史上最全C/C++面试八股文，一文带你彻底搞懂C/C++面试！

前述：本文初衷是为了总结本人在各大平台看到的面经，我会在本文持续更新我所遇到的一些C++面试问题，如有错误请一定指正我。

目录

1.讲一讲封装、继承、多态是什么?

封装：将具体实现过程和数据封装成一个函数，只能通过接口进行访问，降低耦合性，使类成为一个具有内部数据的自我隐藏能力、功能独立的软件模块。意义：保护或防止代码在无意之中被破坏，保护类中的成员，不让类中以外的程序直接访问或者修改，只能通过提供的公共接口访问。

继承：子类继承父类的特征和行为，复用了基类的**全体数据**和**成员函数**，具有从基类复制而来的**数据成员**和**成员函数**（基类私有成员可被继承，但是无法被访问），其中**构造函数、析构函数、友元函数、静态数据成员、静态成员函数**都不能被继承。基类中成员

的访问方式只能决定派生类能否访问它们。增强了代码耦合性，当父类中的成员变量或者类本身被final关键字修饰时，修饰的类不能被继承，修饰的成员变量不能重写或修改。意义：基类的程序代码可以被派生类服用，提高了软件复用的效率，缩短了软件开发的周期

多态：不同继承类的对象对同一消息做出不同的响应，基类的指针指向或绑定到派生类的对象，使得基类指针呈现不同的表现形式。意义：对已存在的代码具有可替代性，对代码具有可扩充性，新增子类不会影响已存在类的各种性质，在程序中体现了灵活多样的操作，提高了使用效率，简化了对应用代码的编写和修改过程。

2.多态的实现原理（实现方式）是什么？以及多态的优点（特点）？

实现方式：多态分为**动态多态**（动态多态是利用虚函数实现运行时的多态，即在系统编译的时候并不知道程序将要调用哪一个函数，只有在运行到这里的时候才能确定接下来会跳转到哪一个函数。）和**静态多态**（又称编译期多态，即在系统编译期间就可以确定程序将要执行哪个函数），其中**动态多态**是通过虚函数实现的，虚函数是类的成员函数，存在存储虚函数指针的表叫做 虚函数表 ，虚函数表是一个存储类成员虚函数的指针，每个指针都指向调用它的地方，当子类调用虚函数时，就会去虚表里面找自己对应的函数指针，从而实现“谁调用、实现谁”从而实现多态。而**静态多态**则是通过函数重载（函数名相同，参数不同，两个函数在同一作用域），运算符重载，和重定义（又叫隐藏，指的是在继承关系中，子类实现了一个和父类名字一样的函数，（只关注函数名，和参数与返回值无关）这样的话子类的函数就把父类的同名函数隐藏了。隐藏只与函数名有关，与参数没有关系.）来实现的。

优点：加强代码的可扩展性，可替换性，增强程序的灵活性，提高使用效率，简化对应用代码的编写和修改过程。

3.final标识符的作用是什么?

放在类的后面表示该类无法被继承，也就是阻止了从类的继承，放在虚函数后面该虚函数无法被重写，表示阻止虚函数的重载

4.虚函数是怎么实现的？它存放在哪里在内存的哪个区？什么时候生成的

在C++中，虚函数的实现原理基于两个关键概念：虚函数表和虚函数指针

虚函数表：每个包含虚函数的类都会生成一个虚函数表，其中存储着该类中所有虚函数的地址。虚函数表是一个由指针构成的数组，每个指针指向一个虚函数的实现代码。

虚函数指针：在对象的内存布局中，编译器会添加一个额外的指针，称为虚函数指针或虚表指针。这个指针指向该对象对应的虚函数表，从而让程序能够动态的调用虚函数。

当一个基类指针或引用调用虚函数时，编译器会使用虚表指针来查找该对象对应的虚函数表，并根据函数在虚函数表中的位置来调用正确的虚函数。

在编译阶段生成，虚函数和普通函数一样存放在代码段，只是它的指针又存放在了虚表之中。

5. 智能指针 的本质是什么，它们的实现原理是什么?

智能指针本质是一个封装了一个原始C++指针的类模板，为了确保动态内存的安全性而产生的。实现原理是通过一个对象存储需要被自动释放的资源，然后依靠对象的析构函数来释放资源。

6. 匿名函数 的本质是什么？他的优点是什么?

匿名函数**本质**上是一个对象，在其定义的过程中会创建出一个栈对象，内部通过重载()符号实现函数调用的外表。

优点：使用匿名函数，可以免去函数的声明和定义。这样匿名函数仅在调用函数的时候才会创建函数对象，而调用结束后立即释放，所以匿名函数比非匿名函数更节省空间。

7.右值引用是什么，为什么要引入右值引用?

右值引用是为一个临时变量取别名，它只能绑定到一个临时变量或表达式（将亡值）上。实际开发中我们可能需要对右值进行修改（实现移动语义时就需要）而右值引用可以对右值进行修改。

为什么:

1.为了支持移动语义，右值引用可以绑定到临时对象、表达式等右值上，这些右值在生命周期结束后就会被销毁，因此可以在右值引用中窃取其资源，从而避免昂贵的复制操作，实现高效的移动语义。

2.完美转发：右值引用可以绑定到任何类型的右值上，可以将其作为参数传递给函数，并在函数内部将其“转发”到其他函数中，从而实现完美转发。

3.拓展可变参数模板，实现更加灵活的模板编程。

8.左值引用和指针的区别?

是否初始化：指针可以不用初始化，引用必须初始化

性质不同：指针是一个变量，引用是对被引用的对象取一个别名

占用内存单元不同：指针有自己的空间地址，引用和被引用对象占同一个空间。

9.指针是什么?

指针全名为指针变量，计算机在存储数据是有序存放的，为了能够使用存放的地址，就需要一个地址来区别每个数据的位置，指针变量就是用来存放这些地址的变量。

10.weak_ptr真的不计数？是否有计数方式，在哪分配的空间。

计数，控制块中有强弱引用计数，如果是使用make_shared初始化的函数则它所在的控制块空间是在所引用的shared_ptr中同一块的空间，若是new则控制器所分配的内存与shared_ptr本身所在的空间不在同一块内存。

11.malloc的内存分配的方式，有什么缺点?

malloc并不是系统调用，而是C库中的函数，用于动态内存分配，在使用malloc分配内存的时候会有两种方式向操作系统申请堆内存

方式1：当用户分配的内存小于128KB时通过brk()系统调用从堆分配内存，实现方式：将堆顶指针向高地址移动，获取内存空间，如果使用free释放空间，并不会将内存归还给操作系统，而是会缓存在malloc的内存池中，待下次使用

方式2：当用户分配的内存大于128KB时通过mmap()系统调用在文件映射区域分配内存，实现方式为：使用私有匿名映射的方式，在文件映射区分配一块内存，也就是从文件映射区拿了一块内存，free释放内存的时候，会把内存归还给操作系统，内存得到真正释放

缺点：容易造成内存泄漏和过多的内存碎片，影响系统正常运行，还得注意判断内存是否分配成功，而且内存释放后（使用free函数之后指针变量p本身保存的地址并没有改变），需要将p的赋值为NULL拴住野指针。

11.1为什么不全部使用mmap来分配内存?

因为向操作系统申请内存的时候，是要通过系统调用的，执行系统调用要进入内核态，然后再回到用户态，状态的切换会耗费不少时间，所以申请内存的操作应该避免频繁的系统调用，如果都使用mmap来分配内存，等于每次都要执行系统调用。另外，因为mmap分配的内存每次释放的时候都会归还给操作系统，于是每次mmap分配的虚拟地址都是缺页状态，然后在第一次访问该虚拟地址的时候就会触发缺页中断。

11.2为什么不全部都用brk

如果全部使用brk申请内存那么随着程序频繁的调用malloc和free，尤其是小块内存，堆内将产生越来越多的不可用的内存碎片。

12.传入一个指针，它如何确定具体要清理多少空间呢?

我们在申请内存的时候，会多分配16字节的内存，里面保存了内存块的详细信息，free会对传入的内存地址向左偏移16字节，然后分析出当前内存块的大小，就知道要释放多大的内存空间了。

13.define和const的区别是什么?

编译阶段：define是在编译预处理阶段进行简单的文本替换，const是在编译阶段确定其值

安全性：define定义的宏常量没有数据类型，只是进行简单的替换，不会进行类型安全检查；const定义的常量是有类型的，是要进行类型判断的

内存占用：define定义的宏常量，在程序中使用多少次就会进行多少次替换，内存中有多个备份，占用的是代码段的内存；const定义常量占用静态存储区域的空间，程序运行过程中只有一份

调试：define定义的宏常量不能调试，因为在预编译阶段就已经进行替换了；const定义的常量是可以进行调试的。

14.程序运行的步骤是什么

预编译：将头文件编译，进行宏替换，输出.i文件

编译：将其转化为汇编语言文件，主要做词法分析，语义分析以及检查错误，检查无误后将代码翻译成汇编语言，生成.s文件

汇编：汇编器将汇编语言文件翻译成机器语言，生成.o文件

链接：将目标文件和库链接到一起，生成可执行文件.exe

15.锁的底层原理是什么?

锁的底层是通过CAS，atomic 机制实现。

CAS机制：全称为Compare And Swap（比较相同再交换）可以将比较和交换操作转换为原子操作，CAS操作依赖于三个值：内存中的值V，旧的预估值X，要修改的新值B，如果旧的预估值X等于内存中的值V，就将新的值B保存在内存之中。（就是每一个线程从主内存复制一个变量副本后，进行操作，然后对其进行修改，修改完后，再刷新回主内存前。再取一次主内存的值，看拿到的主内存的新值与当初保存的快照值，是否一样，如果不一样，说明有其他线程修改，本次修改放弃，重试。）

atomic机制：如16问。

16.原子操作是什么?

原子操作是指不会被线程调度机制打断的操作，这种操作一旦开始，就一直运行到结束，中间不会有任何切换到另一个线程。

原理是：在X86的平台下，CPU提供了在指令执行期间对总线加锁的手段，CPU中有一根引线#HLOCK pin连接到北桥，如果汇编语言的程序在程序中的一条指令前面加上了前缀“LOCK”，经过汇编之后的机器码就使CPU在执行这条指令的时候把#HLOCKpin的电平拉低持续到这条指令结束的时候放开，从而把总线锁住，这样别的CPU就暂时不能够通过总线访问内存了，保证了多处理器环境中的原子性。

17.class与struct的区别

默认继承权限不同：class默认继承的是private继承，struct默认是public继承。

Class还可用于定义模板参数，但是关键字struct不能同于定义模板参数，C++保留struct关键字，原因是保证与C语言的向下兼容性，为了保证百分百的与C语言中的struct向下兼容，，C++把最基本的对象单元规定为class而不是struct，就是为了避免各种兼容性的限制。

18.内存对齐是什么？为什么要进行内存对齐？内存对齐有什么好处?

内存对齐是处理器为了提高处理性能而对存取数据的起始地址所提出的一种要求。

有些CPU可以访问任意地址上的任意数据，而有些CPU只能在特定的地址访问数据，因此不同硬件平台具有差异性，这样的代码就不具有移植性，如果在编译时将进行对齐，这就具有平台的移植性。CPU每次寻址有时需要消耗时间的，并且CPU访问内存的时候并不是逐个字节访问，而是以字长为单位访问，所以数据结构应该尽可能地在自然边界上对齐，如果访问未对齐内存，处理器需要做多次内存访问，而对齐的内存访问可以减少访问次数，提升性能。

优：提高程序的运行效率，增强程序的可移植性。

19.进程之间的通信方式有哪些?

管道：管道分为匿名管道和命名管道，管道本质上是一个内核中的一个缓存，当进程创建管道后会返回两个文件描述符，一个写入端一个输出端。缺点：半双工通信，一个管道只能一个进程写，一个进程读。不适合进程间频繁的交换数据

消息队列：可以边发边收，但是每个消息体都有最大长度限制，队列所包含的消息体的总数量也有上限并且在通信过程中存在用户态和内核态之间的数据拷贝问题

共享内存：解决了消息队列存在的内核态和用户态之间的数据拷贝问题。

信号量：本质上是一个计数器，当使用共享内存的通信方式时，如果有多个进程同时往共享内存中写入数据，有可能先写的进程的内容被其他进程覆盖了，信号量就用于实现进程间的互斥和同步PV操作不限于信号量+-1，而且可以任意加减正整数

信号

套接字

20.线程之间的通信方式有哪些?

信号量

条件变量

互斥量

21.介绍一下socket中的多路复用，及其他们的优缺点，epoll的水平和边缘触发模式

select、poll、epoll都是IO多路复用的一种机制，可以监视多个文件描述符，一旦某个文件描述符进入读或写就绪状态，就能够通知系统进行相应的读写操作。

Select优点：可移植性好，因为在某些Unix系统中并不支持poll和epoll

对于超时时间提供了更好的精度：微妙，而poll和epoll都是毫秒级

Select缺点：支持监听的文件描述符fd的数量有限制，最大数量默认是1024个

Select需要维护一个用来存放文件描述符的数据结构，每次调用select都需要把fd集合从用户区拷贝到内核区，而select系统调用后有需要把fd集合从内核区拷贝到用户区，这个系统开销在fd数量很多的时候会很大。

Poll优点（相对于select而言）：没有最大文件描述符数量的限制，poll基于链表存储主要解决了这个最大文件描述符数量的限制（当然，他还是有限制的，上限为操作系统能支持的能开启的最大文件描述符数量），优化了编程接口，减少了函数调用参数，并且，每次调用select函数时，都必须重置该函数的三个fd_set类型的参数值，而poll不需要重置。

Poll缺点：poll和select一样同样都需要维护一个用来存放文件描述符的数据结构，当注册的文件描述符无限多时，会使得用户态和内核区之间传递该数据结构的复制开销很大。每次poll系统调用时，需要把文件描述符fd从用户态拷贝到内核区，然后poll系统调用返回前，又需要把文件描述符fd集合从内核区拷贝到用户区，这个内存拷贝的系统开销在fd数量很多的时候会很大。

Epoll优点：和poll一样没有最大文件描述符数量的限制，epoll虽然也需要维护用来存放文件描述符的数据结构（epoll_event），但是它只需要将该数据结构拷贝一次，不需要重复拷贝，并且它只在调用epoll_ctl系统调用时拷贝一次要监听的文件描述符数据结构到内核区，在调用epoll_wait的时候不需要再把所有的要监听的文件描述符重复拷贝进内核区，这就解决了select和poll种内存复制开销的问题。

Epoll缺点：目前只有Linux操作系统支持epoll，不支持跨平台使用，而Unix操作系统上是使用kqueue

Epoll水平触发（LT）：对于读操作，只要缓冲区内容不为空，LT模式返回读就绪。

对于写操作，只要缓冲区还不满，LT模式会返回写就绪。

Epoll边缘触发（ET）：对于读操作，当缓冲区由不可读变为可读的时候，有新数据到达时，进程修改了EPOLL_CTL_MOD修改EPOLLIN事件时

在ET模式下，缓冲区从不可读变成可读，会唤醒应用进程，缓冲区数据变少的情况，则不会再唤醒应用进程。

当被监控的文件描述符上有可读写事件发生时，epoll_wait()会通知处理程序去读写。如果这次没有把数据全部读写完(如读写缓冲区太小)，那么下次调用epoll_wait()时，它不会通知你，也就是它只会通知你一次，直到该文件描述符上出现第二次可读写事件才会通知你。通常配合将文件描述符设置为非阻塞状态一起使用，这种模式比水平触发效率高，系统不会充斥大量你不关心的就绪文件描述符。

24.类的生命周期

类从被加载到内存中开始，到卸载出内存为止，它的整个生命周期包括：加载、验证、准备、解析、初始化、使用和卸载七个阶段。其中验证，准备，解析三个部分统称为连接

全局对象在main开始前被创建，main退出后被销毁。

静态对象在第一次进行作用域时被创建，在main退出后被销毁。

局部对象在进入作用域时被创建，在退出作用域时被销毁。

New创建的对象直到内存被释放的时候都存在。

25.父类的构造函数和析构函数是否能为虚函数？这样操作导致的结果？

构造函数不能为虚函数，虚函数的调用是通过虚函数表来查找的，而虚函数表由类的实例化对象的vptr指针指向，该指针存放在对象的内部空间之中，需要调用构造函数完成初始化，如果构造函数为虚函数，那么调用构造函数就需要去寻找vptr，但此时vptr还没有完成初始化，导致无法构造对象。

析构函数可以且经常为虚函数：当我们使用父类指针指向子类时，只会调用父类的析构函数，子类的析构函数不会被调用，容易造成内存泄漏。

26.多线程为什么会发生死锁，死锁是什么？死锁产生的条件，如何解决死锁？

因为在多进程中易发生多进程对资源进行竞争，如果一个进程集合里面的每一个进程都在等待这个集合中的其他一个进程才能继续往下执行，若无外力他们将无法推进，这种情况就是死锁。产生死锁的四个条件：互斥条件、请求和保持条件、不可剥夺条件、环路等待条件。解决死锁的方法就是破坏上述任意一种条件。

27.描述一下面向过程和面向对象

面向对象：就是将问题分解为各个对象，建立对象的目的不是为了完成一个步骤，而是为了描述某个事物在整个解决问题的步骤中的行为，相比面向过程，代码更易维护和复用。但是代码效率相对较低。

面向过程：就是将问题分析出解决问题的步骤，然后将这些步骤一步一步的实现，使用的时候一个一个调用就好。代码效率更高但是代码复用率低，不易维护。

28.C++中左值和右值是什么？++i是左值还是右值，++i和i++哪个效率更高？

第一小问结合本文第七和第八问，++i是左值，因为++i返回的是一个左值没有发生拷贝，所以效率更高。

29.介绍一下vector、list的底层实现原理和优缺点

Vector优点：可使用下标随机访问，尾插尾删效率高。

缺点：前面部分的插入删除效率低，扩容有消耗，可能存在一定的空间浪费。

底层是由一块连续的内存空间组成，由三个指针实现的分别是头指针（表示目前使用空间的头），尾指针（表示目前使用空间的尾）和可用空间尾指针实现

List优点：按需申请内存，不需要扩容，不会造成内存空间浪费。在任意位置的插入删除下效率高。

缺点：不支持下标随机访问

底层是由双向链表实现的

30.静态变量在哪里初始化？在哪一个阶段初始化？（都存放在全局区域）

静态变量，全局变量，常量都在编译阶段完成初始化和内存分配。其他变量都是在编译阶段进行初始化，运行阶段内存分配.。

31.如何实现多进程?

在Linux中C++使用fork函数来创建进程

而windows中C++使用createprocess来创建进程

32.空对象指针为什么能调用函数?

在类的初始化的时候，编译器会将它的函数分配到类的外部，这也包括静态成员函数，这样做主要是为了节省内存，如果我们在调用类中的的成员函数时没有使用类中的任何成员变量，它不会使用到this指针所以可以正常调用这个函数。

33.shared_ptr线程安全吗?

智能指针中的引用计数是线程安全的，但是智能指针所指向的对象的线程安全问题，智能指针没有做任何保障线程不安全。也就是说它所管理的资源可以线程安全的释放，只保证线程安全的管理资源的生命期，不保证其资源可以线程安全地被访问。

34.push_back()左值和右值的区别是什么?

如果push_back（）的参数是左值，则使用它拷贝构造新对象，如果是右值，则使用它移动构造新对象.。

35.move底层是怎么实现的?

Move的功能是将一个左值引用强制转化为右值引用，继而可以通过右值引用使用该值，以用于移动语义，从实现原理上讲基本等同一个强制类型转换。

优点：可以将左值变成右值而避免拷贝构造，将对象的状态所有权从一个对象转移到另一个对象，只是转移，没有内存搬迁或者内存拷贝。

36.完美转发的原理是什么?

完美转发是指函数模板可以将自己的参数完美的转发给内部调用的其他函数，完美是指不仅能够准确的转发参数的值，还能保证被转发参数的左、右值属性不变，使用引用折叠的规则，将传递进来的左值以左值传递出来，将传递进来的右值以右值的方式传出。

37.空类中有什么函数?

默认构造函数、默认拷贝构造函数、默认析构函数、默认赋值运算符

取值运算符、const取值运算符

38.explicit用在哪里？有什么作用?

只能用于修饰只有一个参数的类构造函数（有一个例外就是，当除了第一个参数以外的其他参数都有默认值的时候此关键字依然有效），它的作用是表明该构造函数是显示的，而非隐式的，跟它对应的另一个关键字是implicit，意思是隐藏的，类构造函数默认情况下声明为implicit。作用是防止类构造函数的隐式自动转换。

39.成员变量初始化的顺序是什么?

成员变量在使用初始化列表初始化时，与构造函数中初始化成员列表的顺序无关，只与定义成员变量的顺序有关。如果不使用初始化列表初始化，在构造函数内初始化时，此时与成员变量在构造函数中的位置有关。类中const成员常量必须在构造函数初始化列表中初始化。类中static成员变量，只能在类外初始化。

顺序：基类的静态变量或全局变量，派生类的静态变量或者全局变量，基类的成员变量，派生类的成员变量。

40.指针占用的大小是多少?

64位电脑上占8字节，32位的占4字节，我们平时所说的计算机多少位是指计算机CPU中通用寄存器一次性处理、传输、暂时保存的信息的最大长度。即CPU在单位时间内能一次处理的二进制的位数，因此CPU所能访问的内存所有地址由多少位组成，而8比特位表示1字节，就可以得出在不同位数的机器中指针的大小。

41.野指针和内存泄漏是什么？如何避免?

内存泄漏：是指程序中以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，导致程序运行速度减慢甚至系统崩溃等严重后果

避免：使用智能指针管理资源，在释放对象数组时使用delete[]，尽量避免在堆上分配内存

野指针：指向一个已删除的对象或未申请访问受限内存区域的指针。

避免：对指针进行初始化，用已合法的可访问内存地址对指针初始化，指针用完释放内存，将指针赋值nullptr。

42.malloc和new的区别是什么?

Malloc/free是标准库函数，new/delete是C++运算符

Malloc分配内存失败返回空，new失败抛异常

New/delete会调用构造析构函数，malloc/free不会，所以他们无法满足动态对象的要求。

New返回有类型的指针，malloc返回无类型的指针

分配内存的位置：malloc从堆上动态分配内存，new是从自由存储区为对象动态分配内存（取决于operator new的实现，可以为堆还可以是静态存储区）

New申请内存的步骤：调用operator new函数，分配一块足够大，且原始的，未命名的内存空间来存储特定类型的对象。运行相应的构造函数来构造对象，并为其传入初值，返回一个指向该对象的指针。

Delete：先调用对象的析构函数，再调用operator delete函数释放内存空间

43.多线程会发生什么问题？线程同步有哪些手段?

会引发资源竞争的问题，频繁上锁会导致程序运行效率低下，甚至会导致发生死锁。

线程同步手段：使用atomic原子变量，使用互斥量也就是上锁，使用条件变量或信号量制约对共享资源的并发访问。

44.什么是STL?

它是C++标准库的重要组成部分，不仅是一个可复用的组件库也是一个包含了数据结构与算法的软件架构，它拥有六大组件分别是：仿函数，算法，迭代器，空间配置器，容器，配接器

45.对比迭代器和指针的区别

迭代器不是指针，是一个模板类，通过重载了指针的一些操作符模拟了指针的一些功能，迭代器返回的是对象引用而不是对象的值。

指针能够指向函数而迭代器不行迭代器只能指向容器

46.线程有哪些状态，线程锁有哪些?

五种状态：创建，就绪，运行，阻塞，死亡

线程锁的种类：互斥锁，条件锁，自旋锁，读写锁，递归锁

47.解释说明一下map和unordered_map

Map内部实现是一个红黑树，内部所有的元素都是有序的，而hashmap则是内部实现了一个哈希表，内部存储元素是无序的

Map优点：有序性，其次是内部实现的是一个红黑树，使得很多操作都可以在logn的复杂度下可以实现效率较高。

Map缺点：空间占用率高

Unorderedmap优点：查找效率非常高。缺点：哈希表的建立比较费时间

48.vector中的push_back()和emplace_back()的区别、以及使用场景

当使用Push_back时会先调用类的有参构造函数创建一个临时变量，再将这个元素拷贝或者移动到容器之中，而emplace_back则是直接在容器尾部进行构造比push_back少进行一次构造函数调用。在大部分场景中emplace_back可以替换push_back，但是push_back会比emplace_back更加安全，emplace_back只能用于直接在容器中构造新元素的情况，如果要将现有的对象添加到容器中则需要使用push_back

49.如何实现线程安全，除了加锁还有没有其他的方式?

除了锁之外还可以使用互斥量（防止多个线程来同时访问共享资源，从而避免数据竞争的问题），原子操作（原子操作是不可分割的，使用原子操作可以确保在多线程环境中操作是安全的），条件变量（协调线程之间的协作，用来在线程之间传递信号，从而控

制线程的执行流程）等方式

50.vector扩容，resize和reserve的区别

Reserve只能修改能够存储的元素总数，不能修改当前存储的元素的个数

Resize是只修改能够存储元素总数，不能修改当前存储元素的个数

51.vector扩容为了避免重复扩容做了哪些机制?

当vector内存不够时本身内存会以1.5或者2倍的增长，以减少扩容次数

引入了reserve，自定义vector最大容量

52.C++中空类的大小是多少?

1字节

53.weak_ptr是怎么实现的?

实现依赖于计数器和寄存器实现的，计数器用来记录弱引用的数量，寄存器用来存储shared_ptr

54.虚函数的底层原理是什么?

虚函数表和虚表指针，详细看本文第四问。

55.一个函数f(int a,int b)，其中a和b的地址关系是什么?

a和b的地址是相邻的。

56.移动构造和拷贝构造的区别是什么?

移动构造函数本质上是基于指针的拷贝，实现对堆区内存所有权的移交，在一些特定场景下，可以减少不必要的拷贝。比如用一个临时对象或者右值对象初始化类实例时。我们可以使用move()函数，将一个左值对象转变为右值对象。而拷贝构造则是将传入的对象复制一份然后放进新的内存中

57.lamda表达式捕获列表捕获的方式有哪些？如果是引用捕获要注意什么?

按值捕获和引用捕获，默认的引用捕获可能会导致悬挂引用，引用捕获会导致闭包包含一个局部变量的引用或者形参的引用，如果一个由lambda创建的闭包的生命周期超过了局部变量或者形参的生命期，那么闭包的引用将会空悬。解决方法是对个别参数使用值捕获

58.哈希碰撞的处理方法

开放定址法：当遇到哈希冲突时，去寻找一个新的空闲的哈希地址。

再哈希法：同时构造多个哈希函数，等发生哈希冲突时就使用其他哈希函数知道不发生冲突为止，虽然不易发生聚集，但是增加了计算时间

链地址法：将所有的哈希地址相同的记录都链接在同一链表中

建立公共溢出区：将哈希表分为基本表和溢出表，将发生冲突的都存放在溢出表中

59.unordered_map的扩容过程

当unordered_map中的元素数量达到桶的负载因子（0.75）时，会重新分配桶的数量（通常会按照原有桶的数量*2的方式进行扩容，但是具体的增长策略也可以通过修改容器中的max_load_factor成员变量来进行调整），并将所有的元素重新哈希到新的桶中。

60.vector如何判断应该扩容？（size和capacity）

由当前容器内元素数量的大小和容器最大大小进行比较如果二者相等就会进行扩容，一般是1.5倍，部分的有两倍

61.构造函数是否能声明为虚函数？为什么？什么情况下为错误?

构造函数不能为虚函数，虚函数的调用是通过虚函数表来查找的，而虚函数表由类的实例化对象的vptr指针指向，该指针存放在对象的内部空间之中，需要调用构造函数完成初始化，如果构造函数为虚函数，那么调用构造函数就需要去寻找vptr，但此时vptr还没有完成初始化，导致无法构造对象。

62.类中static函数是否能声明为虚函数?

不能，因为类中的static函数是所有类实例化对象所共有的，没有this指针，而虚函数依靠vptr和vtable来处理，vptr是一个指针，在类中的构造函数中生成，并且只能通过this指针访问，对于静态成员函数来说，他没有this指针，无法访问vptr，因此static函数无法

声明为虚函数

63.哪些函数不能被声明为虚函数?

构造函数，内联函数（内联函数有实体，在编译时展开，没有this指针），静态成员函数，友元函数（C++不支持友元函数的继承），非类成员函数

64.如何保证类的对象只能被开辟在堆上？（将构造函数声明为私有、单例）

将构造函数设置为私有，这样只能使用new运算符来建立对象，但是我们必须准备一个destory函数来进行内存的释放，然后将析构函数设置为protected，提供一个public的static函数来完成构造，类似于单例模式

如果在栈上分配呢？则是重载new操作符，使得new操作符的功能为空，这样就使得外层程序无法在堆上分配对象，只可以在栈上分配

65.讲讲你理解的虚基类

虚基类是 C++ 中一种特殊的类，用于解决多继承所带来的“菱形继承”问题。如果一个派生类同时从两个基类派生，而这两个基类又共同继承自同一个虚基类，就会形成一个“菱形”继承结构，导致派生类中存在两份共同继承的虚基类的实例，从而引发一系列的问题。

为了解决这个问题，我们可以将虚基类作为共同基类，并在派生类中采用虚继承的方式。

虚继承会使得派生类中只存在一份共同继承的虚基类的实例，从而避免了多个实例之间的冲突。

虚基类是可以被实例化的。

66.C++哪些运算符不能被重载?

成员访问操作符，域解析操作符，条件运算符之类的不能重载。其中并不推荐对逗号运算符，逻辑或逻辑与之类运算符进行重载，容易造成歧义。

67.动态链接和静态链接的区别，动态链接的原理是什么?

区别：他们的最大区别就是在于链接的时机不同，静态链接是在形成可执行程序前，而动态链接的进行则是程序执行时。

静态库：就是将库中的代码包含到自己的程序之中，每个程序链接静态库后，都会包含一份独立的代码，当程序运行起来时，所有这些重复的代码都需要占用独立的存储空间，显然很浪费计算机资源。

动态库：不会将代码直接复制到自己程序中，只会留下调用接口，程序运行时再去将动态库加载到内存中，所有程序只会共享这一份动态库，因此动态库也被称为共享库。

动态链接原理：是把程序按照模块拆分成各个相对独立部分，在程序运行时才将它们链接在一起形成一个完整的程序，而不是像静态链接一样把所有程序模块都链接成一个单独的可执行文件

68.C++中怎么编译C语言代码?

使用extern“C”让C++代码按照C语言的方式去编译

69.未初始化的全局变量和初始化的全局变量放在哪里?

初始化的全局变量存放在数据段，数据段数据静态分配。

未初始化的全局变量存放在BSS（Block Started By Symbol）段,属于静态内存分配

70.说一下内联函数及其优缺点

内联函数是在编译期将函数体内嵌到程序之中，以此来节省函数调用的开销。

优点：是节省了函数调用的开销，让程序运行更加快速。

缺点：是如果函数体过长，频繁使用内联函数会导致代码编译膨胀问题。不能递归执行

71.C++11中的auto是怎么实现自动识别类型的?模板是怎样实现转化成不同类型的?

auto仅仅只是一个占位符，在编译期间它会被真正的类型替代，或者说C++中变量必须要有明确类型的，只是这个类型是由编译器自己推导出来的。函数模板是一个蓝图，它本身并不是函数，是编译器用使用方式具体类型函数的模具，所以模板其实就是将原本应该我们做重复的事情交给了编译器。

72.map和set的区别和底层实现是什么?map取值的 find，[]，at方法的区别(at有越界检查功能)

都是红黑树，find查找需要判断返回的结果才知道有没有查询成功。[]不管有没有就是0，如果原先不存在该key，则插入，如果存在则覆盖插入，at方法则会进行越界检查，这会损失性能，如果存在则返回它的值，如果不存在则抛出异常。

73.详细说一说fcntl的作用

作用：用于控制打开的文件描述符的一些属性和行为。

有五个功能：

1.复制一个现有的描述符(cmd=F_DUPFD)

2.获得/设置文件描述符标记（cmd=F_GETFD或F_SETFD）

3.获取/设置文件状态标记（cmd=F_GETFL或F_SETFL）

4.获取设置异步IO所有权（cmd=F_GETOWN或F_SETFL）

5.获取设置记录锁（cmd=F_GETLK或F_SET）

74.C++的面向对象主要体现在那些方面?

体现在C++引入了面向对象的一些特征，例如加入了封装继承多态的特点。（然后介绍一下封装继承多态）

75.介绍一下extern C关键字，为什么会有这个关键字?

是用来实现在C++代码段中用C语言的方式来编译代码，是C++为了兼容C语言所加入的关键字

76.讲一讲迭代器失效及其解决方法

序列式容器迭代器失效：当当前元素的迭代器被删除后，后面所有元素的迭代器都会失效，他们都是一块连续存储的空间，所以当使用erase函数操作时，其后的每一个元素都会向前移动一个位置，此时可以使用erase函数操作可以返回下一个有效的迭代器。

Vector迭代器失效问题总结：1.当执行了erase方法时，指向删除节点的迭代器全部失效，指向删除节点之后的全部迭代器也失效。

2.当进行push_back方法时，end操作返回的迭代器肯定失效。

3.当插入一个元素后，capacity返回值与没有插入元素之前相比有改变，则需要重新加载整个容器，此时first和end操作返回的迭代器失效。

4.当插入一个元素后，如果空间未重新分配，指向插入位置之前的元素的迭代器依然有效，但指向插入元素之后元素的迭代器全部失效。

Deque迭代器失效总结：1.对于deque，插入到除首尾位置之外的任何位置都会导致迭代器、指针和引用都会失效，如果在首尾位置添加元素，迭代器会失效，但是指针和引用不会失效。

2.如果在首尾之外的任何位置删除元素，那么指向被删除元素外其他元素的迭代器都会失效。3.如果在其首部和尾部删除元素则只会使指向被删除元素的迭代器失效。

关联型容器迭代器失效：删除当前的迭代器，仅仅会使当前的迭代器失效，只要erase时，递增当前迭代器即可。

77.编译器是如何实现重载的?

在编译时，编译器如果遇到了函数，就会在符号表里面命名一个符号来存放函数的地址，如果函数的使用在定义之前编译，无法在符号表中找到对应函数地址，则先标记为“？”（暂时未知），在全部编译结束后的链接过程将“？”在符号表里找到并替代为相应的函数地址，如果函数的定义在使用之前编译，则可以直接在符号表里找到对应函数地址直接使用，而在C语言中的符号表是以函数名为符号来存储函数地址，函数名相同的重载函数的地址应该不同，于是符号表中存在两个同符号的函数地址，在查找使用时会存在歧义和冲突。而C++符号表中的符号不是以函数名命名的，称为函数名修饰规则，虽然函数名相同，但是函数参数等其他属性不同，取的符号也不同，所以不会产生查询歧义的问题，使得函数可以重载。

78.什么是函数调用约定?

函数调用约定就是对函数调用的一个约束和规定，描述了函数参数是怎么传递和由谁清除堆栈的。它决定了，函数参数传递的方式（是否采用寄存器传递参数，采用哪个寄存器传递参数，参数压栈的顺序等），函数调用结束后栈指针由谁恢复（被调用的函数恢复还是调用者恢复），函数修饰名的产生方法。

__stdcall：是standardcall的缩写，是C++的标准调用方式，规则如下：所有参数从右到左依次入栈，如果是调用类成员的话，最后一个入栈的是this指针。被调用函数自动清理堆栈，返回值在EAX。函数修饰名约定：VC将函数编译后会在函数名前面加上下划线前缀，在函数名后加上“@”和参数的字节数。

__cdecl：是C DECLaration的缩写（declaration，声明），表示C语言的默认函数调用方法，规定如下：所有参数从右往左依次入栈，所有参数由调用者清除，称为手动清栈。返回值在EAX中。函数修饰名约定：VC将函数编译后会在函数名前面加上下划线前缀，由于由调用者清理栈，所以允许可变参数函数存在。

__fastcall：是快速调用约定，通过寄存器来传送参数，规则如下：用ECX和EDX传送前两个双字（DWORD）或更小的参数，剩下的参数仍然自右向左压栈传送。被调用函数在返回前清理传送参数的内存栈，返回值在EAX中。函数修饰名约定：VC将函数编译后会在函数名前面加上“@”前缀，在函数名后加上“@”和参数的字节数。

__thiscall：是唯一一个不能明确指明的函数修饰符，thiscall只能用于处理C++类成员函数的调用，同时thiscall也是C++成员函数缺省的调用约定，由于成员函数调用还有一个this指针，因此必须特殊处理，规定如下：采用栈传递参数，参数从右向左入栈，如果参数个数确定，this指针通过TCX传递给被调用者，如果参数个数不确定，this指针在所有参数压栈后被压入堆栈。对参数个数不确定的，调用者清理堆栈，否则由被调函数清理堆栈，__thiscal不是关键字，程序员不能使用l

__pascal：与__stdcall一样，在VC中已经被废弃

79.使用条件变量的时候需要注意什么?

当signal先于wait时，该信号会丢失，不会被后续的wait捕获

条件变量wait时，条件的判断和wait操作需要锁来保证原子性，要保证这一点，需要生产者在生产资源、cond signal时加和cond wait相同的锁，这样就会保证cond wait和cond signal先后顺序不会有问题，无论是谁先执行，都不会存在任何问题。

80.类内普通成员函数可以调用类内静态变量吗，类内静态成员函数可以访问类内普通变量吗?

类内普通成员函数可以调用类内静态变量，因为类内静态变量在编译时就已经完成了初始化和内存分配，类内普通函数调用类内静态变量说明类已经完成实例化，所以可以调用。静态函数可以直接访问静态变量，静态函数不能直接访问非静态变量，但是可以通过将类实例化对象后，静态函数去访问对象的非静态成员变量。

81.强制类型转换有哪几种类型，分别有什么特点？原理是什么?

Static_cast：用于数据类型的强制转换，强制将一种数据类型转化为另一种数据类型。

主要用法：

1.用于类层次结构中基类和派生类之间指针或引用的转换，进行上行切换（把派生类的指针或引用转换成基类表示）是安全的，进行下行转换（把基类的指针或引用转换为派生类表示），由于没有动态类型检查，所以是不安全的。

2.用于基本类型之间的转换，如把int转换成char，这种类型的转换也需要开发人员来保证

3.把空指针转换成目标类型的空指针。

4.把任意类型的表达式转换成void类型

5.涉及到类时，只能在有相互联系的类型中进行相互转换，不一定包含虚函数

注意：不能转换掉表达式中的const，volitale，__unaligned属性

Const_cast：用于强制去除类似于const这种不能被修改的常数特性。

用法：

1.用来修改类型的const或者volatile属性，除了const或volatile修饰之外，type_id和expression的类型是一样的。

2.常量指针被转化为非常量指针，并且仍然指向原来的对象

3.常量引用被转换为非常量引用，并且仍指向原来的对象，常量对象被转换成非常量对象。

注意：const_cast不适用于去除变量的常量性，而是去除指向常数对象的指针或引用的常量性，即去除常量性的对象必须为指针或者引用。

Reinterpret_cast：用于改变指针或引用的类型，将指针或引用类型转换成一个足够长的整形，将整形转换为指针或引用。

用法：

1.传入类型必须是一个指针，引用，算术类型，函数指针，成员函数或成员指针

2.它可以把一个指针转换成一个整数，也可以把一个整数转换成一个指针。

注意：在强制转换的过程中只是比特位的拷贝，咋使用中必须特别谨慎。

Dynamic_cast：其他三种都是在编译时完成的，它是在运行时处理的，运行时要进行类型检查。

用法：

1.不能用于内置的基本数据类型的强制转换。

2.如果转换成功会返回一个指向类的指针或者引用，转换失败会返回NULL。

3.进行转换的时候基类中一定要有虚函数，否则编译不通过（因为类中存在虚函数就说明它有想让基类指针或引用指向派生类对象的情况，此时转换才有意义）。

4.在类的转换时，在类层次间进行上行转换时，与static_cast的转换效果是一样的，在下行转换时，它具有类型检查功能，比static_cast更安全。

注意：向下转换的成功与否还与将要转换的类型有关，即要转换的指针指向的对象的实际类型与转换以后的对象类型一定要相同，否则转换失败。如果转换目标是指针类型转换失败，则结果返回0，如果是引用类型则抛出std::bad_cast异常

原理：改变了其内存二进制的存储形式。

82.回调函数是什么，为什么要有回调函数？有什么优缺点？回调的本质是什么？

回调函数是指使用者自己定义一个函数，实现这个函数的程序内容，然后别人把这个函数（入口地址）作为参数传入别人的函数中，由别人的函数在运行时来调用的函数，简单说就是放发生某种事件时，系统或其他函数将会自动调用你定义的一段函数。

可以把调用者和被调用者分开。调用者不关心谁是被调用者，所以它只需要知道的，只是一个存在某种特定类型原型，某些限制条件的被调用函数。

优点：

可以让实现方根据回调方的多种形态进行不同的处理和操作

可以让实现方，根据自己的需要定制回调方的不同形态

可以将耗时的操作隐藏在回调方，不影响实现方其他信息的展示。

让代码的逻辑更加集中，更加易读。

缺点：

回调函数过多会导致代码难以维护

回调函数容易造成资源竞争：如果回调函数中有共享资源访问，容易出现资源争抢，导致程序出错

代码可读性差，可能会破坏代码的结构和可读性

本质：是将函数当作参数使用，目的是为了使程序更加普适。

83.Linux中的信号有哪些?

SIGINT：终端终端符，默认动作：终止。当用户按中断键（Ctrl+C）时，终端驱动程序产生此信号并发送至前台进程组中的每一个进程，当一个进程在运行时失控，特别是在终端输出大量信息时，常用此信号终止它。

SIGQUIT：终端退出符，默认动作：终止+core。当用户在终端按退出键（Ctrl+\）时，终端驱动程序产生此信号，并发送给前台进程中所有进程，此信号不仅终止前台进程组，同时产生一个core文件。

SIGILL：非法硬件指令，默认动作：终止+core。此信号表示进程已执行一条非法硬件指令

SIGRAP：硬件故障，默认动作：终止+core。指示一个实现定义的硬件故障

SIGBUG：硬件故障，默认动作：终止+core。指示一个实现定义的硬件故障，当出现某些类型的内存故障时，常产生此信号。

SIGKILL：终止，默认动作：终止。这是两个不能被捕捉或忽略的信号之一，它向系统管理员提供一个可以杀死任一进程的可靠方法

SIGSEGV：无效的内存引用，默认动作：终止+core。指示进程进行了一次无效的内存引用，通常说明程序有错，比如 访问了一个未经初始化的指针。

SIGALRM：定时器超时，默认动作：终止。如果在管道的读进程终止时写管道，则产生此信号，当类型为SOCK_STREAM的套接字已不再连接时，进程写该套接字也产生此信号。

SIGTERM：终止，默认动作：终止。这是由kill命令发出的系统默认终止信号，由于该信号是由应用程序捕获的，所以使用SIGTERM也让程序有机会在退出之前做好清理工作，与SIGKILL不同的是，SIGKILL不能捕捉。

SIGCONT：使暂停进程继续，默认动作：忽略。此进程发送给需要运行但是目前状态是暂停的进程，如果接收到此信号的进程处于暂停状态则继续运行，否则忽略。

SIGURG：紧急情况，默认动作：忽略。通知进程发生一个紧急情况，在网络上街到带外的数据时，可以选择产生此信号

SIGPOLL：可轮询事件，默认动作：终止。产生条件当一个可轮询设备上发生一个特定事件时产生

SIGIO：异步IO，默认动作：终止。产生异步IO时产生

还有很多就不全部放进来了，全部链接：Linux信号一览_linux 信号表_lqc132的博客-CSDN博客

84.什么是尾递归?

尾递归时递归的一种特殊情形，尾递归时一种特殊的尾调用，即在尾部直接调用自身的递归函数。核心思想是边调用便产生结果。

原理：当编译器检测到一个函数调用是尾递归的时候，它会覆盖当前的活动记录而不是在栈中创建一个新的。编译器可以做到这一点，因为递归调用是当前活跃期内最后一条待执行的语句，于是当这个调用返回时栈帧中并没有其他事情可以做，因此也就没有保存栈帧的必要了，通过覆盖当前的栈帧而不是在其之上重新添加一个，这样所使用的栈空间就大大缩减了，这使得实际的运行效率会变得更高。

特点：在尾部调用的是函数自身，可通过优化使得计算仅占用常量栈空间

85.为什么会有栈溢出，为什么栈会设置容量?

栈空间是预设的，它通常用于存放临时变量，如果你在函数内部定义一个局部变量，空间超出了设置的栈空间大小，就会溢出。不仅如此，如果函数嵌套太多，也会发生栈溢出，因为函数没有结束前，函数占用的变量也不被释放，占用了栈空间。

原因：是栈的地址空间必须连续，如果任其任意成长，会给内存管理带来困难。对于多线程程序来说，每个线程都必须分配一个栈，因此没办法让默认值太大。

86.二叉树和平衡二叉树的区别

二叉树没有平衡因子的限制，而平衡二叉树有。

二叉树可能退化为链表，而平衡二叉树不会。

87.平衡二叉树的优缺点

优点：避免了二叉排序树可能出现最极端情况（退化为链表），其平均查找的时间复杂度为logN

缺点：对AVL树做一些结构修改的操作，性能非常低下，比如：插入时要维护其绝对平衡，旋转的次数比较多，更差的是在删除时，有可能一直要让旋转持续到根的位置。

88.什么是this指针，为什么存在this指针?

类和对象中的成员函数存储在公共的代码段，不同的对象调用成员函数时编译器为了知道具体操作的是哪一个对象给每个“非静态的成员函数”增加了一个隐藏的指针参数，让该指针指向当前对象，在函数体中所有成员变量的操作，都是通过这个指针来完成的由编译器自动完成。

89.什么是重载、重写、隐藏?

重载：函数名相同，函数参数不同，两个函数在同一作用域

重写：两个函数分别在子类和父类中，函数名，返回值，参数均相同，函数必须为虚函数

隐藏：在继承关系中，子类实现了一个和父类名字名字一样的函数。这样子类的函数就把父类的同名函数隐藏了。隐藏只与函数名有关。

90.静态成员函数可以是虚函数吗？为什么?

它不属于类中的任何一个对象或示例，属于类共有的一个函数，不依赖于对象调用，静态成员函数没有this指针，无法放进虚函数表。

91.构造函数可以为虚函数吗？为什么?

虚表指针是存储在对象的内存空间，当调虚函数时，是通过虚表指针指向的虚表里的函数地址进行调用的。如果将构造函数定义为虚函数，就要通过虚表指针指向的虚表的构造函数地址来调用。而构造函数是实例化对象，定义为虚函数后，对象空间还没有实例化，那就没有虚表指针，自然无法调用构造函数，那构造函数就失去意义，所以不能将构造函数定义为虚函数。

92.make_shared函数的优点，缺点?

优点：减少了内存分配的次数，降低了系统开销，提高了效率，使用new构造的话至少会进行两次内存分配，（一次为智能指针本身，一次为共享指针的控制块）

缺点：当构造函数是保护或者私有的时候无法使用make_shared函数。

会导致weak_ptr保持控制块，的生命周期，连带着保持了对象分配的内存，只有当最后一个weakptr离开作用域时，内存才会被释放，对于内存要求高的场景来说，是一个需要注意的问题。

93.函数调用进行的操作：

1.将参数压栈：按照参数顺序的逆序进行，如果参数中有对象则先进行拷贝构造

2.保存返回地址：即函数调用结束返回后接着执行的语句的地址

3.保护维护函数栈帧信息的寄存器内容如，SP（堆栈指针），FP（栈帧指针）等。

4.保存一些通用寄存器的内容：应为有些通用寄存器会被所有函数用到，所以在函数调用之前，这些寄存器就可能已经放置了对函数有用的信息。

5.调用函数，函数执行完毕

6.恢复通用寄存器的值

7.恢复保存函数栈帧信息的那些寄存器的值

8.通过移动栈指针，销毁函数的栈帧

9.将保存的返回地址出栈，并赋给寄存器。

10.通过移动栈指针，回收传给函数的参数所占用的空间

参考链接：

C++面经八股文_c++面试八股文_何处微尘的博客-CSDN博客

C++面试八股文快问快答の基础篇_c++八股文_谁吃薄荷糖的博客-CSDN博客

txinyu的博客_-CSDN博客

详解什么是尾递归（通俗易懂，示例讲解）_Allen Chou的博客-CSDN博客

数据结构--二叉树--详解_清欢有道的博客-CSDN博客

为什么要限制栈的大小？_栈大小限制_千么漾漾的博客-CSDN博客

C语言 全局变量和局部变量的大小限制 - Slyar Home

程序中关于堆栈大小的划定_任务堆栈大小怎么确定_liuhuiyi的博客-CSDN博客

回调函数详解_~青萍之末~的博客-CSDN博客

Linux信号一览_linux 信号表_lqc132的博客-CSDN博客

[C++] 一篇带你了解C++中隐藏的this指针_c++ this_小白又菜的博客-CSDN博客

条件变量详细解说_清风徐来Groot的博客-CSDN博客

迭代器失效的几种情况总结_迭代器失效 性能_BYR_jiandong的博客-CSDN博客

C++的四种强制转换_c++强制转换_酒馆店小二的博客-CSDN博客

【C++】vector的reserve()和resize()用法_vector resize_Amelie_xiao的博客-CSDN博客

cppreference.com

2021dragon_C++,leetcode,C语言-CSDN博客

参考书籍：

STL源码剖析，Unix网络编程，effective C++，C++PrimePlus