代码生成器使用手册

CodeBuilder User Manuals

V2.5.0

1. 概述

CodeBuilder 是一款功能强大的代码生成工具。它能将你所设计的数据库结构转换成你所想要的任何文本形式的文件，如 Java、C#、VB 等代码文件，以及 SQL 脚本、数据库设计文档等。通过开发插件，你甚至可将其转换成 Word、PDF 等二进制文件。

CodeBuilder 基于插件式、开放式的思想，你通过实现其定义的接口，就可轻松地将你所开发的插件集成到 CodeBuilder 中来。目前 CodeBuilder 提供了数据源、模板以及工具三类接口。CodeBuilder 基于.NET Framework 4.0 开发，它使用了动态编译技术，你可以嵌入 C#或 VB.NET 代码对对象属性进行个性化的扩展，结合灵活的模板生成你所想要的任何代码。

数据源(ISourceProvider)。提供数据结构的来源，目前提供数据库结构、PowerDesign 设计文档两种数据源。数据库结构基于 Fireasy 的 SchemaProvider，目前支持 SQLite、MsSql(SqlServer)、MySql、PostgreSql、Oracle、Firebird 等数据库，以及 OleDb、Odbc 驱动。

模板(ITemplateProvider)。提供代码生成的模板，目前提供 Razor 和 T4 两种模板。你可以自行编写符合自己的模板来生成你所想要的代码。

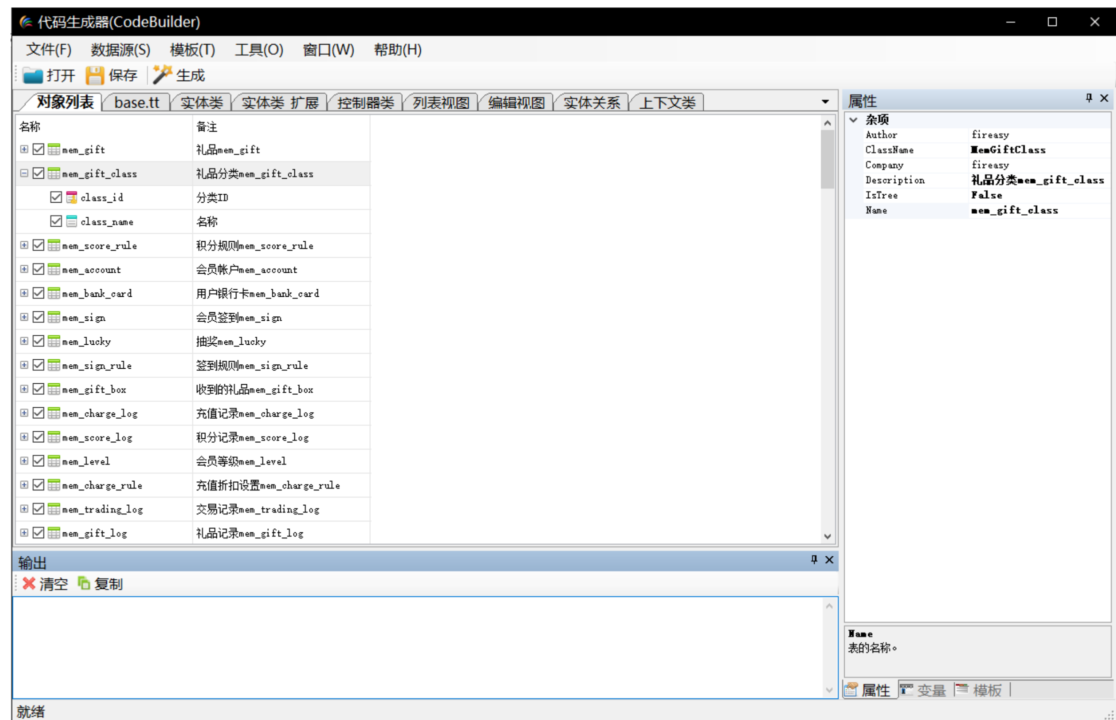
工具(IToolProvider)。工具是一些常用的小程序，你可以自己开发其后集成到 CodeBuilder 里进行使用。

CodeBuilder 版权归帆影易动力(fireasy.cn)所有，目前开源于 github，如果你感兴趣，可到 https://www.github.com/faib920/codebuilder 下载查阅，希望你给予更好的反馈和建议。

2. 使用说明

2.1. 初识 CodeBuilder

打开代码生成器后，界面如下：



界面分为三个区，右侧分别为属性窗口、变量窗口和模板窗口。中间为对象列表窗口、以及模板文件编辑窗口、代码生成预览窗口等。下方为调试输出区。

认识了工具的界面后，下面我将带领你一步一步生成你所需要的代码。

2.2. 配置数据源

数据源主要是指数据库结构，包括数据表、字段、关系等，其架构如下(蓝色的项表示是由工具自动生成的，橙色的项表示是隐藏的，属性窗口里看不到，但是模板里可以使用)。

数据表(Table)：

名称	说明	类型
Name	表的名称	字符串
Description	表的说明	字符串
ClassName	生成的类的名称	字符串
Columns	字段集	集合
SubKeys	一对多的关系集	集合
ForeignKeys	多对一的关系集	集合

PrimaryKeys	主键集	集合

字段(Column)：

名称	说明	类型
Name	字段的名称	字符串
Description	字段的说明	字符串
IsPrimaryKey	是否是主键	布尔
AutoIncrement	是否自增长	布尔
IsNullable	是否可空	布尔
DbType	标准的数据类型，如 String	DbType
DataType	数据类型，如 varchar2	字符串
Length	长度(字符型有效)	整型
Precision	精度	整型
Scale	小数位	整型
DefaultValue	默认值	字符串
ForeignKey	外键	关系
PropertyType	属性类型	字符串
PropertyName	属性名称	字符串
Owner	所性的数据表	数据表

关系(Reference)：

名称	说明	类型
Name	键的名称	字符串
OnDelete	删除时如何处理	0 限制 1 层级 2 设为 NULL
OnUpdate	更新地如何处理	0 限制 1 层级 2 设为 NULL
PkTable	主表	数据表
PkColumn	主表的主键	字段
FkTable	子表	数据表
FkColumn	子表的外键	字段

默认可以使用 Database 和 Power Design 两种方式提供数据源。Database 即基于 Database Provider 方式，由 Fireasy.Data 的 Provider 提供，目前支持 SQLite、MsSql、MySql、PostgreSql、Oracle、Firebird 等常见的数据库，另外还可以使用 OleDb 和 Odbc 驱动适配更多类型的数据，如果你能力足够好，甚至可以自定义除此之外的其他数据库。库。Power Design 则是允许你从 Power Design 的设计文档(pdm)中导入数据库结构。

(1) 使用 Database

选择菜单“数据源”—“Database”，打开数据源管理窗口，如下。

这里列出了你所配置的所有数据源，双击其中的某一项，或选中某一项，然后点击工具栏“选择”按钮即可导入该数据源的结构到“对象列表”中。

点击工具栏“添加”按钮，从下拉菜单中选择你所需要的数据库类型。

在弹出的窗口中，输入数据源名称，以及数据库连接串，你也可以单击“向导”按钮，使用特定的连接串配置向导进行配置。

单击“测试”按钮，测试数据库是否连接成功，当数据库无法连接时你再尝试修改你的连接字符串。

配置好数据源后，双击打开数据源，经过数秒的加载后，数据源中的所有表将加载到以下的窗体中，此时勾选需要生成的表，注意如果有关系，那么关联的表也需要一起勾选过去，否则生成的代码中可能缺失部分内容。

(2) 使用 Power Design

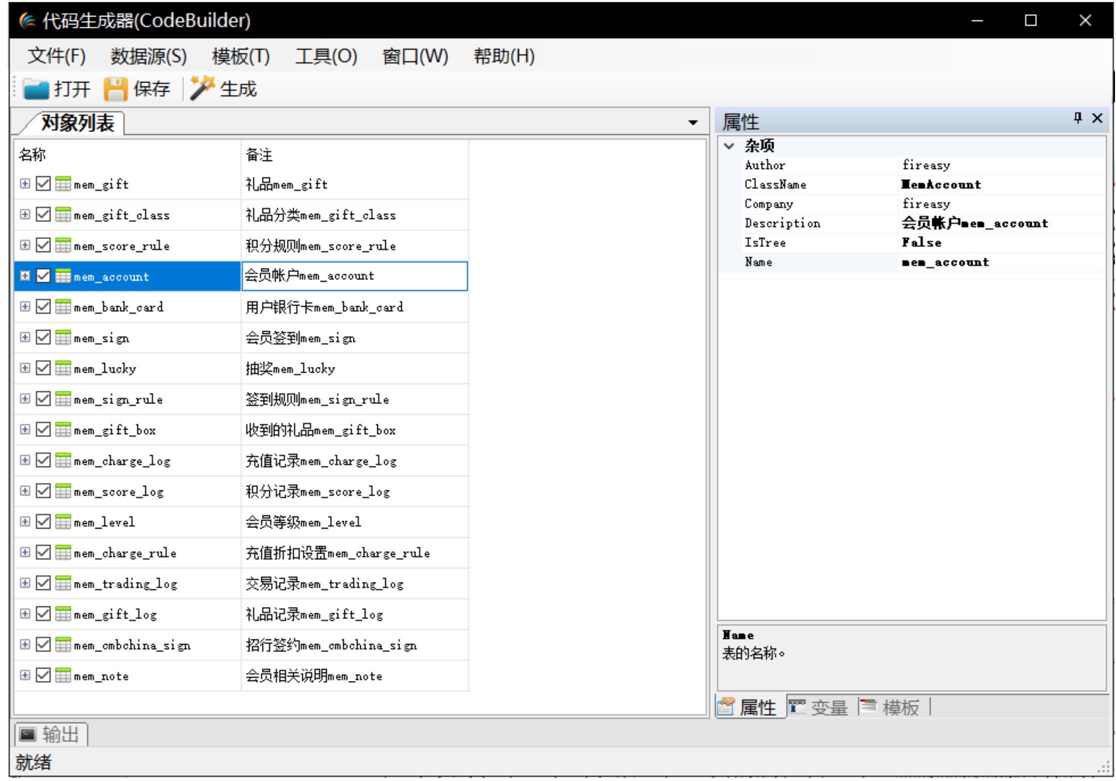
选择菜单“数据源”—“Power Design”，定位到 Power Design 文档，双击打开，如下图：

如果 Power Design 设计文档比较复杂，建议使用包进行划分。

2.3. 调整参数

(1) 对象列表

从数据源中获取到数据库结构后，显示在如下的“对象列表”窗口中，如下图：



对象列表显示了所选择的表，以及表的所有字段。右键菜单可以选择要生成的表及字段，也可以通过“生成预览”查看生成后的代码样式。

对象列表中的备注列可以单击进行修改。

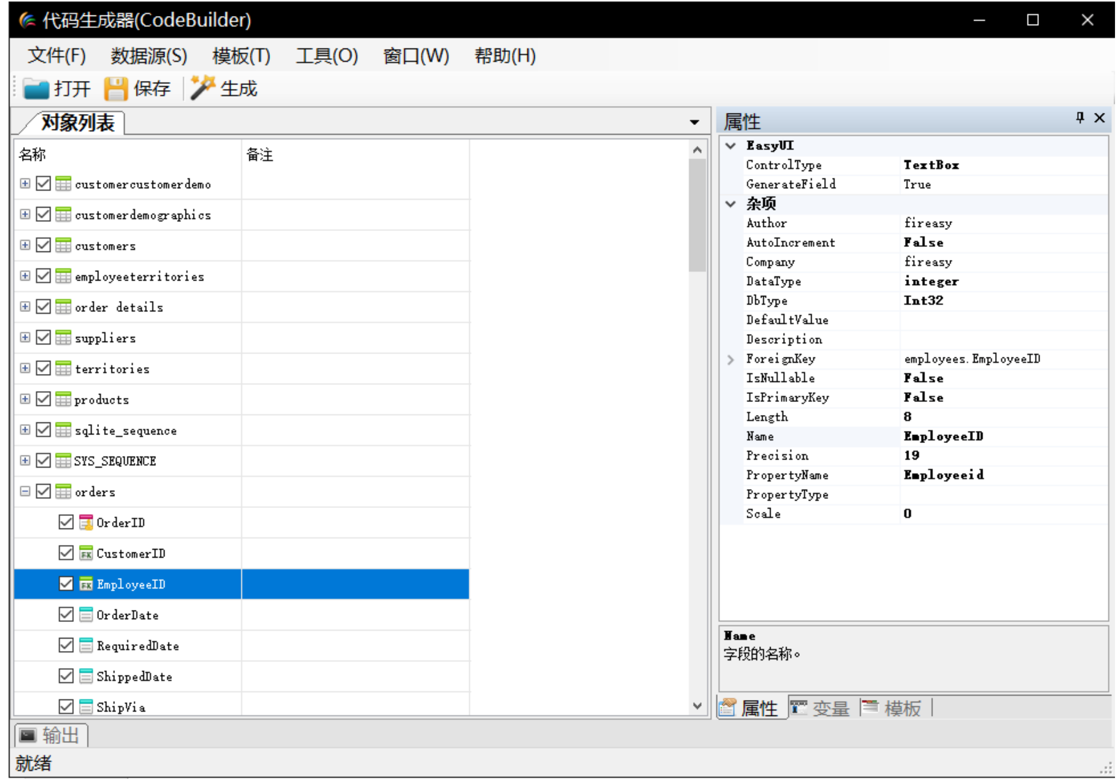
(2) 属性窗口

在对象列表中，单击表后，在右侧的属性窗口中，显示出表的所有属性；单击对象列表中的“备注”列，可以修改其内容，修改后立即在属性窗口中同步显示。

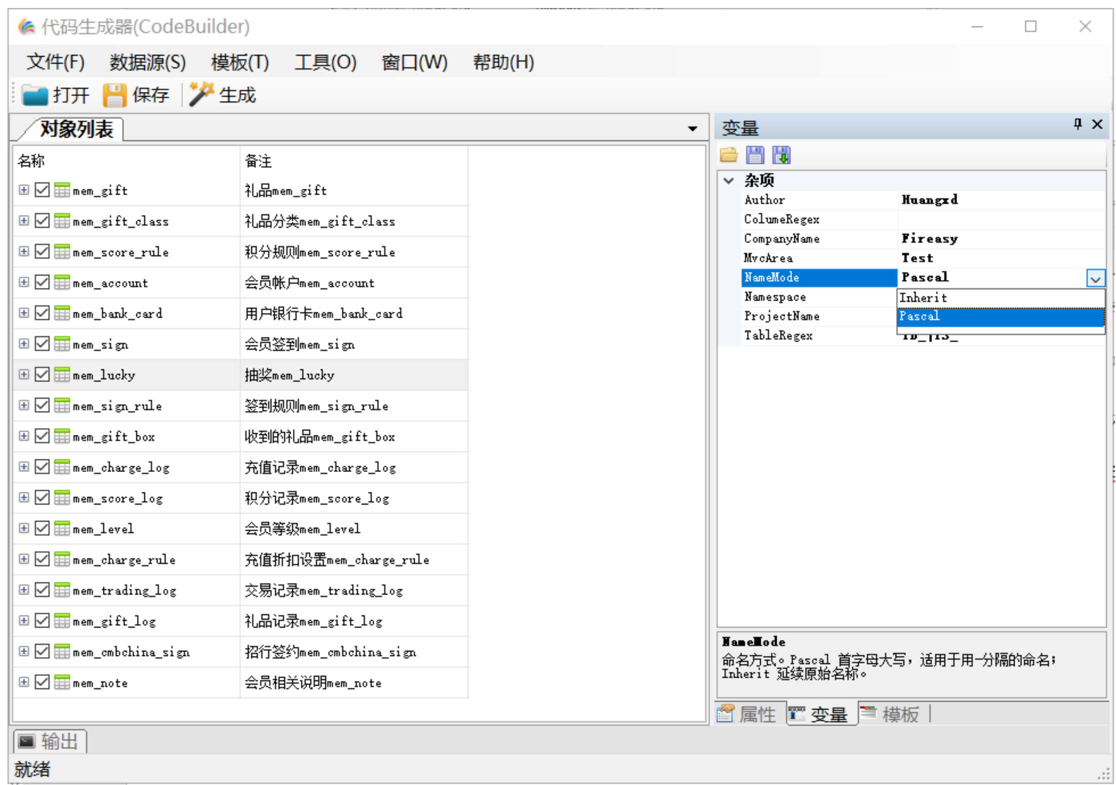
表的属性不是太多，只要注意一下 ClassName 的生成是否正确即可。

在对象列表中，展开节点树，显示表的所有字段，从节点图标中可以区分出，为主键，为外键。单击字段后，在右侧的属性窗口中，显示字段的所有属性。

需要注意的是 IsPrimaryKey 和 AutoIncrement 是否与表结构中的一致。注意 DbType 是否与 DataType 对应。DbType 是一个标准的枚举集，而 DataType 则是每一种数据库的数据类型，如 MsSql 的 varchar、Oracle 的 varchar2、SQLite 的 text，均是对应到 DbType 中的 String。除此这外，还应检查 PropertyType 及 PropertyName 是否正确。



需要注意的是 ClassName 和 PropertyName，默认使用 Pascal 规则进行命名，它与 Name 是不相同的。如以上的 mem_lucky 表，生成的 ClassName 是 MemLucky，如果你的 ORMapper 不没有此种映射，此时需要进行调整。如下图，右侧切换至变量窗口。



NameMode 提供了两种命名方式，Inherit 继承方式，即生成的 ClassName 和 Name 保持

一致；当更改为 Pascal 后，表名首字母大写，其余字母小写，当中间有下划线时，则认为是一个单词的开始，如果没有下划线，则不会以新单词开始，如下：

mem_lucky => MemLucky

memlucky => Memlucky

(3) 变量窗口

变量窗口中的变量名称，在模板里都有可能会用得到，如果你觉得还不够用，你可以自己定义更多的变量名称(见 3.1)。

变量名称	含义
Company	标示生成的文件的版权归属
Author	标示由谁生成的代码
ProjectName	标示所生成的项目的名称
Namespace	标示所生成的代码中的根命名空间
NameMode	命名方式，Inherit 和 Pascal 命名，在上级中已说明
TableRegex	表名的前缀正则表达式，表示需要剔除的前缀，若表达式为 TB_，那么类似 TB_USER 这样命名的表，其 ClassName 将自动剔除 TB_，即为 USER
ColumnRegex	与 TableRegex 类似，应用于字段名称上，PropertyName 受此影响

在生成代码之前，你需要调整这些变量值，使生成的代码更符合你的需求。

设置这些变量值后，你可以单击变量窗口的工具栏上的“保存到文件”按钮，将这些变更保存到文件中，下次通过单击“打开外部文件”加载这些设置。

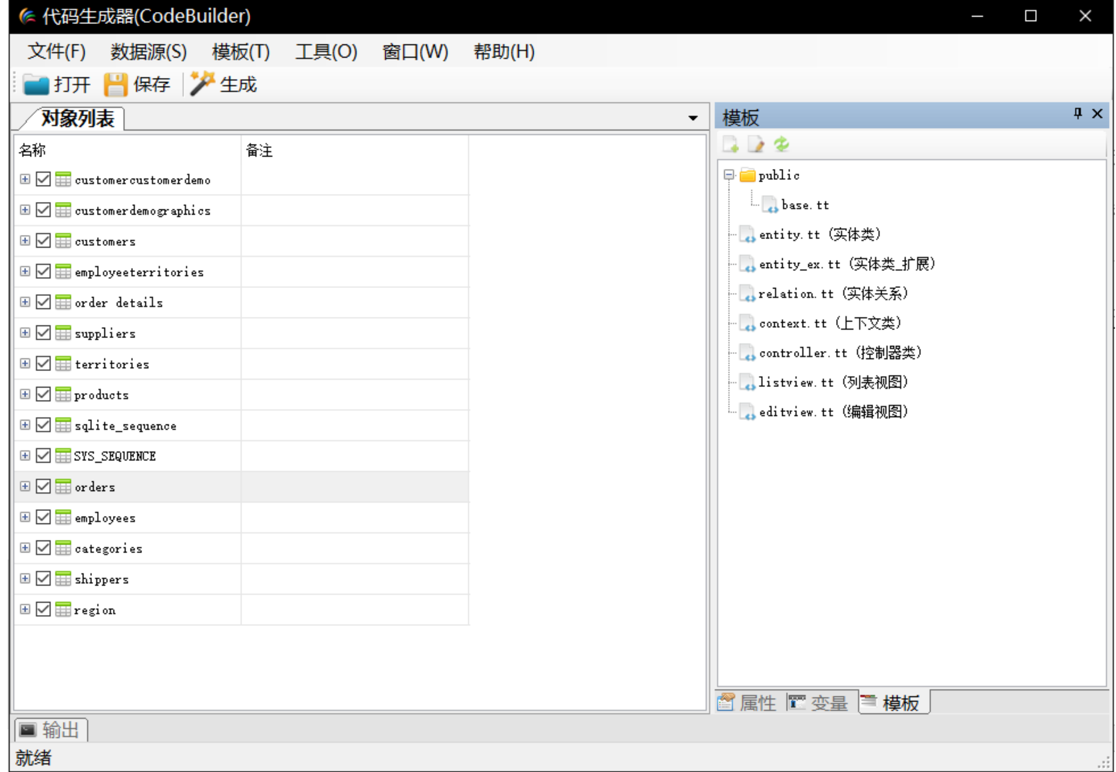
2.4. 使用模板

一切准备就绪后，你可以选择一个模板，生成最终的代码。

(1) 模板文件

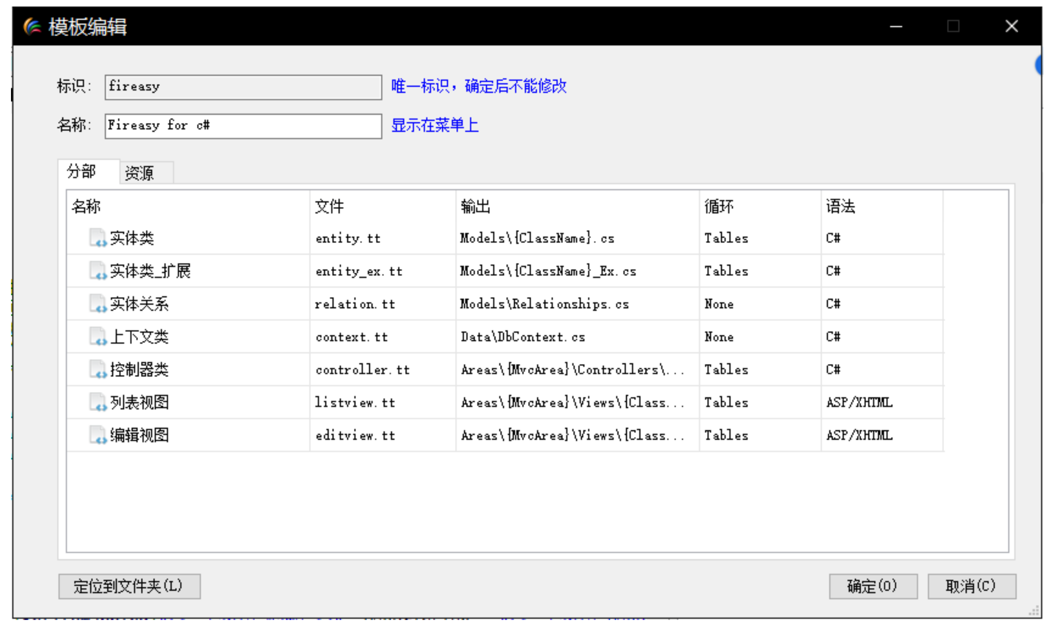
选择菜单“模板”，从下拉菜单里选择一种模板。目前提供的 Razor 和 T4 两种模板引擎。Razor 仅提供了一个生成 Entity Framework 的示例，如果你对此模板比较熟悉，你可尝试在此之下创建自己的 Razor 模板。T4 比较灵活且功能强大，作者建议你使用 T4 模板来生成代码，目前已经提供了好几套 T4 模板，你可以参考项目的模板来创建适合自己的模板。

下面以生成 Fireasy 的 ORMapper 为例。从菜单中选择 T4 Template 中的 Fireasy for C#模板然后，将主界面右侧切换至模板窗口，如下图：



一套模板由多个**分部**构成，每个分部负责生成一类文件。如上图中，entity.tt 负责生成实体类文件，entity_ex.tt 负责生成实体类的扩展文件，context.tt 负责生成 DbContext 上下文类文件，controller.tt 负责生成 MVC 中的控制器类文件，listview.tt 和 editview.tt 分别负责生成列表视图和编辑视图，等等。

单击工具栏“修改模板”按钮，弹出模板定义的窗口，如下图：



列表中的每一项目即是一个分部，它的输出和循环遵循一定的规则。

如果要为每一个表生成一个文件，则**循环**类型选择 Tables，那么**输出**的路径里，就可以

使用表中的相关属性来进行占位，如 ClassName、Name，除此之外，还可以使用变量窗口中设置的变量来进行占位。

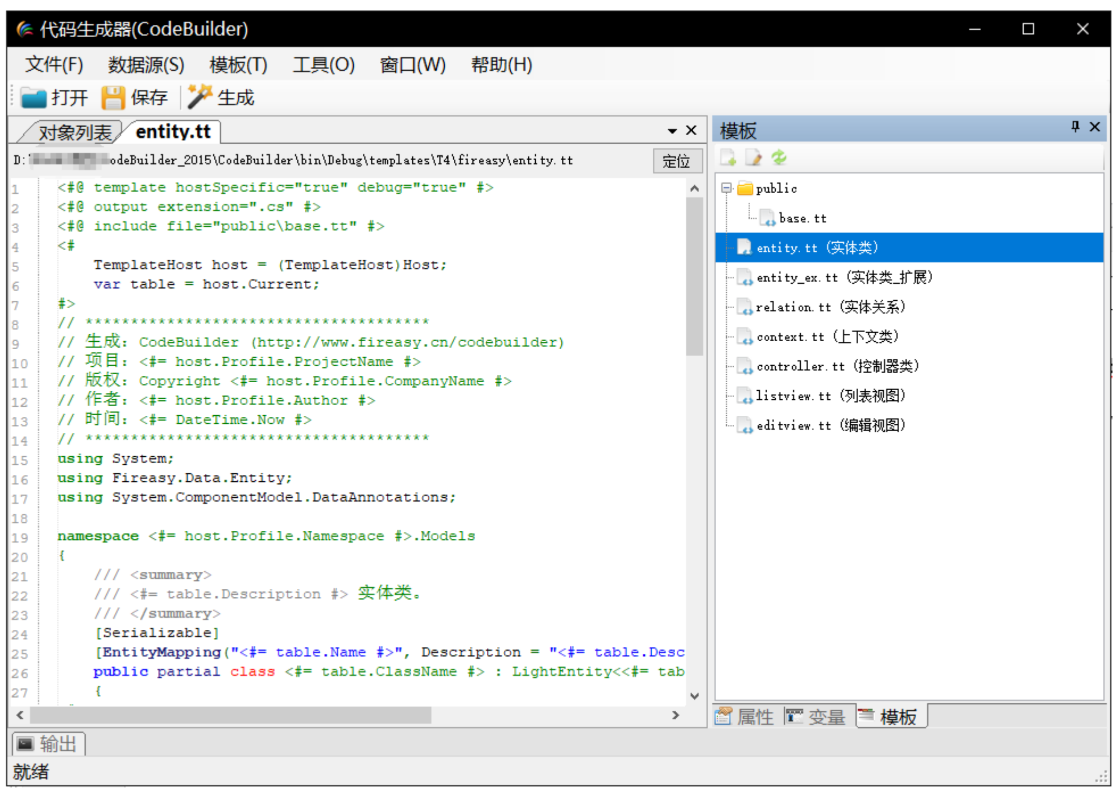
如果只生成一个文件，则循环选择 None，那么输出的路径里就不能再使用数据源中的相关属性，但仍可以使用变量窗口中的变量。

双击打开一个模板文件，映入你眼前的是你所熟悉的 T4 模板语法，因此 T4 的语法这里就不再赘述了，重点介绍一下里面的 Host 变量。

Host 是 T4 模板的核心，其承载了整个数据源中的表、字段、关系，以及变量窗口里所设置的值等等。

如果循环类型为 Tables，host.Current 返回的是当前的表 Table；如果后来类型为 None，则 host 中没有 Current 对象，取而代之的是 Tables、References 两个集合。

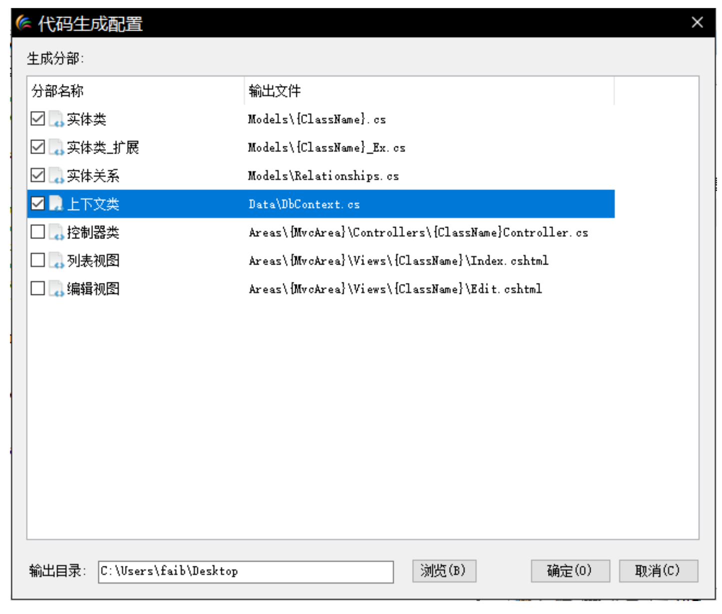
host.Profile 即变量窗口里所设置的值，如下图所示模板中的 host.Profile.ProjectName、host.Profile.Namespace 等等。



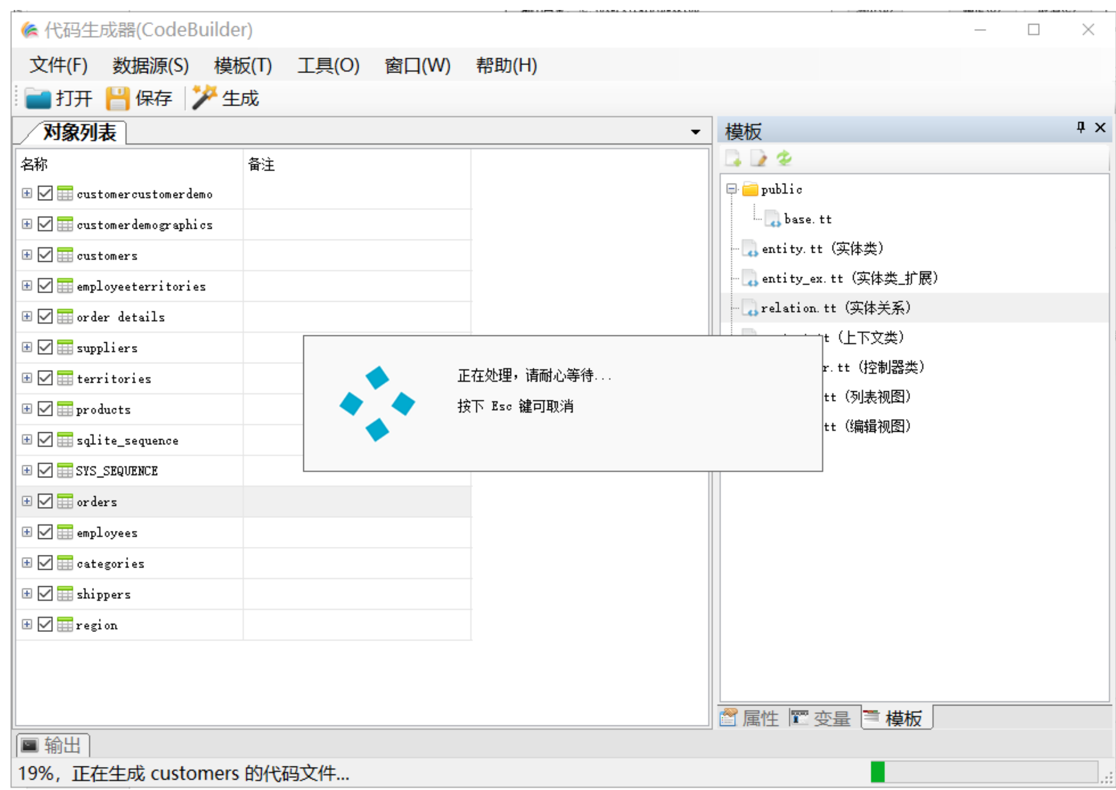
(2) 生成代码

模板选择就绪后，单击主界面工具栏上的“生成”按钮，或选择“菜单”—“生成代码

文件”，在弹出的分部选择对话框中，勾选需要生成的文件，首次生成需要选择文件输出的目录。



单击确定“按钮”后，CodeBuilder 将开始工作，为你生成你所需要的代码文件。



生成完毕，你就可以将生成的代码拷贝至你的项目里面了。

3. 开发说明

3.1. 变量扩展

变量，即变量窗口中的设置项，Razor 模板里的 Model.Profile 和 T4 模板里的 host.Profile 对象。变量是一个动态的、由外部代码文件定义的、经过动态编译而提供给模板使用的对象，其定义文件位于 CodeBuilder 工作目录 extensions\profile 文件夹下。以下是默认的定义，源自 C#语法。

```
using System.ComponentModel;

public class ProfileExt1
{
    [Description("命名空间。")]
    public string Namespace { get; set; }

    [Description("项目名称。")]
    public string ProjectName { get; set; }

    [Description("公司名称。")]
    public string CompanyName { get; set; }

    [Description("作者。")]
    public string Author { get; set; }

    [Description("表名转类名的替换正则。")]
    public string TableRegex { get; set; }

    [Description("字段名转类名的替换正则。")]
    public string ColumeRegex { get; set; }

    [Description("Mvc中区域的名称。")]
    public string MvcArea { get; set; }

    [Description("命名方式。Pascal 首字母大写，适用于用-分隔的命名；Inherit 延续原始名称。")]
    public NameMode NameMode { get; set; }
}

public enum NameMode
{
    Inherit, //延续
    Pascal //首字母大写，适用于用-分隔的命名
}
```

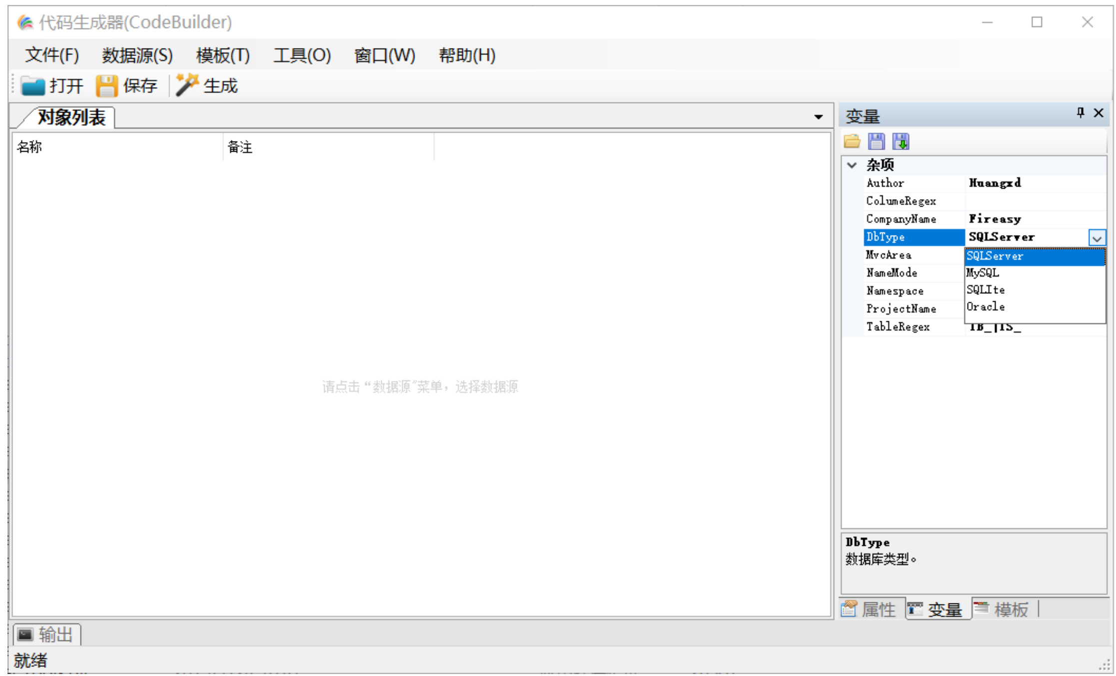
你可以在此基础上增减变量，也可以创建自己的代码文件，以满足你自己的需要。注意，此文件夹下的代码将会自动编译。下面我们通过一个小小的示例来说明该扩展的作用。

新增一个 C#语法的代码文件，编写如下代码，它的作用是在变量窗口里增加一个 DbType 变量。

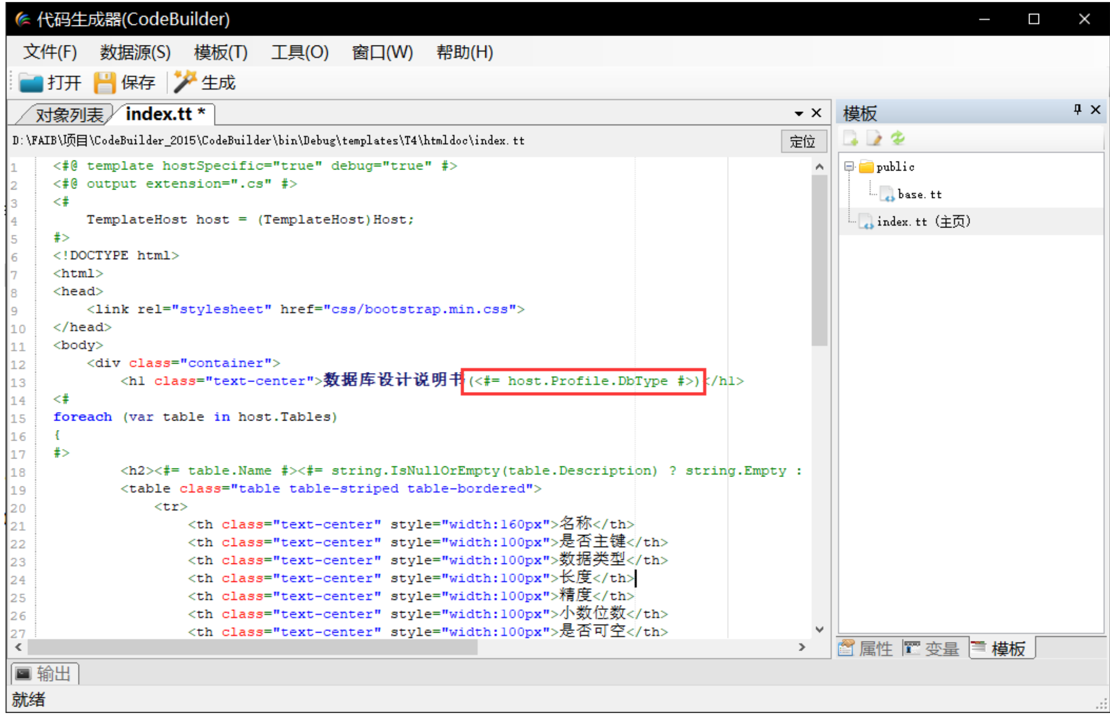
```
using System.ComponentModel;

public class ProfileExt2
{
```

```
        [Description("数据库类型。")]
        public DBType DbType { get; set; }
    }

    public enum DBType
    {
        SQLServer,
        MySQL,
        SQLIte,
        Oracle
    }
```

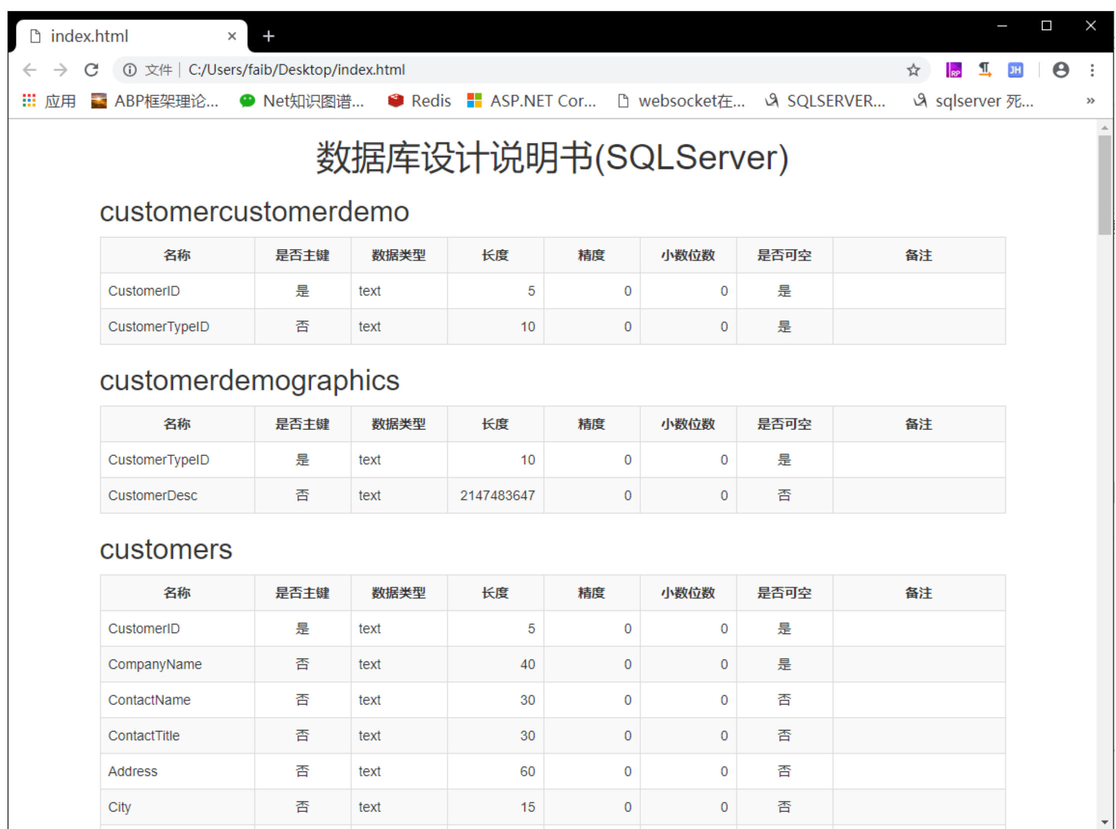
保存完毕后，运行 CodeBuilder，效果如下图所示：



通过扩展后，你就可以在模板里通过 Profile 对象来使用你定义的变量了，如下图：



现在，我们来试试生成最终代码：



数据库设计说明书(SQLServer)

customercustomerdemo

名称	是否主键	数据类型	长度	精度	小数位数	是否可空	备注
CustomerID	是	text	5	0	0	是	
CustomerTypeID	否	text	10	0	0	是	

customerdemographics

名称	是否主键	数据类型	长度	精度	小数位数	是否可空	备注
CustomerTypeID	是	text	10	0	0	是	
CustomerDesc	否	text	2147483647	0	0	否	

customers

名称	是否主键	数据类型	长度	精度	小数位数	是否可空	备注
CustomerID	是	text	5	0	0	是	
CompanyName	否	text	40	0	0	是	
ContactName	否	text	30	0	0	否	
ContactTitle	否	text	30	0	0	否	
Address	否	text	60	0	0	否	
City	否	text	15	0	0	否	

3.2. 数据源扩展

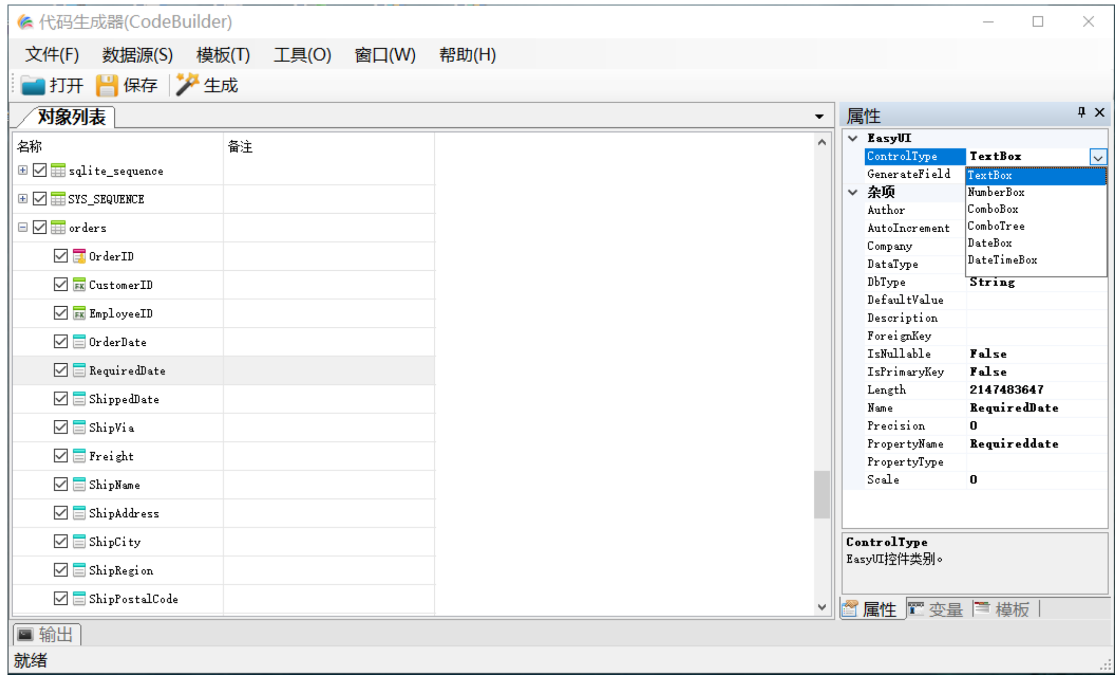
数据源也可在其定义之外进行扩展。数据源固有的属性见 2.2 节。与变量扩展类似，增加的属性也可以在模板里使用。其定义文件位于 CodeBuilder 工作目录 extensions\schema 文件夹下。下面以一个简单的示例来进行说明。

```
using System;
using System.ComponentModel;
using CodeBuilder.Core.Source;

[SchemaExtension(typeof(Column))]
public class ColumnExtForEasyUI
{
    [Category("EasyUI")]
    [Description("EasyUI控件类别。")]
    public EasyUIFieldType ControlType { get; set; }

    [Category("EasyUI")]
    [Description("是否生成域。")]
    [DefaultValue(true)]
    public bool GenerateField { get; set; }
}

public enum EasyUIFieldType
{
    TextBox,
    NumberBox,
    ComboBox,
    ComboTree,
    DateBox,
    DateTimeBox,
}
```

类需要添加 SchemaExtensionAttribute 特性，它表明将此类的属性扩展到对应的数据源，该特性的唯一参数为表 Table、字段 Column 和关系 Reference。

你可以使用 CategoryAttribute 对扩展的属性进行分组，这样会更清晰明了，方便管理。

运行 CodeBuilder，打开数据源后，选择某一个字段，右侧属性窗口可以看到 EasyUI 组下增加了两个属性，分别是 ControlType 和 GenerateField：



在模板里，可以通过字段对象来访问所扩展的属性，如下图：

```
    }
</script>
<div data-options="region:'center',border:false">
    <table class="form-body">
<#
foreach (dynamic column in table.Columns)
{
    if (column.IsPrimaryKey || !column.GenerateField
    {
        continue;
    }
#>
        <tr>
            <td class="addon"><#= column.Description #></td>
            <td>@Html.<#= column.ControlType.ToString() #>(s => s.<#= column.PropertyName
        </tr>
<#
}
#>
    </table>
</div>
<div data-options="region:'south',border:false" class="form-button">
    <a class="easyui-linkbutton" onclick="saveInfo()" id="btnSave" data-options="iconCls:'
    <a class="easyui-linkbutton" onclick="saveInfo(true)" id="btnSaveAndNew" data-options=
```

3.3. 定义新模板

略。

3.4. 开发数据源插件

略。

3.5. 开发模板插件

略。

3.6. 开发工具插件

略。