Asp.Net MVC网站项目搭建规范

一、项目命名规范

1.数据访问层命名要求：项目名称.Repository

例如：Demo.Repository

1.1 实体集合变量名命名要求：实体类名称+Repository

例如：UserInfoRepository

2.业务层命名要求：项目名称.Service

例如：Demo.Service

2.1 接口定义命名要求：I+实体类名称+ Service

例如：IUserInfoService

2.2 接口方法实现命名要求：实体类名称+Service

例如：UserInfoService

2.3 模型定义命名要求：模型名称+DM

例如：UserAllDM

3.网站命名要求：项目名称.Web

例如：Demo.Web

二、项目搭建步骤

1.新建项目选择 ASP.NET Web 应用程序，如图 1 所示。

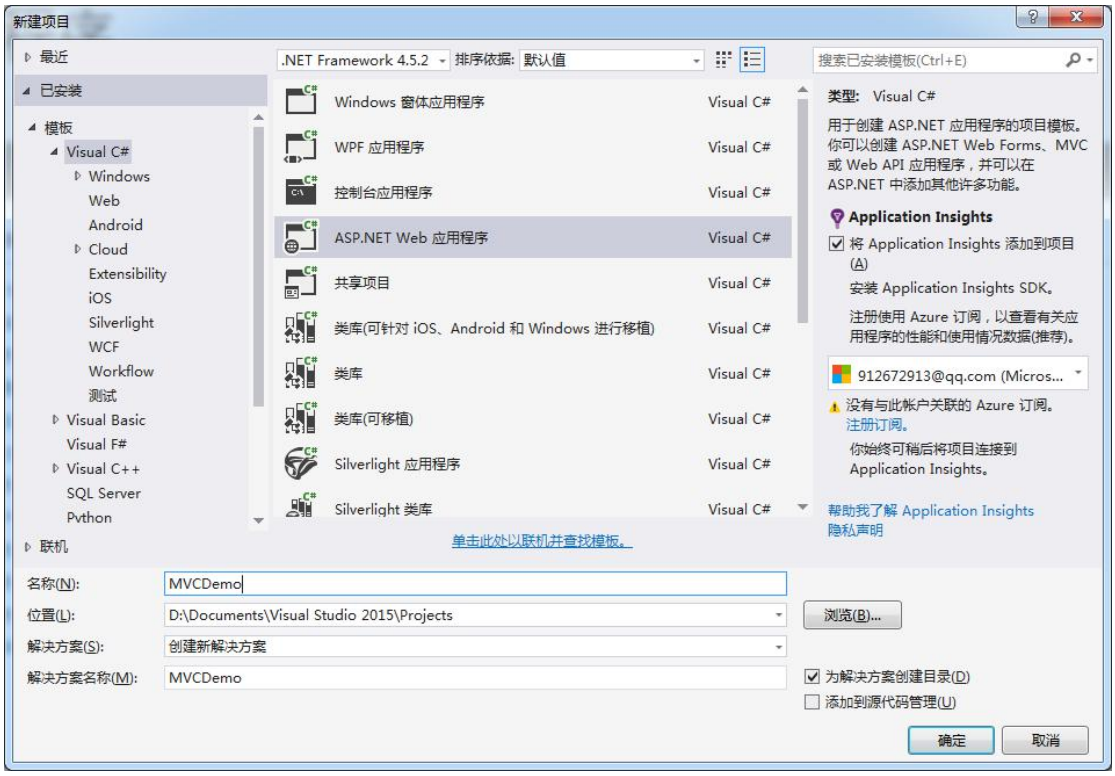


图 1

2.选择 MVC 模板，如图 2 所示。

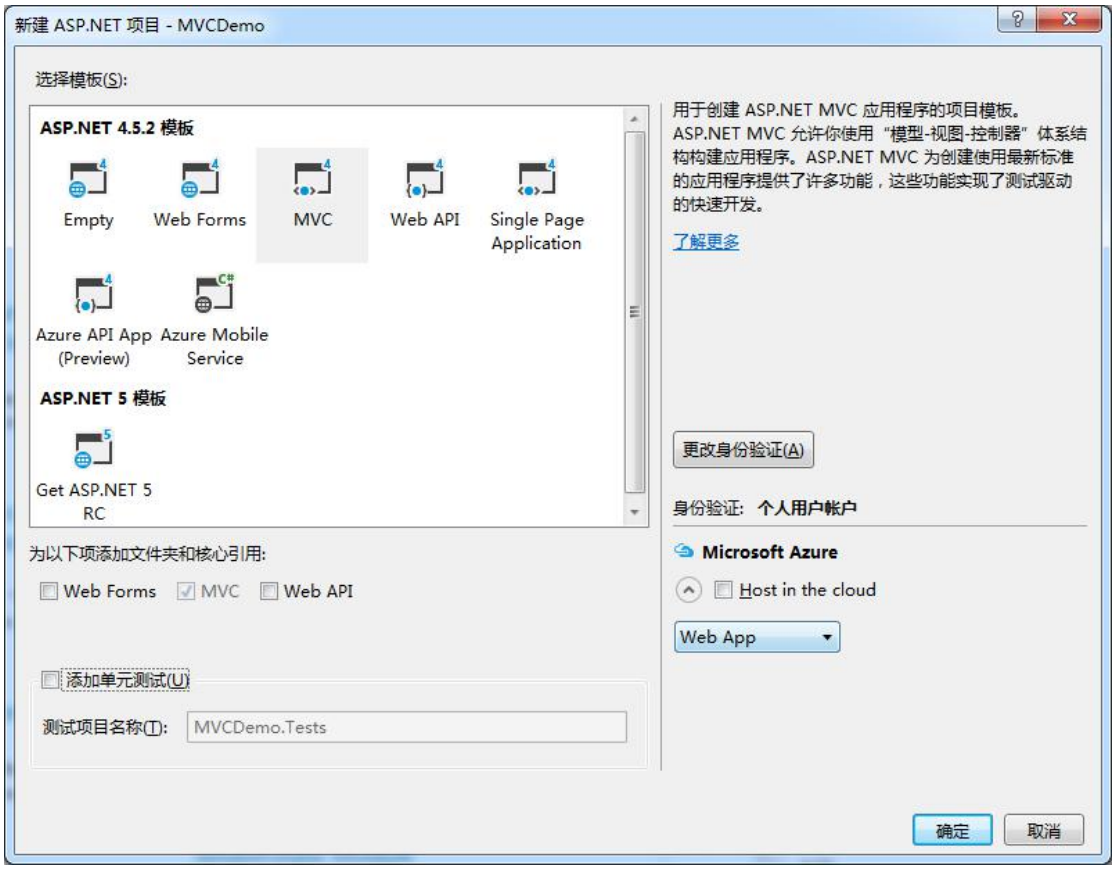


图 2

3.在项目中分别新建两个类库，分别是数据访问层（Repository）、业务层（Service），如图 3、图 4 所示。

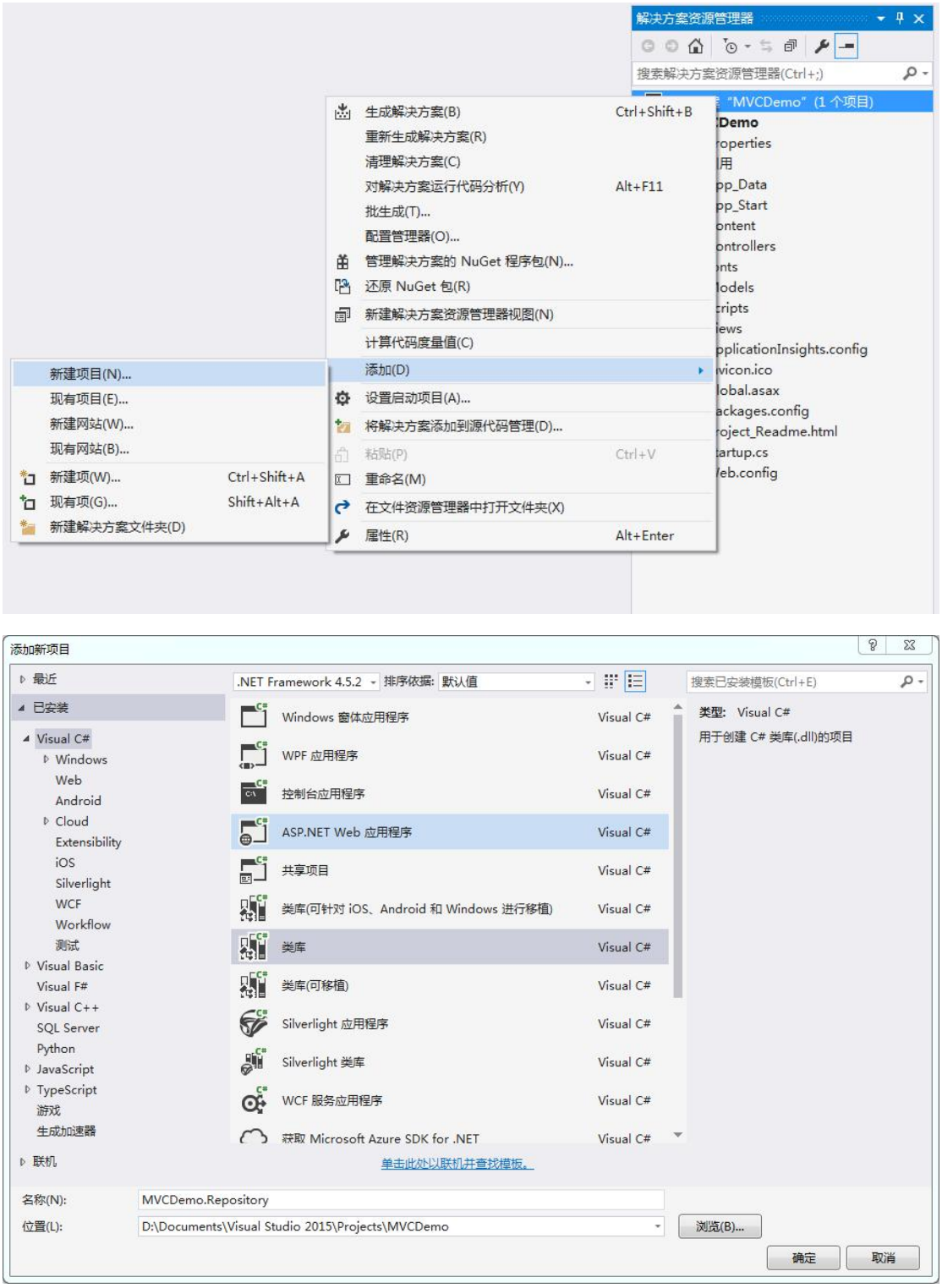

解决方案资源管理器
搜索解决方案资源管理器(Ctrl+;)
"MVCDemo"(1 个项目)
生成解决方案(B) Ctrl+Shift+B
重新生成解决方案(R)
清理解决方案(C)
对解决方案运行代码分析(Y) Alt+F11
批生成(T)...
配置管理器(O)...
管理解决方案的 NuGet 程序包(N)...
还原 NuGet 包(R)
新建解决方案资源管理器视图(N)
计算代码度量值(C)
添加(D)
设置启动项目(A)...
将解决方案添加到源代码管理(D)...
粘贴(P) Ctrl+V
重命名(M)
在文件资源管理器中打开文件夹(X)
属性(R) Alt+Enter
新建项目(N)...
现有项目(E)...
新建网站(W)...
现有网站(B)...
新建项(W)... Ctrl+Shift+A
现有项(G)... Shift+Alt+A
新建解决方案文件夹(D)
添加新项目
最近
已安装
Visual C#
Windows
Web
Android
Cloud
Extensibility
iOS
Silverlight
WCF
Workflow
测试
Visual Basic
Visual F#
Visual C++
SQL Server
Python
JavaScript
TypeScript
游戏
生成加速器
联机
.NET Framework 4.5.2
排序依据: 默认值
搜索已安装模板(Ctrl+E)
Windows 窗体应用程序 Visual C#
WPF 应用程序 Visual C#
控制台应用程序 Visual C#
ASP.NET Web 应用程序 Visual C#
共享项目 Visual C#
类库(可针对 iOS、Android 和 Windows 进行移植) Visual C#
类库 Visual C#
类库(可移植) Visual C#
Silverlight 应用程序 Visual C#
Silverlight 类库 Visual C#
WCF 服务应用程序 Visual C#
获取 Microsoft Azure SDK for .NET Visual C#
单击此处以联机并查找模板。
类型: Visual C#
用于创建 C# 类库(.dll)的项目
名称(N): MVCDemo.Repository
位置(L): D:\Documents\Visual Studio 2015\Projects\MVCDemo
浏览(B)...
确定
取消

图 3

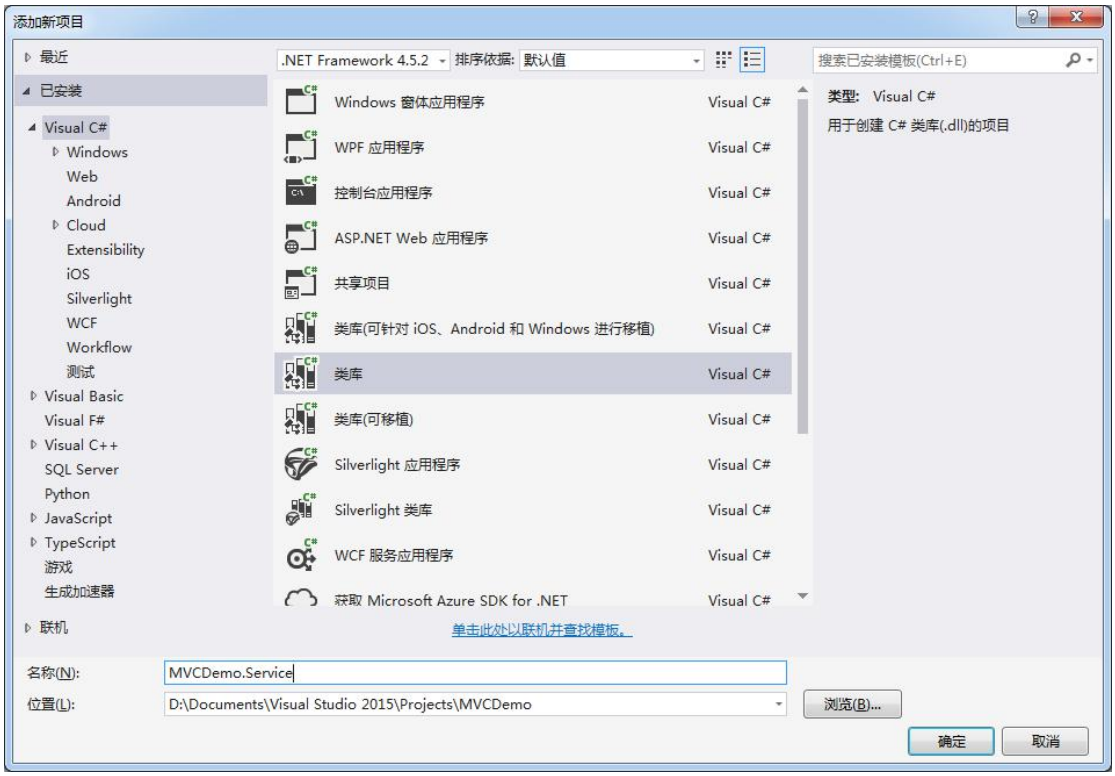
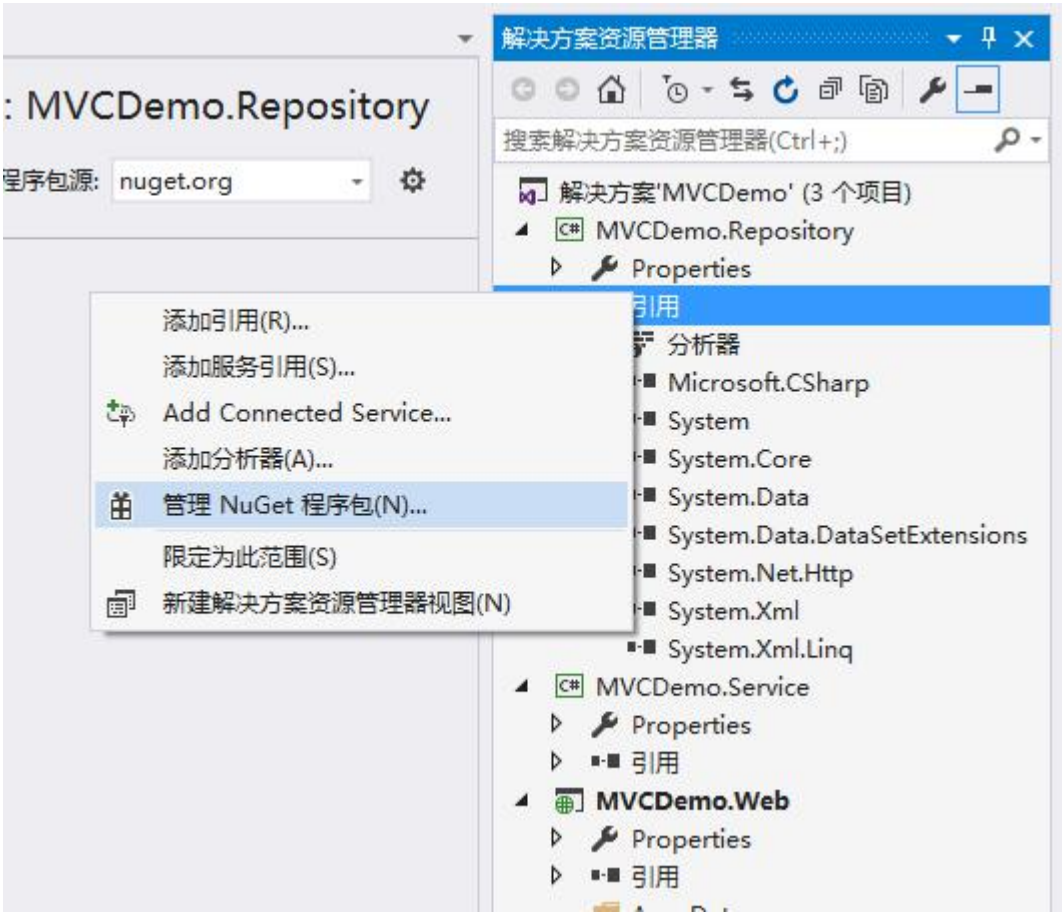

添加新项目
最近
已安装
Visual C#
Windows
Web
Android
Cloud
Extensibility
iOS
Silverlight
WCF
Workflow
测试
Visual Basic
Visual F#
Visual C++
SQL Server
Python
JavaScript
TypeScript
游戏
生成加速器
联机
.NET Framework 4.5.2
排序依据: 默认值
搜索已安装模板(Ctrl+E)
Windows 窗体应用程序 Visual C#
WPF 应用程序 Visual C#
控制台应用程序 Visual C#
ASP.NET Web 应用程序 Visual C#
共享项目 Visual C#
类库(可针对 iOS、Android 和 Windows 进行移植) Visual C#
类库 Visual C#
类库(可移植) Visual C#
Silverlight 应用程序 Visual C#
Silverlight 类库 Visual C#
WCF 服务应用程序 Visual C#
获取 Microsoft Azure SDK for .NET Visual C#
类型: Visual C#
用于创建 C# 类库(.dll)的项目
单击此处以联机并查找模板。
名称(N): MVCDemo.Service
位置(L): D:\Documents\Visual Studio 2015\Projects\MVCDemo
浏览(B)...
确定
取消

图 4

4.在数据访问层（Repository）中选中“引用”右击选择“管理 NuGet 程序包”，搜索 EntityFramework 引用，双击添加，如图 5 所示。



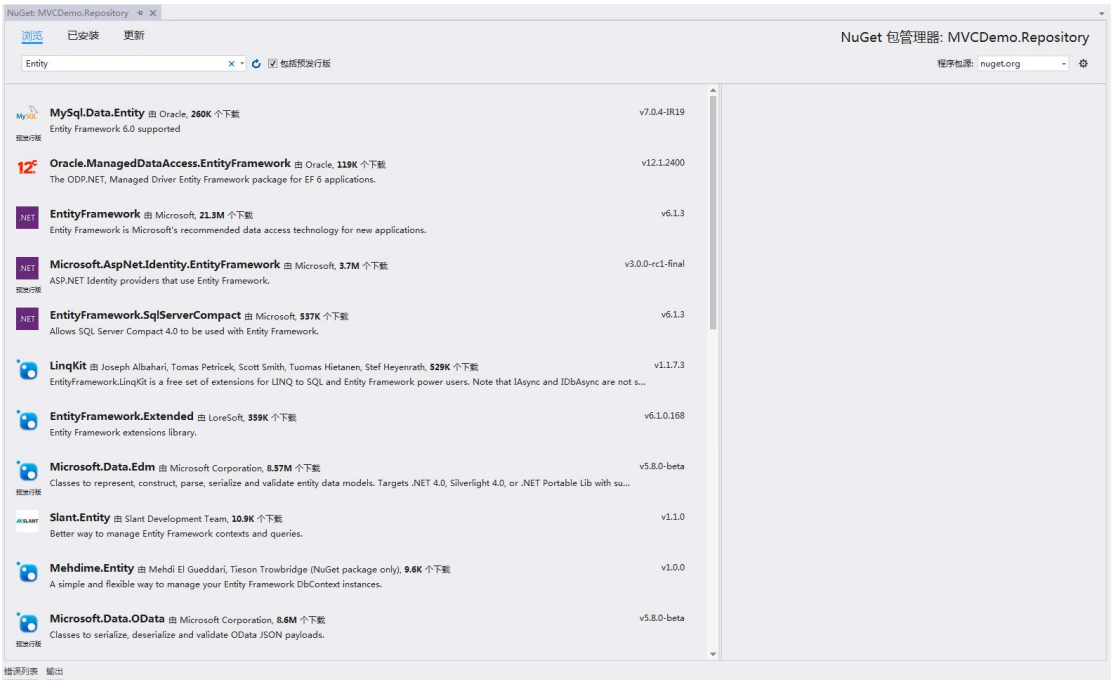


图 5

5.在数据访问层（Repository）中新建三个文件夹，分别是 Constaint、Entities、Enums，用来存放数据库表映射的实体集合、枚举。

在 Constaint 中新建 DataContent 类，用来链接数据库，绑定数据表集合，如图 6 所示。图中”DemoDataBase”为 MVC 项目 WebConfig 中的数据库连接名称。

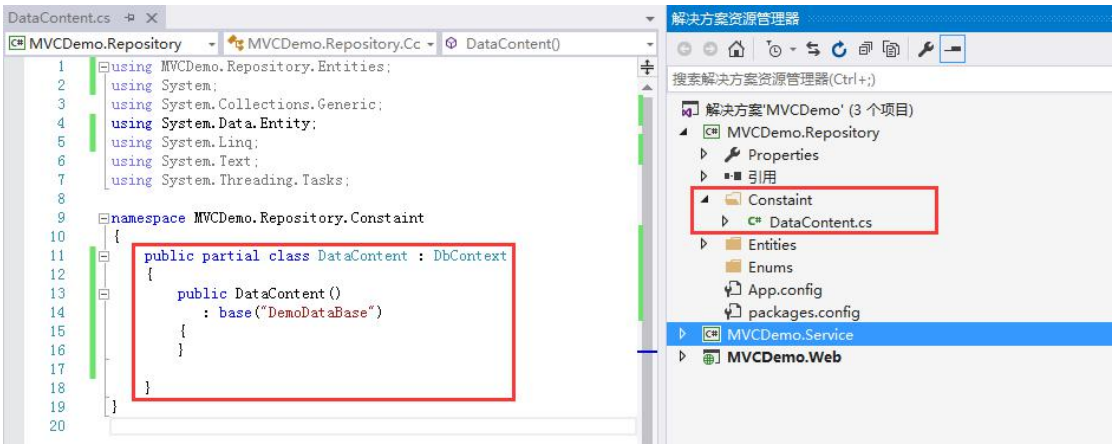


图 6

在 Entities 文件夹中新建类（可理解为数据库表），类中填写实体名称以及对应的类型（可理解为数据库字段）。现假设表为 UserInfo，字段为 UserID 和 UserName，建好后如下图 7 所示。

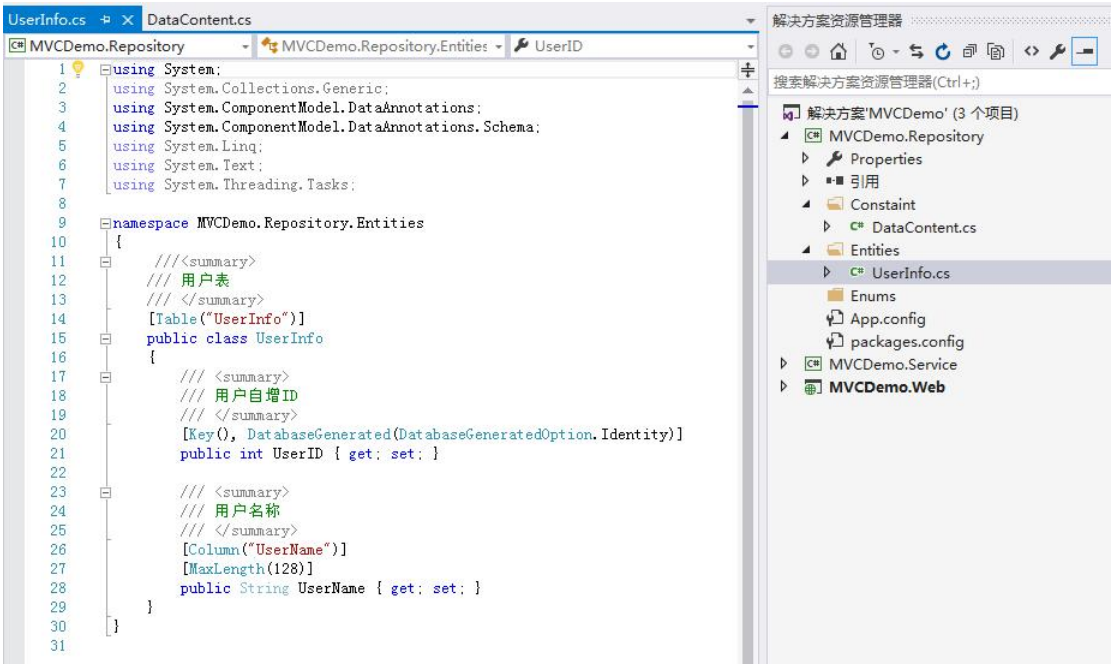


图 7

在 DataContent 类中需定义一个名称为“UserInfoRepository”的实体集合，如图 8 所示。

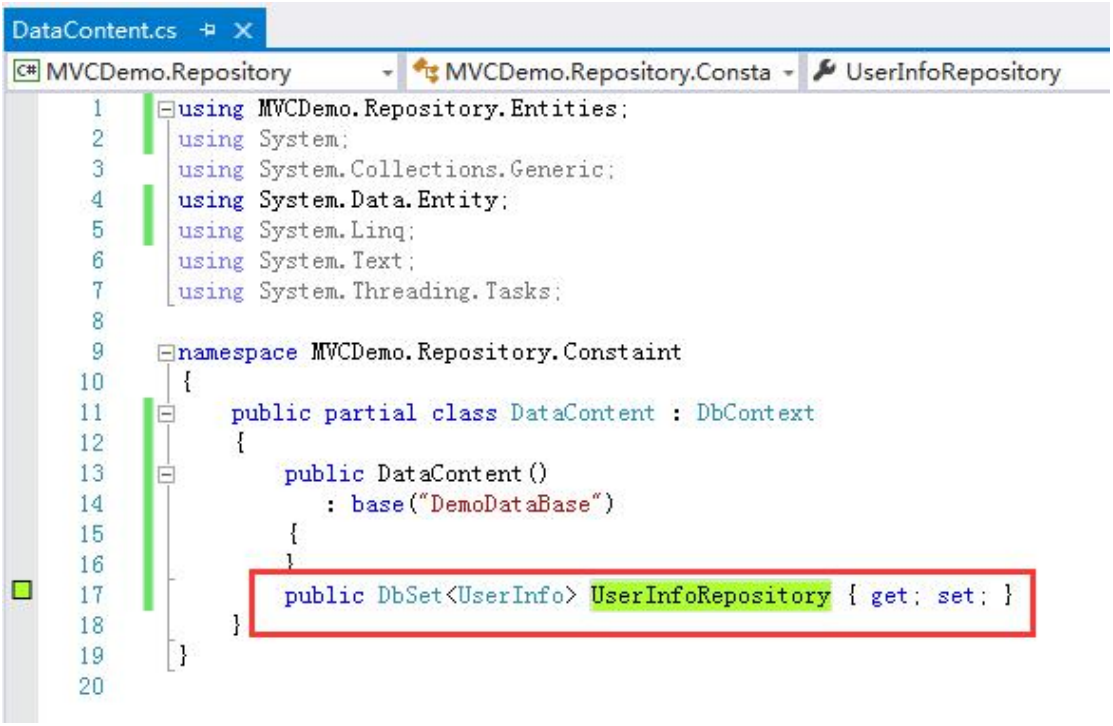


图 8

若当前项目有用到状态，则在 Enums 文件夹下创建对应的类。例如：现有管理员和会员两种用户，需定义用户角色，则在 Enums 文件夹下创建名为“UserRole”的类，在类中定义用户角色枚举，如图 9 所示。

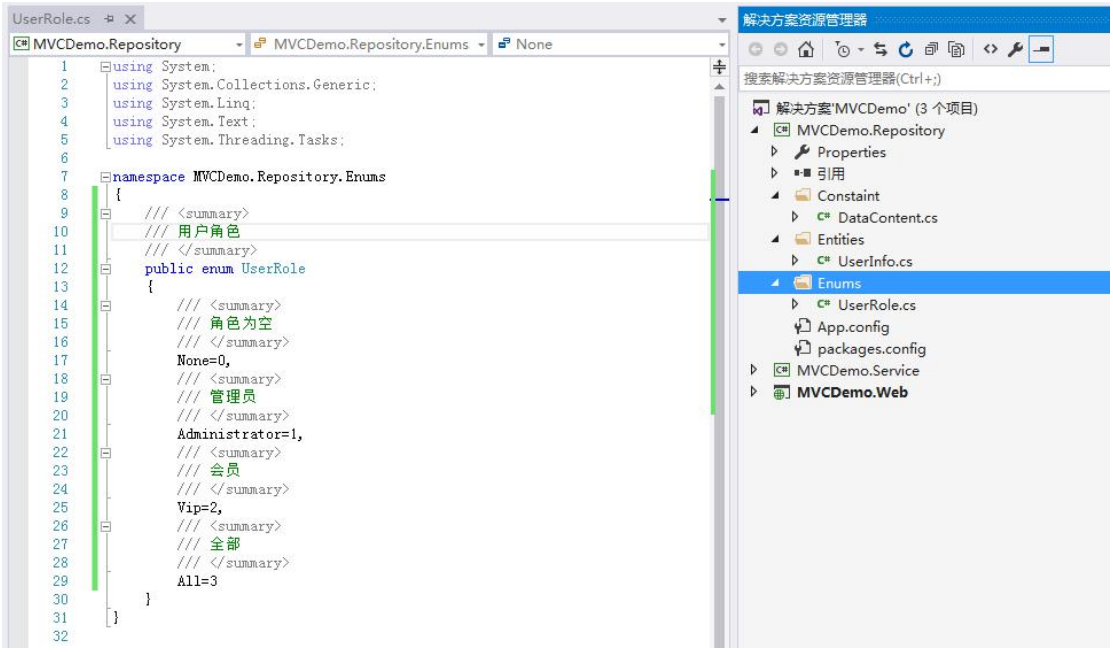


图 9

枚举定义好后将状态添加在 Entities 文件夹下的对应类中，如图 10 所示。

UserInfo.cs | UserRole.cs

MVCDemo.Repository | MVCDemo.Repository.Entities | UserRole

```
1   using MVCDemo.Repository.Enums;
2   using System;
3   using System.Collections.Generic;
4   using System.ComponentModel.DataAnnotations;
5   using System.ComponentModel.DataAnnotations.Schema;
6   using System.Linq;
7   using System.Text;
8   using System.Threading.Tasks;
9
10  namespace MVCDemo.Repository.Entities
11  {
12      ///<summary>
13      /// 用户表
14      /// </summary>
15      [Table("UserInfo")]
16      public class UserInfo
17      {
18          /// <summary>
19          /// 用户自增ID
20          /// </summary>
21          [Key(), DatabaseGenerated(DatabaseGeneratedOption.Identity)]
22          public int UserID { get; set; }
23
24          /// <summary>
25          /// 用户名称
26          /// </summary>
27          [Column("UserName")]
28          [MaxLength(128)]
29          public String UserName { get; set; }
30
31          /// <summary>
32          /// 用户角色
33          /// </summary>
34          [Column("UserRole")]
35          public UserRole UserRole { get; set; }
36      }
37  }
38
```

图 10

6.在业务层（Service）中引用数据访问层（Repository）和“EntityFramework”，添加三个文件夹分为别“Interface”（用于存放定义的接口）、“Method”（用于存放定义的方法）、“Model”（用来存放定义的业务模型）。

在业务层新建“BaseRepository”类，用来调用数据访问层的数据表，如图 11 所示。

BaseRespository.cs

MVCDemo.Service

```
using MVCDemo.Repository.Constaint;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MVCDemo.Service
{
    public class BaseRespository
    {
        protected DataContent Context = new DataContent();
    }
}
```

图 11

接口定义实例，如图 12 所示。

IUserService.cs

MVCDemo.Service

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MVCDemo.Service.Interface
{
    public interface IUserService
    {
        /// <summary>
        /// 用户登录
        /// </summary>
        /// <param name="UserAccount">登录账号</param>
        /// <param name="Password">登录密码</param>
        /// <returns>返回用户编号</returns>
        int Login(String UserAccount, String Password);
    }
}
```

图 12

接口方法实现实例，如图 13 所示。

UserService.cs

MVCDemo.Service　　MVCDemo.Service.Method.UserService

```
using MVCDemo.Repository.Entities;
using MVCDemo.Service.Interface;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MVCDemo.Service.Method
{
    public class UserService : BaseRespository, IUserService
    {
        public int Login(String UserAccount, String Password)
        {
            if (UserAccount.Length > 12 || UserAccount.Length < 4)
                throw new ArgumentException("用户名格式错误！");
            int result = (from a in Context.UserInfoRepository
                          where a.UserName == UserAccount
                          select a).ToList().Count;
            if (result == 0)
                throw new ArgumentException("用户名不存在！");

            UserInfo count = (from item in Context.UserInfoRepository
                              where item.UserName.Equals(UserAccount) && item.Password.Equals(Password)
                              select item).FirstOrDefault();
            if (count != null)
            {
                return count.UserID;
            }
            return 0;
        }
    }
}
```

图 13

7. 在 MVC 项目里引用业务层（Service）和 Ninject（依赖注入），如图 14、15 所示。

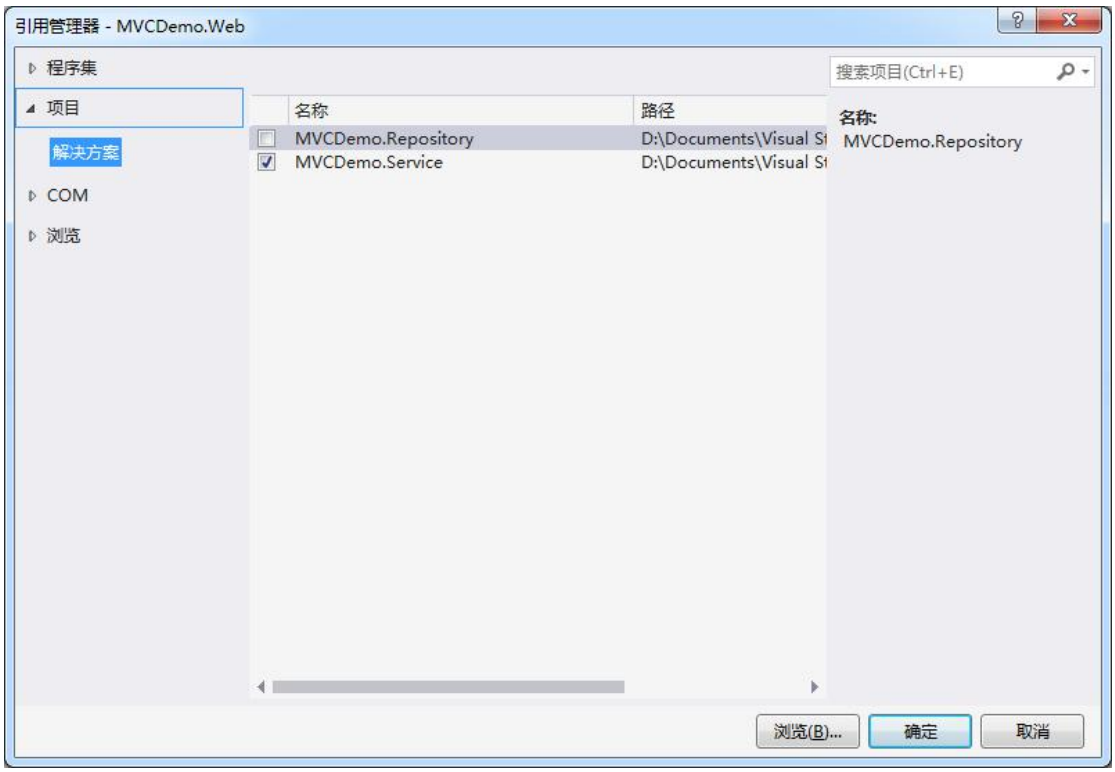


图 14

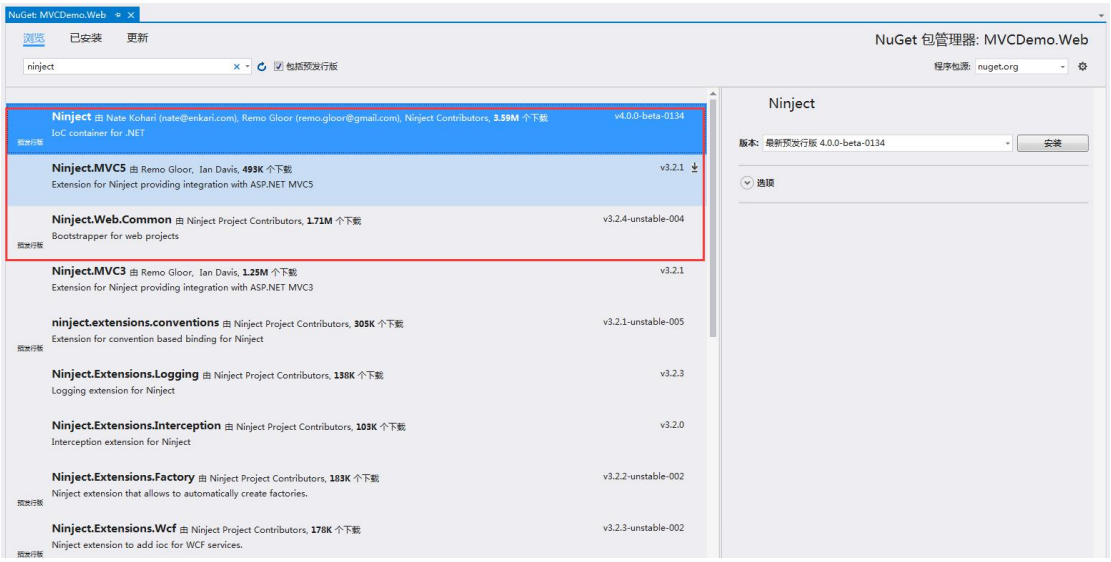


图 15

打开“App_Start”文件夹下的“NinjectWebCommon”类，绑定业务层接口，实例如图 16 所示。

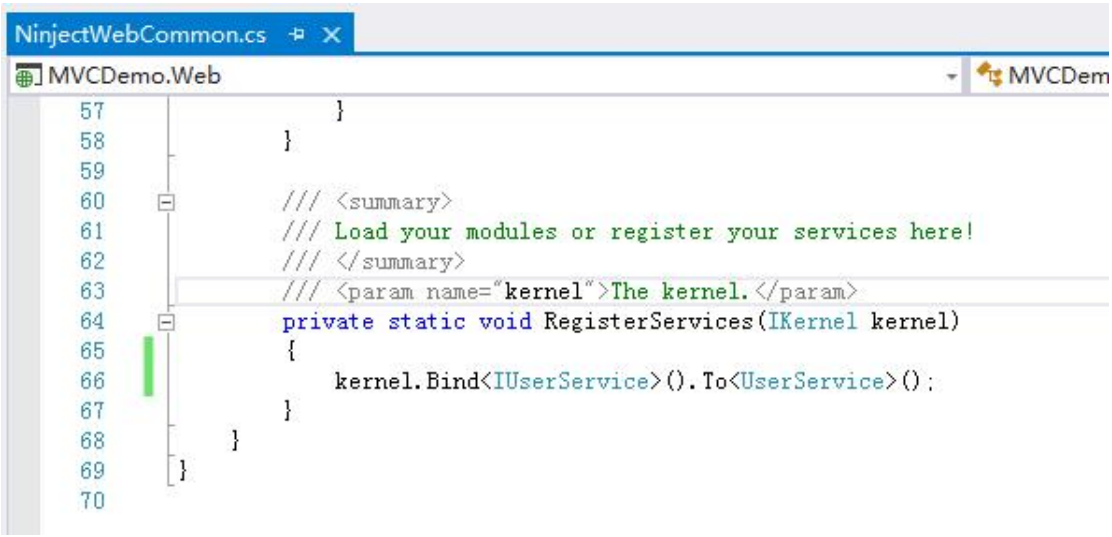


图 16

在 WebConfig 中添加如图 17 红框中标记的语句，完成与数据库的链接。

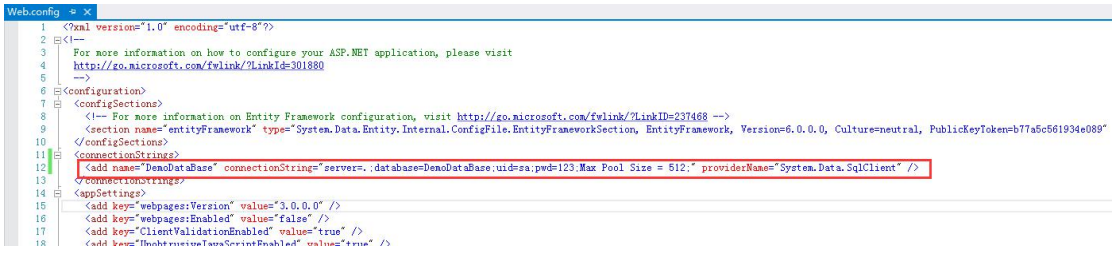


图 17