如何保证只能在堆(heap)或栈(stack)上创建对象

1. **class** OnlyHeapClass
2. {
3. **public**:
4. OnlyHeapClass()
5. {
6. }
8. **void** Destroy()
9. {
10. **delete** **this**; // 等效于"OnlyHeapClass::~OnlyHeapClass();", 写
11. // 成"OnlyHeapClass::~OnlyHeapClass();"更容易理
12. // 解,public成员函数调用private析构函数.
13. }
15. **private**:
16. ~OnlyHeapClass()  //析构函数为私有
17. {
18. }
19. };
21. **int** main()
22. {
23. OnlyHeapClass \*pInst = **new** OnlyHeapClass;
25. pInst ->Destroy();
27. **return** 0;
28. }

总结: 把析构函数定义为private访问权限, 就可以保证只能在堆(heap)上创建(new)一个新的类对象.

原因是C++是一个静态绑定的语言. 在编译过程中, 所有的非虚函数调用都必须分析完成. 即使是虚函数, 也需检查可访问性. 因些, **当在栈(stack)上生成对象时, 对象会在函数调用结束后自动析构, 也就说析构函数必须可以访问**. 而堆上生成对象, 由于析构时机由程序员控制, 所以不一定需要析构函数. 保证了不能在栈上生成对象后, 需要证明能在堆上生成它. 这里OnlyHeapClass与一般对象唯一的区别在于它的析构函数为私有, delete操作会调用析构函数, 所以不能通过编译.

那么如何释放它呢? 答案也很简单, 提供一个公有的成员函数, 完成delete操作. 在成员函数中, 私有的析构函数是可以访问的, 当然detele操作也是可以编译通过.

1. **void** OnlyHeapClass::Destroy()
2. {
3. **delete** **this**;    //可以访问私有的析构函数
4. }

析构函数私有化的类的设计可以保证只能用new在堆(heap)中创建对象, 只能动态的去创建对象, 这样可以自由的控制对象的生命周期. 但是, 这样的类需要提供创建和撤销的公共接口.。

另外重载operator new, operator delete为private，可以达到限制对象只能创建于栈上的目的, 用placement new也可以创建在栈上。