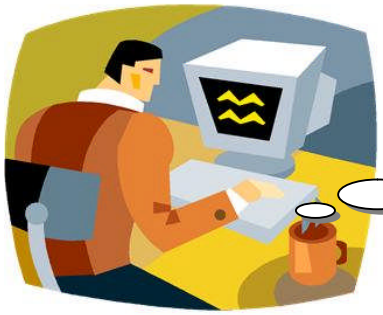
系统架构师学习笔记

阅读前注意：本文档的章节不按正常排序，有些章节没有，这是为什么？

——因为这个是笔记，不是书。可能作者只记下认为有价值的东西^_^



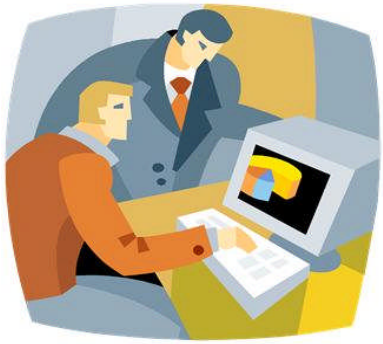
我是一个 Java 软件开发工程师，最近很迷茫，软件开发过程需要学的东西太多，又看不到未来，怎么办呢？有没有一个知识体系或者发展路线什么的？

很多开发人员是想往架构师的途径发展，那么架构师到底需要什么样的知识呢？

首先，针对 Java 开发工程师来说，JDK 里边的每个包是按照什么原则分类的，每个包里边都有哪些类/接口可用，每个类/接口有什么方法？分别在什么时候使用？完整学习和思考这些内容后，你可以成为优秀的 Java 开发工程师了。

这之后，您就可以去熟悉本文 1～9 章的内容，包括：**计算机基础知识、信息化内涵、软件开发过程、软件架构设计、UML 技术、XML 技术、设计模式、面向构件的软件设计**。熟悉完这些，您开始走向架构师的征途了。

看完 1～9 章，可以开始架构设计了吗？



看完 1～9 章，并且从其他书去更深层次地了解这些知识后，步入架构师的门槛还需要看 **10～15 章**的内容，就是关于当前的**软件开发典型架构、信息安全、系统安全、系统可靠性**方面的内容，然后可以学着第 14 章进行一次**架构的实践**，同时通过 15 章学习一些架构师**管理方面的实践**。这就基本学习完并熟悉架构师的全部知识了。

如果您要深入了解并掌握架构，您必须看 **16～20 章**的内容，这几张阐述了当前最常用的**层次式架构、企业集成架构、面向切面的编程架构、嵌入式系统架构**以及**面向服务(SOA)的架构**。

本文所有内容都来自网络，本人只做收集和整理，版权归原作者所有。

龙小宝（ebizs@hotmail.com）收集整理， 2010 年 9 月 14 日

第一章 架构师

1.1.1 系统架构的概念

现代信息系统“架构”三要素：构件、模式、规划；规划是架构的基石，也是这三个贡献中最重要的。

架构本质上存在两个层次：概念层，物理层。

1.2.1 系统架构师的定义

负责 理解、管理 并最终确认和评估 非功能性系统需求，给出开发规范，搭建系统实现的核心架构，对整个软件架构、关键构建、接口 进行总体设计 并澄清关键技术细节。

主要着眼于系统的“技术实现”，同时还要考虑系统的“组织协调”。

要对所属的开发团队有足够的了解，能够评估该开发团队实现特定的功能需求目标和资源代价。

1.2.2 系统架构师技术素质

对软件工程标准规范有良好的把握。

1.2.3 系统架构师管理素质

系统架构师是一个高效工作团队的创建者，必须尽可能使所有团队成员的想法一致，为一个项目订制 清晰的、强制性的、有元件的目标 作为整个团队的动力；

必须提供特定的 方法和模型 作为理想的技术解决方案；

必须避免 犹豫，必须具备及时解决技术问题的 紧迫感和自信心。

1.2.4 系统架构师 与 其他团队角色 的 协调

系统分析师，需求分析，技术实现

系统架构师，系统设计，基于环境和资源的系统技术实现

项目管理师，资源组织，资源实现

由于 职位角度出发产生冲突制约，不可能很好地给出 开发规范，搭建系统实现的 核心架构，并澄清技术细节，扫清主要难点。

所以 把架构师定位在 项目管理师与系统分析师 之间，为团队规划清晰的目标。

对于大型企业或项目，如果一人承担多个角色，往往容易发生顾此失彼的现象。

1.3 系统架构师知识结构

需要从大量互相冲突的系统方法和工具中 区分出 哪些是有效的，那些是无效的。

1.4 从开发人员到架构师

总结自己的架构模式，深入行业总结规律。

几天的培训不太可能培养出合格的软件架构师，厂商的培训和认证，最终目的是培养自己的市场，培养一批忠诚的用户或产品代言人，而不是为中国培养软件架构师。

第二章 计算机基础

《计算机网络基础知识》

计算机系统 由 硬件和软件组成，软件通常分为 系统软件和应用软件。

系统软件支持应用软件的运行，为用户开发应用软件提供平台，用户可以使用它，但不能随意修改它。

常用的系统软件有 操作系统、语言处理程序、连接程序、诊断程序、数据库 等。

应用软件指 计算机用户利用 软硬件资源 为某一专门的应用目的而开发的软件。

2.1 操作系统基础知识

操作系统 Operating System，是计算机系统的核心系统软件。

2.1.1 操作系统的原理、类型、结构

1、操作系统定义

硬件资源包括 中央处理器、存储器、输入输出设备。

软件资源是以 文件形式保存在存储器上的 程序和数据。

操作系统既 有效组织和管理 系统中各种 软硬件资源，合理地组织计算机系统的工作流程，又控制程序的执行，为用户使用计算机 提供了一个 良好的环境和友好的接口。

2、操作系统分类

按功能不同分：单用户操作系统、批处理操作系统；分时操作系统、实时操作系统；网络操作系统、分布式操作系统；嵌入式操作系统。

3、操作系统的特征

并发性、共享性、虚拟性、不确定性。

4、操作系统的功能

进程管理、文件管理、存储管理、设备管理、作业管理。

2.1.2 处理机 与 进程管理

1、进程的定义及其分类

进程通常由 程序、数据、进程控制块 PCB 组成。

2、进程的状态转换与控制

就绪、运行、阻塞。

进程控制是通过 进程控制原语实 现的，进程控制原语主要有：创建原语、撤销原语、挂起原语、激活原语、阻塞原语、唤醒原语。

注：原语不可分割，不允许中断。

3、进程互斥与同步 以及 P/V 操作

同步是使在异步环境下的各进程按一定的 顺序和速度 执行。

互斥 要保证临界资源 一次只能提供一个进程使用，称为 临界资源 CR。

PV 操作是低级通信原语，在执行期间不可分割，P 表示申请一个资源，V 表示释放一个资源。

P 操作定义：S:=S-1，若 S>=0，则执行 P 操作的进程继续执行，否则若 S<0， 则置该进程为阻塞状态（因为无可用资源），并将其插入阻塞队列。

V 操作定义：S:=S+1，若 S>0， 则执行 V 操作的进程继续执行，否则若 S<=0，则从阻塞状态唤醒一个进程，并将其插入就绪队列，然后执行 V 操作的进程继续执行。

4、进程通信与管程

控制信息的交换称为低级通信，数据的交换称为高级通信。

高级通信的类型有 共享存储系统、消息传递系统、管道通信。

在任一时刻最多只有一个进程能够真正地进入管程，其他的只能等待。

5、进程调度与死锁

产生死锁的四个必要条件：互斥条件、请求保持条件、不可剥夺条件、环路条件。

预防策略，破坏死锁的四个必要条件之一。

6、线程

线程是进程中的一个实体，是被系统独立分配和调度的基本单位。

线程只拥有一些运行中必不可少的资源。

同一个进程中的多个线程可以并发执行，线程具有：就绪、运行、阻塞，三个基本状态。

2.1.3 存储管理

存储器的发展方向是：高速、大容量、小体积。

存储管理的主要任务是：如何提高主存的 利用率、扩充主存 以及对主存信息实现有效保护。

2.1.4 设备管理

设备管理的目标是：提高设备的利用率，为用户提供方便统一的界面。

磁盘调度算法：先来先服务 FCFS、最短寻道时间优先 SSTF、扫描算法 SCAN。

2.1.5 文件管理

随机访问是指对文件中的信息可以按任意次序随机读写文件中的信息。

文件控制块 FCB，描述和控制文件的数据结构。

2.1.6 作业管理

常用的作业调度算法有：先来先服务、短作业优先、相应比高优先、优先级调度算法、均衡调度算法。

2.1.7 网络操作系统 NOS

网络操作系统分为：集中模式、客户机/服务器模式、对等模式。

现代操作系统已经把网络功能包含到操作系统的内核中，作为操作系统核心功能的一个组成部分。

2.2 数据库

2.2.1 关系数据库基础

数据库的三要素：数据结构、数据操作、数据约束条件。

特别需要指出的是，E-R 模型强调的是 语义。

关系数据库设计理论的核心是 数据间的函数依赖，衡量的标准是 关系规范化的程度及分解的无损连接 和 保持函数依赖性。

数据依赖包括：函数依赖、非平凡的函数依赖、平凡的函数依赖、完全函数依赖、部分函数依赖、传递依赖、码、主属性、非主属性、外码、值依赖定义、函数依赖的公理系统。

事务是数据库环境中 不可分割 的逻辑工作单位。

四个特性：原子性、一致性、隔离性、持久性，ACID。

SQL 语言中事务定义语句有三条：BEGIN TRANSACTION 事务开始、COMMIT 事务提交、ROLLBAK 事务回滚。

并发操作是指：在多用户共享系统中，用户可能同时对同一数据库进行操作。

带来的问题主要有：丢失更新、不可重复读、读脏数据。

并发控制主要技术是封锁：排他锁（简称 X 锁、写锁）、共享锁（简称 S 锁、读锁）。

保护数据库的关键技术在于 建立冗余数据、即 备份数据。

方法是：数据转储、建立日志。

2.2.2 关系数据库设计

需求分析、概念结构设计、逻辑结构设计、物理结构设计、应用程序设计、运行维护。

E-R 方法的数据库概念结构设计可分三步：设计局部 E-R 模型、设计全局 E-R 模型、全局 E-R 模型优化。

2.2.3 分布式数据库系统

满足 分布性、逻辑相关性、场地透明性、场地自治性 的数据库系统被称为 完全分布式数据库系统。

分布式数据库系统的特点：数据的集中控制性、数据独立性、数据冗余可控性、场地自治性、存取有效性。

4 层模式划分为：全局外层、全局概念层、局部概念、局部内层，各层还有相应的 层间映射。

2.2.4 商业智能

一般认为：数据仓库、连机分析处理、数据挖掘技术 是 商业智能 BI 的三大组成部分。

数据仓库的关键特征：面向主题、集成的、非易失的、时变的。

三层结构：数据仓库服务器、OLAP 服务器（连机分析处理 服务器）、前端工具。

数据仓库的实现步骤：规划、需求研究、问题分析、数据的 抽取 清洗 集成 装载、数据仓库设计、数据仓库管理、分析报表查询、数据仓库性能优化、数据仓库部署发布。

切片、切块、下钻、上卷、旋转 等多维度分析与跨维度分析。

OLAP 系统架构主要分为：基于关系数据库的 ROLAP、基于多维数据库的 MOLAP、基于混合数据组织的 HOLAP。

数据挖掘是在 没有明确架设的前提下 去挖掘信息、发现知识。

所得的信息应具有 先知、有效、实用，三个特征。

主要功能有 5 类：自动预测趋势和行为、关联分析、聚类、概念描述、偏差检测。

2.3 计算机网络基础知识

计算机网络

按通信距离分 广域网、局域网、城域网；按信息交换方式分 电路交换网、分组交换网、综合交换网；按拓扑结构分 星型网、树形网、环形网、总线型网；按传输带宽分 基带网、

宽带网；

按使用范围分 公用网、专用网；按通信传播方式分 广播式、点到点式……

OSI/RM：把复杂的问题分解开，保持了层次之间的独立性。

物理层、数据链路层、网络层、传输层、会话层、表示层、应用层。

2.3.2 计算机网络

1、广域网、局域网、城域网

广域网又称远程网，覆盖范围广，传输速率相对低，以数据通信为主要目的 的数据通信网。数据传输可靠性随着传输介质不同而不同、拓扑结构复杂。

有公共交换电话网、各种公用数据网。

局域网是指传输距离有限，传输速度较高，以共享网络资源为目的的网络系统，数据传输可靠 误码率低，网络控制一般为分布式，总线拓扑、环形拓扑、星型拓扑、混合型。

城域网 是一种较大范围的高速网络。

网络拓扑结构：网络中通信线路和节点的几何排序，反映各节点之间的结构关系，影响着整个网络的 设计、功能、可靠性、通信费用 等重要方面。

局域网和城域网 都是 IEEE802 标准，决定局域网主要技术有：传输介质、拓扑结构、介质访问控制方法。

决定了传数据的类型、网络响应时间、吞吐率、利用率，以及网络应用。

最重要的是 介质控制访问方法。（CSMA/CD）

无线局域网具有以下优点：安装便捷、使用灵活、经济解约、易于扩展。IEEE8.2.11

2、网络互联

网络互联目的是 使一个网络的用户能访问其他网络的资源，使不同网络上的用户能够互相通信、交换信息。

网络互联设备的作用是 连接不同网络。

传输介质是信号传输的 媒体，常用的介质分为 有限介质 和 无线介质。局域网中，其基本组成部件为 服务器、客户机、网络设备、通信介质、网络软件 等。

3、Internet 及应用

世界上规模最大、覆盖面最广 且 最具影响力 的 计算机互联网络，它将分布在世界各

地的计算机利用开放系统互连协议连接在一起，用来进行数据传输、信息交换、资源共享。

TCP/IP 作为 Internet 的核心协议，已被广泛应用于局域网和广域网中，主要特性为：逻辑编址、路由选择、域名解析、错误检测、流量控制、对应用程序的支持 等。

TCP/IP 是一个协议族，网际层除了 IP 协议外，还有 ICMP、ARP、RARP 等几个重要协议……

Internet 的地址主要有两种书写形式：域名格式、IP 地址格式。

www 也成万维网/全球网，是指在 Internet 上 以超文本为基础形成的 信息网。采用统一的资源定位器 URL 和 图文声并茂的用户界面。

2.3.3 网络管理与网络安全

1、网络管理

网络管理是对计算机网络的 配置、运行状态、计费 等进行管理。它提供了 监控、协调、测试 各种网络资源 以及 网络运行状况的手段，还可以提供 安全处理和积分 等功能。

OSI 网络协议标准中定义了 网络管理的 5 大基本功能：配置管理、性能管理、故障管理、安全管理、计费管理。

实际上还应该包括 网络规划、网络操作人员管理 等。

2、计算机网络安全

计算机网络安全是指 计算机、网络系统的 硬件、软件、数据 收到保护，不因偶然或恶意的原因而遭到 破坏、更改、泄漏，确保系统能 连续、可靠 地运行，使网络服务 不中断。

网络安全从本质上讲 就是 网络上的 信息安全。

信息的 传输、存储、访问 提供安全保护，以 防止信息被 窃取、篡改、非法操作。

信息安全的基本要素是 保密性、完整性、可用性、真实性、可控性。

完整的信息安全保障体系应包括：保护、检测、响应、恢复。

信息安全术语：密码学、鉴别、Kerberos 鉴别、公钥基础设施、数字签名、访问控制。

3、VPN

所谓虚拟专用网，是建立在公用网上，没有专用物理连接，而通过 ISP 提供的公共网络来实现通信，VPN 内部用户可以实现安全通信。

关键技术：隧道技术、加密技术、密钥管理技术、身份认证技术。

解决方案：内联网 VPN、外连网 VPN、远程接入 VPN。

2.3.4 网络工程

网络规划、网络设计阶段、工程组织、实施阶段、维护阶段。

2.3.5 存储及负载均衡技术

RAID 磁盘阵列，目的是 建立数据冗余、增强容错、提高容量、增进性能。

网络存储体系结构大致分为三种：直接式存储 DAS、网络连接存储 NAS、存储区域存储 SAN。

负载均衡 LoadBalance 从结构上分为：本地负载均衡、全局负载均衡。

一般情况下从 传输链路聚合、采用更高层网络交换技术、设置服务器集群策略 三个角度实现。

集群 Cluster，大多数模式下，集群中所有的计算机拥有一个共同的名称，各节点服务器通过一个内部局域网相互通讯,集群内任一系统上运行的服务都可被所有的网络客户所使用，当一台

节点服务器发生故障时,这台服务器上所运行的应用程序将在另一节点服务器上被自动接管，客户也能很快自动地连接到新的应用服务器上。

2.4 多媒体技术及其应用

媒体有两种含义：信息的载体、存储信息的实体。

根据 ITU-T（原 CCITT）建议，媒体有 5 种：感觉媒体、表示媒体、显示媒体、存储媒体、传输媒体。

International Consultative Committe On Telecommunication And Telegraphy，CCITT，国际电报电话咨询委员会。

多媒体技术是指：以数字化为基础，对多种媒体信息进行 采集、编码、存储、传输、

处理、表现，使之建立有机的逻辑联系，具有良好的 交互性 的技术。

多媒体的特征：多样性、集成性、交互性、实时性。

2.4.2 多媒体数据压缩编码技术

JPEG，Joint Photographic Experts Group，联合图像专家小组，是一种对静态图像压缩的编码算法。“联合”的含义是：CCITT 和 ISO 联合组成的图像专家小组。

MPEG，Moving Picture Experts Group，运动图像专家小组，是作为一个国际标准来研究制订的，具有很好的兼容性。

其次，比其它算法提供更好的压缩比，最高可达 200:1。更重要的是对数据损失很小。

不存在专利问题，适合大力推广。

数据压缩编码两大类：无损压缩编码法（也称 冗余压缩法、熵编码法），有损压缩编码法（也称 熵压缩法）。

2.4.4 多媒体技术的研究内容

对数据进行有效压缩将是多媒体发展中必须要解决的最关键的技术之一。

数据量大、种类繁多、关系复杂，是多媒体数据的基本特征。

虚拟现实

首先，“逼真”就是要达到 三维视觉、听觉、触觉 等效果；其次，通过人的感官与这个环境进行交互；最后，为用户提供一个逼真的操作环境。

虚拟现实是一种 多技术 多科学 相互渗透集成 的技术。

只能多媒体技术

将具有推理功能的 知识库 与 多媒体数据库 结合起来，形成 智能多媒体数据库。

发展趋势：把 多媒体和通信 功能 集成到 CPU 芯片中。

其一，专用设备、家电及宽带通信设备，可以取代这些设备中的 CPU 及大量 Asic 和其他新品。

其二，与现有的计算机系列兼容，同事具有多媒体和通讯功能。

2.5 系统性能

系统性能 是一个系统提供给用户的 众多性能指标的集合。既包括 硬件性能，也包括软件性能；既包括部件性能指标，也包括综合性能指标。

系统性能包含 性能指标、性能计算、性能设计、性能评估，四个方面内容。

2.5.3 系统性能设计

是一系列重复的受控的性能试验，循环的调整过程为 收集、分析、配置、测试。

阿姆达尔定律 Amdahl：系统中 对某一部件采用某种更快的执行方式所获得的系统性能改变程度，取决于这种方式被利用的频率，或所占总执行时间的比例。

被改进并增强的部分 在总时间中所占的比例，增强比例，永远小于等于 1.

2.5.4 性能评估

对测试结果做出解释，并形成一分文档的技术。

目的是为了性能的优化提供参考。

用得最多、最频繁 的那部分核心程序作为评价计算机性能的标准程序，称为基准测试程序 Benchmark。

第三章 信息化

1975 年，意大利学者 朗高（G • Longo）提出：信息是反映事物的形式、关系相差别的东西，它包含在事物的差异之中，而不在事物本身。

目前，关于信息 比较科学和统一的定义是：信息是对客观事物 变化和特征 的反映，是客观事物之间 互相作用和联系 的表征，是客观事物经过 感知或认知后 的再现。

3.1.2 信息的特征

1、客观性：反映了事物的 运动状态和方式，既事实性。

2、普遍性：信息无所不在。

3、无限性：事物及其变化是 无限多样的。

4、动态性：随着时间变化而变化。

5、依附性：不能完全脱离物质而独立存在。

6、变换性：可以用不同的载体 以不同的方法来负载。

7、传递性：时间上的传递 即存储；空间上的传递 即 转移或扩散。

8、层次性：信息可以分为 战略级、管理级、操作级。

9、系统性：可以形成与现实世界相对应的信息系统。

3.1.3 信息化的定义

信息化 Informationalization，是以信息资源开发利用为核心，以网络技术、通讯技术等高科技技术为依托的 一种新技术扩散的过程。

3.1.4 信息化的内容

1、信息资源的开发利用

2、信息网络的全面覆盖，计算机网络、电信网、电视网等，逐步实现三网合一。

3、信息技术的广泛应用，这是信息化的基础。

4、信息产业的大力发展

5、信息化人才的培养

6、信息化政策和标准规范建设

基于 web 的架构是 松散耦合的，优势在于能够在不同的网络及操作系统中运行；以服务器为中心，客户端瘦小、简单，容易在运行时实现自动升级。

3.3 信息化的典型应用

电子政务的内容

1、政府与政府 G2G

2、政府对企事业 G2B

3、政府对居民 G2C

4、企业对政府 B2G

5、居民对政府 C2G

3.3.3 企业资源规划的结构和功能

物料需求计划 MRP，物料单系统 BOM，制造资源计划 MRPII。

1、ERP 的概念

企业的所有资源包括三大流：物流、资金流、信息流。

ERP 是建立在信息技术基础上，全面地集成了企业的所有资源信息，并为企业提供决策、计划、控制、经营业绩评估 的全方位 和 系统化的管理平台。

ERP 是一种管理理论和管理思想，不仅仅是信息系统。

1.生产预测

市场需求是企业生存的基础，ERP 中首先需要对市场进行较准确的预测，预测主要用于计划。

常用的预测方法有：德尔菲方法、移动平移法、指数平滑法、非线性最小二乘曲线拟合法。

2.销售管理（计划）

销售管理从其计划角度来看，属于最高层计划的范畴，是企业最重要的决策层计划之一。

3.经营计划（生产计划大纲）

4.主生产计划

5.物料需求计划

根据主生产计划 对最终产品的 需求数量和交货期,推导出构成产品的 零部件及材料的 需求数量和需求时期，再导出自制零部件的制作订单下达日期和采购件的采购订单发送日期。

6.能力需求计划 CRP

通过分析比较 MRP 的需求和企业现有生产力，及早发现 能力瓶颈所在。

7.车间作业计划 PAC

将零部件的生产计划以订单的形式下达给适当的车间，属于 ERP 执行层计划。当前主流的车间作业计划模式是 JIT 模式。

8.采购与库存管理

是 ERP 的基本模块，从采购订单产生至货物受到的全过程进行 组织、实施、控制，库存管理 IM 对企业物料的 进、出、存 进行管理。

9.质量与设备管理

全面质量管理 TQM，对企业的全过程进行质量管理，而且明确指出执行质量职能是企业全体人员的责任。

设备管理对 设备寿命周期内的 所有设备 物资运动形态和价值运动形态 进行综合管理。

10.财务管理

以货币的形式 反映和监督 企业的日常经济活动，并对数据进行分类、汇总，为企业管理和决策提供必要的信息支持。

11.ERP 有关扩展应用模块

客户关系管理、分销资源管理、供应链管理、电子商务 等。

3、ERP 的功能

ERP 为企业提供的功能是 多层面的 全方位的。

3.3.4 客户关系管理在企业的应用

1、CRM 的概念

提供的信息要有利于更好地理解客户；

流程管理要为客户提供高效、适当的体验；

提供那些构件强有力关系、提高客户忠诚度的体验。

CRM 的核心思想就是 以客户为中心，

从传统的“以产品为中心”的经营理念解放出来，通过富有意义的交流沟通，理解并影响客户行为，最终实现客户 保留、客户忠诚、客户创利 的目的。

将客户信息 转化为 积极的客户关系 的 反复循环过程。

市场竞争，客户资源逐渐减少，市场主动权让给客户，了解市场和客户 真实需要的基础上 提供令其满意的 产品和服务。

客户能根据自己的需求 量身定做 合适自己需要的产品和服务。

客户信息是 客户关系管理 的基础。

更低成本、更高效率 地 满足客户的需求，与客户建立起 基于学习性关系基础，最大程度提高客户满意度、忠诚度。

销售自动化 SFA

功能：日历和日程安排、联系和客户管理、佣金管理、商业机会、传递渠道管理、销售管理、建议的生产和管理、定价、区域划分、费用报告 等。

产品目录和价格、购买记录、服务记录、存货情况、促销文本资料、信用记录。

SFA 应用往往集成 电子邮件、办公软件 等 其它各种标准应用。

营销自动化 MA

集成客户商业智能信息、产品信息、“营销百科全书”等 信息资源。

CRM 中，客户服务与支持主要是通过 呼叫中心 和 互联网 来实现，在满足客户的个性化要求方面，高速度、准确性、高效率 来完成客户服务人员的各种要求。

当把客户服务与支持功能同销售、营销功能比较好地结合起来时，就能为企业提供很多机会。

客户服务与支持的内容应包括：客户关怀；纠纷、订货、订单跟踪；现场服务；问题及解决方法数据库；维修行为安排调度；服务协议合同；服务请求管理 等。

商业智能是指利用 数据挖掘、知识发现 等 技术 分析和挖掘 结构化的、面向特定领域的存储与数据仓库的信息，帮用户 认清发展趋势、识别数据模式、获取职能决策支持、得出结论。

智能的范围：客户、产品、服务、竞争者 等。

收集和分析市场、销售、服务 和整个企业的各类信息，对客户进行全方位的了解，从而理顺企业资源与客户需求之间的关系。

CRM 尚未有成型的理论出现

对市场的设定、跟踪、分析总结。

呼叫中心支持由合作的硬件厂商参与并提供全套设备，而不仅仅是提供支持呼叫中心的应用软件。

对移动设备的支持。

决策者所掌握的信息完全，能更及时地做出决策。

不管客户由何种渠道与企业联系，与客户的互动都应该是 无缝的、统一的、高效的。

需要任命一名来自企业的 系统管理员，作为内部系统专家。

经特殊调整的系统 必须 伴随技术培训。

由于数据转换过程工作量极大，因此要精确预测该过程的时间表几乎是不可能的。

“培训者”必须接受由软件供应商 进行的培训，称为新系统专家。

对所有用户的 正规培训，用户必须认识到 使用新系统的 即时和明显好处。

对系统的持续支持要求公司配备至少一名全职的内部系统管理员，可保证技术上自给自足的灵活性，CRM 系统的支持是艰巨的工作。

为保证系统带来所希望的益处，在将其推广到所有用户之前一定要加以测试。

间接电子商务，商品是有形货物。

直接电子商务，商品是无形的货物或服务，双方越过地理界限直接进行交易。

3.3.7 供应链管理

供应链是企业赖以生存的商业循环系统，企业供应链可以耗费企业高达 25% 的运营成本。从供应商开始，经由制造商、分销商、零售商，直到最终客户的全要素、全过程的集成化管理模式。

正向 推动式 运作模式是 以生产为中心；逆向 拉动式 运作模式是 以用户为中心；两种不同的运作模式 适用于不同市场环境。

第四章 软件开发

4.1 软件开发方法

4.1.1 软件开发生命周期

传统的软件生命期 是指软件产品 从形成概念（构思）开始，经过定义、开发、使用、维护、废弃，的全过程。

可以把软件生命期划分为 软件定义、软件开发、软件运行与维护，三个阶段。

1、软件定义时期

1.问题定义，目标系统“是什么”，系统的定位以及范围。

2.可行性研究，技术可行性、经济可行性、操作可行性、社会可行性。

3.需求分析，确定软件系统的功能需求、性能需求、运行环境的约束，写出需求规格说明书、软件系统测试大纲、用户手册概要。

充分理解用户的需求，并以书面形式写出规格说明书，这是以后软件设计和验收的依据；用户也许很难 一次性 说清楚系统应该做什么。

系统分析员、软件开发人员、用户，共同完成，逐步细化、一致化、完全化 等。

软件需求规格说明 SRS，内容可以有 系统（或子系统）名称、功能描述、接口、基本数据结构、性能、设计需求、开发标准、验收原则 等。

2、软件开发时期

软件开发时期就是软件的设计与实现，概要设计、详细设计、编码、测试 等。

概要设计是在软件需求规格说明的基础上，建立系统的 总体结构（含子系统的划分） 和 模块间的关系，定义功能模块及各功能模块之间的关系。

详细设计对概要设计 产生的功能模块 逐步细化，包括 算法与结构、数据分布、数据组织、模块间接口信息、用户界面 等，写出详细设计报告。

测试可分成 单元测试、集成测试、确认测试、系统测试 等。通常把编码和测试 称为系统的实现。

3、软件运行和维护

软件维护就是尽可能地延长软件的寿命，没有维护的价值时，宣告退役，软件的生命结束。

4.1.2 软件开发模型

软件生存周期模型 又称 软件开发模型 或 软件过程模型，模型的特点是 简单化，是软件开发实际过程的 抽象与概括。

为软件工程管理提供 里程碑和进度表，为软件开发过程提供 原则和方法。软件过程有各种各样的模型。

1、瀑布型

瀑布型的特点是 因果关系紧密相连，前一个阶段工作的结果是后一个阶段工作的输入，前一个阶段的错漏会隐蔽地带到后一个阶段，每一个阶段工作完成后，都要进行审查和确认，它的出现有利于人员的组织管理，有利于软件开发方法和工具的研究。

2、原型模型

根据用户提出的软件系统的定义，快速地开发一个原型，包含目标系统的关键问题和反映目标系统的大致面貌。

三种途径：

利用模拟软件系统的人机界面和人机交互方式。

真正开发一个原型。

找来一个或几个正在运行的类似软件进行比较。

实际工作中，由于各种原因，大多数原型都废弃不用，仅仅把建立原型的过程 当作帮助定义软件需要的一种手段。

注意：

用户对系统模糊不清，无法准确回答目标系统的需求。

经过对原型若干次修改，应该收敛到目标范围内，否则可能会失败。

对大型软件来说，如果没有现成的，就不应该考虑用原型法。

3、螺旋模型

是生命周期模型与原型模型的一个结合，分成多个阶段，每一个阶段都由 4 部分组成：

1.目标设定，指定对过程和产品的约束，并且制订详细的管理计划。

2.风险分析，制订解决办法。

3.开发和有效性验证，即开发软件产品。

4.评审，确定是否需要进入螺线的下一次回路。

增加一周，软件系统就生成一个新版本，系统应该尽快地收敛到用户允许或可以接受的目标范围内。

该模型支持大型软件开发，适用于面向规格说明、面向过程、面向对象 的软件开发方法，也适用于几种开发方法的组合。

4、基于可重用构件的模型

把软件工程项目所创建的 构件 不断地积累和存储在一个构件库中，系统将依赖构件的健壮性。

5、基于面向对象的模型

构件重用是非常重要的技术之一。一方面进行构件开发，另一方面进行需求开发，快速建立OOA、OOD 原型，由重用构件组装而成，甚至通过组装可重用的子系统而创建更大的系统。

6、基于四代技术的原型

四代语言 完全不用变成方式来构造应用系统，而是利用一些生成器。

与通常的软件工程环境或计算机辅助软件工程不同，只侧重于支持应用软件开发过程中的设计阶段和实现阶段，特别是支持界面以及与界面有关的处理过程。

4.1.3 敏捷方法

1、敏捷方法的特点

敏捷方法是“适应性”而非“预设性”的，重型方法在计划制定完成后拒绝变化，而敏捷方法则欢迎变化。

“面向人的”而非“面向过程的”

传统的软件开发方法的基本思路一般是 只要图纸设计得合理并考虑充分，施工队伍可以完全遵照图纸顺利构造。

但是，一些设计错误只能在编码和测试时才能发现。

传统正规开发方法是 个体不重要，角色才是重要的，尽量减少人的因素对开发过程的影响，但是敏捷方法正好相反。

管理人员已经脱离实际开发活动相当长的时间了，如此设计出来的开发过程是难以为开发人

员所接受的。

只有在第一线的开发人员才能真正掌握和理解开发过程中的技术细节，所以技术方面的决定必须由他们来做出。

敏捷方法特别强调 相关人员之间的信息交流。因为项目失败的原因最终都可以追溯到信息没有及时准确地传递到应该接受它的人。

特别提倡直接的面对面交流，交流成本远远低于文档的交流。

按照高内聚、松散耦合的原则 将项目划分为若干个小组，以增加沟通。

2、敏捷方法的核心思想

1.适应性型，利用变化来发展。

2.以人为本，在无过程控制和过于严格繁琐的过程控制中取得一种平衡，以保证软件的质量。

3.迭代增量式的开发过程，发行版本小型化，根据客户需求的 优先级和开发风险，制订版本发行计划。

3、敏捷方法的含义及其特征

重型方法注重开发文档的完备和充分性；而敏捷方法认为最根本的文档应该是源码。

4、敏捷方法的适用范围

实际上，满足工程设计标准的唯一文档是源代码清单。

敏捷方法比较适合需求变化比较大 或者 开发前期对需求不是很清晰的项目。

敏捷方法对设计者、开发者、客户 之间的有效沟通和及时反馈要求比较高，不易在开发团队比较庞大的项目中实施。

5、敏捷方法的主要内容

四个核心价值观：沟通、简单、反馈、勇气。

简单：只要满足当前功能需求，不做假象设计。

勇气：用于抉择，用于实践，用于重构。

12 条实践规则：简单设计、测试驱动、代码重构、结对编程、继续集成、现场客户、开发版本小型化、系统隐喻、代码集体所有制、规划策略、规范代码、40 小时工作机制。

6、主要敏捷方法简介

极限编程

水晶系列方法

开放式源码，任何人发现 Bug 都可以将补丁发给维护者。

SCRUM

Coad 的功用驱动开发方法：短时迭代阶段 和 可见可用的功能，一个迭代周期一般为两周，编程人员分为 类程序员、首席程序员。

ASD 方法，猜测、合作、学习。

4.1.4 RUP

RUP 把软件开发生命周期划分为多个循环（cycle），每个 cycle 生成产品的一个新版本，每个 cycle 依次由 4 个连续阶段（phase）组成：

初始：定义最终产品视图和业务模型，并确定系统范围。

细化：制定工作计划及资源要求。

构造。

移交。

迭代并不是重复地做相同的事，而是针对不同用例细化和实现，每一个迭代都是一个完整的开发过程。

每个阶段结束前有一个里程碑（milestone）评估该阶段的工作。如果未能通过该里程碑的评估，则决策者应该做出决定，是取消该项目还是继续做该阶段的工作。

RUP 中的核心概念

角色（Role），who 的问题，某个人或一个小组的行为与职责。

活动（Activity），how 的问题，是一个有明确目的的独立工作单元。

制品（Artifact），what 的问题，是活动生成、创建、修改 第一段信息。

工作流（Workflow），when 的问题，每个工作流产生一些有价值的产品，并显示了角色之间的关系。

RUP 的特点

RUP 是用例驱动的、以体系结构为中心的、迭代和增量的软件开发过程。

用例驱动：需求分析、设计、实现、测试，都是用例驱动的。

以体系结构为中心：刻画了系统的整体设计，去掉了细节部分，突出了系统的重要特征。

不依赖于具体语言，是软件设计过程的一个层次。

体系结构层次的设计问题包括：总体组织和全局控制、通讯协议、同步、数据存取、给设计

元素分配特定功能、设计元素的组织、物理分布、系统的伸缩性、性能 等。

一个系统不可能在所有特性上都达到最优，对于一个系统，不同人员所关心的内容也是不一样的，对于不同类型的人员，只需提供这类人员关心的视图即可。

分析和测试人员关心用例图，最终用户关心逻辑视图，程序员关心实现视图，系统工程师关心部署视图。

RUB 强调采用迭代和增量的方法来开发软件，每次迭代中，之考虑系统的一部分需求，每次增加一些新的功能实现。

好处：

早期就可以对关键的、影响大的风险进行处理。

可以提出一个软件体系结构来指导开发。

处理不可避免的需求变更。

可以较早地得到一个可运行的系统，鼓舞开发团队的士气，增强项目成功的信心。

更有效工作的开发过程。

没有一个项目会使用 RUP 中所有的东西，用用 RUP 时要裁剪，裁剪步骤：

1.确定本项目 需要哪些工作流。

2.确定每个工作流要产出哪些制品。

3.确定四个阶段之间（初始阶段、细化阶段、构造阶段、移交阶段）如何演进。

4.确定每个阶段内迭代计划。

5.规划工作流内部结构。

4.1.5 软件系统工具

按软件过程活动将软件工具分为 软件开发工具、软件维护工具、软件管理和软件支持工具。

软件开发工具有：需求分析工具、设计工具、编码与排错工具、测试工具 等。

需求分析工具，生成完整的、清晰的、一致的功能规范。功能规范是软件开发者和用户间的契约，也是软件设计者的和实现者的依据。正确、完整 表达清晰的、无歧义的。

需求分析工具分为 基于自然语言或图形描述的工具，基于形式化需求定义语言的工具。

项目管理工具：项目的 计划、调度、通信、成本估算、资源分配、质量控制等。

4.2 需求管理

需求 最终文档 经过评审批准后，则定义了需求基线 Baseline；构筑了 功能需求 和 非功能需求 的一个 约定 Agreement。约定是需求开发和需求管理之间的桥梁。

需求管理是一个 对系统 需求变更、了解和控制 的过程，初始需求导出的同时 就启动了需求管理规划。

4.2.1 需求管理原则

过程能力成熟度模型 CMM，指导软件过程改进，5 个成熟级别，6 个关键过程域 KPA。

一旦需求 文档化了，开发组和有关团队 需要评审文档。发现问题应与客户或者其他需求源协商解决。软件开发计划是基于 已确认的需求。

绝不要承诺 任何 无法实现的事。

关键处理领域 通过版本控制和变更控制 来管理需求文档。确保与新的需求保持一致。

4.2.2 需求规格说明的版本控制

版本控制是管理需求的一个必要方面，必须统一确定需求文档的每一个版本，当需求发生变更时，及时通知所有涉及人员。

为了尽量减少困惑、冲突、误传，应该仅允许指定的人员来更新需求。

清楚地区分草稿和文档定稿版本。

4.2.4 需求变更

迟到的 需求变更 会对已进行的工作产生非常大的影响。

如果每一个建议的需求变更都采用，该项目将可能永远无法完成。

需求文档应该 精确描述 要交付的产品。

项目负责人 在信息充分的条件下 做出决策。

变更成本计算 应该包括 需求文档的修改、系统修改的设计、实现的成本。

变更控制过程 并不是给变更设置障碍，相反，它是一个渠道和过滤器，确保采纳最合适的变更，使变更产生的负面影响降到最低，变更过程应该做成文档。

绝不能 删除或者修改 变更请求的 原始文档。

变更控制委员会 只要能决定合适的人做正确的事就足够了，在保证权威性的前提下 应尽可能精简人员。

对每个变更 权衡利弊 做出决定。

“利”包括 节省资金 或 额外收入、客户满意度、竞争优势、减少上市时间；

“弊”是指 增加开发费用、推迟交付日期、产品质量下降、减少功能、用户不满意。

变更总是有代价的，即使 拒绝的变更 也因为决策行为 而耗费资源。

接受了重要的需求变更时，为了适应变更情况 要与管理部门和客户重新协商约定。推迟交货时间、增加人手、推迟实现尚未实现的较低优先级的需求，或质量上进行折中。

要是不能获得一些约定的调整，应该把面临的风险写进风险计划中。

4.2.5 需求跟踪

需求、体系结构、其他设计部件、源代码模块、测试、帮助文件、文档 等。

跟踪能力（联系）链（traceability link）是优秀需求规格说明书的一个特征，确保软件需求规格说明包括所有客户需求。

跟踪能力联系链 记录了单个需求之间的 父层、互连、依赖 的关系。

不必拥有所有种类的跟踪能力联系链，要根据具体情况调整。

4.2.6 需求变更的代价和风险

只有在知道变更成本后 才能做出理智的选择，一个表面上很简单的变更 也可能转变成很复杂的局面。

影响分析 确定对现有系统做出是修改或者抛弃的决定，创建新系统以及评估每个任务的工作量，进行 影响分析的能力 依赖于 跟踪能力、数据的质量、完整性。

4.3 开发管理

1、范围

可交付物、架设、约束条件 的基础上准备详细的项目范围说明书，是项目成功的关键。

2、时间

进度安排的准确程度可能比成本估计的准确程度更重要。对于成本估计的偏差，可以靠重新定价或大量的销售来弥补成本的增加，

如果进度计划不能得到实施，则会导致市场机会的丧失或用户不满意，而且会使成本增加。

工作分解结构 Work Breakdown Structure WBS

4.3.2 配置管理 文档管理

1、配置管理

配置项 Configuration Item CI，

属于产品组成部分的工作成果，如 需求文档、设计文档、源代码、测试用例 等。

属于项目管理和机构支撑过程域 产生的文档，如 工作计划、项目质量报告、项目跟踪报告等。

每个配置项的主要属性有 名称、标识符、文件状态、版本、作者、日期 等。

2、文档管理

文档是影响软件可维护性的决定因素，使用过程中必然会经受多次修改，所以文档比程序代码更重要。

用户文档：主要描述系统功能和使用方法。

系统文档：描述系统设计、实现、测试 等各方面内容。

软件文档应该满足下述要求：

1.如何使用

2.怎样安装 和 管理

3.需求 和 设计

4.实现 和 测试

说明用户操作错误时 应该怎样恢复和重新启动。

4.3.3 软件开发的质量与风险

1、软件质量

IOS9000 对 项目质量 的定义：一组固有特性 满足需求的 程度。

质量 与 范围、成本 和 时间，是项目成功的关键因素，通过范围管理 转换隐含需求为项目需求。

质量低 说明产品或服务存在问题，而低等级的产品或服务 不一定存在问题，二者概念不同。

2、软件开发风险

认识不足 或者 没有足够的力量加以控制。

了解、掌握 风险的来源、性质、发生规律，进而施行有效的管理。

或然性、不确定性、涉及到某种选择时，才成为有风险，以上三个是风险定义的必要条件，不是充分条件，具有不确定性的事件不一定是风险。

4.4.1 结构化分析与设计

结构程序设计 较流行的定义为：采用自顶向下 逐步求精 的设计方法 和 单入口单出口的控制构件。

自顶向下逐步求精的方法是：先整体后局部，先抽象后具体，一般具有较清晰的层次。

仅使用单入口单出口的控制构件，具有良好的结构特征。

采用结构程序设计，可能会多占用一些时间和空间资源，这也是那些反对从高级语言中排除 GOTO 语句者的主要依据。实际上，硬件飞速发展，这点耗费，不再是重要的因素。

4.4.2 面向对象的分析设计

面向对象的 分析模型主要由 顶层架构图、用例与用例图、领域概念模型 构成；

设计模型包含：

以包图表示的 软件体系结构图、

以交互图表示的 用例实现图、

完整精确的类图、

针对复杂对象的状态图、

描述流程化处理过程的 活动图 等。

4.5 软件的重用

重复使用 相同或相似 软件元素。

软件元素：需求分析文档、设计过程、设计文档、程序代码、测试用例、领域知识 等，通产这些软件元素称为 软部件。

不断地进行软部件的积累，并将它们组织成软部件库。

横向重用（horizontal reuse）：重用不同应用领域中的软件元素。

标准函数库 是一种 典型的、原始的 横向重用机制。

纵向重用广受瞩目，并称为软件重用技术的真正希望所在，关键点是 域分析，根据应用领域的 特征 以及 相似性 预测软部件的可重用性。

库的组织结构 直接影响软部件的检索效率。

由于软部件大都经过严格的质量认证，并在实际运行环境中得到检验，因此重用软部件有助于改善软件质量。

4.6 逆向工程与重构工程

逆向工程 就是 分析已有的程序，寻找比源代码更高级的抽象表现形式。

相关概念：

重构 Restructuring，在同一抽象级别上转换系统描述形式；

设计恢复 design recovery，

重构工程 re-engineering，也称 修复和改造工程。

1、恢复信息的级别

逆向工程导出的信息，4 个抽象层次

1.实现级

2.结构级

3.功能级

4.领域级

2、恢复信息的方法，4 类：

1.用户指导下搜索与变换

2.变换式方法

3.基于领域知识的

4.铅板恢复法

第五章 软件架构设计

Software Architecture 简称 SA

5.1.2 软件架构设计与生命周期

1、需求分析阶段

需求 和 SA 设计 面临的是不同的对象：一个是问题空间；另一个是解空间。保持二者的可跟踪性和转换。

2、设计阶段

1.传统的设计概念只包括 构件，随着研究的深入，构件间的 互联机制 逐渐独立出来，成为与构件同等级别的实体，称为 连接子。

2.体系结构描述语言（Architecture Description Language ADL）对 连接子 的重视成为区分 ADL 和其他建模语言的重要特征之一。

3.不同的视角 得到多个视图，组织起来以描述整体的 SA 模型；不同侧面的视图反映所关注的系统的特定方面，体现了关注点分离的思想。

3、实现阶段

团队的 结构 应该和体系结构模型有一定的对应关系，提高软件开发 效率和质量。

分析和记录 不同版本构件和连接子之间的演化。

填补高层 SA 模型 和 底层实现 之间的鸿沟，典型的方法如下：

1.引入实现阶段的概念。

2.SA 模型 逐步精化。

3.封装底层称为较大粒度构件。

4、构件组装阶段

可复用构件 组装 可以在较高层次上实现系统，研究内容包括：

1.如何互联。

2.如何检测并消除体系结构失配问题。

中间件跨平台交互。

产品化的中间件更好地保证最终系统的质量，中间件导向的体系结构风格。

失配是指复用过程中，待复用构件对最终系统的体系结构和环境的架设（Assumption）与实际状况下不同而导致的冲突。

5、部署阶段

软件构件的互联性、硬件的拓扑结构、硬件资源占用。

6、后开发阶段

实现中的软件往往具有动态性，一类是软件内部执行所导致的体系结构改变，另一类变化是软件系统外部的请求对软件进行的重配置。

升级或进行其他修改时 不能停机。

SA 重建是指 从已实现的系统中 获取体系结构的过程。

5.2 基于架构的软件开发方法

5.2.1 体系结构的设计方法概述

基于体系结构的软件设计（Architecture-Based Software Design ABSD）方法。

体系结构驱动，指 构成体系结构的 商业、质量、功能 需求的组合驱动。

设计活动的开始 并不意味着 需求抽取和分析活动 就可以终止，而应该 并行，快速开始设计 至关重要。

ABSD 方法有三个基础，功能分解、选择体系结构风格、软件模板的使用。

5.2.2 概念与术语

1、设计元素

ABSD 方法是一个 自顶向下，递归细化 的方法。

2、视角与视图

重要的是从不同的视角（perspective）来检查，考虑体系结构的不同属性。

3、用例和质量场景

在使用用例捕获功能需求时，通过定义特定场景来捕获质量需求，称为质量场景。捕获变更、性能、可靠性、交互性，质量场景必须包括 预期的 和 非预期的。

5.2.4 体系结构需求

可以从需求库中取出，加以利用和修改。

获取需求，体系结构需求一般来自三个方面：系统的质量目标、系统的商业目标、开发人员的商业目标。

5.2.6 体系结构文档化

体系结构规格说明 和 测试体系结构需求的质量设计说明书。

需求模型构件的 精确形式化描述，作为 用户和开发者 之间的一个协约。

从使用者的角度进行编写，必须保证开发者手上的文档是最新的。

5.2.7 体系结构复审

根据架构设计，搭建一个可运行的最小化系统 用于 评估 和 测试 体系架构是否满足需要。是否存在可识别的技术和协作风险。

复审的目的是 标识潜在风险，及早发现 缺陷和错误。

5.2.8 体系结构实现

分割成规定的构件，按规定方式互相交互。

5.3 软件架构风格

体系结构设计 核心目标是 重复的体系结构模式，体系结构级的 软件重用。

5.3.1 软件架构风格概述

一个体系结构 定义 一个词汇表 和 一组约束。词汇表中包含 构件和连接件类型约束指出如何 组合起来。

体系结构风格 反映了 共有的结构和语义特性，并指导如何 组织成一个完整的系统。

5.3.2 经典软件体系结构风格

每个构件都有一组输入和输出，数据输入构件，经过内部处理，然后产生数据输出。这里的构件称为 过滤器。

构件是对象。

分层系统，每一层为上层提供服务，并作为下层的客户。除一些精心挑选的 输出函数外，内部的层接口只对 相邻层可见。由于一层最多只影响两层，为软件重用提供了强大的支持。

仓库风格中，两种不同的构件：中央数据结构、独立构件。

若构件控制共享数据，则仓库是一传统型数据库；若中央数据结构 的当前状态触发进程执行的选择，则仓库是一黑板系统。

C2 体系结构 通过连接件绑定在一起 按照一组规则运作的并行构件网络。构件与构件之间的连接是不允许的。

5.3.3 客户/服务器 风格

宿主机应用程序 既负责与用户的交互（前端），又负责对数据的管理（后端）。

C/S 体系结构 定义了工作站如何与服务器相连，实现部分数据和应用 分布到多个处理机上。

C/S 三个主要组成部分：服务器、客户机、网络。

易于对系统进行扩充和缩小。

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，数据库服务器的开发集中于数据的管理，将大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，模型思想简单。

开发成本高，尤其是软件不断升级，客户端变得越来越臃肿。

信息内容和形式单一，用户获得的只是单纯的字符和数字。

软件移植困难，维护升级困难。

5.3.4 三层 C/S 结构风格。

三层 C/S 体系结构中，可以将 整个应用逻辑 驻留在应用服务器上，只有表示层存在于客户机上，称为“瘦客户机”。表示层、功能层、数据层。

表示层一般要使用图形用户界面 GUI。

功能层之间的数据交互 要 尽可能简洁，一次性传输。

数据层不同层构件 相互独立，层间接口简洁，适合复杂事务处理。

5.3.5 浏览器/服务器 (B/S) 风格

浏览器/服务器 风格 就是 三层应用结构的一种实现方式。浏览器/web 服务器/数据库服务器。

系统安装、修改、维护 全在服务器端解决。仅仅需要一个浏览器就可运行全部模块。

B/S 体系结构还提供了 异种机、异种网、异种应用服务 的 连机、联网 等。

扩展能力差。响应速度慢。交互性不强，不利于在线事务处理 OLTP。

5.4 特定领域软件体系结构

主要目的 在一组相关的应用中 共享 体系结构。

DSSA 的必备特征：

1、一个严格定义的 问题域 和 解域。

2、具有普遍性。

3、对整个领域的 构件 组织模型 其当抽象。

4、具备该领域 固定的、典型的 可重用元素。

5.4.2 DSSA 的基本活动

1、领域分析

主要目标是 获得 领域模型，描述领域中 系统之间的共同需求，定义领域的边界。从而明确分析的对象，识别信息源，确定哪些需求是领域中的系统广泛共享的，从而建立领域模型。

2、领域设计

目标是获得 DSSA，DSSA 描述在领域模型中表示的需求 的解决方案。不是单个系统的表示，而是能够适应领域中 多个系统的需求的 一个高层次设计。

3、领域实现

主要目标是 依据 领域模型 和 DSSA 开发和组织 可重用信息。领域模型 和 DSSA 定义了这些可重用信息的 重用时机。

以上过程是 反复的、逐渐求精 的过程。

5.4.3 参与 DSSA 的人员

4 种角色：领域专家、领域分析师、领域设计人员、领域实现人员。

1、领域专家 可能包括 有经验的用户、从事该领域中系统的需求分析、设计、实现 以及项目管理的有经验的软件工程师等。

主要任务 提供 需求规约和实现的知识，组织规范的、一致的领域字典，选择样本系统，复审领域模型、DSSA。

应该 熟悉该领域 软件设计和实现、硬件限制、未来的用户需求、技术走向 等。

2、领域分析人员 应由 系统分析员来担任。

知识获取 组织到领域模型中，根据 现有系统、标准规范 等 验证模型的 准确性 和 一致性。

应熟悉软件重用和领域分析方法，具有一定的该领域经验，较高的 抽象、关联、类比 能力，较高的 交互合作能力。

3、领域设计人员 控制整个软件设计过程，根据领域模型和现有系统 开发出 DSSA，对 DSSA 的准确性和一致性进行验证，建立领域模型和 DSSA 之间的联系。

应熟悉软件重用和领域设计方法，熟悉软件设计方法，有一定的该领域经验。

4、领域实现人员 根据领域模型和 DSSA，从头开发可重用构件，或 利用再工程技术 从现有系统中提取可重用构件。

5.4.4 DSSA 的建立过程

一般情况下，需要用 开发者习惯使用的工具和方法 建立 DSSA 模型。

DSSA 建立过程分为 5 个阶段，过程是 并发的、递归的、反复的，可能每个阶段经历几遍，每次增加更多的细节。

1、定义领域范围，一系列用户的需求。

2、定义领域特定的元素，编译领域字典、领驭属于的同义词词典。

3、定义特定的设计和实现需求约束，不仅要识别出约束，并且要 记录 约束对设计和实现 造成的后果，还要记录对处理这些问题时所产生的所有问题的讨论。

4、定义领域模型和体系结构，产生一般的体系结构，并说明构成它们的模块或构件的语法、语义。

5、搜集可重用的产品单元，为 DSSA 增加构件。

5.5 系统架构的评估

评估 可以只针对一个体系结构，也可以针对一对一组体系结构。关注的是 质量属性。

1、性能，是指系统的响应能力，多长时间 对某个事件做出响应，或者 某段时间内系统所能处理的事件的个数。

2、可靠性，是最重要的软件特性，平均失效等待时间 MTTF，平均失效间隔时间 MTBF

1.容错，内部修复。

2.健壮性，不受错误使用和错误输入的影响。

3、可用性，正常运行的时间比例。经常用两次故障之间的时间长度或恢复正常的速度来表示。

4、安全性，阻止非授权用户。分为 机密性、完整性、不可否认性、可控性 等特性。

5、可修改性，通过考察 变更的代价 衡量可修改性。

1.可维护性，主要体现在问题修复上，做局部性的修改并能使对其他否见的负面影响最小化。

2.可扩展性，新特性来扩展软件系统，改进版本来替换构件并删除不需要的特性构件，需要松散耦合的构件。

3.结构重组，需要精心设计构件之间的关系。

4.可移植性。

6、功能性，完成所期望的工作 的能力。

7、可变性。

8、互操作性，精心设计的软件入口。

5.5.2 评估中重要概念

敏感点 权衡点，是关键的体系结构决策。

敏感点是 构件（和/或 构建之间的关系）的特性。研究敏感点可使人员明确在实现质量目标时 应注意什么。

权衡点 是多个质量属性的 敏感点。

风险承担着 或称为 收益相关人。

场景，首先要精确地得出具体的质量目标，为得出这些目标采用的机制叫做场景。从风险承担者的角度与系统的交互的简短描述。

刺激、环境、响应，三个方面描述场景。

5.5.3 主要评估方法

1、SAAM 非功能质量属性的体系结构分析方法，是最早形式成文档并得到广泛使用的分析方法。最初它用于比较不同的软件体系结构，以分析 SA 的可修改性。

1.特定目标，目标是对描述应用程序属性的文档，验证假设和原则，有利于评估固有的风险。

2.评估技术，使用场景技术，描述了各种系统 必须支持的活动 和 将要发生的变化。

3.质量属性，可修改性 是 SAAM 分析的主要 质量属性。

4.风险承担者，SAAM 协调不同参与者所感兴趣的方面，作为后续决策的基础，提供了对系统结构的 公共理解。

5.体系结构描述，描述形式 应该被所有参与者理解。功能、结构、分配，三个主要方面。

6.方法活动，SAAM 的主要输入问题是 描述、需求声明、体系结构描述。

SAAM 分析评估 体系结构过程包括 5 个 步骤：场景开发、体系结构描述、单个场景评估、场景交互、总体评估。

通过各类 风险承担者协商讨论，开发一些 任务场景，体现系统所支持的 各种活动。

通过对场景交互的分析，得出系统中所有场景对系统中构件所产生影响的列表。总体的 权衡 和 评价。

2、ATAM

体系结构权衡分析方法，主要针对 性能、实用性、安全性、可修改性。

确定多个质量属性之间 这种 的必要性。

体系结构空间 受到 历史遗留系统、互操作性 和 以前失败的项目 约束。

逻辑视图被分为 功能结构 和 代码结构。这些结构加上他们之间适当的映射可以完整地描述一个体系结构。

用一组 消息顺序图 显示运行时的 交互 和 场景。

从不同的体系结构角度，有三种不同场景，用例、增长场景、探测场景。

ATAM 使用定性的 启发式分析方法 QAH，构造 精确分析模型时 要进行分析。

4 个主要的活动领域（或阶段），场景和需求收集、结构视图和场景实现、属性模型构造和

分析、分析、折中。

属性分析是互相依赖的。获得属性交互的方法有两种，敏感度分析来发现折中点、通过检查假设。

迭代的改进。除了通常从场景派生而来的需求，还有很多对 行为模式和执行环境的 假设。由于属性之间存在折中，每一个架设都要被 检查、验证、提问，完成所有操作后，把分析的 结果和需求 进行对比。

领驭知识库通过基于属性的 体系结构风格 ABAS 维护，变得更为惯例化、更可预测，得到一个标准问题集合。

第六章 UML 相关

UML 建模与架构文档化

方法种类的膨胀，极大地妨碍了用户的使用和交流。

UML 通过统一的表示法，使不同知识背景的 领域专家、系统分析、开发人员、用户 可以方便地交流。

6.1.2 UML 体系结构演变

UML 是用 元模型 描述的，元模型是 4 层元模型体系结构模式中的一层，其他层次分别是元-元模型、模型层、用户对象曾。其中元模型层 由 元-元模型层 导出。

元模型的体系结构模式 可以用来定义 复杂模型 所要求的 精确定义,这种复杂模型通常需要被 可靠地 保存、共享、操作 以及在工具之间进行交换。它的特点如下：

1、在每一层都递归地定义语义结构。

2、可用来定义 重量级和轻量级 扩展机制。

3、在体系结构上 将其他体系结构的标准统一起来。

UML 元模型又被分解为三个逻辑子包：基础包、行为元素包、模型管理包。

6.2 UML 基础

UML 通过 图形化的表示机制 从多个侧面 对系统的分析和设计模型进行刻画。

10 种视图，四类：

1、用例图

2、静态图，包括 类图、对象图、包图。

类图的边表示类之间的联系，包括 继承、关联、依赖、聚合 等。

对象图描述在某种状态下或某一时间段，系统中 活跃的对象及其关系。

包由 子包、类 组成。

3、行为图，包括 交互图、状态图、活动图，他们从不同的侧面刻画系统的动态行为。

交互图分为 顺序图、合作图。顺序图强调 对象之间 消息发送的时序。合作图更强调对象间 的动态协作关系。

状态图 描述 对象的动态行为。

活动图 描述 操作序列，这些操作序列 可以并发、同步，包含控制流、信息流。

4、实现图，包括 构件图、部署图。描述组成和分布情况。

部署图 节点表示实际的计算机和设备，边表示节点之间的物理连接，也可以显示连接的类型及节点之间的依赖性。

6.2.2 用例和用例图

用例图 也翻译为 用况、用按 等，在 UML 中，用例用一个椭圆表示，往往用 动宾结构 或主谓结构 命名。

可选的 动作序列 和 会出现异常的动作序列。

用例是代表系统中 各种相关人员之间 就系统的行为所达成的契约。

需求阶段 用例是 分析人员与客户沟通的工具 项目规模估算的依据；

设计阶段 用例是 系统功能设计的主要输入；

实现阶段 用例是 检测类型为正确性的文档。

本质上，用力分析 是一种功能分解 的技术。

1、参与者 角色，参与者实际上 并不是系统的一部分。

2、用例间的关系，泛化、包含、扩展 等。

包含是比较特殊的依赖关系。

扩展，基本用例必须声明 若干“扩展点”，而这些扩展用例只能在这些扩展点上增加新的行为和含义。

3、用例图

建模人员可以在途中给某些图符加上填充色，在语义上，使用填充颜色和不使用填充颜色的模型是 一样的。

6.2.3 交互图

描述对象之间 对象与参与者之间 动态协作关系 协作过程中行为次序。

通常描述用例的行为，显示该用例中所涉及的对象 对象之间的消息传递。

顺序图、协作图 之间可以互相转化，一个用例需要多个顺序图或协作图。

交互图可以帮助分析人员 对照检查 每个用例中所描述的 用户需求，提醒分析人员去补充遗漏的类或方法。

水平方向为对象维，一般 主要参与者放在最左边，次要参与者放在最右边。

垂直方向为时间维。

6.2.4 类图和对象图

一般而言，类的名字是 名词。

类之间的关系 有 关联、聚集、组合、泛化、依赖 等。

1、关联，链 是关联的实例，关联表示 类与类之间的关系，链表示 对象与对象之间的关系。

关联用 实线表示，角色还具有多重性。

关联类 描述关联的 属性、操作、以及其他信息。

关联类 通过一条虚线与关联连接。

自返关联 又称 递归关联，同一个类的两个对象间的关系。两个关联端，每个关联端的角色不同。

2、聚集和组合

聚集 是一种特殊形式的 关联，类之间整体与部分的关系。

组合 整体与部分具有同样的生存期，是一种特殊形式的聚集。

3、泛化关系，一般和特殊元素之间的关系，就是平常所说的继承关系。

6.2.5 状态图和活动图

1、状态图

描述 对象 生存期间的 动态行为，所经历的状态序列，引起状态转移的 事件、动作。

是 UML 动态行为建模的 5 个图之一，用 状态机 对一个对象的生命周期建模，状态图 用于显示状态机，重点在于 状态之间的控制流。

除了 初态和终态，还有 Idle 和 Running 两个状态，keyPress、finished、shutDown 是事件。

2、活动图

是 UML 动态行为建模的 5 个图之一，描述系统的 工作流程 和 并发行为。状态图的特殊形式，一个活动结束后将立即进入下一个活动。

基本概念：活动、泳道、分支、分叉、汇合、对象流。

1.活动，注意区分 动作状态 和 活动状态，

动作状态是原子的，没有内部转移，没有内部活动，所占用的时间可以忽略，目的是执行进入动作，然后转向另一个状态。

活动状态是可分解的，工作完成需要一定的时间。

2.泳道，是活动图中区域划分，每个泳道代表一个责任区，泳道 和 类 并不是一一对应的关系。

3.分支，同一个触发事件，可以根据不同的警戒条件 转向不同的活动，每个可能的转移 是一个分支。

4.分叉和汇合，如果要表示 系统或对象 中的 并发行为，使用 分叉 fork 和 汇合 join，汇合正好与分叉相反。

5.对象流，活动图中可以出现对象，对象可用作为活动的输入输出。活动图中的对象流表示活动和对象之间的关系。

6.2.6 构件图

构件 是系统中 遵从一组接口 且提供其实现的 物理的、可替换 的部分。

构件图 显示一组构件 以及它们 之间的相互关系，包括 编译、连接、执行时 构建之间的依赖关系。

构件就是一个实际文件，以下几种类型：

1、部署构建

2、工作产品构件

3、执行构件

构件图可以对以下几个方面建模：

1、对源代码文件之间的相互关系建模。

2、对可执行文件之间的相互关系建模。

6.2.7 部署图

部署图 也称 配置图、实施图，显示系统中计算节点的 拓扑结构、通信路径、节点上运行的软构件等。

一个系统模型只有一个部署图，常用语帮助理解分布式系统。

部署图 由 体系结构设计师、网络工程师、系统工程师 等 描述。

6.3 基于 UML 的软件开发过程

6.3.1 开发过程概述

UML 是独立于软件开发过程的，能够在几乎任何一种软件开发过程中使用。迭代的渐进式软件开发过程包含四个阶段：初启、细化、构件、部署。

1、初启

项目的发起人 确定项目的 主要目标 和 范围，初步的可行性分析 和 经济效益分析。

2、细化

细化阶段的开始 标志着 项目的正式确立。

1.初步的需求分析，比较重要、比较有风险的用例。

2.初步的高层设计，用例、用例图、类、类图 将 依据 包 的划分方法 分属于 不同包。

3.部分的详细设计，根据软件元素 的重要性和风险程度 确立优先细化原则，不能将风险的识别和解决延迟到细化阶段后。

4.部分的原型构造。

3、构建

构造阶段，每次迭代中实现一部分用例，用户可以及早参与对已实现用例的实际评价。

原则：

1.用户认为业务价值较大的用例 应 优先安排。

2.开发人员评估后 认为 开发风险较高的用例 优先 安排。

迭代计划中，要确定迭代次数、每次迭代所需时间 以及 每次迭代中应完成的用例。

6.3.2 基于 UML 的需求分析

1、生成用例

如果多个用户扮演同一角色，这些用户将由单一执行者表示。如果一个用户扮演多种角色，则需要多个执行者来表示同一用户。

用例主要来源于分析人员对 场景的 分类和抽象，即将相似的场景进行归类，使一个用例可以通过实例化和参数调节而涵盖多个场景。

2、用活动图表示用例

3、生成用例图

执行者与用例之间的关系有两种：触发执行、信息交换。

执行者指向用例 表示 触发执行 和/或 信息交换，用例指向执行者 表示用例将生成的信息传递给执行者。

4、建立顶层架构

顶层架构便于开发人员 聚焦于系统的不同部分。

模型——视图——控制器（Model、View、Controller，MVC）模式。

模型 维护并保存数据，视图 呈现数据，控制器将动作映射为处理功能并实际调用。

MVC 模式特别适合于分布式应用软件，尤其是 web 应用系统。

分层模式 降低软件系统的 耦合度。

确立顶层架构的过程中需综合考虑以下因素：

包的数量，架构过早地陷入细节，返工的可能性很大，也不合理地限制了后续分析和设计活动的自由空间。

包之间的耦合度。

将不稳引起的软件元素分类聚集于少数几个包中，以提高软件系统的可维护性。

可选功能 和 必须实现的功能 置于 不同的包。

根据开发人员 专长 划分，使每个包都能分配给最合适的开发人员，有利于并行开发。

6.3.3 面向对象的设计方法

1、设计用例实现方案

1.提取边界类，实现类和控制类。

边界类用于描述 系统与外部环境之间的交互。

a.界面控制。

b.外部接口。

c.环境隔离。使目标软件系统的 其余部分 尽可能地 独立于环境软件。

边界类，<<boundary>>。

实体类“内向收敛”特征，仅提供 读/写 信息的必要操作 作接口，并不涉及业务逻辑处理，<<entity>>。

控制类，<<control>>。

边界类的作用范围可以超越单个用例。

2.构造交互图

交互图作为用力的精确实现方案。

事件流中的事件 直接对应交互图中的消息，事件间的先后关系体现为 交互图中的时序，对消息的响应 则构成消息接收者的职责，这种职责被确立为 类的方法。

不应该出现 穿越控制类 生命线 的消息。

为 易于理解，应该用分离的 UML 交互图 分别表示 事件流和每个备选事件流。

原则上，每个类都应该有一个操作来响应交互图中指向其对象的那条消息。

2、设计技术支撑方案

当用户需求发生变化时，技术支撑方案应具有良好的稳定性。

技术支撑方案应该位于层次结构中的较低层次。

一方面取决于 需求，另一方面取决于 对软件技术手段把我和选取。

3、设计用户界面

1.熟悉用户 并对 用户分类，以便尽量照顾到所有用户的合理要求，并优先满足某些特权用户。

2.按用户类别 分析用户的 工作流与习惯，从每类中选取一个用户代表，建立调查表，判断用户对操作界面的需求和喜好。

3.首先应考虑命令的顺序，一般常用命令居先，与用户工作习惯保持一致；其次，根据外部服务之间的聚合关系组织相应的命令；然后充分考虑人类记忆的局限性，最好组织为一颗两层多叉树；提供操作的快捷方式。

5.利用快速原型演示，改进界面设计。并评判系统是否 齐全、方便、好用。

4、精化设计模型

对模型进行改进的活动可以分为 精化 和 合并 两种。一般先从精化开始。设计优秀的粗粒度组件应该只是完成一项功能，这一点是它与子系统的主要区分。

粗粒度组件的范围过于广泛，难以发挥重用价值，粗粒度组件拥有持久化的行为，拥有业务逻辑，需要表示层的支持。

将需求分成几个功能组，基本上就可以得到相应的粗粒度组件了。

过小的范围，将会造成粗粒度组件不容易使用，用户需要理解不同的粗粒度组件之间的复杂关系。

如果可能，在粗粒度组件之间定义单项关联可以有效的减少组件之间的耦合。

尽可能简化组件之间的关系。

我们需要从软件的目标领域中 识别出关键性的实体，或者说领域中的名词。然后决定它们应该归属于那些粗粒度组件。

两个组件之间存在重复的要素，可以从中抽取共性的部分，形成新的组件。

6.4 系统架构文档化

6.4.1 模型概述

以精心选择的形式 将若干结构元素进行装配。

软件架构 = { 元素，形式，关系/约束 }

逻辑视图（logical view）对象模型。

过程视图（process view）并发和同步特征。

物理视图（physical view）分布式。

开发视图（development view）静态组织结构。

场景。

Rational 4.1 视图模型。

每个视图上均独立地应用 Perry&Wolf 软件架构公式。

对每种视图选用特定的 架构风格（architectural style）。

6.4.2 逻辑结构

逻辑架构主要支持功能性需求，系统分解为一系列的关键抽象，（大多数）来自于问题域，表现为对象或对象类的形式。

抽象、封装、继承。

对于数据驱动程度高的应用程序，可以使用其他形式的逻辑视图，如 E-R 图 代替面向对象的方法。

1、逻辑视图的风格

采用面向对象的风格，试图在整个系统中 保持 单一的、一致的 对象模型。

6.4.3 进程架构

进程架构考虑一些非功能性的需求，并发性、分布性、系统完整性、容错性，以及逻辑视图的主要抽象如何与进程结构相配合在一起。

进程是 构成可执行单元任务的分组。

区分主要次要任务：主要任务是 可以唯一处理的架构元素；次要任务是 由于实施原因而引入的局部附加任务。

6.4.4 开发架构

开发架构关注软件开发环境下实际模块的组织。

开发架构用模块和子系统图来表达，显示了“输出”和“输入”关系。

考虑因素：开发难度、软件管理、重用性、通用性、由工具集、语言 所带来的限制。

开发视图 是建立产品线的 基础。

推荐使用分层（layered）的风格，每层具有良好定义的职责。某层子系统依赖同一层或低一层的子系统，最大程度地减少了具有复杂模块依赖关系的 网络的开发量。

6.4.5 物理架构

物理架构主要关注系统非功能性的需求，可用性、可靠性（容错性），性能（吞吐量）、可伸缩性。

软件至节点的映射需要高度的灵活性 及 对源代码产生最小的影响。

6.4.6 场景

4 种视图的元素通过数量比较少的一组重要场景（更常见的是用例）进行无缝协同工作，我们为场景描述相应的脚本（对象之间和过程之间的交互序列）。

在某种意义上 场景是最重要的 需求抽象。

4+1 的 +1 起到了两个作用：

作为一项驱动因素 来发现架构设计过程中的 架构元素。

作为架构原型测试的出发点。

场景表示法与组件逻辑视图非常相似，但它使用过程视图的连接符来表示对象之间的交互。

6.4.7 迭代过程

在进行文档化时，提倡一种更具有迭代性质的方法——架构先被原型化、测试、估量、分析，

然后在一系列的迭代过程中被细化。

除了减少 风险之外，还有其他优点：团队合作、培训、加深对架构的理解、深入程序和工具 等。使 需求被细化、成熟化。

系统大多数关键的功能以场景的形式被捕获，关键意味着：最重要的功能、系统存在的理由、使用频率最高的功能、必须减轻的一些重要技术风险。

第七章 设计模式

7.1 设计模式概述

重复遇到的典型问题，描述这些共同问题 和 解决这些问题的方案 就形成了所谓的 模式。

7.1.1 设计模式的历史

模式分为几个部分：

特定的情景（Context），指模式在 何种情况下发生作用；

动机（System of Force），指问题或预期的目标；

解决方案（Solution），平衡各动机 或 解决所阐述问题的 构造或配置。

每个模式描述了一个在某种特定情境下不断重复发生的问题，以及解决该问题解决方案的核心所在。

7.1.2 为什么要使用设计模式

面向对象设计时需要考虑 封装性、力度大小、依赖关系、灵活性、可重用性 等。

1、简化并加快快设计

无需从底层做起，重用成功的设计，节约开发时间，提高软件质量。

2、方便开发人员之间的通信

可以更准确地 描述问题 及 问题的解决方案，使解决方案具有一致性。

3、降低风险

4、有助于转到面向对象技术

开发人员对新技术往往会有抵触或排斥心理，对新技术可能带来的效果持怀疑态度。

成熟的设计模式具有以下特性：

1.巧妙。

2.通用，不依赖于 系统、语言、领域。

3.不仅仅停留在理论上。

4.简单。

5.可重用。

6.面向对象。

7.1.3 设计模式的组成元素

1、模式名，简洁地描述了 模式的本质，可以帮助我们思考。

2、问题或意图，解释了设计问题和问题存在的前因后果，可能描述了特定的设计问题。

3、情景，告诉我们该模式的适用性。

4、动机，描述相关的动机和约束，通常需要对各期望的目标进行有限排序，动机阐明了问题的复杂性，定义了在相互冲突时所采取的各种权衡手段。

5、解决方案，因为模式就像一个模板，所以解决方案并不描述一个特定而具体的设计或实现，而是提供设计问题的 抽象描述 和怎样用一个 具有一般意义的 元素组合。

6、示例，帮助读者理解模式的具体使用方法。

7、结果情景，阐述了模式后续状态和副作用。

8、基本原理，解释该模式 如何、为何 能解决当前问题。

9、相关模式，包括 静态的 和 动态的，迁到模式、后续模式、替代模式。

10、已知应用，通常好的模式前面都有一个摘要，提供简短的总结和概述，为模式描绘出一个清晰的图画，提供有关该模式能够解决问题的快速信息。

模式应该说明它的目标读者，以及对读者有哪些知识要求。

7.1.4 设计模式的分类

软件模式 主要可分为 设计模式、分析模式、组织和过程模式 等。

设计模式主要用于 得到简洁灵活的 系统设计。

按设计模式的目的划分，创建型、结构型、行为型；

按设计模式范围划分，类设计模式、对象设计模式。

1、创建型模式，对对象实例化过程的 抽象，采用抽象类所定义的接口，封装了对象如何创建、组合 等信息。

2、结构型模式，如何组合已有的类和对象 以及获得更大的结构。

3、行为型模式，不仅描述对象或类的模式，还描述它们之间的通信模式，特别是描述一组 对等的对象怎样互相协作 完成其中任一对象 无法单独完成的任务。

7.2 设计模式实例

7.2.1 创建性模式

通过该了的子类来创建对象的。但是，这可能会 限制在系统内创建对象的 类型或数目。

1、Abstract Factory 模式

在不指定具体类的情况下，为创建一些列 相关 或 相互依赖的对象提供了接口。

提供了一个可以 确定合适的具体类 的抽象类。

优点：

可以与具体类分开。

更容易在产品系列中转换。

提高了产品间的一致性。

以下情况应该使用 Abstract Factory 模式：

系统独立于产品的 创建、组成、表示。

系统配置成 具有多个产品的 系列。

相关产品对象系列 是共同使用的，而且必须确保这一点。

你希望提供产品的类库，只开放其接口。

2、Builder 模式

将复杂对象的 构件与表示 相分离，相同的构造过程可以创建不同的对象，通过只指定对象的 类型和内容。

一次就可以创建所有的复杂对象，而其他模式一次就只能创建一个对象。

优点：

可以对产品内部表示进行改变。

将构造代码与表示代码相分离。

以下情况应该使用 Builder 模式：

算法独立于 组成对象。

构造过程必须允许已构件对象有不同表示。

3、Factory Method 模式

实例化工作交给其子类，可以在不修改代码的情况下引入新类，因为新类只实现了接口。

优点：

代码只处理接口，因此可以使用任何实现了接口的类。

在类中创建对象比直接在客户端创建要更加灵活。

以下情况中，应该使用 Factory Method 模式：

类不能预料它必须创建的对象的类。

类希望其子类指定要创建的对象。

类将责任转给某个帮助子类，而用户希望定位那个被授权的帮助子类。

4、Prototype 模式

只要将对象类定义成能够复制自身就可以实现。

优点如下：

可以在运行时 添加或删除产品。

通过改变值 指定新对象。

通过改变结构 制定新对象。

减少子类的生成和使用。

可以用类 动态地配置 应用程序。

以下情况中，应该使用 Prototype 模式：

运行时，指定需要实例化的类，例如动态载入。

避免构建与产品的类层次结构相似的 工厂类层次结构。

5、Singleton 模式

确保 一个类只有一个实例，并且提供全局访问入口，确保使用这个 实例 所有的对象 使用相同的实例。

优点：

对单个实例的受控访问。

命名空间的减少。

允许改进操作和表示。

允许改变数目的实例。

比类操作更灵活。

7.2.2 结构性模式

机构性模式 控制 较大部分之间的 关系。

它将以不同的方式 影响应用程序。

允许在补充写代码或自定义代码的情况下 创建系统。

具有增强的 重复使用性和应用性能。

1、Adapter 模式

可以充当两个类之间的媒介，可以转换一个类的接口，被另外一个类使用，使得具有不兼容接口的类能够系统使用。

优点：

允许 多个 不兼容的对象 进行交互和通信。

提高已有功能的 重复使用性。

以下情况，应该使用 Adapter 模式：

要使用已有类，而该类接口与所需的接口并不匹配。

要创建可重用的类，该类可以与 不相关 或 未知类 进行协作。

要在一个 不同于已知对象接口的接口环境中 使用对象。

必须要进行多个源之间的接口转换的时候。

2、Bridge 模式

将一个复杂的组件 分成两个独立的 但又相关的 继承层次结构：功能性抽象和内部实现。

优点：

接口与实现相分离。

提高了可扩展性。

对客户端隐藏了实现的细节。

以下情况中，应该使用 Bridge 模式：

避免在抽象及其实现之间 存在永久的绑定。

抽象及其实现 可以使用子类进行扩展。

抽象的实现被改动 不用重新编译代码。

3、Composite 模式

创建树形层次结构来改变复杂性。

优点：

定义了由 主要对象 和 符合对象 组成的类层次结构。

添加新的组件类型更加简单。

结构的灵活性 和 可管理性的接口。

以下情况中，应该使用 Composite 模式：

想要表示对象的整个 或者部分的层次结构。

想要客户端能够忽略符合对象和单个对象之间的差异。

结构可以具有任何级别的复杂性，而且是动态的。

4、Decorator 模式

不修改对象外观和功能的情况下 添加或删除对象功能。

优点：

比静态继承具有更大的灵活性。

避免了特征装载的类 处于层次结构的 过高级别。

简化了编码。

改进了对象的扩展性。

在以下情况中，应该使用 Decorator 模式：

在单个对象中 动态并且透明地 添加责任，不会影响其他对象。

以后可能要修改的对象中添加责任。

无法通过静态子类化实现扩展时。

5、Facade 模式

为子系统中的一组接口 提供了一个统一的接口。更容易使用子系统的高级接口。

优点：

在不减少系统所提供的选项的情况下，为复杂系统提供了简单接口。

屏蔽了子系统组件。

提高若耦合度。

将客户端请求转换后 发送给能够处理这些请求的 子系统。

以下情况中，应使用 Facade 模式：

为复杂的子系统提供简单的接口。

客户端和抽象的实现类中 存在许多依赖关系。

想要对子系统进行分层。

6、Flyweight 模式

通过共享对象 减少对象数目。

通过共享一个接口来避免使用多个具有相同信息的实例 所带来的开销。

优点：

减少了要处理的对象数目。

如果对象能够持续，可以减少内存和存储设备。

以下情况中，应该使用 Flyweight 模式：

应用程序使用大量的对象。

由于对象数目巨大，导致很高的存储开销。

不依赖于对象的身份。

7.2.3 行为性模式

行为性模式 影响 系统的 状态、行为流。

简化、优化 并且 提高应用程序的 可维护性。

1、Chain of Responsibility 模式

在系统中建立一个链，在首先接收到它的级别处 被处理，或者定位到可以处理它的对象。

优点：

降低了耦合度。

增加面向对象制定责任的 灵活性。

类的集合可以作为一个整体。

以下情况中，应该使用 Chain of Responsibility 模式：

多个对象可以处理一个请求，而其处理器却是未知的。

在不指定确切的 请求接受对象的情况下，向几个对象中的 一个 发送请求。

动态地指定能够处理请求的对象集。

2、Command 模式

在对象中封装了请求。

优点：

将调用操作的对象 与 知道如何完成该操作的对象 相分离。

更容易添加新指令，因为不用修改已有类。

以下情况中，应该使用 Command 模式：

要通过执行的动作 来 参数化对象。

在不同的时间 指定、排序、执行 请求。

必须支持 Undo、日志记录 或 事务。

3、Interpreter 模式

解释定义其语法表示的语言，提供了语句解释器。

优点：

容易修改并扩展语法。

更容易实现语法。

以下情况中，应该使用 Interpreter 模式：

语言的语法比较简单。

效率并不是最主要的问题。

4、Iterator 模式

为集合中的有序访问提供了一致的方法，而该集合是独立于基础集合。

优点：

支持集合的不同遍历。

简化了集合的接口。

以下情况中，应该使用 Iterator 模式：

不开放集合对象内部表示的前提下，访问集合对象内容。

支持集合对象的多重遍历。

为遍历集合中的不同结构 提供了统一的接口。

5、Mediator 模式

通过引入一个能够管理对象间消息分布的对象，简化了系统中对象间的通信。提高了对象间的松耦合度，还可以独立地 改变 其间的交互。

优点：

去除对象间的影响。

简化了对象间协议。

集中化了控制。

由于不再需要直接互传消息，单个组件变得更加简单，而且容易处理。

由于不再需要 包含逻辑 来处理组件间的通信，组件变得更加通用。

以下情况中，应该使用 Mediator 模式：

对象集合需要以 一个定义规范但复杂的方式 进行通信。

6、Memento 模式

保持对象状态的“快照”（snapshot），对象可以在不向外界公开其内容的情况下 返回到它的最初状态。

优点：

保持封装的完整性。

简化了返回到初始状态所需的操作。

以下情况中，应该使用 Memento 模式：

必须保存对象状态的快照，恢复状态。

7、Observer 模式

定义了对象间 一到多 的依赖关系，当对象改变状态时，将自动通知 并 更新它所有的依赖对象。

优点：

抽象了主题与 Observer 之间的耦合关系。

支持广播方式通信。

以下情况中，应该使用 Observer 模式：

对一个对象的修改 涉及对其他对象的修改，而且不知道有多少对象 需要进行相应修改。

对象应该能够 在不同假设对象标识的前提下 通知其它对象。

8、State 模式

对象在内部状态变化时，变更其行为，并且修改其类。

优点：

针对不同状态来划分行为，使状态转换显式进行。

9、Strategy 模式

定义了一组能够用来表示 可能行为集合的类。这些行为 可以在应用程序中使用，来修改应用程序功能。

优点：

另一种子类化方法。

在类自身中定义了 每一个行为，减少了条件语句。

更容易扩展模型。

以下情况中，应该使用 Strategy 模式：

许多相关类 只是在行为方面有所区别。

需要算法的不同变体。

算法使用客户端未知的数据。

10、Template Method 模式

不重写方法的前提下 允许 子类重载部分方法 的方法。

将一些步骤由子类实现。

优点：代码重用的基础技术。

以下情况中，应该使用 Template Method 模式：

一次实现算法的不变部分，子类实现算法的可变行为。

11、Visitor 模式

不改变操作元素的类 的前提下 定义一个新操作。

优点：

容易添加新操作。

集中相关 排除不相关 操作。

以下情况中，应该使用 Visitor 模式：

包含许多具有不同接口的对象类，并且想要对这些 依赖具体类的对象进行操作。

定义对象结构的类很少被修改，但想要在此结构上定义新的操作。

第八章 XML

8.1 XML 概述

可扩展标记语言（xml）是标准通用标记语言（SGML）的一个子集；可以用 XML 来开发一种标记语言，它的元素和属性多是为专门行业和产业而定义的。

支持统一字符编码 UCS，使得 XML 成为了国际标准，XML 和 HTML 都支持 样式表（style sheet）。

8.1.2 标签语法

XML 元素的结构与 HTML 基本相同，使用尖括号来界定标签，但二者相同点也就仅此而已。与 HTML 不同，几乎所有的 XML 标签 都是大小写敏感的，主要是满足 XML 国际化的设计目标和简化处理过程的需要。

非英语字母可能没有对应的大小写，合并会存在许多缺陷。

1、字符

XML 指定的字符 均在 16 位的 Unicode 2.1 字符集。

2、命名

XML 命名必须以字母、下划线或冒号 开头，后面跟着的是 有效命名字符（数字、减号、点）。

实际应用中不应该使用冒号，除非是用作命名空间修饰的分隔符。

字母并非局限于 ASCII 码，这一点是非常重要的。

8.1.3 文档部分

格式正规的 XML：

1、一个可选的序言（prolog）

2、文档的主体（body）

3、可选的“繁杂”的尾声（epilog），包括：注释、处理指令（Processing Instruction，PI） 和 /或 紧跟在元素后面的 空白。

8.1.4 元素

元素是 XML 标签的基本组成部分。

元素使用标签（tag）进行分隔：尖括号围住元素类型名。每一个元素 都必须 由一个起始标签 和 一个结束标签 分隔开。

空元素<WebService></WebService>

只是指定一个点，而不是提供一个包容器，空元素可以用缩略形式表示，起始和结束 标签的混合体。

<WebService/>

文档元素，每个文档 有且只有一个 根节点，称为 文档实体（document entity） 或 文档根（document root），它们的根被称为 文档元素（document element）。

XML 对元素 必须正确地嵌套。

如果字符串中包含单引号，分隔符必须使用双引号，反之亦然。

8.1.5 字符数据

字符数据就是任何不是标记的文本，小于号、大于号、&号 是标记分隔符，因此他们绝不能以字符串的形式出现在字符数据中（CDATA 部分除外），必须使用转义字符 “&It;”等。

8.1.6 属性

元素是 XML 中的名词，属性是它的形容词。

attribute name = "attribute alue"

attribute name = 'attribute alue'

起始标记或空标记中 属性只允许有一个实例存在。

非法的：<img src="abc.jpg" src="edf.jpg">

XML 数据中，只有 4 个字符 可以作为 空白使用，09 水平指标（HT），0D 回车（CR），0A 换行（CF），20 空格。

8.1.7 注释

<!--comment text-->

8.1.8 CDATA 部分

是一种用来包含文本的方法，对希望在自己的文档中 包含 XML 标记的使用举例 的作者来说是最有用的。

使用这些部分时 XML 几乎所有的优势都丧失殆尽。

<![DATA[...]]>，“...”可以是任何字符串，只要不包含字符串“]]>”。

8.1.9 格式正规的文档

元素和元素之间唯一的直接关系就是 父子关系；

兄弟关系是通过数据结构推断出来的，既不直接也不可靠，因为元素可能被插入到 某个元素和它的一个或多个子元素之间。

数据对象 如果满足下列条件 就是各市正规的文档。

1、语法合乎 XML 规范。

2、元素构成一个层次树，只有一个根节点。

3、没有对外部实体的引用，除非提供了 DTD。

任何 XML 解析器 发现 不是个是正规的结构，就报告一个“致命”错误，致命错误不一定导致解析器终止操作，但它不再会以正常的方式向应用程序传递字符数据 和/或 XML 结构。

8.2 XML 命名空间

8.2.1 命名空间

XML 命名空间 是 解决多个 义性和名字冲突问题的方案。

命名空间是一组具有结构的名称的集合。

8.2.2 定义和声明 命名空间

命名空间 推荐标准为我们提供了 xmlns 属性，属性值就是 URI。

命名空间前缀经常被提及为前缀，而名称本身是基本名。

默认的命名空间（没有声明别名的，形式为 xmlns="..."），在声明作用域里 所有没有经命名空间前缀修饰的 名称 被假定属于默认的命名空间。

8.3 DTD

一个 XML 文档是有效的，则它必须满足：文档 和 文档类型 相关联。

8.3.1 什么是 DTD

DTD 文档类型定义。

主要 用来查看 XML 文档的格式，出现在 XML 文档的序言中，DTD 声明不是必须出现的。

DTD 中 主要定义以下几个方面的内容：

1、元素声明。

2、实体声明。

3、属性的种类。

8.3.2 为什么引入 DTD

提供一种验证的手段，对 XML 来说是一大贡献，确保 XML 文件确实地遵守了 指定的格式，而这个格式可能是 一个 标准，或者是数据交换双方 所共同定制 的 协议。

实现了 文件格式 的统一化，提高了文件的重用性。

使用 DTD 进行验证，增加了操作时间。

8.3.3 DTD 的声明

略。

8.3.4 元素的声明

略。

8.3.5 实体的声明

实体（entity）是一些预先定义好的数据。

存储部位，内部实体，外部实体；

组成内容，可分解实体，不可分解实体。

引用方式，一般型实体，参数型实体。

不同类型的实体声明和使用方法略有不同。

略。

8.3.6 属性的声明

良构 XML 文档中，属性只要满足命名规则就可以了，但是在一个有效的 XML 文档中，属

性要经过 DTD 的属性声明。

DTD 声明中，属性的声明语法可以归纳为如下形式：

<!ATTLIST 元素名称 属性名称 属性类型 属性默认值类型>

元素名称指的是 属性所属的元素名称。

8.4 XML Schema

DTD 尽管进行了很大的简化，但还是一门 风格 和 XML 完全不同的语言，而 schema 文档是一种特殊的 XML 文档，容易学习和使用。

DTD 的另一个缺点是 数据类型相当有限。DTD 中根本不提供 数值数据 类型。

一个 XML 文档只能使用一个 DTD 文档，schema 则采用了 名域空间的机制，使得一个 XML 文档可以调用多种 schema 文档。

8.4.1 逻辑 XML Schema 的文档结构

略。

8.4.2 元素的定义

略。

8.5 可扩展样式表语言

（eXtensible Stylesheet Language，XSL）是描述 XML 文档样式信息的一种语言，W3C 制订。

XML 的一个优点就是 形式与内容相分离，XSL 就是它的两种样式表单之一，

另一种是 层叠样式表（CSS），是一种静态的样式描述格式，其本身不遵从 XML 的语法规范。

而 XSL 是一个 XML 文档。

是 XML 的一种具体应用。

它有两大部分组成：

第一部分描述了 如何将 XML 文档进行 转换、转换为可浏览或可输出的格式；

第二部分定义了 格式对象（Fomatted Object，FO）源树转换为可以显示的结果树，称为树转换，按照 FO 分析结果树，产生一个输出结果，这个过程称为 格式化。

转换树 日趋成熟，已从 XSL 中分离出来，另取名为 XSLT（XSL Transformations），现在一般所听说的 XSL 大多是指 XSLT。

一同退出的还有 配套标准 Xpath（XML Path Language，XML 路径语言）

在 XML 中 声明 XSL 样式单：

<?xml-styleshee type = "text/xsl" href = "mystyle.xsl" ?>

XSL 在网络中的应用大体分为两种模式：

1、服务器端转换模式

XML 文件下载到 浏览器前先转换成 HTML。

1.动态方式，接到转换请求时再进行实时转换。

2.批量方式。

2、客户端转换模式

XML 和 XSL 文件都传送到客户端，浏览器必须支持 XML+XSL 的工作方式。

8.5.2 XSLT 的常用语法和函数

略。

8.6 其他相关规范

8.6.1 XPath

采用简洁的、非 XML 语法，基于 XML 文档的 逻辑结构，在该结构中进行导航。

XPath 表达式 通常出现在 URL 和 XML 属性值里。

XPath 将 XML 文档描绘为 树或节点 的模型，节点的类型有 根节点、元素节点、属性节点、文本节点、注释节点、名称空间节点、处理指令节点 7 种。

XPath 规范定义了两个主要部分：一部分是表达式语法，另一部分是一组名为 XPath 核心库的基本函数。

指向某个 XML 文档中一个特定节点的路径 由三部分信息构成：一个轴类型、一个节点测试 和 谓词。

轴类型 有多种，指定所选节点和环境之间的关系。节点测试 查找什么类型的节点，测试包括通配符“*”、text()、node()、comment()、processing-instruction()等。

谓词以“[”开始，以“]”结束，谓词通过使用内部函数来 过滤不需要的节点。

<轴>::<节点测试>[<谓词表达式>]

8.6.2 XLink 和 XPointer

XLink 指定一个文档如何连接到另一个文档，XPointer 指定文档内部的位置，都是基于 XPath 推荐标准。

第九章 面向构件的软件设计

9.1.1 术语、概念

1、构件

构件的特征如下：

独立部署单元。

作为第三方的组装单元。

没有（外部的）可见状态。

独立可部署，意味着 必须能 跟他所在的环境 及 其他构件 完全分离。

原子性，构件不但必须具备足够好的内聚性，还必须将自己的依赖条件和所提供的服务说明清楚。

缓存具有这样的特征：当它被清空时，除了可能会降低性能以外，没有其它后果。

构建本质上没有状态，同一操作系统进程中 装载多个构件的拷贝 是毫无意义的，至多会存在一个特定构件的拷贝。

许多系统中，构建被实现为 大粒度的单元，工资管理服务程序就是一个构件，工资数据只是实例（对象），将不易变的“模型”和易变的“实例”分离的做法避免了大量的维护问题。

2、对象

对象的特征如下：

一个实例单元，具有唯一的标志。

可能具有状态，此状态外部可见。

封装了自己的状态和行为。

显式存在的实例化方案称为类，也有隐式的实例化方案，既通过克隆一个已存在的对象来实现，即原型对象。

新生的对象都必须被设置一个初始状态，创建与初始化 对象 的代码可以是一个静态过程—

—类的一部分，称为 构造函数。

如果这个对象是专门用来创建与初始化对象的，称为 工厂。

对象中 专门用来返回其他 新创建的对象的方法 称为 工厂方法。

3、构件与对象

构件通常包含了若干类 或 不可更改的 原型对象。还包括一系列对象。

但构件并非一定要包含类元素，它甚至可以不包含类，可以拥有传统过程体，甚至全局变量。

构件创建的对象——更确切地说是对这些对象的 引用——可以与该构件分离开来，并对构件的客户可见。构件的客户通常是指其他构件。

一个构件可以包含多个类元素，但是一个类元素只能属于一个构建。将一个类拆分进行部署通常没有什么意义。

4、模块

模块化方法成熟的标志是其对分离编译技术的支持，包括跨模块的正确的类型检查能力。

模块没有实例化的概念，在任何情况下，模块都可以包含多个类。类之间的继承关系并不受模块界限的限制。

模块本身就可以作为一个最简单的构件，这些库是功能性的，而不是面向对象的。

资源可以参数化一个构件，重新配置该构件而无需更改构件代码，例如，本地化设置可以通过资源配置实现。

某些情况下，模块并不适合作为构件，构件没有外部可见的状态，但是模块却可以显式地用全局变量来使其状态可见。

5、白盒抽象、黑盒抽象 与 重用

白盒抽象中，可以通过继承对构件的实现细节进行修改，白盒方式中实现细节对外界是完全可见的。

绝大多数系统中，（Application Programming Interface，API）相当于黑盒重用这些接口的实现。

白盒重用不可以轻易地被另外的软件替换，因为 依赖于 细节。

软件构件是一种组装单元，它具有规范的接口规约和显式的语境依赖，软件构件可以被独立地部署并由第三方任意地组装。

6、接口

接口是一个已命名的一组操作集合。

一个构件可以有多个接口，每个接口提供一种服务。

尽量不要重复引入功能相近的接口。

推行标准化，可能会由于笨拙官僚的“委员会设计”问题而不能达到最优；市场竞争，的 非技术本质 也可能导致结果不是最优。

接口标准化 是对消息的 格式、模式、协议 的标准化，XML 提供了一种统一的数据格式。

7、显式语境依赖

对部署环境的具体要求，称为语境依赖。

8、构件的规模

最大化重用 也有一个潜在的缺点——语境依赖的爆炸性增长。

语境依赖越多，能满足构件环境需求的客户构件就越少，降低了可用性。

构件设计者需要为以上两者找到一个平衡点，还必须考虑环境的演化会使构件更加脆弱。

9.1.2 标准化 与 规范化

如果语境依赖能够被广泛支持，就不是什么缺点。

1、通用市场与专业市场

通用市场的标准化是非常困难的，得满足所有人的需求，网络标准就是最好的例子。

专业市场的标准化与通用市场同样艰辛，由于所涉及的人较少，市场经济的机制就不容易很好地发挥作用。

2、标准的构件体系 与 规范化

要发挥标准化的作用，就必须使与之竞争的其他标准数目尽量很少。

9.3 构件框架

9.3.1 体系结构

构件体系结构的核心包括：构件和外部环境的交互；构件的角色；标准化工具的界面；对最终用户和部署人员的用户界面 等。

1、体系结构的角色

体系结构是关于一个系统的整体视图，定义了总体的不变性，规定了恰当的框架，限制自由度，对整体功能、性能、可靠性、安全性 的主要考虑过细的决策可以放一边。

3、构件系统架构特性

构件系统 体系结构 由一组 平台决策、一组 构建框架 和 构件框架之间的 互操作设计 组成。平台是允许在其上安装构件和构件框架的一个基础设施。

构件框架是一种专用的体系结构，常常实现一些协议以连接构件。

多数原子构件永远都不会被单独部署，尽管他们可以被单独部署。

原子构件通常组成地部署。

4、分层的构件体系结构

传统的垂直分层，自底向上地，抽象程度渐增，与应用相关的性质逐渐提高。

水平分层是性能和资源相关性递减而结构相关性递增。

轻量级体系结构把注意力集中到一个问题，而不是覆盖所有问题，如果轻量级构件支持较好的易扩展性，它的商业价值就非常大。

6、构件与生成式编程

必须要精确控制实际的构件边界，包括提供接口和需求接口，必须能精确控制同其他构件间的静态依赖。

9.3.2 语境相关组合构建框架

COM+ 增加了可租赁线程“套间”的概念，一次只允许一个线程入住，但是多个线程能顺序地入住该“套间”。

相同事务域中的对象 共享一个单独的逻辑线程和一个单独共享事务资源集合，一旦线程从事务域中返回，事务要么提交要么终止。

COM+中，如果两个构件共享一组兼容的语境属性集，则它们可以被看作是处于同一域中。

9.3.3 构件开发

异步问题

事件分发机制负责接收这些事件对象，并把它们发送给对其感兴趣的其他构件实例。

多线程

多线程主要关注于对程序执行进行更好的分配,获取性能最大化的手段却根本不依赖于多线程，而是尽量在第一时间内以最快的速度处理用户的请求。

第十章 典型架构

10.1 OMG 方式

对象管理组 OMG，通过规范化对象 开放市场的 所有层次上的互操作性。

10.1.1 对象请求代理

CORBA 的主要目标就是 使用不同语言、不同实现、不同平台 能进行交互。

CORBA 三个基本部分：一套调用接口、对象请求代理 ORB、一套对象适配器。

10.1.2 公共对象服务规范

两类服务：一类服务应用于企业计算系统。一类服务应用于细粒度的对象操作，但目前这些服务的实用价值较差。

1、支持企业分布式计算的服务

1.命名服务、交易器服务

命名服务 允许 任意地给对象赋予一个名字，这个名字在其所属的命名语境中是唯一的。

命名语境所形成的层次结构，使得所有的名字形成名字树。

交易器服务 允许给对象 赋予一个复杂的描述，从而允许客户基于该描述来定位所需的对象。

搜寻结果往往是 满足查询条件的 一组对象列表。

2.事件服务、通告服务

事件服务 允许定义那些 从 时间生产者 被 发送到时间消费者 的事件对象。

信息只能从生产者流向消费者，事件必须通过事件通道传播，事件可以具有类型，而通道可

以根据类型过滤事件。

事件通道支持“推”“拉”两种方式 的事件通告模型。

通告服务为事件服务增加了几个重要的特征——服务质量 QoS 规范 和 管理。

3.对象事务服务

对象事务服务 OTS，是建立分布式应用最重要的服务之一。

OTS 实现必须支持平坦事务，而嵌套事务是可选的。

在基于构件的系统中，嵌套事务似乎不可避免。

平坦事务在构件系统中的价值有限，实际上，现有的主流事务中间件也不支持嵌套事务。

6.并发控制服务

支持对象资源进行 加锁、解锁。

锁必须依赖于 事务的语境 或 其他语境才能获得。

读锁、写锁、升级锁。

读锁允许多个客户同时执行读操作，写锁允许一个客户写操作，升级锁是可以升级为写锁的读锁 支持互斥读。

每个受保护的资源都拥有一个锁集合。锁集合 不是事务型 就是非事务型，并可与其他锁集合建立关联。

8.生命周期服务

支持 创建、复制、移动、删除 CORBA 对象，及其相关的对象组。

包含关系支持嵌套复制。

11.外部化服务

支持对象网 和 对象流 之间的双向映射。对象网外部化后 再内部化 意味着创建该对象网副本。

外部化服务并不保证引用的完整性，仅保留同时外部化的对象之间的引用。

对象必须实现 Streamable 接口才能被外部化。

12.属性服务

允许将任意的属性与对象关联起来，被关联的对象必须实现 ProperySet 接口。

13.对象查询服务

依靠属性定位对象。

15.时间服务

拥有众多异步时钟的分布式系统 固有的误差问题。

10.1.3 CORBA 构件模型

CORBA 对象适配器主要的作用 就是在一个 ORB 和 真正接收调用并且返回结果的 对象之间 进行交互。

10.2 SUN 公司的方式

10.2.1 Java 构件技术的概述

Java 中，编译器会检查 Applet 代码的安全性，通过了编译器检查的 Applet 代码不会带来安全隐患。

由于编译得到的字节码仍然可能被人修改，代码在装载时刻会被再次检查（称为“校验”）。

运行环境（Runtime Environment，RE）、软件开发工具包（Software Development Kit，SDK）、参考实现。

运行环境是 Java 虚拟机 和 必须具有的 J2SE API 的实现。

10.3 Microsoft 的方式

微软选择的是最简单的路线，他没有提出一整套标准；相反，他不断对已有的应用和平台基础进行再工程，这就可以获益于以前的成功技术。

语言无关性，作为 CLR 的一条主要原则。

10.3.1 第一个基础关联模型——COM

COM 所定义的一个基础实体是接口。在二进制层面上，一个接口被表示为指向一个接口节

点的指针。

接口节点 唯一被指定的部分是 置于其内部第一个域的 另一个指针，这个指针指向一个过程变量表（或者说，函数指针表）。

每个 COM 对象都有 IUnknown 接口，通常置于 COM 对象图的顶端。

他的“真实”名字是他的 IID，即 00000000-0000-0000-C000-000000000046 为了方便，所有接口也有一个 可读名。

根据习惯，可读接口名以字母 I 开头。与 IID 不同，可读接口名 并不保证是唯一的。因此，编程中的接口引用均使用 IID。

IUnknown 接口的首要用途是在 最抽象的情况下 标志 COM 对象，此时 COM 对象 没有任何特殊功能。

IUnknown 接口 只提供对任何 COM 接口都必须的三个强制性方法。QueryInterface、AddRef、Release，后两个强制性方法被用来控制对象的生命周期。

类型 HRESULT 被大多数 COM 接口的方法用来表示调用成功或失败。 QueryInterface 表明查询的接口是否被支持。

每个 COM 对象都会进行引用计数，引用计数变量被共享使用的情况下，COM 对象 不能释放接口节点。

一般这样做没有问题，也是通常的做法。

某些情况下占用很多资源，可以使用独立的引用计数变量，以便节点可以尽早释放。这种根据需要创建和删除接口节点的技术有时被称作“快速装卸接口（Tear-Off Interface）”

10.3.2 COM 对象重用

COM 不支持任何形式的实现继承。

COM 支持两种形式的对象组装：包含（Containment）和 聚集（Aggregation）。

包含 是一个对象 拥有 指向另一个对象的唯一引用。

外部对象 只是把请求转发给内部对象，所谓转发 就是调用内部对象的方法。

包含能重用内含于其他构件的实现，是完全透明的。

如果包含层次较深，或者被转发的方法本身相对简单，包含会存在性能上的问题。因此 COM 定义第二类重用形式，聚集。

聚集直接把内部对象接口引用传给外部对象的客户，而不是再转发请求。

保持透明性是很重要的,因为外部对象的客户无法辨别哪个特定接口是从内部对象聚集而来的。

10.3.3 接口和多态

COM 接口可通过（单）接口继承 从其他 COM 接口中派生。

COM 的接口继承与其支持的多态无关。

接口和版本化，一旦公布，COM 接口和他的规范不允许以任何形式改变。

既解决了语法问题，也解决了弱基类问题。

IID 可用于标志接口中的版本，因为接口总是通过 IID 被请求。

CORBA 讨论中所提及的传递性版本冲突问题 在 COM 中不会发生。

构件可以选择实现接口的多个版本，处理方式就像处理 别的不同接口一样。

基于 COM 的系统能并发支持旧接口和新接口。

10.3.4 COM 对象的创建 和 COM 库

创建 COM 类 的实例对象时，COM 需要把给定的 CLSID 映射为包含所请求的类的实际构件。COM 支持系统注册器，它类似 CORBA 存储器。

进程内（inprocess）服务器、本地服务器、远程服务器。

10.3.5 从 COM 到分布式 COM（DCOM）

代理（Proxy）对象 和 服务器 桩（Stub）对象。

为支持 跨进程 或 跨机器 的透明通信，COM 在客户端创建代理对象，在服务器端创建桩对象。

跨进程传递的 接口引用需要被映射为 对象引用。

DCOM 将数据整理成平台无关的网络数据表达（NDR）形式。

10.3.6 复合文档 和 OLE 对象

OLE 可被 概括为 一组预定义的 COM 接口。

文档容器 和 文档服务器。

文档服务器 是提供某种内容模型 和显示、操作内容的能力。文档容器没有自己的内容，但可以接受任意文档服务器提供的内容成分。

许多文档容器也是文档服务器，即是说，他们支持外来的成分，同时也有自己的内容。

10.3.7 .NET 框架

没有原始类型。

好玩的：Java 程序员和.NET 程序员

1、对对方的看法

Java 程序员都看不起.NET 认为没技术含量。

.NET 程序员大都仰慕 Java，认为很 NB。

2、对新技术的看法

Java 社区发明一个新技术会被 Java 程序员拿来到处炫耀，说这是世界上最 NB 技术，完全不用了解这玩意儿干什么的。

MS 发明一个新技术会被.NET 程序员说自己又贬值了，完全不用考虑这个技术能给自己带来什么。

3、解决问题的方式

Java 程序员遇到无法解决的问题后，会认为是自己的思想有问题，所以整天看 Think in XXXX 之类的书。然后去社区上问，花了 N 天后选择一种最 NB 的技术解决，然后说这个需求太 SB。

.NET 程序员遇到无法解决的问题后，会认为自己没找到解决问题的控件，所以去翻 Action in XXXX 之类的书。然后去用金币银币提问，花了 N 个小时后，从网上抄一段代码解决。

4、看待对方阵营的新技术

Java 程序员看到.NET 的新技术后，首先是说这是傻子才会用的傻瓜技术。当发现这个技术 Java 社区也在追捧的时候，就会说 MS 是从 Java 抄的。

.NET 程序员看到 Java 的新技术后，首先是埋怨微软尽搞些没技术含量的东西。当发现这个技术也有.NET 版本的时候，就会说微软什么都能抄。

5、各自社区“牛人”

Java 社区的“牛人”是有着各种 NB 思想的人，他们经常阅读国外大牛的文章，用双语（这里指的是中英混杂）写文章，抛出一个又一个语录，然后赢来众人喝彩，不过所有人都搞不清为什么要这样。

.NET 社区的“牛人”是有着各种 SB 代码的人，他们的博客上有着无数的 Demo 和 Code，对于初级程序员提出的任何问题他们都有现成的代码，然后赢来更多的问题和星星钻石。

6、面对对方阵营的指责

Java 程序员一旦发现有人说 Java 不好首先第一反应就是那人是傻子，然后不屑一顾，然后搬出很多 Java 的牛人和 ZB 软件，当然他们最喜欢的还是说语言排名。

.NET 程序员一旦发现有人说.NET 不好首先第一反应就是自己是傻子，然后反省自己是不是该选择.NET，如果有.NET 的大牛出来声援就会在旁边鼓掌。然后换个马甲提问我该选择 Java 还是.NET。

一个很简单的功能，.net 只要一个小时，可能 java 要花 4-5 个小时～，那些干 java 的，还以为自己这 4-5 个小时一直在干有技术的活，呵呵

第十一章 信息安全

11.1 信息安全关键技术

11.1.1 加密和解密

有意的计算机犯罪 和 无意的数据破坏

被动攻击：非法地从传输信道上截取信息，或从存储载体上 偷窃、复制 信息。

主动攻击：对传输或存储的数据进行 恶意的删除、篡改 等。

密码技术是防止数据攻击的一种有效而经济的方法。

信源、信宿、明文、密文。

传输消息的通道称为信道，参数 称为 密钥，解密算法是加密算法的逆运算。

加密密钥与解密密钥相同，或者可以简单相互推导 的密码体质 称为 对称密码体质。

不能（在有效时间内）相互推导的，称为 非对称密码体质。

1、对称密钥密码体质及典型算法

对称算法（Symmetric Algorithm），有时又称为 传统密码算法，也称 单密钥算法。

安全通信之前，商定一个密钥，安全性依赖于密钥，密钥的保密性对通信至关重要。

优点：算法实现的 效率高、速度快。

缺点：密钥的管理过于复杂。

1. DES 算法简介

DES（Data Encryption Standard，数据加密标准）是 IBM 公司研制，美国国家标准局 1977 年公布，作为 非机要部门 使用的数据加密标准。

DES 是一个分组加密算法，以 64 位为分组对数据加密。密钥长度 56 位（因为每个第 8 位都用作奇偶校验）。

2. IDEA 算法简介

国际数据加密算法（International Data Encryption Algorithm，IDEA）前身是 推荐加密标准（Proposed Encryption Standard，PES）。

分组长度 64b，密钥长度 128b。

运算非常简单，只是 异或，速度极快，穷举破解不现实。

2、不对称密码加密算法

不对称密码体制又称 双密钥和公钥密码体质，1976 年 由 Diffie 和 Hellman 提出的。

私钥 秘密保存。

不需要事先通过安全秘密管道交换密钥。

RSA 的安全性依赖于大素数分解。公钥和私钥 都是两个大素数（大于 100 个十进制 位）的函数。

据猜测，从一个 密钥和密文 中，推断出明文的难度 等同于 分解两个大素数的 积。

具体操作时 考虑到 安全性 和 M 信息量 较大等因素，一般是 先做 HASH 运算。

速度慢一直是 RSA 的缺陷，因此一般来说，RSA 只用于少量数据加密。

11.1.2 散列函数与数字签名

1、MD5 散列算法

散列函数是一种公开的数学函数。散列函数运算的输入信息叫做 报文，运算后所得的结果叫做 散列码 或消息摘要。

特点：

1. 给定 M，要找到另一消息 M，使 H（M）= H（M'）很难。

2. 散列函数都是 单向的，反推 M 很难。

3. 对于任何一个报文，无法预知它的散列码。

4. 散列码具有固定的长度，不管原始报文长度如何。

常见的散列函数有：MD5、SHA、HMAC 等。

MD5（Message Digest 5）已成为国际标准，产生 128 位（16 字节）长度的散列值（或称 消息摘要）。

通过以下 4 个步骤：

1. 附加填充位，填充后数据长度 MOD 512 后 余 448。如果数据长度正好 MOD 512 余 448，增加 512 个填充位，填充个数也就是 1~512。

填充位第一个为 1，其余全部是 0。

2. 补足长度。

3. 初始化 MD 缓存器。

4 个 32 位寄存器，A、B、C、D，初始化为：

A: 01 23 45 67

B: 89 AB CD EF

C: FE DC BA 98

D: 76 54 32 10

4. 处理数据段。

2、数字签名与数字水印

1. 数字签名可以解决 否认、伪造、篡改、冒充 等问题。

凡是需要对用户身份进行判断的情况 都可以使用数字签名。

三个过程：系统的初始化过程、签名产生过程、签名验证过程。

签名者必须注意保护好私有密钥，因为它是公开密钥体系安全的重要基础。

如果密钥丢失，应该立即报告鉴定中心取消认证，鉴定中心必须能够迅速确定用户的身份及其密钥的关系。

RSA、ElGamal、 Fiat-Shamir、美国的数字签名标准/算法（DSS/DSA）、椭圆曲线 等多种。

2. 数字水印（Digital Watermarking）是实现版权保护的有效办法，也是信息隐藏技术研究领域的重要分支。

通过在原始数据中嵌入秘密信息——水印（Watermark）来证实该数据段所有权。

水印可以是一段 文字、标识、序列号 等，通常是不可见或不可察的，与原始数据紧密结合并隐藏其中。

数字水印技术必须具有较强的 鲁棒性、安全性、透明性。

数字水印主要应用领域：

版权保护，作品被盗版或出现版权纠纷时，所有者即可 从盗版作品或水印版作品中 获取水印信号作为依据。

加指纹，将不同用户端 ID 或 序列号 作为不同的水印（指纹）嵌入作品的合法备份中，一旦发现未授权的备份，就可以 确定它的来源。

标题与注释。

篡改提示，可将原始图像分成多个独立块，再将每个块加入不同的水印，来确定作品的完整性，这类水印必须是脆弱的，并且检测水印信号时，不需要原始数据。

使用控制，防复制。

空域算法、变换域算法、压缩域算法、NEC 算法、生理模型算法 等。

11.1.3 密钥分配中心与公钥基础设施

现代密码系统中，算法本身的保密已经不重要了，只要密钥能够保密，即使加密算法公开，甚至加密设备丢失，也不会对加密系统的坚固性和正常使用产生多大影响。

如何高效地分配密钥、安全地管理密钥 对保证数据安全来说 至关重要。

1、密钥分配中心

密钥自动分配 是 密钥分配中心（Key Distribution Center，KDC）技术。

2、数字证书和公开密钥基础设施

数字证书的内容一般包括：唯一标识证书所有者的名称、唯一标识证书签发者的名称、证书所有者的公开密钥、证书签发者的数字签名、证书的有效期、证书的序列号 等。

PKI（Public Key Infrastructure，公钥基础设施）的结构模型有三类实体：管理实体、端实体、证书库。

管理实体是 PKI 的核心，是服务的提供者，端实体是 PKI 的用户。

CA 和 RA 是两种管理实体，CA 能够 发布和撤销 证书，维护证书的生命周期。RA 负责处理用户请求。

证书库的存取对象为证书和 CRL，其完整性由数字签名来保证，因此不需要额外的安全机制。

11.1.4 访问控制

自动、有效 地 防止 对 系统资源进行 非法访问或者不当使用。

它是建立在身份认证的基础之上的。

1、身份认证技术

识别用户的身份有两种不同形式：身份认证、身份鉴定。

认证的方法 归结为 3 大类：知道什么、拥有什么、是什么。

是什么，是一种基于生物识别技术的认证。

1. 用户名和口令认证，三种简单的认证方式：明文传送、单向散列、单向散列函数和随机函数。

2. 使用令牌认证，密钥存储于令牌中。

令牌是一个可以加密存储并运行相应加密算法的设备，完成对用户必须 拥有 某物的验证。

令牌的实现分为：质询响应令牌、时间戳令牌，常用的是时间戳令牌。

系统的安全强度大大增加：私钥采用令牌存储的方式解决了 私钥自身的安全问题，安全强度大大增加。

而且令牌有 PIN 码 保护，对令牌的非法访问超过一定次数后，令牌会死锁。

时间戳令牌利用时间代替随机数，需要重点考虑时间同步问题，目前在安全性较高的认证系统中，多是采用这种方案。

3. 生物识别 与 三因素认证

基于生物识别技术的认证，主要根据认证者的 图像、指纹、气味、声音 等 作为认证数据。

2、访问控制技术

根据 控制手段 和 具体目的 的不同，通常将 访问控制技术 划分为几个方面：入网访问控制、网络权限控制、目录级安全控制、属性安全控制、网络服务器安全控制 等。

入网访问控制，控制准许用户 入网的时间、准许的工作站 等。

由于用户口令验证方式容易被攻破，很多网络都开始采用 基于数字证书的验证方式。

用户 和 用户组 被 赋予一定的权限。

访问机制 两种实现方式：“受托者指派”和“继承权限屏蔽”

“受托者指派”控制用户 和 用户组 如何使用网络服务器。

“继承权限屏蔽”相当于一个过滤器，可以限制 子目录 从 父目录 那里继承哪些权限。

特殊用户、一般用户、审计用户。

对目录和文件的访问权限 一般有 8 种：系统管理员权限、读、写、创建、删除、修改、查找、访问控制。

属性 能够控制以下几个方面的权限：写 数据、复制 文件、删除 目录或文件、察看 目录和文件、执行 文件、隐含 文件、共享、系统属性 等。

11.1.5 安全协议

1、IPSec 协议简述

因特网工程任务组（IETF），IPSec 在 IP 层上 对数据包 进行高强度 的 安全处理 提供 数据源验证、无连接数据完整性、数据机密性、抗重播、有限通信流机密性 等 安全服务。

1. IPSec 协议工作原理

通过使用两种信息安全协议 来为数据报提供高质量的安全性：认证头（AH）协议 和 封装安全载荷（ESP）协议，以及像 Internet 密钥交换（Internet Key Exchange，IKE）协议这样的密钥管理协过程和协议。

IPSec 允许系统或网络用户 控制安全服务提供的 粒度。

由通信双方建立的安全关联（Security Association，SA）来提供。

3. IPSec 协议 安全性分析

可以应用于所有跨越网络边界的通信。

如果所有来自外部的通信必须使用 IP，且防火墙是 Internet 与 组织的唯一入口，则 IPSec 是不能被绕过的。

IPSec 位于传输层（TCP、UDP）之下，因此对应用程序是透明的，实现时，没有必要 在用户或服务器上更改软件。

IPSec 对最终用户是透明的，没有必要 培训用户掌握安全机制。

2、SSL 协议

SSL 协议（Secure Socket Layer）对计算机之间的 整个会话进行加密，位于 TCP 和 应用层 之间，可为应用层 提供 安全业务，主要是 Web 应用。

基本目标是 在通信双方之间 建立安全的连接。

SSL 协议工作原理

两个重要的概念：SSL 连接 和 SSL 会话。

连接 是 提供恰当类型服务的 传输，对于 SSL ，连接是 点到点 的关系。

SSL 的会话是 客户 和 服务器 之间的 关联，会话 通过握手协议来创建。

会话定义了 加密安全 参数的 一个集合。

会话可以用来 避免 为每个连接进行 昂贵的 新安全参数的 协商。

3、PGP 协议

1. PGP 协议的定义

PGP（Pretty Good Privacy）针对 电子邮件 在 Internet 上 通信的 安全问题而设计的一种 混合加密系统。

公钥密码 和 分组密码 是在同一个系统中，PGP 的用户拥有一张公钥表。

PGP 应用程序具有很多优点：速度快、效率高、可移植性好。

2. PGP 协议 的 加密过程

用 IDEA 算法对明文加密，接着用接收者的 RSA 公钥 对这个 IDEA 密钥进行加密。

PGP 没有用 RSA 算法直接对明文加密，而是对 IDEA 密钥 进行加密。

由于 IDEA 算法 速度很快，所以不会因为邮件的数量大而耽误时间，而 IDEA 的密钥位数较少，所以使用 RSA 算法在速度上也不会有很大影响。

11.1.6 数据备份

1、备份的类型

备份数据常常被人们遗忘，造成的后果往往是 毁灭性的。

保证数据完整性以及准确性，一直都面临着极大的考验。

1. 完全备份，所需时间最长，但恢复时间最短，操作最方便可靠。

2. 差异备份，备份上一次的 完全备份后发生变化的所有文件。备份时间较长，占用空间较多，恢复时间较短。

3. 增量备份，上一次备份后，所有发生变化的文件。备份时间较短，占用空间较少，恢复时间较长。

4. 按需备份。有很好的选择性。

2、异地备份

数据异地备份 是 容灾系统 的核心技术，确保一旦本地系统出现故障，远程的容灾中心能够 迅速 进行完整的业务托管。

进行异地备份时，要注意以下几个问题：

避免让备份带上病毒。

保证磁片质量，定期对其进行质量检查。

对于光盘，最大的缺点是 兼容性不好，最好是用哪台刻录机刻录 就用哪台刻录机读取（有时光头歪了也刻歪了，好光驱读不出来）。

对于移动硬盘，要做磁盘检查，保证其性能良好。

3、自动备份软件

具备 多个备份的文件 无论怎样重命名 都只备份一个。

对员工正常工作无任何干扰，就好像这个软件不存在一样。

11.1.7 计算机病毒免疫

1、计算机病毒定义

通过修改其他程序 使之含有该程序本身或它的一个变体，具有感染力，借助 使用者的权限 感染他们的程序。

每个被感染的程序也像病毒一样 可以感染其他程序。

2、计算机病毒免疫的原理

传染模块一般 包括 传染条件判断 和 实施传染 两部分。

一般情况下，传染完一个对象后，都要给被传染对象加上传染标识，若不存在这种标识，则病毒就对该对象实施传染。

不判断是否存标识，反复传染多次，雪球越滚越大。

当某些尚不能被病毒检测软件检查出来的病毒感染了文件，该文件又被免疫外壳包在里面时，查毒软件查不到它。

11.2 信息安全管理和评估

11.2.1 安全管理技术

安全管理技术 就是 监督、组织、控制 网络通信服务 以及 信息处理所必须的 各种技术手段 和 措施 的总称。

目标是 确保 计算机网络的 持续正常运行，出现异常时 能及时 响应和排除 故障。

各种网络安全产品的作用 提现在网络中的不同方面，统一的网络安全管理平台 必然要求对网络中部署的安全设备 进行协同管理。

安全设备的管理、安全策略管理、安全风险控制、安全审计，几个方面。

安全审计：安全设备、操作系统、应用系统 的 日志信息 收集汇总，进一步分析，得出更深层次的分析结果。

第十二章 系统安全架构

12.1 信息系统安全架构的简单描述

信息安全的特征 是为了保证信息的 机密性、完整性、可用性、可控性、不可抵赖性。

以风险策略为基础。

12.1.1 信息安全的现状及其威胁

计算机和网络的普及，会产生两个方面的效应：

其一，各行各业的业务运转几乎完全依赖于计算机和网络。

其二，大多数人对计算机的了解更加全面。

常见的安全威胁有如下几种：

1、信息泄露。

2、破坏信息的完整性。

3、拒绝服务。

4、非法使用。

5、窃听。

6、业务流分析，发现有价值的信息和规律。

7、假冒。

8、旁路控制。

9、授权侵犯。

10、特洛伊木马。

11、陷阱门，设置了“机关”，提供特定的输入数据时，允许违反安全策略。

12、抵赖。

13、重放，处于非法的目的而被重新发送。

14、计算机病毒。

15、人员不慎。

16、媒体废弃。

17、物理侵入。

18、窃取。

19、业务欺骗。

可以从安全技术的角度提取出 5 个方面的内容：认证鉴别、访问控制、内容安全、冗余恢复、审计响应。

12.2 系统安全体系架构规划框架及其方法

安全技术体系架构过程的目标 是 建立可持续改进的安全技术体系架构的能力。

OSI 参考模型：物理、数据链路、网络、传输、会话、表示、应用。

根据网络中风险威胁的存在实体 划分出 5 个层次的实体对象：应用、存储、主机、网络、物理。

信息系统安全规划是一个 非常细致和非常重要的工作，需要对企业信息化发展的历史情况进行深入和全面的调研。

信息系统安全体系主要是由 技术体系、组织结构体系、管理体系 三部分共同构成。

技术体系 由 物理安全技术 和 系统安全技术 两大类组成。

组织体系 由 机构、岗位、人事 三个模块构成。

管理体系 由 法律管理、制度管理、培训管理 三部分组成。

人员安全包括 安全管理的组织结构、人员安全教育与意识机制、人员招聘及离职管理、第三方人员安全管理 等。

12.3 网络安全体系架构设计

12.3.1 OSI 的安全体系架构概述

在 OSI 7 层协议中 除会话层外，每一层 均能提供相应的安全服务。

最适合配置安全服务的是 物理层、网络层、运输层、应用层。

ISO 开放系统互联 安全体系 的 5 类安全服务：鉴别、访问控制、数据机密性、数据完整性、抗抵赖性。

分层多点安全技术体系架构，也称为深度防御安全技术体系架构，通过以下方式将防御能力 分布至 整个信息系统中。

1、多点技术防御，从内部或外部 多点攻击一个目标，通过对以下多个防御核心区域 的防御 达到抵御所有方式的攻击的目的。

1. 网络和基础设施，确保可用性，确保机密性和完整性。

2. 边界，抵御主动的网络攻击。

3. 计算环境，抵御内部、近距离的分布攻击。

2、分层技术防御，有效的措施是 使用多个防御机制。

支撑性基础设施为 网络、边界、计算机环境 中 信息保障机制 运行基础。包括 公钥设施、检测和响应基础设施。

1. 公钥基础设施提供一种 通用的联合处理方式，以便安全地 创建、分发、管理 公钥证书和传统的对称密钥。

公钥基础设施 必须支持受控的互操作性，并与各用户团体所建立的安全策略 保持一致。

2. 迅速检测并响应入侵行为，便于结合其他相关事件 观察某个事件的“汇总”性能。

识别潜在行为模式 或者 新的发展趋势

12.3.2 鉴别框架

鉴别（Authentication）防止其他实体 占用和独立操作 被鉴别实体的身份。

鉴别有两种重要的关系背景：

一是实体由申请者来代表，申请者与验证者之间存在着特定的通信关系（如实体鉴别）。

二是实体为验证者提供数据项来源。

鉴别方式主要基于以下 5 种：

1、已知的，如 一个秘密的口令。

2、拥有的，IC 卡、令牌等。

3、不改变的特性，如 生物特征。

4、相信可靠的第三方建立的鉴别（递推）。

5、环境（如主机地址等）。

鉴别信息（Artificial Intelligence，AI）是指 申请者要求鉴别到 鉴别过程结束 所生成、使用、交换 的信息。

交换 AI、申请 AI、验证 AI。

鉴别服务分为以下阶段：安装阶段、修改鉴别信息阶段、分发阶段、获取阶段、传送阶段、验证阶段、停活阶段、重新激活阶段、取消安装阶段。

12.3.3 访问控制框架

访问控制（Access Control）决定 允许使用哪些资源、在什么地方适合阻止未授权访问的过程。

ACI（访问控制信息） 用于访问控制目的的 任何信息。

ADI（访问控制判决信息）做出判决时 可供 ADF 使用的部分（或全部）ACI。

ADF（访问控制判决功能）做出访问控制判决。

AEF（访问控制实施功能）。

涉及访问控制的有 发起者、AEF、ADF、目标。

12.3.4 机密框架

机密性（Confidentiality）服务 确保信息仅仅是对被授权者 可用。

数据只对那些拥有某种关键信息的人才是可访问的。

被保护的环境 被交叠保护的环境。

从一个环境移到另一个环境的数据的连续保护 必然涉及到 交叠保护环境。

机密性机制

数据的机密性 可以依赖于 所驻留 和 传输的媒体。

数据在传输中的机密性 能通过禁止访问的机制、隐藏数据语义的机制、分散数据的机制。

物理方法保证媒体的数据只能通过特殊的有限设备 才能检测到。

通过路由选择控制。

通过机密提供机密性。

12.3.5 完整性框架

完整性（Integrity）框架 目的是通过组织威胁或探测威胁，保护可能遭到不同方式危害的数据完整性和数据相关属性完整性。

所谓完整性，就是数据不以未经授权方式 进行改变或损毁的特征。

几种分类方式：未授权的数据修改、未授权的数据创建、未授权的数据删除、未授权的数据插入、未授权的数据重放。

依据是否包括恢复机制 分为 具有恢复机制的 和 不具有恢复机制的。

完整性机制的类型

1、组织对媒体访问的控制，包括 物理的、不受干扰的信息；路由控制；访问控制。

2、用以探测 对数据 或 数据项序列的 非授权修改的机制。

按照保护强度，完整性机制可以分为 不作保护；对修改和创建的探测；对修改、创建、删除、重复 的探测；对修改和创建的探测并带恢复功能；对修改、创建、删除、重复 的探测并带恢复功能。

12.3.6 抗抵赖框架

抗抵赖（Non-repudiation）服务 包括证据的 生成、验证、记录，以及 在解决纠纷时 随即进行的 证据恢复 和 再次验证。

目的是 提供有关特定事件或行为的证据。

当涉及消息内容的抗抵赖服务时，为提供原发证明，必须确认数据原发者身份和数据完整性。

为提供递交证明，必须确认接收者身份和数据完整性。

抗抵赖服务提供 在试图抵赖的事件中 使用的设备：证据生成、证据记录、验证生成的证据、证据的恢复和重验。

抗抵赖由 4 个独立的阶段组成：证据生成；证据传输、存储、恢复；证据验证；解决纠纷。

1、证据生成

卷入事件或行为中的实体，称为证据实体。证据实体可由证据实体、或可能与可信第三方的服务一起生成、或者单独由可信第三方生成。

3、证据验证

证据在 使用者的请求下 被证据验证者 验证。让证据使用者 确信 被提供的证据 确实是充分的。

12.4 数据库系统的安全设计

电子政务中所涉及的数据库 密级更高、实时性更强。

实现 数据库系统安全的 完整性、保密性、可用性。

安全策略一般为 用户管理、存取控制、数据加密、审计跟踪、攻击检测。

12.4.1 数据库安全设计的评估标准

1985 年，美国国防部颁布“可信计算机系统评估标准（Trusted Computer System Evaluation Criteria，TCSEC）”橘皮书（简称 DoD85）。

1991 年，美国国家计算机安全中心（The National Computer Seaurity Center，NCSC）颁布了“可信计算机评估标准关于可信数据库管理系统的解释（Trusted Database Interpretation，TDI）”。

TDI 是 TCSEC 在数据库管理系统方面的扩充和解释，从 安全策略、责任、保护、文档 4 个方面进一步描述了每级的安全标准。

按照 TCSEC 标准，D 类产品 基本没有安全保护措施，C 类产品 只提供了 安全保护措施，B 类以上产品 是实行强制存取控制的产品，是真正意义上的安全产品。

12.4.2 数据库的完整性设计

数据库的完整性 是指数据库中数据的 正确性和相容性。

由各种各样的完整性约束来保证，因此可以说数据库完整性设计就是数据库完整性约束的设计。

通过 DBMS 或应用程序来实现。

1、数据库完整性设计原则

1. 根据数据库完整性约束的类型 确定其实现的系统层次和方式，并提前考虑对系统性能的影响。

一般情况下，静态约束应 尽量包含在数据库模式中，动态约束由应用程序实现。

2. 实体完整性约束、参照完整性约束 是关系数据库最重要的完整性约束，尽量应用。

3. 要慎用 触发器，一方面性能开销较大；另一方面，多级触发不好控制，容易发生错误，最好使用 Before 型语句级触发器。

4. 在需求分析阶段就必须制定完整性约束的命名规范。

5. 要根据业务规则 对数据库完整性进行细致的测试。

6. 要专职的数据库设计小组。

7. 应采用合适的 CASE 工具 来降低数据库设计各阶段的工作量。

2、数据库完整性的作用

1. 能够防止合法用户使用数据库时 向数据库中添加不合语义的数据。

2. 实现业务规则，易于定义，易于理解，而且可以降低应用程序的复杂性，提高应用程序的运行效率。集中管理。

3. 能够同时兼顾 数据库的完整性和系统效能。

4. 有助于尽早发现应用软件的错误。

5. 数据库完整性约束 6 类：列级静态约束、元组级静态约束、关系级静态约束、列级动态约束、元组级动态约束、关系级动态约束。

动态约束通常由应用软件来实现。

3、数据库完整性设计示例

首先 需要在需求分析阶段确定要通过数据库完整性约束实现的业务规则。

然后 依据整个系统的体系结构和性能要求，遵照数据库设计方法和应用软件设计方法，合理选择每个业务规则的实现方式。

最后 认真测试，排除隐含的约束冲突和性能问题。

基于 DBMS 的数据库完整性设计大体分为以下几个阶段：

1. 需求分析阶段。

2. 概念结构设计阶段。

3. 逻辑结构设计阶段，就是将概念结构转换为 某个 DBMS 所支持的数据模型，并对其进行优化，包括对关系型的规范化。

每种业务规则都可能有好几种实现方式，应该选择对数据库性能影响小的一种，有时需通过实际测试来决定。

12.5 案例：电子商务系统的安全性设计

1、原理介绍

1. 验证（Authentication）：是否可以获得授权。

2. 授权（Authorization）：可以使用哪些服务。

3. 审计（Accounting）：记录用户使用网络资源的情况，用户 IP 地址、MAC 地址掩码 等。

2、软件架构设计

RADIUS 软件架构分为三个层面：协议逻辑层、业务逻辑层、数据逻辑层。

协议逻辑层 主要实现 RFC 框架中的内容，处理网络通信协议的 建立、通信、停止 方面的工作。

相当于一个转发引擎，起到分发处理的内容分发到不同的协议处理过程中。

业务逻辑进程分为：认证、计费、授权，三种类型。

数据库代理池 统一连接数据库，以减少对数据库系统的压力。同时减小了 系统对数据库的依赖性，增强了系统适应数据库系统的能力。

RADIUS 软件分层架构的实现：

一是 对软件风险进行了深入的分析，

二是 可以构建一个或多个重用的构件单元，也可以继承原来的成果。

RADIUS 的功能：

一是 实际处理大量用户并发的能力，

二是 软件架构的可扩展性。

负载均衡是提高 RADIUS 软件性能的有效方法，主要完成以下任务：

1. 解决网络拥塞问题，就近提供服务。

2. 为用户提供更好的访问质量。

3. 提高服务器响应速度。

4. 提高服务器及其他资源的利用效率。

5. 避免了网络关键部位出现单点失效。

第十三章 系统可靠性

13.1 软件可靠性

目前，硬件可靠性测试技术和评估手段日趋成熟，已经得到了业界的认可。

软件可靠性模型的研究多集中在 开发阶段、测试阶段、评估阶段 的可靠性模型。

13.1.2 软件可靠性的定义

可靠性（Reliability）是指产品在 规定的条件下 和 规定的时间内 完成规定功能 的 能力。

按照产品可靠性的形成，分为 固有可靠性、使用可靠性。

固有可靠性是通过 设计、制造 赋予产品的 可靠性。

使用可靠性 既受设计、制造 的 影响，又受使用条件的影响。

软件与硬件 从可靠性角度来看，主要有 4 个不同点：

1、复杂性，软件内部的逻辑高度复杂，硬件则相对简单。

2、物理退化，一个正确的软件任何时刻均可靠，一个正确的硬件、元器件、系统 则可能在某个时刻失效。

3、唯一性，软件是唯一的，软件复制不改变软件本身，硬件不可能完全相同，概率方法在硬件可靠性领域取得巨大成功。

4、版本更新快，软件版本更新较快，也给软件可靠性评估带来较大的难度。

1983 年，美国 IEEE 对“软件可靠性”做出了更明确的定义。

1989 年，我国国家标准 GB/T-11457 也采用了这个定义。

定义：在规定的条件下，在规定的时间内，软件不引起系统失效的概率。

依然沿用了“产品可靠性”的定义。

1、规定的时间

由于软件运行的环境与程序路径选取的随机性，软件的失效为随机事件，所以运行时间属于随机变量。

2、规定的条件

不同的环境条件下的可靠性是不同的，计算机的配置情况、对输入的要求。

有了明确规定的环境条件，还可以有效地判断软件失效的责任在用户方还是开发放。

3、所要求的功能

软件可靠性还与规定的任务和功能有关。

要准确度量软件系统的可靠性，必须先明确它的任务和功能。

4、“软件可靠性”定义具有如下特点：

1. 用内在的“缺陷” 和 外在的“失效”关系来描述可靠性。

2. 定义使人们对软件可靠性进行量化评估成为可能。

3. 用概率的方法描述可靠性是比较科学的。

13.1.3 软件可靠性的定量描述

软件的可靠性可以基于 使用条件、规定时间、系统输入、系统使用、软件缺陷 等变量构建的数学表达式。

1、规定时间：自然时间、运行时间、执行时间。

使用执行时间来度量软件的可靠性最为准确。

2、失效率：把软件从运行开始，到某一时刻 t 为止，出现失效的概率用 F(t)表示。

F(0)=0，即软件运行初始时刻失效概率为 0。

F(t)在时间域(0,+无穷大）上是单调递增的。

F(+无穷大)=1，即失效概率在运行时间不断增长时 趋向于 1，这也意味着任何软件都存在缺陷。

3、可靠度：在规定的条件下，规定的时间内 不发生失效的概率。

公式略。

4、失效强度（Failure Intensity）单位时间 软件系统出现失效的概率。

公式略。

5、失效率（Failure Rate）又称 风险函数（Hazard Function），也可以称为条件失效强度。

就是当软件在 0~t 时刻内 没有发生失效的条件下，t 时刻 软件系统的失效强度。

公式略。

6、可靠度 与 失效率 之间的换算。

略。

7、平均失效时间（Mean Time to Failure，MTTF）就是软件运行后，到下一次出现失效的平均时间。更直观地表明一个软件的可靠度。

公式略。

需要对 软件可靠度 这个反映软件可靠性的肚量指标作下列补充说明：

1. 需指明它与其他软件的界限。

2. 软件失效必须明确定义。

3. 必须假设硬件无故障（失效）和软件有关变量输入正确。

5. 必须指明时间基准：自然时间（日历时间）、运行时间、执行时间（CPU 时间）、其他时间基准。

6. 通常以概率度量，也可以模糊数学中的可能性加以度量。

7. 在时间域上进行，是一种动态度量，也可以是在数据域上，表示成功执行一个回合的概率。

软件回合是软件运行最小的、不可分的执行单位。

8. 有时将软件运行环境简单地理解为软件运行剖面（Operational Profile）。

运行剖面定义了关于软件可靠性描述中的“规定条件”，测试环境、测试数据 等一系列问题。

13.1.4 可靠性目标

使用 失效强度 表示软件缺陷对软件运行的影响程度。

不仅取决于软件失效发生的概率，还和软件失效的严重程度有很大关系。引出另外一个概念——失效严重程度类（Failure Severity Class）。

失效严重程度类 就是对用户具有相同程度影响的失效集合。

对失效严重程度的分级 可以按照不同的标准进行，对成本影响、对系统能力的影响 等。

对成本的影响 可能包括失效引起的 额外运行成本、修复和恢复成本、现有潜在的业务机会的损失 等。

对系统能力的影响 常常表现为 关键数据的损失、系统异常退出、系统崩溃、导致用户操作无效 等。

可靠性目标 是指客户对软件性能满意程度的期望。通常用 可靠度、故障强度、平均失效时间（MTTF）等指标来描述。

建立定量的可靠性指标 需要对 可靠性、交付时间、成本 进行平衡。

13.1.5 可靠性测试的意义

1、软件失效可能造成灾难性的后果。

2、软件的失效在整个计算机系统失效中的比例较高。

80%和软件有关。

结构太复杂了，一个较简单的程序，其所有路径数量可能是一个天文数字。

3、相比硬件可靠性技术，软件可靠性技术很不成熟。

4、软件可靠性问题是造成费用增长的主要原因之一。

5、系统对于软件的依赖性越来越强。

13.1.6 广义的可靠性测试与侠义的可靠性测试

广义的软件可靠性测试 是指 为了最终评价软件系统的可靠性而运用 建模、统计、试验、分析、和评价 等一系列手段对软件系统实施的一种测试。

侠义的软件可靠性测试 是指 为了获取可靠性数据，按预先确定的测试用例，在软件的预期使用环境中，对软件实施的一种测试。

也叫“软件可靠性试验（Software Reliability Test）”，它是面向缺陷的测试，以用户将要使用的方式来测试软件，所获得的测试数据与软件的实际运行数据比较接近。

可靠性测试是对软件产品的可靠性 进行调查、分析、评价 的一种手段。

对检测出来的 失效的分布、原因、后果 进行分析，并给出纠正建议。

总的来说，可靠性测试的目的可归纳为以下三个方面：

1、发现软件系统在 需求、设计、编码、测试、实施 等方面的 各种缺陷。

2、为软件的 使用、维护 提供可靠性数据。

3、确认软件是否达到可靠性的定量要求。

13.2 软件可靠性建模

13.2.1 影响软件可靠性的因素

软件可靠性模型（Software Reliability Model）是指 为预计或估算软件的可靠性 所建立的可靠性框图和数学模型。

模型 将复杂系统的可靠性 逐级分解为简单系统的可靠性，以便 定量预计、分配、估算、评价 复杂系统的可靠性。

影响软件可靠性的主要因素：缺陷的引入、发现、清除。

缺陷的引入 主要取决于 软件产品的 特征和软件的开发过程特性。

缺陷的发现依靠 运行剖面。

缺陷的清除 依赖于 失效的发现、修复活动、可靠性方面的投入。

影响软件可靠性的主要因素如下：

1、运行剖面（环境）。

2、软件规模。

3、软件内部结构。

4、软件的开发方法和开发环境。

5、软件的可靠性投入。人力、资金、资源、时间 等。

早期重视软件可靠性 并采取措施 开发出来的软件，可靠性有明显的提高。

13.2.2 软件可靠性建模方法

可靠性模型通常 由以下几部分组成：

1、模型假设。模型是实际情况的简化或规范化，总要包含若干假设。

2、性能度量。软件可靠性模型的输出量就是性能度量。

3、参数估计方法。

4、数据要求。

绝大多数模型包含三个共同假设：

1、代表性假设。选取代表软件实际的运行剖面。

2、独立性假设。假设认为软件失效是独立发生于不同时刻。

3、相同性假设。认为所有软件失效的后果（等级）相同，即建模过程只考虑软件失效的具体发生时刻，不区分软件的失效严重等级。

如果在进行预测时发现 引入了新的错误，或修复行为使新的故障不断发生，就应该停止预测。否则，这样的变化会因为增加问题的复杂程度而使模型的 适用性降低。

好的软件可靠性模型 应该具有如下 重要特性：

1、基于可靠性的假设。

2、简单。

3、计算一些有用的量。

4、给出未来失效行为的好的映射。

5、可广泛使用。

13.2.3 软件的可靠性模型分类

可靠性模型 大致可分为如下 10 类：

1、种子方法模型。

利用 捕获一再捕获抽样技术 估计程序中的错误数，在程序中预先有意“播种”一些设定错误的“种子”，然后 根据测试出的原始错误 和 发现的诱导错误比例，估计程序中残留的错误数。

优点是 简单易行，缺点是 诱导错误的“种子”与实际的原始错误之间的类比性估量困难。

2、失效率类模型。

3、曲线拟合类模型。

用回归分析的方法 研究软件 复杂性、缺陷数、失效率、失效间隔时间，包括 参数方法 和 非参数 方法两种。

4、可靠性增长模型。

5、程序结构分析模型。

通过对每一个节点 可靠性、节点间转换的可靠性 和 网络在节点间的转换概率，得出该持续程序的整体可靠性。

6、输入域分类模型。

7、执行路径分析方法模型。

8、非其次泊松过程模型。

NHPP，以软件测试过程中 单位时间的失效次数 为独立泊松随机变量，来预测 今后软件的某使用时间点的积累失效次数。

9、马儿可夫过程模型。

10、贝叶斯模型。

利用失效率的试验前分布 和 当前的测试失效信息，来评估软件的可靠性。

当软件可靠性工程师对软件的开发过程有充分的了解，软件的继承性比较好时 具有良效果的可靠性分析模型。

时间域。

失效数类：失效数是有限的还是无限的。

失效数分布。

有限类：用时间表示的失效强度的函数形式。

无限类：用经验期望失效数表示的失效强度的函数形式。

13.2.4 软件可靠性模型举例

1、模型假设

JM 模型的基本假设如下：

1. 初始错误个数为一个未知的常数。

2. 发现错误立即被完全排除，并且 不引入新的错误，排除时间忽略不记，因此 每次排错后 就要减 1.

3. 失效率 剩余的错误个数 成正比。

2、函数表达式。

略。

软件可靠性模型并不成熟，定量分析方法和数学模型要在实践中不断加以验证和修正。

不同类型的软件，应用方式也有很大区别。

13.2.5 软件可靠性测试概述

可靠性测试 由可靠性目标的确定、运行剖面的开发、测试用例的设计、测试实施、测试结果的分析 等主要活动组成。

软件可靠性测试 还必须考虑对软件开发进度和成本的影响，最好是在受控的自动测试环境下，由专业测试机构完成。

13.2.6 定义软件运行剖面

弧 用来连接状态并表示由各种激励导致的转换，将转换概率分配给每个 弧。

每类用户都可能以不同的方式使用系统。

两种类型分层形式：用户级分层、用法级分层。

用法级分层依赖于 在测试状态下 系统能做什么。

用户级分层 考虑 各种类型的用户，以及他们如何使用系统。

这些概率估计主要是基于如下几个方面：

1、从现有系统收集到的数据。

2、与用户的交谈或对用户进行观察 获得的信息。

3、原型使用与测试分析的结果。

4、相关领域专家的意见。

13.2.8 可靠性测试的实施

有必要检查软件需求与文档是否一致，检查软件开发过程中形成的文档的准确性、完整性、一致性。

可靠性测试依赖于软件的可测试性。

为了获得更多的可靠数据，应该使用多态计算机同时运行软件，以增加累计时间。

用时间定义的软件可靠性数据分为 4 类：

1、失效时间数据。

2、失效间隔时间数据。

3、分组时间内的失效数据。

4、分组时间的累计失效数。

这 4 类数据可以相互转化。

测试过程中必须真实地进行记录，每个测试记录必须包含如下信息：

1、测试时间。

2、含有测试用例的测试说明或标识。

3、所有与测试有关的测试结果，包括失效数据。

4、测试人员。

测试活动结束后要编写《软件可靠性测试报告》具备如下内容：

1、软件产品标识。

2、测试环境配置（硬件和软件）。

3、测试依据。

4、测试结果。

5、测试问题。

6、测试时间。

13.3 软件可靠性评价

13.3.1 软件可靠性评价概述

估计软件当前的可靠性，以确认是否可以终止测试并发布软件，还可以预计软件要达到相应的可靠性水平所需要的时间和工作量，确认软件的执行与需求的一致性。

13.3.2 怎样选择可靠性模型

可以从以下几个方面进行比较和选择：

1、模型假设的适用性。

2、预测的能力与质量。

3、模型输出值能否满足可靠性评价需求。

最重要的几个需要精确估计的可靠性定量指标 包括如下内容：

1. 当前的可靠度。

2. 平均失效时间。

3. 故障密度。

4. 期望达到规定可靠性目标的日期。

5. 达到规定的可靠性目标的成本要求。

4、模型使用的简便性

简便性一般包含如下三层含义：

1. 模型需要的数据 易于收集，成本不能超过可靠性计划的预算。

2. 模型应该简单易懂，测试人员不会花费太多的时间去研究专业的数学理论。

3. 模型应该便于使用。

13.3.3 可靠性数据的收集

面向缺陷的可靠性测试 产生的测试数据经过分析后，可以得到非常有价值的可靠性数据，这部分数据取决

于定义的运行剖面和选取的测试用例集。

可靠性数据的收集工作 是贯穿整个软件生命周期的。

可行的一些办法如下：

1、及早确定所采用的可靠性模型。

2、指定可实施性较强的可靠性数据收集计划，指定专人负责，按照统一的规范收集记录可靠性数据。

3、重视软件测试特别是可靠性测试产生的测试数据的整理和分析。

4、充分利用数据库来完成可靠性数据的存储和统计分析。

13.3.4 软件可靠性的评估和预测

1、判断是否达到了可靠性目标。

2、如未能达到，要再投入多少时间、多少人力、多少资金。

3、在软件系统投入实际运行 若干时间后，能否达到交付或部分交付用户使用的可靠性水平。

没有失效就无法估计可靠性。

要在模型之外运行一些统计技术和手段对可靠性数据进行分析，作为可靠性模型的补充、完善、修正。

辅助方法如下：

1、失效数据的图形分析方法。

1. 积累失效个数图形。

2. 单位时间段内的失效数的图形。

3. 失效间隔时间图形。

2、试探性数据分析技术（Exploratory Data Analysis，EDA）对可靠性分析有用的信息如下：

1. 循环相关。

2. 短期内失效数的急剧上升。

3. 失效数集中的时间段。

13.4 软件的可靠性设计与管理

13.4.1 软件可靠性设计

实践证明，保障软件可靠性 最有效、最经济、最重要的手段是 在软件设计阶段 采取措施进行可靠性控制。

1、软件可靠性设计是软件设计的一部分，必须在软件的总体设计框架中使用，并且不能与其他设计原则相冲突。

2、软件可靠性设计在满足提高软件质量要求的前提下，以提高和保障软件可靠性为最终目标。

3、软件可靠性设计应确定软件的可靠性目标，不能无限扩大化，排在功能度、用户需求、开发费用 之后考虑。

容错设计、检错设计、降低复杂度设计 等技术。

1、容错设计技术

1. 恢复块设计，一旦文本出现故障，用备份文本加以替换。

2. N 版本程序设计，对于相同初始条件和相同输入的操作结果，实行多数表决，防止其中某一软件 模块/版本 的故障提供错误的服务。

必须注意以下两方面：

使软件的需求说明具有完整性和精确性。

设计全过程的不相关性。

3. 冗余设计

在相同的运行环境中，一套软件出故障的地方，另外一套也一定会出现故障。

在一套完整的软件系统之外，设计一种不同路径、不同算法或不同实现方法的模块或系统作为备份。

费用可能接近单个版本软件开发费用的两倍，还有可能导致软件运行时所花费的存储空间、内存消耗、运行时间 有所增加，需要在可靠性要求和额外付出代价之间做出折中。

2、检错技术

检错技术实现的代价一般低于容错技术和冗余技术，但它有一个明显的缺点，就是不能自动解决故障。

着重考虑几个要素：检测对象、检测延时、实现方式、处理方式。

3、降低复杂度设计

模块复杂性主要包含模块内部数据流向和程序长度两个方面，结构复杂性用不同模块之间的关联程度表示。

软件复杂性是产生软件缺陷的重要根源。

在设计师就应该考虑降低软件的复杂性，是提高软件可靠性的有效方法。

在保证实现软件功能的基础上，简化软件结构，缩短程序代码长度，优化软件数据流向，降低软件复杂度，从而提高软件可靠性。

13.4.2 软件可靠性管理

为了进一步提高软件可靠性，又提出软件可靠性管理的概念，把软件可靠性活动贯穿于软件开发的全过程。

各个阶段的可靠性活动的 目标、计划、进度、任务、修正措施 等。

由于软件之间的差异较大，下面的每项活动并不是每一个软件系统的可靠性管理的必须内容，也不是软件可靠性管理的全部内容。

列表略。

第十四章 基于 ODP 的架构师实践

14.1 基于 ODP 的架构开发过程

系统架构 反映了功能在系统系统构件中的 分布、基础设施相关技术、架构设计模式 等，它包含了架构的原则 和 方法、构件关系 与 约束，并能支持 迭加或增量开发。

以软件架构为中心的开发过程是以 质量 和 风险 驱动的，最终提供一个稳定、低风险的 系统架构，并满足客户的需求（包含潜在需求）。

开放分布进程的参考模型（RM-ODP）是一个 ISO 标准，定义了分布系统的重要性质：

开放性、整体性、灵活性、可塑性、联合性、可操作管理性、优质服务、安全性、透明性。

RM-ODP 定义的 5 个观点：

1、企业视点：商业需求 和 策略、系统的范围 和 目的。可能会影响系统中的与企业相关的信息，如组织结构等。

2、信息视点。

3、计算视点。

4、工程视点。

5、技术视点。

每一个观点有具体的 建模目标 和 系统相关者。

分层子系统视图 提供了一个所有子系统高度抽象的 视图。

14.2 系统构想

14.2.1 系统构想的定义

系统构想 是 开发人员 与 用户 之间 共同的协议。

按照该协议，开发人员需要在特定的时间内完成系统用户的需求，系统构想必须 简短而切中要点。

高度概括了 业务架构的核心内容。

14.2.2 架构师的作用

系统构想 有助于 各方 明了系统的目标和范围。

确保系统开发的 计划、设计 等阶段 能依次有序地展开。

系统构想阶段，架构师合理的介入，有以下好处：

1、有利于使系统架构师本身对系统的看法更加全面、准确。

2、统一系统开发人员对系统的看法。

3、正确确定需求的优先次序。

4、最大程度上提高客户对设计等过程的参与程度，更好地与客户沟通。

14.2.3 系统构想面临的挑战

架构师对其控制能力之外的因素通常无能为力，可以通过有效地评估，以及高级经理和架构师之间保持紧密的联系 克服这些困难。

还必须面对以下几种情况：

1、很多架构师把架构看成是他们独自的创造，只要他们认为合适的就进行修改。

2、有些人不是拥有产品线构想的高级经理，却总是由这些人决定雇佣谁来做架构师。

14.3 需求分析

14.3.1 架构师的工作

需求 一般定义系统的 外部行为 和 外观 以及 用户信息，而 不用设计系统的 内部结构。

对需求分析 通常考察以下 6个方面的内容：

1、系统范围对象关系图。

2、用户接口原型，用户操作的一个雏形。

3、需求的适用性，该用什么技术解决，性能怎么样，是否与其他需求 相重合 或 矛盾，需求分析应注意 需求本身的实用或适用，而不必考虑其实现。

4、确定需求的优先级。

5、为需求建立功能结构模型，组件图，实体数据对象图。

6、使用质量功能分配（Quality Function Deploymen，QFD）发现隐藏质量需求，建立相关质量场景，先期预测需求风险。

有效地捕捉行为需求的方法是分析用例（Use Case）

用例包含 图 和 文字描述，符号 简单、抽象，保证 表述 需求时 简单性 和 清晰度。

14.3.2 需求分析的任务

1、需求分析的目的

完整、准确 地 描述用户对系统的需求，跟踪用户需求的变化，准确地 反映到系统架构和设计中，设计和用户的需求保持一致。

具有决策性、方向性、策略性 的作用。

2、需求分析的特点

追求系统需求的 完整性、一致性、验证性。

保持和用户要求同步，

保持需求分析各侧面之间的一致，

保持需求和系统设计间的同步。

14.3.3 需求文档与架构

每个用例都有一个相关需求的文字描述，定义用例应该和领域专家一起进行，如果没有领域专家的长期参与，只能是一种“伪分析”。

用例 为定义架构 提供了一个 系统的领域行为模型。

界面的外观、功能、导航 同用例紧密相连，有效定义屏幕的方法叫 低保真度原型（Low-fidelity Prototyping），

领域专家也始终参与到屏幕定义中去。

需求分析的 项目词汇表，也将在架构规划中被扩展。

14.4 系统架构设计

系统架构沟通了 需求和软件 之间巨大的 语义上 的 鸿沟。

系统架构的第一个任务就是 定义这两个极端之间的映射。

开放分布式处理（Open Distributed Processing，ODP）包括 企业、逻辑信息、计算接口、分布式工程、技术选择。

对每个观点，确认架构需求的一致性是非常重要的。

14.4.1 企业业务架构

企业业务架构 从 IT 角度，对企业的 业务结构、企业机构、业务的关系、内部的关系、与外部机构的关系进行整理定义。

包含如下内容：

1、企业的业务和战略目标，近期、中期、长远 目标。

2、企业的组织结构。

3、业务的分类。

4、各类业务之间的关系。

5、组织机构与业务的关系。

6、企业与外部机构的关系。

这些业务对象模型标识出系统的关键性约束，包括系统目标和重要的系统策略。

策略包含如下三类明确的表达方式：

责任：业务对象必须做什么。

许可：业务对象可以做什么。

禁止：业务对象不可以做什么。

对业务问题进行分析时，要考虑企业业务的发展，如新的服务或产品推出、考虑组织机构的改变 等。

所有这些可能的变化（易变场景）都应该提现在企业业务架构中。

通过对企业业务架构的定义，很清楚地知道由于企业业务特点、业务的流程特点、企业的组织机构 等 原因 对 IT 系统所带来的 自然分块 和 各个分块之间的边界关系。

企业业务架构的维护 是一个长期而反复的工作。

测试结果报告系统（Test Results Reporting System，TRRS）。

对象约束语言（Object Constraint Language，OCL）来定义企业活动者的这些策略（如 许可、禁止、义务 等）。

14.4.2 逻辑信息架构

逻辑信息架构（信息视点）标识出 系统必须知道什么。

强调定义系统状态的属性。

开放分布式处理是一种面向对象的方法，模型包含了关键信息的处理，如 传统的对象概念。

软件架构对象 并不是编程的 对象，它表示对系统的约束和依赖，这些约束能够消除把需求翻译成软件过程中的许多猜测性工作。

架构师应该 把他们的建模 集中于 有高风险、高复杂性、模糊性 的关键方面。

14.4.3 计算接口架构

计算接口对系统架构非常有帮助，但是它常常被架构师所忽略。

消除多个开发者和小组的主要设计争端，这些接口的架构控制对于一个支持变化和控制复杂性的稳定的系统结构来说，是非常重要的。

接口定义语言（IDL），完全独立于编程语言和操作系统。

14.4.4 分布式工程架构

分布式工程架构 定义了 底层结构的需求，独立于所选择的技术，解决了 最复杂的系统策略，包括 物理

位置、系统规模可变性、通信服务质量。

ODP 的一个最大好处是 关注点分离。

14.4.5 技术选择架构

大多数架构是独立的。

基于对候选者的初始选择，根据 产品价格、培训要求、维护风险 之类的项目因素 而反复进行。

14.5 实现模型

最终用户和架构师应在一起审查并贯穿于用例，始终 来证实需求的有效。

对产品设计的可行性做出准确地 评估、论证。

14.6 架构原型

架构原型是很好的需求验证工具，作为改进设计的手段，确保与工程约束相一致。

下面是一些架构师可以在架构原型中寻求解答的具体问题：

1、主要组件 是否得到了良好的定义？是否恰当？

2、主要组件间的协作 是否得到了良好的定义？

3、耦合是否得意最小化？

4、我们能否确定重用的潜在来源？

5、接口定义和各项约束是否可以接受？

6、每个模块 是否能访问到其所需要的数据？

经过 2 次或 3 次迭代之后，架构变得稳定。主要的抽象对象都已找到，子系统和过程都已经完成，所有的接口都已明确定义。

利用架构原型，几个好处：

1、落实之前，让团队成员能自由发表他们自己的看法。

2、统一团队之间的思想看法，提高系统开发的成功率。

3、对系统内部的结构分析与设计也有帮助。

14.7 项目规划

项目规划是通过批准的正式文档，以它为基准跟踪和控制项目，行动方案 和 资源分配，引导项目实施。

主要作用是 将指定规划的 假设 和 决定 批准的范围、成本、进度 的基线等 用正式的文档记录保存。

估算是项目规划的核心。

随着项目的进展，估算会不断校正并逐渐地接近实际。

项目管理者通过计划与规划的差异，不断优化和更新计划策略，使项目按规划的要求得以实现，计划的变更是可管理和可受控的。

规划包括：

1、项目的 目的、范围、目标、对象。

2、软件生存周期 的选择。

3、精选的 规程、方法、标准。

4、待开发的软件工作产品。

5、规模估计、软件项目的工作量和成本估计。

6、关键计算机资源的估计；项目的里程碑。

7、风险的识别和评估。

8、工程实施和支持工具计划。

软件项目计划的目标有：软件估计被文档化，活动和约定形成文档，受影响的组和个人 认同与软件项目规划的约定。

14.8 并行开发

14.8.1 软件并行开发的内容及意义

提高软件生产率，改善软件质量，有效地组织可以重复的资源。

并行开发研究的内容主要如下：

1、软件过程及其模型。

2、并行分成划分。

3、并行控制。

4、支持环境。

5、交互机制与集成技术。

14.8.2 并行开发的过程

把软件系统的开发过程划分为若干个 可以并行的成分，这个成分称之为 子开发过程。

子开发过程 = 开发小组 + 软件对象 + 对软件对象的开发活动。

并行开发活动，称为并行开发系统，实体是个开发小组，实体属性是被开发的软件对象，行为是开发软件对象的活动。

行为模块的 划分 是并行开发中的核心问题，模块独立性是 衡量软件设计质量的关键。

系统划分方法：

1、基于 Petri 网系统模型的动态划分方法。

2、基于脚本的系统划分方法。

软件过程并行控制是一个非常重要的问题。

就是要用正确的方式 调度并行操作，避免造成不一致性，使一个操作的执行不受其他系统的干扰。

保证 一致性、相容性、正确性、可靠性，手段有 加锁、时间戳、管程、Petri 网、PV 操作 等。

继承 和 测试 被分为两个阶段，如果不考虑硬件或软件的集成，两个阶段并没有明显的界限，所以，软件集成的主要问题是 集成测试技术。

14.9 系统转换

系统转换是指 运用某一种方式 由新的系统代替旧的系统的过程，也就是系统 设备、数据、人员 等方面的 转换。

14.9.1 系统转换的准备

转换前，必须认真做好准备。

还需测试 试运行 这项工作。

注意如下两个问题：

1、系统试运行工作的代表性。

2、系统试运行中错误的修正。

14.9.2 系统转换的方式

直接转换、平行转换、分段转换、分批转换。

14.9.3 系统转换的注意事项

1、大量的基础数据，录入工作量很大，应及早准备，尽快完成。

2、应提前做好人员的培训工作。

3、出现一些局部性的问题，应有 足够的准备，并做好记录。如果出现致命问题，要重新设计。

14.10 操作与维护

14.10.1 操作与维护的内容

数据管理与维护。

设备管理与维护。

软件的管理与维护工作。

14.10.2 系统维护与架构

系统架构的好坏，可维护性是一个重要方面，维护人员应参与架构的审评。

可维护性可以定性地定义为：维护人员 理解、改正、改动、改进 的难易程度。

可维护性有如下几个评价指标：

可理解性。

可测试性。

可修改性。

系统维护工作可以分为以下 4 种类型：

更正性维护。

适应性维护。

完善性维护。

预防性维护。

维护人员必须先理解要维护的系统，然后建立一个维护方案。

由于某处修改很可能会影响其他模块程序，所以考虑的重要问题 是 修改的影响范围 和 波及面 的大小。

必须强调的是，维护是对整个系统而言的，必须同时修改涉及的所有文档。

14.11 系统移植

14.11.1 系统移植的方式

不修改已有的软件。

修改软件。

重新编软件。

14.11.2 系统移植的工作阶段划分

计划阶段。

准备阶段，准备转换所需的材料。

转换阶段。

测试阶段。

验证阶段。

使系统移植工作标准化，工具实现自动化。

14.11.3 系统移植工具

系列化、标准化、文档化，使任何人都能以相同的顺序开展工作，提高效率。

第十五章 架构师的管理实践

软件架构师的主要障碍 往往在于组织方面 而非技术，技术上出色的架构往往由于 没有全面地处理好组织管理因素而失效。

15.1 VRAPS 组织管理原则

VRAPS 包括 构想、节奏、预见、协作、简化 5 个相关联的原则。

受益人 是指 建立并长期保持 架构的价值 有重要影响的人或组织。

1、构想原则：描述一副 一致的、有约束力和灵活的未来图景。

2、节奏原则：协调程度，根据可预测的 速度、内容、质量 对制品生产进行检查与规划。

3、预见原则。

4、协作原则：如何识别对架构成功关键的团体，如何确保这些合作伙伴的有效支持。

5、简化原则：理解组织的结构。

所有其他的原则 也是彼此之间相互影响的。

15.2 概念框架

用 准则、模式、反模式 对各项原则进行补充。

准则用于 判断每个原则的实施效果如何。

模式描述了 常见问题和解决方法。

反模式描述了 在实践中可能遇到的陷阱。描述了不该做的事情，或者用在错误背景下的解决方案。

15.3 形成并统一构想

构想描述了架构的未来，提供了 架构使用的环境和动机。

必须把它所能提供的价值 与客户的约束相对应。

明晰、有约束力、一致、灵活。

15.3.1 形成构想

构想需要维持 一致性与协调性。

一致性 是指 受益人的各种期望之间妥协，以及 需求满足程度。

灵活性 是指 不破坏架构的情况下，完成事先没有预料到的需求的容易程度。

RUP 的 “4+1 架构视图”体现了获得这种一致性的方法，逻辑视图、实现视图、进程视图、 部署视图、用例视图，它们根据不同目的表示系统。

架构师可以推荐技术，如何 以及 即使采用这些技术。

架构师更多地意味着 权衡业务、组织运作、使用技术，找出各利益方关注的重点，在一个公共的组织层次上对 信息、决策、资源 进行协调。

Thompson 归纳了 架构构想的三步方法：清楚明确地阐述一条迫切的客户价值；将客户价值映射为 少数特定的能解决的问题；将以上问题转译成一组特定的约束条件。

成功的架构师用明确的客户价值映射规划未来，架构师必须格外关注产品开发人员和最终的客户。

15.3.2 将构想原则付诸实践

用于检验构想原则是否起作用的准则如下：

1、构想 与 发起人、用户、最终客户 期望实现的目标 是否保持一致。

2、实施人员是否信任并使用架构。

3、潜藏知识 对其用户（开发团队）是否是 可见的、可获得的。

准则 1

为获得 一致、迫切、灵活 的架构，产品线经理、架构师、实施经理 等 达成共识。如果没有阐明用户价值，会导致构想脱离了重点。

反模式：风险后置。

用最小的妥协、最大的优化 规划出一个构件以满足所有冲突利益的需要。往往在理论上可行，但实际运行中出现风险。

模式：前后一致。

一个公共的架构 被几个产品共享，已经变得比预期要复杂得多，客户们针对每件产品又提出了以前 没有预计到的 功能特性。

需要评估架构构想的质量和稳定性，只有当两者都正常时，才能采取进一步行动，如果新特性不属于 原来构想的代价范围，就应该放弃。

如果这个特性确实属于一个文档的产品构想，应该在开发组织内 核实这种一致性。

准则 2

开发人员 可能会使架构向许多不同的方向发展。一个良好架构应与构想保持一致，同时又能满足用户的需求。

反模式：墙头草。

因为没有良好的构想，导致架构方向 在竞争和客户压力的影响下经常改变。这种构想 永远不能达到稳定以便有效地被共享。

以后发布 会因需要提供向后兼容而变得更为复杂。

方便变更需求 是成功构想的一部分，高级主管要与架构师一起紧密地工作，理解变更的后果并做出正确的权衡。

需要一种固定的机制。

模式：三个臭皮匠。

架构师并非总是架构构想的来源，共同的架构或平台 是产品线战略的关键。

抵制 创建 无所不能的架构 诱惑。

高级经理只提供 构想、目标、原则。

成功的产品线架构必须能为适应市场变化，能适应和采用新技术，能解决在概念阶段还不知道的但变化场景可预见的问题。

准则 3

反模式：一叶障目。

开发人员过份专注于 应用，以致不知道其他架构解决同类问题的通用解决方案。

需创造一种分享知识的愿望，如，工程师培训、启用知识管理平台。

模式：轮流工作。

帮教制（Apprenticeship）模式 阶段性地轮流交换构件的所有权 可以解决上面的问题。

组织和鼓励 构件的前任负责人 抽出时间帮助新的负责人，轮转周期应该尽可能地与发布进度保持同步。

15.4 节奏：保证节拍、过程、进展

15.4.1 节奏定义

节奏是 反复出现的、可预测的 工件交换活动。

节奏三个元素：速度、内容、质量。

节奏在团体和组织之间 与内部提供一种协调活动的 稳定力量，帮助移交管理。

节奏很强时，能培养很强的 预见、实施移交、交接 的技能。

15.4.2 将节奏原则付诸实践

以下准则出现时，才说明节奏起了作用。

1、经理们定期地 再评估、同步、调整架构。

2、架构用户对架构发布的进度和内容具有高度的信心。

3、通过节奏协调明确的活动。

准则 1：经理们定期地 再评估、同步、调整架构。

必须在稳定的 间隙上 再评估、同步、调整 他们的架构计划。

反模式：一步成功。

过于专注地向市场推出某项功能特性 而 导致内部节奏遭到破坏。

组织被竞争所蒙蔽，全身心地专注于向市场提供该特性，却削减了质量。

如果在实现关键特性的时候难以保持节奏，说明该特性的风险和复杂度比预计的要大，需要重新规划。

模式：发布委员会。

该模式向经理们介绍了 架构发布的最后阶段 再评估、同步、调整架构 的 方法。

在会议中，要复审产品 功能特性 和 优先级 的变更，从而使 产品文档、市场承诺、公共关系、测试、开发 保持一致。

与会人员应该有 足够的决定权。

准则 2：架构用户对架构发布的进度和内容具有高度的信心。

用户对架构发布的进度和内容缺乏信心是一个警告信号，说明没有设立一个良好的节奏。

反模式：超敏捷。

抄近路以维持稳定的发布节拍，对用户所期望的架构质量和内容进行妥协。

是否已经分配了足够的资源 来执行计划中的步骤，经理们是否创建明确的目标 来保证对节奏的维持都有非常重要的影响。

模式：舍兵保帅。

把不太重要的特性移到后面的发布周期。

准则 3：通过节奏协调明确的活动。

软件架构的受益人分布在许多不同的组织中。

反模式：销售未检验的产品。

由于团队不把编译和测试用例的失败当回事，导致积累下来的问题越来越多，无法按时发布。

对定期建立的流程进行修改，防止在修正失败的建立之前开展新的工作。

模式：同步发布。

如果架构中的一些变化需要互补产品做出重大变更，那么应让这些变化出现在最早的发布中。

15.5 预测、验证、调整

为了使对软件产品线的长期投资能产生回报，必须明确架构满足许多应用的需求。

能够预见变化并对变化做出反映，包括那些在设计架构时还没想到的需求。

能够适应新的 技术、标准、市场、竞争对手。

15.5.1 预测、验证、调度 的定义

竞争形式 运行环境 新的组织结构 像银行业这样 合并、接管 司空见惯的领域中，预测不可能总是正确的，所以需要验证。

在架构成型前 要对这些假定进行 检查和确认。

调整要求 具有敏捷性。

15.5.2 将预见原则付诸实践：准则、反模式、模式

1、不断增强架构的影响能力：预见到的风险和架构客户及其客户的需求；市场驱动的标准和演变的技术；战略性业务方向的改变。

2、通过快速复审和开发周期，评估技术和业务上的风险与机会。

3、当认识到关键字的估计或假设有错时，即使调整功能特性、预算。

准则 1：不断增强架构的影响能力：预见到的风险和架构客户及其客户的需求；市场驱动的标准和演变的技术；战略性业务方向的改变。

反模式：遗漏细节。

每个人都关注发布的强大新特性，以致忽视了一些用户必不可少的功能。

需要识别关键用户群，和他们一起找出最重要的需求。

模式：示范区。

挑选一个项目初步实现架构。该项目的客户渴望采用新技术，而且也愿意容忍获得该技术时可能存在的不便。

在架构大范围应用之前，缺陷将被发现并解决，修订后兼容问题的约束较小。

准则 2：通过快速复审和开发周期，评估技术和业务上的风险与机会。

反模式：品尝未成熟的果实。

被以前的换代或升级害苦了的客户 对关于未成熟技术的承诺极度不信任。

要审慎地选择引入新技术的正确场所，要为最初用户提供额外的支持。

不要假定一个未经验证的架构能够实现所有的承诺，应该分别在 开发人员和产品 用户的特定环境下测试你的解决方案。

模式：架构复审。

对开发中的架构组织执行一次有重点的专家评估，有重大影响的问题和机遇。

在开发周期的关键时刻成立一个架构复审委员会以检查架构。

需求基线初步确定首次复审，有经验的架构师、架构小组成员，客户。

注意让这些复审保持重点。

早起发现缺陷，及时修正，能发现取代新的开发活动的构件，还增强了客户对架构提供已承诺能力的信心。

准则 3：当认识到关键字的估计或假设有错时，即使调整功能特性、预算。

反模式：创造奇迹。

解决方法分为如下两个部分：

1. 找到架构的基础假设并积极努力测试这些假设。

2. 一旦发现错误的估计或假设，必须准备好对此采取行动。

无论何种情况，都应该确保把足够的资源编入预算计划，使得当不可避免的意外发生时，有可分配的进度和人员。

模式：外包。

合适及怎样选择一个已有的第三方构件，或者与提供者合作。

要确定潜在的合作伙伴是否把你需要的构件视为其主营业务的一部分。

他们必须为你做的专门开发越多，信任程度要求越高。类似地，信任度越低，你面临的进度和财政风险就就越高。

15.6 协作：建立合作型组织

每一个对架构关键的团体必须知道如何使用、努力改进架构从而为自己的利益服务。

如何识别对架构成功起关键作用的团体，如何确保这些合作伙伴的支持。

15.6.1 协作定义

协作是指架构受益人保持明确的、合作的角色并将所提供和获得的价值最大化的程度。

合作是指受益人彼此之间存在一些共享的预期，应该明确表示出达到或未达到预期会有哪些奖励和惩罚。

15.6.2 将协作原则付诸实践：准则、反模式、模式

正式定义的协作网络与非正式协作网络决定了一个软件架构能否成功。当出现以下几种情况时，说明协作是有效的。

1、架构师不断地努力了解谁是最关键的受益人，他们如何贡献价值，以及他们需要什么。

2、受益人之间达成明确和强制性的契约。

3、通过社会行为制度和非正式规范强化合作。

准则 1：架构师不断地努力了解谁是最关键的受益人，他们如何贡献价值，以及他们需要什么。

挑选一批集中的首要客户，找出保证他们参与需要做些什么，然后交付这些内容，这样做可以增大成功的机会。

反模式：光说不做。

架构师知道了用户的需求 却遗漏了 为了向他们提供有价值的的东西所应该做的事情。

架构师忙于其他事务，没有与开发人员进行稳定的交流，各产品团队按照自己的理解开发并升级了产品，放弃了原来同意的清晰的接口。

模式：了解你的受益人。

利用价值链来识别关键受益人，积极听取他们的一键并获得承诺与支持。

在初步阐明构想之后，确定潜在的合作伙伴以及他们的能力和利益如何与构想保持一致。

准则 2：受益人之间达成明确和强制性的契约。

反模式：不记录讨论结果。

不记录讨论结果说明了当一个架构团队回避采取必要的行动与其最直接的用户达成明确的契约时会发生什

么情况。

要确保取得对关键受益人的利益与职责的明确理解，当互动变得消极或者缺乏建设性时，可以求助于这些文件。

模式：互惠互利。

互惠互利要求在合作伙伴之间进行 公平、主动 的价值交换。

应该对正式和非正式的契约复审以保证公平的交换。预算中应该包括代码负责人响应其他团体请求所花的时间。

准则 3：通过社会行为制度和非正式规范强化合作。

反模式：非正式时间做正式工作。

让工程师利用业余时间修改，架构师就失去了控制其过程和结果的能力，甚至连测试也有可能被删除或完全忽略。

如果这种产品加入到其他团队的工件中，这位工程师在需要完成日常任务同时，还接到大量要求提供支持的请求，导致工程师精疲力竭。

要制订计划奖励工程师花在共享构件上的时间，尽早兑现奖励能减少工作量和大量压力。

把员工用于开发、维护被团体或项目外部所共享的解决方案的时间编入预算，预防工程师在利用非正式时间做正式工作。

模式：杜绝意外。

要尽早提醒用户注意变更，及时协商解决方案。

模式：和 HR 密切合作。

大部分高级技术岗位要求能迅速获得广泛的潜信息。有着广泛的非正式人际网的工程师比没有这种网络的工程师能获得质量更好的信息。

经理们应该避免破坏非正式人际网。

15.7 简化：澄清与最小化。

确定关键价值是不容易的，尤其是当新客户和新产品的加入使架构偏离原来的方向时，困难会显著增加。

简化软件架构的原则概念上看似简单，而实践中 它要求对价值非常坚定地专注，以及对架构所生存的组织理解和支持。

15.7.1 简化定义

在决定简化架构时，应当留意组织的结构；否则，你会发现你所做的改变只是暂时的。因此在简化架构之前，必须澄清组织和架构。

澄清组织意味着 真实地理解你计划部署架构于其中的组织结构及其影响力（Force）

架构对架构团队和客户都必须是清晰的，简化架构之前，必须准确地知道架构 被期望做什么、如何完成这些任务。

澄清架构就是提供用户所需要的细节。

如果一个组织具备简化、协作、节奏 等技能，长期共享架构就能最小化代码、文档、过程。

不好的组织情况下，共享可能导致架构膨胀。

15.7.2 将简化原则付诸实践：准则、反模式、模式

当以下准则都满足时，说明简化原则起作用了。

1、开发人员长期使用架构，减少了总成本和复杂性。

2、架构小组明确理解关键最小需求，并且将其构造成多个应用共享的核心元素。

3、通过长期的预算和行动确保相当关元素没有被共享、增加了不必要的复杂性时，或者是因为有明确的业务理由时，把相关元素从核心移走。

准则 1：开发人员长期使用架构，减少了总成本和复杂性。

反模式：简单复制并修改。

不与构件负责人协商变更 就复制并修改架构的部分代码，通常会带来深远的后果。

模式：由慢而快。

开发人员为了跟上进度，拒绝使用架构，解决方法是：放宽进度，加强过程。

指导开发人员逐步采用架构。

准则 2：架构小组明确理解关键最小需求，并且将其构造成多个应用共享的核心元素。

反模式：缺乏有效抽象。

由于没有被一个共享平台强力支持，每一个分离产品都要求自己的支撑结构。

通过框架团队来建立一个共享平台，防止产品专有特性进入平台。

模式：迁移途径。

准则 3：通过长期的预算和行动确保相当关元素没有被共享、增加了不必要的复杂性时，或者是因为有明确的业务理由时，把相关元素从核心移走。

把架构师拉去参加一个紧急项目以实现一个新特性，而使架构无人照看。

反模式：编码大于架构。

要防止架构师成为实现者，否则问题越来越多。

应该把首席架构师的时间合理分配给实现新特性和调整架构两个 renwu

模式：统计构件变更。

通过观察不稳定程度 来挑选需要调整的架构构件的方法。

重组（Refractor）通过长期观测 每个构件或子系统不稳定的程度，那些最不稳定的构件就是重组的候选者。

第十六章 层次式架构设计

16.1 体系结构设计

整个软件系统结构的设计与规格说明 比算法选择和计算问题的数据结构 更为重要。

因此，代码级别的软件复用已经远远不能满足大型软件开发的需求。

软件体系结构可定义为：为软件系统提供了 结构、行为、属性 的高级抽象，由构成系统的元素描述、这些元素的相互作用、指导元素集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，并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，是构建于软件系统之上的系统级复用。

软件体系结构贯穿于软件研发的整个生命周期，具有重要的影响，主要从以下三个方面进行考察：

1、利益相关人员之间的交流。

2、系统设计的前期决策。

3、可传递的系统级抽象。这种模型可以在多个系统之间传递，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。

分层设计是一种常见的架构设计方法，能有效地使 设计 简化，清晰，便于提高复用能力和产品维护能力。

16.2 表现层框架设计

16.2.1 使用 MVC 模式设计表现层

MVC 强制性地把 输入、处理、输出 流程 按照 视图、控制、模型 的方式进行分离，形成了 控制器、模型、视图 三个核心模块。

1、控制器（Conrtrtollert）：接受用户的输入 并调用模型和视图去完成用户的需求。是用户界面与 Model 的接口。

2、模型（Model）：业务数据和业务逻辑，为多个视图提供数据。

3、视图（View）：用户看到并与之交互的界面，不进行任何实际的业务处理，能接受模型发出的数据更新事件。

使用 MVC 模式来设计表现层，可以有以下优点：

1、允许多种用户界面的扩展，视图与模型没有必然的联系。

2、易于扩展。

3、功能强大的用户界面。

将业务处理与显示分离，增加了应用的可拓展性、强壮性、灵活性。

目前比较先进的 web 应用框架都是基于 MVC 设计模式的。

16.2.2 使用 XML 设计表现层，统一 Web Form 与 Windows Form 的外观

XML 标记用于定义数据本身的结构和数据类型，很少采用 XML 作为表现技术。

GUI 主要是由 GUI 空间组成，包含位置信息、类型、绑定事件 等。

可以被描述成一个 XML 节点，而控件的那些相关属性都可以描述成 这个 XML 节点的 Attribute。

由于 XML 本身就是一种树形结构描述语言，所以可以很好地支持控件之间的层次结构。

在调用显示 GUI 时，不是直接的调用特定的表现技术的 API，而是 装载 GUI 对应的 XML 配置文件，然后根据特定的 表现技术 的 解析器 解析 XML，得到 GUI 视图实例对象。

16.2.3 表现层中的 UIP 设计思想

UIP（User Interface Process Application Block）提供了一个扩展的框架，用于简化用户界面与商业逻辑代码的分离的方法。

将概念上的用户交互流程从实现或者涉及的设备上分离出来，保持内部的事务关联状态。

16.2.4 表现层动态生成设计思想

基于 XML 界面管理技术，包括 界面配置、界面动态生成、界面定制 三部分。

基于 XML 的界面管理技术 实现了用户界面描述信息与功能实现代码的分离，可针对不同用户需求进行界面配置和定制，只需对 XML 文件稍加修改，即可实现系统的移植。

16.3 中间层架构设计

16.3.1 业务逻辑层组件设计

业务逻辑组件分为 接口、实现类 两个部分。

通常按模块来设计业务逻辑组件，每个模块设计一个业务逻辑组件，控制器无需与具体的业务逻辑组件耦合，而是面向接口编程。

16.3.2 业务逻辑层工作流设计

工作流管理联盟（Workflow Management Coalition）将工作流定义为：

业务流程的全部或部分自动化，在此过程中，文档、信息、任务 按照一定的过程规则流转，实现组织成员间的协调工作以达到业务的整体目标。

1、Interface 1：过程定义 导入/导出 接口。转换格式 和 API 调用，从而支持过程定义信息间的互相转换。

2、Interface 2：客户端应用程序接口。通过这个接口工作流机可以与任务表处理器交换，代表用户资源来组织任务，然后由任务表处理器负责，从任务表中选择、推行任务项。

3、Interface 3：应用程序调用接口。允许工作流机直接激活一个应用程序，来执行一个活动。

4、Interface 4：工作流机协作接口。定义相关标准，以使不同开发商的工作流系统产品互相间能够进行无缝的任务项传递。

5、Interface 5：管理和监视接口。用户管理、角色管理、审查管理、资源控制、过程管理、过程状态处理器 等。

用工作流的思想组织业务逻辑，优点是：将应用逻辑与过程逻辑分离。

16.3.3 业务逻辑实体设计

业务逻辑层实体 提供对业务数据及相关功能的状态编程访问。

可以使用具有复杂架构的数据来创建，通常来自数据库的多个相关表。

业务逻辑层实体不直接访问数据库。

将业务逻辑层实体表示为通用 DataSet 的优点如下：

1、灵活性。

2、序列化。

3、数据绑定。

4、排序、过滤。

5、与 XML 的互换性。

6、开放式并发。

7、可扩展性。

将业务逻辑层实体表示为有类型的 DataSet，有类型的 DataSet 是包含具有严格类型的 方法、属性、类型定义 以公开 DataSet 中的数据和元数据的类。

将业务逻辑层表示为有类型的 DataSet 的优点如下：

1、代码易读。

2、IntelliSense 将可用。

3、编译时类型检查。

16.3.4 业务逻辑层框架

业务框架位于系统架构的中间层，是实现系统功能的核心组件。

采用容器的形式，便于系统功能的开发、代码重用、管理。

1、Domain Model 是领域层业务对象，它仅仅包含业务相关的属性。

2、Service 是业务过程实现的的组成部分，是应用程序的不同功能单元，通过在这些服务之间定义良好的接口和契约联系起来。

这种具有中立的接口定义（没有强制绑定到特定的实现上）的特征称为服务之间的松耦合。

3、Control 服务控制器，是服务之间的纽带。

16.4 数据访问层设计（持久层框架设计）

16.4.2 工厂模式在数据访问层应用

做到数据库无关，需要在实际开发过程中将这些数据库访问类再做一次封装，还可以减少操作数据库的步骤。

工厂模式定义了一个用于创建对象的接口，让子类决定实例化哪一个类。

前提是 编写程序时，没有用到特定数据库的特性。

16.4.3 ORM、Hibernate 与 CMP 2.0 设计思想

ORM（Object-Relation Mapping）在关系型数据库和对象之间作一个映射，不需要再去和复杂的 SQL 语句打交道。

ORM 框架 把数据库转变成了 我们熟悉的对象，只需要了解面向对象的开发就可以实现数据库应用程序的开发，不需要浪费时间在 SQL 上。

同时也减少了代码量，减少数据层出错机会。

通过 Cache 的实现，能够对性能进行调优。

16.4.4 灵活运用 Xml Schema

Xml Schema 用来描述 XML 文档合法结构、内容、限制。

逐步替代 DTDs，成为 XML 体系中正式的类型语言。

Xml Schema 是 Schema 组件的集合，这些组件分为三组：基本组件、组件、帮助组件。

基本组建包括简单类型定义、复杂类型定义、属性声明、元素声明。

组件包括属性组、完整性约束定义、模型组、符号声明。

帮助组件包括 注释、模型组、小品词、通配符、属性使用。

Xml Schema 提供了创建 XML 文档的 必要的框架，规范由如下三部分组成：

1、Xml Schema Part 0：Primer。

2、Xml Schema Part 1：Structures。

3、Xml Schema Part 2：Datatypes。

Xml Schema 支持继承。

16.4.5 事务处理设计

事务是现代数据库理论中的核心概念之一。

原子性（Atomicity）。

一致性（Consistency）。

隔离性（Isolation）。

持久性（Durability）。

事务要尽可能短的时间内完成。

16.4.6 连接对象管理设计

JDBC 的数据库应用开发中，数据库连接的管理是一个难点，因为它是决定该应用性能的一个重要因素。

资源池，解决资源频繁分配、释放 所造成的问题。

第一步就是要建立一个静态的连接池，所谓静态，是指池中的连接是在系统初始化时就分配好的，并且不能够随意关闭。

16.4.5 数据架构规划设计

XML 文档的存储方式有两种：基于文件的存储方式、数据库存储方式。

第十七章 企业集成架构设计

企业集成平台的核心是企业集成架构，包括 信息、过程、应用集成的架构。

17.1 企业集成平台

企业集成平台（Enterprise Integration Platform，EIP）目的是：

能够根据业务模型的变化 快速地进行信息系统的配置和调整，保证不同系统、应用、服务、操作人员 之间 顺畅地互操作，进而提高企业适应市场变化的能力，使企业能够在复杂多变的市场环境中生存。

良好的软件支持工具可以帮助企业加快实现企业系统集成，降低实现企业内部的信息孤岛集成的复杂度。

17.1.1 企业集成平台的概念

早起比较简单的集成方式是通过在不同的应用之间开发一对一的专用接口来实现应用之间的数据集成，即采用点到点的集成方式。

优点是比较直观。

问题：工作量大；集成费用高，系统升级和扩展困难；不易于标准化，接口数量过多，给系统管理造成比较大的困难。

为了克服点到点集成方式 给企业应用系统集成和维护管理带来的困难，采用集成平台。

企业集成平台是一个支持复杂信息环境下信息系统开发、集成、协同运行 的软件支撑环境。

可以使分散的信息系统通过一个单一的接口，可以管理、可重复的方式 实现单点集成，使企业内的所有应用都可以通过集成平台进行通信和数据交换。

集成平台 是支持企业集成的支持环境，包括 硬件、软件、软件工具、系统，基本功能主要如下：

1、通信服务

提供分布环境下透明的 同步/异步 通信服务功能。

2、信息集成服务

使集成平台上运行的 应用、服务、用户端 能够以一致的语义和接口实现对数据的访问控制。

3、应用集成服务

通过高层应用编程接口 来实现对应应用程序的访问，在无需对原有系统进行修改（不会影响原有系统的功能）的情况下，只要在原有系统的基础上加上相应的访问接口就可以将现有的、用不同的技术实现的系统互联起来。

4、二次开发工具

一组帮助用户开发特定应用程序的支持工具，其目的是简化用户在企业集成平台实施过程中的开发工作。

5、平台运行管理工具

集成平台的运行管理和控制模块。

17.1.2 集成平台的标准化

采用标准化的技术也是提高集成平台系统开放性和软件模块可重用性的重要方法。

标准化内容涉及通信协议、中间件、企业建模、工作流管理系统、Internet 环境下的数据交换、产品数据标准和应用系统集成标准 等。

17.1.3 实现技术的发展趋势

1、集成的技术实现从 2 层到 n 层过度

无论是在服务器端，还是在客户端，由于业务逻辑和应用表示逻辑的紧密捆绑，对系统的升级和扩展都带来了比较大的困难。

将业务过程逻辑、业务表示逻辑进行分离，将每层的功能集中在一个特定的角色上，提高集成平台和集成系统的柔性。

2、集成支持的方式从面向信息集成扩充到面向过程集成、服务集成

面向信息的集成主要应用于企业内的数据库和数据源上，其具体的实现方法主要有数据复制、数据捆绑、基于接口的信息集成 三种方式。

1. 面向过程的集成：通过工作流引擎对企业内部流程模型的执行来实现业务应用数据或信息在不同应用、子过程、执行任务的人员之间流动。

对业务过程逻辑和应用逻辑进行分离，实现过程建模和数据、功能 分离，保持具体功能单元不变的情况下，通过修改过程模型来改变系统功能。

2.面向服务的集成。

服务提供者将应用作为服务部署在 web 上，通过使用 web 服务描述语言来描述 web 服务提供的功能，并通过统一的服务发布与发现协议将其注册到 UDDI 中心。

只要在原有系统的基础上增加一个对它们进行访问的 SOAP 接口，就可以完成原有系统到集成平台的集成。

将以前主要在企业内部网络基础上实施的集成扩展到了面向开放网络环境下的集成，从而大大扩展了集成的范围。

具有良好的柔性和开放性，牺牲了性能和网络流量。

3、集成规范的标准化程度不断提高

从数据描述的角度来看，逐渐过渡到具有自描述功能的基于 XML 语言的数据表达与存储。

从应用间集成接口的实现与接口表现形式看，发展到更通用的基于 XML 语言的 web 服务几口定义语言（WSDL）的集成接口描述。

从业务过程定义方面来看，如何利用 web 服务集成架构实现过程集成的基于 XML 语言的商业流程模型描述语言。

4、所支持的集成耦合度及集成的粒度的变化

耦合度 不断降低，集成范围 不断扩大，集成粒度 不断缩小。

17.1.4 集成平台的发展趋势

从功能上可以将其划分为 企业应用集成 和 业务到业务的集成（B2B）两种。

其中，EAI 主要侧重于企业内部的纵向集成，B2B 侧重于支持企业间业务往来的横向集成。

17.2 企业集成平台的实现

17.2.1 数据集成

构建企业集成平台的首要目的是实现数据集成，提供具有 完整性、一致性、安全性 的数据访问、信息查

询、决策支持 服务。

具体包括：共享信息管理、共享模型管理、数据操作管理 三部分。

共享模型管理则提供数据资源配置管理、集成资源关系管理、资源运行生命周期管理及相应的业务数据协同监控管理等功能。

体系结构与标准化（规范化）程度对数据集成的水平有非常大的影响。

数据集成主要有以下三种模式：数据联邦、数据复制、基于接口的数据集成。

17.2.2 应用集成

功能之间的相互调用和互操作，需要在数据集成的基础上完成。

实现异构应用系统之间 语用层次上的互操作。

应用集成最初主要采用点对点的紧耦合方式，缺乏必要的柔性，组件化的系统实现及松耦合的应用集成方式逐渐成为构建企业业务处理系统的主流。

应用集成模式包括：集成适配器、集成信使、集成面板、集成代理 4 种。

1、适配器集成模式

通过适配器完成不同的系统间数据格式及访问方式的转换与映射。

2、信使集成模式

系统之间的通信和数据交换通过信使（消息代理）来实现，大大减少了接口数量，将应用之间的交互对通信服务能力的依赖程度降到最低。

3、面板集成模式

集成面板可以为 一对多、多对一、多对多 等 多种应用提供集成接口，为客户端应用与调用服务端应用提供了一种简化的公共接口。

4、代理集成模式

面板集成模式实现了服务器应用交互逻辑的分离。

在代理集成模式中，由于不存在很明显的客户端应用和服务端应用的划分，它仅需要将待集成的应用程序分离出来，并对应用间的交互逻辑进行封装，进而由集成代理来引导多个应用之间的交互。

17.2.3 企业集成

企业应用软件系统从功能逻辑上可以分为：表示、业务逻辑、数据 三个层次。

按照这些逻辑功能层次间是否分离和分离的程度，在软件系统具体实现上可以大致分为如下 4 类：

1、单层结构系统。

2、两层结构系统。

表示层与业务层（胖客户）紧密地耦合在一起，或者 将业务逻辑和数据库层紧密地耦合在一起（只将表示层分离出来为瘦客户）。

3、三层结构系统。

表示、业务逻辑、数据 三个层次分成独立的模块实现。

各层可以并行开发，各层也可以选择各自最适合的开发环境和编程语言。

4、n 层结构系统。

目的是提高系统不同业务功能模块的独立性，可以使系统具有最好的柔性及可扩展能力。

根据企业集成平台功能的支持范围，可以将其分为 侧重与支持企业内部集成化运行的 EAI 和 侧重于支持企业间业务集成的 B2B。一般来说，EAI 是 B2B 的基础。

从企业集成运行的实现策略上看，EAI 主要有如下三种实现模式：

1、前端集成模式

是指 EAI 侧重于业务应用系统表示层的集成，单一的用户入口实现多个应用事务的运作。

2、后端集成模式

主要侧重于应用系统数据层面的集成，就像一个方便多个应用系统之间数据自动交互的数据管道。

基于 EAI 服务器提供的存储——转发机制可以方便地实现对合作伙伴企业之间大量业务数据交换（主要指 B2B 集成）的支持。

3、混合集成模式

是前端集成模式和后端集成模式的组合，主要应用于既需要响应大量服务请求、又需要维护多个数据源的完整性和一致性的情况。

17.3 企业集成的关键应用技术

17.3.1 数据交换格式

企业业务数据可以分为 结构化数据（表单）和非结构化数据（文档），一般存储在不同的数据库或文档管理系统中。

不同的应用系统、数据库所处理的文档和数据格式有很大差别，建立各个应用都可以识别的访问通用数据模型及表示规范，是实现不同应用系统之间交互和互操作的最基本方法。

几种数据交换格式如下：

1、EDI（Electronic Data Interchange）电子数据交换

将 贸易、运输、保险、银行、海关 等行业的信息，用一种国际的公认的标准格式，通过计算机通信网络，供有关部门、公司、企业之间进行数据交换与处理，并完成以贸易为中心的全部业务过程。

目的是将功效上的纸质介质文件等同的电子表单统一的格式进行表示。

按照 UN/EDIFACT 标准，贸易伙伴之间 一次交换的内容称为一个交换，交换由 交换头/尾、功能组头/尾、报文头/尾、数据段（或断组）和数据元（简单数据元和复合数据元）等组成。

数据段（或段组）、数据元等在文本中都被称为报文项。

2、XML

XML 是国际组织 W3C 定制的一个 面向各类信息的数据存储工具和可配置载体的开放式标准。

目的是为了更好地适应 web 应用的需求，解决 HTML 在表达能力、扩展性、交互性 等方面的缺陷。

具有愈发清晰简单、结构无歧义等优点。

利用一套定义标记的规则将文件的内容和外观进行分离，实现了 XML 文档的可延伸性及自我描述特性。

它本身并不是一种标记语言，而是一种创建、设计、使用 标记语言的根规则集，是一种创建标记语言（如 HTML）的元语言。

3、STEP 标准（Standard for the Exchange of Product Model Data）是一种描述如何表达和交换数字化产品信息的 ISO 标准（ISO010303）

目的是提供一种不依赖具体系统的中性模型，并将其用来描述整个生命周期内的产品数据。

4、PDML 是在 STEP 和 XML 基础上实现不同系统间产品数据交换和集成的一种新模式。

PDML 中主要应用了 STEP 的集成资源和 Express 数据规范语言两个部分。

与特定领域词汇表（或数据字典）相应的组件被称为应用事物集（Application Service Set，ATS），与跨多个

应用领域的通用词汇表相应的组件被称为集成方案。

PDML 由 7 个应用事物集、一个集成大纲、应用事物集和集成大纲间的映射规范、PDML 工具集 4 部分组成。

17.3.2 分布式应用集成基础框架

大规模计算机网络的重要特性——异构性。

在面向对象技术和分布式计算基础上产生的 分布式对象计算（Distributed Object Computing，DOC）。

比较有影响的分布式软件对象（组件）标准有下面 4 种：

1、CORBA（Common Object Request Broker Architecture，公共对象请求代理体系结构）解决分布式处理环境中硬件和软件系统的互联而提出的一种标准的面向对象应用程序体系规范。

对象管理参考模型（Object Management Architecture，OMA）把软件作为对象，并通过对象请求代理与其他对象进行通信。

核心是对象请求代理（Object Request Broker，ORB），它支持对象服务、通用设施、领域接口、应用接口 之间的交互和通信。

服务完成后又把执行结果或异常情况返回给请求者。

2、COM+

COM 组件标准的基础是 COM 核心，规定了组件对象与客户通过二进制接口标准进行交互的原则。

COM 主要由 COM 接口、COM 对象、COM 服务器、类工厂和类型库等组成。

每个接口有一个唯一标识（UUID），对 COM 对象的调用是通过一个指向其接口的指针实现的。

客户对组建对象功能的调用接口一般采用 COM IDL 来描述。

两类服务器，进程内部服务器、进程外部服务器。

进程内服务器即本机上的 DLL，进程外服务器分为两类：一是本机上的 exe 可执行程序，而是远程机上的 dll 或 exe 程序。

进程外的对象必须先调用服务控制机制提供的代理，代理生成服务对象的远程过程调用（Remote Process Call，RPC）。

另外，COM 组件标准还包括结构化存储、统一数据传输和智能命名等。

3、J2EE

J2EE 很好地融合了 Internet 技术，有利于企业建立基于 web、具有 n 层结构的分布式应用。

J2EE 的基础是核心 Java 平台或 Java2 平台的标准版，J2EE 将 J2SE 集成到自己的体系结构中。

各种组件可以通过 J2EE 配置工具将其部署到相应的 J2EE 容器中，客户端对各种组件的访问及各种组件之间的调用都通过容器及服务器来完成。

4、Web Service

Web Service（Web 服务）是指服务提供者将应用作为服务部署在 web 上，通过使用 web 服务描述语言来描述特定 web 服务提供的功能。

web 服务可以看成是现有应用面向 Internet 的一个延伸。

目前支持 web 服务的技术标准主要有：

用于进行数据交换和表达的元语言标准 XML，

UDDI（Universal Description，Discovery & Integration），UDDI 用于 web 服务注册和服务查找，

WSDL 用于描述 web 服务的接口和操作功能，

SOAP（Simple Object Access Protocol）为建立 web 服务和服务请求之间的通信提供支持。

17.4 面向整体解决方案的企业模型

17.4.1 企业模型在整体解决方案中的作用

1、企业模型可以为信息化整体解决方案提供 对企业公共一致的、规范的表达和描述。

2、建模和基于模型的分析 是企业信息化工作的入手点和建立有效的实施途径的基础。

3、建模可以对信息系统规划方案进行预评估。

4、基于模型的工作流执行可以导航和监控各信息系统之间及信息系统外界的交互。

17.4.2 整体解决方案中的企业模型重用

不同的企业虽然在生产经营诸多方面都有其特殊性，但是他们都是企业系统的实例，具有企业最本质的行为和特征。

可以将构成企业所有要素（无论是物质实体还是抽象过程）分成三类：

一类是最通用的，适用于任何企业。

二类是在一定范围内通用。

三类是某个企业专有的。

通用层、部分通用层、专用层。

企业模型可以采用从零开始的方法建立，周长和建模质量低。

因此，基于参考模型建立企业具体的专用模型是较好的方法。

包括两个阶段：参考模型的选择、参考模型的实例化。

参考模型的选择具体包括以下几个步骤：

1、确定企业建模的目的和基本需求。

2、划定企业建模的范围。

3、提出候选参考模型。

4、确定最终使用的参考模型。

实例化过程在具体操作中可以采用的方法如下：

1、继承。

2、裁剪。

3、细化。

4、扩充。

5、修改。

17.4.3 整体解决方案中企业模型演化

信息系统实施的生命周期可以分为 需求分析阶段、系统设计阶段、系统实施阶段、运行维护阶段。

1、需求分析阶段

通过对用户需求的抽象形成需求分析模型，以作为下一个阶段的输入。

需求分析模型应该包含有较高层次上的企业业务流程、资源分配、组织结构和产品结构等信息。

最后还需要确定系统的总体目标和评价标准。

2、系统设计阶段

从未来的信息系统相关的业务模型中抽取出功能模型和信息模型，用它们来设计和构造信息系统。

3、系统实施阶段

实现了企业模型从设计模型向可执行模型的转化。

通过定义具体的操作者、执行器、资源实体、组织单元、应用软件 等。

4、运行维护阶段

通过文档管理、版本控制等方法实现对系统的有效管理和监控，并通过集成需求管理软件工具来对运行过程中企业不断提出的新需求进行记录和管理，所积累的需求和文档是下一个生命周期的输入。

一个系统实施后在运行维护阶段搜集的问题和需求又会启动一个新的生命周期。

这个不断循环的生命周期以螺旋式上升的形式实现企业相关状态及行为的改进与发展。

17.4.4 模型驱动的企业集成系统演化

由于这种实施是根据企业当前的市场策略、业务过程规划和当前的信息技术现状进行的，它只能够在当前的企业和市场状态下，通过信息技术支持企业实现其竞争优势。

第十八章 面向方面的编程

AOP（Aspect Oriented Programing）面向方面的编程（有人翻译为：**面向切面的编程**）。

18.1 方面编程的概念

18.1.1 AOP 产生的背景

1、面向过程编程面临的问题

面向过程编程是一种自顶向下的编程方法，其实质是对软件进行功能性分解。

2、传统面向对象编程面临的问题

对象模型可以很好地映射到实际领域。

完成某个特定需求的代码分散到各个类中，很难把它们全部找到，这就给程序的健壮性带来了隐患。

将传统的按功能或按对象划分程序模块的方法 转化为按系统特征划分程序模块，这就是 AOP 的基本思想。

18.1.2 面向方面的编程

某些操作较难实现模块化，称涉及到这些操作的代码是分散的。

方面的定义：一个设计来用于捕捉应用程序横切面功能的程序单位。

方面与类的不同在于它实现了横切程序的功能。

程序中包括类和方面意味着模块性可以在两个因素上实现：类实现基本的功能性（这个因素叫做结构性）；

方面实现横切的功能性（这个因素叫做可操作性）。

方面由两个部分组成：切入点和通知代码。

通知代码包括要执行的代码，切入点定义了程序重要执行的代码处的点。

一个服务可以在一个程序中是非功能性的，但在另一个程序中却是功能性的。

18.1.3 AOP 技术

AOP 可以说是 OOP（Object-Oriented Programming）面向对象编程 的补充和完善。

它利用一种称为“横切”的技术，剖开封装的对象内部，并将那些影响了多个类的公共行为封装到一个可重用模块，并将其命名为 Aspect，即方面。

实现 AOP 采用动态代理技术/采用静态织入的方式。

18.1.4 AOP 特性

改善了软件的 可扩展性、重用性、易理解性、易维护性。

通过将实现基本功能的组件和特定于应用的系统特性分离，使得组件（包括类或者函数）的重用性得到提高。

第十九章 嵌入式系统设计

19.1 嵌入式系统

19.1.1 嵌入式系统的概念

以应用为中心，以计算机技术为基础，可以适应不同应用对功能、可靠性、成本、体积、功耗 等方面的要求，集可配置可裁剪的软件、硬件 于一体的专用计算机系统。

存储方案的选择就是在嵌入式 Linux 系统的可靠性、尺寸、功能、成本之间寻求最佳的平衡点。

19.1.3 嵌入式操作系统

嵌入式操作系统主要由应用程序接口、设备驱动和操作系统内核等几个方面组成。

嵌入式操作系统是一个按时序方式调度执行、管理系统资源并为应用代码提供服务的基础软件。

每个嵌入式操作系统都有一个内核，大多数内核都包含以下三个公共部件：调度器、内核对象、内核服务。

大多数内核支持两种普遍的调度算法：基于优先级的抢占调度、时间轮转调度算法。

19.1.5 嵌入式数据库管理

嵌入式数据库也称为移动数据库或嵌入式移动数据库，主要是解决移动计算机环境下数据的管理问题，移动数据库是移动计算机环境中的分布式数据库。

实际应用中必须解决好数据的一致性（复制性）、高效的事务处理和数据的安全性等问题。

19.1.6 嵌入式网络及其他

现场总线 主要有 总线型、星型 两种拓扑结构。

家庭信息网的拓扑结构有总线型、星型等。

常见的无线网络标准以 IEEE 802.11x 系列 为主。

19.2 嵌入式系统的设计

19.2.1 嵌入式系统分析与设计

嵌入式系统的核心技术有三种：处理器技术、IC 技术、设计/验证技术。

单用途处理器是设计用于执行特定程序的数字电路，也指协处理器、加速器、外设等。

项目计划、可行性分析、需求分析、概要设计、详细设计、程序建立、下载、调试、固化、测试、运行。

19.2.2 嵌入式软件设计模型

1、状态机模型

有限状态机（FSM）是一种描述系统状态及其状态转换的节点网，包括节点和边，节点表示状态，边表示状态之间的转换关系。

缺乏并发和层次化支持。

2、数据流模型

数据流图允许系统作为操作网进行建模，特别适合于对实现进行分区的系统模型。

布尔数据流、层次化流图、Petri 网。

3、并发进程模型

并发进程包括 CSP 与 CCS 等。

CSP 模型将输入、输出操作列为程序语言的基本要素，而将实现顺序进程间通信的并行组合作为基本的程序控制结构。

一个程序，就是一组进程，它们通过一个通信网络彼此通信。

面向对象的基本结构可用 6 个术语来描述，对象、类、属性、消息、操作、关系。

19.2.3 嵌入式系统软件开发环境

交叉平台开发方法（Cross Platform Development），即软件在一个通用平台上开发，而在另一个嵌入式目标平台上运行。

通用平台通常叫做宿主机系统，被开发的嵌入式系统称为目标机系统。

执行环境和开发环境一致时的开发过程称为 本地开发（Native Delelopment）。

第二十章 面向服务的架构

服务是一个由服务提供者提供的，用于满足使用者请求的业务单元。

在 SOA 中，服务的概念有了延伸，泛指系统对外提供的功能集。

20.1 SOA 的相关概念

20.1.1 SOA 的定义

面向服务的体系结构（Service-Oriented Architecture，SOA）

从应用的角度定义：是一种应用框架，着眼于日常的业务应用，并将它们划分为单独的业务功能和流程，即所谓的服务。

从软件的基本原理定义：是一个组件模型，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。

20.1.2 业务流程 与 BPEL

由于业务流程来源于现实世界，传统上是通过复杂的语言进行描述。

BPEL（Bussess Process Execution Language For Web Services）面向 Web 服务的业务流程执行语言。

提供了一种相对简单易懂的方法，可将多个 Web 服务组合到一个新的服务服务（称作业务流程）中。

将现有的 Web Services 按照要求的业务流程整理成为一个新的 Web Services，形成一个从外界看来和单个 Service 一样的 Service。

20.2 SOA 的发展历史

为了能够将公司的业务打包成独立的、具有很强伸缩性的基于因特网的服务，人们提出了 Web 服务的概念，

这可以说是 SOA 的起源。

1、萌芽阶段

以 XML 技术为标志，XML 的出现无疑为 SOA 的兴起奠定了稳固的基石。

2、标准化阶段

2000 年以后，出现了三个著名的 Web 服务标准和规范：简单对象访问协议（Simple Object Access Protocal，SOAP）、Web 服务描述语言（Web Services Descriptin Language，WSDL）、通用服务发现和集成协议（Universal Discovery Description and Integration，UDDI）。

Web 服务也是因特网 Web 2.0 时代的一项重要特征。

3、成熟阶段

从 2005 年开始，体现在三个重量级规范上：SCA/SDO/WS-Policy，标志着 SOA 进入了实施阶段。

美国实现 SOA 架构的关键任务是：对已有系统中的功能进行提取和包装，形成标准的“服务”。

大量“服务”需要全新构造才是中国 SOA 的主要任务。

20.3 SOA 的参考架构

以服务为中心的企业集成采用“关注分离（Separation of Concern）”的方法规划企业集成中的各种架构元素。

从服务为中心的角度来看，企业集成的架构 分为 6 大类：

1、业务逻辑服务（Business Logic Service）

2、控制服务（Control Service）

3、连接服务（Connectivity Service）

4、业务创新和优化服务（Business Innovation and Optimization Service）

5、开发服务（Development Service）

6、IT 服务管理（IT Service Management）

1、连接服务——企业服务总线（Enterprise Service Bus，ESB）

采用了“总线”这样一种模式 来管理和简化 应用之间的集成拓扑结构。

2、业务逻辑服务

1. 整合已有应用——应用和信息访问服务

2. 整合新开发的应用——业务应用服务

3. 整合客户和业务伙伴（B2C/B2B）——伙伴服务

3、控制服务

1. 数据整合——信息服务

2. 流程整合——流程服务

3. 用户访问整合——交互服务

4、开发支持

企业集成的开发工具需要有标准的工具框架，这些工具能够以即插即用方式支持来自多家厂商的开发工具，支持整个软件开发周期，以提高开发过程中各种角色的生产力。

5、业务创新和优化

业务创新和优化服务以业务性能管理（Business Process Management，BPM）技术为核心提供业务事件发布、收集、关键业务指标监控能力。

业务创新和优化服务与开发服务是紧密相连的。

6、管理支持

20.4 SOA 主要技术和标准

Web 服务作为实现 SOA 中服务的最主要手段。

20.4.1 UDDI 协议

UDDI（统一描述、发现和集成协议）计划是一个广泛的、开放的行业计划，它使得商业实体能够彼此发现；定义它们怎样在 Internet 上互相作用，并在一个全球的注册体系架构中共享信息。

UDDI 规范利用了 W3C 和 Internet 工程任务组织的很多标准作为其实现基础，如 XML、HTTP、DNS 等协议。

20.4.2 WSDL 规范

WSDL（Web Services Description Language，Web 服务描述语言），用来描述 Web 服务 和 说明如何与 Web 服务通信的 XML 语言。

通过 WSDL，可描述的三个基本属性：

1、服务做些什么

2、如何访问服务

3、服务位于何处

WSDL 文档被分为两种类型：服务接口（Service Interface）和服务实现（Service Implementations）

20.4.3 SOAP 协议

SOAP 是在分散或分布式的环境中 交换信息的简单的协议，是一个基于 XML 的协议。

20.5 SOA 的特性

20.5.1 文档标准化

20.5.2 通信协议标准

20.5.3 应用程序统一登记与集成

20.5.4 服务品质（Quality of Service，QoS）

关键元素有：安全需求（例如认证和授权）、可靠通信。

1、可靠性

仅且仅仅传送一次（Once-and-only-once Delivery）

最多传送一次（At-most-once Delivery）

重复消息过滤（Duplicate Message Elimination）

保证消息传送（Guaranteed Message Delivery）

2、安全性

认证交换、消息完整性、消息保密性，它借助现有的安全标准。

3、策略

服务提供者有时候会要求服务消费者与某种策略通信。

这些要求被定义为策略断言（Policy Assertions），一项策略可能会包含多个断言。

4、控制

整合应用意味着 像 异步通信、并行处理、数据转换，以及校正等进程请求必须被标准化。

5、管理

随着企业服务端增长，所使用的服务和业务进程的数量也随之增加，一个用来让系统管理员管理所有，运行在多种环境下的服务的管理系统就显得尤为重要。

20.6 SOA 的作用

在一个企业内部，可能存在不同的应用系统，由于开发的时间不同，开发工具不同，这些已有应用系统是孤立的，也就是我们常说的“信息孤岛”。

从头建立一个新的基础环境是不可能的。

SOA 凭借其松耦合的特性，使得企业可以按照模块化的方式来添加新服务或更新现有服务，以解决新的业务需要。

在 SOA 得以普及之前，解决企业内部信息系统“信息孤岛”的问题通常采用 EAI（企业应用整合）的方式。

EAI Server 就好像一个“翻译”，让每两个应用之间可以对话，会带来 EAI Server 呈几何倍数的增长。

SOA 对实现企业资源共享，打破“信息孤岛”的步骤如下：

1、把应用和资源转换成服务。

2、把这些服务变成标准的服务，形成资源的共享。

SOA 不仅仅是 一个技术，而是一个软件架构。

20.7 SOA 设计原则

SOA 架构中，继承了来自对象和组件设计的各种原则，如 封装、自我包含等。保证服务的灵活性、松散耦合、重用能力的设计原则。

常见和讨论的设计原则如下：

1、无状态。以避免服务请求着依赖与服务提供者的状态。

2、单一实例。避免功能冗余。

3、明确定义的接口。服务的接口由 WSDL 定义，用于指明服务的公共接口与其内部专用实现之间的界限。XML 模式（Schema）用于定义所交换的消息格式（即服务的公共数据）。服务定义必须长时间稳定，一旦发布，不能随意更改，服务的定义应尽可能明确，减少使用者的不适当使用。

4、自动包含和模块化。封装了那些业务上稳定、重复出现的活动和组件，实现服务的功能实体是完全独立自主的。

5、粗粒度。服务数量不应该太大。

6、服务之间的松耦合性。其位置、实现技术和当前状态等对使用者是不可见的，服务私有数据对服务使用者是不可见的。

7、重用能力。

8、互操作性、兼容和策略声明。

20.8 SOA 的设计模式

20.8.1 服务注册表模式

服务注册表（Service Registry）主要在 SOA 设计时段使用，虽然它们常常也具有运行时段的功能。

注册表支持驱动 SOA 治理的服务合同、策略、元数据 的开发、发布、管理。

因此，他们提供一个主控制点，或者称为策略执行点（Policy Enforcement Point，PEP）。在这个点上，服务可以在 SOA 中注册和发现。

注册表可以包括有关服务和相关软件组件的配置、遵从性和约束配置文件。任何帮助注册、发现和检索服务合同、元数据和策略的信息库、数据库、目录或其他节点都可以被认为是一个注册表。

主要的服务注册厂商：

一个阵营是提供服务、策略、元数据 注册表 及 信息库的纯 SOA 厂商。

另一个阵营是 SOA 平台厂商，这些厂商将注册表作为集成产品套件的一个组件。

1、注册服务：应用开发者，也叫服务提供者，向注册表公布他们的功能。

2、服务位置：也就是服务应用开发者，帮助他们查询注册服务，寻找符合自身要求的服务。

3、服务绑定：服务的消费者利用检索到的服务合同来开发代码，开发代码将与注册的服务绑定、调用注册的服务以及与他们实现互动。

20.8.2 企业服务总线模式

采用点对点的集成方式存在着复杂度高，可管理性差，复用度差，系统脆弱 等问题。

企业服务总线（Enterprise Service Bus，ESB）提供一种标准的软件底层架构，各种程序组件能够以服务单元的方式“插入”到该平台上运行，并且组件之间能够以标准的消息通信方式来进行交互。

支持异构环境中的服务以基于消息和事件驱动模式的交互，并且具有适当的服务质量和可管理性。

本质上是以中间件形式支持服务单元之间进行交互的软件平台。

这种交互过程不再是点对点的直接交互模式，而是由事件驱动的消息交互模式。

ESB 最大限度上解耦了组件之间的依赖关系，降低了软件系统互联的复杂性。

ESB 以中间件的方式，提供服务容错、负载均衡、QoS 保障和可管理功能。

ESB 的核心功能如下：

1、提供位置透明性的消息路由和寻址服务。

2、提供服务注册和命名的管理功能。

3、支持多种消息传递泛型（如 响应/请求、发布/订阅 等）。

4、支持多种可以广泛使用的传输协议。

5、支持多种数据格式及其相互转换。

6、提供日志和监控功能。

由于采用了基于标准的互联技术，ESP 使得企业内部以及外部系统之间可以很容易地进行异步或同步交互。具有很强的灵活性和扩展性。

服务总线层为整个 EAI 应用环境提供底层支持。

20.9 构建 SOA 架构时应该注意的问题

20.9.1 原有系统架构中的集成需求

一定要注意对原有系统架构中集成需求进行细致的分析和整理。

可以在系统的开发和维护中缩短产品上市时间，因而可以降低企业系统开发的成本和风险。

因此，当 SOA 架构师遇到一个十分复杂的企业系统时，首先考虑的应该是如何重用已有的投资而不是替换遗留系统。

当 SOA 架构师分析原有系统中的集成需求时，不应该只限定为基于组件构建的已有应用程序的集成，必须考虑一些更加具体的集成的类型，主要包括：

应用程序集成的需求；终端用户界面集成的需求；流程集成的需求；已有系统信息集成的需求。

因而，SOA 架构师在着手设计新的体系结构框架时，必须要全面地考虑所有可能的集成需求。

20.9.2 服务粒度的控制以及无状态服务的设计

构建一个企业级的 SOA 系统架构时，关于系统中最重要的元素，首先是：服务粒度的控制；其次是：无

状态服务的设计。

1、服务粒度的控制

通常来说，对于将暴露在整个系统外部的服务推荐使用粗粒度的接口，而相对较细粒度的服务接口通常用于企业系统架构的内部。

粗粒度服务接口保证了服务请求着将以一致的方式使用系统中所暴露出的服务。

2、无状态服务的设计

SOA 系统架构中的具体服务应该都是独立的、自包含的请求，在实现这些服务的时候不需要前一个请求的状态，也就是说服务不应该依赖于其他服务的上下文和状态。

当负载增加时，可以向群集中增加机器。（无状态就可以热插拔）

20.10 SOA 实施的过程

20.10.1 选择 SOA 解决方案

选择最佳的解决方案，是保证 SOA 实施成功的前提条件。

总体来说，必须从以下三个方面进行选择：

1、尽量选择能进行全面规划的方案

既需要选择适当的工具，还需要有专业的技术人才。

2、选择时充分考虑企业自身的需求

必须认识到，用于构建 SOA 项目的前期投资将产生巨大效益。

3、从 平台、实施 等技术方面进行考察

首先要考虑的是平台的开放性和对标准的支持。

以往的成功经验总结有 6 个方面：业务战略和流程、基础架构、构件模块、项目和应用、成本和效益、规划和管理。

20.10.2 业务流程分析

1、建立服务模型

1. 自顶向下分解法

自上而下的领域分解方式从业务着手进行分析，选择端到端的业务流程进行逐层分解至业务活动，并对其间涉及的业务活动和业务对象进行变化分析。

2. 业务目标分析法

通过关键性能指标分析来验证已有业务候选者以及发现遗漏的服务候选者，这可以叫做“目标服务建模（Goal Service Modeling）”。

它的思想是：从企业的业务目标触发，目标分解为子目标，子目标再分派给相关的服务来实现，这样就形成了一颗“目标服务树”。

3. 自底向上分析法

利用已有资产来实现服务，已有资产包括已有系统、套装或定制应用、行业规范或业务模型等。

这也叫做“遗留资产分析”，可以方便地发现哪些在不同的系统中被重复实现的功能模块以及可以复用的功能模块。

2、建立业务流程

1. 建立 业务对象（Business Object，BO）是对数据进行检索和处理的组件，是简单的真实世界的软件抽象。通常位于中间层或业务逻辑层。

业务对象的分类如下：

a. 实体业务对象。表达了一个人、地点、事物或者概念。

b. 过程业务对象。

c. 事件业务对象。

2. 建立服务接口

服务之间的交换可以为有状态、也可以为无状态。

在构建 SOA 的过程中，将无状态接口视为最好的选择，可以方便地供很多服务者应用程序重用。

3. 建立业务流程

流程是指定的活动顺序，包含明确确定的用于提供业务值的输入和输出。

对流程进行建模应当确保捕获的相关信息的一致性及完整性。

第二十一章 案例研究

21.1 价值驱动的体系结构：连接产品策略与体系结构

系统的存在是为了为利益相关方创造价值。

价值模型、体系结构策略。

定义完善的价值模型可以为提高折中方案的质量提供指导。

21.1.1 价值模型概述

这些利益相关者在其他系统中扮演着重要角色。

这些其他系统也是为了为其利益相关者创造价值。

系统的这种递归特性是分析和了解价值流的一个关键。

价格模型核心的特征 三种基本形式：

1、价值期望值：表示对某一特定功能的需求，内容（功能）、满意度（质量）。

2、反作用力：系统部署实际环境中，实现某种价值期望值的难度。

3、变革催化剂：环境中导致价值期望值发生变化的某种事件，或者是 导致不同结果的 限制因素。

反作用力和变革催化剂称为限制因素，把这三个统称为价值驱动因素。

传统方法，都是通过聚焦于系统 进行交互的参与者的类型开始的。

有如下几个突出的局限性：

1、对参与者的行为模型关注较多，而对其中目标关注较少。

2、往往将参与者固定化分成几种角色，其中每个角色所在的个体在本质上都是相同的。

3、往往忽略限制因素之间的差别。

4、结果简单。要求得到满足或未得到满足，用例成功完成或未成功完成。

这种方法 使用顺序推理和分类逻辑，因此易于教授和讲解，并能生成一组易于验证的结果。

效用曲线用于将每一个可选方案所得出的定量测量值映射到其对应值。然后，值级别用期望优先级加权，并进行叠加。

叠加值越高，方案越可取，该方法可能比较主观。

21.1.2 体系结构挑战

体系结构挑战 是因为一个或多个限制因素使得满足一个或多个期望值变得更困难。

在任何环境中，识别体系结构挑战都涉及评估。

1、哪些限制因素影响一个或多个期望值？

2、如果知道了影响，它们满足期望值更容易（积极影响）还是更困难（消极影响）？

3、影响程度如何？简单的 低、中、高 三个等级通常就已经够用了。

制订系统的体系结构策略始于：

1、识别合适的价值背景并对其进行优先化。

2、在每一背景中定义效用曲线和优先化期望值。

3、识别和分析每一背景中的反作用力和变革催化剂。

4、检测限制因素使满足期望值变难的领域。

建议对以下几点进行权衡：

重要性。

程度。

后果：大概多少种方案可供选择？难度或有效性是否有很大差异？

隔离：对最现实的方案的隔离情况如何？影响越广，该因素的重要性就越高。

尽管体系结构样式和模式技术非常有用，不过在该领域，在问题和解决方案领域的身后，经验仍具有无法估量的价值。

21.1.3 结论

价值模型 有助于了解和传达关于价值来源的重要信息。

如价值如何流动，期望值和外部因素存在的相似性和区别，要实现这些价值有哪些子集。

21.2 使用 RUP 和 UML 开发联邦企业体系结构框架

联邦企业体系结构框架（Federal Enterprise Architecture Framework，FEAF）

21.2.1 联邦企业体系结构框架概述

它把企业体系结构划分为 4 部分：业务、数据、应用程序、技术。

FEAF 确定了开发和维护联邦企业体系结构所需的 8 中构件。这八种构件的分解进一步细化了 FEAF 的 4 个层次。

第一层是联邦企业体系结构框架的最高层，它引入了开发和维护联邦企业体系结构所需要的 8 种构件。

1、体系结构推动者（Architecture Drivers）

2、战略方向（Strategic Direction）

3、当前体系结构（Current Architecture）

4、目标体系结构（Target Architecture）

5、过程转换（Transitional Processes）

6、体系结构片段（Architectural Segments）

7、体系结构模型（Architectural Models）

8、标准（Standards）

第二层在更详细的层次上说明了联邦企业体系结构的业务和设计方面以及两者之间的关联。

业务体系结构和设计体系结构之间的关系是 推/拉 关系——业务推动设计以满足自身的需要，设计（既新开发的数据、应用程序和技术）通过支持业务运作来拉动业务到新的服务交付水平。

第三层展开了框架设计部分，显示三种设计体系结构：数据、应用程序、技术。

第四层（最详细视图）三种设计体系结构如何支持业务体系结构开始逐渐明确起来。

21.2.2 FEAF 矩阵概述

36 个单元的矩阵，涵盖了企业中的 谁（who）、什么（what）、何处（where）、何时（when）、为何（why）、如何（how）。

21.3 Web 服务在 HL7 上的应用——Web 服务 基础实现框架

Health Level Seven（HL7）是美国国家标准化协会（ANSI）认可的标准化开发组织中的一个，它正在全世界保健行业里运行着（Level Seven 引用了开放系统互联模型 OSI 的最高层——应用层）。

21.3.1 HL7 模型概念

1、参考信息模型

2、消息结构

所有的 HL7 消息都被放在 Transmission Wrapper，Wrapper 的目的是 支持应用软件之间消息的传输（和确认）。

3、交互

4、应用程序角色

一个角色就体现了应用程序的职责。

5、Storyboard

一个 Storyboard 是由一小段记叙了它本身的目的及交互作用图表的描述所组成的（在应用层）应用程序角色间相互作用的级数。

21.3.3 开发 HL7 Web 服务适配器

为了高效地开发 HL7 Web 服务适配器，需要按如下步骤来做。

1、消息和数据类型的设计。

2、适配器模式的选择。

3、HL7 Web 服务契约开发。

4、生产 Web 服务 Stub 和 代理的实现。

5、开发适配器业务逻辑。