T4

Contents

[一、 What is Metaprogramming 3](#_Toc407721273)

[二、 TT 4](#_Toc407721274)

[Demo 5](#_Toc407721275)

[编辑器—Devart T4 Editor 7](#_Toc407721276)

[三、 T4 Building Blocks 8](#_Toc407721277)

[Expression Blocks 8](#_Toc407721278)

[Statement Blocks 8](#_Toc407721279)

[Class feature Blocks 8](#_Toc407721280)

[Assembly 9](#_Toc407721281)

[Import—就是Using 9](#_Toc407721282)

[Inculde—引用其他TT文件 9](#_Toc407721283)

[输出函数—避免格式问题 回车Tab 10](#_Toc407721284)

[获取当前的TT文件信息—Host 12](#_Toc407721285)

[四、 T4 Building Blocks 13](#_Toc407721286)

[继承扩展默认的Templates 13](#_Toc407721287)

[Preprocessed Template—生成CS, TransformText 14](#_Toc407721288)

[Preprocessed Template的继承 15](#_Toc407721289)

[Tempalte的参数—Session，Initialize（） 17](#_Toc407721290)

[五、 Demo 18](#_Toc407721291)

[从数据库生成Enum 18](#_Toc407721292)

[编译时自动运行TT 19](#_Toc407721293)

[自动实现INotifyProeprtyChanged 19](#_Toc407721294)

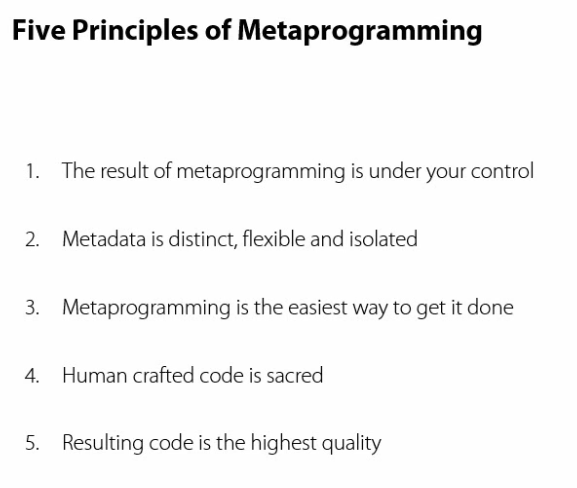
[六、 Trouble Shooting 20](#_Toc407721295)

[Debugger 20](#_Toc407721296)

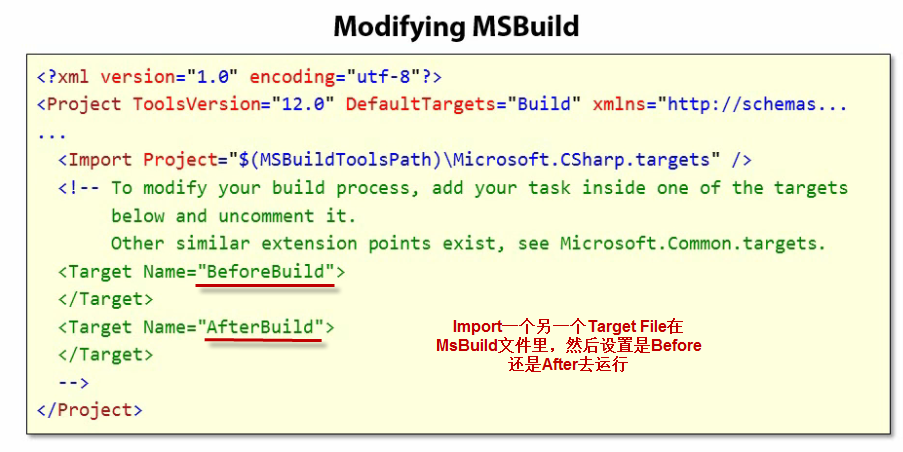
# What is Metaprogramming

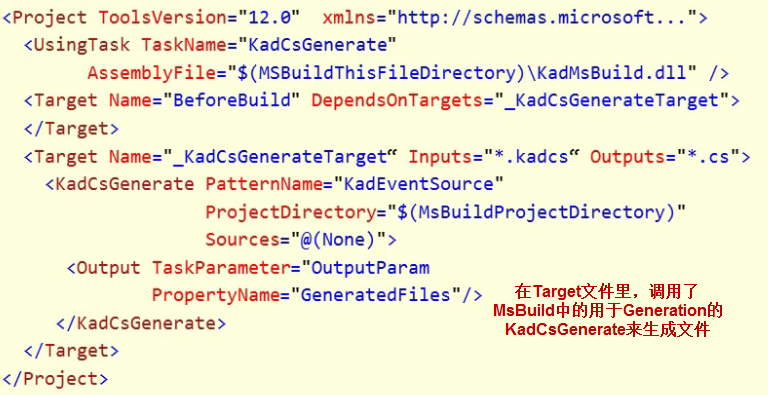
<http://www.pluralsight.com/courses/understanding-metaprogramming>

<http://www.pluralsight.com/courses/t4-templates>



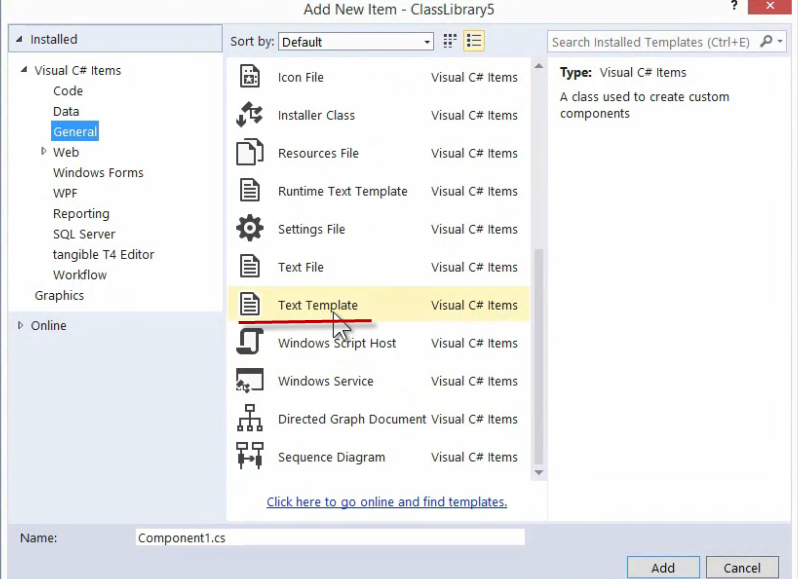
# TT



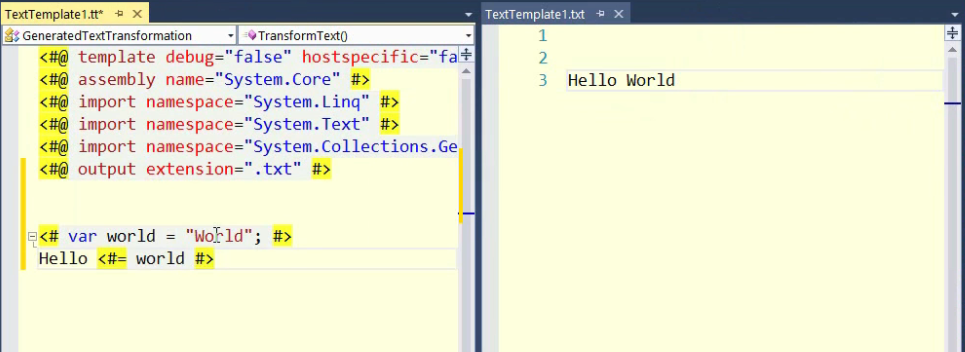


### Demo

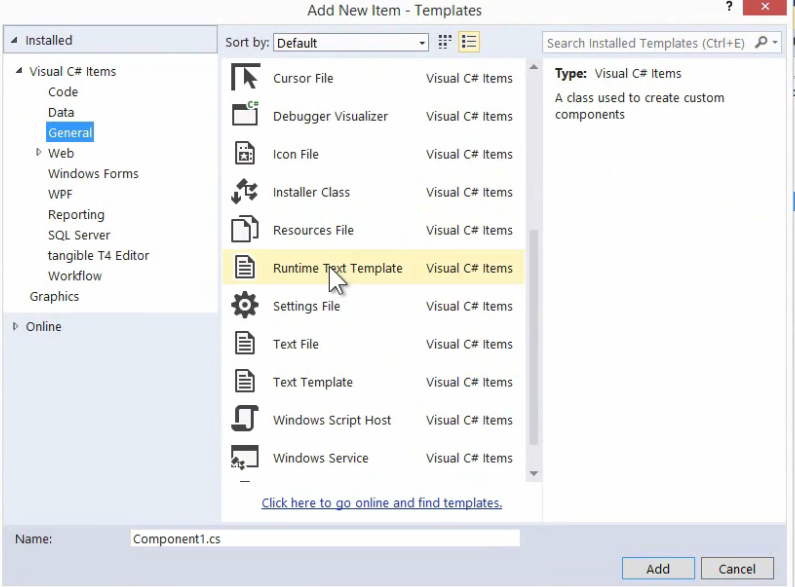
在Project里新增一个Text Template文件



然后写Template，右边就是生成内容的文件



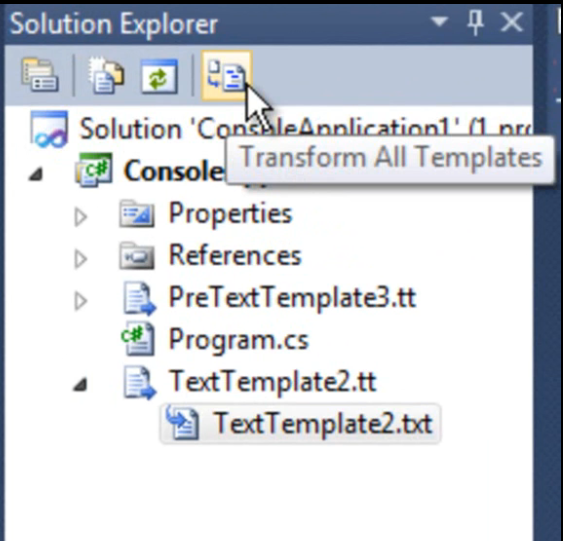
再加一个Runtime Text Template 文件

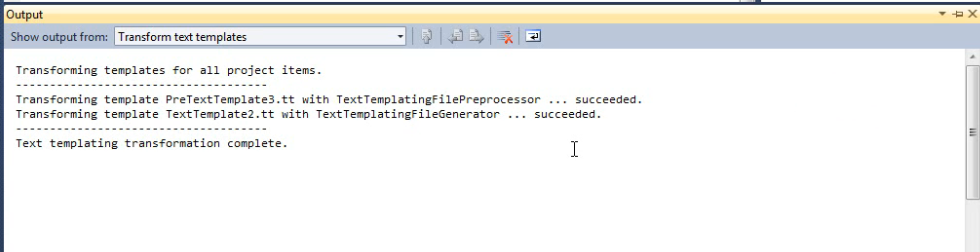


他生成的是一个cs文件



建完tt后可以点右上角的按钮生成所有模板，然后会在Output里出现结果





### 编辑器—Devart T4 Editor

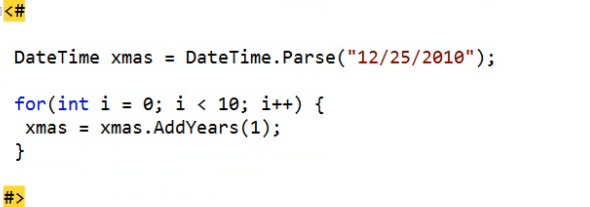
用Extension Manager找Devart T4 Editor

# T4 Building Blocks

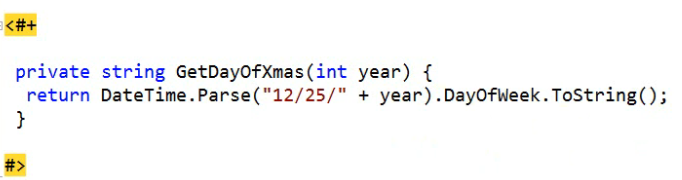
### Expression Blocks



### Statement Blocks

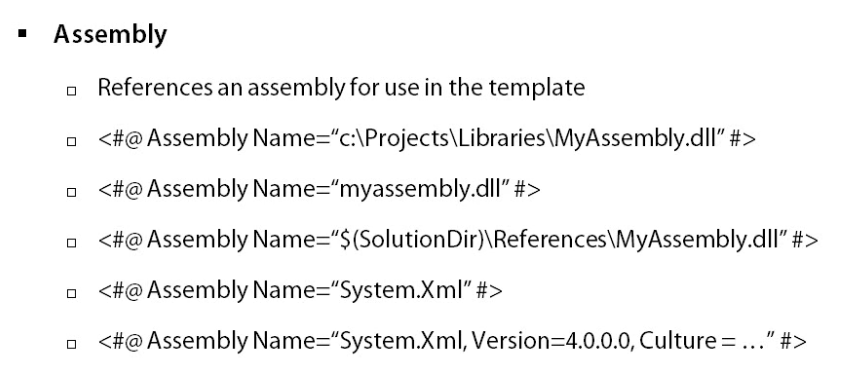


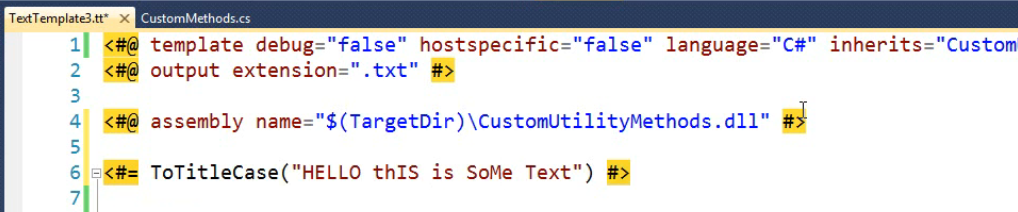
### Class feature Blocks



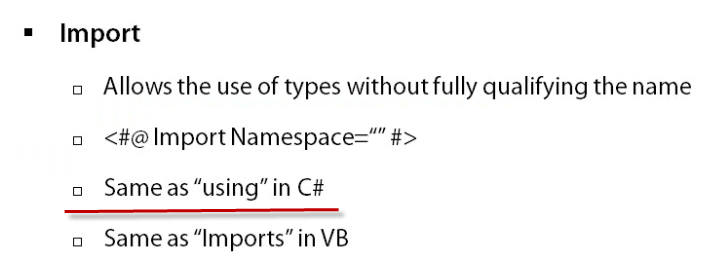


### Assembly

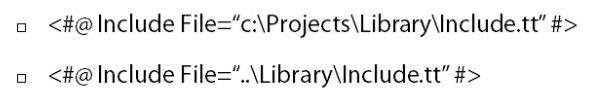




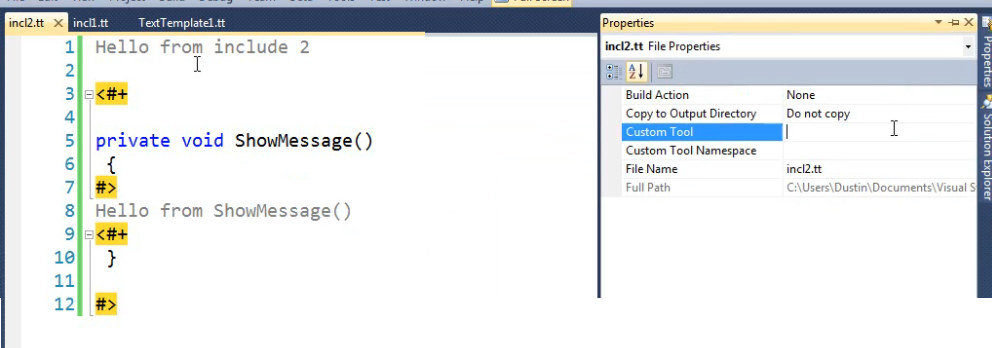
### Import—就是Using

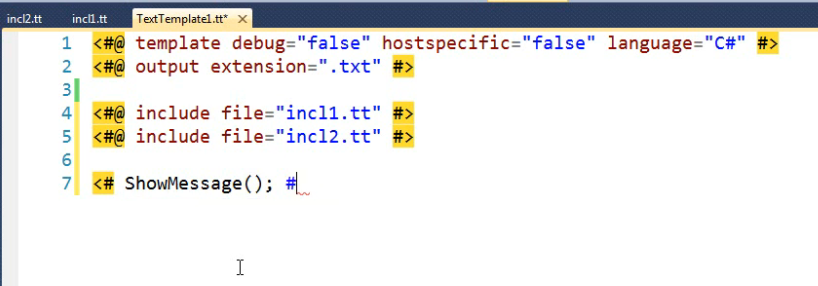


### Inculde—引用其他TT文件

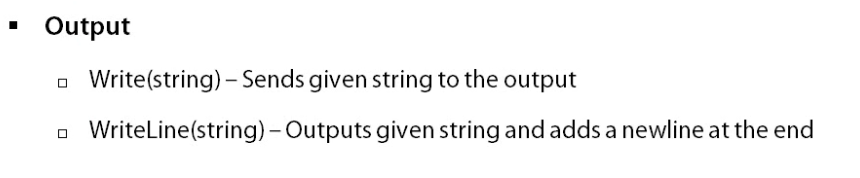


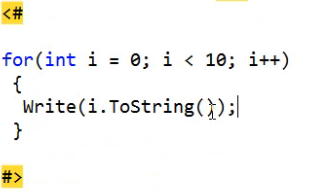
这个Demo建了incl1.tt与Incl2.tt，然后把表头清空，把Custom Tool清空，然后再TextTemp里面去引用他们两个。



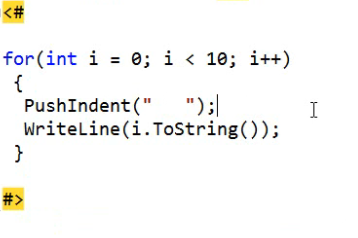


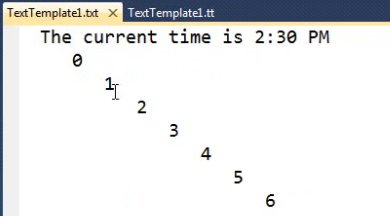
### 输出函数—避免格式问题 回车Tab

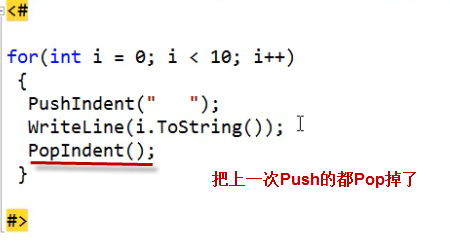


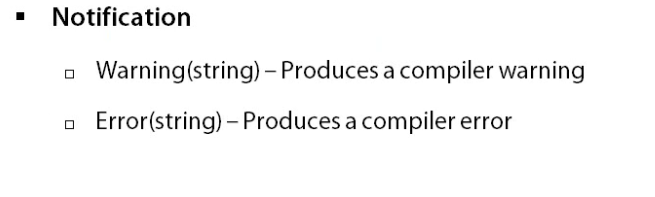


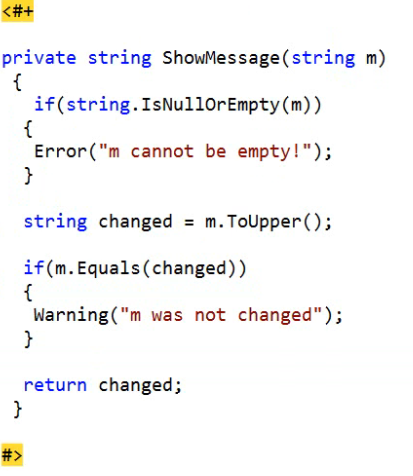












### 获取当前的TT文件信息—Host

要把hostspecific设成true





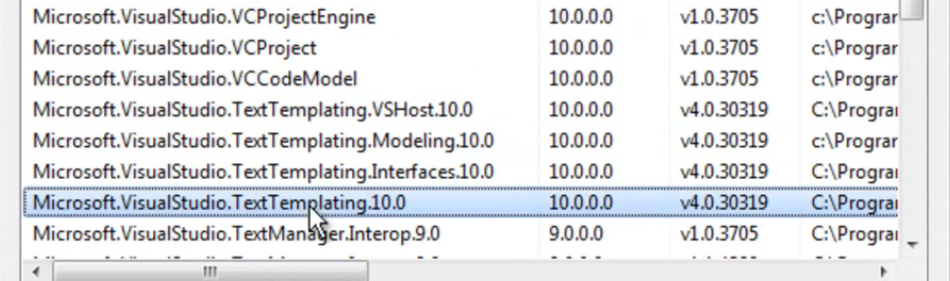
# T4 Building Blocks

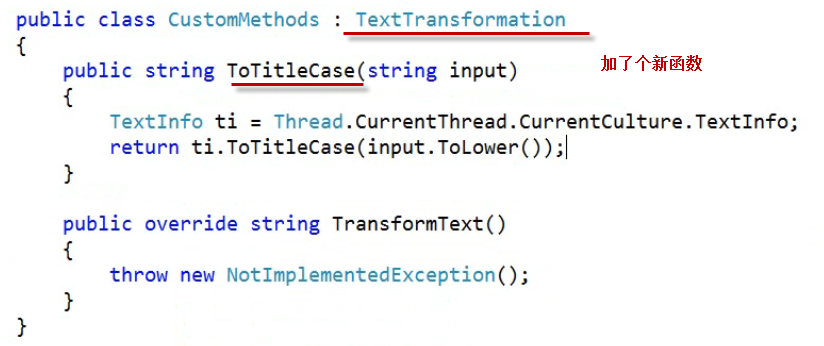
### 继承扩展默认的Templates

基类要写在**一个单独的Library中**，要继承TextTransformation

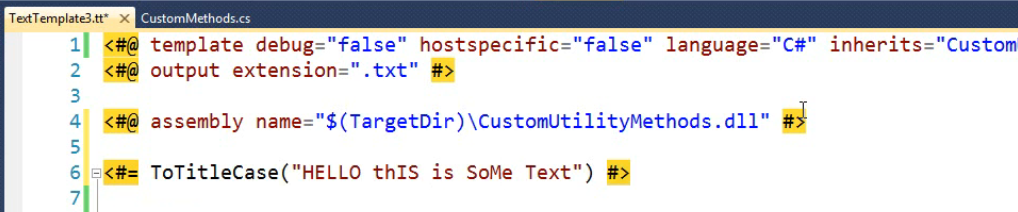
TextTransformation**要安装Visual Studio SDK**，Microsofot.VisuralStudio.TextTemplating命名空间

然后在Template里面写Inherits，和Assembly







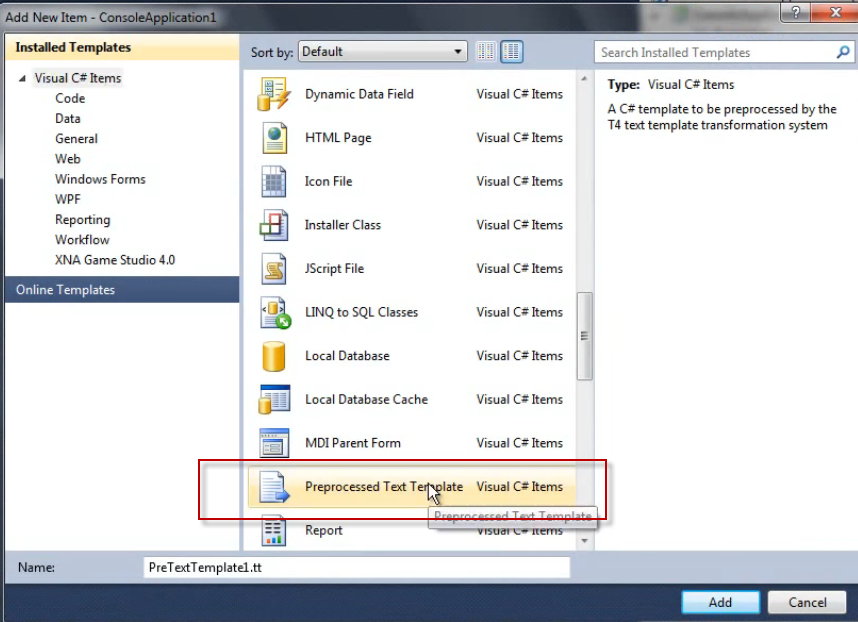


### Preprocessed Template—生成CS, TransformText

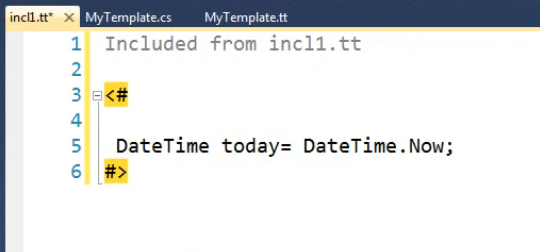
Var t = new PreTextTemplate1();

String s = t.TransformText()





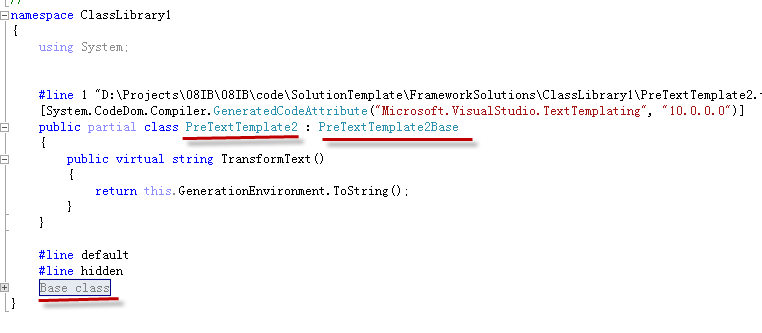
Include一个Template





### Preprocessed Template的继承

Preprocessed Template的继承不能像上面那个来使用，原因是所有默认的Preprocessed Template都会被生成一个基类。如下图，PreTextTemplate2是TT的class,基类是PreTextTemplate2Base，它被生成在下面的BaseClass Region中。生成另一个TT后，也会为这个TT专门生成一个基类。

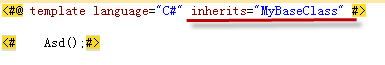


解决方法是新建一个自己的基类，叫MyBaseClass，然后拷贝一个Base Class Region中生成的基类到MyBaseClass.cs中，换个名字，让MyBaseClass继承它。这样MyBaseClass就可以作为公共基类了。

要注意MyBaseClass必须要有一个virtual的Transformtext函数，因为自动生成的函数都会有一个override的TransofrmText函数，否则编译过不了。

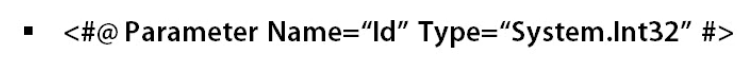


添加完Inherits之后，生成的代码里已经继承自MyBaseClass,继承的基类也没了，虽然没有Intellisense但可以调用基类的代码。



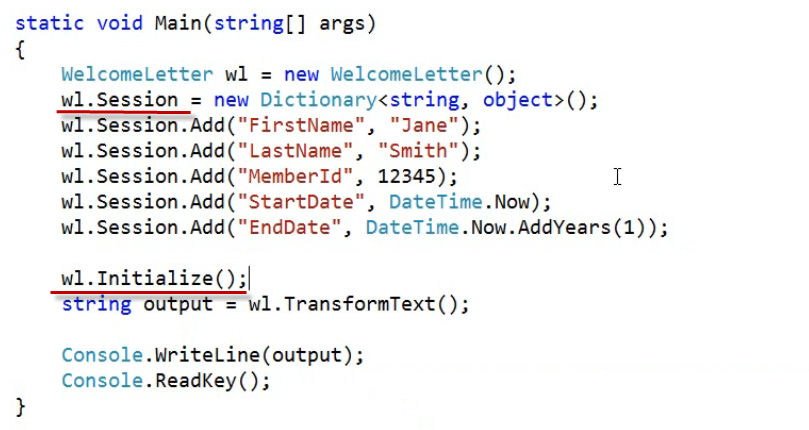


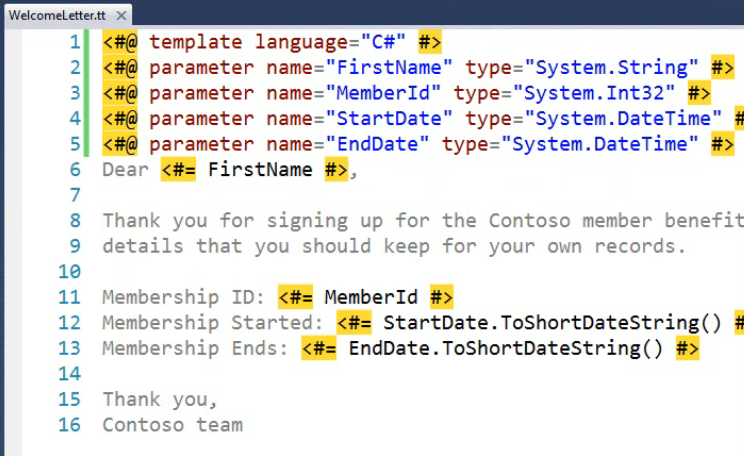
### Tempalte的参数—Session，Initialize（）



传参数用的是 Session属性，类型是Dictionary<string, object>

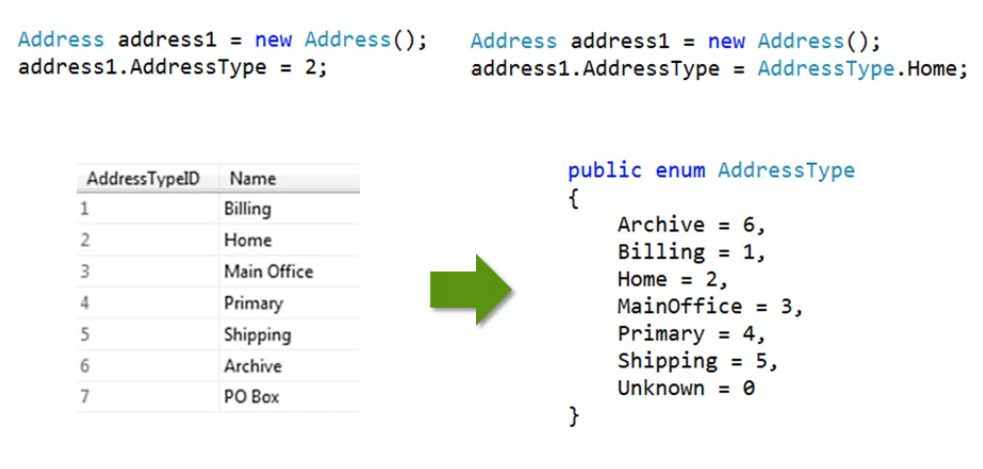
要用Initialize





# Demo

### 从数据库生成Enum

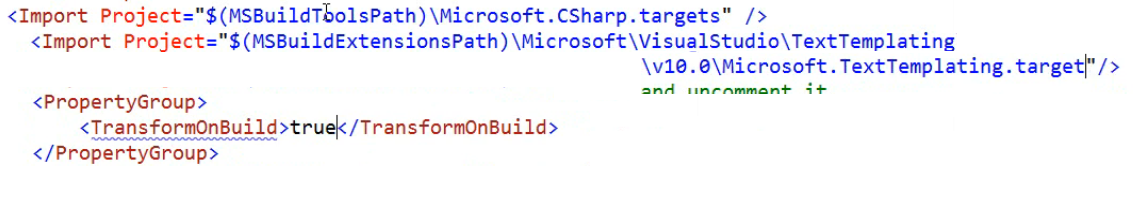


关键是输出类型是.cs，这样每次tt执行完就会生成最新的cs，这样就可以**在编译期用代码**了



### 编译时自动运行TT

要修改Project文件，最后加上Import，把TransdormOnBuild设成true

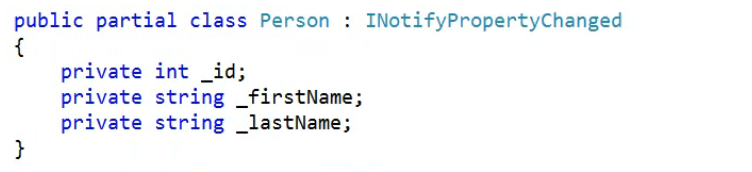


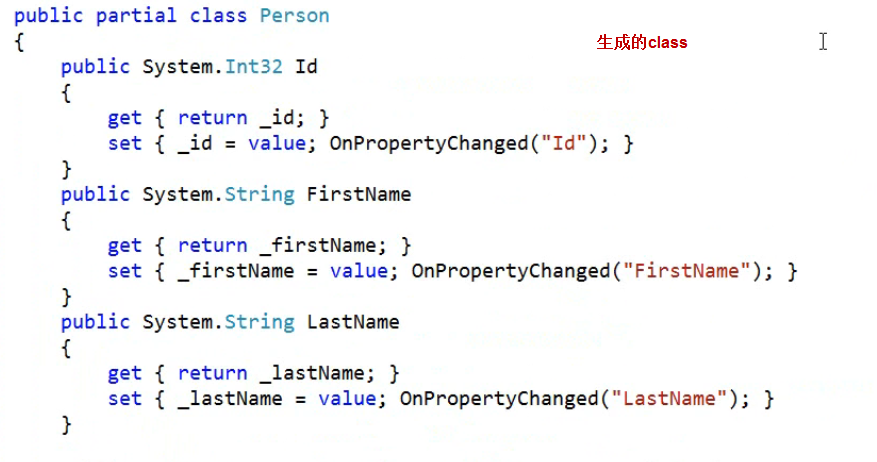
这样在编译后，tt就会自动运行，生成的cs文件就会更新

### 自动实现INotifyProeprtyChanged

做法是手写一个partial **class**，只包含所有property对应的private field，然后会生成一个partial class，包含所有field对应的Property并能Notify。

注意，这个Template将会遍历当前的dll，将所有是INotifyPropertyChanged的对象都生成在一个文件里。





# Trouble Shooting



### Debugger

运行到这行以后，就会Attach一个Debug到当前这行

