第四章 XML技术

苗启广
计算机学院
2012.09

qgmiao@mail.xidian.edu.cn

主要内容

- XML的产生与XML的特征
- XML文档的语法
- XML文档类型定义(DTD)
- XML命名空间（Name Space）
- XML模式（Schema）
- XML应用程序接口（DOM&SAX）
- XML文档的显示（XSL）
- XML的应用

4.1 XML的产生

4.1.1 置标语言

◎ 什么是置标语言

- 描述什么是有效标记的标准
- 描述标记具体含义的标准

◎ HTML

HTML的缺点

- **HTML**不支持严格的数据合法性检查
- 它不允许用户定义私有的标记字或属性
- **HTML**不支持结构化数据，不支持基于数据库模型或面向对象层次的数据结构

4.1.2 XML（eXtensible Markup Language）

◎什么是XML

XML含义：可扩展的置标语言，元置标语言。依赖于描述一定规则的标签和能够读懂这些标签的应用处理工具来发挥它的强大功能。

- XML的制定目标：

1. XML应该支持各种不同的应用方式（不但包括浏览还包括对内容的分析）
2. XML应该可以在互联网上直接使用（就象HTML那样好用）
3. XML应该与SGML兼容（子承父业，SGML是XML的直接先驱）

SGML (Standard Generalized Markup Language)
标准通用标记语言

◎ HTML | 2.0

Overview of SGML Resources

This is an overview of resources related to SGML and the Web, originally assembled for reviewers of the HTML 2.0 specification.

See also: Extensible Markup Language (XML).

Learning and Using SGML

SGML, Standard Generalized Markup Language, is an enabling technology used in applications such as HTML. (see: Toward a Formalism for Communication on the Web).

The HTML specifications assume a working knowledge of SGML. The SGML standard itself is not available online, but there's plenty of code to read if you're a hacker, and a few introductory documents for everybody.

SGML (Standard Generalized Markup Language)
标准通用标记语言

◎ HTML | 2.0

Gentle Introduc  to SGML

from the TEI guidelines. A must-read for folks new to SGML

SGML Bibliography

Robin Cover's extensive (700 item) bibliography on SGML

SGML archive at the University of Oslo, Norway.

by Erik Naggum

update 1996-03-01: "The SGML Repository is no longer actively maintained, and will remain as is for the foreseeable future."

SGML (Standard Generalized Markup Language)
标准通用标记语言

◎ SGML：是基于一种IBM早先创建出的通用标记语言。

SGML语言程序由三部分组成：

- 语法定义：定义了文件类型定义和文件实例的语法结构
- 文件类型定义(Definition Type Document，DTD)：定义了文件实例的结构和组成结构的元素类型
- 文件实例：是SGML语言程序的主体部分

SGML的实际使用中，每一个特定的DTD都定义了一类文件。例如，所有的新闻稿件都可以使用同一个DTD。因此，人们习惯上把具有某一特定DTD的SGML语言，称为某某置标语言。例如用于国际互联网的HTML语言。这样SGML就成为那些派生语言的元语言。

SGML (Standard Generalized Markup Language)
标准通用标记语言

◎ SGML是一种指示文档标准语言或标记集的标准。

这种说明本身就是一种文档标准定义。SGML本身并不是一个文档语言，但它描述了一种文档语言，因此它是一种元语言；

SGML的思想是基于文档应该有结构和语义结构而设计的。它不关心这些语义元素如何显示，而只关心如何组织它。因此显示的方式可能有所不同。

基于SGML的文档有以下优点：

- 基于文档内部结构而非显示而设计；因此不用反复修改；
- 可移植性好，只要SGML解释器存在，就可对文档进行解释；
- 原来用于打印的一些文档可以轻易地被用于显示；

4.1.2 XML（eXtensible Markup Language）

◎ XML的制定目标：

1. XML应该支持各种不同的应用方式（包括浏览和对内容的分析）
2. XML应该可以在互联网上直接使用（就象HTML那样好用）
3. XML应该与SGML兼容（子承父业，SGML是XML的直接先驱）
4. 处理XML文件的应用程序应该容易编写
5. XML中的可选特性的数量应该减到最小，最好没有（可选特性造成混淆）
6. XML文件应该具有良好的可读性，并且比较清晰
7. 用XML设计新的置标语言应该方便快捷，而且正式、简洁
8. XML文件应该容易编制

4.1.2 XML（eXtensible Markup Language）

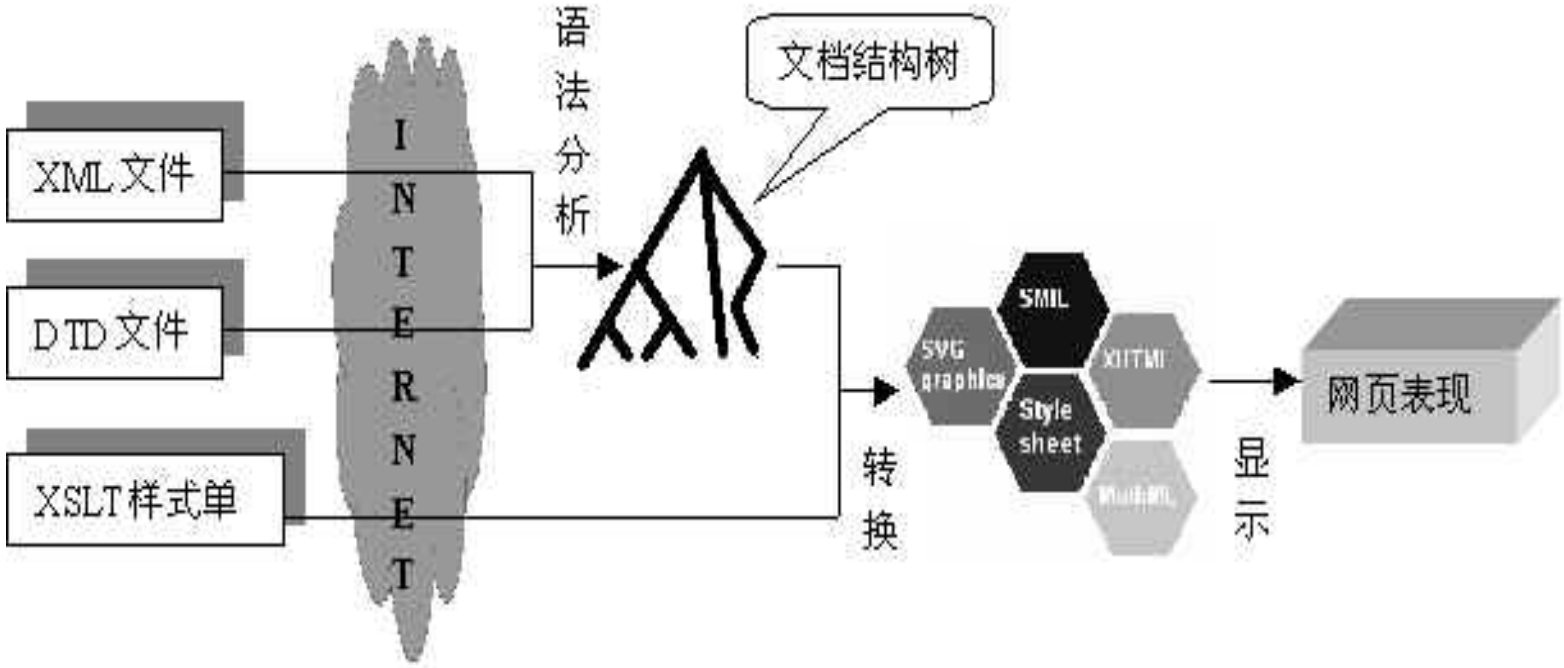
◎ XML的产生历程

- 直接先驱HTML、SGML
- 1996年国际互联网联盟W3C决定成立专门小组研究XML；
- 1997年春天，XML草案拟定；
- 1997年夏，XML第一个真正应用CDF

  (Channel Definition Format,频道定义格式)；
- 1998年W3C于2月批准XML1.0版本。XML修成正果。

◎ XML的基本构成

- XML文档
- 文档类型定义DTD——描述什么是有效标签，定义置标语言结构。
- 样式——向应用程序指示操作（向浏览器指示如何处理显示）。



◎ XML的优点

- 自由民主网上世界——根据自己需要制定标记
- 超越格式之上——标记不必局限于显示格式的描述
- 遵循严格语法要求--配对、嵌套、严格遵守DTD规定
- 便于不同系统传输—跨OS，多数据库
- 良好的保值性——长期通用标准、向其它格式转化

◎ XML与HTML的比较

比较内容	HTML	XML
可扩展性	不具有扩展性	是元置标语言，可用于定义新的置标语言
侧重点	侧重于如何表现信息	侧重于如何结构化地描述信息
语法要求	不要求标记的嵌套、配对等，不要求标记之间具有一定的顺序	严格要求嵌套、配对，和遵循DTD的树形结构
可读性及可维护性	难于阅读、维护	结构清晰，便于阅读、维护
数据和显示的关系	内容描述与显示方式整合为一体	内容描述与显示方式相分离
保值性	不具有保值性	具有保值性
编辑及浏览	已有大量的编辑、浏览工具	编辑、浏览工具尚不成熟

◎ XM

<dl>标记定义了一个定义列表，定义列表中的条目是通过使用<dt>标记(定义标题)和<dd>标记(定义描述)创建的。

<dt>给出了术语名，<dd>标记给出了术语的定义。

HTML文档举例

```
<dt>Hot C
<dd
Bel
<ul
<li
<li
Reco
<li>Length: 6
<li>Written: 1978
<li>Artist: Village People
</ul>
```

相对应的XML文档

```
<PUBLISHER>Polydram Records</PUBLISHER>
<LENGTH>6:20</LENGTH>
<YEAR> 1978</YEAR>
<ARTIST>Village People</ARTIST>
</SONG>
```

代码

```
<dl>
<dt>Coffee</dt>
 <dd>Black hot drink</dd>
<dt>Milk</dt>
<dd>White cold drink</dd>
</dl>
```

输出结果

Coffee
Black hot drink
Milk
White cold drink

4.2 XML语法

- 什么是形式良好的XML文件

XML并没有定义一个特定的标记集，而是描述了一个用来定义标记集的方法，当我们用这个方法规定好一个标记集，并根据这些规定填入文本内容后，这些标记就和纯文本一起构成一个XML文件。

XML中，“形式良好”有着明确的标准，即是要遵守XMLx.x规范中的语法规则。无论是从物理结构上还是从逻辑结构上讲，XML必须符合规范，才能被正确解释处理。

一个格式正规的XML文档由

序言(prolog)、文档的主体(bo

包括XML声明、文档类型声明、处理指令等

序言

```
<?xml version="1.0" encoding="GB2312"?>
<?xml-stylesheet href="style.xsl" type="text/xsl"?>
<!-- 以上是XML文档的序言部分 -->
```

文档根元素包含的部分

主体

```
<COLLEGE>
    <TITLE>计算机学院</TITLE>
    <LEADER>王东</LEADER>
      <STU_NUMBER unit="人">3</STU_NUMBER>

      <STUDENT>
      <NAME>李文</NAME>
      <AGE>21</AGE>
      <SEX>男</SEX>
      <CLASS>9902</CLASS>
      </STUDENT>
</COLLEGE>
```

注释、处理指令或者空白

尾声

```
<!-- 以上是文档的主体部分，以下是文档的尾声部分 -->
```

XML的语法包括下列内容：

- ◎ 声明
- ◎ 元素
- ◎ 注释
- ◎ 内嵌的替代符
- ◎ 处理指令
- ◎ CDATA

4.2.1 XML声明——用XML声明作为开头

一个完整的XML声明：

```
<?xml version  = "1.0"
standalone     = "no"
encoding       = "GB2312"?>
```

- version属性。在XML的处理指示中必须包括version属性指明所采用的XML的版本号，必须在属性列表中排在第一位。

- standalone属性。表明该XML文件是否和一个独立的置标声明文件配套使用。如果该属性置为“yes”，说明没有另外一个配套的DTD文件来进行置标声明。相反如果这个属性置为“no”，则有可能有这样一个文件。

- encoding属性。所有的XML语法分析器都要支持8位和16位的编码标准。只要知道下面几个常见的编码就可以了：简体中文码：GB2312、繁体中文码：BIG5、西欧字符： UTF-8。哪种编码取决于你文件中用到的字符集。

4.2.2 XML元素 – XML文件的精髓

- XML元素

元素是XML文件内容的基本单元。从语法上讲，一个元素包含一个起始标记、一个结束标记以及标记之间的数据内容。

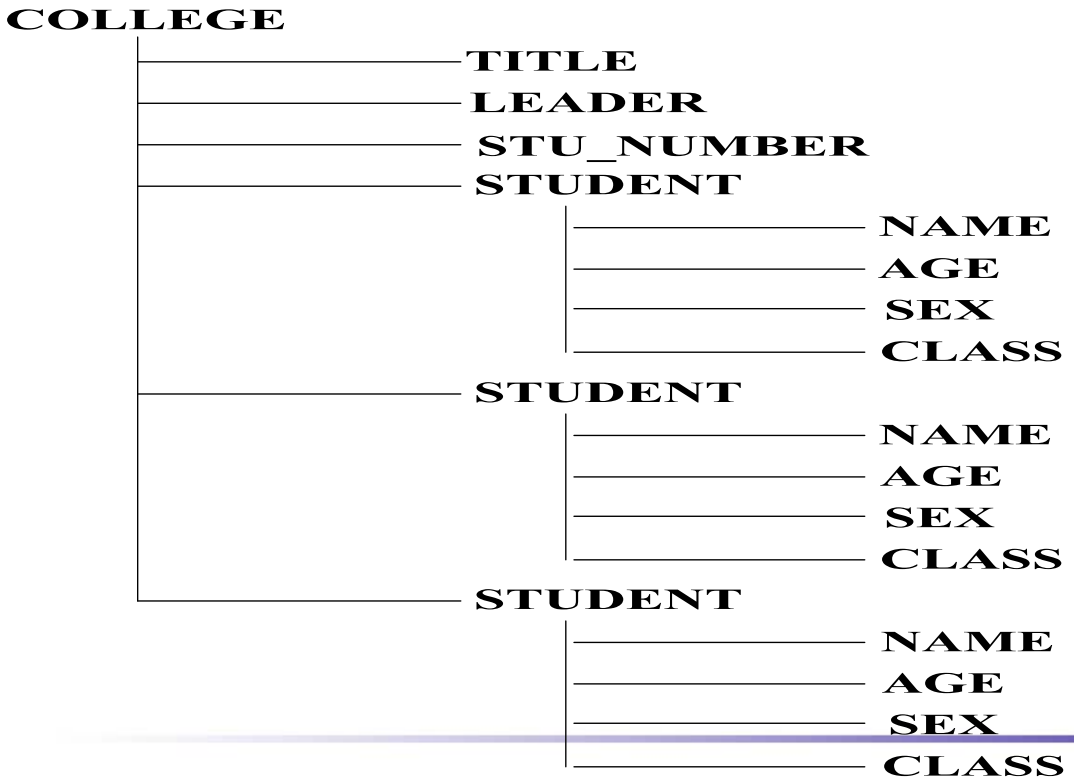
其形式是：

元素

```
<elementNAME (attrName="")>   content   </elementName>
起始标记                      属性      字符数据      结束标记
```

元素可以包含其他元素、字符数据、实体引用、处理指令、注释和CDATA部分。这些统称为元素内容（element content）。

位于文档最顶层的一个元素包含了文档中其它所有元素，称为根元素。另外，元素中还可以再嵌套别的元素。元素之间应正确的嵌套，不能互相交叉。所有元素构成一个简单的层次树，元素和元素之间唯一的直接关系就是父子关系。



■ 标记书写规则

“置标”是XML语言的精髓。因此，标记在XML的元素中、乃至整个XML文件中，占了举足轻重的位置。XML的标记和HTML的标记在模样上大体相同，除了注释和CDATA部分以外，所有符号“<”和符号“>”之间的内容都称为标记。其基本形式为：

<标记名 （属性名=″属性值″）*>

XML对于标记的语法规定比HTML要严格得多

■ 标记书写规则

（1）标记命名要合法

- ◆ 以字母、下划线、冒号开头
- ◆ 字母、数字、句号、冒号、下划线、连字符
- ◆ 中间不能有空格

（2）区分大小写

（3）必须有正确的结束标记

</elementName> 大小写一致，加上反斜线

（4）标记间要正确嵌套

■ 有效使用属性

标记中可以包含任意多个属性。在标记中，属性以“名称/取值”对的形式出现，属性名不能重复，名称与取值之间用等号“=”分隔，且取值用引号引起来。例如：

〈commodity type =″服装″ color=″黄色″〉

- type和color是commodity标记的属性
- “服装”是type属性的取值
- “黄色”是color属性的取值

■ 有效使用属性

空格属性和语言属性是XML系统提供的两个特殊属性，使用它们可以说明具体XML元素中的空格和语言特性。

- ●空格属性xml:space
  - ◆ 用来说明是否需保留XML元素数据内容的空格符
  - ◆ default：可读性，书写时有；自动去除
  - ◆ preserve：保留XML元素中的所有空格和回车
- ●语言属性xml:lang
  - ◆ 用来说明XML元素使用何种语言
  - ◆ ISO639国际标准中的编码：en, fr
  - ◆ 用户自定义的语言编码：“X-”和“x-”

4.2.3 注释

同HTM：用“〈！--”和“--〉”引起来。

下面是一个合法的XML文件：

```
〈示例〉
  〈!-- 一个XML的例子 --〉
  〈![CDATA[
      〈联系人〉
       〈姓名〉张三〈/姓名〉
       〈EMAIL〉zhang@aaa.com〈/EMAIL〉
      〈/联系人〉
   ]]〉
〈/示例〉
```

4.2.4 内嵌的替代符

字符<、>、&、′和“是XML的保留字符，XML利用它们定义和说明元素、标记或属性等。XML的解析器也将这些字符视为特殊字符，并利用它们来解释XML文档的层次内容结构。

当XML内容中包含这些字符，且需要显示时，就可能会带来混乱和错误。XML使用内嵌的替代符来表示这些系统保留字符来解决这个问题。

替代符	含义	例子	解析结果
<	<	3<5	3<5
>	>	5>3	5>3
&	&	A&B	A&B
'	′	Joe's	Joe′s
"	"	"yes"	"yes"

4.2.5 处理指示

处理指示是用来给处理XML文件的应用程序提供信息的。也就是说，XML分析器只是把这些信息原封不动地传给XML应用程序。然后，这个应用程序来解释这个指示，遵照它所提供的信息进行处理，或者再把它原封不动地传给下一个应用程序。

所有的处理指示应该遵循下面的格式：

〈？处理指示名 处理指示信息？〉

4.2.5 处理指示

例如：使用一个处理指示来指定与这个XML文件配套使用的样式单的类型及文件名：

```
<?xml-stylesheet type="text
```

处理指令为XML开发人员提供了一种跨越各种元素层次结构的指令表达方式，从而使得应用程序能够按照指令所代表的意义来处理文档。

再如：

```
<article>
    <title><?beginUseBold?>节约能源
    </title>
    <content>能源危机<?endUseBold?>早已经不是陌生的话题
    </content>
</article>
```

4.2.6 CDATA

在标记CDATA下，所有的标记、实体引用都被忽略，而被XML处理程序**一视同仁地当作字符数据**看待。CDATA的基本语法如下：

<![CDATA[*文本内容*]]>

很显然CDATA的文本内容中是不能出现字符串“]]>”的，因为它代表了CDATA数据块的结束标志。

```
<Adress>
        <![CDATA[
          <联系人>
          <姓名>Jack</姓名>
          <EMAIL>Jack@edu.cn</EMAIL>
          </联系人>
        ]]>
</Adress>
```

```
<?xml version="1.0" encoding="GB2312"?>
<?xml-stylesheet href="style.xsl" type="text/xsl"?>
<!--  以上是XML文档的序言部分  -->
<COLLEGE>
    <TITLE>计算机学院</TITLE>
    <LEADER>王志东</LEADER>
    <STU_NUMBER unit="人">3</STU_NUMBER>
    <STUDENT>
        <NAME>李文</NAME>
        <AGE>21</AGE>
        <SEX>男</SEX>
        <CLASS>9902</CLASS>
    </STUDENT>
</COLLEGE>
<!--  以上是文档的主体部分 ， 以下是文档的尾声部分  -->
```

声明

处理指示

注释

字符数据

元素

元素

元素

注释，尾声

4.3 文档类型定义 DTD

- DTD（Document Type Definition）描述了一个置标语言的语法和词汇表，也就是定义了文件的整体结构及文件的语法。XML必须遵守DTD中定义的种种规定
- **规则：**
  - 简单：列出所有有效元素：元素、标记、属性、实体等；
  - 复杂：除元素外，元素之间内在关系：包含

4.3.1 DTD

XML所描述的置标语言中，DTD提供了语法规定，以给各个语言要素赋予一定的顺序。

为说明特定的语法规则，DTD采用了一系列正则式，语法分析器将这些正则式与XML文件内部的数据模式相匹配，从而判别一个文件是否是有效的。匹配被严格执行，因此，如果XML文件中有任何信息不符合DTD的规定，都不会通过。

4.3.1 DTD

通过创建DTD，能够正式而精确地定义词汇表。所有词汇表规则都包含在DTD中，凡是未在DTD中出现的规则都不属于词汇表的一部分。只需要在XML文档中写入一条声明语句，解析器就能够获取DTD，并利用DTD验证文档的有效性。同时，其他程序或用户也能够根据DTD中的规则在文档中添加合法的元素或属性。

◎ 将DTD与XML文档相关联

序言中包含了XML声明，而文档主体中则是具体的数据信息，还可以含有一些处理指示。**在XML文档序言中还可以包含DTD定义。**

1. DOCTYPE标记

为了将DTD声明与XML文档相关联，XML1.0标准提供了特殊的DOCTYPE声明。DOCTYPE声明必须位于XML声明之后，且在任何文档元素之前。XML声明和DOCTYPE声明之间可以插入注释和处理指令。

```
<?xml version = "1.0" encoding="GB2312" standalone = "yes"?>
<!--  DOCTYPE声明，其中可以包含文档类型定义 -->
<!DOCTYPE 根元素名[
  ]>
<根元素名>


</根元素名>
```

包含了当前XML文档所需的DTD，可以是具体的定义描述信息本身（内部DTD子集），也可以是指向真正包含定义描述信息的外部文件引用（外部DTD子集）

2. 内部DTD子集

内部DTD子集通常定义为：

<!DOCTYPE 文档根元素名[标记描述信息]>

文档根元素名，指的是所定义的文档类型定义信息所适用的XML文档的根元素名称。例如，下面的语句定义了XML程序文档的一个内部DTD子集。

```
<!DOCTYPE myMessage[
<!ELEMENT myMessage(#PCDATA)>
]>
```

◎ 内部DTD实例

```
<?xml version = "1.0" encoding="GB2312" standalone
= "yes"?>
<!DOCTYPE 联系人列表[
    <!ELEMENT 联系人列表 (联系人)*>
    <!ELEMENT 联系人 (姓名,ID,公司,EMAIL,电话,地址)>
    <!ELEMENT 地址 (街道,城市,省份)>
    <!ELEMENT 姓名 (#PCDATA)>
    <!ELEMENT ID (#PCDATA)>
    <!ELEMENT 公司 (#PCDATA)>
    <!ELEMENT EMAIL (#PCDATA)>
    <!ELEMENT 电话 (#PCDATA)>
    <!ELEMENT 街道 (#PCDATA)>
    <!ELEMENT 城市 (#PCDATA)>
    <!ELEMENT 省份 (#PCDATA)>
]>
```

◎ 内部 DTD实例

```
<?xml-stylesheet type="text/xsl" href="mystyle.xsl"?>
<联系人列表>
    <联系人>
        <姓名>张三</姓名>
        <ID>001</ID>
        <公司>A公司</公司>
        <EMAIL>zhang@aaa.com</EMAIL>
        <电话>(010)62345678</电话>
        <地址>
            <街道>五街1234号</街道>
            <城市>北京市</城市>
            <省份>北京</省份>
        </地址>
    </联系人>
</联系人列表>
```

3. 外部DTD

如果一批XML文件有一个相同的DTD，为每一个XML文件加入一段DTD定义是相当繁琐的。例如，对于报社中的每篇稿件，它们都有相同的格式，可以采用一个统一的DTD，为每一篇单独定义既麻烦，又不利于统一格式。这种情况下采用外部DTD。

外部DTD子集可以有以下两种定义方式：

3．外部DTD

（1）利用一个简单的统一资源标识符URI来指明外部DTD文件的位置。其一般形式为：

<!DOCTYPE 文档根元素名 SYSTEM "URI地址">

如，下面的语句定义了XML程序文档的一个外部DTD子集，它告诉XML解释器定义了文档类型的DTD文件的位置。

```
<!DOCTYPE Ticket SYSTEM "http://www.ilusion.org/XmlDemo/tichet.dtd">
```

SYSTEM是系统保留字，其后跟随说明外部DTD文件位置的URI。这个URI告诉XML解析器，包含文档类型定义的外部DTD文件在Web服务器http://www.ilusion.org的XmlDemo目录下，文件名为“tichet.dtd”。同时，DOCTYPE标记还通知XML解析器，该文档类型定义描述中的根元素名为Ticket。

（2）使用PUBLIC系统保留字，采用SGML通用的FPI（Formal Public Identifiers）来定义DTD文件的位置，语法格式：

```
<!DOCTYPE Ticket PUBLIC "-//illusion//XmlDemo//EN">
```

两种定义方式结合如：

```
<!DOCTYPE Ticket PUBLIC "-//illusion//XmlDemo//EN"
 "http://www.ilusion.org/XmlDemo/tichet.dtd">
```

这是目前非常常用的外部DTD声明方式，这时如果XML不具备解释FPI的能力，则可以根据URI方便地找到DTD文件。

外部 DTD实例

程序清单　fclml.xml

```
<?xml version = "1.0" encoding="GB2312" standalone = "no"?>
<!DOCTYPE 联系人列表 SYSTEM "fclml.dtd">
<?xml-stylesheet type="text/xsl" href="mystyle.xsl"?>

<联系人列表>
   <联系人>
      <姓名>张三</姓名>
      <ID>001</ID>
      <公司>A公司</公司>
      <EMAIL>zhang@aaa.com</EMAIL>
      <电话>(010)62345678</电话>
      <地址>
         <街道>五街1234号</街道>
         <城市>北京市</城市>
         <省份>北京</省份>
         <ZIP>100001</ZIP>
      </地址>
   </联系人>

   </联系人列表>
```

程序清单　fclml.dtd

```
<!ELEMENT 联系人列表 (联系人)*>
<!ELEMENT 联系人 (姓名,ID,公司,EMAIL,电话,地址)>
<!ELEMENT 地址 (街道,城市,省份)>
<!ELEMENT 姓名 (#PCDATA)>
<!ELEMENT ID (#PCDATA)>
<!ELEMENT 公司 (#PCDATA)>
<!ELEMENT EMAIL (#PCDATA)>
<!ELEMENT 电话 (#PCDATA)>
<!ELEMENT 街道 (#PCDATA)>
<!ELEMENT 城市 (#PCDATA)>
<!ELEMENT 省份 (#PCDATA)>
```

4. 内部DTD子集和外部DTD的混合使用

一个XML文档可以同时拥有内部DTD子集和外部DTD子集。例如：

```
<!DOCTYPE myMessage SYSTEM "myDTD.dtd"[
<!ELEMENT myMessage(#PCDATA)>
]>
```

或者使用更加通用的方式定义：

```
<!DOCTYPE 文档根元素名 PUBLIC "外部DTD信息FPI"
                        "外部DTD文件URI"
                        [内部标记描述信息]>
```

4.3.2 DTD基本标记声明

DTD通过四种标记声明定义XML文档中允许出现的内容。

DTD关键字	含义
ELEMENT	XML元素类型声明
ATTLIST	特定元素类型可设置的属性以及这些属性的允许值声明
ENTITY	可重用的内容声明
NOTATION	不需要解析的外部内容的格式声明，以及用于处理这些内容的外部应用程序

ELEMENT 与ATTLIST与XML文档中的信息（元素和属性）相关；

ENTITY和NOTATION起辅助作用。特别是实体（ENTITY）用于包含那些通常在DTD和XML文档中反复出现的部分，其作用类似于C++语言中的宏语句。NOTIATION用于处理非XML内容，例如声明特殊的数据类并将之与外部程序相关联。

4.3.3 实体

实体是XML提供的声明内容块的方法，这些内容块可以根据需要多次被引用，它能够节省空间，并减少文档的代码量。为了在DTD中声明实体，需要定义实体的名称和它引用的内容

实体分为预定义实体、通用实体和参数实体三种。

1. 预定义实体

<、>、&、'、和&quot是XML标准预定义的实体，分别用来指代特殊的字符

2. 通用实体

通用实体又可分为内部实体、外部实体，或者分为解析实体（parsed entity）和非解析实体（unparsed entity）

3. 参数实体

仅仅在DTD内部使用的解析实体称为参数实体，使用方法与通用实体类似，只是它只能使用在DTD定义中。一般用来引用DTD中常用的结构块，使得能够方便引用或修改这些结构块。只需维护这一处代码。

参数实体指代的是DTD信息块，而不是XML文档或数据

4.3.3 实体

通用实体

解析实体：符合XML文档规范的内容，一定被解析器解析并使用；

非解析实体：可以是其它格式的文本或二进制块，外部实体，可能忽略

内部实体：定义在使用该实体的XML文件中

外部实体：定义在使用该实体的XML文件外

分类：

内部解析实体，外部解析实体，外部非解析实体

1. 内部解析实体

内部解析实体可以在内部DTD子集中声明。声明实体需要使用ENTITY关键字。声明内部解析实体的一般语法格式为：

<!ENTITY 实体名 "实体内容">

程序清单

```
<?xml version="1.0" encoding="UTF-8"?>
  <!DOCTYPE NEWS[
<!ENTITY TeamA1 "中国队">
<!ENTITY TeamA2 "泰国队">
<!ENTITY ScoreA "1:0">
<!ENTITY TeamB1 "沙特队">
<!ENTITY TeamB2 "韩国队">
<!ENTITY ScoreB "0:0">
  ]>
<NEWS>新华社讯：在今天的两场亚洲杯足球赛中，&TeamA1;以&ScoreA;的比分战胜了
&TeamA2;，&TeamB1;与&TeamB2;以&ScoreB;握手言和。
</NEWS>
```

使用实体时，在实体名前加&,后加;

上面的XML文档中首先包含了一个内部DTD子集，其中定义了六个实体，并在随后用户定义的NEWS标记中使用了这些实体。如果一个XML解释器对NEWS标记中的字符数据加以解释，并且用DTD中的实体内容替代实体引用，则可以得到如下的结果：

新华社讯：在今天的两场亚洲杯足球赛中，中国队以1:0战胜了泰国队，沙特队与韩国队以0:0握手言和。

2. 外部解析实体

内部解析实体可以帮助XML文档实现字符串的替代，如果希望在XML文档中引用一个完整的文件，则需要使用外部解析实体。

当不同的XML文档之间需要共享大量的数据时，可考虑将这些数据保存成一个文件，而在XML程序中使用外部解析实体指代它。

程序清单

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE NEWS SYSTEM "NewsEntity.dtd">
<NEWS>新华社讯：在今天的两场亚洲杯足球赛中，&TeamA1;以&ScoreA;的比分战胜了&TeamA2;，&TeamB1;与&TeamB2;以&ScoreB;握手言和。
</NEWS>
```

2. 外部非解析实体

外部非解析实体可以用来在XML文档中加入二进制数据，如图像、声音等多媒体数据。

当不同的XML文档之间需要共享大量的数据时，可考虑将这些数据保存成一个文件，而在XML程序中使用外部解析实体指代它。

<!ENTITY 实体名 SYSTEM “URI” NDATA 标注名>

或

<!ENTITY 实体名 PUBLIC “FPI” NDATA 标注名>

如：

<!ENTITY MYPIC SYSTEM “http://mymschine:8080/mypic.gif” NDATA gif>

不是引用，而是以数值的方式赋给XML程序文档中的元素

<IMC PIC=“MYPIC”>

4.3.4 元素类型声明

DTD中的元素说明部分，使用元素类型声明（ETD）来声明所有有效的文件元素。

<!ELEMENT 元素名 元素内容描述>

```
<!DOCTYPE mymessage SYSTEM “intro.dtd”>

<mymessage>

     <message> Welcome to XML! </message>
</mymessage>
<!ELEMENT mymessage(message)>
<!ELEMENT message(#PCDATA)>
```

4.3.5 属性声明

用来定义元素可接受的属性

ATTLIST是一个属性的列表，它可以包含很多属性，在实际应用中，一个元素也经常有多个属性。

DTD中定义属性时，使用下面的格式：

```
<!ATTLIST 元素名 （属性名 属性类型 缺省值）*>
```

```
<!ATTLIST 商品
   类型 CDATA #REQUIRED
   颜色 CDATA #IMPLIED
>
```

属性类型

属性类型必须是系统定义的十种类型之一：

（1）CDATA：说明该属性的取值可以是任意的字符串，是限制最少、最自由的属性。

（2）ID　：ID类型说明该属性的取值可以是任意合法的标识符。

ID类型的属性是用来唯一标识元素的属性，所以一个元素只有一个ID类型的属性，而且ID属性的取值必须与XML文档中其他元素的ID类型属性的取值不同。

IDREF、IDREFS、ENTITY 、ENTITIES 、NMTOKEN 、NMTOKENS 、NOTATION 、枚举类型

```
<?xml version= "1.0 ">
<!DOCTYPE myMessage[
<!ELEMENT myMessage(message)>
<!ELEMENT message(#PCDATA)>
<!ATTLIST message id CDATA #REQUIRED>
]>
<myMessage>
  <message id= " 445">Welcome to XML!</message>
</myMessage>
```

4.3.6 条件部分

DTD可以包含条件部分，它用于向解析器说明包含或忽略声明部分。 条件部分使用关键字INCLUDE和IGNORE。

- INCLUDE：其中的声明被认为是DTD的一部分
- IGNORE： 处理器虽读取其中的声明，但在处理时忽略它

INCLUDE和IGNORE标记的使用非常简洁。只要把在复用DTD基础上需要特别加入的部分括入INCLUDE，需要特别去除的部分括入IGNORE即可。

使用INCLUDE和IGNORE 标记的语法为：

<![INCLUDE[需要加入的完整的DTD定义]]> 或

<![IGNORE[需要去除的完整的DTD定义]]>

程序清单

```
<!ELEMENT UNIT (MANUFECTURE,SERIAL,LOCATION,PURCHASE,DEVICE)>
<![%MONITOR[
        <!ELEMENT DEVICE(SCREENSIZE,COLOR)>
        <!ELEMENT SCREENSIZE(#PCDATA)>
        <!ELEMENT COLOR (color|back-with) >
]]>
<!%COMPUTER[
        <!ELEMENT DEVICE (PROCESSOR,RAM,HD)>
        <!ELEMENT PROCESSOR(#PCDATA)>
       <!ELEMENT RAM(#PCDATA)>
       <!ELEMENT HD(#PCDATA)>
]]>
<![%NETWORKCARD[
       <!ELEMENT DEVICE(MEDIA,MAC)>
       <!ELEMENT MEDIA(#PCDATA)>
       <!ELEMENT MAC(color|back-with)>
]]>
```

对应这个DTD的XML程序文档片段可以为：

```
<?xml version= "1.0"?>
<!DOCTYPE UNIT[
     <!ENTITY %MONTIOR "INCLUDE">
     <!ENTITY %COMPUTER "IGNORE">
     <!ENTITY %NETWORKCARE "IGNORE">
     < !ENTITY  %unit.dtd "unit.dtd">
%unit.dtd
]>
```

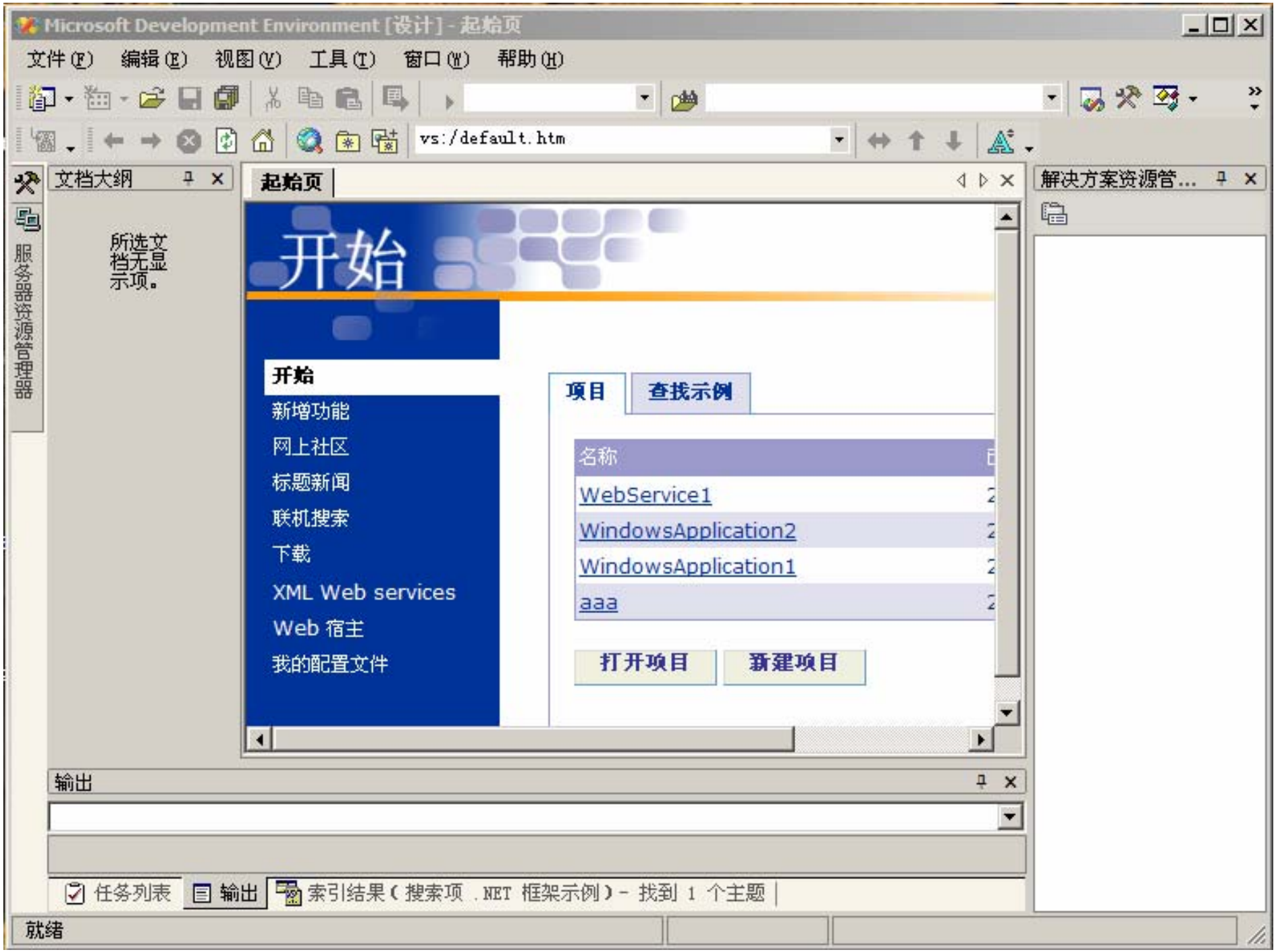
程序清单

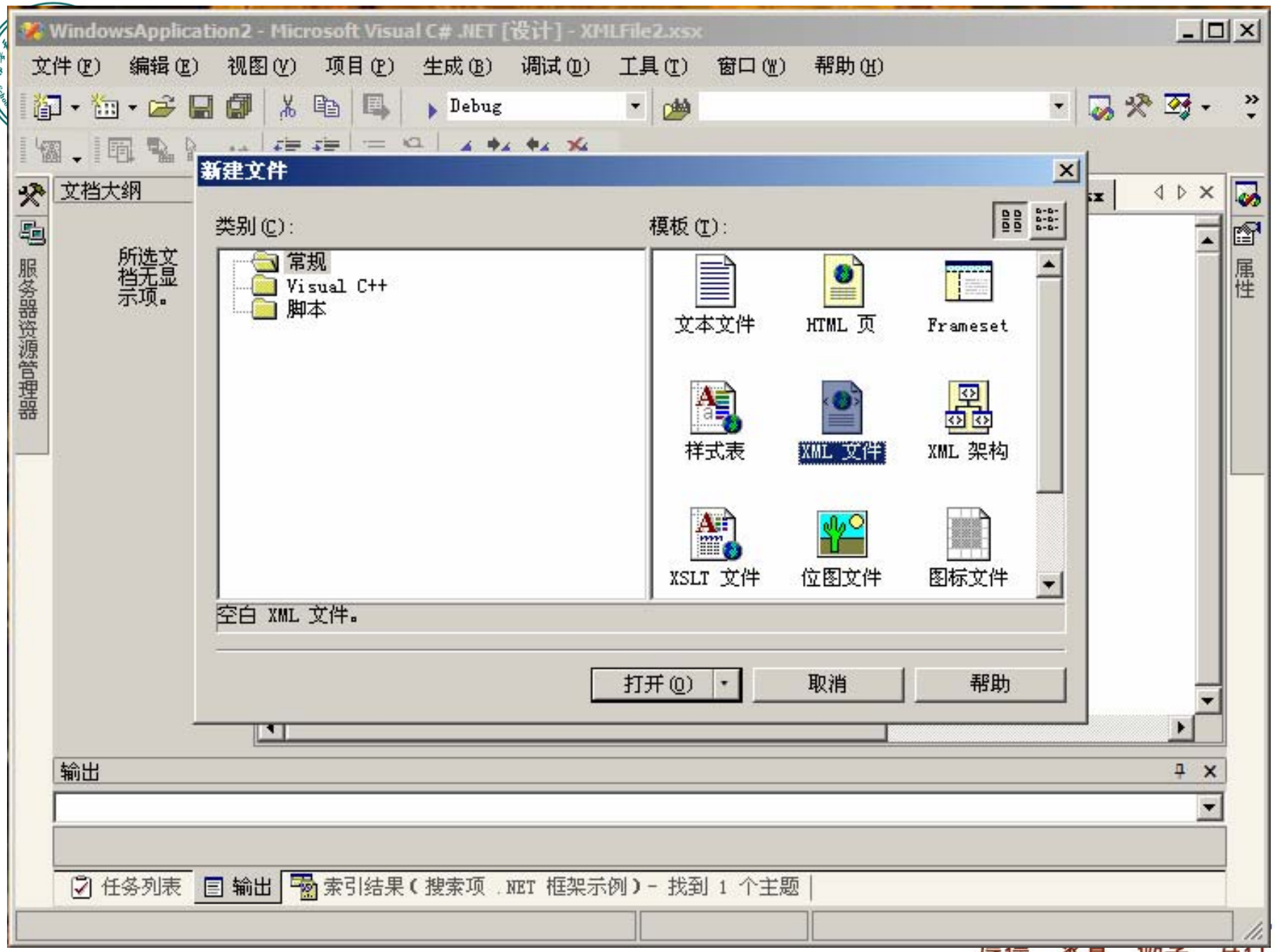
```
<!ELEMENT UNIT (MANUFECTURE,SERIAL,LOCATION,PURCHASE,DEVICE)>
<![%MONITOR[
        <!ELEMENT DEVICE(SCREENSIZE,COLOR)>
        <!ELEMENT SCREENSIZE(#PCDATA)>
        <!ELEMENT COLOR (color|back-with) >   ]]>
<!% IGNORE[
        <!ELEMENT DEVICE (PROCESSOR,RAM,HD)>
        <!ELEMENT PROCESSOR(#PCDATA)>
       <!ELEMENT RAM(#PCDATA)>
       <!ELEMENT HD(#PCDATA)>                 ]]>
<![% IGNORE[
       <!ELEMENT DEVICE(MEDIA,MAC)>
       <!ELEMENT MEDIA(#PCDATA)>
       <!ELEMENT MAC(color|back-with)>        ]]>
```

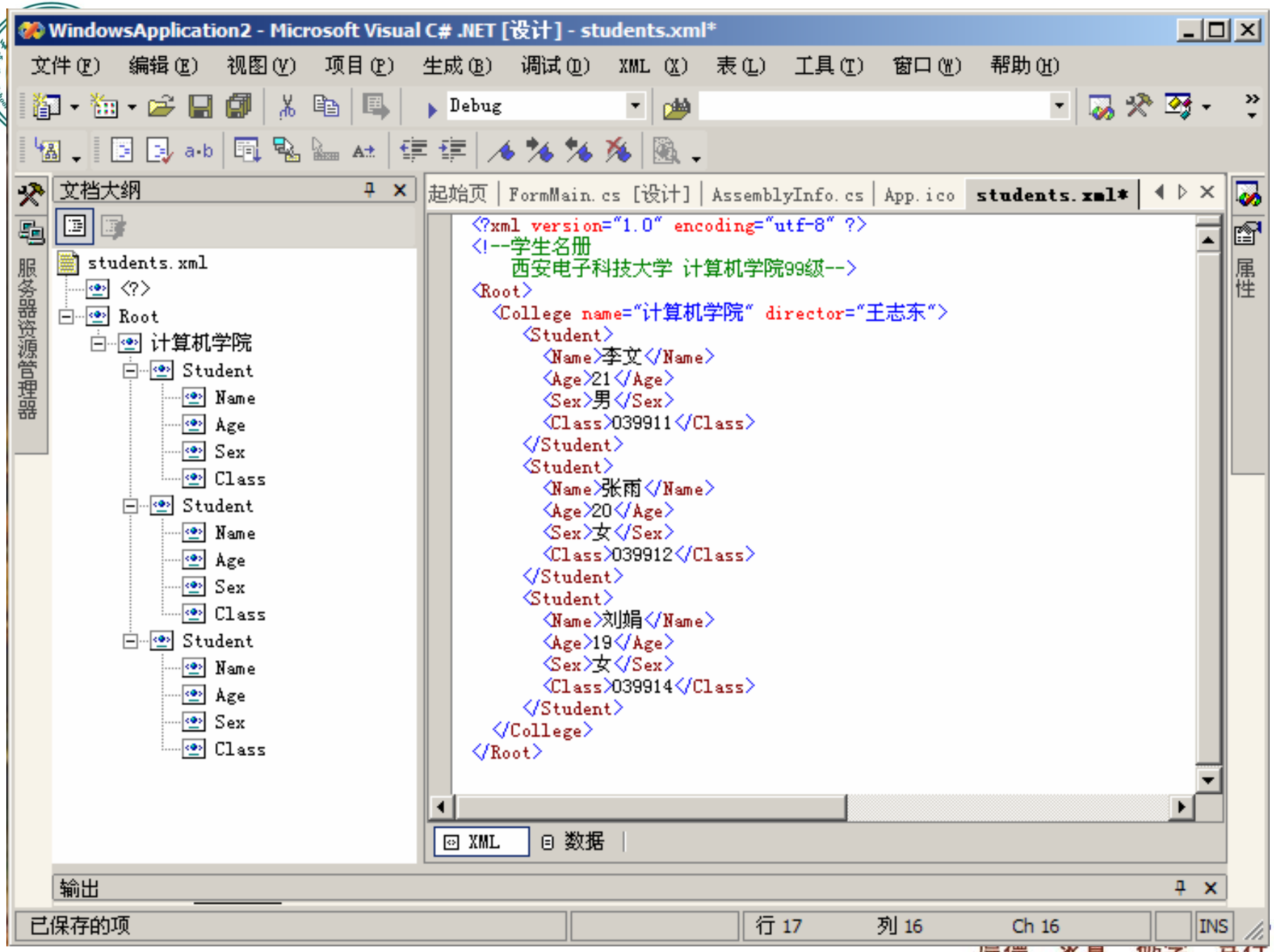
对应这个DTD的XML程序文档片段可以为：

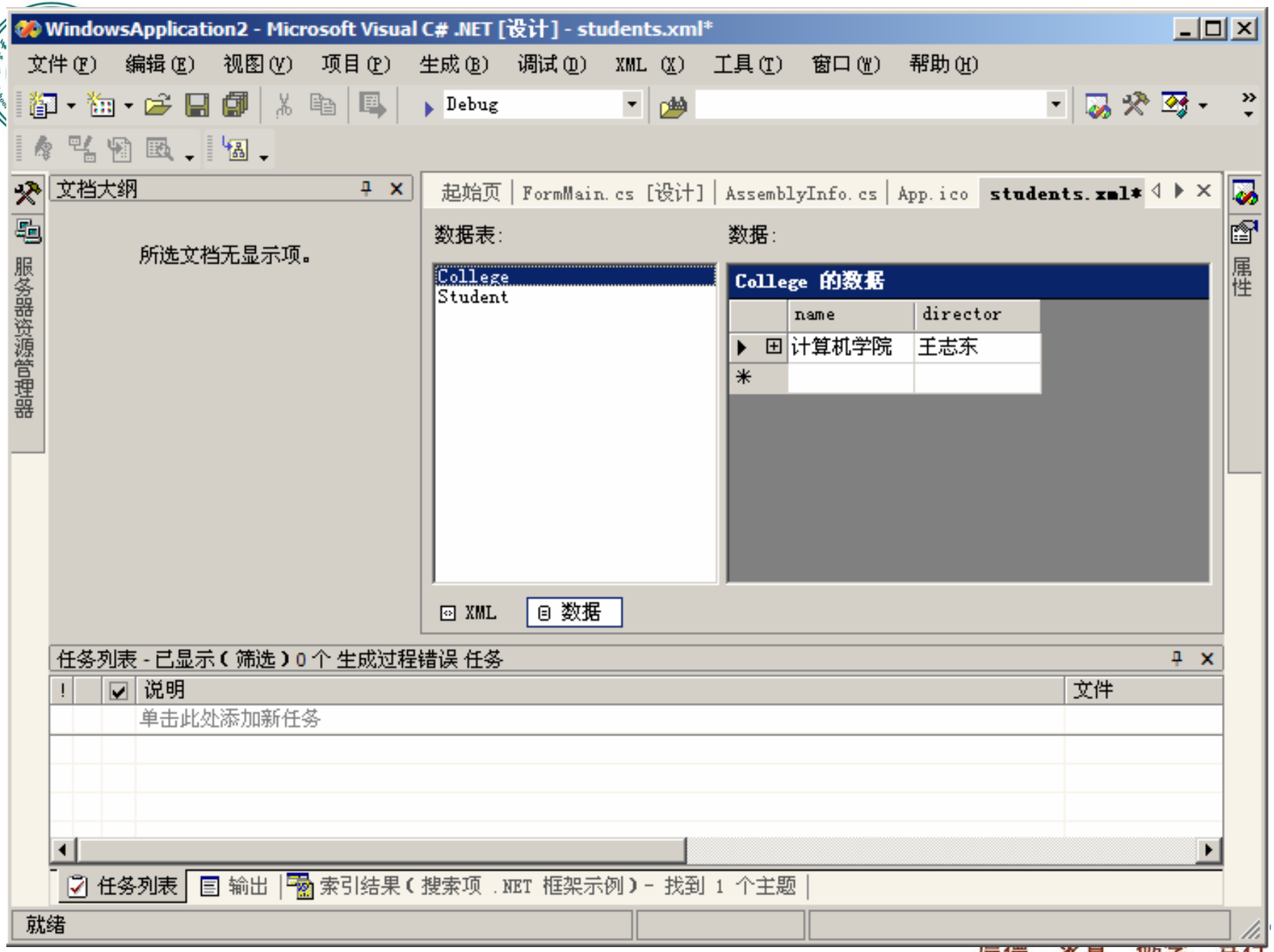
```
<?xml version= "1.0"?>
<!DOCTYPE UNIT[
      <!ENTITY %MONTIOR " IGNORE ">
      <!ENTITY %COMPUTER " INCLUDE ">
      <!ENTITY %NETWORKCARE " IGNORE ">
      < !ENTITY  %unit.dtd "unit.dtd">
%unit.dtd
]>
```

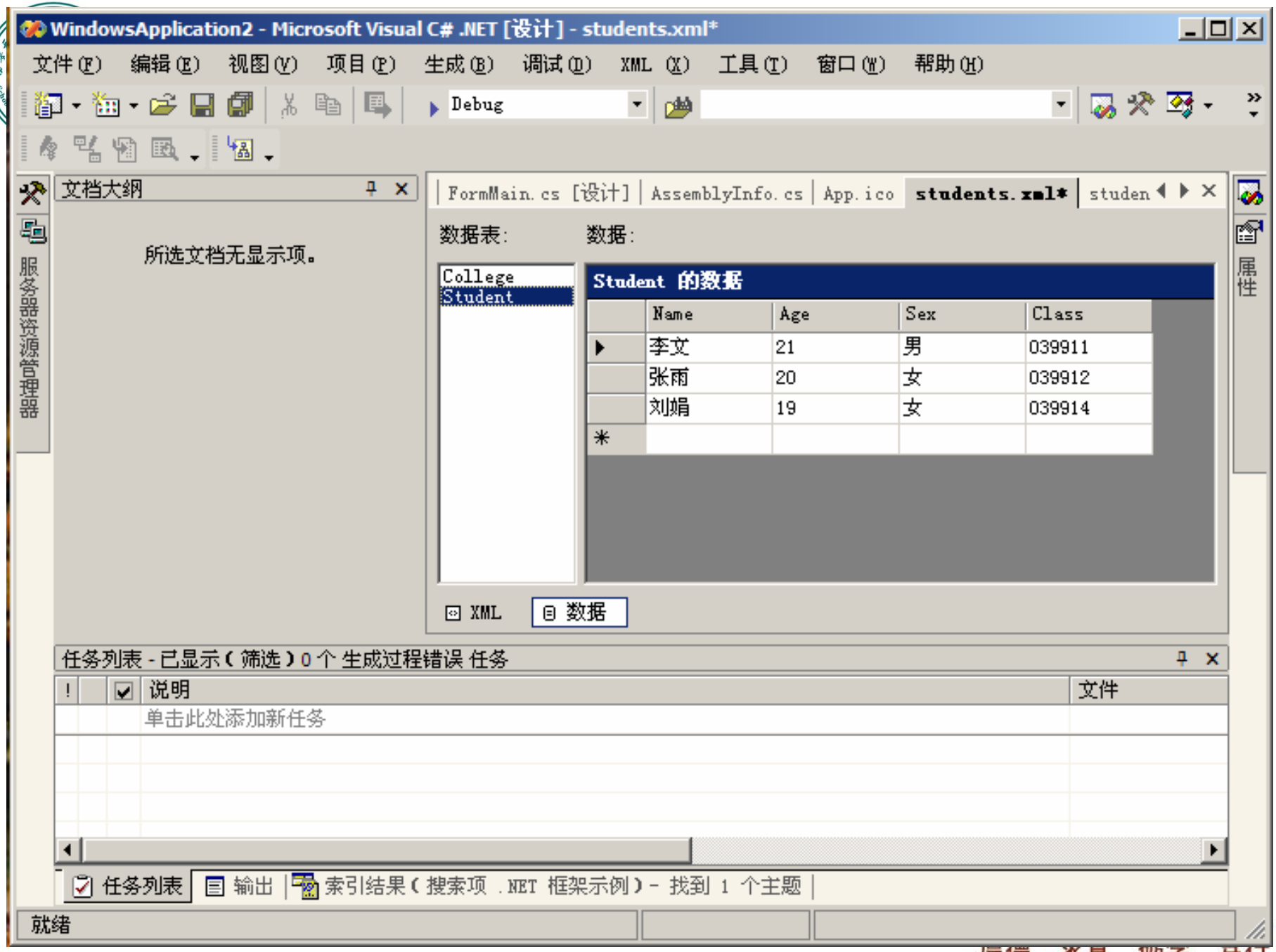
应用实例


Microsoft Development Environment [设计] - 起始页
文件(F) 编辑(E) 视图(V) 工具(T) 窗口(W) 帮助(H)
vs:/default.htm
文档大纲
所选文档无显示项。
服务器资源管理器
起始页
开始
开始
新增功能
网上社区
标题新闻
联机搜索
下载
XML Web services
Web 宿主
我的配置文件
项目
查找示例
名称
WebService1
WindowsApplication2
WindowsApplication1
aaa
打开项目
新建项目
解决方案资源管...
输出
任务列表
输出
索引结果(搜索项 .NET 框架示例) - 找到 1 个主题
就绪


WindowsApplication2 - Microsoft Visual C# .NET [设计] - XMLFile2.xsx
文件(F) 编辑(E) 视图(V) 项目(P) 生成(B) 调试(D) 工具(T) 窗口(W) 帮助(H)
Debug
新建文件
类别(C):
常规
Visual C++
脚本
模板(T):
文本文件
HTML 页
Frameset
样式表
XML 文件
XML 架构
XSLT 文件
位图文件
图标文件
空白 XML 文件。
打开(O)
取消
帮助
文档大纲
所选文档无显示项。
服务器资源管理器
属性
输出
任务列表
输出
索引结果(搜索项 .NET 框架示例) - 找到 1 个主题


WindowsApplication2 - Microsoft Visual C# .NET [设计] - students.xml*
文件(F) 编辑(E) 视图(V) 项目(P) 生成(B) 调试(D) XML(X) 表(L) 工具(T) 窗口(W) 帮助(H)
Debug
文档大纲
students.xml
<?>
Root
计算机学院
Student
Name
Age
Sex
Class
Student
Name
Age
Sex
Class
Student
Name
Age
Sex
Class
起始页 | FormMain.cs [设计] | AssemblyInfo.cs | App.ico | students.xml*
<?xml version="1.0" encoding="utf-8" ?>
<!--学生名册
西安电子科技大学 计算机学院99级-->
<Root>
<College name="计算机学院" director="王志东">
<Student>
<Name>李文</Name>
<Age>21</Age>
<Sex>男</Sex>
<Class>039911</Class>
</Student>
<Student>
<Name>张雨</Name>
<Age>20</Age>
<Sex>女</Sex>
<Class>039912</Class>
</Student>
<Student>
<Name>刘娟</Name>
<Age>19</Age>
<Sex>女</Sex>
<Class>039914</Class>
</Student>
</College>
</Root>
XML
数据
输出
已保存的项
行 17
列 16
Ch 16
INS
服务器资源管理器
属性


WindowsApplication2 - Microsoft Visual C# .NET [设计] - students.xml*
文件(F) 编辑(E) 视图(V) 项目(P) 生成(B) 调试(D) XML(X) 工具(T) 窗口(W) 帮助(H)
Debug
文档大纲
所选文档无显示项。
起始页 | FormMain.cs [设计] | AssemblyInfo.cs | App.ico | students.xml*
数据表:
数据:
College
Student
College 的数据
name
director
计算机学院
王志东
XML
数据
任务列表 - 已显示(筛选)0 个 生成过程错误 任务
说明
文件
单击此处添加新任务
任务列表
输出
索引结果(搜索项 .NET 框架示例)- 找到 1 个主题
就绪
服务器资源管理器
属性


WindowsApplication2 - Microsoft Visual C# .NET [设计] - students.xml*
文件(F) 编辑(E) 视图(V) 项目(P) 生成(B) 调试(D) XML(X) 工具(T) 窗口(W) 帮助(H)
Debug
文档大纲
所选文档无显示项。
服务器资源管理器
FormMain.cs [设计] AssemblyInfo.cs App.ico students.xml* studen
数据表:
数据:
College
Student
Student 的数据
Name Age Sex Class
李文 21 男 039911
张雨 20 女 039912
刘娟 19 女 039914
XML
数据
属性
任务列表 - 已显示(筛选)0 个 生成过程错误 任务
说明
文件
单击此处添加新任务
任务列表
输出
索引结果(搜索项 .NET 框架示例) - 找到 1 个主题
就绪

4.4 命名空间（Name Space）

4.4.1 多义性和名称冲突

不同系统中命名的冲突与差异会给不同系统的集成带来障碍：

- 元素的多义性会造成混乱。
- 相同含义的元素命名为不同的标识符

4.4. 2 定义和声明命名空间

命名空间是解决多义性和名字冲突问题的另一种方法。使用它的主要目的是为所有同一领域中的XML元素和属性提供统一的通用命名法则。

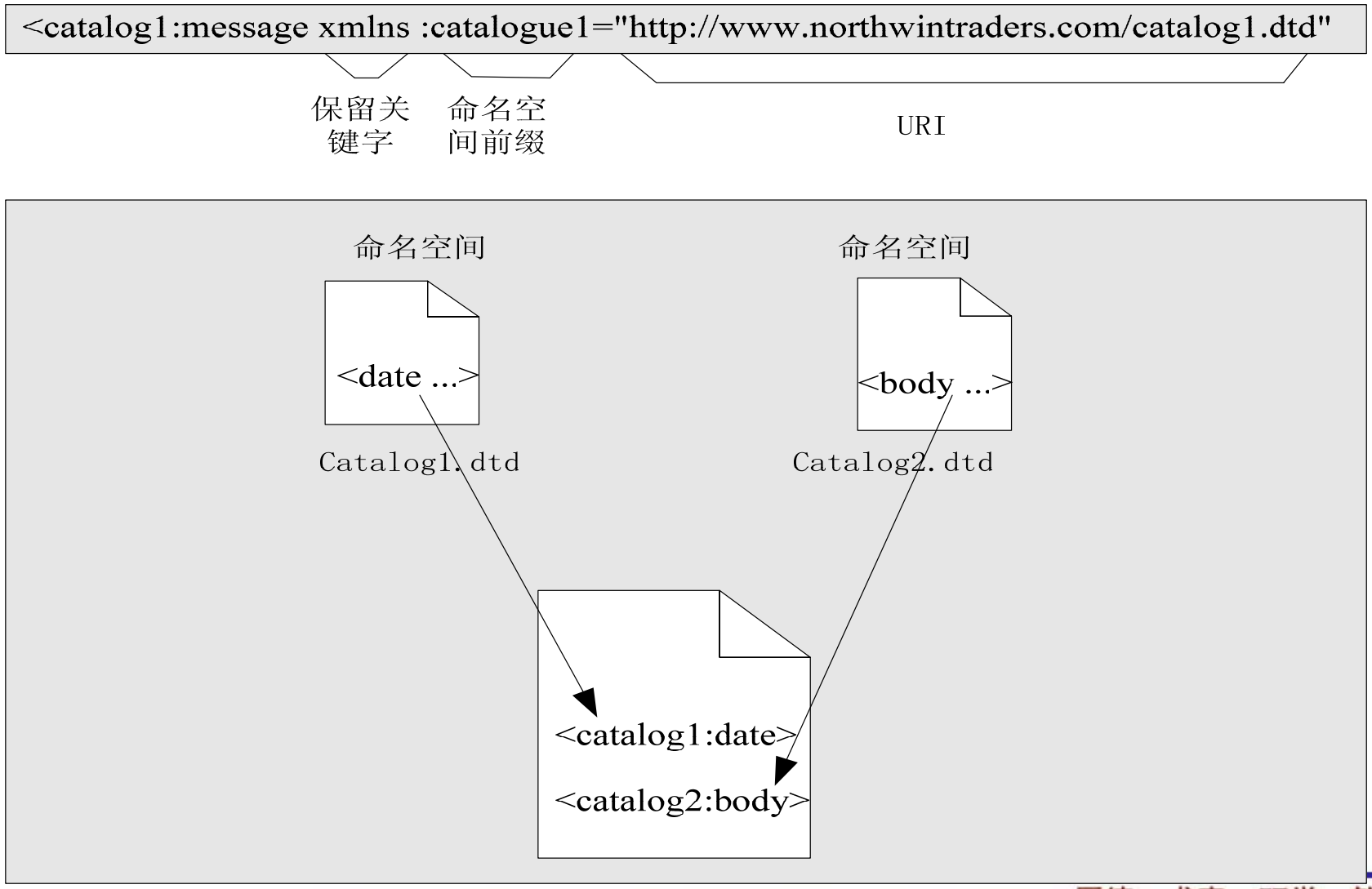
XML命名空间使用“统一资源标识符（URI）”进行命名组织。

命名空间是在元素的开始标签内声明的。声明命名空间的语法如下所示：

<元素名 xmlns="URI"> 或 <元素名 xmlns:前缀="URI">

程序清单

```
<?xml version="1.0"?>
  <message xmlns ="http://www.northwintraders.com/bill">
    <date>1/1/2002</date>
   <body>Your current balance is $1,999,999,00.Please pay
        immediately. </body>
  </message>
```


<catalog1:message xmlns :catalogue1="http://www.northwintraders.com/catalog1.dtd"
保留关键字
命名空间前缀
URI
命名空间
命名空间
<date ...>
<body ...>
Catalog1.dtd
Catalog2.dtd
<catalog1:date>
<catalog2:body>

4.4.3 属性和命名空间

在一个元素中，一个属性名只能被使用一次。使用命名空间前缀来限定属性名，可以使用不同的命名空间限定一个属性名，使得同名的属性可以在一个元素里出现。

```
<?xml version="1.0"?>
<message xmlns="http://www.northwindtrader.com/bill"
 xmlns:catalog="http://www.northwindtrader.com/message">
 <date>1/1/2002</date>
 <catalog:body catalog:importance="High" importance="High">
Your currentbalance is $1,999,999,00. Please pay immediately.
</ catalog:body>
</message>
```

4.5 模式（Schema）

2001年5月W3C正式发布了XML Schema的推荐标准，使得XML建模有了一个国际标准。XML Schema已经基本取代了DTD的地位，成为全球公认首选的XML环境下的建模工具。

一个简单的例子

比如一个简单的XML文档如下：

```
<书本>
   <名称>天涯明月刀
   <作者>古龙
```

一个简单的例子

如果用DTD的形式来定义该XML文档结构，可以表示为：

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

一个简单的例子

用Schema形式如何定义呢？

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，而内容模型实际上就是表示元素的类型。

```
<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>
```

4.5.1 XML模式与DTD

- XML是SGML的一个子集，也继承了SGML的DTD
- XML使用DTD的好处：利用SGML中的DTD工具
- DTD的缺点：
  - 非XML的语法规则；
  - 基于正则式表达，描述能力有限；
  - 不支持多种多样的数据类型；
  - 不支持结构化：继承性和复用性受限；
  - 扩展性较差

4.5.1 XML模式与DTD

- XML Schema正是针对DTD缺点而设计
- 是一种描述信息结构的机制
- 可用来定义XML文档结构、数据类型等
- 包含了DTD所有功能，本身是完全规范的XML文档

Schema优势

- 一致性，采用XML语法

可使用XML编辑器、解析器，也可使用标准编程接口文档对像模型DOM和XML简易API（SAX）对其进行处理

4.5.1 XML模式与DTD

- 扩展性：支持继承，可以构造新的模式

  可利用已经存在的模式中的某些类型构造新的模式。XML模式能够将一个模式分成单独的组件。

  软件复用

- 数据类型丰富

  内嵌的数据类型，可定义新数据类型

  <birthday>五一节</birthday>

  <birthday>2003-10-1</birthday>

- 易用性：简单易用

  在同一个命名空间中创建元素和属性非常容易

- 规范性：有专门的规定

4.5.2 XML模式规范

XML模式规范由三部分组成：

（1）XML Schema Part 0:Primer

提供XML模式的简单可读性的描述，目的是快速地理解如何使用XML Shema语言创建一个模式框架。

（2）XML Schema Part 1:Structure

这一部分详细说明了XML模式定义语言。XML模式由元素和属性声明、数据类型定义等模式组件组成。这些组件分为三组：

1. 基本组件:简单类型定义、复杂类型定义、元素声明、属性声明
2. 组件：属性组、约束定义、模型组、符号声明
3. 辅助组件：注释、模型组、小品词、通配符、属性使用

4.5.2 XML模式规范

模式组件详细说明了抽象数据模型的每个组件的严格语义，每个组件在XML中的表示。定义了对一个或多个框架存取、命名存取的必要前提条件，同时详细说明了模式和命名空间的关系。还包括了文档模式有效性评估的一般途径和模式处理器的责任。

（3）XML Schema Part 2:Datatypes

这一部分定义了用于XML模式和其他XML规范中定义数据类型的方法。

Schema实例：

```
[1]<?xml version="1.0"
encoding="GB2312" ?>
[2]<Schema xmlns="urn:schemas-microsoft-
com:xml-data"
        xmlns:dt="urn:schemas-microsoft-
com:datatypes">
[3]   <AttributeType name="公司"/>
[4]   <ElementType name= "姓名"/>
[5]   <ElementType name= "ID"/>
[6]   <ElementType name= "公司"/>
[7]  <ElementType name= "EMAIL"/>
[8]   <ElementType name= "电话"
dt:type="fixed.14.4"/>
[9]   <ElementType name= "街道"/>
[10] <ElementType name= "城市"/>
[11] <ElementType name= "省份"/>
[12] <ElementType name= "地址"
content="eltOnly">
[13]    <element type= "街道" />
[14]    <element type= "省份" />
[15]    <element type= "城市" />
[16] </ElementType>
[17] <ElementType name= "联系人"
content="eltOnly">
[18]    <element type= "姓名" />
[19]    <element type= "ID" />
[20]    <element type= "公司" />
[21]    <element type= "EMAIL" />
[22]    <element type= "电话" />
[23]    <element type= "地址" />
[24] </ElementType>
[25] <ElementType name= "联系人列表
" content="eltOnly">
[26]    <element type= "联系人" />
[27]    <attribute type="公司"/>
[28] </ElementType>
[29]</Schema>
```

◎ Schema文件的结构

Schema文件实际上就是一个XML文件，它是由一组元素构成的，其根元素是“Schema”。“Schema”元素是XML Schema中第一个出现的元素，“Schema”的结束标记在文档的末尾，用于表明该XML文档是一个Schema文档。

```
<?xml version="1.0">
<xsd:schema xmlns:xsd="http://www/w3.org/2001/XMLSchema">
<xsd:element name="BOOK" type="pl:bookType"/>
< xsd:complexType name="bookType">
    < xsd:sequence>
      < xsd:element name="TITLE" type="string"/>
    </ xsd:sequence>
</ xsd:complexType>
< xsd:simpleType name="authorType">
    < xsd:restriction base="string">
       < xsd:enumeration value="男"/>
       < xsd:enumeration value="女"/>
    </ xsd:restriction>
</ xsd:simpleType>
</ xsd:schema>
```

一个Schema的结构如下：

```
<Schema name="schema-name" xmlns="namespace" >
  ...
</Schema>
```

两个属性：name指定该Schema的名称，而xmlns则指定该Schema包含的命名空间。一个XML Schema文档中可以包含多个命名空间，比如下面的语句指定了三个命名空间：

```
<Schema name="mySchema"
    xmlns="urn:schemas-microsoft-com:xml-data"
    xmlns:dt="urn:schemas-microsoft-com:datatypes"
   xmlns:myNS=http://www.xml_step_by_step.edu\ns.xml>
```

第一个指定本文档是一个XML Schema文档；

第二个定义了在本文档中可以使用的数据类型；

第三个表明下面可能会用到在myNS中定义的元素或属性。

4.5.3 数据类型

XML模式应用最为广泛的方面也许就是其丰富的数据类型。

XML模式数据类型规范定义了两类数据类型：

- 原始数据类型（Primitive Data Types）
- 派生数据类型（Derived Data Types）

所有原始数据类型都在规范中加以定义，因此都属于内置数据类型（Built-In Data Types），而用户创建的都是派生数据类型

◎ 原始数据类型

几种常用的原始数据类型。

- string 任意长度的字符串。
- boolean 是下列任一字符串：真值为1/true，假值为0/false
- decimal 任意精度的十进制数据。可以带有正负号或小数点
- float 单精度32 位浮点数字
- anyURI 代表一个URI，通常是URL
- QName 代表命名空间的限定名。如果后面带有冒号，则冒号前面的文本是命名空间的前缀，否则说明该名称属于默认命名空间
- NOTATION 对应于XML NOTATION属性类型
- hexBinary 使用十六进制表示的二进制数据类型
- base64Binary 用base64编码的二进制数据
- duration 表示时间长度，例如P2Y表示两年，而P1Y3M5D表示一年零三个月零五天
- date 表示日期类型

◎ 派生数据类型

派生数据类型基于原始数据类型之上，XML模式规范规定了一些派生数据类型，并给出了用户派生定义数据类型的工具

- integer 没有分数部分的十进制数字
- byte 8位带符号数字
- short 16位数字
- ID 文档中单个元素独有的标识符
- IDREF 文档中唯一ID的引用
- IDREFS IDREF值的空格分隔列表
- ENTITY 文档中非解析实体的名称
- ENTITIES ENTITY值的空格分隔列表
- NMTOKEN 带有置于其之上的特定字符约束的字符串
- NMTOKENS NMTOKENS值的空格分隔列表

◎ 侧面（facet）

侧面用来从不同角度对数据类型进行约束。下面是几种常用的侧面：

- minInclusive和maxInclusive 约束新数据类型允许的最小值和最大值
- minExlusive和maxExclusive 约束新数据类型取值的上界和下界（不能取设定值）
- minLength和maxLength 约束新数据类型的长度
- totalDigitals 约束十进制表示中的最大位数
- fractionDigis 约束十进制小数部分的最大位数
- enumeration 约束新数据类型的取值只能为枚举出的值
- pattern 用正则表达式约束新类型的取值格式

4.5.4 类型声明

典型定义包括一组元素声明、元素引用和属性声明。元素使用element元素声明，属性使用attribute来声明

◎ 复合类型（co

XML模式支持的复合类型一般为用户自定义的数据类型，它可以包含子元素、属性和字符数据。新的复合类型使用complexType元素来定义：

name+type型声明

```
<xsd:complexType name="USAddr
  <xsd:sequence>
    <xsd:element name="name
    <xsd:element name="stre
    <xsd:element name="city" type
    <xsd:element name="state" type="xsd:string"/>
    <xsd:element name="zip" type="xsd:decimal"/>
  </xsd:sequence>
  <xsd:attribute name="country" type="xsd:NMTOKEN" fixed="US"/>
</xsd:complexType>
```

实例文档中出现的任何类型声明为USAddress的元素必须包含五个元素和一个属性。且属性值必须为US

4.5.4 类型声明

◎ 复合类型（co

将不同的元素名字和相同的复合类型USAddress相关联.
其属性为简单类型

```
<xsd:complexType name="Purcha      erType" >
  <xsd:sequence>
     <xsd:element name="s     To" type="USAddress"/>
     <xsd:element name=     llTo" type="USAddress "/>
     <xsd:element ref=     comment" miniOccurs="0" />
     <xsd:element name="items" type="Items"/>
  </xsd:sequence>
  <xsd:attribute name="OrderDate" type="xsd:date"/>
</xsd:complexType>
```

◎ ref引用

- <xsd:element ref="comme

这个声明定义引用了一个
的其他地方有定义。一般的
通常是在引用定义的下面单
分声明的。这个声明的结果
的关于这个定义的相关部
的元素的类型一致。

- 约束元素的出现次数

在前面中的元素声明中mi
在PurchaseOrderType类型中是一个可选项。minOccurs表示元素最少出现次数，而maxOcurrs则表示元素的最多出现次数。

- 约束元素的缺省值

可以使用<xsd:element>元素的fixed或default属性（一次只能使用其中一个）来限定元素的取值范围。

- 约束属性

属性与元素不同，属性只能是简单类型，而且不能对属性使用minOccurs和maxOccurs约束。但是，可以使用<xsd:attribute>元素的use和value属性来对定义的属性进行约束。

- minOccurs和maxOccurs缺省为1
- minOccurs：某个元素必须出现
- maxOccurs：unbounded表示不限出现的最大数量
- 当一个元素没有定义maxOccurs时，出现不能超过一次
- 若仅仅指定minOccurs，它必须小于等于1
- 若只指定maxOccurs，必须大于等于minOccurs缺省值1
- 都省略：元素必须且只能出现一次

◎ 简单类型（Simple Type）

不包含任何子元素，但可以包含数字、字符串、日期等简单数据内容的元素称为简单类型元素。简单类型包括原子类型（Atomic Types）、列表类型（List Type）和联合类型（Union Type）三种。

- 约束派生简单类型
- 列表派生简单类型
- 联合派生简单类型

4.5.5 模式结构的组织

◎ 元素内容模型：元素类型定义

◎ 注　　释：便于阅读理解

◎ 元 素 组 合 ：组合使用元素

◎ 属 性 编 组 ：创建属性组

◎ 空　　值：元素内容可为空

◎ 元素内容模型

模式文档中包含很多种的元素类型定义。有的元素包含其他元素，有的元素包含其他元素和属性，也有的元素包含一个简单类型值，同时也有元素和文本内容混合在一起的元素。

- 简单内容模型

声明包含了一个属性，同时拥有简单类型值的元素。

<myPrice currency="RMB">423.46</myPrice>

购买订单模式文档中的Price元素声明：

<xsd:element name="myPrice" type="decimal"/>

◎ 元素内容模型

```
<xsd:element name="myPrice">
  <xsd:ComplexType>
    <xsd:simpleContent>
      <xsd:extension base="xsd:decimal">
<xsd:attribute name="currency" type="xsd:string"/>
      <xsd:extension>
    <xsd:simpleContent>
  <xsd:ComplexType>
<xsd:element>
```

混合内容模型

在混合内容模型中，元素中可以混合文本数据和嵌套子元素

```
<letterBody>
   <salutation>Dear Mr.<name>Robert Smith</name>.</salutation>
   Your order of <quantity>1</quantity> <productName>Baby
   Monitor</productName> shipped from our warehouse on
   <shipDate>1999-05-21</shipDate> ....
</letterBody>
```

```
<xsd:element name="letterBody">
 <xsd:complexType mixed="true">
  <xsd:sequence>
   <xsd:element name="salutation">
    <xsd:complexType mixed="true">
     <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
     </xsd:sequence>
    </xsd:complexType>
   </xsd:element>
   <xsd:element name="quantity"    type="xsd:positiveInteger"/>
   <xsd:element name="productName" type="xsd:string"/>
<xsd:element name="shipDate"    type="xsd:date" minOccurs="0"/>
   <!-- etc. -->
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>
```

空模型

- 想让myPrice元素用属性值来传送货币的单位和价格，而不是象先前用一个属性值以及元素内容值来表示。例如：

```
<myPrice currency="RMB" value="423.46"/>
```

```
<xsd:element name=" myPrice">
 <xsd:complexType>
  <xsd:complexContent>一只包
的内容模型
   <xsd:restriction base="xsd:
    <xsd:attribute name="curre
    <xsd:attribute name="value
   </xsd:restriction>——元素restr
  </xsd:complexContent>
 </xsd:complexType>
</xsd:element>
```

这样的元素，内容模型为空；为了定义内容为空的元素：首先定义一个只能包含子元素而不能包含元素内容的元素；然后又不定义任何子元素。

■ 任意模型

anyType是导出所有简单类型和复合类型的基类型。一个anyType类型不以任何形式约束其包含的内容。我们可以像使用其他类型一样使用anyType，如：

```
<xsd:element name="anything" type="xsd:anyType"/>
```

用这种方式声明的元素是不受约束的。所以元素的值可以为423.46，也可以为任何其他的字符序列，或者甚至是字符和元素的混合。实际上，anyType是默认类型，所以上面的可以被重写为：

```
<xsd:element name="anything"/>
```

若需要表示不受约束的元素内容，那么默认的无约束声明或者有些微约束的形式会很合适

◎ 注 释

为了方便读者和应用程序理解模式文档，XML模式提供了三个元素用作注释:documentation、appInfo 和annotation。其中，<xsd:annotation>元素实际包含了<xsd:documentation>和<xsd:appInfo>元素。

```
<xsd:simpleType name=" xsd:string" base="urSimpleType">
<xsd:annotation>
  <xsd:appInfo>
    <has-facet name="length"/>
    <has-facet name="pattern"/>
    <has-facet name="enumeration"/>
    <has-facet name="minInclusive"/
    <has-property name="ordered" va
    <has-property name="bounded" va
    <has-property name="cardinality"
  </xsd:appInfo>
< /xsd:annotation >
</xsd:simpleType>
```

<xsd:documentation>存储普通类型的注释文本，即为阅读模式的人设计的文本

<xsd:appInfo>存储有助于应用程序阅读模式的注释，应用程序可以从中选择有用的信息

◎ 注释

```
<xsd:annotation>
  <xsd:documentation>
      I am very happy!
  </ xsd:documentation >
< /xsd:annotation >
```

◎ 元素组合

模式文档中复合类型定义了一系列的元素，这些元素在实例文档中都必须出现。某个元素的出现是可选的，可以通过属性值maxOccurs=“1”来实现。

也可以通过对minOccurs和maxOccurs赋予不同的值来实施其他不同的约束限制。XML模式也支持对在内容模型中出现的元素组进行约束。

◎ 属性编组

使用<xsd:attributeGroup>元素创建属性组。通过这种方法来使用属性组，可以提高模式文档的可读性，同时也便于更新模式文档。这是因为一组属性能够集中在一个地方定义和编辑，同时能够在多个定义和声明中被引用。注意到一个属性组可以包含其他属性组，同时属性组的声明和引用必须在复合类型定义的最后。

```
<xsd:attributeGroup name="bookAttrGroup">
<xsd:attribute name="bookID" type="xsd:string"/>
<xsd:attribute name="pages" type="xsd:decimal"/>
<xsd:attribute name="coverType" >
   <xsd:simpleType>
     <xsd:enumeration value="leather">
     <xsd:enumeration value="cloth">
     <xsd:enumeration value="viyl">
     </xsd:simpleType>
</xsd:attribute>
<xsd:attributeGroup>

<xsd:element name="book">
   <xsd:complexType>
     <xsd:element name="bookTitle" type="xsd:string"/>
     <xsd:element name="pubDateTitle" type="xsd:date" minOccurs="0"/>
     <xsd:element name="price" type="xsd:decimal"/>
     <xsd:attributeGroup ref=" bookAttrGroup"/>
   </xsd:complexType>
</xsd:element>
```

◎空值（Nil）

XML模式的空值机制表示元素的内容可以为空。可以用以下方式进行声明：

```
<xsd:element name="shipDate" type="xsd:date"
nillable="true"/>
```

为了在实例文档中明确的表示元素可以有空值，可以设置nil属性为真：

```
<shipDate xsi:nil="true"></shipDate>
```

nil属性是作为XML模式命名空间"http://www.w3.org/2001/XMLSchema-instance"的一部分来定义的，并且在实例文档中必须带有与命名空间相对应的前缀(一般为xsi:)出现。需要注意的是：空值机制仅仅适用于元素值，而不适用于属性值。一个元素有xsi:nil="true"可以没有任何元素内容但仍旧可以带有其他属性。

4.6 应用程序接口

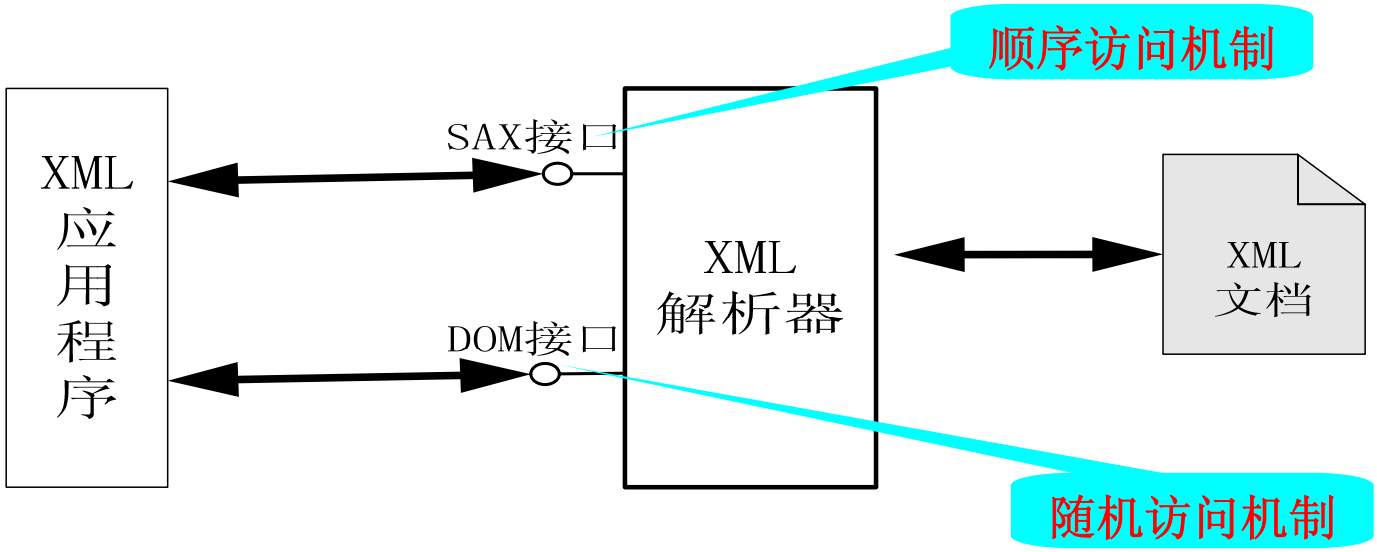
XML文档就是一个文本文件，在访问其内容时，必须首先书写一个能够识别XML的文本阅读器，也就是XML解析器（Parser），由它对XML文档的语法正确性进行验证，并提取其中的内容。XML文档有时是动态生成的，使得用户能够创建、访问和修改一个XML文件。有时，开发的应用程序需要能读懂别人写的XML文件，从中提取所需要的信息

在以上情况下，都需要一个类似于ODBC/JDBC这样的数据库接口规范的统一的XML接口，这个接口使得应用程序与XML文档结合在一起，让应用程序能够对XML文档提供完全的控制

4.6 应用程序接口

W3C制定了一套书写XML解析器的标准接口规范：文档对象模型（Document Object Model, DOM）。除此之外，XML_DEV邮件列表中的成员根据应用的需求也自发地定义了一套对XML文档进行操作的接口规范：简单应用程序接口（Simple APIs for XML, SAX）。这两种接口规范各有侧重、互有长短，都得到了广泛的应用。

DOM和SAX两个接口标准所要实现的目标不同，因此能够并存，它们在接口实现过程中分别侧重于不同的方面，分别满足了不同的应用需求



DOM和SAX在应用程序开发过程中所处地位

4.6.1 DOM

DOM的全称是文档对象模型（Document Object Model）。在应用程序中，基于DOM的XML解析器将一个XML文档转换成一棵DOM树，应用程序正是通过DOM树来实现对XML文档数据的操作。通过DOM接口，应用程序可以在任何时候访问XML文档中的任何一部分数据，因此，这种利用DOM接口的机制也被称作随机访问机制。

◎DOM树

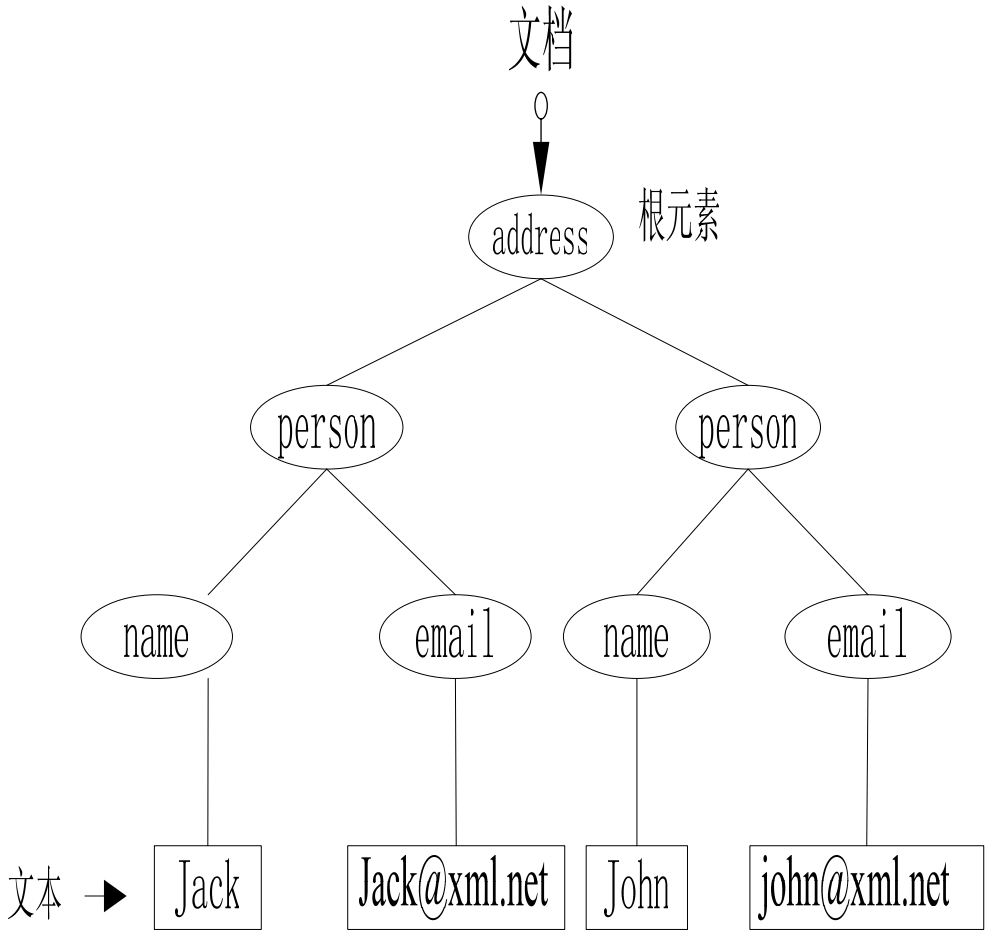
无论XML文档中所描述的是什么类型的信息，利用DOM所生成的模型都是节点树的形式。也就是说，DOM强制使用树模型来访问XML文档中的信息。在这种模型下，每个元素对应一个节点，而每个节点都可以包含它自己的节点子树，在每个文档的顶端是文档根节点。由于XML本质上就是一种分层结构，所以这种描述方法是相当有效的。

程序清单

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

<address>
  <person sex = "male">
     <name>Jack</name>
     <email>Jack@xml.net </email>
  </person>

  <person sex = "male">
     <name>John</name>
     <email>john@xml.net</email>
  </person>
</address>
```



◎ DOM的基本接口

在DOM接口规范中，有四个基本的接口：Node、Document、NodeList以及NamedNodeMap。

●Node接口：其他大多数接口的基接口

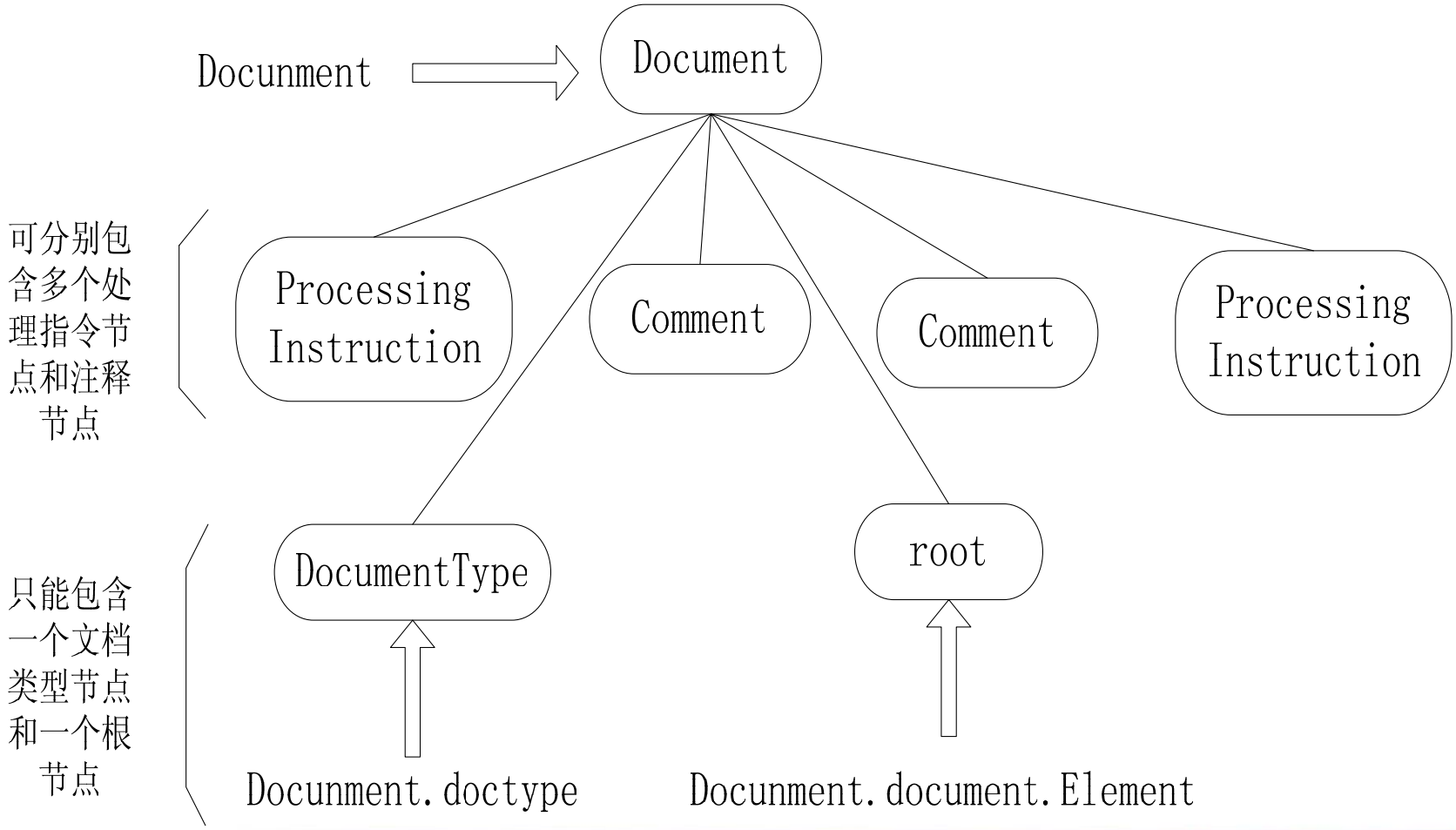
●Document接口：是对文档进行操作的入口，它是从Node接口继承过来的。Element、Attribute、Text、Comment等接口都是从Node接口继承过来的

●NodeList接口：是一个节点的集合，它包含了某个节点的所有子节点

●NamedNodeMap接口：也是一个节点的集合，通过该接口，可以建立节点名和节点之间的一一映射关系，从而利用节点名可以直接访问特定的节点

•Document接口

•Document接口代表了整个XML文档，因此，它是整棵文档树的根，提供了对文档中的数据进行访问和操作的入口。



• Node接口

Node接口在整个DOM树中具有举足轻重的地位，DOM接口中有很大一部分接口是从Node接口继承过来的，例如，Element、Attr、CDATASection等接口，都是从Node继承过来的。在DOM树中，Node接口代表了树中的一个节点。Node接口提供了访问DOM树中元素内容与信息的途径，并给出了对DOM树中的元素进行遍历的支持。

• NodeList接口

NodeList接口提供了对节点集合的抽象定义，它并不包含如何实现这个节点集的定义。NodeList用于表示有顺序关系的一组节点，比如某个节点的子节点序列。另外，还出现在一些方法的返回值中，如GetNodeByName。

在DOM中，对文档的改变，会直接反映到相关的NodeList对象中。如，通过DOM获得一个NodeList对象，该对象中包含了某个Element节点的所有子节点的集合，那么，当再通过DOM对Element节点进行操作（添加、删除、改动子节点）时，这些改变将会自动地反映到NodeList对象中，而不需DOM应用程序再做其他额外的操作。

NodeList中的每个item都可以通过一个索引来访问，该索引值从0开始。

• NamedNodeMap接口

实现了NamedNodeMap接口的对象中包含了可以通过名字来访问的一组节点的集合。但是，NamedNodeMap并不是从NodeList继承过来的，它所包含的节点集中的节点是无序的。尽管这些节点也可以通过索引来进行访问，但只是提供了枚举NamedNodeMap中所包含节点的一种简单方法，并不表明在DOM规范中为NamedNodeMap中的节点规定了一种排列顺序。

NamedNodeMap表示的是一组节点和其唯一名字的一一对应关系，这个接口主要用在属性节点的表示上。

◎ DOM编程

- 加载XML文档

MSXML提供了一个load方法来加载XML文档，在内存中创建DOM树

- 获取根元素节点

在访问XML文档时，可以从根元素开始。获取根元素节点

```
var XML_root=XML_Doc.domentElement;
```

返回一个XMLDOMNode对象，变量XML_root就代表了文档树结构中的元素根节点

- 浏览DOM树

DOM树是一个层次化的容器，层次由节点组成。即XML处理都要浏览树结构，以操作存储在XML文档中的信息。

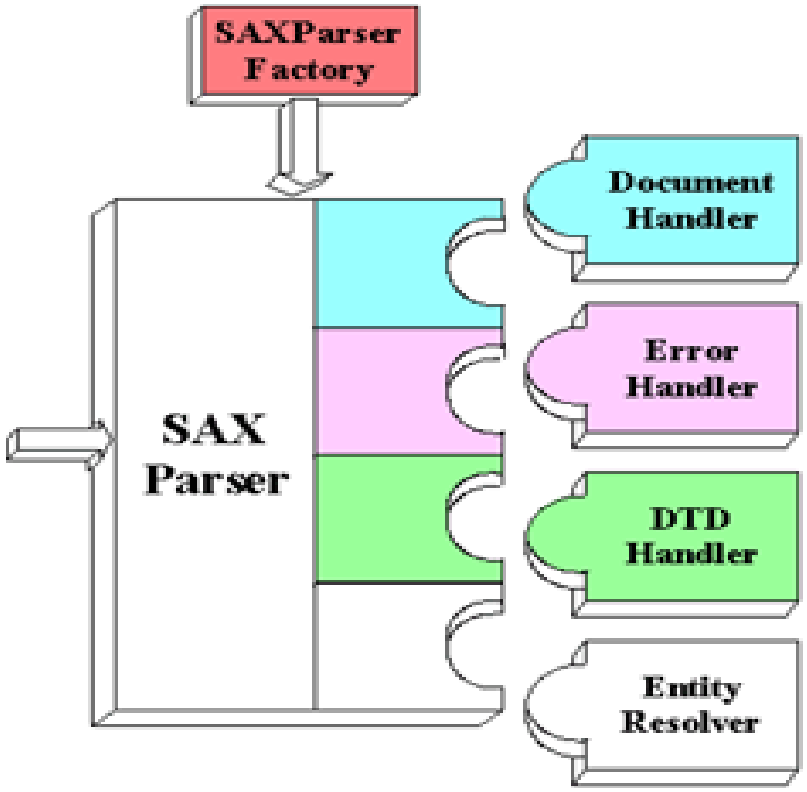
一种通用的方法是遍历节点树及其元素值

4.6.2 简单应用程序接口（SAX)

SAX的全称是XML简单应用程序接口（Simple APIs for XML）。

与DOM不同，SAX采用顺序访问模式，是一种快速读写XML数据的方式。当使用SAX解析器对XML文档进行分析时，会触发一系列事件，并激活相应的事件处理函数，应用程序通过这些事件处理函数实现对XML文档的访问，因而SAX接口也被称作事件驱动接口。

◎ SAX分析器接口



图中最上方的SAXParserFactory用来生成一个分析器实例。XML文档是从左侧箭头所示处读入，当分析器对文档进行分析时，就会触发在DocumentHandler，ErrorHandler，DTDHandler以及EntityResolver接口中定义的回调方法。

4.7 XML文档的显示

层叠样式单（Cascading Style Sheet，简称 CSS）和扩展样式单语言（eXtensible Style sheet Language，简称XSL）是W3C推荐的表达XML文档数据显示格式的两种标准。

4.7.1层叠样式单CSS（Cascading Style Sheets）

◎ CSS的书写规范

利用CSS，可以定义HTML或XML文档中元素的显示效果，包括元素的位置、颜色、背景、边空、字体、排版格式等

CSS的基本思想是为结构文档中的各个标记定义出相对应的一组显示样式。 定义的基本格式为：

选择符 { 样式属性：取值；样式属性：取值；... }

```
<HTML>
<HEAD>
  <STYLE>
   H1{color:red; }
   .myclass{color:green}
  H2.myclass{color:blue}
  #myid{color:brown}
  </STYLE>
</HEAD> <BODY>
  <H1>这是红色的一号标题。</H1>
  <P class="myclass">"myclass"类中的正文是绿色的。</P>
  <H2 class="myclass">但"myclass"类中的二号标题是蓝色的。</H2>
<P class="myclass" id="myid">以"myid"为标识的正文则是棕色的。</P>
</BODY>
</HTML>
```

这是红色的一号标题。

"myclass"类中的正文是绿色的。

但"myclass"类中的二号标题是蓝色的。

以"myid"为标识的正文则是棕色的。

◎使用CSS显示XML文档

引用式

具体实现的方法是，在XML文档的开头写一个关于样式单的声明语句，如下：

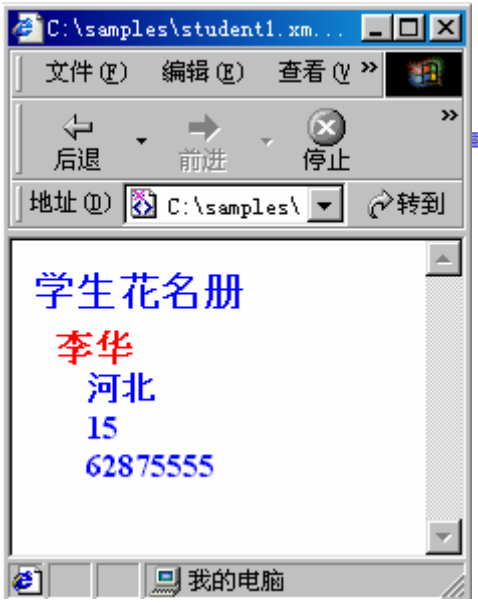
```
<?xml-stylesheet type="text/css" href="mystyle.css" ?>
```

这样，按照声明语句的指示，该文档在浏览器上的表现方式就由样式文件mystyle.css所决定

内嵌式

内嵌式是指将CSS样式直接嵌入到XML文档内部，为元素设置style属性，并在属性值中给出对其样式的定义。这种用法主要出现在一些特殊的XML文档中，一般内嵌CSS样式的XML文档本身就是面向显示的，如SVG、SMIL

◎实例



student.xml

```
<?xml version="1.0" encoding="gb2312" ?>
    <?xml-stylesheet type="text/css"
    href="mystyle.css"?>
    <roster>
        学生花名册
        <student>
            <name>李华</name>
            <origin>河北</origin>
            <age>15</age>
            <telephone>62875555</telephone>
        </student>
    </roster>
```

mystyle.css:

```
roster,student
{
  font-size:15pt;
  font-weight:bold;
  color:blue;
  display:block;
  margin-bottom:5pt;
}
origin,age,telephone
{
  font-weight:bold;
  font-size:12pt;
  display:block;
  color:block;
  margin-left:20pt;
}
name
{
  font-weight:bold;
  font-size:14pt;
  display:block;
  color:red;
  margin-top:5pt;
  margin-left:8pt;
}
```

4.7.2 扩展样式单语言（XSL）

CSS是一种静态的样式描述格式，其本身不遵从XML的语法规范。扩展样式单语言（eXtensible Style sheet Language，XSL）不同，它遵守XML的语法规则，是XML的一种具体应用。这也就是说，XSL本身就是一个XML文档，系统可以使用同一个XML解释器对XML文档及其相关的XSL文档进行解释处理。

XSL语言已经被分割成三个不同的部分：

转换工具XSLT（XSL Transformations）。它描述了如何将一个没有形式表现的XML文档内容进行转换，转换为可浏览或可输出的格式。

格式对象FO（fomatted object）。

XML分级命令处理工具XPath。

一个XML文档的显示过程是这样的：首先根据XML文档构造源树；然后根据给定的XSL将这个源树转换为可以显示的结果树，这个过程称作树转换；最后再按照FO解释结果树，产生一个可以在屏幕上、纸上、语音设备或其它媒体中输出的结果，这个过程称作格式化。

4.2 可扩展样式单语言XSL (eXtensible Stylesheet Language)

为了使用XSL样式单，XML文档中样式单声明语句应改为：

```
<?xml-stylesheet type="text/xsl" href="mystyle.xsl"?>
```

例如：

```
<?xml version="1.0" encoding="gb2312" ?>
<?xml-stylesheet type="text/xsl" href="mystyle.xsl"?>
<roster>
    学生花名册
    <student>
        <name>李华</name>
        <origin>河北</origin>
        <age>15</age>
        <telephone>62875555</telephone>
    </student>
    <student>
        <name>张三</name>
        <origin>北京</origin>
        <age>14</age>
        <telephone>82873425</telephone>
   </student>
</roster>
```

一个XSLT的简单例子

样式单头

```
<?xml version="1.0" encoding="gb2312" ?>
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns="http://www.w3.org/TR/REC-html40">
<xsl:template>
   <xsl:apply-templates/>
</xsl:template>

<xsl:template match="/">
<HTML>
 <HEAD>
  <TITLE>学生花名册</TITLE>
  <STYLE> .title{font-size:15pt; font-weight:bold;
color:blue } .name{color:red}
  </STYLE>
 </HEAD>
 <BODY>
  <P class="title" >学生花名册</P>
  <xsl:apply-templates select="roster"/>
</BODY>
 </HTML>
</xsl:template>
```

声明

模板

匹配

```
<xsl:template match="roster">
<TABLE BORDER="1">
  <THEAD>
   <TD> <B>姓名</B> </TD>
   <TD> <B>籍贯</B> </TD>
   <TD> <B>年龄</B> </TD>
   <TD> <B>电话</B> </TD>
  </THEAD>
  <xsl:for-each select="student" order-by="name">
  <TR>
   <TD><B><xsl:value-of select="name"/></B></TD>
   <TD><xsl:value-of select="origin"/></TD>
   <TD><xsl:value-of select="age"/></TD>
   <TD><xsl:value-of select = "telephone"/></TD>
  </TR>
  </xsl:for-each>
 </TABLE>
</xsl:template>
</xsl:stylesheet>
```

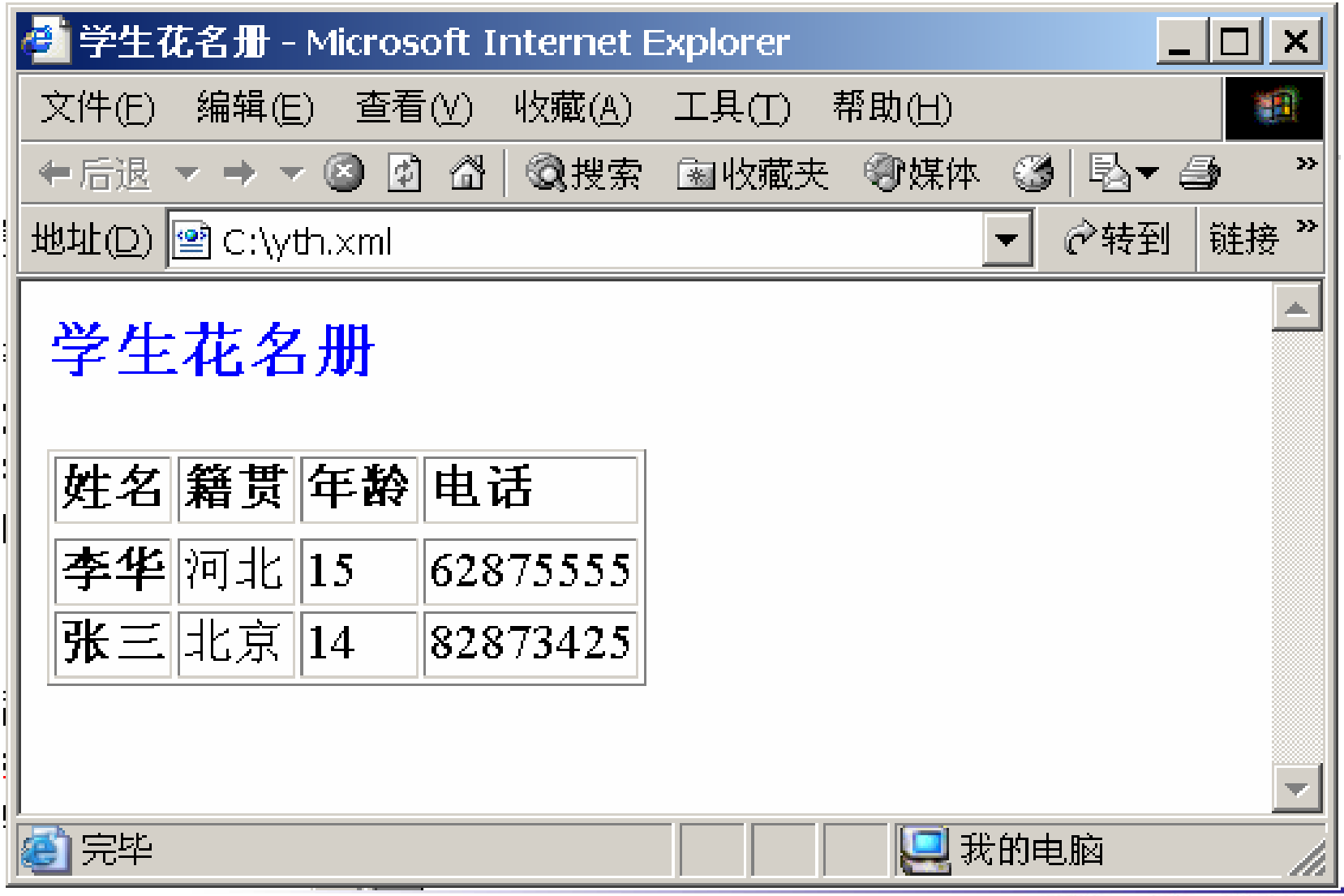
排序

获取节点值

循环

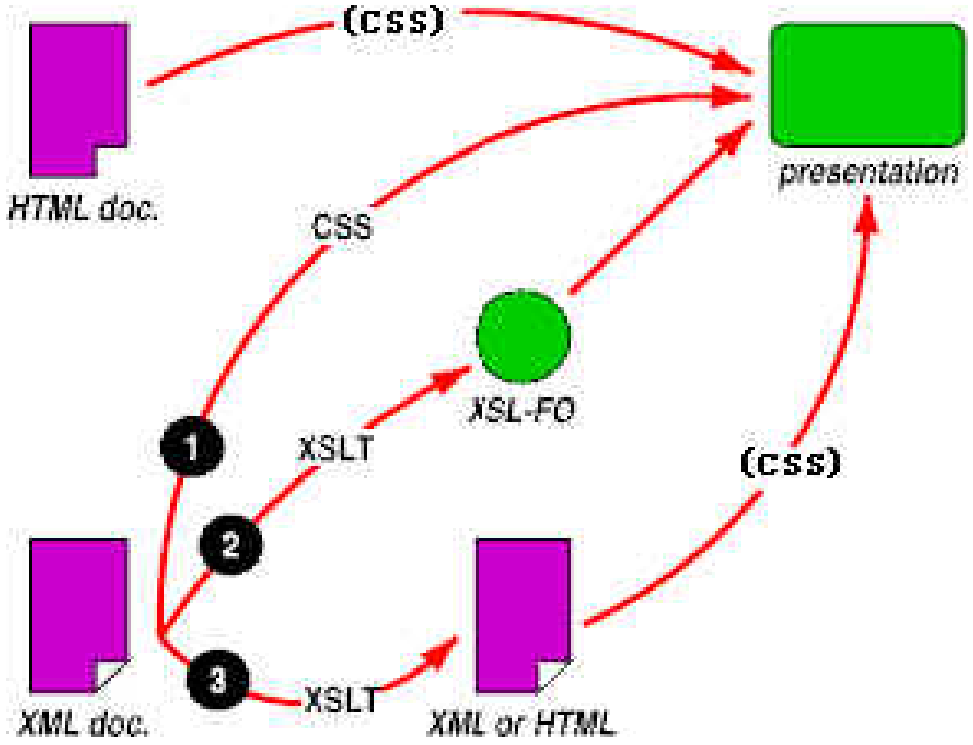
样式单尾

XML文档应用XSL后的显示结果：



一个XML文档的显示方式可以归纳为三种：

- 利用CSS显示
- 利用XSL转化为FO（格式对象）显示
- 以及利用XSL转化为HTML文档显示（这个HTML文档中可包含CSS样式）。



4.8 XML的应用

随着时间的推移，XML的应用范围和方式必将会极大地拓展。

（1）设计置标语言

作为元置标语言，XML为用户提供了定义本行业本领域的置标语言的最好工具。目前这一应用的成功例子比比皆是，例如化学领域的CML，数学领域的MathML，移动通信领域的WML等。

（2）数据交换

数据交换无疑是XML最令人激动的应用。数据交换的核心问题是信息的标准化，主要解决信息的可理解性问题，包括人和机器对信息的理解。而且，更重要的是机器对信息的识别，并能根据数据进行自动处理。XML的出现，为信息的标准化提供了有力的工具。

（3）智能信息检索

有了XML，智能检索代理能够理解接收到的数据，然后作出相应的反应。

（4）Web应用

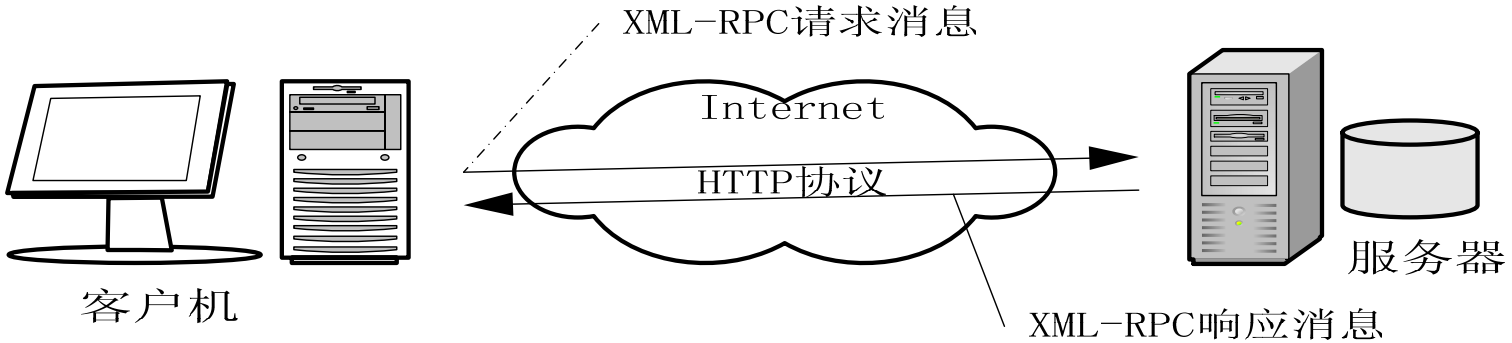
由于XML是由SGML特别为Web简化的，因此XML文档将成为Web资源的重要组成部分，XML使得搜索引擎更为智能和准确。XML还可以用于建立多层Web应用。 集成不同数据。

（5）网络出版

随着互联网的发展，网络已经成为一种新的媒体，人们在网络上发布各种信息，信息的发布形式和发布语言也多种多样，其中基于XML的显示技术和显示语言发挥了重要作用。比如eBook、eNewspaper等，就利用了XML的显示语言。

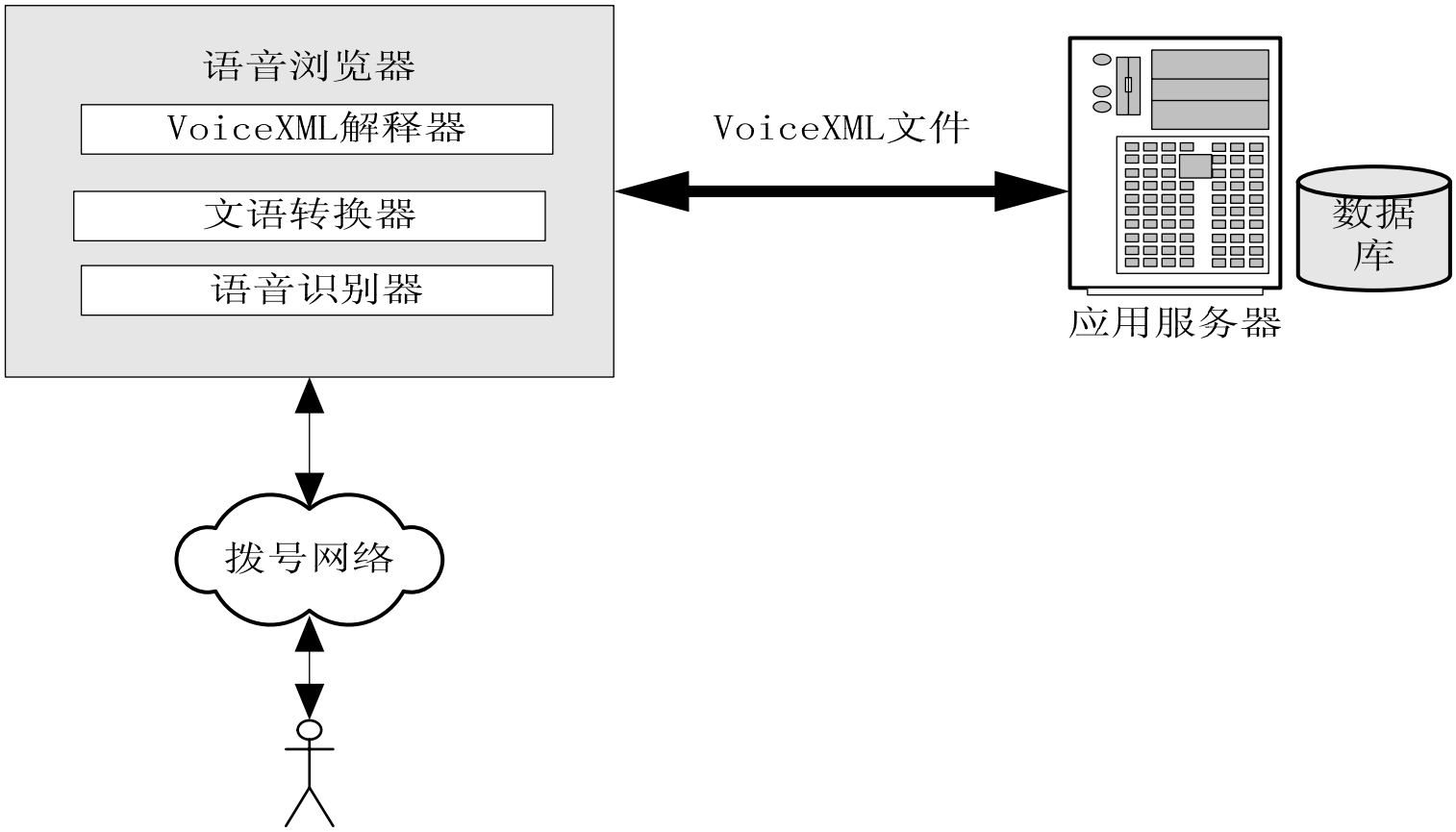
4.8.1 XML-RPC

```
POST/RPC2 HTTP/1.0
User-Agent:Ftrontier/5.1.2 (WinWT)
Host:Betty.userland.com
Content-Type:text/xml
Content-length:180
… …
<?xml version=" 1.0" >
<methodCall>
   <methodName>GetDeliverTime</methodName>
   <Params>
      …
   </Params>
</methodCall>
```




```
HTTP/1.0 200 OK
Connection: Close
Content-length:126
content-Type:text/xml
Date: Fri,17 Jul 2003 19:55:09 GMT
Server:Xidian University/5.1.2-Win2000
… …
<methodResponse>
   <Params>
   </Params>
</methodResponse>
```

4.8.2 移动环境中的VoiceXML

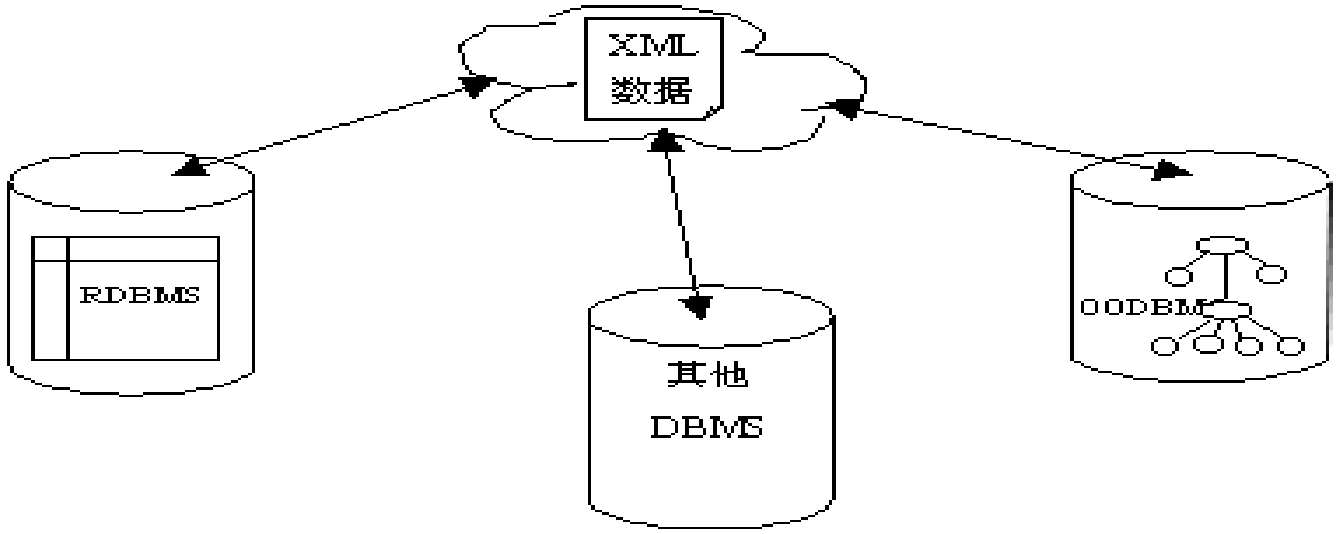


4.8.3 XML在数据库中的应用模式

XML在数据库中的应用模型需要借助三层架构来实现。在这种模式下，一般会有一个代理程序运行于中间层，通过它来访问数据库管理系统中的数据和输出XML文档。

微软在其Windows分布式Internet应用（即Windows DNA）架构中集成了XML技术。通过中间层的代理程序，可获取的数据来源可以不必局限于某台固定的数据库服务器，而可以是分布于企业内，甚至于遍及全球各地的数据库服务器。另外，借助于XML Schema，开发者就能更为精确地描述和交换数据，因而大大地提高这种应用的效率。

XML提供了一种连接关系数据库和面向对象数据库以及其他数据库管理系统之间的纽带。XML文档本身是一种由若干节点组成的结构，这种特点使得数据更适宜于用面向对象格式来存储，同时也有利于面向对象语言（C++、Java等）调用XML编程接口访问XML节点。关系数据库和面向对象数据库首先需要将数据从数据库中提取出来，经过转换为或直接以XML数据形式发布到网上（局域网或Internet网），然后相互交换数据，经应用层系统处理后再转存入库。



开发一个访问数据库的XML应用系统需要同时借助XML编程接口和数据库编程接口，前者用于对XML文档的解析、定位和查询，所需技术包括DOM和SAX；后者则是用于访问数据库，如数据库中数据的更新和检索等等，需要利用的技术有ODBC、JDBC、ADO等。

谢 谢 ！

第二次上机实验

时间：11月18日晚

内容：XML（包括结果显示和实验报告）

用XML＋Schema描述以下数据库表的结构和数据,并用XSLT将表分别以表格和记录两种形式转换成HTML显示在IE浏览器上。

- 数据库表结构

雇员表

	字段名称	数据类型	字段长度
*	员工编号	Auto	8
	姓名	Text	8
	身份证号码	Text	18
	出生日期	Date	
	联系电话	Text	11

● 表中的数据

（1） 表格形式

员工编号	姓名	身份证号码	出生日期	联系电话
A0100001	王鹏	390001198203150041	1982-03-15	88205923
B0100002	章文	390001198204050056	1982-04-05	88205927
D0200003	张国华	390001198307030061	1983-07-03	88205925
E0300004	刘帅	390001198103170082	1981-03-17	88205924
A0500005	李大伟	390001198309110037	1983-09-11	88205922

（2）记录形式

雇员表

员工编号：**A0100001**

姓名：王鹏

身份证号码：**390001198203150041**

出生日期：**1982-03-15**

联系电话：**88205923**

员工编号：**B0100002**

姓名：章文

身份证号码：**390001198204050056**

出生日期：**1982-04-05**

联系电话：**88205927**

●网络计算，未来发展的领域之地！