Date

Visual Studio 2010 DSL 开发与设计实践系列

第 10 讲 DSL的代码生成（下）

Microsoft

自我介绍

- 陈俊先

- .NET 高级讲师

内容提要

- 在Template中使用"\"转义
- 实现输出缩进功能
- 在Template中输出Warning或Error
- Debug
- 文本模板的架构
- T4的自定义宿主

在Template中使用"\"转义

```
\<#@ output extension=".tt" \#>
\<#@ assembly name="System.Xml.dll" \#>
\<#@ import namespace="System.Xml" \#>
\<#
XmlDocument xDoc = new XmlDocument();
//System.Diagnostics.Debugger.Break();
    xDoc.Load(@"E:\CSharp\Overview.xml");
    XmlAttributeCollection attributes = xDoc.Attributes;
    if (attributes != null)
    {
      foreach (XmlAttribute attr in attributes)
      {\#>
        \<#= attr.Name \#>
      \<#}
     }
\#>
```

在Template中使用"\"转义

- 输出：

```
<#@ output extension=".tt" #>
<#@ assembly name="System.Xml.dll" #>
<#@ import namespace="System.Xml" #>
<#
XmlDocument xDoc = new XmlDocument();
//System.Diagnostics.Debugger.Break();
    xDoc.Load(@"E:\CSharp\Overview.xml");
    XmlAttributeCollection attributes = xDoc.Attributes;
    if (attributes != null)
    {
      foreach (XmlAttribute attr in attributes)
      {#>
          <#= attr.Name #>
      <#}
     }
#>
```

TextTransformation类的属性和方法

- 输出：
- Write()
- WriteLine()

- 缩进：
- CuurentIndent
- PushIndent()
- PopIndent()
- ClearIndent()

Write()、WriteLine()

```
<#
    int i = 10;
    while (i-- > 0)
    {
        WriteLine((i.ToString()));
    }
#>
```

缩进方法

```
<#
    WriteLine(CurrentIndent + "Hello");
    PushIndent("    ");
    WriteLine(CurrentIndent + "Hello");
    PushIndent("    ");
    WriteLine(CurrentIndent + "Hello");
    ClearIndent();
    WriteLine(CurrentIndent + "Hello");
    PushIndent("    ");
    WriteLine(CurrentIndent + "Hello");
#>
```

输出：

```
Hello
        Hello
                Hello
Hello
        Hello
```

Demo (IndentExample)

在Template中输出Warning或Error

- TextTransformation类的属性和方法
  Warning()
- Error()

```
<#
 try
 {
   string str = null;
   Write(str.Length.ToString());
 }
 catch (Exception e)
 {
   Error(e.Message);
 }
#>
```

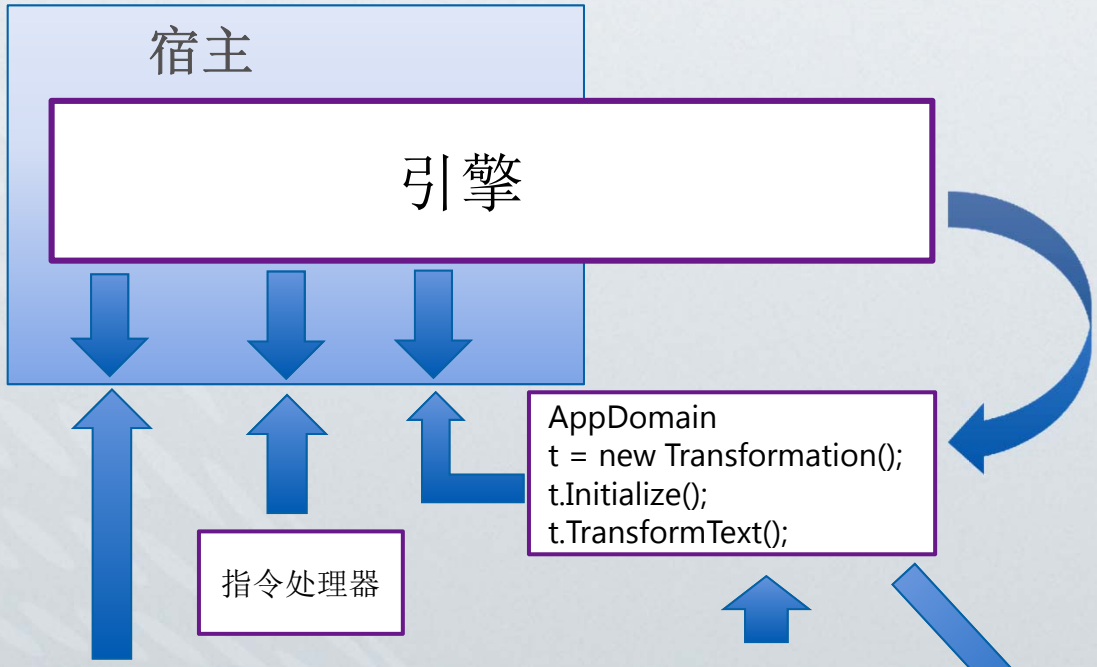
Demo(WarningAndErrorExample)

模板调试

- **Debug**
- 如何跟踪调试模板程序

Demo(DebugExample)

文本模板的架构




```
AppDomain
t = new Transformation();
t.Initialize();
t.TransformText();
```

```
class Transformation : ModedingTextTransformation
{
    private Model mode;
     public void Initialize()
     {
         model = LoadModel("foo.Model");
     }
     public string TransformText()
     {
          WriteLine("List of Item：");
           foreach(Item item in model.Items)
           {
               Write("Name：");
               WriteLine(item.Name);
           }
      }
}
```

```
<#@ template
base=ModelingTextTransformation #>
<#@ itemDirective requries="foo.model"
provides="model" #>
<#
    foreach(Item item in model.Items)
#>
Name：<#= item.Name #>
<# } #>
```

Foo.Model

```
List of Item：
Name： ItemOne
Name：itemTwo
Name：ItemThree
```

自定义宿主

- 文本模板引擎只接受一个模板内容的字符串，并负责生成输出一个字符串，而处理哪个模板，从哪里输出等均由宿主完成。

- class CustomCmdLineHost : ITextTemplatingEngineHost
- {
  - 。。。
- }

自定义宿主

- CustomCmdLineHost host = new CustomCmdLineHost();
- Engine engine = new Engine();

- host.TemplateFileValue = templateFileName;

- //Read the text template.
- string input = File.ReadAllText(templateFileName);

- //Transform the text template.
- string output = engine.ProcessTemplate(input, host);

Demo(CustomTemplateHost)

Microsoft®

Your potential. Our passion.®