# C++

1. C++面向对象的特性是什么？分别介绍下？

C++面向对象的特性有封装、继承、多态；

封装：隐藏类的具体实现方法，只提供接口；

继承：子类可以继承父类，拥有父类的成员和函数；

多态：多态是一个接口多个实现，通过子类重写父类的虚函数，实现了接口重用。提高了代码的复用率和可维护性。

（1）静态多态：静态多态主要是重载和模板，在编译时确定运行时机 （2）动态多态：动态多态主要靠虚函数机制实现，在运行时动态绑定。

**扩展：**虚函数需要虚函数表和虚指针，需要额外的空间，虚指针在类的开头。

重载：重载是指函数名相同、参数列表不同、返回值没有要求、作用域相同；

重写：重写是指一个有子类的基类，子类对父类的虚函数的重新定义。

1. 模板的用法与适用场景？实现原理？

编译器会对函数模板进行两次编译：第一次编译在声明的地方对模板代码本身进行编译，这次编译只会进行语法检查；第二次编译时对代码进行参数替换后再进行编译，生成具体的函数代码。

1. C++重载和重写的区别？

（1）重载是函数名相同，作用域相同，参数列表不同，返回类型不做限制的函数；

（2）重写是子类继承父类时，对父类中的虚函数进行重定义称为重写。

（c中没有重载）

1. 虚函数介绍？

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函数。

1. C++中哪些函数不能被定义为虚函数？

普通函数、内联成员函数、构造函数、友元函数。

1. 如果多重继承和多继承的话，子类的虚函数表是什么样子？

多重继承的情况下越是祖先的父类的虚函数更靠前，

多继承的情况下越是靠近子类名称的类的虚函数在虚函数表中更靠前。

Note：子类中新定义的虚函数在哪里？

1. 编译器处理虚函数表应该如何处理？

编译器处理虚函数表的方法是：

如果类中有虚函数，就将虚函数的地址记录在类的虚函数表中。派生类在继承基类的时候，如果有重写基类的虚函数，就将虚函数表中相对应的函数指针设置为派生类的函数地址，否则指向基类的函数地址。

为每个类的实例添加一个虚表指针，虚表指针指向类的虚函数表。实例在调用虚函数的时候，通过这个虚表指针找到类中的虚函数表，再找到相应函数进行调用。

1. 基类的析构函数一般写成虚函数的原因？

首先析构函数可以写为虚函数，当析构一个指向子类的父类指针时，编译器可以根据虚函数表寻找到子类的析构函数进行调用，从而正确释放子类对象的资源。

如果析构函数不被声明成虚函数，则编译器实施静态绑定，在删除指向子类的父类指针时，只会调用父类的析构函数而不调用子类析构函数，这样就会造成子类对象析构不完全造成内内存泄漏。

1. 构造函数为什么不定义为虚函数？

（1）因为创建对象的时候需要确定对象的类型，而虚函数是在运行时确定类型的。而在构造一个对象时，由于对象还未创建成功，编译器无法知道对象的实际类型，是类本身还是类的派生类。

（2）虚函数调用需要虚表指针，而该指针存放在对象的内存空间中；若构造函数声明为虚函数，那么由于对象还未创建成功，还没有内存空间，更没有虚表指针用来调用构造函数了。

1. 构造函数或者析构函数调用虚函数会怎样？

在构造函数中调用虚函数，由于当前对象还没有构造完成，此时调用的虚函数指向的是基类的函数实现方式。

在析构函数中调用虚函数，此时调用的是子类的函数实现方式。

1. 纯虚函数介绍？

纯虚函数是只有声明没有实现的虚函数，是对子类的约束，是接口继承。

包含纯虚函数的类是抽象类，它不能实例化，是由实现了这个纯虚函数的子类才能生成对象。

使用场景：当这个类本身产生一个实例没有意义的情况下，把这个类的函数实现为纯虚函数，比如动物可以派生出老虎兔子，但是实例化一个动物对象就没有意义。

并且可以规定派生的子类的必须重写某些函数的情况下可以写成纯虚函数。

1. 深拷贝和浅拷贝的区别？（举例说明深拷贝的安全性）

浅拷贝就是将对象的指针进行简单的复制，原对象和副本指向的是相同的资源。

而深拷贝是新开辟一块空间，将原对象的资源复制到新的空间中，并返回该空间的地址。

深拷贝可以避免重复释放和写冲突。例如使用浅拷贝对象进行释放后，对原对象的释放会导致内存泄漏或程序崩溃。

1. 完全二叉树？满二叉树？

完全二叉树就是只有倒数第一层和倒数第二层可以存在叶子节点，并且除了倒数第一层其他层必须达到当前层节点最大数量，倒数第一层节点是从最左边开始放的。满二叉树就是只有倒数一层是叶子节点，并且每一层都达到节点最大数量。

1. C++为什么推荐使用组合而不是继承？

组合关系中各个对象的内部实现是隐藏的，只能通过接口调用，有助于保持每个类的指责单一性。这样类的层次体系和规模都是可控的。

1. new和malloc的区别？

（1）第一点是new是c++中的关键字，而malloc是c语言的库函数

（2）第二点是new操作符申请内存无需指定内存块大小，malloc则需要显示的指出所需内存的大小。

（3）new操作符内存分配成功，返回对象类型的指针，malloc内存分配成功返回的是void\*

（4）new分配失败时会抛出异常，malloc分配内存返回null

1. 既然有了malloc/free，为什么还需要new/delete？

（1）Malloc/free和new/delete都是用来申请和释放内存的

（2）在对非基本数据类型的对象使用的时候，对象创建的时候还需要执行构造函数，销毁的时候要执行析构函数。而malloc/free是库函数，是已经编译的代码，所以不能把构造函数和析构函数的功能强加给malloc/free。

1. free回收的内存是立即返还给操作系统吗？为什么？

不是的，被free回收的内存会首先被ptmalloc使用双链表保存起来，当用户下一次申请内存的时候，会尝试从这些内存中寻找合适的返回。这样就避免了频繁的系统调用，占用过多的系统资源。同时ptmalloc也会尝试对小块内存进行合并，避免过多的内存碎片。

1. 左值和右值？左值引用和右值引用？

左值就是表达式结束之后还存在的变量；右值就是表达式结束后就不存在的临时变量。

左值引用要求右边的值必须可以取地址，右值引用不要求。右值引用可以将右值的生命期延长。

常量左值引用可以像右值引用一样将右值的生命期延长，缺点是只能读不能改。

1. 虚函数调用过程？

先根据虚指针找到虚表，再找到需要调用的虚函数。

1. 内存泄露的定义？如何检测与避免？

内存泄露指已动态分配的堆内存由于某种原因程序未释放或无法释放，造成系统内存的浪费；

造成内存泄漏的原因有：

（1）类的构造函数和析构函数中new和delete没有配套

（2）在释放对象数组的时候没有使用delete[]，使用了delete

（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的时候，如果基类的构造函数不是虚函数，那么子类的析构函数不会被调用，子类的资源没有正确释放，因此造成内存泄漏。

防止内存泄露的办法有（1）避免使用指针（2）使用智能指针；（3）将基类的析构函数设为虚函数

内存泄漏的危害是系统可使用内存 越来越少，也会越来越卡，最终导致系统关机。

1. 栈溢出？有哪些情况会导致栈溢出？

栈溢出是指向栈中写入了超出限定长度的数据，溢出的数据会覆盖栈中其他数据，从而影响程序运行；导致栈溢出的原因有：（1）局部数组过大，（2）递归调用层次太深，（3）指针或数组越界。

1. 常量对象的概念：

（1）c++不允许在常量对象上调用函数，除非成员函数本身也被声明为常量。

（2）对象的构造函数和析构函数不允许被声明为const，但是它们依然可以修改常量对象的初始化

（3）初始化后，不能修改它们的数据成员。

1. 构造函数可以是内联函数吗?

构造函数可以设为内联函数，但是一般不要使用内联，因为构造函数很多情况下会有隐含动作，比如说调用基类成员对象构造函数等。

1. 介绍c++所有的构造函数

C++中的构造函数主要有三种类型：默认构造函数、重载构造函数和拷贝构造函数

（1）默认构造函数是当类没有实现自己的构造函数时，编译器默认提供的一个构造函数

（2）一个类可以有多个重载构造函数，但是需要参数类型或个数不相同。可以在重载构造函数中自定义类的初始化方式。

（3）拷贝构造函数是在发生对象复制的时候调用的。

1. 什么情况下会调用拷贝构造函数？

（1）对象以值传递的方式传入参数

（2）对象以值传递的方式从函数返回

（3）对象需要通过另外一个对象进行初始化

1. 如何理解线程安全？

线程安全是指在多线程的环境下 数据可以被安全有效的访问。（1）可以从多个线程调用，无需调用方有任何操作（2）可以同时被多个线程调用，无需线程之间不必要的交互。

线程安全的容器有哪些？

没有

不线程安全会出现什么问题？

1. C++四种强制类型转换？

Const\_cast将const指针转换成非const指针

Reinterpret\_cast可以改变指针或引用的类型，把指针或引用转换为整型或者把整型转换为指针。

Static\_cast在运行时没有类型的安全检查；可以用于父类与子类指针之间转换时，上行转换时是安全的，下行转换是不安全的。

Dynamic\_cast是运行时处理的，转换时进行类型的安全检查。在进行类的转换时，基类一定要有虚函数并且上行转换和下行转换都是安全的。

1. stl的基本组成？

Stl的基本组成有：**容器、迭代器、算法、仿函数、内存分配器、配接器**；内存分配器给容器分配内存空间，算法通过迭代器访问容器元素，仿函数协助算法完成各种操作，配接器用来套接适配仿函数。

1. 栈内存和堆内存有什么区别？

栈内存是由操作系统分配释放的，堆内存是由程序员分配释放的；栈内存是从高地址往低地址分配的，堆内存是从低地址往高地址分配的；堆内存的效率比栈内存低；堆内存多次的分配释放会产生很多内存碎片。

1. 内存碎片是什么？

内存碎片是指内存分配过程中，产生了不可被用户使用的小分区。 比如说用户申请了21字节的内存空间，操作系统会分配给它24字节的空间，这多出来的3个字节就是内存碎片。

1. 内存碎片在工程中怎么解决？

使用一个内存池，内存池是在真正使用内存之前，先申请分配一定数量、大小的内存块作为备用。当有新的内存需求时，就从内存池中分出一部分内存块，若内存块不够则再继续申请新的内存。

1. C++程序内存结构？全局变量在哪个段？

C++程序内存结构有代码段、数据段、bss段、堆段和栈段；一个可执行程序有代码段、数据段和bss段，在程序运行的时候会使用堆段和栈段。未初始化的和初始化为0的全局变量、静态变量存放在bss区。

1. 程序编译的完整过程？编译过程的具体内容？静态链接和动态链接的而区别？哪种链接更快？

程序编译的过程有：预编译、编译、汇编、链接；（1）预编译是处理“#”号开头的预编译指令，展开宏定义等；（2）编译过程是把预编译生成文件进行词法分析、语法分析和语义分析然后生成汇编文件；（3）汇编过程是把汇编文件转换成机器码文件生成目标文件；（4）链接是把源文件与它们的目标文件链接起来形成可执行文件。

**静态链接和动态链接的区别是**：静态链接把所有的程序模块链接起来形成一个单独的可执行文件，动态链接是把程序拆分成相对独立的模块，在程序运行时再链接到一起；静态链接空间耗费大，但是运行速度快，动态链接节省空间但是运行速度慢，因为是在运行的时候才链接到一起。

1. 编译生成的二进制文件计算机是怎么识别的？

通过控制晶体管的导通和截断来实现的。

1. Main函数之前和之后都会做什么工作？

Main函数之前初始化堆段、栈段、打开输入输出流；

1. 字节序：大端对齐和小端对齐？大端和小端是由什么决定的？如何判断大端还是小端？

大端对齐就是低地址存放高字节，小端对齐是低地址存放低字节；大端和小端是有cpu决定的。使用联合体（union）可以判断大端和小端；（为什么使用联合体可以判断？）因为联合体从低地址开始存放的，并且联合体内部是共享内存空间的。

1. C++中struct和class的区别？

C++中结构体和类都可以继承，结构体默认的继承访问权限为public，类的默认继承访问权限为private，class可以用作模板类形参。

1. struct和union的区别？

结构体各成员拥有自己的内存，联合体各成员共享一块内存，并且同时只有一个成员可以得到这块内存的使用权。结构体还有一个字节对齐的机制。

1. define和const的区别？

联系：它们都是定义常量的一种方法；

区别：

（1）define定义常量没有类型，只是进行了简单的替换，可能有多个拷贝，占用内存空间大；const定义的常量是有类型的，存放在静态存储区，只有一个拷贝，占用内存小

（2）define定义的常量通常在预编译阶段进行替换，而const在编译阶段确定它们的值

（3）define不会进行类型的安全检查，而const会进行类型的安全检查

（4）const可以定义函数，define不可以。

1. define和inline的区别？

（1）宏定义在预编译的时候展开，内联函数在编译的时候展开

（2）内联函数直接被嵌入到目标代码中去，宏定义只是一个简单的文本替换

（3）内联函数有类型安全检查，宏没有

（4）宏不是函数，内联函数是函数

（5）内联函数可以不展开，宏一定要展开

Inline 什么时候会失效？

在内联函数函数体过于复杂的时候，编译器会把该内联函数还原成普通函数。

1. typedef和define的区别？

宏定义是预编译时处理的，只是进行简单的替换，不做类型的安全检查

Typedef是编译时处理的，它时在自己的作用域内给已经存在的类型一个别名

1. const用法？

（1）const修饰变量时，表示是只读的

（2）const修饰类的成员函数时，表示该函数不会修改类中的数据成员，不会调用其他非const的成员函数。

1. Static关键字的作用？（静态全局变量在main函数之前初始化，静态局部变量在第一次调用时初始化。）

Static可以修饰局部变量，全局变量和函数，静态变量存储在静态区中。

对于静态局部变量，它的生存周期比局部变量长，直到程序运行结束而非作用域结束；对于静态全局变量和静态函数，其作用都只是缩小其作用域，让它只在本文件中可见。 在类中修饰变量和函数，表示该静态成员变量和静态成员函数不属于对象，而属于类本身。

Note：静态成员函数访问非静态成员时，要通过对象来引用。

注意和const的区别，const强调值不能被修改，而static强调唯一的拷贝，对所有类的对象都共用。

1. Volatile的作用？

Volatile的意思是脆弱的，表名它修饰的变量的值是十分容易被改变的，所以编译器就不会对这个变量进行优化（cpu的优化是指让改变量存放到cpu寄存器而不是内存），进而提供稳定的访问。每次读取volatile的变量时，系统总是会从内存中读取这个变量，并且将它的值立刻保存。

1. 智能指针介绍一下？循环引用怎么解决？

智能指针有shared\_ptr，weak\_ptr，unique\_ptr。智能指针其实是将指针进行了封装，可以像普通指针一样进行使用，同时可以自行进行释放，避免忘记释放指针指向的内存地址造成内存泄漏。

（1）shared\_ptr可以使得一个对象可以有多个智能指针，当这个对象所有的智能指针被销毁时就会自动进行回收。

（2）unique\_ptr保证同一个对象同一时间只有一个智能指针，不能复制也不能拷贝。

（3）weak\_ptr是为了协助shared\_ptr而出现的。不能访问对象，只能观测shared\_ptr的引用计数，防止出现死锁，解决循环引用的问题。

1. 如何理解智能指针的所有权？

所有权分为单一所有权和共享所有权。单一所有权是指所有权的拥有者有义务释放内存或者转移所有权，同一时刻只有一个所有权拥有者；共享所有权拥有者不必释放内存，引用计数器释放内存，同一时刻可以有多个所有权拥有者。

unique\_ptr管理释放器是编译时绑定；shared\_ptr管理释放器是运行时绑定，使用户重载释放器更方便。

1. Map、multimap、unordered\_map的底层分别是什么数据结构？

Map 的底层是红黑树，multimap也是红黑树，unordered\_map的底层是哈希表。（unordered\_map是无序的，所以底层是哈希表，哈希表的底层是一个数组）

**C++常用容器及其复杂度：**

（1）vector

底层是数组，动态成倍扩容

插入：push\_back() O(1) insert() O(n)

删除：pop\_back() O(1) erase() O(n)

查找：O(1)

（2）list

底层是双向链表

插入：push\_back() O(1) push\_front() O(1) insert() O(1)

删除：pop\_back() O(1) pop\_front() O(1) insert() O(1)

查找：O(n)

（3）deque双端队列

底层是分段连续的线性空间

插入：push\_back() O(1) push\_front() O(1) insert() O(N)

删除：pop\_back() O(1) pop\_front() O(1) erase() O(N)

查找：O(1)

（4）stack栈

底层实现一般是list或者deque

插入：push() O(1)

删除：pop() O(1)

查找：top() O(1)

（5）queue 队列

底层实现一般是list或者deque

插入：push() O(1)

删除：pop() O(1)

查找：front() O(1)

（6）priority\_queue优先队列

底层用堆实现

插入：O(logN)

删除：O(logN)

查找：取堆顶 top() O(1)

（7）map是一个有序映射，存储键值对

底层是红黑树实现

插入：O(logN)

删除：O(logN)

查找：O(logN)

（8） unordered\_map是一个无序映射，存储键值对

底层是哈希表实现，hashcode如何获得？

插入：O(1)

删除：O(1)

查找：O(1)

1. Vector动态扩容机制？

每次容量不够使用时，重新申请一块大小为原来容量两倍的内存，将原容器的元素拷贝至新容器，并释放原空间，返回新空间的指针。 在原来空间不够存储新值时，每次调用push\_back方法都会重新分配新的空间以满足新数据的添加操作。

1. 频繁对vector调用push\_back()对性能的影响和原因？

如果需要频繁插入，最好先指定vector的大小，因为vector在容器大小不够时会重新申请一块大小为原容器两倍的空间，并将原容器的元素拷贝到新容器中，并释放原空间，这个过程时十分耗时和耗内存的。频繁调用push\_back()会使得程序花费很多时间在vector扩容上，会变得很慢。这种情况可以考虑使用list。

1. 迭代器分为哪几种类型？list和vector的迭代器有什么区别？

输入迭代器、输出迭代器、const迭代器、随机访问迭代器

Vector拥有一段连续的内存空间，能很好的支持随机存取，因此vector的迭代器支持+、+=、<等操作

List的内存空间是不连续的，不支持随机访问，因此list的迭代器不支持+，+=，<等。

Vector和list的迭代器都重载了++运算符。

1. Vector迭代器会失效吗？什么情况下迭代器失效？

会

（1）当vector插入的时候，如果原来的空间不够，会将申请新的内存并将原来的元素移动到新的内存，此时指向原内存地址的迭代器就失效了，first和end迭代器都失效。

（2）vector在插入的时候，end迭代器肯定失效

（3）vector删除的时候，被删除元素以及它后面的所有元素迭代器都失效。

1. Unordered\_map和map的区别？

（1）unordered\_map是使用哈希表实现的，占用内存比较多，查询速度快，是O（1）的时间复杂度。它内部是无序的，需要实现==操作符。

（2）map底层是使用红黑树实现的，插入删除查询时间复杂度都是O（logn），内部是有序的，因此需要实现比较操作符（<）。

1. 二叉平衡搜索树和红黑树的优缺点？应用场景？

二叉平衡搜索树搜索数据效率高，但是在插入删除数据的时候需要维护树的平衡性导致效率低，红黑树是一种折中方法，插入删除效率比二叉平衡树高，但是搜索数据的效率没有二叉平衡树高；二叉平衡搜索树适用于搜索比插入删除次数多的情况，红黑树适用于搜索和插入删除次数差不多的情况。

1. 红黑树和二叉平衡搜索树插入删除查询的时间复杂度？左右子树最大高度差为多少

红黑树是一颗近似平衡的二叉搜索树，红黑树查找、搜索、删除的时间复杂度为O(logn)；二叉平衡搜索树也是。二叉平衡搜索树左右子树的最大高度差为1，红黑树的左右子树最大高度差为2，红黑树在插入删除的时候至多旋转三次达到平衡。h(二叉平衡搜索)<=2logn+1,h(红黑树)<=2log(n+1)。

1. this指针指向的内容是什么？this指针什么时候创建的？

this指针指向的对象的地址；this指针在非静态成员函数开始时创建，在非静态成员函数结束时释放。

1. 设计模式
   1. 工厂模式：使用一个工厂类，对同一接口的一些类进行实例创建。
   2. 单例模式：只能生成一个实例化对象。使用场景：windows的任务管理器和回收站，以及应用程序的日志应用。
   3. 装饰模式：动态地给一个对象增加一些职责，比使用子类的方式更加灵活，但是会使程序更加复杂。
   4. 观察者模式：观察者模式主要处理一对多的依赖关系，当被观察者的状态改变时会通知观察者并更新。
2. 指针和引用的区别？
   1. 指针占用固定的内存空间，引用只是被引用对象的一个别名
   2. 指针操作对象需要解引用，引用可以直接操作对象
   3. 指针可以初始化为空，引用不可以
   4. 指针在使用过程中可以指向其他对象，引用不可以
   5. 指针自增和引用自增意义不同
3. 指针和数组的区别？

数组是用于储存多个相同类型数据的集合，指针相当于一个变量，存放的是其他变量在内存中的地址。

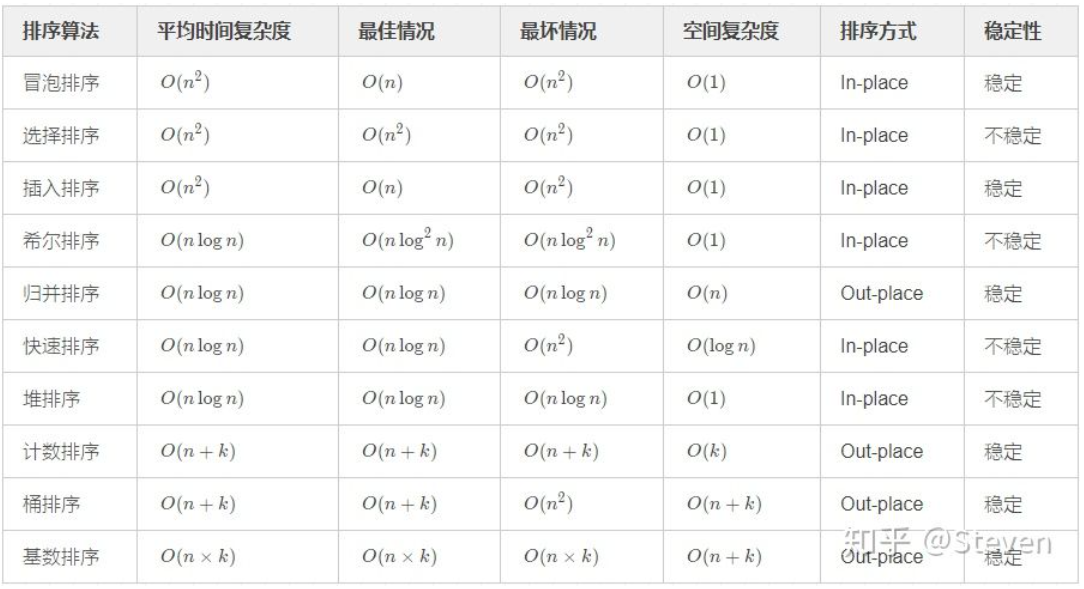
（1）同类型指针变量可以互相赋值，数组不行，需要一个个赋值

（2）数组的存储空间不是在静态区就是在栈上，指针的存储空间不能确定。

（3）sizeof 指针大小在32位平台下一直为4，在64为平台下一直为8。

1. 变量的声明和定义的区别？

为变量分配地址和存储空间的称为定义，不分配地址的称为声明。一个变量可以在多个地方声明，只能在一个地方定义。加入extern修饰的是变量的声明，说明此变量将在文件以外或在文件后面部分定义。

1. 排序算法平均时间复杂度？

快排的原理？

快排思想是将待排序数组分割成两部分，其中一部分比另外一部分小，再把这两部分按照上述的思想继续分割，整个过程可以递归进行，最终一个有序的数组。

快排的优点？

在待排序的数据规模大的时候，快排相比于其他排序的效率高很多。

堆排序的原理？

堆排序的优点？

1. 哈希冲突的解决算法？拉链法链表过长怎么办？

哈希冲突的解决算法有拉链法（拉链法是采用链表的结构去存取发生哈希冲突的关键字）和开发寻址法，为了更好的处理哈希冲突，一般会使用二次探测和双重散列法。（1）二次探测是指二次方探测位置（2）双重散列是指利用多个散列探测空槽，一个散列找不到就找第二个继续计算，找空位，如此循环下去。

拉链法链表达到一定长度时，就在链表后面使用红黑树存储。

**开发寻址法和拉链法的优缺点？**

1. 一致性哈希？

一致性哈希是处理分布式存储的。

1. 讲一讲内存对齐？如何取消内存对齐？

内存对齐可以提高效率，因为CPU访问对齐的内存只需要一次，访问未对齐的内存需要两次；

使用#pragma pack(1)可以取消内存对齐，有了这条命令之后，是按照1字节对齐的

1. A\* a=new A;a->i=10,在内核的内存分配发生了什么？

第一步是给指针a分配4字节的空间，第二步是在堆区申请一个类A大小的空间，指针a的内存存放类A申请到的地址，第三步是a->i是通过a的值以及i在类A中偏移量找到a->i的地址，然后给这个内存空间赋值。

1. static和const是否可以同时修饰成员函数？（static和const可以同时修饰成员变量）

Static和const不可以同时修饰成员函数，因为static函数没有this指针，而const是承诺了this指针指向的是常量，所以用法冲突了。

1. 拷贝构造函数为什么传引用而不是传值？（拷贝构造函数需要参数是同类对象）

利用值传递的话，会导致无限递归调用拷贝构造函数。

1. 构造函数初始化和用初始化列表初始化的优缺点？

初始化列表初始化方法在以下情况必须使用：

（1）类中存在常量和引用

（3）类的基类或成员类没有定义默认构造函数；

使用初始化列表初始化可以提高效率，原因是只调用了一次拷贝构造函数，而在构造函数中赋值调用了一次构造函数和一次赋值运算函数。

1. 拓扑排序？

在一个有向无环图中，对所有的节点进行排序，要求没有一个节点指向它之前的节点。 先统计所有节点的入度，入度为0的节点分离出来，然后把这个节点指向的节点入度减1，直到所有的节点被分离出来。 如果最后不存在入度为0的节点，说明有环，不存在拓扑排序。 场景应用：编译器判断编译顺序。

1. 惊群效应？
2. C++对象生成时，该对象的内存结构是怎样的？
3. C++生成exe可执行文件需要包含哪些要素？
4. 类中函数如何找到地址的？虚函数怎么找到的？

类中函数在编译的时候就已经编译到汇编代码了，然后运行的时候在符号表里找。

虚函数是使用虚表指针找到虚表，然后在虚表中找调用的函数

1. Vector<int> a;sizeof(a)是多少？
2. Vector中push\_back()的时间复杂度是多少？

常量级的时间复杂度

1. Map 插入怎么判断是否成功？

Pair<iterator,bool> 第二个元素显示是否插入成功

1. 计算类大小

空类大小为1，这是为了让对象的实例能够互相区别。当该空类作为基类时，该类的大小就会被优化为0，子类的大小就是子类本身的大小。A a 这样类的实例大小就是类的大小， A\* a就是指针的大小。

静态成员放在静态区中，不占用类的大小，普通函数也不占用类的大小。

1. 对象复用的了解，零拷贝的了解

对象复用指的是设计模式，对象可以采用不同的设计模式达到复用的目的，最常见的就是继承和组合模式。

零拷贝指的是在进行操作时，避免cpu从一处存储拷贝到另一处存储。

1. 调试程序的方法？

（1）设置断点进行调试

（2）打印log进行调试

（3）打印中间结果进行调试

1. C++调用惯例？（C++函数调用的压栈过程）

（1）从栈空间分配存储空间

（2）从实参的存储时间复制值到形参栈空间

（3）进行运算

1. String的底层实现？

String对char\*进行了封装

String可以进行动态扩展，在每次扩展的时候另外申请一块原空间大小两倍的空间，然后将原字符串拷贝过去，并加上新增的内容。

1. 友元函数和友元类？（55）
2. Stl中 sort()算法是什么实现的，stable\_sort()呢？

Sort是快速排序和插入排序结合的方式实现的，stable\_sort()是归并排序。

# 操作系统

1. I/O多路复用？

I/O多路复用就是通过一种机制监视多个描述符，一旦某个描述符就绪，能过通知程序进行相应的读写操作。

1. I/O多路复用什么场景会用到？

使用 IO 多路复用，如果操作系统接收到 1000 个请求，应用程序就不需要开启 1000 个线程来处理

1. select和epoll的区别？知识点说明：select、poll、epoll都是I/O多路复用的机制。（但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）

Select的时间复杂度为O(n)，只知道有I/O事件发生，但是不知道时哪几个流，所以需要轮询所有流。epoll的时间复杂度为O（1），知道I/O事件发生并且也知道是哪个流，直接对该流进行操作。

如何用select实现一个定时器？

1. 阻塞IO和非阻塞IO？

1.阻塞IO： 资源不可用时，IO请求一直阻塞，直到反馈结果（有数据或超时）。  
2.非阻塞IO：资源不可用时，IO请求离开返回，返回数据标识资源不可用

1. 段页式存储管理？内存碎片？

段页式存储管理是操作系统管理内存的一种方式，它把用户程序分成若干个段，再把每个段分成若干个页，并为每个段赋予一个段号。

内存碎片就是指在储存分配过程中产生不能供用户使用的小分区称为内存碎片。

1. 进程和线程的区别？

（1）第一点是进程是cpu资源分配的基本单位，线程是cpu调度和分配的基本单位；（2）第二点是一个进程可以有多个线程，且至少有一个线程，而一个线程只属于一个进程；（3）第三点是多进程之间各自拥有独立的资源，而一个进程的多个线程共享进程的资源；（4）第四点是进程的创建、销毁和切换开销比线程大；（5）第五点是线程之间通信比进程之间的通信方便。（线程切换只需保存寄存器状态和栈，而进程切换需要切换页表）

1. CPU为什么需要使用进程？

多机分布用多进程，多核分布用多线程。

1. CPU为什么需要使用线程？

提高并发

1. 多进程通信方式？

**通信方式**：（1）管道（2）系统IPC（3）套接字socket

1、管道是半双工的，管道分为无名管道和有名管道，无名管道只能用于有血缘关系的进程通信，有名管道没有这个限制；2、系统IPC有消息队列、信号量、信号和共享内存，消息队列是消息的链接表（存放在内核中）；信号量是一个计数器，用来控制多个进程对共享资源的访问，用于实现进程间互斥与同步；信号是通知接收进程某个事件已经发生；共享内存是最快的系统IPC，使得多个进程可以访问一块内存，不同进程可以看到对方进程对共享内存中数据的更新；3、socket是用于不同主机之间的进程通信。

消息队列？至少一次&精确一次

1. 多线程通信方式？

通信方式：（1）临界区（2）信号量（3）信号（4）互斥量

1. 多线程同步的方式？线程同步的作用？

互斥量、信号量和条件变量；（1）互斥量是指在同一时刻只有一个线程可以访问共享资源，其他未能进行加锁的线程进入睡眠状态（2）信号量是一个计数器，控制申请资源的最大线程数；（3）条件变量是某共享变量达到某个值的时候 唤醒等待该变量的一个或多个线程。

线程同步可以使线程一个一个排队访问共享资源，解决线程冲突的问题。

1. 进程线程相关知识：

一个标准的线程由线程ID，当前指令指针（PC），寄存器和堆栈组成，是系统中调度和分配的基本单位，线程自己不拥有系统资源，与同一进程中所有线程共享进程所拥有的资源，共享整个虚拟地址空间。 多线程之间共享代码段、数据段、bss段和堆段，每个线程私有的栈段和寄存器。

1. 多线程开发需要注意什么？

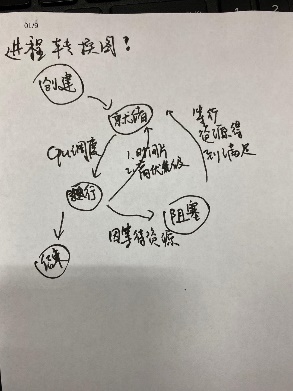
多线程开发需要需要注意对数据的保护。多线程与单线程相比，主要的特点是同一份数据会被多个线程访问，而每个线程执行的程序代码是有可能从中间中断的，因此如果处理不好，数据的改变可能不是我们期望的样子。

1. 多线程适用的场景？

需要提高并发的场景都可以使用多线程，比如说流水线作业。

1. 线程是不是越多越好？

不是，线程多了可以提高系统的并发性，但是不是越多越好。每个线程都要占用内存，多线程就意味着更多内存资源被占用。

1. 进程状态（转换）图？动态就绪、静态就绪、动态阻塞、静态阻塞？

动态就绪：在内存中，有cpu调度即可执行

静态就绪：在外存中，执行需要从外存中调入内存，然后cpu调度执行

动态阻塞：在内存中，进程由于某种原因被阻塞了

静态阻塞：在外存中，进程由于某种原因被阻塞了

1. 僵尸进程和孤儿进程？

子进程先于父进程推出后，子进程的pcb需要父进程释放，但是父进程还没有释放，这样的子进程就成为僵尸进程。

孤儿进程就是父进程退出，但是子进程还在运行，那么这些子进程会成为孤儿进程。孤儿进程会被init进程收养，并由init进程释放它们的pcb。

1. **死锁是什么？**（进程、线程都会发生死锁，只是发生死锁的基本单位不同）

死锁是两个或多个进程因竞争资源而发生互相等待的情况。

**死锁产生原理？**

当一组进程中每个进程都在等待某个事件发生，而只有这组进程中的其他进程才能触发该事件，称这组进程发生了死锁。

**死锁产生的条件？**

（1）互斥条件：同一时刻只有一个进程可以访问某资源 （2）请求与保持条件：进程因请求资源而阻塞时，已有的资源不会释放 （3）不可剥夺条件：进程已获得的资源在未使用完之前 不可被剥夺 （4）环路等待条件：发生死锁是，必然存在 进程-资源的环形等待链

**死锁的解决方法？**

（1）破坏请求条件：资源一次性分配，就不会有请求了 （2）破坏保持条件：若一个资源得不到分配也不给该进程分配其他资源 （3）破坏不可剥夺条件：若进程获得一部分资源，但是得不到其他资源，释放进程已有资源 （4）系统给资源按类别编号，分配资源是升序分配，释放资源是降序释放。

**银行家算法：（解决死锁问题）**

银行家算法是一个动态避免死锁的算法，通过对资源的仔细分配以避免死锁。

1. 介绍一下锁？（线程）

锁有互斥锁、自旋锁、条件变量；互斥锁是线程在进入临界区的时候，对互斥锁上锁，其他未能执行上锁操作的线程进入睡眠状态，直到锁被释放；自旋锁是线程进入临界区的时候，对自旋锁上锁，其他未能执行锁操作的线程进入自旋状态，直到锁被释放；条件变量是临界区的某个共享变量达到一定数值之后唤醒等待它的线程。

1. 什么时候用自旋锁？什么时候用信号量？

Io密集型用自旋，cpu密集型操作用信号量。//

1. 讲一讲虚拟内存？虚拟内存的优点？

虚拟内存是为了防止不同进程同一时刻在同一物理内存中运行，导致对物理内存的争夺和践踏而产生的。虚拟内存的优点是（1）在逻辑上扩大内存容量；（2）实现了内存保护，每个进程运行在各自的虚拟内存中，互不干扰；（3）通过虚拟内存实现进程间通信；（4）公平的内存分配，每个进程拥有同等大小的虚拟内存空间。

虚拟内存地址的大小是与地址总线位数相关，物理内存地址的大小跟物理内存条的容量相关

1. 内存置换算法？段页式内存存储管理，一个页通常有多大？（4KB）

当发生缺页中断时，就要从磁盘上将需要的页面调入内存中；如果内存中没有足够的空间，就需要在现有的页面中选择一个替换。 页面置换的目标是降低随后发生缺页中断的次数。 内存置换算法有LRU算法（最近最少使用）：置换最近一段时间以来最久没有被访问的页面； （FIFO）先进先出算法：置换最先被调入内存的页面。

页表大和页表小分别有什么优缺点？（页表项和内部内存碎片的博弈）

页表大的话可以减少页表项，但是内部内存碎片浪费严重，大的页面不能体现最近访问，缺页率高；页表小可以减少内部碎片的浪费，缺页率低，但是会导致页表项过多。

虚拟内存大小与操作系统位数（地址总数）有关。32位虚拟内存大小为4G，64位虚拟内存大小位128G。

1. 分页的时候物理页大小和虚拟内存页大小不一样行吗？物理内存比虚拟内存小怎么办？

行；缺页中断加内存置换算法

1. 内存池如何实现？
2. 对象池如何实现？
3. 用户态和内核态？如何切换？什么情况下会进行切换？
4. 内核缓冲区满了怎么办？

用二级缓冲，给每个连接维护一个接收缓冲区。

# 计算机网络

1. TCP、UDP使用场景？

TCP适用于文件传输、发送接收邮件；UDP适用于语音、视频、直播。

1. 网络的四层结构？

应用层、传输层、网络层、数字链路层

1. OSI七层模型？

表现层、应用层、会话层、传输层、网络层、链路层、物理层

1. TCP三次握手、四次挥手的过程？

**三次握手：**首先客户端向服务端发送标志位SYN=1，随机数seq=x；然后服务端回复SYN=1，ACK=x+1，以及一个随机数seq=y；第三步是客户端发送确认码ACK=y+1。（在第三次握手时可以传送报文）

**四次挥手：**第一步是客户端向服务端发送标志位FIN=1，ACK=y+1，seq=x+2；第二步是服务端发送确认码ACK=x+3；第三步是服务端发送FIN=1，随机数seq=y+1；第四步是客户端回复ACK=y+2。

为什么是三次握手不是两次？

因为TCP连接是全双工的，三次握手才能保证两个方向都是联通的。

为什么是四次挥手不是三次挥手？

因为服务端向客户端确认关闭的时候，服务端可能还有未发送完的数据，需要等服务端发送完了，服务端这边才能关闭发送。

四次挥手Time\_wait的作用？

Time\_wait的时间为2msl，msl是报文最大生存时间，可为1min，2min。time\_wait是为了保证客户端的应答ack可以到达服务端，若没有到达，服务端可以重传第三次挥手的fin报文。

多个连接出现time\_wait（主动关闭的一方）状态是在什么情况下发生的？应该如何解决？

有太多的close\_wait（被动关闭的一方）的原因以及解决方法？

Close\_wait状态时tcp四次挥手的时候收到FIN但是没有发送自己的FIN时出现的，服务器出现大量close\_wait状态的原因有两种：

（1）服务器内部业务处理占用了过多的时间，都没能处理完业务；或者还有数据需要发送；或者服务器的业务逻辑有问题，没有执行close方法

（2）服务器父进程派生出子进程，子进程继承了socket，收到FIN的时候子进程处理但父进程没有处理该信号，导致socket的引用不为0无法回收。

处理方法：

（1）停止应用程序

（2）修改程序里的bug

Tcp状态转换？

TCP连接已经建立的情况下，主机网络断掉，则对端能感知到这个事件吗？  
TCP有一个保活计时器，一般两个小时内没收到数据，就会每隔75s发送一个探测报文，如果发送了10个报文没有得到响应，就会关闭连接。

Tcp处理包乱序问题？

使用顺序号处理乱序问题，顺序号数字不断累加避免了乱序。

使用确认码解决网络传输过程中数据丢包的问题。

Tcp粘包与避免？

因为tcp采取的是流式传输，所以接收端在一次接收的时候有可能一次接收多个包。而tcp粘包就是发送方的若干个数据包到达接收方的时候粘成了一个包。多个包首尾相接，无法区分。

导致tcp粘包的原因有三点：

（1）发送端等待缓冲区满才进行发送，造成粘包

（2）接收方来不及接受缓冲区内的数据，造成粘包

（3）由于tcp协议在发送较小的数据包的时候，会将几个包合成一个包后发送

避免粘包的措施：

（1）通过编程，强制tcp发生数据传送，不必等到缓冲区满

（2）优化接收方接收数据的过程，使其来得及接收数据包，包括提高接收进程优先级等

（3）设置固定长度的报文或者设置报文头部记录报文长度

Tcp封包和拆包？

因为tcp式无边界的流传输，所以需要对tcp进行封包和拆包，确保发送和接收的数据不粘连。

（1）封包：封包就是在发送数据报的时候为每个tcp数据包加上一个包头，将数据报分为包头和包体两个部分。包头是一个固定长度的结构体，里面包含该数据包的总长度

（2）拆包：接收方在接收到报文后提取包头中的长度信息进行截取

Tcp请求头？

1. http请求方法？http状态码？

http请求方法有Get、post、put、delete四种方法。

http状态码：

100-continue

200-请求成功

301-资源被永久转到其他url

400-请求无效

403-禁止访问

404-请求的资源不存在

500-内部服务器错误

1. get 和 post请求差别是什么？

对于get请求方式会把http header和data一起传送出去，服务器响应200；而post是先传送header，等到服务器响应100，再传送data，服务器响应200。（1）get 参数通过url传递，post 参数通报文传输；（2）get参数传递有长度限制，post没有；（3）get产生一个TCP数据包，post产生两个TCP数据包。

1. get和post使用场景？

Post使用场景为 （1）请求的结果有持续性副作用，例如数据库添加新的数据行 （2）若使用get方法，会让url过长则使用post （3）要传送的数据采用的不是7位ascii编码。

Get使用场景为（1）请求是为了查找资源 （2）请求结果无持续性副作用。

1. 在浏览器中输入网址之后都执行哪些操作？DNS的查询流程？DNS劫持？

首先是把url通过DNS解析成IP地址，浏览器根据IP地址和默认端口80浏览器和服务端建立TCP连接，浏览器和服务器传输数据，释放TCP连接，浏览器显示内容。

**DNS查询流程**是首先按顺序查询浏览器缓存、操作系统缓存和LDNS中是否命中该域名，若没有命中向root server发送请求，root server返回查询域名的主域名服务器的地址，LDNS向主域名服务器发送请求，主域名服务器接受请求并返回一个Name Server，Name Server根据映射关系找到IP地址并返回给LDNS，LDNS缓存域名及其对应IP地址。

**DNS劫持**就是被恶意修改操作系统中的缓存，即hosts文件信息，通过修改hosts文件信息把特定域名解析到他指定的ip地址。

1. IP地址到MAC地址转换的协议？（ARP）

通过arp协议； （MAC地址到IP地址转换的协议RARP（逆地址解析协议））

ARP协议工作原理？

（1）每个主机都会在自己的ARP缓冲区建立一个ARP列表，表示IP地址和MAC地址的映射关系 （2）主机新加入网络时会发送ARP报文把自己的IP地址和MAC地址的映射关系广播给其他主机 （3）网络上主机接受ARP报文时会更新自己的ARP缓冲区，将新的映射关系更新到自己的ARP列表中 （4）某个主机想要发送数据时，会在自己的ARP缓冲区查找是否有目的的主机IP地址的MAC地址，若有则发送数据，若没有就会向本网络的所有主机发送ARP报文，该ARP报文包含源主机IP，源主机MAC，和目的主机IP等信息 （5）当本网络主机接收到ARP报文时 会先检查数据包中IP地址是不是自己的IP地址，若不是则忽略该报文，若是则首先从数据包中取出源主机的IP和MAC地址，更新自己的ARP缓冲区，然后将自己的MAC地址写到ARP响应包中，告诉源主机自己的MAC地址 （6）源主机收到ARP响应包后，将目的主机的MAC地址和IP地址写入自己的ARP列表，并利用此信息发送数据，若一直没有收到ARP相应包则表示查询失败。

1. http协议？

http协议是从服务器传送超文本到本地浏览器的传送协议。http是一个基于tcp/ip通信协议来传递数据。

1. HTTP协议和HTTPS协议的区别？（HTTP协议是无连接、无状态的）

（1）HTTP是以明文的方式在网络中传播，HTTPS是经过TLS加密后的，据有更高的安全性；（2）HTTPS需要服务端申请证书，浏览器端安装对应的根证书；（3）HTTPS在TCP三次握手之后还需要进行SSL的handshake加密，协商加密的对称密钥。

1. HTTPS SSL层建立安全会话的过程（保证可靠性）？数字证书包含的内容？为什么要同时使用非对称加密和对称加密？非对称加密算法有哪些（RSA、DSA）？对称加密算法有哪些（DEA）？

（1）客户端发送https的请求（2）服务端收到请求后，向客户端发送证书和公用密钥（3）客户端用根证书对发来的证书进行验证，如果验证不通过，访问终止（4）客户端随机生成一个对称密钥，使用公钥进行加密，并向服务器发送（5）服务端收到后加密后的对称密钥使用私有密钥进行解密（6）双方都有了对称密钥可以进行加密通信了。

客户端如何验证服务端的第三方CA认证证书？

通过本地根证书的CA公钥解密数字摘要，看是否匹配。

**数字证书包含**（1）证书拥有者的姓名和公钥（2）公钥有效期（3）颁发数字证书的单位及其数字签名（4）数字证数的序列号

非对称加密加密解密时间长，只适合短数据的传输，对称加密速度快，适合长数据的传输。

1. https使用get请求时，url是否会加密？

加密了；客户端首先和服务端建立一个加密的tcp连接（通过ssl/tls协议），然后客户端通过该加密的tcp连接发送http请求。

1. 几种HTTP协议版本之间的区别？

http1.0和http1.1的区别： http1.0传输一次数据建立一次连接，数据传输完释放连接；http1.1支持长连接。

1. TCP和UDP的区别？

（1）从连接方式上讲，TCP是面向连接的传输层协议，即传输数据之前需要先建立连接，UDP无连接；

（２）从服务对象上讲，TCP只支持点对点的服务，UDP支持一对一，一对多，多对一，多对多的交互通信；

（3）从可靠性上来讲，TCP是可靠传输，UDP尽最大努力交付；

（4）从拥塞控制与流量控制上来讲，TCP有拥塞控制和流量控制，数据传输更加安全，UDP无拥塞控制，网络拥塞并不会影响源主机的发送效率；

（5）从报文长度上来讲，TCP是动态报文长度，UDP是面向报文的，对报文不合并、不拆分。

总的来说，TCP可靠但是传输速度慢，UDP不可靠但是传输速度块。

**如何在不可靠的网络上进行可靠传输**？

利用自动重传请求（ARQ），可以把不可靠的传输变成可靠传输；自动重传机制是在发送方发送报文之后，启动一个重传计时器，若超过计时器时间还未收到确认应答，则重传报文。

**流量控制和拥塞控制？**

**流量控制**是通过滑动窗口机制实现的，接收方在返回的数据中包含了自己接收数据窗口的大小，告诉发送方自己可以接受多少数据。如果窗口为0，那么发送方将停止发送数据，并定期向接收端发送窗口探测报文，让接收端把窗口大小告诉发送端。 **拥塞控制**有慢启动过程和拥塞避免，慢启动过程就是把拥塞窗口的大小设为1，以后每收到一次确认应答，窗口大小乘2；拥塞避免就是设置一个慢启动阈值，当拥塞窗口大小超过阈值，变为加法增加；在遇到拥塞时（超时重传）会把阈值设置为当前窗口的一半，窗口大小设为1，重新开始慢启动过程。

**快重传**

发送方收到三次重复确认应答之后，阈值设为当前阈值的一半，窗口大小设置为阈值+3。

1. TCP如何保证可靠性？

（1）序列号、确认应答、超时重传

（2）窗口机制、快速重传

（3）拥塞窗口

# 数据库

1. 数据库索引底层为什么使用b+树？

B+树的非叶子节点存放的是索引，叶子节点存放的是数据；B+树只需遍历叶子节点即可实现整棵树的遍历，而b树必须使用中序遍历按序扫库。B+树支持范围查询十分方便。

B树和b+树的区别？

B+树的非叶子节点存放的是索引，叶子节点存放的是数据。B树的节点存放的是key和value。

B+树和二叉树的区别？

1. 数据库四种隔离级别？

（1）读提交：

（2）读未提交：

（3）可重复读：

（4）序列化：

1. 数据库事务？

事务就是一组逻辑操作的集合。

1. 事务ACID特性？
2. 行锁表锁？
3. 乐观锁和悲观锁？

# 算法

1. 给一个大数组，给一个数k，求该数组中任意一对符合和为k的数？
2. K个链表合并？
3. 最大子数组和，最大两个子数组和