**[c++ 单例模式下的实例自动销毁(单例自动回收器)](http://blog.csdn.net/roy1261/article/details/51425987)**

标签： [单例模式](http://www.csdn.net/tag/%e5%8d%95%e4%be%8b%e6%a8%a1%e5%bc%8f)[自动释放](http://www.csdn.net/tag/%e8%87%aa%e5%8a%a8%e9%87%8a%e6%94%be)[c++](http://www.csdn.net/tag/c%2b%2b)[singleton](http://www.csdn.net/tag/singleton)

2016-05-16 16:17 763人阅读 [评论](http://blog.csdn.net/roy1261/article/details/51425987#comments)(2) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/roy1261/article/details/51425987#report)

本文章已收录于：

版权声明：本文为博主原创文章，未经博主允许不得转载。

前些日子看到一篇博文，关于C++单例模式下m\_pinstance指向空间销毁问题，m\_pInstance的手动销毁经常是一个头痛的问题，内存和资源泄露也是屡见不鲜，  
能否有一个方法，让实例自动释放。网上已经有解决方案(但是具体实现上表述不足，主要体现在自动析构未能正常运行)，那就是定义一个内部垃圾回收类，并且在Singleton中定义一个此类的静态成员。程序结束时，系统会自动析构此静态成员，此时，在此类的析构函数中析构Singleton实例，就可以实现m\_pInstance的自动释放。

附上[**测试**](http://lib.csdn.net/base/softwaretest)代码

**[cpp]** [view plain](http://blog.csdn.net/roy1261/article/details/51425987) [copy](http://blog.csdn.net/roy1261/article/details/51425987) [print?](http://blog.csdn.net/roy1261/article/details/51425987)

1. #include <iostream>
2. using namespace std;
4. class Singleton
5. {
6. public:
7. static Singleton \*GetInstance()
8. {
9. if (m\_Instance == NULL)
10. {
11. m\_Instance = new Singleton();
12. cout<<"get Singleton instance success"<<endl;
13. }
14. return m\_Instance;
15. }
17. private:
18. Singleton(){cout<<"Singleton construction"<<endl;}
19. static Singleton \*m\_Instance;
21. // This is important
22. class GC // 垃圾回收类
23. {
24. public:
25. GC()
26. {
27. cout<<"GC construction"<<endl;
28. }
29. ~GC()
30. {
31. cout<<"GC destruction"<<endl;
32. // We can destory all the resouce here, eg:db connector, file handle and so on
33. if (m\_Instance != NULL)
34. {
35. delete m\_Instance;
36. m\_Instance = NULL;
37. cout<<"Singleton destruction"<<endl;
38. system("pause");//不暂停程序会自动退出，看不清输出信息
39. }
40. }
41. };
42. static GC gc;  //垃圾回收类的静态成员
44. };
46. Singleton \*Singleton::m\_Instance = NULL;
47. Singleton::GC Singleton::gc; //类的静态成员需要类外部初始化，这一点很重要，否则程序运行连GC的构造都不会进入，何谈自动析构
48. **int** main(**int** argc, **char** \*argv[])
49. {
50. Singleton \*singletonObj = Singleton::GetInstance();
51. return 0;
52. } #include <iostream>using namespace std;

class Singleton

{

public:

static Singleton \*GetInstance()

{

if (m\_Instance == NULL)

{

m\_Instance = new Singleton();

cout<<"get Singleton instance success"<<endl;

}

return m\_Instance;

}

private:

Singleton(){cout<<"Singleton construction"<<endl;}

static Singleton \*m\_Instance;

// This is important

class GC // 垃圾回收类

{

public:

GC()

{

cout<<"GC construction"<<endl;

}

~GC()

{

cout<<"GC destruction"<<endl;

// We can destory all the resouce here, eg:db connector, file handle and so on

if (m\_Instance != NULL)

{

delete m\_Instance;

m\_Instance = NULL;

cout<<"Singleton destruction"<<endl;

system("pause");//不暂停程序会自动退出，看不清输出信息

}

}

};

static GC gc; //垃圾回收类的静态成员

};

Singleton \*Singleton::m\_Instance = NULL;

Singleton::GC Singleton::gc; //类的静态成员需要类外部初始化，这一点很重要，否则程序运行连GC的构造都不会进入，何谈自动析构

int main(int argc, char \*argv[])

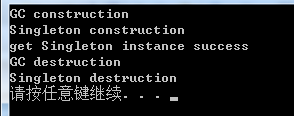
{

Singleton \*singletonObj = Singleton::GetInstance();

return 0;

}

运行结果：



参考文档

http://blog.csdn[**.NET**](http://lib.csdn.net/base/dotnet)/hackbuteer1/article/details/7460019