

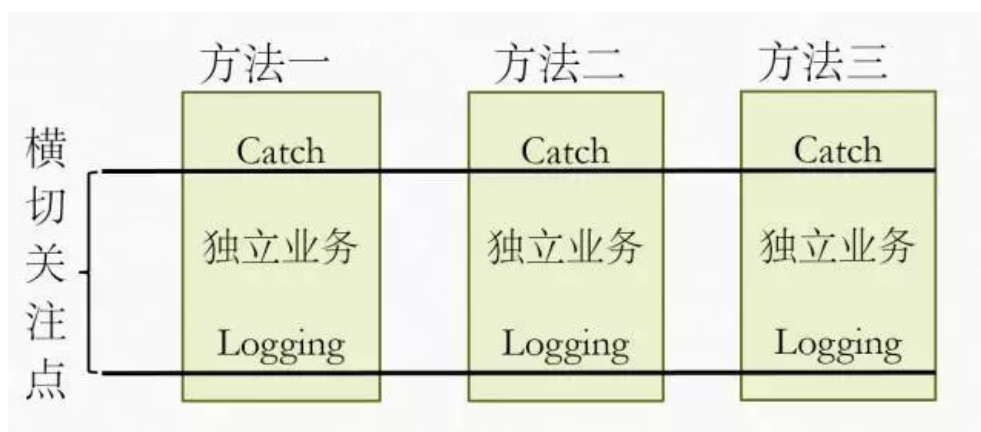
# .NET切面编程—PostSharp

DotNet

## 概念

**Aspect-Oriented Programming ( AOP )**：想想OOP是不是有些熟悉，AOP翻译过来的意思就是面向切面编程。先来关注一下涉及到的以下几个概念点。

**横切关注点**：存在于项目的绝大多数业务中可以通用的一些辅助性的功能。例如日志、安全、持久化等模块。它们存在于核心业务代码块的各个地方，却又独立于这些核心业务逻辑。



**切面**：这些横切关注点的统一抽象。

**所以面向切面编程**，就是将项目的辅助性功能（如日志、异常处理、缓存处理等）与业务逻辑进行分离，把繁琐的辅助性代码抽离出来不用重复Copy，使得程序具备更高的模块化。

## 实现方式

### 静态织入

即编译时织入，实现原理是对编译器做扩展，使得在代码编译时编译器将切面代码织入到指定的切点。

### 动态织入

即运行时织入，编译器在编译时对切面代码和业务代码分别独立编译，而在运行的时候由CLR进行代码混合。

## .NET平台的切面实现——PostSharp

### 为什么选用PostSharp

- 轻量级的静态织入实现（可以通过反编译清晰的知道你的代码构成）

- 使用简单，独立编写切面类，更好的实现模块化，继承自PostSharp提供的各种切面类型的抽象类，并重写其中的拦截方法即可，可以像使用类库内置的Attribute那样使用AOP
- 对调用方法有更多的控制点，比如输入参数、返回结果、异常捕获等
- 但是，但是自从2.0版本之后就不免费了

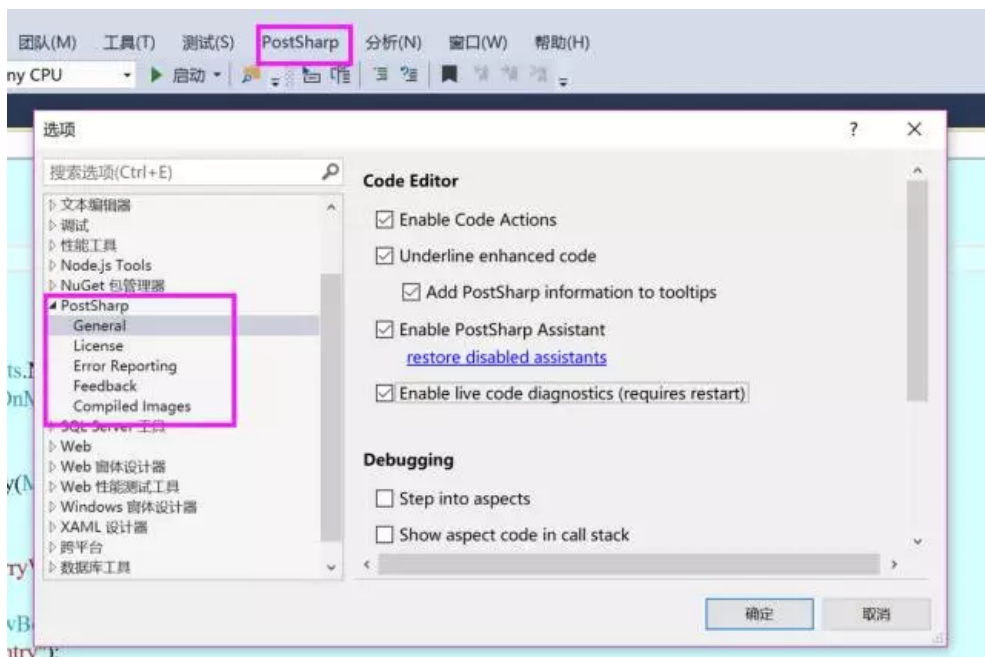
## PostSharp的切面类

- OnMethodBoundaryAspect，针对方法内的各种可能存在切点的情况进行代码注入，实现切面思想，提供了OnEntry,OnExit,OnSuccess,OnException等可重写的虚方法，顾名思义分别是在进入方法、退出方法、方法体成功执行、方法内发生异常的拦截。后续的实例是通过OnEntry来修改方法的输入参数来展示的。
- OnFiledAccessAspect，对Filed的读写进行拦截处理，提供了OnGetValue,OnSetValue的虚方法。
- OnExceptionAspect，实现异常的捕获。
- OnMethodInvocationAspect，方法调用拦截，提供OnInvocation虚方法。
- ImplementMethodAspect，用于extern方法、abstract类的方法进行拦截。

## PostSharp的版本差异

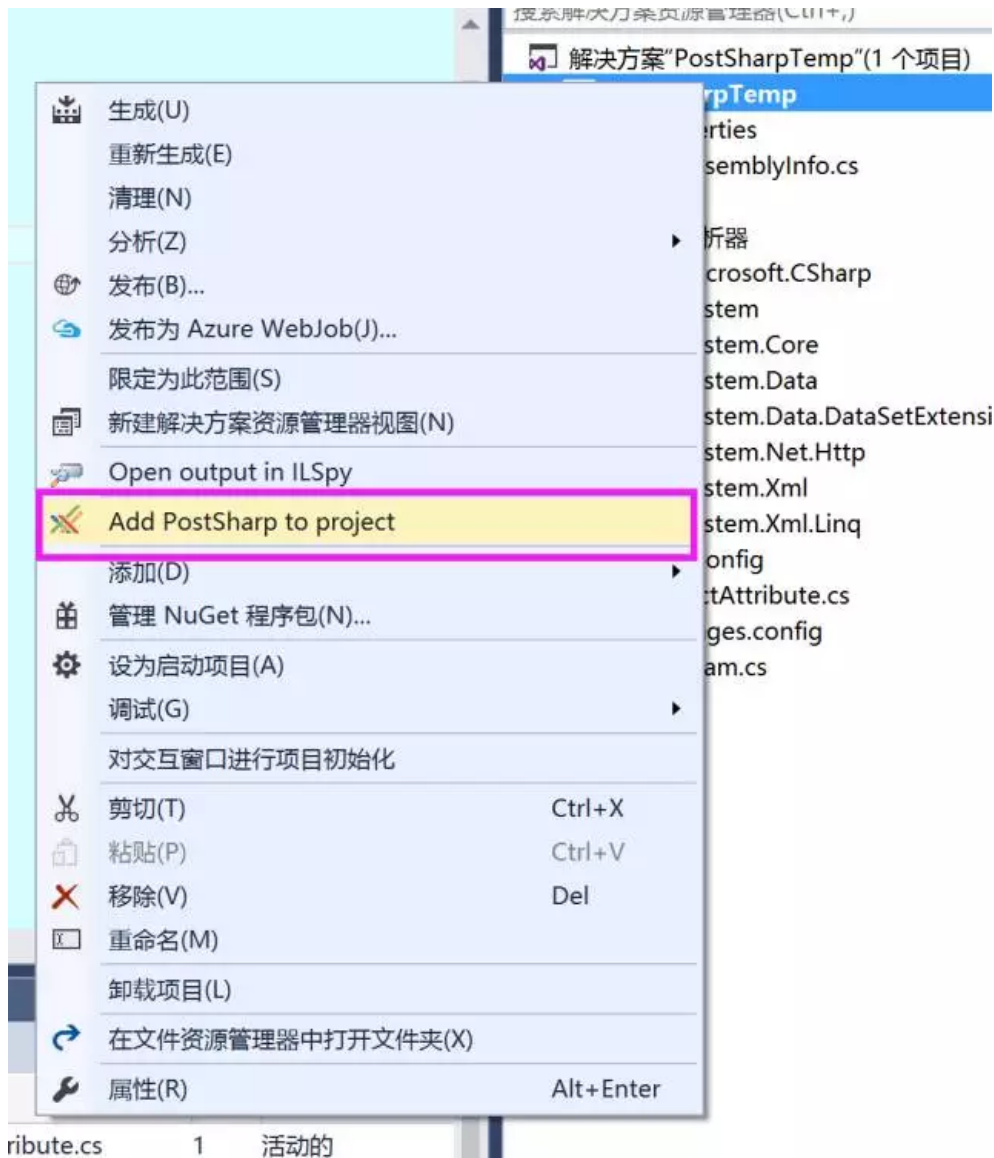
3.0版本是个分水岭，目前最新版本是5.0.28，他们的使用方式有一小点差异。

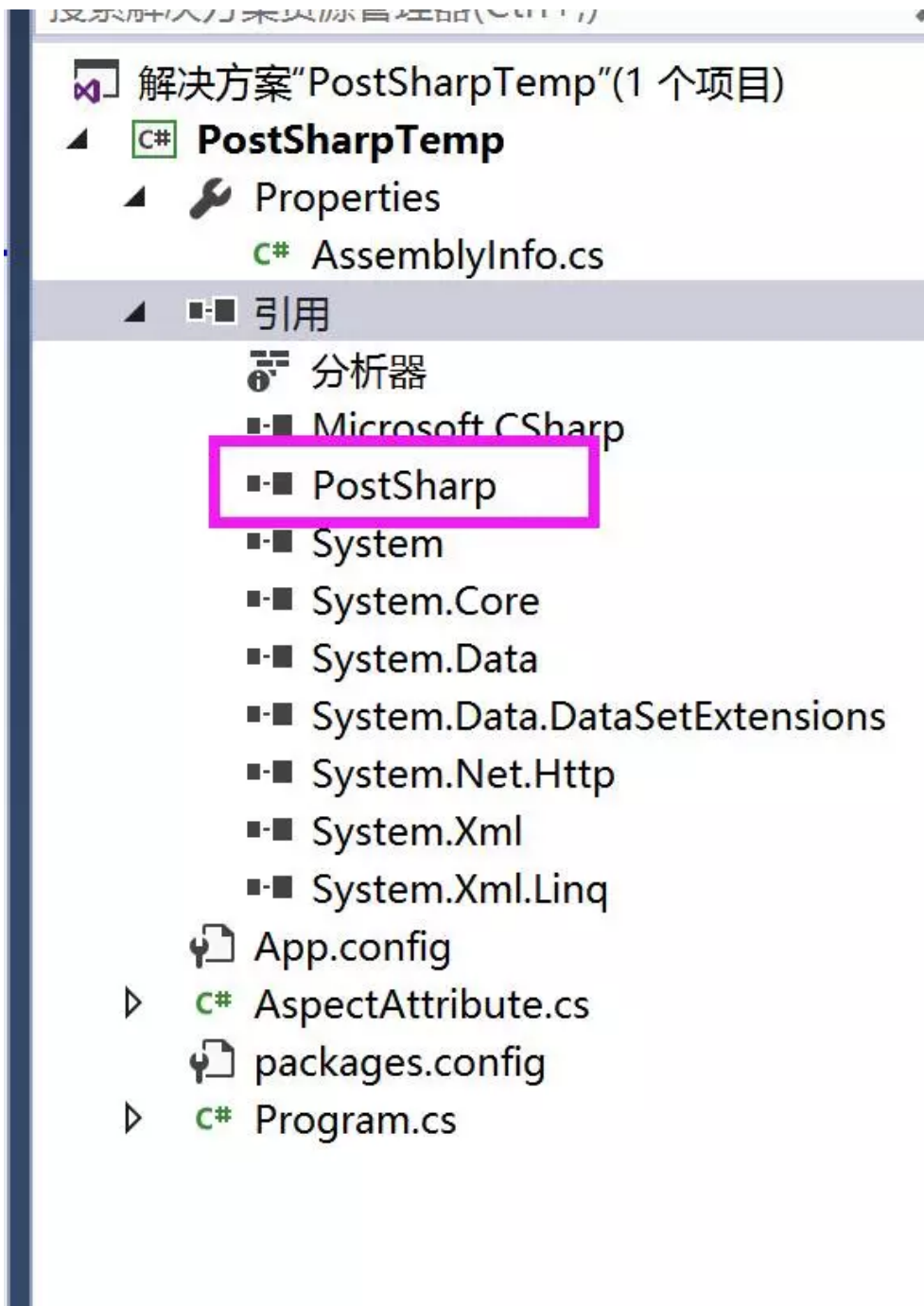
- 3.0版本之前下载了安装包，在项目中引用postsharp.dll，之后编码就可以了
- 但是3.0之后是类似于VS插件的方式工作的，下载postsharp安装包安装之后会在vs的菜单中新增一个postsharp菜单（如下图），可以进行一些设置，在使用的时候不再是引用，而是需要项目上右击“添加postsharp到项目”



## PostSharp示例

- 1、做前期准备工作，从PostSharp官网下载最新版本的安装包，并安装。
- 2、打开VisualStudio，新建解决方案，添加命令行项目。项目上右键后点击 “Add PostSharp to project” 后在弹出窗口按提示操作，会发现已经多了postsharp的引用。





3、添加AspectAttribute切面类，实现对核心方法的输入参数修改。需要注意的是postsharp提供的几种切面类型都是继承自Attribute基类，而且是通过实现对要拦截的类或方法添加特性的方式实现切面思想的。所以我们的切面类 需要按照约定以xxxAttribute的格式命名。

```
[Serializable]
[AttributeUsage(AttributeTargets.Method, AllowMultiple = true, Inherited = true)] public class AspectAttribute :
OnMethodBoundaryAspect
{
    //方法进入时    public override void OnEntry(MethodExecutionArgs args)
    {
        //修改输入参数
        args.Arguments[0] = "jingdong";           //设置方法是否继续执行或退出，若设置的是FlowBehavior.Return方法会直接
退出，不执行后续的所有代码。
        args.FlowBehavior = FlowBehavior.Continue;
    }
    //方法离开时    public override void OnExit(MethodExecutionArgs args)
    {
        Console.WriteLine("exit");
    }
    //方法成功执行时    public override void OnSuccess(MethodExecutionArgs args)
    {
        Console.WriteLine("success");
    }
}
```

```

    }
}

```

4、在program中添加我们的核心方法Start，打印输入参数。在需要实现拦截的方法添加上一个步骤中实现的切面类特性

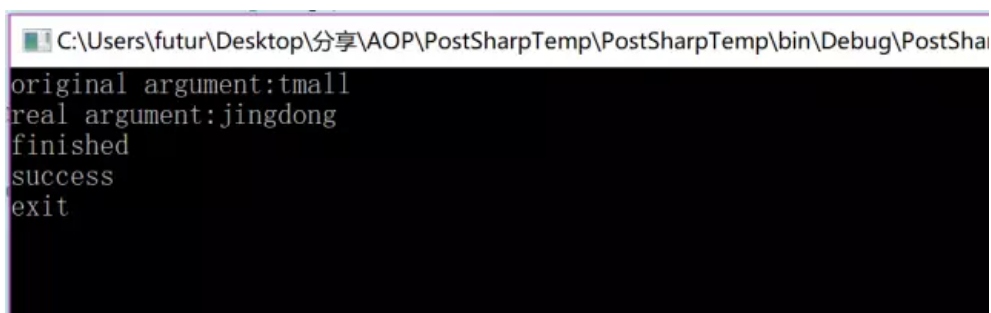
```

public class Program
{
    static void Main(string[] args)
    {
        var arg = "tmall";
        Console.WriteLine($"original argument:{arg}");
        Start(ref arg);
        Console.Read();
    }

    [Aspect]    static void Start(ref string arg)
    {
        Console.WriteLine($"real argument:{arg}");
        Thread.Sleep(1000);
        Console.WriteLine("finished");
    }
}

```

5、由此就简单通过postsharp实现了对方法的拦截，修改输入参数，监听方法成功执行以及退出。看下执行结果，成功的篡改了输入参数，并且可以看出拦截方法的方法体是先于OnSuccess执行的，OnSuccess的拦截执行完成之后才是OnExit。



```

C:\Users\futur\Desktop\分享\AOP\PostSharpTemp\PostSharpTemp\bin\Debug\PostSha
original argument:tmall
real argument:jingdong
finished
success
exit

```

6、之前有说过postsharp是静态织入来实现AOP编程的，那么肯定是通过编译器在编译的时候对代码进行了织入，可以通过反编译exe文件来看下。

```

namespace PostSharpTemp
{
    public class Program
    {
        private static void Main(string[] args)
        {
            string arg = "tmall";
            Console.WriteLine(string.Format("original argument:{0}", arg));
            Program.Start(ref arg);
            Console.Read();
        }

        private static void Start(ref string arg)
        {
            object arg_11_0 = null;
            Arguments<string> arguments = new Arguments<string>();
            arguments.Arg0 = arg;
            MethodExecutionArgs methodExecutionArgs = new MethodExecutionArgs(arg_11_0, arguments);
            <z__a_1.a0.OnEntry(methodExecutionArgs);
            arg = arguments.Arg0;
            switch (methodExecutionArgs.FlowBehavior)
            {
                case FlowBehavior.Return:
                    return;
            }
            try
            {
                Console.WriteLine(string.Format("real argument:{0}", arg));
                Thread.Sleep(1000);
                Console.WriteLine("finished");
                <z__a_1.a0.OnSuccess(methodExecutionArgs);
            }
            finally
            {
                <z__a_1.a0.OnExit(methodExecutionArgs);
            }
        }
    }
}

```

可以看到start方法被加入了若干行代码。

7、有一个小点需要注意，使用postsharp的时候对切面类必须添加Serializable特性，否则在编译的时候就会报错。



## 小结

文章内的小例子主要是为了说明postsharp实现AOP的基本原理，以及实现过程。并没有如同实际项目中使用AOP来做一些日志、安全、持久化之类的辅助功能。