软件架构复习

# 第一章 软件体系结构概论

## 软件危机的表现P1

◎软件成本日益增长

◎开发进度难以控制

◎软件质量差

◎软件维护困难

## 软件危机的原因P2

◎用户需求不明确

◎缺乏正确的理论指导

◎软件规模越来越大

◎软件复杂度越来越高

## 如何克服软件危机P3

人们面临的不光是技术问题，更重要的是管理问题。管理不善必然导致失败 。

要提高软件开发效率，提高软件产品质量，必须采用工程化的开发方法与工业化的生产技术。

在技术上，应该采用基于重用的软件生产技术；在管理上，应该采用多维的工程管理模式

## 构件P4

构件的定义构件是指语义完整、语法正确和有可重用价值的单位软件，是软件重用过程中可以明确辨识的系统；结构上，它是语义描述、通讯接口和实现代码的复合体。

## 构件重用P10

# 第二章 软件体系结构建模

## 软件体系结构模型的分类P31

◎ 结构模型

◎ 框架模型

◎ 动态模型

◎ 过程模型

◎ 功能模型

## 4+1模型P32



## 软件生命周期P40



# 第三章 软件体系结构风格

## 构件与连接件P45

构件P4

连接件是软件体系结构的一个组成部分，它通过对构件之间的交互规则的建模来实现构件之间的连接。与构件不同，连接件不需要编译。

## C/S和B/S架构的区别P67

C/S: P56

C/S软件体系结构是基于资源不对等，且为实现共享而提出来的，是20世纪90年代成熟起来的技术，C/S体系结构定义了工作站如何与服务器相连，以实现数据和应用分布到多个处理机上。

C/S体系结构有三个主要组成部分：数据库服务器、客户应用程序和网络。

B/S: P66

浏览器/服务器（B/S）风格就是上述三层应用结构的一种实现方式，其具体结构为：浏览器/Web服务器/数据库服务器。

B/S体系结构主要是利用不断成熟的WWW浏览器技术，结合浏览器的多种脚本语言，用通用浏览器就实现了原来需要复杂的专用软件才能实现的强大功能，并节约了开发成本。从某种程度上来说，B/S结构是一种全新的软件体系结构。

## CORBA架构的技术规范P67

# 第4章 软件体系结构描述

## C2概述P112

C2风格P56

## UML建模 P126

UML(Unified Modeling Language)是下面这些最好的建模方法中最好部分的集成:

商务流程模型(Work Flow)，对象建模方法，软构件建模思想

UML是一种用可视化方法对软件系统进行描述、实施和说明的标准语言。

支持用不同实现技术进行的软件开发全过程。

**UML主要内容P128 另附电商UML图**

# 第5章 动态软件体系结构

## 基于构件的动态系统结构模型 P155

# 第6章 SOA软件架构

## SOA的概念

面向服务的架构（SOA）是一个组件模型，它将应用程序的不同功能单元（称为服务）通过这些服务之间定义良好的接口和契约联系起来。接口是采用中立的方式进行定义的，它应该独立于实现服务的硬件平台、操作系统和编程语言。这使得构建在各种各样的系统中的服务可以以一种统一和通用的方式进行交互。

**SOA核心理念**

让企业应用彻底摆脱面向技术的解决方案的束缚，轻松应对企业商业服务变化、发展的需要。

 业务决定服务，服务决定技术。

## SOA与传统架构的区别 6.2.3，6.4.1

传统软件架构为了保持自由开放的特性，对软件的执行没有限制，一旦软件开始执行，就会获得相应的系统资源，并且认为软件的每个组成部分都是可靠的

在SOA中，软件的执行过程中不再像过去那样将要在本地安装所有的部分，而是动态地从网上根据需要请求服务，软件的各个部分分散在Internet中，需要时进行动态地组合。这种分散的软件结构，将安全责任由传统的软件使用者全部负担，转换为由软件的使用者和提供者共同负担，因而明确了责任，降低了单个用户的安全风险

传统软件包是被编写为独立的（self-contained）软件，即在一个完整的软件包中将许多应用程序功能整合在一起。实现整合应用程序功能的代码通常与功能本身的代码混合在一起。

SOA旨在将单个应用程序功能彼此分开，以便这些功能可以单独用作单个的应用程序功能或“组件”

让企业应用彻底摆脱面向技术的解决方案的束缚，轻松应对企业商业服务变化、发展的需要

（具体在PPT上）

## SOA基本特征6.4.2

**独立的功能实体**

SOA非常强调架构中提供服务的功能实体的完全独立自主的能力。传统的组件技术，如.NET Remoting，EJB，都需要有一个宿主来存放和管理这些功能实体；当宿主运行结束时这些组件的寿命也随之结束。当宿主本身或者其它功能部分出现问题的时候，在该宿主上运行的其它应用服务就会受到影响。

**大数据量低频率访问**

对于.NET Remoting，EJB这些传统的分布式计算模型而言，服务提供都是通过函数调用的方式进行的，一个功能的完成往往需要通过客户端和服务器多次函数调用才能完成。在Intranet的环境下，这些调用给系统的响应速度和稳定性带来的影响都可以忽略不计，但是在Internet环境下这些因素往往是决定整个系统是否能正常工作的一个关键决定因素。因此SOA系统推荐采用大数据量的方式一次性进行信息交换。

**基于文本的消息传递**

由于Internet中大量异构系统的存在决定了SOA系统必须采用基于文本而非二进制的消息传递方式。在COM、CORBA这些传统的组件模型中，从服务器端传往客户端的是一个二进制编码的对象，在客户端通过调用这个对象的方法来完成某些功能；由于基于文本的消息本身是不包含任何处理逻辑和数据类型的，因此服务间只传递文本，对数据的处理依赖于接收端的方式可以帮忙绕过兼容性这个的大泥坑。

**其他特征（百度百科）**

可从企业外部访问、随时可用、粗粒度的服务接口分级、松散耦合、可重用的服务、服务接口设计管理、标准化的服务接口、支持各种消息模式、精确定义的服务契约。

## SOA三要素 6.4.8

The coordination layer（协调层）

Information Entities（信息实体）

The relationship between BPM and SOA（BPM和SOA之间关系）

SOA是关于构建可在设计时未知的上下文中重用的软件组件

BPM（Business Process Management业务流程管理）是关于能够精确建模并可能更改使用企业组件的上下文

将BPM和SOA结合起来

基础是强大的理论的支持

一个大的部分仍然缺少：“状态”（恕我直言）

从业务流程到业务流程的定义

一旦基础到位我们应该看到特定领域语言（DSL）的出现，更接近用户的方式很多（不是程序员）

想想关于业务流程

## SOA优势

一，SOA可通过互联网服务器发布，从而突破企业内网的限制，实现与供应链上下游伙伴业务的紧密结合。通过SOA架构，企业可以与其业务伙伴直接建立新渠道，建立新伙伴的成本得以降低。

二，SOA与平台无关，减少了业务应用实现的限制。要将企业的业务伙伴整合到企业的“大”业务系统中，对其业务伙伴具体采用什么技术没有限制。

三， SOA具有低耦合性特点，业务伙伴对整个业务系统的影响较低。在企业与各业务伙伴关系不断发生变化的情况下，节省的费用会越来越多。

四， SOA具有可按模块分阶段进行实施的优势。可以成功一步再做下一步，将实施对企业的冲击减少到最小。

五， SOA的实施可能并不具有成本显著性。

# 第7+8章 设计模式分类

## 单例模式P7+8.1.7

**单例模式(Singleton Pattern)：**

单例模式确保某一个类只有一个实例，而且自行实例化并向整个系统提供这个实例，这个类称为单例类，它提供全局访问的方法。

单例模式的要点有三个：一是某个类只能有一个实例；二是它必须自行创建这个实例；三是它必须自行向整个系统提供这个实例。单例模式是一种对象创建型模式。单例模式又名单件模式或单态模式。

适用环境：

•系统只需要一个实例对象，如系统要求提供一个唯一的序列号生成器，或者需要考虑资源消耗太大而只允许创建一个对象。

•客户调用类的单个实例只允许使用一个公共访问点，除了该公共访问点，不能通过其他途径访问该实例。

•在一个系统中要求一个类只有一个实例时才应当使用单例模式。反过来，如果一个类可以有几个实例共存，就需要对单例模式进行改进，使之成为多例模式。

扩展：多例模式

多例模式的定义：

作为对象的创建模式，多例模式中的多例类可以有多个实例，而且多例类必须自己创建、管理自己的实例，并向外界提供自己的实例。

多例模式的特点：

所谓的多例模式（Multiton Pattern），实际上就是单例模式的自然推广。作为对象的创建模式，多例模式或多例类有如下的特点：

（1）多例类可有多个实例

（2）多例类必须自己创建、管理自己的实例，并向外界提供自己的实例。

（3）根据是否有实例上限分为：有上限多例类和无上限多例类。

桥接模式P7+8.3.2

**桥接模式(Bridge Pattern)：**

将抽象部分与它的实现部分分离，使它们都可以独立地变化。它是一种对象结构型模式，又称为柄体(Handle and Body)模式或接口(Interface)模式。

**适用环境：**

如果一个系统需要在构件的抽象化角色和具体化角色之间增加更多的灵活性，避免在两个层次之间建立静态的继承联系，通过桥接模式可以使它们在抽象层建立一个关联关系。

抽象化角色和实现化角色可以以继承的方式独立扩展而互不影响，在程序运行时可以动态将一个抽象化子类的对象和一个实现化子类的对象进行组合，即系统需要对抽象化角色和实现化角色进行动态耦合。

一个类存在两个独立变化的维度，且这两个维度都需要进行扩展。

虽然在系统中使用继承是没有问题的，但是由于抽象化角色和具体化角色需要独立变化，设计要求需要独立管理这两者。

对于那些不希望使用继承或因为多层次继承导致系统类的个数急剧增加的系统，桥接模式尤为适用。

## 适配器模式P7+8.5.5

适配器模式(Adapter Pattern) ：将一个接口转换成客户希望的另一个接口，适配器模式使接口不兼容的那些类可以一起工作，其别名为包装器(Wrapper)。适配器模式既可以作为类结构型模式，也可以作为对象结构型模式。

## **职责链模式P7+8.6.8**

**职责链模式(Chain of Responsibility Pattern)：**

避免请求发送者与接收者耦合在一起，让多个对象都有可能接收请求，将这些对象连接成一条链，并且沿着这条链传递请求，直到有对象处理它为止。由于英文翻译的不同，职责链模式又称为责任链模式，它是一种对象行为型模式。

**适用环境：**

有多个对象可以处理同一个请求，具体哪个对象处理该请求由运行时刻自动确定。

•在不明确指定接收者的情况下，向多个对象中的一个提交一个请求。

•可动态指定一组对象处理请求。

## 中介者模式P7+8.8.5

**中介者模式(Mediator Pattern)定义：**

用一个中介对象来封装一系列的对象交互，中介者使各对象不需要显式地相互引用，从而使其耦合松散，而且可以独立地改变它们之间的交互。中介者模式又称为调停者模式，它是一种对象行为型模式。

**适用环境：**

系统中对象之间存在复杂的引用关系，产生的相互依赖关系结构混乱且难以理解。

•一个对象由于引用了其他很多对象并且直接和这些对象通信，导致难以复用该对象。

•想通过一个中间类来封装多个类中的行为，而又不想生成太多的子类。可以通过引入中介者类来实现，在中介者中定义对象交互的公共行为，如果需要改变行为则可以增加新的中介者类。

## 观察者模式P7+8.10.6

**观察者模式定义：**

观察者模式属于行为型模式，其意图是定义对象间的一种一对多的依赖关系，当一个对象的状态发生改变时，所有依赖于它的对象都得到通知并被自动更新。这一个模式的关键对象是目标（Subject）和观察者（Observer）。

**适用环境：**

在以下情况下可以使用观察者模式：

•当一个抽象模型有两个方面，其中一个方面依赖于另一方面。将这二者封装在独立的对象中可以使他们各自独立地改变和复用。

•当对一个对象的改变需要同时改变其它对象，而不知道具体由多少对象有待改变。

•当一个对象必须通知其他对象，而它又不能假定其他对象是谁，换言之，你不希望这些对象是紧密耦合的。

# 第9章 云计算架构

## 云服务三个层次P9.1.6

根据提供的服务类型，将云计算应用(服务)分为三类：基础设施层(IaaS);应用平台层(PaaS);应用软件层(SaaS)

**Infrastructure as a Service（基础设施）**

以服务的形式提供虚拟硬件资源，如虚拟主机/存储/网络/安全等资源。

用于无需购买服务器、网络设备、存储设备，只需通过互联网租赁即可搭建自己的应用系统。

典型应用：Amazon Web服务 IDC

**Platform as a Service（应用平台）**

提供应用服务引擎，如互联网应用编程接口/运行平台等。

用户基于该应用服务引擎，可以构建该类应用。

典型应用：Google AppEngine, IBM PureScale Application System,SAE

Software as a Service（应用软件）

用户通过标准的Web浏览器来使用Internet上的软件。

**a基础设施即服务（IaaS，Infrastructrue as a Service）**

这种把主机集中管理，以市场机制通过虚拟化层对外提供服务，用按使用量收费的盈利模式，形成了云计算的基础层。这就是基础设施即服务（IaaS，Infrastructrue as a Service），构成了云计算的基础层。

b**平台即服务（PaaS，Platform as a Service）**

平台即服务（PaaS）是指把一个完整的应用程序运行平台作为一种服务提供给客户。在这种服务模式中，客户不需要购买底层硬件和平台软件，只需要利用PaaS平台，就能够创建、测试和部署应用程序。  
c**软件即服务（SaaS，Software as a Service）**

在其概念可以这样来定义：把软件部署为托管服务，用户不需要购买软件，可以通过网络访问所需要的服务，或者把各种服务综合成自己的需要，而客户按照使用量付费。SaaS的出现彻底颠覆了传统软件的运营模式。它不仅仅从价格上，交付模式上，实施风险上带来了明显改观。在云计算上，SaaS有了更好的发展空间。而云计算的推出，给SaaS提供了更好的生态环境。这就形成了云计算的第三层：软件即服务（SaaS，Software as a Service）。  
这三个层结合起来，就形成了典型的云计算的SPI模型。可以预期，在这个模型上，大量的创新企业可以获得更好的生存空间。

用户不必购买软件，只需按需租用软件

典型应用：Lotus Live, Salesforce.com

## 现有IT系统的主要问题P9.1.8

Web 业务应用全景图：

这种方式即是以应用为划分的“烟囱”结构，数据基于应用，并被锁定在应用系统中，形成一个个数据的孤岛。带来以下一些最基本的问题：

1、 数据并没有被作为一个单独的IT组成部分被规划和设计，而是作为应用系统的一部分，由于应用系统的供应商不同，并且其设计工作也缺乏相互之间的协调，因此，数据模型基本按照各个应用系统的功能需求进行设计和实现。

2、 由于缺乏有效的数据共享，在有些业务环节上，一个应用所需的数据无法从相关的其他应用系统中获得，而只好重复录入。

3、 另一方面，由于同一个数据可能存在多个数据源（从多个应用系统中被重复录入），由此导致了信息的不一致。

应用竖井和数据孤岛P9.2.3

这些应用系统按照传统的方式进行设计和开发，从而形成一个个的数据孤岛、应用的竖井等。以至不能形成完整的数据视图（例如相互有关联的数据分散于不同的应用系统，数据编码标准不一致，数据不一致和不可信等；让进行数据的整合和数据分析基本上不太可能）和应用系统集成复杂度和难度很高（没有好的设计规范和架构以及代码质量，基本上代码重用程度都很低；从界面集成、数据集成、应用集成这三个方向来做都很困难）；甚至出现无法集成或者为了集成而集成纯粹完成任务而已）

P9.2.5

对于一个性能要求很高的系统（例如高负载量和高数据量的系统，和将来会做的全省大集中系统和新一代的应用系统）我们会利用大量的硬件资源和相应的集群技术等技术进行相应的数据分区、集群和均衡负载来应对峰值的访问情况。即利用多台机器和技术形成Web服务器的集群，利用多台机器和技术形成应用服务器的集群，利用多台机器和技术形成数据库服务器的分区和集群等。要设计和很好地实施如此复杂的集群牵涉到非常多的技术，所以我会经常看到当系统压力上来的时候会遇到各种各样的问题并且这方面的问题很严重（因为很多应用开发商提出的思路基本上是购买更好的硬件来解决又带来了更多的硬件资源的浪费，而我们都知道仅仅是通过购买好的硬件是解决不了这个问题的）。

## 使用云计算技术后的新架构 P9.3.1等4张ppt

## 云计算解决的问题？？

大规模的存储空间，用于存储海量的数据。

随着业务的发展，新的数据源源不断地增加，存储空间需要相应扩大。用术语讲，这叫可扩展性，scalability。

系统的硬件设备必须便宜，通常使用大宗产品（commodity），譬如PC，或者价格便宜，中等性能的Dell server。

便宜的硬件设备，经常死机。所以在设计这个集群的时候，必须保证不能因为个别机器死机，导致整个系统的崩溃。也就是系统的稳定性要好，reliability。

# 第10章 大数据计算架构

## 大数据4V特征 P10.1.6

Volume: 数据量异常庞大，一般达到PB量级

Variety: 数据呈异构化，数据来源呈多样性

Velocity: 数据处理要求时效性

Value: 单个数据无价值，但大规模数据拥有巨大价值

## 离线批处理模型、内存计算模型、交互计算模型的区别P10.4.2



# 第11章 Hadoop简介

## Hadoop原理：HDFS及MapReduce P11.2.2

Hadoop最主要的就是Core，它又分为HDFS和MapReduce两个部分，前者提供分布式数据存储，后者提供任务的分发和归拢。其他组件都是围绕着这两个核心进行工作。

HDFS处理过程P11.2.7

HDFS 采用Master/Slave的架构来存储数据，这种架构主要由四个部分组成，分别为HDFS Client、NameNode、DataNode和Secondary NameNode。下面我们分别介绍这四个组成部分。

Client：就是客户端。

1、文件切分。文件上传 HDFS 的时候，Client 将文件切分成 一个一个的Block，然后进行存储。

2、与 NameNode 交互，获取文件的位置信息。

3、与 DataNode 交互，读取或者写入数据。

4、Client 提供一些命令来管理 HDFS，比如启动或者关闭HDFS。

5、Client 可以通过一些命令来访问 HDFS。

NameNode：就是 master，它是一个主管、管理者。

1、管理 HDFS 的名称空间。

2、管理数据块（Block）映射信息

3、配置副本策略

4、处理客户端读写请求。

DataNode：就是Slave。NameNode 下达命令，DataNode 执行实际的操作。

1、存储实际的数据块。

2、执行数据块的读/写操作。

Secondary NameNode：并非 NameNode 的热备。当NameNode 挂掉的时候，它并不能马上替换 NameNode 并提供服务。

1、辅助 NameNode，分担其工作量。

2、定期合并 fsimage和fsedits，并推送给NameNode。

3、在紧急情况下，可辅助恢复 NameNode。

MapReduce处理过程

Map-Reduce的目标是在面对节点失效的情况时能够保证大量的文件和数据依然可用。

初始化时数据被分割成许多分块的小数据，这些数据都是以<key,value>的形式存储。用户程序会分配一个master进程和许多worker进程。

任务开始时，Master将用户程序的工作分成两种类型的任务（map 任务和reduce任务），并将这些任务分配给相应的工人（workers）。Master的责任如下：

给map 工人和reduce工人分配相应的任务，检测是否有工人进程死掉，将Map任务处理后的结果通知给Reduce任务。

# 第12章 RHIN

区域系统架构扩展方案

# 课后习题：

**第一章**

1. **根据自己的经验，谈谈对软件危机的看法。**

软件危机是指软件生产方式无法满足迅速增长的计算机需求，

开发和维护过程出现的一系列问题。

以下几个原因导致：

（1）软件自身特点

 （2）开发人员的弱点

 （3）用户需求不明

 （4）缺乏正确理论指导

 （5）开发规模越来越大

 （6）开发复杂度越来越高

可以通过软件生命周期的模型和软件工具的使用来缓解危机，通过程序自动化和软件工业化生产的方法实现软件标准化的目标，进一步缓解软件危机带来的影响。

软件危机有利有弊，除了带来许多麻烦，也给我们带来许多挑战，克服危机的过程，我们在技术上和创新上都有了一个提升，也算是间接为软件产业的发展做了贡献。

**2、就项目管理方面而言，软件重用项目与非重用项目有哪些不同之处？**

使用软件重用技术可减少重复工作, 提高软件生产率, 缩短开发周期。同时，由于软构建大多经过严格的质量认证，因此有助于改善软件质量，大量使用构建， 软件的灵活性和标准化程度可得到提高。

**3、实际参与/组织一个软件重用项目的开发，然后总结你是如何组织该项目的开发的？**

参加了一个网页管理系统的开发，该项目重复使用已有的软件产品用于开发新的软件系统，以达到提高软件系统的开发质量与效率，降低开发成本的目的。在过程中使用了代码的复用、设计结果的复用、分析结果的复用、测试信息的复用等。

**4、为什么要研究软件体系架构**？

软件体系结构为软件系统提供了一个结构.行为和属性的高级抽象，由构成系统的元素的描述。这些元素的相互作用.指导元素成的模式以及这些模式的约束组成。不仅指定了系统的组织结构和拓扑结构，而且显示了系统需求和构成系统的元素之间的对应关系，提供了一些设计决策的基本原理。

**5、**根据软件体系结构的定义，你认为软件体系结构的模型应该有哪些部分组成？

构件: 可以是一组代码，如程序的模块也可以是一个独立的程序。如数据库的SQL服务器；

连接件:是关系的抽象，用以表示构件之间的相互作用。如过程调用、管道、 远程过程调用等；

限制：用于对构件和连接件的语义说明。

6、在软件体系结构的研究和应用中，你认为还有哪些不足之处？

（ 1）缺乏统一的软件体系结构的概念，导致体系结构的研究范畴模糊。

（ 2）ADL繁多，缺乏统一的ADL的支持。

（ 3）软件体系结构研究缺乏统一的理论模型支持。

（ 4）在体系结构描述方便，尽管出现了多种标准规范或建议标准，但仍很难操作。

（ 5）有关软件体系结构性质的研究尚不充分，不能明确给出一个良体系结构的属性或判定标准，没有给出良体系结构的设计指导原则，因而对于软件开发实践缺乏有力的促进作用。

（ 6）缺乏有效的支持环境软件体系结构理论研究与环境支持不同步，缺乏有效的体系结构分析、设计、方针和验证工具支持，导致体系结构应

用上的困难。

（ 7）缺乏有效的体系结构复用方案。

（ 8）体系结构发现方法研究相对欠缺。

## **第二章**

1、选择一个规模合适的系统，为其建立“4+1”模型。

逻辑视图（ Logical View ），设计的对象模型（使用面向对象的设计方法时）。

过程视图（ Process View ），捕捉设计的并发和同步特征。

物理视图（ Physical View ），描述了软件到硬件的映射，反映了分布式特性。

开发视图（ Development View ），描述了在开发环境中软件的静态组织结构。

架构的描述，即所做的各种决定，可以围绕着这四个视图来组织，然后由一些用例（use cases）或场景(scenarios)来说明，从而形成了第五个视图。

1. 引入了软件体系结构以后，传统软件过程发生了哪些变化？这种变化有什么好处？

变化：使得软件开发之前需要进行软件项目的风格分析，并对软件系统有一个前期的预估和决策。在开发过程中，可以套用之前已经存在的软件结构，使得开发过程更简单、具体和形象。

好处：

* 促进利益相关人员之间的交流 ：软件体系结构是一种常见的对系统的抽象，代码级别的系统抽象仅仅可以成为程序员的交流工具，而包括程序员在内的绝大多数系统的利益相关人员都借助软件体系结构来进行彼此理解、协商、达成共识或者相互沟通的基础。
* 系统设计的前期决策 ：软件体系结构是我们所开发的软件系统最早期设计决策的体现，而这些早期决策对软件系统的后续开发、部署和维护具有相当重要的影响。这也是能够对所开发系统进行分析的最早时间点。
* 可传递的系统级抽象：软件体系结构是关于系统构造以及系统各个元素工作机制的相对较小、却又能够突出反映问题的模型。由于软件系统具有的一些共通特性，这种模型可以在多个系统之间传递，特别是可以应用到具有相似质量属性和功能需求的系统中，并能够促进大规模软件的系统级复用。

3、软件体系结构的生命周期模型与软件生命周期模型有什么关系？

软件体系结构是贯穿于软件研发的整个生命周期的系统开发、运行、维护所实施的全部工作和任务的结构框架，给出了软件开发活动各阶段之间的关系。

软件体系结构的生命周期模型为软件生命周期模型提供了很好的结构依据和参考，也为其构建了很好的开发方式。

**第三章**

1. 层次系统结构和基于消息的层次系统结构有什么区别？

层次系统组织成一个层次结构，每一层为上层服务，并作为下层客户。在一些层次系统中，除了一些精心挑选的输出函数外，内部的层只对相邻的层可见。这样的系统中构件在一些层实现了虚拟机（在另一些层次系统中层是部分不透明的）。连接件通过决定层间如何交互的协议来定义，拓扑约束包括对相邻层间交互的约束。这种风格支持基于可增加抽象层的设计。允许将一个复杂问题分解成一个增量步骤序列的实现。由于每一层最多只影响两层，同时只要给相邻层提供相同的接口，允许每层用不同的方法实现，同样为软件重用提供了强大的支持。

而在基于消息的层次系统结构中构件不直接调用一个过程，而是触发或广播一个或多个事件。系统中的其它构件中的过程在一个或多个事件中注册，当一个事件被触发，系统自动调用在这个事件中注册的所有过程，这样，一个事件的触发就导致了另一模块中的过程的调用。这种风格的构件是一些模块，模块既可以是一些过程，又可以是一些事件的集合。过程可以用通用的方式调用，也可以在系统事件中注册一些过程，当发生这些事件时，过程被调用。这种风格的主要特点是事件的触发者并不知道哪些构件会被这些事件影响。这样不能假定构件的处理顺序，甚至不知道哪些过程会被调用，因此，许多隐式调用的系统也包含显式调用作为构件交互的补充形式。

2、试分析和比较B/S，二层C/S和三层C/S，指出各自的优点和缺点。

二层C/S结构的优点：

* C/S 体系结构具有强大的数据操作和事务处理能力，模型思想简单，易于人们理解和接受。
* 系统的客户应用程序和服务器构件分别运行在不同的计算机上，系统中每台服务器都可以适合各构件的要求，这对于硬件和软件的变化显示出极大的适应性和灵活性，而且易于对系统进行扩充和缩小。
* 在C/S体系结构中，系统中的功能构件充分隔离，客户应用程序的开发集中于数据的显示和分析，而数据库服务器的开发则集中于数据的管理，不必在每一个新的应用程序中都要对一个DBMS进行编码。将大的应用处理任务分布到许多通过网络连接的低成本计算机上，以节约大量费用

二层C/S结构的缺点：

* 开发成本较高
* 客户端程序设计复杂
* 信息内容和形式单一
* 用户界面风格不一，使用繁杂，不利于推广使用
* 软件移植困难
* 软件维护和升级困难
* 新技术不能轻易应用

三层C/S结构的优点：

* 允许合理地划分三层结构的功能，使之在逻辑上保持相对独立性，能提高系统和软件的可维护性和可扩展性。
* 允许更灵活有效地选用相应的平台和硬件系统，使之在处理负荷能力上与处理特性上分别适应于结构清晰的三层；并且这些平台和各个组成部分可以具有良好的可升级性和开放性。
* 应用的各层可以并行开发，可以选择各自最适合的开发语言。
* 利用功能层有效地隔离开表示层与数据层，未授权的用户难以绕过功能层而利用数据库工具或黑客手段去非法地访问数据层，为严格的安全管理奠定了坚实的基础。

三层C/S结构的缺点：

* 三层C/S结构各层间的通信效率若不高，即使分配给各层的硬件能力很强，其作为整体来说也达不到所要求的性能。
* 设计时必须慎重考虑三层间的通信方法、通信频度及数据量。这和提高各层的独立性一样是三层C/S结构的关键问题。

B/S体系结构的优点：

* 基于B/S体系结构的软件，系统安装、修改和维护全在服务器端解决。用户在使用系统时，仅仅需要一个浏览器就可运行全部的模块，真正达到了“零客户端”的功能，很容易在运行时自动升级。
* B/S体系结构还提供了异种机、异种网、异种应用服务的联机、联网、统一服务的最现实的开放性基础。

B/S体系结构的缺点：

* B/S体系结构缺乏对动态页面的支持能力，没有集成有效的数据库处理功能。
* B/S体系结构的系统扩展能力差，安全性难以控制。
* 采用B/S体系结构的应用系统，在数据查询等响应速度上，要远远地低于C/S体系结构。 B/S体系结构的数据提交一般以页面为单位，数据的动态交互性不强，不利于在线事务处理应用

3、组织或参与一个采用B/S和C/S混合体系结构的软件项目的开发，总结开发经验。

首先，开发者根据一定的原则，将系统的所有子功能分类，决定哪些子功能适合采用C/S，哪些适合采用B/S。适合采用C/S的子功能应具备以下特点：1安全性要求高；2要求具有较强的交互性；3使用范围小，地点固定；4要求处理大量数据。例如，仓库管理系统中的入库单、领料单的输入功能，财务系统中的凭证输入功能等等。而适合采用B/S的子功能应具备以下特点：1使用范围广，地点灵活；2功能变动频繁；3安全性、交互性要求不同。例如：企业内部信息发布功能，意见箱输入功能，公司财务分析表的查询功能，总裁决策支持系统中的查询功能等等。

相对于单独采用C/S或B/S，这种方案的优点在于：1保证敏感数据的安全性，特别是对数据库的修改和新增记录加强了控制；2经济有效地利用企业内部计算机的资源，简化了一部分可以简化的客户端；3既保证了复杂功能的交互性，又保证了一般功能的易用与统一；4系统维护简便，布局合理；5网络效率最高。

如果系统开发者在系统设计阶段决定采用这种C/S与B/S相结合的模式，那么在系统开发生命周期的如下各个阶段相对这种新模式都应有所响应。 在系统设计阶段主要考虑的是MIS系统平台选择问题。在详细设计阶段，系统开发者需要根据企业自身的业务特点，以及一定的选择原则，来决定各个子功能采用哪一种模式并在系统说明书上分别注明。在编码设计阶段，系统开发者需要针对采用不同模式的子功能，选用不同的编码方式(例如：C/S可以采用VB编程环境，而B/S采用ASP方法)，然后编译生成不同的客户应用及Web服务程序。在安装调试阶段，其特点主要体现在系统的物理结构上，即特定的客户应用程序将被安装在特定的使用者的客户端上，Web服务程序需要被安装在Web服务器上，而每个客户端上都将被安装上浏览器，同时，客户应用的使用者必须接受一定的培训。在软件维护阶段，针对不同模式的子功能应采取不同维护方式。

1. 组织或参与一个采用三层体系结构的软件项目的开发，总结开发经验。

三层体系结构包括：用户界面表示层(USL) 、业务逻辑层(BLL) 、数据访问层(DAL) 。各层的作用 ：1：数据访问层:主要是对原始数据(数据库或者文本文件等存放数据的形式)的操作层，而不是指原始数据，也就是说，是对数据的操作，而不是数据库,具体为业务逻辑层或表示层提供数据服务。2：业务逻辑层:主要是针对具体的问题的操作，也可以理解成对数据层的操作,对数据业务逻辑处理，如果说数据层是积木，那逻辑层就是对这些积木的搭建。3：表示层:主要表示WEB方式,也可以表示成WINFORM方式,WEB方式也可以表现成:aspx, 如果逻辑层相当强大和完善,无论表现层如何定义和更改,逻辑层都能完善地提供服务。

三层是指逻辑上的三层，即使这三个层放置到一台机器上。 三层体系的应用程序将业务规则、数据访问、合法性校验等工作放到了中间层进行处理。在保证客户端功能的前提下，为用户提供一个简洁的界面。这意味着如果需要修改应用程序代码，只需要对中间层应用服务器进行修改，而不用修改成千上万的客户端应用程序。“中间业务层”的用途有很多，例如：验证用户输入数据、缓存从数据库中读取的数据等等....但是，“中间业务层”的实际目的是将“数据访问层”的最基础的存储逻辑组合起来，形成一种业务规则。要保证“数据访问层”的中的函数功能的原子性！即最小性和不可再分。“数据访问层”只管负责存储或读取数据就可以了。

我们用三层结构主要是使项目结构更清楚，分工更明确，有利于后期的维护和升级。它未必会提升性能，因为当子程序模块未执行结束时，主程序模块只能处于等待状态。这说明将应用程序划分层次，会带来其执行速度上的一些损失。但从团队开发效率角度上来讲却可以感受到大不相同的效果。需要说明一下，三层结构不是.NET的专利，也不是专门用在数据库上的技术。它是一种更加普适的架构设计理念。此种架构要在数据库设计上注意表之间的关系，尽力满足主与子的关系。在功能上对用户要有一定的限制，不要表现在对于子表的删除操作一定要慎重，以免造成主表与子表的数据在逻辑上出现的主表的外键在子表中没有相对应的值。

“三层结构”开发模式，入门难度够高，难于理解和学习。这是对于初学程序设计的人来说的。以这种模式开发出来的软件，代码量通常要稍稍多一些。这往往会令初学者淹没在茫茫的代码之中。望之生畏，对其产生反感，也是可以理解的。

5、SIS和DSSA分别用在哪些场合？

SIS（互连系统构成的系统及其体系结构System and architecture of interconnection system）：交互式网络互连服务。

DSSA（特定域软件体系结构domain-specific software architecture）：DSSA就是专用于一类特定类型的任务（领域）的、在整个领域中能有效地使用的、为成功构造应用系统限定了标准的组合结构的软件构件的集合。

即特定领域的软件体系结构。

补充

1.DSSA只对某一个领域进行设计专家知识的提取，存储和组织，但可以同时使用多种体系结构风格；而在某个体系结构风格中进行体系结构设计专家知识的组织时，可以将提取的公共结构和设计方法扩展到多个领域。

2.DSSA的特定领域参考体系结构通常选用一个或多个适合所研究领域的体系结构风格，并设计一个该领域专用的体系结构分析设计工具。

3．体系结构风格的定义和该风格应用的领域是直交的，提取的设计知识比用DSSA提取的设计专家知识的应用范围要广。

4．DSSA和体系结构风格是互为互补的两项技术。

## 6、在软件开发中，采用异构结构有什么好处，其负面影响有哪些？

好处：

采用异构结构可以更好的处理当实际问题需要用不同的结构去解决这种情况。

关于软件包，框架、通信以及其他一些体系结构上的问题，目前存在多种标准。即使在某段时间内某一种标准占统治地位，但变动最终是绝对的。采用异构结构可以很好的处理这些变动。

负面影响：

实际工作中，我们总会遇到一些遗留下来的代码它们仍有效用但是却与新系统有某种程度上的不协调。然在许多场合，将技术与经济综合进行考虑时，总是决定不再写它们。

即使在某一单位中，规定了共享共同的软件包或相互关系的一些标准，仍会存在解释或表示习惯上的不同。

1）结构有不同的处理能力的强项和弱点，一个系统的体系结构应该根据实际需要进行选择，以解决实际问题。

2）软件包，框架，通信以及其他一些体系机构上的问题，目前存在者多中标准。即使再某一段时间内某一标准占据着统治地位，但变动最终是绝对的。

3）工作中，我们总会遇到一些遗留下的代码，它们仍有效用，但是却与新系统有某种程度上的不协调。然而在很多场合，将技术与经济综合进行考虑时，总是决定不重写它们。

4）在某一单位中，规定了共享共同的软件包或相互关系的一些标准，仍会存在解释或表示习惯上的不同。

负面影响：大多数应用程序只使用10%的代码实现系统的公开功能，剩下90%的代码完成系统管理功能：输入和输出，用户界面，文本编辑，基本图表，标准对话框，通信，数据确认和旁听追踪，特定领域的基本定义等。

**第四章**

## 1、体系结构描述有哪些方法？有哪些标准和规范？

体系结构描述方法：

(1) 图形表达工具：由矩形框和有向线段组合而成。

(2) 模块内连接语言：将一种或几种传统程序设计语言的模块连接起来的语言。

(3) 基于软构件的系统描述语言：将软件系统描述成一种是由许多以特定形式相互作用的特殊软件实体构造组成的组织或系统。

(4) 软件体系结构描述语言：是参照传统程序设计语言的设计和开发经验，针对软件体系结构特点重新设计、开发和使用的专门的软件体系结构描述语言。

体系结构描述的标准和规范：

IEEE P1471 ：于2000年9月21日通过IEEE-SA标准委员会评审。

软件体系结构标准定义了一系列的软件开发边界，而开源软件或软件提供商正是应用这些软件边界来进行软件的开发。例如，我们日常软件开发中所用到的应用服务器、Ajax工具包、数据库或Web服务等等。

## 2、体系结构描述语言与程序设计语言有什么区别？

ADL与其他的语言比较具有以下能力：

1. 构造能力：ADL能够使用较小的独立体系结构元素来建造大型软件系统；
2. 抽象能力：ADL使得软件体系结构中的构件和连接件描述可以只关注他们的抽象特性，而不管其具体的实现细节；
3. 重用能力：ADL使得组成软件系统的构件，连接件甚至是软件体系结构都成为软件系统开发和设计的可重用部件；
4. 组合能力：ADL使得其描述的每一系统元素都有其自己的布局结构，这种描述布局结构的特点使得ADL支持软件系统的动态变化组合；
5. 异构能力：ADl允许多个不同的体系结构描述关联存在；
6. 分析和推理能力：ADL允许对其描述的体系结构进行多种不同的性能和功能上的多种推理分析。ADL与需求语言的区别在于后者描述的是问题空间，而前者描述的是接空间。ADL与建模语言的区别在于后者对整体行为的关注要大于部分的关注，而ADl集中在构建的表示上。

## 3、选择一个规模适中的系统，使用UML为其建模。

为一个网上购物系统利用UML为其建模，如下：

统一开发过程RUP把整个软件开发过程分为初始、细化、构造、交付四个阶段，具有用例驱动、以架构为中心、迭代和增量的特点。同其它软件开发方法相比较，RUP具有自身独特的优势，为软件开发提供了重要的方法论指导根据对网上购物系统的体系结构及建模分析，采用UML作为建模语言，结合RUP的基本开发过程，提出适合网上购物系统开发的建模过程。该过程遵循了RUP四个阶段的理论，主要是对初始和细化两个阶段进行了详细的分析。整个过程包括业务建模、需求建模、对象建模、数据库建模和物理建模等五个步骤，每个步骤都会生成一定的系统模型，并用相应的UML图来描述这些模型。在建模过程中，[论文](http://lunwen.1kejian.com/)采用了RUP中迭代增量式的开发思想，把系统的建模进一步分解为迭代，一个迭代是一个从系统的业务建模到物理建模的完整过程，每一个迭代都会产生一个模型版本，是最终模型的一个子集，它增量式地发展，从一个迭代过程到另一个迭代过程，直到成为系统的最终模型。  
1 业务建模  
　　业务建模用于对网上购物系统环境的业务过程进行建模。系统分析人员通过网上购物系统的业务建模能够了解系统所处的环境和业务过程，业务建模能够将这些信息进行体现，并表现环境中存在或可以觉察到的过程，从而详细说明网上购物系统所要支持的业务过程。业务建模既确定了业务过程涉及到的业务对象和领域对象，又确定了每个业务过程所需要的资源和能力，包括人员、每个人员的职责和执行的操作、过程的执行方式和协作等。这些信息对于下一步的需求建模是非常重要和有用的。  
　　业务建模一般通过业务过程图来进行描述。业务过程图是对事件逻辑的归类，这些事件被认为是业务的基本元素。其目的是将整个业务领域作为一个过程集进行描述，而不关心过程的次序或单个过程之间的交互作用。业务过程图不必严格精密，它应该全面而不是精确。重要的是，通过查看业务过程图，系统分析人员、设计人员、开发人员和用户能够迅速获得关于业务范围和活动的总体印象。  
2 需求建模  
　　需求建模的主要工作是获得系统的需求，建立待开发系统的模型。而用例则有助于更好地了解系统需求并以规范化的格式进行描述。需求建模就是要以用例的方式来描述系统的功能，其主要工作成果是用例模型。采用用例模型来描述进行需求建模的主要过程如下：（1）确定所要开发系统的参与者，参与者可以是人，也可以是与系统交互的外部系统。网上购物系统的参与者主要有[管理](http://guanli.1kejian.com/)员、工作人员、顾客、支付系统等。（2）从执行者的角度出发，分析他和系统需要进行的交互作用，并从这些交互过程中抽象出用例。从顾客的角度出发，网上购物系统一般有以下用例：用户登录、用户注册、浏览商品、搜索商品、购买商品、下订单、支付等；从系统[管理](http://guanli.1kejian.com/)员的角度出发，网上购物系统一般有以下用例：用户登录、用户[管理](http://guanli.1kejian.com/)、商品[管理](http://guanli.1kejian.com/)、订单[管理](http://guanli.1kejian.com/)等。（3）对每一用例确定其主要的业务过程。例如“用户登录”用例的业务过程为顾客、系统[管理](http://guanli.1kejian.com/)员、工作人员等通过用户登录可以获得相应的服务；“支付”用例的业务过程为顾客为所够买物品选择付款方式进行付款。（4）以信息流为中心逐步形成完整的用例模型。网上购物系统的完整用例模型包括很多的用例图，其中既包括系统的顶层用例图，也包括各种细化的用例图。  
3 对象建模  
　　确定了系统的需求分析、得出系统的用例模型以后，需要进行的主要任务就是对系统进行对象建模。对象建模的主要工作是把需求建模阶段产生的用例模型转化为系统的静态结构模型和动态行为模型，使建立的系统在特定的环境下完成需求分析中的任务和功能，有利于系统的实现和迭代。这其中主要包括静态结构建模和动态对象建模两部分。  
4 数据库建模  
　　数据库建模是从计算机系统的角度对系统所要处理的数据进行建模。数据库系统是整个网上购物系统的基础，数据库建模的好坏直接影响到整个系统的结构、实现的复杂程度、性能、安全性和可维护性等。传统的逻辑数据库建模工具“实体一联系(E．R)""图只针对数据建模，不能对行为建模。而UML的类图能够更好的用于数据库建模。UML的类图不但对数据建模，而且能对行为建模，这些行为在物理数据库中被设计成触发器和存储过程。即使是关系数据库，也可以在类图设计落实后，再采取标准方法把类图映射到具体的关系模型。从类图到关系模型的转换，按照一个类映射为一个关系的原则进行，而类的属性即为关系的属性，标识的标识符即为关系的主键。  
5 物理建模  
　　物理建模用于网上购物系统建模过程的最后阶段，是对网上购物系统的物理方面进行建模。它使用UML中的组件图描述网上购物系统中代码组件的物理结构及各个组件之间的依赖关系，使用配置图定义网上购物系统的软硬件结构及通信机制，表示软硬件系统之间的协作关系。  
　　以上五个步骤是根据RUP的四个阶段细化的结果，分别对应了RUP中相应的核心工作流。其中业务建模对应RUP的业务建模工作流，需求建模对应RUP的需求建模工作流，对象建模和数据库建模对应RUP的分析和设计工作流，物理建模对应RUP的实施工作流，对开发完成的系统进行测试、部署和[管理](http://guanli.1kejian.com/)分别对应RUP的测试工作流、部署工作流等。

**第五章**

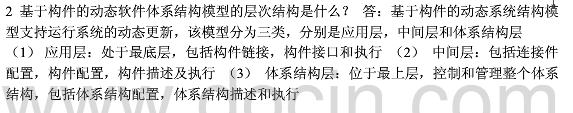
1、什么是动态软件体系结构？动态软件体系结构与静态软件体系结构有什么区别？

动态软件体系结构的动态性包括：交互性动态性，结构化动态性，体系结构动态性。

由于系统需求，技术，环境，分布等因素的变化而最终造成软件体系结构的变动，称之为软件体系结构演化。软件系统在运行时刻的体系

结构变化称之为软件体系结构的动态性，动态软件体系结构的动态性包括：交互性动态性，结构化动态性，体系结构动态性。

1. 基于构件的动态软件体系结构模型的层次结构是什么？



SOA和微服务

**SOA**的提出是在企业计算领域，就是要将紧耦合的系统，划分为面向业务的，粗粒度，松耦合，无状态的服务。服务发布出来供其他服务调用，一组互相依赖的服务就构成了SOA架构下的系统。基于这些基础的服务，可以将业务过程用类似BPEL流程的方式编排起来，而BPEL反映的是业务处理的过程，这些过程对于业务人员更为直观，调整也比hardcode的代码更容易。当然企业还需要对服务治理，比如服务注册库，监控管理等。我们知道企业计算领域，如果不是交易系统的话，并发量都不是很大的，所以大多数情况下，一台服务器就容纳将许许多多的服务，这些服务采用统一的基础设施，可能都运行在一个应用服务器的进程中。虽然说是面向服务了，但还是单一的系统。

**微服务**架构大体是从互联网企业兴起的，由于大规模用户，对分布式系统的要求很高，如果像企业计算那样的系统，伸缩就需要多个容纳续续多多的服务的系统实例，前面通过负载均衡使得多个系统成为一个集群。但这是很不方便的，互联网企业迭代的周期很短，一周可能发布一个版本，甚至可能每天一个版本，而不同的子系统的发布周期是不一样的。而且，不同的子系统也不像原来企业计算那样采用集中式的存储，使用昂贵的Oracle存储整个系统的数据，二是使用MongoDB，HBase，Cassandra等NOSQL数据库和Redis，memcache等分布式缓存。那么就倾向采用以子系统为分割，不同的子系统采用自己的架构，那么各个服务运行自己的Web容器中，当需要增加计算能力的时候，只需要增加这个子系统或服务的实例就好了，当升级的时候，可以不影响别的子系统。这种组织方式大体上就被称作微服务架构。微服务与SOA相比，更强调分布式系统的特性，比如横向伸缩性，服务发现，负载均衡，故障转移，高可用。互联网开发对服务治理提出了更多的要求，比如多版本，比如灰度升级，比如服务降级，比如分布式跟踪，这些都是在SOA实践中重视不够的。Docker容器技术的出现，为微服务提供了更便利的条件，比如更小的部署单元，每个服务可以通过类似Node.js或Spring Boot的技术跑在自己的进程中。可能在几十台计算机中运行成千上万个Docker容器，每个容器都运行着服务的一个实例。随时可以增加某个服务的实例数，或者某个实例崩溃后，在其他的计算机上再创建该服务的新的实例。