AOP Castle PostSharp

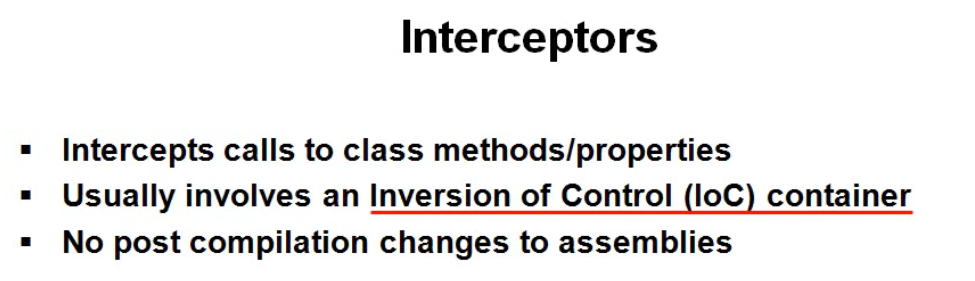
# 作用

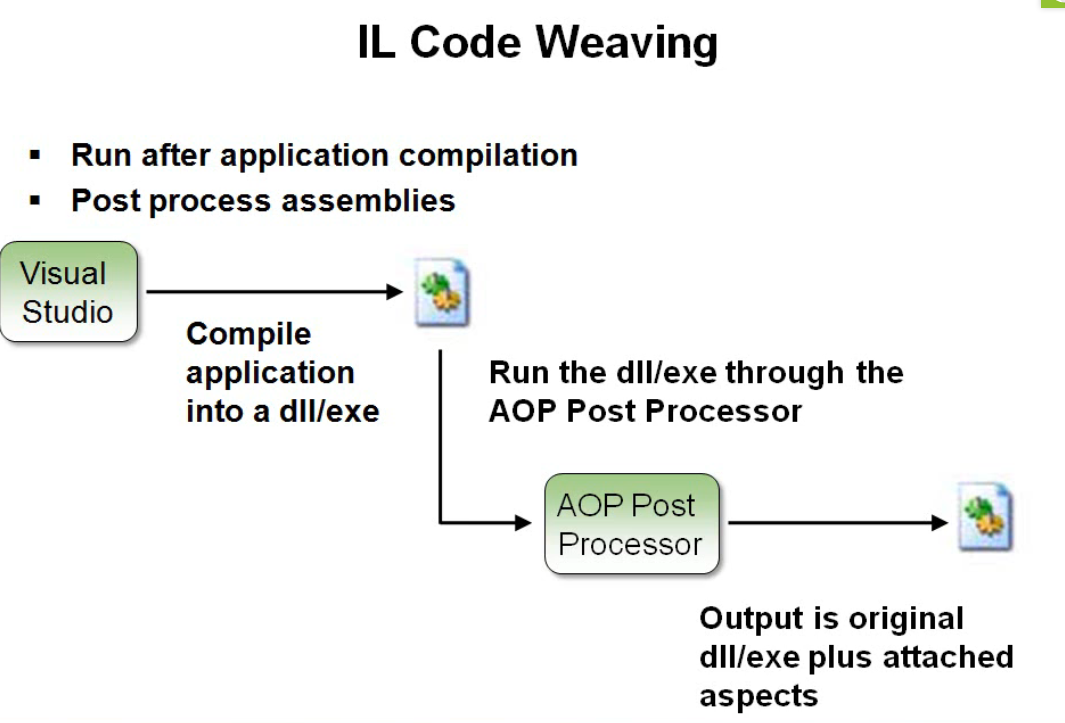
http://www.pluralsight.com/courses/aop

AOP可以改善代码结构，作用是在各种函数被调用前后插入一段代码，通常有OnStart OnSuccess OnError

**本质是自动创建Decorator**

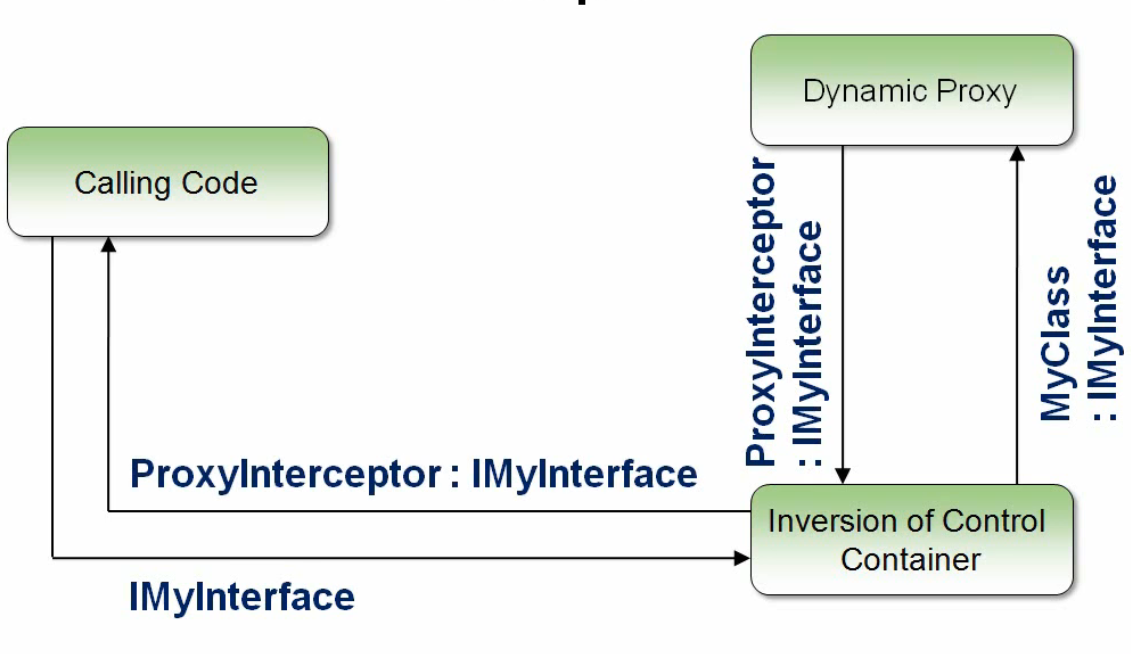
# 分类



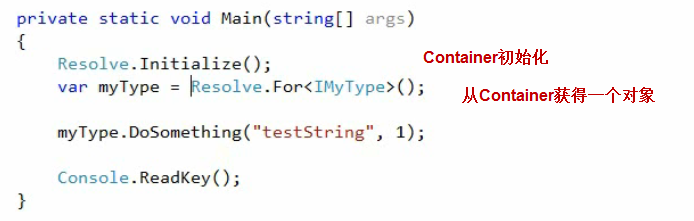


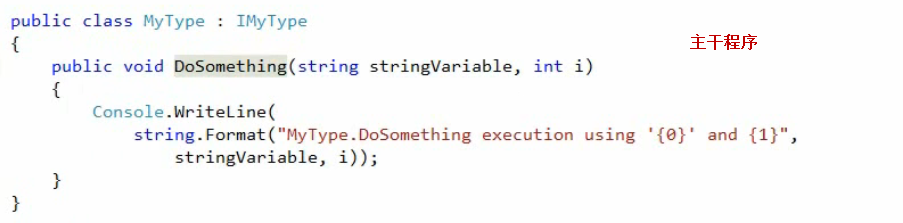
# Interceptors

Calling Code里是原有的主干代码，被插入的代码写在DynamicProxy中。Interceptor的原理就是Calling Code从IoCContainer中去获取一个MyInterface的对象而不是自己new，于是IoCContainer去做了一个新的对象ProxyInterceptor，它即包含了MyCalss的原有功能又包含了Dynamic Proxy，在各种事件时调用Dynamic Proxy中的插入代码，然后调用MyCalss中的主干代码。最后IoC Container把ProxyInterceptor当MyInterface传回给Calling Code去用。

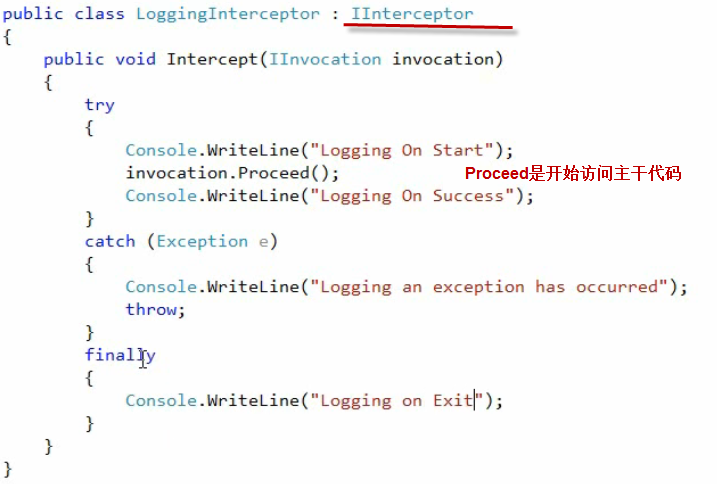


### Demo：用Castle做IoC Container





### 创建Logging Interceptor

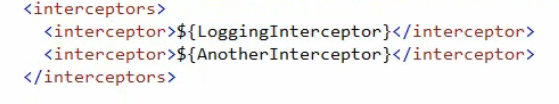


### Attach Logging Interceptor到主干代码

**方法一是Castle的xml config文件**

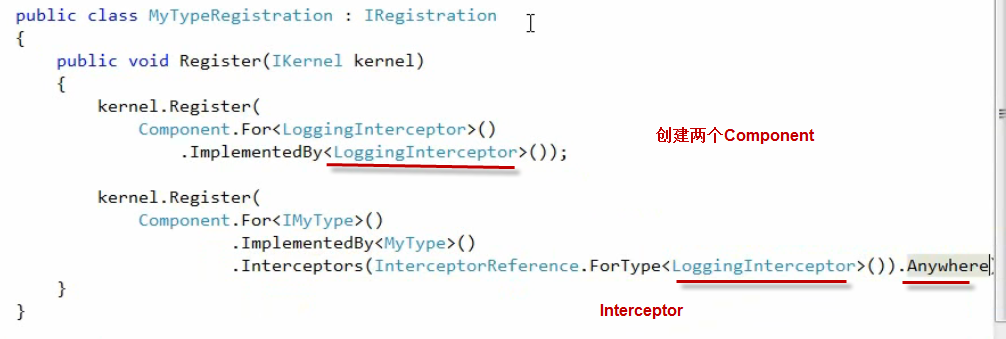


多个就加进去



**方法二：Fluent Interface Registration**

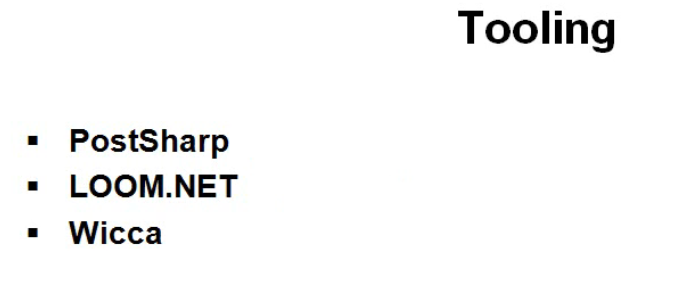
创建一个Class就可以了



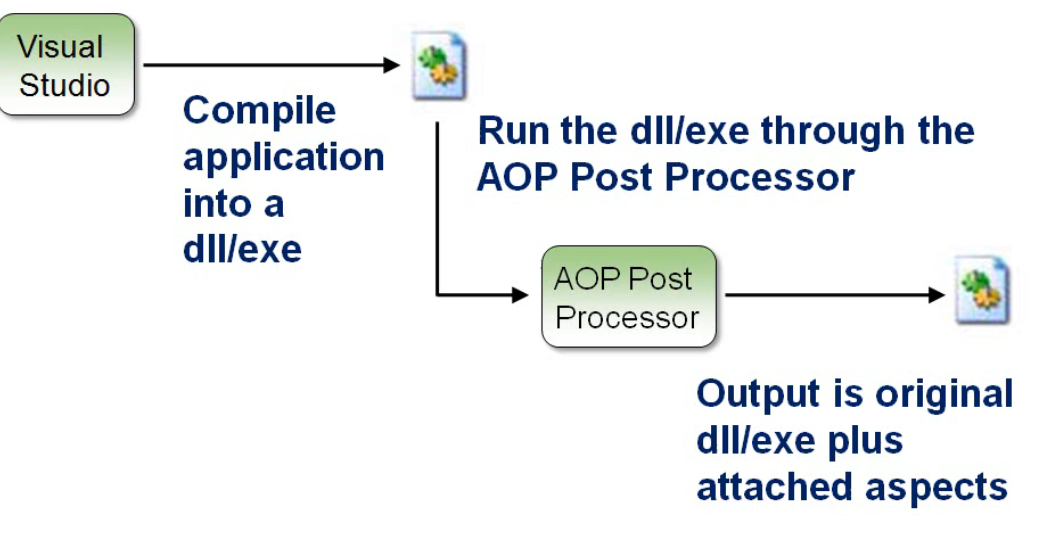
# IL Code Weaving—PostSharp

http://www.postsharp.net/documentation

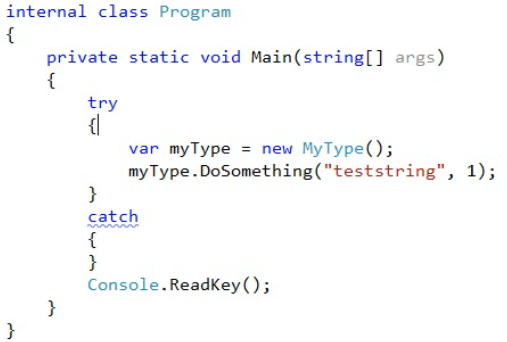
编译后执行，修改IL Code

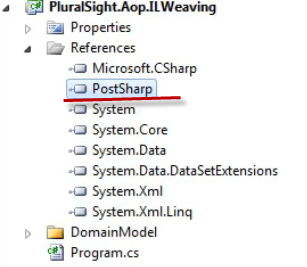


首先由VS编译完成产生dll，然后由AOP Post Processor用这个dll加入代码，产生新的dll



### 主干代码



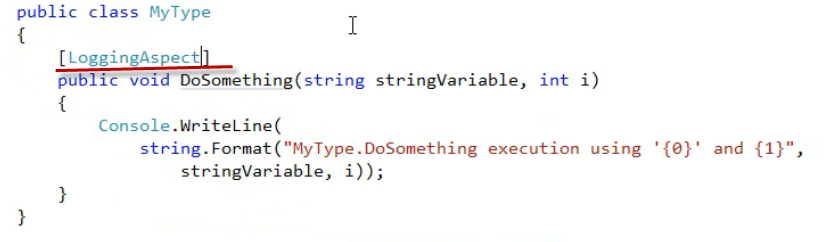


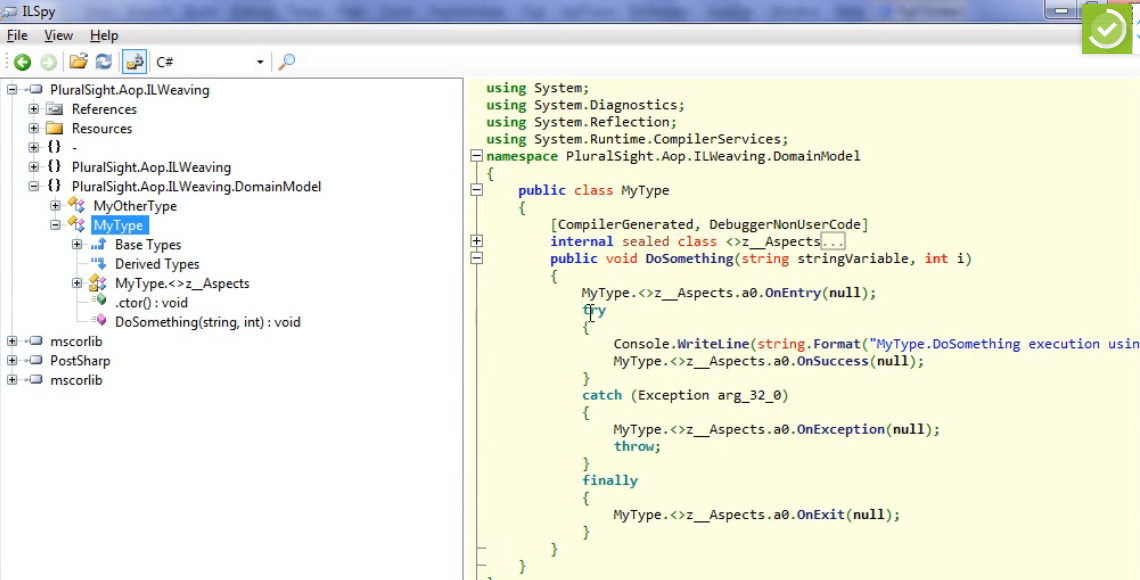
### Logging Aspect



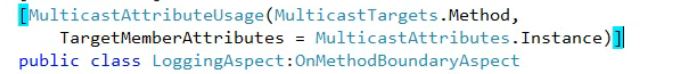
### Attach Aspect

只能通过ILSpy来看编译后的dll里有没有Aspect的代码来判断Weave是否成功





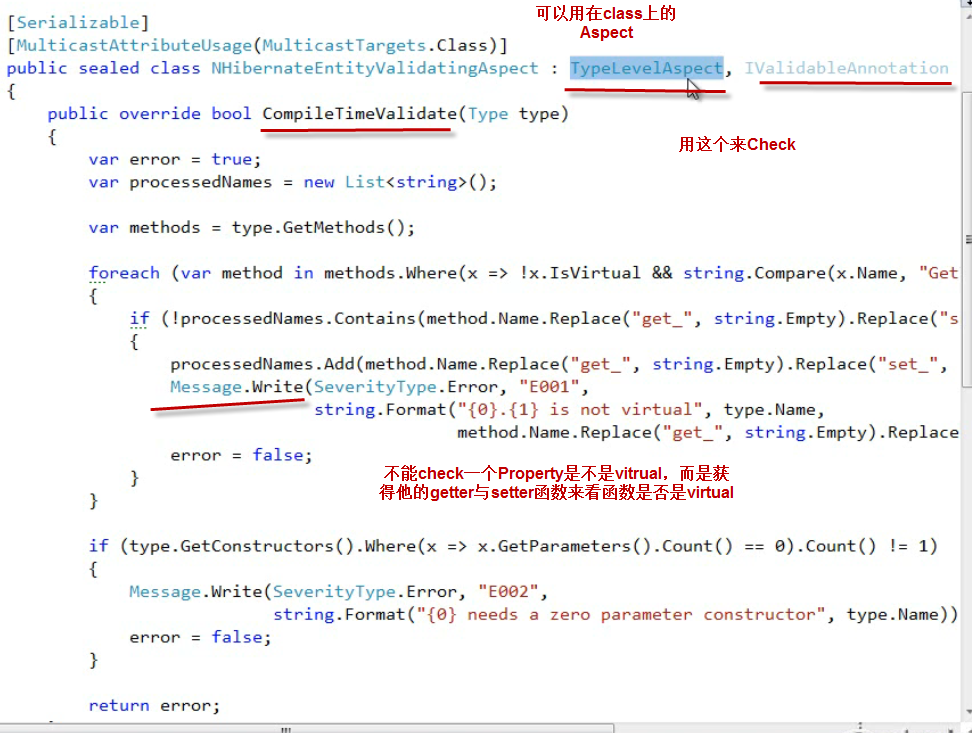
### 限制Attach

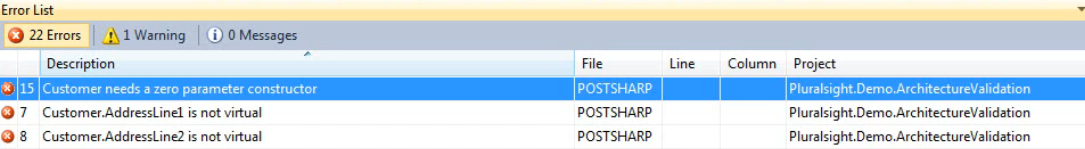


这样这个Aspect只能用于成员函数了

### Demo—编译期Validation

现在有个Class，所有的属性和函数必须是virtual，不然的话编译后要报错





### Demo—Code Generation

