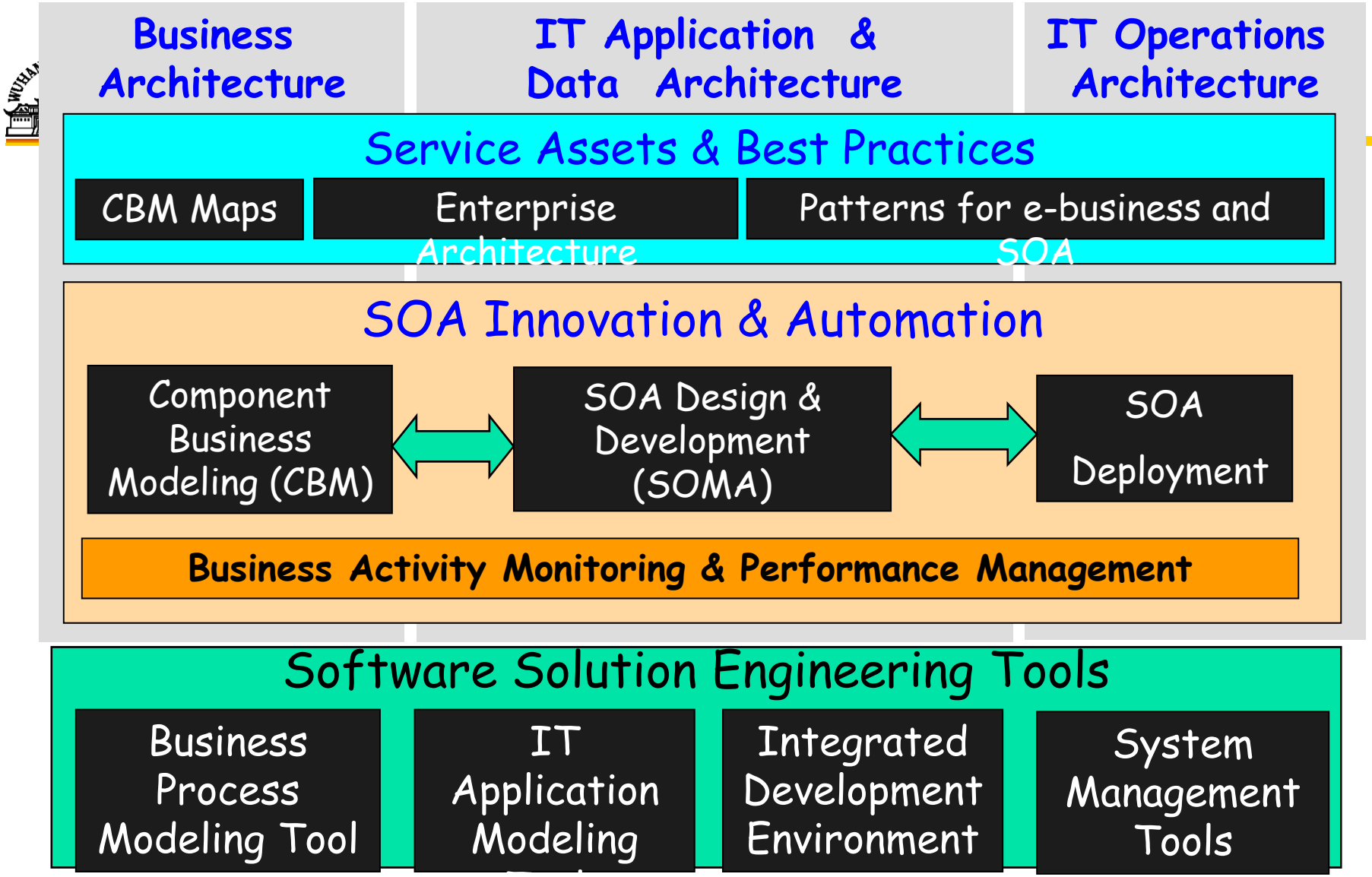
服务科学与管理

武汉大学国际软件学院

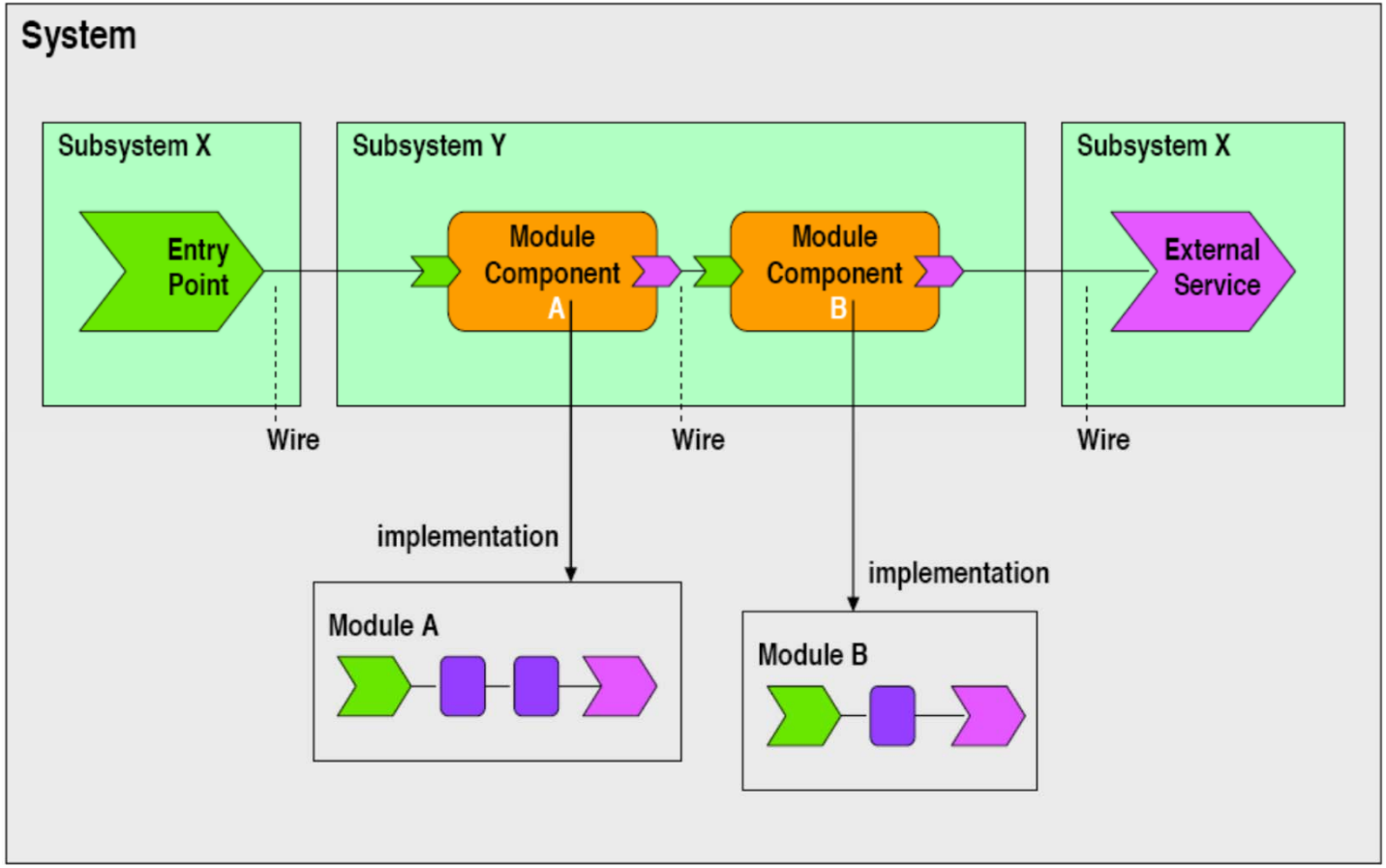
主要内容

- 第一章 从服务到服务科学
- 第二章 服务业务流程
- 第三章 服务工程
- 第四章 服务管理
- 第五章 IT服务管理

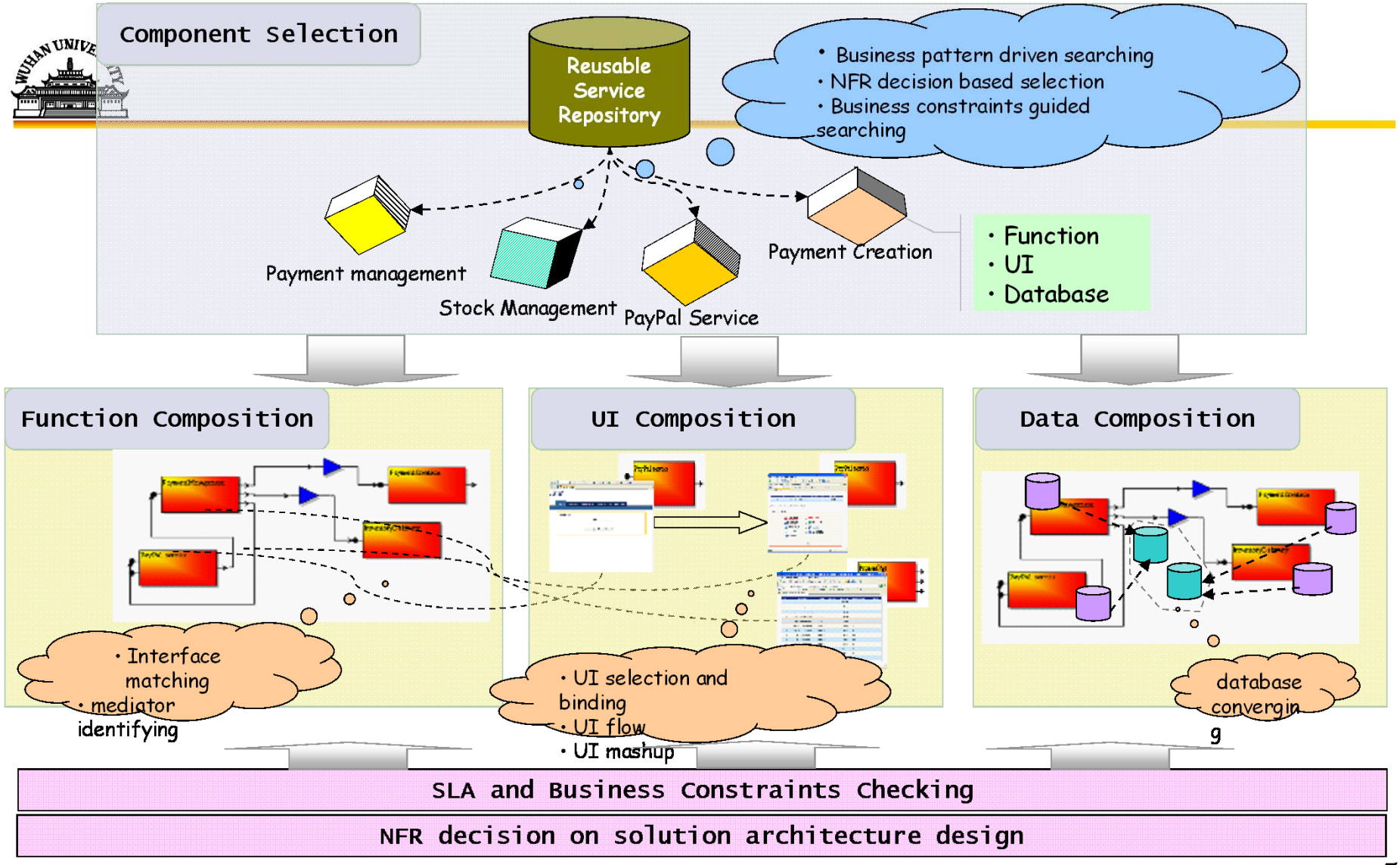
Service Software Engineering : Scope



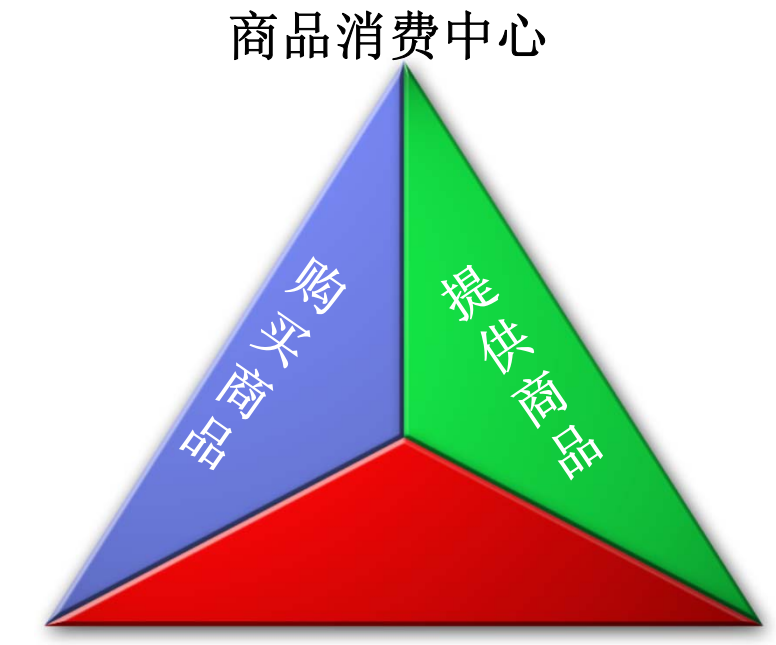
Service Component Architecture (SCA)

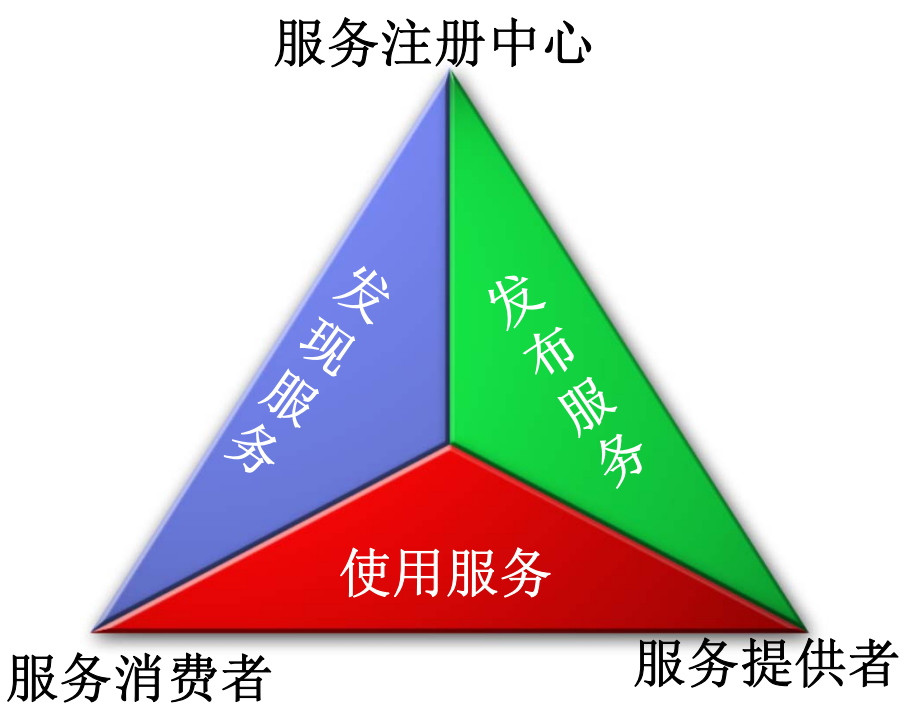


Intelligent Service Composer
- Enable non-experts to easily compose Service Offerings



商品消费vs.软件服务


商品消费中心
购买商品
提供商品
商品消费者
商品提供者


服务注册中心
发现服务
发布服务
使用服务
服务消费者
服务提供者

IBM SOA (Service Orientated Architechure)

•… a service?

•A repeatable business task – e.g., check customer credit; open new account

•… service orientation?

•A way of integrating your business as linked services and the outcomes that they bring

•… service oriented architecture (SOA)?

•An IT architectural style that supports service orientation

•… a composite application?

A set of related & integrated services that support a business process built on an SOA

7

第三章 服务工程

- **3.1 INTERNET环境下的WEB服务**
- **3.2 面向服务的架构**
- **3.3服务网格和云计算**

3.1 INTERNET环境下的WEB服务

- Web服务是一个集自包含、自描述和模块化为一体的应用，在Internet环境下发布和定位，通过Web调用实现从简单请求到复杂业务处理的功能。
  - 3.1.1 Web服务概述
  - 3.1.2 Web服务的体系结构及优势
  - 3.1.3 Web服务的协议架构
  - 3.1.4 Web服务的基本特点
  - 3.1.5 Web服务的技术体系总结
  - 3.1.6 Web服务的发现和制定

3.1.1 Web服务概述

- 业界跨入了务实的阶段，这一轮的电子商务发展中，技术完全是为商业服务的。
- 为了实施电子商务，无论自身的IT部门还是外包的解决方案提供商，其给出的实施计划都是应用正式运营前的。一旦应用被部署之后，由于商务环境和商务需求的不断改进和不断变化，这些电子商务应用不可避免地需要被修订、需要被更新，以符合新的电子商务流程。

3.1.1 Web服务概述

- 传统的解决方案是为每个需要的企业资源或外部资源编写连接代码，以使得应用得以运行。这些资源包括传统系统（legacy systems）和数据库、Web应用及Web资源。
- 传统解决方案的缺点：

1.代码很难再定制

2.由于每个应用都有其自己特有的基础架构，这些应用在部署、更改和维护上的代价都异常高昂。

3. 不能被方便地继承

4.不能随着企业商务的规模扩展而方便地实现应用的规模扩展

3.1.1 Web服务概述

- Web服务的使用将改变目前的开发模式和应用部署的费用规模。
- 各种Web服务构件实现了一定的电子商务功能，通过将各种电子商务的Web服务进行组合和集成以创建动态电子商务应用。
- Web服务能够统一地封装信息、行为、数据表现以及商务流程，而无需考虑应用所在的环境是使用何种系统和设备。

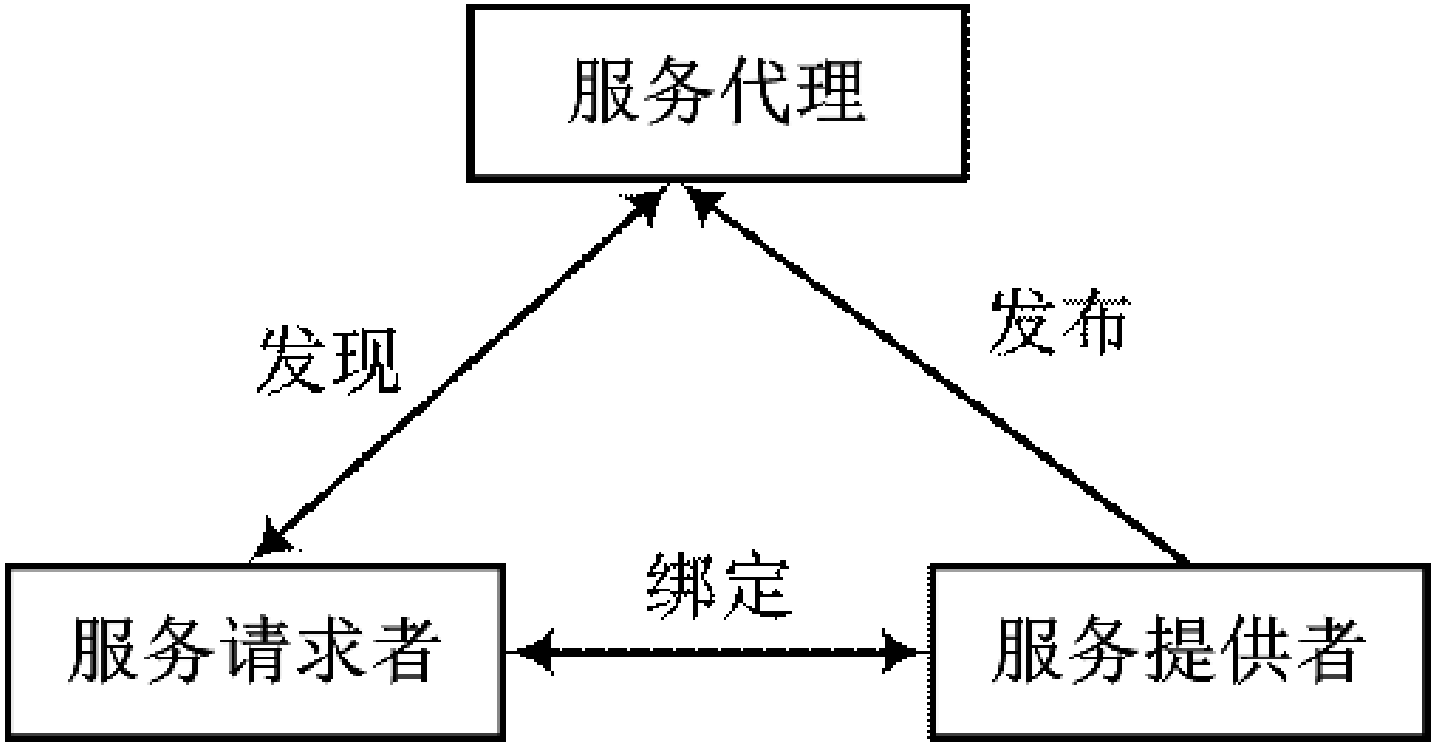
3.1.1 Web服务概述

- **Web服务作为一种新兴的Web应用模式，是一种崭新的分布式计算模型，是Web上数据和信息集成的有效机制。**
- **Web服务就像Web上的构件编程，开发人员通过调用Web应用编程接口，将Web服务集成进他们的应用程序，就像调用本地服务一样。**

3.1.1 Web服务概述

- Web服务通过WSDL来描述，通过SOAP作访问，在商业注册中心（UDDI）发布，从而使开发者和电子商务应用程序可以搜索并定位到该服务。在Web服务开发的过程中有三个清晰的角色定义，它们是服务的提供者（Service provider），服务的请求者（Service request）和服务的代理（Service broker），下图显示了它们之间的关系。

3.1.2 Web服务体系结构模型



3.1.2 Web服务体系结构优势

- 高度的通用性和易用性
- 完全的平台、语言独立性
- 高度的集成性
- 容易部署和发布

3.1.2 Web服务优势—高度通用性和易用性

- Web服务既然是一种部署在Web上的对象，自然具备对象的良好封装性，对于使用者而言，他能且仅能看到该对象提供的功能列表，而不必考虑web服务对象的内部组成，因此有易用性。
- Web服务对象内封装都是一些通用功能，因此也具有高度的复用性。

3.1.2 Web服务优势—完全的平台、语言独立性

- Web服务对象具有松散耦合的特性，这一特征也是源于对象/组件技术，当一个Web服务的实现发生变更的时候，调用者是不会感到这一点的，对于调用者来说，只要Web服务的调用界面不变，Web服务的实现任何变更对他们来说都是透明的，甚至是当Web服务的实现平台从J2EE迁移到了.NET或者是相反的迁移流程，用户都可以对此一无所知。其实现的核心在于使用XML/SOAP作为消息交换协议，也就是说web服务因此具有语言的独立性。
- 作为Web服务，其协约必须使用开放的标准协议（比如HTTP、SMTP等）进行描述、传输和交换。这些标准协议应该完全免费，以便由任意平台都能够实现。一般而言，绝大多数规范将最终有W3C或OASIS作为最终版本的发布方和维护方，因此web服务也拥有了平台独立性。

3.1.2 Web服务优势—高度可集成性

- 由于**Web**服务采取简单的、易理解的标准**Web**协议作为组件界面描述和协同描述规范，完全屏蔽了不同软件平台的差异，无论是**CORBA**、**DCOM**还是**EJB**都可以通过这一种标准的协议进行互操作，实现了在当前环境下最高的可集成性。

3.1.3 Web服务的协议架构（自顶向下）

服务发现层	**UDDI**
服务发布层	**UDDI**
服务描述层	**WSDL**
消息传输层	**SOAP**
数据表现协议层	**XML**
网络传输层	**HTTP,FTP,MQ,IIOP**

3.1.3 UDDI

- Universal Description, Discovery and Integration
  - http://www.uddi.org
- UDDI 为以下几个点创建了一个平台无关的, 开放框架的入口:
  - 描述服务
  - 发现业务
  - 集成业务服务
- UDDI实际使用的情况比预测的少很多

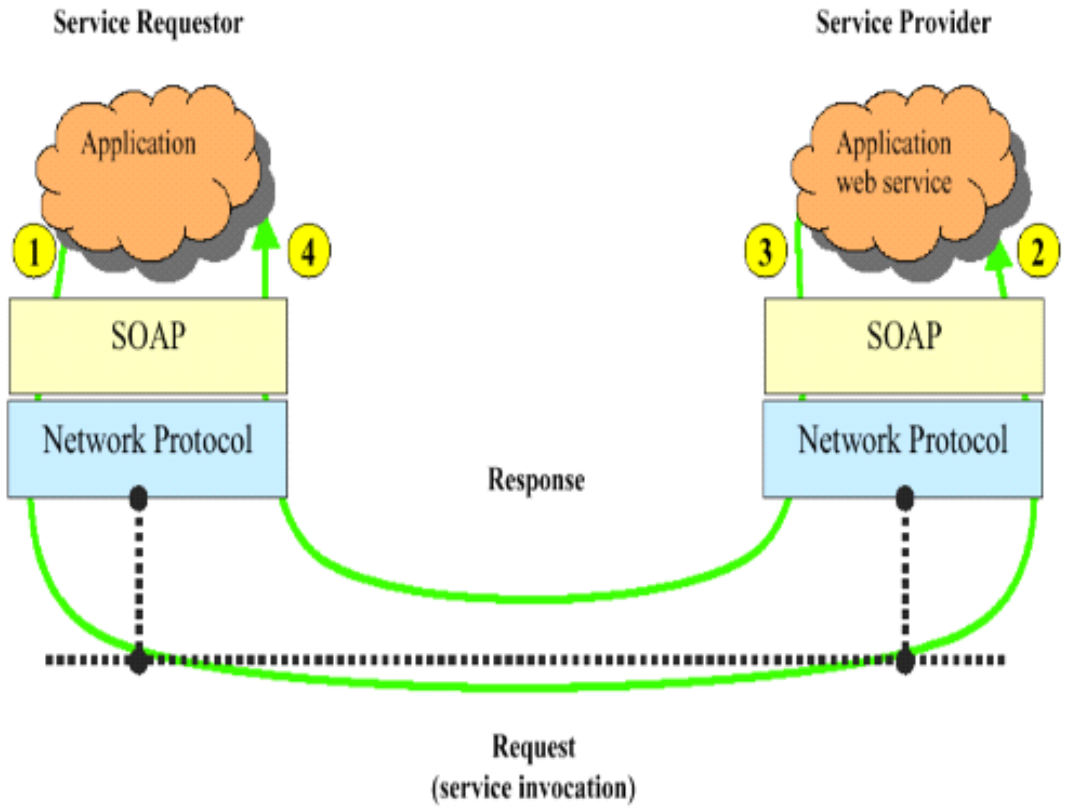
3.1.3 WSDL

- Web Services Definition Language
  - http://www.w3.org/TR/wsdl/
- WSDL是一种基于XML描述web服务的语言，所描述的东西主要有
  - 这个服务要干什么(description)
  - 怎样使用它(方法签名)
  - 在哪里可以找到这项服务
- WSDL不依赖于底层的协议

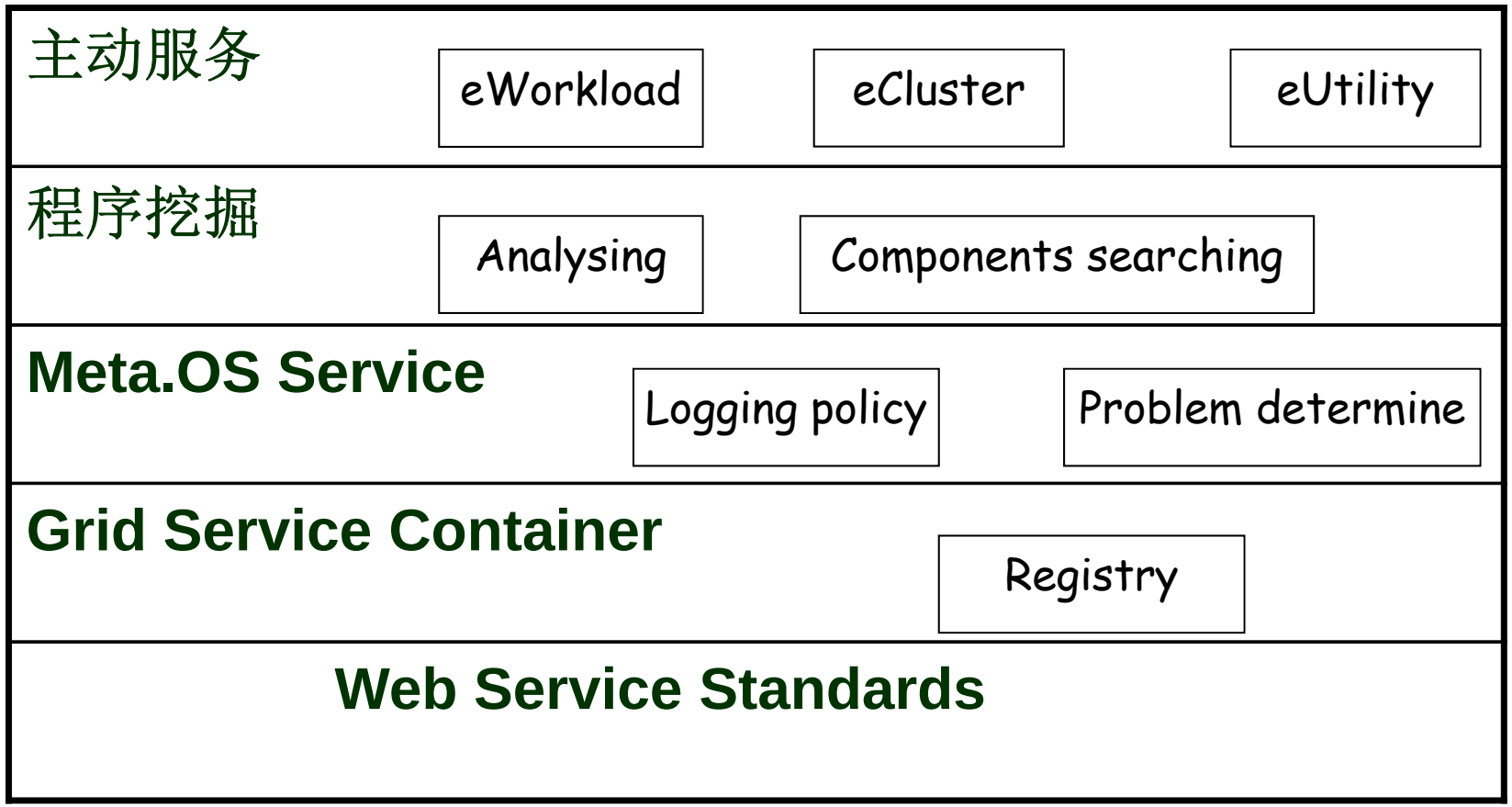
3.1.3 SOAP

- Simple Object Access Protocol
  - http://www.w3c.org/TR/SOAP/
- SOAP是一个轻量级用于分布式环境信息交换的协议
- 两种使用SOAP的不同方法:
  - 利用xml的灵活性 以及 可扩展性封装 远程调用。
  - 发布文本信息不封装消息

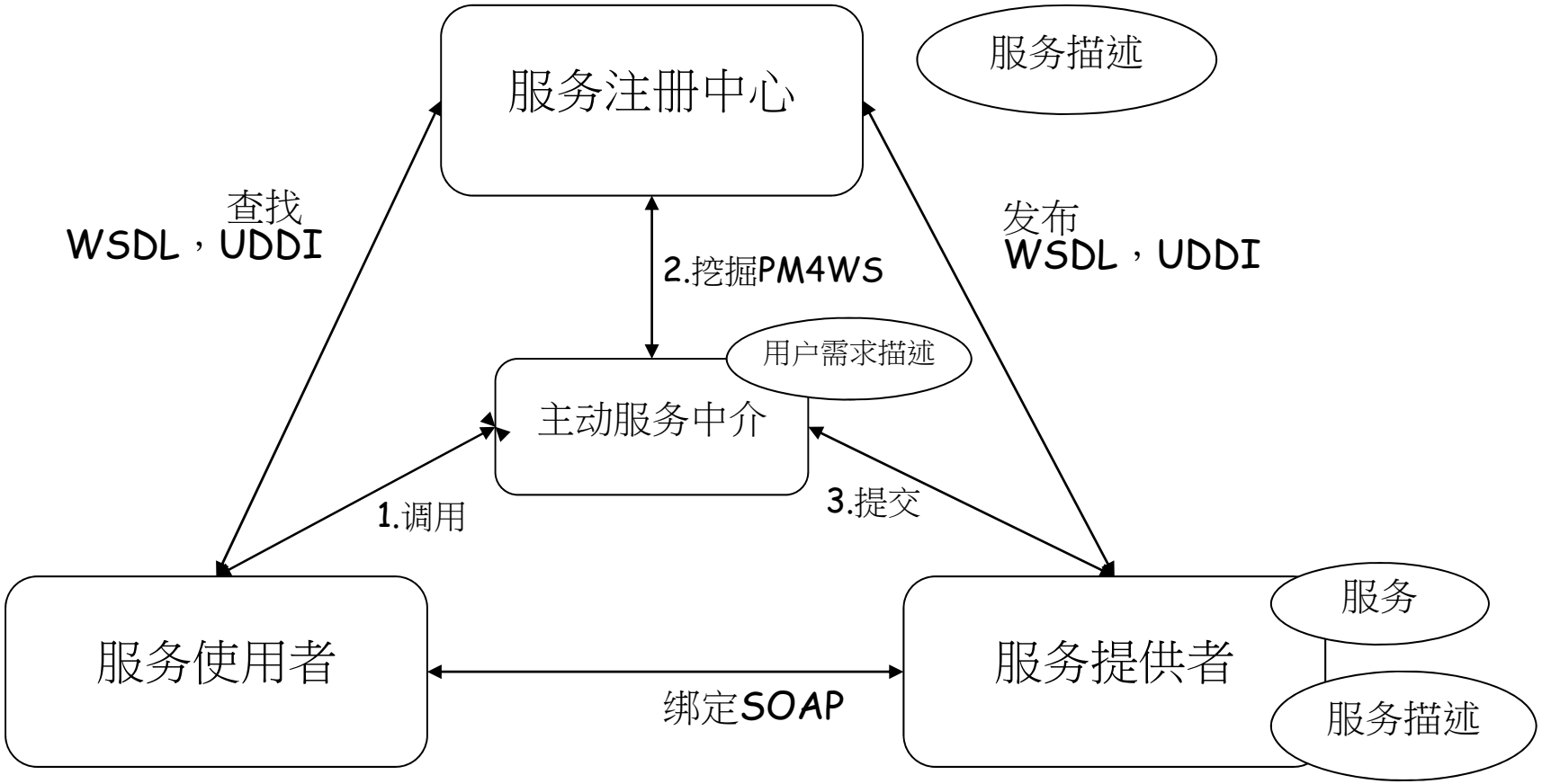
3.1.3 一个SOAP交换信息的例子



3.1.3 开放网络服务架构示例



3.1.3 主动服务模型


服务注册中心
服务描述
查找
WSDL，UDDI
2.挖掘PM4WS
发布
WSDL，UDDI
用户需求描述
主动服务中介
1.调用
3.提交
服务使用者
服务提供者
服务
服务描述
绑定SOAP

3.1.4 Web服务的基本特点

- 从服务平台看，**Web**服务是一个集部署、发现、事务、安全、认证等基本功能为一体的服务平台，是通过标准网络协议和数据格式进行发布、定位和调用的模块化的应用逻辑。
- 从开发人员看，**Web**服务是一组**Internet**上基于标准化**XML**（**eXtensible Markup Language**）的应用程序，被封装后存在于**Web**服务器上，对外提供一个**API(Application Programming Interface)**。

3.1.4 基本特点

- 从外部用户看，**Web**服务是一种部署在**Web**上的对象/组件，具备完好的封装性、松散耦合、自包含、互操作、动态、独立于实现技术、构建于成熟技术、高度可集成、使用标准协议等特征。
- 从实施对象看，把资源、计算能力提供给用户，需要以服务的形式完成。

3.1.5 Web服务的技术体系总结

- 一个Web服务被分为数据层（Data Layer）、数据访问层(Data Access Layer)、业务层(Business Layer)、业务面(Business Facade)和监听者(Listener)五个逻辑层。
- 在Web服务中，通信是基于Internet协议（例如HTTP、SMTP、FTP等）互相传递XML消息的通信协议规范，描述采用一种基于XML的语言描述和定义接口与绑定，发布和发现把Web服务提交到注册中心而让用户通过中介发现Web服务。

3.1.6 Web服务的发现和定制

- 首先，服务发现功能在**Internet**上搜寻已有的程序和数据。若满足用户需求功能的**Web**服务，则调用需求分析功能分解用户的服务需求，进入服务定制过程。
- 其次，需求分析功能在用户参与下准确定位用户需要的服务。
- 第三，把服务组装方案提交到执行用户具体服务的环境，对组装链接的计算资源进行验证与一致性测试。

3.2 面向服务的架构

- 3.2.1 基本概念
- 3.2.2 主要特征
- 3.2.3 基本模型
- 3.2.4 服务级别
- 3.2.5 面向服务的分析和设计

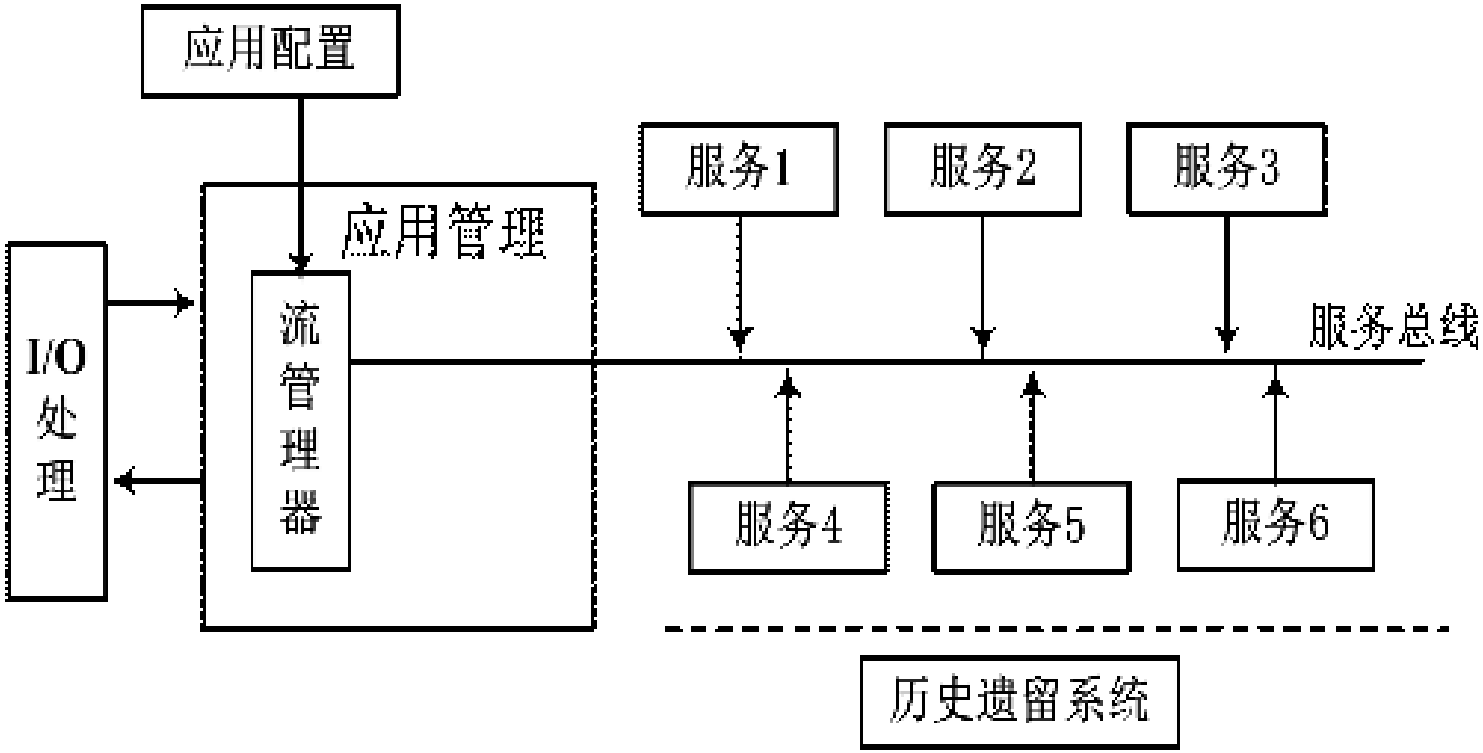
3.2.1 基本概念

- 面向服务的架构（**Service Oriented Architecture，SOA**）是一种面向服务组织计算资源、“抽象、松耦合和粗粒度”的协同架构，强调以服务为基础的资源共享和重用。
- 从概念看，**SOA**中有操作、服务和业务流程三个主要的抽象级别。
- 从实践看，**SOA**的重点是服务而不是对象或组件。
- 从生命周期看，**SOA**贯穿**IT**系统项目规划、系统分析和设计、系统实施、系统部署和维护、整个过程的监控和管理等，实现**SOA**是在建模、组装、部署、管理、控制等五个阶段中重复迭代的过程。

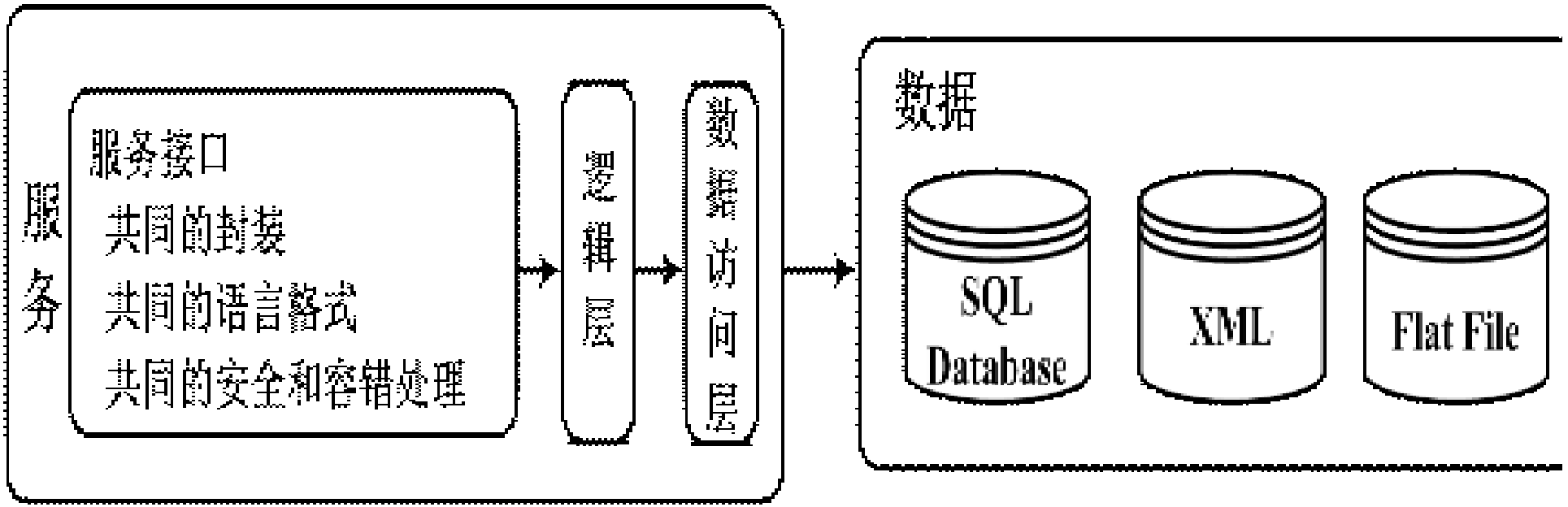
3.2.1 基本概念

- SOA为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，可以以定义好的顺序调用这些服务来形成业务流程。
- SOA为客户端/服务器的软件设计方法，一项应用由软件服务和软件服务使用者组成，SOA与大多数通用的客户端/服务器模型不同之处，在于它着重强调软件构件的松散耦合，并使用独立的标准接口。

面向服务体系结构模型



单个服务内部结构


服务
服务接口
共同的封装
共同的语言格式
共同的安全和容错处理
逻辑层
数据访问层
数据
SQL Database
XML
Flat File

3.2.2 主要特征

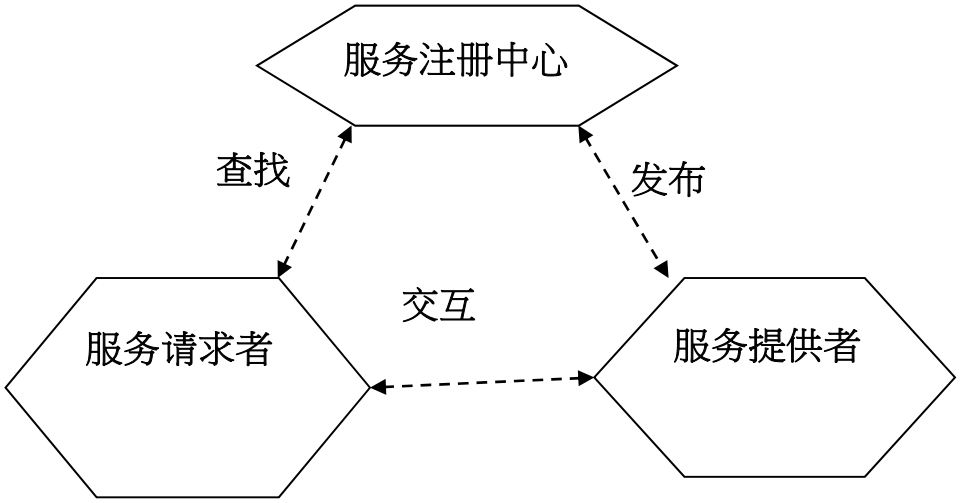
- SOA面向服务需求，根据Internet融合业务和IT设施，开放标准接口，分布式部署、组合和使用松耦合的粗粒度应用组件，按需动态构建应用程序，尽可能重用原有代码，协同利用基于不同技术的异地资源。
  - 首先，融合IT系统和业务流程，组件化协同应变。
  - 其次，松耦合架构，降低人为依赖。
  - 第三，屏蔽服务细节，跨平台跨系统。
  - 第四，SOA使用基于文本而非二进制的消息传递方式。

3.2.2 主要特征

面向服务的架构（SOA）	分布式组件架构
面向流程	面向功能
设计目的是为了适应变化	设计目的是为了实现需求
交互式和重用性开发	开发周期长
业务为中心	成本为中心
服务协调	应用阻塞
敏捷的和松耦合的	紧密耦合
异构技术	同构技术
面向消息	面向对象
独立于实施细节	需深入了解实施细节

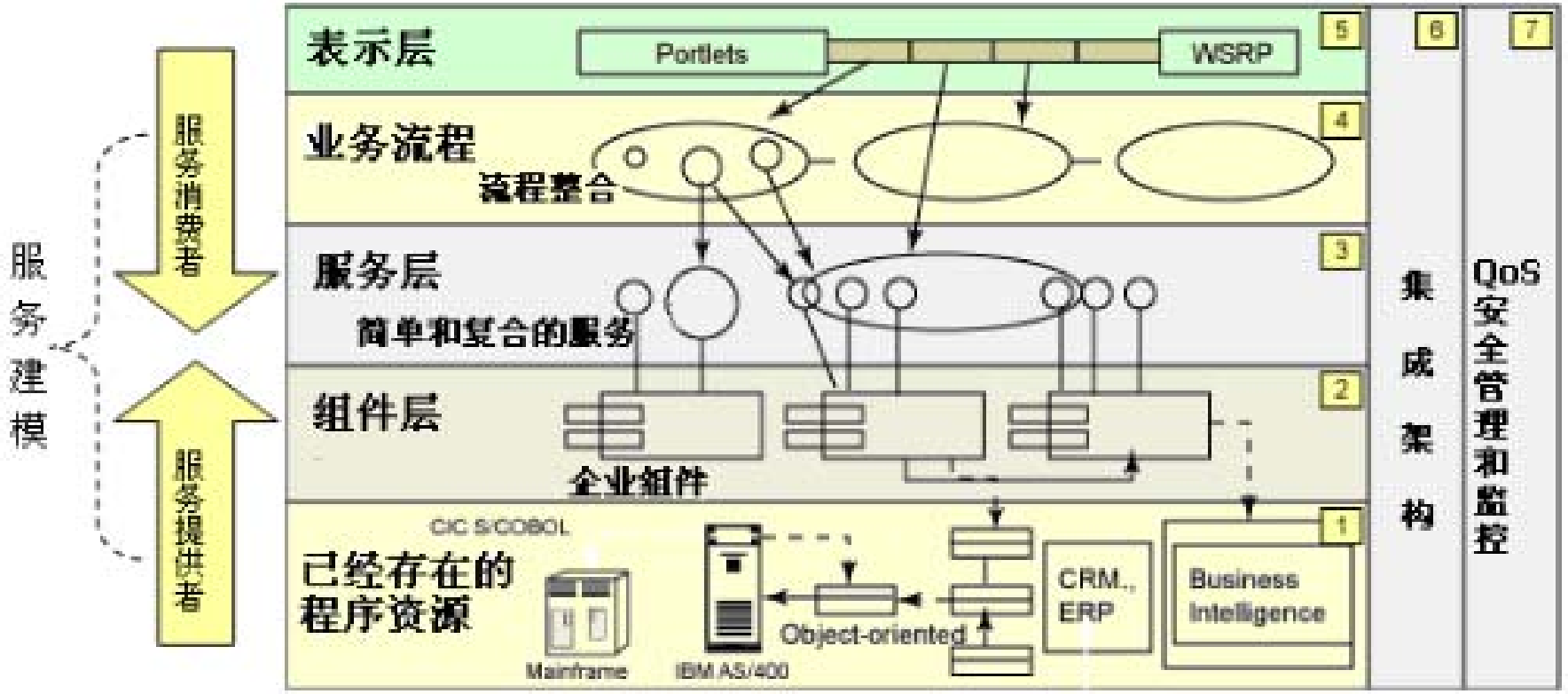
3.2.3 基本模型

- SOA概念模型包含服务提供者、服务注册中心和服务请求者三类角色



SOA概念模型

3.2.3 基本模型

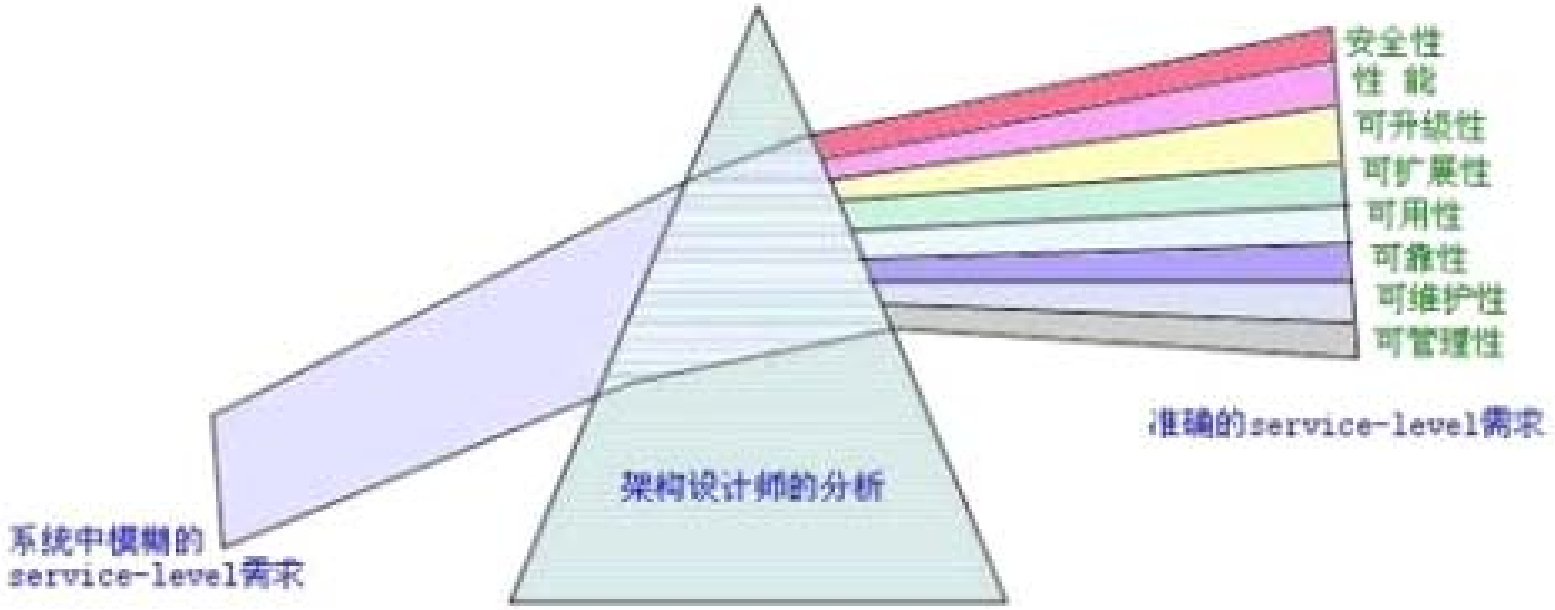


SOA分层模型（Arsanjani，2004）

3.2.4 服务级别

- （1）功能性服务级别
  - 根据服务功能可以分为业务服务、业务功能服务、技术功能服务三种类型。
- （2）非功能性服务级别
  - **SOA** 架构设计师需要分离和抽象系统中性能、可升级性、可靠性、可用性、可扩展性、可维护性、易管理性和安全性等不同的服务级别需求

3.2.4 服务级别



分析过程图

3.2.4 服务级别

- （3）粒度分级
  - 服务粒度的基本原则是粗粒度。为了访问处于不同抽象级别的服务，SOA可以使用不同的协议绑定，不同于Web服务和SOAP。

3.2.5 面向服务的分析和设计

- 面向服务的分析和设计建立在如下三者基础上，是专为面向服务的架构（SOA）设计的软件建模和开发方法
  - 面向对象的分析和设计**(Object-Oriented Analysis and Design，OOAD)**侧重细粒度业务
  - 企业架构**(Enterprise Architecture，EA)**侧重架构分析和设计
  - 业务流程建模侧重业务分析和设计

企业架构

- 将企业应用程序和 IT 基础设施发展成 SOA 可能是一个大的负担，会影响多个业务线和组织单元。因此，需要应用 EA 框架和参考体系结构以努力实现单独的解决方案之间体系结构的一致性。
- 根据过去的经验，大多数现有的 EA 框架都在一个或多个方面有限制。例如，如果主要的问题是表示技术设备的低级构块如何在宏观层次互连，则无法获得业务层流程或服务视图。在 SOA 的背景下，这种考虑问题的方式必须转换为以表示业务服务的逻辑构件为中心并集中于定义服务之间的接口和 服务级协定（Service Level Agreements，SLA）。
- SOAD 必须帮助 SOA 架构师定义服务前景的整体业务级视图。这是当今的 EA 框架所无法提供的，它们需要未来特定于 SOA 的增强；

业务流程建模

- 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。另一种常用的技术是定义表示概念性流程流的 *事件驱动流程链*
- BPM 方法在功能工作单元之上提供了端到端视图，但是它们通常都没有触及体系结构和实现领域。例如，在像 用于 Web 服务的业务流程执行语言出现之前，BPM 表示法缺少操作语义。此外，我们还看到了许多流程建模与开发活动彼此分离的情况。

OO 范式与面向服务 (SO) 范式

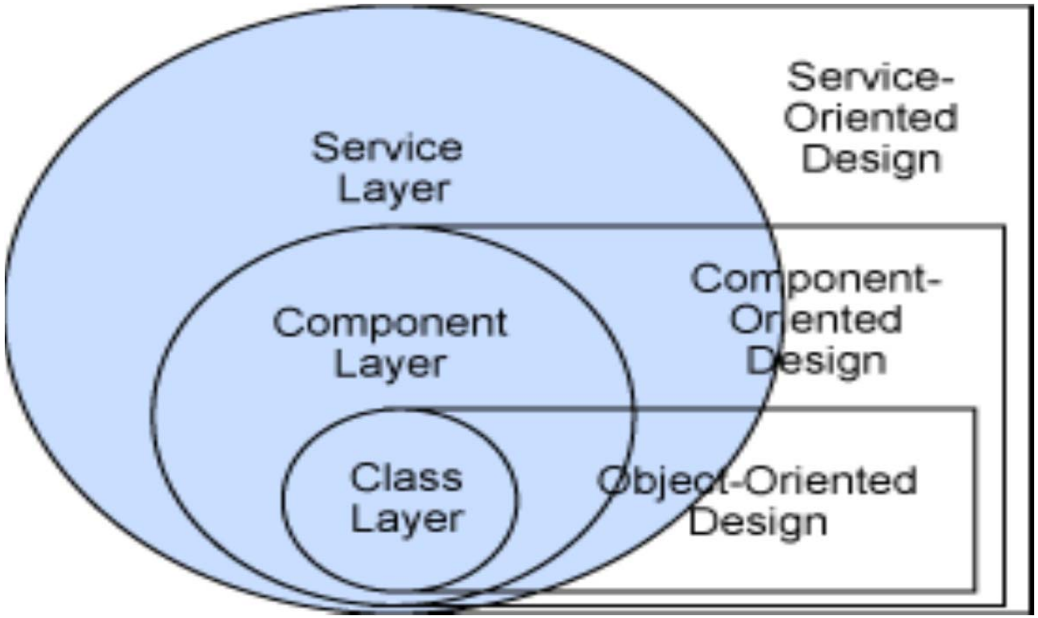
- **OO的基本原则是：**
  - 封装：软件对象就是包含模拟真实世界的对象的物理属性（数据）和功能（行为）的离散包。
  - 信息隐藏：结构良好的对象有简单的接口，并且不向外界显漏任何内部机制。
  - 类和实例： 类是定义特定类型的软件对象具有什么类型的属性和行为的模板，而 实例是具有这些属性值的个别对象。创建类的新实例称为实例化。
  - 关联和继承：在 00 中，表达类和对象之间的关联的能力是一个关键的概念；继承是关联的强形式，用于表达有关系

OO 范式与面向服务 (SO) 范式

- 目前与 SO 有关的 OO 设计实践的主要问题在于，它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了。诸如继承这样的强关联产生了相关方之间一定程度的 紧耦合（因而具有依赖性）。与此相反，SO 范式试图通过 松耦合来促进灵活性和敏捷性。目前，在 SOA 中还没有服务 实例的跨平台继承支持和一流的表示法来避免需要处理服务生命周期维护管理问题

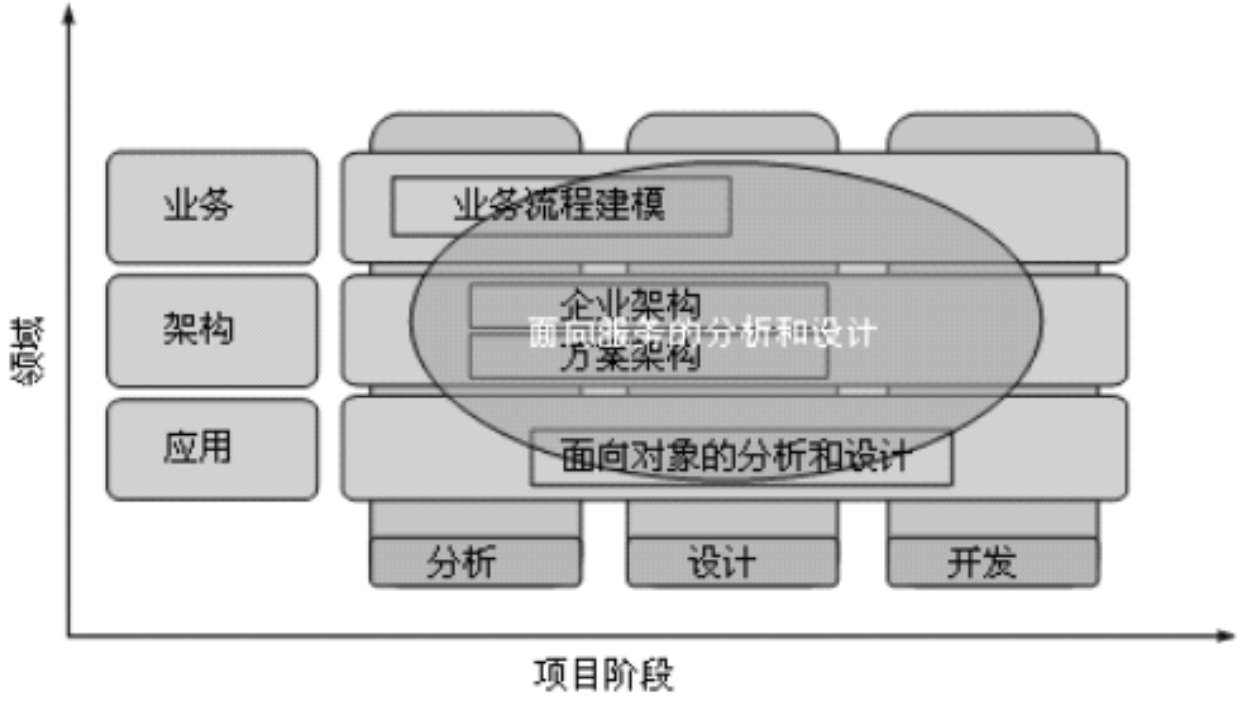
OO 范式与面向服务 (SO) 范式

- 可见性层次和 OO、 面向组件和 SO 设计提供的重点之间的对应关系。它还展示了在 SOA 和 SOAD 背景中它们之间的相互关系。



3.2.5 面向服务的分析和设计

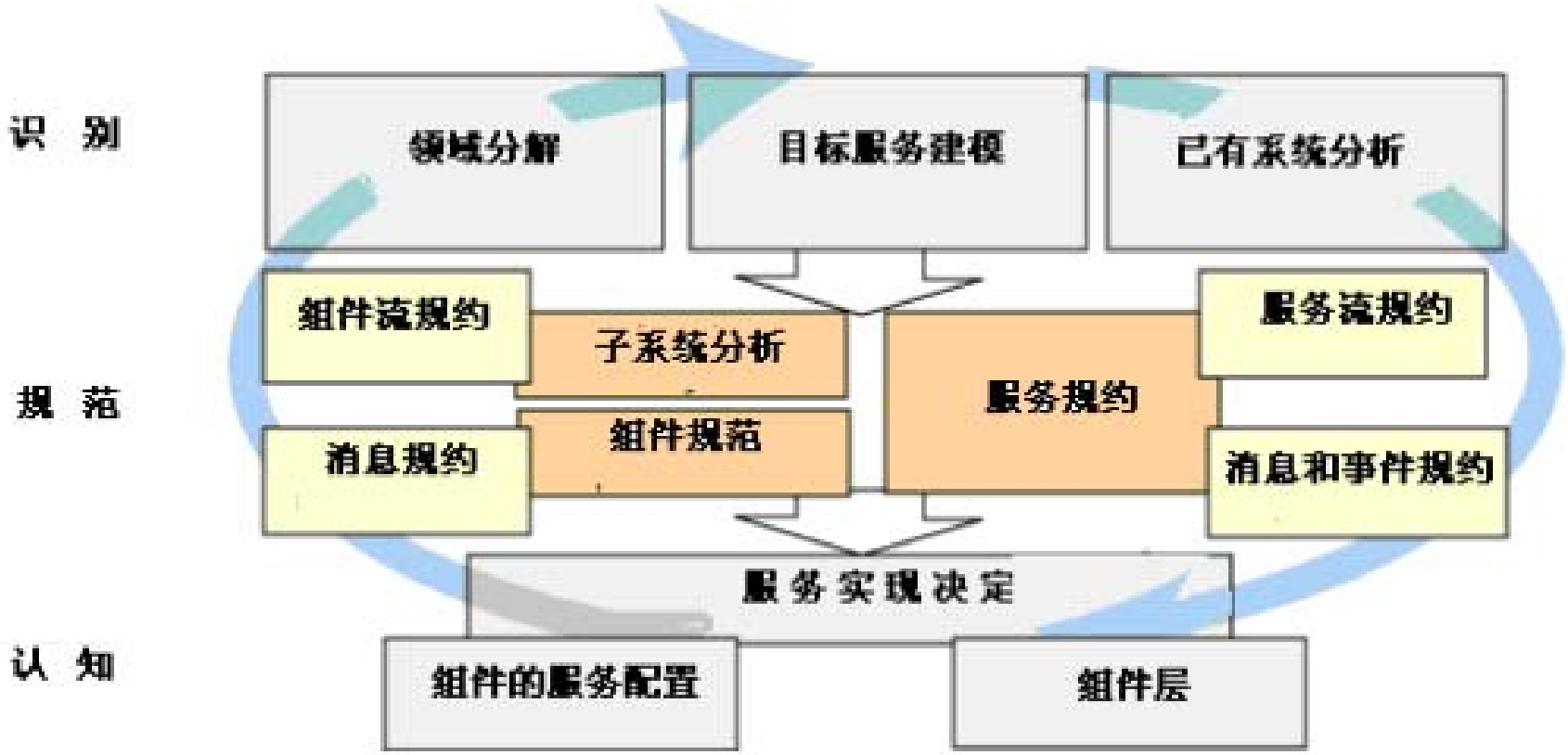
- 面向服务的分析和设计（**Service Oriented Analysis and Design，SOAD**）贯穿项目的三个阶段和不同领域



3.2.5 面向服务的分析和设计

- 在原理上继承了面向对象的分析和设计中信息隐藏、抽象模块化和关注问题分离等设计原则。
- 在操作上，服务模型提供设计类和对象的业务和企业架构的整体指导，补充说明业务流程的用例用于发现和定义服务。
- 结合业务流程建模，把业务分解为各种服务，可以整体指导企业架构的设计，而企业架构设计的结果又是服务实现的输入。

3.2.5 面向服务的分析和设计



3.2.5 面向服务的分析和设计

- 分析在较高的抽象级别上描述系统，可以在服务标识期间使IT与业务保持一致，其输入是一组需求和现有资产，其输出是描述需要构建的各方面，是设计的输入。
- 设计在较低的抽象级别上描述系统及其构建方法。
- 面向服务的分析和设计分为服务发现、服务规约和服务实现。

3.2.5 面向服务的分析和设计

- 服务发现：面向服务的分析活动，发现和整个组织业务相匹配的服务集并定义系统功能将其组织为服务，标识服务逻辑分组、各个分组的概念化服务及其相关操作。
- 服务规约：面向服务的设计活动，完整地指定服务及其操作，为架构的重要服务元素设计结构和行为。
- 服务实现：在设计级描述如何实现服务，包括服务、组件和服务组装的实现，详细程度仅次于代码级。

面向服务的分析和设计原理

- 面向服务的分析和设计需求：
  - 正如任何其他的项目和方法一样，必须正式（至少半正式）地定义 流程和 表示法。通过选择和组合 OOAD、BPM 和 EA 原理，就可以在需要时确定额外的原理。
  - 必须有结构化的方法来概念化服务：
    - ✓ OOAD 为我们提供了应用程序层上的类和对象，而 BPM 具有事件驱动的流程模型。SOAD 需要将它们结合在一起。
    - ✓ 方法不再是面向用况的，而是由业务事件和流程驱动的。用况建模是在更低的层次上作为第二步进行的。
    - ✓ 方法包括语法、语义和策略。这就需要特别的组合、语义代理和运行时发现。

面向服务的分析和设计原理

- 面向服务的分析和设计需求：
  - SOAD 必须提供定义良好的品质因素和最佳实践
  - SOAD 活动还必须回答这样的问题：什么 不是好的服务？
  - SOAD 必须易于进行端到端建模，并且有全面的工具支持。假如 SOA 给业务带来了灵活性和敏捷性，就应该对从企业到体系结构和应用程序设计领域产生的支持方法有相同的期望。

面向服务的分析和设计原理

- 面向服务的分析和设计的基准：
  - 构思良好的服务给业务带来了灵活性和敏捷性；它们通过松散耦合、封装和信息隐藏使重构更加容易。
  - 设计良好的服务是有意义的，并且不只适用于企业级应用程序；服务之间的依赖性减到最少，并且是显式声明的。
  - 服务抽象是内聚、完整和一致的；例如，在设计服务和它们的操作签名时应该考虑 创建（Create）、 读取（Read）、更新（Update）、 删除（Delete）和 搜索（Search）(CRUDS) 隐喻。

面向服务的分析和设计原理

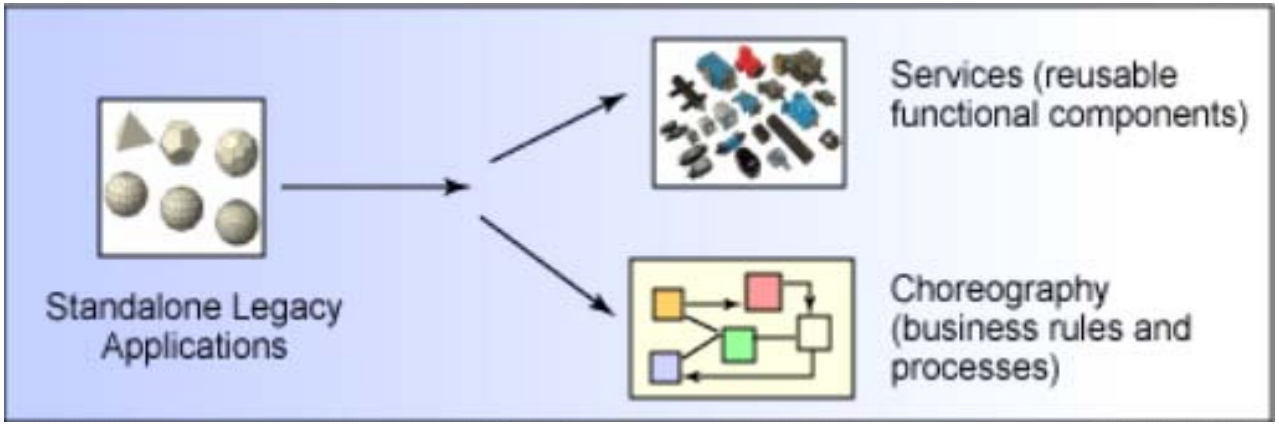
- 面向服务的分析和设计的基准：
  - 常常声明的假定是，服务是无状态的；为了要求服务在特定的问题域和上下文中是无状态的，将削弱这种声明。
  - 领域专家无需深奥的专业知识就可以理解服务命名。
  - 在 SOA 中，所有的服务都遵循相同的设计体系和交互模式；底层体系结构形式可以容易地标识。
  - 服务和服务使用者的开发除了领域知识之外只需要基本的编程语言技能；中间件专业知识只有少数的专业人员才需要，在理想的情况下，这种知识是为工具和运行时厂商所用。

面向服务的分析和设计原理

- 服务标识和定义：自顶向下的业务级建模技术（如 **CBM**）可以为 **SOA** 建模活动提供起点。但是正如我们在前面提到的，**SOA** 实现很少是在全新的项目中开始的；创建 **SOA** 解决方案几乎总需要涉及集成现有的遗留系统，方法是将它们分解成服务、操作、业务流程和业务规则。

面向服务的分析和设计原理

- 服务标识和定义
  - 将现有的应用程序和厂商软件包分解成表示相关操作组的离散服务集（自底向上方法）。
  - 从应用程序中将业务流程和规则抽象为由业务编排模型管理的单独 BPM。



面向服务的分析和设计原理

- 直接和间接业务分析
  - 通过项目相关人员的会谈和 CBM 来进行 BPM 和 直接需求分析是一个容易理解且非常合适的标识候选服务的方法。
  - 通过补充 间接技术来加以改进。在选择候选服务时，产品经理和其他业务领导应该进行协商。
  - 考虑非 SOA 项目中任何现有的用况模型。用于对正在构造的系统是另一个很好的关于服务操作候选者的来源。

面向服务的分析和设计原理

- 域分解
  - 域分解、子系统分析、目标模型创建和相关技术是 SOA 流程构造方法或 服务概念化框架
- 服务粒度
  - 选择正确的抽象级别是服务建模的一个关键问题。由于 SOA 并不等同于 Web 服务和 SOAP，因此可以使用不同的协议绑定来访问抽象级别不同的服务。

面向服务的分析和设计原理

- 除了组合 **OOAD**、**BPM** 和 **EA** 技术之外，还有几个重要的 **SOAD** 概念和方面有待充实：
  - 服务分类和聚合
  - 策略和方面
  - 中间相遇流程
  - 语义代理
  - 服务获取和知识代理

面向服务的分析和设计原理

- 服务分类和聚合
  - 服务有不同的用法和用途；还可以将原子服务编排成级别更高、功能齐全的服务。
  - 服务组合可以通过可执行模型（如 BPEL 建模的模型）来加以简化；这是传统的建模工具和方法不能处理的事情。

面向服务的分析和设计原理

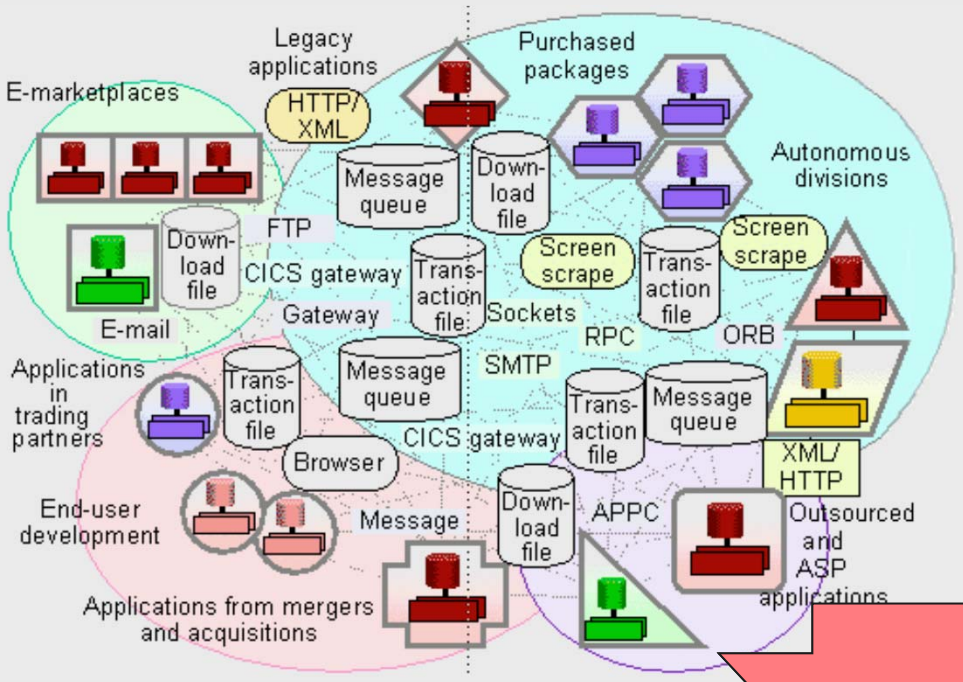
- 策略和方面
  - 服务具有语法、语义和 QoS 特征，所有这些都必须进行建模；正式的接口契约必须涵盖的比 Web 服务描述语言
  - 除了已经制定的良好 体系结构可跟踪性原则外，业务可跟踪性也是一个理想的品质：应该有可能将所有的运行时构件直接与非技术领域专家可以理解的语言联系起来。

面向服务的分析和设计原理

- 流程：中间相遇
  - 在真实世界中，并没有全新的项目，必须始终考虑遗留系统。因此，需要 中间相遇的方法，而不是单纯的自顶向下或自底向上的流程。
  - 在设计取决于现有的 IT 环境而不是现在和将来的业务需要的情况下，自底向上的方法往往会导致不好的业务服务抽象。而自顶向下的方法可能会产生不足的非功能性需求特征

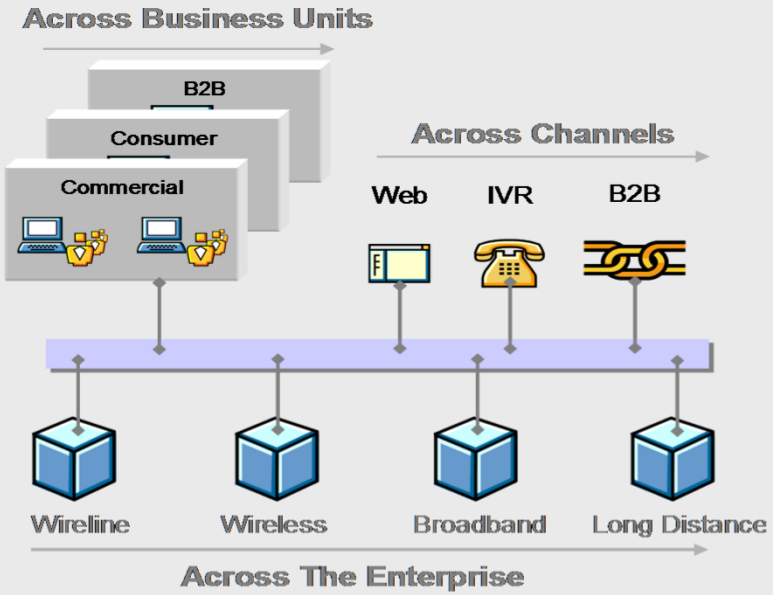
面向服务的分析和设计原理

- 服务获取和知识代理
  - 应该将重用看作是标识和定义服务最主要的推动标准之一。如果组件（或服务）不可能重用，就不无法将其作为服务进行部署。它可以连接到另一个与企业体系结构相关的服务，但是不能单独作为一个服务而存在。



企业现存的信息系统－多种技术的集合:

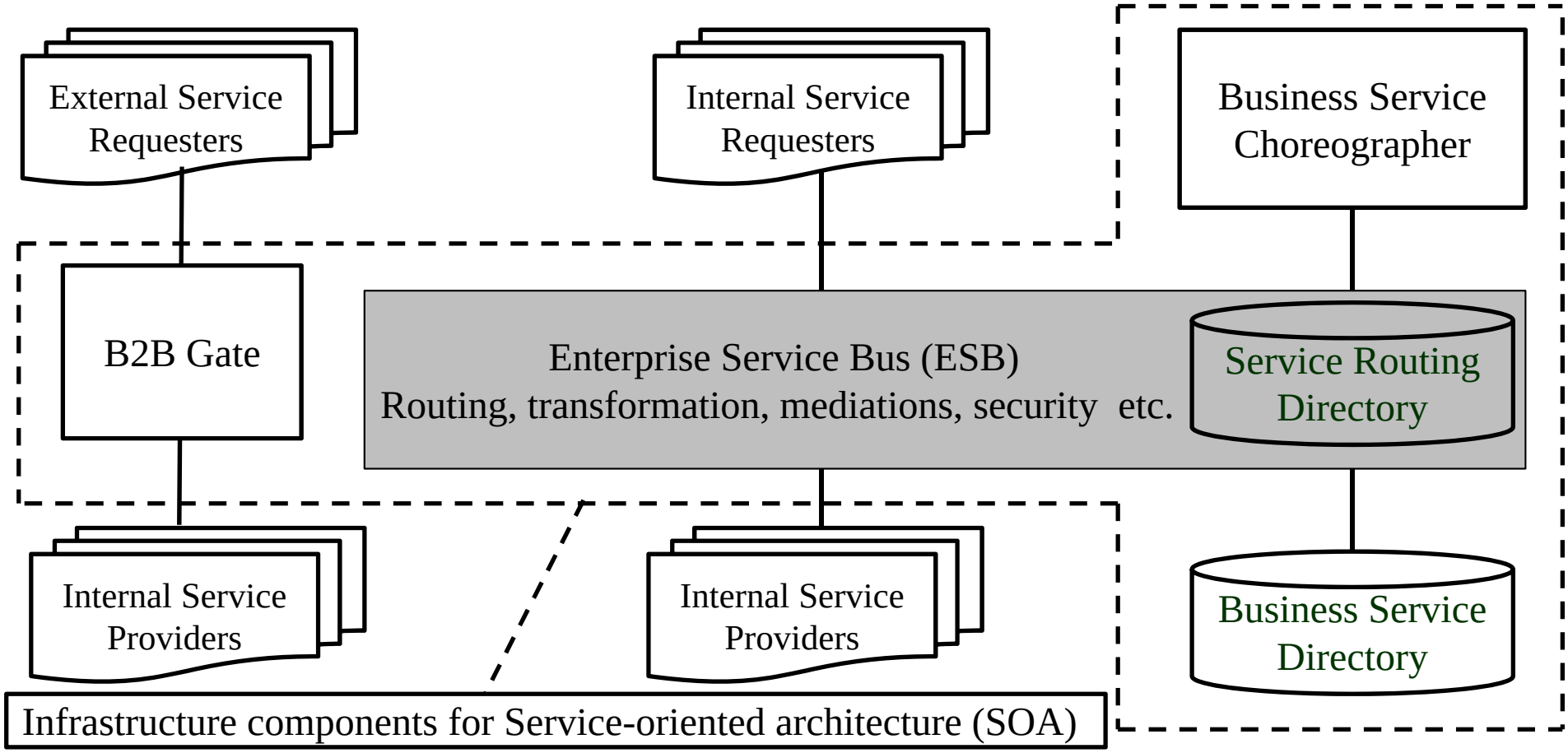
- **Mainframe transactions**
- **C++ Client/Server Apps**
- **Middleware Islands (CORBA,Tuxedo,MQ,Tibco,J2EE…)**
- **Home Grown / Dark Matter**



采用SOA标准化现存的信息系统

- 插件式的可复用的业务逻辑和商务流程
- 屏蔽单个系统的复杂性
- 向基于Web服务的现代软件开发平台演进

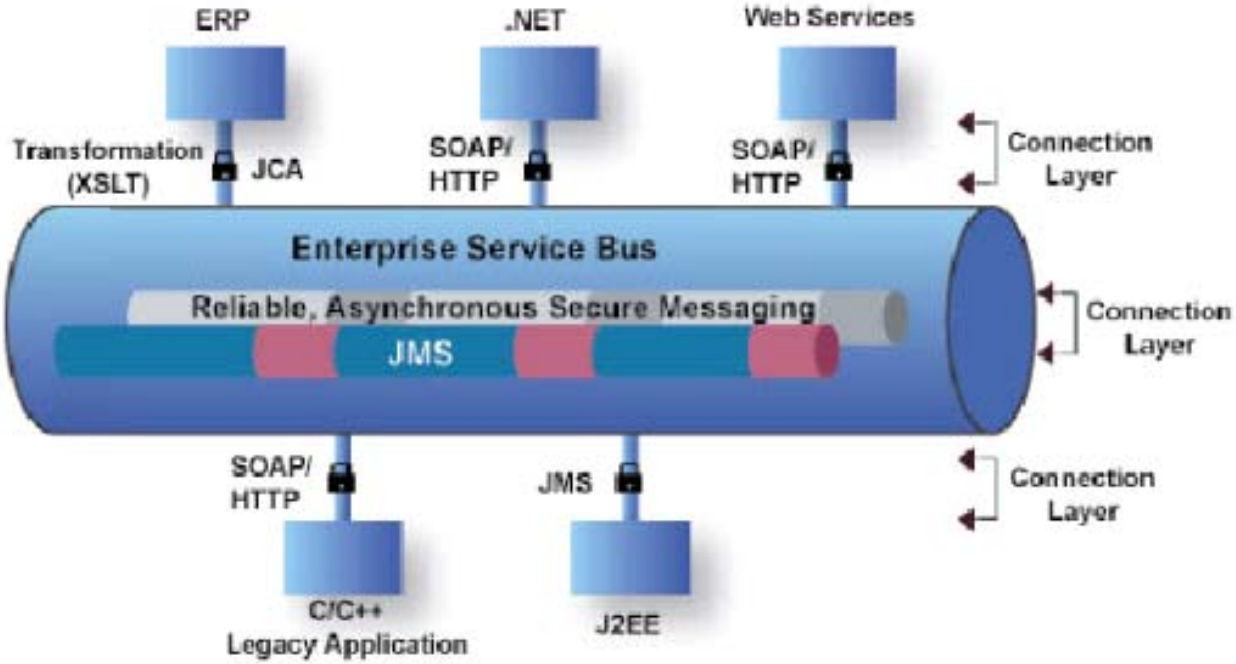
ESB 在SOA 内的工作角色

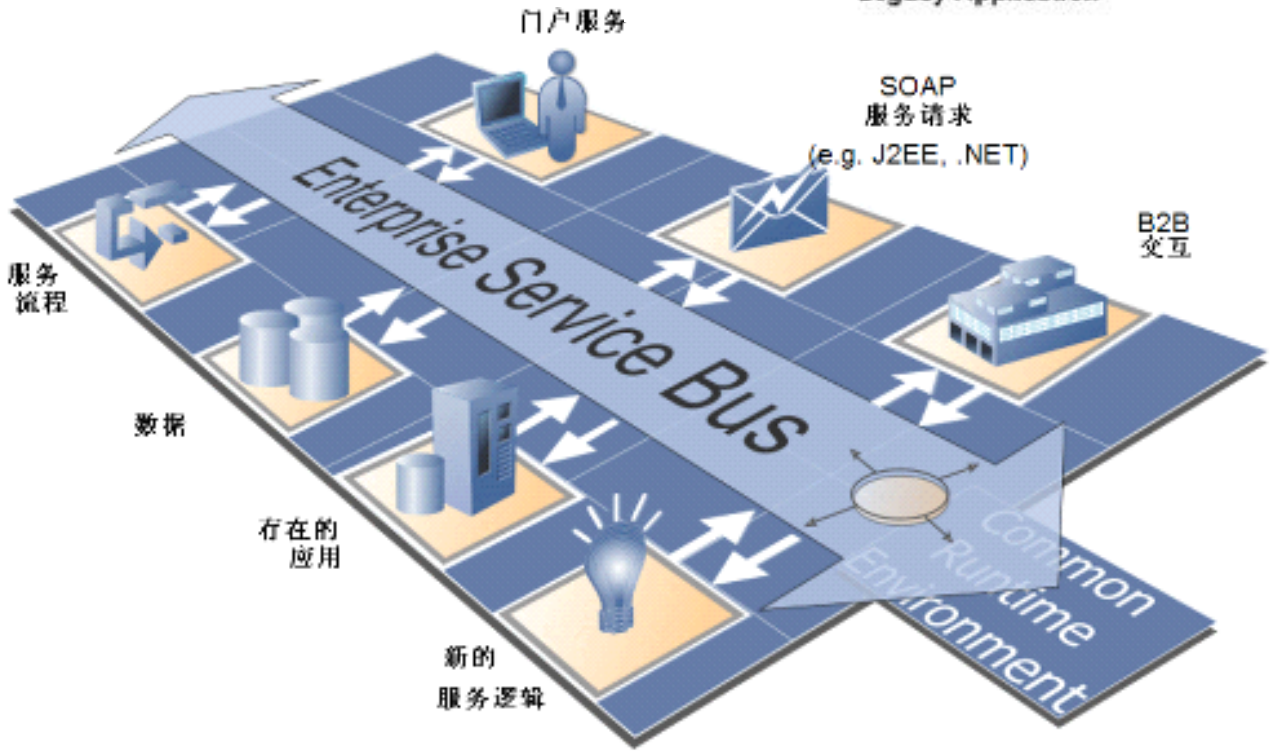


ESB

- **Gartner：An Enterprise Service Bus (ESB) is a new kind of middleware that combines features from several previous types of middleware into one package. ESBs provide the fabric of services required for enterprise system interoperability and building new applications.**
- **ESB (Enterprise Service Bus，企业服务总线) 是一种中介，位于基础架构服务（infrastructure services）和应用服务（application services）之间**

ESB


ERP
.NET
Web Services
Transformation (XSLT)
JCA
SOAP/ HTTP
SOAP/ HTTP
Connection Layer
Enterprise Service Bus
Reliable, Asynchronous Secure Messaging
JMS
Connection Layer
SOAP/ HTTP
JMS
Connection Layer
C/C++ Legacy Application
J2EE


门户服务
SOAP
服务请求
(e.g. J2EE, .NET)
B2B
交互
服务
流程
Enterprise Service Bus
数据
存在的
应用
新的
服务逻辑
Common Runtime Environment

ESB的特点

- 将现存的企业信息系统标准化为**web** 服务接口
- 具有集成多种中间件和多种技术标准的能力
- 开放式的，组件化，基于配置的模型
- 高度的可分布性和高性能
- 简化开发，插件式结构
- 对基于标准的高性能的复杂应用的可靠支持

ESB 和 传统中间件的对比

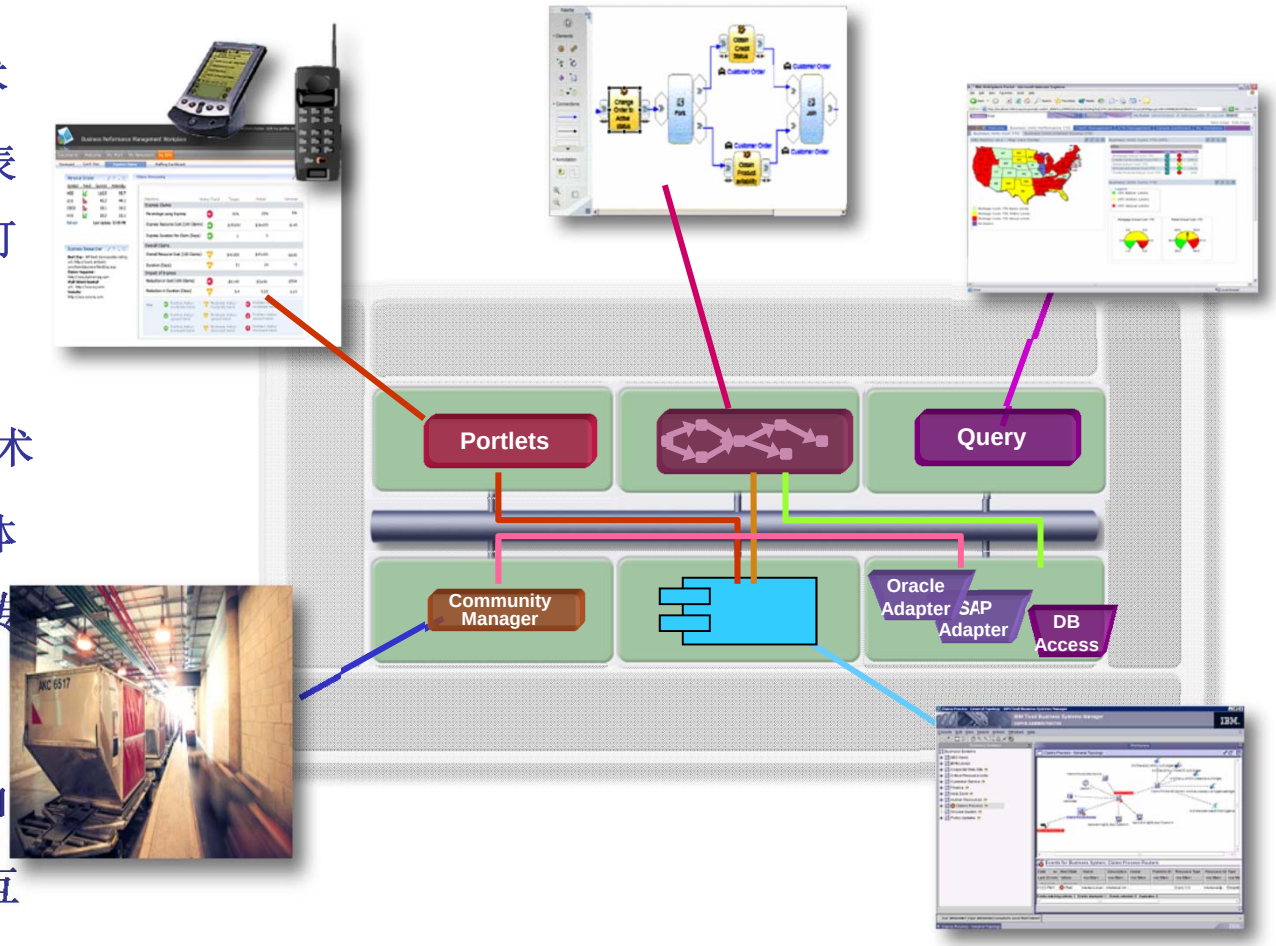
- 相同点（基础架构平台）
  - 开放标准
  - 分布式架构
  - 组件技术
  - 高性能，适合于复杂的应用集成
- 不同点
  - **ESB**支持更多种技术标准（**Middleware of Middlewares**）
  - **ESB**支持更广泛的互操作性（**Web** 服务）
  - **ESB**要求具有更好的可扩展性
  - 对专有系统的支持（便于用户定制化）
  - 对未来标准的支持（保护现有投资，实现非破坏性升级）

基于服务设计的核心元素

服务组件:一种具有技术和语言独立性的服务的表现形式，并且服务组件可以和别的组件组合。

服务数据: 一种具有技术和语言独立性的数据实体，并且它可以在服务间传递。

服务总线:一种具有技术和语言独立性的服务间相互联系的表现形式。



3.3服务网格和云计算

- 3.3.1什么是网格
- 3.3.2网格的特征
- 3.3.3网格的架构
- 3.3.4服务网格概述
- 3.3.5什么是云计算
- 3.3.6服务网格到云计算
- 3.3.7云计算的形式
- 3.3.8云计算的本质
- 3.3.9云计算的风险

3.3.1 什么是网格

- 3.3.1.1网格问题的提出
- 3.3.1.2网格的定义
- 3.3.1.3网格的本质和目标

3.3.1.1网格问题的引出

- 随着社会的发展，人们的应用需求不断提高，同时原有问题的求解领域的不断扩展和复杂化，并层出不穷的涌现出新问题需求，但是现有的计算机和设备无论从自身的性能还是从局部的合作上都无法满足人们日益增加的要求，然而由于网络等相关技术的发展，使得世界各地的计算机可以联网协同工作，利用空闲的以前无法使用和共享的设备等空闲资源。
- 网格是借鉴电力网的概念提出的。提出网格的目的就是能够使得人们在使用网格资源的时候，能够像使用电力资源一样，自由使用，而不用关心我现在使用的电力资源是水力发电的还是热电呢，是从哪个发电厂得来之类的事情。网格也希望给最终用户提供的是与地理位置无关，与具体的计算设施无关的通用的计算能力。

3.3.1.2 网格的定义

到目前为止，关于什么是网格和什么是网格计算还没有一个普遍接受的定义，关于网格概念的分歧和争论仍然存在。

（1）网格就是下一代的Internet。

（2）网格计算就是在动态变化的、拥有多个部门或者团体的复杂虚拟组织内，灵活、安全的协同资源共享与问题求解。

（3）网格就是方便资源管理，有效支持广域分布的、多领域的科学与工程问题解决的中间件系统。

（4）网格是建造分布式科学计算环境的一种一体化的集成方法，这一环境包括计算、数据管理、科学仪器以及人类的协作。

（5）网格是一种无缝的、集成的计算与协作环境。

（6）网络是基于硬件支持的各种服务和功能的提供者。

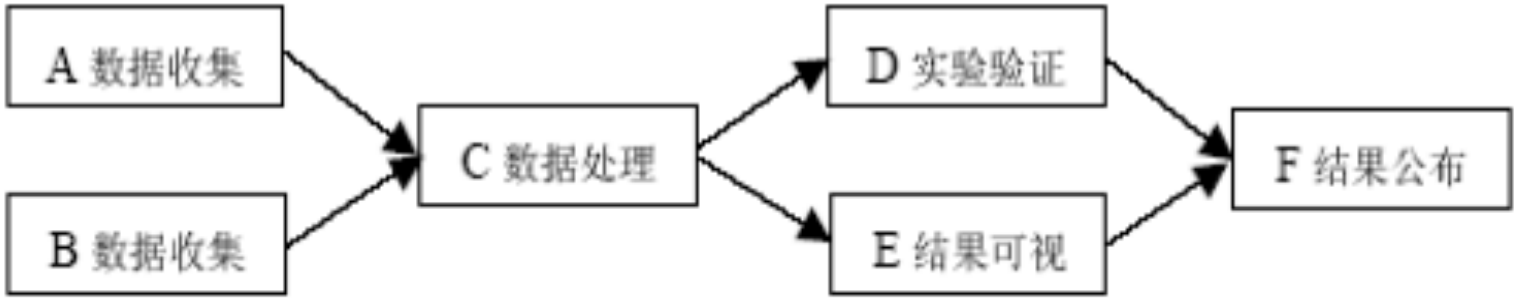
这些都是目前出现的一些有关网格的定义，它们从不同的角度和侧重点来阐述了对网格的认识。

3.3.1.3网格的本质和目标

- 网格的本质
  - 网格的本质不是它的规模，而是充分利用互连网络中的现有软硬件资源，支持广域环境上的计算数据、存储、信息和知识资源的共享、互通与互用，消除资源孤岛。以较低成本获得较高的性能
- 网格的目标
  - 网格的目标是将地理上分布的、系统上异构的多种计算资源通过高速网络连接起来，协同解决大型应用问题，进行广域信息资源的分布共享，最终把整个因特网整合成一个超级虚拟计算机。

3.3.2 网格的特征

- 分布性：组成网格的资源可能是计算资源、存储资源、数据资源、仪器资源等，它们分布在地理位置不同的许多地方。



网格的分布性

3.3.2 网格的特征

- 异构性：组成网格的资源是异构的，对于计算资源，有不同类型的计算机，不同的计算方式，不同的计算接口，不同的系统架构。对于存储资源和其他资源，也面临这样的问题。
- 自治性：网格上的资源首先是属于某一本地的个人或者组织，网格资源的拥有者对资源具有最高级别的管理权限，网格应该允许资源拥有者对其资源有自主的管理能力，因此，网格具有自治性。

3.3.2 网格的特征

- 动态性：由于网格资源具有自治性，因此网格资源可能动态地加入或者退出网格，也可能出现故障而导致不可用。
- 自相似性:网格的局部和整体之间存在着一定的相似性，局部往往在许多地方具有全局的某些特征，而全局的特征在局部也有一定的体现。

3.3.3网格的架构

- 要实现一个网格，我们必须了解它的基本组成部分，每一部分的作用，这些部件之间存在的关系等。而所谓的网格体系结构，就是包含了上述信息，并且描述了怎样建造网格及支持网格有效运转的技术。

3.3.3 网格的架构

1.几个重要的概念

在开始介绍网格体系结构的有关模型之前，有必要先理解下面一些概念：

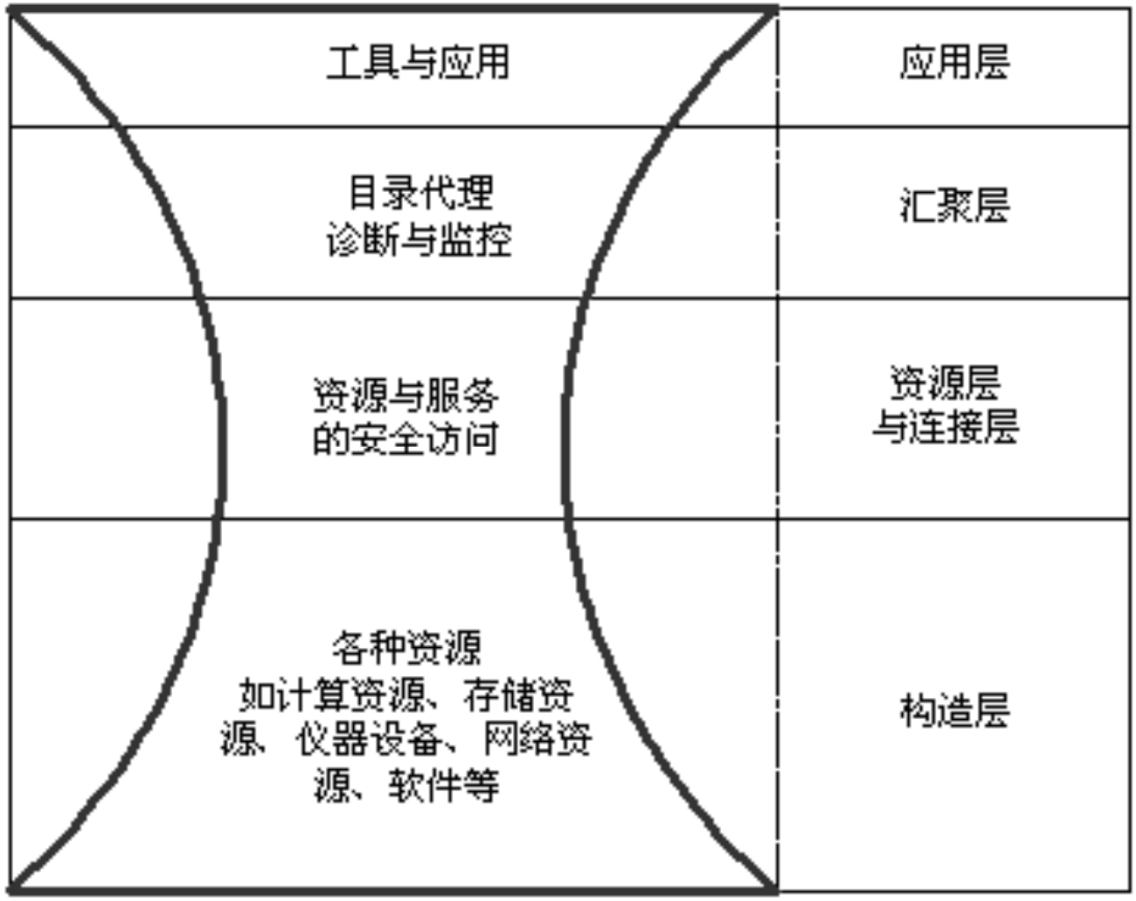
- ① 协议：在分布式系统的各个组成部分确定一种通用的交互方式。实际上就是定义了一种“相互交流的规范--协议”。而所谓的标准协议，就是将协议按统一规则标准化的结果。
- ② 服务：我们将协议实现的功能称之为服务。由标准协议定义的标准服务增强了提供服务的能力，抽象掉了与资源相关的细节。
- ③ **API/SDK**：应用程序接口（**Application Programming Interface**）和软件开发包（**Software Development Kits**）,可以使开发人员跳过对于互操作、协议、服务这些低级的开发，而直接针对高级应用进行工作
- ④ 共享：网格中的共享，更侧重于对于包括硬件软件在内的各种资源的直接访问。
- ⑤ 虚拟组织：基于某些共享规则，由一些个人或团体形成的集合体。

3.3.3 网格的架构

- 五层沙漏结构
  - 五层沙漏结构是一个以协议为中心同时强调服务和**API/SDK**重要性的结构，但它并不侧重于协议的具体定义，而是着重于协议的描述，因此它非常容易理解。

3.3.3网格的架构

- 五层沙漏结构



3.3.3 网格的架构

- 五层结构之所以被称为漏斗，是因为每个层次所包含的协议数量是不相等的。
- 有一部分协议称为核心协议,这类协议需要在所以支持网格技术的地点都被支持。上层协议向核心协议的映射以及核心协议向下层协议的映射都应该可以实现。
- 核心协议部分形成了沙漏的瓶颈部分

3.3.4 服务网格概述

- 开放网格服务结构**OGSA**（**Open Grid Services Architecture**）是一种以服务为核心的结构，这与五层沙漏模型以协议为中心的结构形成鲜明对照。
- 五层沙漏模型中强调的是资源的共享，而**OGSA**则强调服务的共享，这里的服务概念是广义的，不仅指由资源提供的服务，也包括资源本身。
- 为了使**OSGA**中的服务的含义更清晰和明确，出现了网格服务（**Grid Service**）的概念：网格服务是**Web Service**的一种，提供了由良好定义的接口构成的集合，以及遵循特殊的、为支持网格而制订的规范。

3.3.4 服务网格概述

- OSGA将一切都视为服务，因此网格在这里也可以看成可扩展的网格服务构成的集合。
- 五层模型的构造过程更多的是基于协议本身的需要，即更注重实体对象本身；而OGSA强调的则是与协议对应的服务，即更注重实体对象所表现出来的行为特征。
- OGSA以下三个方面的特点值得我们注意：
  - ① 面向服务的体系结构(Service-oriented Architecture)。
  - ② 与Web Service技术的联系。
  - ③ 和商业应用需要的紧密联系。

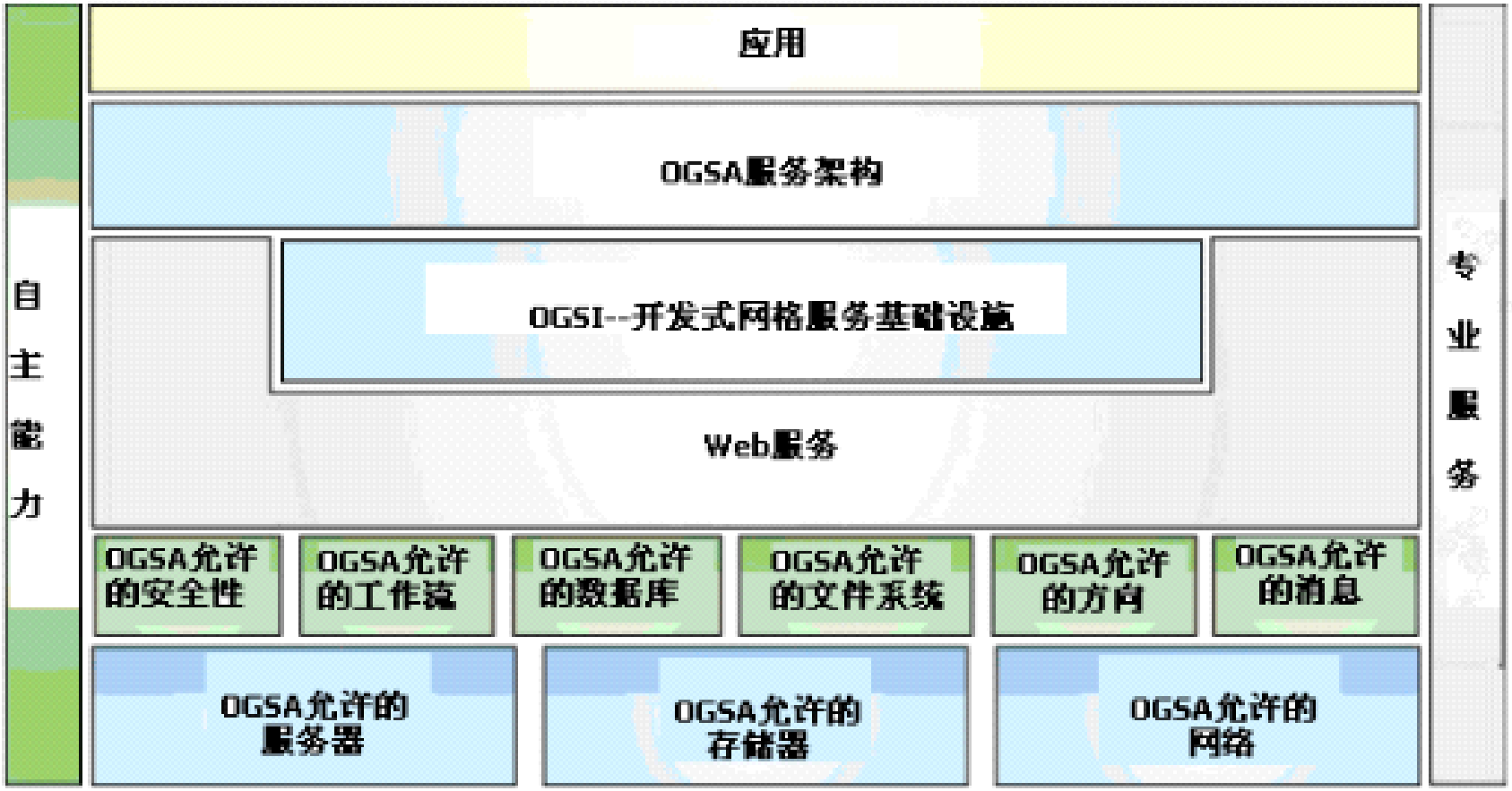
3.3.4 服务网格概述

- 在OGSA体系中存在两种服务，临时性服务(TransientServices)和持久服务（PersistentServices），但是临时服务占了绝大多数。针对这一特点，OSGA具备了相应的基本结构：
- A) 使用标准WSDL规范及其扩展来描述和对服务结构化。在Web Service中，WSDL用来描述服务、实现接口以及完成访问的方法。为了适应网格服务中临时服务居多的特点，OGSA对WSDL进行了必要的扩展。

3.3.4 服务网格概述

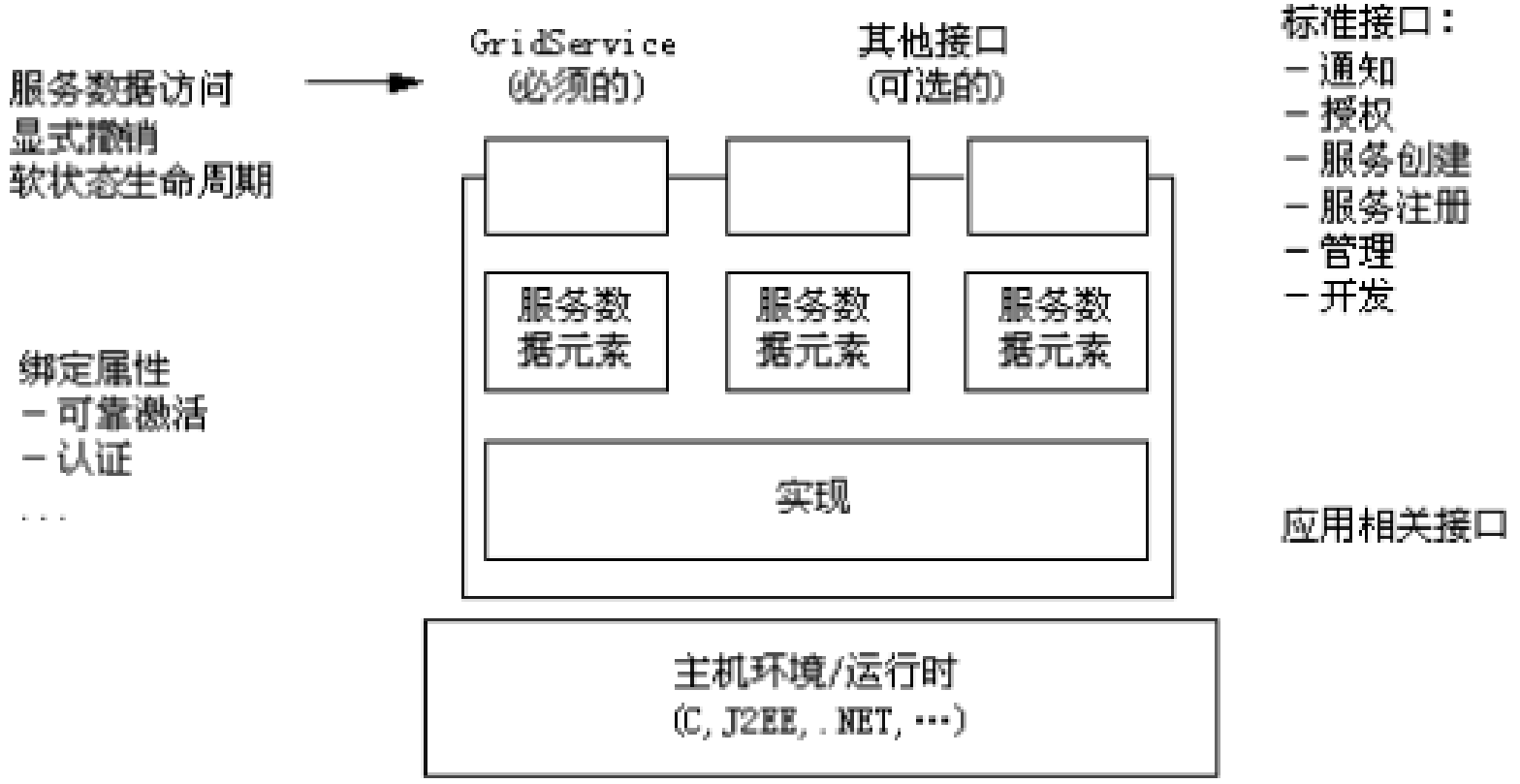
- B) 为了进一步提高效率，增强为虚拟组织用户服务的能力，OGSA针对核心服务定义了一些标准接口和行为：
  - 命名和绑定：每一个服务实例都有唯一的名字——网格服务句柄GSH(Grid Service Handle)，以方便发现其支持的绑定。
  - Factory接口：这个接口定义了CreateService操作，通过这个操作可以创建一个新的GS(Grid Service)实例，并且返回一个GSH。
  - GridService接口：这是唯一一个在OGSA所有服务中都必须包含的接口。通过这一接口，我们实现了对于临时服务的生命周期管理。
  - Registry接口：用来发现服务实例的集合，主要作用是注册一个GSH。
  - Service Data：它是服务数据元素的集合，包括基本的内部信息、接口的特殊信息和应用的数据，其实质是一个XML片段，封装在标准容器中，可以通过FindServiceData操作来查询这些信息。
  - Notification接口：负责对服务的存在（如创建一个新的服务实例）和数据的变化进行通知。

3.3.4 服务网格概述



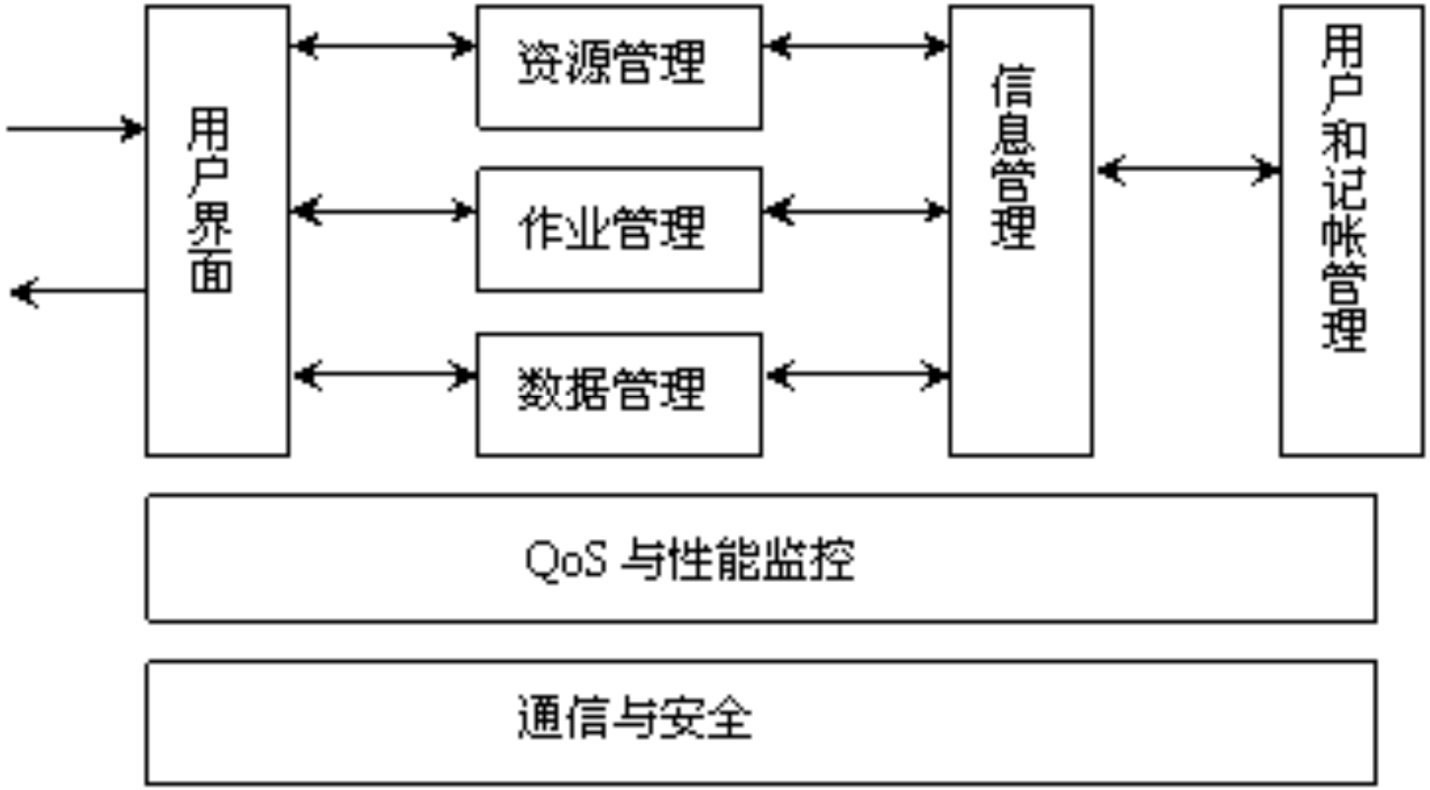
开放网格服务架构（OGSA）

3.3.4服务网格概述



网格服务示意图

3.3.4 服务网格概述



网格系统的基本功能模块示意图

3.3.5 什么是云计算

- 云计算(Cloud Computing)是一种新兴的商业计算模型。它将计算任务分布在大量计算机构成的资源池上，使各种应用系统能够根据需要获取计算力、存储空间和各种软件服务。
- 这种资源池称为“云”。“云”是一些可以自我维护和管理的虚拟计算资源，通常为一些大型服务器集群，包括计算服务器、存储服务器、宽带资源等等。云计算将所有的计算资源集中起来，并由软件实现自动管理，无需人为参与。这使得应用提供者无需为繁琐的细节而烦恼，能够更加专注于自己的业务，有利于创新和降低成本。

3.3.5 什么是云计算

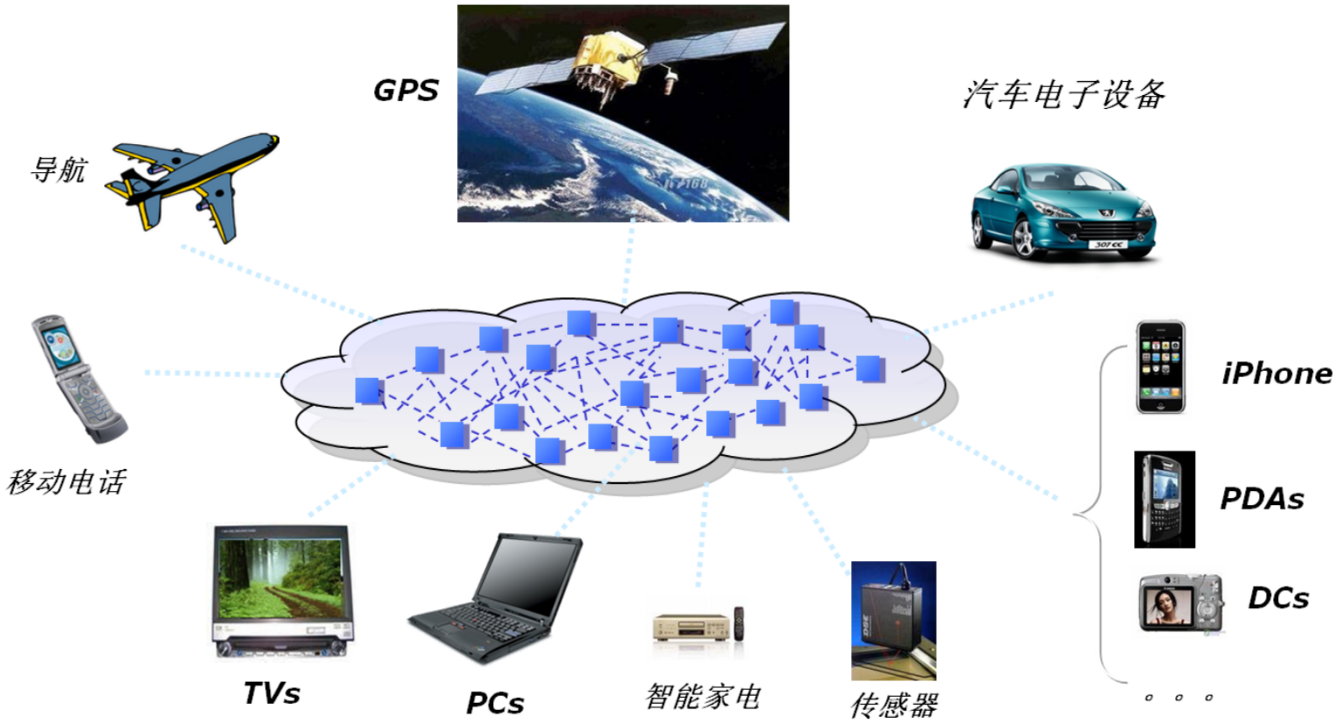
- 之所以称为“云”，是因为它在某些方面具有现实中云的特征：云一般都较大；云的规模可以动态伸缩，它的边界是模糊的；云在空中飘忽不定，你无法也无需确定它的具体位置，但它确实存在于某处。

3.3.6从服务网格到云计算

- 服务网格(Service Grid)结合了Web服务技术和网格技术，能够提供面向服务的创建、过滤、组织、定位及选择的机制。
- 云计算是大规模计算机集群，一个基于互联网的超级计算模式，为主流的企业用户引入，把存储于个人电脑、移动电话和其他设备上的大量信息和处理器资源集中在一起，协同工作。
- 云计算的特征和网格计算的特征基本相同，是网格计算的发展。

3.3.7云计算的形式

- 云计算的形式有SaaS、实用计算、网络API服务、平台即服务、管理服务提供者、商业服务平台、互联网整合等多种。



3.3.7 云计算的形式

- **SAAS（软件即服务）**
  - 这种类型的云计算通过浏览器把程序传给成千上万的用户。在用户眼中看来，这样会省去在服务器和软件授权上的开支；从供应商角度来看，这样只需要维持一个程序就够了，这样能够减少成本。**Salesforce.com**是迄今为止这类服务最为出名的公司。**SAAS**在人力资源管理程序和**ERP**中比较常用。**Google Apps**和**Zoho Office**也是类似的服务

3.3.7 云计算的形式

- 实用计算（**Utility Computing**)
  - 这个主意很早就有了，但是直到最近才在**Amazon.com**、**Sun**、**IBM**和其它提供存储服务和虚拟服务器的公司中新生。这种云计算是为**IT**行业创造虚拟的数据中心使得其能够把内存、**I/O**设备、存储和计算能力集中起来成为一个虚拟的资源池来为整个网络提供服务。

3.3.7 云计算的形式

- 网络服务
  - 同**SAAS**关系密切，网络服务提供者们能够提供**API**让开发者能够开发更多基于互联网的应用，而不是提供单机程序。
- 平台即服务
  - 另一种**SAAS**，这种形式的云计算把开发环境作为一种服务来提供。你可以使用中间商的设备来开发自己的程序并通过互联网和其服务器传到用户手中。

3.3.7 云计算的形式

- **MSP**（管理服务提供商）
  - 最古老的云计算运用之一。这种应用更多的是面向**IT**行业而不是终端用户，常用于邮件病毒扫描、程序监控等等。
- 商业服务平台
  - **SAAS**和**MSP**的混合应用，该类云计算为用户和提供商之间的互动提供了一个平台。比如用户个人开支管理系统，能够根据用户的设置来管理其开支并协调其订购的各种服务

3.3.7 云计算的形式

- 互联网整合
  - 将互联网上提供类似服务的公司整合起来，以便用户能够更方便的比较和选择自己的服务供应商。

软件即服务（Software as Service, SaS ）

- 以产品为中心向以服务为中心转变
- 发展过程:
  - 有服务：方便使用
  - 有好服务：保证服务质量
  - 个性化服务：满足不同个体的服务需求

Internet环境及其应用

网格

从资源共享与管理的角度探讨未来网络系统的应用模式

普适计算

从“人机交互”的角度研究未来网络系统的应用式

服务计算

从“软件即服务”的理念出发，提出了一种新型的软件形态并着重探讨软件的协同性和动态性

模型驱动

以特定于领域的代码自动生成为基础，探讨基于中间件的软件系统的开发方法与技术

对等计算

语义网

自治计算

随需即取

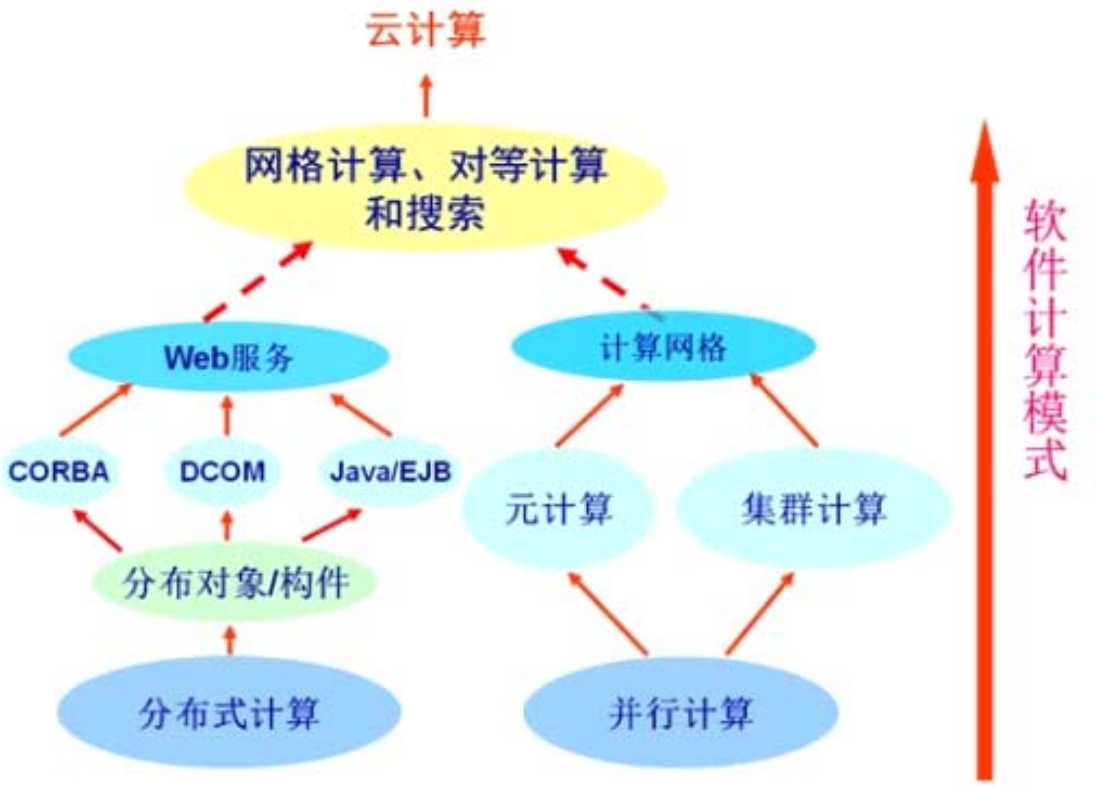
3.3.8云计算的本质

- 云计算的虚拟计算、粒度计算、语义计算、不确定性计算等特点，与自然界的云的相应特点类似

自然云	云计算
云层若隐若显	云计算的虚拟计算
云滴可大可小	云计算的粒度计算
云海丰富多彩	云计算的语义计算
云絮漂浮不定	云计算的不确定性计算

3.3.8云计算的本质

- （1）云计算不是资源托管。
- （2）虚拟计算。
- （3）粒度计算。
- （4）语义计算。
- （5）不确定性计算。



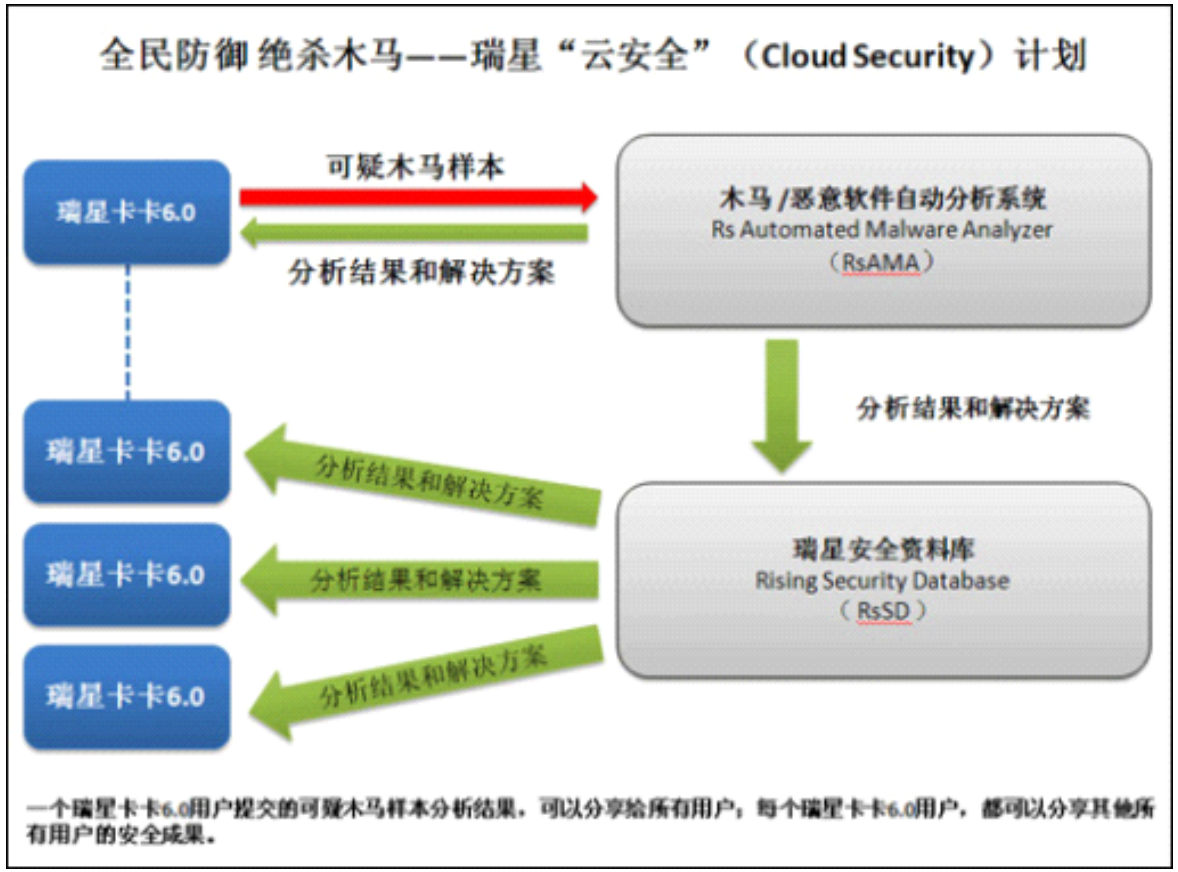
3.3.9云计算的风险

- 云计算的风险主要指敏感资源安全性问题
  - 访问权风险：若使用云计算，你的机密数据将由贵公司外面的人员来处理，所以可想而知：不是贵公司的员工完全可以访问这些数据。
  - 管理风险：在《萨班斯－奥克斯利法案》当道的时代，公司有责任实施严格的数据监控和归档级别。即便一家公司与外部的云计算服务提供商签订了合同，这些法规仍要求这家公司负有责任。云计算服务提供商应当提交审计和安全方面的证书，确保对方能够履行约定的承诺。“如果云计算提供商不愿意或者没能力做到遵从法规，这表明客户只能用它们来处理最不重要的功能。”
  - 数据风险：若使用云计算，你不知道自己的数据到底存放在什么地方。服务器可能建在马来西亚、加拿大或者美国的新泽西州，说不定同时建在上述三个地方。

3.3.9 云计算的风险

- 数据隔离风险：云计算提供商会使用**SSL**来保护传输中的数据，但当贵公司的数据位于存储设备中时，可能与其他公司的数据共用一只“虚拟保管箱”。提供商可能会夸耀自己的加密技术如何强大、安全。你会听到密钥长度有多长、采用哪种深奥的加密算法。不过，如果你的数据能够被提供商读取，那么可以这么认为：数据也会被别人读取。
- 数据恢复风险；你的提供商有能力进行全面恢复吗？需要多少时间才能完成全面恢复？因为他不只为你一家服务。
- 企业数据调查风险：开展内部的法律调查向来就不是容易的事，因为这需要清查可能散布在实体位置和虚拟位置的大批文档。如果你使用云计算服务提供商，那么开展这种调查更是困难重重：许多客户的数据也许散布在地点不断变化的一系列数据中心。

3.3.9云计算的风险


全民防御 绝杀木马——瑞星“云安全”（Cloud Security）计划
瑞星卡卡6.0
可疑木马样本
分析结果和解决方案
木马/恶意软件自动分析系统
Rs Automated Malware Analyzer
（RsAMA）
分析结果和解决方案
瑞星卡卡6.0
分析结果和解决方案
瑞星卡卡6.0
分析结果和解决方案
瑞星卡卡6.0
分析结果和解决方案
瑞星安全资料库
Rising Security Database
（RsSD）
一个瑞星卡卡6.0用户提交的可疑木马样本分析结果，可以分享给所有用户；每个瑞星卡卡6.0用户，都可以分享其他所有用户的安全成果。

思考题

- 【1】分析Internet环境下的Web服务内容及其特点。
- 【2】从不同角度理解面向服务的架构的概念
- 【3】在面向服务的架构中，服务提供者、服务注册中心和服务请求者各有什么作用？
- 【4】服务网格的架构有哪些？
- 【5】分析云计算的形式和风险。