

全国计算机技术与软件专业资格（水平）考试

软件设计师考试复习一本通

精华知识点+软件设计技巧

最新版



★ 软考题库 - 微信搜一搜

Q 软考达人

目录

使用说明	10
一 计算机系统基础	11
1 数据的表示	11
1.1 进制的转化	11
1.2 数的表示	11
1.3 浮点数的运算	12
1.4 算术运算和逻辑运算	12
2 校验码	13
2.1 奇偶校验码	13
2.2 循环冗余校验码 CRC	13
2.3 海明校验码	14
3 计算机硬件和指令	16
4 计算机体系结构分类	18
5 指令系统 CISC 和 RISC	18
6 指令的流水处理	19
6.1 流水线原理	19
6.2 流水线时间计算	19
7 存储系统	20
7.1 计算机存储结构层次图	20
7.2 局部性原理	21
7.3 高速缓存 Cache	21
7.4 虚拟存储器	22
8 输入输出技术	22
9 总线	23
10 系统可靠性分析	23
二 操作系统知识	24
1 进程管理	24
1.1 进程的组成和状态	24
1.2 前趋图和进程资源图	25

1.3	进程间的同步与互斥	25
1.4	信号量操作	26
1.5	进程调度	27
1.6	死锁问题	27
1.7	线程	28
2	存储管理	28
2.1	分区存储管理	28
2.2	页式存储管理	29
2.3	段式存储管理	31
2.4	段页式存储管理	32
3	文件管理	32
3.1	索引文件结构	32
3.2	文件和树形目录结构	33
3.3	空闲存储空间的管理	33
4	设备管理	34
4.1	I/O 软件	34
4.2	虚设备和 SPOOLING 技术	34
4.3	磁盘调度	35
5	微内核操作系统	36
三	数据库技术基础	36
1	三级模式—两级映像	37
2	数据库的设计	38
3	E-R 模型	39
4	关系代数运算	40
5	关系数据库的规范化	42
5.1	函数依赖	42
5.2	键和约束	42
5.3	范式	43
5.4	模式分解	45
5.5	并发控制基本概念图	48
5.6	事务管理	48

5.7	封锁协议	49
6	数据故障与备份	51
6.1	安全措施	51
6.2	数据故障	51
6.3	数据备份	52
7	分布式数据库	52
8	数据仓库与数据挖掘	53
9	反规范化技术	55
10	大数据	55
11	SQL 语言	56
四	计算机网络	57
1	网络概述和模型	57
2	传输介质	60
3	通信方式和交换方式	62
4	IP 地址	64
4.1	分类地址格式	64
4.2	子网划分	65
4.3	无分类编址	66
5	IPv6	67
6	网络规划和设计	67
6.1	层次化局域网模型	67
6.2	建筑物综合布线系统 PDS	68
7	网络管理命令	68
8	网络存储技术	69
8.1	廉价磁盘冗余阵列	69
8.2	网络存储	70
9	其他考点汇总	71
五	信息安全和网络安全	73
1	信息安全和信息系统安全	73
2	信息安全技术	74
3	网络安全技术	77

4	网络安全协议	79
六	知识产权和标准化	81
1	知识产权基础知识	81
2	标准化基础知识	83
七	程序设计语言基础知识	84
1	程序设计语言基本概念	84
2	程序设计语言的基本成分	85
3	传值与传址	86
4	编译程序基本原理	86
5	表达式	87
6	文法定义	88
7	正规式	88
8	有限自动机	89
9	语法分析方法	90
八	数据结构	90
1	线性结构	90
1.1	线性表	90
1.2	栈和队列	92
1.3	串	94
2	数组、矩阵和广义表	94
2.1	数组	94
2.2	稀疏矩阵	95
2.3	广义表	96
3	树与二叉树	96
3.1	树的相关概念	96
3.2	二叉树的特性	97
3.3	二叉树的顺序存储结构	98
3.4	二叉树的遍历	99
3.5	线索二叉树	99
3.6	最优二叉树（哈夫曼树）	100
3.7	树和森林	101

3.8	查找(排序)二叉树	102
3.9	平衡二叉树	103
4	图	103
4.1	图的基本概念	103
4.2	图的存储	103
4.3	图的遍历	104
4.4	图的最小生成树	105
4.5	拓扑序列	106
九	算法分析与设计	107
1	算法分析	107
1.1	算法的特性	107
1.2	算法的复杂度	107
1.3	经典算法	109
2	查找算法	109
2.1	顺序查找	109
2.2	二分(折半)查找	109
2.3	散列(哈希)表	110
3	排序算法	111
3.1	概念和方法分类	111
3.2	直接插入排序	111
3.3	希尔(shell)排序	112
3.4	简单选择排序	112
3.5	堆排序	113
3.6	冒泡排序	114
3.7	快速排序	115
3.8	归并排序	116
3.9	基数排序	116
3.10	排序算法总结	117
4	算法设计	118
4.1	递推和递归	118
4.2	分治法	119

4.3	回溯法	119
4.4	贪心法	119
4.5	动态规划法	120
4.6	分支限界法	120
4.7	概率算法	120
4.8	近似算法	121
5	数据挖掘算法	121
5.1	分类	121
5.2	频繁模式和关联规则挖掘	122
5.3	聚类	122
6	智能优化算法	122
6.1	人工神经网络 ANN	122
6.2	遗传算法	123
6.3	模拟退火算法 SA	123
6.4	禁忌搜索算法 TS	123
6.5	蚁群算法	123
6.6	粒子群优化算法 PSO	124
十	软件工程基础知识	124
1	软件工程基础	124
1.1	能力成熟度模型	124
1.2	软件过程模型	127
1.3	软件开发方法	128
1.4	软件产品线	132
1.5	逆向工程	132
2	需求工程	133
2.1	软件需求	133
2.2	需求获取	135
2.3	需求分析	135
2.4	需求定义	136
2.5	需求验证	136
2.6	需求管理	137

3	系统设计	138
3.1	处理流程设计	138
3.2	系统设计	140
3.3	人机界面设计	140
4	测试基础知识	141
4.1	测试基础	141
4.2	测试阶段	142
4.3	测试用例设计	143
4.4	调试	145
5	系统运行与维护	146
5.1	系统转换	146
5.2	系统维护概述	147
5.3	系统评价	147
十一	项目管理	148
1	范围管理	148
2	进度管理	149
3	成本管理	152
4	软件配置管理	153
5	质量管理	156
6	风险管理	158
7	组织结构	159
十二	结构化开发方法	160
1	系统分析与设计概述	160
2	结构化开发方法	163
3	结构化设计方法	166
4	WebApp 分析与设计	167
4.1	WebApp 五种需求模型	167
4.2	WebApp 设计	168
十三	面向对象技术	169
1	面向对象基础	169
2	UML	172

2.1	事物	172
2.2	关系	173
2.3	图	174
3	设计模式	179
十四	专业英语	182
软件设计考点及分数说明		189
十五	数据流图 DFD	191
1	数据流图基本概念	191
2	数据字典	192
3	数据流图的设计原则	192
3.1	基本设计原则	192
3.2	父图与子图间的平衡	193
3.3	子图内平衡	193
4	解题技巧	193
十六	数据库设计	194
1	E-R 图转换为关系模式	194
2	解题技巧	195
十七	UML 建模	196
1	用例图	196
2	类图	197
3	序列图	198
4	活动图	199
5	状态图	199
6	通信（协作）图	200
7	解题技巧	200
十八	算法设计与 C 程序实现	201
1	解题技巧	201
十九	面向对象的程序设计与实现	202
1	C++ 基本语法	202
2	JAVA 基本语法	202
3	解题技巧	204

使用说明

本次更新说明

章节重新划分，详见目录，和习题、视频章节保持一致。

计算机系统基础——删除信息安全。

计算机网络——删除多媒体和网络安全。

新增 信息安全和网络安全 章节。

软件工程基础，新增内容。

新增项目管理章节。

结构化开放防范，修改。

面向对象技术，修改。

新增专业英语词汇。

本资料使用说明

1、本资料现在定位已经发生变化，不再用于第一阶段学习。第一阶段学习，请大家按照群公告学习计划，利用 视频+课件+课后习题+官方教材（如果时间充足）的方式去整体过一遍所有知识点。

2、本资料建议用在第 2,3 阶段学习使用，在有一定的基础前提下，直接看精华知识点，效果才更好。

3、本资料建议打印出来（不整体装订，随身带几张），利用碎片化时间多学多记，同时建议至少在以下两个时间节点利用整段时间整体看一遍：

第一遍：完成第一阶段学习之后，开始做历年上午真题试卷之前。

第二遍：考前半个月，一鼓作气整体再看一遍。

4、本书是总结性质的精华知识点概括，**配合文老师软件设计师视频来学习能够事半功倍**。买文老师视频花的钱很少，但是大家的时间才是最宝贵的，如果决定要考了，不管有什么困难都要克服，一鼓作气，不要让自己的时间白费。

5、本资料校对工作量巨大，难以避免有一些差错，学员学习过程中有任何疑问，可以直接私聊文老师，指明具体疑问地方，老师会及时解惑，并在下次更新时修订。

6、本资料是文老师软件设计师视频课程学员内部学习资料，仅供学习、打印，禁止编辑，禁止传播给任何其他人，要获取更新请直接联系淘宝旺旺客服或私聊群内客服助理。

一 计算机系统基础

1 数据的表示

1.1 进制的转化

必须要掌握，视频的讲解要搞懂。但不是直接知识点，故在精华知识点里删除。

1.2 数的表示

各种数值在计算机中表示的形式称为机器数，其特点使用二进制计数制，数的符号用 0 和 1 表示，小数点则隐含，不占位置。

机器数有无符号数和带符号数之分。无符号数表示正数，没有符号位。带符号数最高位为符号位，正数符号位为 0，负数符号位为 1。定点表示法分为纯小数和纯整数两种，其中小数点不占存储位，而是按照以下约定：

纯小数：约定小数点的位置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。

带符号数有下列编码方式：

原码：一个数的正常二进制表示，最高位表示符号，数值 0 的源码有两种形式： $+0$ (0 0000000) 和 -0 (1 0000000)。

反码：正数的反码即原码；负数的反码是在原码的基础上，除符号位外，其他各位按位取反。数值 0 的反码也有两种形式： $+0$ (0 0000000)， -0 (1 1111111)。

补码：正数的补码即原码；负数的补码是在原码的基础上，除符号位外，其他各位按位取反，而后末位+1，若有进位则产生进位。因此数值 0 的补码只有一种形式 $+0 = -0 = 0$ 0000000。

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）取反得到移码。

符号表示：要注意的是，原码最高位是代表正负号，且不参与计数；而其他编码最高位虽然也是代表正负号，但参与计数。

机器字长为 n 时各种码制表示的带符号数的取值范围：

码 制	定 点 整 数	定 点 小 数
原码	$-(2^{n-1}-1) \sim +(2^{n-1}-1)$	$-(1-2^{-(n-1)}) \sim +(1-2^{-(n-1)})$
反码	$-(2^{n-1}-1) \sim +(2^{n-1}-1)$	$-(1-2^{-(n-1)}) \sim +(1-2^{-(n-1)})$
补码	$-2^{n-1} \sim +(2^{n-1}-1)$	$-1 \sim +(1-2^{-(n-1)})$
移码	$-2^{n-1} \sim +(2^{n-1}-1)$	$-1 \sim +(1-2^{-(n-1)})$

差别在于 0 的表示，原码和反码分+0 和-0，补码只有一个 0，因此可以多表示一个。

1.3 浮点数的运算

浮点数：表示方法为 $N = F * 2^E$ ，其中 E 称为阶码，F 称为尾数；类似于十进制的科学计数法，如 $85.125 = 0.85125 * 10^2$ ，二进制如 $101.011 = 0.101011 * 2^3$ 。

在浮点数的表示中，阶码为带符号的纯整数，尾数为带符号的纯小数，要注意符号占最高位（正数 0 负数 1），其表示格式如下：

阶符	阶码	数符	尾数
----	----	----	----

很明显，与科学计数法类似，一个浮点数的表示方法不是唯一的，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。

尾数的表示采用规格化方法，也即带符号尾数的补码必须为 1.0xxxx（负数）或者 0.1xxxx（正数），其中 x 可为 0 或 1。

浮点数的运算：对阶（使两个数的阶码相同，**小阶向大阶看齐**，较小阶码增加几位，尾数就右移几位）——尾数计算（相加，若是减运算，则加负数）——结果规格化（即尾数表示规格化，带符号尾数转换为 1.0xxxx 或 0.1xxxx）。

1.4 算术运算和逻辑运算

数与数之间的算术运算包括加、减、乘、除等基本算术运算，对于二进制数，还应该掌握基本逻辑运算，包括：

逻辑与&&：0 和 1 相与，只要有一个为 0 结果就为 0，两个都为 1 才为 1。

逻辑或||：0 和 1 相或，只要有一个为 1 结果就为 1，两个都为 0 才为 0。

异或：同 0 非 1，即参加运算的二进制数同为 0 或者同为 1 结果为 0，一个为 0 另一个为 1 结果为 1。

逻辑非!：0 的非是 1，1 的非是 0。

逻辑左移<<：二进制数整体左移 n 位，高位若溢出则舍去，低位补 0。

逻辑右移>>: 二进制数整体右移 n 位, 低位溢出则舍去, 高位补 0。

算术左移算术右移: 乘以 2 或者除以 2 的算术运算, 涉及加减乘除的都是算术运算, 与逻辑运算区分。

短路计算方式: 指通过逻辑运算符(&&、||)左边表达式的值就能推算出整个表达式的值, 不再继续执行逻辑运算符右边的表达式。

若计算机存储数据采用的是**双符号位**(00 表示正号、11 表示负号), 两个符号相同的数相加时, 如果运算结果的两个符号位经异或运算得 1, 则可断定这两个数相加的结果产生了溢出。

从计算的角度理解, 正数和负数相加其结果肯定不会溢出, 如果有溢出, 必然同为正数或者同为负数, 结果才会更大, 有可能溢出, 因此, 正常两个同符号数相加时, 不考虑溢出, 其符号位必然还是 00 或者 11, 如果有溢出, 那么数据位必然最高位进位 1, 符号位就需要加 1, 符号位就变为 01 或者 10, 因此当符号位为 01 或者 10 时数据溢出, 观察这两种溢出情况的两个符号位都是一个为 0, 一个为 1, 其异或运算必然为 1, 没有其他可能, 而逻辑或运算有可能两个都为 1 也能得出 1。

2 校验码

码距: 就单个编码 A: 00 而言, 其码距为 1, 因为其只需要改变一位就变成另一个编码。在两个编码中, 从 A 码到 B 码转换所需要改变的位数称为码距, 如 A: 00 要转换为 B: 11, 码距为 2。一般来说, 码距越大, 越利于纠错和检错。

2.1 奇偶校验码

在编码中增加 1 位校验位来使编码中 1 的个数为奇数 (奇校验) 或者偶数 (偶校验), 从而使码距变为 2。奇校验可以检测编码中奇数个数据位出错, 即当合法编码中的奇数位发生了错误时, 即编码中的 1 变成 0 或者 0 变成 1, 则该编码中 1 的个数的奇偶性就发生了变化, 从而检查出错误。但无法纠错。

2.2 循环冗余校验码 CRC

CRC 只能检错, 不能纠错, 其原理是找出一个能整除多项式的编码, 因此首先要将原始报文除以多项式, 将所得的余数作为校验位加在原始报文之后, 作为发送数据发给接收方。

使用 CRC 编码, 需要先约定一个生成多项式 $G(x)$ 。生成多项式的最高位和最低位必须是 1。假设原始信息有 m 位, 则对应多项式 $M(x)$ 。生成校验码思想就是在原始信息位后追加若干校验位, 使得

追加的信息能被 $G(x)$ 整除。接收方接收到带校验位的信息，然后用 $G(x)$ 整除。余数为 0，则没有错误；反之则发生错误。

例：假设原始信息串为 10110，CRC 的生成多项式为 $G(x)=x^4+x+1$ ，求 CRC 校验码。

(1) 在原始信息位后面添 0，假设生成多项式的阶为 r ，则在原始信息位后添加 r 个 0，本题中， $G(x)$ 阶为 4，则在原始信息串后加 4 个 0，得到的新串为 101100000，作为被除数。

(2) 由多项式得到除数，多项中 x 的幂指数存在的位置 1，不存在的位置 0。本题中， x 的幂指数为 0,1,4 的变量都存在，而幂指数为 2,3 的不存在，因此得到串 10011。

(3) 生成 CRC 校验码，将前两步得出的被除数和除数进行模 2 除法运算（即不进位也不借位的除法运算）。除法过程如下图所示。

$$\begin{array}{r}
 10011 \overline{) 101100000} \\
 \underline{10011} \\
 10100 \\
 \underline{10011} \\
 11100 \\
 \underline{10011} \\
 1111
 \end{array}$$

得到余数 1111。

注意：余数不足 r ，则余数左边用若干个 0 补齐。如求得余数为 11， $r=4$ ，则补两个 0 得到 0011。

(4) 生成最终发送信息串，将余数添加到原始信息后。上例中，原始信息为 10110，添加余数 1111 后，结果为 10110 1111。发送方将此数据发送给接收方。

(5) 接收方进行校验。接收方的 CRC 校验过程与生成过程类似，接收方接收了带校验和的帧后，用多项式 $G(x)$ 来除。余数为 0，则表示信息无错；否则要求发送方进行重传。

注意：收发信息双方需使用相同的生成多项式。

2.3 海明校验码

本质也是使用奇偶校验方式检验，通过下面例题详解海明校验码：

例：求信息 1011 的海明码。

1. 校验位的位数和具体的数据位的位数之间的关系

所有位都编号，从最低位编号，从 1 开始递增，校验位处于 2 的 n ($n=0, 1, 2, \dots$) 次方中，即处于第 1, 2, 4, 8, 16, 32, $\dots$ 位上，其余位才能填充真正的数据位，若信息数据为 1011，则可知，第 1, 2, 4 位为校验位，第 3, 5, 6, 7 位为数据位，用来从低位开始存放 1011，得出信息位和校验位分布如下：

7	6	5	4	3	2	1	位数
I_4	I_3	I_2		I_1			信息位
			r_2		r_1	r_0	校验位

实际考试时可以依据公式 $n+k \leq 2^k - 1$ 快速得出答案 (n 是已知的数据位个数， k 是未知的校验位个数，依次取 $k=1$ 代入计算，得出满足上式的最小的 k 的值)。

2. 计算校验码

将所有信息位的编号都拆分成二进制表示，如下图所示：

$$\begin{aligned}
 7 &= 2^2 + 2^1 + 2^0, & 6 &= 2^2 + 2^1, & 5 &= 2^2 + 2^0, & 3 &= 2^1 + 2^0; \\
 r_2 &= I_4 \oplus I_3 \oplus I_2; \\
 r_1 &= I_4 \oplus I_3 \oplus I_1; \\
 r_0 &= I_4 \oplus I_2 \oplus I_1;
 \end{aligned}$$

上图中， $7=4+2+1$ ，表示 7 由第 4 位校验位(r_2)和第 2 位校验位(r_1)和第 1 位校验位(r_0)共同校验，同理，第 6 位数据位 $6=4+2$ ，第 5 位数据位 $5=4+1$ ，第 3 位数据位 $3=2+1$ ，前面知道，这些 2 的 n 次方都是校验位，可知，第 4 位校验位校验第 7 6 5 三位数据位，因此，第 4 位校验位 r_2 等于这三位数据位的值异或，第 2 位和第 1 位校验位计算原理同上，最终得到如下：

7	6	5	4	3	2	1	位数
1	0	1		1			信息位
			0		0	1	校验位

计算出三个校验位后，可知最终要发送的海明校验码为 1010101。

3. 检错和纠错原理

接收方收到海明码之后，会将每一位校验位与其校验的位数分别异或，即做如下三组运算：

$$\begin{aligned}
 r_2 &\oplus I_4 \oplus I_3 \oplus I_2; \\
 r_1 &\oplus I_4 \oplus I_3 \oplus I_1; \\
 r_0 &\oplus I_4 \oplus I_2 \oplus I_1;
 \end{aligned}$$

如果是偶校验，那么运算得到的结果应该全为 0，如果是奇校验，应该全为 1，才是正确，假设是偶校验，且接收到的数据为 1011101 (第四位出错)，此时，运算的结果为：

$$\begin{aligned}r_2 \oplus I_4 \oplus I_3 \oplus I_2 &= 1 \oplus 1 \oplus 0 \oplus 1 = 1 \\r_1 \oplus I_4 \oplus I_3 \oplus I_1 &= 0 \oplus 1 \oplus 0 \oplus 1 = 0 \\r_0 \oplus I_4 \oplus I_2 \oplus I_1 &= 1 \oplus 1 \oplus 1 \oplus 1 = 0\end{aligned}$$

这里不全为 0，表明传输过程有误，并且按照 $r_2r_1r_0$ 排列为二进制 100，这里指出的就是错误的位数，表示第 100，即第 4 位出错，找到了出错位，纠错方法就是将该位逆转。

3 计算机硬件和指令

计算机的硬件基本系统由五大部分组成：运算器、控制器、存储器、输入设备（如鼠标键盘）、输出设备（如显示器）。

其中，运算器和控制器合并称为中央处理单元，即 CPU；存储器分为内部存储器（即内存，容量小，速度快，临时存放数据）和外部存储器（即硬盘、光盘等，容量大，速度慢，长期保存数据）；输入设备和输出设备合并称为外设。

鼠标键盘等输入设备都是通过中断的原理来实现控制，点击后触发中断，首先进入中断处理程序。

主机：CPU（运算器、控制器）+主存储器。

CPU：由运算器、控制器、寄存器组和内部总线组成。实现程序控制、操作控制、时间控制、数据处理功能。

运算器：由算术逻辑单元 ALU（实现对数据的算术和逻辑运算）、累加寄存器 AC（运算结果或源操作数的存放区）、数据缓冲寄存器 DR（暂时存放内存的指令或数据）、和状态条件寄存器 PSW（保存指令运行结果的条件码内容，如溢出标志等）组成。执行所有的算术运算，如加减乘除等；执行所有的逻辑运算并进行逻辑测试，如与、或、非、比较等。

控制器：由指令寄存器 IR（暂存 CPU 执行指令）、程序计数器 PC（存放指令执行地址）、地址寄存器 AR（保存当前 CPU 所访问的内存地址）、指令译码器 ID（分析指令操作码）等组成。控制整个 CPU 的工作，最为重要。

CPU 依据指令周期的不同阶段来区分二进制的指令和数据，因为在指令周期的不同阶段，指令会命令 CPU 分别去取指令或者数据。

计算机指令的组成

一条指令由操作码和操作数两部分组成，操作码决定要完成的操作，操作数指参加运算的数据及其所在的单元地址。

在计算机中，操作要求和操作数地址都由二进制数码表示，分别称作操作码和地址码，整条指令以二进制编码的形式存放在存储器中。

计算机指令执行过程

可分为取指令——分析指令——执行指令三个步骤，首先将程序计数器 PC 中的指令地址取出，送入地址总线，CPU 依据指令地址去内存中取出指令内容存入指令寄存器 IR；而后由指令译码器进行分析，分析指令操作码；最后执行指令，取出指令执行所需的源操作数。

指令寻址方式

顺序寻址方式：由于指令地址在主存中顺序排列，当执行一段程序时，通常是一条指令接着一条指令地顺序执行。从存储器取出第一条指令，然后执行这条指令；接着从存储器取出第二条指令，再执行第二条指令；……，以此类推。这种程序顺序执行的过程称为指令的顺序寻址方式。

跳跃寻址方式：所谓指令的跳跃寻址，是指下一条指令的地址码不是由程序计数器给出，而是由本条指令直接给出。程序跳跃后，按新的指令地址开始顺序执行。因此，指令计数器的内容也必须相应改变，以便及时跟踪新的指令地址。

指令操作数的寻址方式

立即寻址方式：指令的地址码字段指出的不是地址，而是操作数本身。

直接寻址方式：在指令的地址字段中直接指出操作数在主存中的地址。

间接寻址方式：与直接寻址方式相比，间接寻址中指令地址码字段所指向的存储单元中存储的不是操作数本身，而是操作数的地址。

寄存器寻址方式：指令中的地址码是寄存器的编号，而不是操作数地址或操作数本身。寄存器的寻址方式也可以分为直接寻址和间接寻址，两者的区别在于：前者的指令地址码给出寄存器编号，寄存器的内容就是操作数本身；而后者的指令地址码给出寄存器编号，寄存器的内容是操作数的地址，根据该地址访问主存后才能得到真正的操作数。

基址寻址方式：将基址寄存器的内容加上指令中的形式地址而形成操作数的有效地址，其优点是可以扩大寻址能力。

变址寻址方式：变址寻址方式计算有效地址的方法与基址寻址方式很相似，它是将变址寄存器的内容加上指令中的形式地址而形成操作数的有效地址。

相对寻址方式：相对于当前的指令地址而言的寻址方式。相对寻址是把程序计数器 PC 的内容加上指令中的形式地址而形成操作数的有效地址，而程序计数器的内容就是当前指令的地址，所以相对寻址是相对于当前的指令地址而言的。

4 计算机体系结构分类

按处理机的数量进行分类：单处理系统（一个处理单元和其他设备集成）、并行处理系统（两个以上的处理机互联）、分布式处理系统（物理上远距离且松耦合的多计算机系统）。

Flynn 分类法：如下图所示

体系结构类型	结构	关键特性	代表
单指令流单数据流 SISD	控制部分：一个 处 理 器：一个 主存模块：一个		单处理器系统
单指令流多数据流 SIMD	控制部分：一个 处 理 器：多个 主存模块：多个	各处理器以异步的形式执行同一条指令	并行处理机 阵列处理机 超级向量处理机
多指令流单数据流 MISD	控制部分：多个 处 理 器：一个 主存模块：多个	被证明不可能，至少是不实际	目前没有，有文献称流水线计算机为此类
多指令流多数据流 MIMD	控制部分：多个 处 理 器：多个 主存模块：多个	能够实现作业、任务、指令等各级全面并行	多处理机系统 多计算机

由上图可知，分类有两个因素，即指令流和数据流，指令流由控制部分处理，每一个控制部分处理一条指令流，**多指令流就有多个控制部分**；数据流由处理器来处理，每一个处理器处理一条数据流，**多数据流就有多个处理器**；至于主存模块，是用来存储的，存储指令流或者数据流，因此，**无论是多指令流还是多数据流，都需要多个主存模块来存储**，对于主存模块，指令和数据都一样。

依据计算机特性，是由指令来控制数据的传输，因此，一条指令可以控制一条或多条数据流，但一条数据流不能被多条指令控制，否则会出错，就如同上级命令太多还互相冲突，不知道该执行哪个，因此多指令单数据 MISD 不可能。

5 指令系统 CISC 和 RISC

CISC 是复杂指令系统，兼容性强，指令繁多、长度可变，由微程序实现；

RISC 是精简指令系统，指令少，使用频率接近，主要依靠硬件实现（通用寄存器、硬布线逻辑控制）。

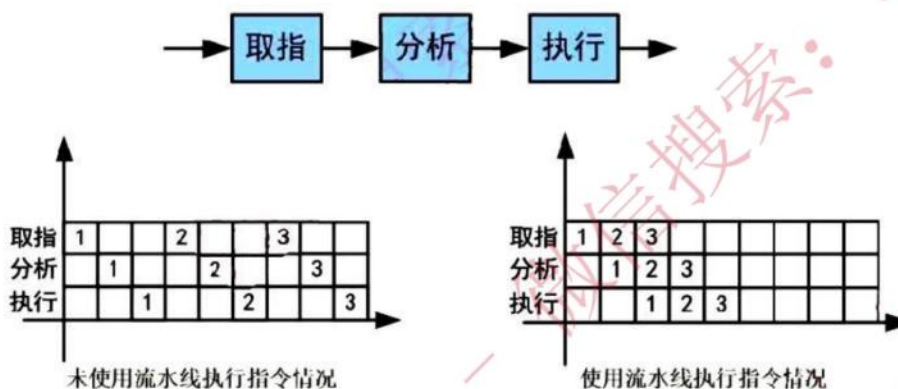
二者各方面区分如下图：

指令系统类型	指令	寻址方式	实现方式	其它
CISC (复杂)	数量多, 使用频率差别大, 可变长格式	支持多种	微程序控制技术 (微码)	研制周期长
RISC (精简)	数量少, 使用频率接近, 定长格式, 大部分为单周期指令, 操作寄存器, 只有Load/Store操作内存	支持方式少	增加了通用寄存器; 硬布线逻辑控制为主; 适合采用流水线	优化编译, 有效支持高级语言

6 指令的流水处理

6.1 流水线原理

将指令分成不同段, 每段由不同的部分去处理, 因此可以产生叠加的效果, 所有的部件去处理指令的不同段, 如下图所示:



RISC 中的流水线技术: 超流水线 (在每个机器周期内能完成一个甚至两个浮点操作, 以时间换空间)、超标量 (内装多条流水线同时执行多个处理, 以空间换时间)、超长指令字 VLIW (同时执行多条指令, 发挥软件作用)。

6.2 流水线时间计算

流水线周期: 指令分成不同执行段, 其中执行时间最长的段为流水线周期。

流水线执行时间: 1 条指令总执行时间 + (总指令条数 - 1) * 流水线周期。

单缓冲区和双缓冲区: 此类题型不给出具体流水线执行阶段, 需要考生自己区分出流水线阶段, 一般来说, 能够同时执行的阶段就是流水线的独立执行阶段; 只能独立执行的阶段应该合并为流水线中的一个独立执行阶段。

例如有三个阶段即读入缓冲区+送入用户区+数据处理，在单缓冲区中，缓冲区和用户区都只有一个，一个盘块必须执行完前两个阶段，下一个盘块才能开始，因此前两个阶段应该合并，整个流水线为送入用户区+数据处理；

而在双缓冲区中，盘块可以交替读入缓冲区，但用户区只有一个，因为缓冲区阶段可以同时进行，流水线前两个阶段不能合并，就是读入缓冲区+送入用户区+数据处理三段。

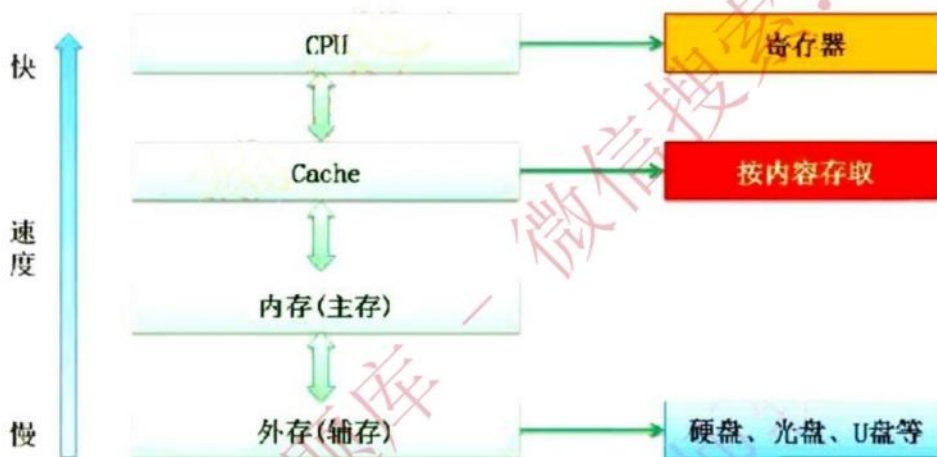
划分出真正的流水线阶段后，套用流水线时间计算公式可以轻易得出答案。

流水线吞吐率计算：吞吐率即单位时间内执行的指令条数。公式：指令条数/流水线执行时间。

流水线的加速比计算：加速比即使用流水线后的效率提升度，即比不使用流水线快了多少倍，越高表明流水线效率越高，公式：不使用流水线执行时间/使用流水线执行时间。

7 存储系统

7.1 计算机存储结构层次图



计算机采用分级存储体系的主要目的是为了解决存储容量、成本和速度之间的矛盾问题。

两级存储映像为：Cache-主存、主存-辅存（虚拟存储体系）。

存储器的分类：

按存储器所处的位置：内存、外存。

按存储器构成材料：磁存储器（磁带）、半导体存储器、光存储器（光盘）。

按存储器的工作方式：可读可写存储器（RAM）、只读存储器（ROM 只能读，PROM 可写入一次，EPROM 和 EEPROM 既可以读也可以写，只是修改方式不用）。

按存储器访问方式：按地址访问、按内容访问（相联存储器）。

按寻址方式：随机存储器（访问任意存储单元所用时间相同）、顺序存储器（只能按顺序访问，如磁带）、直接存储器（二者结合，如磁盘，对于磁道的寻址是随机的，在一个磁道内则是顺序的）。

7.2 局部性原理

总的来说,在 CPU 运行时,所访问的数据会趋向于一个较小的局部空间地址内。(例如循环操作,循环体被反复执行)

时间局部性原理:如果一个数据项正在被访问,那么在近期它很可能被再次访问,即在**相邻的时间里会访问同一个数据项**。

空间局部性原理:在最近的将来会用到的数据的地址和现在正在访问的数据地址很可能是相近的,即**相邻的空间地址会被连续访问**。

7.3 高速缓存 Cache

高速缓存 Cache 用来存储当前最活跃的程序和数据,直接与 CPU 交互,位于 CPU 和主存之间,容量小,速度为内存的 5-10 倍,由半导体材料构成。其内容是主存内存的副本拷贝,对于程序员来说是透明的。

Cache 由控制部分和存储器组成,存储器存储数据,控制部分判断 CPU 要访问的数据是否在 Cache 中,在则命中,不在则依据一定的算法从主存中替换。

地址映射方法

在 CPU 工作时,送出的是主存单元的地址,而应从 Cache 存储器中读/写信息。这就需要将主存地址转换为 Cache 存储器地址,这种地址的转换称为地址映像,由**硬件自动完成映射**,分为下列三种方法:

直接映像:将 Cache 存储器等分成块,主存也等分成块并编号。主存中的块与 Cache 中的块的对应关系是固定的,也即二者块号相同才能命中。地址变换简单但不灵活,容易造成资源浪费。

全相联映像:同样都等分成块并编号。主存中任意一块都与 Cache 中任意一块对应。因此可以随意调入 Cache 任意位置,但地址变换复杂,速度较慢。因为主存可以随意调入 Cache 任意块,只有当 Cache 满了才会发生块冲突,是最不容易发生块冲突的映像方式。

组组相连映像:前面两种方式的结合,将 Cache 存储器先分块再分组,主存也同样先分块再分组,组间采用直接映像,即主存中组号与 Cache 中组号相同的组才能命中,但是组内全相联映像,也即组号相同的两个组内的所有块可以任意调换。

命中率及平均时间

Cache 存储器的大小一般为 K 或者 M 单位，很小，但是最快，仅次于 CPU 中的寄存器，而寄存器一般不算作存储器，CPU 与内存之间的数据交互，内存会先将数据拷贝到 Cache 里，这样，根据局部性原理，若 Cache 中的数据被循环执行，则不用每次都去内存中读取数据，会加快 CPU 工作效率。

因此，Cache 有一个命中率的概念，即当 CPU 所访问的数据在 Cache 中时，命中，直接从 Cache 中读取数据，设读取一次 Cache 时间为 1ns，若 CPU 访问的数据不在 Cache 中，则需要从内存中读取，设读取一次内存的时间为 1000ns，若在 CPU 多次读取数据过程中，有 90% 命中 Cache，则 CPU 读取一次的平均时间为 $(90\% \times 1 + 10\% \times 1000) \text{ns}$ ，很容易理解。

7.4 虚拟存储器

虚拟存储器技术是将很大的数据分成许多较小的块，全部存储在外存中。运行时，将用到的数据调入主存中，马上要用到的数据置于缓存中，这样，一边运行一边进行所需数据块的调入/调出。对于应用程序员来说，就好像有一个比实际主存空间大得多的虚拟主存空间，基本层级为：主存——缓存——外存。与 CPU——高速缓存 Cache——主存的原理类似。但虚拟存储器中程序员无需考虑地址映像关系，由系统自动完成，因此对于程序来说是透明的。

其管理方式分为页式、段式、段页式，在 2.2 存储管理中详细介绍。

8 输入输出技术

程序控制（查询）方式：CPU 主动查询外设是否完成数据传输，效率极低。

程序中断方式：外设完成数据传输后，向 CPU 发送中断，等待 CPU 处理数据，效率相对较高。中断响应时间指的是从发出中断请求到开始进入中断处理程序；中断处理时间指的是从中断处理开始到中断处理结束。中断向量提供中断服务程序的入口地址。多级中断嵌套，使用堆栈来保护断点和现场。

DMA 方式（直接主存存取）：CPU 只需完成必要的初始化等操作，数据传输的整个过程都由 DMA 控制器来完成，在主存和外设之间建立直接的数据通路，效率很高。在一个总线周期结束后，CPU 会响应 DMA 请求开始读取数据；CPU 响应程序中断方式请求是在一条指令执行结束时；区分指令执行结束和总线周期结束。

通道：也是一种处理机，内部具有独立的处理系统，使数据的传输独立于 CPU。分为字节多路通道的传送方式（每一次传送一个通道的一个字节，多路通道循环）和选择通道的传送方式（选择一个通道，先传送完这个通道的所有字节，再开始下一个通道传送）。

9 总线

从广义上讲，任何连接两个以上电子元器件的导线都可以称为总线，通常分为以下三类：

内部总线：内部芯片级别的总线，芯片与处理器之间通信的总线。

系统总线：是板级总线，用于计算机内各部分之间的连接，具体分为数据总线（并行数据传输位数）、地址总线（系统可管理的内存空间的大小）、控制总线（传送控制命令）。代表的有 ISA 总线、EISA 总线、PCI 总线。

外部总线：设备一级的总线，微机和外部设备的总线。代表的有 RS232（串行总线）、SCSI（并行总线）、USB（通用串行总线，即插即用，支持热插拔）。

并行总线适合近距离高速数据传输，串行总线适合长距离数据传输，专用总线在设计上可以与连接设备实现最佳匹配。

总线计算：总线的时钟周期=时钟频率的倒数；总线的宽度（传输速率）=单位时间内传输的数据总量/单位时间大小。

10 系统可靠性分析

串并联系统可靠性

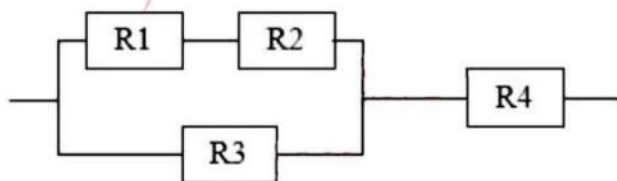
无论什么系统，都是由多个设备组成的，协同工作，而这多个设备的组合方式可以是串联、并联，也可以是混合模式，假设每个设备的可靠性为 $R_1, R_2, \dots, R_n$ ，则不同的系统的可靠性公式如下：

串联系统，一个设备不可靠，整个系统崩溃，整个系统可靠性 $R = R_1 * R_2 * \dots * R_n$ 。

并联系统，所有设备都不可靠，整个系统才崩溃，整个系统可靠性 $R = 1 - (1 - R_1) * (1 - R_2) * \dots * (1 - R_n)$ 。

混合系统，划分串联、并联：

例：某计算机系统的可靠性结构如下所示，若所构成系统的每个部件的可靠度分别为 R_1 、 R_2 、 R_3 和 R_4 ，则该系统的可靠度为（ ）。



解析：由图可知， R_4 和 R_1, R_2, R_3 子系统串联， R_1, R_2 和 R_3 并联，系统可靠性为： $(1 - (1 - R_1)(1 - R_2)(1 - R_3))R_4$ 。

二 操作系统知识

操作系统的作用：通过资源管理提高计算机系统的效率；改善人机界面向用户提供友好的工作环境。

操作系统的特征：并发性、共享性、虚拟性、不确定性。

操作系统的功能：进程管理、存储管理、文件管理、设备管理、作业管理。

操作系统的分类：批处理操作系统、分时操作系统（轮流使用 CPU 工作片）、实时操作系统（快速响应）、网络操作系统、分布式操作系统（物理分散的计算机互联系统）、微机操作系统（Windows）、嵌入式操作系统。

嵌入式系统**初始化过程**按照自底向上、从硬件到软件的次序依次为：片级初始化→板级初始化→系统初始化。芯片级是微处理器的初始化，板卡级是其他硬件设备初始化，系统级初始化就是软件及操作系统初始化。

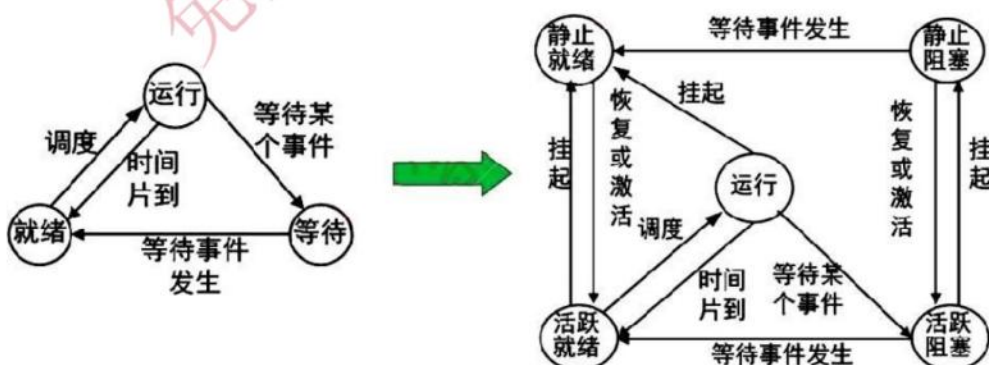
1 进程管理

1.1 进程的组成和状态

进程的组成：进程控制块 PCB（唯一标志）、程序（描述进程要做什么）、数据（存放进程执行时所需数据）。

进程基础的状态是下左图中的**三态图**，这是系统自动控制时只有三种状态，而下右图中的五态，是多了两种状态：静止就绪和静止阻塞，需要人为的操作才会进入对应状态，活跃就绪即就绪，活跃阻塞即等待。

需要熟练掌握左下图中的进程三态之间的转换。

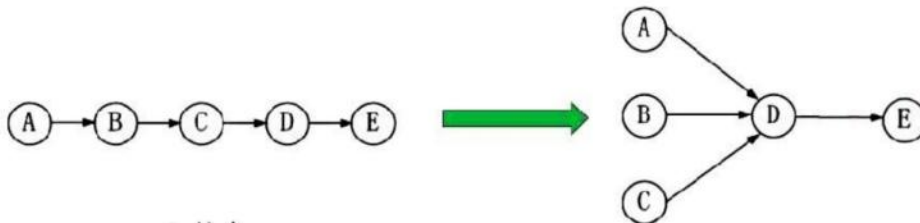


可知，当人为干预后，进程将被挂起，进入静止状态，此时，需要人为激活，才能恢复到活跃状态，之后的本质还是三态图。

1.2 前趋图和进程资源图

前趋图

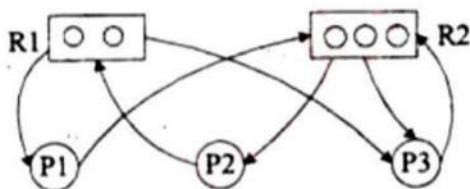
用来表示哪些任务可以并行执行，哪些任务之间有顺序关系，具体如下图：



可知，ABC 可以并行执行，但是必须 ABC 都执行完后，才能执行 D，这就确定了两点：任务间的并行、任务间的先后顺序。

进程资源图

用来表示进程和资源之间的分配和请求关系，如下图所示：



P 代表进程，R 代表资源，R 方框中有几个圆球就表示有几个这种资源，在上图中，R1 指向 P1，表示 R1 有一个资源已经分配给了 P1，P1 指向 R2，表示 P1 还需要请求一个 R2 资源才能执行。

阻塞节点：某进程所请求的资源已经全部分配完毕，无法获取所需资源，该进程被阻塞了无法继续。如上图中 P2。

非阻塞节点：某进程所请求的资源还有剩余，可以分配给该进程继续运行。如上图中 P1、P3。

当一个进程资源图中所有进程都是阻塞节点时，即陷入死锁状态。

进程资源图的化简方法：先看系统还剩下多少资源没分配，再看有哪些进程是不阻塞的，接着把不阻塞的进程的所有边都去掉，形成一个孤立的点，再把系统分配给这个进程的资源回收回来，这样，系统剩余的空闲资源便多了起来，接着又去看看剩下的进程有哪些是不阻塞的，然后又把它们逐个变成孤立的点。最后，所有的资源和进程都变成孤立的点。

1.3 进程间的同步与互斥

互斥和同步并非反义词，互斥表示一个资源在同一时间内只能由一个任务单独使用，需要加锁，使用完后解锁才能被其他任务使用；

同步表示两个任务可以同时执行，只不过有速度上的差异，需要速度上匹配，不存在资源是否单独或共享的问题。

1.4 信号量操作

基本概念

临界资源：各个进程间需要互斥方式对其进行共享的资源，即在某一时刻只能被一个进程使用，该进程释放后又可以被其他进程使用。

临界区：每个进程中访问临界资源的那段代码。

信号量：是一种特殊的变量。

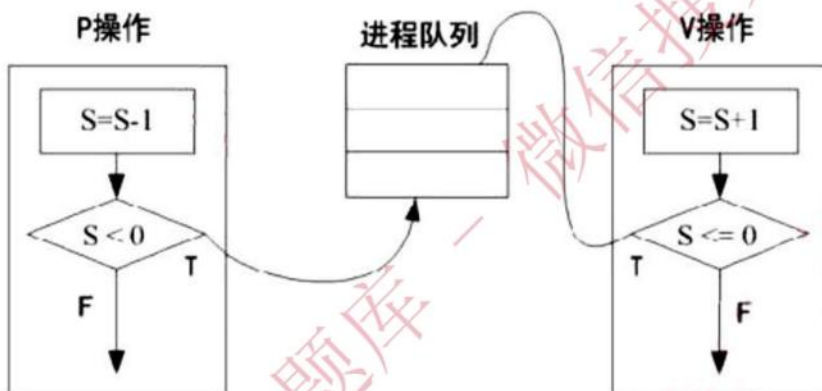
两类信号量

互斥信号量，对临界资源采用互斥访问，使用互斥信号量后其他进程无法访问，初值为 1。

同步信号量，对共享资源的访问控制，初值是共享资料的个数。

P 操作和 V 操作

都是原子操作，用来解释进程间的同步和互斥原理，PV 操作原理如下图所示：



例如在**生产者和消费者**的问题中，生产者生产一个商品 S ，而后要申请互斥的使用该仓库，即首先需要执行互斥信号量 $P(S_0)$ ，申请到仓库独立使用权后，再判断**仓库是否有空闲**（信号量 S_1 ），执行 $P(S_1)$ ，若结果大于等于 0，表示仓库有空闲，再将 S 放入仓库中，此时**仓库商品数量**（信号量 S_2 ）增加 1，即执行 $V(S_2)$ 操作，使用完毕后，释放互斥信号量 $V(S_0)$ 。

对于消费者，首先也需要执行互斥信号量 $P(S_0)$ ，申请到仓库独立使用权后，再判断仓库中是否有商品，执行 $P(S_2)$ ，若结果大于等于 0，表示有商品，可以取出，此时造成了一个结果，即仓库空闲了一个，执行 $V(S_1)$ 操作，使用完毕后，释放互斥信号量 $V(S_0)$ 。

因此，执行 P 操作是**主动的带有判断性质**的-1，执行 V 操作是**被动的因为某操作产生的**+1。

1.5 进程调度

进程调度方式是指当有更高优先级的进程到来时如何分配 CPU。分为可剥夺和不可剥夺两种，可剥夺指当有更高优先级进程到来时，强行将正在运行进程的 CPU 分配给高优先级进程；不可剥夺是指高优先级进程必须等待当前进程自动释放 CPU。

三级调度：**高级调度**（又称长调度或作业调度，决定哪个作业可以调入系统中）、**中级调度**（又称对换调度，决定哪个就绪进程可以调入内存中）、**低级调度**（又称进程调度，决定内存中哪个就绪进程可以占用 CPU）。

调度算法：

先来先服务 FCFS：先到达的进程优先分配 CPU。用于宏观调度。

时间片轮转：分配给每个进程 CPU 时间片，轮流使用 CPU，每个进程时间片大小相同，很公平，用于微观调度。

优先级调度：每个进程都拥有一个优先级，优先级大的先分配 CPU。

多级反馈调度：时间片轮转和优先级调度结合而成，设置多个就绪队列 1,2,3...n，每个队列分别赋予不同的优先级，分配不同的时间片长度；新进程先进入队列 1 的末尾，按 FCFS 原则，执行队列 1 的时间片；若未能执行完进程，则转入队列 2 的末尾，如此重复。

1.6 死锁问题

当一个进程在等待永远不可能发生的事件时，就会产生死锁，若系统中有多个进程处于死锁状态，就会造成系统死锁。

死锁产生的四个必要条件：资源互斥、每个进程占有资源并等待其他资源、系统不能剥夺进程资源、进程资源图是一个环路。

死锁产生后，解决措施是打破四大条件，有下列方法：

死锁预防：采用某种策略限制并发进程对于资源的请求，破坏死锁产生的四个条件之一，使系统任何时刻都不满足死锁的条件。

死锁避免：一般采用银行家算法来避免，银行家算法，就是提前计算出一条不会死锁的资源分配方法，才分配资源，否则不分配资源，相当于借贷，考虑对方还得起才借钱，提前考虑好以后，就可以避免死锁。

死锁检测：允许死锁产生，但系统定时运行一个检测死锁的程序，若检测到系统中发生死锁，则设法加以解除。

死锁解除：即死锁发生后的解除方法，如强制剥夺资源，撤销进程等。

死锁资源计算：系统内有 n 个进程，每个进程都需要 R 个资源，那么其发生死锁的最大资源数为 $n*(R-1)$ 。其不发生死锁的最小资源数为 $n*(R-1)+1$ 。

1.7 线程

传统的进程有两个属性：可拥有资源的独立单位；可独立调度和分配的基本单位。

引入线程后，线程是独立调度的最小单位，进程是拥有资源的最小单位，线程可以共享进程的公共数据、全局变量、代码、文件等资源，但不能共享线程独有的资源，如线程的栈指针等标识数据。

2 存储管理

存储器的结构：寄存器——高速缓存 Cache——主存——外存。

地址重定位：将逻辑地址转换为实际主存物理地址的过程，分为静态重定位（在程序装入主存时就完成了转换）、动态重定位（边运行边转换）。

2.1 分区存储管理

所谓分区存储组织，就是整存，将某进程运行所需的内存整体一起分配给它，然后再执行。有三种分区方式：

固定分区：静态分区方法，将主存分为若干个固定的分区，将要运行的作业装配进去，由于分区固定，大小和作业需要的大小不同，会产生内部碎片。

可变分区：动态分区方法，主存空间的分区是在作业转入时划分，正好划分为作业需要的大小，这样就不存在内部碎片，但容易将整片主存空间切割成许多块，会产生外部碎片。可变分区的算法如下：

系统分配内存的算法有很多，如下图所示，根据分配前的内存情况，还需要分配 9K 空间，对不同算法的结果介绍如下：

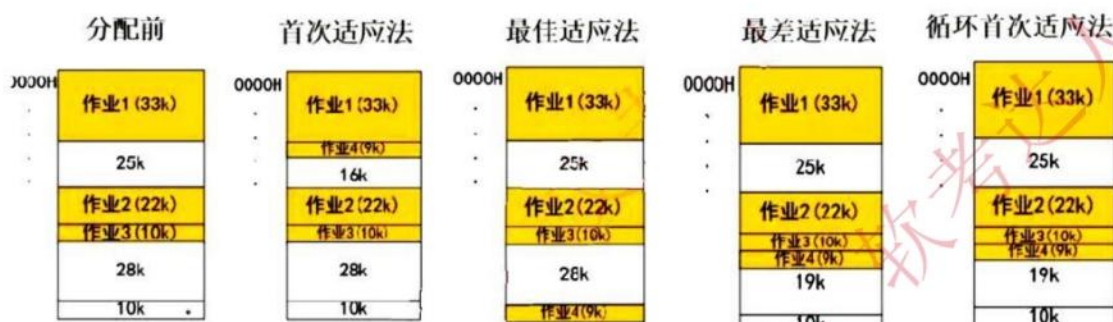
首次适应法：按内存地址顺序从头查找，找到第一个 $\geq 9K$ 空间的空闲块，即切割 9K 空间分配给进程。

最佳适应法：将内存中所有空闲内存块按从小到大排序，找到第一个 $\geq 9K$ 空间的空闲块，切割分配，这个将会找到与 9K 空间大小最相近的空闲块。

最差适应法：和最佳适应法相反，将内存中空闲块空间最大的，**切割 9K 空间**分配给进程，这是为了**预防系统中产生过多的细小空闲块**。

循环首次适应法：按内存地址顺序查找，找到第一个 $\geq 9K$ 空间的空闲块，而后若还需分配，则找下一个，**不用每次都从头查找**，这是与首次适应法不同的地方，下图中**结果有误**，应该和首次适应法结果一样。

某计算机系统的内存大小为128k，采用可变分区分配方式进行内存分配，当前系统的内存分块情况如下图所示，现有作业4申请内存9k，几种不同的存储分配算法在分配中，会产生什么样的结果呢？



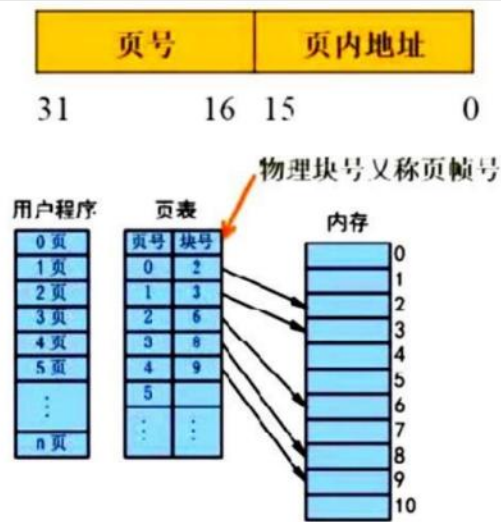
可重定位分区：可以解决碎片问题，移动所有已经分配好的区域，使其成为一个连续的区域，这样其他外部细小的分区碎片可以合并为大的分区，满足作业要求。只在外部作业请求空间得不到满足时进行。

2.2 页式存储管理

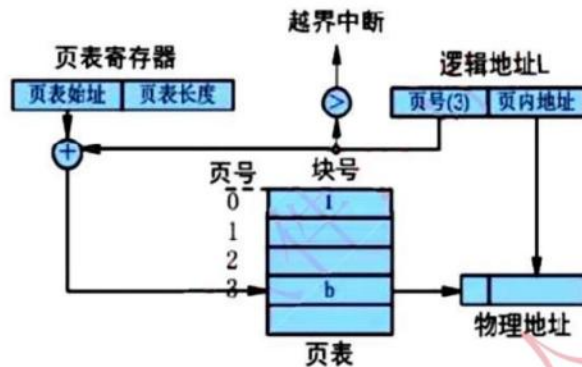
如果采用分区存储，都是整存，会出现一个问题，即当**进程运行所需的内存大于系统内存时**，就无法将整个进程一起调入内存，因此无法运行，若要解决此问题，就要采用段页式存储组织，**页式存储是基于可变分区而提出的**。

先介绍页式存储组织，将进程空间分为一个个页，假设每个页大小为 **4K**，同样的将系统的物理空间也分为一个个 **4K** 大小的物理块，这样，每次将需要运行的逻辑页装入物理块中，运行完再装入其他需要运行的页，这样就可以分批次运行完进程，而无需将整块逻辑空间全部装入物理内存中，解决了空间极大的进程的运行问题。

如下图所示，逻辑页分为页号和页内地址，页内地址就是物理偏移地址，而页号与物理块号并非按序对应的，需要查询页表，才能得知页号对应的物理块号，再用物理块号加上偏移地址才得出了真正运行时的物理地址。



高级程序语言使用逻辑地址；
运行状态，内存中使用物理地址。



优点：利用率高，碎片小，分配及管理简单。

缺点：增加了系统开销，可能产生抖动现象

地址表示和转换

地址组成：页地址+页内偏移地址；（页地址在高位，页内偏移地址在低位）

物理地址：物理块号+页内偏移地址；

逻辑地址：页号+页内偏移地址；

物理地址和逻辑地址的页内偏移地址是一样的，只要求出页号和物理块号之间的对应关系，首先要求出页号的位数，得出页号，再去页表里查询其对应的物理块号，使用此物理块号和页内偏移地址组合，就能得到物理地址。

详细转换流程见文老师视频课程讲解，有详细例题解析。

页面置换算法

有时候，进程空间分为 100 个页面，而系统内存只有 10 个物理块，无法全部满足分配，就需要将马上要执行的页面先分配进去，而后根据算法进行淘汰，使 100 个页面能够按执行顺序调入物理块中执行完。

缺页表示需要执行的页不在内存物理块中，需要从外部调入内存，会增加执行时间，因此，缺页数越多，系统效率越低。页面置换算法如下：

最优算法：OPT，理论上的算法，无法实现，是在进程执行完后进行的最佳效率计算，用来让其他算法比较差距。原理是选择未来最长时间不被访问的页面置换，这样可以保证未来执行的都是马上要访问的。

先进先出算法：FIFO，先调入内存的页先被置换淘汰，会产生**抖动现象**，即分配的页数越多，缺页率可能越多（即效率越低）。

最近最少使用：LRU，在最近的过去，进程执行过程中，过去最少使用的页面被置换淘汰，根据局部性原理，这种方式效率高，且不会产生抖动现象，使用大量计数器，但是没有 LFU 多。

淘汰原则：优先淘汰最近未访问的，而后淘汰最近未被修改的页面。

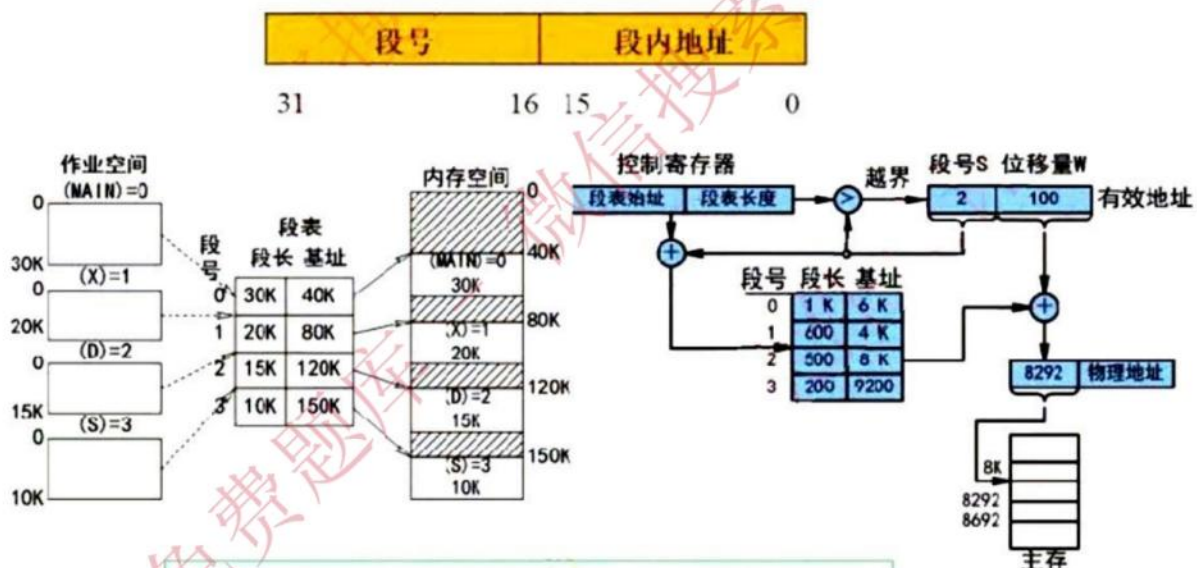
快表

是一块小容量的相联存储器，由快速存储器组成，按内容访问，速度快，并且可以从硬件上保证按内容并行查找，一般用来存放当前访问最频繁的少数活动页面的页号。

快表是将页表存于 Cache 中；慢表示将页表存于内存上。慢表需要访问两次内存才能取出页，而快表是访问一次 Cache 和一次内存，因此更快。

2.3 段式存储管理

将进程空间分为一个个段，每段也有段号和段内地址，与页式存储不同的是，每段物理大小不同，分段是根据逻辑整体分段的，因此，段表也与页表的内容不同，页表中直接是逻辑页号对应物理块号，而下图所示，段表有段长和基址两个属性，才能确定一个逻辑段在物理段中的位置。



优点：多道程序共享内存，各段程序修改互不影响。

缺点：内存利用率低，内存碎片浪费大。

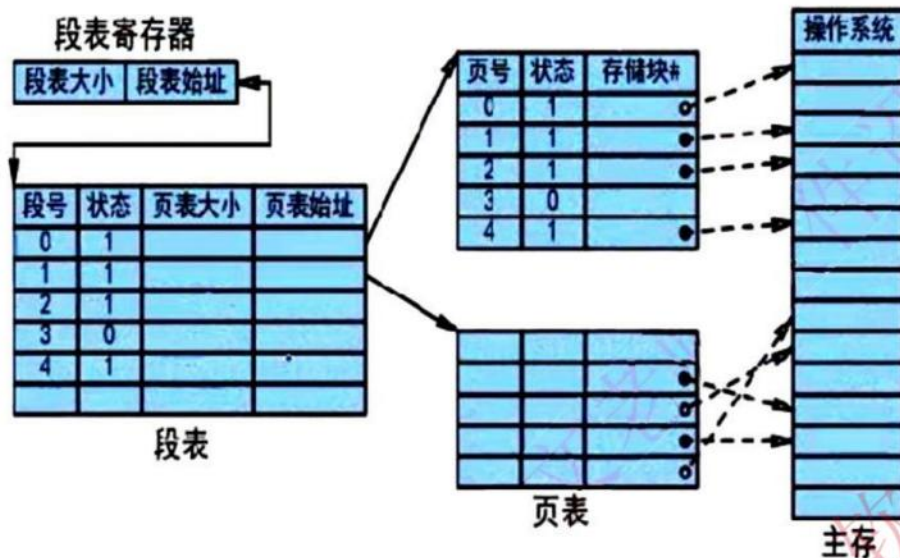
综上：分页是根据物理空间划分，每页大小相同；分段是根据逻辑空间划分，每段是一个完整的功能，便于共享，但是大小不同。

地址表示

(段号，段内偏移)：其中段内偏移不能超过该段号对应的段长，否则越界错误，而此地址对应的真正内存地址应该是：段号对应的基地址+段内偏移。

2.4 段页式存储管理

对进程空间先分段，后分页，具体原理图和优缺点如下：



优点：空间浪费小、存储共享容易、存储保护容易、能动态链接。

缺点：由于管理软件的增加，复杂性和开销也随之增加，需要的硬件以及占用的内容也有所增加，使得执行速度大大下降。

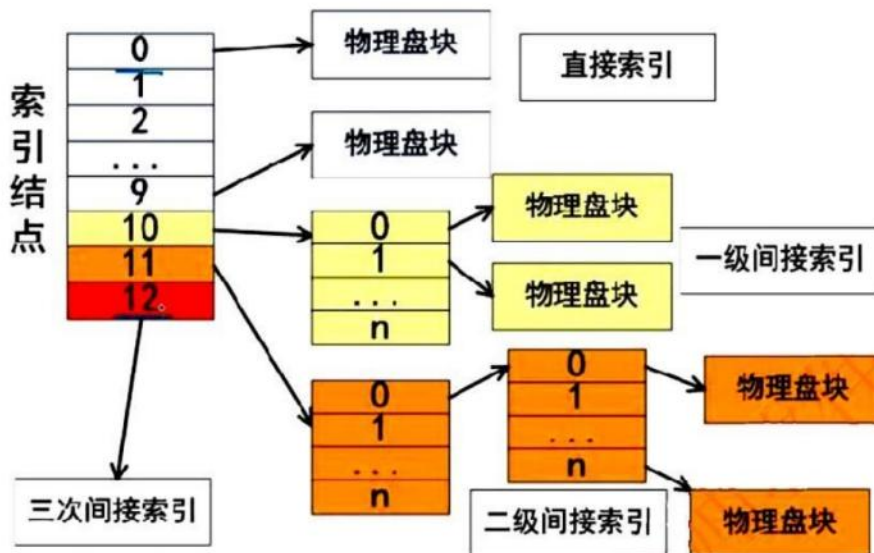
3 文件管理

3.1 索引文件结构

如下图所示，系统中有 13 个索引节点，0-9 为直接索引，即每个索引节点存放的是内容，假设每个物理盘大小为 4KB，共可存 $4KB \times 10 = 40KB$ 数据；

10 号索引节点为一级间接索引节点，大小为 4KB，存放的并非直接数据，而是链接到直接物理盘块的地址，假设每个地址占 4B，则共有 $4KB/4B = 1024$ 个地址，对应 1024 个物理盘，可存 $1024 \times 4KB = 4096KB$ 数据。

二级索引节点类似，直接盘存放一级地址，一级地址再存放物理盘快地址，而后链接到存放数据的物理盘块，容量又扩大了一个数量级，为 $1024 \times 1024 \times 4KB$ 数据。



3.2 文件和树形目录结构

文件安全分为四级，从高到低分别为：系统安全、用户安全、目录安全、文件安全。

相对路径：是从当前路径开始的路径。

绝对路径：是从根目录开始的路径。

全文件名=绝对路径+文件名。要注意，绝对路径和相对路径是不加最后的文件名的，只是单纯的路径序列。

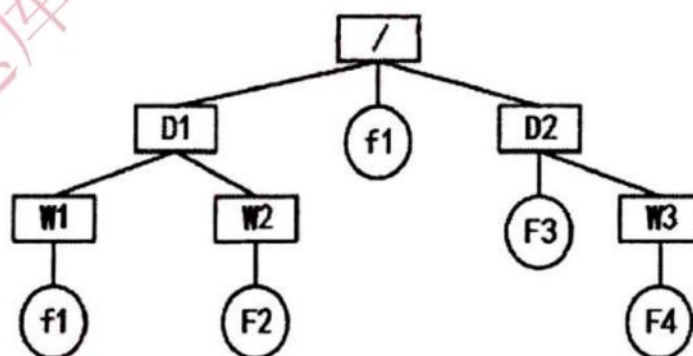
树形结构主要是区分相对路径和绝对路径，如下图所示：

◆文件属性

- R 只读文件属性
- A 存档属性
- S 系统文件
- H 隐藏文件

◆文件名的组成

- 驱动器号
- 路径
- 主文件名
- 扩展名



3.3 空闲存储空间的管理

空闲区表法：将所有空闲空间整合成一张表，即空闲文件目录。

空闲链表法：将所有空闲空间链接成一个链表，根据需要分配。

成组链接法：既分组，每组内又链接成链表，是上述两种方法的综合。

位示图法：重要，对每个物理空间用一位标识，为 1 则使用，为 0 则空闲，形成一张位示图。

4 设备管理

设备的分类方式：

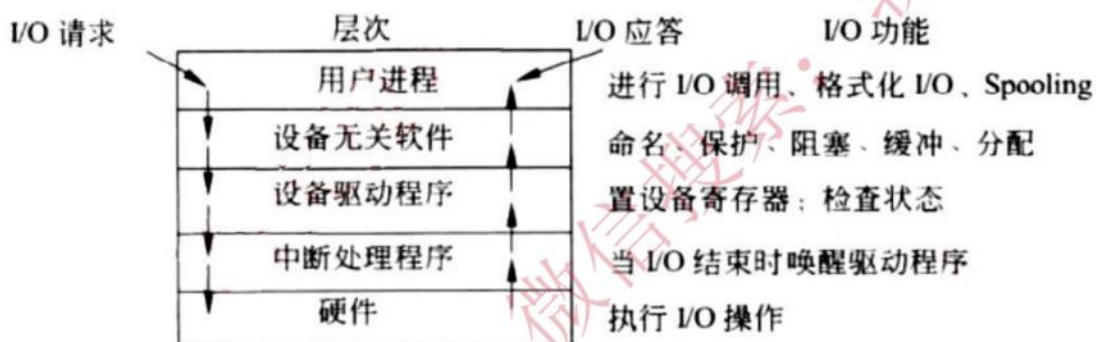
按数据组织分类：块设备、字符设备。

资源分配角度分类：独占设备、共享设备和虚拟设备。

数据传输速率分类：低速设备、中速设备、高速设备。

4.1 I/O 软件

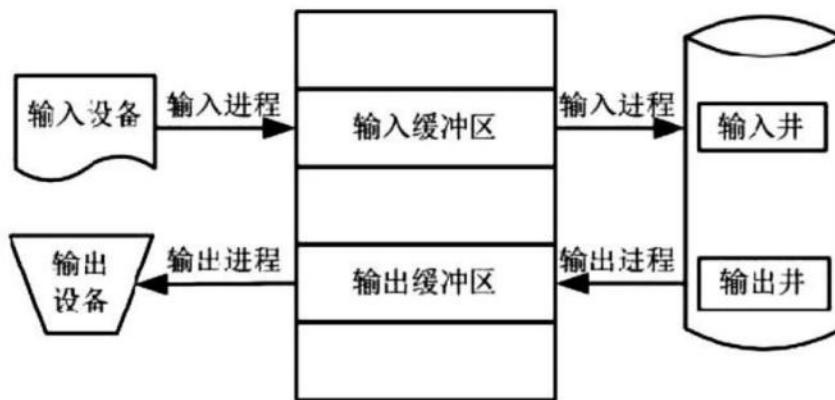
I/O 设备管理软件的所有层次及每一层功能如下图：



4.2 虚设备和 SPOOLING 技术

一台实际的物理设备，例如打印机，在同一时间只能由一个进程使用，其他进程只能等待，且不知道什么时候打印机空闲，此时，极大的浪费了外设的工作效率。

引入 SPOOLING 技术，就是在外设上建立两个数据缓冲区，分别称为输入井和输出井，这样，无论多少进程，都可以共用这一台打印机，只需要将打印命令发出，数据就会排队存储在缓冲区中，打印机会自动按顺序打印，实现了物理外设的共享，使得每个进程都感觉在使用一个打印机，这就是物理设备的虚拟化。如下图所示：



4.3 磁盘调度

磁盘结构和参数

磁盘有正反两个盘面，每个盘面有多个同心圆，每个同心圆是一个磁道，每个同心圆又被划分为多个扇区，数据就被存放在一个个扇区中。

磁头首先要寻找到对应的磁道，然后等待磁盘进行周期旋转，旋转指定的扇区，才能读取到对应的数据，因此，会产生寻道时间和等待时间。公式为：

存取时间=寻道时间+等待时间（平均定位时间+转动延迟）。

注意：寻道时间是指磁头移动到磁道所需的时间；等待时间为等待读写的扇区转到磁头下方所用的时间。

磁盘调度算法

之前已经说过，磁盘数据的读取时间分为寻道时间+旋转时间，也即先找到对应的磁道，而后再旋转到对应的扇区才能读取数据，其中寻道时间耗时最长，需要重点调度，有如下调度算法：

先来先服务 FCFS：根据进程请求访问磁盘的先后顺序进行调度。

最短寻道时间优先 SSTF：请求访问的磁道与当前磁道最近的进程优先调度，使得每次的寻道时间最短。会产生“饥饿”现象，即远处进程可能永远无法访问。

扫描算法 SCAN：又称“电梯算法”，磁头在磁盘上双向移动，其会选择离磁头当前所在磁道最近的请求访问的磁道，并且与磁头移动方向一致，磁头永远都是从里向外或者从外向里一直移动完才掉头，与电梯类似。

单向扫描调度算法 CSCAN：与 SCAN 不同的是，其只做单向移动，即只能从里向外或者从外向里。

磁盘冗余阵列技术

RAID 即磁盘冗余阵列技术，RAID0 没有提供冗余和错误修复技术；

RAID1 在成对的独立磁盘上产生互为备份的数据，可提高读取性能；

RAID2 将数据条块化的分布于不同硬盘上，并使用海明码校验；

RAID3 使用奇偶校验，并用单块磁盘存储奇偶校验信息；

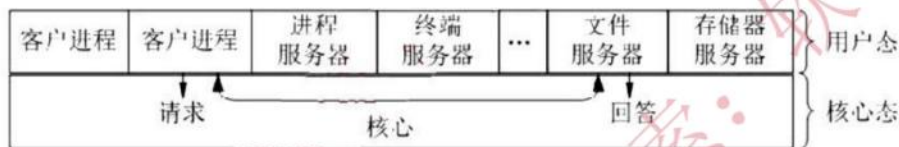
RAID5 在所有磁盘上交叉的存储数据及奇偶校验信息（所有校验信息存储总量为一个磁盘容量），读/写指针可同时操作；

RAID0+1（是两个 RAID0，若一个磁盘损坏，则当前 RAID0 无法工作，即有一半的磁盘无法工作），

RAID1+0（是两个 RAID1，不允许同一组中的两个磁盘同时损坏）与 RAID1 原理类似，磁盘利用率都只有 50%。

5 微内核操作系统

微内核，顾名思义，就是尽可能的将内核做的很小，只将最为核心必要的东西放入内核中，其他能独立的东西都放入用户进程中，这样，系统就被分为了用户态和内核态，如下图所示：



	实质	优点	缺点
单体内核	将图形、设备驱动及文件系统等功能全部在内核中实现，运行在内核状态和同一地址空间。	减少进程间通信和状态切换的系统开销，获得较高的运行效率。	内核庞大，占用资源较多且不易剪裁。 系统的稳定性和安全性不好。
微内核	只实现基本功能，将图形系统、文件系统、设备驱动及通信功能放在内核之外。	内核精练，便于剪裁和移植。 系统服务程序运行在用户地址空间，系统的可靠性、稳定性和安全性较高。 可用于分布式系统	用户状态和内核状态需要频繁切换，从而导致系统效率不如单体内核。

单核和微内核的优缺点上面也介绍了，单核方便进程切换，但是内核庞大，稳定性弱；微内核稳定性高，便于裁剪，但是进程间的切换消耗资源，效率低。

三 数据库技术基础

数据库系统 DBS 的组成：数据库、硬件、软件、人员。

数据库管理系统 DBMS 的功能：数据定义、数据库操作、数据库运行管理、数据的存储管理、数据库的建立和维护等。

DBMS 的分类：关系数据库系统 RDBS、面向对象的数据库系统 OODBs、对象关系数据库系统 ORDBS。

数据库系统的体系结构：集中式数据库系统（所有东西集中在 DBMS 电脑上）、客户端/服务器体系结构（客户端负责请求和数据表示，服务器负责数据库服务）、并行数据库系统（多个物理上在一起的 CPU）、分布式数据库系统（物理上分布在不同地方的计算机）。

1 三级模式—两级映像

内模式：管理如何存储物理的**数据**，对数据的存储方式、优化、存放等。

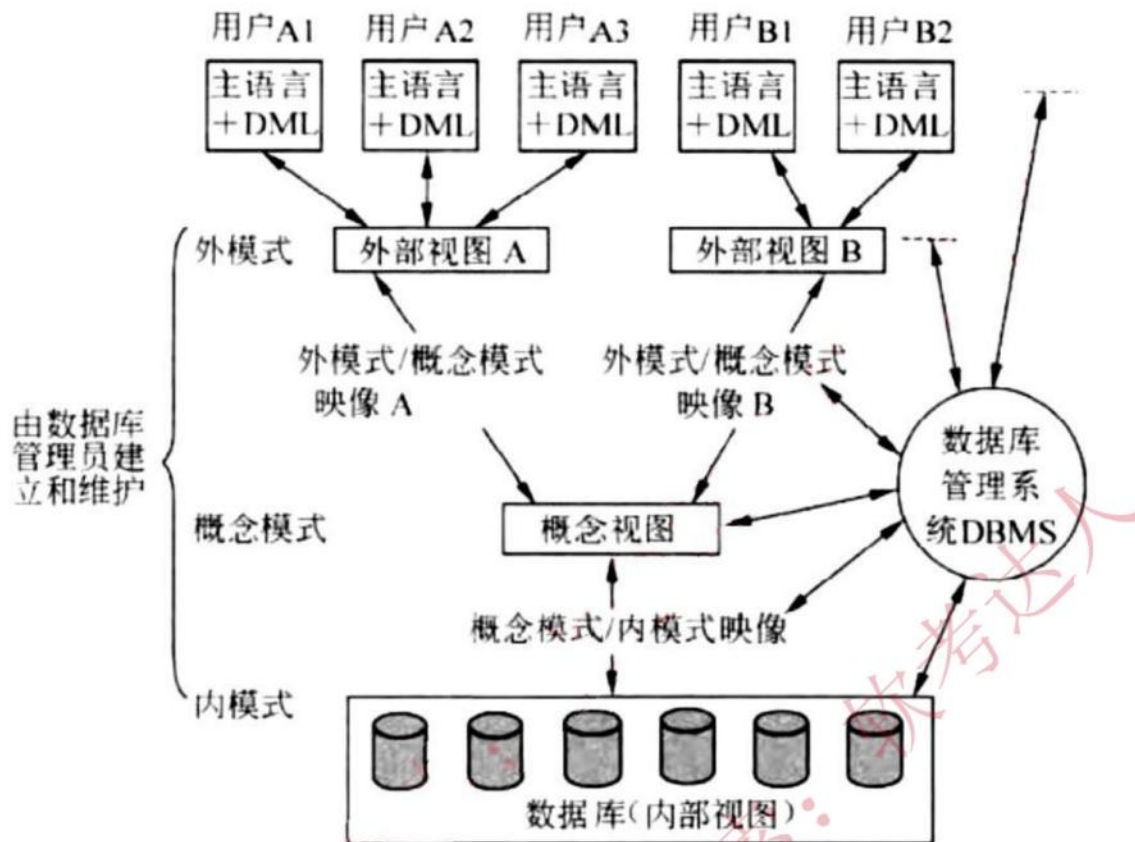
模式：又称为概念模式，就是我们通常使用的**表**这个级别，根据应用、需求将物理数据划分成一张张表。

外模式：对应数据库中的**视图**这个级别，将表进行一定的处理后再提供给用户使用，例如，将用户表中的用户名和密码组成视图提供给登录模块使用，而用户表中的其他列则不对该模块开放，增加了安全性。

外模式—模式映像：是表和视图之间的映射，存在于概念级和外部级之间，若表中数据发生了修改，只需要修改此映射，而无需修改应用程序。

模式—内模式映像：是表和数据的物理存储之间的映射，存在于概念级和内部级之间，若修改了数据存储方式，只需要修改此映射，而不需要去修改应用程序。

以上的数据库系统实际上是一个分层次的设计，从底至上称为物理级数据库（实际为一个数据库文件）、概念级数据库、用户级数据库，各层情况如下：



2 数据库的设计

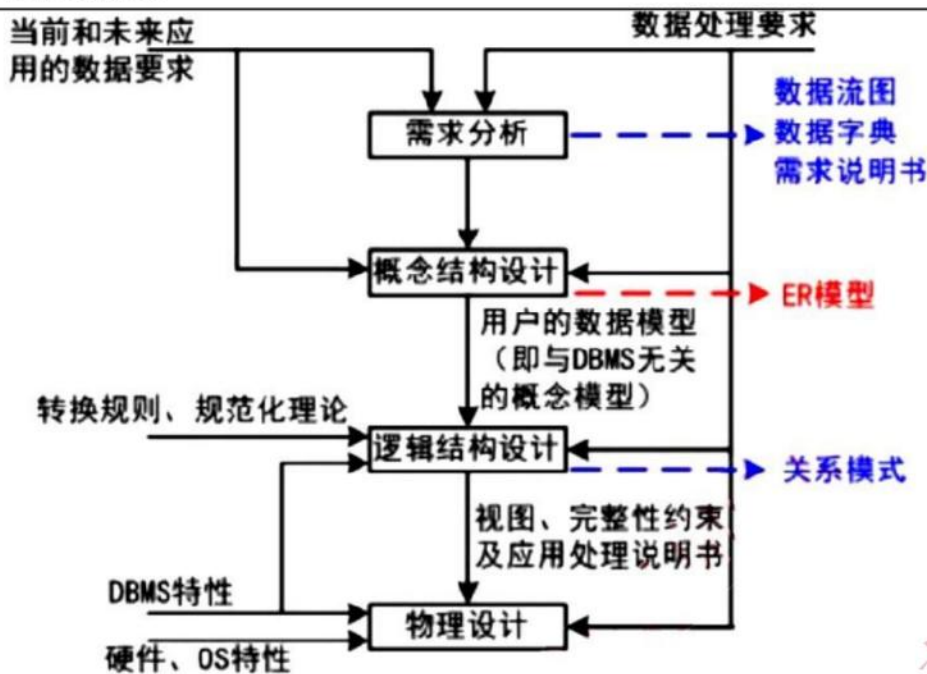
需求分析：即分析数据存储的要求，产出物有数据流图、数据字典、需求说明书。

概念结构设计：就是设计 E-R 图，也即实体-联系图，与物理实现无关，就是说明有哪些实体，实体有哪些属性。

逻辑结构设计：将 E-R 图，转换成关系模式，也即转换成实际的表和表中的列属性，这里要考虑很多规范化的东西。

物理设计：根据生成的表等概念，生成物理数据库。

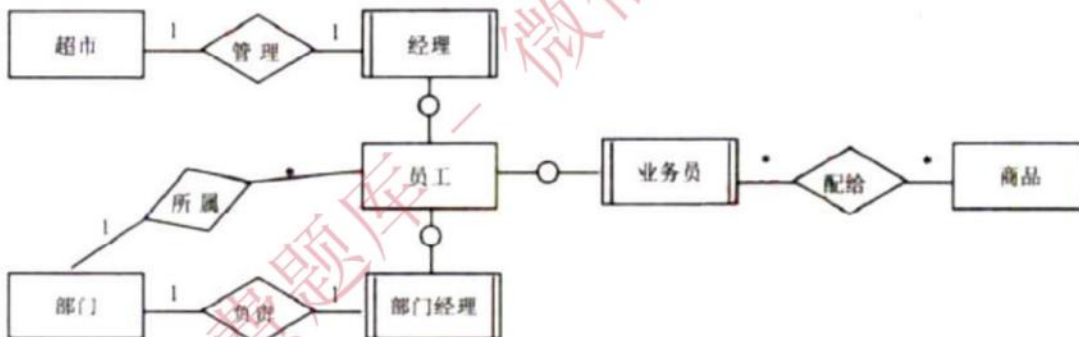
具体各个设计阶段的产出物、要求等如下所示：



3 E-R 模型

数据模型的三要素：数据结构、数据操作、数据的约束条件。

在 E-R 模型中，使用椭圆表示属性（一般没有）、长方形表示实体、菱形表示联系，联系的两端要填写联系类型，示例如下图：



联系类型：一对一 1:1、一对多 1:N、多对多 M:N。

属性分类：简单属性和复合属性（属性是否可以分割）、单值属性和多值属性（属性有多个取值）、NULL 属性（无意义）、派生属性（可由其他属性生成）。

那么 **E-R 模型如何转换为关系模型** 呢（实际就是转换为多少张表）？

每个实体都对应一个关系模式；

联系分为三种：**1:1** 联系中，联系可以放到任意的两端实体中，作为一个属性（**要保证 1:1 的两端关联**）；**1:N** 的联系中，联系可以单独作为一个关系模式，也可以在 **N** 端中加入 **1** 端实体的主键；**M:N** 的联系中，联系必须作为一个单独的关系模式，其主键是 **M** 和 **N** 端的联合主键。

以上，明确了有多少关系模式，就知道有多少张表，同时，表中的属性也确定了，注意联系是作为表还是属性，若是属性又是哪张表的属性即可。

4 关系代数运算

关系模式在代数运算时可以理解为数据库中的表，两个概念通用。

并：结果是两张表中所有记录数合并，相同记录只显示一次。

交：结果是两张表中相同的记录。

差： $S1-S2$ ，结果是 $S1$ 表中有而 $S2$ 表中没有的那些记录。

设有 $S1$ 和 $S2$ 关系如下图，其并交差结果如下图：

关系S1			关系S2		
Sno	Sname	Sdept	Sno	Sname	Sdept
No0001	Mary	IS	No0001	Mary	IS
No0003	Candy	IS	No0008	Katter	IS
No0004	Jam	IS	No0021	Tom	IS

$S1 \cap S2$ (交)			$S1 \cup S2$ (并)		
Sno	Sname	Sdept	Sno	Sname	Sdept
No0001	Mary	IS	No0001	Mary	IS
			No0003	Candy	IS
			No0004	Jam	IS
			No0008	Katter	IS
			No0021	Tom	IS

$S1 - S2$ (差)		
Sno	Sname	Sdept
No0003	Candy	IS
No0004	Jam	IS

笛卡尔积： $S1 \times S2$ ，产生的结果包括 $S1$ 和 $S2$ 的所有属性列，并且 $S1$ 中每条记录依次和 $S2$ 中所有记录组合成一条记录，最终属性列为 $S1+S2$ 属性列，记录数为 $S1 \times S2$ 记录数。

投影：实际是按条件选择某关系模式中的某列，列也可以用数字表示。

选择：实际是按条件选择某关系模式中的某条记录。

设有 $S1$ 和 $S2$ 关系如下图，其笛卡尔积、投影、选择结果如下图：

关系S1		
Sno	Sname	Sdept
No0001	Mary	IS
No0003	Candy	IS
No0004	Jam	IS

关系S2		
Sno	Sname	Sdept
No0001	Mary	IS
No0008	Katter	IS
No0021	Tom	IS

S1 × S2 (笛卡尔积)					
Sno	Sname	Sdept	Sno	Sname	Sdept
No0001	Mary	IS	No0001	Mary	IS
No0001	Mary	IS	No0008	Katter	IS
No0001	Mary	IS	No0021	Tom	IS
No0003	Candy	IS	No0001	Mary	IS
No0003	Candy	IS	No0008	Katter	IS
No0003	Candy	IS	No0021	Tom	IS
No0004	Jam	IS	No0001	Mary	IS
No0004	Jam	IS	No0008	Katter	IS
No0004	Jam	IS	No0021	Tom	IS

(投影)	
Sno	Sname
No0001	Mary
No0003	Candy
No0004	Jam

(选择)		
Sno	Sname	Sdept
No0003	Candy	IS

自然连接：结果显示全部的属性列，但是相同属性列只显示一次，显示两个关系模式中属性相同且值相同的记录。自然联接结果如下：

关系S1		
Sno	Sname	Sdept
No0001	Mary	IS
No0003	Candy	IS
No0004	Jam	IS

关系S2	
Sno	Age
No0001	23
No0008	21
No0021	22

S1▷◁S2			
Sno	Sname	Sdept	Age
No0001	Mary	IS	23

效率问题：关系代数运算的效率，归根结底是看参与运算的两张表格的属性列数和记录数，属性列数和记录数越少，参与运算的次数自然越少，效率就越高。因此，效率高的运算一般都是在两张表格参与运算之前就将条件判断完。如：

$\pi_{1,2,3,8} (\sigma_{2='大数据' \wedge 1=5 \wedge 3=6 \wedge 8='开发平台'} (R \times S))$ 和

$\pi_{1,2,3,8} (\sigma_{1=5 \wedge 3=6} (\sigma_{2='大数据'} (R) \times \sigma_{4='开发平台'} (S)))$ 。

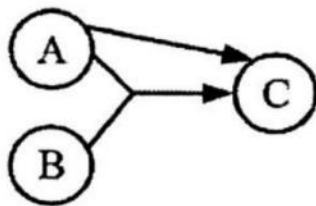
后者效率比前者效率高很多。

5 关系数据库的规范化

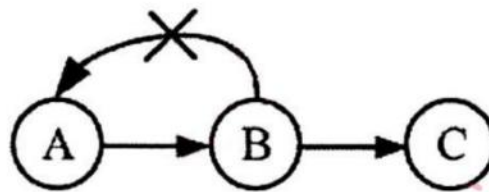
5.1 函数依赖

给定一个 X ，能唯一确定一个 Y ，就称 X 确定 Y ，或者说 Y 依赖于 X ，例如 $Y=X*X$ 函数。

函数依赖又可扩展以下两种规则：



部分函数依赖



传递函数依赖

部分函数依赖：A 可确定 C，(A,B)也可确定 C，(A,B)中的一部分（即 A）可以确定 C，称为部分函数依赖。

传递函数依赖：当 A 和 B 不等价时，A 可确定 B，B 可确定 C，则 A 可确定 C，是传递函数依赖；若 A 和 B 等价，则不存在传递，直接就可确定 C。

Armstrong 公理系统

设 U 是关系模式 R 的属性集， F 是 R 上成立的只涉及 U 中属性的函数依赖集。函数依赖的推理规则有：

自反律：若属性集 Y 包含于属性集 X ，属性集 X 包含于 U ，则 $X \rightarrow Y$ 在 R 上成立。（此处 $X \rightarrow Y$ 是平凡函数依赖）

增广律：若 $X \rightarrow Y$ 在 R 上成立，且属性集 Z 包含于属性集 U ，则 $XZ \rightarrow YZ$ 在 R 上成立。

传递律：若 $X \rightarrow Y$ 和 $Y \rightarrow Z$ 在 R 上成立，则 $X \rightarrow Z$ 在 R 上成立。

合并规则：若 $X \rightarrow Y$ ， $X \rightarrow Z$ 同时在 R 上成立，则 $X \rightarrow YZ$ 在 R 上也成立。

分解规则：若 $X \rightarrow W$ 在 R 上成立，且属性集 Z 包含于 W ，则 $X \rightarrow Z$ 在 R 上也成立。

伪传递规则：若 $X \rightarrow Y$ 在 R 上成立，且 $WY \rightarrow Z$ ，则 $XW \rightarrow Z$ 。

5.2 键和约束

超键：能唯一标识此表的属性的组合。

候选键：超键中去掉冗余的属性，剩余的属性就是候选键。

主键：任选一个候选键，即可作为主键。

外键：其他表中的主键。

候选键的求法：根据依赖集画出有向图，从入度为 0 的节点开始，找出图中一个节点或者一个节点组合，能够遍历完整个图，就是候选键。

主属性：候选键内的属性为主属性，其他属性为非主属性。

实体完整性约束：即主键约束，主键值不能为空，也不能重复。

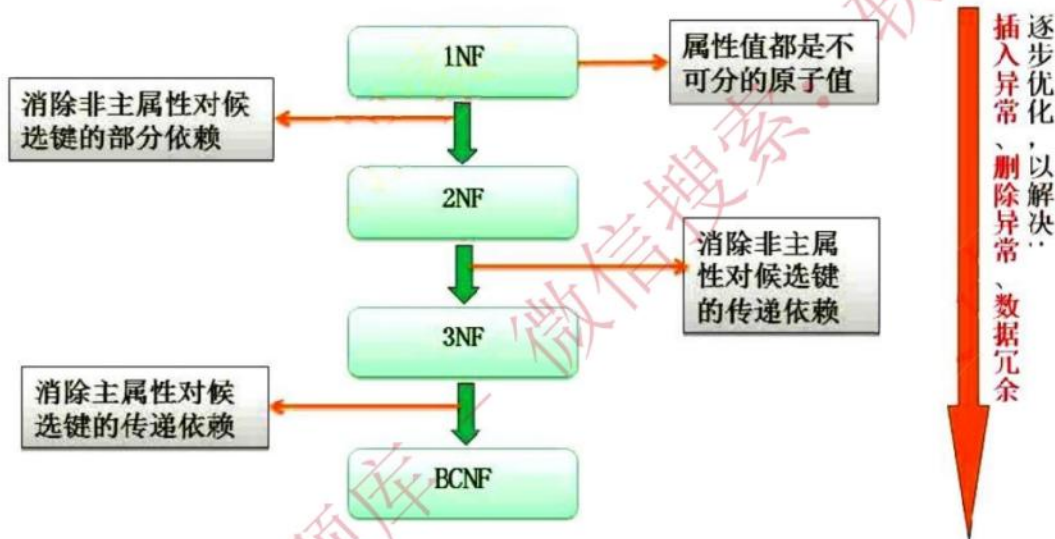
参照完整性约束：即外键约束，外键必须是其他表中已经存在的主键的值，或者为空。

用户自定义完整性约束：自定义表达式约束，如设定年龄属性的值必须在 0 到 150 之间。

触发器：通过写脚本来规定复杂的约束。本质属于用户自定义完整性约束。

5.3 范式

数据库中的范式总体概论如下图：



第一范式 1NF

关系中的每一个分量必须是一个不可分的数据项。通俗地说，第一范式就是表中不允许有小表的存在。比如，对于如下的员工表，就不属于第一范式：

员工编号	员工姓名	出生日期	薪资/月		所属部门
			基本工资/月	补贴/月	
1	王红	19900908	9000	1000	101
...	...	...	...	...	...

上表中，出现了属性薪资又被分为基本工资和补贴两个子属性，就好像表中有分割了一个小表，这就不属于第一范式。如果将基本工资和补贴合并，那么该表符合 1NF。

1NF 可能存在的问题：1NF 是最低一级的范式，范式程度不高，存在很多的问题。比如用一个单一的关系模式学生来描述学校的教务系统：学生(学号,学生姓名,系号,系主任姓名,课程号,成绩)

学号	学生姓名	所在系	系主任姓名	课程号	成绩
201102	张明	计算机系	章三	04	70
201103	王红	计算机系	章三	05	60
201103	王红	计算机系	章三	04	80
201103	王红	计算机系	章三	06	87
201104	李青	机械系	王五	09	79
...	...	...	...	...	...

这个表满足第一范式，但是存在如下问题：

数据冗余：一个系有很多的学生，同一个系的学生的系主任是相同的，所以系主任名会重复出现。

更新复杂：当一个系换了一个系主任后，对应的这个表必须修改与该系学生有关的每个元组。

插入异常：如果一个系刚成立，没有任何学生，那么这个无法把这个系的信息插入表中。

删除异常：如果一个系的学生都毕业了，那么在删除该系学生信息时，这个系的信息也丢了。

第二范式 2NF

如果关系 R 属于 1NF，且每一个非主属性完全函数依赖于任何一个候选码，则 R 属于 2NF。通俗地说，2NF 就是在 1NF 的基础上，表中的每一个非主属性不会依赖复合主键中的某一个列。

按照定义，上面的学生表就不满足 2NF，因为学号不能完全确定课程号和成绩(每个学生可以选多门课)。将学生表分解为：

学生(学号,学生姓名,系编号,系名,系主任)

选课(学号,课程号,成绩)。

每张表均属于 2NF。

第三范式 3NF

在满足 1NF 的基础上，表中不存在非主属性对码的传递依赖。

继续上面的实例，学生关系模式就不属于 3NF，因为学生无法直接决定系主任和系名，是由学号 $\rightarrow$ 系编号，再由系编号 $\rightarrow$ 系主任，系编号 $\rightarrow$ 系名，因此存在非主属性对主属性的传递依赖，将学生表进一步分解为：

学生(学号, 学生姓名, 系编号)

系(系编号, 系名, 系主任)

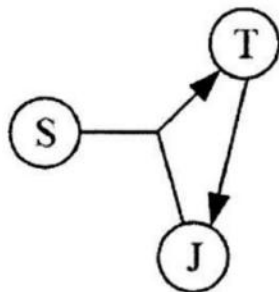
选课(学号, 课程号, 成绩)

每张表都属于 3NF。

BC 范式 BCNF

所谓 BCNF，是指在第三范式的基础上进一步消除主属性对于码的部分函数依赖和传递依赖。

通俗的来说，就是在每一种情况下，每一个依赖的左边决定因素都必然包含候选键，如下：



上图中，候选键有两种情况：组合键(S,T)或者(S,J)，依赖集为{S $\rightarrow$ T, T $\rightarrow$ J}，可知，STJ 三个属性都是主属性，因此其达到了 3NF（无非主属性），然而，第二种情况，即(S,J)为候选键的时候，对于依赖 T $\rightarrow$ J，T 在这种情况下不是候选键，即 T $\rightarrow$ J 的决定因素不包含任意候选码，因此上图不是 BCNF。

要使上图关系模式转换为 BCNF 也很简单，只需要将依赖 T $\rightarrow$ J 变为 TS $\rightarrow$ J 即可，这样其左边决定因素就包含了候选键之一 S。

5.4 模式分解

范式之间的转换一般都是通过拆分属性，即模式分解，将具有部分函数依赖和传递依赖的属性分离出来，来达到一步步优化，一般分为以下两种：

保持函数依赖分解

对于关系模式 R，有依赖集 F，若对 R 进行分解，分解出来的多个关系模式，保持原来的依赖集不变，则为保持函数依赖的分解。另外，注意要消除掉冗余依赖（如传递依赖）。

实例：设原关系模式 R(A,B,C)，依赖集 F(A $\rightarrow$ B, B $\rightarrow$ C, A $\rightarrow$ C)，将其分解为两个关系模式 R1(A,B)和 R2(B,C)，此时 R1 中保持依赖 A $\rightarrow$ B，R2 保持依赖 B $\rightarrow$ C，说明分解后的 R1 和 R2 是保持函数依赖的分解，因为 A $\rightarrow$ C 这个函数依赖实际是一个冗余依赖，可以由前两个依赖传递得到，因此不需要管。

保持函数依赖的判断（补充，第2点不强求）：

1、如果 F 上的每一个函数依赖都在其分解后的某一个关系上成立，则这个分解是保持依赖的（这是一个充分条件）。也即我们课堂上说的简单方法，看函数每个依赖的左右两边属性是否都在同一个分解的模式中。

2、如果上述判断失败，并不能断言分解不是保持依赖的，还要使用下面的通用方法来做进一步判断。

该方法的表述如下：

算法二：

对 F 上的每一个 $\alpha \rightarrow \beta$ 使用下面的过程：

result := α ;

while(result 发生变化)do

for each 分解后的 R_i

$t = (\text{result} \cap R_i) + R_i$

result = result $\cup$ t

以下面的例题纠错作为讲解：正确答案是无损分解，不保持函数依赖。

假设关系模式 $R(U, F)$, 属性集 $U = \{A, B, C\}$, 函数依赖集 $F = \{A \rightarrow B, B \rightarrow C\}$ 。若将其分解为 $p = \{R_1(U_1, F_1), R_2(U_2, F_2)\}$, 其中 $U_1 = \{A, B\}, U_2 = \{A, C\}$ 。那么，关系模式 R, R_1, R_2 分别达到了 (5); 分解 $D(6)$ 。

(5) A. 1NF、2NF、3NF B. 1NF、3NF、3NF C. 2NF、2NF、3NF D. 2NF、3NF、3NF

(6) A. 有损连接但保持函数依赖 B. 既无损连接又保持函数依赖
C. 有损连接且不保持函数依赖 D. 无损连接但不保持函数依赖

【答案】D B

首先，该分解， U_1 保持了依赖 $A \rightarrow B$ ，然而 $B \rightarrow C$ 没有保持，因此针对 $B \rightarrow C$ 需要用第2点算法来判断：

result = B , result $\cap U_1 = B$, $B^+ = BC$, $BC \cap U_1 = B$, result = $B \cup B = B$, result 没变，然后，result 再和 U_2 交是空，结束了，不保持函数依赖。

注意，这里 B^+ , + 的意思是代表由 B 能够推导出的其他所有属性的集合，这里， $B \rightarrow C$ ，因此 $B^+ = BC$ 。

无损分解

分解后的关系模式能够还原出原关系模式，就是无损分解，不能还原就是有损。

当分解为两个关系模式，可以通过以下定理判断是否无损分解：

定理：如果 R 的分解为 $p=\{R_1, R_2\}$ ， F 为 R 所满足的函数依赖集合，分解 p 具有无损连接性的充分必要条件是 $R_1 \cap R_2 \rightarrow (R_1 - R_2)$ 或者 $R_1 \cap R_2 \rightarrow (R_2 - R_1)$ 。

当分解为多个关系模式时，通过表格法求解：

思考题：

有关系模式：成绩（学号，姓名，课程号，课程名，分数）

函数依赖：学号 $\rightarrow$ 姓名，课程号 $\rightarrow$ 课程名，（学号，课程号） $\rightarrow$ 分数

若将其分解为：

成绩（学号，课程号，分数）

学生（学号，姓名）

课程（课程号，课程名）

请思考该分解是否为无损分解？

由于有：学号 $\rightarrow$ 姓名，所以：

成绩（学号，课程号，分数，姓名）

由于有：课程号 $\rightarrow$ 课程名，所以：

成绩（学号，课程号，分数，姓名，课程名）

由思考题可知，无损分解，要注意将候选键和其能决定的属性放在一个关系模式中，这样才能还原。也可以用表格法求解如下（需要依赖左右边的属性同时在一个关系模式中，才能补充）：

初始表如下：

	学号	姓名	课程号	课程名	分数
成绩	√	×	√	×	√
学生	√	√	×	×	×
课程	×	×	√	√	×

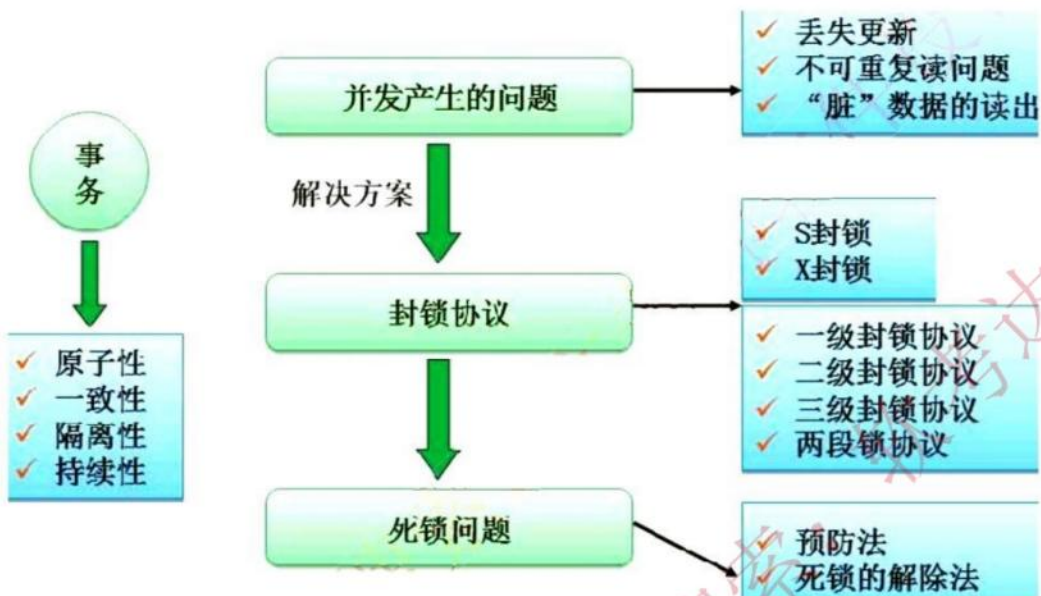
根据学号 $\rightarrow$ 姓名，对上表进行处理，将 $\times$ 改成符号 $\sqrt{}$ ；然后考虑课程号 $\rightarrow$ 课程名，将 $\times$ 改为 $\sqrt{}$ ，得下表：

	学号	姓名	课程号	课程名	分数
成绩	√	√	√	√	√
学生	√	√	×	×	×
课程	×	×	√	√	×

从上图中可以看出，第1行已全部为 $\sqrt{}$ ，因此本次 R 分解是无损联接分解。

注意：拆分成单属性集必然是有损分解，因为单属性不可能包含依赖左右两边属性，这个单属性已经无法再恢复。

5.5 并发控制基本概念图



5.6 事务管理

事务提交 commit，事务回滚 rollback。

事务：由一系列操作组成，这些操作，要么全做，要么全不做，拥有四种特性，详解如下：

（操作）**原子性**：要么全做，要么全不做。

（数据）**一致性**：事务发生后数据是一致的，例如银行转账，不会存在 A 账户转出，但是 B 账户没收到的情况。

（执行）**隔离性**：任一事务的更新操作直到其成功提交的整个过程对其他事务都是不可见的，不同事务之间是隔离的，互不干涉。

（改变）**持续性**：事务操作的结果是持续性的。

事务是并发控制的前提条件，并发控制就是控制不同的事务并发执行，提高系统效率，但是并发控制中存在下面三个问题：

丢失更新：事务 1 对数据 A 进行了修改并写回，事务 2 也对 A 进行了修改并写回，此时事务 2 写回的数据会覆盖事务 1 写回的数据，就丢失了事务 1 对 A 的更新。即对数据 A 的更新会被覆盖。

不可重复读：事务 2 读 A，而后事务 1 对数据 A 进行了修改并写回，此时若事务 2 再读 A，发现数据不对。即一个事务重复读 A 两次，会发现数据 A 有误。

读脏数据：事务 1 对数据 A 进行了修改后，事务 2 读数据 A，而后事务 1 回滚，数据 A 恢复了原来的值，那么事务 2 对数据 A 做的事是无效的，读到了脏数据。

◆ 丢失更新

T1	T2
①读A=10 ② ③A=A-5写回 ④	读A=10 A=A-8 写回

◆ 不可重复读

T1	T2
①读A=20 读B=30 求和=50 ② ③读A=70 读B=30 求和=100 (验算不对)	读A=20 A←A+50 写A=70

◆ 读“脏”数据

T1	T2
①读A=20 A←A+50 写回70 ② ③ROLLBACK A恢复为20	读A=70

5.7 封锁协议

X 锁是排它锁（写锁）。若事务 T 对数据对象 A 加上 X 锁，则只允许 T 读取和修改 A，其他事务都不能再对 A 加任何类型的锁，直到 T 释放 A 上的锁。

S 锁是共享锁（读锁）。若事务 T 对数据对象 A 加上 S 锁，则只允许 T 读取 A，但不能修改 A，其他事务只能再对 A 加 S 锁（也即能读不能修改），直到 T 释放 A 上的 S 锁。

共分为三级封锁协议，定义如下：

一级封锁协议：事务在修改数据 R 之前必须先对其加 X 锁，直到事务结束才释放。可解决丢失更新问题。

二级封锁协议：一级封锁协议的基础上加上事务 T 在读数据 R 之前必须先对其加 S 锁，读完后即可释放 S 锁。可解决丢失更新、读脏数据问题。

三级封锁协议：一级封锁协议加上事务 T 在读取数据 R 之前先对其加 S 锁，直到事务结束才释放。可解决丢失更新、读脏数据、数据重复读问题。

三级封锁协议的应用

丢失更新加锁（一级封锁协议）：

T1	T2
① 对A加写锁 ② ③ 读A=10 ④ A=A-5写回 ⑤ 释放对A的写锁 ⑥ ⑦ ⑧	对A加写锁 等待 等待 等待 读A= 5 A=A-8 写回 释放对A的写锁

读脏数据加锁（二级封锁协议）：

T1	T2
① 对A加写锁 ② 读A=20 ③ A←A+50 ④ 写回70 ⑤ ⑥ ROLLBACK ⑦ A恢复为20 ⑧	对A加读锁 等待 等待 读A=20 释放对A的读锁

不可重复读加锁（三级封锁协议）：

T1	T2
① 对A与B加S锁（读锁） 读A=20 读B=30 求和=50 ② ③ 读A=20 读B=30 求和=50 释放对A和B的读锁	对A加X锁（写锁） 注：由于A已加了读锁，所以等待 等待 等待 等待 等待 读A=20 A←A+50 写A=70 释放对A的写锁

两段锁协议

每个事务的执行可以分为两个阶段：生长阶段（加锁阶段）和衰退阶段（解锁阶段）。

加锁阶段：在该阶段可以进行加锁操作。在对任何数据进行读操作之前要申请并获得 S 锁，在进行写操作之前要申请并获得 X 锁。加锁不成功，则事务进入等待状态，直到加锁成功才继续执行。

解锁阶段：当事务释放了一个封锁以后，事务进入解锁阶段，在该阶段只能进行解锁操作不能再进行加锁操作。

两段封锁法可以这样来实现：事务开始后就处于加锁阶段，一直到执行 ROLLBACK 和 COMMIT 之前都是加锁阶段。ROLLBACK 和 COMMIT 使事务进入解锁阶段，即在 ROLLBACK 和 COMMIT 模块中 DBMS 释放所有封锁。

6 数据故障与备份

6.1 安全措施

措施	说明
用户标识和鉴定	最外层的安全保护措施，可以使用用户帐户、口令及随机数检验等方式
存取控制	对用户进行授权，包括操作类型（如查找、插入、删除、修改等动作）和数据对象（主要是数据范围）的权限。
密码存储和传输	对远程终端信息用密码传输
视图的保护	对视图进行授权
审计	使用一个专用文件或数据库，自动将用户对数据库的所有操作记录下来

6.2 数据故障

故障关系	故障原因	解决方法
事务本身的可预期故障	本身逻辑	在程序中预先设置Rollback语句
事务本身的不可预期故障	算术溢出、违反存储保护	由DBMS的恢复子系统通过日志，撤消事务对数据库的修改，回退到事务初始状态
系统故障	系统停止运转	通常使用检查点法
介质故障	外存被破坏	一般使用日志重做业务

6.3 数据备份

静态转储：即冷备份，指在转储期间不允许对数据库进行任何存取、修改操作；优点是非常快速的备份方法、容易归档（直接物理复制操作）；缺点是只能提供到某一时间点上的恢复，不能做其他工作，不能按表或按用户恢复。

动态转储：即热备份，在转储期间允许对数据库进行存取、修改操作，因此，转储和用户事务可并发执行；优点是可在表空间或数据库文件级备份，数据库仍可使用，可达到秒级恢复；缺点是不能出错，否则后果严重，若热备份不成功，所得结果几乎全部无效。

完全备份：备份所有数据。

差量备份：仅备份上一次完全备份之后变化的数据。

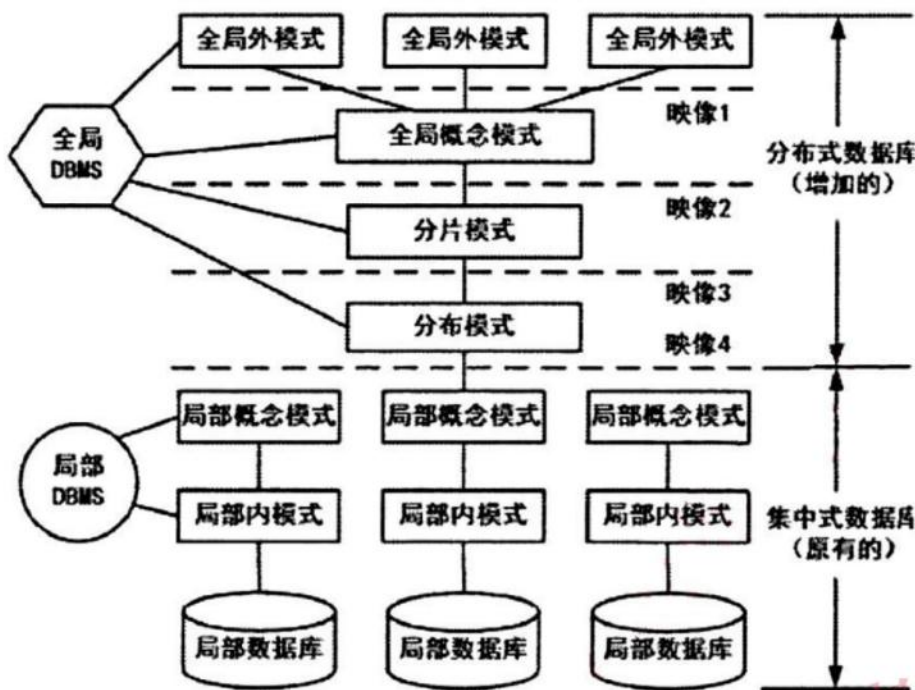
增量备份：备份上一次备份之后变化的数据。

日志文件：在事务处理过程中，DBMS 把事务开始、事务结束以及对数据库的插入、删除和修改的每一次操作写入日志文件。一旦发生故障，DBMS 的恢复子系统利用日志文件撤销事务对数据库的改变，回退到事务的初始状态。

备份毕竟是有时间节点的，不是实时的，例如：上一次备份到这次备份之间数据库出现了故障，则这期间的数据无法恢复，因此，引入日志文件，可以实时记录针对数据库的任何操作，保证数据库可以实时恢复。

7 分布式数据库

局部数据库位于不同的物理位置，使用一个全局 DBMS 将所有局部数据库联网管理，这就是分布式数据库。其体系结构如下图所示：



分片模式

水平分片：将表中水平的记录分别存放在不同的地方。

垂直分片：将表中的垂直的列值分别存放在不同的地方。

分布透明性

分片透明性：用户或应用程序不需要知道逻辑上访问的表具体是如何分块存储的。

位置透明性：应用程序不关心数据存储物理位置的改变。

逻辑透明性：用户或应用程序无需知道局部使用的是哪种数据模型。

复制透明性：用户或应用程序不关心复制的数据从何而来。

8 数据仓库与数据挖掘

数据仓库是一种特殊的数据库，也是按数据库形式存储数据的，但是目的不同：数据库经过长时间的运行，里面的数据会保存的越来越多，就会影响系统运行效率，对于某些程序而言，很久之前的数据并非必要的，因此，可以删除掉以减少数据，增加效率，考虑到删除这些数据比较可惜，因此，一般都将这些数据从数据库中提取出来保存到另外一个数据库中，称为数据仓库。

数据仓库四大特点

面向主题：按照一定的主题域进行组织的。

集成的：数据仓库中的数据是在对原有分散的数据库数据抽取、清理的基础上经过系统加工、汇总和整理得到的，必须消除源数据中的不一致性，以保证数据仓库内的信息是关于整个企业的一致性的全局信息。

相对稳定的：数据仓库的数据主要供企业决策分析之用，所涉及的数据操作主要是数据查询，一旦某个数据进入数据仓库以后，一般情况下将被长期保留，也就是数据仓库中一般有大量的查询操作，但修改和删除操作很少，通常只需要定期的加载、刷新。

反映历史变化：数据仓库中的数据通常包含历史信息，系统记录了企业从过去某一时点(如开始应用数据仓库的时点)到目前的各个阶段的信息，通过这些信息，可以对企业的发展历程和未来趋势做出定量分析和预测。

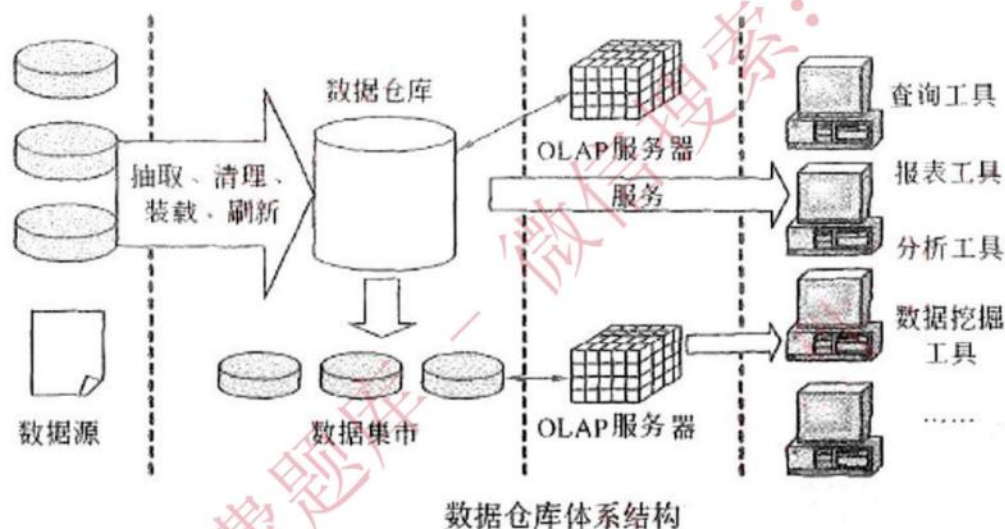
数据仓库的结构通常包含四个层次，如下图所示：

1.数据源：是数据仓库系统的基础，是整个系统的数据源泉。

2.数据的存储与管理：是整个数据仓库系统的核心。

3.OLAP（联机分析处理）服务器：对分析需要的数据进行有效集成，按多维模型组织，以便进行多角度、多层次的分析，并发现趋势。

4.前端工具：主要包括各种报表工具、查询工具、数据分析工具、数据挖掘工具以及各种基于数据仓库或数据集市的应用开发工具。



数据挖掘的分析方法

关联分析：关联分析主要用于发现不同事件之间的关联性，即一个事件发生的同时，另一个事件也经常发生。

序列分析：序列分析主要用于发现一定时间间隔内接连发生的事件，这些事件构成一个序列，发现的序列应该具有普遍意义。

分类分析：分类分析通过分析具有类别的样本特点，得到决定样本属于各种类别的规则或方法。分类分析时首先为每个记录赋予一个标记（一组具有不同特征的类别），即按标记分类记录，然后检查这些标定的记录，描述出这些记录的特征。

聚类分析：聚类分析是根据“物以类聚”的原理，将本身没有类别的样本聚集成不同的组，并且

对每个这样的组进行描述的过程。

商业智能 BI

BI 系统主要包括数据预处理、建立数据仓库、数据分析和数据展现四个主要阶段。

数据预处理是整合企业原始数据的第一步，它包括数据的抽取（Extraction）、转换（Transformation）和加载（Load）三个过程（ETL 过程）；

建立数据仓库则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（OLAP）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；

在海量数据和分析手段增多的情况下，数据展现则主要保障系统分析结果的可视化。

9 反规范化技术

由前面介绍可知，规范化操作可以防止插入异常、更新、删除异常和数据冗余，一般是通过模式分解，将表拆分，来达到这个目的。

但是表拆分后，解决了上述异常，却不利于查询，每次查询时，可能都要关联很多表，严重降低了查询效率，因此，有时候需要使用反规范化技术来提高查询效率。

技术手段包括：增加派生性冗余列，增加冗余列，重新组表，分割表。

主要就是增加冗余，提高查询效率，为规范化操作的逆操作。

10 大数据

特点：大量化、多样化、价值密度低、快速化。

大数据和传统数据的比较如下：

比较维度	传统数据	大数据
数据量	GB或TB级	PB级或以上
数据分析需求	现有数据的分析与检测	深度分析（关联分析、回归分析）
硬件平台	高端服务器	集群平台

要处理大数据，一般使用集成平台，称为大数据处理系统，其特征为：

高度可扩展性、高性能、高度容错、支持异构环境、较短的分析延迟、易用且开放的接口、较低成本、向下兼容性。

11 SQL 语言

SQL 语言中的**语法关键字**，不区分大小写：

创建表 **create table**;

指定主键 **primary key()**;

指定外键 **foreign key()**;

修改表 **alter table**;

删除表 **drop table**;

索引 **index**，视图 **view**;

数据库查询 **select...from...where**;

分组查询 **group by**，分组时要注意 **select** 后的列名要适应分组，**having** 为分组查询附加条件；

更名运算 **as**;

字符串匹配 **like**，**%**匹配多个字符串，**_**匹配任意一个字符串；

数据库插入 **insert into...values()**;

数据库删除 **delete from...where**;

数据库修改 **update...set...where**;

排序 **order by**，默认为升序，降序要加关键字 **DESC**。

授权 **grant...on...to**，允许其将权限再赋给另一用户 **with grant option**;

收回权限 **revoke...on...from**;

with check option 表示要检查 **where** 后的谓词条件。

DISTINCT：过滤重复的选项，只保留一条记录。

UNION：出现在两个 SQL 语句之间，将两个 SQL 语句的查询结果取或运算，即值存在于第一句或第二句都会被选出。

INTERSECT：对两个 SQL 语句的查询结果做与运算，即值同时存在于两个语句才被选出。

SQL 语法原理

SELECT 之后的为要查询显示的属性列名；**FROM** 后面是要查询的表名；**WHERE** 后面是查询条件；涉及到平均数、最大值、求和等运算，必须要分组，**group by** 后面是分组的属性列名，分组的条件使用 **Having** 关键字，后面跟条件。

在 SQL 语句中，条件判断时数字无需打引号，字符串要打单引号。

四 计算机网络

1 网络概述和模型

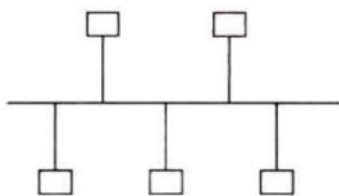
计算机网络是计算机技术与通信技术相结合的产物，它实现了远程通信、远程信息处理和资源共享。

计算机网络的功能：数据通信、资源共享、负载均衡、高可靠性。

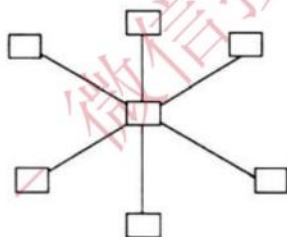
计算机网络按分布范围和拓扑结构划分如下图所示：

网络分类	缩写	分布距离	计算机分布范围	传输速率范围
局域网	LAN	10m 左右	房间	4Mbps~1Gbps
		100m 左右	楼宇	
		1000m 左右	校园	
城域网	MAN	10km	城市	50Kbps~100Mbps
广域网	WAN	100km 以上	国家或全球	9.6Kbps~45Mbps

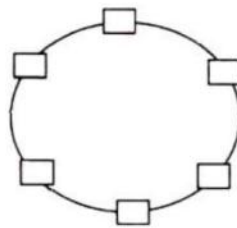
总线型（利用率低、干扰大、价格低）、星型（交换机形成的局域网、中央单元负荷大）、环型（流动方向固定、效率低扩充难）、树型（总线型的扩充、分级结构）、分布式（任意节点连接、管理难成本高）



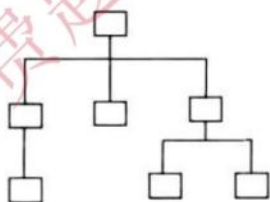
(a) 总线型



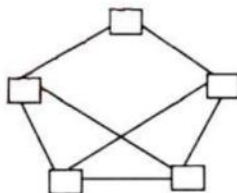
(b) 星型



(c) 环型



(d) 树型



(e) 分布式

一般来说，办公室局域网是星型拓扑结构，中间节点就是交换机，一旦交换机损坏，整个网络都瘫痪了，这就是星型结构。同理，由路由器连接起来的小型网络也是星型结构。

OSI/RM 七层模型

层	功能	单位	协议	设备
1. 物理层	是网络通信的数据传输介质，由连接不同结点的电缆与设备共同构成。主要功能是：利用传输介质为数据链路层提供物理连接，负责处理数据传输并监控数据出错率，以便数据流的透明传输。	比特	EIA/TIA RS-232、RS-449、V.35、RJ-45、FDDI	中继器、集线器
2. 数据链路层	在物理层提供的服务基础上，在通信的实体间建立数据链路连接，传输以“帧”为单位的数据包，并采用差错控制与流量控制方法，使有差错的物理线路变成无差错的数据链路。	帧	SDLC（同步数据链路控制）、HDLC（高级数据链路控制）、PPP（点对点协议）、STP（生成树协议）、帧中继等、IEEE802、ATM（异步传输）	交换机、网桥
3. 网络层	为数据在结点之间传输创建逻辑链路，通过路由选择算法为分组通过通信子网选择最适当的路径，以及实现拥塞控制、网络互联等功能。	IP分组	IP（网络互连）、IPX（互联网数据包交换协议）、ICMP（网际控制报文协议）、IGMP（网络组管理协议）、ARP（地址转换协议）、RARP	路由器
4. 传输层	向用户提供可靠的端到端（End-to-End）服务，处理数据包错误、数据包次序，以及其他一些关键传输问题。传输层向高层屏蔽了下层数据通信的细节	报文段	TCP（传输控制协议）、UDP（用户数据报协议）、SPX（序列分组交换协议）	网关
5. 会话层	负责维护两个结点之间的传输链接，以便确保点到点传输不中断，以及管理数据交换等功能。		RPC（远程过程调用）、SQL（结构化查询语言）、NFS（网络文件系统）	网关
6. 表示层	用于处理在两个通信系统中交换信息的表示方式，主要包括数据格式变换、数据加密与解密、数据压缩与恢复等功能。		JPEG、ASCII、GIF、MPEG、DES	网关
7. 应用层	为应用软件提供了很多服务，例如文件服务器、数据库服务、电子邮件与其他网络软件服务。	数据	Telnet、FTP、HTTP、SMTP、POP3、DNS、DHCP等	网关

以太网规范 IEEE 802.3 是重要的局域网协议，包括：

IEEE 802.3	标准以太网	10Mb/s	传输介质为细同轴电缆
IEEE 802.3u	快速以太网	100Mb/s	双绞线
IEEE 802.3z	千兆以太网	1000Mb/s	光纤或双绞线
IEEE 802.3ae	万兆以太网	10Gb/s	光纤

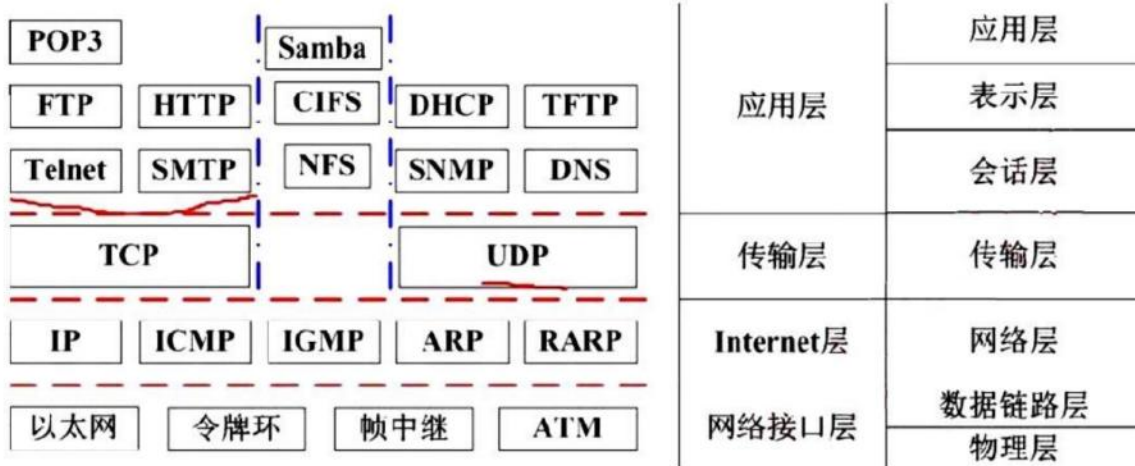
无线局域网 WLAN 技术标准：IEEE 802.11

广域网协议包括：PPP 点对点协议、ISDN 综合业务数字网、xDSL (DSL 数字用户线路的统称：

HDSL、SDSL、MVL、ADSL)、DDN 数字专线、x.25、FR 帧中继、ATM 异步传输模式。

TCP/IP 协议

网络协议三要素：语法、语义、时序。



网络层协议：

IP：网络层最重要的核心协议，在源地址和目的地址之间传送数据报，无连接、不可靠。

ICMP：因特网控制报文协议，用于在 IP 主机、路由器之间传递控制消息。控制消息是指网络不通、主机是否可达、路由是否可用等网络本身的消息。

ARP 和 RARP：地址解析协议，ARP 是将 IP 地址转换为物理地址，RARP 是将物理地址转换为 IP 地址。

IGMP：网络组管理协议，允许因特网中的计算机参加多播，是计算机用做向相邻多目路由器报告多组成员的协议，支持组播。

传输层协议：

TCP：整个 TCP/IP 协议族中最重要的协议之一，在 IP 协议提供的不可靠数据数据基础上，采用了重发技术，为应用程序提供了一个可靠的、面向连接的、全双工的数据传输服务。一般用于传输数据量比较少，且对可靠性要求高的场合。

UDP：是一种不可靠、无连接的协议，有助于提高传输速率，一般用于传输数据量大，对可靠性要求不高，但要求速度快的场合。

应用层协议：

基于 TCP 的 FTP、HTTP 等都是可靠传输。基于 UDP 的 DHCP、DNS 等都是不可靠传输。

FTP：可靠的文件传输协议，用于因特网上的控制文件的双向传输。

HTTP：超文本传输协议，用于从 WWW 服务器传输超文本到本地浏览器的传输协议。使用 SSL 加密后的安全网页协议为 HTTPS。

SMTP 和 POP3：简单邮件传输协议，是一组用于由源地址到目的地址传送邮件的规则，邮件报文采用 ASCII 格式表示。

Telnet：远程连接协议，是因特网远程登录服务的标准协议和主要方式。

TFTP：不可靠的、开销不大的小文件传输协议。

SNMP：简单网络管理协议，由一组网络管理的标准协议，包含一个应用层协议、数据库模型和一组资源对象。该协议能够支持网络管理系统，泳衣监测连接到网络上的设备是否有任何引起管理师行关注的情况。

DHCP：动态主机配置协议，基于 UDP，基于 C/S 模型，为主机动态分配 IP 地址，有三种方式：固定分配、动态分配、自动分配。

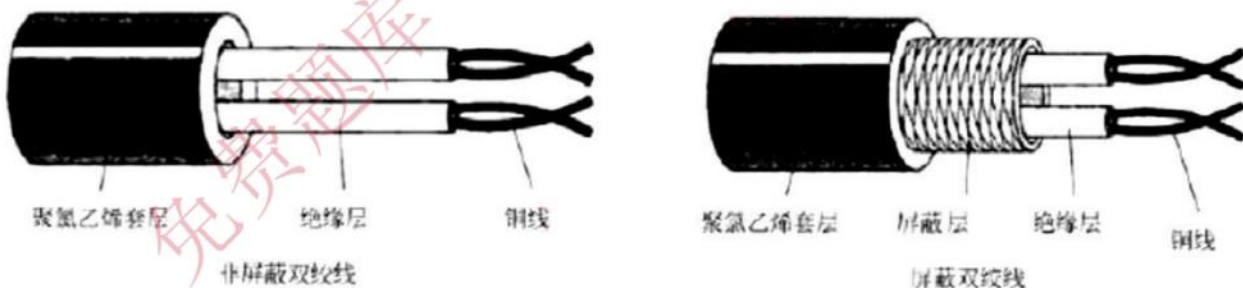
DNS：域名解析协议，通过域名解析出 IP 地址。

协议端口号对照表

端 口	服 务	端 口	服 务
20	文件传输协议（数据）	80	超文本传输协议（HTTP）
21	文件传输协议（控制）	110	POP3 服务器（邮箱接收服务器）
23	Telnet 终端仿真协议	69	简单文件传输协议（TFTP）
67	DHCP（服务端）	68	DHCP（客户端）
25	SMTP 简单邮件发送协议	161	SNMP（轮询）
53	域名服务器（DNS）	162	SNMP（陷阱）

2 传输介质

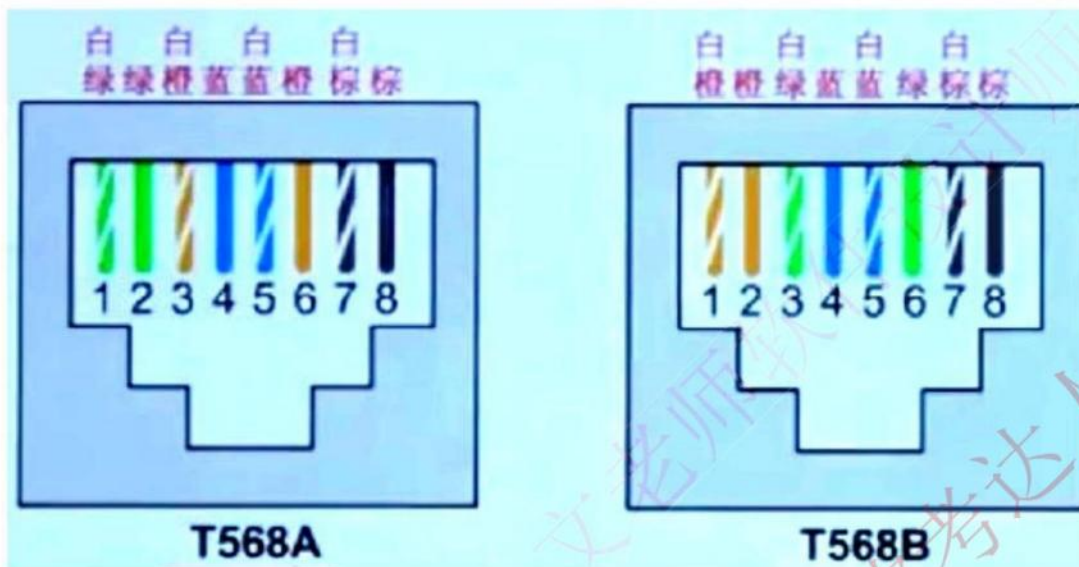
双绞线：将多根铜线按规则缠绕在一起，能够减少干扰；分为无屏蔽双绞线 UTP 和屏蔽双绞线 STP，都是由一对铜线簇组成。也即我们常说的网线；双绞线的传输距离在 100m 以内。



无屏蔽双绞线 UTP：价格低，安装简单，但可靠性相对较低，分为 CAT3（3 类 UTP，速率为 10Mbps）、CAT4（4 类 UTP，与 3 类差不多，无应用）、CAT5（5 类 UTP，速率为 100Mbps，用于快速以太网）、CAT5E（超 5 类 UTP，速率为 1000Mbps）、CAT6（6 类 UTP，用来替代 CAT5E，速率也是 1000Mbps）。

屏蔽双绞线 STP：比之 UTP 增加了一层屏蔽层，可以有效的提高可靠性，但对应的价格高，安装麻烦，一般用于对传输可靠性要求很高的场合。

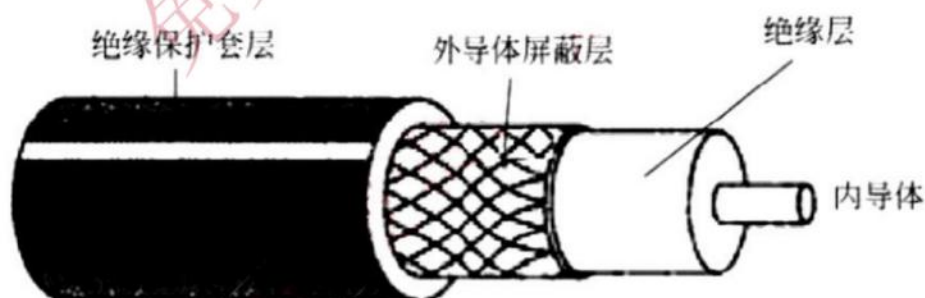
网线有如下两种安装标准：都是八根不同颜色的网线，按照不同的顺序排序，插入水晶头中，区分在第 1236 四根网线的位置不同。



网线又分为直通线和交叉线，直通线就是网线两头采用同一种标准，交叉线就是网线两头采用不同的标准，交叉线和直通线的使用范围如下：

线缆连接设备情况	所采用的线缆种类
计算机-计算机	交叉线
计算机-交换机	直通线
计算机-路由器	交叉线
交换机-交换机	交叉线
交换机-路由器	直通线
路由器-路由器	交叉线

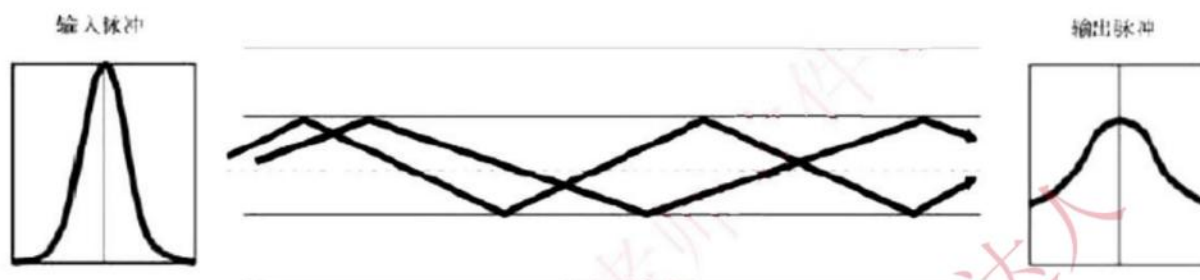
同轴电缆



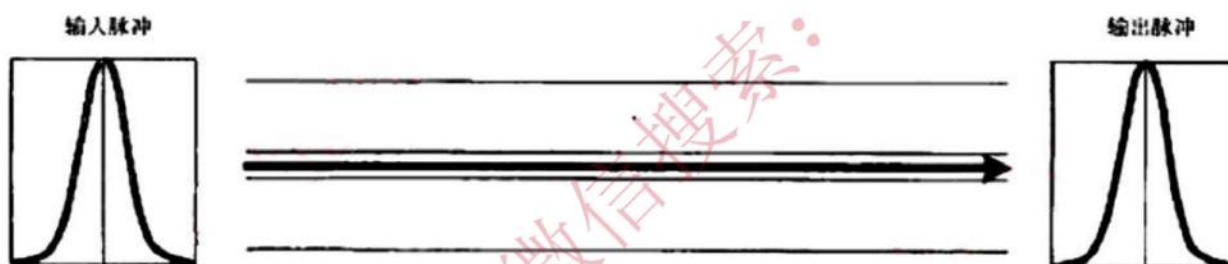
分为粗、细同轴电缆，传输距离长，信号稳定，常用于电视、监视系统、视频信号等。传输距离有几百米，随着数字网络的发展，已经逐步淘汰。

光纤：由纤芯和包层组成，传输的光信号在纤芯中传输，然而从 PC 端出来的信号都是电信号，要经过光纤传输的话，就必须将电信号转换为光信号。

多模光纤 MMF：纤芯半径较大，因此可以同时传输多种不同的信号，光信号在光纤中以全反射的形式传输，采用发光二极管 LED 为光源，成本低，但是传输的效率和可靠性都较低，适合于短距离传输，其传输距离与传输速率相关，速率为 100Mbps 时为 2KM，速率为 1000Mbps 时为 550m。



单模光纤 SMF：纤芯半径很小，一般只能传输一种信号，采用激光二极管 LD 作为光源，并且只支持激光信号的传播，同样是以全反射形式传播，只不过反射角很大，看起来像一条直线，成本高，但是传输距离远，可靠性高。传输距离可达 5KM。



无线信道

分为无线电波和红外光波

无线电波	长波		红外光波	近红外线
	中波			中红外线
	短波			
	超短波			
	微波	地面微波		远红外线
		卫星微波		

3 通信方式和交换方式

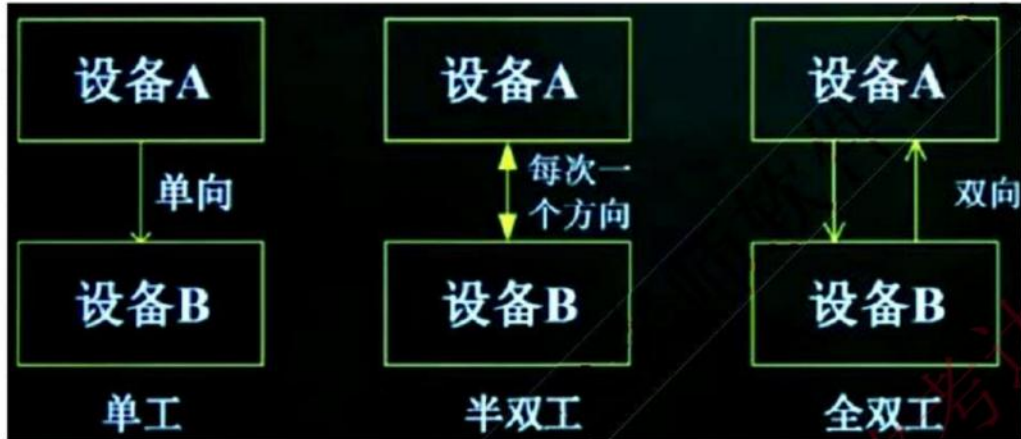
通信方向

数据通信是指发送方发送数据到接收方，这个传输过程可以分类如下：

单工：只能由设备 A 发给设备 B，即数据流只能单向流动。

半双工：设备 A 和设备 B 可以互相通信，但是同一时刻数据流只能单向流动。

全双工：设备 A 和设备 B 在任意时刻都能互相通信。



同步方式

异步传输：发送方每发送一个字符，需要约定一个起始位和停止位插入到字符的起始和结尾处，这样当接收方接收到该字符时能够识别，但是这样会造成资源浪费，传输效率降低。

同步传输：以数据块为单位进行传输，当发送方要发送数据时，先发送一个同步帧，接收方收到后做好接收准备，开始接收数据块，结束后又会有结束帧确认，这样一次传输一个数据块，效率高。

串行传输：只有一根数据线，数据只能 1bit 挨个排队传送，适合低速设备、远距离的传送，一般用于广域网中。

并行传输：有多根数据线，可以同时传输多个 bit 数据，适合高速设备的传送，常用语计算机内部各硬件模块之间。

交换方式

电路交换（线路交换）：通信一方进行呼叫，另一方接收后，在二者之间会建立一个专用电路，特点为面向连接、实时性高、链路利用率低、但没有差错检测，不可靠，一般用于语音视频通信。

报文交换：以报文为单位，存储转发模式，接收到数据后先存储，进行差错校验，没有错误则转发，有错误则丢弃，因此会有延时，但可靠性高，是面向无连接的。

分组交换：以分组为单位，也是存储转发模式，因为分组的长度比报文小，所以时延小于报文交换，又可分为三种方式：

——数据报：是现在主流的交换方式，各个分组携带地址信息，自由的选择不同的路由路径传送到接收方，接收方接收到分组后再根据地址信息重新组装成原数据，是面向无连接的，但是不可靠的。

——虚电路：发送方发送一个分组，接收方收到后二者之间就建立了一个虚拟的通信线路，二者之间的分组数据交互都通过这条线路传送，在空闲的时候这条线路也可以传输其他数据，是面向连接的，可靠的。

——信元交换：异步传输模式 ATM 采用的交换方式，本质是按照虚电路方式进行转发，只不过信元是固定长度的分组，共 53B，其中 5B 为头部，48B 为数据域，也是面向连接的，可靠的。

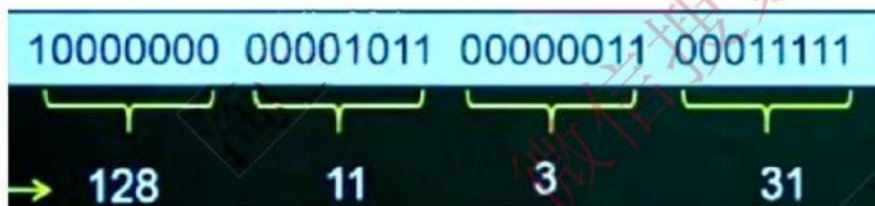
4 IP 地址

4.1 分类地址格式

主机位：全 0 是本地网络地址，全 1 是广播地址，其他是主机地址。

IP 地址的表示

机器中存放的 IP 地址是 32 位的二进制代码，每隔 8 位插入一个空格，可提高可读性，为了便于理解和设置，一般会采用点分十进制方法表示：将 32 位二进制代码每 8 位二进制转换成十进制，就变成了 4 个十进制数，而后在每个十进制数间隔中插入“.”，如下所示，最终为 128.11.3.31：



因为每个十进制数都是由 8 个二进制数转换而来，因此每个十进制数的取值范围为 0-255（掌握二进制转十进制的快速计算方法，牢记 2 的幂指数值，实现快速转换）。

分类 IP 地址

IP 地址分四段，每段八位，共 32 位二进制数组成。

在逻辑上，这 32 位 IP 地址分为网络号和主机号，依据网络号位数的不同，可以将 IP 地址分为以下几类：

类别	点分十进制	二进制	
A类	0.0.0.0	最低	00000000 00000000 00000000 00000000
	127.255.255.255	最高	01111111 11111111 11111111 11111111
B类	128.0.0.0	最低	10000000 00000000 00000000 00000000
	191.255.255.255	最高	10111111 11111111 11111111 11111111
C类	192.0.0.0	最低	11000000 00000000 00000000 00000000
	223.255.255.255	最高	11011111 11111111 11111111 11111111
D类 组播	224.0.0.0	最低	11100000 00000000 00000000 00000000
	239.255.255.255	最高	11101111 11111111 11111111 11111111
E类 保留	240.0.0.0	最低	11110000 00000000 00000000 00000000
	255.255.255.255	最高	11111111 11111111 11111111 11111111

特殊 IP 地址

公有地址：由 Inter NIC（因特网信息中心）负责，这些 IP 地址分配给注册并向 Inter NIC 提出申请的组织机构，通过它直接访问因特网。是全网唯一的 IP 地址。

私有地址：属于非注册地址，专门为组织机构内部使用，在不同企业内部可以重复使用，不能直接访问因特网，下表所示为私有地址范围：

类别	IP地址范围	网络号	网络数
A	10.0.0.0~10.255.255.255	10	1
B	172.16.0.0~172.31.255.255	172.16~172.31	16
C	192.168.0.0~192.168.255.255	192.168.0~192.168.255	256

其他特殊地址如下表所示：

网络号	主机号	源地址使用	目的地址使用	代表的意思
0	0	可以	不可	在本网络上的本主机
全1	全1	不可	可以	在本网络上进行广播
Net-ID	全1	不可	可以	对net-ID上的所有主机进行广播
127	非全0或全1的数	可以	可以	用作本地软件环回测试之用
169.254	非全0或全1的数	可以	可以	Windows主机DHCP服务器故障分配

169.254 是自动专用 IP 地址，当 DHCP 故障时为主机分配的临时 IP 地址。

4.2 子网划分

可变长子网掩码 VLSM

按上述划分的 A B C 三类，并不实用，因为主机数之间相差的太大了，不利于分配，导致 IP 地址空间的利用率很低，并且两级 IP 地址不够灵活。

因此，我们一般采用子网划分的方法来划分网络，将主机号拿出几位作为子网号，就可以划分出多个子网，此时 IP 地址组成为：网络号+子网号+主机号。（本质还是保留了分类编址结构）

网络号和子网号都为 1，主机号都为 0，这样的地址为子网掩码。

要判断两个 IP 地址是否在同一网段，只需要确定其网络号+子网号是否相同，可以将子网掩码和源 IP 地址进行与运算，得出网络地址，判断网络地址是否相同即可。网络号不同的主机属于不同的网段，不能直接通信，必须通过网络路由才能互相通信。

要注意的是：子网号可以为全 0 和全 1，主机号不能为全 0 或全 1，因此，在计算时，主机数需要-2，而子网数不用。

划分子网之后，就变成了三级 IP 地址，如果此时还需要进一步划分子网，还可以继续向主机号借位，将主机号拿出几位再作为子网号，这就是可变长度子网划分，根据需要灵活选择，但要注意的是都是向主机号借位。

4.3 无分类编址

上述的子网划分方式，本质是在分类编址的基础上，向主机位借位，还是保留了分类编址的结构，而无分类编址 CIDR 方式，则是完全忽略分类编址，可以按照需要自定义网络号和主机号位数（只需要保证整体 IP 地址长度为 32bit 即可）。

无分类编址格式为：IP 地址/网络号，示例：128.168.0.11/20 表示的 IP 地址为 128.168.0.11，其网络号占 20 位，因此主机号占 $32-20=12$ 位，无分类编址也可以划分子网，原理相同。要掌握的是，/后的数字是网络号，32 减去此数字就是主机号。

CIDR 路由汇聚

汇聚规则：选择网络地址相同的位进行汇聚，不同的位划分至主机位，从而实现将多个网段汇聚成一个新的超网网段。

可以将汇聚超网的过程理解成划分子网的逆过程，划分子网是不断向主机位借位，导致子网越来越多，每个子网的主机号越来越少；而汇聚超网是将不同网段的相同的网络号作为汇聚后的网络号，不同的网络号给主机号，相当于缩小了子网的数量，增加了网络内主机的数量。

最佳路由匹配原则

使用路由汇聚时，路由表中的每个项目由网络前缀和下一跳地址组成。因此查找路由表时可能会得到不止一个匹配结果。此时应当从匹配结果中选择具有最长网络前缀的路由。

网络前缀越长,其地址块就越小,因而路由就越具体,最长前缀匹配又称为最长匹配或最佳匹配。

5 IPv6

主要是为了解决 IPv4 地址数不够用的情况而提出的设计方案, IPv6 具有以下特性:

IPv6 地址长度为 128 位, 地址空间增大了 2^{96} 倍;

灵活的 IP 报文头部格式, 使用一系列固定格式的扩展头部取代了 IPv4 中可变长度的选项字段。

IPv6 中选项部分的出现方式也有所变化, 使路由器可以简单撸过选项而不做任何处理, 加快了报文处理速度;

IPv6 简化了报文头部格式, 加快报文转发, 提高了吞吐量;

提高安全性, 身份认证和隐私权是 IPv6 的关键特性;

支持更多的服务类型;

允许协议继续演变, 增加新的功能, 使之适应未来技术的发展。

IPv4 和 IPv6 的过渡期间, 主要采用三种基本技术:

(1) 双协议栈: 主机同时运行 IPv4 和 IPv6 两套协议栈, 同时支持两套协议, 一般来说 IPv4 和 IPv6 地址之间存在某种转换关系, 如 IPv6 的低 32 位可以直接转换为 IPv4 地址, 实现互相通信。

(2) 隧道技术: 这种机制用来在 IPv4 网络之上建立一条能够传输 IPv6 数据报的隧道, 例如可以将 IPv6 数据报当做 IPv4 数据报的数据部分加以封装, 只需要加一个 IPv4 的首部, 就能在 IPv4 网络中传输 IPv6 报文。

(3) 翻译技术: 利用一台专门的翻译设备(如转换网关), 在纯 IPv4 和纯 IPv6 网络之间转换 IP 报头的地址, 同时根据协议不同对分组做相应的语义翻译, 从而使纯 IPv4 和纯 IPv6 站点之间能够透明通信。

6 网络规划和设计

6.1 层次化局域网模型

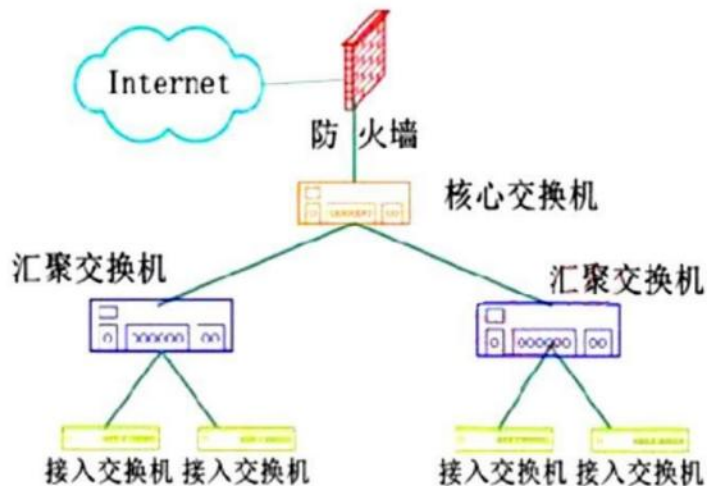
三层模型将网络划分为核心层、汇聚层和接入层, 每一层都有着特定的作用。

核心层提供不同区域之间的最佳路由和高速数据传送;

汇聚层将网络业务连接到接入层, 并且实施与安全、流量、负载和路由相关的策略;

接入层为用户提供了在本地网段访问应用系统的能力，还要解决相邻用户之间的互访需要，接入层要负责一些用户信息（例如用户 IP 地址、MAC 地址和访问日志等）的收集工作和用户管理功能（包括认证和计费等）。

下面是一个三层示例，一般设计思路都是从下往上设计，即先设计接入层、汇聚层，再考虑核心层数据交换的速度以满足下面两层的要求。



6.2 建筑物综合布线系统 PDS

工作区子系统：实现工作区终端设备到水平子系统的信息插座之间的互联。

水平布线子系统：实现信息插座和管理子系统之间的连接。

设备间子系统：实现中央主配线架与各种不同设备之间的连接。

垂直干线子系统：实现各楼层设备间子系统之间的互连。

管理子系统：为连接其他子系统提供连接手段。

建筑群子系统：各个建筑物通信系统之间的互联。

7 网络管理命令

ipconfig 命令：用来查看网络配置信息，如 IP 地址、子网掩码、网管 IP、DNS、DHCP 等。Linux 下是 ifconfig 命令。

ping 命令：利用 ICMP 报文来测试网络的连通性、是否丢包、名称解析等，如 ping 127.0.0.1 检查 TCP/IP 协议，ping 本机 IP 检查网卡工作，ping 网关等。

arp 命令：用于显示和修改 ARP 缓存中的表项。

netstat 命令：用来显示网络活动状态，如 TCP/UDP 的 IP、端口号、统计信息等。

tracert 命令：利用 ICMP 报文，来探测到达目标的路径，参数通常为-d。

pathing 命令：把 ping 和 tracert 结合起来，探测路径、延时、丢包率等。

nbtstat 命令：用来显示 NetBIOS 的名称缓存。

route 命令：显示和修改本地 IP 路由器表，参数 add 添加、delete 删除、change 修改、print 显示路由表（同 netstat -r）。

netsh 命令：命令行脚本程序，可修改计算机的网络配置。

net 命令：管理网络服务。

nslookup 命令：命令用于显示 DNS 查询信息，诊断和排除 DNS 故障。

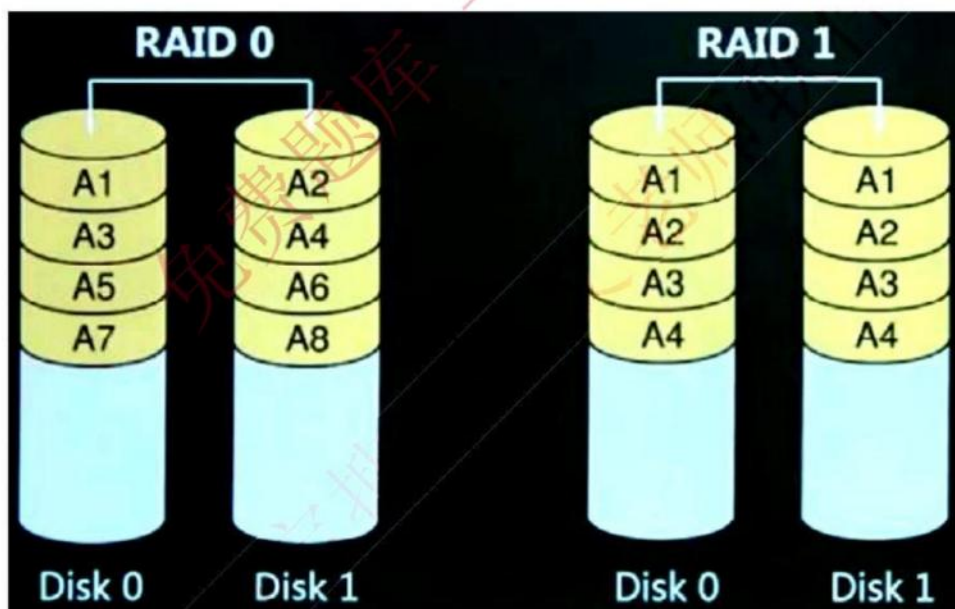
8 网络存储技术

8.1 廉价磁盘冗余阵列

RAID 即磁盘冗余阵列技术，将数据分散存储在不同磁盘中，可并行读取，可冗余存储，提高磁盘访问速度，保障数据安全性。

RAID0 将数据分散的存储在不同磁盘中，磁盘利用率 100%，访问速度最快，但是没有提供冗余和错误修复技术；

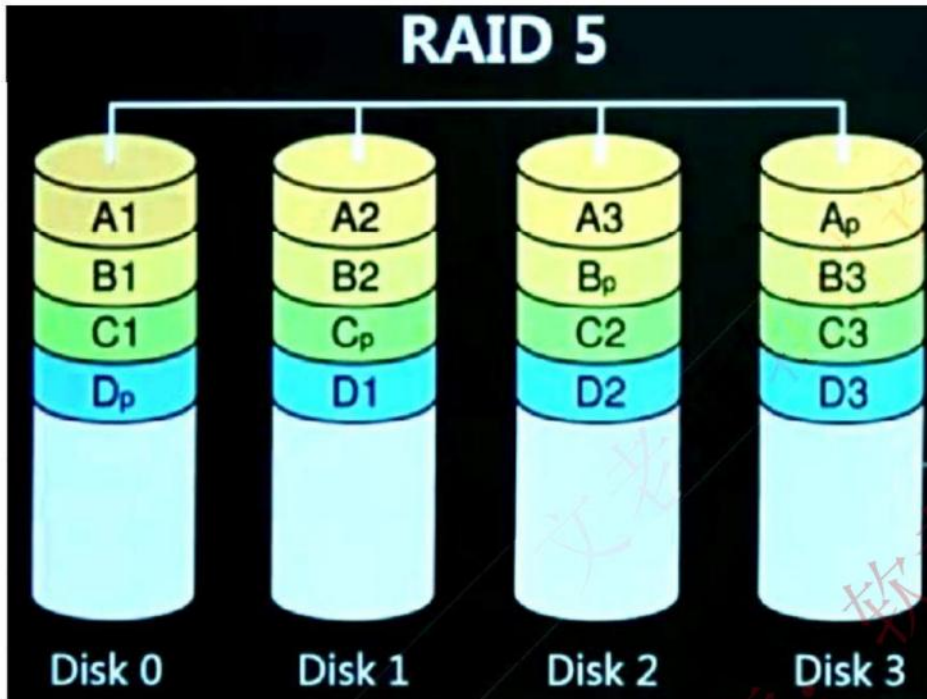
RAID1 在成对的独立磁盘上产生互为备份的数据，增加存储可靠性，可以纠错，但磁盘利用率只有 50%；



RAID2 将数据条块化的分布于不同硬盘上，并使用海明码校验；

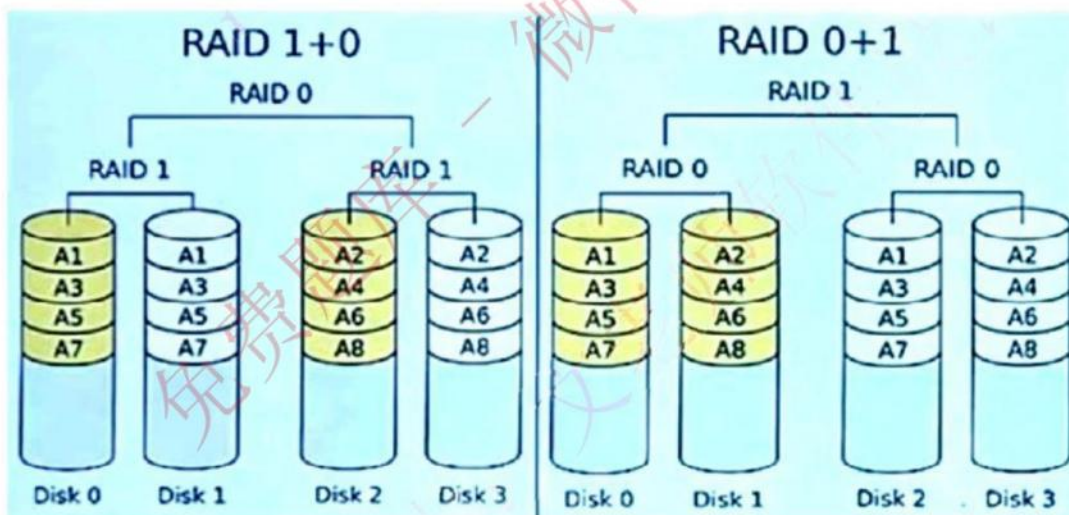
RAID3 使用奇偶校验，并用单块磁盘存储奇偶校验信息（可靠性低于 RAID5）；

RAID5 在所有磁盘上交叉的存储数据及奇偶校验信息（所有校验信息存储总量为一个磁盘容量，但分布式存储在不同的磁盘上），读/写指针可同时操作；



RAID0+1（是两个 RAID0，若一个磁盘损坏，则当前 RAID0 无法工作，即有一半的磁盘无法工作）；

RAID1+0（是两个 RAID1，不允许同一组中的两个磁盘同时损坏）与 RAID1 原理类似，磁盘利用率都只有 50%，但安全性更高。



8.2 网络存储

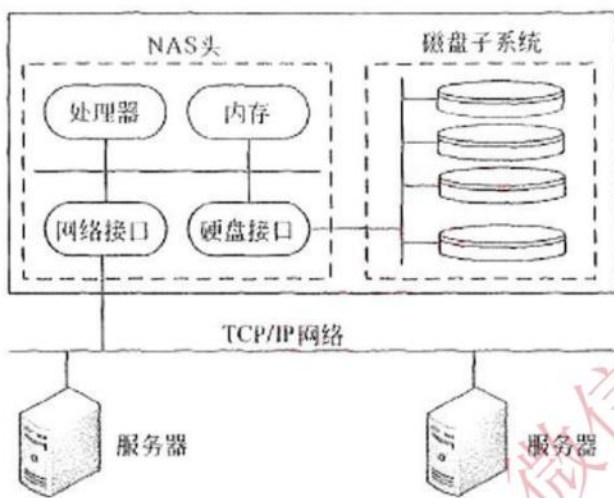
具体有下列四种存储技术：

1.直接附加存储(DAS)：是指将存储设备通过 SCSI 接口直接连接到一台服务器上使用，其本身是硬件的堆叠，存储操作依赖于服务器，不带有存储操作系统。

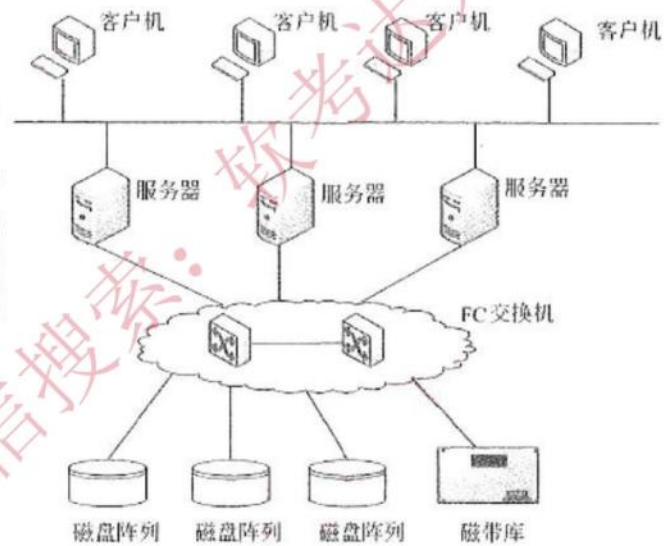
存在问题：在传递距离、连接数量、传输速率等方面都受到限制。容量难以扩展升级；数据处理和传输能力降低；服务器异常会波及存储器。

2.网络附加存储(NAS)：通过网络接口与网络直接相连，由用户通过网络访问，有独立的存储系统。如下图所示。NAS 存储设备类似于一个专用的文件服务器，去掉了通用服务器大多数计算功能，而仅提供文件系统功能。以数据为中心，将存储设备与服务器分离，其存储设备在功能上完全独立于网络中的主服务器。客户机与存储设备之间的数据访问不再需要文件服务器的干预，同时它允许客户机与存储设备之间进行直接的数据访问，所以不仅响应速度快，而且数据传输速率也很高。

NAS 的性能特点是进行小文件级的共享存取；支持即插即用；可以很经济的解决存储容量不足的问题，但难以获得满意的性能。



NAS 存储系统的结构



SAN 存储系统的结构

3.存储区域网(SAN)：SAN 是通过专用交换机将磁盘阵列与服务器连接起来的高速专用子网。它没有采用文件共享存取方式，而是采用块 (block) 级别存储。SAN 是通过专用高速网将一个或多个网络存储设备和服务器连接起来的专用存储系统，其最大特点是将存储设备从传统的以太网中分离了出来，成为独立的存储区域网络 SAN 的系统结构。根据数据传输过程采用的协议，其技术划分为 FC SAN (光纤通道)、IP SAN (IP 网络) 和 IB SAN (无线带宽) 技术。

9 其他考点汇总

网络地址翻译 NAT

公司内有很多电脑，在公司局域网内可以互联通信，但是要访问外部因特网时，只提供固定的少量 IP 地址能够访问因特网，将公司所有电脑这个大的地址集合映射到能够访问因特网的少量 IP 地址集合的过程就称为 NAT。

很明显,使用了 NAT 后,一个公司只有少量固定 IP 地址可以上网,大大减少了 IP 地址的使用量。

默认网关

一台主机可以有多个网关。默认网关的意思是一台主机如果找不到可用的网关,就把数据包发给默认指定的网关,由这个网关来处理数据包。现在主机使用的网关,一般指的是默认网关。

默认网关的 IP 地址必须与本机 IP 地址在同一个网段内,即同网络号。

虚拟局域网 VLAN

是一组逻辑上的设备和用户,这些设备和用户并不受物理位置的限制,可以根据功能、部门及应用等因素将它们组织起来,相互之间的通信就好像它们在同一个网段中一样。

VLAN 工作在 OSI 参考模型的第 2 层和第 3 层,一个 VLAN 就是一个广播域,VLAN 之间的通信是通过第 3 层的路由器来完成的。

与传统的局域网技术相比较,VLAN 技术更加灵活,它具有以下优点:网络设备的移动、添加和修改的管理开销减少;可以控制广播活动;可提高网络的安全性。

虚拟专用网 VPN

是在公用网络上建立专用网络的技术。其之所以称为虚拟网,主要是因为整个 VPN 网络的任意两个节点之间的连接并没有传统专网所需的端到端的物理链路,而是架构在公用网络服务商所提供的网络平台,如 Internet、ATM(异步传输模式)、Frame Relay (帧中继)等之上的逻辑网络,用户数据在逻辑链路中传输。

PPP

安全认证介绍:PPP 的 NCP 可以承载多种协议的三层数据包。PPP 使用 LCP 控制多种链路的参数(建立、认证、压缩、回拨)。

PPP 的认证类型:pap 认证是通过二次握手建立认证(明文不加密),chap 挑战握手认证协议,通过三次握手建立认证(密文采用 MD5 加密)。PPP 的双向验证,采用的是 chap 的主验证风格。PPP 的加固验证,采用的是两种(pap,chap)验证同时使用

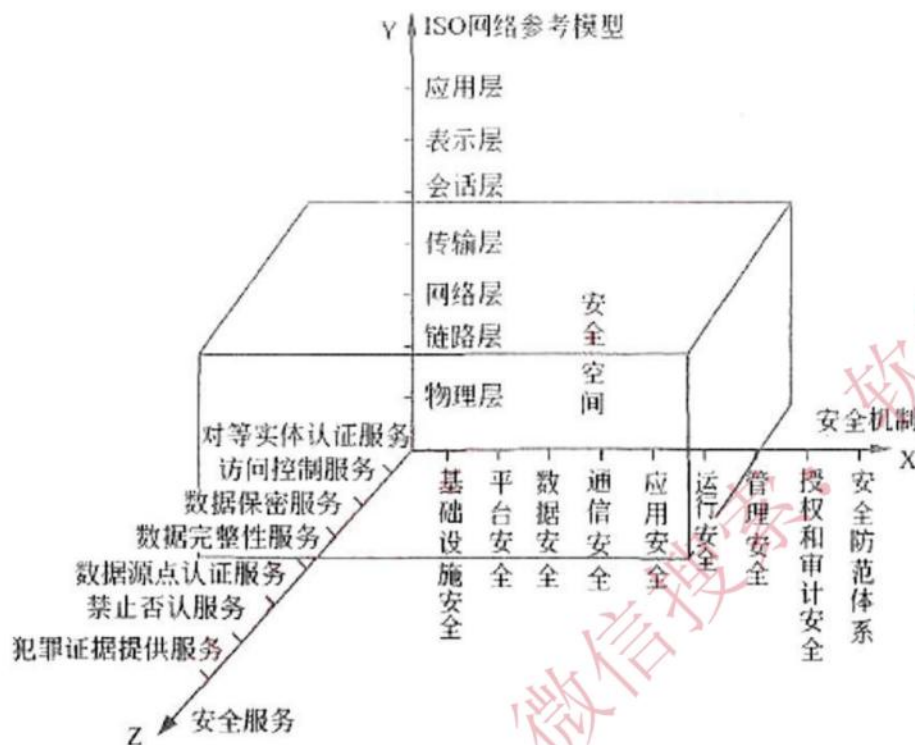
冲突域和广播域

路由器可以阻断广播域和冲突域,交换机只能阻断冲突域,因此一个路由器下可以划分多个广播域和多个冲突域;一个交换机下整体是一个广播域,但可以划分多个冲突域;而物理层设备集线器下整体作为一个冲突域和一个广播域。

五 信息安全和网络安全

1 信息安全和信息系统安全

信息安全系统的体系架构



X 轴是“安全机制”，为提供某些安全服务，利用各种安全技术和技巧，所形成的一个较为完善的机构体系。

Y 轴是“OSI 网络参考模型”。

Z 轴是“安全服务”。就是从网络中的各个层次提供给信息应用系统所需要的安全服务支持。

由 X、Y、Z 三个轴形成的信息安全系统三维空间就是信息系统的“安全空间”。

随着网络逐层扩展，这个空间不仅范围逐步加大，安全的内涵也就更丰富，达到具有认证、权限、完整、加密和不可否认五大要素，也叫作“安全空间”的五大属性。

信息安全含义及属性

保护信息的保密性、完整性、可用性，另外也包括其他属性，如：真实性、可核查性、不可抵赖性和可靠性。

保密性：信息不被泄漏给未授权的个人、实体和过程或不被其使用的特性。包括：（1）最小授权原则（2）防暴露（3）信息加密（4）物理保密

完整性:信息未经授权不能改变的特性。影响完整性的主要因素有设备故障、误码、人为攻击和计算机病毒等。保证完整性的方法包括:

- (1) 协议:通过安全协议检测出被删除、失效、被修改的字段。
- (2) 纠错编码方法:利用校验码完成检错和纠错功能。
- (3) 密码校验和方法。
- (4) 数字签名:能识别出发送方来源。
- (5) 公证:请求系统管理或中介机构证明信息的真实性。

可用性:需要时,授权实体可以访问和使用的特性。一般用系统正常使用时间和整个工作时间之比来度量。

其他属性:

真实性:指对信息的来源进行判断,能对伪造来源的信息予以鉴别。

可核查性:系统实体的行为可以被独一无二的追溯到该实体的特性,这个特性就是要求该实体对其行为负责,为探测和调查安全违规事件提供了可能性。

不可抵赖性:是指建立有效的责任机制,防止用户否认其行为,这一点在电子商务中是极其重要的。

可靠性:系统在规定的的时间和给定的条件下,无故障地完成规定功能的概率。

安全需求

可划分为物理线路安全、网络安全、系统安全和应用安全;从各级安全需求字面上也可以理解:

物理线路就是物理设备、物理环境;

网络安全指网络上的攻击、入侵;

系统安全指的是操作系统漏洞、补丁等;

应用安全就是上层的应用软件,包括数据库软件。

2 信息安全技术

加密技术

一个密码系统,通常简称为密码体制(Cryptosystem),由五部分组成:

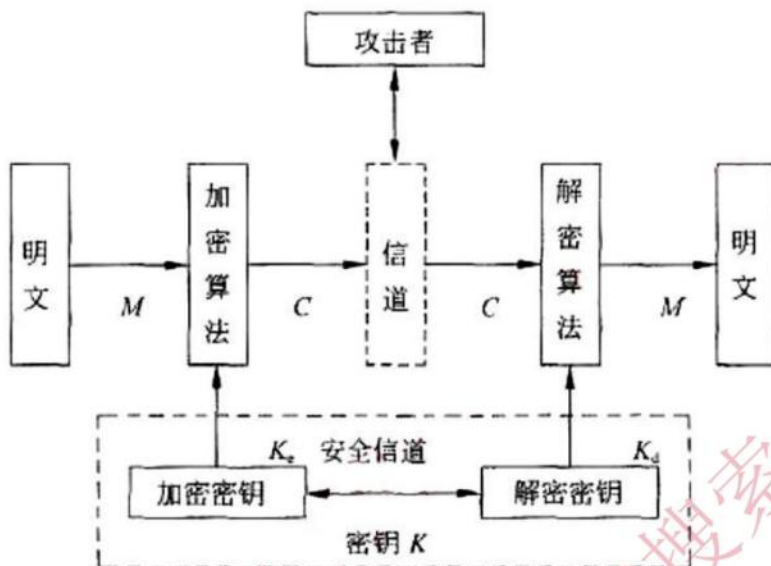
- (1) 明文空间 M ,它是全体明文的集合。
- (2) 密文空间 C ,它是全体密文的集合。
- (3) 密钥空间 K ,它是全体密钥的集合。其中每一个密钥 K 均由加密密钥 K_e 和解密密钥 K_d 组成,即 $K = \langle K_e, K_d \rangle$ 。

(4) 加密算法 E，它是一组由 M 至 C 的加密变换。

(5) 解密算法 D，它是一组由 C 到 M 的解密变换。

对于每一个确定的密钥，加密算法将确定一个具体的加密变换，解密算法将确定一个具体的解密变换，而且解密变换就是加密变换的逆变换。对于明文空间 M 中的每一个明文 M，加密算法 E 在密钥 K_e 的控制下将明文 M 加密成密文 C: $C=E(M, K_e)$

而解密算法 D 在密钥 K_d 的控制下将密文 C 解密出同一明文 M: $M=D(C, K_d)=D(E(M, K_e), K_d)$



对称加密技术

就是对数据的加密和解密的密钥（密码）是相同的，属于不公开密钥加密算法。其缺点是加密强度不高（因为只有一个密钥），且密钥分发困难（因为密钥还需要传输给接收方，也要考虑保密性等问题）。

常见的对称密钥加密算法如下：

DES：替换+移位、56 位密钥、64 位数据块、速度快，密钥易产生。

3DES：三重 DES，两个 56 位密钥 K1、K2。

加密：K1 加密→K2 解密→K1 加密。

解密：K1 解密→K2 加密→K1 解密

AES：是美国联邦政府采用的一种区块加密标准，这个标准用来替代原先的 DES。对其的要求是“至少像 3DES 一样安全”。

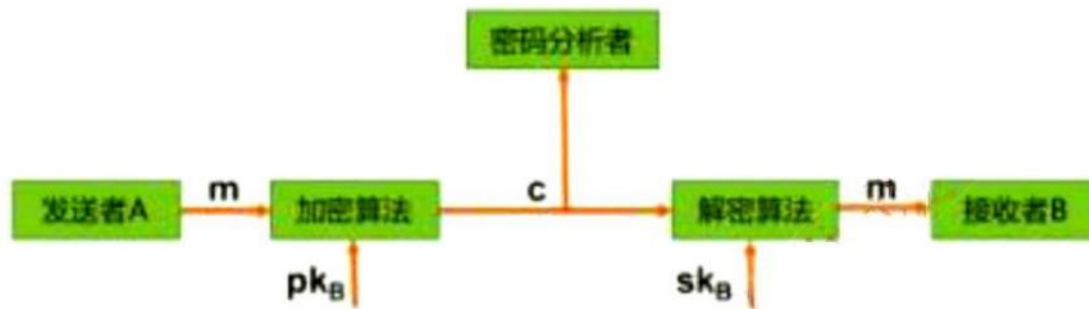
RC-5：RSA 数据安全公司的很多产品都使用了 RC-5。

IDEA：128 位密钥，64 位数据块，比 DES 的加密性好，对计算机功能要求相对低。

非对称加密技术

就是对数据的加密和解密的密钥是不同的，是公开密钥加密算法。其缺点是加密速度慢。

非对称技术的原理是：发送者发送数据时，使用接收者的公钥作加密密钥，私钥作解密密钥，这样只有接收者才能解密密文得到明文。安全性更高，因为无需传输密钥。但无法保证完整性。如下：



常见的非对称加密算法如下：

RSA：512 位（或 1024 位）密钥，计算量极大，难破解。

Elgamal、**ECC**（椭圆曲线算法）、背包算法、**Rabin**、**D-H** 等。

相比较可知，对称加密算法密钥一般只有 56 位，因此加密过程简单，适合加密大数据，也因此加密强度不高；而非对称加密算法密钥有 1024 位，相应的解密计算量庞大，难以破解，却不适合加密大数据，一般用来加密对称算法的密钥，这样，就将两个技术组合使用了，这也是数字信封的原理：

数字信封原理：信是对称加密的密钥，数字信封就是对此密钥进行非对称加密，具体过程：发送方将数据用对称密钥加密传输，而将对称密钥用接收方公钥加密发送给对方。接收方收到数字信封，用自己的私钥解密信封，取出对称密钥解密得原文。

数字信封运用了对称加密技术和非对称加密技术，本质是使用对称密钥加密数据，非对称密钥加密对称密钥，解决了对称密钥的传输问题。

信息摘要

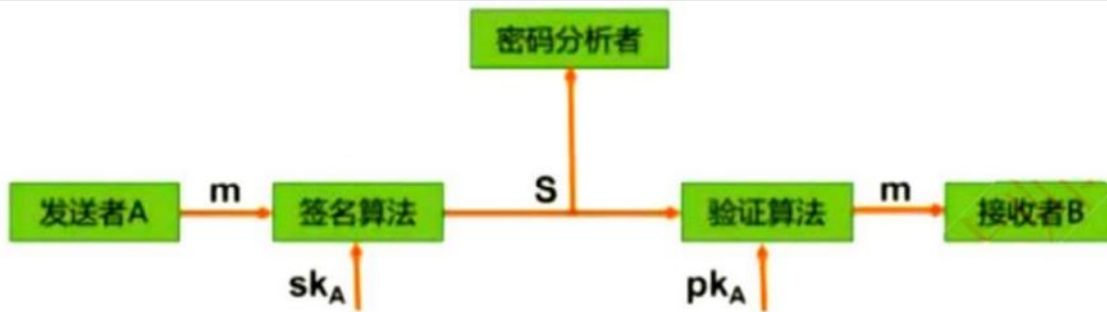
所谓信息摘要，就是一段数据的特征信息，当数据发生了改变，信息摘要也会发生改变，发送方会将数据和信息摘要一起传给接收方，接收方会根据接收到的数据重新生成一个信息摘要，若此摘要和接收到的摘要相同，则说明数据正确。**信息摘要是由哈希函数生成的。**

信息摘要的特点：不算数据多长，都会产生固定长度的信息摘要；任何不同的输入数据，都会产生不同的信息摘要；单向性，即只能由数据生成信息摘要，不能由信息摘要还原数据。

信息摘要算法：**MD5**（产生 128 位的输出）、**SHA-1**（安全散列算法，产生 160 位的输出，安全性更高）。

数字签名：唯一标识一个发送方。

发送者发送数据时，使用发送者的私钥进行加密，接收者收到数据后，只能使用发送者的公钥进行解密，这样就能唯一确定发送方，这也是数字签名的过程。但无法保证机密性。如下：



公钥基础设施 PKI: 是以不对称密钥加密技术为基础，以数据机密性、完整性、身份认证和行为不可抵赖性为安全目的，来实施和提供安全服务的具有普适性的安全基础设施。

(1) 数字证书：一个数据结构，是一种由一个可信任的权威机构签署的信息集合。在不同的应用中有不同的证书。如 X.509 证书必须包含下列信息：(1) 版本号 (2) 序列号 (3) 签名算法标识符 (4) 认证机构 (5) 有效期限 (6) 主题信息 (7) 认证机构的数字签名 (8) 公钥信息。

公钥证书主要用于确保公钥及其与用户绑定关系的安全。这个公钥就是证书所标识的那个主体的合法的公钥。任何一个用户只要知道签证机构的公钥，就能检查对证书的签名的合法性。如果检查正确，那么用户就可以相信那个证书所携带的公钥是真实的，而且这个公钥就是证书所标识的那个主体的合法的公钥。例如驾照。

(2) 签证机构 CA：负责签发证书、管理和撤销证书。是所有注册用户所信赖的权威机构，CA 在给用户签发证书时要加上自己的数字签名，以保证证书信息的真实性。任何机构可以用 CA 的公钥来验证该证书的合法性。

3 网络安全技术

防火墙是在内部网络和外部因特网之间增加的一道安全防护措施，分为网络级防火墙和应用级防火墙。

网络级防火墙层次低，但是效率高，因为其使用包过滤和状态监测手段，一般只检验网络包外在（起始地址、状态）属性是否异常，若异常，则过滤掉，不与内网通信，因此对应用和用户是透明的。但是这样的问题是，如果遇到伪装的危险数据包就没办法过滤，此时，就要依靠**应用级防火墙**，层次高，效率低，因为应用级防火墙会将网络包拆开，具体检查里面的数据是否有问题，会消耗大量时间，造成效率低下，但是安全强度高。

入侵检测系统 IDS

防火墙技术主要是分隔来自外网的威胁，却对来自内网的直接攻击无能为力，此时就要用到入侵检测 IDS 技术，位于防火墙之后的第二道屏障，作为防火墙技术的补充。

原理：监控当前系统/用户行为，使用入侵检测分析引擎进行分析，这里包含一个知识库系统，囊括了历史行为、特定行为模式等操作，将当前行为和知识库进行匹配，就能检测出当前行为是否是入侵行为，如果是入侵，则记录证据并上报给系统和防火墙，交由它们处理。

不同于防火墙，IDS 入侵检测系统是一个监听设备，没有跨接在任何链路上，无须网络流量流经它便可以工作。因此，对 IDS 的部署，唯一的要求是：IDS 应当挂接在所有所关注流量都必须流经的链路上。因此，IDS 在交换式网络中的位置一般选择在：（1）尽可能靠近攻击源（2）尽可能靠近受保护资源

入侵防御系统 IPS

IDS 和防火墙技术都是在入侵行为已经发生后所做的检测和分析，而 IPS 是能够提前发现入侵行为，在其还没有进入安全网络之前就防御。串联接入网络，因此可以自动切换网络。

在安全网络之前的链路上挂载入侵防御系统 IPS，可以实时检测入侵行为，并直接进行阻断，这是与 IDS 的区别，要注意。

杀毒软件

用于检测和解决计算机病毒，与防火墙和 IDS 要区分，计算机病毒要靠杀毒软件，防火墙是处理网络上的非法攻击。

蜜罐系统

伪造一个蜜罐网络引诱黑客攻击，蜜罐网络被攻击不影响安全网络，并且可以借此了解黑客攻击的手段和原理，从而对安全系统进行升级和优化。

网络攻击和威胁

攻击类型	攻击名称	描述
被动攻击	窃听（网络监听）	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。
	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。
	非法登录	有些资料将这种方式归为被动攻击方式。

主动攻击	假冒身份	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。
	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。
	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。
	重放攻击	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。

计算机病毒和木马

病毒：编制或者在计算机程序中插入的破坏计算机功能或者破坏数据，影响计算机使用并且能够自我复制的一组计算机指令或者程序代码。

病毒具有：传染性、隐蔽性、潜伏性、破坏性、针对性、衍生性、寄生性、未知性。

木马：是一种后门程序，常被黑客用作控制远程计算机的工具，隐藏在被控制电脑上的一个小程序监控电脑一切操作并盗取信息。

代表性病毒实例

蠕虫病毒（感染 EXE 文件）：熊猫烧香，罗密欧与朱丽叶，恶鹰，尼姆达，冲击波，欢乐时光。

木马：QQ 消息尾巴木马，特洛伊木马，X 卧底。

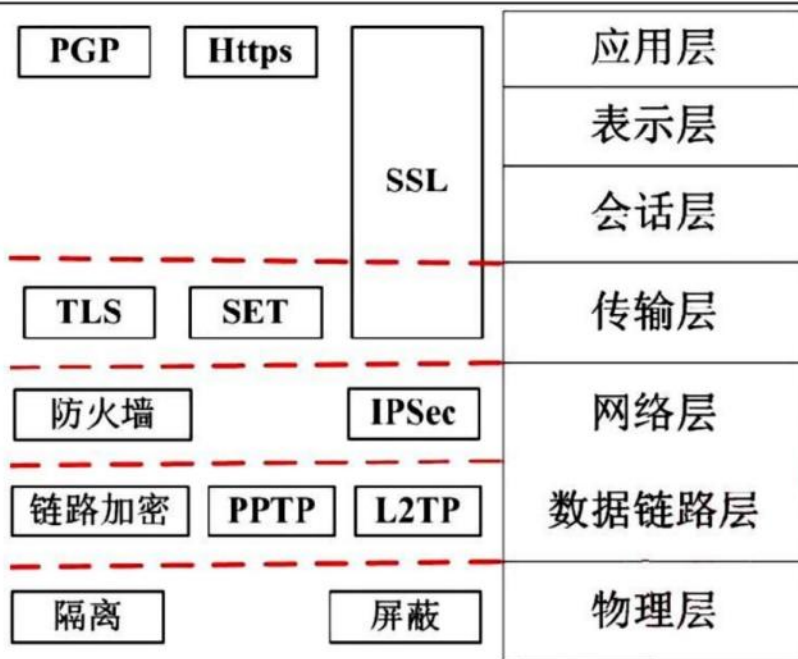
宏病毒（感染 word、excel 等文件中的宏变量）：美丽沙，台湾 1 号。

CIH 病毒：史上唯一破坏硬件的病毒。

红色代码：蠕虫病毒+木马。

4 网络安全协议

物理层主要使用物理手段，隔离、屏蔽物理设备等，其它层都是靠协议来保证传输的安全，具体如下图所示：



◆ **SSL 协议**：安全套接字协议，被设计为加强 Web 安全传输(HTTP/HTTPS/)的协议，安全性高，和 HTTP 结合之后，形成 HTTPS 安全协议，端口号为 443。

◆ **SSH 协议**：安全外壳协议，被设计为加强 Telnet/FTP 安全的传输协议。

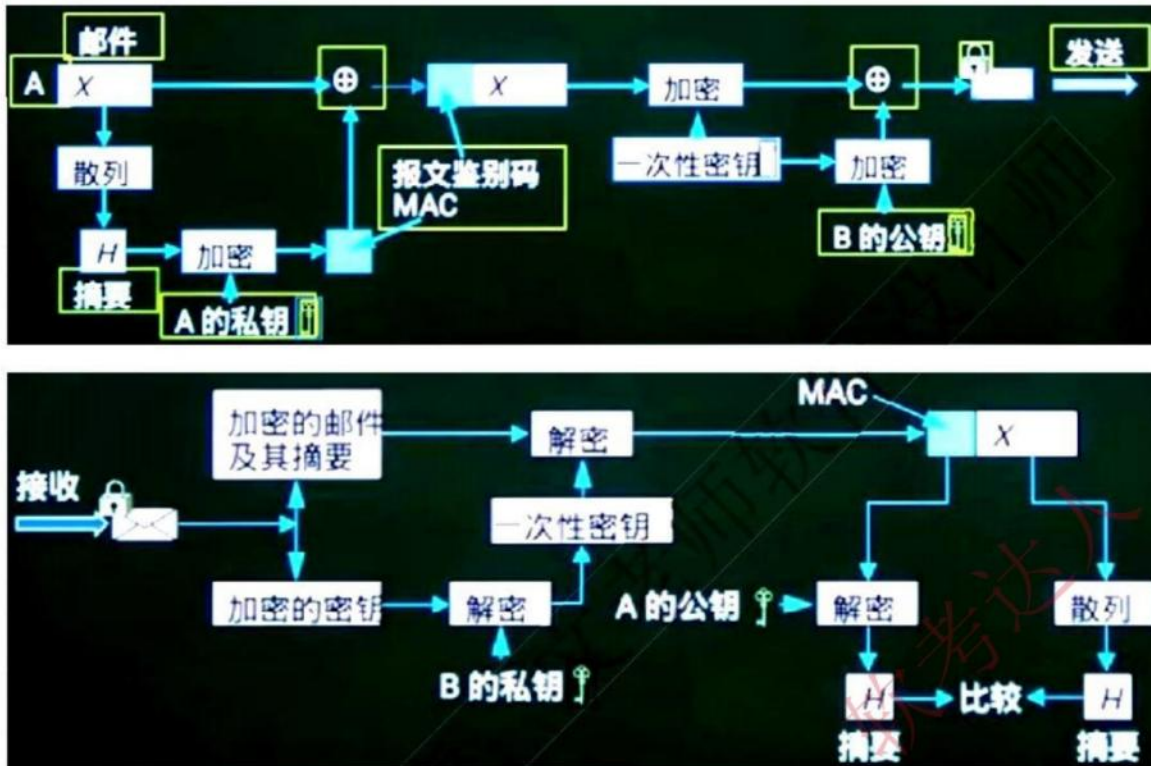
◆ **SET 协议**：安全电子交易协议主要应用于 B2C 模式（电子商务）中保障支付信息的安全性。SET 协议本身比较复杂，设计比较严格，安全性高，它能保证信息传输的机密性、真实性、完整性和不可否认性。SET 协议是 PKI 框架下的一个典型实现，同时也在不断升级和完善，如 SET 2.0 将支持借记卡电子交易。

◆ **Kerberos 协议**：是一种网络身份认证协议，该协议的基础是基于信任第三方，它提供了在开放型网络中进行身份认证的方法，认证实体可以是用户也可以是用户服务。这种认证不依赖宿主机的操作系统或计算机的 IP 地址，不需要保证网络上所有计算机的物理安全性，并且假定数据包在传输中可被随机窃取和篡改。

◆ **PGP 协议**：使用 RSA 公钥证书进行身份认证，使用 IDEA (128 位密钥) 进行数据加密，使用 MD5 进行数据完整性验证。

发送方 A 有三个密钥：A 的私钥、B 的公钥、A 生成的一次性对称密钥；

接收方 B 有两个密钥：B 的私钥、A 的公钥。



六 知识产权和标准化

1 知识产权基础知识

知识产权是指公民、法人、非法人单位对自己的创造性智力成果和其他科技成果依法享有的民事权利。是智力成果的创造人依法享有的权利和在生产经营活动中标记所有人依法所享有的权利的总称。包含著作权、专利权、商标权、商业秘密权、植物新品种权、集成电路布图设计权和地理标志权等。

无体性: 知识产权的对象是没有具体形体，是智力创造成果，是一种抽象的财富。

专有性: 指除权利人同意或法律规定外，权利人以外的任何人不得享有或使用该项权利。

地域性: 指知识产权只在授予其权利的国家或确认其权利的国家产生，并且只能在该国范围内受法律保护，而其他国家则不受保护。

时间性: 仅在法律规定的期限内受到保护，一旦超过期限，权利自行消灭，相关知识产品即成为整个社会的共同财富，为全人类所共同使用。

知识产权具有地域限制，保护期限各种情况如下表所示：

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

知识产权人的确定

（1）职务作品

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

（2）委托作品

单位和委托的区别在于，当合同中未规定著作权的归属时，著作权默认归于单位，而委托创作中，著作权默认归属于创作方个人，具体如下：

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。 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） 同时申请，则根据谁先使用（需提供证据） 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 同时申请则协商归属， 协商未果则驳回所有申请	

侵权判定

相关概念如下：即一般通用化的东西不算侵权，个人未发表的东西被抢先发表是侵权。

中国公民、法人或者其他组织的作品，不论是否发表，都享有著作权。

开发软件所用的思想、处理过程、操作方法或者数学概念不受保护。

著作权法不适用于下列情形：

法律、法规、国家机关的决议、决定、命令和其他具有立法、行政、司法性质的文件，及其官方正式译文；时事新闻；历法、通用数表、通用表格和公式。

只要不进行传播、公开发表、盈利都不算侵权，具体如下：

不侵权	侵权
✓ 个人学习、研究或者欣赏； ✓ 适当引用； ✓ 公开演讲内容 ✓ 用于教学或科学研究 ✓ 复制馆藏作品； ✓ 免费表演他人作品； ✓ 室外公共场所艺术品临摹、绘画、摄影、录像； ✓ 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； ✓ 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； ✓ 未参加创作，在他人作品署名； ✓ 歪曲、篡改他人作品的； ✓ 剽窃他人作品的； ✓ 使用他人作品，未付报酬； ✓ 未经出版者许可，使用其出版的图书、期刊的版式设计的。

2 标准化基础知识

根据标准制定机构和适用范围的不同，可分为国际标准、国家标准、行业标准、区域/地方标准和企业标准；根据类型划分，又可以分为强制性标准和推荐性标准。

(1) 国际标准：是指国际标准化组织（ISO）、国际电工委员会（IEC）和国际电信联盟（ITU）制定的标准，以及国际标准化组织确认并公布的其他国际组织制定的标准。国际标准在世界范围内统一使用，提供各国参考。

(2) 国家标准：是指由国家标准化主管机构制定或批准发布，在全国范围内统一适用的标准。比如：GB——中华人民共和国国家标准；强制性国家标准代号为 GB，推荐性国家标准代号为 GB/T，国家标准指导性文件代号为 GB/Z，国军标代号为 GJB。

ANSI (American National Standards Institute) ——美国国家标准协会标准；

(3) 行业标准：是由某个行业机构、团体等制定的，适用于某个特定行业业务领域的标准。比如：IEEE——美国电气电子工程师学会标准；GA——公共安全标准；YD——通信行业标准；

(4) 区域/地方标准：是由某一区域/地方内的标准化主管机构制定、批准发布的，适用于某个特定区域/地方的标准。比如：EN——欧洲标准；

(5) 企业标准：是企业范围内根据需要协调、统一的技术要求、管理要求和工作要求所制定的标准，适用于本企业内部的标准。一般以 Q 字开头，比如 Q/320101 RER 007—2012，其中 320101 代表地区，RER 代表企业名称代号，001 代表该企业该标准的序号，2012 代表年号。

《中华人民共和国标准化法》规定：企业标准须报当地政府标准化行政主管部门和有关行政主管部门备案。已有国家标准或者行业标准的，国家鼓励企业制定严于国家标准或者行业标准的企业标准，在企业内部适用。

七 程序设计语言基础知识

1 程序设计语言基本概念

解释和编译

都是将高级语言翻译成计算机硬件认可的机器语言加以执行。不同之处在于编译程序生成独立的可执行文件，直接运行，运行时无法控制源程序，效率高。而解释程序不生成可执行文件，可以逐条解释执行，用于调试模式，可以控制源程序，因为还需要控制程序，因此执行速度慢，效率低。

常见程序设计语言

Fortran 语言：科学计算，执行效率高。

Pascal 语言：为教学开发，表达能力强。

C 语言：指针操作能力强，可以开发系统级软件，高效。

C++语言：面向对象，高效。

Java 语言：面向对象，中间代码，跨平台。

C#语言：面向对象，中间代码，.Net 框架。

XML(可扩展标记语言)是标准通用标记语言的子集，是一种用于标记电子文件使其具有结构性的标记语言。

Python 是一种面向对象、解释型计算机程序设计语言。

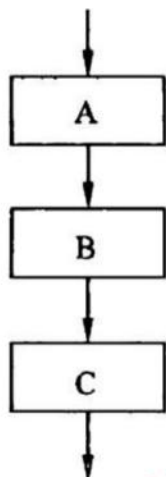
Prolog 是逻辑型程序设计语言。

2 程序设计语言的基本成分

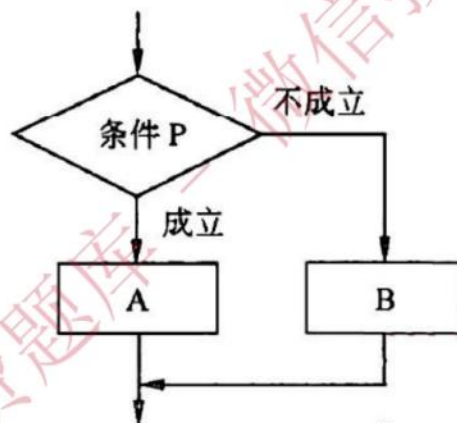
数据成分：指一种程序设计语言的数据和数据类型。数据分为常量（程序运行时不可改变）、变量（程序运行时可以改变）、全局量（存储空间在静态数据区分配）、局部量（存储空间在堆栈区分配）。数据类型有整型、字符型、双精度、单精度浮点型、布尔型等。

运算成分：指明允许使用的运算符及运算规则。包括算术运算、逻辑运算、关系运算、位运算等。

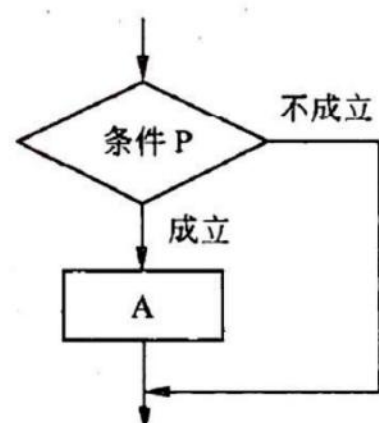
控制成分：指明语言允许表述的控制结构。包括顺序结构、选择结构、循环结构。如下图所示。



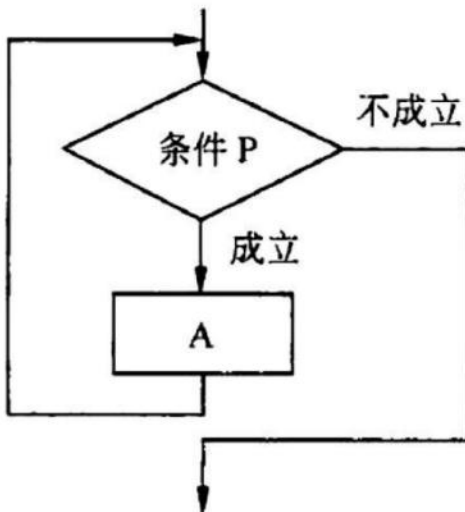
(a) 顺序结构



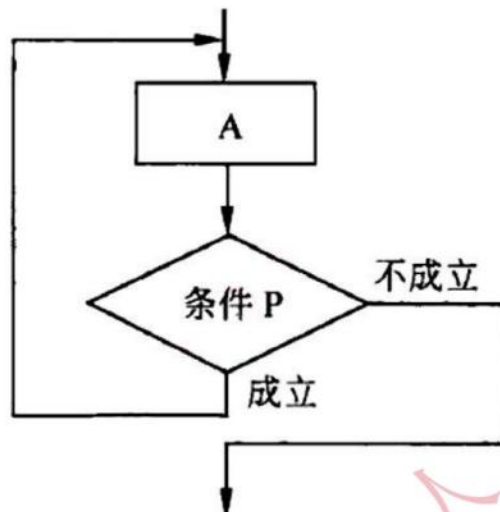
(b) 选择结构



(c) 简化选择结构



(a) while-do 循环结构



(b) do-while 循环结构

传输成分：指明语言允许的数据传输方式。如赋值处理、数据的输入输出等。

3 传值与传址

传值调用：将实参的值传递给形参，形参的改变不会导致调用点所传的实参的值改变。实参可以是合法的变量、常量和表达式。

传址调用：即引用调用，将实参的地址传递给形参，即相当于实参存储单元的地址引用，因此其值改变的同时就改变了实参的值。实参不能为常量，只能是合法的变量和表达式。

因此，在编程时，要改变参数值，就传址，不改变，就传值。

4 编译程序基本原理

编译程序对高级语言源程序进行编译的过程中，要不断收集、记录和使用源程序中一些相关符号的类型和特征等信息，并将其存入符号表中，编译过程如下：

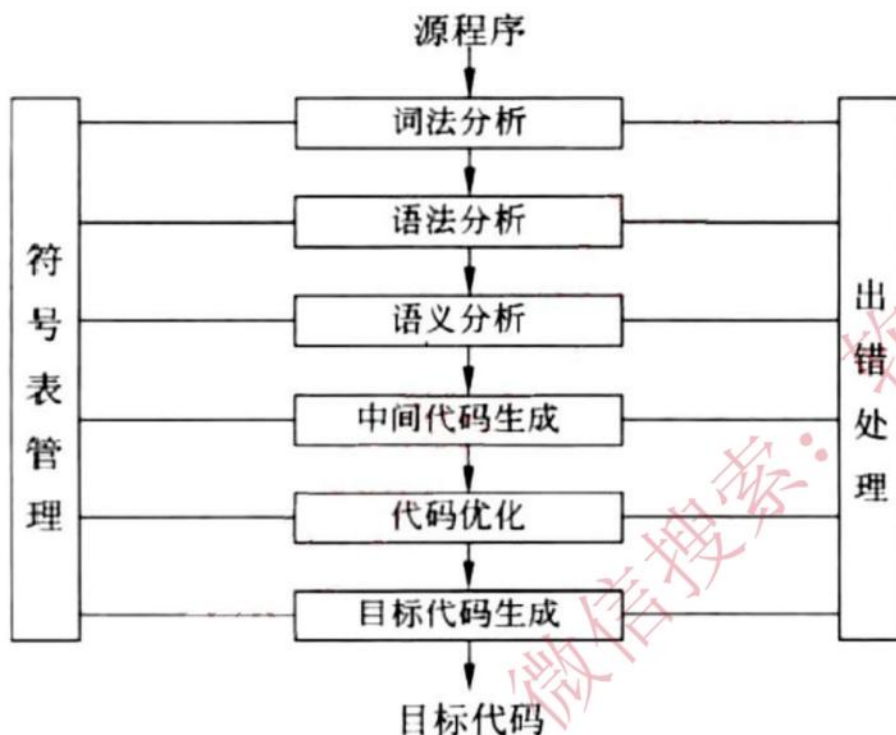
词法分析：是编译过程的第一个阶段。这个阶段的任务是从左到右一个字符一个字符地读入源程序，即对构成源程序的字符流进行扫描然后根据构词规则识别单词(也称单词符号或符号)。

语法分析：是编译过程的一个逻辑阶段。语法分析的任务是在词法分析的基础上将单词序列组合成各类语法短语，如“程序”，“语句”，“表达式”等等。语法分析程序判断源程序在结构上是否正确。源程序的结构由上下文无关文法描述。

语义分析：是编译过程的一个逻辑阶段。语义分析的任务是对结构上正确的源程序进行上下文有关性质的审查，进行类型审查。如类型匹配、除法除数不为0等。又分为静态语义错误（在编译阶段能够查找出来）和动态语义错误（只能在运行时发现）。

中间代码和目标代码：中间代码是根据语义分析产生的，需要经过优化链接，最终生成可执行的目标代码。引入中间代码的目的是进行与机器无关的代码优化处理。常用的中间代码有后缀式（逆波兰式）、三元式（三地址码）、四元式和树等形式。需要考虑三个问题（一是如何生成较短的目标代码；二是如何充分利用计算机中的寄存器，减少目标代码访问存储单元的次数；三是如何充分利用计算机指令系统的特点，以提高目标代码的质量）。

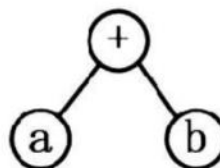
整个编译过程如下图所示：



声明语句，是函数声明、变量定义、宏定义等语句，存入符号表中供查询对应关系，**可执行语句**是逻辑流程代码，需要翻译成中间代码或目标代码。

5 表达式

前缀表达式(+ab)
中缀表达式(a+b)
后缀表达式(ab+)



主要掌握上述三种表达式即可，其实就是树的三种遍历，**一般正常的表达式是中序遍历**，即中缀表达式，根据其构造出树，再按题目要求求出前缀或后缀式。

简单求法：后缀表达式是从左到右开始，先把表达式加上括号，再依次把运算符加到本层次的括号后面。

6 文法定义

定义如下图所示：要求了解四元组每个符号的含义。

一个形式文法是一个有序四元组 $G=(V, T, S, P)$ ，其中：

- 1) V ：非终结符。不是语言组成部分，不是最终结果，可理解为占位符。
- 2) T ：终结符。是语言的组成部分，是最终结果。 $V \cap T = \emptyset$
- 3) S ：起始符。是语言的开始符号。
- 4) P ：产生式。用终结符替代非终结符的规则。形如 $\alpha \rightarrow \beta$

终结符：最终结果，不能推导出其他元素。

非终结符：能够推导出其他元素。

产生式：即非终结符推导出终结符的公式。

闭包，概念如下图，一般考察闭包可以为 0 个的情况代入运算：

正则闭包： $A^+ = A^1 \cup A^2 \cup A^3 \cup \dots \cup A^n \cup \dots$ （也就是所有幂的组合）。

闭包： $A^* = A^0 \cup A^+$ （在正则闭包的基础上，加上 $A^0 = \{\epsilon\}$ ）。

例如 $a^* = \{a, aa, aaa, \dots, \epsilon\}$ ，而 $(ab)^* = \{ab, abab, ababab, \dots, \epsilon\}$

文法类型如下：

类 型	别 称	说 明	对应自动机
<u>0型</u>	短语文法	G 的每条产生式 $\alpha \rightarrow \beta$ 满足 a 属于 V 的正则闭包且至少含有一个非终结符，而 β 属于 V 的闭包	图灵机
<u>1型</u>	上下文有关文法	G 的任何产生式 $\alpha \rightarrow \beta$ 满足 $ \alpha \leq \beta $ ，仅仅 $S \rightarrow \epsilon$ 例外，但 S 不得出现在任何产生式右部	线性界限自动机
<u>2型</u>	上下文无关文法	G 的任何产生式为 $A \rightarrow \beta$ ， A 为非终结符， β 为 V 的闭包	非确定的下推自动机
<u>3型</u>	正规文法	G 的任何产生式为 $A \rightarrow \alpha B$ 或 $A \rightarrow \alpha$ ， α 属于非终结符的闭包， A, B 都属于非终结符	有限自动机

程序设计语言中的大部分语法都是上下文无关文法，当然语义上是相关的，要注意区分语法和语义。

7 正规式

正规式和正规集：



对于字母表 Σ ，其上的正规式及其表示的正规集可以递归定义如下。

- (1) ε 是一个正规式，它表示集合 $L(\varepsilon) = \{\varepsilon\}$ 。
- (2) 若 a 是 Σ 上的字符，则 a 是一个正规式，它所表示的正规集为 $\{a\}$ 。
- (3) 若正规式 r 和 s 分别表示正规集 $L(r)$ 和 $L(s)$ ，则：
 - ① $r|s$ 是正规式，表示集合 $L(r) \cup L(s)$ 。
 - ② $r \cdot s$ 是正规式，表示集合 $L(r)L(s)$ 。
 - ③ r^* 是正规式，表示集合 $(L(r))^*$ 。
 - ④ (r) 是正规式，表示集合 $L(r)$ 。

示例如下：

正 规 式	正 规 集
ab	字符串 ab 构成的集合
$a b$	字符串 a 、 b 构成的集合
a^*	由 0 个或多个 a 构成的字符串集合
$(a b)^*$	所有字符 a 和 b 构成的串的集合
$a(a b)^*$	以 a 为首字符的 a 、 b 字符串的集合
$(a b)^*abb$	以 abb 结尾的 a 、 b 字符串的集合

8 有限自动机

是进行词法分析的工具，其定义如下：

$M = (S, \Sigma, \delta, S_0, Z)$

- 1) S 是一个有限集，每个元素为一个状态
- 2) Σ 是一个有穷字母表，每个元素为一个输入字符
- 3) δ 是转换函数：是一个单值对照
- 4) S_0 属于 S ，是其唯一的初态
- 5) Z 是一个终态集（可空）

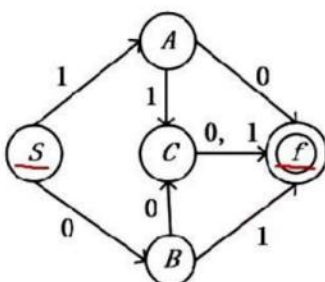
有限状态自动机可以形象地用状态转换图表示，设有限状态自动机：

$DFA = (\{S, A, B, C, f\}, \{1, 0\}, \delta, S, \{f\})$,

其中：

$\delta(S, 0) = B, \delta(S, 1) = A, \delta(A, 0) = f, \delta(A, 1) = C, \delta(B, 0) = C, \delta(B, 1) = f, \delta(C, 0) = f, \delta(C, 1) = f$

其状态转换图示例如下：



主要有五个符号集，由上图示例，可知用状态来表示十分清晰，由 S 输入一个 0，可得出 B，依次类推，一般考试，给出一个状态图，问能否构造出 001 这样的字符串，解决方法就是从起始 S 到终点 f 之间是否有一条路，权值为 001。本质就是有向图从起点到终点的遍历。

确定的有限自动机和不确定的有限自动机：输入一个字符，看是否能得出唯一的后继，若能，则是确定的，否则若得出多个后继，则是不确定的。

9 语法分析方法

自上而下语法分析：最左推导，从左至右。给定文法 G 和源程序串 r。从 G 的开始符号 S 出发，通过反复使用产生式对句型中的非终结符进行替换(推导)，逐步推导出 r。

递归下降思想：原理是利用函数之间的递归调用模拟语法树自上而下的构造过程，是一种自上而下的语法分析方法。

自下而上语法分析：最右推导，从右至左。从给定的输入串 r 开始，不断寻找子串与文法 G 中某个产生式 P 的候选式进行匹配，并用 P 的左部代替(归约)之，逐步归约到开始符号 S。

移进-规约思想：设置一个栈，将输入符号逐个移进栈中，栈顶形成某产生式的右部时，就用左部去代替，称为归约。很明显，这个思想是通过右部来推导出左部，因此是自下而上语法分析的核心思想。

八 数据结构

1 线性结构

1.1 线性表

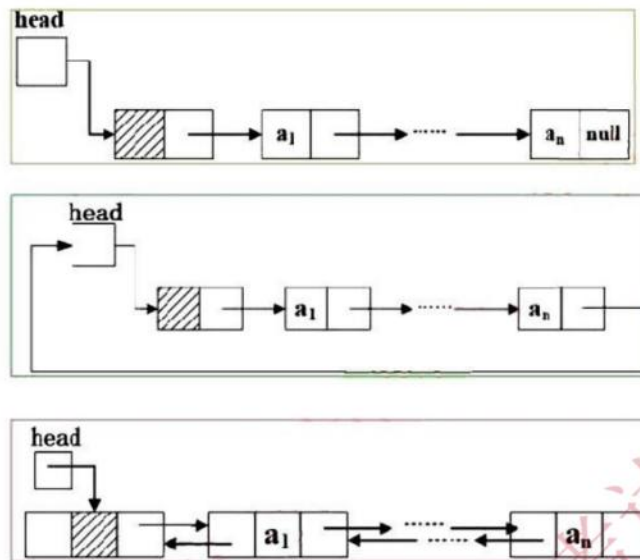
线性表是线性结构（每个元素最多只有一个出度和一个入度，表现为一条线状）的代表，线性表按存储方式分为顺序表和链表，存储形式如下图：

顺序表

1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---

链表

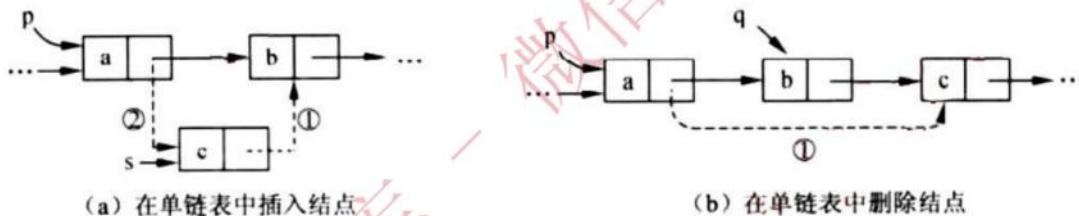
单链表
循环链表
双向链表



上图中，顺序表是需要一段连续的内存空间来存放顺序表中的所有元素，这些元素在物理地址上是相邻的。

而链表中的所有元素只是逻辑上相邻，在实际物理存储时处于不同的空闲块中，元素之间通过指针域连接，如上图所示，又可分为单链表、循环链表、双向链表，具体逻辑上的关系都如上图所示。

单链表的删除和插入



(a) 在单链表中插入结点

(b) 在单链表中删除结点

在上图中 p 所指向的节点后插入 s 所指向的节点，操作为：

$s \rightarrow \text{next} = p \rightarrow \text{next};$

$p \rightarrow \text{next} = s;$

同理，在单链表中删除 p 所指向节点的后继节点 q 时，操作为：

$q = p \rightarrow \text{next};$

$p \rightarrow \text{next} = p \rightarrow \text{next} \rightarrow \text{next};$

$\text{free}(p);$

双向链表的插入和删除（几乎不考，删除）

顺序存储和链式存储的对比如下图所示：

性能类别	具体项目	顺序存储	链式存储
空间性能	存储密度	=1, 更优	<1
	容量分配	事先确定	动态改变, 更优
时间性能	查找运算	$O(n/2)$	$O(n/2)$
	读运算	$O(1)$, 更优	$O([n+1]/2)$, 最好情况为1, 最坏情况为n
	插入运算	$O(n/2)$, 最好情况为0, 最坏情况为n	$O(1)$, 更优
	删除运算	$O([n-1]/2)$	$O(1)$, 更优

在空间方面, 因为链表还需要存储指针, 因此有空间浪费存在。

在时间方面, 由顺序表和链表的存储方式可知, 当需要对元素进行破坏性操作(插入、删除)时, 链表效率更高, 因为其只需要修改指针指向即可, 而顺序表因为地址是连续的, 当删除或插入一个元素后, 后面的其他节点位置都需要变动。

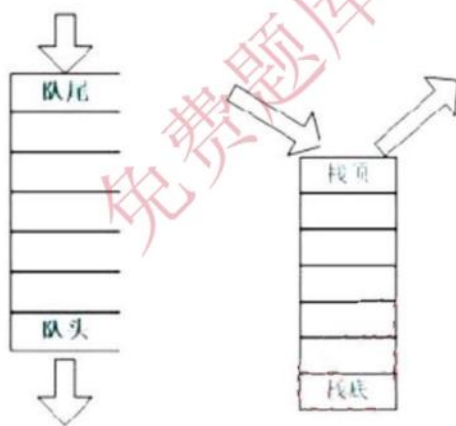
而当需要对元素进行不改变结构操作时(读取、查找), 顺序表效率更高, 因为其物理地址是连续的, 如同数组一般, 只需按索引号就可快速定位, 而链表需要从头节点开始, 一个个的查找下去。

1.2 栈和队列

栈和队列

队列、栈结构如下图, 队列是先进先出, 分队头和队尾;

栈是先进后出, 只有栈顶能进出。

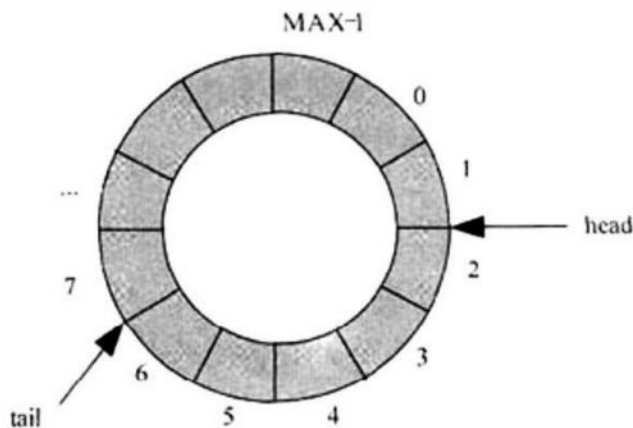


先进先出

先进后出

环形队列

环形队列是一个首尾相连的 FIFO 数据结构，采用数组存储，到达尾部时将转回到 0 位置，该转回是通过取模操作来实现的。因此环形队列逻辑上是将数组元素 $q[0]$ 与 $q[\text{MAX}-1]$ 连接，形成一个存放队列的环形空间。为了方便读写，还要用数组下标来指明队列的读写位置，其中 **head** 指向可以读的位置，**tail** 指向可以写的位置，环形队列如图所示。



环形队列是在实际编程极为有用的数据结构，它有如下特点：它是一个首尾相连的 FIFO 的数据结构，采用数组的线性空间；数据组织简单，能很快知道队列是否满或空；能以很快的速度来存取数据。因为简单高效，甚至在硬件都实现了环形队列。

内存上没有环形的结构，因此环形队列实际上是数组的线性空间来实现。那当数据到了尾部如何处理呢？它将转回到 0 位置来处理。这个转回是通过数组下标索引取模操作($\text{Index} \% \text{MAXN}$)来实现的。

环形队列的关键是判断队列为空，还是为满。当 **tail** 追上 **head** 时，队列为满；当 **head** 追上 **tail** 时，队列为空。但如何知道谁追上谁，还需要一些辅助的手段来判断。

如何判断环形队列为空、为满有两种判断方法。

一是附加一个标志位 **tag**，当 **head** 赶上 **tail**，队列空，则令 $\text{tag}=0$ ，当 **tail** 赶上 **head**，队列满，则令 $\text{tag}=1$ ；

二是限制 **tail** 赶上 **head**，即队尾结点与队首结点之间至少留有一个元素的空间。队列空时 $\text{head}==\text{tail}$ ，队列满时 $(\text{tail}+1)\% \text{MAXN}=\text{head}$ 。

设环形队列数据结构定义如下：

```
typedef struct ringg
{
    int head; /*头部, 出队列方向*/
    int tail; /*尾部, 入队列方向*/
    int tag;
    int size; /*队列总尺寸*/
    int space[RINGQ_MAX]; /*队列空间*/
}RINGQ;
RINGQ *q;
```

入队操作时，如队列不满，则写入

```
q->tail=(q->tail+1)%q->size;
```

出队操作时，如果队列不空，则从 head 处读出。下一个可读的位置在

```
q->head=(q->head+1)%q->size;
```

1.3 串

字符串是一种特殊的线性表，其数据元素都为字符。

空串：长度为 0 的字符串，没有任何字符。

空格串：由一个或多个空格组成的串，空格是空白字符，占一个字符长度。

子串：串中任意长度的连续字符构成的序列称为子串。含有子串的串称为主串，空串是任意串的子串。

串的模式匹配算法：子串的定位操作，用于查找子串在主串中第一次出现的位置的算法。

模式匹配算法

基本的模式匹配算法：也称为布鲁特福斯算法，其基本思想是从主串的第 1 个字符起与模式串的第 1 个字符比较，若相等，则继续逐个字符进行后续的比较；否则从主串中的第 2 个字符起与模式串的第 1 个字符重新比较，直至模式串中每个字符依次和主串中的一个连续的字符序列相等时为止，此时称为匹配成功，否则称为匹配失败。

KMP 算法：对基本模式匹配算法的改进，其改进之处在于：每当匹配过程中出现相比较的字符不相等时，不需要回溯主串的字符位置指针，而是利用已经得到的“部分匹配”结果将模式串向右“滑动”尽可能远的距离，再继续进行比较。

2 数组、矩阵和广义表

2.1 数组

主要掌握数组存储地址的计算，特别是二维数组，要注意理解，假设每个数组元素占用存储长度为 len，起始地址为 a，存储地址计算如下：

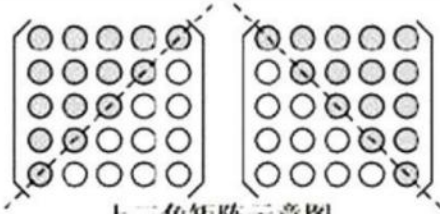
数组类型	存储地址计算
一维数组a[n]	a[i]的存储地址为: $a + i * len$
二维数组a[m][n]	a[i][j]的存储地址 (按行存储) 为: $a + (i * n + j) * len$ a[i][j]的存储地址 (按列存储) 为: $a + (j * m + i) * len$

由二维数组公式可知, 若 $i=j$, 则无论按行存储还是按列存储, 存储在 $a[i,j]$ 之前的元素个数相同, 实际答题时可构造一个简单的二维数组, 取特殊值验证。

另外, 注意, 若题目没有特别说明 (注意审题), 数组元素一般都是从 0 开始计数的, 因此上图中公式都是无需-1 的。要注意区分二维数组行列计算。

2.2 稀疏矩阵

稀疏矩阵就是一个矩阵中大部分数据都是 0 的矩阵, 此时, 只需要存储少部分不为 0 的有效数据即可节省大量空间, 因此有下述两种稀疏矩阵:

矩阵	特点	存储数组 B[k] 与矩阵元素 a_{ij} 对应公式; 或示意图
对称矩阵	矩阵 $A_{n \times n}$ 中的元素特点为 $a_{ij}=a_{ji}(1 \leq i, j \leq n)$	$k = \begin{cases} \frac{i(i-1)}{2} + j & \text{当 } i \geq j \\ \frac{j(j-1)}{2} + i & \text{当 } i < j \end{cases}$
对角矩阵	矩阵中的非零元素都集中在以主对角线为中心的带状区域	三对角矩阵示意图: $A_{n \times n} = \begin{bmatrix} a_{1,1} & a_{1,2} & & & & \\ a_{2,1} & a_{2,2} & a_{2,3} & & & 0 \\ & a_{3,2} & a_{3,3} & a_{3,4} & & \\ & & \vdots & \vdots & \vdots & \\ & & & a_{i,i-1} & a_{i,i} & a_{i,i+1} \\ & & & & \vdots & \vdots \\ & & & & & a_{n,n-1} & a_{n,n} \end{bmatrix}$
三角矩阵	矩阵中主对角线上方都为非零元素, 下方都为零元素 (或相反)。	 上三角矩阵示意图

它们和一维数组之间的对应关系如上公式所示，但在实际考试中，因为都是选择题，可以使用**代入法（特殊值法）**进行排除，具体见下例和其解析：

2.3 广义表

广义表是线性表的推广，是由 0 个或多个单元素或子表组成的有限序列。

广义表与线性表的区别：线性表的元素都是结构上不可分的单元素，而广义表的元素既可以单元素，也可以是有结构的表。

广义表一般记为： $LS = (\alpha_1, \alpha_2, \dots, \alpha_n)$

其中 LS 是表名， α_i 是表元素，它可以是表（称为子表），也可以是数据元素（称为原子）。其中 n 是广义表的**长度**（也就是最外层包含的元素个数）， $n=0$ 的广义表为空表；而递归定义的重数就是广义表的**深度**，即定义中所含括号的重数（单边括号的个数，原子的深度为 0，空表的深度为 1）。

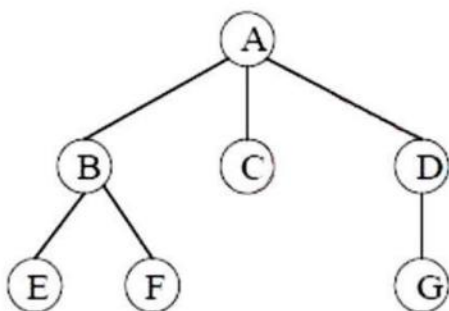
head()和 **tail()**：**取表头**（广义表第一个表元素，可以是子表也可以是单元素）和**取表尾**（广义表中，除了第一个表元素之外的其他所有表元素构成的表，非空广义表的表尾必定是一个表，即使表尾是单元素）操作。

3 树与二叉树

3.1 树的相关概念

树结构是一种非线性结构，树中的每一个数据元素可以有两个或两个以上的直接后继元素，用来描述层次结构关系。

树是 n 个节点的有限集合 ($n \geq 0$)，当 $n=0$ 时称为空树，在任一非空树中，有且仅有一个根节点。其余结点可分为 $m(m \geq 0)$ 个互不相交的有限子集 $T_1, T_2, \dots, T_m$ ，其中，每个 T_i 又都是一棵树，并且称为根结点的子树。



树的基本概念如下：

(1) 双亲、孩子和兄弟。结点的子树的根称为该结点的孩子；相应地，该结点称为其子结点的双亲。具有相同双亲的结点互为兄弟。

(2) 结点的度。一个结点的子树的个数记为该结点的度。例如 A 的度为 3，B 的度为 2，C 的度为 0，D 的度为 1。

(3) 叶子结点。叶子结点也称为终端结点，指度为 0 的结点。例如，E、F、C、G 都是叶子结点。

(4) 内部结点。度不为 0 的结点，也称为分支结点或非终端结点。除根结点以外，分支结点也称为内部结点。例如，B、D 都是内部结点。

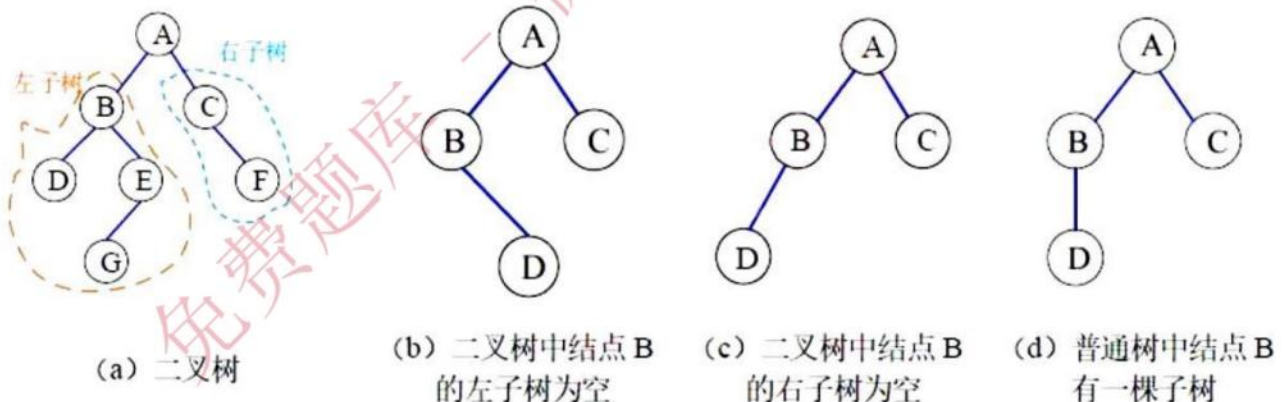
(5) 结点的层次。根为第一层，根的孩子为第二层，依此类推，若某结点在第 i 层，则其孩子结点在第 $i+1$ 层。例如，A 在第 1 层，B、C、D 在第 2 层，E、F 和 G 在第 3 层。

(6) 树的高度。一棵树的最大层数记为树的高度（或深度）。例如，图中所示树的高度为 3。

(7) 有序（无序）树。若将树中结点的各子树看成是从左到右具有次序的，即不能交换，则称该树为有序树，否则称为无序树。

3.2 二叉树的特性

二叉树是 n 个节点的有限集合，它或者是空树，或者是由一个根节点及两颗互不相交的且分别称为左、右子树的二叉树所组成。与树的区别在于每个根节点最多只有两个孩子结点。



二叉树有一些性质如下，要求掌握，在实际考试中可以用**特殊值法**验证。

(1) 二叉树第 i 层 ($i \geq 1$) 上至多有 2^{i-1} 个节点。

(2) 深度为 k 的二叉树至多有 $2^k - 1$ 个节点 ($k \geq 1$)。

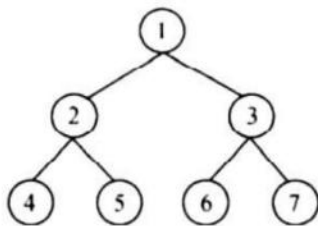
(3) 对任何一棵二叉树，若其终端节点数为 n_0 ，度为 2 的节点数为 n_2 ，则 $n_0 = n_2 + 1$ 。

此公式可以画一个简单的二叉树使用特殊值法快速验证，也可以证明如下：设一棵二叉树上叶结点数为 n_0 ，单分支结点数为 n_1 ，双分支结点数为 n_2 ，则总结点数 $= n_0 + n_1 + n_2$ 。在一棵二叉树中，所有

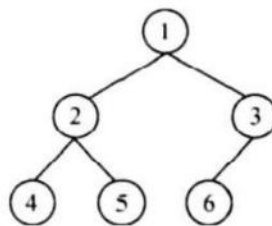
结点的分支数(即度数)应等于单分支结点数加上双分支结点数的 2 倍, 即总的分支数 $=n_1+2n_2$ 。由于二叉树中除根结点以外, 每个结点都有唯一的一个分支指向它, 因此二叉树中: 总的分支数=总结点数-1。

(4) 具有 n 个节点的完全二叉树的深度为 $\lfloor \log_2 n \rfloor + 1$ 。

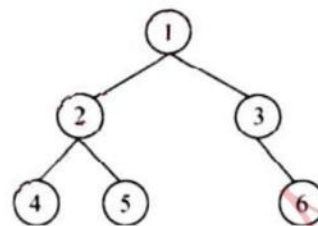
两种特殊的二叉树如下图所示:



(a) 满二叉树



(b) 完全二叉树



(c) 非完全二叉树

可知, 满二叉树每层都是满的。

完全二叉树的 $k-1$ 层是满的, 第 k 层节点从左到右是满的。

3.3 二叉树的顺序存储结构

顺序存储, 就是用一组连续的存储单元存储二叉树中的节点, 按照从上到下, 从左到右的顺序依次存储每个节点。

对于深度为 k 的完全二叉树, 除第 k 层外, 其余每层中节点数都是上一层的两倍, 由此, 从一个节点的编号可推知其双亲、左孩子、右孩子结点的编号。假设有编号为 i 的节点, 则有:

若 $i=1$, 则该节点为根节点, 无双亲; 若 $i>1$, 则该节点的双亲节点为 $\lfloor i/2 \rfloor$ 。

若 $2i \leq n$, 则该节点的左孩子编号为 $2i$, 否则无左孩子。

若 $2i+1 \leq n$, 则该节点的右孩子编号为 $2i+1$, 否则无右孩子。

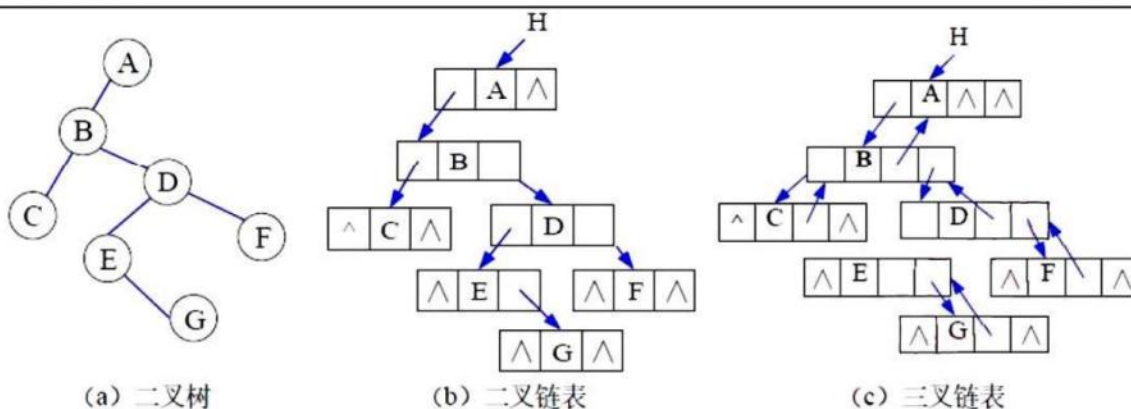
显然, 顺序存储结构对完全二叉树而言既简单又节省空间, 而对于一般二叉树则不适用。

因为在顺序存储结构中, 以节点在存储单元中的位置来表示节点之间的关系, 那么对于一般的二叉树来说, 也必须按照完全二叉树的形式存储, 也就是要添上一些实际并不存在的“虚节点”, 这将造成空间的浪费。

二叉树的链式存储结构

一般用二叉链表来存储二叉树节点, 二叉链表中除了该节点本身的数据外, 还存储有左孩子结点的指针、右孩子结点的指针, 即一个数据+两个指针。

每个二叉链表节点存储一个二叉树节点, 头指针则指向根节点。



3.4 二叉树的遍历

一颗非空的二叉树由根节点、左子树、右子树三部分组成，遍历这三部分，也就遍历了整颗二叉树。这三部分遍历的基本顺序是先左子树后右子树，但根节点顺序可变，以根节点访问的顺序为准有下列三种遍历方式：

先序（前序）遍历：根左右。

中序遍历：左根右。

后序遍历：左右根。

还有层次遍历方法：

层次遍历：按层次，从上到下，从左到右。

反向构造二叉树：

仅仅有前序和后序是无法构造二叉树的，必须要是和中序遍历的集合才能反向构造出二叉树。

构造时，前序和后序遍历可以确定根节点，中序遍历用来确定根节点的左子树节点和右子树节点，而后按此方法进行递归，直至得出结果。

3.5 线索二叉树

引入线索二叉树是为了保存二叉树遍历时某节点的前驱节点和后继节点的信息，二叉树的链式存储只能获取到某节点的左孩子和右孩子结点，无法获取其遍历时的前驱和后继节点，因此可以在链式存储中再增加两个指针域，使其分别指向前驱和后继节点，但这样太浪费存储空间，考虑下述实现方法：

若 n 个节点的二叉树使用二叉链表存储，则必然有 $n+1$ 个空指针域，利用这些空指针域来存放节点的前驱和后继节点信息，为此，需要增加两个标志，以区分指针域存放的到底是孩子结点还是遍历节点，如下：

ltag	lchild	data	rchild	rtag
------	--------	------	--------	------

ltag = $\begin{cases} 0 & \text{lchild 域指示结点的左孩子} \\ 1 & \text{lchild 域指示结点的直接前驱} \end{cases}$

rtag = $\begin{cases} 0 & \text{rchild 域指示结点的右孩子} \\ \bullet & \\ 1 & \text{rchild 域指示结点的直接后继} \end{cases}$

若二叉树的二叉链表采用上述结构，则称为线索链表，其中指向前驱、后继节点的指针称为线索，加上线索的二叉树称为线索二叉树。

3.6 最优二叉树（哈夫曼树）

最优二叉树又称为哈夫曼树，是一类带权路径长度最短的树，相关概念如下：

树的路径长度：根节点到达叶子节点的层次距离（或者说经过的分支数）。

权：叶子节点代表的值。

带权路径长度：叶子节点的路径长度乘以该叶节点的值。

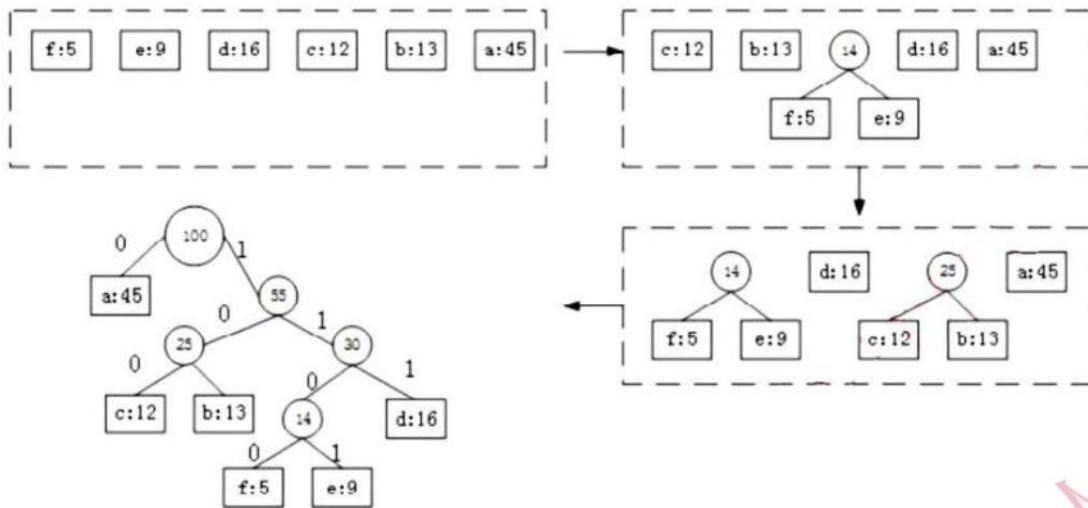
树的带权路径长度（树的代价）：树的所有叶子节点的带权路径长度之和。

哈夫曼树的求法：给出一组权值，将其中两个最小的权值作为叶子节点，其和作为父节点，组成二叉树，而后删除这两个叶子节点权值，并将父节点的值添加到该组权值中。重复进行上述步骤，直至所有权值都被使用完。

构造出的哈夫曼树中，所有初始给出的权值都作为了叶子节点，此时，求出每个叶子节点的带权路径长度，而后相加，就是树的带权路径长度，这个长度是最小的。

若需要构造哈夫曼编码（要保证左节点值小于右节点的值，才是标准的哈夫曼树），将标准哈夫曼树的左分支设为 0，右分支设为 1，写出每个叶节点的编码，会发现，哈夫曼编码前缀不同，因此不会混淆，同时也是最优编码。

构造过程如下：



3.7 树和森林

树的存储结构

双亲表示法：用一组连续的地址单元存储树的节点，并在每个节点中附带一个指示器，指出其双亲节点所在数组元素的下标。

孩子表示法：在存储结构中用指针指示出节点的每个孩子，为树中每个节点的孩子建立一个链表。

孩子兄弟表示法：又称为二叉链表表示法，为每个存储节点设置两个指针域，分别指向该节点的第一个孩子和下一个兄弟节点。

树和森林的遍历

由于树中每个节点可能有多个子树，因此遍历树的方法有两种：

(1) 树的先根遍历。树的先根遍历是先访问树的根结点，然后依次先根遍历根的各棵子树。对树的先根遍历等同于对转换所得的二叉树进行先序遍历。

(2) 树的后根遍历。树的后根遍历是先依次后根遍历树根的各棵子树，然后访问树根结点。树的后根遍历等同于对转换所得的二叉树进行中序遍历。

按照森林和树的相互递归定义，可以得出森林的两种遍历方法。

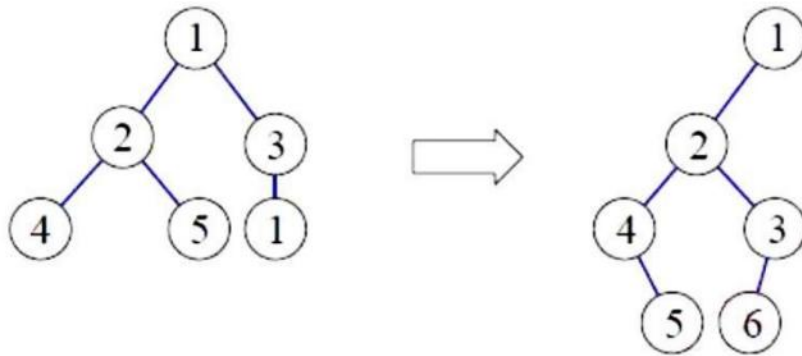
(1) 先序遍历森林。若森林非空，首先访问森林中第一棵树的根结点，然后先序遍历第一棵树的根结点的子树森林，最后先序遍历除第一棵树之外剩余的树所构成的森林。

(2) 中序遍历森林。若森林非空，首先中序遍历森林中第一棵树的子树森林，然后访问第一棵树的根结点，最后中序遍历除第一棵树之外剩余的树所构成的森林。

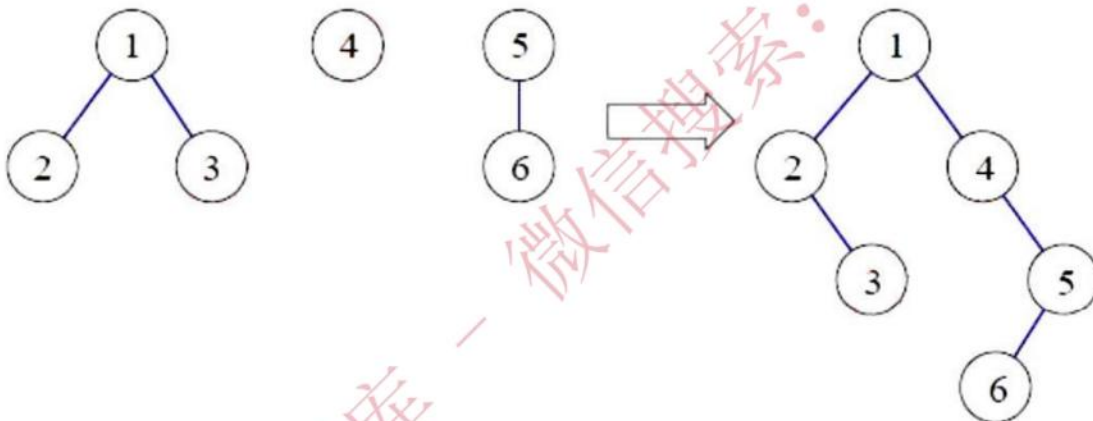
树、森林和二叉树的转换

规则是：树的最左边节点作为二叉树的左子树，树的其他兄弟节点作为二叉树的右子树节点。

示例如下图：采用连线法，将最左边节点和其兄弟节点都连接起来，而原来的父节点和兄弟节点的连线则断开，这种方法最简单，要求掌握。



由于树根没有兄弟，所以树转换为二叉树后，二叉树的根一定没有右子树。这样，将一个森林转换为一棵二叉树的方法是：先将森林中的每一棵树转换为二叉树，再将第一棵树的根作为转换后的二叉树的根，第一棵树的左子树作为转换后二叉树根的左子树，第二棵树作为转换后二叉树的右子树，第三棵树作为转换后二叉树根的右子树的右子树，依此类推，森林就可以转换为一棵二叉树，



3.8 查找（排序）二叉树

查找二叉树上的每个节点都存储一个值，且每个节点的**所有左孩子结点值都小于父节点值**，而**所有右孩子结点值都大于父节点值**，是一个有规律排列的二叉树，这种数据结构可以方便查找、插入等数据操作。

二叉排序树的**查找效率**取决于二叉排序树的深度，二叉树深度越深，则效率最差，对于结点个数相同的二叉排序树，其深度排序为 满二叉树<完全二叉树<平衡二叉树<单枝树，因此效率最低的是最深的单枝树。

3.9 平衡二叉树

前面讲过查找(排序)二叉树,特点是所有左子树值小于根节点值,所有右子树值大于根节点值,而这个特点可以构造出多个不同的二叉树,并不唯一,因此提出平衡二叉树的概念,在查找二叉树特点的基础上,要求每个节点的平衡度只能为 0 或 1 或 -1.

节点的左右子树深度就是其左右子树各自的层数,而后将左子树深度减去右子树深度,就得到了该节点的平衡度。因此,平衡二叉树就是任意左右子树层次相差不超过 1。

4 图

4.1 图的基本概念

图也是一种非线性结构,图中任意两个节点间都可能直接有直接关系。相关定义如下:

无向图:图的结点之间连接线是没有箭头的,不分方向。

有向图:图的结点之间连接线是箭头,区分 A 到 B,和 B 到 A 是两条线。

完全图:无向完全图中,节点两两之间都有连线, n 个结点的连线数为 $(n-1)+(n-2)+\cdots+1=n*(n-1)/2$;

有向完全图中,节点两两之间都有互通的两个箭头, n 个节点的连线数为 $n*(n-1)$ 。

度、出度和入度:顶点的度是关联与该顶点的边的数目。在有向图中,顶点的度为出度和入度之和。出度是以该顶点为起点的有向边的数目。入度是以该顶点为终点的有向边的数目。

路径:存在一条通路,可以从一个顶点到达另一个顶点,有向图的路径也是有方向的。

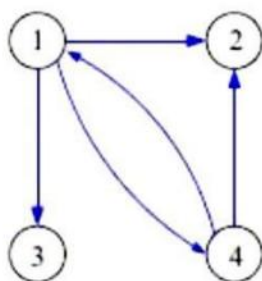
连通图和连通分量:针对无向图。若从顶点 v 到顶点 u 之间是有路径的,则说明 v 和 u 之间是连通的,若无向图中任意两个顶点之间都是连通的,则称为连通图。无向图 G 的极大连通子图称为其连通分量。

强连通图和强连通分量:针对有向图。若有向图任意两个顶点间都互相存在路径,即存在 v 到 u ,也存在 u 到 v 的路径,则称为强连通图。有向图中的极大连通子图称为其强连通分量。

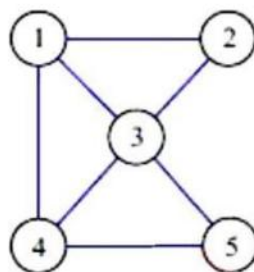
网:边带权值的图称为网。

4.2 图的存储

邻接矩阵:假设一个图中有 n 个节点,则使用 n 阶矩阵来存储这个图中各节点的关系,规则是若节点 i 到节点 j 有连线,则矩阵 $R_{i,j}=1$, 否则为 0, 示例如下图所示:



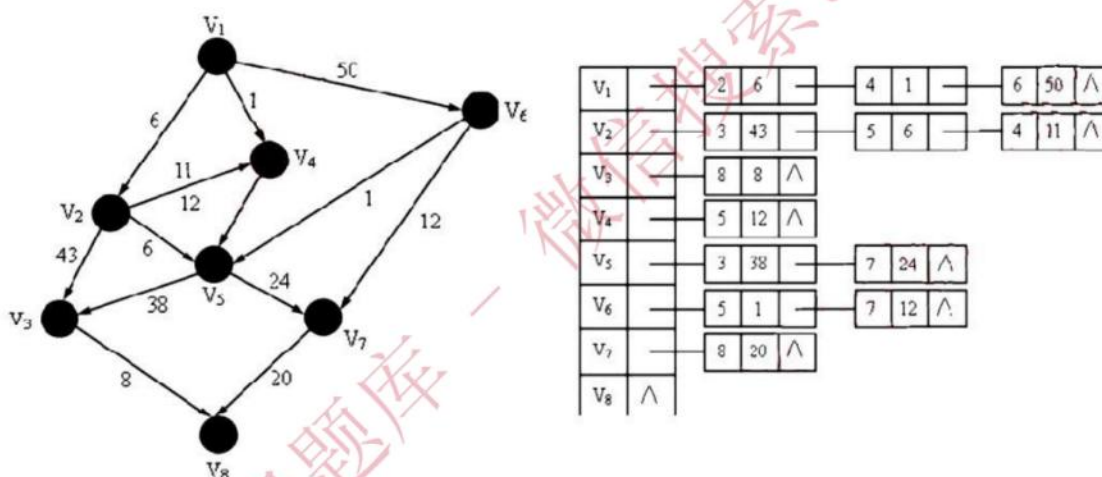
$$A = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix}$$



$$B = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

由上图可知，如果是一个无向图，肯定是沿对角线对称的，只需要存储上三角或者下三角就可以了，而有向图则不一定对称。

邻接链表：用到了两个数据结构，先用一个一维数组将图中所有顶点存储起来，而后，对此一维数组的每个顶点元素，使用链表挂上和其有连线关系的结点的编号和权值，示例如下图所示：



存储特点：图中的顶点数决定了邻接矩阵的阶和邻接表中的单链表数目，无论是对有向图还是无向图，边数的多少决定了单链表中的结点数，而不影响邻接矩阵的规模，因此采用何种存储方式与有向图、无向图没有区别，要看图的边数和顶点数，完全图适合采用邻接矩阵存储。

4.3 图的遍历

图的遍历是指从图的任意节点出发，沿着某条搜索路径对图中所有节点进行访问且只访问一次，分为以下两种方式：

深度优先遍历：从任一顶点出发，遍历到底，直至返回，再选取任一其他节点出发，重复这个过程直至遍历完整个图；

广度优先遍历:先访问完一个顶点的所有邻接顶点,而后再依次访问其邻接顶点的所有邻接顶点,类似于层次遍历。

在实际应用中,一般给出邻接表或者邻接矩阵,来求遍历,这时,可以先画出图,再求最保险,当然简单的结构也可以利用存储结构特点来求。

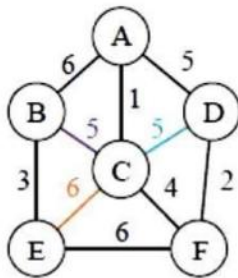
遍历方法	说明	示例	图例
深度优先	1.首先访问出发顶点V 2.依次从V出发搜索V的任意一个邻接点W; 3.若W未访问过,则从该点出发继续深度优先遍历 它类似于树的前序遍历。	$V_1, V_2,$ $V_4, V_8,$ $V_5, V_3,$ V_6, V_7	
广度优先	1.首先访问出发顶点V 2.然后访问与顶点V邻接的全部未访问顶点W、X、Y... 3.然后再依次访问W、X、Y...邻接的未访问的顶点;	$V_1, V_2,$ $V_3, V_4,$ $V_5, V_6,$ V_7, V_8	

4.4 图的最小生成树

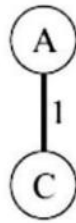
假设有 n 个节点,那么这个图的最小生成树有 $n-1$ 条边(不会形成环路,是树非图),这 $n-1$ 条边应该会将所有顶点都连接成一个树,并且**这些边的权值之和最小**,因此称为最小生成树。

因此,本质就是找出图中权值最小的 $n-1$ 条边,使图中所有节点连接成一棵树,共有下列两种算法:

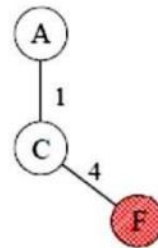
普里姆算法:从任意顶点出发,找出与其邻接的边权值最小的,此时此边的另外一个顶点自动加入树集合中,而后再从这这个树集合的所有顶点中找出与其邻接的边权值最小的,同样此边的另外一个顶点加入树集合中,依次递归,直至图中所有顶点都加入树集合中,此时此树就是该图的最小生成树。普里姆算法的时间复杂度为 $O(n^2)$,与图中的边数无关,因此该算法适合于求边稠密的网的最小生成树。



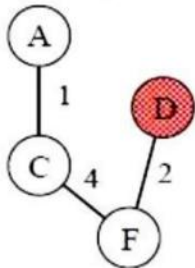
(a)



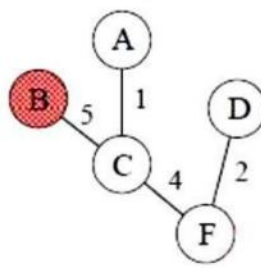
(b)



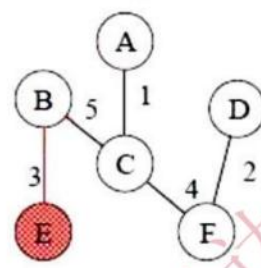
(c)



(d)

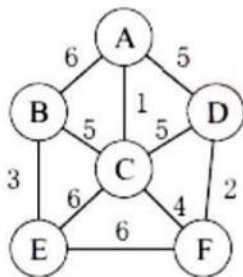


(e)

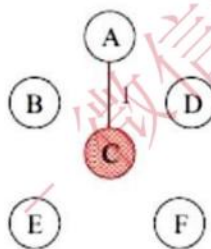


(f)

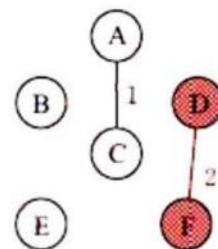
克鲁斯卡尔算法 (推荐): 这个算法是**从边出发**的, 因为本质是选取权值最小的 $n-1$ 条边, 因此, 就将边按权值大小排序, 依次选取权值最小的边, 直至囊括所有节点, 要注意, 每次选边后要检查不能形成环路。克鲁斯卡尔算法的时间复杂度为 $O(e \log e)$, 与图中的顶点数无关, 因此该算法适合于求边稀疏的网的最小生成树。



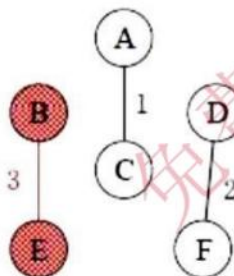
(a)



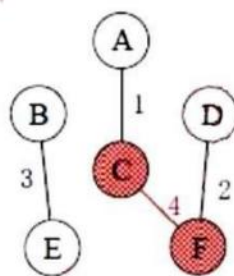
(b)



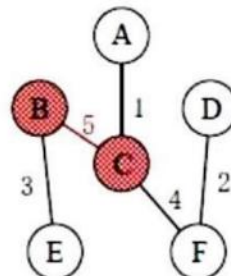
(c)



(d)



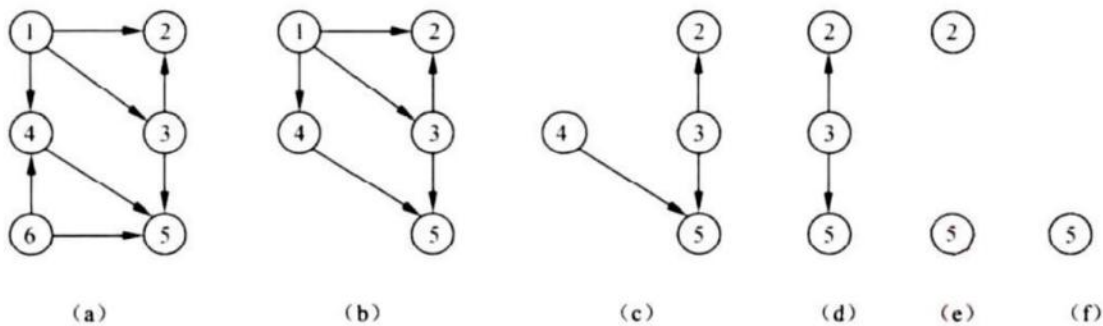
(e)



(f)

4.5 拓扑序列

将有向图的有向边作为活动开始的顺序, 若图中一个节点入度为 **0**, 则应该最先执行此活动, 而后删除掉此节点和其关联的有向边, 再去找图中其他没有入度的结点, 执行活动, 依次进行, 示例如下 (有点类似于**进程的前趋图原理**):



九 算法分析与设计

1 算法分析

1.1 算法的特性

算法 (Algorithm) 是对特定问题求解步骤的一种描述, 它是指令的有限序列, 其中每一条指令表示一个或多个操作。此外, 一个算法还具有下列 5 个重要特性。

(1) 有穷性。一个算法必须总是 (对任何合法的输入值) 在执行有穷步之后结束, 且每一步都可在有穷时间内完成。

(2) 确定性。算法中的每一条指令必须有确切的含义, 理解时不会产生二义性。并且在任何条件下, 算法只有唯一的一条执行路径, 即对于相同的输入只能得出相同的输出。

(3) 可行性。一个算法是可行的, 即算法中描述的操作都可以通过已经实现的基本运算执行有限次来实现。

(4) 输入。一个算法有零个或多个输入, 这些输入取自于某个特定的对象的集合。

(5) 输出。一个算法有一个或多个输出, 这些输出是同输入有着某些特定关系的量。

1.2 算法的复杂度

算法的时间复杂度分析: 主要是分析算法的运行时间, 即算法执行所需要的基本操作数。不同规模的输入所需要的基本操作数是不相同的, 例如用同一个排序算法排序 100 个数和排序 10000 个数所需要的基本操作数是不相同的, 因此考虑特定输入规模的算法的具体操作数既是不现实的也是不必要的。在算法分析中, 可以建立以输入规模 n 为自变量的函数 $T(n)$ 来表示算法的时间复杂度。

即使对于相同的输入规模，数据分布不相同也影响了算法执行路径的不同，因此所需要的执行时间也不同。根据不同的输入，将算法的时间复杂度分析分为 3 种情况：最佳情况、最坏情况、平均情况。

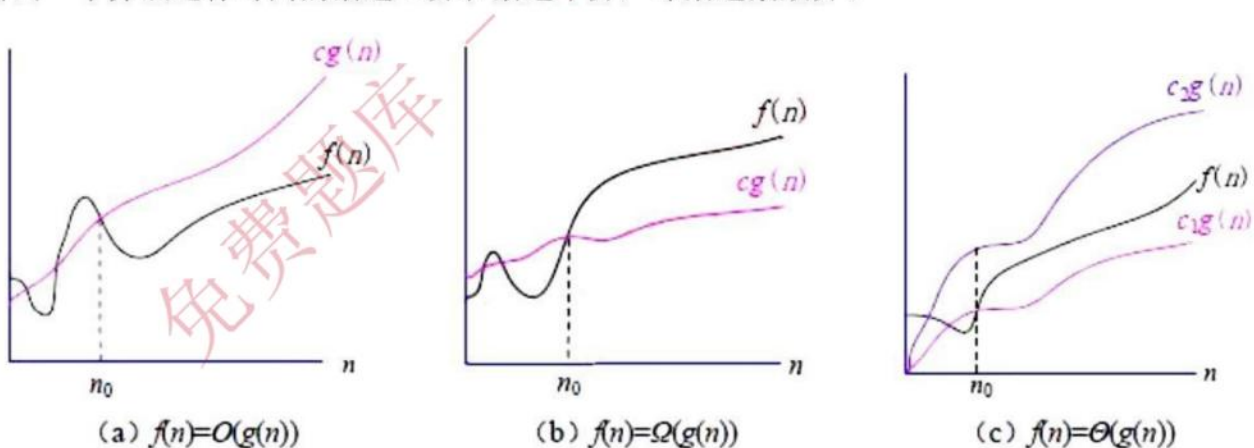
渐进符号

以输入规模 n 为自变量建立的时间复杂度实际上还是较复杂的，例如 an^2+bn+c ，不仅与输入规模有关，还与系数 a 、 b 和 c 有关。此时可以对该函数做进一步的抽象，仅考虑运行时间的增长率或称为增长的量级，如忽略上式中的低阶项和高阶项的系数，仅考虑 n^2 。当输入规模大到只有与运行时间的增长量级有关时，就是在研究算法的渐进效率。也就是说，从极限角度看，只关心算法运行时间如何随着输入规模的无限增长而增长。下面简单介绍 3 种常用的标准方法来简化算法的渐进分析。

(1) O 记号。定义为：给定一个函数 $g(n)$ ， $O(g(n))=\{f(n): \text{存在正常数 } c \text{ 和 } n_0, \text{ 使得对所有的 } n \geq n_0, \text{ 有 } 0 \leq f(n) \leq cg(n)\}$ ，如图 (a) 所示。 $O(g(n))$ 表示一个函数集合，往往用该记号给出一个算法运行时间的渐进上界。考试中一般只涉及到 O 符号。

(2) Ω 记号。定义为：给定一个函数 $g(n)$ ， $\Omega(g(n))=\{f(n): \text{存在正常数 } c \text{ 和 } n_0, \text{ 使得对所有的 } n \geq n_0, \text{ 有 } 0 \leq g(n) \leq f(n)\}$ ，如图 (b) 所示。 $\Omega(g(n))$ 表示一个函数集合，往往用该记号给出一个算法运行时间的渐进下界。

(3) Θ 记号。定义为：给定一个函数 $g(n)$ ， $\Theta(g(n))=\{f(n): \text{存在正常数 } c_1, c_2 \text{ 和 } n_0, \text{ 使得对所有的 } n \geq n_0, \text{ 有 } 0 \leq c_1g(n) \leq f(n) \leq c_2g(n)\}$ ，如图 (c) 所示。 $\Theta(g(n))$ 表示一个函数集合，往往用该记号给出一个算法运行时间的渐进上界和渐进下界，即渐进紧致界。



对上述的时间复杂度，经常考到，需要注意的是，时间复杂度是一个大概的规模表示，一般以循环次数表示（因为程序运行时间与软硬件环境相关，不能以绝对运行时间衡量）， $O(n)$ 说明执行时间是 n 的正比，另外， $\log$ 对数的时间复杂度一般在查找二叉树的算法中出现。渐进符号 O 表示一个渐进变化程度，实际变化必须小于等于 O 括号内的渐进变化程度。

1.3 经典算法

常见算法问题：

背包问题：典型的贪心算法实例，0-1 背包问题，物品是一个整体，要么放进去要么不放，采用动态规划法算法实现，实际选择题可以求出单位重量价值而后穷举计算。部分背包问题，物品是可以拆分的，例如可以放 1/3 重量的该物体进去，采用贪心法实现，实际选择题可以求出单位重量价值后，从高到低依次塞入背包。注意不同问题应用不同算法才能求出最优解，0-1 背包问题若使用贪心法只有局部最优，无法求出最优解。

2 查找算法

2.1 顺序查找

顺序查找的思想：将待查找的关键字为 key 的元素从头到尾与表中元素进行比较，如果中间存在关键字为 key 的元素，则返回成功；否则，则查找失败。

$$ASL_{ss} = \sum_{i=1}^n P_i C_i = \frac{1}{n} \sum_{i=1}^n (n-i+1) = \frac{n+1}{2}$$

平均查找长度为：

时间复杂度为 $O(n)$ 。

2.2 二分（折半）查找

只适用于待查找序列中的元素是**有序排列**的情况，因为是通过确定中间节点的值和待查找的元素值间的比较来确定带查找的元素值是在中间节点的左边还是右边，类似于查找二叉树，因此必须有序排列才适用。

设查找表的元素存储在一维数组 $r[1..n]$ 中，在表中元素已经按照关键字递增方式排序的情况下，进行**折半查找**的方法是：

- 1、首先将待查元素的关键字（key）值与表 r 中间位置上（下标为 mid ）记录的关键字进行比较，若相等，则查找成功；
- 2、若 $key > r[mid].key$ ，则说明待查记录只可能在后半个子表 $r[mid+1..n]$ 中，下一步应在后半个子表中查找；

3、若 $key < r[mid].key$ ，说明待查记录只可能在前半个子表 $r[1..mid-1]$ 中，下一步应在 r 的前半个子表中查找；

4、重复上述步骤，逐步缩小范围，直到查找成功或子表为空失败时为止。

要注意两点：**中间值位置求出若为小数，应该向下取整**，即 $4.5=4$ ，非四舍五入；中间值已经比较过不相等，在划分下一次比较区间时，无需将中间值位置再纳入下一次比较区间。

折半查找的时间复杂度为 $O(\log_2 n)$ 。

当查找的数据越多时，二分查找的效率越高。

2.3 散列（哈希）表

前面的查找方法，由于记录在存储结构中的相对位置是随机的，所以查找时都要通过一系列与关键字的比较才能确定被查记录在表中的位置。也就是说，这类查找都是以关键字的比较为基础的，而哈希表则通过一个以记录的关键字为自变量的函数（称为哈希函数）得到该记录的存储地址，所以在哈希表中进行查找操作时，需要用同一哈希函数计算得到待查记录的存储地址，然后到相应的存储单元去获得有关信息再判定查找是否成功。

散列（哈希）表：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的地址集上，并以关键字在地址集中的“像”作为记录在表中的存储位置。

示例如下：

例如，设关键码序列为“47, 34, 13, 12, 52, 38, 33, 27, 3”，哈希表表长为 11，哈希函数为 $Hash(key)=key \bmod 11$ ，则

$Hash(47) = 47 \bmod 11 = 3$, $Hash(34) = 34 \bmod 11 = 1$,

$Hash(13) = 13 \bmod 11 = 2$, $Hash(12) = 12 \bmod 11 = 1$,

$Hash(52) = 52 \bmod 11 = 8$, $Hash(38) = 38 \bmod 11 = 5$,

$Hash(33) = 33 \bmod 11 = 0$, $Hash(27) = 27 \bmod 11 = 5$,

$Hash(3) = 3 \bmod 11 = 3$ 。

使用线性探测法解决冲突构造的哈希表如下：

哈希地址	0	1	2	3	4	5	6	7	8	9	10
关键字	33	34	13	47	12	38	27	3	52		

在上图中，很明显，哈希函数产生了冲突，使用的是线性探测法解决冲突，还有其它方法如下：

线性探测法：按物理地址顺序取下一个空闲的存储空间。

伪随机数法：将冲突的数据随机存入任意空闲的地址中。

再散列法：原有的散列函数冲突后，继续用此数据计算另外一个哈希函数，用以解决冲突。

3 排序算法

3.1 概念和方法分类

排序分为稳定与不稳定排序，依据是**两个相同的值**在一个待排序序列中的顺序和排序后的顺序应该是**相对不变**的，即开始时 21 在 21 前，那排序结束后，若该 21 还在 21 前，则为稳定排序，**不改变相同值的相对顺序**。

另外，依据排序是在内存中进行的还是在外部进行的，又可以分为内排序和外排序。

排序的算法有很多，大致可以分类如下：

插入类排序：直接插入排序、希尔排序。

交换类排序：冒泡排序、快速排序。

选择类排序：简单选择排序、堆排序。

归并排序。

基数排序。

下面分别介绍各个排序的算法：

3.2 直接插入排序

要注意的是，前提条件是前 $i-1$ 个元素是有序的，第 i 个元素**依次从第 $i-1$ 个元素往前比较，直到找到一个比第 i 个元素值小的元素**，而后插入，插入位置及其后的元素依次向后移动，本质是插入排序。

当给出一队无序的元素时，首先，应该将第 1 个元素看做是一个有序的队列，而后从第 2 个元素起，按插入排序规则，依次与前面的元素进行比较，直到找到一个小于他的值，才插入。示例如下图所示：

初始状态 57 68 59 52

57 68 59 52

68>57，不处理

57 68 59 52

57<59<68，插在57之后

57 59 68 52

52<57，插在57之前

结果：52 57 59 68

上例中，59 依次向前比较，先和 68 比较，再和 57 比较，发现 57 比他小，才插入。

3.3 希尔 (shell) 排序

希尔排序又称“缩小增量排序”，是对直接插入排序方法的改进。

希尔排序的基本思想是：先将整个待排记录序列分割成若干子序列，然后分别进行直接插入排序，待整个序列中的记录基本有序时，再对全体记录进行一次直接插入排序。具体做法是：先取一个小于 n 的整数 d_1 作为第一个增量，把文件的全部记录分成 d_1 个组，将所有距离为 d_1 倍数的记录放在同一个组中，在各组内进行直接插入排序；然后取第二个增量 $d_2(d_2 < d_1)$ ，重复上述分组和排序工作，依此类推，直至所取的增量 $d_i=1(d_i < d_{i-1} < \dots < d_2 < d_1)$ ，即所有记录放在同一组进行直接插入排序为止。

按上述，希尔排序实际是为了解决大数据的排序问题，当待排序的数据很多时，使用直接插入排序效率很低，因此，采取分组的方法，使问题细化，可以提高效率，适用于多数据。示例如下：

当增量序列为“5, 3, 1”时，希尔插入排序过程如下。

[初始关键字]: 48 37 64 96 75 12 26 48 54 03

48 12 37 26 64 48 96 54 75 03

第一趟排序结果: 12 26 48 54 03 48 37 64 96 75

12 54 37 75 26 48 03 64 96

第二趟排序结果: 12 03 48 37 26 48 54 64 96 75

第三趟排序结果: 03 12 26 37 48 48 54 64 75 96

3.4 简单选择排序

n 个记录进行简单选择排序的基本方法是：通过 $n-i(1 \leq i \leq n)$ 在次关键字之间的比较，从 $n-i+1$ 个记录中选出关键字最小的记录，并和第 i 个记录进行交换，当 i 等于 n 时所有记录有序排列。

按上述，本质就是每次选择出最小的元素进行交换，主要是选择过程，交换过程只有一次。示例如下：

初始状态 57 68 59 52

最小值为52, 与第一个交换 $\left. \begin{array}{c} 52 \\ 68 \end{array} \right\} 59 \ 57$

最小值为57, 与第二个交换 $\left. \begin{array}{cc} 52 & 57 \end{array} \right\} 59 \ 68$

59就是最小值, 无需交换, 完成 52 57 59 68

因为在选择最小的元素的时候是无规律的, 因此, 当最小的元素值有两个时, 可能选择任何一个与第一个交换, 所以是不稳定的。

3.5 堆排序

对于 n 个元素的关键字序列 $\{K_1, K_2, \dots, K_n\}$, 当且仅当满足下列关系时称其为堆, 其中 $2i$ 和 $2i+1$ 需不大于 n 。

$$\begin{cases} K_i \leq K_{2i} \\ K_i \leq K_{2i+1} \end{cases} \quad \text{或} \quad \begin{cases} K_i \geq K_{2i} \\ K_i \geq K_{2i+1} \end{cases}$$

堆排序的基本思想是: 对一组待排序记录的关键字, 首先按堆的定义排成一个序列 (即建立初始堆), 从而可以输出堆顶的最大关键字 (对于大根堆而言), 然后将剩余的关键字再调整成新堆, 便得到次大的关键字, 如此反复, 直到全部关键字排成有序序列为止。

初始堆的建立方法是: 将待排序的关键字放到一棵完全二叉树的各个结点中 (此时完全二叉树并不一定具备堆的特性), 显然, 所有 $i > \lfloor \frac{n}{2} \rfloor$ 的结点 K_i 都没有子结点, 以这样的 K_i 为根的子树已经是堆, 因此初始建堆可从完全二叉树的第 $i \left(i = \lfloor \frac{n}{2} \rfloor \right)$ 个结点 K_i 开始, 通过调整, 逐步使以 $K_{\lfloor \frac{n}{2} \rfloor}$ 、 $K_{\lfloor \frac{n}{2} \rfloor - 1}$ 、 $K_{\lfloor \frac{n}{2} \rfloor - 2}$ 、 $\dots$ 、 K_2 、 K_1 为根的子树满足堆的定义。

为序列 (55, 60, 40, 10, 80, 65, 15, 5, 75) 建立初始大根堆的过程如图 3-56 所示, 调整为新堆的过程如图 3-57 所示。初始建堆示例如下:

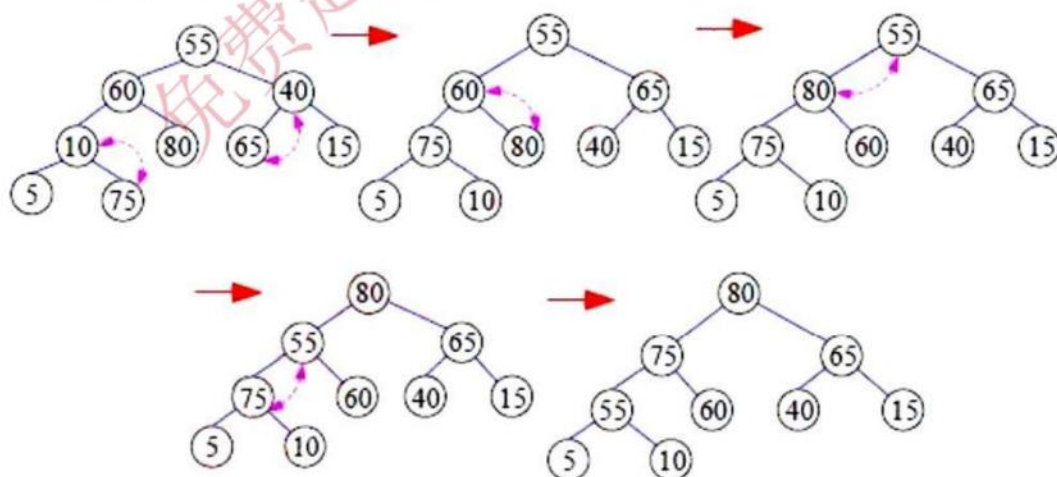


图 3-56 初始大根堆建立过程示意图

由上图可知，首先将给出的数组按完全二叉树规则建立，而后，找到此完全二叉树的最后一个非叶子节点（也即最后一颗子树），比较此非叶子节点和其两个孩子结点的大小，若小，则与其孩子结点中最大的结点进行交换；依据此规则再去找倒数第二个非叶子节点；这是只有一层的情况，当涉及到多层次时，又打破了之前的堆，因此，又要进行变换。

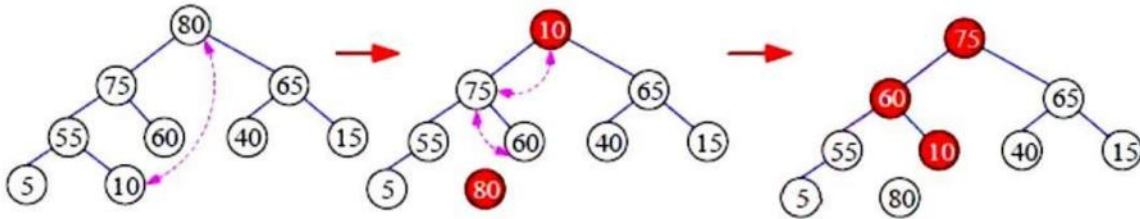


图 3-57 调整为新堆的过程示意图

由上图可知，取走堆顶元素后，将堆中最后一个元素移入堆顶，而后，按照初始建堆中的方法与其孩子结点比较大小，依次向下判断交换成为一个新的堆，再取走堆顶元素，重复此过程。

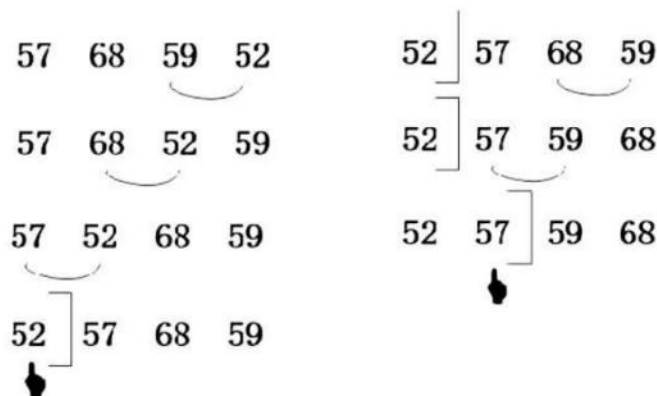
堆排序适用于在多个元素中找出前几名的方案设计，因为堆排序是选择排序，而且选择出前几名的效率很高。

3.6 冒泡排序

n 个记录进行冒泡排序的方法是：首先将第一个记录的关键字和第二个记录的关键字进行比较，若为逆序，则交换这两个记录的值，然后比较第二个记录和第三个记录的关键字，依此类推，直至第 $n-1$ 个记录和第 n 个记录的关键字比较过为止。上述过程称为一趟冒泡排序，其结果是关键字最大的记录被交换到第 n 个记录的位置上。然后进行第二趟冒泡排序，对前 $n-1$ 个记录进行同样的操作，其结果是关键字次大的记录被交换到第 $n-1$ 个记录的位置上。最多进行 $n-1$ 趟，所有记录有序排列。若在某趟冒泡排序过程没有进行相邻位置的元素交换处理，则可结束排序过程。

本质是从最后两个元素开始进行比较，将较小的元素交换到前面去，依次进行比较交换。比较是为了交换，交换次数很多。区分冒泡排序和简单选择排序。

初始状态 57 68 59 52



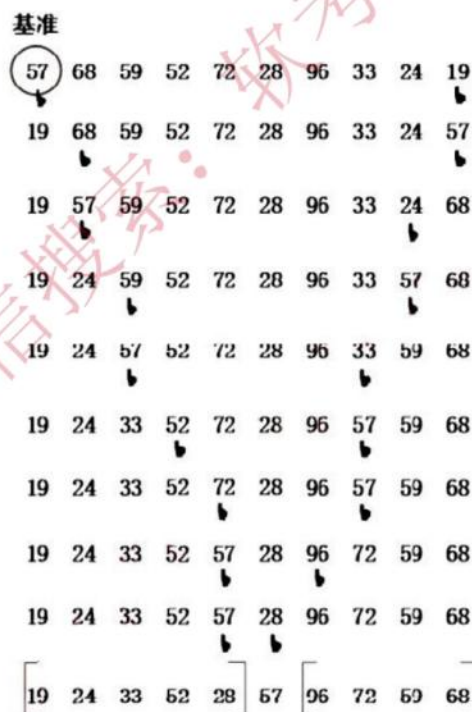
3.7 快速排序

快速排序采用的是分治法，其基本思想是将原问题分解成若干个规模更小但结构与原问题相似的子问题。通过递归地解决这些子问题，然后再将这些子问题的解组合成原问题的解。

快速排序通常包括两个步骤：

第一步，在待排序的 n 个记录中任取一个记录，以该记录的排序码为准，将所有记录都分成两组，第1组都小于该数，第2组都大于该数，如图所示。

第二步，采用相同的方法对左、右两组分别进行排序，直到所有记录都排到相应的位置为止。



快速排序的基本思想如上左图，是将 n 个记录分成两块，再递归，实际分成两块的方法如上右图所示：设定一个基准为57，设定两个指针 $high=1$ ， $low=n$ ，从 low 指向的第 n 个元素开始，与基准值进行比较，若小于基准值，则与基准进行交换 $low--$ ，此时，转而从 $high$ 指向的第 1 个元素开始和基准值进行比较，若大于基准值，则和基准值进行交换，此时，又转而从 low 指向的值和基准进行比较，重复上述过程。

要注意的是：每次都是和基准值进行比较，因此最终是以基准值为中间，将队列分成两块。只有当和基准值发生了交换，才变换 $high$ 和 low 指针的计数，否则，会一直 $low--$ 下去。

上图中，最终以 57 为界，左边都是小于 57 的元素，右边都是大于 57 的元素，完成一次快速排序，接着对两块再分别进行递归即可。

3.8 归并排序

所谓“归并”，是将两个或两个以上的有序文件合并成为一个新的有序文件。归并排序的一种实现方法是把一个有 n 个记录的无序文件看成是由 n 个长度为 1 的有序子文件组成的文件，然后进行两两归并，得到 $\lceil n/2 \rceil$ 个长度为 2 或 1 的有序文件，再两两归并，如此重复，直至最后形成包含 n 个记录的有序文件为止。这种反复将两个有序文件归并成一个有序文件的排序方法称为两路归并排序。

要仔细理解上述过程，一般归并排序都是用来合并多个线性表的，对单列数据，二路归并排序可以对元素进行两两合并，示例如下：

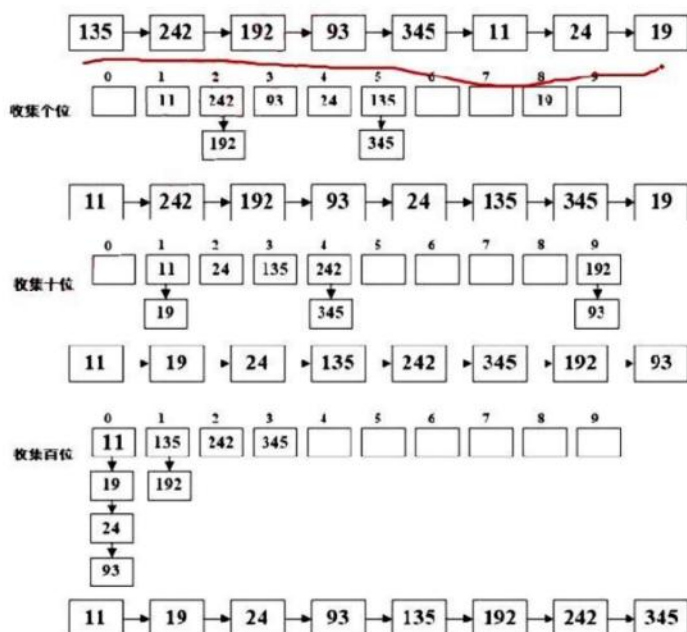
```

57 68 59 52 72 28 96 33
[57 68] [52 59] [28 72] [33 96]
[52 57 59 68] [28 33 72 96]
[28 33 52 57 59 68 72 96]
    
```

对第三次归并，将 52 与 28 比较，28 小，放入新表头，52 再与 33 比较，33 放入新表，52 再与 72 比较，52 放入新表，57 再与 72 比较，57 放入新表……

3.9 基数排序

基数排序是一种借助多关键字排序思想对单逻辑关键字进行排序的方法。基数排序不是基于关键字比较的排序方法，它适合于元素很多而关键字较少的序列。基数的选择和关键字的分解是根据关键字的类型来决定的，例如关键字是十进制数，则按个位、十位来分解。



由上图可知，基数排序是基于多个关键字来进行多轮排序的，本质也是将问题细分，如上右图例子，分别按个位、十位、百位的大小作为关键字进行了三轮排序，最终得出结果。

3.10 排序算法总结

上述内部排序的总体状态如下图所示：**要求记忆**

各种常用排序算法						
类别	排序方法	时间复杂度			空间复杂度	稳定性
		平均情况	最好情况	最坏情况	辅助存储	
插入排序	直接插入	$O(n^2)$	$O(n)$	$O(n^2)$	$O(1)$	稳定
	shell排序	$O(n^{1.3})$	$O(n)$	$O(n^2)$	$O(1)$	不稳定
选择排序	直接选择	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	不稳定
	堆排序	$O(n \log_2 n)$	$O(n \log_2 n)$	$O(n \log_2 n)$	$O(1)$	不稳定
交换排序	冒泡排序	$O(n^2)$	$O(n)$	$O(n^2)$	$O(1)$	稳定
	快速排序	$O(n \log_2 n)$	$O(n \log_2 n)$	$O(n^2)$	$O(n \log_2 n)$	不稳定
归并排序		$O(n \log_2 n)$	$O(n \log_2 n)$	$O(n \log_2 n)$	$O(1)$	稳定
基数排序		$O(d(r+n))$	$O(d(n+rd))$	$O(d(r+n))$	$O(rd+n)$	稳定
注：基数排序的复杂度中， r 代表关键字的基数， d 代表长度， n 代表关键字的个数						

前面已经介绍过，稳定性就是相等的两个元素的相对位置在排序前后保持不变。

空间复杂度中，大部分排序都是比较交换，无需多余空间，快速排序则是需要存储每次的基准值，归并排序需要一个新表，基数排序需要新表，还需要存储关键字的空间。

时间复杂度中，与堆、树、二分有关的算法都是 $n \cdot \log n$ ，直接的算法都是 $n \cdot n$ ，分析算法原理都可以轻易得出上述结论。

依据这些因素，可以得到以下几点结论。

- (1) 若待排序的记录数目 n 较小，可采用直接插入排序和简单选择排序。由于直接插入排序所需的记录移动操作较简单选择排序多，因此当记录本身信息量较大时，用简单选择排序方法较好。
- (2) 若待排序记录按关键字基本有序，则宜采用直接插入排序或冒泡排序。
- (3) 当 n 很大且关键字的位数较少时，采用链式基数排序较好。
- (4) 若 n 较大，则应采用时间复杂度为 $O(n \log n)$ 的排序方法，例如快速排序、堆排序或归并排序。

4 算法设计

4.1 递推和递归

算法是为了解决某个问题而设计的步骤和方法，有了算法，就可以据此编写程序。常用算法主要有迭代法、穷举搜索法、递推法、递归法、贪算法、回溯法等。

解决同一个问题，不同的人(甚至是同一个人)可能会写出几种不同的算法，但算法有优劣之分。

递推法是利用所解问题本身所具有的递推关系来求得问题解的一种算法。

递推法与递归法的关系是，任何可以用递推法解决的问题，可以很方便的用递归法写出程序解决。反之，许多用递归法解决的问题不能用递推法解决。这是因为递归法利用递归时的压栈，可以有任意长度和顺序的前效相关性，这是递推法所不具备的。

C 语言支持递归，即一个函数可以调用其自身。但在使用递归时，程序员需要注意定义一个从函数退出的条件，否则会进入死循环。

例：数的阶乘递归实现代码：

例：数的阶乘递归实现代码：

```
#include <stdio.h>

double factorial(unsigned int i)
{
    if (i <= 1)
    {
        return 1;
    }
    return i * factorial(i - 1);
}

int main()
{
    int i = 15;
    printf("%d 的阶乘为 %f\n", i, factorial(i));
    return 0;
}
```


4.2 分治法

概念：对于一个规模为 n 的问题，若该问题可以容易地解决则直接解决；否则将其分解为 k 个规模较小的子问题，这些子问题互相独立且与原问题形式相同，递归地解决这些子问题，然后将各子问题的解合并得到原问题的解。

步骤：分解（将原问题分解成一系列子问题）——求解（递归地求解各子问题，若子问题足够小，则直接求解）——合并（将子问题的解合并成原问题的解）。

凡是涉及到分组解决的都是分治法。

在分治法中，因为是分解为若干个**相同的问题**，因此会使用到**递归技术**：在函数运行过程中调用自己。

在**二分查找法**、归并排序等算法里，将待排序序列分解为一个个小序列先排序，后再不断扩大范围合并，就是分治法的原理（**将问题规模缩小**）。

4.3 回溯法

概念：有“通用的解题法”之称，可以系统地搜索一个问题的所有解或任一解。在包含问题的所有解的解空间树中，按照深度优先的策略，从根节点触发搜索解空间树。搜索至任一结点时，总是先判断该结点是否肯定不包含问题的解，如果不包含，则跳过对以该结点为根的子树的搜索，逐层向其祖先结点回溯；否则，进入该子树，继续按深度优先的策略进行搜索。

可以理解为先进行深度优先搜索，一直向下探测，当此路不通时，返回上一层探索另外的分支，重复此步骤，这就是回溯，意为先一直探测，当不成功时再返回上一层。

一般用于**解决迷宫类**的问题。

4.4 贪心法

概念：总是做出在当前来说是最好的选择，而并不从整体上加以考虑，它所做的每步选择只是当前步骤的局部最优选择，但从整体来说不一定是最优的选择。由于它不比为了寻找最优解而穷尽所有可能解，因此其耗费时间少，一般可以快速得到满意的解，但得不到最优解。

局部贪心，只针对当前的步骤取最优，而非整体考虑。

判断此类算法，就看算法是否是**每一步都取最优**，并且整体题意没有透露出最终结果是最优的。

4.5 动态规划法

概念：在求解问题中，对于每一步决策，列出各种可能的局部解，再依据某种判定条件，舍弃哪些肯定不能得到最优解的局部解，在每一步都经过筛选，以每一步都是最优解来保证全局是最优解。

本质也是将复杂的问题划分成一个个子问题，与分治法不同的是每个子问题间不是相互独立的，并且不全都相同。

常用于求解具有某种最优性质的问题。

此算法会将大量精力放在前期构造表格上面，其会对每一步，列出各种可能的答案，这些答案会存储起来，最终要得出某个结果时，是通过查询这张表来得到的。动态规划法不但每一步最优，全局也最优。

4.6 分支限界法

原理：在分支限界法中，每一个活结点只有一次机会成为扩展结点。活结点一旦成为扩展结点，就一次性产生其所有儿子结点。在这些儿子结点中，导致不可行解或导致非最优解的儿子结点被舍弃，其余儿子结点被加入活结点表中。此后，从活结点表中取下一结点成为当前扩展结点，并重复上述结点扩展过程。这个过程一直持续到找到所需的解或活结点表为空时为止。

与回溯法的区别：1、求解目标是找出满足约束条件的一个解，或是在满足约束条件的解中找出使某一目标函数值达到极大或极小的解，即在某种意义下的最优解。

2、以广度优先或以最小耗费（最大收益）优先的方式搜索解空间树。

从活节点表中选择下一个扩展节点的类型：

队列式(FIFO)分支限界法：按照队列先进先出（FIFO）原则选取下一个节点为扩展节点。

优先队列式分支限界法：按照优先队列中规定的优先级选取优先级最高的节点成为当前扩展节点。

4.7 概率算法

原理：在算法执行某些步骤时，可以随机地选择下一步该如何进行，同时允许结果以较小的概率出现错误，并以此为代价，获得算法运行时间的大幅度减少（降低算法复杂度）。

基本特征是对所求解问题的同一实例用同一概率算法求解两次，可能得到完全不同的效果。

如果一个问题没有有效的确定性算法可以在一个合理的时间内给出解，但是该问题能接受小概率错误，那么采用概率算法就可以快速找到这个问题的解。

四类概率算法：数值概率算法（数值问题的求解）、蒙特卡洛算法（求问题的精确解）、拉斯维加斯算法（不会得到不正确的解）、舍伍德算法（总能求得问题的一个正确解）。

概率算法的特征：

- （1）概率算法的输入包括两部分，一部分是原问题的输入，另一部分是一个供算法进行随机选择的随机数序列。
- （2）概率算法在运行过程中，包括一处或多处随机选择，根据随机值来决定算法的运行路径。
- （3）概率算法的结果不能保证一定是正确的，但能限制其出错概率。
- （4）概率算法在不同的运行过程中，对于相同的输入实例可以有不同的结果，因此，对于相同的输入实例，概率算法的执行时间可能不同。

4.8 近似算法

原理：解决难解问题的一种有效策略，其基本思想是放弃求最优解，而用近似最优解代替最优解，以换取算法设计上的简化和时间复杂度的降低。

虽然它可能找不到一个最优解，但它总会给待求解的问题提供一个解。

为了具有**实用性**，近似算法必须能够给出算法所产生的解与最优解之间的差别或者比例的一个界限，它保证任意一个实例的近似最优解与最优解之间的相差成比例。显然，这个差别越小，近似算法越具有实用性。

衡量近似算法性能两个标准：

- （1）算法的时间复杂度。近似算法的时间复杂度必须是多项式阶的，这是近似算法的基本目标。
- （2）解的近似程度。近似最优解的近似程度也是设计近似算法的重要目标。近似程度与近似算法本身、问题规模，乃至不同的输入实例有关。

5 数据挖掘算法

分析爆炸式增长的各种数据的技术，以发现隐含在这些数据中的有价值的信息和知识。数据挖掘利用及其学习方法对多种数据进行分析 and 挖掘。其核心是算法，主要功能包括分类、回归、关联规则和聚类等。

5.1 分类

是一种有监督的学习过程，根据历史数据预测未来数据的模型。

分类的数据对象属性：一般属性、分类属性或目标属性。

分类设计的数据：训练数据集、测试数据集、未知数据。

数据分类的两个步骤：学习模型（基于训练数据集采用分类算法建立学习模型）、应用模型（应用测试数据集的数据到学习模型中，根据输出来评估模型的好坏以及将未知数据输入到学习模型中，预测数据的类型）。

分类算法：决策树归纳（自顶向下的递归树算法）、朴素贝叶斯算法、后向传播 BP、支持向量机 SVM。

5.2 频繁模式和关联规则挖掘

挖掘海量数据中的频繁模式和关联规则可以有效地指导企业发现交叉销售机会、进行决策分析和商务管理等。（沃尔玛-啤酒尿布故事）

首先要求出数据集中的频繁模式，然后由频繁模式产生关联规则。

关联规则挖掘算法：类 Apriori 算法、基于频繁模式增长的方法如 FP-growthh，使用垂直数据格式的算法，如 ECLAT。

5.3 聚类

是一种无监督学习过程。根据数据的特征，将相似的数据对象归为一类，不相似的数据对象归到不同的类中。物以类聚，人以群分。

典型算法：基于划分的方法、基于层次的方法、基于密度的方法、基于网格的方法、基于统计模型的方法。

数据挖掘的应用：信用分析、定向促销顾客的分类与聚类等。

6 智能优化算法

优化技术是一种以数学为基础，用于求解各种工程问题优化的应用技术。

6.1 人工神经网络 ANN

一个以有向图为拓扑结构的动态系统，通过对连续或断续的输入作状态响应而进行信息处理。从信息处理角度对人脑神经元网络进行抽象，建立某种简单模型，按不同的连接方式组成不同的网络。

深度学习的概念源于人工神经网络的研究，是机器学习研究中的一个新的领域。

6.2 遗传算法

源于模拟达尔文的“优胜劣汰、适者生存”的进化论和孟德尔.摩根的遗传变异理论，在迭代过程中保持已有的结构，同时寻找更好的结构。其本意是在人工适应系统中设计一种基于自然的演化机制。

6.3 模拟退火算法 SA

求解全局优化算法。基本思想来源于物理退火过程，包括三个阶段：加温阶段、等温阶段、冷却阶段。

将固体加温至充分高，再让其徐徐冷却，加温时，固体内部粒子随温升变为无序状，内能增大，而徐徐冷却时粒子渐趋有序，在每个温度都达到平衡态，最后在常温时达到基态，内能减为最小。

6.4 禁忌搜索算法 TS

模拟人类智力过程的一种全局搜索算法，是对局部邻域搜索的一种扩展。

从一个初始可行解出发，选择一系列的特定搜索方向（移动）作为试探，选择实现让特定的目标函数值变化最多的移动。为了避免陷入局部最优解，TS 搜索中采用了一种灵活的“记忆”技术，对已经进行的优化过程进行记录和选择，指导下一步的搜索方向，这就是 Tabu 表的建立。

6.5 蚁群算法

是一种用来寻找优化路径的概率型算法。

单个蚂蚁的行为比较简单，但是蚁群整体却可以体现一些智能的行为。例如蚁群可以在不同的环境下，寻找最短到达食物源的路径。这是因为蚁群内的蚂蚁可以通过某种信息机制实现信息的传递。后又经进一步研究发现，蚂蚁会在其经过的路径上释放一种可以称之为“信息素”的物质，蚁群内的蚂蚁对“信息素”具有感知能力，它们会沿着“信息素”浓度较高路径行走，而每只路过的蚂蚁都会在路上留下“信息素”，这就形成一种类似正反馈的机制，这样经过一段时间后，整个蚁群就会沿着最短路径到达食物源了。

用蚂蚁的行走路径表示待优化问题的可行解，整个蚂蚁群体的所有路径构成待优化问题的解空间。路径较短的蚂蚁释放的信息素量较多，随着时间的推进，较短的路径上累积的信息素浓度逐渐增高，

选择该路径的蚂蚁个数也愈来愈多。最终，整个蚂蚁会在正反馈的作用下集中到最佳的路径上，此时对应的便是待优化问题的最优解。

6.6 粒子群优化算法 PSO

又称为鸟群觅食算法，在鸟群觅食飞行时，在飞行过程中经常会突然改变方向、散开、聚集，其行为不可预测，但其整体总保持一致性，个体与个体间也保持着最适宜的距离。通过对类似生物群体行为的研究，发现生物群体中存在一种信息共享机制，为群体的进化提供了一种优势，这就是基本粒子群算法形成的基础。

设想这样一个场景：一群鸟在随机搜索食物。在这个区域里只有一块食物。所有的鸟都不知道食物在那里。但是他们知道当前的位置离食物还有多远。那么找到食物的最优策略是什么呢。最简单有效的就是搜寻目前离食物最近的鸟的周围区域。

PSO 从这种模型中得到启示并用于解决优化问题。PSO 中，每个优化问题的解都是搜索空间中的一只鸟。我们称之为“粒子”。所有的粒子都有一个由被优化的函数决定的适应值(fitness value)，每个粒子还有一个速度决定他们飞翔的方向和距离。然后粒子们就追随当前的最优粒子在解空间中搜索。

PSO 初始化为一群随机粒子(随机解)。然后通过迭代找到最优解。在每一次迭代中，粒子通过跟踪两个“极值”来更新自己。第一个就是粒子本身所找到的最优解，这个解叫做个体极值 pBest。另一个极值是整个种群目前找到的最优解，这个极值是全局极值 gBest。另外也可以不用整个种群而只是用其中一部分作为粒子的邻居，那么在所有邻居中的极值就是局部极值。

十 软件工程基础知识

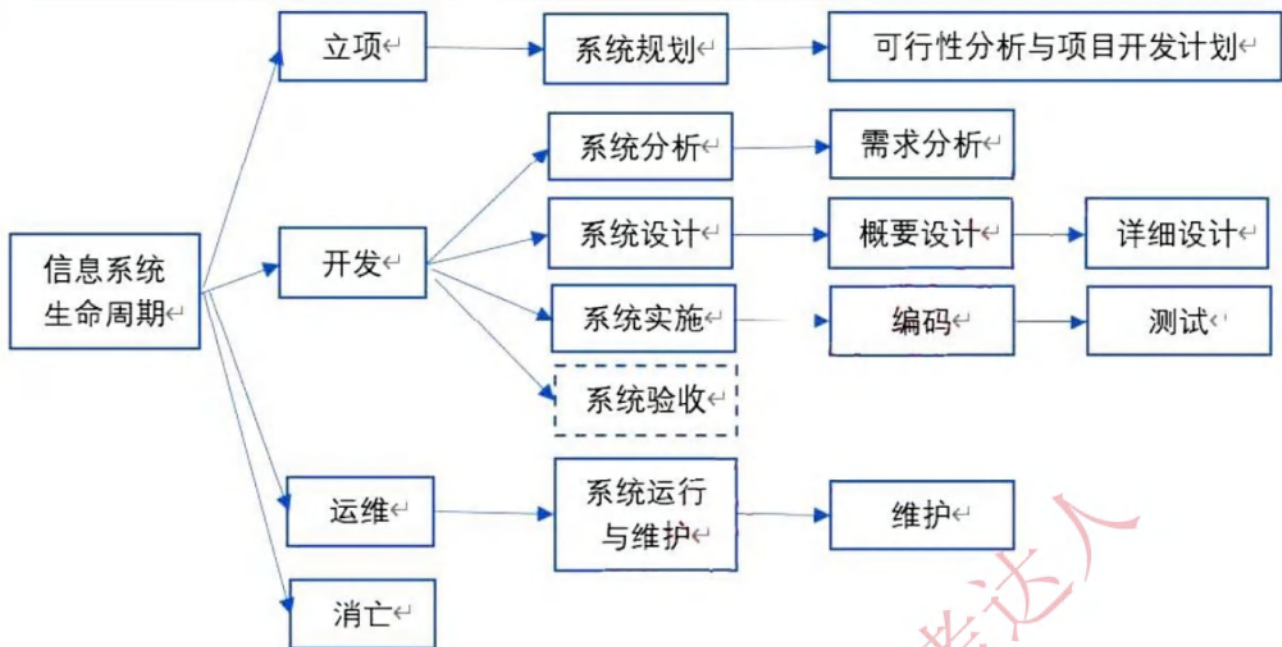
1 软件工程基础

1.1 能力成熟度模型

软件工程基本原理：用分阶段的生命周期计划严格管理、坚持进行阶段评审、实现严格的产品控制、采用现代程序设计技术、结果应能清楚的审查、开发小组的人员应少而精、承认不断改进软件工程实践的必要性。

软件工程的基本要素：方法、工具、过程。

信息系统生命周期



1.系统规划阶段：任务是对组织的环境、目标及现行系统的状况进行初步调查，根据组织目标和发展战略确定信息系统的发展战略，对建设新系统的需求做出分析和预测，同时考虑建设新系统所受的各种约束，研究建设新系统的必要性和可能性。根据需要与可能，给出制建系统的备选方案。

输出：可行性研究报告、系统设计任务书。

2.系统分析阶段：任务是根据系统设计任务书所确定的范围，对现行系统进行详细调查，描述现行系统的业务流程，指出现行系统的局限性和不足之处，确定新系统的基本目标和逻辑功能要求，即提出新系统的逻辑模型。系统分析阶段又称为逻辑设计阶段。这个阶段是整个系统建设的关键阶段，也是信息系统建设与一般工程项目的重要区别所在。

输出：系统说明书。

3.系统设计阶段：系统分析阶段的任务是回答系统“做什么”的问题，而系统设计阶段要回答的问题是“怎么做”。该阶段的任务是根据系统说明书中规定的功能要求，具体设计实现逻辑模型的技术方案，也就是设计新系统的物理模型。这个阶段又称为物理设计阶段，可分为总体设计（概要设计）和详细设计两个子阶段。

输出：系统设计说明书（概要设计、详细设计说明书）。

4.系统实施阶段：是将设计的系统付诸实施的阶段。这一阶段的任务包括计算机等设备的购置、安装和调试、程序的编写和调试、人员培训、数据文件转换、系统调试与转换等。这个阶段的特点是几个互相联系、互相制约的任务同时展开，必须精心安排、合理组织。系统实施是按实施计划分阶段完成的，每个阶段应写出实施进展报告。系统测试之后写出系统测试分析报告。

输出：实施进展报告、系统测试分析报告。

5.系统运行和维护阶段：系统投入运行后，需要经常进行维护和评价，记录系统运行的情况，根据一定的规则对系统进行必要的修改，评价系统的工作质量和经济效益。

软件生存周期：可行性分析与项目开发计划、需求分析、概要设计（选择系统解决方案，规划子系统）、详细设计（设计子系统内部具体实现）、编码、测试、维护。

能力成熟度模型 CMM

对软件组织化阶段的描述，随着软件组织定义、实施、测量、控制和改进其软件过程，软件组织的能力经过这些阶段逐步提高。针对软件研制和测试阶段。分为如下五个级别：

能力等级	特点	关键过程区域
初始级 (Initial)	软件过程的特点是杂乱无章，有时甚至很混乱，几乎没有明确定义的步骤，项目的成功完全依赖个人的努力和英雄式核心人物的作用。	
可重复级 (Repeatable)	建立了基本的项目管理过程和实践来跟踪项目费用、进度和功能特性，有必要的过程准则来重复以前在同类项目中的成功。	软件配置管理、软件质量保证、软件子合同管理、软件项目跟踪与监督、软件项目策划、软件需求管理
已定义级 (Defined)	管理和工程两方面的软件过程已经文档化、标准化，并综合成整个软件开发组织的标准软件过程。所有项目都采用根据实际情况修改后得到的标准软件过程来发和维护软件。	同行评审、组间协调、软件产品工程、集成软件管理、培训大纲、组织过程定义、组织过程集点
已管理级 (Managed)	制定了软件过程和产品质量的详细度量标准。对软件过程和产品质量有定量的理解和控制。	软件质量管理和定量过程管理
优化级 (Optimized)	加强了定量分析，通过来自过程质量反馈和来自新观念、新技术的反馈使过程能不断持续地改进。	过程更改管理、技术改革管理和缺陷预防

能力成熟度模型集成 CMMI

是若干过程模型的综合和改进，不仅仅软件，而是支持多个工程学科和领域的、系统的、一致的过程改进框架，能适应现代工程的特点和需要，能提高过程的质量和工作效率。

CMMI 两种表示方法：

(1) 阶段式模型：类似于 CMM，它关注组织的成熟度，五个成熟度模型如下：

能力等级	特点	关键过程区域
初始级	过程不可预测且缺乏控制	
已管理级	过程为项目服务	需求管理、项目计划、配置管理、项目监督与控制、供应商合同管理、度量和分析、过程和产品质量保证

已定义级	过程为组织服务	需求开发、技术解决方案、产品集成、验证、确认组织级过程焦点、组织级过程定义、组织级培训、集成项目管理、风险管理、集成化的团队、决策分析和解决方案、组织级集成环境
定量管理	过程已度量和控制	组织过程性能、定量项目管理
优化级	集中于过程改进和优化	组织级改革与实施、因果分析和解决方案

其中 2-5 级对应的过程域如下图所示：

成熟度等级	过程域
已管理级	需求管理、项目计划、配置管理、项目监督与控制、供应商合同管理、度量和分析、过程和质量保证
已定义级	需求开发、技术解决方案、产品集成、验证、确认、组织级过程焦点、组织级过程定义、组织级培训、集成项目管理、风险管理、集成化的团队、决策分析和解决方案、组织级集成环境
定量管理级	组织级过程性能、定量项目管理
优化级	组织级改革与实施、因果分析和解决方案

2.连续式模型：关注每个过程域的能力，一个组织对不同的过程域可以达到不同的过程域能力等级。

1.2 软件过程模型

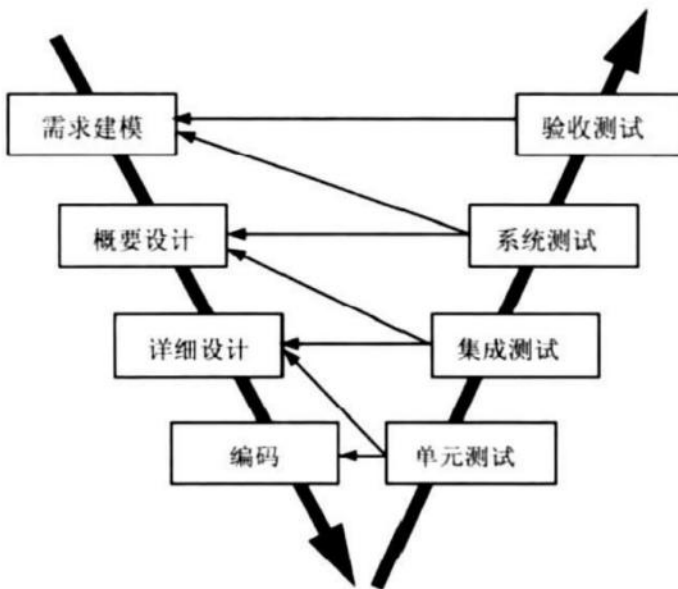
瀑布模型(SDLC)：结构化方法中的模型，是结构化的开发，开发流程如同瀑布一般，一步一步的走下去，直到最后完成项目开发，**只适用于需求明确或者二次开发**（需求稳定），当需求不明确时，最终开发的项目会错误，有很大的缺陷。

原型：与瀑布模型相反，原型针对的就是**需求不明确**的情况，首先快速构造一个功能模型，演示给用户看，并按用户要求及时修改，中间再通过不断的演示与用户沟通，最终设计出项目，就不会出现与用户要求不符合的情况，采用的是迭代的思想。不适合超大项目开发。

增量模型：首先开发核心模块功能，而后与用户确认，之后再开发次核心模块的功能，即每次开发一部分功能，并与用户需求确认，最终完成项目开发，优先级最高的服务最先交付，但由于并不是从系统整体角度规划各个模块，因此不利于模块划分。难点在于如何将客户需求划分为多个增量。与原型不同的是**增量模型的每一次增量版本都可作为独立可操作的作品**，而**原型的构造一般是为了演示**。

螺旋模型：是多种模型的混合，针对需求不明确的项目，与原型类似，但是**增加了风险分析**，这也是其最大的特点。适合大型项目开发。

V模型：特点是增加了**很多轮测试**，并且这些测试贯穿于软件开发的各个阶段，不像其他模型都是软件开发完再测试，很大程度上保证了项目的准确性。V模型开发和测试级别对应如下图：



喷泉模型：特点是**面向对象的模型**，而上述其他的模型都是结构化的模型，使用了迭代思想和无间隙开发。

基于构件的开发模型 CBSD：特点是**增强了复用性**，在系统开发过程中，会构建一个**构件库**，供其他系统复用，因此可以提高可靠性，节省时间和成本。

形式化方法模型：建立在严格**数学基础**上的一种软件开发方法，主要活动是生成计算机软件形式化的数学规格说明。

1.3 软件开发方法

上述的软件过程模型基本都可归属于下面开发方法中，注意特点：

结构化方法

结构是指系统内各个组成要素之间的相互联系、相互作用的框架。结构化方法也称为生命周期法，是一种传统的信息系统开发方法，由结构化分析（Structured Analysis, SA）、结构化设计（Structured Design, SD）和结构化程序设计（Structured Programming, SP）三部分有机组合而成，其精髓是自顶向下、逐步求精和模块化设计。

结构化方法的主要特点

- （1）开发目标清晰化。结构化方法的系统开发遵循“用户第一”的原则。
- （2）开发工作阶段化。每个阶段工作完成后，要根据阶段工作目标和要求进行审查，这使各阶段工作有条不紊地进行，便于项目管理与控制。

(3) 开发文档规范化。结构化方法每个阶段工作完成后, 要按照要求完成相应的文档, 以保证各个工作阶段的衔接与系统维护工作的遍历。

(4) 设计方法结构化。在系统分析与设计时, 从整体和全局考虑, 自顶向下地分解; 在系统实现时, 根据设计的要求, 先编写各个具体的功能模块, 然后自底向上逐步实现整个系统。

结构化方法的不足和局限

(1) 开发周期长: 按顺序经历各个阶段, 直到实施阶段结束后, 用户才能使用系统。

(2) 难以适应需求变化: 不适用于需求不明确或经常变更的项目。

(3) 很少考虑数据结构: 结构化方法是一种面向数据流的开发方法, 很少考虑数据结构。

结构化方法常用工具

结构化方法一般利用图形表达用户需求, 常用工具有数据流图、数据字典、结构化语言、判定表以及判定树等。

面向对象方法

面向对象 (Object-Oriented, OO) 方法认为, 客观世界是由各种对象组成的, 任何事物都是对象, 每一个对象都有自己的运动规律和内部状态, 都属于某个对象类, 是该对象类的一个元素。复杂的对象可由相对简单的各种对象以某种方式而构成, 不同对象的组合及相互作用就构成了系统。

面向对象方法的特点

使用 OO 方法构造的系统具有更好的复用性, 其关键在于建立一个全面、合理、统一的模型 (用例模型和分析模型)。

OO 方法也划分阶段, 但其中的系统分析、系统设计和系统实现三个阶段之间已经没有“缝隙”。也就是说, 这三个阶段的界限变得不明确, 某项工作既可以在前一个阶段完成, 也可以在后一个阶段完成; 前一个阶段工作做得不够细, 在后一个阶段可以补充。

面向对象方法可以普遍适用于各类信息系统的开发。

面向对象方法的不足之处

必须依靠一定的面向对象技术支持, 在大型项目的开发上具有一定的局限性, 不能涉足系统分析以前的开发环节。

当前, 一些大型信息系统的开发, 通常是将结构化方法和 OO 方法结合起来。首先, 使用结构化方法进行自顶向下的整体划分; 然后, 自底向上地采用 OO 方法进行开发。因此, 结构化方法和 OO 方法仍是两种在系统开发领域中相互依存的、不可替代的方法。

原型化方法

原型化方法也称为快速原型法，或者简称为原型法。它是一种根据用户初步需求，利用系统开发工具，快速地建立一个系统模型展示给用户，在此基础上与用户交流，最终实现用户需求的信息系统快速开发的方法。

按是否实现功能分类：分为水平原型（行为原型，功能的导航）、垂直原型（结构化原型，实现了部分功能）。

按最终结果分类：分为抛弃式原型、演化式原型。

原型法的特点

原型法可以使系统开发的周期缩短、成本和风险降低、速度加快，获得较高的综合开发效益。

原型法是以用户为中心来开发系统的，用户参与的程度大大提高，开发的系统符合用户的需求，因而增加了用户的满意度，提高了系统开发的成功率。

由于用户参与了系统开发的全过程，对系统的功能和结构容易理解和接受，有利于系统的移交，有利于系统的运行与维护。

原型法的不足之处

开发的环境要求高。

管理水平要求高。

由以上的分析可以看出，原型法的优点主要在于能更有效地确认用户需求。从直观上来看，原型法适用于那些需求不明确的系统开发。事实上，对于分析层面难度大、技术层面难度不大的系统，适合于原型法开发。

从严格意义上来说，目前的原型法不是一种独立的系统开发方法，而只是一种开发思想，它只支持在系统开发早期阶段快速生成系统的原型，没有规定在原型构建过程中必须使用哪种方法。因此，它不是完整意义上的方法论体系。这就注定了原型法必须与其他信息系统开发方法结合使用。

面向服务的方法

面向服务（Service-Oriented, SO）的方法：进一步将接口的定义与实现进行解耦，则催生了服务和面向服务（Service-Oriented, SO）的开发方法。

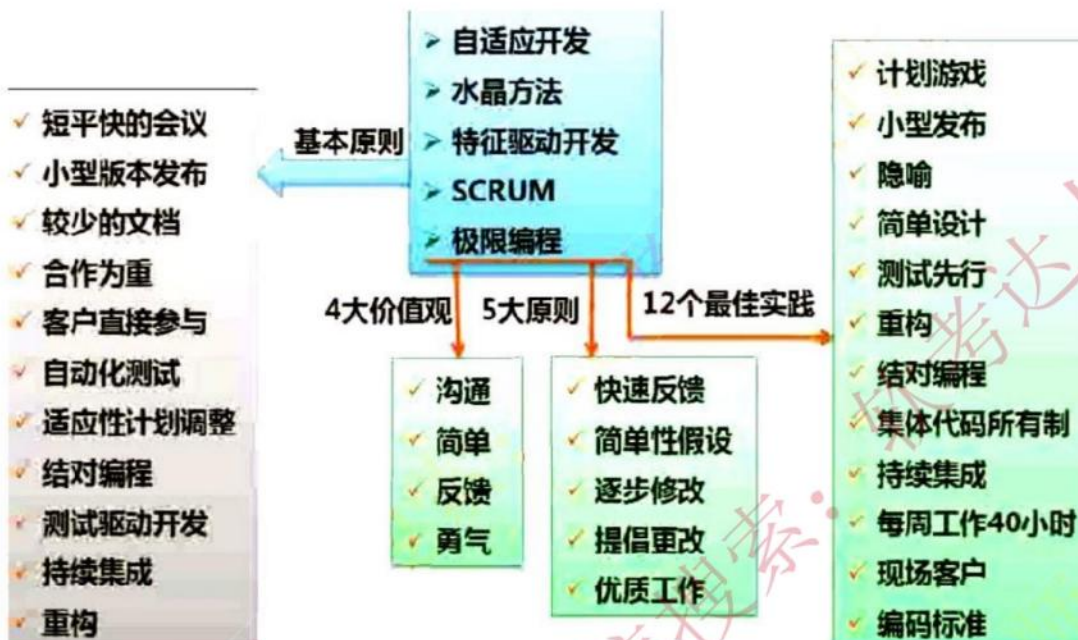
从应用的角度来看，组织内部、组织之间各种应用系统的互相通信和互操作性直接影响着组织对信息的掌握程度和处理速度。如何使信息系统快速响应需求与环境变化，提高系统可复用性、信息资源共享和系统之间的互操作性，成为影响信息化建设效率的关键问题，而 SO 的思维方式恰好满足了这种需求。

Jackson 方法：面向数据结构的开发方法，适合于小规模的项目。

敏捷开发

针对中小型项目，主要是为了给程序员减负，去掉一些不必要的会议和文档。指代一组模型（极限编程、自适应开发、水晶方法……），这些模型都具有相同的原则和价值观，具体如下图（要求对该图眼熟，并掌握重要概念）：

开发宣言：个体和交互 胜过 过程和工具、可以工作的软件 胜过 面面俱到的文档、客户合作 胜过 合同谈判、响应变化 胜过 遵循计划。



重要概念介绍

结对编程：一个程序员开发，另一个程序在一旁观察审查代码，能够有效的提高代码质量，在开发同时对代码进行初步审查，共同对代码负责。

自适应开发：强调开发方法的**适应性**（Adaptive）。不象其他方法那样有很多具体的实践做法，它更侧重为软件的重要性提供最根本的基础，并从更高的组织和管理层次来阐述开发方法为什么要具备适应性。

水晶方法：每一个不同的项目都需要一套不同的策略、约定和方法论。

特性驱动开发：是一套针对中小型软件开发项目的开发模式。是一个**模型驱动的快速迭代**开发过程，它强调的是简化、实用、易于被开发团队接受，适用于需求经常变动的项目。

极限编程 XP：核心是沟通、简明、反馈和勇气。因为知道计划永远赶不上变化，XP 无需开发人员在软件开始初期做出很多的文档。XP 提倡**测试先行**，为了将以后出现 bug 的几率降到最低。

并列争球法 SCRUM：是一种**迭代的增量化**过程，把每段时间（30 天）一次的迭代称为一个“**冲刺**”，并按需求的优先级别来实现产品，多个自组织和自治的小组并行地递增实现产品。

统一过程（RUP）

提供了在开发组织中分派任务和责任的纪律化方法。它的目标是在可预见的日程和预算前提下，确保满足最终用户需求的高质量产品。

3 个显著特点：用例驱动、以架构为中心、迭代和增量。

4 个流程：初始阶段、细化阶段、构建阶段和交付阶段。每个阶段结束时都要安排一次技术评审，以确定这个阶段的目标是否已经达到。

适用：一个通用过程框架，可以用于种类广泛的软件系统、不同的应用领域、不同的组织类型、不同性能水平和不同的项目规模。

1.4 软件产品线

软件产品线是一个产品集合，这些产品共享一个公共的、可管理的特征集，这个特征集能满足特定领域的特定需求。软件产品线是一个十分适合专业的开发组织的软件开发方法，能有效地提高软件生产率和质量，缩短开发时间，降低总开发成本。

核心资源：包括所有产品所共用的软件架构，通用的构件、文档等。

产品集合：产品线中的各种产品。

产品线的建立方式

表 12-1 软件产品线建立方式基本特征

	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

1.5 逆向工程

软件复用是将已有软件的各种有关知识用于建立新的软件，以缩减软件开发和维护的花费。软件复用是提高软件生产力和质量的一种重要技术。早期的软件复用主要是代码级复用，被复用的知识专指程序，后来扩大到包括领域知识、开发经验、设计决定、体系结构、需求、设计、代码和文档等一切有关方面。

逆向工程：软件的逆向工程是分析程序，力图在比源代码更高抽象层次上建立程序的表示过程，逆向工程是设计的恢复过程。逆向工程的四个级别：

实现级：包括程序的抽象语法树、符号表、过程的设计表示。

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、程序和数据结构。

功能级：包括反映程序段功能及程序段之间关系的信息，例如数据和控制流模型。

领域级：包括反映程序分量或程序诸实体与应用领域概念之间对应关系的信息，例如 E-R 模型。其中，领域级抽象级别最高，完备性最低，实现级抽象级别最低，完备性最高。

与逆向工程相关的概念有**重构、设计恢复、再工程和正向工程**。

(1) 重构是指在同一抽象级别上转换系统描述形式。

(2) 设计恢复是指借助工具从已有程序中抽象出有关数据设计、总体结构设计和过程设计等方面的信息。

(3) 再工程是指在逆向工程所获得信息的基础上，修改或重构已有的系统，产生系统的一个新版本。再工程是对现有系统的重新开发过程，包括逆向工程、新需求的考虑过程和正向工程三个步骤。它不仅能从已存在的程序中重新获得设计信息，而且还能使用这些信息来重构现有系统，以改进它的综合质量。在利用再工程重构现有系统的同时，一般会增加新的需求，包括增加新的功能和改善系统的性能。

(4) 正向工程是指不仅从现有系统中恢复设计信息，而且使用该信息去改变或重构现有系统，以改善其整体质量。

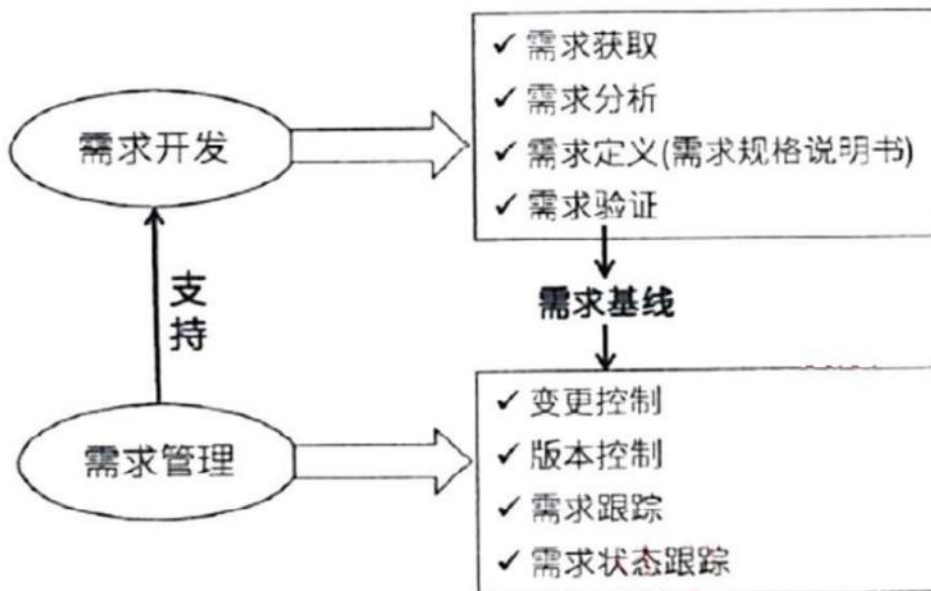
软件系统工具通常可以按软件过程活动将软件工具分为软件开发工具、软件维护工具、软件管理和软件支持工具。软件开发工具：需求分析工具、设计工具、编码与排错工具。软件维护工具：版本控制工具、文档分析工具、开发信息库工具、逆向工程工具、再工程工具。软件管理和软件支持工具：项目管理工具、配置管理工具、软件评价工具、软件开发工具的评价和选择。

2 需求工程

2.1 软件需求

是指用户对系统在功能、行为、性能、设计约束等方面的期望。是指用户解决问题或达到目标所需的条件或能力，是系统或系统部件要满足合同、标准、规范或其他正式规定文档所需具有的条件或能力，以及反映这些条件或能力的文档说明

分为需求开发和需求管理两大过程，如下所示：



需求的层次

业务需求：反映企业或客户对系统高层次的目标要求，通常来自项目投资人、客户、市场营销部门或产品策划部门。通过业务需求可以确定项目视图和范围。

用户需求：描述的是用户的具体目标，或用户要求系统必须能完成的任务。即描述了用户能使用系统来做什么。通常采取用户访谈和问卷调查等方式，对用户使用的场景进行整理，从而建立用户需求。

系统需求：从系统的角度来说明软件的需求，包括功能需求、非功能需求和设计约束等。

1) **功能需求：**也称为行为需求，规定了开发人员必须在系统中实现的软件功能，用户利用这些功能来完成任务，满足业务需要。

2) **非功能需求：**指系统必须具备的属性或品质，又可以细分为软件质量属性（如可维护性、可靠性、效率等）和其他非功能需求。

3) **设计约束：**也称为限制条件或补充规约，通常是对系统的一些约束说明，例如必须采用国有自主知识产权的数据库系统，必须运行在 UNIX 操作系统之下等。

质量功能部署

质量功能部署（QFD）是一种将用户要求转化成软件需求的技术，其目的是最大限度地提升软件工程过程中用户的满意度。为了达到这个目标，QFD 将软件需求分为三类，分别是常规需求、期望需求和意外需求。

（1）**常规需求。**用户认为系统应该做到的功能或性能，实现越多用户会越满意。

（2）**期望需求。**用户想当然认为系统应具备的功能或性能，但并不能正确描述自己想要得到的这些功能或性能需求。如果期望需求没有得到实现，会让用户感到不满意。

(3) 意外需求。意外需求也称为兴奋需求，是用户要求范围外的功能或性能（但通常是软件开发人员很乐意赋予系统的技术特性），实现这些需求用户会更高兴，但不实现也不影响其购买的决策。

2.2 需求获取

是一个确定和理解不同的项目干系人的需求和约束的过程。

常见的需求获取法包括：

- (1) 用户访谈：1 对 1-3，有代表性的用户。其形式包括结构化和非结构化两种。
- (2) 问卷调查：用户多，无法一一访谈。
- (3) 采样：从种群中系统地选出有代表性的样本集的过程。样本数量= $0.25 \times (\text{可信度因子}/\text{错误率})^2$
- (4) 情节串联板：一系列图片，通过这些图片来讲故事。
- (5) 联合需求计划（JRP）：通过联合各个关键用户代表、系统分析师、开发团队代表一起，通过有组织的会议来讨论需求。
- (6) 需求记录技术：任务卡片、场景说明、用户故事、Volere 白卡。

2.3 需求分析

一个好的需求应该具有无二义性、完整性、一致性、可测试性、确定性、可跟踪性、正确性、必要性等特性，因此，需要分析人员把杂乱无章的用户要求和期望转化为用户需求，这就是需求分析的工作。

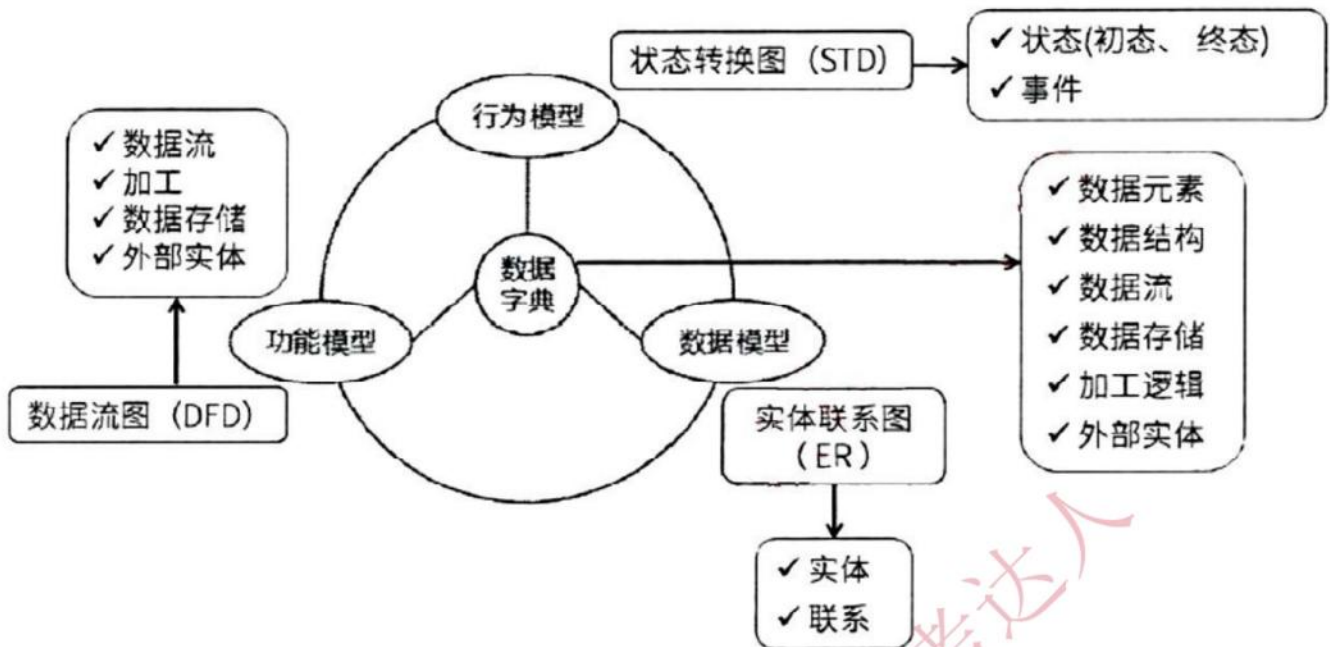
需求分析的任务

- (1) 绘制系统上下文范围关系图
- (2) 创建用户界面原型
- (3) 分析需求的可行性
- (4) 确定需求的优先级
- (5) 为需求建立模型
- (6) 创建数据字典
- (7) 使用 QFD（质量功能部署）

结构化的需求分析

结构化特点：自顶向下，逐步分解，面向数据。

三大模型：功能模型（数据流图）、行为模型（状态转换图）、数据模型（E-R 图）以及数据字典。



2.4 需求定义

(软件需求规格说明书 SRS): 是需求开发活动的产物, 编制该文档的目的是使项目干系人与开发团队对系统的初始规定有一个共同的理解, 使之成为整个开发工作的基础。SRS 是软件开发过程中最重要的文档之一, 对于任何规模和性质的软件项目都不应该缺少。

需求定义方法

(1) 严格定义也称为预先定义, 需求的严格定义建立在以下的基本假设之上: 所有需求都能够被预先定义。开发人员与用户之间能够准确而清晰地交流。采用图形 (或文字) 可以充分体现最终系统。

(2) 原型方法, 迭代的循环型开发方式, 需要注意的问题: 并非所有的需求都能在系统开发前被准确地说明。项目干系人之间通常都存在交流上的困难, 原型提供了克服该困难的一个手段。特点: 需要实际的、可供用户参与的系统模型。有合适的系统开发环境。反复是完全需要和值得提倡的, 需求一旦确定, 就应遵从严格的方法。

2.5 需求验证

也称为需求确认, 目的是与用户一起确认需求无误, 对需求规格说明书 SAS 进行评审和测试, 包括两个步骤:

需求评审: 正式评审和非正式评审。

需求测试: 设计概念测试用例。

需求验证通过后，要请用户签字确认，作为验收标准之一，此时，这个需求规格说明书就是需求基线，不可以再随意更新，如果需要更改必须走需求变更流程。

2.6 需求管理

定义需求基线：通过了评审的需求说明书就是需求基线，下次如果需要变更需求，就需要按照流程来一步步进行。需求的流程及状态如下图所示：

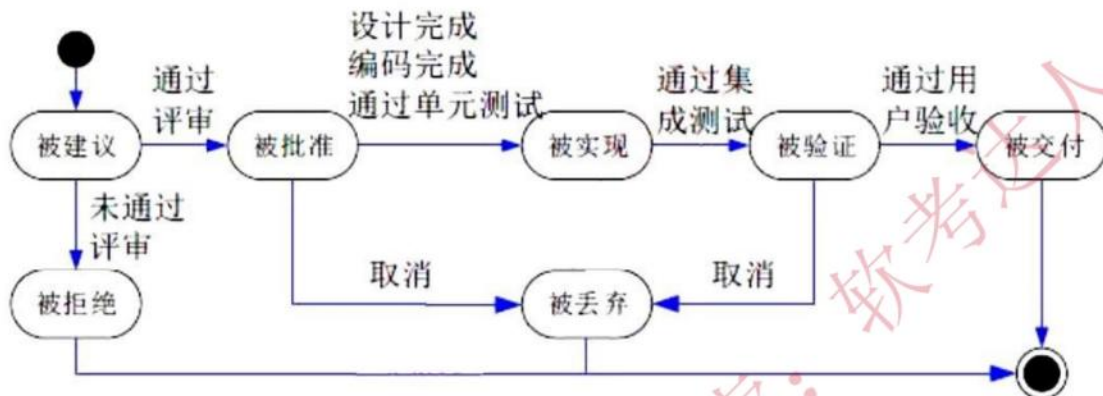


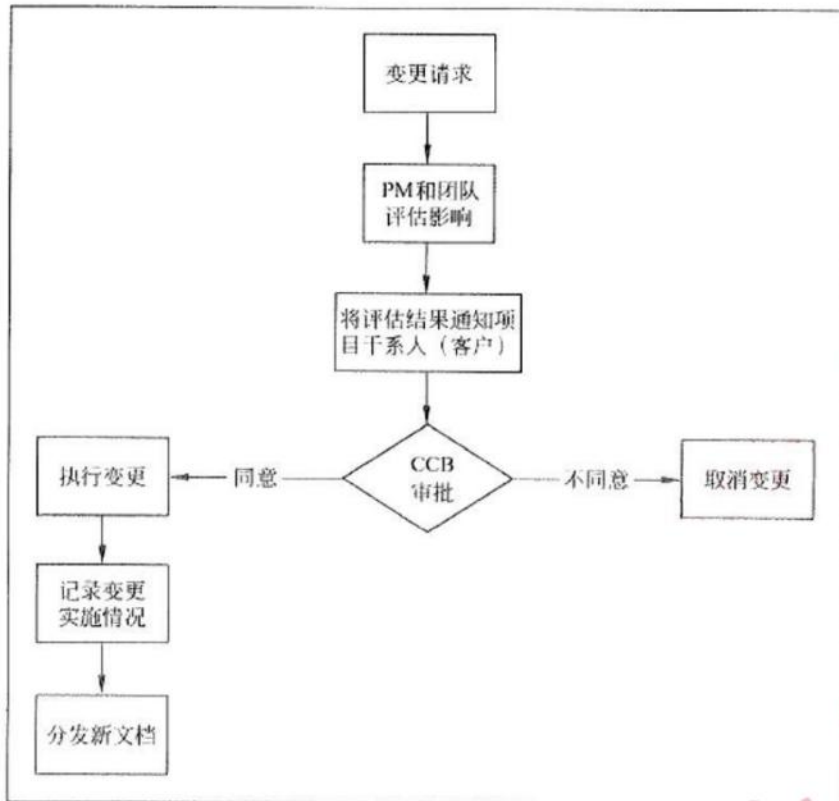
图 11-17 需求状态的变化

需求变更和风险

主要关心需求变更过程中的需求风险管理，带有风险的做法有：无足够用户参与、忽略了用户分类、用户需求的不断增加、模棱两可的需求、不必要的特性、过于精简的 SRS、不准确的估算。

变更产生的原因：外部环境的变化、需求和设计做的不够完整、新技术的出现、公司机构重组造成业务流程的变化。

变更控制委员会 CCB：也称为配置控制委员会，其任务时对建议的配置项变更做出评价、审批，以及监督已经批准变更的实施。



需求跟踪：双向跟踪，两个层次，如下图所示：



正向跟踪表示用户原始需求是否都实现了，反向跟踪表示软件实现的是否都是用户要求的，不多不少，可以用原始需求和用例表格（需求跟踪矩阵）来表示：

若原始需求和用例有对应，则在对应栏打对号，若某行都没有对号，表明原始需求未实现，正向跟踪发现问题；若某列都没有对号，表明有多余功能用例，软件实现了多余功能，反向跟踪发现问题。

3 系统设计

3.1 处理流程设计

业务流程建模

标杆瞄准：以行业领先的标杆企业为目标，结合本企业情况分析建模。

IDEF（一系列建模、分析和仿真方法的统称）。

DEMO（组织动态本质建模法）

Petri 网

业务流程建模语言：BPEL、BPML、BPMN、XPDL。

基于服务的 BPM：基于 Web 服务的思想对业务流程进行建模。

流程表示工具

程序流程图（Program Flow Diagram，PFD）用一些图框表示各种操作，它独立于任何一种程序设计语言，比较直观、清晰，易于学习掌握。任何复杂的程序流程图都应该由顺序、选择和循环结构组合或嵌套而成。

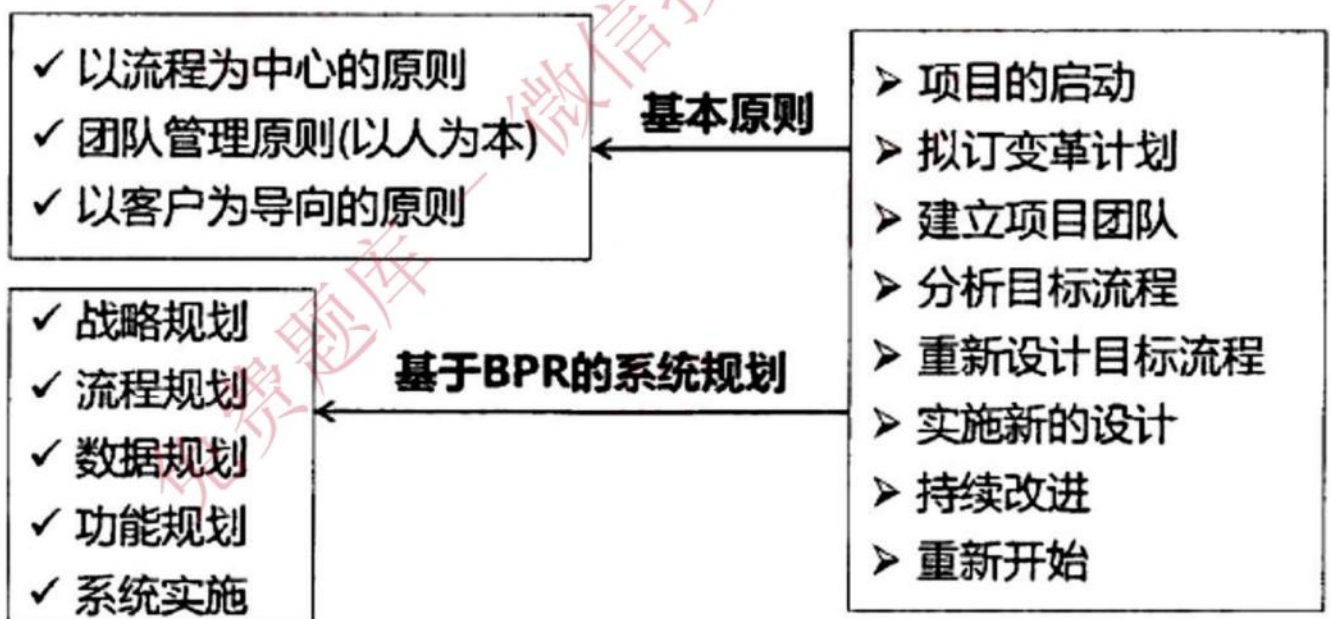
IPO 图也是流程描述工具，用来描述构成软件系统的每个模块的输入、输出和数据加工。

N-S 图容易表示嵌套和层次关系，并具有强烈的结构化特征。但是当问题很复杂时，N-S 图可能很大，因此不适合于复杂程序的设计。

问题分析图（PAD）是一种支持结构化程序设计的图形工具。PAD 具有清晰的逻辑结构、标准化的图形等优点，更重要的是，它引导设计人员使用结构化程序设计方法，从而提高程序的质量。

◆业务流程重组 BPR

BPR 是对企业的业务流程进行根本性的再思考和彻底性的再设计，从而获得可以用诸如成本、质量、服务和速度等方面的业绩来衡量的显著性的成就。BPR 设计原则、系统规划和步骤如下图所示：



◆业务流程管理 BPM

BPM 是一种以规范化的构造端到端的卓越业务流程为中心，以持续的提高组织业务绩效为目的的系统化方法。

BPM 与 BPR 管理思想最根本的不同就在于流程管理并不要求对所有的流程进行再造。构造卓越的业务流程并不是流程再造，而是根据现有流程的具体情况，对流程进行规范化的设计

流程管理包含三个层面：规范流程、优化流程和再造流程

3.2 系统设计

系统设计主要目的：为系统制定蓝图，在各种技术和实施方法中权衡利弊，精心设计，合理地使用各种资源，最终勾画出新系统的详细设计方法。

系统设计方法：结构化设计方法，面向对象设计方法。

系统设计的主要内容：概要设计、详细设计。

概要设计基本任务：又称为系统总体结构设计，是将系统的功能需求分配给软件模块，确定每个模块的功能和调用关系，形成软件的模块结构图，即系统结构图。

详细设计的基本任务：模块内详细算法设计、模块内数据结构设计、数据库的物理设计、其他设计（代码、输入/输出格式、用户界面）、编写详细设计说明书、评审。

系统设计基本原理

抽象化；

自顶而下，逐步求精；

信息隐蔽；

模块独立（高内聚，低耦合）。

系统设计原则

保持模块的大小适中；

尽可能减少调用的深度；

多扇入，少扇出；

单入口，单出口；

模块的作用域应该在模块之内；

功能应该是可预测的。

3.3 人机界面设计

人机界面设计的三大原则：置于用户控制之下、减少用户的记忆负担、保持界面的一致性。

置于用户的控制之下

以不强迫用户进入不必要的或不希望的动作的方式来定义交互方式；

提供灵活的交互；

允许用户交互可以被中断和取消；

当技能级别增加时可以使交互流水化并允许定制交互；
使用户隔离内部技术细节；
设计应允许用户和出现在屏幕上的对象直接交互。

减少用户的记忆负担

减少对短期记忆的要求；
建立有意义的缺省；
定义直觉性的捷径；
界面的视觉布局应该基于真实世界的隐喻；
以不断进展的方式揭示信息。

保持界面的一致性

允许用户将当前任务放入有意义的语境
在应用系列内保持一致性
如过去的交互模型已建立起了用户期望，除非有迫不得已的理由，不要去改变它。

4 测试基础知识

4.1 测试基础

测试原则

应尽早并不断的进行测试；
测试工作应该避免由元开发软件的人或小组承担；
在设计测试方案时，不仅要确定输入数据，而且要根据系统功能确定预期的输出结果；
既包含有效、合理的测试用例，也包含不合理、失效的用例；
检验程序是否做了该做的事，且是否做了不该做的事；
严格按照测试计划进行；
妥善保存测试计划和测试用例；
测试用例可以重复使用或追加测试。

测试类型分为两大类：

动态测试：程序运行时测试，分为

黑盒测试法：功能性测试，不了解软件代码结构，根据功能设计用例，测试软件功能。

白盒测试法：结构性测试，明确代码流程，根据代码逻辑设计用例，进行用例覆盖。

灰盒测试法：即既有黑盒，也有白盒。

静态测试：程序静止时，即对代码进行人工审查，分为

桌前检查：程序员检查自己编写的程序，在程序编译后，单元测试前。

代码审查：由若干个程序员和测试人员组成评审小组，通过召开程序评审会来进行审查。

代码走查：也是采用开会来对代码进行审查，但并非简单的检查代码，而是由测试人员提供测试用例，让程序员扮演计算机的角色，手动运行测试用例，检查代码逻辑。

测试策略

自底向上：从最底层模块开始测试，需要编写驱动程序，而后开始逐一合并模块，最终完成整个系统的测试。优点是较早的验证了底层模块。

自顶向下：先测试整个系统，需要编写桩程序，而后逐步向下直至最后测试最底层模块。优点是较早的验证了系统的主要控制和判断点。

三明治：既有自底向上也有自顶向下的测试方法，二者都包括。兼有二者的优点，缺点是测试工作量大。

4.2 测试阶段

(1) 单元测试：也称为模块测试，测试的对象是可独立编译或汇编的程序模块、软件构件或 OO 软件中的类（统称为模块），测试依据是软件详细设计说明书。

(2) 集成测试：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。测试依据是软件概要设计文档。

(3) 确认测试：主要用于验证软件的功能、性能和其他特性是否与用户需求一致。根据用户的参与程度，通常包括以下类型：

内部确认测试：主要由软件开发组织内部按照 SRS 进行测试。

Alpha 测试：用户在开发环境下进行测试。

Beta 测试：用户在实际使用环境下进行测试，通过改测试后，产品才能交付用户。

验收测试：针对 SRS，在交付前以用户为主进行的测试。其测试对象为完整的、集成的计算机系统。验收测试的目的是，在真实的用户工作环境下，检验软件系统是否满足开发技术合同或 SRS。验收测试的结论是用户确定是否接收该软件的主要依据。除应满足一般测试的准入条件外，在进行验收测试之前，应确认被测软件系统已通过系统测试。

(4) 系统测试：测试对象是完整的、集成的计算机系统；测试的目的是在真实系统工作环境下，验证完成的软件配置项能否和系统正确连接，并满足系统/子系统设计文档和软件开发合同规定的要求。测试依据是用户需求或开发合同。

主要内容包括功能测试、健壮性测试、性能测试、用户界面测试、安全性测试、安装与反安装测试等，其中，最重要的工作是进行功能测试与性能测试。功能测试主要采用黑盒测试方法；性能测试主要指标有响应时间、吞吐量、并发用户数和资源利用率等。

(5) 配置项测试：测试对象是软件配置项，测试目的是检验软件配置项与 SRS 的一致性。测试的依据是 SRS。在此之间，应确认被测软件配置项已通过单元测试和集成测试。

(6) 回归测试：测试目的是测试软件变更之后，变更部分的正确性和对变更需求的符合性，以及软件原有的、正确的功能、性能和其他规定的要求的不损害性。

4.3 测试用例设计

黑盒测试用例：将程序看做一个黑盒子，只知道输入输出，不知道内部代码，由此设计出测试用例，分为下面几类：

等价类划分：把所有的数据按照某种特性进行归类，而后在每类的数据里选取一个即可。等价类测试用例的设计原则：设计一个新的测试用例，使其尽可能多地覆盖尚未被覆盖的有效等价类，重复这一步，直到所有的有效等价类都被覆盖为止；设计一个新的测试用例，使其**仅覆盖一个尚未被覆盖的无效等价类**，重复这一步，直到所有的无效等价类都被覆盖为止。

边界值划分：将每类的边界值作为测试用例，边界值一般为范围的两端值以及在此范围之外的与此范围间隔最小的两个值，如年龄范围为 0-150，边界值为 0,150, -1,151 四个。

错误推测：没有固定的方法，凭经验而言，来推测有可能产生问题的地方，作为测试用例进行测试。

因果图：由一个结果来反推原因的方法，具体结果具体分析，没有固定方法。

白盒测试用例：知道程序的代码逻辑，按照程序的代码语句，来设计覆盖代码分支的测试用例，覆盖级别从低至高分为下面几种：

(1) 语句覆盖 SC：逻辑代码中的所有语句都要被执行一遍，覆盖层级最低，因为执行了所有的语句，不代表执行了所有的条件判断。

(2) 判定覆盖 DC：逻辑代码中的所有判断语句的条件真假分支都要覆盖一次。



语句覆盖 (SC)

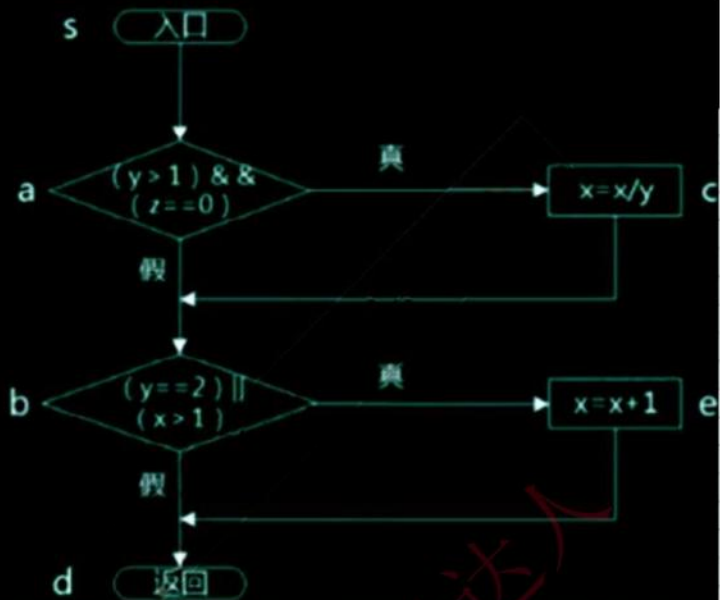
X=4, Y=2, Z=0 sacbed

判定覆盖 (DC)

又称分支覆盖

X=1, Y=3, Z=0 sacbd

X=3, Y=2, Z=1 sabed



(3) 条件覆盖 CC: 针对每一个判断条件内的每一个独立条件都要执行一遍真和假。

(4) 条件判定组合覆盖 CDC: 同时满足判定覆盖和条件覆盖。

条件覆盖 (CC)

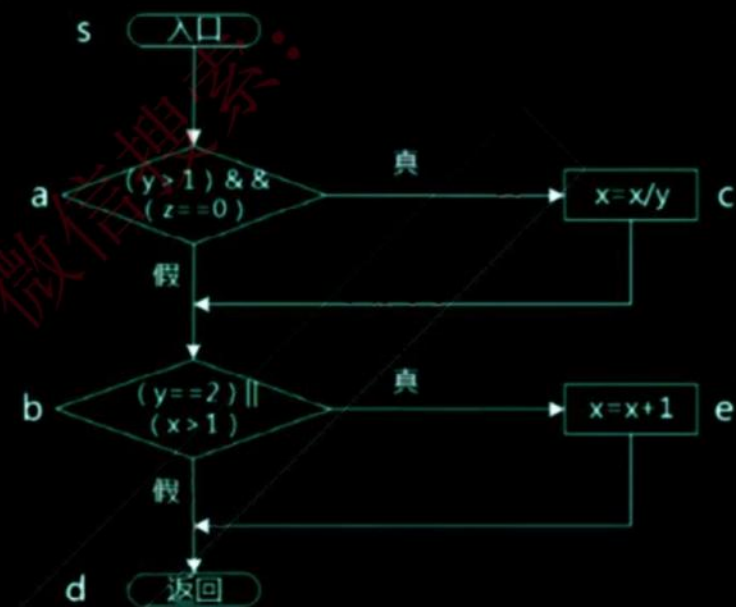
X=1, Y=2, Z=0 sacbed

X=2, Y=1, Z=1 sabed

条件判定组合覆盖 (CDC)

X=4, Y=2, Z=0 sacbed

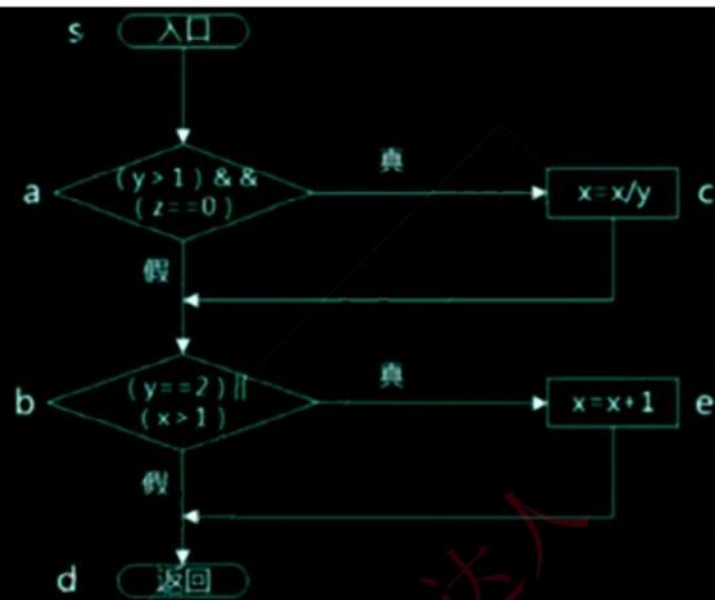
X=1, Y=1, Z=1 sabd



(5) 路径覆盖: 逻辑代码中的所有可行路径都覆盖了, 覆盖层级最高。

路径覆盖

X=1, Y=1, Z=1	sabd
X=3, Y=2, Z=0	sacbed
X=3, Y=3, Z=0	sacbd
X=1, Y=2, Z=1	sabed



4.4 调试

测试是发现错误，调试是找出错误的代码和原因。

调试需要确定错误的准确位置；确定问题的原因并设法改正；改正后要进行回归测试。

调试的方法

蛮力法：又称为穷举法或枚举法，穷举出所有可能的方法一一尝试。

回溯法：又称为试探法，按选优条件向前搜索，以达到目标，当发现原先选择并不优或达不到目标，就退回一步重新选择，这种走不通就退回再走的技术为回溯法。

演绎法：是由一般到特殊的推理方法，与“归纳法”相反，从一般性的前提出发。得出具体陈述或个别结论的过程。

归纳法：是由特殊到一般的推理方法，从测试所暴露的问题出发，收集所有正确或不正确的数据，分析它们之间的关系，提出假想的错误原因，用这些数据来证明或反驳，从而查出错误所在。

软件度量

软件的两种属性：外部属性指面向管理者和用户的属性，可直接测量，一般为性能指标。内部属性指软件产品本身的属性，如可靠性等，只能间接测量。

McCabe 度量法：又称为环路复杂度，假设有向图中有向边数为 m ，节点数为 n ，则此有向图的环路复杂度为 $m-n+2$ 。

注意 m 和 n 代表的含义不能混淆，可以用一个最简单的环路来做特殊值记忆此公式，另外，针对一个程序流程图，每一个分支边（连线）就是一条有向边，每一条语句（语句框）就是一个顶点。

5 系统运行与维护

5.1 系统转换

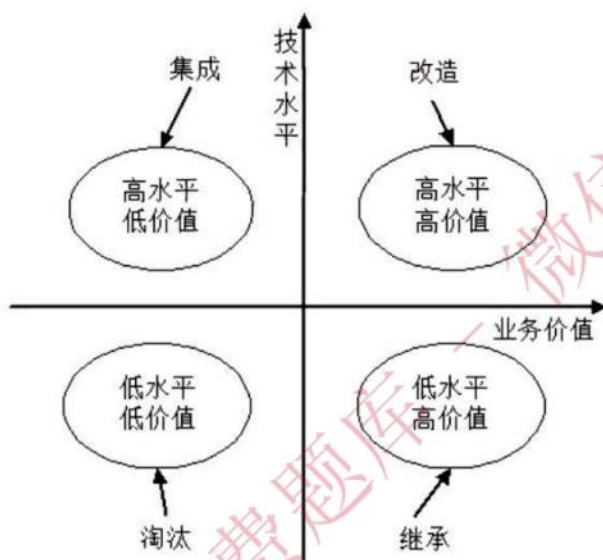
遗留系统是指任何基本上不能进行修改和演化以满足新的变化了的业务需求的信息系统，它通常具有以下特点：

(1) 系统虽然完成企业中许多重要的业务管理工作，但仍然不能完全满足要求。一般实现业务处理电子化及部分企业管理功能，很少涉及经营决策。

(2) 系统在性能上已经落后，采用的技术已经过时。例如，多采用主机/终端形式或小型机系统，软件使用汇编语言或第三代程序设计语言的早期版本开发，使用文件系统而不是数据库。

(3) 通常是大型的软件系统，已经融入企业的业务运作和决策管理机制之中，维护工作十分困难。

(4) 没有使用现代信息系统建设方法进行管理和开发，现在基本上已经没有文档，很难理解。



系统转换是指新系统开发完毕，投入运行，取代现有系统的过程，需要考虑多方面的问题，以实现与老系统的交接，有以下三种转换计划：

直接转换：现有系统被新系统直接取代了，风险很大，适用于新系统不复杂，或者现有系统已经不能使用的情况。优点是节省成本。

并行转换：新系统和老系统并行工作一段时间，新系统经过试运行后再取代，若新系统在试运行过程中有问题，也不影响现有系统的运行，风险极小，在试运行过程中还可以比较新老系统的性能，适用于大型系统。缺点是耗费人力和时间资源，难以控制两个系统间的数据转换。

分段转换：分期分批逐步转换，是直接和并行转换的集合，将大型系统分为多个子系统，依次试运行每个子系统，成熟一个子系统，就转换一个子系统。同样适用于大型项目，只是更耗时，而且现有系统和新系统间混合使用，需要协调好接口等问题。

数据转换与迁移：将数据从旧数据库迁移到新数据库中。要在新系统中尽可能的保存旧系统中合理的数据结构，才能降低迁移的难度。也有三种方法：系统切换前通过工具迁移、系统切换前采用手工录入、系统切换后通过新系统生成。

5.2 系统维护概述

系统的可维护性可以定义为维护人员理解、改正、改动和改进这个软件的难易程度，其评价指标如下：

- (1) 易分析性。软件产品诊断软件中的缺陷或失效原因或识别待修改部分的能力。
- (2) 易改变性。软件产品使指定的修改可以被实现的能力，实现包括编码、设计和文档的更改。
- (3) 稳定性。软件产品避免由于软件修改而造成意外结果的能力。
- (4) 易测试性。软件产品使已修改软件能被确认的能力。
- (5) 维护性的依从性。软件产品遵循与维护性相关的标准或约定的能力。

系统维护包括硬件维护、软件维护和数据维护，其中软件维护类型如下：

正确性维护：发现了 bug 而进行的修改。

适应性维护：由于外部环境发生了改变，被动进行的对软件的修改和升级。

完善性维护：基于用户主动对软件提出更多的需求，修改软件，增加更多的功能，使其比之前的软件功能、性能更高，更加完善。

预防性维护：对未来可能发生的 bug 进行预防性的修改。

5.3 系统评价

系统评价分类

立项评价：系统开发前的预评价，分析是否立项开发，做可行性评价。

中期评价：项目开发中期每个阶段的阶段评审。或者项目在开发中途遇到重大变故，评价是否还要继续。

结项评价：系统投入正式运行后，了解系统是否达到预期的目的和要求而对系统进行的综合评价。

系统评价的指标

(1) 从信息系统的组成部分出发，信息系统是一个由人机共同组成的系统，所以可以按照运行效果和用户需求（人）、系统质量和技术条件（机）这两条线索构造指标。

(2) 从信息系统的评价对象出发，对于开发方来说，他们所关心的是系统质量和技术水平；对于用户方而言，关心的是用户需求和运行质量；系统外部环境则主要通过社会效益指标来反映。

(3) 从经济学角度出发，分别按系统成本、系统效益和财务指标 3 条线索建立指标。

十一 项目管理

1 范围管理

范围管理确定在项目内包括什么工作和不包括什么工作，由此界定的项目范围在项目的全生命周期内可能因种种原因而变化，项目范围管理也要管理项目范围的这种变化。项目范围的变化也叫变更。

对项目范围的管理，是通过 5 个管理过程来实现的：

(1) 规划范围管理（编制范围管理计划）。对如何定义、确认和控制项目范围的过程进行描述。

(2) 定义范围。详细描述产品范围和项目范围，编制项目范围说明书，作为以后项目决策的基础。其输入包括：项目章程。项目范围管理计划。组织过程资产。批准的变更申请。

(3) 创建工作分解结构。把整个项目工作分解为较小的、易于管理的组成部分，形成一个自上而下的分解结构。

(4) 确认范围。正式验收已完成的可交付成果。

(5) 范围控制。监督项目和产品的范围状态、管理范围基准变更。

产品范围和项目范围

产品范围是指产品或者服务所应该包含的功能。产品范围是否完成，要根据产品是否满足了产品描述来判断。产品范围是项目范围的基础，产品范围的定义是产品要求的描述。

项目范围是指为了能够交付产品，项目所必须做的工作。项目范围的定义是产生项目管理计划的基础。判断项目范围是否完成，要以范围基准来衡量。项目的范围基准是经过批准的项目范围说明书、WBS 和 WBS 词典。

产品范围描述是项目范围说明书的重要组成部分，因此，产品范围变更后，首先受到影响的是项目的范围。

WBS 将项目整体或者主要的可交付成果分解成容易管理、方便控制的若干个子项目或者工作包，子项目需要继续分解为工作包，持续这个过程，直到整个项目部分解为可管理的工作包，这些工作包的总和是项目的所有工作范围。最普通的 WBS 如下表所示：

工作编号	工作任务	工 期	负 责 人
0	远程教育项目	8 月	吴函
1	硬件	2 月	何小波
2	第三方软件	2 月	王方
3	系统功能	5 月	张必胜
3.1	设备管理	1 月	桂波阳
3.2	维护管理	1 月	周瑞
3.3	工单管理	1 月	谢敏波

工作编号	工作任务	工 期	负 责 人
3.3.1	模块设计	5 天	郑嘉
3.3.2	代码编制	5 天	马小虎
3.3.3	单元测试	10 天	汪角春
3.3.4	功能测试	5 天	左林标
3.3.5	验证测试	5 天	赵晓
3.4	采购管理	1 月	胡海涛
3.5	库存管理	1 月	王敏捷
4	系统接口	1 月	李鸿海
5	现场实施	1 月	李智

2 进度管理

进度管理就是采用科学的方法，确定进度目标，编制进度计划和资源供应计划，进行进度控制，在与质量、成本目标协调的基础上，实现工期目标。

具体来说，包括以下过程：

- (1) 活动定义：确定完成项目各项可交付成果而需要开展的具体活动。
- (2) 活动排序：识别和记录各项活动之间的先后关系和逻辑关系。
- (3) 活动资源估算：估算完成各项活动所需要的资源类型和效益。
- (4) 活动历时估算：估算完成各项活动所需要的时间。
- (5) 进度计划编制：分析活动顺序、活动持续时间、资源要求和进度制约因素，制订项目进度计划。
- (6) 进度控制：根据进度计划开展项目活动，如果发现偏差，则分析原因或进行调整。

进行活动资源估算的方法主要有专家判断法、替换方案的确定、公开的估算数据、估算软件 and 自下而上的估算。

(1) 专家判断法。专家判断法通常是由项目管理专家根据以往类似项目经验的验和对本项目的判断，经过周密思考，进行合理预测，从而估算出项目资源。

(2) 替换方案的确定。资源估算是为了给项目预算明确空间，为早期的资源筹备提供数据，如果某项活动存在替代方案，或提供的资源有替代支持可能，则需要明确声明。

(3) 公开的估算数据。有些公司会定期地公开一些生产率或人工费率数据，其中包括很多国家和地区的劳动力交易、材料和设备信息。

(4) 估算软件。依靠软件的强大功能，可以定义资源可用性、费率，以及不同的资源日历。

(5) 自下而上的估算。把复杂的活动分解为更小的工作，以便于资源估算。将每项工作所需要的资源估算出来，然后汇总即是整个活动所需要的资源数量。

COCOMO 模型：常见的软件规模估算方法。常用的代码行分析方法作为其中一种度量估计单位，以代码行数估算出每个程序员工作量，累加得软件成本。

模型按其详细程度可以分为三级：

(1) 基本 COCOMO 模型是一个静态单变量模型，它用一个以已估算出来的原代码行数(LOC)为自变量的经验函数计算软件开发工作量。

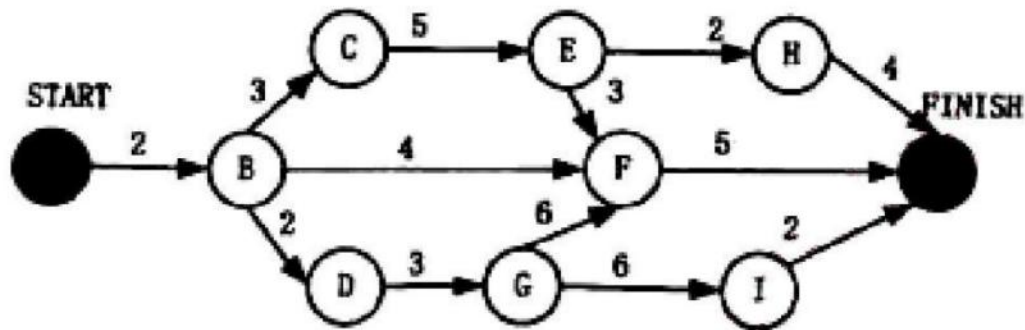
(2) 中间 COCOMO 模型在基本 COCOMO 模型的基础上，再用涉及产品、硬件、人员、项目等方面的影响因素调整工作量的估算。

(3) 详细 COCOMO 模型包括中间 COCOMO 模型的所有特性，将软件系统模型分为系统、子系统和模块 3 个层次，更进一步考虑了软件工程中每一步骤（如分析、设计）的影响。

COCOMO II 模型：COCOMO 的升级，也是以软件规模作为成本的主要因素，考虑多个成本驱动因子。该方法包括三个阶段模型，即应用组装模型、早期设计阶段模型和体系结构阶段模型。包含三种不同规模估算选择：对象点、功能点和代码行。应用组装模型使用的是对象点；早期设计阶段模型使用的是功能点，功能点可以转换为代码行。

进度安排的常用图形描述方法有 Gantt 图（甘特图）和项目计划评审技术（Program Evaluation & Review Technique, PERT）图。





关键路径法

关键路径：是项目的最短工期，但却是从开始到结束时间最长的路径。进度网络图中可能有多条关键路径，因为活动会变化，因此关键路径也在不断变化中。

关键活动：关键路径上的活动，最早开始时间=最晚开始时间。

通常，每个节点的活动会有如下几个时间：

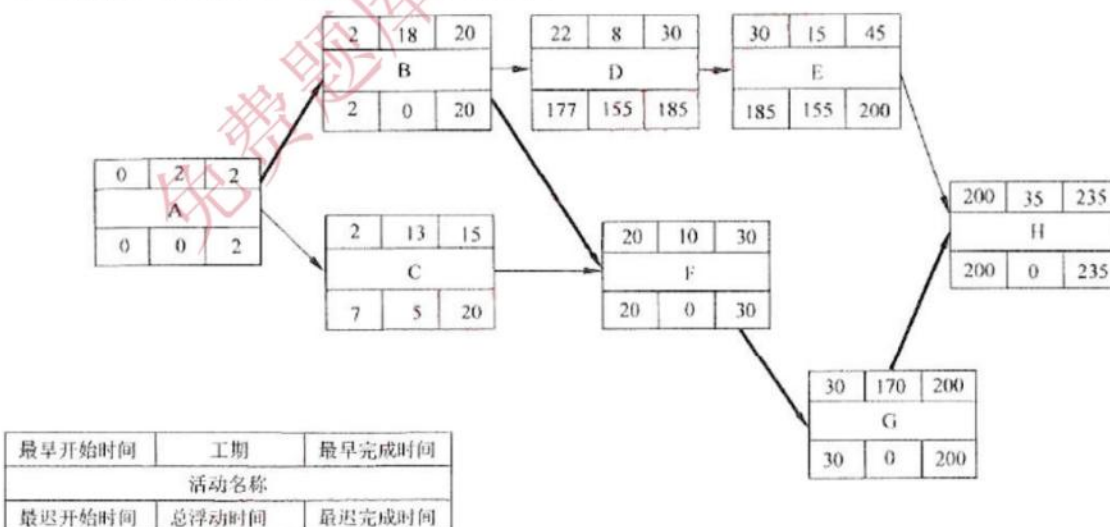
- (1) 最早开始时间 (ES)，某项活动能够开始的最早时间。
- (2) 最早结束时间 (EF)，某项活动能够完成的最早时间。 $EF=ES+工期$
- (3) 最迟结束时间 (LF)。为了使项目按时完成，某项活动必须完成的最迟时间。
- (4) 最迟开始时间 (LS)。为了使项目按时完成，某项活动必须开始的最迟时间。 $LS=LF-工期$

这几个时间通常作为每个节点的组成部分，如图所示：

顺推：最早开始 ES=所有前置活动最早完成 EF 的最大值；最早完成 EF=最早开始 ES+持续时间。

逆推：最晚完成 LF=所有后续活动最晚开始 LS 的最小值；最晚开始 LS=最晚完成 LF-持续事件。

下面教材原图有误，视频里有详细勘误。



总浮动时间（松弛时间）：在不延误项目完工时间且不违反进度制约因素的前提下，活动可以从最早开始时间推迟或拖延的时间量，就是该活动的进度灵活性。正常情况下，关键活动的总浮动时间为零。

总浮动时间=最迟开始LS-最早开始ES 或 最迟完成LF-最早完成EF 或 关键路径-非关键路径时长。

自由浮动时间：是指在不延误任何紧后活动的最早开始时间且不违反进度制约因素的前提下，活动可以从最早开始时间推迟或拖延的时间量。

自由浮动时间=紧后活动最早开始时间的最小值-本活动的最早完成时间。

3 成本管理

项目成本管理是在整个项目的实施过程中，为确保项目在批准的预算条件下尽可能保质按期完成，而对所需的各个过程进行管理与控制。

项目成本管理包括成本估算、成本预算和成本控制三个过程。

（1）成本估算是对完成项目所需成本的估计和计划，是项目计划中的一个重要的、关键的、敏感的部分；成本估算主要靠分解和类推的手段进行，基本估算方法分为三类：自顶向下的估算、自底向上的估算和差别估算法。

（2）成本预算是把估算的总成本分配到项目的各个工作包，建立成本基准计划以衡量项目绩效；应急储备和管理储备。

（3）成本控制保证各项工作在各自的预算范围内进行。

成本的类型

（1）可变成本：随着生产量、工作量或时间而变的成本为可变成本。可变成本又称变动成本。

（2）固定成本：不随生产量、工作量或时间的变化而变化的非重复成本为固定成本。

（3）直接成本：直接可以归属于项目工作的成本为直接成本。如项目团队差旅费、工资、项目使用的物料及设备使用费等。

（4）间接成本：来自一般管理费用科目或几个项目共同担负的项目成本所分摊给本项目的费用，就形成了项目的间接成本，如税金、额外福利和保卫费用等。

（5）机会成本：是利用一定的时间或资源生产一种商品时，而失去的利用这些资源生产其他最佳替代品的机会就是机会成本，泛指一切在做出选择后其中一个最大的损失。

（6）沉没成本：是指由于过去的决策已经发生了的，而不能由现在或将来的任何决策改变的成本。沉没成本是一种历史成本，对现有决策而言是不可控成本，会很大程度上影响人们的行为方式与决策，在投资决策时应排除沉没成本的干扰。

学习曲线：重复生成产品时，产品的单位成本会随着产量的扩大呈现规律性递减。估算成本时，也要考虑此因素。

4 软件配置管理

配置管理是为了系统地控制配置变更，在系统的整个生命周期中维持配置的完整性和可跟踪性，而标识系统在不同时间点上配置的学科。在 GB/T11457-2006 中将“配置管理”正式定义为：“应用技术的和管理的指导和监控方法以标识和说明配置项的功能和物理特征，控制这些特征的变更，记录和报告变更处理和实现状态并验证与规定的需求的遵循性。”

配置管理包括 6 个主要活动：制订配置管理计划、配置标识、配置控制、配置状态报告、配置审计、发布管理和交付。

配置项：GB/T11457-2006 对配置项的定义为：“为配置管理设计的硬件、软件或二者的集合，在配置管理过程中作为一个单个实体来对待”。

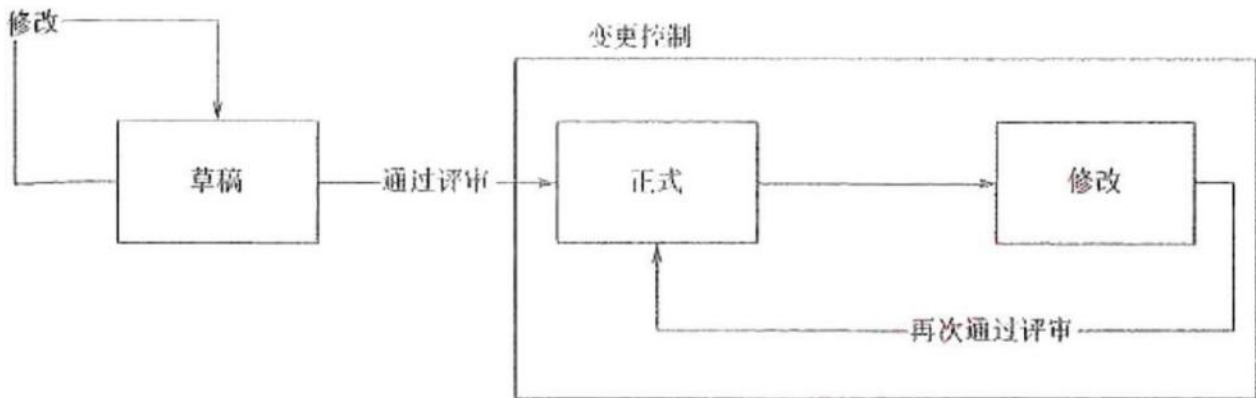
以下内容都可以作为配置项进行管理：外部交付的软件产品、数据、指定的内部软件工作产品和数据、指定的用于创建或支持软件产品的支持工具、供方/供应商提供的软件和客户提供的设备/软件。典型配置项包括项目计划书、需求文档、设计文档、源代码、可执行代码、测试用例、运行软件所需的各种数据，它们经评审和检查通过后进入配置管理。

每个配置项的主要属性有：名称、标识符、文件状态、版本、作者、日期等。

配置项可以分为基线配置项和非基线配置项两类，例如，基线配置项可能包括所有的设计文档和源程序等；非基线配置项可能包括项目的各类计划和报告等。

所有配置项的操作权限应由 CMO（配置管理员）严格管理，基本原则是：基线配置项向开发人员开放读取的权限；非基线配置项向 PM、CCB 及相关人员开放。

配置项的状态可分为“草稿”“正式”和“修改”三种。配置项刚建立时，其状态为“草稿”。配置项通过评审后，其状态变为“正式”。此后若更改配置项，则其状态变为“修改”。当配置项修改完毕并重新通过评审时，其状态又变为“正式”。如图所示



配置项版本号

(1) 处于“草稿”状态的配置项的版本号格式为 0.YZ, YZ 的数字范围为 01~99。随着草稿的修正, YZ 的取值应递增。YZ 的初值和增幅由用户自己把握。

(2) 处于“正式”状态的配置项的版本号格式为 X.Y, X 为主版本号, 取值范围为 1~9。Y 为次版本号, 取值范围为 0~9。

配置项第一次成为“正式”文件时, 版本号为 1.0。

如果配置项升级幅度比较小, 可以将变动部分制作成配置项的附件, 附件版本依次为 1.0, 1.1…。当附件的变动积累到一定程度时, 配置项的 Y 值可适量增加, Y 值增加一定程度时, X 值将适量增加。当配置项升级幅度比较大时, 才允许直接增大 X 值。

(3) 处于“修改”状态的配置项的版本号格式为 X.YZ。配置项正在修改时, 一般只增大 Z 值, X.Y 值保持不变。当配置项修改完毕, 状态成为“正式”时, 将 Z 值设置为 0, 增加 X.Y 值。参见上述规则 (2)。

配置项版本管理: 在项目开发过程中, 绝大部分的配置项都要经过多次的修改才能最终确定下来。对配置项的任何修改都将产生新的版本。由于我们不能保证新版本一定比旧版本“好”, 所以不能抛弃旧版本。版本管理的目的是按照一定的规则保存配置项的所有版本, 避免发生版本丢失或混淆等现象, 并且可以快速准确地查找到配置项的任何版本。

配置基线 (常简称为基线) 由一组配置项组成, 这些配置项构成一个相对稳定的逻辑实体。基线中的配置项被“冻结”了, 不能再被任何人随意修改。对基线的变更必须遵循正式的变更控制程序。

基线通常对应于开发过程中的里程碑, 一个产品可以有多个基线, 也可以只有一个基线。交付给外部顾客的基线一般称为发行基线 (Release), 内部开发使用的基线一般称为构造基线 (Build)。

一组拥有唯一标识号的需求、设计、源代码文卷以及相应的可执行代码、构造文卷和用户文档构成一条基线。产品的一个测试版本 (可能包括需求分析说明书、概要设计说明书、详细设计说明书、已编译的可执行代码、测试大纲、测试用例、使用手册等) 是基线的一个例子。

对于每一个基线，要定义下列内容：建立基线的事件、受控的配置项、建立和变更基线的程序、批准变更基线所需的权限。在项目实施过程中，每个基线都要纳入配置控制，对这些基线的更新只能采用正式的变更控制程序。

建立基线还可以有如下好处：

(1) 基线为开发工作提供了一个定点和快照。

(2) 新项目可以在基线提供的定点上建立。新项目作为一个单独分支，将与随后对原始项目（在主要分支上）所进行的变更进行隔离。

(3) 当认为更新不稳定或不可信时，基线为团队提供一种取消变更的方法。

(4) 可以利用基线重新建立基于某个特定发布版本的配置，以重现已报告的错误。

配置库存放配置项并记录与配置项相关的所有信息，是配置管理的有力工具。主要作用：

(1) 记录与配置相关的所有信息，其中存放受控的软件配置项是很重要的内容。

(2) 利用库中的信息可评价变更的后果，这对变更控制有着重要的意义。

(3) 从库中可提取各种配置管理过程的管理信息。

使用配置库可以帮助配置管理员把信息系统开发过程的各种工作产品，包括半成品或阶段产品和最终产品管理得井井有条，使其不致管乱、管混、管丢。

配置库可以分开发库、受控库、产品库 3 种类型。

(1) 开发库，也称为动态库、程序员库或工作库，用于保存开发人员当前正在开发的配置实体，如：新模块、文档、数据元素或进行修改的已有元素。动态中的配置项被置于版本管理之下。动态库是开发人员的个人工作区，由开发人员自行控制。库中的信息可能有较为频繁的修改，只要开发库的使用者认为有必要，无需对其进行配置控制，因为这通常不会影响到项目的其他部分。可以任意的修改。

(2) 受控库，也称为主库，包含当前的基线加上对基线的变更。受控库中的配置项被置于完全的配置管理之下。在信息系统开发的某个阶段工作结束时，将当前的工作产品存入受控库。可以修改，需要走变更流程。

(3) 产品库，也称为静态库、发行库、软件仓库，包含已发布使用的各种基线的存档，被置于完全的配置管理之下。在开发的信息系统产品完成系统测试之后，作为最终产品存入产品库内，等待交付用户或现场安装。一般不再修改，真要修改的话需要走变更流程。

5 质量管理

质量是软件产品特性的综合，表示软件产品满足明确（基本需求）或隐含（期望需求）要求的能力。质量管理是指确定质量方针、目标和职责，并通过质量体系中的质量计划、质量控制、质量保证和质量改进来使其实现的所有管理职能的全部活动；

主要包括以下过程：

（1）质量规划：识别项目及其产品的质量要求和标准，并书面描述项目将如何达到这些要求和标准的过程。

（2）质量保证：一般是每隔一定时间（例如，每个阶段末）进行的，主要通过系统的质量审计（软件评审）和过程分析来保证项目的质量。

（3）质量控制：实时监控项目的具体结果，以判断它们是否符合相关质量标准，制订有效方案，以消除产生质量问题的原因。

信息技术 软件产品评价 质量特性及其使用指南 GB/T 16260—2002

质量特性及定义↵	质量子特性及定义↵
功能性：一组功能及其指定的性质有关的一组属性↵	适合性：与规定任务能否提供一组功能及这组功能的适合程度有关的软件属性↵
	准确性：与能否得到正确或相符的结果或效果有关的软件属性↵
	互用性/互操作性：与其他指定系统进行交互的能力有关的软件属性↵
	依从性：使软件遵循有关标准、法律、法规及类似规定的软件属性↵
	安全性：防止对程序及数据的非授权的故意或意外访问的能力↵
可靠性：在规定的一段时间和条件下，软件维持其性能水平有关的一组软件属性↵	成熟性：与由软件故障引起失效的频度有关的软件属性↵
	容错性：与在与软件故障或违反指定接口情况下，维持规定的性能水平的能力有关的软件属性↵
	易恢复性：与在失效发生后，重新建立其性能水平、恢复直接受影响数据的能力，以及为达到此目的所需的时间和努力有关的软件属性↵
可用性：与使用的难易程度及规定或隐含用户对使用方式所做的评价有关的软件属性↵	易理解性：与用户为认识逻辑概念及其应用范围所花的努力有关的软件属性↵
	易学性：与用户为学习使用该软件系统所花的努力有关的软件属性↵
	易操作性：与用户为操作和运行控制所花努力有关的软件属性↵

效率：与在规定条件下，软件的性能水平和所用资源之间的关系有关的一组软件属性	时间特性：与软件执行其功能时相应和处理时间以及吞吐量有关的软件属性
	资源特性：与在软件执行其功能时，所使用的资源量及使用资源、持续时间有关的软件属性
可维护性：与进行指定的修改所需的努力有关的一组软件属性	易分析性：与为诊断缺陷或失效原因、判定待修改的部分所需努力有关的软件属性
	可修改性：与进行修改、排除错误或适应环境变化所需努力有关的软件属性
	稳定性：与修改所造成的未预料结果的风险有关的软件属性
	可测试性：与确认已修改软件所需的努力有关的软件属性
可移植性：与软件可从某一环境转移到另一环境的能力有关的一组软件属性	适应性：与软件无需采用有别于为该软件准备的活动或手段就可能适应不同的规定环境有关的软件属性
	易安装性：与在指定环境下安装软件所需努力有关的软件属性
	一致性（遵循性）：使软件遵循与可移植有关的标准或约定的软件属性
	可替换性：软件在特定环境中用来替代指定的其他软件的可能性和难易程度

McCall 质量模型



软件评审

质量两个必要条件：设计的规格说明书符合用户标准，称为设计质量。

程序按照设计规格说明书所规定的情况正确执行，称为程序质量。

软件容错技术：容错就是软件遇到错误的处理能力，实现容错的手段主要是冗余，包括下面四种冗余技术：

结构冗余：分为静态、动态、混合冗余三种，当错误发生时对错误进行备份处理。

信息冗余：为检错和纠错在数据中加上一段额外的信息，例如校验码原理。

时间冗余：遇到错误时重复执行，例如回滚，重复执行还有错，则转入错误处理逻辑。

冗余附加技术：是指为实现结构、信息和时间冗余技术所需的资源和技术，包括程序、指令、数据、存放和调动它们的空间和通道等。在屏蔽硬件错误的容错技术中，

6 风险管理

风险管理就是要对项目风险进行认真的分析和科学的管理，这样，是能够避开不利条件、少受损失、取得预期的结果并实现项目目标的，能够争取避免风险的发生或尽量减小风险发生后的影响。但是，完全避开或消除风险，或者只享受权益而不承担风险是不可能的。

风险管理计划编制：如何安排与实施项目的风险管理，制定下列各步的计划。

风险识别：识别出项目中已知和可预测的风险，确定风险的来源、产生的条件、描述风险的特征以及哪些项目可以产生风险，形成一个风险列表。

风险定性分析：对已经识别的风险进行排序，确定风险可能性与影响、确定风险优先级、确定风险类型。

风险定量分析：进一步了解风险发生的可能性具体由多大，后果具体由多严重。包括灵敏度分析、期望货币价值分析、决策树分析、蒙特卡罗模拟。

风险应对计划编制：对每一个识别出来的风险来分别制定应对措施，这些措施组成的文档称为风险应对计划。包括消极风险（避免策略、转移策略、减轻策略）；积极风险（开拓、分享、强大）。

风险监控：监控风险计划的执行，检测残余风险，识别新的风险，保证风险计划的执行，并评价这些计划对减少风险的有效性。

项目风险：作用于项目上的不确定的事件或条件，既可能产生威胁，也可能带来机会。

通过积极和合理的规划，超过 90% 的风险都可以进行提前应对和管理。

风险应该尽早识别出来，高层次风险应记录在章程里。

应由对风险最有控制力的一方承担相应的风险。

承担风险程度与所得回报相匹配原则，承担的风险要有上限。

风险的属性：

(1) 随机性：风险事件发生及其后果都具有偶然性（双重偶然），遵循一定的统计规律。

(2) 相对性：风险是相对项目活动主体而言的。承受力不同，影响不同。风险承受力影响因素：收益大小（收益越大，越愿意承担风险）；投入大小（投入越大，承受能力越小）；主体的地位和资源（级别高的人能承担较大的风险）。

(3) 风险的可变性：条件变化，会引起风险变化。包括性质、后果的变化，以及出现新风险。

风险的分类：

按照后果的不同，风险可划分为纯粹风险（无任何收益）和投机风险（可能带来收益）。

按风险来源划分，自然风险（天灾）和人为风险（人的活动，又可分为行为风险、经济风险、技术风险、政治和组织风险等）。

按是否可管理划分，可管理（如内部多数风险）和不可管理（如外部政策），也要看主体管理水平。

按影响范围划分，局部风险（非关键路径活动延误）和总体风险（关键路径活动延误）。

按后果承担者划分：业主、政府、承包商、投资方、设计单位、监理单位、保险公司等。

按可预测性划分：已知风险（已知的进度风险）、可预测风险（可能服务器故障）、不可预测风险（地震、洪水、政策变化等）。

在信息系统项目中，从宏观上来看，风险可以分为项目风险、技术风险和商业风险。

项目风险是指潜在的预算、进度、个人（包括人员和组织）、资源、用户和需求方面的问题，以及它们对项目的影响。项目复杂性、规模和结构的不确定性也构成项目的（估算）风险因素。项目风险威胁到项目计划，一旦项目风险成为现实，可能会拖延项目进度，增加项目的成本。

技术风险是指潜在的设计、实现、接口、测试和维护方面的问题。此外，规格说明的多义性、技术上的不确定性、技术陈旧、最新技术（不成熟）也是风险因素。技术风险威胁到待开发系统的质量和预定的交付时间。如果技术风险成为现实，开发工作可能会变得很困难或根本不可能。

商业风险威胁到待开发系统的生存能力，主要有以下 5 种不同的商业风险：

- （1）市场风险。开发的系统虽然很优秀但不是市场真正所想要的。
- （2）策略风险。开发的系统不再符合企业的信息系统战略。
- （3）销售风险。开发了销售部门不清楚如何推销的系统。
- （4）管理风险。由于重点转移或人员变动而失去上级管理部门的支持。
- （5）预算风险。开发过程没有得到预算或人员的保证。

7 组织结构

组织结构模式：项目型（项目经理绝对领导）、职能型（部门领导为主）、矩阵型（二者结合，既有项目经理也有部门领导，但权利分割不同）。

程序设计小组的组织方式：

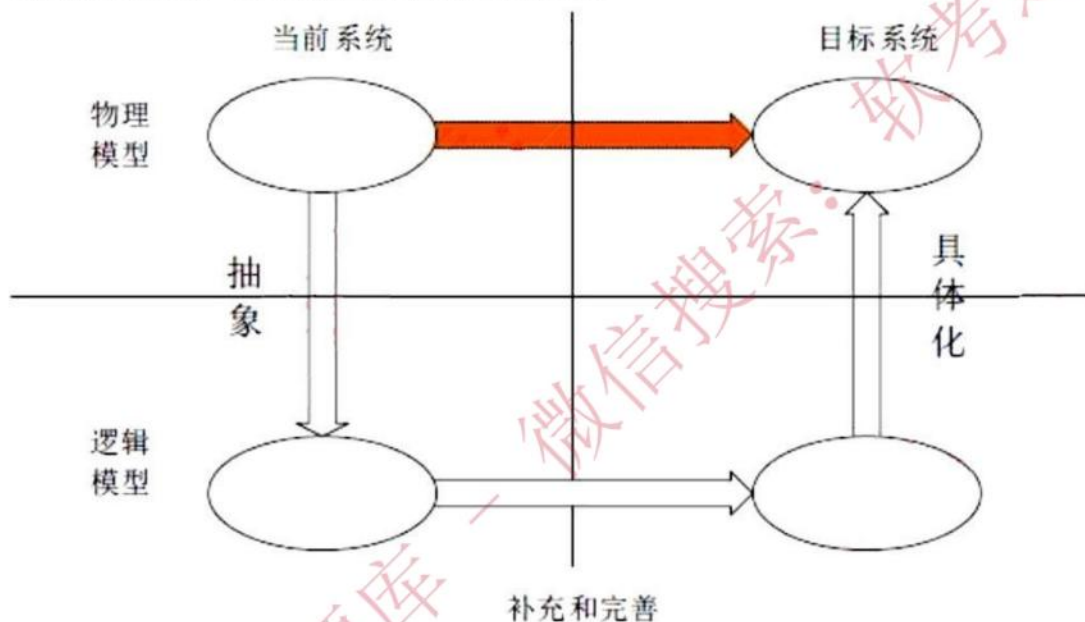
- （1）主程序员制小组（主程序员全权负责，后援工程师必要时能替代主程序员，适合大规模项目）
- （2）民主制小组（也即无主程序员小组，成员之间地位平等，任何决策都是全员参与投票，适合于项目规模小，开发人员少，采用新技术和确定性较小的项目）
- （3）层次式小组（两个层次，一名组长领导若干个高级程序员，每个高级程序员领导若干个程序员）。

十二 结构化开发方法

1 系统分析与设计概述

系统分析的主要任务是对现行系统进一步详细调查，将调查中所得到的文档资料集中，对组织内部整体管理状况和信息处理过程进行分析，为系统开发提供所需的资料，并提交系统方案说明书。系统分析侧重于从业务全过程的角度进行分析，主要内容有业务和数据的流程是否通畅、是否合理；数据、业务过程和组织管理之间的关系；原系统管理模式改革和新系统管理方法的实现是否具有可行性等。

系统分析过程一般按如图所示的逻辑进行：



- (1) 认识、理解当前的现实环境，获得当前系统的“物理模型”。
- (2) 从当前系统的“物理模型”抽象出当前系统的“逻辑模型”。
- (3) 对当前系统的“逻辑模型”进行分析和优化，建立目标系统的“逻辑模型”。
- (4) 对目标系统的逻辑模型具体化（物理化），建立目标系统的物理模型。

系统开发的目的是把现有系统的物理模型转化为目标系统的物理模型，即图中双虚线所描述的路径，而系统分析阶段的结果是得到目标系统的逻辑模型。逻辑模型反映了系统的功能和性质，而物理模型反映的是系统的某一种具体实现方案。

系统设计基本原理：抽象、模块化、信息隐蔽、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，并且与其他模块之间的联系简单。衡量模块独立程度的标准有两个：耦合性和内聚性。模块的设计要求独立性高，就必须高内聚，低耦合，

内聚是指一个模块内部功能之间的相关性，耦合是指多个模块之间的联系，内聚程度从低到高如下表所示：

内聚分类	定义	记忆关键字
偶然内聚	一个模块内的各处理元素之间没有任何联系	无直接关系
逻辑内聚	模块内执行若干个逻辑上相似的功能，通过参数确定该模块完成哪一个功能	逻辑相似、参数决定
时间内聚	把需要同时执行的动作组合在一起形成的模块。	同时执行
过程内聚	一个模块完成多个任务，这些任务必须按指定的过程执行	指定的过程顺序
通信内聚	模块内的所有处理元素都在同一个数据结构上操作，或者各处理使用相同的输入数据或者产生相同的输出数据	相同数据结构、相同输入输出
顺序内聚	一个模块中的各个处理元素都密切相关于同一功能且必须顺序执行，前一个功能元素的输出就是下一个功能元素的输入	顺序执行、输入为输出
功能内聚	最强的内聚，模块内的所有元素共同作用完成一个功能，缺一不可	共同作用、缺一不可

耦合程度从低到高如下所示：

耦合分类	定义	记忆关键字
无直接耦合	两个模块之间没有直接的关系，它们分别从属于不同模块的控制与调用，不传递任何信息	无直接关系
数据耦合	两个模块之间有调用关系，传递的是简单的数据值，相当于高级语言中的值传递	传递数据值调用
标记耦合	两个模块之间传递的是数据结构	传递数据结构
控制耦合	一个模块调用另一个模块时，传递的是控制变量，被调用模块通过该控制变量的值有选择的执行模块内的某一功能	控制变量、选择执行某一功能

外部耦合	模块间通过软件之外的环境联合（如 I/O 将模块耦合到特定的设备、格式、通信协议上）时。	软件外部环境
公共耦合	通过一个公共数据环境相互作用的那些模块间的耦合	公共数据结构
内容耦合	当一个模块直接使用另一个模块的内部数据，或通过非正常入口转入另一个模块内部时。	模块内部关联

系统总体结构设计是要根据系统分析的要求和组织的实际情况对新系统的总体结构形式和可利用的资源进行大致设计，这是一种宏观、总体上的设计和规划。

系统结构设计原则：

- (1) 分解-协调原则。
- (2) 自顶向下的原则。
- (3) 信息隐蔽、抽象的原则。
- (4) 一致性原则。
- (5) 明确性原则。
- (6) 模块之间的耦合尽可能小，模块的内聚度尽可能高。
- (7) 模块的扇入系数和扇出系数要合理。
- (8) 模块的规模适当。

子系统划分的原则：

- (1) 子系统要具有相对独立性。
- (2) 子系统之间数据的依赖性尽量小。
- (3) 子系统划分的结果应使数据冗余较小。
- (4) 子系统的设置应考虑今后管理发展的需要。
- (5) 子系统的划分应便于系统分阶段实现。
- (6) 子系统的划分应考虑到各类资源的充分利用。

子系统结构设计的任务是确定划分后的子系统模块结构，并画出模块结构图。在这个过程中必须考虑以下几个问题。

- (1) 每个子系统如何划分成多个模块。

- (2) 如何确定子系统之间、模块之间传送的数据及其调用关系。
- (3) 如何评价并改进模块结构的质量。
- (4) 如何从数据流图导出模块结构图。

系统模块结构设计

模块是组成系统的基本单位，它的特点是可以组合、分解和更换。系统中的任何一个处理功能都可以看成是一个模块。根据功能具体化程度的不同，模块可以分为逻辑模块和物理模块。

一个模块应具备以下 4 个要素。

- (1) 输入和输出。
- (2) 处理功能。指模块把输入转换成输出所做的工作。
- (3) 内部数据。指仅供该模块本身引用的数据。
- (4) 程序代码。指用来实现模块功能的程序。

前两个要素是模块外部特性，反映了模块的外貌。后两个要素是模块的内部特性。

模块结构图为了保证系统设计工作的顺利进行，结构设计应遵循以下原则。

- (1) 所划分的模块其内部的凝聚性要强，模块之间的联系要少，即模块具有较强的独立性。
- (2) 模块之间的连接只能存在上下级之间的调用关系，不能有同级之间的横向联系。
- (3) 整个系统呈树状结构，不允许网状结构或交叉调用关系出现。
- (4) 所有模块（包括后继 IPO 图）都必须严格地分类编码并建立归档文件。

模块结构图主要关心的是模块的外部属性，即上下级模块、同级模块之间的数据传递和调用关系，并不关心模块的内部。

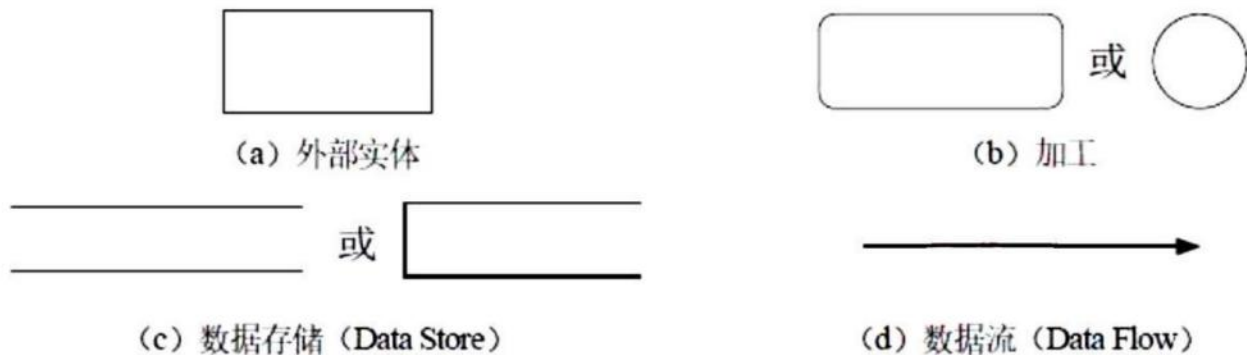
2 结构化开发方法

结构化分析与设计方法是一种面向数据流的传统软件开发方法，它以数据流为中心构建软件的分析模型和设计模型。结构化分析（Structured Analysis, SA）、结构化设计（Structured Design, SD）和结构化程序设计（Structured Programming Design, SPD）构成了完整的结构化方法

结构化方法的分析结果由以下几部分组成：一套分层的数据流图、一本数据词典、一组小说明（也称加工逻辑说明）、补充材料。

数据流图 DFD

基本图形元素：外部实体、加工、数据存储、数据流。



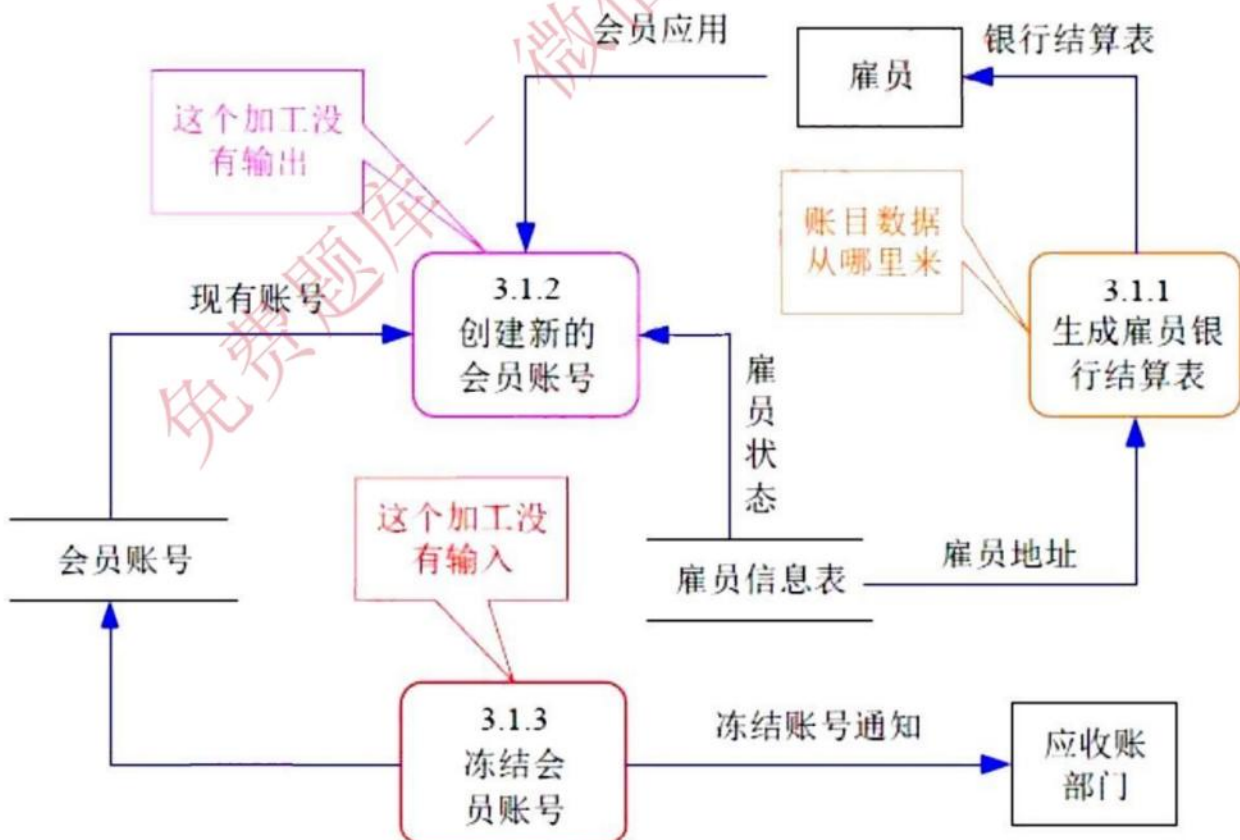
1) 数据流：由一组固定成分的数据组成，表示数据的流向。在 DFD 中，数据流的流向可以有以下几种：从一个加工流向另一个加工；从加工流向数据存储（写）；从数据存储流向加工（读）；从外部实体流向加工（输入）；从加工流向外部实体（输出）。

2) 加工：描述了输入数据流到输出数据流之间的变换，也就是输入数据流经过什么处理后变成了输出数据流。数据流图中常见的三种错误如图所示：

加工 3.1.2 有输入但是没有输出，称之为“黑洞”。

加工 3.1.3 有输出但没有输入，称之为“奇迹”。

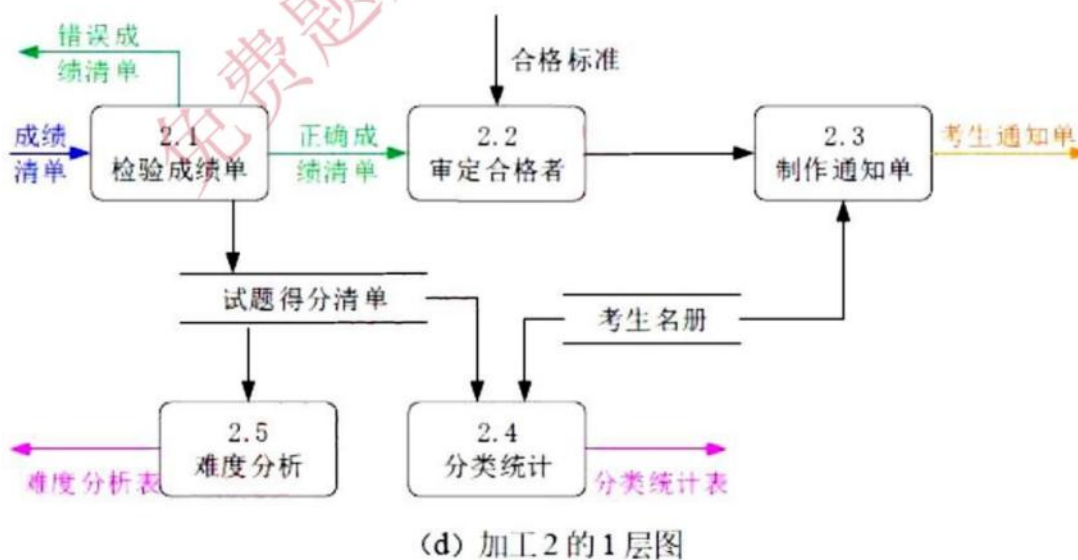
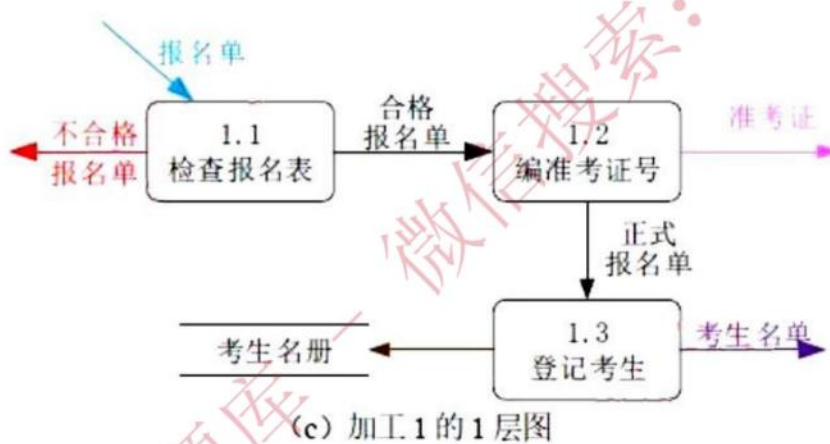
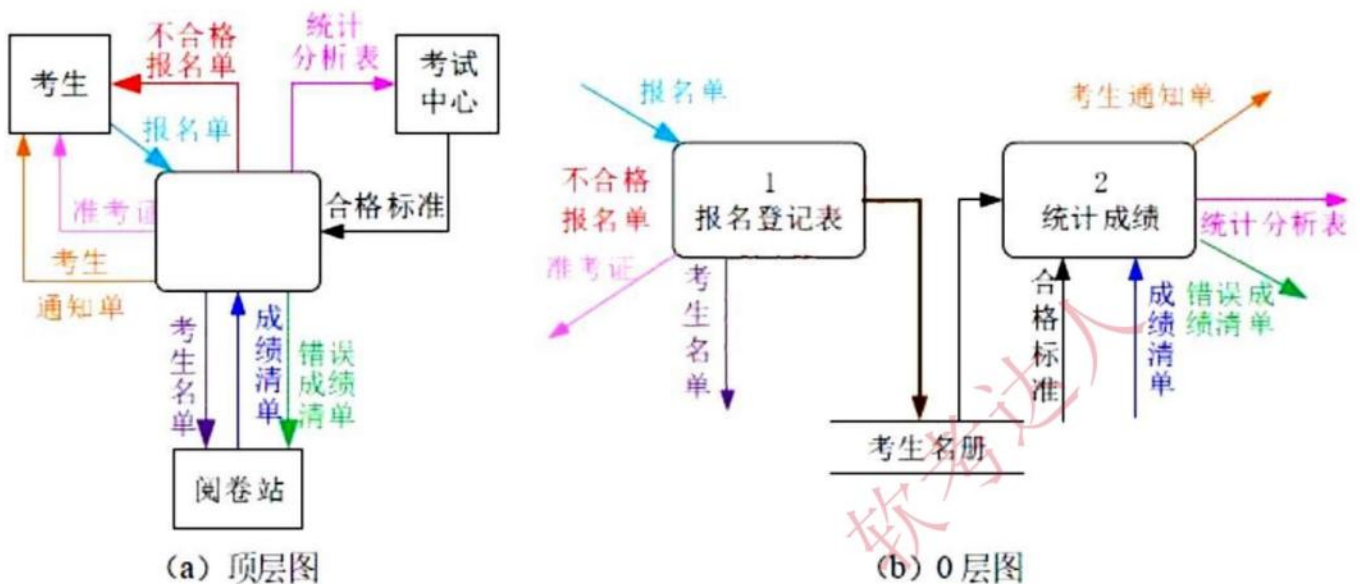
加工 3.1.1 中输入不足以产生输出，我们称之为“灰洞”。这有几种可能的原因：一个错误的命名过程；错误命名的输入或输出；不完全的事实。灰洞是最常见的错误，也是最使人为难的错误。一旦数据流图交给了程序员，到一个加工的输入数据流必须足以产生输出数据流。



3) 数据存储：用来存储数据。在软件系统中还常常要把某些信息保存下来以供以后使用。

4) 外部实体（外部主体）：是指存在于软件系统之外的人员或组织，它指出系统所需数据的发源地（源）和系统所产生的数据的归宿地（宿）。

分层数据流图：



数据字典 DD

数据流图描述了系统的分解，但没有对图中各成分进行说明。数据字典就是为数据流图中的每个数据流、文件、加工，以及组成数据流或文件的数据项做出说明。

数据字典有以下 4 类条目：数据流、数据项、数据存储和基本加工。

符号	含义	举例及说明
=	被定义为	
+	与	$x=a+b$ ，表示 x 由 a 和 b 组成
[... ...]	或	$x=[a b]$ ，表示 x 由 a 或 b 组成
{.....}	重复	$x=\{a\}$ ，表示 x 由 0 个或多个 a 组成

加工逻辑也称为“小说明”。常用的加工逻辑描述方法有结构化语言、判定表和判定树 3 种。

3 结构化设计方法

结构化设计（Structured Design，SD）方法是一种面向数据流的设计方法，它可以与 SA 方法衔接。结构化设计方法的基本思想是将系统设计成由相对独立、功能单一的模块组成的结构。

结构化设计方法中用结构图（Structure Chart）来描述软件系统的体系结构，指出一个软件系统由哪些模块组成，以及模块之间的调用关系。模块结构图是结构化设计的工具，由模块、调用、数据、控制和转接五种基本符号组成。

结构化设计大致可以分为两步进行，第一步是建立一个满足软件需求规约的初始结构图，第二步是对结构图进行改进。

结构化方法本质上是一种功能分解方法。在结构化设计时，可以将整个软件看作一个大的功能模块（结构图中的模块），通过功能分解将其分解成若干个较小的功能模块，每个较小的功能模块还可以进一步分解，直到得到一组不必再分解的模块（结构图中的底层模块）。当一个功能模块分解成若干个子功能模块时，该功能模块实际上就是根据业务流程调用相应的子功能模块，并根据其功能要求对子功能的结果进行处理，最终实现其功能要求。

功能模块的分解应满足自顶向下、逐步求精、信息隐蔽、高内聚低耦合等设计准则，模块的大小应适中。

结构化设计主要包括

- ①体系结构设计：定义软件的主要结构元素及其关系。
- ②数据设计：基于实体联系图确定软件涉及的文件系统的结构及数据库的表结构。
- ③接口设计：描述用户界面，软件和其他硬件设备、其他软件系统及使用人员的外部接口，以及各种构件之间的内部接口。

④过程设计：确定软件各个组成部分内的算法及内部数据结构，并选定某种过程的表达形式来描述各种算法。

4 WebApp 分析与设计

WebApp 是基于 web 的系统和应用。大多数 WebApp 采用敏捷开发过程模型进行开发。

WebApp 的特性：

(1) 网络密集性。WebApp 驻留在网络上，服务于不同客户整体的需求。网络提供开放的访问和通信（如 Internet）或者受限的访问和通信（如企业内联网）。

(2) 并发性。大量用户可能同时访问 WebApp。很多情况下最终用户的使用模式存在很大的差异。

(3) 无法预知的负载量。WebApp 的用户数量每天都可能有数量级的变化。例如，周一显示有 100 个用户使用系统，周四就有可能会有 10000 个用户。

(4) 性能。如果一位 WebApp 用户必须等待很长时间（访问、服务器端处理、客户端格式化显示），该用户就可能转向其他地方。

(5) 可用性。尽管期望百分之百的可用性是不切实际的，但是对于热门的 WebApp，用户通常要求能够 24/7/365（全天候）访问。

(6) 数据驱动。许多 WebApp 的主要功能是使用超媒体向最终用户提供文本、图片、音频及视频内容。除此之外，WebApp 还常被用来访问那些存储在 Web 应用环境之外的数据库中的信息。

4.1 WebApp 五种需求模型

1、内容模型：给出由 WebApp 提供的全部系列内容，包括文字、图形、图像、音频和视频。包含结构元素，为 WebApp 的内容需求提供了一个重要的视图。这些结构元素包含内容对象和所有分析类，在用户与 WebApp 交互时生成并操作用户可见的实体。

内容的开发可能发生在 WebApp 实现之前、构建之中、或者投入运行以后（全过程）。

内容对象：产品的文本描述、新闻文章、照片、视频等。

数据树：由多项内容对象和数据项组成的任何内容都可以生成数据树，是内容设计的基础，定义一种层级关系，并提供一种审核内容的方法，以便在开始设计前发现遗漏和不一致内容。

2、交互模型：描述了用户与 WebApp 采用了哪种交互方式。由一种或多种元素构成，包括用例、顺序图、状态图、用户界面原型等。

用例是交互分析的主要工具，方便客户理解系统的功能。

顺序图是交互分析中描述用户与系统进行交互的方式。用户按照已定顺序使用系统，完成相应的功能，如登录流程。

状态图是交互分析中对系统进行动态的描述。如状态的变化。

用户界面原型展现用户界面布局、内容、主要导航链接、实施的交互机制及用户 WebApp 的整体美观度。

3、功能模型：许多 WebApp 提供大量的计算和操作功能，这些功能与内容直接相关（既能使用又能生成内容，如统计报表）。这些功能常常以用户的交互活动为主要目标。

功能模型定义了将用于 WebApp 内容并描述其他处理功能的操作，这些处理功能不依赖于内容却是最终用户所必需的。

4、导航模型：为 WebApp 定义所有导航策略。考虑了每一类用户如何从一个 WebApp 元素（如内容对象）导航到另一个元素。

5、配置模型：描述 WebApp 所在的环境和基础设施。在必需考虑复杂配置体系结构的情况下，可以使用 UML 部署图。

4.2 WebApp 设计

1、架构设计：使用多层架构来构造，包括用户界面或展示层、基于一组业务规则来指导与客户端浏览器进行信息交互的控制器，以及可以包含 WebApp 的业务规则的内容或模型层，描述将以什么方式来管理用户交互、操作内部处理任务、实现导航及展示内容。

MVC（模型-视图-控制器）结构是 WebApp 基础结构模型之一，将 WebApp 功能及信息内容分离。

2、构件设计

WebApp 构件：定义良好的聚合功能，为最终用户处理内容或提供计算或处理数据；内容和功能的聚合包，提供最终用户所需要的功能。因此，WebApp 构件设计通常包括内容设计元素和功能设计元素。

构件级内容设计：关注内容对象，以及包装后展示给最终用户的方式，应该适合创建的 WebApp 特性。

构件级功能设计：将 WebApp 作为一系列构件加以交付，这些构件与信息体系结构并行开发，以确保一致性。

3、内容设计：着重于内容对象的表现和导航的组织，通常采用线性结构、网格结构、层次结构、网络结构四种结构及其组合。

4、导航设计：定义导航路径，使用户可以访问 WebApp 的内容和功能。

十三 面向对象技术

1 面向对象基础

面向对象的基本概念

(1) 对象：由数据及其操作所构成的封装体，是系统中用来描述客观事务的一个实体，是构成系统的一个基本单位。一个对象通常可以由对象名、属性和方法 3 个部分组成。

(2) 类：现实世界中实体的形式化描述，类将该实体的属性（数据）和操作（函数）封装在一起。对象是类的实例，类是对象的模板。

类可以分为三种：实体类、接口类（边界类）和控制类。实体类的对象表示现实世界中真实的实体，如人、物等。接口类（边界类）的对象为用户提供一种与系统合作交互的方式，分为人和系统两大类，其中人的接口可以是显示屏、窗口、Web 窗体、对话框、菜单、列表框、其他显示控制、条形码、二维码或者用户与系统交互的其他方法。系统接口涉及到把数据发送到其他系统，或者从其他系统接收数据。控制类的对象用来控制活动流，充当协调者。

(3) 抽象：通过特定的实例抽取共同特征以后形成概念的过程。它强调主要特征，忽略次要特征。一个对象是现实世界中一个实体的抽象，一个类是一组对象的抽象，抽象是一种单一化的描述，它强调给出与应用相关的特性，抛弃不相关的特性。

(4) 封装：是一种信息隐蔽技术，将相关的概念组成一个单元模块，并通过一个名称来引用。面向对象封装是将数据和基于数据的操作封装成一个整体对象，对数据的访问或修改只能通过对象对外提供的接口进行。

(5) 继承：表示类之间的层次关系（父类与子类），这种关系使得某类对象可以继承另外一类对象的特征，又可分为单继承和多继承。

(6) 多态：不同的对象收到同一个消息时产生完全不同的结果。包括参数多态（不同类型参数多种结构类型）、包含多态（父子类型关系）、过载多态（类似于重载，一个名字不同含义）、强制多态（强制类型转换）四种类型。多态由继承机制支持，将通用消息放在抽象层，具体不同的功能实现放在低层。

(7) 接口：描述对操作规范的说明，其只说明操作应该做什么，并没有定义操作如何做。

(8) 消息：体现对象间的交互，通过它向目标对象发送操作请求。

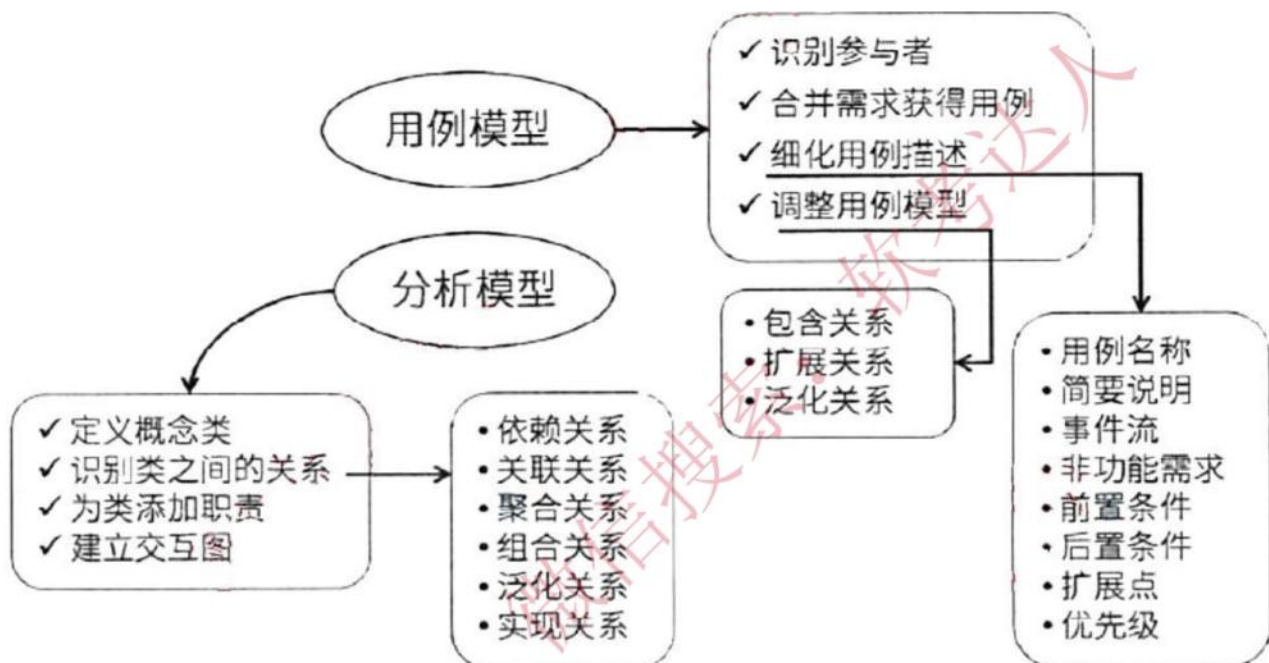
(9) 覆盖：子类在原有父类接口的基础上，用适合于自己要求的实现去置换父类中的相应实现。即在子类中重定义一个与父类同名同参的方法。

(10) 函数重载：与覆盖要区分开，函数重载与子类父类无关，且函数是同名不同参数。

(11) 绑定是一个把过程调用和响应调用所需要执行的代码加以结合的过程。在一般的程序设计语言中，绑定是在编译时进行的，叫作静态绑定。动态绑定则是在运行时进行的，因此，一个给定的过程调用和代码的结合直到调用发生时才进行。

面向对象的分析：是为了确定问题域，理解问题。包含五个活动：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。

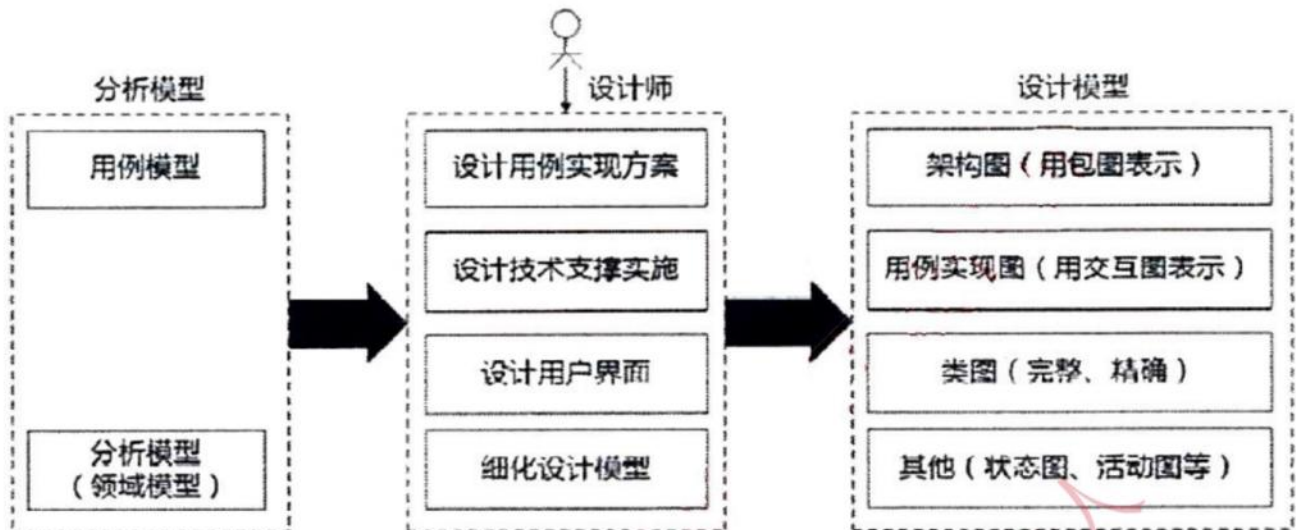
面向对象需求建模：



面向对象的设计：是设计分析模型和实现相应源代码，设计问题域的解决方案，与技术相关。

OOD 同样应遵循抽象、信息隐蔽、功能独立、模块化等设计准则。

面向对象的分析模型主要由顶层架构图、用例与用例图、领域概念模型构成；设计模型则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和用以描述流程化处理过程的活动图等。



面向对象的设计原则:

(1) 单一责任原则。就一个类而言,应该仅有一个引起它变化的原因。即,当需要修改某个类的时候原因有且只有一个,让一个类只做一种类型责任。

(2) 开放-封闭原则。软件实体(类、模块、函数等)应该是可以扩展的,即开放的;但是不可修改的,即封闭的。

(3) 里氏替换原则。子类型必须能够替换掉他们的基类型。即,在任何父类可以出现的地方,都可以用子类的实例来赋值给父类型的引用。

(4) 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。即,高层模块不应该依赖于低层模块,二者都应该依赖于抽象。

(5) 接口分离原则。不应该强迫客户依赖于它们不用的方法。接口属于客户,不属于它所在的类层次结构。即:依赖于抽象,不要依赖于具体,同时在抽象级别不应该有对于细节的依赖。这样做的好处就在于可以最大限度地应对可能的变化。

上述(1)~(5)是面向对象方法中的五大原则。除了这五大原则之外,Robert C. Martin提出的面向对象设计原则还包括以下几个。

(6) 重用发布等价原则。重用的粒度就是发布的粒度。

(7) 共同封闭原则。包中的所有类对于同一类性质的变化应该是共同封闭的。一个变化若对一个包产生影响,则将对包中的所有类产生影响,而对于其他的包不造成任何影响。

(8) 共同重用原则。一个包中的所有类应该是共同重用的。如果重用了包中的一个类,那么就要重用包中的所有类。

(9) 无环依赖原则。在包的依赖关系图中不允许存在环,即包之间的结构必须是一个直接的无环图形。

(10) 稳定依赖原则。朝着稳定的方向进行依赖。

(11) 稳定抽象原则。包的抽象程度应该和其稳定程度一致。

面向对象的测试:

一般来说,对面向对象软件的测试可分为下列 4 个层次进行。

(1) 算法层。测试类中定义的每个方法,基本上相当于传统软件测试中的单元测试。

(2) 类层。测试封装在同一个类中的所有方法与属性之间的相互作用。在面向对象软件中类是基本模块,因此可以认为这是面向对象测试中所特有的模块测试。

(3) 模板层。测试一组协同工作的类之间的相互作用,大体上相当于传统软件测试中的集成测试,但是也有面向对象软件的特点(例如,对象之间通过发送消息相互作用)。

(4) 系统层。把各个子系统组装成完整的面向对象软件系统,在组装过程中同时进行测试。

2 UML

UML (统一建模语言): 是一种可视化的建模语言,而非程序设计语言,支持从需求分析开始的软件开发的全过程。

从总体上来看, UML 的结构包括构造块、规则和公共机制三个部分。

(1) 构造块。UML 有三种基本的构造块,分别是事物(thing)、关系(relationship)和图(diagram)。事物是 UML 的重要组成部分,关系把事物紧密联系在一起,图是多个相互关联的事物的集合。

(2) 公共机制。公共机制是指达到特定目标的公共 UML 方法。

(3) 规则。规则是构造块如何放在一起的规定。

2.1 事物

结构事物: 模型的静态部分,如类、接口、用例、构件等;

行为事物: 模型的动态部分,如交互、活动、状态机;

分组事物: 模型的组织部分,如包;

注释事物: 模型的解释部分,依附于一个元素或一组元素之上对其进行约束或解释的简单符号。

部分事物 UML 图形如下所示:

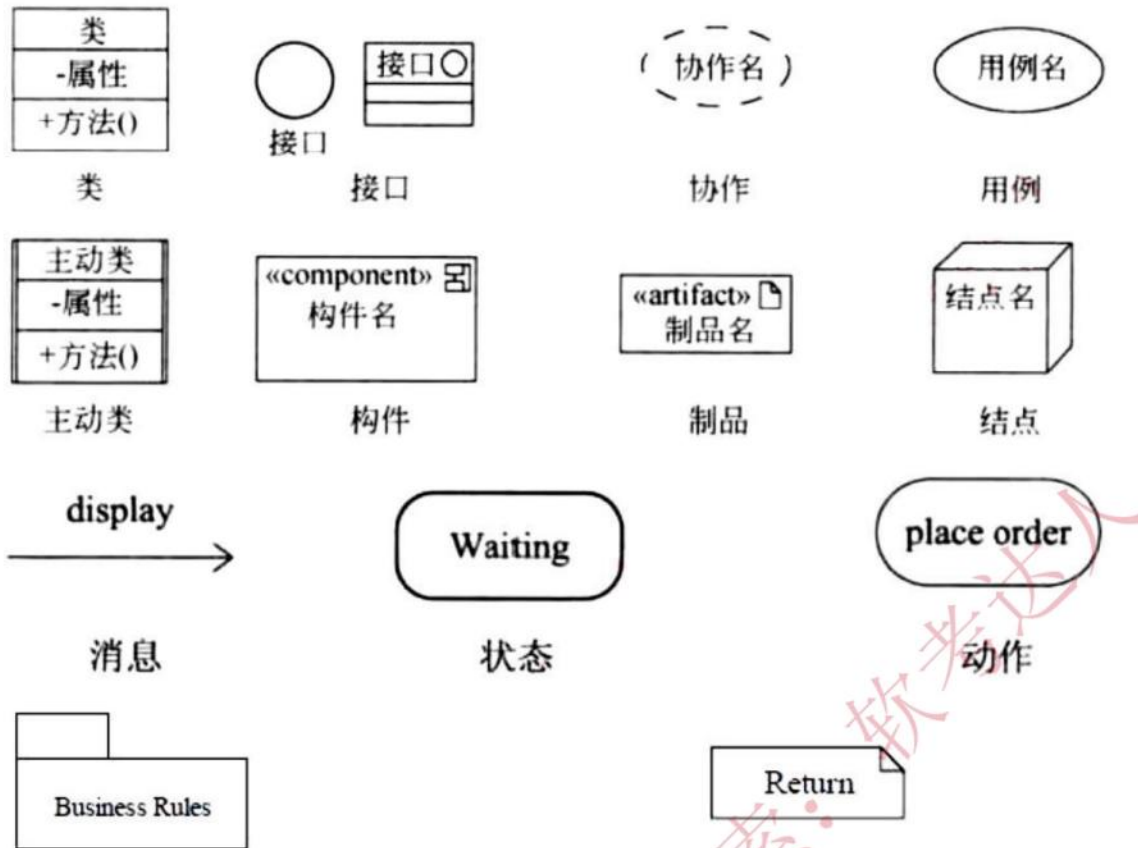


图 7-4 包

图 7-5 注解

2.2 关系

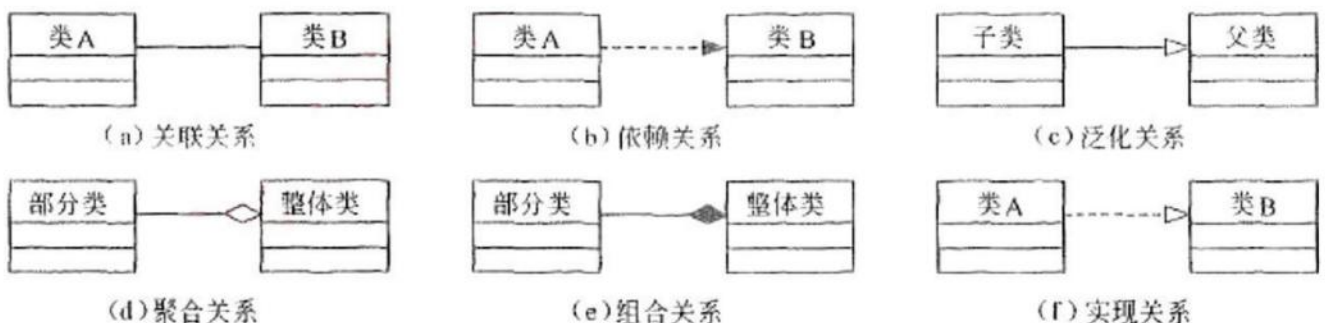
依赖：一个事物的语义依赖于另一个事物的语义的变化而变化

关联：是一种结构关系，描述了一组链，链是对象之间的连接。分为组合和聚合，都是部分和整体的关系，其中组合事物之间关系更强。两个类之间的关联，实际上是两个类所扮演角色的关联，因此，两个类之间可以有多个由不同角色标识的关联。

泛化：一般/特殊的关系，子类和父类之间的关系

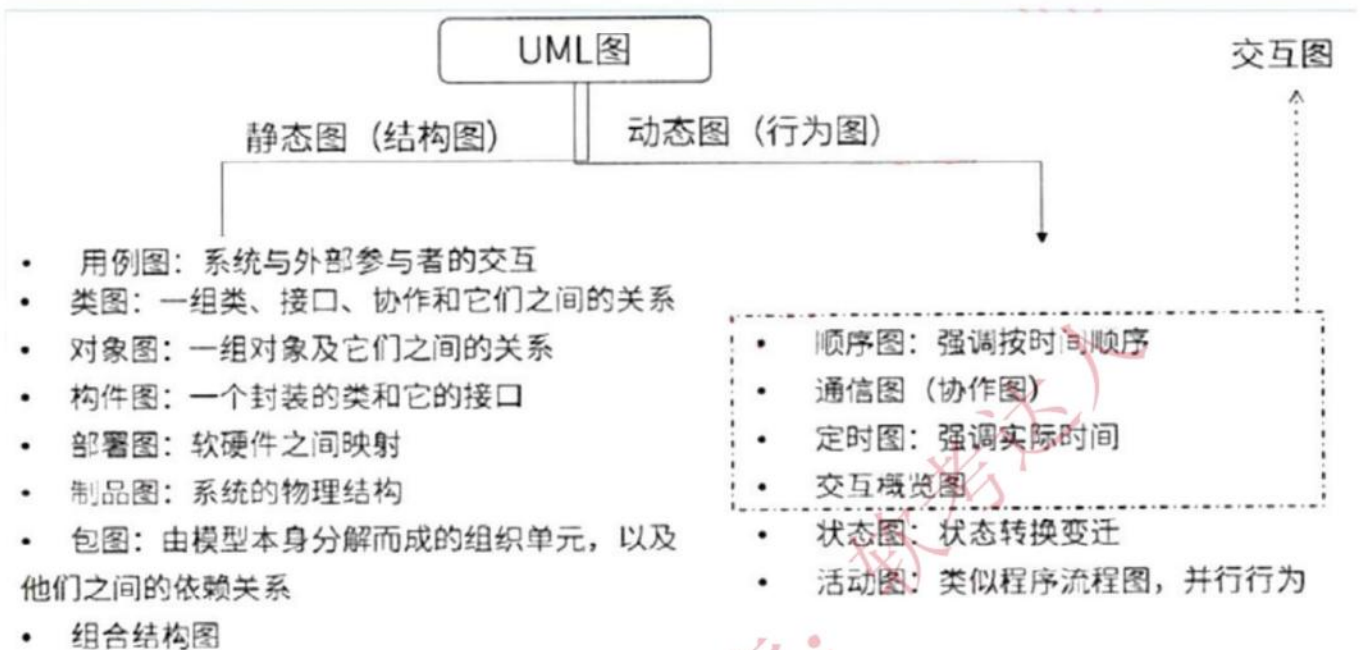
实现：一个类元指定了另一个类元保证执行的契约。

关系 UML 图形代号如下：

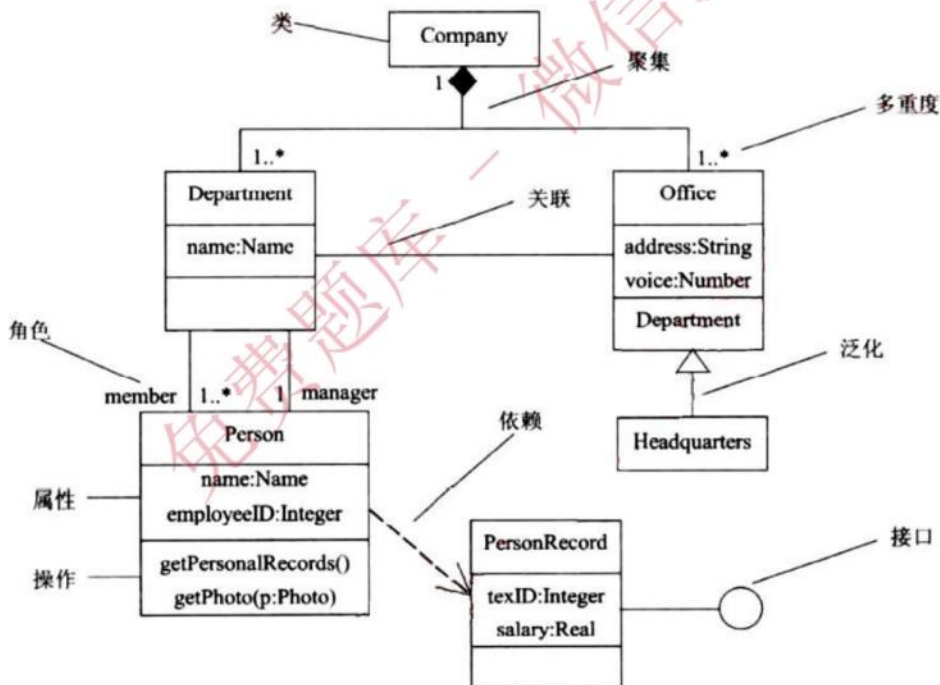


2.3 图

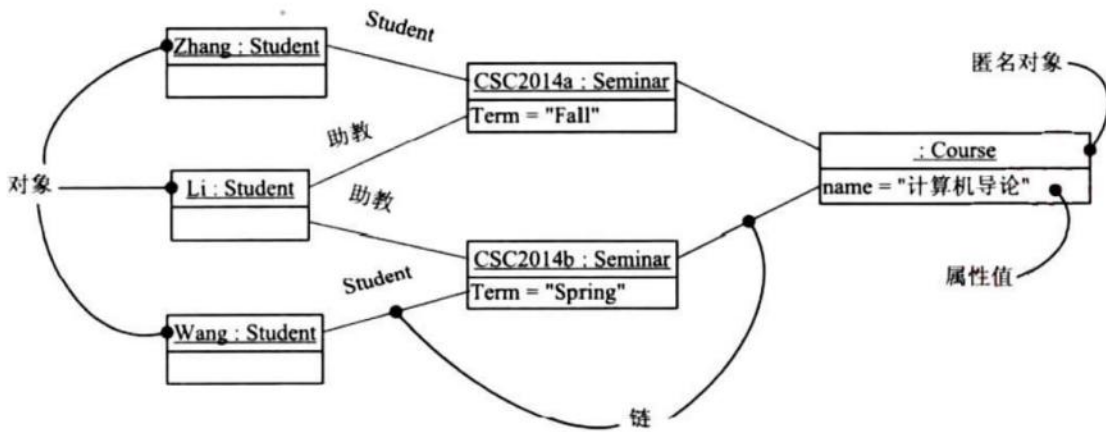
UML2.0 图，书上是 13 种，实际还包含制品图，一共 14 种，了解即可，总分类如下：



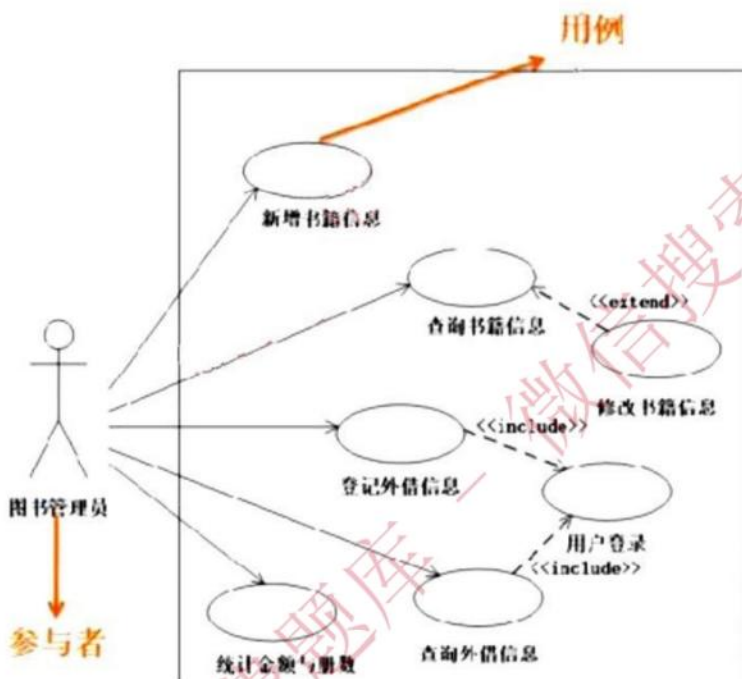
类图：静态图，为系统的静态设计视图，展现一组对象、接口、协作和它们之间的关系。UML 类图如下：



对象图：静态图，展现某一时刻一组对象及它们之间的关系，为类图的某一快照。在没有类图的前提下，对象图就是静态设计视图。如下：

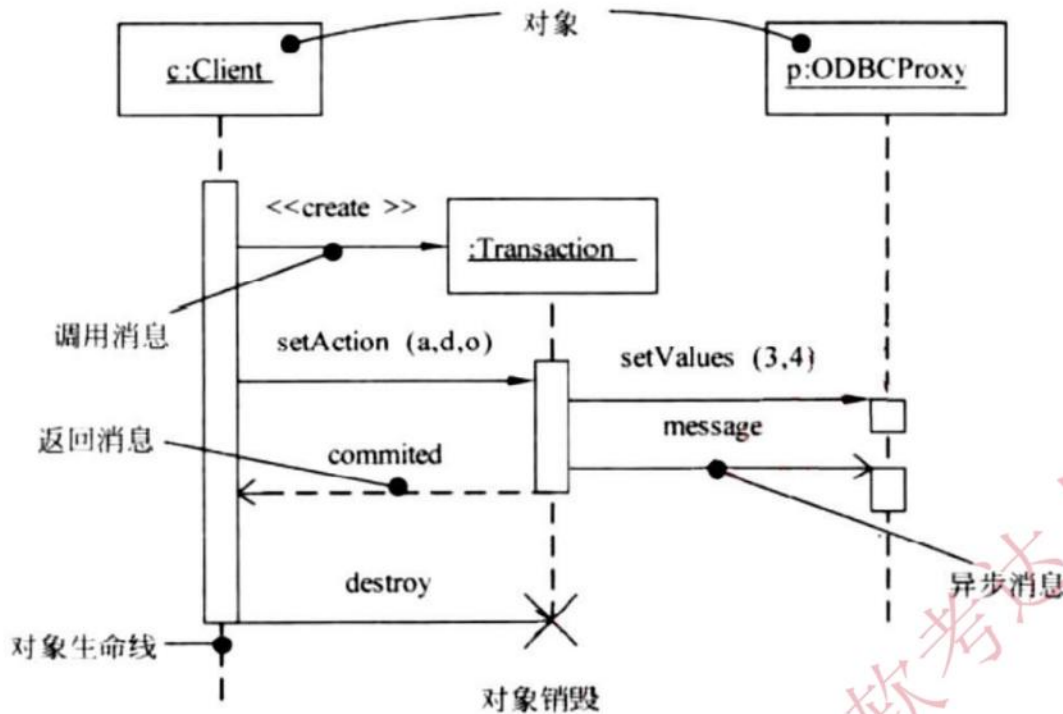


用例图：静态图，展现了一组用例、参与者以及它们之间的关系。用例图中的参与者是人、硬件或其他系统可以扮演的角色；用例是参与者完成的一系列操作，用例之间的关系有扩展、包含、泛化。如下：

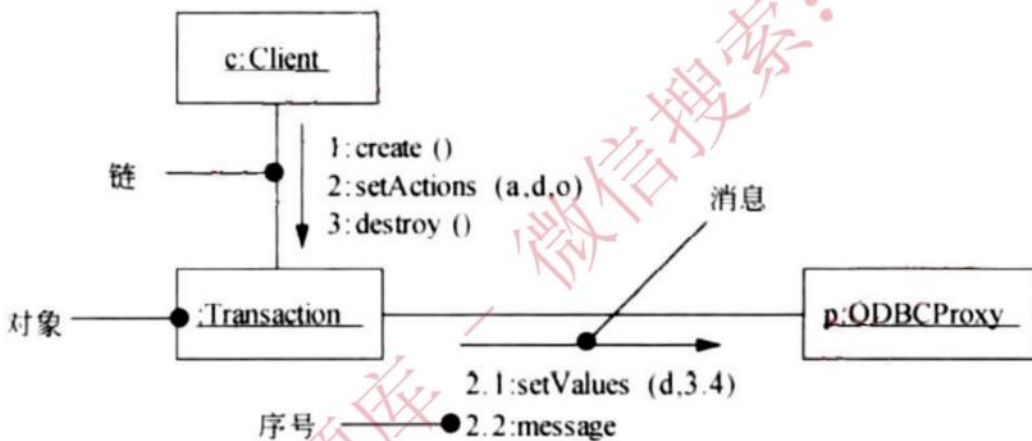


序列图：即顺序图，动态图，是场景的图形化表示，描述了以时间顺序组织的对象之间的交互活动。有**同步消息**（进行阻塞调用，调用者中止执行，等待控制权返回，需要等待返回消息，用实心三角箭头表示），**异步消息**（发出消息后继续执行，不引起调用者阻塞，也不等待返回消息，由空心箭头表示）、**返回消息**（由从右到左的虚线箭头表示）三种。如下：

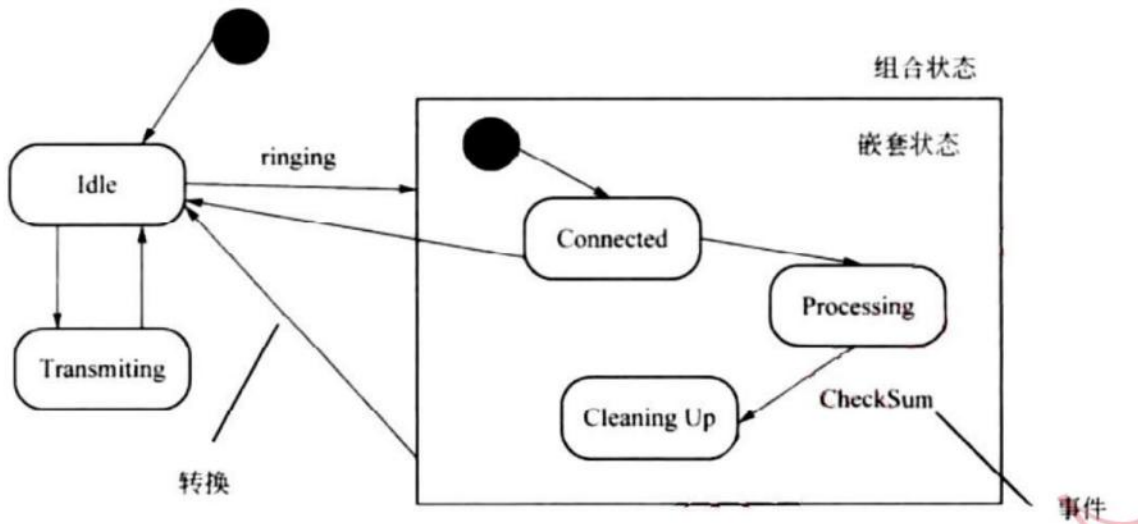
上方的对象对应下方箭头上的的成员和方法。



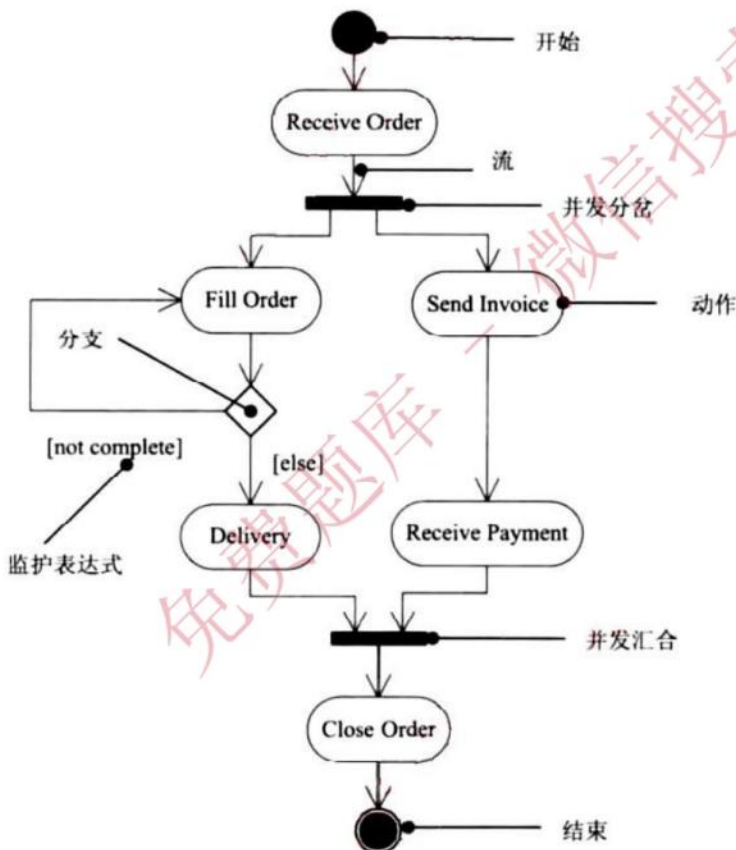
通信图：动态图，即协作图，强调参加交互的对象的组织。如下：



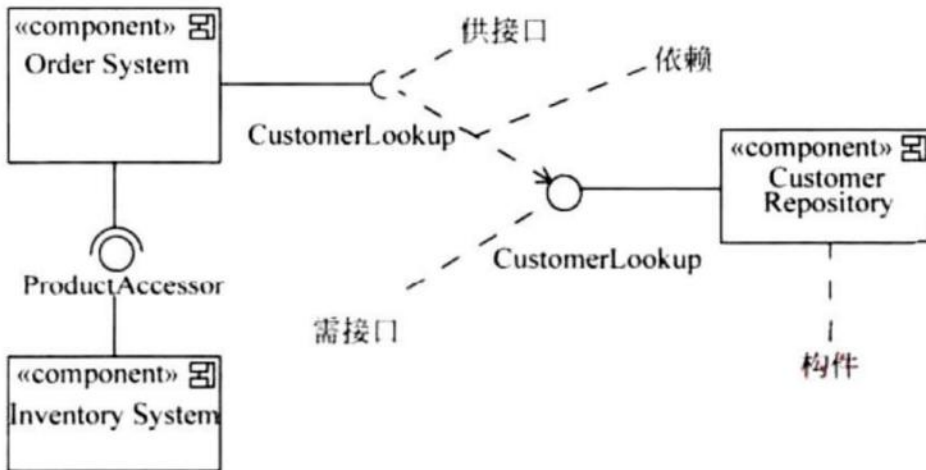
状态图：动态图，展现了一个状态机，描述单个对象在多个用例中的行为，包括简单状态和组合状态。转换可以通过事件触发器触发，事件触发后相应的监护条件会进行检查。状态图中转换和状态是两个独立的概念，如下：图中方框代表状态，箭头上的代表触发事件，实心圆点为起点和终点。



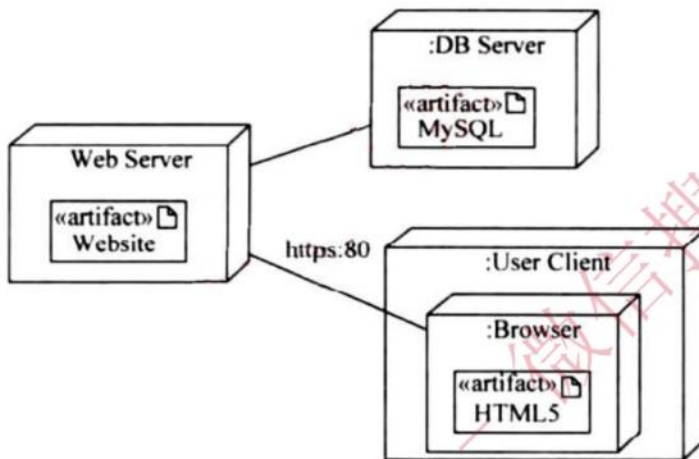
活动图：动态图，是一种特殊的状态图，展现了在系统内从一个活动到另一个活动的流程。活动的分岔和汇合线是一条水平粗线。牢记下图中并发分岔、并发汇合、监护表达式、分支、流等名词及含义。每个分岔的分支数代表了可同时运行的线程数。活动图中能够并行执行的是在一个分岔粗线下的分支上的活动，如下：



构件图（组件图）：静态图，为系统静态实现视图，展现了一组构件之间的组织和依赖。如下：

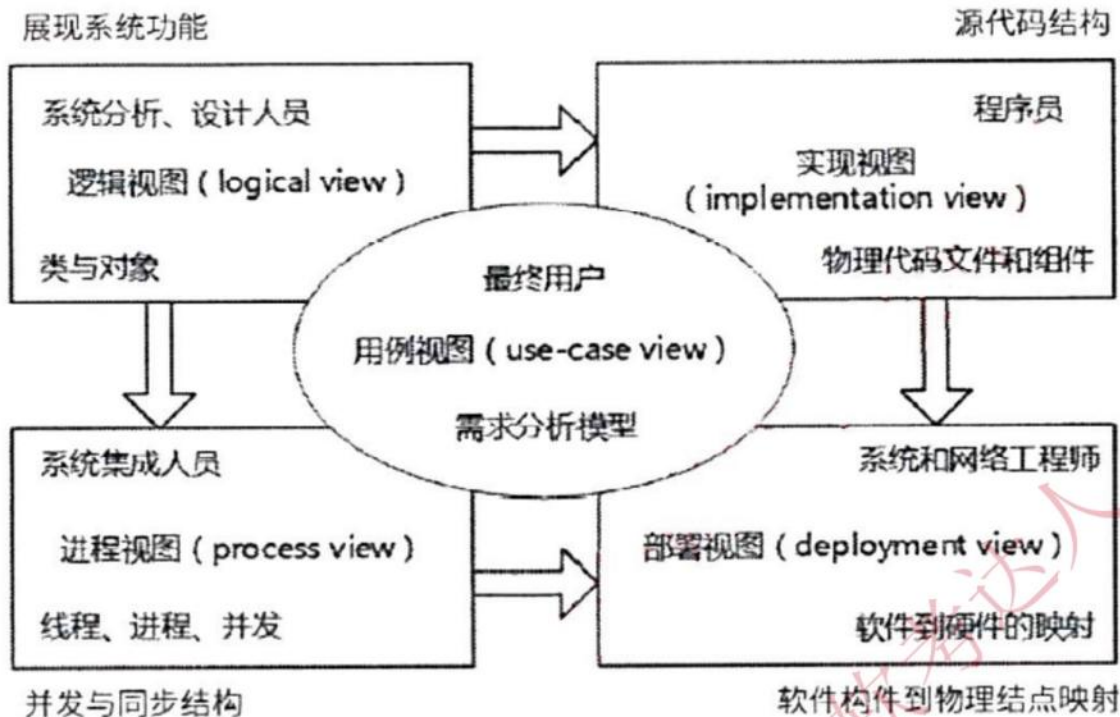


部署图：静态图，为系统静态部署视图，部署图物理模块的节点分布。它与构件图相关，通常一个结点包含一个或多个构件。其依赖关系类似于包依赖，因此部署组件之间的依赖是单向的类似于包含关系。如下：



UML 4+1 视图

如下图所示：



(1) 逻辑视图。逻辑视图也称为设计视图，它表示了设计模型中在架构方面具有重要意义的一部分，即类、子系统、包和用例实现的子集。

(2) 进程视图。进程视图是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，描述了并发与同步结构。

(3) 实现视图。实现视图对组成基于系统的物理代码的文件和构件进行建模。

(4) 部署视图。部署视图把构件部署到一组物理节点上，表示软件到硬件的映射和分布结构。

(5) 用例视图。用例视图是最基本的需求分析模型。

3 设计模式

层次结构

架构模式：软件设计中的高层决策，例如 C/S 结构就属于架构模式，架构模式反映了开发软件系统过程中所作的基本设计决策。

设计模式：每一个设计模式描述了一个在我们周围不断重复发生的问题，以及该问题的解决方案的核心。这样，你就能一次又一次地使用该方案而不必做重复劳动。设计模式的核心在于提供了相关问题的解决方案，使得人们可以更加简单方便的复用成功的的设计 and 体系结构。四个基本要素：模式名称、问题（应该在何时使用模式）、解决方案（设计的内容）、效果（模式应用的效果）。

惯用法：是最低层的模式，关注软件系统的设计与实现，实现时通过某种特定的程序设计语言来描述构件与构件之间的关系。每种编程语言都有它自己特定的模式，即语言的惯用法。例如引用一计数就是 C++ 语言中的一种惯用法。

具体设计模式分类

分为三类，创建型模式主要是处理创建对象，结构型模式主要是处理类和对象的组合，行为型模式主要是描述类或者对象的交互行为，总览图如下：



具体如下（红色粗体记忆关键字表示常考必须记住的）：

创建型设计模式	定义	记忆关键字
Abstract Factory 抽象工厂模式	提供一个接口，可以创建一系列相关或相互依赖的对象，而无需指定它们具体的类	抽象接口
Builder 构建器模式	将一个复杂类的表示与其构造相分离，使得相同的构建过程能够得出不同的表示	类和构造分离
Factory Method 工厂方法模式	定义一个创建对象的接口，但由子类决定需要实例化哪一个类。使得子类实例化过程推迟	子类决定实例化
Prototype 原型模式	用原型实例指定创建对象的类型，并且通过拷贝这个原型来创建新的对象	原型实例，拷贝
Singleton 单例模式	保证一个类只有一个实例，并提供一个访问它的全局访问点	唯一实例

结构型设计模式	定义	记忆关键字
---------	----	-------

Adapter 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，兼容接口
Bridge 桥接模式	将类的抽象部分和它的实现部分分离开来，使它们可以独立的变化	抽象和实现分离
Composite 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	整体-部分，树形结构
Decorator 装饰模式	动态的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 外观模式	定义一个高层接口，为子系统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 享元模式	提供支持大量细粒度对象共享的有效方法	细粒度，共享
Proxy 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	代理控制

行为型设计模式	定义	记忆关键字
Chain of Responsibility 职责链模式	通过给多个对象处理请求的机会，减少请求的发送者与接收者之间的耦合。将接收对象链接起来，在链中传递请求，直到有一个对象处理这个请求	传递请求、职责、链接
Command 命令模式	将一个请求封装为一个对象，从而可用不同的请求对客户进行参数化，将请求排队或记录请求日志，支持可撤销的操作	日志记录、可撤销
Interpreter 解释器模式	给定一种语言，定义它的文法表示，并定义一个解释器，该解释器用来根据文法表示来解释语言中的句子	解释器，虚拟机
Iterator 迭代器模式	提供一种方法来顺序访问一个聚合对象中的各个元素而不需要暴露该对象的内部表示	顺序访问，不暴露内部
Mediator 中介者模式	用一个中介对象来封装一系列的对象交互。它使各对象不需要显式地相互调用，从而达到低耦合，还可以独立的改变对象间的交互	不直接引用
Memento	在不破坏封装性的前提下，捕获一个对象的内部状态，并在	保存，恢复

备忘录模式	该对象之外保存这个状态，从而可以在以后将该对象恢复到原先保存的状态	
Observer 观察者模式	定义对象间的一种一对多的依赖关系，当一个对象的状态发生改变时，所有依赖于它的对象都得到通知并自动更新	通知、自动更新
State 状态模式	允许一个对象在其内部状态改变时改变它的行为	状态变成类
Strategy 策略模式	定义一系列算法，把它们一个个封装起来，并且使它们之间可互相替换，从而让算法可以独立于使用它的用户而变化	算法替换
Template Method 模板方法模式	定义一个操作中的算法骨架，而将一些步骤延迟到子类中，使得子类可以不改变一个算法的结构即可重新定义算法的某些特定步骤	
Visitor 访问者模式	表示一个作用于某对象结构中的各元素的操作，使得在不改变各元素的类的前提下定义作用于这些元素的新操作。	数据和操作分离

十四 专业英语

专业英语没有视频讲解，大家看看下面的英文单词，自己做做历年真题即可。

词汇	含义
access	存取
active-matrix	主动矩阵
adapter	适配器，转换器
adapter cards	适配卡
agents	代理
analog signals	模拟信号
animations	动画
applets	程序
arithmetic operations	算术运算
array	数组，阵列
assembly	汇编，安装，装配

asynchronous	异步的，非同步的
asynchronous communications port	异步通信端口
attachment	附件
audio-output device	音频输出设备
Bandwidth	带宽
Bar code reader	条形码读卡器
Bit	比特
Bluetooth	蓝牙
Bus line	总线
cache	高速缓存
CAD (Computer Aided Design)	计算机辅助设计
CD-R	可记录压缩光盘
CD-ROM	可记录光盘
CD-RW	可重写光盘
certificates	证书
command	命令，指令
compress	压缩，精减
configuration	配置
control unit	操纵单元
controller	控制器
cookies	信息记录程序
cookies-cutter programs	信息记录截取程序
coprocessor	协同处理器
copyright	版权
correspond	通信(联系)
critical	临界的；临界值
cursor	光标
database	数据库
decimal	十进制；十进制的
digital	数字的

digital subscriber line	数字用户线路
digital versatile disc	数字化通用磁盘
digital video disc	数字化视频光盘
directory	目录, 索引簿
disk	盘, 磁盘
display	显示
dot-matrix printer	点矩阵式打印机
drive	驱动
e-commerce	电子商务
E-mail	电子邮件
enclose	封闭, 密封, 围住, 包装
file	文件
firewall	防火墙
Flash RAM	闪存
format	格式
hacker	黑客
hexadecimal	十六进制的
hierarchical	分级的, 分层的
home page	主页
host computer	主机
index	索引, 变址, 指数
integrate	综合, 集成
integrated Software	集成软件
interpreter	释程序, 翻译机
key	键, 关键字, 关键码
line	(数据, 程序)行, 线路
list	列表, 显示
locating	定位, 查找
macro	宏, 宏功能, 宏指令
main board	主板

map	图; 映射, 变址
margin	余量, 边缘, 边际
micro	微的, 百万分之一
memory	内存
network adapter card	网卡
network terminal	网络终端
numerical	数量的, 数字的
on-line	联机的
operate	操作, 运算
optimize	优选, 优化
output	输出, 输出设备
pixel	像素
pop	上托, 弹出 (栈)
printer	打印机, 印刷机
product	(乘)积, 产品
Programming language	程序设计语言
property	性(质), 特征
protocol	规约, 协议, 规程
pseudo	假的, 伪的, 冒充的
push	推, 按, 压, 进 (栈)
recall	撤消, 复活, 检索
replaceable	可替换的
retrieve	检索
retrieve	检索
rewrite	重写
scan	扫描, 扫视, 搜索
scanner	扫描器
series	序列, 系列, 串联
set	设置
space	空格键, 空间

stack	栈，堆栈，存储栈
sub-directory	子目录
subset	子集，子设备
system software	系统软件
Telnet	远程登录
template	模板
terminal	终端
terminal	终端，端子
tracks	磁道
update	更新，修改，校正
variable	可变的
video	视频，电视
video display screen	视频显示屏
virtual memory	虚拟内存
Virus	病毒
voice recognition system	声音识别系统
ADSL (Asymmetric Digital Subscriber Line)	不对称数字订阅线路
AGP (accelerated graphics port)	加速图形接口
AH (Authentication Header)	鉴定文件头
API (Application Programming Interface)	应用程序设计接口
ARP (Address Resolution Protocol)	地址解析协议
ATM (Asynchronous Transfer Mode)	异步传输模式
BCF (Boot Catalog File)	启动目录文件
BIOS (Basic input/output System)	基本输入/输出系统
CA (Certification Authority)	证书管理机构
CD (compact disc)	压缩盘
CGI (Common Gateway Interface)	公共网关界面
CGI (Common Gateway Interface)	通用网关接口
CISC(Complex Instruction Set Computer)	复杂指令集计算机

CMOS (complementary metal-oxide semiconductor)	nductor) 互补金属氧化物半导体
CPU (central processing unit)	中央处理器
CRC (Cyclical Redundancy Check)	循环冗余检查
CRT (Cathode Ray Tube)	阴极射线管
CTI (Computer Telephone Integration)	计算机电话综合技术
DBMS (database management system)	数据库管理系统
DBS (Direct Broadcast Satellite)	直接卫星广播
DES (Data Encryption Standard)	数据加密标准
DIC (Digital Image Control)	数字图像控制
DNS (Domain Name System)	域名系统
DOM (Document Object Model)	文档对象模型
DSP (Digital Signal Processing)	数字信号处理
DTE (Data Terminal Equipment)	数据终端设备
DWDM (Dense Wavelength Division Multiplex)	波长密集型复用技术
EDO (Extended Data Output)	扩充数据输出
FRC (Frame Rate Control)	帧比率控制
FTP (File Transfer Protocol)	文件传输协议
Ghost (General Hardware Oriented System Transfer)	全面硬件导向系统转移
HDTV (high-definition television)	高清晰度电视
HTML (Hyper Text Markup Language)	超文本标示语言
HTTP (Hyper Text Transmission Protocol)	超文本传输协议
Hypertext	超文本
ICMP (Internet Control Message Protocol)	因特网信息控制协议
IP (Internet Protocol)	网际协议
LAN (Local Area Network)	局域网
LCD (iquid crystal display monitor)	液晶显示器

MAC (Media Access Control)	媒体访问控制子层协议
MIDI (Musical Instrument Digital Interface) 乐	乐器数字化接口
MMDS (Multichannel Multipoint Distribution Service)	多波段多点分发服务
NAC (Network Access Control)	网络存取控制
NOS (Network operation system)	网络操作系统
OJI (Open Java VM Interface)	开放 Java 虚拟机接口
OLE (object linking and embedding)	对象链接入
OMR (optical-mark recognition)	光标阅读器
P3P (Privacy Preference Project)	个人私隐安全平台
PCI(peripheral component interconnect)	外部设备互连总线
PCM (Pulse Code Modulation)	脉冲编码调制
PDA (personal digital assistant)	个人数字助理
POP3 (Post Office Protocol Version 3)	第三版电子邮局协议
PSTN (Public Switched Telephone Network)	公用交换式电话
RAM (Random Access Memory)	随机存取存储器
Remote Login	远程登录(注册)
RISC(Reduced Instruction Set Computer)	精简指令集计算机
ROM (read-only memory)	只读存储器
SDRAM (Synchronous Dynamic Random Access Memory)	同步动态随机存储器
SGML (Standard Generalized Markup Language)	标准通用标记语言
SMIL (Synchronous Multimedia Integrate Language)	同步多媒体集成语言
SQL (structured query language)	结构化查询语言
TAPI (Telephony Application Programming Interface)	电话应用程序接口

TCP (Transmission Control Protocol)	传输控制协议
TSAPI (Telephony Services Application Programmin)	电话服务应用
UDP (User Datagram Protocol)	用户数据报协议
URL (Uniform Resource Locator)	统一资源定位符
USB (universal serial bus)	通用串行总线
VDT (Video Display Terminals)	视频显示终端
VOD (Video On Demand)	视频点播
VPN (virtual private network)	虚拟专用网络
VPN (Virtual Private Network)	虚拟局域网
VRML (Virtual Reality Makeup Language)	虚拟现实结构化语言
VRR (Vertical Refresh Rate)	垂直扫描频率
WAIS (Wide Area Information Service)	广域信息服务器
WDM (Wavelength Division Multiplex)	波分多路复用
Web browser	网络浏览器
WWW (World Wide Web)	万维网
XML (Extensible Markup Language)	可扩展置标记语言

软件设计考点及分数说明

题号	试题类型	学科知识点	考察内容
试题 1	必答题	数据流图 DFD	补充数据流图外部实体; 补充数据流图数据存储; 补充数据流 (名称、起点、终点); 数据流图的改错 (较少考察, 包括数据流错误、删除多余数据流); 数据流图相关概念简答。
试题 2	必答题	数据库设计	补充 E-R 图; E-R 图转换为关系模式;

			主键和外键、新增联系判断。
试题 3	必答题	UML 建模	用例图（联系类型，参与者）； 类图 and 对象图（多重度，联系类型）； 顺序图（补充对象名和消息名）； 活动图（补充活动名，分岔线用途）； 状态图（补充状态，状态转换条件）； 通信图（补充对象名，消息名）
试题 4	必答题	C 算法设计	各种经典算法设计和数据结构，如链表、栈、二叉树操作算法、KMP 算法等； 算法类型 （动态规划法、分治法、回溯法、递归法、贪心法）； 时间、空间复杂度 ； 给定输入求输出。
试题 5	选答题	C++ 语言程序设计	不推荐选做： C++ 语法（只考简单语法，不考算法）+ 设计模式。
试题 6	选答题	Java 语言程序设计	推荐选做： Java 语法（只考简单语法，不考算法）+ 设计模式。

本章讲解的是软件设计师考试下午考试科目的知识点，下午的试卷总体来说比上午试卷更难，难在软件设计和长篇幅的题目理解。但是有很多技巧在内，很多题目的答案就藏在题目之中，要掌握这些技巧，即使在一知半解的情况下，也可以快速的答对大部分题目。

在本章的视频课程里，文老师将会**着重介绍审题和解题的技巧**，也请学员们跟随着文老师的视频课程进行学习，初学者可以多听几遍视频课程，在掌握技巧之后，通过做历年真题进行实战演练，文老师也特地将**17-19 年下午真题的详细解答都录制成了视频**供学习，如果在做真题过程中遇到不懂的，可以多听真题详解，加深映像。

下午的题目唯一的难点在于第 4 题算法设计分析，前三题通过技巧训练一般可以拿到 12 分，最后的选做题可以拿到 15 分满分，要保证自己朝着这个方向努力，即使算法设计过难也可以顺利通过考试。其他题目都不难，要跟着文老师视频课程多学多练，掌握解题技巧。

十五 数据流图 DFD

1 数据流图基本概念

数据流图描述数据在系统中如何被传送或变换，以及如何对数据流进行变换的功能或子功能，用于对功能建模，数据流图相关概念如下：

元 素	说 明	图 元
数据流	由一组固定成分的数据组成，表示数据的流向。每个数据流通常有一个合适的名词，反映数据流的含义	
加工	加工描述了输入数据流到输出数据流之间的变换，也就是输入数据流做了什么处理后变成了输出数据流	
数据存储 (文件)	用来表示暂时存储的数据，每个文件都有名字。流向文件的数据流表示写文件，流出的表示读文件	
外部实体	指存在于软件系统外的人员或组织	

数据流图是可以分层的，从顶层（即上下文无关数据流）到 0 层、1 层等，顶层数据流图只含有一个加工处理表示整个管理信息系统，描述了系统的输入和输出，以及和外部实体的数据交互。

数据项：数据流图描述了系统的分解，但它并没有给出图中各成分的说明，数据项就是数据流图的组成部分，不在数据流图中显示，但在题目说明中会描述出某条数据流的组成元素。

加工特点

对每一个基本加工，必须有一个加工规格说明；加工规格说明必须描述把输入数据流变换为输出数据流的加工规则；决策表可以用来表示加工规格说明。

四大模块

传入模块：从下属模块取得数据，经处理再将其传送给上级模块。

传出模块：从上级模块取得数据，经处理再将其传送给下属模块。

变换模块：从上级模块取得数据，进行特定的处理，转换成其他形式，再传送给上级模块。

协调模块：对所有下属模块进行协调和管理的模块。

2 数据字典

符 号	含 义	举 例 说 明
=	被定义为	
+	与	$x=a+b$ ，表示x由a和b组成
[... , ...]或 [... ...]	或	$x=[a, b]$ ， $x=[a b]$ ，表示x由a或由b组成
{...}	重复	$x=\{a\}$ ，表示x由0个或多个a组成
(...)	可选	$x=(a)$ ，表示a可在x中出现，也可以不出现

数据字典是用来定义在数据流图中出现的符号或者名称的含义，在数据流图中，每个存储、加工、实体的含义都必须定义在数据流图中，并且父图和子图之间这些名称要相同。

3 数据流图的设计原则

3.1 基本设计原则

- (1)数据守恒原则：对任何一个加工来说，其所有输出数据流中的数据必须能从该加工的输入数据流中直接获得，或者说是通过该加工能产生的数据。
- (2)守恒加工原则：对同一个加工来说，输入与输出的名字必须不相同，即使它们的组成成分相同。
- (3)对于每个加工，必须既有输入数据流，又有输出数据流。
- (4)外部实体与外部实体之间不存在数据流
- (5)外部实体与数据存储之间不存在数据流
- (6)数据存储与数据存储之间不存在数据流
- (7)父图与子图的平衡原则：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必须一致，此即父图与子图的平衡。父图与子图之间的平衡原则不存在于单张图。
- (8)数据流与加工有关，且必须经过加工。

3.2 父图与子图间的平衡

顶层图中，描述了外部实体和系统之间的数据流关系，在 0 层图中，展开系统，变成了外部实体和系统内部更详细的数据流关系，但是，顶层和 0 层互为父图、子图关系，其输入、输出数据流个数和名称不应该发生变化，也即应该保持平衡，在解题中，可以**关注父图、子图中的外部实体的输入、输出数据流个数、名称是否相同**，就可以确认是否缺少数据流。

3.3 子图内平衡

对于子图内的**每一个加工**，**要求既要有输入，也要有输出**，才是数据平衡，根据这一原则，可以对每个输入进行判断，是否有相应的输出，反之亦然，就可以知道是否缺少某条数据流，从而进行对应的补充。

并且**数据流只能和加工有关**，即从加工流向加工、数据源流向加工或者加工流向数据源。

输入数据流和输出数据流名称不能相同，类型必须匹配。

4 解题技巧

数据流图的考试形式非常固定，第一小题补充外部实体，第二小题补充数据存储，第三小题补充缺失数据流，第四小题考察简单概念。都不算难，以题目描述和数据流图为主，答案都在题目描述里，更像是阅读理解题，技巧如下：

1、补充外部实体：外部实体就是与系统进行交互的其他实体，可以是大型系统、公司部门、相关人员等，外部实体会与系统进行交互，反应在数据流图中就是一个个事件流，依据事件的名称结合题目描述可以轻易得出答案。

2、补充数据存储：数据存储出现在 0 层数据流图中，反应系统内部数据的存储，可以**直接根据数据流图中数据存储的输入数据流和输出数据流判断该数据存储的信息**得出答案，但注意要使用题目说明的数据存储名词作为答案。

3、补充缺失数据流：**详细阅读题目描述**，依据题目描述对涉及的数据流图进行一一核对，这是最为简单直接的方法，因为即使开始就去考虑数据守恒、父图子图平衡等原则，最终还是要根据题目描述核对，不如一开始就直接核对。

4、简单概念：题型不固定，一般只有 2-3 分，都是比较简单的判断。

重点在于**审题、审图**，根据**两个数据平衡原则**。万变不离其宗。

常见问题：

Q: 补充数据存储时, 题目中没有具体存储名称怎么办?

A: 此时可依据数据流图中数据存储的输入输出数据流自行起名。

Q: 补充数据流时, 该写多少条?

A: 一般按分写, 几分就写几条。

Q: 觉得要补充的数据流很多, 比分数更多怎么办?

A: 可以多写, 这个是按点给分的, 多写不会扣分, 但要注意不能太过分。

Q: 总觉得数据流把握不准, 跟答案有出入怎么办?

A: 首先要注意, 软考官方不公布标准答案, 所有答案都是老师校对的, 可能会不全面; 其次, 部分真题本身也不太严谨, 有二义性。因此不用太纠结, 掌握解题方法, 多刷题即可。

Q: 这种题目该怎么学习, 学习到哪种程度呢?

A: 要求能拿到 12 分以上; 看完视频对应专题课程后, 立即去做课后习题 (选自 17-19 年真题), 做完后立即去看对应真题视频讲解, 掌握技巧。

详细示例及应用见文老师课程视频, 有详细技巧的应用讲解。

十六 数据库设计

1 E-R 图转换为关系模式

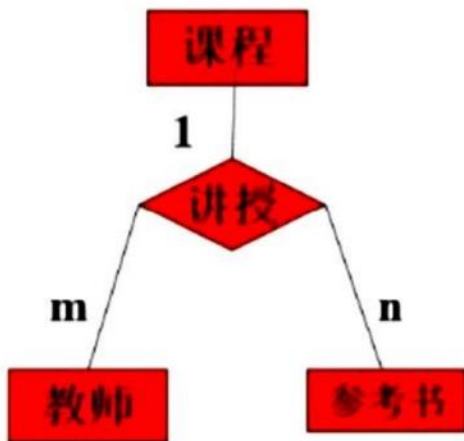
E-R 图中, 有实体和联系两个概念, 实体和实体之间的联系分为三种, 即 1:1, 1:N, M:N, 这三种情况, 转换为关系模式的方法也不同。

首先, 每个实体都要转换为一个关系模式, 对于联系, 一对一, 联系作为一个属性随便加入哪个实体中; 一对多, 联系可以单独转换为一个关系模式, 也可以作为一个属性加入到 N 端中 (N 端实体包含 1 端的主键); 多对多, 联系必须单独转换为一个关系模式 (且此关系模式应该包含两端实体的主键)。

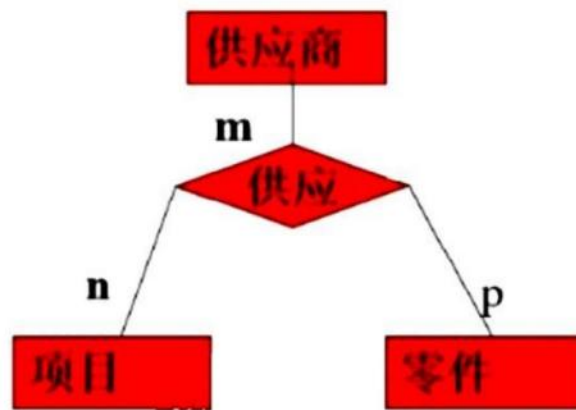
转换之后要注意: **原来的两个实体之间的联系必须还存在**, 能够通过查询方式查到对方。

在实际解题时, 要注意, 某个实体的属性, 还应该包括其联系属性, 具体问题具体分析。

两个以上实体型的联系:



两个以上实体型间1:n联系



两个以上实体型间m:n联系

2 解题技巧

数据库设计的考法也非常固定，第一小题补充 E-R 图，第二小题补充关系模式，第三小题是简单的情景问答题。同样也都不难，结合题目描述和 E-R 图的一些特点可以轻易得出答案，技巧如下：

1、补充 E-R 图：这是重中之重，E-R 图如果弄错了，后续题目都有影响，主要是根据题目描述确认哪些实体之间有联系，联系类型是哪一种，而后进行连线即可，并不难。

2、补充关系模式：实际考察的是将 E-R 图转换为关系模式，补充缺失的属性，分成两步：首先需要审题，题目会给出每个关系模式的属性信息，先将题目中的属性信息和问题对应，将缺少的属性全部补充；而后再按照规则转换，即前面所说的规则，按联系的三种对应方式决定要添加哪些字段。

另：依据最新真题分析，这种题目越来越规范了，如果题目中给出了具体实体的属性，只需要和题目中核对即可，如果题目中没给，而是某个联系转换为关系模式，则需要考虑转换规则。

3、情景问答：一般都是给出一段新的描述，要求新增一种实体-联系类型和关系模式，本质也是考察联系类型和 E-R 图转换为关系模式。

注意审题，常识以及 E-R 图转换为关系模式的原则（主要是联系的归属）。

另外，要注意以下三点：

1、实体和子实体（之间直接用直线连接，是从属关系，无联系类型）；

2、多个实体一个类型（一般是三个实体连接到一个类型上，本质也是两两对应的联系关系，根据题目说明，若有三个实体相关，则是此种情况）；

3、主键和外键（主键是本关系内唯一，外键是其他关系的主键，外键可以有多个）。

常见问题：

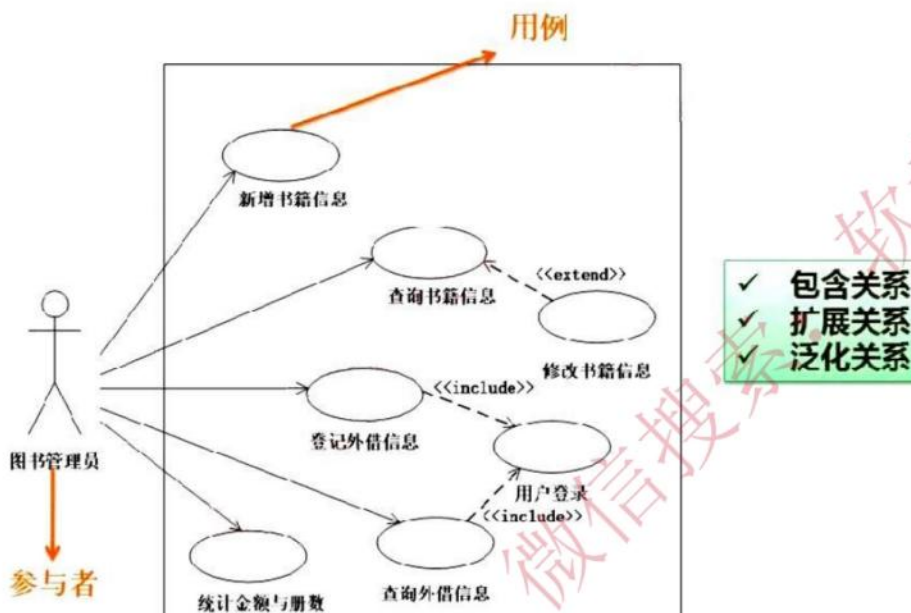
Q：这种题目该怎么学习，学习到哪种程度呢？

A: 要求能拿到 12 分以上; 看完视频对应专题课程后, 立即去做课后习题 (选自 17-19 年真题), 做完后立即去看对应真题视频讲解, 掌握技巧。

详细示例及应用见文老师课程视频, 有详细技巧的应用讲解。

十七 UML 建模

1 用例图



主要考察**参与者和用例的识别、用例之间的关系** (包含 include、扩展 extend、泛化), 如上图所示, 登记外借信息用例**包含**用户登录用例, 因为每次如果要登记外借信息, 必然要先进行用户登录。而查询书籍信息的**扩展**是修改书籍信息, 是因为每次查询书籍信息后, 发现有错误才会修改, 否则不修改, 不是必要的操作。

因此, 区分用例间的关系是包含还是扩展, **关键在于是不是必须操作**。

2 类图

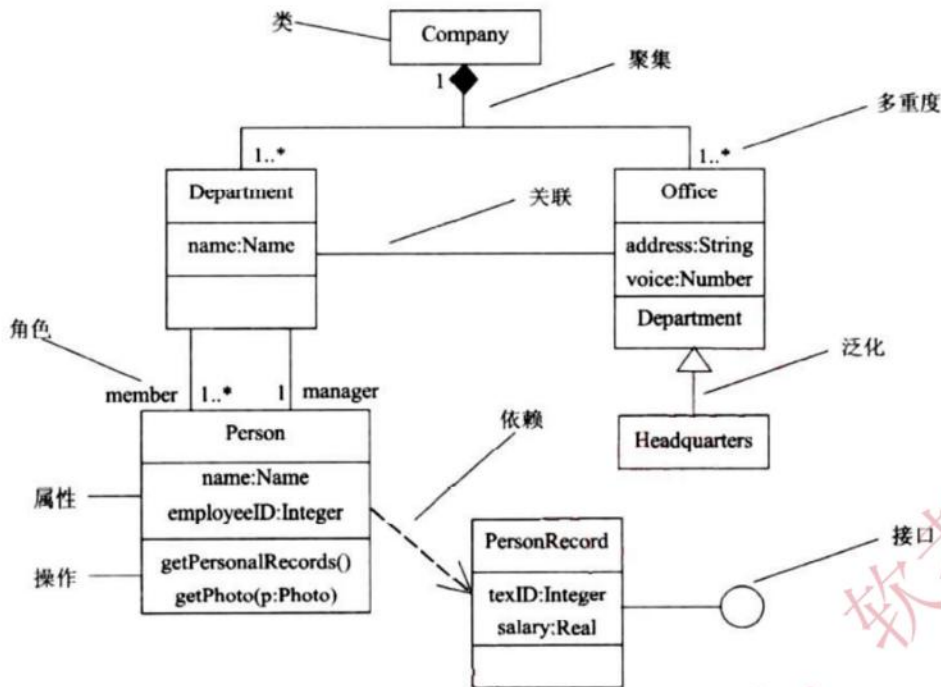


图 10-11 UML 类图

主要考察填**类名**、**多重度**、**类之间的联系**（泛化、组合、聚合、实现、依赖）

多重度（有点类似于 E-R 图中的联系类型）含义如下：

- 1: 表示一个集合中的一个对象对应另一个集合中 1 个对象。
- 0..*: 表示一个集合中的一个对象对应另一个集合中的 0 个或多个对象。
- 1..*: 表示一个集合中的一个对象对应另一个集合中的一个或多个对象。
- *: 表示一个集合中的一个对象对应另一个集合中的多个对象。

类之间的联系如下：

理解聚合和组合都是部分和整体之间的关系，类之间大多考察这两种关系，必须记住其符号，且菱形一端为整体，表示其由另外的一端实体组成，可依据题意找出组成元素解题。



图 10-6 依赖



图 10-9 泛化



图 10-7 关联



图 10-10 实现



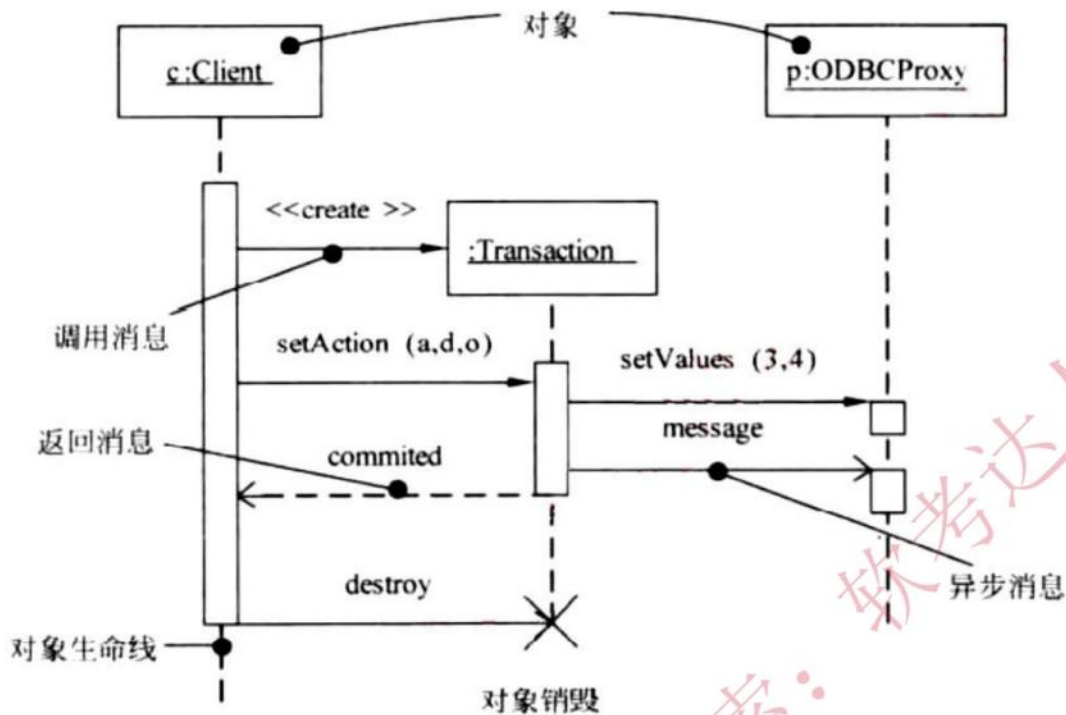
图 10-8 聚集



图 10-11 组合

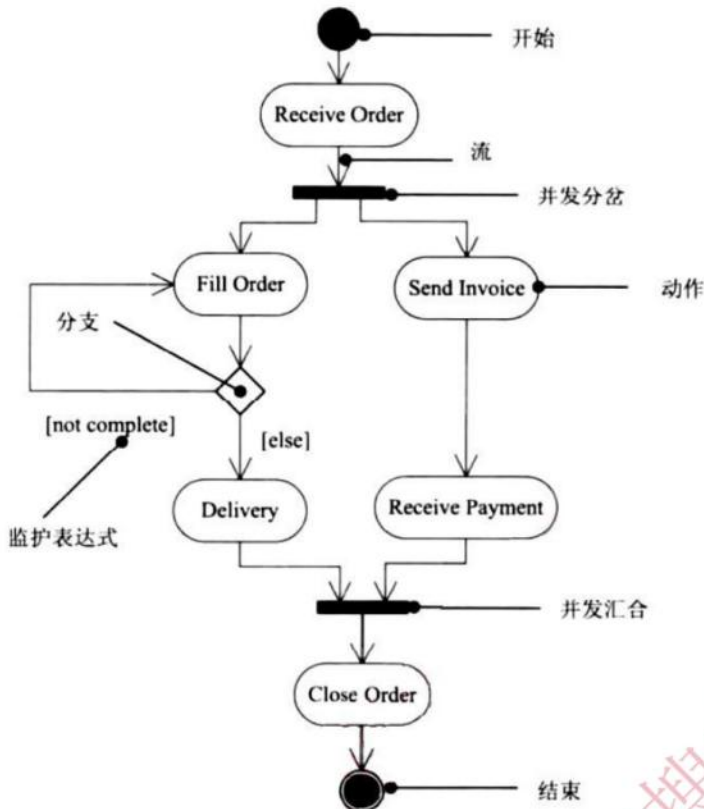
依赖关系最弱，只要有部分相关就是依赖。泛化关系是父子关系；组合和聚合都是部分-整体关系，组合更强。

3 序列图



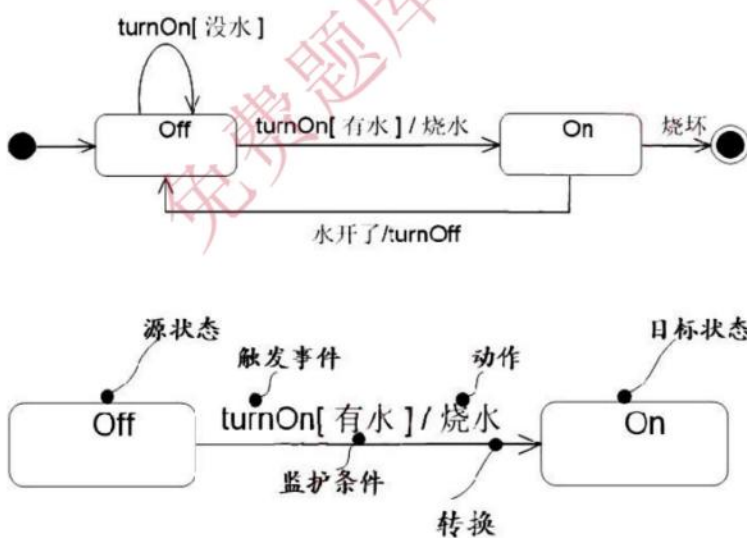
序列图也即顺序图，如上图所示为顺序图，主要考察填对象名、消息名，消息就是一个个箭头上传递的，对象作为实体在最上端，自上而下为时间顺序，反应一个事件的执行过程。

4 活动图



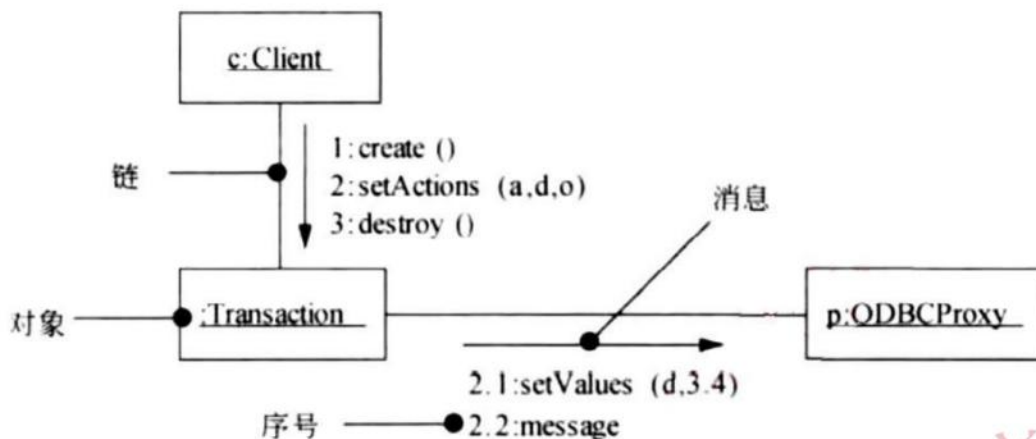
活动图类似于程序流程图，粗线表示该活动分成了多少个并行的任务，最后又会汇总到一起。主要考察填活动名称。

5 状态图



主要描述状态之间的转换，主要考察的就是填状态名、填状态转换的条件，具体如上图所示。

6 通信（协作）图



是顺序图的另一种表示方法，也是由对象和消息组成的图，只不过不强调时间顺序，只强调事件之间的通信，而且也没有固定的画法规则，和顺序图统称为交互图。主要考察填对象名、填消息。

7 解题技巧

由前面的介绍可知，考察 UML 建模就是考察多种图形，对这些图形的考察一般都是缺失一些关键点，而后要求考生补图。

要求认真审题，根据题干说明补齐类名或者对象名或者消息名等等，记住类图和对象图中的**多重度**（互相独立的分析，掌握表示方法）、**类之间的联系标识**（多边形端为整体，直线端为个体）。

认真审题，审图，根据说明查缺补漏，一般来说有以下几种题型：

1、补充用例图：主要考察补充用例名称、参与者、用例之间的关系，只要认真审题，根据题目描述核对，都可以轻易得出答案。

2、补充类图：主要考察补充类名称，需要根据类之间的关系以及多重度来判断，需要牢记类之间关系的图形符号，尤其是组合、聚合和继承的符号，并且观察符号上的多重度数字，与题目描述对应。

3、补充状态图：主要补充状态名称，根据题目描述可以轻易得出答案。

4、识别设计模式，掌握经典设计模式特点，并结合英文等联想。

常见问题：

Q：看到一张类图都是空的，就懵逼了，该怎么办？

A：类和类之间的联系，尤其是泛化联系是非常重要的，也是解题的关键，要记住泛化联系的符号，而后去题目描述里找到具有父子关系的实体，基本上题目就一步步解出来了。其他如组合、聚合联系的符号也是解题关键。

Q: 这种题目该怎么学习, 学习到哪种程度呢?

A: 要求能拿到 12 分以上; 看完视频对应专题课程后, 立即去做课后习题 (选自 17-19 年真题), 做完后立即去看对应真题视频讲解, 掌握技巧。

详细示例及应用见文老师课程视频, 有详细技巧的应用讲解。

十八 算法设计与 C 程序实现

时间复杂度、算法判断见本专题视频讲解。

1 解题技巧

算法设计是下午软件设计中最难的题型, 主要难在 C 代码填空上, 因此文老师的建议是先不解决代码填空题 (因为最难), 先解决其他外围问题 (如时间复杂度、算法技巧、取特殊值计算结果), 最后解决代码填空, 有助于理解整个题目, 技巧如下:

1、代码填空: 第一小题, 最后解决, 并不影响解决其他题目, 要理解题目算法原理, 才能得出答案。另外近几年的算法设计真题有很大的技巧, 即便不理解算法原理也可以推导出答案, 要结合算法描述中的公式, 以及算法代码中类似的分支, 能够发现填空的答案在代码中其中地方已经给出, 因为算法原理的相似可以通用, 尤其是分治法算法。详见文老师真题详解, 详述技巧。

要注意的是, 当遇到有最小值或最大值参与比较时, 若比较出来比最小值更小, 接下来肯定要更新这个最小值以及其下标元素值。当遇到一些条件判断的填空时, 要注意对应上下文查看哪些变量是作为控制的。

2、算法、时间复杂度: 第二小题, 先做, 考察何种算法很好分辨, 涉及到分组就是分治法, 局部最优就是贪心法, 整体规划最优就是动态规划法, 迷宫类的问题是回溯法, 记住关键字很好区分; 时间复杂度就是看 C 代码中的 for 循环层数和每一层的循环次数, 涉及到二分必然有 $O(\log n)$ 。

3、特殊值计算: 第三小题, 一般应该先做, 不需要根据 C 代码, 直接根据题目给出的算法原理, 一步步推导即可得出答案, 耐心推导并不难。但要注意, 如果遇到算法原理十分复杂的, 建议放弃, 掌握问题 1 和 2 的技巧即可。

常见问题:

Q: 文老师说这一题最难, 我也没基础, 能不能干脆放弃?

A: 这种思想是错误的, 这一题共 15 分, 放弃就难通过了, 要尽可能的拿分, 至少关于时间复杂度、算法判断是很简单的; 而算法填空, 平时做练习也要认真对待每一题, 坚持下来, 时间长了即使零基础也能填对几个。

Q: 完全没有 C 语言代码基础, 看不懂怎么办?

A: 文老师在讲解真题和技巧时已经充分考虑到这一点, 对于零基础学员, 先认真听语法讲解(特地将与其他平台合作的课程里关于 C 语言部分剪辑出来给大家学习), 然后听真题里对代码的分析, 从英语的角度去联想其含义。不放弃就有收获。

Q: 算法判断不准确, 不能辨别贪心法和动态规划法怎么办?

A: 算法判断是很简单的了, 题目中都有关键词暗示的, 分而治之是分治法; 迷宫问题回溯法; 当题目中给出“考虑当前情况下”等词, 是局部最优; 当题目给出“最优子结构”、递归、最优公式等, 是动态规划法。

Q: 时间复杂度不知道如何求?

A: 看代码循环重数, 注意是嵌套的重数, 不是个数。

Q: 这种题目该怎么学习, 学习到哪种程度呢?

A: 要求能拿到 8 分以上; 看完视频对应专题课程后, 立即去做课后习题(选自 17-19 年真题), 做完后立即去看对应真题视频讲解, 掌握技巧。

详细示例及应用见文老师课程视频, 有详细技巧的应用讲解。

十九 面向对象的程序设计与实现

1 C++ 基本语法

有 C++ 基础的可选本题, 无基础都选 JAVA, 这里删除, 不做扩展。

2 JAVA 基本语法

JAVA 中类的定义如下:



类的定义格式如下：

[import包]

[类修饰符] class xxxclass [extends超类] [implements 接口] {

public:

公有数据成员或公有函数成员的定义；

protected:

保护数据成员或保护函数成员的定义；

private:

私有数据成员或私有函数成员的定义；

}

说明：

- import包：引入包中的类。
- 类修饰符：主要有四个修饰符，public、abstract、final、private。
- class为关键字，xxxclass为类名，命名遵循Java标识符的命名规则。
- extends为继承关键字，implements 为接口关键字。

示例如下：

```
class Department{/*代码省略*/}
```

```
class SqlserverDepartment (3) {}
```

```
abstract class Shape{  
abstract public void draw()  
}
```

```
class Rectangle extends Shape{  
}
```

当使用 **extends** 关键字时，表明继承的是一般类，如上图，继承的是抽象类，要加 **abstract** 关键字。

JAVA 中的接口是一种特殊的类，其内部的实现方法只有定义，没有实现，这就是特殊所在，接口定义如下：

```
interface IFactory{}
```

```
class SqlServerFactory implements IFactory
```

若类使用 **implements** 关键字，表明是接口类。

3 解题技巧

下午考试的第 5, 6 题为二选一作答, 都是程序填空, 原理一模一样, 只不过一个用 C++ 语言, 一个用 JAVA 语言编写的程序, 并且这个填空只是考基本语法, 基本不涉及算法, 比之第 4 题算法设计的填空简单很多, 完全可以**拿满分**。

文老师建议, 如果是初学者, 或者对于两种语言都不太精通, 就**专攻 JAVA 程序题**, 因为 JAVA 的语法比之 C++ 要简单并且容易理解记忆, 容易拿到满分。

面向对象的程序填空分为两类, 一个是**考察纯定义**, 如接口类, 抽象类, 接口类中的函数定义等, 这些根据程序代码可以快速判断出。

另一类, 就是关于**设计**的, 填写函数体, 但是这个函数体并不是要写一段真正的程序实现代码, 而是调用形式的, 如上面的例题, 都有调用函数的, 这些调用函数一般在程序中, 或者在说明和类图中可以找到, 考察的是**调用形式**。

注意以下几点:

- 1、定义了类的对象后, 必须先**初始化** (使用 new 关键字);
- 2、接口 (interface) 和抽象类 (abstract class) (子类继承关键字不同, 为 implements 和 extends);
- 3、**抽象类中可以有普通的方法** (有函数体), 也可以有抽象方法 (无函数体, 方法前要加关键字 abstract), 而接口中的方法都是默认为抽象方法 (因为默认, 无需再加任何关键字标识);
- 4、this 的使用, 指代当前对象, 一般有两个重名变量的赋值时会使用到, 如在构造函数中, 参数名和私有变量名相同都为 name, 就要使用 this.name=name;
- 5、题目所给的**类图很重要**, 从中可以查看类之间的关系以及类中的方法。

JAVA 题总结:

- 1、抽象类、接口相关, 继承、实现关键字, 抽象类中有抽象方法和普通方法, 抽象方法加 abstract, 接口中方法都是抽象方法, 因此反而**不需要加 abstract**。
- 2、**类的成员变量、类的方法中参数**很重要, 若参数是类的对象, 则一般会在实现中使用此对象调用类的方法; 另外赋值中要注意 this 的使用, 当参数名和变量名相同时使用 this。
- 3、结合代码的上下文, 明确类之间的关系, 结合方法实现的功能, 方法中的参数, 以及类的成员变量, 去解决问题。

记住: JAVA 填空都不难, 只是考简单的语法问题, 或者类之间的关联, 方法的调用, 不涉及算法原理。

常见问题:

Q: 零基础完全不懂 JAVA 语言怎么办?

A: 这一题完全可以从英语角度去理解, 根据关键词去搜索代码上下文, 都能得出答案, 毫无难度, 每个人都能拿满分, 要先树立信心。

Q: 视频里为什么没有对于 JAVA 语言的详细讲解?

A: 语言之间都是相通的, 结合文老师前面对 C 语言的讲解, 学员再去分析 C 语言或者 JAVA 语言, 其关键语法都是类似的, 这个考试又不需要大家去关注细节, 因此完全够用了。如果了解具体 JAVA 语法可在百度搜索“java 教程”, 找一个简短的页面整体浏览一遍即可, 没有必要去看相关 java 代码视频。关键还在于多做真题, 多听视频讲解, 自己多总结。

Q: 这种题目该怎么学习, 学习到哪种程度呢?

A: 要求能拿到 12 分以上; 看完视频对应专题课程后, 立即去做课后习题 (选自 17-19 年真题), 做完后立即去看对应真题视频讲解, 掌握技巧。

免费题库 - 微信搜索: 软考达人