1、C++三大特性：

封装，继承，多态

封装：将属性和方法封装一起，隐藏内部实现，向外暴露接口

继承：子类继承基类的方法与属性，可以重写或扩展基类的功能

多态：允许同一接口呈现不同行为

2、C++中struct和class的区别：

struct 继承与访问都是默认public，存储一些结构简单的数据结构，与C语言兼容

Class继承和访问都是默认private，鼓励了更加严格的封装与抽象，与C语言不兼容

struct与union的区别：

struct每个成员都有自己的内存空间，大小为各成员之和，存储不同但是相关联的数据，可以同时访问多个数据

union所有成员共享一块内存空间，大小为最大成员大小，存储多个类型的数据，但一次只能使用一种，只能访问当前存储的成员

指针和引用的区别

指针存储另一个内存变量的地址，初始化的时候可以为空，存在多级指针，sizeof得到的是指针本身的大小，灵活但是容易出现野指针、内存泄漏

引用是对象的别名，在定义时需要初始化，不能为nullptr，sizeof引用得到的是对象本身的大小，不存在多级引用，一旦绑定某个变量就不可改变

空类可以实例化吗？

可以，示例会占用最小内存，大小为1字节，以确保每个对象有一个唯一的地址

当一个类中没有任何成员变量和成员函数，这是sizeof(A)的A值为多少？

其实就是空类，空类会占用最小内存，大小为1字节，编译器必须为每个对象分配一个独特的地址，以便在程序中区分不同的对象示例，因此会占用1字节的空间

子类不能继承父类的函数有哪些？

构造函数

析构函数

拷贝构造函数

赋值构造函数

私有成员函数

C++构造函数初始化列表执行顺序：

先调用基类的构造函数，如果有多个基类则按照它们在列表中的顺序构造

按照成员变量在类中声明的顺序进行初始化，而不是按照它们在初始化列表中的顺序初始化

再调用派生类构造函数的主体部分

构造函数的顺序，析构函数的顺序

构造函数：

先调用基类的构造函数，如果有多个基类则按照它们在列表中的顺序构造

按照成员变量在类中声明的顺序构造

最后调用构造函数的主体部分

析构函数：

先调用析构函数的主体部分

在根据成员变量在类中声明的顺序，反向销毁成员变量

最后调用基类的析构函数，如果有多个基类，也是按照它们在列表中的顺序反向调用

10、C++有哪些构造函数？

默认构造函数

参数化构造函数

委托构造函数：在构造函数中调用其它的构造函数

移动构造函数：转移构造所有权，以&&作为参数，销毁传进来的对象，调用的话临时对象作为右值，直接返回类对象，或者显式调用std::move

拷贝构造函数

显示构造函数

11、成员变量初始化顺序：

成员在类中声明的顺序

12、局部变量与全局变量的区别：

局部变量的声明周期就是所在代码块的运行周期，通常存储在栈上，作用范围为代码块内，初始化时为未定义值

全局变量存储在数据段，初始化默认为零值，生命周期从程序开始到结束，作用范围在整个程序范围内有效

13、常量指针与指针常量的区别

const int \* ptr 常量指针 指针指向的对象的值不可改变

int \* const ptr 指针常量 指针指向的对象不可更改

14、智能指针有什么缺点？

指针指针能极大的简化内存管理，但是也会带来性能开销、循环引用导致的内存泄露和代码复杂性增加的问题，要避免循环引用的情况。

15、shared\_ptr是不是线程安全的？

引用计数是线程安全的，被管理的对象和读写机制需要我们自己添加同步机制来保证线程安全。

指向同一对象（浅拷贝）共享同一实体，任何修改对所有指针可见

指向深拷贝后的资源，初始相同，修改后可能不同，初始内容复制自身资源，然后续修改仅影响当前指针指向的资源。

16、什么时候使用智能指针，什么时候使用裸指针？

智能指针：

适用于需要自动管理声明周期的场景

适用于需要防止内存泄露的场景

适用于需要管理复杂对象的所有权和生命周期的场景

适用于需要保证异常情况下资源能被正确释放的场景

裸指针：

适用于简单的、临时的、不涉及所有权的对象

对性能开销要求高的场景

使用与不需要管理资源声明周期的场景

17、share\_ptr和weak\_ptr的区别与联系？

共享指针共享资源的所有权，增加对象会增加引用计数，控制资源的生命周期，可能导致循环引用导致内存泄露

弱指针不拥有资源的所有权，不增加引用技术，不控制资源的声明周期，可以打破循环引用，防止内存泄露

18、引用作为函数参数在C++中有一下几个好处：

直接使用对象，避免了不必要的拷贝操作

可以通过引用直接修改函数参数的值

引用对象和直接使用对象在语法上保持一致

对于大禹像，使用引用的传递的效率更高

避免指针失效问题，不会控制到不可靠的数据

19、模板是在什么时候进行实例化？

模板在编译器碰到模板的具体使用时，进行实例化

开发者显示指定模板实例化

模板在编译时实例化，当编译器碰到模板的具体使用时，例如调用模板函数或创建模板类实例时，编译器会生成相应的代码

20、为什么模板类一般都是放在一个h文件中？

因为模板类的实例化发生在编译期，编译器需要在实例化模板时看到模板的完整定义与实现。

将模版类的定义与实现都放在头文件中，确保编译器在实例化模板时能够访问到完整的信息，避免链接错误。

如果将模板类的实现放在源文件中，可能导致链接错误，因为编译器在链接阶段无法找到模板示例化的具体实现

21、define宏定义和const、static的区别：

define 无类型检查，全局，编译时直接替换文本，无法设置断点，定义简单常量或者表达式

const 有类型检查，作用域内，编译时常量，无法设置断点，定义常量通常表示为只读

static 有类型检查，文件内或函数内的局部作用域内，静态存储，生命周期贯穿整个程序，无法设置断点，定义静态变量或者静态成员变量

22、new和malloc申请的是哪里的内存？如何减少内存碎片？

new和malloc都是申请的堆区的内存，但new会调用构造函数进行初始化，并在分配失败的时候抛出异常，malloc不调用构造函数，需手动初始化，并在分配失败的时候返回NULL。

减少内存碎片的策略：

使用内存池、按需分配大块内存、避免频繁的分配和释放操作、合并相邻的空闲块

23、实际情况中想要new分配内存但又不确定大小怎么做？

总结：

确定最大可能的大小，然后用new分配内存；

使用std：：vector或std：：unique\_ptr等动态容器，根据动态调整内存；

使用基类指针指向派生类对象，动态确定对象实际类型和大小。

24、堆和栈的区别：

25、ICMP：

处于TCP/IP的网络层，会封装在IP数据包内传输，可以让端点知道数据包哪里出了问题。

会报告网络错误、网络状态、然后辅助一些网络调试指令，比如我们最常的使用ping、traceroute

26、TCP粘包：

TCP粘包是指在网络传输中，多个原本独立的应用层数据包被封装合并成一个TCP数据包传输，导致接收端无法拆分出原始数据边界的现象。

比如：发送端依次发送数据100字节、200字节

而接收端依次收到了300字节，这就是粘包

27、TCP粘包的原因：

因为TCP是面向流的协议，不保留应用层数据边界，需要应用层自己处理。

为什么TCP是面向流的协议呢？是因为

发送端的Nagle算法与缓冲区合并。

Nagle算法为了减少网络中的微小数据包（如每次只发1字节），TCP会将小数据缓存到缓冲区，待积累到一定大小或超时后合并发送。

例如：发送端连续调用三次send发送100b、200b、300b，最后可能合并为600b发送。

发送缓冲区满或者主动flush，缓冲区数据会打包发送。

接收端的缓冲区积累：

接收端TCP将接收到的数据存入接收缓冲区，应用层通过recv读取时，若未读完缓冲区数据，剩余数据会与后续包合并。

例如：接收端recv一次读取200b，但缓冲区内有300b，那么剩余的100b会与下一个包的数据合并。

刚才说的都是小包，网络层的MTU限制：

可能将发送端的大包分为若干个小包发送，但接收端重组后仍可能导致应用层数据边界丢失。

28、粘包的解决办法：

固定长度分包

分隔符

长度加到包头里面

封装数据包，循环接收一个完整数据一个个的解析

29、UDP丢包与乱序：

UDP是无连接、不可靠的传输层协议，它的设计理念就是尽最大努力交付，不保证数据有序、不重传丢失包。

它本身不保证：

不会为了有序就增加序列号、增加应答。

丢包也不去重传。

丢包的其他原因：：：：

接收端的缓冲区溢出了：接收端应用层数据处理速度比接收数据的速度快，UDP接收端缓冲区就会丢弃新到达的包。例如：服务器繁忙。

网络拥塞与路由器丢包：

当网络流量超过链路的带宽时，路由器缓冲区满会主动丢弃数据包。

例如：我们直播时，客户端发送的视频流数据包会因为网络拥堵被路由器丢弃。

链路错误与物理层问题：

无线信号衰减、网线接触不良等物理层问题，可能触发ICMP协议啦报了哪个数据包路由有问题，或者导致数据包CRC错误，数据包被直接丢弃。

NAT超时与防火墙过滤：

NAT设备（路由器）对UDP连接有超时机制（通常数分钟），超时后会删除映射表，导致后续包被丢弃。

防火墙可能误判UDP包为异常流量而过滤，如无状态防火墙。

乱序的其它原因：：：：：：：

网络路径差异（网络层）（多路径传输）：

数据包可能通过不同的路径到达目的地，各路径的延迟和长度不同导致后发送的包先到达。

IP分片与重组顺序：

大UDP包被IP层分片后，不同的分片可能经不同路径传输，重组时顺序错乱。

若分片丢失，接收端可能无法重组完整的包，可能导致后续包被缓存等待，可能进一步加剧乱序。

操作系统内核调度延迟：

接收端内核处理数据包的顺序可能与到达顺序不同（如中断处理优先级差异）

30、UDP丢包解决方案：

FEC前向纠错：

原理：发送端按一定规则添加冗余数据（如异或校验），接收端通过冗余数据恢复丢失包。

示例：发送3个原始包，添加一个校验包，若丢失一个原始包，可通过其它包计算恢复。

ARQ自动重传请求：（应用层实现）

发送端为每个端添加序列号，接收端接收后返回ACK；超时未收到ACK则重传。

注意：ARQ会增加延迟，使用于非实时场景（如UDP文件传输）

流量控制与缓冲区优化：

接收端动态调整窗口大小（类似与TCP的滑动窗口），通过反馈机制告诉发送端发送速率。

示例：RTP协议中的RTCP反馈机制用于流媒体流量控制

NAT穿透与防火墙适配

使用STUN/TURN协议穿越NAT，或通过周期性发送心跳包维持NAT映射（防止超时）

31、乱序解决方案：

序列号与顺序重排：

发送端为每个UDP包添加序列号，接收端维护缓冲区，按序列号排序后再交付与应用层

示例：RTP协议通过序列号处理音视频包的乱序问题

时间戳与抖动缓冲区：

包中添加时间戳，接收端设置抖动缓冲区，按时间戳顺序缓冲包，等待一定时间后再处理。

权衡：缓冲区越大，抗乱序能力越强，但延迟越高。（适用于直播而非游戏）

32、TCP与UDP区别

TCP面向连接，可靠，保证顺序，不丢包，因为握手、确认、重传机制延迟高，适用于文件传输、HTTP、邮件；UDP无连接，不可靠，不保证交付，延迟低，适用于实时音视频、游戏、DNS查询

33、this指针何时传入？

this指针是C++编译器自动为类的非静态成员函数添加的隐含参数，指向当前调用函数的对象实例。它并非对象的成员变量，而是通过函数参数传递机制传入的。

存储方式：x64，寄存器传递或栈传递，这取决于编译器约定

this指针的传递时机：

对象调用函数：

例如:ClassName obj; obj.method(); // 调用非静态函数

编译器会将其转换为类似ClassName::method(&obj),将obj的地址作为this指针传入。

指针调用或引用调用，等价与对象调用。

构造函数与析构函数中的this：

构造函数执行时，this指针指向正在创建的对象内存地址。

析构函数执行时，this指针指向即将销毁的对象。

34、this指针的限制：

静态成员属于类而不是对象，因此无法通过this指针访问非静态成员。

不能显式传递this

35、静态函数可以为虚函数吗？

不能。静态函数属于类而不是对象。virtual关键字修饰的函数为虚函数，virtual关键字修饰静态函数会报错。而且静态函数可以直接被类名：：调用，就与多态冲突了。

36、静态函数可以被继承吗？

静态函数属于类，当然可以被继承。派生类可以声明与基类同名的静态函数，属于隐藏而非覆盖。

37、想要实现静态函数多态：

静态函数的参数，传入仿函数，仿函数可以让函数动态，只确定返回类型与参数类型，甚至参数类型也可以用模板显示

38、流量控制与拥塞控制

39、超时重传的时机

40、TCP可靠传输的几种机制

41、TCP三次握手过程

42、TCP三次握手的过程。tcp建立了连接，后续如果数据包有损坏和乱序，是如何处理的？TCP遇到网络拥塞或者大流量的情况如何去做？

43、https是如何进行加密的？

44、遇到多线程同步问题有哪些机制？举例来说信号量如何实现？它们一般会调用到什么样的函数？如果想等待线程a执行完任务再去执行b，这种情况要怎么处理？

45、多线程切换cpu和内存会做一些什么处理？

46、归并排序的事件复杂度，归并排序的思路

47、C++多态如何实现？

48、类对象内存模型：

49、说一下你对内存对齐的理解：

50、内存对齐可能带来哪些问题？

51、如果使用指针+偏移量来访问对象的话，会访问到什么数据？

52、线程：

53、线程的最大缺点是什么？

54、你能说一下有几种锁吗？

55、互斥锁线程的阻塞和唤醒的原理？

56、智能指针全部：

57、熟悉ftp吗？音频的流式传输是如何做的？

58、C++标准库set底层的实现？

59、讲一下红黑树的特性以及它和二插搜索树的区别？

60、缺页中断什么时候发生？

61、常见的缺页中断算法？

62、音频如何实现进度的跳转的，这个如何实现？

63、音频中如何把连续信号转成离散信号？

64、static

65、http的报文协议，它的header大概长什么样子？

66、为什么不能开太多线程？

67、什么时候会做线程的切换？

68、单例模式有哪些优点和缺点？

69、了解一些常用的设计模式吗？

70、迭代器模式讲一讲？

71、双向队列与单向队列在数据结构上有啥差别？

72、观察者模式起什么作用？

73、unordered\_map底层实现是什么数据结构？

74、用过哪些C++新特性？

75、除了unordered\_map还有一些其它的问题吗？

76、智能指针用的多吗？

77、智能指针有什么好处？

78、使用普通指针只有内存泄露的问题吗？还有其它什么问题？

79、shared\_ptr它大概是如何实现的？

80、移动构造的使用场景

81、创建线程来实现自己的一个消息队列

82、音频采集的输入、音效处理的输入是什么，数据类型是什么？

83、调用的音频库的采集的库，它的输入和输出分别是什么？

84、音频数据是怎么播放的？

85、实现的最复杂，或者改动比较多的功能是什么？

86、C++如何实现多线程？

87、C++多线程的理解与应用

88、介绍线程间同步的方式

89、C++在返回一个局部对象的时候是如何避免多次调用构造函数的？（RVO）

90、C++虚函数实现原理？如何实现多态？

91、stl常用什么容器？

92、多线程访问一个vector时怎么办？

93、如何转移一个线程，把操作丢给另外一个线程去做

94、如何去做C++库的裁剪？哪些C++符号是可以被裁剪的？如何做这个裁剪操作？

95、C++编译链接的整个过程？

96、详细说一下链接的详细情况，编译时与运行时的区别是什么？

97、使用句柄fd会遇到什么问题？如果一个句柄被其他人关掉，然后你再去用会报什么错？

98、OpenGL的坐标系是怎么的？Camera2和Camera1的区别

99、I帧与P帧的概念

100、git常用哪些操作？merge和rebase的区别

101、网络劫持是如何实现的？如何去防止劫持？

102、堆栈区别：

堆的内存是动态分配和管理，需要程序员或者垃圾回收机制管理

栈的内存是自动管理的，函数调用时自动分配，函数返回时自动释放

堆主要存储动态分配的对象和数据

栈主要存储局部变量和函数调用的执行环境

堆通常较大，可以动态扩展

栈的大小固定，由操作系统和编译环境预设

堆的访问速度要小于栈

堆的生命周期不固定，栈的生命周期通常随着函数的执行开始和结束

103、堆的访问为什么比栈快？

104、C++栈溢出：

函数递归调用过深

函数嵌套过深、或无限调用

分配过大的局部变量

105、如何防止栈溢出：

用堆分配大的数据

在递归函数中添加基准条件，限制递归深度

避免过深的嵌套，合理设计函数调用关系

106、C++内存泄露及检测工具：

107、介绍一下RALL思想，他是怎么保证对象没有内存泄露的？

资源在对象创建时获取

资源在对象析构时自动释放

通过对象的生命周期控制着资源的生命周期

在异常发生时，确保资源被正确释放

108、友元函数

类外的函数可以访问类内的私有和保护成员

109、友元类

可以访问另一个类的私有和保护成员

110、什么是虚函数？

被Virtual关键字修饰的函数是虚函数

（虚函数的调用不是在编译时解析的，而是在运行时才决定调用哪个函数

每个含有虚函数的类都有一个函数表，这个表是一个指针数组，每个元素对应了类中的一个虚函数。对象通过这个表来来查找正确的函数调用。

每个含有虚函数的对象都有一个虚指针，这个指针指向其类的虚函数表，这个指针是动态绑定的关键。

111、什么函数不能声明为虚函数？

构造函数：对象构造的时候还没虚指针

静态函数：无this指针，与对象无关，不能声明为虚函数

内联函数：可以声明为虚函数，但内联特性会让编译器忽略

112、构造函数和析构函数能不能是虚函数，说明原因：

构造函数不能为虚函数，因为构造函数在调用的时候，虚指针还没设置或未指向最终的类

析构函数可以是虚函数：

析构函数对象销毁时调用，基类指针指向子类对象，当通过基类指针删除子类对象时，会调用派生类的析构函数，保证子类和基类的资源都能被正确的释放。

113、虚函数表在哪个阶段被分配？

在编译阶段，编译器会确定那些类包含虚函数，为这些类创建虚函数表。虚函数表中包含指向类中所有虚函数的指针。

在链接阶段，虚函数表会被分配具体的内存地址，链接器会将虚函数表放入全局内存空间中，确保每个虚函数表都有唯一的内存地址。

运行时，类对象被实例化，虚表指针被初始化，指向对应的类的虚函数表

114、没有虚函数的话，C++如何实现多态？

函数指针，通过函数指针来实现运行多态

std：：function 通过仿函数进行对象的存储

模板实现编译时多态

115、同一个类的不同对象的虚函数表是一个吗？

是一个，类的虚函数表是固定的，在编译期及已经被创建

116、基类的虚函数表放在内存什么区？虚表指针的初始化时间？

全局共享且大小固定，放在全局内存区，linux放在只读数据段，windows放在常量段

虚表指针在类的构造函数执行时被初始化。虚表指针存在与对象内存布局的最前端，对象时局部变量就在栈，对象是动态分配的就在堆

117、内存分哪些区？

118、虚函数表是对象拥有，还是类拥有？

虚函数表是对象拥有

119、虚函数内部调用非虚函数是调用指针类还是对象类？

当一个虚函数在调用时，调用的是实际对象的虚函数实现，这是通过虚函数表来实现的。

调用函数就是当前对象所属类的非虚函数。

120、纯虚函数的实现原理。

纯虚函数用于定义接口，没有具体实现。含有纯虚函数的类称为抽象类，不能实例化。

具体实现就是virtual关键字修饰的函数=0

121、纯虚函数，使用场景有哪些？

纯虚函数主要用于定义接口，确保所有派生类实现特定多态的行为。

122、虚继承解决了什么问题？

虚继承用于解决菱形继承问题，在多继承结构中，不是用虚继承可能会导致基类成员的重复继承。

通过虚继承，基类成员在继承链中只保存一份，避免了冗余和潜在的二义性。

123、野指针和悬空指针

野指针：未初始化。使用野指针可能会读写到不属于程序的内存区域，导致数据损坏或者程序崩溃

悬空指针：释放内存后未将指针设为空，可能会导致未定义的内容，比如访问到已经释放了的内存。

124、内存对齐

以结构体内最大成员进行对齐，如果是结构体嵌套，就以结构体内最大成员进行对齐

优点：

提高了cpu的访问效率

兼容硬件架构

优化缓存效率

缺点：

浪费内存空间

增加了数据布局的复杂度

影响数据序列化和网络传输

125、extern关键字

extern “c” 告诉编译器按C语言的简单名称处理，能够C++兼容c语言的语法，防止链接不到程序。

126、C++编译流程

预处理 宏定义、头文件包含、条件编译指令

编译 根据语法将代码转换为汇编代码

汇编 将汇编代码转换为机器语言

链接 将多个目标文件与库文件连接生成可执行文件

127、编译器在底层如何实现重载？

编译器是通过名称改变 函数 a（int x） 边以为 a\_i

128、C++函数调用的压栈过程：

函数调用时从左向右将参数压入栈，随后压入返回地址，跳转到函数代码执行，处理返回值并清理栈，最后返回到调用位置

129、什么是回调函数？回调函数的使用场景？

回调函数是一种委托机制，通过把函数指针作为参数传递给另一个函数，然后在函数内调用这个函数指针指向的函数。这样的话，就可让其它的代码在适当位置，调用并且这个函数传递适当的参数了。

130、内联函数和宏定义的区别。

内联函数有类型检查，编译器在调用处展开。

宏定义是预处理在编译时文本替换，无类型检查，难以发现错误。

131、如何正确使用内联函数？

内联函数适用于短小、调用频繁、且需要类型检查和调试支持的场景。

太大的函数即使被inline关键字修饰，编译器可能也不会将它作为内联函数。

132、浅拷贝与深拷贝的区别：

浅拷贝只复制对象的指针，不复制指针指向的内村，多个对象共享一块内存资源，一个对象的修改会影响到共享内存的所有对象。

深拷贝复制对象及其指针指向的所有内存内容，每个对象拥有自己对立的内存副本，一个对象的修改不会影响其它对象。

133、如何消除隐式转换？

编译器自动讲义中类型转换为另一种类型称为隐式转换。

使用explicit关键字可以防止构造函数的隐式转换，确保只能显式的调用构造函数。

134、说一下零拷贝：

零拷贝就是在数据传输过程中避免了内存拷贝。直接在模块间（内核空间和用户空间）之间传递数据，减少了cpu和内存带宽的消耗。

常用于高性能网络传输和文件I/O操作。

sendfile：在linux中，将文件数据直接从文件描述符发送到网络套接字

mmap：内存映射文件，允许文件的内容直接映射到进程的地址空间

splice：将数据在两个文件描述符之间移动，而无需数据拷贝到内存空间

DMA：允许设备直接访问主内存，减少CPU参与数据传输

135、strlen和sizeof的区别：

strlen是在运行时获取字符串的长度

sizeof是直接在编译时获取对象的大小

136、头文件如何避免出现重定义的问题：

#ifndef #define #endif

137、静态多态与动态多态：

静态多态 是 编译时多态，如模板和重载

动态多态 是运行时多态，如虚函数和继承实现

138、重载和重写的区别

重载是在同一个类内，不同的参数列表的同名函数

重写是子类重写父类的虚函数，是同一函数

139、临时对象在什么时候产生？

返回值、返回对象

函数参数可能被隐式转换时

运算符重载，有操作结果返回

：：

如果临时对象被绑定到const引用上，作为参数或者返回值生命周期会被延长

140、C++返回值优化

返回值优化：直接返回构造函数

命名返回值优化：返回 创建的类对象

141、局部static变量为什么线程安全？只有局部变量static能线程安全。

static只会进行一次初始化

不同的线程访问static变量时，访问的是同一内存地址

staic，在初始化时会加锁，确保同一时刻只有一个线程能执行初始化

145、memcpy和strcpy的区别

memcpy：用于内存块拷贝，可以拷贝任意类型的数据，需指定拷贝的字节数，适用于二进制数据

strcpy：字符串拷贝，拷贝直到遇到\0结束符，适用于字符串拷贝

146、在成员函数中调用delete this会出现什么问题？

this会将函数被哪个对象调用传递给函数，如果函数不嗲用this指针的任何函数，就不会报错，调用就会报错