一些概念：

1.Web服务是实现SOA的核心技术，但SOA并不等同于Web服务。Web服务是一套技术体系，可以用来构建应用解决方案，解决特定的消息通信和应用集成问题。而SOA是一种软件架构，不局限于某种技术组合（如Web服务），它超越技术范畴，甚至可以用来组织公司。

2.企业服务总线（ESB）是SOA基础架构（basic frastruct）的关键组件，是一种消息代理架构，管理消息通信、服务交互等等。

3.WSDL：Web服务描述语言，基于XML，但它才是Web服务的核心。因为它描述Web服务提供的操作（服务能力）以及这些操作接收和返回的参数。WSDL包含的信息：服务做什么，应该如何使用它们，它们在哪里。也就是说提供者和调用者都需要参考WSDL，从这个意义上来说WSDL是核心。

一、为何需要构建服务生态系统？什么是服务生态系统中的垂直服务和水平服务？它们有何联系和区别？试举例加以说明

1.服务生态系统（参考商业生态系统的答案）

一方面：面向服务的快速发展导致单个组织无法独立提供全套服务，提供的有限服务也无法被广泛运用；已存在的服务并不能很好的被发现和调用，也导致了大量冗余服务

另一方面：原先的服务系统是复杂、脆弱、特殊的，从上层业务看，无法灵活应对实际业务的变更；从底层实现看，也无法及时应对底层技术的更新、或者新增的功能

因此构建服务生态系统，运用面向服务的分析和设计原则，使得产生的服务具有良好的可发现性和可复用性，同时能灵活应对业务领域和技术领域的变更。

2.垂直服务

· 单独面向一个客户，提供系列功能的服务

· 从消费者的角度说，垂直服务可以被同时、独立地使用，分为纯IT服务和IT-enabled服务

3.水平服务

· 用于构建垂直服务、可重用的、跨行业的公共服务

· 分为公共业务服务和IT服务

4.联系与区别

· 垂直服务和水平服务都是通过服务系统来实现业务服务

· 水平服务是功能相关的，简单且相对稳定，一般由IT专家开发

· 垂直服务是流程相关的，复杂且易变更，需要领域专家参与开发

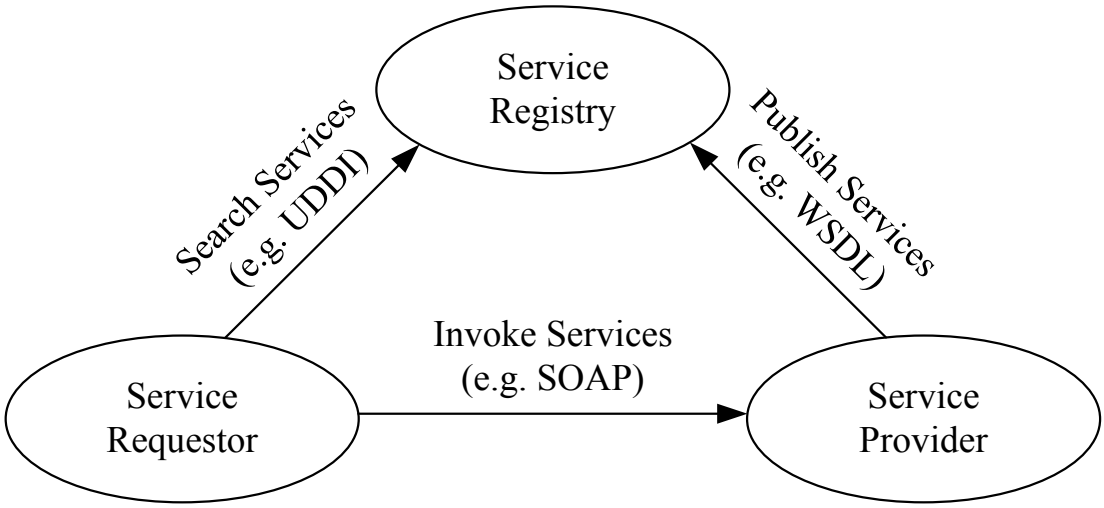
· 垂直服务实际上是一系列水平服务的封装，将上下文无关的水平服务根据特定的业务流程进行编排，最后打包为一个解决特定问题的垂直服务

举例：

学校的借书服务、打印成绩单服务是垂直服务，都需要进行身份认证，身份认证就是一个水平服务；借书服务和打印成绩单服务除了身份认证还包含其他的服务，就是将简单的水平服务构建成能实现业务目标或流程的垂直服务

二、试描述SOA三角操作模型，并对比传统的端到端服务调用模式，阐述其IT优势和商业优势

1.SOA三角操作模型



`

1）三种角色：

· 服务提供者：发布自己的服务，并且对服务请求进行响应

· 服务请求者：利用服务注册查找所需要的服务，然后使用该服务

· 服务注册：注册已经发布的服务，对其进行分类，并提供搜索服务

2）三个操作：

· 发布：为了使服务可访问，需要发布服务描述以使服务使用者可以发现它

· 查找：服务请求者查询服务注册来找到满足其标准的服务

· 绑定：检索到服务描述后，服务使用者继续根据服务描述中的信息调用服务

对比传统端到端服务调用模式：

降低服务调用者和服务提供者之间的耦合；调用者只需要向服务注册请求调用，服务注册会在满足条件的服务提供方中选一个交给调用者，调用者不需要担心提供者是否会停止服务；提供者需要变更自己的服务时，服务注册就换一个实现给调用者。

2.IT优势

1）松耦合，消除假依赖——复用

· 语言、平台和厂商中立

· 消除时间依赖

· 消除访问地址依赖

· 消除访问协议依赖

2）服务间接寻址 —— 灵活

3.商业优势

· 保护企业投资，提升现有IT资源的作用，促进IT资源的复用

· 提高企业灵敏度

· 支持企业外包管理模式

三、XML的结构（树状结构）？与原先的文本/json/二进制相比有什么好处?

XML形成的是树状结构，一个XML文档有唯一的element作为根元素，该元素是所有其他元素的父元素，同时所有的元素均可拥有子元素。

优点：XML文档的内容和结构完全分离，基于这个特性可以实现服务的功能管理和流程管理彻底分离；

互操作性强，纯文本文件可以方便地在不同的系统之间通信；

规范统一，XML有统一的标准语法，可以跨平台、跨系统

可扩展性强，可以根据XML的基本语法，进一步限定在特殊环境下，使用的XML文档格式

支持多种编码，XML包含所使用的编码，方便多语言系统对数据进行处理

四、试描述soap包的结构，并结合该结构，阐述soap处理模型（无需准确给出相关命名空间和元素名称）

1.soap包结构

· soap包本质是一个xml文档，包含下列元素：

1）Envelope元素

· 必需元素，根元素，标识此XML文档为一条 SOAP 消息

· 可以包含命名空间和声明额外的属性。如果出现额外属性，则必须使用命名空间修饰

2）Header 元素

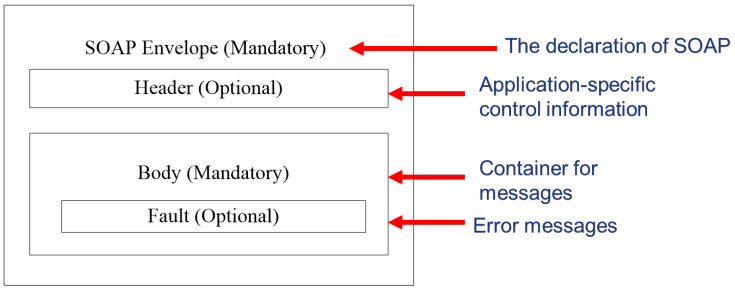
· 可选元素，有关 SOAP 消息的应用程序专用信息（比如认证、支付等）

3）Body 元素

· 必需元素，包含所有的调用和响应信息

4）Fault 元素

· 可选元素，提供有关在处理此消息所发生错误的信息



2.处理模型

1）用XML打包请求

· 将接口名作为根节点

· 方法和参数作为结点

2）将请求发给服务器

· 不创建自己的TCP/IP信息，利用HTTP

· 将请求封装成HTTP POST请求格式发出

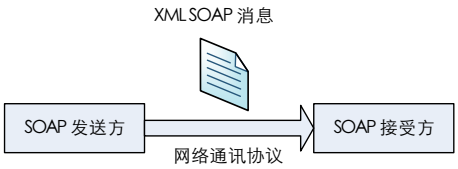
3）服务器接收到请求，解码XML，处理请求，以XML格式返回响应

· 与请求比较，方法的结点名字变为请求的方法名后缀Response

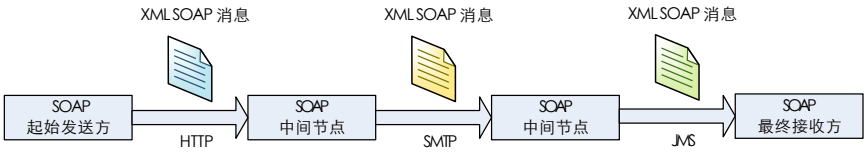
· 客户程序知道自己调用了哪个方法，根据方法名后缀Response寻找调用方法的返回值

3.处理模式

1）简单模式



2）复杂模式



4.交互模式

1）远程过程调用模式

· 一种同步请求/响应交互的方法

· 发出请求后会一直等待响应

2）面向文档模式

· 一种异步交互的方法

· 发送一个完整XML文档，然后等待通知，结果会在处理后发送回来

补充：SOAP为什么被设计成两块？实际SOAP如何利用这两个信道?

被设计成header和body两块，一方面分离了控制信息和主要数据，让信息结构更加清晰；另一方面，在复杂模式中，header中的头块信息可以和中间节点进行角色上的转变。

body是必须的部分，内嵌的XML是完成当前任务的主要数据；而如果是一些附加的、用来协助完成的控制信息，则放在header中

五、试结合WSDL文件的结构，说明WSDL文档记载了服务的哪些信息，并阐述为何需要将WSDL文件分为抽象部分和具体部分

1.结构（WSDL 2.0）

以description元素为根结点

import、include：拼装不同部门/组织定义的文档，形成完整的WSDL语意

1）抽象部分

· Types：使用到的数据结构或者叫数据格式规范，独立于语言和平台

· Interface：operation的集合即服务能力的集合，描述服务能力

operation：input、output、infault、outfault

2）具体部分

· Binding：特定端口类型的具体协议和数据格式规范的绑定。

· Service：对服务整体的抽象，包含若干个endpoint

endpoint：将绑定与当前地址关联

2.服务的哪些信息

功能 也叫服务能力，即调用某一个特定的操作，这个操作能被用来完成面向服务的一项功能

数据结构 通信消息中使用的数据结构

协议的绑定 即确定消息如何在具体的网络协议上传输

总的来说就是服务做什么、如何使用服务以及服务在哪里

3.分开原因

一方面：

· 抽象部分以独立于平台和语言的方式定义服务的逻辑意义，并不包含任何随机器或语言而变的元素，使不同的服务调用者都能调用实现

· 具体部分则包含了随网站而异的东西，由各实现方依据自己的情况定制

· 在保证可复用的同时又不局限各实现方的个性化定制

另一方面：

· 定义服务与实现服务可能是不同的部门或者组织，那么抽象部分与具体部分的WSDL也应当由各自的部门定义、持有与管理（C9高校做让选课无缝衔接，教育部持有抽象部分，各高校持有具体部分）

六、以UDDI和WSIL为例，分别阐述集中式和分布式服务发布/查询的过程，并对比这两种方法

1.集中式发布

S1：软件公司和标准组织向服务注册发布规范,即tModel 。

S2：公司完成服务的开发，注册关于业务及提供的服务的描述。

S3：UDDI 服务注册给每个实体指定一个唯一的标识符，从而能时刻了解所有实体的情况。

2.集中式查询

查询者使用UDDI提供的API，根据业务信息、服务信息或服务类别等搜索相关的服务

UDDI找到相应服务的WSDL并生成SOAP发送给查询者，查询者根据WSDL中描述的接口等信息调用服务

3.分布式发布

S1：将服务描述为一个XML文档发布在Web服务器上

S2：使用WSIL从现有的服务描述文档生成引用

S3：引用指针连接到下一个服务，既可以是发布在UDDI注册中心的服务，也可以是另一个WSIL

这样不断的连接就最终形成WSIL的超链

4.分布式查询

通常使用构建好的查询工具查询；遍历WSIL超链，进行查找

S1：确定起始WSIL的位置

S2：执行查找

S3：显示该WSIL中的链接表

S4：选择一个链接打开，如果打开的WSIL还包含其他链接，则继续追踪

S5：重复步骤3和4，直到找到目标信息

3.对比

主要区别是代价和复杂性

UDDI：适用于希望得到最大复用、得到最大访问范围的服务；

UDDI使用黄页机制，维护大量服务，代价大，但好处是分门别类，便于管理和共享

WSIL：适用于当前提供服务的企业事先已经规划完毕，（我把它写成了一个目录，这个目录挂载在某个网站，让别人可以直接下载）根据超链的方式，对外提供这个企业内部所有分门别类的服务；复用性稍差，好处是不需要付出额外的代价

七、试对比面向服务泛型和面向对象泛型（提示：方法论、抽象和写作层次、代码共享和复用、动态绑定和重新组合、重组、组件通讯和接口，系统维护、可靠性、软件拥有）

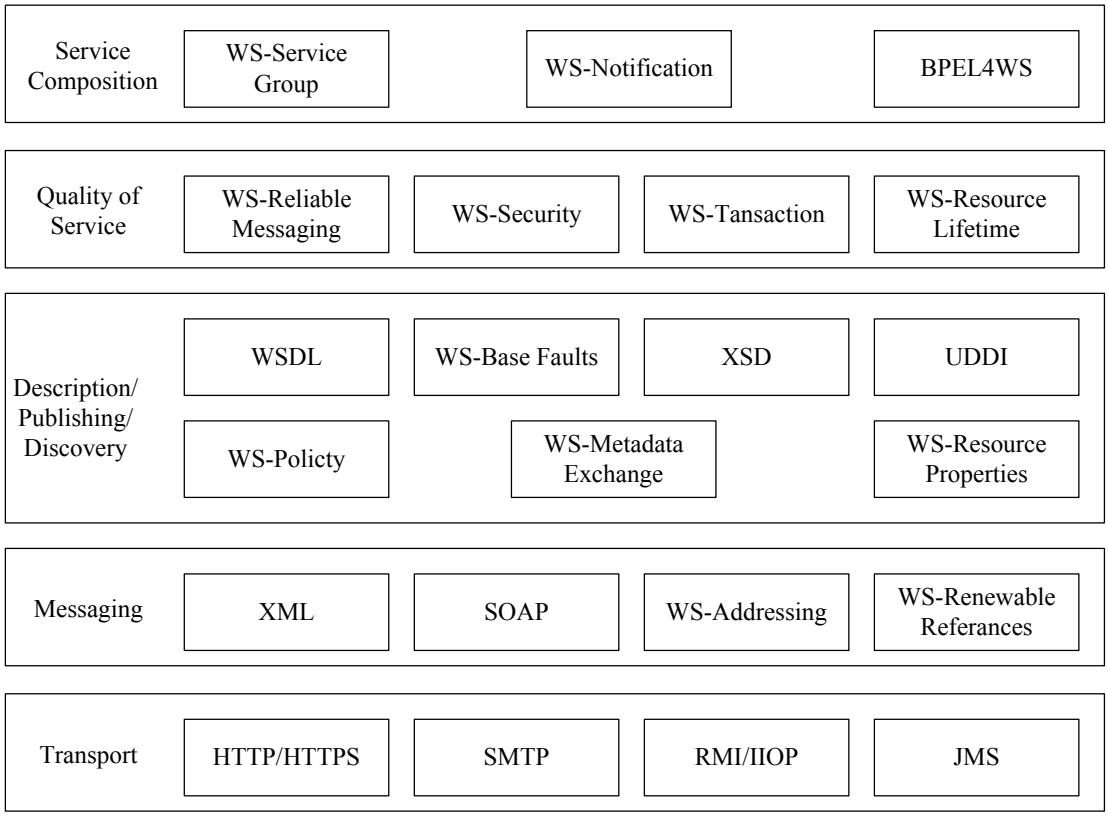
特点	面向对象的计算	面向服务计算
方法论	通过定义紧耦合的类来进行应用开发 应用架构为基于继承关系的层次式架构 从构造函数——通过类或模型——到系统设计	通过定义松耦合的服务来进行应用开发，并将服务组装成可执行的应用 从系统模型到服务模块，从服务抽象定义到服务实现绑定 通过搜索获得可用的服务实现
抽象和协作层次	往往由一个团队来负责应用的开发，并负责整个生命周期 开发者必须了解应用领域知识和编程	开发任务由三个独立方承担：应用程序开发者，服务提供方和服务代理 其中，应用程序开发者需要了解应用逻辑，但不需要了解具体的服务是如何实现的 服务提供者需要编程能力，但不必了解使用服务的应用

特点	面向对象的计算	面向服务计算
代码共享和复用	代码复用通过类成员的继承和库函数加以实现。其中库函数在编译时引入，且往往是平台相关的	代码在服务层次复用。服务使用标准的结构，并发布在Internet库中。服务是平台无关的，且能够被查找并远程调用。服务代理支持系统的服务共享
动态绑定和重新组合	在运行时将名称和方法进行关联。方法必须在应用部署前链接到可执行的代码	在运行时将服务调用和服务进行绑定。可以在应用部署后，再进行服务选定。这一特色使得应用可以在运行时重组
重组	多在设计时决定导入的组件	可以动态改变应用系统中服务的组合关系，以及服务定义与服务实现之间的绑定关系，即实现动态地添加、修改、删除各个服务节点
组件通讯和接口	与平台和语言有关，例如C++程序难以直接和Java程序通信	与平台和语言无关。组件间通过标准协议通信，如XML，WSDL和SOAP

特点	面向对象的计算	面向服务计算
系统维护	用户需要时常升级软件，且在执行升级时，应用必须停止	通过互联网升级系统，因为服务多运行在远程服务器上，用户通过互联网进行访问。维护对用户透明
可靠性	在设计时决定可靠性的方法	对于服务提供者，每个服务相对简单，更加可靠。对于应用程序存在多个满足同一需求的服务，可用过将故障服务的节点断开并重新绑定到备选服务节点上，获得不间断的应用系统
软件拥有	软件作为产品销售，为用户所拥有	软件存在并执行于独立的服务提供商的设备上，用户按照每次对服务使用付费，而不是按照软件产品付费

综合题

一、试结合相关协议和框架，描述一个web service从创建开始到被最终服务消费者调用的全过程中对服务的建模、查询和调用的全过程



1.服务的建模

XML(Namespace、XML Schema)、SOAP、WSDL

XML定义了Web服务中的消息交换格式，使用XML Schema定义不同的数据结构，引入Namespace使得XML、XML Schema中的元素和属性全球唯一且全球共享；

SOAP提供了一种标准的方法，使得运行在不同平台、使用不同的技术和编程语言的应用程序可以互相进行通信，服务的发布、查找、调用，都通过SOAP传递XML消息

WSDL对服务能力、服务中使用的数据结构以及传输绑定给出定义和描述；提供了一种基于XML的标准接口定义语言/服务能力定义语言，用以在服务的提供者/调用者/服务注册之间，交换必要的有关Web Service的信息

对于大多数服务，用以上三个协议和框架可以完成建模；对于一些更为复杂的服务，如复合服务或者是带有非功能性需求的服务，还需要用到其他协议和框架完成建模。BPEL定义多个服务间如何交互和合作，从而将一组现有的服务根据业务流程构建起来，实现业务服务。WS-Policy可以实现一非功能性需求，如信息加密，权限验证等。

建模完成后，服务提供者通过UDDI或者WSIL将服务发布出去。其中，UDDI利用分页机制，让服务得到最大可能的复用和共享范围；WSIL使用树形连接结构，适用于企业既定的服务。

2.服务的查询

消费者程序发送SOAP消息给服务注册，描述自己需要的服务；服务注册查询注册表，通过WSDL服务合约找到一系列符合条件的服务；服务注册将查询到的WSDL通过SOAP发送给消费者程序，让消费者程序从中选择可用的服务；或者服务注册自动化筛选出当前最符合消费者程序要求的服务，通知消费者程序。

3.服务的调用

消费者程序根据WSDL中提供的服务位置进行调用；其中，消费者和提供者基于WSDL中约定的接口进行消息的发送和接收；另一方面，当前服务可能同时被多个消费者程序使用，创建了一系列服务实例，WS-Addressing提供了相应的机制，确保服务消费者能在实例池中找到特定的实例并与之通信。另一方面，由于创建的实例是有状态的，利用WSRF对状态数据进行存取，进行状态管理，提高资源利用率。

二、以电信企业为应用背景，举例描述服务分析和服务设计的过程。并结合面向服务的设计原则（标准化服务合约、服务松散耦合、服务抽象、服务可复用性、服务自治、服务无状态性、服务可发现性、服务可组合性），讨论“schema集中化”“合约集中化”“逻辑集中化”在设计过程中的应用。

举例：电信企业有订购、退订套餐，账单结算等基本业务流程

1.服务分析流程

面向服务分析的目标是讨论需要构建哪些服务，每个服务应该封装哪些逻辑。分析的核心是业务服务。

1.进行文档化的需求描述，**定义流程自动化需求**，作为服务候选建模的依据；由于电信企业发展比较完善，可以直接使用之前的需求文档分析；
2.对**现有的自动化系统进行分析、识别**；分析企业正在使用的系统具有的功能；
3.对**服务候选建模**，识别服务操作候选，并将其分组。

在面向服务分析流程中，需要考虑服务可复用性、服务自治和服务可发现性；
可复用性：在服务建模中，需要：精化已有的服务能力候选，使其更加一般化和可复用；定义额外的服务能力候选，这些能力是在构成服务建模过程的基础的业务流程自动化所需之外的
自治：对已有自动化系统收集得到的信息，会影响服务系统所能达到的自治级别；比如根据信息决定保留遗留系统，那么达到共享自治，独立开发的可能达到逻辑自治或完全自治
可发现性：从服务生命周期开始，尤其是在产生服务操作候选时，需要以统一的方式，记录所有元数据；在服务建模过程中，业务和技术专家需要一起合作，建立服务候选

2.服务设计流程
服务设计过程，是从服务候选（逻辑）派生出具体的服务设计（物理），然后装配到实现业务流程的抽象组合中。
1.组合SOA：选择编排、业务、应用服务层中的哪些进行实现，定义核心的SOA标准，选择SOA扩展（WS-*协议）
2.根据业务层级，分别设计以实体为核心的业务服务，应用服务，以任务为核心的业务服务
3.设计面向服务业务过程，组合服务构建出业务流程

“schema集中化”：
传统的做法是在订购服务、退订服务中使用不同的套餐数据结构，而按照标准化服务合约，所有使用的数据结构都应该被单独定义、管理，与具体的操作流程无关。
采用Schema集中化的设计模式，将电信企业划分为多个分离的领域（部门），每个领域都可以被独立地进行标准化和治理，每个领域定义和管理自己的schema，作为整个服务系统的基本数据结构；在不同的服务中，使用这些schema，避免了频繁且不必要的数据转换；在必要的情况下，可以利用这些schema定义新的数据结构

“合约集中化”
为了保证服务松散耦合，避免消极耦合，采用合约集中化，将对服务的访问严格控制在合约内：
1.所有的合约应该被集中管理，拥有一致的设计原则和设计目标；
2.在服务生态系统中，任何情况都不可以绕开合约去访问具体内容

服务抽象&服务可发现性设计服务暴露的信息
服务抽象：技术信息、功能、程序逻辑、服务质量抽象
服务抽象出来并对外界可用的信息就是服务合约，服务合约的设计标准会影响到其他

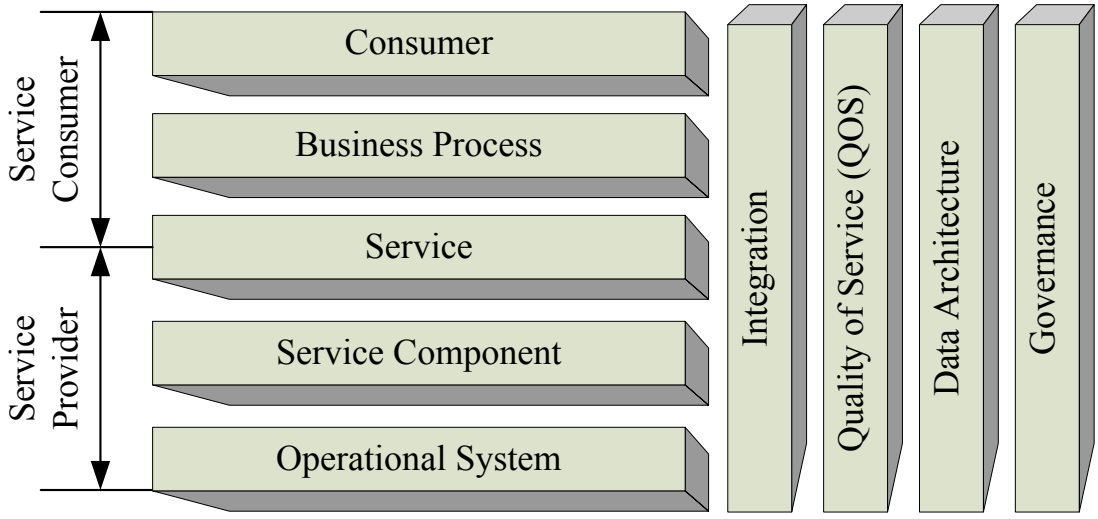
“逻辑集中化”
为了实现服务可复用性，让消费者程序只调用指定的服务，要建立服务库存，在规范的服务库存中，每个服务代表来一个独特的功能域，这就要求服务边界之间没有重叠。

设置专家管理服务库存，应用开发人员不能直接往服务库存中增改需要的服务，只能请求当前服务库存管理人员进行审查，作出恰当的决策。
同时，服务可发现性是实现服务可复用的前提，服务自治是可复用服务潜在高性能和并行使用的保证；无状态性能提高服务的可用性

逻辑集中化和合约集中化

- 集中化与Web 服务
  - Web服务的WSDL 、XML schema 和WS-Policy定义必须正确地表达访问一个正式逻辑体(按逻辑集中化)的一个正式访问点(根据合约集中化)
- 实现逻辑集中化的挑战
  - 实际企业中很难实现逻辑集中化
  - 采用领域库存(Domain Inventory)模式描述企业的子集，并在领域库存中实现逻辑集中化
- 逻辑集中化要求：设计者在建立消费者程序并需要特定类型的信息处理时，这些消费者程序只调用指定的服务
  - 逻辑集中化不会描述怎样访问这一逻辑
- 合约集中化要求：设计者建立消费者程序时，这些消费者程序仅通过已发布的合约访问一个服务
  - 合约集中化不需要指定为了何种目的应该访问什么服务

补充：SOA参考架构模型（SOA-RA）



- Horizontal layers implements functional requirements
- The five Horizontal layers can be divided to two layers:
  - Services Provider tier (back-end)
    - Operational System
    - Services Component
    - Services
  - Services consumer tier (front-end)
    - Services
    - Business Process
    - Consumer

- Vertical layers provides system-support facilities and enablement
  - Integration layer
    - Is a key enabler for an SOA solution as it provides the capability to mediate, rout, and transform service request between service requestors and service providers
    - ESB is one example
  - Quality of Service (QoS) layer
    - Provides solution-level QoS management in various aspects, such as availability, reliability, security, and safety.
    - This layer does not focus on service-level QoS control; instead it concentrates on providing a mechanism to support, track, monitor and manage solution-level QoS control
  - Data Architecture layer
    - Provides a unified representation and enablement framework that integrates with domain-specific data architecture to facilitate value chain integration (i.e., integration of services developed by different parties)
    - Such as:
      - The Shared Information and Data model (SID) in eTOM of telecommunication industry
      - The RosettaNet Technical Dictionary and RosettaNet Business Dictionary defined by RosettaNet for electronics industry
  - Governance layer
    - Provides design guidance to ensure the proper design of the SOA solution architecture
    - Typically, this layer helps to establish best practices or their reference, to establish principles of how to define SOA solutions in each layer, and to establish principles of how to monitor in running system and how to handle exceptions at runtime