

软件体系结构原理、方法与实践

SOA的概念--架构的演化



应用程序部署在一台服务器上，应用程序以单独的实例运行的情况。

场景：适合初创企业，快速为客户提供服务。

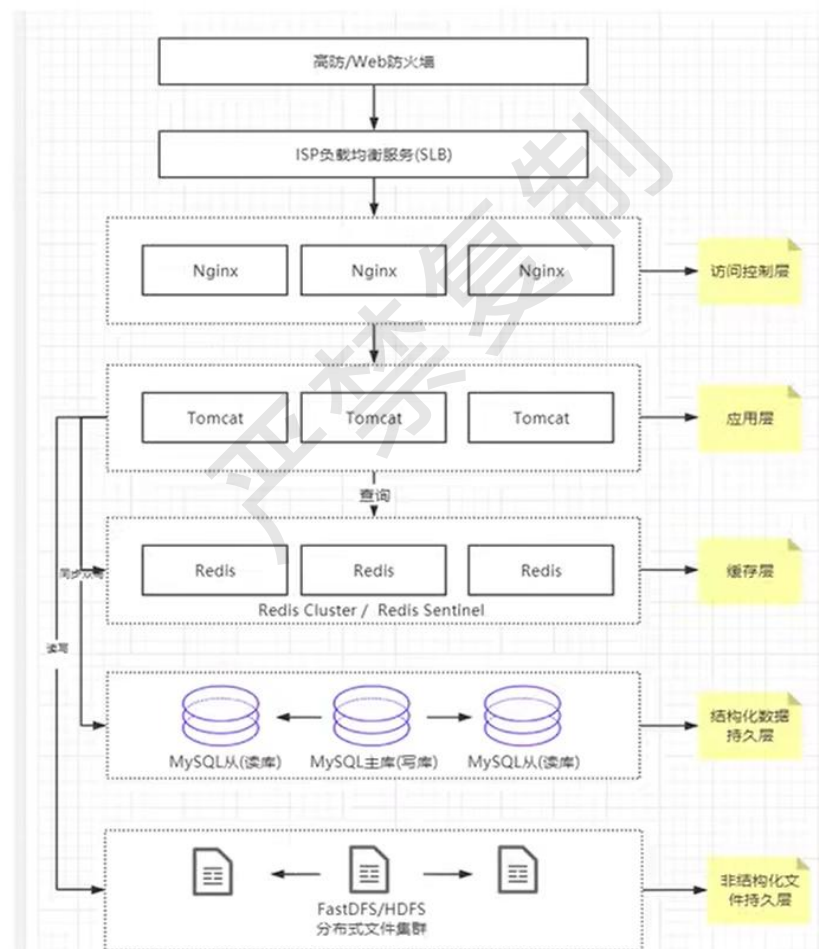
特点：快速交付，成本低。

❁ SOA的概念--架构的演化

- 单点架构存在的问题
 - 资源利用率低
 - 不具备分区容错性
 - 查询效率低
- 改进办法
 - 部署应用集群
 - 部署分布式缓存
 - 实现数据库高可用和读写分离

SOA的概念--架构的演化

集群架构模式-强调可用性



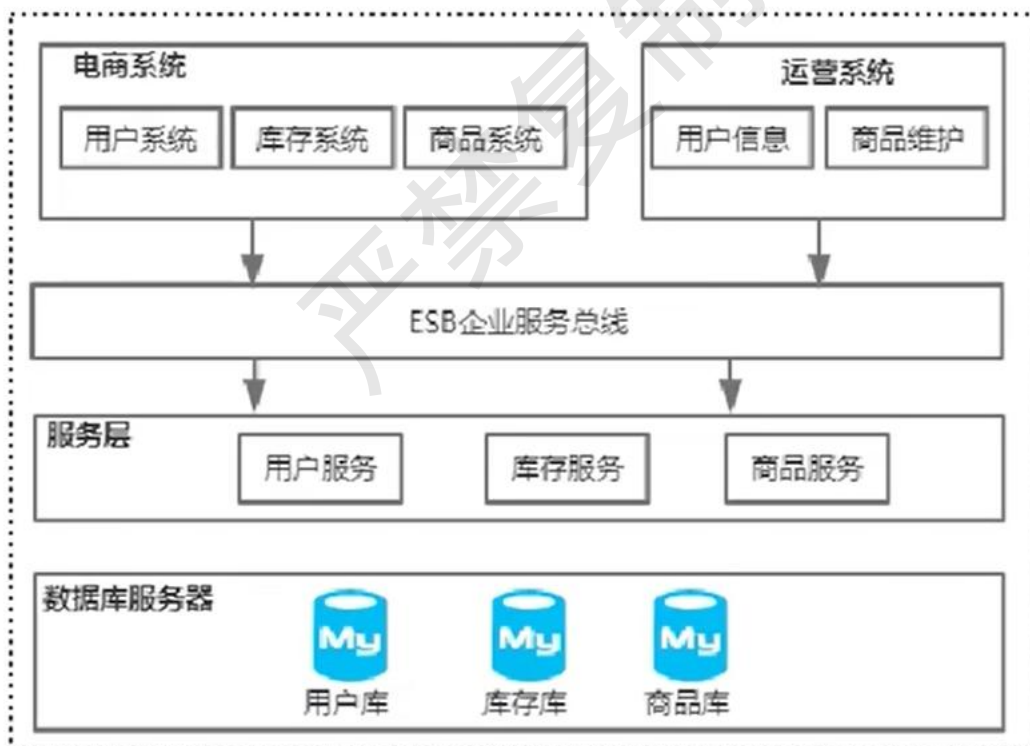


❁ SOA的概念--架构的演化

- 集群架构存在的问题
 - 技术驱动，非业务驱动
 - 业务耦合严重
 - 单点稳定性差
 - 多人协作困难
- 改进办法
 - 服务化改造，以业务为中心
 - 垂直拆分业务子系统降低耦合

SOA的概念--架构的演化

服务化架构-强调对业务垂直拆分形成多个服务模块
服务层独立管理独立维护

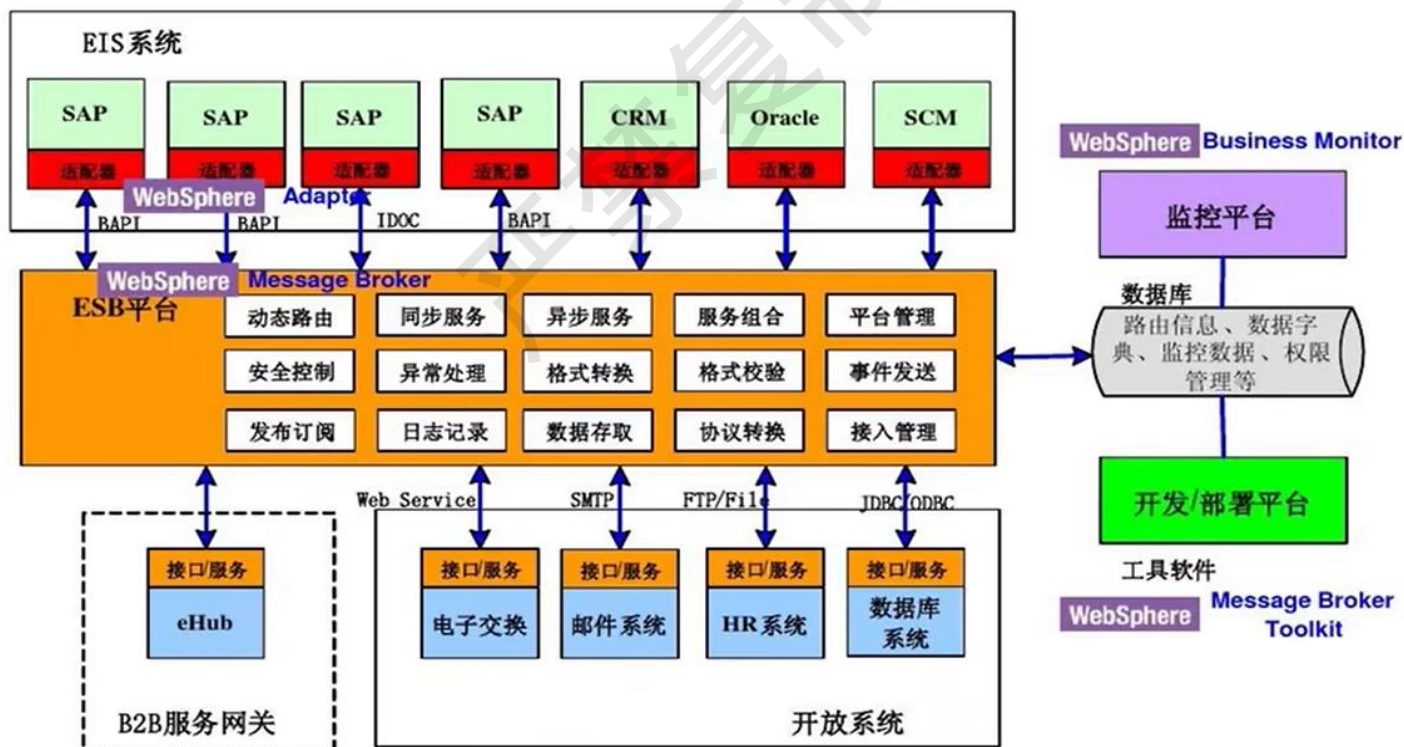


SOA的概念--架构的演化

面向服务架构 (SOA)

核心Enterprise Service Bus产品

分布式应用是缺少标准的，而ESB是重量级的产品



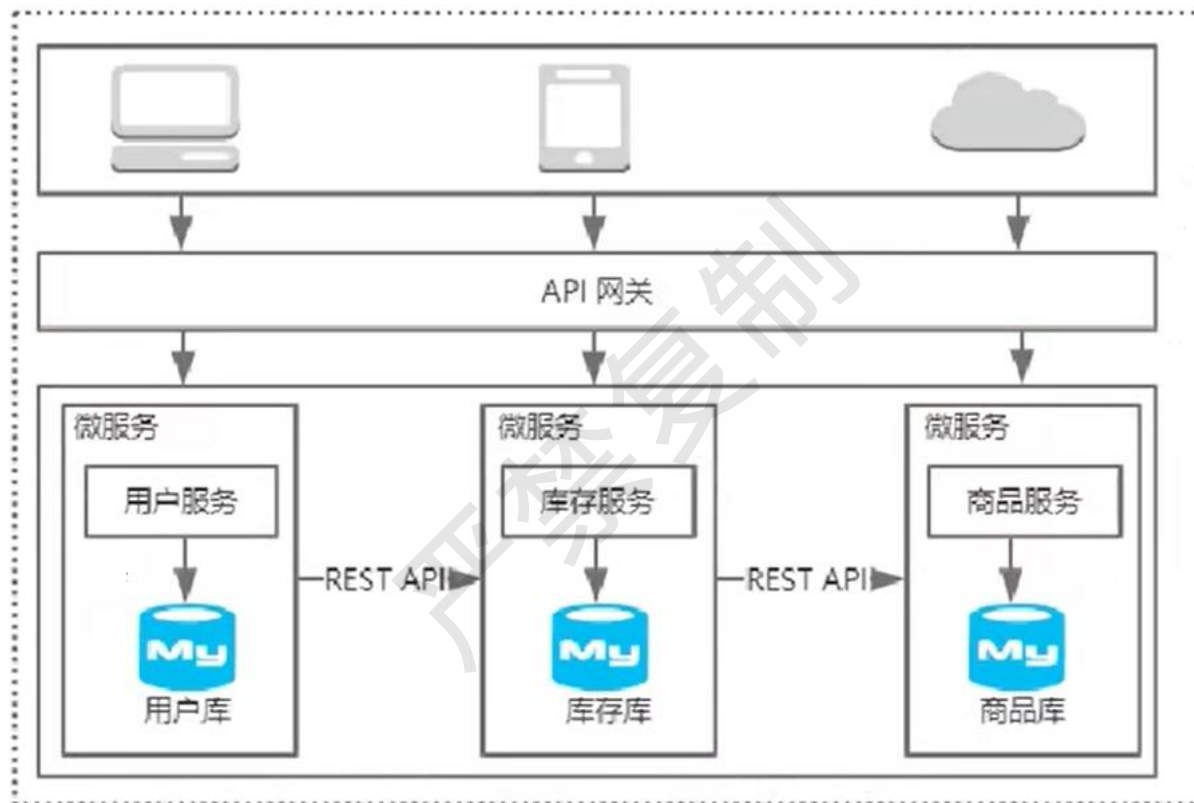
新的开始，新的起点，让我们一起为梦想而努力。



❁ SOA的概念--架构的演化

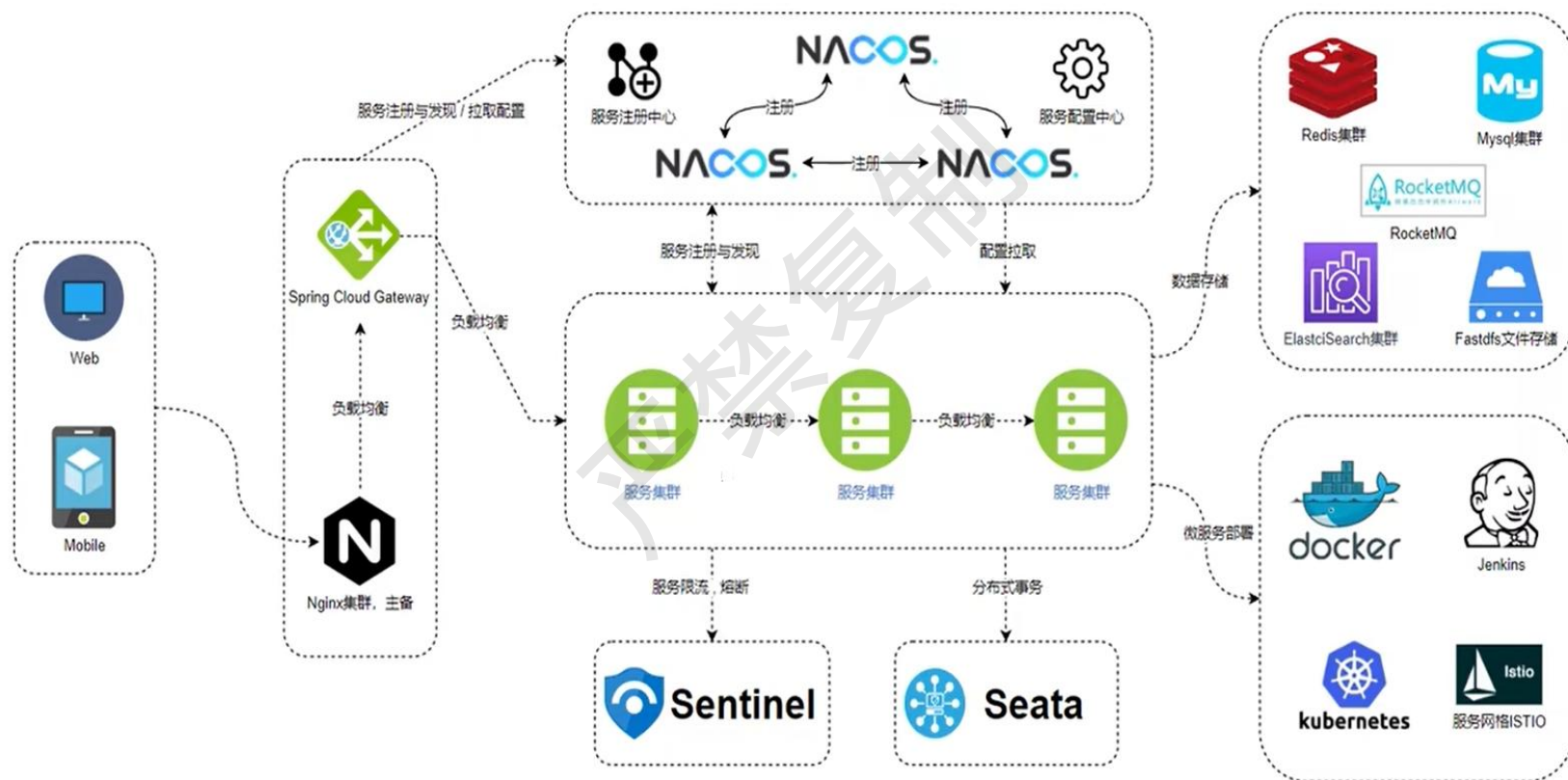
- SOA的问题
 - 子系统之间没有统一的通信标准
 - ESB成本高，没有好用的开源，被大厂绑架
 - ESB属于重量级产品，部署规划异常笨重
- 改进办法
 - 引入微服务架构模式

SOA的概念--架构的演化




微服务架构风格是一种将单个应用程序开发为一组小型服务的方法，每个小服务运行在自己的进程中，并且以轻量级机制(通常是HTTP REST API)通信。这些服务是围绕业务能力建立的，并且可以由完全自动化的部署机构独立部署。这些服务的集中管理只有最低限度，可以用不同的编程语言编写并且使用不同的数据存储技术。

SOA的概念--架构的演化



❁ SOA的概念--架构的演化



总结

单体架构特点？

- 简单方便，高度耦合，扩展性差，适合小型项目。例如：学生管理系统

分布式架构特点？

- 松耦合，扩展性好，但架构复杂，难度大。适合大型互联网项目，例如：京东、淘宝

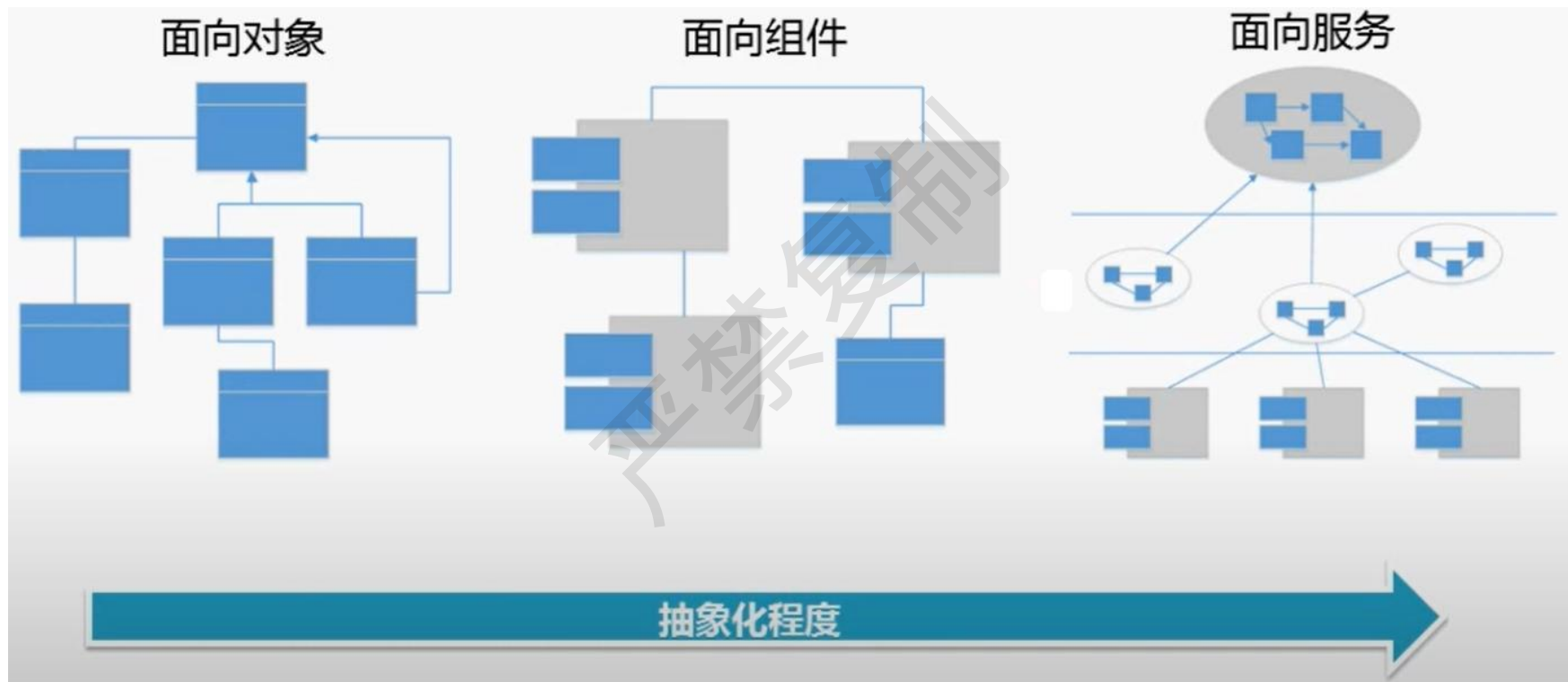
微服务：一种良好的分布式架构方案

- 优点：拆分粒度更小、服务更独立、耦合度更低
- 缺点：架构非常复杂，运维、监控、部署难度提高

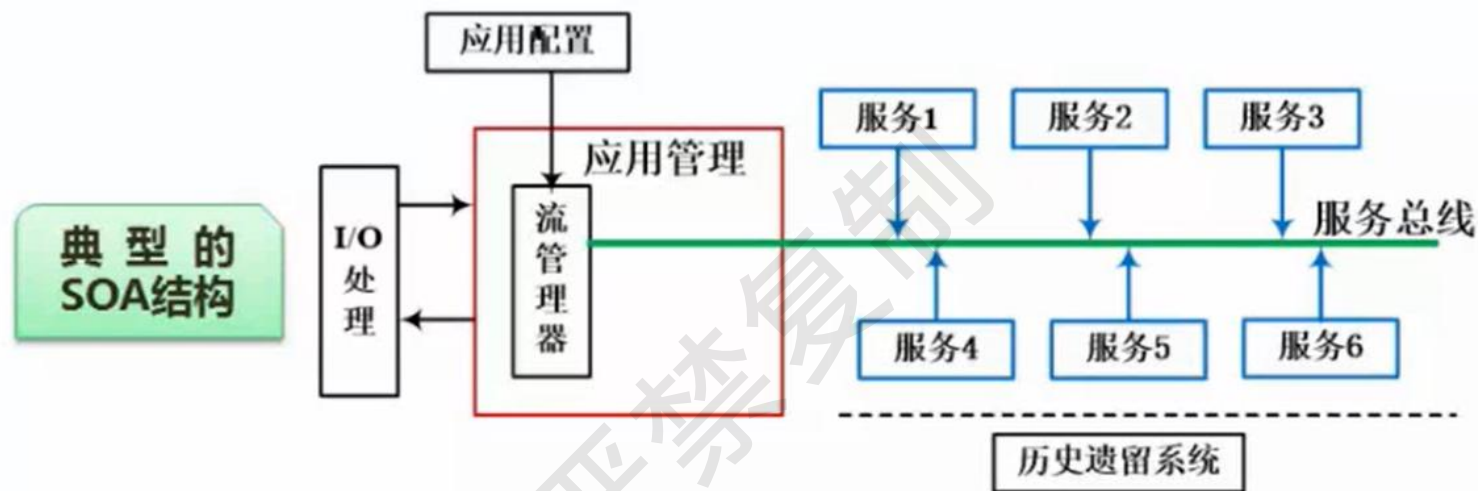
✿ SOA的概念

- 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 – SOA必然性



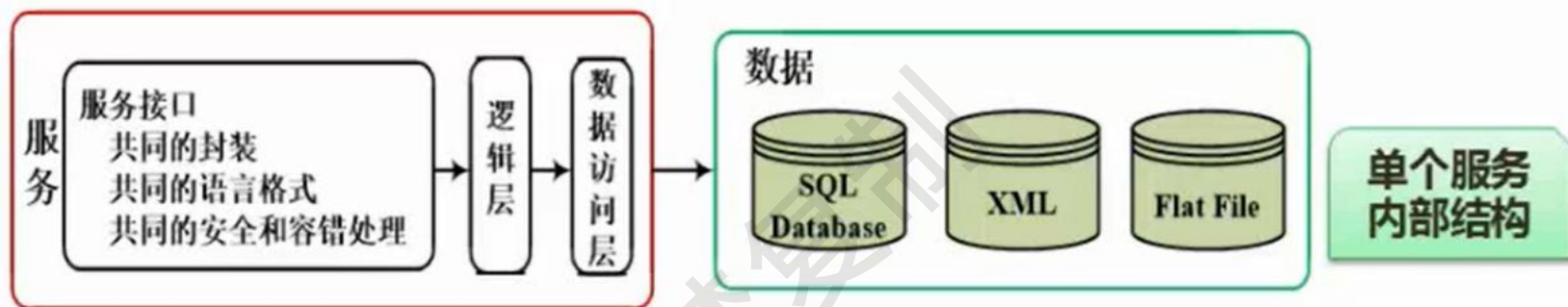
SOA的概念 – SOA模型示例



- 在基于服务的体系结构中，所有的功能都定义成了独立的服务，服务之间通过交互完成业务整体逻辑。
- 所有的服务都通过服务总线或流程管理器进行连接。
- 无须考虑各个服务的内部实现细节及部署平台

新的开始，新的起点，让我们一起为梦想而努力。

SOA的概念 – 单个服务的内部结构



表示层与逻辑层分离，增加了接口层，通过对接口的标准化使服务对任何异构平台或用户提供服务；服务请求者无须知道服务的运行环境，编程语言以及消息的传输路径等。

SOA的概念 – SOA的特征



- ✓ 服务构件粗粒度，传统构件细粒度居多
- ✓ 服务构件的接口是标准的，主要是WSDL接口，传统构件常以具体API形式出现
- ✓ 服务构件的实现与语言无关，传统构件绑定某种特定语言
- ✓ 服务构件可以通过构件容器提供QoS的服务，传统构件完全由程序代码直接控制

以下关于SOA的特征说法正确的是()

- ☒ A SOA是一种松散耦合的服务体系结构
- ☒ B SOA模型具有标准化接口
- ☒ C SOA是一种粗粒度的服务体系结构
- ☐ D SOA是一种细粒度的服务体系结构

提交

17/57

❁ SOA的概念 – 服务构件与传统构件

- 服务构件体系结构（SCA）是基于SOA的思想描述服务之间组合和协作的规范，它描述用于使用SOA构建应用程序和系统的模型。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。

❁ SOA的概念 – 服务构件与传统构件

SCA服务构件	传统服务构件
粗粒度构件	细粒度构件居多
标准的接口，主要是WSDL	非标准的接口，具体的API形式
实现与语言无关	绑定某种特定的语言
通过构件容器提供QoS服务	QoS服务由代码控制

服务构件体系结构（SCA）是基于SOA的思想描述服务之间组合和协作的规范,以下关于SCA的叙述不正确的是:()

- ☐ A SCA主要是为了满足软件集成的需要而创建的体系结构
- ☐ B SCA将传统中间件编程从业务逻辑分离出来，可使开发人员集中精力编写业务逻辑代码。
- ☐ C 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
- ☒ D 服务构件可以通过构件容器提供QoS的服务，而传统构件无法提供QoS的服务



❁ SOA的概念 – SOA常见的设计原则

- 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

关于SOA的设计原则下列说法正确的是:()

A

服务定义必须长时间稳定，一旦公布不能随意修改

B

服务之间要依赖消息传递，通常消息量较大，交互频度较低

C

服务请求者只能看到服务的接口；实现技术，当前状态等对服务请求者不可见

D

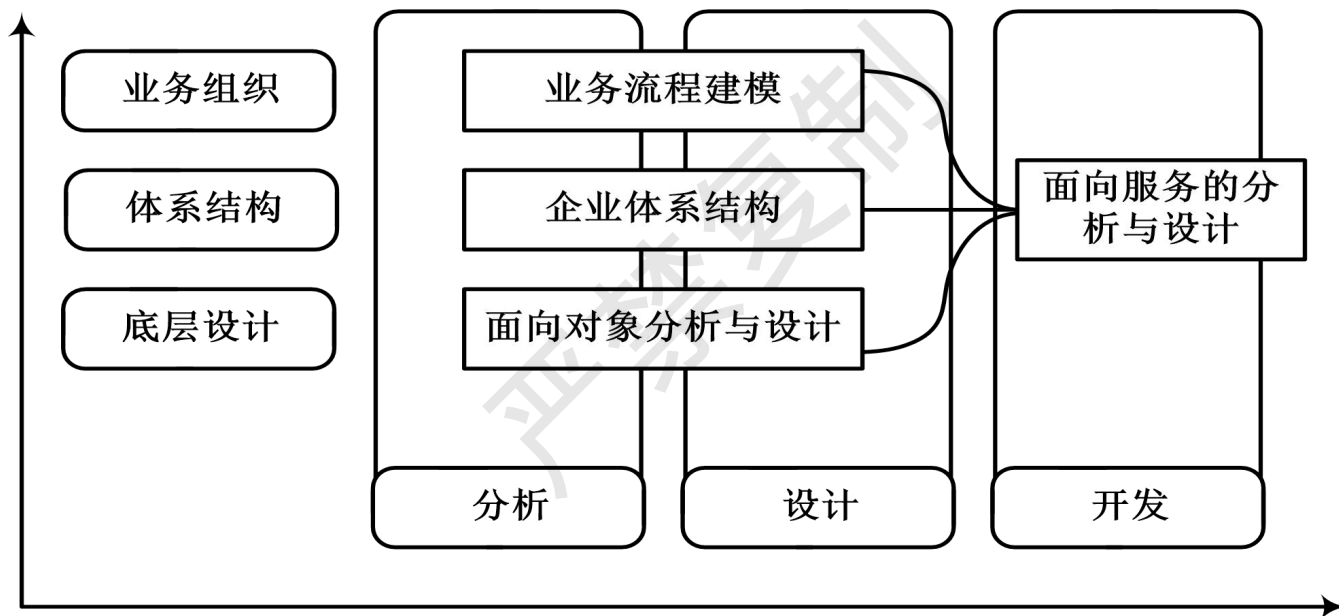
为确保服务规约的全面和明确，策略越来越重要

E

实现功能的实体完全是独立的，自主的，能够独立进行部署和版本控制，自我管理和恢复。

提交

❁ 面向服务的分析与设计



✿ 面向服务的分析与设计

1. 基础设计层

采用面向对象的分析与设计，对于设计已定义的服务中的底层类和构件结构，OO是一种很有价值的方法。OO的粒度级别集中在类级，对于业务服务建模来说，这样的抽象级别太低。

2. 体系结构层

体系结构层采用了EA的理论框架。应用EA框架和参考体系结构，以实现单独的解决方案之间体系结构的一致性。在SOA中，体系结构层必须以表示业务服务的逻辑构件为中心，并且集中于定义服务之间的接口和服务级协定。

3. 业务层

业务层采用了BPM规则。BPM是一个不完整的规则，其中有许多不同的形式、表示法和资源，其中应用较为广泛的是UML。SOA必须利用所有现有的BPM方法作为SOAD的起点，同时需要服务流程编排模型中用于驱动候选服务和它们的

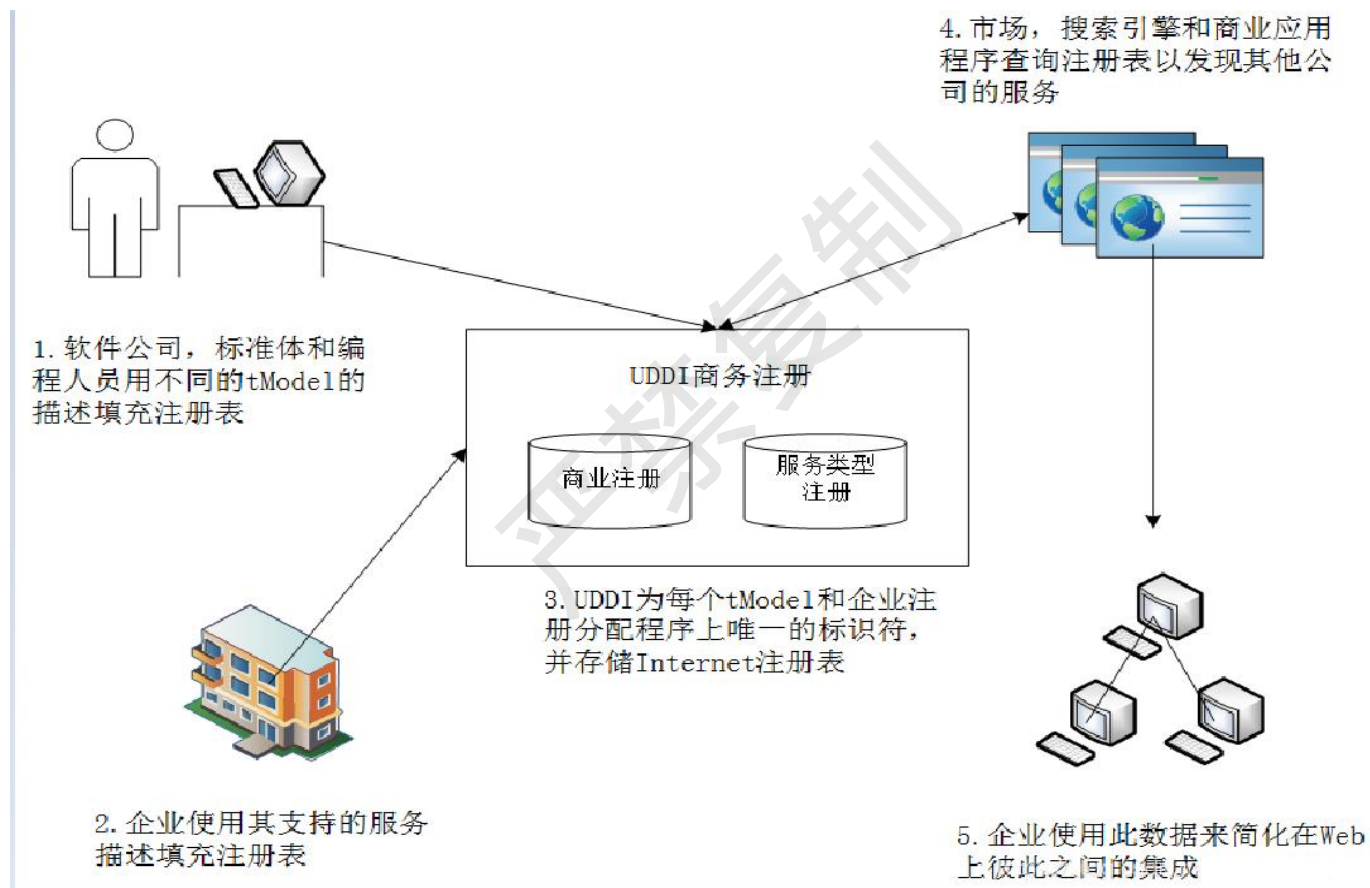
❁ SOA的关键技术*

服务栈

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

服务要以一种互操作的方式执行发布、发现和绑定。必须有一个包含每一层标准的服务栈。整个SOA技术系列被称为服务栈。

SOA的关键技术-发现服务层



关于UDDI的描述正确的是：()

A

帮助客户端应用程序解析远程服务的位置

B

通过UDDI，Web服务可以真正实现信息的“一次注册到处访问”

C

企业可通过UDDI提供的标准接口发布自己的服务，也可查询特定服务描述信息，并动态绑定到该服务上。

D

UDDI规范是服务的信息注册规范，以便被需要该服务的用户发现和使用。

E

UDDI是一种用于描述，发现集成WebService的技术

提交

SOA的关键技术-描述服务层

```

- <definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="http://d.ws.it.cn/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://schemas.xmlsoap.org/wsdl/" targetNamespace="http://d.ws.it.cn/" name="MyWebService">
- <types>
- <xsd:schema>
  <xsd:import namespace="http://d.ws.it.cn/" schemaLocation="http://192.168.10.1:6666/ws?xsd=1" />
  </xsd:schema>
</types>
- <message name="sayHello">
  <part name="parameters" element="tns:sayHello" />
</message>
- <message name="sayHelloResponse">
  <part name="parameters" element="tns:sayHelloResponse" />
</message>
- <portType name="MyWebService">
- <operation name="sayHello">
  <input message="tns:sayHello" />
  <output message="tns:sayHelloResponse" />
</operation>
</portType>
- <binding name="MyWebServicePortBinding" type="tns:MyWebService">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
- <operation name="sayHello">
  <soap:operation soapAction="" />
- <input>
  <soap:body use="literal" />
</input>
- <output>
  <soap:body use="literal" />
</output>
</operation>
</binding>
- <service name="MyWebServiceService">
- <port name="MyWebServicePort" binding="tns:MyWebServicePortBinding">
  <soap:address location="http://192.168.10.1:6666/ws" />
</port>
</service>
- <xs:schema xmlns:tns="http://d.ws.it.cn/" xmlns:xs="http://www.w3.org/2001/XMLSchema" targetNamespace="http://d.ws.it.cn/">
  <xs:element name="sayHello" type="tns:sayHello" />
  <xs:element name="sayHelloResponse" type="tns:sayHelloResponse" />
  <xs:complexType name="sayHello">
    <xs:sequence>
      <xs:element name="arg0" type="xs:string" minOccurs="0" />
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="sayHelloResponse">
    <xs:sequence>
      <xs:element name="return" type="xs:string" minOccurs="0" />
    </xs:sequence>
  </xs:complexType>
</xs:schema>

```

operation name: 方法的名称
input message: 说明有接受参数
output message: 有返回值
为了跨平台, 所有的wsdl文件描述都采用的是 schema

name: 真正实现此ws的名称 (不重要)
type: 实现此服务的真实类型
transport: 说明此服务soap服务
operation: 此服务可以提供被调用的方法

Service name: 指定ws服务的名称, 通过此名称获取ws服务
port name: 此服务下面提供的服务端口 (服务的类型get post soap1.1 soap1.2) 注意由于是通过jdk1.6生成的ws服务, 因此只支持soap1.1
address location: 服务的访问地址, 如果要看wsdl说明, 则在此地址后面添加 "?WSDL"

关于WSDL的描述正确的是:()

- ☒ **A** 描述服务层为客户端应用程序提供正确地与远程服务交互的描述信息，主要通过WSDL来实现
- ☒ **B** WSDL为服务提供者提供以XML格式描述服务请求的标准格式
- ☐ **C** 从WSDL文件中无法获取到服务的地址和端口信息
- ☒ **D** WSDL将网络服务描述为能够进行消息交换的通信端点集合，以表达一个服务能做什么，它的位置在哪，如何调用它等信息

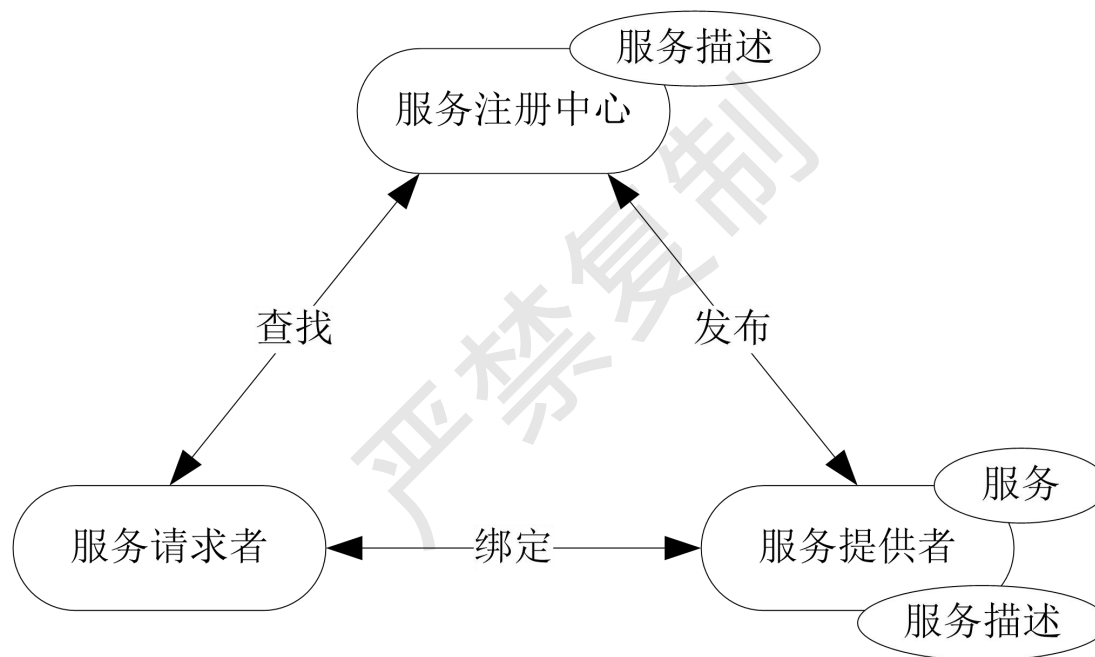
提交

以下关于SOA服务栈的描述正确的是:

- ☒ **A** 发现服务层主要用来帮助客户端应用程序解析远程服务的位置，通过UDDI来实现。
- ☐ **B** 描述服务层为客户端应用程序提供正确地与远程服务交互的描述信息，主要通过SOAP来实现
- ☒ **C** 描述服务层为客户端应用程序提供正确地与远程服务交互的描述信息，主要通过WSDL来实现
- ☒ **D** 消息格式层主要用来保证客户端应用程序和服务端在格式设置上保持一致，一般通过SOAP来实现。
- ☒ **E** 编码格式层为客户端与服务端提供一个标准的独立于平台的交换编码格式，一般通过XML来实现
- ☒ **F** 传输协议层为客户端和服务端提供交互的网络通信协议，如HTTP、TCP/IP、SMTP等

提交

SOA的实现方法 – Web服务*



webservice即web服务，是一种跨编程语言，跨操作系统平台的远程调用技术。

SOA的实现方法 – WEB服务

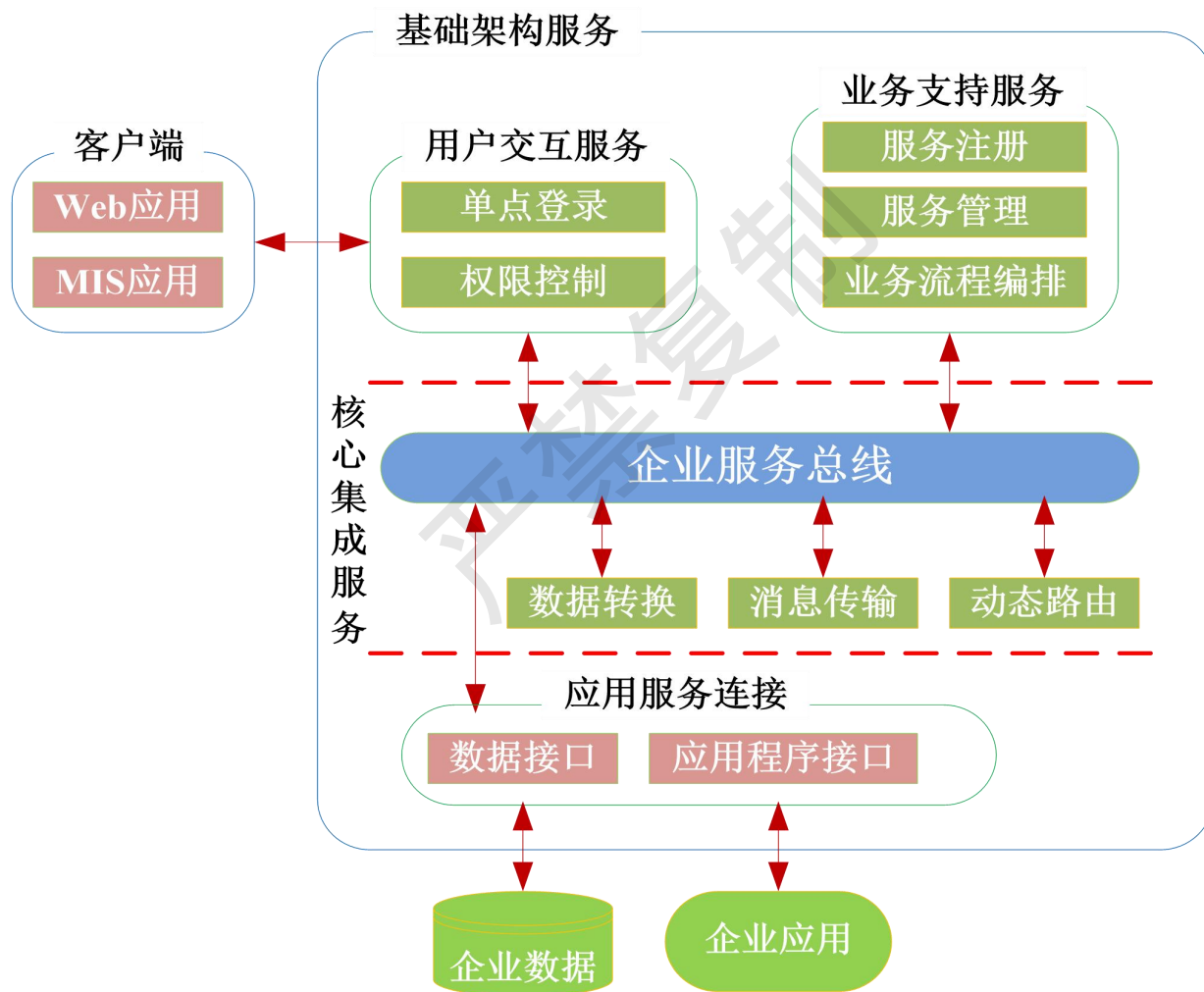
web Service技术实现的SOA应用层次

底层传输层	负责消息传输机制	Http, SMTP, JMS
服务通信协议层	描述并定义服务之间进行消息传递所需技术标准	SOAP, REST
服务描述层	用统一的方式描述服务的接口和消息的交换方式	WSDL
服务层	包装遗留系统并通过发布的WSDL接口被定位和调用	
业务流程层	支持服务发现和服务调用, 将业务流程从服务的底层抽取出来	
服务注册层	通过WSDL发布服务, 支持服务请求者查找所需的服务	UDDI

❁ 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

关于使用web服务实现SOA的方法，下列说法正确的是：

A

服务提供者是服务的所有者，该角色负责定义并实现服务。使用WSDL对服务进行详细、准确、规范的描述，并将该描述发布到服务注册中心，供服务请求者查找并绑定使用

B

服务请求者是服务的使用者,虽然服务面向的是程序，但程序的最终使用者仍然是用户。从体系结构的角度看，服务请求者是查找、绑定并调用服务，或服务进行交互的应用程序

C

服务注册中心是连接服务提供者和服务请求者的纽带，服务提供者在此发布他们的服务描述，服务请求者在服务注册中心查找他们需要的服务。

D

服务请求者与服务提供者是通过注册中心相互调用进行消息传递的

E

Web Service模型中的操作包括发布、查找和绑定，这些操作可以单次或反复出现。

F

Web Service技术实现的SOA应用层次分为：底层传输层，服务通信协议层，服务描述层，服务层，业务流程层，服务注册层

提交

关于服务注册表，下列说法错误的是:()

- ☐ A 服务注册表的功能有服务注册，服务位置和服务绑定
- ☐ B 服务注册是指服务提供者向服务注册表发布服务的功能
- ☐ C 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
- ☒ D 服务绑定是指服务提供者利用查找到的服务合约来开发代码

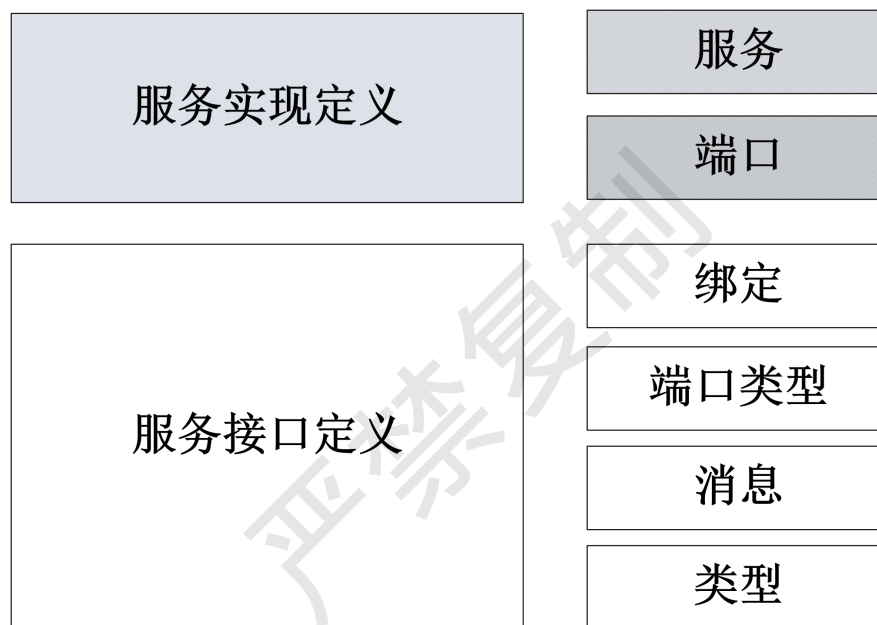
提交

关于SOA的实现方法，下来说法正确的是：

- ☐ A WEB服务实现方法中，服务请求者，服务提供者，服务注册中心都是必须的。
- ☒ B 服务注册表主要在SOA设计时使用
- ☒ C ESB消除了服务请求者和服务提供者之间的直接连接，是两者进一步解耦
- ☒ D 用WEB服务实现方法中，服务描述层主要使用WSDL技术
- ☒ E ESB具有灵活，可扩展的优势

提交

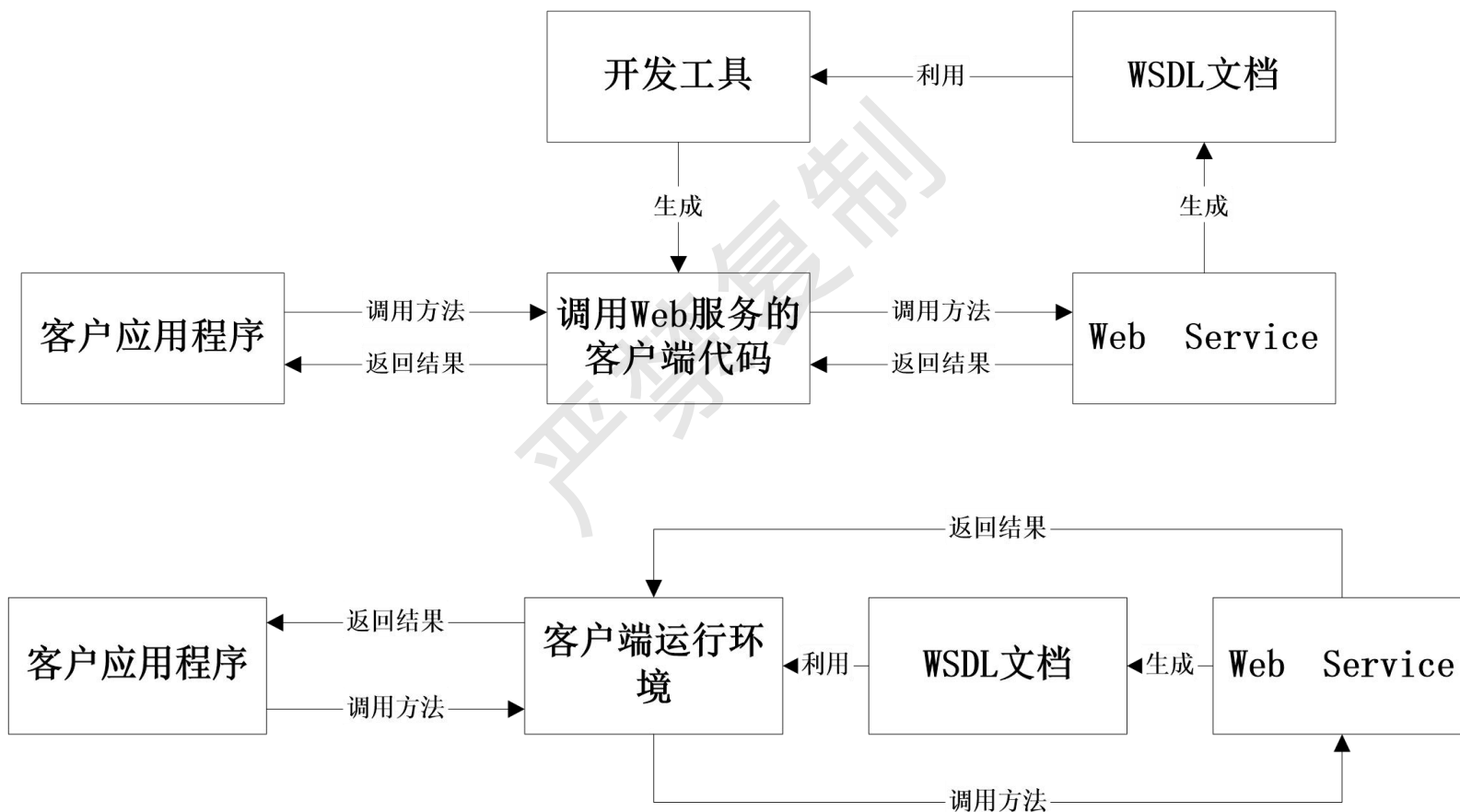
❁ WSDL – 基本服务描述



服务接口定义就是一种抽象的、可重用的定义，行业标准组织可以使用这种抽象的定义来规定一些标准的服务类型，服务实现者可以根据这些标准定义来实现具体的服务。比如：Java中定义的一个抽象接口可以有多个实现。

服务实现定义描述了给定服务提供者如何实现特定的服务接口。服务实现定义中包含服务和端口描述。端口描述的是一个服务访问入口的部署细节，包括 URL，消息调用模式等。

❁ WSDL – 使用WSDL文档



WSDL – 文档结构

```

- <definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="http://d.ws.it.cn/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://schemas.xmlsoap.org/wsdl/" targetNamespace="http://d.ws.it.cn/" name="MyWebService">
- <types>
- <xsd:schema>
  <xsd:import namespace="http://d.ws.it.cn/" schemaLocation="http://192.168.10.1:6666/ws?xsd=1" />
  </xsd:schema>
</types>
- <message name="sayHello">
  <part name="parameters" element="tns:sayHello" />
</message>
- <message name="sayHelloResponse">
  <part name="parameters" element="tns:sayHelloResponse" />
</message>
- <portType name="MyWebService">
  <operation name="sayHello">
    <input message="tns:sayHello" />
    <output message="tns:sayHelloResponse" />
  </operation>
</portType>
- <binding name="MyWebServicePortBinding" type="tns:MyWebService">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
  <operation name="sayHello">
    <soap:operation soapAction="" />
    <input>
      <soap:body use="literal" />
    </input>
    <output>
      <soap:body use="literal" />
    </output>
  </operation>
</binding>
- <service name="MyWebServiceService">
  <port name="MyWebServicePort" binding="tns:MyWebServicePortBinding">
    <soap:address location="http://192.168.10.1:6666/ws" />
  </port>
</service>
- <xs:schema xmlns:tns="http://d.ws.it.cn/" xmlns:xs="http://www.w3.org/2001/XMLSchema"
  <xs:element name="sayHello" type="tns:sayHello" />
  <xs:element name="sayHelloResponse" type="tns:sayHelloResponse" />
  <xs:complexType name="sayHello">
    <xs:sequence>
      <xs:element name="arg0" type="xs:string" minOccurs="0" />
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="sayHelloResponse">
    <xs:sequence>
      <xs:element name="return" type="xs:string" minOccurs="0" />
    </xs:sequence>
  </xs:complexType>
</xs:schema>

```

operation name: 方法的名称
 input message: 说明有接受参数
 output message: 有返回值
 为了跨平台, 所有的wsdl文件描述都采用的是 schema

name: 真正实现此ws的名称 (不重要)
 type: 实现此服务的真实类型
 transport: 说明此服务soap服务
 operation: 此服务可以提供被调用的方法

Service name: 指定ws服务的名称, 通过此名称获取ws服务
 port name: 此服务下面提供的服务端口 (服务的类型get post soap1.1 soap1.2) 注意由于是通过jdk1.6生成的ws服务, 因此只支持soap1.1
 address location: 服务的访问地址, 如果要看wsdl说明, 则在此地址后面添加 "?WSDL"

❁ 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。

关于WSDL对服务描述下列说法正确的是:

- ☐ A WSDL描述的重点是服务，包括服务的实现和接口的定义。
- ☒ B 服务接口的定义是一种抽象的，可重用的定义，抽象接口定义有助于提高系统的扩展性
- ☒ C 服务实现定义了服务提供者如何实现特定的接口，包含服务和端口的描述
- ☐ D 服务描述了访问入口的部署细节，一个服务只有一个访问入口
- ☒ E 端口描述的是一个服务访问入口的部署细节，包括URL，消息调用细节等

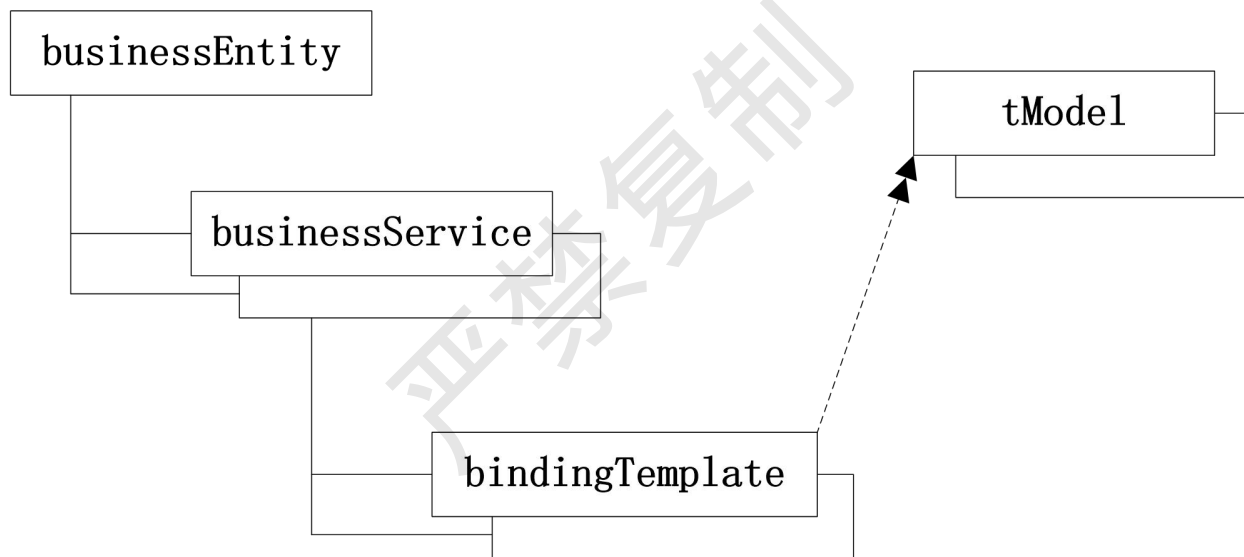
提交



❁ 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



❁ 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服务的主要联系方式。
- 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

- 通过UDDI注册中心来定位并获得businessEntity。
- 进一步获得详细的businessService信息，或是得到一个完整的businessEntity结构。
- 从businessService中选择一个bindingTemplate待以后使用。
- 根据web服务在tMode中提供的引用地址编写程序
- 使用已经保存下来的bindingTemplate信息，调用Web Service。

关于UDDI下列说法正确的是

- ☒ A UDDI是一种用于描述，发现，集成Web Service的技术
- ☒ B UDDI数据模型用于描述企业和服务的XMLSchema
- ☒ C UDDI API是一组基于SOAP的，用于查找或发布UDDI数据的方法
- ☒ D UDDI数据模型中的tMode是一个描述了技术规范的模式，可以是外部技术规范的引用地址
- ☒ E UDDI模型中businessEntity对企业基本信息进行了描述。
- ☐ F UDDI模型中businessService描述了如何访问一个特定的webservice，并给出了访问的地址

提交

❁ SOAP – 概述

SOAP即简单对象访问协议，Simple Object Access Protocol，用户交换XML编码信息的轻量级协议。

- 1、SOAP作为一个基于XML语言的协议用于网上传输数据。
- 2、SOAP = 在HTTP的基础上+XML数据
- 3、SOAP 是基于HTTP的。



❁ SOAP – 概述(组成)

- **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(Envelope):** 封装是顶层元素，在SOAP消息中必须出现。(必须的)
- **SOAP头(Header):**向SOAP消息中添加关于这条SOAP消息的某些要素 (feature) 的机制。(可选的)
- **SOAP体(Body):**是包含消息的最终接收者想要的信息的容器。(必须的)

❁ SOAP – 消息封装实例

```
POST /regcodeService.asmx HTTP/1.1
Host:.weixin.fscut.com
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "https://weixin.fscut.com/MakeRegCode"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
    <SessionHeader xmlns="https://weixin.fscut.com">
      <LangID>int</LangID>
      <SvcVersion>int</SvcVersion>
      <OEMCode>int</OEMCode>
      <SessionID>string</SessionID>
      <OperFrom>string</OperFrom>
    </SessionHeader>
  </soap:Header>
  <soap:Body>
    <MakeRegCode xmlns="https://weixin.fscut.com">
      <Serial>string</Serial>
      <ExpireDate>string</ExpireDate>
      <Flags>int</Flags>
      <CustomName>string</CustomName>
      <Mobile>string</Mobile>
      <Comment>string</Comment>
    </MakeRegCode>
  </soap:Body>
</soap:Envelope>
```

❁ SOAP – 编码规则

- **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 – 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。

```
<element>
  <element name="age" type="int"/>
  <element name="address" type="string"/>
</element>
<age>45</age>
<address>乌鲁木齐</address>
```

- 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。

```
<element name="Book">
  <complexType>
    <element name="author" type="xsd:string"/>
    <element name="preface" type="xsd:string"/>
    <element name="intro" type="xsd:string"/>
  </complexType>
</e:Book>
```

```
<e:Book>
  <author>Henry Ford</author>
  <preface>Prefatory text</preface>
  <intro>This is a book.</intro>
</e:Book>
```

❁ SOAP – 编码规则

- 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。

```
<return xmlns:ns2="http://www.w3.org/2001/09/soap-encoding"
      xsi:type="ns2:Array" ns2:arrayType="xsd:double[2]">
  <item xsi:type="xsd:double">54.99</item>
  <item xsi:type="xsd:double">19.99</item>
</return>
```

- 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。

```
<return xmlns:ns2="urn:examples" xsi:type="ns2:product">
  <name xsi:type="xsd:string">Red Hat Linux</name>
  <price xsi:type="xsd:double">54.99</price>
  <description xsi:type="xsd:string">Red Hat Linux Operating System</description>
  <SKU xsi:type="xsd:string">A358185</SKU>
</return>
```


❁ 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
- 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

❁ SOAP – 在HTTP中使用SOAP

- 把SOAP绑定到HTTP使得SOAP的请求和应答参数可以包含在HTTP请求和应答中。
- 在HTTP消息中包含SOAP消息时，应用程序必须使用媒体类型 “text/xml”。

```
POST /regcodeService.asmx HTTP/1.1
Host: weixin.fscut.com
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "https://weixin.fscut.com/MakeRegCode"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
    <SessionHeader xmlns="https://weixin.fscut.com">
      <LangID>int</LangID>
      <SvcVersion>int</SvcVersion>
```

关于SOAP的说法正确的是:

- ☒ A SOAP组成主要包括：封装结构，编码规则，RPC表示和绑定
- ☐ B SOAP编码规则中的值有简单值和复合值，数组属于复合值，结构属于简单值
- ☒ C 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。
- ☐ D 把SOAP绑定到HTTP上，使得SOAP的请求和应答参数并不在HTTP请求和应答中

提交



❁ REST

REST(Representational State Transfer, 表述性状态转移)是一种只使用 HTTP 和 XML 进行基于 Web 通信的技术,可以降低开发的复杂性,提高系统的可伸缩性。它的简单性和缺少严格配置文件的特性,使它与 SOAP 很好地隔离开来,REST 从根本上来说只支持几个操作(POST、GET、PUT 和 DELETE),这些操作适用于所有的消息。REST 提出了如下一些设计概念和准则:

1. 网络上的所有事物都被抽象为资源(resource);
2. 每个资源对应一个唯一的资源标识(resource identifier);
3. 通过通用的连接器接口(generic connector interface)对资源进行操作;
4. 对资源的各种操作不会改变资源标识;
5. 所有的操作都是无状态的(stateless)。

对于当今最常见的网络应用来说,resource identifier 是 url, generic connector interface 是 HTTP, 第 4 条准则就是我们常说的 url 不变性。这些概念中的 resource 最容易使人产生误解。resource 所指的并不是数据,而是数据+特定的表现形式(representation),这也是为什么 REST 的全名是 Representational State Transfer 的原因。举个例子来说,“本月卖得最好的 10 本书”和“你最喜欢的 10 本书”在数据上可能有重叠(有一本书即卖得好,你又喜欢),甚至完全相同。但是它们的 representation 不同,因此是不同的 resource。