**C++面试总结之常用基础知识**

[](https://www.jianshu.com/u/4ca8507b95ec)

[大叔龙](https://www.jianshu.com/u/4ca8507b95ec)

2015.12.15 14:04\* 字数 4837 阅读 4073评论 4喜欢 39赞赏 1

C++程序员面试一般都是以下三板斧  
1.基础问答  
2.然后一顿虚函数、虚函数表、纯虚函数、抽象类、析构函数、拷贝构造函数  
3.操作数重载、STL、智能指针

－－－－－－－－－－－－－－－－－－－分割线－－－－－－－－－－－－－

* 问题1，你知道虚函数吗？  
  答案：实现多态所必须，父类类型的指针指向子类的实例，执行的时候会执行之类中定义的函数。
* 问题2，析构函数可以是虚函数吗？  
  答案： 如果有子类的话，析构函数必须是虚函数。否则析构子类类型的指针时，析构函数有可能不会被调用到。
* 问题3，多态的实现。  
  答案：简而言之编译器根据虚函数表找到恰当的虚函数。对于一个父类的对象指针类型变量，如果给他赋父类对象的指针，那么他就调用父类中的函数，如果给他赋子类对象的指针，他就调用子类中的函数。函数执行之前通过查虚函数表来查找调用的函数。
* 问题5，虚函数表是针对类还是针对对象的?  
  答案：虚函数表是针对类的，一个类的所有对象的虚函数表都一样。
* 问题6，纯虚函数和虚函数有什么区别  
  答案：  
  A. 纯虚函数就是定义了一个虚函数但并没有实现，原型后面加"=0"。  
  B.包含纯虚函数的类都是抽象类，不能生成实例。  
  C.在虚函数表中春夏函数
* 问题7，构造函数可以是虚函数吗？  
  答案：构造函数不能是虚函数,每个对象的虚函数表指针是在构造函数中初始化的，因为构造函数没执行完，所以虚函数表指针还没初始化好，构造函数的虚函数不起作用，所以构造函  
  数不能是虚函数。

问题8，构造函数中可以调用虚函数吗？ 答案：就算调用虚函数也不起作用，调用虚函数同调用一般的成员函数一样。

问题9，析构函数中可以调用虚函数吗？  
答案： 可以调用会跟普通函数一样的，析构函数中调用虚函数也不起作用，调用虚函数同调用一般的成员函数一样。析构函数的顺序是先派生类后基类，有可能内容已经被析构没了，所以虚函数不起作用。

问题10，构造初始化的执行顺序，析构函数的执行顺序?  
构造时  
父类的构造函数 －－> 子类的构造函数  
析构时  
子类的析构函数 －－> 父类的析构函数

问题11.  
除了栈以外，堆、只读数据区、全局变量地址增长方向都是从低到高的

问题12.  
struct和class的区别  
struct和class大体相似，但是区别的访问权限  
struct默认的访问权限是public，class默认是private

问题13.C＋＋内存管理方式  
堆、栈、自由存储区、全局\\静态存储区、常量存储区  
自由存储区存储malloc申请的内存  
问题14.C++是不是类型安全的?  
不是两个不同类型的指针之间可以强制转换（用reinterpret cast)。

问题15.一个类没有定义属性和函数sizeof大小是多少，编辑器为什么这么做？  
一个空类对象的大小是1byte。这是被编译器安插进去的一个字节，这样就使得这个空类的实例得以在内存中配置独一无二的地址

问题16.函数内存空间放在哪里？  
放在代码段里

问题17.局部变量可否与全局变量重名？  
可以，类型也可以不同

问题18.引用和指针有什么区别？  
1.引用必须初始化  
2.引用不能为空  
3.引用在赋值之后不能不能修改  
4.指针通过某个指针变量指向一个对象后，对它所指向的变量间接操作,而引用是相应对象的别名。

问题19.将引用作为函数参数和返回值有哪些特点？有哪些注意事项？  
作为函数参数时候不产生拷贝，直接传递内存地址的值。  
作为返回值的时候必须的有效的，而指针可以是无效，例如空指针等等

问题20.阐述extern "C"和extern的作用？  
extern "C"告知编译器以C的形式编译，因为c中没有函数重载，函数在底层的签名是**函数名**，而c＋＋的函数签名是**函数名**返回值**参数类型1**参数类型2\_\_  
***extern作用声明外部变量***，现在现代编译器一般采用按文件编译的方式，因此在编译时，各个文件中定义的全局变量是互相透明的，也就是说，***在编译时，全局变量的可见域限制在文件内部。***

问题21.多态的作用  
多态的作用：

1. 应用程序不必为每一个派生类编写功能调用，只需要对抽象基类进行处理即可。大大提高程序的可复用性。//继承
2. 派生类的功能可以被基类的方法或引用变量所调用，这叫向后兼容，可以提高可扩充性和可维护性。 //多态的真正作用，以前需要用switch实现

问题22.内存对齐  
[http://www.cppblog.com/cc/archive/2006/08/01/10765.html](https://link.jianshu.com?t=http:/www.cppblog.com/cc/archive/2006/08/01/10765.html)  
(PS:内存块的声明是按照内存变量书写顺序来设定的)

问题23.什么是内联函数  
内联函数是指用inline关键字修饰的函数。在类内定义的函数被默认成内联函数。内联函数从源代码层看，有函数的结构，而在编译后，却不具备函数的性质。内联函数不是在调用时发生控制转移，而是在编译时将函数体嵌入在每一个调用处。编译时，类似宏替换，使用函数体替换调用处的函数名。一般在代码中用inline修饰，但是能否形成内联函数，需要看编译器对该函数定义的具体处理。

问题24.内联函数和宏定义的区别？  
内联函数在编译时展开，宏在预编译时展开。

在编译的时候，内联函数可以直接被镶嵌到目标代码中，而宏只是一个简单的文本替换。  
内联函数可以完成诸如类型检测、语句是否正确等编译功能，宏就不具有这样的功能。

问题25.为什么需要内存对齐？!!!  
为什么要字节对齐

需要字节对齐的根本原因在于CPU访问数据的效率问题。假设上面整型变量的地址不是自然对齐，比如为0x00000002，则CPU如果取它的值的话需要访问两次内存，第一次取从0x00000002-0x00000003的一个short，第二次取从0x00000004-0x00000005的一个short然后组合得到所要的数据，如果变量在0x00000003地址上的话则要访问三次内存，第一次为char，第二次为short，第三次为char，然后组合得到整型数据。而如果变量在自然对齐位置上，则只要一次就可以取出数据。一些系统对对齐要求非常严格，比如sparc系统，如果取未对齐的数据会发生错误.

问题26.计算类的大小  
class A  
{  
};  
class B  
{  
public:  
B() {}  
~B() {}  
};  
class C  
{  
public:  
C() {}  
virtual ~C() {}  
};

int \_tmain(int argc, \_TCHAR\* argv[])  
{  
printf("%d, %d, %d\n", sizeof(A), sizeof(B), sizeof(C));  
return 0;  
}  
分析：答案是1, 1, 4。class A和B是一个空类型，它的实例不包含任何信息，本来求sizeof应该是0。但当我们声明该类型的实例的时候，它必须在内存中占有一定的空间，否则无法使用这些实例。至于占用多少内存，由编译器决定。Visual Studio 2008中每个空类型的实例占用一个byte的空间。

class C在class B的基础上把析构函数标注为虚拟函数。C++的编译器一旦发现一个类型中有虚拟函数，就会为该类型生成虚函数表，并在该类型的每一个实例中添加一个指向虚函数表的指针。在32位的机器上，一个指针占4个字节的空间，因此sizeof(C)是4

问题27. 描述内存分配方式以及它们的区别 ?  
1 ） 从静态存储区域分配 。内存在程序编译的时候就已经分配好，这块内存在程序的整个运行期间都存在。例如 全局变量， static 变量 。  
2 ） 在栈上创建 。在执行函数时， 函数内局部变量的存储单元都可以在栈上创建 ，函数执行结束时这些存储单元自动被释放。栈内存分配运算内置于处理器的指令集。  
3 ） 从堆上分配 ， 亦称动态内存分配 。程序在运行的时候用 malloc 或 new 申请任意多少的内存，程序员自己负责在何时用 free 或 delete 释放内存。动态内存的生存期由程序员决定，使用非常灵活，但问题也最多。

问题28  
简述数组与指针的区别？  
数组要么在静态存储区被创建（如全局数组），要么在栈上被创建。指针可以随时指向任意类型的内存块。  
(2) 用运算符 sizeof 可以计算出数组的容量（字节数）。 sizeof(p),p 为指针得到的是一个指针变量的字节数，而不是 p 所指的内存容量 。 C++/C 语言没有办法知道指针所指的内存容量，除非在申请内存时记住它。 注意当数组作为函数的参数进行传递时，该数组自动退化为同类型的指针。

问题29指针常量和常量指针



image.png

问题30.static 和const分别怎么用，类里面static和const可以同时修饰成员函数吗。  
static的作用：  
对变量：  
1.局部变量：  
在局部变量之前加上关键字static，局部变量就被定义成为一个局部静态变量。

1）###内存中的位置：静态存储区###

2）初始化：未经初始化的全局静态变量会被程序自动初始化为0（自动对象的值是任意的，除非他被显示初始化）

3）作用域：作用域仍为局部作用域，当定义它的函数或者语句块结束的时候，作用域随之结束。

注：当static用来修饰局部变量的时候，它就改变了局部变量的存储位置（从原来的栈中存放改为静态存储区）及其生命周期（局部静态变量在离开作用域之后，并没有被销毁，而是仍然驻留在内存当中，直到程序结束，只不过我们不能再对他进行访问），但未改变其作用域。

2.全局变量

在全局变量之前加上关键字static，全局变量就被定义成为一个全局静态变量。

1）内存中的位置：静态存储区（静态存储区在整个程序运行期间都存在）

2）初始化：未经初始化的全局静态变量会被程序自动初始化为0（自动对象的值是任意的，除非他被显示初始化）

3）作用域：全局静态变量在声明他的文件之外是不可见的。准确地讲从定义之处开始到文件结尾。

注：static修饰全局变量，并为改变其存储位置及生命周期，而是改变了其作用域，使当前文件外的源文件无法访问该变量，好处如下：（1）不会被其他文件所访问，修改（2）其他文件中可以使用相同名字的变量，不会发生冲突。对全局函数也是有隐藏作用。

对类中的

1.成员变量

用static修饰类的数据成员实际使其成为类的全局变量，会被类的所有对象共享，包括派生类的对象。因此，static成员必须在类外进行初始化(初始化格式： int base::var=10;)，而不能在构造函数内进行初始化，不过也可以用const修饰static数据成员在类内初始化 。

特点：

不要试图在头文件中定义(初始化)静态数据成员。在大多数的情况下，这样做会引起重复定义这样的错误。即使加上#ifndef #define #endif或者#pragma once也不行。  
静态数据成员可以成为成员函数的可选参数，而普通数据成员则不可以。  
静态数据成员的类型可以是所属类的类型，而普通数据成员则不可以。普通数据成员的只能声明为 所属类类型的指针或引用。  
2.成员函数

用static修饰成员函数，使这个类只存在这一份函数，所有对象共享该函数，不含this指针。  
静态成员是可以独立访问的，也就是说，无须创建任何对象实例就可以访问。base::func(5,3);当static成员函数在类外定义时不需要加static修饰符。  
在静态成员函数的实现中不能直接引用类中说明的非静态成员，可以引用类中说明的静态成员。因为静态成员函数不含this指针。

**不可以同时用const和static修饰成员函数。###**

C++编译器在实现const的成员函数的时候为了确保该函数不能修改类的实例的状态，会在函数中添加一个隐式的参数const this\*。但当一个成员为static的时候，该函数是没有this指针的。也就是说此时const的用法和static是冲突的。

我们也可以这样理解：两者的语意是矛盾的。static的作用是表示该函数只作用在类型的静态变量上，与类的实例没有关系；而const的作用是确保函数不能修改类的实例的状态，与类型的静态变量没有关系。因此不能同时用它们。

const的作用：

1.限定变量为不可修改。

2.限定成员函数不可以修改任何数据成员。

3.const与指针：

const char \*p 表示 指向的内容不能改变。

char \* const p，就是将P声明为常指针，它的地址不能改变，是固定的，但是它的内容可以改变。

.tcp为什么要三次握手，tcp为什么可靠。

问题31.为什么不能两次握手：（防止已失效的连接请求又传送到服务器端，因而产生错误）

假设改为两次握手，client端发送的一个连接请求在服务器滞留了，这个连接请求是无效的，client已经是closed的状态了，而服务器认为client想要建立

一个新的连接，于是向client发送确认报文段，而client端是closed状态，无论收到什么报文都会丢弃。而如果是两次握手的话，此时就已经建立连接了。

服务器此时会一直等到client端发来数据，这样就浪费掉很多server端的资源。

（校注：此时因为client没有发起建立连接请求，所以client处于CLOSED状态，接受到任何包都会丢弃，谢希仁举的例子就是这种场景。但是如果服务器发送对这个延误的旧连接报文的确认的同时，客户端调用connect函数发起了连接，就会使客户端进入SYN\_SEND状态，当服务器那个对延误旧连接报文的确认传到客户端时，因为客户端已经处于SYN\_SEND状态，所以就会使客户端进入ESTABLISHED状态，此时服务器端反而丢弃了这个重复的通过connect函数发送的SYN包，见第三个图。而连接建立之后，发送包由于SEQ是以被丢弃的SYN包的序号为准，而服务器接收序号是以那个延误旧连接SYN报文序号为准，导致服务器丢弃后续发送的数据包）  
三次握手的最主要目的是保证连接是双工的，可靠更多的是通过重传机制来保证的。

TCP可靠传输的实现：

TCP 连接的每一端都必须设有两个窗口——一个发送窗口和一个接收窗口。TCP 的可靠传输机制用字节的序号进行控制。TCP 所有的确认都是基于序号而不是基于报文段。  
发送过的数据未收到确认之前必须保留，以便超时重传时使用。发送窗口没收到确认不动，和收到新的确认后前移。

发送缓存用来暂时存放： 发送应用程序传送给发送方 TCP 准备发送的数据；TCP 已发送出但尚未收到确认的数据。

接收缓存用来暂时存放：按序到达的、但尚未被接收应用程序读取的数据； 不按序到达的数据。