(一)什么是T4模板？

T4，即4个T开头的英文字母组合：Text Template Transformation Toolkit。

T4文本模板，即一种自定义规则的代码生成器。根据业务模型可生成任何形式的文本文件或供程序调用的字符串。（模型以适合于应用程序域的形式包含信息，并且可以在应用程序的生存期更改）

VS本身只提供一套基于T4引擎的代码生成的执行环境，由下面程序集构成：

Microsoft.VisualStudio.TextTemplating.10.0.dll

Microsoft.VisualStudio.TextTemplating.Interfaces.10.0.dll

Microsoft.VisualStudio.TextTemplating.Modeling.10.0.dll

Microsoft.VisualStudio.TextTemplating.VSHost.10.0.dll

便利工具：

1、  [T4Toolbox.msi（CodePlex上开源的工具）](http://t4toolbox.codeplex.com/releases/view/44623" \t "http://blog.csdn.net/airingyuan/article/details/_blank)

a)         提供一些可以直接使用的代码生成器，比如Enum SQL View、AzMan wrapper、LINQ to SQL classes、LINQ to SQL schema和Entity Framework DAL等。

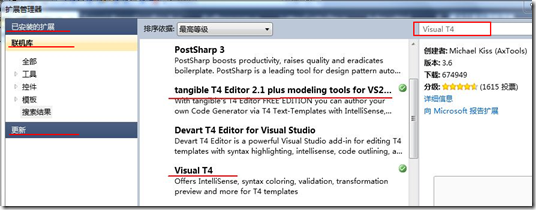
b)         提供一些基于T4方面的VS的扩展：当你安装之后，在“Add New Item”对话框中就会多出一个命名为“Code Generation”的类别，其中包括若干文件模板。

2、  T4 模板编辑器（eg：支持代码着色、智能提示）

a)         [tangible T4 Editor （下载）](http://t4-editor.tangible-engineering.com/Download_T4Editor_Plus_ModelingTools.html" \t "http://blog.csdn.net/airingyuan/article/details/_blank)

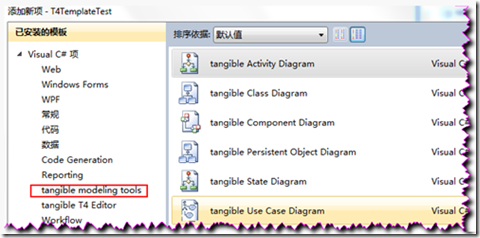
b)         [Visual T4 （下载）](http://visualstudiogallery.msdn.microsoft.com/40a887aa-f3be-40ec-a85d-37044b239591" \t "http://blog.csdn.net/airingyuan/article/details/_blank)

    当然我们也可以通过VS2010中新增的扩展管理器（Extension Manager）来添加Vs扩展插件。扩展管理器（Extension Manager），这和Eclipse/Netbeans有些相似，用户可以直接在IDE中从[Visual Studio 库（Visual Studio Gallery）](http://visualstudiogallery.msdn.microsoft.com/" \t "http://blog.csdn.net/airingyuan/article/details/_blank)找到并下载扩展。**通过VS的菜单Tools->Extension Manager**,这里你可以添加，删除已经安装的VS的扩展插件。打开界面如下：

[](http://images.cnitblog.com/blog/106337/201305/03092644-b0b006d5246c48f8b81b8134dd5ff4e5.png)

笔者在学习 T4 的时候使用过上面两个 T4 模板编辑器。稍作几点对比：

a)         tangible T4 Editor可选择安装内嵌的 UML 模板模型

[](http://images.cnblogs.com/cnblogs_com/heyuquan/201207/20120726233317288.png)

b)         对于不是常用的dll( eg：EnvDTE.dll )，tangible T4 Editor免费版和 Visual T4 都不支持导航，并且所报的提示页不一样

tangible T4 Editor免费版中提示如下：

[](http://images.cnblogs.com/cnblogs_com/heyuquan/201207/201207262333285871.png)

Visual T4中则直接提示：

[image](http://images.cnblogs.com/cnblogs_com/heyuquan/201207/201207262333364711.png)

但是在 Visual T4 中，我们可以通过在程序集中引入 EnvDTE.dll 解决此错误的提示（完成开发后可移除程序集引用），并且还能完美的获得该程序集的智能提示功能，如下图所示：

[](http://images.cnblogs.com/cnblogs_com/heyuquan/201207/201207262333393438.png)

         同时我们也可以看到 Visual T4 中代码着色也更加贴近 VS （蓝色字体标注对象）。

小结：

1.         就“代码着色”和“智能提示”方面Visual T4 工具表现更完美（前提是必须主动在项目中引入对应程序集），但目前最新版本存在缩进问题实在可惜，悲愤中等更新。.

2.         可能你还想要tangible T4 Editor提供的 UML 模板模型，呵呵……现在我本机同时装了这两款 T4编辑器，暂时还没发现冲突。

(二)T4基本结构

T4模板可以分为：指令块、文本块、控制块。

1.         指令块 - 向文本模板化引擎提供关于如何生成转换代码和输出文件的一般指令。

2.         文本块 - 直接复制到输出的内容。

3.         控制块 - 向文本插入可变值并控制文本的条件或重复部件的程序代码，不能在控制块中嵌套控制块。

n  指令块

6个指令<#@ template #>、<#@ parameter#>、<#@ assembly #>、<#@ import #>、<#@ include #>、<#@ output #>、

其中， output 和 assembly 只能用在设计时模板。

1)         T4 模板指令

<#@ template [language="C#"] [hostspecific="true"] [debug="true"] [culture="code"] [inherits="templateBaseClass"] [[compilerOptions="options"]](http://www.cnblogs.com/heyuquan/archive/2012/04/01/2427957.html" \t "http://blog.csdn.net/airingyuan/article/details/_blank)#>

这里只说明下 inherits 属性，其余属性在本文更合适的地方有进行说明。

inherits

指定模板的程序代码继承自另一个类，该基类可以是由其他模板生成。

1)         运行时（预处理过的）文本模板中的继承

如果不指定 inherits 特性，则会从您的文本模板生成基类和派生类。指定 inherits 特性时，仅生成派生类。

2)         设计时文本模板中的继承

设计时模板会生成任何类型的“文本文件”，此文件将组成 Visual Studio 项目的一部分。T4 模板引擎首先要将模板转换为中间程序代码文件，中间代码文件将写入您的 %TEMP% (环境变量) 目录。默认该生成的中间代码继承自 Microsoft.VisualStudio.TextTemplating.TextTransformation 类，但你也可根据需求使用 inherits 特性指定派生于 TextTransformation 类的任何基类。

                   模板引擎生成转换类更详细的请参考本文后面的[何时编译，编译过程](http://www.cnblogs.com/heyuquan/archive/2012/07/26/2610959.html" \l "bianyi" \t "http://blog.csdn.net/airingyuan/article/details/_blank)节。

2)         T4 参数指令

<#@ parameter type="Full.TypeName" name="ParameterName" #>

在 Visual Studio 文本模板中，parameter 指令声明模板代码中从自外部上下文传入的值初始化的属性。可以声明任何远程类型的参数。也就是说，类型必须使用SerializableAttribute进行声明，或者必须从MarshalByRefObject派生。这样可以将参数值传递到在其中处理模板的AppDomain中。

如何使用及内部运作机制请查看我的另一篇文章 [《（译）理解 T4 模板：<#@ parameter #> 指令》](http://www.cnblogs.com/heyuquan/archive/2012/07/26/2609412.html" \t "http://blog.csdn.net/airingyuan/article/details/_blank)。

3)         T4 导入指令

<#@ import namespace="namespace" #>

4)         T4 包含指令

<#@ include file="filePath" #>

a)         为了增加代码的可维护性，将公用函数做为类功能块（<#+ 类功能控制块 #>）存放在单独的文件中，该文件可以被 <#@include#> 到一个或多个模板文件中。

b)         对于包含文件，文件扩展名使用 .ttinclude可读性更好。（以区分后缀为 .tt的运行时或设计时文本模板）

5)         T4 输出指令

<#@ output extension=".fileNameExtension" [encoding="encoding"] #>

运行时（预处理）文本模板中不需要 output 指令。应用程序通过调用TextTransform() 来获取已生成的字符串。

6)         T4 程序集指令

<#@ assembly name="[assembly strong name|assembly file name]" #>

在预处理文本模板中，assembly 指令无效。改为在 Visual Studio 项目中直接“添加引用”。

程序集名称应为以下各项之一：

1.         GAC 中程序集的强名称，例如 System.Xml.dll。还可以使用长名称，例如 name="System.Xml, Version=4.0.0.0, Culture=neutral,PublicKeyToken=b77……"。

2.         程序集的绝对路径

可以使用 $(variableName) 语法引用 Visual Studio 或MSBuild变量（如 $(SolutionDir)），以及使用 %VariableName% 来引用环境变量。

                   另，给出一些常用的 [【生成命令和属性的宏】](http://msdn.microsoft.com/zh-cn/library/cc461408(v=VS.71).aspx" \t "http://blog.csdn.net/airingyuan/article/details/_blank)

|  |  |
| --- | --- |
| $(ConfigurationName) | 当前项目配置的名称（如“Debug”）。 |
| $(PlatformName) | 当前项目平台的名称（如“Win32”）。 |
| $(ProjectName) | 项目的基本名称。 |
| $(TargetDir) | 生成的主输出文件的目录（定义为驱动器 + 路径）；包括尾部的反斜杠“\”。 |
| $(TargetName) | 生成的主输出文件的基本名称。 |
| $(FrameworkDir) | 安装 .NET Framework 的目录。 |
| $(FrameworkVersion) | Visual Studio 使用的 .NET Framework 版本。 |
| $(WebDeployPath) | 从 Web 部署根到项目输出所属于的位置的相对路径。返回与RelativePath相同的值。 |
| $(WebDeployRoot) | 指向<localhost>位置的绝对路径。例如，c:\inetpub\wwwroot。 |

n  控制块

有三种类型的控制块，根据其左括号对它们进行区分：

1.      <# 标准控制块 #>                            可以包含语句。

2.      <#= 表达式控制块 #>            将一个可以计算为字符串的表达式括起来,用于提供要写入“输出”文件的字符串的代码。

3.      <#+ 类功能控制块 #>            可以使用类功能控制块向文本模板添加方法、属性、字段甚至是嵌套类。必须作为文件中的最后一个块显示，或者用<#@ include #>引入外部文件。

注意：

1)         始终使用 {...}花括号来包含内嵌的嵌套语句，否则会报错。（哪怕花括号中只有一句代码）

2)         控制块不能互相嵌套。必须先终止之前的控制块，然后才能打开另一个。

(三)设计时模板和运行时模板

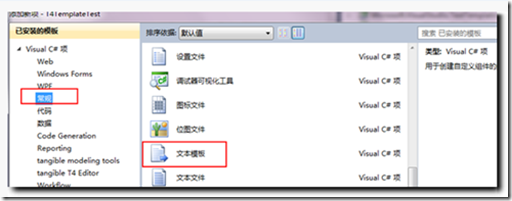
T4文本模板分为：设计时模板和运行时模板

n  添加模板

1.         设计时模板（文本模板）

优势：当需求变化时，可以根据业务需求调整模型（输入），按照指定规则将“模型”生成任何类型的“文本文件”，例如：网页、资源文件或任何语言的程序源代码。（模型：是描述应用程序特定方面的数据源。它可以是任何形式、任何类型的文件或**[数据库](http://lib.csdn.net/base/mysql" \o "MySQL知识库" \t "http://blog.csdn.net/airingyuan/article/details/_blank)**。如：数据库、配置文件、UML 模型、DSL 模型或其他源）

a)         VS中新建文件——常规——文本模板。（如图）

[](http://images.cnblogs.com/cnblogs_com/heyuquan/201207/201207262333457163.png)

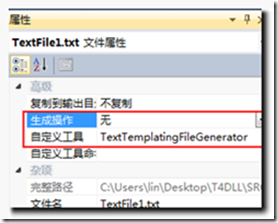
该模板文件中已包含下列指令：

<#@ template debug="false" hostspecific="false" language="C#" #>

<#@ output extension=".txt" #>

b)         或则，添加“纯文本文件”并设置下图属性，加入相关指令。（后缀推荐改为标准的 \*.tt）

设计时模板: TextTemplatingFileGenerator

[](http://images.cnblogs.com/cnblogs_com/heyuquan/201207/201207262333533496.png)

2.         运行时模板（已预处理的文本模板）

优势：当需求变化时，可以根据业务需求调整模型（输入），在运行时按照指定规则将“模型”生成为“文本字符串”。

1.         VS中新建文件——常规——已预处理的文本模板。

该模板文件包含指令：<#@ template language="C#" #>

2.         或则，添加“纯文本文件”并设置相应属性，加入相关指令。

运行时模板:TextTemplatingFilePreprocessor

n  何时编译，编译过程

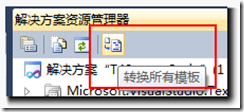
1.         何时编译

在下列任何一种情况下，将执行模板，同时生成附属文件，生成的文件将作为项目的一部分编译。（属性框----生成操作：编译）

1)         编辑模板（模板有异动且没有被保存），当前编辑模板失去焦点。

2)         保存模板。

3)         在“解决方案资源管理器”工具栏中单击“转换所有模板”。转换解决方案中的所有模板。

[](http://images.cnblogs.com/cnblogs_com/heyuquan/201207/201207262333595552.png)

4)         右击“解决方案资源管理器”中的一个或多个模板文件，然后选择“运行自定义工具”。

2.         编译过程

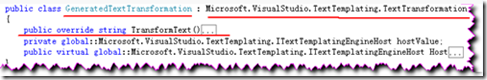
设计时模板

1)         文本模板转换引擎将“文本模板”转换为可执行的cs代码——“转换类”。转换类(\*.cs)存于临时目录下。（临时目录在“环境变量”中设置：右键“我的电脑”—“属性”—“高级系统设置”—“高级”选项卡中“环境变量”—TEMP变量）

命名空间：Microsoft.VisualStudio.TextTemplating + 随机码

基类：Microsoft.VisualStudio.TextTemplating.TextTransformation

类名：GeneratedTextTransformation

[](http://images.cnblogs.com/cnblogs_com/heyuquan/201207/201207262334035948.png)

2)         引擎编译生成的“转换类”生成dll，dll存于临时目录下。具体是哪个dll可以在模板的“调试环境”下使用System.Reflection.Assembly.GetExecutingAssembly();获取。

3)         执行已编译的转换类，生成“文件”。新文件会在“解决方案资源管理器”中出现在文本模板文件下。

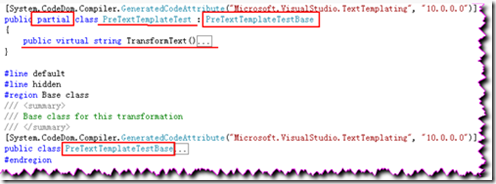
运行时模板

1)         运行时模板没有<#@ output #>指令，文本模板引擎将“运行时模板”直接编译为cs文件，作为项目的一部分编译。新文件会在“解决方案资源管理器”中出现在文本模板文件下。

命名空间：默认为所属程序集的命名空间

基类：模板文件名 + Base

类名：模板文件名（PreTextTemplateTest.tt）——注意是“分部类”

[](http://images.cnblogs.com/cnblogs_com/heyuquan/201207/201207262334074458.png)

2)         生成的代码文件随着项目一起编译，并可在应用程序中通过调用生成类中的TransformText() 方法输出“文本字符串”。

另外，若要在特定命名空间中放置模板转换生成的类，需设置模板文件的“自定义工具命名空间”属性。

3.         注意事项

1)         控制块使用陷进

TransformText() 方法是由模板引擎将模板中的所有“控制块”代码（包括“包含的模板”）组合生成。所以在使用控制块时应注意以下几点：

a)         语言：只能使用一种语言。

b)         局部变量：确保局部变量的名称不会冲突。

2)         文本模板在单独的AppDomain中运行

请注意，文本模板在与主应用程序分开的AppDomain中运行。在大多数情况下这并不重要，但在某些复杂的情况下您可能会发现一些限制。例如，如果要从单独的服务将数据传入模板或从中传出数据，则该服务必须提供可序列化的 API。

(四)技巧

l  快速编写模板

以生成文件为原型，然后逐步插入用于改变结果的控制块。

l  T4文本模板的断点调试

1.         注册表：设置DbgJITDebugLaunchSetting值为 2。

(x86系统): HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\.NETFramework

(x64 系统): HKEY\_LOCAL\_MACHINE\SOFTWARE\Wow6432Node\Microsoft\.NETFramework

2.         为template指令添加debug="true"特性：<#@ template debug="true"#>

3.         命令：

<# System.Diagnostics.Debugger.Launch();#>                  在模板执行到特定点启动调试器。如果用Debugger.Break()启动调试器在调试完后会导致 VS 奔溃。

<#System.Diagnostics.Debugger.Break();#>                     启动调试器后，使用此命令在后续特定点处再次进入调试模式，相当于断点。

使用方法：必须使用“Debugger.Launch()”命令启动调试器（如下图，启动新实例或使用已存在的VS附加。注意，若此处点击取消则将关闭当前IDE），调试完后可以不用中断调试，不影响模板编辑，当再次编译模板时如果存在“Debugger.Break()”命令则自动进入调试模式。

[](http://images.cnblogs.com/cnblogs_com/heyuquan/201210/201210230952102044.jpg)

l  向模板传递参数的两种方法

1.         使用 <#@ parameter#> 指令引入参数，由模板引擎生成属性访问代码。详细请看 [《（译）理解 T4 模板：<#@ parameter #> 指令》](http://www.cnblogs.com/heyuquan/archive/2012/07/26/2609412.html" \t "http://blog.csdn.net/airingyuan/article/details/_blank)。

2.         在构造函数中传递参数。只适用于运行时模板，此种模板生成的代码以分部类的形式编写。可以在项目的另一个文件中创建同一个类的其他部分，该文件可以包含一个带参数的构造函数、若干属性和函数，在调用 TransformText() 实例方法前进行初始化。

l  使用模板继承共享内容

可以通过编写基类模板（可以是抽象模板）在文本模板之间共享内容。使用<@#template#> 指令的 inherits 特性指定基类。

l  运行时调用设计时模板返回字符串

调用 Microsoft.VisualStudio.TextTemplating.Engine 的 ProcessTemplate 方法。

publicstring ProcessTemplate(

     string content,

     ITextTemplatingEngineHost host

)

        content    参数指定文本模板的内容，eg： 使用System.IO.File.ReadAllText(Path) 进行读取

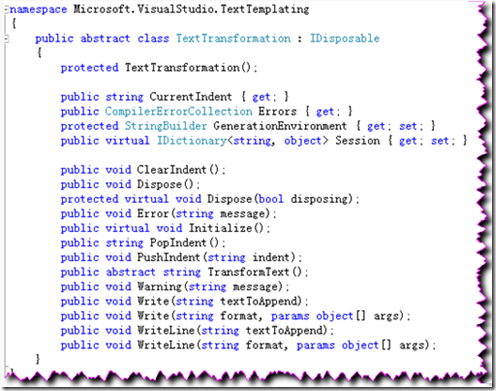
host        参数指定的宿主，必须是实现 ITextTemplatingEngineHost 的类。这是由模板引擎回调的。宿主必须能记录错误、解析对程序集和包含文件的引用、提供可在其中执行模板的应用程序域并为每条指令调用相应的处理器。

[演练：创建自定义文本模板宿主](http://msdn.microsoft.com/zh-cn/library/bb126579.aspx" \t "http://blog.csdn.net/airingyuan/article/details/_blank)

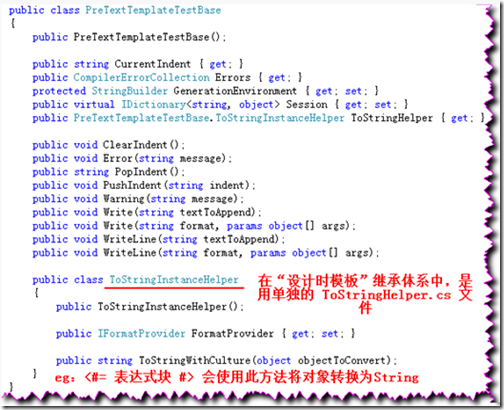
(五)常用方法

n  模板基类提供的方法

设计时模板继承TextTransformation抽象类

[](http://images.cnblogs.com/cnblogs_com/heyuquan/201207/201207262334166222.png)

运行时模板默认继承自动生成的基类

[](http://images.cnblogs.com/cnblogs_com/heyuquan/201207/201207262334253276.png)

1.         Write() 和WriteLine() 方法

写入目的输出文本的三种方式：

a)         文本块

b)         表达式控制块：      <#= 变量 #>

c)         标准控制块：           <# Write() | WriteLine() #>，因为控制块不能嵌套，所以此种方式比<#= 变量 #>书写更优雅。

2.         输出文本缩进设置

可以使用缩进方法设置文本模板输出的格式。

a)         PushIndent(string indent)         添加指定格式，内部会将字符长度加入到缓存变量indentLengths列表（List<int>）。

b)         PopIndent()                 以“堆栈（先进后出）”形式移除格式，内部按indentLengths列表中存的字符长度进行移除。

c)         ClearIndent()              删除所有缩进。

注意：格式用完后要注意清除，否则可能出现模板中的空行会生成 Write(“\r\n”) 中间代码，最终造成将缩进的格式错误输出到了目的文件。

Eg:

[](http://images.cnblogs.com/cnblogs_com/heyuquan/201207/201207262334316968.png)

3.         错误报告

若要在 Visual Studio 错误窗口中放置错误消息和警告消息，可以使用以下方法：

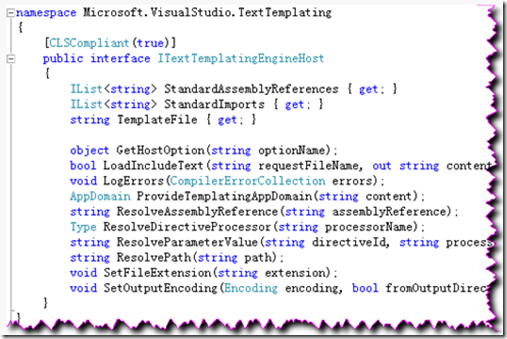
<# this.Error("An error message"); #>

<# Warning("A warning message"); #>

n  使用执行模板的主机（例如 Visual Studio）公开的方法和属性。这适用于常规文本模板，而不是预处理过的文本模板。

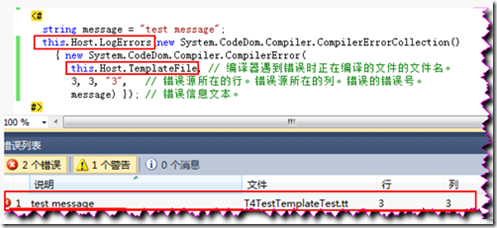
首先，给 template 指令添加hostspecific="true" 特性，以便使用this.Host对象。

(Microsoft.VisualStudio.TextTemplating.ITextTemplatingEngineHost)接口提供方法

[](http://images.cnblogs.com/cnblogs_com/heyuquan/201207/201207262334386124.png)

1.         使用this.Host.ResolvePath()从相对路径名打开文件

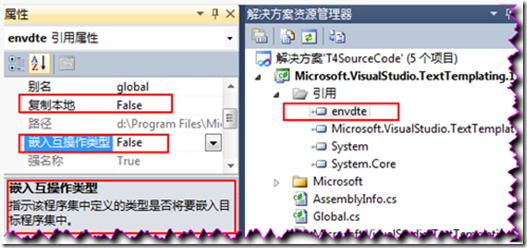
2.         使用LogErrors() 显示错误消息，如下图：

[](http://images.cnblogs.com/cnblogs_com/heyuquan/201207/201207262334504904.png)

3.         使用 Visual Studio 中提供的服务（加载EnvDTE.dll ）

EnvDTE是組件包裝 COM 程式庫，其中包含了 Visual Studio 核心 Automation 的物件及成員。

引入 EnvDTE.dll 组件后应按下图“属性”进行设置：

[](http://images.cnblogs.com/cnblogs_com/heyuquan/201207/201207262334568006.png)

示例：

<#@ assembly name="EnvDTE" #>

<#

IServiceProvider serviceProvider = (IServiceProvider)this.Host;

EnvDTE.DTEdte = (EnvDTE.DTE) serviceProvider.GetService(typeof(EnvDTE.DTE));

dte.Solution.SaveAs("C:\\backup\_Solution");

#>

http://blog.csdn.net/airingyuan/article/details/17240255