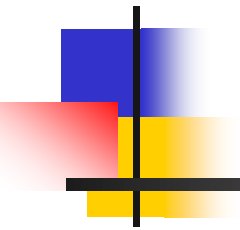
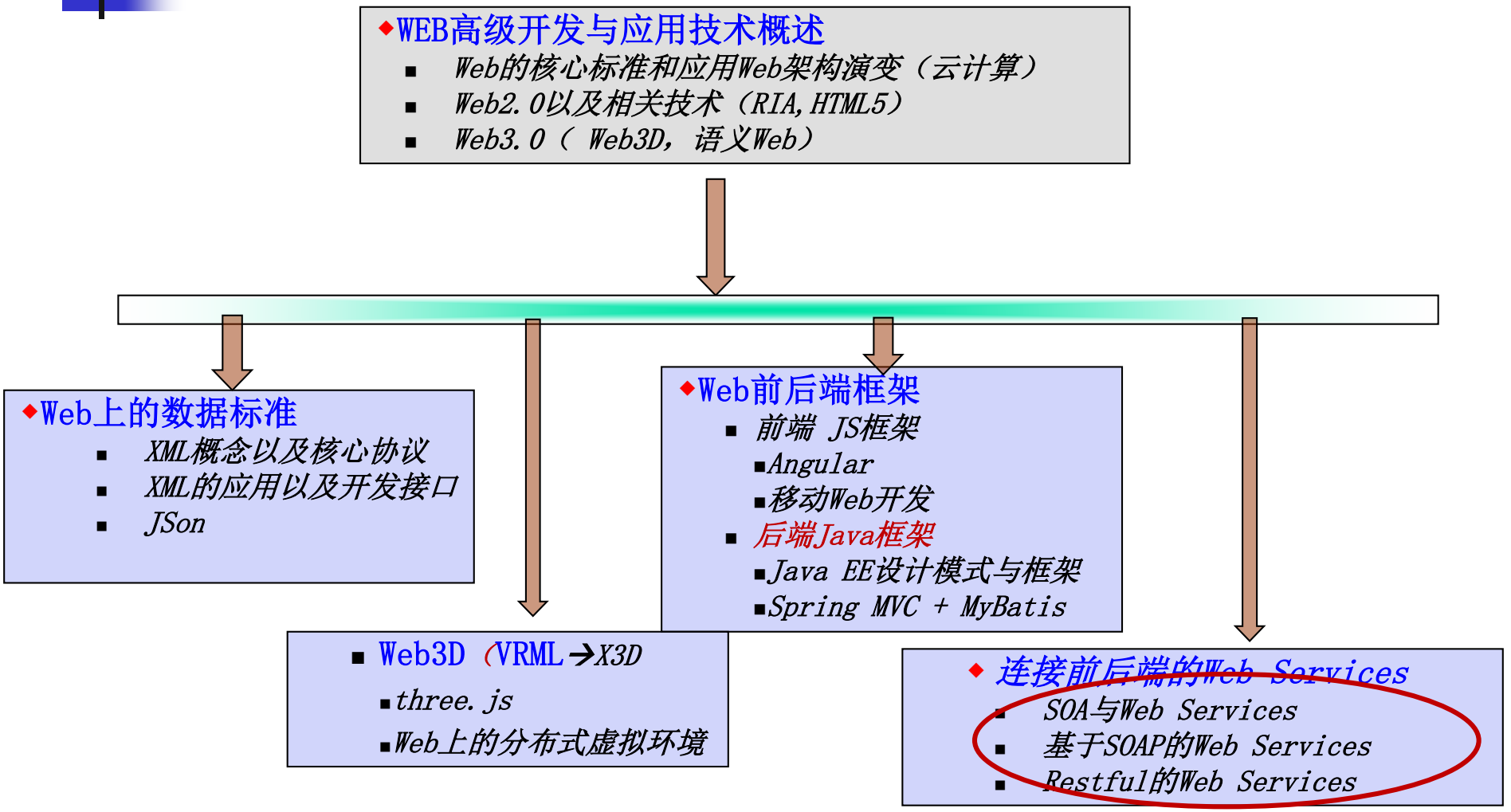
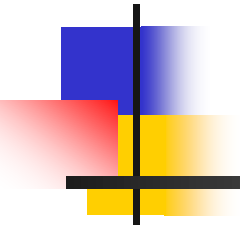


高级Web技术

Web服务

课程内容组织

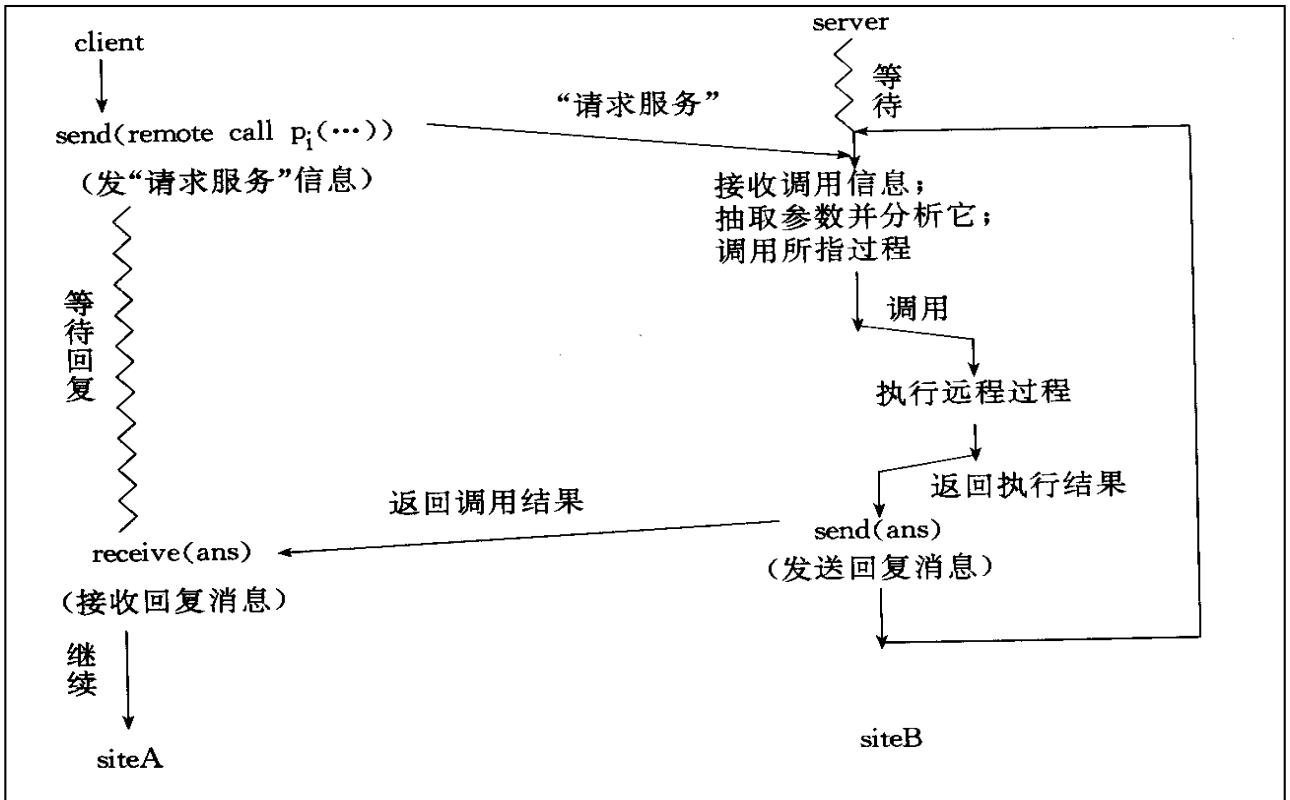

◆WEB高级开发与应用技术概述
Web的核心标准和应用Web架构演变（云计算）
Web2.0以及相关技术（RIA,HTML5）
Web3.0（ Web3D，语义Web）
◆Web上的数据标准
XML概念以及核心协议
XML的应用以及开发接口
JSon
Web3D （VRML→X3D
three.js
Web上的分布式虚拟环境
◆Web前后端框架
前端 JS框架
Angular
移动Web开发
后端Java框架
Java EE设计模式与框架
Spring MVC + MyBatis
◆连接前后端的Web Services
SOA与Web Services
基于SOAP的Web Services
Restful的Web Services



Web 服务概述

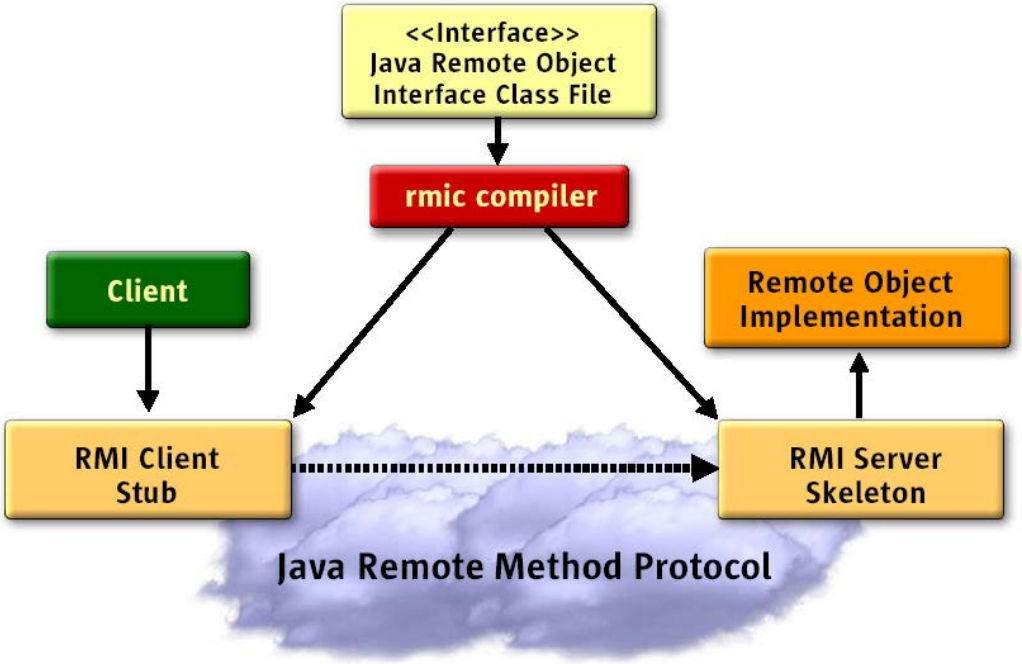
分布式计算技术

- RPC(Remote Procedure Call)
  - 第一个得到广泛应用的通信中间件技术
  - 调用远端过程并将结果返回。通信一般采用同步方式(Request-Wait-Reply)。



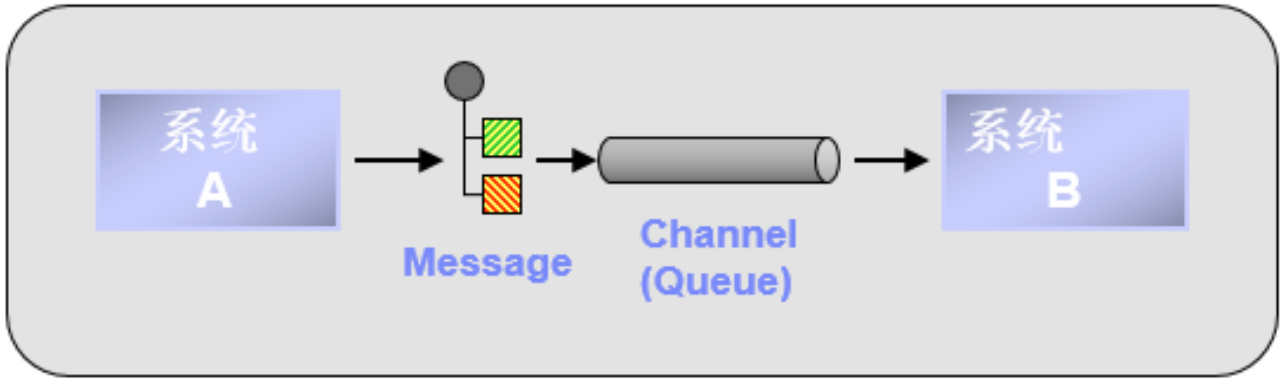
分布式计算技术

- RMI简介
  - 远程方法调用(Remote Method Invoke)，是面向对象版本的RPC
  - RMI使用两种特殊类型的对象：*存根（stub）*和*框架（skeleton）*



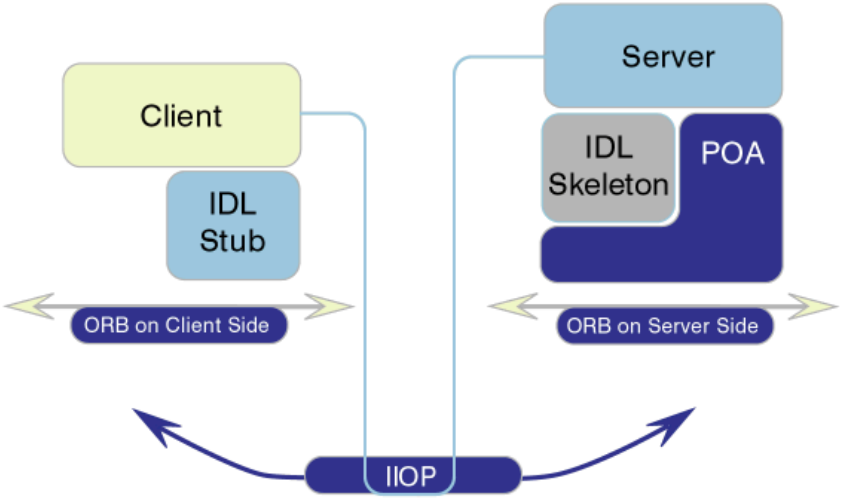
分布式计算技术

- 面向消息中间件
  - *面向消息的中间件（**MOM**）* 提供了一种以松散组织的、灵活的风格来集成应用程序的机制
  - **MOM**产品：IBM MQSeries， BEA WebLogic JMS Server，Oracle AQ，Tibco， SwiftMQ， AcitveMQ



分布式计算技术

- CORBA
  - CORBA(Common Object Request Broker Architecture)支持异构（heterogeneous）对象的连接与互操作性。
  - ORB(Object Request Broker)对象请求代理:它作为一个“软件总线”来连接网络上的不同对象
  - IIOP(Internet Inter-ORB Protocol)因特网ORB之间的协议
  - IDL接口描述语言（Interface Description Language）使得所有CORBA对象以一种方式被描述

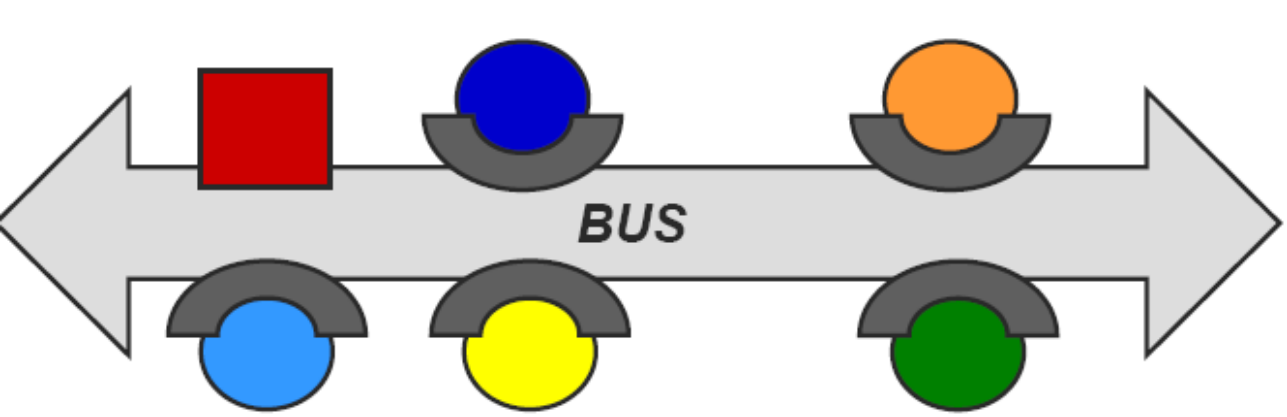


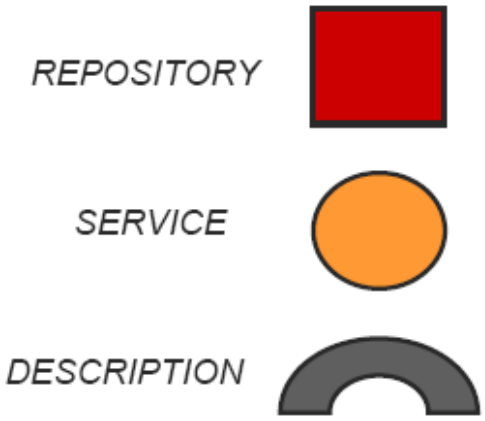
web services概述

- 什么是Web Services
  - Web Services采用Web技术和架构，提供一个松散耦合的分布式计算环境
  - Web Services是一系列标准的集合
    - SOAP
    - WSDL
    - UDDI
    - WSFL
    - …

SOA（面向服务的架构）

- W3C : A set of components which can be invoked, and whose interface descriptions can be published and discovered.





什么是面向服务的架构（SOA）

SOA是Gartner于1996年提出的架构思想

面向服务的架构（Service-Oriented Architecture，SOA）是一种设计方式，它指导着业务服务（business services）在其生命周期（从构思开始，直至停止使用）中包括创建和使用的方方面面。SOA 也是一种定义和提供 IT 基础设施（IT infrastructure）的方式，它允许不同应用相互交换数据、参与业务流程（business processes），无论它们各自背后使用的是何种操作系统或采用了何种编程语言。

IT基础设施（IT infrastructure）指机构中与IT相关的各种硬件、软件、服务及数据通信设施等的总和

web services概述

Web Services Definition by W3C

- A Web service is a software application
- identified by a URI,
- whose interfaces and binding are capable of

being defined, described and discovered by XML artifacts and

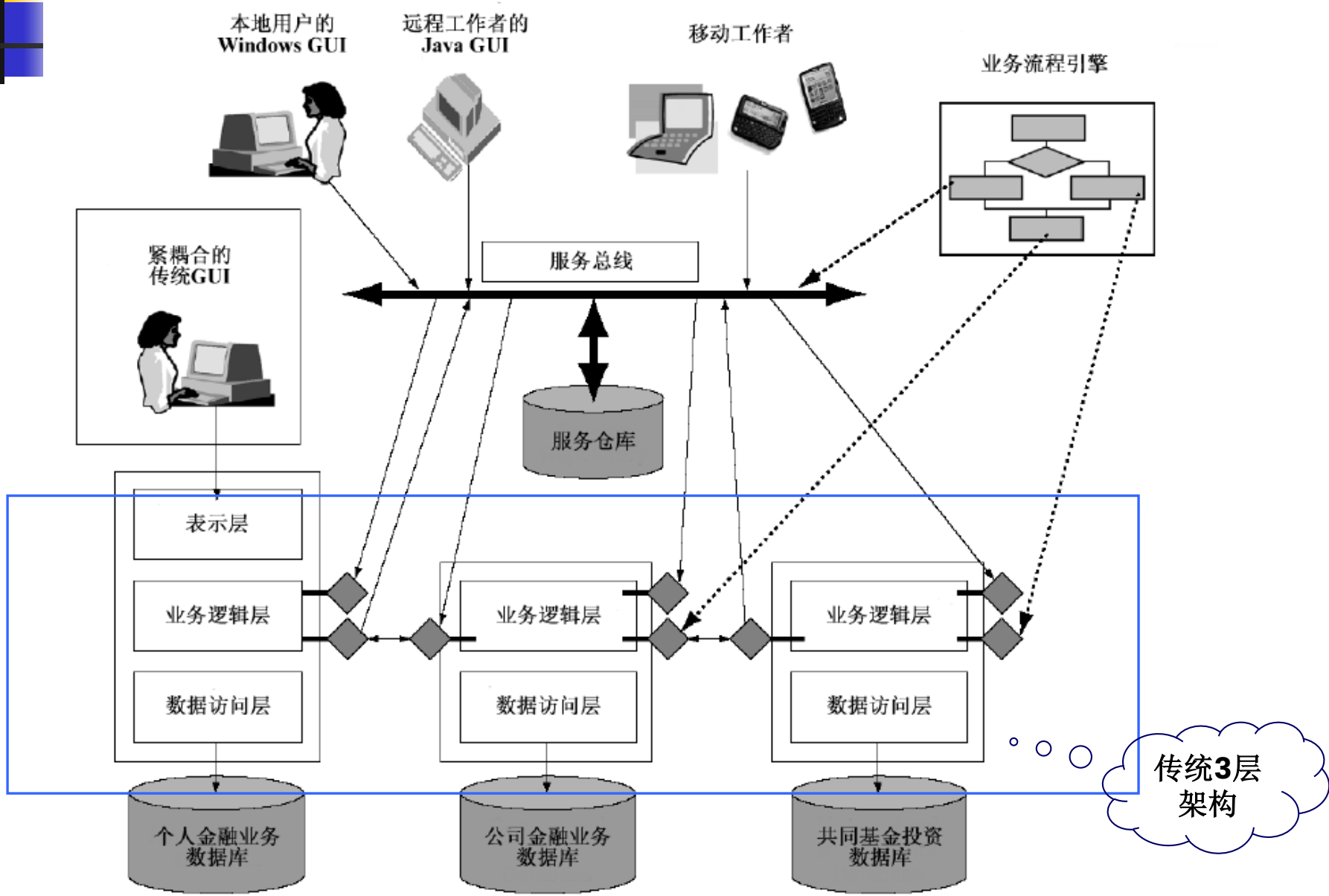
- supports direct interactions with other software applications
- using XML based messages
- via internet-based protocols

单个的Web 服务，就是一个支持机器之间通过网络进行交互的软件系统。Web 服务使用机器可以处理的格式（特别是WSDL）来描述自己的接口。其他系统则依据该接口描述，使用SOAP 报文格式与Web 服务通信。典型的SOAP 报文是一个利用HTTP 传输的XML 序列，在传输中通常与其他Web 相关的标准结合

Web服务介绍

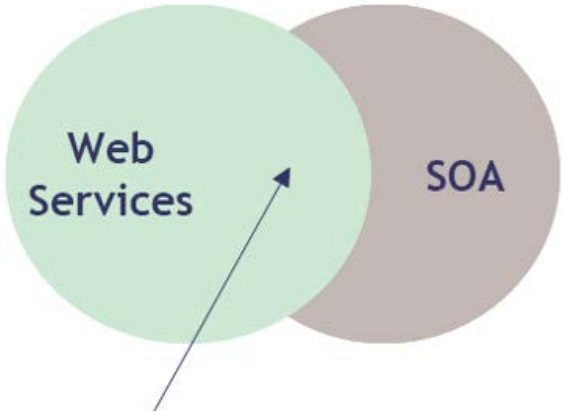
- Web服务介绍
  - Web服务技术代表了*分布式计算*的下一个阶段，将大大改变IT结构和伙伴关系。
  - Web服务包括一系列相关的标准，与DCOM和CORBA不同，Web服务使用了开放的标准，这意味着Web服务理论上可以使任何两个软件组件进行通信（即使它们所采用的编程语言或平台有所不同）
  - Web服务实现起来相对简单和便宜一些，因为它使用现有的基础架构（比如Web）来交换信息。显著降低企业应用集成（EAI）和B2B通信的成本
  - Web服务可以通过许多因特网协议交换信息，但大多是使用超文本传输协议（HTTP）

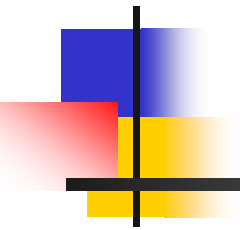
面向服务的集成示例


本地用户的
Windows GUI
远程工作者的
Java GUI
移动工作者
业务流程引擎
紧耦合的
传统GUI
服务总线
服务仓库
表示层
业务逻辑层
数据访问层
业务逻辑层
数据访问层
业务逻辑层
数据访问层
个人金融业务
数据库
公司金融业务
数据库
共同基金投资
数据库
传统3层
架构

Web 服务与SOA

- 不同
  - SOA 是一套面向服务架构的标准规范。Web 服务则是一套技术体系，可以用来建立应用解决方案，解决特定的消息通信和应用集成问题
  - 在实际的业务环境中，SOA 是一种信息规划理念和应用软件架构，Web 服务可以用来实现SOA
  - 学术界有人认为没有Web 服务，也可以很好地实现SOA。网格计算、多代理技术等也是面向服务技术，也可以实现SOA
- 相同
  - 在SOA 规范中，明确提出使用Web 服务技术实现SOA
  - Web 服务的架构与SOA 在应用实践上是等同、可互换的



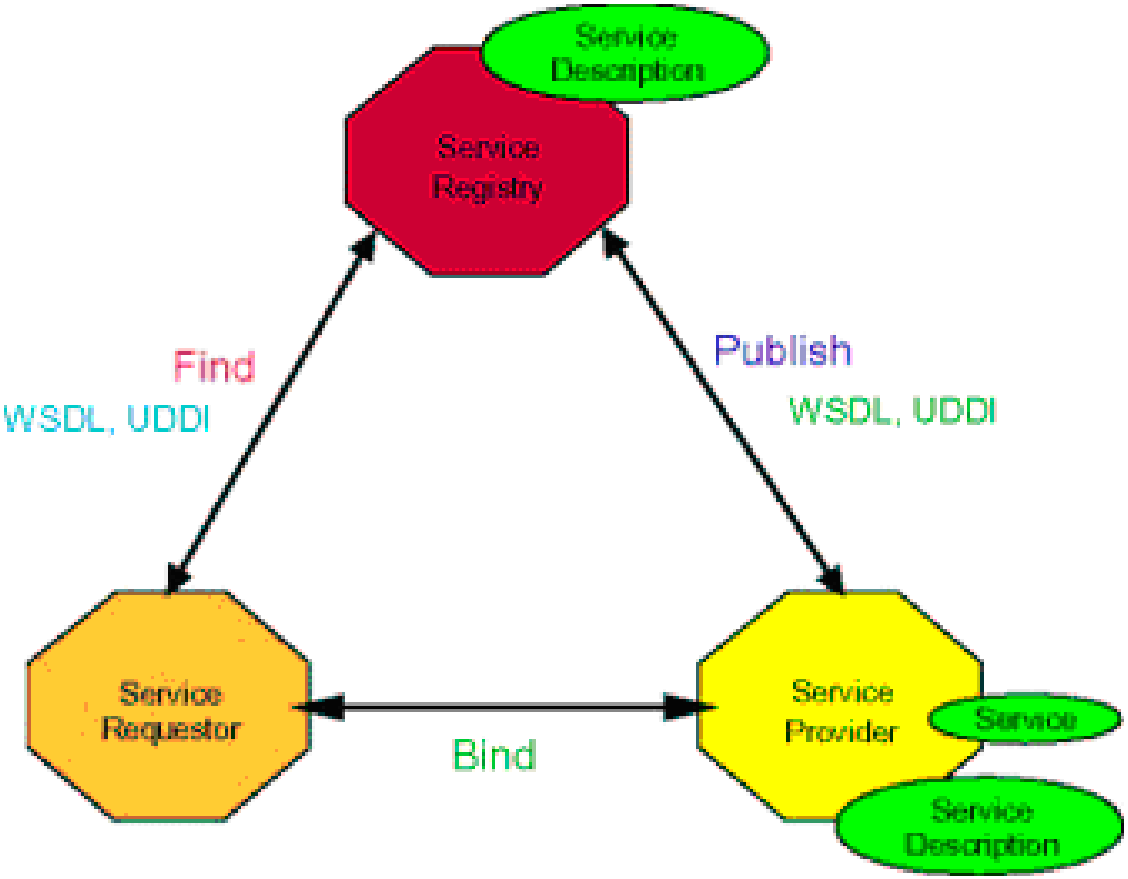


Web Service与相关技术

Web Services架构和协议

Web Service架构

- SOA架构



Web Service架构

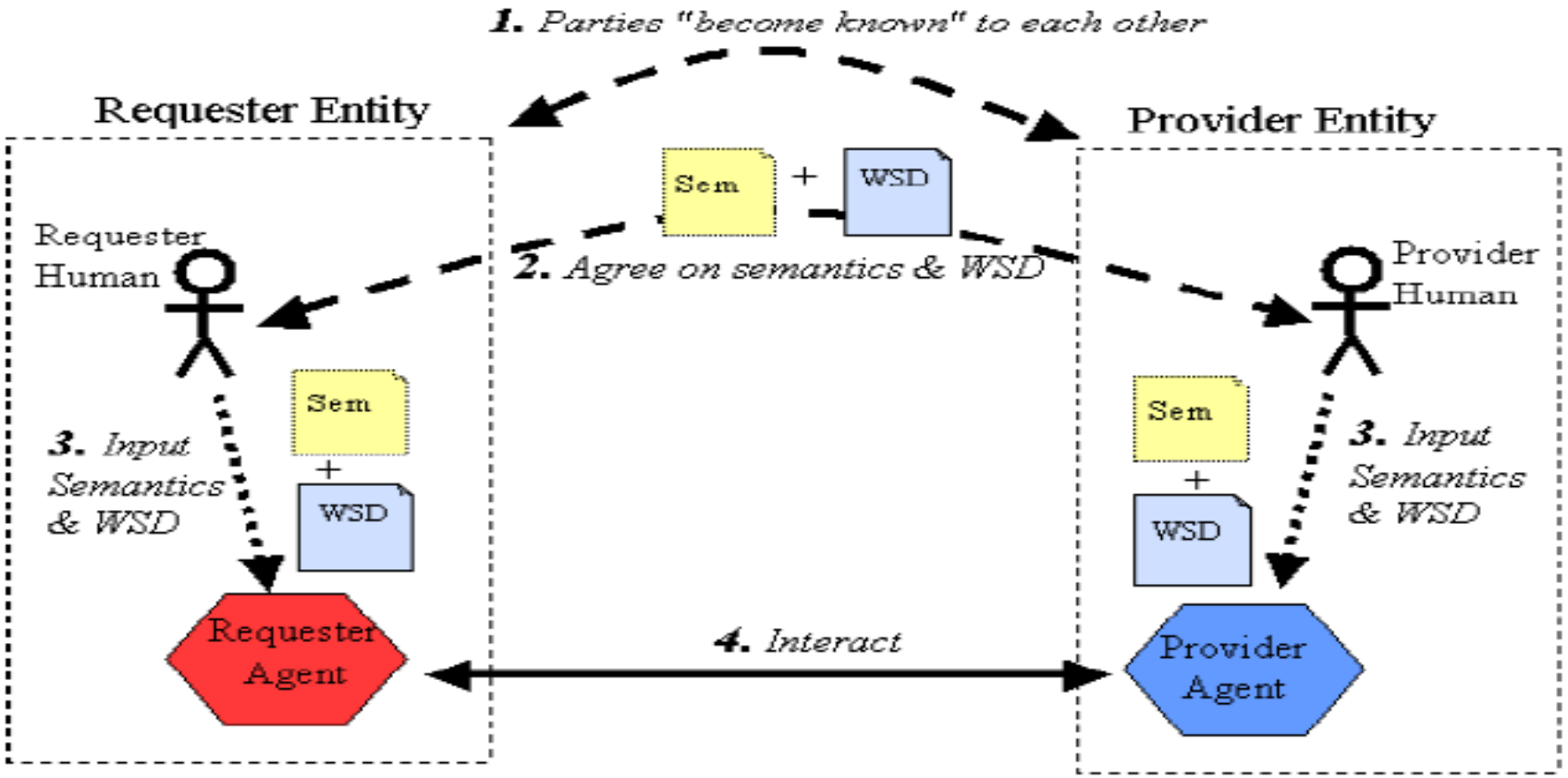


图 3.1. Web 服务的架构（SOA）[16]（2004.2.11 由 W3C 发布）

Web Service架构

- **Web 服务的构件**
  - 服务
    - 服务是一个软件模块，它部署在由服务提供者提供的可以通过网络访问的平台上。
    - 当服务的实现中利用到其它 的**Web** 服务时，它也可以作为请求者。
  - 服务描述
    - 要声明**Service provider** 的语义特征
    - 服务描述包含服务的接口和实现的细节。其中包括服务的数据类型、操作、绑定信息和网络位置。
    - 服务描述可以被发布给服务请求者或服务注册中心。
    - 服务描述和服务实现是分离的

Web Service架构

- 三个角色：
  - 服务提供者（**Service Provider**）
    - 发布自己的服务，并且对使用自身服务的请求进行响应
  - 服务请求者（**Service Requestor**）
    - 利用Service broker 查找所需的服务，然后使用该服务
  - 服务注册中心（**Service broker**）
    - 注册已经发布的Service provider，对其进行分类，并提供搜索服务

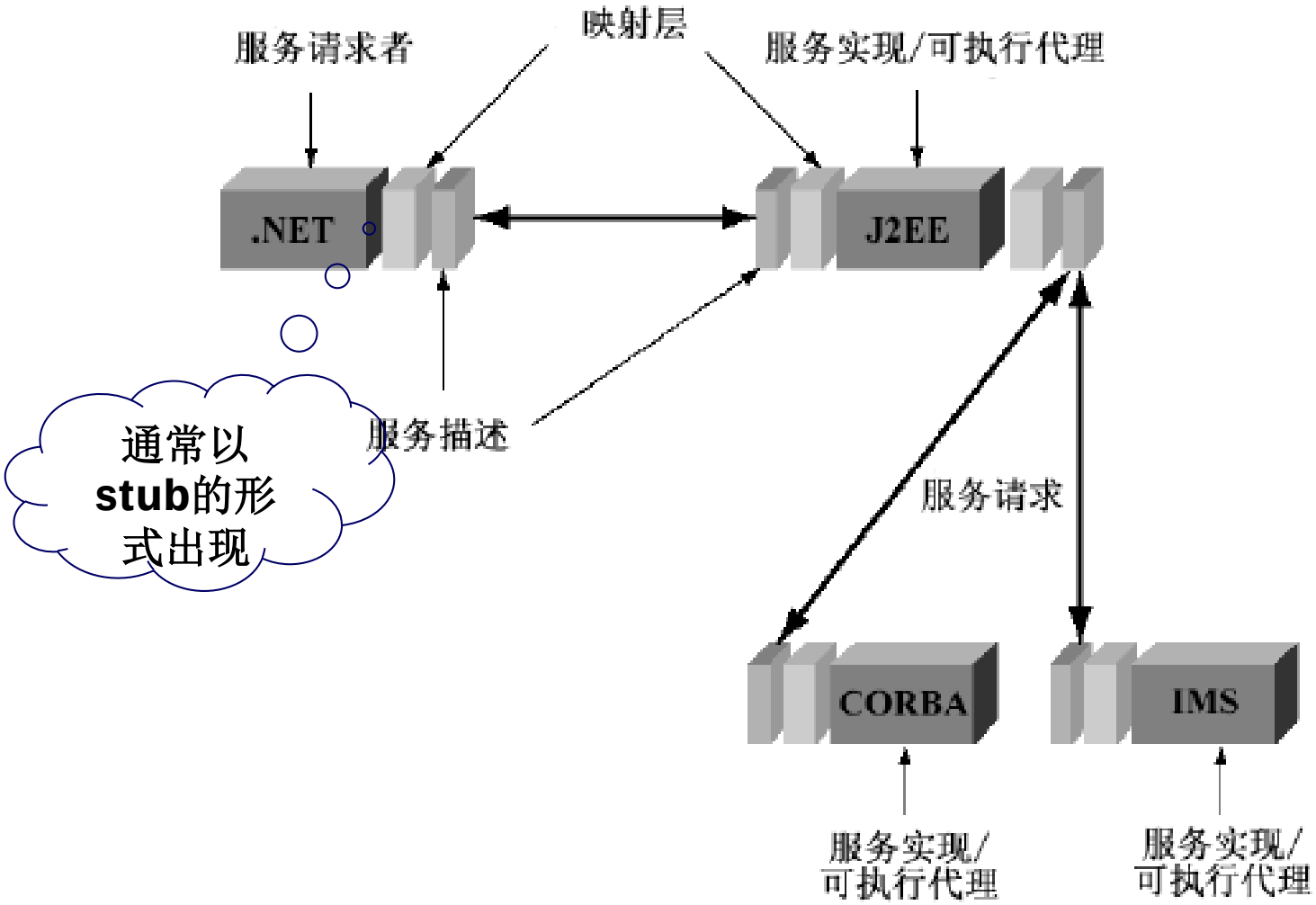
Web Service架构

- 三个基本操作
  - 发布（Publish）
    - 直接发布：服务提供者直接向服务请求者发送 WSDL 文档
    - 服务提供者还可以将描述服务的文档发布到主机本地 WSDL 注册中心、专用 UDDI 注册中心或 UDDI 运营商节点
  - 查找（Find）：对于服务请求者，可能会在两个不同的生命周期阶段中牵涉到查找操作：
    - 在设计时为了程序开发而检索服务的接口描述
    - 在运行时为了调用而检索服务的绑定和位置描述
  - 绑定/调用（Bind/Invoke）
    - 服务请求者使用服务描述中的绑定细节来定位、联系和调用服务，从而在运行时调用或启动与服务的交互

Web Service架构

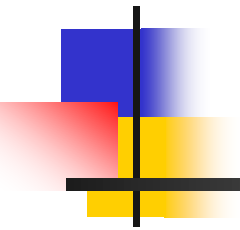
- 工作过程
  - 服务提供者:
    - 定义 **Web** 服务的服务描述,声明**Service provider** 的语义特征,并把它发布到服务请求者或服务注册中心。
  - 服务注册中心Service broker
    - 使用语义特征将**Serviceprovider** 进行分类，以帮助具体服务的查找。
  - 服务请求者
    - 根据语义特征来匹配那些满足要求的**Service provider**。
  - 服务请求者
    - 使用服务描述与服务提供者进行绑定并调用 **Web** 服务实现或同它交互。

Web Service架构



Web Service架构

- **Web 服务开发生命周期**
  - 构建
    - 开发和测试 **Web** 服务实现、定义服务接口描述和定义服务实现描述。
  - 部署
    - 部署阶段包括向服务请求者或服务注册中心发布服务接口和服务实现的定义，以及把 **Web** 服务的可执行文件部署到执行环境中。
  - 运行
    - 在运行阶段，可以调用 **Web** 服务。在此，**Web** 服务完全部署、可操作并且服务提供者可以通过网络访问服务。现在服务请求者可以进行查找和绑定操作。
  - 管理
    - 管理阶段包括持续的管理和经营 **Web** 服务应用程序。解决安全性、可用性、性能、服务质量和业务流程问题。



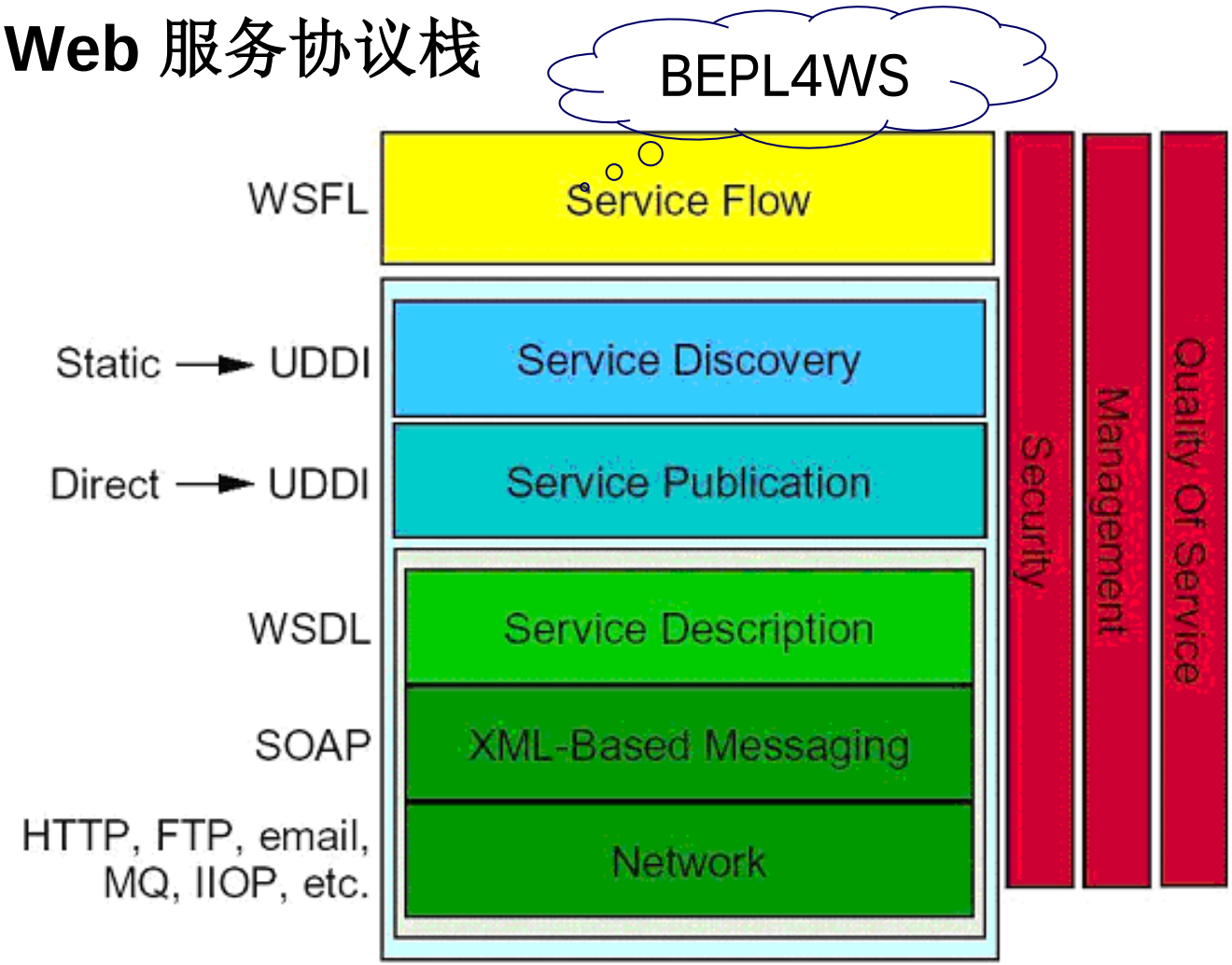
SOAP Web服务

Web Service架构

- **Web 服务协议栈**
  - **WSDL(Web Service Description Language)**
    - 用来描述服务
  - **UDDI(Universal Description, Discovery and Integration)**
    - 用来发布、查找服务
  - **SOAP(Simple Object Access Protocol)**
    - 用来执行服务调用
  - **WSFL(Web Service Flow Language)**
    - 将分散的、功能单一的Web服务组织成一个复杂的有机应用

Web Service架构

- **Web 服务协议栈**

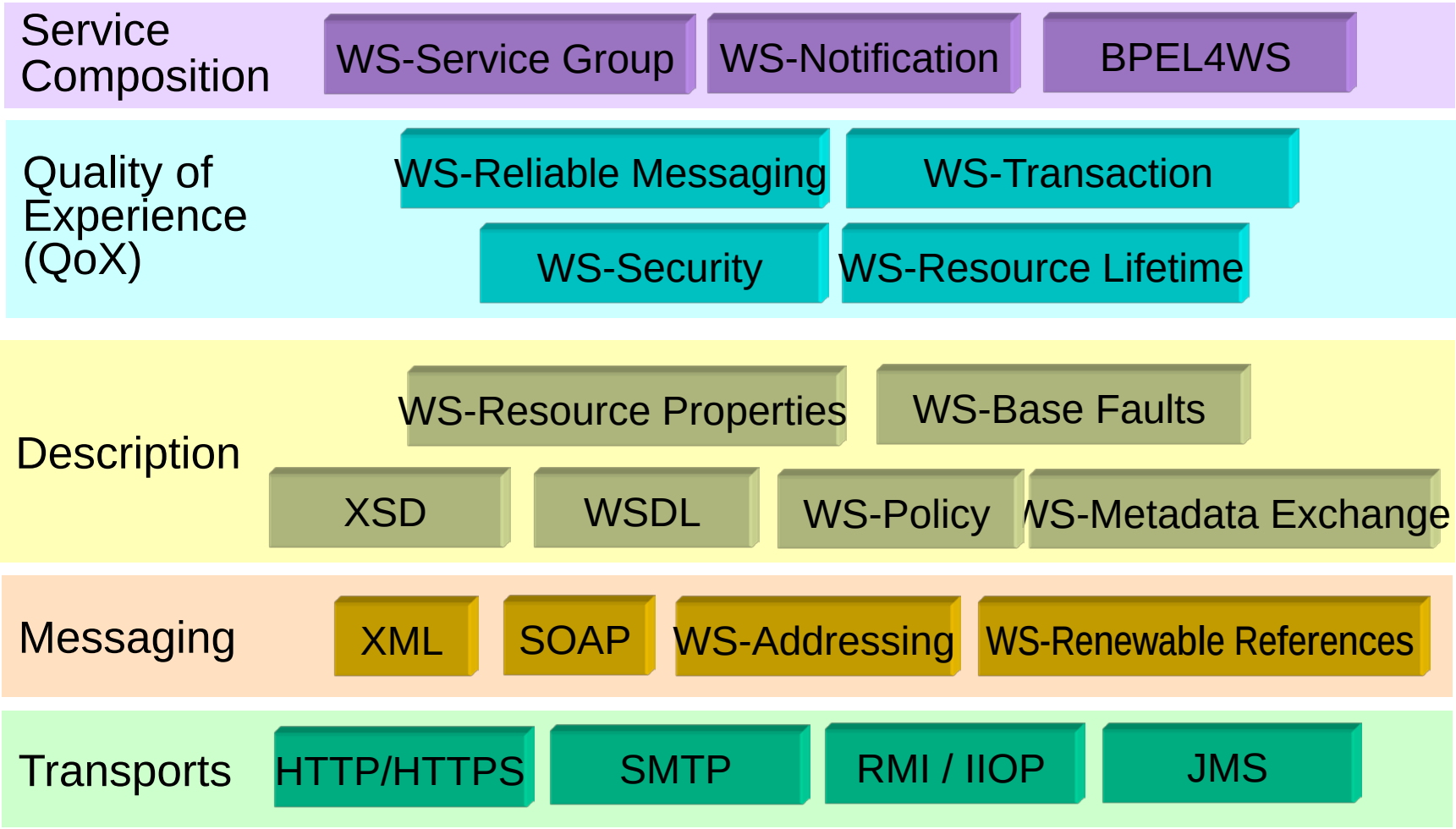


Web Services 相关协议

Source: http://roadmap.cbdiforum.com/reports/protocols/

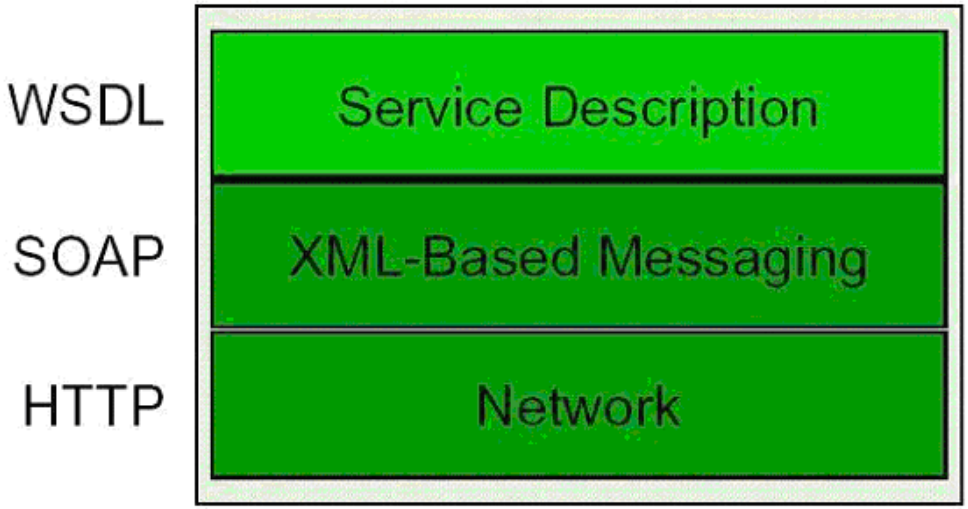
复旦大学课程《高级Web技术》 版权@ Dai Kaiyu

Web Services 相关协议



Web Service架构

- 可互操作的基础协议栈
  - 简单的协议栈将包括网络层的 **HTTP**、**XML** 消息传递层的 **SOAP** 协议以及服务描述层的 **WSDL**。
  - 所有企业间或公用 **Web** 服务都应该支持这种可互操作的基础协议栈



SOAP简介

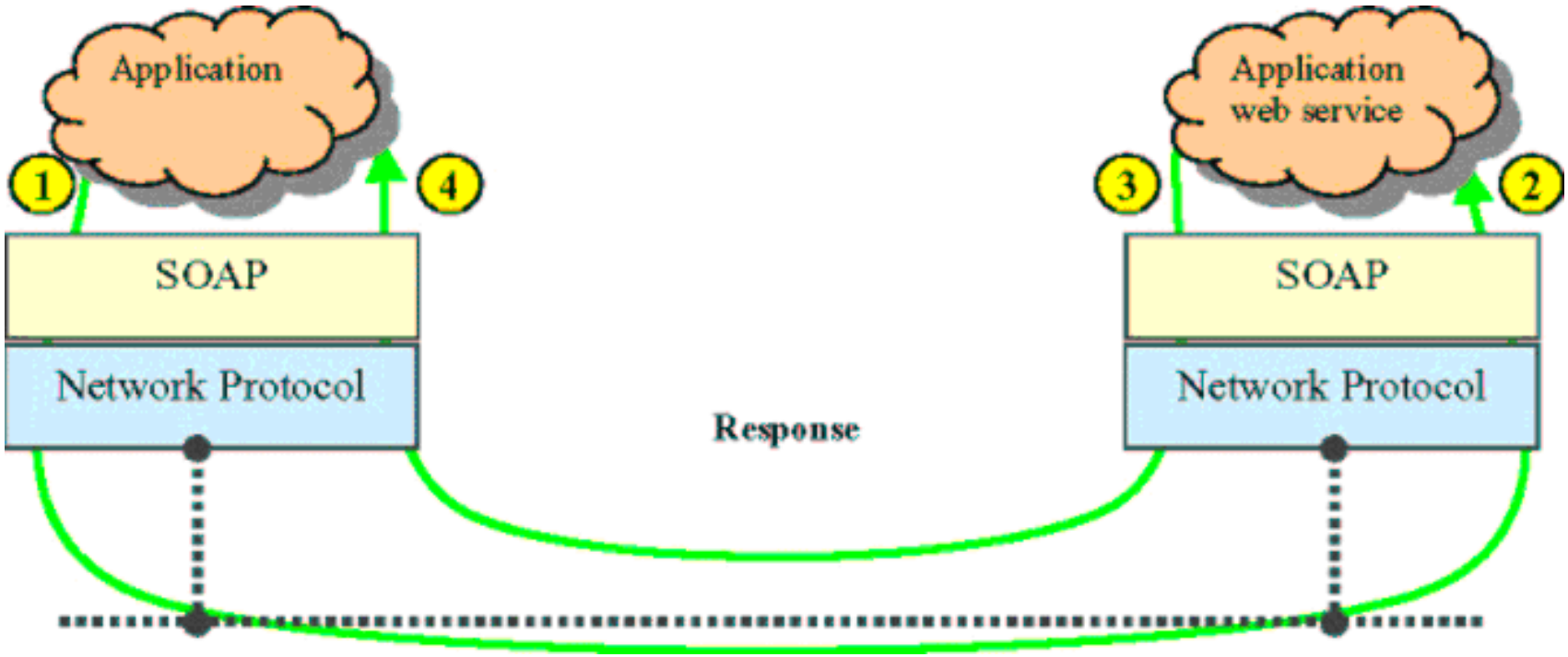
- 简单对象访问协议 **SOAP**（Simple Object Access Protocol）是在松散的、分布的环境中使用**XML**交换结构化的和类型化的信息的一种简单协议。
- **XML**的标准化为跨平台之间的交流带来了极大的方便，而**SOAP**的实质就是把这种交流的方式标准化了
- **SOAP**本身并不定义任何应用语义，如编程模型或特定语义实现，它只定义了一种简单的以模块化的方式包装数据的机制。
  - 将数据/对象打包成**XML**格式的数据

```
public class Person{
    String name;
    int age;
    //方法的定义...
}
```

```
<Person>
    <name>zhang3</name>
    <age>20</age>
</Person>
```

使用 SOAP 的 XML 消息传递

- 使用 SOAP 的 XML 消息传递



一个简单的SOAP请求/响应

```
public interface Hello
{
    public String sayHelloTo(String name);
}
```

- 假设有上述接口定义的服务存在
  - 客户端在远程调用**sayHelloTo**方法时，提供一个名字，期望返回一个字符串
  - 假设**Java RMI**、**CORBA**、**DCOM**都不存在，开发者必须负责将方法调用串行化，并把消息发给远程服务器。
  - 一个简单的方式是使用**XML**

一个简单的SOAP请求/响应

```
<?xml version="1.0"?>
<Hello>
   <sayHelloTo>
        <name>John</name>
   </sayHelloTo>
</Hello>
```

- 用XML打包请求
  - 将接口名作为根结点
  - 方法和参数作为结点
- 将请求发送给服务器
  - 不创建自己的TCP/IP消息，利用HTTP
  - 将请求封装成HTTP POST请求格式发出

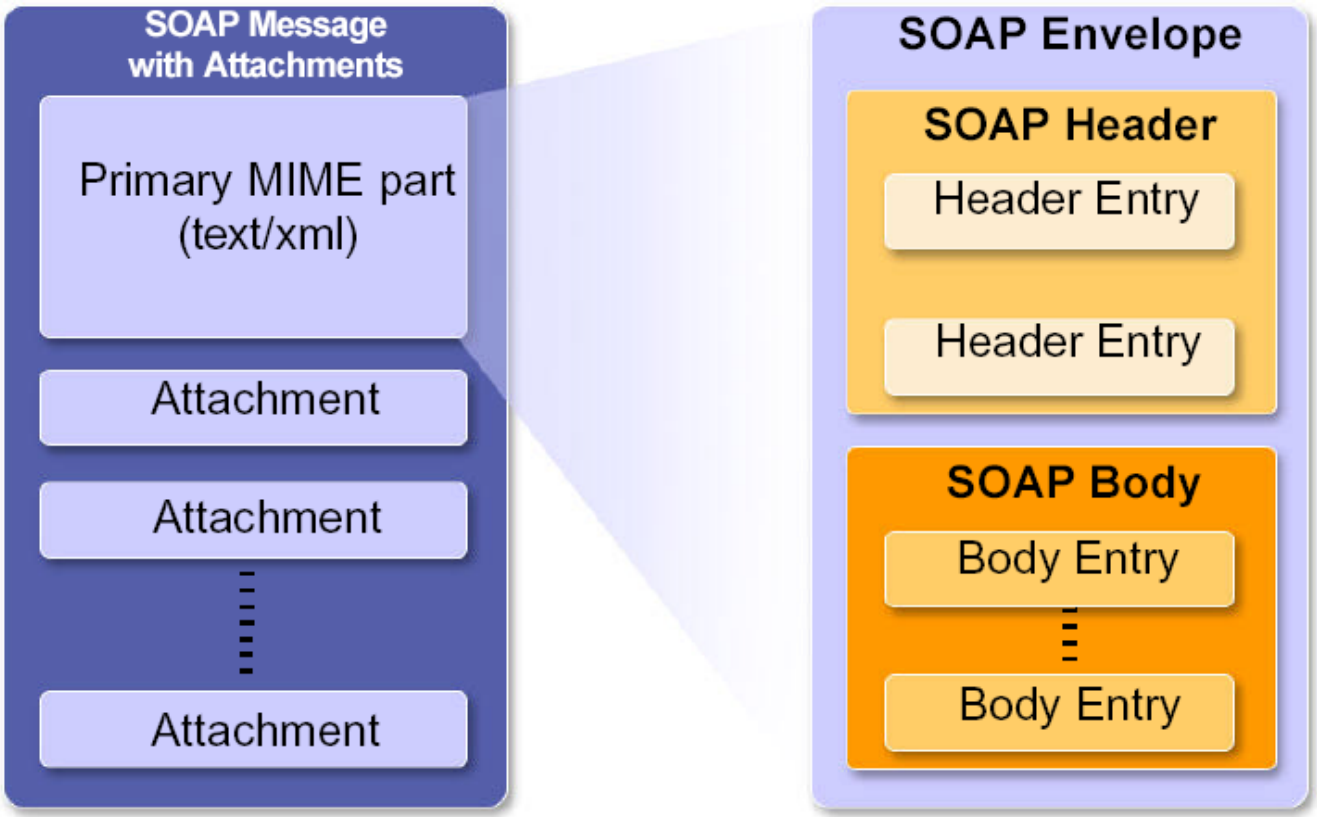
一个简单的SOAP请求/响应

```
<?xml version="1.0"?>
<Hello>
   <sayHelloToResponse>
        <message>Hello John, How are you?</message>
   </sayHelloToResponse>
</Hello>
```

- 服务器接收到请求，解码XML，处理请求，然后以XML格式向客户发送响应
  - 与请求比较，方法的结点名字变为请求的方法名后缀Response
  - 客户程序知道自己调用了哪个方法，根据方法名后缀Response寻找调用方法的返回值

SOAP消息

- 具有附件的SOAP消息结构



SOAP消息

- **SOAP Header**
  - 用于扩充
    - Authentication
    - Transaction
    - etc.
  - 属性
    - *mustUnderstand*
    - *Actor*
      - **指定SOAP头部块的目标处理者**
      - **使用URI 形式，如: http://www.w3.org/2001/06/soapenvelope/actor/next**
      - **缺省表示“最终SOAP接受者 "**

SOAP消息

- **SOAP Header**
  - 一个SOAP Header例子

```
<env:Header
    xmlns:env="http://www.w3.org/2001/06/soap-envelope">
    <t:Transaction
      xmlns:t="http://example.org/2001/06/tx"
      env:mustUnderstand="1">
       5
    </t:Transaction>
</env:Header>
```

当使用的是FTP这种不提供发送方信息的协议时，在信息头中就可以提供返回路径

SOAP消息

- **SOAP Body**
  - 由 Body blocks组成
  - Body blocks 可以是
    - Application data
    - RPC method and parameters
    - SOAP fault

SOAP消息

- SOAP Body
  - SOAP Fault
    - 用于携带error and/or status 信息
    - 四个子元素
      - faultcode
      - faultstring：*给人读的错误描述信息*
      - faultactor
      - detail
    - 预定义SOAP faultcode 值
      - VersionMismatch：Invalid namespace in SOAP envelope
      - MustUnderstand： Receiver mode cannot handle *mustUnderstand* SOAP header block
      - Client： Indicates client side error
      - Server：Indicates server side error

SOAP请求实例

```
<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

SOAP响应实例

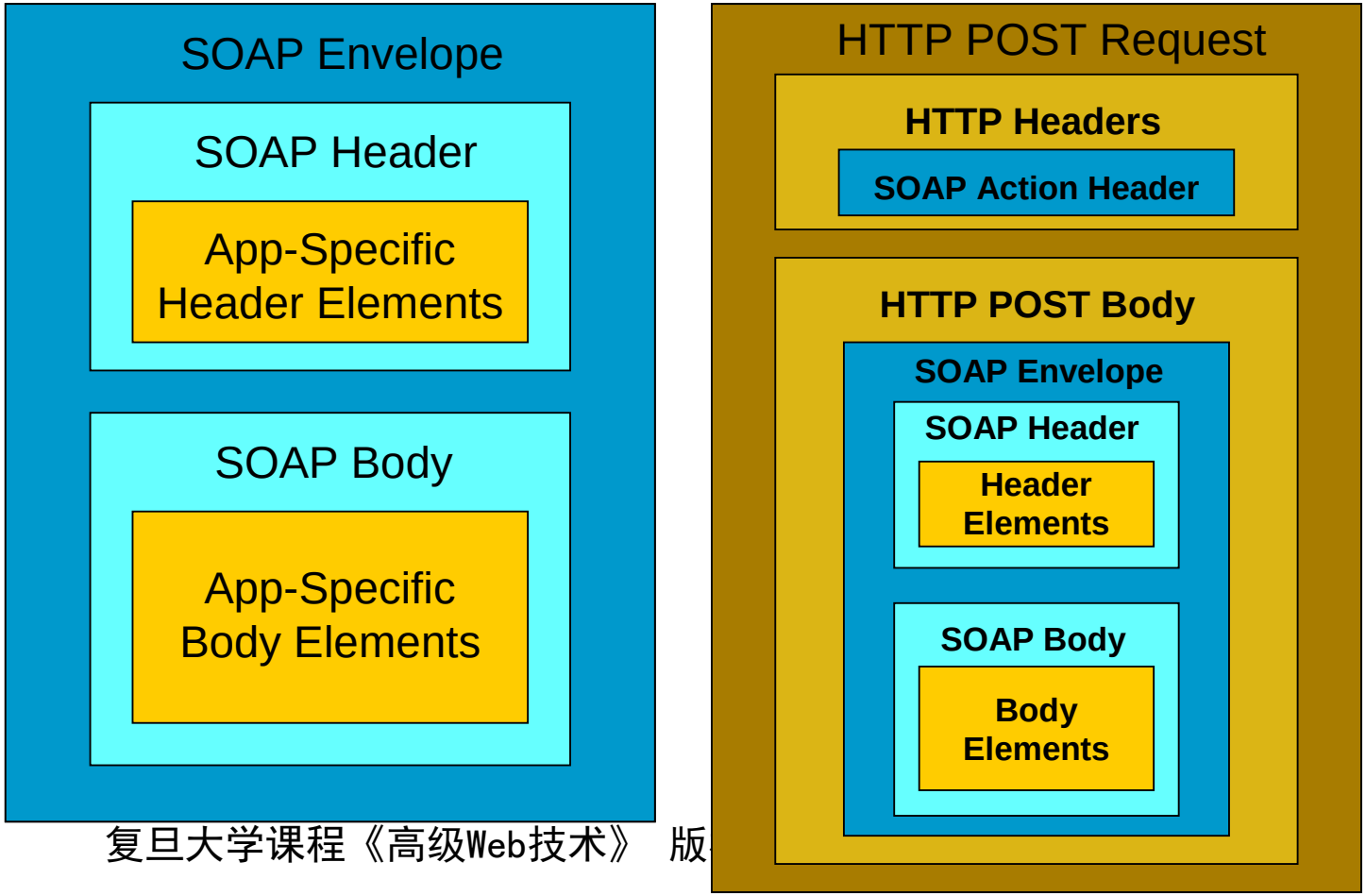
```
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="..."
    SOAP-ENV:encodingStyle="...">
  <SOAP-ENV:Header>
     <!-- Optional context information -->
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
   <m:GetLastTradePriceResponse xmlns:m="some_URI">
       <price>100.5</price>
   </m:GetLastTradePriceResponse>
  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

SOAP调用错误

```
<SOAP-ENV:Envelope xmlns:SOAP-
    ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Body>
        <SOAP-ENV:Fault>
            <faultcode>SOAP-ENV:Server</faultcode>
            <faultstring>Server Error</faultstring>
            <detail>
                <e:myfaultdetails xmlns:e="Hello">
                    <message>
                        Sorry, my silly constraint says
                        that I cannot say hello on Tuesday.
                    </message>
                    <errorcode>1001</errorcode>
                </e:myfaultdetails>
            </detail>
        </SOAP-ENV:Fault>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

绑定到HTTP的SOAP

- 将SOAP绑定到HTTP上可以利用HTTP丰富的特性
  - SOAP很自然的利用HTTP的请求/响应机制



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>
```

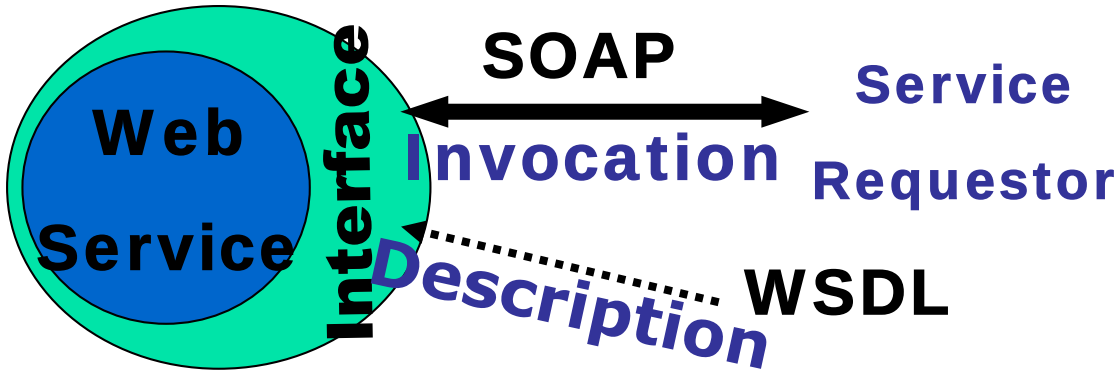
绑定到HTTP的SOAP响应

```
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>
```

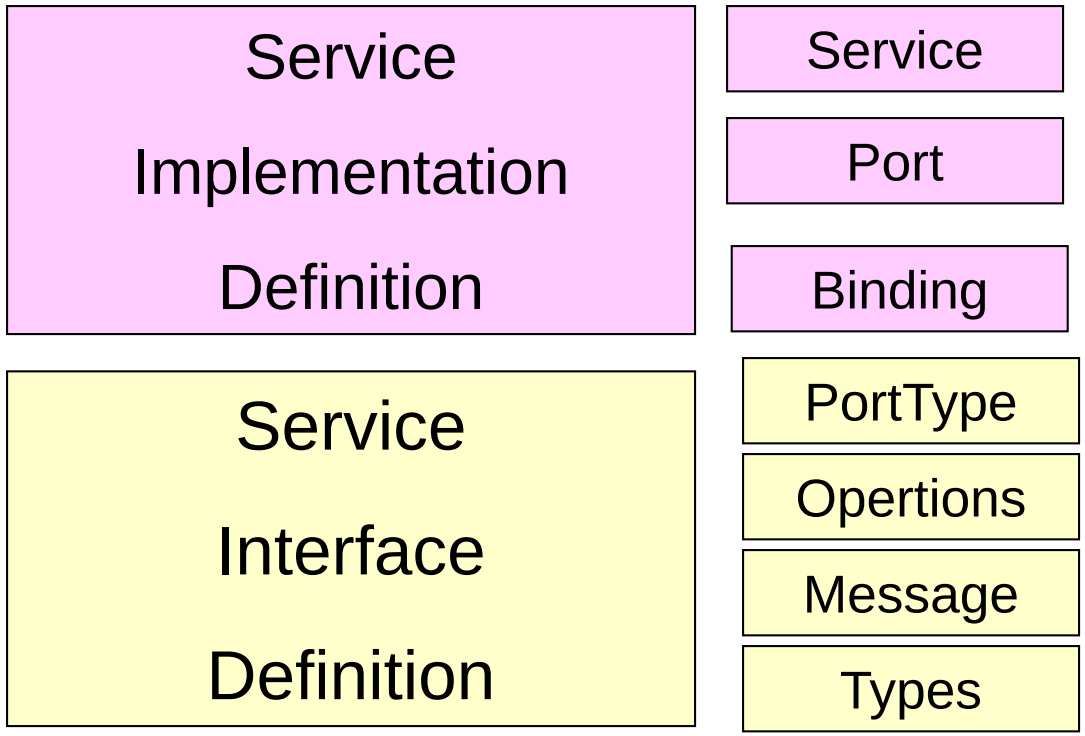
什么是WSDL

- 如果将Web Service作为一个分布式对象来看，WSDL就是Web Service的接口描述语言（IDL）。
- WSDL定义了一套基于XML的语法，将Web Service描述为能够进行消息交换的服务访问点的集合。
- WSDL所处的地位



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。



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>
```

WSDL

- 基本服务描述
  - 服务接口定义
    - 是一种抽象的服务定义，它可以被多个服务实现定义实例化和引用
    - 与具体的**Web Services**部署细节无关，是可以复用的描述
    - 服务接口包含 **WSDL** 元素，它们组成了服务描述中的可重用部分
      - **types**：
      - **message**：
      - **operations**
      - **portType**：

WSDL

- 基本服务描述
  - 服务接口定义
    - **types**：
      - 是一个数据类型定义的容器，包含了所有在消息定义中需要的**XML**元素的类型定义
      - 采用**XML Schema**作为数据的规范定义
      - 数据类型将映射到相应语言的相应类型,如**xsd:anyType**元素一般映射为**java.lang.object**类.

```
<complexType name="SOAPStruct">
  <all>
    <element name="varString" type="string"/>
    <element name="varInt" type="int"/>
    <element name="varFloat" type="float"/>
  </all>
</complexType>
```

复

WSDL

- 基本服务描述
  - 服务接口定义
    - **message**：
      - 具体定义了在通信中使用的消息的数据结构
      - 一个消息包含一个或者多个**<part>**子元素， **<part>**子元素标识独立的数据块和数据块所属的数据类型.

```
<message name="EncodeValueRequest">
   <part name="Value" type="xs:int"/>
</message>
<message name="EncodeValueResponse">
   <part name="return" type="xs:double"/>
</message>
<message name="DecodeValueRequest">
   <part name="Value" type="xs:double"/>
</message>
<message name="DecodeValueResponse">
   <part name="return" type="xs:int"/>
</message>
```

WSDL

- 基本服务描述
  - 服务接口定义
    - **operations**
      - 抽象定义了 **Web** 服务的操作。
      - 操作定义了输入和输出数据流中可以出现的 **XML** 消息
      - 类似于**java**的方法定义
    - **portType**：
      - 指定**Web**服务端点所支持的活动的子集。为一系列**operations**的集合
      - 为一组能在单个端点上执行的活动提供了一个唯一的标志符

WSDL

- 基本服务描述
  - 服务接口定义
    - portType实例

```
<portType name="IEncodeDecode">
   <operation name="EncodeValue">
      <input message="EncodeValueRequest"/>
      <output message="EncodeValueResponse"/>
   </operation>
   <operation name="DecodeValue">
      <input message="DecodeValueRequest"/>
      <output message="DecodeValueResponse"/>
   </operation>
</portType>
```

WSDL

- 基本服务描述
  - 服务实现定义
    - 描述给定服务提供者如何实现特定服务接口的 **WSDL** 文档
    - 描述的内容与具体服务的部署相关了
    - **Web** 服务被建模成 **service** 元素。
      - **binding**：
      - **Service:**
      - **Port:**

WSDL

- 基本服务描述
  - 服务实现定义
    - **binding**：
      - 定义了某个**PortType**与某一种具体的网络传输协议或消息传输协议相绑定
      - 多种抽象消息交换模式与具体的传输协议和消息交换格式的绑定方式：与**SOAP**的绑定、与**HTTP GET/POST**的绑定以及与**MIME**的绑定

WSDL

- Binding的一个实例：

某个portType元素的名字

```
<binding name="IEncodeDecodebinding" type="IEncodeDecode">
  <operation name="EncodeValue">
    <input>
      <soap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="urn:EncodeDecode-IEncodeDecode"/>
    </input>
    <output>
      <soap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="urn:EncodeDecode-IEncodeDecode"/>
    </output>
  </operation>
  <operation name="DecodeValue">
    <input>
      <soap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="urn:EncodeDecode-IEncodeDecode"/>
    </input>
    <output>
      <soap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="urn:EncodeDecode-IEncodeDecode"/>
    </output>
  </operation>
</binding>
```

WSDL

- 基本服务描述
  - 服务实现定义
    - **Service:**
      - 包含一组相关**port** 元素。
    - **Port:**
      - 描述的是一个服务访问入口的部署细节
      - 将端点（例如网址位置或 **URL**）与**binding** 元素关联起来。
      - 给出 **HTTP**协议的**URL,** 或者**SMTP** 的**email address**

WSDL

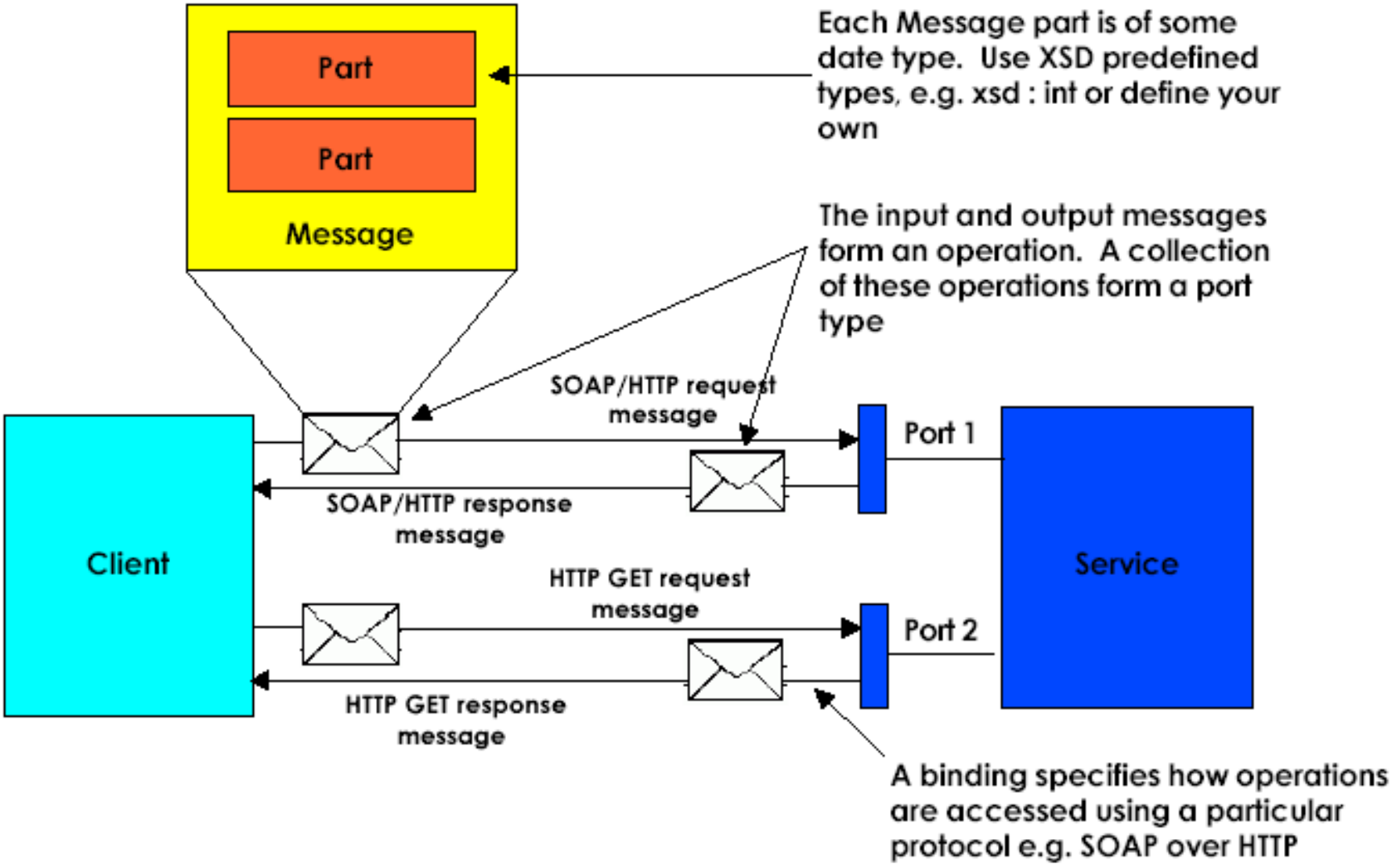
- 基本服务描述
  - 服务实现定义
    - **Service的一个实例**

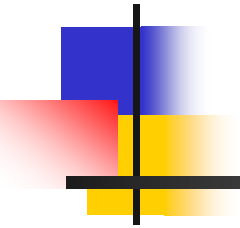
```
<service name="IEncodeDecodeservice">
  <port name="IEncodeDecodePort" binding="IEncodeDecodebinding">
    <address
location="http://localhost:1024/WSServer.WadSoapDemo/soap/IEncodeDecode"/>
  </port>
</service>
```

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>
```

Web Service Invocation





REST化Web服务

面向资源的架构（ROA）

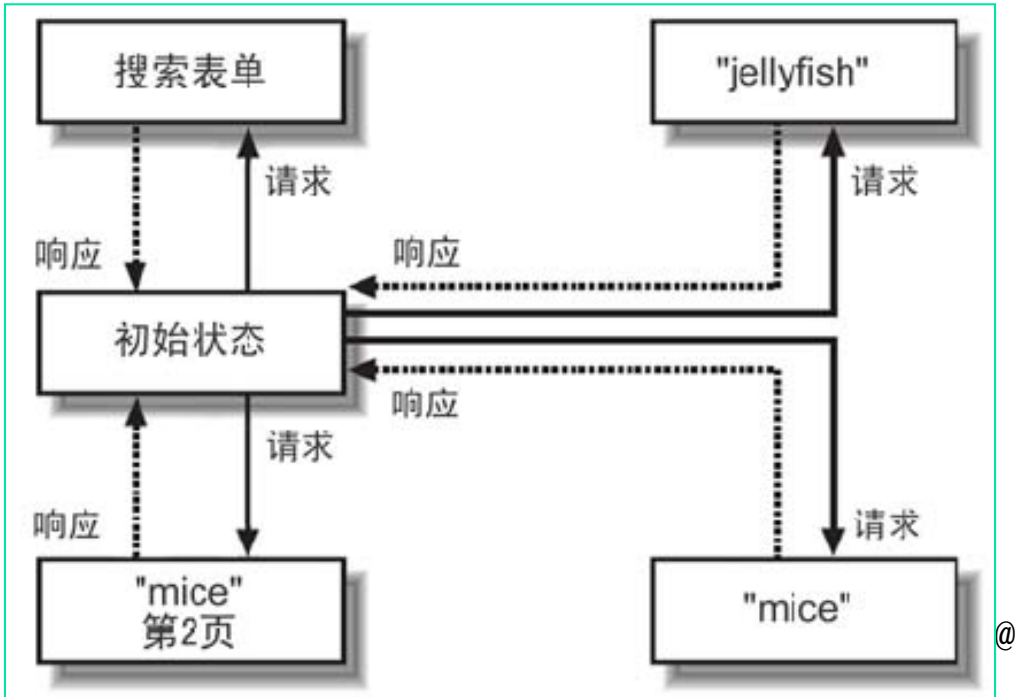
- 资源
  - 任何事务，只要有被引用的必要，就称为资源。
  - 通常为某个存放在计算机上并体现为比特流的东西。
  - 在web上，资源至少有一个URI。
    - 两个资源不可能是同一个
    - 两个不同的资源在某个时期指向同样的数据
    - 一个资源可以有一个或者多个URI（其中一个是规范资源）
    - 一个URI只能标识一个资源。

面向资源的架构（ROA）

- 特征
  - 可寻址性（addressability）
    - 一个应用将其数据集中有价值的部分作为资源通过URI暴露出来
    - 否则，你将要用语言来描述操作，或者保存某网页并发给别人，如http://www.google.com/search?q=rest这个“有关rest的资源列表”
    - URI应该具有描述性
    - 支持了缓存和书签
    - Amazon s3中，每个桶（bucket）和对象（object）都有URI
    - Ajax不是可寻址的

面向资源的架构（ROA）

- 特征
  - 无状态性 （**statelessness**）
    - 每个**HTTP**请求都是独立的，包含了服务器操作所需要的所有信息
    - 一个无状态搜索引擎



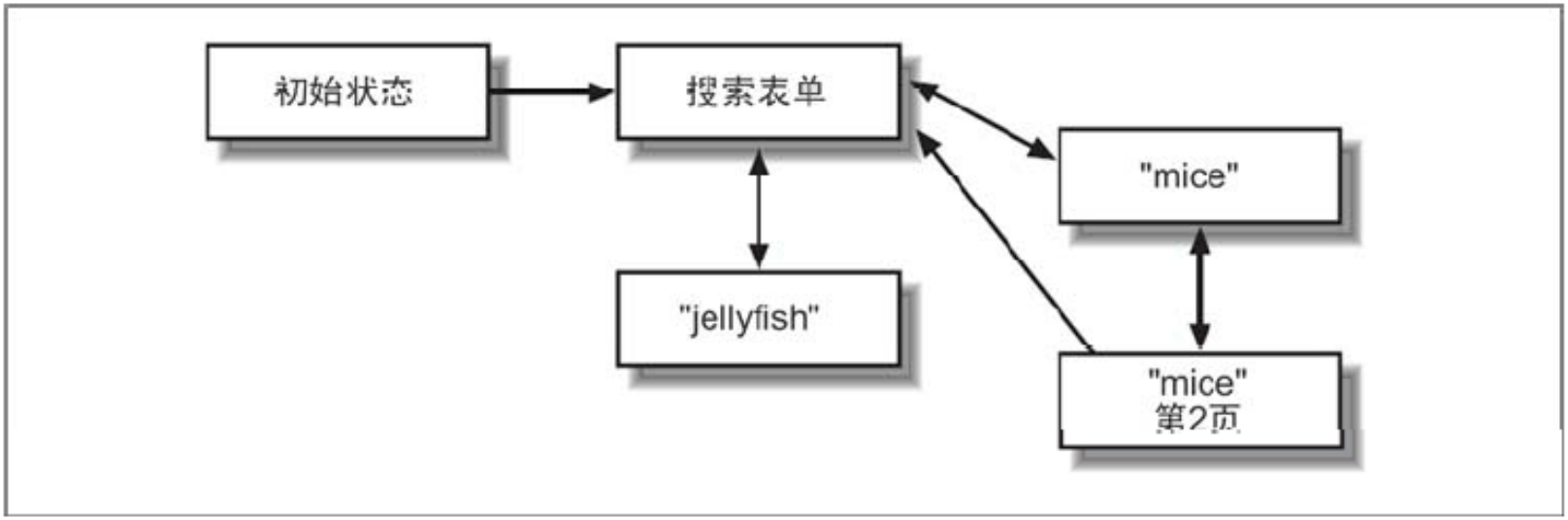
每次搜索都回到初始状态重新开始

http://www.google.com/search?q=jellyfish&start=10

@ Dai Kaiyu

面向资源的架构（ROA）

- 特征
  - 无状态性 （**statelessness**）
    - 一个有状态的搜索引擎示例
    - 每次可以基于前面一次搜索的状态，可以少发送一些信息，但是？



面向资源的架构（ROA）

- 特征
  - 无状态性 的优势
  - 对于服务器端
    - 大大减少出错条件
    - 负载均衡变得容易
    - 容易作缓存处理
  - 对于客户端
    - 不需要重复前面的状态操作可以在新的会话中直接进入某个状态
    - **REST**要求把状态保存在客户端，并且发给服务器的每个请求中都包含这些状态。

REST架构

- **REST的概念**
  - **表述性状态转移（REST）风格是对分布式超媒体系统中的架构元素的一种抽象。**

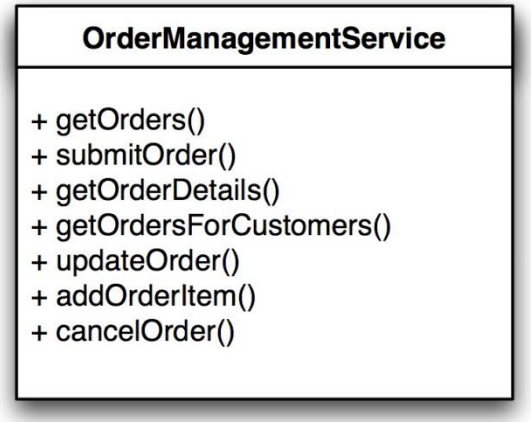
这个名称“表述性状态转移”是有意唤起人们对于一个良好设计的Web应用如何运转的印象：一个由网页组成的网络（一个虚拟状态机），用户通过选择链接（状态转移）在应用中前进，导致下一个页面（代表应用的下一个状态）被转移给用户，并且呈现给他们，以便他们来使用。

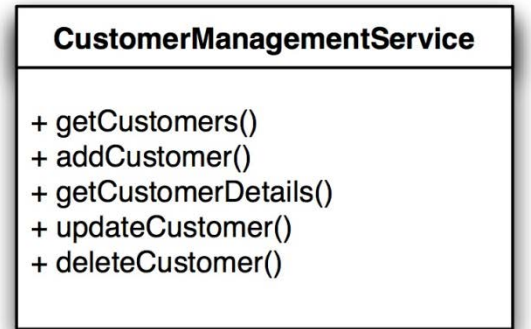
REST关键原则

- REST关键原则
  - 为所有“事物”定义ID
    - 在web中，URI
  - 将所有事物链接在一起
    - 超媒体被当作应用状态引擎
  - 使用标准方法
    - 使用Http操作
    - GET方法具有幂等性[指多个相同请求返回相同的结果]
    - 所有理解HTTP应用协议的组件能与你的应用交互
  - 资源多重表述
  - 无状态通信

REST关键原则

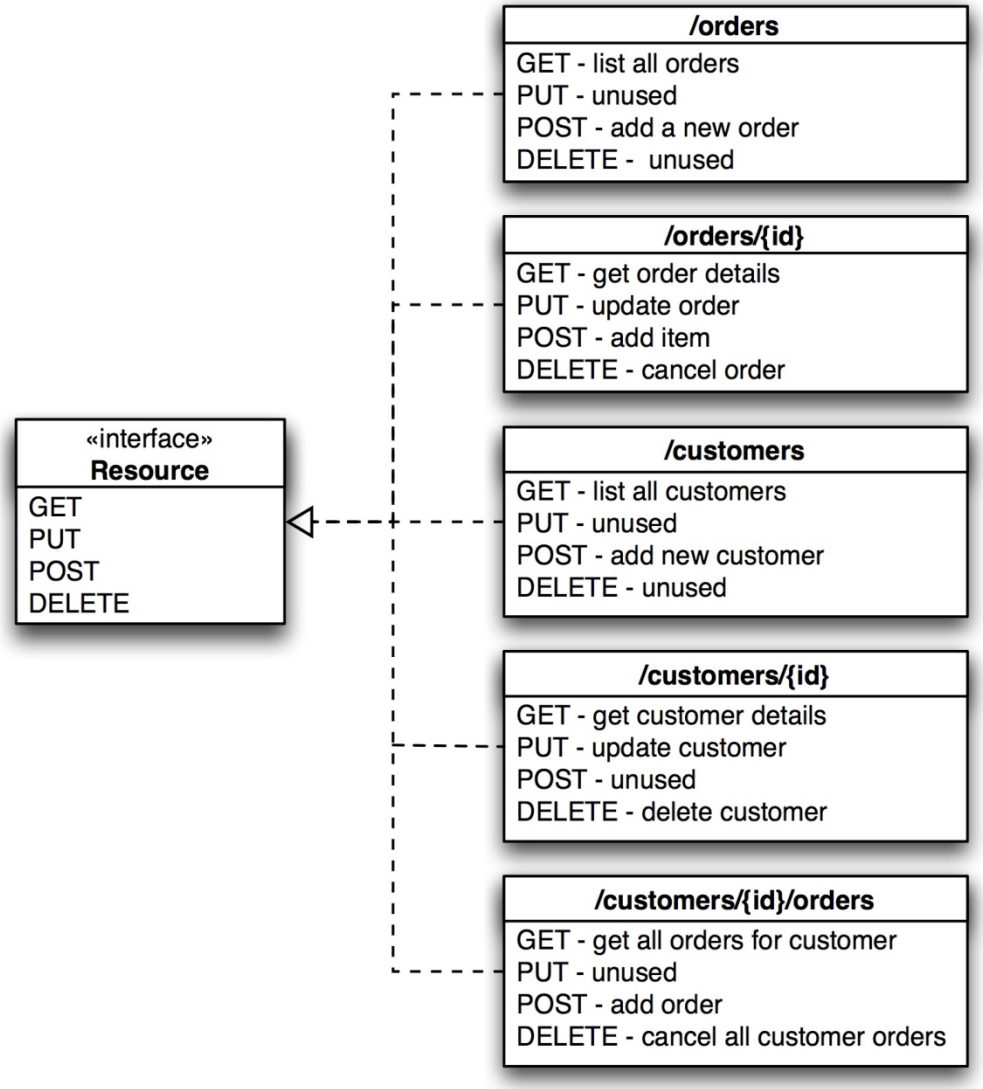
- 不使用REST化方法
  - 如果客户端程序试图使用这些服务，那它必须针对这些特定接口进行编码
  - 只贡献一个端点，“一个非常小的门，仅仅允许有钥匙的人进入其中的资源域”





REST关键原则

- 使用标准方法(http)
  - 使你的应用成为Web的一部分
  - 添加到Web中更多的资源



REST关键原则

- 资源多重表述
  - 同一个资源的不同表述，可以有利于不同的客户端。
  - 资源不仅可以被你的应用所用，还可以被任意标准Web浏览器所用

```
GET /customers/1234 HTTP/1.1
Host: example.com
Accept: application/vnd.mycompany.customer+xml
```

```
GET /customers/1234 HTTP/1.1
Host: example.com
Accept: text/x-vcard
```

三类web服务架构

- REST：面向资源的架构
  - REST架构表示方法信息都在HTTP方法中；ROR意味着作用域信息都在URI中。
- RPC式
  - 方法信息和作用域都在信封或报头中。一般通过HTTP POST访问。
- REST-RPC混合式
  - Del.icio.us
  - Flickr:

三类web服务架构

- web 服务方法信息位置
  - REST化的：直接使用http的操作，如GET等
  - REST-RPC式的：
    - Flickr的照片搜索，放置在URL中：
      http://www.flickr.com/services/rest?method=flickr.photos.search&apikey=xxx&tag=penguin
  - SOAP服务一般放在实体主体和Http报头里，如：

```
POST search/beta2 HTTP/1.1
Host: api.google.com
Content-Type: application/soap+xml
SOAPAction: urn:GoogleSearchAction

<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
 <soap:Body>
  <gs:doGoogleSearch xmlns:gs="urn:GoogleSearch">
   <q>REST</q>
   ...
  </gs:doGoogleSearch>
 </soap:Body>
</soap:Envelope>
```

三类web服务架构

- 作用域信息位置
  - **SOAP web服务，放在实体主体中**
  - **REST化的：URL中**
    - **http://www.google.cn/search?q=REST**
  - **REST-RPC混合式：URL中**
    - **http://www.flickr.com/services/rest?method=flickr.photos.search&apikey=xxx&tag=penguin**
  - 后两者不同： **REST-RPC混合式的方法信息也是在URL中，而REST化的方法信息是http方法。**

三类web服务架构

- REST式服务为不同的作用域信息暴露不同的URI，为客户端可能操作的每一则数据暴露一个URI

- 一个REST-RPC混合式服务为客户端可能进行的每一个操作暴露一个URI(比如获取数据用一个URI,删除数据用另外一个URI)

- RPC式服务一般为每个“文档处理器”（用于打开信封，并把信封转换为软件指令）暴露一个URI，即为每个处理远程调用的进程暴露一个URI

Sample REST services

- 提供 Atom 发布协议（*http://www.ietf.org/html.char ters/atompub-charter.html*）及其变型的服务，例如 GData（*http://code.google.com/apis/gdata/*）
- Amazon S3（Simple Storage Service）（*http://aws.amazon.com/s3*）
- Yahoo!提供的大部分 Web 服务（*http://developer.yahoo.com/*）
- 许多其他未采用 SOAP 的、只读的 Web 服务
- 静态网站
- 很多 Web 应用（尤其是像搜索引擎这种只读的）

Restlet

Demo Screencast

Demo simpleRestlet