服务计算
Service Computing

王旭
http://xuwang.tech

二、基于网络的软件服务技术

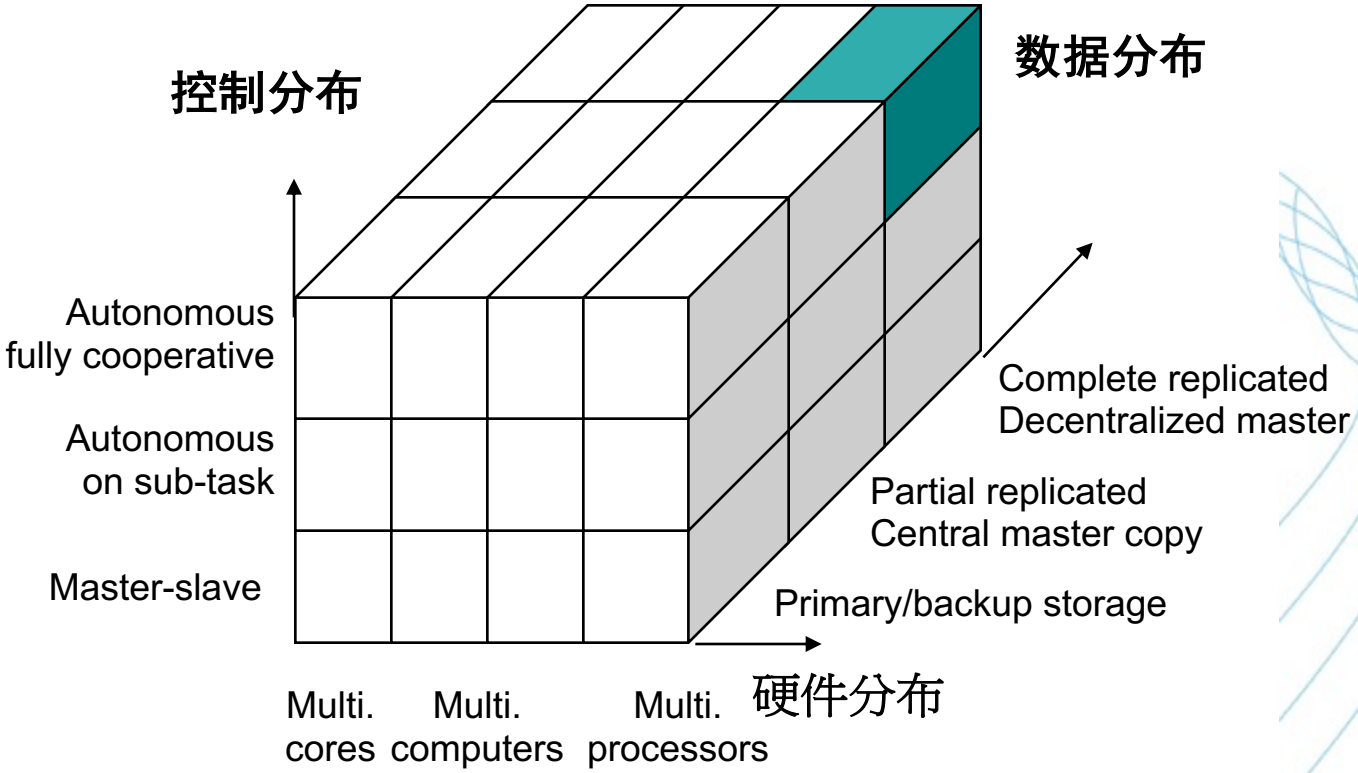
分布式计算系统

- 1945-1985
  - 计算机大、笨重
  - 之间互不相连
  - 所有系统都是单机系统
- 1980年代中期
  - 微型计算机兴起
  - 计算机网络(LAN，WLAN)
- 至今
  - 分布式系统大量出现

为什么需要分布式系统?

- 用户本身是分布的
  - 互联网及万维网、移动互联网
- 数据本身是分布的
  - 银行系统、跨国企业的信息系统
- 计算本身是分布的
  - 普适计算（传感器、可穿戴设备等）
- 单个计算机的计算能力不够
  - 集群并行计算、网格计算
- 单个计算机的存储容量不够
  - 大数据存储和计算系统

分布式系统的三个维度



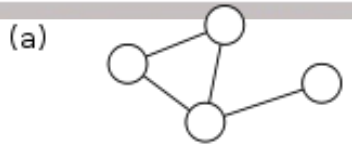
Enslow´s model

分布式系统定义

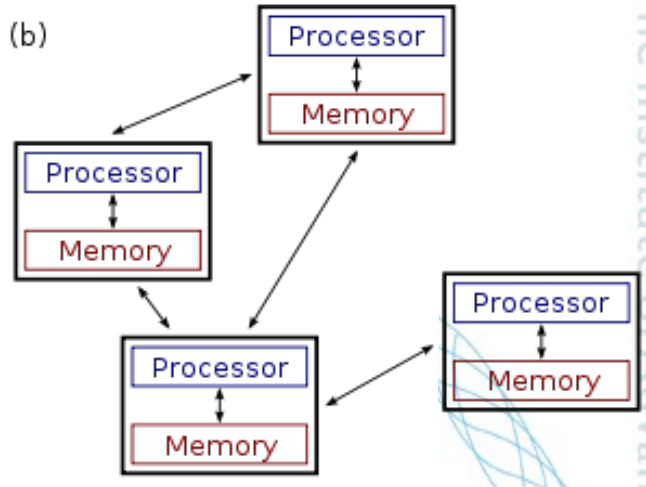
- A *distributed system* is a model in which components located on networked computers communicate and coordinate their actions by passing messages
- A computer program that runs in a distributed system is called a distributed program
- 三大特征
  - 无全局时钟：**Lack of a global 'clock'**
  - 节点并行：**Concurrency of components**
  - 失效独立：**Independent failures of components**

Colouris, Dollimore and Kindberg (2005)

单机系统



- 唯一的自主计算单元
- 单点数据
- 单点控制



- 例子：不联网的单机系统

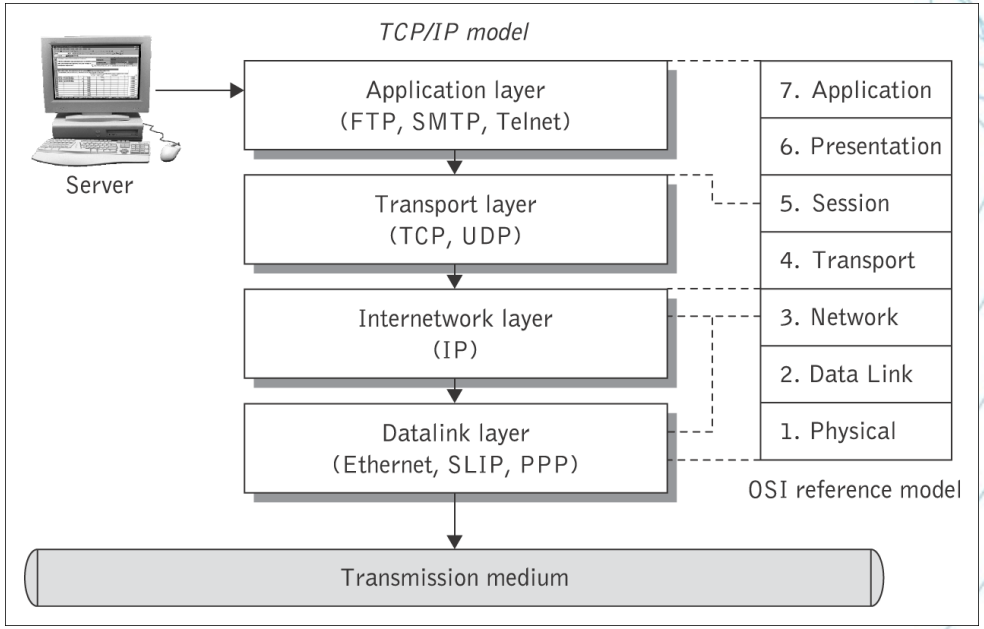
- 区别：单机多核系统、单机多CPU系统

消息传递与共享内存

- 消息传递
  - Send(Msg)
  - Receive(Msg)
- 分布式共享内存
  - Write(key,value)
  - Read(key)
- 分布式共享内存是一种抽象逻辑概念
  - 可简化分布式编程
- 两者可相互转化，是等价的

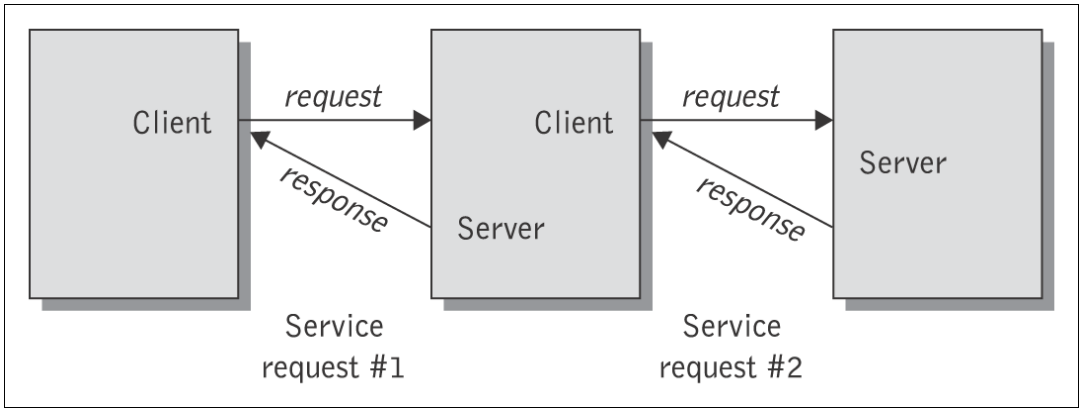
网络通信协议

- 分布式系统由一组异构网络计算机组成，它们通过消息传递的进行通信和动作协调
  - 分布式对于用户透明，整个系统体现为单一的综合设施
  - 进程间通信机制尤为重要
- 互联网协议
  - TCP/IP
  - ISO参考模型



通信模型：CS和BS

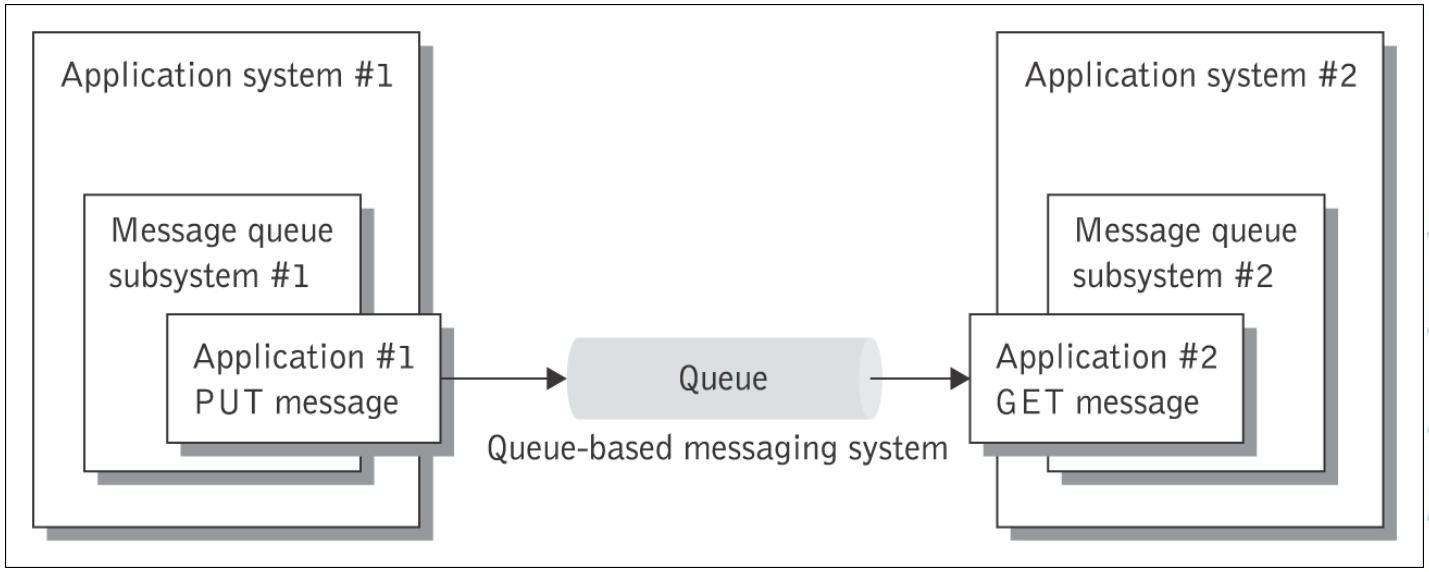
- CS架构将任务处理和存储分别放到客户端和服务器端完成
- 包括客户端进程和服务器端进程，提供服务
  - 客户端运行软件和应用，向服务器端发送请求，用户接口
  - 服务器端提供应用的数据存储



- BS: 浏览器服务器模型

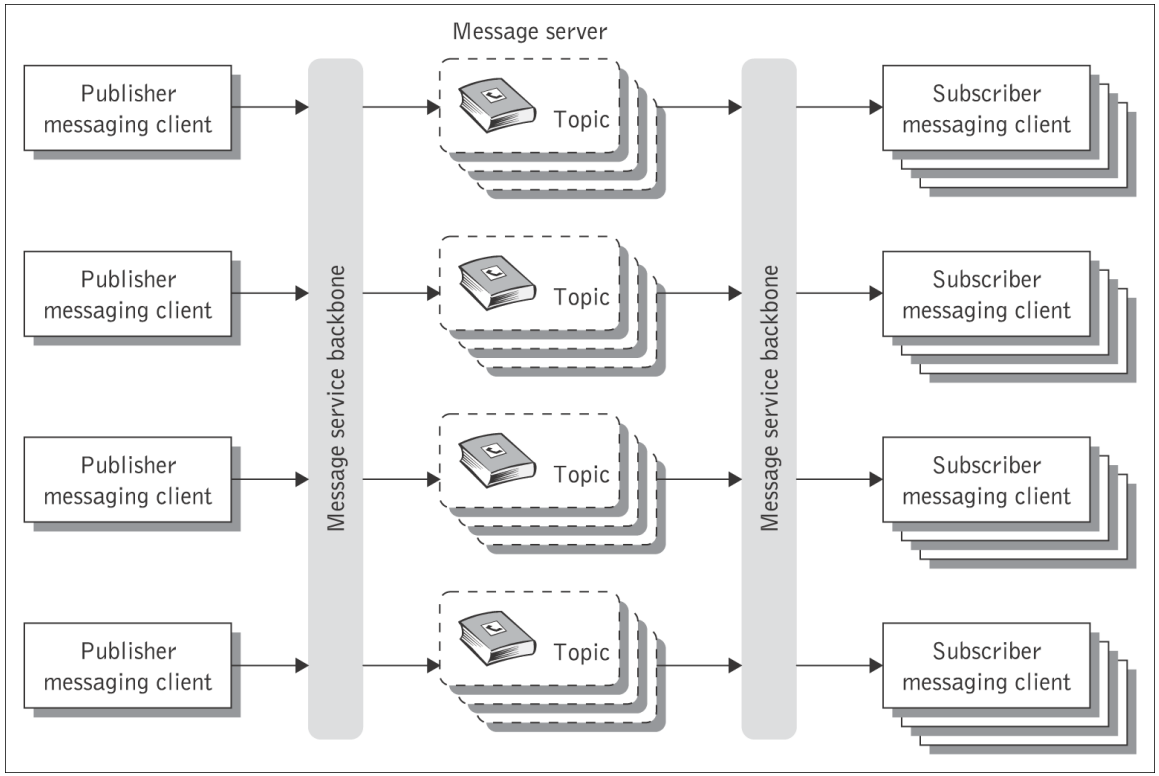
通信模型：消息队列

- 基于存储和转发队列机制，发送程序将消息发送到一个虚拟的通道（消息队列）中，接收程序从通道中获取消息
  - 队列是一个容器，可以缓存消息，直到接收者收集该消息



通信模型：消息发布/订阅

- 生产信息的应用发布信息、所有其他应用订阅特定类型的信息
  - 包含信息的消息被放置到一个被订阅者使用的队列中
  - 每个应用可为双重角色：不同信息类型的发布者和订阅者



基本消息通信原语

- 同步/异步
  - 同步/异步Send，同步Receive
- 阻塞/非阻塞
  - 阻塞：wait直到需要的值拿到
  - 非阻塞：无论需要的值是否准备好，直接返回

	Blocking	Non-blocking
Synchronous	Read/write	Read/wirte (O_NONBLOCK)
Asynchronous	i/O multiplexing (select/poll)	AIO

高级通信原语：软件服务技术

- 远程过程调用(RPC)
  - 基于本地代理，屏蔽参数和返回值传递，直接调用远程节点中的某个函数
- RESTful服务
  - 基于HTTP协议，有GET/SET/POST/DELETE等操作
- Web服务
  - 基于SOAP协议的远程接口调用，包含SOAP服务、WSDL、XML协议等

基础知识

- 网络协议
  - TCP/IP： Socket
  - HTTP
- 数据表示
  - 网络传输中：字节流 Byte[]
  - 二进制表示
  - JSON表示： person { name: wangshan， age:22}
  - XML表述：自描述的树形文本

HTTP协议

什么是WWW

- Word wide web “万维网”是在瑞士由欧洲粒子实验室（CERN）的物理学家Tim Berners-Lee和Robert Calliau于1989年首先提出的，他们的最初动机是想让几千名经常访问CERN的科学家，在世界上任何地方的计算机上都可以用同一种方式共享信息资源。
- 为了利用INTERNET实现它，Berners-Lee在1984年提出了WWW所依存的超文本（Hyper-text）数据结构，采用超文本和多媒体技术，将不同的文件通过关键字进行链接，WWW采用客户/服务器体系结构，客户和服务器间使用HTTP(Hyper Text Transfer Protocol)协议进行通信。

WWW服务系统

- 采用C/S工作模式
  - 核心协议
    - 超文本标记语言（hyper text markup language，HTML）
    - 超文本传输协议（hyper text transfer protocol，HTTP）
  - 为用户提供界面一致的信息浏览系统
- WWW服务器
  - 保存超文本文档
  - 接收和处理遵循http协议的浏览器请求，并按照http协议返回应答
- WWW浏览器
  - 接收用户的请求（用户键盘输入/鼠标输入）
  - 利用HTTP协议将用户的请求传送给WWW服务器
  - 接收服务器发来的页面文档
  - 解释页面文档，显示在用户屏幕上

HTTP server和client通信过程

浏览器确定web页面的URL

浏览器请求域名服务器解析URL地址中的域名

浏览器通过下层的TCP软件向URL所对应的IP地址的指定端口请求一个TCP连接

服务器对连接请求进行确认，建立连接的过程完成

浏览器发出请求页面报文（get /xxx.htm）

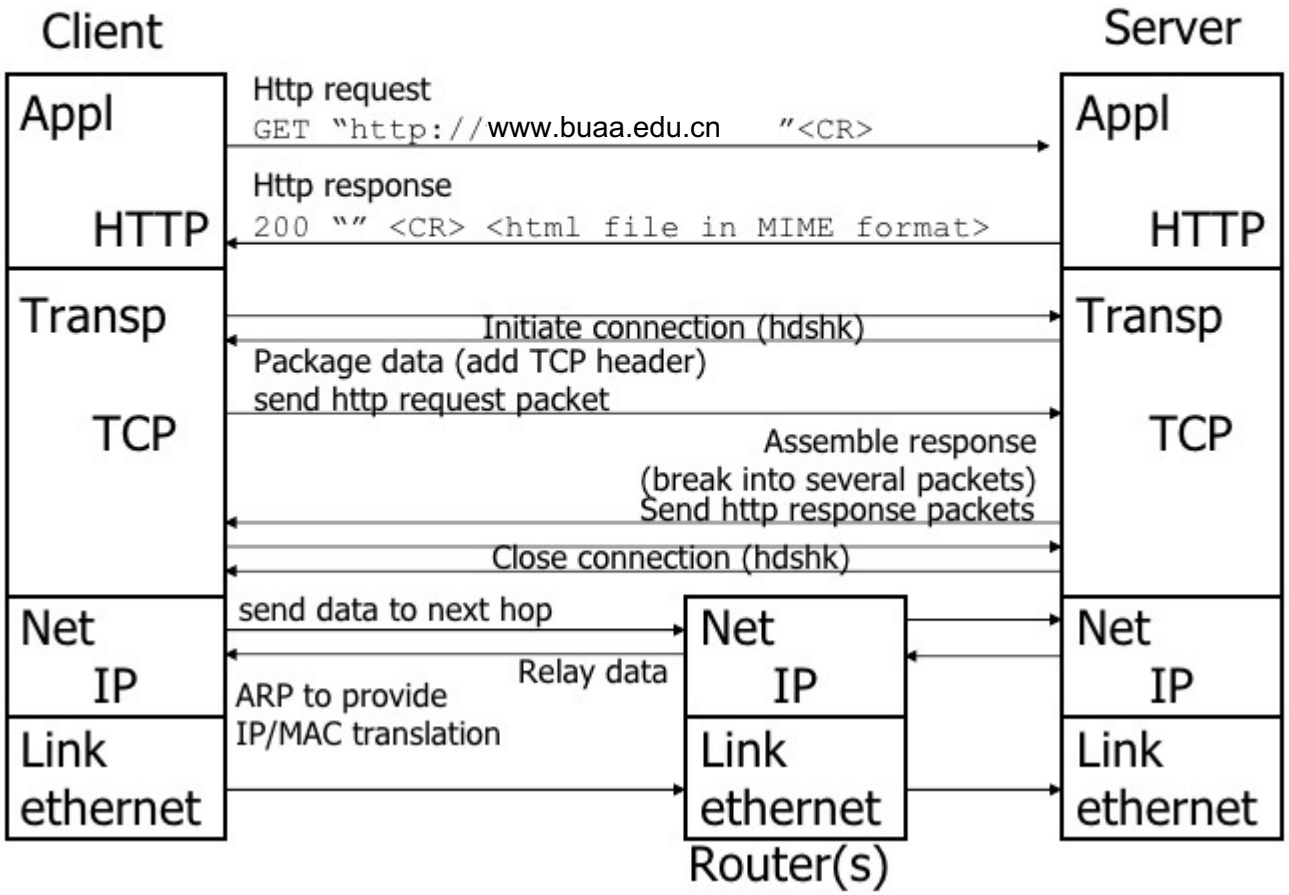
服务器响应请求，将xxx.htm文档发送给浏览器

浏览器将页面信息显示在屏幕上

WWW服务器关闭TCP连接

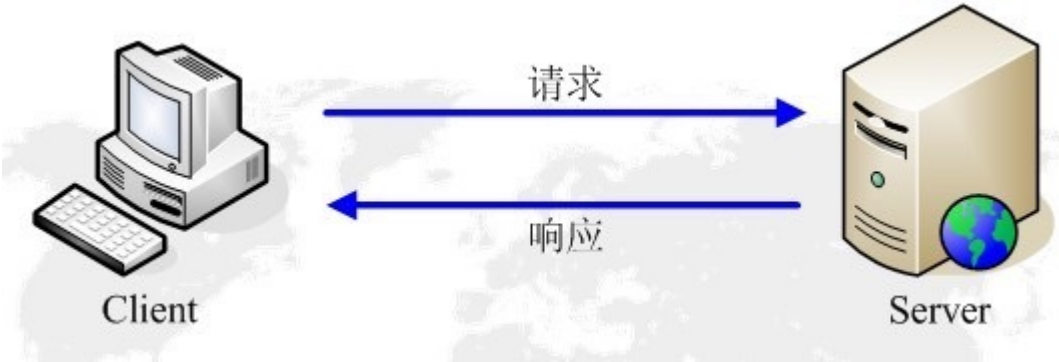
如果页面包含非文本信息，则浏览器为每个图像建立一个新的TCP连接，从服务器获得信息。

19


Client
Server
Appl
HTTP
Http request
GET "http://www.buaa.edu.cn "<CR>
Http response
200 "" <CR> <html file in MIME format>
Appl
HTTP
Transp
TCP
Initiate connection (hdshk)
Package data (add TCP header)
send http request packet
Assemble response
(break into several packets)
Send http response packets
Close connection (hdshk)
Transp
TCP
Net
IP
send data to next hop
Relay data
ARP to provide
IP/MAC translation
Net
IP
Net
IP
Link
ethernet
Link
ethernet
Link
ethernet
Router(s)

HTTP

- HTTP是Hyper Text Transfer Protocol（超文本传输协议）的缩写。RFC 1945定义了HTTP/1.0版本。其中最著名的就是RFC 2616。RFC 2616定义了今天普遍使用的一个版本——HTTP 1.1
- HTTP协议是用于从WWW服务器传输超文本到本地浏览器的传送协议。它可以使浏览器更加高效，使网络传输减少。它不仅保证计算机正确快速地传输超文本文档，还确定传输文档中的哪一部分，以及哪部分内容首先显示(如文本先于图形)等。
- HTTP是一个应用层协议，由请求和响应构成，是一个标准的客户端服务器模型。HTTP是一个无状态的协议。



HTTP

- HTTP协议的版本：HTTP/1.0、HTTP/1.1
  - 在HTTP1.0协议中，客户端与web服务器建立连接后，只能获得一个web资源。
  - HTTP1.1协议，允许客户端与web服务器建立连接后，在一个连接上获取多个web资源。

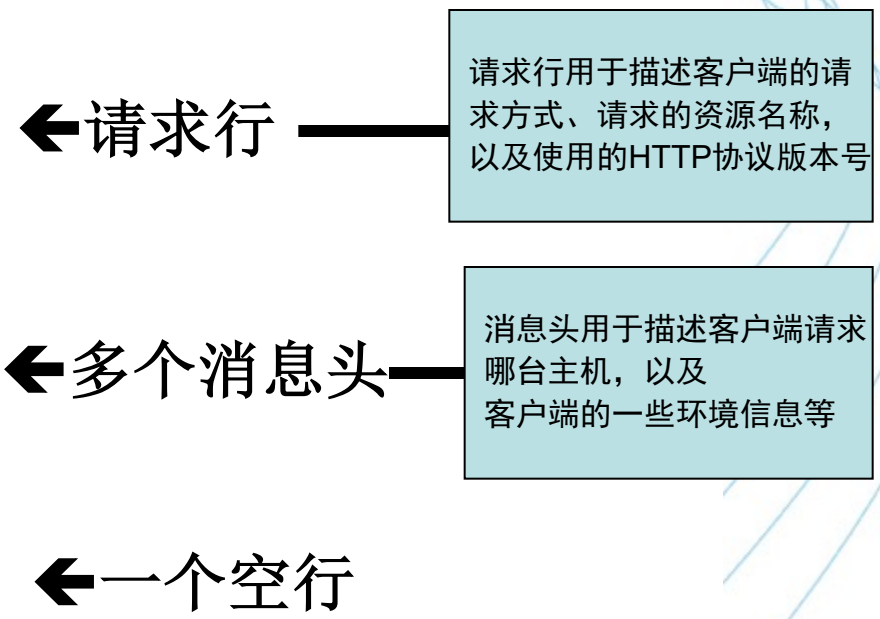
HTTP请求

- 客户端连上服务器后，向服务器请求某个web资源，称之为客户端向服务器发送了一个HTTP请求。一个完整的HTTP请求包括如下内容：

  一个请求行、若干消息头、以及实体内容，其中的一些消息头和实体内容都是可选的，消息头和实体内容之间要用空行隔开。如下所示：

- **举例：**

```
GET /books/java.html HTTP/1.1
Accept: */*
Accept-Language: en-us
Connection: Keep-Alive
Host: localhost
Referer: http://localhost/links.asp
User-Agent: Mozilla/4.0
Accept-Encoding: gzip, deflate
```



HTTP请求的细节——请求行

- 请求行中的GET称之为请求方式，请求方式有：

  POST、*GET*、*HEAD*、*OPTIONS*、*DELETE*、*TRACE*、*PUT*

  常用的有：POST、GET

- 不管POST或GET，都用于向服务器请求某个WEB资源，这两种方式的区别主要表现在数据传递上，客户端通过这两种方式都可以带一些数据给服务器：
  - 如请求方式为GET方式，则可以在请求的URL地址后以?的形式带上交给服务器的数据，多个数据之间以&进行分隔，例如：

    GET /mail/1.html?name=abc&password=xyz HTTP/1.1

    GET方式特点：URL地址后附带的参数是有限制的，其数据容量不能超过某些值。
- 如请求方式为POST方式，则可以在请求的实体内容中向服务器发送数据，例如：

  POST /servlet/ParamsServlet HTTP/1.1

  Host:

  Content-Type: application/x-www-form-urlencoded

  Content-Length: 28

  name=abc&password=xyz

  Post方式的特点：*传送的数据量无限制。*

HTTP请求的细节——消息头

- 用于HTTP请求中的常用头
  - Accept: text/html,image/*
  - Accept-Charset: ISO-8859-1
  - Accept-Encoding: gzip,compress
  - Accept-Language: en-us,zh-cn
  - Host: www.it315.org:80
  - If-Modified-Since: Tue, 11 Jul 2000 18:23:51 GMT
  - Referer: http://www.it315.org/index.jsp
  - User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0)
  - Cookie
  - Connection: close/Keep-Alive
  - Date: Tue, 11 Jul 2000 18:23:51 GMT
- Linux中的Curl命令

HTTP响应

- 一个HTTP响应代表服务器向客户端回送的数据，它包括：

  一个状态行、若干消息头、以及实体内容 ，其中的一些消息头和实体内容都是可选的，消息头和实体内容之间要用空行隔开。

- 举例：

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Date: Thu, 13 Jul 2000 05:46:53 GMT
Content-Length: 2291
Content-Type: text/html
Cache-control: private

<HTML>
<BODY>
……
```

←状态行 —— 状态行用于描述服务器对请求的处理结果。

←多个消息头 —— 消息头用于描述服务器的基本信息，以及数据的描述，服务器通过这些数据的描述信息，可以通知客户端如何处理等一会儿它回送的数据。

←一个空行

←实体内容 —— 代表服务器向客户端回送的数据

HTTP响应的细节——状态行

- **状态行**

  格式： *HTTP版本号　状态码　原因叙述<CRLF>*

  举例： *HTTP/1.1 200 OK*

- **状态码用于表示服务器对请求的处理结果，它是一个三位的十进制数。响应状态码分为5类，如下所示：**

状态码	含义
100～199	表示成功接收请求，要求客户端继续提交下一次请求才能完成整个处理过程
200～299	表示成功接收请求并已完成整个处理过程，常用200
300～399	为完成请求，客户需进一步细化请求。例如，请求的资源已经移动一个新地址，常用302、307和304
400～499	客户端的请求有错误，常用404
500～599	服务器端出现错误，常用 500

其它常用状态码

- 301： 永久重定向
  - a.com/old/url Reponse: 301, a.com/new/url
- 302: 临时重定向

```
location /old/url/ {
    return 301 /new/url/;
}
```

Nginx配置的例子

Client request:

```
GET /index.php HTTP/1.1
Host: www.example.org
```

Server response:

```
HTTP/1.1 301 Moved Permanently
Location: http://www.example.org/index.asp
```

Client request:

```
GET /index.html HTTP/1.1
Host: www.example.com
```

Server response:

```
HTTP/1.1 302 Found
Location: http://www.iana.org/domains/example/
```

- 401 （未授权） 请求要求身份验证。 对于需要登录的网页，服务器可能返回此响应。
- 具体可参照 en.wikipedia.org/wiki/List_of_HTTP_status_codes

HTTP响应细节——常用响应头

- HTTP请求中的常用响应头
  - Location: http://www.it315.org/index.jsp
  - Server:apache tomcat
  - Content-Encoding: gzip
  - Content-Length: 80
  - Content-Language: zh-cn
  - Content-Type: text/html; charset=GB2312
  - Last-Modified: Tue, 11 Jul 2000 18:23:51 GMT
  - Refresh: 1;url=http://www.it315.org
  - Content-Disposition: attachment; filename=aaa.zip
  - Transfer-Encoding: chunked
  - Set-Cookie:SS=Q0=5Lb_nQ; path=/search
  - Expires: -1
  - **Cache-Control: no-cache**
  - **Pragma: no-cache**
  - **Connection: close/Keep-Alive**
  - **Date: Tue, 11 Jul 2000 18:23:51 GMT**

JSON技术

JSON简述

- JSON(JavaScript Object Notation)
  - 一种轻量级的数据交换格式
  - 易于人阅读和编写，同时也易于机器解析和生成
  - 基于JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 1999的一个子集
  - JSON采用完全独立于语言的文本格式，兼容C, C++, C#, Java, JavaScript, Perl, Python等语言习惯
- 特点
  - 文本字符串形式
  - Key-Value二元组，可嵌套定义
  - 支持数组[“a”,”b”,”c”]
  - 一般无自描述信息
- 文法可参考www.json.org/json-zh.html

例子

```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 27,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    },
    {
      "type": "mobile",
      "number": "123 456-7890"
    }
  ],
  "children": [],
  "spouse": null
}
```

```
request.open('GET', requestURL);
request.responseType = 'text'; // now we're getting a string!
request.send();

request.onload = function() {
  var superHeroesText = request.response; // get the string from the response
  var superHeroes = JSON.parse(superHeroesText); // convert it to an object
  populateHeader(superHeroes);
  showHeroes(superHeroes);
}
```

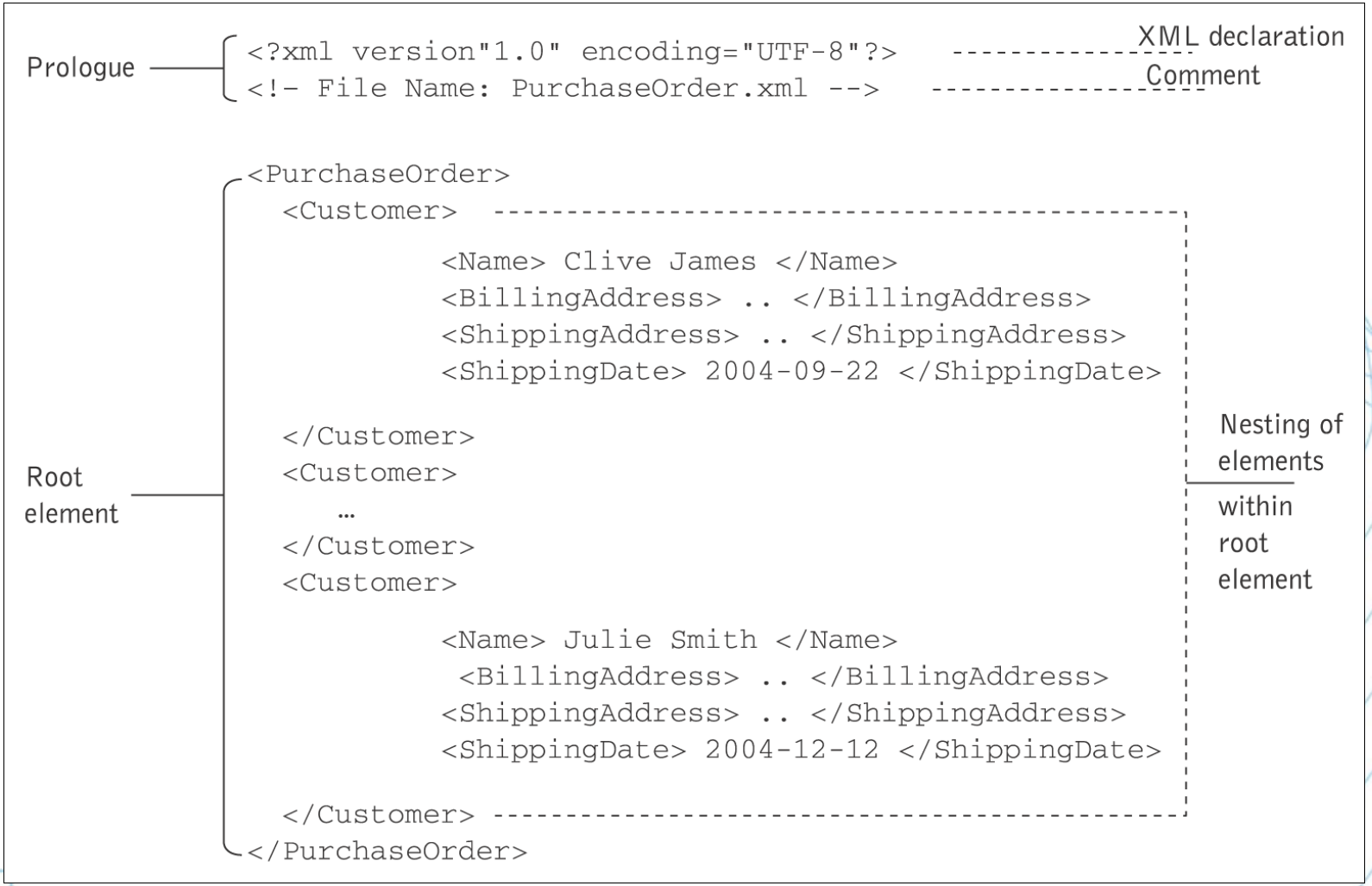
```
var myJSON = { "name": "Chris", "age": "38" };
myJSON
var myString = JSON.stringify(myJSON);
myString
```

XML技术

概述

- XML的产生
  - XML，或称为可扩展标记语言（Extensible Markup Language），是一种可以用来创建自己的标记的标记语言。它由万维网协会（W3C）创建， 1998年2月，XML1.0成为了W3C的推荐标准，用来克服 HTML（即超文本标记语言，它是所有网页的基础）的局限。
  - XML与HTML一样都是SGML-标准通用标记语言（Standard Generalized Markup Language）的子集。由于SGML十分庞大，既不容易学，又不容易使用，在计算机上实现也十分困难。所以W3C使用精简的SGML版本-XML
- XML的功能
  - XML是一套定义语义标记的规则，标记将文档分成许多部件并对这些部件加以标识。XML提供了一个直接处理Web数据的通用方法，描述的是Web页面的内容。

XML文档示例


Prologue
<?xml version"1.0" encoding="UTF-8"?>
XML declaration
<!- File Name: PurchaseOrder.xml -->
Comment
<PurchaseOrder>
<Customer>
<Name> Clive James </Name>
<BillingAddress> .. </BillingAddress>
<ShippingAddress> .. </ShippingAddress>
<ShippingDate> 2004-09-22 </ShippingDate>
</Customer>
<Customer>
...
</Customer>
<Customer>
<Name> Julie Smith </Name>
<BillingAddress> .. </BillingAddress>
<ShippingAddress> .. </ShippingAddress>
<ShippingDate> 2004-12-12 </ShippingDate>
</Customer>
</PurchaseOrder>
Root element
Nesting of elements within root element

XML和HTML的区别

XML和HTML都是用于操作数据或数据结构，在结构上大致是相同的，但它们在本质上却存在着明显的区别，它们的区别主要有以下几点：

- 语法要求不同：在HTML中不区分大小写，在XML中对大小写要求非常严格。
- 标记不同：HTML使用固有的标记，而XML没有固有标记。
- 作用不同：HTML用于显示页面，而XML用于描述页面内容的数据或数据的结构。HTML把数据和显示合在一起，在页面中把这些数据显示出来，而XML则将数据和显示分开。

一段HTML文档

```
<html>
<body>

<p>
Each table starts with a table tag.
Each table row starts with a tr tag.
Each table data starts with a td tag.
</p>

<h4>One column:</h4>
<table border="1">
<tr>
  <td>100</td>
</tr>
</table>

<h4>One row and three columns:</h4>
<table border="1">
<tr>
  <td>100</td>
  <td>200</td>
  <td>300</td>
</tr>
</table>

<h4>Two rows and three columns:</h4>
<table border="1">
<tr>
  <td>100</td>
  <td>200</td>
  <td>300</td>
</tr>
<tr>
  <td>400</td>
  <td>500</td>
  <td>600</td>
</tr>
</table>

</body>
</html>
```

Each table starts with a table tag. Each table row starts with a tr tag. Each table data starts with a td tag.

One column:

100

One row and three columns:

100	200	300

Two rows and three columns:

100	200	300
400	500	600

XML的优势

XML最大的优势在于它能对各种编程语言编写的数据进行管理，使得在任何平台下都能通过解析器来读取XML数据。它的优势可归纳为以下几点：

- 数据的搜索：在XML中可以提取文档中任何位置的数据，
- 数据的显示：XML将数据的结构和数据的显示形式分开，根据需要使数据呈现出多种显示方式。如HTML、PDF等格式。
- 数据的交换：XML标记语言的语法非常简单，可以通过解析器在任何机器上解读。并可以在各种计算机平台上使用。逐渐成为一种数据交换的语言。

XML 正改变着 Web

- 使用 XML 创建具有自我描述性数据的文档来改进 Web。以下是几个关键领域：
- XML 简化了数据交换
  - 因为不同组织（乃至同一组织的不同部门）很少就单一工具集形成标准，所以要使应用程序相互交流需要进行大量工作。使用 XML，每个组织可以创建单一的实用程序，该实用程序将该组织的内部数据格式转换成 XML，反之亦然。最好有这样的机会：这些组织的软件供应商已经提供了在它们的数据库记录（或 LDAP 目录，或采购订单等等）与 XML 之间进行相互转换的工具
- XML 支持智能代码
  - 因为可以使 XML 文档结构化以标识每个非常重要的信息片段（以及这些片段之间的关系），所以可以编写无需人工干预就能处理这些 XML 文档的代码。
- XML 支持智能搜索
  - 尽管搜索引擎这些年在稳步改进，但从搜索中得到错误的结果仍很常见。如果您正在搜索包含名叫“Chip”的人的 HTML 页面，您可能还会找到有关功克力片、计算机芯片、木片以及许多其它无用匹配的页面。搜索 XML 文档查找包含文本 Chip 的 <first-name> 元素会给出一个好得多的结果集

XML相关文档

- XML Information Set
  - http://www.w3.org/TR/xml-infoset
- XML Linking Language (XLink) Version 1.0
  - http://www.w3.org/TR/xlink/
- XML Path Language (XPath)
  - http://www.w3.org/TR/xpath
- XML Pointer Language (XPointer) Version 1.0
  - http://www.w3.org/TR/xptr
- Namespaces in XML
  - http://www.w3.org/TR/REC-xml-names
- The Extensible Stylesheet Language Family (XSL)
- Document Object Model (DOM)
- HyperText Markup Language (HTML)
- The Extensible HyperText Markup Language
  - XHTML™ 1.0
- Resource Description Framework (RDF)

XML文档结构

XML文档也属于纯文本文件，该文档一般如下四部分组成：

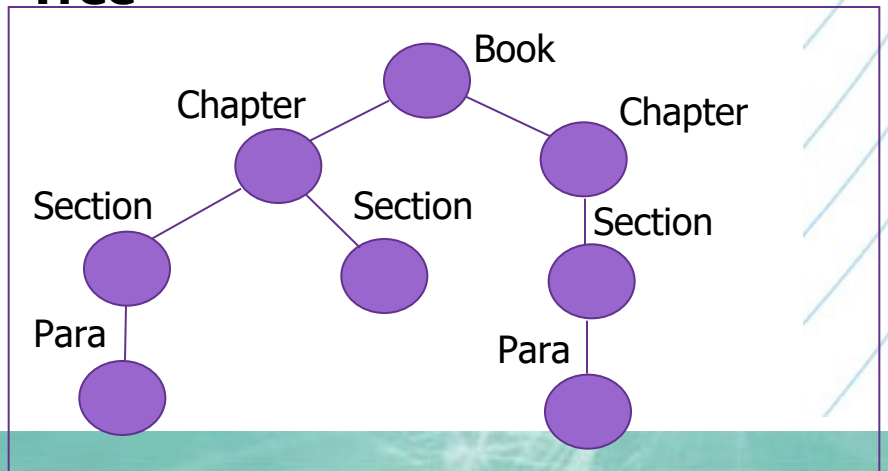
- XML文档的声明
- XML文档类型定义
- XML文档注释
- XML标识及其内容

按照这种文档格式来编写的一个XML文件，如下所示：

Linear sequence

```
<book>
 <chapter n="1">  Title 1 </chapter>
 <section n="1.1"> Section 1.1 </section>
 <paragraph> .... <paragraph>
 <section n="1.2"> Section 1.2 </section>
 <chapter n="2">  Title 2 </chapter>
 <section n="2.1"> Section 2.1 </section>
 < paragraph > .... < paragraph >
</book>
```

Tree



元素

元素是XML文档的重要组成部分，在XML文档中必须存在元素。XML文档的元素一般是由标记头、标记末和标记间的字符串数据构成，如下代码所示：

```
<root>
        <a>this is test</a>
</root>
```

元素a的值

元素a的元素名或标签名

XML文档中的第一个元素被称为根元素，在任何一个XML文档中有且只有一个元素被称为根元素。其余所有的元素都是子元素，子元素必须正确的嵌套在根元素中。

标记间的字符串数据就是该元素的值，在XML中，如果元素的值中存在空格，那么这些空格将按原样解析出来（注意空格与空元素的区别 <a> = <a/>）

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。
- `<?xml version="1.0"?>`
- `<!-- A well-formed document -->`
- `<XML文档>`
- `  Hello, World!`
- `</XML文档>`

属性

属性是用来修饰某个元素的，如：

```
<root>
        <a attribute="aa">this is test</a>
</root>
```

属性名

属性值

关于元素的属性需注意如下几个问题：

- 属性的值必须用引号括起来，如： attribute1=“aa” 或attribute3=‘aa' ；
- 元素的属性以名和值成对出现；
- 用来修饰同一个元素的属性的属性名不能相同 ；
- 属性值不能包含“&”、“’ ”、“<”等字符。

注释、处理指令与实体

- 注释<!--……-->
- 处理指令<?……?>: 为了给处理页面的程序（例如XML解析器）提供额外的信息。
- 实体<!ENTITY dw "developerWorks">: 实体是对数据的引用；根据实体种类的不同，XML 解析器将使用实体的替代文本或者外部文档的内容来替代实体引用。

预定义实体表如下所示:

实体名	引用格式	表示的符号
lt	<	<
gt	>	>
amp	&	&
apos	'	'
quot	"	"

实体在XML文档中的一般引用格式如下：

&实体名;

CDATA节

通过CDATA节可以通知分析器，在CDATA节包含的字符中没有标记。这样，如果文档包含可能会出现的标记字符，但我们又不是把它当作标记来使用，而只是属于文本字符，那么使用CDATA节来创建这样的文档就容易得多。CDATA节主要用于脚本语言内容、示例XML文档内容和HTML内容。如下所示：

```
<?xml version="1.0" encoding="gb2312"?>
    <程序>
        <title>test</title>
        <内容>
        <![CDATA[
            if(20<10){
                return "你好";
            }else{
                return "hello";
            }
        ]]>
        </内容>
    </程序>
```

注意：在“<![CDATA[”和“]]>”之间不能再加入CDATA节 或“]]>”

名称空间

- 名称空间定义了URI；
- 名称空间重要性在于其唯一性

```
<root
xmlns:h="http://www.w3.org/TR/html4/"
xmlns:f="http://www.w3schools.com/furniture">

<h:table>
  <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
  </h:tr>
</h:table>

<f:table>
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>

</root>
```

定义文档内容

- 两种方法定义数据元素：
  - 文档类型定义（Document Type Definition）或简称 DTD
  - XML Schema
- XML Schema特点：
  - 使用XML语法
  - 支持数据类型
  - 可扩展
  - 表达能力强

XML Schema规范

- XML Schema Part 0: Primer
  - 提供入门知识，解释模式是什么？如何建立模式？
  - http://www.w3.org/TR/xmlschema-0/
- XML Schema Part 1: Structures
  - 定义XML文档结构模式
  - http://www.w3.org/TR/xmlschema-1/
- XML Schema Part 2: Datatypes
  - 定义了一组简单类型
  - http://www.w3.org/TR/xmlschema-2/

XML schema和实例文档

File “Person.xml”

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

<Person
xmlns:xsi="http://www.w3.org/
   2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="
Person.xsd">

      <First>Sophie</First>

      <Last>Jones</Last>

      <Age>34</Age>

</Person>
```

File “Person.xsd”

```
<?xml version="1.0" encoding="UTF-
8"?>
<xs:schema
   xmlns:xs="http://www.w3.org/2001/
             XMLSchema">
  <xs:element name="Person">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="First"
                    type="xs:string"/>
        <xs:element name="Middle"
                    type="xs:string"
                    minOccurs="0"/>
        <xs:element name="Last"
                    type="xs:string"/>
        <xs:element name="Age"
                    type="xs:integer"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

XML Schema结构文档

- 注释
- 声明
  - 元素声明
  - 属性声明
  - 表示法声明
- 类型定义
  - 简单类型
  - 复杂类型
- 属性组
- 模型组

Schema Wrapper

```
<xs:schema
    xmlns:xs="http://www.w3.org/1999/XMLSchema"
    targetNamespace="http://www.mpeg.org/mpeg-7"
    version="1.1">
….
</xs:schema>
```

xmlns:xs - 'xs' prefix to reference elements defined in a schema from another namespace

targetNamespace - all the elements and types defined in this schema come from this namespace. Use this URI to import or include these definitions in other schemas

XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="loan">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="initialBalance" type="balance" />
        <xs:element name="currentBalance" type="balance" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:simpleType name="balance">
    <xs:restriction base="xs:decimal">
      <xs:totalDigits value="10" />
      <xs:fractionDigits value="2" />
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
```

元素声明

- <element name='QName'
  ref='QName‘
  type='QName‘
  use='optional|prohibited|required‘
  form='qualified|unqualified‘
  id='ID‘
  default='String‘
  fixed='String'>
  Content:
  (annotation?,((simpleType|complexType)?
  (unique|key|keyref)*))
- </element>

元素声明

```
<element name="myglobaleIt1" type="mySimpleType"/>
<element name="myglobaleIt2" type="myComplexType"/>
<element name="myglobaleIt3">
    <complexType>
        <element name="mylocaleIt" type="otherType"/>
        <element ref="myglobaleIt2"/>
        <attribute name="mylocalattr" type="date"/>
    </complexType>
</element>
```

属性声明

- <attribute name='QName'
  ref='QName‘
  type='QName‘
  use='optional|prohibited|required‘
  form='qualified|unqualified‘
  id='ID‘
  default='String‘
  fixed='String'>
  (any attribute with non-schema namespace)>
  Content: (annotation?,(simpleType?))
- </attribute>

简单类型

- <simpleType
  final='#all|list|union|restriction)'
  id='ID‘
  mixed='boolean‘
  name='NCName‘
  (any attribute with non-schema namespace)>>
  Content: (annotation?,(restriction|list|union))
- </simpleType>

简单类型定义

```
<simpleType name="US-State" base="string">
    <enumeration value="AK"/>
    <enumeration value="AL"/>
    <enumeration value="AR"/>
……
</simpleType>
<attribute name="State1" type="string"/>
<attribute name="State2" type="US-State"/>
```

复杂类型定义

- <complexType abstract='boolean'
  block='#all|List of
  (extension|restraction)‘
  final='#all|List of
  (extension|restraction)‘
  id='ID‘
  mixed='boolean‘
  name='NCName‘
  (any attribute with non-schema namespace)>
  Content: (annotation?,(simplecontent|complexComtent
  ((group|all|choice|sequence)?,
  ((attribute|attributeGroup)*,anyAttribute?)))
- </complexType>

复杂类型定义

```
<complexType name="personName">
    <element name="title" type="string"/>
    <element name="forename" type="string"/>
    <element name="surname" type="string"/>
    <attribute name="age" type="integer"/>
</complexType>
<element name="producer"
  type="personName"/>
```

匿名简单类型定义

```
<attribute name="myAttribute" default="42">
    <simpleType base="integer">
        <minExclusive value="0"/>
    </simpleType>
</attribute>
```

匿名复杂类型定义

```
<element name="personName">
   <complexType>
      <element name="title"/>
      <element name="forename"/>
      <element name="surname"/>
      <attribute name="age" type="integer"/>
   </complexType>
</element>
```

组元素

- **组元素将元素声明组成一个集合；**
- **组元素仅仅将元素声明组成一个集合，而不能将属性声明聚集；**
- 排序 - *all|sequence|choice*

- all规定子元素能够以任意顺序出现，每个子元素可出现零次或一次。
- choice仅允许在 <choice> 声明中包含一个元素出现在包含元素中。
- sequence要求子元素必须按顺序出现。每个子元素可出现 0 到任意次数。

组示例

```
<group name="myModelGroup">
    <sequence>
        <element name="firstThing" type="type1"/>
        <element name="secondThing" type="type2"/>
    </sequence>
</group>

<complexType name="newType"
   <choice>
      <group ref="myModelGroup"/>
      <element ref="anotherThing"/>
  </choice>
</complexType>
```

属性组

```
<attributeGroup name="myAttrGroup">
    <attribute name="myD1" type="string"/>
    <attribute name="myD2" type="integer"/>
    <attribute name="myD3" type="date"/>
</attributeGroup>
<complexType name="myDS">
    <element name="myelement" type="myType"/>
    <attributeGroup ref="myAttrGroup"/>
</complexType>
```

注：属性声明总是在最后

import 元素

- Import元素允许引用另一个名字空间的元素

```
<schema xmlns="http://www.w3.org/1999/XMLSchema"
            targetNamespace="http://www.mpeg.org/Examples"
            xmlns:html="http://www.w3.org/1999/XHTML">
   <import namespace="http://www.w3.org/1999/XHTML"
              schemaLocation="http://www.w3.org/XHTML/xhtml.xsd"/>
   ...
   <element ref="html:table">
   …
</schema>
```

XML处理库

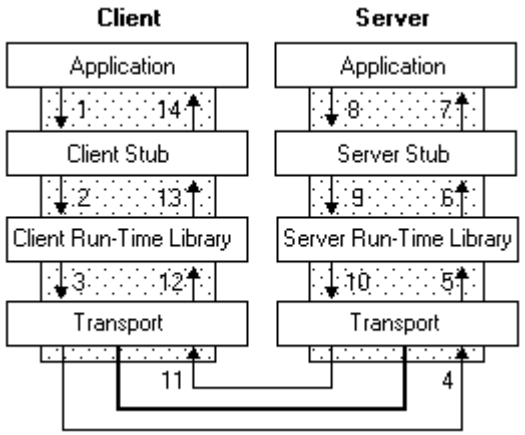
- JAXP: Java API for XML Processing
- JAXB: Java Architecture for XML Binding
- JDOM: Java DOM
- DOM4J: an alternative to JDOM
- JAXM: Java API for XML Messaging (asynchronous)
- JAX-RPC: Java API for XML-based Remote Process Communications (synchronous)
- JAXR: Java API for XML Registries

RPC(Remote Procedure Call)
远程过程调用

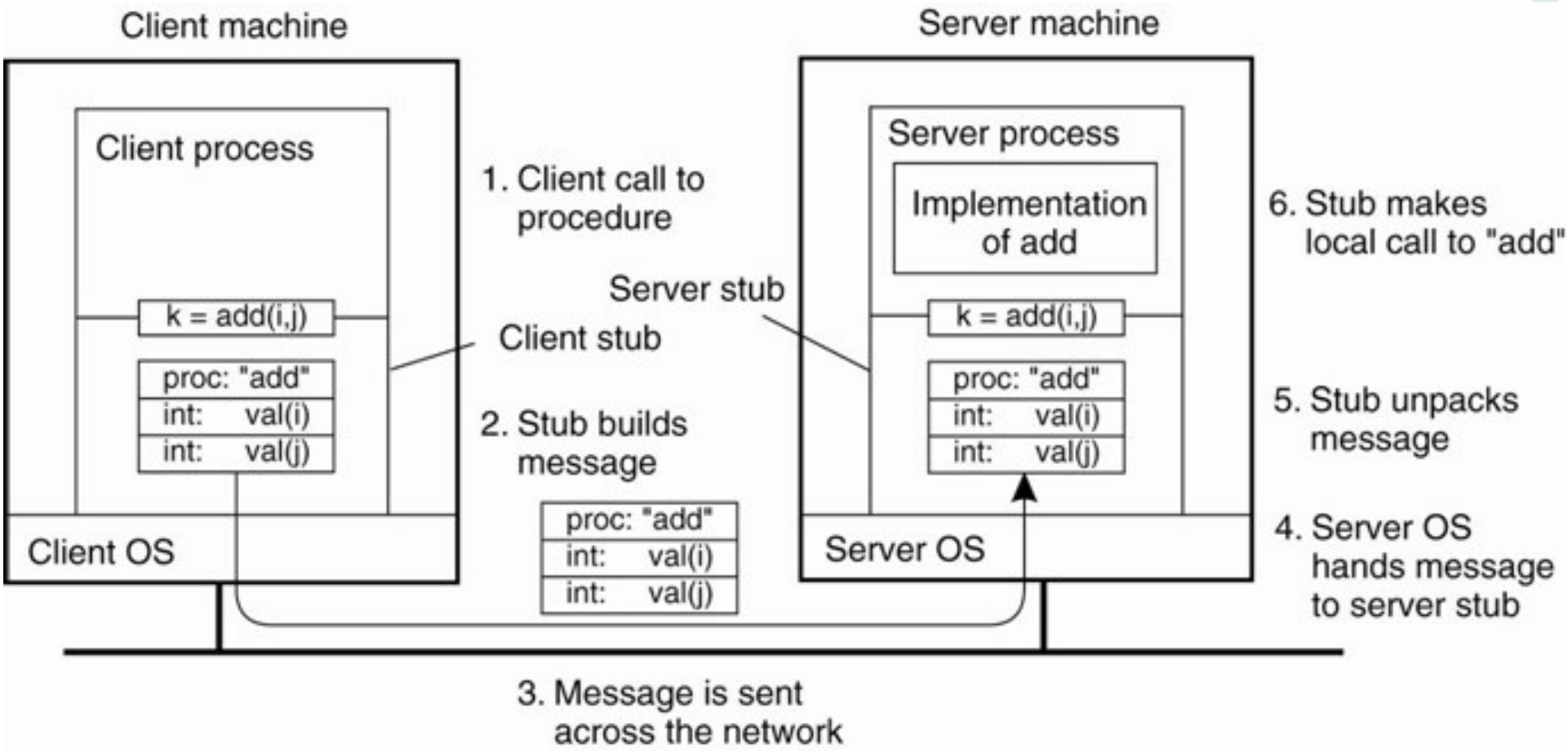
过程调用

- 本地过程调用
  - 函数调用
  - 类方法的调用
  - 原理是：同一进程内部，由操作系统在内存中进行参数和返回值的拷贝，不经过网络

- 远程过程调用
  - 在本地使用本地调用的方式进行编码，但实际上将参数和返回值进行编码、传输、远程执行、反编码等方式，从而调用远端进程中的某个函数或方法
  - 中间过程由两端的代理完成



基本原理


Client machine
Server machine
Client process
Server process
Implementation of add
k = add(i,j)
proc: "add"
int: val(i)
int: val(j)
1. Client call to procedure
Server stub
Client stub
2. Stub builds message
6. Stub makes local call to "add"
5. Stub unpacks message
4. Server OS hands message to server stub
Client OS
Server OS
3. Message is sent across the network

序列化和反序列化

- 序列化(Serialize)
  - 将参数和返回值编码成字节码，通过网络传输
- 反序列化(Deserialize)
  - 将网络接收到的字节码还原回参数和返回值

- 屏蔽参数和返回值类型的差异性，参数间分割
  - Java自带的Serialize/ Deserialize: 功能受限
  - 自己设计的数据协议：比如定义字节数
  - XML描述：自描述、可读性好。较大容量、性能较差

例子：Google Protobuf

```
message Person {
  required string name = 1;
  required int32 id = 2;
  optional string email = 3;
}
```

```
Person john = Person.newBuilder()
    .setId(1234)
    .setName("John Doe")
    .setEmail("jdoe@example.com")
    .build();
output = new FileOutputStream(args[0]);
john.writeTo(output);
```

```
Person john;
fstream input(argv[1],
    ios::in | ios::binary);
john.ParseFromIstream(&input);
id = john.id();
name = john.name();
email = john.email();
```

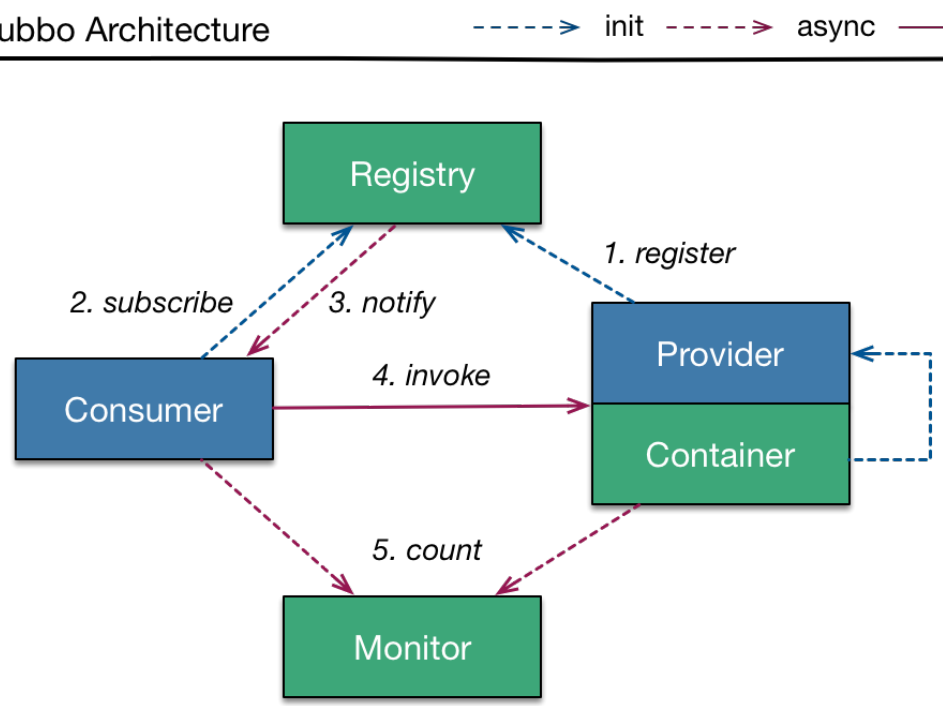
- 用户自定义数据结构Person，支持嵌套
  - 根据定义自动生成对应类代码Person和Builder
  - 提供get/set方法和API支持序列化和反序列化
- 多语言支持
  - Java, C++, C#, Go, Python
- 2进制数据编码，效率高
- 应用于Hadoop MapReduce, HBase

RPC框架

- 主要功能
  - 序列化/反序列化
  - 调用代理
  - 网络传输，基于TCP/IP或HTTP协议等实现高并发的请求发送和接收，并对异常进行处理
  - 提供灵活可配置的方式，包括远端地址、端口等
  - 为开发人员提供友好的APIs，尽可能屏蔽实现细节、像本地调用一样简单易用

例子: Apache Dubbo

Dubbo Architecture




```java
package com.alibaba.dubbo.demo;

public interface DemoService {
    String sayHello(String name);
}
```

```java
package com.alibaba.dubbo.demo.provider;
import com.alibaba.dubbo.demo.DemoService;

public class DemoServiceImpl implements DemoService {
    public String sayHello(String name) {
        return "Hello " + name;
    }
}
```

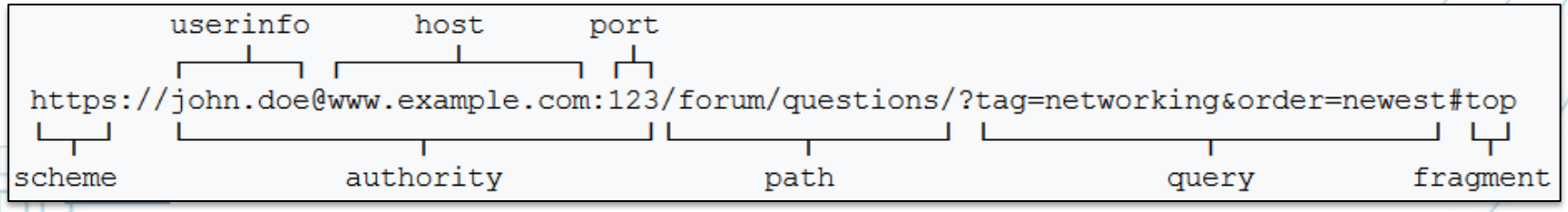
```java
public class Consumer {
    public static void main(String[] args) throws Exception {
        ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext(
                new String[]{"META-INF/spring/dubbo-demo-consumer.xml"});
        context.start();
        // obtain proxy object for remote invocation
        DemoService demoService = (DemoService) context.getBean("demoService");
        // execute remote invocation
        String hello = demoService.sayHello("world");
        // show the result
        System.out.println(hello);
    }
}
```

支持的编码和网络协议有：
Dubbo（自定义），HTTP, Hessian, RMI等

RESTful服务

URI

- 统一的资源标识符URI（Uniform Resource Identifier），用于定义网络上的某个资源
  - URI = scheme:[//authority]path[?query][#fragment]
  - scheme可以是http, https, ftp, mailto等网络协议
  - authority = [userinfo@]host[:port]用于安全认证
  - path: 描述资源的路径
  - query:参数，例如key1=value1&key2=value2
  - fragment: 资源内部的二级定位，比如某文章的章节



REST

- Representational State Transfer (REST) 是一种通过对URI的状态进行转换从而对外提供服务的架构模式
  - 特点：无状态、可缓存、标准接口等
- URL： 一般是基于HTTP/HTTPS的URI

HTTP methods

Uniform Resource Locator (URL)	GET	PUT	PATCH	POST	DELETE
Collection, such as `https://api.example.com/resources/`	**List** the URIs and perhaps other details of the collection's members.	**Replace** the entire collection with another collection.	Not generally used	**Create** a new entry in the collection. The new entry's URI is assigned automatically and is usually returned by the operation.[17]	**Delete** the entire collection.
Element, such as `https://api.example.com/resources/item17`	**Retrieve** a representation of the addressed member of the collection, expressed in an appropriate Internet media type.	**Replace** the addressed member of the collection, or if it does not exist, **create** it.	**Update** the addressed member of the collection.	Not generally used. Treat the addressed member as a collection in its own right and **create** a new entry within it.[17]	**Delete** the addressed member of the collection.

REST、RESTful和Web API

- REST
  - http://myserver.com/catalog/item/1729 Get
- RESTful
  - http://myserver.com/catalog?item=1729 Get
- 反例, 不是RESTful,但是都是Web APIs
  - http://myserver.com/addToCart?cart=314159&item=1729 Get
  - GET /delete_user.x?id=123
  - GET /user/new
- 在Web背景下，RESTful也称为CRUD
  - Create：POST
  - Read: GET
  - Update: PUT
  - Delete: DELETE

RESTful服务

Request

```
PUT /user/1
Accept: application/json+userdb
Content-Type: application/json+userdb

{
    "name": "Emil",
    "country": "Bhutan"
}
```

Rsponse

```
200 OK
Content-Type: application/json+userdb

{
    "user": {
        "id": 1,
        "name": "Emil",
        "country": "Bhutan",
        "links": [
            {
                "href": "/user/1",
                "rel": "self",
                "method": "GET"
            },
            {
                "href": "/user/1",
                "rel": "edit",
                "method": "PUT"
            },
            {
                "href": "/user/1",
                "rel": "delete",
                "method": "DELETE"
            }
```

RESTful服务开发框架

- Java Spring
  - Spring MVC
  - Spring Boot

```
//-------------------Retrieve Single User--------------------------------------------------------

@RequestMapping(value = "/user/{id}", method = RequestMethod.GET, produces = MediaType.APPLICATION_JSON_VALUE)
public ResponseEntity<User> getUser(@PathVariable("id") long id) {
    System.out.println("Fetching User with id " + id);
    User user = userService.findById(id);
    if (user == null) {
        System.out.println("User with id " + id + " not found");
        return new ResponseEntity<User>(HttpStatus.NOT_FOUND);
    }
    return new ResponseEntity<User>(user, HttpStatus.OK);
}



//-------------------Create a User--------------------------------------------------------

@RequestMapping(value = "/user/", method = RequestMethod.POST)
public ResponseEntity<Void> createUser(@RequestBody User user,    UriComponentsBuilder ucBuilder) {
    System.out.println("Creating User " + user.getName());

    if (userService.isUserExist(user)) {
        System.out.println("A User with name " + user.getName() + " already exist");
        return new ResponseEntity<Void>(HttpStatus.CONFLICT);
    }

    userService.saveUser(user);
```

- JAX-RS
- Jersey
- Restlet

Web服务

Web服务协议栈

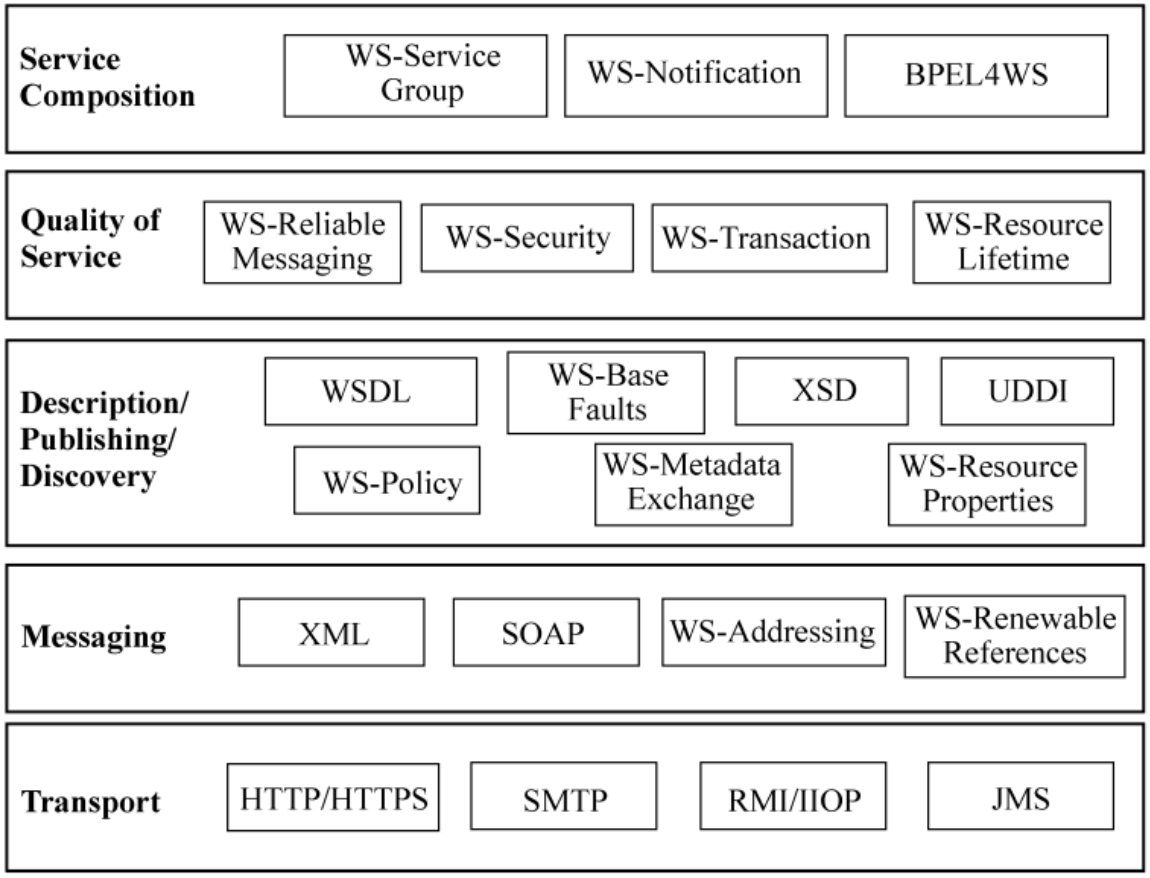


Figure 7.1 Web services standard stack

Web服务的核心功能与协议

- **WSDL：Web服务描述语言**
  - 用于服务接口的描述——What can the service do?
- **UDDI：统一描述、发现和集成协议**
  - 服务使用者通过UDDI发现相应的服务并据此将服务集成在自身的系统中——What kind of services are needed?
- **SOAP：简单对象访问协议**
  - 用户在服务客户端与服务提供者之间传递信息
  - 通过HTTP或JMS等各类基于文本的消息传递协议来运输——How can we interact with the service?

SOAP

SOAP发展历史

- SOAP最早由Dave Winner、Don Box和Bod Atkinson提出。
- 在1998年初，S0AP名字就已经被确定。Userland在1998年发布了一个XML-RPC规范。
- 1999年9月SOAP0.9提交IETF。
- 2000年5月8日，SOAP1.1作为Note提交W3C。IBM发布Java SOAP实现，并给开放源代码组织Apache XML Project。Sun公司将Web服务集成到J2EE中。
- 2000年9月13日，W3C组建了XML协议工作组，专门负责设计XML协议，以便成为基于XML分布式计算的核心。这个工作组将SOAP1.1作为基础，并于2001年7月9日提交了第一份工作组草案SOAP1.2。

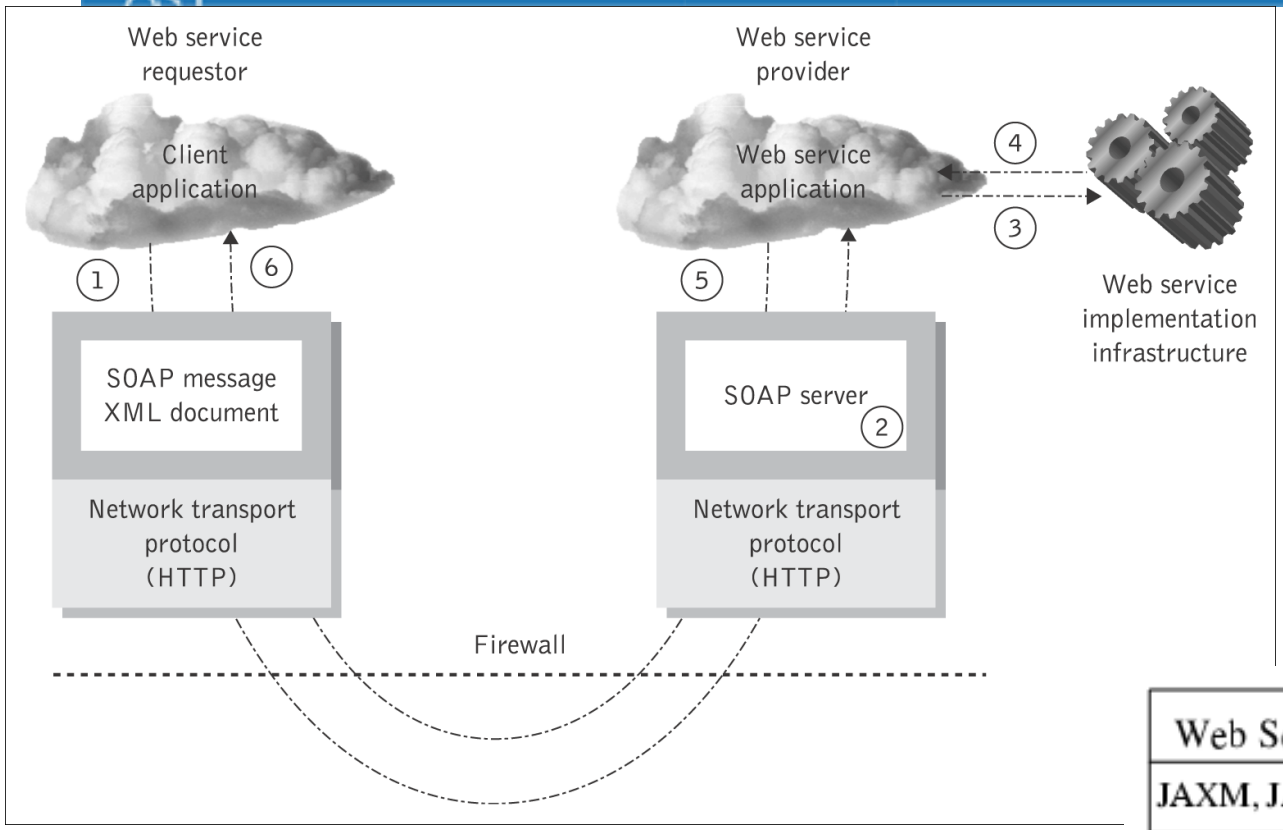
SOAP=HTTP+XML

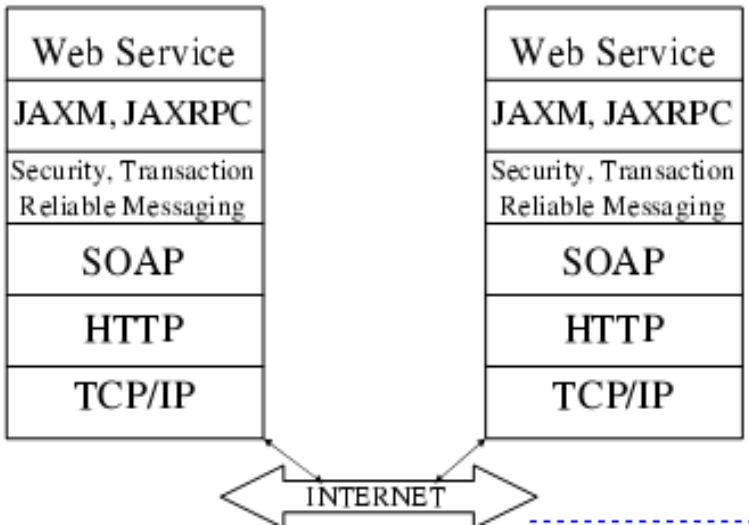
- SOAP(简单对象访 问协议)是为了解决由于用传统方式提供 Web服务所产生的问题而提出的。它有助于实现大量异构程序和平台之间的互操作，从而使存在的应用能够广泛地被用户所访问
- HTTP 作为传输协议
- XML 作为数据编码格式

SOAP概述

- SOAP 为在一个松散的、分布的环境中使用XML对等地交换结构化和类型化信息提供了一个简单且轻量级的机制。
- SOAP1.1简单对象访问协议(Simple Object Access Propotol)是Web服务的事实标准。
  - 支持应用程序与应用程序之间的通信；
  - 应用于商务对商务的通信以及企业应用集成。
- 以独立于各种编程语言或平台的方式来构造消息、处理消息，从而使用不同编程语言编写的程序之间具有互操作性，并能够在不同的操作系统上运行。
  - 通过对模块中特定格式编码的数据的重编码机制来表示应用语义。

SOAP概述


Web service requestor
Web service provider
Client application
Web service application
4
3
1
6
5
Web service implementation infrastructure
SOAP message XML document
SOAP server
2
Network transport protocol (HTTP)
Network transport protocol (HTTP)
Firewall


Web Service
JAXM, JAXRPC
Security, Transaction Reliable Messaging
SOAP
HTTP
TCP/IP
Web Service
JAXM, JAXRPC
Security, Transaction Reliable Messaging
SOAP
HTTP
TCP/IP
INTERNET

SOAP主要功能

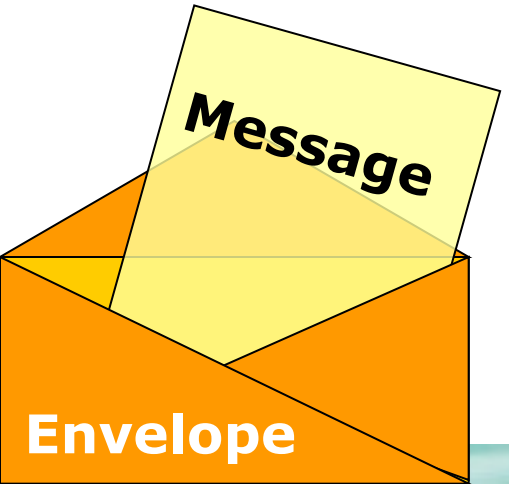
- 定义通信单元的机制：
  - 一个SOAP信封封装了所有其他的信息。
  - 一个消息可以有一个消息体，消息体中可以包含任何XML格式文档。
- 错误处理机制：
  - 标识错误源和导致错误的原因，并允许错误诊断信息在共享者和交互者之间传递。
- 可扩展件机制：
  - 使用XML模式和名字空间技术，灵活扩展元素。
- 灵活的数据表示机制：
  - 允许交换已经以某种格式序列化的数据，同时也提供了以XML格式表示诸如编程语言数据类型这样的抽象数据结构的规则。
- 支持以文档为中心的方法。
- 将SOAP消息绑定到HTTP的机制，因为HTTP是Internet上最常用的通信协议。

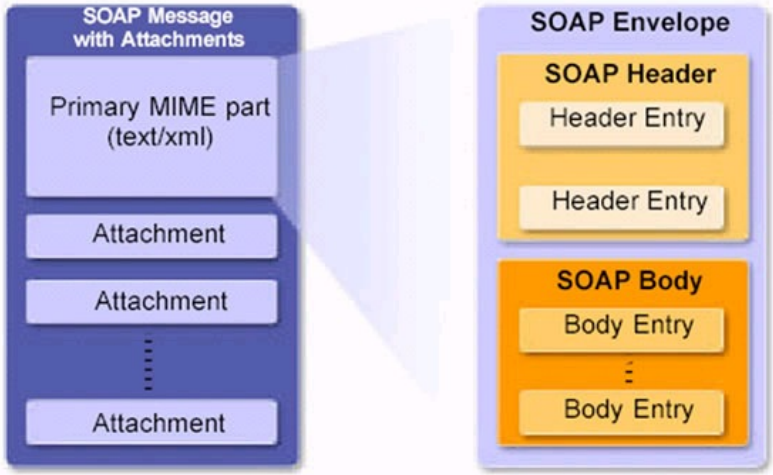
SOAP

- 基于XML的协议，由四部分组成:
  - 信封（Envelope）：定义了一个消息框架，描述消息的内容是什么，是谁发送的，谁应当接受并处理它以及如何处理。
  - 编码规则（Encoding Rules）：用于表示应用程序需要使用的数据类型的实例。
  - 绑定（Binding）：定义底层通信协议，进行消息交换。
  - RPC：表示远程过程调用和应答的协定。

SOAP

- SOAP信封包装传输的消息。SOAP定义4个XML元素：
  - 信封（env:Envelope）
  - 标题（env:Header）
  - 体（env:Body）
    一组和多组SOAP条目的信息。
  - 故障（env:Fault）
    协议层错误信息的特殊SOAP条目





SOAP用例

```
<env:Envelope xmlns:env="http://www.w3.org/2001/06/soap-envelope">
<env:Header>
    <n:alertcontrol xmlns:n="http://example.org/alertcontrol">
        <n:priority>1</n:priority>
        <n:expires>2010-06-22T14:00:00-05:00</n:expires>
    </n:alertcontrol>
</env:Header>
<env:Body>
    <m:alert xmlns:m="http://example.org/alert">
        <m:msg>Pick up Mary at school at 2pm</m:msg>
    </m:alert>
 </env:Body>
</env:Envelope>
```

SOAP结构

- SOAP消息是由一个SOAP Envelope、一个可选的SOAP Header和一个SOAP Body组成的XML文档。
- 元素和属性的命名空间标识是"http://www.w3.org/2001/06/soap-envelope"。SOAP消息应当包含如下部分：
- 一个SOAP envelope。
  - Envelope是表示该消息的XML文档的根元素。
- 一个SOAP Header。
  - Header是为了支持在松散环境下在通讯方之间尚未预先达成一致的情况下为SOAP消息增加特性的通用机制。
  - SOAP定义了很少的一些属性来用于指明谁可以处理该特性以及它是可选的还是强制的。
- 一个SOAP Body。
  - Body为该消息的最终接收者所想要得到的信息提供了一个容器。此外，SOAP定义了Body的一个子元素Fault用于报告错误。

SOAP Envelope 元素

- 必需的 SOAP 的 Envelope 元素是 SOAP 消息的根元素。它可把 XML 文档定义为 SOAP 消息。

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  ...
  Message information goes here
  ...
</soap:Envelope>
```

  - SOAP的命名空间：http://www.w3.org/2001/12/soap-envelope
  - SOAP的EncodingStyle属性用于定义在文档中使用的数据类型。

SOAP Header 元素

- SOAP提供了一个可伸缩的机制用于在分散的模块化的环境下扩展SOAP消息，而通讯双方并不需要有预先的约定知识
- 包含有关 SOAP 消息的应用程序专用信息
- 典型的扩展例子可以是实现一些诸如认证、事务管理以及支付的Header条目

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Header>
<m:Trans
xmlns:m="http://www.w3school.com.cn/transaction/"
soap:mustUnderstand="1">234</m:Trans>
</soap:Header>

...
...

</soap:Envelope>
```

SOAP Body 元素

- SOAP Body元素提供一个简单的用于与消息的最终接收者交换强制信息的机制
- Body元素在编码上作为SOAP Envelope元素的一个直接子元素。如果包含Header元素，则Body元素必须直接跟随Header元素，否则Body元素必须是Envelope元素的第一直接子元素。

- 必需的 SOAP Body 元素可包含打算传送到消息最终端点的实际 SOAP 消息。

- 包含SOAP请求和响应

SOAP请求消息—Envelope

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body>
   <m:GetPrice xmlns:m="http://www.w3school.com.cn/prices">
      <m:Item>Apples</m:Item>
   </m:GetPrice>
</soap:Body>

</soap:Envelope>
```

上面的例子请求苹果的价格。其中m:GetPrice 和 Item 元素是应用程序专用的元素。它们并不是 SOAP 标准的一部分。

SOAP响应消息—Envelope

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body>
   <m:GetPriceResponse xmlns:m="http://www.w3school.com.cn/prices">
      <m:Price>1.90</m:Price>
   </m:GetPriceResponse>
</soap:Body>

</soap:Envelope>
```

SOAP Fault 元素

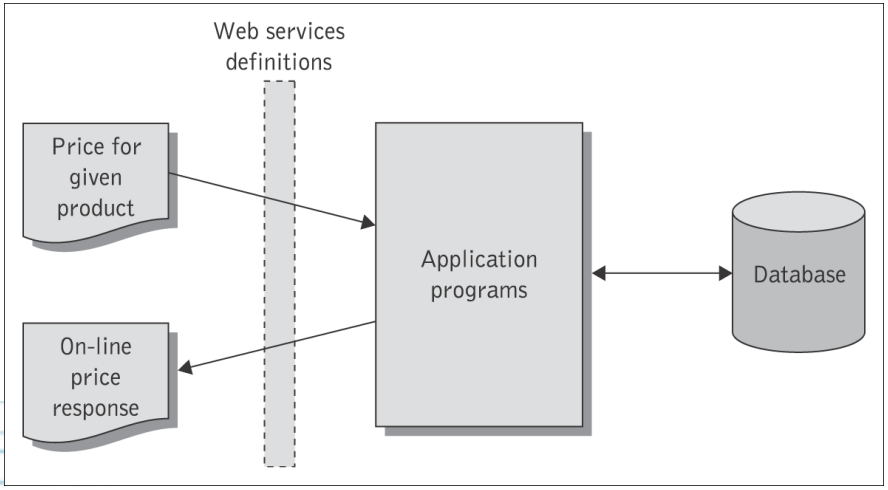
- 可选的SOAP Fault元素是用于在SOAP消息中传输错误或状态信息。如果SOAP消息需要包含SOAP Fault元素的话，它必须作为一个Body条目出现，同时在Body元素内它必须至多出现一次。

子元素	描述
<faultcode>	供识别故障的代码
<faultstring>	可供人阅读的有关故障的说明
<faultactor>	有关是谁引发故障的信息
<detail>	存留涉及 Body 元素的应用程序专用错误信息

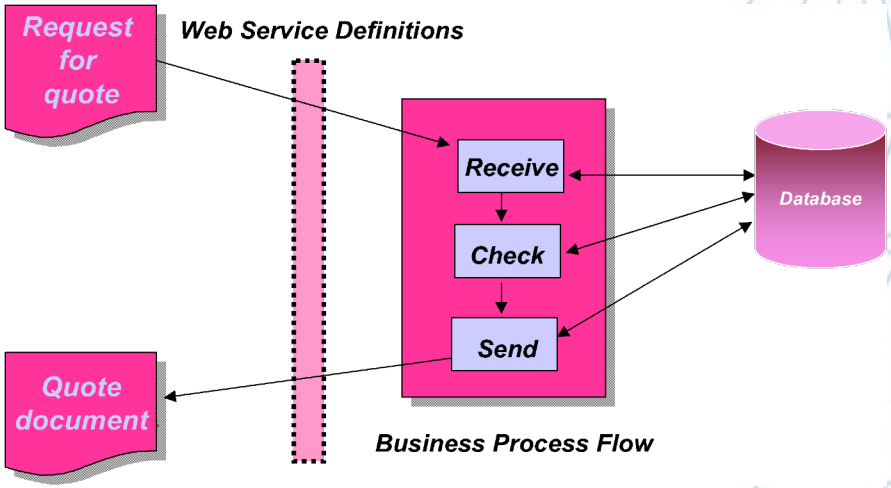
错误	描述
VersionMismatch	SOAP Envelope 元素的无效命名空间被发现
MustUnderstand	Header 元素的一个直接子元素（带有设置为 "1" 的 mustUnderstand 属性）无法被理解。
Client	消息被不正确地构成，或包含了不正确的信息。
Server	服务器有问题，因此无法处理进行下去。

SOAP通信模型

- SOAP 支持两种可能的通信方式:
  - remote procedure call (RPC) and
  - document (or message).
- 尽管Webservice是基于XML的，但仍然可以使用远程方法调用这种模式来进行Webservice的实现，尤其是在那种简单的请求相应的模型中。在这个过程中，传输中的XML文件所描述的更多是有关远程方法的信息，比如方法名，方法参数等等。
- 而文档交换方式，与RPC相比较在XML文件中不是做远程方法的映射，而是一份完整的自包含的业务文档，当Service端收到这份文档后，先进行预处理（比如词汇的翻译和映射），然后再构造出返回消息。这个构造返回消息的过程中，往往不再是简简单单的一个方法调用，而是多个对象协同完成一个事务的处理，再将结果返回。



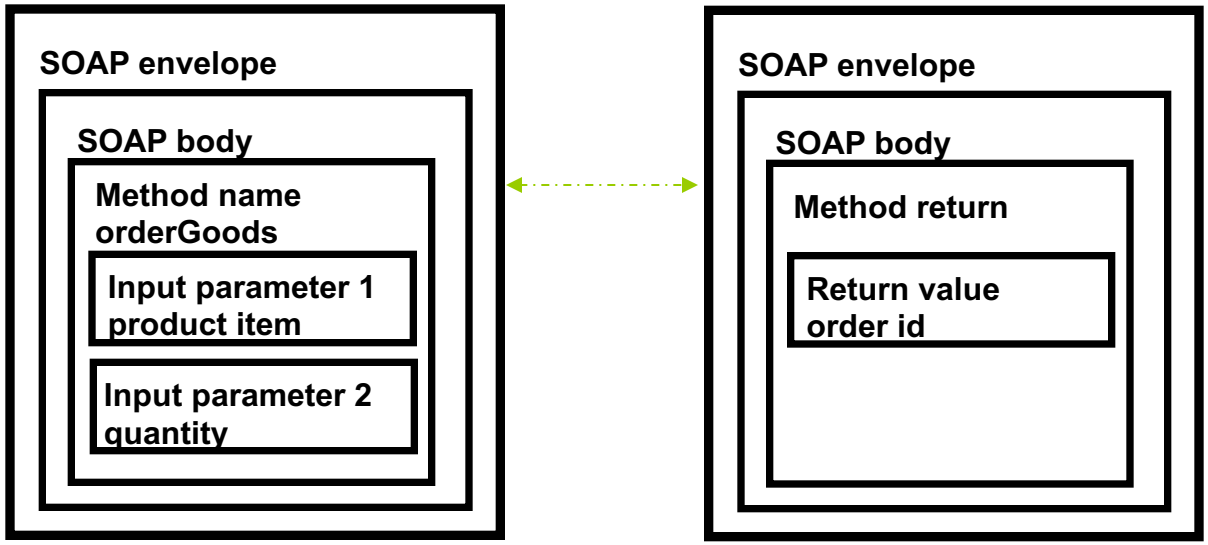
RPC-style interaction



Document-style interaction

RPC-style SOAP Services

- A remote procedure call (RPC)-style Web service appears as a remote object to a client application. The interaction between a client and an RPC-style Web service centers around a service-specific interface. Clients express their request as a method call with a set of arguments, which returns a response containing a return value.



RPC-style web services

```
<env:Envelope
 xmlns:SOAP="http://www.w3.org/2003/05/soap-envelope"
 xmlns:m="http://www.plastics_supply.com/product-prices">
    <env:Header>
            <tx:Transaction-id
           xmlns:t="http://www.transaction.com/transactions"
                env:mustUnderstand='1'>
               512
            </tx:Transaction-id>
    </env:Header>
    <env:Body>
            <m:GetProductPrice>
               <product-id>  450R6OP  </product-id >
            </m:GetProductPrice >
    </env:Body>
</env:Envelope>
```

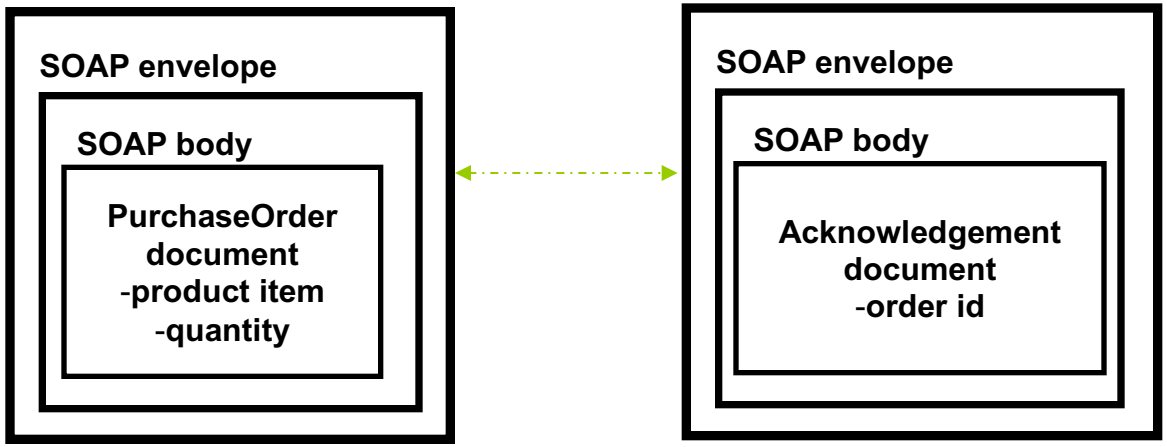
Example of RPC-style SOAP body

```
<env:Envelope
 xmlns:SOAP="http://www.w3.org/2003/05/soap-envelope"
 xmlns:m="http://www.plastics_supply.com/product-prices">
    <env:Header>
            <--! - Optional context information -->
    </env:Header>
    <env:Body>
            <m:GetProductPriceResponse>
               <product-price>   134.32  </product-price>
            </m:GetProductPriceResponse>
    </env:Body>
</env:Envelope>
```

Example of RPC-style SOAP response message

Document-style SOAP Services

- In the document-style of messaging, the SOAP <Body> contains an XML document fragment. The <Body> element reflects no explicit XML structure.
- The SOAP run-time environment accepts the SOAP <Body> element as it stands and hands it over to the application it is destined for unchanged. There may or may not be a response associated with this message.



Example of document-style SOAP body

```
<env:Envelope
 xmlns:SOAP="http://www.w3.org/2003/05/soap-envelope">

    <env:Header>
            <tx:Transaction-id
           xmlns:t="http://www.transaction.com/transactions"
                env:mustUnderstand='1'>
              512
    </env:Header>
    <env:Body>
            <po:PurchaseOrder oderDate="2004-12-02"
           xmlns:m="http://www.plastics_supply.com/POs">
           <po:from>
             <po:accountName>   RightPlastics   </po:accountName>
                  <po:accountNumber>   PSC-0343-02  </po:accountNumber>
           </po:from>
           <po:to>
             <po:supplierName>  Plastic Supplies Inc.  </po:supplierName>
             <po:supplierAddress>  Yara Valley Melbourne  </po:supplierAddress>
           </po:to>
           <po:product>
               <po:product-name> injection molder </po:product-name>
               <po:product-model> G-100T </po:product-model>
               <po:quantity> 2 </po:quantity>
           </po:product>
            </ po:PurchaseOrder >
    </env:Body>
</env:Envelope>
```

Example of document-style SOAP body

SOAP实例

- 在这个例子中，一个 GetStockPrice 请求被发送到了服务器。此请求有一个 StockName 参数，而在响应中则会返回一个 Price 参数。此功能的命名空间被定义在此地址中：
"http://www.example.org/stock"

SOAP实例—请求

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

  <soap:Body xmlns:m="http://www.example.org/stock">
    <m:GetStockPrice>
      <m:StockName>IBM</m:StockName>
    </m:GetStockPrice>
  </soap:Body>

</soap:Envelope>
```

SOAP实例—响应

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

  <soap:Body xmlns:m="http://www.example.org/stock">
    <m:GetStockPriceResponse>
      <m:Price>34.5</m:Price>
    </m:GetStockPriceResponse>
  </soap:Body>

</soap:Envelope>
```

WSDL

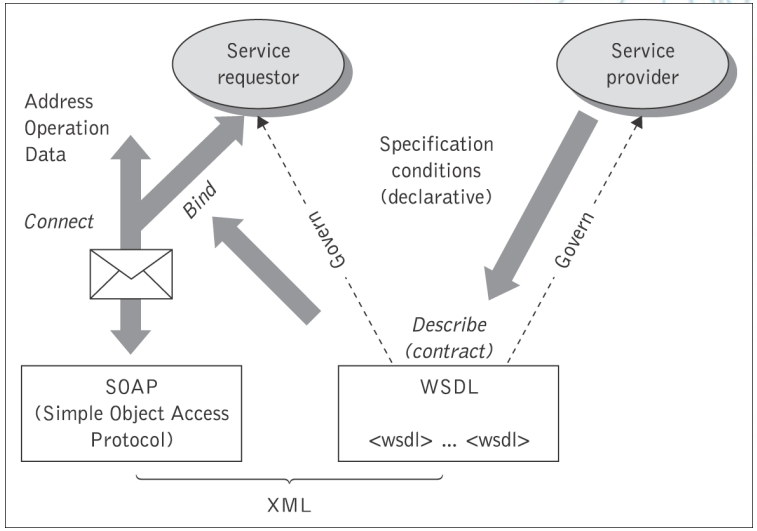
WSDL发展历史

- Web服务描述语言WSDL（Web Services Description Language）是一个建议性标准。用于描述Web服务的技术调用语法。
- 在Microsoft的SDL（Service Description Language）和SCL（SOAP Contract Language）和IBM的NASSL（Network Accessible Service Specification Language）这两项技术的结合，形成了WSDL的基础。SCL采用XML来描述应用程序所交换的消息，NASSL描述服务接口和实现细节。
- 2000年9月25日IBM、Microsft和Ariba提出WSDL1.0。2001年3月15日，他们提交的WSDL1.1成为W3C的Note。WSDL1.1规范网址是http://www.w3.org/TR/wsdl。2002年7月9日提出WSDL1.2，2003年11月10日提出WSDL2.0。

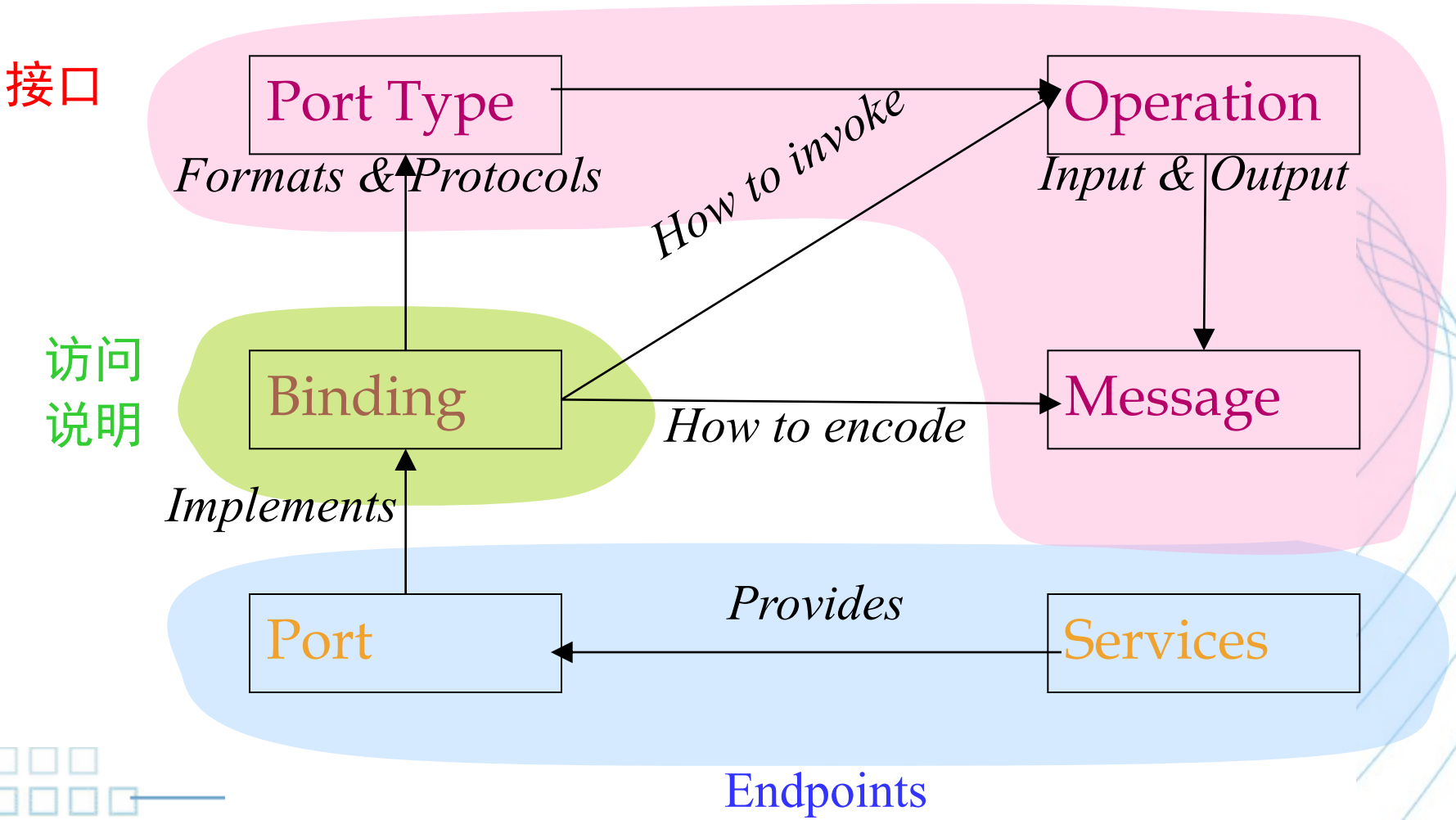
Web Services描述语言 (WSDL)

- WSDL是一种基于XML的接口定义语言，它分离功能和执行，给出了与特定Web服务交互的机制，使SOA建议的契约设计成为可能
- 它本质上为限制使用该服务的提供者和请求者，WSDL实质上是服务请求者和提供者之间的一个契约(Contract)
- WSDL独立于平台和语言，主要用于描述基于SOAP的服务

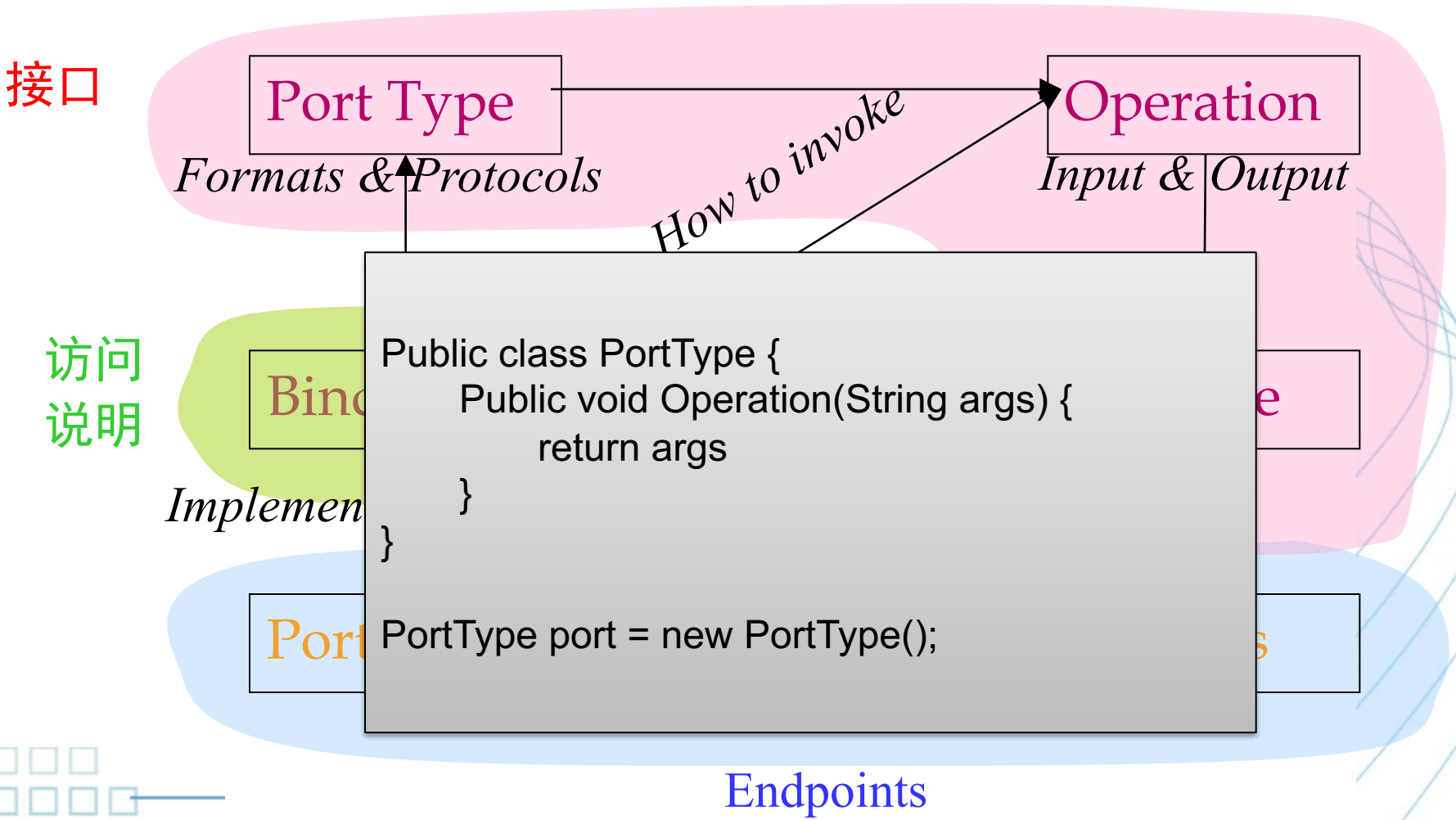
- WSDL 定义
  - **服务做些什么?**
    - 服务所提供的操作(方法)；
  - **如何访问服务？**
    - 数据格式以及访问服务操作的必要协议；
  - **服务位于何处？**
    - 由特定协议决定的网络地址，如URL。



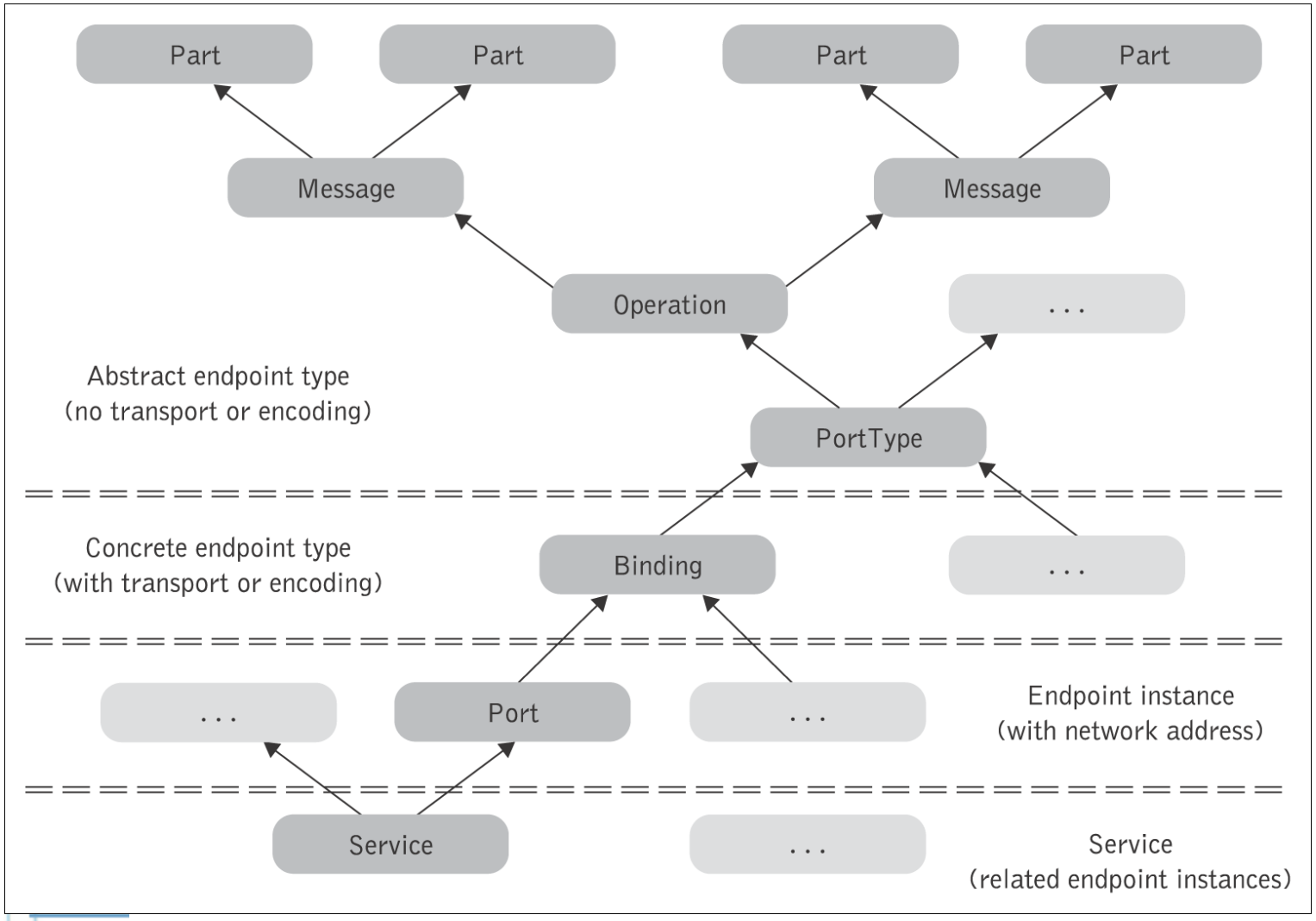
WSDL要素


接口
Port Type
Operation
Formats & Protocols
How to invoke
Input & Output
访问
说明
Binding
Message
How to encode
Implements
Port
Provides
Services
Endpoints

WSDL要素


接口
Port Type
Operation
Formats & Protocols
How to invoke
Input & Output
访问
说明
Bind
e
Implemen
Public class PortType {
Public void Operation(String args) {
return args
}
}
PortType port = new PortType();
Port
s
Endpoints

WSDL Elements Hierarchy



WSDL主体结构

```
<definitions namespace = "http://… ">
   <types> XML schema types </type>
   <message>  definition of a message消息定义 </message>
   <portType> a set of operations一组操作 </portType>
   <binding> communication protocols通信协议 </binding>
   <Services> a list of binding and ports绑定和端口列表 </Services>
</definitions>
```

类型

- types元素包含了交换消息的数据类型定义。为了实现最大的互操作性（interoperability）和平台中立性（neutrality），WSDL选用XML Schema DataTypes，简称XSD作为标准类型系统，并将它作为固有类型系统。
- <definitions .... >
- <types>
- <xsd:schema .... />*
- </types>
- </definitions>

类型

```
<types>
  <schema targetNamespace="http://example.com/stockquote.xsd"
                  xmlns="http://www.w3.org/2000/10/XMLSchema">
    <element name="TradePriceRequest">
      <complexType>
        <all>
          <element name="tickerSymbol" type="string"
                                        minOccur = "1"  maxOccur="10"/>
          <element name = "payment">
            <complexType>
              <choice>
                <element name = "account"     type="string" />
                <element name = "creditcard" type="string" />
              </choice>
            </complexType>
          </element>
        </all>
      </complexType>
    </element>
  </schema>
</types>
```

消息

- 消息由若干个逻辑部件（part）构成。每个部件使用一个消息类型属性与某个类型系统的类型相关联。
- 消息定义与法如下：
- <definitions .... >
- <message name="nmtoken"> *
- <part name="nmtoken"
- element="qname"? type="qname"?/> *
- </message>
- </definitions>
- 消息(message)name属性指定了消息的名称。
- 如果消息具有多个逻辑单位，则需要使用多个part元素。

```
<message name="GetLastTradePriceInput">
    <part name="body" element="TradePriceRequest"/>
</message>

<message name="GetLastTradePriceOutput">
    <part name="body" element="TradePrice" />
</message>
```

端口类型定义

- 端口类型是一个由抽象操作和抽象消息构成的有名称的集合。
- <wsdl:definitions .... >
- <wsdl:portType name="nmtoken"> *
- <wsdl:operation name="nmtoken">
- <wsdl:input name="nmtoken"? message="qname"/>
- <wsdl:output name="nmtoken"? message="qname"/>
- <wsdl:fault name="nmtoken" message="qname"/>*
- </wsdl:operation>
- </wsdl:portType >
- </wsdl:definitions>
- 端口类型定义的name属性表示端口类型名称，操作定义的name属性表示操作名称。

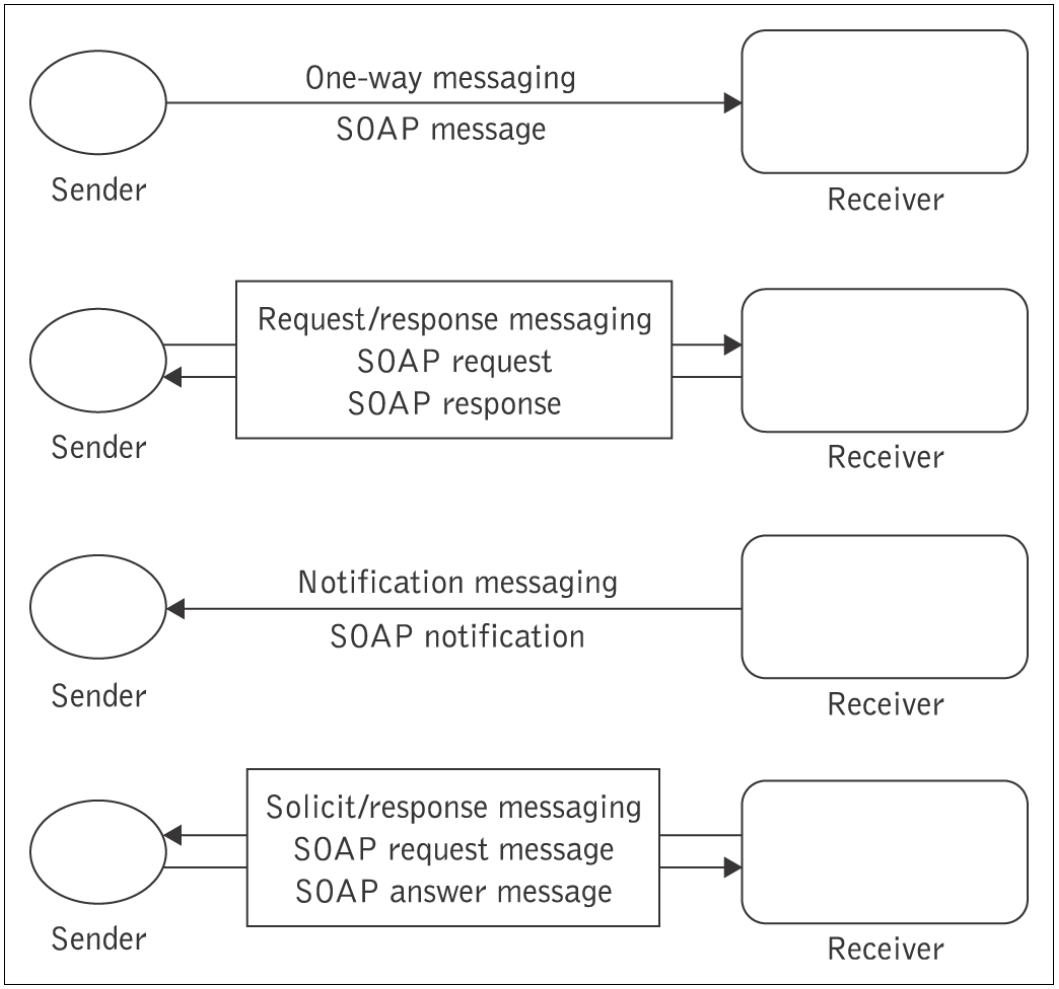
portType端口类型

```
<portType name="StockQuotePortType">
  <operation name="GetLastTradePrice">
      <input message="tns:GetLastTradePriceInput" />
      <output message="tns:GetLastTradePriceOutput" />
  </operation>
</portType>
```

操作

- WSDL支持4种消息交换方式，来访问服务端点。
  - 单向（One-way）：服务访问端点接收消息；
  - 请求响应（Request-response.）：服务访问端点接收请求消息，然后发送响应消息；
  - 要求应答（Solicit-response）：服务访问端点发送要求消息，然后接收应答消息；
  - 通知（Notification）：服务访问端点发送通知消息。
- 操作中引用到的消息通过message属性指定。

Message Exchange Patterns



单向操作

- 单向操作语法:
- <wsdl:definitions .... >
- <wsdl:portType .... > *
- <wsdl:operation name="nmtoken">
- <wsdl:input name="nmtoken"? message="qname"/>
- </wsdl:operation>
- </wsdl:portType >
- </wsdl:definitions>
- input元素指定用于单向操作的抽象消息格式。

请求响应操作

- 请求响应操作语法
- <wsdl:definitions .... >
- <wsdl:portType .... > *
- <wsdl:operation name="nmtoken"
- parameterOrder="nmtokens">
- <wsdl:input name="nmtoken"? message="qname"/>
- <wsdl:output name="nmtoken"? message="qname"/>
- <wsdl:fault name="nmtoken" message="qname"/>*
- </wsdl:operation>
- </wsdl:portType >
- </wsdl:definitions>

要求应答操作

- 要求应答操作语法
- <wsdl:definitions .... >
- <wsdl:portType .... > *
- <wsdl:operation name="nmtoken"
- parameterOrder="nmtokens">
- <wsdl:output name="nmtoken"? message="qname"/>
- <wsdl:input name="nmtoken"? message="qname"/>
- <wsdl:fault name="nmtoken" message="qname"/>*
- </wsdl:operation>
- </wsdl:portType >
- </wsdl:definitions>

通知操作

- 通知操作语法
- <wsdl:definitions .... >
- <wsdl:portType .... > *
- <wsdl:operation name="nmtoken">
- <wsdl:output name="nmtoken"?
- message="qname"/>
- </wsdl:operation>
- </wsdl:portType >
- </wsdl:definitions>

Operation Types操作类型

One-way:

```
<message name="newTermValues">
        <part name="term" type="xs:string"/>
        <part name="value" type="xs:string"/>
</message>
<portType name="glossaryTerms">
        <operation name="setTerm">
                <input name="newTerm" message="newTermValues"/>
        </operation>
</portType >
```

Request-response:

```
<message name="getTermRequest">
        <part name="term" type="xs:string"/>
</message>
<message name="getTermResponse">
        <part name="value" type="xs:string"/>
</message>

<portType name="glossaryTerms">
        <operation name="getTerm">
                <input message="getTermRequest"/>
                <output message="getTermResponse"/>
        </operation>
</portType>
```

Binding绑定

- 定义消息如何传递，以及服务的位置
- <binding> element有两个属性：
  - type: the port type
  - name: name of the binding
- <soap:binding>有两个属性：
  - style: either "document" or "rpc"
  - transport: protocol to use, e.g., "http"

Binding绑定

```
<binding name="StockQuoteSoapBinding“
      type="tns:StockQuotePortType">
  <soap:binding style=“document”
       transport=“http://schemas.xmlsoap.org/soap/http” />
  <operation name="GetLastTradePrice">
    <soap:operation
      soapAction=“http://example.com/GetLastTradePrice” />
    <input>
      <soap:body use=“literal” />
    </input>
    <output>
      <soap:body use=“literal” />
    </output>
  </operation>
</binding>
```

访问端点

- 访问端点（port）通过为绑定指定唯一地址来定义一个访问端点。访问端点语法如下：
- <wsdl:definitions .... >
- <wsdl:service .... > *
- <wsdl:port name="nmtoken" binding="qname">*
- <-- extensibility element (1) -->
- </wsdl:port>
- </wsdl:service>
- </wsdl:definitions>
- 在WSDL文档范围内，访问端点的name属性具有唯一性。

服务

- 服务是访问端点集合，服务语法如下：
- <wsdl:definitions .... >
- <wsdl:service name="nmtoken">*
- <wsdl:port ..../>*
- </wsdl:service>
- </wsdl:definitions>
- 在WSDL文档中，服务的name属性具有唯一性

Services

```
<Services name="StockQuoteServices">
    <documentation>
        My first Services
    </documentation>
    <port name="StockQuotePort"
                binding="tns:StockQuoteBinding">
        <soap:address
                location="http://example.com/stockquote" />
    </port>
</Services>
```

- 服务中的访问端点具有如下的关系：
  - 所有访问端点都不相互通信，即一个服务的访问端点的输出不会是另一个访问端点的输入。
  - 如果一个服务中有多个访问端点属于同一端口类型，但是对应不同的绑定或者地址，则这些访问端点可以相互替换。这使得WSDL文档使用者根据需要选择访问端点。
  - 通过检查访问端点可以确定服务的端口类型。这使得WSDL文档的使用者能够根据它支持的端口类型决定是否与一个特定的服务通讯。

Example

```
<?xml version="1.0"?>
<definitions name="StockQuote"

targetNamespace="http://example.com/stockquote.wsdl"
        xmlns:tns="http://example.com/stockquote.wsdl"
        xmlns:xsd1="http://example.com/stockquote.xsd"
        xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
        xmlns="http://schemas.xmlsoap.org/wsdl/">

    <types>
      <schema targetNamespace="http://example.com/stockquote.xsd"
            xmlns="http://www.w3.org/2000/10/XMLSchema">
        <element name="TradePriceRequest">
          <complexType>
            <all>
              <element name="tickerSymbol" type="string"/>
            </all>
          </complexType>
        </element>
        <element name="TradePrice">
          <complexType>
            <all>
              <element name="price" type="float"/>
            </all>
          </complexType>
        </element>
      </schema>
    </types>

    <message name="GetLastTradePriceInput">
       <part name="body" element="xsd1:TradePriceRequest"/>
    </message>

    <message name="GetLastTradePriceOutput">
       <part name="body" element="xsd1:TradePrice"/>
    </message>

    <portType name="StockQuotePortType">
        <operation name="GetLastTradePrice">
          <input message="tns:GetLastTradePriceInput"/>
          <output message="tns:GetLastTradePriceOutput"/>
        </operation>
    </portType>

    <binding name="StockQuoteSoapBinding"
type="tns:StockQuotePortType">
        <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="GetLastTradePrice">
          <soap:operation
soapAction="http://example.com/GetLastTradePrice"/>
          <input>
            <soap:body use="literal"/>
          </input>
          <output>
            <soap:body use="literal"/>
          </output>
        </operation>
    </binding>

    <service name="StockQuoteService">
       <documentation>My first service</documentation>
       <port name="StockQuotePort" binding="tns:StockQuoteBinding">
          <soap:address location="http://example.com/stockquote"/>
       </port>
    </service>

</definitions>
```

Web服务注册
UDDI

服务注册

- 通过在服务注册库中发布一个Web服务，其他的应用将能发现该服务，这需要两个同样重要的操作：Web服务的描述和注册
  - 服务发布需要从业务、服务和技术方面对Web服务进行合适的描述
  - 注册则涉及在Web服务注册库中持久化存储在Web服务的描述
    - 服务注册主要关于服务的辨别和控制
    - 在最简单的层次，服务注册记录企业所能提供的服务，以及那些服务的特性

服务发现

- 服务发现是SOA的一个重要基础
  - 服务发现的实质是确定Web服务提供者的位置，并获取已经发布的Web服务的描述
  - Web服务发现需要确定Web服务的位置以及了解Web服务的定义，这是访问Web服务的一项基本工作
    - Web服务客户端可以了解是否存在所需的特定Web服务，以及了解相关Web服务的能力和如何与Web服务进行合适的交互

服务发现的步骤

- 服务查询+选择

- 服务查询在注册库中查询满足服务请求者需求的Web服务
  - 查询由一些搜索条件组成
    - 所需的服务类型、首选价格、返回结果的最大数量
  - 查询将对服务提供者所发布的信息进行搜索
  - 在进行发现处理之后，服务开发者或者客户端应用将了解到Web服务的具体位置（所查找到的服务的URI）、Web服务的能力，以及如何与其进行交互
    - 对于服务发现返回的Web服务集合，服务选择将决定从中选择调用哪一个服务

两类服务发现的方式

- 静态服务发现
  - 通常发生在设计阶段
  - 确定了服务实现细节，并从服务注册库中检索服务
  - 设计者需要分析检索操作的结果，并将检索操作所返回的结果合并
- 动态服务发现
  - 发生在运行时，设计时并不确定具体的服务实现细节
  - Web服务请求者必须指定首选项
  - 应用将在服务注册库上进行检索操作，以确定与应用所使用的服务接口定义相匹配的一个或多个服务实现定义

通用描述、发现和集成规范UDDI

- Universal Description Discovery & Integration（UDDI)
  - **UDDI是一个跨行业的注册标准草案**
    - 基于该规范以及支持服务发布和发现处理的注册工具，可实现Web服务的描述和发现
  - **UDDI利用了W3C和IETF的一些标准，如XML、HTTP和DNS协议**
  - **UDDI的目的是供开发工具以及使用Web服务标准的应用使用**
  - **UDDI提供一个全球的、平台独立的、开放的框架，使得企业更容易开展业务、发现合作伙伴以及与这些合作伙伴在互联网上进行互操作**

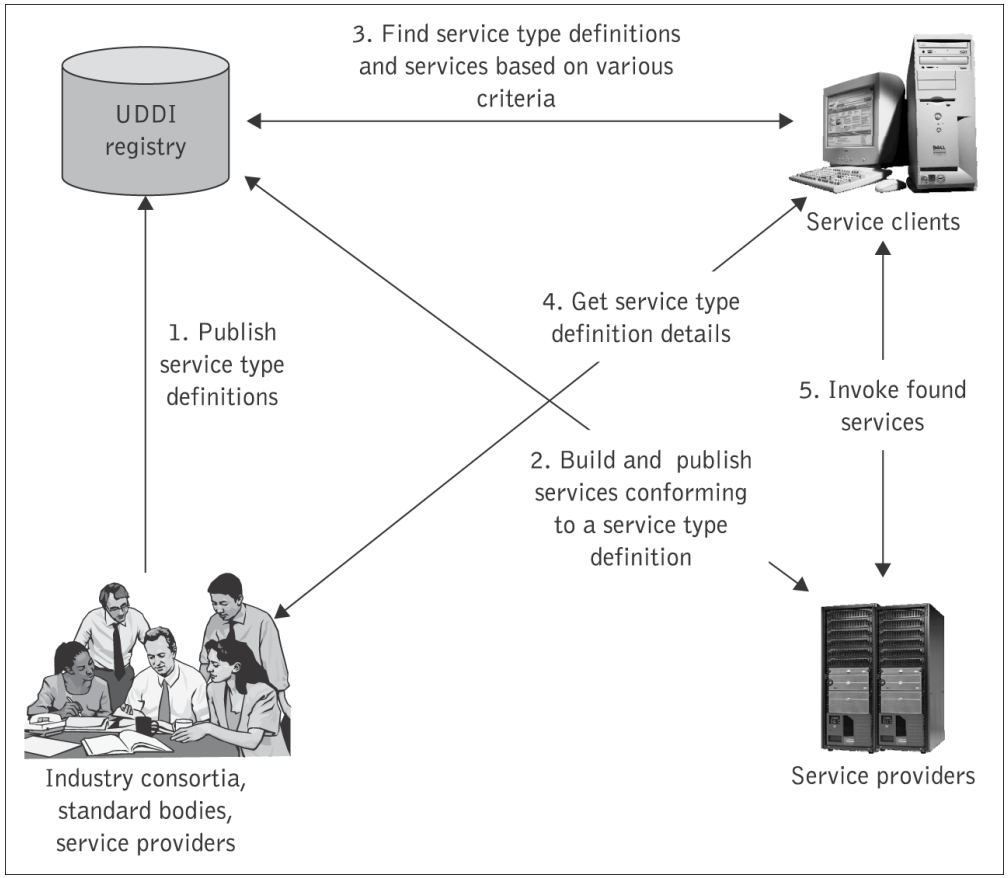
UDDI

- UDDI是一个包含轻量级数据的注册库
  - 目的是提供它所描述的资源（例如模式、接口定义和跨网络的端点）的网络地址
  - 核心概念是UDDI业务注册库，用来描述业务实体和它的Web服务的XML文档
  - UDDI业务注册提供的信息包含三个相关的组成部分
    - 白页：地址、联系方式以及其他的一些联系信息
      - 企业可以发现潜在的合作伙伴以及有关这些合作伙伴的基本信息
    - 黄页：基于行业分类法对信息进行分类
      - 可以发现按照具体行业进行分类的公司
    - 绿页：关于服务的业务能力和相关信息
      - 如何与提供服务的企业进行联系

UDDI

- UDDI是按标准化方式设计的，并不受限于任何技术
  - 注册库中的条目可以包含任何类型的资源，无论这些资源是否基于XML
    - UDDI注册中，可以包含企业电子文档交换系统的有关信息，DCOM或COBRA接口的有关信息
    - 企业使用基于SOAP的XML API调用与UDDI交互，发现企业服务的相关数据
- UDDI注册库与目录或其他注册库的主要的不同点在于
  - UDDI提供了按照分类法对业务和服务进行分类的一种机制
    - UDDI采用了标准的分类系统，因此可以基于分类法来发现相关信息

UDDI用例模型



概念区分

- UDDI协议 - Universal Description Discovery & Integration
  - OASIS组织提出
  - Web服务描述信息的发布、定位和发现
  - 有5个核心数据结构
  - 基于SOAP、操作这5个核心数据结构的API组成
- UDDI注册中心
  - 使用UDDI协议实现的服务注册中心
- 服务注册中心
  - 实现Web服务描述信息的发布、发现和定位
  - 有UDDI服务注册中心、ebXML服务注册中心等

UDDI Server的功能

- UDDI Server是一个通过实现UDDI v2或v3协议，建立核心数据结构，对外提供查询和发布等多种API，以Web Services或Servlet为展现方式，以SOAP为通信协议，为服务请求者提供服务查询，服务提供者发布服务信息的服务注册中心。
- UDDI Client可以通过UDDI4J或网页形式来查询和发布服务注册信息。

UDDI Servers的特征对比

Server	厂商	授权方式	我们的获得相关资料的情况	验证过的部署环境	支持UDDI标准
Web Sphere UDDI Registry	IBM	商业软件	评估版和文档	Web Sphere	V2.0
Enterprise UDDI Service	Microsoft	商业软件	程序和文档	Windows Server 2003 Family, Active Directory	v1.0, v2.0
Nsure UDDI Server	Novell	商业软件 开放源码	源码、程序和文档	Netware, eDirectoty	v2.0
WASP UDDI Server	Systinet & Sun	商业软件	文档，评估版本	Solaris, Sun ONE App Server	V1.0, 2.0以及v3.0的一个子集
SOAPUDDI	SourceForge.net	开放源码	源码和文档	Linux, AIX, Unix, Windows	V2.0
JUDDI	SourceForge.net	开放源码	源码和文档	Redhat Linux 9, Tomcat	V2.0

UDDI数据结构

- UDDI的主要目的是Web服务的数据和元数据表示
- 无论是在公共域还是在防火墙后使用，UDDI注册库都提供了对Web服务分类、编目和管理的机制，从而可以发现和使用那些Web服务
  - 发现Web服务实现：基于公共的抽象接口定义
  - 发现Web服务提供者：按照分类模式或标识系统进行分类
  - 基于常规的关键字搜索服务
  - 确定一个特定的Web服务所支持的安全性和传输协议
  - 存储Web服务的技术信息，并在运行时更新那些信息

UDDI数据表类型

- １．白页：包含了基本的企业信息，诸如企业名称、文字性介绍（可能是多国语言）以及联系方式，包括名称、电话号码、电子邮件以及属于这些企业的网站。
- ２．黄页：按分类法对企业信息进行分类，在UDDI的第一个版本中，这种分类法包括了对行业、产品或服务以及位置的分类。
- ３．绿页：包含了如何与企业进行电子交互的信息，包含交易过程（也就是，创建订单和检查存货等多种WEB服务）、服务描述（个人Web服务和它们的用途）以及解释如何通过调用一个给定的Web服务的绑定信息。

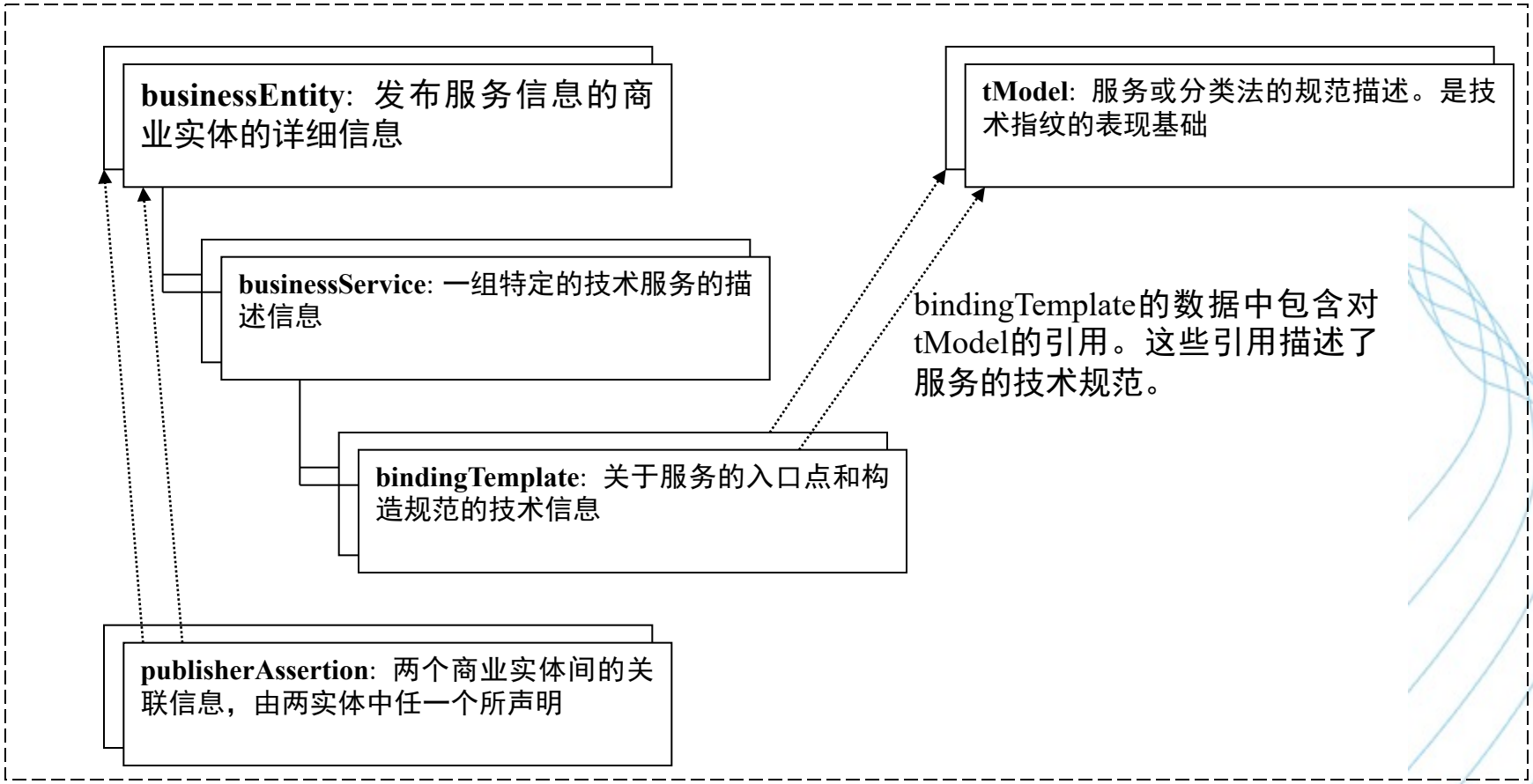
UDDI数据结构

- 对于表示公司和服务描述信息，UDDI定义了一个数据结构标准
  - 在XML模式中定义了UDDI注册库所使用的数据模型
    - XML提供一个平台中立的数据视图，并可以以中立方式来描述层次关系
  - UDDI XML模式定义了提供白页、黄页、绿页功能的四类核心信息类型
    - 业务实体
    - 业务服务
    - 绑定模板
    - 服务规范（技术或tModel）

UDDI核心数据结构

- businessEntity
  - Representing Businesses and Providers
- businessService
  - Representing Services
- bindingTemplate
  - Representing Web services
- tModel
  - Technical Models
- publisherAssertion

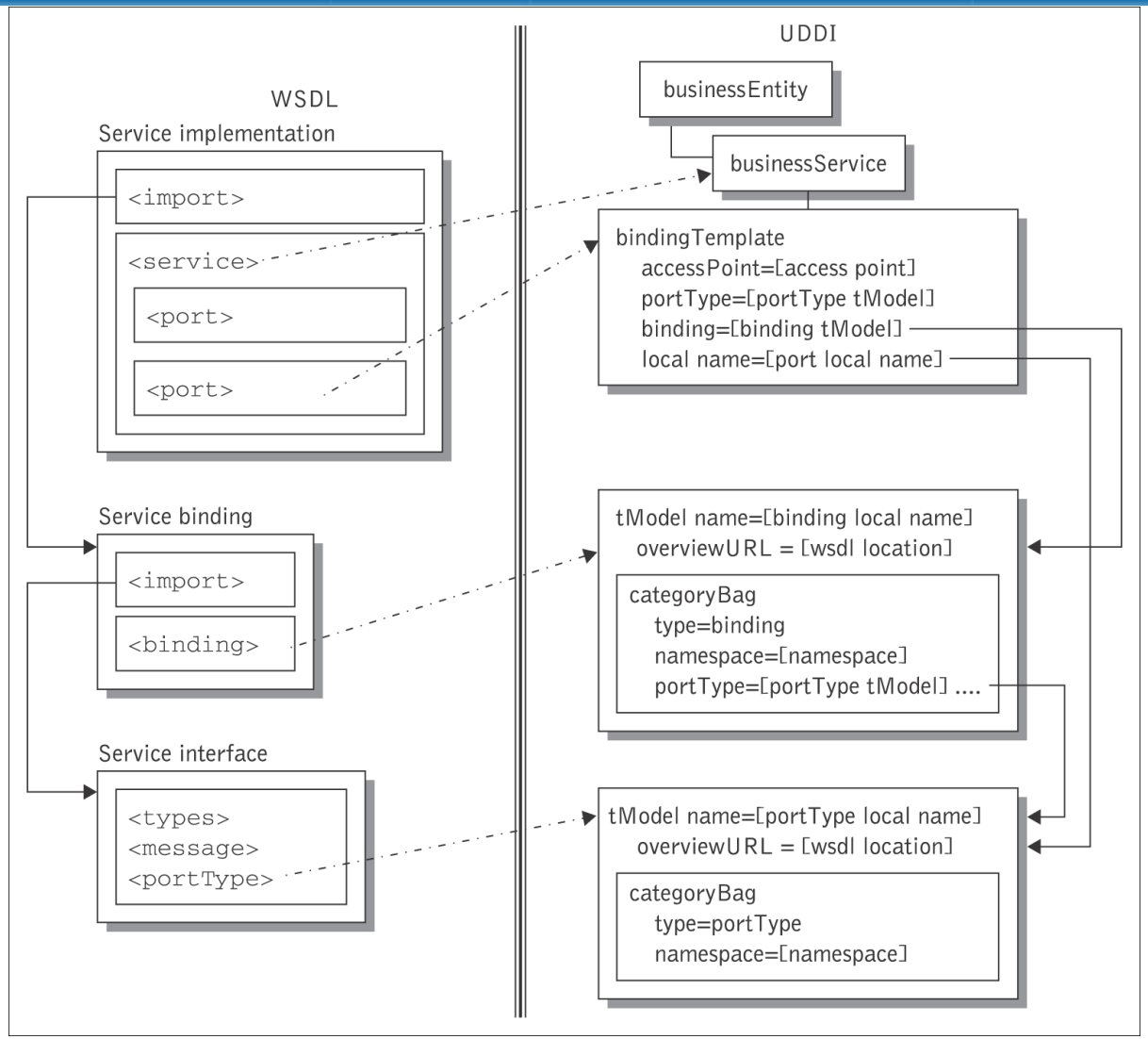
UDDI数据结构


businessEntity: 发布服务信息的商业实体的详细信息
tModel: 服务或分类法的规范描述。是技术指纹的表现基础
businessService: 一组特定的技术服务的描述信息
bindingTemplate的数据中包含对tModel的引用。这些引用描述了服务的技术规范。
bindingTemplate: 关于服务的入口点和构造规范的技术信息
publisherAssertion: 两个商业实体间的关联信息，由两实体中任一个所声明

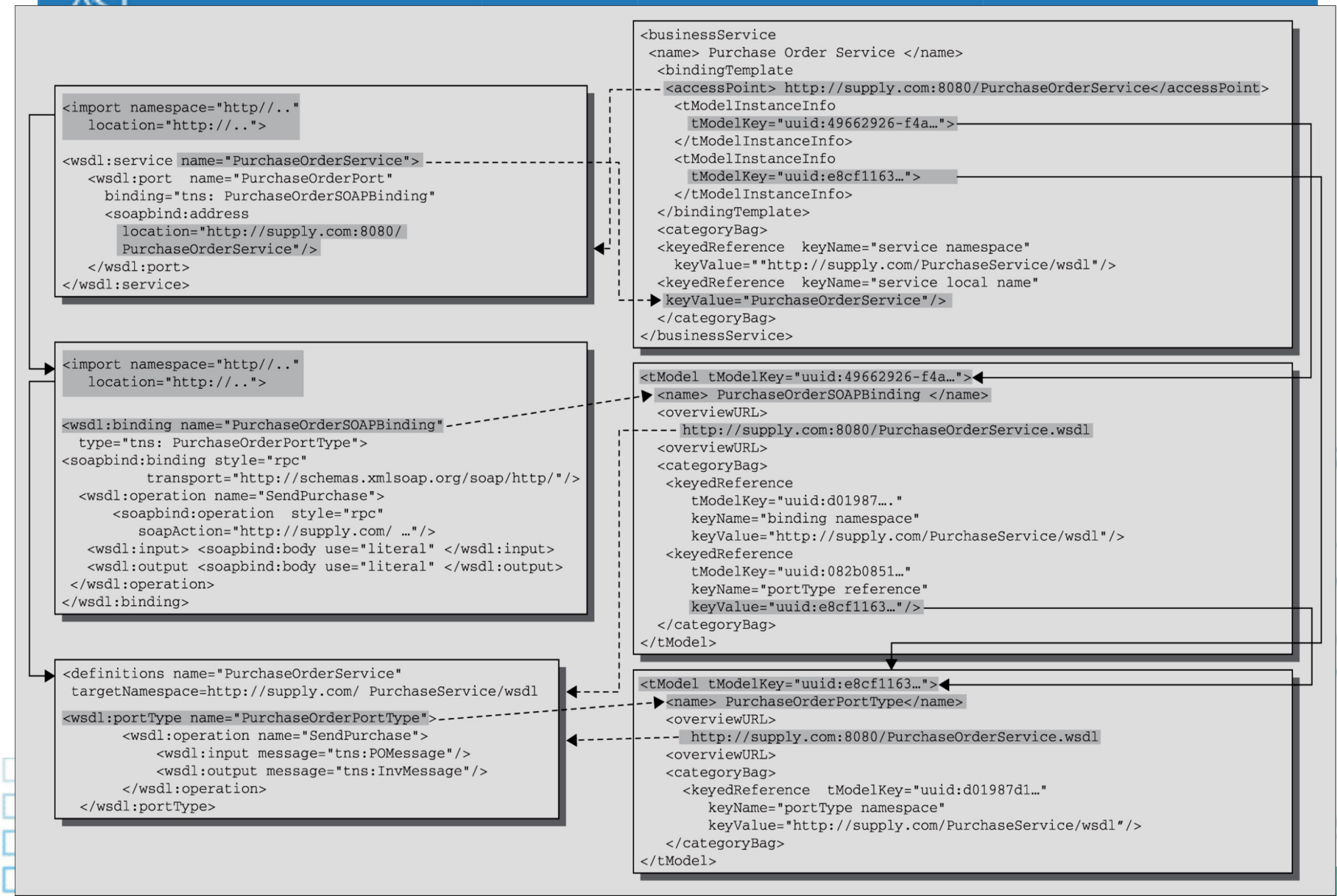
WSDL到UDDI的映射模型

- 怎样根据WSDL文档获得UDDI中服务的描述
- UDDI和WSDL都清晰地、系统地刻画了接口和实现，它们可以相互补充、相互协作
- WSDL到UDDI的映射模型可帮助用户发现那些实现了标准定义的服务。映射模型描述了：
  - WSDL portType元素和binding元素规范如何变成tModel
  - WSDL port如何变成UDDI bindingTemplate
  - 每个WSDL服务如何注册为businessService
- 对于UDDI业务和服务条目中的信息，WSDL文档中定义的服务信息是对其的一个补充。UDDI的目标是提供多种类型的服务描述，不直接支持WSDL

WSDL到UDDI的映射模型



WSDL到UDDI的映射模型



UDDI API规范

- UDDI API规范概述了公共可调用SOAP接口在UDDI站点上执行的每项操作
- 由两部分组成
  - Inquiry API，用于查询和浏览UDDI注册表来发现最终用户查询的企业和服务；
  - Publisher API，用于添加、更新和删除UDDI注册表中的企业和服务信息。

查询用API

- 每个UDDI数据结构（businessEntity,businessService,bindingTemplate和tModel）都有一个find_xxx和get_xxx函数。这8个函数构成了查询API。它允许用户在数据实体上的注册表中搜索关键词或者值，然后给出所有与这个条目相关的数据。这个API主要作为查找和显示最终用户想查找的企业、服务等的一种方法。
- Find_xxx一般是用于定位特定的服务，get_xxx一般是用于得到完整的信息。

发布用API

- 每个UDDI数据结构都有一个save_xxx和delete_xxx函数。加上权限认证函数（get_authToken,discard_authToken)这些函数形成了Pulication(发布)API，它允许用户（经过注册授权的用户）对现有的注册标目进行更新，用save_xxx创建新的条目，用delete_xxx能完全删除给出的数据结构。但是用户必须是已经授权的终端使用者。

API举例：save_business 语法

- <save_business *generic*="2.0" xmlns="urn:uddi-org:api_v2" >
- <authInfo/>
- <businessEntity/> [<businessEntity/>…]
- </save_business>

API举例： save_business 参数

- *authInfo*:
  - 这个参数是必需的，它是一个包含了认证令牌的元素。认证令牌可以使用 get_authToken API调用来获得。
- *businessEntity*:
  - 一个或多个的完整的businessEntity结构可以被传入。这些结构可以事先通过get_businessDetail API调用或者其他方式获得。

API举例： save_business 行为

- 如果在businessEntity结构中的任一*uuid_key*键值为空，该条数据将注册。
- 如果一个或多个businessService和bindingTemplate在注册中心中出现，但本调用提供的businessEntity信息中不出现，则从注册信息中删除。
- 如果一个被保存的businessEntity包含了一个businessService元素，并且这个businessService元素所包含的businessKey键值与待保存的这个businessEntity的businessKey不相等，则UDDI注册中心会记录一个引用，称为”服务散出引用(service projection)”，该引用指向那个已有的businessService。

API举例： save_business 返回

- 本API返回一个businessDetail消息
- 这个bindingDetail消息包含有本调用所涉及的所有businessEntity的最新注册信息
- 这个返回结果将包含所有通过引用所包含的businessService。

其他功能及规范

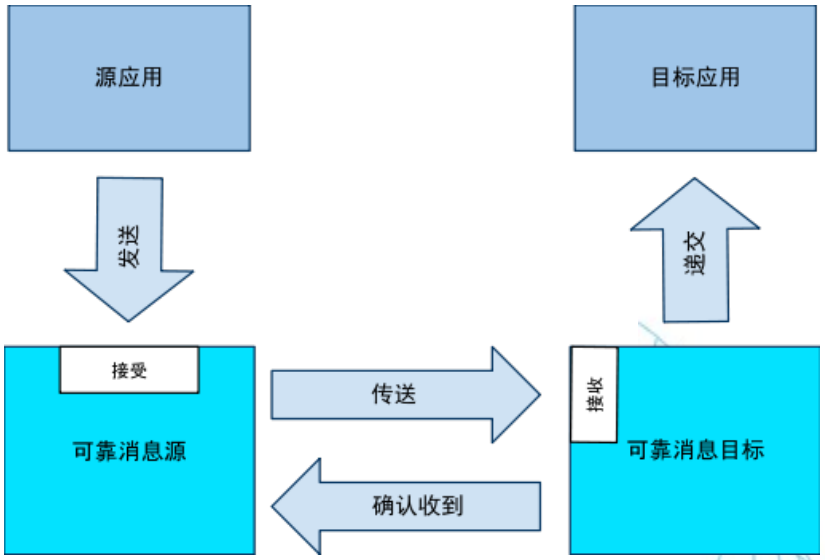
WSRF

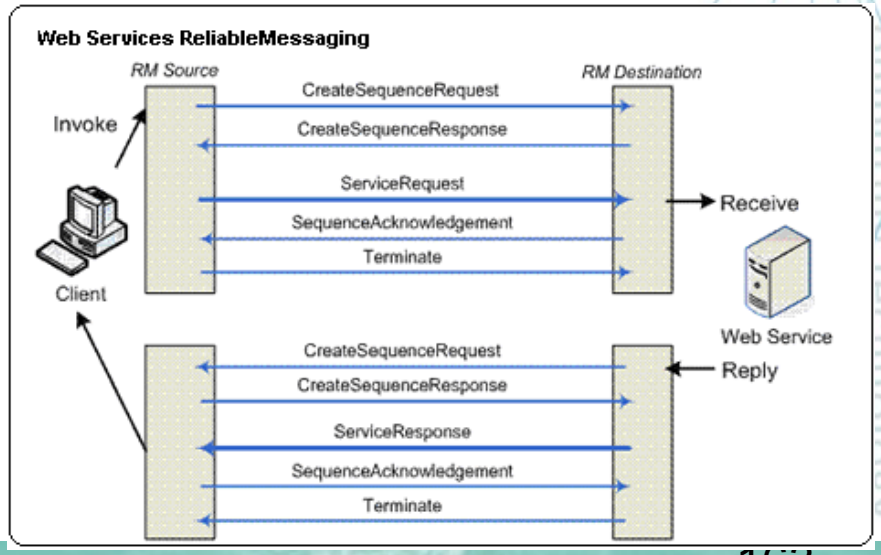
- Web Services Resource Framework
- 有状态服务

Name	Description
WS-ResourceProperties	Describes associating stateful resources using Web services, and how elements of publicly visible properties of a resource are, retrieved, changed, and deleted.
WS-ResourceLifetime	Allow a requestor to destroy a WS-Resource either immediately or at a scheduled future point in time.
WS-RenewableReferences	Annotate a WS-Addressing endpoint reference with information needed to retrieve a new endpoint reference when the current reference becomes invalid.
WS-ServiceGroup	Create and use heterogeneous by-reference collections of Web services.
WS-BaseFault	Describes a base fault type used for reporting errors.
WS-Notification	Standard approaches to notification using a topic-based publish and subscribe pattern.

可靠消息

- Web Services Reliable Messaging(WSRM)
- 定义一个消息协议，用于确认、跟踪、管理两个节点间消息的可靠分发
  - AtMostOnce
  - AtLeastOnce
  - ExactlyOnce
  - InOrder



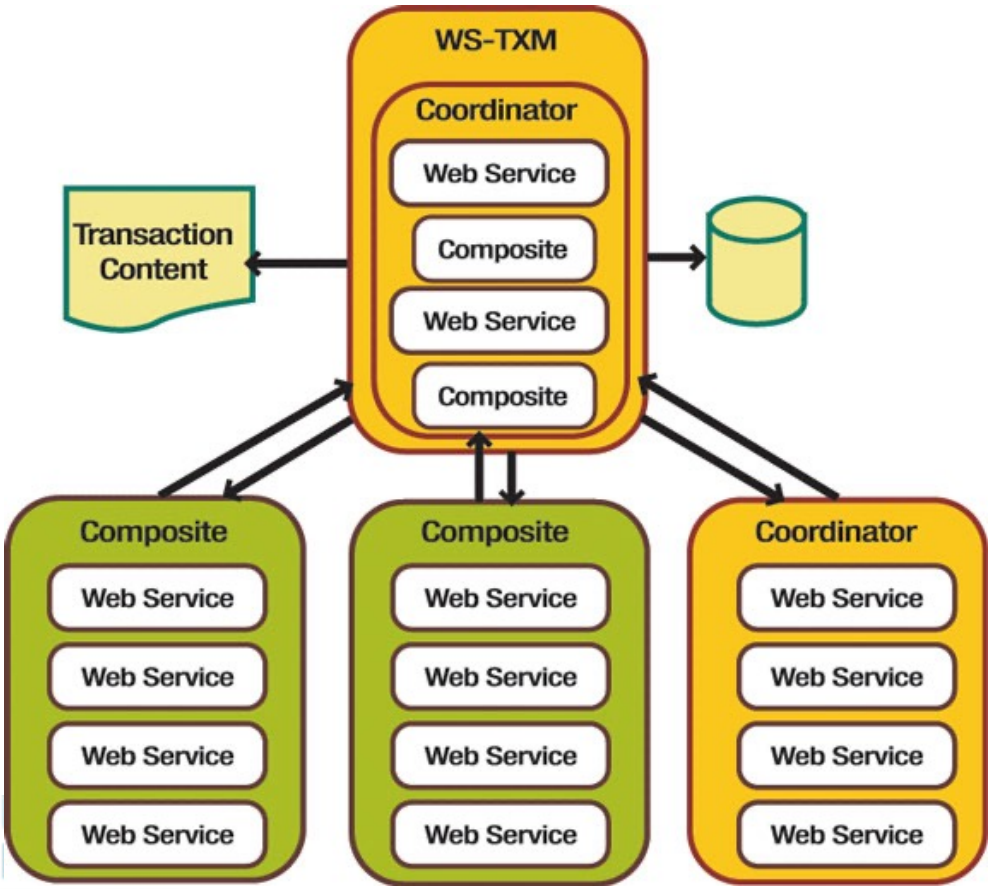


事务

- ACID (atomicity, consistency, isolation, durability)事务隐含
  - 紧耦合系统
  - 短时的活动
- 基于Web服务的B2B通信则包括：
  - 长期运行的计算
  - 松耦合系统
  - 组件不共享数据和位置

事务架构

- Web Services Transaction Management (WS-TXM)



WS-TXM定义了一个可插拔事务协议的集合，用于coordinator与所有参与者协商动作以协调相关Web服务的执行

通过共享的contex将服务执行关联起来

Web服务事务的可选规范

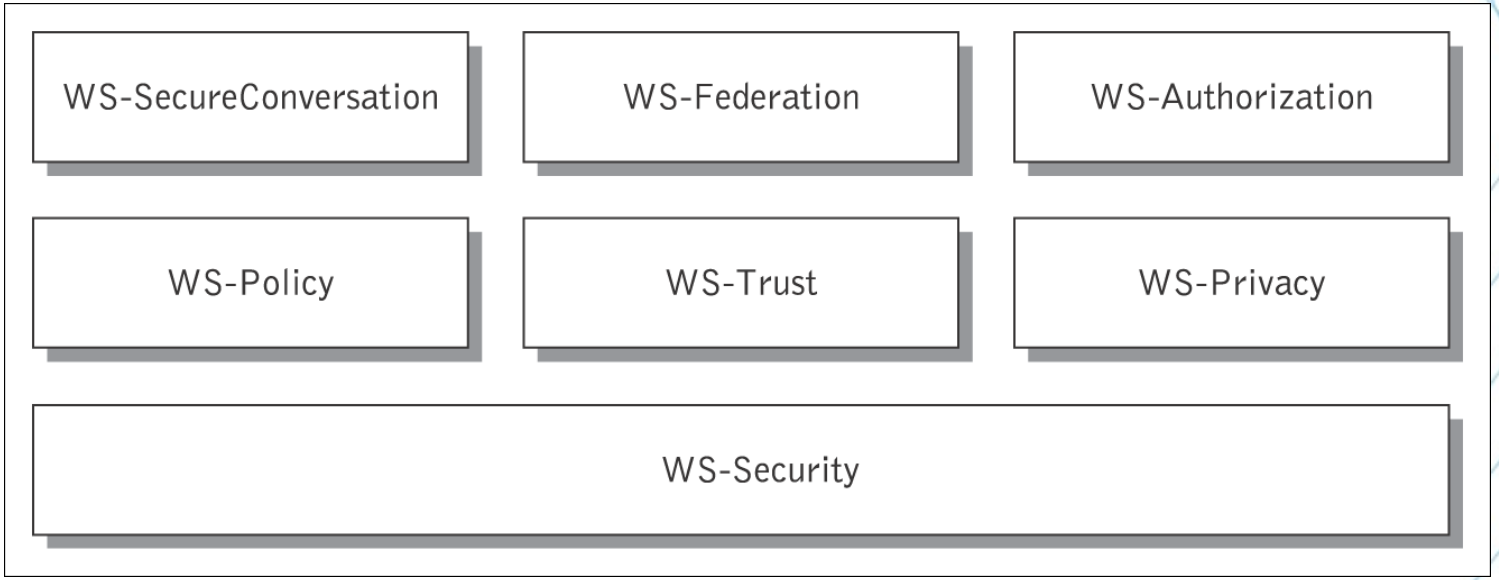
- OASIS-BTP:
  - 第一个标准(HP, Sun BEA, Oracle等)
  - 定义两个事务模型(原子、内聚cohension)
  - 不解决事务互操作
  - HP, Choreology, Collaxa等提供参考实现
- WS-C/T
  - IBM, Microsoft and BEA等发布的专有规范
  - 将协调coordination从事务分离出来
  - 定义两个事务模型：原子事务、业务活动
  - 简单性、与已有重要协议的互操作性

安全

- XML Security Standards
  - **XML Trust Services** is a suite of open XML specifications for application developers to make it easier to integrate a broad range of XML security services into integrated applications over the Web. The main technologies for XML Trust Services encompass:
  - **XML Signature** for cryptographically authenticating data;
  - **XML Encryption** for encrypting data;
  - **XML Key Management Specification (XKMS)** for managing key registration and key authentication;
  - **Security Assertions Markup Language (SAML)** for specifying entitlement and identity and
  - **XML Access Control Markup Language (XACML)** for specifying fine-grained data access rights.

Web Services Security Roadmap

- The Web services security roadmap is used for developing a set of Web service security standard specifications and technologies. They describe a unifying approach for dealing with protection for messages exchanged in a Web services environment.



总结

- 分布式系统基础知识
- 网络服务相关协议
- 三种类型网络服务
  - RPC
  - RESTful服务
  - SOAP Web服务

结 束!