

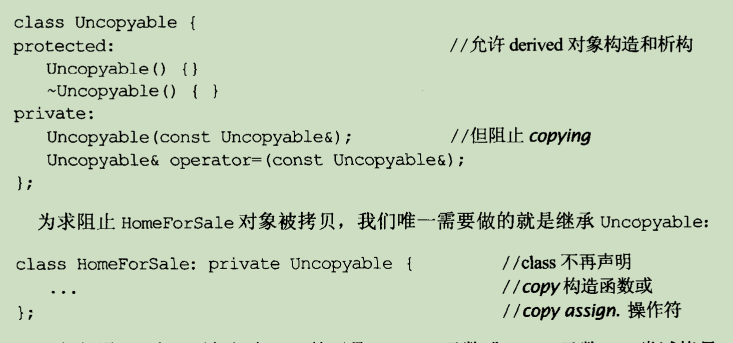
**条款5**

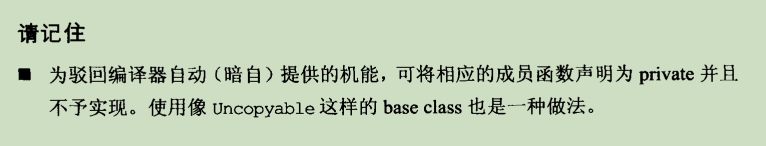


如果我们内部成员存在const ，所以调用copy的操作符存在一定问题、

**条款6**

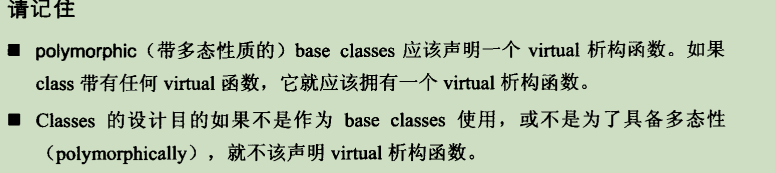
如果我们不像让类有拷贝功能，如果我们不实现那么编译器会给我们实现一个默认拷贝函数，所以无法达到不让拷贝的目的，我们需要拒绝拷贝，可以通过将拷贝函数写在private中间不给实现，缺点就是friend函数和内部函数依据可以调用。如果我们这么做了，可以作用于pubilc继承的子类。





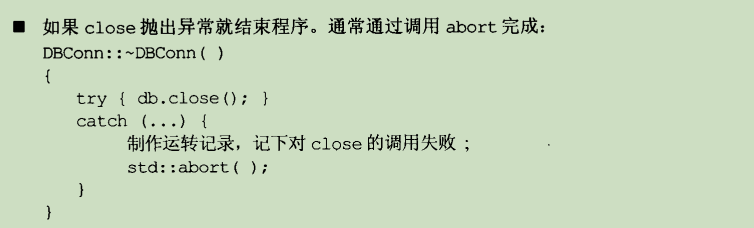
**条款7**

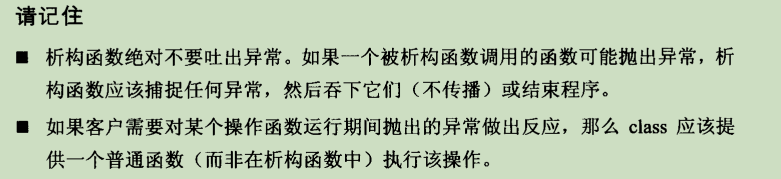
如果我们写的base class 的析构函数没有将析构函数virtual ，那么得出派生类的有可能不会实现自己的析构函数，我们利用c++多态特写，他通过base class 指针处理派生class对象，我们想要通过指针释放对象，会导致释放不充分，内存泄漏。



**条款8**

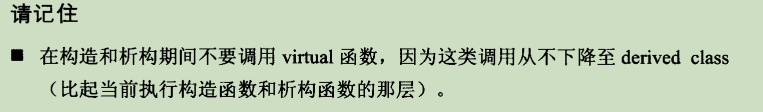
如果析构函数书写中。存在一个可能出现的问题，有可能导致析构函数释放不了，内存泄漏，所以我们可以加上一个不太好的无奈之举，try吞下他。



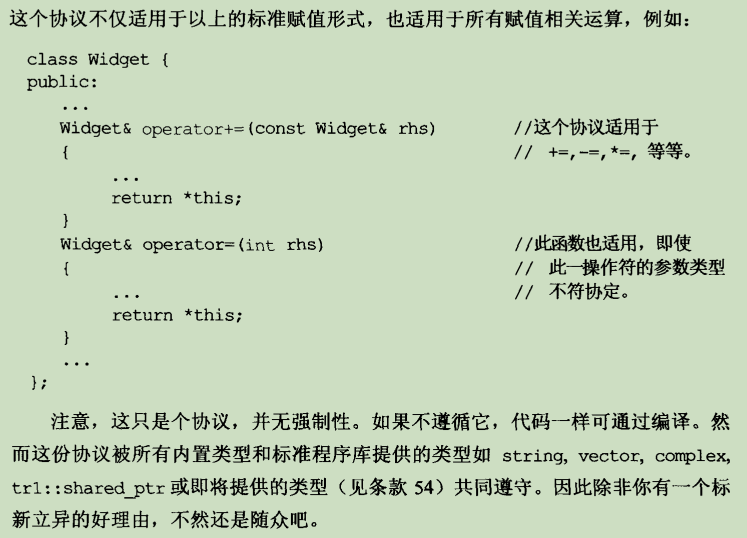


**条款9**

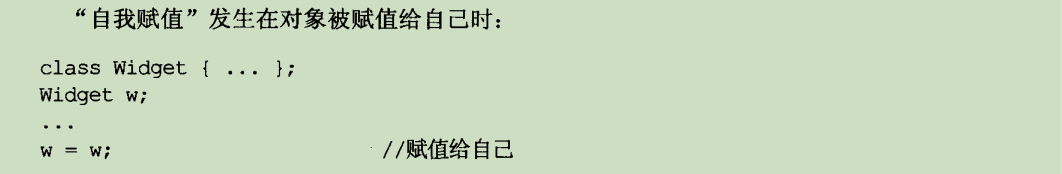
derived class 构造前先调用base class的构造函数，如果在base class 中调用一个virtual 函数，那么就会调用的是base class 版本的（base class 先构造），这个肯定是不符合我们的目的，因为此时我们创建的是一个derived class对象，



**条款10**

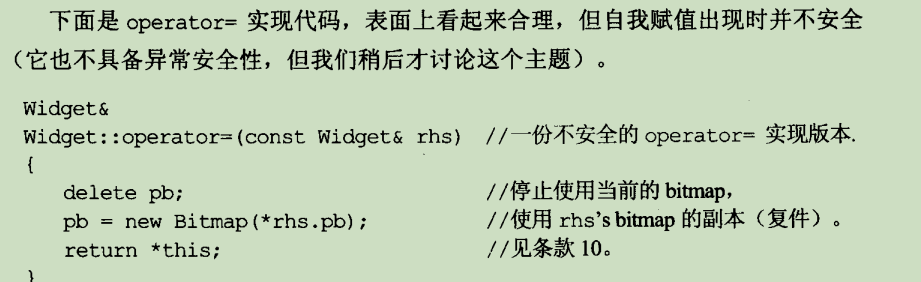


**条款11**

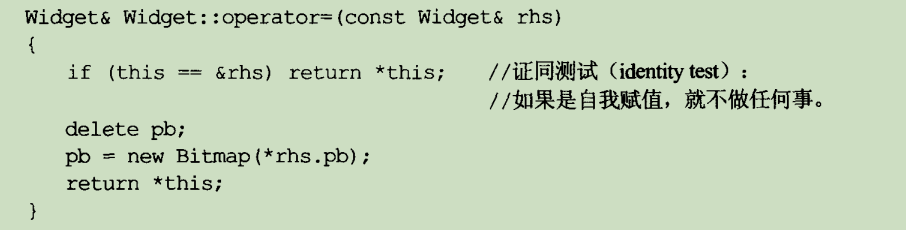


看起来蠢蠢的但是他是合法的。

这样写法看起来合理但是，存在严重漏洞，如果rhs和\*this指向同一个对象，delete操作把\*this的pb删除，同样把rhs的pb删除，下面一行的new pb将会失败，pb指向一个不知道的地方，pb成了一个谁也伤不到的指针。



**解决方法**



**条款12**

一个deriv类中调用构造函数，编译器会自觉调用base类中的构造函数。

对于拷贝函数 operator&=（自己在derive类中编写的，拒绝编译提供的）需要注意两点。

1.拷贝需要把derive类中所有对象都进行拷贝。

2.base类中继承的对象，编译器不会帮你调用base类的拷贝函数，需要你自己调用。

3.

