.Net 开发技术实验指导书

（第三版）

实验一　Asp.Net　MVC 入门

一、实验目的

- 了解.NET Framework 的结构。
- 理解.NET Framework 的基本概念
- 了解 .NET Framework 命名空间
- 掌握 MVC 程序的基本构成
- 掌握 MVC 程序最基本的编写方法

二、实验内容

- MVC 基本开发
- 简单学生表的管理

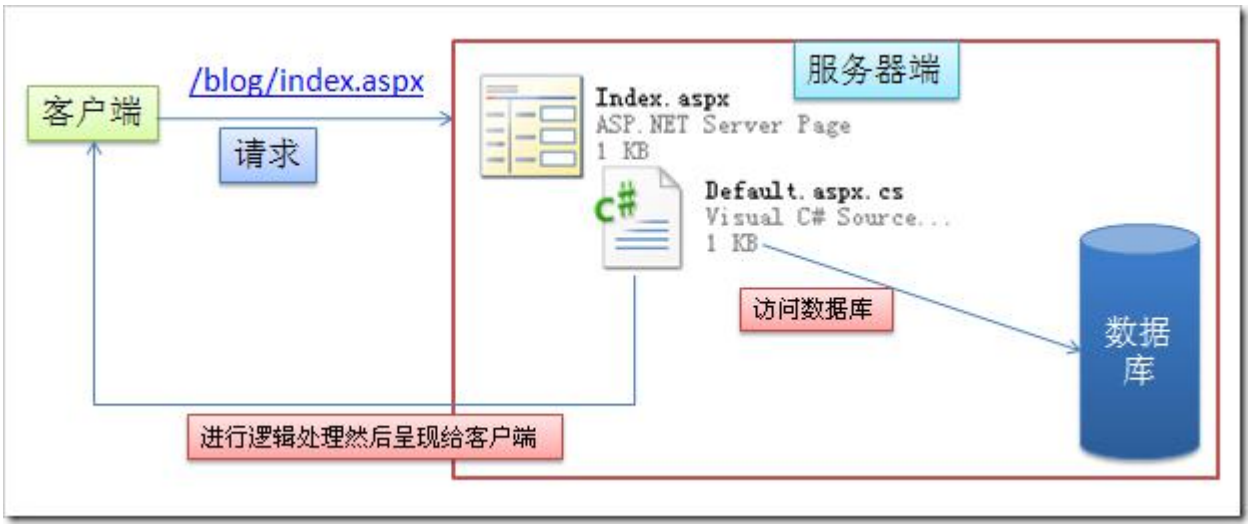
三、基础知识

什么是 MVC 模式

MVC（Model-View-Controller，模型—视图—控制器模式）用于表示一种软件架构模式。它把软件系统分为三个基本部分：模型（Model），视图（View）和控制器（Controller）。

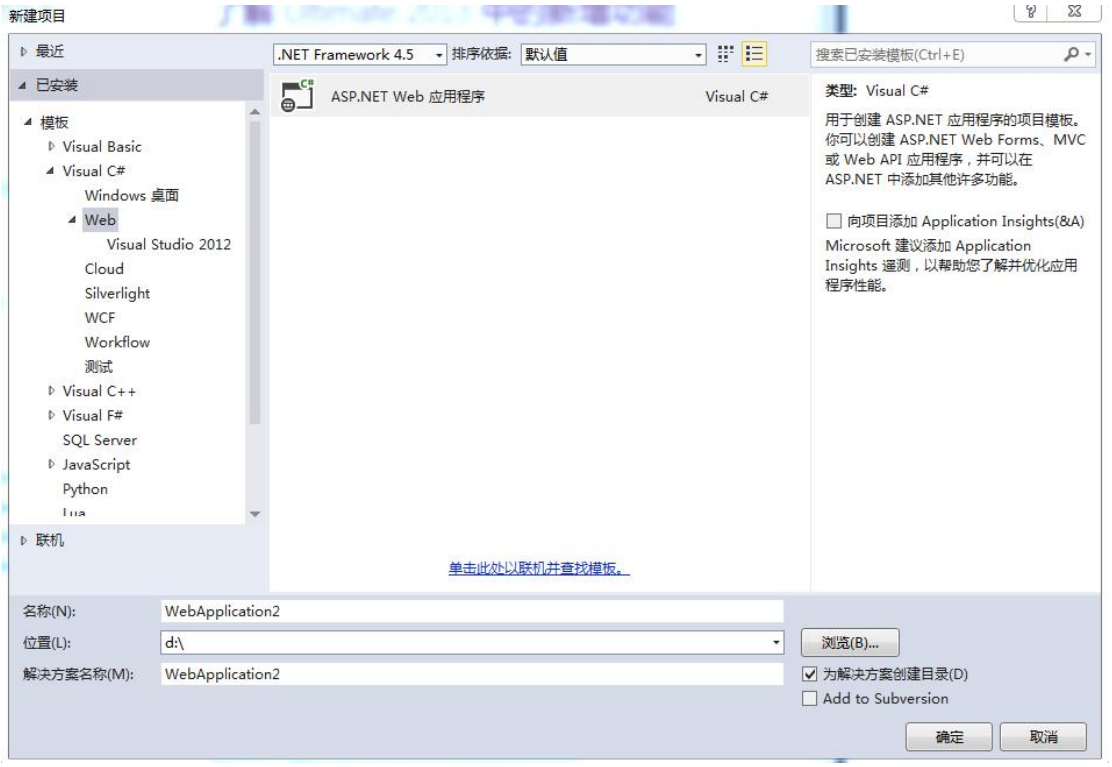
那么 MVC 模式和熟悉的 WebForm 模式有什么不同呢？他的各个部分又是怎样分工的呢？

先来看一下普通的 WebForm 模式下，请求一个例如 http://www.51mvc.com/blog/index.aspx 的 URL，那么的 WebForm 程序会到网站根目录下去寻找 blog 目录下的 index.aspx 文件，然后由 index.aspx 页面的 CodeBehind 文件(.CS 文件)进行逻辑处理，其中或许也包括到数据库去取出数据(其中的经过怎样的 BLL 到 DAL 这里就不谈了)，然后再由 index.aspx 页面来呈现给用户。简单的示意图如下所示：

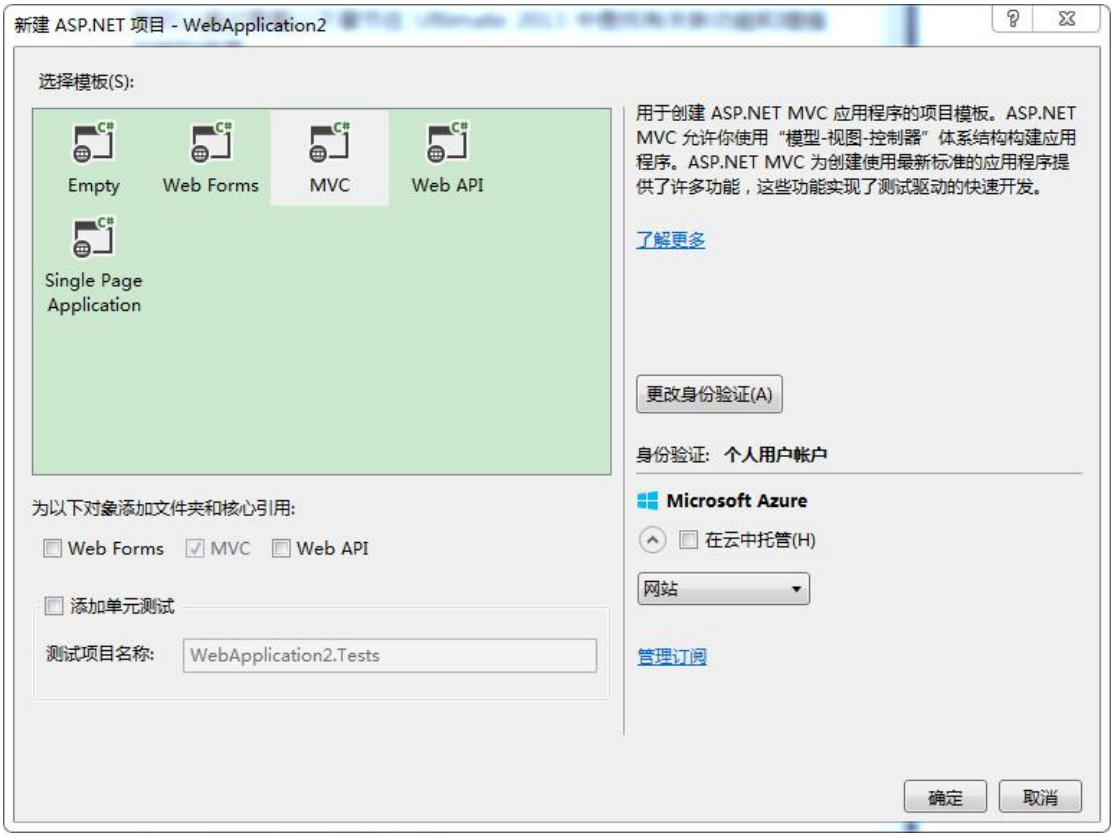


也就是一个 URL 请求的是在服务器与该 URL 对应路径上的物理文件(ASPX 文件或其他)，

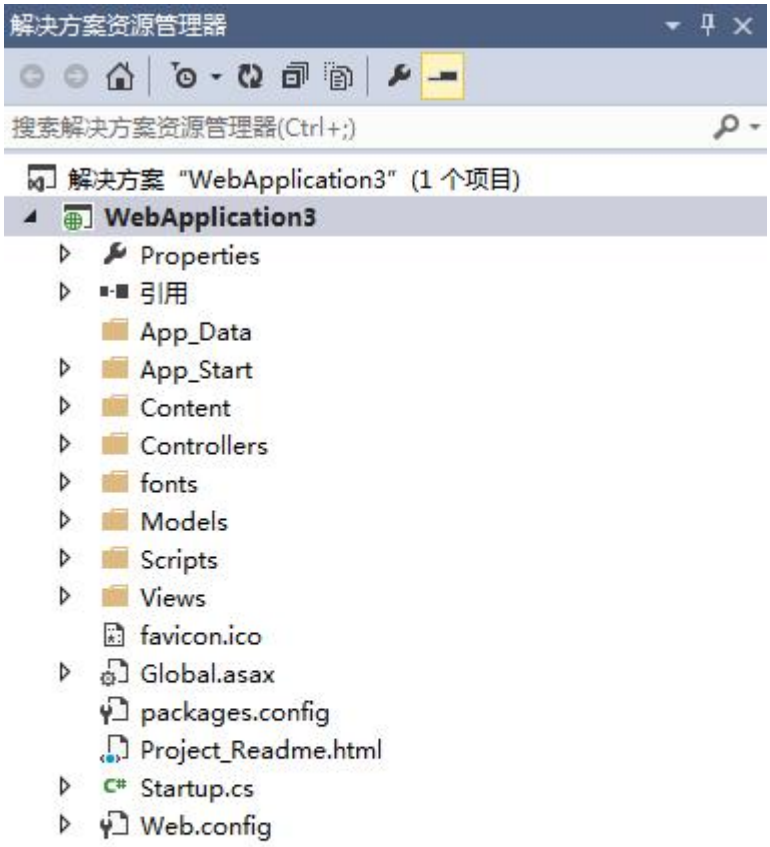
然后由该文件来处理这个请求并返回结果给客户端。
但是，对于 MVC 模式，这是怎样的一个过程呢？
先来建一个 ASP.NET MVC 的项目吧。



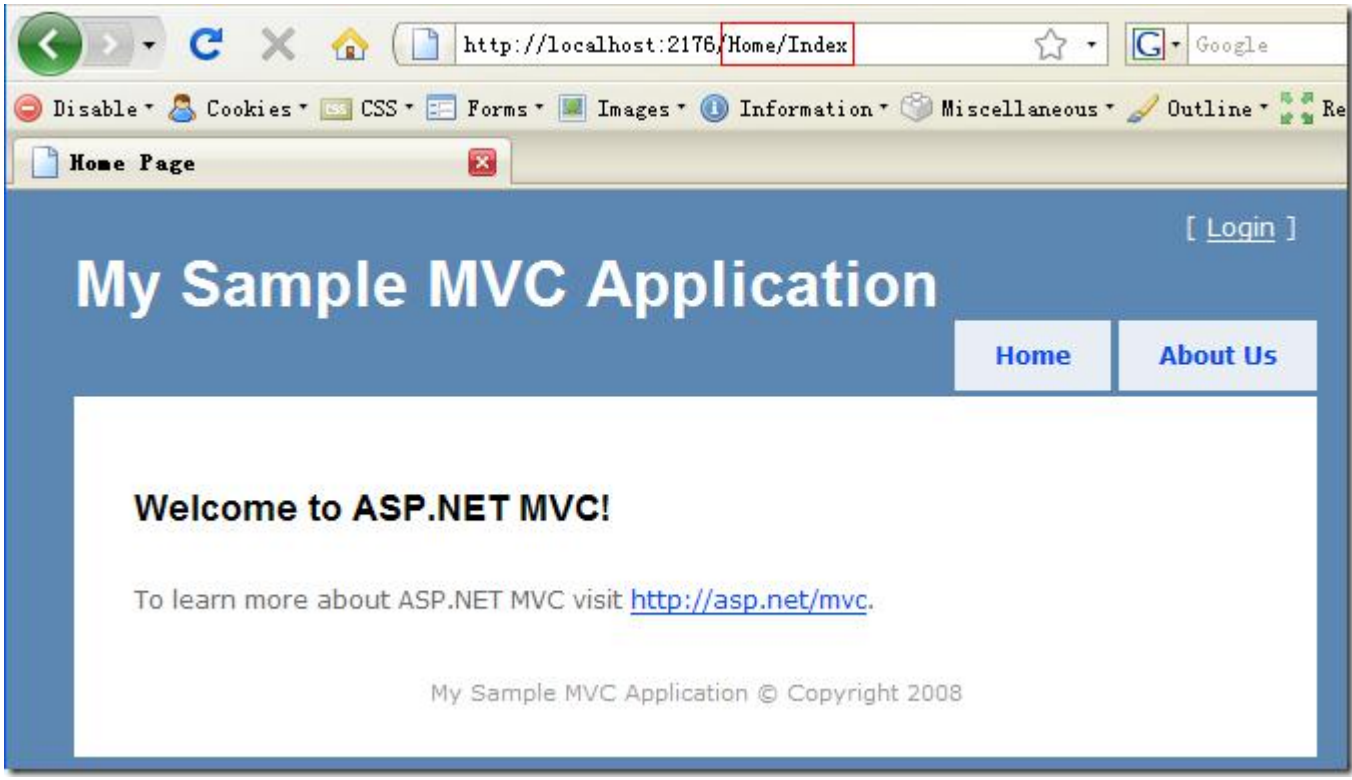
选择 MVC



注：如果的是中文版的 VS，安装完后可能会出现找不到这个模板的现象。
建立一个 ASP.NET MVC 项目后，默认的项目大概如下图：



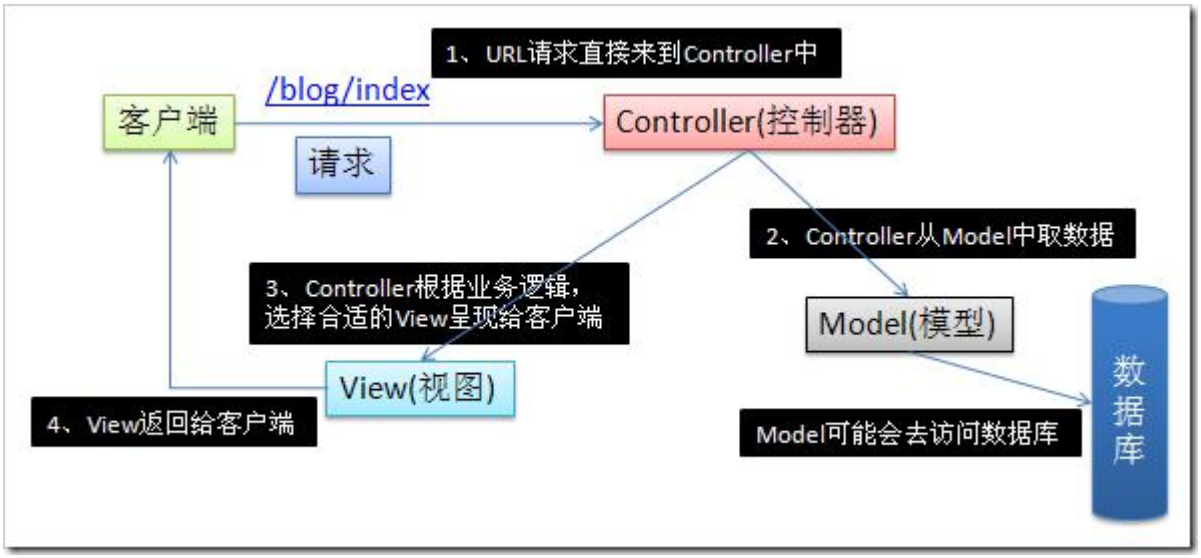
可以看到项目中有几个文件夹的命名和 MVC（Model-View-Controller，模型—视图—控制器模式）是对应的。然后运行一下项目看看：



注意到地址栏的 URL 是 Home/Index,如果按照前面说的 WebForm 的模式的话，应该可以在的项目的根目录下找到 Home 目录，然后 Home 目录下有个 Index 的文件，但是并不能在根目录下找到 Home 这个目录。不过还是让在 Views 目录下找到了 Views/Home/Index.aspx 文件，输入这个地址运行看看：

路径是对的，文件也存在，但为什么会是 404,说找不到文件呢？如果不是直接访问存在的物理文件，那么 MVC 又是怎样工作的呢？

原来啊，MVC 模式的工作过程是这样的：



在 MVC 中，客户端的所请求的 URL 是被映射到相应的 Controller 去，然后由 Controller 来处理业务逻辑，或许要从 Model 中取数据，然后再由 Controller 选择合适的 View 返回给客户端。再说回前面运行的 ASP.NET MVC 程序访问的 http://localhost:2176/Home/Index 这个 URL，它访问的其实是 HomeController 中的 Index 这个 Action，见下图：

其中 public ActionResult Index()这个方法称为 Controller 的 Action，他返回的是 ActionResult 的类型。一个 Controller 可以有很多个 Action。

那么一个 URL 是怎样被定位到 Controller 中来的呢？先来看一下 web.config 文件，在 web.config 文件的 httpModules 配置节中，可以看到一个 UrlRoutingModule：

<add name="UrlRoutingModule" type="System.Web.Routing.UrlRoutingModule, System.Web.Routing, Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35"/>

就是这个 UrlRoutingModule 来把 URL 定位到 Controller 中去的。而对于 URL 会被路由到哪一个 Controller 中去，这些是完全可以自己定义的。到 Global.asax 文件去看一下：

```
public static void RegisterRoutes(RouteCollection routes)
{
    //忽略对.axd文件的Route，也就是和WebForm一样直接去访问.axd文件
    routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

    routes.MapRoute(
        "Default",                                              // Route 的名称
        "{controller}/{action}/{id}",                           // 带有参数的URL
        new { controller = "Home", action = "Index", id = "" }  // 设置默认的参数
    );

}

protected void Application_Start()
{
    //在程序启动的时候注册我们前面定义的Route规则
    RegisterRoutes(RouteTable.Routes);
}
```

可以看到这里定义了一个名为"Default"的 Route，还定义了默认的参数。默认参数的意义在于，当访问例如 http://localhost:2176/的 URL 的时候，他会将不存在的参数用默认的参数补上，也就是相当于访问 http://localhost:2176/Home/Index 一样。

注意：知道在 IIS 中，访问网站的根目录的时候，如果不指定要访问的路径，IIS 会自己根据在 IIS 中设置的默认文档去访问。例如访问 http://localhost:2176/这个 URL 的时候，IIS 会去寻找网站根目录下的 Default.aspx 文件(假设设置了 IIS 的默认文档为 Default.aspx)。而在 ASP.NET MVC 中对于类似 http://localhost:2176/这样的网站根目录的路径，并不会经过 Route 的处理，所以看到建立的 ASP.NET MVC 程序的根目录下有个 Default.aspx 文件，该文件就是用于处理前面的访问根目录的情况的。请不要删除该文件。它会将 http://localhost:2176/Default.aspx 交由 ASP.NET MVC 来处理，具体请看 Default.aspx.cs 文件。

知道了一个 URL 是怎样定位到相应的 Controller 中去的了，那么 View 又是怎么被返回给客户端的呢？从前面的截图中看到，Controller 中的 Action 方法中有个 return View()的方法。默认情况下它会返回与 Action 同名的 view. 在 ASP.NET MVC 默认的视图引擎(WebFormViewEngine)下，view 是按如下路径访问的：

/Views/{Controller}/{Action}.aspx

也就是说对于 http://localhost:2176/Home/Index 这个路径，在默认情况下，在 Index 这个 Action 中用 return View()来返回 view 的时候，会去寻找/Views/Home/Index.aspx 文件，如果找不到这个文件，就会去 Share 目录中寻找：/Views/Share/Index.aspx,如果都找不到，就会抛出找不到 View 的异常。return View("lulu.aspx")来指定要返回哪一个 view：/Views/Home/lulu.aspx。

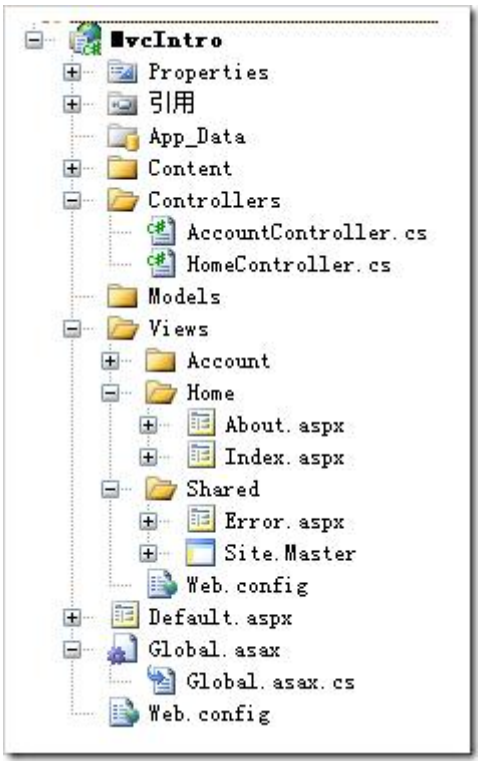
那么为什么前面直接访问 Views/Home/Index.aspx 这里文件的时候会出现 404 错误，说找不到文件呢？因为在 MVC 中，是不建议直接去访问 View 的，所以建立的 ASP.NET MVC 程序在默认情况下就在 Views 目录下加了一个 web.config 文件，内容如下：

```
<system.web>
  <httpHandlers>
    <add path="*" verb="*"
         type="System.Web.HttpNotFoundHandler"/>
  </httpHandlers>
  <pages validateRequest="false">
  </pages>
</system.web>
```

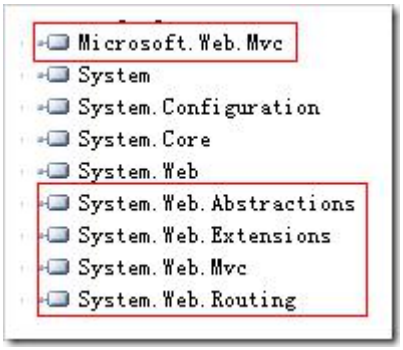
也就是访问 Views 目录下的所有的文件都会由 System.Web.HttpNotFoundHandler 来处理，所

以请不要将资源文件(CSS、JS、图片等)放到 Views 目录中。如果确实要放到 Views 目录下的话，请修改 Views/web.config 文件。

新建一个 ASP.NET MVC 的 Web Application 后，默认的情况下，项目的目录结构如下：



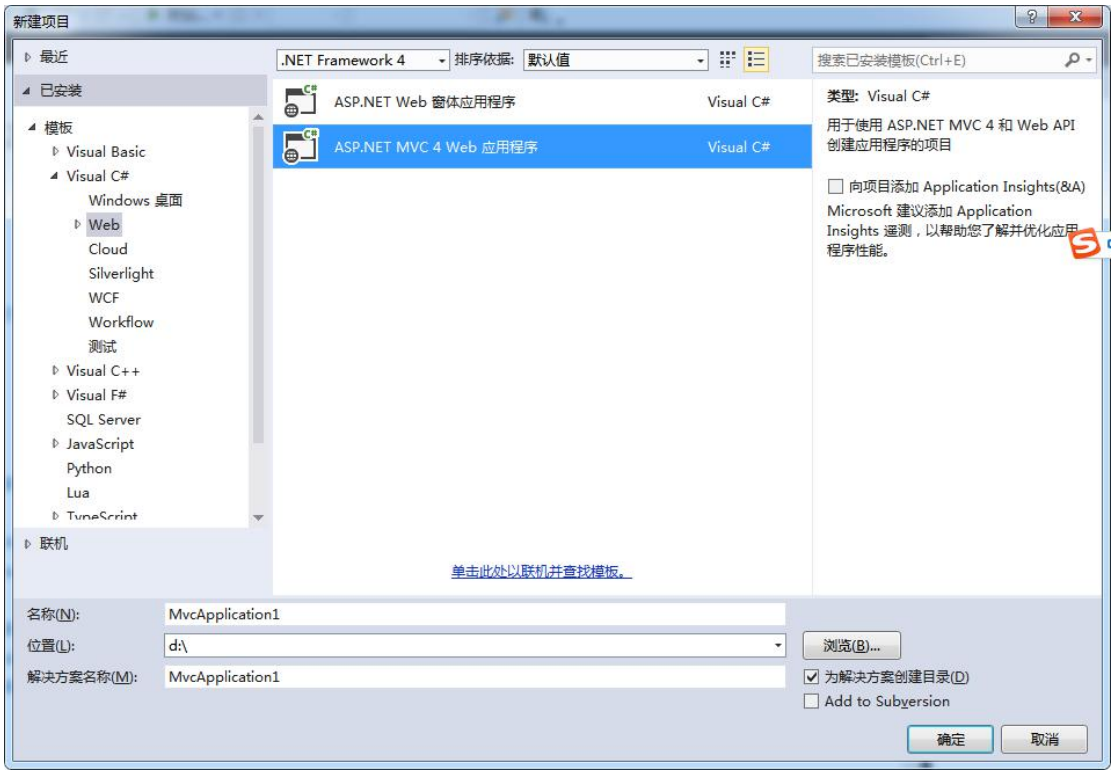
App_Data：这个目录跟一般的 ASP.NET website 是一样的，用于存放数据。
Content：这个目录是建议用来存放一下资源文件的。例如 CSS、JS、图片等等。当然不愿意的话，完全可以不放到这里来。
Controllers：这个目录是建议将 Controller 类都放到这里来，方便管理。Controller 类的命名必须以 Controller 结尾，例如一个名为 Home 的 Controller 则要命名为 HomeController。
Models：这个目录是建议用来存放的业务实体、数据访问层代码的类的。当然，更好的做法觉得应该是将 Models 独立为一个类库。
Views：在默认情况下，所有的 view 文件都必须放到这个目录下来，每一个 Controller 对应一个子目录，而且子目录的命名必须以 Controller 的命名一样。例如，HomeController 的 view 就应该放到 Home 子目录中。见到 Views 目录下还有一个 Shared 的子目录，这个子目录是用于存放一些共享的 view 的，例如 Error.aspx 和 Site.Master。Controller 在 Views\ControllerNmae 中找不到指定的 view 的时候，会到 Shared 中去寻找。
下面来看一下 ASP.NET MVC 比较核心的 DLL，见下图红框部分：

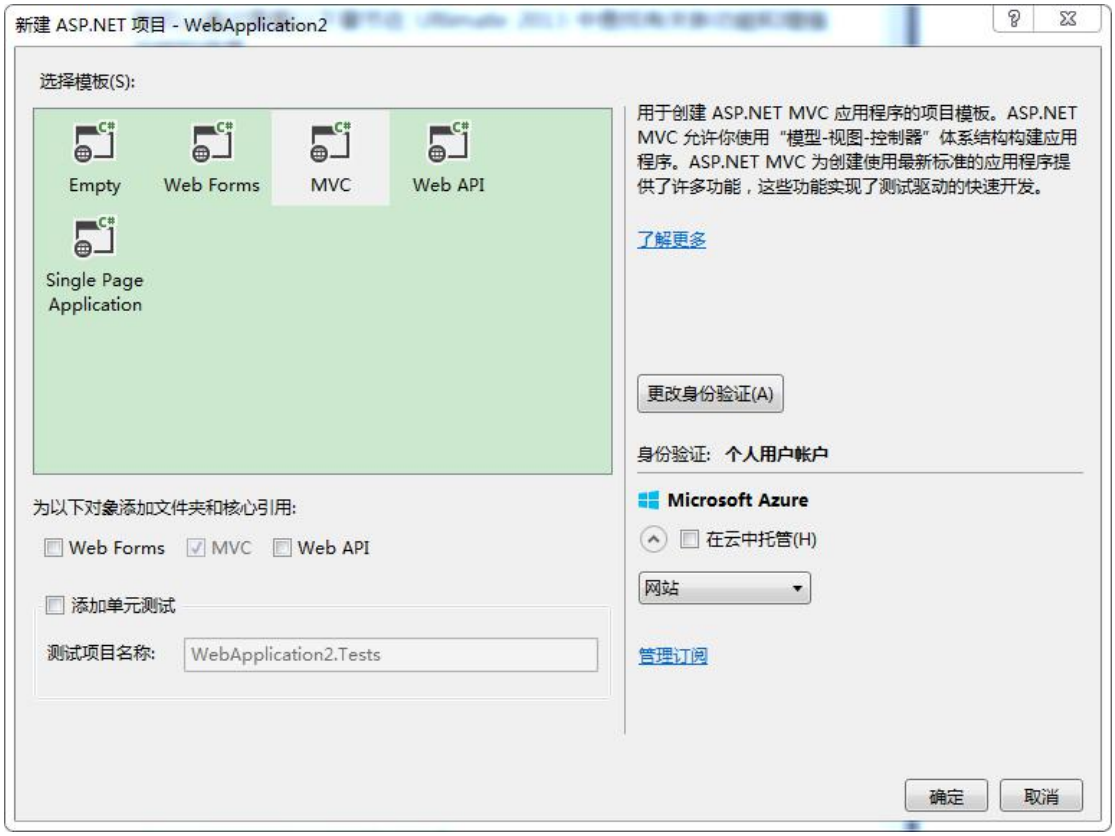


System.Web.Routing ： URL 路由。将一个 URL 路由到对应的 Controller 上靠的就是这个。是在 HttpModule 里面处理的。
System.Web.Extensions ： 这个是 ASP.NET AJAX 的。
System.Web.Mvc: ASP.NET MVC 最主要的程序集。在 CodePlex 上放出源代码的就是这个 DLL。
System.Web.Abstractions ： 这个程序集是一些相关的基类来的。例如 HttpContextBase、HttpRequestBase 等等。
Microsoft.Web.Mvc ： 这个程序集只要放一些 MVC 的特性与扩展的方法。

四、实验步骤：

新建数据库 test,并建表 T_Student
字段包括 学号，姓名，性别 QQ 号，宿舍,请用英文做字段名。
建立新的 ASP.net MVC 项目
① 打开 VS，新建项目，选择 ASP.NET MVC4 Web Application

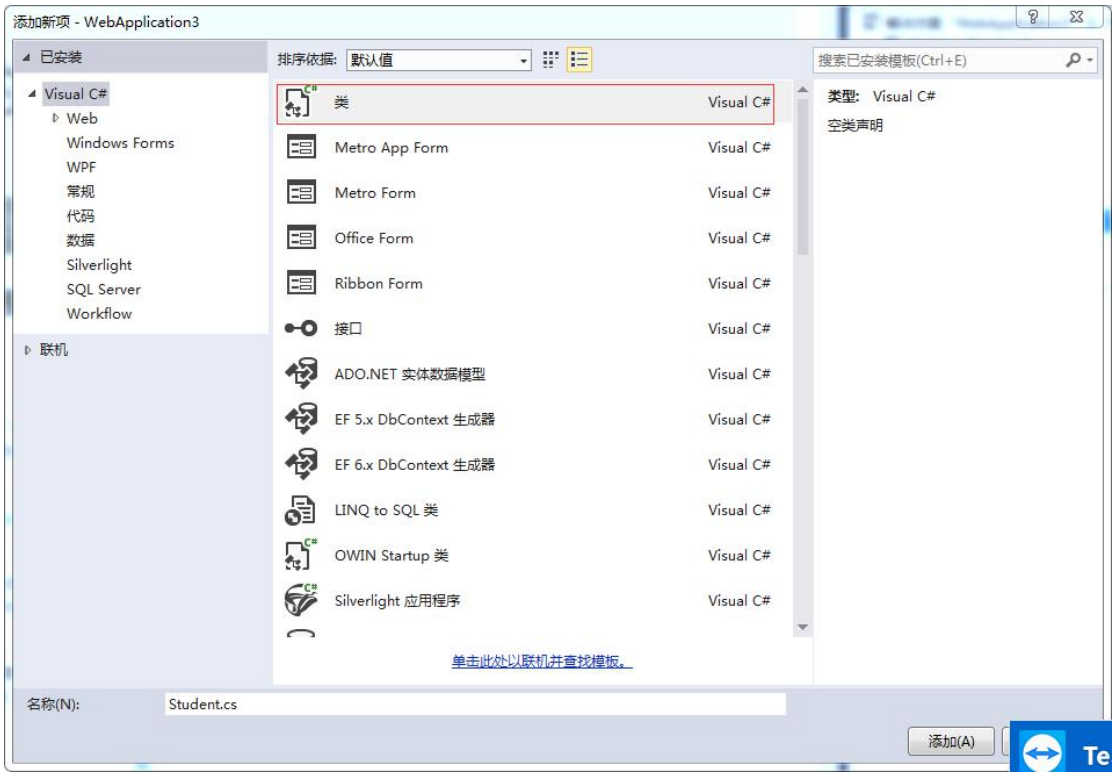




② ．创建数据模型类

右键 Models 文件夹，选择 “添加” → “类” →T_Student.cs 类。

选择”类”，并命名为 T_Student



输入如下代码：

```
public class T_student
{
    [Key]
    [Required(ErrorMessage = "不能为空")]
    [Display(Name = "学号")]
    0 个引用
    public string Sno { get; set; }

    [Required(ErrorMessage = "不能为空")]
    [Display(Name = "姓名")]
    0 个引用
    public string Sname { get; set; }

    [Required(ErrorMessage = "不能为空")]
    [Display(Name = "性别")]
    0 个引用
    public string Sex { get; set; }

    [Required(ErrorMessage = "不能为空")]
    [Display(Name = "qq")]
    0 个引用
    public string SQQ { get; set; }

    [Display(Name = "宿舍")]
    [Required(ErrorMessage = "不能为空")]
    0 个引用
    public string Sdormitory { get; set; }
}
```

并添加引用：

```
using System.ComponentModel.DataAnnotations;
using System.Data.Entity;
```

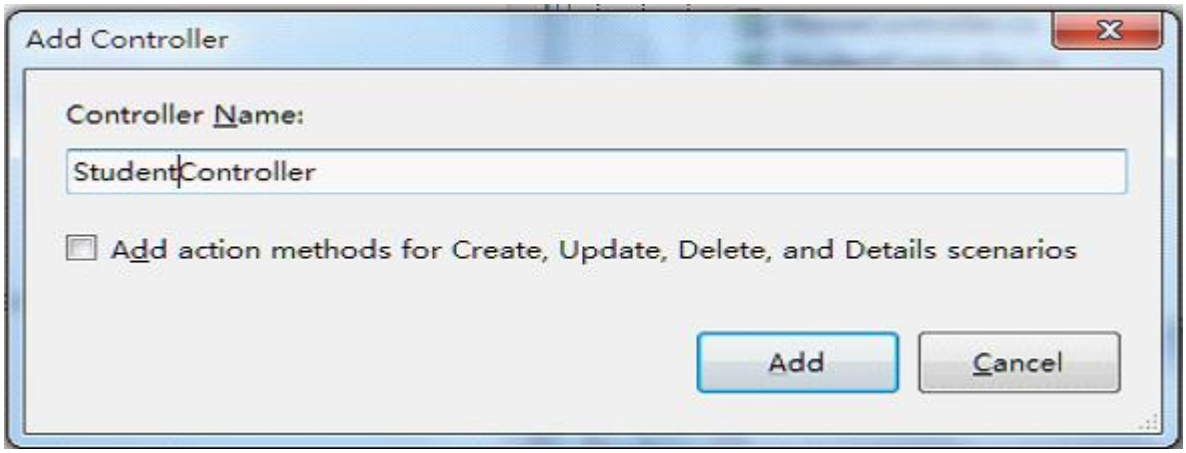
. 创建数据库上下文类

```
public class studentDBContext : DbContext
    {
        public DbSet<T_student> students { get; set; }
    }
```

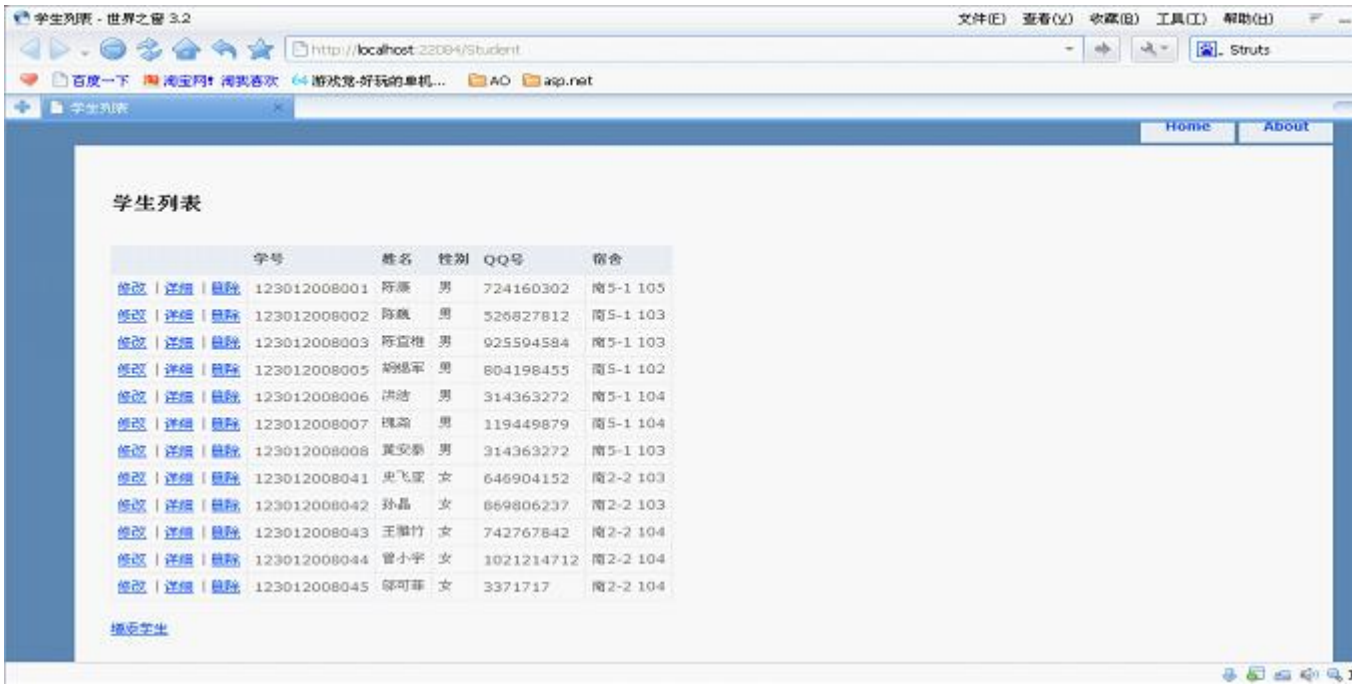
```
public class T_student
{
    [Key]
    [Required(ErrorMessage = "不能为空")]
    [Display(Name = "学号")]
    0 个引用
    public string Sno { get; set; }

    [Required(ErrorMessage = "不能为空")]
    [Display(Name = "姓名")]
    0 个引用
    public string Sname { get; set; }

    [Required(ErrorMessage = "不能为空")]
    [Display(Name = "性别")]
    0 个引用
    public string Sex { get; set; }

    [Required(ErrorMessage = "不能为空")]
    [Display(Name = "qq")]
    0 个引用
    public string SQQ { get; set; }

    [Display(Name = "宿舍")]
    [Required(ErrorMessage = "不能为空")]
    0 个引用
    public string Sdormitory { get; set; }
}

0 个引用
public class studentDBContext : DbContext
{
    [Key]
    0 个引用
    public DbSet<T_student> students { get; set; }
}
```

编写 Controller

① 在项目中找到"Controllers"文件夹右击，单击"添加" 再单击菜单"Controller…"，按下图填写：



实现学生信息的查询、增加、删除和修改功能，主要功能界面展示如下：

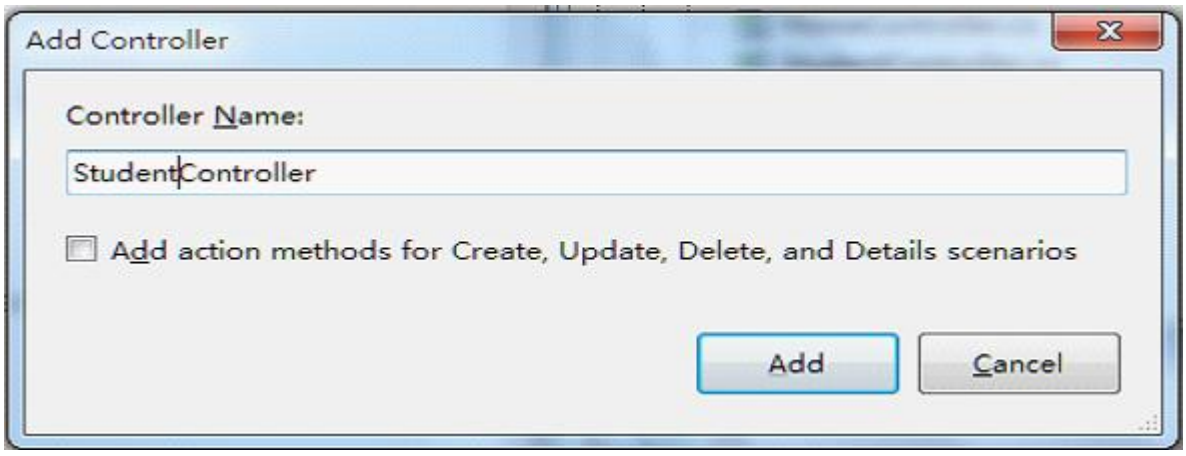


并配置好路由。

实验步骤

1. **编写 Controller**

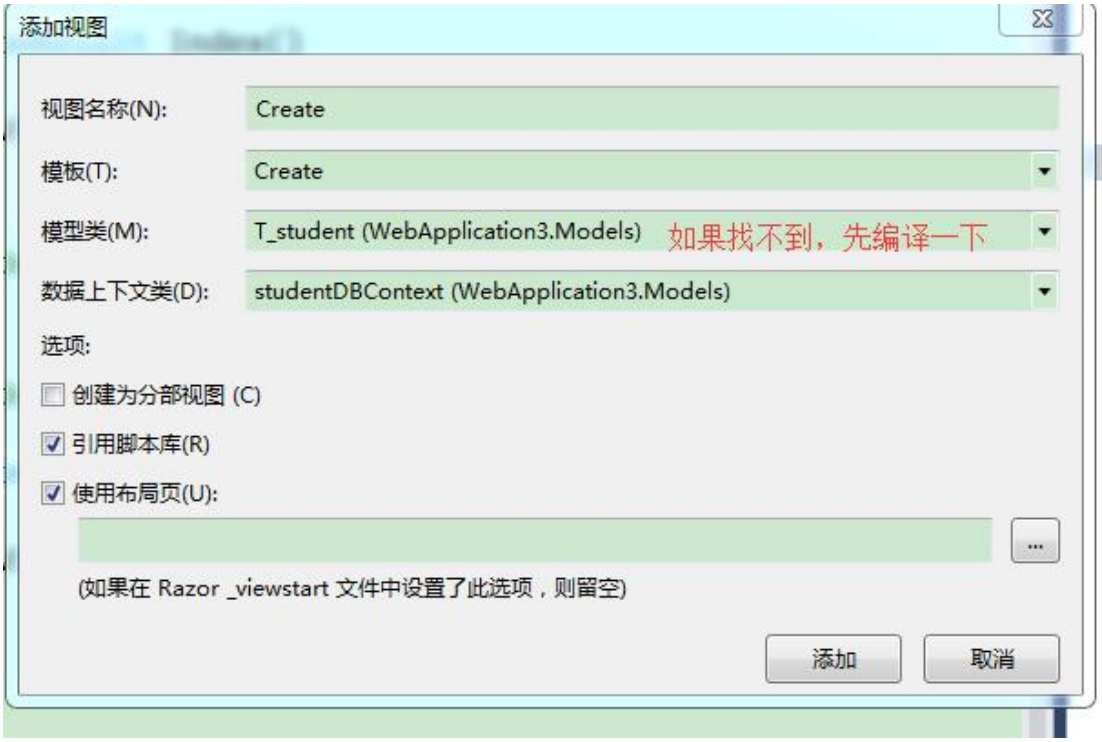
在项目中找到"Controllers"文件夹右击，单击"添加" 再单击菜单"Controller…"，按下图填写：



- 添加 Action：1 个 Create()用来显示添加界面、Index()用来显示所有学生
- 添加视图：Create、Index

```
namespace WebApplication3.Controllers
{
    public class studentController : Controller
    {
        // GET: student
        public ActionResult Index()
        {
            return View();
        }

        // GET: student/Details/5

        // GET: student/Create
        public ActionResult Create()
        {
            return View();
        }

    }
}
```

添加 Create 视图 ，在 create action 右击

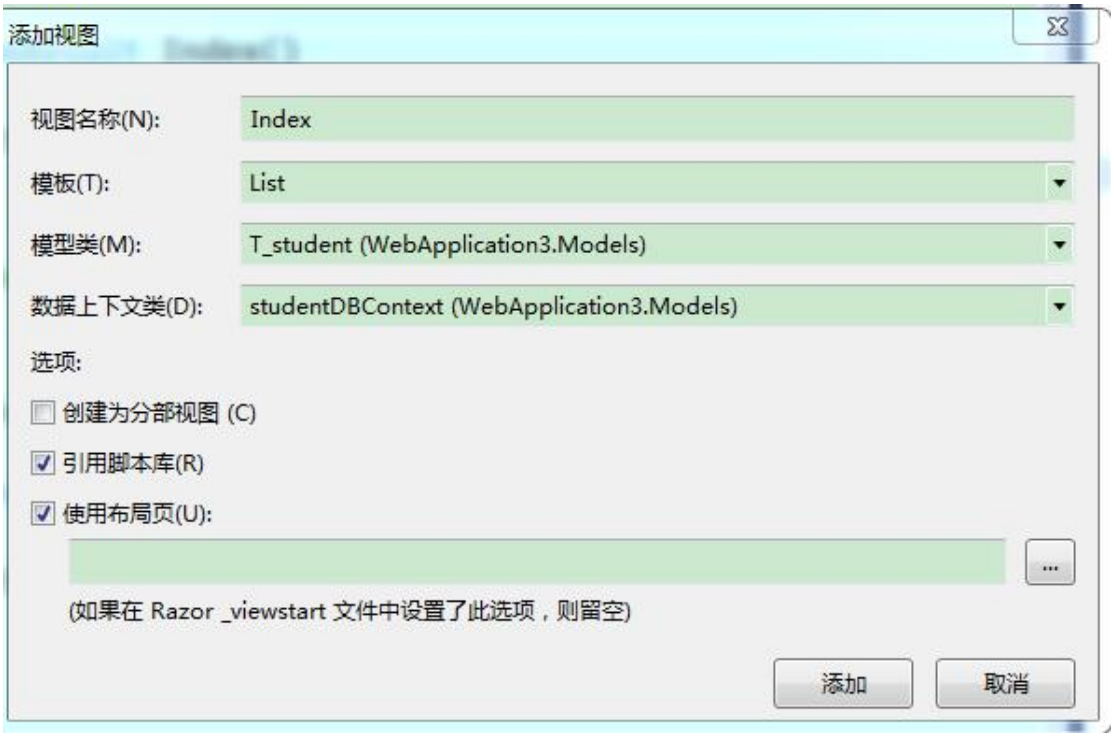



```
@section Scripts {
```

```
    @Scripts.Render("~/bundles/jqueryval")
}
```

添加 index 视图

在 index action 右击

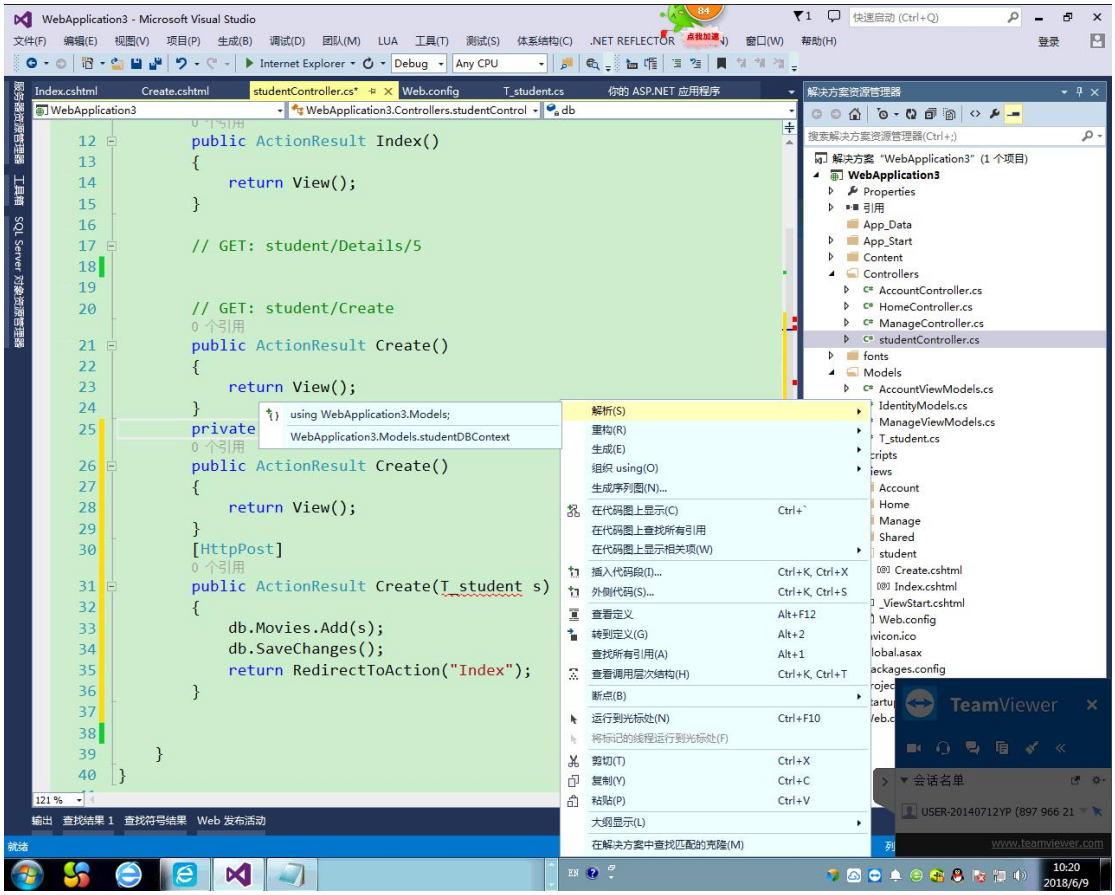


- 添加一个同名为 Create 的 Action，用 [HttpPost] 标识：

```
public class studentController : Controller
    {
        private studentDBContext db = new studentDBContext();
            // GET: student
        public ActionResult Index()
        {
            return View(  db.students.ToList() );
        }

          // GET: student/Create
        public ActionResult Create()
        {
            return View();
        }

        [HttpPost]
```

```
        public ActionResult Create(T_student s)
        {
            db.students.Add(s);
            db.SaveChanges();
            return RedirectToAction("Index");
        }

    }
```

若出现 db 识别不了，可右击 ，选择解析



修改，删除，详情
代码如下：

```
public class studentController : Controller
    {
        private studentDBContext db = new studentDBContext();
            // GET: student
        public ActionResult Index()
        {
            return View(  db.students.ToList() );
```

```
        }

          // GET: student/Create
        public ActionResult Create()
        {
            return View();
        }

        [HttpPost]
        public ActionResult Create(T_student s)
        {
            db.students.Add(s);
            db.SaveChanges();
            return RedirectToAction("Index");
        }


        public ActionResult Edit(string id)
        {
            T_student s = db.students.Find(id);  //查找给定主键的
实体
            if (s == null)
            {
                return HttpNotFound();
            }
            return View(s);
        }

        [HttpPost]
        [ValidateAntiForgeryToken]
        public ActionResult Edit(T_student s)
        {
            if (ModelState.IsValid)
            {
                db.Entry(s).State = EntityState.Modified;
                db.SaveChanges();
                return RedirectToAction("Index");
            }
            return View(s);
        }
        public ActionResult Delete(string id)
        {
            if (id == null)
            {
```

```
                return new
HttpStatusCodeResult(HttpStatusCode.BadRequest);
            }
            T_student s = db.students.Find(id);
            if (s == null)
            {
                return HttpNotFound();
            }
            return View(s);
        }
        [HttpPost]
        [ValidateAntiForgeryToken]
        [ActionName("Delete")]
        public ActionResult Delete1(string id)
        {
            T_student s = db.students.Find(id);
            db.students.Remove(s);
            db.SaveChanges();
            return RedirectToAction("Index");
        }
        public ActionResult Details(string id)
        {
            if (id == null)
            {
                return new
HttpStatusCodeResult(HttpStatusCode.BadRequest);
            }
            T_student s = db.students.Find(id);
            if (s == null)
            {
                return HttpNotFound();
            }
            return View(s);
        }



    }
```

实验二　表单和数据验证

一、实验目的

- 掌握 MVC 程序的数据提交
- 掌握 MVC 数据之间的传递

二、实验内容

- 用户注册
- 用户登录
- 修改密码
- 修改资料

1、新建用户模型：

```
public class User
    {
        [Key]
        public int UserId { get; set; }
        /// <summary>

        /// <summary>
        /// 用户名
        /// </summary>
        [Display(Name = "用户名", Description = "4-20个字符。")]
        [Required(ErrorMessage = "×")]
        [StringLength(20, MinimumLength = 4, ErrorMessage = "×")]
        public string UserName { get; set; }
        /// <summary>
        /// 密码
        /// </summary>
        [Display(Name = "密码")]
        [Required(ErrorMessage = "×")]
        [StringLength(512)]
        public string Password { get; set; }
        /// <summary>
        /// 性别【0-男；1-女；2-保密】
        /// </summary>
        [Display(Name = "性别",Description = "0-男；1-女；2-保密")]
        [Required(ErrorMessage = "×")]
        [Range(0, 2, ErrorMessage = "×")]
        public byte Gender { get; set; }
        /// <summary>
```

```
        /// Email
        /// </summary>
        [Display(Name = "Email", Description = "请输入您常用的
Email。")]
        [Required(ErrorMessage = "×")]

        public string Email { get; set; }
        /// <summary>
        /// 密保问题
        /// </summary>
        [Display(Name = "密保问题", Description = "请正确填写，在
您忘记密码时用户找回密码。4-20个字符。")]
        [Required(ErrorMessage = "×")]
        [StringLength(20, MinimumLength = 4, ErrorMessage = "×")]
        public string SecurityQuestion { get; set; }
        /// <summary>
        /// 密保答案
        /// </summary>
        [Display(Name = "密保答案", Description = "请认真填写，忘
记密码后回答正确才能找回密码。2-20个字符。")]
        [Required(ErrorMessage = "×")]
        [StringLength(20, MinimumLength = 2, ErrorMessage = "×")]
        public string SecurityAnswer { get; set; }
        /// <summary>
        /// QQ号码
        /// </summary>
        [Display(Name = "QQ号码",Description ="6-12位数")]
      //  [RegularExpression("^[1-9][0-9]{4-13}$", ErrorMessage
= "×")]
        [StringLength(12, MinimumLength = 6, ErrorMessage = "×")]
        public string QQ { get; set; }
        /// <summary>
        /// 电话号码
        /// </summary>
        [Display(Name = "电话号码", Description = "常用的联系电话
（手机或固话），固话格式为：区号-号码。")]
      //  [RegularExpression("^[0-9-]{11-13}$", ErrorMessage =
"×")]
        public string Tel { get; set; }
        /// <summary>
        /// 联系地址
        /// </summary>
        [Display(Name = "联系地址", Description = "常用地址，最多
80个字符。")]
```

```
        [StringLength(80, ErrorMessage = "×")]
        public string Address { get; set; }
        /// <summary>
        /// 邮编
        /// </summary>
        [Display(Name = "邮编", Description = "6位字符。")]
        [RegularExpression("^[0-9]{6}$", ErrorMessage = "×")]
        public string PostCode { get; set; }
        /// <summary>
        /// 注册时间
        /// </summary>
        public DateTime? RegTime { get; set; }
        /// <summary>
        /// 上次登录时间
        /// </summary>
        public DateTime? LastLoginTime { get; set; }

    }

    public class userDBContext : DbContext
    {
        public DbSet<User> Users { get; set; }
    }

    }
```

2、新建注册模型

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.ComponentModel.DataAnnotations;
using System.Web.Mvc;

namespace MvcApplication2.Models
{
    public class UserRegister :User
    {
        /// <summary>
        /// 密码
        /// </summary>
        [Display(Name = "密码", Description = "6-20个字符。")]
        [Required(ErrorMessage = "×")]
```

```
        [StringLength(20, MinimumLength = 6, ErrorMessage = "×")]
        [DataType(DataType.Password)]
        public new string Password { get; set; }
        /// <summary>
        /// 确认密码
        /// </summary>
        [Display(Name = "确认密码", Description = "再次输入密码。
")]
        [Compare("Password", ErrorMessage = "×")]
        [DataType(DataType.Password)]
        public string RePassword { get; set; }

    }
}
```


```
  public ActionResult Register()
        {
            return View();
        }
        userDBContext db = new userDBContext();
        [HttpPost]
       public ActionResult Register(UserRegister userReg)
        {
            string username = userReg.UserName;
            var result = db.Users.Where(rrrr => rrrr.UserName ==
username).FirstOrDefault();
```

```
            if (result != null)
            {
                ModelState.AddModelError("UserName", "用户名已存在
");
                return View();
            }
            else
            {

                User _user =userReg;
                _user.Password = userReg.Password;
                _user.UserName = userReg.UserName;
                db.Users.Add(_user);
                db.SaveChanges();

            }
            return View("Index", db.Users);
        }


 public ActionResult Index()
        {
            return View();
        }
```

创建 index 视图，内容包括如下：

```
<ul>
    <li>@Html.ActionLink("用户首页","Default","User")</li>
    <li>@Html.ActionLink("修改信息","ChangeInfo","User")</li>
    <li>@Html.ActionLink("修改密码","ChangePassword","User")</li>
    <li>@Html.ActionLink("退出系统","Logout","User")</li>
</ul>
```

2、登陆

1）创建登陆类

```
 public class UserLogin
    {
        //用户名
```

```
        [Display(Name = "用户名", Description = "4-20 个字符。")]
        [Required(ErrorMessage = "x")]
        [StringLength(20, MinimumLength = 4, ErrorMessage = "x")]
        public string UserName { get; set; }

        //密码
        [Display(Name = "密码", Description = "6-20 个字符。")]
        [Required(ErrorMessage = "x")]
        [StringLength(20, MinimumLength = 6, ErrorMessage = "x")]
        [DataType(DataType.Password)]
        public string Password { get; set; }
    }
```

2）显示登陆界面

```
   public ActionResult Login()
        {
            FormsAuthentication.SignOut();
            return View();
        }
```

创建登陆视图，模板类选择 UserLogin,视图选择 create

3）登陆处理模块

```
  public ActionResult Login(UserLogin login)
        {
            string username = login.UserName;
            string password = login.Password;
            var result = db.Users.Where(u => u.UserName ==
login.UserName && u.Password == login.Password).FirstOrDefault();

            if (result == null)
            {
                ModelState.AddModelError("Message", "登录失败！");
                return View("Login");
            }
            else
            {
                FormsAuthentication.SetAuthCookie(login.UserName,
false);
                return RedirectToAction("Index");
            }
        }
```

3、修改个人资料

1） 显示个人修改界面

```
 public class MyAuth : AuthorizeAttribute
    {
        public override void OnAuthorization(AuthorizationContext
filterContext)
        {
            string currentrole =
GetRole(filterContext.HttpContext.User.Identity.Name);
            if (Roles.Contains(currentrole))
            {

            }
            else
            {

//filterContext.RequestContext.HttpContext.Response.Write("没权
限");

//filterContext.RequestContext.HttpContext.Response.End();
            }
            /// base.OnAuthorization(filterContext);
        }

        public string GetRole(string name)
        {
            //switch (name)
            //{
            //    case "123456": return "User";
            //    case "666666": return "User";
            //    case "zzyzzy": return "Admin";
            //    default: return "error";

            //}
            if (name == "123456")
                return "Admin";
            else
                return "User";
        }
    }


 [MyAuth]
            public ActionResult ChangeInfo()
```

```
            {
                string username = HttpContext.User.Identity.Name;
                var result = db.Users.SingleOrDefault<User>(u =>
u.UserName == username);

                if (result != null)
                {
                    result.Password = "";
                    return View(result);
                }
                else
                {
                    return Content("<script
language='javascript' type = 'text/javascript' > alert('无接收数
据！ '); window.location.href = 'login' </ script > ");
                }
            }
```

创建修改视图。模板类选择 User ,视图选择 Edit

2）保存处理

```
  [HttpPost]
            public ActionResult ChangeInfo(User user)
            {
                var result = db.Users.Where(u => u.UserName ==
user.UserName && u.Password == user.Password);
                User us = result.FirstOrDefault<User>();
                if (us == null)
                {
                    ModelState.AddModelError("Message", "密码错
误！ ");
                    return View(user);
                }
                else
                {
                    us.Address = user.Address;
                    us.Email = user.Email;
                    us.PostCode = user.PostCode;
                    us.QQ = user.QQ;
                    us.Tel = user.Tel;
                    db.Entry(us).State = EntityState.Modified;

                    DbEntityEntry<User> entry = db.Entry(us);
```

```
                    entry.Property("Password").IsModified =
true;
                    db.Configuration.ValidateOnSaveEnabled =
false;
                    int count = db.SaveChanges();
                    db.Configuration.ValidateOnSaveEnabled =
true;

                    return Content("<script
language='javascript' type='text/javascript'>alert('信息修改成
功！');window.location.href= 'default'</script>");
                }
            }
```

3、修改密码

1）显示修改界面

```
 public class UserChangePassword
    {
        [NotMapped]

            //原密码
            [Display(Name = "原密码")]
            [Required(ErrorMessage = "x")]
            [StringLength(20, MinimumLength = 6, ErrorMessage =
"x")]
            [DataType(DataType.Password)]
            public string Password { get; set; }

            //新密码
            [Display(Name = "新密码", Description = "6-20 个字符")]
            [Required(ErrorMessage = "x")]
            [StringLength(20, MinimumLength = 6, ErrorMessage =
"x")]
            [DataType(DataType.Password)]
            public string NewPassword { get; set; }

            //确认密码
            [Display(Name = "确认密码", Description = "再次输入密
码")]
            [Compare("NewPassword", ErrorMessage = "x")]
            [DataType(DataType.Password)]
            public string ConfirmPassword { get; set; }
```

```
        }
    public ActionResult ChangePassword()
        {
            return View();
        }
```

创建修改视图。模板类选择 UserChangePassword,视图选择 Create

2）保存

```
    [HttpPost]
        public ActionResult ChangePassword(string Password,
string NewPassword, string ConfirmPassword)
        {

            string username = HttpContext.User.Identity.Name;
            var result = db.Users.Where(u => u.UserName == username
&& u.Password == Password);
            User us = result.FirstOrDefault<User>();
            if (us == null)
            {
                ModelState.AddModelError("Message", "修改失败，无
此用户！");
                return View();
            }
            else
            {
                if (ModelState.IsValid)
                {
                    us.UserName = username;
                    us.Password = NewPassword;
                    db.Entry(us).State = EntityState.Modified;

                    DbEntityEntry<User> entry = db.Entry(us);

                    entry.Property("Password").IsModified = true;
                    db.Configuration.ValidateOnSaveEnabled =
false;
                    int count = db.SaveChanges();
                    db.Configuration.ValidateOnSaveEnabled =
true;

                    return Content("<script language='javascript'
```

```
type='text/javascript'>alert('修改密码成功!
');window.location.href= 'index'</script>");
                }
                else
                {
                    ModelState.AddModelError("Message", "修改失败!
");
                    return View();
                }
            }
        }
```

实验三　个人新闻管理

一、实验目的

- 掌握基本的数据库编程知识，
- 实现个人新闻管理，包括新闻列表，添加，删除，修改

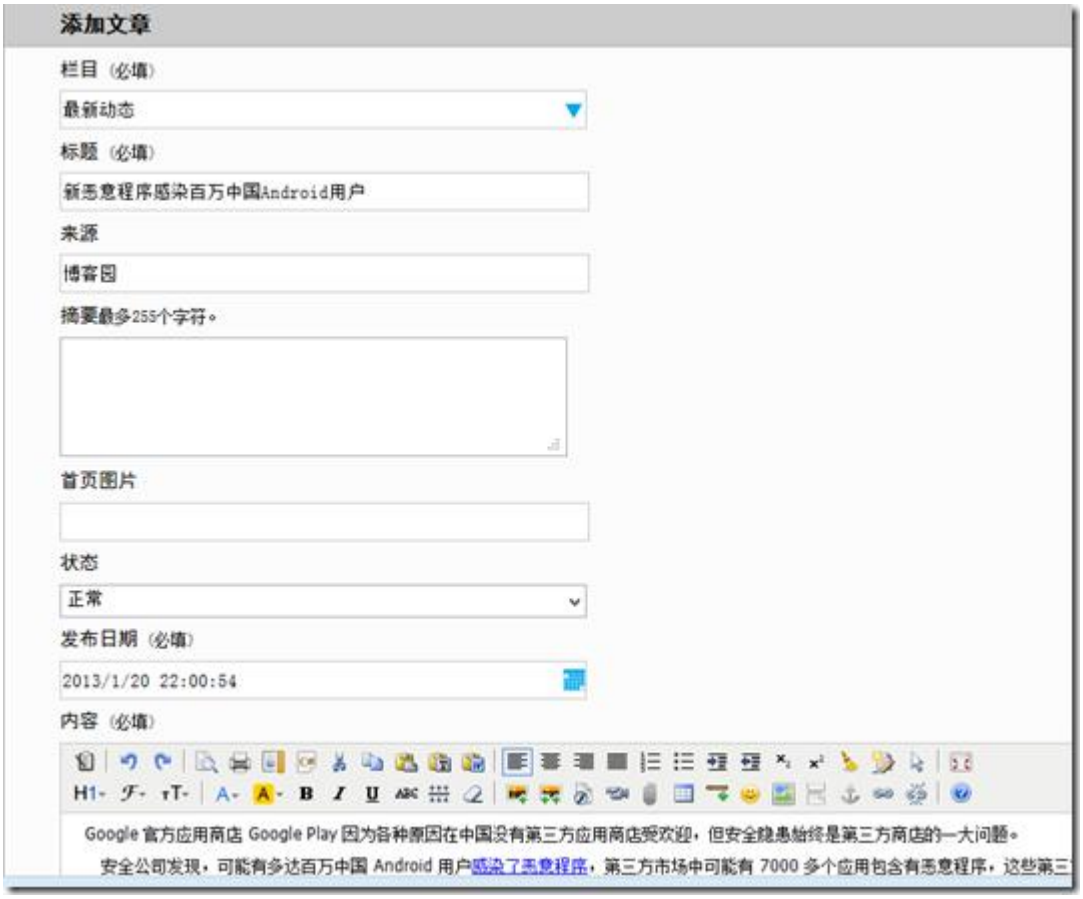
二、实验内容

一般来说一个网站最主要的部分就是文章，实现了文章功能网站的核心也就出来了。包含以下字段。

字段	名称	类型	必填
ArtickeId	文章 id	Int[key]	是
CommonModelId	栏目 Id	Int	是
Source	来源	string(255)	
Author	作者	string(50)	
Intro	摘要	string(255)	
Content	内容	string	是

栏目表：

字段	名称	类型	必填
CategoryId	栏目 Id	Int[key]	是
CategoryName	栏目名	Int	是



参考代码：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using System.Data.Entity;
```

```
namespace MvcApplication2.Models
{
    public class User
    {
        [Key]
        public int UserId { get; set; }

        public string UserName { get; set; }

        /// </summary>

    }

    public class Category
    {
        [Key]
        public int CategoryId { get; set; }
        public string CategoryName { get; set; }
    }
    public class Article
    {
        [Key]
        public int ArticleId { get; set; }
        /// <summary>
        ///  栏目 id
        /// </summary>
        [Display(Name = "栏目ID")]
        [Required(ErrorMessage = "×")]
        public int CategoryId { get; set; }
        /// <summary>
        ///  来源
        /// </summary>
        [Display(Name = "来源")]
        [StringLength(255, ErrorMessage = "×")]
        public string Source { get; set; }
        /// <summary>
        ///  作者
        /// </summary>
        [Display(Name = "作者")]
        [StringLength(50, ErrorMessage = "×")]
        public string Author { get; set; }
        /// <summary>
        ///  摘要
        /// </summary>  [NotMapped]
```

```
        [Display(Name = "摘要")]
        public string Intro { get; set; }
        /// <summary>
        ///  内容
        /// </summary>
        [Display(Name = "内容")]
        [Required(ErrorMessage = "×")]
        [DataType(DataType.Html)]
        public string Content { get; set; }

    }
    public class userDBContext : DbContext
    {
        public DbSet<User> Users { get; set; }
        public DbSet<Category> Categorys { get; set; }
        public DbSet<Article> Articles { get; set; }
    }

}
```

栏目：

```
using MvcApplication2.Models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace MvcApplication2.Controllers
{
    public class lanmuController : Controller
    {
        userDBContext db = new userDBContext();
        //
        // GET: /lanmu/

        public ActionResult Index()
        {
            return View(db.Categorys);
        }
        public ActionResult Create()
        {
            return View();
        }
```

```
        [HttpPost]
        public ActionResult Create(Category c)
        {
            db.Categorys.Add(c);
            db.SaveChanges();
            return View();
        }

    }
}
```

文章：

```
using MvcApplication2.Models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using MvcApplication2.Models;
using System.Data;
namespace MvcApplication2.Controllers
{

    public class NewsController : Controller
    {
        //
        // GET: /News/
        userDBContext db = new userDBContext();
        static int now = 1;
        static int z = 0;
        public ActionResult Index(int? pageno)
        {
            if (pageno == null)
                pageno = 1;

            int prepage = (int)pageno - 1;
            int nextpage = (int)pageno + 1;
            int pagecount = db.Articles.Count() / 6 + 1;

            if (prepage < 1) prepage = 1;
            if (nextpage > pagecount) nextpage = pagecount;
```

```
            ViewBag.page = "<a href='/news/index?pageno=1'>首页
</a>    ";
            ViewBag.page += "<a href='/news/index?pageno=" +
prepage + "'>上一页</a>    ";
            ViewBag.page += pageno +
"</a>    ";

            ViewBag.page += "<a href='/news/index?pageno=" +
nextpage + "'>下一页</a>    ";
            ViewBag.page += "<a href='/news/index?pageno=" +
pagecount + "'>末页</a>    ";

            var q = db.Articles.OrderBy(o =>
o.ArticleId).Skip(((int)pageno - 1) * 6).Take(6);
            return View("index", q);


        }
        public ActionResult Index1(int? pageno, int? kind)
        {
            if (pageno == null)
                pageno = 1;

            int prepage = (int)pageno - 1;
            int nextpage = (int)pageno + 1;
            if (kind == null)
            {
                kind = 0;
            }
            z = (int)kind;
            var news = from m in db.Articles
                       select m;
            if (kind != 0)
            {
                news = news.Where(s => s.CategoryId.Equals(z));
            }
            int pagecount = news.Count() / 6 + 1;

            if (prepage < 1) prepage = 1;
            if (nextpage >= pagecount) nextpage = pagecount;

            ViewBag.page = "<a
href='/news/index1?pageno=1&&kind=" + kind + "'>首页
```

```
</a>    ";
            ViewBag.page += "<a href='/news/index1?pageno=" +
prepage + "&&kind=" + kind + "'>上一页
</a>    ";
            ViewBag.page += pageno +
"</a>    ";

            ViewBag.page += "<a href='/news/index1?pageno=" +
nextpage + "&&kind=" + kind + "'>下一页
</a>    ";
            ViewBag.page += "<a href='/news/index1?pageno=" +
pagecount + "&&kind=" + kind + "'>末页
</a>    ";
            now = (int)pageno;


            return View();
        }
        public ActionResult Index2()
        {
            var news = from m in db.Articles
                       select m;
            if (z != 0)
            {
                news = news.Where(s => s.CategoryId.Equals(z));
            }
            var q = news.OrderBy(o => o.ArticleId).Skip(((int)now
- 1) * 6).Take(6);
            return Json(q, JsonRequestBehavior.AllowGet);
            //     return Json(db.Articles,
JsonRequestBehavior.AllowGet);
        }
        public ActionResult Index3(int? id)
        {
            Article article = db.Articles.Find(id); //找到
            if (article == null)
            {
                return HttpNotFound();
            }
            return View(article); //放到视图里访问
        }
        public ActionResult Create()
        {
            ViewData["CategoryId"] = new SelectList(db.Categorys,
```

```
"CategoryId", "CategoryName");
            return View();
        }
        [HttpPost, ValidateInput(false)]
        public ActionResult Create(Article a)
        {
            if (ModelState.IsValid) //如果数据符合要求 （ 按照规则
校验）
            {
                db.Articles.Add(a);
                db.SaveChanges();
            }
            return RedirectToAction("index");
        }
        public ActionResult Edit(int id)
        {
            Article a = db.Articles.SingleOrDefault(m =>
m.ArticleId == id);
            ViewData["CategoryId"] = new SelectList(db.Categorys,
"CategoryId", "CategoryName", a.CategoryId);
            return View(a);
        }
        [HttpPost, ValidateInput(false)]
        public ActionResult Edit(Article article) //修改的时候提
交的数据时一个类
        {
            if (ModelState.IsValid)
            {
                db.Entry(article).State = EntityState.Modified;//
修改
                db.SaveChanges(); //保存
                return RedirectToAction("index");
            }

            return View("index");
        }


        public ActionResult Delete(int? id)
        {
            if (id == null)
            {
                return HttpNotFound();
            }
```

```
            Article article = db.Articles.Find(id);
            if (article == null)
            {
                return HttpNotFound();
            }
            return View(article);
        }
        [HttpPost]
        public ActionResult Delete(int id) //删除的时候提交的数据只是id  和修改不一样
        {
            Article article = db.Articles.Find(id); //找到
            db.Articles.Remove(article);//删除
            db.SaveChanges();//保存
            return RedirectToAction("index");
        }

        public ActionResult Details(int? id) //删除的时候提交的数据只是id  和修改不一样
        {
            Article article = db.Articles.Find(id); //找到
            if (article == null)
            {
                return HttpNotFound();
            }
            return View(article); //放到视图里访问
        }

    }
}
```

添加、修改视图：

```
@model MvcApplication2.Models.Article

@{
    ViewBag.Title = "Create";
}

<h2>Create</h2>

@using (Html.BeginForm()) {
    @Html.AntiForgeryToken()
    @Html.ValidationSummary(true)
```

```
    <fieldset>
        <legend>Article</legend>

        <div class="editor-label">
            @Html.LabelFor(model => model.CategoryId)
        </div>
        <div class="editor-field">
            @Html.DropDownList("CategoryId")
            @Html.ValidationMessageFor(model => model.CategoryId)
        </div>

        <div class="editor-label">
            @Html.LabelFor(model => model.Source)
        </div>
        <div class="editor-field">
            @Html.EditorFor(model => model.Source)
            @Html.ValidationMessageFor(model => model.Source)
        </div>

        <div class="editor-label">
            @Html.LabelFor(model => model.Author)
        </div>
        <div class="editor-field">
            @Html.EditorFor(model => model.Author)
            @Html.ValidationMessageFor(model => model.Author)
        </div>

        <div class="editor-label">
            @Html.LabelFor(model => model.Intro)
        </div>
        <div class="editor-field">
            @Html.EditorFor(model => model.Intro)
            @Html.ValidationMessageFor(model => model.Intro)
        </div>

        <div class="editor-label">
            @Html.LabelFor(model => model.Content)
        </div>
        <div class="editor-field">
         @Html.TextAreaFor(model => model.Content, new { @class =
"content" })
            @Html.ValidationMessageFor(model => model.Content)
        </div>
```

```
        <p>
            <input type="submit" value="Create" />
        </p>
    </fieldset>
}

<div>
    @Html.ActionLink("Back to List", "Index")
</div>

@section Scripts {
    @Scripts.Render("~/bundles/jqueryval")
 <script src="~/Scripts/kindeditor/kindeditor-min.js"></script>
<script type="text/javascript">
    KindEditor.ready(function (K) {
        K.create('#Content');
    });
    </script>
    }
```

实验四　个人新闻显示

一、实验目的

- 掌握 JQuery, AJAX 辅助方法，实现个人新闻的前台列表显示和和新闻的详情显示，要求有分页

二、实验内容

文章的前台文章列表，点击可查看详情。


Firefox
Twitter开源了基于事件的组件框架Fligh...
localhost:52270/Items/10
首页 | 登陆 | 注册 | 加入收藏
网站首页
最新动态
产品介绍
客户服务
互动论坛
关于我们
最新动态
111
栏目测试
22222
3333
您现在的位置：网站首页 >> 最新动态 >> 详细内容
Twitter开源了基于事件的组件框架Flight
来源： 作者： 发布时间：2013年2月4日 阅读：0
Twitter 开源了 Flight，这是该公司内部用于产品中的、为网站提供功能的 JavaScript 框架。
Twitter Flight 是基于事件的框架，可用于为现有 DOM 结构添加功能。Flight 既不用于建模数据，也不用于 HTML 渲染，它不处理请求路由和页面模版，也不处理浏览器或服务器端的渲染。相反，它是一款高度解耦和的组件框架，这些组件连接到 DOM 节点，当在节点上调用事件时，可以提供预期的功能。
Flight 组件完全不知道其他组件的存在，一个组件也不能持有指向另一个组件的引用。组件是分别定义的，当它们被连接到节点时才会创建实例。组件会持有所连接节点的 引用。之后为事件注册组件，事件会使用 DOM 的事件机制进行发送和接收。在接收到事件时，组件并不知道事件是来自节点还是来自另一个组件。它只是执行在组件定义时确定的、与各个事件关联的功能。
借助这种事件机制，Flight 支持开发者编写、测试和调试解耦和的组件，而无需依赖其他组件。即便其他组件出现问题，当前组件应该还能工作。
该框架也定义了 mixin，这是可以在组件和其他 mixin 间共享的小功能片段。Mixin 和面向对象语言（如 Java 或C#）中的接口比较相似。
去年，Twitter 在 对其网站进行更新时 开始在产品中使用 Flight。现在 Twitter 基于 MIT 许可证 开放了 Flight 的源代码，同时提供了一个 演示应用，还提供了用于解释如何设置 Flight、如何开始创建、测试和调试组件的 文档。
Flight 也依赖其他框架，包括使用 ES5-shim 来支持较老的浏览器，使用 JQuery 的 DOM 操作 API。该框架支持所有主要浏览器，包括
0K/S