

# 高级编程技术

- 第1章 高级开发概述
- 第2章 组件化开发
- 第3章 XML技术
- 第4章 文件和数据库访问技术
- 第5章 线程和异步编程
- 第6章 分布式组件技术
- 第7章 XML Web Service
- 第8章 设计模式与构建
- 第9章 面向服务的架构（SOA）
- 第10章 开发智能设备应用程序

## 第3章 XML技术

- XML概述
- XML的格式
- 设计XML词汇表
- 命名空间
- XML解析器
- 使用DOM浏览XML
- 使用DOM创建新节点
- XML转换概述
- XSLT处理器
- 扩展XSLT样式表单
- 参考资源

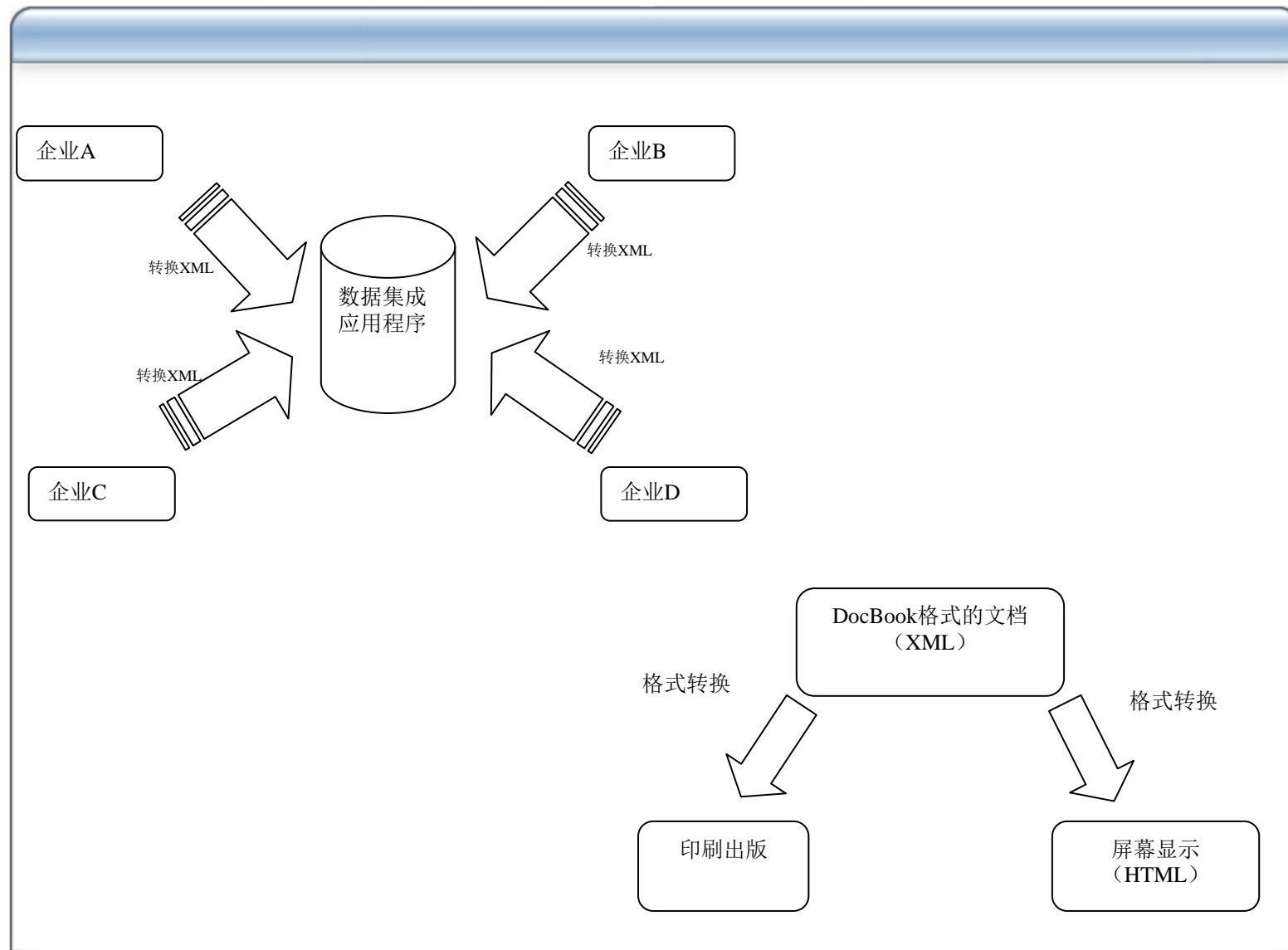
# XML概述

## 3.1 XML概述

- XML应用需求
- XML的实际应用

# XML应用

## 3.1 XML概述



# XML应用需求

## 3.1.1 XML应用需求

- 电子商务——数据共享，整合业务系统
- 关系数据库和非结构化的存储方式
- 标记语言：
  - HTML (HyperText Markup Language)
  - XML (Extensible Markup Language)

- XML的应用行业
  - 存储数据库
  - 结构化文档
  - 存储矢量图形等等
- XML使用者群体
  - 文档设计人员和科研人员
  - Web开发人员
  - 数据库和面向对象程序设计开发人员

## 第7章 XML基础

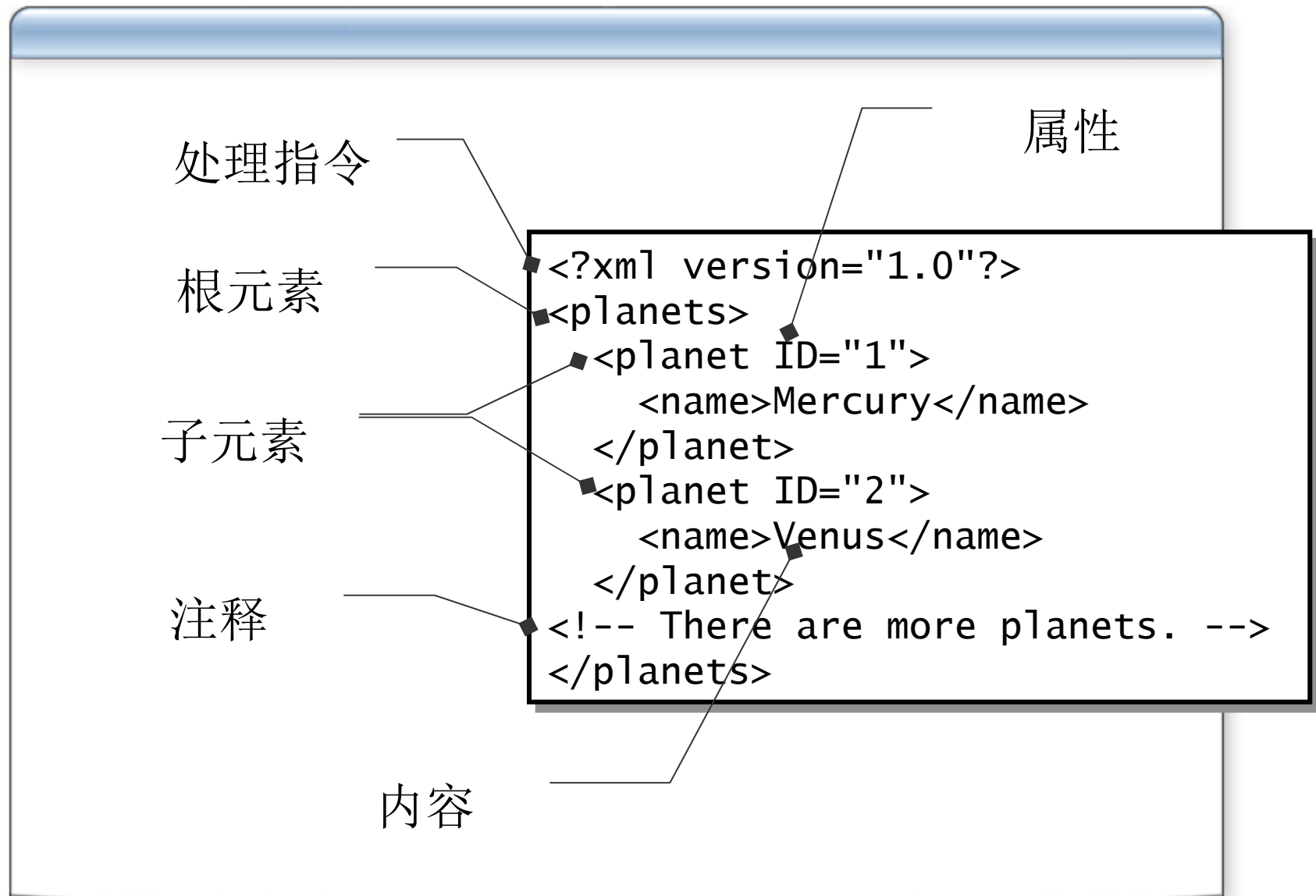
- XML概述
- XML的格式
- 设计XML词汇表
- 命名空间
- XML解析器
- 使用DOM浏览XML
- 使用DOM创建新节点
- XML转换概述
- XSLT处理器
- 扩展XSLT样式表单
- 参考资源

- XML文档的组成
- 格式正确的XML
- 有效的XML



# XML文档的组成

## 3.2.1 XML文档的组成



# 处理指令

## 3.2.1 XML文档的组成

- 与应用程序相关的处理指令

- 用于调用外部应用程序
- 可以有多个外部应用程序处理指令

```
<?MyDbApp SELECT * FROM orders ?>
```

应用程序名称

传递的命令

- 与XML本身处理器相关的处理指令 – XML声明

- 以XML关键字开头
- 在一个XML文件中只出现一次并且只能出现在文件的头部
- 用于声明XML的版本和采用的字符集

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

# 元素规则

## 3.2.1 XML文档的组成

- 名字中不能包含空格
- 名字不能以数字或标点符号开头
- 名字不能以任何大小写的xml开头
- 左尖括号（<）后不可以有空格
- 起始和结束标签的大小写必须一致
- XML文件中出现的第一个元素是根元素
- 根元素必须有完整的起始和结束标签
- 所有的子元素必须嵌套在一个根元素中
- 嵌套元素不可以相互重叠
- 子元素如果内容为空可以缩写标签

```
<Root>  
  <ChildA>  
    <ChildB>content  
  </ChildB>  
</ChildA>  
</Root>
```

```
<ElementName />
```

# 属性规则

## 3.2.1 XML文档的组成

- 属性可以在起始标签和处理指令之间声明
- 多个属性之间使用空格分隔
- 每条属性包含属性名和属性值两个部分
  - 一个元素中不能有重名的属性
  - 在同一个XML文件中不同元素中属性名可以重用
  - 属性名不可以包含空格
  - 赋值时可以使用单引号或双引号

```
<tree species ="Salix">Willow</tree>
```

属性名

属性值

# XML属性的常见错误

## 3.2.1 XML文档的组成

- `<Book ID=1 ID=2>1</Book>`

属性重名

- `<Author First Name="Tom">...</Author>`

属性名中间有空格

# 选择XML元素还是属性

## 3.2.1 XML文档的组成

- 元素用于封装数据，而属性通常用于提供有关元素的伴随信息，而不是封装原始数据本身
- 当信息需要简单类型的数据并且存在以下情况时，使用属性
  - 信息需要默认值或固定值
  - 信息需要的数据是现有元素的元数据
  - 如果 XML 文件的大小很重要，那么属性所需的字节数往往比元素要少

# 注释

## 3.2.1 XML文档的组成

- 注释不能嵌套在标签中

`<plants><!--native --></plants>`

`<plants<!--native -->></plants>`

- 只有在注释的开始和结尾可以使用双短横

`<!--native -frost tolerant -->`

`<!--native -- frost tolerant-->`

- 三短横只能在注释的开头使用而不能用在结尾处

`<!--10 Centigrade -->`

`<!--10 Centigrade --->`

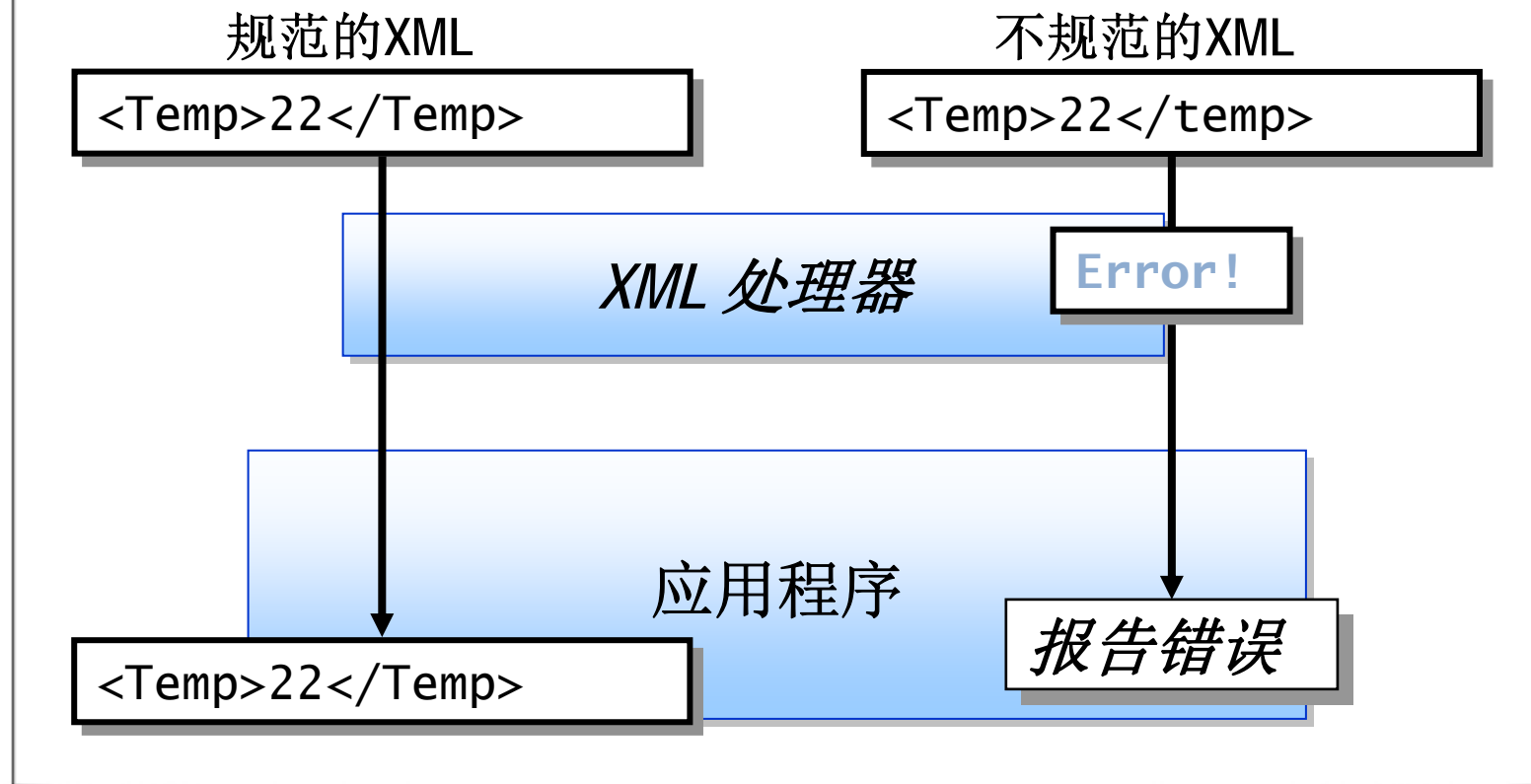
规范

不规范

# 格式正确的XML

## 3.2.2 格式正确的XML

- 规范的XML文件是严格按照W3C标准生成的
- 当遇到语法错误时XML处理器会停止工作





# 使用IE浏览器打开一个规范的XML文件

## 3.2.2 格式正确的XML

```
<?xml version="1.0" ?>
- <planets>
  - <planet ID="1">
    <name>Mercury</name>
  </planet>
  - <planet ID="2">
    <name>Venus</name>
  </planet>
  <!-- There are more planets. -->
</planets>
```

可以点击元素前面的符号展开或合并信息

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

# 一种不规范的XML文档

## 3.2.2 格式正确的XML

```
<?xml version="1.0"?>
<planets>
  <planet ID="1">
    <name>Mercury</name>
  </planet>
  <planet ID="2">
    <name>Venus</name>
  </Planet>
  <!-- There are more planets. -->
</planets>
```

把小写的“p”写  
为大写的“P”

# 不规范的XML文件不能正确显示

## 3.2.2 格式正确的XML

无法显示 XML 页。

使用 XSL 样式表无法查看 XML 输入。请更正错误然后单击 [刷新](#)按钮，或以后重试。

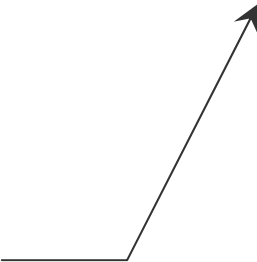
---

**结束标记 'categoryID' 与开始标记 'CategoryID' 不匹配。处理资源 'file:///D:/ProductCatalogData.xml' 时出错。第 4 行，位置：18**

```
<CategoryID>1</categoryID>
```

-----^

报告出错的具体位置



### XML架构

- 内置类型

类似于大多数编程语言中的数据类型，如 **string**、**integer** 和 **decimal** 等。大多数绑定在XML元素中的内容可以使用某种内置类型来处理，如日期和时间、URL 和各种专用字符串。

- 简单类型

用于定义不包含子元素和属性的元素，也可以定义属性。

- 复合类型

用于定义至少包含一个子元素或属性的元素。

# 有效的XML

## 3.2.3 有效的XML

- 简单类型的示例：

```
<xsd:simpleType name="SimpleType_SKU">  
  <xsd:restriction base="xsd:string">  
    <xsd:pattern value="\d{3}-[A-Z]{2}"/>  
  </xsd:restriction>  
</xsd:simpleType>
```

# 有效的XML

## 3.2.3 有效的XML

- 复杂类型的示例：

```
<xsd:complexType name="USAddress">  
  <xsd:sequence>  
    <xsd:element name="street" type="xsd:string"/>  
    <xsd:element name="city" type="xsd:string"/>  
    <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>
```

- XML验证

- 验证概述

- XML验证的定义：XML验证是将一个XML数据源和一系列预设条件进行比较的过程。

- 验证的原因：

- 过滤公司之间和应用程序之间共享的数据

- 防止数据库存放无效数据类型

- 标准化数据结构

- .NET Framework 对验证技术的支持

- .NET Framework 支持XSD架构，并且为了向后兼容，还支持DTD和XDR架构。

- 验证场景
  - 验证是在系统之间或公司之间进行数据传递的必要条件
- 需要验证的情况
  - 客户端
  - 商业伙伴之间
  - 应用程序内部
- 不需要验证的情况
  - 准备进行处理的XML文档中包含的XML词汇表、数据类型和数据格式都能可靠地满足系统需求



## 第7章 XML基础

- XML概述
- XML的格式
- 设计XML词汇表
- 命名空间
- XML解析器
- 使用DOM浏览XML
- 使用DOM创建新节点
- XML转换概述
- XSLT处理器
- 扩展XSLT样式表单
- 参考资源

# 设计XML词汇表

## 3.3 设计XML词汇表

- XML词汇表
- 词汇表创建原则
- 演示 词汇表比较

# XML词汇表

## 3.3.1 XML词汇表

- 问题提出：两个XML文件很难合并

```
<Inventory>  
  <Product SKU="1">Goo</Product>  
</Inventory>
```

+

```
<StockListing>  
  <Item Number="2">Glob</Item>  
</StockListing>
```

= ?

- 解决方案：相关的XML文件采用相同的结构、元素名称和大小写方案

```
<Inventory>  
  <Product SKU="1">Goo</Product>  
</Inventory>
```

+

```
<Inventory>  
  <Product SKU="2">Glob</Product>  
</Inventory>
```

=

```
<Inventory>  
  <Product SKU="1">Goo</Product>  
  <Product SKU="2">Glob</Product>  
</Inventory>
```

# 词汇表创建原则

## 3.3.2 词汇表创建原则

- 判断是否已有XML架构文件
  - 如果已有XML架构文件，不用再定义词汇表
- 使XML文件更加易读易懂
- 用首字母大写格式来定义元素和属性
  - 使用PASCAL命名惯例
- 尽量避免缩写
  - 缩写反而会使文件可读性降低

# 比较两种XML的文件

## 3.3.2 词汇表创建原则

```
<docprops>  
  <create>2002-07-24T22:39:55Z</create>  
  <lsavd>2002-08-10T16:13:07Z</lsavd>  
  <comp>Microsoft Corporation</comp>  
  <ver>10.2625</ver>  
</docprops>
```

请比较这两个文件

```
<DocumentProperties>  
  <Created>2002-07-24T22:39:55Z</Created>  
  <LastSaved>2002-08-10T16:13:07Z</LastSaved>  
  <Company>Microsoft Corporation</Company>  
  <Version>10.2625</Version>  
</DocumentProperties>
```

## 第7章 XML基础

- XML概述
- XML的格式
- 设计XML词汇表
- 命名空间
- XML解析器
- 使用DOM浏览XML
- 使用DOM创建新节点
- XML转换概述
- XSLT处理器
- 扩展XSLT样式表单
- 参考资源

- 命名空间
- 使用默认命名空间
- 使用显式命名空间
- 命名空间URI
- 命名空间URI的选择原则

# 命名空间

## 3.4.1 命名空间

- XML 命名空间将 XML 文档中的元素和属性名称与自定义和预定义的 URI 关联起来。为命名空间 URI 定义的前缀用来限定 XML 数据中的元素和属性的名称以实现此关联
- 使用命名空间可以有效防止在合并多个XML源文件时发生名称混淆



# 不使用命名空间时存在的问题

## 3.4.1 命名空间

- 合并前状态

```
<Order>
  <Employee>
    <Name>Jane Doe</Name>
    <Title>Developer</Title>
  </Employee>
  <Product>
    <Title>The Joshua Tree</Title>
    <Artist>U2</Artist>
  </Product>
</Order>
```

- 合并后状态

```
<Order>
  <Name>Jane Doe</Name>
  <Title>Developer</Title>
  <Title>The Joshua Tree</Title>
  <Artist>U2</Artist>
</Order>
```

# 使用默认命名空间

## 3.4.2 使用默认命名空间

- 缺省命名空间用URI关联所有的元素和子元素

```
<ElementName xmlns="URI">
```

- 使用缺省命名空间使不同的XML数据源合并后比较容易理解，这种状态下如果不修改XML的结构则不会发生名字混淆

```
<Order>  
  <Employee xmlns="http://hrweb">  
    <Name>Jane Doe</Name>  
    <Title>Developer</Title>  
  </Employee>  
  <Product xmlns="http://market">  
    <Title>The Joshua Tree</Title>  
    <Artist>U2</Artist>  
  </Product>  
</Order>
```

} 这些元素属于  
http://hrweb 命名空间

} 这些元素属于  
http://market 命名空间

# 默认命名空间的语法

## 3.4.2 使用默认命名空间

*<ElementName xmlns="URI">ElementContent</ElementName>*

语法组成	解释
ElementName	使用此命名空间的元素名称
<b>xmlns</b>	命名空间使用的关键字，通知处理器把后面的元素和此URI关联
URI	统一资源标示符，用于标识命名空间的惟一性，以避免名字混淆

# 使用显式命名空间

## 3.4.3 使用显式命名空间

- 使用显式命名空间使每个URI和一个前缀关联
- 使用前缀标识某个元素所属的命名空间

```
<ElementName xmlns:Prefix="http://contoso.msft/namespace_for_examples">  
  <Prefix:AnyElement>Some Data</Prefix:AnyElement>  
  <AnotherElement>More Data</AnotherElement>
```

# 显式命名空间的语法

## 3.4.3 使用显式命名空间

<ElementName xmlns:prefix="URI">

显式命名空间的定义

<prefix:ChildElement>ElementContent</prefix:ChildElement>  
</ElementName>

显式命名空间的使用

# 显式命名空间的使用

## 3.4.3 使用显式命名空间

- 使用显式命名空间把两个分属不同命名空间的XML源文件进行合并

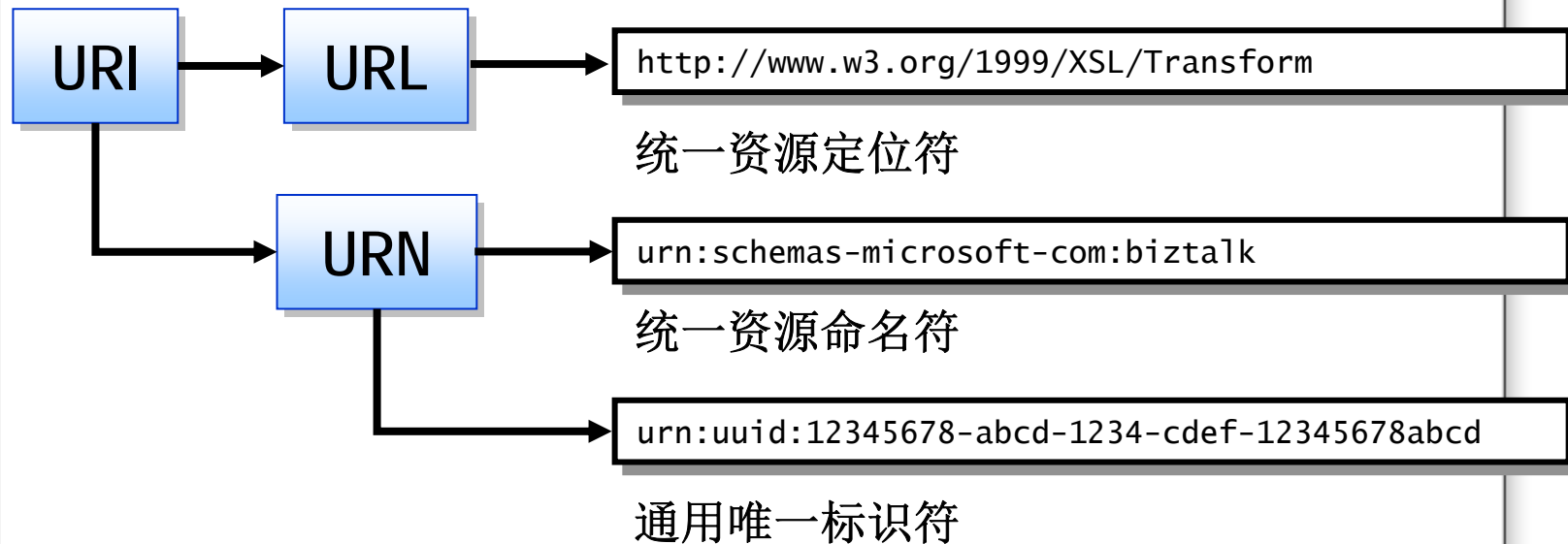
```
<Order xmlns:hr="http://hrweb" xmlns:mkt="http://market">  
  <hr:Name>Jane Doe</hr:Name>  
  <hr:Title>Developer</hr:Title>  
  <mkt:Title>The Joshua Tree</mkt:Title>  
  <mkt:Artist>U2</mkt:Artist>  
</Order>
```

# 命名空间URI

## 3.4.4 命名空间URI

- XML处理器只是把URI作为一个标记，并不校验其惟一性和独立性

URI可以是URL、URN 或 UUID



# 命名空间URI的选择原则

## 3.4.5 命名空间URI的选择原则

- 使用厂商自己的URI
  - 目的是防止与其他应用程序发生冲突
- 使用持久不变的URI
  - 目的是使应用程序保持一贯性
- 所有的URI定义应该以文档形式记录下来
  - 有助于记忆



## 第7章 XML基础

- XML概述
- XML的格式
- 设计XML词汇表
- 命名空间
- XML解析器
- 使用DOM浏览XML
- 使用DOM创建新节点
- XML转换概述
- XSLT处理器
- 扩展XSLT样式表单
- 参考资源

# XML解析器

## 3.5 XML解析器

- XML文档对象模型和简单API
- DOM节点与XML的对应
- DOM节点和相关的节点类型
- 支持DOM的类
- 从XML源中加载DOM
- 将DOM保存到文档

# 文档对象模型

## 3.5.1 XML文档对象模型定义

- 微软对DOM的支持
  - Microsoft® XML Core Services 4 (MSXML4)
  - Microsoft Visual Studio® .NET
  - Microsoft .NET Framework
- DOM的主要功能
- 不适合使用DOM的场景
  - 读取XML文档
  - 查询XML文档
  - 转换XML格式

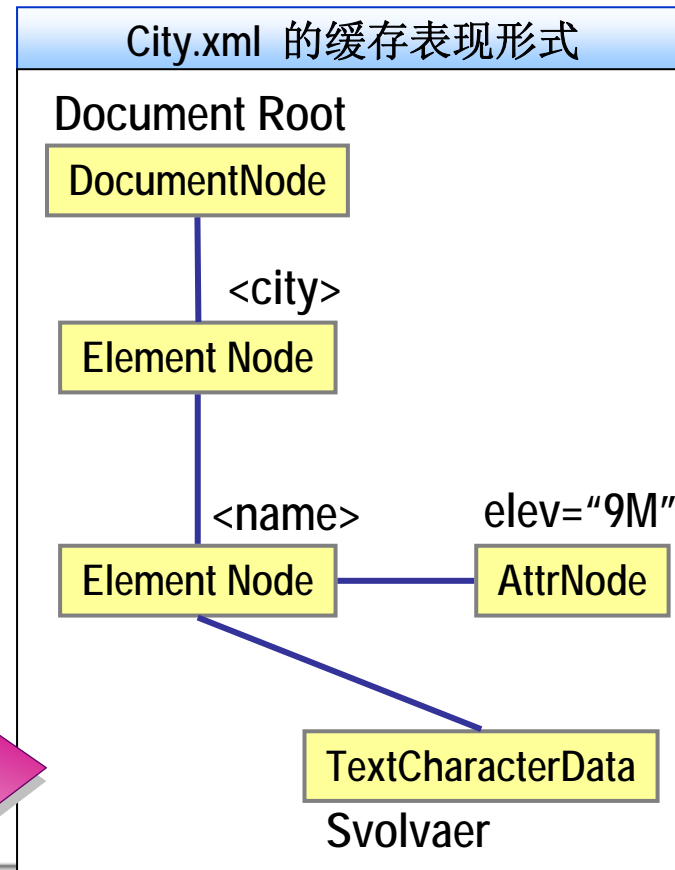
# XML文档对象模型与简单API

## 3.5.1 XML文档对象模型与简单API

- DOM是W3C规定的XML编程接口
- DOM在缓存中以树状节点的形式描述XML数据源
- 使用DOM可以
  - 定位和浏览
  - 添加和删除内容

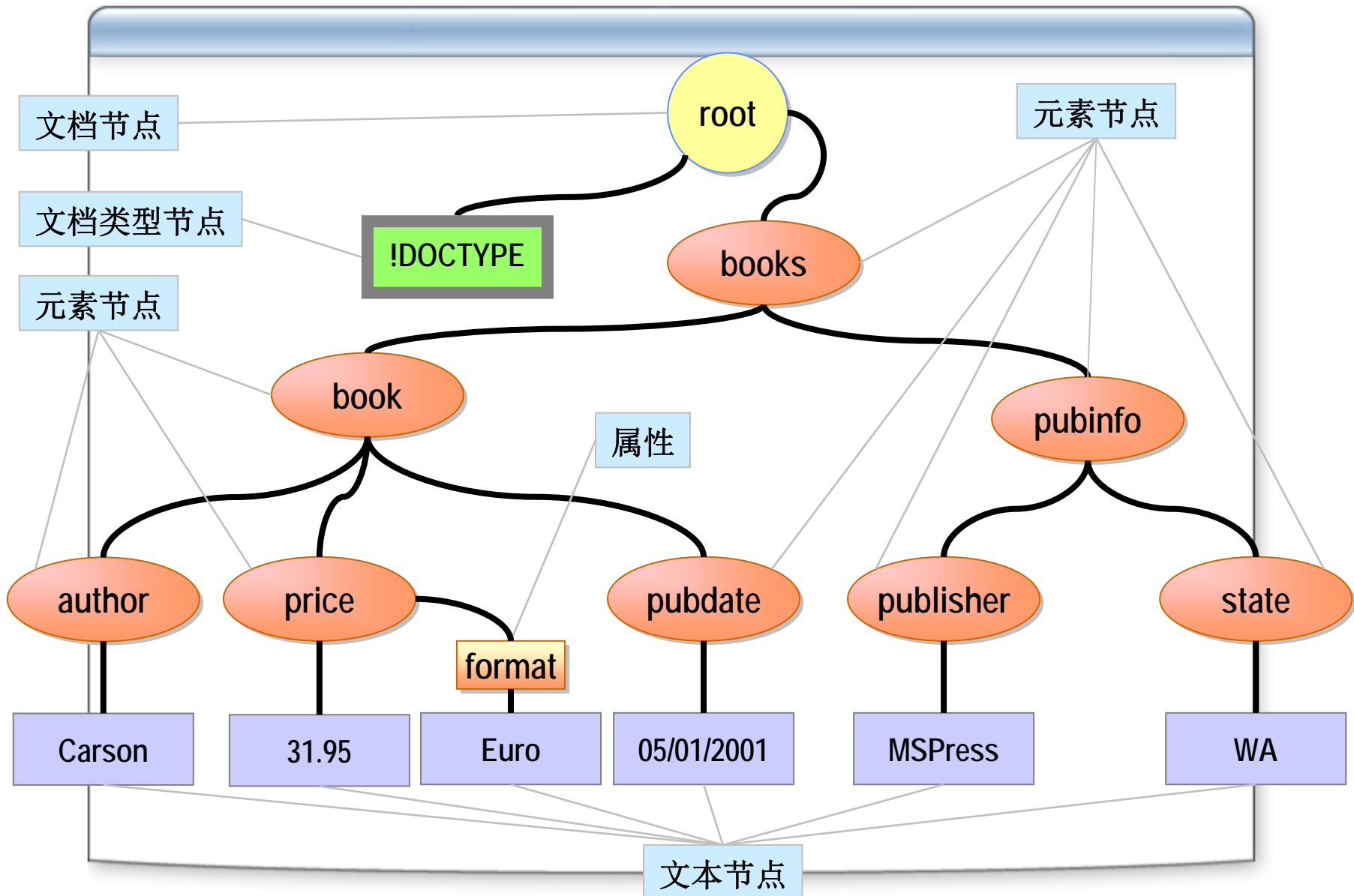
City.xml

```
<city>
<name
elev="9M">Svolvaer</name>
</city>
```



# DOM节点与XML的对应

## 3.5.2 DOM节点与XML的对应



# DOM节点的特性

## 3.5.2 DOM节点与XML的对应

- DOM节点通常都有惟一父节点
- DOM节点通常可以拥有多个子节点
- 某些DOM节点不能拥有子节点
- XML的属性被视为DOM节点的属性

# DOM节点和相关的节点类型

## 3.5.3 DOM节点和相关的节点类型

W3C 定义的DOM节点类型	对应的 .NET DOM 节点类型
Document	XmlDocument
DocumentFragment	XmlDocumentFragment
DocumentType	XmlDocumentType
EntityReference	XmlEntityReference
Element	XmlElement
Attr	XmlAttribute
ProcessingInstruction	XmlProcessingInstruction
Comment	XmlComment
Text	XmlText
CDATASection	XmlCDATASection
Entity	XmlEntity
Notation	XmlNotation

# 非W3C标准的.NET节点类

## 3.5.3 DOM节点和相关的节点类型

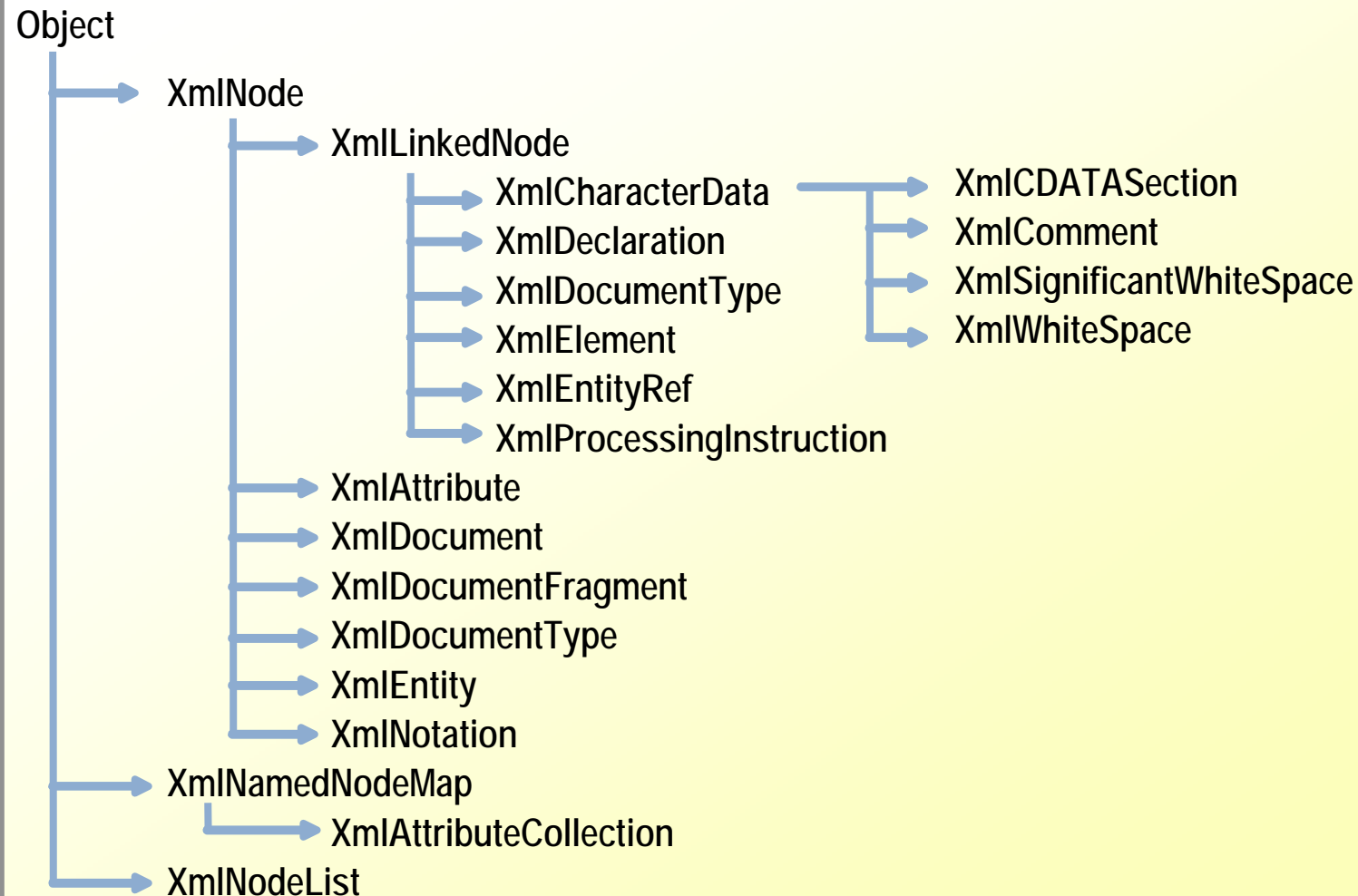
.NET节点类	功能描述
XmlDeclaration	表示声明节点 <?xml version="1.0"...>
XmlSignificant Whitespace	表示有效空白（混合内容模式中标记之间的空白）
XmlWhiteSpace	表示元素内容中的空白
EndElement	当 XmlReader 到达元素的末尾时返回。 示例 XML: </item>
EndEntry	由于调用 ResolveEntity 而在 XmlReader 到达实体替换的末尾时返回



# 支持DOM的类

## 3.5.4 支持DOM的类

### .NET Framework中DOM类的继承树



# DOM类继承树中重要类

## 3.5.4 支持DOM的类

- XmlNode
- XmlDocument
- XmlNodeList
- XmlNamedNodeMap

# 从XML源中加载DOM

## 3.5.5 从XML源中加载DOM

- 使用XmlDocument对象加载DOM
- 具体方法由加载对象的类型决定

加载的 XML数据源类型	调用的方法
String	LoadXml()
Stream File XmlReader TextReader	Load()

# XmlDocument对象加载数据的过程

## 3.5.5 把XML数据源加载到DOM

- 初始化XmlDocument对象

```
XmlDocument doc = new XmlDocument();
```

- 调用Load()方法加载文件、流、XmlReader对象或TextReader对象中的XML数据

```
doc.Load("C:\\BookData.xml");
```

- 调用LoadXml()方法加载字符串中的XML数据

```
doc.LoadXml("<book genre='novel' ISBN='1-861001-57-5'>"  
            + "<title>Pride And Prejudice</title>" + "</book>");
```

# 将DOM保存到文档

## 3.5.6 将DOM保存到文档

- 可以调用Save()方法把XmlDocument对象保存到下列类型的XML文件中
  - file
  - Stream
  - TextWriter
  - XmlWriter
- Save() 方法的语法

```
XmlDocument.Save(filename | Stream | TextWriter |  
XmlWriter)
```

## 第7章 XML基础

- XML概述
- XML的格式
- 设计XML词汇表
- 命名空间
- XML解析器
- 使用DOM浏览XML
- 使用DOM创建新节点
- XML转换概述
- XSLT处理器
- 扩展XSLT样式表单
- 参考资源

# 使用DOM浏览XML

## 3.6 使用DOM浏览XML

- Xml 节点
- Xml 节点的属性
- 引用单个节点
- 解析内存中的 Xml 文档对象
- Xml 有序节点列表
- Xml 无序节点集

# Xml节点

## 3.6.1 Xml节点

- 用于根据节点特性操作XML文档
- 作为浏览文档时的节点指针
- 子类
  - XmlDocument
  - XmlDocumentFragment
  - XmlEntity
  - XmlLinkedNode
  - XmlAttribute
  - XmlNotation



# XmlNode类代码示例

## 3.6.1 Xml节点

```
XmlDocument doc = new XmlDocument();//创建DOM对象
doc.Load("books.xml");//加载XML数据
//显示所有的书
XmlNode root = doc.DocumentElement;//返回根节点
IEnumerator NodePointer = root.GetEnumerator();//获得枚举指针
XmlNode book;
while (NodePointer.MoveNext())//遍历节点
{
    book = (XmlNode) NodePointer.Current;
    MessageBox.Show(book.OuterXml);
}
```

- 使用XmlNode类的原因

在创建了XmlDocument对象并为其加载了XML数据之后，就可以解析和遍历这个XML数据以及增加、删除或修改节点。

通常可以创建并使用一个新的XmlNode对象，这将有助于在XmlDocument对象中浏览、创建或者修改节点。在XmlDocument对象结构中，XmlNode对象的功能类似于指向节点的指针（pointer）或游标（cursor）。由于XmlDocument中的很多节点继承于XmlNode类，因此可以使用XmlNode对象来引用XmlDocument对象中所有的这些节点。

在选择了XmlDocument对象中的某个节点之后，就可以通过查看属性来提取信息，以及修改内容、插入新节点或删除现有的节点。

# 使用Xml节点的属性

## 3.6.2 使用Xml节点的属性

- XmlElement和XmlAttribute对象也是节点
- 使用属性和方法访问XML元素内容

```
<?xml version="1.0"?>  
<book isbn="123456789">  
    <title>XML.NET</title>  
    <price>19.99</price>  
</book>
```

```
XmlNode book = doc.FirstChild;  
XmlNode priceNode = book.ChildNodes[1];  
XmlNode isbnNode = book.GetAttributeNode("isbn");  
string price = priceNode.FirstChild.Value;  
string isbn = isbnNode.Value;
```

# XmlNode类获取信息的属性

## 3.6.2 使用Xml节点的属性

属性	功能描述
<b>InnerText</b>	节点及其所有子节点的串联值，不包含XML标签
<b>InnerXml</b>	仅代表该节点的子节点的标记
<b>OuterXml</b>	此节点及其所有子节点的标记
<b>Name</b>	节点的限定名
<b>NodeType</b>	节点的类型，例如文档、元素、属性、注释等
<b>Value</b>	以字符串类型返回节点的内容

# XmlNode类控制枚举指针的属性

## 3.6.2 使用Xml节点的属性

属性	功能描述
<b>FirstChild</b>	节点的第一个子节点
<b>LastChild</b>	节点的最后一个子节点
<b>HasChildNodes</b>	判断当前节点是否有子节点
<b>NextSibling</b>	紧接在当前节点之后的同级节点
<b>PreviousSibling</b>	紧接在该节点之前的同级节点
<b>ParentNode</b>	其直接父节点

# 使用Xml节点的属性

## 3.6.2 使用Xml节点的属性

- 返回XmlDocument对象的XML根元素

加载好XML文档之后，就可以查询缓存中的XML数据信息的属性。**XmlDocument**类继承于XML节点类（**XmlNode**），并通过增加一些方法和属性扩展了这个类。

可以使用**XmlDocument**对象的**DocumentElement**属性为文档返回根元素。这个属性对于浏览内存中的**XmlDocument**对象是很有用处的。找到**XmlDocument**对象的根元素节点之后，就可以使用其它的属性和方法来返回有关**XmlDocument**对象的信息，并浏览内存中文档的节点。

# 引用单个节点

## 3.6.3 引用单个节点

- 使用下列方法引用节点
  - 调用SelectSingleNode()
  - 使用XMLNode类的浏览属性

# 解析内存中的Xml文档对象

## 3.6.4 解析内存中的Xml文档对象

- 使用XmlElement类
- 查看元素属性

```
public virtual string GetAttribute(string);
```

- 遍历XmlDocument对象中的元素



# 查看元素属性

## 3.6.4 解析内存中的XmlDocument对象

```
XmlDocument doc = new XmlDocument();  
doc.LoadXml("<book genre='novel' ISBN='1-861001-57-5'>" +  
            "<title>Pride And Prejudice</title>" + "</book>");  
XmlElement root = doc.DocumentElement;//创建对象  
  
if (root.HasAttribute("genre")){//检查是否有genre属性  
    String genre = root.GetAttribute("genre");//获取genre属性的值  
    MessageBox.Show(genre);  
}
```

# 遍历XmlDocument对象中的元素

## 3.6.4 解析内存中的XmlDocument对象

```
XmlDocument doc = new XmlDocument();
doc.Load("books.xml");
XmlElement root = doc.DocumentElement;
IEnumerator NodePointer = root.GetEnumerator(); //获得枚举指针
XmlElement book;
while (NodePointer.MoveNext()) { //遍历节点
    book = (XmlElement) NodePointer.Current;
    MessageBox.Show(book.InnerText);
    if (book.HasAttribute("genre")) { //检查属性
        String genre = book.GetAttribute("genre");
        MessageBox.Show(genre);
    }
}
```

# Xml有序节点列表

## 3.6.5 Xml有序节点列表

- **使用XmlNodeList的原因**

可以使用XmlNodeList来包含和处理一个有序节点集。在将节点加载到XmlNodeList对象时，需要对节点进行选择 and 过滤。就这一点而言，XmlNodeList类似于Transact-SQL查询中的一个返回结果集。

在加载完节点之后，便可以遍历这个列表并解析节点以获取信息。

- **XmlNodeList的数据源**

从XmlDocument对象中查找和过滤节点，并将所选的节点填充到XmlNodeList中。根据用来包含未过滤节点的节点对象，可以使用表 3-13 中这几种方法和属性来生成XmlNodeList。

# Xml有序节点列表

## 3.6.5 Xml有序节点列表

- **使用ChildNodes来填充XmlNodeList**

使用任一XmlNode类型的ChildNodes属性，都可以返回XmlNode对象的所有子节点。如果使XmlNode对象指向文档的根节点，就可以使用这种方式返回XmlDocument对象的所有节点。

- **使用SelectNodes()来填充XmlNodeList**

使用任一XmlNode类型的SelectNodes()方法，可以从XmlDocument对象中过滤信息。该方法是一种用来查找信息的特定方法。SelectNodes()方法使用XPath查询来查找节点，并且将节点加载到XmlNodeList。可以使用XPath表达式来查找XmlDocument对象中所有基于元素和属性值的节点。

# SelectNodes()方法代码示例

## 3.6.5 XmlNodeList类

```
XmlDocument doc = new XmlDocument();  
doc.Load("booksort.xml");  
XmlNodeList nodeList;//定义变量  
XmlElement root = doc.DocumentElement;  
nodeList = root.SelectNodes("/bookstore/book/@bk:ISBN");  
//获得节点列表  
    foreach (XmlNode isbn in nodeList){//遍历节点列表  
        MessageBox.Show(isbn.Value);  
    }
```

# Xml有序节点列表

## 3.6.5 Xml有序节点列表

- 使用GetElementsByTagName()填充XmlNodeList

使用GetElementsByTagName()方法，可以从Xml-Document或XmlElement对象中查找指定名称的所有元素及其子元素。这个方法以要获取的元素名称作为它的字符串参数。

下面的示例使用GetElementsByTagName()方法将文档中所有book节点中的title元素填充到XmlNode-List中。

# GetElementsByTagName()方法代码示例

3.6.5 XmlNodeList类

```
XmlDocument doc = new XmlDocument();  
doc.Load("books.xml");  
XmlNodeList elemList = doc.GetElementsByTagName("title");  
    //获得节点列表  
    for (int i=0; i < elemList.Count; i++) //遍历节点  
    {  
        MessageBox.Show(elemList[i].InnerXml);  
    }
```

# Xml无序节点集

## 3.6.6 Xml无序节点集

- **使用XmlNamedNodeMap的原因**

通常，可以使用XmlNamedNodeMap对象来保存一个元素节点的属性集合。

在使用继承自XmlNamedNodeMap的XmlAttributeCollection来检索属性时，可以通过属性的XML名称来获得属性的节点，这样处理某一元素节点的属性集合就会很方便。

- **使用XmlNamedNodeMap数据源**

下面的示例说明了如何使用XmlNamedNodeMap根据名称来选择属性。



# Xml无序节点集

## 3.6.6 Xml无序节点集

```
//加载文档
XmlDocument doc = new XmlDocument();
doc.LoadXml("<book genre='novel' publicationdate='1997'>" +
            "<title>Pride And Prejudice</title>" +
            "</book>");
XmlAttributeCollection attrColl =
            doc.DocumentElement.Attributes;

//修改genre属性的值
XmlAttribute attr = (XmlAttribute)attrColl.GetNamedItem("genre");
attr.Value = "fiction";

//显示已修改的XML
MessageBox.Show(doc.OuterXml);
```

## 第7章 XML基础

- XML概述
- XML的格式
- 设计XML词汇表
- 命名空间
- XML解析器
- 使用DOM浏览XML
- 使用DOM创建新节点
- XML转换概述
- XSLT处理器
- 扩展XSLT样式表单
- 参考资源

# 使用DOM创建新节点

## 3.7 使用DOM创建新节点

- 增加节点
- 创建元素节点
- 为元素节点设置属性

# 为XmlDocument对象增加节点

## 3.7.1 增加节点

- 在XmlDocument对象中添加新节点的步骤
  1. 从XML数据源创建和加载XmlDocument对象
  2. 在原始文档中定位新节点的插入位置
  3. 创建新节点
  4. 把新节点添加到XmlDocument对象中

# 添加节点时调用的方法

## 3.7.1 增加节点

方法	功能描述
<b>XmlNode.AppendChild</b>	添加到当前节点的子节点列表的末尾
<b>XmlNode.InsertBefore</b>	添加到紧接着插入指定的引用节点之前
<b>XmlNode.InsertAfter</b>	添加到紧接着插入指定的引用节点之后

# 创建元素节点

## 3.7.2 创建元素节点

- 调用XmlDocument.CreateElement()方法
- 对于简单元素，直接创建文本的内容

```
XmlDocument doc = new XmlDocument();  
XmlNode bookNode = doc.CreateElement("book");  
  
XmlNode titleNode = doc.CreateElement("title");  
XmlNode title = doc.CreateTextNode("XML is Cool");  
titleNode.AppendChild(title);  
  
XmlNode priceNode = doc.CreateElement("price");  
XmlNode price = doc.CreateTextNode("19.99");  
priceNode.AppendChild(price);  
  
bookNode.AppendChild(titleNode);  
bookNode.AppendChild(priceNode);
```

# 使用InnerText属性的代码示例

## 3.7.2 创建元素节点

```
XmlDocument doc = new XmlDocument();
doc.LoadXml("<book genre='novel' ISBN='1-861001-57-5'>" +
            "<title>Pride And Prejudice</title>" + "</book>");
//选定插入点位置
XmlNode root = doc.DocumentElement;
//创建新节点，设置InnerText属性值
XmlElement elem = doc.CreateElement("price");
elem.InnerText="19.95";
//把节点加入到文档.
root.AppendChild(elem);
//保存并显示
doc.Save(Console.Out);
```

# 为元素节点设置属性

## 3.7.3 为元素节点设置属性

- 在元素节点中设置属性需要调用的方法

XmlElement.SetAttribute()

```
bookNode.SetAttribute("sku", "XYZ-123");
```

- CreateAttribute和SetAttributeNode方法



## CreateAttribute() 和 SetAttributeNode() 方法

### 3.7.3 为元素节点设置属性

```
XmlDocument doc = new XmlDocument();
doc.LoadXml("<book genre='novel' ISBN='1-861001-57-5'>" +
            "<title>Pride And Prejudice</title>" + "</book>");
//创建属性对象
XmlAttribute attr = doc.CreateAttribute("publisher");
//设置属性值
attr.Value = "WorldWide Publishing";
//把属性添加到DOM对象中
doc.DocumentElement.SetAttributeNode(attr);
//保存并显示
doc.Save(Console.Out);
```

## 第7章 XML基础

- XML概述
- XML的格式
- 设计XML词汇表
- 命名空间
- XML解析器
- 使用DOM浏览XML
- 使用DOM创建新节点
- XML转换概述
- XSLT处理器
- 扩展XSLT样式表单
- 参考资源

- XSLT
- XSLT样式表的组成部分
- 转换XML文档的原因
- XSLT结构

- 模板
  - 模板是控制转换或格式化的规则
  - 样式表单使用XML文档作为输入，XSLT处理器将该文档视为一棵层次节点树
  - 样式表单模板定义了不同的树节点应该如何转换，也定义了输出是根据源XML的原始文本来创建，还是根据源XML来编程确定

- XSLT模板的规则

XSLT 模板规则由以下几部分组成：

- 一个XPath（XML Path Language，XML路径语言）表达式，决定了源树中要转换的节点
- 一个唯一的名称，使模板可以象函数一样被调用
- 直接复制到结果文档的原始文本
- 由XSLT指令生成的内容

- 示例

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

- **XSLT样式表单命名空间**

有些样式表单，在命名空间声明中使用了过时的XSL草案的URI（Uniform Resource Identifier，统一资源标志）。该命名空间提示XML处理器，通过使用转换XML的处理器中的特殊功能来处理所有前缀为xsl:的元素。

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

- **支持XSL和XSLT的MSXML处理器**

MSXML 3.0（Microsoft XML Parser，Microsoft XML解析器）仍然认可XSL命名空间是表示样式表单的一种有效方法。

然而，最新版本的Microsoft XML解析器——MSXML 4.0（Microsoft XML Core Services），不再认可XSL URI是表示样式表单的一种有效方法。为新开发项目而生成的样式表单应该使用XSLT URI。

# XSLT样式表单的组成部分

## 3.8.2 XSLT样式表单的组成部分

- 识别XSLT样式表单

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

- 模板匹配

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

- 应用格式转换

```
<xsl:apply-templates select="">
```

- 创建输出

```
<xsl:value-of select="">
```

```
<xsl:copy-of select="">
```

# 转换格式前的XML文档

## 3.8.2 XSLT样式表单的组成部分

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="emp1.xsl"?>
<employees xmlns:au="http://www.demotest.com">
  <employee>
    <name>Adam Stein</name>
    <salary>23500</salary>
    <jobtitle>Programmer</jobtitle>
    <region>Redmond</region>
  </employee>
  .....
  <employee>
    <name>Paula Thurman</name>
    <au:salary>34500</au:salary>
    <jobtitle>Programmer</jobtitle>
    <region>Sydney</region>
  </employee>
  .....
</employees>
```

命名空间不同，希望  
以不同的形式显示在  
转换后的HTML页面中



# 关联的样式表单

## 3.8.2 XSLT样式表单的组成部分

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:au="http://www.demotest.com" version="1.0">
  <xsl:template match="/">
    <HTML><BODY>
      <H1>Salaries Report</H1><xsl:apply-templates/>
    </BODY> </HTML>
  </xsl:template>
  <xsl:template match="employees">
    <xsl:apply-templates select="employee[salary]"/>
    <xsl:apply-templates select="employee[au:salary]"/>
  </xsl:template>
  <xsl:template match="employee[salary]">
    <p><xsl:value-of select="name"/>, <xsl:value-of select="jobtitle"/>,
      <xsl:value-of select="region"/>, <xsl:value-of select="salary"/> USD (US dollars)</p>
  </xsl:template>
  <xsl:template match="employee[au:salary]">
    <p><xsl:value-of select="name"/>, <xsl:value-of select="jobtitle"/>,
      <xsl:value-of select="region"/>, <xsl:value-of select="au:salary"/> AUD (Australian dollars)</p>
  </xsl:template>
</xsl:stylesheet>
```

逐级调用

# IE中的输出结果

## 3.8.2 XSLT样式表单的组成部分

### Salaries Report

Adam Stein, Programmer, Redmond, 23500 USD (US dollars)

Susan Tjarnberg, Tester, Minneapolis, 51000 USD (US dollars)

Catherine Turner, System Architect, Dallas, 45000 USD (US dollars)

Wendy Vasse, Project Manager, Washington D.C., 72000 USD (US dollars)

Paula Thurman, Programmer, Sydney, 34500 AUD (Australian dollars)

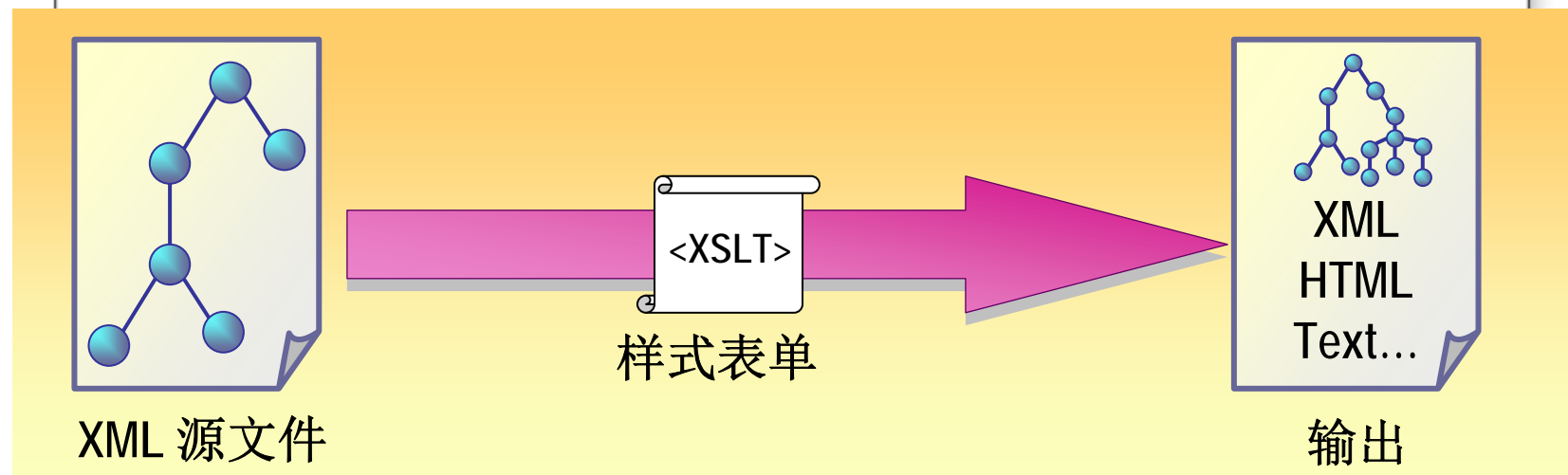
Richard Marshall, Programmer, Canberra, 30000 AUD (Australian dollars)

显示不  
同货币

# 转换XML文档的原因

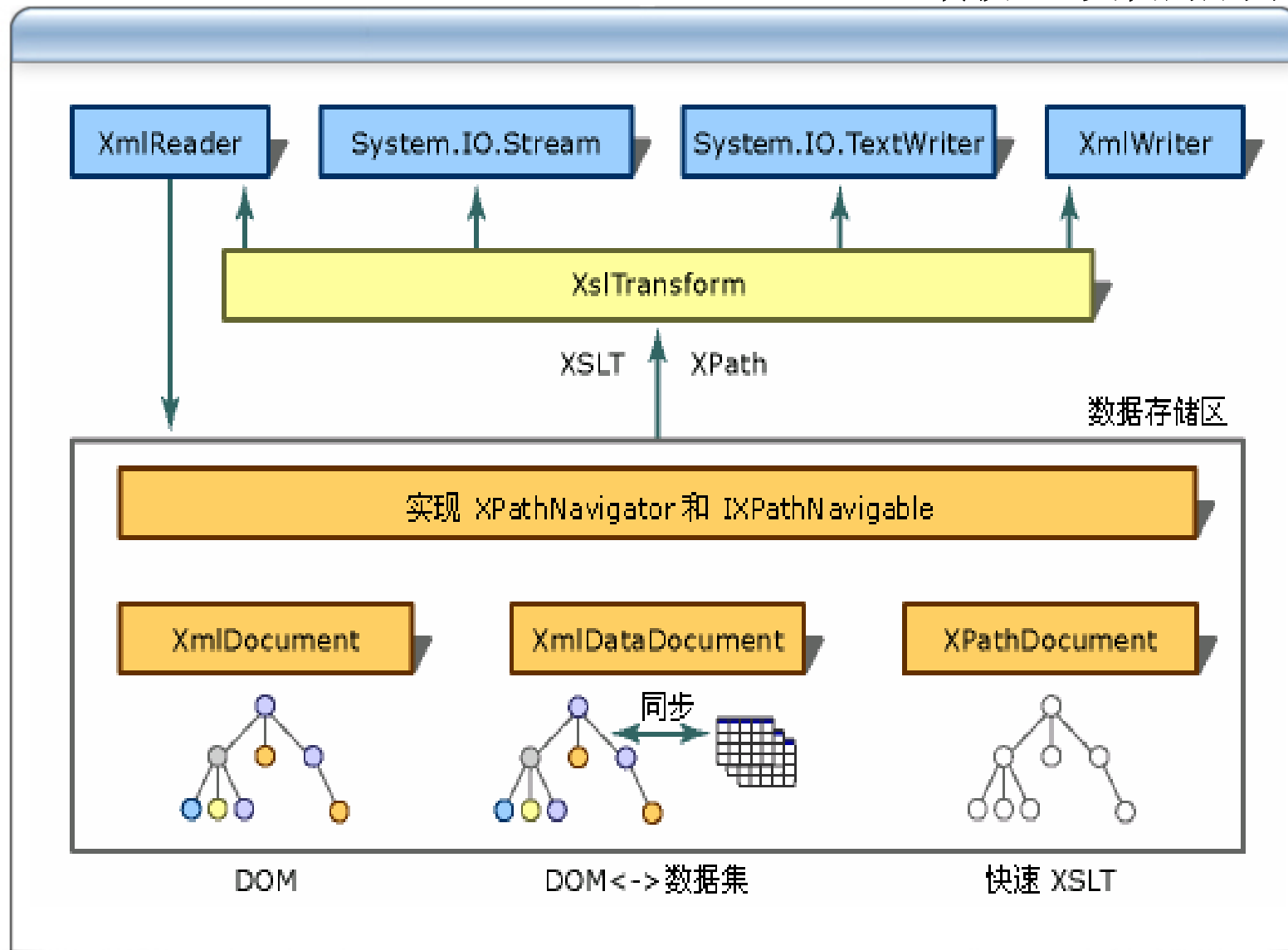
## 3.8.3 转换XML文档的原因

- XSLT可以转换和筛选XML文档
  - XML → XML
  - XML → HTML
  - XML → 任意文本（如PDF、CSV等）



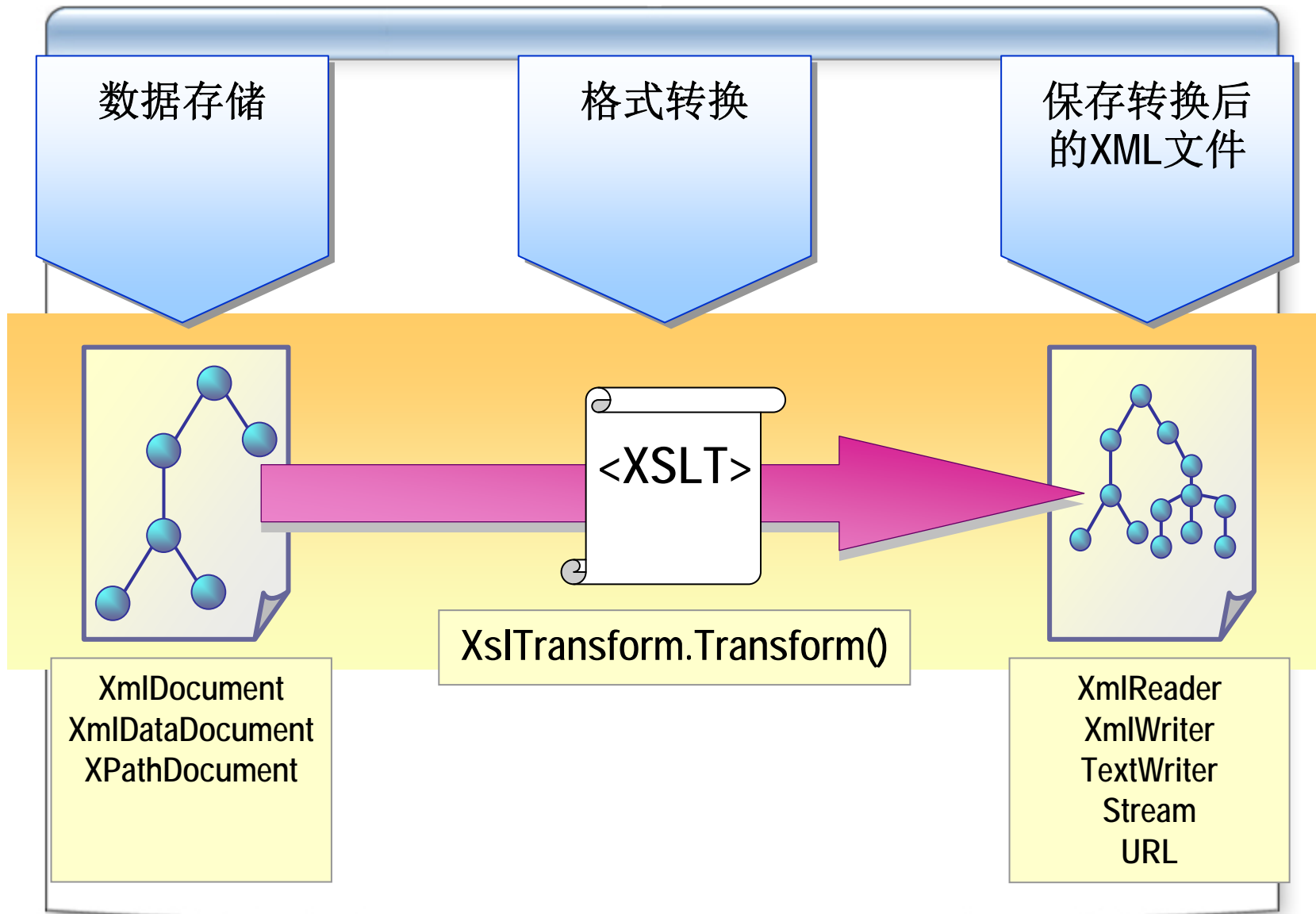
# XSLT转换功能在应用程序中的结构

## 3.8.3 转换XML文档的原因



# XSLT结构

## 3.8.4 XSLT结构



## 第7章 XML基础

- XML概述
- XML的格式
- 设计XML词汇表
- 命名空间
- XML解析器
- 使用DOM浏览XML
- 使用DOM创建新节点
- XML转换概述
- XSLT处理器
- 扩展XSLT样式表单
- 参考资源

- XSLT处理器的实现
- 创建XSLT处理器对象
- 应用XSLT样式表单
- 在Web应用程序中显示XML数据

# XSLT处理器

## 3.9.1 XSLT处理器

- 使用XSLT转换XML格式
- 属于System.Xml.Xsl命名空间
- 使用XslTransform对象转换XML数据格式的步骤
  1. 创建XslTransform对象
  2. 调用Load()方法加载XSLT样式表单对象
  3. 调用Transform()方法转换缓存中数据的格式



# 用XslTransform对象转换格式的过程

## 3.9.1 XSLT处理器

```
// 创建并加载XML文档对象
XPathDocument InputDoc = new XPathDocument("catalog.xml");
//创建XslTransform对象
XslTransform Transformer = new XslTransform();
// 加载样式表单到XslTransform对象
Transformer.Load("DisplayProducts.xsl");
// 创建输出流以便保存结果
StreamWriter SW = new StreamWriter("products.htm");
// 转换数据
Transformer.Transform(InputDoc, null, SW);
```

# 创建XSLT处理器对象

## 3.9.2 创建XSLT处理器对象

- 新建XslTransform对象
- 调用Load()方法把样式表单加载到XslTransform对象中

```
// C#  
XslTransform ExportStyle = new XslTransform();  
// 将样式表单加载到对象中  
ExportStyle.Load("http://myserver/ExportProduct.xsl");
```

```
' Visual Basic  
Dim ExportStyle As XslTransform = New XslTransform()  
' 将样式表单加载到对象中  
ExportStyle.Load("http://myserver/ExportProduct.xsl")
```

# Load()方法可以使用的数据源参数

## 3.9.2 创建XSLT处理器对象

数据源	功能说明
String	包含有样式表单URL或文件路径的字符串
XmlReader	XmlReader类的某个子类对象，例如可以包含样式表单的XmlTextReader
XPathNavigator	XPathNavigator对象可以包含样式表单
IXPathNavigable	使用实现过IXPathNavigable接口的某个类的对象，例如 XmlNode 或 XPathDocument 对象

# 应用XSLT样式表单

## 3.9.3 应用XSLT样式表单

- 调用XSLTransform对象的Transform()方法把XSLT样式表单应用到数据源
- Transform()方法可以带三个参数
  - 输入数据
  - XsltArgumentList对象（可选）
  - 输出目标

# 在Web应用程序中显示XML数据

## 3.9.4 在Web应用程序中显示XML数据

- ASP.NET中包含Xml Web窗体控件

```
<asp:Xml id="xml Control "  
    Document="Xml Document object to display"  
    DocumentContent="String of XML"  
    DocumentSource="Path to XML Document"  
    Transform="Xsl Transform object"  
    TransformSource="Path to XSL Document"  
    runat="server" />
```

- 可以在Visual Studio .NET的属性窗口中配置Xml控件
- 在编程中通过代码设置属性

# 控件使用的数据源属性

## 3.9.4 在Web应用程序中显示XML数据

属性	代表的数据源
Document	来自XmlDocument对象的数据
DocumentContent	XML数据字符串，也可以直接把字符串放在<asp:Xml>控件标签之间
DocumentSource	来自XML文件

# 可视化设置控件属性

## 3.9.4 在Web应用程序中显示XML数据

XML 数据源

添加XML 控件

样式表单文件

Toolbox

- My User Controls
- Data
- Web Forms
  - DataGrid
  - DataList
  - Repeater
  - ☒ CheckBox
  - ☐ CheckBoxList
  - ☐ RadioButtonList
  - ☐ RadioButton
  - Image
  - Panel
  - Placeholder
  - Calendar
  - AdRotator
  - Table
  - RequiredFieldValidator
  - CompareValidator
  - RangeValidator
  - RegularExpressionVa...
  - CustomValidator
  - ValidationSummary
  - Xml

Properties

Xml1 System.Web.UI.WebControls.Xml

(DataBindings)	
(ID)	Xml1
DocumentSource	Xml1
EnableViewState	True
TransformSource	
Visible	True

## 第3章 XML基础

- XML概述
- XML的格式
- 设计XML词汇表
- 命名空间
- XML解析器
- 使用DOM浏览XML
- 使用DOM创建新节点
- XML转换概述
- XSLT处理器
- 扩展XSLT样式表单
- 参考资源



# 扩展XSLT样式表单

## 3.10 扩展XSLT样式表单

- 为XSLT样式表单传递参数
- 扩展对象
- 使用扩展对象的原因
- 将扩展对象传递给样式表单
- 调用扩展对象

# 为XSLT样式表单传递参数

## 3.10.1 为XSLT样式表单传递参数

- XSLT样式表单可以接受参数

```
<xsl:param name="discount" />  
...  
<xsl:value-of select="@price - (@price*$discount)" />
```

- 使用XsltArgumentList对象向样式表单传递参数
  - 创建XsltArgumentList对象
  - 调用AddParam()方法添加新的参数
  - 在样式表单中引用参数
  - 在Transform()方法中传递XsltArgumentList对象

# 传递参数代码示例

## 3.10.1 为XSLT样式表单传递参数

```
private String InputFile = "order.xml";
private String StyleFile = "order.xsl";
private String ResultFile = "currentOrder.xml";
// 创建 XsltTransform对象并加载样式表单
XsltTransform Transformer = new XsltTransform();
Transformer.Load(StyleFile);
// 创建参数列表对象 XsltArgumentList.
XsltArgumentList ArgList = new XsltArgumentList();
// 创建参数保存当前时间
DateTime RightNow = DateTime.Now;
ArgList.AddParam("date", "", RightNow.ToString());
// 创建XmlTextWriter对象以便保存输出
XmlTextWriter Xtw = new XmlTextWriter(ResultFile, null);
// 传递 XsltArgumentList参数给 Transform()方法
Transformer.Transform(new XPathDocument(InputFile), ArgList, Xtw);
Xtw.Close();
```

# 扩展对象

## 3.10.2 扩展对象

- XSLT样式表单可引用的对象
- 扩展对象中的每个方法都可以在样式表单中以自定义 XSLT函数的形式调用

```
Public Class MyClass  
    Sub MyMethod(ByVal someInt As Int32)  
        ...  
    End Sub  
End Class  
...  
Dim MyObject As New MyClass()
```

应用程序代码

XSLT 样式表单

```
<xsl:stylesheet ...  
    xmlns:myns="urn:myURN">  
    ...  
    <xsl:value-of  
        select="myns:MyMethod(123)" />  
    ...  
</xsl:stylesheet>
```

# 使用扩展对象的原因

## 3.10.3 使用扩展对象的原因

- 扩展对象是为XSLT样式表单增强功能的有效方法
- 使用扩展对象可以
  - 补充XSLT的功能
  - 允许现有业务逻辑的重用
  - 有助于保持样式表单的简单性
  - 采用模块化结构
  - 提高性能

# 将扩展对象传递给样式表单

## 3.10.4 将扩展对象传递给样式表单

- 传递扩展对象时需要
  - 定义类同时至少声明一个公有方法
  - 实例化该类的对象
  - 把该对象加入到XsltArgumentList对象，并用唯一的命名空间URI标识
- 执行格式转换时需要
  - 与前面一样，创建一个XslTransform对象
  - 调用Transform()方法，XsltArgumentList 对象作为参数传递

# 扩展对象类声明代码示例

## 3.10.4 将扩展对象传递给样式表单

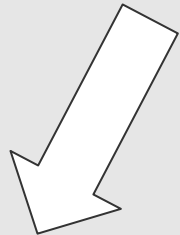
```
public class Calculate {  
    public double GetCircumference(double radius) {  
        return Math.PI*2*radius;  
    }  
}
```

# 调用扩展对象

## 3.10.5 调用扩展对象

- 在样式表单中使用扩展对象，需要
  - 声明与扩展对象相同的命名空间
  - 将命名空间URI与某一前缀相关联
  - 在样式表单中调用方法，使用命名空间的前缀作为方法的前缀

```
<xsl:stylesheet version="1.0" xmlns:myns="urn:MyObject"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="circles|circle">
    <xsl:copy>
      <xsl:apply-templates />
    </xsl:copy>
  </xsl:template>
  <xsl:template match="radius">
    <circumference>
      <xsl:value-of select="myns:GetCircumference(.)"/>
    </circumference>
  </xsl:template>
</xsl:stylesheet>
```





# 回顾

## 3.10.5 调用扩展对象

学习完本章后，将能够：

- 了解XML的基本概念，如格式、词汇表、命名空间
- 理解XML文档对象模型
- 使用DOM浏览XML和创建新节点
- 掌握XML转换
- 掌握XSLT处理器
- 理解扩展XSLT样式表单