XSLT 轻松入门

目　录

XSLT 轻松入门提纲’

提纲
本文共分五大章：XSLT 概念，XSLT 的实例，XSLT 的元素语法，XPath 的语法，XSLT 的资源。
1.XSLT 的概念
1.1 什么是 XSLT
1.2 为什么要用 XSLT
1.3 XSLT 的历史
1.4 什么是 XPath
1.5 XSLT 和 CSS 的比较
1.6 XSLT 和 IE5
2. XSLT 的实例
2.1 XSLT 如何转换 XML
2.2 一个实例
2.3 过程解析
2.4 XSLT 的用途
3.XSLT 的元素语法
3.1 xsl:template 和 xsl:apply-templates
3.2 xsl:value-of
3.3 xsl:for-each
3.4 xsl:if
3.5 Xxsl:choose, when, otherwise
3.6 xsl:sort
4.XPath 的语法
4.1 当前位置
4.2 寻址操作
4.3 运算符
4.4 功能函数

XSLT 轻松入门第一章：XSLT 概念

提纲

本文共分五大章：XSLT 概念，XSLT 的实例，XSLT 的元素语法，XPath 的语法，XSLT 的资源。

1.XSLT 的概念

1.1 什么是 XSLT

1.2 为什么要用 XSLT

1.3 XSLT 的历史

1.4 什么是 XPath

1.5 XSLT 和 CSS 的比较

1.6 XSLT 和 IE5

1.XSLT 的概念

我们首先来澄清一个概念，大家可能听说过 XSL(eXtensible Stylesheet Language)，XSL 和我们这里说的 XSLT 从狭义上理解是一样的,而按照 W3C 的标准，XSLT 的说法更严格些，因此我们在文章中统一使用 XSLT 的称法。它们之间具体的关系我们会在下面讲述。

1.1 什么是 XSLT

XSLT 的英文标准名称为 eXtensible Stylesheet Language Transformation。根据 W3C 的规范说明书(http://www.w3.org/TR/xslt)，最早设计 XSLT 的用意是帮助 XML 文档(document)转换为其它文档。但是随着发展，XSLT 已不仅仅用于将 XML 转换为 HTML 或其它文本格式，更全面的定义应该是：

XSLT 是一种用来转换 XML 文档结构的语言。

1.2 为什么要用 XSLT

我们已经知道，XML 是一种电脑程序间交换原始数据的简单而标准的方法。它的成功并不在于它容易被人们书写和阅读，更重要的是，它从根本上解决了应用系统间的信息交换。因为 XML 满足了两个基本的需求：

(1).将数据和表达形式分离。就象天气预报的信息可以显示在不同的设备上，电视，手机或者其它。

(2).在不同的应用之间传输数据。电子商务数据交换的与日俱增使得这种需求越来越紧迫。

为了使数据便于人们的阅读理解，我们需要将信息显示出来或者打印出来，例如将数据变成一个 HTML 文件，一个 PDF 文件，甚至是一段声音；同样，为了使数据适合不同的应用程序，我们必须有能够将一种数据格式转换为另一种数据格式，比如需求格式可能是一个文本文件，一个 SQL 语句，一个 HTTP 信息，一定顺序的数据调用等。而 XSLT 就是我们用来实现这种转换功能的语言。将 XML 转换为 HTML，是目前 XSLT 最主要的功能。

1.3 XSLT 的历史

想很多其他 XML 家族成员一样，XSLT 是由 W3C 起草和制定的。它的主要发展历程如下：

.1995 年由 James Clark 提议；

.1997 年 8 月正式提案为 XSL；

.1998 年 5 月由 Norman Walsh 完成需求概要；

.1998 年 8 月 18 日 XSL 草案发布；

.1999 年 11 月 16 日正式发布 XSL 1.0 推荐版本。

目前，XSLT 仍然在快速的发展中，XSLT1.1 的草案已经可以在 W3C 网站(http://www.w3.org/TR/xslt11)上看到。

1.4 什么是 XPath

XPath 是 XSLT 的重要组成部分，我们将在第四章讲解它的详细语法。那么 XPath 是什么呢？我们首先来了解一下 XSL 系列的"家族"关系。如下图：

XSL 在转换 XML 文档时分为明显的两个过程，第一转换文档结构；其次将文档格式化输出。这两步可以分离开来并单独处理，因此 XSL 在发展过程中逐渐分裂为 XSLT(结构转换)和 XSL-FO(formatting objects)(格式化输出)两种分支语言，其中 XSL-FO 的作用就类似 CSS 在 HTML 中的作用。而我们这里重点讨论的是第一步的转换过程，也就是 XSLT。

另外，在学习 XML 时我们已经知道 XML 是一个完整的树结构文档。在转换 XML 文档时可能需要处理其中的一部分(节点)数据，那么如何查找和定位 XML 文档中的信息呢，XPath 就是一种专门用来在 XML 文档中查找信息的语言。XPath 隶属 XSLT，因此我们通常会将 XSLT 语法和 XPath 语法混在一起说。

用一种比较好理解的解释：如果将 XML 文档看作一个数据库，XPath 就是 SQL 查询语言；如果将 XML 文档看成 DOS 目录结构，XPath 就是 cd,dir 等目录操作命令的集合。

1.5 XSLT 和 CSS 的比较

CSS 同样可以格式化 XML 文档，那么有了 CSS 为什么还需要 XSLT 呢？因为 CSS 虽然能够很好的控制输出的样式，比如色彩，字体，大小等，但是它有严重的局限性，就是：

(1) CSS 不能重新排序文档中的元素；

(2) CSS 不能判断和控制哪个元素被显示，哪个不被显示；

(3) CSS 不能统计计算元素中的数据；

换句话说，CSS 只适合用于输出比较固定的最终文档。CSS 的优点是简洁，消耗系统资源少；而 XSLT 虽然功能强大，但因为要重新索引 XML 结构树，所以消耗内存比较多。

因此，我们常常将它们结合起来使用，比如在服务器端用 XSLT 处理文档，在客户端用 CSS 来控制显示。可以减少响应时间。

1.6 XSLT 和 IE5

在 XSLT 草案发布不久，微软就在 IE4 中提供了支持 XSL 功能的预览版本，到 IE5.0 发布时，正式全面支持 XSLT，可是由于 IE5 发布的比 XSLT1.0 标准时间早，因此在 IE5.0 中支持的 XSTL 功能和 XSLT 1.0 略有不同。(呵呵～～XML 推行的主要原因之一就是解决 HTML

过分依赖浏览器的问题，现在微软又想标新立异？)。好在微软的 IE5.5 中执行的标准已经和 W3C 的 XSLT1.0 基本相近。但令人头疼的是 IE5.0 已经发行了几百万套，您使用的 XSLT 很可能不能被客户的浏览器正确执行。目前 XSLT 1.1 仍在发展中，W3C 及有关组织也在和微软协商争取获得统一。呵呵~~故事还远远没有结束噢。

注意：本文中提到的语法都是根据 XSLT 1.0 的标准来讲的，没有任何微软的"方言"。

1.3 XSLT 的历史

想很多其他 XML 家族成员一样，XSLT 是由 W3C 起草和制定的。它的主要发展历程如下：

.1995 年由 James Clark 提议；

.1997 年 8 月正式提案为 XSL；

.1998 年 5 月由 Norman Walsh 完成需求概要；

.1998 年 8 月 18 日 XSL 草案发布；

.1999 年 11 月 16 日正式发布 XSL 1.0 推荐版本。

目前，XSLT 仍然在快速的发展中，XSLT1.1 的草案已经可以在 W3C 网站(http://www.w3.org/TR/xslt11)上看到。

1.4 什么是 XPath

XPath 是 XSLT 的重要组成部分，我们将在第四章讲解它的详细语法。那么 XPath 是什么呢？我们首先来了解一下 XSL 系列的"家族"关系。如下图：

XSL 在转换 XML 文档时分为明显的两个过程，第一转换文档结构；其次将文档格式化输出。这两步可以分离开来并单独处理，因此 XSL 在发展过程中逐渐分裂为 XSLT(结构转换)和 XSL-FO(formatting objects)(格式化输出)两种分支语言，其中 XSL-FO 的作用就类似 CSS 在 HTML 中的作用。而我们这里重点讨论的是第一步的转换过程，也就是 XSLT。

另外，在学习 XML 时我们已经知道 XML 是一个完整的树结构文档。在转换 XML 文档时可能需要处理其中的一部分(节点)数据，那么如何查找和定位 XML 文档中的信息呢，XPath 就是一种专门用来在 XML 文档中查找信息的语言。XPath 隶属 XSLT，因此我们通常会将 XSLT 语法和 XPath 语法混在一起说。

用一种比较好理解的解释：如果将 XML 文档看作一个数据库，XPath 就是 SQL 查询语言；如果将 XML 文档看成 DOS 目录结构，XPath 就是 cd,dir 等目录操作命令的集合。

1.5 XSLT 和 CSS 的比较

CSS 同样可以格式化 XML 文档，那么有了 CSS 为什么还需要 XSLT 呢？因为 CSS 虽然能够很好的控制输出的样式，比如色彩，字体，大小等，但是它有严重的局限性，就是：

(1) CSS 不能重新排序文档中的元素；

(2) CSS 不能判断和控制哪个元素被显示，哪个不被显示；

(3) CSS 不能统计计算元素中的数据；

换句话说，CSS 只适合用于输出比较固定的最终文档。CSS 的优点是简洁，消耗系统资源少；而 XSLT 虽然功能强大，但因为要重新索引 XML 结构树，所以消耗内存比较多。

因此，我们常常将它们结合起来使用，比如在服务器端用 XSLT 处理文档，在客户端用 CSS 来控制显示。可以减少响应时间。

1.6 XSLT 和 IE5

在 XSLT 草案发布不久，微软就在 IE4 中提供了支持 XSL 功能的预览版本，到 IE5.0 发布时，正式全面支持 XSLT，可是由于 IE5 发布的比 XSLT1.0 标准时间早，因此在 IE5.0 中支持的 XSTL 功能和 XSLT 1.0 略有不同。(呵呵~~XML 推行的主要原因之一就是解决 HTML 过分依赖浏览器的问题，现在微软又想标新立异？)。好在微软的 IE5.5 中执行的标准已经和 W3C 的 XSLT1.0 基本相近。但令人头疼的是 IE5.0 已经发行了几百万套，您使用的 XSLT 很可能不能被客户的浏览器正确执行。目前 XSLT 1.1 仍在发展中，W3C 及有关组织也在和微软协商争取获得统一。呵呵~~故事还远远没有结束噢。

注意：本文中提到的语法都是根据 XSLT 1.0 的标准来讲的，没有任何微软的"方言"。

XSLT 轻松入门第二章：XSLT 的实例

2. XSLT 的实例
2.1 XSLT 如何转换 XML
2.2 一个实例
2.3 过程解析
2.4 XSLT 的用途
2.1 XSLT 如何转换 XML
我们打个有趣的比方，你玩过橡皮泥吧，用不同的模子按上去，就可以做出需要的形状。如果我们假设 XML 数据文档是一块大橡皮泥，XSLT 就象是一个模子，用力一按，就做出需要的形状来---符合不同需要的 HTML 文档。

我们将 XML 原文档输入，用 XSL 作为模板，通过转换引擎，最终输出需要的 HTML 文档。其中的转换引擎就是比喻中"用力一按"的过程。在具体应用中，有专门的软件来实现这个转换过程，名为 XML Processor。目前已经有很多 Processor 软件(下面将详细提到)，在 IE5.5 中也已经内嵌了 XML Processor。
2.2 一个实例

现在我们来看一个简单的 XSLT 实际应用例子，获得一些感官上的认识。很多网页设计师看到类似 HTML 的代码才会放心，代码是那样的亲切和熟悉。

例子 1: "Hello, world!"
hello world 作为第一个教程已经是程序语言中的惯例了。我们也遵守这个惯例，看看如何利用 XSLT 来显示"hello world"。虽然这个例子没有什么实际用途，但是请大家不要急，后面还有更详细的例子。
第一步：建立要输入 XML 文档 hello.xml。

```
<?xml version="1.0" encoding="iso-8859-1"?>
<greeting>Hello, world!</greeting>
```

这是一个很简单 XML 文档，只包含一个节点的 XML 结构树。
第二步：建立 XSLT 文档 hello.xsl。提示：默认的 XSLT 文件的后缀名为.xsl。

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
<html>
<head>
<title>First XSLT example</title>
</head>
<body>
<p><xsl:value-of select="greeting"/></p>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

你现在可以用 IE5.0 以上版本浏览器打开这个 hello.xsl 文件，看到 XSL 的结构树。
第三步：在 XML 中调用这个 XSL 文件。修改 hello.xml 的代码为：

```
<?xml version="1.0" encoding="iso-8859-1"?>
<?xml-stylesheet type="text/xsl" href="hello.xsl"?>
<greeting>Hello, world!</greeting>
```

ok,到这一步原理上已经完成了所有的代码，接下来只要用一个 XSLT 的处理器(XML Processor)来执行 hello.xml 就可以看到"hello world"的显示结果了。流行的处理器软件有以下几种：
(1).James Clark 的 XT。下载网址：http://www.jclark.com/xml/xt.html
(2).IBM 的 XML for Java 软件包，名为 LotusXSL。下载网址：www.alphaworks.ibm.com/tech/xml4j
(3).Saxon。下载网址：http://www.wrox.com
(4).微软的 MSXML3。下载网址：http://www.microsoft.com/xml

有网友要问，我想在浏览器中看到"hello world"的效果应该怎么做？在微软的 IE5.5 内嵌了 MSXML3 解释器，你可以用 IE5.5 打开 hello.xml 文件，就可以看到结果。如果只看到 XML 结构树，不是单独的"hello world"字样，说明你的浏览器没有安装 MSXML3 版本。
如果没有安装又想看效果怎么办？那还是用我们在 XML 教程中的老办法，采用 JS 实现。(这已经超出了本文要讲的范围，但为了更加直观，便于理解，我们在这里提供实例代码。)下面是一种实现的代码，可以保存为 hello.htm，和上面的 hello.xml，hello.xsl 放在同一目录下面。最后用 IE5.0 以上版本打开 hello.htm 就可以看到效果了。

```
<html>
<head>
<script language="JavaScript" for="window" event="onload">
var xmlDoc = new ActiveXObject("Microsoft.XMLDOM");
xmlDoc.async="false";
xmlDoc.load("hello.xml");
nodes = xmlDoc.documentElement.childNodes;
greeting.innerText = nodes.item(0).text;
</script>
<title>First XSLT Example</title>
</head>
<body bgcolor="#FFFFFF">
<span id="greeting"></span><br>
</body>
</html>
```

2.3 过程解析

如果你成功的看到效果，你也许想知道这些代码的具体含义，我们来详细解说：看 hello.xsl 文件

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

这是标准的 XML 文档的首行代码，因为 XSLT 本身也是 XML 文档。encoding 属性用来定义文档使用的编码形式，iso-8859-1 主要支持西欧和北美的语言编码。如果你想使用简体中文，那么就应该写成：

```
<?xml version="1.0" encoding="GB2312"?>
```

接下去的代码是:

```
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
```

这是标准的 XSLT 文件首行代码。xsl:stylesheet 代码的意思是将文档作为一个样式表来(stylesheet)处理。xmlns:xsl 属性是一个名字空间声明，和 XML 中的名字空间使用方法一样，用来防止元素名称重复和混乱。其中前缀 xsl 的意思是文档中使用的元素遵守 W3C 的 XSLT 规范。最后的 version 属性说明样式表只采用 XSLT
1.0 的标准功能，这也是目前仅有的标准。

```
<xsl:template match="/">
```

一个<xsl:template>元素定义一个模板规则。属性 match="/"说明 XML 源文档中，这个模板规则作用的起点。"/"是一种 XPath 语法，我们在后面将详细讲述，这里的"/"代表 XML 结构树的根(root)。

接下去的代码是:

```
<html>
<head>
<title>First XSLT Example</title>
</head>
<body>
<p><xsl:value-of select="greeting"/></p>
</body>
</html>
```

说明：当模板规则被触发，模板的内容就会控制输出的结果。例子中，模板大部分内容由 HTML 元素和文本构成。只有<xsl:value-of>元素是 XSLT 语法，这里<xsl:value-of>的作用是拷贝原文档中的一个节点的值到输出文档。而 select 属性则详细指定要处理的节点名称。这是 XPath 语法，"greeting"的意思就是寻找根节点名为 greeting 的元素，并用模板来处理这个节点。具体的就是找到<greeting>元素，然后将元素的值"hello
world"按模板样式拷贝到输出文件。
提示：由于 XML 文档是严格的层级结构(用 IE5 查看 XML 文件，会看见 XML 文档类似多级关联菜单)，所以我们形象的称 XML 文档为文档树，其中每一对元素称作树的一个节点。根元素就是根节点。

最后关闭所有元素：

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

好，例子解说完毕。你是否想过为什么要用这么复杂的方法来显示"hello world"呢？
关键不在表面，而在于实质：用这种方法，hello world 可以从 XML 文档中被提取出来，并用各种不同的 XSLT 模板处理，来输出不同需求的文档。我们来看看 XSLT 的主要用途：

2.4 XSLT 的用途

XSLT 的主要用途就是数据转换应用。
由于以 XML 为基础的电子商务广泛普及，XSLT 作为数据转换的角色也越来越重要。例如直接将电视新闻的数据格式转换成报纸新闻需要的数据格式;将股票数据直接转换成图片显示在网页上；对 EDI(电子数据交换)数据进行统计，排序等等。
XSLT 是处理类似工作的理想工具。

XSLT 轻松入门第三章：XSLT 的元素语法

通过前面两章的介绍，我们已经对 XSLT 的基本概念和它的转换过程有了一些了解。下面我们一起来学习 XSLT 的具体语法。说到语法总是比较枯燥的，您可以大体上浏览一遍，等您真正需要使用 XSLT 的时候，再仔细研究它们。
3.XSLT 的元素语法
3.1 xsl:template 和 xsl:apply-templates
3.2 xsl:value-of
3.3 xsl:for-each
3.4 xsl:if
3.5 Xxsl:choose, when, otherwise
3.6 xsl:sort
3.1 xsl:template 和 xsl:apply-templates

模板(template)是 XSLT 中最重要的概念之一。XSLT 文件就是由一个一个的模板组成，任何一个 XSLT 文件至少包含一个模板。模板的概念就象是搭积木；你如果是程序员，也可以将模板看作一个方法，一个类，或者一个模块。它们可以被拼装组合，也可以单独成块，不同的模板控制不同的输出格式。
模板(template)由两部分组成：匹配模式(match pattern)和执行。简单的讲模式定义 XML 源文档中哪一个节点将被模板处理，执行则定义输出的是什么格式。两部分对应的语法为 xsl:template 和 xsl:apply-templates。

xsl:template 的语法是：

```
<xsl:template
match = pattern
name = qname
priority = number
mode = qname>
<!-- 执行内容 -->
</xsl:template>
```

xsl:template 的作用是定义一个新模板。属性中 name,priority,和 mode 用来区别匹配同一节点的不同模板。它们不是常用的属性。match 属性则控制模板的匹配模式(pattern)，匹配模式是用来定位 XML 源文档中哪一个节点被模板处理。一个模板匹配一个节点。我们用一个例子来帮助理解：
假设我们要处理一个包含章节和段落文档。我们用 para 元素定义段落，用 chapter 元素定义章节。我们来看看 match 属性可能的值。下面的语句写法说明模板匹配所有的 para 元素

```
<xsl:template match="para">
</xsl:template>
```

下面的语句写法说明模板匹配所有的 para 元素和所有的 chapter 元素：

```
<xsl:template match="(chapter|para)">
</xsl:template>
```

下面的语句写法说明模板匹配所有的父节点为 chapter 元素的 para 元素：

```
<xsl:template match="chapter//para">
</xsl:template>
```

下面的语句写法说明模板匹配根节点:

```
<xsl:template match="/">
</xsl:template>
```

我们再来看 apply-templates 语法：

```
<xsl:apply-templates
select = node set-expression
mode = qname>
</xsl:apply-templates>
```

xsl:apply-templates 用来执行那一个节点被模板具体处理。你可以将它理解为程序中调用子函数。select 属性用来定义确切的节点名称。xsl:apply-templates 总是包含在 xsl:template 元素中，象这样：

```
<xsl:template match="/">
<xsl:apply-templates select="para"/>
</xsl:template>
```

这段代码说明摸板匹配整个文档(根节点)，具体执行时处理根节点下所有 para 元素。

```
<xsl:template match="para">
<p><xsl:apply-templates/></p>
</xsl:template>
```

而这一段代码则表示摸板匹配 para 节点，所有 para 下的子元素都将被处理。

3.2 xsl:value-of

XSL:value-of 用来将源文档中元素的文本值写到输出文档中。例如：
有一个个人资料的 XML 文档：

```
<?xml version="1.0" encoding="iso-8859-1"?>
<PERSON>
<name>ajie</name>
<age>28</age>
</PERSON>
```

我如果想在输出文档中显示上面这个 XML 源文档中的 name 元素的值，可以这样写 XSLT 代码：

```
<xsl:template match="PERSON">
<xsl:value-of select="name"/>
</xsl:template>
```

执行后，你会看到"ajie"被单独显示出来。其中 match="PERSON"定义摸板匹配 PERSON 节点，xsl:value-of
语法说明需要输出一个节点的值，而 select="name"则定义需要被输出的元素为 name。看这个过程是不是和数据库里查询一个人的名字很象？当然，xsl:value-of 查询还有更多，更复杂的语法，因为是涉及寻找和定位的功能，我们会放在后面的 XPath 语法中在仔细讲解。同样功能的还有 xsl:copy-of，用法一样，就不重复解释了。

3.3 xsl:for-each

xsl:for-each 语法允许你循环处理被选择的节点。例如：有一个含多个个人资料的 XML 文档：

```
<?xml version="1.0" encoding="iso-8859-1"?>
<PEOPLE>
<PERSON>
<name>ajie</name>
<age>28</age>
</PERSON>
<PERSON>
<name>tom</name>
<age>24</age>
</PERSON>
<PERSON>
<name>miake</name>
<age>30</age>
</PERSON>
</PEOPLE>
```

我需要显示所有人的姓名，则可以将 XSLT 代码写成：

```
<xsl:template match="PEOPLE">
<xsl:for-each select="child::PERSON">
<xsl:value-of select="name"/>
</ xsl:for-each>
</xsl:template>
```

3.4 xsl:if

xsl:if 类似普通程序语言的 if 条件语句，允许设定节点满足某个条件时，被模板处理。xsl:if 的语法格式为：

```
<xsl:if test=布尔表达式>
template body
</xsl:if>
```

例如：

```
<xsl:template match="PEOPLE">
<xsl:if test="@name">
<p><xsl:value-of select="@name"/></p>
</xsl:if>
</xsl:template>
```

这段代码的意思是检测 PEOPLE 节点下所有的属性，如果发现有<name>属性，则将<name>属性的值输出。其中@符号是属性的统配符，表示该节点下所有的属性。

3.5 xsl:choose, xsl:when 和 xsl:otherwise

xsl:if 语法没有 else 的属性。如果我们要进行多项选择，那么就要使用 xsl:choose / xsl:when / xsl:otherwise 系列流程控制语法了。具体的使用请看下面的 XSL 文件例子：

```
<xsl:template match="PEOPLE">
<xsl:choose>
<xsl:when test="@name = 'ajie'">
<B><xsl:value-of select="@name"/></B>
</xsl:when>
<xsl:when test="@name">
<I><xsl:value-of select="@name"/></I>
</xsl:when>
<xsl:otherwise>
No name available
</xsl:otherwise>
<xsl:choose>
</xsl:template>
```

说明：首先在 PEOPLE 节点下寻找<name>属性值为 ajie 的元素，如果找到，将 ajie 用粗体输出；如果没有发现值为 ajie 的<name>元素，则将所有的<name>元素的值都用斜体输出；如果没有发现任何<name>元素，则显示"No name available"。

3.6 xsl:sort

在 XSLT 中可以对 XML 源文档的元素进行重新排序，排序的语法就是 xsl:sort。举例：下面的代码就是将文档元素按 name 排序。

```
<xsl:template match="PEOPLE">
<xsl:apply-templates select="PERSON">
<xsl:sort select="@name"/>
</xsl:apply-templates>
</xsl:template>
```

以上是 XSLT 的元素的主要语法，还有很多其他的语法，例如：import, include, element, attribute, number, param 等等语法，在这里就不一一解释。我们的目的是让您对 XSLT 的语法有基本的概念，理解 XSLT 作为一种转换语言的强大功能。

XSLT 轻松入门第四章：XPath 的语法

4.XPath 的语法
我们在前面已经提到过，XPath 是用来帮助 XSLT 在 XML 源文档中查找定位信息的语言。在实际使用过程中，XPath 和 XSLT 总是混在一起使用，在上面一章的语法例子中我们已经有使用到 XPath 的语法，只是没有明确点出。但 W3C 将它们分成两个标准，所以我们也将它们拆成两章来讲解。
4.XPath 的语法
4.1 当前位置
4.2 寻址操作
4.3 运算符
4.4 功能函数
4.1 当前位置
当我们使用 XSLT 处理 XML 源文档是，我们用 Context 来表示当前正在被模板处理的节点位置。比如 xsl:template match="/"语句中表示 Context 在文档的根(root)节点。我不知道如何准确的翻译 Context 这个词，它类似于 C 语言里的指针，表示程序当前运行的位置。理解 Context 对于正确处理 XSL 模板非常重要，当您的 XSL 模板输出的文档和您想要的不一样，最先应该分析的就是 Context 在哪里。
Location Paths 是用于设定你想要寻找的 Context 节点位置。就类似 DOS 的目录命令。我们看个例子
<xsl:for-each select="child::PEOPLE/descendant::PERSON">
其中 child::PEOPLE/descendant::PERSON 就是 XPath 语法，这个表达式就是一个 Location Paths，代码说明要显示所有 PEOPLE 元素的子元素和所有 PERSON 元素的子元素。通常我们会采用更简单的写法：
<xsl:for-each select="PEOPLE//PERSON">
我们来解释 path 的两种表示方法："/"和"//"。
"/"是表示当前文档的节点,类似 DOS 目录分割符。例如：/PEOPLE 表示选择根节点下的 PEOPLE 元素；PEOPLE/PERSON 表示选择 PEOPLE 元素下所有的 PESON 子元素。
"//"则表示当前文档所有的节点。类似查看整个目录。例如：//PEOPLE 表示选择文档中所有的 PEOPLE 元素，无论它在什么层次；PEOPLE//PERSON 表示在 PEOPLE 元素下所有的 PERSON 元素，无论它的层次多深。

4.2 寻址操作
Axis 和 Predicate 是 XPath 语法中对 Location Paths 进行定位操作的语法，具体的用法列表如下
Axis 语法表

表达式 简写 说明
---
self . 选择当前的节点.。
例子 :
<TD><xsl:value-of select="."/></TD>
代码表示在当前位置插入当前的节点包含的文本(text)值，
---
parent .. 选择当前节点的父节点。
---
attribute @ 选择一个元素的所有属性。
例子:
<TD><xsl:value-of select="@PERSONID"/></TD>
选择 PERSON 元素的所有属性.
---
child 选择当前节点的所有子元素。
---
ancestor 选择当前节点的所有父元素(包括父元素的父元素，类推)

Axis 帮助我们选择当前节点周围所有的节点，而 Predicate 则用来定位当前节点内部的元素。
表示方法为方括号[]中加表达式：[Expression]。具体举例如下:
PERSON[position()=2]
这句代码表示寻找第二个"PERSON" 元素
PERSON[starts-with(name, "B")]
这句代码表示寻找所有名称以"B"开头的 PERSON 元素。
4.3 运算符
这一节介绍 XPath 的运算符(Expressions)，列表如下：

运算符 说明
---
and, or 就是普通意义的 and, or

= 等于

!= 不等于

>, >= 大于，大于等于

<, <= 小于，小于等于。注意：在 XSL 文件中，<符号要用< 表示

+, -, *, div 加减乘除
---
mod 取模

| 两个节点一起计算

4.4 功能函数(Functions)
在 XPath 里有很多功能函数可以帮助我们精确寻找需要的节点。
count()功能

作用：统计计数，返回符合条件的节点的个数。
举例： <xsl:value-of select="count(PERSON[name=tom])"/>
说明：代码的用途是显示 PERSON 元素中姓名属性值为 tom 有几个。
number()功能
作用：将属性的值中的文本转换为数值。
举例： The number is: <xsl:value-of select="number(book/price)"/>
说明：代码的用途是显示书的价格。
substring() 功能
语法：substring(value, start, length)
作用：截取字符串。
举例： <xsl:value-of select="substring(name, 1, 3)"/>
说明：代码的用途是截取 name 元素的值，从第一个字母开始显示到第三个。
sum()功能
作用：求和。
举例： Total Price = <xsl:value-of select="sum(//price)"/>
说明：代码的用途是计算所有价格的和。
上面这些功能只是 XPath 语法中的一部分，还有大量的功能函数没有介绍，而且目前 XPath 的语法仍然在不断发展中。通过这些函数我们可以实现更加复杂的查询和操作。
看到这里，我们的入门教程就快结束了。通过走马观花式的快速学习，希望大家对 XSLT 应该有了一点基本概念：XSLT 是一种转换 XML 文档的语言，它包含两个过程：转换和格式化。XSLT 的功能比 CSS 强大得多，它有类似数据查询的语法。如果您对 XSLT 感兴趣，那么以上的知识是远远不够的，需要查询更多的资料。阿捷在最后一章附录为大家提供了主要的 XSLT 资源。

XSLT 轻松入门第五章：XSLT 的资源

5.附录：XSLT 的资源
◇ 最权威的网站
http://www.w3.org/Style/XSL/
◇ 有关标准
XSLT1.0 http://www.w3.org/TR/xslt.html
XSLT1.1 http://www.w3.org/TR/xslt11/
XPath1.0 http://www.w3.org/TR/xpath.html
◇ 学习教程
http://www.w3schools.com/xsl/
http://www.wirelessdevnet.com/channels/wap/training/xslt.html
http://msdn.microsoft.com/library/default.asp?URL=/library/psdk/xmlsdk/xslp8tlx.htm
http://www.jenitennison.com/xslt/
http://www.arbortext.com/xsl/
◇ 相关资讯
http://www.xslinfo.com/
http://www.xslt.com/
http://www.xml.com
http://www.oasis-open.org/cover/xsl.html#resources
http://www.xml.org
http://www.ibm.com/developer/xml
http://www.biztalk.org
◇ 相关软件
http://www.xmlsoftware.com/xsl/
◇ 邮件列表
http://www.15seconds.com/faq/XML/748.htm

http://www.mulberrytech.com/xsl/xsl-list/index.html

XSL 概述

XSL(eXtensible Stylesheet Language,可扩展样式语言)是为 XML 文件定义的一种标识语言，它将提供远远超过 CSS 的强大功能，如将元素再排序等。实际上简单的 XML 已可被 CSS 所解释，然而复杂的高度结构化的 XML 数据或 XML 文档则只能依赖于 XSL 极强的格式化的能力而现给用户。

XSL 以包含了一套元素集的 XML 语法规范而定义，该语法规范将被用来把 XML 文件转换成 HTML 文件或 XML 文档。一个 XSL 样式表集合了一系列设计规则以用于将信息从 XML 文件中汲取出，并将其转换成 HTML 等其它格式。这种转换将采用一种公开的方式，使其更加容易方便地被程序员描述。而且 XSL 还将提供多种脚本语言的通道以满足更为复杂的应用需求，因此尽管 XSL 是一项新的标识语言，但程序员完全可以继续充分发挥其所熟练的 HTML 或脚本语言的优势。XSL 凭借其可扩展性能够控制无穷无尽的标签,而控制每个标签的方式也是无穷尽的。这就给 Web 提供了高级的布局特性。例如旋转的文本、多列和独立区域。它支持国际书写格式,可以在一页上混合使用从左至右、从右至左和从上至下的书写格式。

XSL 能使 Web 浏览器直接根据用户的不同需求改变文档的表示法,例如数据的显示顺序改变,从而不需要再与服务器进行交互通信。通过变换样式表,同一个文档可以显示得更大，或者经过叠折只显示外面的一层,或者变为打印格式。可以设想一个适合用户学习特点的技术手册，它为初学者和更高一级的用户提供不同的样式,而所有的样式都是根据同样的文本产生的。

正如 XML 介于 HTML 和 SGML 之间一样，XSL 标准介于 CSS 和 SGML 的 DSSSL(Document Style Semanticsand Specification Language,文档样式语义和规范语言)之间。DSSSL 定义格式化对象的全特征模式。由于 DSSSL 使用框架语法,而且是很复杂的,所以 DSSSL 未能得到推广应用。XSL 支持 DSSSL 流对象和 CSS 对象,并对复杂的任务提供进入脚本语言的通道,而且允许扩展。实现从 CSS 到 XSL 的映射是可能的,因而内容开发商无需学习这种语言的全部。

作为一种技术预展,微软最近发布了两种 XSL 处理器:一个是可以从 XML 文档和 XSL 样式层产生 HTML 输出的命令行应用程序,另一个是一种 ActiveX 控件，用于在浏览器中显示 XML。微软的这种 XSL 处理器适合在 Windows95 和 WindowsNT 环境下通过 InternetExplorer4.0 浏览器使用。

IBM 公司及其 Lotus 子公司日前发布了 XSL 的原型，这是一个能将 XML 格式转换成 HTML 或其它 Web 格式的转换引擎，现在已可在 WWW.alphaworks.ibm.com 免费下载。这个转换引擎称为 LotusXSL，基于 WWW 联合会最新的 XSL 工作草案完成的。除了能将 XML 文档转换成 HTML 外，XSL 还能将 XML 转换为 PGML（Precision Graphics Markup Language 精确图形描述语言）。如果电子商务中用 XML 表示产品数据，用户可以使用 XSL 定义网站中数据的格式以及信息图形显示方式等。LotusXSL 打包成一个 JavaBean。用户可用 LotusXSL 创建样单，定义转换方式，就可将文档转换为相应的格式，供浏览器显示。

XSL 基础教程第一章

XSL 介绍

XML 的样式表语言 XSL 比 CCS 要复杂得多。

CSS：HTML 的样式表语言

由于 HTML 使用预先确定的标记，因此这些标记的含义都很好理解：<p>元素定义一段，<h1>元素定义一个标题。浏览器知道如何显示这些元素。

使用 CSS 向 HTML 元素增加显示格式是一个简单的过程：很容易告诉浏览器用某种特殊字体或颜色来显示各个元素，浏览器也很容易理解。

XSL：XML 的样式表

由于 XML 不使用预先确定的标记（我们可以根据需要使用任意标记），因此标记的含义并不能被直接理解：<table>可以表示一个 HTML 表格，也可以表示一件家具。由于 XML 的特性，浏览器不知道如何显示一个 XML 文档。

为了显示 XML 文档，必须要有一个机制来描述如何显示文档。这些机制之一是 CSS，但是 XSL（可扩展的样式表语言）是 XML 的首选样式表语言，它要比 HTML 使用的 CSS 复杂得多。

XSL：不仅仅是一个样式表

XSL 包含 3 部分：

一个转换 XML 文档的方法；

一个定义 XML 部分和模式的方法；

一个格式化 XML 文档的方法。

如果对此还不能理解，那么可以先将 XSL 理解成：一种将 XML 转换成 HTML 的语言，一种可以过滤和分类 XML 数据的语言，一种可以对一个 XML 文档的部分进行寻址的语言，一种可以基于数据值格式化 XML 数据的语言（如用红色显示负数），一种向不同设备输出 XML 数据的语言（如屏幕、纸或声音）。

XSL 是一种 WWW 标准

XSL 是 WWW 协会推荐的一种标准。这种语言的前两部分在 1999 年 11 月已经成为 W3C 推荐标准。2000 年，包括 XSL 格式化部分的完整 XSL 推荐标准成为 W3C 的候选标准。

XSL 语言

XSL 实际上包含三种语言，其中最重要的是 XSLT。

XSL 是三种语言的结合体

上面提到，XSL 实际上包含三种语言，具体是：

XSLT 是一种转换 XML 的语言；

XPath 是一种定义 XML 部分或模式的语言；

XSL 格式化对象是一种定义 XML 显示方式的语言。

XSLT 是一种用来将 XML 文档转换成其他类型文档或其它 XML 文档的语言。XPath 是一种对 XML 文档的部分进行寻址的语言。设计 XPath 是要让 XSLT 使用的。格式化是将一个 XSL 转换的结果变成适于读者或听众使用的输出格式的过程。

1999 年 11 月 16 日，XSLT 和 XPath 被作为两个单独的 W3C 推荐标准发布。目前对于 XSL 格式化对象还没有单独的 W3C 文档，但是在 XSL1.0 推荐标准内有一个描述。

XSLT：XSL 转换

XSLT 是 XSL 标准中最重要的部分，它用于将一个 XML 文档转换成另一个 XML 文档或另一种类型的文档，也就是将一个 XML 文档转换成浏览器所能识别的一种格式。这其中之一就是 HTML。通常，XSLT 将每个 XML 元素都转换成一个 HTML 元素。

XSLT 还可以向输出文件中增加全新的元素，或去掉一些元素。它可以重新安排这些元素并对元素进行分类，测试并确定显示哪些元素等等。

描述这种转换过程的一个常用说法是：XSL 用 XSLT 将一个 XML 来源树转换成另一个 XML 结果树（或将一个 XML 源文档转换成另一个 XML 结果文档）。

XSL 如何工作

在转换的过程中，XSLT 用 XPath 来定义源文档中与一个或多个预先确定的模板相匹配的部分。当找到了一个匹配时，XSLT 就将源文档中的匹配部分转换成结果文档；而源文档中不与任何一个模板匹配的部分最终在结果中保持不变。

本文集中介绍 XSLT 和 XPath

本文的大部分章节都集中在 XSLT 和 XPath 上。我们将用 XSLT 来定义 XML 转换，用 XPath 来为转换定义匹配模式。即使 XSL 包含了 3 个不同名称的不同部分，我们仍将使用 XSL 的通用术语。

XSL 浏览器

目前支持 XSL 的浏览器很少，我们将用 Internet Explorer 5.0 来演示 XSL。

Internet Explorer 的 XML 解析器

为了用 XSL 来处理一个 XML 文档，你需要一个带有 XSL 引擎的 XML 解析器。目前，Internet Explorer 5.0 是符合这一条件的唯一浏览器。所以，本文举例中的代码只在 Internet Explorer 5.0 或更高版本中工作。

Internet Explorer 的 XSL 引擎

Internet Explorer 5.0 中的 XSL 并不是 100%的与最新发布的 W3C XSL 标准相吻合。在 XSL 标准完全固定下来之前，Internet Explorer 5 就已经发布了，并且当时的 XSL 标准还是一个工作草稿。但是 Microsoft 已经承诺在下一个版本中解决这个问题。

本文中的例子与正式的 W3C XSL 推荐中的例子只有很小的不同，这些例子很适用于 XSL 的学习。

例子中唯一可见的区别就是 XSL 样式表声明：

这是 W3C XSL 推荐中的标准方式：<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

这是 Internet Explorer 的方式（来自 XSL 工作草稿）：<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">

Internet Explorer MSXML

MSXML 2.0 是 IE 5.0 所携带的 XML 解析器的名字，MSXML 2.5 是 Windows 2000 所携带的解析器的名字，MSXML 3.0 是 XML 解析器的最新版本。MSXML3.0 可以从 Microsoft 下载，未来版本的 Internet Explorer 和 Windows 都将携带它。

按照 Microsoft 的说法，MSXML 3.0 与正式的 W3C XSL 标准 100%兼容："MSXML 3.0 比 MSXML 2.5 有明显的进步：安全的服务器 HTTP 访问，XSLT 和 XPath 的完整执行，到 SAX 的改变（用于 XML 的简单 API），与 W3C 标准更加一致，以及许多臭虫的修复。

XSL 基础教程第二章

XSL–转换

本节将举例学习如何用 XSL 将 XML 转换成 HTML。这个举例的细节将在下一节中解释。

从 XML 文档开始

首先从打算转换成 HTML 的 XML 文档开始：

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

<CATALOG>

 <CD>

    <TITLE>Empire Burlesque</TITLE>

    <ARTIST>Bob Dylan</ARTIST>

    <COUNTRY>USA</COUNTRY>

    <COMPANY>Columbia</COMPANY>

    <PRICE>10.90</PRICE>

    <YEAR>1985</YEAR>

 </CD>
```

如果使用的是 Internet Explorer 5.0 或更高版本，就可以查看这个 XML 文件的显示结果。

创建一个 XSL 样式表文档

现在用转换模板来创建一个 XSL 样式表：

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

<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">

<xsl:template match="/">
```

```
<html>
<body>
<table border="2" bgcolor="yellow">
<tr>
<th>Title</th>
<th>Artist</th>
</tr>
<xsl:for-each select="CATALOG/CD">
<tr>
<td><xsl:value-of select="TITLE"/></td>
<td><xsl:value-of select="ARTIST"/></td>
</tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

如果使用的是 Internet Explorer 5.0 或更高版本，就可以查看这个 XSL 文件的显示结果。

将样式表连接到 XML 文档

现在向 XML 文档中增加一个 XSL 样式表引用：

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="cd_catalog.xsl"?>
<CATALOG>
<CD>
<TITLE>Empire Burlesque</TITLE>
<ARTIST>Bob Dylan</ARTIST>
<COUNTRY>USA</COUNTRY>
<COMPANY>Columbia</COMPANY>
```

```
    <PRICE>10.90</PRICE>
    <YEAR>1985</YEAR>
  </CD>
  .
  .
  .
```

如果有一个与 XSL 兼容的浏览器，例如 Internet Explorer 5.0 或更高版本，那么就能很好地将 XML 转换成 HTML。点击这里查看结果。

XSL 模板

XSL 用模板来描述如何输出 XML。

CSS 的使用规则

如果已经学习过 CSS 的知识，我们就会知道 CSS 是用一个或多个规则来定义 HTML 元素的输出，用一个选择器将规则与一个 HTML 元素联系起来。比如以下这个 CSS 规则中的 p 选择器说明应该用一种叫做 arial 的字体来显示一个<p>元素：

```
p { font-family: arial }
```

XSL 使用模板

XSL 使用一个或多个模板来定义如何输出 XML 元素，用一个匹配属性来将模板与一个 XML 元素联系起来，还可以用匹配属性来为 XML 文档的一个完整分支来定义模板。

请看以下的 XSL 样式表，它包含一个模板以输出前一节中的 XML CD 目录：

```
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
  <html>
  <body>
  <table border="1">
      <tr>
      <th>Title</th>
      <th>Artist</th>
      </tr>
      <tr>
      <td>.</td>
      <td>.</td>
```

```
    </tr>
  </table>
  </body>
  </html>
 </xsl:template>
 </xsl:stylesheet>
```

由于样式表本身就是一个 XML 文档，因此文档以一个 xml 声明开始：<?xml version='1.0'?>。第二行中的 xsl:stylesheet 标记定义了样式表的开始。第三行中的 xsl:template 标记定义了一个模板的开始。模板属性 match="/"将模板与 XML 源文档的根 (/)联系（匹配）起来。文档的其它部分包含了模板本身，最后两行定义了模板的结束和样式表的结束。

用 Internet Explorer 5 来看看 XML 文件、XSL 文件以及结果。

<xsl:value-of>元素

前面例子的结果有点令人失望，因为没有将数据从 XML 文档复制到输出中。XSL 的 <xsl:value-of>元素可以用来选择进入 XSL 转换输出流中的 XML 元素：

```
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
  <html>
  <body>
  <table border="1">
      <tr>
      <th>Title</th>
      <th>Artist</th>
      </tr>
      <tr>
      <td><xsl:value-of select="CATALOG/CD/TITLE"/></td>
      <td><xsl:value-of select="CATALOG/CD/ARTIST"/></td>
      </tr>
  </table>
  </body>
  </html>
```

```
</xsl:template>

</xsl:stylesheet>
```

注意：选择属性值用到的语法被称为 XSL 模式。它工作起来就象是在一个文件系统中航行，其中用一个前斜线 (/) 来选择子目录。

用 Internet Explorer 5 来看看 XML 文件、XSL 文件以及结果。

<xsl:for-each>元素

前面例子中的结果还是有点不太令人满意，因为从 XML 文档中只复制了一行数据到输出。XSL 的<xsl:for-each>元素可以用来将每个 XML 元素选择到 XSL 转换的输出流中：

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

<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">

<xsl:template match="/">

  <html>

  <body>

      <table border="1">

       <tr>

           <th>Title</th>

           <th>Artist</th>

       </tr>

       <xsl:for-each select="CATALOG/CD">

       <tr>

           <td><xsl:value-of select="TITLE"/></td>

           <td><xsl:value-of select="ARTIST"/></td>

       </tr>

       </xsl:for-each>

      </table>

  </body>

  </html>

</xsl:template>

</xsl:stylesheet>
```

xsl:for-each 元素在 XML 文档中查找元素，然后为每个元素重复模板的一部分。

用 Internet Explorer 5 来看看 XML 文件、XSL 文件以及结果。

XSL 基础教程第三章

客户端 XSL

如果浏览器支持 XML,，就可以用 XSL 在浏览器中将文档转换成 HTML。

一个 JavaScript 的解决方法

在前文中，我们解释了如何用 XSL 将一个文档从 XML 转换成 HTML。窍门就是向 XML 文件中增加一个 XSL 样式表，然后让浏览器来进行转换。即使这种方法能奏效，在 XML 文件中包含一个样式表引用也并非令人满意的方法，并且在不支持 XSL 的浏览器上这种方法还不能奏效。

一个更通用的方法应该是用一个 JavaScript 来进行从 XML 到 HTML 的转换。使用一个 JavaScript，就更有以下可能性：

允许 JavaScript 进行浏览器细节测试；

根据浏览器和用户需求使用不同的样式表。

这就是 XSL 的美妙之处。XSL 设计目的之一就是使数据从一个格式转换成另一个格式成为可能，从而支持不同的浏览器和不同的用户需求。

客户端 XSL 转换将成为未来浏览器工作任务的一个主要部分，我们还将看到专业化浏览器市场的成长，比如 Braille、 发声网络、网络打印机、手持 PC、移动电话等。

XML 文件和 XSL 文件

现在重新来看看前面章节中的 XML 文档：

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

<CATALOG>

  <CD>

    <TITLE>Empire Burlesque</TITLE>

    <ARTIST>Bob Dylan</ARTIST>

    <COUNTRY>USA</COUNTRY>

    <COMPANY>Columbia</COMPANY>

    <PRICE>10.90</PRICE>

    <YEAR>1985</YEAR>

  </CD>
```

.

.

.

还有附带的 XSL 样式表：

```
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
  <html>
  <body>
      <table border="2" bgcolor="yellow">
       <tr>
           <th>Title</th>
           <th>Artist</th>
       </tr>
       <xsl:for-each select="CATALOG/CD">
       <tr>
           <td><xsl:value-of select="TITLE"/></td>
           <td><xsl:value-of select="ARTIST"/></td>
       </tr>
       </xsl:for-each>
      </table>
  </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

要确保 XML 文件没有对 XSL 文件的引用，XSL 文件也没有对 XML 文件的引用。注意：上面的句子说明一个 XML 文件可以用许多不同的 XSL 文件进行转换。

在浏览器中将 XML 转换到 HTML

以下是在客户机上将 XML 文件转换成 HTML 所需要的源代码，很简单：

```
<html>
```

```
<body>

<script language="javascript">

// Load XML

var xml = new ActiveXObject("Microsoft.XMLDOM")

xml.async = false

xml.load("cd_catalog.xml")

// Load the XSL

var xsl = new ActiveXObject("Microsoft.XMLDOM")

xsl.async = false

xsl.load("cd_catalog.xsl")

// Transform

document.write(xml.transformNode(xsl))

</script>

</body>

</html>
```

如果使用的是 Internet Explorer 5.0 或更高版本，请点击这里查看结果。

代码的第一块创建了 Microsoft XML 解析器(XMLDOM)的一个例示，并将 XML 文档加载到内存中。代码的第二块创建解析器的另一个例示，并将 XSL 文档加载到内存中。代码的最后一行用 XSL 文档转换 XML 文档，将结果写入 HTML 文档中。

服务器端 XSL

由于不是所有的浏览器都支持 XML 和 XSL，因此就有了一个在服务器上将 XML 转换成 HTML 的方法。

一个跨浏览器的解决方法

在前面的章节中，我们解释了如何用 XSL 在浏览器中将 XML 文档转换成 HTML，窍门就是让 JavaScript 使用一个 XML 解析器来进行转换。但是当浏览器不支持 XML 解析器时，这种方法是不奏效的。要使 XML 数据对所有浏览器都可用，我们就必须在服务器上转换 XML 文档，并将它作为纯 HTML 发送到浏览器。

这是 XSL 的另一个美妙之处。XSL 的设计目的之一是使得在服务器上将数据从一种格式转换成另一种格式成为可能，并将可读数据返回到所有未来的浏览器中。

在服务器上进行 XSL 转换正在成为未来 Internet 信息服务器工作任务的一个主要部分，同时我们将看到专用浏览器市场的发展，如：Braille、有声网络、网络打印机、手持 PC、移动电话等。

XML 文件和 XSL 文件

现在来重新看看前面章节中的 XML 文档：

```
<?xml version="1.0"?>
<CATALOG>
 <CD>
    <TITLE>Empire Burlesque</TITLE>
    <ARTIST>Bob Dylan</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>Columbia</COMPANY>
    <PRICE>10.90</PRICE>
    <YEAR>1985</YEAR>
 </CD>
.
.
.
```

如果使用的是 Internet Explorer 5.0 或更高版本，可以点击这里查看 XML 文件。

再看看伴随的 XSL 样式表：

```
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
 <html>
 <body>
    <table border="2" bgcolor="yellow">
     <tr>
        <th>Title</th>
        <th>Artist</th>
     </tr>
     <xsl:for-each select="CATALOG/CD">
     <tr>
        <td><xsl:value-of select="TITLE"/></td>
        <td><xsl:value-of select="ARTIST"/></td>
```

```
    </tr>
    </xsl:for-each>
   </table>
  </body>
  </html>
 </xsl:template>
 </xsl:stylesheet>
```

如果使用的是 Internet Explorer 5.0 或更高版本，可以点击这里查看 XSL 文件。

以上 XSL 文档的语法在前面章节中已经解释过了，因此这里不再做解释。但是要确保 XML 文件没有对 XSL 文件的引用，XSL 文件也没有对 XML 文件的引用。同时请注意：上面的句子表明一个服务器上的 XML 文件可以用许多不同的 XSL 文件进行转换。

在服务器端将 XML 转换成 HTML

以下是在服务器上转换 XML 文件所需要的简单源代码：

```
<%
'Load the XML
set xml = Server.CreateObject("Microsoft.XMLDOM")
xml.async = false
xml.load(Server.MapPath("cd_catalog.xml"))
'Load the XSL
set xsl = Server.CreateObject("Microsoft.XMLDOM")
xsl.async = false
xsl.load(Server.MapPath("cd_catalog.xsl"))
'Transform the file
Response.Write(xml.transformNode(xsl))
%>
```

代码的第一块创建 Microsoft XML 解析器(XMLDOM)的一个例示，并将 XML 文件装载到内存中。代码的第二块创建解析器的另一个例示，并将 XSL 文档装载到内存。代码的最后一行用 XSL 文档转换 XML 文档，并将结果返回浏览器。

XSL 基础教程第四章

XSL 索引

XSL 可以用来对一个 XML 文档进行索引。

将索引信息放在哪里

现在重新看看在以前许多章节中都曾看到过的 XML 文档：

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

<CATALOG>

 <CD>

    <TITLE>Empire Burlesque</TITLE>

    <ARTIST>Bob Dylan</ARTIST>

    <COUNTRY>USA</COUNTRY>

    <COMPANY>Columbia</COMPANY>

    <PRICE>10.90</PRICE>

    <YEAR>1985</YEAR>

 </CD>

.

.

.
```

要想将这个 XML 文件作为一个普通的 HTML 文件输出，并且同时对它进行索引，只需要在 XSL 文件中增加一个 order-by 属性，如下：

```
<xsl:for-each select="CATALOG/CD" order-by="+ ARTIST">
```

order-by 属性使用加号(+)或减号(-)来定义是使用升序还是降序，再用一个元素名称来定义排序的元素。

现在来看看经过轻微调整的 XSL 样式表（或在 IE5 中打开它）：

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

<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">

<xsl:template match="/">

 <html>

 <body>

    <table border="2" bgcolor="yellow">

     <tr>

        <th>Title</th>
```

```
            <th>Artist</th>
          </tr>
          <xsl:for-each select="CATALOG/CD"
          order-by="+ ARTIST">
          <tr>
            <td><xsl:value-of select="TITLE"/></td>
            <td><xsl:value-of select="ARTIST"/></td>
          </tr>
          </xsl:for-each>
        </table>
      </body>
      </html>
    </xsl:template>
    </xsl:stylesheet>
```

在浏览器中转换

以下是在浏览器中将 XML 文件转换成 HTML 所需要的简单代码：

```
<html>
<body>
<script language="javascript">
// Load XML
var xml = new ActiveXObject("Microsoft.XMLDOM")
xml.async = false
xml.load("cd_catalog.xml")
// Load the XSL
var xsl = new ActiveXObject("Microsoft.XMLDOM")
xsl.async = false
xsl.load("cd_catalog_sort.xsl")
// Transform
document.write(xml.transformNode(xsl))
```

```
</script>

</body>

</html>
```

如果使用的是 Internet Explorer 5.0 或更高版本，请点击这里查看结果。

XSL 过滤器查询

XSL 可以用来过滤一个 XML 文件。

在哪里放置过滤器信息

现在重新看看你以前已经看过多次的 XML 文档：

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

<CATALOG>

 <CD>

  <TITLE>Empire Burlesque</TITLE>

  <ARTIST>Bob Dylan</ARTIST>

  <COUNTRY>USA</COUNTRY>

  <COMPANY>Columbia</COMPANY>

  <PRICE>10.90</PRICE>

  <YEAR>1985</YEAR>

 </CD>

.

.

.
```

要过滤 XML 文件，只需要为 XSL 文件中的 for-each 元素的选择属性增加一个过滤器，如下：

```
<xsl:for-each select="CATALOG/CD[ARTIST='Bob Dylan']">
```

合法的过滤器操作符是：

= 等于

!= 不等于

< 小于

> 大于

现在看看经过轻微调整的 XSL 样式表：

```
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
 <html>
 <body>
 <table border="2" bgcolor="yellow">
 <tr>
      <th>Title</th>
      <th>Artist</th>
 </tr>
 <xsl:for-each select="CATALOG/CD[ARTIST='Bob Dylan']">
 <tr>
      <td><xsl:value-of select="TITLE"/></td>
      <td><xsl:value-of select="ARTIST"/></td>
 </tr>
 </xsl:for-each>
 </table>
 </body>
 </html>
</xsl:template>
</xsl:stylesheet>
```

在浏览器中转换

以下是在浏览器中将 XML 文件转换成 HTML 所需要的简单代码：

```
<html>
<body>
<script language="javascript">
// Load XML
var xml = new ActiveXObject("Microsoft.XMLDOM")
xml.async = false
```

```
xml.load("cd_catalog.xml")

// Load the XSL

var xsl = new ActiveXObject("Microsoft.XMLDOM")

xsl.async = false

xsl.load("cd_catalog_filter.xsl")

// Transform

document.write(xml.transformNode(xsl))

</script>

</body>

</html>
```

如果使用的是 Internet Explorer 5.0 或更高版本，请点击这里查看结果。

XSL 基础教程第五章

XSL 条件 IF

XSL 可以用一个 IF 语句过滤来自 XML 文档的信息。

在哪里放置 IF 条件

现在来重新看看你已经看过多次的 XML 文档：

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

<CATALOG>

  <CD>

    <TITLE>Empire Burlesque</TITLE>

    <ARTIST>Bob Dylan</ARTIST>

    <COUNTRY>USA</COUNTRY>

    <COMPANY>Columbia</COMPANY>

    <PRICE>10.90</PRICE>

    <YEAR>1985</YEAR>

  </CD>

.

.
```

.

要想放置一个对文件内容的条件测试 if 命令，只需要向 XSL 文档中增加一个 xsl:if 元素，如下：

```
<xsl:if match=".[ARTIST='Bob Dylan']">
... 一些输出...
</xsl:if>
```

现在看一下经过轻微调整的 XSL 样式表：

```
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
 <html>
 <body>
     <table border="2" bgcolor="yellow">
      <tr>
          <th>Title</th>
          <th>Artist</th>
      </tr>
      <xsl:for-each select="CATALOG/CD">
      <xsl:if match=".[ARTIST='Bob Dylan']">
      <tr>
           <td><xsl:value-of select="TITLE"/></td>
           <td><xsl:value-of select="ARTIST"/></td>
 </tr>
          </xsl:if>
      </xsl:for-each>
     </table>
 </body>
 </html>
</xsl:template>
```

```
</xsl:stylesheet>
```

在浏览器中转换

以下是在浏览器中将 XML 文件转换成 HTML 所需要的简单代码：

```
<html>

<body>

<script language="javascript">

// Load XML

var xml = new ActiveXObject("Microsoft.XMLDOM")

xml.async = false

xml.load("cd_catalog.xml")

// Load the XSL

var xsl = new ActiveXObject("Microsoft.XMLDOM")

xsl.async = false

xsl.load("cd_catalog_if.xsl")

// Transform

document.write(xml.transformNode(xsl))

</script>

</body>

</html>
```

如果使用的是 Internet Explorer 5.0 或更高版本，请点击这里查看结果。

XSL 条件选择 Choose

XSL 可以使用条件选择过滤 XML 文档。

在哪里放置选择条件

重新看看几乎在前面每个章节都看到过的 XML 文档：

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

<CATALOG>

 <CD>

  <TITLE>Empire Burlesque</TITLE>

  <ARTIST>Bob Dylan</ARTIST>

  <COUNTRY>USA</COUNTRY>
```

```
    <COMPANY>Columbia</COMPANY>
    <PRICE>10.90</PRICE>
    <YEAR>1985</YEAR>
  </CD>
.
.
.
```

要想插入一个对文件内容的条件选择测试，只需要向 XSL 文档中增加 xsl:choose、xsl:when 以及 xsl:otherwise 元素，如下：

```
<xsl:choose>
  <xsl:when match=".[ARTIST='Bob Dylan']">
    ... 一些代码 ...
  </xsl:when>
  <xsl:otherwise>
    ... 一些代码 ...
  </xsl:otherwise>
</xsl:choose>
```

现在来看看经过轻微调整的 XSL 样式表：

```
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
  <html>
  <body>
    <table border="2" bgcolor="yellow">
      <tr>
        <th>Title</th>
        <th>Artist</th>
      </tr>
      <xsl:for-each select="CATALOG/CD">
```

```
<tr>
<td><xsl:value-of select="TITLE"/></td>
<xsl:choose>
<xsl:when match=".[ARTIST='Bob Dylan']">
<td bgcolor="#ff0000">
<xsl:value-of select="ARTIST"/>
</td>
</xsl:when>
<xsl:otherwise>
<td><xsl:value-of select="ARTIST"/></td>
</xsl:otherwise>
</xsl:choose>
</tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

在浏览器中转换

以下是在浏览器中将 XML 文件转换成 HTML 所需要的简单代码：

```
<html>
<body>
<script language="javascript">
// Load XML
var xml = new ActiveXObject("Microsoft.XMLDOM")
xml.async = false
xml.load("cd_catalog.xml")
// Load the XSL
```

```
var xsl = new ActiveXObject("Microsoft.XMLDOM")

xsl.async = false

xsl.load("cd_catalog_choose.xsl")

// Transform

document.write(xml.transformNode(xsl))

</script>

</body>

</html>
```

如果使用的是 Internet Explorer 5.0 或更高版本，请点击这里查看结果。

XSLT 是什么类型的语言？

分析和概述

内容：

XSLT 是什么类型的语言，其用途是什么，为什么要这样设计它？这些问题可以有许多不同的答案，初学者往往会感到困惑，因为这种语言与他们以前习惯使用的语言之间有很大差别。本文尝试说明 XSLT。本文并不试图教您编写 XSLT 样式表，它将说明这种语言的起源，它擅长什么，以及您为什么应该使用它。

我撰写本文的初衷是为一篇关于 Saxon 的技术文章提供必要的背景知识，打算提供在传统 XSLT 处理器中使用的实现技巧内幕，从而帮助用户使其样式表的性能达到最大化。但 developerWorks 的编辑们劝说我：这篇介绍应该吸引更广泛的读者，值得作为 XSLT 语言的独立说明而单独发表。

什么是 XSLT?

XSLT 语言由万维网联盟 (W3C) 定义，并且该语言的 1.0 版本在 1999 年 11 月 16 日作为“推荐书”发布（请参阅参考资料）。我已经在拙作 XSLT Programmers' Reference 中提供

了全面的规范和用户指南，因此我不打算在本文中涵盖相同内容。确切地讲，本文的目的只是使读者理解 XSLT 适合大规模事物的哪些位置。

XSLT 的角色

XSLT 的最初目的是将信息内容与 Web 显示分离。如其最初定义那样，HTML 通过按抽象概念（如段落、重点和编号列表）定义显示来实现设备独立性。随着 Web 变得越来越商业化，出版人希望其输出质量能达到与印刷品相同的质量。这逐渐导致越来越多地使用具体显示控件，如页面上材料的明确字体和绝对位置。然而不幸的是完全可以预料其副作用，即将相同的内容传递到替代设备，如数字电视机和 WAP 电话（印刷业的行话再现效果）将会变得日益困难。

由于吸收了印刷业使用 SGML 的经验，在 1998 年初定义了一种标记语言 XML，它用于表示独立于显示的结构化内容。与 HTML 使用一组固定概念（如段落、列表和表）不同，XML 标记中使用的标记完全是用户定义的，其用意是这些标记应该与所关注的对象（如人、地点、价格和日期）相关。尽管 HTML 中的元素本质上都是印刷样式（虽然处于抽象级别），而 XML 的目标是元素应该描述实际对象。例如，清单 1 显示了表示足球锦标赛结果的 XML 文档。

清单 1. 表示足球锦标赛结果的 XML 文档

```
<results group="A">
<match>
  <date>10-Jun-1998</date>
  <team score="2">Brazil</team>
  <team score="1">Scotland</team>
</match>
<match>
  <date>10-Jun-1998</date>
  <team score="2">Morocco</team>
  <team score="2">Norway</team>
</match>
<match>
  <date>16-Jun-1998</date>
  <team score="1">Scotland</team>
  <team score="1">Norway</team>
</match>
<match>
  <date>16-Jun-1998</date>
  <team score="3">Brazil</team>
  <team score="0">Morocco</team>
</match>
<match>
  <date>23-Jun-1998</date>
  <team score="1">Brazil</team>
  <team score="2">Norway</team>
</match>
<match>
  <date>23-Jun-1998</date>
  <team score="0">Scotland</team>
  <team score="3">Morocco</team>
</match>
</results>
```

如果要通过 Web 浏览器显示这些足球赛的结果，不要指望系统会产生合理的布局。需要其它一些机制来告诉系统如何在浏览器屏幕、电视机、WAP 电话或真正在纸张上显示数据。这就是使用样式表的目的。样式表是一组说明性的规则，它定义了应如何表示源文档中标记标识的信息元素。

W3C 已经定义了两个系列的样式表标准。第一个是在 HTML 中广泛使用的 CSS（级联样式表），当然它也可以在 XML 中使用。例如，可以使用 CSS 来表示何时显示发票，应支付的总额应该用 16 点 Helvetica 粗体字显示。但是，CSS 不能执行计算、重新整理或排序数据、组合多个源码中的数据或根据用户或会话的特征个性化显示的内容。在这个足球赛结果的例子中，CSS 语言（即使是最新版本 CSS2，尚未在产品中完全实现）的功能还不够强大，不能处理这项任务。由于这些原因，W3C 已着手开发更强大的样式表语言 XSL（可扩展样式表语言），并采纳了 SGML 社区中开发的 DSSSL（文档样式、语义和规范语言）中许多好的构思。

在 XSL 的开发过程中（这在 DSSSL 中已有所预示），发现在准备 XML 文档以备显示的过程中执行的任务可以分成两个阶段：转换和格式化。转换是将一个 XML 文档（或其内存中的表示法）转换成另一个 XML 文档的过程。格式是将已转换的树状结构转换成两维图形表示法或可能是一维音频流的过程。XSLT 是为控制第一阶段“转换”而开发的语言。第二阶段“格式化”的开发工作还是进行中。但实际上，大多数人现在使用 XSL 将 XML 文档转换成 HTML，并使用 HTML 浏览器作为格式化引擎。这是可行的，因为 HTML 实际上只是 XML 词汇表的一个示例，而 XSLT 可以使用任何 XML 词汇表作为其目标。

将转换成一种语言和格式化成另一种语言这两个操作分离经证实的确是一种好的决策，因为转换语言的许多应用程序经证明无法向用户显示文档。随着 XML 日益广泛地用作电子商务中的数据互换语法，对于应用程序将数据从一个 XML 词汇表转换成另一个 XML 词汇表的需求也在不断增加。例如，某个应用程序可能从电视收视指南中抽取电视节目的细节，并将它们插入按次付费客户的月帐单中。同样，还有许多实用的数据转换，在这些转换中源词汇表和目标词汇表是相同的。它们包括数据过滤，以及商务操作，如施行涨价。因此，随着在系统中开始越来越多地以 XML 语法的形式使用数据，XSLT 就逐渐成为由于处理这些数据的随处可见的高级语言。

在拙作中，我做了这样一个比喻：XSLT 与 XML 的关系，就好象 SQL 与表格化数据的关系一样。关系模型的强大功能并非来自用表存储数据的思想，而是源于 SQL 中可行的基于关系运算的高级数据操作。同样，XML 的层次化数据模型对应用程序开发者的帮助实际上也非常小。正是因为 XSLT 作为 XML 数据的高级操作语言提供了如此强大的功能。

XSLT 作为语言

就某些方面而言，XSLT 作为一种语言来说是非常古怪的。我不打算在本文中讨论已做出的设计决策的基本原理，尽管可以通过它们在逻辑上追溯到语言设计者确定的对 XSLT 的要求。如需更完整的说明，请参阅拙作的第 1 章。

以下概述了 XSLT 语言的部分主要特性。

XSLT 样式表是一个 XML 文档。通过使用 XML 的尖括号标记语法来表示文档的结构。这种语法在某种程度上是比较笨拙的，而此决策可以使该语言变得更罗嗦。但是，它确实有好处。它表示可以自动使用 XML 的所有词汇设备（例如，Unicode 字符编码和转义，使用外部实体等等）。它表示很容易使 XSLT 样式表变成转换的输入或输出，使该语言可以作用于自身。它还使将期望的 XML 输出块嵌入样式表变得很容易。实际上，许多简单的样式表基本上可以写作期望输出文档的模板，并且可以将一些特殊指令嵌入文本中，以便插入输入中的变量数据或计算某个值。这就使 XSLT 在这个简单的级别上非常类似于许多现有的专用 HTML 模板语言。

基本处理范例是模式匹配。在这方面，XSLT 继承了文本处理语言（如 Perl）的传统，这种传统可以一直追溯到 1960 年代的语言，如 SNOBOL。XSLT 样式表包括一组模板规则，每条规则都使用以下方式：“如果在输入中遇到此条件，则生成下列输出。”规则的顺序是无关紧要的，当有几条规则匹配同一个输入时，将应用冲突解决算法。然而，XSLT 与串行文本处理语言的不同之处是 XSLT 对输入并非逐行进行处理。实际上，XSLT 将输入 XML 文档视为树状结构，每条模板规则都适用于树中的一个节点。模板规则本身可以决定下一步处理哪些节点，因此不必按输入文档的原始顺序来扫描输入。

XSLT 处理器的操作

XSLT 处理器使用树状结构作为其输入，并生成另一个树状结构作为输出。图 1 中显示了这一点。

图 1. XSLT 输入和输出的树状结构

常常通过对 XML 文档进行语法分析来生成输入树状结构，而输出树状结构通常被串行化到另一个 XML 文档中。但 XSLT 处理器本身操作的是树状结构，而不是 XML 字符流。这个概念最初给许多用户的感觉是不切实际的，结果却对理解如何执行更复杂的转换起了关键作用。首先，它表示 XSLT 处理器可以理解源文档中与树状结构无关的特殊之处。例如，无论属性是包括在单引号中还是在双引号中，都不可能应用不同的处理，因为会将这两种形式视为同一个基本文档的不同表示方法。更深入地看，它表示处理输入元素或生成输出元素是一个原子操作。不可能将处理元素的开始标记和结束标记分成单独的操作，因为一个元素会自动表示成树模型的单节点。

XSLT 使用叫作 XPath 的子语言来引用输入树中的节点。XPath 本质上是与具有层次结构的 XML 数据模型相匹配的查询语言。它可以通过按任何方向浏览树来选择节点，并根据节点的值和位置应用谓词。它还包括用于基本字符串处理、数字计算和布尔代数的工具。例如，XPath 表达式 ../@title 选择当前节点的父代元素的标题属性。XPath 表达式用于选择要进行处理的输入节点、在条件处理期间测试条件，以及计算值以便插入结果树中。模板规则中还使用了 XPath 表达式的简化形式“模式”来定义特定模板规则适用于哪些节点。XPath 在单独的 W3C 推荐书中定义，它允许使用在其它上下文中再使用的查询语言，特别是用于定义扩展超链接的 XPointer。

XSLT 以传统语言（如 Lisp、Haskell 和 Scheme）中的功能性编程的概念为基础。样式表由模板组成，这些模板基本上是单一功能 -- 每个模板将输出树的一部分定义成一部分输入树的功能，并且不产生副作用。使用无副作用的规则受到严格控制（除了转义成用类似 Java 的语言编写的外部代码）。XSLT 语言允许定义变量，但不允许现有变量更改它的值 -- 即没有赋值语句。这个策略使许多新用户感到困惑，其目的是为了允许逐步应用样式表。其原理是如果语言没有副作用，那么对输入文档做很小的改动时，不必从头执行整个转换就应该可以计算出对输出文档的最后更改。目前必须说这只是理论上的可能，任何现有 XSLT 处理器还不能实现。（注：虽然 XSLT 以功能性编程概念为基础，但它还不是一个完整的功能性编程语言，因为它缺少将函数当作一级数据类型进行处理的能力。）

示例样式表

在这个阶段，使用示例会使语言变得更清楚。清单 2 显示了列出足球赛结果的简单样式表。

清单 2. 足球赛结果的基本样式表

```
<xsl:transform
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

<xsl:template match="results">
```

```
  <html>
  <head><title>
  Results of Group <xsl:value-of select="@group">
  </title></head>
  <body><h1>
  Results of Group <xsl:value-of select="@group">
  </h1>
  <xsl:apply-templates>
  </body></html>
 </xsl:template>

 <xsl:template match="match">
  <h2>
  <xsl:value-of select="team[1]"> versus <xsl:value-of select="team[2]">
  </h2>
  <p>Played on <xsl:value-of select="date"></p>
  <p>Result:
  <xsl:value-of select="team[1] ">
  <xsl:value-of select="team[1]/@score">,
  <xsl:value-of select="team[2] ">
  <xsl:value-of select="team[2]/@score">
  </p>
 </xsl:template>

 </xsl:transform>
```

这个样式表包括两个模板规则，一个匹配 <results> 元素，另一个匹配 <match> 元素。<results> 元素的模板规则输出页面的标题，然后调用 <xsl:apply-templates>，这是一个 XSLT 指令，它将处理当前元素的所有子代，对于每个子代都使用其适当的模板规则。在本例中，<results> 元素的所有子代都是 <match> 元素，所以会用第二个模板规则来处理它们。规则输出了一个标识比赛的次级 HTML 标题（以 "Brazil versus Scotland" 的形式），然后生成 HTML 段落，给出了比赛的日期和两队的比分。

该转换的结果就是一个 HTML 文档，该文档在浏览器中的表示如图 2 所示。

图 2. 清单 2 中样式表的结果

这是一种非常简单的表示信息的方法。然而，XSLT 的功能比这要强大得多。清单 3 包含了另一个可以操作相同源数据的样式表。这次，样式表计算一个比赛名次表，用来显示锦标赛结束时各队的名次。

清单 3. 计算球队名次表的样式表

```
<xsl:transform
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0">


 <xsl:variable name="teams" select="//team[not(.=preceding::team)]">
 <xsl:variable name="matches" select="//match">

 <xsl:template match="results">
```

```
<html><body>
  <h1>Results of Group <xsl:value-of select="@group"></h1>

  <table cellpadding="5">
  <tr>
  <td>Team</td>
  <td>Played</td>
  <td>Won</td>
  <td>Drawn</td>
  <td>Lost</td>
  <td>For</td>
  <td>Against</td>
  </tr>
  <xsl:for-each select="$teams">
  <xsl:variable name="this" select=".">
  <xsl:variable name="played" select="count($matches[team=$this])">

  <xsl:variable name="won"
  select="count($matches[team[.=$this]/@score > team[.!=$this]/@score])">
  <xsl:variable name="lost"
  select="count($matches[team[.=$this]/@score < team[.!=$this]/@score])">
  <xsl:variable name="drawn"
  select="count($matches[team[.=$this]/@score = team[.!=$this]/@score])">
  <xsl:variable name="for"
  select="sum($matches/team[.=current()]/@score)">
  <xsl:variable name="against"
  select="sum($matches[team=current()]/team/@score) - $for">

  <tr>
  <td><xsl:value-of select="."></td>
  <td><xsl:value-of select="$played"></td>
  <td><xsl:value-of select="$won"></td>
  <td><xsl:value-of select="$drawn"></td>
  <td><xsl:value-of select="$lost"></td>
  <td><xsl:value-of select="$for"></td>
  <td><xsl:value-of select="$against"></td>
  </tr>
  </xsl:for-each>
  </table>
</body></html>
</xsl:template>

</xsl:transform>
```

这里没有足够的篇幅来完整地说明这个样式表，简而言之，它为球队声明了一个变量，变量值是一个节点集合，其中每个参赛球队都有一个实例。然后它计算每支球队的胜、平或负的比赛场次总数，以及球队进球或失球的总数。图 3 显示了它在浏览器中的最终输出结果。

图 3. 清单 3 中名次样式表的结果

这个示例的目的是说明 XSLT 不单单能够对源文档中出现的文本指定字体和布局。它是一个完整的编程语言，能够以任何方式转换源数据以供显示，或者输入另一个应用程序。

XSLT 的优点

您为什么考虑使用 XSLT?

XSLT 给了您传统高级声明编程语言的所有好处，特别是对于转换 XML 文档的任务。

高级语言带来的实际好处是开发生产力。但实际上，真正的价值源自于更改的潜力。与使用低级 DOM 和 SAX 接口编码的过程性应用程序相比，用于转换 XML 数据结构的 XSLT 应用程序更能适应对 XML 文档细节的更改。在数据库世界中，这种特性叫做数据独立性，正是由于数据独立性导致了诸如 SQL 之类声明性语言的成功，并使旧的引导性数据访问语言走向衰亡。我坚信在 XML 世界中也会这样。

当然与所有声明性语言一样，XSLT 也会降低性能。但是对于大多数应用程序，今天的 XSLT 处理器的性能已经完全能够满足应用程序的需要，并且它会变得越来越好。在我的第二篇文章中，我将讨论 XSLT 处理器中使用的一些优化技巧，如我自己的 Saxon 产品。

结束语

我想要在本文中展示的是 XSLT 是一种用于操作 XML 文档的完整高级语言，就如同 SQL 是操作关系表的高级语言一样。应该注意到 XSLT 不仅是一种样式设计语言，它比 CSS（或者甚至 CSS2）的功能更强大。

我见到过一些应用程序，它们的所有商务逻辑都用 XSLT 编码。在一个三层在线银行系统中，我看到：

从后端操作系统以 XML 消息的形式检索所有数据。
在联机会话的持续时间内，用户的帐户数据在内存中以 XML DOM 形式表示。
所有给用户的信息首先封装成 XML 消息，然后用服务器或客户机附带的 XSLT 转换根据浏览器的性能将这些消息转换成 HTML。

该应用程序的数据都是 XML 格式的，并且逻辑（包括数据访问逻辑、商务逻辑和显示逻辑）都由 XSLT 来实现。我建议每个项目都采用那种体系结构，但这还需要很长时间，我认为我们会在几年之内见到那种系统。

作为一种编程语言，XSLT 有许多特性 -- 从它使用 XML 语法到其功能性编程原理的基础 -- 还不为一般 Web 程序员所熟悉。那意味着一条陡峭的学习曲线和通常遇到许多挫折。当初对于 SQL 也是如此，所有这些表示 XSLT 与以前的编程语言有着本质的区别。但不要放弃：它是功能非常强大的技术，值得努力学习。

参考资料

同一个作者撰写的 Wrox 书籍 XSLT Programmer's Reference。XSLT 语言的综合指南。
W3C 出版的 XSLT 1.0 Recommendation。XSLT 语言的权威性规范。
W3C 出版的 XPath 1.0 Recommendation。XSLT 样式表中使用的 XPath 表达式语法的权威性规范。
XSL-List，一个有关 XSLT 所有事物的繁忙邮件列表，它附带有可搜索档案，由 MulberryTech 进行管理。
www.xslinfo.com，一个很好的网络中央页面，带有到 XSLT 资源的链接，内容包括软件、书籍、教程和其它内容。

关于作者

Michael Kay 在 XML 界非常著名，他是 Saxon XSLT 处理器和 Wrox 书籍 XSLT Programmer's Reference 的作者。多年以前，他就获得了博士学位，他的研究领域是数据库技术。自那时起，他设计了 Codasyl 数据库、关系数据库、面向对象数据库和自由文本数据库软件。

在写作时，Michael 还兼了几份工作（并没有休息）。他刚结束了在 ICL（一家英国 IT 服务公司）24 年的工作，并投奔 Software AG 成为体系结构小组的一员，该小组负责掌控未来 XML 产品的方向。

作者选择这张照片来证明他并非总是象他在 Wrox 书籍中所表现得那样严肃。

XSL 对象格式使用指南（一）

World Wide Web 联盟的规范书将可扩展样式语言(XSL)规划成两个部分：

■XSLT 用来转换 XML 文件
■XSL 对象格式(XSL FO) 用来详细说明格式语句的 XML 函数库。

XSLT 非常容易学习和使用。只需要花费很少的时间，开发者就能将 XML 文档转换成可以在用户自己浏览器上看到的 HTML 文档。这也是开发者喜欢使用 XSLT 的原因。

XSL 对象格式是一个以 XML 为基础的扩展函数库，能详细定义标记页数、版面和字体等网页内容信息。XSL FO 扩展语言非常复杂，也很冗长；但只有通过 XSL FO 才能快速把 XSLT 转换成文档源文件。

除了完全介绍 XSL FO，这篇文章还将提供足够的信息和例子来说明怎样使用 XSL FO。我们将通过 XSL FO 学习制作一个西班牙语的评论手册并将它打印出来。我们将使用 Apache Software Foundation 的 FOP 工具软件使 FO 文件转换成 PDF 文件。

初始化

把 XSL FO 文件转换成 XML 文档，先需要基本的 XML 处理指令和 FO 根元素：

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

文档的结构为:

- ●主要的版面设计，由以下组成
  - ○文档可能出现的各种页面
  - ○安排这些页面的格式序列
- ●页面和它们的内容

页面规划

在 FO 文档的开始<fo:root> 标签后，我们必须描述文档所拥有的页面类型。文档拥有下面图表中的三种页面。按照空白区域分类为封面、左边页和右边页。封面和左边页在左面有较多的空白区域。每个页面都有页眉和页脚。

让我们从定位页面长度、宽度及空白空间开始。在文章里单位都使用厘米，也可以使用任何 CSS 单位，如象素 (pixel)、毫米、英寸等。这些规格被命名为 simple-page-master 并用 master-name 语句指定。请看代码示例。需要注意的是页面空白区域不能有任何需要印刷出来的内容。

内容区

在所有的印刷品的图表中都会有零星的线条。那是页面的内容区(正式上被称作页面参照

区)， 在下面可以看出内容区被分成五个区域。

说明

我们必须先了解一些术语。当我们设定页面边的空白时，我们会使用 top, bottom, left, and right 这样的词汇。因为每个人都认为纸张的一块边缘就是纸张的左边、顶部、底部和右边。当我们谈到内容区域时会使用不同的词汇，因为不是所有的语言都能描述左到右或顶部到底部。

FO 认为一个页面由两种元素构成: block elements (例如文章的段落)它由新的一行起头。inline elements (例如字体的粗体和斜体)。FO 的 block-progress-direction 决定了段落在页面的位置。before-edge 表示在段落之前，after-edge 表示在段落之后。

inline-progress-direction 决定了在一行话中每个字符的位置。start-edge 表示位置在一行话之前， end-edge 表示位置在一行话之后。

对于希伯莱语来说，象以下所示：开头和结尾中文字位置与英语习惯是互相对应的。(阿拉伯语的写法也相似)

日语写法有时象下面一样写，下面是 XSL 规范手册中的图片：

这种新函数库最大的优势是它使用的语言不受约束。如果你希望页面的标题出现在和普通文本相对应的位置，你可以象下面一样设定 text-align="end"

一个有趣的标题

标题一般在最上方，这样才能吸引读者的注意

如果文档最后要翻译成阿拉伯语或日语，你仍然能确定标题仍然出现在文本相对应的位置。就不需要来回调整文档的左右和上下。

指定区域的尺寸

封面页不需要页眉和页脚，所以我们只需要象以下所示的用粗体指定 region-body 信息。

```
<fo:simple-page-master master-name="cover"

page-height="12cm"
page-width="12cm"
margin-top="0.5cm"
margin-bottom="0.5cm"
margin-left="1cm"
margin-right="0.5cm">
<fo:region-body
margin-top="3cm" />

</fo:simple-page-master>
```

左边页和右边页拥有页眉和页脚，所以我们需要指定 region-before 和 region-after 的范围。

要点:你得给 region-body 指定范围，相当于指定 region-before 和 region-after 的范围 (FOP 现在不支持 region-start 和 region-end)。如果你象下面这样做 ：

```
<fo:region-before extent="1cm"/>
<fo:region-after extent="1cm"/>

<fo:region-body
margin-top="0.20cm"
margin-bottom="0.20cm" />
```

结果会是：

XSL 对象格式使用指南（二）

现在页面大致已经被定位，你可以能够定位页面的顺序了。
我们建立的文档由封面页和内容页组成。页面有两种排序方法，如果页面为奇数页码就命名为：封面页 (它只有一页),其后是"内容页", 它由左边页和右边页交替排列。如果文档由单独的封面页构成就不需要再给页面排序了，我们不能因这么做而得到任何好处。(如果文档页数象一本书，那就值得去这样做了)我们会集中精力定位包括主要内容的页面顺序。在英文中，一本书内容包括偶数页码的左边页和奇数页码的右边页。这样一本打开的书就有两个页面。规范如这里所示，行号是附加的参考。

封面页
现在主要的页面和页面顺序都已经被确定，可以开始在这些页面中放置内容了。在内容的开头，我们可以使用实际的代码©；作为版权信息。请看代码示例。

创建 PDF 文件：
现在我们拥有一些内容了，我们可以将页面打印出来。如果你想自己尝试一下，你需要下载 Apache Software Foundation 的 FOP 工具软件并安装，还需要：

- Java 1.1.x 或更新的版本
- 一个支持 SAX 和 DOM 的 XML 分析软件
- 一个 XSLT 分析软件 (如果你已经下载了 Xalan，你会得到所有 Xerces 软件，XML 分析软件和 Xalan- XSLT 分析软件)
- 一个 SVG 数据库，它可以从 FOP 的 w3c.jar 数据包中得到

例如，在一个 Linux 系统上，你可以把所有的.jar 文件放进一个方便的文件夹中，创建一个名为 fop.sh 的脚本。

```
java -cp \
/usr/local/xml-jar/fop.jar:/usr/local/xml-jar/w3c.jar:\
/usr/local/xml-jar/xml.jar:/usr/local/xml-jar/xerces.jar:\
/usr/local/xml-jar/xalan.jar:/usr/local/xml-jar/bsf.jar \
org.apache.fop.apps.CommandLine $1 $2
```

调用脚本并键入 fop.sh spanish1.fo spanish1.pdf 来产生一个 PDF 文件。阅读这个文件，你需要一个 PDF 文件阅读器；Adobe 公司的 Acrobat 阅读器能够在 Linux、acintosh 和 Windows 上工作。Linux 用户也可以使用 xpdf，一个 X-Window PDF 阅读器。我们看到的文档的输出和源代码完全不同。

文档美化一下效果会更好，在 src 属性里加入象 external-graphic 这样的图象 URI。附加的语句用粗体表示。

```
<fo:block font-family="Helvetica" font-size="12pt"

text-align="end" space-after="36pt">
Copyright #169; 2001 J. David Eisenberg

</fo:block>
<fo:block text-align="end">
```

```
<fo:external-graphic src="file:images/catcode_logo.jpg"
width="99px" height="109px"/>

</fo:block>
<fo:block>

A Catcode Production

</fo:block>
```

现在，图象已经变得非常漂亮了。

开始加入内容
在离开这篇文章之前，我们开始使页面加入内容。在这个例子中，我们在 xsl-region-before 和 xsl-region-after 或 xsl-region-body 之间加入内容。

结果：

在后面两篇里，我们将向你显示使用 XSLT 创造 FO 元素是多么简单。你将学到怎么在你的文档中放进清单和表格。

XSL 对象格式使用指南（三）

在前两篇中我们已经讨论了怎样处理封面页和内容页，现在我们准备把以下内容放入这西班牙评论手册里。

标题和段落属性由<fo:block> 元素控制，粗体和带下划线的词语由<fo:inline>元素控制。现在开始描述第一个标题：

```
<fo:block

font-size="14pt" font-family="sans-serif"
font-weight="bold" color="green"
space-before="6pt" space-after="6pt">
Introduction

<fo:block>
```

space-before 属性和 space-after 属性是储存区域中属性设置值中的两种。这些属性和层叠样式表 CSS 相类似

字体属性
font-family，font-weight，font-style (斜体)，font-size，font-stretch，font-variant
背景属性
background-color，background-image，background-repeat，background-attachment (卷动或固定)
边界属性

border-location-info：
location 可以为 before，after，start,，end,，top，bottom，left 或 right
info is 可以为字体，宽度或颜色
填充属性
padding-location：`
location is 可以为 before，after,，start，end，top，bottom，left 或 right
空白属性
margin-location：
location 可以为 top，bottom，left 或 right
文本排列属性
text-align 和 text-align-last (用作文本最后一行); 函数可以为 start，end，left，right 或 center
缩排属性
text-indent (第一行)，start-indent，end-indent
其他属性
wrap-option (没有约束或有约束)；widows 和 orphans (决定页面的顶部或左部有多少行)break-after 和 break-before；reference-orientation (将正文旋转 90 度)

通过这些属性，我们可以明确定位一个复杂的段落。定位使用 “复合数据类型”的标点符号可以使页面的版面可以在不同的图像分辨率下自动适应：

```
<fo:block

text-indent="1em"
font-family="sans-serif" font-size="12pt"
space-before.minimum="2pt"
space-before.maximum="6pt"
space-before.optimum="4pt"
space-after.minimum="2pt"
space-after.maximum="6pt"
space-after.optimum="4pt">
This handbook covers the major topics in Spanish, but is by no means complete.

<fo:block>
```

如果你的文档有很多标题或段落时，你不需要将所有的文档逐一格式化。有 XSLT ，我们能用 XSLT 把文档写入 HTML 或转换成详细的 XSL:FO 译本。以下是已经转换好的 HTML ：

```
<h3>Introduction</h3>
<p>
This handbook covers the major topics in Spanish, but is by no means complete.
</p>
<h3>Accents</h3>
<p>
When we pronounce English words, one syllable is usually
emphasized (<b>stressed</b>, in linguistic terms).
The stressed syllable is underlined in the following words: com<u>pu</u>ter,<u>lan</u>guage, de<u>vel</u>opment, suc<u>ceeds</u>. Spanish words also have a stressed syllable, and there are rules for determining which syllable carries the emphasis.
</p>
```

下边是标题和段落的模板：

```
<xsl:template match="h3">
```

```
<fo:block font-size="14pt" font-family="sans-serif"
font-weight="bold" color="green"
space-before="6pt" space-after="6pt">
<xsl:apply-templates/>
</fo:block>

</xsl:template>

<xsl:template match="p">

<fo:block
text-indent="1em"
font-family="sans-serif" font-size="12pt"
space-before.minimum="2pt"
space-before.maximum="6pt"
space-before.optimum="4pt"
space-after.minimum="2pt"
space-after.maximum="6pt"
space-after.optimum="4pt">
<xsl:apply-templates/>
</fo:block>

</xsl:template>
```

在这篇文章初始化时已经设定了模板并将<html>和 <body>标记了，我们在这时不用再重复，只需用浏览器检查一下。

这是文档的树结构。用<b> 和 <u> 标记了最后一个子接点。它们在行列中的句柄为<fo:inline>元素。

```
<xsl:template match="b">

<fo:inline font-weight="bold">
<xsl:apply-templates/>
</fo:inline>

</xsl:template>

<xsl:template match="u">

<fo:inline text-decoration="underline">
<xsl:apply-templates/>
</fo:inline>

</xsl:template>

<xsl:template match="i">

<fo:inline font-style="italic">
<xsl:apply-templates/>
</fo:inline>

</xsl:template>
```

当我们通过 XSLT 和 FOP 建立 HTML 文件将得到：

XSL 对象格式使用指南（四）

下面是我们将在文档中增加列表的内容：

1.If a syllable has an accent mark, that syllable always gets the stress: acción (action), teléfono.

2.If the word ends with a vowel, n, or s, the next-to-last syllable gets the stress: amigo, hablan (they talk), animales.

3.All other words are accented on the last syllable: hotel, similar, espa?ol.

一个列表由四个元素构成。<fo:list-block>属性包含单独的<fo:list-items>属性。列表不同的部件被<fo:list-item-label>属性和<fo:list-item-body>属性固定。你可以通过下面的图表所示属性来设定列表的间隔：

A.provisional-distance-between-starts
B.provisional-label-separation
C.start-indent for list-item-label
D.start-indent for list-item-body
E.end-indent for list-item-label
F.end-indent for list-item-body

现在我们来创建一个 XSLT 模板来处理一个规划好的列表。开始先设定列表的各项部件的标签，再通过 FOP 输出。使用相关的 em 间隔，列表将拥有合理的间隔和字体大小：

```
<xsl:template match="ol">

<fo:list-block space-before="0.25em" space-after="0.25em">
<xsl:apply-templates/>
</fo:list-block>

</xsl:template>

<xsl:template match="ol/li">

<fo:list-item space-after="0.5ex">
<fo:list-item-label start-indent="1em">
<fo:block>
<xsl:number/>
</fo:block>
</fo:list-item-label>
<fo:list-item-body>
<fo:block>
<xsl:apply-templates/>
</fo:block>
</fo:list-item-body>
</fo:list-item>

</xsl:template>
```

制作无序列表跟以上类似。在一个无序列表中各部件的相关定义为：

```
<xsl:template match="ul/li">
```

```
<fo:list-item>
<fo:list-item-label start-indent="1em">
<fo:block>
•
</fo:block>
<!-- etc. -->
```

定位列表

使用列表样式创建一个有限的表格并定位它们的条款和解说并不能依靠 XSLT。我们将在分开的行内放入条款和解说，象普通的 HTML 所演示的。

```
<xsl:template match="dl">

<fo:block space-before="0.25em" space-after="0.25em">
<xsl:apply-templates/>
</fo:block>

</xsl:template>

<xsl:template match="dt">

<fo:block>
<xsl:apply-templates/>
</fo:block>

</xsl:template>

<xsl:template match="dd">

<fo:block start-indent="2em">
<xsl:apply-templates/>
</fo:block>

</xsl:template>
```

这儿是小册子的一部分，展示了怎么规划列表和定位列表。注意下面的文本流程不需要我们做任何改动。

表格

下面显示了一个典型的表格。

Singular Plural
yo canto nosotros cantamos
tú cantas vosotros cantáis
él canta
ella canta ellos cantan
ellas cantan

通过 XSL 格式化过的表格有以下元素：

```
<fo:table-and-caption>
```

<fo:table-caption>
<fo:table>
<fo:table-column>
<fo:table-header>
<fo:table-row>
<fo:table-cell>
<fo:table-body>
<fo:table-row>
<fo:table-cell>
<fo:table-footer>
<fo:table-row>
<fo:table-cell>

<fo:table> 属性相当于 HTML 的<table>标签； <fo:table-body>属性相当于 HTML 的<tbody>属性。注意只需要定义 <fo:table-column> 属性，它允许你指定表格队列的宽度。你也可以用标签定义单元格具有相同的队列和范围。<table-and-caption> 元素在当前的 FOP 执行中不能实现。你必须在<fo:table-column>元素定义 column-width 属性来调整表格队列的宽度。FOP 不能自动调节并显示你的表格的宽度。

XSLT 可以制作简单的表格，假定已经定位了第一行表格的宽度，还得确定是 72 象素/英寸宽度单位。但还没有处理行和列的跨距。请看代码示例。

第三人称需要一个
标签，可以用 FO 转换成：

<xsl:template match="br">

<fo:block><xsl:text> </xsl:text></fo:block>

</xsl:template>

表格处理结果显示为：

概要

正如你见到的，XSLT 和 FO 相结合允许你将 XHTML 文档 或其他 XML 文档转换成印刷格式。用 XSL 的对象格式化功能只能做初步的版面设计。

XSL 简明教程(1)XSL 入门

一. XSL 入门

二. XSL 的转换
三. XSL --- 在客户端的实现
四: XSL --- 在服务器端的实现
五. XSL 的索引
六. XSL 的过滤和查询
七. XSL 的控制语句
一. XSL 入门
1.XSL---XML 的样式表
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。
2.XSL --- 不仅仅是一种样式表
XSL 由两部分组成：
一是转化 XML 文档；二是格式化 XML 文档。
如果你不理解这个意思，可以这样想：XSL 是一种可以将 XML 转化成 HTML 的语言，一种可以过滤和选择 XML 数据的语言，一种能够格式化 XML 数据的语言。(比如用红色显示负数。)
3.XSL --- 它能做什么？
XSL 可以被用来定义 XML 文档如何显示,可以将 XML 文档转换成能被浏览器识别的 HTML 文件，通常的，XSL 是通过将每一个 XML 元素"翻译"为 HTML 元素，来实现这种转换的。XSL 能够向输出文件里添加新的元素，或则移动元素。XSL 也能够重新排列或者索引数据，它可以检测并决定哪些元素被显示，显示多少。
4.XSL 在 IE5 中的显示
注意：IE5.0 中，并不能完全兼容 W3C 组织发布的最新 XSL 标准。因为 IE5.0 是在 XSL 标准最终确定以前发布的。微软已经承诺在 IE5.5 中修正。

XSL 简明教程(2)XSL 转换

二.XSL 的转换

1.将 XML 转换成 HTML
XSL 是如何将 XML 文档转换成 HTML 文件的呢？我们来看一个例子，下面是 XML 文档的一部分：

```
<?xml version="1.0" encoding="ISO8859-1" ?>
<CATALOG>
<CD>
<TITLE>Empire Burlesque</TITLE>
<ARTIST>Bob Dylan</ARTIST>
<COUNTRY>USA</COUNTRY>
<COMPANY>Columbia</COMPANY>
<PRICE>10.90</PRICE>
<YEAR>1985</YEAR>
</CD>
...
```

然后我们将下面的 XSL 文件作为 HTML 的模板将 XML 数据转换为 HTML 文件：

```
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
<html>
<body>
```

```
<table border="2" bgcolor="yellow">
<tr>
<th>Title</th>
<th>Artist</th>
</tr>
<xsl:for-each select="CATALOG/CD">
<tr>
<td><xsl:value-of select="TITLE"/></td>
<td><xsl:value-of select="ARTIST"/></td>
</tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

在上面的代码中， xsl:for-each 元素的作用是定位 XML 文档中的哪些元素需要按以下模板显示。select 属性用来定义源文件中的元素名。指定属性的这种语法又称为 XML Pattern(模式)，类似文件子目录的表示形式。xsl:value-of 元素用来在当前层次中插入子元素的内容模板。
因为 XSL 样式表自身也是一个 XML 文档，因此，XSL 文件的开头以一个 XML 声明开始。xsl:stylesheet 元素用来声明这是一个样式表文件。<xsl:template match="/">语句表示 XML 的源文档在当前目录下。
如果为 XML 文档加上 XSL 样式表，看下面代码第 2 行，你的浏览器就可以精确的将 XML 文档转换为 HTML 文件。

```
<?xml version="1.0" encoding="ISO8859-1" ?>
<?xml-stylesheet type="text/xsl" href="cd_catalog.xsl"?>
<CATALOG>
<CD>
<TITLE>Empire Burlesque</TITLE>
<ARTIST>Bob Dylan</ARTIST>
<COUNTRY>USA</COUNTRY>
<COMPANY>Columbia</COMPANY>
<PRICE>10.90</PRICE>
<YEAR>1985</YEAR>
</CD>
```

XSL 简明教程(3)在客户端的实现

三. XSL--在客户端的实现

1.JavaScript 解决方案
在上面的章节中我们已经解释了 XSL 是如何将 XML 转换成 HTML 文件。方法就是在 XML 文档的头部加入一个 XSL 样式表信息，然后让浏览器执行转换过程。
这种方法在大部分情况下都做得很好，但是在不支持 XML 的浏览器中就无法正确显示了。
一个更好的更全面的解决方案是使用 Javascript 来实现 XML 到 HTML 的转换。但是使用

JavaScript 必须得到以下功能支持：
a.允许 Javascript 代替浏览器进行细节检测；
b.根据不同的需要和不同的浏览器使用不同的样式表。
对于XSL来说这是完全可行的。设计XSL的目标之一就是允许将一种格式转换成另一种格式，支持不同的浏览器，支持不同的用户需求。未来的浏览器的重要任务就是在客户端执行 XSL 的转换工作。

2.一个具体的实例
下面是我们上面提到的一个 XML 文档(cd_catalog.xml)例子的部分代码：

```
<?xml version="1.0" encoding="ISO8859-1" ?>
<CATALOG>
<CD>
<TITLE>Empire Burlesque</TITLE>
<ARTIST>Bob Dylan</ARTIST>
<COUNTRY>USA</COUNTRY>
<COMPANY>Columbia</COMPANY>
<PRICE>10.90</PRICE>
<YEAR>1985</YEAR>
</CD>
.
.
.
```

下面是完整的 XSL 文件(cd_catalog.xsl)：

```
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
<html>
<body>
<table border="2" bgcolor="yellow">
<tr>
<th>Title</th>
<th>Artist</th>
</tr>
<xsl:for-each select="CATALOG/CD">
<tr>
<td><xsl:value-of select="TITLE"/></td>
<td><xsl:value-of select="ARTIST"/></td>
</tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

注意，现在 XML 文件还没有加入 XSL 样式表，还没有被转换成 HTML 文件。
下面是用 JavaSript 来实现最后转换的 HTML 代码：

```
<html>
<body>
```

```
<script language="javascript">
// Load XML
var xml = new ActiveXObject("Microsoft.XMLDOM")
xml.async = false
xml.load("cd_catalog.xml")

// Load the XSL
var xsl = new ActiveXObject("Microsoft.XMLDOM")
xsl.async = false
xsl.load("cd_catalog.xsl")

// Transform
document.write(xml.transformNode(xsl))
</script>

</body>
</html>
```

上面代码中使用了 Javascript，如果你不知道如何写 JavaScript，您最好专门学习一下。第一段代码建立一个 Microsoft Parser(XMLDOM)解析的对象，并将 XML 文档读入内存；第二段代码建立另外一个对象并导入 XSL 文档；最后一行代码将 XML 文档用 XSL 文档转换，并将结果输出到 HTML 文件中。

XSL 简明教程(4)在服务器端的实现

四: XSL --- 在服务器端的实现

1.兼容所有的浏览器
在上面一章我们介绍了可以通过 JavaScript 调用浏览器的 XML parser(解析软件)来转换 XML 文档。但是这个方案依然有个问题：如果浏览器没有 XML
parser 插件怎么办？(注：IE5 内自带 XML parser)
为了使我们的 XML 数据能被所有的浏览器正确显示,我们不得不在服务器端将 XML 转换成纯 HTML 代码，再输出给浏览器。
这也是使用 XSL 的另一个好处。在服务器端将一种格式转换为另一种格式也是 XSL 的设计目标之一。
同样，转换工作也将成为未来服务器段的主要工作。

2.一个具体实例
下面是我们上面提到的一个 XML 文档(cd_catalog.xml)例子的部分代码：

```
<?xml version="1.0" encoding="ISO8859-1" ?>
<CATALOG>
<CD>
<TITLE>Empire Burlesque</TITLE>
<ARTIST>Bob Dylan</ARTIST>
<COUNTRY>USA</COUNTRY>
<COMPANY>Columbia</COMPANY>
<PRICE>10.90</PRICE>
<YEAR>1985</YEAR>
```

```
</CD>
.
.
.
```

下面是完整的 XSL 文件(cd_catalog.xsl)：

```
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
<html>
<body>
<table border="2" bgcolor="yellow">
<tr>
<th>Title</th>
<th>Artist</th>
</tr>
<xsl:for-each select="CATALOG/CD">
<tr>
<td><xsl:value-of select="TITLE"/></td>
<td><xsl:value-of select="ARTIST"/></td>
</tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

下面是在服务器端转换 XML 文件为 HTML 文件的原代码：

```
<%
'Load the XML
set xml = Server.CreateObject("Microsoft.XMLDOM")
xml.async = false
xml.load(Server.MapPath("cd_catalog.xml"))

'Load the XSL
set xsl = Server.CreateObject("Microsoft.XMLDOM")
xsl.async = false
xsl.load(Server.MapPath("cd_catalog.xsl"))

Response.Write(xml.transformNode(xsl))
%>
```

注意：我们这里的例子采用的是 ASP 文件，用 VBScript 编写的。如果您不了解 ASP 或者 VBScript，建议阅读有关书籍。(当然，也可以采用其他的语言编写服务器端程序)

第一段代码建立一个 Microsoft Parser(XMLDOM)解析的对象，并将 XML 文档读入内存；第二段代码建立另外一个对象并导入 XSL 文档；最后一行代码将 XML 文档用 XSL 文档转换，并将结果输出到 HTML 文件中。

XSL 简明教程(5)XSL 的索引

五. XSL 的索引

如果我需要将元素的显示按一定的顺序排列，应该如何建立 XSL 的索引呢？
我们还是来看前面的例子，还是这段代码：

```
<?xml version="1.0" encoding="ISO8859-1" ?>
<CATALOG>
<CD>
<TITLE>Empire Burlesque</TITLE>
<ARTIST>Bob Dylan</ARTIST>
<COUNTRY>USA</COUNTRY>
<COMPANY>Columbia</COMPANY>
<PRICE>10.90</PRICE>
<YEAR>1985</YEAR>
</CD>
.
.
.
```

当 XML 文档被转换成 HTML 文件，索引应该同时建立。简单的办法就是给你的 for-each 元素增加一个 order-by 属性，就象这样：

```
<xsl:for-each select="CATALOG/CD" order-by="+ ARTIST">
```

order-by 属性带有一个"+"或者"-" 的符号，用来定义索引的方式，是升序还是降序排列。符号后面的名字就是要索引的关键字。
例如(cd_catalog_sort.xsl)：

```
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
<html>
<body>
<table border="2" bgcolor="yellow">
<tr>
<th>Title</th>
<th>Artist</th>
</tr>
<xsl:for-each select="CATALOG/CD" order-by="+ ARTIST">
<tr>
<td><xsl:value-of select="TITLE"/></td>
<td><xsl:value-of select="ARTIST"/></td>
</tr>
</xsl:for-each>
```

```
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

最后，我们用下面的 HTML 代码来显示索引结果，你可以自己尝试一下。

```
<html>
<body>
<script language="javascript">
// Load XML
var xml = new ActiveXObject("Microsoft.XMLDOM")
xml.async = false
xml.load("cd_catalog.xml")

// Load the XSL
var xsl = new ActiveXObject("Microsoft.XMLDOM")
xsl.async = false
xsl.load("cd_catalog_sort.xsl")

// Transform
document.write(xml.transformNode(xsl))
</script>

</body>
</html>
```

XSL 简明教程(6)XSL 过滤和查询

六. XSL 的过滤和查询

如果我们希望只显示满足一定的条件的 XML 数据应该怎么做呢？还是上面的例子代码，我们只需要在 xsl:for-each 元素的 select 属性中加入参数就可以，类似：

```
<xsl:for-each select="CATALOG/CD[ARTIST='Bob Dylan']">
```

参数的逻辑选择有：

= (等于)
=! (不等于)
&LT& 小于
&GT& 大于等于

和前面同样的例子(cd_catalog_sort.xsl)：

```
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
<html>
<body>
<table border="2" bgcolor="yellow">
<tr>
<th>Title</th>
<th>Artist</th>
</tr>
<xsl:for-each select="CATALOG/CD[ARTIST='Bob Dylan']">
<tr>
<td><xsl:value-of select="TITLE"/></td>
<td><xsl:value-of select="ARTIST"/></td>
</tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

你可以自己测试一下，看到的结果有什么不同。

XSL 简明教程(7)XSL 的控制语句

七. XSL 的控制语句

1.条件语句 if...then
XSL 同样还有条件语句(呵呵~~好厉害吧，象程序语言一样)。具体的语法是增加一个 xsl:if 元素，类似这样

```
<xsl:if match=".[ARTIST='Bob Dylan']">
... some output ...
</xsl:if>
```

上面的例子改写成为：

```
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
<html>
<body>
<table border="2" bgcolor="yellow">
<tr>
```

```
<th>Title</th>
<th>Artist</th>
</tr>
<xsl:for-each select="CATALOG/CD">
<xsl:if match=".[ARTIST='Bob Dylan']">
<tr>
<td><xsl:value-of select="TITLE"/></td>
<td><xsl:value-of select="ARTIST"/></td>
</tr>
</xsl:if>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

2. XSL 的 Choose

choose 的用途是出现多个条件，给出不同显示结果。具体的语法是增加一组 xsl:choose,xsl:when,xsl:otherwise 元素：

```
<xsl:choose>
<xsl:when match=".[ARTIST='Bob Dylan']">
... some code ...
</xsl:when>
<xsl:otherwise>
... some code ....
</xsl:otherwise>
</xsl:choose>
```

上面的例子改写成为：

```
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
<html>
<body>
<table border="2" bgcolor="yellow">
<tr>
<th>Title</th>
<th>Artist</th>
</tr>
<xsl:for-each select="CATALOG/CD">
<tr>
<td><xsl:value-of select="TITLE"/></td>
```

```
<xsl:choose>
<xsl:when match=".[ARTIST='Bob Dylan']">
<td bgcolor="#ff0000"><xsl:value-of select="ARTIST"/></td>
</xsl:when>
<xsl:otherwise>
<td><xsl:value-of select="ARTIST"/></td>
</xsl:otherwise>
</xsl:choose>
</tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

XSL 语法介绍

例子已经放在上面，下面我们可以来仔细的分析其中的语法结果和关键所在：

首先注意到的是，XSL 文件本身即是一份 XML 文件，所以在 XSL 文件的开头，一样有和 XML 文件相同的声明。W3C 这个 XML 的标准机构为 XSL 定义了很多标记（元素），XSL 文件就是这些标记和 HTML 标记的组合。在 XSL 文件中，必须有如下一行的代码：

```
＜xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl"＞
```

这里，xsl:stylesheet 是 XSL 文件的根元素，在根元素中包含了所有的排版样式，样式表就是由这些排版样式组合成的；xmlns:xsl="http://www.w3.org/TR/WD-xsl"这一句主要用来说明该 XSL 样式表是使用 W3C 所制定的 XSL，设定值就是 XSL 规范所在的 URL 地址。

实际上，这里"http://www.w3.org/TR/WD-xsl"就是一个名字空间（namespace），我们在上面关于 XML Schema 语法介绍的时候已经进行过介绍。这是一个标准的名字空间。"stylesheet"，"template"，"for-each"等等关键字都是这个名字空间所定义的。

当然在 xsl:stylesheet 还可以设定其他的属性，其他的属性有：

1. default-space：决定是否保留 XML 文件中的空白，仅当值为"default"时保留。
2. indent-result：决定是否保留 XSL 文件中的空白，值为"yes"时保留。
3. language：设定在 XSL 文件中使用的脚本语言。

然后，我们在上面的代码中看到有如下的代码：

```
＜xsl:template match="/"＞
    ……
＜/xsl:template＞
```

这里实际上是表示了 XSL 解析器对 XML 文档的处理过程，它从根节点（由 match="/" 决定，这里"/"就表示根节点）开始，对 XML 文档进行遍历，并根据具体的代码从 XML 文档中取出相关的内容。这里关于属性 match 的取值是一个比较复杂的问题。它实际上表示的含义是从 XML 文档中取出一个特定的节点集合（XML 文档可以被看成一个树的结构，这个在后面关于 XML 解析器分析中有详细的介绍）。这里，我们通过几个简单的例子来说明属性 match 的取值。

比如下面一行代码：

＜xsl:template match="/"＞

这行代码的意思是告诉 XSL 解析程序，当前需要处理的节点是根节点下的内容（用"/"来表示根节点），其实这里的 match 值内容的编写是要符合 XPath 的语意。关于 XPath 我们在后面的章节中会进行详细的介绍。

再举一个例子：

＜xsl:template match="shoppingcart/item"＞

这行代码要匹配的是 shoppingcart 元素下的 item 元素。而不管 shoppingcart 在 XML 文档结构树下的哪一个位置。比如 XML 文档的其中一段是这样的。

```
＜shoppingcart＞
＜item＞
＜itemNo＞3333＜/itemNo＞
＜itemName＞屠龙刀＜/itemName＞
＜/item＞
＜item＞
＜itemNo＞4444＜/itemNo＞
＜itemName＞离别钩＜/itemName＞
＜/item＞
＜/shoppingcart＞
```

那么它匹配的内容就是

```
  ＜item＞
  ＜itemNo＞3333＜/itemNo＞
  ＜itemName＞屠龙刀＜/itemName＞
  ＜/item＞
  ＜item＞
  ＜itemNo＞4444＜/itemNo＞
  ＜itemName＞离别钩＜/itemName＞
  ＜/item＞
```

而下面这个例子：

＜xsl:template match="/shoppingcart/item"＞

表示只匹配 XML 文档根节点下的 shoppingcart 这个节点下的所有 item 元素。也就是说如果 shoppingcart 不是直接在根节点下的，它就不符合这个匹配条件。

再看一个例子：

＜xsl:template match="price[@unit='USD']"＞

这个例子说明要匹配的是这样的节点：一个带 unit 属性的元素 price，而且 unit 的值必须为 "USD"。比如一段 XML 代码是这样的。

```
＜price＞
  ＜unit＞USD＜/unit＞
  ＜amount＞100＜/amount＞
＜/price＞
＜price＞
  ＜unit＞RMB＜/unit＞
  ＜amount＞300＜/amount＞
＜/price＞
```

那么它匹配的内容就是：

```
＜price＞
  ＜unit＞USD＜/unit＞
  ＜amount＞100＜/amount＞
＜/price＞
```

实际上，存在许多各种符号用来表示匹配规则，我们在 XPath 语法介绍中会详细涉及到。现在知道这么一个大概的概念就可以了。

我们用＜xsl:template match="具体匹配表达式"＞这条语句找到了一些节点集合以后，我们就要从这个集合中找到特定的元素或者元素属性的值，那么采用什么语句呢？就是用 xsl:value-of select = ""这样的语句来寻找特定的内容。

比如下面的例子中＜xsl:value-of select="名称"/＞这行代码就是表示定位 XML 文档中的名称元素的内容。在指定集合中可能存在多个名字元素，如果我们需要把它们一一列举出来进行处理的话，就需要用到语句 xsl:for-each select = ""，注意这里涉及到一个作用范围的概念，也就是说 xsl:for-each select = ""这条语句是在一个指定的集合空间中执行的。比如上面例子中如下的代码

这里的＜xsl:for-each select="词语"＞是在＜xsl:template match="网络用语集合"＞所指定的集合空间里面寻找元素"词语"的。

同时，我们需要注意的是上面的代码中，出现了一条语句

```
  ＜xsl:apply-templates select="网络用语集合" /＞
```

它表示什么意思呢，它实际上相当于 C++中的一个过程调用，当 XSL 解析器执行到该语句的时候，它就会在代码中寻找以＜?xml:namespace prefix = xsl /＞＜xsl:template match="网络用语集合"＞开头的代码，所以在上面的例子程序中，以下的代码可以看成是过程的实现。

把看成是一个过程调用，把＜xsl:template match="网络用语集合"＞

```
…….
＜/xsl:template＞
```

看成是过程的实现，有助于我们对 XSL 解析器执行过程的理解。这里 match="网络用

语集合"可以理解为是传递给过程的参数，它表示过程实现体的集合范围是该 match 所匹配的节点集合空间（"网络用语集合"）。

如果我们要对表格中的元素进行排序该什么办呢？比如说，在上面的例子中，我们需要按照名称进行排序。很简单对，改写为如下的形式即可：

＜xsl:for-each select="词语" order-by="+名称"＞，其中"+"表示按降序排列；"-"表示按升序排列。"order-by"是 XSL 语法中的关键字。

如果我们只想在列表中取出某几行该怎么操作呢？比如我们只想取出名称为"恐龙"的行，见下面的代码：

这里有一个新的句法为：＜xsl:template match="网络用语集合"＞

```
＜table Border="1"＞
  ＜xsl:for-each select="词语" order-by="-名称"＞
  ＜xsl:if test=".[名称='恐龙']"＞
  ＜tr＞
    ＜td＞＜xsl:value-of select="名称"/＞＜/td＞
    ＜td＞＜xsl:value-of select="含义"/＞＜/td＞
  ＜/xsl:if＞
  ＜/xsl:for-each＞
＜/table＞
＜/xsl:template＞
```

它表示如果".[名称='恐龙']"为真（TRUE）的话，就执行该段里面的语句，要是为假（FALSE）的话就不执行。它和 C++中的 if 语句的概念基本是一样的。

前面我们用＜xsl:value-of select=""/＞取出的都是一个元素的值，但是我们要取出元素某一个属性的值该怎么做呢？采用下面的形式：

＜xsl:value-of select="元素名称/@属性名称"/＞

比如一段 XML 代码是这样的：

＜王朔 网址="www.wangshuo.com"＞知名作家加著名评论家王朔先生的地方＜/王朔＞

我们可以用＜xsl:value-of select="王朔/@网址"/＞来得到值"www.wangshuo.com"。

以上包括了 XSL 的大多数基本的语法，更加详细和完整的介绍需要参看 W3C 相关的最新的文档，可以在 WWW.W3C.ORG/TR 下找到。

根据上面的分析，我们可以看到 XSL 实际上采用的是一种转换的思想，它最终将 XML 文档转换为另一种可用于输出的文档，而 CSS 则没有任何转换动作，在整个过程中没有任何新码产生。 另外，在 XSL 中 90%的样式规定在 CSS 中都有定义，但仍然有一些效果是 CSS 无法描述的，必须使用 XSL 不可。这些功能包括文本的置换、根据文本内容决定显示方式、文档内容排序等，都是 XSL 所独有的。再者，XSL 遵从 XML 的语法，而 CSS 的语法自成体系。

选择样式单还要考虑不同浏览器对样式单的支持程度。目前 IE5 与 Netscape 的最新版

本都支持 CSS，但支持的程度都有限。至今为止，IE5 尚不能完全支持 CSS1，即便是支持的部分也存在很多错误，对于 CSS2 也只提供部分支持。Netscape 在对 CSS 的支持上已经优于 IE5，它采用新一代的 Raptor/Gecko 引擎技术，已经能够完全支持 CSS1，但对 CSS2 的支持计划尚不明朗。而对 XSLT 而言，只有 IE5 支持，Netscape5 并不支持。

跟我学 XSL（一）

第一个 XML 文档

随着 Internet 的发展，越来越多的信息进入互联网，信息的交换、检索、保存及再利用等迫切的需求使 HTML 这种最常用的标记语言已越来越捉襟见肘。HTML 将数据内容与表现融为一体，可修改性、数据可检索性差，而 XML 借鉴了 HTML 与数据库、程序语言的优点，将内容与表现分开，不仅使检索更为方便，更主要的是用户之间数据的交换更加方便，可重用性更强。

XML 是一种元标记语言，没有许多固定的标记，为 WEB 开发人员提供了更大的灵活性。当我们使用 HTML 时，标记只是简单的表示内容的显示形式，而与表示的内容没有任何关联，为文档的进一步处理带来极大的不便。比如要表示个人简历，用 HTML 的表示方式如下：

```
< HTML >

< BODY >

< TABLE border=1 cellspacing=0 >

< TH >姓名< TD >禹希初< TH >性别< TD >男< TH >生日< TD >1977.5

< TR >

< TH >技能< TD colspan=5 >数据库设计与维护、WEB 开发

< /TABLE >

< /BODY >

< /HTML >
```

在这里，我们无法从标记 TH、TD 得知其内容表示什么，如果用 XML，相应的文档（文件名：个人简历.xml）就可写成如下形式：

```
< ?xml version="1.0" encoding="GB2312"? >

< resume >

< name >禹希初< /name >

< sex >男< /sex >

< birthday >1977.5< /birthday >

< skill >数据库设计与维护、WEB 开发< /skill >
```

< /resume >

说明：

version──规定了 XML 文档的版本，此处只能是 1.0；

encoding── 此处规定了 XML 文档的编码类型，此处取值为“GB2312”，也就是“简体中文”。

对比两例，使用 XML 我们可以做到自定义标记，用标记表明内容的含义。这样在 Internet 上交流资料时，为用计算机处理文档提供了极大的方便，同时我们阅读源文件时也不会被一大堆格式弄得晕头转向。

然而，由于 XML 并没有为标记规定显示方式，如果我们在游览器中查看以上两个文档（建议使用 IE5.0 或更新版本），我们将看到 xml 文档并没有以诸如表格的方式来显示。难道我们就不能像 HTML 一样显示文档吗？回答是否定的。以个人简历为例，需要另建一个格式文件说明各个标记的显示方式，其内容如下（假设文件名为 resume.css）：

```
resume{ display: block;}

name{ display: block; font-size:120%;}

sex{ display:block; text-indent:2em}

birthday{ display:block; text-indent:2em}

skill{ display:block; text-indent:2em}
```

说明：

以上均为 CSS 样式，建议读者参考有关资料熟悉 CSS，在以后学习中必须用到，此处由于篇幅关系不作介绍。建立文件 resume.css 后，在个人简历.xml 文件的第一行后添加以下文字：

```
< ?xml:stylesheet type="text/css" href="resume.css"? >
```

说明：

此处表示引用一个外部 CSS 样式文件，其中 type 规定样式类型（可取值为 text/css 或 text/xsl），href 规定文件路径。

保存文件，再以 IE5.0 打开文件。怎么样？格式有些不一样吧。好象还不令人满意，文档内容是清晰了，但显示效果比 HTML 编写的文档就差得多了，XML 编写的文档就只能以这种方式显示吗？！

提示：

1. 为了更好的理解与掌握 XML，建议大家熟悉 HTML 4.0 与 CSS 2.0 语法；掌握 JavaScript、VBscript 中至少一种；编程经验、对数据库理论与 SQL 的了解均能使大家在学习 XML 时获益。

2. XML 文档中标记必须成对出现，如果是空标记也必须有前加“/”的同名标记结束，或使用此种文式< xml_mark/ >表示空标记。

3. XML 以及下周将要介绍的 XSL 文档，属性值必须用双引号（"）或单引号（'）括起来。

4. XML 文档必须是好结构的（XSL 文档也是 XML 文档中一种），也就是说标记必须有结束标记、标记可以嵌套但不可交叉，如

< outer >< inner >< /inner >< inner/ >< /outer >

是合法的，而下面的形式

< outer >< inner >< /outer >< /inner >

则是错误的。如果 XML 文档在浏览时出错，多半是违反了上面提到的规则。

跟我学 XSL（二）

XSL 入门

上期我们讲到用 CSS（层叠样式表）来格式化 XML 文档，其效果并不很令人满意。实际上 CSS 用来格式化 HTML 标记比较合适些，只是因为它简单才在上例中采用。

XML 在更多的时候只是一种数据文件，怎样将它变为我们日常所看到的 HTML 格式那样的文件呢？如果我们将 XML 文件比作结构化的原料的话，那么 XSL 就好比“筛子“与“模子“，筛子选取自己需要的原料，这些原料再通过模子形成最终的产品：HTML。

这个模子大致是这样：我们先设计好表现的页面，再将其中需要从 XML 中获取数据来填充内容的部分“挖掉“，然后用 XSL 语句从 XML 中筛出相关的数据来填充。一言以譬之：这 XSL 实际上就是 HTML 的一个“壳子“，XML 数据利用这个“壳“来生成“传统“的 HTML。

XML 在展开时是一个树形结构，我们将树形结构中自定义标记称为节点，节点之间存在父子、兄弟关系，我们要访问其中的结点从根结点就要以” / ”来层层进入。

在 XSL 这个壳中，我们要从原料库？？XML 里提取相关的数据，就要用到 XSL 提供的模式化查询语言。所谓模式化查询语言，就是通过相关的模式匹配规则表达式从 XML 里提取数据的特定语句，即我们上所说的“筛子“。

参考微软的”XSL 开发者指南”，我们大致可将模式语言分为三种：

选择模式
< xsl:for-each >、< xsl:value-of >,和 < xsl:apply-templates >

测试模式
< xsl:if > 和< xsl:when >

匹配模式
< xsl:template >
我们现在就分别对之进行介绍。
一、 选择模式

选择模式语句将数据从 XML 中提取出来，是一种简单获得数据的方法，这几个标记都有一个 select 属性，选取 XML 中特定的结点名的数据。
1、< xsl:for-each >
如在 XML 中有这样的数据:
< author >

< name >小禹< /name >
< name >春华< /name >
< name >秋实< /name >
< /author >

我们要读取这三个作者名字，是一个一个地按”author/name”方法来读取吗，可有多个这样的 name 呀？如果有一种程序性的语句来循环读取有多好啊！

想得很对，XSL 提供了这样的具有程序语言性质的语句< XSL:for-each >

用它读取这三个作者名字的方法如下:

< xsl:for-each select=”author/name” >
…….
< ./xsl:for-each >

select，顾名思义：选取，它可以选定 XML 中特定唯一的标记，也可以选择某一类相同的标记，我们称之为结点集。

语法：

< xsl:for-each select="pattern" order-by="sort-criteria-list">

属性：

1. select

根据 XSL 样式查询考察上下文以决定哪类结点集（满足 select 条件）使用此样式描述。作为一种简化的表示就是，如果你想对文档中的某一种标记的内容的显示方式进行格式化，就可以将让 select 等于此元素的标记名。例如欲对标记 xml_mark 进行格式化，即可用如下方式表示：

< xsl:for-each select="xml_mark" >
< !--样式定义-- >
< /xsl:for-each >

2. order-by

以分号（;）分隔、作为排序标准的列表。在列表元素前添加加号（+）表示按此标记的内容以升序排序，添加减号（-）表示逆序排序。作为一种简化的表示就是，排序标准列表就是由 select 规定的标记的子标记的序列，每个标记之间以（;）分隔。

2、< xsl:value-of >

< xsl:for-each >模式只是选取节点，并没有取出节点的值，好比猴子只是爬到了树的某个枝干上，那么就用< xsl:value-of >来摘“胜利果实“吧！

语法：

< xsl:value-of select="pattern" >提取节点的值

属性：

select 用来与当前上下文匹配的 XSL 式样。简单的讲，如果要在 XSL 文档某处插入某个 XML 标记（假定是 xml_mark 标记）的内容，可用如下方式表示：

< xsl:value-of select="xml_mark" >< /xsl:value-of >

或

< xsl:value-of select="xml_mark"/ >

示例：

此处仍以上期的个人简历的作为例子，我们需要对文件（个人简历.xml）作一定修改，确切的说是将其中的第二行

< ?xml:stylesheet type="text/css" href="resume.css"? >

修改为

< ?xml:stylesheet type="text/xsl" href="resume.xsl"? >

然后建立一个新文件：resume.xsl，其内容如下：

```
< ?xml version="1.0" encoding="GB2312"? >
< HTML xmlns:xsl="http://www.w3.org/TR/WD-xsl" >
< HEAD >
< TITLE >个人简历< /TITLE >
< /HEAD >< BODY >
```

< xsl:for-each select="resume" >
< P/ >
< TABLE border="1" cellspacing="0" >
< CAPTION style="font-size: 150%; font-weight: bold" >
个人简历
< /CAPTION >
< TR >
< TH >姓名< /TH >< TD >< xsl:value-of select="name"/ >< /TD >
< TH >性别< /TH >< TD >< xsl:value-of select="sex"/ >< /TD >
< TH >生日< /TH >< TD >< xsl:value-of select="birthday"/ >< /TD >
< /TR >
< TR >
< TH >技能< /TH >< TD colspan="5" >< xsl:value-of select="skill"/ >< /TD >
< /TR >
< /TABLE >
< /xsl:for-each >
< /BODY >
< /HTML >

完成这些以后再来让我们看一下辛勤劳动的成果，怎么样？效果不错吧。更酷还在后头呢。现在我们对文件（个人简历.xml）作进一步的修改：

1. 在标记< resume >前添加一个新标记< document >；
2. 将标记对< resume >< /resume >之间的内容（包括这一对标记）复制并粘贴在其后，并在最后用< document >结束。
3. 以 Notepad.exe 打开文件 resume.xsl，在标记< HTML >之后添加文字：< xsl:for-each select="document" >；在标记< /HTML >之前添加文字：< /xsl:for-each >，保存文件。
4. 在浏览器中打开文件（个人简历.xml）。看到了什么？两份个人简历！

就这样，利用 XML 我们可以编写内容与样式完成分离的文档！当然，XSL 文件比一般的 HTML 文件要复杂一些，然而一旦完成则可用于格式化所有同类的 XML 文档。

注：如果拷贝代码，请将空格删除

跟我学 XSL（三）

XSL 模板与匹配模式

经过前几日的学习，我们学习了 XHTML 文档的编写和以及三个 XSL 元素，已能编写相当灵活的 XSL 文档，今天将学习的是 XSL 模板的编写。我们都知道，短的文档、程序十分好读，但当规模增大后，其复杂性也以更快的速度增加。

前面我们学了< xsl:for-each >、< xsl:value-of >等，可以用它们对 XML 数据实现简单的格式化输出，但如果遇到比较复杂的 XML 格式输出，将 XSL 按照要求依次写下来的话，一是设计困难，可扩展性差，不利于人员之间的分工协作；另则，可修改性很差，可能会出现牵一发而动全军的情况，不利于维护。程序中模块化设计逐步细化的方法在这里得到了应用！

XSL 模板将 XSL 的设计细化成一个个模板（块），最后再将这些模板（块）组合成一个完整的 XSL；好比船与集装箱，我们不是将所有的货物一件件地堆起来，而是装在各自的集装箱中，然后再在船上将这些集装箱堆放起来。这种方法可以使你先从整体上考虑整个 XSL 的设计，然后将一些表现形式细化成不同的模块，再具体设计这些模块，最后将它们整合在一起，这样，将宏观与微观结合起来，符合人们条理化、规范化要求。

装集装箱？？书写模板（块）：< xsl:template >

< xsl:template >
语法：
< xsl:template match="node-context" language="language-name" >

属性：

match —— 确定什么样的情况下执行此模板。作为一种简化的说明，在此处使用标记的名字；其中最上层模板必须将 match 设为“/”

language —— 确定在此模板中执行什么脚本语言，其取值与 HTML 中的 SCRIPT 标记的 LANGUAGE 属性的取值相同，缺省值是 Jscript

< xsl:template >用 match 属性从 XML 选取满足条件的节点，征对这些特定的节点形成一个特定输出形式的模板。

吊集装箱上船-？？调用模板（块）：< xsl:apply-templates >

< xsl:apply-templates >

语法：

< xsl:apply-templates select="pattern" order-by="sort-criteria-list" >

属性：

select —— 确定在此上下文环境中应执行什么模板，即选取用< xsl:template >标记建立的模板（块）。

order-by —— 以分号（;）分隔的排序标准，通常是子标记的序列

示例：

以个人简历为例，为便于处理我们希望“技能”中每一项都用标记对< skill >< /skill >括起来，有多少项技能就有多少个这种标记对，经过修改后的个人简历 XML 文档内容如下：

```
< ?xml version="1.0" encoding="GB2312"? >
< ?xml:stylesheet type="text/xsl" href="resume_template.xsl"? >
< document >
< resume >
< name >禹希初< /name >
< sex >男< /sex >
< birthday >1977.5< /birthday >
< skill >数据库设计与维护< /skill >
```

```
< skill >WEB 开发< /skill >
< /resume >
< /document >
```

然后，建立一个新 XSL 文件 resume_template.xsl，采用模板的形式，其内容如下：

```
< ?xml version="1.0" encoding="GB2312"? >
< xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl" >
< !--根模板-- >
< xsl:template match="/" >
< HTML >< HEAD >< TITLE >个人简历< /TITLE >< /HEAD >
< BODY >
< xsl:apply-templates select="document/resume"/ >
< /BODY >
< /HTML >
< /xsl:template >
< !--简历模板-- >
< xsl:template match="resume" >
< TABLE border="1" cellspacing="0" >
< CAPTION >个人简历（
< xsl:eval >formatIndex(childNumber(this), "I")< /xsl:eval >
）< /CAPTION >
< xsl:apply-templates select="name" / >
< xsl:apply-templates select="sex" / >
< xsl:apply-templates select="birthday" / >
< TR/ >
< TD >技能< /TD >< TD COLSPAN="5" >
< TABLE cellspacing="0" >
< xsl:apply-templates select="skill"/ >
< /TABLE >
< /TD >
< /TABLE >
< BR/ >
< /xsl:template >
< !--姓名模板-- >
< xsl:template match="name" >< TD >姓名< /TD >
< TD >< xsl:value-of/ >< /TD >
< /xsl:template >
< !--性别模板-- >
< xsl:template match="sex" >< TD >性别< /TD >
< TD >< xsl:value-of/ >< /TD >
< /xsl:template >
< !--生日模板-- >
< xsl:template match="birthday" >< TD >生日< /TD >
< TD >< xsl:value-of/ >< /TD >
< /xsl:template >
< !--技能模板-- >
< xsl:template match="skill" >
< TR >< TD >< xsl:value-of/ >< /TD >< /TR >
< /xsl:template >
< /xsl:stylesheet >
```

保存文件，打开文件（个人简历.xml），效果令人满意吧。其实要做到同样的效果，用前面三周介绍的方法也可做，但你得把它作为一整体考虑。

在上面的 XSL 文件中，我们将性别、生日、技能等数据项分别用模板来单独写，再用< xsl:apply-template >来调用，这样，即使你日后要对这些模板作相应的修改与扩充也很方便，不致于出现互相干扰、混杂不清的情况。这种从上至下、逐层细化的设计方法，极大地减少工作复杂程度，也大大减少了差错的产生，可以实现多人的协作设计。

注意：

如果 XML 文档中不同标记有同名的子标记，在为其编写模板时，应把父标记作为其前缀，格式为（parent_mark/child_mark）。

模板文件必须有一个根模板，其属性 match 是“/”。

跟我学 XSL（四）

测试模式

XML 技术的优势之一就在于数据输出的可选择性，即选择需要的数据输出。前面我们所讲到的选择模式语句:<xsl:for-each>、<xsl:value-of>及<xsl:apply-template>只是简单的选取通过”/”符号层层到达的节点，如果我们对 XML 数据不需要全部输出，而只需要其中的满足某条件的部分数据，“萝卜青菜、各取所需“，那么条件判断<xsl:if>与多条件判断<xsl:choose>及<xsl:when>则迎合了这种需要，如果你对程序设计熟悉的话，会觉得它们似曾相识。

XSL 中的 IF，首先，介绍 XSL 元素<xsl:if>的语法结构：

<xsl:if>

语法：

<xsl:if expr="script-expression" language="language-name" test="pattern">

属性：

expr ——脚本语言表达式，计算结果为“真”或“假”；如果结果为“真”，且通过 test，则在输出中显示其中内容（可省略此项属性）。

language ——expr 属性中表达式的脚本语言类型，其取值与 HTML 标记 SCRIPT 的 LANGUAGE 属性的取值相同，缺省为“JScript”test ——源数据测试条件。

示例：

此处以一份报表为例，文件名为 report.xml，其内容如下：

<?xml version="1.0" encoding="GB2312"?>

<?xml:stylesheet type="text/xsl" href="report.xsl"?>

<document>

<report>

<class>

甲班

</class><q1>50</q1><q2>70</q2>

<q3>30</q3><q4>10</q4></report>

<report><class>

乙班

</class><q1>20</q1><q2>30</q2>

<q3>40</q3><q4>50</q4></report>

<report><class>

丙班

</class><q1>70</q1><q2>40</q2>

<q3>20</q3><q4>10</q4></report>

</document>

我们采用 XSL 模板结合今天所学的<xsl:if>，为其编写一个 XSL 文档，要求季度产量小于等于 20 的用红色表示，文件名为 report.xsl，内容如下：

<?xml version="1.0" encoding="GB2312"?>

<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">

<xsl:template match="/">

<HTML><HEAD><TITLE>1999 年生产统计</TITLE></HEAD>

<BODY><xsl:apply-templates select="document ;"/></BODY>

</HTML>

</xsl:template>

```
<xsl:template match="document">

<H3>1999 年生产统计</H3>

<TABLE    border="1" cellspacing="0">

<TH>班组</TH><TH>

一季度

</TH><TH>

二季度

</TH>


<TH>

三季度

</TH><TH>

四季度

</TH>


<xsl:apply-templates select="report"/>


</TABLE>

</xsl:template>

<xsl:template match="report">

<TR>

<TD><xsl:value-of select="class"/></TD>

<TD><xsl:apply-templates select="q1"/></TD>
```

```
<TD><xsl:apply-templates select="q2"/></TD>

<TD><xsl:apply-templates select="q3"/></TD>

<TD><xsl:apply-templates select="q4"/></TD>

</TR>

</xsl:template>

<xsl:template match="q1|q2|q3|q4">

<!--此处测试产量，如小于等于 20 则添加一 STYLE 属性 color，其值为 red（红色）-->


<xsl:if test=".[value() $le$ 20]">


<xsl:attribute name="style">color:red</xsl:attribute>

</xsl:if>

<xsl:value-of/>

</xsl:template>

</xsl:stylesheet>
```

说明：

q1|q2|q3|q4 ——标记 q1、q2、q3、q3 均用此模板确定输出

le ——是关系运算符中的“小于等于”，其它关系有小于（lt）、大于（gt）、大于等于（ge）、等于（eq）、不等于（ne）等

. ——表示引用当前标记

[ ]——表示筛选，只有满足筛选条件的标记才能被选取

value()——XSL 函数，其他常用 XSL 函数有 text()、end()、index()等。

用 XSLT 轻松实现树形折叠导航栏

一般我们见到的 XML 文件是以元素为结点的，随着层次的不断加深，逐渐成了一棵树，这种文件的好处是我们一看就很明白其中的子、父、祖宗、兄弟关系，不方便之处在于，我个人认为，如果层次很深又有很多的兄弟结点的话，那么文件可能很大而影响处理的效率。正由于 XML 对描述数据结构的灵活性，所以在某些环境下采用属性值来描述元素之间的关系。例如花园左边的 TOC（TABLE OF CONTENT），实现它的 XML 文件通过属性值来说明元素的类型（NODE OR LEAF），不过里面仍有子结点存在，所以用来 TRANSFORM 它的 XSL 文件很复杂，分了好几种情况。当然今天我们不是谈花园 TOC 的实现方法而是用一种更快速、更巧妙的方法来实现类似的 TOC，当然也可以叫"树形折叠导航栏"。 好了，废话少说，进入正题。先来看一个很简单的 DTD。 NAVIGATOR.DTD

```
<!ELEMENT Navigation (Navigator*)>
<!ELEMENT Navigator EMPTY>
<!ATTLIST Navigator ID CDATA #IMPLIED AncestorID CDATA #IMPLIED Layer CDATA #IMPLIED Title CDATA #IMPLIED Childs CDATA #IMPLIED Url CDATA #IMPLIED Image CDATA #IMPLIED>
```

文件很简单，可以这样理解，顶层元素 Navigation 包含了多个 Navigator 元素定义了，Navigator 不包含元素但有一系列属性。 也许您已经发现， 属性中有两个叫 AncestorID Childs 的，对了，这两个属性是关键， 当然还有 Layer， 在他们的共同作用下， Navigator 元素之间的关系将被明确描述。 好了， 我们来看 Navi.xml 文件， 以花园 TOC 做为例子。

查看花园 TOC 例子:

NAVI.xml

```
<?xml version="1.0" encoding="gb2312"?>
<!DOCTYPE Navi SYSTEM "navigator.dtd">
<?xml-stylesheet type="text/xsl" href="navigator.xsl" ?>
<Navigation>
<Navigator ID='1' AncestorID='1' Layer='0' Title='花园首页' Childs='0' Url='default.asp' Image='images/dc.gif'/>
<Navigator ID='2' AncestorID='2' Layer='0' Title='我的花园' Childs='4' Url='#' Image='default'/>
<Navigator ID='3' AncestorID='2' Layer='1' Title='收藏夹' Childs='4' Url='#' Image='default'/>
<Navigator ID='21' AncestorID='3' Layer='2' Title='我管理的花坛' Childs='0' Url='mybbs.asp?cat=g' Image='images/dc-new.gif'/>
<Navigator ID='22' AncestorID='3' Layer='2' Title='我种下的种子' Childs='0' Url='mybbs.asp?cat=t' Image='images/dc-new.gif'/>
<Navigator ID='23' AncestorID='3' Layer='2' Title='我喜欢的花园' Childs='0' Url='myfavorite.asp?cat=g' Image='images/dc-new.gif'/>
<Navigator ID='24' AncestorID='3' Layer='2' Title='我收藏的文章' Childs='0' Url='myfavorite.asp?cat=t' Image='images/dc-new.gif'/>
<Navigator ID='4' AncestorID='2' Layer='1' Title='个人工具箱' Childs='2' Url='#' Image='default'/>
<Navigator ID='25' AncestorID='4' Layer='2' Title='配置和管理' Childs='0' Url='personal.asp' Image='images/dc-config.gif'/>
<Navigator ID='26' AncestorID='4' Layer='2' Title='花瓣兑换点' Childs='0' Url='apetal.asp' Image='images/dc-config.gif'/>
<Navigator ID='27' AncestorID='2' Layer='1' Title='我的日记本' Childs='0' Url='mydiary.asp' Image='images/dc-diary.gif'/>
```

<Navigator ID='6' AncestorID='2' Layer='1' Title='好友和短讯' Childs='0' Url='myfriend.asp' Image='images/dc-friends.gif'/>
<Navigator ID='7' AncestorID='7' Layer='0' Title='计算机技术' Childs='2' Url='#' Image='default'/>
<Navigator ID='8' AncestorID='7' Layer='1' Title='DHTML,JScript' Childs='0' Url='bbsgroup.asp?c=6&g=16' Image='images/dc.gif'/>
<Navigator ID='9' AncestorID='7' Layer='1' Title='.NET,ASP+ 探讨' Childs='0' Url='bbsgroup.asp?c=6&g=17' Image='images/dc.gif'/>
<Navigator ID='10' AncestorID='7' Layer='1' Title='ASP 互助' Childs='0' Url='bbsgroup.asp?c=6&g=18' Image='images/dc.gif'/>
<Navigator ID='11' AncestorID='11' Layer='0' Title='箐箐校园' Childs='2' Url='#' Image='default'/>
<Navigator ID='12' AncestorID='11' Layer='1' Title='南京大学' Childs='0' Url='bbsgroup.asp?c=7&g=19' Image='images/dc.gif'/>
<Navigator ID='13' AncestorID='11' Layer='1' Title='东南大学' Childs='0' Url='bbsgroup.asp?c=7&g=20' Image='images/dc.gif'/>
<Navigator ID='14' AncestorID='14' Layer='0' Title='花园·有个广场' Childs='2' Url='#' Image='default'/>
<Navigator ID='15' AncestorID='14' Layer='1' Title='意见箱' Childs='0' Url='bbsgroup.asp?c=8&g=21' Image='images/dc.gif'/>
<Navigator ID='16' AncestorID='14' Layer='1' Title='花园·人物故事' Childs='0' Url='bbsgroup.asp?c=8&g=22' Image='images/dc.gif'/>
<Navigator ID='17' AncestorID='17' Layer='0' Title='园丁办公室' Childs='0' Url='bbsgroup.asp?c=9&g=23' Image='images/dc-key.gif'/>
<Navigator ID='18' AncestorID='18' Layer='0' Title='青青芳草地' Childs='0' Url='bbsgroup.asp?c=9&g=24' Image='images/dc.gif'/>
<Navigator ID='19' AncestorID='19' Layer='0' Title='统计信息' Childs='0' Url='viewlog.asp' Image='images/dc-chart.gif'/>
<Navigator ID='20' AncestorID='20' Layer='0' Title='ActiveCard' Childs='0' Url='activecard?fromgarden' Image='images/dc-card.gif'/>
</Navigation>
结合上面我讲的和花园左边的 TOC， 仔细分析这个文件后， 找出元素间存在的关系是很容易的， 难的是怎么想到这么来创建 XML 文件的。 好了， 有了数据， 下一步就是如何 MANUPILATE 了。

用 XSLT 轻松实现树形折叠导航栏

我引用花园的 TOC，一是让大家能有个初步印象，等文章完成后， 把几个文件 C&P 加上几个图片， 在 IE5 以上的机器上象打开一个 html 文件一样打开 navi.xml 后，就会出现跟花园很类似的 TOC 了;二是希望大家根据它的层次结构来分析我的 xml 文件，因为除顶层外，我的层次安排和花园是一样的。 我来解释一下：Layer 相同表示元素处在同一层次即兄弟关系， Childs 的值表示该元素是否有子结点， 父子之间用 AncestorID 和 ID 联系， 依次类推可以扩充至无限次深。 在 xsl 文件中根据 Layer 的值用 padding-left 属性来实现树形，根据 Layer 的值用 display：none 或 block 来实现折叠。 原理即此， 好，来看看这个关键的 Navigator.xsl: <?xml version="1.0" encoding="gb2312" ?>
<HTML>
<HEAD>
<TITLE>XSLT 树形导航栏</TITLE>
<LINK rel="stylesheet" type="text/css" href="navigator.css"/>
<SCRIPT src="toggle.js"></SCRIPT>
</HEAD>
<BODY>

```
<DIV xmlns:xsl="http://www.w3.org/TR/WD-xsl" >
<TABLE>
<TR>
<TD><DIV noWrap="true" STYLE="padding-left:0em;">有座花园分类目录</DIV></TD>
</TR>
<xsl:for-each select="Navigation/Navigator">
<TR>
<xsl:attribute name="TITLE"><xsl:value-of select="@Title" /></xsl:attribute>
<xsl:attribute name="Class">Navigator<xsl:if test="@Layer[.>0]">-Hidden</xsl:if></xsl:attribute>
<xsl:attribute name="ID"><xsl:value-of select="@ID"/></xsl:attribute>
<xsl:attribute name="AncestorID"><xsl:value-of select="@AncestorID"/></xsl:attribute>
<xsl:attribute name="Depth"><xsl:value-of select="@Layer"/></xsl:attribute>
<xsl:if test="@Childs[.>0]">
<xsl:attribute name="Expanded">no</xsl:attribute>
</xsl:if>
<TD STYLE="cursor:hand">
<DIV noWrap="true"><xsl:attribute name="STYLE">padding-left:<xsl:value-of select="@Layer"/>em;</xsl:attribute>
<xsl:choose>
<xsl:when test="@Childs[.>0]"><IMG src="images/bs.gif"></IMG></xsl:when>
<xsl:otherwise><IMG><xsl:attribute name="src"><xsl:value-of select="@Image" /></xsl:attribute></IMG></xsl:otherwise>
</xsl:choose>
<A><xsl:if test="@Childs[.>0]"><xsl:attribute name="onclick">toggle('<xsl:value-of select="@ID" />')</xsl:attribute></xsl:if><xsl:attribute name="href"><xsl:value-of select="@Url" /></xsl:attribute><xsl:value-of select="@Title" /></A></DIV></TD>
</TR>
</xsl:for-each>
</TABLE>
</DIV>
</BODY>
</HTML>
```

用 XSLT 轻松实现树形折叠导航栏

当然， 少了 navigator.css 是不行的。

navigator.css

```
BODY
{
font-family:Verdana;
cursor:default;
font-size:9pt;
}
TABLE
{
font-size:110%;
}
A
{
color:"#003366";
text-decoration:none;
}
A:hover
{
```

```
text-decoration:underline;
color:#003366;
}
DIV IMG
{
MARGIN-BOTTOM: 0px;
MARGIN-RIGHT: 5px;
MARGIN-TOP: -3px;
VERTICAL-ALIGN: middle
}
DIV A:hover
{
BACKGROUND-COLOR: greenyellow
}
DIV A
{
FONT-WEIGHT: normal;
MARGIN-RIGHT: 5px;
VERTICAL-ALIGN: middle
}
.Navigator
{
color: #003366;
}
.Navigator-Hidden
{
display:none;
}
```

现在运行 Navi.xml 的话， 您就会看见所有 Layer=0 的 Navigator 了（因为那些 Layer>0 的被 Class="Navigator-Hidden"隐藏起来了）， 当然现在还不能实现展开和折叠， 缺少 toggle 这个函数。

用 XSLT 轻松实现树形折叠导航栏

展开和折叠其实就是显示或不显示（display：none or block）它与可见与不可见（visible or invisible）是有区别的， 前者不在页面预留空间。 这个 toggle 函数完成两个功能， 改变 TR 原来的 Hidden 属性， 使原来不显示的显示； 改变 IMG 的 src 属性， 更改图片。

toggle.js

```
function toggle(id)
{
var thisRow = document.all.item(id);
if (thisRow)
{
if (thisRow.getAttribute("Expanded") == 'yes')
{
thisRow.setAttribute("Expanded", "no");
thisRow.children(0).children(0).children(0).src = "images/bs.gif";
var allRows = document.all.tags("TR");
for (var i=1; i < allRows.length; i++)
{
var row = allRows[i];
if (row.getAttribute("AncestorID") == id)
{
if (row.getAttribute("Expanded") == 'yes') {
```

```
toggle(row.getAttribute("id"));
}
row.className = 'Navigator-Hidden';
}
}
thisRow.className = 'Navigator';
}
else
{
thisRow.setAttribute("Expanded", "yes");
thisRow.children(0).children(0).children(0).src = "images/bo.gif";
var allRows = document.all.tags("TR");
var depth = parseInt(thisRow.getAttribute("Depth"));
for (var i=1; i < allRows.length; i++)
{
var row = allRows[i];
if (row.getAttribute("AncestorID") == id &&
parseInt(row.getAttribute("Depth")) == depth + 1 )
{
row.className = 'Navigator';
}
}
}
}
}
```

到此结束。

诚然这个 TOC 的功能还是最基本的， 例如我还未做内容和目录的同步，其中有的地方还可以修改， 对 xml 和 xsl 文件可以进一步瘦身。 不过对一般用户来讲， 这已经足够了。

真诚希望这篇文章能对您有所启发、有所帮助， 以后做出更酷、更快、更方便、功能更强的 TOC。

用 XSL 显示 XML

你可以用 XSL 向 XML 文档中增加显示信息。

用 XSL 显示 XML

XSL 是 XML 首选的格式表语言。 XSL (可扩展的格式表语言) 比 CSS 要复杂得多。使用 XSL 的一种方法是在它被浏览器显示之前，将 XML 转换成 HTML，就象下面例子：

点击这里可以查看原始 XML 文件。

点击这里查看用 XSL 格式表进行格式化的同一个文件。

点击这里查看 XSL 格式表。

下面显示的是这个文件的缩写版本。注意在第二行的 XSL 引用：

```
< ?xml version="1.0"?>
< ?xml:stylesheet type="text/xsl" href="simple.xsl" ?>
< breakfast-menu>
< food>
< name>Belgian Waffles< /name>
```

< price>$5.95< /price>
< description>
two of our famous Belgian Waffles
< /description>
< calories>650< /calories>
< /food>
< /breakfast-menu>

有关 XSL 的更多信息，可以访问 W3Schools' XSL School。

在数据岛中的 XML

用 Iternet Explorer 5.0 可以将 XML 放在数据岛中嵌入 HTML 页面内。

将 XML 嵌入 HTML

用非正式的< xml>标记将 XML 数据嵌入 HTML 中。可以直接将 XML 数据嵌入一个 HTML 页面，象这样:

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

或者嵌入一个单独的 XML 文件：

```
< xml id="note" src="note.xml">
< /xml>
```

注意< xml>标记是一个 HTML 元素，而不是 XML 元素。

数据捆绑

可以将数据岛捆绑到 HTML 元素中（如 HTML 表格）。在下面的例子中，一个 ID 为 "cdcat" 的 XML 数据岛从一个外部 XML 文件装载近来。用一个数据源属性将数据岛捆绑到一个 HTML 表格，最后用在一个范围内的数据域属性将 XML 数据捆绑到表格数据元素。

```
< html>
< body>


< xml id="cdcat" src="cd_catalog.xml">< /xml>


< table border="1" datasrc="#cdcat">
< tr>
< td>< span datafld="artist">< /span>< /td>
< td>< span datafld="title">< /span>< /td>
< /tr>
< /table>


< /body>
< /html>
```

如果你正在运行 IE 5,你就可以自己试一试。你还可以用 IE 5.0 查看外部 XML 文件。也可以使用这个例子，示范< thead>、 < tbody>和 < tfoot>。