**C++中为什么存在this指针？**

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候，那么编译器如何知道具体操作的是哪一个对象呢？在这里就引出了this指针。

C++编译器给每个 “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(函数运行时调用该函数的对象)，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。只不过所有的操作对用户来说都是透明的，不需要用户来传递，编译器自动完成。

this指针特性：

1、只能在“成员函数”中使用

2、this指针类型： 类类型 \*const （加const是为了保证，指针的指向不被更改）

3、this指针不存储在对象中，不影响对象大小，且始终指向当前对象

4、this指针是“成员函数”的第一个隐藏参数，由编译器自动给出

5、主要是通过ecx寄存器来传递（并不是所有的）

## 构造函数可以成为虚函数吗？

不可以。因为在构造函数执行结束后，虚函数表指针才会被正确的初始化。

在c++ 的多态中，虚函数表是由编译器自动生成与维护的，虚函数表指针是由构造函数初始化完成的，即构造函数相当于是虚函数的入口点，负责调用虚函数的前期工作；在构造函数执行的过程中，虚函数表指针有可能未被正确的初始化；由于在不同的c++编译器中，虚函数表 与 虚函数表指针的实现有所不同，所以禁止将构造函数声明为虚函数。

### 虚表指针初始化时机

由于每个对象调用的虚函数都是通过虚表指针来索引的，也就决定了虚表指针的正确初始化是非常重要的。换句话说，在虚表指针没有正确初始化之前，我们不能够去调用虚函数。那么虚表指针在什么时候，或者说在什么地方初始化呢？

答案是在构造函数中进行虚表的创建和虚表指针的初始化。还记得构造函数的调用顺序吗，在构造子类对象时，要先调用父类的构造函数，此时编译器只“看到了”父类，并不知道后面是否后还有继承者，它初始化父类对象的虚表指针，该虚表指针指向父类的虚表。当执行子类的构造函数时，子类对象的虚表指针被初始化，指向自身的虚表。

### 虚函数表存放位置

1、虚函数表是class specific的，也就是针对一个类来说的，这里有点像一个类里面的staic成员变量，即它是属于一个类所有对象的，不是属于某一个对象特有的，是一个类所有对象共有的。

2、虚函数表是编译器来选择实现的，编译器的种类不同，可能实现方式不一样，就像前面我们说的vptr在一个对象的最前面，但是也有其他实现方式，不过目前gcc 和微软的编译器都是将vptr放在对象内存布局的最前面。

3、虽然我们知道vptr指向虚函数表，那么虚函数表具体存放在内存哪个位置呢，虽然这里我们已经可以得到虚函数表的地址。实际上虚函数指针是在构造函数执行时初始化的，而虚函数表是存放在可执行文件中的。下面的一篇博客测试了微软的编译器将虚函数表存放在了目标文件或者可执行文件的常量段中，<http://blog.csdn.net/vicness/article/details/3962767>，不过我在gcc下的汇编文件中没有找到vtbl的具体存放位置，主要是对可执行文件的装载和运行原理还没有深刻的理解，相信不久有了这些知识之后会很轻松的找到虚函数表到底存放在目标文件的哪一个段中。

4、经过测试，在gcc编译器的实现中虚函数表vtable存放在可执行文件的只读数据段.rodata中。

**C++ 多态**

子类跟父类一样有一个虚表指针。

子类的虚函数表一部分继承自父类。如果重写了虚函数，那么子类的虚函数会在虚表上覆盖父类的虚函数。

本质上虚函数表是一个虚函数指针数组，最后一个元素是nullptr，代表虚表的结束。

所以，如果继承了虚函数，那么

1 子类先拷贝一份父类虚表，然后用一个虚表指针指向这个虚表。

2 如果有虚函数重写，那么在子类的虚表上用子类的虚函数覆盖。

3 子类新增的虚函数按其在子类中的声明次序增加到子类虚表的最后。



