# 代码生成器原理

* 连接数据库并获取某个表的列名、值类型、是否为空等元信息。
* 转换数据库类型为c#类型，如个类型为值类型，并可为空则在其类型后加？
* 拼接属性字符串
* 读取模板文件并替换类名为表名，属性占位符为上一步拼接的字符串。

## 测试能否连接数据库

在sqlhelper中添加如下代码

|  |
| --- |
| public override bool isConnected()  {  using (SqlConnection conn = getConn())  {  try  {  conn.Open();  }  catch  {  return false;  }  }  return true;  } |

连接能顺利打开则返回true，否则返回false。

## 加载表名到listbox

|  |
| --- |
| private void button1\_Click(object sender, EventArgs e)  {  lbtalbeName.Items.Clear();  connStr = txtConnectionString.Text.Trim();  SqlHelper sqlhelper = new MSSqlHelper(connStr);  if (sqlhelper.isConnected())  {  string sql = "select TABLE\_NAME from INFORMATION\_SCHEMA.TABLES";  SqlDataReader reader = sqlhelper.ExecuteReader(sql);  if (reader.HasRows)  {  while (reader.Read())  {  lbtalbeName.Items.Add(reader.GetString(0));  }  }  }  else  {  MessageBox.Show("==Unknown");  }  } |

## 创建模板

创建名为“Temple.cs”的模板文件

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace @nameSpace  {  class @ClassName  {  @modeL  }  } |

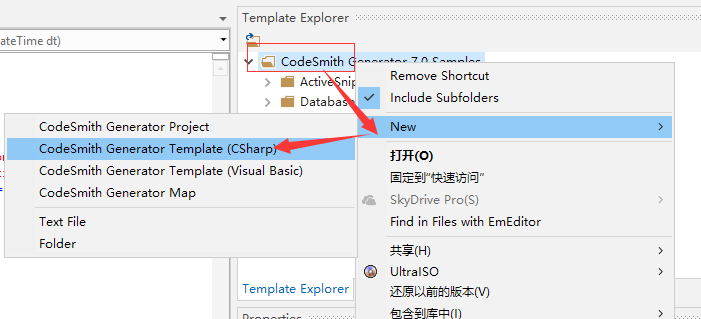
## 生成代码

|  |
| --- |
| /// <summary>  /// 生成代码  /// </summary>  /// <param name="sender"></param>  /// <param name="e"></param>  private void button2\_Click(object sender, EventArgs e)  {  if (lbtalbeName.SelectedItem == null)  {  MessageBox.Show("请选择要创建model的表");  return;  }  else  { //读取模板并替换命名空间和类名。  tableName = lbtalbeName.SelectedItem.ToString();  }  if (txtNameSpace.Text == "")  {  MessageBox.Show("请输入命名空间的名称");  return;  }  List<ColumnInfo> list = new List<ColumnInfo>();  string temple = File.ReadAllText("Temple.cs");  temple = temple.Replace("@nameSpace", txtNameSpace.Text.Trim());  temple = temple.Replace("@ClassName", tableName);  //查询某一个表（tableName）中包含的列名、值类型以及是否可为空。  string sql = "select COLUMN\_NAME,IS\_NULLABLE,DATA\_TYPE from INFORMATION\_SCHEMA.COLUMNS where TABLE\_NAME =@tname ";  SqlParameter sp = new SqlParameter("@tname", tableName);  SqlHelper sqlhelper = new MSSqlHelper(connStr);  SqlDataReader reader = sqlhelper.ExecuteReader(sql, sp);  if (reader.HasRows)  {  int columnName = reader.GetOrdinal("COLUMN\_NAME");  int nullAble = reader.GetOrdinal("IS\_NULLABLE");  int dataType = reader.GetOrdinal("DATA\_TYPE");  while (reader.Read())  {  //读取信息到ColumnInfo对象中。  ColumnInfo ci = new ColumnInfo();  ci.ColumnName = reader.GetString(columnName);  ci.DataType = reader.GetString(dataType);  ci.NullAble = reader.GetString(nullAble);  list.Add(ci);  }  }  //拼接字符串  StringBuilder sb = new StringBuilder();  foreach (ColumnInfo item in list)  {  sb.AppendLine("\t\tpublic\t" + GetCSharpTypeByDBType(item) + "\t" + ConvertFirstLetterToUpper(item) + " { get; set; }");  }  //替换模板中的@modeL  temple = temple.Replace("@modeL", sb.ToString());  txtCode.Text = temple;  }  private string GetCSharpTypeByDBType(ColumnInfo item)  {  string str = "";  switch (item.DataType)  {  case "char":  case "nchar":  case "varchar":  case "nvarchar":  case "text":  case "ntext":  str = "string";  break;  case "int":  str = "int";  break;  }  Type type = str.GetType();  if (type != null)  {  if (item.NullAble.ToLower() == "yes" && type.IsValueType)  {  str += "?";  }  }  return str;  }  /// <summary>  /// 首字母转换为大写  /// </summary>  /// <param name="item"></param>  /// <returns></returns>  private string ConvertFirstLetterToUpper(ColumnInfo item)  {  return char.ToUpper(item.ColumnName[0]).ToString() + item.ColumnName.Substring(1);  } |

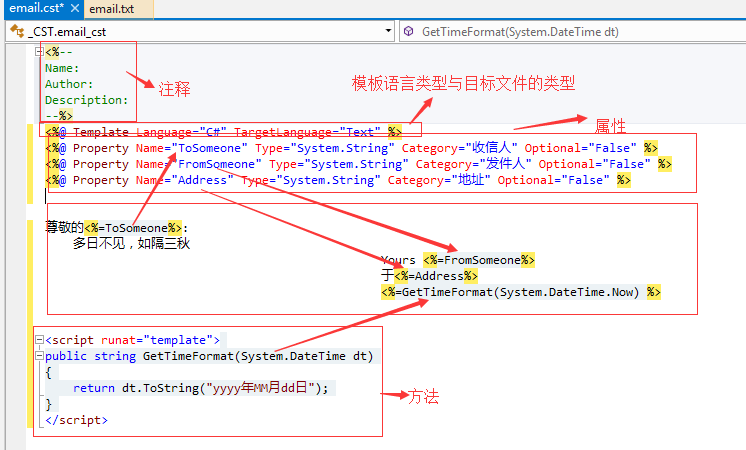
# CodeSmith自定义模板

## 新建模板文件

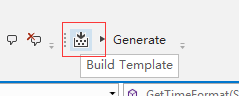
在CodeSmith Generator 7.0 Samples上右击创建cSharp代码类型的模板



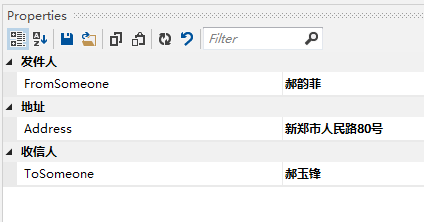
## 编写模板文件



## 创建模板



## 属性面板中填写内容



## 生成代码



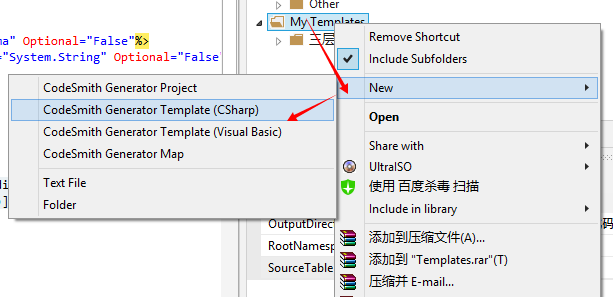
## 最终效果



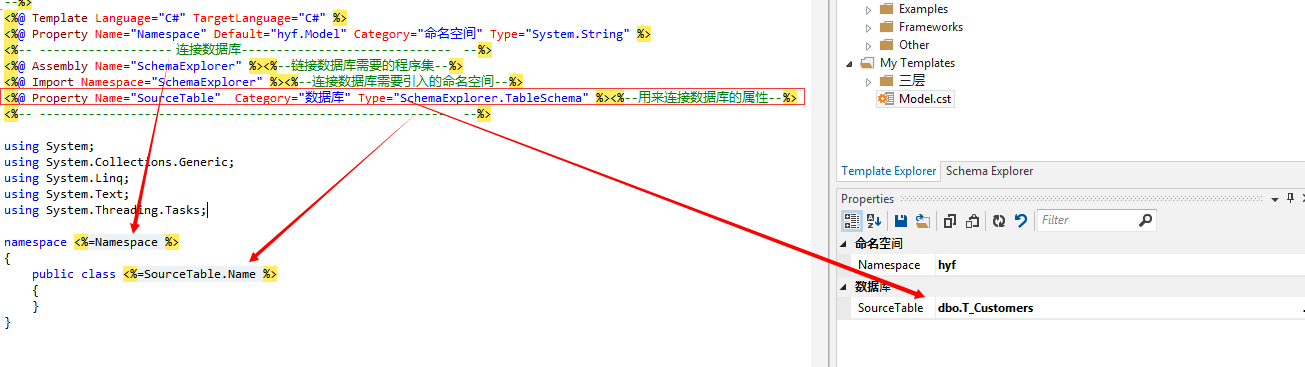
# 代码生成器写Model

## 创建模板文件

在文件夹上右击，按照以下步骤创建模板



## 链接数据库



## 命名空间和类名的替换

|  |
| --- |
| namespace <%=Namespace %> { public class <%=SourceTable.Name %> { } } |

## 转换类名的首字母为大写

|  |
| --- |
| <script runat="template"> public string ConvertFirstLetterToUpper(string str) { return str.Substring(0,1).ToUpper()+str.Substring(1); }</script> |
| public class <%=ConvertFirstLetterToUpper(SourceTable.Name) %> |

## 转换数据库类型为C#类型

|  |
| --- |
| public string ConvertDBTypeToCSharpType(ColumnSchema column) { string type; switch(column.DataType) { case DbType.AnsiString: type ="string";break; case DbType.AnsiStringFixedLength: type ="string";break; case DbType.Binary: type="byte[]";break; case DbType.Boolean: type="bool";break; case DbType.Byte: type="byte";break; case DbType.Currency: type="decimal";break; case DbType.Date: type="DateTime";break; case DbType.DateTime: type="DateTime";break; case DbType.Decimal: type="decimal";break; case DbType.Double: type="double";break; case DbType.Guid: type="Guid";break; case DbType.Int16: type="short";break; case DbType.Int32: type="int";break; case DbType.Int64: type="long";break; case DbType.Object: type="object";break; case DbType.SByte: type="sbyte";break; case DbType.Single: type="float";break; case DbType.String: type="string";break; case DbType.StringFixedLength: type="string";break; case DbType.Time: type="TimeSpan";break; case DbType.UInt16: type="ushort";break; case DbType.UInt32: type="uint";break; case DbType.UInt64: type="ulong";break; case DbType.VarNumeric: type="decimal";break; default : { type="\_\_UnKNWON\_\_"+ column.DataType.ToString(); break; } if(column.AllowDBNull && column.SystemType.IsValueType) { type +="?"; } } return type; } |

## 遍历表中列的数量，为每一个表创建属性

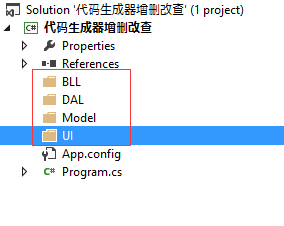
|  |
| --- |
| <% for(int i=0; i<SourceTable.Columns.Count;i++){ %> public <%= ConvertDBTypeToCSharpType(SourceTable.Columns[i]) %> <%=ConvertFirstLetterToUpper(SourceTable.Columns[i].Name) %> { get; set; } <%} %> |

## 完整代码

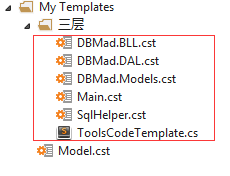
|  |
| --- |
| <%-- Name: Author: Description: --%><%@ Template Language="C#" TargetLanguage="C#" %><%@ Property Name="Namespace" Default="hyf.Model" Category="命名空间" Type="System.String" %><%-- -------------------连接数据库------------------------------ --%><%@ Assembly Name="SchemaExplorer" %><%--链接数据库需要的程序集--%><%@ Import Namespace="SchemaExplorer" %><%--连接数据库需要引入的命名空间--%><%@ Property Name="SourceTable" Category="数据库" Type="SchemaExplorer.TableSchema" %><%--用来连接数据库的属性--%><%-- ---------------------------------------------------------- --%>using System;using System.Collections.Generic;using System.Linq;using System.Text;using System.Threading.Tasks; namespace <%=Namespace %> { public class <%=ConvertFirstLetterToUpper(SourceTable.Name) %> { <% for(int i=0; i<SourceTable.Columns.Count;i++){ %> public <%= ConvertDBTypeToCSharpType(SourceTable.Columns[i]) %> <%=ConvertFirstLetterToUpper(SourceTable.Columns[i].Name) %> { get; set; } <%} %> } }<script runat="template"> public string ConvertFirstLetterToUpper(string str) { return str.Substring(0,1).ToUpper()+str.Substring(1); } public string ConvertDBTypeToCSharpType(ColumnSchema column) { string type; switch(column.DataType) { case DbType.AnsiString: type ="string";break; case DbType.AnsiStringFixedLength: type ="string";break; case DbType.Binary: type="byte[]";break; case DbType.Boolean: type="bool";break; case DbType.Byte: type="byte";break; case DbType.Currency: type="decimal";break; case DbType.Date: type="DateTime";break; case DbType.DateTime: type="DateTime";break; case DbType.Decimal: type="decimal";break; case DbType.Double: type="double";break; case DbType.Guid: type="Guid";break; case DbType.Int16: type="short";break; case DbType.Int32: type="int";break; case DbType.Int64: type="long";break; case DbType.Object: type="object";break; case DbType.SByte: type="sbyte";break; case DbType.Single: type="float";break; case DbType.String: type="string";break; case DbType.StringFixedLength: type="string";break; case DbType.Time: type="TimeSpan";break; case DbType.UInt16: type="ushort";break; case DbType.UInt32: type="uint";break; case DbType.UInt64: type="ulong";break; case DbType.VarNumeric: type="decimal";break; default : { type="\_\_UnKNWON\_\_"+ column.DataType.ToString(); break; } if(column.AllowDBNull && column.SystemType.IsValueType) { type +="?"; } } return type; } </script> |

# 三层与代码生成器

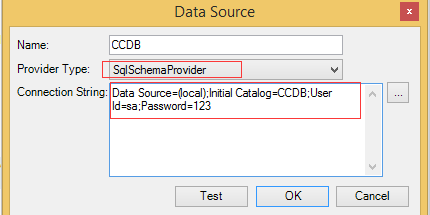
## 用文件夹创建三层架构



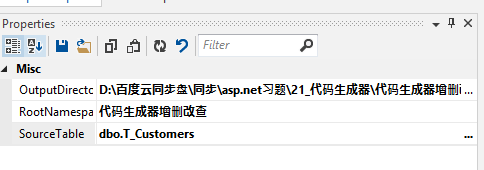
## CodeSmith导入三层模板



## 设置数据库



## 在Main.cst中设置输出位置、命名空间、数据库表

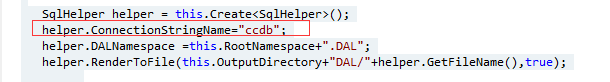


设置OutputDirectory：为3.1节创建的项目所在目录。

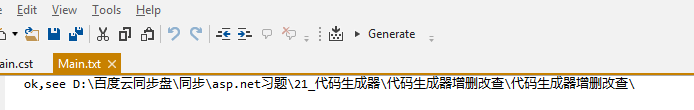
RootNamespace：为“代码生成器增删改查”（即项目中默认的命名空间）。

SourceTalbe：为将要创建三层的表

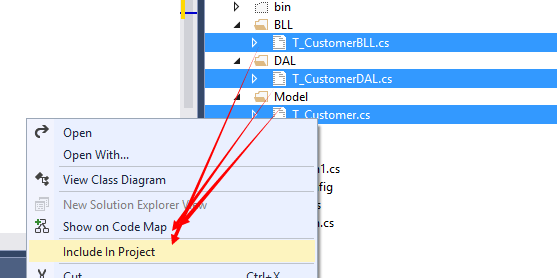
## 设置连接字符串



## 生成代码



## 把创建的文件包含到项目中



## 设置config



## 绑定数据到DataGridView

在winfrom中添加DataGridView。在窗体加载时绑定数据

|  |
| --- |
| private void Form1\_Load(object sender, EventArgs e)  {  T\_CustomerBLL bll = new T\_CustomerBLL();  IEnumerable<T\_Customer> list = bll.GetAll();  dataGridView1.DataSource = list;  } |

## 删除数据

|  |
| --- |
| private void button1\_Click(object sender, EventArgs e)  {  int id = (int)dataGridView1.SelectedRows[0].Cells[0].Value;  T\_CustomerBLL bll = new T\_CustomerBLL();  bll.DeleteByCC\_AutoId(id);  MessageBox.Show("OK");  } |