c＋＋基础

vector、list、map这些，复杂度，内部实现，什么场景用什么基本必问，

虚函数、内存布局、对象切割，偶尔问问友元模板啥的，还有野指针、单例、为什么要虚析构、一个对象里几个虚表指针

算法

好几家都是先说说快排的思路，然后其他一些基本排序偶尔也问，但是不多。有家问了个记录栈的最大值没答上来，有家问的两个栈组成一个队列，还有个战力榜实现，这玩意就无脑扯扯堆排序好吧。

网络

握手挥手基本都问，握手挥手中间有一次丢掉怎么办，处于什么状态，问就是time＿wait2和重传。tcp／udp什么区别，tcp自带心跳和应用层心跳啥的。

数据库

虽然也没啥可问的，但基本都提一嘴，答增删改查授权这些用一用，其他都不怎么用，往这个方向引大概率是想问索引，索引原理。

操作系统

线程进程区别，锁、互斥元9、条件变量这些基本都要提一下，然后自己主动扯扯原子操作、线程屏障也就混过去了

# 高频被问(41条)

1.空类占用内存空间：1字节

2.explicit作用： 关闭函数的类型自动转换（防止隐式转换）

3.当初始化列表时，被初始化的顺序是声明是的顺序不是列表顺序。

4.命名空间 作用：解决同名冲突，使用

a. 方法一：使用命名空间名称::标识符的方式来访问

b. 方法二：使用命名using namespace 命名空间名称;的方式作前置声明，在声明之后，可以直接使用标符来访问。

c. 特点：允许不连续的命名空间、允许嵌套

类：具有相同的属性和行为的对象的集合。从

访问权限：public（共有的） protected(保护的) private(私有的)

5.函数缺省参数注意事项：

a.如果某个位置已经有了默认参数，那么从这个位置往后，从左往右都必须有默认值.如:func(int a, int b = 20, int c)错误

b.缺省值放在声明位置

6.拷贝构造函数（一般类中有指针才需要自己实现）

三种场景：

无须实现

a. 无指针

需要实现

a. 一般类中有指针才需要自己实现

无法实现

a. File(const File& other) = delete;//表示禁止使用 拷贝构造

7.浅拷贝：简单的赋值拷贝操作

深拷贝：在堆区重新申请空间，进行拷贝操作

浅拷贝常见问题：堆区数据重复释放

8.成员变量的初始化

构造函数体内

构造函数的初始化列表 (速度快)

在成员变量声明时

9.类对象作为类成员

c++类的成员可以是另一个类的的对象，我们称该成员为 对象成员

10.静态成员、函数

概念：成员变量和成员函数前加一个static，称为静态成员

目的：为了实现一个类的不同对象之间的数据和函数共享。

静态成员变量特点：

• 所有对象共享同一份数据

• 在编译阶段分配内存

• 类内的声明，类外初始化

a. 语法：类型 类名::静态变量 = 表达式

b. 访问：对象.静态成员变量 类名.静态成员变量

静态成员函数：

a.所有对象共享同一个函数

b.静态成员函数只能访问静态成员变量

11. 在C++中，类内的成员变量和成员函数分开存储、只有非静态成员变量才属于类的对象上

12. this指针：当形参和成员变量同名时，可用this指针来区分(解决同名冲突)、在类的非静态成员函数中返回对象本身，可使用return \* this

13. const修饰成员函数

常函数：

a.成员函数后加const，叫常函数

b.常函数内不可以修改成员属性

c.成员属性声明时加关键词mutable后，在常函数依然可以修改

常对象：

a.声明对象前加const,叫常对象

b.常对象只能调用常函数

14. 静态多态

重载(在编译时期就可以通过函数名和参数确定需要调用那个函数)

模板

动态多态

* 虚函数(通过运行阶段才能知道需要调用那个对象)，虚函数是带有 virtual 关键字的成员函数，构造函数和静态函数不能是虚函数。

纯虚函数，完全没有函数体的虚函数，函数声明后=0，包含纯虚函数的类称作抽象类，抽象类不能实例化。

虚析构函数

虚函数表

15. 虚析构作用：使用父类指针释放子类对象时可以让子类的析构函数和父类的析构函数同时被调用到。

16. 虚析构和纯虚析构共性：

• 可以解决父类指针释放子类对象

• 都需要具体的函数实现

17. 虚析构语法：virtual ~类名（）{}；

纯虚析构语法： virtual ~类名（） = 0；

纯虚析构实现 类名：：~类名（）{}

18. 继承方法

访问权限

• public：在子类和外类可以访问

• protected：在子类中可以访问，外类不可以访问

• private：在子类和外部都不可以访问，仅自己可以访问

继承方式：

子类权限 => 父类权限与继承权限中取严谨值

影响的是外面对该子类的访问权限以及当该子类当父类时，别的子类的访问权限。

19. 创建对象时：String s1;  
a. 分配空间  
b. 调用构造函数

20. **一个类可以没有缺省构造函数，但最多有一个 无缺构造函数时，无法创建对象数组**

21.匿名对象 A() 生命周期只有本行(当前行执行结束后，系统会立即回收掉)。

22.\*\*函数重载作用：\*\*达到行为标识符统一，减少程序中标识符的个数

23.父类中所有非静态成员属性都会被子类继承下去

父类中私有属性 是被编译器隐藏了 因此访问不到 但是的确继承下去了

24.多继承初始化顺序：从左向右

25.菱形继承：一个类被俩个类继承 这俩个又被一个类继承

26.\*\*无法继承的内容：\*\*构造函数、析构函数、友元、运算符重载

27.子类对象可以赋给父类的指针(引用)

**28.final关键字** 用处：当前我这个类就是最终类，我不想让别的类再继承我自己。

32.继承语法：class 子类 ：public 父类 目的：减少重复代码(代码复用) 实现方式：单继承、多继承

33.重载：作用域相同函数名相同

34.重写：父类的方法，子类重写，要求父类的该方法必须是虚函数或者纯虚函数virtual

35.隐藏：父类的方法，子类重写，要求父类的该方法不能被virtual修饰

36.在当前类的初始化列表调用基类的构造函数

37.类是具有相同属性和行为的一组对象的集合

38.封装是将抽象出的属性成员、行为成员相结合，将它们视为一个整体。

目的是增强安全性和简化编程，使用者不必了解具体的实现细节，而只需要通过外部接口（函数），以特定的访问权限，来使用类的成员。

39

40.浅拷贝和深拷贝有什么区别？

浅拷贝只复制指向某个对象的指针，而不复制对象本身，新旧对象还是共享一块内存；而深拷贝会创造⼀个相同的对象，新对象与原对象不共享内存，修改新对象不会影响原对象。

41.虚继承 作用：解决多继承时产生的菱形继承问题

## 内存五大分区(超级重要)

表格

中度可信度描述已自动生成

内存总共分为五大分区：栈区，堆区，全局静态区，常量文本区，程序代码区。

从生存周期来说：栈区是运行时分配，结束就释放。堆区是程序员主动分配和释放。全局静态区是程序运行前分配，程序结束释放。常量文本区也是程序运行前分配，程序结束释放

从管理角度来说：只有堆区是由程序员自己来管理，其他四个区都是由系统来管理的。

从可能产生的问题来说：栈区可能会造成栈溢出(1、深度递归，栈帧太多 2、只分配不释放，栈耗尽)。堆区可能造成内存泄漏(就是申请空间，然后忘记释放了)，野指针（指向了一块已经被释放的空间），内存碎片。

从效率来说：栈区比堆区效率高

从内存大小来说：栈区大小m，堆区大小和内存有关。最大可1G。其他三个区都很小

代码区：存放程序体的二进制代码。比如我们写的函数，都是在代码区的。

## C++与C的区别

C是面向过程的语言，而C++是面向对象的语言

• C只能写面向过程的代码，而C++既可以写面向过程的代码，也可以实现面向对象的代码

• C和强制类型转换上也不一样 const\_cast static\_cast reinterpret\_cast dynamic\_cast

• C和C++的输入输出方式也不一样

• C++引⼊入 new/delete 运算符，取代了了C中的 malloc/free 库函数；

• C++引⼊入引⽤用的概念

• C++引⼊入类的概念

• C++引⼊入函数重载的特性

## 静态链接和动态链接有什么区别？

静态链接

静态链接是在编译链接时直接将需要的执行代码拷贝到调⽤用处；

优点: 在于程序在发布时不需要依赖库，可以独立执行，

缺点: 在于程序的体积会相对较大，⽽而且如果静态库更更新之后，所有可执行文件需要重新链接；

动态链接

动态链接是在编译时不直接拷贝执行代码，而是通过记录一系列符号和参数，在程序运⾏行行或加载时将这些信息传递给操作系统，操作系统负责将需要的动态库加载到内存中，然后程序在运行行到指定代码时，在共享执行内存中寻找已经加载的动态库可执⾏代码，实现运行时链接；

优点 在于多个程序可以共享同一个动态库，节省资源；

缺点 在于由于运行时加载，可能影响程序的前期执行性能

## 指针和引用的区别

表格

描述已自动生成

## 类和结构体的区别

**类和结构体的区别**：struct默认访问权限公有 class默认访问权限私有  
struct更适合看成是一个数据结构的实现体，class更适合看成是一个对象的实现体。  
C++之所以要引入结构体，是为了保持和C程序的兼容性。

## define与inline区别

图形用户界面, 文本, 应用程序

描述已自动生成

## 前置++与后置++区别

后置++中tmp是一个临时对象，会造成一次构造函数和一次析构函数的额外开销  
效率高：前置++，不产生临时对象

## 友元　friend

友元：让一个函数或者类，访问另一个类的私有成员(打破封装)  
三种实现：  
• 全局函数做友元  
• 类做友元(友元类)  
• 成员函数做友元

# 关键字

## 1.static

1. 隐藏；未加static的全局变量和函数是全局可见的，当同时编译多个文件时，文件之间的变量或者函数可能会有命名冲突的情况，可以使用static来隐藏，使得变量或函数只能在该文件可见。
2. 改变变量的存储方式和生命周期；static 关键字会使得它修饰的变量存储在静态存储区，在程序开始运行是就初始化，并且会一直存在直到程序结束，并且不会被再次初始化。
3. 对类成员声明 static，同样有上述的几种功能，对该成员函数or静态变量进行隐藏以及改变其生命周期，并且所有的对象共用一个静态成员函数或静态变量。另外静态内部类也是实现单例模式的一种方法，其优点在于外部类加载时并不需要立即加载内部类，内部类不被加载则不去初始化 INSTANCE，故而不占内存，只有当 getInstance() 方法第一次被调用时，才会去初始化 INSTANCE。

## 2.const

1. 常量；const 用来表示常量，即不允许他修饰的变量被修改，在进行参数传递的时候，也常常使用const修饰引用来传参，既保证了不复制参数，又防止对象被修改。
2. 封装；C++ 中还可以用 const 修饰成员函数和数据成员来达到封装的目的，不允许使用const修饰的成员函数来修改数据成员。

## 3.extern

extern表明函数和全局变量作用范围（可见性）的关键字，该关键字告诉编译器，其声明的函数和变量可以在全局使用(只是声明，不分配内存) ，被extern "C"修饰的变量和函数是按照C语言方式编译和链接的

与 extern 对应的关键字是 static，被它修饰的全局变量和函数只能在本模块中使用。因此，一个函数或变量只可能被本模块使用时，其不可能被 extern “C”修饰。

之所以要如此是因为编译器在将C++源代码编译成目标文件时，会将函数和变量的名称进行修饰，形成符号名，目标文件中所使用的符号名就是修饰后名称，所以对于不同函数签名的函数，即使函数名相同，编译器和链接器都认为它们是不同的函数（重载）。因此有时候在调用一些库函数的时候，会出现无法链接的情况(因为函数签名不同)所以对于C++来说，必须使用extern "C"来声明这些函数。

# 底层原理

## 1.RTTI；

RTTI是运行时类型检测，也是 C++11 的新特性，主要是在dynamic\_cast、typeid中使用，在进行类型转换前或者获取类型时，会查询type\_info指针来进行类型检测。

## 2.cast转换

* const\_cast，用于将 const 变量转换为非 const
* static\_cast，用于隐式转换，如非 const 转 const，void\*转指针等，能用于多态向上转化，向下不安全
* static\_cast，动态类型转换，可以用于层次间的向上向下转化，只能转指针和引用
* reinterpret\_cast，什么都可以转，可能会出问题

# 虚函数（多态）

## 1.多态及其实现条件

c++ 中的多态就是在父类的成员函数前加上 virtual 关键字，在子类中重写该函数，运行时将会根据对象的实际类型来调用相应的函数，如果对象类型是子类，就调用子类的函数，如果对象类型是父类，就调用父类的函数。

多态的实现需要两个条件：

* 虚函数重写，设置不同的状态
* 对象调用虚函数时必须是指针或者引用

多态指多种形态，同一个方法，根据对象不同执行不同的行为；

分类： 静态多态，通过函数重载(参数不同)和运算符重载实现，函数地址在编译时确定

动态多态，通过派生类和虚函数实现，在运行阶段确定函数地址

动态的实现方式：继承，方法重写，父类引用指向子类对象。

1.父类在成员函数前+virtual，将函数变成虚函数。

2.根据父类继承出一个子类，子类重写父类的虚函数，参数，函数名，返回值相同

3.创建父类指针，指向子类对象 （eg: Animal a1 = new Dog()）

优点：

• 代码组织结构清晰

• 可读性强

• 利与前期和后期的扩展和维护

## 2.虚函数与重写

* 虚函数是带有 virtual 关键字的成员函数
* 子类有个和父类完全相同(函数名,形参,返回值都相同,协变和析构函数除外)的虚函数,就称子类虚函数重写父类虚函数

## 3.多态的原理

* 多态是用虚函数表来实现的
* 有虚函数的类都会生成一个虚函数表，在编译的时候使用
* 虚函数表是一个存储虚函数地址的数组
* 生成子类的虚函数表需要经过以下步骤
* 将父类的虚函数表拷贝
* 将子类中重写的虚函数覆盖掉父类中的虚函数
* 如果有新增加的虚函数则放到表的最后
* 在调用时会根据对象中的虚表指针来找实际应该调用的函数

## 4.静态多态（编译时多态）与动态多态（运行时多态）分别是怎么实现的？有什么区别？

动态多态的设计思想：

对于相关的对象类型，确定它们之间的一个共同功能集，然后在基类中，把这些共同的功能声明为多个公共的虚函数接口。各个子类重写这些虚函数，以完成具体的功能。客户端的代码（操作函数）通过指向基类的引用或指针来操作这些对象，对虚函数的调用会自动绑定到实际提供的子类对象上去。

静态多态的设计思想：

对于相关的对象类型，直接实现它们各自的定义，不需要共有基类，甚至可以没有任何关系。只需要各个具体类的实现中要求相同的接口声明，这里的接口称之为隐式接口。客户端把操作这些对象的函数定义为模板，当需要操作什么类型的对象时，直接对模板指定该类型实参即可（或通过实参演绎获得）

动态多态和静态多态的比较

静态多态

优点：

由于静多态是在编译期完成的，因此效率较高，编译器也可以进行优化；

有很强的适配性和松耦合性，比如可以通过偏特化、全特化来处理特殊类型；

最重要一点是静态多态通过模板编程为C++带来了泛型设计的概念，比如强大的STL库。

缺点：

由于是模板来实现静态多态，因此模板的不足也就是静多态的劣势，比如调试困难、编译耗时、代码膨胀、编译器支持的兼容性

不能够处理异质对象集合

动态多态

优点：

OO设计，对是客观世界的直觉认识；

实现与接口分离，可复用

处理同一继承体系下异质对象集合的强大威力

缺点：

运行期绑定，导致一定程度的运行时开销；

编译器无法对虚函数进行优化

笨重的类继承体系，对接口的修改影响整个类层次；

不同点：

本质不同，静态多态在编译期决定，由模板具现完成，而动态多态在运行期决定，由继承、虚函数实现；

动态多态中接口是显式的，以函数签名为中心，多态通过虚函数在运行期实现，静态多台中接口是隐式的，以有效表达式为中心，多态通过模板具现在编译期完成

相同点：

都能够实现多态性，静态多态/编译期多态、动态多态/运行期多态；

都能够使接口和实现相分离，一个是模板定义接口，类型参数定义实现，一个是基类虚函数定义接口，继承类负责实现。

## 5.其他问题

虚函数和普通函数一样在代码段,虚表在只读常量区

inline 函数(内联函数)没有地址,无法放到虚函数表中

静态成员不能是虚函数，因为静态成员函数没有this指针,因为有 this 指针才能访问到虚表指针,有虚表指针才能找到虚表从而调用实际应该调用的函数

构造函数不能是虚函数，因为对象中的虚函数表指针是在构造函数初始化列表阶段才初始化的

析构函数可以是虚函数，并且最好把基类的析构函数定义成虚函数，当父类指针指向子类对象时,如果析构函数不是虚函数,析构就只会释放父类对象,造成内存泄漏

普通对象访问普通函数和访问虚函数是一样快的(不会触发多态);指针对象或者是引用对象，调用普通函数更快一些，因为构成了多态，运行时调用虚函数要先到虚函数表中去查找。这样然后才拿到韩式的地址，这样就不如直接可以拿到函数地址的普通函数快。

# 内存管理与内存泄露

可以通过静态扫描来检测内存泄露，具体的操作为对象计数；重载 new/delete，记录分配点等

解决内存泄露可以使用智能指针，但要小心循环引用的问题。

# C++11特性

## 1.Auto

auto实际上很早就被引入到 C++，用来进行类型推导，比如用在迭代器中。

## 2.Nullptr

传统的 C++ 无法区分 NULL 和 0，有的编译器会把 NULL 定义为 void\*的指针，有的直接就定义为0了，这就会导致一系列的问题，比如在重载时会发生混乱，所以 C++11 引入了 nullptr 来区分空指针和0。

## 3.基于范围的 for 循环

进行遍历的时候可以像python一样写auto iter:arr来遍历。

## 4.初始化列表

提供了统一的语法来初始化任意的对象,这两个特性就让人感觉 C++ 越来越智能而简洁了

## 4.智能指针

C++11 引入智能指针主要是因为C++没有自动的内存回收机制，所以当业务逻辑过于复杂的时候难免会出现内存没有 delete 的现象，这就容易造成内存泄漏的问题，智能指针主要就是被用来对这些资源进行动态管理，并及时释放无用的资源。

C++11 提供四种智能指针，有unique\_ptr, shared\_ptr, weak\_ptr,以及auto\_ptr，不过auto\_ptr已经被弃用。

shared\_ptr就像引用，允许多个指针指向相同的对象，每次被引用就内部计数+1，析构则-1，当引用计数为0时则释放内存。

unique\_ptr只允许一个指针指向给定的对象，

weak\_ptr最大的作用在于协助shared\_ptr，可以观测目标对象的引用计数。

智能指针会遇到的最大问题是循环引用，就像是类之间的循环引用一样，两个指针互相引用对方就会造成都无法析构的现象。

循环引用的现象可以通过让引用链上的一方持用普通指针或弱智能指针 (weak\_ptr) 来解决。

实现智能指针可以通过引入辅助类或者使用句柄。

辅助类实现即单独使用一个类来存储引用计数，每次在构造和析构智能指针类的时候对辅助类进行操作。

而句柄实现则是把指针封装起来，通过重载来重新定义指针的行为，将计数内置进去

## 5.右值引用

C++11 引入右值引用的目的是解决拷贝临时对象时的性能低下问题，在 C++11 之前，进行临时对象拷贝时会多调用一次构造函数并复制，这就使得性能低下。实际上直接移动变量的指向就会快得多，但 C++11 之前右值是不能被取地址的，所以没法进行右值引用。

从本质上来说，左值引实际上是用const 指针来实现的，它保存的是一个指向数据地址的指针，而如果进行反汇编可以发现右值引用是通过取地址来实现的，一个变量的右值引用存储的就是它的地址。

而对于传入的引用，在不知道是左值还是右值的情况下，可以用move函数来将其转换为右值，也就是实现了取地址的操作。

## 5.新增 STL 容器 array 以及 tuple(元组)

array 保存在栈内存中，相比堆内存中的 vector，速度更快，但 array 长度无法更改，没有 vector 灵活。

而新增了 tuple 感觉就是 C++ 在像 python 看齐