**百度研发类面试题（C++方向）**

## 1、在函数内定义一个字符数组，用gets函数输入字符串的时候，如果输入越界，为什么程序会崩溃？

答：因为gets无法截断数组越界部分，会将所有输入都写入内存，这样越界部分就可能覆盖其他内容，造成程序崩溃。

## 2、C++中引用与指针的区别

答：联系：引用是变量的别名，可以将引用看做操作受限的指针；

区别：

1） 指针是一个实体，而引用仅是个别名；

2）引用只能在定义时必须初始化，指针可以不初始化为空；

3）引用初始化之后其地址就不可改变（即始终作该变量的别名直至销毁，即从一而终。注意：并不表示引用的值不可变，因为只要所指向的变量值改变。引用的值也就改变了），但指针所指地址是不可变的；如下：

int m=23,n=13;

int& a=m;

a=12; //合法，相当于修改m=12

a=n;//合法，相当于修改m=13

## 3、C/C++程序的内存分区

答：**其实C和C++的内存分区还是有一定区别的，但此处不作区分：**

1）、栈区（stack）— 由编译器自动分配释放 ，存放函数的参数值，局部变量的值等。其   
操作方式类似于数据结构中的栈。   
2）、堆区（heap） — 一般由程序员分配释放， 若程序员不释放，程序结束时可能由OS回   
收 。注意它与数据结构中的堆是两回事，分配方式倒是类似于链表。   
3）、全局区（静态区）（static）—，全局变量和静态变量的存储是放在一块的，初始化的   
全局变量和静态变量在一块区域， 未初始化的全局变量和未初始化的静态变量在相邻的另   
一块区域。 - 程序结束后由系统释放。   
4）、文字常量区 —常量字符串就是放在这里的。 程序结束后由系统释放   
5）、程序代码区—存放函数体的二进制代码。

**栈区与堆区的区别：**

1）堆和栈中的存储内容：栈存局部变量、函数参数等。堆存储使用new、malloc申请的变量等；

2）申请方式：栈内存由系统分配，堆内存由自己申请；

3）申请后系统的响应：栈——只要栈的剩余空间大于所申请空间，系统将为程序提供内存，否则将报异常提示栈溢出。  
堆——首先应该知道操作系统有一个记录空闲内存地址的链表，当系统收到程序的申请时，会遍历该链表，寻找第一个空间大于所申请空间的堆结点，然后将该结点从空闲结点链表 中删除，并将该结点的空间分配给程序；

4）申请大小的限制：Windows下栈的大小一般是2M，堆的容量较大；

5）申请效率的比较：栈由系统自动分配，速度较快。堆使用new、malloc等分配，较慢；

总结：栈区优势在处理效率，堆区优势在于灵活；

**内存模型：自由区、静态区、动态区；**

根据c/c++对象生命周期不同，c/c++的内存模型有三种不同的内存区域，即：自由存储区，动态区、静态区。

自由存储区：局部非静态变量的存储区域，即平常所说的栈；

动态区： 用new ，malloc分配的内存，即平常所说的堆；

静态区：全局变量，静态变量，字符串常量存在的位置；

注：代码虽然占内存，但不属于c/c++内存模型的一部分；

## 4、快速排序的思想、时间复杂度、实现以及优化方法

答：**快速排序的三个步骤：**  
(1)选择基准：在待排序列中，按照某种方式挑出一个元素，作为 "基准"（pivot）；  
(2)分割操作：以该基准在序列中的实际位置，把序列分成两个子序列。此时，在基准左边的元素都比该基准小，在基准右边的元素都比基准大；  
(3)递归地对两个序列进行快速排序，直到序列为空或者只有一个元素。

**基准的选择：**

对于分治算法，当每次划分时，算法若都能分成两个等长的子序列时，那么分治算法效率会达到最大。

即：同一数组，时间复杂度最小的是每次选取的基准都可以将序列分为两个等长的；时间复杂度最大的是每次选择的基准都是当前序列的最大或最小元素；

**快排代码实现：**

我们一般选择序列的第一个作为基数，那么快排代码如下：

**[cpp]** [view plain](https://blog.csdn.net/Xiongchao99/article/details/74524807?locationNum=6&fps=1) [copy](https://blog.csdn.net/Xiongchao99/article/details/74524807?locationNum=6&fps=1)

1. **void** quicksort(vector<**int**> &v,**int** left, **int** right)
2. {
3. **if**(left < right)//false则递归结束
4. {
5. **int** key=v[left];//基数赋值
6. **int** low = left;
7. **int** high = right;
8. **while**(low < high)    //当low=high时，表示一轮分割结束
9. {
10. **while**(low < high && v[high] >= key)//v[low]为基数，从后向前与基数比较
11. {
12. high--;
13. }
14. swap(v[low],v[high]);
16. **while**(low < high && v[low] <= key)//v[high]为基数，从前向后与基数比较
17. {
18. low++;
19. }
20. swap(v[low],v[high]);
21. }
22. //分割后，对每一分段重复上述操作
23. quicksort(v,left,low-1);
24. quicksort(v,low+1,right);
25. }
26. }

注：上述数组或序列v必须是引用类型的形参，因为后续快排结果需要直接反映在原序列中；

**优化：**

上述快排的基数是序列的第一个元素，这样的**对于有序序列，快排时间复杂度会达到最差的o(n^2)**。所以，**优化方向就是合理的选择基数**。

常见的做法“三数取中”法（序列太短还要结合其他排序法，如插入排序、选择排序等），如下：

①当序列区间长度小于 7 时，采用插入排序；  
②当序列区间长度小于 40 时，将区间分成2段，得到左端点、右端点和中点，我们对这三个点取中数作为基数；  
③当序列区间大于等于 40 时，将区间分成 8 段，得到左三点、中三点和右三点，分别再得到左三点中的中数、中三点中的中数和右三点中的中数，再将得到的三个中数取中数，然后将该值作为基数。  
具体代码只是在上一份的代码中将“基数赋值”改为①②③对应的代码即可：

**[cpp]** [view plain](https://blog.csdn.net/Xiongchao99/article/details/74524807?locationNum=6&fps=1) [copy](https://blog.csdn.net/Xiongchao99/article/details/74524807?locationNum=6&fps=1)

1. **int** key=v[left];//基数赋值
2. **if**(right-left+1<=7){
3. insertion\_sort(v,left,right);//插入排序
4. **return**;
5. }**else** **if**(right-left+1<=8){
6. key=SelectPivotOfThree(v,left,right);//三个取中
7. }**else**{
8. //三组三个取中，再三个取中（使用4次SelectPivotOfThree，此处不具体展示）
9. }

需要调用的函数：

**[cpp]** [view plain](https://blog.csdn.net/Xiongchao99/article/details/74524807?locationNum=6&fps=1) [copy](https://blog.csdn.net/Xiongchao99/article/details/74524807?locationNum=6&fps=1)

1. **void** insertion\_sort(vector<**int**> &unsorted,**int** left, **int** right)  //插入排序算法
2. {
3. **for** (**int** i = left+1; i <= right; i++)
4. {
5. **if** (unsorted[i - 1] > unsorted[i])
6. {
7. **int** temp = unsorted[i];
8. **int** j = i;
9. **while** (j > left && unsorted[j - 1] > temp)
11. {
12. unsorted[j] = unsorted[j - 1];
13. j--;
14. }
15. unsorted[j] = temp;
16. }
17. }
18. }
20. **int** SelectPivotOfThree(vector<**int**> &arr,**int** low,**int** high)  //三数取中，同时将中值移到序列第一位
21. {
22. **int** mid = low + (high - low)/2;//计算数组中间的元素的下标
24. //使用三数取中法选择枢轴
25. **if** (arr[mid] > arr[high])//目标: arr[mid] <= arr[high]
26. {
27. swap(arr[mid],arr[high]);
28. }
29. **if** (arr[low] > arr[high])//目标: arr[low] <= arr[high]
30. {
31. swap(arr[low],arr[high]);
32. }
33. **if** (arr[mid] > arr[low]) //目标: arr[low] >= arr[mid]
34. {
35. swap(arr[mid],arr[low]);
36. }
37. //此时，arr[mid] <= arr[low] <= arr[high]
38. **return** arr[low];
39. //low的位置上保存这三个位置中间的值
40. //分割时可以直接使用low位置的元素作为枢轴，而不用改变分割函数了
41. }

这里需要注意的有两点：

①插入排序算法实现代码；

②三数取中函数不仅仅要实现取中，还要将中值移到最低位，从而保证原分割函数依然可用。

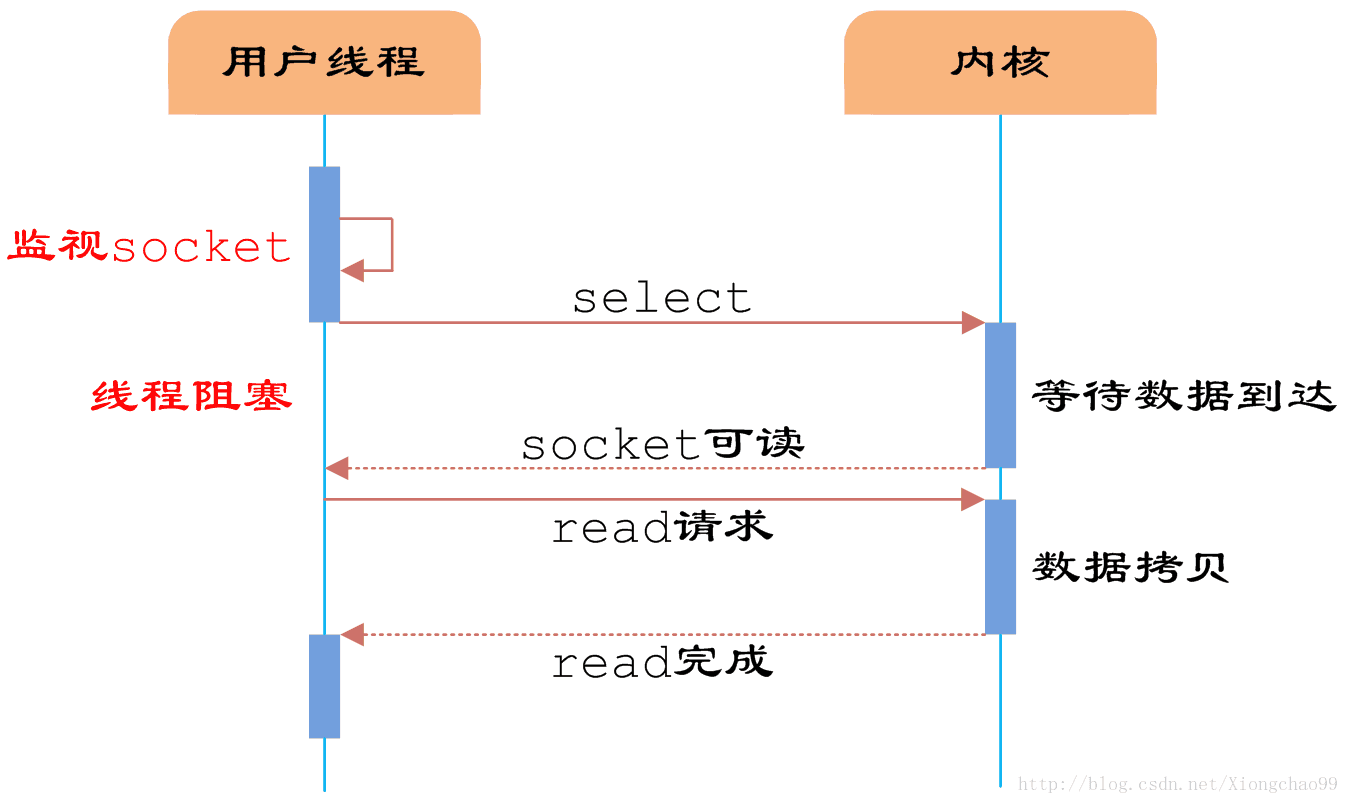
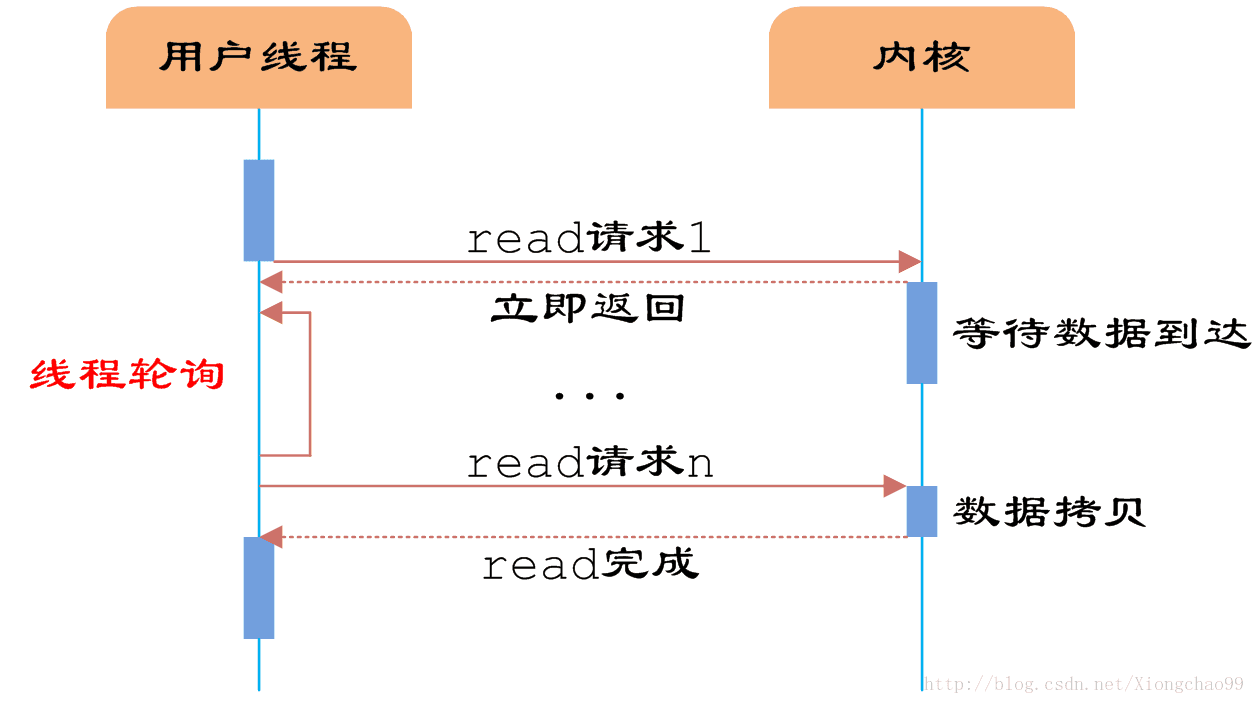
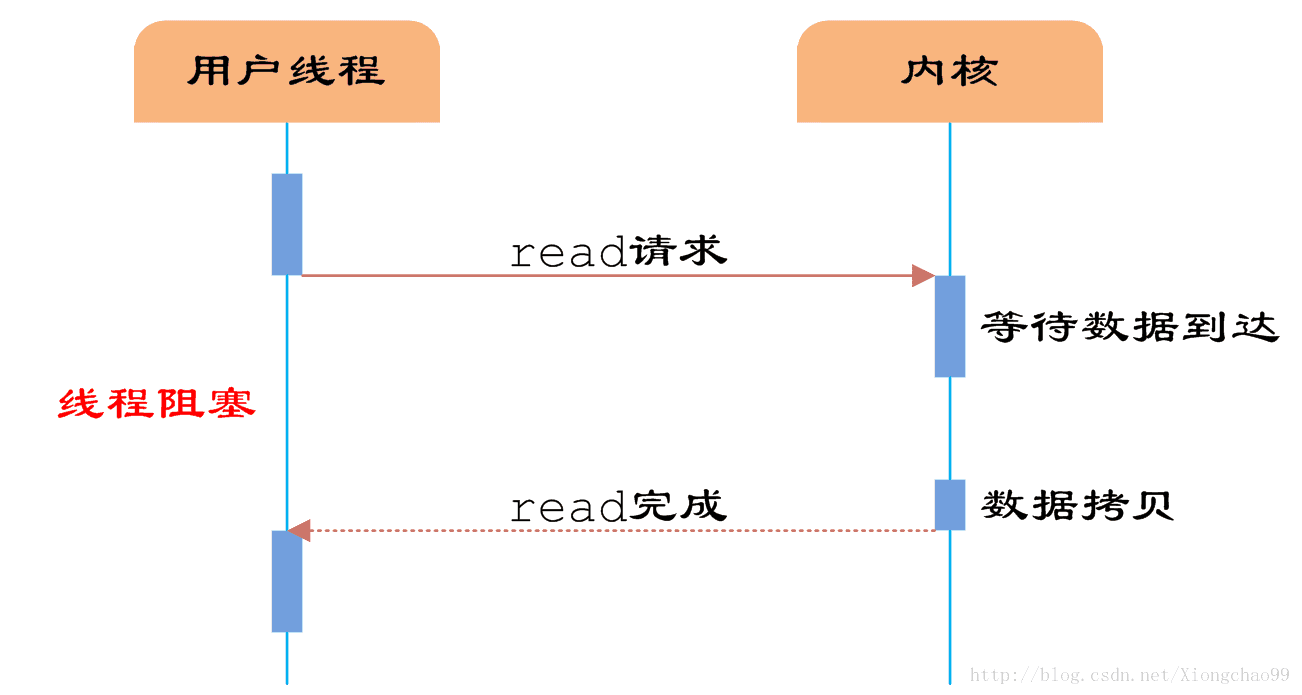
## 5、 IO模型——IO多路复用机制

答：预备知识介绍：

I**O模型有4中**：同步阻塞IO、同步非阻塞IO、异步阻塞IO、异步非阻塞IO；IO多路复用属于IO模型中的异步阻塞IO模型，在服务器高性能IO构建中常常用到。

上述几个模型原理如下图：

同步阻塞IO：                                                           同步非阻塞IO：                                            IO多路复用（异步阻塞IO）：



如上：同步异步是表示服务端的，阻塞非阻塞是表示用户端，所以可解释为什么IO多路复用（异步阻塞）常用于服务器端的原因；

**文件描述符（FD，又叫文件句柄）**：描述符就是一个数字，它指向内核中的一个结构体(文件路径，数据区等属性)。具体来源：Linux内核将所有外部设备都看作一个文件来操作，对文件的操作都会调用内核提供的系统命令，返回一个fd(文件描述符)。

下面开始介绍IO多路复用：

**（1）I/O多路复用技术通过把多个I/O的阻塞复用到同一个select、poll或epoll的阻塞上，从而使得系统在单线程的情况下可以同时处理多个客户端请求。**与传统的多线程/多进程模型比，I/O多路复用的最大优势是系统开销小，系统不需要创建新的额外进程或者线程。

**（2）select，poll，epoll本质上都是同步I/O，因为他们都需要在读写事件就绪后自己负责进行读写，也就是说这个读写过程是阻塞的，而异步I/O则无需自己负责进行读写，异步I/O的实现会负责把数据从内核拷贝到用户空间。**

**（3）I/O多路复用的主要应用场景如下：**

服务器需要同时处理多个处于监听状态或者多个连接状态的套接字；  
服务器需要同时处理多种网络协议的套接字；

**（4）目前支持I/O多路复用的系统调用有 select，poll，epoll**，epoll与select的原理比较类似，但**epoll作了很多重大改进，现总结如下：**

①支持一个进程打开的文件句柄FD个数不受限制（为什么select的句柄数量受限制：select使用位域的方式来传递关心的文件描述符，因为位域就有最大长度，在Linux下是1024，所以有数量限制）；

②I/O效率不会随着FD数目的增加而线性下降；

③epoll的API更加简单；

**（5）三种接口调用介绍：**

①select函数调用格式：

**[cpp]** [view plain](https://blog.csdn.net/Xiongchao99/article/details/74524807?locationNum=6&fps=1) [copy](https://blog.csdn.net/Xiongchao99/article/details/74524807?locationNum=6&fps=1)

1. #include <sys/select.h>
2. #include <sys/time.h>
3. **int** select(**int** maxfdp1,fd\_set \*readset,fd\_set \*writeset,fd\_set \*exceptset,**const** **struct** timeval \*timeout)
4. //返回值：就绪描述符的数目，超时返回0，出错返回-1

②poll函数调用格式：

**[cpp]** [view plain](https://blog.csdn.net/Xiongchao99/article/details/74524807?locationNum=6&fps=1) [copy](https://blog.csdn.net/Xiongchao99/article/details/74524807?locationNum=6&fps=1)

1. # include <poll.h>
2. **int** poll ( **struct** pollfd \* fds, unsigned **int** nfds, **int** timeout);

③epoll函数格式（操作过程包括三个函数）：

**[cpp]** [view plain](https://blog.csdn.net/Xiongchao99/article/details/74524807?locationNum=6&fps=1) [copy](https://blog.csdn.net/Xiongchao99/article/details/74524807?locationNum=6&fps=1)

1. #include <sys/epoll.h>
2. **int** epoll\_create(**int** size);
3. **int** epoll\_ctl(**int** epfd, **int** op, **int** fd, **struct** epoll\_event \*event);
4. **int** epoll\_wait(**int** epfd, **struct** epoll\_event \* events, **int** maxevents, **int** timeout);

**（6）作用：**一定程度上替代多线程/多进程，减少资源占用，保证系统运行的高效率；

更多细节待续……

## 6、常用的Linux命令

答：（1）查看CPU利用率：top

（2）查看当前目录：pwd和ls（ls -a可以查看隐藏目录）

（3）切换目录：cd

（4）查看文件占用磁盘大小：du和df

（5）创建文件夹：mkdir

（6）新建文件：touch

（7）查看文件：cat

（8）拷贝：cp  移动：mv  删除：rm

（9）查看进程：ps，如ps aux

（10）删除进程：kill -9 PID，注-9是参数

（11）程序运行时间：time，使用时在命令前添加time即可，如：time ./test，可得到三个时间：real 0m0.020s，user 0m0.000s，sys 0m0.018s

grep命令（重要的常用命令之一）：常用于打开文本修改保存，类似打windows开开TXT文本并修改；

sed命令（常用重要命令之一）：主要用于对文件的增删改查；

awk命令（重要常用命令之一）：取列是其擅长的；

find 命令（常与xargs命令配合）：查找 -type 文件类型-name 按名称查找-exec执行命令；

xargs命令：配合find/ls查找，将查找结果一条条的交给后续命令处理；

gdb调试工具：

要调试C/C++的程序，一般有如下几个步骤：

①首先在编译时，我们必须要把调试信息加到可执行文件中，编译生成可执行文件-------> g++  -g hello.cpp -o hello；

②启动GDB编译hello程序----------> gdb hello;

③显示源码------------> l;

④开始调试：break 16——设置断点在16行，break func——设置断点在函数func()入口处，info break——查看断点信息，n——单步运行，c——继续运行程序，r——运行程序；p i——打印i的值，finish——退出程序，q——退出gdb。

## 7、C中变量的存储类型有哪些？

答：c语言中的存储类型有auto, extern, register, static 四种；

## 8、动态规划的本质

答：动归，本质上是一种划分子问题的算法，站在任何一个子问题的处理上看，当前子问题的提出都要依据现有的类似结论，而当前问题的结论是后面问题求解的铺垫。任何DP都是基于存储的算法，核心是状态转移方程。

## 9、实践中如何优化MySQL

答：四条从效果上第一条影响最大，后面越来越小。

① SQL语句及索引的优化   
② 数据库表结构的优化   
③ 系统配置的优化   
④ 硬件的优化

## 10、 什么情况下设置了索引但无法使用

答：① LIKE语句，模糊匹配   
② OR语句  
③ 数据类型出现隐式转化（如varchar不加单引号的话可能会自动转换为int型）

## 11、 SQL语句的优化

答：alter尽量将多次合并为一次；

insert和delete也需要合并；

尽量使用union而不是or；

## 12.、数据库索引的底层实现原理和优化

答：B树，经过优化的B+树。主要是在所有的叶子结点中增加了指向下一个叶子节点的指针，因此InnoDB建议为大部分表使用默认自增的主键作为主索引。

## 13、HTTP和HTTPS的主要区别

答：见另一文章解析：<http://blog.csdn.net/xiongchao99/article/details/73381280#t8>

## 14、 如何设计一个高并发的系统

答：① 数据库的优化，包括合理的事务隔离级别、SQL语句优化、索引的优化；

② 使用缓存，尽量减少数据库 IO；

③ 分布式数据库、分布式缓存；

④ 服务器的负载均衡；

## 15. 两条相交的单向链表，如何求他们的第一个公共节点

答：思想：

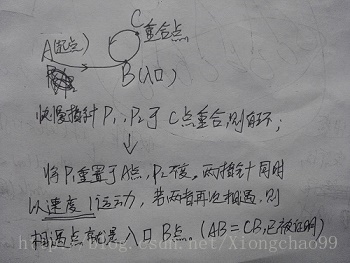
①如果两个链表相交，则从相交点开始，后面的节点都相同，即最后一个节点肯定相同；  
②从头到尾遍历两个链表，并记录链表长度，当二者的尾节点不同，则二者肯定不相交；  
③尾节点相同，如果A长为LA，B为LB，如果LA>LB,则A前LA-LB个先跳过；

——更多如链表相关经典问题：求单向局部循环链表的入、将两个有序链表合并合成一个有序链表、链表逆序、求倒数第K个节点，判断是否有环等。

## 16、****求单向局部循环链表的环入口****

答：思路：

假如有快慢指针判断一个链表有局部环，链表起点是A，环的入口是B，快慢指针在环中的相遇点是C。那么按照原来的运动方向，有AB=CB，这是可以证明的结论。具体如下图说明：



## 17、IP地址如何在数据库中存储

答：常有以下几种存储方式：



说明一下：int类型的num存储在解码时是这样做的：

65=num%256；num=num/256;

120=num%256；num=num/256;

……

## 18、new/delete和malloc/free的底层实现

答：**malloc和new的区别：**

1）malloc与free是C++/C语言的标准库**函数**，new/delete是C++的**运算符**。它们都可用于申请动态内存和释放内存；

2）new 返回指定类型的指针，并且可以自动计算所需要大小。而 malloc 则必须要由程序员计算字节数，并且在返回后强行转换为实际类型的指针；

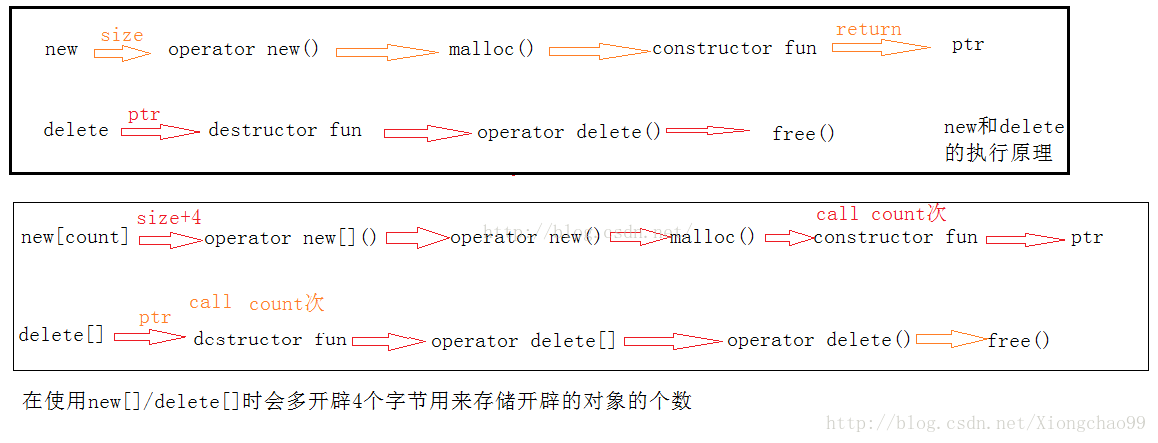
3）new/delete在对象创建的同时可以自动执行构造函数初始化，在对象在消亡之前会自动执行析构函数。而malloc 只管分配内存，并不能对所得的内存进行初始化，所以得到的一片新内存中，其值将是随机的；

既然new/delete的功能覆盖了malloc/free，为什么C++还要保留malloc/free？因为C++程序经常要调用C函数，而C程序只能用malloc/free管理动态内存。

**new/delete、malloc/free底层实现原理：**

**概述：new/delete的底层实现是调用malloc/free函数实现的，而malloc/free的底层实现也不是直接操作内存而是调用系统API实现的。**

new/delete的两种分配方式原理图如下：



**注意，针对上图最末尾所述的“new[]/delete[]时会多开辟4字节用于存储对象个数”，作如下说明：**  
①对于内置类型：   
new []不会在首地址前4个字节定义数组长度。   
delete 和 delete[]是一样的执行效果，都会删除整个数组，要删除的长度从new时即可知道。   
②对于自定义类型：   
new []会在首地址前4个字节定义数组长度。   
当delete[]时，会根据前4个字节所定义的长度来执行析构函数删除整个数组。   
如果只是delete数组首地址，只会删除第一个对象的值。

## 19、overload、override、overwrite的介绍

答：（1）**overload（重载），即函数重载：**  
①在同一个类中；  
②函数名字相同；  
③函数参数不同（类型不同、数量不同，两者满足其一即可）；  
④不以返回值类型不同作为函数重载的条件。  
**（2）override（覆盖，子类改写父类的虚函数），用于实现C++中多态：**  
①分别位于父类和子类中；  
②子类改写父类中的virtual方法；  
③与父类中的函数原型相同。  
**（3）overwrite（重写或叫隐藏，子类改写父类的非虚函数，从而屏蔽父类函数）：**  
①与overload类似，但是范围不同，是子类改写父类；  
②与override类似，但是父类中的方法不是虚函数。

## 20、小端/大端机器

答：小端/大端的区别是指低位数据存储在内存低位还是高位的区别。其中小端机器指：数据低位存储在内存地址低位，高位数据则在内存地址高位；大端机器正好相反。

当前绝大部分机器都是小端机器，就是比较符合人们逻辑思维的数据存储方式，比如intel的机器基本就都是小端机器。

## 21、守护进程

答：（1）什么是守护进程？  
守护进程（Daemon Process），也就是通常说的 Daemon 进程（精灵进程），是 Linux 中的后台服务进程。它是一个生存期较长的进程，通常独立于

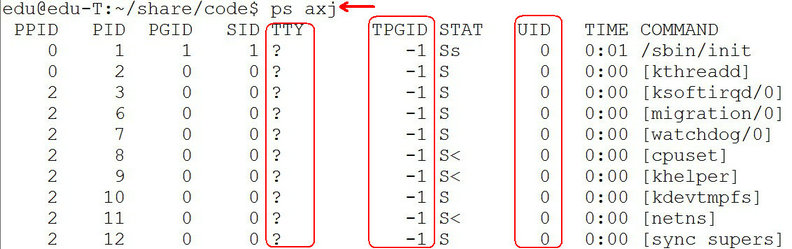
控制终端并且周期性地执行某种任务或等待处理某些发生的事件。

守护进程是个特殊的孤儿进程，这种进程脱离终端，为什么要脱离终端呢？之所以脱离于终端是为了避免进程被任何终端所产生的信息所打断，其在执

行过程中的信息也不在任何终端上显示。

（2）如何查看守护进程？

在终端敲：ps axj



从上图可以看出守护进行的一些特点：  
守护进程基本上都是以超级用户启动（ UID 为 0 ）  
没有控制终端（ TTY 为 ？）  
终端进程组 ID 为 -1 （ TPGID 表示终端进程组 ID）

更多守护进程相关参考：<http://blog.csdn.net/lianghe_work/article/details/47659889>

## 22、多线程

答：Java提供了3中多线程实现：Thread类、runable接口、使用ExecutorService、Callable、Future实现有返回结果的多线程。

而C++本身并没有提高多线程编程功能的库或接口，但Windows系统下的C++多线程编程还是可以通过<windows.h>库中的相关多线程接口实现，具体见：<http://blog.csdn.net/xiongchao99/article/details/64441017#t107>；

Linux写的C++多线程可以用头文件pthread.h，常用到其中两个函数pthread\_create和pthread\_join。下面是一个Linux下的简单C++多线程程序：

**[cpp]** [view plain](https://blog.csdn.net/Xiongchao99/article/details/74524807?locationNum=6&fps=1) [copy](https://blog.csdn.net/Xiongchao99/article/details/74524807?locationNum=6&fps=1)

1. //Threads.cpp
2. #include <iostream>
3. #include <unistd.h>
4. #include <pthread.h>
5. **using** **namespace** std;
7. **void** \***thread**(**void** \*ptr)
8. {
9. **for**(**int** i = 0;i < 3;i++) {
10. sleep(1);
11. cout << "This is a pthread." << endl;
12. }
13. **return** 0;
14. }
16. **int** main() {
17. pthread\_t id;
18. **int** ret = pthread\_create(&id, NULL, **thread**, NULL);//创建线程
19. **if**(ret) {
20. cout << "Create pthread error!" << endl;
21. **return** 1;
22. }
23. **for**(**int** i = 0;i < 3;i++) {
24. cout <<  "This is the main process." << endl;
25. sleep(1);
26. }
27. pthread\_join(id, NULL);//等待线程结束
28. **return** 0;
29. }

多线程相关的同步的知识不再累述，见<http://blog.csdn.net/xiongchao99/article/details/74858900>，此处来说说多线程优缺点：

**多线程的主要优点包括:**

(1)多线程技术使程序的响应速度更快 ,因为用户界面可以在进行其它工作的同时一直处于活动状态；

(2)占用大量处理时间的任务使用多线程可以提高CPU利用率，即占用大量处理时间的任务可以定期将处理器时间让给其它任务；

(3)多线程可以分别设置优先级以优化性能。

**以下是最适合采用多线程处理：**

(1)耗时或大量占用处理器的任务阻塞用户界面操作;

(2)各个任务必须等待外部资源 (如远程文件或 Internet连接)。  
  
**多线程的主要缺点包括：**

(1)等候使用共享资源时造成程序的运行速度变慢。这些共享资源主要是独占性的资源 ,如打印机等。

(2)对线程进行管理要求额外的 CPU开销，线程的使用会给系统带来上下文切换的额外负担。

(3)线程的死锁。即对共享资源加锁实现同步的过程中可能会死锁。

(4)对公有变量的同时读或写，可能对造成脏读等；

## 23、长连接与短连接

答：（1）就是TCP长连接和TCP短连接：

①TCP长连接：TCP长连接指建立连接后保持连接而不断开。若一段时间内没有数据传输，服务器会发送心跳包给客户端，判断客户端是否还在线，叫做TCP长连接中的keep alive。一般步骤：连接→数据传输→保持连接(心跳)→数据传输→保持连接(心跳)→……→关闭连接；

②TCP短连接：指连接建立并传输数据完成后，就断开连接。一般步骤：连接→数据传输→关闭连接；

③使用场景：长连接适合单对单通信且连接数不太多的情况；短连接适合连接数多且经常更换连接对象的；

**（2）HTTP是什么连接：**

①在HTTP/1.0中，默认使用的是短连接。但从 HTTP/1.1起，默认使用长连接，用以保持连接特性。使用长连接的HTTP协议，会在响应头有加入这行代码：

**[html]** [view plain](https://blog.csdn.net/Xiongchao99/article/details/74524807?locationNum=6&fps=1) [copy](https://blog.csdn.net/Xiongchao99/article/details/74524807?locationNum=6&fps=1)

1. Connection:keep-alive

注意：此处的keep-alive和上述TCP长连接原理介绍中的keep alive不是一个意思：此处表示告知服务器本http请求是长连接模式，而TCP长连接中的keep alive表示对客户端的保活检测。

②http长连接并不是一直保持连接

http的长连接也不会是永久保持连接，它有一个保持时间如20s（从上一次数据传输完成开始计时），可以在不同的服务器软件（如Apache）中设定这个时间，**若超过该时间限制仍然无数据通信传输，服务器就主动关闭该连接**。注：实现长连接要客户端和服务端都支持长连接。

③http连接实质：http的长连接/短连接实质上就是TCP的长/短连接。

## 24、二分图应用于最佳匹配问题（游客对房间的满意度之和最大问题）

答：题目：有n个游客和n个客房，每个游客对每间房有一个满意度，现要求做出一个入住安排，使得所有游客的满意度最大。

思路：用二分图解决，游客作为一边的顶点，客房作为另一边的顶点，取出所有最大匹配中满意度之和最大的方案。

实现：涉及匈牙利算法；

# 第二篇

## 1、class与struct的区别

答：C++中的struct对C中的struct进行了扩充，它已经不再只是一个包含不同数据类型的数据结构了，它已经获取了太多的功能：  
①struct能包含成员函数吗？ 能！  
②struct能继承吗？ 能！！  
③struct能实现多态吗？ 能！！！

既然这些它都能实现，那它和class还能有什么区别？

**最本质的一个区别就是成员默认属性和默认继承权限的不同：**

**①若不指明，struct成员的默认属性是public的，class成员的默认属性是private的；**

**②若不指明，struct成员的默认继承权限是public的，class成员的默认继承权限是private的；**

## 2、虚函数和纯虚函数

答：参考另一博文：<http://blog.csdn.net/xiongchao99/article/details/64441017#t80>

## 3、menset()函数

答：Memset用来将buffer开始的长为size的内存空间全部设置为字符c，一般用在对定义的字符串进行初始化为''或'/0'；这个函数在socket中多用于清空数组。参见：<http://blog.csdn.net/xiongchao99/article/details/64441017#t124>

# 腾讯研发类笔试面试试题（C++方向）

### 1、C和C++的特点与区别？

答：**（1）C语言特点：**  
1.作为一种面向过程的结构化语言，易于调试和维护；  
  
2.表现能力和处理能力极强，可以直接访问内存的物理地址；  
  
3.C语言实现了对硬件的编程操作，也适合于应用软件的开发；  
  
4.C语言还具有效率高，可移植性强等特点。  
  
**（2）C++语言特点：**

1.在C语言的基础上进行扩充和完善，使C++兼容了C语言的面向过程特点，又成为了一种面向对象的程序设计语言；

2.可以使用抽象数据类型进行基于对象的编程；

3.可以使用多继承、多态进行面向对象的编程；

4.可以担负起以模版为特征的泛型化编程。

**C++与C语言的本质差别**：在于C++是面向对象的，而C语言是面向过程的。或者说C++是在C语言的基础上增加了面向对象程序设

计的新内容，是对C语言的一次更重要的改革，使得C++成为软件开发的重要工具。

### 2、C++的多态

答：**C++的多态性用一句话概括**：在基类的函数前加上virtual关键字，在派生类中重写该函数，运行时将会根据对象的实际类型来

调用相应的函数。如果对象类型是派生类，就调用派生类的函数；如果对象类型是基类，就调用基类的函数。

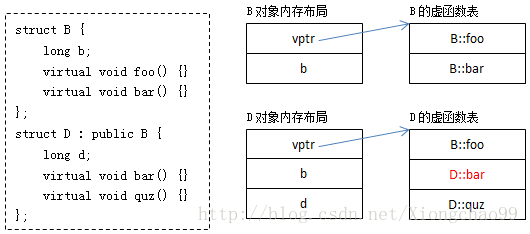
1）：用virtual关键字申明的函数叫做虚函数，虚函数肯定是类的成员函数；   
  
2）：存在虚函数的类都有一个一维的虚函数表叫做虚表，类的对象有一个指向虚表开始的虚指针。虚表是和类对应的，虚表指针是

和对象对应的；  
  
3）：**多态性是一个接口多种实现，是面向对象的核心，分为类的多态性和函数的多态性**。；  
  
4）：多态用虚函数来实现，结合动态绑定.；  
  
5）：纯虚函数是虚函数再加上 = 0；   
  
6）：抽象类是指包括至少一个纯虚函数的类；  
  
纯虚函数：virtual void fun()=0;即抽象类，必须在子类实现这个函数，即先有名称，没有内容，在派生类实现内容。

### 3、虚函数实现

答：简单地说，每一个含有虚函数（无论是其本身的，还是继承而来的）的类都至少有一个与之对应的虚函数表，其中存放着该类

所有的虚函数对应的函数指针。例：



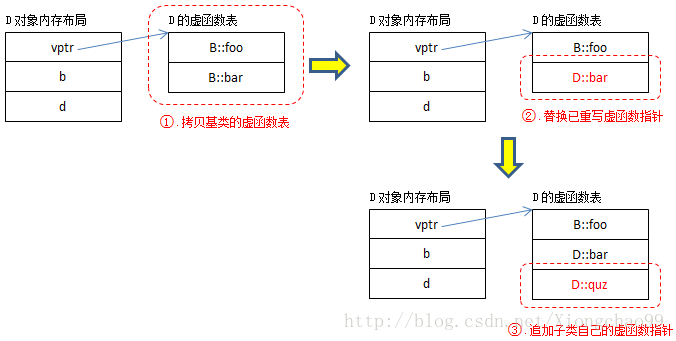
其中：

B的虚函数表中存放着B::foo和B::bar两个函数指针。

D的虚函数表中存放的既有继承自B的虚函数B::foo，又有重写（override）了基类虚函数B::bar的D::bar，还有新增的虚函数D::quz。

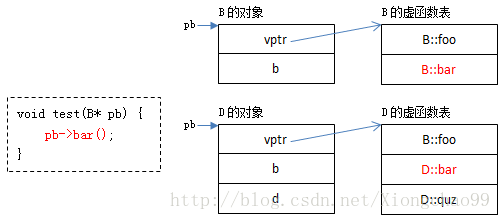
虚函数表构造过程：

从编译器的角度来说，B的虚函数表很好构造，D的虚函数表构造过程相对复杂。下面给出了构造D的虚函数表的一种方式（仅供参考）：



虚函数调用过程

以下面的程序为例：



### 4、C和C++内存分配问题

答：（1）**C语言**编程中的内存基本构成

**C的内存基本上分为4部分**：静态存储区、堆区、栈区以及常量区。他们的功能不同，对他们使用方式也就不同。

1.栈 ——由编译器自动分配释放；

2.堆 ——一般由程序员分配释放，若程序员不释放，程序结束时可能由OS回收；

3.全局区（静态区）——全局变量和静态变量的存储是放在一块的，初始化的全局变量和静态变量在一块区域，未初始化的全局变量

和未初始化的静态变量在相邻的另一块区域（C++中已经不再这样划分），程序结束释放；

4.另外还有一个专门放常量的地方，程序结束释放；

(a)函数体中定义的变量通常是在栈上；

(b)用malloc, calloc, realloc等分配内存的函数分配得到的就是在堆上；

(c)在所有函数体外定义的是全局量；

(d)加了static修饰符后不管在哪里都存放在全局区（静态区）；

(e)在所有函数体外定义的static变量表示在该文件中有效，不能extern到别的文件用；

(f)在函数体内定义的static表示只在该函数体内有效；

(g)另外，函数中的"adgfdf"这样的字符串存放在常量区。

（2）C++编程中的内存基本构造

**在C++中内存分成5个区**，分别是堆、栈、全局/静态存储区、常量存储区和代码区；

1、栈，就是那些由编译器在需要的时候分配，在不需要的时候自动清楚的变量的存储区，里面的变量通常是局部变量、函数参数等。

2、堆，就是那些由new分配的内存块，他们的释放编译器不去管，由我们的应用程序去控制，一般一个new就要对应一个delete。如

果程序员没有释放掉，那么在程序结束后，操作系统会自动回收。

3、全局/静态存储区，全局变量和静态变量被分配到同一块内存中，在以前的C语言中，全局变量又分为初始化的和未初始化的，在

C++里面没有这个区分了，他们共同占用同一块内存区。

4、常量存储区，这是一块比较特殊的存储区，他们里面存放的是常量，不允许修改（当然，你要通过非正当手段也可以修改）。

5、代码区 （.text段），存放代码（如函数），不允许修改（类似常量存储区），但可以执行（不同于常量存储区）。

**内存模型组成部分：自由存储区，动态区、静态区；**

根据c/c++对象生命周期不同，c/c++的内存模型有三种不同的内存区域，即：**自由存储区，动态区、静态区**。

自由存储区：局部非静态变量的存储区域，即平常所说的栈；

动态区： 用new ，malloc分配的内存，即平常所说的堆；

静态区：全局变量，静态变量，字符串常量存在的位置；

注：代码虽然占内存，但不属于c/c++内存模型的一部分；

**一个正在运行着的C编译程序占用的内存分为5个部分：代码区、初始化数据区、未初始化数据区、堆区 和栈区；**

（1）代码区（text segment）：代码区指令根据程序设计流程依次执行，对于顺序指令，则只会执行一次（每个进程），如果反复，则需要使用跳转指令，如果进行递归，则需要借助栈来实现。注意：代码区的指令中包括操作码和要操作的对象（或对象地址引用）。如果是立即数（即具体的数值，如5），将直接包含在代码中；

（2）全局初始化数据区/静态数据区（Data Segment）：只初始化一次。  
  
（3）未初始化数据区（BSS）：在运行时改变其值。  
  
（4）栈区（stack）：由编译器自动分配释放，存放函数的参数值、局部变量的值等，其操作方式类似于数据结构中的栈。  
  
（5）堆区（heap）：用于动态内存分配。

为什么分成这么多个区域？

主要基于以下考虑：

#代码是根据流程依次执行的，一般只需要访问一次，而数据一般都需要访问多次，因此单独开辟空间以方便访问和节约空间。  
#未初始化数据区在运行时放入栈区中，生命周期短。  
#全局数据和静态数据有可能在整个程序执行过程中都需要访问，因此单独存储管理。  
#堆区由用户自由分配，以便管理。

更多内容见：<http://www.cnblogs.com/Stultz-Lee/p/6751522.html>

### 5、协程

答：定义：协程是一种用户态的轻量级线程。

协程拥有自己的寄存器上下文和栈。协程调度切换时，将寄存器上下文和栈保存到其他地方，在切回来的时候，恢复先前保存的寄存器上下文和栈。因此：协程能保留上一次调用时的状态（即所有局部状态的一个特定组合），每次过程重入时，就相当于进入上一次调用的状态，换种说法：进入上一次离开时所处逻辑流的位置；

线程是抢占式，而协程是协作式；

协程的优点：

跨平台  
跨体系架构  
无需线程上下文切换的开销  
无需原子操作锁定及同步的开销  
方便切换控制流，简化编程模型  
高并发+高扩展性+低成本：一个CPU支持上万的协程都不是问题。所以很适合用于高并发处理。  
  
协程的缺点：

无法利用多核资源：**协程的本质是个单线程**,它不能同时将 单个CPU 的多个核用上,协程需要和进程配合才能运行在多CPU；  
进行阻塞（Blocking）操作（如IO时）会阻塞掉整个程序：这一点和事件驱动一样，可以使用异步IO操作来解决。

### 6、CGI的了解

答：CGI：通用网关接口（Common Gateway Interface）是一个Web服务器主机提供信息服务的标准接口。通过CGI接口，Web服务

器就能够获取客户端提交的信息，转交给服务器端的CGI程序进行处理，最后返回结果给客户端。

CGI通信系统的组成是两部分：一部分是html页面，就是在用户端浏览器上显示的页面。另一部分则是运行在服务器上的Cgi程序。

### 7、进程间通信方式和线程间通信方式

答：（1）进程间通信方式：

# 管道( pipe )：管道是一种半双工的通信方式，数据只能单向流动，而且只能在具有亲缘关系的进程间使用。进程的亲缘关系通常是指父子进程关系。  
  
# 信号量( semophore ) ： 信号量是一个计数器，可以用来控制多个进程对共享资源的访问。它常作为一种锁机制，防止某进程正在访问共享资源时，其他进程也访问该资源。因此，主要作为进程间以及同一进程内不同线程之间的同步手段。

# 消息队列( message queue ) ： 消息队列是由消息的链表，存放在内核中并由消息队列标识符标识。消息队列克服了信号传递信息少、管道只能承载无格式字节流以及缓冲区大小受限等缺点。  
  
# 共享内存( shared memory ) ：共享内存就是映射一段能被其他进程所访问的内存，这段共享内存由一个进程创建，但多个进程都可以访问。共享内存是最快的 IPC 方式，它是针对其他进程间通信方式运行效率低而专门设计的。它往往与其他通信机制，如信号两，配合使用，来实现进程间的同步和通信。

# 套接字( socket ) ： 套解口也是一种进程间通信机制，与其他通信机制不同的是，它可用于不同及其间的进程通信。

（2）线程间通信方式：

#全局变量；

#Messages消息机制；

#CEvent对象（MFC中的一种线程通信对象，通过其触发状态的改变实现同步与通信）。

### 8、TCP握手与释放

答：（1）握手

#第一次握手：主机A发送握手信号syn＝1和seq=x（随机产生的序列号）的数据包到服务器，主机B由SYN=1知道，A要求建立联机；

#第二次握手：主机B收到请求后要确认联机信息，向A发送syn=1，ack=x（x是主机A的Seq）+1，以及随机产生的确认端序列号

seq=y的包；

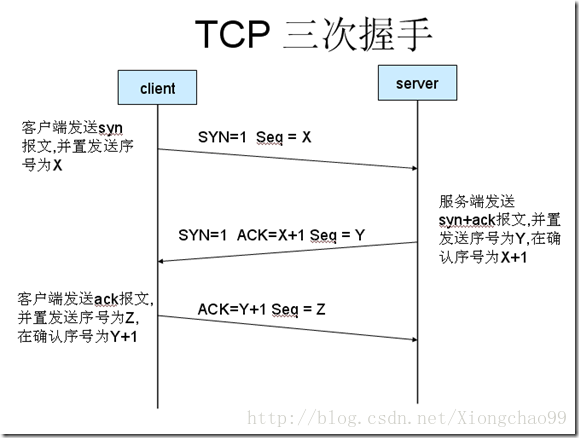
#第三次握手：主机A收到后检查ack是否正确（ack=x+1），即第一次发送的seq+1，若正确，主机A会再发送ack=y+1，以及随机序

列号seq=z，主机B收到后确认ack值则连接建立成功；

#完成三次握手，主机A与主机B开始传送数据。

注：上述步骤中，第二和第三次确认包中都还包含一个标志位未予以说明，该标志位为1表示正常应答；

具体可见图片：



**为什么需要“三次握手”？**

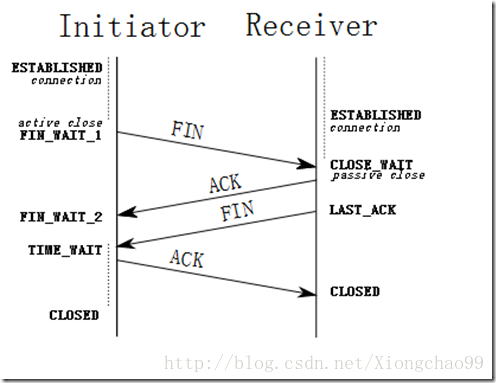
**“三次握手”的目的是“为了防止已失效的连接请求报文段突然又传送到了服务端，因而产生错误”。**具体例如：client发出的第一个连接请求报文段并没有丢失，而是在某个网络结点长时间的滞留了，以致延误到连接释放以后的某个时间才到达server。本来这是一个早已失效的报文段。但server收到此失效的连接请求报文段后，就误认为是client再次发出的一个新的连接请求。于是就向client发出确认报文段，同意建立连接。假设不采用“三次握手”，那么只要server发出确认，新的连接就建立了。由于现在client并没有发出建立连接的请求，因此不会理睬server的确认，也不会向server发送数据。但server却以为新的运输连接已经建立，并一直等待client发来数据。这样，server的很多资源就白白浪费掉了。采用“三次握手”的办法可以防止上述现象发生。例如刚才那种情况，client不会向server的确认发出确认。server由于收不到确认，就知道client并没有要求建立连接。**主要目的防止server端一直等待，浪费资源。**

（2）挥手

**由于TCP连接是全双工的，因此每个方向都必须单独进行关闭。**这原则是当一方完成它的数据发送任务后就能发送一个FIN来终止这个方向的连接。收到一个 FIN只意味着这一方向上没有数据流动，一个TCP连接在收到一个FIN后仍能发送数据。首先进行关闭的一方将执行主动关闭，而另一方执行被动关闭。  
　　(1) TCP客户端发送一个FIN，用来关闭客户到服务器的数据传送(报文段4)；  
　　(2) 服务器收到这个FIN，发回一个ACK，确认序号为收到的序号加1(报文段5)。和SYN一样，一个FIN将占用一个序号；  
　　(3) 服务器关闭客户端的连接后，再发送一个FIN给客户端(报文段6)；  
　　(4) 客户段收到服务端的FIN后，发回ACK报文确认，并将确认序号设置为收到序号加1(报文段7)；

注意：TCP连接的任何一方都可以发起挥手操作，上述步骤只是两种之一；

具体过程见图：



**为什么是“四次挥手”？**  
因为当收到对方的FIN报文通知时，它仅仅表示对方没有数据发送给你了；但未必你所有的数据都全部发送给对方了，所以你可能还需要发送一些数据给对方，再发送FIN报文给对方来表示你同意现在可以关闭连接了，故这里的ACK报文和FIN报文多数情况下都是分开发送的，也就造成了4次挥手。

**握手，挥手过程中各状态介绍：**  
（1）3次握手过程状态：  
　　#LISTEN: 这个也是非常容易理解的一个状态，表示服务器端的某个SOCKET处于监听状态，可以接受连接了。

#SYN\_SENT: 当客户端SOCKET执行CONNECT连接时，它首先发送SYN报文，因此也随即它会进入到了SYN\_SENT状态，并等待服务端的发送三次握手中的第2个报文。SYN\_SENT状态表示客户端已发送SYN报文。(发送端)

　　#SYN\_RCVD: 这个状态与SYN\_SENT遥想呼应这个状态表示接受到了SYN报文，在正常情况下，这个状态是服务器端的SOCKET在建立TCP连接时的三次握手会话过程中的一个中间状态，很短暂，基本上用netstat你是很难看到这种状态的，除非你特意写了一个客户端测试程序，故意将三次TCP握手过程中最后一个ACK报文不予发送。因此这种状态时，当收到客户端的ACK报文后，它会进入到ESTABLISHED状态。(服务器端)

#ESTABLISHED：这个容易理解了，表示连接已经建立了。

（2）4次挥手过程状态：  
　　**#FIN\_WAIT\_1:** 这个状态要好好解释一下，其实FIN\_WAIT\_1和FIN\_WAIT\_2状态的真正含义都是表示等待对方的FIN报文。而这两种状态的区别是：FIN\_WAIT\_1状态实际上是当SOCKET在ESTABLISHED状态时，它想主动关闭连接，向对方发送了FIN报文，此时该SOCKET即进入到FIN\_WAIT\_1状态。而当对方回应ACK报文后，则进入到FIN\_WAIT\_2状态，当然在实际的正常情况下，无论对方何种情况下，都应该马上回应ACK报文，所以FIN\_WAIT\_1状态一般是比较难见到的，而FIN\_WAIT\_2状态还有时常常可以用netstat看到。(主动方)

**#FIN\_WAIT\_2：**上面已经详细解释了这种状态，实际上FIN\_WAIT\_2状态下的SOCKET，表示半连接，也即有一方要求close连接，但另外还告诉对方，我暂时还有点数据需要传送给你(ACK信息)，稍后再关闭连接。(主动方)

　　#TIME\_WAIT: 表示收到了对方的FIN报文，并发送出了ACK报文，就等2MSL后即可回到CLOSED可用状态了。如果FIN\_WAIT\_1状态下，收到了对方同时带FIN标志和ACK标志的报文时，可以直接进入到TIME\_WAIT状态，而无须经过FIN\_WAIT\_2状态。(主动方)

#CLOSING(比较少见): 这种状态比较特殊，实际情况中应该是很少见，属于一种比较罕见的例外状态。正常情况下，当你发送FIN报文后，按理来说是应该先收到(或同时收到)对方的ACK报文，再收到对方的FIN报文。但是CLOSING状态表示你发送FIN报文后，并没有收到对方的ACK报文，反而却也收到了对方的FIN报文。什么情况下会出现此种情况呢?其实细想一下，也不难得出结论：那就是如果双方几乎在同时close一个SOCKET的话，那么就出现了双方同时发送FIN报文的情况，也即会出现CLOSING状态，表示双方都正在关闭SOCKET连接。

　　#CLOSE\_WAIT: 这种状态的含义其实是表示在等待关闭。怎么理解呢?当对方close一个SOCKET后发送FIN报文给自己，你系统毫无疑问地会回应一个ACK报文给对方，此时则进入到CLOSE\_WAIT状态。接下来呢，实际上你真正需要考虑的事情是察看你是否还有数据发送给对方，如果没有的话，那么你也就可以close这个SOCKET，发送FIN报文给对方，也即关闭连接。所以你在CLOSE\_WAIT状态下，需要完成的事情是等待你去关闭连接。(被动方)

#LAST\_ACK: 这个状态还是比较容易好理解的，它是被动关闭一方在发送FIN报文后，最后等待对方的ACK报文。当收到ACK报文后，也即可以进入到CLOSED可用状态了。(被动方)

### 9、http和https的区别？

答：HTTPS协议是由SSL+HTTP协议构建的可进行加密传输、身份认证的网络协议，要比http协议安全。

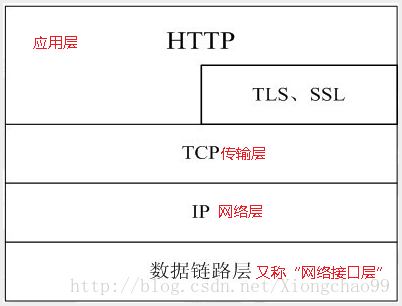
HTTPS（Secure Hypertext Transfer Protocol）安全超文本传输协议，与http主要区别在于：

#http是超文本传输协议，信息是明文传输，https 则是具有安全性的ssl加密传输协议；

#http和https使用的是完全不同的连接方式用的端口也不一样,前者是80,后者是443；

**下面具体介绍一下HTTP和HTTPS协议：**

**首先说明一下：HTTP和HTTPS协议是应用层协议；**

****

**上图充分表明：HTTP是应用层协议，并且HTTPS是在HTTP协议基础上添加SSL等加密策略后的协议；**

**TLS/SSL中使用了非对称加密，对称加密以及HASH算法。**

**（1）Http协议**

1）HTTP协议和TCP协议之间的区别联系

①TPC/IP协议是传输层协议，主要解决数据如何在网络中传输，而HTTP是应用层协议，主要解决如何包装数据；

②HTTP的默认端口号是80，TCP/IP协议通信编程时端口号需要自己指定（例如socket编程）；

③HTTP协议是在TCP/IP协议基础上实现的，即HTTP数据包是经过TCP/IP协议实现传输的；

④HTTP是无状态的短连接协议，TCP是有状态的长连接协议；

HTTP是在有状态长连接TCP/IP协议的基础上实现的，为什么却是无状态短连接协议？

答：因为HTTP协议每次请求结束就会自动关闭连接，这样就变成了短连接；

短连接又导致了该次请求相关信息的丢失，也就造成了HTTP协议对于前期事务处理没有记忆能力，故为无状态协议。

2）HTTP协议其完整的工作过程可分为四步：

**①连接**：首先客户机与服务器需要建立连接（由TCP/IP握手连接实现）。只要单击某个超级链接，HTTP的工作开始；

**②请求**：建立连接后，客户机发送一个请求给服务器，请求方式的格式为：统一资源标识符（URL）、协议版本号，后边是MIME信息包括请求修饰符、客户机信息和可能的内容；

**③应答**：服务器接到请求后，给予相应的响应信息，其格式为一个状态行，包括信息的协议版本号、一个成功或错误的代码，后边是MIME信息包括服务器信息、实体信息和可能的内容。客户端接收服务器所返回的信息通过浏览器显示在用户的显示屏上；

**④关闭：**当应答结束后，浏览器和服务器关闭连接，以保证其他浏览器可以与服务器进行连接。

**更完整的过程可能如下**：

域名解析 --> 发起TCP的3次握手 --> 建立TCP连接后发起http请求 --> 服务器响应http请求，浏览器得到html代码 --> 浏览器解析html代码，并请求html代码中的资源（如js、css、图片等） --> 浏览器对页面进行渲染呈现给用户。

       如果在以上过程中的某一步出现错误，那么产生错误的信息将返回到客户端，有显示屏输出。对于用户来说，这些过程是由HTTP自己完成的，用户只要用鼠标点击，等待信息显示就可以了。

**（2）Https协议**

HTTPS握手过程包括五步：

1）浏览器请求连接；

2）服务器返回证书：证书里面包含了网站地址，加密公钥，以及证书的颁发机构等信息。

3）浏览器收到证书后作以下工作：

    a) 验证证书的合法性；

    b) 生成随机（对称）密码，取出证书中提供的**公钥**对随机密码加密；

    c) 将之前生成的加密随机密码等信息发送给网站；

4）服务器收到消息后作以下的操作：

    a) 使用自己的**私钥**解密浏览器用公钥加密后的消息，并验证HASH是否与浏览器发来的一致；

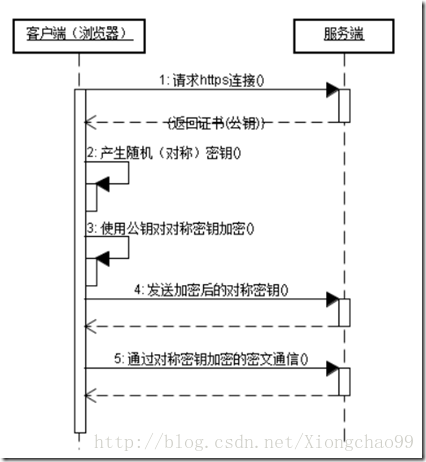
    b) 使用加密的随机对称密码加密一段消息，发送给浏览器；

5）浏览器解密并计算握手消息的HASH：如果与服务端发来的HASH一致，此时握手过程结束，之后所有的通信数据将由之前浏览

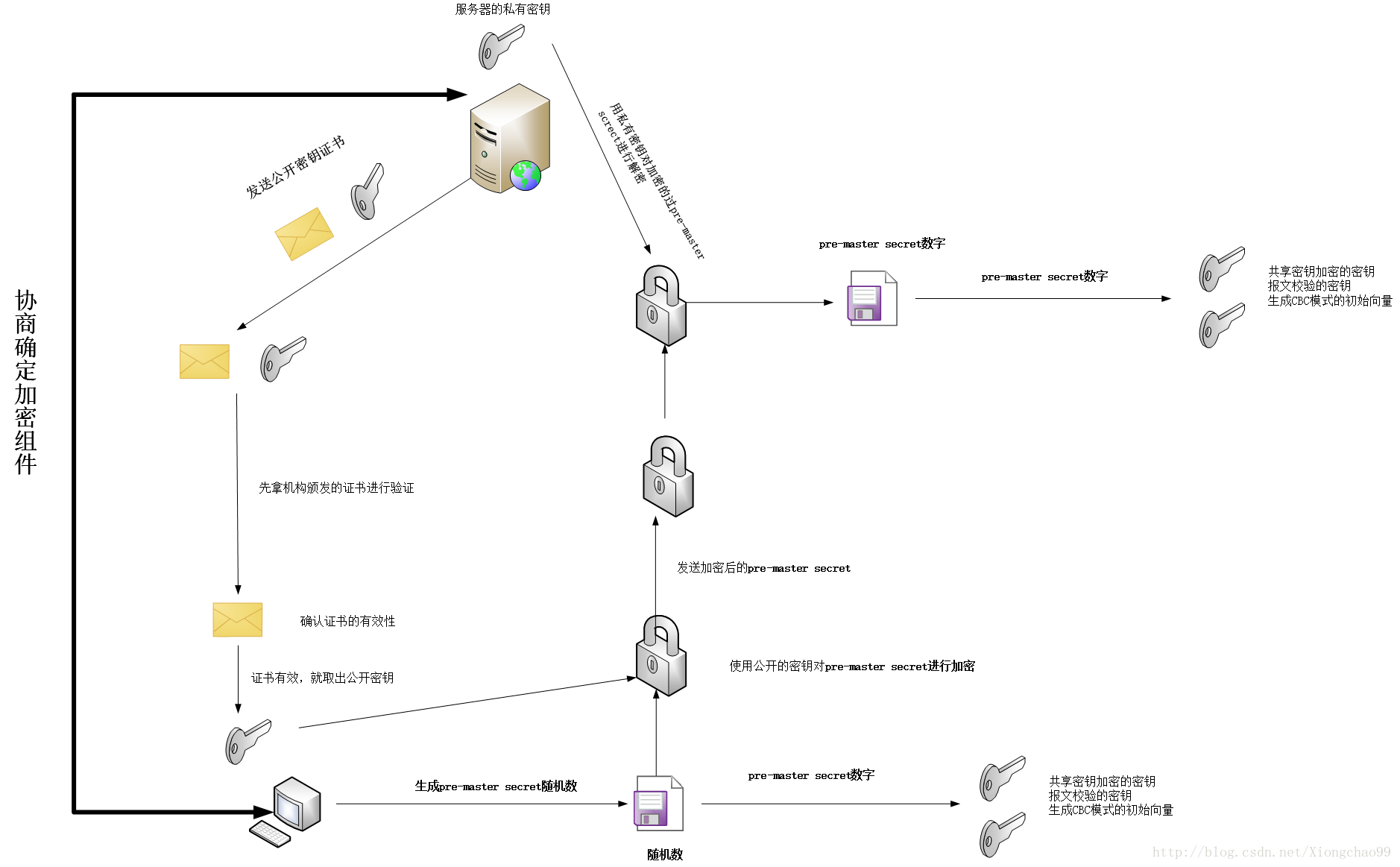
器生成的随机密码并利用对称加密算法进行加密。

注意：**服务器有两个密钥，一个公钥、一个私钥，只有私钥才可以解密公钥加密的消息；**

如图：



或者如下图：



**HTTPS协议、SSL、和数字证书的关系介绍：**

概述：**对于HTTPS协议，所有的消息都是经过SSL协议方式加密，而支持加密的文件正是数字证书；**

（1）SSL

SSL常用的加密算法：对称密码算法、非对称密码算法、散列算法；

SSL的加密过程：需要注意的是非对称加解密算法的效率要比对称加解密要低的多。所以SSL在握手过程中使用非对称密码算法来

协商密钥，实际使用对称加解密的方法对http内容加密传输；

（2）数字证书

数字证书是用于在INTERNET上标识个人或者机构身份的一种技术手段，它通过由一些公认的权威机构所认证，从而可以保证其

安全地被应用在各种场合。**证书里面包含了网站地址，加密公钥，以及证书的颁发机构等信息。**

### 10、虚拟内存的概念与介绍

答：虚拟内存中，允许将一个作业分多次调入内存，需要时就调入，不需要的就先放在外存。因此，虚拟内存的实需要建立在离散

分配的内存管理方式的基础上。虚拟内存的实现有以下三种方式：  
#请求分页存储管理  
#请求分段存储管理  
#请求段页式存储管理

虚拟内存的意义：

一，**虚拟内存可以使得物理内存更加高效**。虚拟内存使用置换方式，需要的页就置换进来，不需要的置换出去，使得内存中只保存了需要的页，提高了利用率，也避免了不必要的写入与擦除；

二，**使用虚拟地址可以使内存的管理更加便捷**。在程序编译的时候就会生成虚拟地址，该虚拟地址并不是对应一个物理地址，使得也就极大地减少了地址被占用的冲突，减少管理难度；

三，**为了安全性的考虑。**在使用虚拟地址的时候，暴露给程序员永远都是虚拟地址，而具体的物理地址在哪里，这个只有系统才了解。这样就提

高了系统的封装性。

### 11、单链表的反转算法

答：思想：创建3个指针，分别指向上一个节点、当前节点、下一个节点，遍历整个链表的同时，将正在访问的节点指向上一个节点，当遍历结束后，就同时完成了链表的反转。

实现代码：

**[cpp]** [view plain](https://blog.csdn.net/xiongchao99/article/details/73381280) [copy](https://blog.csdn.net/xiongchao99/article/details/73381280)

1. ListNode\* ReverseList(ListNode\* pHead) {
2. ListNode \*p,\*q,\*r;
3. **if**(pHead==NULL || pHead->next==NULL){
4. **return** pHead;
5. }**else**{
6. p=pHead;
7. q=p->next;
8. pHead->next=NULL;
9. **while**(q!=NULL){
10. r=q->next;
11. q->next=p;
12. p=q;
13. q=r;
14. }
15. **return** p;
16. }
17. }

### 12、红黑树以及其查找复杂度

答：（1）红黑树来源于二叉搜索树，其在关联容器如map中应用广泛，主要优势在于其查找、删除、插入时间复杂度小，但其也有缺点，就是容易偏向一边而变成一个链表。

红黑树是一种二叉查找树，但在每个结点上增加一个存储位表示结点的颜色，可以是Red或Black。也就是说，红黑树是在二叉

查找树基础上进一步实现的；

红黑树的五个性质：

性质1. 节点是红色或黑色；

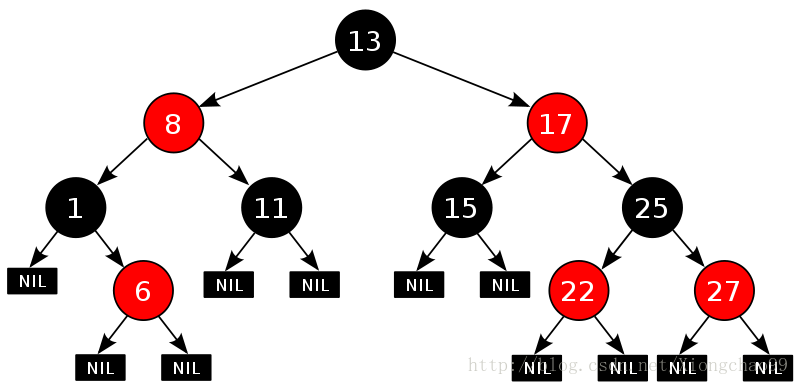
性质2. 根节点是黑色；

性质3 每个叶节点（指树的末端的NIL指针节点或者空节点）是黑色的；

性质4 每个红色节点的两个子节点都是黑色。(从每个叶子到根的所有路径上不能有两个连续的红色节点)；

性质5. 从任一节点到其每个尾端NIL节点或者NULL节点的所有路径都包含相同数目的黑色节点。

**（注：上述第3、5点性质中所说的NIL或者NULL结点，并不包含数据，只充当树的路径结束的标志，即此叶结点非常见的叶子结点）。**



因为一棵由n个结点随机构造的二叉查找树的高度为lgn，所以顺理成章，二叉查找树的一般操作的执行时间为O(lgn)。但二叉查

找树若退化成了一棵具有n个结点的线性链后，则这些操作最坏情况运行时间为O(n)；

红黑树虽然本质上是一棵二叉查找树，但它在二叉查找树的基础上增加以上五个性质使得红黑树相对平衡，从而保证了

**红黑树的查找、插入、删除的时间复杂度最坏为O(log n)**。

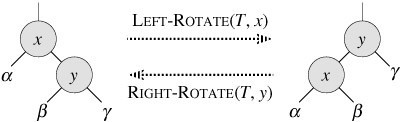
**（2）左旋右旋**

红黑树插入或删除后，一般就会改变红黑树的特性，要恢复红黑树上述5个性质，一般都要那就要做2方面的工作：

1、部分结点颜色，重新着色

2、调整部分指针的指向，即左旋、右旋。

左选右旋如图所示：



左旋，如图所示（左->右），以x->y之间的链为“支轴”进行，使y成为该新子树的根，x成为y的左孩子，而y的左孩子则成为x的右孩

子。算法很简单，旋转后各个结点从左往右，仍然都是从小到大。

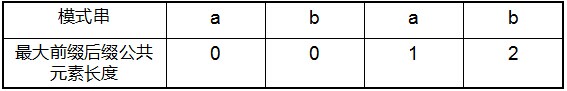
左旋代码实现，分三步:  
（1） 开始变化，y的左孩子成为x的右孩子；  
（2） y成为x的父结点；  
（3） x成为y的左孩子；  
  
右旋类似，不再累述；

### 13、KPM字符串匹配

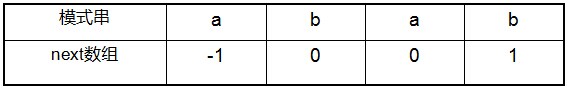
**（1）KMP匹配算法代码实现：**

**[cpp]** [view plain](https://blog.csdn.net/xiongchao99/article/details/73381280) [copy](https://blog.csdn.net/xiongchao99/article/details/73381280)

1. **int** KmpSearch(**char**\* s, **char**\* p)
2. {
3. **int** i = 0;
4. **int** j = 0;
5. **int** sLen = strlen(s);
6. **int** pLen = strlen(p);
7. **while** (i < sLen && j < pLen)
8. {
9. //①如果j = -1，或者当前字符匹配成功（即S[i] == P[j]），都令i++，j++
10. **if** (j == -1 || s[i] == p[j])
11. {
12. i++;
13. j++;
14. }
15. **else**
16. {
17. //②如果j != -1，且当前字符匹配失败（即S[i] != P[j]），则令 i 不变，j = next[j]
18. //next[j]即为j所对应的next值
19. j = next[j];
20. }
21. }
22. **if** (j == pLen)
23. **return** i - j;
24. **else**
25. **return** -1;
26. }

**（2）next数组求取**  
上述（1）中最重要的就是：一旦不匹配，模式串不是向后移动一位，而是根据前面匹配信息移动多位。而这个多位获得就是根据next数组，下面有next数组的求取方式：  
Next数组是根据模式串的前缀后缀获取的，如下：  
①寻找前缀后缀最长公共元素长度  
举个例子，如果给定的模式串为“abab”，那么它的各个子串的前缀后缀的公共元素的最大长度如下表格所示：  


比如对于字符串aba来说，它有长度为1的相同前缀后缀a；而对于字符串abab来说，它有长度为2的相同前缀后缀ab（相同前缀后缀的长度为k + 1，k + 1 = 2）。  
  
  
②求next数组  
next 数组考虑的是除当前字符外的最长相同前缀后缀，所以通过第①步骤求得各个前缀后缀的公共元素的最大长度后，只要稍作变形即可：**将第①步骤中求得的数组整体右移一位，然后第一个元素赋为-1即可**（注意：字符串下标需要从0开始），如下表格所示：



比如对于aba来说，第3个字符a之前的字符串ab中有长度为0的相同前缀后缀，所以第3个字符a对应的next值为0；而对于abab来说，第4个字符b之前的字符串aba中有长度为1的相同前缀后缀a，所以第4个字符b对应的next值为1（相同前缀后缀的长度为k，k = 1）。

KMP的next 数组相当于告诉我们：当模式串中的某个字符跟文本串中的某个字符匹配失配时，模式串下一步应该跳到哪个位置（具体：保持测试串的下标i不变，使得匹配串的下标j=next[j]）。

**前缀后缀长度求取以及next数组获取：**

如果给定的模式串是：“ABCDABD”，从左至右遍历整个模式串，其各个子串的前缀后缀分别如下表格所示：



也就是说，原模式串子串对应的各个前缀后缀的公共元素的最大长度表为：

0 0 0 0 1 2 0；

故对应的next数组为：-1 0 0 0 0 1 2；

（注意：这里的字符串下标是从0开始的，若从1开始，next数组所有元素都对应要加1。）

求取next的实现代码：

**[cpp]** [view plain](https://blog.csdn.net/xiongchao99/article/details/73381280) [copy](https://blog.csdn.net/xiongchao99/article/details/73381280)

1. string T;  //T为模式串
2. cin>>T;
3. **int** len=T.size();
4. queue<**int**> MaxLen;
5. vector<**int**> next;
6. MaxLen.push(0);  //第一个元素都设为0
7. **for**(**int** i=1;i<len;i++)
8. {
9. **int** k=1,maxLen=0;
10. **while**(k<=i)
11. {
12. **if**(T.substr(0,k)==T.substr(i-k+1,k))
13. {
14. maxLen=k;
15. }
16. k++;
17. }
18. MaxLen.push(maxLen);
19. }
20. cout<<endl;
22. next.push\_back(-1); //第一个元素都设为-1
23. **while**(MaxLen.size()>1)
24. {
25. **int** temp=MaxLen.front();
26. next.push\_back(temp);
27. MaxLen.pop();
28. cout<<temp<<' ';
29. }

### 14、TCP超时等待、重传以及流量控制

答：TCP等待时间需要设定，超过了就认为丢包，需要重传；

为了防止拥塞情况，一般会采用流量控制，其实现手段是用滑动窗口限制客户端发送分组数量；

### 15、数据库引擎

答：数据库引擎是用于存储、处理和保护数据的核心服务。利用数据库引擎可控制访问权限并快速处理事务，从而满足企业内大多

数需要处理大量数据的应用程序的要求。

简言之，数据库引擎就是一段用于支撑所有数据库操作的核心程序，就如名称一样，是一个车的引擎功能；

常见的数据库引擎有：

（1）Microsoft JET (Joint Engineering Technologe) 用于Access和VB的内嵌数据库功能的核心元素；

（2）ODBC（Open DataBase Connectivity，开放数据库互连）是由Microsoft定义的一种数据库访问标准，它提供一种标准的数据

库访问方法以访问不同平台的数据库。一个ODBC应用程序既可以访问在本地PC机上的数据库，也可以访问多种异构平台上的数据

库，例如SQL Server、Oracle或者DB2；

（3）OLE DB是Microsoft开发的最新数据库访问接口，Microsoft将其定义为ODBC接班人；

（4）MYSQL支持三个引擎：ISAM、MYISAM和HEAP。另外两种类型INNODB和BERKLEY（BDB）也常常可以使用；

①ISAM执行读取操作的速度很快，而且不占用大量的内存和存储资源。ISAM的两个主要不足之处在于，它不 支持事务处理，也不能够容错；

②MyISAM是MySQL的ISAM扩展格式和缺省的数据库引擎MYISAM。除了提供ISAM里所没有的索引和字段管理的大量功能，

MyISAM还使用一种表格锁定的机制，来优化多个并发的读写操作，其代价是你需要经常运行OPTIMIZE TABLE命令，来恢复被更新

机制所浪费的空间；

③HEAP允许只驻留在内存里的临时表格。驻留在内存里让HEAP要比ISAM和MYISAM都快，但是它所管理的数据是不稳定的，

而且如果在关机之前没有进行保存，那么所有的数据都会丢失。

### 16、数据库索引

答：**定义**：数据库索引是对数据库表中一列或多列的值进行排序的一种结构，使用索引可快速访问数据库表中的特定信息；

**举例**：employee 表的人员编号列（id）就是数据库索引，select \* from employee where id=10000即可查找编号10000的人员信息。如果没有索引，必须遍历整个表直到id=10000；

**数据库索引作用：**

一，大大加快 数据的检索速度，这也是创建索引的**最主要的原因**；  
二，保证数据库表中每一行数据的唯一性；  
三，可以加速表和表之间的连接，特别是在实现数据的参考完整性方面特别有意义；   
四，在使用分组和排序子句进行数据检索时，同样可以显著减少查询中分组和排序的时间；   
五，通过使用索引，可以在查询的过程中，使用优化隐藏器，提高系统的性能。

**数据库索引缺陷：**

一，表的增删改查、创建索引和维护索引要耗费时间；  
二，索引需要占物理空间；

**数据库索引的两个特征**：索引有两个特征，即唯一性索引和复合索引；

①唯一 性索引保证在索引列中的全部数据是唯一的，不会包含冗余数据；

②复合索引就是一个索引创建在两个列或者多个列上，搜索时需要两个或者多个索引列作为一个关键值；

数据库索引好比是一本书前面的目录，索引分为聚簇索引和非聚簇索引两类：

1）聚簇索引是按照数据存放的物理位置为顺序的，其多个连续行的访问速度更快；

2）非聚簇索引是按照数据存放的逻辑位置为顺序的，其单行访问速度更快；

**局部性原理与磁盘预读**

**局部性原理**：当一个数据被用到时，其附近的数据也通常会马上被使用。程序运行期间所需要的数据通常比较集中；

**磁盘预读**：正是由于局部性原理以及数据存储磁盘的读写速度慢的原因，每次对数据库进行读取都不是按需读取，而是读取多

于需求数据区域内的数据到内存，用于后续使用，提高写读取数据速度；

注：磁盘预读一般都是每次读取逻辑上的一页，或物理上的一块，不管实际需求是多少；

数据库索引的实现通常使用B树及其变种B+树，下面进行**B-/+Tree结构的数据库索引的性能分析**：

**（1）B树索引结构：**

数据库系统的设计者巧妙利用了磁盘预读原理，将B树的一个节点的大小设为等于一个页，这样每个节点只需要一次I/O就可以

完全载入。为了达到这个目的，在实际实现B-Tree还需要使用如下技巧：

——**每次新建节点时，直接申请一个页的空间，这样就保证一个节点物理上也存储在一个页；**

**B-Tree中一次检索最多需要h-1次I/O（磁盘IO不包括根节点，因为根节点常驻内存），渐进复杂度为O(h)=O(logdN)。**一

般实际应用中，出度d是非常大的数字，通常超过100，因此h非常小（通常不超过3）。

而红黑树这种结构，h明显要深的多。由于逻辑上很近的节点（父子）物理上可能很远，无法利用局部性，所以红黑树的I/O渐进

复杂度也为O(h)，效率明显比B-Tree差很多。

**所以，B树结构的数据库索引，在元素查找上效率很高；**

**（2）B+树的索引结构：**

B+树则适当牺牲检索的时间复杂度（都必须检索到叶子结点），但改善了节点插入和删除的时间复杂度（类似用链表改善数组的效

果），所以B+树属于一种折中选择。

**1. C 和 C++ 区别**

**2. const 有什么用途**

主要有三点：

      1：定义只读变量，即常量

      2：修饰函数的参数和函数的返回值

      3： 修饰函数的定义体，这里的函数为类的成员函数，被const修饰的成员函数代表不修改成员变量的值

**3. 指针和引用的区别**

   1：引用是变量的一个别名，内部实现是只读指针

    2：引用只能在初始化时被赋值，其他时候值不能被改变，指针的值可以在任何时候被改变

    3：引用不能为NULL，指针可以为NULL

    4：引用变量内存单元保存的是被引用变量的地址

    5：“sizeof 引用" = 指向变量的大小 ， "sizeof 指针"= 指针本身的大小

    6：引用可以取地址操作，返回的是被引用变量本身所在的内存单元地址

    7：引用使用在源代码级相当于普通的变量一样使用，做函数参数时，内部传递的实际是变量地址

**4. C++中有了malloc / free , 为什么还需要 new / delete**

1,malloc与free是C++/C语言的标准库函数，new/delete是C++的运算符。它们都可用于申请动态内存和释放内存。

2,对于非内部数据类型的对象而言，光用maloc/free无法满足动态对象的要求。

对象在创建的同时要自动执行构造函数，对象在消亡之前要自动执行析构函数。

由于malloc/free是库函数而不是运算符，不在编译器控制权限之内，不能够把执行构造函数和析构函数的任务强加于malloc/free。

3,因此C++语言需要一个能完成动态内存分配和初始化工作的运算符new，以一个能完成清理与释放内存工作的运算符delete。注意new/delete不是库函数。

**5. 编写类String 的构造函数，析构函数，拷贝构造函数和赋值函数**

**6. 多态的实现**

**7. 单链表的逆置**

**8. 堆和栈的区别**

一个由c/C++编译的程序占用的内存分为以下几个部分

1、栈区（stack）― 由编译器自动分配释放 ，存放函数的参数值，局部变量的值等。其操作方式类似于数据结构中的栈。

2、堆区（heap） ― 一般由程序员分配释放， 若程序员不释放，程序结束时可能由OS回收 。

注意它与数据结构中的堆是两回事，分配方式倒是类似于链表，呵呵。

3、全局区（静态区）（static）―，全局变量和静态变量的存储是放在一块的，

初始化的全局变量和静态变量在一块区域， 未初始化的全局变量和未初始化的静态变量在相邻的另一块区域。 - 程序结束后有系统释放

4、文字常量区 ―常量字符串就是放在这里的。 程序结束后由系统释放

5、程序代码区―存放函数体的二进制代码。

**10. 不调用C/C++ 的字符串库函数，编写strcpy**

char \* strcpy(char \* strDest,const char \* strSrc)

{

if ((strDest==NULL)||strSrc==NULL))

return NULL;

char \* strDestCopy=strDest;

while ((\*strDest++=\*strSrc++)!='\0');

\*strDest = '\0';

return strDestCopy;

}

**11. 关键字static的作用**

   1.  函数体内 static 变量的作用范围为该函数体，不同于 auto 变量， 该变量的内存只被分配一次，因此其值在下次调用时仍维持上次的值

    2.  在模块内的 static 全局变量可以被模块内所有函数访问，但不能被模块外其他函数访问

    3.  在模块内的static 函数只可被这一模块内的其他函数调用，这个函数的使用范围被限制在声明它的模块内

    4.  在类的static 成员变量属于整个类所拥有，对类的所以对象只有一份拷贝

    5.  在类中的 static 成员函数属于整个类所拥有，这个函数不接收 this 指针，因而只能访问类的 static 成员变量

     介绍它最重要的一条：隐藏。（static函数，static变量均可） --> 对应上面的2、3项  
        当同时编译多个文件时，所有未加static前缀的全局变量和函数都具有全局可见性。  
        举例来说明。同时编译两个源文件，一个是a.c，另一个是main.c。

//a.c

char a = 'A'; // global variable

void msg()

{

printf("Hello\n");

}

[复制代码](javascript:void(0);)

//main.c

int main()

{

extern char a; // extern variable must be declared before use

printf("%c ", a);

(void)msg();

return 0;

}

[复制代码](javascript:void(0);)

       程序的运行结果是：

|  |
| --- |
| A Hello |

      为什么在a.c中定义的全局变量a和函数msg能在main.c中使用？

      前面说过，所有未加static前缀的全局变量和函数都具有全局可见性，其它的源文件也能访问。此例中，a是全局变量，msg是函数，并且都没有加static前缀，

        因此对于另外的源文件main.c是可见的。

     如果加了static，就会对其它源文件隐藏。例如在a和msg的定义前加上static，main.c就看不到它们了。

     利用这一特性可以在不同的文件中定义同名函数和同名变量，而不必担心命名冲突。static可以用作函数和变量的前缀，对于函数来讲，static的作用仅限于隐藏

**12. 在c++程序中调用被C编译器编译后的函数，为什么要加extern“C”**

C++语言支持函数重载，[C语言](http://lib.csdn.net/base/c)不支持函数重载，函数被C++编译器编译后在库中的名字与C语言的不同，

      假设某个函数原型为：

1. void foo(int x, inty);

    该函数被C编译器编译后在库中的名字为:  \_foo

    而C++编译器则会产生像: \_foo\_int\_int   之类的名字。  
    为了解决此类名字匹配的问题，C++提供了C链接交换指定符号 extern "C"。

**13. 头文件种的ifndef/define/endif 是干什么用的**

   防止头文件被重复包含

**14. 线程和进程的联系和区别**

**http://blog.csdn**[**.NET**](http://lib.csdn.net/base/dotnet)**/wolenski/article/details/7969908**

**15. 线程有哪几种状态**

**http://blog.csdn**[**.Net**](http://lib.csdn.net/base/dotnet)**/wolenski/article/details/7969908**

**16. 进程间的通信方式**

管道、有名管道、信号、共享内存、消息队列、信号量、套接字、文件.

**17. 线程同步和线程互斥的区别**

**http://blog.csdn.net/wolenski/article/details/7969908**

**18. 线程同步的方式**

[**Linux**](http://lib.csdn.net/base/linux)**:**互斥锁、条件变量和信号量

**http://blog.csdn.net/zsf8701/article/details/7844316**

**19. 网络七层**

**20. TCP和UDP有什么区别**

TCP---传输控制协议,提供的是面向连接、可靠的字节流服务。

                 当客户和服务器彼此交换数据前，必须先在双方之间建立一个TCP连接，之后才能传输数据。

                 TCP提供超时重发，丢弃重复数据，检验数据，流量控制等功能，保证数据能从一端传到另一端。

     UDP---用户数据报协议，是一个简单的面向数据报的运输层协议。

                 UDP不提供可靠性，它只是把应用程序传给IP层的数据报发送出去，但是并不能保证它们能到达目的地。

                 由于UDP在传输数据报前不用在客户和服务器之间建立一个连接，且没有超时重发等机制，故而传输速度很快

**21. 编写socket套接字的步骤**

**22. TCP三次握手和四次挥手, 以及各个状态的作用**

**http://hi.baidu.com/suxinpingtao51/item/be5f71b3a907dbef4ec7fd0e?qq-pf-to=pcqq.c2c**

**23. HTTP协议**

            http（超文本传输协议）是一个基于请求与响应模式的、无状态的、应用层的协议，常基于TCP的连接方式，

     HTTP1.1版本中给出一种持续连接的机制，绝大多数的Web开发，都是构建在HTTP协议之上的Web应用。

    TCP 和 HTTP区别： http://blog.csdn.net/lemonxuexue/article/details/4485877

**24. 使用过的 shell 命令**

**cp , mv , rm , mkdir , touch , pwd , cd  , ls , top , cat , tail , less , df , du , man , find , kill , sudo , cat**

**25. 使用过的 vim 命令**

wq!, dd , dw , yy , p , i , %s/old/new/g , /abc 向后搜索字符串abc ， ？abc向前搜索字符串abc

**26. 使用过的 gdb 命令**

**http://blog.csdn.net/dadalan/article/details/3758025**

**27. 常见**[**算法**](http://lib.csdn.net/base/datastructure)

  快速排序、堆排序和归并排序

**堆排序 ： http://blog.csdn.net/xiaoxiaoxuewen/article/details/7570621**

**快速排序、归并排序： http://blog.csdn.net/morewindows/article/details/6684558**

稳定性分析 **http://baike.baidu.com/link?url=ueoZ3sNIOvMNPrdCKbd8mhfebC85B4nRc-7hPEJWi-hFo5ROyWH2Pxs9RtvLFRJL**

**28. C库函数实现**

**29. 静态链表和动态链表的区别**

 http://blog.csdn.net/toonny1985/article/details/4868786

**31. 大并发( epoll )**

**优点:**

**http://blog.csdn.net/sunyurun/article/details/8194979**

**实例：**

**http://www.cnblogs.com/ggjucheng/archive/2012/01/17/2324974.html**

**32. 海量数据处理的知识点，（hash表， hash统计）**

hash表： http://hi.baidu.com/05104106/item/62736054402852c09e26679b

    海量数据处理方法： http://blog.csdn.net/v\_july\_v/article/details/7382693

**33. 什么时候要用虚析构函数**

       通过基类的指针来删除派生类的对象时，基类的析构函数应该是虚的。否则其删除效果将无法实现。

       一般情况下，这样的删除只能够删除基类对象，而不能删除子类对象，形成了删除一半形象，从而千万内存泄漏。

      原因：

              在公有继承中，基类对派生类及其对象的操作，只能影响到那些从基类继承下来的成员。

              如果想要用基类对非继承成员进行操作，则要把基类的这个操作（函数）定义为虚函数。  
              那么，析构函数自然也应该如此：如果它想析构子类中的重新定义或新的成员及对象，当然也应该声明为虚的。

      注意：

      如果不需要基类对派生类及对象进行操作，则不能定义虚函数（包括虚析构函数），因为这样会增加内存开销。

**34. c++怎样让返回对象的函数不调用拷贝构造函数**

    拷贝构造函数前加 “explicit” 关键字

**35. 孤儿进程和僵尸进程**

   http://www.cnblogs.com/Anker/p/3271773.html

**各大计算机公司 笔试及面试 题目**

**http://blog.csdn.net/huyfaeng/article/category/880022**

面试题有难有易，不能因为容易，我们就轻视，更不能因为难，我们就放弃。我们面对高薪就业的态度永远不变，那就是坚持、坚持、再坚持。出现问题，找原因；遇到困难，想办法。我们一直坚信只有在坚持中才能看到希望，而不是看到希望才去坚持。

人生没有如果，只有结果和后果。既然选择了，就不后悔。年轻就是资本，年轻就要吃苦，就要历练。就要学会在坚持中成长。如此感慨，至深的心得体会，绝对的经验之谈。

OK，进入正题，下面就是《必须掌握的20道技术面试题》。

**问1：请用简单的语言告诉我C++ 是什么？**

答：C++是在C语言的基础上开发的一种面向对象编程语言，应用广泛。C++支持多种编程范式 －－面向对象编程、泛型编程和过程化编程。 其编程领域众广，常用于系统开发，引擎开发等应用领域，是最受广大程序员受用的最强大编程语言之一,支持类：类、封装、重载等特性!

**问2：C和C++的区别？**

答：c++在c的基础上增添类，C是一个结构化语言，它的重点在于算法和数据结构。C程序的设计首要考虑的是如何通过一个过程，对输入（或环境条件）进行运算处理得到输出（或实现过程（事务）控制），而对于C++，首要考虑的是如何构造一个对象模型，让这个模型能够契合与之对应的问题域，这样就可以通过获取对象的状态信息得到输出或实现过程（事务）控制。

**问3：什么是面向对象（OOP）？**

答：面向对象是一种对现实世界理解和抽象的方法、思想，通过将需求要素转化为对象进行问题处理的一种思想。

**问4：什么是多态？**

答：多态是指相同的操作或函数、过程可作用于多种类型的对象上并获得不同的结果。不同的对象，收到同一消息可以产生不同的结果，这种现象称为多态。

**问5：设计模式懂嘛，简单举个例子？**

答：设计模式（Design pattern）是一套被反复使用、多数人知晓的、经过分类编目的、代码设计经验的总结。

比如单例模式，保证一个类仅有一个实例，并提供一个访问它的全局访问点。

适用于：当类只能有一个实例而且客户可以从一个众所周知的访问点访问它时；当这个唯一实例应该是通过子类化可扩展的，并且客户应该无需更改代码就能使用一个扩展的实例时。

比如工厂模式，定义一个用于创建对象的接口，让子类决定实例化哪一个类。Factory Method 使一个类的实例化延迟到其子类。

适用于：当一个类不知道它所必须创建的对象的类的时候；当一个类希望由它的子类来指定它所创建的对象的时候；当类将创建对象的职责委托给多个帮助子类中的某一个，并且你希望将哪一个帮助子类是代理者这一信息局部化的时候。

**问6：STL库用过吗？常见的STL容器有哪些？算法用过哪几个？**

答：STL包括两部分内容：容器和算法。（重要的还有融合这二者的迭代器）

容器，即存放数据的地方。比如array等。

在STL中，容器分为两类：序列式容器和关联式容器。

序列式容器，其中的元素不一定有序，但都可以被排序。如：vector、list、deque、stack、queue、heap、priority\_queue、slist；

关联式容器，内部结构基本上是一颗平衡二叉树。所谓关联，指每个元素都有一个键值和一个实值，元素按照一定的规则存放。如：RB-tree、set、map、multiset、multimap、hashtable、hash\_set、hash\_map、hash\_multiset、hash\_multimap。

下面各选取一个作为说明。

vector：它是一个动态分配存储空间的容器。区别于c++中的array，array分配的空间是静态的，分配之后不能被改变，而vector会自动重分配（扩展）空间。

set：其内部元素会根据元素的键值自动被排序。区别于map，它的键值就是实值，而map可以同时拥有不同的键值和实值。

算法，如排序，复制……以及个容器特定的算法。这点不用过多介绍，主要看下面迭代器的内容。

迭代器是STL的精髓，我们这样描述它：迭代器提供了一种方法，使它能够按照顺序访问某个容器所含的各个元素，但无需暴露该容器的内部结构。它将容器和算法分开，好让这二者独立设计。

**问7：数据结构会吗？项目开发过程中主要用到那些？**

答：数据结构中主要会用到数组，链表，树（较少），也会用到栈和队列的思想。

**问8：const知道吗？解释其作用。**

答：

1.const 修饰类的成员变量，表示成员常量，不能被修改。

2.const修饰函数承诺在本函数内部不会修改类内的数据成员，不会调用其它非 const 成员函数。

3.如果 const 构成函数重载，const 对象只能调用 const 函数，非 const 对象优先调用非 const 函数。

4.const 函数只能调用 const 函数。非 const 函数可以调用 const 函数。

5.类体外定义的 const 成员函数，在定义和声明处都需要 const 修饰符。。

**问9：类的static变量在什么时候初始化？函数的static变量在什么时候初始化？**

答：类的静态成员变量在类实例化之前就已经存在了，并且分配了内存。函数的static变量在执行此函数时进行初始化。

**问10：堆和栈的区别？堆和栈的生命周期？**

答：

**一、堆栈空间分配区别：**

1、栈（操作系统）：由操作系统自动分配释放 ，存放函数的参数值，局部变量的值等。其操作方式类似于数据结构中的栈；

2、堆（操作系统）： 一般由程序员分配释放， 若程序员不释放，程序结束时可能由OS回收，分配方式倒是类似于链表。

**二、堆栈缓存方式区别：**

1、栈使用的是一级缓存， 他们通常都是被调用时处于存储空间中，调用完毕立即释放；

2、堆是存放在二级缓存中，生命周期由虚拟机的垃圾回收算法来决定（并不是一旦成为孤儿对象就能被回收）。所以调用这些对象的速度要相对来得低一些。

**三、堆栈数据结构区别：**

堆（数据结构）：堆可以被看成是一棵树，如：堆排序；

栈（数据结构）：一种先进后出的数据结构。

**问11：C和C++的区别？**

答：C++在C的基础上增添类

C是一个结构化语言，它的重点在于算法和数据结构。

C程序的设计首要考虑的是如何通过一个过程，对输入（或环境条件）进行运算处理得到输出（或实现过程（事务）控制），而对于C++，首要考虑的是如何构造一个对象模型，让这个模型能够契合与之对应的问题域，这样就可以通过获取对象的状态信息得到输出或实现过程（事务）控制。

**问12：解释下封装、继承和多态？**

答：

**一、封装：**

封装是实现面向对象程序设计的第一步，封装就是将数据或函数等集合在一个个的单元中（我们称之为类）。

封装的意义在于保护或者防止代码（数据）被我们无意中破坏。

**二、继承：**

继承主要实现重用代码，节省开发时间。

子类可以继承父类的一些东西。

**三、多态**

多态：同一操作作用于不同的对象，可以有不同的解释，产生不同的执行结果。在运行时，可以通过指向基类的指针，来调用实现派生类中的方法。

**问13：指针和引用的区别？**

答：

1. 指针是一个变量，只不过这个变量存储的是一个地址，指向内存的一个存储单元；而引用仅是个别名；

2. 引用使用时无需解引用(\*)，指针需要解引用；

3. 引用只能在定义时被初始化一次，之后不可变；指针可变；

4. 引用没有 const，指针有 const；

5. 引用不能为空，指针可以为空；

6. “sizeof 引用”得到的是所指向的变量(对象)的大小，而“sizeof 指针”得到的是指针本身的大小；

7. 指针和引用的自增(++)运算意义不一样；

8. 指针可以有多级，但是引用只能是一级（int \*\*p；合法 而 int &&a是不合法的）

9.从内存分配上看：程序为指针变量分配内存区域，而引用不需要分配内存区域。

**问14：什么是内存泄漏？面对内存泄漏和指针越界，你有哪些方法？你通常采用哪些方法来避免和减少这类错误？**

答：用动态存储分配函数动态开辟的空间，在使用完毕后未释放，结果导致一直占据该内存单元即为内存泄露。

使用的时候要记得指针的长度。

malloc的时候得确定在那里free.

对指针赋值的时候应该注意被赋值指针需要不需要释放.

动态分配内存的指针最好不要再次赋值.

**问15：常用的排序算法有哪些？简单描述几个排序算法的优缺点？**

答：选择、冒泡、快速、\*\*、希尔、归并、堆排等。

1.快排：是冒泡排序的一种改进。

优点：快，数据移动少

缺点：稳定性不足

2.归并：分治法排序，稳定的排序算法，一般用于对总体无序，但局部有序的数列。

优点：效率高O(n)，稳定

缺点：比较占用内存

**问16：new和malloc的区别？**

答：

1、malloc与free是C++/C语言的标准库函数，new/delete是C++的运算符。它们都可用于申请动态内存和释放内存。

2、对于非内部数据类型的对象而言，光用maloc/free无法满足动态对象的要求。对象在创建的同时要自动执行构造函数，对象在消亡之前要自动执行析构函数。

3、由于malloc/free是库函数而不是运算符，不在编译器控制权限之内，不能够把执行构造函数和析构函数的任务强加于malloc/free。因此C++语言需要一个能完成动态内存分配和初始化工作的运算符new，以一个能完成清理与释放内存工作的运算符delete。注意new/delete不是库函数。

4、C++程序经常要调用C函数，而C程序只能用malloc/free管理动态内存。

5、new可以认为是malloc加构造函数的执行。new出来的指针是直接带类型信息的。而malloc返回的都是void指针。

**问17：TCP和UDP通信的差别？什么是IOCP？**

答：

1.TCP面向连接， UDP面向无连接的

2.TCP有保障的，UDP传输无保障的

3.TCP是效率低的，UDP效率高的

4.TCP是基于流的，UDP基于数据报文

5.TCP传输重要数据，UDP传输不重要的数据

IOCP全称I/O Completion Port，中文译为I/O完成端口。

IOCP是一个异步I/O的API，它可以高效地将I/O事件通知给应用程序。

与使用select()或是其它异步方法不同的是，一个套接字[socket]与一个完成端口关联了起来，然后就可继续进行正常的Winsock操作了。然而，当一个事件发生的时候，此完成端口就将被操作系统加入一个队列中。然后应用程序可以对核心层进行查询以得到此完成端口。

**问18：同步IO和异步IO的区别？**

答：

**A. 同步**

所谓同步，就是在发出一个功能调用时，在没有得到结果之前，该调用就不返回。

按照这个定义，其实绝大多数函数都是同步调用（例如sin isdigit等）。

但是一般而言，我们在说同步、异步的时候，特指那些需要其他部件协作或者需要一定时间完成的任务。

最常见的例子就是 SendMessage。

该函数发送一个消息给某个窗口，在对方处理完消息之前，这个函数不返回。

当对方处理完毕以后，该函数才把消息处理函数所返回的值返回给调用者。

**B. 异步**

异步的概念和同步相对。

当一个异步过程调用发出后，调用者不会立刻得到结果。

实际处理这个调用的部件是在调用发出后，通过状态、通知来通知调用者，或通过回调函数处理这个调用。

**问19：解释C++中静态函数和静态变量？**

答：

(1)类静态数据成员在编译时创建并初始化：在该类的任何对象建立之前就存在，不属于任何对象，而非静态类成员变量则是属于对象所有的。类静态数据成员只有一个拷贝，为所有此类的对象所共享。

(2)类静态成员函数属于整个类，不属于某个对象，由该类所有对象共享。

1、static 成员变量实现了同类对象间信息共享。

2、static 成员类外存储，求类大小，并不包含在内。

3、static 成员是命名空间属于类的全局变量，存储在 data 区的rw段。

4、static 成员只能类外初始化。

5、可以通过类名访问（无对象生成时亦可），也可以通过对象访问。

1、静态成员函数的意义，不在于信息共享，数据沟通，而在于管理静态数据成员，完成对静态数据成员的封装。

2、静态成员函数只能访问静态数据成员。原因：非静态成员函数，在调用时 this指针时被当作参数传进。而静态成员函数属于类，而不属于对象，没有 this 指针。

**问20：说下你对内存的了解？**

答：

1.栈 - 由编译器自动分配释放

2.堆 - 一般由程序员分配释放，若程序员不释放，程序结束时可能由OS回收

3.全局区(静态区)，全局变量和静态变量的存储是放在一块的，初始化的全局变量和静态变量在一块区域，未初始化的全局变量和未初始化的静态变量在相邻的另一块区域。- 程序结束释放

4.另外还有一个专门放常量的地方。- 程序结束释放

5 程序代码区，存放2进制代码。

在函数体中定义的变量通常是在栈上，用malloc， calloc， realloc等分配内存的函数分配得到的就是在堆上。在所有函数体外定义的是全局量，加了static修饰符后不管在哪里都存放在全局区(静态区)，在所有函数体外定义的static变量表示在该文件中有效，不能extern到别的文件用，在函数体内定义的static表示只在该函数体内有效。另外，函数中的"adgfdf"这样的字符串存放在常量区。