2023上半年软考知识点

目录

第 1章 计算机系统知识

计算机系统由两部分组成：硬件、软件。

• 计算机硬件系统五大组成部分

- **控制器、运算器、存储器、输入设备、输出设备**
- 存储器分为内部存储器（**内存、容量小、速度快、存放临时数据，断电消失**）和外部存储器（**硬盘、光盘、容量大、速度慢、长期数据保存**）
- 输入设备、输出设备统称**外设**
- 主机（ **CPU + 主存储器**）

• 中央处理单元 CPU

- CPU 组成：由**运算器、控制器、寄存器组**（读取速度最快）、**内部总线组成**
- CPU 功能：实现**程序控制、操作控制、时间控制、数据处理**功能
- 运算器组成（常考）：
  - 算数逻辑单元 ALU(Arithmetic logic unit)：实现对数据的**算数和逻辑**运算，提供一个工作区
  - 累加寄存器 AC(Accumulator)： **运算结果或者源操作数**的存放区
  - 数据缓冲寄存器 DR(Data Register)： **暂时存放**内存的指令或数据
  - 状态条件寄存器 PSW(Program Status Word)： 保存指令**运行结果**的条件吗内容，如溢出标志等
- 运算器功能：执行**算数运算和逻辑运算**
- 控制器：
  - 指令寄存器 IR （ Instruction Register ） ： 暂存当前

CPU 正在执行的指令

- ◆ 程序计数器 PC （ Program Counter ）：存档**指令执行地址**
- ◆ 地址寄存器 AR （ Address Register ）：保存**当前 CPU 所访问的内存地址**
- ◆ 指令译码器 ID （ Instruction Decoder）：**分析指令操作码**

- 控制器功能：**控制整个 CPU 的工作**，最为重要，包括程序控制、时序控制
- 程序员可以**访问通用寄存器**存取数据，也可以**访问状态寄存器和程序计数器，但是不能访问指令寄存器**

• 数据进制转换

- 二进制、十六进制（ 0x18F、 18FH ）
- R 进制转 10 进制 案例： 6 进制 5043 转为十进制 => 36^0 + 46^1 + 06^2 + 56^3 => 1107
- 十进制转 R 进制 案例：十进制 200 转 6 进制 => 200/6 = 33 余 2 => 33/6 = 5 余 3 => 5/6 = 0 余 5 => 6 进制为余数的从后向前排列 532
- m 进制转为 n 进制：**通过十进制中转**

• 数的表示 （考得少）

- 最小数据单位 b(比特 bit)
- 最小存储单位 1B(字节 byte)=8b
- 1KB=1024B; 1MB=1024KB; 1GB=1024MB
- 机器数：各种数值在计算机中表示的形式，特点是使用二进制计数制，数的符号用 0 和 1 表示，小数点隐含不占位置 （例如 +0 （ 0 0000000 ） -0(1 0000000) ）**其中第一位是符号为，后七位表示数值位**
- 定点表示法：分为纯小数与纯整数两种，其中小数点不占存储位

- ◆ 纯小数：约定小数点在机器数的**最高数值位之前**
- ◆ 纯整数：约定小数点的位置在机器数的**最低数值位之后**

- 真值：机器数对应的实际数值

• 数的编码方式（不怎么考）

- 原码：**一个数的正常二进制表示** 例如 +0 （ 0 0000000 ） -0(1 0000000)
- 反码：**正数的反码即为原码；负数的反码是在原码的基础上，除了符号位以外，其他各位按位取反**（例如如上数值的反码为 +0 （ 0 0000000） -0 （ 1 1111111 ））
- 补码：**正数的补码即原码；负数的补码是在原码基础上，除了符号位以外，其他各位按位取反，而后在末位 +1** ，若有进位则产生进位 +0 （ 0 0000000 ） -0 （ 0 0000000 ）） -0 的补码有溢出
- 移码：用作浮点运算的阶码，无论正数负数，都是将**该原码的补码的首位（符号位）取反得到**
- 计算机系统中常采用**补码来表示和运算数据**，原因是采用补码可以**简化计算机运算部件的设计**

• 浮点表示（常考）

- 浮点数 N = F * 2^E, 其中 E 称为阶码（带符号的纯整数）， F 称为尾数（带符号的纯小数），类似于十进制的科学记数法 例如 101.011=0.101011*2^3
- 浮点数所能表示的**数值范围由阶码确定**，所表示的**数值精度由尾数确定**
- 浮点数运算需要先 1. 对阶，即将阶码换算成相同的后再计算，**小阶向大阶对接**，否则会损失尾数的精度 》 2. 尾数计算 》 3. 结果格式化
- 浮点数存储格式： | 阶符 | 阶码 | 数符 | 尾数 | ，一

般尾数用补码，阶码用移码

- 规格化浮点数：将尾数的绝对值限定在 **[0.5, 1]**
- 浮点数的范围： M: 尾数补码位（包括数符 ）， R: 阶码补码位（包括阶符 ），则最大正数 +(1-2^-M+1) * 2(2R-1 - 1)， 最小负数 -1 * 2(2R-1 - 1)
- 定点表示法与浮点表示法：定点表示法分为定点整数和定点小数，定点表示法的**小数点不需要占用存储位**，总位数相同时浮点表示法可以表示更大的数
- 定点小数在机器字长为 n 的表示范围是定点整数表示范围除以 **2^n-1**

• 寻址

- 立即寻址：操作数包含在**指令**中
- 直接寻址：操作数存在**内存单元**中，指令中给出操作数所在存储单元的地址
- 寄存器寻址：操作数存在某一个寄存器中，指令中给出存放操作数的寄存器名，**比直接寻址要快，寄存器 〞距离 〞 CPU 更近**
- 寄存器间接寻址：操作数存在内存单元中，寄存器中存放了操作数所在的内存地址，指令中则存了寄存器名，也就是说寻址路径 **指令 -> 寄存器 -> 内存**
- 间接寻址：指令中给出操作数地址的地址
- 寻址效率：**立即寻址 > 寄存器寻址 > 直接寻址 > 寄存器间接寻址 > 间接寻址**
- 采用不同寻址方式的目的：**扩大寻址空间，提高编程灵活性**

• 校验码

- 码距：指一个编码系统中，两个合法编码之间至少有多少个二进制不同

- 奇偶校验码：在编码中增加一个校验位来使 1 的个数为奇数或者为偶数，**码距为 2**
- **奇偶校验只能校验错误不能纠正错误**
- 海明码：**一种利用奇偶性来纠错的校验方法，码距为 3**，设数据位有 n 位，校验位有 k 位，则 n 与 k 必须满足以下关系： **2^k-1 >= n+k**
- 循环冗余校验码(CRC)：**可以检错但不能纠错**，码距为 2，**由 k 个数据位 + r 个校验位组成**，校验码由信息码产生，校验码位数越多校验能力越强，求 CRC 编码时，**采用模二运算**

• RISC 和 CISC

- RISC: 精简指令集计算机，**指令少，复杂度低**，指令长度固定，寻址方式少，通用寄存器数量多，**支持流水线技术**，采用**硬布线控制逻辑，组合逻辑控制器**
- CISC：复杂指令集计算机，**指令多，复杂**，指令长度变化，寻址方式复杂多样，通用寄存器数量一般，**也支持流水线技术**，使用**微程序控制技术**

• 流水线技术

- 流水线：多条指令重叠进行操作的一种准并行处理实现技术
- 指令分为三个部分：**取指 -> 分析 -> 执行**
- **一条完整指令的执行时间 = 取指时间 + 分析时间 + 执行时间**
- 流水线周期：指令步骤中所花时间最长的一段，例如：取指 1ms -> 分析 3ms -> 执行 2ms 则流水线周期为 3ms，表示除了第一条外，后边的指令都只需要多花 3ms 就能完成
- 流水线总共用时：理论公式：**一条完整指令完成时间 +（指令总数 - 1)*流水线周期**；实践公式：给第一条指令充分的时间，及第一条指

令的每一个步骤都用一个流水线周期的时间

- 吞吐率：指单位时间内流水线所完成的任务数量，计算公式：**指令条数 / 流水线总共用时，最大吞吐率 = 1 / 流水线周期**
- 异步控制会延长时间，降低性能，每次操作结束后要发出结束信号

• 存储器

- 按照所处位置分类：**内存；外存**
- 按工作方式分类：**读\写存储器 RAM；只读存储器 ROM**
- 按访问方式分类：按**内容访问存储器**（例如：相连存储器），按**地址访问存储器**
- 按寻址访问存储器：**随机存储器、顺序存储器、直接存储器**
- 闪存，一种只读存储器 ROM，删除时**以块为单位删除**，类比为 U 盘
- 虚拟存储器，由**主存 + 辅存** 组成
- 存储系统的层次结构，由内而外：**CPU 内部通用寄存器 》Cache(SRAM 静态随机存储器) 》主\内存储器 (DRAM 动态随机存储器，需要周期性刷新来保持数据) 》外存储器**
- 空间局部性：若一个存储单元被访问，**则其临近的存储单元在不久的将来也很可能被访问**，这种特性就是空间局部性
- 时间局部性：若一个存储单元被访问，**则这个单元在以后也可能被再次访问**，这种特性就是时间局部性

• Cache 高速缓存

- Cache 高速缓存：位于 **CPU 与 主存**之间，用来存放当前最活跃的程序和数据（主存的部分拷贝信息），速度比主存块 5~10 倍，**对程序员来说是透明的**（程序员不可操作）
- Cache **容量越大，命中率越高**，逐渐接近 100%，但是随着容量变大，**Cache 成本和命中时间也在增大**

- 替换算法：目标是**使 Cache 获得更高的命中率**
  - 随机替换算法
  - 先进先出算法
  - 近期最少使用算法
  - 优化替换算法
- Cache 中的地址映像方法
  - 地址映像： CPU 工作时**送出的主存地址**，而要从 Cache 中读写信息，就需要**将主存地址换算成 Cache 地址**，这种转换称为地址映像
  - 直接映像：**主存分区， Cache分块**，主存每个区有与 Cache 相同的分块，**主存每个区的块与 Cache 的块的对应关系时固定的**，硬件电路简单，**但是冲突率高**
  - 全联映像：**主存不分区**，主存与 Cache按照相同的大小分块 Cache 的块可以对应任意的主存上的块，电路设计难，**只适用于小容量的 Cache，冲突率低**
  - 组相连映像：是直接相连与全联映像的折中，**先分组，组与组之间的直接映像**，组内是全联映像
  - 发生冲突概率：**全联映像《 组相连映像 《 直接映像**
  - Cache 与 主存之间的映射是由**硬件自动完成**的

• 中断

- 中断：遇到急需处理的事件时，暂停当前正在运行的程序，**转去执行有关程序，处理完后返回源程序**，这个过程称为中断
- 中断向量：**提供中断服务程序的入口地址**
- 中断响应时间：**发出中断请求开始到进入中断服务程序**，这一段时间
- 保存现场目的：**为了能正确的返回被中断的程序然后继续执行**

• 输入输出（ I/O ）控制方式

- 程序查询方式（程序直接控制方式）：
  - **CPU 和 I/O 只能串行工作**， CPU 需要一直轮询检查状态，长期处于忙等状态， **CPU 利用率低**
  - **一次只能读写一个字**
  - 由 **CPU** 将数**放入内存**
- 中断驱动方式：
  - I/O 设备通过中断信号，主动向 CPU 报告 I/O 操作已完成
  - **CPU 与 I/O 外设可并行操作**， CPU 利用率提升
  - **一次只能读写一个字**
  - 由 **CPU** 将数据**放入内存**
- 直接存储器存储方式（ DMA 方式）：
  - CPU 向 I/O 模块发出数据读写的命令，然后 CPU 就可以做其他事情， I/O 模块与内存建立直接的数据通路， I/O 模块操作完成后，通过中断信号告知 CPU
  - **CPU 与 I/O 外设可并行工作**
  - 由**外设**直接将**数据放入内存**
  - **一次读写的单位是块而不是字**
  - 仅在传输数据块的**起始和结束需要 CPU 的干预**
  - CPU 在一个**总线周期结束时响应 DMA 的请求**，每传输一个数据都要占用一个**存储周期**
  - 由 **I/O 设备**提出的中断请求是**可屏蔽中断**，**电源掉电是不可屏蔽中断**

• 总线

- 总线：链接计算机有关部件的一组信号线，是计算机用来传送信息代码的公共通道
- 总线分类：**数据总线，地址总线，控制总线**

- 总线的优点：**简化系统结构，减少连接线数目**，便于接口设计，便于故障诊断和维修，同时降低了成本
- 总线带宽计算：**时钟频率 * 每秒传送的字节**
- 地址总线宽度计算：地址总线的宽度表明 CPU 的寻址能力，与内存大小相关，内存多大就需要多宽的地址总线，例如：**内存容量为 4GB -> 2^32 B -> 地址总线宽度为 32**
- 数据总线宽度计算：**数据总线宽度就是处理机的字长**
- PCI：**并行内总线，系统总线**； SCSI: **并行外总线**

• 加密技术与认证技术

- 加密解决的问题：**窃听**
- 认证解决的问题：**篡改、假冒、否认**
- 加密技术
  - 对称加密：**加密解密用的是同一把秘钥，且只有一把秘钥**；加解密速度快，适合大量明文数据，秘钥分发有缺陷
  - 非对称加密：**加解密不是同一把秘钥**，一共有两把秘钥（公钥私钥）；加解密速度慢，秘钥分发没缺陷，公钥私钥之间不可推算
  - 混合加密：就是将对称加密与非对称加密混合使用：先用对称加密将大量明文数据加密，再用非对称公钥对“对称加密的秘钥”加密，并随着加密明文一起传给接收方，接收方使用非对称的私钥解密出 “对称加密的秘钥”，再用 “对称加密的秘钥” 解密明文
- 认证技术
  - 摘要：**将发送的明文 Hash 算法后得到摘要**，放在密文后一起发送过去，与接收方解密后的明文 Hash 算法的摘要结果对比，一致则没有篡改
  - 数字签名：在摘要的基础上，**对摘要进行私钥签名**，接收方通过**公钥对数字签名进行解密**，可以判定是否被篡改，假冒，否认，用来验证消息来源的真实性

- 数字证书：**使用第三方机构 CA 的私钥来对用户的公钥进行数字签名**，来保证公钥不被篡改，接收方用 CA 的公钥解密来得到发送方的公钥
- 用**数字证书来认证用户身份**，用数字签名来防篡改、假冒、否认

- 加密算法
  - 对称加密算法（私钥，私有秘钥加密，共享秘钥加密算法）
    - DES 3DES RC-5 IDEA AES RC4
  - 非对称加密算吗（公钥，公开秘钥加密）
    - ECC RSA DSA
  - Hash 函数、 MD5 摘要算法、 SHA-1 安全散列算法

• 系统可靠性

- 设一个系统由 N 个子系统组成，子系统的可靠性分别为 R1 R2 R3
- 串联系统可靠性： R = R1R2R3
- 并联的系统可靠性： R = 1 - (1-R1)(1-R2)(1-R3)

• 其他

- 指令寄存器的位数**取决于指令字长**
- 计算机分级存储速度依次为： CPU **内部通用寄存器 》** Cache **》内存 》外存**
- 安全需求：
  - 物理线路安全 - **机房安全**
  - 网络安全 - **入侵检测**
  - 系统安全 - **漏洞补丁管理**
  - 应用安全 - 数据库安全

第 2章 程序设计语言

• 低级语言与高级语言

- 低级语言：**机器语言与汇编语言**
- 高级语言：**面向应用的各类程序设计语言**，例如 C JAVA Python ，与自然语言比较接近，提高程序设计效率
- 高级语言或者汇编语言编写的叫源程序，**源程序不能直接在计算机上执行**

• 解释器（解释程序）

- 翻译源程序时**不生成独立的目标程序**
- 解释程序和源程序**需要参与到程序的运行过程中**

• 编译器（编译程序）

- 翻译时，**将源程序翻译成独立保存的目标程序**
- 机器上运行的时与源程序**等价的目标程序**
- 编译器与源程序都**不参与目标程序的运行过程**

• 程序设计语言的数据成分

- 标识符：由**数字、字母、下划线**组成的标记
- 常量与变量：按照程序运行时数据的值能否改变来区分，**常量存储在池中，没有自己的存储单元**
- 全局量和局部量：按照数据在程序代码中的**作用范围来划分**
- 控制结构：**顺序结构、循环结构、选择结构**
- 程序中的数据具有类型的作用： 1. 便于为**数据分配存储单元**； 2. 便于对参与表达式计算的**数据对象进行检查**； 3. 规定数据对象的取

值**范围以及能够进行的运算**

- 表达式的左结合：由左向右执行例如： a && b
- 表达式的右结合：由右向左执行例如： x = y = z

• 传值调用和传址调用

- 函数定义：函数首部（返回值类型 函数名 （形参 ）） + 函数体（{}）
- 传值调用：将**实参的值传给形参，实参可以是常量、变量、表达式，不可以实现实参形参之间双向传递数据的效果**
- 传址调用：将实参的地址传给形参，**实参必须有地址，实参不能是常量（值）或者表达式，可以实现实参形参之间双向传递数据的效果，**即改**形参的值，实参的值也同时改掉了**

• 编译、解释程序翻译阶段

- 编译方式各个阶段：**词法分析 - 语法分析 - 语义分析 - 中间代码生成（可省略） - 代码优化（可省略） - 目标代码生成**
- 解释方式各个阶段：**词法分析 - 语法分析 - 语义分析**
- 编译器与解释器都不可省略且变换顺序的阶段是 **词法分析、语法分析、语义分析**
- 编译器的**中间代码生成和代码优化阶段**不是必要的，可省略

• 符号表的作用

- **不断收集、记录和使用源程序中一些符号的类型和特征信息**，将其存入符号表中
- 记录**源程序中各个字符的必要信息**，以**辅助语义的正确性检查和代码**生成

• 词法分析

- 将源程序看做**一个多行的字符串**，从上到下、由左到右进行逐个字符扫描，从中识别出一个个单词符号（如关键字、标识符、运算符等）
- 词法分析分析出的单词常常以 **二元组的形式**输出即 单词种别和单词的值
- 词法分析的依据是**语言的词法规则**
- 词法分析输入的是**源程序**，输出的时**记号流**
- 词法分析的主要作用：**分析构成程序的字符及由字符按照构造规则构成的符号是否符合程序语言的规定**

• 语法分析

- 在词法分析基础上，根据语言的语法规则将单词符号序列**分解成各类语法单位**，如 ’ 表达式 ‘ ’ 语句 ‘ ’ 程序 ‘ 等
- 如果没有语法错误，**语法分析后会构造出一个语法树**，否则指出错误，给出诊断信息
- 语法分析输入的是**词法分析产生的记号流**，输出的时**语法树**
- 语法分析主要作用：**对各条语句的结构进行合法性分析，发现程序中的所有语法错误**

• 语义分析

- 用来检查源程序**是否包含静态语义错误**，主要用来**类型分析和检查**
- 语义分析输入的是**语法分析阶段产生的语法树**
- 语义分析阶段**不能发现所有的语义错误，只能分析出静态的语义错误**，动态的语义错误要在**运行时才能发现**
- 动态语义错误常见的有**死循环**

• 目标代码生成

- 这一个阶段的任务是把**中间代码变成特定机器上的绝对指令代码**、可重定位的指令代码或汇编指令代码
- 目标代码生成阶段的工作与**具体的机器密切相关**
- **寄存器的分配**处于目标代码生成阶段

• 中间代码生成

- 根据语义分析的**输出生成中间代码**，是一种简单的记号系统，**可以有多种形式，其特征是与具体的机器无关**
- 语义分析和中间代码生成所依据的时**语言的语义规则**
- 常见的中间代码：**后缀式、三地址码、三元式、四元式、树（图）**等形式
- 可以**将不同的高级语言编译成同一种中间代码**
- 中间代码**可以跨平台**
- 中间代码**有利于进行与机器无关的优化处理**和**提高编译程序的可移植性**

• 正规式

- 正规式是**词法分析的工具**
- 大致可以类比 JS 中的正则的基本规则，主要**记住 * 代表的取值范围**时 [0, 无穷大]，然后将选项带入看看符不符合规则即可

• 有限自动机

- 有限自动机是**词法分析的工具**，他能正确的识别正规集
- 状态，分为**初态和终态，一个状态既可以是初态也可以是终态**
- 识别成功的依据是：**状态机的路跑的通并且跑完后的终点是终态**
- 确定的有限自动机：**对每一个字符来说识别字符后的转移状态是唯一**

的

- 不确定的有限自动机：**对每一个字符来说识别字符，后的转移状态是不唯一的**
- 区分确定的有限自动机和不确定的有限自动机**在于给定一个数字或者字母，它只有一条路可以跑，那就是确定的**，反之是不确定的
- 空串

• 上下文无关文法

- 被广泛的用于表示各种程序设计语言的语法规则 - **上下文无关文法**
- 做题：**由开始符号起始，推到出选项中的终结符号**，一个选项一个选项的尝试

• 后缀式、中缀式

- 中缀式就是常见的表达式： 1*2
- 后缀式的符号放在后边： 12*
- 中缀式转后缀式：按照 () 、 * / 、 + - 的优先级，一个表达式一个表达式的转换成后缀式，同等优先级的从右向左转换
- 后缀式转中缀式：**使用栈的方式**（先进后出、后进先出）
- 语法树中缀遍历 -> 生成中缀式：**左根右**
- 语法树后缀遍历 -> 生成后缀式：**左右根**
- 后缀式又称**逆波兰式**

• 其他

- 反编译通常**不能将可执行文件还原成高级语言源代码**，只能转换成功能**等价的汇编程序**
- 动态语言指的是**程序运行时可以改变其结构**
- 指针变量：变量是**内存单元的抽象**，用于在程序中保持数据，当变量

存储的时**内存单元地址时**，称为指针变量

- 链表中的节点控件需要**程序员申请和释放**，数据控件应采用**堆存储分配策略**
- 可视化程序设计特点：
  - ◆ 基于面向对象思想，**引入控件概念和事件驱动**
  - ◆ 研发过程遵循，**先界面绘制，再基于事件编写程序代码**
  - ◆ 设计人员**可以不用补编写或者少量编写代码**
- **编译**过程中为变量分配存储单元所用的是**逻辑地址**，**程序运行时**再映射为**物理地址**
- C 中全局变量的存储空间在**静态数据区**
- 语法指导翻译是一种**静态语义分析方法**
- 语法分析方法：
  - ◆ **递归下降分析法和预测分析法**都是属于自上而下的分析法
  - ◆ **移进 －归约分析法**属于自下而上的分析法

第 3章 知识产权

• 著作权

- 著作权分为**人身权和财产权**
- 人身权分为：**发表权、署名权、修改权、保护作品完整权**
- 除发表权外，其他权利的时限都是永久，**发表权的时限是终生 + 死后 50 年**
- 知识产权的地域性：在哪里授予的知识产权，**只在授予国家有效，在外部国家不受保护**

• 计算机软件著作权

- 计算机软件著作权受**《中华人民共和国著作权法》**和**《计算机软件保护条例》**保护
- 计算机软件著作权的主体是**公民**
- 计算机软件著作权的客体的是**指受保护的计算机程序**（源程序和目标程序）和**相关文档**（流程图、说明书、用户手册）
- 计算机软件著作人身权：**发表权、开发者身份权（署名权、永久）**
- 计算机软件著作权的保护期：**自软件开发完成之日起，保护期为 50 年**，保护期满，**除了开发者身份权外，其他权利终止**
- 《计算机软件保护条例》是**国务院颁布**的
- 侵权行为鉴别：**未经著作权人的同意，发表、登记、署名、更改、翻译、复制、出售、出租**

• 职务作品

- 职务软件作品指公民在**单位任职期间**为执行本单位的工作所开发的计算机软件作品

- 公民在单位任职期间所开发的软件，**著作权属于单位**。如果开发软件不是职务工作，那著作权就不是单位所有，**但是如果用了单位设备，则不能归个人享有**
- 如果是职务软件作品，**那开发者只有署名权**

• 委托开发

- 委托开发的作品，**著作权由委托方和受委托方订立的合同决定**，无合同约定的，**著作权为受委托方所有**

• 商业秘密

- 商业秘密的基本内容：**经营秘密和技术秘密**
- 商业秘密的构成条件：
  - **具有未公开性**，不为公众所知悉
  - **具有实用性**，能给权利人带来利益
  - **具有保密性**，即采取了保密措施

• 专利权

- 由书面形式申请的，**一份申请一项发明**
- **专利权保护期限 20 年（实用新型专利是 10 年）**
- 专利权就**申请之日起算**，两人以上申请，**现申请先获得**，同一天申请，**由两人协商决定**

• 商标权

- **自核准之日起， 10 年有效**，届满前可以续，**每次续 10 年**
- **谁先注册，谁享有商标**；同时注册，**谁先使用，谁享有商标**；同时注册，都没有使用，**则协商或者抓阄决定**

• 软件许可使用

- 独占许可使用：软件著作权人**不能再给他人许可**，软件著作权人**也不可使用**
- 独家许可使用：软件著作权人**不能再给他人许可**，但是软件著作权人**可以使用**
- 普通许可使用：软件著作权人**可以再给他人许可**，软件著作权人**也可以使用**

• 软件著作权中的翻译权

- 将软件**从一种程序设计语言转化成另一种程序设计语言**

第 4章 数据库知识

• 数据模型的分类

- 概念数据模型：从信息世界中抽象的数据模型，独立与计算机系统，一般采用 **实体 -联系方法（ E-R 方法）**来表示，用于信息世界建模
- 结构数据模型（数据模型 DBSM)：**直接面向数据库的逻辑结构**，一般会考察数据模型中的关系模型和相应的关系模式

• 概念数据模型常用术语

- 实体：客观存在相互区别的**事务**，例如 一个人，一个单位，一个外部系统
- 属性：用来描述实体的**特性**，例如 人的姓名，单位的地址
- 码 |键：**唯一标识实体的属性或属性集**
- 域：**属性的取值范围**
- 联系：**实体之间的对应关系**

• 实体之间的联系分为三种

- **一对一**（一个班级对应一个班长）
- **一对多**（一个班级对应多个学生）
- **多对多**（一个老师可以对应多个班级，一个班级也可以有多个老师）

• E-R 图（实体 -联系）

- **实体 - 矩形**表示
- **属性 - 椭圆**表示
- **联系 - 菱形**表示

- 使用无向边链接，用 1:n，n:1，n:m 来表示联系的类型
- 结构数据模型主要分为：**层次模型，网状模型，关系模型和面向对象模型**

• 三级模式和两级映射

- 外模式（用户模式或者子模式）：**对应外部视图，和用户交互**
- 概念模式：**对应数据库基本表**
- 内模式（存储模式）：**实际数据库存储文件**
- 模式 -内模式映像：实现了概念模式和内模式之间的转换，保持数据的**物理独立性**
- 外模式 -模式映像：实现了外模式和概念模式之间的转换，保持数据的**逻辑独立性**

• 关系模型中基本术语

- 关系：一个关系就是**一个二维表**
- 元祖：表中的一行就是一个元祖，对应**一个记录值**
- 属性：表中的一个列就是一个属性，列的第一行就是属性名，其他为属性值
- 域：**属性的取值范围**
- 关系模式：对关系的描述，格式：**实体**（属性 1，属性 2，…属性 n）
- 候选码 |候选键：**能够唯一标识一个元祖的属性**或者属性组合
- 主码 |主键：一个关系中**可能会有多个候选码**，从中选择一个作为主码
- 外码 |外键：一个关系中的**属性或属性组并非该关系的码**，但是是别的关系中的主码，则称其为该关系的外码 |外键
- 全码：**一个候选码，包含关系中的所有属性**，则该候选码为全码
- 超码：包含候选码的**属性集**

- 主属性：**候选码中的都是主属性**，其他的时非主属性

• 关系模型完整性约束

- 实体完整性：**主键的值不能空**
- 参照完整性：**外键的值可以为空**，但是如果有值，则其值一定能在对应的表中找到
- 用户定义完整性：用户定义的**对某一数据的指定约束条件**

• 关系代数运算

- ∪ （并集 ）： R ∪ S，R,S 所有元祖 | 记录 合并，**删除重复记录后的结果**
- — （差 ）： R-S，从 R 中**删除存在于 S 中的记录**
- ∩ （交集 ）： R ∩ S，R,S 中**同时存在的记录组成**
- × （笛卡尔积 ）: R × S，R 中每条记录与 S 中的**每条记录拼接组合成新的记录**
- π （投影 ）： π 1,3 ® 从关系 R 中抽取第 1， 3 **列属性**，组成新的关系，（其中 1， 3 列号，也可以用对一个属性名代替）
- 6（选择 ）： 6 1=5 ® 从关系 R 中选择所有第 1 列的值等于第 5 列值的记录，**组成新的记录**
- 链接：链接就是在**两个关系的笛卡尔积中选择符合条件的行**
- 等值链接：连接条件为**两列值相等**
- 自然链接（ |><| ）：不需要写链接条件，自然链接中会选出**两个表中同名属性对应值相同的记录**，并且生成的新关系中**去除了重复的属性列**
- 左外链接： R 自然链接 S，在自然链接基础上，将左侧关系 R 中在自然链接结果中丢失的记录与自然链接结过拼接的结果，**用空值 NULl 来填充右侧属性**

- 右外链接：与**左侧链接相反**
- 全外链接：左外链接与右外链接的结果叠加

• 关系模式

- R<U,D,dom,F> 也就是 **关系名 < 属性组 ， 属性的域 ， 属性到域的映射 ， 属性组中属性的数据依赖关系 >** 通常 D dom 可以省略，例如： R<{A, B, C, D}, {A->B, A->C, C->D}> “->” 可以理解为“推导出”或者“决定”的意思
- 完整函数依赖：例如 （学号，课程 ）-> 成绩，但是单独的 学号 或者 课程不能直接推导出 成绩，这就是**完整函数依赖**
- 部分函数依赖：例如（学号，课程号） -> 姓名，单独的 学号 也可以推导出 姓名，这就是**部分函数依赖**
- 依赖传递： A->B ， B->C ，则称 **C 对 A 传递依赖**
- 属性闭包计算（挑选主键），由推导关系选出可以完全推导出所有属性集的某个属性或者某几个属性组合，例如：R<{A, B, C, D}, {A->B, A->C, C->D}> 中 A 属性可以完全推导出 {A, B, C, D}， **A 就是主键，有多个情况，则可以有多个候选键**
- 冗余函数依赖：例如 {A->B, B->C, A->C, C->D} 中 A->C 即为冗余，**因为 A->B->C 有传递依赖，传递依赖优先**（自己的理解）

• 范式 - 应试技巧

- 1NF -> 2NF: 消除**非主属性对码的部分函数依赖**
- 2NF -> 3NF: 消除**非主属性对码的传递函数依赖**
- 3NF -> BCNF: 消除**主属性对码的部分函数依赖和传递函数依赖**
- BCNF -> 4NF: 消除**非平凡且非函数依赖的多值依赖**
- 结题技巧：
  - **一般都满足 1NF**，除非有类似 工资（基本工资，加班工资，实发工资）这种可在细分的属性

- 通过函数依赖集，**找到 “码”**，以及主属性与非主属性（**能被推导出的属性都不是主属性或者码**）
- 判断非主属性是否对码有部分函数依赖，即**非主属性是否能通过码的一部分就推导出来**（如果是 码中的成员 + 非主属性 A -> 非主属性 B 这种则不算部分函数依赖）符合则至少是 2NF
- 看有没有传递函数依赖（类似 A->B, B->C; 或者是伪传递时的某些情况： A->B, BC->D 那么可以有 AC->D），符合则至少是 3NF,
- 查看**主属性对候选码是否有部分函数依赖或者传递依赖**，符合则至少是 BCNF
- 看有没有多值依赖，并且多值依赖的左边是码，例如 A->B,A->C, 并且 A 是码，那就符合第四范式

• 判断是否为无损连接

- 直接把分解后的关系，**通过“自然链接”链接**，看看得到的结果是否与原来一致，**不一致则是有损**

• 判断是否为“保持函数依赖”

- 就看分解后的两个关系中**是有原来函数依赖集的那些依赖关系**，如果没有就是不保持

• 数据库设计步骤

- 用户需求分析 - **数据流图**;
- 概念设计 - **E-R 图**;
- 逻辑设计 - **关系模式**

• E-R模型

- 实体：**矩形表示**；
- 弱实体：**双边矩形表示**，一个实体的存在必须依赖另外一个实体为前提，这类实体就是弱实体
- 联系：**菱形表示**，无向边上标明联系的类型 1:n n:1 n:m，弱实体的联系为双边菱形
- 属性：**椭圆形表示**；
- 简单属性：**不可再分的属性**；复合属性：**可以细分为更小的部分**
- 多值属性：**双边椭圆表示**，多值属性就是一个属性可以对应一组值，
- 派生属性：**虚线椭圆表示**，可以通过其他属性计算得来的属性

• E-R 图之间的冲突类型

- 属性冲突：属性的类型、取值范围、数据单位冲突
- 命名冲突：相同意义属性**在不同 E-R 图中有不同命名**
- 结构冲突：同一个对象在**某个 E-R 图中抽象为实体**，在另一个 E-R 图中抽象为属性，或者同一实体在不同 E-R 中有不同属性

• 关系模型转换

- **1:1 关系转换为关系模式**：把联系对应的属性放到任意一个实体中，同时将另外的实体的主键也放到这个实体中
- **1:n 关系转换为关系模式**：把联系对应的属性放到 n 对应的实体中，同时将另外的实体的主键也放到这个实体中
- **n:n 关系转换为关系模式**：把联系单独作为一个新的关系，其他实体的主键组合为这个新关系的主键
- 关系模式规范化的要求：**至少满足 3NF**

• 事务管理的特性

- 原子性：**事务是原子的**，要么都做，要么都不做
- 一致性：事务执行的**结果必须保证从一个一致性状态 变到另一个一致性状态**
- 隔离性：**事务间互相隔离，多个事务并发时**，任意事务的变更操作知道其成功提交的**整个过程对其他事务都是不可见的**
- 持久性：一旦事务成功提交，**即使数据库崩溃，其对数据库的更新操作也永久有效**

• 数据库备份方法

- 静态转储与动态转储，动态指在转储期间**允许对数据库的存取修改操作**，静态则不允许
- 海量存储与增量存储，增量是指**仅转储距离上次转储更新的数据**
- 日志文件，把对数据库的**每次操作写入日志文件**，一旦发生故障，则利用日志文件撤销事务，**回退到事务前的数据状态**

• 封锁

- 排它锁：就是加了这个锁那么**其他的排它锁和共享锁都不能加了**，直到这个锁被释放了才可以加其他的排它锁和共享锁
- 共享锁：就是加了这个锁那么**不能加排它锁，但是可以加共享锁**，直到这个锁被释放了才可以加排它锁

• 分布式数据库

- 分片透明：指**不需要知道表具体是如何分块存储的**
- 复制透明：采用复**制技术的分布式方法时**，用户不需要知道复制到了那些节点，和如何复制的
- 位置透明：无需知道数据存放的**物理位置**

- 逻辑透明：无需知道**局部使用的时那种数据模型**
- 共享性：数据存储在**不同的节点数据共享**
- 自治性：每个节点对**本地数据独立管理**
- 可用性：某一个场地故障，**系统可以使用其他场地的副本而不至于整个系统瘫痪**
- 分布性：数据**在不同场地存储**

• 存储过程

- 存储过程是在大型数据库系统中，**一组为完成特定功能的 SQL 语句集**
- 通过提供存储过程让第三方调用，将需要更新的数据传入存储过程，从而避免了**向第三方提供系统的表结构，保证了系统的数据安全**

第 5章 面向对象基础

• 面向对象基本概念

- 如何识别是面向对象：**对象 + 分类 + 继承 + 通过消息**的通信
- 类：**定义了一组大体上相似的对象**，是一组对象的**抽象**
- 类分为三种
  - ◆ 实体类：表示显示世界中的**真实实体**，如人物等
  - ◆ 接口类（边界类）：为用户提供一种与系统合作**交互的方式**，例如 二维码、条形码等
  - ◆ 控制类：用来**控制活动流**，充当协调者
- 对象：是基本的运行时实体，一个对象中**既有属性又有行为**（作用于数据的操作），一个对象通常可由**对象名、属性、方法**组成
- 消息：是**对象之间通信**的一种构造
- 重载：是在同一个位置上一系列 方法名相同，但是**形参类型或者个数不同的方法**
- 封装：是一种信息隐蔽技术，目的是**使对象的使用者和生产者分离**，使对象的定义和实现分开
- 继承：是父类和子类之间共享数据和方法的机制，**一个父类可以有多个子类，这些子类都是父类的特例**，父类描述了子类的公共属性和方法
- 一个子类有一个父类叫**单重继承**，一个子类有多个父类叫**多重继承**，多重继承可能导致子类中存在二义性的成员
- 覆盖：一个继承关系中，**子类以更具体的方式实现从父类继承的方法**，称为**覆盖或者重写**
- 多态：收到消息是对象要予以响应，**不同的对象收到同一个消息可以产生完全不同的结果**，这个现象称为多态，
- **多态的实现受继承支持**，利用继承的层次关系，**将具有通用功能的消**

息放在高层次，将不同的实现放在低层次，这些低层次生成的对象能够给消息不同的响应

- 多态的四类形式：
  - 通用多态：**参数多态**（应用最广泛）、**包含多态**（常见的例子：子类型化）
  - 特定多态：**过载多态**（同一个名字在不同的上下文所代表含义不同）、**强制多态**

• 静态绑定和动态绑定

- 绑定就是**一个方法的调用与调用这个方法的类连接在一起的过程**
- 静态绑定，就是在程序运行前，**编译阶段就能够确定方法由谁调用**
- 动态绑定，在运行时根据具体的对象类型进行绑定，**动态绑定支持继承和多态**

• 面向对象设计原则

- 单一职责原则：**一个类仅有一个引起他变化的原因**，一个类只做一种类型责任
- 开放 －封闭原则：软件实体应该是**可扩展的**（开放），但是**不可修改**（封闭）
- 里式替换原则：**子类必须要能够完全替代父类**
- 依赖倒置原则：**抽象不应该依赖于细节，细节应该依赖于抽象**；高层模块不应依赖于低层模块，**二者都应该依赖于抽象**
- 接口分离原则：接口**属于客户**，不属于它所在的类层次结构，**抽象级别不应有对细节的依赖**
- 共同封闭原则：一个变化如果对一个包产生影响，**则将对该包中的所有类产生影响**，而对于其他包不产生影响
- 共同重用原则：如果重用包中的一个类，**那么就要重用包中的所有类**

• 面向对象分析

- 面向对象分析的目的：**为了获得对应用问题的理解**
- 面向对象分析的五个活动：**认定对象 - 组织对象 - 描述对象间的相互作用 - 确定对象的操作 - 定义对象的内部信息**
- 认定对象：**用来定义问题域**，以自然存在的 名词作为一个对象
- **定义领域模型**是面向对象分析的关键步骤之一，它按照对象分类的角度来创建对象领域的描述，包括**定义概念、属性和重要的关联**，其结果用类图来组织

• 面向对象设计

- 基于面向对象分析的结果，**将分析结果，转化为设计模型**，定义系统构造蓝图
- 面向对象设计：获得对应问题的解决方案，**实现系统，关注技术及实现层面的细节**
- 面向对象设计的五个活动：**识别类及对象 - 定义属性 - 定义服务 - 识别关系 - 识别包**

• 面向对象程序设计

- 实质是选用**一种面向对象语言**，采用 **对象、类等相关概念所进行的程序设计**

• 面向对象测试

- 面向对象测试的四个层次：**算法层 - 类层 - 模板层 - 系统层**

第 6章 UML

• UML 概念

- UML 词汇表包含三种构造块：**事务**（对模型中最具代表性的成分抽象）、**关系**（把事务结合在一起）、**图**（聚集相关的事务）

• UML 事务

- 结构事务：**模型的静态部分**，名词，描述概念或物理元素，包括 类、接口、用例、构件等
- 行为事务：**模型的动态部分**，动词，描述跨越时间空间的行为，包括交互、状态机、活动等
- 分组事务：**模型的组织部分**，最主要的分组事务是包，结构事务、行为事务或者其他分组事务都可以放在包里
- 注释事务：**模型的解释部分**，用来描述、说明、标注，注解是最主要的注释事务

• UML 关系

- 依赖关系：**是两个事务之间的语义关系**，一个事务（独立事务）发生变化，会影响另一个事务（依赖事务）的语义；
  - 图形上通过**虚线 + 箭头**实现
  - A(依赖事务) ·····> B(独立事务) A 依赖于 B， B 事务的变化会引起 A 事务的语义变化
- 关联关系：是一种结构关系，描述了一组链，**链是对象之间的链接**，通常分为**聚合关系和组合关系**，描述了**整体和部分**之间的关联
  - 单向关联：实线 + 箭头 A() ————————> B() ，它使得**一个类知道另一个类的属性和方法**， A 类依赖于 B 对象，并把 B 作

为 A 的一个成员变量，则 A， B 存在关联关系，**关联可以是单向的也可以是双向的**（双向关联不用箭头）

- 双向关联： 实线 A ———————— B ，**关联多重度位于实线的上方**，表示一个 A 类实例可以关联多少个 B 类实例，一个 B 类实例又可以关联多少个 A 类实例，**通常多重度可以用 （ 1 *）（ 1 1…*）等表示** ， 多对多的双向关联，**一般可以将关联关系提取出一个“关联类”**
- 聚合关系 ： **实线 + 空菱形 A(部分)** ————————<> B(整体) 整体的生命周期与部分不同步，**整体消失，部分依然可以存在**
- 组合关系：**实线 + 实心菱形 A(部分)** ————————<+>(实心菱形不好画，用 <+> 代替) B(整体) 整体生命周期与部分同步，**整体消失，部分也消失**

- 泛化关系：是一种 **“特殊” 和 “一般”** 之间的关系，特殊元素（子）的对象可以替代一般元素（父）的对象，**子元素共享了父元素的结构和行为**
  - 图形上通过 **实线 + 空心三角箭头**
  - **A(特殊、子)** ————————|> **B(一般、父)**
- 实现关系：是类元之间的语义关系，其中一个类元指定了由另一个类元保证执行的契约
  - 通常的使用情况：1. **接口和实现他们的类或构件之间**； 2. **用例和实现他们的协作之间**
  - 图形上通过 **虚线 + 空心三角箭头**
  - A(类) · · · · · · · · |> B(接口)

• 类图（静态）

- 类图：展现了**一组对象、接口、协作**和他们之间的关系，类图中可以包含注解和约束，也可以有包或子系统
- 类图对静态设计视图建模的三种方式

- 对系统**词汇建模**
- 对简单的**协作建模**
- 对逻辑**数据库模式建模**

- 类图中类的组成
  - 第一层 **类名**
  - 第二层 **属性名** （属性名 1: 属性的类型 ）
  - 第三层 **方法名** （方法名 1(): 方法返回值类型），其中属性名和方法名前可以有修饰符 **+ public 公有的； - private 私有的； # protected 受保护的；**

• 对象图（静态）

- 对象图：展现了**某一时刻**，一组对象及他们之间的关系
- 对象图中对象与类图中类的区别：对象分为两层，第一层是 **对象名**（对象名 :类名 ）， 第二层 **属性**

• 用例图（静态）

- 用例图：展现了一组用例、参与者、以及他们之间的关系
  - 参与者：参与者是与系统交互的外部实体，可以是使用者或者参与系统交互的外部系统，基础设备等
  - 用例：是从用户角度描述的系统行为，用例是一个类，代表了一类功能，而不是该功能的某一个具体实现
- 包含关系：用例与用例之间的关系，图形上使用虚线 +箭头 +《 include 》表示： A （基本用例） •••《 include》 •••> B （被包含用例） ，箭头指向被包含的用例， A用例包含 B用例，则 A执行用例 B一定也会被执行
- 扩展关系：用例与用例之间的关系，图形上使用虚线 +箭头 +《 extend 》表示： A （扩展用例） •••《 extend》 •••> B （被扩展用例） ， 箭头指向被扩展的

用例，被扩展后的用例 B 在执行时可能会遇到特殊的情况或者可选的情况，这个时候就可以用 扩展用例

- 包含关系与扩展关系区分：
  - ◆ 区分包含关系：使用某个用例，必然会使用另外一个用例
  - ◆ 区分扩展关系：当执行某个用例，不一定要去执行另外一个用例

• 序列图（动态）

- 序列图，描述了**以时间顺序**组织的对象之间的交互活动，序列图是**对一个用例进行详细过程的分解**
- 图形上，参与交互的对象放在**图的上方**，沿水平方向排列，**发起交互的对象放左边**；然后把对象间发送和接收的消息，**按照时间顺序由上到下排列**
- 对象生命线：由对象起始的一条**垂直向下的虚线**，表示对象在**一段时间内存在**
- 控制焦点：对象生命线之上的**一段瘦高的矩形**，表示对象**执行一个动作所经历的时间段**
- 调用消息用**实线＋箭头**表示，返回消息用 **虚线 ＋箭头**表示
- 调用消息所要执行此消息方法的是**箭头指向的对象**

• 通信图（动态）

- 通信图也成**协作图**，强调**收发消息对象的结构组织**
- 通信图与序列图的不同，在于通信图**有路径**、通信图**有顺序号**，延同一个链可以**展示许多消息**，每个消息都**有唯一的顺序号**
- 通信图与顺序图是**同构**的，**可以相互转换**
- 通信图展现了对象间的消息流及其顺序

• 状态图（动态）

- 状态图展现了一个**状态机**，它由**状态、转换、事件和活动**组成
- 状态图对系统的**动态方面**建模
- 当对**系统、类或用例的动态方面建模**时。通常是对**反应型对象建模**
- 状态：任何可以被观察到的**系统行为模式**，一个状态代表系统的一种行为模式
  - 状态分为**初态（实心圆）终态（实心圆外再加一层圆）和中间状态**
  - 状态图中的状态是一个圆角矩形，第一层是**状态名**，中间一层时**状态变量**（可以没有），最后一层为**活动表**（也可以没有）
  - 状态之间用**带箭头的线表示 “转换（迁移）”** 箭头线上的事件发生时，转换开始
  - 一个状态图**只能有一个初态，可以有多个终态或者没有终态**
- 活动：由 **“事件名 / 动作表达式”** 组成 位于状态的活动表中 ，有如下三种标准事件
  - **entry： 入口动作，进入状态，立即执行**
  - **do： 内部活动，占用有限时间，可以中断工作**
  - **exit:出口动作，退出状态，立即执行**
- 事件：某个特定时刻发生的时间，它是对引起系统做动作，和**一个状态转换成另一个状态的事件的抽象**
  - 转换由 **“事件（监护条件） / 动作”** 组成
  - 转换包括**两个状态**
  - 事件**触发转换**
  - 活动（动作）可以在**状态内执行**也可以在**状态转换时执行**
  - 监护条件是**一个 boolean 表达式**
  - **“事件”发生且“监护条件为真”状态转换才发生**，状态转换开始后才会**执行“动作”**
- 组合状态：一组状态转换作用矩形包围，**作为另一个状态图中的一个**

状态存在

- ◆ 组合状态中的**所有子状态完成**，才会走组合状态外的其他状态

• 活动图（动态）

- 活动图展现了**系统从一个活动到另外一个活动的流程**，对系统的功能建模很重要，强调了对象间的控制流程
- 活动图有**起始、终止、圆角矩形**组成的活动、**实线 +箭头组成的流**、并发分叉、并发汇合、分支、分支上的监护表达式组成
- 用活动图来对**工作流建模、对操作建模**
- **并发分叉后直接相连**的活动可同时执行

• 构件图（静态）

- 构件图也称为**组件图**，展现了**一组构件之间的组织和依赖**
- 与**类图相关**，通常把构件映射为**一个或多个类、接口或者协作**
- 构件图有**特殊的标记**，构件图之间通过 **供接口（空心圆）和 需接口（圆弧）对接**，供接口**提供相应的方法实现**，需接口来**调用这个方法**

• 部署图

- 部署图用来对**系统物理层面建模**
- 部署图立体图形，**《 artifact》表示制品**
- 部署图展现了**系统软件和硬件间关系**，在**实施阶段**使用
- 部署组件之间的关系**类似包依赖**

• 总结

- 静态建模：**类图、对象图、用例图**
- 动态建模：**序列图、通信图、状态图、活动图**
- 物理建模：**构件图、部署图**

- 交互图：**序列图（顺序图、时序图）、通信图（协作图）**

第 7章 设计模式

• 创造型设计模式

- 概念：创造型设计模式**抽象了实例化过程**，帮助一个系统如何创建、组合、和表示那些对象

• 简单工厂模式

- 定义一个工厂类，可以根据**传入的参数的不同，返回不同类的实例**，这些**“不同类”**继承自同一个父类
- 工厂类中创建实例的方法通常是**一个静态方法**，因此又称为静态工厂
- 使用者无需知道**产品对象是如何创建的**

• 工厂方法模式（ Factory Method ）

- 在简单工厂基础上，定义一个用于**创建对象的工厂接口**，让实现这个接口的**子类来决定实例化哪个类**
- 工厂方法模式使**一个类的实例化延迟到了子类**
- 适用性：
  - ◆ 当一个类**不知道它所必须创建的对象的类的时候**
  - ◆ 当一个类希望**由他的子类来指定它所创建的对象的时候**

• 抽象工厂模式 (Abstract Factory)

- 意图：**提供一个创建一系列相关或相互依赖对象的接口**，而无需指定他们具体的类
- 可以理解为：“工厂方法” 创建的是**一中对象**，“抽象工厂” 创建的是**一系列对象**
- 适用性：（抽工独立组合多个产品，联合显示接口不实现）

- ◆ 一个系统要**独立于它的产品创建、组合和表示时**
- ◆ 一个系统要**由多个产品系列中的一个来配置时**
- ◆ 当要强调一系列相关产品对象的设计以便**进行联合使用时**
- ◆ 当提供一个产品类库，**只想显示他们的接口而不是实现时**

• 生成器模式 (Builder)

- 意图：将**一个复杂对象的构建和他的表示分离**，使得同样的构建过程**可以创建不同的表示**
- 理解：定义**一个抽象**的生成器；再定义多个**具体的生成器类**（封装对象复杂的算法）来继承它；定义**一个管理者**（对象装配）来操作生成器，**这样管理者与不同的生成器类组合就可以创建不同的产品表示**
- 适用性：（生成复杂算法，构造对象不同）
  - ◆ 当创建对象的**复杂的算法**应该**独立于该对象的组成部分和装配方式时**
  - ◆ 当构造过程**必须允许被构造对象有不同表示时**

• 原型模式 (Prototype)

- 意图：用原型实例指定**创建对象的种类**，通过**复制**这些原型创建新的对象
- 适用性：（原型独立构成，运行指定，不同组合状态）
  - ◆ 当一个系统应该**独立于它的产品创建、构成和表示时**
  - ◆ 当要实例化的类是**在运行时刻指定的**，例如动态装载
  - ◆ 当一个类的实例**只能有几个不同状态组合中的一种**

• 单例模式 (Singleton)

- 意图：保证**一个类仅有一个实例**，并提供**一个访问它的全局访问点**
- 适用性：（单例有一个实例）

- 当类**只能有一个实例**，且客户从一个众所周知的访问点访问它时

• 结构型设计模式

- 概念：涉及如何组合类和对象，以获得更大的结构

• 适配器模式 (Adapter)

- 意图：将一个类的接口**转换成用户希望的另一个接口**，使得原本由于接口不兼容不能一起工作的那些类**可以一起工作**
- 适用性：（适配接口不符合）
  - 想使用一个已存在的类，但是**接口不符合要求**

• 桥接模式 (Bridge)

- 意图：将**抽象部分与实现部分分离**，使他们都可以**独立的变化**
- 理解：通过**聚合或组合关系**，将一个有多种组合情况的类**拆分成几个相互关联的类**，这个每个类各自可以独立变化
- 适用性：（桥接绑定以扩充，不影响不编译类层次）
  - 不希望在**抽象和他的实现部分**之间有一个固定的**绑定关系**
  - 类的抽象和它的实现都可以通过**生成子类的方法加以扩充**
  - 对抽象的实现部分的**修改应该对客户不产生影响，不必重新编译**
  - 有许多类要生成的**类层次结构**

• 组合模式 (Composite)

- 意图：将对象组合成**树型结构**，以表示 **部分－整体** 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
- 理解：以**文件夹与文件间**的关系去理解
- 适用性：（组合部分整体，使用组合对象）
  - 想要表示对象的**部分－整体层次结构**

- 希望用户**忽略组合对象与单个对象间的不同**，统一的**使用组合中的所有对象**

• 装饰器模式 (Decorator)

- 意图：动态的**给一个类添加一些额外的职责**
- 适用性：（装饰添加和撤销职责，不能扩充）
  - 在不影响其他对象的情况下，动态透明的方式**给单个对象添加职责**
  - 处理**那些可以撤销的职责**
  - 当**不能采用生成子类的方式进行扩充时**

• 外观模式 (Facade)

- 意图：为子系统中的一组**接口提供一个一致的界面**，定义一个**高层接口**，使得子系统更加**容易使用**
- 适用性：（外观子系统简单接口，依赖构建层次结构入口点）
  - 要为一个**复杂的子系统提供一个简单的接口时**
  - 客户程序与**抽象类的实现部分**之间存在很大的**依赖性**
  - 当需要**构建一个层次结构的子系统**时，使用外观模式**定义子系统每层的入口点**

• 享元模式 (Flyweight)

- 意图：运用**共享技术**，有效的支持**大量细粒度的对象**
- 理解：用一个享元工厂来创建并维护实例，**一种实例只创建一个**，后续创建都是**直接返回已创建的实例**
- 适用性：（享元大量开销，外部状态取代多组对象）
  - 一个应用程序使用了**大量的对象**
  - 由于使用了大量的对象，造成了**很大的存储开销**

- 对象的大多数状态都可以变为**外部状态**
- 如果删除对象的外部状态，那么可以用**相对较少的共享对象取代很多组对象**

• 代理模式 (Proxy)

- 理解：为其他对象**提供一种代理**，以**控制**对这个对象的访问
- 适用性：（代理简单的指针）
  - 在需要比较通用和复杂的对象指针**代替简单的指针的时候**

• 行为型设计模式

- 概念：涉及**算法和对象间职责的分配**，描述了它们之间的**通信模式**，使用继承机制在对象间分配行为

• 责任链模式 (Chain of Responsibility)

- 意图：使多个对象都有**机会处理请求**，从而避免请求的发送者和接收者之间的**耦合关系**，将对象连城一条链，**沿着链传递请求**，直到**有一个对象处理它为止**
- 适用性：（责任链请求自动确定，不明确接收者动态指定）
  - 有**多个对象处理一个请求**，哪个对象处理该请求在**运行时自动确定**
  - 想在**不明确指定接受者**的前提下向多个对象中的**一个提交请求**
  - 可处理一个请求的对象集合**被应动态指定**

• 命令模式 (Command)

- 意图：将**一个请求封装成一个对象**，从而使得可以**用不同的请求对客**

户进行参数化，对**请求排队、记录请求日志、以及支持可撤销**的操作

- 适用性：（命令参数化，指定排列，取消操作修改）
  - ◆ 抽象出待执行的动作以**参数化某对象**
  - ◆ 在不同的时刻**指定、排列和执行请求**
  - ◆ 支持**取消操作、修改日志**

• 解释器模式 (Interpreter)

- 意图：给定一个语言，**定义它的文法的一种表示**，并定义一个解释器，解释语言中的句子
- 适用性：（解释抽象语法树）
  - ◆ 当一个语言**需要解释**执行时，可将语言中的句子表现为**一个抽象语法树**

• 迭代器模式 (Iterator)

- 意图：提供一种**方法顺序**的访问一个**聚合对象**中的各个元素，且**不需要暴露该对象的内部表示**
- 适用性：（迭代聚合无需暴露。多种遍历统一接口）
  - ◆ 访问一个**聚合对象的内容而无需暴露它的内部表示**
  - ◆ 支持**对聚合对象的多种遍历**
  - ◆ 为遍历不同的聚合结构**提供一个统一的接口**

• 中介模式 (Mediator)

- 意图：用一个**中介对象来封装一系列的对象交互**，中介者使得各个对象**不需要显示的相互引用**，从而使其**耦合松散**，而且可以独立的**改变它们之间的交互**
- 适用性：（中介复杂通信，依赖难以理解）
  - ◆ 一组对象定义良好，但是以**复杂的方式进行通信**，产生的**相互依**

赖关系结构混乱且难以理解

• 备忘录模式 (Memento)

- 意图：在**不破坏封装性**的前提下，捕获**一个对象的内部状态**，并在对象之外**保存**，这样以后就可以将**对象恢复到原先保存的状态**
- 适用性：（备忘录某一时刻状态，破坏封装性）
  - 必须保存**一个对象在某一时刻的（部分）状态，这样以后在需要时才能恢复之前的状态**
  - 如果用**一个接口**来让其他对象直接得到这些状态，将会暴露对象的实现细节，**并破坏对象的封装性**

• 观察者模式 (Observer)

- 意图：定义对象间的**一对多的依赖关系**，当一个对象发生改变时，所有**依赖它的对象都得到通知，并自动更新**
- 适用性：（观察者改变其他对象，不是耦合的）
  - 当一个对象的改变**需要同时改变其他对象**，而不知道具体有多少对象待改变时
  - 当一个对象**必须通知其他对象**，而它又不能假定其他对象是谁，即**不希望这些对象是紧耦合的**

• 状态模式 (State)

- 意图：允许一个对象再其**内部状态改变时改变它的行为**，对象似乎修改了它的类
- 适用性：（状态改变行为，多分支语句依赖状态）
  - 一个对象的**行为决定于它的状态**，并且必须在运行时刻**根据状态改变它的行为**
  - 一个操作中含有**庞大的多分支条件语句**，这些分支**依赖**于该对象

的状态。

• 策略模式 (Strategy)

- 意图：定义**一系列算法**，把他们一个个封装起来，并使得他们**可以相互替换**，此模式使得算法可以**独立于使用他们的客户而变化**
- 适用性：（策略多个行为，算法变体避免暴露，多条件语句）
  - ◆ 许多相关的类仅仅是**行为有异**，策略提供了一种用**多个行为**中的一个行为来配置一个类的方法
  - ◆ 需要使用**一个算法的不同变体**
  - ◆ 使用算法的客户**不应该知道数据结构，避免暴露**
  - ◆ 一个类中定义了**多种行为**，这些行为在这个类的操作中以**多个条件语句的形式出现**，将相关的条件分支移入他们各自的策略类中以代替这些条件语句

• 模板方法 (Template Method)

- 定义一个操作中的**算法骨架**，而将一些步骤**延迟到子类中**，使得一个子类可以**不改变一个算法的结构即可重新定义该算法的某些步骤**
- 适用性：（模板可变给子类，避免代码重复，子类扩展）
  - ◆ 一次性实现**一个算法的不变的部分**，并将**可变的行为留给子类实现**
  - ◆ 各子类中的公共行为应被**提取出来并集中到一个公共的父类**，以避免代码重复
  - ◆ 控制**子类扩展**，模板方法仅允许在特定点进行扩展

• 访问者模式 (Visitor)

- 意图：表示一个作用于**某对象结构中的各元素的操作**。允许在不改变各元素类的前提下，**定义作用于这些元素的新操作**

- 适用性：（访问者依赖具体类，不相关对象的类，定义新的操作）
  - 一个对象结构包含很多类对象，他们有不同的接口，而用户想对这些对象实施一些**依赖于其具体类的操作**
  - 需要对一个对象结构中的对象，进行**很多不同的并且不相关的操作**，而有不想这些**操作污染这些对象的类**
  - 定义对象的类很少改变，但**经常需要在此结构上定义新的操作**

第 8章 信息安全

• 防火墙

- 防火墙建立在**内外网络边界上**的过滤封锁机制，认为**内部网络是安全可信赖的，外部网络是不安全和不可信任**的
- 防火墙对通过**受控干线**的任何通信进行安全处理，例如：**控制、审计、报警、反应**等
- DMZ（屏蔽子网防火墙）：位于**内网与外网**之间，通常作为**隔离区**，在这里可以放置一些**公用服务器**，例如 **web 服务器**、 **E-mail**、 **FTP** 等
- 包过滤防火墙：通过一个**包过滤器**，根据数据的包头中各项信息来控制站点、网络之间的访问性
  - 包过滤防火墙对**用户完全透明、访问速度快、低水平控制**
  - 包过滤防火墙处在**网络层和数据链路层之间**
  - 每个 IP 字段都被检查：**源地址、目的地址、协议、端口**
  - 缺点：**不能防黑客攻击、不支持应用层协议、访问控制粒度粗糙、不能处理新的安全威胁**
- 应用代理网关防火墙：**彻底隔绝内外网之间**的直接通信，内外网之间的互相访问**需要经过应用层代理软件转发**
  - 优点：可以**检查应用层、传输层、网络层的协议**特征，对数据包的**检测能力较强**
  - 缺点：**难以配置、处理速度较慢**
- 状态检测技术防火墙：结合了包过滤防火墙和应用代理网关防火墙两者的优点，即**安全性和高速度**

• 病毒

- 计算机病毒特性：**传播性、隐蔽性、感染性、潜伏性、触发性、破坏**

性

- Worm：**蠕虫病毒**；Trojan：**特洛伊木马**；Backdoor：**后门病毒**；Macro：**宏病毒**
- 宏病毒感染对象：**文本文档、电子表格**
- 木马软件：**冰河**
- 木马感染流程：**通过软件下载捆绑等方式**，在用户主机上建立木马病毒的服务器端，这个木马病毒的服务器端再与攻击者主机上的木马病毒客户端建立网络连接，这样攻击者就**可以借由此来盗取或者破坏用户主机数据**
- 蠕虫病毒：**欢乐时光、熊猫烧香、红色代码、爱虫病毒、震网**

• 网络攻击

- 拒绝服务攻击（Dos 攻击 ）：通过**不断向计算机发起请求**，让目标服务器**没有资源**去接收其他正常的请求，从而达到**“使计算机或者网络无法提供正常服务”**的目的
- 重放攻击：攻击者发送**一个目标主机已经接收过的报文来达到攻击目的**，主要用于**身份认证过程、破坏认证正确性**；可通过在报文中增加时间戳来防范
- 口令入侵攻击：使用某些**合法的用户账号和口令登录到目的主机**，然后实施攻击活动
- 特洛伊木马：用**户下载了含有木马的软件后**，这个木马程序就会向**黑客发起连接请求**，建立连接后，黑客就可以实施攻击
- 端口欺骗攻击：**采用端口扫描找到系统漏洞**，从而实施攻击
- 网络监听：攻击者可接口某一网段在统一**物理通道上传输的所有信息，截取账号和口令**
- IP 欺骗攻击：**伪造源 IP 地址**，冒充其他系统或发件人身份
- SQL 注入攻击：通过注入**某些 SQL 查询代码，获得数据库权限，从而窃取及修改信息**
- 入侵检测技术：**专家系统、模型检测、简单匹配**

• 网路安全

- SSL（安全套接层）：传输层安全协议 端口号 443
- TLS（传输层安全协议）：也是传输层安全协议，是 SSL 3.0 的后续版本
- SSH：用于**终端设备与远程站点之间**建立连接的安全协议，建立在**应用层和传输层基础上的按全协议**
- HTTPS：使用 SSL 加密的 http
- MIME：**电子邮件扩展相关协议，不具有安全性**
- PGP：通过 RSA **非对称加密的邮件协议**
- IPSec：对 IP **数据报文进行加密**
- ARP：**地址解析成物理地址协议**
- Telnet：**不安全的远程登录协议**
- WEP：**有限等效保密协议**
- TFTP：**简单文件传输协议**
- PP2P：**链路加密**
- RFB：**远程登录图形化用户界面协议**
- IGMP：**因特网组管理协议**
- 信息安全的五个基本要素：**机密性、完整性、可用性、可控性、审可查性**

第 9章 计算机网络

• 网络设备

- 物理层的互连设备：中继器 (Repeater) 和集线器 (Hub)，其中集线器可以看做是一种多端口的中继器
- 数据链路层互连设备：网桥（ Bridge ）和交换机（ Switch ） ，其中交换机是多端口的网桥
- 网络层互连设备：路由器
- 应用层互连设备：网关
- 物理层设备不能隔离广播域和冲突域，数据链路层设备可以隔离冲突域不能隔离广播域，网络层可以隔离广播域与冲突域

• TCP/IP 协议簇分类

- 网络层协议

IP ：一种尽力传送的通信协议，传送的数据可能丢失

ICMP: Internet 控制信息协议，利用 IP 传送报文

- 传输层协议：TCP UDP，都是基于 IP 之上的
- 应用层协议
  - FTP: 文件上传协议，文件上传的端口是 20 ，控制端口为 21
  - SNMP ：网络管理协议
  - 方便记忆： 1. 名字带 ″IP″ “AP” 的都是网络层， 2. 应用层协议所有带 ″T″ 的，除了 TFTP 都是基于 TCP 的，其他是基于 UDP 的，不带 “ T” 的只有 POP3 是 TCP 的，其他都是 UDP 的

• 网络层协议 IP TCP UDP

- IP 提供的服务，是无连接（指没有确定目标系统在已做好接收准备前就发送数据）、不可靠的（目的系统不对成功接收的分组进行确认）
- TCP 面向连接、可靠的传输控制协议，采用三次握手实现可靠性
- 可靠传输、连接管理、差错校验、重传、流量控制（可变大小滑动窗口协议）、端口寻址、
- UDP 无连接、不可靠的传输协议，可以保证应用程序进程间的通信，有助于提高传输的高效率
  - ◆ 端口寻址

• 电子邮件服务协议

- SMTP: 简单邮件传输协议，用来发送邮件，端口 25 ，基于 TCP，只能传输文本和 ASCII码的文件
- SMTP 基于 C/S 模式，即 客户端 /服务器模式来进行通信
- MIME 邮件附件扩展类型
- PEM 私密邮件
- POP3 用来接收邮件的协议，基于 TCP ，端口号 110 基于 C/S 模式通信

• 地址解析 ARP RARP

- ARP: 地址解析协议，将 IP 地址转换为物理地址（ MAC 地址，每个网卡唯一）
- RARP ：反地址解析协议，将物理地址转换为 IP 地址
- 计算机间使用 ARP 通信流程 PC1 向 PC2 通信
  - ◆ 查询 ARP 高速缓存
  - ◆ 如果有 PC2 的 IP 地址缓存，则直接使用其对应的物理地址发送

数据

- 如果没有缓存，则在局域网以广播形式发送 ARP 请求包
- 如果局域网上某个计算机 IP 地址一致，则那台计算会响应一个 ARP 应答，包含对应的物理地址
- ARP 将 IP 及应答的物理地址存到高速缓存中

• DHCP

- 动态主机配置协议，集中管理分配 IP 地址，使网络中的主机获得，IP 地址、网关地址、 DNS 地址、 DHCP 服务器地址等

• URL

- 协议名 ://主机名 . 域名 . 域名后缀 . 域名分类 / 目录 / 网页文件

• DNS 域名查询次序

- 本地 hosts 文件 》 本地 DNS 缓存 》本地 DNS 服务器 》根域名服务器

• 主域名服务器在接收到请求后的查询次序

- 本地缓存 》 本地 hosts 文件 》本地数据库 》 转发域名服务器

• IP 地址和子网划分

- Internet 中网络分为 5 类 A、 B、 C、 D、 E
- A 类网络，网络地址 8 位（第一位为 0 ），其余为主机地址
- B 类网络，网络地址 16 为（前两位为 10 ），其余为主机地址
- C 类网络，网络地址 24 为（前两位为 110），其余为主机地址

- 子网掩码：识别报文是只存放在网络内部，还是被路由转发到其他地方，用 1 表示网络地址， 0 表示主机地址，例如 C 类掩码为 11111111.11111111.11111111.00000000 即 255.255.255.0
- 子网划分：
- 将一个网络划分成多个子网：取部分主机号当子网号，取了多少位 k 就有 2^k 个子网
- 将多个网络合并成一个大网络：去部分网络号主机号
- 222.125.80.128/26 中 /26 表示由 26 位的网络地址，有 32-26 位的主机地址
- 全 1 是广播地址，全 0 是网络地址

• IPv6

- IPv4 为 32 位， IPv6 128 位，地址空间 不可用完

• 无线网路

- 无限网络中蓝牙覆盖范围最小，通信距离最短

• Windows 命令

- ipconfig: 显示所有网络适配器的 IP 地址、子网掩码和缺省网关值
- ipconfig/release: 释放 IP 地址
- ipconfig/flushdns: 清除 dns 缓存，或刷新 dns
- ipconfig/displaydns: 显示 dns
- ipconfig/registerdns: DNS 客户端手工向服务器注册
- ipconfig/all: 显示所有 所有网络适配器的完整 TCP/IP 配置信息，包括 DHCP 服务是否启动
- ipconfig/renew: DHCP 客户端刷新，重新申请 IP

• 路由

- 五种路由类型
  - ◆ 主机路由：子网掩码 255.255.255.255
  - ◆ 直连网络
  - ◆ 远程网络
  - ◆ 默认路由：目标网络和网络掩码都是 0.0.0.0
  - ◆ 持久路由
  - ◆ 服务器接收到一个 IP 数据包时先查找主机路由，在查找网络路由（直连网络、远程网络）、最后查找默认路由
- 如果路由器收到关于某个目标的多条路由，那么要比较各个路由的管理距离，并采用管理距离最小的

• 其他

- 网络的可用性：用户可利用网络时间的百分比
- 园区子系统：链接各个建筑物的通信系统
- DNS 实现负载均衡：为同一个域名设置多个主机记录，启用循环，添加每个 web 服务器的主机记录
- 要使得两个 IPv6 通过现有的 IPv4 网路通信需要使用“隧道技术”
- 要使得 IPv6 与 IPv4 通信，需要使用 “翻译技术”
- DNS 服务器与计算机不在一个子网，不会导致计算机网络不可访问，只要路由可达 DNS 都可以正常工作
- 层次化局域网模型核心层的主要功能：将分组从一个区域高速的转发到另一个区域
- 默认网关要和当前 IP 地址在同一个子网段中

第 10章 操作系统

• 操作系统地位

- 计算机系统由软件、硬件组成，没有配置软件的称为裸机
- 操作系统地位：计算机硬件 》操作系统 》 系统软件 》 应用软件 》 用户
- 所有其他软件，如编译程序、汇编程序、数据库管理系统等，以及大量应用软件都是建立在操作系统之上的
- 把操作系统看做用户与计算机之间的接口

• 进程管理

- 进程是源分配和独立运行的基本单位
- 进程管理重点是要研究诸进程之间的并发特性，以及进程间相互合作与资源竞争产生的问题

• 前趋图

- 有向无循环图，由节点和单向边组成，节点代表各个程序段的操作，单向边表示前前趋关系 Pi(节点、前趋) ——> Pj(节点、后继)，Pi 执行结束 Pj 才能执行
- 前趋图，有 n 个箭头就要设置 n 个信号量，按照从小到大顺序写到图中，箭头方向是 P 操作，箭头尾部是 V 操作
- 程序顺序执行的主要特征：顺序性、封闭性（独享资源）、可再现性
- 程序并发执行的主要特征：失去程序封闭性、程序和机器执行程序的活动不再一一对应、并发程序之间的相互制约性

• 进程的三态模型

- 在多道程序系统中，进程在处理器上交替运行，一般由三种状态 运行 就绪、阻塞
- 运行：当一个进程在处理机（ CPU ）上运行时，就是运行态
- 就绪：一个进程获得了除了处理机（ CPU ）外的所有资源 ， 一旦得到处理机即可运行，这个时候就是就绪态
- 阻塞：等待、睡眠，一个进程正在等待某个事件（例如等待 I/O 完成）发生而停止运行

• 同步与互斥

- 同步是合作资源进程之间直接制约的问题
- 互斥是申请临界资源进程间的间接制约问题

• 临界区管理原则

- 有空即进，临界区无进程，则允许进入，且只能在临界区运行有限时间
- 无空则等，临界区有进程，其他进程则要等待
- 有限等待，在外等待的进程，要保证在有限时间可进入
- 让权等待，进程有 CPU 没有资源时，不能进入自己的临界区，要立即释放 CPU 资源，避免忙等

• 信号量机制

- 信号量 S 的物理意义， S>=0 表示资源的可用数， S<0 表示等待该资源的进程数

• PV 操作是实现同步和互斥的常用方法

- P 表示申请一个资源（ S = S-1，可以理解为从信号量 S 中申请出一个来使用，申请后 S<0 则进程转为阻塞态，插入阻塞队列），
- V 表示释放一个资源（ S = S+1，可以理解为 释放一个资源到信号量，释放后， S<=0 则从阻塞队列唤起一个进程，插入就绪队列）

• PV 操作实现进程互斥

- 令信号量 mutex 初值为 1，当进入临界区时 执行 P操作，退出临界区时执行 V 操作，这两就利用 PV 实现代码互斥

• PV 操作实现进程的同步

- 单缓冲区同步（缓冲区只能放一个产品 ）：分为生产者和消费者，需要设置两个信号量 ，S1 初值为 1表示可放入缓冲区的产品数， S2 初值为 0，表示可从缓冲区取出的产品数；每次生产产品 要进行 P(S1) 和 V(S2)，每次消费产品要进行 P(S2) V(S1)
- 多缓冲区同步（缓冲区可以放多个产品 ）：在 单缓冲区的基础上增加一个信号量 S，名为互斥信号量，初始值为 1，标记缓冲区可操作的量（缓冲区是个互斥资源）；每次生产产品及消费产品是都要增加一个 P(S1) P(S) V(S) V(S2) 操作 ，S 的 PV 操作放中间

• 死锁

- 同类资源分配不当引起的死锁：若采用的资源分配策略是，轮流的为每个进程分配，则可能会导致分配几轮后，没有一个进程达到所需的资源数，这个时候，每个进程都在等待资源分配，形成死锁；
- 同类资源分配不当引起的死锁的解法： m 为资源总量、 n 为进程数、 k 为每个进程需要的资源，满足 m >= n * (k-1) + 1 就可以避免死锁

• 进程资源图

- Pi 代表进程， Ri 代表资源类型；每个 Ri 可以有多个资源； 指向进程的箭头表示分配资源；指向资源的箭头表示申请资源；
- 先分配资源再申请资源，经过分配申请后没有满足资源的进程即为“阻塞”
- 是否可化简：取决于是否可以在某个进程完成后释放资源，并使得后续进程得以完成
- 可化简的就是非死锁的

• 死锁避免

- 死锁处理策略：鸵鸟策略（不理睬）、预防策略、避免策略、检测与解除死锁
- 死锁避免算法：银行家算法，即在每次分配资源前检测分配资源后系统是否安全（是否安全取决与分配资源后，系统是否可以有某种进行序列，来将所有的进程都执行完），资源利用率高，但增加了检测的开销
- 银行家算法题计算：， 1. 先算出仍需资源数， 2. 在算出，剩余资源数

• 线程

- 进程在创建、撤销、切换中，系统会付出较大的时空开销，故系统引入的进程不易过多，进程切换频率不易太高，因此引入了线程
- 线程作为调度和分配的基本单位，进程作为独立分配资源的单位，线程是进程中的一个实体
- 线程与线程之间不可见，但是线程与线程可以共享进程的资源

• 局部性原理

- 时间局限性：程序的某一条指令执行，那在不久的将来该指令可能被再次执行，如果某个存储单元被访问，那不久的将来还可能被再次访问
- 空间局限性：程序访问了某个存储单元，不久的将来其附近的存储单元也可能被访问

• 相关题型“淘汰”问题

- 在内存中才能被淘汰
- 先淘汰未访问过的
- 再淘汰未修改过的

• 分页存储管理

- 纯分页存储管理的地址结构： n 位的页号 + m 位的页内地址
- 计算机的页面大小为 4k => 则代表 n 位页内地址 2^n = 4 * 1024 => n=12
- 页面变换表逻辑地址转物理地址 => 逻辑地址即为纯分页存储管理的地址结构，由 n 位的页号 + m 位的页内地址组成 => 页内地址组成不变，将页号替换成“页面变换表”中对应的物理块号即可

• 段页式存储管理

- 段页式存储管理地址结构： n 位段号 + k 位段内页号 + m 位的页内地址

• 单缓冲区

- 缓冲区只能有一个“作业”，缓冲区为空时可以输入，缓冲区有作业

时可以传送

- I/O 设备 —输入 (T)—> 缓冲区 —传送 (M)—> 工作区 （处理 C)
- 计算 n 个作业单缓冲区所花时间： (T+M)*n + C

• 双缓冲区

- 缓冲区有两个，每个可存一个”作业“
- 计算 n 个作业双缓冲区所花时间： T*n + M + C

• 磁盘调度算法

- 先来先服务（ FCFS ） ： 按请求访问者的先后顺序来启动磁盘驱动器，平均寻道长度大
- 最短寻道时间优先（ SSTF ） ：让距离当前磁道位置最短的先执行，不考虑访问者的先后顺序
- 扫描算法或者电梯调度算法（ SCAN ） ： 从磁头当前位置开始，沿着磁头移动方向，选择最近的柱面，如果磁头移动方向无请求柱面，则调转方向，选择最近的
- 循环扫描算法（CSCAN） ： 在扫描算法的基础上，调转方向后不再选择最近的柱面，而是移动到最里端

• 旋转调度算法

- 磁盘旋转不会停下，磁盘旋转完一个扇区，就代表读取了一个扇区的记录，记录读取后的处理时间内，磁盘不会停下
- 如果是顺序处理 n，总时间 =（读时间 + 扇区一圈时间)*(n-1) + 第一个扇区的读时间 + 第一个扇区的处理时间
- 优化处理：重排扇区，让第一个扇区处理后所停在的位置，在第二个记录所在扇区的起始位置，所耗时 =（读时间 +处理时间)*n

• 多级索引结构

- 直接地址索引：索引从 0 开始，一个地址项指向一个磁盘数据块
- 一级间接地址索引：一个地址项指向一个磁盘索引块（也可以叫一级索引块），一个磁盘索引块又包含很多个地址项，磁盘索引块中的地址项指向一个磁盘数据块
- 二级间接地址索引：比一级间接地址索引，多了一级磁盘索引块

• 文件目录

- 为了实现按名存取，系统为每个文件设置用于描述和控制的数据结构，至少包含文件名和存放文件的物理地址，这个结构称为文件数据块 FCB
- 文件目录是由文件控制块组成的，用来文件检索
- 文件控制块包含三类信息
  - ◆ 基本信息：文件名、文件物理地址、文件长度、文件块数等
  - ◆ 存取控制信息：文件存取权限
  - ◆ 使用信息：建立日期、最后修改日期、当前使用信息
  - ◆ 目录文件的修改时发生崩溃对系统的影响很大

• 目录结构

- 多级目录结构：倒置的有根树，也称为树形目录结构
- 全路径名：从根目录开始，一直到文件名 D:\
- 绝对路径：从根目录开始，最后是 /
- 相对路径：从当前目录开始，最后是 /

• 位视图

- 位视图用二进制来表示一个物理块的使用情况， 0 表示空闲 1 表示使用

- 位视图的大小由磁盘空间大小（物理块数）决定，位视图描述能力强，适合各种物理结构
- 假设计算机系统 n 位，那位视图第 0 个字能对应存储器上的第 0~n-1 号物理块，第 1 个字能对应存储器上的第 n~2n-1 号物理块后边以此类推

• 其他

- 可变式分区分配方案：进程 P 有上邻空闲区 或 有下邻空闲区，那么 P 进程释放后，空闲区合并成一个
- 当用户通过鼠标或键盘进入某应用系统时，中断处理程序最先获得键盘或鼠标的输入信息
- 实时操作系统的实时是指，计算机对于外来信息能够以足够快的速度处理，并在被控制对象允许的时间范围内做出快速响应
- I/O 系统的层次结构：硬件 -》中断处理程序 -》设备驱动程序 -》设备无关程序 -》用户进程
- I/O 软件隐藏了 I/O 操作的实现细节，方便用户使用
- 磁盘调度管理中，先进行移臂调度在进行旋转，访问不同柱面信息时要先移臂调度，访问同一磁道只需要进行旋转

第 11章 结构化开发

• 模块化

- 模块化是指将一个待开发的软件分解成若干个小的简单部分 - 模块，每个模块独立开发测试

• 模块独立

- 模块独立是指每个模块完成一个相对独立的特定子功能，并且与其他模块间的联系简单，衡量模块独立的标准有两个：内聚性、耦合性

• 耦合

- 耦合是模块之间相对独立性（互相连接的紧密程度）的度量，耦合程度越高，模块独立性越弱
- 耦合的七种类型由低到高排序
  - 无直接耦合：两个模块之间没有直接关系（没有调用、不传递任何信息）
  - 数据耦合：两个模块间有调用关系，传递简单的数据值（值传递）
  - 标记耦合：有调用关系，传递的是数据结构
  - 控制耦合：有调用关心，传递的是控制变量，控制变量可以让被调用方有选择的执行某一功能
  - 外部耦合：模块间通过软件之外的环境联结
  - 公共耦合：通过公共数据环境相互作用的那些模块间的耦合
  - 内容耦合：一个模块直接使用另一个模块的内部数据，或通过非正常入口传入另一模块内部时

• 内聚

- 内聚是对一个模块内部各个元素之间彼此结合紧密程度的度量，内聚程度越高，模块独立性越强
- 内聚的其中类型由低到高排序
  - ◆ 偶然内聚（巧合内聚）：模块内各元素间没有任何联系
  - ◆ 逻辑内聚：指模块内执行若干个结逻辑相似的功能，通过参数来确定要执行哪一个
  - ◆ 时间内聚：特定时间需要同时执行的动作组合在一起形成的模块、
  - ◆ 过程内聚：一个模块完成多个任务，这些任务必须按照指定过程执行
  - ◆ 通信内聚：模块内的处理元素都在同一个数据结构上操作
  - ◆ 顺序内聚：单一功能，模块内各个处理元素密切相关，且需要顺序执行
  - ◆ 功能内聚：模块内所有元素共同完成同一功能，缺一不可

• 系统结构设计原则

- 分解 －协调原则：大问题分解成小问题
- 自顶向下原则：抓住系统的主功能，由上到下分层分解
- 信息隐蔽 －抽象原则：上层规定下层做什么和模块间协调，但不规定怎么做
- 一致性原则：统一规范、统一标准、统一的文件模式
- 明确性原则：每个模块功能明确、接口明确，消除多重功能、无用接口，避免病态链接、消除接口复杂度
- 高内聚低耦合
- 扇入扇出适中：模块调用其他模块称为扇出，被其他模块调用称为扇如
- 模块规模适中：过大，则是分解的不充分，过小，则可能降低模块独立性

- 模块的作用范围应该再其控制范围之内

• 系统文档

- 系统文档是系统建设过程的”痕迹“，是系统维护人员的指南，是开发人员与用户的交流工具
- 系统文档在系统开发人员、项目管理人员、系统维护人员、系统评价人员、用户之间的作用
  - 用户－系统分析人员：可行性研究报告、总体规划报告、系统开发合同、系统方案说明书
  - 系统开发人员－项目管理人员：系统开发计划、系统开发月报、系统开发总结报告、项目管理文件
  - 系统测试人员－系统开发人员：系统方案说明书、系统开发合同、系统设计说明书、测试计划文档
  - 系统开发人员－用户：用户手册、操作指南
  - 系统开发人员－系统维护人员：系统设计说明书、系统开发总结报告、技术手册
  - 用户－维修人员：系统运行报告、维护修改建议

• 数据流图

- 基本图形元素
  - 外部实体：矩形，一般用 Ei 表示
  - 数据存储：两条横线或者缺边矩形，一般用 Di 表示
  - 数据流：有向边，起点 ———— > 终点
  - 加工：圆角矩形或圆，一般用 Pi 表示
- 顶层数据流图描述了系统的输入输出， 0 层数据流图是对顶层数据流图的细分
- 外部实体：当前系统之外的人、物、外部系统
  - 人：学生、老师、员工、主管

- ◆ 物 ： 传感器、控制器、车辆、采购部门
- ◆ 外部系统：支付系统、车辆交易系统、库存管理系统

- 数据存储：存储数据、提供数据
  - ◆ 存储加工后的输出数据，提供加工的输入数据
  - ◆ 例如： xxx 表、 xxx 文件
- 加工 ： 将输入数据处理后得到输出数据
  - ◆ 一个加工至少有一个输入数据流和一个输出数据里
  - ◆ 只有输入没有输出称为”黑洞“
  - ◆ 只有输出没有输入称为”白洞“
  - ◆ 加工的数据不足以产生输出”灰洞“
- 数据流：数据流的起点或者重点必有一个是加工

• 父图子图平衡

- 父图中的数据流，子图中也要有，其实就是根据父图去子图里一个个找看看有没有是父图里有的数据流但是子图没有
- 找出丢失数据流的技巧
  - ◆ 父图子图平衡
  - ◆ 加工既要有输入数据里、也要有输出数据流
  - ◆ 数据守恒（到题目的内容中去找图中有哪些丢失的部分）
- 数据建模 - E-R 图；行为建模 - UML；功能建模 - 数据流图

• 数据字典

- 数据字典为数据流图中的每个数据流、文件、加工，以及组成数据流的数据项做出说明
- 数据字典有四类条目：数据流、数据存储、基本加工、数据项
- 数据项是组成数据流和数据存储的最小元素，外部实体不再字典中说明
- 常用的加工逻辑描述方法：结构化语言、判定表、判定树

• 结构化开发方法

- 总的指导思想是自顶而下，逐层分解（从抽象到具体）
- 基本原则是功能的分解与抽象
- 软件工程中最早提出的方法，特别适合数据处理领域的问题
- 不适合解决大规模的、特别复杂的项目，难以适应需求的变化

• 结构化设计

- 体系结构设计：定义软件的主要结构元素及关系
- 数据设计：确定文件系统结构和数据库表结构
- 接口设计：描述软件使用放的外部接口，及各种构件间的内部接口
- 过程设计：定义各组成部分内的算法及内部数据结构
- 界面设计黄金原则
- 用户操纵控制
- 减少用户记忆负担
- 保持界面一致
- 构造分层数据流图时需要注意的问题
- 适当命名
- 图中要表示出数据流
- 一个加工不适合过多数据流
- 分解尽可能均匀

第 12章 软件工程

• CMM(能力成熟度模型)

CMM 将软件过程改进分为以下五个级别

- 初始级：杂乱无章，没有明确定义的步骤
- 可重复级：建立了基本的项目管理过程和实践，有必要的过程准则来重复以前在同类项目中的成功
- 已定义级：软件过程文档化、标准化
- 已管理级：制定了软件过程和产品质量的详细度量标准
- 优化级：加强了质量分析，通过过程质量反馈、新观念、新技术等不断改进

• CMMI(能力成熟度集成模型)

- 阶段式模型，结构类似 CMM，关注组织的成熟度
  - 初始的，过程不可预测缺乏控制
  - 已管理的，过程为项目服务
  - 已定义的，过程为组织服务
  - 定量管理的，过程已度量和控制
  - 优化的，集中于过程改进
- 连续式模型，关注每个过程域的能力
  - CL0(未完成的)：表明过程域的一个或多个目标没有被满足
  - CL1(已执行的)：过程域特定目标完成，转化可识别的输入目标产品，产生可识别的输出目标产品
  - CL2(已管理的)：已管理的过程制度化，关注针对单个过程实例的能力
  - CL3(已定义级)：已定义的过程制度化，关注过程的组织标准化及部署

- CL4（定量管理级）：可定量管理的过程制度化
- CL5（优化的 ）：优化的过程制度化，持续改进优化

• 瀑布模型

- 瀑布模型是将软件生命周期中各个活动，规定为依线性顺序链接的若干阶段模型
- 需求分析 》 设计 》 编码 》 测试 》 运行与维护，由前至后，相互衔接的固定次序
- 瀑布模型假设，一个待开发的系统需求是完整的，简明的，一致的，而且可以先于设计和实现之前完成
- 以项目的阶段评审和文档控制来对开发过程进行指导
- 优点：
  - 容易理解，管理成本低
  - 强调开发早期计划，需求调研，和产品测试
  - 适合开发需求明确的，大致固定不会随意变更的系统
- 缺点：
  - 客户必须要完整的，正确的，清晰的表达出需求
  - 开始的两到三个阶段很难评估进度
  - 接近项目结束时，出现大量的集成和测试工作
  - 项目结束之前都不能演示系统能力
  - 需求或者设计中的错误，到了项目后期才能发现
  - 项目风险控制能力较弱
- V 模型是瀑布模型的一个变体，重点在于质量保证活动和沟通，将基本问题逐步细化，实际上执行了一系列测试

• 增量模型

- 融合了瀑布模型的基本成分，和原型实现的迭代特征，它假设可以将需求分段为一系列的增量，每一个增量可以分别开发

- 第一个增量往往是核心产品，客户对每个增量的使用和评估都作为下一个增量的新特征和功能
- 每个增量均发布一个可操作版本
- 优点：
  - 具有瀑布模型的所有优点
  - 第一个可交付版本所需的成本时间很少
  - 开发由增量表示的小系统所承担的风险不大
- 缺点：
  - 如果没有对用户的变更要求有规划，那么产生的初始增量可能造成后续增量的不稳定
  - 管理成本、进度、复杂性

• 演化模型

- 演化模型是迭代的过程模型，使得软件开发人员能够逐步的开发出更完整的软件版本
- 演化模型特别适合对软件需求缺乏准确认识的情况，
- 典型的演化模型有，原型模型和螺旋模型
- 演化模型与增量模型区别：增量每次开发的时小的功能模块，演化每次开发整个产品

• 原型模型

- 原型模型比较适合用户需求不清、需求经常发生变化的情况，当系统规模不是很大，也不太复杂时采用比较合适
- 一个原型不必满足目标软件的所有约束，其目的是能快速、低成本的构建原型，迅速的开发出一个看得见摸得着的系统框架
- 步骤：交流沟通 》 快速计划 》 快速设计方式建模 》构建原型 》部署交付和反馈 - 完成后继续循环步骤
- 原型始于沟通其目的是为了定义软件的总体，有效的捕获用户需求

- 原型模式不适合大规模的系统开发

• 螺旋模型

- 螺旋模型将瀑布模型和演化模型相结合，加入了两个模型都忽略了的风险分析
- 每个螺旋周期和瀑布模型大致相符合
- 每个螺旋周期分为四个步骤
  - ◆ 制定计划：确定软件目标，制定实施方案，明确项目开发的限制条件
  - ◆ 风险分析：识别风险、消除风险
  - ◆ 实施工程：软件开发、验证阶段性产品
  - ◆ 用户评估：提出修正建议，制定下一周期开发计划
- 螺旋模型的特点是加入了风险分析，适合大规模、高风险、需求变化的系统
- 缺点：过多的迭代次数，会增加开发成本，延迟提交时间

• 喷泉模型

- 喷泉模型是一种以用户需求为动力，以对象作为驱动的模型，适合与面向对象的开发方法
- 喷泉模型克服了瀑布模型不支持软件重用，和多项开发活动集成的局限性
- 喷泉模型的开发过程具有迭代性和无间隙性
- 无间隙是指在开发活动（分析、设计、编码）之间不存在明显边界，允许活动交叉、迭代进行
- 优点：
  - ◆ 软件开发效率高，节省开发时间
- 缺点：
  - ◆ 开发阶段重叠，需要大量开发人员，管理成本高

- 需要严格的管理文档，审核难度大

• 统一过程模型（ UP ）

- 是一种 用例和风险驱动，以架构为中心，迭代并增量的开发过程
- 每次迭代有 5 个核心工作流
- 统一过程的四个技术阶段及里程碑
  - 起始阶段：专注于项目初创；里程碑：生命周期目标
  - 精化阶段：需求分析和架构演进；里程碑：生命周期架构
  - 构建阶段：关注系统的构建，产生实现模型；里程碑：初试运作功能
  - 移交阶段：关注软件提交方面的工作，产生软件增量；里程碑：产品发布

• 敏捷开发

- 总体目标是尽可能早、持续地对有价值的软件交付
- 敏捷开发使用户能够在开发周期的后期增加或改变需求
- 极限编程 (XP)
  - XP 是一种轻量级（敏捷）、高效、低风险、柔性、可预测、科学的软件开发方式
  - XP 的四大价值观：沟通、简单性、反馈、勇气
  - 5 个原则：快速反馈、简单性假设、逐步修改、提倡更改和优质工作
  - 12 个最佳实践：计划游戏、小型发布、隐喻、简单设计、测试先行、重构、结对编程、集体代码所有制、持续集成、每周 40h、现场客户和编码标准
- 水晶法 (Crystal)：每一个不同的项目都需要一套不同的策略、约定和方法论
- 并列征求法 (Scrum)：使用迭代的方法，每 30 天一个冲刺，按需求

级别来实现产品

- 自适应软件开发(ASD)： 6 个基本原则
  - ◆ 有一个使命作为指导
  - ◆ 特征被视为客户价值的关键点
  - ◆ ”重做“与”做“同样关键
  - ◆ 变化不被视为改正，而是 ”调整“
  - ◆ 交付时间迫使考虑每一个生产版本的关键需求
  - ◆ 风险包含
- 敏捷统一过程（AUP） ： 采用 ”在大型上连续，在小型上迭代“的原理来构建
  - ◆

    每个 AUP 执行以下活动：建模、实现、测试、部署、配置及项目管理、环境管理

• 需求分析

- 软件需求：指用户对目标软件在功能、行为、性能、设计约束等方面的期望
- 功能需求 ： 考虑系统做什么、何时做
- 性能需求
- 用户或人因素：用户理解使用系统难度、错误操作的可能性
- 环境需求：硬件或软件环境，机型，操作系统，平台
- 界面需求：考虑来自其他系统输入输出，数据格式存储介质规定
- 文档需求：文档针对那些读者
- 数据需求：接收发送数据格式
- 资源使用需求：运行所需计算机资源，维护所需人力
- 安全保密需求：数据隔离，系统备份
- 可靠性需求：隔离错误，出错重启等待
- 软件成本消耗和开发进度
- 其他非功能性需求

• 概要设计

- 设计软件系统的总体结构：将复杂系统按功能分成模块，并确定模块的功能、调用关系、接口、结构和质量
- 数据结构及数据库设计：数据库设计（概念设计 E-R、逻辑设计、物理设计）
- 编写概要设计文档：概要设计说明书、数据库设计说明书、用户手册、修订测试计划
- 评审

• 详细设计

- 对每个模块进行详细的算法设计
- 对模块内的数据结构进行设计
- 对数据库进行物理设计
- 其他设计
- 编写详细设计说明书
- 评审

• 系统测试

- 系统测试的意义：为了发现错误而执行程序的过程，成功的测试就是发现了至今为发现的错误
- 测试的目的：以最小的人力和时间发现潜在的各种错误和缺陷
- 测试的基本原则：
  - ◆ 尽早和不断的测试，贯穿整个开发阶段
  - ◆ 避免由原开发人员进行测试
  - ◆ 测试时要有预期的输出结果，与测试结果作比较
  - ◆ 关心不合理的输入或操作造成的结果
  - ◆ 不仅要检查程序是否做了该做的事，还要检查程序是否做了不该

做的事

- 严格按照计划测试，避免随意性
- 妥善保存测试用例及相关文档
- 后续测试以前次测试的基础上修改
- 系统测试的测试目标来源于需求分析阶段

• 单元测试

- 单元测试又称为模块测试，在模块编写完成且没有编译错误后开始执行
- 单元测试侧重于模块内部的处理逻辑与数据结构
- 单元测试检查模块的 5 个特征
  - 模块接口：保证测试模块的数据流可以正常输入输出；测试：形参与实参是否匹配，全局变量使用， I/O格式 ， 文件处理等
  - 局部数据结构：测试：变量定义及使用
  - 重要的执行路径
  - 出错处理
  - 边界条件
- 单元测试过程
  - 由于模块不是独立运行的，各模块之间存在调用与被调用关系，因此测试时需要开发两种模块
    - 驱动模块：相当于一个主程序，接收测试用例的数据，将数据输入到被测试模块，输出测试结果
    - 桩模块（存根模块）：用来代替 被测试模块所调用的子模块，检测测试模块的输入
- 高内聚可以简化单元测试

• 集成测试

- 集成测试就是将模块按照系统说明书组合起来测试，旨在发现与接口

相关错误

- 集成后可能出现的问题：
  - 穿过模块的数据丢失
  - 一个模块的功能对其他模块造成有害影响
  - 模块集成后没有达到预期功能
  - 全局数据结构出现问题
  - 模块组合后误差累积
- 自顶向下集成测试：属于增量方法，模块集成顺序从主控模块开始，沿着控制层次逐步向下，不用编写驱动模块，需要编写桩模块
- 自底向上集成测试：模块集成顺序从底层原子模块开始，沿着控制层次逐步向上，不用编写桩模块，需要编写驱动模块

• 回归测试

- 软件发生变更，可能会使原来正常的功能产生问题，这时需要回归测试重新执行一下已经测试过的某些子集，确保没有传播不期望的副作用
- 回归测试有助于保证不引入无意识的行为或者额外的错误

• 冒烟测试

- 将已经转换为代码的软件构件集成到构件中
- 设计一系列测试以暴露影响构建正确的完成其功能的错误

• 测试方法

- 静态测试：被测程序不在机器上运行，采用人工检测和计算机辅助静态分析来对程序进行检测
- 动态测试：通过运行程序发现错误，可以采用黑盒测试和白盒测试
- 测试用例：由测试的输入数据和预期输出结果组成，设计用例时应包

含合理的输入条件和不合理的输入条件

• 黑盒测试

- 不考虑软件内部结构和特性，把软件当做和黑盒子，测试软件的外部特性
- 常用的黑盒测试技术
  - ◆ 等价类划分：将程序输入输出划分为若干等价类，然后每个等价类中取一个代表性数据作为测试用例，测试时需要同时测试有效等价类和无效等价类
  - ◆ 边界值分析：输入的边界比中间更容易发生错误，应测试边界值和刚刚超过边界值的值
  - ◆ 错误推测：基于经验推测
  - ◆ 因果图：通过因果图转换判定表

• McCabe 度量法

- 通过定义环路复杂度，建立程序复杂度的度量，它基于一个程序模块的程序图中环路的个数
- 计算公式为 V(G) = m - n + 2 ; G 表示程序图，V(G) 表示程序图的环路复杂度， m 表示有向弧的个数， n 表示节点的个数
- 也可以通过查找图中有多少个闭合区域 k ， 然后 V(G) = k + 1

• 白盒测试

- 白盒测试也称结构测试，根据程序的内部结构来设计测试用例
- 白盒测试的常用技术是：逻辑覆盖、循环覆盖、基本路径测试
- 白盒测试原则：
  - ◆ 程序模块中的所有独立路径至少执行一次
  - ◆ 逻辑判断中 取”真“ 和 ”假“的情况都至少执行一次

- ◆ 每个循环都要在边界条件和一般条件下各执行一次
- ◆ 测试程序内部数据结构的有效性

- 逻辑覆盖
  - ◆ 语句覆盖：选择足够的测试数据，使得被测试程序中每条语句至少执行一次（只要保证每条语句执行，因此可能错过一些判断逻辑分支），语句覆盖对程序执行逻辑的覆盖程度很低，是很弱的逻辑覆盖
  - ◆ 判定覆盖：设计足够的测试用例，是的测试程序的每个判定表达式至少获得一次 ”真“值和”假“值，每个取”真“和”假“的分支都至少跑一次，因此也称为分支覆盖
  - ◆ 条件覆盖：创造一组测试用例，是的每一个判断语句中的每个逻辑条件的各种可能的值都至少满足一次
  - ◆ 判定 /条件覆盖：需要同时满足判定覆盖和条件覆盖的要求
  - ◆ 条件覆盖组合：在 判定 /条件覆盖 的要求前提下，判定表达式的所有真假不同组合都要测试一遍，例如 A>0 && b>0 就要测试四种 T&&T, F&&F, T&&F, F&&T
  - ◆ 路径覆盖：指测试用例要覆盖程序中所有可能路径
    - ■ 路径覆盖： 每一条路径可能都要有
    - ■ 如何覆盖所有路径：只要能把路径都走一遍就行
- 如果有伪代码则先将其转换成流程图，然后计算

• 运行和维护

- 软件维护是软件生命周期最后一个阶段，不属于系统开发过程
- 系统可维护性评价指标：可理解性、可测试性、可修改性
- 文档是软件可维护性的决定因素
- 可维护性在开发阶段就需要考虑到，此后的每个阶段也都需要考虑

• 软件文档

- 编写高质量的文档可以提高软件开发质量
- 文档也是软件产品的一部分，没有文档的软件不能称之为软件
- 软件文档的编制在软件开发中占有突出的地位，和相当大的工作量
- 高质量的文档对软件产品的效益有着重要意义
- 总的来说软件文档只好不坏

• 软件维护内容

- 正确性维护：改正在系统开发和测试阶段未发现的问题
- 适应性维护：适应行业环境变化和管理需求而做出的修改
- 完善性维护：为扩充功能和改善性能而做出的修改
- 预防性维护：为了适应未来的软硬件环境变化而主动做出的预防

• 软件可靠性、可用性、可维护性公式

- 可靠性：给定时间间隔、给定条件下无失效运作的概率 MTTF/(1+MTTF) MTTF: 平均无故障时间
- 可用性：给定时间，系统正确运作的概率 MTBF/(1+MTBF) MTBF: 平均失效间隔时间
- 可维护性：给定时间，使用规定的过程和资源完成维护活动的概率 1/(1+MTTR) MTTR: 平均修复时间

• 沟通路径计算

- n 个程序员，无主程序员： (n-1)*n/2
- n 个程序员，一个主程序员： n-1

• 软件项目估算

- COCOMO 估算模型：精确的易于使用的成本估算模型，按精细程度分为基本 COCMO、中级 COCOMO 、详细 COCOMO
  - ◆ 基本 COCOMO 模型 ： 静态单变量模型
  - ◆ 中级 COCOMO 模型：静态多变量模型，将系统模型分为系统和部件两个层次
  - ◆ 详细 COCOMO 模型：将软件系统模型分为系统、子系统和模块三个部分
- COCOMOII 估算模型：层级结构的估算模型，分为三个阶段
  - ◆ 应用组装模型：软件开发的前期阶段使用，使用对象点估算
  - ◆ 早期设计阶段模型：在需求已经稳定，并且基本的软件体系结构已经建立的时期使用，使用功能点估算
  - ◆ 体系结构阶段模型：软件构造过程中使用，使用代码行估算

• Gannt 图（甘特图）

- 一种简单的水平条形图，以日历为基准描述项目任务，水平轴展示日历时间线，每个横条表示一个任务，水平条的起点终点对应日历时间，表示任务的开始和结束，其长度代表任务持续时间，同一时间段的多个水平条间是并发关系
- 优点：清晰描述任务的开始结束时间，任务进展情况，和任务间的并行关系
- 缺点：不能清晰的反应出任务间的依赖关系，不能反映项目的关键，不能反应计划中有潜力的部分

• PERT 图

- PERT 图是一个有向图
- 图中的箭头表示”任务“，箭头上的时间表示完成”任务“所需时间

- 图中的节点表示”事件“，表示指向当前节点的”任务“结束。并且由当前节点指出的”任务“开始
- 当流入该节点的所有任务都结束时，由当前节点指出的任务才会开始
- 事件本身不消耗时间和资源，仅表示一个时间点
- ”事件“节点由三部分组成：事件号、出现事件的最早时刻、最迟时刻
- 开始节点：没有流入的任务的节点，可以有多个，开始节点的最早时刻为 0
- 结束节点：没有流出的任务的节点，只能有一个，结束节点的最迟时刻等于其自身的最早时刻
- 最早时刻计算要从开始节点向结束节点推算，最晚时刻要从结束节点向开始节点推算
- 最早时刻：能开始该事件节点出发的后续任务的最早时刻，有多个流入的任务时，则选择计算后的最大的那个
- 节点的最早时刻计算：流入的任务的所需时间 + 流入前一个节点的最早时刻
- 最迟时刻：从此节点出发的任务，必须在此时刻前开始，否则工程不能如期完成，有多个流出的任务时，则选择计算后最小的那个
- 节点的最迟时刻计算：流出任务指向的节点的最晚时刻 - 流出任务所需时间
- 松弛时间：在不影响工期的前提下，完成该任务有多少机动余地，是挂在任务下的
- 松弛时间计算：链接任务的两个节点中，箭头指向的节点的最迟时刻 - 任务的耗时 - 箭头尾部的节点的最早时刻
- 关键路径：从开始节点到结束节点，松弛时间都是 0 的路径
- PERT 图优点：给出任务开始结束时间，还能给出任务建的依赖关系、关键路径
- PRET 图缺点：不能反映出任务键的并行关系

• 项目活动图

- 与 PERT 图类似，其中：顶点表示里程碑，链接定点的变表示活动，边上的数字表示活动所需时间
- 除了图形及命名与 PERT 图不一样，其他计算方面基本相似

• 软件配置管理

- 软件配置管理主要目标：变更标识、变更控制、版本控制、确保变更正确的实现、变更报告
- 软件配置管理内容：版本管理、配置支持、变更支持、过程支持、团队支持、变化报告、审计支持
- 软件配置管理内容（第二版）：软件配置标识、变更管理、版本控制、系统建立、配置审核、配置状态报告
- 配置数据库分为三类 ： 开发库、受控库、产品库

• 风险管理

- 软件风险的特性：不确定性（可能发生也可能不发生）和损失（如果发生就会产生恶性后果）
- 软件风险的分类
  - ◆ 项目风险：拖延项目进度、增加项目成本。例如：预算、进度、人员、资源、项目复杂度、规模及结构不确定
  - ◆ 技术风险：质量和交付时间。例如：设计、实现、接口、维护等方面的问题
  - ◆ 商业风险：威胁软件生存能力

• 风险识别

- 系统的指出对项目计划（估算、进度、资源分配等）的威胁，识别出已知风险和可预测风险后，项目管理者要回避这些风险，必要时控制

这些风险

- 识别风险的一种方法是建立 ”风险条目检查表“
- 风险条目检查表格式：列出每一种类型的有关特性，最终给出一组风险因素和驱动因子及发生概率，风险因素包含性能、成本、支持、进度

• 风险预测

- 又称风险估计，通过两个方面进行评估， 1. 风险发生概率 2. 风险发生后果
- 风险预测活动：
  - ◆ 建立一个尺度或者标准，反应风险发生的可能性
  - ◆ 描述风险产生的后果
  - ◆ 估算风险对项目和产品的影响
  - ◆ 标注风险预测的精确度，以免产生误解
- 风险预测技术：建立风险表
- 影响风险产生后果的三个因素：风险的本质、范围、时间
- 整体风险显露度 = 风险发生的概率 * 风险发生给项目带来的成本
- 风险评估：一种对风险评估很有用的技术是定义风险参照水准

• 风险控制

- 风险控制的目的：辅助项目组建立处理风险的策略
- 风险避免：应对风险的最好办法，就是主动的避免风险
- 风险监测：项目管理者应监控某些因素，这些因素可以提供风险是否正在变低或者变高的指示
- RMMM 计划：将所有风险分析工作文档化，并由项目管理者作为整个项目计划的一部分来使用
- 风险缓解是一种问题规避活动，风险监测是一种项目跟踪活动

- 风险监测的另一个任务就是找到“起源”

• 软件质量

- ISO/IEC 9126 软件质量模型：由三层组成：质量特性 》 质量子特性 》 质度量指标
- 功能性：满足规定或隐含需求的那些功能
  - 适应性：是否适合有关的软件属性
  - 准确性：能够得到正确的结果
  - 互用性：与其他系统进行交互的能力
  - 依从性：服从有关标准、法规
  - 安全性：避免非授权访问，和意外访问
- 可靠性：一定时间内软件维持性能水平的能力
  - 成熟性：软件故障引起失效的频率
  - 容错性：在软件错误或违反指定接口的情况下维持指定的性能水平
  - 易恢复性：故障后，回复的能力
- 易使用性：是否容易使用，使用付出学习成本
  - 易理解性：
  - 易学性
  - 易操作性
- 效率：软件性能水平和所占资源
  - 时间特性：响应处理时间
  - 资源特性：所使用的资源量
- 可维护性
  - 易分析性：诊断所需要的成本
  - 易改变性：缺陷改正成本
  - 稳定性：不发生风险的能力

4. 易测试性

- 可移植性

- 适应性：软件转移到不同环境耗时和成本
- 易安装性：指定环境下安装软件成本
- 一致性：不同环境安装后软件能力一致
- 易替换性

• 软件评审 （低概率考）

- 设计质量：设计的规格说明书符合用户要求
- 程序质量：程序按照规格说明书所规定的情况正确执行
- 设计质量评审内容：
  - 评价软件的规格说明是否符合用户要求
  - 评审可靠性，即是否能避免输入异常
  - 评审保密措施实施情况
  - 评审性能实现情况
  - 评审软件可修改性、可扩充性、可互换性、和可移植性
  - 评审软件可测试性
  - 评审软件复用性
- 程序质量评审内容： 从开发者角度进行评审，与开发技术直接相关，着眼于软件本身结构
  - 功能结构
  - 功能的通用性
  - 模块的层次
  - 模块结构：控制流结构、数据流结构、模块结构与功能结构之间的对应关系
  - 处理过程的结构
  - 与运行环境的接口：与硬件的接口、与用户的接口
- 完成质量评估的中心活动是技术评审，目的是揭露质量问题

• 软件容错技术

- 对无可避免的差错，使其影响减到最小
- 容错软件的定义
  - 对自身错误具有屏蔽能力的软件
  - 一定程度上能从错误状态恢复到正常状态的软件
  - 发生错误仍能完成预期任务的软件
  - 一定程度上具有容错能力
- 容错的一般方法：实现容错的主要手段是冗余，冗余是指对于实现系统功能的多余的那部分资源

• 四类冗余技术

- 结构冗余：静态冗余、动态冗余、混合冗余
- 信息冗余：为检测或纠正信息在运算或传输中的错误而外加的一部分信息
- 时间冗余：重复执行程序或指令来先出瞬时错误的影响
- 冗余附加技术：
  - 屏蔽硬件错误的附加技术： 1. 关键程序和数据的冗余存储。 2. 检测、表决、切换、重构、纠错、复算
  - 屏蔽软件错误的附加技术： 1. 冗余备份程序的存储及调用。 2. 实现错误检查和错误恢复。 3. 实现容错软件所需的固化程序

• 软件工具

- 软件开发工具
  - 需求分析工具
  - 设计工具
  - 编码与排错工具
  - 测试工具

- 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

• 其他

- 高质量文档特性：针对性、精确性、清晰性、完整性、灵活性、可追溯性
- 软件工程基本要素：方法、工具和过程
- 数据处理领域不太复杂的软件适合结构化开发方法
- 软件调试方法：
  - 试探法：猜测问题所在位置，通过输出语句获得错误线索
  - 回溯法：从发现问题的位置开始，沿着程序的控制流程往回跟踪代码
  - 对分查找法：通过缩小错误范围，找出问题
  - 归纳法：收集正确的与不正确的数据，分析他们之间的关系，提出假想的错误原因
  - 演绎法：列出所有可能的错误原因，排除、尝试

第 13章 数据结构与算法

• 复杂度

- 大 O表示法：以算法中基本操作重复执行的次数（频度）作为算法时间点的度量，一般只要大致的计算出数量级即可
- O(1) < O(log2 n) < O(n) < O(nlog2 n) < O(n^2) < O(n^3) < O(n!) < O(n^n)
- 常数阶 < 对数阶 < 线性阶 < 线性对数阶 < 平方阶 < 立方阶 < 阶乘阶 < n 次方阶
- 复杂度计算规则：多项相加保留最高项；多项相乘都保留；加乘混合，则按照计算规则；系数化为 1
- 时间复杂度，与循环相关
- 空间复杂度，看有没有开辟新空间，比如数组

• 渐进符号

- O(g(n))： 表示渐进上界， 10n^2+4n+2 = O(n^2) 是成立的，因为渐进上界括号内的复杂度要大于等于等式左边的计算结果
- Ω(g(n))： 表示渐进下界， 10n^2+4n+2 = O(n^3) 是不成立的，因为渐进下界括号内的复杂度要小于等于等式左边的计算结果
- Θ(g(n))： 表示渐进紧致界， 10n^2+4n+2 = O(n^3) 是不成立的，因为渐进紧致界括号内的复杂度要等于等式左边的计算结果

• 递归的时间空间复杂度

- 递归的时间复杂度 = 递归的次数 * 每次递归的时间复杂度
- 递归的空间复杂度：如果递归中有变量声明赋值，则相当于 长度为递归次数的数组

- 递归式主方法：
  - 假如题目给了一个递归式的表达式长得像 T(n) = aT(n/b) + f(n) 那么可以按照下边方法尝试一下
  - 比如给了个题目 T(n) = 2T(n/2) + nlgn 让求复杂度
  - 那么按照公式换算得出 a=2; b=2; f(n) = nlgn;
  - 如果 f(n) 中有 lg 相关那么就套这个公式 f(n) = Θ(nˆ(logb a)lgk n)；将换算的数据代入，即 nlgn = (nˆ(log2 2)lgk n)；得到 k = 1；然后再代入到 这个公式 T(n) = Θ(nˆ(logb a)lgk+1 n) 得出 T(n) 复杂度为 nlg2n
  - 若果 f(n) 中没有 lg 则直接代入到 T(n) = Θ(nˆ(logb a))

• 线性表

- 线性关系：具有单一前趋和后继的数据关系，元素一个接一个的排列
- 线性表：最简单基本常见的线性数据结构，通常表示为 (a1,a2,…an)
- 线性表特点：‘第一个元素’、‘最后一个元素’都是唯一且只有一个的；除第一个元素只有后继，最后一个元素只有前趋外，其余元素都含有前趋和后继
- 线性表顺序存储：指用一组连续的存储单元，一次存储线性表中的数据，也就是说物理位置相邻
- 优点：可随机存取表中元素，查询效率高
- 缺点：插入和删除需要移动元素，删除插入效率低，表长为 n，插入新的值平均移动 n/2；删除值平均移动 (n-1)/2
- 顺序表插入元素时间复杂度：顺序表最后插入 O(1)；顺序表首位插入 O(n);平均复杂度 O(n)
- 顺序表删除元素时间复杂度：顺序表最后一位删除 O(1)；顺序表首位删除 O(n)；平均复杂度 O(n)
- 查找元素时间复杂度：直接根据数组下标查询，所以为 O(1)

• 线性表链式存储

- 通过指针链接起来节点，来存储数据元素，分为数据域 +指针域；数据元素的节点地址不是连续的，节点空间在需要的时候才申请
- 若节点只有一个指针域，则成为线性链表或单链表
- 头节点：不存储数据 (!!也可以存储链表长度)，只存储链表第一个节点的地址
- 头指针、尾指针：有了尾指针就可以直接从尾部遍历查找，有了尾指针时间复杂度会有变化
- 连式存储插入元素时间复杂度：首位插入 O(1)； 末位插入 O(n)；平均复杂度 O(n)
- 连式存储删除元素时间复杂度：首位删除 O(1)； 末位删除 O(n)；平均复杂度 O(n)
- 连式存储查找元素时间复杂度：首位查找 O(1)； 末位查找 O(n)；平均复杂度 O(n)
- 循环单链表：就是在单链表的基础上，尾部节点的指针指向头部节点，时间复杂度与单链表一致
- 双链表：每个节点指针不但指向后驱节点，还会指向前趋节点，也就是一个节点即知道前一个节点地址也知道后一个节点的地址

• 栈

- 栈的定义：只能通过访问它的一端来实现数据存储和检索的一种线性数据结构
- 栈的修改按照先进后出，后进先出的原则进行，插入和删除操作的一端称为栈顶 Top，另外一端称为栈底，不含元素的称为空栈
- 理解：可以将栈想象成一个杯子，先进先出，与 递归的执行过程类似
- 栈的链式存储：用链表作为存储结构的栈，也称为 链栈，不必设置头指针，链表的头指针就是栈顶指针

• 队列

- 队列的定义：一种先进先出的线性表，只允许在表的一端插入值，在另一端删除元素
- 顺序队列：使用顺序存储的队列，需要设置队头指针和队尾指针
- 循环队列：可处理顺序队列中插入值溢出越界，只需要改变队头和队尾指针即可，避免采用线性表插值导致的遍历

• 队列链式存储

- 双端队列：入队出队在两端都可以进行
- 两个栈可以模拟一个队列，但是两个队列无法模拟一个栈

• 串

- 串是一种特殊的线性表，其数据元素为字符，是由字符构成的有限序列，例如： ‘ abc’
- 空串：长度为零，不包含字符
- 子串：串中任意长度的连续子串构成的序列 ‘ abc’ 的 子串可以是 ‘ab’ 但不能是 ‘ ac’
- 串比较：两个串比较时以字符的 ASCII 码值作为依据
- 串的模式匹配：可以理解为 JS 中所要的效果 a.indexOf(b)
  - ◆ 复杂度：主串长度 n ， 子串长度 m
  - ◆ 最好情况（首位即匹配成功）：复杂度 O(m) | O(1)
  - ◆ 最坏请款（比较到最后 m 位）：复杂度 O(n*m) => (n-m+1)*m
  - ◆ 平均复杂度： O(n+m)

• 串的模式匹配 KMP 算法

- 串的前缀：包含第一个字符，但是不包含最后一个字符的子串
- 串的后缀：包含最后一个字符，但是不包含第一个字符的子串

- KMP：可以提高串的模式匹配效率，时间复杂度为：O(n+m)
- KMP next 的数值计算：第 i 个字符的 next 值 = 第 i 个字符前的字符串中，”串前缀 ===串后缀“最长的那个的长度 + 1；其中 next[1] = 0

• 一维数组

- LOC: 表示第一个元素的首地址； L: 表示每个元素的大小
- 计算数组中某个元素 i 的地址： ai = LOC + i*L

• 二维数组

- 二维数组的存储，会将将第二行在第一行后边连续存储（列优先存储，则第二列在第一列存储之后连续存储）
- LOC: 表示第一个元素的首地址； L: 表示每个元素的大小 ; N: 行数 ; M: 列数；
- 计算二维数组 [i][j] 的地址 = LOC + ([i][j] 前有多少个元素）* L
- 行优先存储： LOC + (i*M + j) * L
- 列优先存储： LOC + (j*N + i) * L
- N == M 且 i == j 时，按行还是按列的地址都是一样的，偏移量也是一样的

• 对称矩阵

- 矩阵内任意元素具有 Ai,j = Aj,i 的特点
  - 按照主对角线对称，分为上三角区和下三角区
  - 存储时只需要存储下三角区 + 主对角线即可，一般使用一个一维数组存储
  - 按行存储： i >=j 时 Ai,j = (i+1)i/2 + j + 1; i < j 时，因为

是主对角线堆成 则可以改为计算 Aj,i

• 三对角矩阵

- 只有主对角线两侧紧邻区域有值，其他区域都是 0
- 存储时只存中间区域的值，不存 0 的位置，存到一个一维数组中
- 按行存储： Ai,j = 2i+j+1

• 稀疏矩阵

- 矩阵很大，但是存的非 0 元素很少
- 压缩存储方式：使用三元组顺序表存储 [i, j, data] [行 , 列, 值]
- 另一种压缩方式：十字链表

• 树

- 一种非常重要的非线性结构，一个元素可以有 0 个或多个的后继元素
- 树是 n 个节点的有限结合， n=0 时称为空树，有且只有一个根节点
- 兄弟节点：具有相同父节点的节点
- 节点的度：一个节点的子树的个数计为该节点的度
- 叶子节点：终端节点，没有子节点的节点，度为 0 的节点
- 内部节点：分支节点，度不为 0 的节点
- 节点的层次： 根为第一层，跟的子为第二层，以此类推
- 树的高度：一棵树最大的层数计为树的高度或者树的深度
- 树的度：树中所有节点度的最大值

• 树的性质

- 树中节点总数 = 所有节点的度数之和 + 1

- 度为 m 的树中第 i 层上最多有 m^(i-1) 个节点，最多的情况就是每层都有 m 个节点
- 高度为 h 的 度为 m 的树，最多有 (m^h - 1)/(m-1) 个节点，最多的情况就是每层都有 m 个节点，一共 h 层
- 具有 n 个节点度为 m 的树的最小高度为 logm (n(m-1) + 1)，要想高度最小，那还是每层都要有 m 个节点

• 二叉树

- n 个节点的有限集合， n=0 时为空树，或者是由一个根节点及两个不想交的且分别称为左右子树的二叉树组成
- 树和二叉树的区别
  - ◆ 二叉树中子树分为左子树和右子树，即使只有一个子树也要区分左右
  - ◆ 二叉树中节点最大度数为 2

• 二叉树的性质

- 二叉树第 i 层最多有 2^(i-1) 个节点，实际就是树的公式中国将度 ==2 后代入
- 高度为 h 的二叉树最多有 2^h - 1 个节点；最多的情况下，每一层的节点数都是前边所有层的节点数 +1
- 任意一个二叉树，度数为 0 的叶子节点个数 = 度数为 2 的节点个数 + 1；由 ”树中节点总数 = 所有节点的度数之和 + 1“ 推断出来的
- 有 n 个节点的完全二叉树的高度为 (log2 n + 1) 的向下取整或者 (log2 (n+1)) 的向上取整

• 满二叉树

- 高度为 k 的二叉树，如果有 2^k -1 个节点，那就是满二叉树，可以从上往下，从左往右编号

• 完全二叉树

- 除最后一层外，其他层都是”满的“，最后一层的节点也是从左至右依次放置；这个情况下，完全二叉树中的每一个节点都能与同样深度的满二叉树一一对应

• 卡特兰数

- n 个节点的二叉树有多少种 : (C2n n)/(n+1); 其中 (Cn m) = n!/m!*(n-m)!

• 二叉树的顺序存储

- 用一组连续的存储单元，存储二叉树中的节点
- 树节点与编号 i 间关系
  - 求父节点： i=1, 则为根节点，根节点没有父节点；若 i>1 则该节点的父节点为 i/2 的向下取整
  - 求左子节点： 2i<=n 则该节点左子节点编号为 2i, 否则无左子节点
  - 求右子节点： 2i+1<=n 则该节点右子节点编号为 2i+1, 否则无右子节点
- 顺序存储对完全二叉树比较合适，对普通二叉树来说则为了保持关系，会有很多”虚节点“
- 单支树，除了叶子节点，其他节点的度都为 1

• 二叉树链式存储

- 二叉链表存储，每个二叉链表节点存储 [当前节点的数据元素 ，左子节点指针，右子节点指针]，如果没有对应的子节点则存 NULL
- 二叉链表存储中的有效指针域数量：即为树结构中的有效关联数量，每个子节点都仅有一个父节点（除了根节点 ），因此有效的数量 = 总的节点数 - 1 个
- 三叉列表：在 二叉链表的基础上增加了一个指向父节点的指针域

• 二叉树的遍历

- 先序遍历：按照 根左右的次序遍历
- 中序遍历：按照 左根右的次序遍历
- 后序遍历：按照 左右根的次序遍历
- 层次遍历：从根节点开始，每一层从左往右访问

• 还原二叉树

- 单独一种遍历结果没法还原出树
- 先序遍历和层次遍历的的首位就是根节点，后序遍历的末位也是根节点，因此 中序遍历与其他任意一种遍历结合就能还原二叉树

• 平衡二叉树

- 二叉树中任意节点的左右子树高度只差不大于 1，完全二叉树一定是平衡二叉树

• 二叉排序树

- 又称二叉检查树
- 根节点关键字：就是根节点的值

- 根节点的关键字大于左子树的所有节点关键字
- 根节点的关键字小于右子树的所有节点关键字
- 左右子树也是二叉排序树，依次递归
- 二叉排序树的中序遍历是一个有序序列
- 计算题：会给一个关键字序列，关键字序列的收个元素就是根节点，后边数字比根大就放右边，比根小就放左边，节点为空就直接插入，不为空就与其比较后向下层插入，其他的元素依次判断插入二叉排序树中即可
- 二叉排序树查找的效率，是受查找的层数相关的，层数越高效率越差

• 最优二叉树

- 又称哈弗曼树 它是一类带权路径长度最短的树
- 路径：从树的一个节点，到另一个节点之间的通路
- 路径长度：路径上的分支数目（几条线）
- 树的路径长度：从根节点到每一个叶子节点路径长度之和，乘以权值则代表树的带权路径长度

• 最优二叉树构造

- 题：将一个权值集合（例如： {1,3,3,4} ），构造成一个二叉树
- 构造方法：
  - 从前往后找两个权值最小的
  - 这两个里边小的作为左子树，大的作为右子树，构造新的二叉树，这个二叉树的根的权值等于两者相加
  - 将计算后的根加入权值集合末尾
  - 继续上述步骤，直到集合中只剩下一个
- 权值大的距离根节点近，权值小的距离根节点远
- 最优二叉树只有度为 0 的节点和度为 2 的节点
- 总节点个数 =（权值数量 * 2）- 1

• 哈夫曼编码

- 等长编码：对每个字符编制相同长度的二进制码，例如英文 26 个字符，需要 2^5 即需要 5 位二进制串表示
- 哈夫曼编码不是等长编码
- 接收方按照 5 位一组将电文进行分割后，通过对应实现译码
- 题：一般会给一串字符并说明字符的权重
- 我们根据权重画出哈夫曼树，并将节点替换为相应的字符
- 根节点与左子节点的连线为 0 ，与右子节点的连线为 1，将每条节点间连线标出
- 某个字符的编码就是从根节点开始到当前字符节点所经过路径上的 0,1 组成
- 哈夫曼编码压缩比：即每个字符从等长编码到哈夫曼编码的压缩情况
- 哈夫曼编码是基于贪心策略的

• 线索二叉树

- 普通二叉树，采用二叉链表做存储结构，则链表中会有空指针域，利用这个空指针域来存放节点的前驱后继信息

• 图

- 图中，任意两个节点之间都可能有关系，一个节点可能有多个前趋或者多个后继
- 图，标记为 G(V,E) V 表示顶点的非空有限集合； E 是图中边的有限集合
- 有向图：每个边是有方向的，那么顶点关系用 <v1,v2> v1 是起点，v2 是终点
- 无向图：每个边是无方向的，那么顶点关系用 (v1,v2)
- 完全图：每个顶点都与其他顶点有一个边，则称为完全图

- 假设无向完全图有 n 个顶点，那完全图的边一共有 n(n-1)/2
- 有向完全图的边总数则为 n(n-1)，因为每两个顶点间有来回两条边

- 无向图顶点的度：关联于该顶点的边的数量
- 有向图的出度与入度：出度 - 从该顶点指出去的边的数量；入度 - 指向该顶点的边的数量；总度数 = 出度 + 入度
- 图的总度数 = 边数 * 2
- 路径：就是通过那几条边的组合，实现从一个顶线到另一个顶点；路径长度就是路径上边或弧的数目
  - 第一个顶点和最后一个顶点相同的路径称为回路或者环
  - 简单路径：路径上，除了起点和终点可以相同外，其余顶点都不相同的路径

• 连通图

- 连通图：无向图中，任意两个顶点间都有至少一条的路径可以连通，则称为连通图
- 对于 n 个顶点的无向图，最少有 n-1 条边就可以实现连通，最多 n(n-1)/2
- 强连通图：有向图中，任意两个顶点间都有来回的两条路径连通，称为强连通图
- 对于 n 个顶点的有向图，最少有 n 条边就可以实现连通，最多 n(n-1)

• 图的存储结构

- 邻接矩阵表示法：使用一个矩阵来表示图中顶点间的关系，有 n 个定点的图，其邻接矩阵就是 n 阶的，矩阵中值为 1 表示有边，为 0 表示无边
- 无向图的邻接矩阵是对称的，有向图则不一定是

- 无向图邻接矩阵计算某个定点的度数：定点 vi 的度数就是第 i 行非 0 元素的个数
- 有向图邻接矩阵计算某个定点的度数：定点 vi 的出度 - 第 i 行非 0 元素的个数 ； 入度：第 i 列非 0 元素的个数
- 邻接链表表示法：为图的每个节点建一个单链表，具体的还得看图，这里不做详解
- 有向图的邻接链表，有几个指出来的表结点就有几条边；无向图的邻接链表，有 n 指出来的表结点就有 n/2 条边
- 稠密图和稀疏图，边多的就是稠密图，边少的就是稀疏图
- 邻接矩阵表示法适合稠密图，邻接链表适合稀疏图
- 网：边或弧带有权值的图，称为网；网的邻接矩阵中有边的会用权值表示，没有边的用 oo 无穷表示

• 图的遍历

- 深度优先搜索：从一个顶点 A 按照出度向另一个顶点 B， B 在按照出度向顶点 C, 这样先从路径的起始遍历到路径的末尾，然后在通过回溯，换一个路径遍历
- 深度优先搜素的时间复杂度： n 表示顶点数， e 表示边数，邻接矩阵存储的复杂度为 O(n^2)；邻接链表的时间复杂度 O(n+e)，用栈的方式
- 广度优先搜索：先遍历一个顶点 A 的所有出度的节点，在遍历出度节点的所有出度节点，以此类推，相当于一层层遍历
- 广度优先搜素的时间复杂度： n 表示顶点数， e 表示边数，邻接矩阵存储的复杂度为 O(n^2)；邻接链表的时间复杂度 O(n+e)，用队列的方式

• 拓扑排序

- AOV 网：一种有向无环图

- AOV 网中弧的尾部是前趋，弧指向的是后继，前趋对后继有制约关系
- 拓扑排序：是 AOV 网中所有定点排出的线性序列，并且网中任意路径 <vi,vj>的前后顶点在这个线性序列中 vi 排在 vj 前
- 假设 AOV 图是一个工程的计划，那 AOV 网的一个拓扑排序就是工程顺利完成的可行方案
- 拓扑排序计算方式：
  - 在 AOV 网中选择一个入度为 0 的顶点，且输出它
  - 在网中删除该顶点及与该顶点相关的所有弧
  - 重复上述两步直到不存在入度为 0 的顶点为止
  - 例如：得到 614325 这个拓扑序列，那么对于 6 与 4 来说，可能存在弧 6->4;不可能存在弧 4->6;可能存在 6->4 的路径，一定不存在 4->6 的路径

• 查找

- 查找表：由同一类型元素构成的集合，集合元素之间存在着完全松散的关系
- 静态查找表只进行以下两种操作
  - 查询某个特定元素是否在查找表中
  - 检索某个特定元素的各种属性
- 动态查找表，除了静态查找表的功能还进行如下操作
- 在查找表中插入一个数据
- 从查找表中删除一个数据
- 关键字是数据元素某个数据项的值，可以用来标记数据元素
- 静态查找表有：顺序查找、二分查找、分块查找
- 动态查找表有：二叉排序树、平衡二叉树、 B_ 树、哈希表
- 查找的基本操作：将记录的关键字与给定的值进行比较
- 顺序查找：就是从左向右查找，不需要有序，适用于顺序存储和链式存储方式，平均查找长度为 (n+1)/2

• 二分查找

- 又称折半查找，就是将给定的值与查找表中间的值进行比较，下标求中间值（比较值），中间值为小数则向下取整，比较后舍弃中间值进行下一轮比较
- 例如 10 个值，则依次取 (1+10)/2 => 5； (6+10) => 8; (9+10)/2 => 9; (10+10)/2 => 10;
- 需要顺序存储，且要有序存储
- 二分查找成功时，给定值的比较次数最多为 [log2 n) + 1 向下取整
- 二分查找的平均查找长度为： (log2 (n+1)) - 1

• 哈希表

- 哈希表：通过计算一个已记录的关键字为自变量的函数（哈希函数），来得到该记录的存储地址
- 根据设定的哈希函数 H(key) 和处理冲突的方法，将一组关键字映射到一个有限的连续地址集上
- 对于一个哈希函数，两个不同的关键字，经过哈希函数查找后的地址相同时，称为冲突，具有相同哈希函数值的关键词称为同义词
- 一般情况下，冲突可以尽可能的减少，而不能避免，要减少冲突，就要尽可能均匀的将关键字映射到存储区的各个存储单元
- 对于哈希表来说，主要考虑两个问题，一个是如何构造哈希函数，一个是如何解决冲突
- 哈希函数构造方法：
  - 构造哈希函数时，一般都会对关键字进行计算，尽可能的使得关键字的所有成分都起作用
  - 除留余数法：H(key) = key % m = 地址； 求关键字 key 的余数作为地址，其中 m 是一个接近 n 但不大于 n 的质数， n 是散列表的长度，地址一般从 0 开始

- 解决冲突的方法
  - 解决冲突就是在冲突时，给冲突的关键字再找个地址存储
  - 线性探测法：如果 H(key) 冲突了 那就按照 Hi = (H(key) + i) % m；再计算一个地址，其中 i=1,2,3,... 表示再次计算如果还是冲突就，加码再次计算，直到不冲突为止
  - 二次探测法：如果 H(key) 冲突了 那就按照 Hi = (H(key) + di) % m；再计算一个地址，其中 i=1^2,-1^2,2^2,-2^2,... 表示再次计算如果还是冲突就，就按照 1^2,-1^2,2^2,-2^2,... 的方式依次尝试，直到不冲突为止 ，与线性探测法相比，是在 H(key) 地址的左右来回试探
  - 装填因子： a = 表中装入的记录树 / 哈希表长度， a 代表哈希表的装满程度， a 越大冲突概率越大

• 堆

- n 个关键码构成的序列 {k1,k2,…kn}，满足以下关系称为堆
- 小顶堆： ki <= k2i && ki <= k(2i+1) 即 根节点比子节点小的二叉树，层次遍历的结果
- 大顶堆： ki >= k2i && ki >= k(2i+1) 即 根节点比子节点大的二叉树，层次遍历的结果
- 将一个序列调整为一个大顶堆或者小顶堆
  - 先按照层次遍历的结果将其还原成一个二叉树
  - 从叶子节点开始向上依次判断，比如说要还原成一个小顶堆，就要使得局部根节点的值比子节点要小，不符合的就互换一下

• 排序

- 通过排序使得关键字满足递增或者递减的关系
- 稳定的：原序列中有 Ri 和 Rj 的关键字相同，且 Ri 在 Rj 之前，经过排序算法后，还能保持 Ri 在 Rj 之前，那就是稳定的，

否则是不稳定的

- 归位：在排序时能确定最终排序位置，比如 Ri 排序后应该放在位置 3，那在计算时如果开始没将它放在 3 这个位置就是不归位

• 直接插入排序

- 新序列中以 R1 开始，遍历 原序列 R, 每个 Ri, 依次与新序列中从后开始的关键字相比较，大的直接插入到后边，小的继续往前判断，直到插入
- 直接插入排序，稳定、不归位，平均复杂度 O(n^2), 最大复杂度 O(n^2), 最小复杂度 O(n), 空间复杂度 O(1)
- 适用于基本有序的情况

• 希尔排序

- 是直接插入排序算法的改进
- 方法：
  - 设定一个增量序列，例如： 5， 3， 1
  - 按照增量序列依次将原序列切割成多段，比如增量为 5时，将第 0, 5, 10 位置的元素分为一组， 1， 6, 11 分为一组，以此类推
  - 每组间元素进行直接插入排序，排序后的值互换位置，插入回原序列
  - 按照增量序列依次处理，直到增量为 1
- 希尔排序：不稳定、不归位，平均复杂度 O(n^1.3), 最大复杂度 O(n^2), 最小复杂度 O(n), 空间复杂度 O(1)

• 计数排序

- 适合比较少数据量的排序

- 就是将把要排序的数统计一下每种数据有多少个，然后依次添加到序列中

• 简单选择排序

- 从首位开始，依次用后边的关键字与当前的关键字比较，选出最小的那个与当前位替换顺序，直到最后一位为止
- 简单选择排序 ： 不稳定、归位，平均复杂度 O(n^2)， 最大复杂度 O(n^2)， 最小复杂度 O(n^2)， 空间复杂度 O(1)

• 堆排序

- 先将原序列，按照层级遍历，成一个二叉树；然后构造一个大根堆或者小根堆；将堆的根与堆的末位进行交换，然后再次构造大根堆或者小根堆；重复进行操作，直到整个堆变成一个新的序列
- 堆排序：不稳定、归位，平均复杂度 O(nlog2 n)， 最大复杂度 O(nlog2 n)， 最小复杂度 O(nlog2 n)， 空间复杂度 O(1)

• 冒泡排序

- 从原序列首位开始，分别进行 ki 与 k(i+1) 位的比较，如果 如果 ki 更大则交换顺序，这样直到交换到最后一位，可以保证最后一位是最大的，重复进行
- 冒泡排序：稳定、归位，平均复杂度 O(n^2)， 最大复杂度 O(n^2)， 最小复杂度 O(n)， 空间复杂度 O(1)

• 快速排序

- 基于分治的思想，通过一趟排序将待排序的记录分为两个部分（前半区、后半区），前半区的关键字都不大于后半区的关键字，然后载分别对两部分进行快速排序，依次递归操作

- 基本有序序列的快速排序效率最低，时间复杂度最大 O(n^2)
- 快速排序：不稳定、归位，平均复杂度 O(nlog2 n)，最大复杂度 O(n^2)，最小复杂度 O(nlog2 n)，空间复杂度 O(log2 n)

• 归并排序

- 基于分治的思想，将一个序列一分为二，每一半再一分为二，如此递归，直到每一项为 1 个，在从从底向上，每两项间进行比较、每四项间进行比较，如此向上递归，直到最外层；
- 归并排序：稳定、归位，平均复杂度 O(nlog2 n)，最大复杂度 O(nlog2 n)，最小复杂度 O(nlog2 n)，空间复杂度 O(n)

• 回溯法

- N 皇后问题：给定 N * N 的棋盘，要在棋盘上摆放 N 个皇后，皇后中的任意两个都不处于同一行，同一列，同一对角线上
- 判断是否同一列： Qi 列 == Qj 列；判断是否在一个对角线：|Qi 行 - Qj 行 | == |Qi 列 - Qj 列 |
- 代码求解 N 皇后问题
  - ◆ 非递归（循环、迭代）
  - ◆ 递归
- 深度优先
- 主要考的是下午 C 语言计算题，放弃。。。

• 分治法

- 用递归来实现的
- 递归是指自己调用自己，或者间接的自己调用自己，有两个基本要素：1. 需要有边界条件（递归出口），2. 递归模式（递归体）
- 分治法的基本思想：

- 规模越小，解题所需时间越少，越容易处理
- 将一个难以解决的大问题，分解成一些规模较小的相同问题，以便各个击破，分而治之

- 分治算法三个步骤：
  - 3. 分解：原问题分解为子问题
  - 4. 求解：递归求解各个子问题
  - 5. 合并：子问题的解合并成原问题的解

• 动态规划法

- 与分治法类似，基本思想都是将问题分解为若干个子问题，然后求解子问题，在通过子问题的解得到原问题的解
- 不同点：适合动态规划法的问题分解为的子问题往往不是独立的（有相同的子问题）
- 操作上，将动态规划法会用一个表记录所有已解决的子问题答案，在后续计算中如果有相同的问题，则直接找出已求解的答案，避免重复计算
- 动态规划算法，通常用来求解某种最优性质的问题（全局最优解）
- 适合动态规划法求解的问题的两个特征：
  - 最优子结构：一个问题的最优解中包含其子问题的最优解（需要注意，贪心算法也有这个特性）
  - 重叠子问题：原问题的递归算法可反复的解同样的子问题，对每个子问题只解一次，保存在表中，需要时查表

• 0-1 背包问题

- 问题详情表： n 个物品，第 i 个物品价值为 vi，重量 wi，背包容量 W，如何装，使得背包物品价值最大
- 0-1 表示物品要么装入，要么不装入
- 代码实现：放弃~~~

- 背包问题时间复杂度： O(n * w)； 空间复杂度： O(n * w)

• 矩阵连乘

- 由动态规划法实现
- 时间复杂度： O(n^3)； 空间复杂度： O(n^2)
- 计算方法 ：
  - 矩阵 A(mn) 与 B(np) 相乘所需的乘法次数为 m * n * p
  - 相乘后的结果可以类似表示为 AB(mp) ，再次与 C(pk) 相乘则所需乘法次数为 m * p * k
  - 因此 A(mn)，B(np)，C(p*k) 三者相乘的乘法次数可以为 m * n * p + m * p * k
  - 多个矩阵连乘，最优的计算次序是先将 m，n，p，k 中最大的那个乘以，消除掉

• 贪心法

- 贪心法与动态规划法类似，也用来解决最优化问题，但是在解决问题的策略上，贪心法不是从整体最优上考虑，而是局部最优
- 适合贪心法求解的问题的两个特征：
  - 最优子结构 ： 一个问题的最优解中包含其子问题的最优解（需要注意，贪心算法也有这个特性）
  - 贪心选择性质：问题的整体最优解，可以通过一系列局部最优的选择，即贪心选择来实现
  - 部分背包问题
  - 在 0-1 背包问题基础上，物品可以部分装入背包

• 分支限界法

- 类似于回溯法，也是一种在问题的解空间树 T 上搜索问题解的方法，

用来找出满足约束条件的一个解

- 搜索方式上采用广度优先或以最小消耗优先