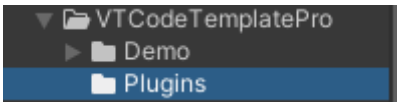
VTCodeTemplate介绍

`VTCodeTemplate`是一款基于模板格式语言生成文本文件的插件。包括但不限于C#代码文件，Lua，Python，Json格式的文件等。

对于有经常需要生成格式化代码的项目非常有用。本工具设计了一套模板语法，简单实用。

插件目录

`VTCodeTemplate`库文件位于`Plugins`目录下，可以拷贝到自己的工程中



模板文件

`VTCodeTemplate`仅支持后缀为`.vtemplate`的模板文件。包括主文件和附加文件都遵循这个规则。

语法介绍

`VTCodeTemplate`的语句用两个美元符号`$`来表示，形如：`$.....$`。且每个语句独占一行，前后可以有空白字符。

普通文本替换语句

普通文本替换语句的语法格式为`${Tag}$`。例如：`$Content$`，Content是可以用来映射需要替换的值。

循环语句

循环语句用两行语句来标记，它们分别表示循环区域的开始和循环区域的结束。形式如下：

```
$Repeat LoopArea$
...
$EndRepeat$
```

`$Repeat {UniqueTag}$`是开始循环语句的语法格式。其中`UniqueTag`用作该循环节点的索引。

`$EndRepeat$`是结束循环语句的语法格式。

在开始循环语句和结束循环语句中间的模板文本块，会归属于该循环区域，可以使用循环区域的数据上下文来生成目标文本。

可以针对循环区域创建每一次迭代的数据上下文，且优先于全局数据上下文。

条件控制语句

条件控制用两行语句来标记，它们分别表示条件控制区域的开始和条件控制区域的结束。形式如下：

```
$If ConditionArea$
...
$EndIf$
```

`$If {Tag}$`是条件控制语句的语法格式。其中的Tag是条件标记。该标记是全局的，且不必唯一。标记被激活后，所有使用该标记的条件控制区域都会生成

`$EndIf$`是结束条件控制语句的语法格式。

在开始条件控制语句和结束条件控制语句中间的模板文本块，会归属于该区域。不过需要注意的是，条件控制区域没有数据上下文，不会生成文本，只根据Tag是否被激活来判断是否返回该区域模板，由更上层区域来生成目标文本。

子区域语句

子区域用两行语句来标记，它们分别表示区域的开始和区域的结束。形式如下：

```
$SubArea AreaTag$
...
$EndSubArea$
```

`$SubArea {UniqueTag}$`是子区域开始语句的语法格式。其中的Tag是子区域的标记，用来索引子区域。该标记要求全局唯一。

`$EndSubArea$`是结束子区域的语法格式。

在开始子区域语句和结束子区域语句中间的模板文本块，会归属于该区域，可以使用该区域的数据上下文来生成目标文本。

子区域有自己的数据上下文，且优先于全局的数据上下文。

包含语句

包含语句只有一行，格式如下：

```
$Include "OtherTmpl.vtemplate"$
```

`$Include "{FilePath}"$`是包含语句的语法格式。其中，需要输入被包含文件的相对路径。该路径是相对于主文件的文件路径。

使用包含语句，在预处理阶段会打开该文件，并从该行开始替换成文件的内容。

数据上下文

`VTCodeTemplate`的数据上下文分为`全局数据上下文`，`循环区域数据上下文`，`条件控制数据上下文`，`子区域数据上下文`四类。在代码表现上，`全局数据上下文`和`子区域数据上下文`是相同的接口。`条件控制数据上下文`用来控制条件的激活状态。`循环区域数据上下文`用来控制每一轮迭代使用的数据上下文。

获取全局数据上下文

全局数据上下文通过`TemplateHandler`的`GlobalDataContext`属性获取。如下：

```
IDataContext globalDataContext = tempalteHandler.GlobalDataContext;
```

获取循环区域数据上下文

循环数据上下文需要循环标记来进行索引，通过`TemplateHandler`的`GetRepeatDataContext`方法来获取。获取方法如下：

```
IRepeatDataContext repeatDataContext =
tempalteHandler.GetRepeatDataContext("LoopArea");
```

为循环区域增加数据上下文

每一轮迭代生成时，需要定制循环生成文本时使用的数据上下文。下面是一个使用示例

```
IRepeatDataContext repeatDataContext =
tempalteHandler.GetRepeatDataContext("LogArea");
for(int i = 0; i < 10; i++)
{
    IDataContext currentDataContext =
repeatDataContext.CreateNextIteratorDataContext();
    currentDataContext.AddReplaceValue("Count", (i + 1).ToString());
}
```

获取子区域数据上下文

子区域数据上下文需要子区域标记来索引，通过`TemplateHandler`的`GetSubAreaDataContext`方法来获取。获取方法如下：

```
IDataContext subAreaDataContext =
tempalteHandler.GetSubAreaDataContext("SubAreaTag");
```

设置条件状态

条件控制数据上下文并不对外开放。而是通过`TemplateHandler`的`SetConditionActive`方法来设置激活状态。使用方法如下：

```
tempalteHandler.SetConditionActive("ConditionTag", true);
```

数据上下文接口

IDataContext

```
public interface IDataContext
{
    string[] ReplaceTags { get; }

    void AddReplaceValue(string tag, string value);

    string GetReplaceValue(string tag);
}
```

- ReplaceTas
  - 获取所有上下文中的标记，对应`普通文本替换语句`中的内容。
- AddReplaceValue
  - 设置标记对应的值
- GetReplaceValue
  - 获取标记对应的值

IRepeatDataContext

```
public interface IRepeatDataContext
{
    IDataContext CreateNextIteratorDataContext();

    IDataContext[] DataContexts { get; }

    IDataContext GetDataContext(int index);
}
```

- CreateNextIteratorDataContext
  - 创建一轮迭代的数据上下文
- DataContexts
  - 获取所有创建在内的数据上下文数组
- GetDataContext
  - 获取索引对应的数据上下文（不常用）

使用方法

在我们的Demo示例中，展示了上面每个语句的使用方法。以HelloWorld为例子。

首先，定义模板文件

```
using UnityEngine;

namespace VTCodeTemplateDemo
{
    public class HelloWorld
    {
        public void Say()
        {
            Debug.Log("$Content$");
        }
    }
}
```

然后，定义Editor方法，用来生成目标文件，需要注意填写正确的路径：

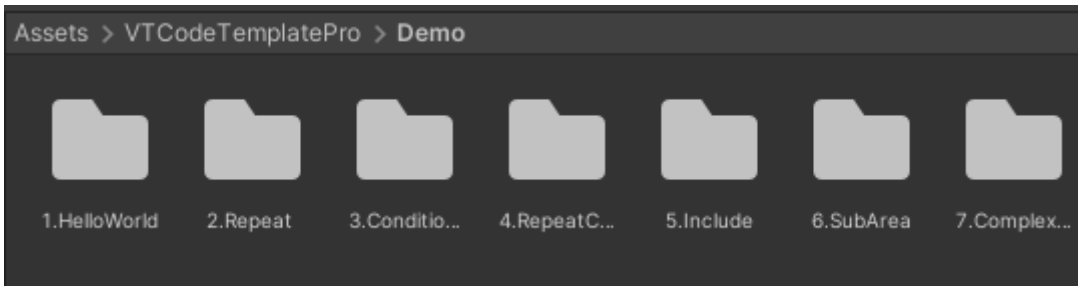
```
using UnityEditor;
using VTCodeTemplate.Core;

namespace VTCodeTemplateDemo
{
    0 个引用
    public class HelloWorldGen
    {
        [MenuItem("Tools/VTCodeTemplate/Demo/Gen HelloWorld", priority = 0)]
        0 个引用
        public static void GenHelloWorld()
        {
            TemplateHandler tempalteHandler = new TemplateHandler("Assets/VTCodeTemplatePro/Demo/1.HelloWorld/HelloWorld.vtemplate");

            IDataContext globalDataContext = tempalteHandler.GlobalDataContext;
            globalDataContext.AddReplaceValue("Content", "Hello World");

            tempalteHandler.Compile("Assets/VTCodeTemplatePro/Demo/1.HelloWorld/Generate/HelloWorld.cs");
        }
    }
}
```

在上面的代码中，使用全局数据上下文来设置需要替换的值。

最后执行`Compile`方法，生成目标文件到指定路径。

Demo介绍

在`VTCodeTemplatePro/Demo`中存放了上面所有语法的使用示例。



里面的每一个编辑器代码对应到`Tools/VTCodeTemplate/Demo`的菜单。