR/WD-xml)。因为 XML 是个公共格式， (它不专属于任何一家公司)，你不必担心 XML 技术会成为少数公司的盈利工具，XML 不是一个依附于特定浏览器的语言

你可以在 W3C 位于 http://www.w3c.org /TR/REC-xml 的 Web 站点上阅读整篇文档。正如所看到的，XML 是一种专门在 World Wide Web 上传递信息的语言，就像 HTML（超文本标记语言）一样（自从 Web 出现以来，HTML 已经成为了创建 Web 页的标准语言）。

1.3 XML 历史

XML 是从 1996 年开始有其雏形，并向 W3C（全球信息网联盟）提案，而在 1998 二月发布为 W3C 的标准（XML1.0）。XML 的前身是 SGML（The Standard Generalized Markup Language），是自 IBM 从 60 年代就开始发展的 GML（Generalized Markup Language）标准化后的名称。

GML 的重要概念：

文件中能够明确的将标示与内容区隔;所有文件的标签使用方法均一致。

1978 年，ANSI 将 GML 加以整理规范，发布成为 SGML，1985 年起为 ISO 所采用（ISO 8879），并且被广泛地运用在各种大型的文件计划中，但是 SGML 是一种非常严谨的文件描述法，导致过于庞大复杂（标准手册就有 500 多页），难以理解和学习，进而影响其推广与应用。

于是，人们对 SGML 进行了简化衍生出 HTML。HTML 简单，在初期没有任何定义文档外观的相关方法，仅用来在浏览器里显示网页文件。而后，随着因特网的发展，人们为了控制其文件样式，扩充了描述如何显现数据的卷标。在 Netscape 与 Microsoft 之间的浏览器大战后，HTML 标准权威性遭受重大的考验，所幸，到了 HTML4.0 时，W3C 又恢复了其地位。

同时 W3C 意识到 HTML 的原罪：

- 不能解决所有解释数据的问题
- 像是影音文件或化学公式、音乐符号等其它型态的内容。
- 效能问题 - 需要下载整份文件，才能开始对文件做搜寻的动作。
- 扩充性、弹性、易读性均不佳。

为了解决以上问题，专家们使用 SGML 精简制作，并依照 HTML 的发展经验，产生出一套使用上规则严谨，但是简单的描述数据语言：XML。 XML 是在一个这样的背景下诞生的—是不是能有一个更中立的方式，让消费端自行决定要如何消化、呈现从服务端所提供的信息？

而 XML 目的即在于提供一个对信息能够做精准描述的机制，藉以弥补 HTML 太过于表现导向的特质。

标记语言家谱表如下图所示：

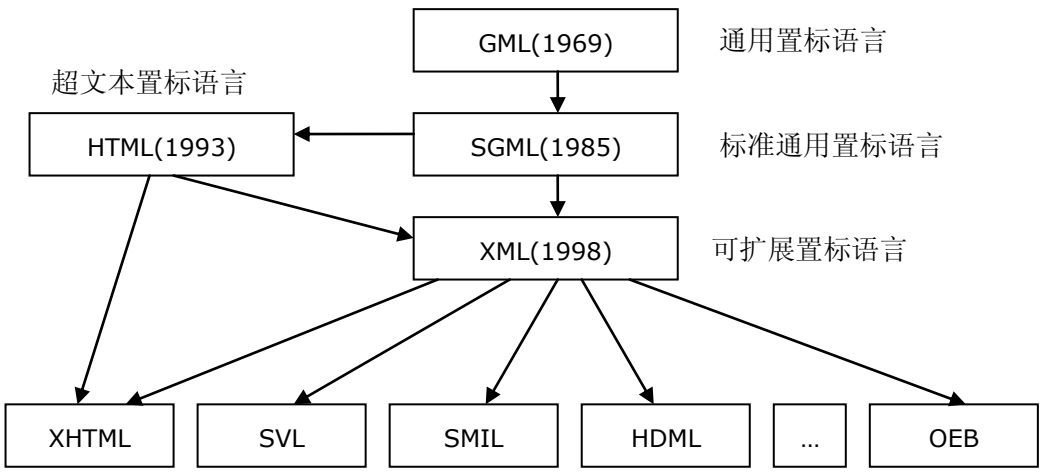


图 1-2

可能很多时候我们总是在困扰到底 SGML，XML,HTML 还有 XHMTL 他们之间的关系是怎么样的呢？

这张"标记语言家谱表图"给了我们答案 GML 自从 1969 年发明标记语言第一次在计算机中出现，大量的用在文件描述中，一直到 1985 年有国际标准化组织，做成行业标准对外发布，SGML 应用的领域主要是文档的描述，由于 SGML 的语法结构过于复杂，不便使用。于是在 1993 推出了一个基于 SGML 语言的一个网页应用 HTML。HTML 是 SGML 的语言的具体应用。用面向对象的概念来理解 SGML 如果是类，那 HTML 就是类的实例。虽然 HTML 的出现解决了文档描述的问题，但是对于标记语言的其他应用来说使用 SGML 还是很复杂，这样会影响到标记语言的发展，于是在 1998 国际标准化组织对 SGML 进行简化推出了 XML 可扩展标记语言。SGML 和 XML 的关系类似于父类和子类的关系，XML 继承了 SGML 的在标记语言方面的特性，简化了复杂性。之后基于 XML 的应用大量涌现 XHTML 可扩展的超文本标记语言，SVG 可缩放矢量图形语言，SMIL 同步多媒体综合语言，HDML 手持设备标志语言，XAML 可扩展应用程序标记语言。MXML 是 Flex 中布局用户界面组件的一种 XML 语言。

笔记区

1.4 XML 的应用

在很多时候我遇到过许多人，他们还不明白为什么要使用 XML 也不知道如何在他们的应用中使用 XML。如果你还不清楚 XML 到底有什么好处的话，你并不是唯一的人。

我决定把与人们和媒体关于 XML 话题的交谈整理成文，列出 XML 在应用中的五个最令人喜爱的用法。尽管这些并不能包含 XML 的所有潜在应用，至少是些最重要的领域。

1. 数据交换

用 XML 在应用程序和公司之间作数据交换已不是什么秘密了，毫无疑问应被列为第一位。那么为什么 XML 在这个领域里的地位这么重要呢?原因就是 XML 使用元素和属性来描述数据。在数据传送过程中，XML 始终保留了诸如父/子关系这样的数据结构。几个应用程序可以共享和解析同一个 XML 文件，不必使用传统的字符串解析或拆解过程。

相反，普通文件不对每个数据段做描述(除了在头文件中)，也不保留数据关系结构。使用 XML 做数据交换可以使应用程序更具有弹性，因为可以用位置(与普通文件一样)或用元素名(从数据库)来存取 XML 数据。

2、Web 服务

Web 服务是最令人激动的革命之一，它让使用不同系统和不同编程语言的人们能够相互交流和分享数据。其原理在于 Web 服务器用 XML 在系统之间交换数据。交换数据通常用 XML 标记，能使协议取得规范一致，比如在简单对象处理协议(Simple Object Access Protocol, SOAP)平台上。

SOAP 可以在用不同编程语言构造的对象之间传递消息。这意味着一个 C#对象能够与一个 Java 对象进行通讯。这种通讯甚至可以发生在运行于不同操作系统上的对象之间。DCOM, CORBA 或 Java RMI 只能在紧密耦合的对象之间传递消息，SOAP 则可在松耦合对象之间传递消息。

3、内容管理

XML 只用元素和属性来描述数据，而不提供数据的显示方法。这样，XML 就提供了一个优秀的方法来标记独立于平台和语言的内容。

使用象 XSLT 这样的语言能够轻易地将 XML 文件转换成各种格式文件，比如 HTML, WML, PDF, flat file, EDI, 等等。XML 具有的能够运行于不同系统平台之间和转换成不同格式目标文件的能力使得它成为内容管理应用系统中的优秀选择。

4、Web 集成

现在有越来越多的设备也支持 XML 了。使得 Web 开发商可以在个人电子助理和浏览器之间用 XML 来传递数据。

为什么将 XML 文本直接送进这样的设备去呢?这样作的目的是让用户更多地自己掌握数据显示方式，更能体验到实践的快乐。常规的客户/服务(C/S)方式为了获得数据排序或更换显示格式，必须向服务器发出申请;而 XML 则可以直接处理数据，不必经过向服务器申请查询-返回结果这样的双向"旅程"，同时在设备也不需要配制数据库。

甚至还可以对设备上的 XML 文件进行修改并将结果返回给服务器。想像一下，一台具有互

联网功能并支持 XML 的电冰箱将会给市场带来多么大的冲击吧。你从此不必早起去取牛奶了!

5、配置

许多应用都将配置数据存储在各种文件里，比如.INI 文件。虽然这样的文件格式已经使用多年并一直很好用，但是 XML 还是以更为优秀的方式为应用程序标记配制数据。使用.NET 里的类，如 XmlDocument 和 XmlTextReader，将配制数据标记为 XML 格式，能使其更具可读性，并能方便地集成到应用系统中去。使用 XML 配制文件的应用程序能够方便地处理所需数据，不用象其他应用那样要经过重新编译才能修改和维护应用系统。

6、图形图像

SVG 是 XML 来描述二维图形的语言。SVG 可以构造 3 种类型的图形对象：矢量图形、位图图象和文字。图形对象可被组化、样式化、变形和重组，包括图象嵌套、变形处理、剪辑路径、Alpha 蒙板、滤镜特效和模板对象。

7、编程语言

毫无疑问，RIA（Rich Internet Applications，富网络应用程序）技术是下一代网络应用的主流开发技术。而 XML 技术的应用，使得 RIA 技术的可用性和易用性又得到了极大的提高。在可以预见的未来，我们可以期待 RIA 和 XML 技术的更完美结合，以及它们的结合所带来的巨大影响力。

传统网络程序的开发是基于页面的、服务器端数据传递的模式，把网络程序的表示层建立于 HTML 页面之上，而 HTML 是适合于文本的传统的基于 HTML 的应用，他不能满足网络浏览者的更高的、全方位的体验要求了，这就是被 Macromedia 公司称之为的"体验问题"（"Experience Matters"），而富网络应用程序（Rich Internet Applications，缩写为 RIA）的出现也就是为了解决这个问题。

富网络应用程序是下一代网络应用程序，他是将桌面应用程序的交互的用户体验与传统的

笔记区

Web 应用的部署灵活性和成本分析结合起来的网络应用程序。富网络应用程序中的富客户技术通过提供可以承载已编译客户端应用程序（以文件形式，用 HTTP 传递）的运行环境，客户端应用程序使用异步客户/服务器架构连接现有的后端应用服务器，这是一种安全、可升级、具有良好适应性的新的面向服务模型，这种模型由采用的 Web 服务所驱动。结合了声音、视频和实时对话的综合通信技术使富网络应用程序（RIA）具有前所未有的网上用户体验。

“富”的概念包含两方面，分别是数据模型的丰富和用户界面的丰富。数据中的“富”意思是用户界面可以显示和操作更为复杂的嵌入在客户端的数据模型，它可以操作客户端的计算和非同步的发送接收数据。这种模式相对于传统的 HTML 页面的优点是程序运行于客户端并且程序更多的是和用户进行交互同时更少的和服务器进行交互。平衡客户端和服务器端的复杂的数据模型可以让你有更大的空间去创建更高效和更具有交互性的网络应用程序。“富”同样也描述了全面提升的用户界面，HTML 只给用户提供了非常有限的界面控制元素，而富网络应用程序（RIA）的用户界面提供了灵活多样的界面控制元素，这些控制元素可以很好的与数据模型相结合。传统的网络模型使用线性的设计，提供给用户一些选择然后用户发送选择结果给服务器，这种单一的模式不符合应用程序的灵活交互的要求和用户的意愿。频繁的服务器请求和页面刷新有很多的缺点包括页面打开缓慢和降低网络带宽。如果采用富客户界面，可以从以前的服务器响应影响整个界面，转移到只有收到请求的应用程序部分才会做出相应的变化。这本质上意味着界面被分解成许多独立模块，这些模块都会对收到的信息做出相应的反应，有些会和服务器端进行交互，有些是这些模块之间的通信。

流行的 RIA 开发技术术主要有 Macromedia 公司的 Flash/Flex、Microsoft 公司的 Avalon 技术、Mozilla 的 XUL 技术等。

Flash 是一个已经成熟的商业产品，它可以在 Web 网页中引入交互式的图形界面。最近经过升级后，新版本包含了建立窗体风格的应用程序的功能。尽管 Flash 作为一个在 Web 上最广泛部署的前端技术还有争议（取决于所选用的 Flash Player 版本），但据称已经有 98%以上的桌面系统都支持 Falsh。由于 Flash 具有强大的和可视化的动画创建功能（与之相反，其他技术则要求进行低级的图形编码），所以图形设计人员使用起来十分得心应手。Flash 采用的脚本语言是 ActionScript--ECMAScript 1.5 的一个变种，该脚本语言又被称为 JavaScript。Flex 产品对 Flash 增加了一个 XML 描述语言，即 MXML，使其可以编译用户界面，并且能够用 Flash Player 来随时进行描述。Flex 使得传统的开发机构能更好地了解和使用 Flash。Flex 和 Flash 的最大缺点在于对 XML 和 Web 服务等标准的支持很有限，而且作为应用开发工具的环境还不大成熟。Flex 和 Flash 的优点在于它可以很容易地用来创建复

杂的动画式显示，以及可以使用第三方附件。

Avalon 其实是 WinFX 三大支柱技术之一的表示子系统，而 WinFX 则是.NET 2.0 Framework 的延伸。Avalon 与 GDI 的区别在于，前者使用描述性模型来描述各种图形实体：窗口、网页、布局面板、向量图形、可重用控件、动画、3D 对象和景物等，而后者采用过程化的方式。Avalon 应用程序的图形输出与 GDI 不同，不是按照过程化指令顺序执行的，而是用不同层次的对象及其属性提供的。Avalon 中所有图形场景都是由 Avalon 的类模型中不同层次的对象——即对象树——构成的。本质上说，用 GUI 元素创建对象树，就可以开发出 Avalon 应用程序的 UI（用户界面）。这时，对象树常被称为 UI 树。不过创建 UI 树只是开发图形界面的一种方式，此外还可以用 XAML。

XUL（念作"zool"）是一个基于 XML 的用户界面语言，它来自于 Mozilla 的开放源码项目。它可用于建立窗体应用程序，这些应用程序不但可以在 Mozilla 浏览器上运行，而且也可以运行在其他描述引擎上，如 Zulu（一个 Flash MX 组件）和 Thinleys（一个 Java 实现）。XUL 描述引擎都非常小（100KB 以下），它可以使用 XML 数据也可以生成 XML 数据。同 Java 的情况一样，XUL 也有一个非常大的用户团体，这个团体有大量的开放源代码工具，如 Theodore ThinletEditor，一个能够以图形化方式布局用户界面，且可以生成相应 XUL 的 Java 应用程序。XUL 的一个主要缺点在于它目前还没有获得一个主要商业实体的支持。XUL 最大的优点在于它与 Gecko 引擎的集成（打开了通向大量 Web 标准的大门），以及与大多数其他 XML 用户界面描述语言相比它是一种非常具有表达力和简洁的语言。

以上这些 RIA 技术的共同特点便是都可以采用某种 XML 来描述 UI 的设计，表示数据，甚至进行数据通信。由此，我们可以看出 XML 技术对于 RIA 的重要性。

XML 在 RIA 中的应用：

XML（eXtendable Markup Lanuage）从其诞生起，就以其强大的功能成为人们关注的焦点，进而被广泛应用于许多领域。XML 的诸多先进性令其在产生后迅速得到发展。包括 IBM、微软在内的诸多国际顶级 IT 企业、著名研究机构和国际标准化组织纷纷积极参与到基于 XML 的数据标准的制定和相关软件研发等工作中，几乎每个专业 XML 标准的制订都有该领域在全球占据技术领导地位的企业或权威机构参与。

正因为 XML 技术的强大，RIA 应用领域也逐渐采用 XML 作为主要或辅助技术。XML 在 RIA 方面的应用包括以下几种：

1、 数据描述和交换标准

XML 数据标准通常是通过词汇表的形式存在的，XML 数据的描述，是元素及其属性、以

笔记区

及你所指定的文档结构的规范。作为信息交换的媒介，它经常是与人类在某种领域（例如商业、化学、法律、音乐）的活动息息相关的。XML 词汇表的高效性也正是 XML 应用成功的关键因素之一。

2、 图形化用户界面描述和构造

而仅仅有 XML 数据是不够的，必须使得信息消费者通过相应的应用程序所提供的图形化用户界面对数据进行操作才能够发挥数据存在的意义。

图形用户界面（GUI）工具包作为高级编程语言的核心技术之一，能够充分地体现一门高级语言的强大功能和先进的理念。Java 作为一门最受全世界软件开发者们欢迎的高级编程语言，其 GUI 工具包的丰富多样，一直被人们所津津乐道。不过，面对众多的选择，也有无法解决的难题。

工具包实现之间的差异导致了互相之间的不兼容性，解决之道自然是总结和概括这些工具包之间的异同，并抽象出一种跟实现无关的图形用户界面描述方法。正因为抽象描述方法有如此的优点，很早开始，就有研究者着手于抽象描述方法的研究。

但是，一直到 XML 技术的广泛使用，才使得研究者们认识到 XML 正是用来抽象描述图形用户界面的最佳载体。上文所提到的 MXML，XAML 以及 XUL 正是这方面研究的成果。

当然，MXML 等语言并不仅仅能描述 GUI 的构成，还可以定义后台的数据以及界面元素与数据之间的绑定关系。这就为简化 RIA 的开发提供了可能——开发者可以想像使用高级编程语言一样编写文本格式的 XML 脚本，RIA 引擎在运行时自动将 XML 脚本 GUI 以及相应的逻辑。

3、 数据表现样式控制

XML 技术不仅通过标准化和用户自定义的词汇表使得内容数据的描述能够得到统一的处理，更重要的是这些技术与 XSLT 的共同控制为重复出现的数据的排版和显示带来了操作上的极大便利。

XML 格式的内容数据虽然便于在网络和计算机之间交换和处理，但是对于人来说，仍然是难以阅读和理解的，特别是一些大型的工业用数据集。在这方面，纯 XML 可能还比不上 HTML，毕竟通过浏览器 HTML 能提供丰富多样和接近人类阅读习惯的数据表现手段。其实，稍稍了解 HTML 的人都知道，HTML 其实可以看作 XML 技术的一个非常特别的特例。因此，基于标准的 XML 规范，W3C 组织也提出了一系列用于控制数据表现样式的技术：XSL-FO 和 XSLT 等。

4、 应用流程控制

在商业领域，把一个商务事件的处理流程过程称为"商业流程"；在应用软件领域，把完成一件事情的过程称为"应用流程"，在电子商务日益发展的现在，这两者已经几乎可以划上等号。BPM 是 Business Process Management 的简称，我们通常所说的 BPM 事实上指的是商业过程管理系统，是一个软件组件，它以商业过程的描述为输入，监控整个商业过程的执行过程，同时安排工作，调用其他应用程序。在 BPM 系统中，其核心内容就是商业过程的描述和建模。在这方面，XML 技术也拥有其不可替代的优势。

1.5 XML 技术架构概要介绍

XML 是目前很普遍的一种数据交换技术，当然 XML 的应用并不仅限于在数据交换领域，其应用范围在不断的扩展。随着 XML 越来越多领域的应用及推广，XML 的体系架构越变得越来越庞大和复杂。

在记忆中，XML 刚推出时，仅是为了解决 HTML 结构松散导致的数据结构混乱的问题。后来发现 XML 有很好的数据描述和结构约束特性，因此便开始在数据交互领域使用。后来 XML 在许多需要这些特性的领域也开始流行起来，这便是 XML 技术的现状。

XML 本身的语法、结构和概念都不算太复杂。理解 HTML 的人，用几十分钟便可以掌握 XML 的结构及 XML 的描述方式。让人感到为难的是 XML 的相关技术和其扩展技术。以下是 XML 相关技术的框架图：

笔记区

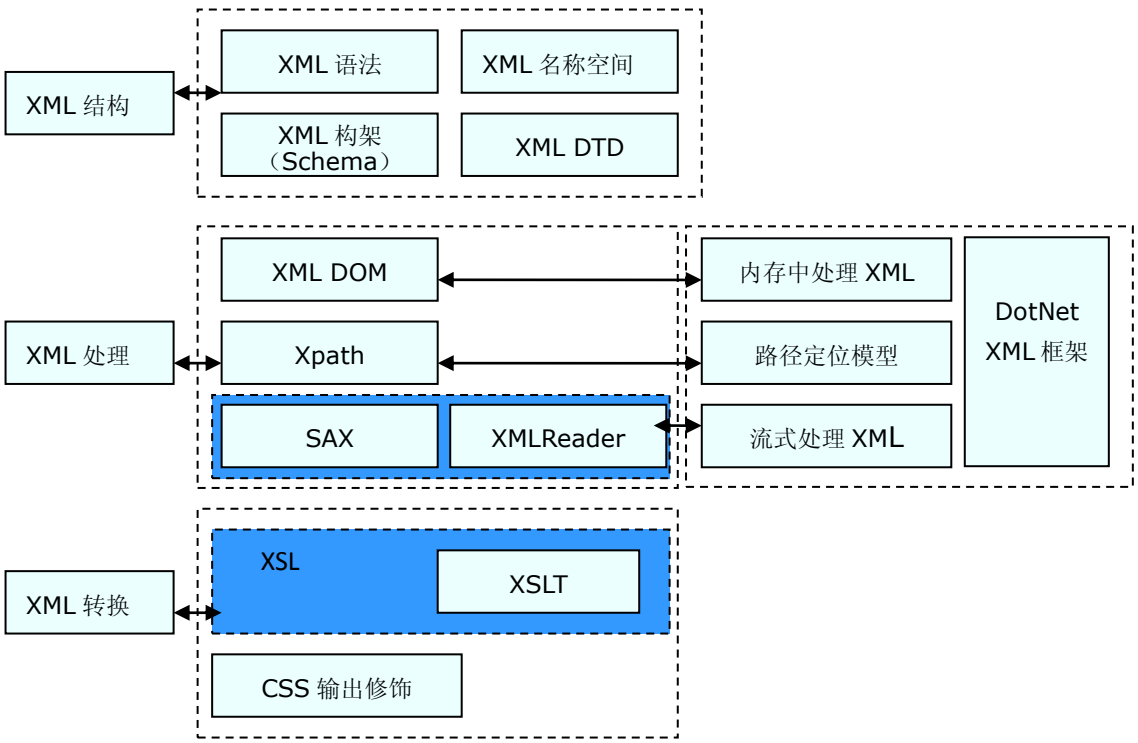


图 1-3

XML 结构：包括 **XML 语法**和 **XML 基本概念**，是 XML 最基本的东西，也就是描述 XML 是什么，怎么写，有什么特征。**DTD** 和 **XML Schema** 是 XML 的结构描述技术，XML 本身是一个结构性的描述语言，但是其并没有描述和约束数据结构的技术，因此便有 DTD 和 Schema 对其补充。简单的说，XML 是包装数据的，而 DTD 和 Schema 是描述数据和数据结构的。另外 Schema 比 DTD 有这更完整的描述特性，当然 Schema 也并不是完美的。

XML 处理：这部分的技术架构及关系比较复杂。简单的说它包括各种领域下对 XML 的操作技术。比较典型有 **XML DOM**,**XPath**,**SAX**,**XMLReader** 等。XML DOM 和 HTML 的 DOM 类似，就是通过对象模型来描述 XML 文档的结构，是一种把 XML 文档结构映射成对象结构的技术。XPath 是 XML 文档查找技术，通过使用 XPath 特定的描述语法，能快速定位 XML 文档内容。SAX 和 XMLReader 都是使用流式操作 XML 文档的技术。上面的技术在 DotNet 的 FrameWork 内都提供支持。

XML 转换：包括 **XSL** 和 **CSS**，它们的作用是把 XML 文档转换成其它形式。其它形式有很多，比如显示形式。显示形式简单地说，就是使用转换技术把一份 XML 文档在浏览器或者其它浏览工具中按照特定格式显示出来。XSL 是 XML 转换技术的一个体系，其中的 XSLT

是用于显示转换的。CSS 仅仅用于 XML 文档的显示转换。

XML 和 XML 扩展技术的大致体系结构就是这样。这里仅仅是从一个大体上进行介绍。其实 XML 本身并不难，复杂的是其扩展和外延技术。只要弄清楚 XML 的扩展技术，以及在各领域起到的作用，再结合 XML 的数据描述性去进行理解，那么就能很好的掌握它。

本书围绕这图中相关技术点讲解,你准备好了吗？跟随本书进入 XML 的精彩世界!

总结

本章我们了解到了 XML(eXtensible Markup Language)即可扩展标记语言，它与 HTML 一样，都是 SGML(Standard Generalized Markup Language,标准通用标记语言)：

- XML 是 EXtensible Markup Language 的缩写
- XML 是一种类似于 HTML 的标记语言
- XML 是用来描述数据的
- XML 的标记不是在 XML 中预定义的，你必须定义自己的标记
- XML 使用文档类型定义（DTD）或者模式（Schema）来描述数据
- XML 使用 DTD 或者 Schema 后就是自描述的语言

使用 XML 几种场合

1、 数据交换

2、 Web 服务

3、 内容管理

4、 Web 集成

5、 配置信息

6、 图形图像

7、 编程语言

XML 在 RIA（富互联网应用程序）中的几个应用方向

1、 数据描述和交换标准

2、 图形化用户界面描述和构造

3、 数据表现样式控制

笔记区

4、 应用流程控制

XML 中常见技术

1. XML 结构包括

   a) XML 语法、XML 命名空间、XML DTD 、XML Schema

2. XML 处理

   a) Dom、XPath

3. XML 转换

   a) XSLT、CSS

   b)

问与答

1. 用你自己的话来说明为什么需要使用 XML
2. XML 与 HTML 的区别是什么
3. XML 的使用场合
4.

作业:

1. XML 的全称是:
2. XML 主要是用来_____数据,HTML 主要是用来_____数据

第二次课 XML 基本概念

学习内容	难度
我的第一个 XML 文件	★
※XML 文档结构	★★★
XML 文档类型	★★

注：带※为重点学习内容

笔记区

2.1 我的第一个 XML 文件

通过第一次课的学习，相信大家都迫不及待想动手去写 XML 文档，那就让我们一起来写我们的第一个 XML 文件吧。

Code 2-1

HelloXML.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<Welcome>Hello,XML</Welcome>
```

在浏览器中显示效果如图 2-1 所示：

图 2-1

从这个案例中我们可以看到 XML 的一些特点：

XML 文件第一行文档声明

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

和 HTML 一样是标记语言<Welcome>,标记中有数据内容。

```
<Welcome>Hello,XML</Welcome>
```

名师提点

我们在这里看到的这两个特性仅仅表现出了 XML 的一小部分，在接下来的内容中会详细讲解其他部分。

2.2 XML 文档结构

HelloXML.xml 文档仅仅是一个最简化的 XML 文档，XML 文档的标准结构到底是怎样的呢？下图是一个比较典型的 XML 文档结构,XML 文档包括以下几部分：XML 声明、文档类型定义、注释、根元素、属性、元素文本、CDDATA。

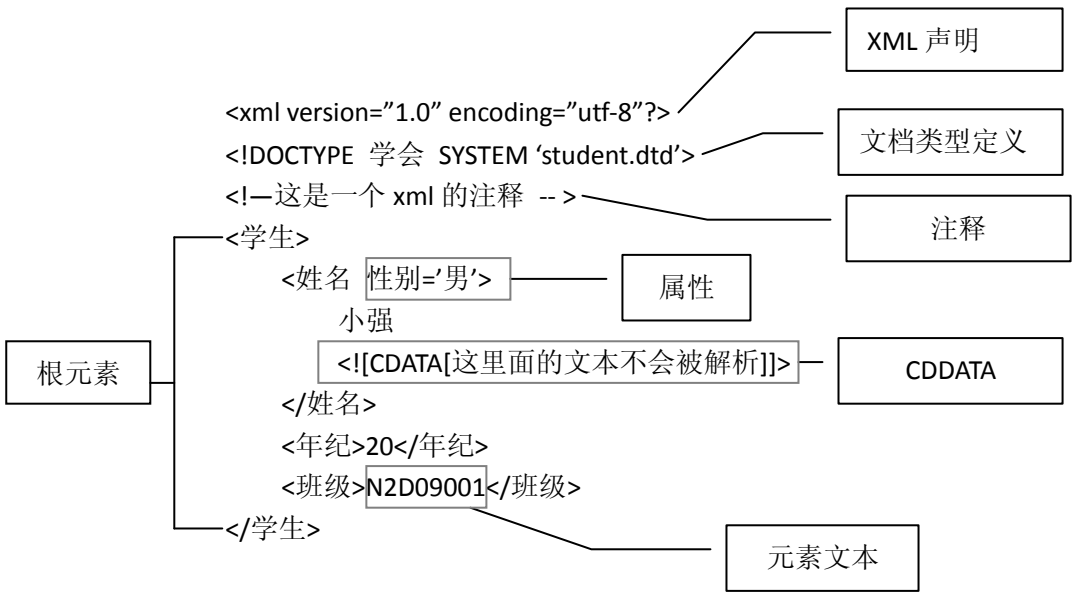


图 2-2

XML 文档的序言部分一般包含出现在文档或者文档根元素开始标记之前的信息。它包含应用与整个文档的信息，如：字符编码,样式表引用，另外,还包含 XML 声明,注释,处理指令，但是所有这些都是可选的.

1、 XML 声明

XML 文档总是以一个 XML 声明开始的，其格式如下：

<?xml 版本信息 [编码信息] [文档独立性信息]?>

需要注意的是，XML 声明总是在 XML 文档的最前面，其前面不能出现任何字符。下面是几种 XML 申明的形式：

笔记区

Code 2-2

```
<?xml version="1.0"?>
<!--版本信息声明-->
<?xml version="1.0" encoding="ISO-8859-1"?>
<!--编码信息声明.默认值是"UTF-8"-->
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!--文档独立性信息声明,如果文档依赖于外部文档，则需要将 standalone 属性设置为
"no"-->
```

2、 文档类型定义 DTD

DTD（Document Type Definition），文档类型定义。

XML 最为强大的功能是它允许你创建自己的标签名称。但是对于任何既有的应用程序来说，任何类型的标签以任意顺序出现并没有实际意义。如果正在编写样式表或应用程序来使得 XML 确实具有意义，那么就必须对次序和标签的嵌套指定一些约束。DTD 是可以用来标识约束的一种方式。

首先提一个问题，如果你的一个 XML 文档给别人使用，别人怎么才能正确地定义 XML 文档中的元素、属性呢？就好像我们用 ASP.NET 时，我们需要配置 web.config,但是怎么保证我们的配置是正确的？答案就是可以使用相对的格式化的文档来验证。这种格式化的文档有两种,一种是 DTD,一种是 SCHEMA.

一个遵循 XML 语法规则，并遵守相应 DTD 文件约束的 XML 文档称为有效的 XML 文档。

XML 从 SGML 继承了用于定义语法规则的 DTD 机制，DTD 文件本身是不需要遵循 XML 规则的。大部分的 XML 应用都是使用 DTD 来定义的（还有一部分是用 XML Schema 来定义的，XML Schema 在其专题中学习）。HTML 就有一个标准的 DTD 文件，所以其组织结构和所有的标签都是固定的，DTD 文件本身也是一个文本文件，通常用".DTD"为其扩展名。

文档类型声明的作用就是指出 XML 文档所用的 DTD，文档类型声明有两种形式。

一种是在 XML 文档中给出 DTD 内部 DTD。

Code 2-3

```
<?xml version="1.0" encoding="gb2312" ?>
<!DOCTYPE note [
  <!ELEMENT note (to,from,head,body)>
  <!ELEMENT to (#PCDATA)>
  <!ELEMENT from (#PCDATA)>
  <!ELEMENT head (#PCDATA)>
  <!ELEMENT body (#PCDATA)>
]>
<note>
  <to>aa@qq.com</to>
  <from>bb@163.com</from>
  <head>下午有事吗？</head>
  <body>今天下午要是没事情，就一起去游泳</body>
</note>
```

一种是外部 DTD，引用 XML 文件外面单独的 DTD 文件。

Code 2-4

```
<?xml version="1.0" encoding="gb2312" ?>
<!DOCTYPE note SYSTEM 'mail.dtd'>
<note>
  <to>aa@qq.com</to>
  <from>bb@163.com</from>
  <head>下午有事吗？</head>
  <body>今天下午要是没事情，就一起去打球.</body>
</note>
```

对应 DTD 文件 mail.dtd 内容为：

Code 2-5

```
<?xml version="1.0" encoding="utf-8" ?>
<!ELEMENT note (to,from,head,body)>
<!ELEMENT to (#PCDATA)>
```

笔记区

```
<!ELEMENT from (#PCDATA)>
<!ELEMENT head (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

关于 DTD 的详细内容将在后面详细讲解。

3、 根元素

根元素是必须的,它是 XML 文档中序言和 DTD 部分后面的第一个元素,它可以包含属性,子元素,注释等.前面的例子中的<note>就是一个文档根的例子.在 XML 文档中只能有且仅有一个根元素。

4、 元素

元素的结构：

在 XML 中，元素由开始标签、元素内容和结束标签构成，对于空元素，由空元素标签构成。

图 2-3

元素的四种形式：

1、空元素；　　2、带有属性的空元素；

3、带有内容的元素；　　4、带有内容和属性的元素。

元素的命名规则：

每一个元素都有一个用名字标识的类型，同时它可以由一个属性说明集，每一个属性说明有一个名字和一个值。在给元素命名的时候要注意，以”xml”或其他任何匹配 (‘X’|’x’) (‘M’|’m’) (‘L’|’l’))的字符串开头的名字，被保留用于 XML 规范的当前版本或者

后续版本的标准化。此外，在给元素命名时，还要遵循下列规范：

1、名字只能以字母、下划线（_）或者冒号（:）开头；

2、名字中可以包含字母、数字、下划线以及其他在 XML 标准中允许的字符；

3、名字中不能包含空格；

4、名字中尽可能不要使用冒号（:），因为冒号在名字空间中被用于分隔名字空间前缀和本地部分。

自己"发明"的 XML 元素还必须注意下面一些简单的规则：

任何的名字都可以使用，没有保留字（除了 XML），但是应该使元素的名字具有可读性，名字使用下划线是一个不错的选择。

例如：<first_name>, <last_name>.

尽量避免使用"-"，"."，因为有可能引起混乱。

只要你愿意元素的名字可以很长，但也不要太夸张了哦。命名应该遵循简单易读的原则，例如：<book_title>是一个不错的名字，而<the_title_of_the_book>则显得罗嗦了。

XML 文档往往都对应着数据表，我们应该尽量让数据库中的字段的命名和相应的 XML 文档中的命名保持一致，这样可以方便数据变换。

非英文/字符/字符串也可以作为 XML 元素的名字，例如<阿里西西><WEB 开发论坛>这都是完全合法的名字。但是有一些软件不能很好的支持这种命名，所以尽量使用英文字母来命名。

元素的内容：

如果在 XML 文档中元素内容使用类似"<"的字符，那么解析器将会出现错误，因为解析器会认为这是一个新元素的开始。所以不应该象下面那样书写代码：

Code 2-6

```
<message>if salary < 1000 then</message>
```

为了避免出现这种情况，必须将字符"<"转换成实体，象下面这样：

Code 2-7

```
<message>if salary &lt; 1000 then</message>
```

下面是五个在 XML 文档中预定义好的实体：

<	<	小于号

笔记区

>	>	大于号
&	&	和
'	'	单引号
"	"	双引号

图 2-4

实体必须以符号"&"开头，以符号";"结尾,只有"<" 字符和"&"字符对于 XML 来说是严格禁止使用的。剩下的都是合法的，为了减少出错，使用实体是一个好习惯。

5、 注释

注释用于对文档中的内容起到一个说明作用。

XML 的注释和 HTML 的注释类似，都是以<!--开始，以-->结束。位于<!--和-->之间的数据将被 XML 处理器忽略。

使用 XML 注释需要注意一下几点：

1）注释不能出现在 XML 声明之前，XML 声明必须是文档最前面的部分。下面的情况是不允许的：

Code 2-8

```
<!--Author:Jason Chen-->
<?xml version="1.0"?>
```

2）注释不能出现在标记中，下面的例子是非法的：

Code 2-9

```
<author<!—This document author-->>Jason Chen</author>
```

3）注释可以包围和隐藏标记，但是要注意，在注释掉标记以后，要保持文档的完整结构。

4）字符串"--"（双连字符）不能出现在注释中。

6、 处理指令

处理指令（Processing Instructions）允许文档中包含由应用程序来处理的指令。在 XML 文档中，有可能会包含一些非 XML 格式的数据，这些数据 XML 处理器无法处理，我们可以通过处理指令来通知其它应用程序来处理这些数据。

处理指令的语法和 XML 声明类似，以<？开始， 以？>结束。一个常见的使用样式表单

的处理指令如下所示：

Code 2-10

```
<？xml-stylesheet href="style.css" type="text/css"？>
```

在开始标记<？后的第一个字符"xml-stylesheet"叫做处理指令的目标，它必须标识要用到的应用程序，要注意的是对于其它的非 W3C 定义的处理指令不能以字符串"XML"和"xml"开头；其余的部分是传递给应用程序的字符数据。应用程序从处理指令中取得目标和数据，执行要求的动作。

处理指令的目标可以是要使用的程序的名字，或者是一个类似于 xml-stylesheet 这样的很多程序可以识别的通用标识符。不同的应用程序支持不同的处理指令，对于不认识的处理指令，大多数应用程序采取忽略的方式进行处理。

xml-stylesheet 处理指令总是放在 XML 声明之后，第一个元素之前。其它的处理指令可以放在除了标记的内部和 XML 声明之前的任何位置。

名师提点

虽然 XML 声明和处理指令的语法形式相似，但 XML 声明并不是处理指令，XML 处理程序对 XML 声明和处理指令采取的是不同的处理方式。

7、 文本内容 CDATA PCDATA

包含在 XML 文档中的文本数据可以作为字符数据（Character Data，CDATA），已解析的字符数据（Parsd Character Data，PCDATA）或两者结合来表示。出现在<![CDATA[和]]>标签之间的数据是 CDATA，而任何其他的数据则是 PCDATA。下面元素包含了 PCDATA：

Code 2-11

```
<title>非诚勿扰</title>
```

Code 2-12

```
<title><![CDATA[非诚勿扰]]> </title>
```

也可以是下面的元素包括以上两者：

Code 2-13

```
<title>非诚勿扰<![CDATA[冯小刚出品]]> </title>
```

当你想让 XML 文档中的一部分被解析器忽略而不进行处理时，CDATA 是非常有用的。

笔记区

你可以在<![CDATA[和]]>标签中放置任何内容而 XML 解析器不会解析。但是那些没有包括在<! [CDATA[和]]>标签中的数据必须符合 XML 的规则。一般 CDATA 部分会放置其他语言的代码，比如 VBScript，JavaScript，MXML（Flex 中使用）。

XML 解析器忽略 CDATA 而分析 PCDATA—也就是说将其解释成为标记语言。您也许想知道为什么是 XML 解析器要区分 CDATA 和 PCDATA。某些字符，特别是<,>和&，在 XML 中具有特殊的意义，所以，如果需要使用他们，那么就必须包括在 CDATA 中，例如，你想定义一个 range 元素，其值为”0<counter<100”,因为<是保留字符，所以不能按如下方式定义元素。

Code 2-14

```
  <range> 0 < x < 1000 </range>
```

但是可以这样定义

Code 2-15

```
  <range>
    <![CDATA[0 < x < 1000]]>
  </range></range>
```

CDATA 段中包含的都是纯字符数据，在字符数据可以出现的任何地方都可以使用 CDATA 段。

CDATA 段主要用于需要将整个文本解释为字符数据而不是标记的情况下。CDATA 段中的内容不被 XML 处理器分析，所以可以在其中包含任意的字符。

CDATA 对于 XML 文档中包括数学等式，程序清单甚至其他 XML 文档来说都非常有用。

CDATA 部件之间不能再包含 CDATA 部件（不能嵌套）。如果 CDATA 部件包含了字符"]]>" 或者"<![CDATA[" ，将很有可能出错哦。

同样要注意在字符串"]]>"之间没有空格或者换行符。

8、 属性

XML 允许你通过元素开始标签中包含属性来添加元素的附加信息。属性是名称/值的结构。下面 Movie 元素将 ID 编号作为属性而不是一个元素。

笔记区

Code 2-16

```
<?xml version="1.0" encoding="utf-8" ?>
<Movie ID="09001">
  <title>非诚勿扰</title>
  <price>70</price>
</Movie>
```

属性值必须包括在一对单引号或者双引号之间，并且可以包含空格和引号（属性值用双引号（"）或单引号（'）分隔（如果属性值中有'，用"分隔；有"，用'分隔）

9、 错误处理

XML 文档中发生的错误将导致 XML 程序停止。

在 HTML 中，HTML 文件可能包含很多错误，（比如一个元素有开始标记没有结束标记）这也是 HTML 浏览器体积之所以很大的一个原因，当他们发现错误的时候，他们有各自不同的方法来决定此 HTML 文件应该如何显示。在 XML 中决不会发生这种情况。

W3C 的 XML 规范声明：如果程序在处理 XML 文档中发现一个有效的错误，那么此程序应该终止。这就是 XML 软件相对于容易编写的原因。所有的 XML 文档都应该是协调一致的。

10、 注意：

在定义文档结构时，特别是对 XML 新手来说，有时并不清楚指定的项是应该定义为属性还是元素。一般来说，属性应当用来定义外部数据，而元素则定义文档所需的数据。在上面的示例中，ID 一般不是定义为属性，因为 ID 为 Movie 的重要信息。

Code 2-17

```
<?xml version="1.0" encoding="utf-8" ?>
<Movie image=”09001.jpg”>
  <title>非诚勿扰</title>
  <price>70</price>
  <id>09001</id>
</Movie>
```

Image属性包含的附加信息，应用程序可以利用该信息以图片的形式来显示Movie信息。

笔记区

因为只有处理这个文档的软件才可能关心该图片，还应为 image 附属于 Movie 的定义，所以应该将 image 看成是属性而不是元素。

2.3 XML 文档的类型

XML 文档分为结构良好的文档和有效的文档两种.

1、 结构良好的 XML 文档

如果文档符合由 XML 规范定义的标准,那么该文档就被认为是格式良好的.一个格式良好的 XML 文档必需满足如下规定:

- 必须有 XML 声明语句
- 必须有且仅有一个根元素
- 标记大小写敏感
- 属性值用引号
- 标记成对
- 空标记关闭
- 元素正确嵌套
- 名称不能以数字开头
- 不能以 XML/xml/Xml/...开头
- 名称中不能含空格
- 名称中不能含冒号(注：冒号留给命名空间使用)

2、 有效的 XML 文档

首先，在考虑一个 XML 文档是否有效之前，必须考虑该文档是否是一个格式良好的文档,也就是说一个有效的 XML 文档它首先必须是一个结构良好的文档。格式良好的要求应该相当简单，但关键是让 XML 文档从格式良好到有效的 XML 文档却比较困难。为了达到有效，XML 文档必须经过验证。可以通过文档类型定义（Document Type Definition，DTD）或者 XML 模式定义（XML Schema Definition ，XSD）来对文档进行验证,DTD 和 Schema 将在接下来的章节中进行详细讲解。对于将要被验证的 XML 文档来说，他必须符合相应的 DTD 或者 XSD 模式的限制。

总结

笔记区

本章节我们了解到了，XML 的文档结构，XML 文档是如何声明的， PI 指令的用法，根元素的注意和要求，普通元素的定义和注意事项，属性的定义规则，文本类型 PCDATA 的意义，以及它与 CDATA 类型的区别，以及在 XML 中实体的意义等,这些知识点的学习都是为了确保大家去书写一个格式良好的 XML 文档。

问与答：

1. XML 文档有哪两种类型
2. 一个结构良好的 XML 文档必须满足哪些规定

作业：

1：下列标记在 html 中合法吗？

```
<书>
    <作者>张三</作者>
 </书>
```

2：下面什么标记是合法的？

a） <:TEST>

b) <TEST_XML>

c） <TEST XML>

3：指出下列代码是否错误，错误在哪里？

a）<BOOK>
<AUTHOR>abc……<AUTHOR>
</BOOK>

b）<BOOK>
<AUTHOR>
</BOOK>
</AUTHOR>

c）<BOOK>
<AUTHOR>abc……<AUTHOR>
</book>

笔记区

4：把下面数据库的表内容转化为 xml 文件

2005 年本地大学学生注册

学院	新生	毕业生	变动
Cedar 大学	110	103	+7
Elm 学院	223	214	+9
Maple 高等专科院校	197	120	+77

(**来源：**虚构数据，仅用作图表示例)

5：把下面数据库的表内容转化为 xml 文件，以属性方式表示。

项　　目	所需数目
图　　书	1
铅　　笔	3
笔　记　本	1
便　笺　簿	1

2部分

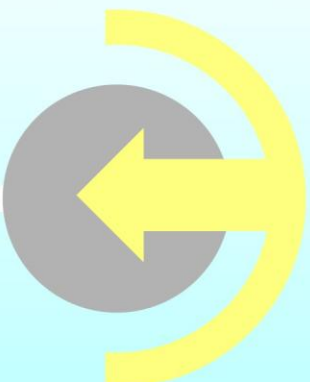
有效的XML文档

第三次课 DTD文档类型定义

第四次课 命名空间与schema构架

第五次课 shcema XML构架以及数据类型

通过第一部分的学习，我们能写一个结构良好的文档了，但我们写的这个文档是有效的吗？在第二部分的章节中主要包括的内容是DTD和Schema ，对于将要被验证的XML文档来说，它必须符合相应的DTD或者Schema 的限制。通过第二部分的学习，我们能使用DTD或者S chema的限制熟练写出一个结构良好又有效的XML文档。



第三次课 DTD 文档类型定义

学习内容	难度
为什么需要使用 DTD	★
※XML 文档如何使用 DTD	★★
※DTD 结构	★★★

注：带※为重点学习内容

笔记区

3. 1 为什么需要 DTD

对于一个格式良好的 XML 文档，我们只能保证这个文档的格式符合 XML 规范，但是元素与元素的关系，元素与属性的关系，属性的取值，我们就无法保证了。对于一个格式良好的文档，如果只是在有限的应用中使用，或者是用于存储和传输，那么它也能能够很好的满足我们的应用，但是如果要其它的用户了解你所写的 XML 文档，或者与其它应用进行数据的交换，那么就有必要提供一种机制，来保证我们写的 XML 文档和别人所写的 XML 文档在结构上是相同的，元素与元素之间的关系是正确的，属性的取值也是符合要求的，那么这种机制在 XML 规范中已经为我们提供了，那就是前一章中介绍过的文档类型声明中提到的 DTD。

DTD（Document Type Definition），文档类型定义。

在 XML 标准中，描述了如何创建 DTD，以及如何将它与根据它的规范所编写的 XML 文档相关联，并且还定义了 XML 处理器应该如何对 DTD 进行处理。有了 DTD 就可以检测 XML 文档的结构是否正确。DTD 为 XML 文档的编写者与处理者提供了共同遵循的原则，使得与文档相关的各种工作有了统一的标准。

3. 2 XML 文档如何使用 DTD

通过在 XML 文档中包含文档类型声明，来建立当前文档和 DTD 的关联。当进行有效性验证的 XML 处理器读到该声明时，它获取 DTD，并根据其中定义的规则对文档进行检验。文档类型声明必须位于 XML 声明之后，且在根元素（文档元素）之前。不过，在 XML 声明和文档类型声明之间可以插入注释和处理指令。我们可以直接在 XML 文档中定义 DTD，也可以通过 URI 引用外部的 DTD 文件，或者同时采用这两种方式。上一章中，已经学习过如何通过包含文档类型声明来建立与 DTD 的关联。现在来分析一下这两种包含方式。

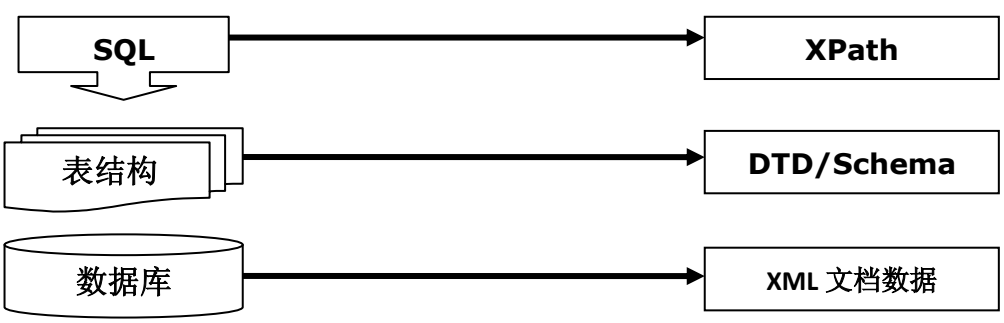


图 3-1

笔记区

上面这幅图中我们可以看到，左侧是我们常见的数据库的结构，数据部分是存储在数据库中，在数据库之上是表的结构，而我们平时是使用 SQL 结构化查询语言在按照表结构检索数据，存储数据。而 XML 的结构和数据库的比较类似，在图的右侧，我们一般用 XML 来存储数据，在其之上我们使用 DTD、Schema 来定义 XML 的文档结构，在其之上我们使用 XPath 技术在 XML 中检索数据。

在 XML 文档内部定义 DTD 的方式：

Code 3-1

```
<?xml version='1.0' encoding='gb2312'?>
<!DOCTYPE poem[          <----------------根元素的名称
<!ELEMENT poem (author,title,content) ><--------------子元素的名称及顺序
<!ELEMENT author (#PCDATA)><-----------------子元素的数据类型
<!ELEMENT title (#PCDATA)>
<!ELEMENT content (#PCDATA)>
]> <-----------------------结束标签
<poem>
    <author>王维</author>
    <title>鹿柴</title>
    <content>空山不见人,但闻人语声.
                返景入深林,复照青苔上.
    </content>
</poem>
```

名师提点

在 DTD 中，所有的关键字都是大写的。不过，在 DTD 中定义的元素和属性的大小写是可以任意制定的，但是要注意，因为 XML 文档是大小写相关的，所以一旦给一个元素命名，那么在整个文档中要使用相同的大小写。例如，Student 和 student 是不同的两个元素名。

文档类型声明由<!开始，后面紧跟一个关键字 DOCTYPE，然后是文档根元素的名字，接下来是标记声明块，标记声明块放在中括号[]之间，由一个或者多个标记声明构成，最后由>结束。

笔记区

在 XML 文档中定义 DTD，比较直观，修改也比较方便，而且不用担心 XML 处理器找不到 DTD，但是它也有一些缺点：

1、在文档中定义 DTD 会导致文档本身的长度增加，在传输数据时，即使不需要验证文档的有效性，这些声明也要一起传输。

2、如果多个 XML 文档要共用同一个 DTD，我们就需要在每一个文档中加入 DTD。

引进外部 DTD 方式：

Code 3-2

```
<? xml version='1.0' encoding='gb2312' ?>
<!DOCTYPE poem SYSTEM "ex2.dtd">
<poem>
  <author>王维</author>
  <title>鹿柴</title>
  <content>空山不见人,但闻人语声.返景入深林,复照青苔上.</content>
</poem>
```

外部 DTD

Code 3-3

```
<?xml version="1.0" encoding="gb2312"?>
<!ELEMENT poem (author,title,content)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT content (#PCDATA)>
```

内部 DTD 和引进外部 DTD 方式同时使用

Code 3-4

```
<? xml version='1.0' encoding='gb2312' ?>
<!DOCTYPE poem SYSTEM "ex3.dtd"[
  <!ELEMENT poem (author,title,content)>
  <!ELEMENT content (#PCDATA)>
]>
<poem>
  <author>王维</author>
```


```
  <title>鹿柴</title>
  <content>空山不见人,但闻人语声.返景入深林,复照青苔上.</content>
</poem>
```

外部 **DTD**

Code 3-5

```
<?xml version='1.0' encoding='gb2312'?>
<!ELEMENT author (#PCDATA)>
<!ELEMENT title (#PCDATA)>
```

在文档类型声明时，用关键字 SYSTEM 或 PUBLIC 来指出外部 DTD 文件的位置，使用 **SYSTEM** 关键字的声明语法如下：

<!DOCTYPE 根元素的名字 SYSTEM "外部 DTD 文件的 URI">

SYSTEM 关键字表示文档使用的是私有的 DTD 文件，“外部 DTD 文件的 URI”可以是相对 URI 或者绝对 URI。例如上面的例子使用的就是相对 URI：

Code 3-6

```
<!DOCTYPE poem SYSTEM "Student.dtd">
```

使用 **PUBLIC** 关键字的声明语法如下：

<!DOCTYPE 根元素的名字 PUBLIC "DTD 的名字" "外部 DTD 文件的 URI">

PUBLIC 关键字用于声明公共的 DTD，并且这个 DTD 还有一个名称，"DTD 的名称"也称为公共标识符(public identifier)。这个 DTD 可以存放在某个公共的地方，XML 处理程序会根据名称按照某种方式去检索 DTD，如果 XML 处理器不能根据名称检索到 DTD，就会使用"外部 DTD 文件的 URI"来查找该 DTD。例如 Java web 开发的 web.xml 中的 DTD 声明（版本不同会稍有不同，我们只关注它的结构）：

例如你在 Visual Studio2008 中创建一个普通.aspx 页面,在页面的头部就会有这样一段 XHTML 的 dtd 文档声明：

Code 3-7

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

DTD 名称与 XML 名称略有不同，他们只能包含 ASCII 字母和数字符号、空格、回车符、换行符和一些标点符号：-'()+,/:=?;!#@*$_% 。

笔记区

公共 DTD 名称要遵守一些约定。如果一项 DTD 是 ISO 标准，它的名称要以字符串"ISO"开始。如果是一个非 ISO 的标准组织批准的 DTD，它的名字以加号（＋）开始。如果不是标准组织批准的 DTD，它的名称以连字符（－）开始。这些开始字符或字符串后接双斜杠（//）和 DTD 所有者的名字（比如上面例子的 Sun Microsystems，Inc.），之后是另一个双斜杠和 DTD 描述的文档类型，接着又是一个双斜杠后接 ISO 639 语言标识符，如 EN 表示英语，ZH 表示中文。例如：

Code 3-8

```
<!DOCTYPE OrganizationChart PUBLIC "-//Jason Chen//DTD organization chart 1.0//ZH" "dtdTest.dtd">
```

在上一章我们提到，如果我们的文档不依赖于外部文档，在 XML 声明中，可以通过 standalone="yes"来声明这个文档是独立的文档。如果文档依赖于外部文档，可以通过 standalone="no"来声明。当我们使用外部 DTD 文件时，就需要将属性 standalone 的值设置为"no"。

在实际应用中，很少使用 standalone 属性，它的主要用途是作为 XML 处理器行业其他应用程序的标志，表示是否需要获取外部内容。如果文档依赖于外部文档，即使我们不使用 standalone 属性,XML 处理器也能很好地进行处理。

例中的定义关键字一定要大写，如 DOCTYPE、ELEMENT、#PCDATA，且元素名称与数据类型之间也要有空格。

3.3 DTD 的结构

DTD 的结构一般由以下三种声明构成：

元素类型声明、属性列表声明、实体声明

一个典型的文档类型定义文件会把所要创建的 XML 文档的元素结构、属性类型、实体引用等预先进行定义。

下面分别介绍这三种声明。

1、元素类型声明：

一个 DTD 不仅要告诉 XML 处理器它所对应的文档根元素，而且还要告知处理器，文档

的内容和结构，描述清楚文档结构中的每一个细节。元素类型声明不但说明了每一个文档中可能存在的元素，给出了元素的名称，而且给出了元素的具体类型。一个 XML 元素可以为空，也可以只包含字符数据，还可以有若干个子元素，而这些子元素同时又可以拥有它们的子元素。元素类型声明采用如下的语法格式：

<!ELEMENT 元素名称 元素内容说明>

元素内容说明可以指明五种可能的元素内容形式：#PCDATA、子元素、混合内容、EMPTY 和 ANY。下面详细说明每一种元素内容说明。

#PCDATA：关键字#PCDATA 说明元素可以包含任何字符数据，但是不能在其中包含任何子元素其它类型(组合)；

Empty：关键字说明该元素不能包含子元素和文本，但可以有属性－（空元素）；

Any：关键字说明该元素可以包含任何在 DTD 中定义的元素内容；

(子元素|子元素) ：表示组合类型。

#PCDATA 关键字用法:

定义 XML 元素名称为"人"类型为#PCDATA

Code 3-9

```
<!ELEMENT 人 (#PCDATA)>
```

XML 元素正确写法

Code 3-10

```
<人>小强</人>
```

XML 元素错误写法

Code 3-11

```
<人>
<姓名>小强</姓名>
</人>
```

在这个案例中 XML 元素名称要求为"人"，而其中的可以包含任何字符数据，但是不能在其中包含任何子元素其它类型(组合)，所以第二段 XML 代码元素编写错误。

ANY 关键字用法:

定义 XML 元素名称为"人"类型为 ANY

笔记区

Code 3-12

```
<!ELEMENT 人 ANY>
```

Code 3-13

```
<人>
<姓名>小强</姓名><姓名>小明</姓名>
</人>
```

当我们将 XML 元素类型定义为 ANY 时,将根元素设为 ANY 类型后，元素出现的次数和顺序不受限制。

Empty 关键字用法:

定义 XML 元素名称为"人"类型为 Empty 空元素.

Code 3-14

```
<!ELEMENT 人 ANY>
  <人 姓名="小强" 性别="男" 年龄="18"/>
```

Empty 标识表示元素内不能任何类型的数据。

组合类型的用法:

声明元素为组合类型，里面包括学生最少一个或多个，姓名 0 个或多个。详细关于+和*的意义在后面内容中会详细讲解。

Code 3-15

```
<!ELEMENT 银河学院 (学生+)>
  <!ELEMENT 学生 (姓名*,年龄)>
  <!ELEMENT 姓名 (#PCDATA)>
  <!ELEMENT 年龄 ANY>
```

Code 3-16

```
<银河学院>
  <学生>
    <姓名>张三</姓名>
    <年龄>18</年龄>
  </学生>
```


```
  <学生>
        <年龄>22</年龄>
  </学生>
</银河学院>
```

2、属性的定义

属性类型－ CDATA:

CDATA 类型表示其指定的属性值可以是任何字符（包括数字和中文）定义如下:

Code 3-17

```
<!ATTLIST 学生 编号 CDATA #REQUIRED>
下面是调用代码:
Code 3-17
  <学生 编号="1">
    <姓名>张三</姓名>
    <年龄>18</年龄>
  </学生>

  <学生 编号="2">
    <姓名>李四</姓名>
    <年龄>22</年龄>
  </学生>
```

属性类型－NMTOKEN/NMTOKENS:

某些情况下，你可能希望将属性值作为离散的记号，而不是文本。为此我们可以使用枚举，但是，假如我们希望值列表能够无限扩展呢？这就需要依靠 XML 中称为名称记号（name token）的类型。简称为 NMTOKEN。

- NMTOKENS 与 NMTOKEN 类似，包含多个由空格分隔的字符。

- **NMTOKEN** 是 CDATA 的一个子集，表示属性值必须是英文字母、数字、句号、破折号、下划线或冒号（可以是中文!）。

笔记区

- **NMTOKENS** 与 **NMTOKEN** 类似，包含多个由空格分隔的字符。

下面一个案例中定义了 title 的 author 属性为 NMTOKEN 类型，而在使用的时候却在 author 属性中写的"**#杜甫**"所以该属性赋值错误，在该属性中的值必须是英文字母、数字、句号、破折号、下划线或冒号（可以是中文！）

Code 3-19

```
<?xml version="1.0" encoding="gb2312" ?>
<!DOCTYPE poems[
<!ELEMENT poems (title,content)>
<!ELEMENT title (#PCDATA)>
<!ATTLIST title author NMTOKEN #REQUIRED>
<!ELEMENT content (#PCDATA)>
]>
<poems>
    <title author="#杜甫">八阵图</title>
    <content>
        功盖三分国,名成八阵图,
        江流石不转,遗恨失吞吴.
    </content>
</poems>
```

以下是正确写法：

Code 3-20

```
<?xml version="1.0" encoding="gb2312" ?>
<!DOCTYPE poems[
  <!ELEMENT poems (title,content)>
  <!ELEMENT title (#PCDATA)>
  <!ATTLIST title author NMTOKEN #REQUIRED>
  <!ELEMENT content (#PCDATA)>
]>
<poems>
  <title author="杜甫">八阵图</title>
```

```
  <content>
    功盖三分国,名成八阵图,
    江流石不转,遗恨失吞吴.
  </content>
</poems>
```

在下面案例中我们可以看到 NMTOKENS 和 NMTOKEN 的区别，author 属性定义为 NMTOKEN 的值，如果有两个作者,我们这样写 author="**杜甫 李白**"，两个作者中用空格分开,则该文档验证失败。

Code 3-21

```
<?xml version="1.0" encoding="gb2312" ?>
<!DOCTYPE 文章[
  <!ELEMENT 文章 (标题,内容)>
  <!ELEMENT 标题 (#PCDATA)>
  <!ATTLIST 标题 作者 NMTOKEN #REQUIRED>
  <!ELEMENT 内容 (#PCDATA)>
]>
<文章>
  <标题 作者="杜甫 李白">八阵图</标题>
  <内容>
    功盖三分国,名成八阵图,
    江流石不转,遗恨失吞吴.
  </内容>
</文章>
```

正确的写法如下：

Code 3-22

```
<?xml version="1.0" encoding="gb2312" ?>
<!DOCTYPE 文章[
  <!ELEMENT 文章 (标题,内容)>
  <!ELEMENT 标题 (#PCDATA)>
  <!ATTLIST 标题 作者 NMTOKENS #REQUIRED>
```

笔记区

```
  <!ELEMENT 内容 (#PCDATA)>
]>
<文章>
  <标题 作者="杜甫 李白">八阵图</标题>
  <内容>
    功盖三分国,名成八阵图,
    江流石不转,遗恨失吞吴.
  </内容>
</文章>
```

属性类型－ ID:

ID 属性类型表示该属性取值必须是唯一的。

Code 3-23

```
<!ELEMENT 公司职员 ANY>
<!ATTLIST 公司职员
    编号 ID #REQUIRED
    姓名 CDATA #REQUIRED
>
```

以下为错误写法，在元素公司职员中有属性编号类型为 ID，这个属性的值在文档中只能出现一次,且不能重复。

Code 3-24

```
<公司职员 编号="Z001" 姓名="张三"/>
<公司职员 编号="Z001" 姓名="李四"/>
```

正确的写法:

Code 3-25

```
<公司职员 编号="Z001" 姓名="张三"/>
<公司职员 编号="Z002" 姓名="李四"/>
```

属性类型－ IDREF/IDREFS:

- IDREF 属性的值指向文档中其它地方声明的 ID 类型的值。

- IDREFS 同 **IDREF**，但是可以具有由空格分开的多个引用。

在下面案例中定义了一个元素家庭其中包括 1 个或多个子元素人，而元素人是空元素，但他包括三个属性 relID 类型为 ID，parentID 类型为 IDREFS ，name 类型 CDATA。

Code 3-26

```
<!ELEMENT 家庭 (人+)>
<!ELEMENT 人 EMPTY>
<!ATTLIST 人
    relID ID #REQUIRED
    parentID IDREFS #IMPLIED
    name CDATA #REQUIRED
>
```

XML 文档写法如下：

Code 3-27

```
<家庭>
  <人 relID="P_1" name="爸爸"/>
  <人 relID="P_2" name="妈妈"/>
  <人 relID="P_3" parentID="P_1 P_2" name="儿子">
</家庭>
```

人在文档中有多个元素，爸爸、妈妈、儿子为 ID 分别定义了 P_1，P_2，P_3 三个不同的 ID 值，而在属性值 name 为儿子的这个元素中有个，parentID 属性指向的是爸爸和妈妈的 refID 中的内容，并以空格隔开多个名称。

属性类型－Enumerated:

事先定义好一些值，属性的值必须在所列出的值的范围内。

Code 3-28

```
<!ATTLIST person 婚姻状态 (single|married|divorced|widowed) #IMPLIED>

<!ATTLIST persion 性别 (男|女) #REQUIRED>
```

笔记区

属性类型-ENTITY/ENTITIES:

- **ENTITY** 类型的属性的值必须对应一个在 DTD 文档内声明的实体。
- **ENTITIES** 类型的属性的值与 ENITITIES 类似，不同的是可以包含多个实体。

Code 3-29

```
<?xml version="1.0" encoding="gb2312"?>
<!DOCTYPE library[
  <!ELEMENT library (number,img)>
  <!ELEMENT number (#PCDATA)>
  <!ELEMENT img EMPTY>
  <!ATTLIST img src ENTITIES #REQUIRED>
]>
<library>
  <number> A001 </number>
  <img src="&lt;&gt;"/>
</library>
```

属性类型－NOTATION:

- 属性的值必须引用已在 DTD 文档其它地方声明过的某个注解的名称。

Code 3-30

```
<!NOTATION mpeg SYSTEM "mplayer.exe">
<!ATTLIST media player NOTATION   mpeg #REQUIRED>
```

- 属性的值必须匹配 NOTATION 名称列表中的某个名称。

Code 3-31

```
<!NOTATION mpeg SYSTEM "mplayer.exe">
<!NOTATION jpeg SYSTEM "netscape.exe">
<!ATTLIST media player NOTATION  ( mpeg | jpeg ) #REQUIRED>
```

定义符号 NOTATION:

Notation 主要是用来表明文档中需要来自外部源的数据，而该数据 XML 本身是不能进行解析的，比如各种格式的二进制文件（比如图形文件、声音文件等），需要外部的应用程序进行处理。

Notation 声明的语法格式为：

<!NOTATION NAME ExternalID>

名师提点

需要注意的是 NAME 必须由字母、数字、句点、破折号或冒号组成，并且第一个字符必须为字母或者是下划线。

下面的例子表示 GIF 图象作为不解析的外部内容。

Code 3-32

```
<!NOTATION gif system "iexplore.exe">
<!ENTITY logo SYSTEM "http://somewebsite/somecategory/something.gif"
NDATA gif>
<!-- 这里 NDATA 表示 XML 不解析该数据-->
<!ELEMENT PIC EMPTY>
<!ATTLIST PIC loc ENTITY #REQUIRED>
```

然后，在具体的实例化文档中包含下面一行代码：<PIC loc="&logo;" />根据 DTD 定义，loc 属性值是一个不解析的实体。解析器可以根据 DTD 定义知道这一点，然后它就不对其进行解析，也不会象解析实体一样把它包括到 XML 文档里面。同时，XML 解析器将通知 iexplore.exe 该引用的存在。

属性的特点－#REQUIRED:

元素的所有实例都必须有该属性的值(NOT NULL)

语法：

<!ATTLIST 元素名 属性名 属性类型 #REQUIRED>

DTD 示例:

Code 3-33

```
<!ATTLIST person number CDATA   #REQUIRED>
```

笔记区

XML 示例：

Code 3-34

```
<person number="5677" />
```

属性的特点－#IMPLIED:

元素的实例中可以忽略该属性(NULL)。

语法：

<!ATTLIST 元素名 属性名 属性类型 #IMPLIED>

DTD 示例：

Code 3-35

```
<!ATTLIST contact fax CDATA #IMPLIED>
```

XML 示例：

Code 3-36

```
<contact fax="555-667788" />
```

属性的特点－#FIXED value:

元素实例中该属性的值必须为指定的固定值。

语法：

<!ATTLIST 元素名 属性名 类型 #FIXED "value">

DTD 示例：

Code 3-37

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

XML 示例：

笔记区

Code 3-38

```
<sender company="Microsoft" />
```

属性的特点—Default value:

为属性提供一个默认的值

语法：

<!ATTLIST 元素名 属性名 类型 "value">

DTD 示例：

Code 3-39

```
<!ATTLIST payment type CDATA "check">
```

XML 示例：

Code 3-40

```
<payment type="check" />
```

修饰符号:

表 3-1

符号	用途	示例	示例说明
()	用来给元素分组	（网一|网二|数码）	分成三组
|	在列出的对象中选择一个	（男人|女人）	表示男人或者女人必须出现，两者至少选一
+	该对象最少出现一次，可以出现多次（1 或多次）	（成员+）	表示成员必须出现，而且可以出现多个成员
*****	该对象允许出现零次到任意多次（0 到多次）	（爱好*）	爱好可以出现零次到多次
?	该对象可以出现，但只能出现一次（0 到 1 次）	（菜鸟?）	菜鸟可以出现，也可以不出现，如果出现的话，最多只能出现一次

笔记区

,	对象必须按指定的顺序出现	（西瓜,苹果,香蕉）	表示西瓜、苹果、香蕉必须出现，并且按这个顺序出现

3、定义实体

实体类型:

- 普通内部实体
- 普通外部实体
- 内部参数实体
- 外部参数实体

表 3-2

类型		普通实体	参数实体
使用场合		用在 XML 文档中	只用在 DTD 中元素和属性的声明中
声明方式	内部	<!ENTITY 实体名 "文本内容">	<!ENTITY % 实体名 "文本内容">
	外部	<!ENTITY 实体名 SYSTEM" 外部文件 URL 地址">	<!ENTITY % 实体名 SYSTEM "外部文件 URL 地址">
引用方式		&实体名;	%实体名;

普通内部实体的示例

Code 3-41

```
<?xml version="1.0" encoding="GB2312"?>
<!DOCTYPE COMPANY [
  <!ELEMENT COMPANY (NAME, ADDRESS)>
  <!ELEMENT NAME (#PCDATA)>
  <!ELEMENT ADDRESS (#PCDATA)>
  <!ENTITY name "湖北银河信息技术学院">
  <!ENTITY address "武汉市武昌区临江大道号">
]>
<!--this is a comment-->
<COMPANY>
```

```
  <NAME>&name;</NAME>
  <ADDRESS>&address;</ADDRESS>
</COMPANY>
```

普通外部实体的示例:

Code 3-42

```
<?xml version="1.0" encoding="gb2312"?>
<!DOCTYPE COMPANY [
  <!ELEMENT COMPANY (NAME, ADDRESS)>
  <!ELEMENT NAME (#PCDATA)>
  <!ELEMENT ADDRESS (#PCDATA)>

    <!ENTITY address SYSTEM
"http://somewebsite/somecategory/something.xml ">
]>
<COMPANY>
    < NAME > meimei</ NAME >
    <ADDRESS >&address;</ADDRESS >
</COMPANY>
```

外部实体的概念实际上很简单，比如在上面的例子中，我们的实体定义为：

Code 3-43

```
<!ENTITY address "武汉市武昌区临江大道号">
```

这里表示用"武汉市武昌区临江大道号"来代替 address,但是如果 address 指代的内容很大很复杂的时候，我们可以用一个外部文件来保存这部分的内容。比如采用如下的代码：

Code 3-44

```
<!ENTITY address system "http://somewebsite/somecategory/something.xml">
```

这里表示用文档 http://somewebsite/somecategory/something.xml 来表示实体 address 的具体内容。需要指出的是，这里的 something.xml 文档必须是一个格式完善的 XML 文档。

笔记区

参数实体:

所谓参数实体的概念就是说该实体实际上不是在具体实例化文档中使用，而是在 DTD 文档内部被使用，比如我们可以定义一个如下的实体：

Code 3-45

```
<!ENTITY %  地址 "街道,城市,邮编,国家">
```

然后可以在 DTD 内部通过%地址;来引用它，具体例子如下：

Code 3-46

```
<!ELEMENT 联系（人名,电话,%地址;）>
```

上面就是参数实体的概念。

外部参数实体和内部参数实体:

外部参数实体和内部参数实体的关系和普通外部实体与普通内部实体的关系一样，也就是说，实体的内容不是在两个引号之间表示，而是用一个外部的 XML 文档来表示，比如：

Code 3-47

```
<!ENTITY %地址 system "http://somewebsite/somecategory/something.xml">
```

然后可以在 DTD 内部通过%地址;来引用它。这里%地址;相当于一个普通的元素（ELEMENT），这就是外部参数实体的概念。

外部参数实体示例如下

1. 建立一个 XML 文档 A.XML 引用外部 DTD 文档 A.dtd

Code 3-48

```
A.xml
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE root SYSTEM "A.dtd">
<root>
  <name></name>
  <address>
    <C>Hubei</C>
    <S>wuhan</S>
  </address>
</root>
```

2. 建立一个 A.dtd 文档,在 A.dtd 文档中引用外部的参数实体 B.dtd

Code 3-49

```
A.dtd
<?xml version="1.0" encoding="utf-8" ?>
<!ELEMENT root (name,address)>
<!ELEMENT name (#PCDATA)>
<!ENTITY % myaddress SYSTEM "B.dtd">
%myaddress;
```

3. 建立一个 B.dtd 文档

Code 3-50

```
B.dtd
<?xml version="1.0" encoding="utf-8" ?>
<!ELEMENT address (C,S)>
<!ELEMENT C (#PCDATA)>
<!ELEMENT S (#PCDATA)>
```

总结

- 符合语法的 XML 文档称为结构良好的 XML 文档。
- 通过 DTD 验证的 XML 文档称为有效的 XML 文档。
- 文档类型定义--Document Type Definition
- DTD 用来描述 XML 文档的结构，一个 DTD 文档包含：
    - 元素(ELEMENT)的定义规则
    - 元素之间的关系规则
    - 属性(ATTLIST)的定义规则
    - 可使用的实体(ENTITY)或符号(NOTATION)规则
- DTD 文档与 XML 文档实例的关系
    - 类与对象
    - 数据库表结构与数据记录
- 有了 DTD，每个 XML 文件可以携带一个自身格式的描述。

笔记区

- 有了 DTD，不同组织的人可以使用一个通用 DTD 用来交换数据。
- 应用程序可以使用一个标准 DTD 校验从外部世界接受来的 XML 数据是否有效

可以使用 DTD 校验自己的 XML 数据

问与答：

1：用自己的语言来解释 DTD 与 XML 的关系

作业

1.分析完整的 DTD 文档，写 XML 实例

```
<?xml version="1.0" encoding="utf-16"?>
<!ELEMENT 属性列表 (联系人)*>
<!ELEMENT 联系人 (姓名,性别,EMAIL,电话+,地址)>
<!ELEMENT 地址 (街道,城市,省份)>
<!ELEMENT 姓名 (#PCDATA)>
<!ELEMENT 性别 (#PCDATA)>
<!ELEMENT EMAIL (#PCDATA)>
<!ELEMENT 电话 (#PCDATA)>
<!ELEMENT 街道 (#PCDATA)>
<!ELEMENT 城市 (#PCDATA)>
<!ELEMENT 省份 (#PCDATA)>
```

2.分析 XML 实例，写 DTD 文档

```
<?xml version="1.0" encoding="GB2312"?>
<学生名册>
    <学生 学号="1">
        <姓名>张三</姓名>
        <性别>男</性别>
        <年龄>20</年龄>
```

笔记区

```
    </学生>
    <学生 学号="2">
        <姓名>李四</姓名>
        <性别>女</性别>
        <年龄>19</年龄>
    </学生>
    <学生 学号="3">
        <姓名>王二</姓名>
        <性别>男</性别>
        <年龄>21</年龄>
    </学生>
</学生名册>
```

第四次可 命名空间与 Schema XML 架构

学习内容	难度
命名空间声明	★★
命名空间的角色	★★
※Schema 的产生及语法	★★★
※XML 的数据类型(Schema 几种数据类型)	★★★

注：带※为重点学习内容

笔记区

4.1 命名空间简述

命名空间这个名字对于我们来说应该不会陌生，如果你有编程的经验，那么你就会对某门编程语言比较熟悉，编程语言为了防止类型的命名冲突，里面都会有命名空间的概念，比如在 Microsoft .NET C# 语言中也叫命名空间，在 Java 语言中叫做包。相同名称的不同功能的类型名称装载在不同包或命名空间中，方便编程人员在编写代码时使用。

在 XML 中也有命名空间的概念，命名空间通过将元素和属性放入逻辑区域并分别提供区分元素和属性得的方式，将元素组织在一起。命名空间还用来引用特定的 DTD 或者 XML 模式，命名空间是在 XML1.0 正式发布之后才定义的。在 XML1.0 发布后，W3C 开始着手解决一些问题，其中之一就是命名的冲突，为了能理解这个问题的重要性，先考虑一下 Web 的未来。

在 W3C 进入 XML1.0 之后不久，大量语言开始出现，如数学标记语言 MathML，同步多媒体集成语言 SMIL，可缩放矢量图形（Scalable Vector Graphics ，SVG），XLink，XForms 和可扩展超文本标记语言（XHTML），为了改变因依赖于一种语言而需要忍受 Web 通信的负荷，有人提出了让多种语言共同工作的想法。如果可以将功能模块化，那么每一种语言就可以发挥其最大的优势；但是当开发人员需要在同一个应用程序中使用多个单词表的时候，问题就出现了。比如，有人需要在一个交互式的 Web 站点上使用 SVG、SMIL、XHTML 和 XForms 语言的组合。当词汇表混合的时候你就必须找到一种能够区分元素类型的办法。如下所示：

Code 4-1

```
<html>
    <head>
        <title>Book List</title>
    </head>
    <body>
        <books>
            <book>
                <title>企业级应用数据传技术</title>
                <author>漆美云，肖发辉，李灯登</author>
            </book>
```

笔记区

```
        <books>
    </body>
</htiml>
```

在这个示例中，有两个 title 元素我们无法区分，即使它们具有不同的语义。而命名空间则可以通过为元素和/或属性集合提供唯一的标识符来解决这个问题.做法主要是为每个成员元素和属性添加一个前缀名称，该名称在命名空间中唯一标识元素和属性.将元素组织在命名空间内可以让多个 XML 文档很容易地应用，而且可以让一个 XML 文档中引用多个命名空间.XML 命名空间是限定属性和元素名称的一种方式.这是通过 XML 文档内部将其命名空间相关联来完成的，而这些命名空间则通过资源指示器(Universal Resource Indicators，URI)来标识.

其实命名空间大家平时应该都用过，下面在 XHTML 中使用命名空间

<html xmlns="http://www.w3.org/1999/xhtml">

Xmlns 关键字是属性的一种特殊类型，用于声明一个 XML（元素）的命名空间。在引号之间的信息就是 URI，指向实际的命名空间——这里是 schema 它表明该文档的结构。URI 是区分命名空间的一种正式的方式，他不需要指向任何事物。URI 只用于唯一界定元素的属性。Xmlns 声明放置在使用命名空间的元素标签内。

名师提点

XML 初学者可能会比较困惑，因为命名空间名称使用的 URI，所以有很多人经常与指向其他资源的 Web 地址相混淆；但是，XML 命名空间名称是并不需要指向任何事物的 URL，比如如果您访问 XSLT 命名空间（http://www.w3c.org/1999/XSL/Transform）你将发现只有一句话："这是一个由 XSL 转换（XSLT）1.0 规范定义的 XML 命名空间"。唯一的标识符只是一个符号；因此对文档来说就不需要定义了。URL 是为了命名空间名称而选择的，因为它们包含了可以在全球 Internet 上使用的域名并且是唯一的。

下面的代码显示了使用命名空间来解决前面示例中出现的命名冲突的问题。

Code 4-2

```
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title>Book List</title>
```

笔记区

```
    </head>
    <body>
        <books xmlns=”http://www.yinheedu.com/books/xml”>
            <book>
                <title>企业级应用数据传技术</title>
                <author>漆美云，肖发辉，李灯登</author>
            </book>
        <books>
    </body>
</html>
```

Books 元素隶属于命名空间 http://www.yinheedu.com/books/xml，而所有的 XHTML 元素属于 XHTML 命名空间 http://www.w3.org/1999/xhtml。

4.2 命名空间的声明

声明一个命名空间需要三个部分：

Xmlns 指定值作为ＸＭＬ的命名空间，他是声明空间必须的，而且可以附加在任何ＸＭＬ元素上。

Prefix 指定一个命名空间前缀。它（包括冒号）只用于声明一个命名空间前缀。如果使用了这个前缀，那么在文档中使用该前缀（prefix：element）的任何元素都被认为是位于声明命名空间范围内。

NamespaceURI 这是唯一的标识符。该值不必指向一个 Web 资源；它只有一个具有符号意义的标识符。这个值是必需的并且必须定义在单个引号或者双引号之内。可以使用两种不同的方式定义一个命名空间；

- **默认命名空间**使用不带前缀的 xmlns 属性来定义命名空间，并且所有子元素都被认为是属于已定义的命名空间。默认命名空间就是一个让 XML 文档更具可读性且更易于写入的工具。如果您打算在文档系统中统一使用一个命名空间，那么利用这个命名空间的前缀就能很容易地消除每个元素的前缀。

- **带有前缀的命名空间**使用带有前缀的 xmlns 属性来定义命名空间。当前缀附加在一个元素之前时，认为它属于那个命名空间。

下面的示例演示了**默认命名空间**和**具有前缀的命名空间**的用法。

Code 4-3

```
<html xmlns=”http://www.w3.org/1999/xhtml”>
    <head>
        <title>Book List</title>
    </head>
    <body>
        <booklist:books
xmlns:booklist=”http://www.yinheedu.com/books/xml”>
            < booklist:book>
                < booklist:title>企业级应用数据传技术</ booklist:title>
                < booklist:author>漆美云，肖发辉，李灯登</ booklist:author>
            </ booklist:book>
        < booklist:books>
    </body>
</html>
```

在 HTML 元素中定义 xmlns 是应用于所有的元素的默认命名空间，它没有显式地定义一个命名空间：但是 books 元素使用 booklist 前缀显式地定义一个命名空间。因为这个前缀是声明在 books 元素的时候使用的，所以在 books 下所有元素都被认为使用了具有前缀的命名空间。

4.3 命名空间的角色

命名空间是一套 XML 元素和它们的构成属性。当您继续深入研究 XML 的时候，如为 Web 站点创建交互式的 XML Web 页面和为传输数据建立指导等，您将会发现 XML 命名空间绝对是重要。以下是关于命名空间的一些用法。

1、 重用

命名空间可以允许任何数量的 XML 文档引用它们。这样就可以按照需求重用命名空间，而不是强迫开发人员为他们创建每一文档都重复开发。例如，考虑一个普通的业务场景，有两个需要交换普通 XML 格式的应用程序：服务器依赖于一个特定的命名空间并生成 XML，

笔记区

而客户机同样依赖于同一个命名空间并使用这个 XML。这样就不用生成两个命名空间（一个应用程序使用一个），而是两个应用程序在它们生成 XML 中引用同一个命名空间。这样使得可以重用命名空间，这是一个很重要的特性。命名空间不但可以被一个应用程序的不同部分重用，而且还可以被任意多个应用程序的不同部分重用。因此花精力开发一个考虑周全的命名空间有时可以获得很好的回报。

2、 多命名空间

就像多个 XML 文档可以引用同一个命名空间一样，一个文档也可以引用多个命名空间。这是将元素按逻辑，顺序进行分组后自然产生的副产品。正如软件开发经常需要将大型程序分解成更小的程序一样，命名空间通常也会被分成更小，更合理的组。创建一个具有您认为需要的所有元素的大型命名空间是没有意义的。这将使得开发困难并且使得任何必须使用这样的 XML 文档结构的人感到困惑。所以，应当开发小型的，更加自热的命名空间来包含必要的元素。

例如，可以按照组件来创建命名空间，并将它们集合到一起形成大型的程序所需的词汇表中。比如，某个应用程序可以执行服务来帮助永固从一个电子商务 Web 站点中购买商品，这个应用程序将会需要元素来定义商品种类，生产者，销售人员等。命名空间中就可以将这些词汇表包括一个 XML 文档中，并按需求从每个命名空间中抽取出使用。

3、 混淆

命名空间有时会重叠并包含相同元素。所以在 XML 文档依赖于命名空间的时候毫无疑问地会产生问题。这样一恶搞冲突的示例是包含了 book 顺序的命名空间和另外一个包含了 book 目录元素的命名空间。两个命名空间都会使用元素来引用书籍的标题或作者的姓名。当一个文档试图应用这两个命名空间的元素时，就会给 XML 解析器造成麻烦。可以通过将书籍顺序元素和数据目录元素限制在单独的命名空间中来解决这个问题因为属于特定命名空间的元素和属性也会因为各自的命名空间而被区分开，所以它们不会与其他命名空间元素属性冲突。这样就解决了前面所提到的混淆的问题。通过使用命名空间前缀来定义特定的元素和属性名称，解析器就能够正确的识别任何潜在的命名空间冲突，使用命名空间前缀的过程是为文档中所使用每个元素和属性创建了限定名称。

4.4 Schema XML 架构简述与产生

由于 DTD 的局限性，使得 W3C 致力于寻找解决的方案。从而花费了大量的时间，但

是其中几个已经形成的重要建议似乎都不能完全解决问题，其中包括颇具影响的微软的专有 XML 数据规范。当然 W3C 自己的工作草案就是制定了 XML Schema，W3C XML Activity Page 声明，尽管 XML 1.0 提供了一种机制，文档类型定义 DTD 给标记的使用加了限制，但是对 XML 文档的自动处理需要更严格更全面的工具，其需要主要体现在对应用软件各部分的结合，文档结构、属性和数据类型等等的约束 XML Schema 的语法完全遵循 XML 的语法规范，因此用户不用像学习 DTD 一样再学习一遍其他的语法，而可以自己使用。

4.5 Schema 的产生

关于 DTD 和 XML SCHEMAS 的关系，前面已经介绍过一些内容。DTD 的出现只是简单的从 SGML 中发展而来的 DTD 中明显还深刻着 SGML 的烙印 与 XML 的巨变和成功相比 DTD 尚没有实质性的飞跃，其中关键的一点是 DTD 有自己的特殊语法，它虽然可以用来规定限制 XML 。但我们需要认真学习 DTD 才能熟练地创建和编写 XML 文档，而且我们必须有两套不兼容的解析器，一套用来分析 XML 如判断其结构是否良好，另一套用来分析 DTD 再用分析的结果去检查 XML 文档，判断 XML 是否有效， DTD 只提供了非常有限的几种数据类型 DTD 中规定的文档内容都是字符数据，连整型、浮点型、数据型、布尔型这样简单的数据类型都不提供,更别谈更复杂的数据类型了 DTD 不支持名域(Namespace)机制。虽然 DTD 可以为不同的 XML 所引用，但一个 XML 文档只能有一个相对应的 DTD 不同的 DTD 并不能同时对同一个 XML 文档进行规定，哪怕只是对其中的部分元素，也就是说传统的 DTD 机制使得 XML 的继承性和重用性受到限制 DTD 有扩展的机制，但这个机制太复杂而且很脆弱 DTD 扩展机制的最大毛病在于不能清楚的表达相互之间的关系，两个有着完全相同内容的元素怎么做也不能互相联系。同样的，被定义为参数实体的不同属性之间不能建立任何联系，另外 DTD 中的内容模型是不开放的，它不能随意扩充内容，否则将无法被解析，以上的这些缺点事实上已经在束缚 XML 的发展，重新制定符合 XML 的大纲机制已经是大势所趋，这样的新机制就是 Schema。

下面这段代码是一个 XML 文件我们来看看，使用 DTD 文档，和使用 Schema 来定义这个文档的差异。

XML 数据

Code 4-1

```
<书本>
```

笔记区

```
    <名称>企业级应用数据传技术</名称>
    <作者>漆美云，肖发辉，李灯登</作者>
</书本>
```

DTD 定义

Code 4-2

```
<!ELEMENT 书本 (名称，作者)>
<!ELEMENT 名称 (#PCDATA)>
<!ELEMENT 作者 (#PCDATA)>
```

Schema 定义

Code 4-3

```
<element name="书本" type="书本类型" />
<complexType name="书本类型" >
        <element name="名称" type="string"/>
        <element name="作者" type="string"/>
</complexType>
```

在这里我们可以看到，DTD 文档的语法是独立的体系，而 Schema 的语法则是符合 XML 的用法。而且我们可以看到 Schema 中的 type 属性指定元素类型，是可以自定义的，这也是 Schema 相对于 XML 的优势和改进的地方。

Schema 相对于 DTD 的明显好处是 XML Schema 文档本身也是 XML 文档，而不是像 DTD 一样使用自成一体的语法。这就方便了用户和开发者，因为可以使用相同的工具来处理 XML Schema 和其他 XML 信息，而不必专门为 Schema 使用特殊工具。Schema 简单易懂，懂得 XML 语法、规则的人都可以立刻理解它。

什么是 XML 模式(Schema)

1. 在 Schema 之前另有一种定义 XML 结构的方式，即 DTD， 两者的区别在于 XML Schema 是 XML 文档，而 DTD 有自己的特殊语法。这样一来你只需懂得 XML 的语法规则即可编写 Schema，无需学习其他语法规则，xml 文件与 xml schema 文件可以用相同的

笔记区

语法分析器来解析，而无须写两套分析器 xml schema 有强大 易用的扩展功能。

2. XML Schema 利用名域将文档中特殊的节点与 schema 说明相联系，一个 xml 文件可以有多个对应的 schema，而用 DTD 的一个 xml 文件中只能有一个相对应的 DTD 。

3. XML Schema 内容模型是开放的，可以随意扩充而 DTD 将无法解析扩充的内容。

4. DTD 只能把内容类型定义为一个字符串，而 XML Schema 允许你把内容类型定义为整型、浮点型、数据型、布尔型或者许多其他的简单数据类型 而无须重定义。

如下图结合前面的内容，我们可以看到 DTD 和 Schema 所处的位置，和他的作用。

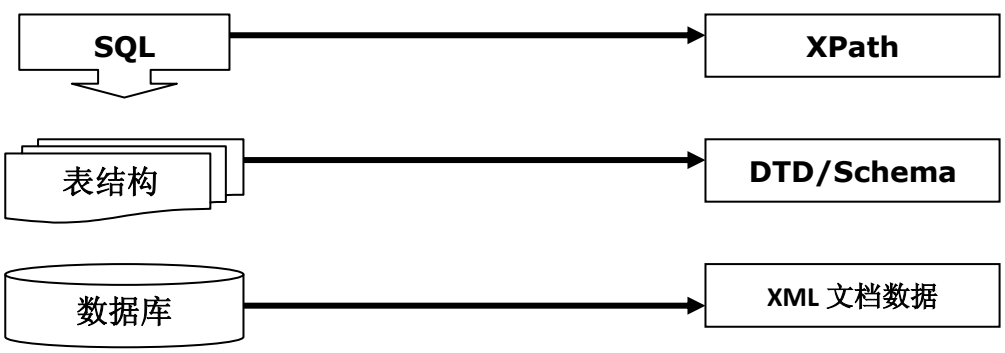


图 4-1

4.6 Schema 语法

现在我们已经知道 Schema 实际上是使用 XML 语法写成的，这意味着 XML 模式实际上只是 XML 的另一个应用而已，一个为了定义 XML 文档类的词汇表，正是如此 XML 才拥有了一个可以描述自己的模式，每个 Schema 都有自己定义模式的元素和属性的地方我们称之为结构，在这里元素的内容模型得到了描述，内容模型明确地描述了允许的元素的内部结构，可以说结构是 XML 模式的核心。

第一个 Schema 文档 Visual Studio 2008 Schema 设计器

首先可以在 **Visual Studio 2008** 项目中新建一个 XML 架构。

笔记区

图 4-2

然后 Visual Studio 2008 会自动打开 XML 架构的设计界面如下：

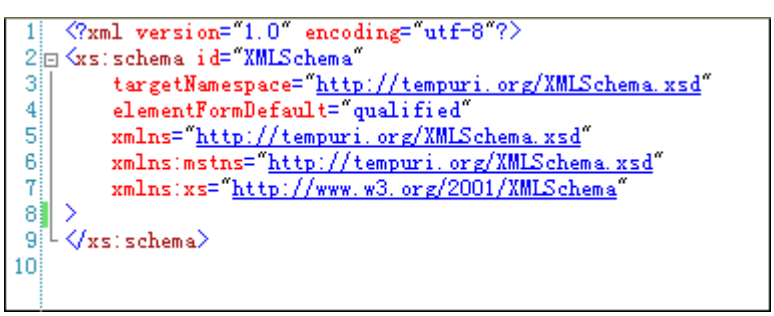


图 4-3

Code 4-4

```
<xs:schema id="XMLSchema"
    targetNamespace="http://tempuri.org/XMLSchema.xsd"
    elementFormDefault="qualified"
    xmlns="http://tempuri.org/XMLSchema.xsd"
    xmlns:mstns="http://tempuri.org/XMLSchema.xsd"
```

```
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="书本">
    <xs:complexType>
      <xs:sequence />
    </xs:complexType>
  </xs:element>
</xs:schema>
</xs:schema>
```

这样一个元素就定义好了，我们可以修改一下元素的名称，然后再来看看设计器为我们做了些什么。从上面的文档我们可以看到，VisualStudio 设计 XML 架构设计器，帮我们完成了大量的代码编写工作，这样我我们就可以从复杂繁琐的代码编辑中抽出来，专心分析和设计 XML 架构。紧接着我们可以给他添加两个子元素，一个属性。

Visual Studio 2008 中的设计代码

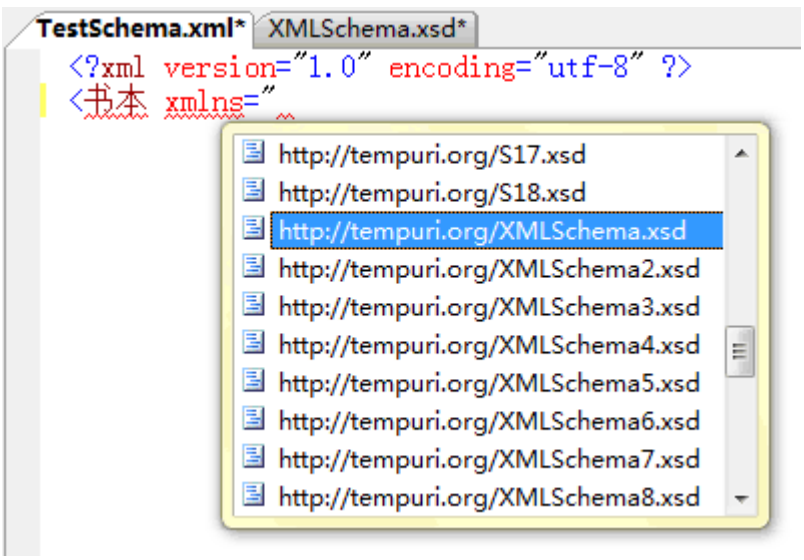
Code 4-5

```
<xs:schema id="XMLSchema"
    targetNamespace="http://tempuri.org/XMLSchema.xsd"
    elementFormDefault="qualified"
    xmlns="http://tempuri.org/XMLSchema.xsd"
    xmlns:mstns="http://tempuri.org/XMLSchema.xsd"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="书本">
<xs:complexType>
      <xs:sequence>
    <xs:element name="名称" type="xs:string" />
<xs:element name="作者" type="xs:string" />
      </xs:sequence>
      <xs:attribute name="编号" type="xs:string" />
    </xs:complexType>
  </xs:element>
</xs:schema>
```

笔记区

这样一个 XML 架构文档就定义好了，如何在 XML 文档中应用 XML 架构呢？

在 xml 中应用 schema 的方法是，首先写入根元素，然后定义根元素的命名空间，在命名空间中选择特定的命名空间，然后就可以编写 XML 其他部分的内容了。




```
<?xml version="1.0" encoding="utf-8" ?>
<书本 编号="1"  xmlns="http://tempuri.org/XMLSchema.xsd">
  <名称>企业级应用数据传技术</名称>
  <作者>信息安全部</作者>
</书本>
```

图 4-4

这样我们就在.net 中完成了我们的第一个 XML 架构文档了，下面的内容我们将会看到在 XML 架构设计中的一些细节。

笔记区

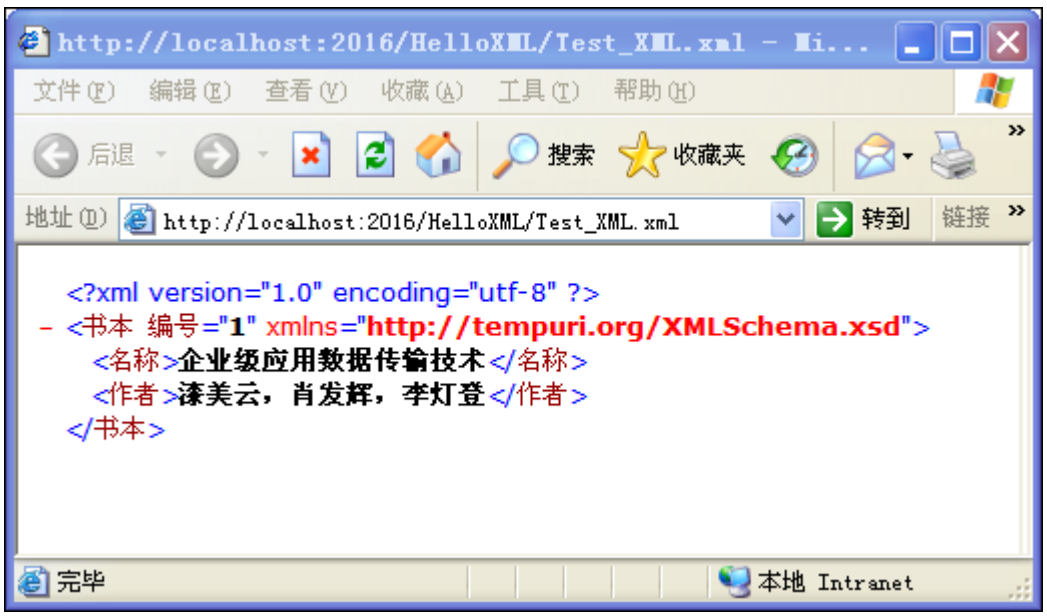


图 4-5

4.7 XML 的数据类型

- 简单类型
  - 内置的数据类型（**built-in data types**）
    - 基本的数据类型
    - 扩展的数据类型
  - 用户自定义数据类型（通过 **simpleType** 定义）
- 复杂类型（通过 **complexType** 定义）

1、 Schema 的基本数据类型

表 4-1

数据类型	描述
string	表示字符串
Boolean	布尔型
decimal	代表特定精度的数字
float	表示单精度 32 位浮点数

笔记区

double	表示双精度 64 位浮点数
duration	表示持续时间
dateTime	代表特定的时间
time	代表特定的时间，但是是每天重复的
date	代表日期
hexBinary	代表十六进制数
anyURI	代表一个 URI，用来定位文件
NOTATION	代表 NOTATION 类型

2、 Schema 扩展的数据类型

表 4-2

数据类型	描述
ID	用于唯一标识元素
IDREF	参考 ID 类型的元素或属性
ENTITY	实体类型
NMTOKEN	NMTOKEN 类型
NMTOKENS	NMTOKEN 类型集
long	表示整型数，大小介于-9223372036854775808 和 9223372036854775807 之间
int	表示整型数，大小介于-2147483648 和 2147483647 之间
short	表示整型数，大小介于-32768 和 32767 之间
byte	表示整型数，大小介于-128 和 127 之间

2、 Schema 数据类型的特性

表 4-3

特性	描述
enumeration	在指定的数据集中选择，限定用户的选值
fractionDigits	限定最大的小数位，用于控制精度
length	指定数据的长度
maxExclusive	指定数据的最大值（小于）
maxInclusive	指定数据的最大值（小于等于）
maxLength	指定长度的最大值
minExclusive	指定最小值（大于）
minInclusive	指定最小值（大于等于）
minLength	指定最小长度
Pattern	指定数据的显示规范

4.8 Schema 的元素类型

表 4-4

元素名称	描述
schema	schema 文档根元素
element	元素
attribute	属性
group	组
attributeGroup	属性组
simpleType	简单类型

笔记区

simpleContent	简单内容
complexType	复杂类型
choice	选择结构
list	列表
union	组合
unique	唯一
sequence	顺序结构
restriction	限定

schema 元素：

作用：Schema 声明

用法：<xs:schema>

属性：xmlns targetNamespace elementFormDefault

XML Schema 文档(*.xsd)的根元素为 schema 元素，一般位于 XML Schema 的命名空间 xs (XML Schema)或 xsd (XML Schema definition)中。Schema 元素含有多个属性，常用的有 XML Schema 的命名空间声明（必须）、其他在本文档中要用到的命名空间的声明（可选）、该文档描述的目标名字空间（可选）、语言（可选）等。

文件名 **XMLSchema.xsd**

Code 4-6

```
<xs:schema id="XMLSchema"
    targetNamespace="http://tempuri.org/XMLSchema.xsd"
elementFormDefault="qualified"
    xmlns="http://tempuri.org/XMLSchema.xsd"
    xmlns:mstns="http://tempuri.org/XMLSchema.xsd"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
<!--放入文档定义内容 -->
```


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

targetNamespace=http://mynamespace/myschema

当前 schema 定义的元素和数据类型属于 http://tempuri.org/ XMLSchema.xsd 命名空间要与当前文档的文件名相同。

elementFormDefault="unqualified"

如果该值是 unqualified，实例 xml 的根元素必须有命名空间的的限定，这个命名空间必须是 schema 中定义的 targetNameSpace。但是其下子元素无须也不允许用命名空间前缀限定目标命名空间。 子元素的命名空间为空命名空间。

如果该值是 qualified，实例 xml 根元素及其下所有子元素都必须通过命名空间前缀限定目标命名空间。这个命名空间必须是 schema 中定义的 targetNameSpace。

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

所有 Schema 文档使用 schema 作为其根元素，这里指定了用于构造 schema 的元素和数据类型来自 http://www.w3.org/2001/XMLSchema 命名空间，并设定了命名空间的前缀 xs 之后的所有 Schema 的元素都要以 xs 打头。

element 元素：

作用：声明一个元素

属性：name　type　ref

这里我们来看一个案例，定义了 3 个元素 cat，dog，pets，其中 pets 为复合类型，它包括对 cat 和 dog 元素的引用。

Code 4-7

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema id="S10" targetNamespace="http://tempuri.org/S10.xsd"
elementFormDefault="qualified" xmlns="http://tempuri.org/S10.xsd"
xmlns:mstns="http://tempuri.org/S10.xsd"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="cat" type="xs:string" />
  <xs:element name="dog" type="xs:string" />
  <xs:element name="pets">
    <xs:complexType>
      <xs:sequence>
```

笔记区

```
        <xs:element ref="cat" />
        <xs:element ref="dog" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

S10.xml

Code4-8

```
<?xml version="1.0" encoding="utf-8" ?>
<pets xmlns="http://tempuri.org/S10.xsd">
  <cat>mimi</cat>
  <dog>wangwang</dog>
</pets>
```

总结

本章讲解了 XML 在应用过程中碰到的元素名称冲突问题，以及之后出现的解决这个问题的方案--命名空间，以及命名空间是如何声明的，命名空间前缀的使用。以及多命名空间的概念。同时在本章中我们了解到了 DTD 在现在 XML 应用中的局限性，XML 应用中急需一个能更完善定义 XML 文档的结构，Schema 的出现解决了这个问题。Schema 是另一种文档类型定义，它遵循 XML 的语言规范。Schema 是可扩展的，支持命名空间；Schema 支持更多的数据类型与元素类型；Schema 用 element 元素声明元素，用 attribute 声明元素的属性。

问与答

1、 命名空间需要哪三个部分?

2、 DTD 与 Schema 有什么区别？

3、 为什么会出现 Schema?

笔记区

作业

1、书写 Schema 文件， 要求以下 XML 文件能通过验证

```
<学生>
  <信息>
    <姓名>张三</姓名><性别>男</性别><年龄>20</年龄>
  </信息>
  <信息>
    <姓名>李四</姓名><性别>女</性别><年龄>30</年龄>
  </信息>
</学生>
```

2、编写关于班级学生信息的 XSD 文档，要求如下：

（1）用记事本编写某班级的学生信息，要求符合 XML 语言的规范。

（2）编写中每个学生要有姓名、年龄、电子邮箱、身高、电话、单位等信息，单位又包含地址、邮编等信息。每个学生要有电话或手机。每个学生都要有一个“编号”属性作为标识

第五次课 Schema XML 架构及数据类型

学习内容	难度
※XML Schema 简单类型定义	★★★
※XML Schem 复杂类型定义	★★★

注：带※为重点学习内容

笔记区

5.1 简单类型定义(simpleType)

除了内置数据类型外，XML Schema 允许用户自己定义新的数据类型，包括：

简单类型定义(simpleType)：定义数据的结构和范围，不允许含文档元素

复杂类型定义(complexType)：定义文档的逻辑结构，允许含文档元素带属性

在 Schema 中，也是通过对元素的定义和元素关系的定义来实现对整个文档性质和内容的定义。同时需要注意的是，在 Schema 中，元素是通过它的名字和内容模型来确定，名称就是该元素的名字，这个大家都可以理解，而内容模型实际上就是表示元素的类型。就象在 C#中，我们可以随便定义一个变量，但是必须定义变量的类型，变量的类型就可能有多种形式，它可以是一个简单的变量（如 C#内部指定的类型，bool,int,double,char 等等），也可以是很复杂的类型(比如是一个 struct 或者是 class)，在 Schema 中也是一样，类型(type)可以分为两种形式，一种是非常简单的类型，被称为 simple，一种是复杂的类型，被称为 complex。简单类型不能包含元素和属性（注意在 Schema 中和 DTD 中一样，都有元素属性的说法，大道相同）。而复杂类型不仅可以包含属性，而且可以在其中嵌套其他的元素，或者可以和其他元素中的属性相关联。

用户可以用元素 simpleType 在内置数据类型的基础上定义自己的新类型，简单类型定义(Simple Type Definition)的完整格式为：

Code 5-1

```
<simpleType
  final = (#all | (list | union | restriction))
  id = ID
  name = "类型名"
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?,  (restriction | list | union))
</simpleType>
```

常用格式为：

<simpleType name = "类型名">

子元素：可选的 annotation（注释）后跟 restriction（约束）、list（列表）、union（联合）之一

</simpleType>

约束(restriction):restriction 元素用于定义新类型对基类型的约束和限制

完整格式为：

Code 5-2

```
<simpleType name='Sku'>
  <restriction base='string'>
    <pattern value='\d{3}-[A-Z]{2}'/>
  </restriction>
</simpleType>
```

常用格式为：

<restriction base = “基类型”>

子元素：可选的一个注释 annotation 和（用于无 base 属性时定义匿名基类型的）另一个简单类型定义 simpleType，后跟若干约束

</restriction>

Sku 为可用于电子商务的条码(barcode)数字类型，取值为 nnn-XX，即 3 个十进制数字后跟连字符再跟两个大写字母。

length（长度）：

限制数据长度单位之数量为指定值，对 string 以及派生类 length 为字符数。Length 的常用格式为:

<length value = ”长度值” fixed= ”true | false”/>

Code 5-3

```
<simpleType name='productCode'>
  <restriction base='string'>
    <length value='8' fixed='true'/>
  </restriction>
</simpleType>
```

minLength（最小长度）：

笔记区

限制长度的最小值，值也必须为 nonNegativeInteger 类型，minLength 必须小于等于 maxLength。minLength 元素常用格式为：

<minLength value = ”最小值”/>

Code 5-4

```
<simpleType name='non-empty-string'>
  <restriction base='string'>
    <minLength value='1'/>
  </restriction>
</simpleType>
```

maxLength（最大长度）：

限制长度的最大值，值也必须为 nonNegativeInteger 类型，maxLength 必须大于等于 minLength。maxLength 元素常用格式为：

<maxLength value = ”最大值”/>

Code 5-5

```
<simpleType name='form-input'>
  <restriction base='string'>
    <maxLength value='50'/>
  </restriction>
</simpleType>
```

pattern（样式）：

限制被约束类型的取值空间，用特殊的正则表达式来表示。pattern 元素常用格式为：

<pattern value = ”样式正则表达式”/>

Code 5-6

```
<simpleType name='better-us-zipcode'>
  <restriction base='string'>
    <pattern value='[0-9]{5}(-[0-9]{4})?'/>
  </restriction>
</simpleType>
```

表示 better-us-zipcode（更好的美国邮政编码）类型的数据可以取值形为

nnnnn[-nnnn]，即 5 个十进制数字符号后跟可选的 4 个十进制数字符号，如 92697-3425、94034。

enumeration（枚举）：

限制数据取值空间为若干值，常用格式为：

<enumeration value = ”枚举值”/>

Code 5-7

```
<xsd:simpleType name="dayType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="Sunday"/>
    <xsd:enumeration value="Monday"/>
    <xsd:enumeration value="Tuesday"/>
    <xsd:enumeration value="Wednesday"/>
    <xsd:enumeration value="Thursday"/>
    <xsd:enumeration value="Friday"/>
    <xsd:enumeration value="Saturday"/>
  </xsd:restriction>
</xsd:simpleType>
```

maxInclusive（最大包含）：

指定取值的上限(<=)，包括上限值。常用格式为：

<maxInclusive value = ”最大值”/>

Code 5-8

```
<simpleType name='one-hundred-or-less'>
  <restriction base='integer'>
  <maxInclusive value='100'/>
  </restriction>
</simpleType>
```

maxExclusive（最大不包含）：

指定取值的上限(<)，不包括上限值。常用格式为：

<maxExclusive value = ”最大值”/>

Code 5-9

笔记区

```
<simpleType name='less-than-one-hundred-and-one'>
  <restriction base='integer'>
    <maxExclusive value='101'/>
  </restriction>
</simpleType>
```

minExclusive（最小不包含）：

指定取值的下限(>)，不包括下限值。常用格式为：

<minExclusive value = ”最小值”/>

Code 5-10

```
<simpleType name='more-than-ninety-nine'>
  <restriction base='integer'>
    <minExclusive value='99'/>
  </restriction>
</simpleType>
```

minInclusive（最小包含）：

指定取值的下限(>=)，包括下限值。常用格式为：

<minInclusive value = ”最小值”/>

Code 5-11

```
<simpleType name='one-hundred-or-more'>
  <restriction base='integer'>
    <minInclusive value='100'/>
  </restriction>
</simpleType>
```

下面是其他若干使用约束的简单类型定义的例子：

性别类型（枚举约束）：

Code 5-12

```
<xsd:simpleType name="性别类型">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="男"/>
```

```
    <xsd:enumeration value="女"/>
  </xsd:restriction>
</xsd:simpleType>
```

汉族姓名类型（长度约束）：

Code 5-13

```
<simpleType name="姓名类型">
  <restriction base='string'>
    <minLength value='2'/>
    <maxLength value='4'/>
  </restriction>
</simpleType>
```

价格类型（位数约束）：8 位整数（亿元）+ 两位小数（角分）

Code 5-14

```
<simpleType name='价格类型'>
  <restriction base='decimal'>
    <totalDigits value='10'/>
    <fractionDigits value='2' fixed='true'/>
  </restriction>
</simpleType>
```

成绩类型（范围约束）：

Code 5-15

```
<simpleType name='scoreType'>
  <restriction base='integer'>
    <minInclusive value='0'/>
    <maxInclusive value='100'/>
  </restriction>
</simpleType>
```

摄氏体温类型（位数与范围约束）：

Code 5-16

笔记区

```
<simpleType name='celsiusBodyTemp'>
  <restriction base='decimal'>
    <totalDigits value='4'/>
<fractionDigits value='1'/>
    <minInclusive value='36.4'/>
    <maxInclusive value='40.5'/>
  </restriction>
</simpleType>
```

列表(list):

由同一类型的数据组成的表项序列。常用格式：

<list itemType=”表项类型”/>

- 其中的表项类型必须是原子类型（内置类型或用户用 simpleType 定义的简单类型）或联合类型，不能是列表类型或用户用 complexType 定义的复杂类型
- 若无 itemType 属性，则必须包含一个 simpleType 子元素来定义匿名表项类型
- 使用时，文档中列表数据的表项之间用白空符分隔

浮点数列表：

Code 5-17

```
<simpleType name='listOfFloat'>
  <list itemType='float'/>
</simpleType>
```

成绩表：

Code 5-18

```
<element name="成绩表">
  <simpleType>
    <!-- 匿名元素类型 -->
    <list itemType='scoreType'/>
```

笔记区

```
  </simpleType>
</element>
```

使用：<成绩表>87 100 60 95 59 0 … </成绩表>

移动序列：

Code 5-19

```
<simpleType name='moveList'>
  <list>
<simpleType>
          <!-- 匿名表项类型-->
  <xsd:restriction base="xsd:string">
          <xsd:enumeration value="left"/>
          <xsd:enumeration value="right"/>
          <xsd:enumeration value="up"/>
          <xsd:enumeration value="down"/>
        </xsd:restriction>
      </simpleType>
    </list>
  </simpleType>
<element name="moves" type="moveList"/>
```

使用：<moves> up left down down left up right </moves>

联合(union):

若干简单的成员类型的联合（~C 语言的 union），成员类型可以是原子类型、列表和其他联合类型。常用格式为：

<union memberTypes = "类型列表"/>

其中成员类型列表中的表项以白空符分隔。注意，若无 memberTypes 属性，则必须含若干 simpleType 子元素来定义匿名成员类型。如整数类型：

Code 5-20

```
<simpleType name="intType">
  <union memberTypers="byte short int long"/>
</simpleType>
```

笔记区

```
<element name="number" type="intType"/>
<element name="moves" type="moveList"/>
```

使用：<number>2</number> <number>1000000</number>

又如字体尺寸：

Code 5-21

```
<xsd:attribute name="size">
  <xsd:simpleType>
    <!-- 匿名属性类型 -->
    <xsd:union>
      <xsd:simpleType>
<!-- 匿名成员类型 1 -->
        <xsd:restriction base="xsd:positiveInteger">
  <xsd:minInclusive value="8"/>
          <xsd:maxInclusive value="72"/>
        </xsd:restriction>
      </xsd:simpleType>
      <xsd:simpleType>
        <!-- 匿名成员类型 2 -->
        <xsd:restriction base="xsd:NMTOKEN">
          <xsd:enumeration value="small"/>
          <xsd:enumeration value="medium"/>
          <xsd:enumeration value="large"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:union>
  </xsd:simpleType>
</xsd:attribute>
```

使用： A header

 this is a test

笔记区

5.2 复杂类型定义(complexType)

在 XML Schema 中，复杂类型定义（complexType 元素）被用来定义文档的结构，具体功能有：

- 利用提供的属性声明来控制属性的外观与内容，以约束元素信息项
- 限制元素信息项子元素为空、或符合指定的只有元素或混合内容模型，或限制字符信息项子元素符合指定的简单类型定义
- 使用类型定义层次机制从另一简单或复杂类型派生一个复杂类型
- 指定元素的后模式验证信息集分布(post-schema-validation infoset contributions)
- 限制从一个给定的复杂类型派生额外类型的能力
- 控制在一个实例中替换该内容模型，声明为一个给定复杂类型元素的派生类型元素的权限

常用格式为：

```
<complexType name = “类型名”>
    ((group | all | choice | sequence)?， ((attribute | attributeGroup)*， anyAttribute?)))
</complexType>
```

其中(group | all | choice | sequence)为内容模型，((attribute | attributeGroup)*，anyAttribute?))为属性声明。

下面是内容模型中的几种常用子元素的含义与用法：

用于定义元素的子元素为顺序序列，格式为：

Code 5-22

```
<sequence
  maxOccurs = "非负整数，并大于 minOccurs"
  minOccurs = "非负整数">
</sequence>
```

例如：美国地址

Code 5-23

```
<complexType name="USAddress">
```

笔记区

```
  <sequence>
    <element name="name" type="string"/>
    <element name="street" type="string"/>
    <element name="city" type="string"/>
    <element name="state" type="string"/>
    <element name="zip" type="decimal"/>
  </sequence>
  <attribute name="country" type="NMTOKEN" fixed="US"/>
</complexType>
<element name="address" type="USAddress"/>
```

使用：

Code 5-24

```
<address>
  <name>Paul Hoffman</name>
  <street>127 Segre Place</street>
  <city>Santa Cruz</city>
  <state>CA</state>
  <zip>95060</zip>
</address>
```

DTD：

Code 5-25

```
<?xml version="1.0" encoding="utf-8"?>
<!ELEMENT address (name， street， city， state， zip)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT street (#PCDATA)>
<!ELEMENT city (#PCDATA)>
<!ELEMENT state (#PCDATA)>
<!ELEMENT zip (#PCDATA)>
<!ATTLIST address country CDATA #FIXED "US">
```

又如：

Code 5-26

```
<xsd:complexType name="学生类型">
  <xsd:sequence>
    <xsd:element name="姓名" type="xsd:string"/>
    <xsd:element name="性别" type="性别类型"/>
    <xsd:element name="年龄" type="年龄类型"/>
    <xsd:element name="地址">
      <xsd:complexType>
        <!-- 匿名元素类型定义 -->
        <xsd:sequence>
          <xsd:element name="省份" type="xsd:string"/>
          <xsd:element name="城市" type="xsd:string"/>
          <xsd:element name="街道" type="xsd:string"/>
        </xsd:sequence>
      </xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
```

选择(choice):

用于定义元素的子元素为多中选一，格式为：

Code 5-27

```
< choice
  maxOccurs = "非负整数，并大于 minOccurs"
  minOccurs = "非负整数">
</ choice >
  <xs:attribute name="MyAttribute" type="xs:decimal" />
  <xs:group name="myGroupOfThings">
    <xs:sequence>
      <xs:element ref="thing1" />
```

笔记区

```
      <xs:element ref="thing2" />
    </xs:sequence>
  </xs:group>
  <xs:complexType name="myComplexType">
    <xs:group ref="myGroupOfThings" />
    <xs:attribute ref="MyAttribute" />
  </xs:complexType>
  <xs:element name="root" type="myComplexType">
  </xs:element>
</xs:schema>
```

例如：

Code 5-28

```
<complexType name="个人信息类型">
  <sequence>
    <element name="姓名" type="姓名类型"/>
    <element name="性别" type="性别类型"/>
    <choice>
      <element name="妻子" type="姓名类型"/>
      <element name="丈夫" type="姓名类型"/>
    </choice>
  </sequence>
</complexType>
```

使用：

Code 5-29

```
<个人信息>
  <姓名>张三</姓名>
  <性别>男</性别>
  <妻子>李四</妻子>
</个人信息>
```

以下案例首先定义 4 个被选的元素数据类型都为 string，名称分别为 selected，unselected，dimpled，perforated，并定义一个复杂类型 chadState 对以上元素进行引用，同时也定义了一个名为 candidate 类型的 stirng 的属性。最后定义一个名为 Root 的元素，类型为 chadState。大家要注意的是 choice 的使用是在当选中复杂类型 chadState 之后选择属性，可以看到 Order 属性，在其中可以看到有多种元素结构，我们这里悬着 choice，然后在 GroupChild 中有会有两个属性 minOccurs 和 maxOccurs，设置元素最小出现次数和最大出现次数。

Visual Studio 2008 中的设计代码

S14.xsd

Code 5-30

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema id="S14" targetNamespace="http://tempuri.org/S14.xsd"
elementFormDefault="qualified" xmlns="http://tempuri.org/S14.xsd"
xmlns:mstns="http://tempuri.org/S14.xsd"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="selected" type="xs:string">
  </xs:element>
  <xs:element name="unselected" type="xs:string">
  </xs:element>
  <xs:element name="dimpled" type="xs:string">
  </xs:element>
  <xs:element name="perforated" type="xs:string">
  </xs:element>
<xs:complexType name="chadState">
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="selected" />
```

笔记区

```
        <xs:element ref="unselected" />
        <xs:element ref="dimpled" />
        <xs:element ref="perforated" />
</xs:choice>
    <xs:attribute name="candidate" type="xs:string" />
  </xs:complexType>
  <xs:element name="Root" type="chadState">
  </xs:element>
</xs:schema>
```

S14.xml

Code 5-31

```
<?xml version="1.0" encoding="utf-8" ?>
<Root xmlns="http://tempuri.org/S14.xsd">
  <dimpled>dimpleda</dimpled>
  <dimpled>dimpledb</dimpled>
  <selected>selectedabc</selected>
  <unselected>unselectedabc</unselected>
  <dimpled>dimpledc</dimpled>
</Root>
```

全部(all):

用于定义元素的子元素可以以任意顺序出现至多一次。

例如：

Code 5-32

```
<complexType name="提包">
  <all>
    <element name="化妆品" type="string"/>
    <element name="钱包" type="string"/>
    <element name="手机" type="string" minOccurs="0"/>
    <element name="银行卡" type="string" minOccurs="0"/>
```


```
  </all>
</complexType>
```

组(group):

由 group 元素包含模型组的 all、choice 或 sequence 所定义的一系列元素可以作为一个原子元素来引用。格式为：

<group ref = "组名" id= "ID" minOccurs = "最少出现次数" maxOccurs = "最多出现次数"/>

例如：

Code 5-33

```
<xs:group name="myModelGroup">
  <xs:sequence>
    <xs:element ref="someThing"/>
    . . .
  </xs:sequence>
</xs:group>

<xs:complexType name="trivial">
  <xs:group ref="myModelGroup"/>
  <xs:attribute things1=""/>
</xs:complexType>

<xs:complexType name="moreSo">
  <xs:choice>
    <xs:element ref="anotherThing"/>
    <xs:group ref="myModelGroup"/>
  </xs:choice>
  <xs:attribute things2=""/>
</xs:complexType>
```

group 案例 2

笔记区

在本案例中我们定义了 2 个 string 类型的普通元素 thing1、thing2，以及定义了一个普通的 decimal 类型的属性 MyAttribute，之后定义了一个 Group 名字叫 myGroupOfThing，对两个元素进行引用，之后定义一个复杂类型 myComplexType 对组和 1 个属性进行了引用，最后声明一个名叫 root 的元素，为这个 myComplexType 类型。

Visual Studio 2008 中的设计代码

S11.xsd

Code 5-34

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema id="S11"
           targetNamespace="http://tempuri.org/S11.xsd"
           elementFormDefault="qualified"
           xmlns="http://tempuri.org/S11.xsd"
           xmlns:mstns="http://tempuri.org/S11.xsd"
           xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="thing1" type="xs:string">
  </xs:element>
  <xs:element name="thing2" type="xs:string">
  </xs:element>
  <xs:attribute name="MyAttribute" type="xs:decimal" />
  <xs:group name="myGroupOfThings">
    <xs:sequence>
      <xs:element ref="thing1" />
      <xs:element ref="thing2" />
    </xs:sequence>
  </xs:group>
  <xs:complexType name="myComplexType">
    <xs:group ref="myGroupOfThings" />
    <xs:attribute ref="MyAttribute" />
  </xs:complexType>
<xs:element name="root" type="myComplexType">
```


```
  </xs:element>
</xs:schema>
```

S11.xml

Code 5-35

```
<?xml version="1.0" encoding="utf-8" ?>
<root a:MyAttribute="MyAtt" xmlns:a="http://tempuri.org/S11.xsd">
  <thing1>tha</thing1>
  <thing2>thb</thing2>
</root>
```

attribute 元素:

作用：声明一个属性

属性：**name type ref use**

在本案例中定义了一个 TextBox 的元素，其中包括了三个属性分别是 ID 类型 string，Runat 类型 string，Text 类型 string。然后创建 Form 元素，并在其中声明对于 TextBox 元素的引用。

Visual Studio 2008 中的设计代码

S12.xsd

Code 5-36

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema id="s12" targetNamespace="http://tempuri.org/s12.xsd"
elementFormDefault="qualified" xmlns="http://tempuri.org/s12.xsd"
xmlns:mstns="http://tempuri.org/s12.xsd"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="TestBox">
    <xs:complexType>
      <xs:attribute name="ID" type="xs:string" />
      <xs:attribute name="Runat" type="xs:string" />
      <xs:attribute name="Text" type="xs:string" />
    </xs:complexType>
```

笔记区

```
  </xs:element>
  <xs:element name="Form">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="TestBox" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

S12.xml

Code 5-37

```
<?xml version="1.0" encoding="utf-8" ?>
<Form xmlns="http://tempuri.org/S12.xsd">
  <TestBox ID="TextBox1" Runat="Server" Text="Hello"/>
</Form>
```

attributeGroup 元素:

作用：把一组属性声明组合在一起，以便可以被复合类型应用

属性：**name　ref**

在本案例中我们可以将多个属性装到统一的属性组里面，属性组在我们平时可能都见过，比如在 ASP.NET 中的网页控件，所有的服务器控件基本上都少不了，这样几个属性 ID、Runat、Visible、Enable 等…这样我们就可以把这一系列属性定义到一个属性组里面，方面我们后期的反复调用。

本案例首先定义了一个属性组 myAttributeGroup 并且在其中包括两个属性名称为 someattribute1 类型 string，someattribute2 类型 integer，然后定义一个复杂类型 myElementType 引用这个属性组，最后声明 Root 元素，其中包括一个子元素名叫 SubNode 并且他的类型为 myElementType 类型。

Visual Studio 2008 中的设计代码

S13.xsd

笔记区

Code 5-38

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema id="S13" targetNamespace="http://tempuri.org/S13.xsd"
elementFormDefault="qualified" xmlns="http://tempuri.org/S13.xsd"
xmlns:mstns="http://tempuri.org/S13.xsd"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:attributeGroup name="myAttributeGroup">
    <xs:attribute name="someattribute1" type="xs:string" />
    <xs:attribute name="someattribute2" type="xs:integer" />
  </xs:attributeGroup>

  <xs:complexType name="myElementType">
    <xs:sequence />
    <xs:attributeGroup ref="myAttributeGroup" />
  </xs:complexType>

  <xs:element name="Root">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="SubNode" type="myElementType" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

笔记区

S13.xml

Code 5-39

```
<?xml version="1.0" encoding="utf-8" ?>
<Root xmlns="http://tempuri.org/S13.xsd">
  <SubNode someattribute1="s1" someattribute2="s2" />
</Root>
```

实体类型(ENTITY)

在 DTD 中有元素定义、属性声明、符号声明和实体定义，但在 XML Schema 中有前三者而无后者，不过 Schema 有从 string 派生出来的内置数据类型 ENTITY。

在 DTD 中的实体既可以是用于实例文档的通用实体，也可以是用于 DTD 本身的参数实体；既可以是解析实体，也可以是非解析实体；而且实体可以在各种语法成分中使用。

但在 Schema 中，ENTITY 类型只能在实例文档的属性值中使用，而且必须是非解析实体，当然该实体必须已经在 DTD 中被定义过。而 DTD 中的解析参数实体的功能，被 Schema 中的元素组(group)和属性组(attributeGroup)所替代。

在 Schema 中，ENTITY 类型的取值为 NCName（实际上 NCName 为 ENTITY 的基类）。可以 ENTITY 为基类来定义简单数据类型，能加在 ENTITY 类型上的约束有：length、minLength、maxLength、pattern、enumeration、whiteSpace。

ENTITY 有一个派生类型 ENTITIES，为 ENTITY 的有限非空列表，除了无 pattern 外，其他约束相同，例如：

Code 5-40

```
<xsd:element name="reference">
  <xsd:complexType>
    <!-- 匿名元素类型定义 -->
    <xsd:attribute name="sound" type="xsd:ENTITY"/>
    <!--局部属性定义 -->
    <xsd:attribute name="pic" type="xsd:ENTITY"/>
    <xsd:attribute name="data" type="xsd:ENTITIES"/>
  </xsd:complexType>
```

笔记区

```
</xsd:element>
```

使用：<reference sound=”bgsound” pic=”lena” data=”mystuff1 mystuff2 mystuff3”/>

名师提点

. Sechma 的一些优点：

它具有丰富的数据类型，支持的数据类型包括字符串、字符型、整数、浮点数和数值型、布尔型、时间型、日期型、统一资源标识（URL）、全 球唯一标识（UUID）、来自 XML 本身的类型（entity,entities, enumeration,id,idref,idrefs,nmotoken,nmotokens,notation 等等），而且还支持由这些简单类型生成的更加复杂的类型，这个类似于 C ++中结构（struct）的概念,我们可以建立一个 struct,它可以是对简单类型的一个扩展。

2. 可以由用户自己定义数据类型

3. 支持属性分组，属性的应用范围是多种多样的，有的是针对所有元素，有的则专门针对图形元素。

4. 支持名字空间。允许把文档中特殊的节点与模式中的类型说明联系起来。联系 XML 节点和 DTD 的唯一方法是通过 DOCTYPE 说明，即每一个文档只能使用一个 DTD，但是可以由多个 XML 模式来描述。

总结

本章讲解了为何需要 XML Schema 在当今的 IT 行业中，XML 被赋予了越来越多的职责

和功能，例如，作为文本数据库存储数据，作为某一行业中数据交换的标准表示，等等。这些都需要相应的 XML 文件中对数据的描述，如数据类型、长度进行严格的定义，这样才能真正做到数据的安全性和行业统一标准并有通用的规则对其进行解析。然而，XML 在其产生的时候就被定义为高开放性，用文本方式浏览，用标签来定义的标记语言，鉴于 XML 的这些特点，就像 HTML 文件一样，XML 文件本身无法对文件中的数据进行严格的定义。

为了解决以上的矛盾，XML Schema 应运而生了，它是用来定义 XML 文件的文本结构，数据类型等 XML 文件描述规则的。在 Visual Studio2008 中实现 XML Schema 编写中遇到

笔记区

的问题及 XML Schema 简单类型定义(simpleType)，复杂类型定义(complexType)等其它类型及实例文档的属性值的使用。

问与答

1．XML Schema 允许用户自己定义新的数据类型有那几种？

2．简单类型定义(simpleType)的解释？

3．复杂类型定义(complexType)的解释?

4．复杂类型定义（complexType 元素）被用来定义文档的结构，具体功能有那些请分别描述？

作业

1、分析 XML 实例，书写 Schema 文件

```
<?xml version="1.0" encoding="GB2312"?>
<学生名册>
    <学生 学号="1">
        <姓名>张三</姓名>
        <性别>男</性别>
<年龄>20</年龄>
    </学生>
    <学生 学号="2">
        <姓名>李四</姓名>
        <性别>女</性别>
        <年龄>19</年龄>
    </学生>
    <学生 学号="3">
        <姓名>王二</姓名>
        <性别>男</性别>
```

```
        <年龄>21</年龄>
    </学生>
</学生名册>
```

2、把以下 DTD 改成 XSD

```
<?xml version="1.0" encoding="GB2312"?>
<!ELEMENT 联系人 (姓名，性别， 年龄，EMAIL，电话，地址)>
<!ELEMENT 姓名 (#PCDATA)>
<!ELEMENT 性别 (#PCDATA)>
<!ELEMENT 年龄 (#PCDATA)>
<!ELEMENT EMAIL (#PCDATA)>
<!ELEMENT 电话 (#PCDATA)>
<!ELEMENT 地址 (#PCDATA)>
```

3部分

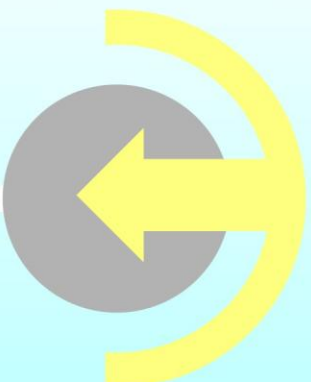
可扩展的样式表语言XSL

第六次课 Xpath

第七次课 用CSS和XSL显示XML文档

第八次课 XSL综合应用

在前一部分我们学习了两种用于xml文档结构描述的技术，它们分别是DTD和xml schema 。但是xml schema比DTD更有优势，schema支持命名空间，内置多种简单和复杂数据类型，并支持自定义数据类型。由于存在这么多的优点，所以xml schema渐渐成为xml应用的统一规范。



第六次课 Xpath

学习内容	难度
使用 XPath 的目的	★★
※Xpath 针对 XML 文档内容定义的语法	★★★
※Xpath 检索的应用	★★★
※XPath 中对元素和属性的匹配	★★★

注：带※为重点学习内容

注：带※为重点学习内容

笔记区

XPath 是W3C定义的语言和正式的W3C推荐的语言，W3C拥有XML Path Language (XPath) Version 1.0 规范。XPath 诞生于 1999 年，作为对 XSLT 和 XPointer 语言的补充，但近来已成为流行的独立语言，因为单个 XPath 表达式可用于替代多行 DOM API 代码。

使用XPath的目的：为了在匹配XML文档结构时能够准确地找到某一个节点元素。可以把 XPath 比作文件管理路径，通过文件管理路径，可以按照一定的规则查找到所需要的文件；同样，依据XPath所制定的规则，也可以很方便地找到XML结构文档树中的任何一个节点，显然这对 XSLT 来说是一个最最基本的功能。

XPath 是一种表达式语言，它的返回值可能是节点，节点集合，原子值，以及节点和原子值的混合等。在 XPath 里，有 7 中不同的节点：元素，属性，文本，名称空间，处理指令，内容，文档（根）节点，XML 文档是节点树状结构。"树根"称作文档节点（或根节点）。

6.1 Xpath 表达式

Xpath 是针对 XML 文档内容定义的语法，常用表达式如下：

表 6-1

表达式	描述
节点名称	选择所有当前节点的子节点
/	从根节点进行选择
//	选择文档中相应的节点，而不管节点在文档的何处
.	选择当前节点
..	选择当前节点的父节点
@	选择属性

Xpath 表达式示例：见下表

表 6-2

表达式	描述
学生	选择所有[学生]节点的子节点
/学生	选择[学生]的根元素

信息/姓名	选择所有在[联系人]子节点[姓名]节点下的所有元素
//学号	选择所有[学号]节点的内容，而不管节点在文档的何处
信息//姓名	选择[信息]节点内所有含有[姓名]节点的内容
//@电话	选择所有属性名为电话的属性

6.2 限定检索范围

Xpath 可以使用轴的概念来进一步限定检索的范围。

表 6-3

轴名	说明
Ancestor	选择了当前节点的所有祖先
ancestor-or-self	选择当前节点的所有祖先并且还有当前节点自己
Attribute	选择当前节点的属性
Child	选择所有当前节点的孩子
Descendant	选择所有当前节点的后代
descendant-or-self	选择当前节点的所有后代以及它本身
following-sibling	选择所有当前节点的兄弟
Namespace	选择所有当前节点的命名空间
Parent	选择当前节点的父亲
preceding-sibling	选择所有当前节点之前的兄弟
Self	选择当前节点

限定检索范围示例

表 6-4

轴名	说明

笔记区

ancestor::信息	选择了当前祖先节点为信息的节点
ancestor-or-self::信息	当前节点和祖先节点为[信息]的节点
attribute::学号	选择当前节点下所有属性为学号的内容
Attribute::*	选择当前节点所有的子节点
Child::信息	选择所有当前节点下所有为[信息]的子节点
Child::*	选择当前节点下的所有子节点
Child::*/child::姓名	当前节点所有含[姓名]的孙子节点
Child::text()	选择当前节点所有子节点的文字
Child::node()	选择所有当前节点的子节点
descendant::信息	选择所有当前节点为[信息]的后代节点

使用函数来设置检索条件

XPath 中的函数用来指定检索的节点所满足的条件，被嵌入在中括号内。如下表

表 6-5

函数	说明
/信息/姓名[1]	[信息]里的第一个[姓名]节点
/信息/姓名[last()]	[信息]里的最后一个[姓名]节点
/信息/姓名[last()-1]	[信息]里的倒数第二个[姓名]节点
/信息/姓名[position()<5]	在[信息]里的前四个[姓名]节点
//信息[@学号]	所有含有学号属性的[联系人]节点
//信息[@id='1000']	所有含有学号属性且值为'1000'的[信息]节点
/信息/地址[城市='武汉']	选择所有[信息]中[地址]节点中的[城市]为'武汉'的[信息]节点

使用运算符设置检索条件

请参考本书前面关于 XML 的相关章节，使用通配符进行模糊查询。

结合上面的函数，Xpath 还可以使用通配符来模糊查询 XML 元素，见下表。

表 6-6

通配符	说明

*	匹配的所有元素节点
@*	匹配的所有属性节点
Node()	匹配任何类型的节点

Xpath 通配符示例

表 6-7

通配符	说明
//信息/*	选择所有[信息]的节点
//*	选择文档中的所有元素
//信息[@*]	选择所有含有属性的节点

6.3 XPath 中对元素和属性的匹配

XPath 中对元素和属性的匹配，主要有以下几种：

- 定位节点
- 选择未知元素
- 选择分支
- 选择属性

1、 定位节点

每个 XML 文档都可看成是一棵树，该树与计算机中的树形文件夹非常类似，XPath 使用以斜线分隔的子元素名的列表来描述某个 XML 文档的路径所匹配的元素。

我们通过下面这个 student.xml 文件来讲解，Xpath 是如何定位节点的。

Code 6-1

```
<?xml version="1.0" encoding="UTF-8"?>
<学院>
  <班级 id="N2D07020">
    <学生 学号="051010">
      <姓名>张三</姓名>
      <性别>男</性别>
```

笔记区

```
      <电话>13904546451</电话>
      <地址>
        <城市>黄石</城市>
        <街道>武汉市洪山区新路村 1 号</街道>
      </地址>
    </学生>
    <学生 学号="051011">
      <姓名>李四</姓名>
      <性别>男</性别>
      <电话>15904546451</电话>
      <地址>
        <城市>武汉</城市>
        <街道>武汉市洪山区新路村 1 号</街道>
      </地址>
    </学生>
  </班级>
  <班级 id="N2D07001">
    <学生 学号="051010">
      <姓名>小兵</姓名>
      <性别>男</性别>
      <电话>13304546451</电话>
      <地址>
        <城市>孝感</城市>
        <街道>武汉市洪山区新路村 1 号</街道>
      </地址>
    </学生>
    <学生 学号="051011">
      <姓名>张芳</姓名>
      <性别>女</性别>
      <电话>15304546451</电话>
```

```
      <地址>
        <城市>武汉</城市>
        <街道>武汉市洪山区新路村 1 号</街道>
      </地址>
    </学生>
  </班级>
</学院>
```

基本的 XPath 语法类似于在一个文件系统中定位文件，如果路径以斜线 / 开始， 那么该路径就表示一个元素的绝对路径

[/学院] 选择根元素学院：

```
<学院>
<班级 id ="N2D07020">…</班级>
<班级 id ="N2D07001">…</班级>
<学院>
```

注意：这里<班级 id ="N2D07020">…</班级>中"…"代表部分内容省略，其中详细内容请参见前面完整文档内容，如在后后面内容碰到类似结构，意义相同。

[/学院/班级] 选择学院下的所有班级元素：

```
<班级 id ="N2D07020">…</班级>
<班级 id ="N2D07001">…</班级>
```

[/学院/班级/学生] 选择学院下的所有班级元素中的所有学生子元素：

```
<学生 学号="051010">…</学生>
<学生 学号="051011">…</学生>
<学生 学号="051010">…</学生>
<学生 学号="051011">…</学生>
```

如果路径以双斜线 // 开头， 则表示选择文档中所有满足双斜线//之后规则的元素(无论层级关系)，文档任何位置都可以查询出来。

[//班级] 选择所有班级元素：

```
<班级 id ="N2D07020">…</班级>
<班级 id ="N2D07001">…</班级>
```

笔记区

[//班级/学生] 选择所有班级元素下的子元素学生：

```
<学生 学号="051010">…</学生>
<学生 学号="051011">…</学生>
<学生 学号="051010">…</学生>
<学生 学号="051011">…</学生>
```

2、 选择未知元素

星号 * 表示选择所有由星号之前的路径所定位的元素

[/学院/班级/*] 选择所有路径依附于/学院/班级 的元素：

```
<学生 学号="051010">…</学生>
<学生 学号="051011">…</学生>
<学生 学号="051010">…</学生>
<学生 学号="051011">…</学生>
```

[/*/*/姓名] 选择有两个祖先元素的姓名元素：

```
<姓名>张三</姓名>
<姓名>李四</姓名>
<姓名>小兵</姓名>
<姓名>张芳</姓名>
```

[//*] 选择所有元素：

```
<学院>
<班级 id ="N2D07020">…</班级>
<班级 id ="N2D07001">…</班级>
<学院>
```

属性的值可以被用来作为选择的准则， normalize-space 函数删除了前部和尾部的空格， 并且把连续的空格串替换为一个单一的空格。

[//学生[@学号='051011']] 匹配所有学号等于 051011 的学生：

```
<学生 学号="051011">…</学生>
<学生 学号="051011">…</学生>
```

3、 选择分支

方块号里的表达式可以进一步的指定元素， 其中数字表示元素在选择集里的位置， 而last()函数则表示选择集中的最后一个元素。

[/学院/班级[1]] 选择学院下的班级中的第一个元素：

```
<班级 id ="N2D07020">…</班级>
```

[/学院/班级[last()]] 选择学院下的班级中的最后一个元素：

```
<班级 id ="N2D07001">…</班级>
```

选择多个路径

通过在 XPath 语句中使用“|”操作符来选择多个路径。

[//姓名|//性别] 查询所有的姓名或性别节点：

```
<姓名>张三</姓名>
<性别>男</性别>
<姓名>李四</姓名>
<性别>男</性别>
<姓名>小兵</姓名>
<性别>男</性别>
<姓名>张芳</姓名>
<性别>女</性别>
```

[//姓名|/学院/班级/学生/性别] 查询所有的姓名或学院/班级/学生下面的性别元素：

```
<姓名>张三</姓名>
<性别>男</性别>
<姓名>李四</姓名>
<性别>男</性别>
<姓名>小兵</姓名>
<性别>男</性别>
<姓名>张芳</姓名>
<性别>女</性别>
```

4、 选择属性

在 XPath 语法中，要获得属性信息必须以前缀“@”来指定

[//@学号] 选择所有的学号属性：

笔记区

```
学号="051010"
学号="051011"
学号="051010"
学号="051011"
```

[/班级/学生[@学号]] 选择所有有学号属性的学生：

```
<学生 学号="051010">…</学生>
<学生 学号="051011">…</学生>
<学生 学号="051010">…</学生>
<学生 学号="051011">…</学生>
```

[/班级/学生[@*]] 选择所有有属性的学生：

```
<学生 学号="051010">…</学生>
<学生 学号="051011">…</学生>
<学生 学号="051010">…</学生>
<学生 学号="051011">…</学生>
```

XPath 辅助学习工具制作：

本案例主要能实现使用 xpath 语句在指定文档中查找对应元素并显示

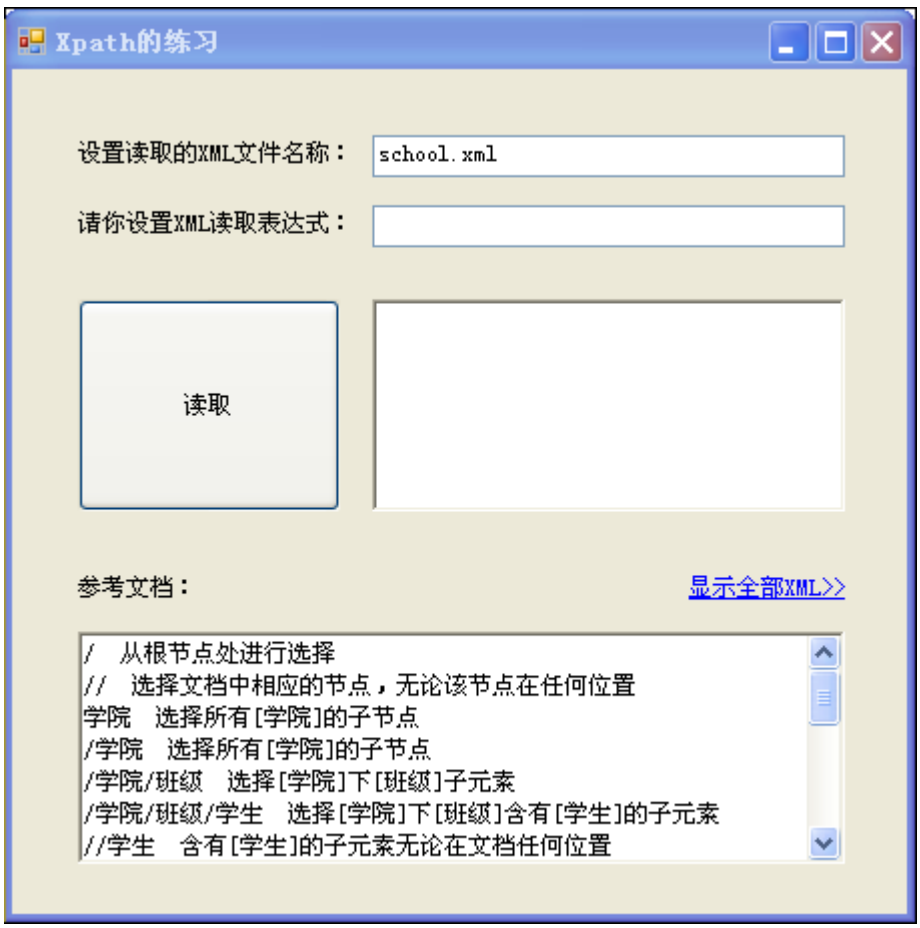


图 **6-1**

Code 6-2

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Xml;
using System.IO;
```

笔记区

```
namespace XPath
{
    public partial class Form1 : Form
    {
//添加 xpath 基本语法到 richTextBox2
public Form1()
    {
            InitializeComponent();
            richTextBox2.Text = "/  从根节点处进行选择"+"\r";
            richTextBox2.AppendText("//  选择文档中相应的节点，无论该节点在任何位置"+"\r");
            richTextBox2.AppendText("学院  选择所有[学院]的子节点" + "\r");
            richTextBox2.AppendText("/学院  选择所有[学院]的子节点" + "\r");
            richTextBox2.AppendText("/学院/班级  选择[学院]下[班级]子元素" + "\r");
            richTextBox2.AppendText("/学院/班级/学生  选择[学院]下[班级]含有[学生]的子元素" + "\r");
            richTextBox2.AppendText("//学生  含有[学生]的子元素无论在文档任何位置" + "\r");
            richTextBox2.AppendText("//@学号  选择含有属性名为学号的属性" + "\r");
            richTextBox2.AppendText("/学院/班级[2]  选择学院里第二个班级" + "\r");
            richTextBox2.AppendText("/学院/班级[2]/学生[last()]  选择学院里第二个班级中最后一个学生" + "\r");
            richTextBox2.AppendText("/学院/班级[2]/学生[last()-1]  选择学院里第二个班级中倒数第二个学生" + "\r");
          richTextBox2.AppendText("/学院/班级[2]/学生[position()>1]  选择学院里第二个班级中第二个学生" + "\r");
```

```
            richTextBox2.AppendText("//学生[@学号='051012']      选择学院里学
号为 051012 的学生" + "\r");
richTextBox2.AppendText("/学院/班级/学生/地址[城市='孝感']      选择城市为孝感的
地址节点信息" + "\r");
        }

        private void button1_Click(object sender,  EventArgs e)
        {
            //创建 XML 文档对象
            XmlDocument doc = new XmlDocument();
            //从当前程序运行的目录中，加载 textBox1 中的文件
            doc.Load(AppDomain.CurrentDomain.BaseDirectory +
textBox1.Text);
            try
            {
                //通过 XML 文档对象，查询单个节点的方法查找 textBox2 的 XPath
                XmlNode xn = doc.SelectSingleNode(textBox2.Text);
                //将查找到的节点中的内容通过 richTextBox1 显示出来
                richTextBox1.Text = xn.OuterXml;
            }
            catch
            {
                //如果 textBox2 中的 xpath 为找到指定节点，则会产生异常
                MessageBox.Show("节点未找到或 XPath 语法错误!");
            }
        }
}
}
```

笔记区

总结

本章我们了解到了：

- 路径表达式语法
- 相对/绝对路径
- 表达式上下文
- 谓词（筛选表达式）及轴的概念
- 运算符及特殊字符
- 常用表达式实例
- 函数及说明

这里给出一个实例 Xml 文件。下面的说明及实例都是基于该 XML 文件。

路径表达式语法：

1．路径 = 相对路径 | 绝对路径

2．路径表达式 = 步进表达式 | 相对路径 "/"步进表达式。

3．步进表达式=轴 节点测试 谓词

说明：

其中轴表示步进表达式选择的节点和当前上下文节点间的树状关系（层次关系），节点测试指定步进表达式选择的节点名称扩展名，谓词即相当于过滤表达式以进一步过滤细化节点集。谓词可以是 0 个或多个。多个多个谓词用逻辑操作符 and， or 连接。取逻辑非用 not()函数。XML 中的查询语言 XPath 表达式的语法规则，XPath 中对元素和属性的匹配四种方式：定位节点、选择未知元素、选择分支、选择属性。

问与答

1、 使用 XPath 的目的？

2、 XPath 中对元素和属性的匹配，主要有那几种?请分别讲述.

作业

练习题一：完成上课案例

第七次课 用 CSS 和 XSL 显示 XML 文档

本章内容	难度
Xsl 基础	★★
用 css 显示 xml 文档	★★
※ XSL 的基本语法	★★★

注：带※为重点学习内容

笔记区

在前面的章节中,我们知道了怎样去写一个结构良好的文档和有效的文档，但这些 XML 文档怎样让一般的用户可以看得懂。本章关注 XSLT（可扩展的样式表语言转换），可以用于转换 XML 并生成网页在浏览器中显示。W3C 将 XSLT 定义为一种将 XML 文档转换为其他 XML 文档的语言。

7.1 XSL 基础

XSL(eXtensible Stylesheet Language)是显示 XML 文档的首选样式语言，它比 CSS 更适合于 XML。

CSS = HTML 样式表：

HTML 使用预先定义的标签，标签的意义很容易被理解。HTML 元素中的 <table> 元素定义表格 - 并且浏览器知道如何显示它。向 HTML 元素添加样式是很容易的。通过 CSS，很容易告知浏览器用特定的字体或颜色显示一个元素。

XSL = XML 样式表：

XML 不使用预先定义的标签（我们可以使用任何喜欢的标签名），并且这些标签的意义并不都那么容易被理解。<table> 元素意味着一个 HTML 表格，一件家具，或是别的什么东西 - 浏览器不清楚如何显示它。XSL 可描述如何来显示 XML 文档！

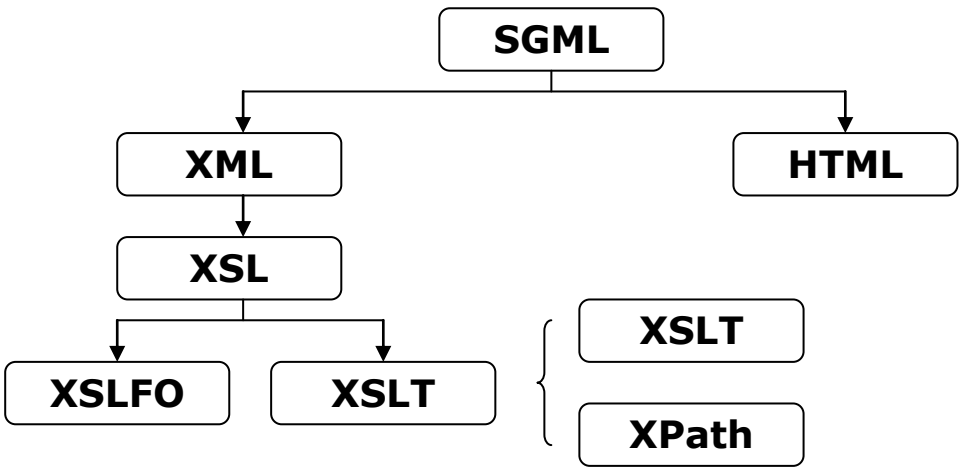
XSL - 不仅仅是样式表语言，XSL 包括三部分：

XSLT：一种用于转换 XML 文档的语言。

XPath：一种用于在 XML 文档中导航的语言。

XSL-FO：一种用于格式化 XML 文档的语言。

XSL、XSLT 、XSL FO 和 XPath 概述：



什么是 XSLT

- XSLT 指 XSL 转换（XSL Transformations）。
- XSLT 是 XSL 中最重要的部分。
- XSLT 可将一种 XML 文档转换为另外一种 XML 文档。
- XSLT 使用 XPath 在 XML 文档中进行导航。

XSLT 用于将一种 XML 文档转换为另外一种 XML 文档，或者可被浏览器识别的其他类型的文档，比如 HTML 和 XHTML。通常，XSLT 是通过把每个 XML 元素转换为 (X)HTML 元素来完成这项工作的。

什么是 XPath

- XPath 使用路径表达式在 XML 文档中进行导航
- XPath 包含一个标准函数库
- XPath 是 XSLT 中的主要元素
- XPath 是一个 W3C 标准

XPath 是一门在 XML 文档中查找信息的语言。XPath 用于在 XML 文档中通过元素和属性进行导航。

什么是 XSL-FO

- XSL-FO 是用于格式化 XML 数据的语言
- XSL-FO 指可扩展样式表语言格式化对象（Extensible Stylesheet Language Formatting Objects）
- XSL-FO 是一个 W3C 推荐标准

笔记区

- XSL-FO 目前通常被称为 XSL

为什么会存在这样的混淆呢？XSL-FO 和 XSL 是一回事吗？可以这么说，不过我们需要向您作一个解释：样式化（Styling）是关于转换信息和格式化信息两方面的信息。在万维网联盟编写他们的首个 XSL 工作草案的时候，这个草案包括了有关转换和格式化 XML 文档的语言语法。后来，XSL 工作组把这个原始的草案分为独立的标准：

- XSLT，用于转换 XML 文档的语言
- XSL 和 XSL-FO，用于格式化 XML 文档的语言
- XPath，是通过元素和属性在 XML 文档中进行导航的语言

CSS 与 XSL 对比

1、 CSS

优点：编写与学习简单

缺点：但是功能有限，需要浏览器支持(NS√ IE×)，不能表现属性（只能表现元素内容），不能添加其他显示内容，不能实现条件/选择处理。

2、 XSL

优点：功能强大，不需要浏览器的专门支持（如处理成 HTML 后再传给客户端），可为输出添加元素和内容，能用任何语言输出，可使用封装或用户自定义函数，可使用条件处理、能排序/过滤后再输出，能实现复杂的页面布局和样式。

缺点：复杂、对 XSL-FO 有争议（如可用 CSS+DOM 替代 XSLT+XSL-FO）

7.2 用 CSS 显示 xml 文档

1、 嵌入 HTML 文档

使用 STYLE 元素（标签）:可以使用样式 STYLE 标签将 CSS 的定义嵌入 HTML 文档的头部

Code 7-1

```
<HEAD>
<TITLE>标题串</TITLE>
… …
    <STYLE type="text/css">若干 CSS 定义</STYLE>
</HEAD>
```

使用 STYLE 属性: 也可以在 HTML 文档元素的开始标签中，使用样式 STYLE 属性来定义元

素内容的表现方式

<标签名 style="属性名：属性值; 属性名：属性值;...">

Test1.html

Code 7-2

```
<HTML>
  <HEAD><TITLE>title</TITLE>
    <STYLE type="text/css"> H1 { color: blue } </STYLE></HEAD>
  <BODY>
    <H1>这是一号字,字体的颜色蓝色</H1><P style="color: green">这是一句话的字体颜色为绿色的.</P><P>这是一句话的字体颜色为黑色的.</P>
  </BODY>
</HTML>
```

显示为：

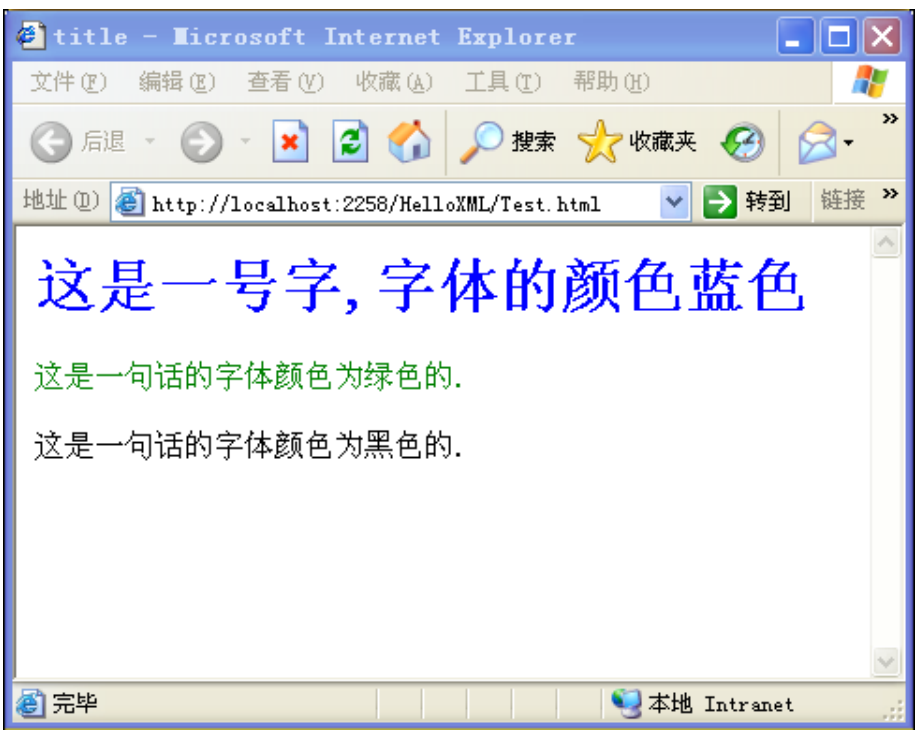


图 7-1

2、 独立 CSS 文件

笔记区

可以在独立 CSS 文件*.css 中定义好 CSS 后，然后再在 HTML 或 XML 文档中引用：

HTML:在 HTML 文档的头部 HEAD 用链接标签引用

<LINK rel="stylesheet" type="text/css" href="URL">

css1.css

Code 7-3

```
H1 {color: blue}
P {color: green}
```

Test2.html

Code 7-4

```
<HTML>
  <HEAD>
<TITLE>title</TITLE>
<LINK rel="stylesheet" type="text/css" href="css1.css">
  </HEAD>
  <BODY>
<H1>这是一号字,字体的颜色蓝色</H1>
    <P>这是一句话的字体颜色为绿色的.</P>
<P>这是一句话的字体颜色为黑色的.</P>
  </BODY>
</HTML>
```

显示结果同图 7-1。

3、 CSS 在 XML 中的应用

在 XML 文档的文档类型声明之后，用 xml-stylesheet 声明来引用所定义的 CSS：

```
<?xml-stylesheet type="text/css" href="URL" ?>
```

C01.css

Code 7-4

```
xsampdoc
{
    background-color: #99ffff;
```

笔记区

```
    width: 250px;
    height:200px;
}
greeting
{
    background-color:Blue;
     width: 250px;
     height: 80px;
}
string
{
background-color:Yellow;
    width: 50px;
    height: 50px;
}
answer
{
    background-color:Gray;
width: 250px;
    height: 120px;
}
extension
{
    background-color:Aqua;
    width: 80px;
    height: 120px;
    float:left;
}
```

C01.xml

Code 7-5

笔记区

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/css" href="C01.css"?>
<xsampdoc>
  <greeting><string> 你好!银河</string></greeting>
  <answer class="ans">
    <extension>早上好! </extension>
    <question> 好!我很好</question>
  </answer>
</xsampdoc>
```

显示结果为：

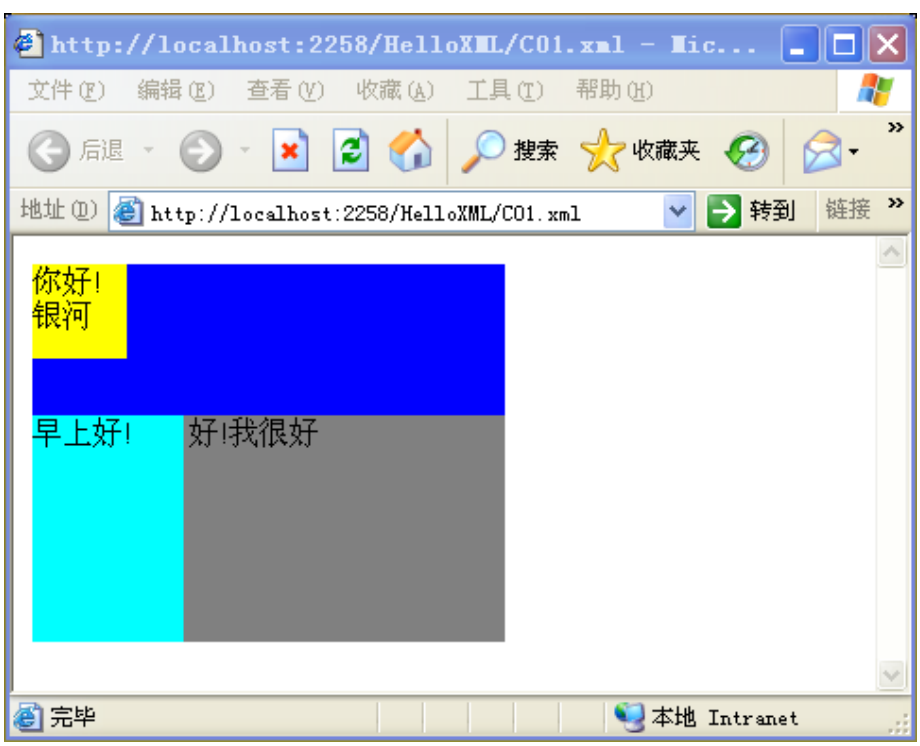


图 7-2

7.3 XSL 的基本语法

- XSL 基本结构
- XSL 模板

- 节点选择语句<xsl:value-of>
- 循环判断语句<xsl:for-each>
- 条件判断语句<xsl:if>
- 多条件判断语句<xsl:choose>
- 排序语句<xsl:sort>

1、 XSL 基本结构

Code 7-6

```
<?xml version="1.0" encoding="gb2312">
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <!--模板规则-->
    <!--输出模板-->
</xsl:stylesheet>
```

XSL 声明、根元素：

<xsl:stylesheet>　　</xsl:stylesheet>

XSL 的命名空间：

xmlns:xsl="http://www.w3.org/1999/XSL/Transform"

2、 XSL 模板

模板是在 XSL 中非常重要的概念,模板在 XSL 中的应用,有点类似于面向对象编程语言中的方法,它可以使用 XPath 语句选定匹配的 XML 元素并对其进行处理,也可以在其他地方对其调用.类似于程序设计中的模块化的概念,在一个 XSL 文件中至少有一个模板.

模板的定义语法,在 XSL 中模板一般会有两个元素,<xsl:template>定义模板和<xsl:apply-templates>应用模板，定义模板：

```
<xsl:template match=”匹配模式 XPath 语句”>
    模板内容
</xsl:template>
```

应用模板：

```
<xsl:apply-templates select=”匹配模式” />
```

那我们就先来看个案例吧！看看 XSL 具体是怎么用的。下表是 xml 通过 xsl 样式后的显示效果：

笔记区

姓名	性别	生日	分数	技能
李华	男	1978.9.12	92	JavaOracleC SharpSQL Server
倪冰	女	1979.1.12	89	Visual BasicSQL ServerASP
张君宝	男	1982.9.9	98	C SharpSQL ServerUML
杨惠	女	1980.5.16	85	Visual C++SQL ServerUML
崔春晓	男	1981.4.19	86	UMLC SharpXMLSQL Server

图 7-3

C02.xml

Code 7-7

```
<?xml version="1.0" encoding="gb2312"?>
<?xml-stylesheet href="C02.xsl" type="text/xsl"?>
<roster>
  <student ID="101">
    <name>李华</name>
    <sex>男</sex>
<birthday>1978.9.12</birthday>
    <score>92</score>
    <skill>Java</skill>
    <skill>Oracle</skill>
    <skill>C Sharp</skill>
    <skill>SQL Server</skill>
  </student>
<student ID="102">
    <name>倪冰</name>
    <sex>女</sex>
    <birthday>1979.1.12</birthday>
    <score>89</score>
    <skill>Visual Basic</skill>
    <skill>SQL Server</skill>
```

```
    <skill>ASP</skill>
  </student>
 <student ID="103">
    <name>张君宝</name>
    <sex>男</sex>
    <birthday>1982.9.9</birthday>
    <score>98</score>
    <skill>C Sharp</skill>
    <skill>SQL Server</skill>
    <skill>UML</skill>
  </student>
<student ID="104">
    <name>杨惠</name>
    <sex>女</sex>
    <birthday>1980.5.16</birthday>
    <score>85</score>
<skill>Visual C++</skill>
<skill>SQL Server</skill>
    <skill>UML</skill>
  </student>
 <student ID="105">
<name>崔春晓</name>
<sex>男</sex>
    <birthday>1981.4.19</birthday>
    <score>86</score>
    <skill>UML</skill>
<skill>C Sharp</skill>
    <skill>XML</skill>
    <skill>SQL Server</skill>
</student>
```

笔记区

```
</roster>
```

C02.XSL

Code 7-8

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
    <html>
    <body>
      <table border="1">
        <tr>
          <td>姓名</td>
          <td>性别</td>
          <td>生日</td>
          <td>分数</td>
          <td>技能</td>
        </tr>
        <xsl:apply-templates select="/roster/student"/>
      </table>
    </body>
    </html>
</xsl:template>
  <xsl:template match="/roster/student">
    <tr>
      <td>
        <xsl:value-of select="name"/>
      </td>
      <td>
        <xsl:value-of select="sex"/>
```

笔记区

```
      </td>
      <td>
        <xsl:value-of select="birthday"/>
      </td>
      <td>
        <xsl:value-of select="score"/>
      </td>
      <td>
        <xsl:apply-templates select="skill"/>
      </td>
    </tr>
  </xsl:template>
  <xsl:template match="skill">
    <xsl:value-of select="." />
  </xsl:template>
</xsl:stylesheet>
```

在这里案例中我们可以看到在第一行 xml 声明之后，xsl:stylesheet 的声明和 XSL 的命名空间："http://www.w3.org/1999/XSL/Transform"，再紧接着是模板定义语句<xsl:template match="/">match 属性表示，当前模板匹配 xml 文档的根节点，并且在这个模板中定义了 html 的基本页面结构，以及在其中定义了一个表格的结构，在表格中行的地方有一条语句<xsl:apply-templates select="/roster/student"/>表示应用一个模板，这个模板匹配/roster/student 这个 XPath 语句。然后在后面定义了这个匹配的模板<xsl:template match="/roster/student">并将 student 的子元素的值都读取出来了<xsl:value-of select="sex"/>。在原 xml 文档中 skill 元素出现次数比较多所以单独为他做了一个模板，获取其中数据。

再来看一个案例：

C03.xml

Code 7-9

```
<?xml version="1.0" encoding="gb2312"?>
<?xml-stylesheet type="text/xsl" href="C03.xsl"?>
```

笔记区

```
<唐诗>
  <五言绝句>
    <作者>李白</作者>
    <标题>玉阶怨</标题>
    <内容>
      玉阶生白露，夜久侵罗袜。
      却下水晶帘，玲珑望秋月。
    </内容>
  </五言绝句>
  <五言绝句>
    <作者>白居易</作者>
    <标题>问刘十九</标题>
    <内容>
      绿蚁新醅酒，红泥小火炉。
      晚来天欲雪，能饮一杯无。
    </内容>
  </五言绝句>
</唐诗>
```

C03.xsl

Code 7-10

```
<?xml version="1.0" encoding="gb2312"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/唐诗/五言绝句">
    <html>
    <body>
        <xsl:apply-templates select="作者"/>
        <xsl:apply-templates select="标题"/>
    </body>
```


```
    </html>
</xsl:template>
  <xsl:template match="作者">
    <xsl:value-of select="."/>
    <br></br>
  </xsl:template>
<xsl:template match="标题">
    <xsl:value-of select="."/>
    <hr></hr>
  </xsl:template>
</xsl:stylesheet>
```

这个案例和上一个比较类似，可以类比理解，到这里我对 XSL 如何显示 XML 中的内容有些了解了，后面我们会更加深入。

3、 节点选择语句<xsl:value-of>

value-of 可以用来取出 XML 文件中被选择的元素或属性的内容，语法：

<xsl:value-of select=".”/>

输出当前节点及其所有后继节点的取值

<xsl:value-of select="匹配模式"/>

输出指定节点的取值，用 select 属性进行限定。

C04.xml

Code 7-11

```
<?xml version="1.0" encoding="gb2312"?>
<?xml-stylesheet type="text/xsl" href="C04.xsl"?>
<唐诗>
  <五言绝句>
    <作者>李白</作者>
    <标题>玉阶怨</标题>
    <内容>
```

笔记区

```
        玉阶生白露，夜久侵罗袜。
        却下水晶帘，玲珑望秋月。
    </内容>  </五言绝句>  <五言绝句>
    <作者>白居易</作者>
    <标题>问刘十九</标题>
    <内容>
        绿蚁新醅酒，红泥小火炉。
        晚来天欲雪，能饮一杯无。
    </内容>
  </五言绝句>
</唐诗>
```

C04.xsl

Code 7-12

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
    <html>
    <body>
      <xsl:value-of select="."/>
    </body>
    </html>
</xsl:template>
</xsl:stylesheet>
```

预览显示效果：

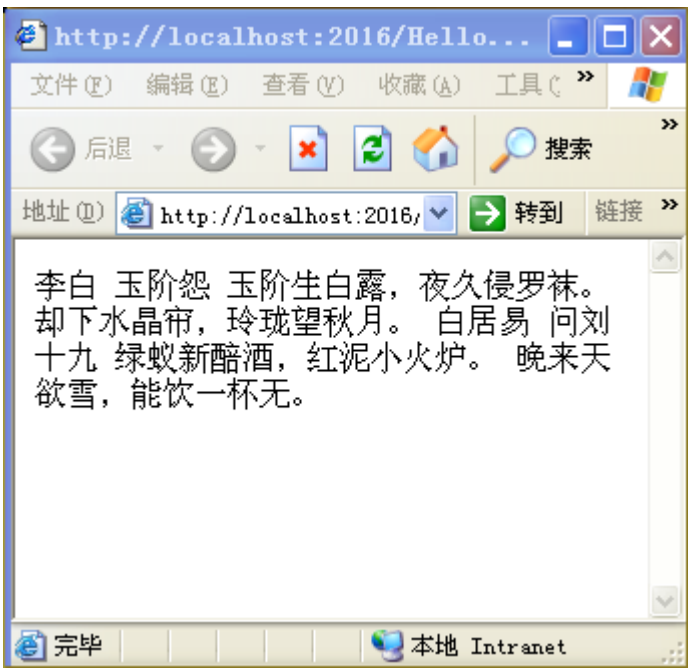


图 7-4

在这个例子中我们看到在<xsl:template match="/">模板中直接使用<xsl:value-of select="."/>的结果就是将根元素下所有子元素中的值都给查询出来了。接下来我们再来看看<xsl:value-of select="匹配模式"/>的用法。

C05.xsl

Code 7-13

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
    <html>
    <body>
      <xsl:value-of select="唐诗/五言绝句[2]/作者"/>
    </body>
    </html>
</xsl:template>
</xsl:stylesheet>
```

笔记区

显示效果：

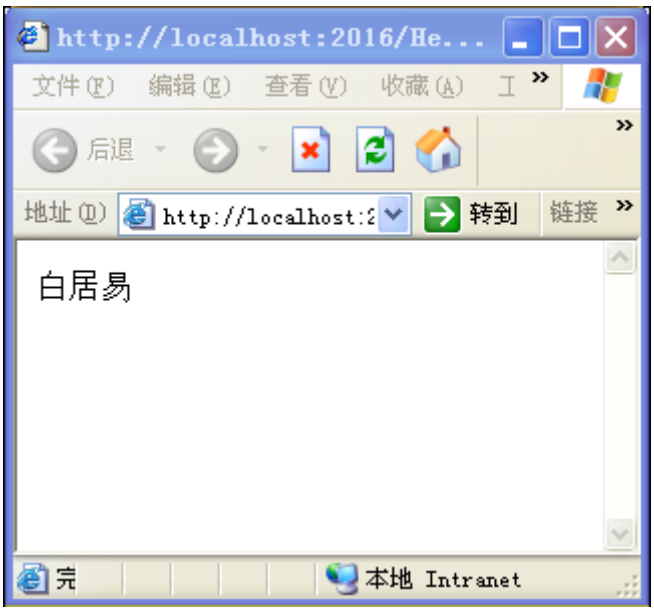


图 7-5

从这里大家可以发现“.”和“匹配模式”用法的区别，匹配模式可以匹配 XPath 查到的节点中的内容。

XSL 将属性视为属于 XML 文件中的一个元素，好像是 XML 文件中的子元素。然而，要在 XSL 中读取属性，你必须在属性名称前加上字符@，作为属性名称与元素名称的区别。

<xsl:value-of select=”@属性名” />

新闻列表案例：

C12.xml

Code 7-14

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="C12.xsl"?>
<mop>
  <type typeid="1">
    <base id="1" title="爱上一次成像的简单快乐"/>
    <base id="11" title="十一月主题活动：《十一月的橙色》"/>
    <base id="10" title="公告活动【总索引帖】新新火星人进"/>
    <base id="12" title="给新手免费玩 LOMO 交换平台诞生"/>
```

```
    <base id="13" title="LOMO 相机资料大全(最新更新 holga 系列)"/>
    <base id="14" title="另类摄影潮流 lomo 反传统的新傻瓜"/>
    <base id="15" title="LOMOChina 曾经的一些主题"/>
    <base id="16" title="LOMOChina.com 桌面上线,欢迎下载"/>
    <base id="17" title="超级金牌吾五组的幸福生活..."/>
    <base id="18" title="热点讨论越来越虚伪的 LOMO"/>
  </type>
</mop>
```

C12.xsl

Code 7-15

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/mop">
    <html> <body>
      <table border="1">
        <tr>
          <td>
            <xsl:for-each select="type[@typeid=1]/base">
              <a href="NewsDetails.aspx?nid={@id}">
                <xsl:value-of select="@title"/>
              </a><br></br>
            </xsl:for-each>
          </td>
        </tr>
      </table>
    </body> </html>
</xsl:template>
</xsl:stylesheet>
```

演示效果:

笔记区

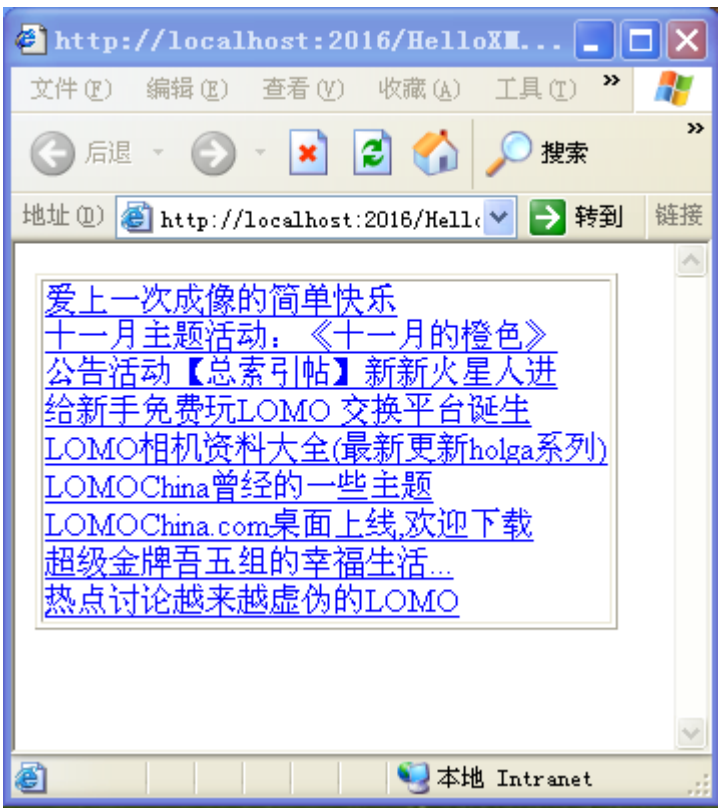


图 7-6

本案例是用 XML 制作站点的基础案例，本案演示了如何通过 XSL 显示超链接。读者可以将多个新闻板块生成不同的 XML 内容，然后结合对应的 XSL 就可以生成，基于 XML 技术的网站。

4、 循环判断语句<xsl:for-each>

<for-each>语句可以循环选择多条数据，这个语句的作用类似于 C#语言中的 foreach 语句可以循环遍历 XML 文档中的的数据。语法：

```
<xsl:for-each select="XPath" >
        <!--输出内容-->
</xsl:for-each>
```

下面我们来看一个语句的 for-each 输出的案例：

C06.xml

Code 7-16


```
<?xml version="1.0" encoding="big5"?>
<?xml-stylesheet href="C06.xsl" type="text/xsl"?>
<唐詩>
<五言絕句>
    <作者>柳宗元</作者>
    <標題>江雪</標題>
    <內容>
      千山鸟飞绝，万径人踪灭。孤舟蓑笠翁，独钓寒江雪。
    </內容>
  </五言絕句>
  <五言絕句>
    <作者>西鄙人</作者>
    <標題>哥舒歌</標題>
    <內容>
      北斗七星高，哥舒夜带刀。至今窥牧马，不敢过临洮。
    </內容>
  </五言絕句>
  <五言絕句>
    <作者>李频</作者>
    <標題>渡汉江</標題>
    <內容>
      岭外音书绝，经冬复立春。近乡情更怯，不敢问来人。
    </內容>
  </五言絕句>
</唐詩>
```

C06.xsl

Code 7-17

笔记区

```
<?xml version="1.0" encoding="big5"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"><xsl:template
match="/">
<html><body>
      <xsl:for-each select="唐詩/五言絕句">
        <p style="color:blue"><xsl:value-of select="作者"/>
        <xsl:value-of select="標題"/></p>
       <p><xsl:value-of select="內容"/></p>
      </xsl:for-each>
    </body></html>
</xsl:template>
</xsl:stylesheet>
```

显示效果：

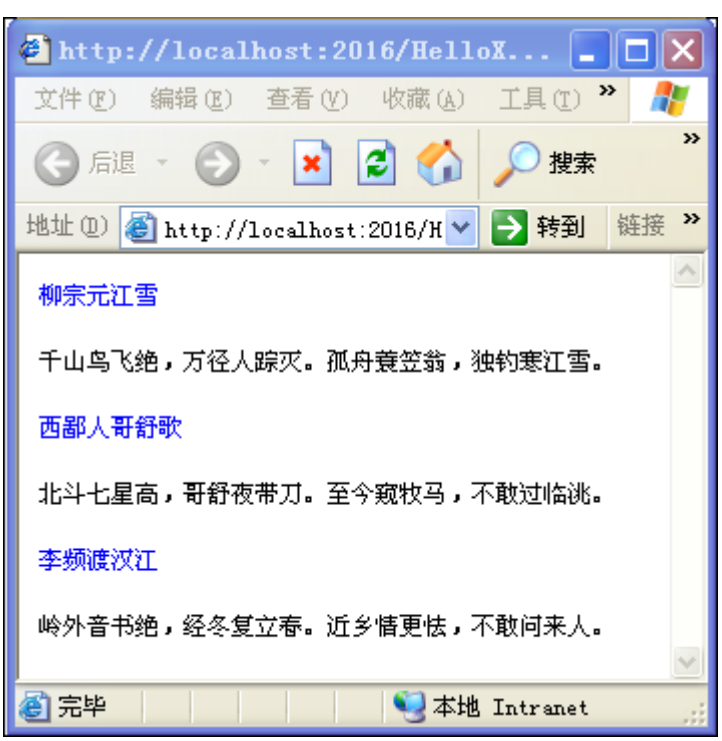


图 7-7

在本案例中名为"五言絕句"元素有 3 个,这样就构成了使用 for-each 语句循环的条件，然后在每次循环中把"五言絕句"元素中的子元素"作者"，" 標題"，" 內容"的值取出显示。

注意：使用循环遍历数据的方式除了使用 for- each 语句外还可以使用前面课程所学的模板来实现如下：

Code 7-19

```
<?xml version="1.0" encoding="big5"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
    <html><body>
      <xsl:apply-templates select="唐詩/五言絕句" />
</body></html>
</xsl:template>
<xsl:template match="唐詩/五言絕句">
  <p style="color:blue">
    <xsl:value-of select="作者"/>
    <xsl:value-of select="標題"/>
  </p>
  <p>
    <xsl:value-of select="內容"/>
  </p>
  </xsl:template>
</xsl:stylesheet>
```

输出结果和前面一样这里就省略过去，在这里定义模板的 match 中的 xpath"唐詩/五言絕句"是可以匹配所有根节点下的"五言絕句"元素，当应用模板时，这就无形的起到了循环的作用。

foreach 案例 2（排序）

可以在循环语句中添加排序功能,使用 xsl:sort 语句：

<xsl:sort select="排序的元素" order="排序方向"/>排序方向"descending" 降序，"ascending" 升序。

笔记区

C06.3.xsl

Code 7-20

```
<?xml version="1.0" encoding="big5"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
    <html> <body>
   <xsl:for-each select="唐詩/五言絕句">
        <xsl:sort select="作者" order="descending"/>
        <p style="color:blue">
          <xsl:value-of select="作者"/>
          <xsl:value-of select="標題"/>
        </p>
        <p>
          <xsl:value-of select="內容"/>
        </p>
 </xsl:for-each>
    </body></html>
</xsl:template>
</xsl:stylesheet>
```

当前为降序（descending）显示效果

笔记区

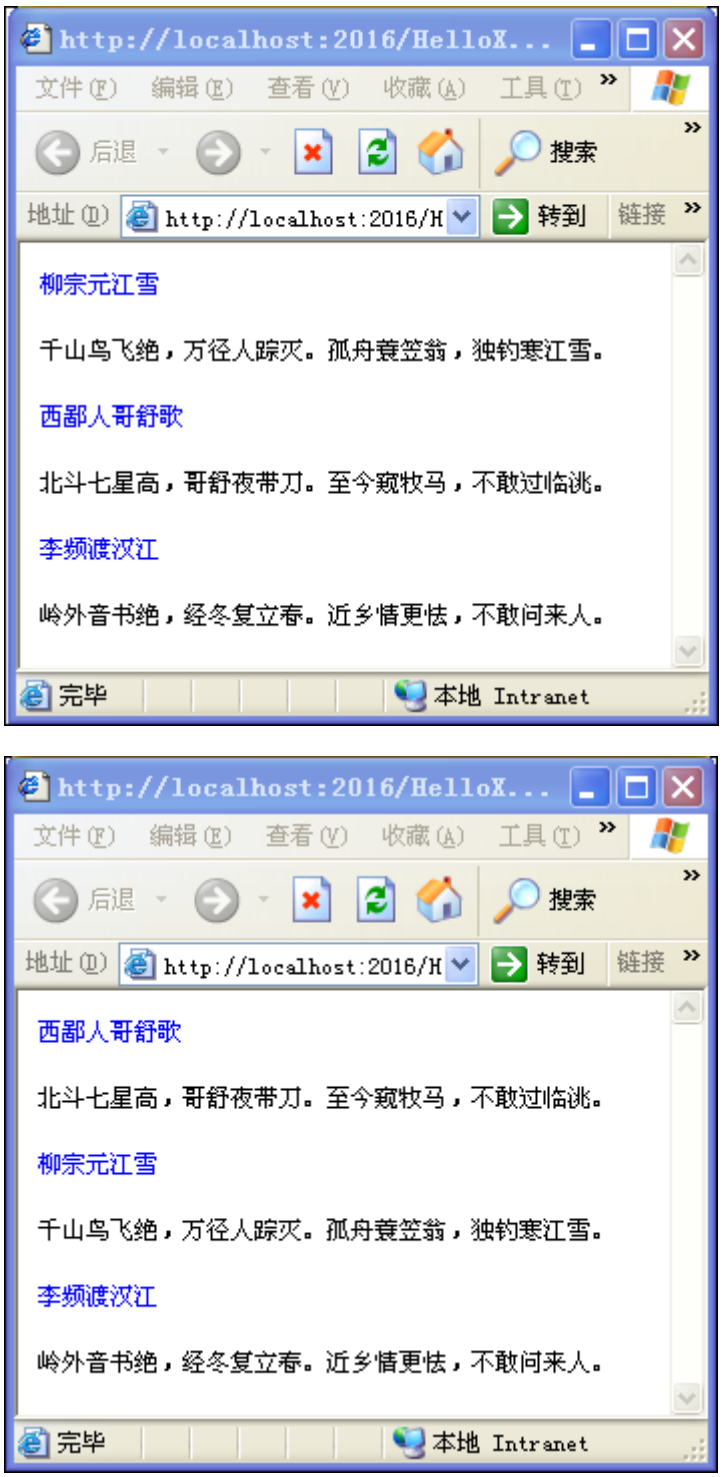

http://localhost:2016/HelloX...
文件(F) 编辑(E) 查看(V) 收藏(A) 工具(T)
后退
搜索
地址(D) http://localhost:2016/H
转到
链接
柳宗元江雪
千山鸟飞绝，万径人踪灭。孤舟蓑笠翁，独钓寒江雪。
西鄙人哥舒歌
北斗七星高，哥舒夜带刀。至今窥牧马，不敢过临洮。
李频渡汉江
岭外音书绝，经冬复立春。近乡情更怯，不敢问来人。
完毕
本地 Intranet
http://localhost:2016/HelloX...
文件(F) 编辑(E) 查看(V) 收藏(A) 工具(T)
后退
搜索
地址(D) http://localhost:2016/H
转到
链接
西鄙人哥舒歌
北斗七星高，哥舒夜带刀。至今窥牧马，不敢过临洮。
柳宗元江雪
千山鸟飞绝，万径人踪灭。孤舟蓑笠翁，独钓寒江雪。
李频渡汉江
岭外音书绝，经冬复立春。近乡情更怯，不敢问来人。
完毕
本地 Intranet

图 7-8

笔记区

5、 单条件判断语句<xsl:if>

If 对匹配条件进行判断，如果为真就执行条件内部的规则。

语法：

用元素的名称作为匹配条件：

<xsl:if test="元素名称"> 如果元素名等于什么时则执行

用元素内容作为匹配条件：

<xsl:if test="name[.='元素内容']"> 如果元素内容等于什么时则执行

用元素的属性值作为匹配条件：

<xsl:if test="@ID[.='属性值']"> 如果元素的属性值等于什么时则执行

演示一：元素的名称判断演示

C07.xml

Code 7-21

```
<?xml version="1.0" encoding="gb2312" ?>
<?xml-stylesheet type="text/xsl" href="C07.xsl"?>
<roster>
  <student ID="101">
    <name>李华</name>
    <score>92</score>
  </student>
  <student ID="102">
    <name>倪冰</name>
    <score>89</score>
  </student>
  <student ID="103">
    <name>张君宝</name>
    <score>98</score>
  </student>
  <student ID="104">
    <name>杨慧</name>
    <score>85</score>
```

```
  </student>
  <student ID="105">
<name>崔春晓</name>
    <score>86</score>
  </student>
</roster>
```

C07.xsl

Code 7-22

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
    <html>
<body>
 <xsl:for-each select="roster/student">
         <xsl:if test="name">
           <font style="color:blue">
             <xsl:value-of select="name"/>
           </font>
</xsl:if>
         <xsl:value-of select="score"/>
         <br></br>
       </xsl:for-each>
    </body>
    </html>
</xsl:template>
</xsl:stylesheet>
```

显示效果：

笔记区

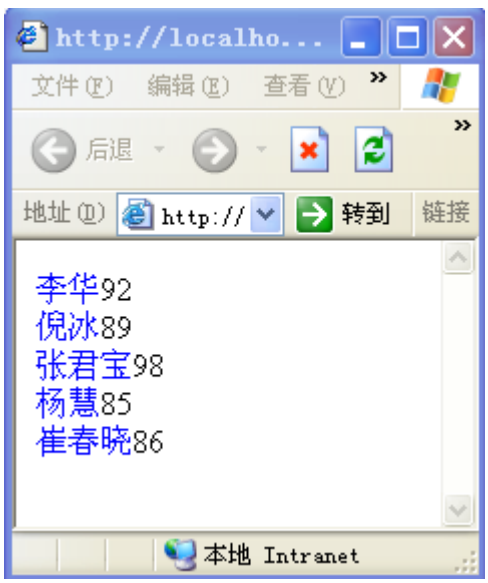


图 7-8

在本案例中，循环访问所有元素，如果元素名为 name 的元素的值，显示的时候设置为蓝色。

演示二 元素的内容判断演示 （xml 文档内容同上）

C08.xsl

Code 7-23

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
    <html>
    <body><xsl:apply-templates select="roster/student" />
    </body>
</html>
</xsl:template>
 <xsl:template match="roster/student">
    <xsl:value-of select="name"/>
    <xsl:if test="name[.='张君宝']">
      牛人，高分
```

```
    </xsl:if>
    <br></br>
  </xsl:template>
</xsl:stylesheet>
```

演示效果:

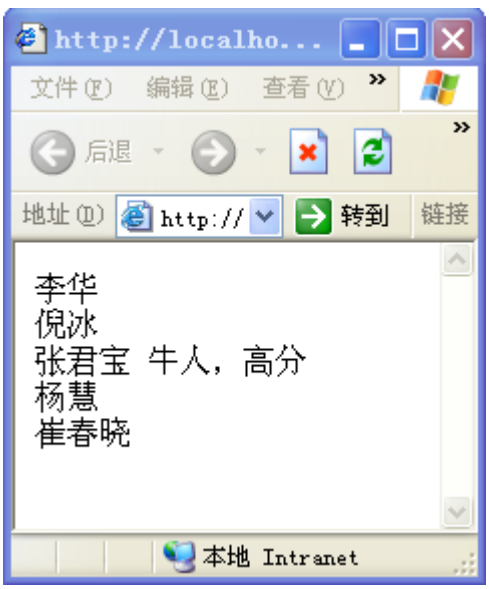


图 7-9

演示三 元素属性值判断演示 （xml 文档内容同上）

C09.xsl

Code 7-24

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
    <html>
    <body>
      <xsl:apply-templates select="roster/student" />
    </body>
    </html>
```

笔记区

```
</xsl:template>
    <xsl:template match="roster/student">
      <xsl:value-of select="name"/>
      <xsl:if test="@ID[.='105']">
     <font color="red">
          学号 105
        </font>
      </xsl:if>
 <br></br>
    </xsl:template>
</xsl:stylesheet>
```

演示效果

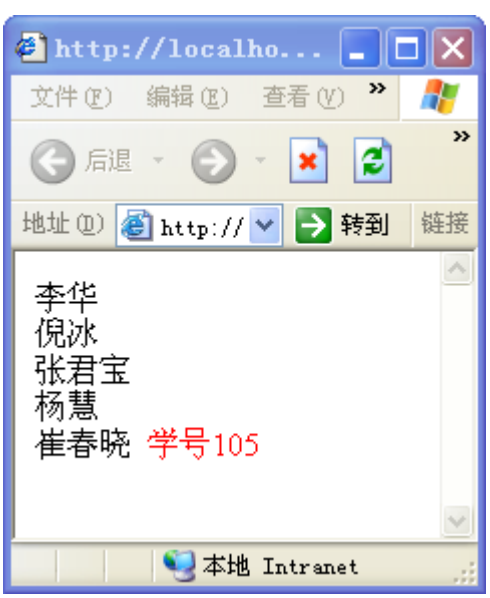


图 7-10

6、 多条件判断语句 choose when

多条件判断语句,C#中 switch case 多分支结构非常类似，可以实现多个条件匹配，如果匹配则执行，不匹配则继续匹配下一个，逐级匹配,直到最后；如果都不匹配则执行 otherwise。

语法：

```
<xsl:choose>
    <xsl: when test="匹配模式">
        <!--输出内容-->
    </xsl:when>
<xsl:when test="匹配模式">
        <!--输出内容-->
    </xsl:when>
        ....
    <xsl:otherwise>
        <!--输出内容-->
    </xsl:otherwise>
</xsl:choose>
```

多条件判断语句演示

C10.xml

Code 7-25

```
<?xml version="1.0" encoding="gb2312" ?>
<?xml-stylesheet type="text/xsl" href="C10.xsl"?>
<roster>
  <student ID="101">
    <name>李华</name>
    <score>92</score>
  </student>
  <student ID="102">
    <name>倪冰</name><score>89</score>
  </student>
  <student ID="103">
    <name>张君宝</name><score>98</score>
  </student>
  <student ID="104">
<name>杨慧</name>
```

笔记区

```
    <score>85</score>
  </student>
  <student ID="105">
    <name>崔春晓</name>
    <score>66</score>
  </student>
</roster>
```

C10.xsl

Code 7-26

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
    <html><body>
        <xsl:apply-templates   select="roster/student"/>
    </body></html>
</xsl:template>
  <xsl:template match="roster/student">
    <xsl:value-of select="name"/>
    <xsl:value-of select="score"/>
    <xsl:choose >
      <xsl:when test="score[. &gt; 90]">优</xsl:when>
      <xsl:when test="score[. &gt; 80]">良</xsl:when>
      <xsl:otherwise>一般</xsl:otherwise>
    </xsl:choose>
    <br></br>
  </xsl:template>
</xsl:stylesheet>
```

显示效果:

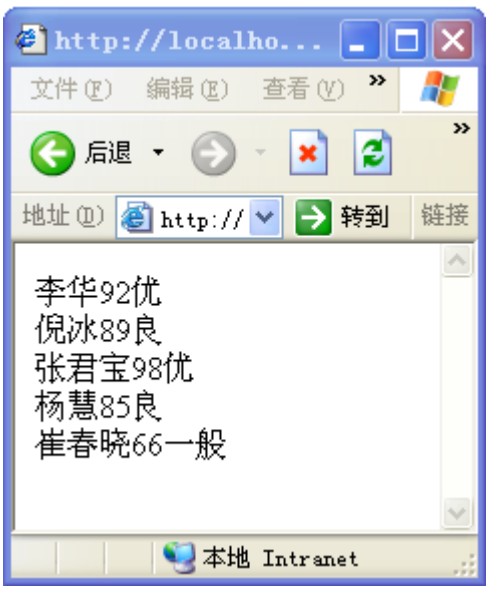


图 7-11

从这个例子中我们可以看到,通过 choose when 多条件判断可以判定每个元素中的值,给出相应的处理.

总结

HTML 网页使用预先确定的标识(tags)，这就是说所有的标记都有明确的含义，例如<p>是另起一行<h1>是标题字体。所有的浏览器都知道如何解析和显示 HTML 网页。

然而，XML 没有固定的标识,我们可以建立我们自己需要的标识，所以浏览器不能自动解析它们，例如<table>可以理解为表格，也可以理解为桌子。由于 XML 的可扩展性，使我们没有一个标准的办法来显示 XML 文档。为了控制 XML 文档的显示，我们有必要建立一种机制，CSS 就是其中的一种，但是 XSL(eXtensible Stylesheet Language)是显示 XML 文档的首选样式语言，它比 CSS 更适合于 XML。

通过本章的学习我们应该掌握 XSL 基本语法。

主要包括：

XSL 模板

节点选择语句<xsl:value-of>

循环判断语句<xsl:for-each>

条件判断语句<xsl:if>

笔记区

多条件判断语句<xsl:choose>

问与答

1：在 XML 中引用样式表的语法是什么？

2：有这样一个 xml 文件，创建一个带有转换模板的 XSL 样式表，实现以下效果

Xml 文档：

```
<?xml version="1.0" encoding="utf-8" ?>
<document>
  <grade>
    <name>张三</name>
    <english>80</english>
    <math>90</math>
    <sql>100</sql>
  </grade>
  <grade>
    <name>李四</name>
    <english>70</english>
    <math>50</math>
    <sql>80</sql>
  </grade>
  <grade>
    <name>王刚</name>
    <english>94</english>
    <math>98</math>
    <sql>92</sql>
  </grade>
</document>
```

效果：

学生姓名	英语	数学	SQL数据库
张三	及格	良好	优秀
李四	及格	不及格	及格
王刚	优秀	优秀	优秀

第八次课　XSL 综合应用

本章内容介绍

结合前面所讲的 XSL 技术，我可以将这个技术运用在求职简历的制作过程中，对于这样一个简历的制作，一方面加深了我们对于 XML 技术以及 XSL 样式表使用的了解，另一方面在简历制作中附带了技术附加值，体现了应聘软件开发相关岗位的技术特性，废话少说我们来看看这个例子是如何制作的。

本章内容	难度
Xml 简历制作	★★★

注：带※为重点学习内容

笔记区

8.1 Xml 简历制作

个人简历

个人资料

姓名	陈飞	性别	男
生日	1986年 10月 23日	健康状况	良好
籍贯	湖北	固定电话	027-88898123
手机	13871101234	邮编	430061
电子邮件	chengfei@163.com	通讯地址	武汉市武昌区民主路182号

自我评价

聪明伶俐，勤劳朴实，真诚勇敢

教育培训经历

教育类型	起止时间	教育培训机构	证书
初中	1993-09-1996-07	荆门石化三中	初中毕业证
高中	1996-09-1999-07	荆门石化一中	高中毕业证
大学	1999-09-2002-07	湖北广播电视大学计算应用专业	大专毕业证
培训	2002-04-2002-07	银河IT教育软件工程师认证	软件工程师

语言能力

汉语精通
英语熟练
日语了解

IT技能

熟悉 Visual Studio 2005
熟悉 C/S、B/S开发
熟悉 HTML、Css,JavaScript
熟悉 ORACLE、SQLServer2000大型数据库的开发
熟悉 Photoshop图片处理
能够独立完成设计

首先设计简历的 XML 主体结构,个人简历主要分为 5 个方向,分别是“个人资料”、“自我评价”、“教育培训经历”、“语言能力”、“专业技能”。

Code 8-1


```
<?xml version="1.0" encoding="gb2312" ?>
<?xml-stylesheet type="text/xsl" href="简历.xsl"?>
<简历>
<个人资料>...
<自我评价>聪明伶俐，勤劳朴实，真诚勇敢</自我评价>
<教育培训经历>...
<语言能力>...
<专业技能>...
</简历>
```

对于文档中有“...”符号的表示其中还有详细的格式，我们接下来逐个来看。

1、个人资料

Code 8-2

```
<个人资料>
    <姓名>陈飞</姓名>
    <性别>男</性别>
    <生日>
      <年>1986</年>
      <月>10</月>
      <日>23</日>
    </生日>
    <健康状况>良好</健康状况>
    <籍贯>湖北</籍贯>
    <联系方式>
      <固定电话>027-88898123</固定电话>
      <手机>13871101234</手机>
      <邮编>430061</邮编>
      <电子邮件>chengfei@163.com</电子邮件>
      <通讯地址>武汉市武昌区民主路号</通讯地址>
    </联系方式>
  </个人资料>
```

笔记区

这里个人资料各个信息分别列出来了，要注意的是这里把生日元素，分别列出了年月日，联系方式也是单独列出来多种联系方式，从这里可以看到本书前面提到的 XML 可以表示文档的层次结构。

2、 自我评价

自我评价部分没有进行详细的划分。

3、 教育培训经历

教育培训经历重要填写的是简历填写者的教育相关信息，包括"教育类型"、" 起止时间"、" 教育培训机构"、"证书"等。

Code 8-3

```
<教育培训经历>
    <经历>
      <教育类型>初中</教育类型>
      <起始时间>1993-09</起始时间>
      <结束时间>1996-07</结束时间>
      <教育培训机构>荆门石化三中</教育培训机构>
      <教育培训内容/>
      <学历证书>初中毕业证</学历证书>
    </经历>
    <经历>
      <教育类型>高中</教育类型>
      <起始时间>1996-09</起始时间>
      <结束时间>1999-07</结束时间>
      <教育培训机构>荆门石化一中</教育培训机构>
      <教育培训内容/>
      <学历证书>高中毕业证</学历证书>
    </经历>
    <经历>
      <教育类型>大学</教育类型>
      <起始时间>1999-09</起始时间>
      <结束时间>2002-07</结束时间>
```

```
    <教育培训机构>湖北广播电视大学</教育培训机构>
    <教育培训内容>计算应用专业</教育培训内容>
    <学历证书>大专毕业证</学历证书>
  </经历>
   <经历>
    <教育类型>培训</教育类型>
    <起始时间>2002-04</起始时间>
    <结束时间>2002-07</结束时间>
    <教育培训机构>银河 IT 教育</教育培训机构>
    <教育培训内容>软件工程师认证</教育培训内容>
    <学历证书>软件工程师</学历证书>
  </经历>
</教育培训经历>
```

我们每个人或多或少都会有一些教育和培训的经历，所以在这里体现 XML 结构的时候就需要出现这样的循环结构。

4、 语言能力

语言能力包括简历填写者所掌握的语言和熟练程度。

Code 8-3

```
<语言能力>
  <语言>
    <语言类型>汉语</语言类型>
    <熟练程度>精通</熟练程度>
  </语言>
  <语言>
    <语言类型>英语</语言类型>
    <熟练程度>熟练</熟练程度>
  </语言>
  <语言>
    <语言类型>日语</语言类型>
    <熟练程度>了解</熟练程度>
```

笔记区

```
    </语言>
  </语言能力>
```

语言能力和教育培训经历比较类似，每一个人也会掌握一到多门语言，所以使用这样循环结构。

5、 专业技能

Code 8-4

```
  <专业技能>
    <技能>熟悉 Visual Studio 2005</技能>
    <技能>熟悉 C/S、B/S 开发</技能>
    <技能>熟悉 HTML、Css,JavaScript</技能>
    <技能>熟悉 ORACLE、SQLServer2000 大型数据库的开发</技能>
    <技能>熟悉 Photoshop 图片处理</技能>
    <技能>能够独立完成设计</技能>
  </专业技能>
```

以上简历为 XML 内容部分的结构和内容，接下来我们来看看 XSL 是如何制作的。首先是 XSL 样式表文档的声明部分。

Code 8-5

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

紧接着是根模板的定义。

```
  <xsl:template match="/">
    <html>
      <body>
        <h1 align="center">个人简历</h1>
        <table align="center" width="600" border="1">
          <tr>
            <td><strong>个人资料</strong></td>
```

笔记区

```
            </tr>
            <tr>
              <td>
                <xsl:apply-templates select="简历/个人资料
"></xsl:apply-templates>
              </td>
            </tr>
            <tr>
              <td><strong>自我评价</strong></td>
            </tr>
            <tr>
              <td>
                <xsl:apply-templates  select ="简历/自我评价"/>
              </td>
            </tr>
            <tr>
              <td><strong>教育培训经历</strong></td>
            </tr>
            <tr>
              <td>
             <xsl:apply-templates  select ="简历/教育培训经历"/>
              </td>
            </tr>
            <tr>
              <td><strong>语言能力</strong></td>
            </tr>
            <tr>
              <td>
                <xsl:apply-templates select="简历/语言能力"/>
              </td>
```

笔记区

```
        </tr>
        <tr>
          <td>
            <xsl:apply-templates select="简历/语言能力"/>
          </td>
        </tr>
        <tr>
          <td>
            <strong>IT 技能</strong>
          </td>
        </tr>
        <tr>
          <td>
            <xsl:apply-templates select="简历/专业技能"/>
          </td>
        </tr>
      </table>
    </body>
  </html>
</xsl:template>
```

在这个 xsl 中手工定义一个 10 行一列的表格，其中五行是板块名称，另外五行调用的是各个板块内容模板。接下来我们继续看看各个板块的模板定义。

6、 个人资料

Code 8-6

```
<xsl:template match ="简历/个人资料">
  <table width="600" border="1">
    <tr>
      <td>姓名</td>
<td><xsl:value-of select ="姓名"/></td>
      <td>性别</td>
```

```
      <td><xsl:value-of select ="性别"/></td>
    </tr>
    <tr>
<td>生日</td>
      <td><xsl:apply-templates select ="生日"/></td>
      <td>健康状况</td>
      <td><xsl:value-of select ="健康状况"/></td>
    </tr>
    <tr>
      <td>籍贯</td>
      <td><xsl:value-of select ="籍贯"/></td>
      <td>固定电话</td>
      <td><xsl:value-of   select ="联系方式/固定电话"/></td>
    </tr>
    <tr>
      <td>手机</td>
      <td><xsl:value-of   select ="联系方式/手机"/></td>
      <td>邮编</td>
      <td><xsl:value-of   select ="联系方式/邮编"/></td>
    </tr>
    <tr>
      <td>电子邮件</td>
      <td><xsl:value-of   select ="联系方式/电子邮件"/></td>

      <td>通讯地址</td>
      <td><xsl:value-of   select ="联系方式/通讯地址"/></td>
    </tr>
  </table>
</xsl:template>
```

这个模板定义的时候匹配了“简历/个人资料”的所有元素，逐个的使用 xsl:value-of 元素

笔记区

把值取出来。值得注意的是在生日数据读取的时候也是调用的模板生日模板结构如下：

Code 8-7

```
<xsl:template match ="生日">
  <xsl:value-of select ="年"/>年
  <xsl:value-of select ="月"/>月
  <xsl:value-of select ="日"/>日
</xsl:template>
```

下面一个是教育经历模板的定义：

Code 8-8

```
<xsl:template match ="教育培训经历">
    <table border="1" width="600">
      <tr>
      <td>教育类型</td><td>起止时间</td>
      <td>教育培训机构</td><td>证书</td>
      </tr>
      <xsl:for-each select="经历">
        <tr>
          <td>
            <xsl:value-of select ="教育类型"/>
          </td>
<td>
            <xsl:value-of select ="起始时间"/>-<xsl:value-of select ="结束时间
"/>
          </td>
          <td>
            <xsl:value-of select ="教育培训机构"/>
            <xsl:value-of select ="教育培训内容"/>
          </td>
          <td>
            <xsl:value-of select ="学历证书"/>
```


```
        </td>
      </tr>
    </xsl:for-each>
  </table>
</xsl:template>
```

在这个模板的定义过程中首先定义了一行四列的表头，之后使用了一个循环元素 for-each 遍历“教育培训经历”中的所有“经历”，并且每循环一次生成一个新的一行表记录。这样就可以将培训经历循环生成一张表格。

下面一个是语言能力模板的定义：

Code 8-9

```
<xsl:template match="语言能力">
  <xsl:for-each select="语言">
    <xsl:value-of select="语言类型"/>
<xsl:value-of select="熟练程度"/>
    <br></br>
  </xsl:for-each>
</xsl:template>
```

这个模板也是使用 for-each 循环结构遍历数据。

最后一个是专业技能模板的定义：

Code 8-10

```
  <xsl:template match="专业技能/技能">
    <xsl:value-of select="."/>
    <br></br>
  </xsl:template>
</xsl:stylesheet>
```

和前面的结构比较类似，这里就不详细累述了。以上这就是一个简历 xml 文档和 xsl 编写的过程，读者可以在此基础之上对部分功能结构进行一些特殊的改动。

总结

笔记区

CSS 与 XSL 在某种程度上是重复的。XSL 的功能确实比 CSS 更强大，但是 XSL 的功能与其复杂性是分不开的。这一章仅仅涉及了 XSL 最基本的用途。实际上 XSL 更复杂，而且比 CSS 更难学习和使用，同时也带来了一个问题："什么时候应该使用 CSS，什么时候应该使用 XSL？"

XSL 终将成为现实世界和大量数据应用的最佳选择，CSS 更适合于简单的页面，如祖母用于向她们孙子寄送图片的页面。但对于这些用途，HTML 已经足够。如果使用 HTML 行不通，XML+CSS 不会有多大的帮助。相比而言，XML+XSL 能够解决更多 HTML 不能解决的困难。对于传统的浏览器来说，仍然需要 CSS，但长远看来使用 XSL 才是发展方向。

问与答

1：请同学们思考并尝试：能否用 xml+xsl 实现一个简单网页？

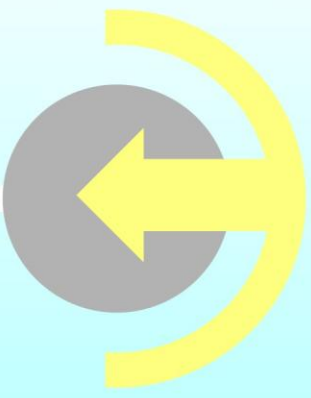
4部分

XML实际应用

通过前三个部分的学习，我们征集到了xml的各种专业术语，掌握了定义DTD或Schema来描述xml文档结构，会编写结构良好且有效的xml文档,并且掌握了如何应用xsl来显示xml文档。相信同学们已经从一个xml菜鸟变身为一个xml大虾了，那么在接下来的课程中，我们将结合.net学习xml在.net中的实际应用。

第九次课 .net中的XML

第十次课 XML序列化



第九次课 .NET 中的 XML

学习内容	难度
.net 中 xml 命名空间	★
Xml 文档对象模型(DOM)	★
※.net 中如何读取 xml 文件	★★★
使用 XmlDataDocument 加载关系数据	★
※使用 XmlNode 对象操作 xml 节点	★★

注：带※为重点学习内容

笔记区

当微软的 XML 小组开始为.NET Framework 设计 XML 特性的时候，如下这些已经在他们的脑海中了：更好的标准兼容—主要的 W3C XML 标准提供跨平台交互操作支持，如 XML 1.0 、XML Namespaces 1.0 、XSLT 1.0、XPath 1.0 和 W3C XML 模式 1.0.

易用——XML API 不仅应该易用而且应该直观。

与 ADO.NET 的无缝整合—XML API 中的类能够真正的看做 ADO.NET 的一部分并作为 XML 数据访问的 API。System.Data.DataSet 和 System.Xml.XmlDataDocument 类的组合为在 XML 与相关数据之间移动提供了无缝体验。

明显的性能改善——这是.NET Framework 版本的的第一个需求。通过引入新的 System.Xml.Xsl.XslCompiliedTransform 类，新的 XSLT 处理机制是.NET Framework 中众多 XML API 性能改善之一。

提高开发人员的效率——这些代码改善都是为了提供开发人员的效率，允许他们执行通用任务而无需编写代码。

对强类型和 XML 模式的支持——在.NET Framework 1.1 中，几乎所有的 XML API 都是无类型的。因为数据的存储和获取都是作为字符串类型。这种情况将在.NET Framework2.0 及后续版本中通过将 XML 命名空间的模式信息进行整合来改善。这将提供更加有效的存储，提升的性能以及与.NET 编程更好的整合。

9.1 .NET 中的 XML 命名空间

在.NET Framework 中,XML API 主要封装的五个命名空间中.这些命名空间装载了.NET Framework 类库中所有的 XML 功能.

表 9-1

命名空间	说明
System.Xml	包含了可以提供所有 XML 的核心功能
System.Xml.Schema	为 XML 模式定义（XSD）语言模式提供支持
System.Xml.Serialization	提供允许将对象串行化为 XML 格式化的文档的类
System.Xml.XPath	为 XPath 解析器和评估功能提供支持
System.Xml.Xsl	为 XSLT 转换提供支持

笔记区

1、 System.Xm 命名空间

System.Xml 命名空间中的类可以为你在 XML 编程方面提供完全的支持。它包括对 XML 的读取和写入 XML 的保存 XML。实际上，你的应用程序中甚至可以实现对 XML 的查询和转换。在这个命名空间中有很多功能丰富的来可以用来读取、写入，操作 XML 文档。

2、 System.Xml.Schema 命名空间

该命名空间提供用于支持 XSD 语言的需求类、代理和枚举。它完全支持 W3C 规定的 XML 模式的结构和 XML 模式的数据类型，此命名空间中的类服务于 Schema Object Model（SOM）

3、 System.Xml.XPath 命名空间

这个命名空间通过一些类、接口和枚举提供对 XPath 解析器的支持（查询支持）。这个命名空间中常用的两个类是 XPathDocument（快速，只读缓存，为 XSLT 而优化）和 XPathNavigator（可编辑，随机访问，指针模型）。请注意现在的 XPathNavigator 类可以用于编辑 XML 数据，还可以为导航 XML 数据提供指针模型。

4、 System.Xml.Xsl 命名空间

这个命名空间对可扩展样式表转换（XSLT）技术提供了完全的支持。虽然在此命名空间中提供了一些类和接口，但你还是可能经常会使用 XslCompiledTransform 类和 XsltArgumentList 类。

5、 System.Xml.Serialization 命名空间

这个命名空间提供了类和代理来帮助对象的串行化。在提供的很多托管类当中，你将更多的使用 XmlSerializer 类。通过 XmlSerializer 类，你可以串行化和反串行化 XML 文档或者数据流。对象串行化是一种很有用的技术，可用来保持（保存）对象的状态以便你能够再次创建一个与该对象完全一样的副本。当你想在.NET Remoting 中传递对象，对象串行化是很有用的。

9.2 XML 文档对象模型(DOM)

为了让读者更好的理解后面使用.NET 操作 XML 的例子，在这里我们给出一个 XML 的例子。

Code 9-1

```
<?xml version="1.0" encoding="gb2312" ?>
<学生>
```

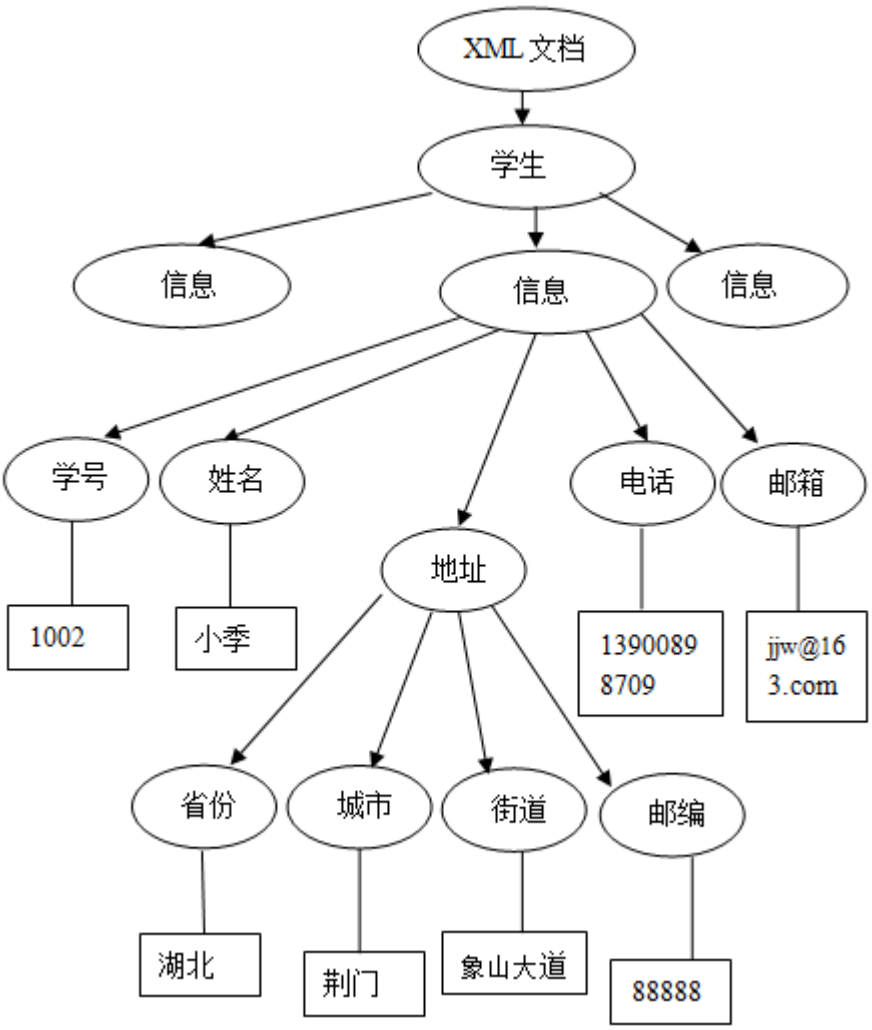
笔记区

```
  <信息>
    <学号>1001</学号>
    <姓名>李四</姓名>
    <地址>
      <省份>湖北</省份>
      <城市>武汉</城市>
      <街道>武昌大道号</街道>
        <邮编>999999</邮编>
    </地址>
<电话>13900898709</电话>
<邮箱>lipu@163.com</邮箱>
  </信息>
  <信息>
    <学号>1002</学号>
    <姓名>小季</姓名
    <地址>
      <省份>湖北</省份>
      <城市>荆门</城市>
          <街道>象山大道号</街道>
      <邮编>88888</邮编>
    </地址>
    <电话>13900898709</电话>
    <邮箱>jjw@163.com</邮箱>
  </信息>
</学生>
```

上面给出了一个很简单的 XML 文件，这个”student.xml”文件将作为示例在后面各章节中使用。

XML 文档层次结构图

笔记区



XML 文档对象模型图(DOM,Document Object Model)正是如上图所示的一个结构模型，用于在内存中表示 XML 文档。上面的 DOM 树定义了一个 XML 文档的逻辑结构，给出了一种应用程序访问和处理 XML 文档的方法。在 DOM 树中，有一个根节点 Document 节点，所以其它的节点都是根节点的后代。在应用程序中，基于 DOM 树的 XML 分析器将一个 XML 文档转换成一模 DOM 树，应用程序通过对 DOM 树的操作来实现对 XML 文档中的数据的操作。

有了 DOM 模型，.NET 只需要实现一系列能够方便操作 DOM 树的类，就能实现以编程的方式去读取、修改和操作 XML 文档了。下面我们来介绍 DOM 中的一些基本述语。

节点(Node)：指 DOM 树中的节点，树的根部被称为文档节点或根节点。在 DOM 模型中有 7 种节点：命名空间、根节点、元素、属性、注解、说明、文字。

笔记区

原子节点：指那些没有子节点或父节点的节点。

父子关系：节点之间的父子关系包括父关系、子关系。

兄弟关系：有相同父节点的节点。例如上面 DOM 模型图中的学号和姓名。

祖先(Ancestors)：节点的祖先指其父亲，父亲的父亲……。

后代(Descendants)：节点的后代指其孩子，孩子的孩子……。

9. 3 在.NET 中如何读取 XML 文档

在.NET 框架中，操作 DOM 模型的类位于 System.Xml 命名空间中，其中常用的类有：XmlAttribute、XmlNotation、XmlDocumentType、XmlNodeReader、XmlElement、XmlReader、XmlTextWriter、XmlTextreader、XmlText、XmlNode、XmlNodeList、XmlDocument、XmlDataDocument 等。

1、 使用 XmlDocument 读取 XML

使用 XmlDocument 对象的 Load 方法,可以从指定的字符串中加载 XML 文档。例如：

Code 9-2

```
//创建 XmlDocument 对象 xdoc
XmlDocument xdoc = new XmlDocument();
//读取 Xml 文件”student.xml”
xdoc.Load(Server.MapPath("FileXML\\student.xml"));
//在 TextBox 中显示
this.TextBox1.Text = xdoc.InnerXml;
```

2、 使用 XMLReader 读取 XML

XmlReader 是一个抽象类，提供对 XML 数据进行快速、只进的访问。它能非常高效地读取 XML 文档中的单个节点。例如：

Code 9-3

```
XmlTextReader xread = new
XmlTextReader(Server.MapPath("FileXML\\info.xml"));
xread.WhitespaceHandling = WhitespaceHandling.None;
//解析 XML 文档，并输出所有节点
while (xread.Read())
```


```
{
    for (int i = 0; i < xread.Depth; i++)
    {
        Response.Write("    ");
    }
    switch (xread.NodeType)  //判断节点类型
    {
        case XmlNodeType.Element: //元素
            Response.Write(xread.Name);
            break;
        case XmlNodeType.Text:  //内容
            Response.Write(xread.Value);
            break;
    }
}
if (xread != null)
{
    xread.Close();
}
```

使用 XmlDataDocument 加载关系数据:

XmlDataDocument 类扩展了 XmlDocument 类，利用它不但可以加载 XML 数据，还可以加载关系数据。XmlDataDocument 与 ADO.NET 中的数据集 DataSet 类有着密切的联系，可以将 XML 数据加载至 DataSet，以关系数据的形式显示，并且可以对 XML 进行同步操作。

XmlDataDocument 扩展自 Xmldocument，因为它与后者非常的类似。与后者相比，XmlDataDocument 具有属性 DataSet，这个属性就是 XmlDatadocument 中所加载的 XML 数据的关系表示形式。

上面的案例演示了如果使用 DataSet 加载 XML 数据,返过来我们把关系数据以 DOM 对象的形式加载致电 XmlDataDocument 中,然后就可以使用 XML 的操作技术进行数据操作。DataSet 中的任何更改都将同步反应到 XmlDataDocumnet 中，在 XmlDataDocument

笔记区

中进行的所有更改也将同步反应到 DataSet 中。

使用 **XmlDataDocument** 加载关系数据

Code 9-4

```
SqlConnection conn = new
SqlConnection("server=.;database=northwind;uid=sa;pwd=;");
SqlDataAdapter da = new SqlDataAdapter("select * from employees", conn);
DataSet ds = new DataSet();
da.Fill(ds, "emp");
Response.Write(ds.GetXml());
```

DataSet 与 Xml 交互

ADO.NET 对 XML 提供了强大的支持，它主要通过 DataSet 与 XML 进行交互。DataSet 中提供了很多方法来支持对 XML 的操作。

表 9-2

方法	说明
GetXml	返回存储在 DataSet 数据中的 XML 数据。
GetXmlSchema	返回存储在 DataSet 数据中的 XML 表示 Schema。
ReadXml	已重载。将 XML 架构和数据读入 DataSet。
ReadXmlSchema	已重载。将 XML 架构读入 DataSet。
WriteXml	已重载。从 DataSet 中， 写 XML 数据。
WriteXmlSchema	已重载。从 DataSet 中， 写 XML 数据的 Schema。

9.4 使用 XmlNode 对象操作 Xml 节点

名师提点

XML 的每一个节点都包括很多内容，如节点标签名，节点属性，数值等。XmlNode 对象用于实现一个 XML 节点，使用此对象可以完成对节点的绝大部分操作。

示例 1

笔记区

Code 9-5

```
//命名用 XmlDocument 对象读取 XML
XmlDocument xdoc = new XmlDocument();
xdoc.Load(Server.MapPath("FileXML\\info.xml"));
XmlNode xn = xdoc.DocumentElement.FirstChild;
//输出第一个节点的详细信息。
Response.Write("节点名称：" + xn.Name + "<br>");
Response.Write("节点类型：" + xn.NodeType + "<br>");
Response.Write("是否有了节点：" + xn.HasChildNodes + "<br>");
Response.Write("了节点的值：" + xn.InnerText + "<br>");
Response.Write("父节点名称：" + xn.ParentNode.Name + "<br>");
```

示例 2：修改 web.config,实现更换站点主题

Code 9-5

```
    //创建 XML 文档实例
        XmlDocument XMLWebSetting = new XmlDocument();
        //打开 XML 文档
        XMLWebSetting.Load(Server.MapPath("Web.Config"));
        //查找 pages 这个节点
        XmlNode node =
XMLWebSetting.SelectSingleNode("configuration/system.web/pages");
        if (node != null)
        {
            node.Attributes["theme"].InnerText = theme;
            XMLWebSetting.Save(Server.MapPath("Web.Config"));
        }
```

总结

本章我们了解到了在.NET 中的 5 个对应 XML 编程的命名空间，这些命名空间提供了对于 XML 遍历、检索、读取、修改、保存、文档结构验证、导航、样式显示、序列化等支持。

笔记区

以及在这里详细讲解了 DOM 文档对象模型，XML 读取和保存。

通过对本章的学习，同学们应该熟练掌握在.net 中对 xml 文档、xml 节点的相关操作。

问与答

1：有这样一个 xml 文档（如下），请用编程的方式实现：能够动态修改姓名、性别和年龄。

```
<?xml version="1.0" encoding="gb2312"?>
<学生>
  <信息>
    <姓名>张三</姓名>
    <性别>男</性别>
    <年龄>20</年龄>
  </信息>

  <信息>
    <姓名>李四</姓名>
    <性别>女</性别>
    <年龄>30</年龄>
  </信息>
</学生>
```

2:动态修改 web.config 中数据库连接字符串。

第十次课 XML 序列化

学习内容	难度
※序列化的概念、作用	★
※.net 中 xml 序列化与反序列化	★★
Xml 序列化高级方案示例	★★★

注：带※为重点学习内容

笔记区

10.1 序列化概念

1、 序列化的概念

序列化是将对象状态转换为可保持或传输格式的过程。与序列化相对的是反序列化，它将流转换为对象。这两个过程结合起来，可以轻松的存储和传输数据。

.NET Framework 提供两种序列化技术：二进制序列化，XML 序列化。

二进制序列化保持类型保真度，这对于在应用程序的不同调用之间保留对象的状态很有用。例如，通过将对象序列化到剪贴板，可在不同的应用程序之间共享对象。您可以将对象序列化到流、磁盘、内存和网络等等。远程处理使用序列化"通过值"在计算机或应用程序域之间传递对象。

XML 序列化仅序列化公共属性和字段，且不保持类型保真度。当我们要提供或使用数据而不限制使用该数据的应用程序时，这一点是很有用的。由于 XML 是一个开放式标准，因此，对于通过 Web 共享数据而言，这是一个很好的选择。

这里我们主要学习 XML 序列化。

2、 序列化的应用目的

为什么我们想要使用序列化？有两个最重要的原因：

一个原因是将对象的状态永久保存在存储媒体中，以便可以在以后重新创建精确的副本；

另一个原因是通过值将对象从一个应用程序域发送到另一个应用程序域中。例如，序列化可用于在 ASP.NET 中保存会话状态并将对象复制到 Windows 窗体的剪贴板中。远程处理还可以使用序列化通过值将对象从一个应用程序域传递到另一个应用程序域中。

10.2 XML 序列化/反序列化

XML 序列化仅将对象的公共字段和属性值序列化为 XML 流。XML 序列化中最主要的类是 XmlSerializer 类，它的最重要的方法是 Serialize 和 Deserialize 方法。

1、 使用 XmlSerializer 类可将下列项序列化：

- 公共类的公共读/写属性和字段
- 实现 ICollection 或 IEnumerable 的类
- XmlElement 对象
- XmlNode 对象
- DataSet 对象

名师提点

XML 序列化不转换方法、索引器、私有字段或只读属性（只读集合除外）。要序列化对象的所有字段和属性（公共的和私有的），请使用 BinaryFormatter，而不要使用 XML 序列化。

2、 使用 XML 序列化的好处

XmlSerializer 类在您将对象序列化为 XML 时为您提供完整而灵活的控制。如果您正在创建 XML Web 服务，则可以将控制序列化的属性应用于类和成员，以确保 XML 输出符合特定的架构。

例如，XmlSerializer 使您能够：

指定将字段或属性编码为特性还是元素。

指定要使用的 XML 命名空间。

如果字段或属性名不合适，指定元素或特性的名称。

XML 序列化的另一个好处是：只要生成的 XML 流符合给定的架构，对于所开发的应用程序就没有约束。 假定有这样一个用于描述图书的架构。该架构具有标题、作者、出版商和 ISBN 编号元素。您可以开发一个以您希望的任何方式（例如，作为图书订单或作为图书清单）处理 XML 数据的应用程序。在任何一种情况下，唯一的要求是 XML 流应当符合指定的 XML 架构定义语言 (XSD) 架构。

3、 序列化对象

要序列化对象，首先创建要序列化的对象并设置它的公共属性和字段。为此，必须确定要用以存储 XML 流的传输格式（或者作为流，或者作为文件）。例如，如果 XML 流必须以永久形式保存，则创建 FileStream 对象。

序列化对象的步骤：

- 创建对象并设置它的公共字段和属性。
- 使用该对象的类型构造 XmlSerializer。
- 调用 Serialize 方法以生成对象的公共属性和字段的 XML 流表示形式或文件表示形式。

下面的示例创建一个文件：

Code 10-1

笔记区

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Xml.Serialization;//此命名空间包含用于将对象序列化为 XML 格式文档或流的类。
using System.IO;//此命名空间包含操作文件和流的类
namespace XmlTest
{
        //定义一个类 MySerializableClass
    public class MySerializableClass
    {
        //公共字段
        public string strName;
        public int iAge;
    }
 class Class1
    {
        //在这里我们将 MySerializableClass 的对象序列化
        static void Main()
        {
            MySerializableClass myObject = new MySerializableClass();
            myObject.strName = "张三";
            myObject.iAge = 21;
            //使用 myObject 的类型创建 XmlSerializer 对象 mySerializer
            XmlSerializer mySerializer = new
XmlSerializer(typeof(MySerializableClass));
            // 创建一个 StreamWriter 对象
            StreamWriter myWriter = new StreamWriter("myFileName.xml");
            //调用 Serialize 方法将对象序列化 写入 XML 文件中
            mySerializer.Serialize(myWriter, myObject);
```

```
            myWriter.Close();
        }
    }
}
```

myFileName.xml 的内容

Code 10-2

```
<?xml version="1.0" encoding="utf-8"?>
<MySerializableClass
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <strName>张三</strName>
  <iAge>21</iAge>
</MySerializableClass>
```

4、 将对象反序列化

当您反序列化对象时，传输格式确定您将创建流还是文件对象。确定了传输格式之后，就可以根据需要调用 Serialize 或 Deserialize 方法。

反序列化对象的步骤：

- 使用要反序列化的对象的类型构造 XmlSerializer。

- 调用 Deserialize 方法以生成该对象的副本。在反序列化时，必须将返回的对象强制转换为原始对象的类型。 示例：

Code 10-3

```
//反序列化
MySerializableClass myObject;
//使用对象 myObject 的类型创建 XmlSerializer 对象 mySerializer
XmlSerializer mySerializer = new XmlSerializer(typeof(MySerializableClass));
//创建一个文件流来读取 XML 文件
FileStream myFileStream = new FileStream("myFileName.xml", FileMode.Open);
// 调用 Deserialize 方法将对象反序列化
myObject = (MySerializableClass)mySerializer.Deserialize(myFileStream);
```

笔记区

```
Console.WriteLine("iAge:{0}", myObject.iAge);
Console.WriteLine("strName:{0}", myObject.strName);
```

运行程序后：

图 10-1

10.3 XML 序列化高级方案示例

XML 序列化可具有多种形式，从简单到复杂的都有。例如，可以序列化仅包含公共字段和属性的类。下面的代码示例用于处理各种高级方案，包括如何使用 XML 序列化来生成符合特定 XML 架构 (XSD) 要求的 XML 流。

1、 序列化数据集

除了对公共类的实例进行序列化之外，DataSet 的实例也可以被序列化，如下面的代码示例所示：

Code 10-4

```
private void SerializeDataSet(string filename)
{
    XmlSerializer ser = new XmlSerializer(typeof(DataSet));

    // 创建一个 DataSet 对象它拥有一个列行的的 DataTable
    DataSet ds = new DataSet("myDataSet");
    DataTable t = new DataTable("table1");
    DataColumn c = new DataColumn("thing");
    t.Columns.Add(c);
    ds.Tables.Add(t);
    DataRow r;
    for (int i = 0; i < 10; i++)
```

```
    {
        r = t.NewRow();
        r[0] = "Thing " + i;
        t.Rows.Add(r);
    }
    TextWriter writer = new StreamWriter(filename);
    ser.Serialize(writer, ds);
    writer.Close();
}
```

2、 序列化 XmlElement 和 XmlNode

您还可以对 XmlElement 或 XmlNode 类的实例进行序列化，如下面的代码示例所示：

Code 10-5

```
private void SerializeElement(string filename)
{
    XmlSerializer ser = new XmlSerializer(typeof(XmlElement));
    XmlElement myElement =
    new XmlDocument().CreateElement("MyElement", "ns");
    myElement.InnerText = "Hello World";
    TextWriter writer = new StreamWriter(filename);
    ser.Serialize(writer, myElement);
    writer.Close();
}
private void SerializeNode(string filename)
{
    XmlSerializer ser = new XmlSerializer(typeof(XmlNode));
    XmlNode myNode = new XmlDocument().
```

笔记区

```
    CreateNode(XmlNodeType.Element, "MyNode", "ns");
    myNode.InnerText = "Hello Node";
    TextWriter writer = new StreamWriter(filename);
    ser.Serialize(writer, myNode);
    writer.Close();
}
```

3、 序列化返回复杂对象的类

如果属性或字段返回一个复杂对象（如数组或类实例），则 XmlSerializer 将其转换为嵌套在主 XML 文档内的元素。例如，下面代码示例中的第一个类返回第二个类的实例：

Code 10-6

```
public class PurchaseOrder
{
    public Address MyAddress;
}
public class Address
{
    public string FirstName;
}
```

序列化的 XML 输出如下：

Code 10-7

```
<PurchaseOrder>
    <Address>
        <FirstName>George</FirstName>
    </Address>
</PurchaseOrder>
```

4、 序列化对象数组

您还可以序列化返回对象数组的字段，如下面的代码示例所示：

Code 10-8

```
public class PurchaseOrder
```


```
{
    public Item[] ItemsOrders;
}
public class Item
{
    public string ItemID;
    public decimal ItemPrice;
}
```

如果订购了两件商品，则序列化后输出如下：

Code 10-9

```
<PurchaseOrder xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <Items>
        <Item>
            <ItemID>aaa111</ItemID>
            <ItemPrice>34.22</ItemPrice>
        <Item>
        <Item>
            <ItemID>bbb222</ItemID>
            <ItemPrice>2.89</ItemPrice>
        <Item>
    </Items>
</PurchaseOrder>
```

总结

XML 的应用不断发展，XML 在构建现代化信息系统中的作用越来越重要，.NET 技术作为当今的主流软件开发平台，对于 XML 支持也是越来越大，在 Visual Studio 2008 中提供了很多新的开发工具，帮助开发人员更好的将 XML 技术运用到实际开发项目中。

笔记区

问与答

1：序列化有什么作用？

2：.net 中提供哪些序列化的方式?

3：二进制度序列化与 XML 序列化有什么区别？

4：使用 XmLSerializer 类可以序列化哪些对象？

5：建 1 个公共类，对其公共属性和字段 序列化。

6：对题 5 序列化生成的 xml 文件，进行反序列化。

7：序列化 DataSet 有什么简便方法？

附录 1 （单词表）

单词表

XML（eXtensible Markup Language）：

即可扩展标记语言，它与 HTML 一样，都是 SGML(Standard Generalized Markup Language,标准通用标记语言)

Mac OS:

Mac OS 是一套运行于苹果 Macintosh 系列电脑上的操作系统。Mac OS 是首个在商用领域成功的图形用户界面。现行的最新的系统版本是 Mac OS X 10.5.x 版

SGML:

SGML(Standard Generalized Markup Language,标准通用标记语言)，是一种定义电子文档结构和描述其内容的国际标准语言，是所有电子文档标记语言的起源，早在 Web 发明之前 SGML 就已存在.SGML 是 1986 年出版发布的一个信息管理方面的国际标准(ISO 8879)。

W3C:

W3C 是英文 World Wide Web Consortium 的缩写，中文意思是 W3C 理事会或万维网联盟。W3C 于 1994 年 10 月在麻省理工学院计算机科学实验室成立。创建者是万维网的发明者 Tim Berners-Lee。

Linux:

Linux 操作系统，是一种计算机操作系统，读音为 ['li:nəks]。Linux 操作系统的内核的名字也是"Linux"。Linux 操作系统也是自由软件和开放源代码发展中最著名的例子。简单地说，Linux 是一套免费使用和自由传播的类 Unix 操作系统，它主要用于基于 Intel x86 系列 CPU 的计算机上

GML（Generalized Markup Language）：标准化后的名称。

ANSI:

美国国家标准学会（American National Standards Institute，ANSI）是负责制定美国国家标准的非营利组织。美国国家标准学会授权标准起草机构按照一系列规范编写标准草案。由此产生的候选文献通过 ANSI 审核批准后成为美国国家标准。国国家标准学会成立于 1918 年 10 月 19 日，最初名称为美国工程标准委员会，1928 年改为现名。

Netscape:

网景（Netscape）是一个自 1994 年开始的品牌。它亦是网景通讯公司（Netscape Communications Corporation，1994 年 4 月 4 日—2003 年 7 月 5 日）的常用简称。网景通讯公司曾经是一家美国的电脑服务公司，以其生产的同名网页浏览器而闻名。就市场占有率来说，网景的网页浏览器曾占首位，但在“第一次浏览器大战”中被微软的 Internet Explorer 所击败。现时网景各版本的网页浏览器的使用率的总和不足一个百分比。

XHTML:

可扩展超文本置标语言（eXtensible HyperText Markup Language，XHTML），是一种置标语言，表现方式与超文本置标语言（HTML）类似，不过语法上更加严格。从继承关系上讲，HTML 是一种基于标准通用置标语言（SGML）的应用，是一種非常灵活的置標語言，而 XHTML 则基于可扩展置标语言（XML），XML 是 SGML 的一个子集。XHTML 1.0 在 2000 年 1 月 26 日成为 W3C 的推荐标准

SVL:

SMIL:

SMIL 是同步多媒体集成语言（Synchronized Multimedia Integration Language）的缩写,它是由 3W(World Wide Web Consortium)组织规定的多媒体操纵语言

HDML:

HDML(手持装置标示语言)——通常与 WML(无线标示语言)进行对比——是一种能够让手机或 PDA 可以上网浏览信息的一种语言

MXML:

MXML 是一种用于创建用户截面的功能强大的标记性语言

SOAP:

SOAP 是一种简单的基于 XML 的协议，它使应用程序通过 HTTP 来交换信息

XSLT:

XSLT 是扩展样式表转换语言（Extensible Stylesheet Language Transformations）的简称，这是一种对 XML 文档进行转化的语言，XSLT 中的 T 代表英语中的“转换”（transformation）。它是 XSL（Extensible stylesheet language）规范的一部分。XSL 规范的另外一部分是 XSL-FO（FO 代表格式化对象 Formatting Objects）。

WML:

Wireless Markup Language，缩写为 WML，是 WAP 规范指定的基于 XML 的基本内容格式，使用支持该规范的设备例如移动电话可以浏览 WML 的页面

EDI:EDI 是 Electronic Data Interchange 的缩写，即电子数据交换，它是一种利用计算机进行商务处理的方式

XmlDocument:

Xml 就提供了一种可以很方便的操作 xml 的类 XmlDocument，它使用起来非常容易，XmlDocument 其实就是一个简单的树

XmlTextReader:

表示提供对 XML 数据进行快速、非缓存、只进访问的读取器

RIA（Rich Internet Applications，富网络应用程序）技术：

是下一代网络应用的主流开发技术。流行的 RIA 开发技术术主要有 Macromedia 公司的 Flash/Flex、Microsoft 公司的 Avalon 技术、Mozilla 的 XUL 技术等。

Gecko:

Gecko 是套開放原始碼的、以 C++編寫的網頁排版引擎。目前為 Mozilla 家族網頁瀏覽器以及 Netscape 6 以後版本瀏覽器所使用

XAML：

XAML 是 eXtensible Application Markup Language 的英文缩写，相应的中文名称为可扩展应用程序标记语言，它是微软公司为构建应用程序用户界面而创建的一种新的描述性语言

version:

版本

encoding：

编码方式

ELEMENT：

元素

OrganizationChart：

组织系统图

chart：

图表

standalone：

独立

NMTOKEN:

名称标记

Enumerated:

枚举

NOTATION:

记号

REQUIRED:

必选

IMPLIED:

可以忽略的

FIXED:

固定的

XSD 元素

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。

XSD Restrictions/Facets for Datatypes

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。

Exetensible：

可扩展的　例：XSL（Exetensible Stylesheet Language）

Stylesheet：

样式表

Transformations：

转换　例：XSLT(Exetensible Stylesheet Language Transformations)

Formatting：

格式化

Objects：

对象

Choose：

选择

Sort：

排序

Templates：

模板

Value：

值

Select：

选择

Apply：

应用

Encoding：

编码格式 如：encoding=”GB2312”

Version：

版本 如 version=”1.0”

Framework：

框架 例：.net Freamework2.0

Serialization：

.net framework 中的一个命名空间的名字，序列化时会用到。System.Xml. Serialization

DOM(Document Object Model)：

文档对象模型

Node：

节点

Ancestors：

祖先

Descendants：

后代

Attribute：

属性

Element：

元素

Serizalizer：

序列化时使用的类名。

Binary：

二进制

XmlElement：

XML 元素

XmlNode：

XML 节点

附录 2 （参考答案）

习题与参考答案

第一次课

总与答：

1. 答：当今的网络世界，电子商务日渐兴盛，异质电脑系统之间的互动日益频繁（商务信息传递），自动化程序也愈来愈重要（譬如前述的爬虫程序），我们需要一个比目前 HTML 更好的方法，因为 HTML 主要是用来显示数据的，而 XML 主要是用来存储数据的，所以我们需要使用 XML，否则未来网络的发展将会越来越慢。

2. 答：HTML 主要是用来显示数据的

XML 是用来存放数据的

XML 不是 HTML 的替代品和升级品，XML 和 HTML 是两种不同用途的语言。

XML 是被设计用来描述数据的，重点是：什么是数据，如何存放数据。

HTML 是被设计用来显示数据的，重点是：显示数据以及如何显示数据更好上面。

HTML 是与显示信息相关的, XML 则是与描述信息相关的。

3. 答：

1.数据交换；2.Web 服务；3.内容管理；4.Web 集成；5.配置信息；6.图形图像；7.编程语言；

作业：

1. XML 的全称是：XML（eXtensible Markup Language）即可扩展标记语言

2. XML 主要是用来存储数据,HTML 主要是用来显示数据.

第二次课

问与答：

1. 答：XML 有结构良好的文档和有效的文档两种类型
2. 答：- 必须有 XML 声明语句
   - 必须有且仅有一个根元素
   - 标记大小写敏感
   - 属性值用引号
   - 标记成对
   - 空标记关闭
   - 元素正确嵌套
   - 名称不能以数字开头
   - 不能以 XML/xml/Xml/…开头
   - 名称中不能含空格
   - 名称中不能含冒号(注：冒号留给命名空间使用)

作业：

1：合法

2： a）不合法 不能以：开头

b）<TEST_XML>合法

c）<TEST XML>不合法 标记名称中间不能有空格

3： a）<BOOK>

<AUTHOR>abc……<AUTHOR>错误，该标记没有关闭

</BOOK>

b）<BOOK>

<AUTHOR>

</BOOK> //错误，标记嵌套不正确

</AUTHOR>

c）<BOOK>

<AUTHOR>abc……</AUTHOR>

</book> //错误，XML 中区分大小写

4：<root>

```
        <学院>
            <学院名称>Cedar 大学</学院名称>
            <新生人数>110</新生人数>
            <毕业生人数>103</毕业生人数>
            <变动人数>+7</变动人数>
        <学院>
        <学院>
            <学院名称> Elm  学院大学</学院名称>
            <新生人数>223</新生人数>
            <毕业生人数>214</毕业生人数>
            <变动人数>+9</变动人数>
        <学院>
        <学院>
            <学院名称> Maple  高等专科院校</学院名称>
            <新生人数>197</新生人数>
            <毕业生人数>120</毕业生人数>
         <变动人数>+77</变动人数>
    <学院>
</root>
```

5：

```
<root>
    <商品  名称=”图书” 所需数目=”1”/>
    <商品  名称=”铅笔” 所需数目=”3”/>
    <商品  名称=”笔记本” 所需数目=”1”/>
    <商品  名称=”便笺簿” 所需数目=”1”/>
<root>
```

第三次课

与答：

1：答：DTD 文档与 XML 文档实例的关系就像类与对象，DTD 是类，XML 是对象。他们的

关系还像数据库表结构与数据记录，DTD 是表的结构，XML 是表中的一条条记录。

作业

1. <属性列表>

```
    <联系人>
        <姓名>aa</姓名>
        <性别>男</性别>
        <EMAIL>yinhe@163.com</EMAIL>
        <电话>13987654322</电话>
        <电话>027-88888888</电话>
        <地址>
            <街道>洪山区</街道>
            <城市>武汉市</城市>
            <省份>湖北省</省份>
        </地址>
    </联系人>
    <联系人>
        <姓名>bb</姓名>
        <性别>男</性别>
        <EMAIL>yinhe@163.com</EMAIL>
        <电话>13987654311</电话>
        <电话>027-77777777</电话>
        <地址>
            <街道>洪山区</街道>
            <城市>武汉市</城市>
            <省份>湖北省</省份>
        </地址>
    </联系人>
</属性列表>
```

2. <?xml version="1.0" encoding="utf-8"?>

```
<!ELEMENT 学生名册 (学生+)>
<!ELEMENT 学生 (姓名, 性别, 年龄)>
<!ATTLIST 学生      学号 ID   #REQUIRED >
<!ELEMENT 姓名 (#PCDATA)>
<!ELEMENT 性别 (#PCDATA)>
<!ELEMENT 年龄 (#PCDATA)>
```

第四次课

问与答

1. 答：**Xmlns**——指定值作为ＸＭＬ的命名空间，他是声明空间必须的，而且可以附加在任何ＸＭＬ元素上。

Prefix——指定一个命名空间前缀。它（包括冒号）只用于声明一个命名空间前缀。如果使用了这个前缀，那么在文档中使用该前缀（prefix：element）的任何元素都被认为是位于声明命名空间范围内。

NamespaceURI——这是唯一的标识符。该值不必指向一个 Web 资源；它只有一个具有符号意义的标识符。这个值是必需的并且必须定义在单个引号或者双引号之内。可以使用两种不同的方式定义一个命名空间；

2. 答：XML Schema 是标准的 XML 文件,而 DTD 则使用自己的特殊语法,因此,只需要知道 XML 的语法规则就可以编写 Schema 了,不需要再学习其它语法规则.可以使用相同的处理器来解读,XML 文件与 XML Schema 文件.XML Schema 利用命名空间将文件中特殊的节点与 Schema 说明相联系,一个 XML 文件可以有多个对应的 Schema,若是 DTD 的话,一个 XML 文件只能有一个相对应的 DTD 文件.XML Schema 的内容模型是开放的,可以随意扩充,而 DTD 则无法解读扩充的内容.DTD 只能把文件类型定义为一个字符串,而 XML Schema 却允许把文件类型定义为整数,浮点数,字符串,布尔值或其他各种数据类型,而无须重新定义.XML Schema 相对 DTD 的明显优势是 XML Schema 文件本身也是 XML 文件,而不是像 DTD 那样使用特殊格式,因而方便了用户和设计者.因为他们可以使用相同的工具来处理与开发 XML Schema 和其他的 XML 数据,而不必用专门的开发或处理工具.

- xsd 相比 dtd 提供了元素和属性的数据类型的更多的控制
- xsd 能使你创建自己的数据类型，dtd 不可以
- xsd 可以对你指定的数据进行约束，比如你可以确保一个元素的内容是一个正整数值

3. 答：XML Schema 最重要的作用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

作业

1. 答：

UserInfo.xsd

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema id="UserInfo"
    targetNamespace="http://tempuri.org/UserInfo.xsd"
    elementFormDefault="qualified"
    xmlns="http://tempuri.org/UserInfo.xsd"
    xmlns:mstns="http://tempuri.org/UserInfo.xsd"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
>
  <xs:element name="学生">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="信息" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
```

```
                <xs:element name="姓名" type="xs:string" />
                <xs:element name="性别" type="xs:string" />
                <xs:element name="年龄" type="xs:int" />
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
</xs:schema>
```

UserInfo.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<学生 xmlns="http://tempuri.org/UserInfo.xsd">
  <信息>
    <姓名>张三</姓名>
    <性别>男</性别>
    <年龄>20</年龄>
  </信息>

  <信息>
    <姓名>李四</姓名>
    <性别>女</性别>
    <年龄>30</年龄>
  </信息>
</学生>
```

2. 答：

（1）MyUserInfo.xsd

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema id="MyUserInfo"
    targetNamespace="http://tempuri.org/MyUserInfo.xsd"
```

```
    elementFormDefault="qualified"
    xmlns="http://tempuri.org/MyUserInfo.xsd"
    xmlns:mstns="http://tempuri.org/MyUserInfo.xsd"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
>
  <xs:element name="学生信息">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="姓名" type="xs:string" />
        <xs:element name="年龄" type="xs:int" />
        <xs:element name="电子邮箱" type="xs:string" />
        <xs:element name="身高" type="xs:double" />
        <xs:element name="电话" type="xs:string" />
        <xs:element ref="单位信息"/>
       </xs:sequence>
      <xs:attribute name="编号" type="xs:int" />
    </xs:complexType>
  </xs:element>
  <xs:element name="单位信息">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="公司地址" type="xs:string" />
        <xs:element name="家庭住址" type="xs:string" />
        <xs:element name="邮编" type="xs:int" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

（2）MyUserInfo.xml

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

```
<学生信息 编号="1" xmlns="http://tempuri.org/MyUserInfo.xsd">
  <姓名>张学友</姓名>
  <年龄>30</年龄>
  <电子邮箱>abc@yinhe.com</电子邮箱>
  <身高>1.78</身高>
  <电话>13922348888</电话>
  <单位信息>
    <公司地址>武汉市洪山区新路村 1 号</公司地址>
    <家庭住址>武汉市洪山区新路村 2 号</家庭住址>
    <邮编>437000</邮编>
  </单位信息>
</学生信息>
```

第五次课

问与答

1. 答：简单类型，复杂类型.
2. 答：简单类型定义(simpleType)——定义数据的结构和范围，不允许含文档元素
3. 答：复杂类型定义(complexType)——定义文档的逻辑结构，允许含文档元素带属性
4. 答：利用提供的属性声明来控制属性的外观与内容，以约束元素信息项

- 限制元素信息项子元素为空、或符合指定的只有元素或混合内容模型，或限制字符信息项子元素符合指定的简单类型定义
- 使用类型定义层次机制从另一简单或复杂类型派生一个复杂类型
- 指定元素的后模式验证信息集分布(post-schema-validation infoset contributions)
- 限制从一个给定的复杂类型派生额外类型的能力
- 控制在一个实例中替换该内容模型，声明为一个给定复杂类型元素的派生类型元素的权限

作业

1. 答：MyStudent.xsd

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema id="MyStudent"
    targetNamespace="http://tempuri.org/MyStudent.xsd"
    elementFormDefault="qualified"
    xmlns="http://tempuri.org/MyStudent.xsd"
    xmlns:mstns="http://tempuri.org/MyStudent.xsd"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
>
  <xs:element name="学生名册">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="学生" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="姓名" type="xs:string" />
              <xs:element name="性别" type="xs:string" />
              <xs:element name="年龄" type="xs:int" />
            </xs:sequence>
            <xs:attribute name="学号" type="xs:int" />
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

2. 答：

```
<element name='联系人' type='联系人类型'/>
<complexType name='联系人类型'>
<element name='姓名' type='string'/>
<element name='性别' type='string'/>
<element name='年龄' type='int'/>
<element name=' EMAIL ' type='string'/>
<element name='性别' type='string'/>
<element name='电话' type='string'/>
<element name='地址' type='string'/>
</complexType>
```

第六次课

问与答

1. 答：为了在匹配 XML 文档结构时能够准确地找到某一个节点元素。可以把 XPath 比作文件管理路径，通过文件管理路径，可以按照一定的规则查找到所需要的文件；同样，依据 XPath 所制定的规则，也可以很方便地找到 XML 结构文档树中的任何一个节点，显然这对 XSLT 来说是一个最最基本的功能。

2. 答：XPath 中对元素和属性的匹配，主要有以下几种：

- 定位节点
- 选择未知元素
- 选择分支
- 选择属性

第七次课

问与答

1. 答：<?xml-stylesheet type="text/xsl" href="abc.xsl"?>

2. 答：

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:fo="http://www.w3.org/1999/XSL/Format"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:fn="http://www.w3.org/2005/xpath-functions"
xmlns:xdt="http://www.w3.org/2005/xpath-datatypes">
  <xsl:template match="/">
    <html>
      <head>
        <meta http-equiv="Content-Type" content="text/html;
charset=gb2312" />
        <title>成绩单</title>
        <style type="text/css">
          .Title {
          font-family: "宋体";
          font-size: 14px;
          font-weight: bolder;
          color: #FFFFFF;
          text-decoration: none;
          }
        </style>
      </head>
      <body>
        <xsl:apply-templates select="document"/>
      </body>
    </html>
  </xsl:template>
  <!--document 节点的定义-->
```

```
  <xsl:template match="document">
    <table width="600" border="0" align="center" cellpadding="0"
cellspacing="1" bgcolor="#1768FF">
      <tr>
        <td width="25%" height="30" align="center" bgcolor="#A4C4FF"
class="Title">学生姓名</td>
        <td width="25%" height="30" align="center" bgcolor="#A4C4FF"
class="Title">英语</td>
        <td width="25%" height="30" align="center" bgcolor="#A4C4FF"
class="Title">数学</td>
        <td width="25%" height="30" align="center" bgcolor="#A4C4FF"
class="Title">SQL 数据库</td>
      </tr>
      <xsl:apply-templates select="grade"/>
    </table>
  </xsl:template>
  <!--grade 节点的定义-->
  <xsl:template match="grade" >
    <tr>
      <td width="25%" height="25" align="center" bgcolor="#D9E7FF">
        <xsl:value-of select="name"/>
      </td>
      <td width="25%" height="25" align="center" bgcolor="#D9E7FF">
        <xsl:apply-templates select="english" />
      </td>
      <td width="25%" height="25" align="center" bgcolor="#D9E7FF">
        <xsl:apply-templates select="math"/>
      </td>
      <td width="25%" height="25" align="center" bgcolor="#D9E7FF">
        <xsl:apply-templates select="sql" />
```

```
      </td>
    </tr>
  </xsl:template>
  <xsl:template match="english|math|sql">
    <xsl:choose>

      <xsl:when test=".&gt;90">优秀</xsl:when>
      <xsl:when test=".&gt;80">良好</xsl:when>
      <xsl:when test=".&gt;60">及格</xsl:when>

      <xsl:otherwise >不及格</xsl:otherwise>
    </xsl:choose>
  </xsl:template>
</xsl:stylesheet>
```

第八次课

问与答

1. 答：（略）

第九次课

问与答

1. 答：

```
XmlDocument xml = new XmlDocument();
      //加载 xml 文档
      xml.Load(Server.MapPath("~/student.xml"));
      //利用 xpath 表达式找到  第 2 个"信息"节点里的 "年龄"子节点
```

```
        XmlNode node1 = xml.SelectSingleNode("//信息[2]/年龄");
      //修改年龄为 18 岁
        node1.InnerText = "18";
        //保存
  xml.Save(Server.MapPath("~/student.xml"));
```

第十次课

问与答

1. 答：①、以某种存储形式使自定义对象持久化；②、将对象从一个地方传递到另一个地方。

2. 答：.NET Framework 提供两种序列化技术：

① 二进制序列化保持类型保真度，这对于在应用程序的不同调用之间保留对象的状态很有用。例如，通过将对象序列化到剪贴板，可在不同的应用程序之间共享对象。您可以将对象序列化到流、磁盘、内存和网络等等。

②XML 序列化仅序列化公共属性和字段，且不保持类型保真度。当您要提供或使用数据而不限制使用该数据的应用程序时，这一点是很有用的。