

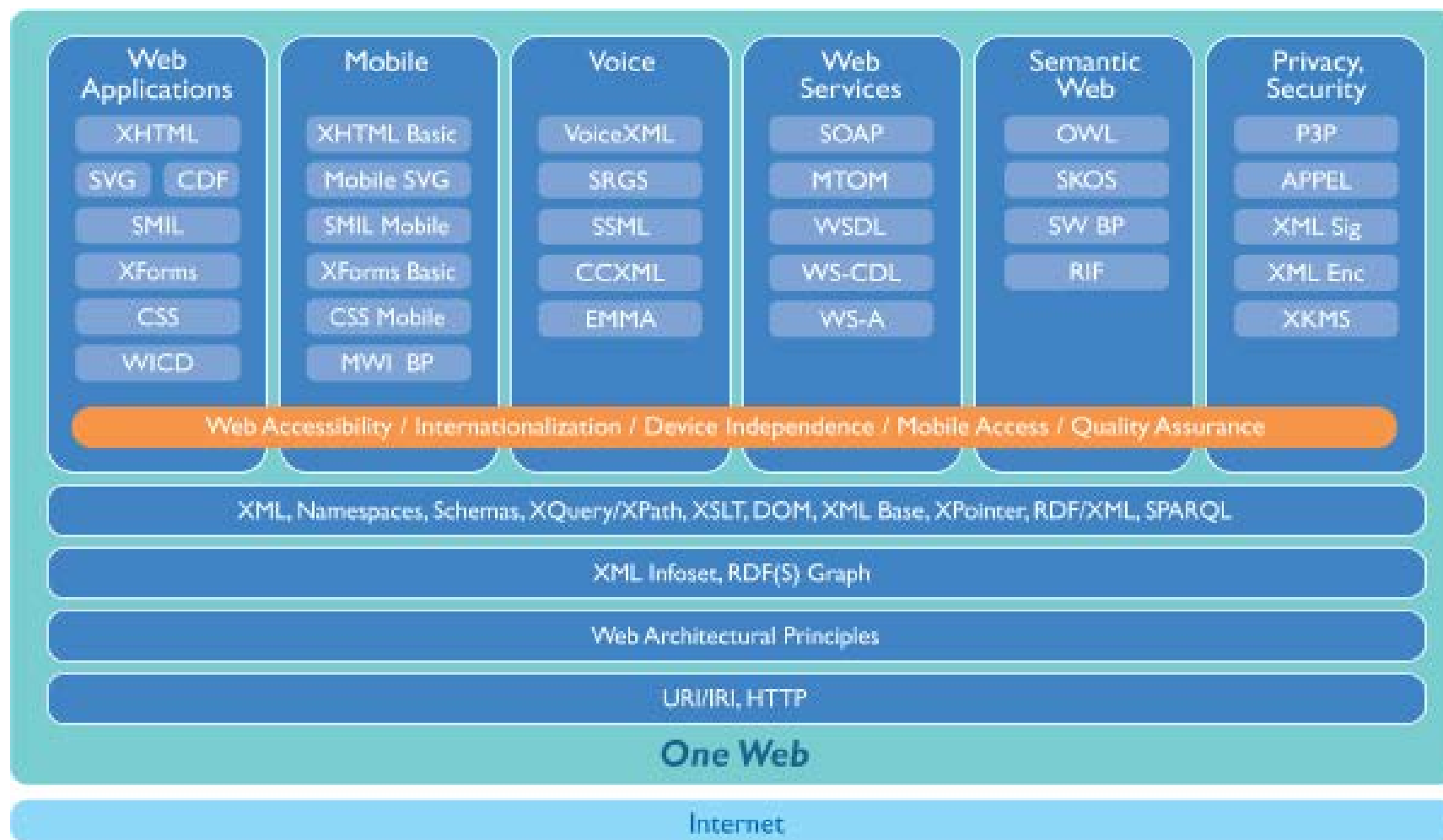
高级Web技术

Web上的数据标准XML

本部分课程内容

- XML概述与语法
- XML描述和验证规范： DTD,SCHEMA
- XML转换标准： XSL
- XML编程接口规范： DOM,SAX,Stax
- XML应用概述（Web服务与语义Web协议）

w3c技术架构图



XML简介与起源

- XML是 Extensible Markup Language (可扩展标记语言)的简称，是一种元语言(定义其他语言的系统)
- 1996年, W3C (World Wide Web Consortium, 互联网联合组织) 为了克服HTML的局限性, 在SGML的基础上创建XML
- 计算机产业界 的需求: 开发简单而又可扩展的、结构化和半结构化信息文本表示机制
- HTML是SGML的一个应用, XML是SGML的一个子集, 并且已经成为Internet上事实的数据交换标准
- 1998年2月推出第一个正式的XML1.0版本; 目前XML已经以扩展巴科斯-诺尔范式 EBNF(Extended Backus-Naur Form notation)的形式正式定义了下来
 - (*XML 1.1 (Second Edition)*, W3C Recommendation, 16 August 2006)
 - <http://www.w3.org/TR/2006/REC-xml11-20060816/>
- 独立于平台的XML是为Web开发的, 这是它最具影响的地方.

HTML的局限性

- 可扩展性差

- HTML的标记集是固定的，即HTML语法是不可扩展的

- 缺乏对信息含义的描述能力，信息检索效率低

- HTML是针对人机交流而设计的，标记几乎全都是用来设计网页的布局和外观的。检索信息时，需要对全部页面的所有内容扫描，并且检索质量往往极差

- 与应用程序的数据自动交换受限制

- 服务器端在HTML中嵌入动态数据是非常困难的
- 客户端应用程序也很难自动从HTML中获取所需的数据

HTML的局限性

■ 描述能力有限

- HTML语言不能描述矢量图形、数学公式、化学符号等特殊对象

■ 链接功能有限

- 链路丢失后不能自动纠正
 - HTML不能维持文档间的任何历史和关系，因此如果页面的URL地址变化了，浏览这些页面时就会遇到404 URL地址未找到的信息。
- 链接方式是纯单向的
 - 虽然链接文档知道它要链接到的地法，但被链接的文档却无法知道它是从何处被链接的，而这一点对于开发者往往是很重要的信息。

XML的设计目的

- 设计目的

- 使得在Web上以现有的HTML方式提供、接收和处理通用的SGML成为可能

Its goal is to enable generic SGML to be served, received, and processed on the Web in the way that is now possible with HTML. XML has been designed for ease of implementation and for interoperability with both SGML and HTML.

-----(XML) 1.1 W3C Recommendation

XML的设计目的

- **W3C建议的XML 1.0中对XML设计目标:**

- (1) 应该可以在Internet上直接使用
- (2) 应该广泛的支持不同的应用
- (3) 与SGML兼容
- (4) 处理XML的文档应该容易编写
- (5) 可选特征应该尽可能少,最好为0
- (6) XML文件要易读,清晰
- (7) XML应易于设计。
- (8) XML的设计应该正式而且简洁。
- (9) XML文档应易于创建。
- (10) XML标记的简洁性较为次要

- **此外,还有两个隐含目标:**

- 可扩充性
- 语义与表现形式的分离

一个XML样本文档

```
<person>
  <name>
    <title>Teacher</title>
    <first-name> kaiyu </first-name>
    <last-name> dai </last-name>
  </name>
  <email> wenuanhappy@gmail.com </email>
  <hometown province= "Hunan">Xiangtan</hometown>
</person>
```

[View in browser](#)

- **标记**是左尖括号（<）和右尖括号（>）之间的文本。有开始标记（例如 <name>）和结束标记（例如 </name>）
- **元素**是开始标记、结束标记以及位于二者之间的所有内容。在上面的样本中，<name> 元素包含三个子元素：<title>、<first-name> 和 <last-name>。
- **属性**是一个元素的开始标记中的名称-值对，属性间用空白符号隔开。在该示例中，province 是 < hometown > 元素的属性；

XML的特点

■ 可扩展性

- XML是一种元标记语言，它定义了一组用来形成语义标记的规则集，用户可以构造自己的标记。
- 已有化学标记语言CML、数学标记语言MathML等

■ 对文档内容具有自描述能力，支持智能代码和智能搜索

- XML是一种语义化的标记语言，具有自描述性. XML文档本身仅包含描述文档内容的标记，并不描述文档的外观格式
- 应用程序更容易定位文档中的信息

XML的特点

■ 结构化数据表达能力

- 结构化的数据指的是其内容,意义或应用被标记的数据
- 通过文档类型定义DTD或XML Schema, XML指定文档中的元素以及元素之间的关系.
- XML还提供了一种将多个数据源数据集成成为单个文档的客户端包括机制; 数据位置也可以重排; 根据用户的操作, 部分数据还可以被隐藏/显示。

■ 良好的通用数据格式, 跨平台以及语言独立

- XML 成为Web上交换数据 和文档的标准机制
- XML是一种独立于平台的信息表示格式.
- 并没有定义数据文件中数据出现的具体规范, 而是在数据中附加tag来表达数据的逻辑结构和含义。这使XML成为一种跨平台程序能自动理解的规范

XML的特点

■ 强大的超链接功能

- XLL代表XML的链接语言,由用于链接的XLinks和用于定位文档中某部分的Xpointers的两部分组成
- XLL支持可扩展的链接和多方向的链接。它打破了HTML只支持超级文本概念下最简单的链接限制,能支持独立于地址的域名、双向链路、环路、多个源的集合链接等。

■ 文档的表示形式多样化

- 通过将结构,内容和表现分离,同一个XML源文档只写一次,可以用不同的方法表现出来。数据表示是由层叠样式表CSS或可扩展样式语言XSL来实现的
- 用户可以根据具体需要为同一个XML文件编写多个样式文件,为文档中的元素设计不同的显示/打印样式。这将使网络的用户界面更趋于个性化、风格化

XML的特点

■ 降低了对服务端的要求

- XML没有SGML那么复杂, 它是设计用于有限带宽的网络的, 如Internet.
- 服务器只须发出同一个XML文件, 而由客户根据自己的需求选择和制作不同的应用程序以处理数据.服务端可以集中精力尽可能完善、准确地将数据封装进XML文件中
- 使广泛、通用的分布式计算成为可能

XML的特点

■ 以文档为中心与以数据为中心的XML

– 文档为中心

- 作为表示半结构化文档的机制，被出版系统采用
- 文档的关键元素是半结构化的、带置标的正文

– 数据为中心

- 用于标记高度结构化的信息
- 一般由机器生成，由机器消费
- 与文档为中心的XML区别
 - 内容置标率很高，且一般没有大段正文
 - 一般包含机器生成的信息
 - 被组织成高度结构化的形式，一个标记与其他标记的相对次序和放置位置非常重要
 - 用置标来描述一个片段信息

XML规范组成

- XML相关规范可以被分为两大部分:

- 核心层:

- XML语法

- Ref: www.w3.org/TR/REC-xml

- 包含文档结构, 元素, 属性, 注释, 实体等多个部分内容的规定

- XML的描述和验证:

- DTD (Document Type Definition, 文档类型定义)

- 本身不使用XML语法

- 任何进行有效性验证的解析器都可以通过使用DTD来验证XML文档的有效性

- 带有一定的局限性

- Schema(模式)

- 由W3C官方制定, 使用XML 1.0语法

- 可以对数据进行更详细的控制

- 可以使用工具将DTD转换为Schema

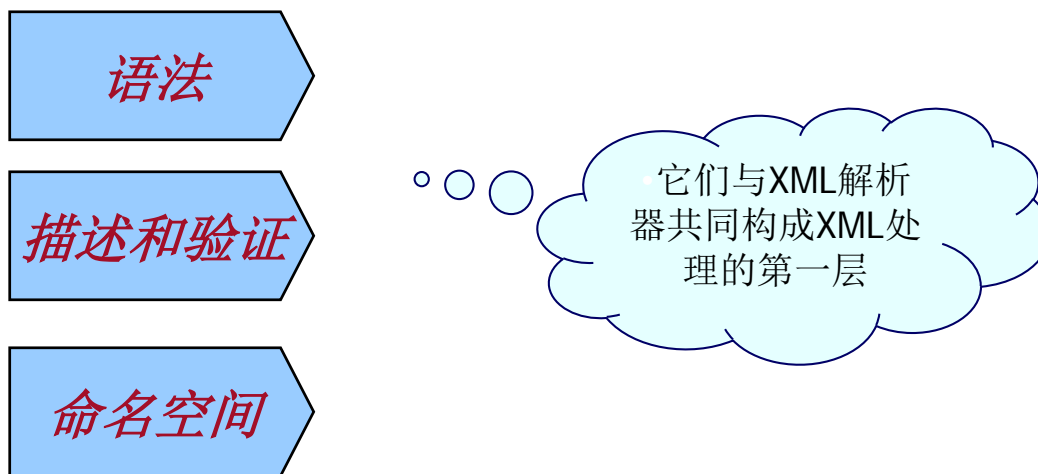
XML规范组成

- XML及其相关规范可以被分为两大部分:

- 核心层:

- XML命名空间(Namespace)

- 为了防止共享词汇表的时候发生名字冲突, 在单一文档中使用多个词汇表
 - 一种复合命名方法
 - 是其他XML技术如Schema或InfoSet关键组成部分



XML规范组成

- XML相关规范可以被分为两大部分:

- 应用程序支持层:

- XML定位与链接:

- XML Base规范

- 设置XML文档的基本URI

- XPath (XML路径语言)

- 用于对XML数据对象的特定部分进行编址

- 其表达式使用了一种压缩的非XML语法,可以在URI和属性内部使用

- 提供了一种查询数据的简单方法

XML规范组成

- XML规范可以被分为两大部分:

- 应用程序支持层:

- XPointer (XML指针语言)

- 基于XPath表达式,并扩展了XPath:
 - 提供在XML数据中任意的点和范围的相对位置的编址
 - XML数据内部的字符串匹配
 - 用其找到的位置可以作为链接目标

- XLink (XML链接语言)

- 定义了一些特殊元素,可以在XML数据中使用,以创建资源间链接
 - 可以不用改变文档的内容而描述指向不同的XML文档的链接
 - 不仅指定了链接的数据结构,也定义了一个简单的链接行为模型,可以被高级应用层扩展

XML规范组成

- XML规范可以被分为两大部分:

- 应用程序支持层:

- XInclude (XML包含)

- 为XML的通用包含提供了过程模式和语法

- XFI(文档片断交换)

- 用于对XML数据段进行描述和传递
 - 可以使查看和编辑XML文件时不用发送整个数据对象,也不用使用显式定义的外部实体

- XQuery (XML查询语言)

- XQuery是一种新的查询语言提案,是用非XML语法表达的
 - 可以使用XPath2.0表达式,也可是使用类似于SQL查询的表达式

XML规范组成

- XML规范可以被分为两大部分:

- 应用程序支持层:

- XML转换:

- CSS(Cascading Style Sheet层叠样式图)

- 被设计用于使数据作为网页表示,还有许多媒体方面的扩展功能
 - 可以使数据的描述和表示分开
 - 是XSL的一个有用的补充


- XSL (eXtensible Style Language可扩展样式语言)

- 有一个子集为XSLT(用于转换的XSL)
 - 是一种描述性语言,只需要陈述一些规则即可自动调用模板
 - 另一个子集XSL-FO (Formatted Object格式化对象) 目标是将XML数据精确显示和打印

XML文档规则- XML 文档分类

■ 三种 XML 文档

- 无效文档 (Invalid)
 - 没有遵守 XML 规范定义的语法规则
 - 开发人员已经在 DTD 或模式中定义了文档能够包含什么，而某个文档没有遵守那些规则
- 有效文档 (Valid)
 - 遵守 XML 语法规则也遵守在其 DTD 或模式中定义的规则
- 格式良好的文档 (Well-Formed)
 - 满足XML规范的语法要求



XML文件至少应该是格式良好的!

XML文档规则-字符

- XML 规范定义了五个可以用来替代不同的特殊字符的实体：

- < 代表小于符号
- > 代表大于符号
- " 代表一个双引号
- ' 代表一个单引号（或撇号）
- & 代表一个“与”符号。

```
<?xml version="1.0" standalone="yes"?>
<document>
  <greeting>
    This is &lt;greeting&gt; element
  </greeting>
</document>
```

[View in browser](#)

XML文档规则-声明

■ XML 声明

- 向解析器提供了关于文档的基本信息不是必需的。如果有的话，那么它一定是文档的第一项。
- 声明包含名称-值对
 - version 是使用的 XML 版本
 - encoding 是该文档所使用的字符集。缺省为 UTF-8 字符集
 - 在XML声明中添加encoding="GB2312"属性，就可以在XML中使用简体中文

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

[View in browser](#)

```
<?xml version="1.0" encoding="GB2312"?>
<!-- 一个使用中文的例子-->
<诗歌 作者="曹雪芹">
    满纸荒唐言，
    一把辛酸泪。
    都云作者痴，
    谁解其中味？
</诗歌>
```

XML文档规则-注释

■ 注释

- <!-- 开始，以 --> 结束。注释不是文档字符数据的一部分，XML处理器不对<!--和-->中间的文本进行解析
- 注释不能在结束部分以外包含双连字符（--）；除此之外，注释可以包含任何内容。最重要的是，注释内的任何标记都被忽略；
- 不能出现在XML声明之前，XML声明绝对是文档的第一项内容
- 不能放在任何一个标记中
- 示例：

```
<!-- Here's a PI for Cocoon: -->  
<?cocoon-process type="sql"?>
```


XML文档规则-根元素

■ 根元素

- XML 文档必须包含在一个单一元素中。这个单一元素称为根元素，它包含文档中所有文本和所有其它元素。一个XML文档只能有一个根元素!
- 一个无效XML文档

```
<?xml version="1.0"?>
<!-- An invalid document -->
<song> sound of silence </song >
<song > green sleeve </song >
```

[View in browser](#)



XML文档规则-元素不能重叠

■ 元素不能重叠

- 在一个元素a中开始了另一个元素b，则必须在 a 元素中结束 b 元素
- 一个无效XML文档

```
<!-- NOT legal XML markup -->  
<p>  
  <b>I <i>really love  
  </b> XML. </i>  
</p>
```

[View in browser](#)



XML文档规则-结束标记

■ 结束标记是必需的

- 尽管这在 HTML（以及某些情况下在 SGML）中可以接受，但 XML 解析器将拒绝它
- 一个无效XML文档

```
<!-- NOT legal XML markup -->  
<p>I am sailing, I am sailing  
<p>Home again across the sea  
<p>I am sailing stormy waters  
<p>to be with you to be free
```



[View in browser](#)

XML文档规则

■ 元素内容

- 元素起始标记和结束标记之间的一切是元素的内容
- 内容类型
 - 只含子元素
 - 只包含正文的元素常常被称为含有“数据”内容
 - 混合
 - 嵌套元素和文本的组合
 - 空内容

XML文档规则-空元素

■ 空元素规定

- 如果一个元素根本不包含内容，则称为空元素；
- 在 XML 文档的空元素中，可以把结束斜杠放在开始标记中
- 空元素示例

```
<Myfavouratesongs>
<!-- Two equivalent elements -->

• <film name= " legend of the fall"></film >

• <film name= " legend of the fall " />
</Myfavouratesongs>
```

[View in browser](#)

XML文档规则-区分大小写

- 元素是区分大小写的

- 在 HTML 中，`<h1>` 和 `<H1>` 是相同的；在 XML 中，它们是不同的
- XML文档示例

```
<!-- NOT legal XML markup -->  
<h1>Elements are case sensitive</H1>
```



```
<!-- legal XML markup -->  
<h1>Elements are case sensitive</h1>
```



XML文档规则-属性

■ 属性必须有用引号括起的值

- 属性必须有值
- 那些值必须用引号括起,可以使用单引号,也可以使用双引号,但要始终保持一致
- 同一个标记不能包含相同名称的两个属性
- 如果属性值包含单引号或双引号,则可以使用另一种引号来括起该值(如 `name="Doug's car"`), 或使用实体 `"` 代表双引号, 使用 `'` 代表单引号。

```
<!-- NOT legal XML markup -->  
<ol compact />
```



```
<!-- legal XML markup -->  
<ol compact="yes" />
```



XML文档规则-处理指令

■ 处理指令

- 处理指令是为使用一段特殊代码而设计的标记
- 进程指令以应用程序的名字（即进程指令所要执行的操作）开头，后面是该指令的数据。
- 示例：Cocoon 是来自 Apache 软件基金会（Apache Software Foundation）的 XML 处理框架。当 Cocoon 处理 XML 文档时，它会寻找以 cocoon-process 开头的处理指令，然后相应地处理 XML 文档。在该示例中，type="sql" 属性告诉 Cocoon：XML 文档包含一个 SQL 语句。

```
<?cocoon-process type="sql"?>
```

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

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


XML文档规则- CDATA段

■ CDATA段

- 以字符串 “<![CDATA [”开始，以字符串 “]]>”结束
- XML将保留CDATA段中的字符数据不被解析，这样CDATA段中包括的<、>、&或 “无需使用实体引用来转义
- CDATA段中唯一禁用的文本是关闭CDATA的标记]]>
- CDATA段不能嵌套

```
<DOCUMENT>
<MARKUP>
  <![CDATA[
    <greeting> Hello, &world! </greeting>
    This is a showcase of CDATA usage,
    and including a <greeting> element description
  ]]>
</MARKUP>
</DOCUMENT>
```

[View in browser](#)

XML文档规则-命名空间

■ 命名空间

- 为了区分相同名字代表的不同含义的元素
- 要使用名称空间，您要定义一个名称空间前缀，然后将它映射至一个特殊字符串
- 名称空间定义中的字符串仅仅是字符串，要确保其唯一性！
- 示例：

```
<?xml version="1.0"?>
<customer_summary
  xmlns:addr="http://www.xyz.com/addresses/"
  xmlns:books="http://www.zyx.com/books/"
  xmlns:mortgage="http://www.yyz.com/title/" >
  <addr:name><title>Mrs.</title></addr:name>
  <books:title>Lord of the Rings</books:title>
  <mortgage:title>NC2948-388-1983</mortgage:title>
</customer_summary>
```

名称空间前缀

唯一的标志符

XML文档规则-命名空间

- xmlns并没有限定在根元素中使用，它可以存在于子元素中
- 冒号不能出现在名称空间前缀中
- xml和xmlns不能作为名称空间前缀

```
<?xml version="1.0"?>
<customer_summary
  xmlns:addr="http://www.xyz.com/addresses/"
  xmlns:xml="http://www.zyx.com/books/"
  xmlns:mortgage="http://www.yyz.com/title/" >
  <addr:name><title>Mrs.</title></addr:name>
  <books:title>Lord of the Rings</books:title>
  <mortgage:title>NC2948-388-1983</mortgage:title>
</customer_summary>
```

[View in browser](#)

XML文档规则-命名空间

- 只使用xmlns属性而不指定任何前缀，代表一个缺省名称空间。所有的封装元素被假设属于该名字空间

缺省名称空间

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/tr/xhtml1/DTD/xhtml1-transitional.dtd">
<!-- A Sample XML -->
<?xml-stylesheet href="mystyle.css" type="text/css" ?>
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" >
    ....
</html>
```

XML文档规则-小结

■ 一个格式良好的XML文档必须满足以下基本条件

- 包含一个或多个元素；
- 有且仅有一个称为根（root）或文档元素的元素，它不出现在其他任何元素的内容中；
- 非空元素的开始标记和结束标记必须匹配；
- 元素可以嵌套，但是不能交叉，即应符合树型结构；
- 文档中直接或间接引用的每一个已析实体都是格式良好的

DTD简介

- **文档类型定义(Document Type Definition, DTD)起源于SGML**
中更复杂的**DTD**, 是一组能融合在XML数据中或者以单独的文档存在的声明,用于对XML文档进行描述和校验,以保证XML文档的有效性
- **验证与有效性**
 - 验证: 确认 XML 数据遵循特定的预定的结构从而使应用程序可以以可预知的方式来接收数据
 - 验证方法主要包括文档类型定义 (Document Type Definition, DTD) 和 XML Schema
 - 有效性检查将一个特定的文档与一个DTD进行比较验证。用这种方法对照 DTD 或模式检查过的文档被认为是“有效的 (valid)” 文档。
 - 良构性 (Well-form) 是有效的前提

DTD简介

- **DTD使用非XML语法的文法来定义一些规则,这些规则描述了XML的结构和语法,以及被允许出现的XML数据内容**
- **DTD优点**
 - 可以促使不同的应用程序可以读取相互的文件, 实现共享.每一个XML文档都可携带对其本身格式的说明; 独立的人员小组可统一使用共同的DTD来交换数据;
 - 应用程序可使用一种标准的DTD来核实将要处理的数据是否有效。

DTD简介

- **DTD本身是可选的**

- XML文档可以只是一个正规的文档
- 还有其他可以用来保证XML文档有效性的技术:
 - Schema
 - XML-Data
 -

- **DTD的内容和特征:**

- 对数据结构进行描述和确认
- 将这些数据结构传送给其他应用程序和人员
- 限制元素内容
- 限制属性类型和值,提供默认值
- 对可置换内容--实体(Entity)进行描述和定义
- 可以有条件节

DTD简介

■ DTD结构:

- 一个文档只能与一个DTD相关联,但是一个DTD可以分成两部分:内部子集和外部子集
- 内部子集被包含于XML文档中
- 外部子集可以存放在以dtd为后缀名的独立文件中
- 当内部子集和外部子集存在声明冲突时,内部子集的声明覆盖外部子集的声明
- 外部子集和内部子集的选择:
 - 一般情况尽量使用外部子集,便于修改和复用
 - 下面情况下使用内部子集:
 - 为了使用已经存在但不能完全满足我们需要的DTD, 通过修改和扩充该外部DTD
 - 在DTD开发过程中进行测试

DTD简介-示例

- greeting.xml:

```
<?xml version="1.0"?>
<GREETING>
  Hello XML!
</GREETING>
```

- greeting.dtd:

```
<!ELEMENT GREETING (#PCDATA)>
```

- 有效的文档

```
<GREETING>
```

various random text
but no markup

```
</GREETING>
```

- 无效的文档

```
<GREETING>
```

```
  <sometag>
```

random text

```
  </sometag>
```

```
  <someEmptyTag/>
```

```
</GREETING>
```

文档类型声明

- **文档类型声明(Document Type Declaration)将XML文档与DTD关联起来**

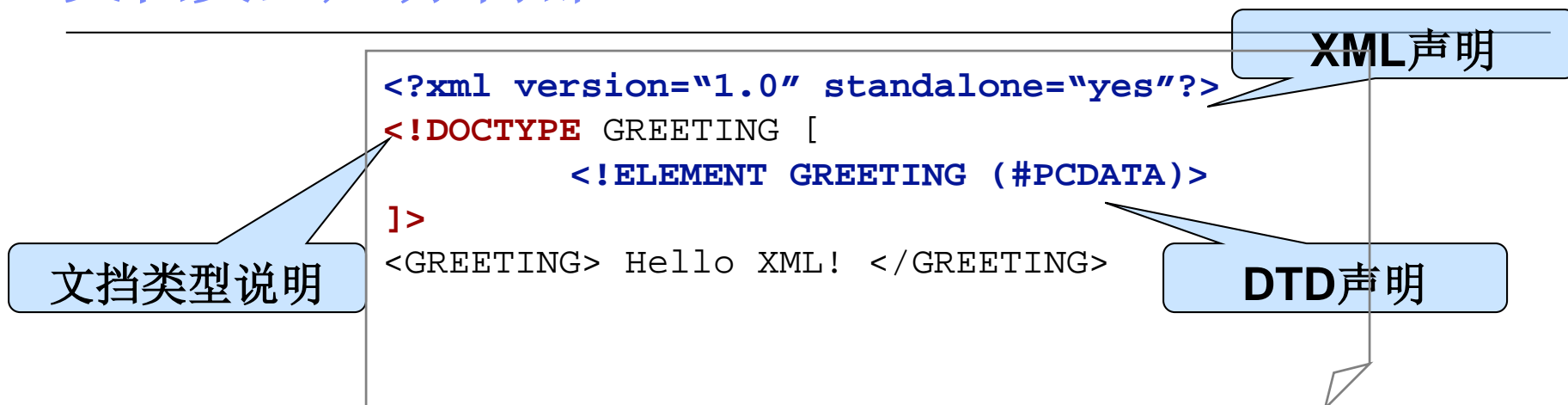
- 文档类型声明出现在文档的prolog中，在XML声明之后以及根元素之前
- 文档类型声明包含或者指向一个DTD，但在DTD中不能再包含文档类型声明

- **语法:**

```
<!DOCTYPE doc_elem [SYSTEM|PUBLIC] [identifier] [location] [internal subset]>
```

- **doc_elem: XML文档的根元素**

文档类型声明-内部 DTD



- DTD可以直接包含在使用该DTD的文档内
- 文档类型声明
 - 文档类型声明开始于标记 `<!DOCTYPE`, 结束于标记 `]>` .
 - DTD: `<!ELEMENT GREETING (#PCDATA)>`
- 根元素: GREETING

文档类型声明-外部 DTD

- 外部DTD独立存在于另一个文件中，从一个或多个外部URL链接到XML文档中；XML的一个重要特点来自于公共DTDs，能在不同的XML文档中共享这些DTDs

- 私有 DTDs

- 语法（使用*SYSTEM* 关键字）：

`<!DOCTYPE root_element_name SYSTEM "DTD_URL">`

- 公用 DTDs

- 用于更加广泛的组织，可作为一个工业界标准，如ISO，或者IEEE，可以标准化公用DTDs
 - 语法：

`<!DOCTYPE root_element_name PUBLIC FPI URL >`

```
<!DOCTYPE DOCUMENT PUBLIC "-//starpowder//Custom XML Version
1.0//EN" "http://www.starpowder.com//steve/order.dtd" >
```

DTD声明

- 整个DTD可以分为内部子集与外部子集,而每个子集又是由零条到多条基本**DTD声明**构成的
- 语法:
 - **<!keyword param1 param2 ... paramN>**
 - **<!keyword**中不能有空格
 - 基本的四个关键字为:
 - **ELEMENT** 用于声明元素与子元素
 - **ATTLIST** 用于声明属性与默认值
 - **ENTITY** 用于声明各种实体
 - **NOTATION** 用于描述非XML内容

DTD声明-ELEMENT

元素声明

语法:

- `<!ELEMENT name content_category>` 或者
`<!ELEMENT name (content_model) cardinality>`

- 没有integer, floating point, date, 或者其他数据类型定义
- 元素的声明次序是没有关系的

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE DOCUMENT[
  <!ELEMENT files EMPTY>
  <!ELEMENT DOCUMENT (files)>
] >
<DOCUMENT>
<files></files>
</DOCUMENT>
```

DTD声明- ATTLIST

- 声明元素的属性
- 语法:

```
<!ATTLIST elementName  
    attrName1 attrType1 attrDefault1 defaultValue1  
    attrName2 attrType2 attrDefault2 defaultValue2  
    ...  
    attrNameN attrTypeN attrDefaultN defaultValueN  
>
```

- **ATTLIST**关键字加零个或多个属性定义
- **elementName**: 属性所属元素名
- **attrName**: 属性名,必须为合法XML名字
- **attrType**: 属性类型,十类合法的类型之一
- **attrDefault**: 属性是否必须等特征
- **defaultValue**: 属性默认值,可选

DTD声明- ATTLIST

属性声明

– DTD中

```
<!ELEMENT User (Assignment)>
```

```
<!ATTLIST User Identity CDATA "guest">
```

```
...
```

– 使用中

```
<User>...</User>
```

```
<User Identity="maintainer">...</User>
```

Schema简介

- XML Schema是W3C的推荐标准，于2001年5月正式发布
- XML Schema同DTD一样是负责定义和描述XML文档的结构和内容模式
- 它可以定义XML文档中存在哪些元素和元素之间的关系，并且可以定义元素和属性的数据类型。
- XML Schema本身是一个XML文档，它符合XML语法结构。可以用通用的XML解析器解析它
- 它使用 XML 名称空间而不是 DOCTYPE

Schema简介

■ DTD的缺陷：

- DTD是基于正则表达式的，描述能力有限
- DTD没有数据类型的支持，在大多数应用环境下能力不足
- DTD的约束定义能力不足，无法对XML实例文档作出更细致的语义限制
- DTD的结构不够结构化，重用的代价相对较高
- DTD并非使用XML作为描述手段，而DTD的构建和访问并没有标准的编程接口，无法使用标准的编程方式进行DTD维护。
- DTD不直接支持命名空间

Schema简介

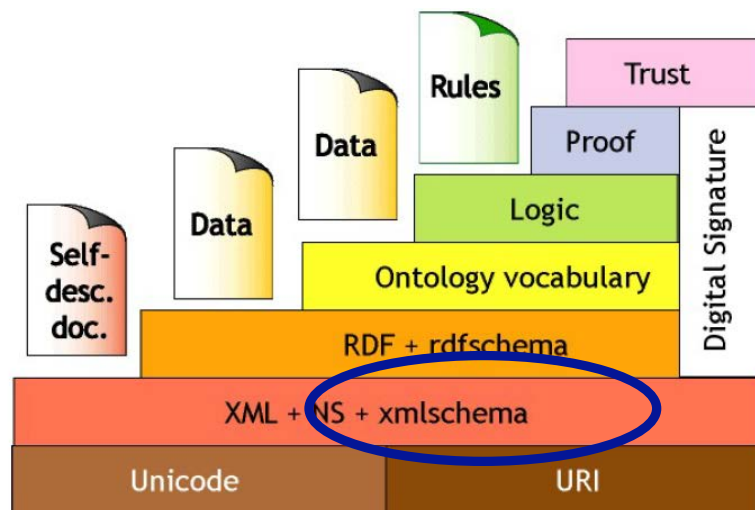
■ XML Schema的优点:

- XML Schema基于XML,没有专门的语法
- XML Schema可以象其他XML文件一样解析和处理
- XML Schema支持一系列的数据类型(int、float、Boolean、date等)
- XML Schema提供可扩充的数据模型
- XML Schema支持综合命名空间
- XML Schema支持属性组
- XML Schema支持继承和扩充,类似面向对象思想

Schema简介

■ XML Schema 语言定义成以下三个部分：

- 入门位于 w3.org/TR/xmlschema-0，它介绍了 XML 模式文档及其设计用途；
- 文档结构的标准位于 w3.org/TR/xmlschema-1，它说明了如何定义 XML 文档的结构；
- 数据类型的标准位于 w3.org/TR/xmlschema-2，它定义了一些常用数据类型以及创建新类型的规则



Schema简介

■ XML Schema的软件以及工具支持情况

- XML Spy的支持情况:
Ref: http://new.xmlspy.com/features_schema
- 其他软件实现:
Ref: <http://www.w3.org/XML/Schema.html> (包括将DTD转换成Schema的工具)
- Altova MapForce™ 用于两个schema间的映射, 以及将符合某一schema的XML文件转换为符合另一个schema的XML文件
- Altova XMLSPY可以实现数据库表结构到schema的转换
- Apache的Xerces XML解析器
Ref: <http://xml.apache.org/xerces-j>

模式文档和实例文档

- 模式文档只是一个 XML 文档，它的预定义的元素和属性描述另一个 XML 文档的结构
- 模式文档由一个schema元素和一系列子元素组成，大多数子元素为 `element`, `complexType`, 和 `simpleType`，这些决定了在实例文档中的元素的表现方式和内容
- 通过出现在schema元素中的命名空间声明 `xmlns:xsd="http://www.w3.org/2001/XMLSchema"`，在模式文档中的每一个元素都有一个与XML Schema命名空间相联系的命名空间前缀“`xsd:`”。前缀“`xsd:`”被约定用于表示XML Schema命名空间，而不是模式文档作者自己的词汇表



模式文档和实例文档

■ 模式文档示例

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:po="http://www.example.com/PO1"
  targetNamespace="http://www.example.com/PO1"
  elementFormDefault="unqualified"
  attributeFormDefault="unqualified">

  <element name="purchaseOrder" type="po:PurchaseOrderType"/>
  <element name="comment" type="string"/>

  <complexType name="PurchaseOrderType">
    <sequence>
      <element name="shipTo" type="po:USAddress"/>
      <element name="billTo" type="po:USAddress"/>
      <element ref="po:comment" minOccurs="0"/>
      <!-- etc. -->
    </sequence>
    <!-- etc. -->
  </complexType>

  <complexType name="USAddress">
    <sequence>
      <element name="name" type="string"/>
      <element name="street" type="string"/>
      <!-- etc. -->
    </sequence>
  </complexType>

  <!-- etc. -->

</schema>
```




模式文档和实例文档

■ 实例文档示例

```
<?xml version="1.0" ?>
- <apo:purchaseOrder
  xmlns:apo="http://www.example.com/PO1"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema
  -instance"
  xsi:schemaLocation="http://www.example.com/PO1
  schema1-withtargetNamespace.xsd">
- <shipTo>
  <name>Alice Smith</name>
  <street>123 Maple Street</street>
  <!-- etc. -->
</shipTo>
- <billTo>
  <name>Robert Smith</name>
  <street>8 Oak Avenue</street>
  <!-- etc. -->
</billTo>
<apo:comment>Hurry, my lawn is going wild!
  </apo:comment>
  <!-- etc. -->
</apo:purchaseOrder>
```

模式文档元素定义

- 在一个模式文档中定义元素就是给这个元素命名并给它分配一个类型，类型可以分为复合类型和简单类型。
- **复合类型**: 元素如果包含子元素或者是带有属性则被称为复合类型
- **简单类型**: 元素如果仅仅包含数字、字符串或者其他数据等，但不包含任何子元素则称为简单类型。属性值通常是简单类型，因为属性值不能包含任何结构
- 在实例文档中复合类型（通过<xsd:complexType>）和一些简单类型(通过<xsd:simpleType>) 在模式文档中定义。而其他一些标准的简单类型则是作为XML Schema内置的简单类型的指令表的一部分定义的

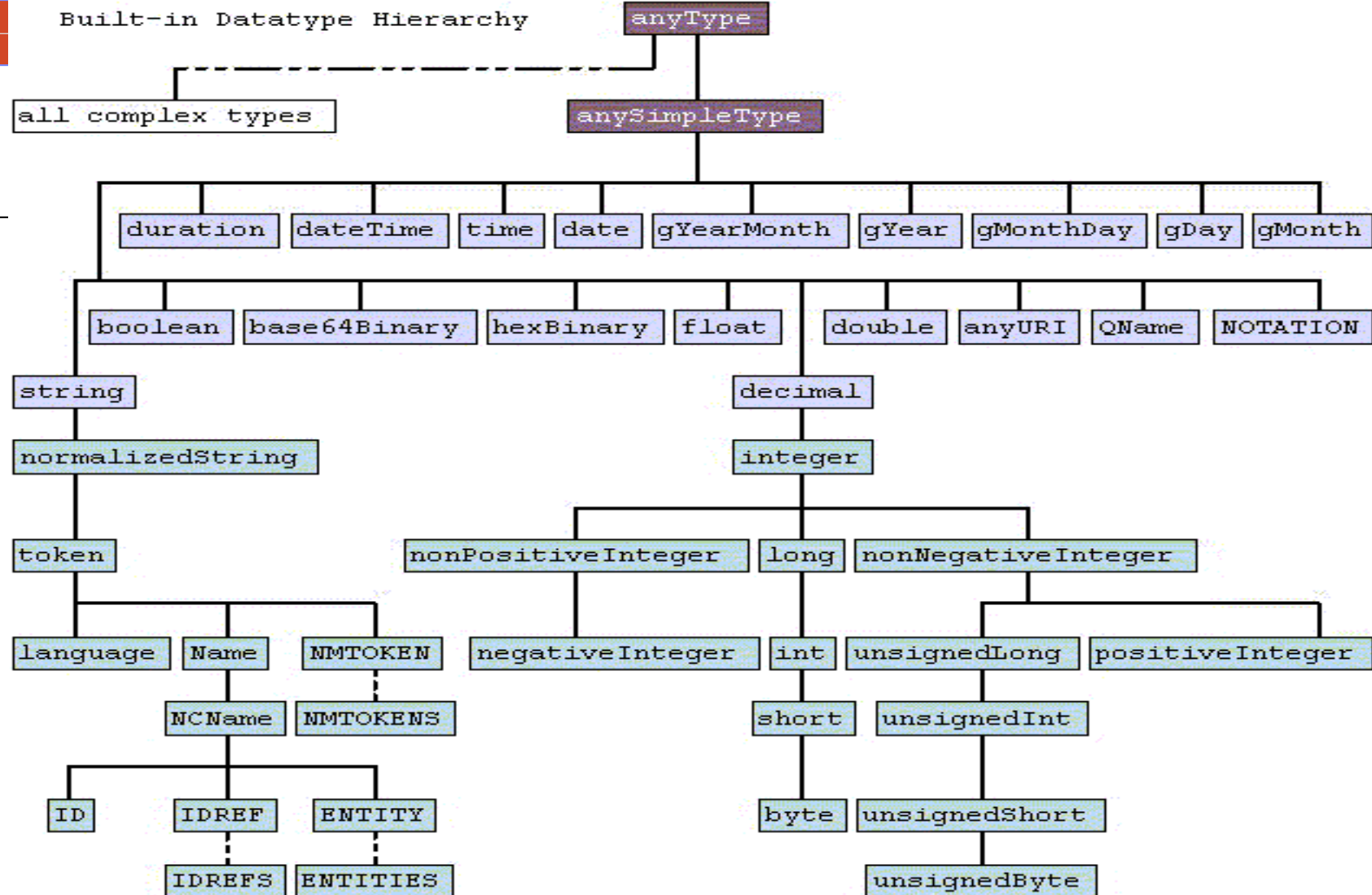
模式文档元素定义-内建简单类型

■ 内建简单类型

- W3C XML Schema 建议书中包括了 42 种简单类型的定义，其中包括 string、int、date、decimal、boolean、timeDuration 和 uriReference等。为了在XML Schema和XML 1.0 DTD之间保持兼容，简单类型ID、IDREF、IDREFS、ENTITY、ENTITIES、NOTATION、NMTOKEN、NMTOKENS只能用在属性定义中
- 使用模式内建的这些类型时，必须在前面加上W3C模式前缀，通常是“xsd:”

```
<xsd:element name="subdate" type="xsd:date"/>  
<xsd:element name="donor" type="xsd:string"/>  
<xsd:element name="subject" type="xsd:string"/>  
<xsd:element name="description" type="xsd:string" />  
<xsd:element name="place" type="xsd:string" />
```

Built-in Datatype Hierarchy



- ur types
- built-in primitive types
- built-in derived types
- complex types

- derived by restriction
- derived by list
- derived by extension or restriction

模式文档元素定义-创建新的简单类型

■ 使用层面（facets）创建简单类型

- 使用层面可以对简单类型存储的数据加以限制
- 几个常用的层面：
 - minInclusive, maxInclusive限定下限值和上限值
 - enumeration允许建立一个值列表
 - pattern指定一个正则表达式，要求文本符合该格式

```
<xsd:simpleType name="idNumber" base="xsd:integer">  
    <xsd:pattern value=" \d{3}-\d{4}- \d{3}" />  
</xsd:simpleType >
```

可以取 “123-4567-890”

模式文档元素定义-创建新的简单类型

■ 有限制的值

使用base属性声明基于内建的简单类型

```
<xsd:simpleType name="idNumber">  
  <xsd:restriction base="xsd:integer">  
    <xsd:minInclusive value="1" />  
    <xsd:maxInclusive value="100000" />  
  </xsd:restriction>  
</xsd:simpleType >
```

把值限制在 1 到 100000 之间的整数

模式文档元素定义-创建复杂类型

■ 复杂类型：

- 复合类型定义本身通常包含元素声明、对其他元素的引用和属性声明。使用<xsd:element>元素声明元素，使用<xsd:attribute>元素声明属性
- 定义具有属性的元素
 - 简单类型的限制之一就是这些元素不能包含属性。为了给元素添加属性，必须把它转化成 complexType

media 元素目前有两个属性，其中一个遵循枚举的 *mediaType*

complexTypees。这需要一个 complexType

元素添加

```
<xsd:element name="media">
  <xsd:complexType>
    <xsd:attribute name="mediaid" type="xsd:integer" />
    <xsd:attribute name="status" type="mediaType" />
  </xsd:complexType>
</xsd:element>
```

模式文档元素定义-创建复杂类型

▪ choice

- 在一些情况下，只想从可供选择的元素列表中选择一个元素，对此，可以使用 choice 元素
- 定义可选择的属性以及元素

```
<xsd:complexType name="locationType">
  <xsd:choice>
    <xsd:element name="description" type="xsd:string" />
    <xsd:element name="place" type="xsd:string" />
  </xsd:choice>
</xsd:complexType>
```

不论是 **description** 元素还是 **place** 元素都可以作为任何 **locationType** 元素的子元素出现，但是不能同时出现

XSL简介

■ 问题的提出:

– XML的重要特点:

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

– 如何把一个XML文件以某种可视形式表现出来？

- 直接让浏览器或者阅读器支持
- CSS + 脚本
- 转换为HTML: XPath + XSLT
-

– 如何把一个XML文件中内容提取出来并转换为其他类型或者格式的内容？

- 程序实现: DOM, SAX...
- XSLT
-



XSL简介

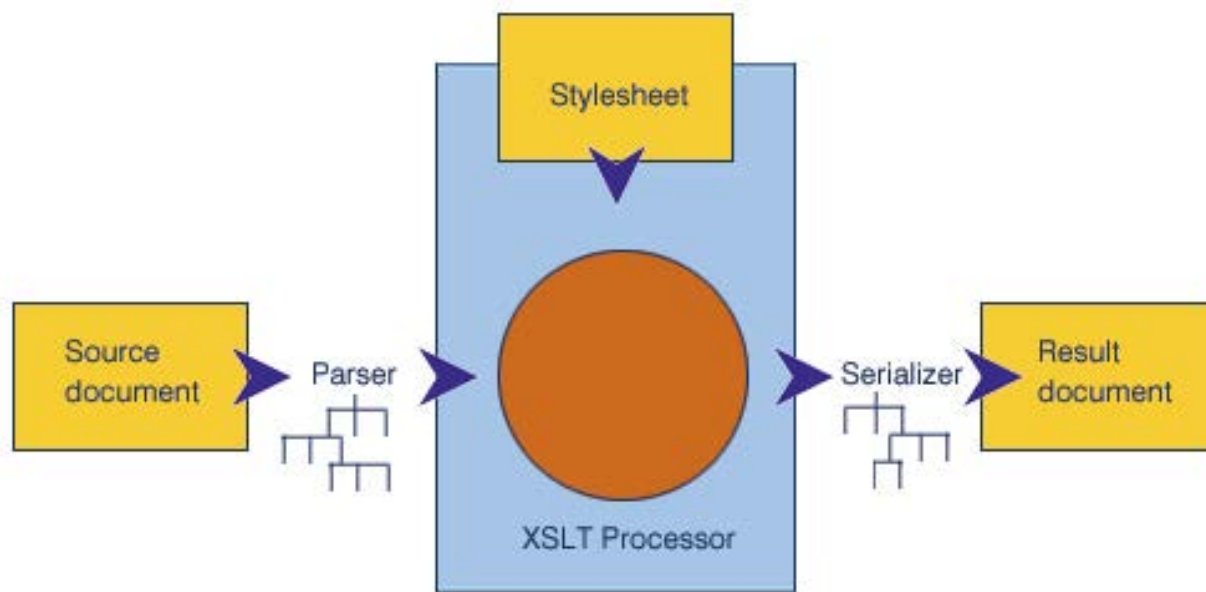
- 许多问题的解决都依赖于将一个XML文档转换为其他格式的文档, 此即XSL的产生背景
- XSL是可扩展样式语言(Extensible Stylesheet Language)的简称, XSLT 在 1999 年 11 月 16 日被确立为 W3C 标准.
Ref: <http://www.w3.org/TR/xslt>
- XSLT是一种用来转换XML文档结构的语言,一个XSL文件本身就是一个XML文档。该文档的元素是一系列规则
- XSL 包含三部分:
 - XSLT – 转换XML文档
 - XPath – 浏览定位XML文档
 - XSL-FO – 格式化XML文档

XSL简介

■ CSS的缺点

- 不能重新排序文档中的元素;
- 不能判断和控制哪个元素被显示, 哪个不被显示
- 不能统计计算元素中的数据

- XSL的功能不仅仅是应用样式; 当使用XML处理器时, XML源文档中的信息将被评价、重新安排和组装, 形成结果树 (Result Tree), 仍然是格式良好的文档



XSL简介

- 典型的XML转换:
 - XML到HTML的转换
 - XML到XML的转换

- 示例

```
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet href="SalesToHTML.xsl" type="text/xsl" ?>
<!-- SalesReport.cml -->
<SalesReport>
  <Company>XMML.com</Company>
  <Period>2001-2002</Period>
    <Sales Region="EU">50,000</Sales>
    <Sales Region="NA">150,000</Sales>
    <Sales Region="AU">10,000</Sales>
</SalesReport>
```



XSL简介

- 使用XSL进行XML转换所涉及的主要技术问题有:

- 如何从源XML文档中读出信息?

- 如何在源XML文档中定位到所需信息所在位置?

- XPath

- 如何在定位之后提取信息?

- XSLT元素

- 如何向目标XML文档中写入信息?

- XSLT元素

- 模板驱动(template-driven)模型

XSL最基本的功能是提供了一种归并功能。XSL样式表包含了一个描述结果结构的模板，并描述了如何从原文档中选取数据插入这个模板。这个归并数据的模型和模板被称为是**模板驱动**

XSL简介

■ 三种方式变换XML文档

- 服务器上

- 服务器程序完成变换后发送到客户端程序
Servlet: IBM XML Enabler

- 客户端

- IE

- 应用独立的程序

- James Clark的XT
- IBM LotusXSL

Xpath

- XPath是一种专门用来在XML文档中查找信息的语言，提供了一种简单的在Xml文档中鉴别节点的查询语言，这种语言基于节点的类型、名称、数值以及文档中节点与节点之间的关系
- XPath主要与XSLT一起使用,另外与XPointer, XML Schema, XQuery等技术有密切联系
- XPath是基于XML文档的层次结构的,体现了XML文档的抽象逻辑结构

In relational databases, parts of a database can be selected and retrieved using query languages such as SQL. The same is true for XML documents, for which there exist a number of proposals for query languages, such as XQL, XML-QL, and XQuery.

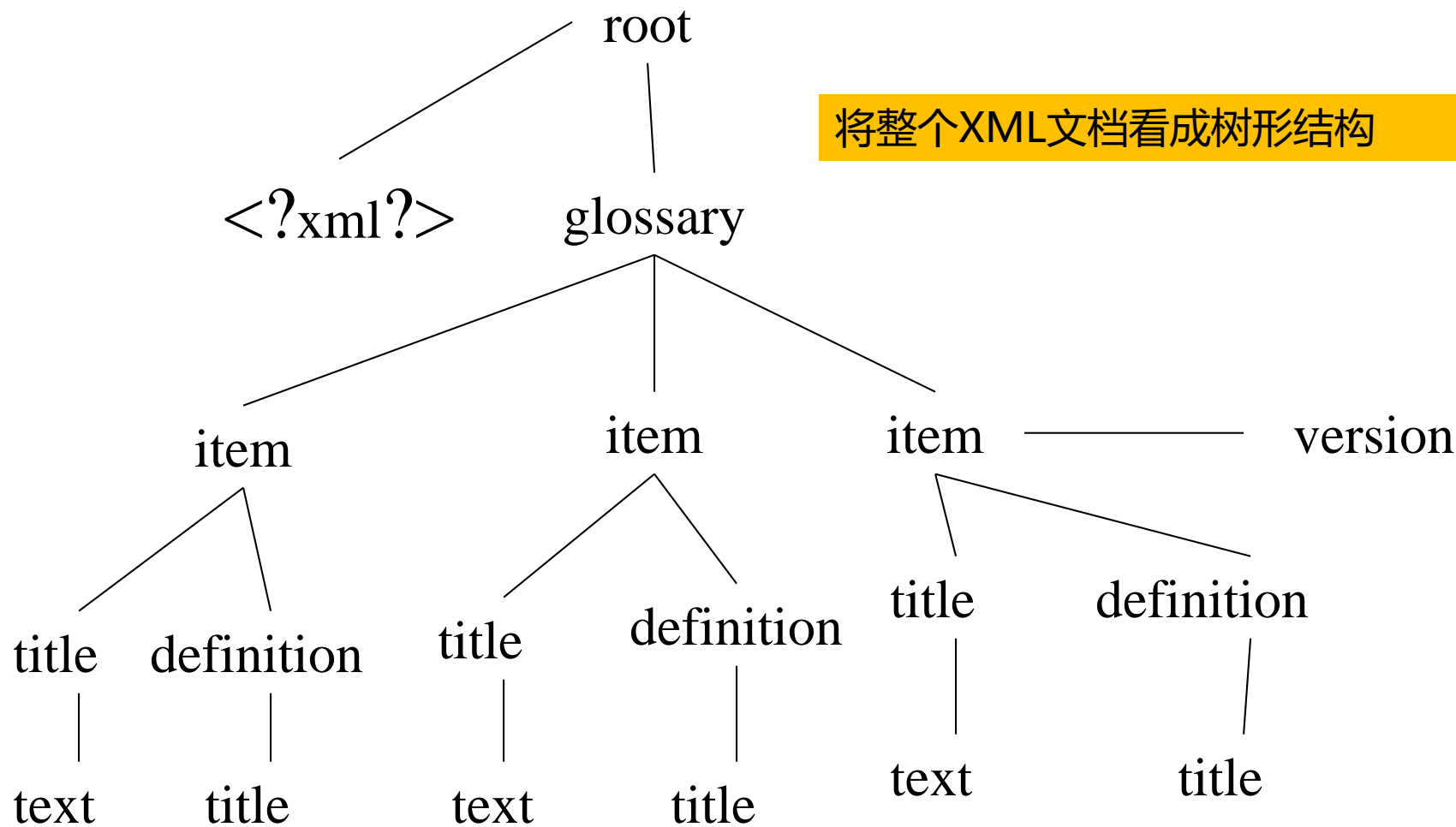
The central concept of XML query languages is a *path expression* that specifies how a node, or a set of nodes, in the tree representation of the XML document can be reached.

Xpath

```
<?xml version="1.0" encoding="UTF-8"?>
<glossary>
  <item>
    <title> text </title>
    <definition><title></title></definition>
  </item>
  <item>
    <title> text </title>
    <definition><title></title></definition>
  </item>
  <item version="9.9" >
    <title> text </title>
    <definition><title></title></definition>
  </item>
</glossary>
```


Xpath

■ XML文档树形结构



Xpath

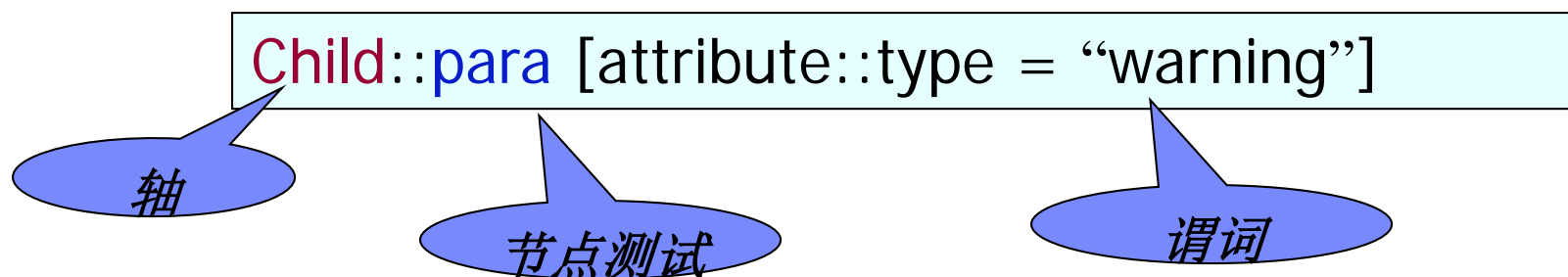
■ Xpath描述特点

- 在一个XML文档中，层次由元素和其他节点组成；在每一层中元素名称不是唯一的，Xpath能鉴别所有的匹配元素的集合
- Xpath描述用“/”分开的，贯穿于Xml树的路径，直到所指的元素名。XSL Pattern会使识别所有匹配这条路径的元素。例：
authors/author/name. (2.0开始采用括号序列的形式)
- Xpath还能含有通配符，可用于描述未知的元素。任何名称的元素都能用“*”号描述。
例：authors/*/name

XPath

■ Xpath定位节点

- 由轴和节点测试产生初始节点集,生成节点与基准节点的关系由轴规定,节点类型和扩展名称由节点测试来规定
- 依次通过谓词的限制对初始节点集进行选择 and 过滤.



XPath

■ XPath实例

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

```
<glossary>
```

```
  <item>
```

```
    <title> text </title>
```

```
    <definition><title></title></definition>
```

```
  </item>
```

`/glossary/item[position() = 1]`

`/glossary/item/definition/title`

XPath

<item>

<title> text </title>

<definition><title></title></definition>

全部item
/glossary/item

</item>

<item version="9.9" >

<title> text </title>

<definition><title></title></definition>

在此item位置
选attribute node
./@version

</item>

</glossary>

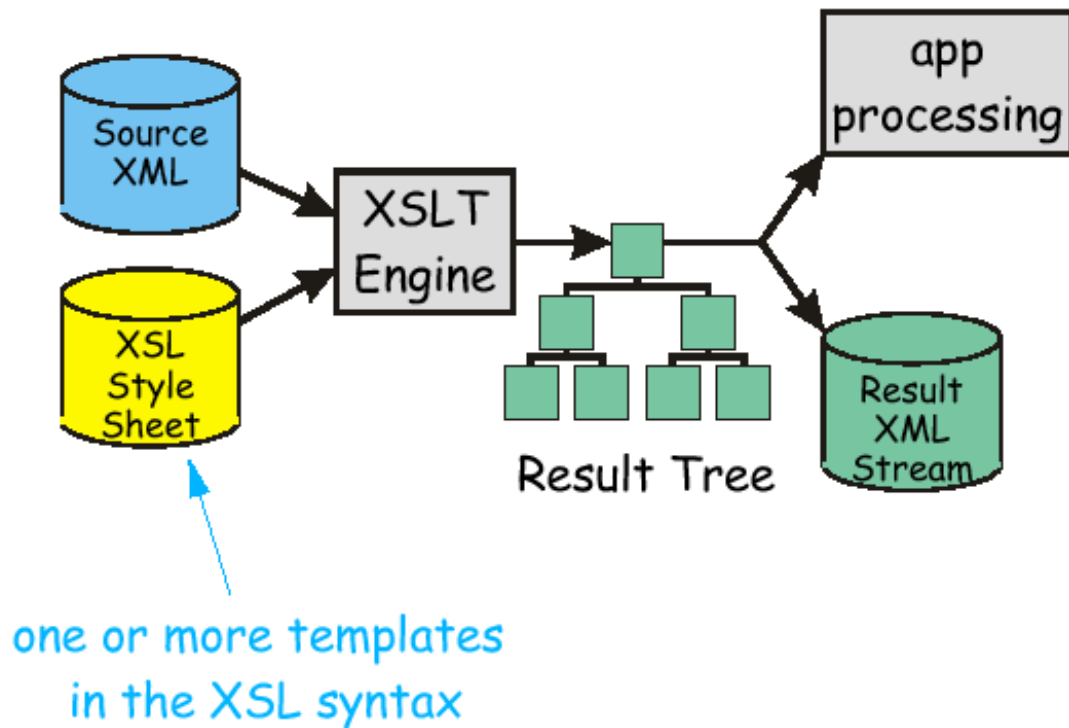


XSLT

- XSLT是基于XML语法的,本身是一个XML词汇表,包括37个元素
- XSLT使用XPath技术解决如下问题:
 - 匹配节点以执行模板
 - 比较节点集合以控制条件XSLT元素的执行
 - 选择节点集合来改变当前的上下文并且使执行流过整个源文档
 - 选择节点集内容,产生文本内容输出

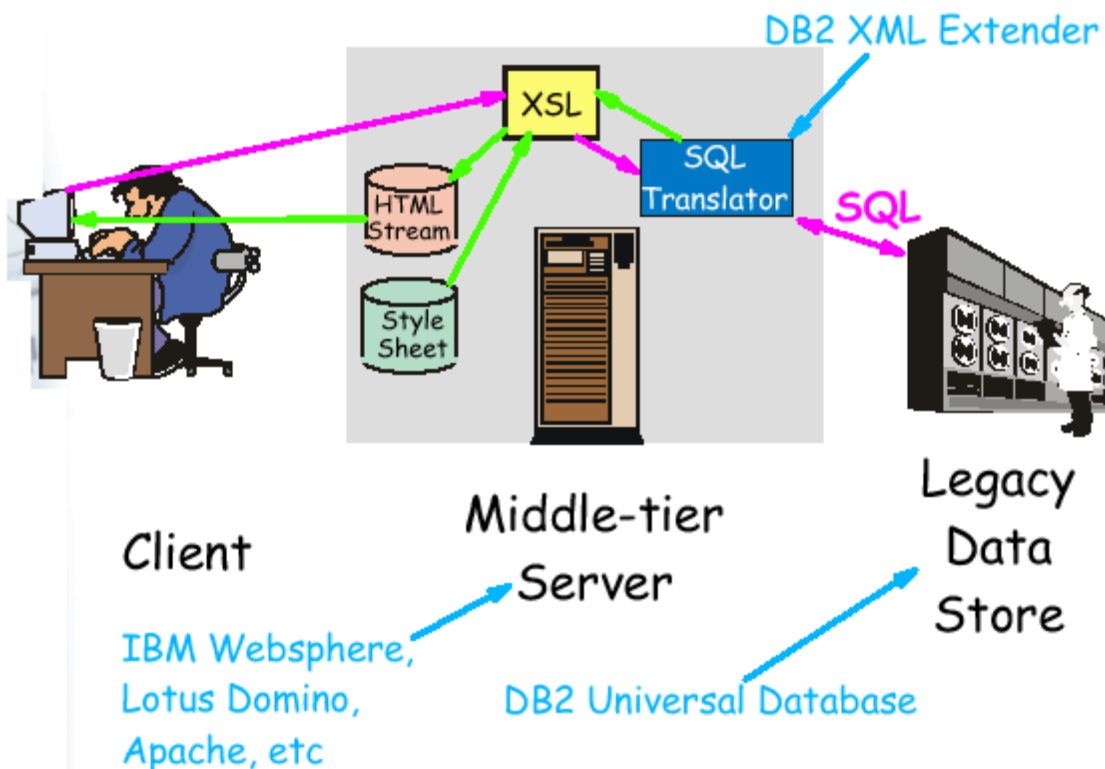
XSLT

- XSL 处理



XSLT

利用XML+XSL 实现与数据库的交互.



Sql="select * from 订购人 where 种类='个人' "

//客户[@种类="个人"]

XSLT

■ 在XML文件中链接样式表:

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

- *type* 属性值描述了样式表应用的具体类型: “*text/css*”表示是一个CSS样式表; “*text/xsl*”表示使用了一个XSI样式表匹配节点以执行模板
- *href* 属性值给出了样式表的URI, 相对URI是指相对于xml文档的URI

■ XSLT文件声明

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

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

XSLT

■ 一个XSLT文件的基本结构

- 由于一个XSL样式表示本身必须是一个XML文件，因此，XSL文件也应该从XML定义开始。
- 根元素<xsl:stylesheet>表明这是一个样式表文件，其中xmlns:xsl属性值提供了定义XSL名域的URI。
- 空元素xsl:output 中的encoding属性可用来指定输出的HTML使用的编码。

```
<xsl:output encoding="GB2312"/>
```

- 用<xsl:template match="/">元素把一个模板包裹起来，其中match属性值指明了该模板匹配的XML源文档的根（根元素的上一级）

XSLT

使用时记得在元素名前加xsl命名空间标识符

■ XSLT的元素可以根据作用分为以下几个大类:

元素类型	元素名称	元素类型	元素名称
XSLT文档结构元素:	<ul style="list-style-type: none"> ● stylesheet transform ● template 	流程控制元素:	<ul style="list-style-type: none"> ● if ● choose when otherwise ● for-each
输出类型指定元素:	<ul style="list-style-type: none"> ● output ● preserve-space, strip-space 	模块化元素:	<ul style="list-style-type: none"> ● apply-template ● with-param ● sort ● call-template ● param ● include ● import ● variable
简单输出元素:	<ul style="list-style-type: none"> ● value-of ● copy, copy-of ● attribute ● comment ● element ● processing-instruction ● text 		

```
<?xml version="1.0" encoding="UTF-8" ?>
<?xml-stylesheet href="catalog.xsl" type="text/xsl" ?>
<catalog>
  <cd>
    <title>Empire of the Senses</title>
    <artist>Bob Dylan</artist>
    <country>USA</country>
    <company>Columbia</company>
    <price>10.90</price>
    <year>1985</year>
  </cd>
  .
  .
  .
</catalog>
```

```
- <xsl:stylesheet version="1.0">
  <xsl:output method="html" version="1.0" encoding="UTF-8" indent="yes"/>
- <xsl:template match="/">
  - <html>
    - <body>
      <h2>My CD Collection</h2>
      - <table border="1">
        - <tr bgcolor="#9acd32">
          <th align="left">Title</th>
          <th align="left">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>
```

XSLT

- A example

Title	Artist
Empire Burlesque	Bob Dylan
Hide your heart	Bonnie Tyler
Greatest Hits	Dolly Parton
Still got the blues	Gary Moore
Eros	Eros Ramazzotti
One night only	Bee Gees
Sylvias Mother	Dr.Hook
Maggie May	Rod Stewart
Romanza	Andrea Bocelli
When a man loves a woman	Percy Sledge
Black angel	Savage Rose
1999 Grammy Nominees	Many
For the good times	Kenny Rogers
Big Willie style	Will Smith
Tupelo Honey	Van Morrison
Soulsville	Jorn Hoel
The very best of	Cat Stevens

```

<xsl:for-each select="catalog/cd">
  <tr>
    <td><xsl:value-of select="title"/></td>
    <xsl:choose>
      <xsl:when test="price > 10">
        <td bgcolor="#ff00ff">
          <xsl:value-of select="artist"/> </td>
        </xsl:when>
      <xsl:when test="price > 9">
        <td bgcolor="#cccccc">
          <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>

```

DOM&SAX简介

- 应用程序需要和XML文档进行交互。通常使用现成的解析器，而不自行编写
- **解析器**是设计用来分析XML 文档 — 并对信息做一些特定的事情的软件应用程序，SAX和DOM是解析器所采用的两种重要的解析API
- **SAX** 这种基于事件的 API 中，解析器发送一些事件给某类侦听器。在如 **DOM** 这种基于树的 API 中，解析器在内存中构建数据树
- **DOM** (Document Object Model) ,由w3c制定
- www.xml.org XML-DEV邮件列表成员共同开发了SAX

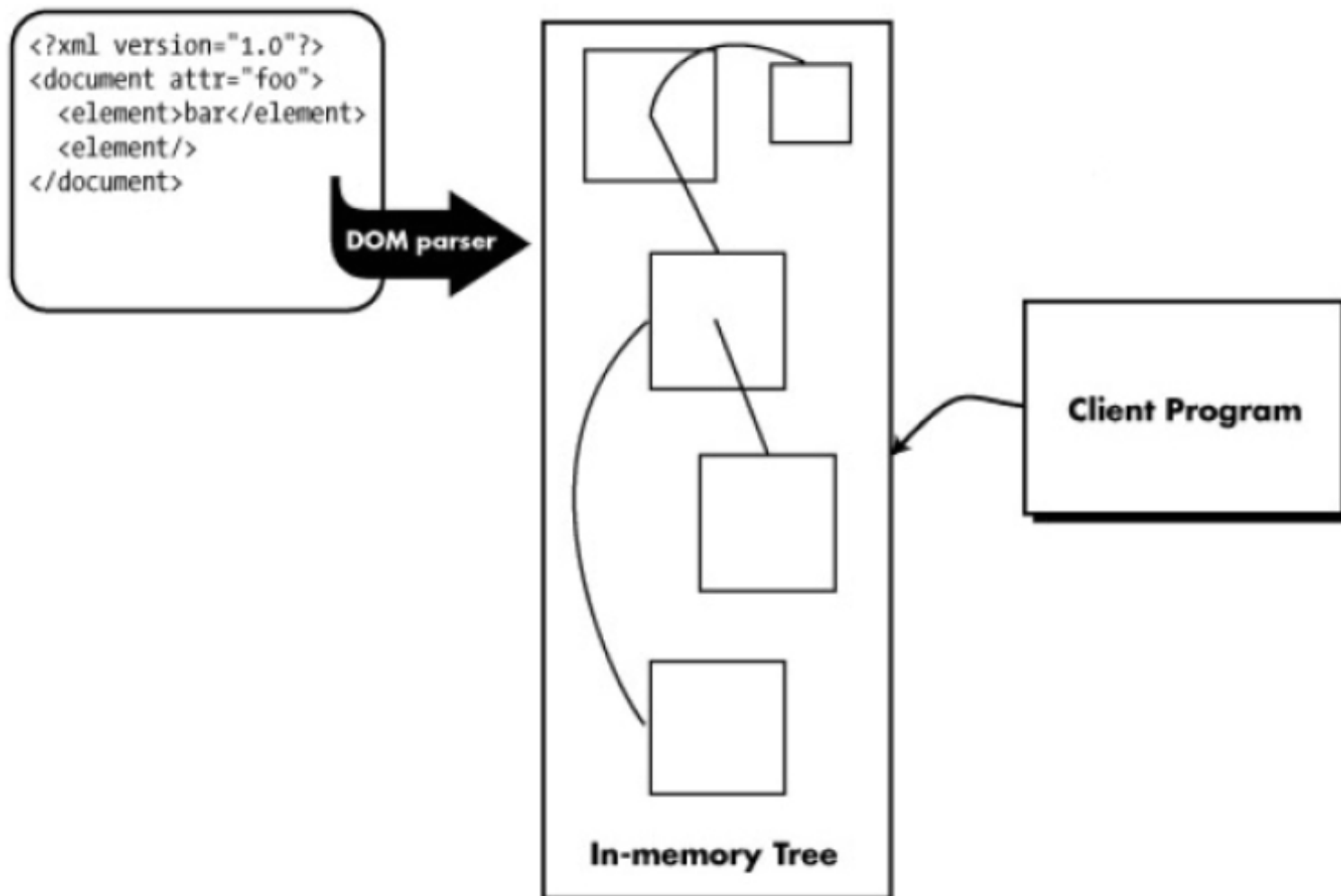
DOM

■ DOM简介

- XML 文档具有称为 *节点* 的信息单元的层次结构； “文档对象模型（DOM）” 是描述节点和节点间关系的方式。
- 由于 DOM 是基于信息的层次结构，因此它被称为是 *基于树的*
- 整棵对象树建立在内存中,性能较低
- 文档有较好随机访问性能
- DOM 是与语言 and 平台无关的、设计用来使用 XML 数据的 API。提供了创建、修改、删除以及重新排序节点的功能
- DOM 实现可以有Java， C++、Perl 和其它语言的版本

DOM

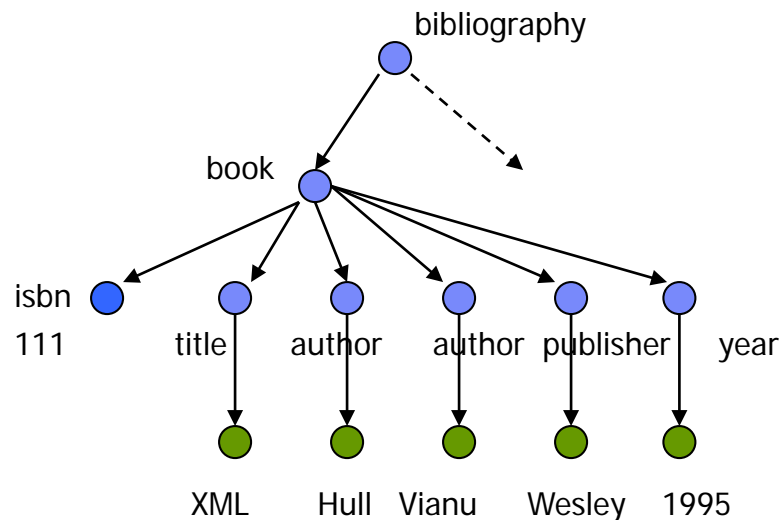
■ 基于树的XML应用程序典型流程



DOM

– 树模型示例

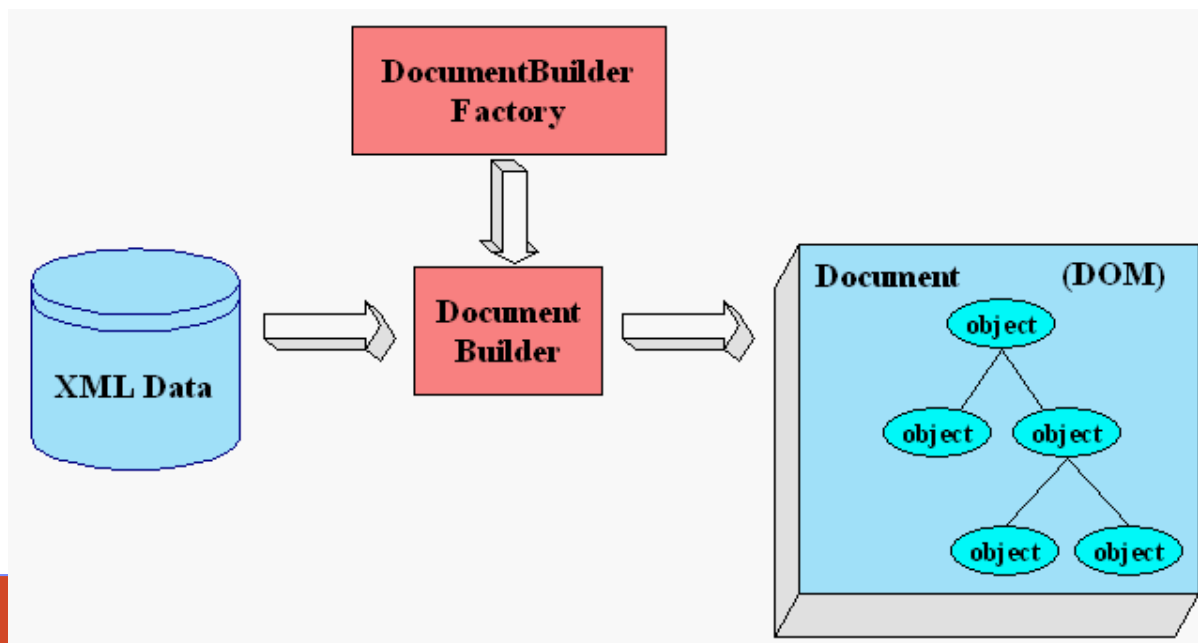
```
<bibliography>  
  <book isbn="111">  
    <title> XML </title>  
    <author> Hull </author>  
    <author> Vianu </author>  
    <publisher> Wesley </publisher>  
    <year> 1995 </year>  
  </book>  
  ...  
</bibliography>
```



DOM

■ 解析的三步骤：

- 创建 DocumentBuilderFactory。该对象将创建 DocumentBuilder
- 创建 DocumentBuilder。 DocumentBuilder 将实际进行解析以创建 Document 对象
- 解析该文件以创建 Document 对象



DOM

■ A example

```
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import java.io.File;
import org.w3c.dom.Document;

public class OrderProcessor {
    public static void main (String args[]) {
        File docFile = new File("orders.xml");
        Document doc = null;
        try {
            DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
            DocumentBuilder db = dbf.newDocumentBuilder();
            doc = db.parse(docFile);
        } catch (Exception e) {
            System.out.print("Problem parsing the file.");
        }
    }
}
```

DOM

■ 遍历

- 一旦解析完文档，并且创建了 Document 之后，应用程序可以遍历结构来复查、查找或显示信息
- 获取根元素:遍历文档是从根元素开始。格式良好的文档只有一个根元素

```
import org.w3c.dom.Element;
```

```
...  
Element root = doc.getDocumentElement();  
System.out.println("The root element is " +  
root.getNodeName());  
...
```

DOM

■ 遍历

– 获取节点的子节点

- 一旦应用程序确定了根元素，则它检索根元素的所有子节点，形成一个 `NodeList` 的列表

```
...  
import org.w3c.dom.NodeList;  
...  
//STEP 1: Get the root element  
Element root = doc.getDocumentElement();  
System.out.println("The root element is "+root.getNodeName());  
//STEP 2: Get the children  
NodeList children = root.getChildNodes();  
System.out.println("There are "+children.getLength() +" nodes in this document");  
}  
}
```

DOM

■ 编辑文档

– 添加节点

- `createTextNode` 方法创建一个新的文本节点
- `createElement` 方法创建一个新的元素节点
- `appendChild` 方法追加子节点到元素中
- `insertBefore` 方法将新元素插入到指定元素前

```
...
```

```
Node totalNode = doc.createTextNode("Hello");
```

```
Element totalElement = doc.createElement("NEW_ELEMENT");
```

```
totalElement.appendChild(totalNode);
```

```
thisOrder.insertBefore(totalElement, thisOrder.getFirstChild());
```

```
...
```

SAX

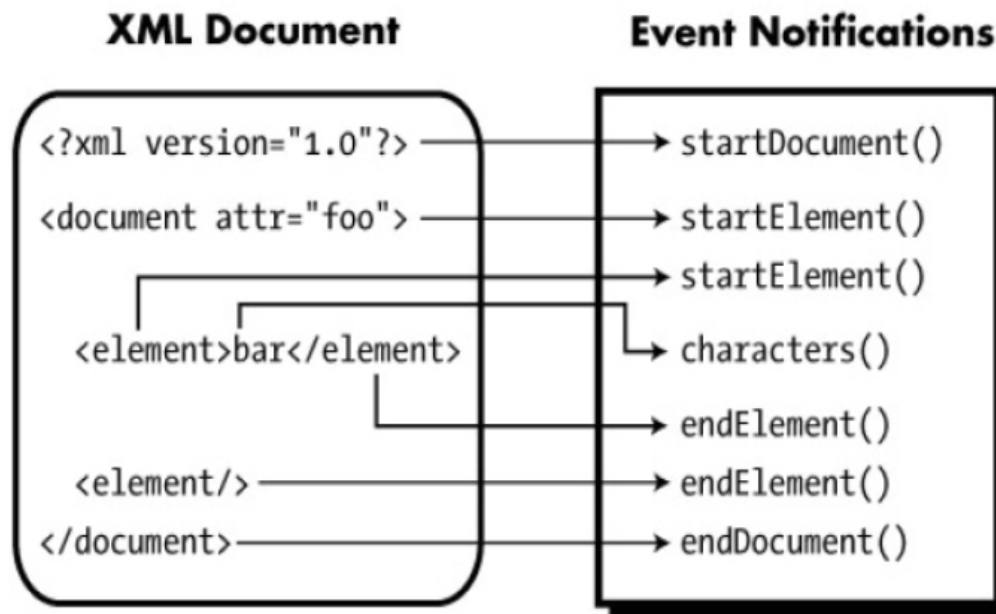
■ SAX (Simple API for XML)简介

- 顺序遍历文档,然后产生所有节点事件,随后在内存中只保留当前处理部分
- 非W3C推荐标准,由XML-DEV邮件列表成员创建,属公共领域软件
- 最早用Java开发, JAXP。有其他语言的版本: 如Perl、C++ 和 Python
 - <http://www.megginson.com/SAX/applications.html>
- 虽然原始文档保持不变,但 SAX 提供了操纵数据的方法,然后将该方法导向另一个过程或文档
- 适于:
 - 在大数据量的情况下, SAX特别有优势
 - 在文档中检索一个特定的值
 - 创建文档子集,以便后继DOM处理

SAX

■ 基于事件的机制

- SAX是一个事件驱动的API。SAX 分析经过其的 XML 流，并产生相应的事件。
- SAX API 允许开发者捕获这些事件，并对它们进行操作。应用程序通过注册自身，以接受XML解析器遇到文档不同部分而发出的信息



SAX的解析过程

<POEM>

<AUTHOR>Ogden Nash</AUTHOR>

<TITLE>Fleas</TITLE>

<LINE>Adam</LINE>

</POEM>

遇到的项目	方法回调
{文档开始}	startDocument()
<POEM>	startElement(null, "POEM", null, {Attributes})
"\n"	characters("<POEM>\n...", 6, 1)
<AUTHOR>	startElement(null, "AUTHOR", null, {Attributes})
"Ogden Nash"	characters("<POEM>\n...", 15, 10)
</AUTHOR>	endElement(null, "AUTHOR", null)
"\n"	characters("<POEM>\n...", 34, 1)
<TITLE>	startElement(null, "TITLE", null, {Attributes})
"Fleas"	characters("<POEM>\n...", 42, 5)
</TITLE>	endElement(null, "TITLE", null)
"\n"	characters("<POEM>\n...", 55, 1)
<LINE>	startElement(null, "LINE", null, {Attributes})
"Adam"	characters("<POEM>\n...", 62, 4)
</LINE>	endElement(null, "LINE", null)
"\n"	characters("<POEM>\n...", 67, 1)
</POEM>	endElement(null, "POEM", null)
{文档结束}	endDocument()

SAX

■ SAX处理机制的优缺点

– 优点 😊

- 在事件驱动模型下，API不保存任何文档的内容，相关的内容被传给事件相关的方法。所以需要的内存与XML文件大小关系不大
- 使用SAX，应用程序不需要等到文档解析结束，就可以开始相应的处理(类似流媒体)

– 缺点

- 😞 不允许XML文档的随机访问
- 程序处理比DOM复杂
- 不可能对数据进行更改，或者“返回”至数据流中前面的数据。

SAX

■ 四个核心SAX接口

- ContentHandler

- 处理文档的内容，比如元素和操作指令

- DTDHandler

- 处理DTD事件

- EntityResolver

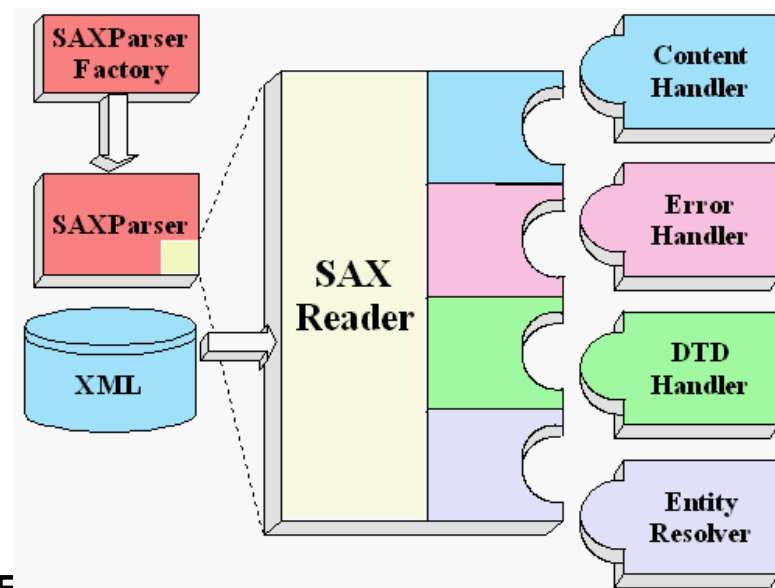
- 处理外部实体

- ErrorHandler

- 处理在解析时候出现的错误

- 回调（callback）方法

- SAX解析器可调用它来通知在解析过程中遇到的各种事件。



SAX

■ 一个完整实例

```
1 import java.io.File;
2 import javax.xml.parsers.SAXParserFactory;
3 import javax.xml.parsers.SAXParser;
4 import org.xml.sax.Attributes;
5 import org.xml.sax.helpers.DefaultHandler;
6
7 public class BasicSAX {
8     public static void main(String[] args) {
9         try {
10             SAXParserFactory factory = SAXParserFactory.newInstance();
11             factory.setValidating(false);
12             SAXParser saxParser=factory.newSAXParser();
13             saxParser.parse(new File(args[0]), new MyHandler());
14         } catch (Exception e) {
15             //错误处理
16         }
17     }
18 }
19 class MyHandler extends DefaultHandler {
20     public void startDocument() {
21         System.out.println("\n Start Document\n");
22     }
23     public void startElement(String namespaceURI, String localName,
24         String qualifiedName, Attributes elementAttributes) {
25         System.out.println("Start Element-> " + qualifiedName);
26     }
27     public void endElement(String namespaceURI, String localName,
28         String qualifiedName) {
29         System.out.println("End Element-> " + qualifiedName);
30     }
31     public void endDocument() {
32         System.out.println("\n End Document\n");
33     }
34 }
35
```

DOM&SAX

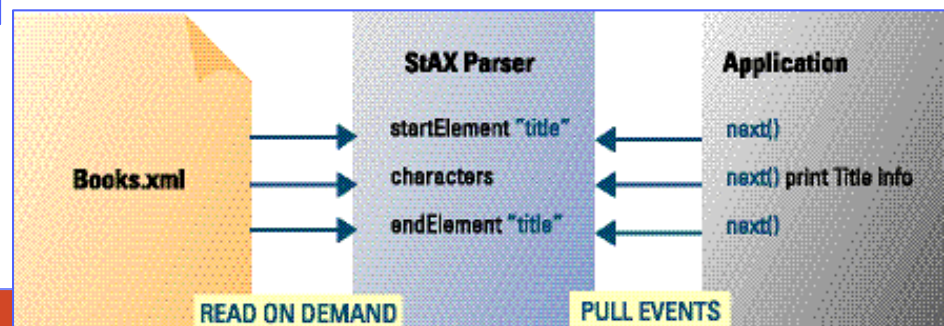
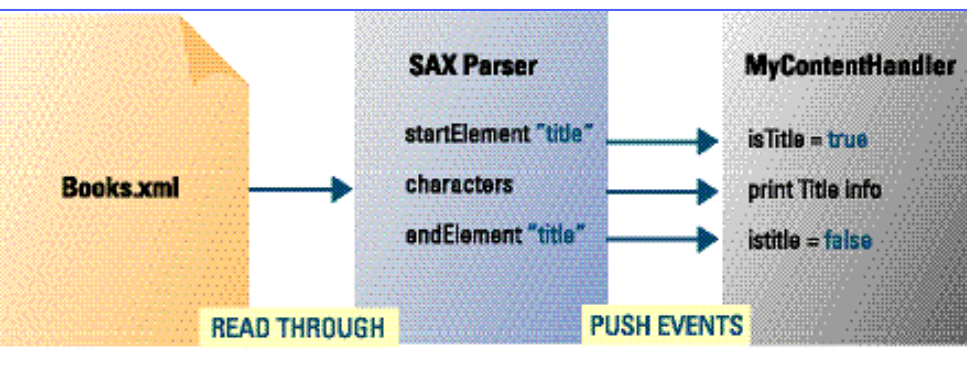
■ DOM与SAX应用场合选择

- 应用程序的目的：如果必须对数据进行更改，并且作为 XML 将它输出，则在大多数情况下，使用 DOM。与使用 XSL 转换来完成的简单结构更改不一样，如果是对数据本身进行更改，则尤其应该使用 DOM。
- 数据的数量：对于大文件，SAX 是更好的选择。
- 将如何使用数据：如果实际上只使用一小部分数据，则使用 SAX 将数据抽取到应用程序中，这种方法更好些。另一方面，如果知道将需要向后引用已经处理过的信息，则 SAX 可能不是正确的选择。
- 需要速度：通常，SAX 实现比 DOM 实现快。
- SAX 和 DOM 不是互斥的，可以相互结合使用

StAX (Streaming API for XML)

■ 简介

- JSR-173 提出了一种面向流的解析方式
- 使用一种事件驱动的模式，使用“拉”模型进行事件处理
- SAX向ContentHandler返回不同类型的事件，StAX将事件返回给应用程序，甚至可以以对象的形式提供事件



StAX (Streaming API for XML)

- **StAX 对象**

- **XMLInputFactory**

- **XMLStreamConstants**

- 表示当前指针所指向标记（或事件）的类型

- XMLStreamConstants.START_ELEMENT

- XMLStreamConstants.END_ELEMENT

- XMLStreamConstants.START_DOCUMENT

- XMLStreamConstants.END_DOCUMENT

- ...

- **XMLStreamReader**

- 基于指针的API的接口

- **XMLEventReader和XMLEvent**

- 迭代程序模型

StAX

■ 两个解析模型

– 指:

```
// 列出所有用户名称
public static void listNames() {
    XMLInputFactory factory = XMLInputFactory.newFactory();
    XMLStreamReader reader = null;
    try {
        reader = factory.createXMLStreamReader(new FileReader("users.xml"));
    } catch (Exception e) {
        e.printStackTrace();
    }
    // 遍历XML文档
    try {
        while (reader.hasNext()) {
            int event = reader.next();
            // 如果是元素的开始
            if (event == XMLStreamConstants.START_ELEMENT) {
                // 列出所有用户名称
                if ("user".equalsIgnoreCase(reader.getLocalName())) {
                    System.out.println("Name:"
                        + reader.getAttributeValue(null, "name"));
                }
            }
        }
        reader.close();
    } catch (XMLStreamException e) {
        e.printStackTrace();
    }
}
```


StAX

两个迭代器

-
-

后

```

public static void listAllByXMLeventReader() {
    String xmlFile = "users.xml";
    XMLInputFactory factory = XMLInputFactory.newInstance();
    try {
        // 创建基于迭代器的事件读取器对象
        XMLeventReader reader = factory
            .createXMLeventReader(new FileReader(xmlFile));
        // 遍历XML文档
        while (reader.hasNext()) {
            XMLevent event = reader.nextEvent();
            // 如果事件对象是元素的开始
            if (event.isStartElement()) {
                // 转换成开始元素事件对象
                StartElement start = event.asStartElement();
                // 打印元素标签的本地名称
                System.out.print(start.getName().getLocalPart());
                // 取得所有属性
                Iterator attrs = start.getAttributes();
                while (attrs.hasNext()) {
                    // 打印所有属性信息
                    Attribute attr = (Attribute) attrs.next();
                    System.out.print(":" + attr.getName().getLocalPart()
                        + "=" + attr.getValue());
                }
                System.out.println();
            }
        }
        reader.close();
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (XMLStreamException e) {
        e.printStackTrace();
    }
}

```

Demo stAX两种方式读取，以及写出XML

XML的三种典型解析方式比较

技术	优点	缺点	最适于...
DOM解析	<ul style="list-style-type: none">•易于使用•丰富的API集合，可用于轻松地导航•整棵树加载到内存，允许对XML文档进行随机访问	<ul style="list-style-type: none">•整个XML文档必须一次解析完•将整棵树加载到内存成本较高•一般的DOM节点对于必须为所有节点创建对象的对象类型绑定不太理想	<ul style="list-style-type: none">•需要修改XML文档的应用程序或XSLT应用程序(不可用于只读XML的应用程序)
SAX解析	<ul style="list-style-type: none">•无需将整个文档加载到内存，因而内存消耗少•推模型允许注册多个ContentHandler	<ul style="list-style-type: none">•没有内置的文档导航支持•不能够随机访问XML文档•不支持在原地修改XML•不支持名字空间作用域	<ul style="list-style-type: none">•只从XML读取数据的应用程序（不可用于操作或修改XML文档）
StAX解析	<ul style="list-style-type: none">•提供两种解析模型，分别出于简单和性能的考虑•应用程序控制解析，轻松支持多个输入•强大的过滤功能提供高效数据获取	<ul style="list-style-type: none">•没有内置的导航支持•不能够随机访问XML文档•不能在原地修改XML	<ul style="list-style-type: none">•需要数据流模型和支持名字空间的应用程序(不可用于操作或修改XML文档)

XML应用概述

■ Web发布

- XHTML（可扩展超文本标记语言）是作为一种 XML 应用被重新定义的 HTML；MIME是application/xhtml+xml
 - 德拉克式错误处理：文档必须是良构的；所有的标签必须小写；
<!DOCTYPE> 是强制使用的
 - XHTML 1.0 Strict: **div+css**

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

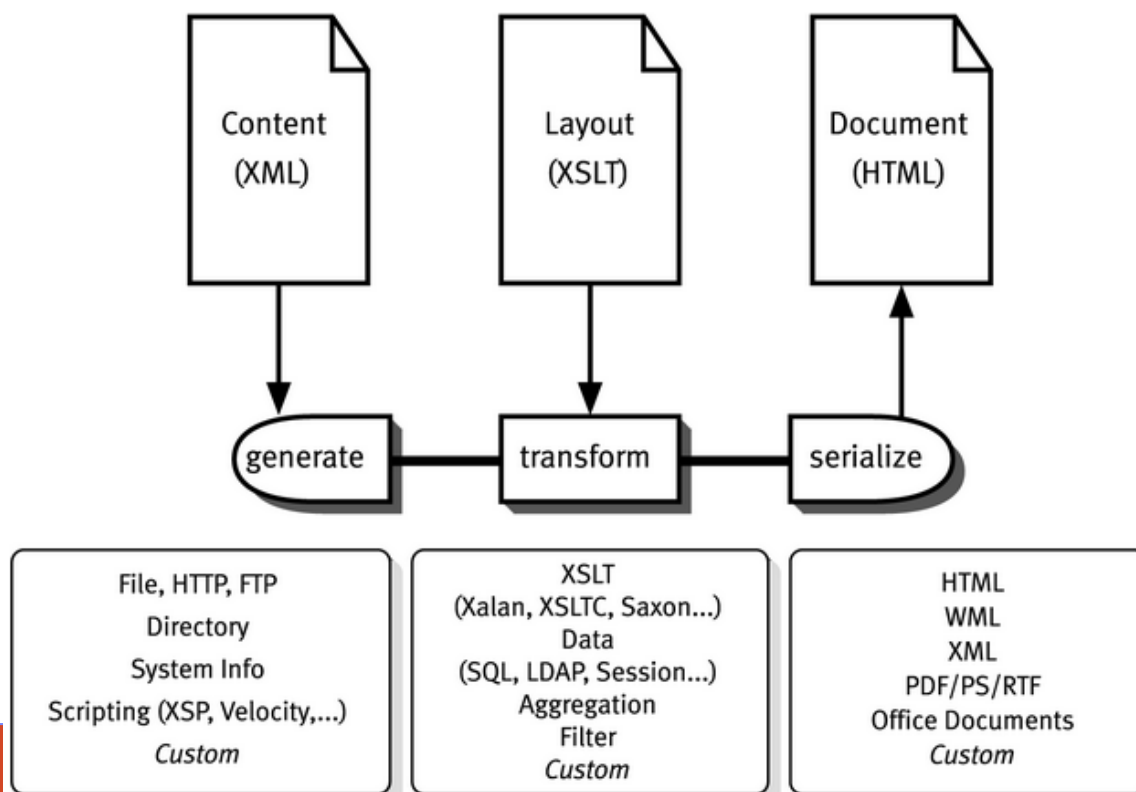
■ XHTML 1.0 Transitional

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

XML应用概述

■ 媒体无关的发布

- 在文档管理中，XML最大优势是可直接在web上显示XML数据，工具丰富。面向多种形态媒体可输出同一数据。
- 基于XML的网站管理 Cocoon



XML应用概述

■ 企业间电子商务

- ebXML

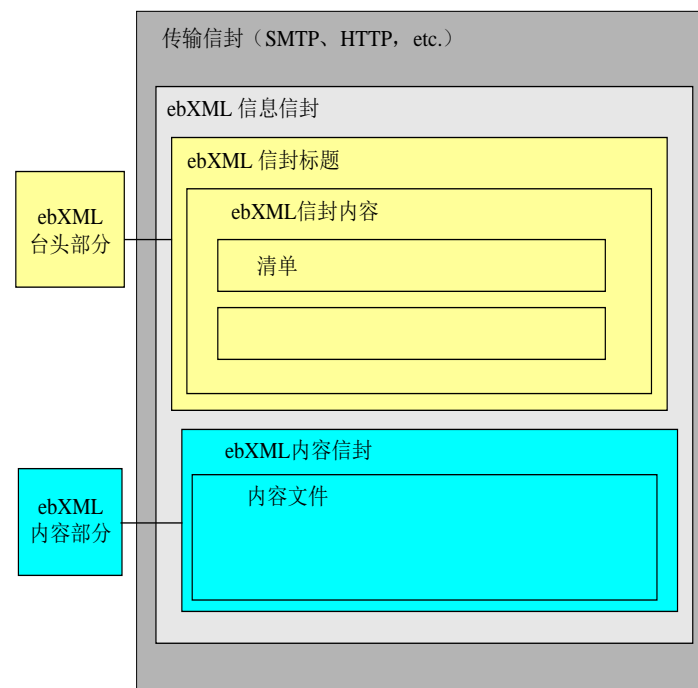
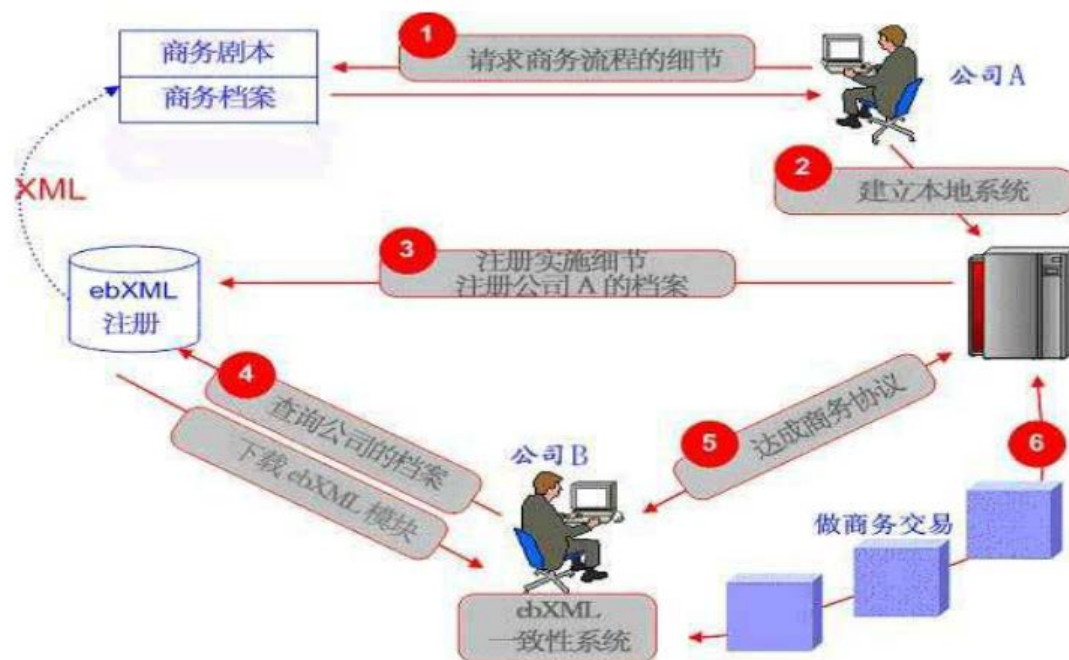
- 99年开始由Oasis等组织推动的企业间一套相互关联的电子商务标准，能够形成一个完整的电子商务框架模块
- 提供一个安全一致的XML底层框架，解决传统EDI实施中的问题
 - 通过专有信息网络传输
 - 需要压缩数据形式以减少带宽的使用
 - 语法复杂，撰写程序处理EDI信息比较困难
 - 费用高，覆盖面小

XML应用概述

■ 企业间电子商务

- ebXML

- 定义了高层商务流程和底层信息结构



XML应用概述

■ 企业间电子商务

- ebXML

```
<CollaborationProtocolProfile
xmlns="http://www.ebxml.org/namespaces/tradePartner"
xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
xmlns:xlink="http://www.w3.org/1999/xlink"
version="1.1">
  <PartyInfo>  <!--one or more-->
    ...
  </PartyInfo>
  <Packaging id="ID"> <!--one or more-->
    ...
  <Packaging>
    <ds:Signature>  <!--zero or one-->
      ...
    </ds:Signature>
    <Comment>text</Comment> <!--zero or more-->
  </CollaborationProtocolProfile>
```

PartyInfo元素提供有关组织的详细信息

Packaging 元素提供了有关信息台头和内容的封装方法

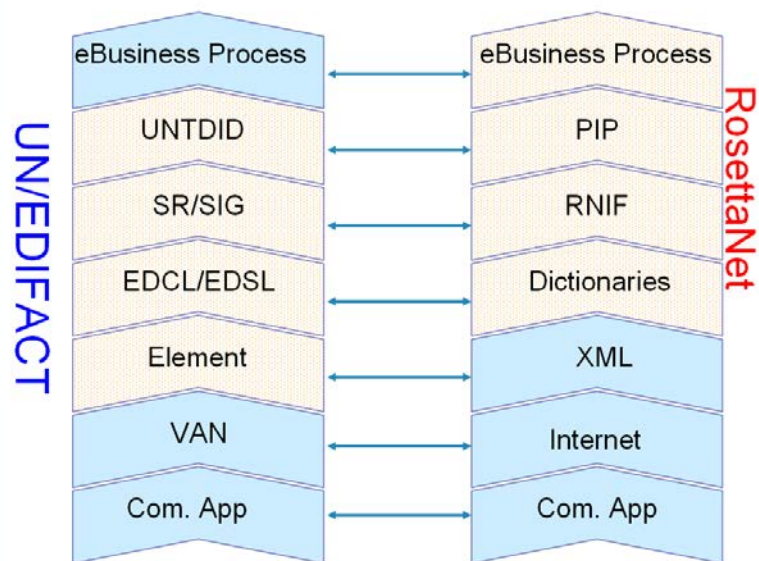
Comment元素用于记录交易者希望达到的各种意愿

XML应用概述

■ 企业间电子商务：

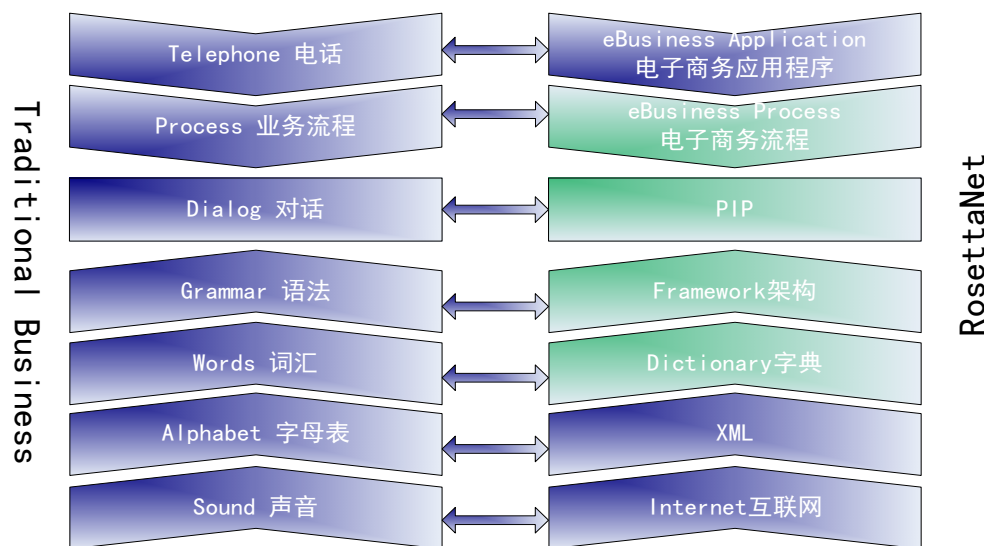
- EDI：异构信息系统的数据对接
- RosettaNet：致力于制定种种标准，使供应链上的伙伴能够相互沟通信息

GRADUATION PAPER FOR MSE 03FALL 033053122



■ RosettaNet和EDI构架之对比

Copyright Chen YiBing 2005



从EDI到RosettaNet

和传统商务的类比

XML应用概述

■ 传输协议和远程调用

- Ajax调用返回的数据
- 各种Web API返回的数据

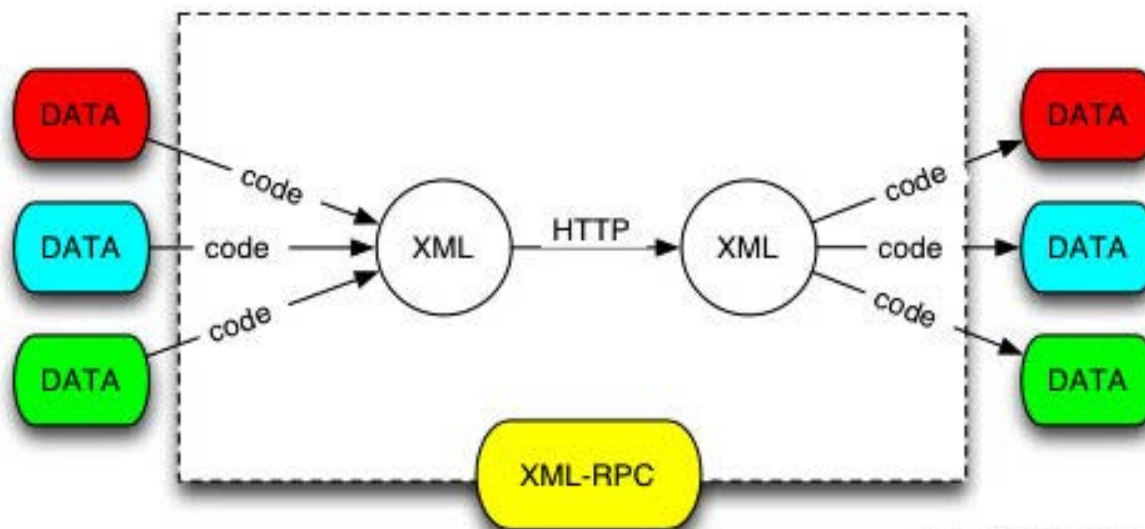
```
<xml>
  <ToUserName><![CDATA[toUser]]></ToUserName>
  <FromUserName><![CDATA[fromUser]]></FromUserName>
  <CreateTime>1348831860</CreateTime>
  <MsgType><![CDATA[text]]></MsgType>
  <Content><![CDATA[this is a test]]></Content>
  <MsgId>1234567890123456</MsgId>
</xml>
```

微信文本消息

XML应用概述

■ 传输协议和远程调用

- 平台独立，很方便的进行应用程序级的定义和表示协议
- 各种传输协议
 - web service中的SOAP调用
 - **xml-rpc**是一套允许运行在不同操作系统、不同环境的程序实现基于internet过程调用的规范和一系列的实现

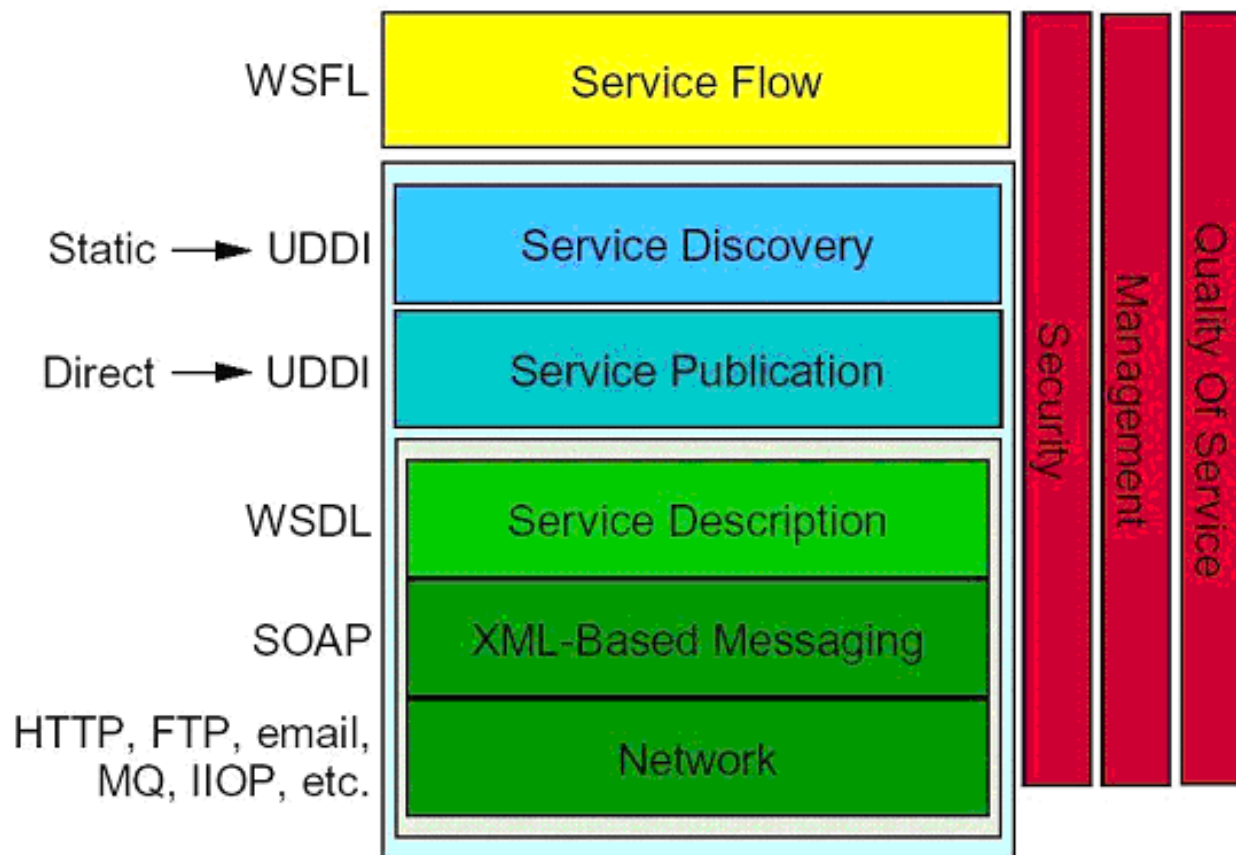


Source: JY Stervinou

XML应用概述

■ 传输协议和远程调用

- Web 服务协议栈全部采用XML作为标准



XML应用概述

■ 传输协议和远程调用

– SOAP请求

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="..."
  SOAP-ENV:encodingStyle="...">
  <SOAP-ENV:Header>
    <!-- Optional context information -->
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <m:GetLastTradePrice xmlns:m="some_URI">
      <tickerSymbol>SUNW</tickerSymbol>
    </m:GetLastTradePrice>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Method
name

Parameters

XML应用概述

■ 传输协议和远程调用

– SOAP响应

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="..."
  SOAP-ENV:encodingStyle="...">
  <SOAP-ENV:Header>
    <!-- Optional context information -->
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <m:GetLastTradePriceResponse xmlns:m="some_URI">
      <price>100.5</price>
    </m:GetLastTradePriceResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

XML应用概述

- 传输协议和远程调用
 - 绑定到HTTP的SOAP请求

POST http://www.SmartHello.com/HelloApplication HTTP/1.0

Content-Type: text/xml; charset="utf-8"

Content-Length: 587

SOAPAction: <http://www.SmartHello.com/HelloApplication#sayHelloTo>

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="
  http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <SOAP-ENV:Header>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <ns1:sayHelloTo xmlns:ns1="Hello" SOAP-ENV:
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <name xsi:type="xsd:string">
        Tarak
      </name>
    </ns1:sayHelloTo>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

XML应用概述

- 传输协议和远程调用
 - 绑定到HTTP的SOAP响应

HTTP/1.0 200 OK

Content-Type: text/xml; charset="utf-8"

Content-Length: 615

```
<SOAP-ENV:Envelope xmlns:SOAP-
  ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <SOAP-ENV:Body>
    <ns1:sayHelloToResponse xmlns:ns1="Hello" SOAP-ENV:
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <return xsi:type="xsd:string">
        Hello John, How are you doing?
      </return>
    </ns1:sayHelloToResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

XML应用概述

- 传输协议和远程调用
 - 绑定到HTTP的SOAP响应

HTTP/1.0 200 OK

Content-Type: text/xml; charset="utf-8"

Content-Length: 615

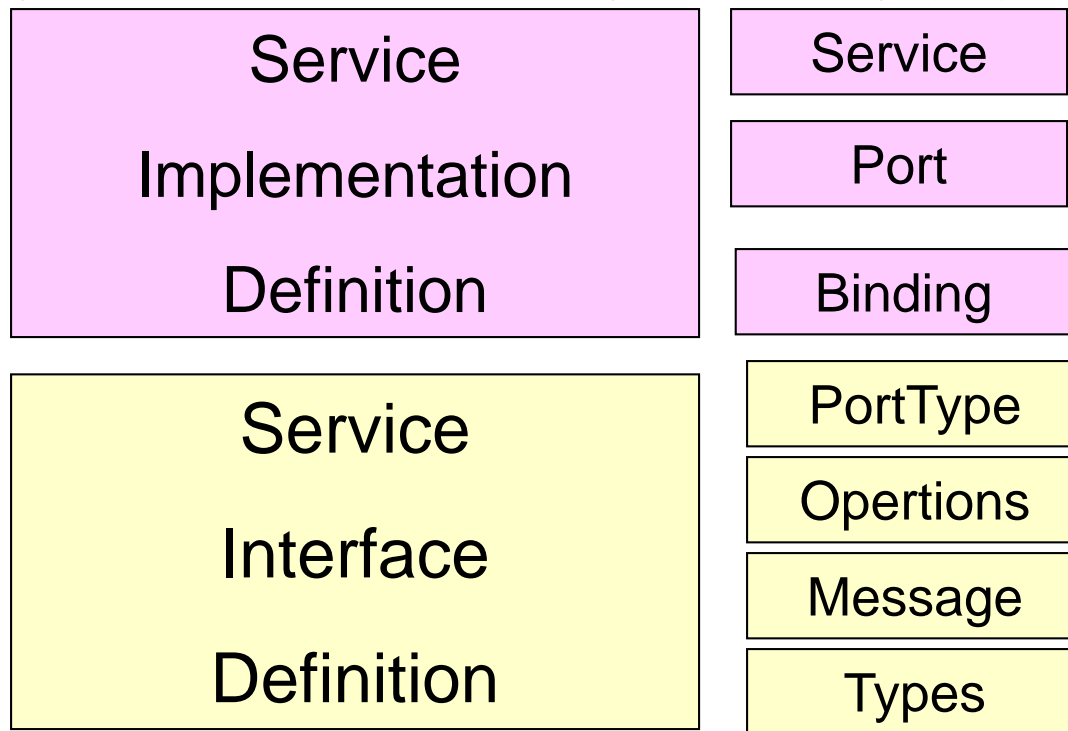
```
<SOAP-ENV:Envelope xmlns:SOAP-
  ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <SOAP-ENV:Body>
    <ns1:sayHelloToResponse xmlns:ns1="Hello" SOAP-ENV:
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <return xsi:type="xsd:string">
        Hello John, How are you doing?
      </return>
    </ns1:sayHelloToResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```


XML应用概述

■ 传输协议和远程调用

■ WSDL基本服务描述

- 基本的服务描述分成了两部分：服务接口和服务实现。



XML应用概述

- 传输协议和远程调用

- WSDL文档结构

```
<definitions>
  <import>*
  <types>
    <schema></schema>*
  </types>
  <message>*
    <part></part>*
  </message>
  <PortType>*
    <operation>*
      <input></input>
      <output></output>
      <fault></fault>*
    </operation>
  </PortType>
  <binding>*
    <operation>*
      <input></input>
      <output></output>
    </operation>
  </binding>
  <service>*
    <port></port>*
  </service>
</definitions>
```

XML应用概述

■ 传输协议和远程调用

■ WSDL文档示例

```
<?xml version="1.0">
<definitions name="urn:AddressFetcher2" ...
  <types>
    //定义服务使用的任何复杂数据类型
  </types>
  <message name="AddEntryRequest">
    //一个message对应应在调用者和服务之间传递的一条消息，要用到前面定义的数据类型
  </message>
  ...
  <portType name="AddressBook">
    //定义服务提供什么操作，要用到前面定义的消息
  </portType>
  <binding name="AddressBookSOAPBinding">
    //定义服务如何被调用
  </binding>
  <service name="AddressBookService">
    //描述服务位于哪里
  </service>
</definitions>
```

XML应用概述

■ 传输协议和远程调用

■ RSS

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<rss version="2.0">

<channel>
  <title>W3Schools Home Page</title>
  <link>http://www.w3schools.com</link>
  <description>Free web building tutorials</description>
  <item>
    <title>RSS Tutorial</title>
    <link>http://www.w3schools.com/rss</link>
    <description>New RSS tutorial on W3Schools</description>
  </item>
  <item>
    <title>XML Tutorial</title>
    <link>http://www.w3schools.com/xml</link>
    <description>New XML tutorial on W3Schools</description>
  </item>
</channel>

</rss>
```

XML应用概述

■ 传输协议和远程调用

■ RSS

Element	Description
<u><category></u>	Optional. Defines one or more categories for the feed
<u><cloud></u>	Optional. Register processes to be notified immediately of updates of the feed
<u><copyright></u>	Optional. Notifies about copyrighted material
<u><description></u>	Required. Describes the channel
<u><docs></u>	Optional. Specifies an URL to the documentation of the format used in the feed
<u><generator></u>	Optional. Specifies the program used to generate the feed
<u><image></u>	Optional. Allows an image to be displayed when aggregators present a feed
<u><language></u>	Optional. Specifies the language the feed is written in
<u><lastBuildDate></u>	Optional. Defines the last-modified date of the content of the feed
<u><link></u>	Required. Defines the hyperlink to the channel
<u><managingEditor></u>	Optional. Defines the e-mail address to the editor of the content of the feed
<u><pubDate></u>	Optional. Defines the last publication date for the content of the feed
<u><rating></u>	Optional. The PICS rating of the feed
<u><skipDays></u>	Optional. Specifies the days where aggregators should skip updating the feed
<u><skipHours></u>	Optional. Specifies the hours where aggregators should skip updating the feed
<u><textInput></u>	Optional. Specifies a text input field that should be displayed with the feed
<u><title></u>	Required. Defines the title of the channel
<u><ttl></u>	Optional. Specifies the number of minutes the feed can stay cached before refreshing it from the source
<u><webMaster></u>	Optional. Defines the e-mail address to the webmaster of the feed

XML应用概述

■ 传输协议和远程调用

■ Atom

- Google

- Atom

证明

Repr

- http:

```
<?xml version="1.0" encoding="utf-8"?>
<feed xmlns="http://www.w3.org/2005/Atom">

  <title>Example Feed</title>
  <link href="http://example.org/" />
  <updated>2003-12-13T18:30:02Z</updated>
  <author>
    <name>John Doe</name>
  </author>
  <id>urn:uuid:60a76c80-d399-11d9-b93C-0003939e0af6</id>

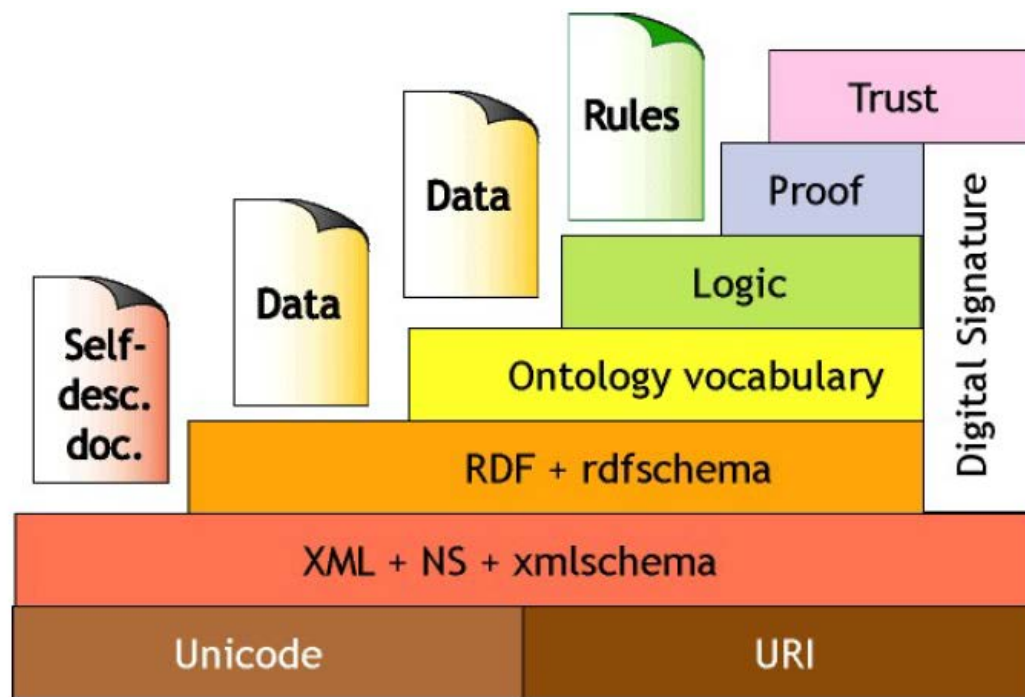
  <entry>
    <title>Atom-Powered Robots Run Amok</title>
    <link href="http://example.org/2003/12/13/atom03" />
    <id>urn:uuid:1225c695-cfb8-4ebb-aaaa-80da344efa6a</id>
    <updated>2003-12-13T18:30:02Z</updated>
    <summary>Some text.</summary>
  </entry>

</feed>
```

XML应用概述

■ 语义Web核心协议

- W3C 的“语义网远景 (Semantic Web Vision)”的目标是
 - Web 信息拥有确切的含义
 - Web 信息可被计算机理解并处理
 - 计算机可从 Web 上整合信息



XML应用概述

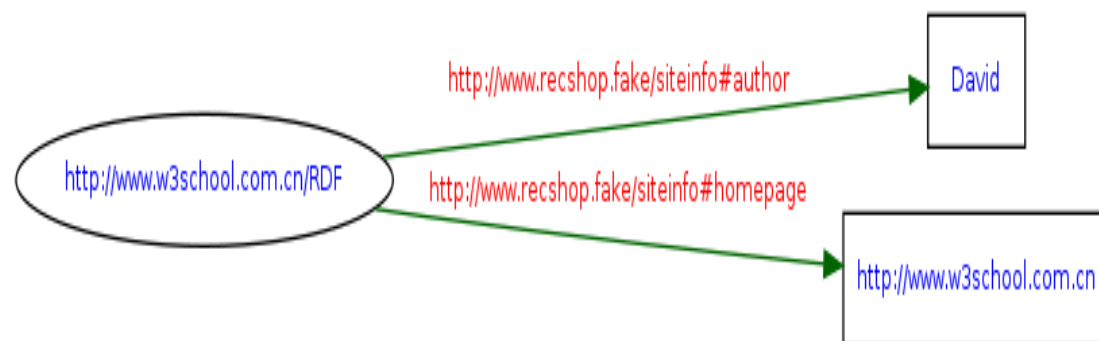
■ 语义Web核心协议

– RDF

- RDF 使用 **URIs**来标识事物，并通过属性和属性值来描述资源
 - 资源是可拥有 URI 的任何事物，比如 "http://www.w3school.com.cn/rdf"
 - 属性是拥有名称的资源，比如 "author"
 - 属性值是某个属性的值，比如 "David"
- 资源、属性和属性值的组合可形成一个**陈述**（被称为陈述的主体、谓语句和客体）

XML应用概述

- 语义Web核心协议
 - **RDF的XML表示法**



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

<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:si="http://www.recshop.fake/siteinfo#">
  <rdf:Description rdf:about="http://www.w3school.com.cn/RDF">
    <si:author>David</si:author>
    <si:homepage>http://www.w3school.com.cn</si:homepage>
  </rdf:Description>
</rdf:RDF>
```

XML应用概述

■ 语义Web核心协议

– RDF的Turtle(*Terse RDF Triple Language*)表示法

```
@prefix foaf:      <http://xmlns.com/foaf/0.1/> .
@prefix rdf:       <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix people:    <http://semwebprogramming.net/people/> .
@prefix ext:       <http://semwebprogramming.net/2008/06/ont/foaf-
extension#> .
# This is a comment.

people:Ryan ext:worksWith people:John .
people:Matt foaf:knows people:John .
people:Andrew
    foaf:knows people:Matt ;
    foaf:surname "Perez-Lopez" .
```

XML应用概述

■ 语义Web核心协议

– RDFs

- 提供定义应用程序专业的类和属性的方法
- RDF Schema 中的类与面向对象编程语言中的类非常相似

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

<rdf:RDF
  xmlns:rdf= "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xml:base=  "http://www.animals.fake/animals#">

  <rdfs:Class rdf:ID="animal" />

  <rdfs:Class rdf:ID="horse">
    <rdfs:subClassOf rdf:resource="#animal"/>
  </rdfs:Class>

</rdf:RDF>
```

XML应用概述

■ 语义Web核心协议

– Web本体语言OWL

- 对于 web，本体是关于对 web 信息及 web 信息之间的关系的精确描述
- 与 RDFs 相比，OWL 拥有更大的词汇表以及更强大的语言
- OWL标准分为OWL Lite，OWL DL，OWL Full; 概念上有个体（Individual）、属性（Property）和类（Class）

```
<owl:Class rdf:ID="WineDescriptor" />
<owl:Class rdf:ID="WineColor">
  <rdfs:subClassOf rdf:resource="#WineDescriptor" />
  ...
</owl:Class>
<owl:ObjectProperty rdf:ID="hasWineDescriptor">
  <rdfs:domain rdf:resource="#Wine" />
  <rdfs:range rdf:resource="#WineDescriptor" />
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasColor">
  <rdfs:subPropertyOf rdf:resource="#hasWineDescriptor" />
  <rdfs:range rdf:resource="#WineColor" />
  ...
</owl:ObjectProperty>
```

XML应用概述

■ 语义Web核心协议

– 基于XML的本体语言堆栈总结

名称	描述
XML	结构化文档的表层语法，对文档没有任何语义约束。
XML Schema	定义XML文档的结构约束的语言。
RDF	对象（或者资源）以及它们之间关系的数据模型，为数据模型提供了简单的语义，这些数据模型能够用XML语法进行表达。
RDF Schema	描述RDF资源的属性和类型的词汇表，提供了对这些属性和类型的普遍层次的语义。
OWL	添加了更多的用于描述属性和类型的词汇，例如类型之间的不相交性（disjointness），基数（cardinality），等价性，属性的更丰富的类型，属性特征（例如对称性，symmetry），以及枚举类型（enumerated classes）。

XML应用概述

■ 数据存储：文件格式



– Office Open XML

- Microsoft Office 系统采用的使用 XML 参考架构和 ZIP 容器，以部件为基础的文件格式规范
- 改善了文件和数据管理、数据恢复以及与行业系统的互操作性；
- 可靠性。Office XML 格式设计为比二进制格式更为可靠
- 高效性。Office XML 格式使用 ZIP 和压缩技术存储文档。
- 安全性：不能包含可执行宏代码

– Win7采用Xml格式保存日志文件

- 阅读和分析方便；兼容性好；可以进行各种日志统一分析，利于Web发布

– QQ，MSN聊天记录...

– ...

XML应用概述

- **数据存储：数据文件格式**
- **各种配置文件（DD）**
 - 通过配置文件说明和配置应用环境中的构成组件
 - J2EE环境下几乎所有的配置文件
 - Tomcat服务器通过server.xml配置其组成组件
 - Web应用发布描述符文件（web.xml）配置web层的组件
 - Hibernate的配置文件
 - Maven项目配置
 - ...

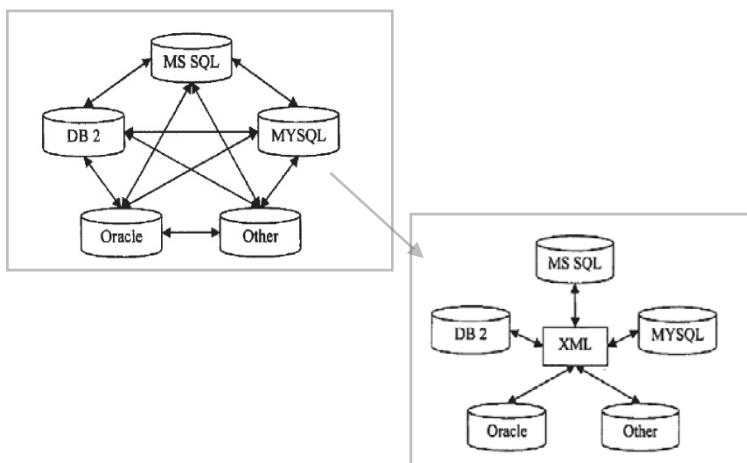
通过DD告诉容器如何与其中的组件交互以及进行管理

- 很好的可移植性和兼容性
- 模块化，不需要更改代码

XML应用概述

■ 数据存储：基于XML的数据库

- 支持XML的(XML-enabled)数据库，如
- 一般针对面向数据的XML
- XML文件的schema到数据库schema之间的映射
 - 基于表格的映射
 - 对象-关系映射



```

<DEPARTMENT deptid="15" deptname="Sales">
  <EMPLOYEE>
    <EMPNO>10</EMPNO>
    <FIRSTNAME>CHRISTINE</FIRSTNAME>
    <LASTNAME>SMITH</LASTNAME>
    <PHONE>408-463-4963</PHONE>
    <SALARY>52750.00</SALARY>
  </EMPLOYEE>
  <EMPLOYEE>
    <EMPNO>27</EMPNO>
    <FIRSTNAME>MICHAEL</FIRSTNAME>
    <LASTNAME>THOMPSON</LASTNAME>
    <PHONE>406-463-1234</PHONE>
    <SALARY>41250.00</SALARY>
  </EMPLOYEE>
</DEPARTMENT>
    
```

Department

DEPTID	DEPTNAME
15	Sales

Employee

DEPTID	EMPNO	FIRSTNAME	LASTNAME	PHONE	SALARY
15	27	MICHAEL	THOMPSON	406-463-1234	41250
15	10	CHRISTINE	SMITH	408-463-4963	52750

XML应用概述

■ 数据存储：基于XML的数据库

- 原生XML数据库(native XML database)
- 一般针对面向文档的XML
- Xindice
 - Apache的用来存储和查询XML数据的数据库服务器
 - 使用的查询语言是XPath，使用的更新语言是XML:DB Xupdate
 - 查询符合条件的数据条目示例

```
xindice xpath -c xmldb:xindice://localhost:8080/db/booksdb  
-q "//books/book[date='2006-4-8']"
```



XML应用概述

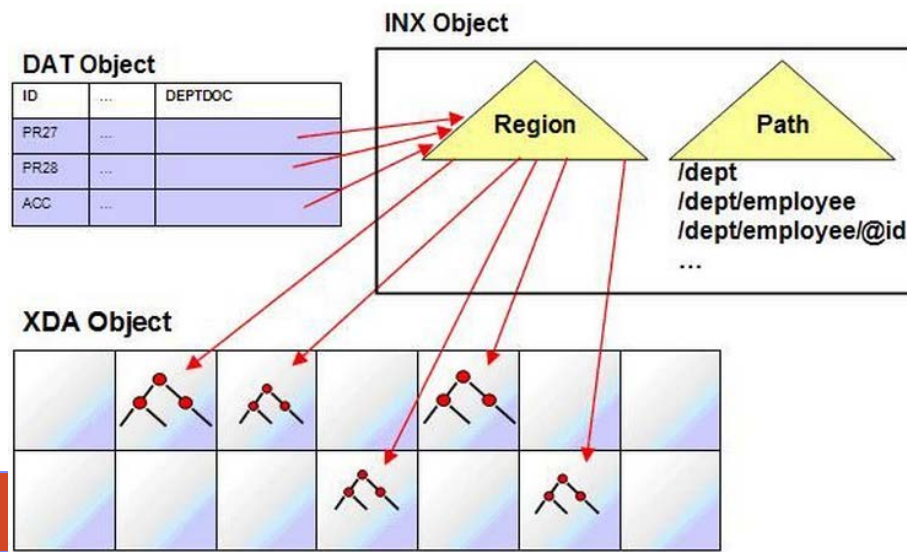
■ 数据存储：基于XML的数据库

– 之前管理 XML 数据的方法

- 将XML保存为大对象CLOB/BLOB 或者 Varchar
- 将XML经过拆分的方式，映射到关系型的数据库

– 混合型数据库：DB2 PureXML

- XML数据模型被真正地作为层次模型来支持
- 支持标准的Xquery（XPath）查询XML层次模型

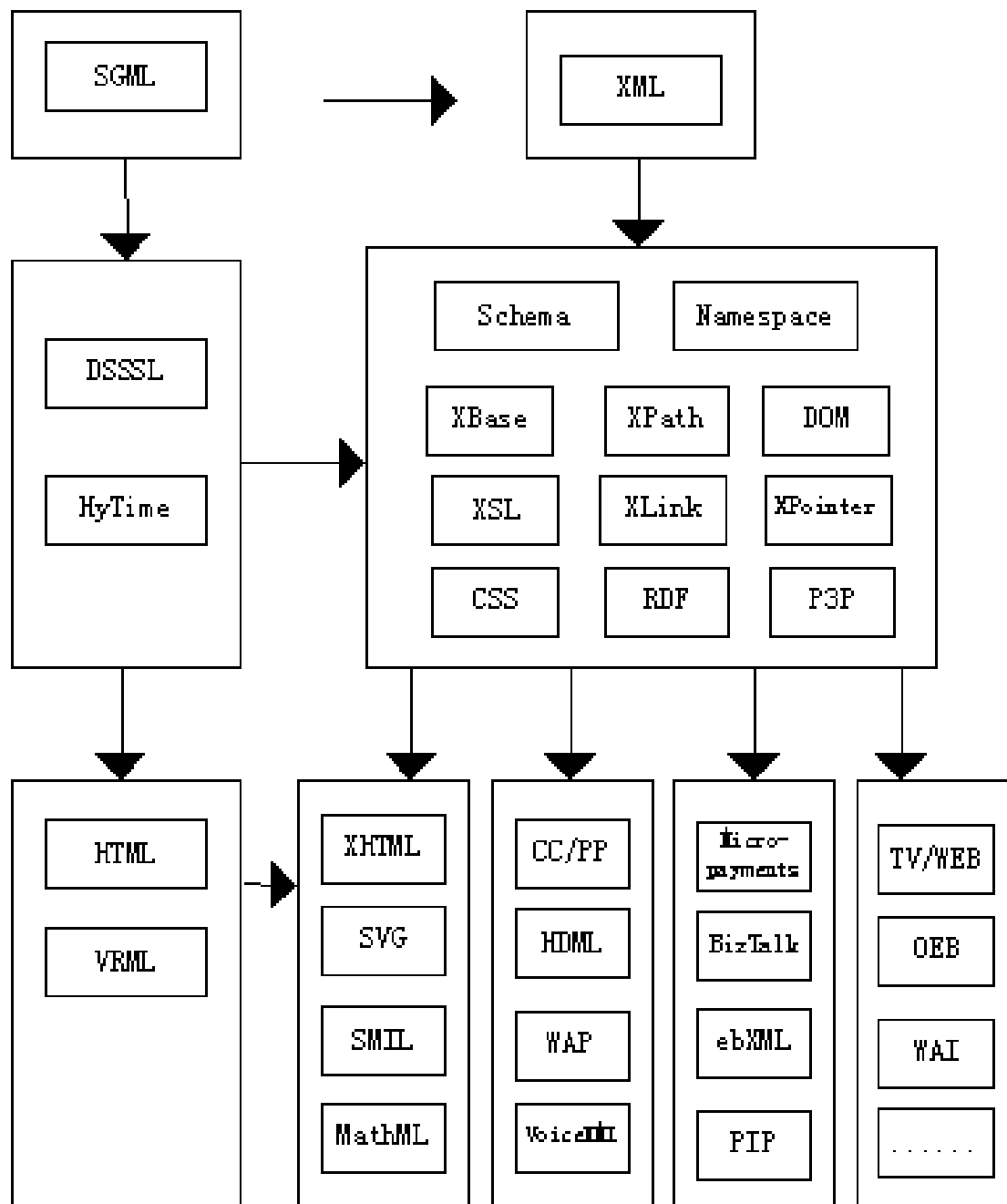


XML典型应用语言

- 建立于XML之上的应用标准

- 扩展文档标准

- SMIL（同步多媒体集成语言）
 - MathML(数学标记语言)
 - SVG(可伸缩矢量图)
 - CDF(频道定义语言,Channel definition Format)
 - WML(无线标记语言,Wireless Markup Language)
 - CML(化学标记语言)
 - IML(仪器标记语言)
 - ...



元语言标准

基础标准

应用标准



SVG

■ SVG简介

- SVG代表可伸缩矢量图(Scalable Vector Graphic),是基于XML用描述二维矢量图形和矢量/栅格混合图形的语言,支持智能搜索,并可以用JavaScript语言等自动生成和处理。HTML5图形功能的重要组成部分。
- 矢量图形和栅格图形
 - 栅格图形(.PNG, .JPEG)文件包含每个像素的颜色值,二进制存放。如HTML5 Canvas 2d API
 - 向量图形 通过制定为确定每个像素的值所需的指令而不是这些值本身,由浏览器做其余的事情。
 - Flash 二进制
 - VRML 3D

SVG

■ SVG简介

- SVG规范定义了SVG的特征、语法和显示效果，包括模块化的XML命名空间（namespace）和SVG文档对象模型(DOM)。
- 提供了大量针对图形、图象、动画的特定标记。大大丰富了网页显示效果，同时减小了文件长度，缩短了传输时间
- SVG支持脚本语言（script），数据驱动、可交互、动态图形
- 将对图形效果的编辑和显示任务由服务器端移到客户端，可充分利用客户端的资源，减轻了服务器的负担
- HTML5内联
- 支持Xlink和Xpointer:SVG文档之间的超链接

SVG

■ 基本元素支持

一个简单的svg矩形示例

```
<?xml version="1.0" standalone="no" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="300" height="100" xmlns="http://www.w3.org/2000/svg">
  <rect x="25" y="10" width="280" height="50"
    fill="red" stroke="blue" stroke-width="3"/>
</svg>
```



SVG

■ SVG变换:

– transform属性

```
<svg width="200" height="200">
  <g transform="translate(60,0) rotate(30) scale(0.75)" id="ShapeGroup">
    <rect x="10" y="20" width="100" height="80" stroke="red" fill="#ccc" />
    <circle cx="120" cy="80" r="40" stroke="#00f" fill="none" stroke-width="8" />
  </g>
</svg>
```

■ SVG复用内容

– <defs>和<use>

```
<svg width="200" height="200">
  <defs>
    <g id="ShapeGroup">
      <rect x="10" y="20" width="100" height="80" stroke="red" fill="#ccc" />
      <circle cx="120" cy="80" r="40" stroke="#00f" fill="none" stroke-width="8" />
    </g>
  </defs>

  <use xlink:href="#ShapeGroup" transform="translate(60,0) scale(0.5)"/>
  <use xlink:href="#ShapeGroup" transform="translate(120,80) scale(0.4)"/>
  <use xlink:href="#ShapeGroup" transform="translate(20,60) scale(0.25)"/>
</svg>
```

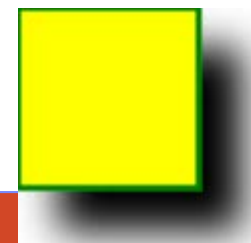

SVG

■ 过滤器和渐变

- 使用过滤器向 SVG 图形应用特殊的效果:feBlend,feColorMatrix,feComponentTransfer,feComposite,feConvolveMatrix,feDiffuseLighting...

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1">
  <defs>
    <filter id="f1" x="0" y="0"
      width="200%" height="200%">
      <feOffset result="offOut" in="SourceAlpha"
        dx="20" dy="20"/>
      <feGaussianBlur result="blurOut"
        in="offOut" stdDeviation="10"/>
      <feBlend in="SourceGraphic"
        in2="blurOut" mode="normal"/>
    </filter>
  </defs>
  <rect width="90" height="90" stroke="green"
    stroke-width="3" fill="yellow" filter="url(#f1)"/>
</svg>
```

[rect-filter.svg](#)





SVG

■ 支持脚本操作

evt.getClientX()
evt.getClientY()
evt.getScreenX()
evt.getScreenY()
evt.getCharCode()

```
<defs>
  <script type="text/ecmascript">
    <![CDATA[
      function hideReveal(evt) {
        var imageTarget = evt.target;
        var theFill = imageTarget.getAttribute("fill");
        if (theFill == 'white')
          imageTarget.setAttribute("fill", "red");
        else
          imageTarget.setAttribute("fill", "white");
      }
    ]]>
  </script>
</defs>

<!-- Outline the drawing area with a blue line -->
<rect x="1" y="1" width="350" height="200" fill="none" stroke="blue"/>
<ellipse onclick="hideReveal(evt)" cx="175" cy="100" rx="125" ry="60"
  fill="red" stroke="black" stroke-width="5"/>
```

[onclick_svg.svg](#)



SVG

- 动画制作
 - 相关元素

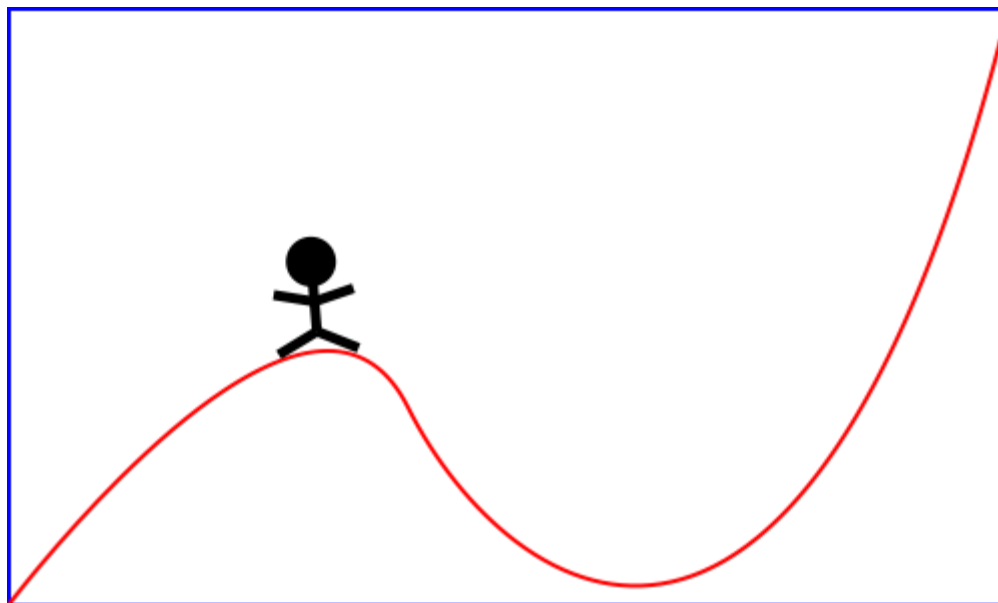
`<animate>`

`<set>`

`<animateMotion>`

`<animateColor>`

`<animateTransform>`



SVG

■ SVG工具

- Adobe Illustrator CS2
- svg-edit :
<http://svg-edit.googlecode.com/svn/trunk/editor/svg-editor.html>
- Inkscape
<http://www.inkscape.org/>

