# 系统架构设计师
# 考试笔记

# 目录


一. 系统规划 7
1.1 项目的提出与选择 7
1.2 可行性研究与效益分析 7
1.3 方案的制订和改进 7
1.4 新旧系统分析和比较 7
二. 系统分析与设计 8
2.1 定义问题与归结模型 8
2.2需求 8
2.2.1 需求工程 8
2.2.2 需求分析 8
2.3 系统设计(软件设计) 8
2.4 结构化分析与设计 9
2.4.1 分析步骤 9
2.4.2 分析工具 9
2.4.3 结构化设计 9
2.4.4 数据流图、流程图 10
2.5 面向对象分析与设计 11
2.5.1 基本概念 11
2.5.2 UML(统一建模语言) 11
a 组成 11
b 软件架构(4+1视图模型), (非UML特有) 12
c 类间关系表示 12
d 各种图 13
2.5.3 用例模型 17
2.5.4 分析模型 17
2.5.5 设计模型 17
2.6 用户界面设计 17
2.7 工作流设计 18
2.8 简单分布式计算机应用系统设计 18
2.8.1 分布式系统开发 18
2.8.2 分布式计算架构 18
2.9 系统运行环境的集成与设计 18

2.10 系统过渡设计 19
三. 软件架构设计 19
3.1 概述 19
3.2 软件质量属性 19
3.2.1 ISO质量属性 20
3.2.2 优化方法 22
3.3 软件架构风格 22
3.4 分层C/S架构 25
3.4 面向服务的架构SOA 27
3.4.1 WEB SERVICE 28
3.4.2 企业服务总线 ESB 29
3.4.3 微服务 29
3.5 架构设计 30
3.6 软件架构文档化 30
3.7 软件架构评估 31
3.7.1 关键步骤 31
3.7.2 评估方法 31
3.8 构件复用 32
3.8.1 商用构件标准规范 32
3.8.2 应用系统簇与构件系统 33
3.8.3 基于复用开发的组织结构 33
3.9 产品线及系统演化 33
3.9.1 特定领域软件架构(DSSA) 33
3.9.2 过程模型 34
3.9.2 组织结构 35
3.9.3 建立方式 35
3.10 软件架构视图 35
四 设计模式 36
4.1 设计原则 36
4.2 设计模式概念 36
4.3 分类 36
四. 数据库系统 41
4.1 数据库管理系统的类型 41

4.2 数据库模式与范式 41
4.2.1 数据库模式 41
4.2.2 数据模型 42
4.2.3 关系代数 42
4.2.4 数据规范化 42
4.2.5 反规范化 42
4.3数据库设计 42
4.4 数据库备份与恢复 43
4.4.1 备份 43
4.4.2 恢复 43
4.5 事务管理 43
4.6 分布式数据库系统 44
杂 45
五. 计算机网络 46
5.1 网络互联模型与协议 46
5.2 网络工程 47
a 网络规划 47
b 网络设计 47
c 网络实施 48
5.3 网络存储技术 48
5.4 综合布线 49
六 系统性能评价 50
6.1 主要性能指标 50
6.2 性能设计 51
八 开发方法 51
8.1软件生命周期 51
8.2系统开发方法 51
8.3软件开发模型 51
8.4 几个大概念 53
8.6基于架构的软件设计ABSD 54
十 系统测试评审 54
10.1 测试阶段 54
10.1.1单元测试(模块测试) 54

10.1.2集成测试 54
10.1.3系统测试 54
10.2 测试类型 55
10.3 软件调试 55
10.4 维护 56
十一 嵌入式系统设计 56
11.1 硬件 56
11.2 软件 57
a 嵌入式操作系统 57
b 应用支撑软件 57
c 应用软件 57
11.3 开发与调试 58
11.4 嵌入数据库系统 58
11.5 实时嵌入式操作系统 59
十二 开发管理 59
12.1 范围管理 60
12.2 成本管理 60
12.3 时间管理 60
12.4 配置管理 60
12.5 需求管理 60
12.6 质量管理 60
12.7 风险管理 61
12.8 软件过程改进 61
12.9 人力资源管理 61
12.10 软件工具及开发环境 61
12.10.1 软件工具 61
12.10.2 软件开发环境 61
十三 信息系统基础知识 62
13.1 信息系统分类 62
13.2 信息系统战略规划 62
13.3 信息化需求 62
13.4 政务信息化 63
13.5 企业信息化 63

13.3.1 企业资源规划ERP 63
13.3.2 客户关系管理CRM 64
13.3.3 企业应用集成EAI 64
13.3.4 电子数据交换EDI 65
13.3.5 企业门户 65
13.3.6 电子商务 65
13.3.7 供应链 66
13.6 商业智能 66
13.7 其它 66
十四 安全性和保密设计 67
14.1 对称、非对称、数字签名、数字信封 67
14.2 密钥管理 67
14.2.1 密钥分配中心KDC（Key Distribution Center） 67
14.2.2 数字证书（如X.509） 67
14.2.3 公开密钥基础设施PKI (public key infrastructure) 67
14.3 网络安全体系 67
a 网络各层次安全保障 67
b 网络攻击类型 68
c 其它 69
14.4 身份认证和访问控制 70
14.4.1 身份认证 70
14.4.2 访问控制 70
14.5 系统安全性设计 70
14.5.1 物理安全 70
14.5.2 防火墙 70
14.5.3 入侵检测 71
14.6 几个概念总结 72
十五 系统可靠性 74
15.1 基本概念 74
15.2 系统容错 74
15.2.1 容错技术分类 74
15.2.2 单机容错 75
15.2.3 双机容错 75

15.2.4 服务器集群 75
15.2.5 软件容错 75
15.2.6 检错(自检)技术 76
15.3 备份与恢复 76
15.3.1 备份 76
15.3.2 恢复 76
十六 软件的知识产权保护 77
16.1 保护年限 77
16.2 知识产权人确定 77
16.3 侵权判定 78
16.4 标准化 79
十七 基于中间件的开发 79
17.1 相关概念 79
17.2 J2EE 80
17.3 .NET 82
17.4 MVC 82
十八 新技术 83
18.1 云计算 83
18.2 物联网 83
18.3 虚拟化 83
E其它 83
1 集群/分布式/负载均衡 83
2 Internet服务三种类型 85
3 管理距离 85
4 看门狗watchDog 85
二十 案例分析 85
二十一 论文 85
二十二 英语 87

# 一. 系统规划

## 1.1 项目的提出与选择

该步骤生成”产品/项目建议书”.

## 1.2 可行性研究与效益分析

包括经济可行性/技术可行性/法律可行性/执行可行性/方案选择 5 个部分. 该步骤生成”可行性研究报告”.

## 1.3 方案的制订和改进

包括确定软件架构/确定关键性要素?/确定计算体系(NET,J2EE).

## 1.4 新旧系统分析和比较

根据技术水平和商业价值进行评价, 演化策略有 4 种: 改造/集成/继承/淘汰

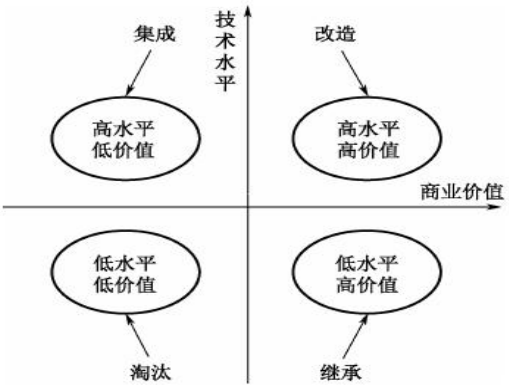



图 7-4 评价结果分析

移植工作大体上分为计划阶段、准备阶段、转换阶段、测试阶段、验证阶段。

1、计划阶段，在计划阶段，要进行现有系统的调查整理，从移植技术、系统内容（是否进行系统提炼等）、系统运行三个方面，探讨如何转换成新系统，决定移植方法，确立移植工作体制及移植日程。

2、准备阶段，在准备阶段要进行移植方面的研究，准备转换所需的资料。该阶段的作业质量将对以后的生产效率产生很大的影响。

3、转换阶段，这一阶段是将程序设计和数据转换成新机器能根据需要工作的阶段。提高转换工作的精度，减轻下一阶段的测试负担是提高移植工作效率的基本内容。

4、测试阶段，这一阶段是进行程序单元、工作单元测试的阶段。在本阶段要核实程序能否在新系统中准确地工作。所以，当有不能准确工作的程序时，就要回到转换阶段重新工作。

5、验证阶段，这是测试完的程序使新系统工作，最后核实系统，准备正式运行的阶段。

# 二. 系统分析与设计

## 2.1 定义问题与归结模型

## 2.2 需求

### 2.2.1 需求工程

需求工程包括需求管理和需求开发。

**需求开发**：包括需求捕获、需求分析、编写规格说明书和需求验证 4 个阶段。需求开发是努力更清晰、更明确地掌握客户对系统的需求。

**需求管理**：通常包括定义需求基线、处理需求变更、需求跟踪等方面的工作。而需求管理则是对需求的变化进行管理的过程。

### 2.2.2 需求分析

- 包括内容

  1 问题识别：用于发现需求、描述需求。预先估计以后系统可能达到的目标。

  2 分析与综合：对问题进行分析，然后在此基础上整合出解决方案。

  3 编制需求分析文档：对已经确定的需求进行文档化描述，该文档通常称为“需求规格说明书”。

  4 需求分析与评审：对功能的正确性、完整性和清晰性，以及其他需求给予评价。

- 需求分类 1

  功能需求

  非功能需求

  设计约束

- 需求分类 2

  业务需求

  用户需求

  系统需求：包括 功能需求/质量属性/非功能需求/设计约束

- 需求获取方式：121

  收集资料

  联合需求计划：召集多部门开会

  用户访谈：结构化，准备好了问题；非结构化，随机应变问问题. 明确访谈目的/ 确定访谈对象/ 明确访谈问题；

  书面(问卷)调查/情节串联板/现场观摩/参加业务实践/阅读历史文档/ 抽样调查

## 2.3 系统设计(软件设计)

步骤：概要设计(高层设计)/详细设计(低层设计)

设计方法：结构化设计/面向对象设计

## 2.4 结构化分析与设计

### 2.4.1 分析步骤

研究物质环境/建立系统逻辑模型/划清人机界限

### 2.4.2 分析工具

**数据流图**/数据字典/结构化语言/判定表/判定树

一个数据流图里面有数据流、加工、外部实体、数据存储.因此在描述完数据流图之后,可紧接着描述数据字典(数据流名称、来源、去向、结构、组织等)和处理(加工).

### 2.4.3 结构化设计

概要设计----模块结构图/层次图/HIPO 图(层次+input process output)

详细设计----**程序流程图**/盒图 NS/PAD 问题分析图/PDL 程序设计语言(伪码)

模块(化)---- 指执行某一特定任务的数据结构和程序代码

外部特征: 模块的接口和功能/

内部特征: 模块的局部数据和程序代码

关键原则: 信息隐蔽/模块独立性(高内聚、低耦合)

| 内聚类型 | 描　述 |
|---|---|
| 功能内聚 | 完成一个单一功能，各个部分协同工作，缺一不可 |
| 顺序内聚 | 处理元素相关，而且必须顺序执行 |
| 通信内聚 | 所有处理元素集中在一个数据结构的区域上 |
| 过程内聚 | 处理元素相关，而且必须按特定的次序执行 |
| 瞬时内聚(时间内聚) | 所包含的任务必须在同一时间间隔内执行 |
| 逻辑内聚 | 完成逻辑上相关的一组任务 |
| 偶然内聚(巧合内聚) | 完成一组没有关系或松散关系的任务 |

| 耦合类型 | 描　述 |
|---|---|
| 非直接耦合 | 两个模块之间没有直接关系，它们之间的联系完全是通过主模块的控制和调用来实现的 |
| 数据耦合 | 一组模块借助参数表传递简单数据 |
| 标记耦合 | 一组模块通过参数表传递记录信息(数据结构) |
| 控制耦合 | 模块之间传递的信息中包含用于控制模块内部逻辑的信息 |
| 外部耦合 | 一组模块都访问同一全局简单变量，而且不是通过参数表传递该全局变量的信息 |
| 公共耦合 | 多个模块都访问同一个公共数据环境 |
| 内容耦合 | 一个模块直接访问另一个模块的内部数据；一个模块不通过正常入口转到另一个模块的内部；两个模块有一部分程序代码重叠；一个模块有多个入口 |

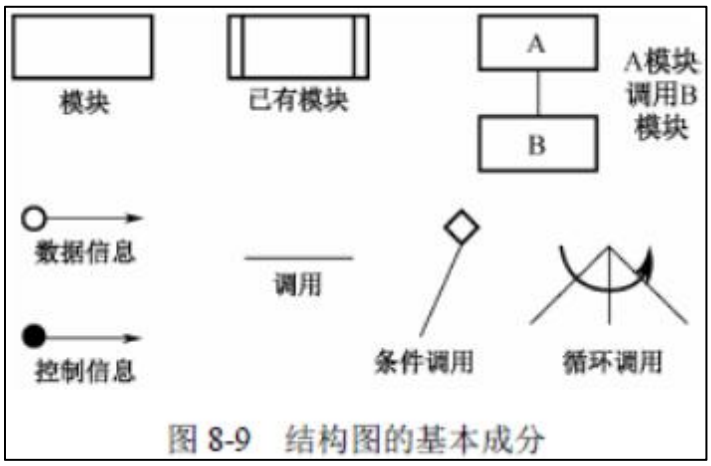


图 8-9　结构图的基本成分

### 2.4.4 数据流图、流程图

数据流图:作为一种图形化工具,用来说明业务处理过程、系统边界内所包含的功能和系统中的数据流。(1 Context 图 / 2 DFD 0 层图)

设计原则:

- 1 父图与子图的平衡:子图的输入输出数据流同父图相应加工的输入输出数据流一致。
- 2 数据守恒: 指加工的输入、输出数据流是否匹配，即每一个加工既有输入数据流又有输出数据流。
- 3 守恒加工: 对于同一个加工，其输入与输出的名字必须不同。
- ~~4 加工细节隐蔽: 父图内部细节留给子图去画。~~
- ~~5 简化加工之间的关系: 尽量减少加工间输入/输出数据流的数目.~~

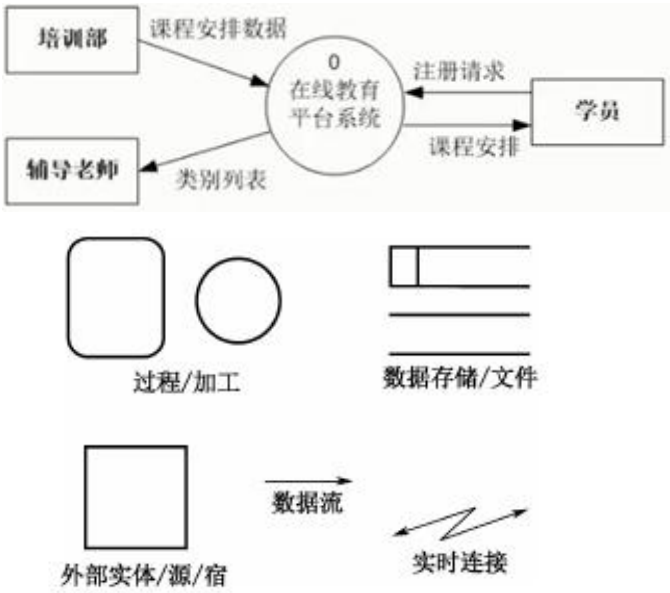


图 8-5　数据流图的符号集

流程图:以图形化的方式展示应用程序从数据输入开始到获得输出为止的逻辑过程,描述处理过程的控制流。

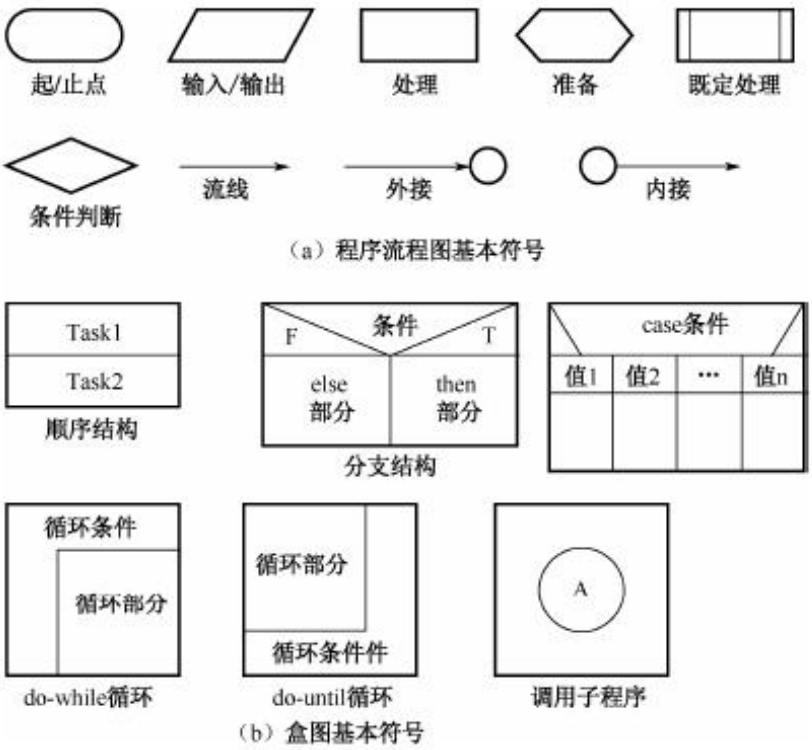



图 8-10　程序流程图和盒图基本符号示意图

| 区别 | 数据流图 | 流程图 |
|---|---|---|
| 并行性 | 处理过程可并行 | 某个时间点只能处于一个处理过程 |
| 展示流 | 展示数据流 | 展示控制流 |
| 计时标准 | 展示全局的处理过程,过程之间遵循不同计时标准 | 遵循一致的计时标准 |
| 用途 | 系统分析中的逻辑建模 | 系统设计中的物理建模 |

## 2.5 面向对象分析与设计

### 2.5.1 基本概念

类/对象；继承/泛化；多态/重载

### 2.5.2 UML(统一建模语言)

**a 组成**

- 构造块

  事物(建模元素)---结构事物/行为事物/分组事物/注释事物

  关系---关联/依赖/泛化/实现 http://c.biancheng.net/view/1319.html

  图（如下)---(结构)静态模型(类图、对象图、包图、构件图、部署图、制品图)/(行为)动态模型（用例图、顺序图、通信图、定时图、状态图、活动图、交互概览图）

- 公共机制

  规格说明/修饰/公共分类/扩展机制

- 规则

**b 软件架构(4+1 视图模型)，(非 UML 特有)**

速记：“逻进部实用”，”逻进物开场”

**逻辑视图：**以问题域的语汇组成的类和对象集合。(功能)

**进程视图：**可执行线程和进程作为活动类的建模，它是逻辑视图的一次执行实例。(运行)．主要关注一些非功能性需求，例如，系统的性能和可用性等。进程视图强调并发性、分布性、系统集成性和容错能力。

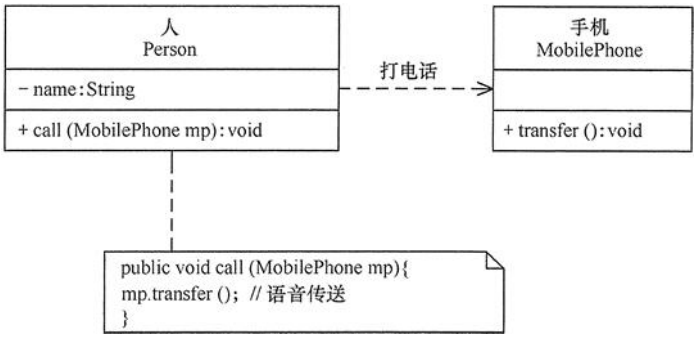
**实现(开发)视图：**对组成基于系统的物理代码的文件和组件进行建模。(模块组织管理)

**部署(物理)视图：**把组件物理地部署到一组物理的、可计算的节点上。

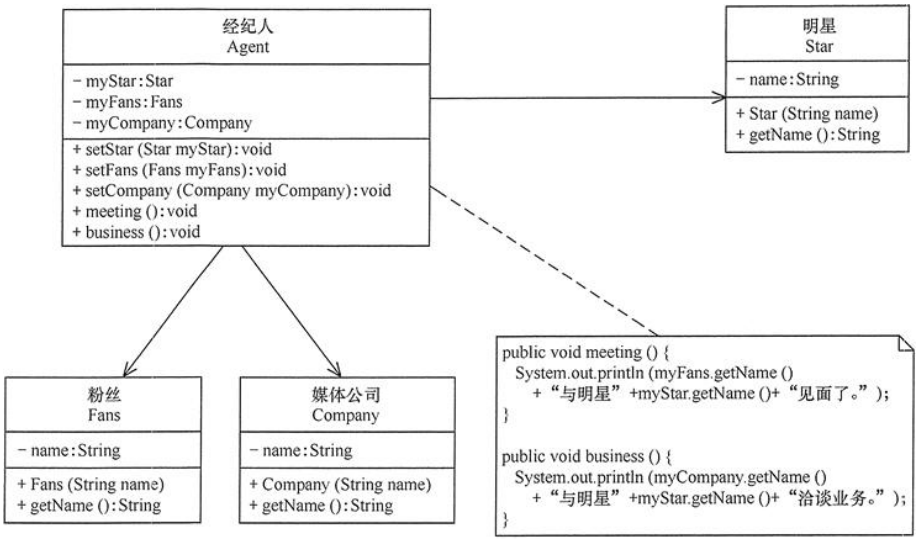
**用例(场景)视图：**最基本的需求分析模型.

**c 类间关系表示**

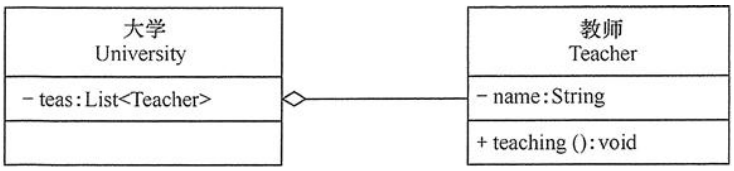
- 依赖：某个类的方法通过局部变量、方法的参数或者对静态方法的调用来访问另一个类（被依赖类）



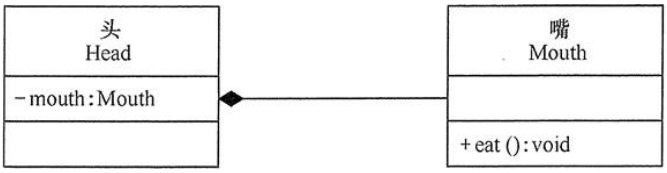

- 关联：(包括一般关联关系、聚合关系和组合关系)．将一个类的对象作为另一个类的成员变量来实现关联关系．关联可以是双向的，也可以是单向的。



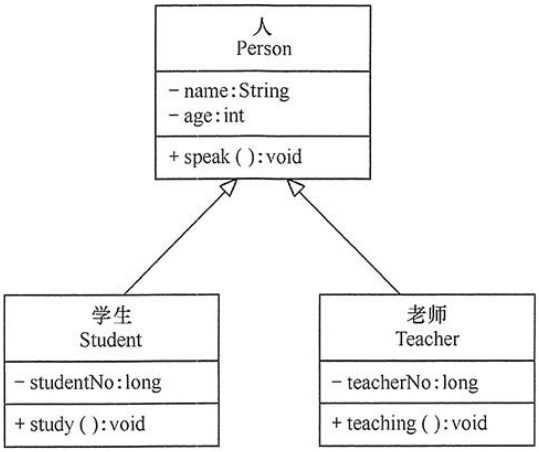

- 聚合：关联关系的一种．通过成员对象来实现的，其中成员对象是整体对象的一部分，但是成员对象可以脱离整体对象而独立存在



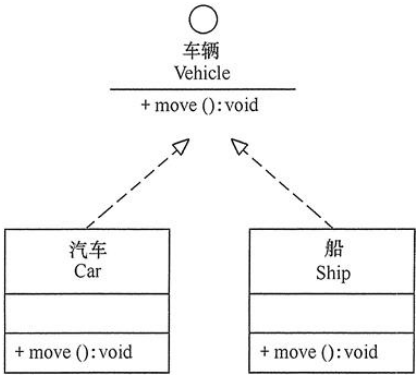

- 组合：关联关系的一种，也表示类之间的整体与部分的关系，部分对象不能脱离整体对象而存在，即共存.




- 泛化：父类与子类之间的关系




- 实现：接口与实现类之间的关系




**d 各种图**

用例图(系统与外部的交互关系，详见下一节)：参与者/用例/~~包含和扩展~~/通信关联

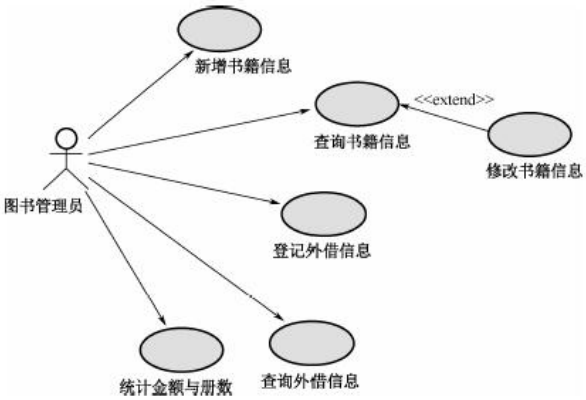



图 8-12　用例图示例

包图：表示软件体系结构图

类图和对象图：

类图：描述类的内部属性和行为，以及类集合之间的交互关系；

对象图：描述一组对象及它们之间的关系；

类关系：依赖(箭头虚线)/泛化(空心箭头实线)/关联(实线)/聚合(空心菱形实线)

多重性：如下图，1 本书籍包含 0 个或者 1 个借阅记录，即 1 对 0..1

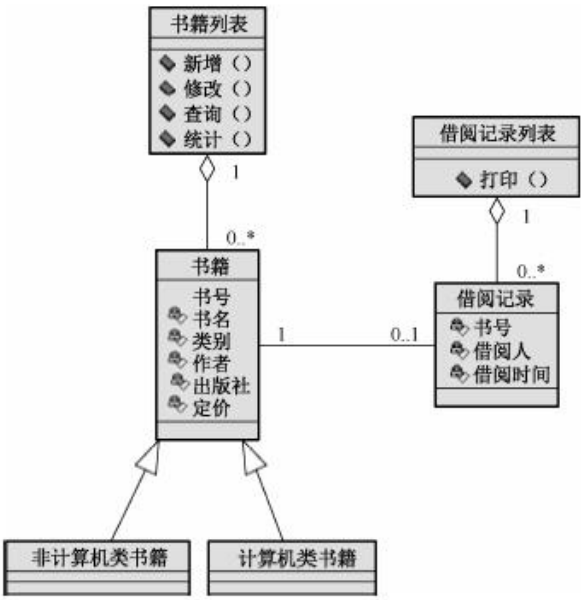



图 8-15　类图示例

交互图：表示各组对象如何依某种行为进行协作。可以用来表示用例的行为。

分类：顺序图/通信图/定时图

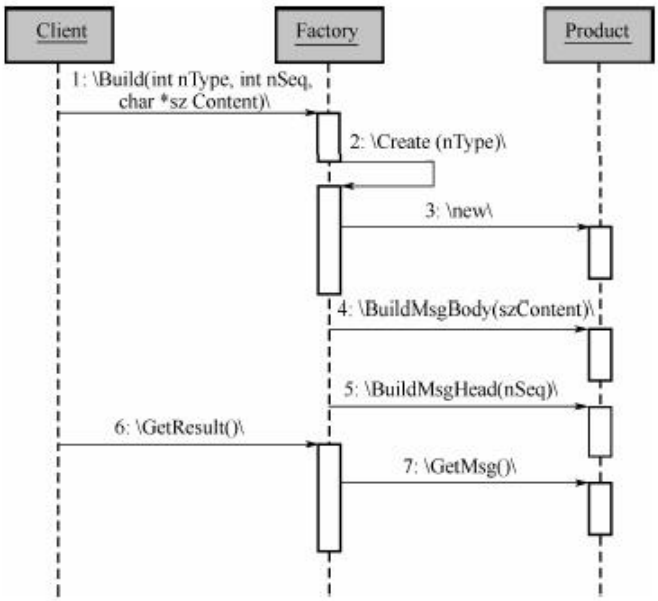



图 8-16　顺序图示例

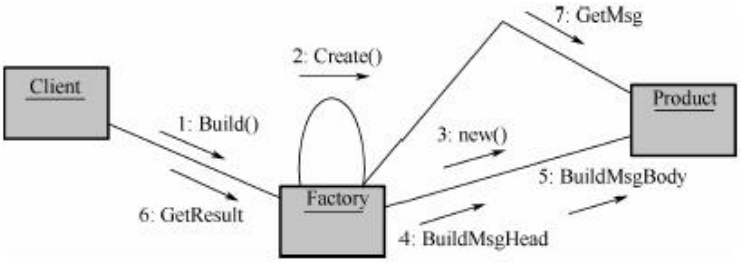



图 8-17　通信图示例

状态图：主要用于描述一个对象在其生存周期的动态行为，表现一个对象所经历的状态序列，引起状态转移的事件，以及因状态转移而伴随的动作。侧重描述行为的结果。（又：描述一个特定的复杂对象的所有可能状态及其引起状态转移的事件）

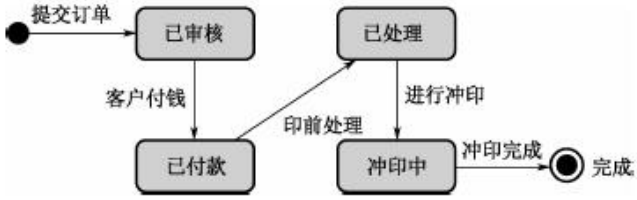



图 8-19　状态图示例

活动图：用于描述系统的工作流程和并发行为。活动图可以看作是状态图的特殊形式。侧重描述行为的动作。活动图中一个活动结束后将立即进入下一个活动，而在状态图中状态的转移可能需要事件的触发。

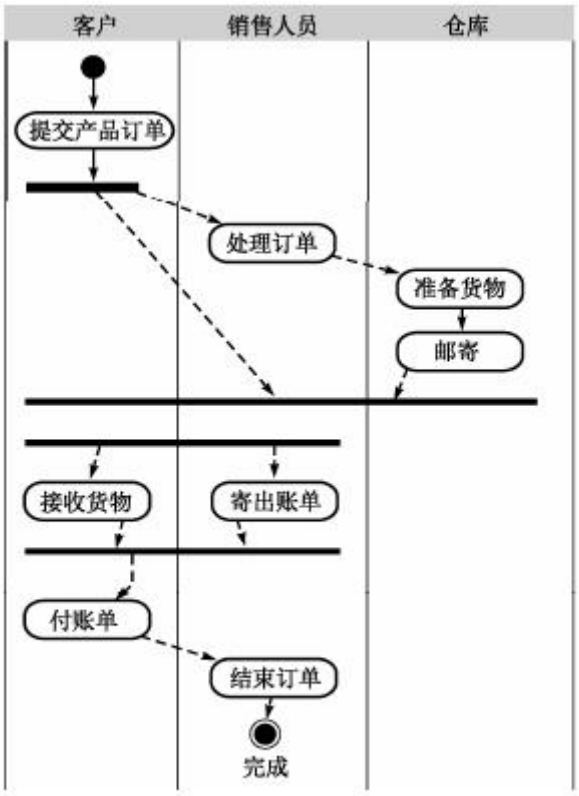



图 8-21　带泳道活动图示例

构件图：一组源代码**文件**、二进制代码文件和可执行文件(构件)之间的关系.

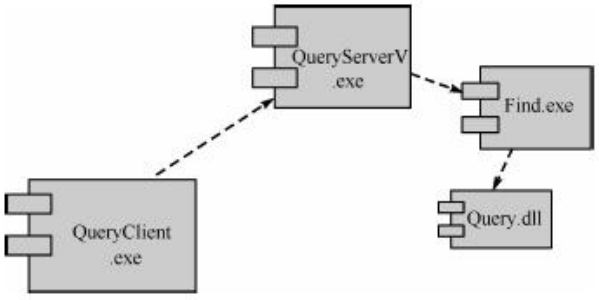



图 8-22　构件图示例

部署图(实施图)：在构件图基础上更进一步地描述系统硬件的物理拓扑结构及在此结构上执行的软件。

组成：结点和连接 / 构件和接口

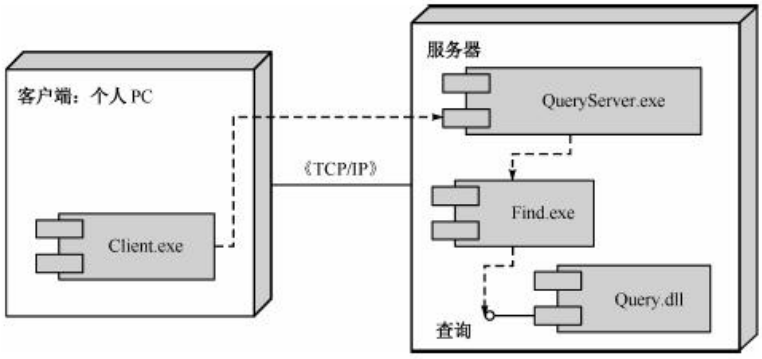



图 8-23　部署图示例

### 2.5.3 用例模型

利用图：用例图

用来描述系统需求，用例图的元素包括参与者、用例和通信关联。

建模步骤包括 4 步：

- a 识别参与者
- b 合并需求获得用例
- c 细化用例描述(例子见书 P444)

  用例名称

  简要说明

  事件流

  非功能需求

  前置条件和后置条件

  扩展点

  优先级
- d 调整用例模型

  用例之间的关系包括 **包含/扩展/泛化**，基于这些关系对用例进行调整.

### 2.5.4 分析模型

利用图：用例图/类图/交互图

用来描述系统的基本逻辑结构，展示对象和类如何组成系统(静态模型),以及它们如何保持通信,实现系统行为(动态模型)．由顶层架构图、用例与用例图和领域概念模型构成.

步骤：

- a 定义概念类
- b 确定类之间的关系

  关系包括：关联/依赖/泛化/聚合/组合/实现 （见课本 P448）

  用类图表示
- c 为类添加职责

  成员变量(属性) 和 成员方法
- d 建立交互图

  包括顺序图/交互概览图/通信图/定时图
- e 分析模型的详细程度问题

### 2.5.5 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**；描述流程化处理的**活动图**

面向对象设计原则：见本文档 4.1

## 2.6 用户界面设计

黄金法则

置用户于控制之下/减少用户记忆负担/保持界面一致

过程

用户、任务和环境分析/界面设计/实现/界面确认

## 2.7 工作流设计

流程定义工具

工作流管理系统

## 2.8 简单分布式计算机应用系统设计

基于实例的协作:对远程实例有较大控制权,小范围内”近连接”,常使用代理方式.

基于服务的协作:只能调用远程对象的接口方法,无法创建销毁远程对象.跨平台”远连接”.

### 2.8.1 分布式系统开发

五个逻辑计算层：

- **表示层**实现用户界面；
- **表示逻辑层**为了生成数据表示而必须进行的处理任务，如输入数据编辑等；
- **应用逻辑层**包括为支持实际业务应用和规则所需的应用逻辑和处理过程,如信用检查、数据计算和分析等；
- **数据处理层**包括存储和访问数据库中的数据所需的应用逻辑和命令,如查询语句和存储过程等；
- **数据层**是数据库中实际存储的业务数据。

### 2.8.2 分布式计算架构

①**分布式表示架构**是将表示层和表示逻辑层迁移到客户机，应用逻辑层、数据处理层和数据层仍保留在服务器上；

②**分布式数据架构**是将数据层和数据处理层放置于服务器，应用逻辑层、表示逻辑层和表示层放置于客户机；

③**分布式数据和应用架构**是将数据层和数据处理层放置在数据服务器上,应用逻辑层放置在应用服务器上，表示逻辑层和表示层放置在客户机上。

## 2.9 系统运行环境的集成与设计

软件运行环境: 包括 系统运行的设备/操作系统/网络配置

集中式系统: 单计算机结构/集群结构/多计算机结构

分布式系统: 基于网络

C/S 结构: 如数据库/网络打印服务系统/网络游戏

多层结构 3: 扩展了 C/S, 由存储数据的数据库服务器作为数据层、实现商业规则的程序作为逻辑层、管理用户输入输出的视图层

Internet、Intranet、Extranet

### 2.10 系统过渡设计

直接过渡/并行过渡/阶段过渡

# 三. 软件架构设计

### 3.1 概述

**a 定义**

软件或计算机系统的软件架构是该系统的一个（或多个）结构，而结构由软件元素、元素的外部可见属性及它们之间的关系组成。

软件系统架构是关于软件系统的 **结构、行为和属性** 的高级抽象。指定了软件系统的**组织结构和拓扑结构**。

**b 架构的模型**

4+1 视图模型：

**逻辑视图**：主要支持系统的功能需求，即系统提供给最终用户的服务。

**开发视图**：也称为模块(实现)视图，主要侧重于软件模块的组织和管理。

**进程视图**：侧重于系统的运行特性，主要关注一些非功能性的需求，例如系统的性能和可用性。

**物理视图**：主要考虑如何把软件映射到硬件上，它通常要考虑到解决系统拓扑结构、系统安装、通信等问题。

**场景**：可以看作是那些重要系统活动的抽象，它使四个视图有机地联系起来，从某种意义上说，场景是最重要的需求抽象。

逻辑视图和开发视图描述系统的静态结构，而进程视图和物理视图描述系统的动态结构。

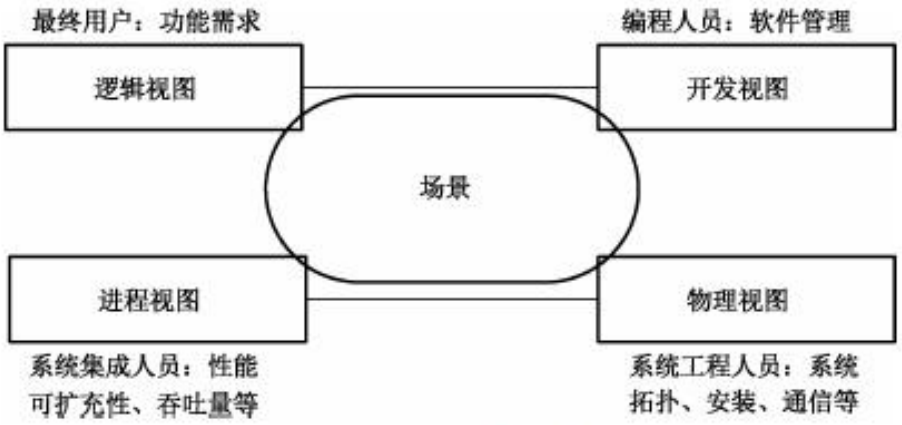



图 9-1 “4+1”视图模型

### 3.2 软件质量属性

**可用性**

**可修改性**(可维护性/可扩展性/结构重构/可移植性)

**性能**

**安全性**

可测试性、易用性、功能性、互操作性、健壮性

(1) **性能**是指系统的响应能力，即要经过多长时间才能对某个事件做出响应，或者在某段时间内系统所能处理事件的个数。

(2) **可用性**是系统能够正常运行的时间比例。(通过用两次故障之间的时间长度或出现故障时系统能够恢复的速度来表示)

~~(3) 可靠性是指软件系统在应用或错误面前，在意外或错误使用的情况下维持软件系统功能特性的基本能力。~~

(4) 健壮性是指在处理或环境中，系统能够承受压力或变更的能力。

(5) 安全性是指系统向合法用户提供服务的同时能够阻止非授权用户使用的企图或拒绝服务的能力。

(6) **可修改性**是指能够快速地以较高的性能价格比对系统进行变更的能力。

(7) 可变性是指体系结构经扩充或变更成为新体系结构的能力。

(8) 易用性是衡量用户使用一个软件产品完成指定任务的难易程度。

(9) 可测试性是指软件发现故障并隔离、定位其故障的能力特性，以及在一定的时间和成本前提下，进行测试设计、测试执行的能力。

(10) 功能性是系统所能完成所期望工作的能力。

(11) 互操作性是指系统与外界或系统与系统之间的相互作用能力。

### 3.2.1 ISO 质量属性


**一、功能性：**

1、适合性：软件是否提供了相应的功能

2、准确性：软件提供的功能是否正确（用户需要的）

3、互操作性：产品与产品之间交互数据的能力,例如 word 对其他文档的支持能力

4、保密安全性：允许经过授权的用户和系统能够正常的访问相应的数据和信息，禁止未授权的用户访问

5、功能性的依从性：国际/国家/行业/企业 标准规范一致性

**二、可靠性(度)：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(概率)。**

***子特性速记：成容一一***

1、成熟性：软件产品为避免软件内部的错误扩散而导至系统失效的能力（主要是对内错误的隔离）,exception 等的处理；

2、容错性：软件防止外部接口错误扩散而导致系统失效的能力（主要是对外错误的隔离）；

3、易恢复性：系统失效后，重新恢复原有的功能和性能的能力；

4、可靠性的依从性。

**三、易用性：在指定使用条件下，产品被理解、 学习、使用和吸引用户的能力**

1、易理解性：软件交互给用户的信息时，要清晰，准确，且要易懂，使用户能够快速理解软件。

2、易学性：软件使用户能学习其应用的能力。

3、易操作性：软件产品使用户能易于操作和控制它的能力。

4、吸引性：

5、易用性的依从性：

**四、效率性：在规定台条件下，相对于所用资源的数量，软件产品可提供适当性能的能力**

1、时间特性：平均事务响应时间，吞吐率，TPS（每秒事务数）. 软件处理特定的业务请求所需要的响应时间。

2、资源利用性：CPU 内存 磁盘 IO 网络带宽 队列 共享内存. 软件处理特定的业务请求所消耗的系统资源。

3、效率依从性

**五、软件维护性："四规"， 在规定条件下，规定的时间内，使用规定的工具或方法修复规定功能的能力**

1、易分析性：分析定位问题的难易程度

2、易改变性：软件产品使指定的修改可以被实现的能力

3、稳定性：防止意外修改导致程序失效

4、易测试性：使已修改软件能被确认的能力

5、维护性的依从性

**六、软件可移植性：从一种环境迁移到另一种环境的能力**

1、适应性：适应不同平台

2、易安装性：被安装的能力

3、共存性：软件产品在公共环境中与其它软件分享公共资源共存的软件。

4、易替换性: 软件产品在同样的环境下，替代另一个相同用途的软件产品的能力。

5、可移植性的依从性：

### 3.2.2 优化方法

提高可用性：Ping/Echo；命令/响应机制、心跳机制、异常处理机制、冗余机制

提高性能：队列调度；引入并发、维持数据或计算的多个副本、增加可用资源、控制采样频度、限制执行时间、固定优先级调度

提高安全性：检测攻击、抵御攻击、从攻击中恢复、身份认证、限制访问、维护完整性；入侵检测；追踪审计

提高可修改性：运行时注册(即插即用)；接口-实现分离；信息隐藏（封装）

提高可测试性：记录-回放

### 3.3 软件架构风格

软件架构风格是描述特定软件系统**组织方式的惯用模式**。组织方式描述了系统的组成构件和这些构件的组织方式；惯用模式则反映众多系统共有的结构和语义特性。

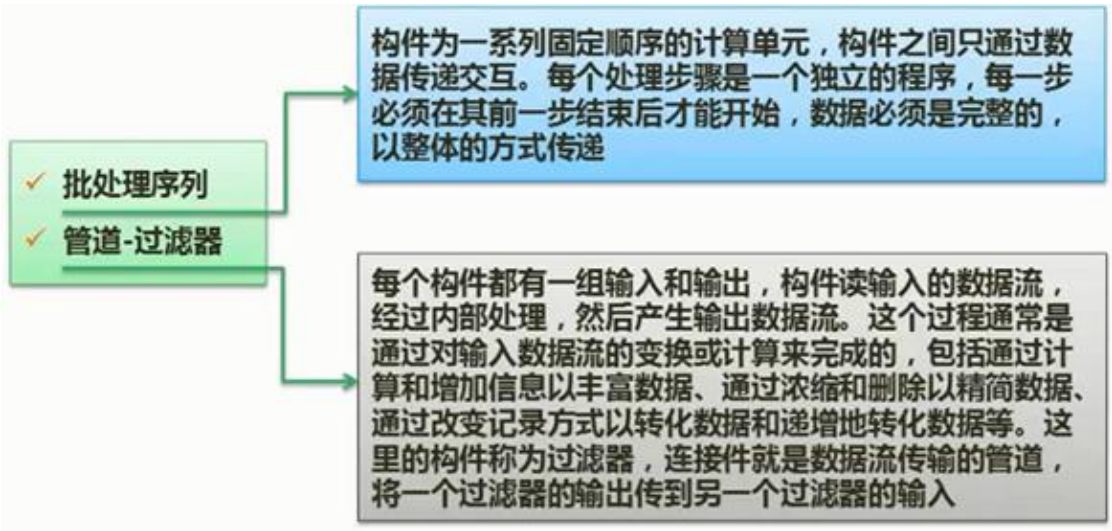
架构风格定义了一个系统家族，即一个架构定义一个**词汇表**和一组**约束**。词汇表中包含一些构件和连接件类型，而这组约束指出系统是如何将这些构件和连接件组合起来的。

架构风格反映了领域中众多系统所共有的**结构**和**语义特性**，并指导如何将各个模块和子系统有效地组织成一个完整的系统。

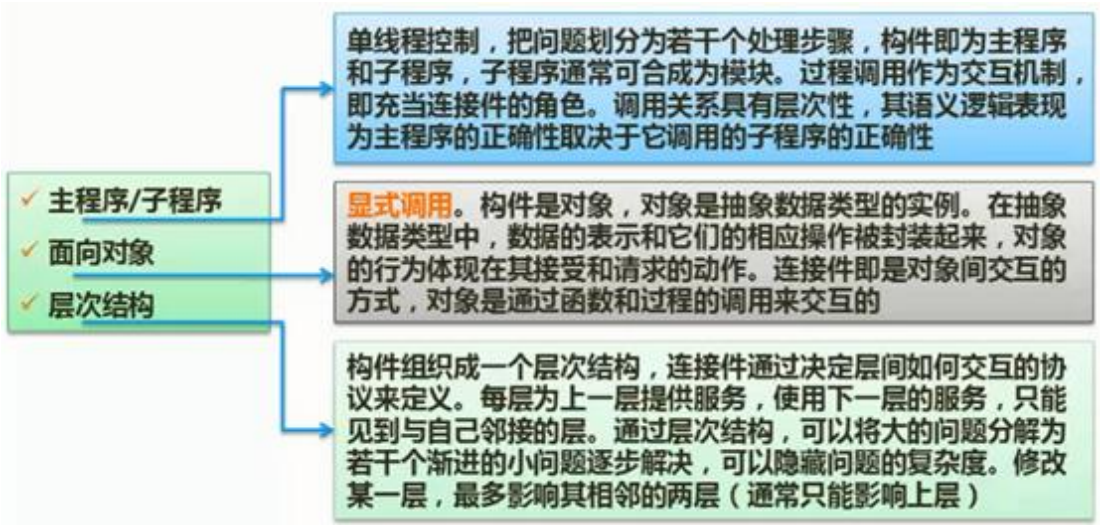
#### a 数据流风格

批处理序列：强调数据作为一个整体

管道和过滤器:每个构件都有一组输入和输出,构件读输入的数据流，经过内部处理，然后产生输出数据流. （构件-->过滤器;连接件-->管道）




#### b 调用/返回风格

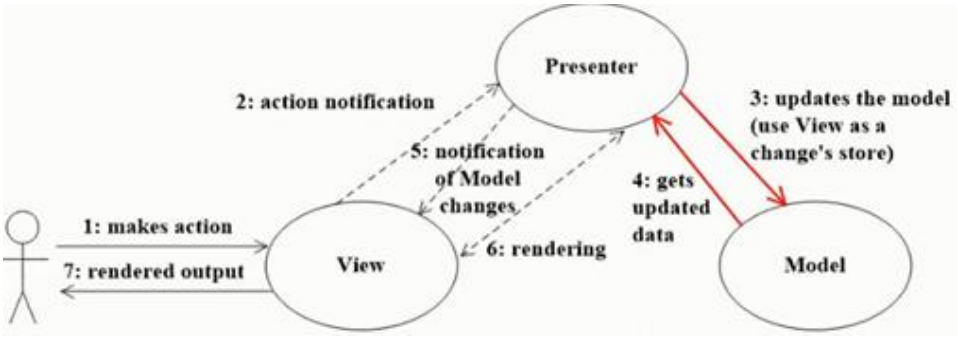



主程序/子程序:计算构件作为子程序协作工作,由一个主程序顺序地调用这些子程序,构件通过共享存储区交换数据. 曾经作为结构化开发方法的主要选择，具有结构清晰，维护方便的特点，缺点是主子程序划分缺乏标准，较难实现不同设计人员间设计的子程序复用。

面向对象风格：面向对象在类的层次实现高度内聚，整个系统通过不同类的组合调用实现不同功能，便于类的复用，只是面向对象是一个通用风格，类的划分不同设计人员设计结果有很大不同，对实际架构设计指导意义不大。

层次结构风格：分层结构将整个系统按照抽象层次不同分为多层，每个层次的程序只需要负责与相邻的上下两层打交道，简化了系统中调用关系复杂度。允许每层用不同的方法实现，为软件重用提供了强大的支持。

- 二层 C/S
- 三层 C/S: 表示层/功能层/数据层. B/S 为其一种特例
- MVC: 模型(model)－视图(view)－控制器(controller)
- MVP: 将视图 V 和模型 M 解耦

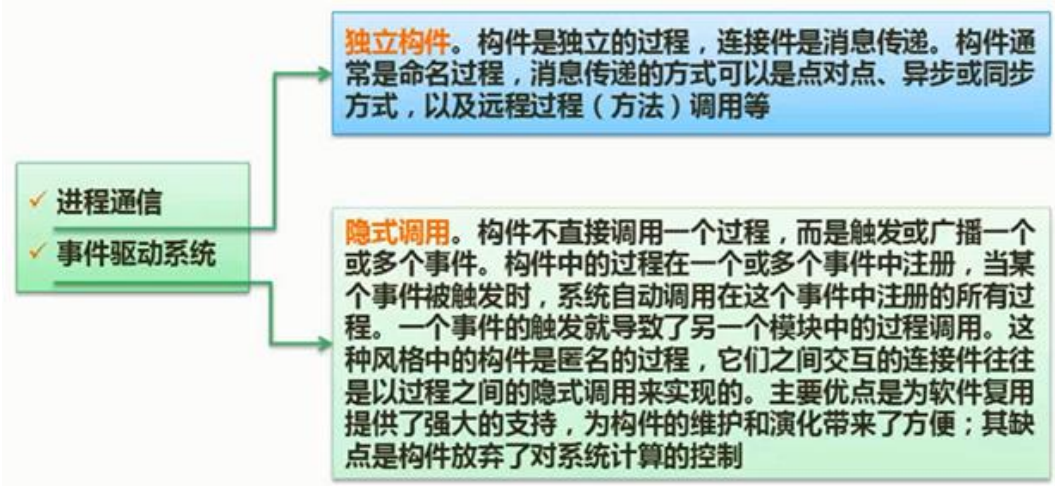



**c 独立构件风格**

进程通讯：进程通讯架构将系统建设成一个个独立构件，构件间采用命名的消息传递来实现沟通与协作。

事件系统子风格：事件驱动架构风格与进程通讯风格类似，也是将系统分各个为独立的构件，不同的是不同构件间通讯不采用命名的消息，而是采用隐式调用的方式，先将一个个构件的过程注册到某个事件中，当这个事件发生时，所有注册到该事件的过程自动被触发执行。这类风格的好处是独立构件间耦合度进一步降低，方便构件修改及替换，缺点是触发事

件放弃了对被触发执行程序组的控制。




**d 虚拟机风格**

解释器：具有运行时系统行为**(自)定义**与改变能力. 如专家系统

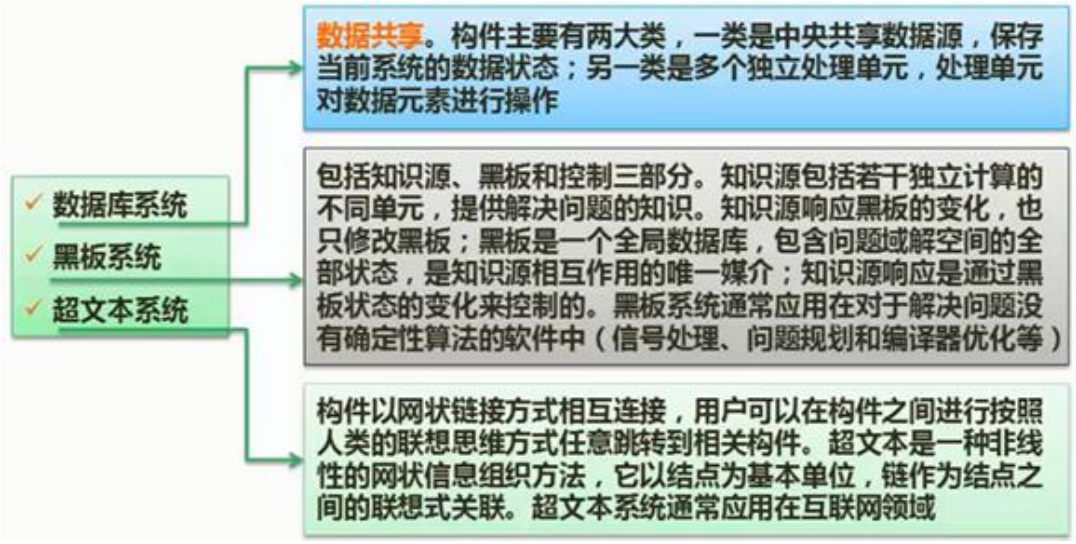
基于规则的系统

**e 仓库风格**

数据库系统：构件主要有两大类,一个是中央共享数据源,保存当前系统的数据状态,另一个是多个独立处理元素,处理元素对数据元素进行操作。中央数据库管理系统通过自身机制如数据排它锁，共享锁等，实现数据高速访问，数据一致性，数据完整性。同时各独立数据处理单元之间互相不受约束。（如*编译器*，传统编译器采用批处理架构，现代编译器采用数据共享架构风格。分析树是共享数据。）

超文本系统：主要应用于静态网页。

黑板风格：由一个作为全局共享数据的黑板，一个控制单元和多个知识源组成，主要应用与专家问题解决系统。通过专家知识和反馈逐步得到正确结果.（如语音识别）



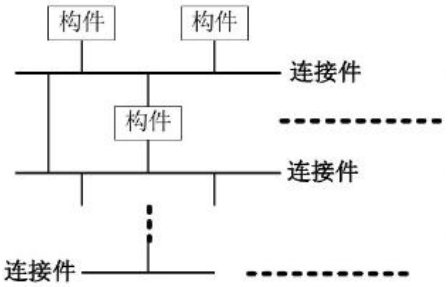

**其它风格:** 消息总线/面向服务架构

**补充**

**过程控制**：工业中的过程控制是指以温度、压力、流量、液位和成分等工艺参数作为被

控变量的自动控制。过程控制也称**实时控制**，是计算机及时的采集检测数据，按最佳值迅速地对控制对象进行自动控制和自动调节，如数控机床和生产流水线的控制等。

C2：通过连接件绑定在一起按照一组规则运作的**并行**构件.




### 3.4 分层 C/S 架构

◆ **二层 C/S 架构缺点**

1. 二层 C/S 结构为单一服务器且以局域网为中心，所以难以扩展至大型企业广域网或 Internet；（使用范围）

2. 软、硬件的组合及集成能力有限；**（扩展性）**

3. 服务器的负荷太重，难以管理大量的客户机，系统的性能容易变坏；**（性能）**

4. 数据安全性不好。因为客户端程序可以直接访问数据库服务器，那么，在客户端计算机上的其他程序也可想办法访问数据库服务器，从而使数据库的安全性受到威胁。**（安全）**

（1）开发成本较高。C/S 架构对客户端软硬件配置要求较高，尤其是软件的不断升级，对硬件要求不断提高，增加了整个系统的成本，且客户端变得越来越臃肿。

（2）客户端程序设计复杂。采用 C/S 架构进行软件开发，大部分工作量放在客户端的程序设计上，客户端显得十分庞大。

（3）信息内容和形式单一，因为传统应用一般为事务处理，界面基本遵循数据库的字段解释，开发之初就已确定，用户获得的只是单纯的字符和数字，既枯燥又死板。

（4）用户界面风格不一，使用繁杂，不利于推广使用。

（5）软件移植困难。采用不同开发工具或平台开发的软件，一般互不兼容，不能或很难移植到其它平台上运行。

（6）软件维护和升级困难。采用 C/S 架构的软件要升级，开发人员必须到现场为客户机升级，每个客户机上的软件都需维护。对软件的一个小小改动，每一个客户端都必须更新。

（7）新技术不能轻易应用。因为一个软件平台及开发工具一旦选定，不可能轻易更改。

◆ **三层 C/S 架构**

**概念 (以三层 B/S 为例):**

**1 表现层（Web 层）**

- 负责接收客户端请求，向客户端响应结果，通常客户端使用 http 协议请求 web，web 层需要接收 http 请求，完成 http 响应。
- 表现层包括展示层和控制层：控制层负责接收请求，展示层负责结果的展示。
- 表现层依赖业务层，接收到客户端请求一般会调用业务层进行业务处理，并将处理结果响应给客户端。

- 表现层的设计一般都使用 MVC 模型。 MVC 是表现层的设计模型，和其他层没有关系。

**2 业务层 （Service 层）**

- 它负责业务逻辑处理，和我们开发项目的需求息息相关。web 层依赖业务层，但是业务层不依赖 Web 层。
- 业务层在业务处理时可能会依赖持久层，如果要对数据持久化需要保证事务一致性。（事务应该放到业务层来控制）

**3 持久层 （dao 层）**

- 负责数据持久化，包括数据层即数据库和数据访问层，数据库是对数据进行持久化的载体，数据访问层是业务层和持久层交互的接口；业务层需要通过数据访问层将数据持久化到数据库中。
- 持久层就是和数据库交互，对数据库表进行增删改查的。

**优点:**

（1）允许合理地划分三层结构的功能，使之在逻辑上保持相对独立性，从而使整个系统的逻辑结构更为清晰，能提高系统和软件的可维护性和可扩展性。**(逻辑独立清晰，可维护性/可扩展性)**

（2）允许更灵活有效地选用相应的平台和硬件系统，使之在处理负荷能力上与处理特性上分别适应于结构清晰的三层;并且这些平台和各个组成部分可以具有良好的可升级性和开放性。**(可升级性/开放性)**

（3）三层 C/S 架构中，应用的各层可以并行开发，各层也可以选择各自最适合的开发语言。使之能并行地而且是高效地进行开发，达到较高的性能价格比；对每一层的处理逻辑的开发和维护也会更容易些。**(开发维护成本/速度/技术门槛)**

（4）允许充分利用功能层有效地隔离开表示层与数据层，未授权的用户难以绕过功能层而利用数据库工具或黑客手段去非法地访问数据层，这就为严格的安全管理奠定了坚实的基础；整个系统的管理层次也更加合理和可控制。**(安全)**

**◆ B/S 架构**

**优点:**

1. 用户在使用系统时，仅仅需要一个浏览器就可运行全部的模块，真正达到了“零客户端”的功能，很容易在运行时自动升级。**(客户端)**

2. 基于 B/S 架构的软件，系统安装、修改和维护全在服务器端解决。**(服务端)**

3. B/S 架构还提供了异种机、异种网、异种应用服务的联机、联网、统一服务的最现实的开放性基础。**(开放性)**

**缺点:**

（1）B/S 架构缺乏对**动态页面**的支持能力，没有集成有效的数据库处理功能。

（2）B/S 架构的系统扩展能力差，**安全性**难以控制。

（3）采用 B/S 架构的应用系统，在数据查询等响应速度上，要远远地低于 C/S 架构。**(性能)**

（4）B/S 架构的数据提交一般以页面为单位，数据的动态**交互性**不强，不利于 OLTP 应用.

### 3.4 面向服务的架构 SOA

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

基于 XML
主要有两种实现方式: web service 和 ESB
实现方法: Web Service / 服务注册表 / 企业服务总线 ESB

**6 个 服务类型**:

1. 连接服务

连接服务又称连通服务，是面向服务架构的骨干，在完成**服务的接入，服务间的通信和交互**基础上，还提供**安全性、可靠性、高性能**的服务能力保障。连接服务的一个典型实现就是**企业服务总线**(Enterprise Service Bus，ESB)。

2. 协作服务

协作服务通常由通信代理和 Web 服务代理两部分组成。通信代理与连通服务中的通信代理实现内部有效的数据通信，Web 服务代理与外部的公共注册中心交互，注册本平台对外开放的 Web 服务以及查找所需要访问的外部 Web 服务。协作服务既可以实现**组织之间**(如供应链的合作伙伴之间)的交互通信，也可以实现**组织内部**(如跨地域的分支机构之间，并有防火墙进行保护的情况)之间的交互通信。

3. 业务服务

业务服务指**为新建服务提供的特定运行支持环境**。新建服务包括单个服务以及合成服务，不包括流程化的服务。合成服务一般由应用编码实现，它可以调用其他的服务(包括：单个服务、合成服务和流程化的服务)。业务服务与连通服务相联接，其中的新建服务与其他服务的通信和交互通过连通服务来实现。业务服务的运行信息由运行管理服务保存，业务服务也接受并执行运行管理服务的管理和控制命令。

4. 业务流程服务

流程服务是**业务流程的运行环境，提供流程驱动、服务调用、事务管理等功能**。流程服务是为业务流程的运行提供的一组标准服务。业务流程是**一组服务**的集合，可以按照特定的顺序并使用一组特定的规则进行调用。业务流程可以由不同粒度的服务组成，其本身也可

视为服务。

5. 交互服务

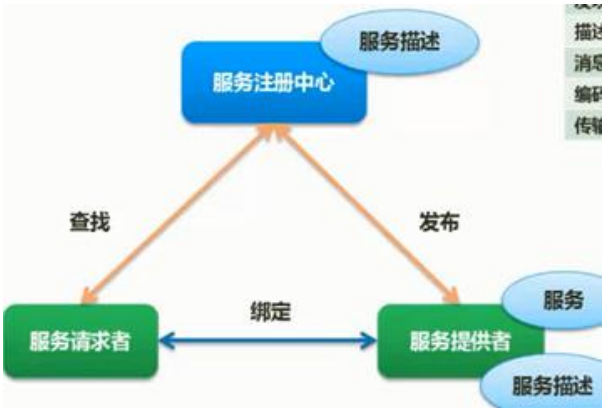
交互服务**实现人与服务之间的交互功能**。人可以是服务的消费者，也可以是服务的提供者。人不能直接消费服务，也不能直接提供服务，需要通过相应的程序实现代理操作(即人通过操作程序实现与服务的交互)。交互服务就是需要提供一组完整的功能，以实现人与服务的交互，并能够方便地进行交互。人员需要请求服务时，向连通服务发送消息请求，由连通服务查找服务，并将请求消息传递给服务提供者。

6. 信息服务

信息服务特指为上层应用系统、同层的其他服务等**提供数据访问及资源访问**服务。其目标是使应用系统能够统一、透明、高效地访问和操纵位于网络环境中的各种分布、异构的数据资源，为实现全局数据访问、加快应用开发、增强网络应用和方便系统管理提供支持。

### 3.4.1 WEB SERVICE

由服务请求者、服务提供者、服务注册中心三个角色构成, 支持服务发布、查找和绑定。




| 功能 | 协议 |
| --- | --- |
| 发现服务 | UDDI、DISCO |
| 描述服务 | WSDL、XML Schema |
| 消息格式层 | SOAP、REST |
| 编码格式层 | XML |
| 传输协议层 | HTTP、TCP/IP 、SMTP等 |

关键技术: UDDI / WSDL / SOAP / REST /XML

**UDDI**(universal description,**discovery**,intergration)统一描述、发现和集成：用于 Web 服务注册和服务查找；

**WSDL**(**web service description** language)web 服务描述语言：用于描述 Web 服务的接口和操作功能；

**SOAP**(simple object access protocol)简单对象访问协议：为建立 Web 服务和服务请求之间的通信提供支持；

**BPEL**(business process execution language) 企业过程执行语言：用来将分散的、功能单一的 web 服务组织成一个复杂的有机应用.

**REST**(representational state transfer)表述性状态转移：REST 是一种使用 HTTP、XML 技术进行基于 web 通信的技术，它将网络中所有的事物抽象为资源，每个资源对应唯一的统

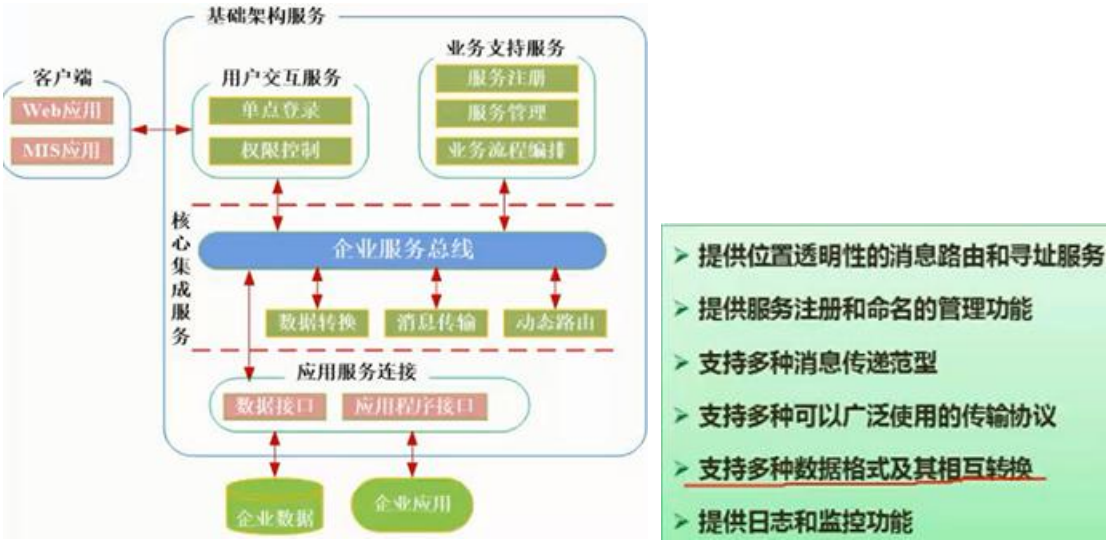
一资源标识符 URI。客户端通过 URI 获取资源的表述，并通过获得资源的表述使得其状态发生改变。REST 将*资源/资源的表述/获取资源的动作*三者进行分离。

### 3.4.2 企业服务总线 ESB

**概念：**ESB（enterprise service bus）是传统中间件技术与 XML、web 服务等技术结合的产物，主要支持异构系统集成。ESB 基于内容的路由和过滤，具备复杂数据的传输能力，并可以提供一系列标准接口。

**功能：**速记—监消三传服安

- 监控与管理
- 消息路由
- 消息增强
- 消息格式转换
- 传输协议转换
- 服务位置透明性
- 安全性




### 3.4.3 微服务

概念：微服务是一种新型的软件架构，把一个大型的单体应用程序和服务拆分为多个支持的微服务.

内容：一个微服务需要包含完整的业务功能,开放一种或多种接口为其他服务使用,并可能包含一个自己私有的数据库.

优势：

**（1）技术异构性**

在微服务架构中，每个服务都是一个相对独立的个体，每个服务都可以选择适合于自身的技术来实现。可以混合使用多种编程语言、开发框架以及数据存储技术.

**（2）扩展**

单块系统中，我们要做扩展，往往是整体进行扩展。而在微服务架构中，可以针对单个

服务进行扩展。

**（3）简化部署**

简单的服务更容易部署，每个服务的部署都是独立的，单个服务出问题不会导致整个系统的故障.

**（4）与组织结构相匹配**

为服务强调模块化的结构，可以将架构与组织结构相匹配，避免出现过大的代码库，从而获得理想的团队大小及生产力。

**（5）可组合性**

在微服务架构中，系统会开放很多接口供外部使用。当情况发生改变时，可以使用不同的方式构建应用（而整体化应用程序只能提供一个非常粗粒度的接口供外部使用）。

挑战：

**（1）分布式系统的复杂度**

分布式系统的编程难度更大，服务间的通信都是通过网络远程调用，因此性能和可靠性都会受影响.

**(2) 最终一致性**

强一致性较难实现，开发人员需要处理最终一致性的问题.

**(3) 运维成本**

每个服务都需要独立的**配置、部署、监控、日志收集**等，因此运维成本较高。

**(4) DevOps 与组织结构**

微服务不仅表现出一种架构模型，同样也表现出一种组织模型----全功能团队。这种新型的组织模型意味着开发人员和运维的角色发生了变化，开发者将承担起服务整个生命周期的责任，包括部署和监控，而运维也越来越多地表现出一种顾问式的角色，尽早考虑服务如何部署。

## 3.5 架构设计

演变交付生命周期

属性驱动设计法

按架构组织开发团队

开发骨架系统

利用商用构件进行开发

## 3.6 软件架构文档化

视图编档

视图概述

元素目录

上下文图

可变性指南

架构背景

术语表

其他信息

跨视图文档

使用 UML

## 3.7 软件架构评估

### 3.7.1 关键步骤

风险点：架构设计中潜在的、存在问题的架构决策所带来的隐患。（业务逻辑不明确的地方）

非风险点：非隐患

敏感点：为了实现某种特定的质量属性，一个或多个构件所具有的特征；

权衡点：影响多个质量属性的特征，是多个质量属性的敏感点（如对两个质量属性特征产生相反的影响，一个好一个坏）

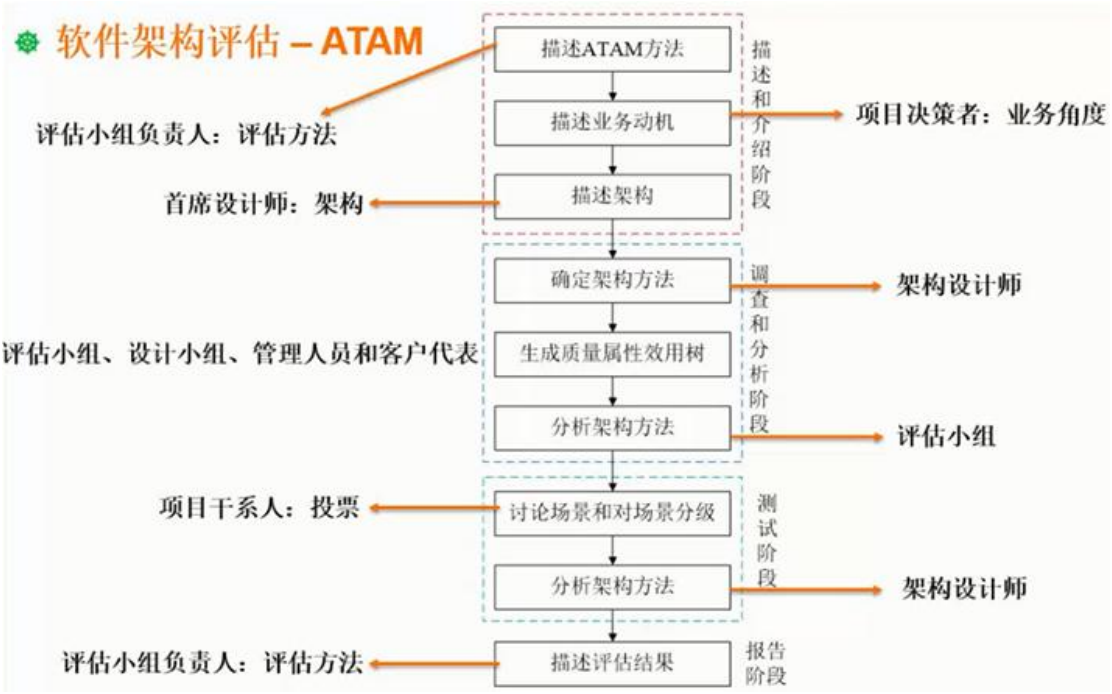
### 3.7.2 评估方法

- 基于调查问卷或检查表的方式
- 基于度量的方式
- 基于场景的方式

以**属性**作为架构评估的核心概念.

**a 架构权衡分析法 ATAM(architecture tradeoff analysis method)**

包括 **场景和需求收集、架构视图和场景实现、属性模型构造和分析、属性模型折中。**




**b 软件架构评估 SAAM(software architecture analysis method)**

主要输入：**问题描述/需求说明/架构描述**

分析过程：**场景开发/架构描述/单个场景评估/场景交互/总体评估**

先评估单个，再评估多个相互作用


**c 成本效益分析法 CBAM(cost benefit analysis method)**

### 3.8 构件复用

构件概念：软件构件是软件系统中具有一定意义的、相对独立的可重用单元。与对象相比，构件可以基于对象实现，也可以不作为对象实现。构件需要在容器中管理并获取容器提供的服务；客户程序可以在运行状态下利用接口动态确定构件所支持的功能并调用。

面向构件的编程需要下列基本的支持：多态性（可替代性）、模块封装性（高层次信息的隐藏）、后期的绑定和装载（部署独立性）、安全性（类型和模块安全性）。

构件组装成软件系统的过程可以分为 3 个层次：定制/集成/拓展

#### 3.8.1 商用构件标准规范

- CORBA

✧ 三个层次

**对象请求代理(ORB)**： 最底层，规定了分布对象的定义（接口）和语言映射，实现对象间的通信和互操作，是分布对象系统中的“软总线”.

**公共对象服务**：提供诸如并发服务、名字服务、事务（交易）服务、安全服务等各种各样的服务.

**公共设施**：最上层，定义了构件框架，提供可直接为业务对象使用的服务，规定业务对象有效协作所需的协定规则.

✧ 四种构件

**实体（Entity）构件**需要长期持久化并主要用于事务性行为，由容器管理其**持久化**。

**加工（Process）构件**同样需要容器管理其持久化，但没有客户端可访问的主键。

**会话（Session）构件**不需要容器管理其持久化，其状态信息必须由构件自己管理。负责完成服务端和客户端的交互。

**服务（Service）构件**是无状态的。

- EJB/J2EE

支持的 5 种构件模型：Applet、Servlet、JSP、EJB、Application Client。

其中,EJB 中的 BEAN 分三种:

  - session bean 会话 bean：负责维护一个短暂的会话；
  - entity bean 实体 bean:负责维护一行稳固持久的数据；
  - message-driven bean 消息驱动 bean:异步接受消息。

无状态的 bean 没有成员变量，用单例模式 ；有状态的 bean 有成员变量，非线程安全，

适合用 prototype 原型模式。

- 微软的 COM/DCOM/COM+

### 3.8.2 应用系统簇与构件系统

### 3.8.3 基于复用开发的组织结构

## 3.9 产品线及系统演化

概念：软件产品线是指一组软件密集型系统，它们**共享**一个公共的、可管理的特性集，满足某个**特定市场**或任务的具体需要，是以规定的方式用公共的**核心资产**集成开发出来的。

包含的技术：软件架构/领域工程/DSSA

可复用的资产：///

### 3.9.1 特定领域软件架构(DSSA)

- 概念

以一个特定问题领域为对象，形成由领域**参考模型、参考需求、参考架构**等组成的开发基础架构，其目标是支持一个特定领域中多个应用的生成。

- 活动阶段

（1）**领域分析**：主要目标是获得领域模型。即通过分析领域中系统的需求（领域**需求**），确定哪些需求是被领域中的系统广泛共享的，从而建立领域模型。

（2）**领域设计**：这个阶段的目标是获得 DSSA，它是一个能够适应领域多个系统的需求的一个高层次的设计。由于领域模型中的领域需求具有一定的变化性，DSSA 也要相应地具有变化性，它可以通过表示多选一的、可选的解决方案等来做到这一点。

（3）**领域实现**：主要目标是依据领域模型和 DSSA 开发与组织可复用信息。这些复用信息可以是从现有系统中提取得到的，也可能通过新的开发得到。这个阶段可以看作复用基础设施的实现阶段。

- 参与人员

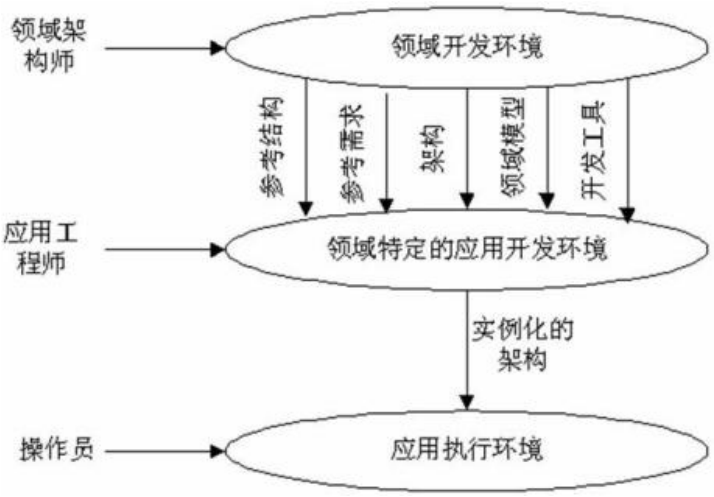
**领域专家**：主要任务包括提供关于领域中系统的需求规约和实现的知识，帮助组织规范的一致的领域字典，帮助选择样本系统作为领域工程的依据，复审领域模型、DSSA 等领域工程产品.

**领域分析人员**：主要任务包括控制整个领域分析过程，进行知识获取，将获取的知识组织到领域模型中，根据现有系统、标准规范等验证领域模型的准确性和一致性，维护领域模型。

**领域设计人员**：主要任务包括控制整个软件设计过程，根据领域模型和现有的系统开发出 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.

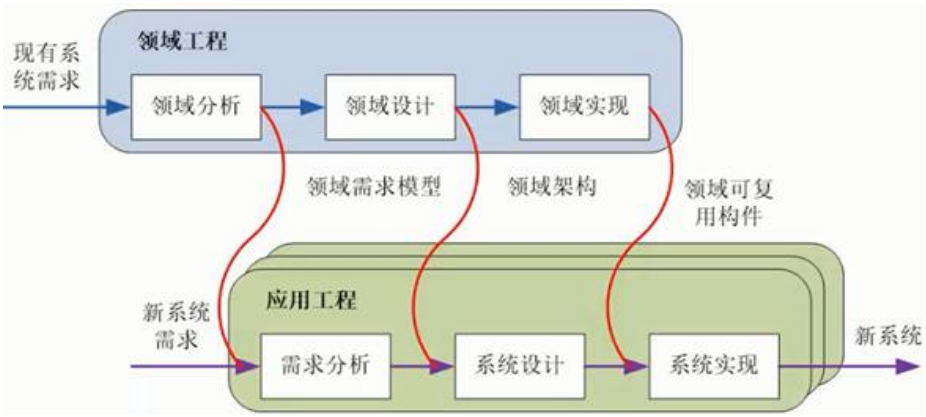
**领域实现人员**：主要任务包括根据领域模型和 DSSA，或者从头开发可重用构件或者利用再工程的技术从现有系统中提取可重用构件，对可重用构件进行验证，建立 DSSA 与可重用构件间的联系.

- DSSA 系统模型

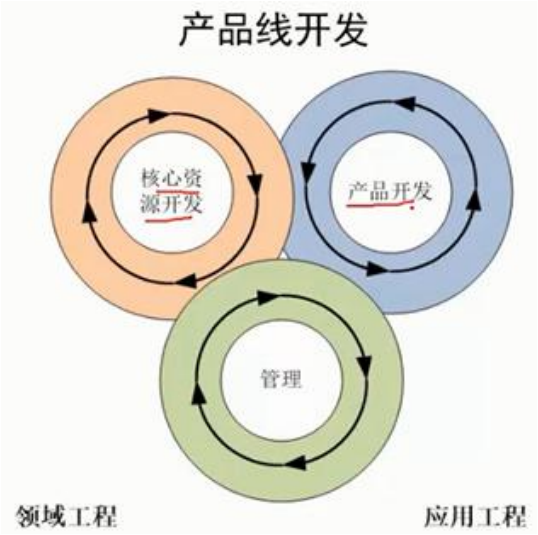



### 3.9.2 过程模型

- 双生命周期模型

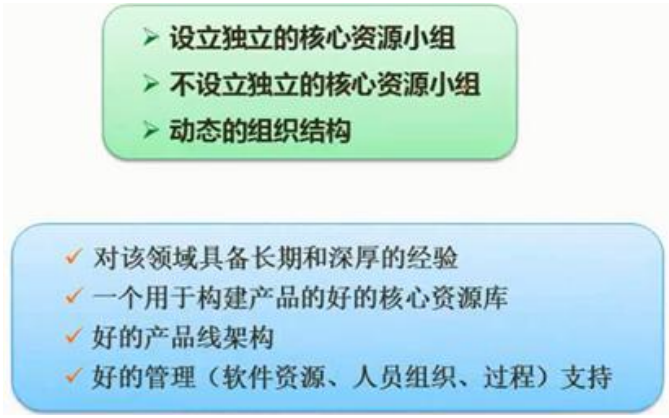



- SEI 模型




- 三生命周期模型

### 3.9.2 组织结构




### 3.9.3 建立方式

| | 演化方式 | 革命方式 |
|---|---|---|
| 基于现有产品 | 基于现有产品架构设计产品线的架构，经演化现有构件，开发产品线构件 | 核心资源的开发基于现有产品集的需求和可预测的、将来需求的超集 |
| 全新产品线 | 产品线核心资源随产品新成员的需求而演化 | 开发满足所有预期产品线成员的需求的核心资源 |

### 3.10 软件架构视图

表 9-10　系统的架构视图

| 组　别 | 架构风格 | 说　明 | 应用于 |
|---|---|---|---|
| 模块<br>视图类型 | 分解 | 大模块分解为小模块，小到容易理解 | 资源分配、项目结构化和规划；信息隐蔽、封装；配置控制 |
| | 使用 | 一个单元的正确性依赖于另一个单元的正确性（如版本） | 设计子集；设计扩展（增量开发） |
| | 分层 | 上层使用下层的服务；实现隐藏细节的抽象 | 增量式开发；基于“虚拟机”上的可移植性 |
| | 类或泛化 | “继承自”或“是一个实例”；共享访问方法 | 面向对象的设计（使用公共模板） |
| 构件-连接器<br>视图类型 | 客户机-服务器 | 构件是客户机和服务器，连接件是协议及共享消息 | 分布式操作；关注点分离（支持可修改性）；负载均衡 |
| | 进程或通信进程 | 通过通信、同步或排除操作形成进程或线程之间的关联 | 调度分析；性能分析 |
| | 并发 | 在相同的“逻辑线程”上运行 | 确定资源争用；分析线程 |
| | 共享数据 | 运行时产生数据、使用数据（共享数据存储库） | 性能；数据完整理；可修改性 |
| 分配<br>视图类型 | 部署 | 软件功能分配给软件（进程）、硬件（处理器）和通信路径 | 性能、可用性、安全性说明。尤其在分布式或并行系统中 |
| | 实现 | 模块映射到开发活动中 | 配置控制、集成、测试活动 |
| | 工作分配 | 将责任分配到适当的开发小组，特别是公共部分不是每个人去实现 | 项目管理、管理通用性，最好的专业技术安排 |

# 四 设计模式

详见： http://c.biancheng.net/view/1366.html

概念：是一套被反复使用、多数人知道的、经过分类编目的、代码设计经验的总结。

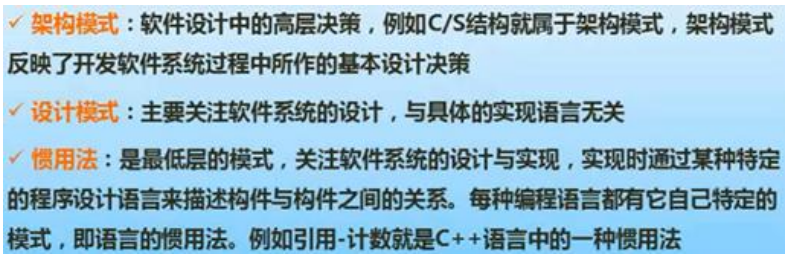
## 4.1 设计原则

- 开放-封闭原则：对拓展开放，对修改封闭. 当应用的需求改变时，在不修改软件实体的源代码或者二进制代码的前提下，可以扩展模块的功能，使其满足新的需求。实现方法：接口+实现类. 例;windows 桌面主题
- 里氏替换：子类可以扩展父类的功能，但不能(要)改变父类原有的功能(方法)。子类可以替代父类。尽量将一些需要扩展的类或者存在变化的类设计为抽象类或者接口，并将其作为基类，在程序中尽量使用基类对象进行编程。
- 依赖倒置：要依赖于抽象，而不是具体实现；针对接口编程，不要面向实现编程.
- 单一职责；设计目的单一的类
- 接口隔离原则：使用多个专门的接口比使用单一的总接口要好
- 迪米特(最少知识法则)：如果两个软件实体无须直接通信，那么就不应当发生直接的相互调用，可以通过第三方转发该调用。一个对象应当对其它对象有尽可能少的了解。
- 组合重用原则：优先使用组合或者聚合关系复用，少用继承关系复用。

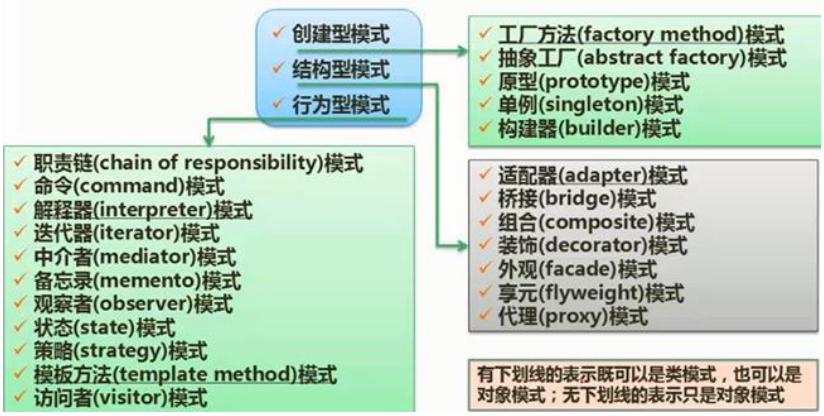
## 4.2 设计模式概念

组成：模式名称/问题/解决方案/效果

几个相关概念：架构模式/设计模式/惯用法




## 4.3 分类



| 设计模式名称 | 简要说明 | 速记关键字 |
| --- | --- | --- |
| Adapter<br>适配器模式 | 将一个类的接口转换成用户希望得到的另一种接口。它使原本不相容的接口得以协同工作 | 转换接口 |
| Bridge<br>桥接模式 | 将类的抽象部分和它的实现部分分离开来，使它们可以独立地变化 | 继承树拆分 |
| Composite<br>组合模式 | 将对象组合成树型结构以表示“整体-部分”的层次结构，使得用户对单个对象和组合对象的使用具有一致性 | 树形目录结构 |
| Decorator<br>装饰模式 | 动态地给一个对象添加一些额外的职责。它提供了用子类扩展功能的一个灵活的替代，比派生一个子类更加灵活 | 附加职责 |
| Façade<br>外观模式 | 定义一个高层接口，为子系统中的一组接口提供一个一致的外观，从而简化了该子系统的使用 | 对外统一接口 |
| Flyweight<br>享元模式 | 提供支持大量细粒度对象共享的有效方法 | |
| Proxy<br>代理模式 | 为其他对象提供一种代理以控制这个对象的访问 | |

**创建型模式：*构原工单***

用于描述“怎样创建对象”，它的主要特点是“将对象的创建与使用分离”。

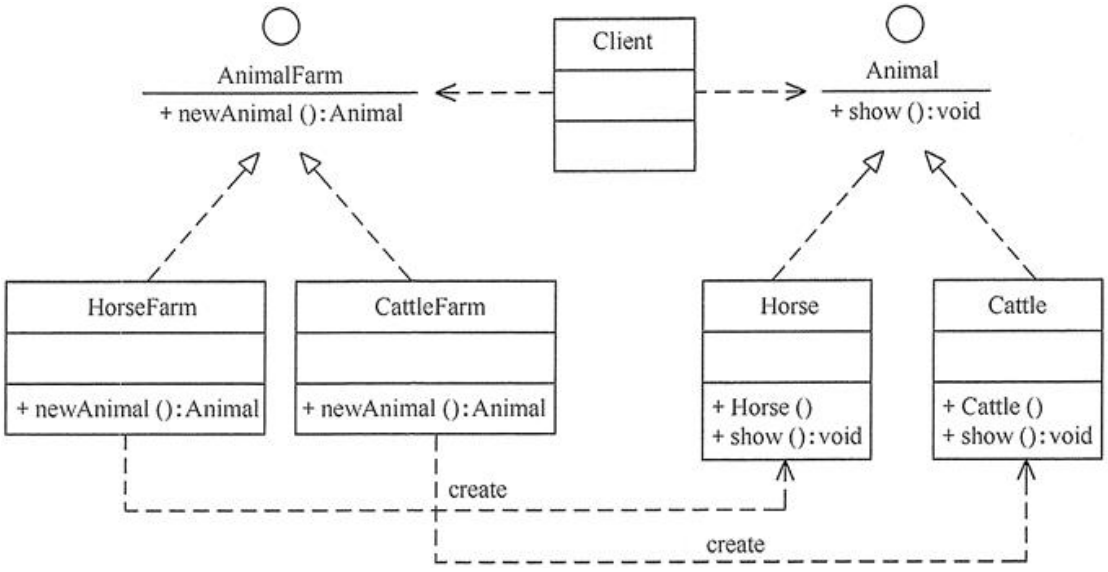
**结构性模式：*桥适享外代组装***

用于描述如何将类或对象按某种布局组成更大的结构.

**行为型模式：*ELSE***

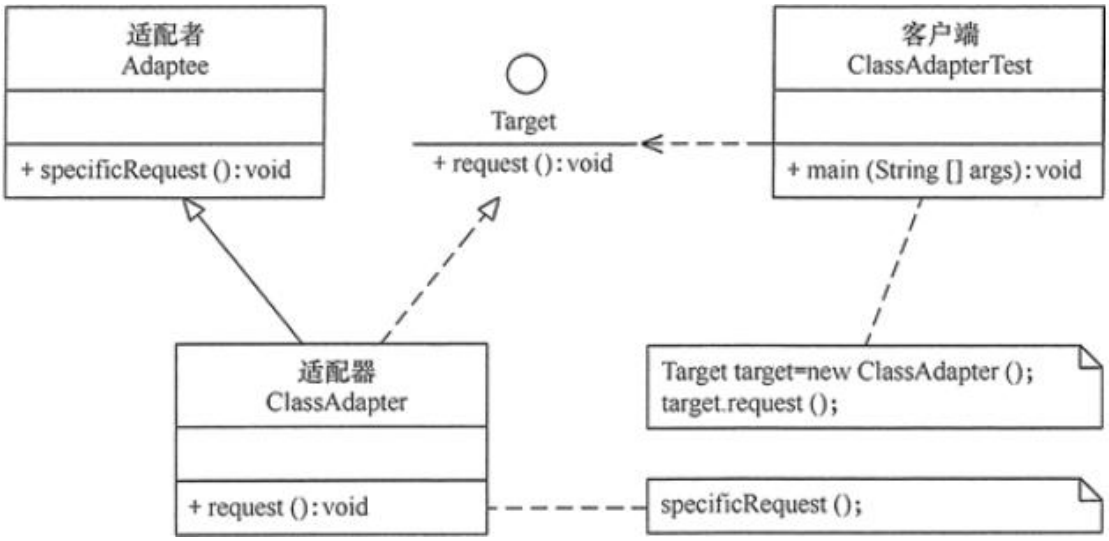
用于描述类或对象之间怎样相互协作共同完成单个对象都无法单独完成的任务，以及怎样分配职责。

- **单例模式**：一个类只能生成一个实例，且该类能自行创建这个实例.
- **工厂方法**：实现软件对象的生产和使用相分离。包括 4 个角色：1 抽象工厂 abstract factory；2 具体工厂 concrete factory；3 抽象产品 product；4 具体产品 concreteProduct.

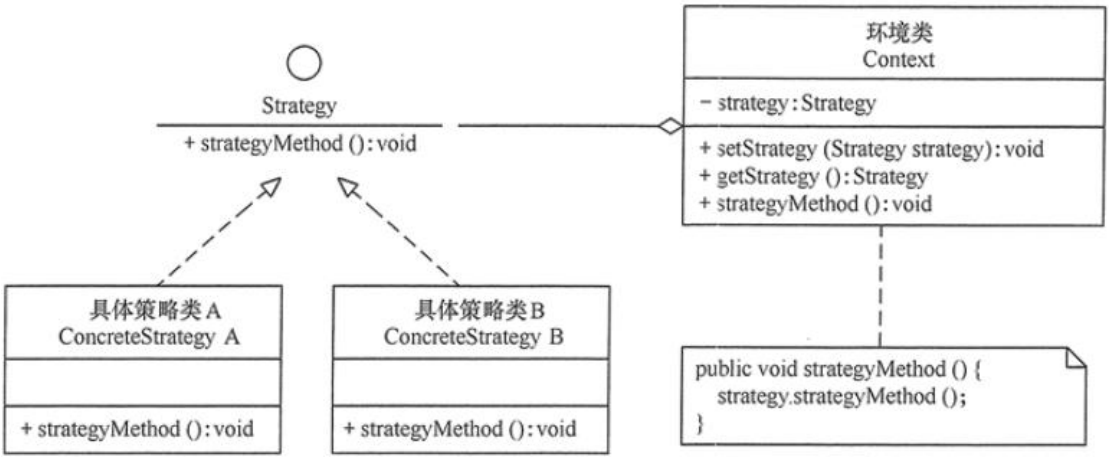



- **抽象工厂方法**：抽象工厂模式是工厂方法模式的升级版本，工厂方法模式只生产一个等级的产品，而抽象工厂模式可生产多个等级的产品.（如异构数据库平台，oracle/mysql 等数据库，建立多种数据库的抽象工厂，当指定具体工厂为 oracle 时，创建出来的数据库连接、数据集等一系列对象都符合 oracle 规范）

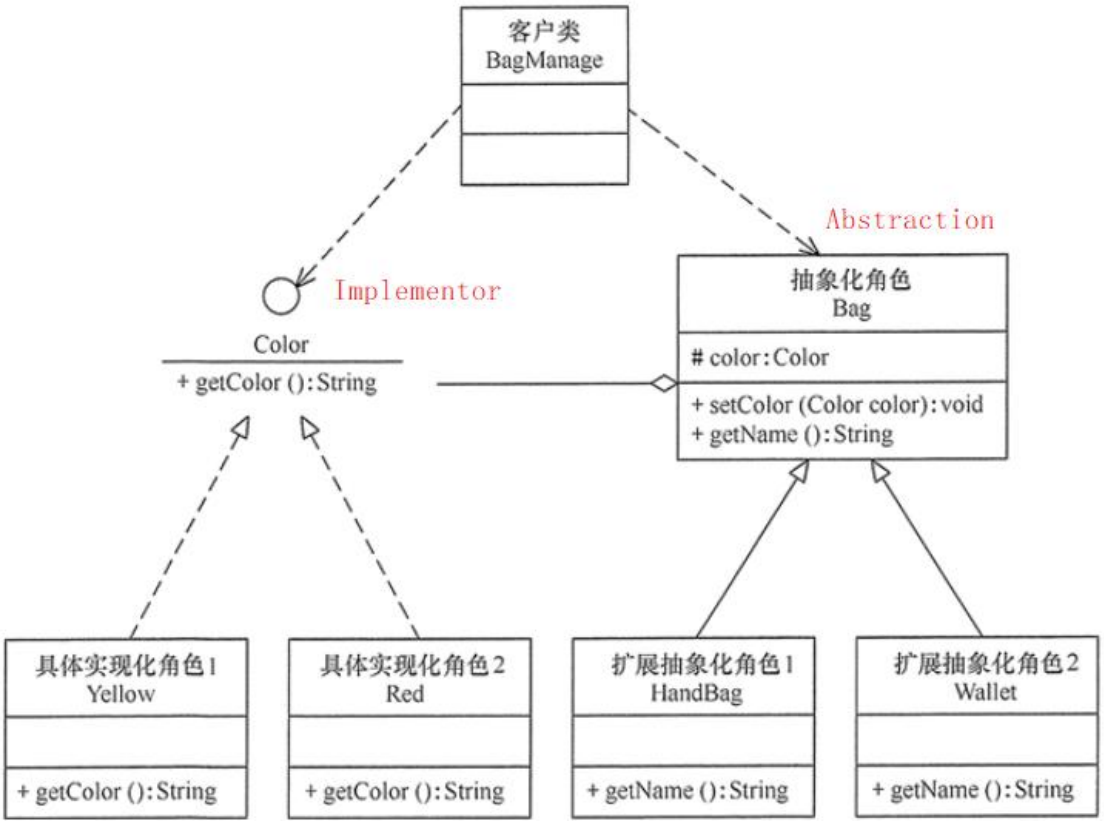
- **组合模式 Composite**：将对象组合成树形结构以表示“部分-整体”的层次结构。它使得客户对单个对象和复合对象的使用具有一致性。往往有多层关系，树—树枝—树叶，树枝 composite 上可以有树叶，树枝上也可以有树枝，树枝和树叶都属于 component，composite 可以有好几个 component
- **装饰者模式 decorator**:动态地给一个对象添加一些额外的职责。就扩展功能而言，它比生成子类方式更为灵活。
- **责任链模式**：为了避免请求发送者与多个请求处理者耦合在一起,将所有请求的处理者通过前一对象记住其下一个对象的引用而连成一条链；当有请求发生时，可将请求沿着这条链传递，直到有对象处理它为止. 应用如拦截器/过滤器.
- **状态模式**：将每一个条件分支放入一个独立的类中,这样就可以根据对象自身的情况将对象的状态作为一个对象，这一对象可以不依赖于其他对象而独立变化.
- **外观模式 facade**：要求外部与一个子系统的通信必须通过一个统一的外观对象进行，为子系统中的一组接口提供一个一致的界面，外观模式定义了一个高层接口，这个接口使得这一子系统更加容易使用。
- **模板方法模式**：定义一个操作中的算法骨架，而将算法的一些步骤延迟到子类中，使得子类可以不改变该算法结构的情况下重定义该算法的某些特定步骤。它是一种类行为型模式。固定了流程但没有固定里面的内容.
- **解释器模式 Interpreter**：描述了如何为语言定义一个文法，如何在该语言中表示一个句子，以及如何解释这些句子，这里的“语言”是使用规定格式和语法的代码。解释器模式主要用在编译器中，在应用系统开发中很少用到.
- **中介者模式 mediator**：通过一个中介对象来封装一系列的对象交互。中介者使得各对象不需要显式地相互引用，从而使其耦合松散，而且可以独立地改变它们之间的交互。中介者对象的存在保证了对象结构上的稳定，也就是说，系统的结构不会因为新对象的引入带来大量的修改工作。
- **迭代器模式 iterator**：提供了一种方法来访问聚合对象，而不用暴露这个对象的内部表示。迭代器模式支持以不同的方式遍历一个聚合对象，复杂的聚合可用多种方法来进行遍历；允许在同一个聚合上可以有多个遍历，每个迭代器保持它自己的遍历状态，因此，可以同时进行多个遍历操作。
- **访问者模式 visitor**：将作用于某种数据结构中的各元素的操作分离出来封装成独立的类，使其在不改变数据结构的前提下可以添加作用于这些元素的新的操作，为数据结构中的每个元素提供多种访问方式。例如：医院医生开的处方单中包含多种药元素，查看它的划价员和药房工作人员对它的处理方式也不同,划价员根据处方单上面的药品名和数量进行划价，药房工作人员根据处方单的内容进行抓药。
- **观察者模式 observer**：指多个对象间存在一对多的依赖关系，当一个对象的状态发生改变时，所有依赖于它的对象都得到通知并被自动更新。
- **适配器模式 adapter：将一个类的接口转换成客户希望的另一个接口。**包括 1 目标接口（客户希望的接口）、2 适配者类（被访问的现存组件库中的组件接口）、3 适配器类（转换器）。



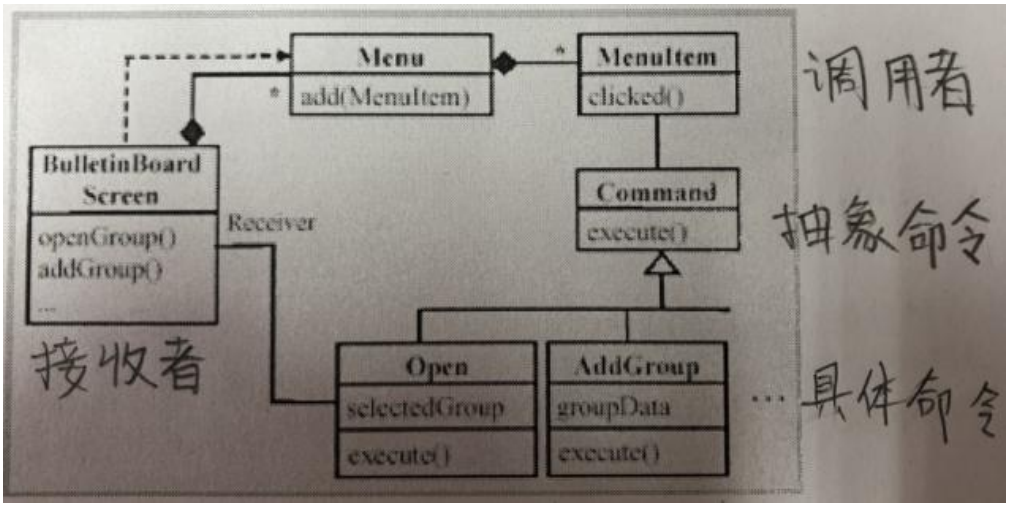

- **策略模式 strategy**：定义一系列的算法，并将这组算法封装到一系列的策略类里面，作为一个抽象策略类的子类。再定义一个环境（Context）类：持有一个策略类的引用，最终给客户端调用。

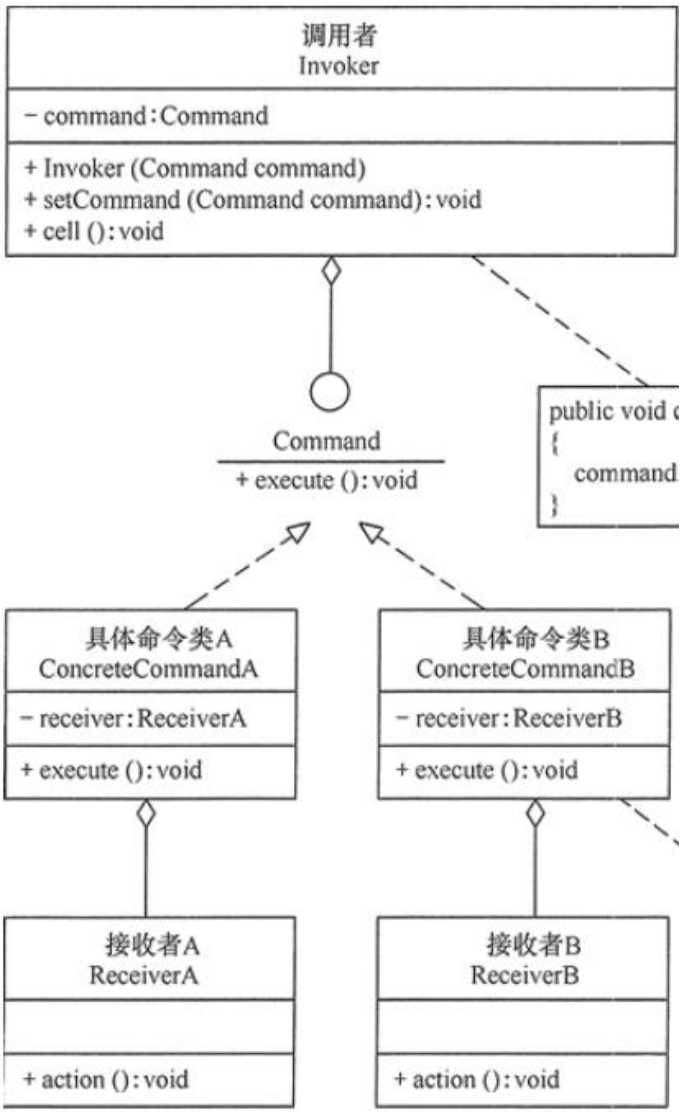



- **桥接模式 bridge**：将抽象部分 abstraction 与它的实现部分 implementor 分离，使它们都可以独立地变化. 它是用组合关系代替继承关系来实现。如黑色/红色箱子，大/小箱子，M*N 种搭配。




- **命令模式 Command**：将一个请求封装为一个对象，使发出请求的责任和执行请求的责任分割开。将系统中的相关操作抽象成命令，使调用者与实现者相互分离。包括 4 个角色：1 抽象命令类 Command/2 具体命令类 ConcreteCommand/3 接收者 receiver/4 调用者 invoker. 支持命令的撤销（Undo）操作和恢复（Redo）操作.






# 四. 数据库系统

### 4.1 数据库管理系统的类型

有多种分类方式，常见的 DBMS 按数据模型划分，包括：关系型 DBMS、文档型 DBMS、键值型 DBMS、对象型 DBMS 等

### 4.2 数据库模式与范式

#### 4.2.1 数据库模式

用户级 / 外模式 / 用户视图

外部记录 ，一个数据库可以有多个外模式/ 一个应用程序只能使用一个外模式

概念级 / 概念模式(模式、逻辑模式) / DBA 视图

数据库管理员 DBA 视图 ，一个数据库只有一个概念模式

物理级 / 内模式 / 内部视图

内部记录,不是真正的物理存储,而是最接近物理存储的一级，一个数据库只有一

个内模式

### 4.2.2 数据模型

概念数据模型 (ER 实体联系模型)

基本数据模型 (结构数据模型)

层次模型 / 网状模型 / 关系模型 / 面向对象模型

### 4.2.3 关系代数

笛卡尔乘积 X

自然连接. 智能匹配,自动去重字段

### 4.2.4 数据规范化

1NF 字段都是单一属性的，不可再分

2NF 要求实体的属性完全依赖于主关键字。所谓完全依赖是指不能存在仅依赖主关键字一部分的属性

3NF 任何非主属性不依赖于其它非主属性（在 2NF 基础上消除传递依赖）

BCNF 任何非主属性不传递依赖于码

4NF 略

### 4.2.5 反规范化

对完全规范的数据库查询，通常需要更多的连接操作，从而影响查询速度。因此，有时为了提高某些**查询**或应用的性能而破坏规范规则，即反规范化（非规范化处理）。主要方法包括:

（1）增加冗余列

（2）增加派生列

（3）重新组表

（4）分割表（分表）. 水平分割/垂直分割。

## 4.3 数据库设计

- 步骤

a 需求分析

通过调查研究, 了解用户的数据和处理需求, 并按一定格式整理形成需求说明书.(还有数据字典和数据流程图)

b 概念设计

在需求分析阶段产生的需求说明书的基础上, 按照特定的方法将它们抽象为一个不依赖于任何 DBMS 的**数据模型**, 即概念模型. (如 E-R 模型)

c 逻辑设计

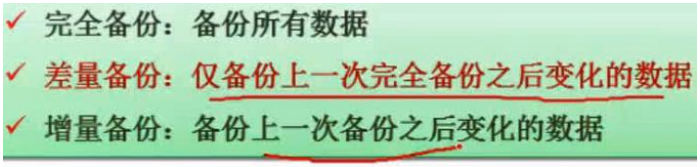
将概念模型转化为某个特定的 DBMS 上的逻辑模型(如关系型数据库、对象型数据库), 并对所设计的逻辑模型进行优化.(发生在概要设计阶段；设计表包括所有细节---表名、字段、主外键等)

d 物理设计

对给定的逻辑模型选取一个最适合应用环境的物理结构，以确定数据库在物理设备上的存储结构和存取方法.

## 4.4 数据库备份与恢复

### 4.4.1 备份




注意差量备份和增量备份的区别

物理备份(数据文件)，逻辑备份(数据库自带工具)

### 4.4.2 恢复

已经提交的事务不代表对数据库的更新已经完成;

事务未完成不代表所有的写操作都还没完成;

## 4.5 事务管理

- 事务特性:

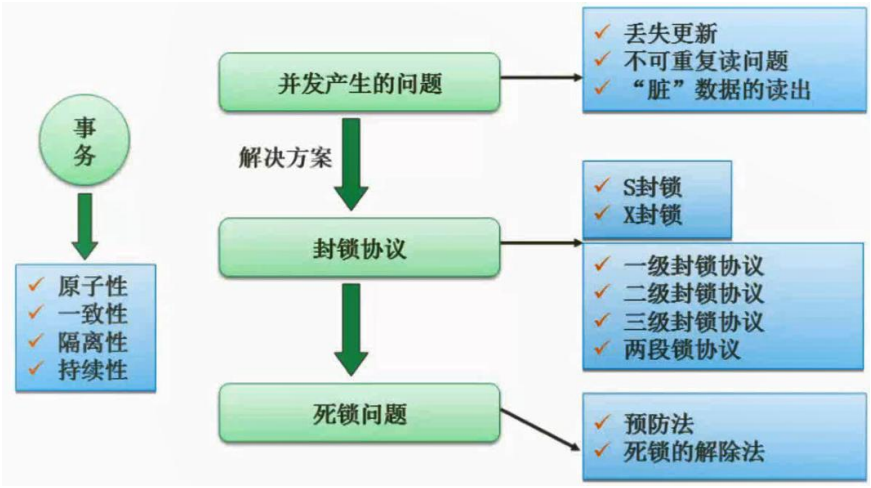
  原子性

  一致性: 如银行转账总额一致

  隔离性

  永久性

- **数据库并发控制**




- 并发带来的问题

  丢失更新: 两个事务都去写，结果只写了一边

  不可重复读 ： 读了两次，前后不一样

  读’脏’数据 ： 改了之后又撤销，在中间被读了.

- 封锁协议 *https://www.cnblogs.com/zzlback/p/12614790.html*

  X 写锁：只能自己读和改，其它事务不能读不能改（排他）

  S 读锁：只能读，不能改（共享），其它事务可以一起读

  某数据加了其中一种，就不能加另外一种，但是 S 锁可以多次加.

| | | |
|---|---|---|
| 一级封锁协议 | 修改数据之前加 X<br>事务结束释放 | 解决丢失修改 |
| 二级封锁协议 | X 基础上<br>读数据 R 之前加 S<br>读完释放 | 解决丢失修改、读脏数据 |
| 三级封锁协议 | X 基础上<br>读数据 R 之前加 S<br>事务结束释放 | 解决了丢失修改、读脏数据、不可重复读 |
| 两段封锁协议 | 可串行化，但有可能出现死锁 | 解决死锁：<br>1 预防法<br>2 死锁解除法 |

二级和三级相比，三级是事务结束之后才释放的，更晚，所有解决了更多问题—不可重复读

### 4.6 分布式数据库系统

概念：分布式数据库是由一组数据组成的，这组数据分布在计算机网络的不同计算机上，网络中的每个结点具有独立处理的能力，成为场地自治，它可以执行局部应用，同时，每个结点也能通过网络通信子系统执行全局应用。

如何提高性能：

1、 采用数据分片技术，提高访问的局部性，提升系统性能。

2、 采用查询优化技术（包括：全局查询树的变换、副本的选择与多副本的更新策略、查询树的分解、半连接与直接连接）提高查询速度。

3、 读写分离技术

分区：分区并不是生成新的数据表，而是将表的数据均衡分摊到不同的硬盘，系统或是不同服务器存储介质中，实际上还是一张表。分区可以做到将表的数据均衡到不同的地方，提高数据检索的效率，降低数据库的频繁 IO 压力值。分为水平分区和垂直分区。

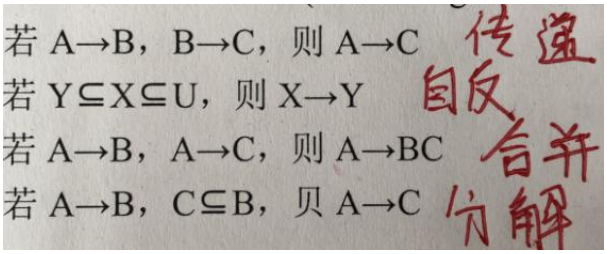
分表：把一张表按一定的规则分解成 N 个具有独立存储空间的实体表。分为水平分割和垂直分割。

**分布式数据库缓存**：是在内存中管理数据并提供数据的一致性保障，采用数据复制技术实现高可用性，具有较优的扩展性和性能组合. 这种数据存储机制，实现了更短的响应时间，同时极大地降低数据库的事务处理负载，极好地解决了大流量情况下数据库服务器和 web 服

务器之间的瓶颈.

**杂**

- armstrong 公理

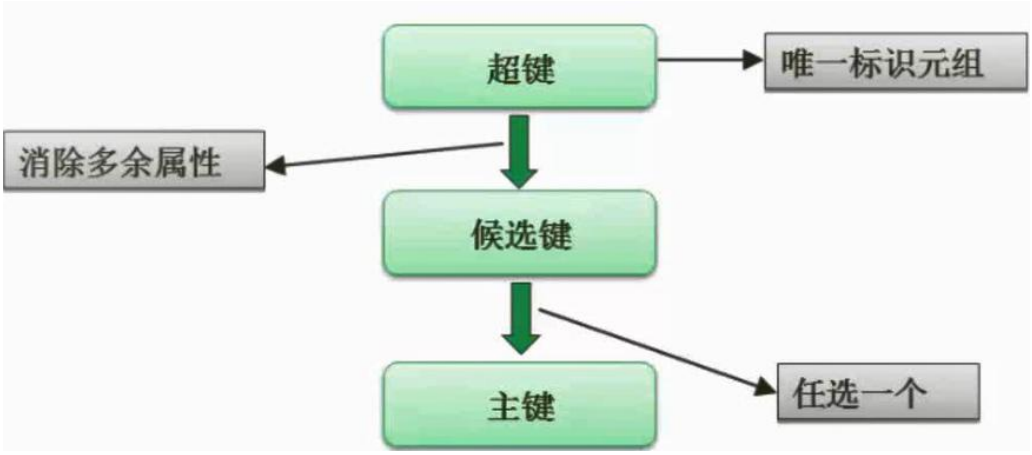



- **几个键的概念**

  超键，可能有多余的属性

  候选键，没有多余的属性，可以有多对候选键

  主键：从候选键里面挑出一个最为最终的主键.




**函数依赖、无损**

函数依赖：保持函数依赖

无损：通过自然连接或投影**可以还原**出原来的(U 全表)关系模式

具体算例可见 真题--第三章 3.2.4 2009 年题 5 或者是视频教程

一分为 2 特例公式:

定理：如果R的分解为ρ＝{ $R_1$，$R_2$ }，F为R所满足的函数依赖集合，分解ρ具有无损联接性的充分必要条件是：

$R_1 \cap R_2 \rightarrow (R_1 - R_2)$

或 $R_1 \cap R_2 \rightarrow (R_2 - R_1)$

例：设R＝ABC，F＝{ A→B }，则$\rho_1$＝{ $R_1$（AB），$R_2$（AC）} 、$\rho_2$＝{ $R_1$（AB），$R_3$（BC）}是不是无损分解？

$R_1 \cap R_2 = A$　　$R_1 - R_2 = B$　　$R_2 - R_1 = C$

A →B 或 A →C。

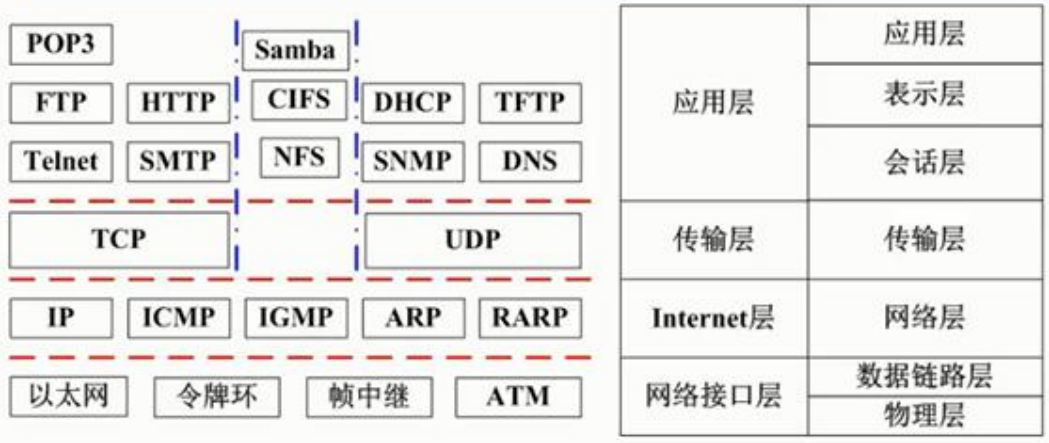
$R_1 \cap R_3 = B$　　$R_1 - R_3 = A$　　$R_3 - R_1 = C$

B →A 或 B →C。

# 五. 计算机网络

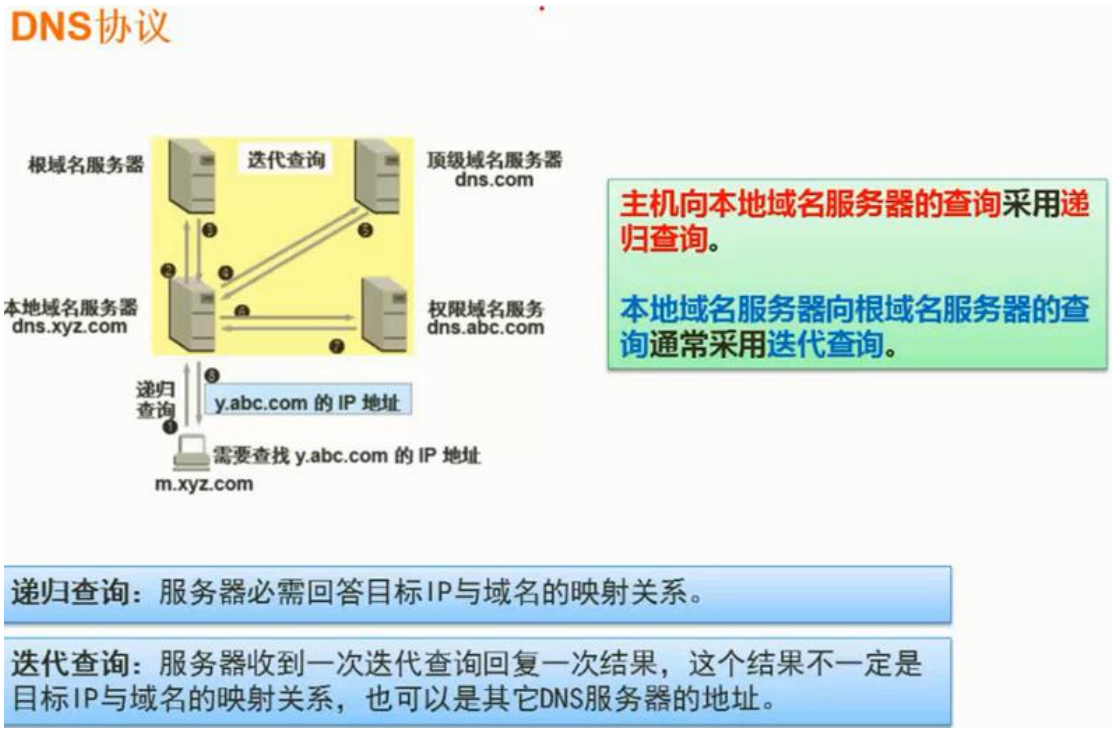
## 5.1 网络互联模型与协议

- ✓ TCP/IP协议：Internet，可扩展，可靠，应用最广，牺牲速度和效率
- ✓ IPX/SPX协议：NOVELL，路由，大型企业网
- ✓ NETBEUI 协议：IBM，非路由，快速




IPv6 有 3 种地址类型，分别是单播、多播（也称作组播）、泛播（也称作任意播）

.




递归查询负责到底,负担大; 迭代查询转移负责人,负担轻

PTR(Point Record) : 通过 IP 查询域名.

根域名服务器是最高层次的域名服务器

## 5.2 网络工程

包括网络规划/网络设计/网络实施三部分.

### a 网络规划

网络需求分析

可行性分析

对现有网络的分析与描述

### b 网络设计

**网络逻辑结构设计**，根据需求规范和通信规范，选择一种比较适宜的网络逻辑结构，并基于该逻辑结构实施后续的资源分配规划、安全规划等内容，输出包括:

① 逻辑网络设计图;

② IP 地址方案;

③ 安全方案;

④ 具体的软件、硬件、广域网连接设备和基本的服务;

⑤ 雇佣和培训新网络员工的具体说明;

⑥ 初步对软件、硬件、服务、网络雇佣员工和培训的费用估计

**网络物理结构设计**，通过对设备的具体物理分布、运行环境等的确定，确保网络的物理

连接符合逻辑连接的要求. 输出包括:

① 物理网络图和布线方案;

② 设备和部件的详细列表清单;

③ 软件、硬件和安装费用的估计;

④ 安装日程表，用以详细说明实际和服务中断的时间及期限;

⑤ 安装后的测试计划;

⑥ 用户培训计划

**分层设计**，分为接入层/汇聚层/核心层

接入层的目的是允许终端用户连接到网络;

汇聚层是核心层和接入层的分界面，完成网络访问策略控制、数据包处理、过滤、寻址，以及其他数据处理的任务。

核心层的主要目的在于通过高速转发通信，提供优化、可靠的骨干传输结构，因此，核心层交换机应拥有更高的可靠性，性能和吞吐量。核心层为网络提供了骨干组件或高速交换组件，在纯粹的分层设计中，核心层只完成数据交换的特殊任务。

**c 网络实施**

在网络设计的基础上进行设备的购买、安装、调试和系统切换工作，包括:

①工程实施计划

②网络设备到货验收

③设备安装

④系统测试

⑤系统试运行

⑥用户培训

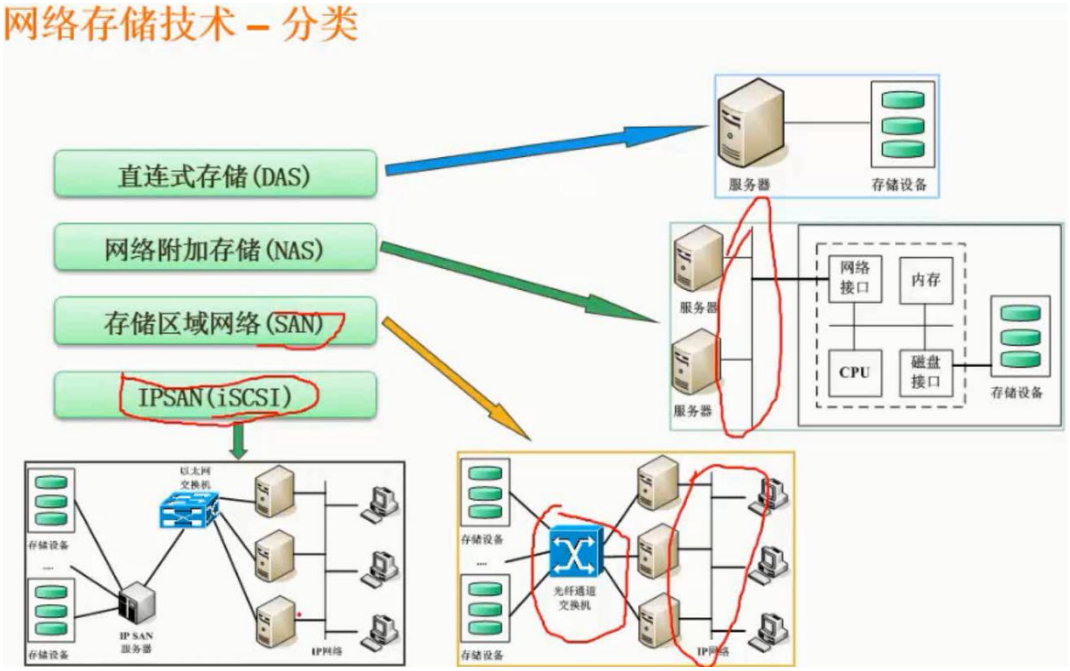
7.系统转换

## 5.3 网络存储技术

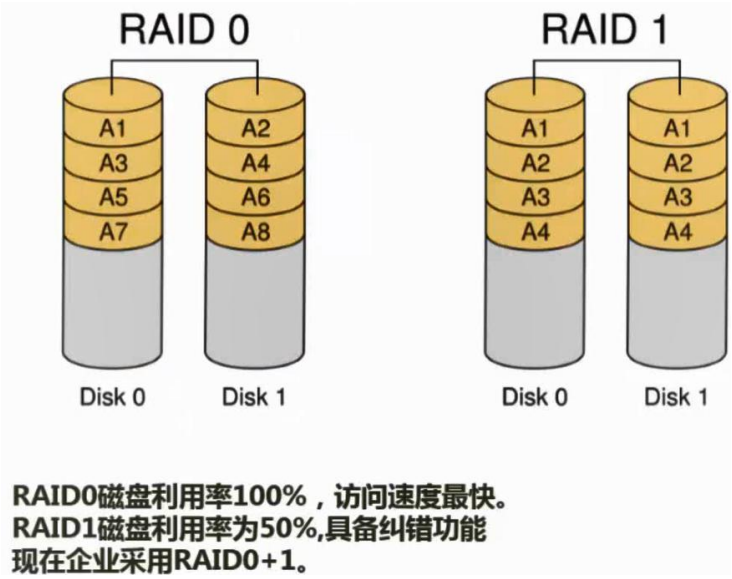
DAS (direct attached storage)

NAS (network attached storage)

SAN (storage area network)



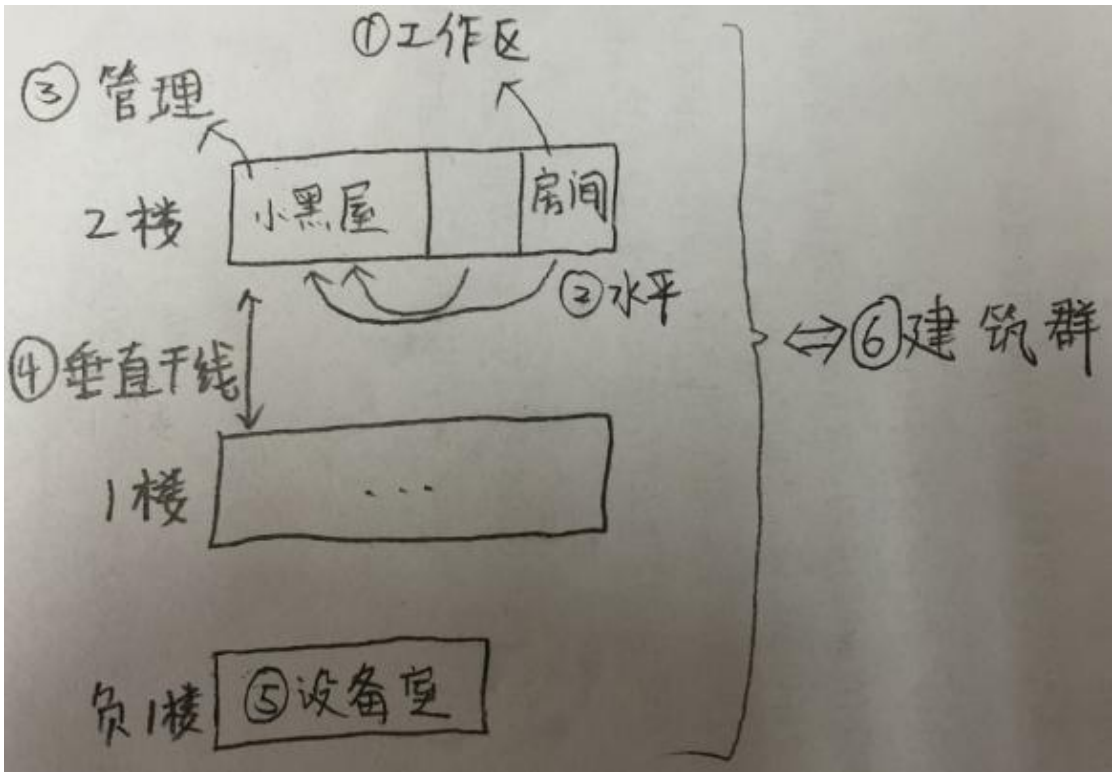

磁盘阵列 RAID




- ✓ Raid 0(条块化)：性能最高，并行处理，无冗余，损坏无法恢复
- ✓ Raid1(镜像结构)：可用性，可修复性好，仅有50%利用率
- ✓ Raid0+1(Raid10)：Raid0与Raid1长处的结合，高效也可靠
- ✓ Raid3(奇偶校验并行传送)：N+1模式，有固定的校验盘，坏一个盘可恢复
- ✓ Raid5(分布式奇偶校验的独立磁盘)：N+1模式，无固定的校验盘，坏一个盘可恢复
- ✓ Raid6(两种存储的奇偶校验)：N+2模式，无固定的校验盘，坏两个盘可恢复

## 5.4 综合布线

6 个子系统




# 六 系统性能评价

## 6.1 主要性能指标

计算机性能评价指标有：时钟频率（主频=倍频*外频）；运算速度；运算精度；内存的存储容量；存储器的存取周期；数据处理速率；吞吐率；各种响应时间；各种利用率；平均故障响应时间；兼容性；可扩充性；性能价格比。

*MIPS（百万条指令/秒）和 MFLOPS（百万次浮点运算/秒）。MIPS 用于描述计算机的定点运算能力；MFLOPS 则用来表示计算机的浮点运算能力。*

数据库管理系统的主要性能指标包括数据库本身和管理系统两部分，有：数据库的大小、数据库中表的数量、单个表的大小、表中允许的记录（行）数量、单个记录（行）的大小、表上所允许的索引数量、数据库所允许的索引数量、最大并发事务处理能力、负载均衡能力、最大连接数等等。

网络的性能指标有：设备级性能指标；网络级性能指标；应用级性能指标；用户级性能指标；吞吐量。

操作系统的性能指标有：系统的可靠性、系统的吞吐率（量）、系统响应时间、系统资源利用率、可移植性。

Web 服务器的性能指标有：最大并发连接数、响应延迟、吞吐量。评估方法有*基准测试、压力测试和可靠性测试*。

### 6.2 性能设计

**a 阿姆达尔**

$$加速比=\frac{不使用增强部件时完成整个任务的时间}{使用增强部门时完成整个任务的时间}$$

**b 负载均衡**

负载均衡是由多台服务器以对称的方式组成一个服务器集合，每台服务器都具有等价的地位，都可以单独对外提供服务而无须其他服务器的辅助。通过某种负载分担技术，将外部发送来的请求均匀地分配到对称结构中的某一台服务器上，而接收到请求的服务器独立地回应客户的请求。

## 八 开发方法

### 8.1 软件生命周期

可行性研究与计划、需求分析、概要设计(外部设计)、详细设计(内部设计)、实现、集成测试、确认测试、使用和维护

概要设计：外部设计又称为概要设计，其主要职能是设计各个部分的功能、接口、相互如何关联。

详细设计：内部设计又称为详细设计，其主要职能是设计具体一个模块的实现。

### 8.2 系统开发方法

结构化法

原型法

面向对象方法

面向服务方法

### 8.3 软件开发模型

瀑布模型. 软件计划/需求分析/软件设计/程序编码/软件测试/运行维护. 瀑布模型是面向文档的软件开发模型。

瀑布 V 模型. 在瀑布模型的基础上做了修改，更强调**测试**.

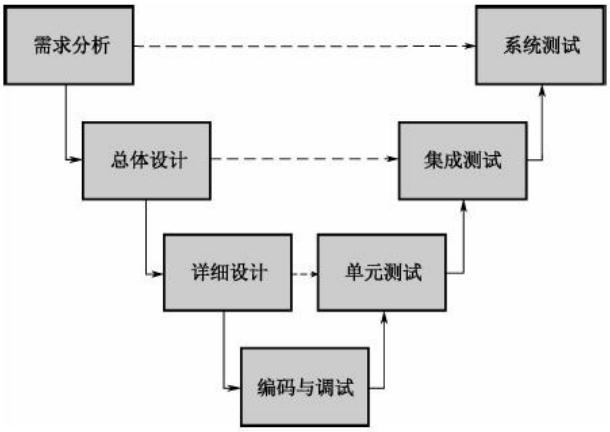



图 6-2　瀑布 V 模型

演化模型. 演化模型可以看做若干次瀑布模型的迭代，当完成一个瀑布模型后，重新进入下一个迭代周期，软件在这样的迭代过程中得以演化、完善。根据不同的迭代特点，演化模型可以演变为螺旋模型、增量模型和原型法开发。

螺旋模型. 融合了多种模型(将瀑布模型、**原型**、和演化模型结合起来)；引入了**风险**分析. 将软件开发分为多个阶段，每个阶段分为 4 步：目标设定/风险分析/开发和有效性验证/评审.

增量模型. (原型+瀑布模型). 一次发布一点功能，发布多次. 策略：1 增量发布；2 原型法

原型模型. 应用于需求分析阶段，需求不明确，构建简易系统。原型开发分两大类：**快速原型法**(又称抛弃式原型法 )和**演化式原型法**。其中快速原型法是快速开发出一个原型，利用该原型获取用户需求，然后将该原型抛弃。而演化式原型法是将原型逐步进化为最终的目标系统。

构件组装模型(CBSD). 搭积木. 步骤：需求分析/软件架构(体系结构) 设计/构件库建立/应用软件构建/测试与发布. 标准：CORBA/COM,DCOM,DOM+/EJB.

失配是指在软件复用的过程中，由于待复用构件对最终系统的体系结构和环境的假设(assumption)与实际状况不同而导致的冲突。在构件组装阶段失配问题主要包括：

(1)由构件引起的失配，包括由于系统对构件基础设施、构件控制模型和构件数据模型的假设存在冲突引起的失配；

(2)由连接子引起的失配，包括由于系统对构件交互协议、连接子数据模型的假设存在冲突引起的失配；

(3)由于系统成分对全局体系结构的假设存在冲突引起的失配等。要解决失配问题，首先需要检测出失配问题，并在此基础上通过适当的手段消除检测出的失配问题。

喷泉模型：面向对象.

RAD：快速开发模型. VB/DELPHi. 结合了瀑布模型和构件组装模型. 通过使用基于构件的开发方法获得快速开发，当系统模块化程度较高时，适合用该模型.

统一过程(UP/RUP):

特点: 1 用例驱动/2 以架构为中心/3 迭代和增量.

步骤: “粗细构交”

- **初始**. 确定项目范围和边界/识别系统的关键用例/展示系统的候选架构/估计项目费用和时间/评估项目风险
- **细化**. 分析系统问题领域/**建立软件架构基础**/淘汰最高风险元素
- **构建**. 开发剩余的构件/构件组装与测试
- **交付**. 进行 B 测试/ 制作发布版本/用户文档定稿/确认新系统/培训、调整产品

敏捷方法

敏捷方法以原型开发思想为基础，采用迭代增量式开发，发行版本小型化，比较适合需求变化较大或者开发前期对需求不是很清晰的项目。面向对象。


## 8.4 几个大概念

### a 逆向工程

逆向工程就是分析已有的程序，寻求比源代码**更高级的抽象表现形式**。一般认为，凡是在软件生命周期内将软件某种形式的描述转换成**更为抽象**形式的活动都可称为逆向工程。

逆向工程导出的信息可分为如下 4 个抽象层次。

实现级：包括程序的抽象语法树、符号表等信息。

结构级：包括反映程序分量之间相互依赖关系的信息，例如调用图、结构图等。

功能级：包括反映程序段功能及程序段之间关系的信息。

领域级：包括反映程序分量或程序与应用领域概念之间对应关系的信息。

### b 再工程

再工程（re-engineering），也称修复和改造工程，它是在逆向工程所获信息的基础上修改或重构已有的系统，产生系统的一个新版本。

c 重构

重构（restructuring），指在同一抽象级别上转换系统描述形式.

d 软件重用

软件重用（软件复用）是使用已有的软件产品（如设计、代码和文档等）来开发新的软件系统的过程。

**水平式重用**是重用*不同应用领域*中的软件元素，如 标准函数库。

**垂直式重用**是在一类具有较多公共性的应用领域之间重用软件构件。

### 8.6 基于架构的软件设计 ABSD

- 基础

  （1）**功能的分解**。在功能分解中，ABSD 方法使用已有的基于模块的内聚和耦合技术。

  （2）通过**选择架构风格**来实现质量和业务需求。

  （3）**软件模板的使用**。软件模板利用了一些软件系统的结构。
- 采用视角与视图来描述软件架构，采用用例与质量场景来描述需求.
- 架构文档化的输出结果为：架构规格说明书 和 架构质量说明书

# 十 系统测试评审

### 10.1 测试阶段

#### 10.1.1 单元测试(模块测试)

单元测试的技术依据是软件详细设计说明书。

驱动模块用来调用被测模块；桩模块用来模拟被测模块所调用的子模块。顶层模块测试时不需要驱动模块，底层模块测试时不需要桩模块。

#### 10.1.2 集成测试

集成测试的技术依据是软件概要设计文档。

集成测试的目的是检查模块之间，以及模块和已集成的软件之间的接口关系，并验证已集成的软件是否符合设计要求。

#### 10.1.3 系统测试

系统测试的技术依据是用户需求或开发合同。

具体测试内容有恢复测试（恢复测试监测系统的容错能力）、安全性测试、压力测试、性能测试、可靠性测试、可用性测试、可维护性测试和安装测试。

**确认测试**：有用户参与，主要依据软件需求说明书检查软件的功能、性能及其他特征是否与用户的需求一致；以及软件配置复查。如 Alpha/Beta

**验收测试**：定制产品的客户

## 10.2 测试类型

- 动态测试

  包括白盒/黑盒/灰盒。

  需要运行被测试系统，并设置探针，向代码生成的可执行文件中插入检测代码，用于软件的覆盖分析和性能分析，也可用于软件的模拟、建模、仿真测试和变异测试等。

- 静态测试

  包括桌前检查/ 代码走查/ 代码审查。

  对代码进行语法扫描，找到不符合编码规范的地方，根据某种质量模型评价代码的质量，生成系统的调用关系图等。它直接对代码进行分析，不需要运行代码，也不需要对代码编译链接和生成可执行文件，静态测试工具可用于对软件需求、结构设计、详细设计和代码进行评审、走审和审查，也可用于对软件的复杂度分析、数据流分析、控制流分析和接口分析提供支持。

- 黑盒(功能)测试

  等价类划分：等价类就是某个输入域的集合，对每一个输入条件确定若干个有效等价类和若干个无效等价类，分别设计覆盖有效等价类和无效等价类的测试用例。

  边界值分析：边界值分析通过选择等价类边界作为测试用例，不仅重视输入条件边界，而且也必须考虑输出域边界。

  因果图：因果图方法是从用自然语言书写的程序规格说明的描述中找出因(输入条件)和果(输出或程序状态的改变)，可以通过因果图转换为判定表。

  正交试验法：就是使用已经造好了的正交表格来安排试验并进行数据分析的一种方法，目的是用最少的测试用例达到最高的测试覆盖率。

  错误猜测

- 白盒(结构)测试 https://www.cnblogs.com/Ming8006/p/5798186.html

  **语句覆盖**：程序中每个语句至少都能被执行一次。

  **判定(分支)覆盖**：每个判断的取真分支和取假分支至少经历一次，即判断的真假值均曾被满足。

  **条件覆盖**：程序中每个判断的每个条件的每个可能取值至少执行一次；条件覆盖深入到判定中的每个条件，但可能不能满足判定覆盖的要求。

  **判定/条件覆盖**：使得判定中每个条件取到各种可能的值，并使每个判定取到各种可能的结果。

  **条件组合覆盖**：使得每个判定中条件的各种可能组合都至少出现一次。它可覆盖所有条件的可能取值的组合，还可覆盖所有判断的可取分支，但可能有的路径会遗漏掉。

## 10.3 软件调试

调试方法：蛮力/ 回溯/ 原因排除

### 10.4 维护

在系统交付使用后，改变系统的任何工作，都可以被称为维护。在系统运行过程中，软件需要维护的原因是多样的，根据维护的原因不同，可以将软件维护分为以下 4 种：

*速记：改死裕王*

- **正确性(改正性)维护**。改正在系统开发阶段已发生而系统测试阶段尚未发现的错误。
- **适应性维护**。在使用过程中，外部环境(新的硬、软件配置)、数据环境(数据库、数据格式、数据输入/输出方式、数据存储介质)可能发生变化。为使软件适应这种环境变化，而去修改软件的过程就称为适应性维护。
- **完善性维护**。在软件的使用过程中，用户往往会对软件提出新的功能与性能要求。为了满足这些要求，需要修改或再开发软件，以扩充软件功能、增强软件性能、改进加工效率、提高软件的可维护性。这种情况下进行的维护活动称为完善性维护。
- **预防性维护**。这是指为了适应未来的软硬件环境的变化，应主动增加预防性的新的功能，以使应用系统适应各类变化而不被淘汰。

# 十一 嵌入式系统设计

### 11.1 硬件

a 嵌入式处理器

嵌入式微处理器 EMPU: 类似通用计算机中的 CPU.

嵌入式微控制器 EMCU(单片机) : 整个计算机系统都集成到一块芯片中. 嵌入式微控制器一般以某一种微处理器内核为核心，芯片内部集成有 ROM/EPROM/E2PROM、RAM、总线、总线逻辑、定时器/计数器、WatchDog（监督定时器）、并口/串口、数模/模数转换器、闪存等必要外设。

嵌入式数字信号处理器 EDSP: 用来快速实现各种数字信号的处理算法。

嵌入式片上系统: 在一块芯片上集成很多功能模块的复杂系统, 如 USB/蓝牙.

b 总线

片内总线/片外总线

c 存储器

高速缓存 cache/ 主存 / 外存(Flash 是属于外存)

d I/O 设备与接口

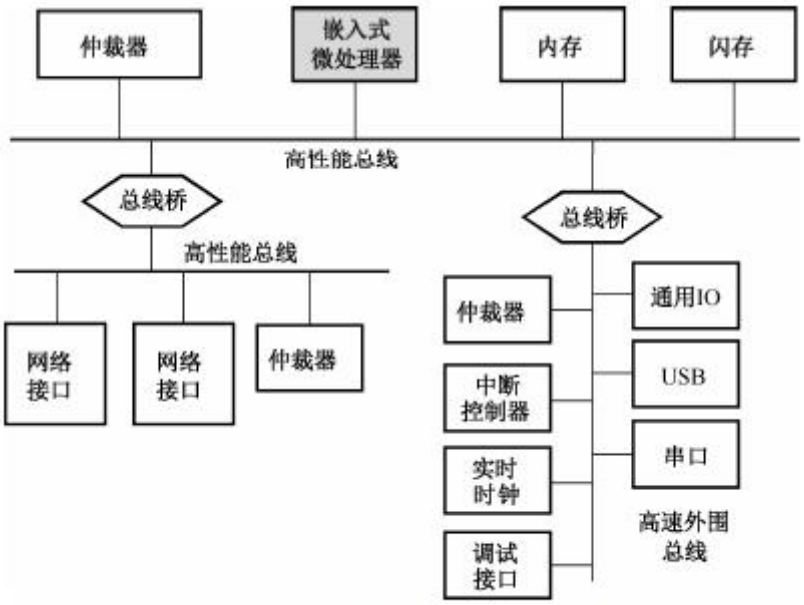



图 12-1　嵌入式硬件平台的系统架构

## 11.2 软件

嵌入式操作系统/ 应用支撑软件/ 应用软件

### a 嵌入式操作系统

**定义：**由操作系统内核、应用程序接口、设备驱动程序接口等几部分组成。嵌入式操作一般采用微内核结构。操作系统只负责进程的调度、进程间的通信、内存分配及异常与中断管理最基本的任务.

### b 应用支撑软件

由窗口系统、网络系统、数据库管理系统及 Java 虚拟机等几部分组成。

### c 应用软件

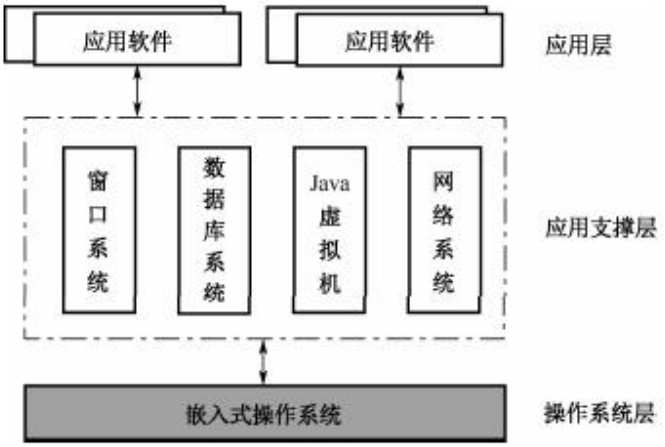



图 12-3　嵌入式系统的软件架构

## 11.3 开发与调试

### 11.3.1 开发平台

交叉平台开发方法，即软件在一个通用的平台上开发，而在另一个嵌入式目标平台上运行。

宿主机提供的基本开发工具是**交叉编译器、交叉链接器和源代码调试器**。作为目标机的嵌入式系统则可能提供一个**动态装载器、链接装载器、监视器和一个调试代理**等。


图 12-4　典型交叉平台开发环境

### 11.3.2 调试

调试方法：直接调试/调试监控法(桩)/在线仿真/片上调试(CPU)/模拟器(纯软件)

- 直接调试：将目标代码下载到目标机上直接执行
- 调试监控法(插桩法)：在宿主机的调试器内和目标机的操作系统上分别启动一个**功能模块**(桩)，然后通过这两个功能模块的相互通信来实现对应用程序的调试. 改进方法：ROM 仿真器----一种用于替代目标机上 ROM 芯片的硬件设备
- 在线仿真法 ICE：目标机来说，在线仿真器就相当于它的 CPU
- 片上调试法：CPU 芯片内部的一种用于支持调试的功能模块. 常用接口有 JTAG
- 模拟器法：纯软件工具

## 11.4 嵌入数据库系统

(狭义)嵌入式数据库管理系统：在嵌入式设备上使用的数据库管理系统.

(广义)一个完整的嵌入式数据库管理系统由若干子系统组成，包括主数据库管理系统、同步服务器、嵌入式数据库管理系统、连接网络等几个子系统.

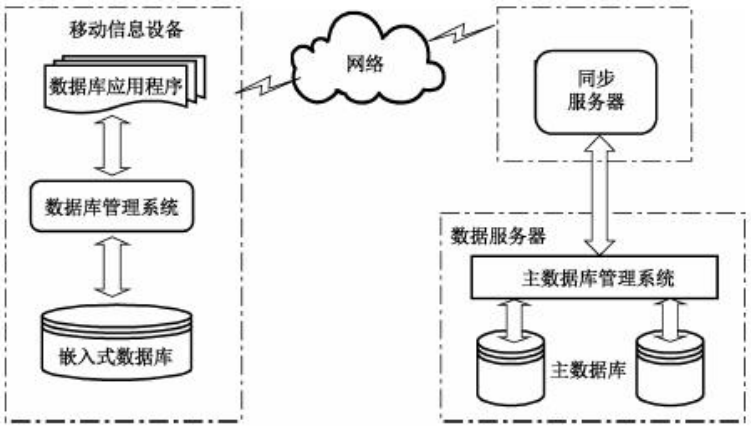



图 12-6　嵌入式数据库系统组成

### 11.5 实时嵌入式操作系统

#### 11.5.1 实时系统

功能正确和时间正确同时满足的系统. 根据对错失时限的容忍程度或后果的严重性分类: 硬实时系统 / 软实时系统.

（1）逻辑（功能）正确，是指系统对外部事件的处理能够产生正确的结果。

（2）时间正确，是指系统对外部事件的处理必须在预定的周期内完成。

（3）死线（Deadline）或时限、死限、截止时间，是指系统必须对外部事件进行处理的最迟时间界限，错过此界限可能产生严重的后果。通常，计算必须在到达时限前完成。

#### 11.5.2 嵌入式实时操作系统 RTOS

时间性能指标: 任务切换时间 / 中断处理相关时间 / 系统响应时间

#### 11.5.3 硬实时系统调度算法

- 单调执行速率调度法 RMS

一种静态优先级调度算法, 基本原则: 执行最频繁的任务优先级最高, 即任务的周期越短, 任务的优先级越高.

要使一个具有$n$个任务的实时系统中的所有任务都满足硬实时条件，必须满足 $\sum_i \frac{E_i}{T_i} \leqslant n\left(2^{1/n}-1\right)$

$E_i$是任务i最长执行时间，$T_i$是任务i的周期，$E_i/T_i$是任务i所需的CPU时间

基于RMS定理，要所有的任务满足硬实时条件，所有具有时间要求的任务总的CPU利用时间（或利用率）应当小于70%。通常，作为实时系统设计的一条原则，CPU利用率应当在60%到70%之间。

| 周期性任务 | 执行时间 | 周期 |
|---|---|---|
| Task 1 | 20 | 100 |
| Task 2 | 25 | 150 |
| Task 3 | 50 | 300 |

$$\frac{20}{100}+\frac{25}{150}+\frac{50}{300} \leqslant 3\left(2^{1/3}-1\right)$$

$$53.34\% \leqslant 77.98\%$$

- 时间轮转调度

当优先级相同时，每个任务轮流给相同的处理时间；当优先级不同时，抢占式，优先级高的先执行.

- 最早截止时间优先调度 EDF(earliest deadline first) (动态)

哪个任务先完成，就先执行.

给定一组n个独立的任务和一组任意的到达时间，任务可调度性的充分必要条件是 $\sum_i \frac{E_i}{T_i} \leqslant 1$

## 十二 开发管理

*速记----范时成需风配人量（范实诚需分配能量）*

### 12.1 范围管理

控制项目的全部活动都在需求范围内，以确保项目资源的高效利用。

- 项目启动
- 范围计划编制
- 范围定义

  输入：① 项目章程。② 项目范围管理计划。③ 组织过程资产。④ 批准的变更申请。
- 范围核实
- 范围变更控制

### 12.2 成本管理

- 资源计划编制
- 成本估算：计算出完成一个项目的各活动所需各资源成本的近似值。
- 成本预算：把估算的总成本分配到单个活动或工作包上去。
- 成本控制

### 12.3 时间管理

- 活动定义

  工作分解结构：WBS(Work Breakdown Structure)
- 活动排序
- 活动历时估算
- 进度计划编制
- 进度控制

### 12.4 配置管理

***配置管理***是通过技术和行政手段对产品及其开发过程和生命周期进行控制、规范的一系列措施和过程。

***产品配置***是指一个软件产品在生存周期各个阶段所产生的各种形式和各种版本的文档、计算机程序、部件及数据的集合。

配置项：1 ***产品组成部分***（如需求文档、设计文档、源代码和测试用例等）；2 项目**管理**和机构支撑过程域产生的文档（如工作计划、项目质量报告和项目跟踪报告等）。

配置项的状态：草稿/正式发布/正在修改

### 12.5 需求管理

需求变更流程：问题分析与变更描述/变更分析与成本计算/变更实现

### 12.6 质量管理

a 软件质量计划

b 软件质量保证

c 软件质量控制

软件评审/测试

## 12.7 风险管理

## 12.8 软件过程改进

过程能力成熟度模型: CMM (capacity maturity model)

## 12.9 人力资源管理

## 12.10 软件工具及开发环境

### 12.10.1 软件工具

软件工具包括:

- 软件开发工具：需求分析工具、设计工具、编码与排错工具。
- 软件维护工具：版本控制工具、文档分析工具、开发信息库工具、逆向工程工具、再工程工具。
- 软件管理和软件支持工具：项目管理工具、配置管理工具、软件评价工具、软件开发工具的评价和选择。

### 12.10.2 软件开发环境

软件开发环境（software development environment）是支持软件产品开发的软件系统。它由软件工具集和环境集成机制构成.

- 软件工具集: 用来支持软件开发的相关过程、活动和任务年;（如上）
- 环境集成: 为工具集成和软件开发、维护和管理提供统一的支持，包括:
  - **数据集成机制**提供了存储或访问环境信息库的统一的数据接口规范;
  - **界面集成机制**采用统一的界面形式，提供统一的操作方式;
  - **控制集成机制**支持各开发活动之间的通信、切换、调度和协同工作。

或者

- **环境信息库**。环境信息库是软件开发环境的核心,用以存储与系统开发有关的信息,并支持信息的交流与共享。环境信息库中主要存储两类信息，一类是开发过程中产生的有关被开发系统的信息，例如分析文档、设计文档和测试报告等；另一类是环境提供的支持信息，如文档模板、系统配置、过程模型和可复用构件等。
- **过程控制与消息服务器**。过程控制与消息服务器是实现过程集成和控制集成的基础。过程集成时按照具体软件开发过程的要求进行工具的选择与组合,控制集成使各工具之间进行并行通信和协同工作。
- **环境用户界面**。环境用户界面包括环境总界面和由它实行统一控制的各环境部件及工具的界面。统一的、具有一致性的用户界面是软件开发环境的重要特征，是充分发挥环境的优越性、高效地使用工具并减轻用户的学习负担的保证。

# 十三 信息系统基础知识

## 13.1 信息系统分类

- 按数据环境

  数据文件

  应用数据库

  主题数据库

  信息检索系统(数据仓库)

- 按应用层次分类

  战略级/ 战术级/ 操作级/ 事务级

## 13.2 信息系统战略规划

a 三个阶段：

（1）以数据处理为核心，围绕职能部门需求

方法：企业系统规划法 BSP/关键成功因素法 CSF/战略集合转化法 SST

**关键成功因素法（CSF）:**在现行系统中，总存在着多个变量影响系统目标的实现，其中若干个因素是关键的和主要的（即关键成功因素）。通过对关键成功因素的识别，找出实现目标所需的关键信息集合，从而确定系统开发的优先次序。

**企业系统规划法（BSP）**:信息支持企业运行。通过自上而下地识别系统目标、企业过程和数据，然后对数据进行分析，自下而上地设计信息系统。该管理信息系统支持企业目标的实现，表达所有管理层次的要求，向企业提供一致性信息，对组织机构的变动具有适应性。

**战略目标集转化法（SST）:**把整个战略目标看成是一个“信息集合”，由使命、目标、战略等组成,管理信息系统的规划过程即是把组织的战略目标转变成为管理信息系统的战略目标的过程。

（2）以企业内部 MIS 为核心，围绕企业整体需求

方法：战略数据规划法 SDP/信息工程法 IE/战略栅格法 SG

（3）以集成为核心，综合考虑企业内外环境，围绕企业战略需求

方法：价值链分析法 VCA / 战略一致性模型 SAM

## 13.3 信息化需求

包括**战略需求、运作需求和技术需求。**

一是**战略需求**。组织信息化的目标是提升组织的竞争能力、为组织的可持续发展提供一个支持环境。从某种意义上来说，信息化对组织不仅仅是服务的手段和实现现有战略的辅助工具；信息化可以把组织战略提升到一个新的水平，为组织带来新的发展契机。特别是对于企业，信息化战略是企业竞争的基础。

二是**运作需求**。组织信息化的运作需求是组织信息化需求非常重要且关键的一环，它包含三方面的内容：一是实现信息化战略目标的需要；二是运作策略的需要。三是人才培养的

需要。

三是**技术需求**。由于系统开发时间过长等问题在信息技术层面上对系统的完善、升级、集成和整合提出了需求。也有的组织，原来基本上没有大型的信息系统项目，有的也只是一些单机应用，这样的组织的信息化需求，一般是从头开发新的系统。

### 13.4 政务信息化

- 主体：

  政府 / 企事业单位 / 公民

- 应用模式：

  政府对政府 / 政府对企业/ 政府对公民(citizen) / 政府对公务员(employee) ； 以及反过来的几个

### 13.5 企业信息化

企业信息化涉及对管理理念的创新：对企业现有的管理流程重新整合，从作为管理核心的财务、资金管理，向技术、物资、人力资源的管理。

企业信息化一定要建立在企业战略规划基础之上，以企业战略规划为基础建立的企业管理模式是建立企业战略数据模型的依据。

企业信息化方法主要包括 **业务流程重构、核心业务应用、信息系统建设、主题数据库、资源管理和人力资本投资**方法。

#### 13.3.1 企业资源规划 ERP

- 企业资源：物流 / 资金流 / 信息流
- 5 个层次计划

**生产预测计划 a** 是对市场需求进行比较准确的预测，是经营计划、生产计划大纲和主生产计划编制的基础；

**销售管理计划**是针对企业的销售部门的相关业务进行管理，属于最高层计划的范畴，是企业最重要的决策层计划之一；

**生产计划大纲 b** 根据经营计划的生产目标制定，是对企业经营计划的细化；

**主生产计划 c** 说明了在一定时期内生产什么，生产多少和什么时候交货，它的编制是 ERP 的主要工作内容；

**物料需求计划**是对主生产计划的各个项所需的全部制造件和全部采购件的网络支持计划和时间进度计划；

**能力需求计划**是对物料需求计划所需能力进行核算的一种计划管理方法，能够帮助企业尽早发现企业生产能力的瓶颈，为实现企业的生产任务提供能力帮面的保障。

### 13.3.2 客户关系管理 CRM

**概念**：CRM 是一套先进的管理思想及技术手段，它通过将**人力资源**、**业务流程**与**专业技术**进行有效的整合，最终为企业涉及到客户或者消费者的各个领域提供了完美的集成，使得企业可以更低成本、更高效率地满足客户的需求，并与客户建立起基于学习性关系基础上的一对一营销模式，从而让企业可以最大程度提高客户满意度和忠诚度。

**主要模块**：销售自动化 / 营销自动化 / 客户服务与支持 / 商业智能

### 13.3.3 企业应用集成 EAI

**a 集成类型（按业务）：**

表示（界面）集成

数据集成

控制（应用/API）集成

业务流程（过程）集成：与上者相比，考虑了业务流程的优化

**b 集成类型（按数据）：**

消息集成：数据量小，要求频繁、立即、异步地数据交换。

共享数据库：实时性强，可以频繁交互，数据交换属于同步方式。

文件传输：数据量大，频率小，即时性要求低。

**c 企业内部信息集成**

（1）**技术平台**的集成

系统**底层**的体系结构、软件、硬件以及异构网络的特殊需求首先必须得到集成。这个集成包括信息技术硬件所组成的新型操作平台，如各类大型机、小型机、工作站、微机、通信网络等信息技术设备，还包括置入信息技术或者说经过信息技术改造的机床、车床、自动化工具、流水线设备等新型设施和设备。

（2）**数据**的集成

为了完成应用集成和业务流程集成，需要解决数据和数据库的集成问题。数据集成的目的是实现不同系统的数据交流与共享，是进行其他更进一步集成的基础。数据集成的特点是简单、低成本，易于实施，但需要对系统内部业务的深入了解。

数据集成是对数据进行标识并编成目录，确定**元数据模型**。只有在建立统一的模型后，数据才能在数据库系统中分布和共享。数据集成采用的主要数据处理技术有数据复制、数据聚合和接口集成等。

(如果是单表即可完成整合,则可以将该表 包装为记录,采用**主动记录**的方式进行集成;如果需要多张表进行数据整合，则需要采用**数据映射**的方式完成数据集成与处理。)

（3）**应用系统**的集成

应用系统集成是实现不同系统之间的互操作,使得不同应用**系统之间**能够实现**数据和方法的共享**。它为进一步的过程集成打下了基础。

（4）**业务过程(流程)**的集成

对业务过程进行集成的时候，企业必须在各种业务系统中定义、授权和管理各种业务信息的交换，以便改进操作、减少成本、提高响应速度。业务流程的集成使得在不同应用系统

中的流程能够无缝连接，实现流程的**协调运作**和流程信息的充分共享。

**d 企业集成平台**

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，包括硬件、软件、软件工具和系统。基本功能包括：

- **通信服务**：提供分布环境下透明的同步/异步通信服务功能;
- **信息集成服务**:为应用提供透明的信息访问服务,实现异种数据库系统之间数据的交换、互操作、分布数据管理和共享信息模型定义：
- **应用集成服务**：通过高层应用编程接口来实现对相应应用程序的访问，能够为应用提供数据交换和访问操作，使各种不同的系统能够相互协作；
- **二次开发工具**：是集成平台提供的一组帮助用户开发特定应用程序的支持工具；
- **平台运行管理工具**：是企业集成平台的运行管理和控制模块。

### 13.3.4 电子数据交换 EDI

三个要素：软硬件/ 通信网络/ 数据标准化


### 13.3.5 企业门户

a 门户类型：

信息门户：企业官网就算

知识门户：分享知识，内部员工使用。

应用门户：业务流程的集成。

### 13.3.6 电子商务

a 参与实体

顾客（个人消费者或集团购买）

商户（包括销售商、制造商、储运商）

银行（包括发卡行、收单行）

认证中心

b 包含流

信息流/资金流/物流/商流

### 13.3.7 供应链

需求信息(如客户订单、生产计划和采购合同等)从需方向供方流动时，便引发物流。

供应信息(如入库单、完工报告单、库存记录、可供销售量和提货发运单等)又同物料一起沿着供应链从供方向需方流动。

### 13.6 商业智能

a 主要技术：数据仓库 / 数据挖掘 / 联机分析处理 OLAP

b 阶段：

**数据预处理**：数据的抽取、转换和装载

**建立数据仓库**：是处理海量数据的基础

**数据分析**：是体现系统智能的关键，一般采用联机分析处理和数据挖掘两大技术。联机分析处理不仅进行数据汇总/聚集，同时还提供切片、切块、下钻、上卷和旋转等数据分析功能，用户可以方便地对海量数据进行多维分析。数据挖掘的目标则是挖掘数据背后隐藏的知识，通过关联分析、聚类和分类等方法建立分析模型，预测企业未来发展趋势和将要面临的问题。

**数据展现**：主要保障系统分析结果的可视化。

### 13.7 其它

- 系统工程

  系统工程是从整体出发合理开发、设计、实施和运用系统科学的工程技术。它根据总体协调的需要，综合应用自然科学和社会科学中有关的思想、理论和方法，利用计算机作为工具，对系统的结构、元素、信息和反馈等进行分析，以达到最优规划、最优设计、最优管理和最优控制的目的。

  霍尔三维机构包括时间维、逻辑维和知识维。

  对于一个具体的工作项目，从制定规划起一直到更新为止，全部过程可分为七个阶段：

  ①规划阶段。即调研、程序设计阶段，目的在于谋求活动的规划与战略；

  ②拟定方案。提出具体的计划方案。

  **③研制阶段。作出研制方案及生产计划。**

  ④生产阶段。生产出系统的零部件及整个系统，并提出安装计划。

  ⑤安装阶段。将系统安装完毕，并完成系统的运行计划。

  ⑥运行阶段。系统按照预期的用途开展服务。

  ⑦更新阶段。即为了提高系统功能，取消旧系统而代之以新系统，或改进原有系统，使之更加有效地工作。

# 十四 安全性和保密设计

## 14.1 对称、非对称、数字签名、数字信封

见 OneNote 表格

## 14.2 密钥管理

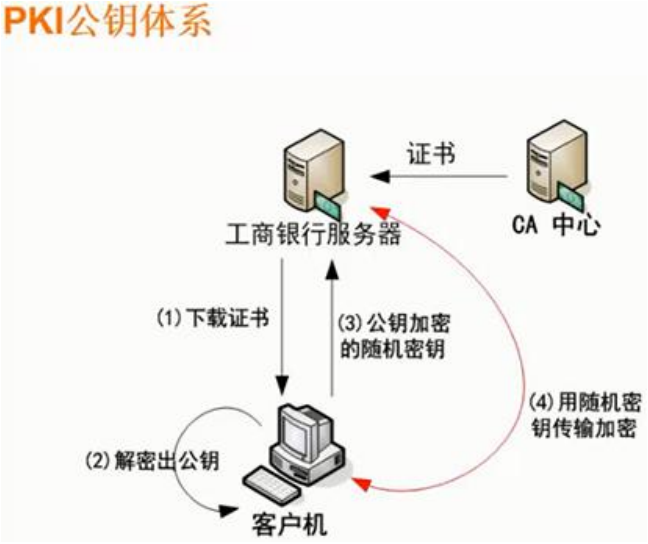
### 14.2.1 密钥分配中心 KDC（Key Distribution Center）

是保存了数字证书的服务器

### 14.2.2 数字证书（如 X.509）

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、证书的有效期及证书的序列号等。

### 14.2.3 公开密钥基础设施 PKI (public key infrastructure)




PKI 公钥体系工作原理实例：

（1）客户端登录工行官网，https：//xxxx

（2）下载工行的数字证书，验证数字证书的真实性，然后从中提取公钥 A；

（3）客户端随机产生一个随机密钥 R，用公钥 A 加密，得到 KK，将 KK 发送给工行官网

（4）工行官网用私钥解密 KK，得到随机密钥 R；

（5）此时，两端都拥有密钥 R，在之后的通信中用密钥 R 进行对称加密。

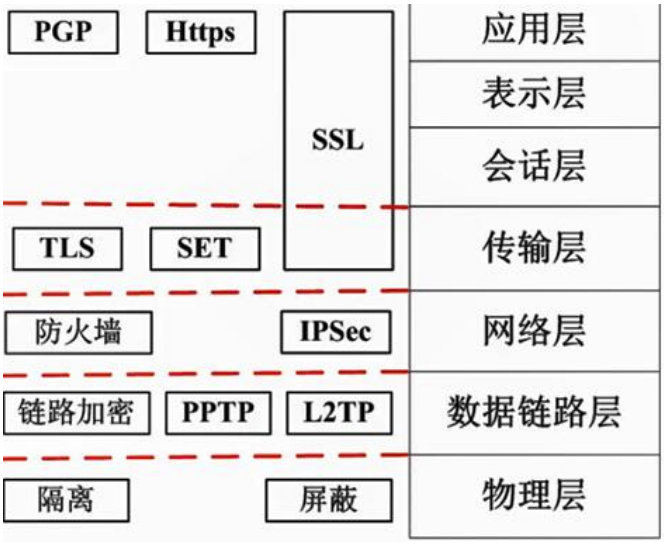
## 14.3 网络安全体系

### a 网络各层次安全保障

IPSec（IP Security）:工作在网络层

SSL：是用于安全传输数据的一种通信协议。它采用公钥加密技术、对称密钥加密技术等保护两个应用之间的信息传输的机密性和完整性。

PGP（Pretty Good Privacy）：实现了加密和数字签名，用在电子邮件系统中.




*物数网传会表应*

b 网络攻击类型

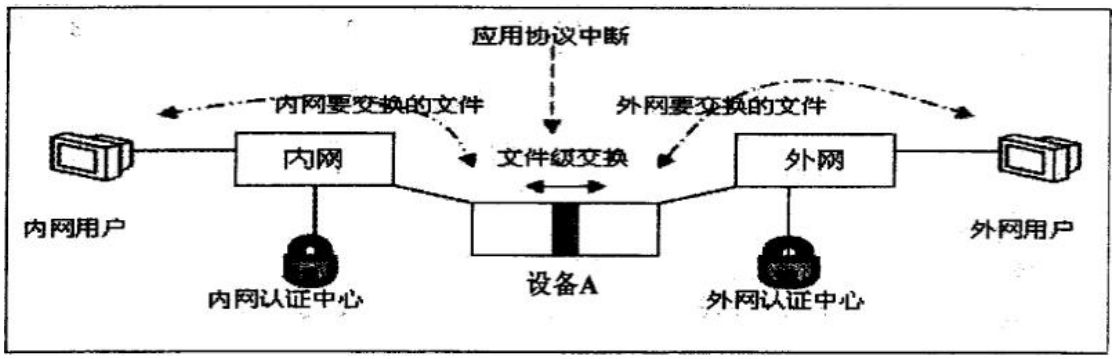
| 威胁名称 | 描述 |
| --- | --- |
| 重放攻击（ARP） | 所截获的某次合法的通信数据拷贝，出于非法的目的而被重新发送。 |
| 拒绝服务（DOS） | 对信息或其它资源的合法访问被无条件地阻止。 |
| 窃听 | 用各种可能的合法或非法的手段窃取系统中的信息资源和敏感信息。例如对通信线路中传输的信号进行搭线监听，或者利用通信设备在工作过程中产生的电磁泄露截取有用信息等。 |
| 业务流分析 | 通过对系统进行长期监听，利用统计分析方法对诸如通信频度、通信的信息流向、通信总量的变化等参数进行研究，从而发现有价值的信息和规律。 |
| 信息泄露 | 信息被泄露或透露给某个非授权的实体。 |
| 破坏信息的完整性 | 数据被非授权地进行增删、修改或破坏而受到损失。 |
| 非授权访问 | 某一资源被某个非授权的人、或以非授权的方式使用。 |

| 威胁名称 | 描述 |
| --- | --- |
| 假冒 | 通过欺骗通信系统（或用户）达到非法用户冒充成为合法用户，或者特权小的用户冒充成为特权大的用户的目的。黑客大多是采用假冒进行攻击。 |
| 旁路控制 | 攻击者利用系统的安全缺陷或安全性上的脆弱之处获得非授权的权利或特权。例如，攻击者通过各种攻击手段发现原本应保密，但是却又暴露出来的一些系统“特性”。利用这些“特性”，攻击者可以绕过防线守卫者侵入系统的内部。 |
| 授权侵犯 | 被授权以某一目的使用某一系统或资源的某个人，却将此权限用于其它非授权的目的，也称作“内部攻击”。 |
| 特洛伊木马 | 软件中含有一个察觉不出的或者无害的程序段，当它被执行时，会破坏用户的安全。 |
| 陷阱门 | 在某个系统或某个部件中设置了“机关”，使得当提供特定的输入数据时允许违反安全策略。 |
| 抵赖 | 这是一种来自用户的攻击，比如：否认自己曾经发布过的某条消息、伪造一份对方来信等。 |

授权侵犯:被授权以某一目的使用某一系统或资源的某个人,将此权限用于其它非授权的目的,也称作”内部攻击”.

c 其它

**网闸**: 一个物理离隔离装置，与 IDS 与防火墙不同，网闸连接的两个网络是不相通的。网闸与内网相联时，会断开与外网的连接，与外网相联时，会断开与内网的连接。网闸其实就是模拟人工数据倒换，利用中间数据倒换区，分时地与内外网连接，但一个时刻只与一个网络连接，保持“物理的分离”，实现数据的倒换。




**重放攻击 ARP**: 是针对以太网地址解析协议（ARP）的一种攻击技术，此种攻击可让攻击者取得局域网上的数据封包甚至可篡改封包,且可让网络上特定计算机或所有计算机无法正常连接。ARP 攻击造成网络无法跨网段通信的原因是伪造网关 ARP 报文使得数据包无法发送到网关。

## 14.4 身份认证和访问控制

### 14.4.1 身份认证

认证方式: 口令认证/基于公钥签名/持卡认证/基于人体生物特征/动态口令/PPP（点到点）认证/ RADIUS

### 14.4.2 访问控制

a 概念：主体/客体/访问规则

b 策略

（1）自主访问控制 DAC（Discretionary Access Control， DAC）: 一个拥有一定权限范围的主体可以直接或者间接地把权限授予其他的主体. 如 windows 和 unix

（2）强制访问控制 MAC（Mandarory Access Control，MAC)：层级多，军事领域，为主体、客体分级。（需要对访问控制权限集中管理；）

（3）基于角色的访问控制 RBAC：角色的种类和访问权限由系统管理员来定义，每一个成员属于哪种类型的角色也由系统管理员来规定,即只有系统管理员才有权定义和分配角色。（将用户与权限分离；适合大型企业；XACML 实现企业各部门或各子系统分布管理访问控制权限；）

## 14.5 系统安全性设计

系统安全考虑层面: 物理环境、通信链路、网络系统、操作系统、应用系统、管理.(参考 *物数网传会表应*)

**物理安全**威胁是指对系统所用设备的威胁，如自然灾害、电源故障、数据库故障和设备被盗等造成数据丢失或信息泄漏。

**通信链路**安全威胁是指在传输线路上安装窃听装置或对通信链路进行干扰。

**网络安全**威胁当前主要是指由于因特网的开放性、国际性与无安全管理性，对内部网络形成的严重安全威胁。

**操作系统安全**威胁指的是操作系统本身的后门或安全缺陷，如“木马”和“陷阱门”等。

**应用系统安全**威胁是指对于网络服务或用户业务系统安全的威胁,包括应用系统自身漏洞，也受到“木马”的威胁。

**管理系统安全**威胁指的是人员管理和各种安全管理制度。

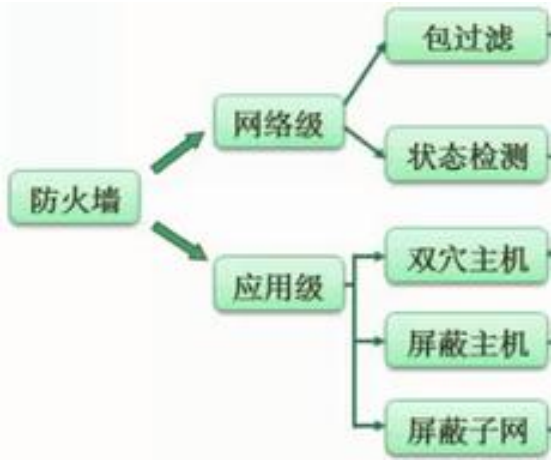
### 14.5.1 物理安全

### 14.5.2 防火墙

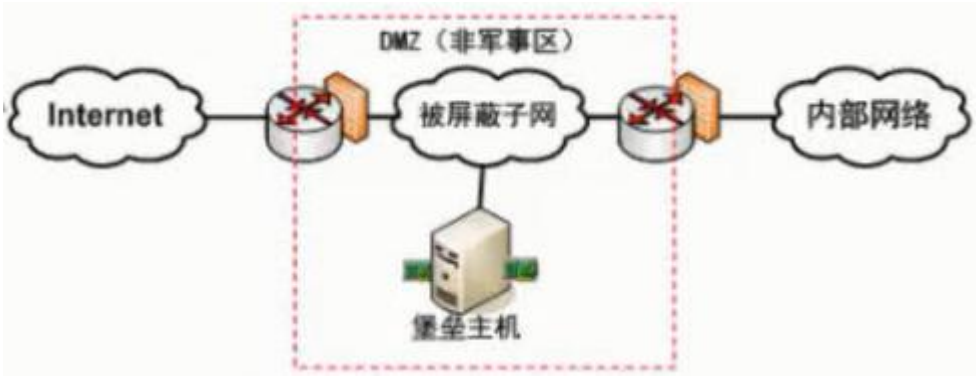
**包过滤防火墙：** 允许或禁止与哪些源地址、目的地址、端口号有关的网络连接。

**状态检测防火墙：**在防火墙的内部建立状态连接表，维护了连接。

**应用网关防火墙：**检查所有应用层的信息包




*屏蔽子网：*




### 14.5.3 入侵检测

如杀毒软件。

**概念**：通过对计算机网络或计算机系统中的若干关键点收集信息并对其进行分析，从中发现网络或系统中是否有违反安全策略的行为和被攻击的迹象。

**处理过程**：数据采集阶段、数据处理及过滤阶段、入侵分析及检测阶段、报告及响应阶段。

**检测技术**：基于标识/基于异常

**分类**：主机型/网络型

## 14.6 几个概念总结

### 加密与数字签名

| | 对称 | 非对称 | 数字签名 | 数字信封 |
|---|---|---|---|---|
| 概念 | 如果采用的加密密钥与解密密钥相同，或者从一个很容易计算出另一个，则这种方法叫作对称密钥密码体制，也叫作单钥密码体制 | 如果加密和解密的密钥并不相同，或者从一个很难计算出另外一个，就叫作不对称密钥密码系统或者公开密钥密码体制，也叫作双钥密码体制 | 接收者能够确认自己得到的信息确实是由该信息所声称的发送者发出的，而不是由非法入侵者伪造、冒充发出的，并且还要能够保证信息在传送、存储中没有被恶意篡改，这样这份信息才能真实地反映发送方的意图(完整性)。另外，对于发送方来说，如果发出一份信息，还必须有一定的措施阻止其否认自己发出信息的行为，即不可否认性(不可否认性)。 | |
| 算法 | DES（56 位密钥）替换+移位<br>3DES（两个 56 位密钥）三重 DES<br>AES/IDEA/RC5<br>流加密：RC5 效率更高<br>块加密：DES/3DES/AES 更安全 | RSA：512 或 1024 位密钥，它是第一个既能用于数据加密也能用于数字签名的算法．由于效率问题，一般不直接用于明文加密）<br>Elgamal<br>ECC | RSA/DSS/DSA<br>散列函数运算的输入信息也可叫作报文。散列函数运算后所得到的结果叫作散列码或者叫作消息摘要。<br>MD5 算法在对输入的报文进行计算时，是以 512 位为单位进行处理的，结果生成一个 128（8-2-1=MD5)位长的消息摘要；<br>SHA、HMAC 等算法都是对任意长度的报文以 512 位为单位进行处理，最后得出一个 160 位的消息摘要。 | |
| 处理过程 | | 只有使用私钥才能解密用公钥加密的数据，同时使用私钥加密的数据只能用公钥解密。<br>在通信过程中，如果发送者要向接收者发送保密信息，则需要先用接收者的公开密钥对信息进行加密，然后发送给该接收者，接收方用其私钥能够顺利解密。而其他人即使收到加密的密文也无法正确解读，从而达到保密通信的目的。<br>**公钥加密 --> 私钥解密** | RSA 结合 MD5 数字签名的具体步骤：<br>（1）信息发送者 A 要向 B 发送一份信息，A 先按双方约定的散列算法对该信息进行散列运算，得到一个该信息特有的消息摘要 H，从前面所述可以知道，只要改动信息中任何一位，重新计算出的消息摘要值就会与原先的值不相符。这样就保证了信息的不可更改性。<br>（2）接着把该消息摘要**用 A 自己的私钥加密**，得到 A 对该信息的**数字签名** S。<br>（3）然后 A 把信息原文与数字签名 S 一起发送给 B。<br>（4）当 B 收到后，先用 A 的公钥对数字签名 S 解密得到 A 的消息摘要 H。<br>（5）再用同样的散列算法对收到的信息进行散列运算，得到消息摘要 H'。<br>（6）比较 H 与 H'，如相等则说明信息确实来自它所声称的发送者 A。 | 信息发送方采用对称密钥来加密信息内容，然后将此对称密钥用接收方的**公开密钥来加密**（这部分称**数字信封**），之后，将它和加密后的信息一起发送给接收方，接收方先用相应的私有密钥打开数字信封，得到对称密钥，然后使用对称密钥解开加密信息。 |
| 实 | 传输数据量较大的内 | 传输数据量较小的内容，如对称加密 | | |

| | | | | |
|---|---|---|---|---|
| 例 | 容 | 的密钥.<br>RSA 加密算法不仅可以用于信息的加密，而且还可以用于发送者的身份验证或数字签名。例如，用户 B 要向 A 发送一个信息 m，而且要让 A 确信该信息就是 B 本人发出的。为此，B 用自己的私钥 SK = (N, d)对信息加密得到密文 c：c = md mod N，然后把 c 发送给 A。A 收到密文后，使用 B 的公钥 PK = (N, e)对密文进行解密得到明文 m：m = ce mod N。这样，经过验证，A 可以确认信息 m 确实是 B 发出的，因为只有 B 本人才有与该公钥对应的私钥，其他人即使知道公钥，也无法猜出或计算出 B 的私钥来冒充他发送加密信息。<br>**私钥加密 --> 公钥解密** | | |

# 十五 系统可靠性

可靠性：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。就是系统无故障运行的概率。

可靠性的子特征：成熟性、容错性、易恢复性、依从性。(子特性速记：成容一一)

失效率：系统运行至此刻未出现失效的情况下,单位时间系统出现失效的概率。

容错：系统在某些组成部分出现故障时仍能正常工作。

## 15.1 基本概念

a 平均无故障时间 MTTF

MTTF = 1/失效率

b 平均故障修复时间 MTTR

MTTR = 1/修复率

c 平均故障间隔时间 MTBF

MTBF = MTTR+MTTF

d 故障来源： 失效/故障/错误

e 故障表现：永久性/间歇性/瞬时性

f 故障模型：

逻辑级：电路元器件出问题

数据结构级：数据结构（二进制位）出问题

软件级：软件设计出错，与设计说明不一致

系统级：功能错误。

g 错误/缺陷/故障/失效

- 软件错误:在软件生存周期内的不希望或不可接受的人为错误,其结果导致软件缺陷的产生。
- 软件缺陷：存在于软件（文档、数据、程序）之中的那些不希望或不可接受的偏差.
- 软件故障：软件故障是指软件运行过程中出现的一种不希望或不可接受的内部状态。
- 软件失效：软件失效是指软件运行时产生的一种不希望或不可接受的外部行为结果。

## 15.2 系统容错

### 15.2.1 容错技术分类

分类 1

硬件冗余： 如三模冗余表决器

软件冗余

信息冗余： 如海明校验码 / 循环冗余校验码

时间冗余： 多次运行

分类 2

动态冗余：又称主动冗余,它是通过故障检测、故障定位及故障恢复等手段达到容错的目的。主要实现方式是多重模块待机储备,当系统检测到某工作模块出现错误时,就用一个备用的模块来替代它并重新运行。

静态冗余：如多模冗余;

混合冗余

### 15.2.2 单机容错

自检技术

冗余技术（重复线路/备份线路）

### 15.2.3 双机容错

双机热备：主系统坏了，备用系统上

双机互备：两台服务器都具有完整服务，平时各自提供不同服务，当 A 坏了，B 就接手 A 的任务

双机双工：同时提供相同的服务，轮流工作。集群的一种

### 15.2.4 服务器集群

特点：可伸缩性/高可用性/可管理/性价比/透明性

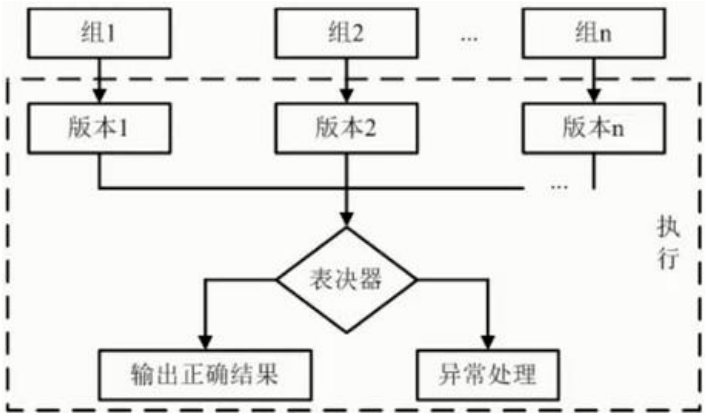
### 15.2.5 软件容错

**a N 版本程序设计**

概念:是一个静态的故障屏蔽技术,用 N 个相同功能的程序同时执行一项计算,结果通过多数表决选择。(其中 N 个版本的程序必须由不同的人独立设计,使用不同的方法、设计语言、开发环境和工具实现,目的是减少 N 个版本的程序在表决点上错误的概率。)

属于前向恢复;

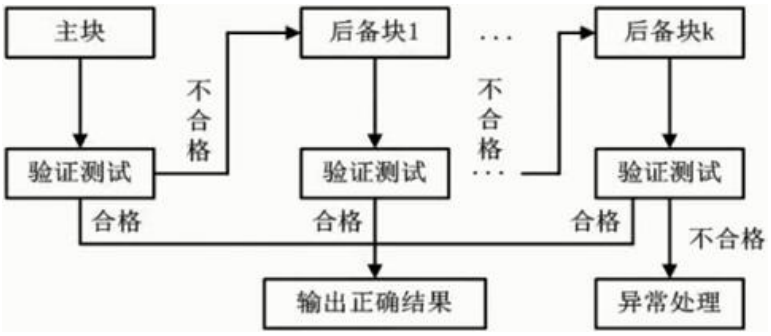
不同规格需求说明书 / 不同团队 / 相同原始需求




**b 恢复块方法**

对于可靠性要求高的软件，在程序运行的某时刻，将数据或程序进行备份，一旦发现主程序块(主块)有异常发生时，可将已备份的数据或程序进行恢复，保证程序的正确性。

属于后向（反向）恢复




| | 恢复块方法 | N版本程序设计 |
|---|---|---|
| 硬件运行环境 | 单机 | 多机 |
| 错误检测方法 | 验证测试程序 | 表决 |
| 恢复策略 | 后向恢复 | 前向恢复 |
| 实时性 | 差 | 好 |

**c 防卫式程序设计**

如 try catch finally

### 15.2.6 检错(自检)技术

**优缺点**

检错技术实现的代价一般低于容错技术和冗余技术，但有一个明显的缺点，就是不能自动解决故障，出现故障后如果不进行人工干预，将最终导致软件系统不能正常运行。

**实现方式**

判断返回结果，如果返回结果超出正常范围，则进行异常处理；

计算运行时间也是一种常用技术，如果某个模块或函数运行时间超过预期时间，可以判断出现故障；

置状态标志位?

**处理方式**

查处故障-停止软件运行-报警。但根据故障的不同情况，也有采用不停止或部分停止软件系统运行的情况，这一般由故障是否需要实时处理来决定。

## 15.3 备份与恢复

### 15.3.1 备份

a 联机备份（冷备份）

b 脱机备份（热备份）

### 15.3.2 恢复

前向恢复：当前计算继续，把系统恢复成连贯的正确状态。如 N 版本表决器

后向恢复：将状态返回到过去的某个时刻。如恢复块

# 十六 软件的知识产权保护

在《反不正当竞争法》中，对商业秘密进行了保护。商业秘密是指不为公众所知，具有经济利益，具有实用性，并且已经采取了保密措施的技术信息与经营信息。

## 16.1 保护年限

| 客体类型 | 权力类型 | 保护期限 |
| --- | --- | --- |
| 公民作品 | 署名权、修改权、保护作品完整权 | 没有限制 |
| | 发表权、使用权和获得报酬权 | 作者终生及其死亡后的50年（第50年的12月31日） |
| 单位作品 | 发表权、使用权和获得报酬权 | 50年（首次发表后的第50年的12月31日），若其间未发表，不保护。 |
| 公民软件产品 | 署名权、修改权 | 没有限制 |
| | 发表权、复制权、发行权、出租权、信息网络传播权、翻译权、使用许可权、获得报酬权、转让权 | 作者终生及死后50年（第50年12月31日）。合作开发，以最后死亡作者为准。 |
| 单位软件产品 | 发表权、复制权、发行权、出租权、信息网络传播权、翻译权、使用许可权、获得报酬权、转让权 | 50年（首次发表后的第50年的12月31日），若其间未发表，不保护 |
| 注册商标 | | 有效期10年（若注册人死亡或倒闭1年后，未转移则可注销，期满后6个月内必须续注） |
| 发明专利权 | | 保护期为20年（从申请日开始） |
| 实用新型和外观设计专利权 | | 保护期为10年（从申请日开始） |
| 商业秘密 | | 不确定，公开后公众可用 |

## 16.2 知识产权人确定

职务作品默认著作权归单位；委托创作默认著作权则归创作者,注意两者不同。

| 情况说明 | | 判断说明 | 归属 |
|---|---|---|---|
| 作品 | 职务作品 | 利用单位的物质技术条件进行创作，并由单位承担责任的 | 除署名权外其他著作权归单位 |
| | | 有合同约定，其著作权属于单位 | 除署名权外其他著作权归单位 |
| | | 其他 | 作者拥有著作权，单位有权在业务范围内优先使用 |
| 软件 | 职务作品 | 属于本职工作中明确规定的开发目标 | 单位享有著作权 |
| | | 属于从事本职工作活动的结果 | 单位享有著作权 |
| | | 使用了单位资金、专用设备、未公开的信息等物质、技术条件，并由单位或组织承担责任的软件 | 单位享有著作权 |
| 专利权 | 职务作品 | 本职工作中作出的发明创造 | 单位享有专利 |
| | | 履行本单位交付的本职工作之外的任务所作出的发明创造 | 单位享有专利 |
| | | 离职、退休或调动工作后1年内，与原单位工作相关 | 单位享有专利 |

| 情况说明 | | 判断说明 | 归属 |
|---|---|---|---|
| 作品软件 | 委托创作 | 有合同约定，著作权归委托方 | 委托方 |
| | | 合同中未约定著作权归属 | 创作方 |
| | 合作开发 | 只进行组织、提供咨询意见、物质条件或者进行其他辅助工作 | 不享有著作权 |
| | | 共同创作的 | 共同享有，按人头比例。<br>成果可分割的，可分开申请。 |
| 商标 | | 谁先申请谁拥有（除知名商标的非法抢注）<br>同时申请，则根据谁先使用（需提供证据）<br>无法提供证据，协商归属，无效时使用抽签（但不可不确定） | |
| 专利 | | 谁先申请谁拥有<br>同时申请则协商归属，但不能够同时驳回双方的专利申请 | |

### 16.3 侵权判定

- 作品不管是否发表，都享有著作权
- 不能引用未发表的作品
- 开发软件所用的思想、处理过程、操作方法或数学概念 不受保护
- 著作权不适用于以下
  - 法律法规，国际机关的决议/决定/命令和其它具有立法/行政/司法性质的文件，机器官方正式译文。
  - 时事新闻
  - 历法/通用数表/通用表格和公式
- 商标要注册后才受商标法保护。
- 艺术作品的著作权不因所有权（买卖）的转移而改变，一直属于创作者，持有者（买家）

拥有所有权和展览权。

- 商标申请分行业领域，不同行业领域可以有相同的商标
- 读音相同（用友/用有）或主题相同（哇哈哈/哇哇哈）属于相同商标

| 不侵权 | 侵权 |
| --- | --- |
| ✓ 个人学习、研究或者欣赏；<br>✓ 适当引用；<br>✓ 公开演讲内容<br>✓ 用于教学或科学研究<br>✓ 复制馆藏作品；<br>✓ 免费表演他人作品；<br>✓ 室外公共场所艺术品临摹、绘画、摄影、录像；<br>✓ 将汉语作品译成少数民族语言作品或盲文出版。 | ✓ 未经许可，发表他人作品；<br>✓ 未经合作作者许可，将与他人合作创作的作品当作自己单独创作的作品发表的；<br>✓ 未参加创作，在他人作品署名；<br>✓ 歪曲、篡改他人作品的；<br>✓ 剽窃他人作品的；<br>✓ 使用他人作品，未付报酬；<br>✓ 未经出版者许可，使用其出版的图书、期刊的版式设计的。 |

### 16.4 标准化

注意 GJB 中国军用标准属于行业标准

- ✓ 国际标准：ISO、IEC等国际标准化组织
- ✓ 国家标准：GB—中国、ANSI—美国、BS—英国、JIS—日本
- ✓ 区域标准：又称为地区标准，如PASC—太平洋地区标准会议、CEN—欧洲标准委员会、ASAC—亚洲标准咨询委员会、ARSO—非洲地区标准化组织
- ✓ 行业标准：GJB—中国军用标准、MIT-S—美国军用标准、IEEE—美国电气电子工程师协会
- ✓ 地方标准：国家的地方一级行政机构制订的标准
- ✓ 企业标准
- ✓ 项目规范

- ➢ 国际、国外标准代号：标准代号+专业类号+顺序号+年代号
- ➢ 我国国家标准代号：强制性标准代号为GB、推荐性标准代号为GB/T
  指导性标准代号为GB/Z、实物标准代号GSB
- ➢ 行业标准代号：由汉语拼音大写字母组成（如电子行业为SJ）
- ➢ 地方标准代号：由DB加上省级行政区划代码的前两位
- ➢ 企业标准代号：由Q加上企业代号组成

## 十七 基于中间件的开发

### 17.1 相关概念

**中间件**：中间件是一种独立的系统软件或服务程序，分布式应用软件借助这种软件在不

同的技术之间**共享资源**。（它的位置处于系统软件和应用软件之间）。

主要产品：IBM MQSeries / BEA Tuxedo

**应用服务器**：是通过各种协议将商业逻辑曝露给客户端的程序。（应用服务器是中间件的一种）。

主要产品：IBM WebSphere / BEA WebLogic

## 17.2 J2EE

教程：https://blog.csdn.net/csdnlijingran/article/details/88533914

- 概念

J2EE 是开发和部署企业应用程序的一种平台或环境。它采用多层分布式应用模型，由一系列服务、应用程序编程接口（API）、提供多层开发的功能性的协议以及基于 Web 的应用程序组成。

- 定义构件

Applet、Servlet、JSP、EJB、Application Client（应用客户端）

- 分层

1 客户层

web 浏览器 / 小应用程序<applet> / application client 应用客户端

2 应用服务层

web 容器：响应客户请求和返回结果

web 组件：JSP+Servlet

EJB 容器：支持 EJB 组件的事务处理和生命周期管理以及 Bean 的查找和其它服务。

EJB 组件：

session Bean:维护一个短暂的会话。负责与 web 层通信。

entity Bean：维护一行持久稳固的数据。负责保存业务数据。

message-dirven Bean：异步接受消息。接受、处理客户通过 JMS 发送过来的消息。

3 企业信息系统层

包括 ERP、数据库、目录服务、其他遗留系统等

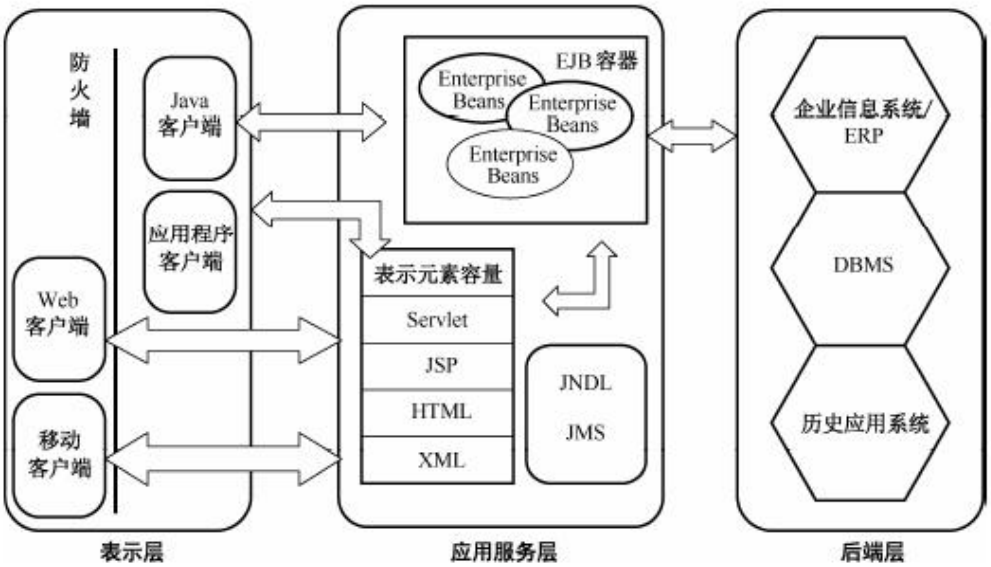
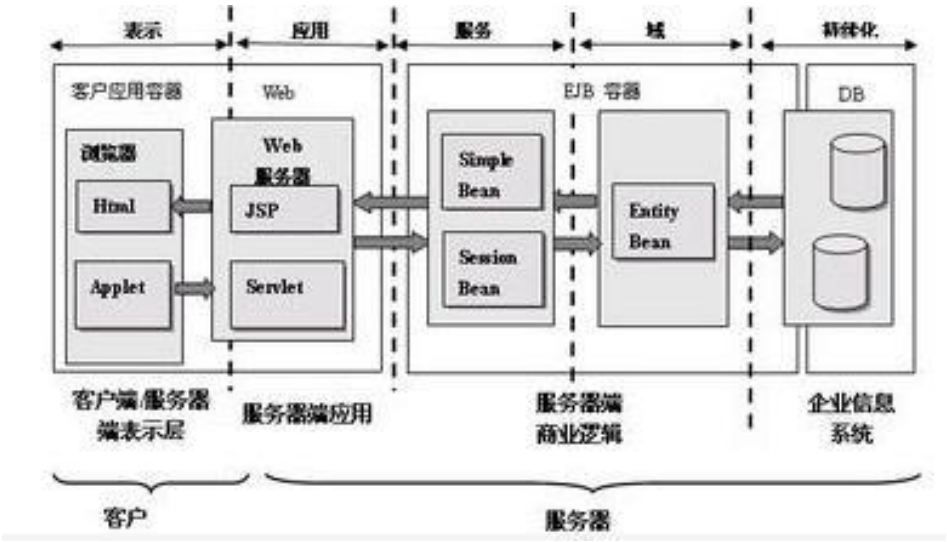
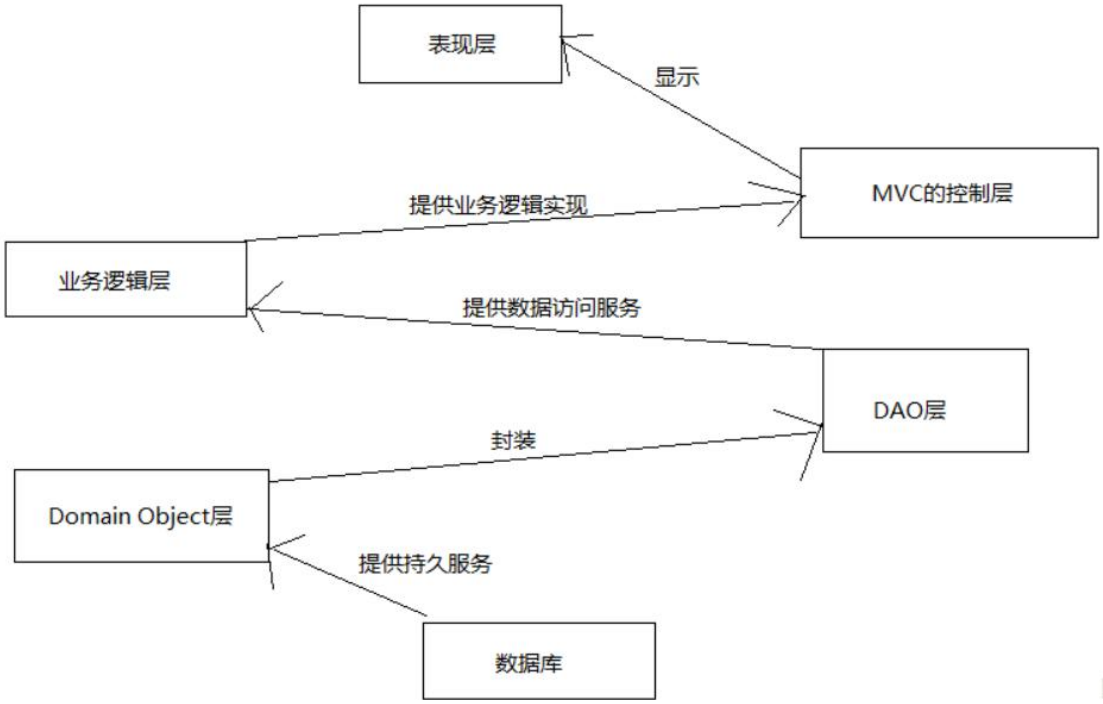



图 15-2　J2EE 多层分布式应用模型图




表示层==视图层==表现层

注意表示层和 web 层的区别




DAO 数据访问对象（data access object）----对应数据持久化层：通过建立逻辑数据操作接口，采取一定的对象/关系映射策略，隐藏了数据库访问代码细节(数据库链接、数据读写命令和事务管理),为开发人员提供给了透明的对象持久化操作机制.

## 17.3 .NET

## 17.4 MVC

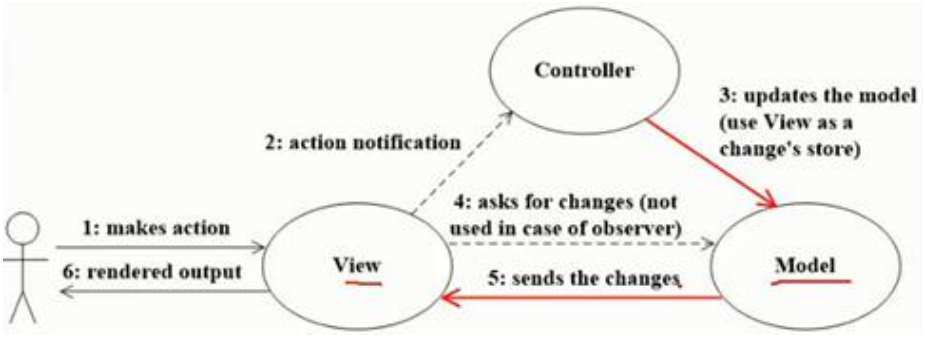
**MVC**: 用一种业务逻辑、数据、界面显示分离的方法组织代码,将业务逻辑聚集到一个部件里, 在改进和个性化定制界面及用户交互的同时,不需要重新编写业务逻辑.

**MODEL 模型**：模型是应用程序的主体部分。模型表示业务数据和业务逻辑。一个模型能为多个视图提供数据。

**view 视图**：视图是用户看到并与之交互的界面。视图向用户显示相关的数据，并能接收用户的输入数据，但是它并不进行任何实际的业务处理。

**controller 控制器**：控制器接受用户的输入并调用模型和视图去完成用户的需求。该部分是用户界面与 Model 的接口。一方面它解释来自于视图的输入，将其解释成为系统能够理解的对象，同时它也识别用户动作，并将其解释为对模型特定方法的调用；另一方面，它处理来自于模型的事件和模型逻辑执行的结果，调用适当的视图为用户提供反馈。

*在一个典型的基于 MVC（Model View Controlle）的 J2EE 应用中，系统的界面由 JSP 构件实现，分发客户请求、有效组织其他构件为客户端提供服务的控件器由 Servlet 构件实现，数据库相关操作由 Entity Bean 构件实现，系统核心业务逻辑由 Session Bean 构件实现.*




# 十八 新技术

## 18.1 云计算

- 概念

云计算是一种基于互联网的计算方式，通过这种方式，共享的软硬件资源和信息可以按需提供给计算机和其他设备。

狭义云计算指 IT 基础设施的交付和使用模式，指通过网络以按需、易扩展的方式获得所需资源；

广义云计算指服务的交付和使用模式，指通过网络以按需、易扩展的方式获得所需服务。这种服务可以是 IT 和软件、互联网相关，也可是其他服务。

- 基本类型

软件即服务（Software-as-a-Service，SaaS）
平台即服务（Platform-as-a-Service，PaaS）
基础设施即服务（Infrastructure as a Service，IaaS）

## 18.2 物联网

- 层次结构

感知层 / 网络层 / 应用层

## 18.3 虚拟化

- 技术分类

平台虚拟化
资源虚拟化
应用程序虚拟化

# E 其它

## 1 集群/分布式/负载均衡

- 集群

集群是个物理概念，是指同一个系统，部署在多台服务器上，将很多服务器集中起来一起进行同一种服务，在客户端看来就像是只有一个服务器。集群可以利用多个计算机进行并行计算从而获得很高的计算速度，也可以用多个计算机做备份，从而使得任何一个机器坏了整个系统还是能正常运行。

就比如新浪网，访问的人多了，他可以做一个集群，前面放一个响应服务器，后面几台服务器完成同一种业务。如果有业务访问的时候，响应服务器看哪台服务器的负载不是很重，就将给哪一台去完成。 一台服务器垮了，其它的服务器可以顶上来。

- 负载均衡

至于集群服务器之间如何分工，需要引入负载均衡的概念了，负载均衡是指将请求分摊到多个操作单元也就是分开部署的服务器上，nginx 是常用的反向代理服务器，可以用来做负载均衡。集群与负载均衡之间有紧密联系，可以结合理解。

实现负载均衡的技术主要有:

1.基于特定服务器软件。利用“重定向”网络协议让客户端的访问跳转到其它服务器上。

2.基于 DNS 域名服务器。在 DNS 服务器中为同一个主机名配置多个 IP 地址，在应答 DNS 查询时，按一定的顺序返回不同的解析结果（IP 地址），将客户端的访问引导到不同的节点。

3.反向代理。将来自 Internet 上的连接请求以反向代理的方式动态地转发给内部网络上的多个节点进行处理。

- 分布式

分布式侧重将一个系统拆分成多个业务单元，例如一个门户网站里面可能有登录、视频、图片等，每一个都可以拆分出来独立部署，而且每一个都可以弄成集群，视频服务集群，图片服务集群，主系统可以对这些子系统进行调用，子系统之间可以有调用关系也可以没有，看实际业务情况。

例子：小饭店原来只有一个厨师，切菜洗菜备料炒菜全干。后来客人多了，厨房一个厨师忙不过来，又请了个厨师，两个厨师都能炒一样的菜，这两个厨师的关系是集群。为了让厨师专心炒菜，把菜做到极致，又请了个配菜师负责切菜，备菜，备料，厨师和配菜师的关系是分布式，一个配菜师也忙不过来了，又请了个配菜师，两个配菜师关系是集群。

| 技术 | 描述 | 支持软件 |
| --- | --- | --- |
| 负载均衡 | 采用软件级和硬件级负载均衡实现分流和后台减压 | HAProxy / LVS |
| 缓存服务器 | 保存静态文件，减少网络交换量，加速响应请求 | Squid / Memcached |
| 分布式文件系统 | 文件存储系统，快速查找文件 | FastDFS<br>Hadoop Distributed File System(HDFS)<br>MooseFS |
| Web 应用服务器 | 加速对请求进行处理 | Apache Tomcat / Jetty / JBoss |

| | | |
|---|---|---|
| 分布式数据库 | 缓存、分割数据、加速数据查找 | MongoDB<br>Mysql |

**2 Internet 服务三种类型**

- 保证质量的服务(Guranteed services)：对带宽、时延、抖动和丢包率提供定量的保证。
- 控制负载的服务(Controlled—load services)：提供一种类似于网络欠载情况下的服务，这是一种定性的指标。
- 尽力而为的服务(Best-Effort)：这是 Intemet 提供的一般服务，基本上无任何质量保证。

**3 管理距离**

管理距离是指一种路由协议的路由可信度。每一种路由协议按可靠性从高到低，依次分配一个信任等级，这个信任等级就叫管理距离。

正常情况下，管理距离越小，它的优先级就越高，也就是可信度越高。一个管理距离是一个从 0-255 的整数值，**0 是最可信赖的，而 255 则意味着不会有业务量通过这个路由。**

**4 看门狗 watchDog**

看门狗软件检测死循环。不断监测程序循环运行的时间，一旦发现程序运行时间超过循环设定的时间，就认为系统已陷入死循环，然后强迫程序返回到已安排了出错处理程序的入口地处，使系统回到正常运行。

# 二十 案例分析

5 个题目选做 3 个

# 二十一 论文

技巧很重要，哪怕离题

记住思路，依葫芦画瓢

题目是 4 选 1

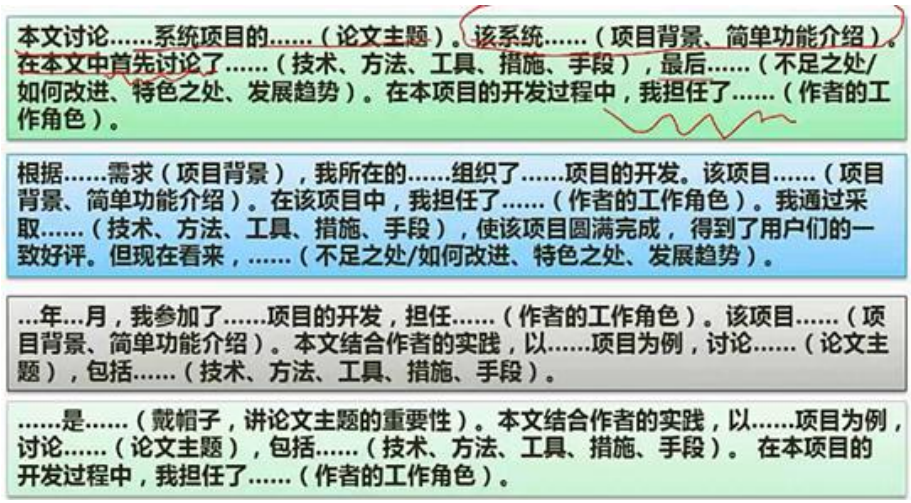
始终围绕一个项目来写

摘要 320 字，正文 2500 左右。


| 过程 | 字数 |
| --- | --- |
| （1）摘要 | 300字～400字 |
| （2）项目概要<br>•开发项目的概要<br>•开发的体制和“我”担任的工作<br>•项目在系统设计方面的情况 | 400字～600字 |
| （3）把握用户需求的重要性 | 100～200字 |
| （4）采用过的手段 | 1000字～1400字 |
| （5）采取的手段中有效果的手段，效果体现在什么地方 | 200字～300字 |
| （6）采取的手段中无效果的手段，为什么没有效果 | 200字～300字 |
| （7）总结 | 100～200字 |

担任的角色： 写系统架构师，不要写编码人员

摘要模板：




不要一致写我我我，也要写我们

图 一定要有必要的时候才画

# 二十二 英语

vocabulary: 词汇表

semantic models: 语义的模型

pattern:模式

metadata:元数据

architecture style:架构风格

architecture pattern:架构模式

aspect:方面

in conjuction:结合

adjunct:附属的

business architecture:业务架构

application architecture :应用架构

reference architecture:参考架构

particular domain:特定领域

specification:规格说明

vendor:供应商

evaluate:评估

feasibility:可行性

encrypt：加密

decrypt：解密

fusion: 融合