第6章 面向服务的软件体系结构

XML学习网站

- **W3school**

https://www.w3school.com.cn/xml/index.asp

- **RUNOOB.com**

https://www.runoob.com/xml/xml-tutorial.html

XML的目标

- 能够直接应用在Internet上
- 能被各式应用软件使用
- 能与SGML兼容
- 能轻易发展XML相关软件
- 能简化SGML
- XML文件可读性高
- XML规范能尽快完成
- XML规范必须简洁
- XML文件易于建立
- 语法不可模糊不清

XML的特点

- 简洁有效
- 易学易用
- 开放的国际化标准
- 高效且可扩充

XML的特点 – 开放的国际化标准

- XML标准。这是W3C正式批准的，这意味着这个标准是稳定的，完全可用于Web和工具的开发。
- XML名域标准。用来描述名域的句法，支持能识别名域的XML解析器。
- DOM（Document Object Model，文档对象模型）标准。为给结构化的数据编写脚本提供标准，这样，开发人员就能够与计算机在基于XML的数据上进行交互。
- XSL标准。XSL有两个模块：XSL转换语言和XSL格式化对象。其中转换语言可用来转换XML以满足显示要求。由于XSL的两部分是模块化的，因此，转换语言能够独立地用来进行多用途的转换，包括把XML转换成结构完整的HTML。
- XLL标准和XML指针语言（XPointer）标准。XLL提供类似与HTML的链接，但功能更强大。例如，链接可以是多方向的，可以存在于对象上而不仅仅是页面上。

XML的作用

- 使得搜索更加有意义
- 开发灵活的Web应用软件
- 实现不同数据的集成
- 使用于多种应用环境
- 客户端数据处理与计算
- 数据显示多样化
- 局部数据更新
- 与现有Web发布机制相兼容
- 可升级性
- 压缩性能高

XML的应用

- 应用于客户需要与不同的数据源进行交互时
- 应用于将大量运算负荷分布在客户端
- 应用于将同一数据以不同的面貌展现给不现的用户
- 应用于网络代理对所取得的信息进行编辑、增减以适应个人用户的需要

解析XML

```
<PERSON ID="p1100" SEX="M">
        <NAME>
            <GIVEN>Judson</GIVEN>
            <SURNAME> McDaniel</SURNAME>
        </NAME>
        <BIRTH>
            <DATE>2 Feb 1934</DATE>
        </BIRTH>
        <DEATH>
            <DATE>9 Dec 2005</DATE>
        </DEATH>
</PERSON>
```

XML与HTML的区别

- HTML是一种格式化的语言，一个HTML文本可以看作一个格式化的程序
- XML是一种元标记语言
- XML定义了一套元句法，与特定领域有关的标记语言（例如，MusicML、MathML和CML等）都必须遵守。

XML文档

```
<?xml version="1.0"?>
<visit>
        <to>alluser</to>
        <from>educity.cn</from>
        <heading>welcome</heading>
        <body>Welcome to the best online education website!</body>
</visit>
```

（1）文档以XML定义<?xml version="1.0"?>开始。

（2）有一个包含所有其它内容的根元素，如上面例子中的<visit>和</visit>标记符。

（3）所有元素必须合理地嵌套，不允许交叉嵌套。

CSS

```
POEM { display: block }
TITLE { display: block; font-size: 16pt; font-weight: bold }
POET { display: block; margin-bottom: 10px }
STANZA { display: block; margin-bottom: 10px }
VERSE { display: block }
```

XSL

portfolio.xml

```
<?xml version="1.0"?>
<portfolio>
  <stock exchange="nyse">
      <name>zacx corp</name>
      <symbol>ZCXM</symbol>
      <price>28.875</price>
  </stock>
  <stock exchange="nasdaq">
      <name>zaffymat inc</name>
      <symbol>ZFFX</symbol>
      <price>92.250</price>
  </stock>
</portfolio>
```

portfolio.xsl

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
<html>
<body>
  <table border="2" bgcolor="yellow">
    <tr>
      <th>Symbol</th>
      <th>Name</th>
      <th>Price</th>
    </tr>
    <xsl:for-each select="portfolio/stock">
    <tr>
      <td><xsl:value-of select="symbol"/></td>
      <td><xsl:value-of select="name"/></td>
      <td><xsl:value-of select="price"/></td>
    </tr>
    </xsl:for-each>
  </table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

CSS与XSL的比较

- CSS只能改变特定元素的格式，也只能以元素为基础。但XSL样式单可以重新排列元素并对元素进行重排序。
- CSS的优越性在于具有广泛的浏览器支持。但是XSL更为灵活和强大，可更好地适用于XML文档。而且，带XSL样式单的XML文档可以很容易地转换为带CSS样式单的HTML文档。
- 如果只是要对一些固定数据进行排版，可以使用“HTML + CSS”方式；如果这些数据是与某些应用程序相关，并且独立于程序存在的，并且要独立于程序来使用，则应该充分使用XML技术，采用“HTML + XML + XSL”。

XML开发工具

```
<?xml version="1.0"?>
<visit>
        <to>alluser</to>
        <from>educity.cn</from>
        <heading>Welcome</heading>
        <body>Welcome to the best online education website!</body>
</visit>
```

```
<html>
<head>
<script language="JavaScript" for="window" event="onload">
        var xmlDoc = new ActiveXObject("Microsoft.XMLDOM");
        xmlDoc.async="false";
        xmlDoc.load("visit.xml");
        nodes = xmlDoc.documentElement.childNodes;
        to.innerText = nodes.item(0).text;
        from.innerText = nodes.item(1).text;
        header.innerText = nodes.item(2).text;
        body.innerText = nodes.item(3).text;
</script>
<title>HTML using XML data</title>
</head>
<body bgcolor="yellow">
        <h1>Refsnes Data Internal Note</h1>
        <b>To: </b>
        <span id="to"> </span>
        <br>
        <b>From: </b>
        <span id="from"></span>
        <hr>
        <b><span id="header"></span></b>
        <hr>
        <span id="body"></span>
</body>
</html>
```

XML建模

- 描述具体数据
- 描述数据结构和模式
- 描述数据的表现
- 描述数据中的位置
- 描述数据中的链接关系
- 描述数据的应用关系

基于XML的ADL – XADL 2.0

```
<types:component xsi:type="types:Component" types:id="xArchADT">
  <types:description xsi:type="instance:Description">xArchADT</types:description>
  <types:interface xsi:type="types:Interface" types:id="xArchADT.IFACE_TOP">
    <types:description xsi:type="instance:Description">xArchADT Top Interface</types:description>
    <types:direction xsi:type="instance:Direction">inout</types:direction>
    <types:type xsi:type="instance:XMLLink" xlink:type="simple" xlink:href="#C2TopType" />
  </types:interface>
  <types:interface xsi:type="types:Interface" types:id="xArchADT.IFACE_BOTTOM">
    <types:description xsi:type="instance:Description">xArchADT Bottom
Interface</types:description>
    <types:direction xsi:type="instance:Direction">inout</types:direction>
    <types:type xsi:type="instance:XMLLink" xlink:type="simple" xlink:href="#C2BottomType" />
  </types:interface>
  <types:type xsi:type="instan0063e:XMLLink" xlink:type="simple" xlink:href="#xArchADT_type" />
</types:component>
```

基于XML的ADL – XBA

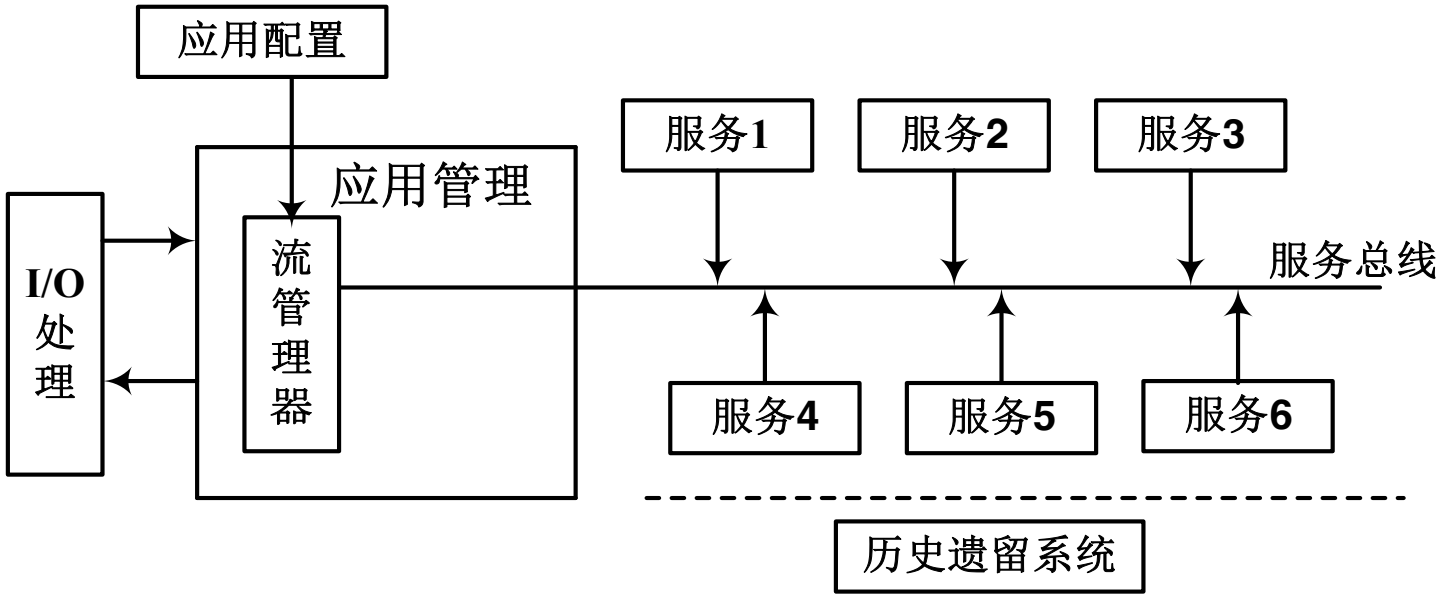
```
<complexType name="componentType">
    <sequence>
        <element name="Port" type="portType" minOccurs="0" maxOccurs="unbounded">
        <element name="Computation" type="computationType" minOccurs="0">
        <element name="Port" type="portType" minOccurs="0" maxOccurs="unbounded">
    </sequence>
    <attribute name="Name" type="string"/>
</complexType>
<complexType name="portType">
    <element name="Description" type="string" minOccurs="0"/>
    <attribute name="Name" type="string"/>
</complexType>
<complexType name="computationType">
    <element name="Description" type="string"/>
    <attribute name="Name" type="string"/>
</complexType>
```

基于XML的ADL – XBA

```
<Connector name="Rpc">
    <Role name="Source">
        <Description>
        Get request from client
        </Description>
    </Role>
    <Role name="Sink">
        <Description>
        Give request from server
        </Description>
    </Role>
    <Glue>
    Get request from Client and pass it to server through some protocol
    </Glue>
</Connector>
```

```
<Configuration name="Client-Server">
    <Component name="Client">
    <Component name="Server">
    <Component name="Rpc">
    <Instance>
        <ComponentInstance>
            <ComponentName>MyClient</ComponentName>
            <ComponentType>Client</ComponentType>
        </ComponentInstance>
        <!-- ect  -->
        <ConnectorInstance>
            <ComponentName>MyRpc</ComponentName>
            <ComponentType>Rpc</ComponentType>
        </ConnectorInstance>
    </Instance>
    <Attachments>
        <Attachment>
            <From>MyClient.request</From>
            <To>MyRpc.Source</To>
        <Attachment>
        <!-- ect  -->
    </Attachments>
</Configuration>
```

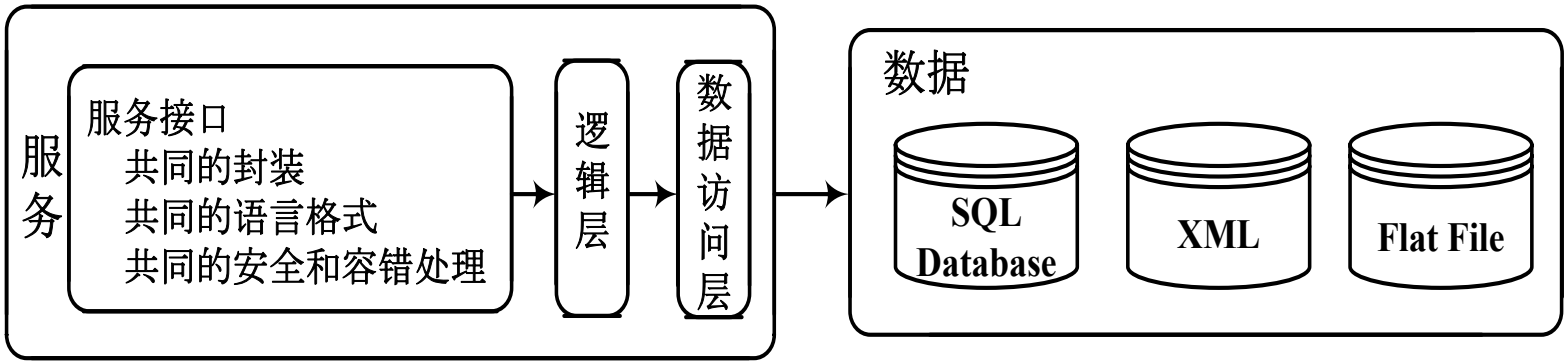
SOA的概念 – SOA模型示例



SOA的概念

- W3C：SOA是一种应用程序体系结构，在这种体系结构中，**所有功能都定义为独立的服务**，这些服务带有定义明确的**可调用接口**，能够以定义好的顺序调用这些服务来形成业务流程。
- Service-architecture.com：服务是**精确定义、封装完整**、独立于其它服务所处环境和状态的函数。SOA本质上是服务的集合，**服务之间彼此通信**，这种通信可能是简单的数据传送，也可能是两个或更多的服务协调进行某些活动。服务之间需要某些方法进行连接。
- Gartner：SOA是一种**C/S体系结构**的软件设计方法，应用由服务和服务使用者组成，SOA与大多数通用的C/S体系结构模型不同之处，在于它着重**强调构件的松散耦合，并使用独立的标准接口。**

SOA的概念 – 单个服务的内部结构



SOA的概念 – SOA的特征

- 松散耦合
- 粗粒度服务
- 标准化接口

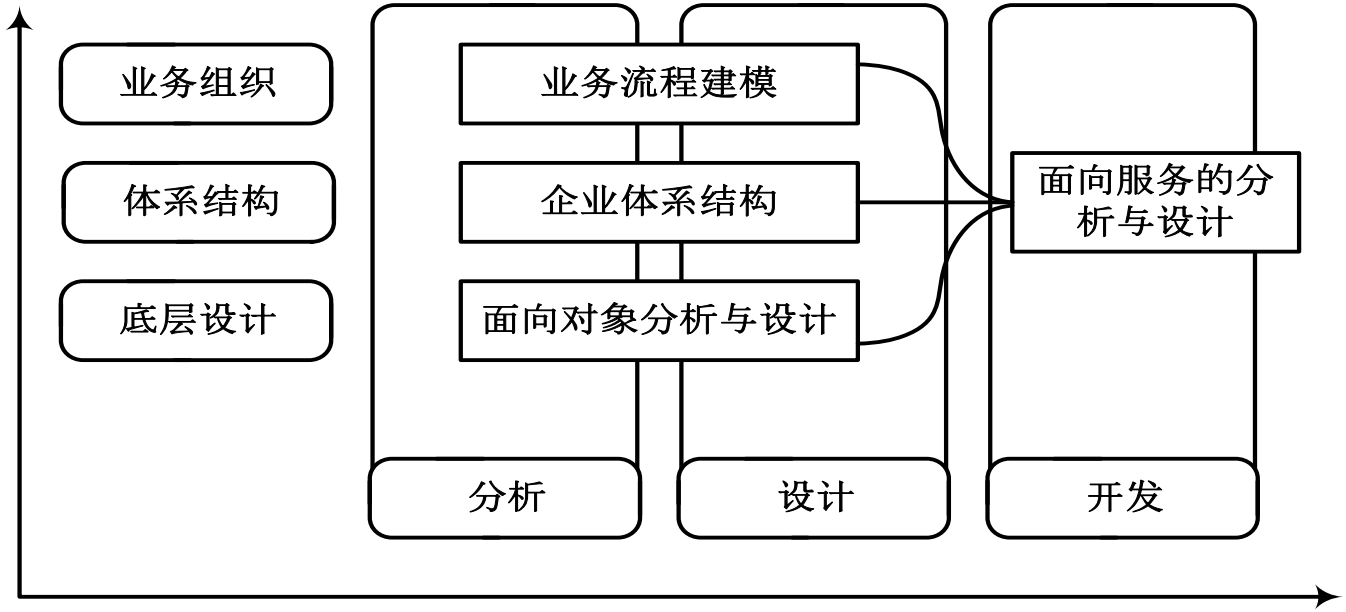
SOA的概念 – 服务构件与传统构件

- **服务构件体系结构**：它可简化使用 SOA 进行的应用程序开发和实现工作。SCA 提供了构建粗粒度构件的机制，这些粗粒度构件由细粒度构件组装而成。SCA将传统中间件编程从业务逻辑分离出来，从而使程序员免受其复杂性的困扰。它允许开发人员集中精力编写业务逻辑，而不必将大量的时间花费在更为底层的技术实现上。
- **SCA服务构件与传统构件的主要区别**：服务构件往往是粗粒度的，而传统构件以细粒度居多；服务构件的接口是标准的，主要是服务描述语言接口，而传统构件常以具体API形式出现；服务构件的实现与语言是无关的，而传统构件常绑定某种特定的语言；服务构件可以通过构件容器提供QoS的服务，而传统构件完全由程序代码直接控制。

SOA的概念 – SOA的设计原则

- 明确定义的接口
- 自包含和模块化
- 粗粒度
- 松耦合
- 互操作性、兼容和策略声明

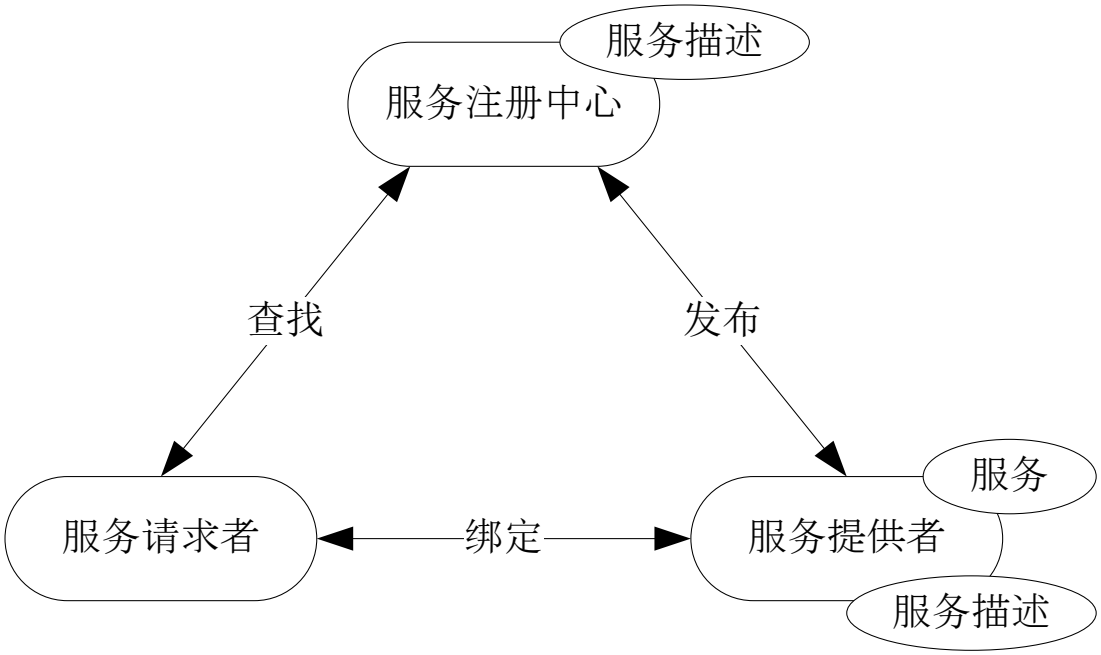
面向服务的分析与设计



SOA的关键技术

发现服务层	UDDI、DISCO
描述服务层	WSDL、XML Schema
消息格式层	SOAP、REST
编码格式层	XML
传输协议层	HTTP、TCP/IP、SMTP等

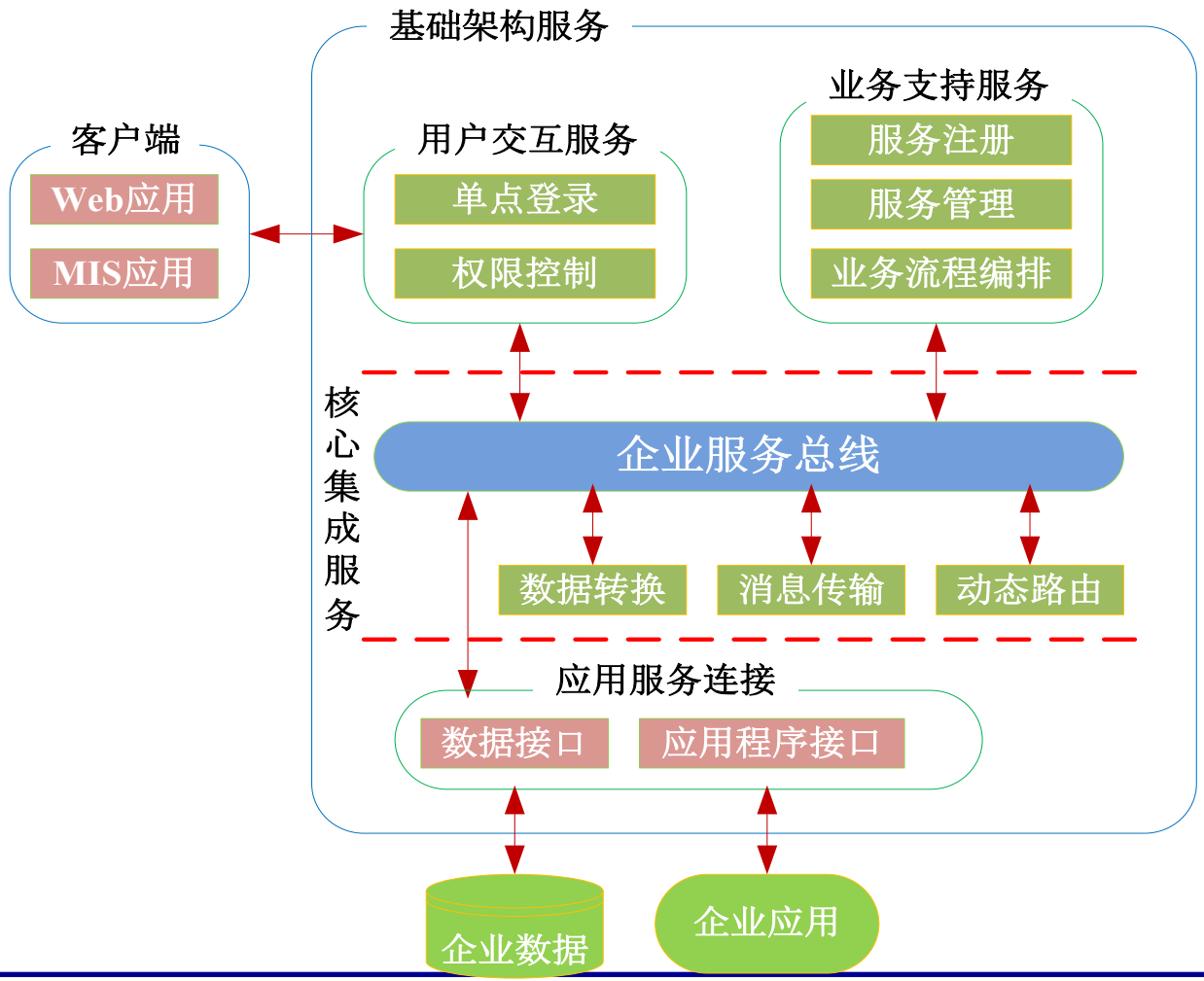
SOA的实现方法 – Web服务



SOA的实现方法 – 服务注册表

- **服务注册。**服务注册是指服务提供者向服务注册表发布服务的功能（服务合约），包括服务身份、位置、方法、绑定、配置、方案和策略等描述性属性。
- **服务位置。**服务位置是指服务使用者，帮助它们查询已注册的服务，寻找符合自身要求的服务。
- **服务绑定。**服务使用者利用查找到的服务合约来开发代码，开发的代码将与注册的服务进行绑定，调用注册的服务，以及与它们实现互动。

SOA的实现方法 – 企业服务总线 – 模型



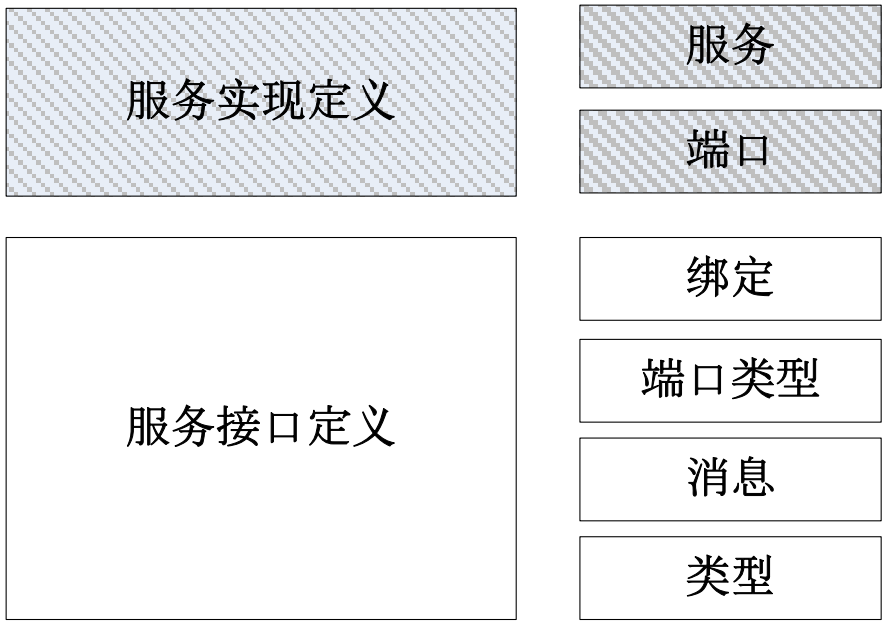
SOA的实现方法 – 企业服务总线 – 功能

- 提供位置透明性的消息路由和寻址服务
- 提供服务注册和命名的管理功能
- 支持多种的消息传递范型
- 支持多种可以广泛使用的传输协议
- 支持多种数据格式及其相互转换
- 提供日志和监控功能

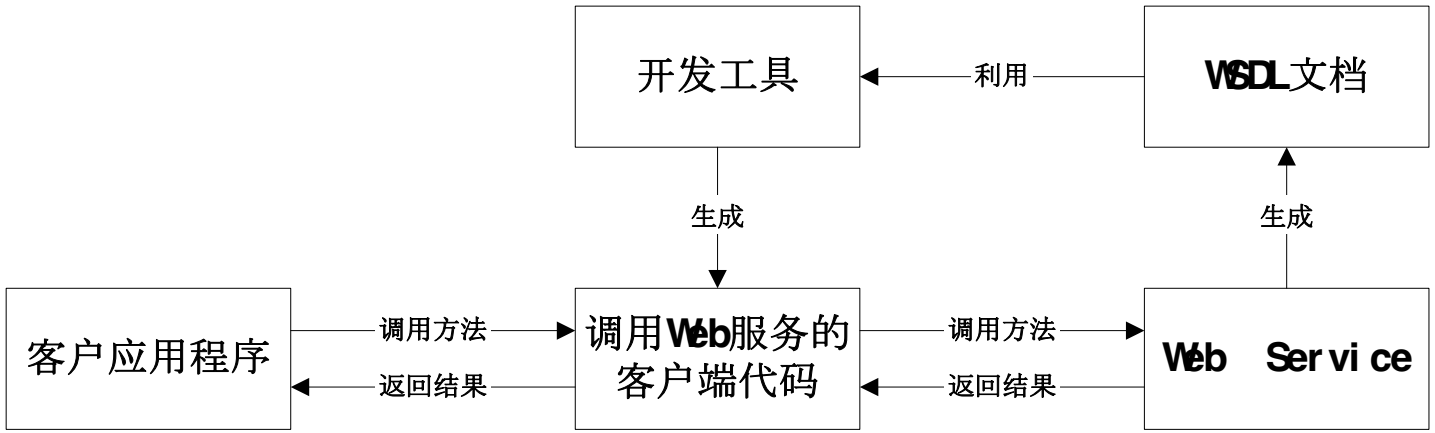
SOA的实现方法 – 企业服务总线 – 优势

- 扩展的、基于标准的连接。
- 灵活的、服务导向的应用组合
- 提高复用率，降低成本
- 减少市场反应时间，提高生产率

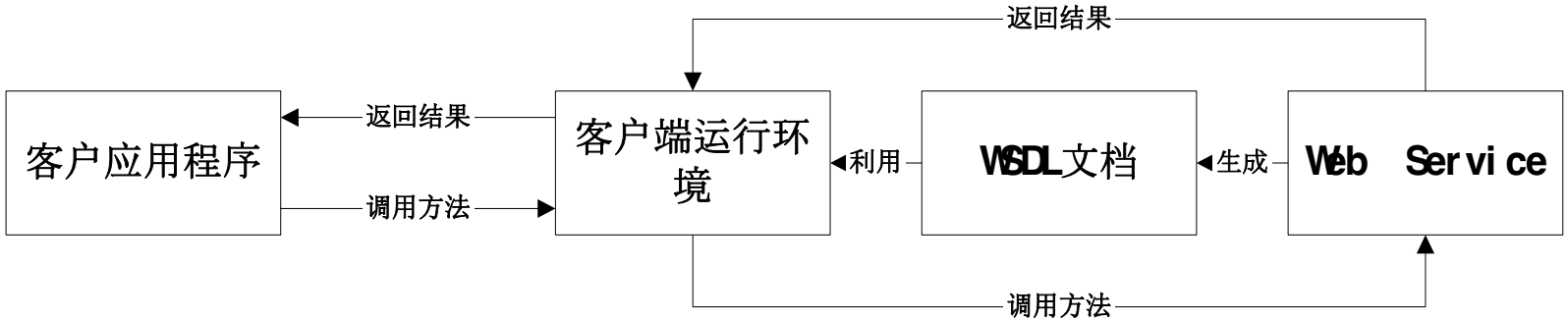
WSDL – 基本服务描述



WSDL – 使用WSDL文档



设计期WSDL文档的使用方法

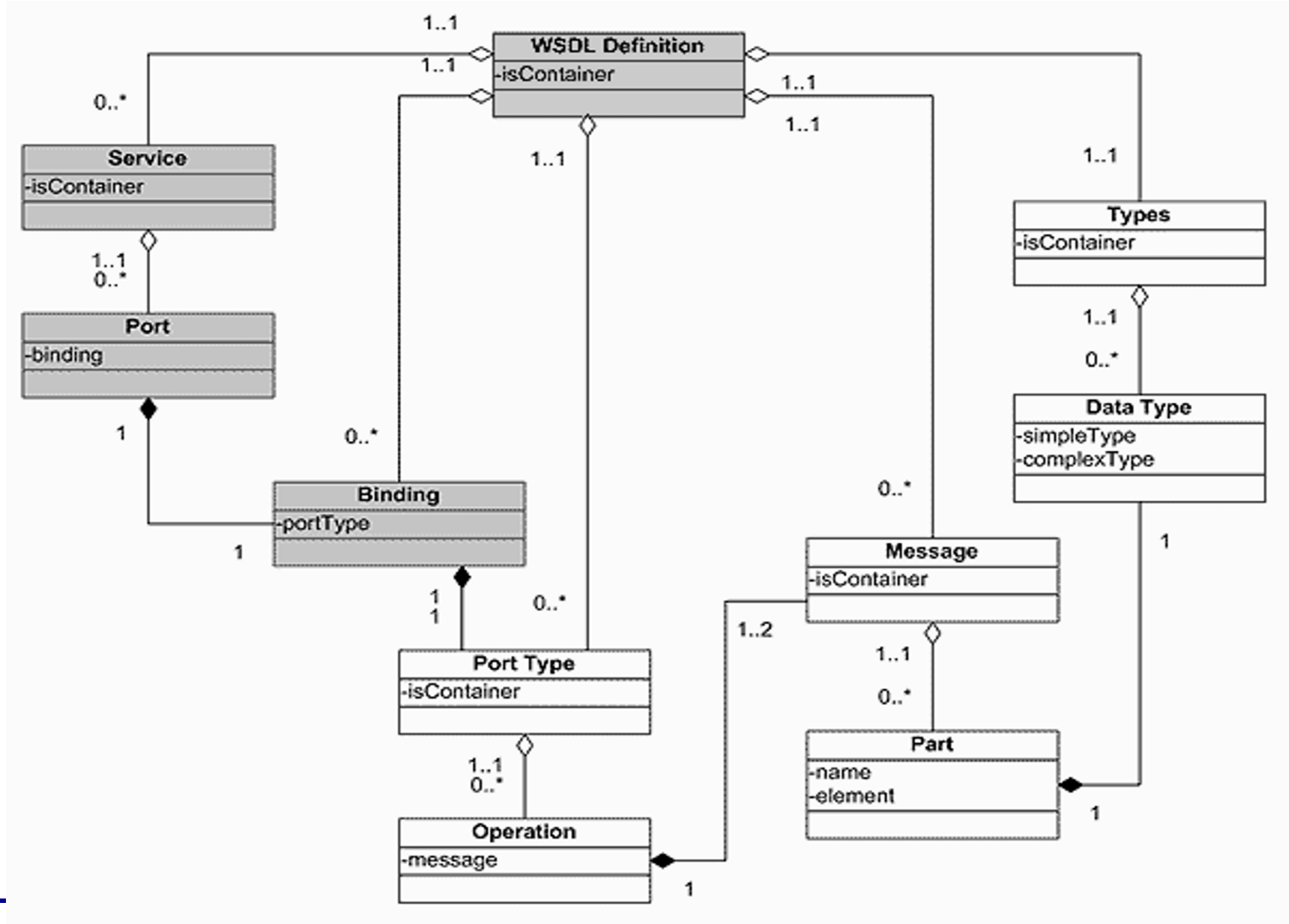


运行期WSDL文档的使用方法

WSDL – 文档结构

- **types（类型）**：数据类型定义的容器，提供了用于描述正在交换的消息的数据类型定义，一般使用XML Schema中的类型系统。
- **message（消息）**：通信消息数据结构的抽象定义。message使用types所定义的类型来定义整个消息的数据结构。
- **operation（操作）**：对服务中所支持的操作的抽象描述，一般单个operation描述了一对访问入口的请求/响应消息。
- **porttype（端口类型）**：描述了一组操作，每个操作指向一个输入消息和多个输出消息规范。
- **binding（绑定）**：为特定端口类型定义的操作和消息制定具体的协议和数据格式。
- **port（端口）**：指定用于绑定的地址，定义服务访问点。
- **service**：相关服务访问点的集合。

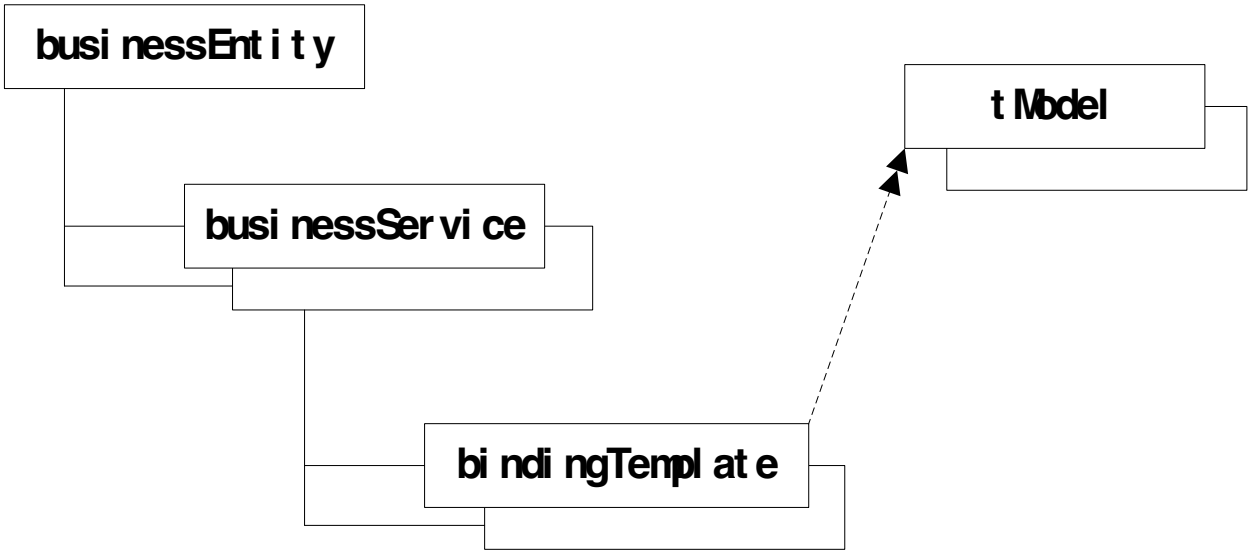
WSDL – 文档结构


WSDL Definition
-isContainer
Service
-isContainer
Port
-binding
Binding
-portType
Port Type
-isContainer
Operation
-message
Message
-isContainer
Part
-name
-element
Types
-isContainer
Data Type
-simpleType
-complexType
1..1
0..*
1..2

UDDI – 概述

- **UDDI数据模型。**一个用于描述企业和服务的XML Schema
- **UDDI API。**一组用于查找或发布UDDI数据的方法，基于SOAP
- **UDDI注册服务。**一种基础设施，对应着服务注册中心的角色

UDDI – 数据模型


businessEntity
businessService
bindingTemplate
tModel

UDDI – 注册Web服务

- 确定Web服务的tModel（WSDL文档）。
- 确定组织（企业、事业单位、各类机构等，下同）名称、简介以及所提供的Web服务的主要联系方法。
- 确定组织正确的标识和分类。
- 确定组织通过UDDI提供的Web服务。
- 确定所提供的Web服务的正确分类

UDDI – 调用Web服务

- 编写调用Web服务的程序时，程序员使用UDDI注册中心来**定位并获得businessEntity**。
- 程序员可以进一步获得**更详细的businessService信息**，或是得到一个完整的businessEntity结构。因为businessEntity结构中包含了已发布的Web服务的全部信息，所以，程序员可以从中选择一个bindingTemplate待以后使用。
- Web服务在**tModel中提供了相关规范的引用地址**，程序员可以根据引用地址获得规范并编写程序。
- 运行时，程序员可以按需要使用已经保存下来的**bindingTemplate信息**，调用Web Service。

SOAP – 概述

- **SOAP封装。**定义一个整体框架，用来表示消息中包含什么内容，谁来处理这些内容，以及这些内容是可选的或是必需的
- **SOAP编码规则。**定义了一种序列化的机制，用于交换系统所定义的数据类型的实例
- **SOAP RPC表示。**定义一个用来表示远程过程调用和应答的协议
- **SOAP绑定。**定义一个使用底层传输协议来完成在节点间交换SOAP信封的约定

SOAP – 消息封装

- **封装。**封装的元素名是“Envelope”，在表示消息的XML文档中，封装是顶层元素，在SOAP消息中必须出现。
- **SOAP头。**SOAP头的元素名是“Header”，提供了向SOAP消息中添加关于这条SOAP消息的某些要素（feature）的机制。SOAP定义了少量的属性用来表明这项要素是否可选以及由谁来处理。SOAP头在SOAP消息中可能出现，也可能不出现。如果出现的话，必须是SOAP封装元素的第一个直接子元素。SOAP头可以包含多个条目，每个条目都是SOAP头元素的直接子元素。
- **SOAP体。**SOAP体的元素名是“Body”，是包含消息的最终接收者想要的信息的容器。SOAP体在SOAP消息中必须出现且必须是SOAP封装元素的直接子元素。如果有头元素，则SOAP体必须直接跟在SOAP头元素之后；如果没有头元素，则SOAP体必须是SOAP封装元素的第一个直接子元素。SOAP体可以包括多个条目，每个条目必须是SOAP体元素的直接子元素。

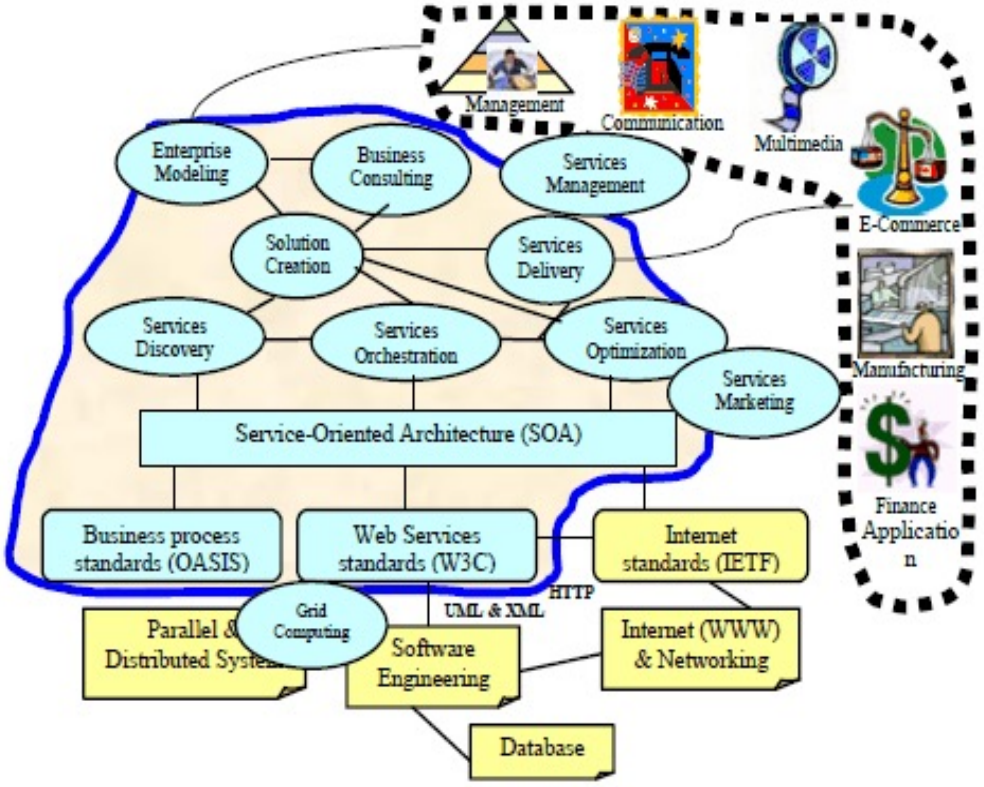
SOAP – 编码规则

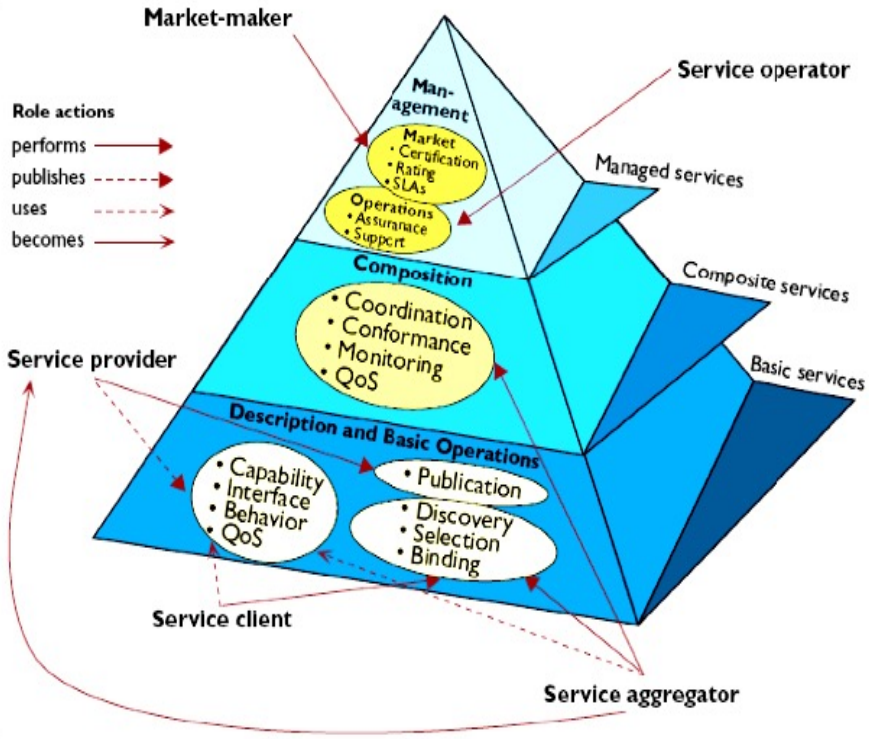
- 值（value）。值是一个字符串、类型（数字、日期、枚举等）的名或是几个简单值的组合。所有的值都有特定的类型。
- 简单值（simple value）。简单值没有名部分，例如，特定的字符串、整数、枚举值等。
- 复合值（compound value）。复合值是相关的值的组合，例如，定单、股票报表、街道地址等。在复合值中，每个相关的值都以名、序号或这两者来区分，这称为访问者（accessor）。在复合值中，多个访问者有相同的名是允许的。
- 数组（array）。数组是一个复合值，成员值按照在数组中的位置相互区分。
- 结构（struct）。结构也是一个复合值，成员值之间的唯一区别是访问者的名字，访问者名互不相同。
- 简单类型（simple type）。简单类型是简单值的类，例如，字符串类、整数类、枚举类等。
- 复合类型（compound type）。复合类型是复合值的类，它们有相同的访问者名，但可能会有不同的值。

SOAP – 在RPC中使用SOAP

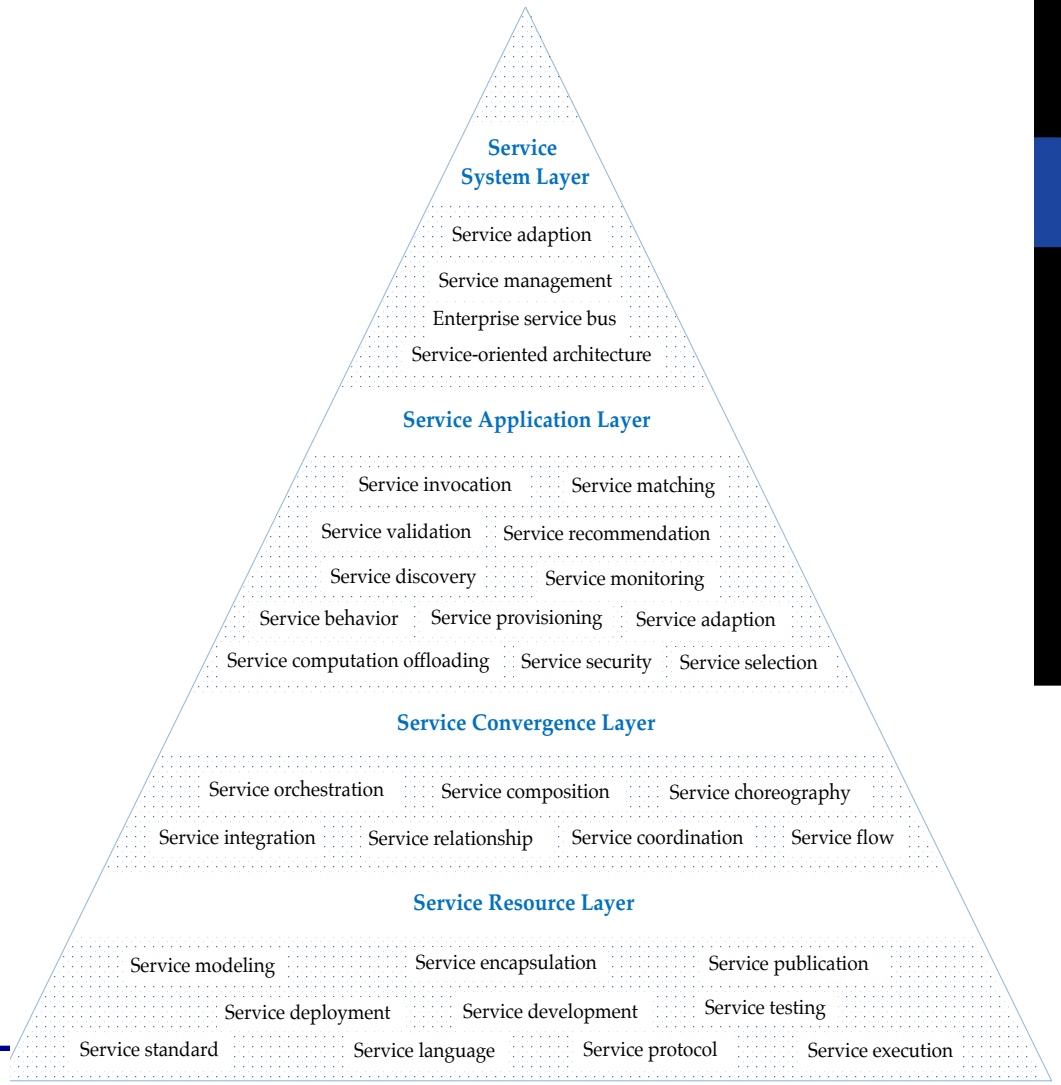
- 设计SOAP的目的之一就是利用XML的扩展性和灵活性来封装和交换RPC调用，在RPC中使用SOAP和SOAP协议绑定是紧密相关的。在使用HTTP作为绑定协议时，一个RPC调用自然地映射到一个HTTP请求，RPC应答同样映射到HTTP应答。但是，在RPC中使用SOAP并不限于绑定HTTP协议。
- 要进行方法调用，一般需要以下信息：目标对象的URI、方法名、方法签名（signature）、方法的参数、头数据，其中方法签名和头数据是可选的。SOAP依靠协议绑定提供传送URI的机制。例如，对HTTP来说，请求的URI指出了调用的来源。除了必须是一个合法的URI之外，SOAP对一个地址的格式没有任何限制

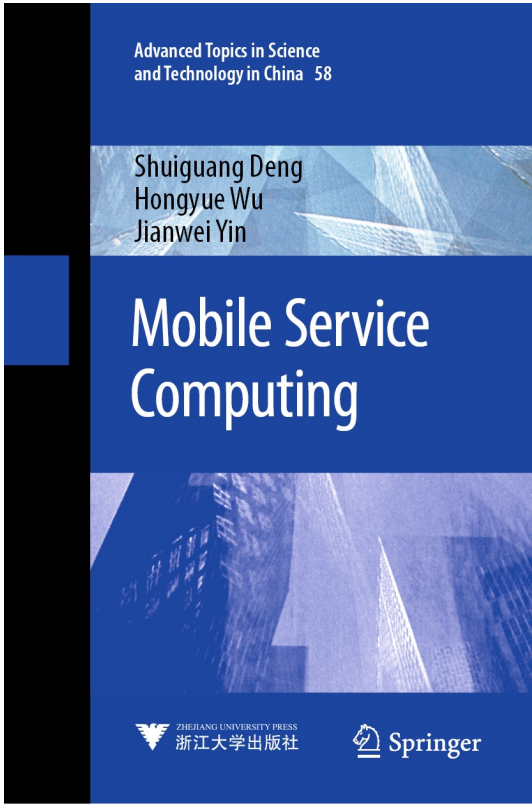
服务计算的技术体系


Management
Communication
Multimedia
E-Commerce
Manufacturing
Finance Application
Enterprise Modeling
Business Consulting
Services Management
Solution Creation
Services Delivery
Services Discovery
Services Orchestration
Services Optimization
Services Marketing
Service-Oriented Architecture (SOA)
Business process standards (OASIS)
Web Services standards (W3C)
Internet standards (IETF)
HTTP
UML & XML
Grid Computing
Parallel & Distributed Systems
Software Engineering
Internet (WWW) & Networking
Database

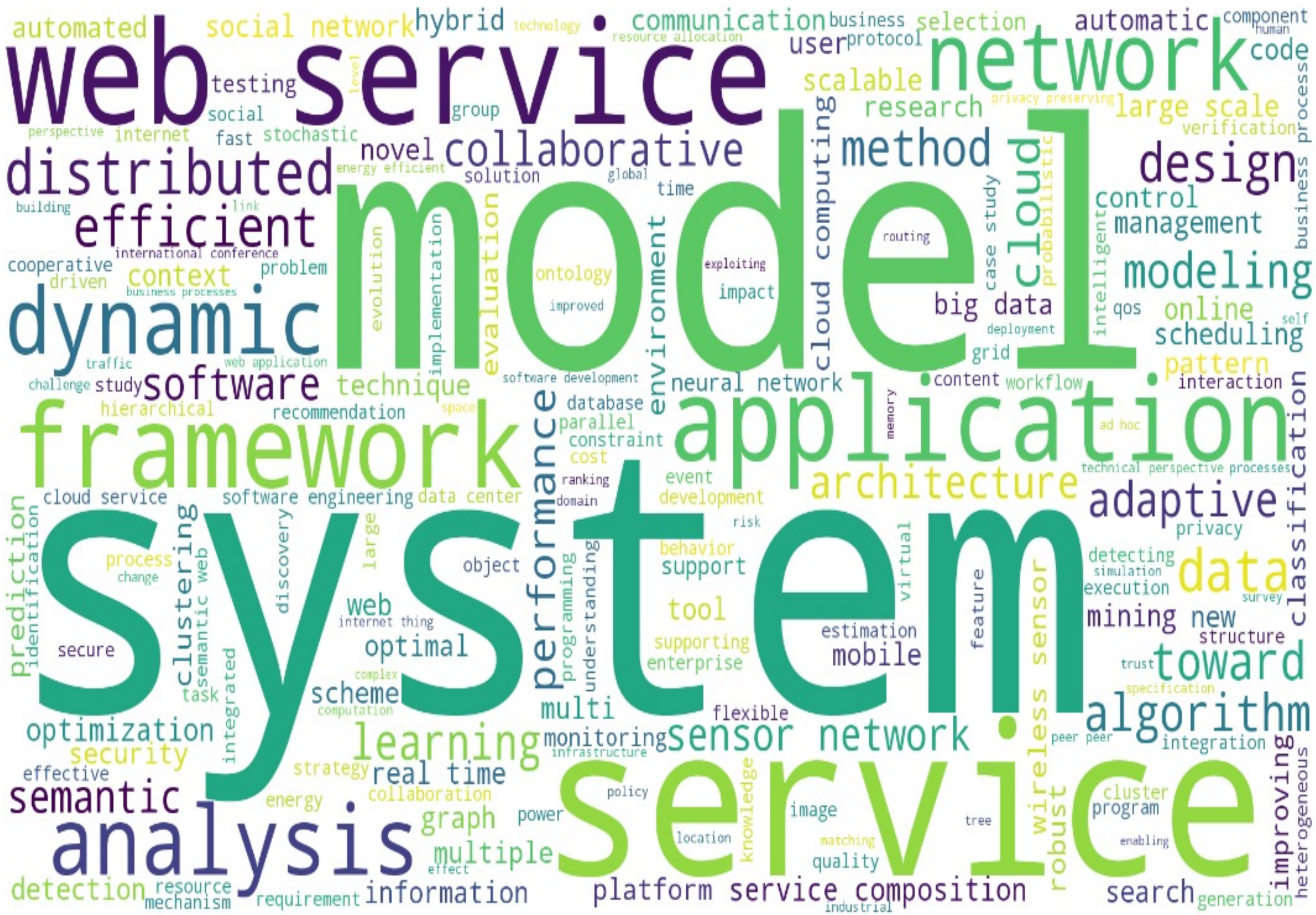

Market-maker
Service operator
Role actions
performs
publishes
uses
becomes
Man-agement
Market
• Certification
• Rating
• SLAs
Operations
• Assurance
• Support
Managed services
Composition
• Coordination
• Conformance
• Monitoring
• QoS
Composite services
Service provider
Basic services
Description and Basic Operations
• Capability
• Interface
• Behavior
• QoS
• Publication
• Discovery
• Selection
• Binding
Service client
Service aggregator

服务计算的技术体系


Service System Layer
Service adaption
Service management
Enterprise service bus
Service-oriented architecture
Service Application Layer
Service invocation
Service matching
Service validation
Service recommendation
Service discovery
Service monitoring
Service behavior
Service provisioning
Service adaption
Service computation offloading
Service security
Service selection
Service Convergence Layer
Service orchestration
Service composition
Service choreography
Service integration
Service relationship
Service coordination
Service flow
Service Resource Layer
Service modeling
Service encapsulation
Service publication
Service deployment
Service development
Service testing
Service standard
Service language
Service protocol
Service execution

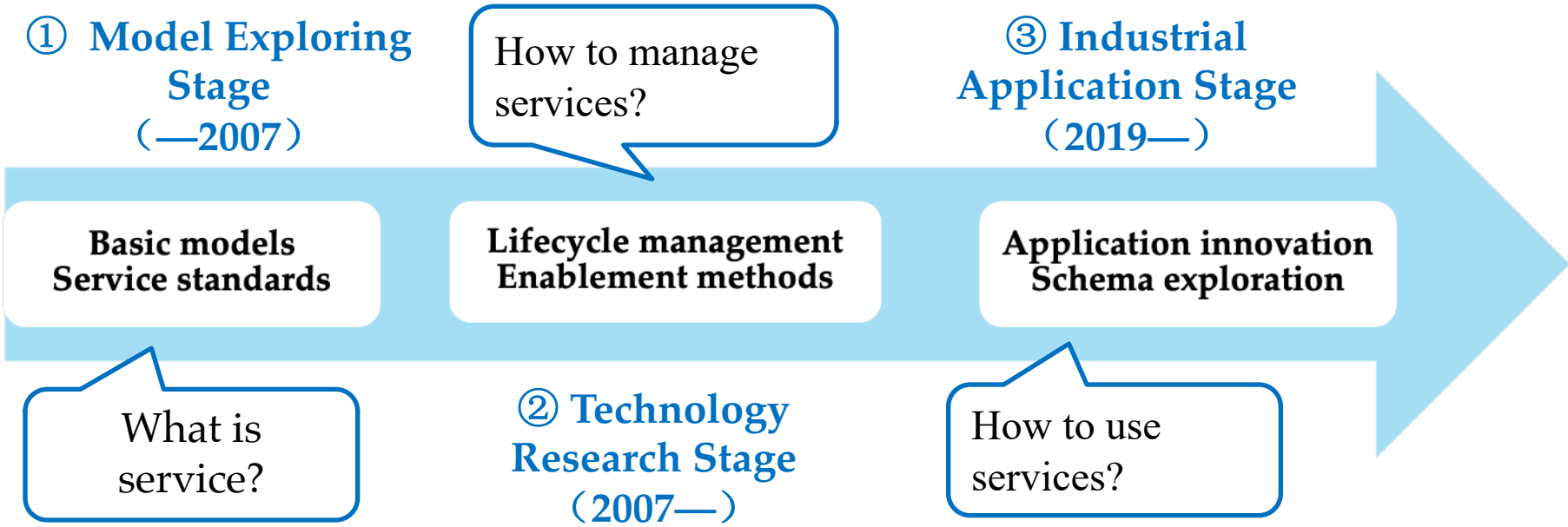

Advanced Topics in Science and Technology in China 58
Shuiguang Deng
Hongyue Wu
Jianwei Yin
Mobile Service Computing
ZHEJIANG UNIVERSITY PRESS
浙江大学出版社
Springer

服务计算的研究热点



system
model
service
web service
application
framework
network
analysis
dynamic
distributed
efficient
collaborative
performance
cloud computing
cloud
method
design
modeling
algorithm
data
adaptive
architecture
sensor network
toward
learning
software
semantic
optimization
security
detection
information
platform service composition
search
mobile
management
control
scheduling
online
big data
neural network
technique
environment
evaluation
implementation
evolution
ontology
impact
clustering
prediction
identification
discovery
optimal
scheme
multi
monitoring
graph
multiple
real time
quality
image
location
knowledge
wireless sensor
robust
improving
heterogeneous
classification
business process
intelligent
probabilistic
case study
grid
qos
pattern
interaction
workflow
content
database
parallel
constraint
cost
event
development
support
tool
supporting
enterprise
estimation
virtual
feature
mining
new
structure
detecting
simulation
execution
privacy
cluster
program
generation
integration
peer
automated
social network
hybrid
communication
business
selection
automatic
component
code
user
protocol
scalable
research
large scale
verification
testing
social
group
internet
fast
stochastic
novel
solution
time
energy efficient
cooperative
driven
context
problem
international conference
building
link
traffic
challenge
study
hierarchical
recommendation
cloud service
software engineering
data center
process
change
secure
task
integrated
large
object
web
internet thing
complex
computation
strategy
collaboration
energy
power
effective
resource
mechanism
requirement
policy
matching
tree
enabling
flexible
industrial
trust
specification
survey
routing
exploiting
improved
deployment
ranking
domain
risk
behavior
programming
understanding
technical perspective processes

服务计算的发展历程-三个阶段



服务计算的发展历程-七次浪潮

- Web services
- Social computing services
- Mobile services
- Enterprise computing
- Cloud services
- Big data and data analytics
- The vision of making everything as a service