

fisher\_jiang的专栏


修身，齐家，治国，平天下

目录视图

摘要视图

RSS 订阅

个人资料



fisher\_jiang

访问：895758次

积分：10205

等级：BLOG 7

排名：第767名

原创：154篇

转载：144篇

译文：0篇

评论：199条

文章搜索

文章分类

C/C++ (52)

Linux (55)

STL (13)

Web编程 (10)

Windows编程 (18)

并行编程 (0)

程序人生 (50)

算法与数据结构 (67)

编程珠玑 (3)

网络安全 (28)

备忘 (0)

BitBlaze (0)

文章存档

2013年05月 (1)

2013年04月 (1)

2013年02月 (2)

2013年01月 (2)

2012年12月 (1)

展开

阅读排行

JS:attachEvent和addEve

(37489)

解决VMware Taking owr

(25543)

解决Error: 'nmake' 不是F

(20424)

想听课？来发话题吧

CSDN APP

博客上线

有奖征文：云服务器使用初体验

有奖试读—增长黑客，创业公司必知的“黑科技”

虚析构造函数(√)、纯虚析构造函数(√)、虚构造函数(X)

分类：C/C++

2008-05-24 15:41

3458人阅读

评论(2)

收藏

举报

delete

编译器

语言

class

gcc

java

一. 虚析构造函数

我们知道，为了能够正确的调用对象的析构造函数，一般要求具有层次结构的顶级类定义其析构造函数为虚函数。因为在delete一个抽象类指针时候，必须要通过虚函数找到真正的析构造函数。

如：

```
class Base
{
public:
    Base(){}
    virtual ~Base(){}
};

class Derived: public Base
{
public:
    Derived(){};
    ~Derived(){};
};

void foo()
{
    Base *pb;
    pb = new Derived;
    delete pb;
}
```

这是正确的用法，会发生动态绑定，它会先调用Derived的析构造函数，然后是Base的析构造函数

如果析构造函数不加virtual，delete pb只会执行Base的析构造函数，而不是真正的Derived析构造函数。因为不是virtual函数，所以调用的函数依赖于指向静态类型，即Base

二. 纯虚析构造函数

现在的问题是，我们想把Base做出抽象类，不能直接构造对象，需要在其中定义一个纯虚函数。如果其中没有其他合适的函数，可以把析构造函数定义为纯虚的，即将前面的CObject定义改成：

```
class Base
{
public:
    Base(){}
    virtual ~Base()= 0
```

http://blog.csdn.net/fisher\_jiang/article/details/2477577

1/4

linux 下 读取某个文件的...	(19709)
开源代码网站	
如何解决"Offending key i...	(19629)
execve的使用方法	(17766)
windbg跟踪NtOpenProce...	(17301)
ubuntu 11.10 安装完更新...	(16279)
Some IoControlCodes fo...	(15572)
	(15522)

评论排行	
开源代码网站	(23)
JS:attachEvent和addEvent	(22)
Miller_Rabin素数测试	(14)
一道经典的面试题：如何	(9)
宏与内联函数(面试常考)	(8)
我的求职经历	(7)
海量数据面试题整理	(6)
并查集 (Union-Find Sets	(6)
解决VMware Taking ownr	(6)
由rand7生成rand10以及!	(5)

推荐文章	
*在R中使用支持向量机（SVM）进行数据挖掘（上）	
* 你不再需要动态网页——编辑-发布-开发分离	
*Android性能优化之使用线程池处理异步任务	
* Nginx初探	
*编译器架构的王者LLVM——（6）多遍翻译的宏翻译系统	
* 我的第一个Apple Watch小游戏——猜数字（Swift）	

最新评论	
用cpuid指令获取cpu信息不是为我: 博主还有这个PDF吗? 能不能发我一份谢谢 83920322@qq.com	
解决VMware Taking ownership (baidu_23208481: 太厉害了。	
JS:attachEvent和addEventListener_1347709817: 不错，刚学js，易懂。	
C++好的面试题和不好的面试题 s876659656: 有点道理	
开源代码网站 springmvc_springdata: 最代码 http://www.zuidaima.com/提供最全面，最专业的代码分享站，近万名用户...	
Linux下定时执行脚本 hello_linux_love: 顶!	
开源代码网站 q125969287: http://www.qbsos.com ---开源代码 这个网站也很不错代码很全	
解决VMware Taking ownership (hhn0625: 谢谢啦，已经解决了	
最长单调递增子序列( O(nlgn) ) 昊iwi_ac: .....	
位运算在算法编程中的使用技巧 岁月小龙: 很有价值	

};

可是，这段代码不能通过编译，通常是link错误，不能找到~Base()的引用(gcc的错误报告)。这是因为，析构函数、构造函数和其他内部函数不一样，在调用时，编译器需要产生一个调用链。也就是，Derived的析构函数里面隐含调用了Base的析构函数。而刚才的代码中，缺少~Base()的函数体，当然会出现错误。

这里面有一个误区，有人认为，virtual f()=0这种纯虚函数语法就是没有定义体的语义。

其实，这是不对的。这种语法只是表明这个函数是一个纯虚函数，因此这个类变成了抽象类，不能产生对象。我们完全可以为纯虚函数指定函数体。通常的纯虚函数不需要函数体，是因为我们一般不会调用抽象类的这个函数，只会调用派生类的对应函数。这样，我们就有了一个纯虚析构函数的函数体，上面的代码需要改成：

```
class Base
{
public:
    Base()
{
}
    virtual ~Base() = 0; //pure virtual
};

Base::~Base()//function body
{
}
```

从语法角度来说，不可以将上面的析构函数直接写入类声明中（内联函数的写法）。这或许是一个不正变化的地方。但是这样做的确显得有点累赘

这个问题看起来有些学术化，因为一般我们完全可以在Base中找到一个更加适合的函数，通过将其定义为没有实现体的纯虚函数，而将整个类定义为抽象类。但这种技术也有一些应用，如这个例子：

```
class Base //abstract class
{
public:
    virtual ~Base(){};//virtual, not pure
    virtual void Hiberarchy() const = 0;//pure virtual
};

void Base::Hiberarchy() const //pure virtual also can have function body
{
    std::cout <<"Base::Hiberarchy";
}

class Derived : public Base
{
public:
    Derived(){}
    virtual void Hiberarchy() const
    {
        CB::Hiberarchy();
        std::cout <<"Derived::Hiberarchy";
    }
    virtual void foo(){}
};

int main(){
    Base* pb=new Derived();
    pb->Hiberarchy();
    pb->Base::Hiberarchy();
    return 0;
```

链接

zik的Blog (同寝室的哥们)  
bevin 的 Blog (死党)  
Csdn 技术中心  
rover的Blog  
会飞的鱼  
Simonjo的blog (睡在我上铺的兄弟)  
我手写我心(韦煜-朗讯的兄弟)(RSS)  
琢思碰文轩(北软师兄的blog)  
照妖镜--中学时代的才女  
北斗星君 -- Dev-Cpp/Mingw32/GCC专栏  
小强的BLOG  
程序员面试题精选  
kevinsID的天空--安全大牛  
信息安全专栏 -- 褚诚云

```
    }  
}
```

在这个例子中，我们试图打印出类的继承关系。在根基类中定义了虚函数Hiberarchy，然后在每个派生类中都重载此函数。我们再一次看到，由于想把Base做成个抽象类，而这个类中没有其他合适的方法成员可以定义为纯虚的，我们还是只好将Hiberarchy定义为纯虚的。（当然，完全可以定义~Base函数，这就和上面的讨论一样了。^\_^）

另外，可以看到，在main中有两种调用方法，第一种是普通的方式，进行动态链接，执行虚函数，得到结果"Derived::Hiberarchy"；第二种是指定类的方式，就不再执行虚函数的动态链接过程了，结果是"Base::Hiberarchy"。

通过上面的分析可以看出，**定义纯虚函数的真正目的是为了定义抽象类**，而并不是函数本身。与之对比，在java中，定义抽象类的语法是 abstract class，也就是在类的一级作指定（当然虚函数还是要加上abstract关键字）。是不是这种方式更好一些呢？在Stroustrup的《C++语言的设计与演化》中找到这样一段话：

“我选择的是将个别的函数描述为纯虚的方式，没有采用将完整的类声明定义为抽象的形式，这是因为纯虚函数的概念更加灵活一些。我很看重能够分阶段定义类的能力；也就是说，我发现预先定义一些纯虚函数，并把另外一些留给进一步的派生类去定义也是很有用的”。

我还没有完全理解后一句话，我想从另外一个角度来阐述这个概念。那就是，在一个多层复杂类结构中，中间层次的类应该具体化一些抽象功能并不是所有的。中间类没必要知道是否具体化了所有的虚函数，以及其祖先已经具体化了哪些函数——自己的职责就可以了。也就是说，中间类没必要知道自己是否是一个真正的抽象类，设计者也就不需要考虑是否需要在这个中间类的类级别上加上类似abstract的说明了。

当然，一个语言的设计有多种因素，好坏都是各个方面的。这只是一个解释而已。

最后，总结一下关于虚函数的一些常见问题：

- 1) 虚函数是动态绑定的，也就是说，使用虚函数的指针和引用能够正确找到实际类的对应函数，而不是执行定义类的函数。这是虚函数的基本功能，就不再解释了。
- 2) 构造函数不能是虚函数。而且，在构造函数中调用虚函数，实际执行的是父类的对应函数，因为自己还没有构造好，多态是被disable的。
- 3) 析构函数可以是虚函数，而且，在一个复杂类结构中，这往往是必须的。
- 4) 将一个函数定义为纯虚函数，实际上是将这个类定义为抽象类，不能实例化对象。
- 5) 纯虚函数通常没有定义体，但也完全可以拥有。
- 6) 析构函数可以是纯虚的，但纯虚析构函数必须有定义体，因为析构函数的调用是在子类中隐含的。
- 7) 非纯的虚函数必须有定义体，不然是一个错误。
- 8) 派生类的override虚函数定义必须和父类完全一致。除了一个特例，如果父类中返回值是一个指针或引用，子类override时可以返回这个指针（或引用）的派生。例如，在上面的例子中，在Base中定义了 virtual Base\* clone(); 在Derived中可以定义为 virtual Derived\* clone()。可以看到，这种放松对于Clone模式是非常有用的。其他，有待补充。

上一篇 [vector中的erase方法跟algorithm的remove有什么区别？](#)

下一篇 [C++中禁止异常信息传递到析构函数外](#)