**C++面试宝典基础知识汇总**

**（GDTDH内部资料，谢绝外传）**

**目录**

[一、C++基础知识 5](#_Toc12735342)

[1、static关键字的作用 5](#_Toc12735343)

[2、C++和C的区别 6](#_Toc12735344)

[3、C++中四种cast转换 6](#_Toc12735345)

[4、C/C++ 中指针和引用的区别？ 6](#_Toc12735346)

[5、数组和指针的区别 7](#_Toc12735347)

[6、野指针是什么？ 7](#_Toc12735348)

[7、C++中的智能指针 7](#_Toc12735349)

[8、智能指针的内存泄漏如何解决 7](#_Toc12735350)

[9、为什么析构函数必须是虚函数？为什么C++默认的析构函数不是虚函数 8](#_Toc12735351)

[10、函数指针 8](#_Toc12735352)

[11、fork函数 8](#_Toc12735353)

[12、C++中析构函数的作用 8](#_Toc12735354)

[13、静态函数和虚函数的区别 9](#_Toc12735355)

[14、重载和覆盖 9](#_Toc12735356)

[15、static关键字 9](#_Toc12735357)

[16、strcpy和strlen 9](#_Toc12735358)

[17、虚函数和多态 9](#_Toc12735359)

[18、C++里是怎么定义常量的？常量存放在内存的哪个位置？ 9](#_Toc12735360)

[19、const修饰成员函数的目的是什么？ 10](#_Toc12735361)

[20、说一说隐式类型转换 10](#_Toc12735362)

[21、说一说extern“C” 10](#_Toc12735363)

[22、new/delete与malloc/free的区别是什么 10](#_Toc12735364)

[23、虚函数表具体是怎样实现运行时多态的? 10](#_Toc12735365)

[24、C语言是怎么进行函数调用的？ 10](#_Toc12735366)

[25、C++中拷贝赋值函数的形参能否进行值传递？ 10](#_Toc12735367)

[26、说说fork,wait,exec函数 10](#_Toc12735368)

[二、容器和算法 11](#_Toc12735369)

[1、map和set有什么区别，分别又是怎么实现的？ 11](#_Toc12735370)

[2、介绍一下STL的allocaotr 11](#_Toc12735371)

[3、说一说STL迭代器删除元素 11](#_Toc12735372)

[4、STL有什么基本组成 12](#_Toc12735373)

[5、STL中map与unordered\_map 12](#_Toc12735374)

[6、说一说vector和list的区别，应用 12](#_Toc12735375)

[7、STL中迭代器的作用，有指针为何还要迭代器 13](#_Toc12735376)

[8、STL里resize和reserve的区别 14](#_Toc12735377)

[9、C++中类成员的访问权限 14](#_Toc12735378)

[10、C++中struct和class的区别 14](#_Toc12735379)

[11、C++类内可以定义引用数据成员吗？ 14](#_Toc12735380)

[12、什么是右值引用，跟左值又有什么区别？ 14](#_Toc12735381)

[13、一个C++源文件从文本到可执行文件经历的过程？ 15](#_Toc12735382)

[14、include头文件的顺序以及双引号””和尖括号<>的区别？ 15](#_Toc12735383)

[15、C++的内存管理是怎样的？ 15](#_Toc12735384)

[16、如何判断内存泄漏？ 16](#_Toc12735385)

[17、什么时候会发生段错误 16](#_Toc12735386)

[18、什么是memory leak，也就是内存泄漏 16](#_Toc12735387)

[19、C++11有哪些新特性？ 16](#_Toc12735388)

[三、操作系统 18](#_Toc12735389)

[1、进程与线程的概念，为什么要有进程线程，有什么区别，各自是怎么同步的 18](#_Toc12735390)

[2、说一说Linux虚拟地址空间 20](#_Toc12735391)

[3、说一说操作系统中的缺页中断 21](#_Toc12735392)

[4、fork和vfork的区别 21](#_Toc12735393)

[5、说一说并发(concurrency)和并行(parallelism) 22](#_Toc12735394)

[6、MySQL的端口号是多少，如何修改这个端口号 22](#_Toc12735395)

[7、说一说操作系统中的页表寻址 22](#_Toc12735396)

[8、线程需要保存哪些上下文，SP、PC、EAX这些寄存器是？ 22](#_Toc12735397)

[9、进程和线程的区别 23](#_Toc12735398)

[10、说一说OS缺页置换算法 23](#_Toc12735399)

[11、多进程和多线程的使用场景 23](#_Toc12735400)

[12、死锁发生的条件以及如何解决死锁 23](#_Toc12735401)

[13、讲述一下互斥锁（mutex）机制，以及互斥锁和读写锁的区别 24](#_Toc12735402)

[14、说一说进程状态转换图，动态就绪，静态就绪，动态阻塞，静态阻塞 25](#_Toc12735403)

[15、静态变量什么时候初始化 25](#_Toc12735404)

[16、用户态和内核态区别 25](#_Toc12735405)

[17、多线程的同步，锁的机制 25](#_Toc12735406)

[18、windows消息机制知道吗，请说一说 26](#_Toc12735407)

[19、说一说协程 26](#_Toc12735408)

[20、系统调用是什么，你用过哪些系统调用 26](#_Toc12735409)

[21、用户态到内核态的转化原理 27](#_Toc12735410)

[22、介绍一下5种IO模型 28](#_Toc12735411)

[23、操作系统为什么要分内核态和用户态 28](#_Toc12735412)

[24、如何设计server，使得能够接收多个客户端的请求 28](#_Toc12735413)

[25、怎么实现线程池 28](#_Toc12735414)

[四、计算机网络 29](#_Toc12735415)

[1、说一下TCP怎么保证可靠性，并且简述一下TCP建立连接和断开连接的过程 29](#_Toc12735416)

[2、三次握手和四次挥手 29](#_Toc12735417)

[3、HTTP和HTTPS的区别，以及HTTPS有什么缺点？ 30](#_Toc12735418)

[4、HTTP返回码 31](#_Toc12735419)

[5、说一说IP地址作用，以及MAC地址作用 31](#_Toc12735420)

[6、介绍一下操作系统中的中断 31](#_Toc12735421)

[7、回答OSI七层模型和TCP/IP四层模型，每层列举2个协议 32](#_Toc12735422)

[8、搜索baidu，会用到计算机网络中的什么层？每层是干什么的 32](#_Toc12735423)

[9、说一说TCP拥塞控制？以及达到什么情况的时候开始减慢增长的速度？ 33](#_Toc12735424)

[10、TCP用了哪些措施保证其可靠性 34](#_Toc12735425)

[11、传递到IP层怎么知道报文给哪个应用程序，怎么区分UDP报文和TCP报文 35](#_Toc12735426)

[12、TCP和UDP的区别和各自适用的场景 35](#_Toc12735427)

[13、说一下数字证书是什么，里面都包含那些内容 36](#_Toc12735428)

[14、讲述一下Socket编程的send() recv() accept() socket()函数？ 36](#_Toc12735429)

[15、说一说TCP/IP数据链路层的交互过程 37](#_Toc12735430)

[五、数据库 38](#_Toc12735431)

[1、数据库索引 38](#_Toc12735432)

[2、数据库事务 38](#_Toc12735433)

[3、说一说数据库事务隔离 38](#_Toc12735434)

[4、说一说inner join和left join 38](#_Toc12735435)

[5、说一下数据库事务以及四个特性 38](#_Toc12735436)

[6、数据库的三大范式 39](#_Toc12735437)

[7、介绍一下mysql的MVCC机制 39](#_Toc12735438)

[8、SQL优化方法有哪些 39](#_Toc12735439)

[9、说一下MySQL引擎和区别 39](#_Toc12735440)

[10、mongodb和redis的区别 40](#_Toc12735441)

[11、Redis是单线程的，但是为什么这么高效呢? 40](#_Toc12735442)

[12、Redis的数据类型有哪些，底层怎么实现? 40](#_Toc12735443)

[13、Redis和memcached的区别 40](#_Toc12735444)

[六、数据结构与算法 42](#_Toc12735445)

[1、红黑树和AVL树的定义，特点，以及二者区别 42](#_Toc12735446)

[2、哈夫曼编码 42](#_Toc12735447)

[3、map底层为什么用红黑树实现 42](#_Toc12735448)

[4、请你介绍一下B+树 43](#_Toc12735449)

[5、epoll怎么实现的 44](#_Toc12735450)

[6、栈和堆的区别，以及为什么栈要快 44](#_Toc12735451)

[7、小根堆特点 44](#_Toc12735452)

[8、Array&List， 数组和链表的区别 44](#_Toc12735453)

[9、各个排序的时间复杂度、空间复杂度及稳定性如下 45](#_Toc12735454)

[10、说一说hash表的实现，包括STL中的哈希桶长度常数 45](#_Toc12735455)

[11、回答一下hash表如何rehash，以及怎么处理其中保存的资源 46](#_Toc12735456)

[12、解决hash冲突的方法 46](#_Toc12735457)

[13、链表翻转 46](#_Toc12735458)

# 一、C++基础知识

## 1、static关键字的作用

1. 全局静态变量

在全局变量前加上关键字static，全局变量就定义成一个全局静态变量.

静态存储区，在整个程序运行期间一直存在。

初始化：未经初始化的全局静态变量会被自动初始化为0（自动对象的值是任意的，除非他被显式初始化）；

作用域：全局静态变量在声明他的文件之外是不可见的，准确地说是从定义之处开始，到文件结尾。

2. 局部静态变量

在局部变量之前加上关键字static，局部变量就成为一个局部静态变量。

内存中的位置：静态存储区

初始化：未经初始化的全局静态变量会被自动初始化为0（自动对象的值是任意的，除非他被显式初始化）；

作用域：作用域仍为局部作用域，当定义它的函数或者语句块结束的时候，作用域结束。但是当局部静态变量离开作用域后，并没有销毁，而是仍然驻留在内存当中，只不过我们不能再对它进行访问，直到该函数再次被调用，并且值不变；

3. 静态函数

在函数返回类型前加static，函数就定义为静态函数。函数的定义和声明在默认情况下都是extern的，但静态函数只是在声明他的文件当中可见，不能被其他文件所用。

函数的实现使用static修饰，那么这个函数只可在本cpp内使用，不会同其他cpp中的同名函数引起冲突；

warning：不要再头文件中声明static的全局函数，不要在cpp内声明非static的全局函数，如果你要在多个cpp中复用该函数，就把它的声明提到头文件里去，否则cpp内部声明需加上static修饰；

4. 类的静态成员

在类中，静态成员可以实现多个对象之间的数据共享，并且使用静态数据成员还不会破坏隐藏的原则，即保证了安全性。因此，静态成员是类的所有对象中共享的成员，而不是某个对象的成员。对多个对象来说，静态数据成员只存储一处，供所有对象共用

5. 类的静态函数

静态成员函数和静态数据成员一样，它们都属于类的静态成员，它们都不是对象成员。因此，对静态成员的引用不需要用对象名。

在静态成员函数的实现中不能直接引用类中说明的非静态成员，可以引用类中说明的静态成员（这点非常重要）。如果静态成员函数中要引用非静态成员时，可通过对象来引用。从中可看出，调用静态成员函数使用如下格式：<类名>::<静态成员函数名>(<参数表>);

## 2、C++和C的区别

设计思想上：C++是面向对象的语言，而C是面向过程的结构化编程语言

语法上：C++具有封装、继承和多态三种特性；C++相比C，增加多许多类型安全的功能，比如强制类型转换；C++支持范式编程，比如模板类、函数模板等。

## 3、C++中四种cast转换

C++中四种类型转换是：static\_cast, dynamic\_cast, const\_cast, reinterpret\_cast

1、const\_cast

用于将const变量转为非const

2、static\_cast

用于各种隐式转换，比如非const转const，void\*转指针等, static\_cast能用于多态向上转化，如果向下转能成功但是不安全，结果未知；

3、dynamic\_cast

用于动态类型转换。只能用于含有虚函数的类，用于类层次间的向上和向下转化。只能转指针或引用。向下转化时，如果是非法的对于指针返回NULL，对于引用抛异常。要深入了解内部转换的原理。

向上转换：指的是子类向基类的转换

向下转换：指的是基类向子类的转换

它通过判断在执行到该语句的时候变量的运行时类型和要转换的类型是否相同来判断是否能够进行向下转换。

4、reinterpret\_cast

几乎什么都可以转，比如将int转指针，可能会出问题，尽量少用；

5、为什么不使用C的强制转换？

C的强制转换表面上看起来功能强大什么都能转，但是转化不够明确，不能进行错误检查，容易出错。

## 4、C/C++ 中指针和引用的区别？

**1、引用：**

C++是C语言的继承，它可进行过程化程序设计，又可以进行以抽象数据类型为特点的基于对象的程序设计，还可以进行以继承和多态为特点的面向对象的程序设计。引用就是C++对C语言的重要扩充。引用就是某一变量的一个别名，对引用的操作与对变量直接操作完全一样。引用的声明方法：类型标识符 &引用名=目标变量名；引用引入了对象的一个同义词。定义引用的表示方法与定义[**指针**](https://baike.baidu.com/item/%E6%8C%87%E9%92%88/2878304)相似，只是用&代替了\*。

**2、指针：**

指针利用地址，它的值直接指向存在电脑存储器中另一个地方的值。由于通过地址能找到所需的变量单元，可以说，地址指向该变量单元。因此，将地址形象化的称为“指针”。意思是通过它能找到以它为地址的内存单元。

1.指针有自己的一块空间，而引用只是一个别名；

2.使用sizeof看一个指针的大小是4，而引用则是被引用对象的大小；

3.指针可以被初始化为NULL，而引用必须被初始化且必须是一个已有对象 的引用；

4.作为参数传递时，指针需要被解引用才可以对对象进行操作，而直接对引 用的修改都会改变引用所指向的对象；

5.可以有const指针，但是没有const引用；

6.指针在使用中可以指向其它对象，但是引用只能是一个对象的引用，不能 被改变；

7.指针可以有多级指针（\*\*p），而引用至于一级；

8.指针和引用使用++运算符的意义不一样；

9.如果返回动态内存分配的对象或者内存，必须使用指针，引用可能引起内存泄露。

## 5、数组和指针的区别



## 6、野指针是什么？

野指针就是指向一个已删除的对象或者未申请访问受限内存区域的指针。

## 7、C++中的智能指针

智能指针主要用于管理在堆上分配的内存，它将普通的指针封装为一个栈对象。当栈对象的生存周期结束后，会在析构函数中释放掉申请的内存，从而防止内存泄漏。C++ 11中最常用的智能指针类型为shared\_ptr,它采用引用计数的方法，记录当前内存资源被多少个智能指针引用。该引用计数的内存在堆上分配。当新增一个时引用计数加1，当过期时引用计数减一。只有引用计数为0时，智能指针才会自动释放引用的内存资源。对shared\_ptr进行初始化时不能将一个普通指针直接赋值给智能指针，因为一个是指针，一个是类。可以通过make\_shared函数或者通过构造函数传入普通指针。并可以通过get函数获得普通指针。

## 8、智能指针的内存泄漏如何解决

为了解决循环引用导致的内存泄漏，引入了weak\_ptr弱指针，weak\_ptr的构造函数不会修改引用计数的值，从而不会对对象的内存进行管理，其类似一个普通指针，但不指向引用计数的共享内存，但是其可以检测到所管理的对象是否已经被释放，从而避免非法访问。

## 9、为什么析构函数必须是虚函数？为什么C++默认的析构函数不是虚函数

将可能会被继承的父类的析构函数设置为虚函数，可以保证当我们new一个子类，然后使用基类指针指向该子类对象，释放基类指针时可以释放掉子类的空间，防止内存泄漏。

C++默认的析构函数不是虚函数是因为虚函数需要额外的虚函数表和虚表指针，占用额外的内存。而对于不会被继承的类来说，其析构函数如果是虚函数，就会浪费内存。因此C++默认的析构函数不是虚函数，而是只有当需要当作父类时，设置为虚函数。

## 10、函数指针

1、定义

函数指针是指向函数的指针变量。

函数指针本身首先是一个指针变量，该指针变量指向一个具体的函数。这正如用指针变量可指向整型变量、字符型、数组一样，这里是指向函数。

C在编译时，每一个函数都有一个入口地址，该入口地址就是函数指针所指向的地址。有了指向函数的指针变量后，可用该指针变量调用函数，就如同用指针变量可引用其他类型变量一样，在这些概念上是大体一致的。

2、用途：

调用函数和做函数的参数，比如回调函数。

3、示例：

char \* fun(char \* p)  {…}       // 函数fun

char \* (\*pf)(char \* p);             // 函数指针pf

pf = fun;                        // 函数指针pf指向函数fun

pf(p);                        // 通过函数指针pf调用函数fun

## 11、fork函数

Fork：创建一个和当前进程映像一样的进程可以通过fork( )系统调用：

#include <sys/types.h>

#include <unistd.h>

pid\_t fork(void);

成功调用fork( )会创建一个新的进程，它几乎与调用fork( )的进程一模一样，这两个进程都会继续运行。在子进程中，成功的fork( )调用会返回0。在父进程中fork( )返回子进程的pid。如果出现错误，fork( )返回一个负值。

最常见的fork( )用法是创建一个新的进程，然后使用exec( )载入二进制映像，替换当前进程的映像。这种情况下，派生（fork）了新的进程，而这个子进程会执行一个新的二进制可执行文件的映像。这种“派生加执行”的方式是很常见的。

在早期的Unix系统中，创建进程比较原始。当调用fork时，内核会把所有的内部数据结构复制一份，复制进程的页表项，然后把父进程的地址空间中的内容逐页的复制到子进程的地址空间中。但从内核角度来说，逐页的复制方式是十分耗时的。现代的Unix系统采取了更多的优化，例如Linux，采用了写时复制的方法，而不是对父进程空间进程整体复制。

## 12、C++中析构函数的作用

析构函数与构造函数对应，当对象结束其生命周期，如对象所在的函数已调用完毕时，系统会自动执行析构函数。

析构函数名也应与类名相同，只是在函数名前面加一个位取反符~，例如~stud( )，以区别于构造函数。它不能带任何参数，也没有返回值（包括void类型）。只能有一个析构函数，不能重载。

如果用户没有编写析构函数，编译系统会自动生成一个缺省的析构函数（即使自定义了析构函数，编译器也总是会为我们合成一个析构函数，并且如果自定义了析构函数，编译器在执行时会先调用自定义的析构函数再调用合成的析构函数），它也不进行任何操作。所以许多简单的类中没有用显式的析构函数。

如果一个类中有指针，且在使用的过程中动态的申请了内存，那么最好显示构造析构函数在销毁类之前，释放掉申请的内存空间，避免内存泄漏。

类析构顺序：1）派生类本身的析构函数；2）对象成员析构函数；3）基类析构函数。

## 13、静态函数和虚函数的区别

静态函数在编译的时候就已经确定运行时机，虚函数在运行的时候动态绑定。虚函数因为用了虚函数表机制，调用的时候会增加一次内存开销

## 14、重载和覆盖

重载：两个函数名相同，但是参数列表不同（个数，类型），返回值类型没有要求，在同一作用域中  
 重写：子类继承了父类，父类中的函数是虚函数，在子类中重新定义了这个虚函数，这种情况是重写

## 15、static关键字

1.加了static关键字的全局变量只能在本文件中使用。例如在a.c中定义了static int a=10;那么在b.c中用extern int a是拿不到a的值得，a的作用域只在a.c中。

2.static定义的静态局部变量分配在数据段上，普通的局部变量分配在栈上，会因为函数栈帧的释放而被释放掉。  
 3. 对一个类中成员变量和成员函数来说，加了static关键字，则此变量/函数就没有了this指针了，必须通过类名才能访问

## 16、strcpy和strlen

strcpy是字符串拷贝函数，原型：char \*strcpy(char\* dest, const char \*src);从src逐字节拷贝到dest，直到遇到'\0'结束，因为没有指定长度，可能会导致拷贝越界，造成缓冲区溢出漏洞,安全版本是strncpy函数。  
 strlen函数是计算字符串长度的函数，返回从开始到'\0'之间的字符个数。

## 17、虚函数和多态

多态的实现主要分为静态多态和动态多态，静态多态主要是重载，在编译的时候就已经确定；动态多态是用虚函数机制实现的，在运行期间动态绑定。举个例子：一个父类类型的指针指向一个子类对象时候，使用父类的指针去调用子类中重写了的父类中的虚函数的时候，会调用子类重写过后的函数，在父类中声明为加了virtual关键字的函数，在子类中重写时候不需要加virtual也是虚函数。  
 虚函数的实现：在有虚函数的类中，类的最开始部分是一个虚函数表的指针，这个指针指向一个虚函数表，表中放了虚函数的地址，实际的虚函数在代码段(.text)中。当子类继承了父类的时候也会继承其虚函数表，当子类重写父类中虚函数时候，会将其继承到的虚函数表中的地址替换为重新写的函数地址。使用了虚函数，会增加访问内存开销，降低效率。

## 18、C++里是怎么定义常量的？常量存放在内存的哪个位置？

常量在C++里的定义就是一个top-level const加上对象类型，常量定义必须初始化。对于局部对象，常量存放在栈区，对于全局对象，常量存放在全局/静态存储区。对于字面值常量，常量存放在常量存储区。

## 19、const修饰成员函数的目的是什么？

const修饰的成员函数表明函数调用不会对对象做出任何更改，事实上，如果确认不会对对象做更改，就应该为函数加上const限定，这样无论const对象还是普通对象都可以调用该函数。

## 20、说一说隐式类型转换

首先，对于内置类型，低精度的变量给高精度变量赋值会发生隐式类型转换，其次，对于只存在单个参数的构造函数的对象构造来说，函数调用可以直接使用该参数传入，编译器会自动调用其构造函数生成临时对象。

## 21、说一说extern“C”

C++调用C函数需要extern C，因为C语言没有函数重载。

## 22、new/delete与malloc/free的区别是什么

首先，new/delete是C++的关键字，而malloc/free是C语言的库函数，后者使用必须指明申请内存空间的大小，对于类类型的对象，后者不会调用构造函数和析构函数。

malloc需要给定申请内存的大小，返回的指针需要强转。

new会调用构造函数，不用指定内存大小，返回的指针不用强转。

## 23、虚函数表具体是怎样实现运行时多态的?

子类若重写父类虚函数，虚函数表中，该函数的地址会被替换，对于存在虚函数的类的对象，在VS中，对象的对象模型的头部存放指向虚函数表的指针，通过该机制实现多态。

## 24、C语言是怎么进行函数调用的？

每一个函数调用都会分配函数栈，在栈内进行函数执行过程。调用前，先把返回地址压栈，然后把当前函数的esp指针压栈。

## 25、C++中拷贝赋值函数的形参能否进行值传递？

不能。如果是这种情况下，调用拷贝构造函数的时候，首先要将实参传递给形参，这个传递的时候又要调用拷贝构造函数。。如此循环，无法完成拷贝，栈也会满。

## 26、说说fork,wait,exec函数

父进程产生子进程使用fork拷贝出来一个父进程的副本，此时只拷贝了父进程的页表，两个进程都读同一块内存，当有进程写的时候使用写实拷贝机制分配内存，exec函数可以加载一个elf文件去替换父进程，从此父进程和子进程就可以运行不同的程序了。fork从父进程返回子进程的pid，从子进程返回0.调用了wait的父进程将会发生阻塞，直到有子进程状态改变,执行成功返回0，错误返回-1。exec执行成功则子进程从新的程序开始运行，无返回值，执行失败返回-1。

# 二、容器和算法

## 1、map和set有什么区别，分别又是怎么实现的？

map和set都是C++的关联容器，其底层实现都是红黑树（RB-Tree）。由于 map 和set所开放的各种操作接口，RB-tree 也都提供了，所以几乎所有的 map 和set的操作行为，都只是转调 RB-tree 的操作行为。

map和set区别在于：

（1）map中的元素是key-value（关键字—值）对：关键字起到索引的作用，值则表示与索引相关联的数据；Set与之相对就是关键字的简单集合，set中每个元素只包含一个关键字。

（2）set的迭代器是const的，不允许修改元素的值；map允许修改value，但不允许修改key。其原因是因为map和set是根据关键字排序来保证其有序性的，如果允许修改key的话，那么首先需要删除该键，然后调节平衡，再插入修改后的键值，调节平衡，如此一来，严重破坏了map和set的结构，导致iterator失效，不知道应该指向改变前的位置，还是指向改变后的位置。所以STL中将set的迭代器设置成const，不允许修改迭代器的值；而map的迭代器则不允许修改key值，允许修改value值。

（3）map支持下标操作，set不支持下标操作。map可以用key做下标，map的下标运算符[ ]将关键码作为下标去执行查找，如果关键码不存在，则插入一个具有该关键码和mapped\_type类型默认值的元素至map中，因此下标运算符[ ]在map应用中需要慎用，const\_map不能用，只希望确定某一个关键值是否存在而不希望插入元素时也不应该使用，mapped\_type类型没有默认值也不应该使用。如果find能解决需要，尽可能用find。

## 2、介绍一下STL的allocaotr

STL的分配器用于封装STL容器在内存管理上的底层细节。在C++中，其内存配置和释放如下：

new运算分两个阶段：(1)调用::operator new配置内存;(2)调用对象构造函数构造对象内容

delete运算分两个阶段：(1)调用对象希构函数；(2)掉员工::operator delete释放内存

为了精密分工，STL allocator将两个阶段操作区分开来：内存配置有alloc::allocate()负责，内存释放由alloc::deallocate()负责；对象构造由::construct()负责，对象析构由::destroy()负责。

同时为了提升内存管理的效率，减少申请小内存造成的内存碎片问题，SGI STL采用了两级配置器，当分配的空间大小超过128B时，会使用第一级空间配置器；当分配的空间大小小于128B时，将使用第二级空间配置器。第一级空间配置器直接使用malloc()、realloc()、free()函数进行内存空间的分配和释放，而第二级空间配置器采用了内存池技术，通过空闲链表来管理内存。

## 3、说一说STL迭代器删除元素

这个主要考察的是迭代器失效的问题。

1.对于序列容器vector,deque来说，使用erase(itertor)后，后边的每个元素的迭代器都会失效，但是后边每个元素都会往前移动一个位置，但是erase会返回下一个有效的迭代器；

2.对于关联容器map set来说，使用了erase(iterator)后，当前元素的迭代器失效，但是其结构是红黑树，删除当前元素的，不会影响到下一个元素的迭代器，所以在调用erase之前，记录下一个元素的迭代器即可。

3.对于list来说，它使用了不连续分配的内存，并且它的erase方法也会返回下一个有效的iterator，因此上面两种正确的方法都可以使用。

## 4、STL有什么基本组成

STL主要由：以下几部分组成：  
 容器、迭代器、仿函数、算法、分配器、配接器；他们之间的关系：分配器给容器分配存储空间，算法通过迭代器获取容器中的内容，仿函数可以协助算法完成各种操作，配接器用来套接适配仿函数。

## 5、STL中map与unordered\_map

1、Map映射，map 的所有元素都是 pair，同时拥有实值（value）和键值（key）。pair 的第一元素被视为键值，第二元素被视为实值。所有元素都会根据元素的键值自动被排序。不允许键值重复。

底层实现：红黑树

适用场景：有序键值对不重复映射

2、Multimap

多重映射。multimap 的所有元素都是 pair，同时拥有实值（value）和键值（key）。pair 的第一元素被视为键值，第二元素被视为实值。所有元素都会根据元素的键值自动被排序。允许键值重复。

底层实现：红黑树

适用场景：有序键值对可重复映射

## 6、说一说vector和list的区别，应用

**参考回答：**

1、概念：

1）Vector

连续存储的容器，动态数组，在堆上分配空间

底层实现：数组

两倍容量增长：

vector 增加（插入）新元素时，如果未超过当时的容量，则还有剩余空间，那么直接添加到最后（插入指定位置），然后调整迭代器。

如果没有剩余空间了，则会重新配置原有元素个数的两倍空间，然后将原空间元素通过复制的方式初始化新空间，再向新空间增加元素，最后析构并释放原空间，之前的迭代器会失效。

性能：

访问：O(1)

插入：在最后插入（空间够）：很快

在最后插入（空间不够）：需要内存申请和释放，以及对之前数据进行拷贝。

在中间插入（空间够）：内存拷贝

在中间插入（空间不够）：需要内存申请和释放，以及对之前数据进行拷贝。

删除：在最后删除：很快

在中间删除：内存拷贝

适用场景：经常随机访问，且不经常对非尾节点进行插入删除。

2、List

动态链表，在堆上分配空间，每插入一个元数都会分配空间，每删除一个元素都会释放空间。

底层：双向链表

性能：

访问：随机访问性能很差，只能快速访问头尾节点。

插入：很快，一般是常数开销

删除：很快，一般是常数开销

适用场景：经常插入删除大量数据

2、区别：

1）vector底层实现是数组；list是双向 链表。

2）vector支持随机访问，list不支持。

3）vector是顺序内存，list不是。

4）vector在中间节点进行插入删除会导致内存拷贝，list不会。

5）vector一次性分配好内存，不够时才进行2倍扩容；list每次插入新节点都会进行内存申请。

6）vector随机访问性能好，插入删除性能差；list随机访问性能差，插入删除性能好。

3、应用

vector拥有一段连续的内存空间，因此支持随机访问，如果需要高效的随即访问，而不在乎插入和删除的效率，使用vector。

list拥有一段不连续的内存空间，如果需要高效的插入和删除，而不关心随机访问，则应使用list。

## 7、STL中迭代器的作用，有指针为何还要迭代器

1、迭代器

Iterator（迭代器）模式又称Cursor（游标）模式，用于提供一种方法顺序访问一个聚合对象中各个元素, 而又不需暴露该对象的内部表示。或者这样说可能更容易理解：Iterator模式是运用于聚合对象的一种模式，通过运用该模式，使得我们可以在不知道对象内部表示的情况下，按照一定顺序（由iterator提供的方法）访问聚合对象中的各个元素。

由于Iterator模式的以上特性：与聚合对象耦合，在一定程度上限制了它的广泛运用，一般仅用于底层聚合支持类，如STL的list、vector、stack等容器类及ostream\_iterator等扩展iterator。

2、迭代器和指针的区别

迭代器不是指针，是类模板，表现的像指针。他只是模拟了指针的一些功能，通过重载了指针的一些操作符，->、\*、++、--等。迭代器封装了指针，是一个“可遍历STL（ Standard Template Library）容器内全部或部分元素”的对象， 本质是封装了原生指针，是指针概念的一种提升（lift），提供了比指针更高级的行为，相当于一种智能指针，他可以根据不同类型的数据结构来实现不同的++，--等操作。

迭代器返回的是对象引用而不是对象的值，所以cout只能输出迭代器使用\*取值后的值而不能直接输出其自身。

3、迭代器产生原因

Iterator类的访问方式就是把不同集合类的访问逻辑抽象出来，使得不用暴露集合内部的结构而达到循环遍历集合的效果。

## 8、STL里resize和reserve的区别

resize()：改变当前容器内含有元素的数量(size())，eg: vector<int>v; v.resize(len);v的size变为len,如果原来v的size小于len，那么容器新增（len-size）个元素，元素的值为默认为0.当v.push\_back(3);之后，则是3是放在了v的末尾，即下标为len，此时容器是size为len+1；  
 reserve()：改变当前容器的最大容量（capacity）,它不会生成元素，只是确定这个容器允许放入多少对象，如果reserve(len)的值大于当前的capacity()，那么会重新分配一块能存len个对象的空间，然后把之前v.size()个对象通过copy construtor复制过来，销毁之前的内存；

## 9、C++中类成员的访问权限

C++通过 public、protected、private 三个关键字来控制成员变量和成员函数的访问权限，它们分别表示公有的、受保护的、私有的，被称为成员访问限定符。在类的内部（定义类的代码内部），无论成员被声明为 public、protected 还是 private，都是可以互相访问的，没有访问权限的限制。在类的外部（定义类的代码之外），只能通过对象访问成员，并且通过对象只能访问 public 属性的成员，不能访问 private、protected 属性的成员。

## 10、C++中struct和class的区别

在C++中，可以用struct和class定义类，都可以继承。

区别在于：structural的默认继承权限和默认访问权限是public，而class的默认继承权限和默认访问权限是private。

另外，class还可以定义模板类形参，比如template <class T, int i>。

## 11、C++类内可以定义引用数据成员吗？

可以，必须通过成员函数初始化列表初始化。

## 12、什么是右值引用，跟左值又有什么区别？

右值引用是C++11中引入的新特性 , 它实现了转移语义和精确传递。它的主要目的有两个方面：

1. 消除两个对象交互时不必要的对象拷贝，节省运算存储资源，提高效率。

2. 能够更简洁明确地定义泛型函数。

左值和右值的概念：

左值：能对表达式取地址、或具名对象/变量。一般指表达式结束后依然存在的持久对象。

右值：不能对表达式取地址，或匿名对象。一般指表达式结束就不再存在的临时对象。

右值引用和左值引用的区别：

1. 左值可以寻址，而右值不可以。

2. 左值可以被赋值，右值不可以被赋值，可以用来给左值赋值。

3. 左值可变,右值不可变（仅对基础类型适用，用户自定义类型右值引用可以通过成员函数改变）。

## 13、一个C++源文件从文本到可执行文件经历的过程？

对于C++源文件，从文本到可执行文件一般需要四个过程：

预处理阶段：对源代码文件中文件包含关系（头文件）、预编译语句（宏定义）进行分析和替换，生成预编译文件。

编译阶段：将经过预处理后的预编译文件转换成特定汇编代码，生成汇编文件

汇编阶段：将编译阶段生成的汇编文件转化成机器码，生成可重定位目标文件

链接阶段：将多个目标文件及所需要的库连接成最终的可执行目标文件

## 14、include头文件的顺序以及双引号””和尖括号<>的区别？

Include头文件的顺序：对于include的头文件来说，如果在文件a.h中声明一个在文件b.h中定义的变量，而不引用b.h。那么要在a.c文件中引用b.h文件，并且要先引用b.h，后引用a.h,否则汇报变量类型未声明错误。

双引号和尖括号的区别：编译器预处理阶段查找头文件的路径不一样。

对于使用双引号包含的头文件，查找头文件路径的顺序为：当前头文件目录—>编译器设置的头文件路径（编译器可使用-I显式指定搜索路径）—>系统变量CPLUS\_INCLUDE\_PATH/C\_INCLUDE\_PATH指定的头文件路径

对于使用尖括号包含的头文件，查找头文件的路径顺序为：编译器设置的头文件路径（编译器可使用-I显式指定搜索路径）—>系统变量CPLUS\_INCLUDE\_PATH/C\_INCLUDE\_PATH指定的头文件路径。

## 15、C++的内存管理是怎样的？

在C++中，虚拟内存分为代码段、数据段、BSS段、堆区、文件映射区以及栈区六部分。

代码段:包括只读存储区和文本区，其中只读存储区存储字符串常量，文本区存储程序的机器代码。

数据段：存储程序中已初始化的全局变量和静态变量

bss 段：存储未初始化的全局变量和静态变量（局部+全局），以及所有被初始化为0的全局变量和静态变量。

堆区：调用new/malloc函数时在堆区动态分配内存，同时需要调用delete/free来手动释放申请的内存。

映射区:存储动态链接库以及调用mmap函数进行的文件映射

栈：使用栈空间存储函数的返回地址、参数、局部变量、返回值

## 16、如何判断内存泄漏？

内存泄漏通常是由于调用了malloc/new等内存申请的操作，但是缺少了对应的free/delete。为了判断内存是否泄露，我们一方面可以使用linux环境下的内存泄漏检查工具Valgrind,另一方面我们在写代码时可以添加内存申请和释放的统计功能，统计当前申请和释放的内存是否一致，以此来判断内存是否泄露。

## 17、什么时候会发生段错误

段错误通常发生在访问非法内存地址的时候，具体来说分为以下几种情况：1使用野指针；2试图修改字符串常量的内容

## 18、什么是memory leak，也就是内存泄漏

内存泄漏(memory leak)是指由于疏忽或错误造成了程序未能释放掉不再使用的内存的情况。内存泄漏并非指内存在物理上的消失，而是应用程序分配某段内存后，由于设计错误，失去了对该段内存的控制，因而造成了内存的浪费。

内存泄漏的分类：

1. 堆内存泄漏 （Heap leak）。对内存指的是程序运行中根据需要分配通过malloc,realloc new等从堆中分配的一块内存，再是完成后必须通过调用对应的 free或者delete 删掉。如果程序的设计的错误导致这部分内存没有被释放，那么此后这块内存将不会被使用，就会产生Heap Leak.

2. 系统资源泄露（Resource Leak）。主要指程序使用系统分配的资源比如 Bitmap,handle ,SOCKET等没有使用相应的函数释放掉，导致系统资源的浪费，严重可导致系统效能降低，系统运行不稳定。

3. 没有将基类的析构函数定义为虚函数。当基类指针指向子类对象时，如果基类的析构函数不是virtual，那么子类的析构函数将不会被调用，子类的资源没有正确是释放，因此造成内存泄露。

## 19、C++11有哪些新特性？

C++11 最常用的新特性如下：

auto关键字：编译器可以根据初始值自动推导出类型。但是不能用于函数传参以及数组类型的推导

nullptr关键字：nullptr是一种特殊类型的字面值，它可以被转换成任意其它的指针类型；而NULL一般被宏定义为0，在遇到重载时可能会出现问题。

智能指针：C++11新增了std::shared\_ptr、std::weak\_ptr等类型的智能指针，用于解决内存管理的问题。

初始化列表：使用初始化列表来对类进行初始化

右值引用：基于右值引用可以实现移动语义和完美转发，消除两个对象交互时不必要的对象拷贝，节省运算存储资源，提高效率

atomic原子操作用于多线程资源互斥操作

新增STL容器array以及tuple

# 三、操作系统

## 1、进程与线程的概念，为什么要有进程线程，有什么区别，各自是怎么同步的

**进程**是对运行时程序的封装，是系统进行资源调度和分配的的基本单位，实现了操作系统的并发；

**线程**是进程的子任务，是CPU调度和分派的基本单位，用于保证程序的实时性，实现进程内部的并发；线程是操作系统可识别的最小执行和调度单位。每个线程都独自占用一个虚拟处理器：独自的寄存器组，指令计数器和处理器状态。每个线程完成不同的任务，但是共享同一地址空间（也就是同样的动态内存，映射文件，目标代码等等），打开的文件队列和其他内核资源。

**1、区别：**

1.一个线程只能属于一个进程，而一个进程可以有多个线程，但至少有一个线程。线程依赖于进程而存在。

2.进程在执行过程中拥有独立的内存单元，而多个线程共享进程的内存。（资源分配给进程，同一进程的所有线程共享该进程的所有资源。同一进程中的多个线程共享代码段（代码和常量），数据段（全局变量和静态变量），扩展段（堆存储）。但是每个线程拥有自己的栈段，栈段又叫运行时段，用来存放所有局部变量和临时变量。）

3.进程是资源分配的最小单位，线程是CPU调度的最小单位；

4.系统开销： 由于在创建或撤消进程时，系统都要为之分配或回收资源，如内存空间、I／o设备等。因此，操作系统所付出的开销将显著地大于在创建或撤消线程时的开销。类似地，在进行进程切换时，涉及到整个当前进程CPU环境的保存以及新被调度运行的进程的CPU环境的设置。而线程切换只须保存和设置少量寄存器的内容，并不涉及存储器管理方面的操作。可见，进程切换的开销也远大于线程切换的开销。

5.通信：由于同一进程中的多个线程具有相同的地址空间，致使它们之间的同步和通信的实现，也变得比较容易。进程间通信IPC，线程间可以直接读写进程数据段（如全局变量）来进行通信——需要进程同步和互斥手段的辅助，以保证数据的一致性。在有的系统中，线程的切换、同步和通信都无须操作系统内核的干预

6.进程编程调试简单可靠性高，但是创建销毁开销大；线程正相反，开销小，切换速度快，但是编程调试相对复杂。

7.进程间不会相互影响 ；线程一个线程挂掉将导致整个进程挂掉

8.进程适应于多核、多机分布；线程适用于多核

**2、进程间通信的方式：**

进程间通信主要包括管道、系统IPC（包括消息队列、信号量、信号、共享内存等）、以及套接字socket。

**1.管道：**

管道主要包括无名管道和命名管道:管道可用于具有亲缘关系的父子进程间的通信，有名管道除了具有管道所具有的功能外，它还允许无亲缘关系进程间的通信

1.1 普通管道PIPE：

1)它是半双工的（即数据只能在一个方向上流动），具有固定的读端和写端

2)它只能用于具有亲缘关系的进程之间的通信（也是父子进程或者兄弟进程之间）

3)它可以看成是一种特殊的文件，对于它的读写也可以使用普通的read、write等函数。但是它不是普通的文件，并不属于其他任何文件系统，并且只存在于内存中。

1.2 命名管道FIFO：

1)FIFO可以在无关的进程之间交换数据

2)FIFO有路径名与之相关联，它以一种特殊设备文件形式存在于文件系统中。

2. 系统IPC：

**2.1 消息队列**

消息队列，是消息的链接表，存放在内核中。一个消息队列由一个标识符（即队列ID）来标记。 (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特点)具有写权限得进程可以按照一定得规则向消息队列中添加新信息；对消息队列有读权限得进程则可以从消息队列中读取信息；

特点：

1)消息队列是面向记录的，其中的消息具有特定的格式以及特定的优先级。

2)消息队列独立于发送与接收进程。进程终止时，消息队列及其内容并不会被删除。

3)消息队列可以实现消息的随机查询,消息不一定要以先进先出的次序读取,也可以按消息的类型读取。

**2.2 信号量semaphore**

信号量（semaphore）与已经介绍过的 IPC 结构不同，它是一个计数器，可以用来控制多个进程对共享资源的访问。信号量用于实现进程间的互斥与同步，而不是用于存储进程间通信数据。

特点：

1)信号量用于进程间同步，若要在进程间传递数据需要结合共享内存。

2)信号量基于操作系统的 PV 操作，程序对信号量的操作都是原子操作。

3)每次对信号量的 PV 操作不仅限于对信号量值加 1 或减 1，而且可以加减任意正整数。

4)支持信号量组。

**2.3 信号signal**

信号是一种比较复杂的通信方式，用于通知接收进程某个事件已经发生。

**2.4 共享内存（Shared Memory）**

它使得多个进程可以访问同一块内存空间，不同进程可以及时看到对方进程中对共享内存中数据得更新。这种方式需要依靠某种同步操作，如互斥锁和信号量等

特点：

1)共享内存是最快的一种IPC，因为进程是直接对内存进行存取

2)因为多个进程可以同时操作，所以需要进行同步

3)信号量+共享内存通常结合在一起使用，信号量用来同步对共享内存的访问

**3.套接字SOCKET：**

socket也是一种进程间通信机制，与其他通信机制不同的是，它可用于不同主机之间的进程通信。

**3、线程间通信的方式:**

临界区：通过多线程的串行化来访问公共资源或一段代码，速度快，适合控制数据访问；

互斥量Synchronized/Lock：采用互斥对象机制，只有拥有互斥对象的线程才有访问公共资源的权限。因为互斥对象只有一个，所以可以保证公共资源不会被多个线程同时访问

信号量Semphare：为控制具有有限数量的用户资源而设计的，它允许多个线程在同一时刻去访问同一个资源，但一般需要限制同一时刻访问此资源的最大线程数目。

事件(信号)，Wait/Notify：通过通知操作的方式来保持多线程同步，还可以方便的实现多线程优先级的比较操作。

## 2、说一说Linux虚拟地址空间

为了防止不同进程同一时刻在物理内存中运行而对物理内存的争夺和践踏，采用了虚拟内存。

虚拟内存技术使得不同进程在运行过程中，它所看到的是自己独自占有了当前系统的4G内存。所有进程共享同一物理内存，每个进程只把自己目前需要的虚拟内存空间映射并存储到物理内存上。 事实上，在每个进程创建加载时，内核只是为进程“创建”了虚拟内存的布局，具体就是初始化进程控制表中内存相关的链表，实际上并不立即就把虚拟内存对应位置的程序数据和代码（比如.text .data段）拷贝到物理内存中，只是建立好虚拟内存和磁盘文件之间的映射就好（叫做存储器映射），等到运行到对应的程序时，才会通过缺页异常，来拷贝数据。还有进程运行过程中，要动态分配内存，比如malloc时，也只是分配了虚拟内存，即为这块虚拟内存对应的页表项做相应设置，当进程真正访问到此数据时，才引发缺页异常。

请求分页系统、请求分段系统和请求段页式系统都是针对虚拟内存的，通过请求实现内存与外存的信息置换。

**虚拟内存的好处：**

1.扩大地址空间；

2.内存保护：每个进程运行在各自的虚拟内存地址空间，互相不能干扰对方。虚存还对特定的内存地址提供写保护，可以防止代码或数据被恶意篡改。

3.公平内存分配。采用了虚存之后，每个进程都相当于有同样大小的虚存空间。

4.当进程通信时，可采用虚存共享的方式实现。

5.当不同的进程使用同样的代码时，比如库文件中的代码，物理内存中可以只存储一份这样的代码，不同的进程只需要把自己的虚拟内存映射过去就可以了，节省内存

6.虚拟内存很适合在多道程序设计系统中使用，许多程序的片段同时保存在内存中。当一个程序等待它的一部分读入内存时，可以把CPU交给另一个进程使用。在内存中可以保留多个进程，系统并发度提高

7.在程序需要分配连续的内存空间的时候，只需要在虚拟内存空间分配连续空间，而不需要实际物理内存的连续空间，可以利用碎片

**虚拟内存的代价：**

1.虚存的管理需要建立很多数据结构，这些数据结构要占用额外的内存

2.虚拟地址到物理地址的转换，增加了指令的执行时间。

3.页面的换入换出需要磁盘I/O，这是很耗时的

4.如果一页中只有一部分数据，会浪费内存。

## 3、说一说操作系统中的缺页中断

malloc()和mmap()等内存分配函数，在分配时只是建立了进程虚拟地址空间，并没有分配虚拟内存对应的物理内存。当进程访问这些没有建立映射关系的虚拟内存时，处理器自动触发一个缺页异常。

缺页中断：在请求分页系统中，可以通过查询页表中的状态位来确定所要访问的页面是否存在于内存中。每当所要访问的页面不在内存是，会产生一次缺页中断，此时操作系统会根据页表中的外存地址在外存中找到所缺的一页，将其调入内存。

缺页本身是一种中断，与一般的中断一样，需要经过4个处理步骤：

1、保护CPU现场

2、分析中断原因

3、转入缺页中断处理程序进行处理

4、恢复CPU现场，继续执行

但是缺页中断是由于所要访问的页面不存在于内存时，由硬件所产生的一种特殊的中断，因此，与一般的中断存在区别：

1、在指令执行期间产生和处理缺页中断信号

2、一条指令在执行期间，可能产生多次缺页中断

3、缺页中断返回是，执行产生中断的一条指令，而一般的中断返回是，执行下一条指令。

## 4、fork和vfork的区别

fork:创建一个和当前进程映像一样的进程

成功调用fork( )会创建一个新的进程，它几乎与调用fork( )的进程一模一样，这两个进程都会继续运行。在子进程中，成功的fork( )调用会返回0。在父进程中fork( )返回子进程的pid。如果出现错误，fork( )返回一个负值

fork和vfork的区别：

1. fork( )的子进程拷贝父进程的数据段和代码段；vfork( )的子进程与父进程共享数据段

2. fork( )的父子进程的执行次序不确定；vfork( )保证子进程先运行，在调用exec或exit之前与父进程数据是共享的，在它调用exec或exit之后父进程才可能被调度运行。

3. vfork( )保证子进程先运行，在它调用exec或exit之后父进程才可能被调度运行。如果在调用这两个函数之前子进程依赖于父进程的进一步动作，则会导致死锁。

4.当需要改变共享数据段中变量的值，则拷贝父进程。

## 5、说一说并发(concurrency)和并行(parallelism)

并发（concurrency）：指宏观上看起来两个程序在同时运行，比如说在单核cpu上的多任务。但是从微观上看两个程序的指令是交织着运行的，你的指令之间穿插着我的指令，我的指令之间穿插着你的，在单个周期内只运行了一个指令。这种并发并不能提高计算机的性能，只能提高效率。

并行（parallelism）：指严格物理意义上的同时运行，比如多核cpu，两个程序分别运行在两个核上，两者之间互不影响，单个周期内每个程序都运行了自己的指令，也就是运行了两条指令。这样说来并行的确提高了计算机的效率。所以现在的cpu都是往多核方面发展。

## 6、MySQL的端口号是多少，如何修改这个端口号

查看端口号：

使用命令show global variables like 'port';查看端口号 ，mysql的默认端口是3306。（补充：sqlserver默认端口号为：1433；oracle默认端口号为：1521；DB2默认端口号为：5000；PostgreSQL默认端口号为：5432）

修改端口号：

修改端口号：编辑/etc/my.cnf文件，早期版本有可能是my.conf文件名，增加端口参数，并且设定端口，注意该端口未被使用，保存退出。

## 7、说一说操作系统中的页表寻址

页式内存管理，内存分成固定长度的一个个页片。操作系统为每一个进程维护了一个从虚拟地址到物理地址的映射关系的数据结构，叫页表，页表的内容就是该进程的虚拟地址到物理地址的一个映射。

页表中的每一项都记录了这个页的基地址。通过页表，由逻辑地址的高位部分先找到逻辑地址对应的页基地址，再由页基地址偏移一定长度就得到最后的物理地址，偏移的长度由逻辑地址的低位部分决定。一般情况下，这个过程都可以由硬件完成，所以效率还是比较高的。

页式内存管理的优点就是比较灵活，内存管理以较小的页为单位，方便内存换入换出和扩充地址空间。

## 8、线程需要保存哪些上下文，SP、PC、EAX这些寄存器是？

线程在切换的过程中需要保存当前线程Id、线程状态、堆栈、寄存器状态等信息。其中寄存器主要包括SP PC EAX等寄存器，其主要功能如下：

SP:堆栈指针，指向当前栈的栈顶地址

PC:程序计数器，存储下一条将要执行的指令

EAX:累加寄存器，用于加法乘法的缺省寄存器

## 9、进程和线程的区别

1）进程是cpu资源分配的最小单位，线程是cpu调度的最小单位。

2）进程有独立的系统资源，而同一进程内的线程共享进程的大部分系统资源,包括堆、代码段、数据段，每个线程只拥有一些在运行中必不可少的私有属性，比如tcb,线程Id,栈、寄存器。

3）一个进程崩溃，不会对其他进程产生影响；而一个线程崩溃，会让同一进程内的其他线程也死掉。

4）进程在创建、切换和销毁时开销比较大，而线程比较小。进程创建的时候需要分配系统资源，而销毁的的时候需要释放系统资源。进程切换需要分两步：切换页目录、刷新TLB以使用新的地址空间；切换内核栈和硬件上下文（寄存器）；而同一进程的线程间逻辑地址空间是一样的，不需要切换页目录、刷新TLB。

5）进程间通信比较复杂，而同一进程的线程由于共享代码段和数据段，所以通信比较容易。

## 10、说一说OS缺页置换算法

当访问一个内存中不存在的页，并且内存已满，则需要从内存中调出一个页或将数据送至磁盘对换区，替换一个页，这种现象叫做缺页置换。

当前操作系统最常采用的缺页置换算法如下：先进先出(FIFO)算法：置换最先调入内存的页面，即置换在内存中驻留时间最久的页面。按照进入内存的先后次序排列成队列，从队尾进入，从队首删除。

最近最少使用（LRU）算法: 置换最近一段时间以来最长时间未访问过的页面。根据程序局部性原理，刚被访问的页面，可能马上又要被访问；而较长时间内没有被访问的页面，可能最近不会被访问。

当前最常采用的就是LRU算法。

## 11、多进程和多线程的使用场景

多进程模型的优势是CPU

多线程模型主要优势为线程间切换代价较小，因此适用于I/O密集型的工作场景，因此I/O密集型的工作场景经常会由于I/O阻塞导致频繁的切换线程。同时，多线程模型也适用于单机多核分布式场景。

多进程模型，适用于CPU密集型。同时，多进程模型也适用于多机分布式场景中，易于多机扩展。

## 12、死锁发生的条件以及如何解决死锁

**死锁**是指两个或两个以上进程在执行过程中，因争夺资源而造成的下相互等待的现象。死锁发生的四个必要条件如下：

（1）互斥条件：进程对所分配到的资源不允许其他进程访问，若其他进程访问该资源，只能等待，直至占有该资源的进程使用完成后释放该资源；

（2）请求和保持条件：进程获得一定的资源后，又对其他资源发出请求，但是该资源可能被其他进程占有，此时请求阻塞，但该进程不会释放自己已经占有的资源

（3）不可剥夺条件：进程已获得的资源，在未完成使用之前，不可被剥夺，只能在使用后自己释放

（4）环路等待条件：进程发生死锁后，必然存在一个进程-资源之间的环形链

**解决死锁的方法**即破坏上述四个条件之一，主要方法如下：

资源一次性分配，从而剥夺请求和保持条件

可剥夺资源：即当进程新的资源未得到满足时，释放已占有的资源，从而破坏不可剥夺的条件

资源有序分配法：系统给每类资源赋予一个序号，每个进程按编号递增的请求资源，释放则相反，从而破坏环路等待的条件

## 13、讲述一下互斥锁（mutex）机制，以及互斥锁和读写锁的区别

1、互斥锁和读写锁区别：

互斥锁：mutex，用于保证在任何时刻，都只能有一个线程访问该对象。当获取锁操作失败时，线程会进入睡眠，等待锁释放时被唤醒。

读写锁：rwlock，分为读锁和写锁。处于读操作时，可以允许多个线程同时获得读操作。但是同一时刻只能有一个线程可以获得写锁。其它获取写锁失败的线程都会进入睡眠状态，直到写锁释放时被唤醒。 注意：写锁会阻塞其它读写锁。当有一个线程获得写锁在写时，读锁也不能被其它线程获取；写者优先于读者（一旦有写者，则后续读者必须等待，唤醒时优先考虑写者）。适用于读取数据的频率远远大于写数据的频率的场合。

互斥锁和读写锁的区别：

1）读写锁区分读者和写者，而互斥锁不区分

2）互斥锁同一时间只允许一个线程访问该对象，无论读写；读写锁同一时间内只允许一个写者，但是允许多个读者同时读对象。

2、Linux的4种锁机制：

互斥锁：mutex，用于保证在任何时刻，都只能有一个线程访问该对象。当获取锁操作失败时，线程会进入睡眠，等待锁释放时被唤醒

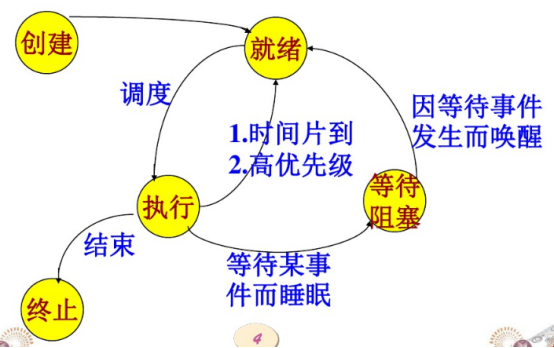
读写锁：rwlock，分为读锁和写锁。处于读操作时，可以允许多个线程同时获得读操作。但是同一时刻只能有一个线程可以获得写锁。其它获取写锁失败的线程都会进入睡眠状态，直到写锁释放时被唤醒。 注意：写锁会阻塞其它读写锁。当有一个线程获得写锁在写时，读锁也不能被其它线程获取；写者优先于读者（一旦有写者，则后续读者必须等待，唤醒时优先考虑写者）。适用于读取数据的频率远远大于写数据的频率的场合。

自旋锁：spinlock，在任何时刻同样只能有一个线程访问对象。但是当获取锁操作失败时，不会进入睡眠，而是会在原地自旋，直到锁被释放。这样节省了线程从睡眠状态到被唤醒期间的消耗，在加锁时间短暂的环境下会极大的提高效率。但如果加锁时间过长，则会非常浪费CPU资源。

RCU：即read-copy-update，在修改数据时，首先需要读取数据，然后生成一个副本，对副本进行修改。修改完成后，再将老数据update成新的数据。使用RCU时，读者几乎不需要同步开销，既不需要获得锁，也不使用原子指令，不会导致锁竞争，因此就不用考虑死锁问题了。而对于写者的同步开销较大，它需要复制被修改的数据，还必须使用锁机制同步并行其它写者的修改操作。在有大量读操作，少量写操作的情况下效率非常高。

## 14、说一说进程状态转换图，动态就绪，静态就绪，动态阻塞，静态阻塞

1、进程的五种基本状态：



1）创建状态：进程正在被创建

2）就绪状态：进程被加入到就绪队列中等待CPU调度运行

3）执行状态：进程正在被运行

4）等待阻塞状态：进程因为某种原因，比如等待I/O，等待设备，而暂时不能运行。

5）终止状态：进程运行完毕

2、活动阻塞，静止阻塞，活动就绪，静止就绪

1）活动阻塞：进程在内存，但是由于某种原因被阻塞了。

2）静止阻塞：进程在外存，同时被某种原因阻塞了。

3）活动就绪：进程在内存，处于就绪状态，只要给CPU和调度就可以直接运行。

4）静止就绪：进程在外存，处于就绪状态，只要调度到内存，给CPU和调度就可以运行。

## 15、静态变量什么时候初始化

静态变量存储在虚拟地址空间的数据段和bss段，C语言中其在代码执行之前初始化，属于编译期初始化。而C++中由于引入对象，对象生成必须调用构造函数，因此C++规定全局或局部静态对象当且仅当对象首次用到时进行构造。

## 16、用户态和内核态区别

用户态和内核态是操作系统的两种运行级别，两者最大的区别就是特权级不同。用户态拥有最低的特权级，内核态拥有较高的特权级。运行在用户态的程序不能直接访问操作系统内核数据结构和程序。内核态和用户态之间的转换方式主要包括：系统调用，异常和中断。

## 17、多线程的同步，锁的机制

同步的时候用一个互斥量，在访问共享资源前对互斥量进行加锁，在访问完成后释放互斥量上的锁。对互斥量进行加锁以后，任何其他试图再次对互斥量加锁的线程将会被阻塞直到当前线程释放该互斥锁。如果释放互斥锁时有多个线程阻塞，所有在该互斥锁上的阻塞线程都会变成可运行状态，第一个变为运行状态的线程可以对互斥量加锁，其他线程将会看到互斥锁依然被锁住，只能回去再次等待它重新变为可用。在这种方式下，每次只有一个线程可以向前执行。

## 18、windows消息机制知道吗，请说一说

当用户有操作(鼠标，键盘等)时，系统会将这些时间转化为消息。每个打开的进程系统都为其维护了一个消息队列，系统会将这些消息放到进程的消息队列中，而应用程序会循环从消息队列中取出来消息，完成对应的操作。

## 19、说一说协程

1、概念：

协程，又称微线程，纤程，英文名Coroutine。协程看上去也是子程序，但执行过程中，在子程序内部可中断，然后转而执行别的子程序，在适当的时候再返回来接着执行。

例如：

def A() :

print '1'

print '2'

print '3'

def B() :

print 'x'

print 'y'

print 'z'

由协程运行结果可能是12x3yz。在执行A的过程中，可以随时中断，去执行B，B也可能在执行过程中中断再去执行A。但协程的特点在于是一个线程执行。

2）协程和线程区别

那和多线程比，协程最大的优势就是协程极高的执行效率。因为子程序切换不是线程切换，而是由程序自身控制，因此，没有线程切换的开销，和多线程比，线程数量越多，协程的性能优势就越明显。

第二大优势就是不需要多线程的锁机制，因为只有一个线程，也不存在同时写变量冲突，在协程中控制共享资源不加锁，只需要判断状态就好了，所以执行效率比多线程高很多。

3）其他

在协程上利用多核CPU呢——多进程+协程，既充分利用多核，又充分发挥协程的高效率，可获得极高的性能。

Python对协程的支持还非常有限，用在generator中的yield可以一定程度上实现协程。虽然支持不完全，但已经可以发挥相当大的威力了。

## 20、系统调用是什么，你用过哪些系统调用

1）概念：

在计算机中，系统调用（英语：system call），又称为系统呼叫，指运行在使用者空间的程序向操作系统内核请求需要更高权限运行的服务。系统调用提供了用户程序与操作系统之间的接口（即系统调用是用户程序和内核交互的接口）。

操作系统中的状态分为管态（核心态）和目态（用户态）。大多数系统交互式操作需求在内核态执行。如设备IO操作或者进程间通信。特权指令：一类只能在核心态下运行而不能在用户态下运行的特殊指令。不同的操作系统特权指令会有所差异，但是一般来说主要是和硬件相关的一些指令。用户程序只在用户态下运行，有时需要访问系统核心功能，这时通过系统调用接口使用系统调用。

应用程序有时会需要一些危险的、权限很高的指令，如果把这些权限放心地交给用户程序是很危险的(比如一个进程可能修改另一个进程的内存区，导致其不能运行)，但是又不能完全不给这些权限。于是有了系统调用，危险的指令被包装成系统调用，用户程序只能调用而无权自己运行那些危险的指令。另外，计算机硬件的资源是有限的，为了更好的管理这些资源，所有的资源都由操作系统控制，进程只能向操作系统请求这些资源。操作系统是这些资源的唯一入口，这个入口就是系统调用。

2）系统调用举例：

对文件进行写操作，程序向打开的文件写入字符串“hello world”，open和write都是系统调用。还有写数据write，创建进程fork，vfork等都是系统调用。

## 21、用户态到内核态的转化原理

1）用户态切换到内核态的3种方式

1、系统调用

这是用户进程主动要求切换到内核态的一种方式，用户进程通过系统调用申请操作系统提供的服务程序完成工作。而系统调用的机制其核心还是使用了操作系统为用户特别开放的一个中断来实现，例如Linux的ine 80h中断。

2、异常

当CPU在执行运行在用户态的程序时，发现了某些事件不可知的异常，这是会触发由当前运行进程切换到处理此。异常的内核相关程序中，也就到了内核态，比如缺页异常。

3、外围设备的中断

当外围设备完成用户请求的操作之后，会向CPU发出相应的中断信号，这时CPU会暂停执行下一条将要执行的指令，转而去执行中断信号的处理程序，如果先执行的指令是用户态下的程序，那么这个转换的过程自然也就发生了有用户态到内核态的切换。比如硬盘读写操作完成，系统会切换到硬盘读写的中断处理程序中执行后续操作等。

2）切换操作

从出发方式看，可以在认为存在前述3种不同的类型，但是从最终实际完成由用户态到内核态的切换操作上来说，涉及的关键步骤是完全一样的，没有任何区别，都相当于执行了一个中断响应的过程，因为系统调用实际上最终是中断机制实现的，而异常和中断处理机制基本上是一样的，用户态切换到内核态的步骤主要包括：

1、从当前进程的描述符中提取其内核栈的ss0及esp0信息。

2、使用ss0和esp0指向的内核栈将当前进程的cs,eip，eflags，ss,esp信息保存起来，这个过程也完成了由用户栈找到内核栈的切换过程，同时保存了被暂停执行的程序的下一条指令。

3、将先前由中断向量检索得到的中断处理程序的cs，eip信息装入相应的寄存器，开始执行中断处理程序，这时就转到了内核态的程序执行了。

## 22、介绍一下5种IO模型

1.阻塞IO:调用者调用了某个函数，等待这个函数返回，期间什么也不做，不停的去检查这个函数有没有返回，必须等这个函数返回才能进行下一步动作  
 2.非阻塞IO:非阻塞等待，每隔一段时间就去检测IO事件是否就绪。没有就绪就可以做其他事。  
 3.信号驱动IO:信号驱动IO:linux用套接口进行信号驱动IO，安装一个信号处理函数，进程继续运行并不阻塞，当IO时间就绪，进程收到SIGIO信号。然后处理IO事件。  
 4.IO复用/多路转接IO:linux用select/poll函数实现IO复用模型，这两个函数也会使进程阻塞，但是和阻塞IO所不同的是这两个函数可以同时阻塞多个IO操作。而且可以同时对多个读操作、写操作的IO函数进行检测。知道有数据可读或可写时，才真正调用IO操作函数  
 5.异步IO:linux中，可以调用aio\_read函数告诉内核描述字缓冲区指针和缓冲区的大小、文件偏移及通知的方式，然后立即返回，当内核将数据拷贝到缓冲区后，再通知应用程序。

## 23、操作系统为什么要分内核态和用户态

为了安全性。在cpu的一些指令中，有的指令如果用错，将会导致整个系统崩溃。分了内核态和用户态后，当用户需要操作这些指令时候，内核为其提供了API，可以通过系统调用陷入内核，让内核去执行这些操作。

## 24、如何设计server，使得能够接收多个客户端的请求

多线程，线程池，io复用

## 25、怎么实现线程池

1.设置一个生产者消费者队列，作为临界资源  
 2.初始化n个线程，并让其运行起来，加锁去队列取任务运行  
 3.当任务队列为空的时候，所有线程阻塞  
 4.当生产者队列来了一个任务后，先对队列加锁，把任务挂在到队列上，然后使用条件变量去通知阻塞中的一个线程。

# 四、计算机网络

## 1、说一下TCP怎么保证可靠性，并且简述一下TCP建立连接和断开连接的过程

TCP保证可靠性：

（1）序列号、确认应答、超时重传

数据到达接收方，接收方需要发出一个确认应答，表示已经收到该数据段，并且确认序号会说明了它下一次需要接收的数据序列号。如果发送发迟迟未收到确认应答，那么可能是发送的数据丢失，也可能是确认应答丢失，这时发送方在等待一定时间后会进行重传。这个时间一般是2\*RTT(报文段往返时间）+一个偏差值。

（2）窗口控制与高速重发控制/快速重传（重复确认应答）

TCP会利用窗口控制来提高传输速度，意思是在一个窗口大小内，不用一定要等到应答才能发送下一段数据，窗口大小就是无需等待确认而可以继续发送数据的最大值。如果不使用窗口控制，每一个没收到确认应答的数据都要重发。

使用窗口控制，如果数据段1001-2000丢失，后面数据每次传输，确认应答都会不停地发送序号为1001的应答，表示我要接收1001开始的数据，发送端如果收到3次相同应答，就会立刻进行重发；但还有种情况有可能是数据都收到了，但是有的应答丢失了，这种情况不会进行重发，因为发送端知道，如果是数据段丢失，接收端不会放过它的，会疯狂向它提醒......

（3）拥塞控制

如果把窗口定的很大，发送端连续发送大量的数据，可能会造成网络的拥堵（大家都在用网，你在这狂发，吞吐量就那么大，当然会堵），甚至造成网络的瘫痪。所以TCP在为了防止这种情况而进行了拥塞控制。

慢启动：定义拥塞窗口，一开始将该窗口大小设为1，之后每次收到确认应答（经过一个rtt），将拥塞窗口大小\*2。

拥塞避免：设置慢启动阈值，一般开始都设为65536。拥塞避免是指当拥塞窗口大小达到这个阈值，拥塞窗口的值不再指数上升，而是加法增加（每次确认应答/每个rtt，拥塞窗口大小+1），以此来避免拥塞。

将报文段的超时重传看做拥塞，则一旦发生超时重传，我们需要先将阈值设为当前窗口大小的一半，并且将窗口大小设为初值1，然后重新进入慢启动过程。

快速重传：在遇到3次重复确认应答（高速重发控制）时，代表收到了3个报文段，但是这之前的1个段丢失了，便对它进行立即重传。

然后，先将阈值设为当前窗口大小的一半，然后将拥塞窗口大小设为慢启动阈值+3的大小。

这样可以达到：在TCP通信时，网络吞吐量呈现逐渐的上升，并且随着拥堵来降低吞吐量，再进入慢慢上升的过程，网络不会轻易的发生瘫痪。

## 2、三次握手和四次挥手

**三次握手：**

1. Client将标志位SYN置为1，随机产生一个值seq=J，并将该数据包发送给Server，Client进入SYN\_SENT状态，等待Server确认。

2. Server收到数据包后由标志位SYN=1知道Client请求建立连接，Server将标志位SYN和ACK都置为1，ack=J+1，随机产生一个值seq=K，并将该数据包发送给Client以确认连接请求，Server进入SYN\_RCVD状态。

3. Client收到确认后，检查ack是否为J+1，ACK是否为1，如果正确则将标志位ACK置为1，ack=K+1，并将该数据包发送给Server，Server检查ack是否为K+1，ACK是否为1，如果正确则连接建立成功，Client和Server进入ESTABLISHED状态，完成三次握手，随后Client与Server之间可以开始传输数据了。

**四次挥手：**

由于TCP连接时全双工的，因此，每个方向都必须要单独进行关闭，这一原则是当一方完成数据发送任务后，发送一个FIN来终止这一方向的连接，收到一个FIN只是意味着这一方向上没有数据流动了，即不会再收到数据了，但是在这个TCP连接上仍然能够发送数据，直到这一方向也发送了FIN。首先进行关闭的一方将执行主动关闭，而另一方则执行被动关闭。

1.数据传输结束后，客户端的应用进程发出连接释放报文段，并停止发送数据，客户端进入FIN\_WAIT\_1状态，此时客户端依然可以接收服务器发送来的数据。

2.服务器接收到FIN后，发送一个ACK给客户端，确认序号为收到的序号+1，服务器进入CLOSE\_WAIT状态。客户端收到后进入FIN\_WAIT\_2状态。

3.当服务器没有数据要发送时，服务器发送一个FIN报文，此时服务器进入LAST\_ACK状态，等待客户端的确认

4.客户端收到服务器的FIN报文后，给服务器发送一个ACK报文，确认序列号为收到的序号+1。此时客户端进入TIME\_WAIT状态，等待2MSL（MSL：报文段最大生存时间），然后关闭连接。

## 3、HTTP和HTTPS的区别，以及HTTPS有什么缺点？

**HTTP协议和HTTPS协议区别如下：**

1）HTTP协议是以明文的方式在网络中传输数据，而HTTPS协议传输的数据则是经过TLS加密后的，HTTPS具有更高的安全性

2）HTTPS在TCP三次握手阶段之后，还需要进行SSL 的handshake，协商加密使用的对称加密密钥

3）HTTPS协议需要服务端申请证书，浏览器端安装对应的根证书

4）HTTP协议端口是80，HTTPS协议端口是443

**HTTPS优点：**

HTTPS传输数据过程中使用密钥进行加密，所以安全性更高

HTTPS协议可以认证用户和服务器，确保数据发送到正确的用户和服务器

**HTTPS缺点：**

HTTPS握手阶段延时较高：由于在进行HTTP会话之前还需要进行SSL握手，因此HTTPS协议握手阶段延时增加

HTTPS部署成本高：一方面HTTPS协议需要使用证书来验证自身的安全性，所以需要购买CA证书；另一方面由于采用HTTPS协议需要进行加解密的计算，占用CPU资源较多，需要的服务器配置或数目高。

## 4、HTTP返回码

HTTP协议的响应报文由状态行、响应头部和响应包体组成，其响应状态码总体描述如下：

1xx：指示信息--表示请求已接收，继续处理。

2xx：成功--表示请求已被成功接收、理解、接受。

3xx：重定向--要完成请求必须进行更进一步的操作。

4xx：客户端错误--请求有语法错误或请求无法实现。

5xx：服务器端错误--服务器未能实现合法的请求。

常见状态代码、状态描述的详细说明如下。

200 OK：客户端请求成功。

206 partial content服务器已经正确处理部分GET请求，实现断点续传或同时分片下载，该请求必须包含Range请求头来指示客户端期望得到的范围

300 multiple choices（可选重定向）:被请求的资源有一系列可供选择的反馈信息，由浏览器/用户自行选择其中一个。

301  moved permanently（永久重定向）：该资源已被永久移动到新位置，将来任何对该资源的访问都要使用本响应返回的若干个URI之一。

302 move temporarily(临时重定向)：请求的资源现在临时从不同的URI中获得，

304：not modified :如果客户端发送一个待条件的GET请求并且该请求以经被允许，而文档内容未被改变，则返回304,该响应不包含包体（即可直接使用缓存）。

403 Forbidden：服务器收到请求，但是拒绝提供服务。

t Found：请求资源不存在，举个例子：输入了错误的URL。

## 5、说一说IP地址作用，以及MAC地址作用

MAC地址是一个硬件地址，用来定义网络设备的位置，主要由数据链路层负责。而IP地址是IP协议提供的一种统一的地址格式，为互联网上的每一个网络和每一台主机分配一个逻辑地址，以此来屏蔽物理地址的差异。

## 6、介绍一下操作系统中的中断

中断是指CPU对系统发生的某个事件做出的一种反应，CPU暂停正在执行的程序，保存现场后自动去执行相应的处理程序，处理完该事件后再返回中断处继续执行原来的程序。中断一般三类，一种是由CPU外部引起的，如I/O中断、时钟中断，一种是来自CPU内部事件或程序执行中引起的中断，例如程序非法操作，地址越界、浮点溢出），最后一种是在程序中使用了系统调用引起的。而中断处理一般分为中断响应和中断处理两个步骤，中断响应由硬件实施，中断处理主要由软件实施。

## 7、回答OSI七层模型和TCP/IP四层模型，每层列举2个协议

OSI七层模型及其包含的协议如下:

物理层: 通过媒介传输比特,确定机械及电气规范,传输单位为bit，主要包括的协议为：IEE802.3 CLOCK RJ45

数据链路层: 将比特组装成帧和点到点的传递,传输单位为帧,主要包括的协议为MAC VLAN PPP

网络层：负责数据包从源到宿的传递和网际互连，传输单位为包,主要包括的协议为IP ARP ICMP

传输层：提供端到端的可靠报文传递和错误恢复，传输单位为报文,主要包括的协议为TCP UDP

会话层：建立、管理和终止会话，传输单位为SPDU，主要包括的协议为RPC NFS

表示层: 对数据进行翻译、加密和压缩,传输单位为PPDU，主要包括的协议为JPEG ASII

应用层: 允许访问OSI环境的手段,传输单位为APDU，主要包括的协议为FTP HTTP DNS

**TCP/IP 4层模型包括：**

网络接口层：MAC VLAN

网络层:IP ARP ICMP

传输层:TCP UDP

应用层:HTTP DNS SMTP

## 8、搜索baidu，会用到计算机网络中的什么层？每层是干什么的

**浏览器中输入URL**

浏览器要将URL解析为IP地址，解析域名就要用到DNS协议，首先主机会查询DNS的缓存，如果没有就给本地DNS发送查询请求。DNS查询分为两种方式，一种是递归查询，一种是迭代查询。如果是迭代查询，本地的DNS服务器，向根域名服务器发送查询请求，根域名服务器告知该域名的一级域名服务器，然后本地服务器给该一级域名服务器发送查询请求，然后依次类推直到查询到该域名的IP地址。DNS服务器是基于UDP的，因此会用到UDP协议。

**得到IP地址后**，浏览器就要与服务器建立一个http连接。因此要用到http协议，http协议报文格式上面已经提到。http生成一个get请求报文，将该报文传给TCP层处理，所以还会用到TCP协议。如果采用https还会使用https协议先对http数据进行加密。TCP层如果有需要先将HTTP数据包分片，分片依据路径MTU和MSS。TCP的数据包然后会发送给IP层，用到IP协议。IP层通过路由选路，一跳一跳发送到目的地址。当然在一个网段内的寻址是通过以太网协议实现(也可以是其他物理层协议，比如PPP，SLIP)，以太网协议需要直到目的IP地址的物理地址，有需要ARP协议。

其中：

1、DNS协议，http协议，https协议属于应用层

应用层是体系结构中的最高层。应用层确定进程之间通信的性质以满足用户的需要。这里的进程就是指正在运行的程序。应用层不仅要提供应用进程所需要的信息交换和远地操作，而且还要作为互相作用的应用进程的用户代理，来完成一些为进行语义上有意义的信息交换所必须的功能。应用层直接为用户的应用进程提供服务。

2、TCP/UDP属于传输层

传输层的任务就是负责主机中两个进程之间的通信。因特网的传输层可使用两种不同协议：即面向连接的传输控制协议[TCP](https://www.baidu.com/s?wd=TCP&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)，和无连接的用户数据报协议[UDP](https://www.baidu.com/s?wd=UDP&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)。面向连接的服务能够提供可靠的交付，但无连接服务则不保证提供可靠的交付，它只是“尽最大努力交付”。这两种服务方式都很有用，备有其优缺点。在分组交换网内的各个交换结点机都没有传输层。

3、IP协议，ARP协议属于网络层

网络层负责为分组交换网上的不同主机提供通信。在发送数据时，网络层将运输层产生的报文段或用户数据报封装成分组或包进行传送。在[TCP](https://www.baidu.com/s?wd=TCP&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)/IP体系中，分组也叫作IP数据报，或简称为数据报。网络层的另一个任务就是要选择合适的路由，使源主机运输层所传下来的分组能够交付到目的主机。  
4、数据链路层

当发送数据时，数据链路层的任务是将在网络层交下来的IP数据报组装成帧，在两个相邻结点间的链路上传送以帧为单位的数据。每一帧包括数据和必要的控制信息（如同步信息、地址信息、差错控制、以及[流量控制](https://www.baidu.com/s?wd=%E6%B5%81%E9%87%8F%E6%8E%A7%E5%88%B6&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)信息等）。控制信息使接收端能够知道—个帧从哪个比特开始和到哪个比特结束。控制信息还使接收端能够检测到所收到的帧中有无差错。  
5、物理层

物理层的任务就是透明地传送比特流。在物理层上所传数据的单位是比特。传递信息所利用的一些物理媒体，如[双绞线](https://www.baidu.com/s?wd=%E5%8F%8C%E7%BB%9E%E7%BA%BF&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)、同轴电缆、光缆等，并不在物理层之内而是在物理层的下面。因此也有人把物理媒体当做第0层。

## 9、说一说TCP拥塞控制？以及达到什么情况的时候开始减慢增长的速度？

拥塞控制是防止过多的数据注入网络，使得网络中的路由器或者链路过载。流量控制是点对点的通信量控制，而拥塞控制是全局的网络流量整体性的控制。发送双方都有一个拥塞窗口——cwnd。

1、慢开始

最开始发送方的拥塞窗口为1，由小到大逐渐增大发送窗口和拥塞窗口。每经过一个传输轮次，拥塞窗口cwnd加倍。当cwnd超过慢开始门限，则使用拥塞避免算法，避免cwnd增长过大。

2、拥塞避免

每经过一个往返时间RTT，cwnd就增长1。

在慢开始和拥塞避免的过程中，一旦发现网络拥塞，就把慢开始门限设为当前值的一半，并且重新设置cwnd为1，重新慢启动。（乘法减小，加法增大）

3、快重传

接收方每次收到一个失序的报文段后就立即发出重复确认，发送方只要连续收到三个重复确认就立即重传（尽早重传未被确认的报文段）。

4、快恢复

当发送方连续收到了三个重复确认，就乘法减半（慢开始门限减半），将当前的cwnd设置为慢开始门限，并且采用拥塞避免算法（连续收到了三个重复请求，说明当前网络可能没有拥塞）。

采用快恢复算法时，慢开始只在建立连接和网络超时才使用。

达到什么情况的时候开始减慢增长的速度？

采用慢开始和拥塞避免算法的时候

1. 一旦cwnd>慢开始门限，就采用拥塞避免算法，减慢增长速度

2. 一旦出现丢包的情况，就重新进行慢开始，减慢增长速度

采用快恢复和快重传算法的时候

1. 一旦cwnd>慢开始门限，就采用拥塞避免算法，减慢增长速度

2. 一旦发送方连续收到了三个重复确认，就采用拥塞避免算法，减慢增长速度

## 10、TCP用了哪些措施保证其可靠性

1、序列号、确认应答、超时重传

数据到达接收方，接收方需要发出一个确认应答，表示已经收到该数据段，并且确认序号会说明了它下一次需要接收的数据序列号。如果发送发迟迟未收到确认应答，那么可能是发送的数据丢失，也可能是确认应答丢失，这时发送方在等待一定时间后会进行重传。这个时间一般是2\*RTT(报文段往返时间）+一个偏差值。

2、窗口控制与高速重发控制/快速重传（重复确认应答）

TCP会利用窗口控制来提高传输速度，意思是在一个窗口大小内，不用一定要等到应答才能发送下一段数据，窗口大小就是无需等待确认而可以继续发送数据的最大值。如果不使用窗口控制，每一个没收到确认应答的数据都要重发。

使用窗口控制，如果数据段1001-2000丢失，后面数据每次传输，确认应答都会不停地发送序号为1001的应答，表示我要接收1001开始的数据，发送端如果收到3次相同应答，就会立刻进行重发；但还有种情况有可能是数据都收到了，但是有的应答丢失了，这种情况不会进行重发，因为发送端知道，如果是数据段丢失，接收端不会放过它的，会疯狂向它提醒......

3、拥塞控制

如果把窗口定的很大，发送端连续发送大量的数据，可能会造成网络的拥堵（大家都在用网，你在这狂发，吞吐量就那么大，当然会堵），甚至造成网络的瘫痪。所以TCP在为了防止这种情况而进行了拥塞控制。

慢启动：定义拥塞窗口，一开始将该窗口大小设为1，之后每次收到确认应答（经过一个rtt），将拥塞窗口大小\*2。

拥塞避免：设置慢启动阈值，一般开始都设为65536。拥塞避免是指当拥塞窗口大小达到这个阈值，拥塞窗口的值不再指数上升，而是加法增加（每次确认应答/每个rtt，拥塞窗口大小+1），以此来避免拥塞。

将报文段的超时重传看做拥塞，则一旦发生超时重传，我们需要先将阈值设为当前窗口大小的一半，并且将窗口大小设为初值1，然后重新进入慢启动过程。

快速重传：在遇到3次重复确认应答（高速重发控制）时，代表收到了3个报文段，但是这之前的1个段丢失了，便对它进行立即重传。

然后，先将阈值设为当前窗口大小的一半，然后将拥塞窗口大小设为慢启动阈值+3的大小。

这样可以达到：在TCP通信时，网络吞吐量呈现逐渐的上升，并且随着拥堵来降低吞吐量，再进入慢慢上升的过程，网络不会轻易的发生瘫痪。

## 11、传递到IP层怎么知道报文给哪个应用程序，怎么区分UDP报文和TCP报文

根据端口区分；看ip头中的协议标识字段，17是udp，6是tcp

## 12、TCP和UDP的区别和各自适用的场景

1）TCP和UDP区别

1） 连接

TCP是面向连接的传输层协议，即传输数据之前必须先建立好连接。

UDP无连接。

2） 服务对象

TCP是点对点的两点间服务，即一条TCP连接只能有两个端点；

UDP支持一对一，一对多，多对一，多对多的交互通信。

3） 可靠性

TCP是可靠交付：无差错，不丢失，不重复，按序到达。

UDP是尽最大努力交付，不保证可靠交付。

4）拥塞控制，流量控制

TCP有拥塞控制和流量控制保证数据传输的安全性。

UDP没有拥塞控制，网络拥塞不会影响源主机的发送效率。

5） 报文长度

TCP是动态报文长度，即TCP报文长度是根据接收方的窗口大小和当前网络拥塞情况决定的。

UDP面向报文，不合并，不拆分，保留上面传下来报文的边界。

6)   首部开销

TCP首部开销大，首部20个字节。

UDP首部开销小，8字节。（源端口，目的端口，数据长度，校验和）

2）TCP和UDP适用场景

从特点上我们已经知道，TCP 是可靠的但传输速度慢，UDP 是不可靠的但传输速度快。因此在选用具体协议通信时，应该根据通信数据的要求而决定。

若通信数据完整性需让位与通信实时性，则应该选用TCP 协议（如文件传输、重要状态的更新等）；反之，则使用 UDP 协议（如视频传输、实时通信等）。

## 13、说一下数字证书是什么，里面都包含那些内容

1）概念：

数字证书是数字证书在一个身份和该身份的持有者所拥有的公/私钥对之间建立了一种联系，由认证中心（CA）或者认证中心的下级认证中心颁发的。根证书是认证中心与用户建立信任关系的基础。在用户使用数字证书之前必须首先下载和安装。

认证中心是一家能向用户签发数字证书以确认用户身份的管理机构。为了防止数字凭证的伪造，认证中心的公共密钥必须是可靠的，认证中心必须公布其公共密钥或由更高级别的认证中心提供一个电子凭证来证明其公共密钥的有效性，后一种方法导致了多级别认证中心的出现。

2）数字证书颁发过程：

数字证书颁发过程如下：用户产生了自己的密钥对，并将公共密钥及部分个人身份信息传送给一家认证中心。认证中心在核实身份后，将执行一些必要的步骤，以确信请求确实由用户发送而来，然后，认证中心将发给用户一个数字证书，该证书内附了用户和他的密钥等信息，同时还附有对认证中心公共密钥加以确认的数字证书。当用户想证明其公开密钥的合法性时，就可以提供这一数字证书。

3）内容：

数字证书的格式普遍采用的是X.509V3国际标准，一个标准的X.509数字证书包含以下一些内容：

1、证书的版本信息；

2、证书的序列号，每个证书都有一个唯一的证书序列号；

3、证书所使用的签名算法；

4、证书的发行机构名称，命名规则一般采用X.500格式；

5、证书的有效期，通用的证书一般采用UTC时间格式；

6、证书所有人的名称，命名规则一般采用X.500格式；

7、证书所有人的公开密钥；

8、证书发行者对证书的签名。

## 14、讲述一下Socket编程的send() recv() accept() socket()函数？

send函数用来向TCP连接的另一端发送数据。客户程序一般用send函数向服务器发送请求，而服务器则通常用send函数来向客户程序发送应答,send的作用是将要发送的数据拷贝到缓冲区，协议负责传输。

recv函数用来从TCP连接的另一端接收数据，当应用程序调用recv函数时，recv先等待s的发送缓冲中的数据被协议传送完毕，然后从缓冲区中读取接收到的内容给应用层。  
accept函数用了接收一个连接，内核维护了半连接队列和一个已完成连接队列，当队列为空的时候，accept函数阻塞，不为空的时候accept函数从上边取下来一个已完成连接，返回一个文件描述符。

## 15、说一说TCP/IP数据链路层的交互过程

网络层等到数据链层用mac地址作为通信目标，数据包到达网络等准备往数据链层发送的时候，首先会去自己的arp缓存表(存着ip-mac对应关系)去查找改目标ip的mac地址，如果查到了，就讲目标ip的mac地址封装到链路层数据包的包头。如果缓存中没有找到，会发起一个广播：who is ip XXX tell ip XXX,所有收到的广播的机器看这个ip是不是自己的，如果是自己的，则以单拨的形式将自己的mac地址回复给请求的机器

# 五、数据库

## 1、数据库索引

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。如果想按特定职员的姓来查找他或她，则与在表中搜索所有的行相比，索引有助于更快地获取信息。

索引的一个主要目的就是加快检索表中数据的方法，亦即能协助信息搜索者尽快的找到符合限制条件的记录ID的辅助数据结构。

## 2、数据库事务

数据库事务(Database Transaction) ，是指作为单个逻辑工作单元执行的一系列操作，要么完全地执行，要么完全地不执行。 事务处理可以确保除非事务性单元内的所有操作都成功完成，否则不会永久更新面向数据的资源。通过将一组相关操作组合为一个要么全部成功要么全部失败的单元，可以简化错误恢复并使应用程序更加可靠。一个逻辑工作单元要成为事务，必须满足所谓的ACID（原子性、一致性、隔离性和持久性）属性。事务是数据库运行中的逻辑工作单位，由DBMS中的事务管理子系统负责事务的处理。

## 3、说一说数据库事务隔离

同一时间，只允许一个事务请求同一数据，不同的事务之间彼此没有任何干扰。比如A正在从一张银行卡中取钱，在A取钱的过程结束前，B不能向这张卡转账。

## 4、说一说inner join和left join

left join(左联接) 返回包括左表中的所有记录和右表中联结字段相等的记录

right join(右联接) 返回包括右表中的所有记录和左表中联结字段相等的记录  
 inner join(等值连接) 只返回两个表中联结字段相等的行。

## 5、说一下数据库事务以及四个特性

事务（Transaction）是由一系列对系统中数据进行访问与更新的操作所组成的一个程序执行逻辑单元。事务是DBMS中最基础的单位，事务不可分割。

事务具有4个基本特征，分别是：原子性（Atomicity）、一致性（Consistency）、隔离性（Isolation）、持久性（Duration），简称ACID。

1. 原子性（Atomicity）

原子性是指事务包含的所有操作要么全部成功，要么全部失败回滚，[删删删]因此事务的操作如果成功就必须要完全应用到数据库，如果操作失败则不能对数据库有任何影响。

2. 一致性（Consistency）

一致性是指事务必须使数据库从一个一致性状态变换到另一个一致性状态，也就是说一个事务执行之前和执行之后都必须处于一致性状态。

3. 隔离性（Isolation）

隔离性是当多个用户并发访问数据库时，比如操作同一张表时，数据库为每一个用户开启的事务，不能被其他事务的操作所干扰，多个并发事务之间要相互隔离。

多个事务并发访问时，事务之间是隔离的，一个事务不应该影响其它事务运行效果。

这指的是在并发环境中，当不同的事务同时操纵相同的数据时，每个事务都有各自的完整数据空间。由并发事务所做的修改必须与任何其他并发事务所做的修改隔离。

不同的隔离级别：

Read Uncommitted（读取未提交[添加中文释义]内容）：最低的隔离级别，什么都不需要做，一个事务可以读到另一个事务未提交的结果。所有的并发事务问题都会发生。

Read Committed（读取提交内容）：只有在事务提交后，其更新结果才会被其他事务看见。可以解决脏读问题。

Repeated Read（可重复读）：在一个事务中，对于同一份数据的读取结果总是相同的，无论是否有其他事务对这份数据进行操作，以及这个事务是否提交。可以解决脏读、不可重复读。

Serialization（可串行化）：事务串行化执行，隔离级别最高，牺牲了系统的并发性。可以解决并发事务的所有问题。

4. 持久性（Durability）

持久性是指一个事务一旦被提交了，那么对数据库中的数据的改变就是永久性的，即便是在数据库系统遇到故障的情况下也不会丢失提交事务的操作。

## 6、数据库的三大范式

第一范式：当关系模式R的所有属性都不能再分解为更基本的数据单位时，称R是满足第一范式，即属性不可分

第二范式：如果关系模式R满足第一范式，并且R得所有非主属性都完全依赖于R的每一个候选关键属性，称R满足第二范式

第三范式：设R是一个满足第一范式条件的关系模式，X是R的任意属性集，如果X非传递依赖于R的任意一个候选关键字，称R满足第三范式，即非主属性不传递依赖于键码

## 7、介绍一下mysql的MVCC机制

MVCC是一种多版本并发控制机制，是MySQL的InnoDB存储引擎实现隔离级别的一种具体方式，用于实现提交读和可重复读这两种隔离级别。MVCC是通过保存数据在某个时间点的快照来实现该机制，其在每行记录后面保存两个隐藏的列，分别保存这个行的创建版本号和删除版本号，然后Innodb的MVCC使用到的快照存储在Undo日志中，该日志通过回滚指针把一个数据行所有快照连接起来。

## 8、SQL优化方法有哪些

通过建立索引对查询进行优化；对查询进行优化，应尽量避免全表扫描

## 9、说一下MySQL引擎和区别

1、MySQL引擎

MySQL中的数据用各种不同的技术存储在文件（或者内存）中。这些技术中的每一种技术都使用不同的存储机制、索引技巧、锁定水平并且最终提供广泛的不同的功能和能力。通过选择不同的技术，你能够获得额外的速度或者功能，从而改善你的应用的整体功能。

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。使用数据库引擎创建用于联机事务处理或联机分析处理数据的关系数据库。这包括创建用于存储数据的表和用于查看、管理和保护数据安全的数据库对象（如索引、视图和存储过程）。

MySQL存储引擎主要有： MyIsam、InnoDB、Memory、Blackhole、CSV、Performance\_Schema、Archive、Federated、Mrg\_Myisam。

但是最常用的是InnoDB和Mylsam。

在Mysql数据库中，常用的引擎为Innodb和MyIASM,其中Innodb是一个事务型的存储引擎，有行级锁定和外键约束，提供了对数据库ACID事物的支持，实现了SQL标准的四种隔离级别，即读未提交，不可重复读，可重复读以及串行,其涉及目标就是处理大数据容量的数据库系统。而MyIASM引擎是Mysql默认的引擎，不提供数据库事务的支持，也不支持行级锁和外键，因此当写操作时需要锁定整个表，效率较低。不过其保存了表的行数，当金星select count(\*)form table时，可直接读取已经保存的值，不需要进行全表扫描。因此当表的读操作远多于写操作，并且不需要事务支持时，可以优先选择MyIASM。

## 10、mongodb和redis的区别

内存管理机制上：Redis 数据全部存在内存，定期写入磁盘，当内存不够时，可以选择指定的 LRU 算法删除数据。MongoDB 数据存在内存，由 linux系统 mmap 实现，当内存不够时，只将热点数据放入内存，其他数据存在磁盘。

支持的数据结构上：Redis 支持的数据结构丰富，包括hash、set、list等。

MongoDB 数据结构比较单一，但是支持丰富的数据表达，索引，最类似关系型数据库，支持的查询语言非常丰富

## 11、Redis是单线程的，但是为什么这么高效呢?

虽然Redis文件事件处理器以单线程方式运行，但是通过使用I/O多路复用程序来监听多个套接字，文件事件处理器既实现了高性能的网络通信模型，又可以很好地与Redis服务器中其他同样以单线程运行的模块进行对接，这保持了Redis内部单线程设计的简单性。

## 12、Redis的数据类型有哪些，底层怎么实现?

1）字符串：整数值、embstr编码的简单动态字符串、简单动态字符串（SDS)

2）列表：压缩列表、双端链表

3）哈希：压缩列表、字典

4）集合：整数集合、字典

5）有序集合：压缩列表、跳跃表和字典

## 13、Redis和memcached的区别

1）数据类型 ：redis数据类型丰富，支持set liset等类型；memcache支持简单数据类型，需要客户端自己处理复杂对象

2）持久性：redis支持数据落地持久化存储；memcache不支持数据持久存储。)

3）分布式存储：redis支持master-slave复制模式；memcache可以使用一致性hash做分布式。

4）value大小不同：memcache是一个内存缓存，key的长度小于250字符，单个item存储要小于1M，不适合虚拟机使用

5）数据一致性不同：redis使用的是单线程模型，保证了数据按顺序提交；memcache需要使用cas保证数据一致性。CAS（Check and Set）是一个确保并发一致性的机制，属于“乐观锁”范畴；原理很简单：拿版本号，操作，对比版本号，如果一致就操作，不一致就放弃任何操作

6）cpu利用：redis单线程模型只能使用一个cpu，可以开启多个redis进程

# 六、数据结构与算法

## 1、红黑树和AVL树的定义，特点，以及二者区别

平衡二叉树（AVL树）：

平衡二叉树又称为AVL树，是一种特殊的二叉排序树。其左右子树都是平衡二叉树，且左右子树高度之差的绝对值不超过1。一句话表述为：以树中所有结点为根的树的左右子树高度之差的绝对值不超过1。将二叉树上结点的左子树深度减去右子树深度的值称为平衡因子BF，那么平衡二叉树上的所有结点的平衡因子只可能是-1、0和1。只要二叉树上有一个结点的平衡因子的绝对值大于1，则该二叉树就是不平衡的。

红黑树：

红黑树是一种二叉查找树，但在每个节点增加一个存储位表示节点的颜色，可以是红或黑（非红即黑）。通过对任何一条从根到叶子的路径上各个节点着色的方式的限制，红黑树确保没有一条路径会比其它路径长出两倍，因此，红黑树是一种弱平衡二叉树，相对于要求严格的AVL树来说，它的旋转次数少，所以对于搜索，插入，删除操作较多的情况下，通常使用红黑树。

性质：

1. 每个节点非红即黑

2. 根节点是黑的;

3. 每个叶节点（叶节点即树尾端NULL指针或NULL节点）都是黑的;

4. 如果一个节点是红色的，则它的子节点必须是黑色的。

5. 对于任意节点而言，其到叶子点树NULL指针的每条路径都包含相同数目的黑节点;

**区别：**

AVL 树是高度平衡的，频繁的插入和删除，会引起频繁的rebalance，导致效率下降；红黑树不是高度平衡的，算是一种折中，插入最多两次旋转，删除最多三次旋转。

## 2、哈夫曼编码

哈夫曼编码是哈夫曼树的一种应用，广泛用于数据文件压缩。哈夫曼编码算法用字符在文件中出现的频率来建立使用0，1表示个字符的最优表示方式，其具体算法如下：

(1)哈夫曼算法以自底向上的方式构造表示最优前缀码的二叉树T。

(2)算法以|C|个叶结点开始，执行|C|－1次的“合并”运算后产生最终所要求的树T。

(3)假设编码字符集中每一字符c的频率是f(c)。以f为键值的优先队列Q用在贪心选择时有效地确定算法当前要合并的2棵具有最小频率的树。一旦2棵具有最小频率的树合并后，产生一棵新的树，其频率为合并的2棵树的频率之和，并将新树插入优先队列Q。经过n－1次的合并后，优先队列中只剩下一棵树，即所要求的树T。

## 3、map底层为什么用红黑树实现

1、红黑树：

红黑树是一种二叉查找树，但在每个节点增加一个存储位表示节点的颜色，可以是红或黑（非红即黑）。通过对任何一条从根到叶子的路径上各个节点着色的方式的限制，红黑树确保没有一条路径会比其它路径长出两倍，因此，红黑树是一种弱平衡二叉树，相对于要求严格的AVL树来说，它的旋转次数少，所以对于搜索，插入，删除操作较多的情况下，通常使用红黑树。

性质：

1. 每个节点非红即黑

2. 根节点是黑的;

3. 每个叶节点（叶节点即树尾端NULL指针或NULL节点）都是黑的;

4. 如果一个节点是红色的，则它的子节点必须是黑色的。

5. 对于任意节点而言，其到叶子点树NULL指针的每条路径都包含相同数目的黑节点;

2、平衡二叉树（AVL树）：

红黑树是在AVL树的基础上提出来的。

平衡二叉树又称为AVL树，是一种特殊的二叉排序树。其左右子树都是平衡二叉树，且左右子树高度之差的绝对值不超过1。

AVL树中所有结点为根的树的左右子树高度之差的绝对值不超过1。

将二叉树上结点的左子树深度减去右子树深度的值称为平衡因子BF，那么平衡二叉树上的所有结点的平衡因子只可能是-1、0和1。只要二叉树上有一个结点的平衡因子的绝对值大于1，则该二叉树就是不平衡的。

3、红黑树较AVL树的优点：

AVL 树是高度平衡的，频繁的插入和删除，会引起频繁的rebalance，导致效率下降；红黑树不是高度平衡的，算是一种折中，插入最多两次旋转，删除最多三次旋转。

所以红黑树在查找，插入删除的性能都是O(logn)，且性能稳定，所以STL里面很多结构包括map底层实现都是使用的红黑树。

4、红黑树旋转：

旋转：红黑树的旋转是一种能保持二叉搜索树性质的搜索树局部操作。有左旋和右旋两种旋转，通过改变树中某些结点的颜色以及指针结构来保持对红黑树进行插入和删除操作后的红黑性质。

左旋：对某个结点x做左旋操作时，假设其右孩子为y而不是T.nil：以x到y的链为“支轴”进行。使y成为该子树新的根结点，x成为y的左孩子，y的左孩子成为x的右孩子。

## 4、请你介绍一下B+树

B+是一种多路搜索树，主要为磁盘或其他直接存取辅助设备而设计的一种平衡查找树，在B+树中，每个节点的可以有多个孩子，并且按照关键字大小有序排列。所有记录节点都是按照键值的大小顺序存放在同一层的叶节点中。相比B树，其具有以下几个特点：

每个节点上的指针上限为2d而不是2d+1（d为节点的出度）

内节点不存储data,只存储key

叶子节点不存储指针。

## 5、epoll怎么实现的

Linux epoll机制是通过红黑树和双向链表实现的。 首先通过epoll\_create()系统调用在内核中创建一个eventpoll类型的句柄，其中包括红黑树根节点和双向链表头节点。然后通过epoll\_ctl()系统调用，向epoll对象的红黑树结构中添加、删除、修改感兴趣的事件，返回0标识成功，返回-1表示失败。最后通过epoll\_wait()系统调用判断双向链表是否为空，如果为空则阻塞。当文件描述符状态改变，fd上的回调函数被调用，该函数将fd加入到双向链表中，此时epoll\_wait函数被唤醒，返回就绪好的事件。

## 6、栈和堆的区别，以及为什么栈要快

堆和栈的区别：

堆是由低地址向高地址扩展；栈是由高地址向低地址扩展

堆中的内存需要手动申请和手动释放；栈中内存是由OS自动申请和自动释放，存放着参数、局部变量等内存

堆中频繁调用malloc和free,会产生内存碎片，降低程序效率；而栈由于其先进后出的特性，不会产生内存碎片

堆的分配效率较低，而栈的分配效率较高

栈的效率高的原因：

栈是操作系统提供的数据结构，计算机底层对栈提供了一系列支持：分配专门的寄存器存储栈的地址，压栈和入栈有专门的指令执行；而堆是由C/C++函数库提供的，机制复杂，需要一些列分配内存、合并内存和释放内存的算法，因此效率较低。

## 7、小根堆特点

堆是一棵完全二叉树（如果一共有h层，那么1~h-1层均满，在h层可能会连续缺失若干个右叶子）。

1）小根堆

若根节点存在左子女则根节点的值小于左子女的值；若根节点存在右子女则根节点的值小于右子女的值。

2）大根堆

若根节点存在左子女则根节点的值大于左子女的值；若根节点存在右子女则根节点的值大于右子女的值。

## 8、Array&List， 数组和链表的区别

**数组的特点：**

数组是将元素在内存中连续存放，由于每个元素占用内存相同，可以通过下标迅速访问数组中任何元素。数组的插入数据和删除数据效率低，插入数据时，这个位置后面的数据在内存中都要向后移。删除数据时，这个数据后面的数据都要往前移动。但数组的随机读取效率很高。因为数组是连续的，知道每一个数据的内存地址，可以直接找到给地址的数据。如果应用需要快速访问数据，很少或不插入和删除元素，就应该用数组。数组需要预留空间，在使用前要先申请占内存的大小，可能会浪费内存空间。并且数组不利于扩展，数组定义的空间不够时要重新定义数组。

**链表的特点：**

链表中的元素在内存中不是顺序存储的，而是通过存在元素中的指针联系到一起。比如：上一个元素有个指针指到下一个元素，以此类推，直到最后一个元素。如果要访问链表中一个元素，需要从第一个元素开始，一直找到需要的元素位置。但是增加和删除一个元素对于链表数据结构就非常简单了，只要修改元素中的指针就可以了。如果应用需要经常插入和删除元素你就需要用链表数据结构了。不指定大小，扩展方便。链表大小不用定义，数据随意增删。

**各自的优缺点**

**数组的优点：**

1. 随机访问性强

2. 查找速度快

**数组的缺点:**

1. 插入和删除效率低

2. 可能浪费内存

3. 内存空间要求高，必须有足够的连续内存空间。

4. 数组大小固定，不能动态拓展

**链表的优点:**

1. 插入删除速度快

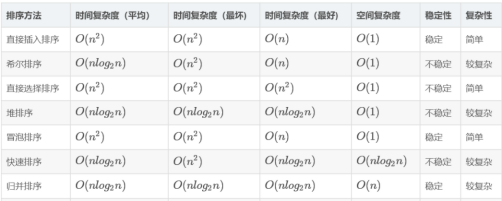
2. 内存利用率高，不会浪费内存

3. 大小没有固定，拓展很灵活。

链表的缺点:

不能随机查找，必须从第一个开始遍历，查找效率低。

## 9、各个排序的时间复杂度、空间复杂度及稳定性如下



## 10、说一说hash表的实现，包括STL中的哈希桶长度常数

hash表的实现主要包括构造哈希和处理哈希冲突两个方面：

对于构造哈希来说，主要包括直接地址法、平方取中法、除留余数

法等。

对于处理哈希冲突来说，最常用的处理冲突的方法有开放定址法、再哈希法、链地址法、建立公共溢出区等方法。SGL版本使用链地址法，使用一个链表保持相同散列值的元素。

虽然链地址法并不要求哈希桶长度必须为质数，但SGI STL仍然以质数来设计哈希桶长度，并且将28个质数（逐渐呈现大约两倍的关系）计算好，以备随时访问，同时提供一个函数，用来查询在这28个质数之中，“最接近某数并大于某数”的质数。

## 11、回答一下hash表如何rehash，以及怎么处理其中保存的资源

C++的hash表中有一个负载因子loadFactor，当loadFactor<=1时，hash表查找的期望复杂度为O(1). 因此，每次往hash表中添加元素时，我们必须保证是在loadFactor <1的情况下，才能够添加。

因此，当Hash表中loadFactor==1时，Hash就需要进行rehash。rehash过程中，会模仿C++的vector扩容方式，Hash表中每次发现loadFactor ==1时，就开辟一个原来桶数组的两倍空间，称为新桶数组，然后把原来的桶数组中元素全部重新哈希到新的桶数组中。

## 12、解决hash冲突的方法

当哈希表关键字集合很大时，关键字值不同的元素可能会映象到哈希表的同一地址上，这样的现象称为哈希冲突。目前常用的解决哈希冲突的方法如下：

开放定址法: 当发生地址冲突时，按照某种方法继续探测哈希表中的其他存储单元，直到找到空位置为止。

再哈希法：当发生哈希冲突时使用另一个哈希函数计算地址值，直到冲突不再发生。这种方法不易产生聚集，但是增加计算时间，同时需要准备许多哈希函数。

链地址法：将所有哈希值相同的Key通过链表存储。key按顺序插入到链表中

建立公共溢出区：采用一个溢出表存储产生冲突的关键字。如果公共溢出区还产生冲突，再采用处理冲突方法处理。

## 13、链表翻转

struct ListNode

{

int val;

struct ListNode \*next;

ListNode(int x) :val(x), next(NULL) {}

}；

ListNode\* ReverseList(ListNode\* pHead)

{

if(!pHead || !pHead->next) return pHead;

ListNode \*pre=nullptr;

ListNode \*p=pHead;

ListNode \*next=pHead->next;

while(p)

{

p->next=pre;

pre=p;

p=next;

if(next)

next=next->next;

}

return pre;

}