#### 封装、继承、多态是什么？

封装：将实现过程和数据封装成一个类，使类成为一个功能独立的软件模块，这些数据只能通过接口进行访问，这样做的好处是可以保护类中的成员，不让类中以外的程序直接访问或者修改。

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，修饰的成员函数不能重写或修改。意义：基类的程序代码可以被派生类复用，提高了软件复用的效率，缩短了软件开发的周期

多态：如果在基类的成员函数前加virtual 关键字，把它声明为虚函数，基类指针就可以调用派生类中同名的成员函数，通过派生类中同名的成员函数，就可以访问派生对象的成员变量。有了虚函数，基类指针指向基类对象时就使用基类的成员函数和数据，指向派生类对象时就使用派生类的成员函数和数据，基类指针表现出了多种形式，这种现象称为多态。

#### 多态的实现原理,多态的优点

动态多态是通过虚函数实现的。如果基类中有虚函数，创建对象的时候除了给对象的成员分配内存，还会创建虚函数表。创建派生类对象时，将用派生类的函数取代虚函数表中基类的函数。如果调用派生类对象，表里面的地址已经换成派生类的函数地址，效果就是调用派生类的函数。

而静态多态也成为编译时的多态；在编译时期就已经确定要执行了的函数地址了；主要有函数重载和函数模板。

#### 虚析构函数

构造函数和析构函数都不能被继承，创建派生类对象时，先执行基类构造函数，再执行派生类构造函数。而销毁派生类对象时，先执行派生类析构函数，再执行基类析构函数。

当用基类指针指向派生类对象时，delete基类指针调用的是基类的析构函数，不是派生类的，如果希望调用派生类的析构函数，就要把基类的析构函数设置为虚函数。（多态的原理？）

**注意**

析构函数可以手工调用，如果对象中有堆内存，析构函数中以下代码是必要的：

delete ptr;

ptr=nulllptr;

（有可能手工调用）(C++在使用 delete 或 delete[] 后，指针本身不会变为 nullptr，它仍然保留之前的值。这意味着指针仍然指向先前分配的[内存地址](https://so.csdn.net/so/search?q=%E5%86%85%E5%AD%98%E5%9C%B0%E5%9D%80&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/weixin_73964834/article/details/_blank)，但是这块内存已经被释放。)

#### 纯虚函数和抽象类

在某些情况下，[基类](https://baike.baidu.com/item/%E5%9F%BA%E7%B1%BB/9589663" \t "_blank)中不能对[虚函数](https://baike.baidu.com/item/%E8%99%9A%E5%87%BD%E6%95%B0/2912832" \t "_blank)给出有意义的[实现](https://baike.baidu.com/item/%E5%AE%9E%E7%8E%B0/10801654" \t "_blank)，把它声明为纯虚函数。

语法：virtual 返回值类型 函数名 (参数列表)=0;

[纯虚函数](https://so.csdn.net/so/search?q=%E7%BA%AF%E8%99%9A%E5%87%BD%E6%95%B0&spm=1001.2101.3001.7020" \t "_blank)在基类中为派生类保留一个函数的名字，以便派生类它进行重定义。如果在基类中没有保留函数名字，主要用于支持[多态](https://so.csdn.net/so/search?q=%E5%A4%9A%E6%80%81&spm=1001.2101.3001.7020" \t "_blank)性。含有[纯虚函数](https://so.csdn.net/so/search?q=%E7%BA%AF%E8%99%9A%E5%87%BD%E6%95%B0&spm=1001.2101.3001.7020" \t "_blank)的类被称为抽象类，不能实例化对象，可以创建指针和引用。有时候，想使一个类成为抽象类，但刚好又没有任何纯虚函数，怎么办？将它的析构函数声明为纯虚析构函数。（有必要非要整成抽象类吗，没好处吧？）

#### 虚函数存放在哪里在内存的哪个区？什么时候生成的

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。

#### 构造函数是否能声明为虚函数？

虚函数的调用是通过虚函数表指针vptr指向的虚函数表，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。

#### 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。而虚函数靠vptr，vptr靠构造函数，且只能通过this指针访问。因此static函数无法声明为虚函数。

#### 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

#### 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，作用是关闭自动类型转换。

#### 线程之间的通信方式有哪些？

信号量

条件变量

互斥量

#### 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

#### 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。

#### 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。

#### 右值引用是什么，为什么要引入右值引用？

右值引用就是给右值取名字，左值是指表达式结束后依然存在的持久化对象，右值是指表达式结束后就不再存在的临时对象。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。

引入右值引用是为了支持移动语义，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，从而避免了深拷贝，实现高效的移动语义。

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。

#### 移动语义

#### 完美转发

在函数模板中，将参数转发给其它函数，过程中保证被转发参数的左、右值属性不变。

1.如果模板中（包括类模板和函数模板）函数的参数书写成为T&& 参数名，那么，函数既可以接受左值引用，又可以接受右值引用。（但是接受之后都会变成左值）

2.用forward<T>(参数)转发参数，如果 参数是一个右值，转发之后仍是右值引用；如果参数是一个左值，转发之后仍是左值引用。

#### 引用和指针的区别？

引用的本质是指针常量，指向性不可变的指针。指针可以不用初始化，引用必须在声明的时候初始化，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。

#### 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

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.-。

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，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。

#### 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？

限定CPU只能在特定的地址访问数据。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。

#### 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。

#### 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。

#### 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：野指针就是指针指向的不是一个有效（合法）的地址。

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。

#### push\_back()和emplace\_back()的区别、以及使用场景

Push\_back是将已经存在的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。

#### 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

#### 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。

#### 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？

回调函数是作为别的函数的参数的函数，别的函数在运行时来调用的函数。可以把调用者和被调用者分开。

优点是可以让调用方根据回调方的多种形态实现不同的操作，代码的框架更稳定，让代码的逻辑更加集中，更加易读。优于虚函数。

缺点是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。

#### 哈希表

存储结构采用的是数组链表，通过数据元素的key值用哈希函数计算，得出其下标对应的数组下标。数组长度称为哈希表长或者桶的个数。装填因子是元素总个数/桶的个数，值越大，效率越低。

#### unordered\_map

容器封装了哈希表，无序的表。

#### 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原始指针的类模板，通过类的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。

#### 智能指针Unique\_ptr

需要包含头文件#include<memory。unique\_ptr主要属性是独享，也就是说，同时只有一个unique\_ptr指向同一个对象，当这个unique\_ptr被销毁时，指向的对象也随即被销毁。智能指针的本质是一个类模板，为了适应独享特性，unique\_ptr类禁用了拷贝构造函数和赋值函数，且不可以隐式转换。当他用于函数的参数,一般使用传引用的方式，因为unique\_ptr没有拷贝构造函数。 有一种情况允许使用=赋值，当将另一个unique\_ptr赋给这个unique\_ptr，且另一个unique\_ptr是一个临时右值

#### 智能指针Shared\_ptr

需要包含头文件#include<memory。shared\_ptr智能指针也是一个类模板，但是shared\_ptr共享它指向的对象，多个shared\_ptr可以指向（关联）相同的对象，在内部采用计数机制来实现。当新的shared\_ptr与对象关联时，引用计数增加1。当shared\_ptr超出作用域时，引用计数减1。当引用计数变为0时，则表示没有任何shared\_ptr与对象关联，则释放该对象。

use\_count()方法返回引用计数器的值。

如果unique\_ptr能解决问题，就不要用shared\_ptr。unique\_ptr的效率更高，占用的资源更少。

#### 智能指针weak\_ptr

shared\_ptr内部维护了一个共享的引用计数器，多个shared\_ptr可以指向同一个资源。

如果出现了循环引用的情况，引用计数永远无法归0，资源不会被释放。

它指向一个由shared\_ptr管理的资源但不影响资源的生命周期。也就是说，将一个weak\_ptr绑定到一个shared\_ptr不会改变shared\_ptr的引用计数。

不论是否有weak\_ptr指向，如果最后一个指向资源的shared\_ptr被销毁，资源就会被释放。

weak\_ptr不控制对象的生命周期，但是，它知道对象是否还活着。

用lock()函数把它可以提升为shared\_ptr，如果对象还活着，返回有效的shared\_ptr，如果对象已经死了，提升会失败，返回一个空的shared\_ptr。

提升的行为（lock()）是线程安全的。

#### Lambda函数

捕获列表+参数列表+函数选项+返回值类型+函数体。通过捕获列表，lambda函数可以访问父作用域中的局部变量（静态局部变量可以直接访问，不能访问全局变量）

[=] 值捕获 [&] 引用捕获

#### 匿名函数（lambda）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：免去声明和定义。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，更节省空间。

通过adc模块测出芯片的电压，电流，温度监控

[9.STM32之ADC测电池电压程序实现\_哔哩哔哩\_bilibili](https://www.bilibili.com/video/BV1iS4y1E7vs/?spm_id_from=333.337.search-card.all.click&vd_source=ffb1a92f7203f91f34b88b358b6b8124)

PA4设置为mode\_ain(模拟输入)

寄存器写入和读取

DMA信号验证

日志文件：

时间，序号，故障类型

#### orscle和mysql的特点