CodeSmith使用教程

极客学院出版

前言

CodeSmith 是一种基于模板的代码生成工具，它使用类似于 ASP.NET的语法来生成任意类型的代码或文本。

适用人群

前面基本介绍了CodeSmith的基本用法和编写代码模板的基本方法，这只是CodeSmith功能的一部分，其它部分可以参考CodeSmith文档和类文件定义。此外可以参考CodeSmith附带的示例模板。

> 鸣谢：引路蜂移动软件

目录

前言 1

第 1 章 概述 4

第 2 章 编写第一个代码模板 11

第 3 章 自动生成Yii Framework ActiveRecord类简单模板 20

第 4 章 基本语法-CodeTemplate 指令 27

第 5 章 基本语法-使用注释 30

第 6 章 基本语法-声明和使用属性 32

第 7 章 基本语法-转义Asp.Net标记 36

第 8 章 CodeTemplate 对象 38

. 40

. 40

. 40

. 40

. 40

第 9 章 Progress 对象 46

第 10 章 CodeTemplateInfo 对象 49

第 11 章 引用其它文件或 .Net 类库 51

第 12 章 使用主从代码模板 53

. 40

. 40

. 40

. 40

第 13 章　调试 59
第 14 章　使用 SchemaExplorer 来获取数据库定义 62
第 15 章　为 Yii Framework 创建生成 ActiveRecord 的代码模板 69
第 16 章　使用 XMLProperty. 81
第 17 章　Merge 策略 87
. 40
. 40

1

概述

前面正在介绍 hibernate 的开发教程，提到 hibernate 在 .Net 平台上相应的 ORM 工具为NHibernate，使用 NHibernate 就不能不提到 CodeSmith。

CodeSmith 是一种基于模板的代码生成工具，它使用类似于 ASP.NET 的语法来生成任意类型的代码或文本。与其他许多代码生成工具不同，CodeSmith 不要求您订阅特定的应用程序设计或体系结构。使用 CodeSmith，可以生成包括简单的强类型集合和完整应用程序在内的任何东西。当您生成应用程序时，您经常需要重复完成某些特定的任务，例如编写数据访问代码或者生成自定义集合。CodeSmith 在这些时候特别有用，因为您可以编写模板自动完成这些任务，从而不仅提高您的工作效率，而且能够自动完成那些最为乏味的任务。CodeSmith 附带了许多模板，包括对应于所有 .NET 集合类型的模板以及用于生成存储过程的模板，但该工具的真正威力在于能够创建自定义模板。

CodeSmith 可以从网站 http://www.codesmithtools.com/ 下载，个人开发版费用在300美元左右，个人认为还是物有所值。

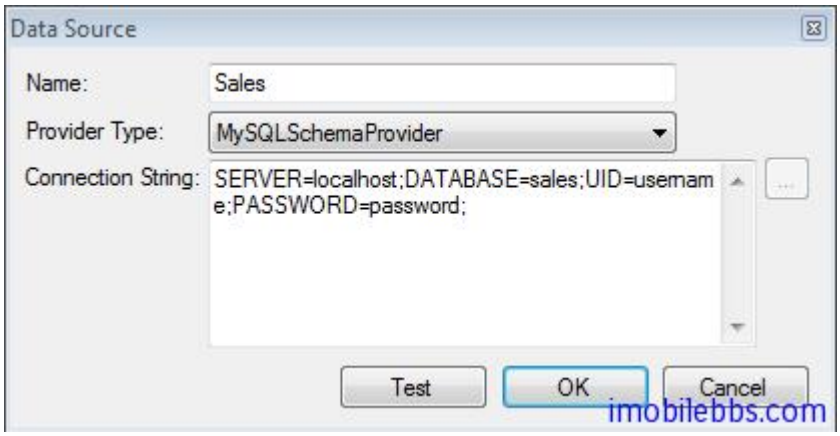
还是用一个例子来说明一下使用 CodeSmith 可以大大减轻程序代码的工作量，对于数据库应用来说，尽管数据库表不尽相同，但基本过程都是定义数据库表，设计表对应的类，然后使用 ADO 或是 SQL 语句来访问数据库，创建对应的类对象等。

使用 CodeSmith 提供的模板，可以几乎不用手工编写一行代码，就可以自动生成上述数据库相关的代码。

本例使用 Visual Studio 2010 ，Codesmith 安装时提供了 Visual Studio 插件支持。使用的示例数据库也是 Sales，可以参见 Hibernate 开发教程(2):准备开始。

由于使用 MySQL 数据库，需要下载 MySQL .Net 库。使用 SQL Server 可以直接使用。

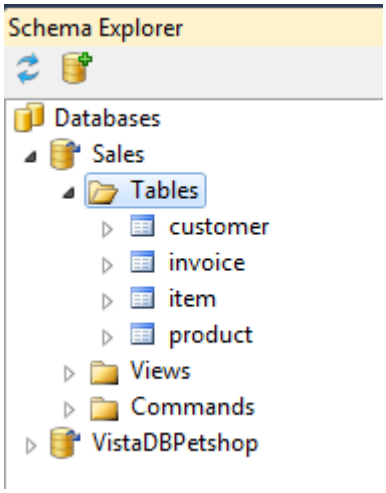
1. 创建一个命令行应用 solution。
2. 使用 CodeSmith 的 Schema Explorer 添加一个 MySQL 数据源



图片 1.1 第1张

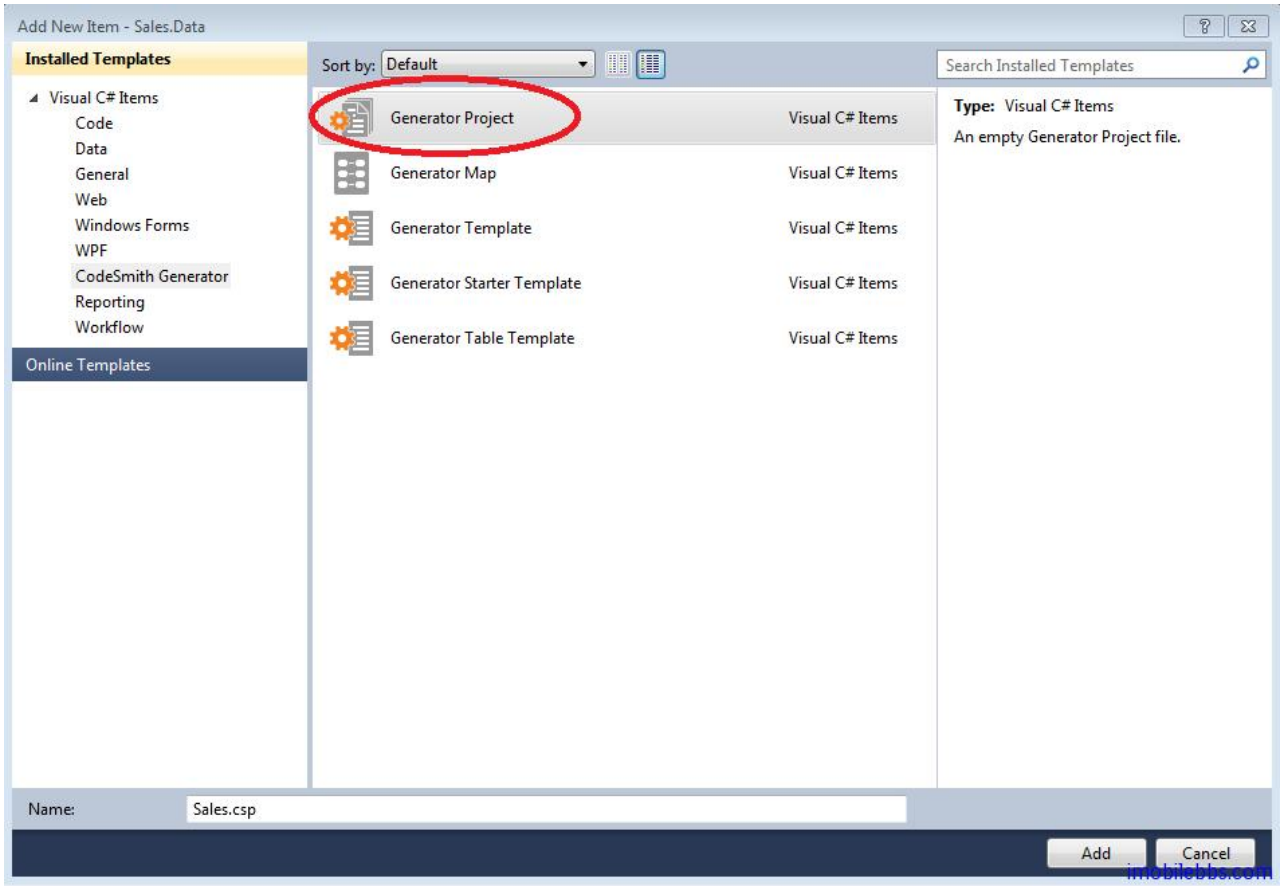
这里的 Connection String 为： SERVER=localhost;DATABASE=sales;UID=username;PASSWORD=password;（根据你自己服务器自行修改参数）

添加成功后，在 Schema Explorer 中会显示所连接的数据库的表定义等



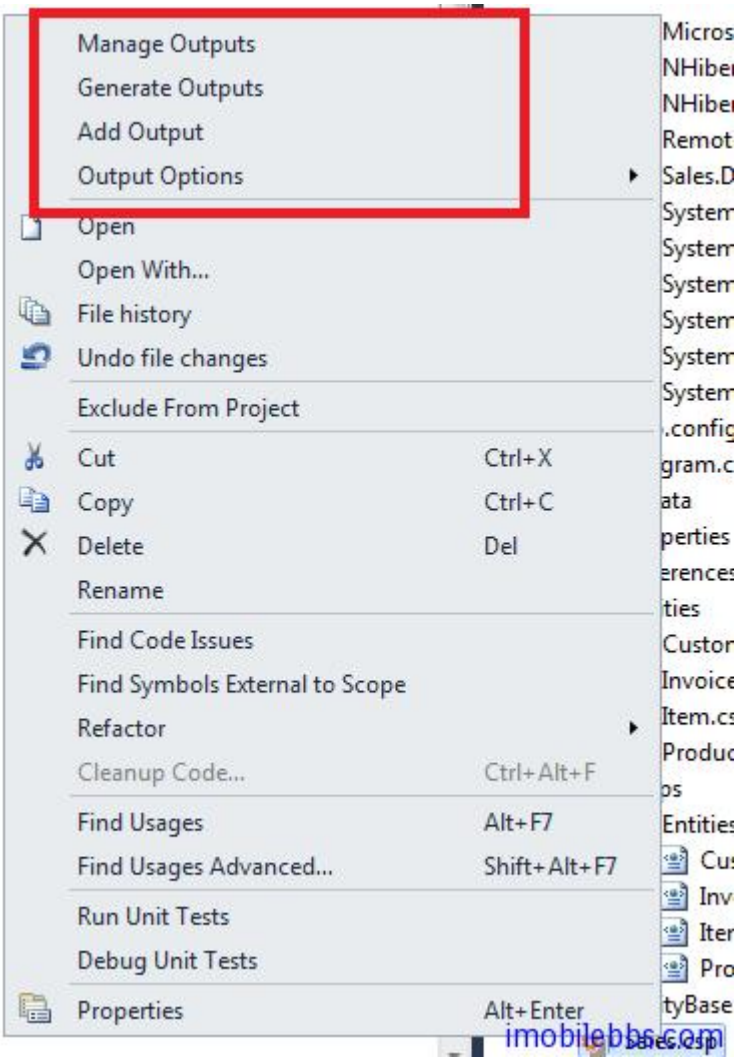
图片 1.2 第2张

1. 在这个 solution 中添加一个 Sales.Data Class Library.然后在项目中添加一个 CodeSmith 项目 Item



图片 1.3 第3张

1. 点击 Sales.csp 使用鼠标右键 Context Menu

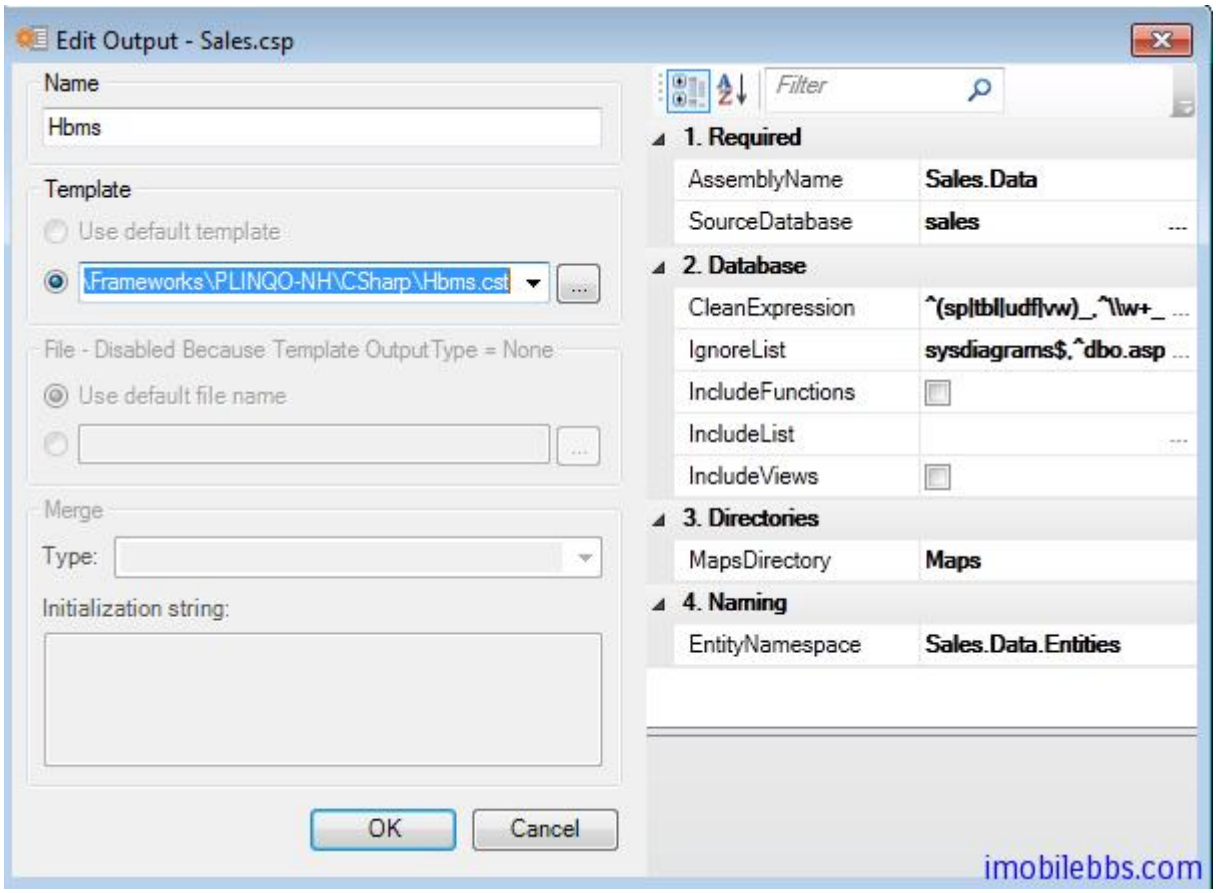


图片 1.4 第4张

使用有 Add Output 可以在项目中添加生成代码的模板，这里选用 CodeSmith 自带的 PLINQO-NH\CSharp 下的三个模板，

模板路径为..\Users\…\Documents\CodeSmith Generator\Samples\v6.5\Templates\Frameworks\PLINQO-NH\CSharp\

分别添加三个模板，SourceDatabase 选择 Schema Explorer 中添加的 Sales 数据库，其它属性使用缺省值。



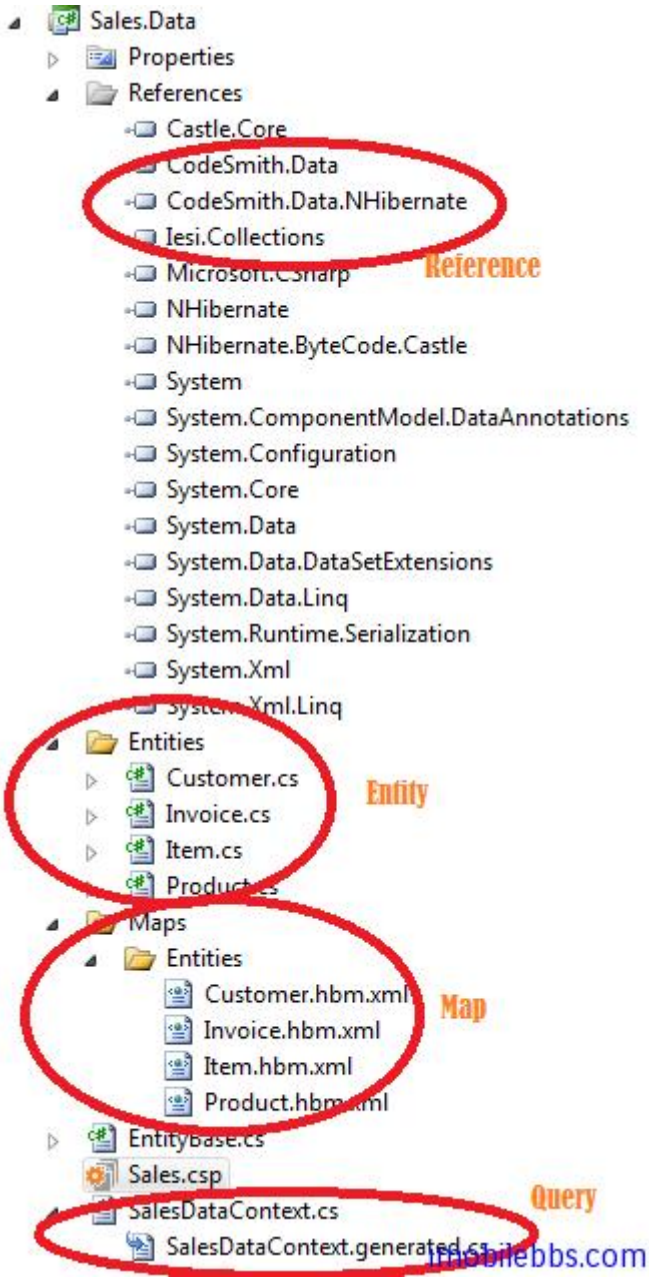
图片 1.5 第5张

三个模板分别为

- Hbms 生成 hbm.xml 映射文件
- Entities 生成和数据库表对应的.Net类定义
- Queries 生成查询数据对应的类

然后通过 Sales.csp 的 Generate code 生成代码。

可以看到 CodeSmith 自动生成了很多代码，并添加了所需的引用。



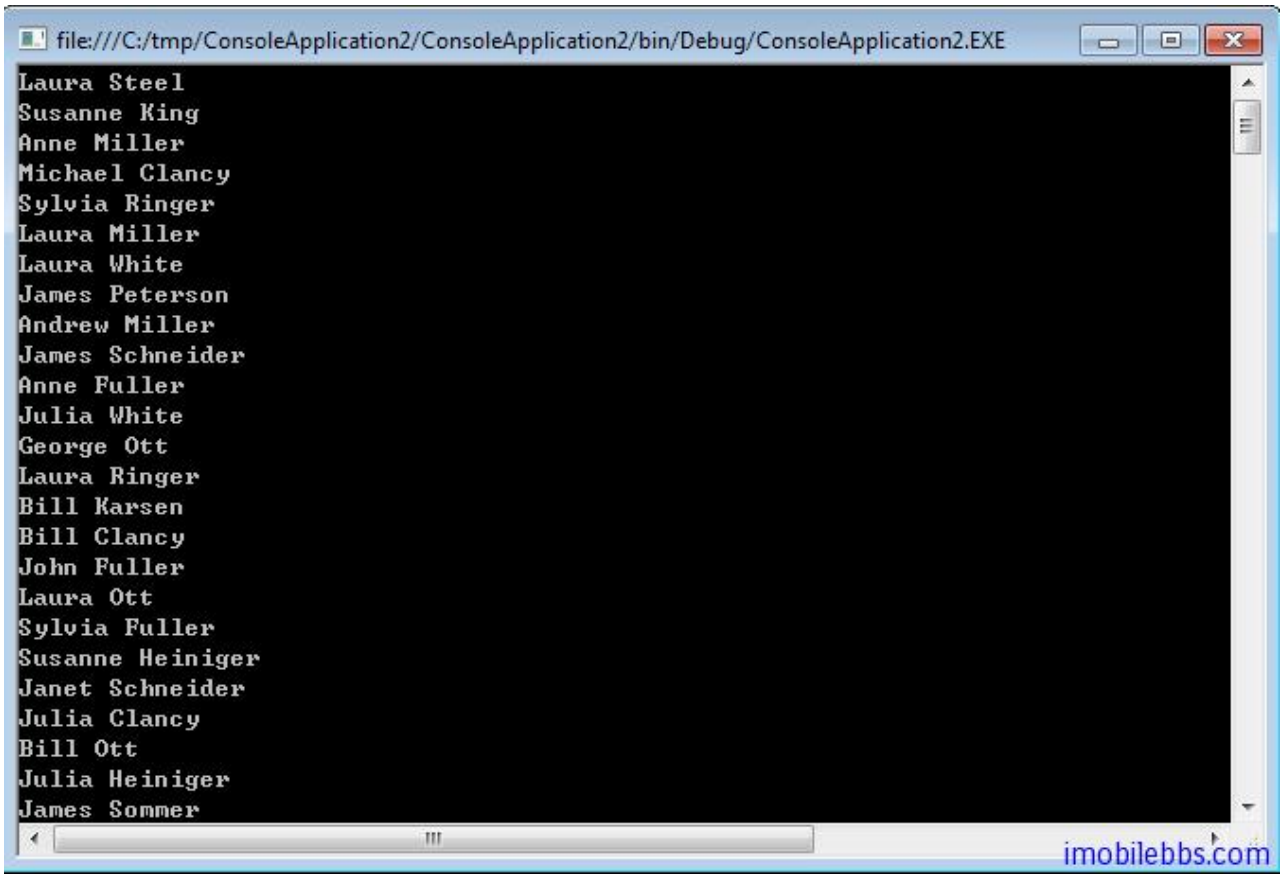
图片 1.6 第6张

此时用来访问数据的类全部由 CodeSmith 生成，无需自己写一行代码。

1. 看看如果使用生成的代码来访问数据库，修改命令行应用的 Program.cs 的 main 函数，打印出所有 Customer 的姓名。

```
var salesDataContext = new SalesDataContext();
foreach (var s in salesDataContext.Customer.ToList())
{
   Console.WriteLine(s.FirstName+" " + s.LastName);
}
```

使用 SalesDataContext 中 Customer 查询对象。然后枚举列表中每个 Customer 对象并打印出FirstName 和 Last Name.可以看到代码自动完成了对数据库的读取访问。



图片 1.7 第7张

由本例看到使用 CodeSmith 可以大大减轻手工代码量，其使用的一般步骤是

1. 选择使用合适的模板，CodeSmith 随开发包自带了大量常用的模板，如果找不到合适的模板，CodeSmith 支持自定义模板。
2. 为模板选择合适的参数设置。
3. 自动生成代码（可以为任意类型的代码，C#，Java, .XML 文本等）

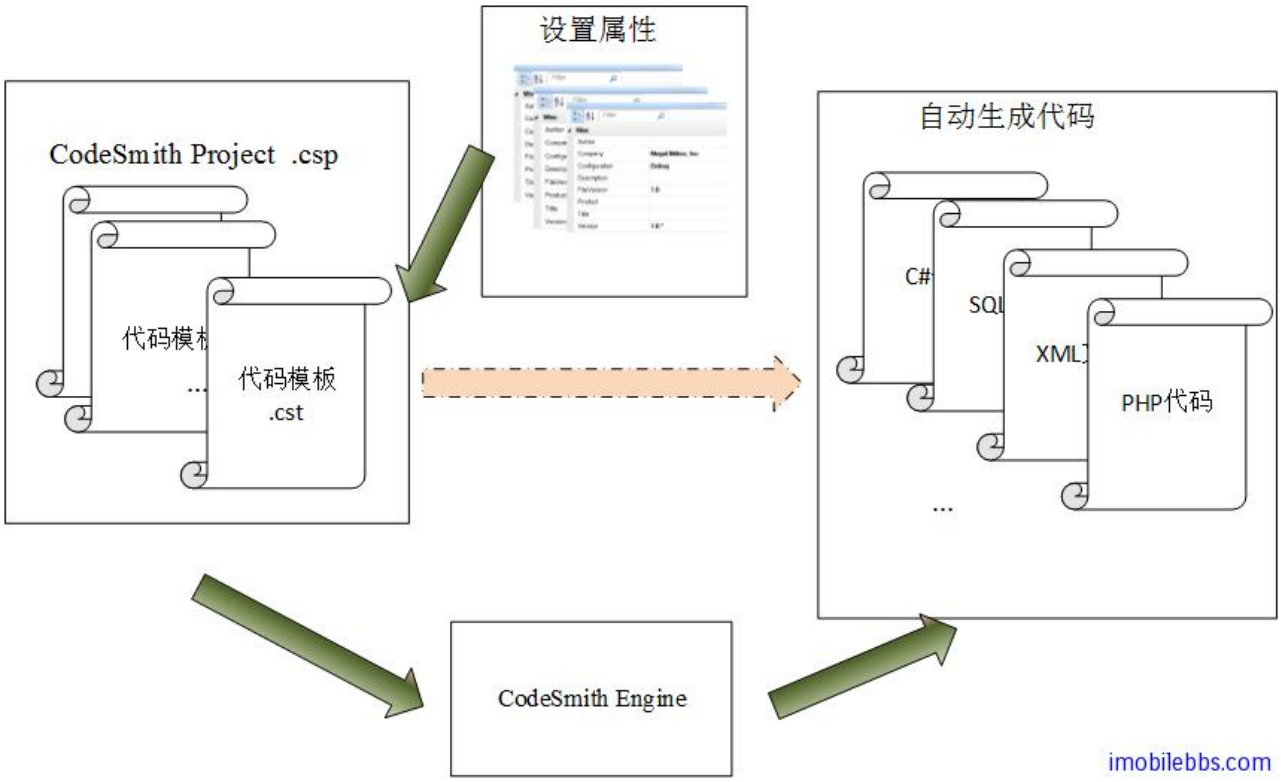
后面将详细介绍 CodeSmith 使用的基本方法，CodeSmith 的核心为模板，因此重点在模板的设计和使用。

2

编写第一个代码模板

在CodeSmith 使用教程(1): 概述我们通过使用 CodeSmith 从数据库自动生成 NHiberate 代码，可以了解到使用 CodeSmith 自动生成代码的基本步骤：

1. 选择使用合适的模板，CodeSmith 随开发包自带了大量常用的模板，如果找不到合适的模板，CodeSmith 支持自定义模板。
2. 为模板选择合适的参数设置。
3. 自动生成代码（可以为任意类型的代码，C#，Java, .XML 文本等）



图片 2.1 第8张

其核心为代码模板文件，随 CodeSmith 自带了不少常用的模板，可以通过模板浏览器来查询，此外网上也有很多第三方开发的模板，在使用前可以先查查是否已有现成的模板，或是可以通过修改现有的模板来完成自动生成代码的需要。

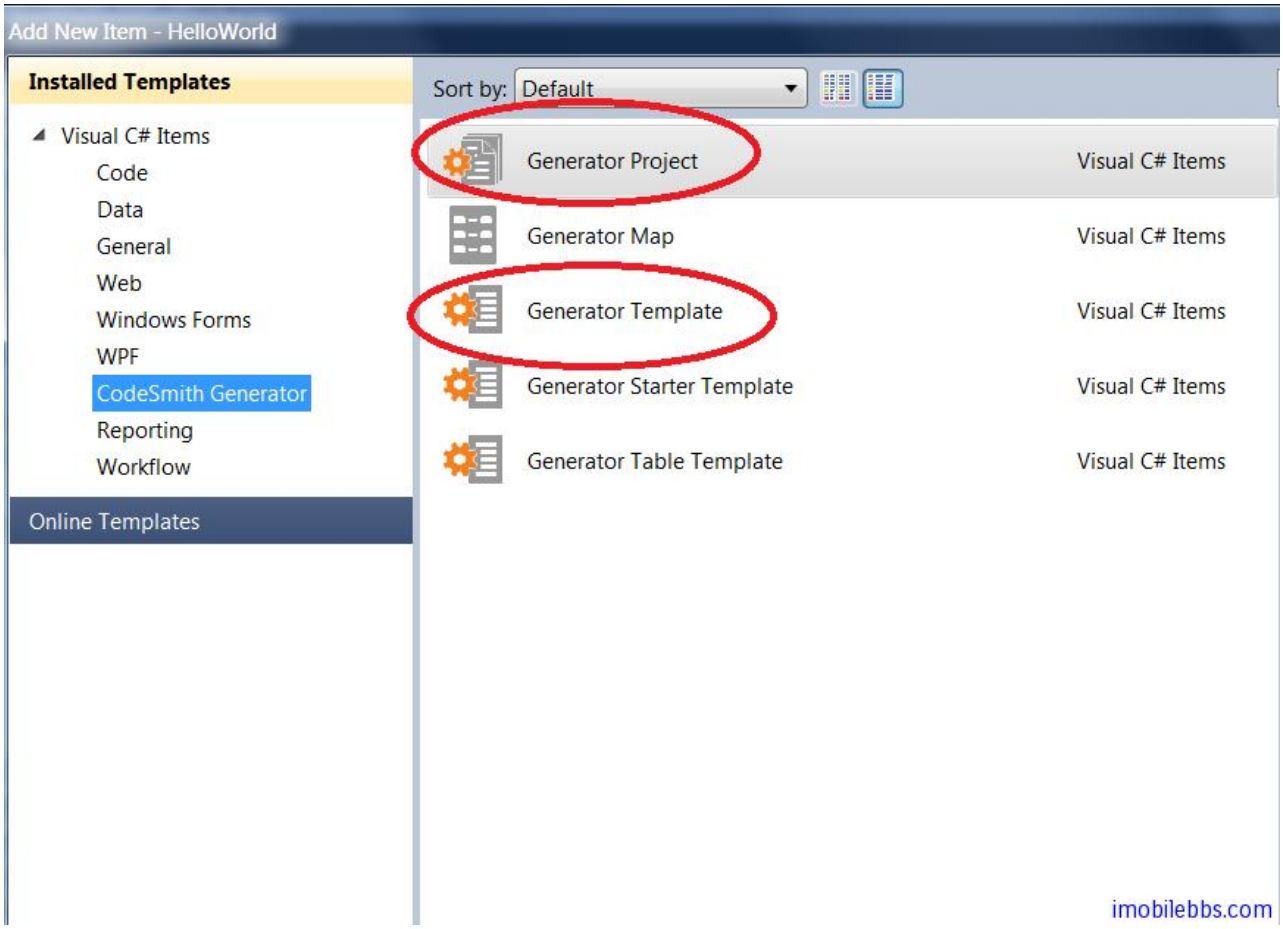
在开发应用时，很多人都喜欢通过复制以前的项目中的代码，然后通过修改以满足新项目，这些重用的代码通常具有很多共性（可以想想 C++ 的模板类，C# 的 Generic 等），CodeSmith 就是用来为这些具有相似性的代码创建模板，然后通过设置属性（代码直接的不同点），就可以自动创建所需代码。

本例通过一个简单的例子来介绍创建一个自定义代码模板的方法。CodeSmith 提供了 Visual Studio的集成开发环境的支持，本例也是通过创建模板自动生成简化每个的 C# 项目都需要的 AssemblyInfo.cs，在开发 C# 应用时，一般是通过手工修改 AssemblyInfo.cs 的中属性（或者是 Copy & Paste： -））.

首先我们使用 Visual Studio 创建一个 C# HelloWorld 下面（Console 或是 WinForm 项目都可以），可以打开项目中的 AssemblyInfo.cs

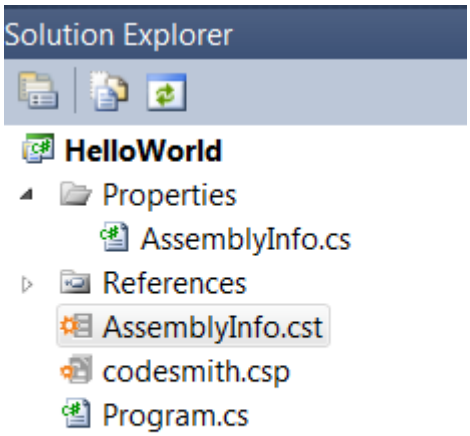
```
using System.Reflection;
using System.Runtime.CompilerServices;
using System.Runtime.InteropServices;

// General Information about an assembly is controlled through the following
// set of attributes. Change these attribute values to modify the information
// associated with an assembly.
[assembly: AssemblyTitle("HelloWorld")]
[assembly: AssemblyDescription("")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyCompany("Microsoft")]
[assembly: AssemblyProduct("HelloWorld")]
[assembly: AssemblyCopyright("Copyright © Microsoft 2013")]
[assembly: AssemblyTrademark("")]
[assembly: AssemblyCulture("")]

// Setting ComVisible to false makes the types in this assembly not visible
// to COM components.  If you need to access a type in this assembly from
// COM, set the ComVisible attribute to true on that type.
[assembly: ComVisible(false)]

// The following GUID is for the ID of the typelib if this project is exposed to COM
[assembly: Guid("72797715-64b9-4bab-a49f-f55e8a0a18d7")]

// Version information for an assembly consists of the following four values:
//
//      Major Version
//      Minor Version
//      Build Number
//      Revision
//
// You can specify all the values or you can default the Build and Revision Numbers
// by using the '*' as shown below:
// [assembly: AssemblyVersion("1.0.*")]
[assembly: AssemblyVersion("1.0.0.0")]
[assembly: AssemblyFileVersion("1.0.0.0")]
```

为了使用 CodeSmith，我们在 HelloWorld 中添加 CodeSmith 的项目文件并创建一个模板文件AssemblyInfo.cst



图片 2.2 第9张

创建好的项目文件如下：



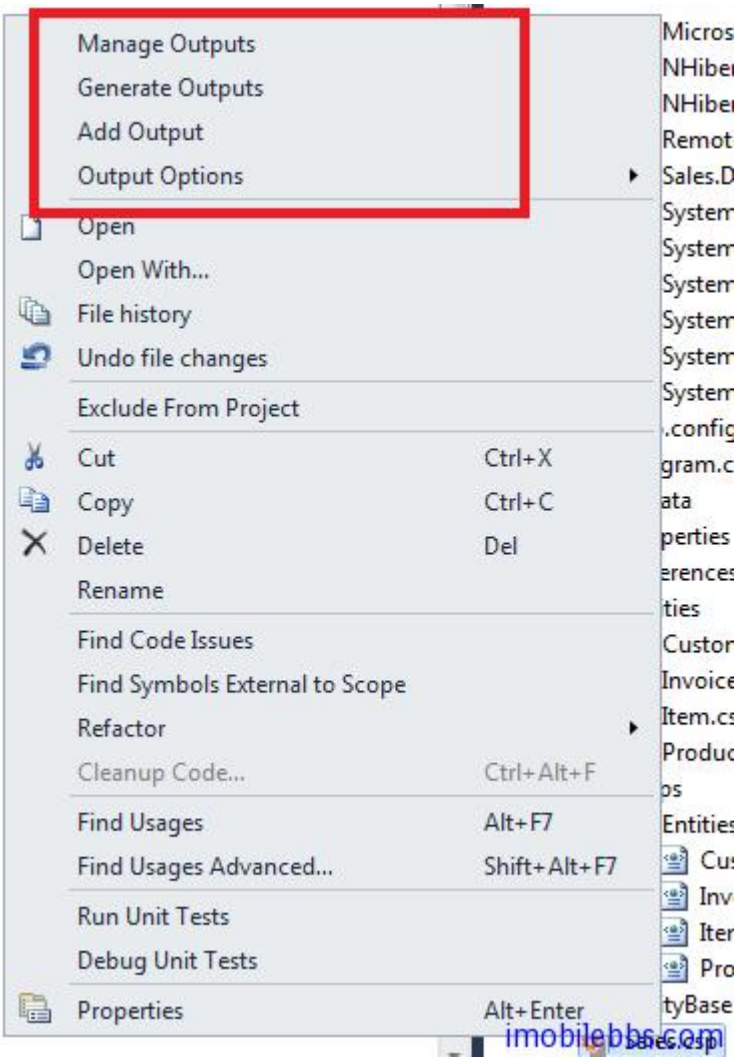
图片 2.3 第10张

编写 CodeSmith 的代码模板和编写 Asp.Net 的 Page 非常类似，CodeSmith 支持以 C#，VB.Net和 JavaScript 做为脚本语言来编写模板，本例使用 C# 做为脚本语言（源代码/语言），计划生成的也是 C# 语言（目标代码/语言），打开 AssemblyInfo.cst，修改代码为

```
<%@ CodeTemplate Language="C#" TargetLanguage="C#" Description="Create an AssemblyInfo.cs file." %>
```

每个 CodeSmith 的代码模板都是以 CodeTemplate 开始，定义代码模板使用的源语言，目标语言和简单的描述。

然后将这个模板添加到 CodeSmith 项目中，可以右键单击 codesmith.csp ,选择 Add output



图片 2.4 第11张

这时 CodeSmith 的项目将创建好了，但单击”Generate code”不会生成任何代码，因为我们的代码模板 AssemblyInfo.cst 没做任何事。

创建代码模板可以从生成的结果开始，可以直接先把要生成的代码复制到代码模板 AssemblyInfo.cst中，比如：

```
using System.Reflection;
using System.Runtime.CompilerServices;
//
// Created: 1/1/2013
// Author: James Shen
//
[assembly: AssemblyTitle("User storage utility")]
[assembly: AssemblyDescription("Helps manage data in Isolated Storage files.")]
[assembly: AssemblyConfiguration("Retail")]
[assembly: AssemblyCompany("Guidebee Pty Ltd, Inc.")]
[assembly: AssemblyProduct("StorageScan")]
[assembly: AssemblyCopyright("Copyright (c) Guidebee Pty Ltd.")]
[assembly: AssemblyCulture("")]
[assembly: AssemblyVersion("1.0.*")]
[assembly: AssemblyFileVersion("1.0")]
[assembly: AssemblyDelaySign(true)]
```

可以把要生成的代码模板的内容分成三部分：

- 固定内容
- 可以通过代码动态生成的部分（如上面的日期）
- 需要用户提供属性配置的部分

此时如果使用 Codesmith 的 Generate Codes, 将自动生成 AssemblyInfo.cs (缺省为模板名），不过 AssemblyInfo.cs 位置不是我们所需的 Properties/AssemblyInfo.cs, 这可以通过重载代码模板的 GetFileName 方法来实现：

```
<%@ CodeTemplate Language="C#" TargetLanguage="C#"
  Description="Create an AssemblyInfo.cs file." %>
...
<script runat="template">
public override string GetFileName() {
    return "Properties/AssemblyInfo.cs";
}
</script>
```

这样在使用 CodeSmith 项目的 Generate Codes，就自动覆盖原来的 Properties/AssemblyInfo.cs 文件。内容就是模板中的代码部分。

但每次生成的代码都是固定的，作为模板来说没有什么灵活性，下面我们可以通过检查模板的内容，觉定那些内容是可变的。比如 AssemblyInfo.cs 的日期和 Assembly 的各个属性对于不同的项目来说是可变的。

这些可变的内容其中一部分可以通过代码自动生成（如日期），有一部分需要用户来配置，比如AssemblyTitle，AssemblyDescription 等。

对于日期部分可以通过C#代码实现如下：

```
// Created: <%= DateTime.Now.ToLongDateString() %>
```

可以看出来 CodeSmith 的模板文件如 AssemblyInfo.cst 和 Asp.Net 的 Page 文件中功能是非常类似，可以通过<%= 和%>直接嵌入 C# 代码（或 VB.Net，JavaScripts)。

对于属性来说，可以通过先定义属性：

```
<%@ Property Name="Author" Type="System.String" Description="Lead author of the project." %>
<%@ Property Name="Title" Type="System.String" Description="Title of the project." %>
<%@ Property Name="Description" Type="System.String" Description="Description of the project." %>
<%@ Property Name="Configuration" Type="System.String" Default="Debug" Description="Project configuration." %>
<%@ Property Name="Company" Type="System.String" Default="Guidebee Pty Ltd." %>
<%@ Property Name="Product" Type="System.String" Description="Product Name." %>
<%@ Property Name="Version" Type="System.String" Default="1.0.*" Description=".NET assembly version." %>
<%@ Property Name="FileVersion" Type="System.String" Default="1.0" Description="Win32 file version." %>
```

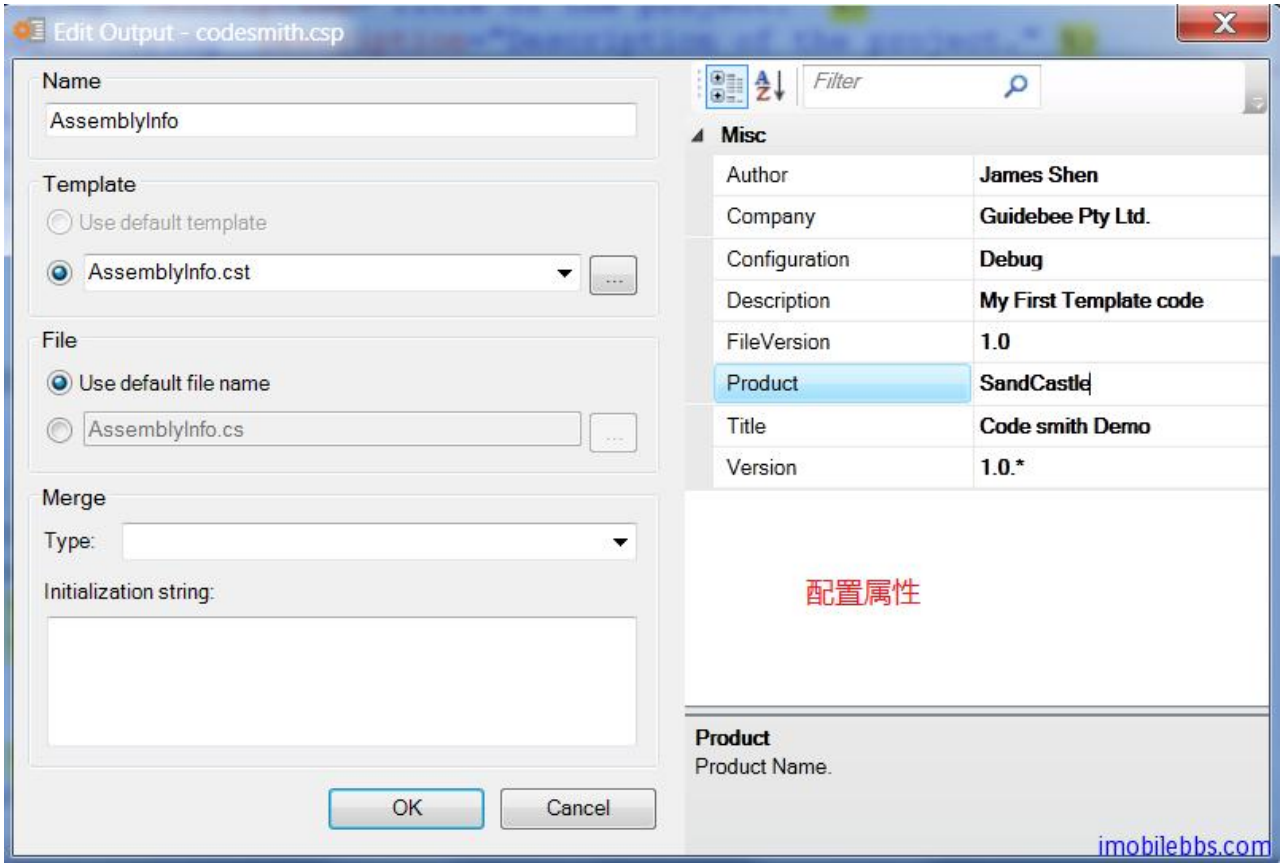
属性定义通过 Property 定义，Name 定义属性名，Type 为属性的数据类型，Default 定义属性的缺省值， Description 可以定义属性的作用及说明。

然后就可以在 C# 代码中使用这些属性，完整的代码模板如下：

```
<%@ CodeTemplate Language="C#" TargetLanguage="C#" Description="Create an AssemblyInfo.cs file." %>
<%@ Property Name="Author" Type="System.String" Description="Lead author of the project." %>
<%@ Property Name="Title" Type="System.String" Description="Title of the project." %>
<%@ Property Name="Description" Type="System.String" Description="Description of the project." %>
<%@ Property Name="Configuration" Type="System.String" Default="Debug" Description="Project configuration." %>
<%@ Property Name="Company" Type="System.String" Default="Guidebee Pty Ltd." %>
<%@ Property Name="Product" Type="System.String" Description="Product Name." %>
<%@ Property Name="Version" Type="System.String" Default="1.0.*" Description=".NET assembly version." %>
<%@ Property Name="FileVersion" Type="System.String" Default="1.0" Description="Win32 file version." %>
using System.Reflection;
using System.Runtime.CompilerServices;
//
// Created: <%= DateTime.Now.ToLongDateString() %>
// Author:  <%= Author %>
//
[assembly: AssemblyTitle("<%= Title %>")]
[assembly: AssemblyDescription("<%= Description %>")]
[assembly: AssemblyConfiguration("<%= Configuration %>")]
[assembly: AssemblyCompany("<%= Company %>")]
[assembly: AssemblyProduct("<%= Product %>")]
[assembly: AssemblyCopyright("Copyright (c) <%= DateTime.Now.Year.ToString() %> <%= Company %>")]
[assembly: AssemblyCulture("")]
```

```
[assembly: AssemblyVersion("<%= Version %>")]
[assembly: AssemblyFileVersion("<%= FileVersion %>")]
[assembly: AssemblyDelaySign(true)]
```

此时如果需要“Generate output” 首先要配置代码模板的属性，这通过”Manage output” 来完成，



图片 2.5 第12张

次数如果打开 codesmith.csp 文件可以看到为 AssemblyInfo.cst 配置的属性内容：

```
<?xml version="1.0" encoding="utf-8"?>
<codeSmith xmlns="http://www.codesmithtools.com/schema/csp.xsd">
  <propertySets>
    <propertySet name="AssemblyInfo" template="AssemblyInfo.cst">
      <property name="Configuration">Debug</property>
      <property name="Company">Guidebee Pty Ltd.</property>
      <property name="Version">1.0.*</property>
      <property name="FileVersion">1.0</property>
      <property name="Author">James Shen</property>
      <property name="Title">Code smith Demo</property>
      <property name="Description">My First Template code</property>
      <property name="Product">SandCastle</property>
    </propertySet>
```

```
  </propertySets>
</codeSmith>
```

生成代码如下：

```
using System.Reflection;
using System.Runtime.CompilerServices;
//
// Created: Thursday, 3 January 2013
// Author:  James Shen
//
[assembly: AssemblyTitle("Code smith Demo")]
[assembly: AssemblyDescription("My First Template code")]
[assembly: AssemblyConfiguration("Debug")]
[assembly: AssemblyCompany("Guidebee Pty Ltd.")]
[assembly: AssemblyProduct("SandCastle")]
[assembly: AssemblyCopyright("Copyright (c) 2013 Guidebee Pty Ltd.")]
[assembly: AssemblyCulture("")]
[assembly: AssemblyVersion("1.0.*")]
[assembly: AssemblyFileVersion("1.0")]
[assembly: AssemblyDelaySign(true)]
```

本例下载

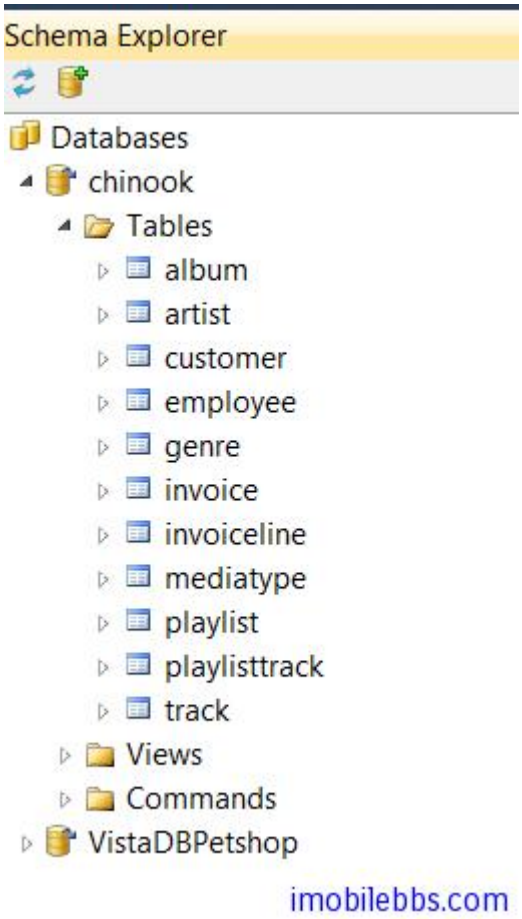
3

自动生成Yii Framework ActiveRecord类简单模板

上例介绍了使用 CodeSmith 编写代码模板的基本方法，本例实现一个较为实用的代码模板，通过数据库自动为 Yii Framework 生成所需要的 ActiveRecord 类。

本例通过修改 Yii Framework 开发教程(26) 数据库-Active Record 示例，原例是手工编写 Employee.php ActiveRecord。

首先为工程添加一个 C# 项目（任意类型，我们只是利用这个项目来包含 CodeSmith 项目），然后添加一个 CodeSmith 项目和一个 CodeSmith 模板。然后参考 CodeSmith 使用教程(1): 概述 使用Schema Explorer 添加一个数据连接，本例连接到 Chinook 数据库：



图片 3.1 第13张

创建的代码模板 PhpActiveRecord.cst 定义个属性 TableName（数据库表名），复制 Yii Framework 开发教程(26) 数据库-Active Record 示例中 Employee.php 的定义并使用属性，代码如下：

```
<%@ Template Language="C#" TargetLanguage="PHP" Debug="False" %>

<%@ Property Name="TableName" Type="System.String" Description="Table name" %>

<?php
```

```
class <%= TableName %> extends CActiveRecord
{
    public static function model($className=__CLASS__)
    {
        return parent::model($className);
    }

    public function tableName()
    {
        return '<%= TableName %>';
    }
}

?>

<script runat="template">
  public override string GetFileName() {
    return TableName + ".php" ;
}
</script>
```

这时就可以通过定义 TableName 的属性给任意数据表生成对应的 ActiveRecord PHP 类了。 不过这还是要手工来一个一个来配置表名。 本例通过一个主模板和一个从模板的方式通过连接数据库自动为所有的表生成对应的 ActiveRecord

使用主从模板的具体用法后面再介绍，简单的说子模板相当于子函数，主模板类似于主函数可以调用子函数，主模板通过调用子模板，传给子模板属性从而可以生成多个文件。

创建一个代码模板 YiiDataModel.cst 作为主模板，使用子模板首先需要在主模板中进行注册才能使用：

```
<%@ Register Name="ActiveRecord" Template="PhpActiveRecord.cst" MergeProperties="false"  %>
```

完整代码如下：

```
<%@ CodeTemplate Language="C#" TargetLanguage="Text"
  Description="List all database tables" %>
<%@ Import Namespace="System.IO" %>
<%@ Property Name="SourceDatabase" Type="SchemaExplorer.DatabaseSchema"
  Category="Context" Description="Database containing the tables." %>

<%@ Register Name="ActiveRecord" Template="PhpActiveRecord.cst"
 MergeProperties="false"  %>
<%@ Assembly Name="SchemaExplorer" %>
<%@ Import Namespace="SchemaExplorer" %>
```

```
<script runat="template">
public string FirstLetterToUpper(string str)
{
   if (str != null)
   {
      if(str.Length > 1)
         return char.ToUpper(str[0]) + str.Substring(1);
      else
         return str.ToUpper();
   }
   return str;
}

</script>

<% for (int i = 0; i < SourceDatabase.Tables.Count; i++) { %>
  <%  string name= FirstLetterToUpper(SourceDatabase.Tables[i].Name); %>
  <%  string filename= @"../ActiveRecordDemo/protected/models/"+name+".php"; %>
    // instantiate the sub-template
   <%  ActiveRecord activeRecord = this.Create<ActiveRecord>();%>
   <%  activeRecord.TableName= name; %>
   <%  activeRecord.RenderToFile(filename,true); %>
<% } %>
```

FirstLetterToUpper 为C#函数，主要是把数据库表名的第一个字母变为大写（纯 C# 代码）。

SchemaExplorer 为 CodeSmith 提供的数据库访问库，可以用来获取数据库 Schema 的信息，如包含的表名，字段属性，主键外键等（后面具体介绍）

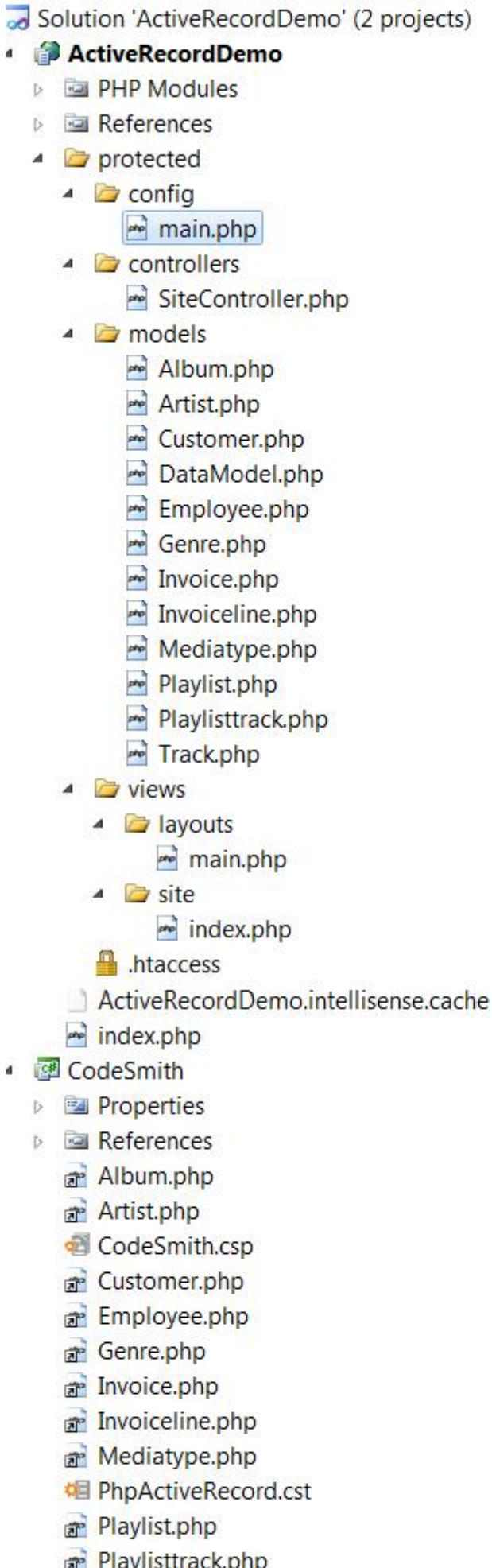
在主模板中，通过 ActiveRecord 来访问子模板（名字 ActiveRecord 为注册子模板时定义），使用 this.create 创建子模板实例，然后传入 TableName 属性，调用 RenderToFile 将子模板的结果写道指定的文件中。

此时在 CodeSmith.csp 中添加主模板，配置数据库为 Chinook，然后生成代码

```
Rendering output 'YiiDataModel'...

  Generated: D:\tmp\ActiveRecordDemo\ActiveRecordDemo\protected\models\Album.php
  Generated: D:\tmp\ActiveRecordDemo\ActiveRecordDemo\protected\models\Artist.php
  Generated: D:\tmp\ActiveRecordDemo\ActiveRecordDemo\protected\models\Customer.php
  Generated: D:\tmp\ActiveRecordDemo\ActiveRecordDemo\protected\models\Employee.php
  Generated: D:\tmp\ActiveRecordDemo\ActiveRecordDemo\protected\models\Genre.php
  Generated: D:\tmp\ActiveRecordDemo\ActiveRecordDemo\protected\models\Invoice.php
  Generated: D:\tmp\ActiveRecordDemo\ActiveRecordDemo\protected\models\Invoiceline.php
  Generated: D:\tmp\ActiveRecordDemo\ActiveRecordDemo\protected\models\Mediatype.php
  Generated: D:\tmp\ActiveRecordDemo\ActiveRecordDemo\protected\models\Playlist.php
```

```
  Generated: D:\tmp\ActiveRecordDemo\ActiveRecordDemo\protected\models\Playlisttrack.php
  Generated: D:\tmp\ActiveRecordDemo\ActiveRecordDemo\protected\models\Track.php
  Generated: D:\tmp\ActiveRecordDemo\CodeSmith\YiiDataModel.txt
Done rendering outputs: 1 succeeded, 0 failed, 0 skipped (1
```

刷新项目可以看到自动生成的代码文件


Solution 'ActiveRecordDemo' (2 projects)
ActiveRecordDemo
PHP Modules
References
protected
config
main.php
controllers
SiteController.php
models
Album.php
Artist.php
Customer.php
DataModel.php
Employee.php
Genre.php
Invoice.php
Invoiceline.php
Mediatype.php
Playlist.php
Playlisttrack.php
Track.php
views
layouts
main.php
site
index.php
.htaccess
ActiveRecordDemo.intellisense.cache
index.php
CodeSmith
Properties
References
Album.php
Artist.php
CodeSmith.csp
Customer.php
Employee.php
Genre.php
Invoice.php
Invoiceline.php
Mediatype.php
PhpActiveRecord.cst
Playlist.php
Playlisttrack.php

图片 3.2 第14张

本例只是为每个数据表生成最简单的 ActiveRecord，如果需要生成关联 ActiveRecord，可以进一步根据表之间的关系为每个 ActiveRecord 生成所需的 relations 方法，后面有时间进一步介绍。

本例下载

4

基本语法-CodeTemplate 指令

前面的几篇介绍了使用 CodeSmith 模板自动生成代码和编写代码模板的基本知识。也说过 CodeSmith最核心的部分是代码模板，从本篇开始介绍 CodeSmith 代码模板的基本语法，对于 Asp.Net 程序员来说，可以说是碰到老朋友了:-)，CodeSmith 的代码模板和 Asp.Net Page 几乎如出一辙。

本篇介绍 CodeTemplate 指令，这个是模板中唯一必须的声明，包含一些模板特殊的属性，包含模板使用的语言、生成的语言和一些对于模板的描述。比如：

```
<%@ CodeTemplate Language="C#" TargetLanguage="C#" Description="This is a demo template" %>
```

参数的介绍：

- Language：在开发编写模板时使用的语言，例如 C#，VB.NET，Jscript 等。
- TargetLanguage：只是对模板代码的一个分类，不会影响生成的代码语言。是模板的一个属性，说明模板要基于那种语言生成相应的代码。例如你可以用 CodeSmith 从任何一种语言生成C#代码。
- Description：对于模板的一些说明信息，在 CodeSmith Explorer 中选中该模板时会显示这里的信息。
- Inherits：所有 CodeSmith 模板默认继承自 CodeSmith.Engine.CodeTemplate，这个类提供模板使用的一些基本功能，像 ASP.NET 页面的 Page 类，这些被继承的类的属性可以被修改，但是这些新的类也必须继承 CodeSmith.Engine.CodeTemplate。CodeSmith 也同样可以找到这个类，当然你要引入一个组件包含这个类。
- Src：在某些方面 Src 和继承 Inherits 比较相似，它们都允许你从其他的类包含一些功能进模板。这两个属性的区别是，Src 可以让类与你的模板被动态编译，而 Inherits 仅允许你提供一个已经编译好的类或组件。
- Debug：可以确定是否在模板中可以包含调试符号。如果将这个属性设置为 True，则可以使用System.Diagnostics.Debugger.Break()方法来设置断点。
- LinePragmas：设置为 True，模板的错误将被指向到模板的源代码。设置为 False，模板的错误将被指向到编译的源代码。
- ResponseEncoding 指明代码模板的输出文件的编码方式，可以为 System.Text.Encoding.GetEncoding 支持的所有编码方式，如果输出文件已存在并且和要生成的内容一致，输出文件的编码方式不会变化。
- OutputType 指明输出文件的的输出模式，可以有三种模式:

Normal: 正常模式，代码模板输出内容写到正常的输出流（Response Stream）。 Trace： 输出内容写到 Trace（调试）输出流中。 None： 控制代码模板不输出任何内容，主要用在主-从模板的主模板中，有些情况下无需主模板输出任何内容。

- NoWarn 不显示某些编译警告，Warning 的 ID 使用逗号分隔，主要用在编译 C# 和 VB.Net 时用到。
- ClassName 使用 Code-Behind 时对应的类名称，类似于 Asp.Net 代码。
- Namespace 使用 Code-Behind 时对应的类命名空间名称。

- Encoding 代码模板自身使用的编码方式，缺省为 UTF–8.

5

基本语法–使用注释

在模板中可以添加注释，注释通过 <% – 和 – %>块来定义，注释可以有多行。比如：

```
<%--
Name: TestHarness.cst
Description: Generates a standard test harness for an object
--%>
```

如果在代码模板中使用 C#，VB.Net 或是 JavaScripts 脚本，可以使用所使用语言对应的注释，比如C# 语言可以使用 // 或 /* commented */。

如果需要在输出的文件中使用注释，和其它要输出的内容一样，直接写到模板中，在生成输出文件时这些内容都会直接被复制到输出文件中。如：

```
<%@ CodeTemplate Language="C#" TargetLanguage="VB">
' This class generated by CodeSmith Generator
```

6

基本语法–声明和使用属性

CodeSmith 的核心是模板，而使模板具有活力的就是属性，通过定义属性从而使代码模板能够根据配置生成所需的代码。在使用代码模板时首先也必须给模板定义的属性定义值才能使用 CodeSmith 通过模板产生代码。有些属性具有缺省值，这些属性可以不需要配置。 模板中的属性通过 Property 指令来定义：

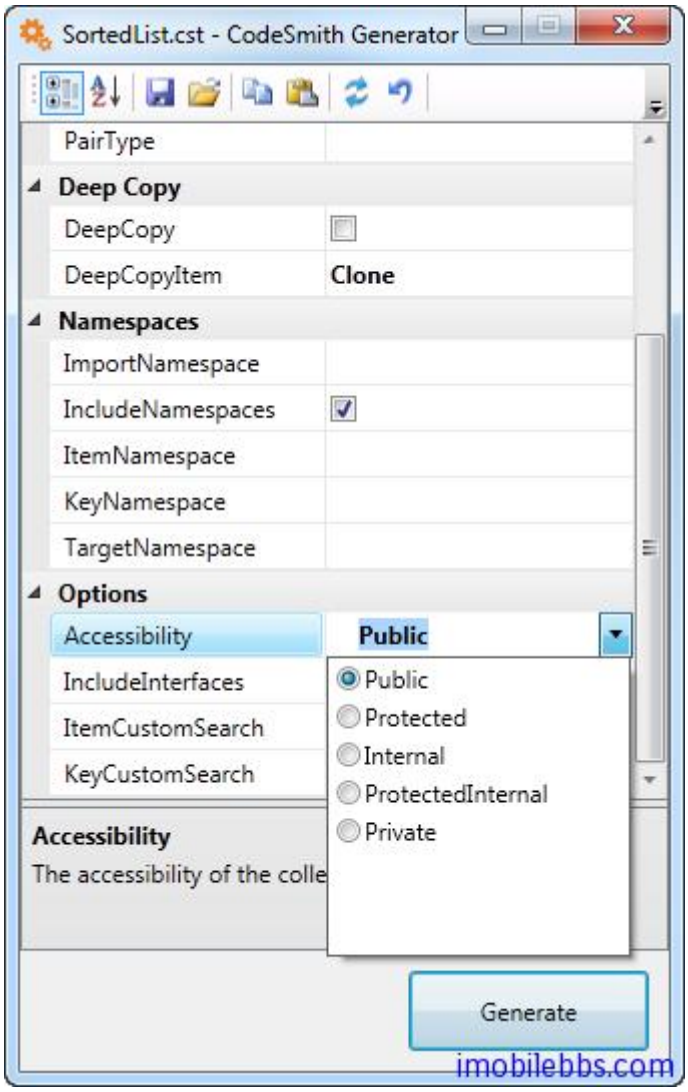
```
<%@ Property Name="ClassName" Type="String" Default="Class1" Category="Context"   Description="The name of the c
```

属性参数的介绍：

- Name：模版使用的参数的名称。
- Type：参数类型可以是任何 .NET 有效的数据类型，例如简单的 String 类型或者是 CodeSmith 的 SchemaExplorer.DatabaseSchema 类型。注意，类型必须是基类库的类型，例如用 String 或者Int32 代替 string 和 int。
- Default：设置默认值。
- Category：用来说明这个属性在 CodeSmith Explorer 的属性面板中显示成什么类型，例如下拉选择、直接输入等。
- Description：在属性面板中对于这个属性的描述。
- Optional：设置这个属性是否是必须的，设置为 True 表明这个参数值可有可无，设置为 False 则这个参数必须有值。
- Editor：表明在属性面板中输入这个属性的值时使用何种 GUI（图形界面编辑器）编辑器。
- EditorBase：编辑器使用的基本类型，如果没有被说明，UITypeEditor 为默认编辑器。
- Serializer 定义用于属性的 IPropertySerializer 类型。
- OnChanged 为属性发生变化时定义事件处理代码。
- DeepLoad 只用在 SchemaExplorer 对象，当为 True，SchemaExplorer 一次性取得有关数据库 Schema 的所有信息而避免多次查询数据库。

在配置属性时，每个属性根据其类型和 Editor 不同而使用不同的配置界面，对应一些简单的类型，比如 Int，String 可以直接编辑，而对于数据库类型可以使用 Schema Explorer，CodeSmith 预先定义了一些属性编辑器，此外也可以通过自定义为某些特殊的属性类型定义新的属性编辑器，这在后面再介绍。通常情况下无需自定义。

在某些情况下，如果所定义的属性值为一个列表中的某个值，比如在 CodeSmith 自带的模板 SortedList.cst 中定义了一个属性用来为所生成的类设置可见性：



图片 6.1 第15张

这可以通过定义一个枚举类型来实现：

```
<script runat="template">
Public Enum AccessibilityEnum
    [Public]
    [Protected]
    [Friend]
    [ProtectedFriend]
    [Private]
End Enum
</script>
```

然后为所定义的属性的类型定义为这个枚举类型：

```
<%@ Property Name="Accessibility" Type="AccessibilityEnum"
Category="Options" Description="The accessibility of the class to be generated." %>
```

由于属性可以定义为可选（Optional），因此在模板中需要检测某个的属性是否配置过，比如下面定义了一个属性为 Optional

```
<%@ Property Name="ClassNamespace" Type="System.String" Optional="True"
Category="Context"
Description="The namespace that the generated class will be a member of." %>
```

在模板中脚本需要检测这个属性是否有值可以通过下面代码来实现：

```
<% if (ClassNamespace != null && ClassNamespace.Length > 0)
{ %>namespace <%= ClassNamespace %>{<% }
%>
```

从上面也可以看出，在脚本中使用属性直接使用属性名字即可，无需加前缀（比如$之类的前缀）。

7

基本语法-转义Asp.Net标记

由于 CodeSmith 的代码模板使用了和 Asp.Net 类似的语法，因此如果要使用 CodeSmith 模板生成Asp.Net 脚本时比如“<%” 就碰到了问题， <% 会被 CodeSmith 解释成 CodeSmith 自己的标记，因此需要使用转义标签来代替需要插入到 Asp.Net 代码中的标签。

具体方法是使用 <%% 来替换需要生成的 Asp.Net 中的 <%标记。

比如我们要生成如下的 Asp.Net 代码：

```
<asp:FormView ID="FormView1" DataSourceID="SqlDataSource1" DataKeyNames="ProductID" RunAt="server">
 <ItemTemplate>
  <table>
   <tr>
    <td align="right"><b>Product ID:</b></td>
    <td><%# Eval("ProductID") %></td>
   </tr>
  </table>
 </ItemTemplate>
</asp:FormView>
```

可以在 CodeSmith 的模板中使用<%% 来替换 <%

```
<asp:FormView ID="FormView1" DataSourceID="SqlDataSource1" DataKeyNames="ProductID" RunAt="server">
 <ItemTemplate>
  <table>
   <tr>
    <td align="right"><b>Product ID:</b></td>
    <td><%%# Eval("ProductID") %></td>
   </tr>
  </table>
 </ItemTemplate>
</asp:FormView>
```

8

CodeTemplate 对象

在使用代码模板产生代码时，CodeSmith 引擎背后使用了不少对象来帮助代码的生成，其中常用的有

- CodeTempate（类似于 Asp.Net 的 Page 类）
- Progress 用于显示代码生成的进度
- CodeTemplateInfo 可以返回关于当前模板自身的一些信息。

本篇介绍 CodeTemplate，CodeTemplate 代表了由 CodeSmith 引擎处理的代码模板对象，可以通过 CodeTemplate 对象直接和 CodeSmith 引擎交互，比如：

- 使用 GetFileName 修改模板生成的缺省文件名
- 使用 Render method 把模板的输出到多个文件中
- 通过 events 把代码插入到 CodeSmith 引擎处理模板的过程中。
- 通过 Response 属性直接在输出文件中写内容。

# #

使用 GetFileName 修改模板输出的文件名

在前面的例子 CodeSmith 使用教程(2): 编写第一个代码模板，我们已经使用 GetFileName 修改过输出的文件名，比如在你的模板中定义了一个 ClassName 属性，可以通过 GetFileName 把模板输出的缺省文件名改成类名

```
<%@ Template Language="C#" TargetLanguage="Text" %>
<%@ Property Name="ClassName" Type="System.String" Default="ClassName" %>
```

This template shows off how to override the GetFileName method.

```
<script runat="template">
public override string GetFileName()
{
    return ClassName + ".cs";
}
</script>
```

#

重载 ParseDefaultValue 方法

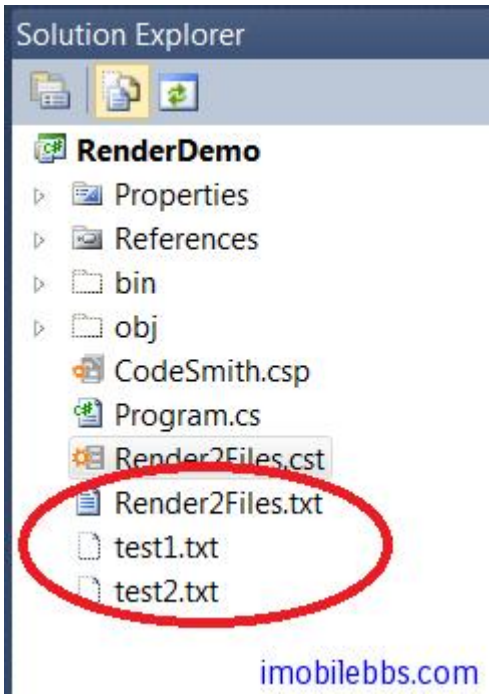
在定义属性的缺省值时，有时有些属性的缺省值可能无法从 String 转换，此时可以通过重载ParseDefaultValue 方法，这个方法会被 CodeSmith 引擎中处理每个属性时调用，如果你重载了这个方法，可以按照你自己的逻辑来处理属性的缺省值。

# #

重载 Render 方法

CodeTemplate 的 Render 方法是 CodeSmith 引擎生成最终输出时调用的，可以通过重载这个方法来修改输出的内容或是把输出写到多个文件中。 比如下面代码除了生成缺省的输出外，还把输出写到另外两个文件中：

```
<%@ CodeTemplate Language="C#" TargetLanguage="Text"
  Description="AddTextWriter Demonstration." %>
<%@ Import Namespace="System.IO" %>
//This template demonstrates using the AddTextWriter method
//to output the template results to multiple locations concurrently.
<script runat="template">
public override void Render(TextWriter writer)
  {
    StreamWriter fileWriter1 = new StreamWriter(@"test1.txt", true);
    this.Response.AddTextWriter(fileWriter1);

    StreamWriter fileWriter2 = new StreamWriter(@"test2.txt", true);
    this.Response.AddTextWriter(fileWriter2);

    base.Render(writer);

    fileWriter1.Close();
    fileWriter2.Close();
  }
</script>
```



图片 8.1 第16张

注意调用基类的 base.Render，否则你就不会输出到缺省的文件。本例下载

# #

模板事件

CodeTemplate 类定义了下面几个事件，你可以中这些事件发生时添加自动的事件处理。

- OnInit 事件帮助中代码模板创建时
- OnPreRender 事件发生在准备写输出文件前
- OnPostRender 事件发生在准备写输出文件后
- OnPropertyChanged 事件发生在属性值发生变化时.

# #

使用 Response 对象

和 Asp.Net 的 Page 对象一样，可以通过 CodeTemplate 的 Response 属性直接在输出流中写入内容。比如

```
<%@ CodeTemplate Language="C#" TargetLanguage="Text"
Description="This template demonstrates writing directly to the Response property" %>
<% RenderDirect(); %>
<script runat="template">
public void RenderDirect()
{
Response.WriteLine("Written directly to the Response property.");
Response.WriteLine("Hello " + System.Environment.UserName + "!");
}
</script>
```

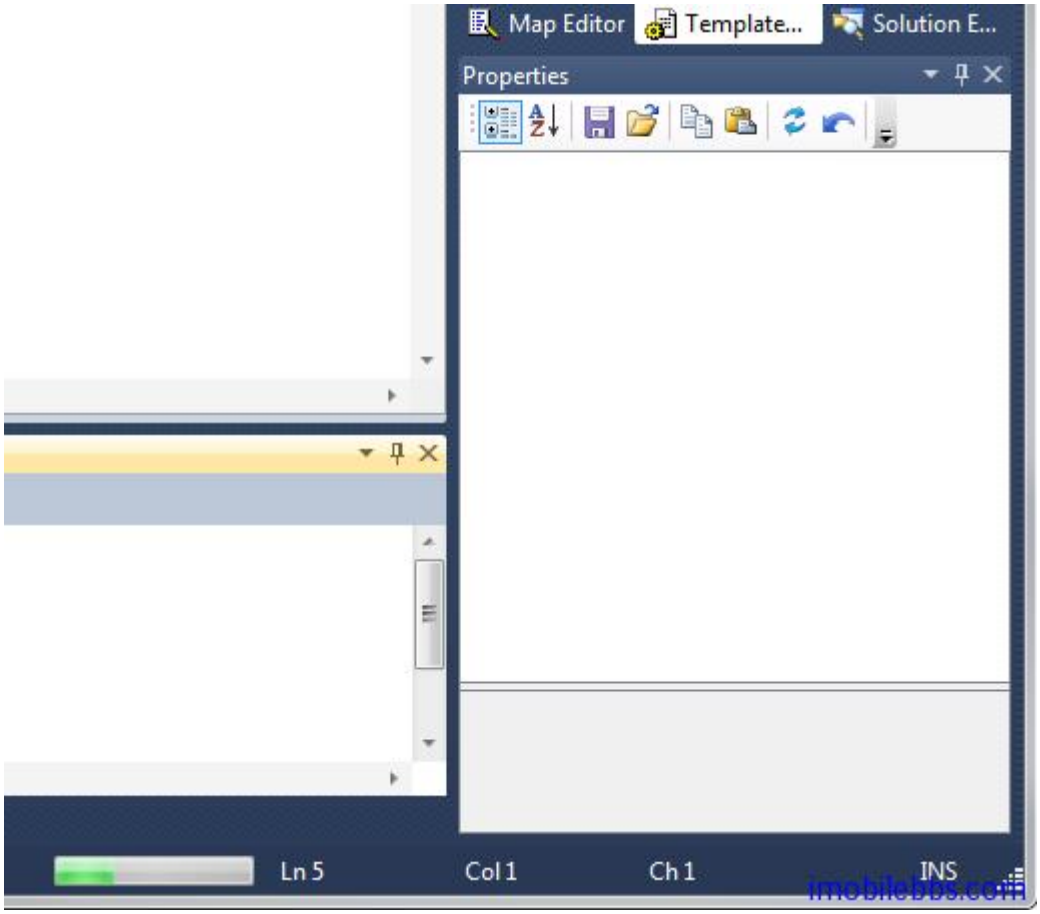
直接在输出流中写入两行文字。Response 对象的类型为 CodeTemplateWriter 类 ， 常用的方法有：

- AddTextWriter –添加额外的输出位置
- Indent – 为输出添加一个缩进
- Unindent – 为输出减少一个缩进
- Write –写入内容
- WriteLine – 写入内容并添加分行符

9

Progress 对象

使用 Progress 对象可以为 CodeSmith 生成代码的过程显示一个进度条，这对于生成比较费时的模板操作是非常有用的，如果你使用 Visual Studio，可以在状态栏中显示一进度条：



图片 9.1 第17张

使用进度条的方法是通过 CodeTemplate 对象的 Progress 属性对象，首先是设置 Progress 对象的最大值和步长，本例通过一个简单的循环来模拟一个费时的操作：

```
<%@ Template Language="C#" TargetLanguage="Text" Debug="False" %>

<%@ Import Namespace="System.Threading" %>
This is a progress demo.

<% SimulateProgress(); %>

<script runat="template">
public void SimulateProgress(){

Progress.MaximumValue = 25;
Progress.Step = 1;

   for(int i=0;i<25;i++){
```

```
        Progress.PerformStep();
        Thread.Sleep(100);
        Response.WriteLine("step {0} ",i);
    }
}
</script>
```

让进度条前进一步是通过 Progress 对象的 PerfStep 方法来实现的。

本例下载

10

CodeTemplateInfo 对象

通过 CodeTemplateInfo 对象可以获取代码模板文件本身的一些信息，比如文件名，源语言，编码方法，其支持的属性有：

属性名	描述
CodeBehind	该模板的 Code-behind 的文件名或者模板不使用 CodeBehind 时为空字符串
ContentHashCode	返回代码模板的一个 Hash 值
DateCreated	返回模板创建的时间
DateModified	返回模板修改的时间
Description	返回模板说明
DirectoryName	返回模板所处的目录
FileName	返回模板的文件名
FullPath	返回模板的完整路径
Language	返回模板的源语言类型
TargetLanguage	返回模板生成的目标语言类型

本例通过 CodeTempalte 对象的 CodeTemplateInfo 属性对象中输出文件中显示上面个各个属性值：

```
<%@ CodeTemplate Language="C#" TargetLanguage="Text"
Description="Demonstrates CodeTemplateInfo." %>
<% DumpInfo(); %>
<script runat="template">
public void DumpInfo()
{
   Response.WriteLine("Template: {0}", CodeTemplateInfo.FileName);
   Response.WriteLine("Created: {0}", CodeTemplateInfo.DateCreated);
   Response.WriteLine("Description: {0}", CodeTemplateInfo.Description);
   Response.WriteLine("Location: {0}", CodeTemplateInfo.FullPath);
   Response.WriteLine("Language: {0}", CodeTemplateInfo.Language);
   Response.WriteLine("Target Language: {0}", CodeTemplateInfo.TargetLanguage);
}
</script>
```

显示结果如下：

```
Template: CodeTemplateInfo.cst
Created: 6/01/2013 12:49:57 PM
Description: Demonstrates CodeTemplateInfo.
Location: D:\tmp\CodeTemplateInfoDemo\CodeTemplateInfoDemo\CodeTemplateInfo.cst
Language: C#
Target Language: Text
```

本例下载

11

引用其它文件或 .Net 类库

在 CodeSmith 模板中可以引用 .Net 类库，和普通的 .Net 项目不同的是，对 .Net 库的引用不是通过项目的 Add reference 来实现，而是通过在代码模板中指明所要引用的 Assembly.

比如引用 CodeSmith 自带的 CodeSmith.CustomProperties.dll ，可以使用如下语句：

```
<%@ Assembly Name="CodeSmith.CustomProperties" %>
```

- Name指明所有需要引用的 Assembly 的名称，也可以使用 Assembly 的全名，比如ExampleAssembly, Version=0.0.0.0, Culture=neutral, PublicKeyToken=null
- Src 指明可以动态编译的源码的相对路径名称
- Path 指明应用的 Assembly 存放的路径

引用合适的 Assembly 之后，和普遍 C# 语言类似，对于使用到的 .Net 类，需要通过 Import 引入该类所在的命名空间。

比如 CodeSmith 使用教程(9): Progress对象引入 Thread 类所在的 System.Threading

```
<%@ Import Namespace="System.Threading" %>
```

此外，如果在代码模板中需要引入一些源代码（比如一些公用的代码）可以通过 include ,比如：

```
<!– #include file="CommonScript.cs" –>
```

共享代码的方法除了上面使用的 include 方法外，还可以通过设置 CodeTemplate 和 Assembly 的Src 属性来实现等。

12

使用主从代码模板

在前面的教程 CodeSmith 使用教程(3): 自动生成 Yii Framework ActiveRecord 我们使用了主，从模板来实现了从数据库为 Yii Framework 生成多个表的 ActiveRecord 类定义，中 CodeSmith 项目中通过主模板和从模板的配合可以实现复杂的代码生成过程，主模板和从模板的关系有点类似主程序和子函数的关系。使用主-从模板的基本步骤如下：

- 定义从模板，从模板可以定义属性
- 定义主模板，中主模板中如果要使用从模板，首先需要在主模板中注册从模板，主模板中也也可以定义属性，主模板和从模板中的属性可以通过定义“合并”模式构造最终模板所定义的属性集合。
- 调用主模板，设置主模板和从模板所需的属性生成所需代码

# #

注册子模板

```
<%@ Register Name="Header" Template="Header.cst"
  MergeProperties="True" ExcludeProperties="IncludeMeta" %>
```

Name：子模板在主模板中的类型名称，在主要模板中可以通过该类型创建子模板的实例 Template: 子模板文件名 MergeProperties： 是否需要把子模板中定义的属性：“合并”到主模板中。缺省为 False ExcludeProperties： 如果子模板的属性合并到主模板中时需要排除的属性列表，以逗号分隔。

#

子模板复制主模板中的属性

MergeProperties=”True” 可以把从模板中的属性合并到主模板中，如果从模板需要引用主模板的属性，比如主模板中定义了服务器地址，在多个子模板中都需要引用这个属性，此时可以通过复制父模板属性 CopyPropertiesTo 来实现：

```
// instantiate the sub-template
Header header = this.Create<Header>();

// copy all properties with matching name and type to the sub-template instance
this.CopyPropertiesTo(header);
```

CopyPropertiesTo 方法比较主模板中定义的属性和子模板中定义的属性，如果发现从模板中定义的属性和主模板中定义的属性名称类型相同（匹配）则把主模板中属性值复制到子模板中。

# #

设置子模板属性

在主模板中要创建子模板的实例，可以直接通过 Create 方法

```
// instantiate the sub-template
Header header = this.Create<Header>();

// include the meta tag
header.IncludeMeta = true;
```

Create 中的 Header 为注册子模板时 Name 来定义的类型，通过 Create 创建子模板的实例后，就直接可以通过该实例的属性来访问子模板中的属性，比如上面代码中 IncludeMeta 为子模板中定义的一个属性。

# #

从子模板输出结果

创建好子模板的实例，设置好子模板的属性，在主模板中就可以让子模板输出结果，有几种方法可以从子模板输出内容。

第一种是把子模板生成的结果直接插入到主模板中

```
// instantiate the sub-template.
Header header = this.Create<Header>();
// render the sub-template to the current output stream.
header.Render(this.Response);
```

第二种方法是把结果输出到单独的文件中：

```
// instantiate the sub-template.
Header header = this.Create<Header>();
// render the sub-template to a separate file.
header.RenderToFile("Somefile.txt");
```

具体的例子可以参见 CodeSmith 使用教程(3): 自动生成 Yii Framework ActiveRecord

13 调试

编写 CodeSmith 模板和编写程序一样，也需要进行调试，CodeSmith 支持使用 CLR’s Just-in-Time debugger 调试模板。

要调试模板，首先要在 CodeTemplate 声明中打开调试 Debug=”True”：

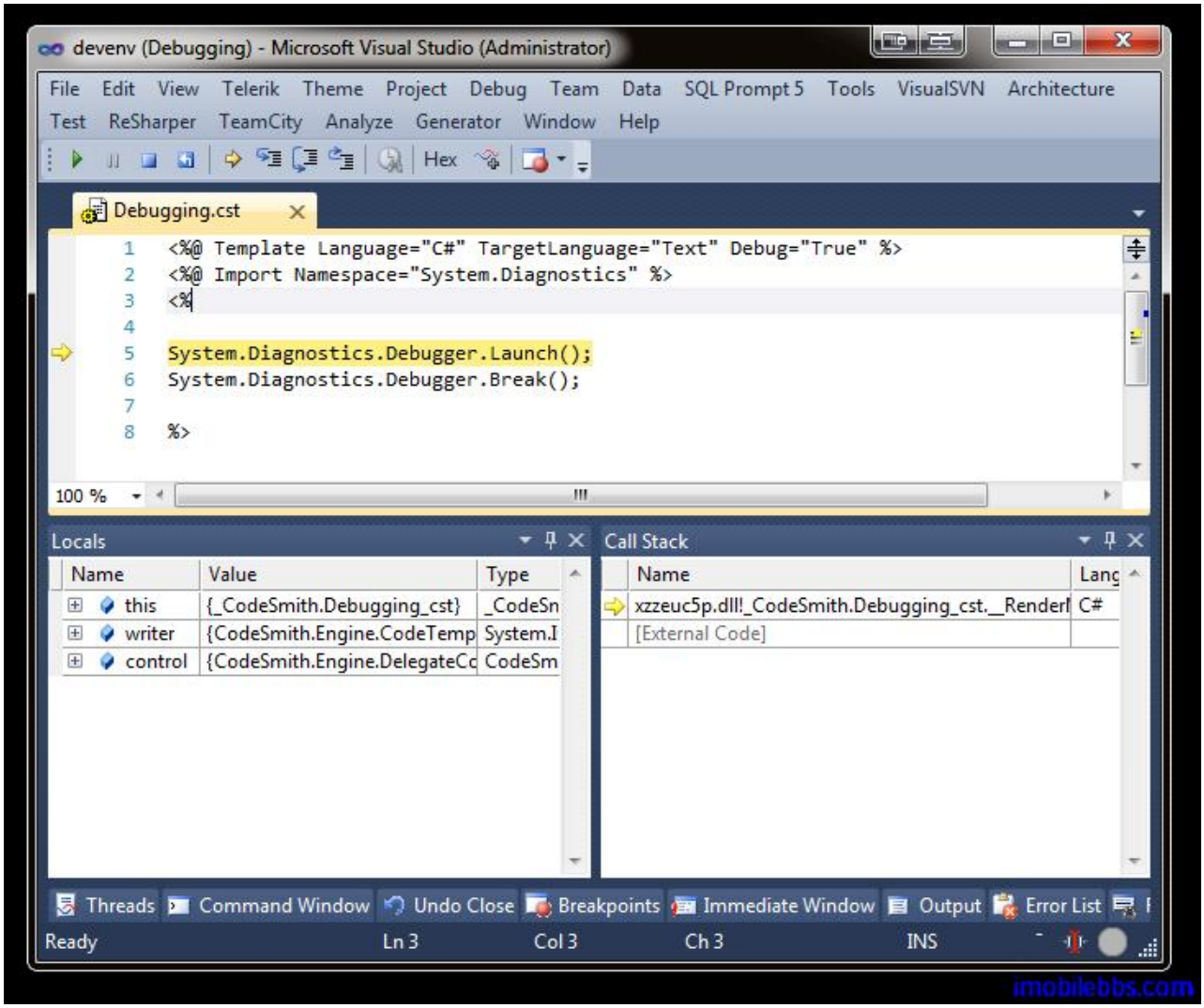
```
<%@ CodeTemplate Language="C#" TargetLanguage="C#" Debug="True" %>
```

第二步是设置断点：在需要设置断点的地方调用 System.Diagnostics.Debugger.Break();

```
System.Diagnostics.Debugger.Launch();
System.Diagnostics.Debugger.Break();
```

在调用 System.Diagnostics.Debugger.Break();之前需要首先调用System.Diagnostics.Debugger.Launch();

这样在 Generate Output 时 Visual Studio 在指定的断点暂停运行：



图片 13.1 第18张

此外也可以利用 .Net 的 System.Diagnostics.Trace 和 System.Diagnostics.Debug 添加调试信息。

14

使用 SchemaExplorer 来获取数据库定义

Contents

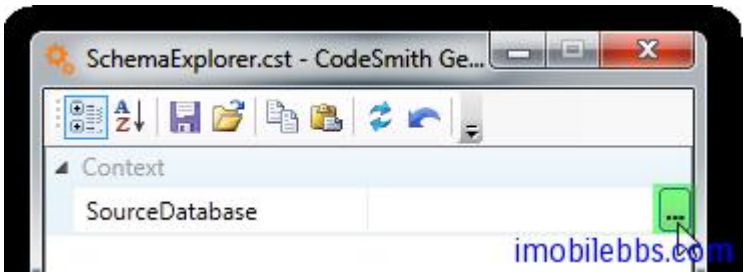
1.表的列
2.视图的列
3.命令参数

在前面例子 CodeSmith 使用教程(3): 自动生成 Yii Framework ActiveRecord 我们使用了SchemaExplorer 来获取数据的 MetaData（数据库 Schema 定义）来自动生成 Yii Framework 的数据库表对应的 ActiveRecord 定义，本篇较详细的介绍一下的 SchemaExplorer 的用法，下一篇通过实例除了自动生成自动生成 Yii Framework 的数据库表对应的 ActiveRecord 定义外，还自动生成关联 ActiveRecord 的关系定义，也就是根据数据库表之间的关系（一对多，一对一，多对多）为ActiveRecord 定义 relations.

CodeSmith 的 SchemaExplorer 定义在 Assembly SchemaExplorer.dll 中，其命名空间为SchemaExplorer ，因此如果需要使用 CodeSmith 的 SchemaExplorer 功能的话，需要添加对SchemaExplorer.dll 的引用，如下：

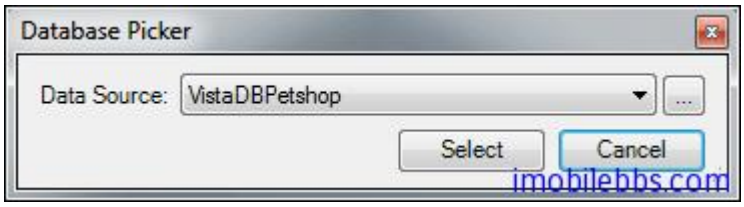
```
<%@ CodeTemplate Language="C#" TargetLanguage="Text" Description="List all database tables" %>
<%@ Property Name="SourceDatabase" Type="SchemaExplorer.DatabaseSchema"
 Category="Context" Description="Database containing the tables." %>
<%@ Assembly Name="SchemaExplorer" %>
<%@ Import Namespace="SchemaExplorer" %>
Tables in database "<%= SourceDatabase %>":
<% for (int i = 0; i < SourceDatabase.Tables.Count; i++) { %>
      <%= SourceDatabase.Tables[i].Name %>
<% } %>
```

以上代码添加了 SchemaExplorer 库的引用，并定义了一个属性 SourceDatabase，其类型为SchemaExplorer.DatabaseSchema ，在运行这个模板前，必须设置 SourceDatabase 的值：



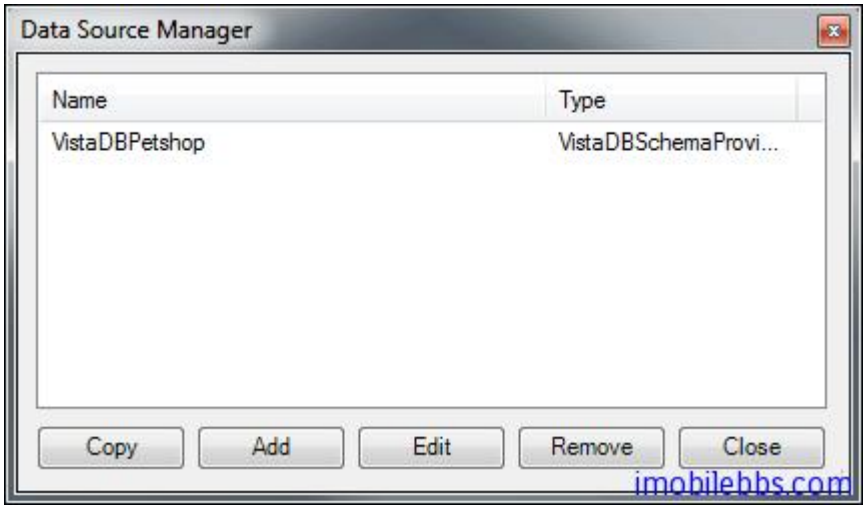
图片 14.1 第19张

SourceDatabase 属性后显示一个“…”的按钮，表示使用一个附加的专用的编辑器来定义这个属性，点击这个按钮将启动数据库选择对话框：



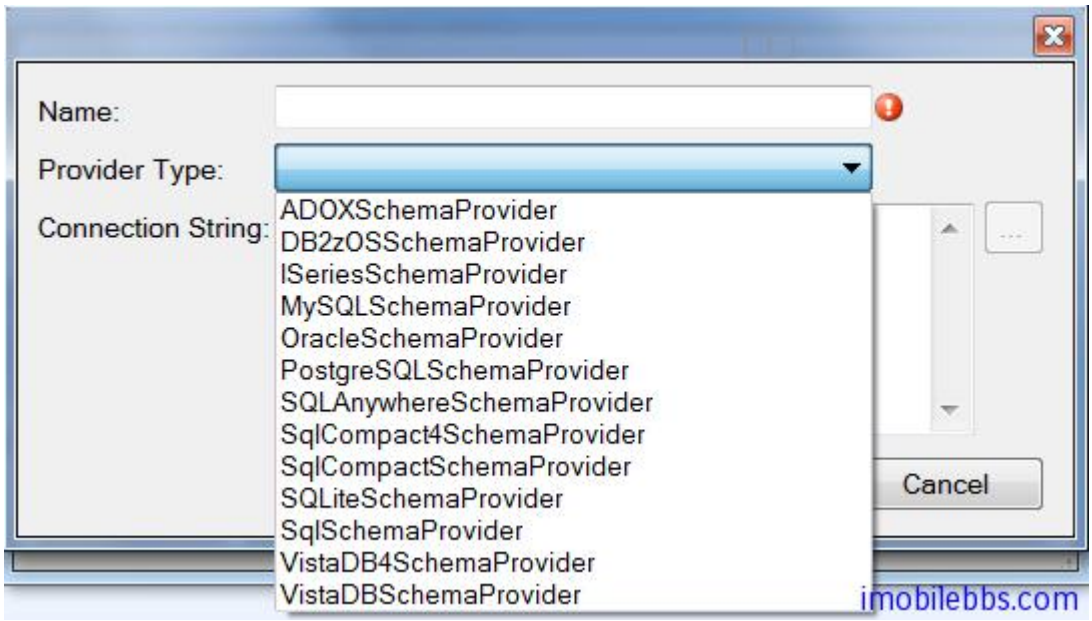
图片 14.2 第20张

使用这个对象框可以选择已通过 Schema Explorer 定义过的数据库或者添加新的数据库，通过单击“…”来添加新的数据库定义：



图片 14.3 第21张

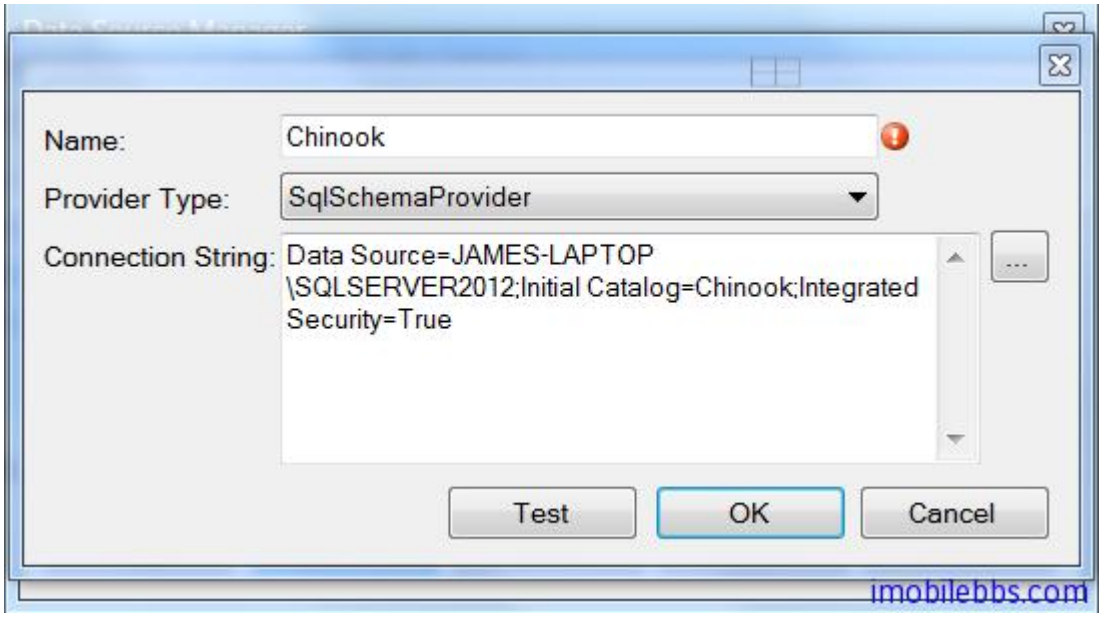
如果添加一个新的数据源，SchemaExplorer 打开了 数据源对话库 ，选择合适的数据源类型：



图片 14.4 第22张

CodeSmith 缺省支持的数据源类型有很多，包括了常用的 ADO， DB2,MySQL，Oracle，PostgreSQL， SQL Server，Sqlite 等，也可以自定义新的数据源类型。

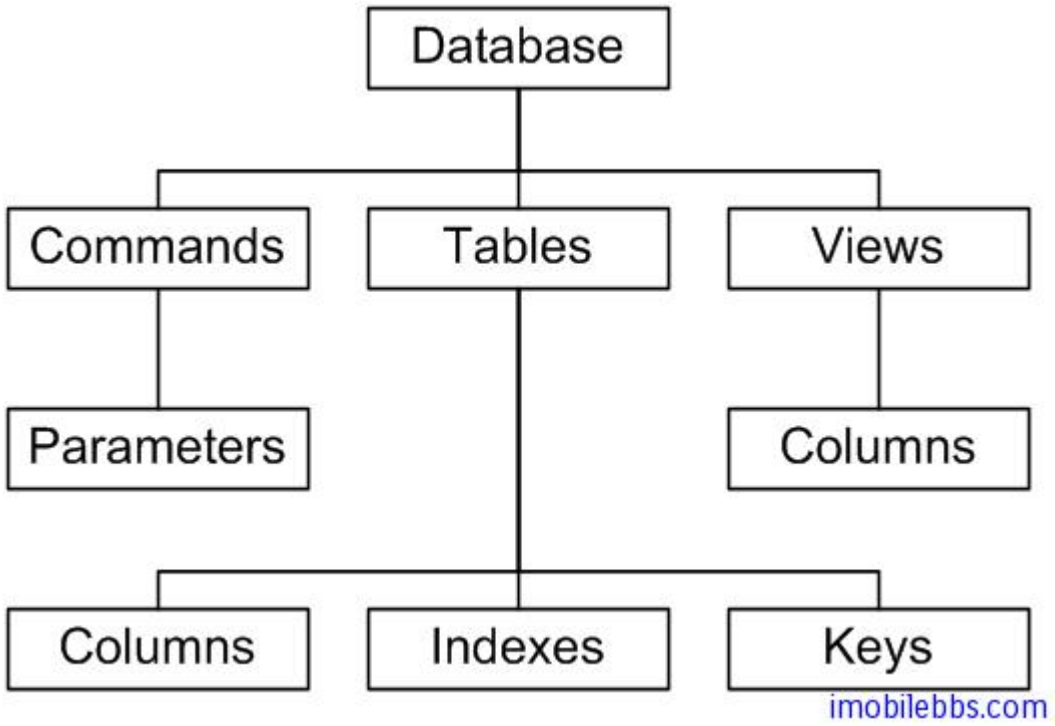
本例我们选用 SQL Server 类型 ，并使用 Chinook 示例数据库:



图片 14.5 第23张

选择数据库 Chinook，显示结果:

Tables in database "Chinook": Album Artist Customer Employee Genre Invoice InvoiceLine MediaType Playlist PlaylistTrack Track SchemaExplorer 对应数据库的 MetaData（表定义，列定义，主键，外键定义等）定义如下的对象模型，可以在代码模板中使用:



图片 14.6 第24张

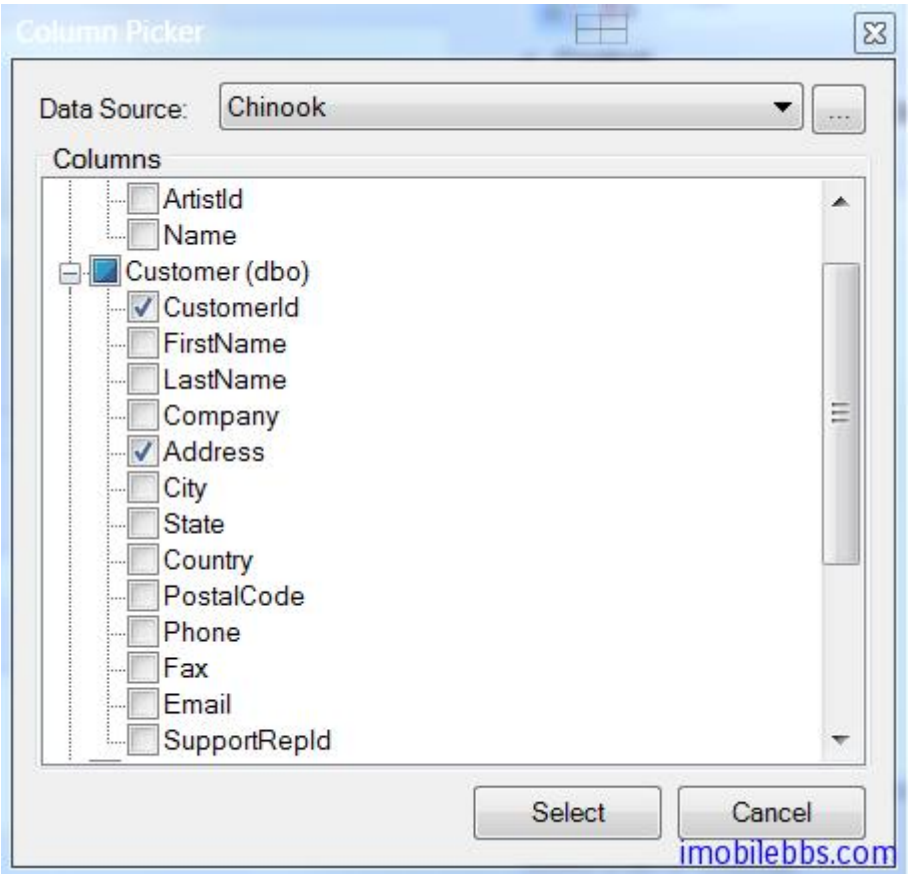
上图表示 SchemaExplorer 定义了多种对象集合类型，对象类型，比如 DatabaseSchema 定义了Commands 属性， 其类型为 CommandSchemaCollection，这个集合的每项类型为 CommandSchema ，对应到数据库定义中的一个命令。 通过这个属性可以获取 Command 的定义等信息。

使用 SchemaExplorer 除了可以使用 SchemaExplorer.DatabaseSchema 类型来定义属性，还可以通过下面四种类型 - TableSchema和 TableSchemaCollection - ViewSchema 和 ViewSchemaCollection - CommandSchema 和 CommandSchemaCollection - ColumnSchema 和 ColumnSchemaCollection

分别对应到表类型，视图类型，命令类型，列类型，比如使用

```
<%@ Property Name="SourceColumns"  Type="SchemaExplorer.ColumnSchemaCollection"
Category="Database"  Description="Select a set of columns." %>
```

选择一个表的多个列（ColumnSchemaCollection）



图片 14.7 第25张

对应这些集合类型（比如 TableSchemaCollection，ColumnSchemaCollection）缺省的排序是由数据库决定的，因此可能不是排好序的，如果需要排序的话，可以通过 Sort 方法来实现，比如：

```
TableSchemaCollection tables = new TableSchemaCollection(SourceDatabase.Tables);
tables.Sort(new PropertyComparer("Name"));
```

SQL Server 数据库可以对表或列定义一些附加的属性（Extended Property）SchemaExplorer 也提供了方法可以来访问/添加 这些 Extended Property。 比如 SQL Server 定义一个扩展属性表示某个列是否为 Identity 列，这可以通过下面代码来获取：

```
Identity Field = <% foreach(ColumnSchema cs in SourceTable.Columns) {
   if( ((bool)cs.ExtendedProperties["CS_IsIdentity"].Value) == true) {
      Response.Write(cs.Name);
   }
} %>
```

更好的方法是使用SchemaExplorer.ExtendedPropertyNames类和ExtendedProperty定义的扩展方法。

例如：

```
Identity Field = <% foreach(ColumnSchema cs in SourceTable.Columns) {
   if(cs.ExtendedProperties.GetByKey<bool>(SchemaExplorer.ExtendedPropertyNames.IsIdentity) == true) {
      Response.Write(cs.Name);
   }
} %>
```

CodeSmith 缺省支持的扩展属性如下：

表的列

Extended Property Key	SchemaExplorer.ExtendedPropertyName Property Name	描述
CS_Description	Description	The Description
CS_IsRowGuidCol	IsRowGuidColumn	The Column is a Row Guid
CS_IsIdentity	IsIdentity	Identity Column
CS_IsComputed	IsComputed	Computed Column or Index
CS_IsDeterministic	IsDeterministic	Column is Deterministic
CS_IdentitySeed	IdentitySeed	Identity Seed
CS_IdentityIncrement	IdentityIncrement	Identity Increment
CS_SystemType	SystemType	The System Type (E.G., System.String)
CS_Default	DefaultValue	The default value

视图的列

Extended Property Key	SchemaExplorer.ExtendedPropertyName Property Name	描述
CS_Description	Description	The Description
CS_IsComputed	IsComputed	Computed Column or Index
CS_IsDeterministic	IsDeterministic	Column is Deterministic

命令参数

Extended Property Key	SchemaExplorer.ExtendedPropertyName Property Name	描述
CS_Description	Description	The Description
CS_Default	DefaultValue	The default value

下一篇通过 Table 的 Key（外键和主键）为 Yii Framework 表的 ActiveRecord 添加 Relations

15

为 Yii Framework 创建生成 ActiveRecord 的代码模板

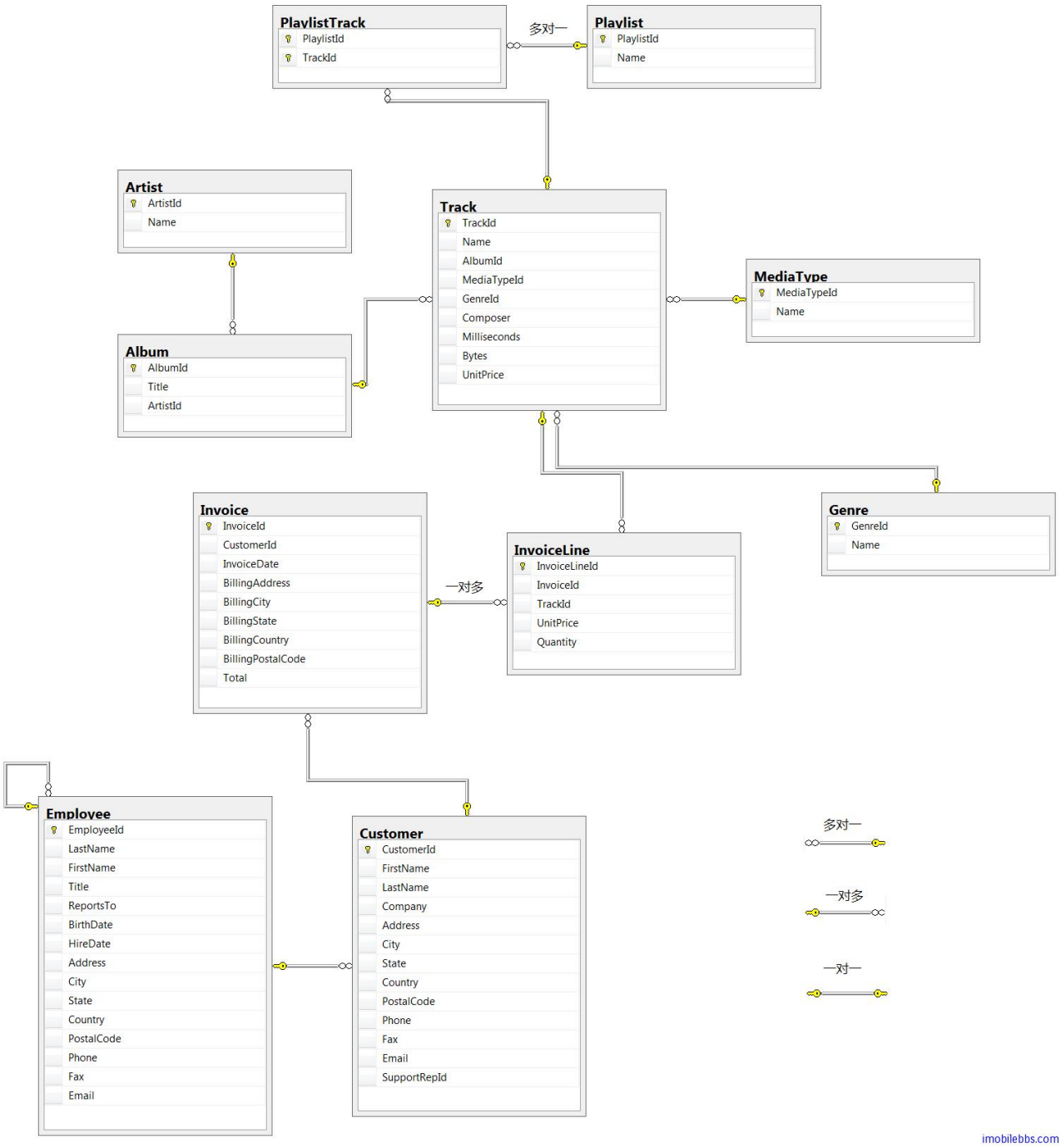
在 CodeSmith 使用教程(3): 自动生成 Yii Framework ActiveRecord 我们通过 SchemaExploer 为 Yii Framework 从数据库生成简单的 ActiveRecord 类，没有考虑到表和表之间的关系。本例我们使用 CodeSmith 为 Yii Framework 创建一个通用的代码模板，可以使用上例介绍的SchemaExploer ，不过在查看 CodeSmith 自带的例子中有个生成 Hibernate 的例子，这个模板的使用可以参见 CodeSmith 使用教程(1): 概述 ，CodeSmith 提供了这个模板的源码，使用到了 CodeSmith.SchemaHelper （ CodeSmith 没有提供相应的文档 ），不过可以通过阅读 NHiberante 的模板了解其一般用法。

为生成 Yii Framework ActiveRecord 类之间的 relation ，先要了解一下表和表之间的关系：

两个 AR 类之间的关系直接通过 AR 类所代表的数据表之间的关系相关联。 从数据库的角度来说，表 A 和 B 之间有三种关系：一对多（ one-to-many，例如 tbl_user 和 tbl_post ），一对一（ one-to-one 例如 tbl_user 和tbl_profile ）和 多对多（ many-to-many 例如 tbl_category 和 tbl_post ）。 在 AR 中，有四种关系：

- BELONGS_TO（ 属于 ）: 如果表 A 和 B 之间的关系是一对多，则 表 B 属于 表 A (例如 Post 属于 User);
- HAS_MANY（ 有多个 ）: 如果表 A 和 B 之间的关系是一对多，则 A 有多个 B (例如 User 有多个 Post);
- HAS_ONE（ 有一个 ）: 这是 HAS_MANY 的一个特例，A 最多有一个 B (例如 User 最多有一个 Profile);
- MANY_MANY: 这个对应于数据库中的 多对多 关系。 由于多数 DBMS 不直接支持 多对多 关系，因此需要有一个关联表将 多对多 关系分割为 一对多 关系。 在我们的示例数据结构中，tbl_post_category 就是用于此目的的。在 AR 术语中，我们可以解释 MANY_MANY 为 BELONGS_TO 和 HAS_MANY 的组合。例如，Post 属于多个（ belongs to many ） Category ，Category 有多个（ has many ） Post.

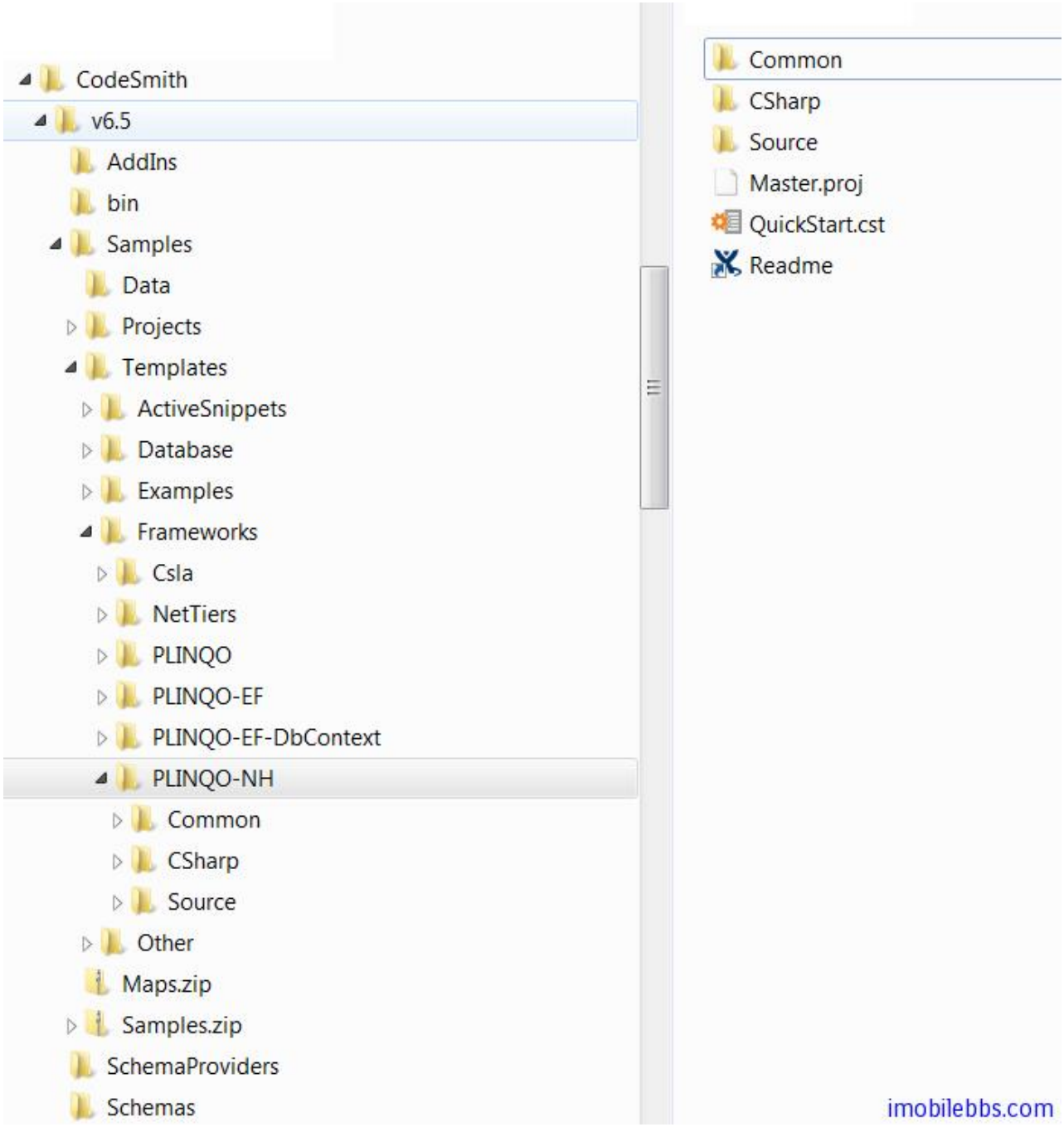
本例还是使用 Chinook 数据库，修改 Yii Framework 开发教程(27) 数据库-关联 Active Record 示例。数据表之间的关系如下：



图片 15.1 第26张

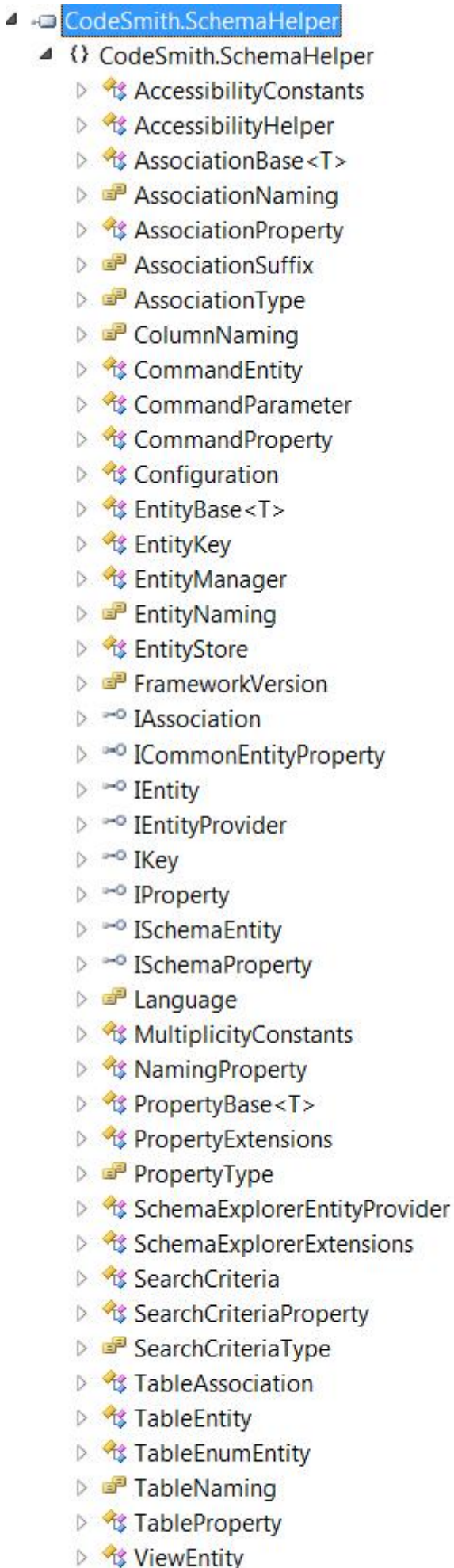
CodeSmith 中 PLINQO-NH 代码位置：

缺省目录为 C:\Program Files (x86)\CodeSmith\v6.5\Samples\Templates\Frameworks\PLINQO-NH



图片 15.2 第27张

CodeSmith.SchemaHelper 定义的主要类有：


CodeSmith.SchemaHelper
CodeSmith.SchemaHelper
AccessibilityConstants
AccessibilityHelper
AssociationBase<T>
AssociationNaming
AssociationProperty
AssociationSuffix
AssociationType
ColumnNaming
CommandEntity
CommandParameter
CommandProperty
Configuration
EntityBase<T>
EntityKey
EntityManager
EntityNaming
EntityStore
FrameworkVersion
IAssociation
ICommonEntityProperty
IEntity
IEntityProvider
IKey
IProperty
ISchemaEntity
ISchemaProperty
Language
MultiplicityConstants
NamingProperty
PropertyBase<T>
PropertyExtensions
PropertyType
SchemaExplorerEntityProvider
SchemaExplorerExtensions
SearchCriteria
SearchCriteriaProperty
SearchCriteriaType
TableAssociation
TableEntity
TableEnumEntity
TableNaming
TableProperty
ViewEntity

图片 15.3 第28张

几个主要的类为

- EntityManager 管理所有的 Entity（对应于整个数据库）
- Entity 实体类（对应到单个表，视图）
- IAssoication 关系（定义表和表之间的关系）
- AssoicationType 关系的类型 (见下表)

根据 AssociationType，数据库之间的关系以及 Yii AR 支持的几种关系，可以定义下表:

AssociationType	Table Relation	Yii Relation
ManyToMany	Many To Many	MANY_MANY
ManyToOne	Many To One	BELONGS_TO
ManyToZeroOrOne		
ZeroOrOneToMany	One To Many	HAS_MANY
OneToMany		
OneToOne	One To One	HAS_ONE
OneToZeroOrOne		

图片 15.4 第29张

整个模板也是采用主-从模板的方式，主模板枚举 EntityManager 中的每个 Entity，然后调用子模板为每个表 生成 AR 类:

```
public void Generate()
{
  EntityManager entityManager = CreateEntityManager();
  foreach(IEntity entity in entityManager.Entities)
   {
      if (!(entity is CommandEntity)) {
         RenderEntity(entity);
      }
   }
}
```

...

```
private void RenderEntity(IEntity entity)
{

   string folder=@"../models/";
   EntityTemplate entityTemplate = this.Create<EntityTemplate>();
```

```
    entityTemplate.SourceEntity = entity;
    entityTemplate.RenderToFile(folder+entity.Name+".php", true);
}
```

子模板则根据每个 Entity 的 Assoications(关系属性) 为 AR 生成 relations 函数，

```
<?php

class <%= SourceEntity.Name %> extends CActiveRecord
{
    public static function model($className=__CLASS__)
    {
        return parent::model($className);
    }

    public function tableName()
    {
        return '<%= SourceEntity.GetSafeName() %>';
    }

    <%if (SourceEntity.Associations.Count>0){ %>
    public function relations()
    {
        return array(
          <% IEnumerable<IAssociation> associations = SourceEntity.Associations; %>
         <% foreach(IAssociation association in associations) { %>
         <% if(association.Entity.Name!=association.ForeignEntity.Name) {%>
            <% if (association.AssociationType == AssociationType.ManyToOne
                || association.AssociationType==AssociationType.ManyToZeroOrOne) { %>
            '<%= ToCameral(association.Name) %>'=>array(self::BELONGS_TO,
            '<%= association.ForeignEntity.Name %>',
            <%=GetBelongToKey(association) %>
            <% } %>
            <% if (association.AssociationType == AssociationType.OneToMany
                || association.AssociationType==AssociationType.ZeroOrOneToMany) { %>
            '<%= ToCameral(association.Name) %>'=>array(self::HAS_MANY,
            '<%= association.ForeignEntity.Name %>',
            <%=GetKey(association) %>
            <% } %>
            <% if (association.AssociationType == AssociationType.OneToOne
                || association.AssociationType==AssociationType.OneToZeroOrOne) { %>
            '<%= ToCameral(association.Name) %>'=>array(self::HAS_ONE,
            '<%= association.ForeignEntity.Name %>',
            <%=GetKey(association) %>
            <% } %>
            <% if (association.AssociationType == AssociationType.ManyToMany) { %>
```

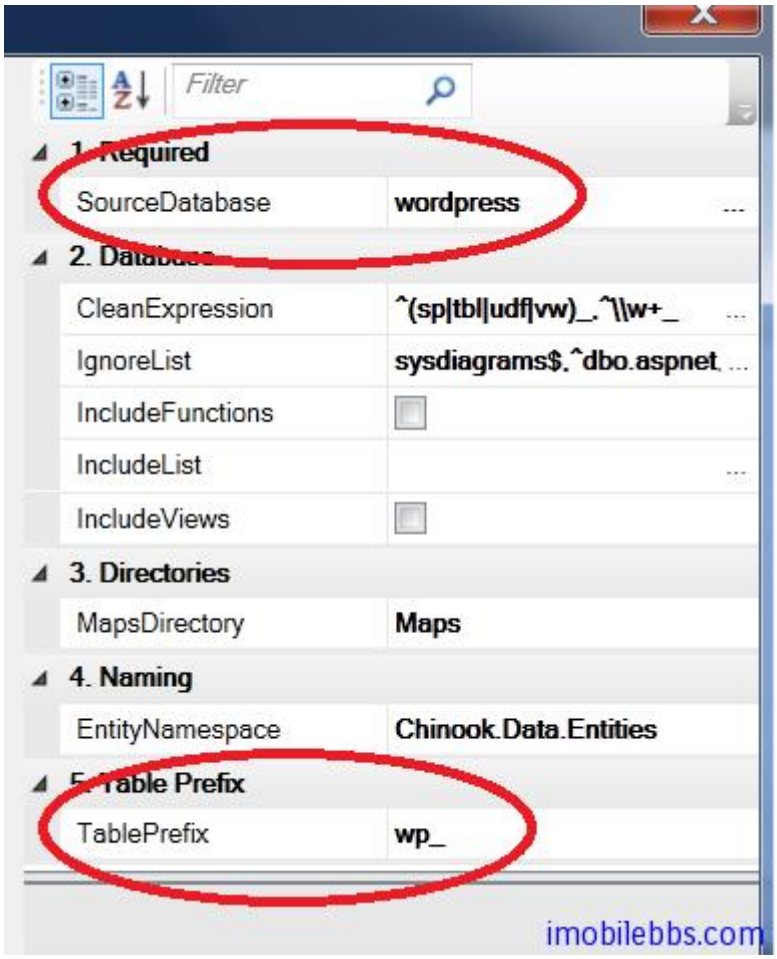
```
        '<%= ToCameral(association.Name) %>'=>array(self::MANY_MANY,
        '<%= association.IntermediaryAssociation.Entity.Name %>',
        <%=GetManyToManyKey(association) %>
        <% } %>
      <% } %>
   <% } %>
     );
  }
  <% } %>
}

?>

<script runat="template">

public string ToCameral(string name)
{
  return StringUtil.ToCamelCase(name);
 }

public string GetKey(IAssociation association)
{
  string retString=string.Empty;

  if(association.Properties.Count>1)
  {
    retString="array(";
    foreach (AssociationProperty associationProperty in association.Properties)
    {
      retString+="'"+associationProperty.ForeignProperty.GetSafeName()+"',";
    }
    retString+="),";
  }else{
    foreach (AssociationProperty associationProperty in association.Properties)
    {
      retString+="'"+associationProperty.ForeignProperty.GetSafeName()+"'),";
    }

  }
  return retString;
}

public string GetBelongToKey(IAssociation association)
{
  string retString=string.Empty;
```

```
    if(association.Properties.Count>1)
    {
        retString="array(";
        foreach (AssociationProperty associationProperty in association.Properties)
        {
            retString+="'"+associationProperty.Property.GetSafeName()+"',";
        }
        retString+="),";
    }else{
        foreach (AssociationProperty associationProperty in association.Properties)
        {
            retString+="'"+associationProperty.Property.GetSafeName()+"'),";
        }

    }
    return retString;
}

public string GetManyToManyKey(IAssociation association)
{

    string retString="'"+association.ForeignEntity.GetSafeName()+"(";

    foreach (AssociationProperty associationProperty in association.Properties)
    {
        retString+=associationProperty.ForeignProperty.GetSafeName()+",";
    }
    IAssociation intermidateAssociation=association.IntermediaryAssociation;
    if(intermidateAssociation!=null)
    {
          foreach (AssociationProperty associationProperty in intermidateAssociation.Properties)
        {
            retString+=associationProperty.ForeignProperty.GetSafeName()+",";
        }
    }

    retString=retString.Substring(0,retString.Length-1);
    retString+=')),";
    return retString;
}
</script>
```

然后 generated output 就可以为数据库的表生成对应的 AR 类，比如生成的 Track 类

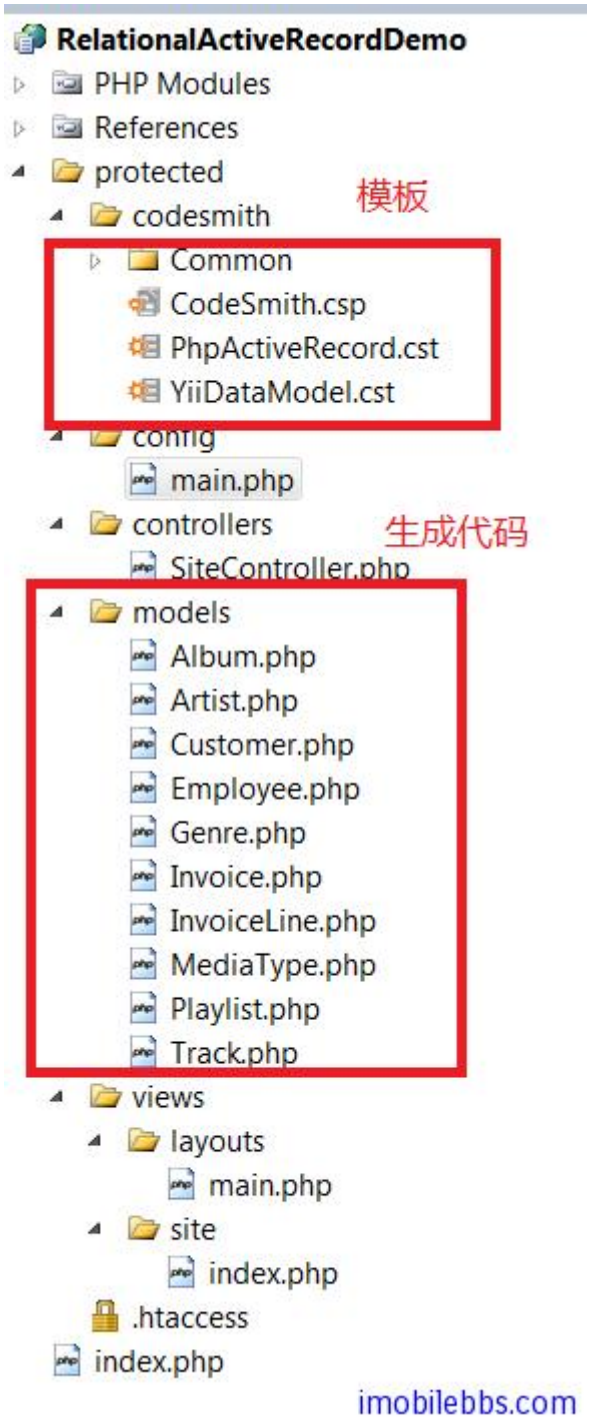
```
class Track extends CActiveRecord
{
    public static function model($className=__CLASS__)
    {
        return parent::model($className);
    }

    public function tableName()
    {
        return 'track';
    }

    public function relations()
    {
        return array(
            'album'=>array(self::BELONGS_TO,'Album','AlbumId'),
            'genre'=>array(self::BELONGS_TO,'Genre','GenreId'),
            'mediatype'=>array(self::BELONGS_TO,'Mediatype','MediaTypeId'),
            'invoicelines'=>array(self::HAS_MANY,'Invoiceline','TrackId'),
            'playlists'=>array(self::MANY_MANY,'Playlist','playlisttrack(TrackId,PlaylistId)'),
        );
    }
}
```

如果实在看不懂本例也无所谓，可以直接使用该模板，只要设置数据源 ，如果数据库的表有前缀，比如Wordpress 的表有 wp_ 可以设置表前缀（不是必须的）



图片 15.5 第30张

本例下载，如果需要使用本例的模板，直接把项目中 protected 下的codesmith 目录拷贝到你自己的项目中，然后为 codesmith.csp 配置数据源（或者还有表前缀），然后生成代码即可。



图片 15.6 第31张

本例下载

16

使用 XMLProperty

在前面 CodeSmith 使用教程(6): 基本语法–声明和使用属性介绍了 CodeSmith 中使用属性的基本方法，模板中的属性是通过 Property 指令来定义。

CodeSmith 也支持使用 XML 文档来定义属性，可以把一些配置属性定义到 XML 文件中，定义 XML 的属性是使用 XmlProperty 来定义：

```
<%@ XmlProperty Name="PurchaseOrder"
  Schema="PO.xsd"
  Optional="False"
  Category="Data"
  Description="Purchase Order to generate packing list for." %>
```

XmlProperty 指令可以有多个参数，除 Name 为必须的外，其它的参考都是可选的。

属性参数的介绍：

- Name：模版使用的参数的名称，必须为有效的模板语言名称，比如使用 C#，Name 必须为有效的 C#变量名。但提供 XML 的 Schema 文件时，这个变量的类型为一个 XmlDocument 实例。
- Schema：XML 属性对应的 Schema 文件名，可以用来校验存放 XML 属性的 XML 文件是否有效，如果提供了 Schema 文件，CodeSmith 在代码模板中支持 IntelliSense。
- Default：设置默认值。
- Category：用来说明这个属性在 CodeSmith Explorer 的属性面板中显示成什么类型，例如下拉选择、直接输入等。
- Description：在属性面板中对于这个属性的描述。
- Optional：设置这个属性是否是必须的，设置为 True 表明这个参数值可有可无，设置为 False 则这个参数必须有值。
- OnChanged 为属性发生变化时定义事件处理代码。
- RootElement： 指定 XML 根元素的相对路径。

本例使用 CodeSmith 自带的一个例子，使用 PurchaseOrder.xsd ，XML 的定义如下：

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema targetNamespace="http://www.codesmithtools.com/purchaseorder"
elementFormDefault="qualified"
xmlns="http://www.codesmithtools.com/purchaseorder"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="PurchaseOrder">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="OrderDate" type="xs:string" minOccurs="1" maxOccurs="1" />
```

```
      <xs:element name="SubTotal" type="xs:string" minOccurs="1" maxOccurs="1" />
      <xs:element name="ShipCost" type="xs:string" minOccurs="0" maxOccurs="1" />
      <xs:element name="TotalCost" type="xs:string" minOccurs="1" maxOccurs="1" />
      <xs:element name="ShipTo" minOccurs="0" maxOccurs="1">
       <xs:complexType>
        <xs:sequence>
         <xs:element name="Line1" type="xs:string" minOccurs="0" maxOccurs="1" />
         <xs:element name="City" type="xs:string" minOccurs="0" maxOccurs="1" />
         <xs:element name="State" type="xs:string" minOccurs="0" maxOccurs="1" />
         <xs:element name="Zip" type="xs:string" minOccurs="0" maxOccurs="1" />
        </xs:sequence>
        <xs:attribute name="Name" type="xs:string" />
       </xs:complexType>
      </xs:element>
      <xs:element name="Items" minOccurs="0" maxOccurs="1">
       <xs:complexType>
        <xs:sequence>
         <xs:element name="OrderedItem" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
           <xs:sequence>
            <xs:element name="ItemName" type="xs:string" minOccurs="1" maxOccurs="1" />
            <xs:element name="Description" type="xs:string" minOccurs="0" maxOccurs="1" />
            <xs:element name="UnitPrice" type="xs:string" minOccurs="1" maxOccurs="1" />
            <xs:element name="Quantity" type="xs:string" minOccurs="1" maxOccurs="1" />
            <xs:element name="LineTotal" type="xs:string" minOccurs="1" maxOccurs="1" />
           </xs:sequence>
          </xs:complexType>
         </xs:element>
        </xs:sequence>
       </xs:complexType>
      </xs:element>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
</xs:schema>
```

与这个 XML Schema 配合使用的用来存放 XML 属性的 XML 文件为 SamplePurchaseOrder.xml，其定义如下：

```
<?xml version="1.0"?>
<PurchaseOrder xmlns="http://www.codesmithtools.com/purchaseorder">
  <ShipTo Name="Eric J. Smith">
    <Line1>123 Test Dr.</Line1>
    <City>Dallas</City>
    <State>TX</State>
```

```
  <Zip>75075</Zip>
 </ShipTo>
 <OrderDate>05-01-2003</OrderDate>
 <Items>
  <OrderedItem>
   <ItemName>Item #1</ItemName>
   <Description>Item #1 Description</Description>
   <UnitPrice>5.45</UnitPrice>
   <Quantity>3</Quantity>
   <LineTotal>16.35</LineTotal>
  </OrderedItem>
  <OrderedItem>
   <ItemName>Item #2</ItemName>
   <Description>Item #2 Description</Description>
   <UnitPrice>12.75</UnitPrice>
   <Quantity>8</Quantity>
   <LineTotal>102.00</LineTotal>
  </OrderedItem>
 </Items>
 <SubTotal>45.23</SubTotal>
 <ShipCost>5.23</ShipCost>
 <TotalCost>50.46</TotalCost>
</PurchaseOrder>
```

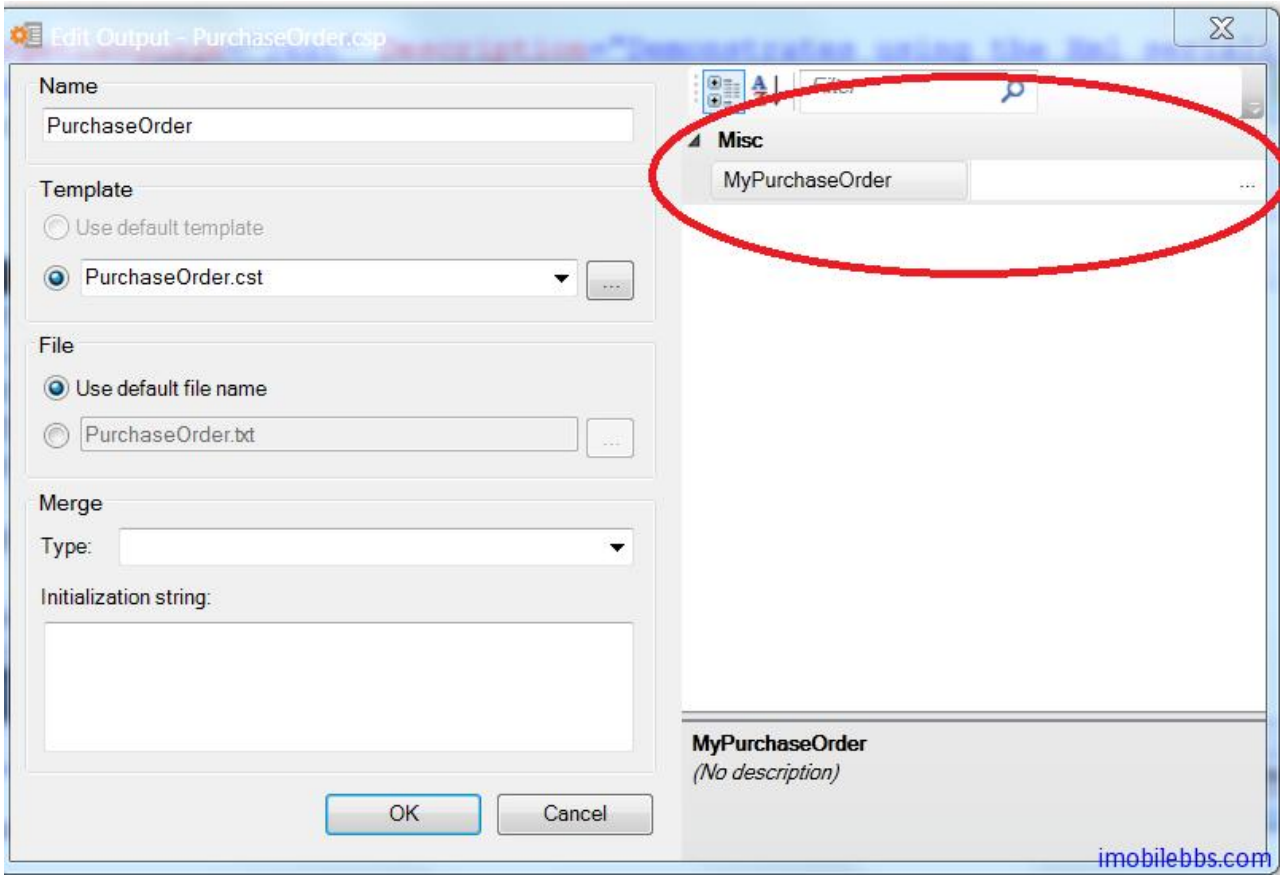
定义一个简单的模板，把 SamplePurchaseOrder.xml 中的内容重新输出，可以在代码模板中定义一个XMLProperty ，其 Schema 指定为 PurchaseOrder.xsd

```
<%--
This template demonstates using the XmlProperty directive
--%>
<%@ CodeTemplate Language="C#" TargetLanguage="Text"
  Description="Demonstrates using the Xml serializer." %>
<%@ XmlProperty
   Name="MyPurchaseOrder"
   Schema="PurchaseOrder.xsd"
   Default="SamplePurchaseOrder.xml" %>
This file generated by CodeSmith on <%= DateTime.Now.ToLongDateString() %>

PurchaseOrder:
    Address:
        Name: <%= MyPurchaseOrder.ShipTo.Name %>
        Line1: <%= MyPurchaseOrder.ShipTo.Line1 %>
        City: <%= MyPurchaseOrder.ShipTo.City %>
        State: <%= MyPurchaseOrder.ShipTo.State %>
        Zip: <%= MyPurchaseOrder.ShipTo.Zip %>
```

```
OrderDate: <%= MyPurchaseOrder.OrderDate %>
Items:
   <% for (int i = 0; i < MyPurchaseOrder.Items.Count; i++) { %>
   <%= i %>:
      ItemName: <%= MyPurchaseOrder.Items[i].ItemName %>
      Description: <%= MyPurchaseOrder.Items[i].Description %>
      UnitPrice: <%= MyPurchaseOrder.Items[i].UnitPrice %>
      Quantity: <%= MyPurchaseOrder.Items[i].Quantity %>
      LineTotal: <%= MyPurchaseOrder.Items[i].LineTotal %>
   <% } %>
SubTotal: <%= MyPurchaseOrder.SubTotal %>
ShipCost: <%= MyPurchaseOrder.ShipCost %>
TotalCost: <%= MyPurchaseOrder.TotalCost %>
```

模板中定义的 XML 属性名为 MyPurchaseOrder 对应的 Schema 为 PurchaseOrder.xsd，因此在代码模板可以通过 MyPurchaseOrder.ShipTo.Name 的格式来直接引用 XML Schema 中定义的元素，CoddSmith 也支持 IntelliSense。

运行该模板，首先需要为 MyPurchaseOrder 选择合适的 XML 文件：



图片 16.1 第32张

如果选择的文件不符合指定的 XML Schema，CodeSmith 不允许选择该文件，本例使用预先定义的SamplePurchaseOrder.xml ，生成的文件如下：

```
This file generated by CodeSmith on Saturday, 12 January 2013

PurchaseOrder:
   Address:
      Name: Eric J. Smith
      Line1: 123 Test Dr.
      City: Dallas
      State: TX
      Zip: 75075
   OrderDate: 05-01-2003
   Items:
      0:
         ItemName: Item #1
         Description: Item #1 Description
         UnitPrice: 5.45
         Quantity: 3
         LineTotal: 16.35
      1:
         ItemName: Item #2
         Description: Item #2 Description
         UnitPrice: 12.75
         Quantity: 8
         LineTotal: 102.00
   SubTotal: 45.23
   ShipCost: 5.23
   TotalCost: 50.46
```

本例下载

17 Merge 策略

前面介绍了 CodeSmith 使用的基本用法，通过代码模板来生成代码，但如果你修改了自动生成的代码，再次使用代码模板生成代码后，你修改的代码也就丢失了，CodeSmith 支持多种“合并（Merge）”来解决这个问题，以保留你自己修该过的部分。

CodeSmith 支持如下三种“合并策略”：

- InsertRegion Merge 策略
- PreserveRegions Merge 策略
- InsertClass Merge 策略

不过这些策略主要是针对 C#，VB 这些支持 Region 的语言，对于其它语言可能就需要使用其它方法，比如自定义 Merge 策略，CodeSmith 允许通过 CodeSmith.Engine.IMergeStrategy 来扩展“合并”策略，本人推荐 CodeSmith 的一个原因就是 CodeSmith 提供了很多接口而不仅仅是一个工具，比如除了 CodeSmith 支持的属性，XML 属性，你也可以通过 CodeSmith.CustomProperties 来自定义属性种类，除了 CodeSmith 支持的数据源种类（MySQL，Oracle），你也可以通过自定义的 Schema Provider 支持新的数据库类型或是其它数据类型。

#

InsertRegion 策略

InsertRegion 顾名思义，就是在源码中定义一个 Region，然后让 CodeSmith 自动生成的代码只插入到该区域，而在区域外的代码 CodeSmith 不会去碰它们，从而实现了自定义的代码和自动生成代码的合并。

#

PreserveRegion 策略

PreserveRegion 是定义多个区域，然后通知 CodeSmith 保持这些区域代码不变，自动创建的代码添加到这些区域的外面，和 InsertRegion 作用相反。

下面还是借用 CodeSmith 自带的 Merge 示例说明一下这两种策略的基本用法：

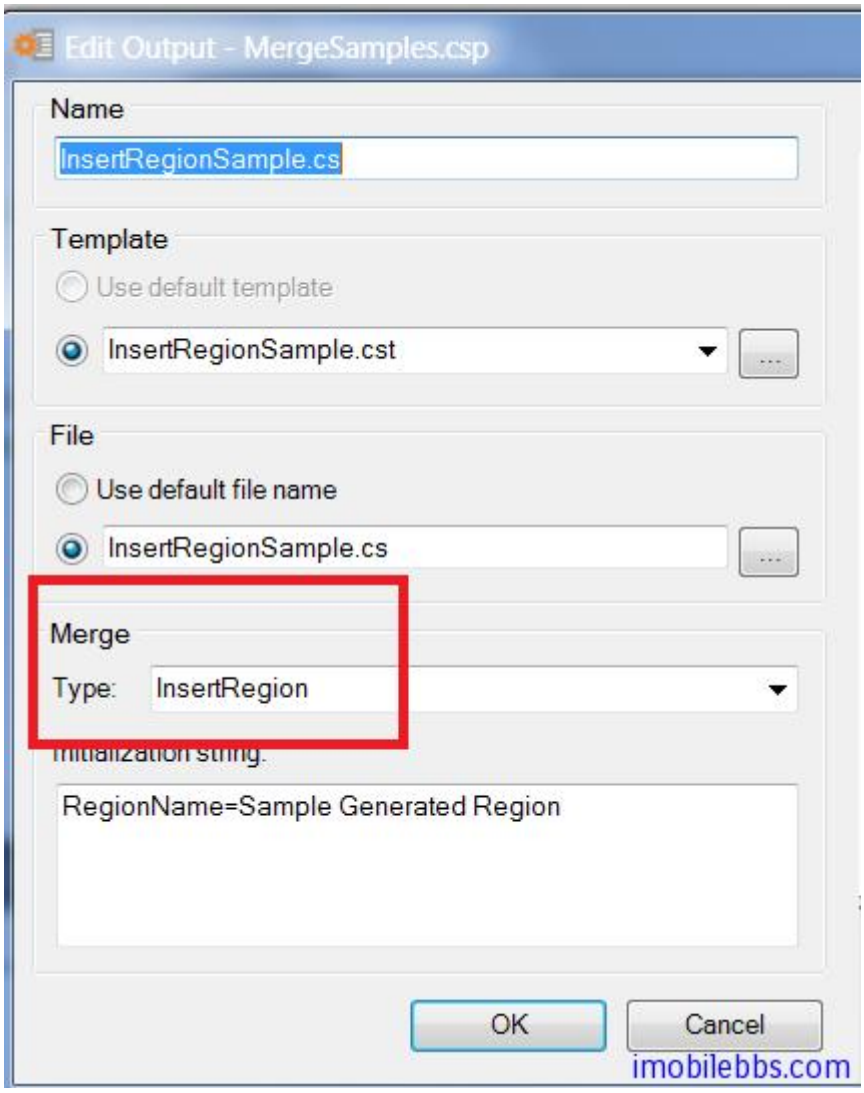
首先是 InsertRegion 策略，定义一个类文件 InsertRegionSample.cs

```
public class InsertRegionsSample
{

    public void SomeCustomMethod()
    {
        // This is my custom code that I want to preserve.
        // I can make changes to it and my changes will
        // not be overwritten.
    }

    #region Sample Generated Region
// This region generated by CodeSmith on Saturday, 12 January 2013
    #endregion
}
```

其中定义了一个 Region，名为 Sample Generated Region ，准备让 CodeSmith 查入代码，编写一个简单的代码模板，插入当前时间：

```
<%@ Template Language="C#" TargetLanguage="C#"   Description="Demonstrates using an InsertRegion merge strate
// This region generated by CodeSmith on <%= DateTime.Now.ToLongDateString() %>
```

然后通过 CodeSmith 项目为模板设置 Merge 策略：



图片 17.1 第33张

选择 InsertRegion 策略， 然后设置要插入的 RegionName。

生成后的代码如下：

```
public class InsertRegionsSample
{

    public void SomeCustomMethod()
    {
        // This is my custom code that I want to preserve.
        // I can make changes to it and my changes will
        // not be overwritten.
    }

    #region Sample Generated Region
// This region generated by CodeSmith on Saturday, 12 January 2013
```

```
    #endregion
}
```

可以看到 CodeSmith 只在 Region 处插入代码，而该 Region 外的部分保持不变。

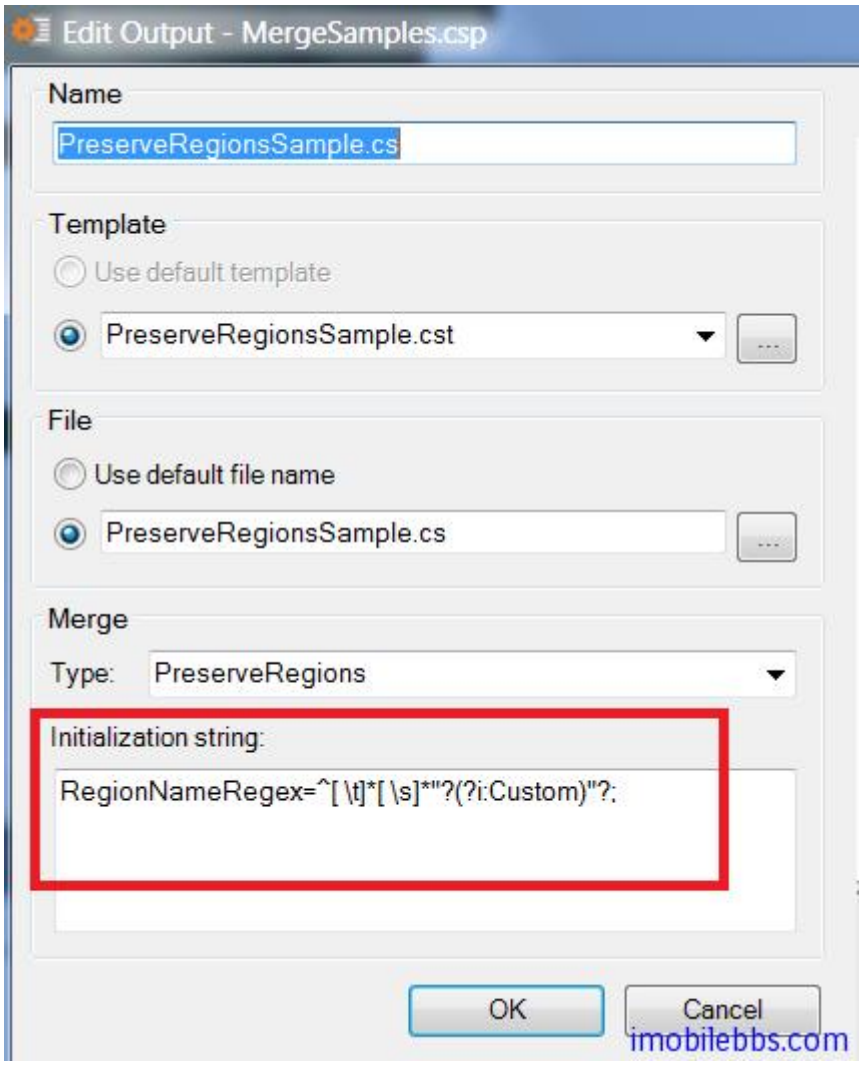
类似的 PreserveRegions 策略，代码和模板定义如下： PreserveRegionsSample.cs

```
public class PreserveRegionsSample
{

 #region "Custom Region 1"


    // This is a place holder for your custom code.
    // It must exist so that CodeSmith knows where
    // to put the custom code that will be parsed
    // from the target source file.
    // The region name is used to match up the regions
    // and determine where each region of custom code
    // should be inserted into the merge result.

 #endregion


    public void SomeGeneratedMethod()
    {

        // This section and all other non-custom code
        // regions will be overwritten during each
        // template execution.
        // Current Date: Saturday, 12 January 2013
    }

 #region "Custom Region 2"


    // The contents of this region will also be preserved
    // during generation.

 #endregion


}
```

模板定义如下：

```
<%@ Template Language="C#" TargetLanguage="C#"    Description="Demonstrates using a PreserveRegions merge st
public class PreserveRegionsSample
{

 #region "Custom Region 1"


   // This is a place holder for your custom code.
   // It must exist so that CodeSmith knows where
   // to put the custom code that will be parsed
   // from the target source file.
   // The region name is used to match up the regions
   // and determine where each region of custom code
   // should be inserted into the merge result.

 #endregion


   public void SomeGeneratedMethod()
   {

      // This section and all other non-custom code
      // regions will be overwritten during each
      // template execution.
      // Current Date: <%= DateTime.Now.ToLongDateString() %>
   }

 #region "Custom Region 2"


   // The contents of this region will also be preserved
   // during generation.

 #endregion


}
```

模板中也定义了两个区域，然后为该模板设置 Merge 策略，使用 PreserveRegion 时可能有多个Region 需要保留，因此可以使用 RegX 来定义要保留的 Region：



图片 17.2 第34张

本例下载

InsertClass 策略用在给以重载的代码中插入自动生成的代码,挺起来和 InsertRegion 功能很类似，的确也是如此，但 InsertClass 支持更多的配置，可以实现更加灵活和强大的功能。

它支持的配置有：

Language	String, Required	只支持VB和C#
ClassName	String, Required	需插入代码的类名.
PreserveClassAttributes	Boolean, defaults to False	是否保留类已有的Attributes，缺省CodeSmith替代类原来的Attributes
OnlyInsertMatchingClass	Boolean, defaults to False	是否只插入匹配的类定义中
MergeImports	Boolean, defaults to False	是否合并Import语句

Language	String, Required	只支持VB和C#
NotFoundAction	Enum, defaults to None	如果指定的类没找到后的行动，可以None,InsertAtBottom，InsertInParent几种选项
NotFoundParent	String, no default	如果指定NotFoundAction为InsertInParent对应的父类名称.

比如使用如下配置：

Language: C# ClassName: “Pet” PreserveClassAttributes: True OnlyInsertMatchingClass: True MergeImports: True

现有类定义：

```
using System;
using System.ComponentModel.DataAnnotations;
namespace Petshop
{
   [ScaffoldTable(true)]
   public class Pet
   {
      public int Age { get; set; }
      public string FirstName { get; set; }
      public string LastName { get; set; }
   }
}
```

自动生成的代码如下：

```
using System;
using System.Text;
namespace Petshop
{
   public class Pet
   {
      public string FirstName { get; set; }

      public string LastName { get; set; }

      public string FullName
      {
         get { return String.Format("{0} {1}", FirstName, LastName); }

      }
   }
}
```

使用 InsertClass 合并后的代码如下：

```
using System;
using System.ComponentModel.DataAnnotations;
using System.Text;
namespace Petshop
{
   [ScaffoldTable(true)]
   public class Pet
   {
      public string FirstName { get; set; }

      public string LastName { get; set; }

      public string FullName
      {
         get { return String.Format("{0} {1}", FirstName, LastName); }

      }
   }
}
```