链滴

首次抓到编译器 Bug

作者：localvar

原文链接：https://ld246.com/article/1522377523500

来源网站：链滴

写代码这么多年，多次怀疑碰到了编译器Bug，但最终却总是自己的责任，这次终于抓到了一个真正编译器Bug，记录一下。

最近公司的一个项目的编译环境升级到了 VS2013，完成后发现注册调试版本的一个 DLL (foo.dll)时发了下面的断言：

```
ATLASSERT(pComModule->m_hInstTypeLib != NULL)
```

这里，pComModule指向的 _AtlComModule定义如下：

```
#pragma managed(push, off)
__declspec(selectany) CAtlComModule _AtlComModule;
#pragma managed(pop)
```

在ATL源码中，可以看到CAtlComModule的构造函数对m_hInstTypeLib进行了赋值。但奇怪的是，试发现这个构造函数根本没被调用过，所以m_hInstTypeLib一直是NULL，最终触发了上面的断言。

经过调查，发现了以下情况：

1. foo.dll是一个既有原生代码和又有托管代码的混合型项目，它的大多数.cpp文件是原生代码，但有个文件是托管代码。因为_AtlComModule定义在atlbase.h中，而后者被stdafx.h包含了，所以，每个编译后的.obj文件里都会有它的一个实例。但因为定义它时使用了__declspec(selectany)，所以链器会只留下一个并把其他都丢掉。

2. 对于动态链接库中的全局变量，编译器会自动生成代码，保证它们的初始化在调用 DllMain之前成（这也就保证了在调用DllRegisterServer时它们已经完成初始化），但这一点仅限于原生代码。对托管代码，为避免可能发生的死锁（参见这篇文档），编译器在调用DllMain之前，不会调用任何托代码，所以，托管代码中的全局变量无法在调用DllMain之前初始化，它们的初始化是在首次调用托代码之前完成的。

3. COM Dll 注册过程是先加载 Dll 文件（调用 DllMain），然后调用DllRegisterServer完成注册默认情况下，这个过程不涉及任何托管代码的调用。

把以上几点综合在一起，问题就清楚了：如果链接器选择了托管代码中的_AtlComModule，注册就失败；反之，如果选择了原生代码中的实例，注册就可以成功（定义_AtlComModule时的managed ff指令在这里并没有什么作用，因为它只能保证生成原生代码来调用构造函数，但只要编译选项包含了clr，这段代码本身还是需要从托管代码中调用）。但问题是，链接器的选择毫无规律可循，所以，这锅只能它背了。

目前，这个问题只出现在程序的Debug版本中，但并没有证据显示它不会出现在Release版本中。而，除了_AtlComModule，ATL中还有很多使用了__declspec(selectany)的全局变量，它们也可能引类似的问题。

作为临时的解决方案，可以使用下面的方法来保证所有全局变量都能在被使用前完成初始化。

1. 在任意一个被编译为托管代码的 .cpp文件中，添加下面的函数：

```
// DO NOT DELETE THIS FUNCTION!!!
// This function is to ensure CLR is loaded and all global variables are initialized
// before calling: DllRegisterServer / DllUnregisterServer / DllGetClassObject
// If it is removed, calling to above functions may fail
void ManagedEnsureClrLoaded()
{
    // calling to GetTickCount() is only to prevent compiler optimization from removing
    // this function, you're free to do other things as your wish
```

```
    ::GetTickCount();
}
```

2. 仿照下面的方式修改 module class 的定义（这里以 CFooModule 为例）。需要注意的是，如果Dll还有其它导出的函数，也需要在它里面调用一下NativeEnsureClrLoaded。

```
// Add this function
void NativeEnsureClrLoaded()
{
    void ManagedEnsureClrLoaded();
    static bool loaded = false;
    if( !loaded ) // no need locks, run twice does not cause a logical error
    {
        ManagedEnsureClrLoaded();
        loaded = true;
    }
}

[module(dll, uuid = "{E6915FF1-AAAA-CCCC-BBBB-E4AEFB2C67CB}",name = "Foo",
helpstring = "Foo 1.0 Type Library",resource_name = "IDR_Foo")]
class CFooModule
{
public:
    // Override CAtlDllModuleT members
    HRESULT DllRegisterServer( _In_ BOOL bRegTypeLib = TRUE ) throw()
    {
        NativeEnsureClrLoaded(); // call NativeEnsureClrLoaded
        return __super::DllRegisterServer( bRegTypeLib );
    }

    HRESULT DllUnregisterServer( _In_ BOOL bUnRegTypeLib = TRUE ) throw()
    {
        NativeEnsureClrLoaded(); // call NativeEnsureClrLoaded
        return __super::DllUnregisterServer( bUnRegTypeLib );
    }

    HRESULT DllGetClassObject( _In_ REFCLSID rclsid, _In_ REFIID riid, _COM_Outptr_ LPVOID*
pv ) throw()
    {
        NativeEnsureClrLoaded(); // call NativeEnsureClrLoaded
        return __super::DllGetClassObject( rclsid, riid, ppv );
    }
};
```

最后，吐槽一下微软的技术支持，把问题报给他们之后各种拖延，基本没有主动的状态更新，这个问的调试他们也一点忙没帮上。找到原因后，发了好几封邮件才终于确认是编译器（链接器）的Bug，表示不会在VS2013上修复了，不知道VS2015有没有指望。