

- 1.遇到问题有点陌生，或者你就是想思考一下。可以说老师我需要考虑一分钟再回答。（是被允许的，且也能尽量把语言组织好）
- 2.非常感谢老师的提问，对于这个问题，在学习过程中，暂时还未涉及到，不过根据我先有的知识和理解，我觉得是不是可以从以下三个方面来看，首先.....其次.....最后.....。
- 3.这个问题我确实不是很清楚，请老师赐教。

目录

数据库	5
1.游标是什么？作用？	5
2.索引的作用	6
3.DBA（数据库管理员）的职责？	6
4.什么是事务？	6
5.事务四大特性（ACID）	6
6.并发控制（没记住）	7
7.并发一致性问题	7
8.封锁	7
9.事务的隔离级别	8
10.什么是视图	8
11.什么是存储过程？优缺点？（没记住）	8
12.三大范式（规范化）	9
13.ER 图（实体关系图）、关系模型	9
14.DBMS 的数据模型	10
15.三大模式/三级模式，两层映像（没记住）	11
16.数据库的事务故障，DBMS 如何恢复数据库？	11
17.数据库的系统故障，DBMS 如何恢复数据库？（没记住）	11
18.数据库的介质故障，DBMS 如何恢复数据库？	11
19.数据库的一些基本指令（29）	12
20.主键、外键（没记住）	12
21.动态数据库（DLL）（没记住）	12
计算机网络	12
1.Linux 和 Unix 的区别	12
2.数字签名、加密算法（对称算法和非对称算法）	13
3.子网掩码	13
4.计算机网络模型（OSI、TCP/IP）	13
5.计算机网络为什么要分层？优点？	15
6.流量控制和拥塞控制（没记住）	15
7.TCP/IP 的流量控制（滑动窗口）（没记住）	16
8.CSMA/CD 协议	16
9. TCP/IP 中传输层的作用	17
10.网络层协议与运输层协议的区别	17
11.ARP 地址解析协议、RARP	17
12.IP 地址、MAC 地址、子网掩码、子网（没记住）	18
13.协议三个核心要素（没记住）	18
14.TCP 的三次握手和四次握手，为什么采用三次握手，二次握手可以吗？（5）	18
15.TCP、UDP 协议特点、头部结构（5）	20

16.TCP 拥塞控制，慢开始、拥塞避免、快重传、快恢复（4）	22
17.TCP 建立连接的过程，如何保证可靠传输？（6）	22
18.TCP 快速重传机制（4）	23
19.DNS 域名解析（5）	23
20.点击页面一次 HTTP 过程，输入一个网址，发什么？（3）	24
21.HTTP 和 HTTPS（4）	24
22.网卡（3）	25
23.socket（4）	25
24.cookie 和 session（3）	25
25.集线器、路由器、交换机	26
26.WWW 万维网	26
27.IP 地址分段	27
28.单播、多/组播	27
29.IPV4、IPV6	27
30.C/S、B/S	27
31.P2P	27
32.曼彻斯特编码、差分曼彻斯特编码	27
33.MAC 地址	28
34.PPP 点对点协议	28
数据结构	28
1.栈和队列的区别	28
2.顺序结构和链式结构的区别？	28
3.二叉树和度为二的树的区别	28
4.二叉树概念和性质	28
5.二叉树的顺序存储和二叉链表	29
6.最优二叉树（哈夫曼树）	29
7.图的概念	29
8.哈希表（散列表）	30
9.稳定的排序算法	30
10.二路归并算法、快速排序算法	30
11.时间复杂度、空间复杂度	30
12.头指针、头节点区别	31
13.判断链表是否有环	31
14.循环队列	31
15.KMP 算法（字符串匹配算法）	31
16.BFS	31
17.DFS	32
18.最小生成树	32
19.Prim 算法（最小生成树）	32
20.Kruskal 算法（最小生成树）	32
21.红黑树（RBT）	32
22.B 树和 B+树（未完成）	33
23.堆	33
24.排序	34

25.线索化	34
操作系统	34
1.并行、并发	34
2.线程、进程	34
3.死锁	35
4.总线	36
5.分段、分页	36
6.页面替换算法	36
7.进程（作业）调度算法	36
8.操作系统特征	36
9.中断	37
10.进程互斥、同步	37
11.内核态和用户态	37
12.进程间通信方式	37
13.内存中堆、栈	38
14.磁盘调度算法	38
15.逻辑地址、物理地址	38
16.分页怎么优化	38
17.哲学家进餐有那些实现方式	38
18.虚拟存储器	38
19.并行处理	39
20.作业、进程	39
计算机组成原理	39
1. 内存、外存	39
2.RAM、ROM	40
3.指令	40
4.寻址方式	40
5.Cache 和主存之间的映射	41
6.虚拟存储器	41
7.冯诺依曼机器	41
8.CISC 和 RISC	41
9.MAR、MDR、存储器最大容量	42
10.机器字长、存储字长	42
11.奇偶校验码、汉明码	42
12.补码表示真值范围	43
13.IO 控制方式	43
14.原码、反码、补码	43
15.流水线技术	43
16.衡量计算机性能指标	43
17.指令周期	44
18.cache	44
19.如何判断整数数据溢出	44
20.微程序、微指令、微命令	44
21.CPU 的组成部分	44

编译原理	45
1.编译过程的五个基本步骤	45
2.有限自动机 (DFA)	45
软件工程	45
1.设计模式	45
2.黑盒、白盒测试	46
3.软件危机	46
4.数据库完整性、数据完整性、实体完整性、参照完整性、用户定义完整性	46
离散数学	47
1.笛卡尔集	47
C++ (真题)	47
1.++、--运算符重载	47
2.为什么析构函数要用虚函数	47
3.浅构造、深构造	48
4.全局变量和局部变量在内存中是否有区别?如果有,是什么区别?	48
5.引用与指针有什么区别?	48
程序设计 (真题)	49
1. 什么是比较好的编码风格	49
2.如何不用循环的方式判断 2 的 n 次方 (如何递归)	49
3. 01 背包复杂度	49
4.单淘汰相关算法	49
5.简述弗洛伊德算法求最短路径或如何求最短路径	49
6.在字符串中找出只出现一次的数字, 其余数字出现两次	49
7.排序的时间、空间复杂度	49
8.字符串匹配算法	50
9.图书管理系统, 图书用什么结构保存, b 树	50
10.简述二叉树左右子树逆置	50
11.迪杰斯特算法求最短路径	50
12.最小生成树算法	50
13.On 实现链表反转	50
14.有序数组建立平衡二叉树	50
15.如何判断无向图是树	50
16.二叉树节点相关性质	50
17.改进字符串匹配	51
其他 (真题)	51
1.为什么报我们学校 (英语口语里有)	51
2.你想读什么方向	51
3.你的研究生计划是什么 (英语口语里有)	51
4.项目是什么规模的项目, 用的什么语言, 这个功能你怎么实现的	51
5.最近在看什么书	51
6.你的毕业论文做的什么, 用了什么技术	52
7.你还有什么想问我们的	52
专业问题 (真题)	52
1.线程、进程概念及其区别 (上面有)	52

2.为什么说操作系统是由中断驱动的（上面有）	52
3.递归栈会不会溢出，为什么	52
4.栈泄漏是什么，栈与堆的区别	52
1.十进制、八进制、二进制转换	53
1.为什么要三次握手和四次挥手	53
1.正交矩阵是什么	53
1.范式模型（数据库）（上面有）	53
2.蠕虫病毒怎么到我电脑上的	53
3.编译器原理（编译—>链接—>运行）	53
4.反射是什么，请用 c 语言实现一下	54
5.不用加减乘除算除法	54
生活问题（真题）	54
1.是否有读博打算？	54
2.有没有提前联系导师	54
3.为什么读研？	54
4.本科期间有什么科研成果？	54
5.你的本科成绩为什么不够出彩？	55
6.某一科为什么这么好？	55
C++代码编写注意事项	55
1.虚继承	57
2.多态（虚函数）	58
3.虚析构函数、虚函数	58
4.纯虚函数、抽象类	58
5.Vector	59
6.操作符重载	59

数据库

1.游标是什么？作用？

1.作用：游标是 SQL 的一种数据访问机制，游标简单的看成是结果集的一个指针，可以根据需要在结果集上面来回滚动，浏览我需要的数据。

2.原因：众所周知，使用 SQL 的 select 查询操作返回的结果是一个包含一行或者是多行的数据集，如果我们要对查询的结果再进行查询，比如（查看结果的第一行、下一行、最后一行、前十行等等操作）简单的通过 select 语句是无法完成的，因为这时候索要查询的结果不是数据表，而是已经查询出来的结果集。游标就是针对这种情况而出现的。

2.索引的作用

1.定义: 索引问题就是**查找问题**, 相当于图书的目录, 可根据目录中的页码快速找到所需的内容。数据库索引, 是数据库管理系统中一个**排序的数据结构**, 以协助**快速查询, 更新数据库中的数据**。索引的实现通常使用 **B 树** 和变种的 **B+树** (mysql 常用的索引就是 B+树)

2.常见的索引结构: a. B-Tree 索引 (平衡查找树索引); b. Hash 索引; c.全文索引;

3.优点: a.快速取数据; b.保证数据记录的**唯一性**; c.实现表与表之间的**参照完整性**; d.在 **group by** 和 **order by** 进行数据检索时, 利用索引可以减少排序和分组时间;

4.缺点: a.创建和维护索引需要耗费时间; b.索引需要占用物理空间;

5.什么样的字段适合索引: a.经常需要搜索的列上; b.作为主键的列上; c.经常排序的列上;

3.DBA (数据库管理员) 的职责?

1.决定数据库中的信息内容和结构

2.决定数据库的**存储结构和存取策略**, 获得较高的存取效率和存储空间利用率。

3.定义数据的**安全性要求和完整性约束条件**, 负责确定各个用户对数据的权限数据保密级别和完整性约束条件

4.监控数据库的使用、运行、转储数据、维护日志文件, 故障恢复。

5.数据库的改进和重组重构, 对运行情况进行记录, 统计分析,

4.什么是事务?

事务 (Transaction) 是并发控制的基本单位。事务是一个操作序列, 这些操作要么都执行, 要么都不执行, 它是一个不可分割的工作单位。事务是数据库维护数据一致性的单位, 在每个事务结束时, 都能**保持数据一致性**。

5.事务四大特性 (ACID)

1.原子性: 事务是一个完整的操作, 各操作是不可分的。要么操作全部执行, 要么不执行

Eg: 以银行转账事务为例, 如果该事务提交了, 则这两个账户的数据将会更新。如果由于某种原因, 事务在成功更新这两个账户之前终止了, 则不会更新这两个账户的余额, 并且会撤销对任何账户余额的修改, 事务不能部分提交。

2.一致性: 事务前后, 数据总额一致。数据库在事务执行前后都保持一致性状态。在一致性状态下, 所有事务对一个数据的读取结果都是相同的。

Eg: 以银行转账事务为例。在事务开始之前, 所有账户余额的总额处于一致状态。在事务进行的过程中, 一个账户余额减少了, 而另一个账户余额尚未修改。因此, 所有账户余额的总额处于不一致状态。事务完成以后, 账户余额的总额再次恢复到一致状态。

3.隔离性: 所有的操作全部执行完以前, 其他事务不能看到过程;

Eg: 另外, 当事务修改数据时, 如果任何其他进程正在同时使用相同的数据, 则直到该事务成功提交之后, 对数据的修改才能生效。张三和李四之间的转账与王五和赵二之间的转账, 永远是相互独立的。

4.持久性：一旦事务提交，对数据的改变就是永久的。

6.并发控制（没记住）

1.并发控制是确保及时纠正由并发操作导致的错误的一种机制。

2.基本单位：事务。

3.并发控制：当多个用户同时更新运行时，用于保护数据库完整性的各种技术。并发机制不正确可能导致脏读、不可重复读、幻读等问题。

4.目的：是保证一个用户的工作不会对另一个用户的工作产生不合理的影响。

7.并发一致性问题

1.丢失数据（同时修改）：T1 和 T2 两个事务对一个数据进行修改，T1 先修改，T2 后修改，T2 的修改覆盖了 T1 的修改。简记为同时修改。

2.脏读（读取未提交的数据）：T1 对一个数据做了修改，T2 读取这一个数据。若 T1 执行 ROLLBACK 操作，则 T2 读取结果和第一次结果不一样。简记为读取失败的修改。场景：修改完后，紧接着查询结果。

3.不可重复读（前后多次读取，数据内容不一致）：T2 读取一个数据，T1 对该数据做了修改。如果 T2 再次读取这个数据，此时读取的结果和第一次读取的结果不同。重复读取的结果不同。

4.幻读（前后多次读取，数据总量不一致）：T1 读取某个范围的数据，T2 在这个范围内插入新的数据，T1 再次读取这个范围的数据，此时读取的结果和第一次读取结果不同。重复读取的结果不同。

5.解决方案：发生原因：在并发环境下，事务隔离性很难保证，因此会出现并发一致性问题。解决方案是通过并发控制来保证隔离性。并发控制可以通过封锁来实现。但封锁操作需要用户自己控制，相当复杂。数据库管理系统提供了事务隔离级别，让用户方便处理并发一致性问题。

8.封锁

1.封锁粒度：行级锁、表级锁。应尽量只锁定需要修改的那部分数据，而不是所有资源。

2.封锁类型

a.读写锁：

排它锁 (Exclusive)，简称为 X 锁，又称写锁。

共享锁 (Shared)，简称为 S 锁，又称读锁。

一个事务对数据对象 A 加了 X 锁，就可以对 A 进行读取和更新。加锁期间其它事务不能对 A 加任何锁。

一个事务对数据对象 A 加了 S 锁，可以对 A 进行读取操作，但是不能进行更新操作。加锁期间其它事务能对 A 加 S 锁，但是不能加 X 锁。

b.意向锁：

意向锁在原来的 X/S 锁之上引入了 IX / IS，IX / IS 都是表级别的锁，用来表示一个事务稍后会对表中的某个数据行上加 X 锁或 S 锁。整理可得以下两个规定：

一个事务在获得某个数据行对象的 S 锁之前，必须先获得表的 IS 锁或者更强的锁；
一个事务在获得某个数据行对象的 X 锁之前，必须先获得表的 IX 锁。

9.事务的隔离级别

隔离级别	脏读	不可重复读	幻读
读未提交 (Read uncommitted)	×(未解决)	×	×
读已提交 (Read committed)	√(解决)	×	×
可重复读 (Repeatable read)	√	√	×
可串行化 (Serializable)	√	√	√

1.读未提交：都不能避免。

a.原因：在这种隔离级别下，所有事物能读取其他事务未提交的数据，造成脏读。

b.解决：使用读已提交。

2.读已提交（Oracle、SQL server 采用）：可避免脏读的发生。

a.原因：在这种隔离级别下，所有事物只能读取其他事务已经提交的内容。能够解决脏读。但是会出现一个事务前后多次的查询中返回了不同内容的数据现象，造成不可重复读。

b.解决：使用可重复读

3.可重复读（Mysql 采用）：可避免脏读、不可重复读的发生。

a.原因：在这种隔离级别下，所有事务前后多次读取的数据内容是不变的。也就是某个事务在执行过程中，不允许其他事务执行 update 操作，但允许其他事务执行 add 操作，造成事务前后多次读取的数据总量不一致的现象，产生幻读。

b.解决：使用串行化

4.可串行化（一般不使用）：可避免脏读、不可重复读、幻读的发生。

a.原因：在这种隔离级别下，所有事务顺序执行，之间不会发生冲突，可以解决脏读、不可重复读、幻读。但是安全和效率不能兼得，导致大量操作超时和锁竞争，降低数据库性能。

10.什么是视图

1.概念：是一种虚拟的表，通常是有一个表或者多个表的行或列的子集。具有和物理表相同的功能。可以对视图进行增、改、查、操作，对视图的修改会影响真实表。

2.作用：相比多表查询，使得我们获取数据更容易。

11.什么是存储过程？优缺点？（没记住）

存储过程简单来说就是为了以后使用而保存的一条或多条预编译 SQL 语句，这些语句块像一个方法一样执行一些功能。

优点：

1.类似于封装，简化操作；

2.不用反复建立一系列处理步骤，保证了数据的完整性；

3.安全性高：只有特定的用户才有权限使用，完成某个特定功能的存储过程一般只有特定的用户可以使用，具有使用身份限制，更安全；

4.存储过程是一个编译过的代码块，速度快，性能高；

缺点：

- 1.存储过程的编写比基本 SQL 语句复杂，需要较高的技能；
- 2.可能没有创建存储过程的权限；

12.三大范式（规范化）

1.作用：处理以下异常

冗余数据：例如 学生-2 出现了两次。

修改异常：修改了一个记录中的信息，但是另一个记录中相同的信息却没有被修改。

删除异常：删除一个信息，那么也会丢失其它信息。

插入异常：例如想要插入一个学生的信息，如果这个学生还没选课，那么就无法插入。

2.第一范式（1NF）：每列属性不可再分；符合数据表的原子性。同一列中不能有多个值，即实体中的某个属性不能有多个值或者不能有重复的属性。

Eg：表中一项有 aa 和 bb，那么必须拆分成两项：aa 一项、bb 一项；

歌手		歌手
iKun、吴亦凡		iKun
李易峰		李易峰
		吴亦凡

拆分为

3.第二范式（2NF）：满足第一范式的基础上，表中的非主属性必须完全依赖主属性，每张表只描述一件事情，有主键；

学号	姓名	为何入狱
1	iKun	发放律师函
2	李易峰	细狗
3	吴亦凡	唱大碗宽面

学号决定姓名和为何入狱。

解决方法，拆表

4.第三范式（3NF）：满足第二范式基础上，消除传递依赖，每列的数据都和主键直接相关，不可以间接相关；

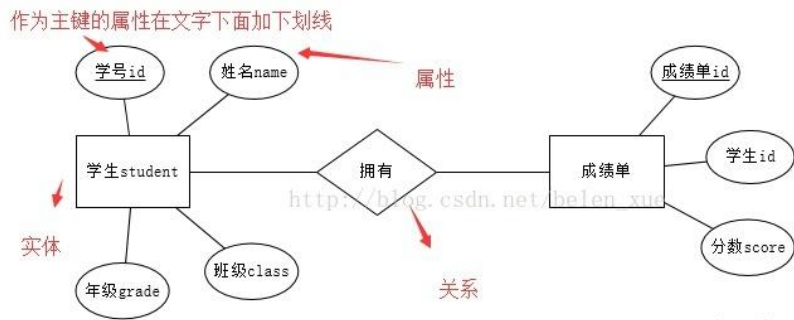
Eg：在同一张表中，A 依赖于 B，B 依赖于 C。那么应该拆表，把 A 和 B 一张表，B 和 C 一张表。

5.BCNF：消除主属性对主键的部分与传递依赖。

Eg：在同一张表中 A 依赖于 B，A 依赖于 C，那么应该拆表为 A 和 B 一张表，A 和 C 一张表

13.ER 图（实体关系图）、关系模型

1.组成部分：实体（矩形）【现实生活中的对象】、属性（椭圆）【实体的特性】、联系（菱形）【事物之间的关系】。用来进行设计关系型数据库系统



知乎 @麻辣仙女

2.ER 模型转换为关系模式的原则：

实体关系：一对一、一对多、多对多

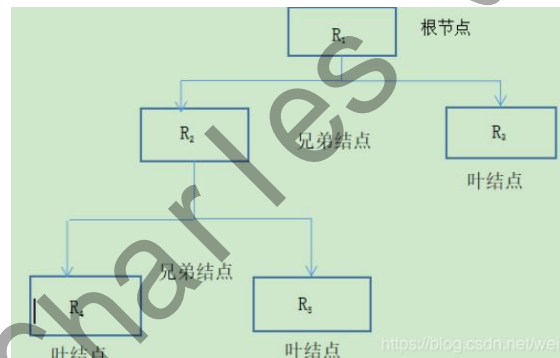
一对一：遇到一对一关系的话，在两个实体任选一个添加另一个实体的主键即可。

一对多：遇到一对多关系的话，在多端中添加一端的主键。

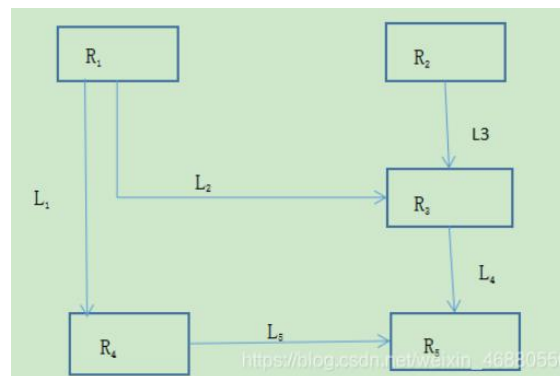
多对多：遇到多对多关系的话，我们需要将联系转换为实体，然后在该实体上加上另外两个实体的主键，作为联系实体的主键，然后再加上该联系自身带的属性即可。

14.DBMS 的数据模型

1.层次模型：数据被组织成树状结构，两种数据类型之间具有一对多的关系。



2.网状模型：数据被组织成图。



3.关系模型：数据被组织在二维表中。

行和列组成。			
学号	姓名	年龄	性别
2013004	王小明	19	女
2013006	黄大鹏	20	男
2013008	张文斌	18	女

15.三大模式/三级模式，两层映像（没记住）

1.三大模式

a.内模式：又称存储模式，对应于物理级。

b.外模式：又称子模式或用户模式，对应于用户级。

c.概念模式：又称逻辑模式，对应概念级。

2.两级映像

A.作用：保障了数据库中的数据具有较高的物理独立性和逻辑独立性（数据独立性）。

B.外模式/模式：实现了外模式到概念模式之间的相互转换。

C.模式/内模式：实现了概念模式到内模式之间的相互转换。

D.数据独立性：数据与程序独立，将数据从程序中分离出去，由 DBMS 管理，从而简化应用程序，大大减少应用程序编制的工作量。

16.数据库的事务故障，DBMS 如何恢复数据库？

1.事务故障：指某个事务在运行过程中由于种种原因未运行至正常终止点就夭折了。

2.恢复方法：撤销事务。即清除该事务对数据库的所有修改，使得这个事务像根本没有启动过一样。（需要从后到前撤销，最新完成的操作的更新影响要先消失。因此，需要从后到前扫描日志文件。）

17.数据库的系统故障，DBMS 如何恢复数据库？（没记住）

1.系统故障：是由于某些原因系统停止运转，使得系统需要重新启动（数据库软件、操作系统漏洞、突然停电）。

2.恢复方法：撤销所有未提交的事务，重做所有已提交的事务。

①清除尚未完成的事务对数据库的所有修改，UNDO（撤销）所有未完成的事务（从后往前）。

②将缓冲区中已完成事务提交的结果写入数据库，REDO（重做）所有已提交的事务（从前往后）。

18.数据库的介质故障，DBMS 如何恢复数据库？

1.介质故障：是指硬件故障使存储在硬盘上的数据丢失（磁头碰撞、磁盘损坏、强磁干扰）。

(破坏性最大)

2.恢复方法: ①在新磁盘上, 导入最新的数据库备份文件。

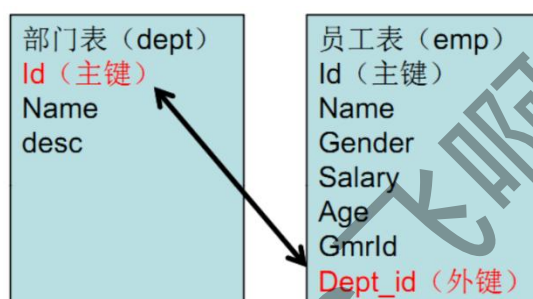
②根据日志文件恢复数据, 找出来从该最新备份后开始, 到故障发生时, 哪些事务已经完成。REDO(重做)所有已提交的事务。(从前往后)。

19.数据库的一些基本指令 (29)

20.主键、外键 (没记住)

1.主键: 是唯一标识一条记录, 不能有重复, 不允许为空, 用来保证数据完整性;

2.外键: 是另一张表的主键, 外键可以有重复的, 可以是空值, 用来跟其他表建立联系用的。所以说, 如果谈到了外键, 一定至少涉及到两张表。



21.动态数据库 (DLL) (没记住)

动态数据库 (DLL) 是作为共享函数库的可执行文件, 动态数据库提供了一种方法, 使进程可以调用不属于其可执行代码的函数。

计算机网络

1.Linux 和 Unix 的区别

1.linux 是开发源代码的自由软件, Unix 是对源代码实行知识产品保护的传统商业软件。这也是最大不同。

2.Unix 大多是与硬件配套, Linux 则可以运行在多种硬件平台上。

3.Linux 的核心是免费的开源的, 而 Unix 的核心并不公开。

4.Linux 的开发是处于一个完全开放的环境之中, Unix 的开发完全处于黑箱之中, 只有相关的开发人员才能接触到。

2.数字签名、加密算法（对称算法和非对称算法）

1.数字签名: 提供可甄别的数字信息验证自身身份的一种方式，由发送方持有能够代表自己身份的私钥，有接收方持有与私钥对应的公钥，能够在接受发送者信息时用于验证其身份。

2.对称加密算法: 加密和解密使用同一个密钥，所以叫对称加密。对称加密只有一个密钥，作为私钥；

优点：算法公开、计算量小、加密速度快、加密效率高。

缺点：密钥的管理和分发困难，不够安全。

3.非对称加密算法: 加密和解密使用不同的密钥。公钥用来加密，私钥用来解密。私钥只能由一方安全保管，不能外泄，公钥则可以发给任何请求他的人。

有点：安全性高，公钥是公开的，私钥是自己保存的。

缺点：加密和解密花费时间长、速度慢

3.子网掩码

子网掩码（网络掩码、地址掩码），他用来指明一个 IP 地址的那些位标识的是主机所在的子网，哪些位标识的是主机的位掩码。子网掩码不能单独存在，必须结合 IP 地址一起使用。

作用：将 IP 地址划分为网络号和主机号。

其中网络号为 1，主机号为 0；

位置：网络号+主机号

4.计算机网络模型（OSI、TCP/IP）

OSI模型	五层模型	TCP/IP模型
应用层	应用层	应用层
表示层		
会话层		
传输层	传输层	传输层
网络层	网络层	网络层
数据链路层	数据链路层	主机-网络层
物理层	物理层	

OSI 模型：物联网淑惠使用（物、链、网、输、会、示、用）；

主机-网络层=网络接口层

1.物理层

数据输出单元：比特

利用传输介质为通信的网络结点之间建立、管理和释放物理连接。

提出了物理层设备的机械特性、电气特性、功能特点

实现了比特流的透明传输，为数据链路层提供数据传输服务

解决使用何种信号来传输比特的问题

代表设备：中继器、集线器

2.数据链路层

可以使用物理层提供的服务，在通信的实体间建立链路链接，能实现相邻结点通信。采用差错控制（停止等待协议）与流量控制方法，使有差错的物理线路变成无差错的物理链路。

传输“帧”为单位的数据包，数据组装成帧

解决分组在一个网络上传输的问题

典型设备：二层交换机

3.网络层

数据传输单元：分组（数据报）

由路由选择算法为分组通过通信子网选择适当传输路径（为数据在结点之间传输创建逻辑链路）实现流量控制、拥塞控制和网络互连等功能。能寻找到达对方主机的路径（即IP寻址）。

解决数据报在多个网络上路由选择问题。

典型设备：路由器（router、TPlink）、三层交换机。

4.传输层

数据传输单元：报文段。

解决进程之间的通信问题。向用户提供可靠端到端（端口）服务。向高层屏蔽下层数据通信的细节。

典型代表协议：TCP、UDP 协议。

5.会话层

对传输数据进行管理，与对方建立会话。负责维护两节点传输不中断。

允许不同主机上的各个进程间进行会话。

6.表示层

为应用层进程提供格式化表示和转换数据服务。协议：jpeg

处理两个通信系统中交换信息的表示方式。关注传递信息的语法语义。

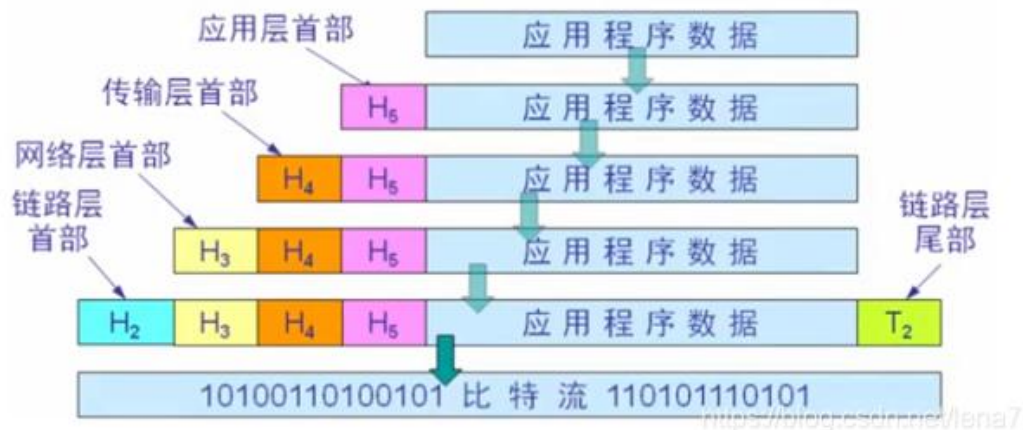
数据格式变换、数据加密与解密、数据压缩与恢复、编码。

7.应用层

为应用程序提供网络服务，需识别并保证通信对方的可用性，使协同工作的程序能同步。

建立传输错误纠正与保护数据完整性的控制机制。

解决通过应用进程的交互来实现特定网络应用的问题。



各层协议：

- 1.应用层：FTP（文件传输协议）、SMTP（电子邮件协议）、HTTP（超文本传输协议）、DNS（域名解析服务）
- 2.传输层：TCP（传输控制协议）、UDP（用户数据报协议）
- 3.网络层：IP、ARP、OSPF
- 4.链路层：PPP、以太网、帧中继

传输单元：

- 1.应用层：报文
- 2.传输层：报文段
- 3.网络层：数据报
- 4.链路层：帧
- 5.物理层：比特

5.计算机网络为什么要分层？优点？

1.作用：分层是为了封装对下层的变化；

2.优点：

- A.分层可以将复杂的问题划分成若干易于处理的问题，降低问题的复杂性；
- B.各层之间相互独立。某一层不需要知道它下一层是如何实现的，某层仅仅需要知道如何使用下一层接口所提供的服务。
- C.灵活性好，易于维护和升级。当任何一层发生变化时，只要层间接口关系保持不变，其他层均不受到影响。

6.流量控制和拥塞控制（没记住）

1.流量控制：如果发送方把数据发送的过快，接收方可能会来不及接受，会造成数据丢失。（让发送方慢点，要让接收方来得及接受）

2.拥塞控制：防止过多的数据注入网络中，防止网络中的路由器或链路过载。解决方法：慢开始+拥塞避免，快重传+快恢复。

3.关系：流量控制是为了预防拥塞。流量控制指点对点通信量的控制。拥塞控制是全局性的，涉及到所有主机和降低网络性能的因素。

7.TCP/IP 的流量控制（滑动窗口）（没记住）

1.流量控制：对发送端和接收端而言，TCP 需要把发送的数据放到发送缓冲区，将接收的数据放到接收缓冲区。流量控制就是控制接收缓冲区的大小来控制发送端的发送速度。

2.作用：如果发送端把数据发送的过快，接收方可能会来不及接受，会造成数据丢失。（让发送发慢点，要让接收方来得及接受）

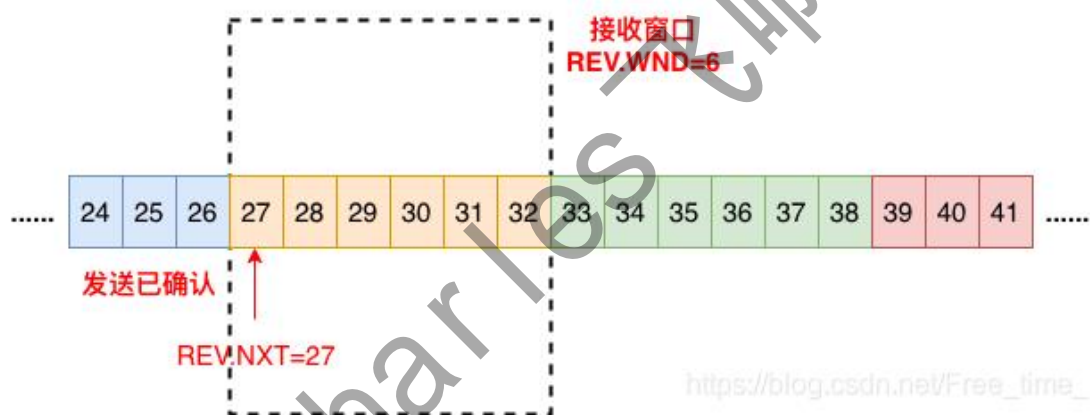
3.原理：滑动窗口实现流量控制

4.滑动窗口：发送窗口、接收窗口。

发送窗口：



接收窗口：



8.CSMA/CD 协议

1.作用：解决当多个主机同时发送数据，发生碰撞问题。

2.CSMA 载波侦听多路访问：方法是发前先听（终端设备在发送前要先侦听线路状态），如果侦听到信号忙则推迟发送，如果侦听到信号空闲则立即发送整个帧。冲突（存在另一个设备同时发送数据）也要发完，因此会浪费信道。

3.CSMA/CD 载波侦听多路访问/冲突检测技术：方法是发前先听，如果听到信号忙则推迟发送，如果侦听到信号空闲则立即发送整个帧。边发边听，冲突则中止发送，发送特殊阻塞信息（以强化冲突信号，使线路上其他站点能够尽早检测到冲突），等待一段时间后再发送数据，可以节省信道。

9. TCP/IP 中传输层的作用

类似于传输层：

定义传输数据的协议端口号，以及流量控制和差错校验。

协议有：TCP UDP 等，数据包一旦离开网卡即进入网络传输层。

定义了一些传输数据的协议和端口号（WWW 端口 80 等），如：TCP（传输控制协议，传输效率低，可靠性强，用于传输可靠性要求高，数据量大的数据），UDP（用户数据报协议，与 TCP 特性恰恰相反，用于传输可靠性要求不高，数据量小的数据，如 QQ 聊天数据就是通过这种方式传输的）。主要是将从下层接收的数据进行分段和传输，到达目的地后再进行重组。常常把这一层数据叫做段。OSI 模型中最重要的一层。传输协议同时进行流量控制或是基于接收方可接收数据的快慢程度规定适当的发送速率。除此之外，传输层按照网络能处理的最大尺寸将较长的数据包进行强制分割。例如，以太网无法接收大于 1500 字节的数据包。发送方节点的传输层将数据分割成较小的数据片，同时对每一数据片安排一序列号，以便数据到达接收方节点的传输层时，能以正确的顺序重组。该过程即被称为排序。工作在传输层的一种服务是 TCP/IP 协议套中的 TCP（传输控制协议），另一项传输层服务是 IPX/SPX 协议集的 SPX（序列包交换）。

10.网络层协议与运输层协议的区别

网络层协议负责的是提供主机间的逻辑通信。端到端通信。

运输层协议负责的是提供进程间的逻辑通信。进程间通信。

11.ARP 地址解析协议、RARP

1.ARP 作用：通过 IP 地址查询 MAC 地址

```
C:\Users\wwwli>ARP/A

接口: 192.168.48.1 --- 0x4

Internet 地址      物理地址      类型
192.168.48.254     00-50-56-ef-0e-1b 动态
192.168.48.255     ff-ff-ff-ff-ff-ff 静态
224.0.0.22         01-00-5e-00-00-16 静态
224.0.0.251        01-00-5e-00-00-fb 静态
224.0.0.252        01-00-5e-00-00-fc 静态
239.192.152.143    01-00-5e-40-98-8f 静态
239.255.255.250    01-00-5e-7f-ff-fa 静态
```

(1)首先，每个主机都会在自己的 ARP 缓冲区中建立一个 ARP 列表，以表示 IP 地址和 MAC 地址之间的对应关系。

(2)当源主机要发送数据时，首先检查 ARP 列表中是否有对应 IP 地址的目的主机的 MAC 地址，如果有，则直接发送数据，如果没有，就向本网段的所有主机发送 ARP 数据包，该数据包包括的内容有：源主机 IP 地址，源主机 MAC 地址，目的主机的 IP 地址。

(3)当本网络的所有主机收到该 ARP 数据包时，首先检查数据包中的 IP 地址是否是自己的 IP 地址，如果不是，则忽略该数据包。如果是，则首先从数据包中取出源主机的 IP 和 MAC 地址写入到 ARP 列表中，如果已经存在，则覆盖，然后将自己的 MAC 地址写入 ARP 响应包中，告诉源主机自己是它要找的 MAC 地址。

(4) 源主机收到 ARP 响应包后。将目的主机的 IP 和 MAC 地址写入 ARP 列表，并利用此信息发送数据。如果源主机一直没有收到 ARP 响应数据包，表示 ARP 查询失败。

广播发送 ARP 请求，单播发送 ARP 响应。

2.RARP: 反向地址转换协议，允许局域网的物理机器从网关服务器的 ARP 表或者缓存上请求其 IP 地址。

12.IP 地址、MAC 地址、子网掩码、子网（没记住）

1.IP 地址: 是网络中设备的唯一标识，每台连网计算机都依靠 IP 地址来标识自己。IPv4，32 位，包括网络部分和主机部分。

2.子网掩码: 用来指明一个 IP 地址的那些位是网络部分，那些位是主机部分。

3.子网: 当一个较大的网络被划分为较小的网络时，这就是子网。对于较小的子网，维护更容易。

4.MAC 地址: 用于在网络中唯一标识一个网卡，每个网卡都有一个全球唯一的 MAC 地址。
区别:

1. IP 地址可变，MAC 地址不可变。
2. 长度不同。IP 地址为 32 位，MAC 地址为 48 位。
3. IP 地址是网络层使用地址。MAC 地址是数据链路层使用地址。

13.协议三个核心要素（没记住）

(1) 语义。语义是解释控制信息每个部分的意义。它规定了需要发出何种控制信息，以及完成的动作与做出什么样的响应。

(2) 语法。语法是用户数据与控制信息的结构与格式，以及数据出现的顺序。

(3) 时序。时序是对事件发生顺序的详细说明。（也可称为“同步”）。

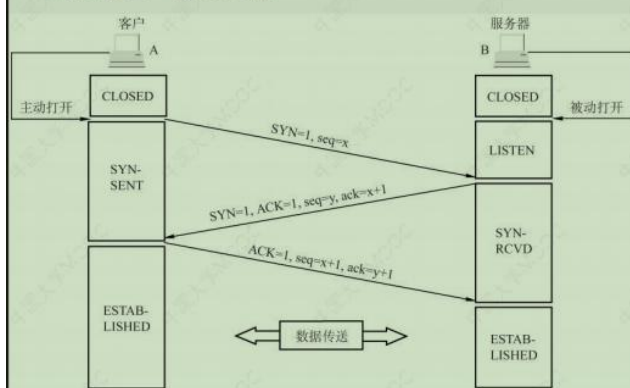
通俗的讲: 语义表示要做什么，语法表示要怎么做，时序表示做的顺序。

14.TCP 的三次握手和四次握手，为什么采用三次握手，二次握手可以吗？（5）

三次握手: 建立连接

TCP的连接建立

假设运行在一台主机（客户）上的一个进程想与另一台主机（服务器）上的一个进程建立一条连接，客户应用进程首先通知客户TCP，他想建立一个与服务器上某个进程之间的连接，客户中的TCP会用以下步骤与服务器中的TCP建立一条TCP连接：



ROUND 1:

客户端发送连接请求报文段，无应用层数据。

SYN=1, seq=x(随机)

ROUND 2:

服务器端为该TCP连接分配缓存和变量，并向客户端返回确认报文段，允许连接，无应用层数据。

SYN=1, ACK=1, seq=y(随机), ack=x+1

ROUND 3:

客户端为该TCP连接分配缓存和变量，并向服务器端返回确认的确认，可以携带数据。

SYN=0, ACK=1, seq=x+1, ack=y+1

1.SYN 洪泛攻击

2.SYN: 报文段首部中的同步位。

3.Seq=x: 选择一个初始序号（SYN 报文段不能携带数据，但是需要消耗一个序号）

4.确认号 ack: 其数值等于发送方的发送序号 seq+1（即接收方期望接收的下一个序列号）

5.Seq=y: ack=x+1: 发送的报文段的序号是 y，下一个想要接收到的报文段序号为 x+1；

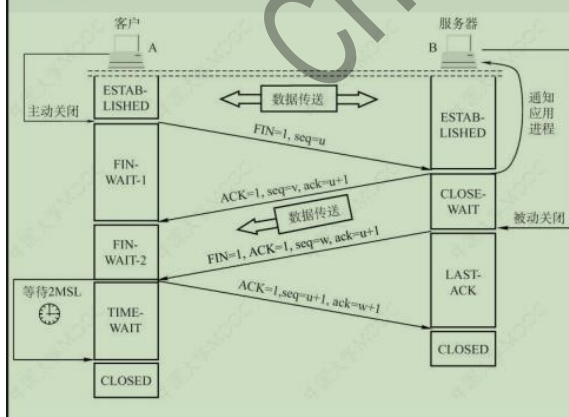
握手过程中传递的包里不包含数据，三次握手完毕后，客户端与服务器才正式开始传送数据。

理想状态下，TCP 连接一旦建立，在通信双方中的任何一方主动关闭连接之前，TCP 连接都被一直保持下去。

四次握手：断开连接

TCP的连接释放

参与一条TCP连接的两个进程中的任何一个都能终止该连接，连接结束后，主机中的“资源”（缓存和变量）将被释放。



ROUND 1:

客户端发送连接释放报文段，停止发送数据，主动关闭TCP连接。

FIN=1, seq=u

ROUND 2:

服务器端回送一个确认报文段，客户到服务器这个方向的连接就释放了——半关闭状态。

ACK=1, seq=v, ack=u+1

ROUND 3:

服务器端发完数据，就发出连接释放报文段，主动关闭TCP连接。

FIN=1, ACK=1, seq=w, ack=u+1

ROUND 4:

客户端回送一个确认报文段，再等到时间等待计时器设置的2MSL（最长报文段寿命）后，连接彻底关闭。

ACK=1, seq=u+1, ack=w+1

为什么采用三次握手？

采用三次握手是为了防止失效的连接请求报文段突然又传送到主机 B，因而产生错误，失效的连接请求报文段是指：主机 A 发出的连接请求没有收到主机 B 的确认，于是经过一段时间后，主机 A 又重新向主机 B 发送连接请求，且建立成功，顺序完成数据传输。考虑这一种特殊情况，主机 A 第一次发送的连接请求并没有丢失，而是因为网络结点导致延迟

达到主机 B，主机 B 以为是主机 A 又发出新的连接，于是主机 B 同意连接，并向主机 A 发出确定，但是此时主机 A 根本不会理会，主机 B 就一直等待主机 A 发送数据，导致主机 B 的资源浪费。

因而，二次握手不能满足上述情况，四次握手更没有必要。

15.TCP、UDP 协议特点、头部结构（5）

1.TCP：传输控制协议

A.TCP 是面向连接的，发送数据前需要再客户端和服务端间建立连接。

B.TCP 是一种可靠的数据传输。传送的数据无差错、不丢失、不重复、按序到达。

C.TCP 可以提供流量控制和拥塞控制。

D.TCP 是面向字节流的，数据是分批次发送。

E.每一条 TCP 连接只能有两个端点（即两个套接字），只能点对点的。

E.提供全双工通信。

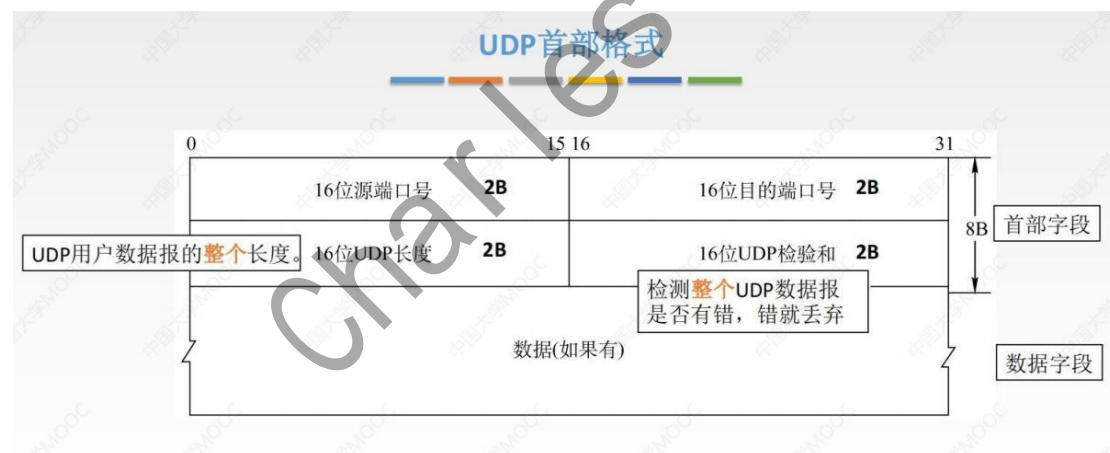
2.UDP：用户数据报协议

A.UDP 是面向非连接的，发送数据前不需要建立连接。

B.UDP 是一种不可靠的数据传输，只是尽最大可能交付而不提供其他保障。

C.UDP 不提供流量控制和拥塞控制，因此网络出现拥塞不会使源主机发送速率降低（UDP 对实时应用很有用，如视频会议）

D.UDP 使用面向报文的，发送端发送什么，接收端就接受什么。

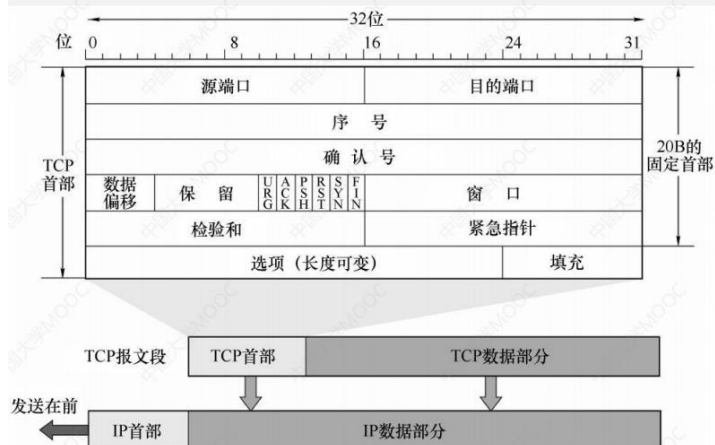


分用时，找不到对应的目的端口号，就丢弃报文，并给发送方发送 ICMP “端口不可达” 差错报告报文。

UDP 报文段分为首部和数据两部分，头部长度为 8B。

1. 源端口号：告知主机该报文段来自哪里。
2. 目的端口号：记录目标端口号，这在终点交付报文时必须使用到。
3. UDP 长度：记录 UDP 数据报长度（包括首部和数据）。
4. UDP 校验和：检测 UDP 数据报在传输中是否有错，有错就丢弃。

TCP报文段首部格式



序号：在一个TCP连接中传送的字节流中的每一个字节都按顺序编号，本字段表示本报文段所发送数据的**第一个字节的序号**。

确认号：期望收到对方下一个报文段的第一个数据字节的序号。若确认号为N，则证明到序号N-1为止的所有数据都已正确收到。

数据偏移（首部长度）：TCP报文段的数据起始处距离TCP报文段的起始处有多远，以4B位单位，即1个数值是4B。

1.序号：表示本报文段所发送数据的第一个字节的序号。

2.确认号：期望收到对方下一个报文段的第一个数据字节的序号。若确认号为N，则证明序号N-1为止所有数据都已经正确收到。

3.数据偏移（首部长度）

4.URG 紧急位：URG=1 证明此报文紧急，是高优先级数据，应尽快传送。

5.确认号 ACK：ACK=1 在连接建立后，所有传送的报文段都必须把 ACK 置为 1；

6.同步位 SYN：SYN=1 时，表明是一个连接请求/连接接受报文。

7.终止位 FIN：FIN=1，报文段数据已经发完，要求释放连接。

8.校验和：表明 TCP 报文是否有错。

9.源端口号

10.目的端口号

TCP报文段首部格式



6个控制位

紧急位URG：URG=1时，表明此报文段中有紧急数据，是高优先级的数据，应尽快传送，不用在缓存里排队，配合紧急指针字段使用。

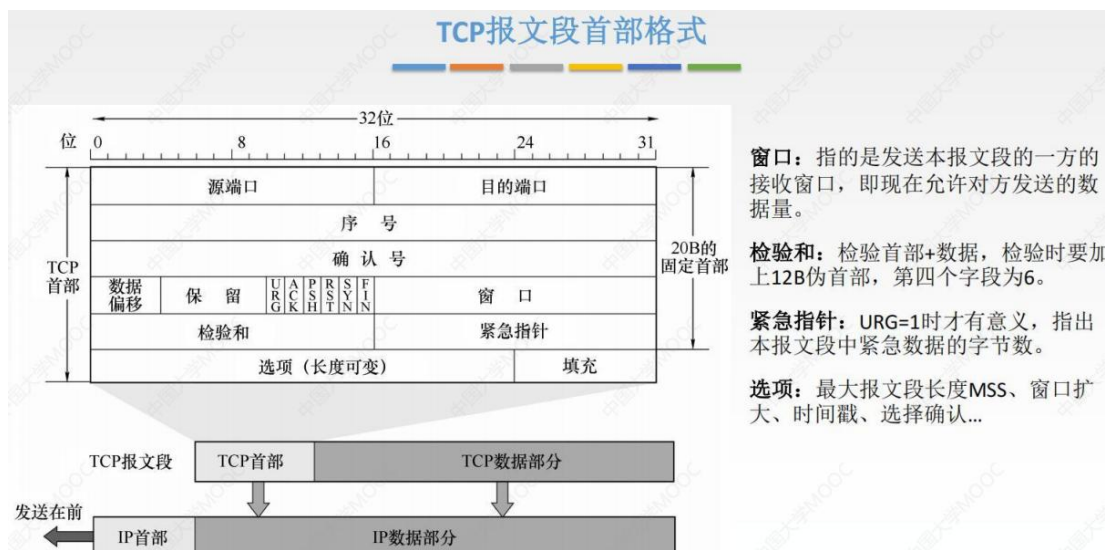
确认位ACK：ACK=1时确认号有效，在连接建立后所有传送的报文段都必须把ACK置为1。

推送位PSH：PSH=1时，接收方尽快交付接收应用进程，不再等到缓存填满再向上交付。

复位RST：RST=1时，表明TCP连接中出现严重差错，必须释放连接，然后再重新建立传输链接。

同步位SYN：SYN=1时，表明是一个连接请求/连接接受报文。

终止位FIN：FIN=1时，表明此报文段发送方数据已发完，要求释放连接。



16.TCP 拥塞控制，慢开始、拥塞避免、快重传、快恢复（4）

流量控制：让发送放慢点，要让接收方来得及接受。TCP 利用滑动窗口机制实现流量控制。

1.慢开始算法（接收窗口 $rwnd$ ，拥塞窗口 $cwnd$ ）（慢启动阶段）

在开始时，将拥塞窗口（ $cwnd$ ）初始值设为 $1MSS$ （ MSS 是最大报文段长度），等接受方确认数据之后，拥塞窗口成倍增加，直到窗口大小等于满开始门限值时，改用拥塞避免算法。

2.拥塞避免算法（拥塞避免阶段）

发送端的拥塞窗口 $cwnd$ 每经过一个往返时延 RTT 就增加一个 MSS 的大小，而不是加倍，使 $cwnd$ 按线性规律缓慢增加（即加法增大）。

3.拥塞处理

每当出现一次超时（网络拥塞）时，令慢开始门限 $ssthresh$ 等于当前拥塞窗口的一半（即乘法减小）。然后把拥塞窗口设置为 1，重新执行慢开始算法。

4.快重传

当发送方连续收到三个重复的 ACK 报文时，直接重传对方尚未收到的报文段，而不必等待那个报文段设置的重传计时器超时。（快重传并非取消重传计时器，而是在某些情况下可更早地重传丢失的报文段）

5.快恢复

发送端收到三个冗余 ACK （即重复确认）时，把慢开始门限设置为出现拥塞时发送方拥塞窗口的一半。与慢开始的不同之处是，它把拥塞窗口的值设置为慢开始门限 $ssthresh$ 改变后的数值，然后开始执行拥塞避免算法使拥塞窗口缓慢地线性增大。

17.TCP 建立连接的过程，如何保证可靠传输？（6）

1.TCP 采用三次握手建立连接。

2.检验和：通过检验和的方式，接收端可以检测出数据是否有差错，假如有就会直接丢弃并重新发送。

3.ACK 确认号：发送端发送一个包，但接收端没有回应确认包（ ACK 包），则重传，可以保

证数据的完整性。

4.超时重传: 发送方发送一个包，但是在规定时间内没有收到他的确认包，则被认为是丢包了，进行重传。

5.拥塞控制: 解决两台主机之间因传送速率而可能引起的丢包问题，保证了 TCP 数据传送的可靠性。

6.流量控制: 根据接收端的处理能力，来决定发送端的速度。

18.TCP 快速重传机制（4）

重传

确认重传不分家，TCP的发送方在规定的时间内没有收到确认就要重传已发送的报文段。**超时重传**

重传时间

TCP采用自适应算法，动态改变重传时间RTTs（加权平均往返时间）。

等太久了!!!

冗余ACK（冗余确认）

每当比期望序号大的失序报文段到达时，发送一个**冗余ACK**，指明下一个期待字节的序号。

发送方已发送1，2，3，4，5报文段

接收方收到1，返回给1的确认（确认号为2的第一个字节）

接收方收到3，仍返回给1的确认（确认号为2的第一个字节）

接收方收到4，仍返回给1的确认（确认号为2的第一个字节）

接收方收到5，仍返回给1的确认（确认号为2的第一个字节）

发送方收到3个对于报文段1的冗余ACK → 认为2报文段丢失，重传2号报文段 **快速重传**

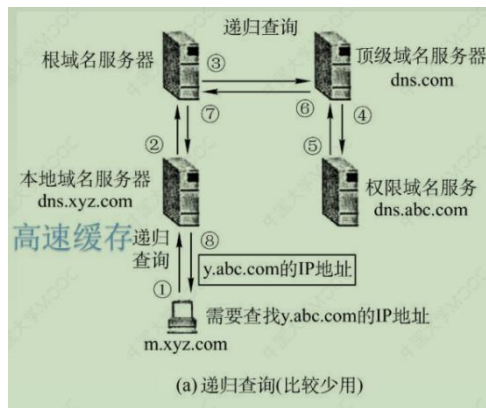
当发送方连续收到 **3个重复的 ACK 报文** 时（其实是 4 个同样的 ACK，第一个正常，后三个才是冗余），直接重传对方尚未收到的报文段，而不必等待那个报文段设置的重传计时器超时（快重传**并非取消重传计时器**，而是在某些情况下可更早地重传丢失的报文段）。

19.DNS 域名解析（5）

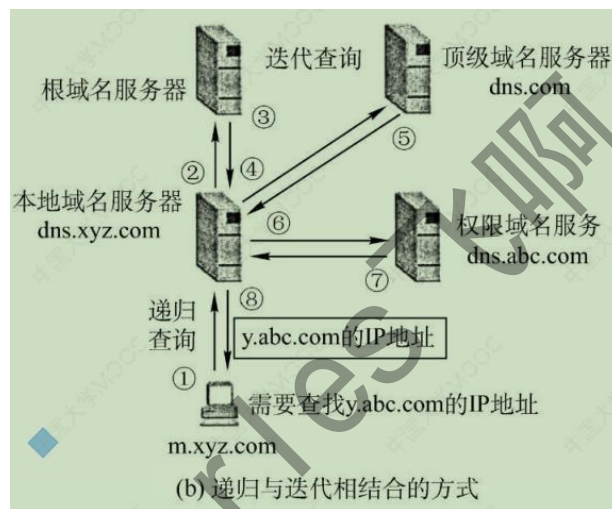
域名：www.baidu.com（网站）

1.作用: 把域名解析成为 IP 地址。

2.流程: a.递归查询（比较少用）：当主机发送 DNS 查询请求时，首先查询本地 DNS server，如果本地 DNS server 缓存中查询到域名对应的 IP 地址，则直接返回。否则，本地 DNS server 会继续查询根 DNS server，如果在根 DNS server 缓冲中查到，会返回给本地 DNS server，然后在返回给主机。如果在根 DNS server 没有查到，依次访问顶级 DNS server 和权限 DNS server，中间过程相同。



b.递归与迭代相结合方式：当主机发出 DNS 请求时，会查询本地 DNS server，如果在缓冲区中查到对应 IP 地址，则直接返回给主机。如果没有查到，本地 DNS server 会继续查询根 DNS server，



20.点击页面一次 HTTP 过程，输入一个网址，发什么？（3）

- 1.根据 URL 开始 DNS 域名解析。
- 2.发起 TCP3 次握手，建立连接。
- 3.建立 TCP 连接后发起 http 请求。
- 4.服务器响应请求，返回结果。（http 状态码）
- 5.浏览器得到 html 标签代码。
- 6.浏览器解析 html 代码的资源，例如 js，css，img 等。
- 7.浏览器对页面进行渲染并呈现给用户。

21.HTTP 和 HTTPS（4）

1.HTTP（超文本传输协议）含义：定义了浏览器（万维网客户进程）怎样向万维网服务器发出请求，以及服务器怎样把文档传给浏览器。

当浏览器访问一个网页时，浏览者的浏览器会向网页所在服务器发出请求。此网页所在的服务器会返回一个包含 **HTTP 状态码**的信息头，用以响应浏览器请求。

2.流程: a.用户输入 URL 或者点击超链接; b.浏览器向 DNS 请求解析 IP 地址; c.DNS 解析出 IP 地址; d.浏览器与服务器建立 TCP 三次握手连接; e.浏览器发出取文件命令; f.服务器响应。g.释放 TCP 连接; h.浏览器显示;

3.特点: a. http 协议是无状态的; b. http 采用 TCP 作为运输层协议, 但是 http 协议本身是无连接的 (通信双方在交换 http 报文之前不需要建立 http 连接)

4.http 连接方式: a.持久连接; b.非持久连接。

5.http 报文: a.请求报文; b.响应报文。

5.常见的 HTTP 状态码:

A.200—请求成功;

B.301—资源 (网页等) 被永久转移到其他 URL;

C.404—请求的资源 (网页) 不存在;

D.500—内部服务器错误;

E.503—服务器超时;

6.HTTPS 含义: 在 HTTP 协议基础上, 使用 SSL (secure sockets layer) 协议用于对 HTTP 协议传输的数据进行加密。

7.区别:

A.HTTP 是明文传输, 数据未加密, 安全性较差。HTTPS 数据传输过程是加密的, 安全性好。

B.使用 HTTPS 协议需要到 CA (数字证书认证机构) 申请证书, 一般免费证书较少。

C.HTTP 页面响应速度比 HTTPS 快, 主要因为 HTTP 使用 TCP 三次握手建立连接, 客户端和服务端需要交换 3 个包, 而 HTTPS 除了 TCP 三个包, 还要加上 SSL 握手的 9 个包, 所以一共是 12 个包。

D.HTTP 端口是 80, HTTPS 端口是 443。

E.HTTPS 其实就是构建在 SSL/TLS 之上的 HTTP 协议, 所以 HTTPS 比 HTTP 更耗费服务器资源。

22.网卡 (3)

1.概念: 网卡是一块被设计用来允许计算机在计算机网络上进行通讯的计算机硬件。由于其拥有唯一的 MAC 地址, 因此属于 OSI 模型的物理层和数据链路层。

(MAC 地址用来唯一标识一个网卡)

2.作用:

A.数据的封装与解封, 发送时将上一层传递的数据加上首部和尾部, 封装成帧。

B.链路管理, 通过 CSMA/CD 协议来实现。

C.数据编码与译码。

23.socket (4)

Socket 是对 TCP/IP 协议的封装, 它本身并不是协议, 而是一组用来调用 TCP/IP 网络 API 函数的接口。它的出现只是使得程序员更方便地使用 TCP/IP 协议而已。

24.cookie 和 session (3)

Cookie: 是存储在用户主机的文本文件, 记录一段时间内某用户 (使用识别码识别用户) 的

访问记录。

Cookie 作用：提供个性化服务（eg：登录淘宝，出现 hi, Charles!）

区别：

- 1.cookie 存储在客户端，session 存储在服务器。
 - 2.session 机制更安全，因为 cookie 存放在客户端，容易被窃取。
 - 3.session 会在一定时间内保存在服务器上。当访问增多，会比较占用你服务器的性能。
 - 4.cookie 存储了有限；session 可以无限量地往里面添加内容。
- Eg：经常访问网页，会让你删除 cookie。
- 5.将登录信息等重要信息存放为 session，其他信息如果需要保留，可以放在 cookie 中。

25.集线器、路由器、交换机

	路由器	交换机
工作层次	网络层	数据链路层
转发依据	IP地址	Mac地址
功能	连接不同的网络	连接局域网中的电脑
宽带影响	共享宽带	独享宽带

	交换机	集线器
工作层次	数据链路层	物理层
宽带影响	独享	共享
数据传输	有目的发送	广播发送
传输模式	全双工或半双工	半双工

26.WWW 万维网

- 1.万维网：是一个大规模的、联机式的信息存储所，是无数个网络站点和网页的集合。
- 2.统一资源定位符 URL，唯一标识资源（文字、视频、图片）。
- 3.URL 形式：<协议 http、ftp>://<主机：域名、IP 地址>:<端口>/<路径>
eg: <http://www.baidu.com>
- 4.这些资源通过超文本传输协议（http）传送给使用者。
- 5.万维网使用超文本标记语言 HTML，使万维网页面设计者可以很方便从一个界面链接到另一个界面，并能够在自己屏幕显示。

27.IP 地址分段

28.单播、多/组播

1.单播: 信息的接收和传递只在两个结点之间进行（点对点通信）。Eg: 两个人之间谈话, 一个人对于另一个人说话。常见的单播: 发电子邮件;

2.多播: 信息的一次传送可以到达多个目标结点。一般是通过 D 类 IP 地址来实现, 即 224.0.0.0 至 239.255.255.255 之间 IP 地址。

29.IPV4、IPV6

1.IPV4: 4B, 共 32 位; IPV6: 16B, 共 128 位。

2.IPV6 的优势:

A.IPV6 有更大的地址空间: IPV4 最大地址数 2^{32} , IPV6 最大地址数 2^{128} ;

B.IPV6 使用更小的路由表;

C.IPV6 增强了组播的支持, 提高了网络服务质量。

D.IPV6 加入了自动配置, 这也是对 DHCP 协议（动态主机配置协议）的改进和扩展。

E.IPV6 安全性更高。

30.C/S、B/S

1.C/S: Client/Server 即“客户端/服务器”模式, 一般建立在专用网络、小范围里的网络、局域网, 常见应用: 手机软件, 微信、qq。

2.B/S: Browser/Server 即“浏览器/服务器”模式, 一般建立在广域网上, eg: 网站上网。

31.P2P

1.peer-to-peer: 对等网络, 每个参与者都具有同等的能力, 可以发起一个通信会话。

2.来源: 在单一服务器向大量主机（对等方）分发大文件, 由于服务器必须向每个客户机发送该文件的拷贝, 这给服务器造成极大的负担。在 P2P 文件分发中, 每个对等方都能重新分发其所有的该文件的任何部分, 从而协助服务器分发。

2.优点: 数据的一致性容易控制, 系统也容易管理。

3.应用: 文件分发软件、语音服务软件。

32.曼彻斯特编码、差分曼彻斯特编码

1.曼彻斯特编码:

33.MAC 地址

又称**物理地址**，用于在网络中**唯一标识一个网卡**，**每个网卡都有一个全球唯一的 MAC 地址**。MAC 地址是**48 位的（6 个字节）**，**前 3 个字节表示网络硬件制造商的编号**，**后 3 个字节代表该制造商所制造的某个网络产品（如网卡）的系列号**。

34.PPP 点对点协议

Point-to-point protocol 点对点协议：为两个对等节点之间的 IP 流量传输提供一种封装协议

数据结构

1.栈和队列的区别

- 二者都是**线性结构**，**栈采用先进后出**，**队列采用先进先出**。
- 对插入和删除操作的“限定”不同：**a.栈是限定只能在一端进行插入和删除的线性表**。**b.队列限定在一端进行插入和另一端进行删除的线性表**。
- 用途不同：**栈主要用于递归、后缀表达式、函数调用、倒序**。**队列主要用于树的层次遍历、图的广度遍历、缓冲区**。

2.顺序结构和链式结构的区别？

- 链式结构的内存地址不需要连续**，**顺序结构的内存地址需要连续**。
- 链式结构必须顺序存取**，**顺序结构可以随机存取也可以顺序存取**。
- 链式结构占的内存空间更大**，**因为需要空间存储下一地址**。
- 链式结构更加适合频繁地插入删除更新数据**。
- 顺序结构存储空间可以静态分配，也可以动态分配**。**链表的存储空间是动态分配**。

3.二叉树和度为二的树的区别

- 度为 2 的树是不区分左子树和右子树**，**而二叉树是分左子树和右子树的**。
- 度为 2 的树不包含空树且树的最大结点的度为 2**，**而二叉树是可以有空树的**。

4.二叉树概念和性质

- 所有结点中最大的度就是树的度**。
- 树的层次是来自树的深度**。
- 二叉树：每个结点最多有两个子树，其结点有左右子树之分，可以为空树**。通常包含：**满**

二叉树、完全二叉树、平衡二叉树、二叉搜索树（二叉排序树）。

4. **满二叉树**：除了最后一层外，其他结点都有两棵子树。二叉树所有分支结点的度数都为 2，并且叶子结点都在同一层次上。从形状上看，满二叉树像一个三角形。从数学上看，首项为 1，公比为 2 的等比数列。

5. **完全二叉树**：除了最后一层外，其他任何一层的结点数都达到了 2，且最后一层只在右侧缺少结点。从根节点到倒数第二层满足满二叉树，最后一层可以不完全填充，其叶子结点都靠左对齐。满二叉树一定是完全二叉树，完全二叉树不一定是满二叉树。

6. **二叉排序树**：它可以是一颗空树，若它的左子树不空，则左子树上所有结点的值均小于它的根结点的值；若它的右子树不空，则右子树上所有结点的值均大于它根结点的值；它的左、右子树也分别为二叉排序树。

7. **平衡二叉树**：在二叉排序树的基础上，只要保证每个结点左子树和右子树的高度差小于等于 1。

8. 已知前序遍历和中序遍历，中序遍历和后序遍历，可以唯一确定一颗二叉树。

性质：

1. 二叉树第 i 层最多有 $2^{(i-1)}$ 个结点；
2. 深度为 k 的二叉树至多有 $2^k - 1$ 个结点；
3. $n_0 = n_2 + 1$ ；
4. 具有 n 个结点的完全二叉树深度为 $\log_2 n + 1$ ；

5. 二叉树的顺序存储和二叉链表

完全二叉树使用顺序存储是最好的，不会浪费存储空间；

对于一般的二叉树用二叉链表，结点结构 `lchild + data + rchild`；

6. 最优二叉树（哈夫曼树）

1. **定义**：给定 N 个权值作为 N 个叶子结点，构造一棵二叉树，该树的带权路径长度达到最小。权值较大的结点离根越近。

2. **具体做法**：每次寻找最小两个结点，将他们值相加，为最新的结点。

3. **哈夫曼编码**：针对字符集和字符出现的频率，构建哈夫曼树。规定哈夫曼树左分支代表 0，右分支代表 1，形成根到叶子结点对应编码。

7. 图的概念

1. **有向图**：顶点、弧（弧头、弧尾）；**无向图**：顶点、边；

2. **稀疏图、稠密图**：边的数量

3. **完全图、有向完全图**：任意两个顶点之间都存在边。

4. **度**：无向图顶点的边数；**入度和出度**：有向图顶点边。

5. **网**：图上边的权值。

6. **连通图（无向）、强连通图（有向）**：任意两个顶点都是连通的。

7. **连通分量（无向）、强连通分量（有向）**：图中，极大连通图。

8. **生成树**：无向图、连通、 n 个结点、 $n-1$ 条边。

9.图的存储：邻接矩阵、邻接表、十字链表、邻阶多重表

10.图的遍历：深度优先遍历、广度优先遍历

8.哈希表（散列表）

1.定义：根据关键码值 key 而直接进行访问数据 value 的结构。

2.装填因子： $a=n/m$ 其中 n 为关键字个数， m 为表长。

3.填充因子是表示 Hash 表中元素的填满的程度。若：加载因子越大，填满的元素越多，好处是，空间利用率高了，但：冲突的机会加大了。反之，加载因子越小，填满的元素越少，好处是：冲突的机会减小了，但：空间浪费多了。

4.解决冲突：线性探测法、平方探测法、拉链法、伪随机序列法

9.稳定的排序算法

1.定义：两个相同数字排序前后相对位置保持不变。

2.稳定排序：冒泡排序、基数排序、插入排序、归并排序。

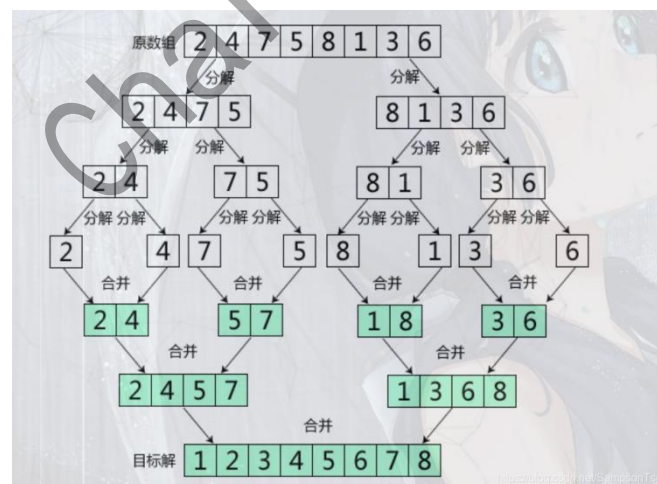
3.不稳定排序：选择排序、希尔排序、堆排序、快速排序。

10.二路归并算法、快速排序算法

1.相同：都借鉴了分治法思想。

2.区别：快速排序：只有“分”的过程，在“分”的过程中完成序列排序工作。

二路归并排序：具有“分”与“合”两个过程，真正的排序在“合”的过程中完成。



11.时间复杂度、空间复杂度

1.时间复杂度：用来衡量程序运行时间。如果某一算法的时间复杂度为 $O(n^2)$ ，表明该算法的运行时间与 n^2 这个数量级成正比。 n 表示问题规模，比如要遍历一个 $n*n$ 个矩阵所有元素，时间复杂度就是 $O(n^2)$ 。

2.空间复杂度：算法所消耗的存储空间，用来存放指令、常数、变量。

12.头指针、头节点区别

不管带不带头节点，**头指针**始终指向链表的第一个结点，而**头结点**是链表的第一个结点，结点通常不存信息，它通常是为了方便的一种操作。

引入头结点的优点：

1. 带头结点，**在表头插入删除更方便**。
2. 无论链表是否为空，其头指针都是指向头结点的非空指针，因此**空表和非空表的处理也得到了统一**。

13.判断链表是否有环

“**快慢指针**”：创建两个 fast 和 slow，开始两个指针都指向表头 head，在每一步操作中 slow 向前一步，fast 向前两步，由于 fast 比 slow 移动快，**如果有环那么两个指针就会相遇**。

14.循环队列

来源：为了解决“**假溢出**”问题。

问题：**队满和队空都为 $rear == front$** ，方法：**入队时少用一个存储单元**。

判断：

队空： $rear == front$

队满： $(rear + 1) \% maxsize == front$

入队： $rear = (rear + 1) \% maxsize$

出队： $front = (front + 1) \% maxsize$

队列长度： $(rear - front + maxsize) \% maxsize$

15.KMP 算法（字符串匹配算法）

- 1.思想：利用匹配失败后的信息，尽量介绍子串与主串的匹配次数以达到最快速匹配的目的。
- 2.关键：求出 next 数组， $next[j]$ 表示：在字串的第 j 个字符与主串发生失败匹配时，则跳到字串指针应该移动到的位置。
- 3.求 next 数组的方法：在不匹配的位置前，画出一个分界线，模式串一步步右移，指导分界线之前能匹配或者模式串完全跨过分界线为止。此时子串指针 j 指向哪儿，next 数组的值就是多少。
- 4.匹配方法：匹配过程产生失败时，主串指针 i 不变，子串指针 j 退回到 $next[j]$ 的位置，重新进行比较。
- 5.时间复杂度：求 next 数组 $O(m)$ ，模式匹配 $O(n)$ ，总 $O(n+m)$ ，主要优点是主串不回溯。

16.BFS

- 1.思想：类似于二叉树的**层次遍历**，是一种分层的查找过程，**先访问起始顶点 v，再由 v 出**

发，依次访问 v 未被访问过的邻接顶点 w_1, w_2, \dots ，然后依次访问 w_1, w_2 的未被访问过的顶点，重复这个过程，直到所有顶点都被访问。

2.实现：借用 `queue`，先把初始顶点入队，然后每次访问一个顶点，就要把此顶点出队，把未访问过邻接顶点依次入队，直到队空。

3.时间复杂度：邻接表 $O(v+e)$ ，邻接矩阵 $O(v^2)$ 。

17.DFS

1.思想：类似于树的先序遍历，首先访问 v 结点，然后从 v 出发，访问与 v 邻接但未被访问的顶点 w_1 ，然后再访问与 w_1 邻接但未被访问的顶点 w_2 ，重复此过程。当不能再往下访问时，依次退出到最近访问的结点，若还有其他顶点未被访问，则从该点开始继续上述搜索流程，直到所有顶点都被访问。

2.实现：递归、栈。

3.时间复杂度：邻接表 $O(v+e)$ ，邻接矩阵 $O(v^2)$ 。

18.最小生成树

包含图中全部顶点的极小连通子图。

19.Prim 算法（最小生成树）

1.思路：从某一顶点开始构建生成树，每次将代价最小的新顶点纳入生成树，直到所有顶点都纳入为止。

2.方法：两个数组，一个用来存储各顶点加入生成树的最小代价，一个用来存储各顶点是否已经加入了最小生成树。每次归并一个新顶点前，要遍历这两个数组找到还未加入树且权值最小的顶点；归并一个新顶点后要更新这两个数组。

3.时间复杂度： $O(n^2)$ 。

20.Kruskal 算法（最小生成树）

1.思路：每次选择一条权值最小的边，且这条边所连的两个顶点间还未存在路径，我们使这条边两头连通，重复上述操作直到所有结点都连通。

2.方法：将图中边按照权值从小到大排列，然后从最小的边开始扫描，设置一个边的集合来记录，如果该边并未构成回路，则将该边并入当前生成树。直到所有边都检测完为止。

3.时间复杂度： $O(E \log E)$ 。

21.红黑树（RBT）

1.作用：为了保持 AVL 树的平衡性，在进行插入和删除操作后，需要频繁地调整全数的整体拓扑结构，代价较大。因此在 AVL 树的平衡标准上进一步放宽条件，形成红黑树。对于红黑树来说，插入删除操作很多时候不会破坏其红黑树特性，无需频繁调整树的形态，即使要调

整也可以在常数级时间内完成。

2.定义：红黑树满足如下红黑性质的二叉排序树。

A.每个结点使红色或者黑色

B.根、叶结点是黑色

C.不存在两个相邻的红结点

D.对于每个结点，从该结点到任一叶结点的简单路径上，所含黑节点数量相同。

3.插入：先进行查找操作，确定插入位置并插入。如果新节点是根，则染为黑色，若是新节点非根，则染为红色。如果插入以后不满足红黑树定义，则需要调整使其重新满足红黑树定义。

22.B 树和 B+树（未完成）

1.B 树：是一种多路搜索树（并不一定是二叉的），是一种平衡多叉树。

2.m 阶 B 树性质：

a.根结点至少有两个结点。

b.所有叶子结点位于同一层。

1.在 B+树中，具有 n 个关键字的结点只含有 n 棵子树，即每个关键字对应一棵子树。在 B 树中 n 个关键字的结点含有 $n+1$ 棵子树。

2.在 B+树中，叶结点包含了全部关键字，即在非叶结点中出现的关键字也会出现在叶结点中。而在 B 树中，叶结点包含的关键字与其他结点包含的关键字是不重复的。

3.在 B+树中，非叶结点仅起索引作用，叶结点包含关键字对应记录的存储地址，而 B 树的结点都包含关键字对应记录的存储地址。

23.堆

1.堆是一种特殊的完全二叉树。

2.大根堆：每个父结点的值都大于孩子结点。

3.小根堆：每个父结点的值都小于子结点。

4.堆排序思路（逆序）：首先将元素构成初始大根堆，输出堆顶元素，并把堆底元素送入堆顶。此时根结点已经不满足大根堆的性质，堆被破坏，将堆顶元素向下调整使其继续满足大根堆的性质，再输出堆顶元素，重复上述步骤，直至堆中仅剩一个元素为止。

5.插入：先将新结点放在堆的末端，再将新结点向上调整。

6.删除：被删除元素用堆底元素代替，然后将该元素不断向下调整。

24.排序

	排序方法	平均情况	最好情况	最坏情况	辅助空间	稳定性
插入排序	直接插入排序	$O(n^2)$	$O(n)$	$O(n^2)$	$O(1)$	稳定
	折半插入排序	$O(n^2)$				稳定
	希尔排序	$O(n \log_2 n) \sim O(n^2)$	$O(n^{1.3})$	$O(n^2)$	$O(1)$	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	$O(n \log_2 n)$	$O(n \log_2 n)$	$O(n^2)$	$O(\log_2 n) \sim O(n)$	不稳定
选择排序	简单选择排序	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	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
	归并排序	$O(n \log_2 n)$	$O(n \log_2 n)$	$O(n \log_2 n)$	$O(n)$	稳定
	基数排序	$O(d(n+r))$			$O(r)$	稳定

25.线索化

- 1.解决二叉树遍历中，访问某一结点的前驱和后继。
- 2.方法：通常利用二叉树中左右子树的空指针来存放结点的前驱和后继结点。 n 个结点的二叉树中含有 $n+1$ 个空指针域。
 - A.如果一个结点有左孩子，那么 lchild 指向左孩子，否则指向前驱；
 - B.如果一个结点有右孩子，那么 rchild 指向右孩子，否则指向后继；
 - C.LTag=0，则指向左孩子；LTag=1，则指向前驱；
 - D.RTag=0，则指向右孩子；RTag=1，则指向后继；

操作系统

1.并行、并发

- 1.并行：指两个或多个事件在同一时刻发生，即同时做不同事的能力。
- 2.并发：指两个或多个事件在同一时间间隔发生，即交替做不同事的能力，多线程是并发的一种形式，这些事件在宏观上是同时完成的，在微观上的交替发生完成的。

2.线程、进程

- 1.线程：线程是处理机调度的单元，它被包含在进程之中，是进程中的实际运作单元。线程根据实现方式分为用户级线程、内核级线程，内核级线程才是处理机分配的单元。在同一个进程的各个线程共享进程拥有的资源，同一个进程内的线程切换不会导致进程切换。线程提升了程序的并发性。

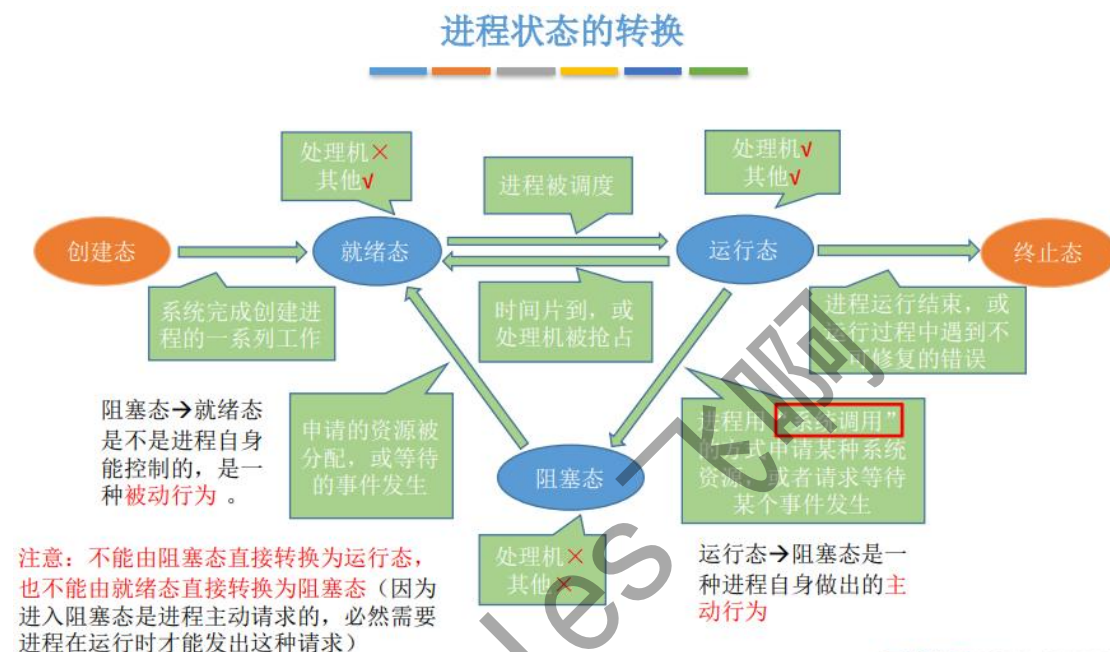
2.进程：有代码段、数据段、PCB 组成，其中 PCB 是进程存在的唯一标志。进程是操作系统资源分配的最小单元。进程具有动态性、并发性、独立性、异步性、结构性特征。

3.区别：

a.线程是处理机调度的单元，而进程是操作系统分配资源的最小单元。

b.一个程序至少一个进程，一个进程至少一个线程。

c.进程之间相互独立，但是同一个进程下的各个线程之间共享程序的内存（包括代码段、数据集、堆等）及一些进程级的资源（如打开文件和信号），某进程内的线程对其他进程不可见。



3.死锁

1.产生原因：死锁是由两个或多个进程相互等待对方手里的资源，导致各个进程都阻塞，无法向前推进；资源分配不当；系统资源不足；

2.产生死锁必要条件：a.互斥条件；b.资源不可剥夺；c.请求和保持条件；d.循环等待条件。

3.处理方法：a.死锁预防；b.死锁避免；c.死锁检测；d.死锁解除。



4.总线

计算机各个功能部件之间传递信息的公共通信干线，它是由导线组成。按照传输的信息种类，总线可以划分为数据总线、地址总线、控制总线，分别用来传递数据、地址和控制信号。

5.分段、分页

- a.段是信息的逻辑单位，它是根据用户的需要划分的，因此段对用户是可见的；
页是信息的物理单位，是为了管理主存而划分的，对用户是透明的。
- b.段的大小不固定，由它所完成的功能决定；eg：一个函数就是一个段。
页的大小固定，由系统决定段向用户提供二维地址空间；页向用户提供的是一维地址空间
- c.段是信息的逻辑单位，便于存储保护和信息的共享，页的保护和共享受到限制。

6.页面替换算法

- 1.最佳置换算法（OPT）：优先淘汰最长时间不会被访问的页面。但无法实现。
- 2.先进先出置换算法（FIFO）：优先淘汰最先进入内存的页面。实现简单，可能会出现 Belady 异常。
- 3.最近最久未使用置换算法（LRU）：优先淘汰最近最久没有访问的页面。性能好，但需要硬件支持。
- 4.CLOCK（NRU）算法

7.进程（作业）调度算法

- 1.先来先服务（FCFS）：按照作业到达后备队列的先后顺序选择进程（作业）。非抢占式，对短作业不利。
- 2.短作业优先（SJF）：选择作业运行时间最小的作业先进入主存。分抢占式和非抢占式，对长作业不利。
- 3.高响应比优先（HRRN）：按照 $(\text{已等待时间} + \text{运行时间}) / \text{运行时间}$ ，选择作业运行。是对以上算法综合权衡的算法。
- 4.时间片轮转算法（RR）：为进程分配时间片，当时间片运行结束后，进入就绪队列，等待下一个时间片。
- 5.优先权调度算法
- 6.多级反馈队列算法

8.操作系统特征

- 1.并发：只两个或多个事件在同一时间间隔内发生。这些事件宏观上是同时发生的，但微观上是交替发生的，多线程是并发的一种形式，
- 2.虚拟：指把一个物理上的实体变为若干个逻辑上的对应物。物理实体（前者）是实际存在

的，而逻辑上对应物（后者）是用户感受到的；eg：把一台物理 CPU 变成多台逻辑上独立的 CPU。

3.共享：资源共享，指系统中的资源可供内存中多个并发执行的进程共同使用。同时共享（磁盘）和互斥共享（打印机）。

4.异步（不确定）：多道程序下，程序并发执行，但是由于资源有限，进程的执行不是一贯到底，而是走走停停，以不可预知的速度向前推进。

9.中断

1.定义：CPU 中止正在执行的程序，转去处理随机提出的请求（中断源），待处理完后，再回到原先被打断的程序（断点）继续恢复执行的过程称为中断。

2.作用：为实现多道程序并发执行而引入的一种技术。发生中断时，CPU 会立即进入核心态。中断也是 CPU 从用户态进入内核态唯一途径。

3.分类：内中断（硬件故障、软件中断）、外中断（外设请求、人工干预）。

10.进程互斥、同步

1.互斥：指的是多个进程之间由于竞争临界资源而相互制约。

2.临界资源（互斥共享）：就是指一次仅允许一个进程使用的资源，即不能同时被共享的资源。

3.同步：指多个进程中发生的事件存在某种时序关系，需要相互合作，共同完成一项任务。

4.进程互斥 4 个部分：进入区、临界区、退出区、剩余区。

5.原则：a. 空闲让进；b. 忙则等待；c. 有限等待；d. 让权等待。

11.内核态和用户态

1.用户态可以执行 cpu 调用的非特权指令

2.内核态可以执行特权指令和非特权指令

3.用户态到内核态的切换是通过中断实现的

4.内核态到用户态的切换是通过修改（PSW）实现的

12.进程间通信方式

1.共享存储：设置一个共享空间，各个进程互斥访问。

2.管道通信：设置一个特殊的共享文件（缓冲区），各个进程互斥访问。写满时，不能再写。读空时，不能再读。没写满，不能读。没读空，不能写。

3.消息传递：a.直接通信方式：直接发送到对方消息队列中。b.间接（信箱）通信方式：消息发送到中间体（信箱）。

13.内存中堆、栈

1. **栈**的申请是由系统自动调度的，所以申请效率较快；**堆**的申请是由人为申请，效率没栈快。
2. **栈**的空间由系统开辟，空间较小。**堆**的空间较大。
3. **栈**是一级缓存，调用完立即释放；**堆**是二级缓存，生命周期由虚拟机的垃圾回收算法决定。

14.磁盘调度算法

1. **先来先服务算法（FCFS）**：按照访问请求到达的先后顺序处理。
2. **最短寻道时间有限（SSTF）**：贪心算法思想，每次都优先相应距离磁头最近的磁道。可能会导致饥饿。
3. **扫描算法（电梯算法 SCAN）**：只有磁头移动到最边缘的磁道时，才改变磁头的方向。
4. **循环扫描算法（C-SCAN）**：磁头移动到最边缘磁道时，立即让磁头返回原点，返回中途不做响应。

15.逻辑地址、物理地址

1. **逻辑地址**：程序自身看到的地址，它是一个抽象的地址，需要映射到物理地址，因为程序无法知道可用的物理地址，所以必须做出映射。
2. **逻辑地址与物理地址映射**：（分页），把逻辑内存分为多个页，把物理地址分为多个帧，通过页表来把逻辑地址和物理地址联系起来。

16.分页怎么优化

从时间上优化可以使用快表；从空间上优化可以采用多级页表。

17.哲学家进餐有那些实现方式

1. 共 5 个哲学家，至多允许 4 个哲学家同时去拿左边的筷子，然后在允许拿右边的筷子，最终能保障至少一个哲学家能进餐
2. 仅当哲学家的左、右两只筷子都可用时，才允许他去拿起筷子进餐。
3. 规定奇数号的哲学家先拿起它左边的筷子，然后再去拿它右边的筷子；偶数号的哲学家先拿起它右边的筷子，然后再去拿它左边的筷子。

18.虚拟存储器

1. 常规存储管理方式有两个特点：**一次性**和**驻留性**。**一次性**是指作业必须一次性的全部装入内存方能开始运行。**驻留性**：作业被装入内存后，整个作业一直驻留在内存直到运行结束。
2. **问题**：但有的作业很大不能一次性全部装入内存，因此要从逻辑上扩充内存容量，引入**虚拟技术**。

3.实现: 将当前需要的少数页面或是段先装入内存, 其余部分驻留在外存。程序运行过程中, 若要访问的页或段未调入内存, 则发出**缺页中断请求**, OS 利用**请求调页**功能将他们调入内存, 同时利用**置换功能**将不用的页换出内存。

4.概念: 虚拟存储器是主存的扩展, 虚拟存储器的空间大小取决于计算机的访存能力, 而不是实际外存的大小。

5.分类: 页式虚拟存储器、段式虚拟存储器、段页式虚拟存储器。

19.并行处理

1.同一时刻或同一时间内完成两种或两种以上性质相同或不相同的工作, 只要在时间上互相重叠, 都存在并行性。

2.常见的并行处理技术: a.时间并行; b.空间并行; c.时间并行+空间并行;

20.作业、进程

作业是用户需要计算机完成某项任务而要求计算机所做工作的集合。

进程是已提交完毕的作业的执行过程, 是资源分配的基本单位。

一个作业可由多个进程组成, 且必须至少由一个进程组成, 但一个进程不能构成多个作业

计算机组成原理

1.内存、外存

1.处理速度: 内存快, 外存慢。

2.存储容量: 内存小, 外存大。

3.价格上: 内存价格高, 外存价格低。

4.断电后: 内存 RAM 中的信息丢失, 外存中的信息不丢失。

5.本质区别: 内存为内部运行提供缓存和处理的功能, 也可以理解为协同处理的通道;

外存主要是储存文件、图片、视频、文字等信息的载体, 也可以理解为储存空间。

6.内存是计算机中重要的部件之一, 它是与 CPU 进行沟通的桥梁。计算机中所有程序的运行都是在内存中进行的, 因此内存的性能对计算机的影响非常大。

以下作为参考:

1.内存:

a.易失性

b.内存包括 RAM 中的缓存和主内存。它正式包括存储器和辅助存储器。

c.与 CPU 非常接近的高性能数据; SRAM 比 DRAM 更贵; DRAM 比外存更贵。

d.可升级的; 与外部存储介质相比, 价格昂贵。

e.存储 CPU 指令:使用频繁重复的指令进行缓存以提高效率, 主要用于将 CPU 指令与其他计

算机设备和组件进行通信。

2.外存:

a.非易失性

b.尽管外存也是一种存储类型，但它与缓存和主内存不同，因为它是非易失性的。

c.速度较慢，但能够以更低成本获得更高的容量

e.可升级的：HDD 成本在广泛可接受的范围内，而 SSD 的价格正在逐年降低，与 HDD 十分接近。

f.可存储数据，直到预定的数据被移动或删除。没有电源的硬盘和磁带将无限期地保存数据。无电源 SSD 可以保留数据长达两年，但实际上这段时间要短得多

2.RAM、ROM

1.RAM (Random Access Memory)：是可读写操作的，一般用于存储临时信息，易失性，断电后数据会消失。

a. **DRAM (动态 Dynamic RAM)**：主要用于主存，破坏性读出，需要刷新，运行速度慢，集成度高，储存成本低，断电后信息消失。

b. **SRAM (静态 static RAM)**：主要用于 cache，非破坏性读出，不需要刷新，运行速度快，集成度低，存储成本高，断电后信息消失。

2.ROM 是只读存储器，一般用于存储长期信息，非易失性，断电后数据不会丢失

a. **MROM (Mask Read-Only Memory)**

b. **PROM (Programmable Read-Only Memory)**

c. **EPROM (Erasable Programmable Read-Only Memory)**

d. **SSD (Solid State Drives) 固态硬盘**

3.指令

计算机指令一般由操作码和地址码组成，操作码主要是用来指明指令的操作类型，地址码用来标识操作数存储位置的字段。

4.寻址方式

1.立即寻址：操作数直接由地址码给出。

2.直接寻址：指令的地址码给出操作数在存储器中的地址。

3.隐含寻址：不是明显地给出操作数地址，而是在指令中隐含着操作数的地址（ACC）。

4.间接寻址：指令的地址码给出操作数在内存中地址的地址。

5.寄存器寻址：在指令中直接给出操作数所在寄存器编号。

6.寄存器间接寻址：寄存器给出操作数所在主存单元的地址。

7.基址寻址：

8.变址寻址：

9.相对寻址：

5.Cache 和主存之间的映射

1.全相联映射：任何主存地址可以映射到任何 Cache 地址的方式。在这种方式下，主存中存储单元的数据可以调入到 Cache 中的任意位置，只有在 Cache 中的块全部装满后才会发生冲突。

2.直接映射：将主存地址映射到 Cache 中的一个指定地址。任何时候，主存中存储单元的数据只能调入到 Cache 的一个位置，这个位置是固定的，若这个位置已有数据，则产生冲突，原来的块将无条件被替换。

3.组相联映射：将存储空间的页面分为若干组，各组之间直接映射，而组内各块之间则是全相联映射。

6.虚拟存储器

1.概念：虚拟存储器是主存的扩展，虚拟存储器的空间大小取决于计算机的访存能力，而不是实际外存的大小。

2.分类：页式虚拟存储器、段式虚拟存储器、段页式虚拟存储器。

7.冯诺依曼机器

1.计算机由**运算器、控制器、存储器、输入和输出设备**组成。

2.指令和数据**存储在存储器**中，并**按地址访问**。

3.指令和数据以**二进制表示**。

4.指令由**操作码和地址码**构成，**操作码**指明指令的操作类型，**地址码**表示操作数在存储器中的位置。

5.指令在存储器中**按顺序存放**，通常按照自动的顺序执行。

6.机器以**运算器为中心**。

8.CISC 和 RISC

1.CISC 的指令能力强，但多数指令使用率低，指令长度可变，RISC 的指令能力弱，指令长度固定（32 位宽），操作寄存器，只允许两类指令访问存储器，分别是 load 和 store

2.CISC 支持多种寻址方式；RISC 支持方式少

3.CISC 通过微程序控制技术实现；RISC 增加了通用寄存器，硬布线逻辑控制为主

4.CISC 的研制周期长

5.RISC 优化编译，有效支持高级语言

	CISC	RISC
指令系统	复杂,庞大	简单,精简
指令数	一般大于200	一般小于100
指令格式	一般大于4	一般小于4
指令字长	一般大于4	一般小于4
寻址方式	不固定	固定32位
可访问指令	不加限制	只有LOAD/STORE指令
各种指令使用频率	相差很大	相差不大
各种指令执行时间	相差很大	绝大多数在一个机器周期完成
优化编译实现	很难	较容易
程序源代码长度	较短	较长
控制逻辑实现方式	绝大多数为微程序控制	绝大多数为硬连线控制
RISC机的主要优点可归纳如下		
①充分利用VLSI芯片的面积		
②提高了计算机运行速度		
③便于设计,降低成本,提高可靠性		
④有效支持高级语言程序		

CISC: 指令集复杂, 指令长度不固定, 不便于优化, 一条指令可以完成整个功能 (eg: 加法: 取指、取数、执行、存储), 用到寄存器少, 用微程序控制。

RISC: 高频指令, 经常使用, 指令长度固定为 32bit, 容易优化, 一条指令只能完成固定功能 (eg: 加法), 用到寄存器多, 用硬布线控制。

9. MAR、MDR、存储器最大容量

1. MAR: 存储地址寄存器, 保存需要访问的存储单元地址。反映存储单元的个数。

2. MDR: 存储数据寄存器, 缓存读出/写入存储单元的数据。反映存储字长。

3. 存储器最大容量: 由 MAR 和 MDR 位数决定。

假设 MAR 寄存器的位数为 16 位, MDR 寄存器的位数为 16 位, 存储器的最大容量是多少?

- 1) MAR 寄存器的位数为 16 位, 能表示的地址个数为 2 的 16 次方, 为 64K;
- 2) MDR 寄存器的位数为 16 位, 说明存储字长为 16 位, 也即 2 个字节;
- 3) 存储器的最大容量为 $64K * 2B = 128K \text{ Byte}$

10. 机器字长、存储字长

1. 机器字长: CPU 一次能够处理的二进制数据的位数。机器字长通常与主存单位的位数一致。

2. 存储字长: 一个存储单元存储二进制代码的位数。按照某个地址访问某个存储单元获取的二进制数据的位数。

11. 奇偶校验码、汉明码

奇偶校验只能检错, 不能纠错

汉明码可以纠错。

12.补码表示真值范围

8 位: $-128 \sim +127$

-128: 1000,0000

13.IO 控制方式

1.程序直接控制方式（轮询）

2.中断驱动方式

3.DMA 方式

4.通道控制方式

14.原码、反码、补码

1.原码是由 0, 1 两个数字组成的二进制数

2.其最高位表示符号, ‘1’ 表示负号, ‘0’ 表示正号。

3.反码是为了解决原码的一个问题（两个相反数相加不为零）

4.所以正数的反码还是原码, 但是负数的反码就是原码除去符号位取反

5.补码是为了解决 0 这个特殊的数字存在（反码: 1111,1111 和 0000,0000 都表示 0）。

6.正数补码等于原码, 负数补码等于反码加一（补码: 1000,0000: -128, 0000,0000 表示 0）。

15.流水线技术

将指令执行重叠在一起大幅度提高计算机性能, 一种并行处理实现技术, 指令执行分为 5 个阶段: 取指, 译码, 读操作数, 执行, 保存操作数

16.衡量计算机性能指标

1.吞吐量: 表征一台计算机在某一段时间间隔能够处理的信息量, 单位 bit/s。

2.响应时间: 输入到系统产生响应之间的时间。

3.利用率: 在给定时间间隔内, 系统被实际使用的时间所占的比例。

4.处理机字长（处理字长）: 处理机一次能够完成二进制数运算的位数。

5.总线宽度: 一般指 CPU 中运算器和存储器之间进行互连的内部总线二进制位数。

6.存储器容量: 存储器中所有存储单元的总数目。KB、MB、GB、TB。

7.存储器带宽: 单位时间内从存储器读出的二进制数信息量。

8.主频/时钟频率: CPU 的工作节拍受主频控制单位 Hz。主频的倒数称为 CPU 时钟周期 (T), 单位 ms（微秒）。

9.CPU 执行时间: CPU 执行一段程序所花费时间。

10.MIPS: 每秒百万条数据。

17.指令周期

- 1.取指：**从 **PC（程序计数器）** 中找到对应指令地址，根据地址从内存中取出指令，加载到 **指令寄存器（IR）**，之后 **PC++**。
- 2.译码：**根据 **指令寄存器（IR）** 里面的指令，解析指令的操作。Eg: R、I、J。
- 3.执行：**具体执行相应的逻辑操作、数据传输或者地址跳转。

18.cache

- 1.cache** 是由 **SRAM（静态）** 组成，叫做 **高速缓冲存储器**，是介于 **CPU** 和主存之间的桥梁，用来缓冲高速 **CPU** 与低速内存速度不匹配的问题。
- 2.作用：****CPU** 速度远高于内存，cache 是用来保存 **CPU** 刚刚用过或循环使用的一部分数据，如果 **CPU** 再次使用，可以直接从 cache 中调用，这样就避免了重复存取数据，减少了 **CPU** 的等待时间，提高系统运行效率。
- 3.一级 cache、二级 cache 区别**
 - A.位置不同：**一级 cache 是一级缓存，在 **CPU** 内部；二级 cache 是二级缓存，位于 **CPU** 和内存之间。
 - B.读取数据的顺序不同：**一级 cache → 二级 cache → 内存。

19.如何判断整数数据溢出

20.微程序、微指令、微命令

- 1.概念不同：**
 - a.微程序：**是实现程序的一种手段，具体就是将一条 **机器指令** 编写成一段微程序。
 - b.微指令：**控制部件通过控制线向执行部件发出各种控制命令。
 - c.微命令：**在微指令的控制字段中，每一位代表一个微命令。
- 2.内容不同：**一个微程序包含若干微指令，在微指令的控制字段中，每一位代表一个微命令。

21.CPU 的组成部分

- 1.运算器、控制器（CU，Control Unit）、寄存器组和内部总线。**
- 2.运算器：** **算术逻辑单元 ALU**、通用寄存器、数据暂存器组成。
- 3.控制器：**
 - a.程序计数器（PC）：**指明下一个要执行指令的地址；
 - b.指令寄存器（IR）：**存放当前正在执行的指令；
 - c.指令译码器（ID）：**对指令进行“翻译”，确定执行什么操作。
 - d.控制电路：**根据指令译码器的分析，发出控制信号。
- 4.PSW（状态寄存器）：**保存程序当前执行的状态。

编译原理

1.编译过程的五个基本步骤

a.作用：实现输入高级语言源程序，输出目标语言代码

b.词法分析

词法分析器：通过词法分析程序对源程序的字符串从左到右的扫描，逐个读取字符，识别出每个单词符号（eg: if、else），然后以二元式形式输出。

c.语法分析

在词法分析的基础上，将识别出的单词符号序列组成各类语法（eg: $Z=X+Y$ ），如“语句”，“表达式”。判断源程序语句是否符合定义的语法规则。

d.语义分析

了解各个语法单位之间的关系是否合法。

e.中间代码生成与优化

在语法分析和语义分析之后，将源代码变成一种内部表示形式（中间代码），这种中间代码介于源程序和机器语言之间，很容易翻译成目标代码，同时也方便优化。

f.目标代码生成

根据优化有的中间代码，生成有效的目标代码，也就是汇编代码，之后还需要将汇编代码转为机器代码。

2.有限自动机（DFA）

又称为有穷状态的机器，它由一个有限的内部状态集合一组控制规则组成，这些规则是用来控制当前状态下输入应转向什么状态。

软件工程

1.设计模式

1.目的：为了可重用代码，提高代码的可扩展性和可维护性，降低代码耦合度，主要是基于OOP编程提炼。

2.原则：

a.开闭原则：软件应该对扩展开放，对修改关闭。在增加新功能时，能不改代码就尽量不改，如果只增加代码就能完成新功能，那是最好的。

b.里氏替换原则：如果我们调用一个父类的方法可以成功，那么替换成子类调用也应该可以完全运行。

2.黑盒、白盒测试

- 1.白盒测试：又称**结构测试**，他把测试对象（源代码）看成一个**透明**的盒子，允许测试人员利用程序**内部的逻辑结构**设计测试用例，对程序所有逻辑路径进行测试。
- 2.原则：a.保证一个模块中的**所有独立路径至少被测试一次**；b.所有逻辑判断均需测试**取真和取假**两种情况；c.在上下边界及可操作范围内允许所有循环；d.检测程序的内部数据结构，保证其结构的有效性。
- 3.方法：a.程序结构分析：根据源代码可以首先绘制程序的流程图，然后根据流程图分析程序的结构；b.逻辑覆盖测试：根据程序的内部结构，对所有路径进行测试，是一种穷举路径的测试方法；c.基本路径测试；
- 4.实施步骤：a.撰写测试计划；b.撰写测试用例；c.执行测试用例；d.撰写测试结果。
- 1.黑盒测试：又称**功能测试**，盒子里的源代码测试员不能看到，只能看到软件或者某些模块的简单功能描述，**主要是发现软件设计的需求或软件规格说明书中的错误缺陷**。
- 2.测试方法：a.等价类划分法；b.边界值分析法；c.因果图法；d.错误推测法；

3.软件危机

软件危机是指落后的软件生成方式无法满足迅速增长的计算机软件需求，从而导致软件开发与维护过程中出现一系列严重问题的现象。

表现：

- 1.软件开发进度难以预测。
- 2.拖延工期几个月甚至几年现象并不罕见。
- 3.软件开发成本难以控制。
- 4.用户对产品功能难以理解。
- 5.开发员和用户之间难以沟通，矛盾很难统一。
- 6.软件产品难以维护。

4.数据库完整性、数据完整性、实体完整性、参照完整性、用户定义完整性

- 1.数据库完整性：数据的正确性（数据时符合现实世界语义，反映当前实际状况）、数据的相容性（数据库同一对象在不同关系表中的数据时符合逻辑的）
- 2.数据完整性：防止数据库中存在不正确的数据。
- 3.实体完整性：保证关系中的每个元组都是可识别的和唯一的，指关系数据库中所有的表都必须有主键；不能出现：a.无主键值得记录；b.主键值相同的记录；
- 4.参照完整性（引用完整性）：主要用来描述实体之间的联系。
- 5.用户定义完整性（语义完整性、域完整性）：指明关系中属性的取值范围，防止属性的值与应用语义矛盾。

离散数学

1.笛卡尔集

笛卡尔集是集合的一种，假设 A 和 B 都是集合， A 和 B 的笛卡尔积用 $A*B$ 来表示，是所有有序偶 (a, b) 的集合，其中 a 属于 A ， b 属于 B 。

$A*B=\{(a, b) \mid a \text{ 属于 } A \text{ 且 } b \text{ 属于 } B\}$ ，则 $A*B$ 所形成的集合就叫笛卡尔集。

C++（真题）

1.++、--运算符重载

代码参考链接（最好去看一下，也不是很难，毕竟机试也可能会考）：

https://blog.csdn.net/weixin_44190648/article/details/122018049

1. $++i$ 不需要默认参数， $i++$ 需要默认参数；默认参数是用来判断是前++，还是后++。
2. 前++需要返回&引用，因为可以++++i；后++不需要返回&引用，因为不可以i++++。

2.为什么析构造函数要用虚函数

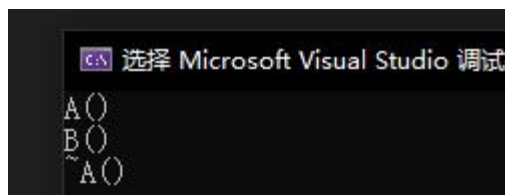
参考链接：https://blog.csdn.net/weixin_44190648/article/details/122121736

未使用虚析构造函数

```
class A {
public:
    A() { cout << "A()" << endl; }
    ~A() { cout << "~A()" << endl; }
};

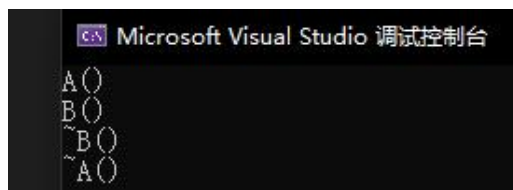
class B : public A {
public:
    B() { cout << "B()" << endl; }
    ~B() { cout << "~B()" << endl; }
};

int main() {
    A* a = new B();
    delete a;
}
```



使用虚析构函数:

```
virtual ~A() { cout << "~A()" << endl; }
```



3.浅构造、深构造

代码参考链接: https://blog.csdn.net/weixin_44190648/article/details/121946941

一般 C++ 在类中含有指针变量时, 需要深拷贝。

- 1.深拷贝是指, 自己手写一个拷贝函数, 拷贝函数中包含: a.给指针申请空间; b.赋值。
- 2.浅拷贝是指, 拷贝函数中, 没有给指针申请空间, 导致的内存访问错误。

4.全局变量和局部变量在内存中是否有区别?如果有, 是什么区别?

有区别。

全局变量保存在内存的全局存储区中, 占用静态的存储单元; **局部变量**保存在栈中, 只有在所在函数被调用时才动态地为变量分配存储单元

5.引用与指针有什么区别?

- 1.从语法规则上讲, 指针变量存储某个实例(变量或对象)的地址; 引用是某个实例的别名。
- 2.程序为指针变量分配内存区域; 而不为引用分配内存区域
- 3.解引用是指针使用时要在前面加 “ * ” ; 引用可以直接使用
- 4.指针变量的值可以发生变化, 存储不同实例的地址; 引用在定义时就被初始化, 之后无法改变(不能是其他实例的引用)
- 5.指针变量的值可以为空(NULL, nullptr); 但是没有空引用, 定义时就初始化
- 6.指针变量作为形参时需要测试它的合法性(判空 NULL, 因为有空指针, 野指针, 失效指针); 7.不需要判空, 可以直接使用
- 8.指针变量使用 “sizeof” 得到的是指针变量的大小; 对引用变量使用 “sizeof” 得到的是变量的大小
- 9.理论上指针的级数没有限制; 但是引用只有一级。即不存在引用的引用, 但可以有指针的指针

10. 用与++指针的效果不一样

Eg: 指针++: 指向下一地址。引用++: 数据+1;

程序设计（真题）

1. 什么是比较好的编码风格

1. 缩进良好，没有过多嵌套。
2. 变量名，空间名，类名都取有意义的字眼。（高中数学教科书上给出的求最小公倍数最大公约数的代码段太恶心了，尽是 i, s, j, k 之类的变量,,,）
3. 有一些必要注释。
4. 符合思维习惯，而不是为了追求字符的简洁而使用奇怪的运算符。
5. 模块化。

2. 如何不用循环的方式判断 2 的 n 次方（如何递归）

```
int mutiple(int n) {  
    if (n == 0)  
        return 1;  
    return 2 * mutiple(n - 1);  
}
```

3. 01 背包复杂度

4. 单淘汰相关算法

5. 简述弗洛伊德算法求最短路径或如何求最短路径

6. 在字符串中找出只出现一次的数字，其余数字出现两次

7. 排序的时间、空间复杂度。

8.字符串匹配算法

9.图书管理系统，图书用什么结构保存，b 树

10.简述二叉树左右子树逆置

11.迪杰斯特算法求最短路径

12.最小生成树算法

13.On 实现链表反转

14.有序数组建立平衡二叉树

15.如何判断无向图是树

16.二叉树节点相关性质

17.改进字符串匹配

其他（真题）

1.为什么报我们学校（英语口语里有）

华南大学是一所著名的文科类大学在中国，我相信这里知名的教授和勤奋的同学将帮助我很多在我未来的学习。我的大学是一种理工大学，主要的学科是与水利工程相关，如果我可以录取，我将遇到并且交很多其他专业的朋友，这能拓宽我的视野。

此外，我看过一些你们学校推广视频，大学悠久的历史和文化非常吸引我，他给我一种快乐和和平的感觉，这使我能沉下心做研究。

老实讲，华南大学是我梦想中的大学，从我是个孩子开始，所以我非常想让我的梦想成真。

2.你想读什么方向

3.你的研究生计划是什么（英语口语里有）

第一年，我计划主要学习专业课知识，然后尽可能读与专业相关的论文和书籍。参加一些专业论坛或者讲座，也参加一些哈佛或者耶鲁大学的网上公开课，去更多的学习我的专业领域，构建我的知识体系。

第二年，我计划定期读一些权威的学术研究报告，跟随导师做一些研究性的课题。然后，将我的学术研究论文发表在知名的期刊。

第三年，我计划完成我的毕业论文，毕业答辩，然后更多的探索这个领域未来的潜在可能。如果有机会，我还想读博继续深造一下。不过目前还是希望能被华师录取。

4.项目是什么规模的项目，用的什么语言，这个功能你怎么实现的

5.最近在看什么书

最近我在看 python 的一些教材，包括一些黑马地 python 的教学。因为我认为在未来的研

研究生生活中，python 应该是经常能用到的语言。

6.你的毕业论文做的什么，用了什么技术

我的毕业论文是《光谱图像多语义可视化分析软件》，主要是做一款手机软件，用来对图像进行分析处理。首先，我使用 Matlab，计算图像的相似性分析等功能。其次，我使用 Android studio 搭建界面，包括一些联网登录、身份验证、查询本地图库等功能，用户的身份信息都存储在 Bmob 后端云中。最后，我将 Matlab 代码的 jar 功能包集成在软件中。实现整个完整功能。

7.你还有什么想问我们的

- 1.假如未来我被录取了，那么我可以自由出入华师的其他校区吗？
- 2.什么时候开始调档案？

专业问题（真题）

1.线程、进程概念及其区别 （上面有）

- 1.线程：线程是 cpu 调度的基本单元，线程包含在进程中。线程又分为内核级线程、用户及线程。内核级线程才是 cpu 分配的基本单元。同一个进程内的线程共享进程的资源，且同一个进程内的线程切换不会引起进程切换，线程的并发的一种形式。
- 2.进程是计算机资源分配的基本单元，进程由代码段、数据的、pcb 组成，其中 pcb 是进程存在的唯一标志。进程具有动态性、并发性、结构性、独立性、异步性。
- 3.一个程序至少有一个进程，一个进程至少有一个线程。

2.为什么说操作系统是由中断驱动的（上面有）

3.递归栈会不会溢出，为什么

- 1.会
- 2.为什么：栈是一种先进后出的数据结构，当我们递归一个函数时，每一次的递归运行都会做压栈操作，系统中栈也是有最大的空间限制的，Linux 下用户默认的栈空间大小为 8MB，当栈的存放容量超出这个限制之后，通常我们的程序会得到栈溢出得到错误。

4.栈泄漏是什么，栈与堆的区别

- 1.栈泄露：栈的大小不是无限的。大量的方法调用过程，导致不断压栈最终将栈内存占满产

生 `stackOverflowError` 的错误。

Eg: 递归太深

2. 区别:

A. 栈的空间申请是由系统自动进行, 因此速度快, 一般空间较少; 堆的空间一般由用户申请, 因此速度较慢, 但是空间较大。

B. 栈是一级缓冲, 在程序运行结束后, 栈的空间自动释放; 堆是二级缓冲, 生命周期由虚拟机垃圾回收机制。

1. 十进制、八进制、二进制转换

1. 为什么要三次握手和四次挥手

1. 正交矩阵是什么

1. 是一个方块矩阵, 行向量和列向量都是正交的单位向量。

2. 正交: 任意两行 (列) 相乘为 0.

3. 单位: 任意一行 (列) 自身相乘为 1。

4. $AA^T = E$

1. 范式模型 (数据库) (上面有)

2. 蠕虫病毒怎么到我电脑上的

1. 蠕虫病毒: 是一种十分古老的计算机病毒, 他是一种 **自包含的程序** (或是一套程序), **通常通过网络传播**, 侵入到一台新的计算机, 然后在此计算机上复制自己, 并自动执行自身的程序, 占用 CPU, 导致电脑运行很卡, 上网也很卡。

Eg: 熊猫烧香

2. 解决方法: 安装 360 杀毒软件。

3. 编译器原理 (编译—>链接—>运行)

1. 编译器是 **将源代码转化为机器码** 的软件。

2. 编译过程:

a. 编译前段: 将源代码转化成中间代码。详细过程: 预处理、词法分析、语法分析、生成中间代码。

b. 中间代码优化: 对编译器生成的中间代码进行一些优化, 最终提供给编译后端。

c.编译后端：将中间代码汇编，产生汇编代码，最后解析汇编指令，生成机器码。

4.反射是什么，请用 c 语言实现一下

1.反射：是在运行的时候来自我检查，并对内部成员进行操作。

5.不用加减乘除算除法

生活问题（真题）

1.是否有读博打算？

读博基本上是两三年之后的事了，就目前而言，我不太确定将来还会不会适合去读博。但如果有机会的话，我还是很愿意去争取读博深造的，现在还是更希望老师可以给我一个读研的机会。

2.有没有提前联系导师

我的意向导师是 XX 导师，我通过贵校官网了解老师的研究成果，并且拜读了他的 XX 主题论文，收益良多。我认为 XX 老师的某些观点值得我去引申和思考，因此我希望有机会上岸能够跟老师深入交流，得到老师指导。

3.为什么读研？

在本科四年中，通过学习专业知识和阅读专业书籍，我发现我对这个专业非常感兴趣，并且想为这个领域做出一点贡献。同时，我意识到我现在所学的知识离这个目标还相差很远，所以我想读研并且继续深造。

4.本科期间有什么科研成果？

对于这个问题我表示很惭愧，说实话大学期间我非常普通，甚至没有拿过什么奖项，这也是我大学期间最大的遗憾。

同时也说明我不够优秀，我还需要更加努力，希望在研究生阶段能够提高这些方面的能力，争取在各大比赛和科研工作中有所收获。

5.你的本科成绩为什么不够出彩？

因为我大学期间都处于比较迷茫的状态，不仅对自己的理解不够深刻，而且目标也不够明确，没有好好学习，导致成绩一般。

直到工作以后我才认识到自己想要的未来是什么样子，才知道自己的专业是多么的有魅力，决定考研并付出努力，我不想自己的青春有遗憾。

事实也证明只要努力一切都不晚，一切都有可能。

6.某一科为什么这么好？

C++代码编写注意事项

```
class String
{
private:
    int len;
    char* data; // 指针要特别注意！ 要注意释放空间！
public:
    String(const char* s = ""); // class里面有默认参数，写函数时需要去掉默认参数
    String(const String& s);
    void Copy(const String& src);
    void Cat(const String& src);
    int Len() const { return len; };
    void Print() const { cout << data; }
    ~String() { delete data; } // 析构函数一定要释放指针
};
```

```
String::String(const char* s) // 不能写默认参数
{
    this->len = strlen(s);
    this->data = new char[this->len + 1];
    for (int i = 0; s[i] != '\0'; i++)
        this->data[i] = s[i];
    this->data[this->len] = '\0';
}

String::String(const String& s) {
    this->len = s.len;
    this->data = new char[this->len + 1];
    for (int i = 0; s.data[i] != '\0'; i++)
        this->data[i] = s.data[i];
    this->data[this->len] = '\0';
}

void String::Copy(const String& src) {
    delete this->data; // 释放之前的空间
    this->len = src.len;
    this->data = new char[this->len + 1];
    for (int i = 0; src.data[i] != '\0'; i++)
        this->data[i] = src.data[i];
    this->data[this->len] = '\0';
}
```

```
class SavingAccount
{
private:
    int no;
    double balance;
    static double rate; // 静态变量, 必须申明对应的静态函数
    static int totalNo;
public:
    SavingAccount(double deposit);
    void updateMonthly();
    void print() const;
    static void setRate(double); // 静态函数才能对静态变量操作
    static int generateNo();
};
```

```
double SavingAccount::rate = 0; // static 初始化在外
int SavingAccount::totalNo = 0;

SavingAccount::SavingAccount(double deposit) {
    this->balance = deposit;
    this->no = generateNo();
}

void SavingAccount::updateMonthly() {
    this->balance += this->balance * this->rate;
}

void SavingAccount::print() const {
    cout << this->no << "\t" << balance << endl;
}

void SavingAccount::setRate(double newRate) { // 静态函数, 此处不能写 static
    rate = newRate;
}

int SavingAccount::generateNo() {
    return totalNo++;
}
```

```

class Teacher
{
private:
    char className[30];
    char time[20];
    char place[20];
    int numOfStudent; // 学生人数
    int totalQuiz; // 总的小测验数
    int curQuiz; // 已进行的小测验数
    struct student {
        char name[10];
        int scoreMid; // 期中成绩
        int scoreFinal; // 期末成绩
        int* scoreQuiz; // 保存每次小测验成绩
    };
    student* sInfo;
public:
    enum Type { MID, FINAL, QUIZ };
    Teacher(char* cName, char* cTime, char* cPlace, int noS, int noQuiz);
    ~Teacher();
    void inputStudent();
    void inputScore(Type);
    void analysis(Type);
};

```

```

// 初始化列表赋值
Teacher::Teacher(char* cName, char* cTime, char* cPlace, int noS, int noQuiz) : numOfStudent(noS), totalQuiz(noQuiz)
{
    strcpy(this->className, cName); // char要记得str函数!!!
    strcpy(this->time, cTime);
    strcpy(this->place, cPlace);
    sInfo = new student[this->numOfStudent];
    for (int i = 0; i < this->numOfStudent; i++)
        sInfo[i].scoreQuiz = new int[totalQuiz];
}

```

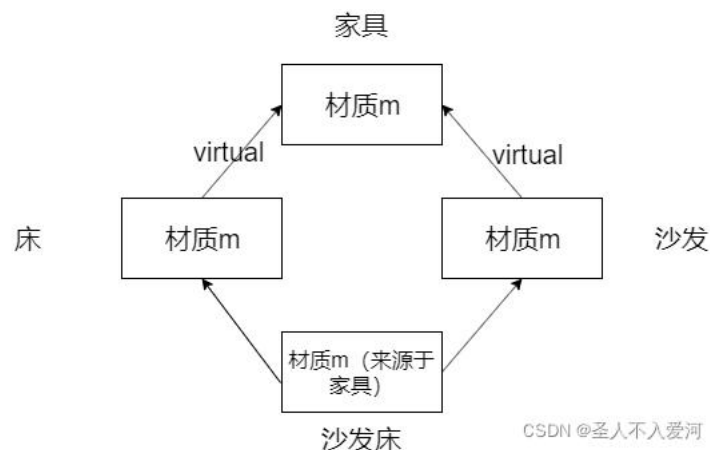
```

Teacher::~~Teacher() { // 注意析构函数
    for (int i = 0; i < this->numOfStudent; i++)
        delete sInfo[i].scoreQuiz;
    delete sInfo;
}

```

1. 虚继承

参考链接: https://blog.csdn.net/weixin_44190648/article/details/122108830



2.多态（虚函数）

参考链接: https://blog.csdn.net/weixin_44190648/article/details/122111016

3.虚析构函数、虚函数

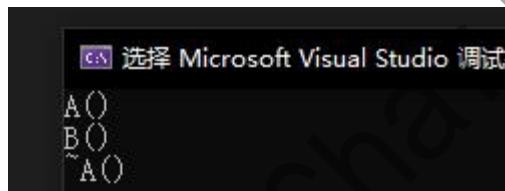
参考链接: https://blog.csdn.net/weixin_44190648/article/details/122121736

未使用虚析构函数

```
class A {
public:
    A() { cout << "A()" << endl; }
    ~A() { cout << "~A()" << endl; }
};

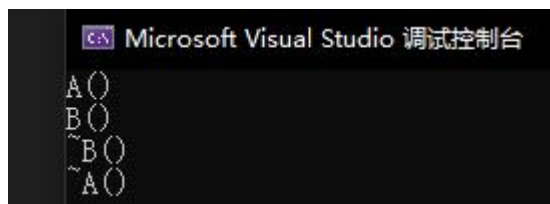
class B :public A {
public:
    B() { cout << "B()" << endl; }
    ~B() { cout << "~B()" << endl; }
};

int main() {
    A* a = new B();
    delete a;
}
```



使用虚析构函数:

```
virtual ~A() { cout << "~A()" << endl; }
```



4.纯虚函数、抽象类

参考代码: https://blog.csdn.net/weixin_44190648/article/details/122148318

5.Vector

参考代码: https://blog.csdn.net/weixin_44190648/article/details/122406761

6.操作符重载

参考代码: https://blog.csdn.net/weixin_44190648/article/details/122018049

Charles 飞阿阿