# CodeDom

CodeDOM 提供了表示许多常见的源代码元素类型的类型。您可以设计一个生成源代码模型的程序，使用 CodeDOM 元素构成一个对象图。可以使用受支持的编程语言的 CodeDOM 代码生成器，将该对象图呈现为源代码。CodeDOM 也可以用于将源代码编译成二进制程序集。

CodeDOM 的一些一般用途包括：

* 模板化代码生成：生成 ASP.NET、XML Web 服务客户端代理、代码向导、设计器或其他代码发出机制的代码。
* 动态编译：支持以一种或多种语言进行代码编译。

Microsoft®.NET Framework 中用 CodeDOM 以一种语言中立的方式来表示源代码文档。通过操作和使用 CodeDOM 对象图可以生成源代码，甚至生成编译后的程序集，并且可以多次使用。System.CodeDom 命名空间包括 70 多种表示典型的面向对象的高级语言（例如 C#、Visual Basic® .NET、Eiffel 或 Java 等}的特性实体的定义。System.CodeDom.Compiler 命名空间定义了处理 CodeDOM 对象图抽象类的框架，通过实现 CodeDomProvider 基类，希望支持 CodeDOM 的任何语言都可以做到这一点。

在 .NET Framework 和 Visual Studio® .NET 中，CodeDOM 起着重要的幕后作用。ASP.NET 还用它来生成可执行代码，实际上为了在 ASP.NET 页面中使用它，用某种语言实现 CodeDomProvider 是必需的。CodeDOM 还用于 Web 服务描述语言代理生成、以及 Visual Studio .NET 的设计器和代码向导。许多语言都可以实现 CodeDomProvider 类，因而可用来处理 CodeDOM 对象图。可以预想，通过在 Microsoft.CSharp 和 Microsoft.VisualBasic 命名空间中分别定义的 CSharpCodeProvider 和 VBCodeProvider 类，C# 与 Visual Basic .NET 可以实现该类。另外，Microsoft.JScript 命名空间和 Visual J# 中还有 JscriptCodeProvider。.NET 公开了 Microsoft.VJSharp™ 命名空间中的 VJSharpCodeProvider。除此之外，许多 Microsoft .NET 语言合作伙伴都已经或者正在实现 CodeDomProvider 类，作为其语言与 .NET 集成的一部分。其中包括 Interactive Software Engineering 的 Eiffel、由 ActiveState 开发的 ActivePERL 以及 Fujitsu 开发的 COBOL 语言。

简单浏览一下 CodeDOM 之后，对象模型的一些实际应用就变得显而易见。我发现从中可以受益匪浅、同时也是本文关注的一个主要的焦点 — 就是使用 CodeDOM 创建常见的、可重用的源代码模板。这并不是个新概念。许多书、文章、语言特性和代码生成向导都讨论了这个主题。不管怎样，我觉得还是想赞美一下用 CodeDOM 来实现这些目标的优点和益处。花费一定时间了解一下相对简单的（尽管大且麻烦）对象模型，就有可能用 CodeDOM 来创建模板。另外，由于大量语言已经实现了 CodeDomProviders，因此可以用创建的模板来生成这些语言的代码，将您的模板的实用性提供给不使用您的语言的其他编程人员。

.NET Framework 中包含一个名为“代码文档对象模型”(CodeDOM) 的机制，该机制使编写源代码的程序的开发人员可以在运行时，根据表示所呈现代码的单一模型，用多种编程语言生成源代码。

为表示源代码，CodeDOM 元素相互链接以形成一个数据结构（称为 CodeDOM 图），它以某种源代码的结构为模型。

System.CodeDom 命名空间定义可以表示源代码的逻辑结构（与具体的编程语言无关）的类型。System.CodeDom.Compiler 命名空间定义从 CodeDOM 图生成源代码的类型，和在受支持的语言中管理源代码编译的类型。编译器供应商或开发人员可以扩展受支持语言的集合。

           //生成CodeDOM图，这一步是最复杂的部分，后面生成代码与编译都是以这里的东西为蓝本

       private static CodeCompileUnit ComplieUnit() {

             //生成一个可编译的单元，这是最根部的东西

             CodeCompileUnit compunit = new CodeCompileUnit();

             CodeNamespace sample = new CodeNamespace("Sample");//定义一个名为Sample的命名空间

             CodeTypeDeclaration MyClass = new CodeTypeDeclaration("DemoClass");//定义一个名为DemoClass的类

             CodeEntryPointMethod Start = new CodeEntryPointMethod();//定义程序入口点，就是Main()

             //下面两句产生调用方法的语句

             CodeMethodInvokeExpression cs = new CodeMethodInvokeExpression(new CodeTypeReferenceExpression("System.Console"),

                  "WriteLine",new CodePrimitiveExpression("Hello World!"));//这句会产生如下的C#代码 System.Console.WriteLine("Hello World!");

             CodeMethodInvokeExpression wt = new CodeMethodInvokeExpression(new CodeTypeReferenceExpression("System.Console"),

                  "Read");//这句是 System.Console.Read();

             //下面一系列语句把上述定义好的元素联接起来

             compunit.Namespaces.Add(sample);

             sample.Imports.Add(new CodeNamespaceImport("System"));//导入System命名空间

             sample.Types.Add(MyClass);

             MyClass.Members.Add(Start);

             Start.Statements.Add(cs);

             Start.Statements.Add(wt);

             return compunit;

           }

CompileUnit之后根本不用再生成源代码，直接用CodeDom编译就可以了

另外将需要的namespace引用了之后，就可以直接使用CodeSnippetExpression来写连续的代码块了

**System.Object**

**System.CodeDom.CodeObject**

      System.CodeDom.CodeComment//注释

      System.CodeDom.CodeCompileUnit//编译单元，整个CodeDOM的根

      System.CodeDom.CodeExpression

      System.CodeDom.CodeNamespace//命名空间

      System.CodeDom.CodeNamespaceImport//命名空间导入

      System.CodeDom.CodeStatement

      System.CodeDom.CodeTypeMember

      System.CodeDom.CodeTypeReference//类型引用

 CodeExpression顾名思义是代码表达式,如：CodeCastExpression(强制类型转换)、CodeBaseReferenceExpression(基类的引用)、CodeTypeOfExpression(typeof()表达式)、CodeMethodInvokeExpression(调用方法的表达式)等。

CodeStatement直接译就是代码语句.语句显然比表达式要复杂一点，比如一个简单的赋值语句，其左边与右边本身就是一个表达式，如左边是一个变量引用表达式，右边可能是一个调用方法的表达式等等(它有一个构造函数就是如下 这样的public CodeAssignStatement(CodeExpression, CodeExpression))。这还是比较简单的，复杂的如那种for循环语句的构造函数里有四个部分要设置，每一部分又是表达式，又是语句，语句组。

CodeDOM最管用的应该是下面这个CodeTypeMember了。这个东西把类里面的字段、方法、属性、事件、构造函数等一网打尽。见下：

System.Object

   System.CodeDom.CodeObject

**System.CodeDom.CodeTypeMember**

         System.CodeDom.CodeMemberEvent

         System.CodeDom.CodeMemberField

         System.CodeDom.CodeMemberMethod

         System.CodeDom.CodeMemberProperty

         System.CodeDom.CodeSnippetTypeMember

         System.CodeDom.CodeTypeDeclaration

由CodeTypeMember派生出来的类里面最重要的是CodeMemberMethod ，它增加了一些比较重要的属性，比如参数声明、返回值类型等。还有就是Statements属性了，就是该方法的语句集合喽，它的类型是CodeStatementCollection

CodeDOM的逻辑结构基本的路子就是：

1、由一些CodeExpression组成简单的CodeStatement

2、复杂点的CodeStatement再由CodeExpression和CodeStatement(数组)组成(一个for循环或一个if组语句(包括else部分)组成的就是一个大的CodeStatement；异常也是，从try起到catch、finally结束的部分就是一个大的CodeStatement)

3、大的CodeStatement与一些小的不能组合的CodeStatement(或CodeExpression)依序加到CodeMemberMethod的Statements里。

4、CodeMemberMethod与CodeMemberEvent、CodeMemberField等一道加到CodeTypeDeclaration里

5、把CodeTypeDeclaration加到CodeNamespace里

6、把CodeNamespace加到CodeCompileUnit里

用这个CodeCompileUnit可以生成完整的文件了可以编译的

CodeDOM这个东西虽然能生成大多数的程序代码，但它还是有点局限性的, 某些我们用到的语言选项可能无法支持.