

项目基础组件选型

一、Rupeng.Common 通用类库项目

- a) 复习：什么是 MD5 算法（三层架构中）？特点？应用？
- b) 封装计算字符串和 Stream 的两个 CalcMD5 方法（代码不用背）

```
public static string CalcMD5(this string str)
{
    byte[] bytes = System.Text.Encoding.UTF8.GetBytes(str);
    return CalcMD5(bytes);
}

public static string CalcMD5(byte[] bytes)
{
    using (MD5 md5 = MD5.Create())
    {
        byte[] computeBytes = md5.ComputeHash(bytes);
        string result = "";
        for (int i = 0; i < computeBytes.Length; i++)
        {
            result += computeBytes[i].ToString("X").Length == 1 ? "0" + computeBytes[i].ToString("X") : computeBytes[i].ToString("X");
        }
        return result;
    }
}

public static string CalcMD5(Stream stream)
{
    using (MD5 md5 = MD5.Create())
    {
        byte[] computeBytes = md5.ComputeHash(stream);
        string result = "";
        for (int i = 0; i < computeBytes.Length; i++)
        {
            result += computeBytes[i].ToString("X").Length == 1 ? "0" + computeBytes[i].ToString("X") : computeBytes[i].ToString("X");
        }
        return result;
    }
}
```

c) 封装一个生成 n 位验证码的函数，为了避免混淆，不生成'1'、'l'、'0'、'o'等字符：

```
public static string GenerateCaptchaCode(int len)
{
    char[] data = {'a','c','d','e','f','g','k','m','p','r','s','t','w','x','y','3','4','5','7','8'};
    StringBuilder sbCode = new StringBuilder();
    Random rand = new Random();
    for(int i=0;i<len;i++)
    {
        char ch = data[rand.Next(data.Length)];
        sbCode.Append(ch);
    }
    return sbCode.ToString();
}
```

二、邮件发送

如果大量通过邮箱发送验证码，可以借助于 **SendCloud**、**阿里云** 等第三方的“触发邮件”服务，可以保证到达率。

如果是公司内部发邮件，可以自己搭建邮件服务器或者购买企业邮箱服务；

如果使用第三方免费的 **smtp** 服务器，可能有“授权码”的问题，需要用授权码（为什么）代替密码，还可能要启用 **https**。如果搞不定，就看课程旁边，我们会推荐当时能用的免费邮件服务。项目中不会有这个问题，因为都用自己的服务器或者购买收费的邮局服务。

开发测试最好自己发给自己，否则会有垃圾邮件问题，但也别发的太多，如果被封只能换服务商。

以 **163** 为例，注册过程自己搞。登录账号：**rupengtest01@163.com** 密码 **rupeng123**（别用老师的，自己搞）

1) 首先确保开启了 **Smtp** 服务；

2) 还需要开启、设置“客户端授权码”（有的邮箱不需要） 授权码：**123rupeng**

3) 发邮件的代码：

```
using (MailMessage mailMessage = new MailMessage())
using (SmtpClient smtpClient = new SmtpClient("Smtp 服务器"))
{
    mailMessage.To.Add("接收邮箱");
    mailMessage.To.Add("接收邮箱 2");
    mailMessage.Body = "邮件正文";
    mailMessage.From = new MailAddress("发送邮箱");
    mailMessage.Subject = "邮件标题";
    smtpClient.Credentials = new System.Net.NetworkCredential("发送用户名", "发送密码");//如果启用了“客户端授权码”，要用授权码代替密码
    smtpClient.Send(mailMessage);
}
```

如果启用了 **ssl**，并且不支持非安全连接，还需要设置 **smptClient.EnableSsl=true;** 只要支持 **ssl** 建议都开启 **ssl**，这样数据传输更安全。

三、缩略图、水印

为什么要生成缩略图？降低列表页面图片加载的速度、降低流量，想仔细看的时候再看大图；

什么要有水印？宣誓主权，增加曝光；

怎么样自己做缩略图？水印？要考虑什么问题？

nuget 上寻宝，发现一个用起来最爽的开发包：**CodeCarvings.Piczard**

1) 安装：**Install-Package CodeCarvings.Piczard**

2) 缩略图

```
ImageProcessingJob jobThumb = new ImageProcessingJob();
jobThumb.Filters.Add(new FixedResizeConstraint(200, 200)); //缩略图尺寸 200*200
```

调用 `ImageProcessingJob` 的 `SaveProcessedImageToFileSystem(object source, string destFilePath)` 可以把 `source`（一般是文件流、也可以是文件路径，这个接口设计不好）；`destFilePath` 是目标文件名

也可以调用 `ImageProcessingJob` 的 `SaveProcessedImageToStream(object source, Stream destStream)` 把 `source`（一般是文件流、也可以是文件路径，这个接口设计不好）；`destStream` 是目标流

保存的默认是 `png` 格式。如果想保存成其他格式，只要传最后一个 `FormatEncoderParams` 类型的参数即可，有 `BmpFormatEncoderParams`、`GifFormatEncoderParams`、`JpegFormatEncoderParams`、`PngFormatEncoderParams` 等子类。

3) 水印

```
ImageWatermark imgWatermark = new ImageWatermark(水印图片路径);
imgWatermark.ContentAlignment = System.Drawing.ContentAlignment.BottomRight; //水印位置
```

```
imgWatermark.Alpha = 50; //透明度，需要水印图片是背景透明的 png 图片
ImageProcessingJob jobNormal = new ImageProcessingJob();
jobNormal.Filters.Add(imgWatermark); //添加水印
jobNormal.Filters.Add(new FixedResizeConstraint(600, 600)); //限制图片的大小，避免生成大图。如果想原图大小处理，就不用加这个 Filter
```

然后调用 `SaveProcessedImageToFileSystem` 或者 `SaveProcessedImageToStream` 来保存文件

这个组件还可以完成很多其他工作：截图图片一部分；图片缩放等。

四、验证码组件

1、复习：为什么需要验证码；验证码存在哪里；如何检验验证码输入是否正确。

2、验证码图形生成不当会造成很容易的被识别出来，需要专业人员研究。如鹏从 Nuget 中寻宝了一个识别难度较大的验证码组件：`CaptchaGen`。

3、安装：**Install-Package CaptchaGen**

4、`CaptchaGen` 官方文档给出的用法太复杂，耦合性太强。

`GenerateImage` 的参数依次是：验证码文字（需要自己写代码生成，调用 `Rupeng.Common` 我们封装好的）；宽度；高度；字体大小；扭曲程度，数值越大扭曲越厉害；

//jpeg 格式的图片流

```
using (MemoryStream ms = ImageFactory.GenerateImage("AB12", 60, 100, 20, 6))
using (FileStream fs = File.OpenWrite(@"c:\temp\1.jpg"))
{
```

```
        ms.CopyTo(fs);  
    }  
5、(*) 闲聊：打码平台；  
6、(*) 闲聊：行为识别验证码；极验验证 (http://www.geetest.com/)
```

五、日志系统

- 1、什么是日志，为什么需要记录日志？查看“如鹏网日志例子.txt”感受一下；
- 2、Log4NET 是一个从 Java 版的 Log4J 移植过来的日志框架，可以简化日志的记录；
- 3、Log4NET 的概念：
 - a) 级别：trace、debug、info、warn、error、fatal。常用 **debug**（调试信息，程序员临时跟踪执行，在正式运行的项目中应该不显示）；**warn**（警告）；**error**（错误）。
 - b) 特殊的级别：all（全部显示）；off（全部不显示）；
 - c) appender：可以把日志输出到控制台、文件、数据库、ftp 服务器，甚至可以把日志输出到邮件、短信等。不同的输出场景就是不同的 appender，可以添加多个 appender，可以设定不同的级别级别使用不同的 appender
 - d) 什么是“滚动日志”？为什么要限制日志文件的大小和个数？
- 4、具体用法：

- a) Install-Package Log4NET
- b) 如果 NuGet 没有给自动配置 App.config，那么就要在<configuration>的<configSections>节点下 新 增 （要在头部）：

```
<section name="log4net" type="log4net.Config.Log4NetConfigurationSectionHandler,log4net" />
```
- c) 然后在<configuration>

根节点下新增：

```
<log4net>  
    <!-- OFF, FATAL, ERROR, WARN, INFO, DEBUG, ALL -->  
    <!-- Set root logger level to ERROR and its appenders -->  
    <root>  
        <level value="DEBUG" />  
        <appender-ref ref="RollingFileTracer" />  
    </root>  
    <!-- Print only messages of level DEBUG or above in the packages -->  
    <appender name="RollingFileTracer" type="log4net.Appender.RollingFileAppender,log4net">  
        <param name="File" value="App_Data/Log/" />  
        <param name="AppendToFile" value="true" />  
        <param name="RollingStyle" value="Composite" />  
        <param name="MaxSizeRollBackups" value="10" />  
        <param name="MaximumFileSize" value="1MB" />  
        <param name="DatePattern" value="\"Logs_\\"yyyy-MMdd\".txt\" />  
        <param name="StaticLogFileName" value="false" />  
        <layout type="log4net.Layout.PatternLayout,log4net">  
            <param name="ConversionPattern" value="%d [%t] %-5p %c - %m%n" />  
        </layout>
```

```
</appender>
```

```
</log4net>
```

注意：视频中讲的是`<param name="RollingStyle" value="Date" />`，应该改成`<param name="RollingStyle" value="Composite" />`

- d) App_Data 下（如果是控制台项目是生成到 bin\Debug 下）的文件无法被浏览器下载，不希望访问者下载的文件放到这里。
 - e) 在程序启动的时候：`log4net.Config.XmlConfigurator.Configure();`。容易忘，如果发现日志文件一直没有，先想是不是忘了写这句话。
 - f) 记录信息：`ILog logger = LogManager.GetLogger(typeof(WebForm1)); logger.Debug("aaaaaaaaaaaaaaa"); logger.Error("aaaaaaaaaaaaaaa");` 注意不是`LoggerManager`、不是`ILogger`
 - g) 还可以记录异常对象，这样异常堆栈就会记录到日志中：
`logger.Error("aaaaaaaaaaaaaaa",ex);`
- 5、性能优化：`logger.DebugFormat("hello {0} {}", "hello")`等。用`{n}`占位符，而不是字符串拼接，这样如果配置中不输出这个级别的时候，就不会进行字符串拼接，提升性能。
- 6、(*) 非常全的 Log4Net 使用教程：
<http://blog.csdn.net/ydm19891101/article/details/50561638> 介绍一下都可以干嘛
- 7、(*) 更适合分布式系统的日志框架：`Exceptionless`
http://www.cnblogs.com/Leo_wl/p/5863040.html

六、定时任务框架 Quartz.net

Quartz.Net 是一个从 Java 版的 Quartz 移植过来定时任务框架，可以实现异常灵活的定时任务，开发人员只要编写少量的代码就可以实现“每隔 1 小时执行”、“每天 22 点执行”、“每月 18 日的下午执行 8 次”等各种定时任务。

1、Quartz.net 基本概念

Quartz.Net 中的概念：计划者（`IScheduler`）、工作（`IJob`）、触发器（`Trigger`）。给计划者一个工作（`Job`），让他在 Trigger（什么条件下做这件事）触发的条件下执行这个工作（`Job`）

将要定时执行的任务的代码写到实现 `IJob` 接口的 `Execute` 方法中即可，时间到来的时候 `Execute` 方法会被调用。

2、用法：

1) 安装 Quartz.net，版本变化太大，前后不兼容，因此一定要和老师保持版本一致：

`Install-Package Quartz -Version 2.5.0`

2) 定义一个实现了 `IJob` 接口的类 `TestJob`，把要定时执行的代码写到 `Execute` 方法中

3) 把下面代码写到程序启动的时候，初始化一次即可。

```
IScheduler sched = new StdSchedulerFactory().GetScheduler();
JobDetailImpl jdBossReport = new JobDetailImpl("jdTest", typeof(TestJob));
IMutableTrigger triggerBossReport = CronScheduleBuilder.DailyAtHourAndMinute(23, 45).Build(); // 每天 23:45 执行一次
triggerBossReport.Key = new TriggerKey("triggerTest");
sched.ScheduleJob(jdBossReport, triggerBossReport);
sched.Start();
```

3、CronScheduleBuilder 其他定时模式：

每周固定时间：`CronScheduleBuilder.AtHourAndMinuteOnGivenDaysOfWeek(13, 55,`

DayOfWeek.Friday, DayOfWeek.Sunday) 每周五、周日的 13:55 执行;

每周固定时间: CronScheduleBuilder.WeeklyOnDayAndHourAndMinute()

每月固定时间: CronScheduleBuilder.MonthlyOnDayAndHourAndMinute()

使用 Crond 表达式设定: CronScheduleBuilder.CronSchedule("0 0 10,14,16 * * ?") Crond 表达式介绍

<http://www.cnblogs.com/junrong624/p/4239517.html> 了解 crond 强大到什么程度

4、CalendarIntervalScheduleBuilder 定时模式

```
CalendarIntervalScheduleBuilder builder = CalendarIntervalScheduleBuilder.Create();
```

```
builder.WithInterval(3, IntervalUnit.Second); //每 3 秒钟执行一次
```

还有 SimpleScheduleBuilder 等。其实不用这么多。太复杂就自己写 crond 表达式。

5、IJob 的 Execute 中异常问题: 由于 Job 是运行在单独的线程中, 因此如果 Execute 中如果发生异常, 调试的时候也是不会断点暂停的, 好像什么都没发生一样。如果运行在 ASP.Net 中, 也不会触发 ASP.net 的“未处理异常处理程序”, 就好像任务没执行一样。为了当出现异常的时候我们能及早发现, 需要把 Execute 的代码 try...catch...然后把异常处理(比如记录到日志)。

6、IJob 中怎么样 MapPath(复习): 由于 Job 是运行在单独的线程中, 是拿不到 HttpContext.Current 的, 那怎么 MapPath 呢? HostingEnvironment.MapPath(). 也不能在 Job 中做 Request、Session 等和 Web 相关的工作。

7、如果一些复杂的任务无法通过 Crond 完成怎么办? 比如“春节期间不提醒【未回答问题】”, 在 Execute 的开始检测一下是否允许执行, 不允许执行则直接 return。

8、如何执行多个任务?

写多个下面代码红圈包围的部分即可, 注意 JobDetail 和 Trigger 的名字不要重复:

```
IScheduler sched = new StdSchedulerFactory().GetScheduler();
```

```
JobDetailImpl jdBossReport = new JobDetailImpl("jdTest",
    typeof(TestJob));
CalendarIntervalScheduleBuilder builder =
    CalendarIntervalScheduleBuilder.Create();
builder.WithInterval(1, IntervalUnit.Second);
IMutableTrigger triggerBossReport = builder.Build();
triggerBossReport.Key = new TriggerKey("triggerTest");
sched.ScheduleJob(jdBossReport, triggerBossReport);
```

```
sched.Start();
```

举个例子。为了避免赋值粘贴变量造成“忘了改”的问题, 每段代码都放到单独的{}代码作用域中。

9、(*)其他优秀的定时调度框架: HangFire <http://hangfire.io/> 优点: 带任务监控界面。高级版收费。

七、IOC 容器: Autofac

基于接口编程。

1、IOC 概念

之前我们写程序的时候所有对象都是程序员手动 new 的, 当项目大了之后这样做的坏处:

- 1) 各模块之间耦合严重;
- 2) 想要更换为其他实现类的时候很麻烦;
- 3) 有的程序员只关心“给我一个实现了***接口的类”，它不想关心这个类是怎么来的。

因此就诞生了 IOC (Inversion of Control, 控制反转) 容器。使用 IOC 容器后，不再是由程序员自己 new 对象，而是由框架帮你 new 对象。

现在 IOC 有很多：Spring.Net、Unity、Castle、AutoFac 等。目前最火的就是 AutoFac。

使用 IOC 容器的时候，一般都是建议基于接口编程，也就是把方法定义到接口中，然后再编写实现类。在使用的时候声明接口类型的变量、属性，由容器负责赋值。接口、实现类一般都是定义在单独的项目中，这样减少互相的耦合。

刚接触这种思想的时候可能会感觉没用，但是随着使用深入就知道这样做的好处了。故事：Eclipse 插件体系。

2、AutoFac 基本使用

- 1) 创建接口项目和实现类项目；编写接口，编写实现类。
- 2) 安装 Install-Package Autofac
- 3) 编写一个接口 IService1，定义一个 Test()方法，定义实现类 Service1Impl
- 4) 基本用法：

```
ContainerBuilder builder = new ContainerBuilder();
builder.RegisterType<Service1>().As<IService1>(); //注册实现类 Service1，当请求 IService1 接口的时候返回 Service1 的对象。原理代码，很少写
```

```
IContainer resolver = builder.Build();
```

```
IService1 service1 = resolver.Resolve<IService1>();
```

- 5) 还可以 把 builder.RegisterType<Service1>().As<IService1>(); 写成
builder.RegisterType<Service1>().AsImplementedInterfaces(); 这样请求 Service1 实现的任何接口的时候都会返回 Service1 对象。//原理代码，很少写

如果再有一个接口 IService2、Service2Impl 那再注册一次。

如果有很多接口、很多实现类，每次这样注册都很麻烦。因此可以如下：

```
Assembly asm = Assembly.Load("实现类所在的程序集名称");
builder.RegisterAssemblyTypes(asm).AsImplementedInterfaces(); //这是最常用的用法！
```

如果有多个程序集，RegisterAssemblyTypes 还接收多个程序集。注意：如果是多个程序集，还要注意主项目要添加对相关程序集的引用。如果有多个程序集，一定注意相关的类、接口一定要是 public

- 6) 如果 Resolve() 抛异常或者返回 null，说明没有解析到实现类。如果有多个实现类，resolver.Resolve<IService1>() 只会返回其中一个类的对象。如果想返回多个实现类的对象，改成 resolver.Resolve<IEnumerable<IService1>>() 即可。
- 7) 如果一个实现类中定义了其他类型的接口属性，还可以在注册的时候加上 PropertiesAutowired，也就是 AsImplementedInterfaces().PropertiesAutowired() 还会自动给属性进行“注入”。如果可能有多个实现类，还可以声明 IEnumerable<IService1> 类型的属性。只有被 Autofac 创建出来的对象才会被“属性自动注入”

3、AutoFac 对象的生命周期

根据到底创建多少个对象，AutoFac 有如下的生命周期，在 Register*** 后面以 Instance***() 进行配置：

- 1) Per Dependency：每次请求 Resovle 都返回一个新对象： InstancePerDependency()

-
- 2) **Single Instance:** 单例，每次都返回同一个对象：SingleInstance()
 - 3) **Per Lifetime Scope:** 每个生命周期一个对象；
 - 4) InstancePerRequest: **ASP.Net MVC** 专用，每个请求一个对象。InstancePerRequest()

建议：最好配置成无状态的（实现类中不要有成员变量），使用单例方式。

4、**AutoFac+ASP.NetMVC**

AutoFac 还有 ASP.net MVC、WinForm 等的插件，用这些插件在这些平台下开发可以进一步简化开发。

MVC 下的配置：

- 1) 安装 MVC 的 AutoFac 插件：Install-Package Autofac.Mvc5 较慢

- 2) 在 Global 中调用：

```
var builder = new ContainerBuilder();
builder.RegisterControllers(typeof(MvcApplication).Assembly).PropertiesAutowired(); // 把当前
程序集中的 Controller 都注册
// 不要忘了.PropertiesAutowired()
// 获取所有相关类库的程序集
Assembly[] assemblies = new Assembly[] { Assembly.Load("ZSZ.Service") };
builder.RegisterAssemblyTypes(assemblies)
.Where(type => !type.IsAbstract)
.AsImplementedInterfaces().PropertiesAutowired();
```

```
var container = builder.Build();
```

// 注册系统级别的 DependencyResolver，这样当 MVC 框架创建 Controller 等对象的时候都是管 Autofac 要对象。

```
DependencyResolver.SetResolver(new AutofacDependencyResolver(container)); // !!!
```

可选代码：

```
builder.RegisterFilterProvider();
// 为 ActionFilter 注入
builder.RegisterAssemblyTypes(typeof(MvcApplication).Assembly)
.Where(type => typeof(ActionFilterAttribute).IsAssignableFrom(type) && !type.IsAbstract)
.PropertiesAutowired();
```

- 3) 如果想为自定义 ModelBinder 等注入，还可以使用

```
builder.RegisterModelBinders() 等方法
```

- 4) 这样在 Controller 中只要声明 Service 的属性即可，AutoFac 会自动完成属性注入。

```
public class AdminUserController : Controller
{
    public IRoleService roleService { get; set; }
    public IAdminUserService userService { get; set; }
    public ICityService cityService { get; set; }
```

在 **ActionFilter** 也可以这样用。

4) 只有 **Autofac** 帮我们创建的对象才有可能给我们自动进行属性的赋值 **PropertiesAutowired**。但是在没有注入的地方（比如 **Helper** 中），如果想获取对象，就还要 **ICityService** `cityService = DependencyResolver.Current.GetService<ICityService>()`

5) 如 果 在 **Quartz** 等 单 独 的 线 程 中 ， 无 法 通 过 `DependencyResolver.Current.GetService<ICityService>()` 获取，就要

```
var container = AutofacDependencyResolver.Current.ApplicationContainer;
using (container.BeginLifetimeScope())
{
    cityService = container.Resolve<ICityService>();
}
```

八、Json.Net(newtonjs)和 ASP.net MVC 的结合

回顾：ASP.Net MVC 默认也是使用 **JavaScriptSerializer** 做 json 序列化，不好用（DateTime 日期的格式化、循环引用、属性名开头小写等）。而 **Json.Net(newtonjs)** 很好的解决了这些问题，是.NET 中使用频率非常高的 json 库。

使用 **Json.Net** 代替最简单的方法就是使用下面的 **JsonNetResult** 来作为 **ActionResult** 返回。但是比较麻烦，还有很多其他方法。下面使用我研究的最没有“侵入性”的方法（什么是“侵入性”？引入这个技术对系统的改动量），知道原理、会照着配置即可，不用记住。

下面这种做法是体现了“一夫当关万夫莫开”的 AOP 的思想。

- 1) Install-Package newtonsoft.json
- 2) 创建一个 **JsonNetResult** 继承自 **JsonResult**（相当于自定义 **ActionResult**）

```
public class JsonNetResult : JsonResult
{
    public JsonNetResult()
    {
        Settings = new JsonSerializerSettings
        {
            ReferenceLoopHandling = ReferenceLoopHandling.Ignore, //忽略循环引用，如果设置为Error，则遇到循环引用的时候报错（建议设置为Error，这样更规范）
            DateFormatString = "yyyy-MM-dd HH:mm:ss", //日期格式化，默认的格式也不好看
            ContractResolver = new Newtonsoft.Json.Serialization.CamelCasePropertyNamesContractResolver() //json中属性开头字母小写的驼峰命名
        };
    }
}
```

```
public JsonSerializerSettings Settings { get; private set; }

public override void ExecuteResult(ControllerContext context)
{
    if (context == null)
        throw new ArgumentNullException("context");
    if (this.JsonRequestBehavior == JsonRequestBehavior.DenyGet
        && string.Equals(context.HttpContext.Request.HttpMethod, "GET",
        StringComparison.OrdinalIgnoreCase))
        throw new InvalidOperationException("JSON GET is not allowed");

    HttpResponseBase response = context.HttpContext.Response;
    response.ContentType = string.IsNullOrEmpty(this.ContentType) ? "application/json" :
    this.ContentType;

    if (this.ContentEncoding != null)
        response.ContentEncoding = this.ContentEncoding;
    if (this.Data == null)
        return;

    var scriptSerializer = JsonSerializer.Create(this.Settings);
    scriptSerializer.Serialize(response.Output, this.Data);
}
```

3) 新建一个类 JsonNetActionFilter

```
public class JsonNetActionFilter : IActionFilter
{
    public void OnActionExecuted(ActionExecutedContext filterContext)
    {
        //把 filterContext.Result从 JsonResult换成JsonNetResult
        //filterContext.Result值得就是Action执行返回的ActionResult对象
        if (filterContext.Result is JsonResult
            && !(filterContext.Result is JsonNetResult))
        {
            JsonResult jsonResult = (JsonResult)filterContext.Result;
            JsonNetResult jsonNetResult = new JsonNetResult();
            jsonNetResult.ContentEncoding = jsonResult.ContentEncoding;
            jsonNetResult.ContentType = jsonResult.ContentType;
            jsonNetResult.Data = jsonResult.Data;
            jsonNetResult.JsonRequestBehavior = jsonResult.JsonRequestBehavior;
            jsonNetResult.MaxJsonLength = jsonResult.MaxJsonLength;
            jsonNetResult.RecursionLimit = jsonResult.RecursionLimit;
        }
    }
}
```

```
        filterContext.Result = jsonNetResult;
    }
}

public void OnActionExecuting(ActionExecutingContext filterContext)
{
}
```

4) Global 中加一句: GlobalFilters.Filters.Add(new JsonNetActionFilter());

九、因为所有 ajax 请求返回的内容都包含“处理结果、数据、消息”，为了统一 ajax 的响应，我们定义一个包含 Status、Data、Msg 三个属性的类 AjaxResult，并且提供方便使用的几个构造函数。

十、WebMVCHelper

1) WebMVCHelper

```
public static string GetValidMsg(ModelStateDictionary ModelState)// 有 两 个
ModelStateDictionary 类，别弄混乱了，要用 using System.Web.Mvc; 下的
{
    StringBuilder sb = new StringBuilder();
    foreach (var ms in ModelState.Values)
    {
        foreach (var modelError in ms.Errors)
        {
            sb.AppendLine(modelError.ErrorMessage);
        }
    }
    return sb.ToString();
}
```

十一、自动截取空格和进行全角转换

有时候用户会输入全角字符（搜狗拼音输入法是那个月牙的图标: a b c d 1 2 3 4，正常的是 abcd1234，虽然看起来一样，但是他们的编码是不一样的，也就是' a ' 不等于' a'）。

用户有时候会不小心多输入空格，要自动帮用户去掉空格。

如果每个 Action 都自己处理这个太麻烦，而且容易忘，因此要提供一种符合“一夫当关万夫莫开”的 AOP 思想的无侵入式方案。

//一定要使用 using System.Web.Mvc 下的 DefaultModelBinder

编写一个自定义 ModelBinder:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
```

```
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Web.Mvc;

namespace ZSZ.CommonMVC
{
    //一定要使用using System.Web.Mvc下的DefaultModelBinder
    public class TrimToDBCModelBinder : DefaultModelBinder
    {

        public override object BindModel(ControllerContext controllerContext, ModelBindingContext
bindingContext)
        {
            object value = base.BindModel(controllerContext, bindingContext);
            if(value is string)
            {
                string strValue = (string)value;
                string value2 = ToDBC(strValue) .Trim();
                return value2;
            }
            else
            {
                return value;
            }
        }

        /// <summary> 全角转半角的函数(DBC case) </summary>
        /// <param name="input">任意字符串</param>
        /// <returns>半角字符串</returns>
        /// <remarks>
        /// 全角空格为12288，半角空格为32
        /// 其他字符半角(33-126)与全角(65281-65374)的对应关系是：均相差65248
        /// </remarks>
        private static string ToDBC(string input)
        {
            char[] c = input.ToCharArray();
            for (int i = 0; i < c.Length; i++)
            {
                if (c[i] == 12288)
                {
                    c[i] = (char)32;
                    continue;
                }
            }
        }
    }
}
```

```
if (c[i] > 65280 && c[i] < 65375)
{
    c[i] = (char)(c[i] - 65248);
}
return new string(c);
}

}
```

然后在 Global 中：

```
ModelBinders.Binders.Add(typeof(string),
    new TrimToDBCModelBinder());
```

ModelBinder 能影响简单参数类型(可以)和 FormCollection 类型 (NO) 的参数吗？

十、自定义分页组件：RuPengPager

- 1、什么是分页？为什么分页？降低数据库的压力，加载速度快
- 2、拼接生成 html，考虑问题：
 - 1)当前页码之前绘制最多 10 个页码(Math.max)；当前页面之后最多 4 个页面(Math.min)；首页和末页链接
 - 2)总页数：Math.Ceiling(totalCount / pageSize)。为什么要 Ceiling。

```
public class RuPengPager
{
    ///<summary>
    /// 每页数据条数
    ///</summary>
    public int PageSize { get; set; }

    ///<summary>
    /// 总数据条数
    ///</summary>
    public int TotalCount { get; set; }

    ///<summary>
    /// 当前页码（从 1 开始）
    ///</summary>
    public intPageIndex { get; set; }
    public string UrlPattern { get; set; } // "/Role>List?pageIndex={pn}"

    ///<summary>
    /// 最多的页码数
    ///</summary>
```

```
public int MaxPagerCount { get; set; }

public string CurrentLinkClassName { get; set; }

public RuPengPager()
{
    this.PageSize = 10;
    this.MaxPagerCount = 10;
}

public string GetPager()
{
    StringBuilder sb = new StringBuilder();

    //算出来的页数
    int pageCount = (int)Math.Ceiling(TotalCount * 1.0f / PageSize);
    int startPageIndex = Math.Max(1, pageIndex - MaxPagerCount / 2); //第一个页码
    int endPageIndex = Math.Min(pageCount, pageIndex + MaxPagerCount - 1); //最后一个页码

    sb.AppendLine("<ul>");
    for(int i=startPageIndex;i<=endPageIndex;i++)
    {
        if(i==pageIndex)
        {
            sb.AppendLine("<li class='" + CurrentLinkClassName + "'>" + i + "</li>");
        }
        else
        {
            sb.AppendLine("<li><a href='" + UrlPattern.Replace("{pn}", i.ToString()) + "'>" + i +
"</a></li>");
        }
    }
    sb.AppendLine("</ul>");
    return sb.ToString();
}
```

3、nuget 寻宝： MvcPager <https://www.nuget.org/packages/Webdiyer.MvcPager/>

十二、短信验证码

- 1、编程发送短信（SMS）有两种方法：短信猫和短信平台。
- 2、现在短信平台有很多：容联、阿里大鱼、九天企信、网易云信等。

-
- 1) 为了避免广告短信，这些短信只能使用短信模板发送验证码，不能随意定制内容。
 - 2) 费用一般是 6-8 分一条。
 - 3) 短信到达率，一般都能达到 98%以上。短信收不到的情况：信号不好；运营商之间的纠葛；手机软件拦截等。补充手段：语音验证码。
 - 4) 一天发往同一个号码的短信数量是受限的；
 - 5) 为了防止恶意攻击者（短信炸弹），短信平台开通的时候都会检查发送验证码的地方是否有图形验证码
 - 6) 接口一般是通过 http 通讯，接口各不相同，但是原理都是类似的。
- 3、如鹏短信平台模拟器
- 1) 完全模拟短信平台，当然不同厂商的接口都是不一样的。没有模拟充值的过程，注册初始送 1000 元，一条短信 8 分钱，花完为止
 - 2) 后台地址： <http://sms.rupeng.cn/> rupengtest1 123456
 - 3) 调用接口需要传递用户名和 appkey（为什么不传密码）
 - 4) 短信模板管理，模板需要审核：短信中应该用{0}指定验证码替换符；短信最后必须以 2-4 个字符并且以【】包含的品牌名结束。用程序模拟一分钟内人工审核（好幸福）；
 - 5) 由于不是真的发短信，所以只能在“短信发送记录”看日志，自动刷新
 - 6) 调用方式见《开发文档》
 - 7) 封装一个 RuPengSMS 类出来：UserName、AppKey 搞成属性，再封装一个方法：public RuPengSmsResult Send(int templatId, string code, string phoneNum)

十三、单元测试

单元测试的想法：写一个方法之前就想好这个方法有什么样的输入输出，在开发完成就测试一下给定的输出是不是产生期望的输出。

什么是自动化测试、单元测试（Unit Test）、改完了代码做回归测试？好处：自动化的运行测试案例（TestCase），避免测试的死角，保证开发的代码“步步为营”。

单元测试的核心思想：把输出对应的期望输出提前写好（我“断言（Assert）” $1+1=2$ ），每次改代码都运行一遍，如果和期望的不一致，就说明有 bug 了，要改 bug 了。程序员最喜欢“一片绿”！

单元测试工具有很多，最著名的是 XUnit 系列。Visual Studio 也内置了单元测试工具，用法都是大同小异，这里我们直接用 VS 内置的单元测试工具。

步骤：

- 1、先正常编写项目；
- 2、新建一个测试项目：【测试】→【单元测试项目】
- 3、测试项目添加对被测试项目的引用
- 4、因为测试项目相当于 UI 项目，因此数据库连接字符串配置、EF 配置等也要在测试项目中配置一遍；
- 5、对每个类的测试建一个 UnitTest 类，类上标注[TestClass]，每个 TestCase 建一个方法，方法上标注[TestMethod]，也就是：一个类测试一类，一个方法测试一个方面。
- 6、在测试方法：使用 `Assert.AreEqual(Calc.Add(1, 2), 3)` 写等于断言。如果“断言失败”，那么就会“飘红”。
- 7、在断言类上点右键【运行测试】，会运行这个类中的所有标注了[TestMethod]的方法。如果【调试测试】也可以以调试方式运行 TestCase，可以设置断点。如果飘绿就 ok，红就不 ok。
- 8、还有其他断言方法：

- a) `Assert.AreEqual`: 断言不相等
- b) `Assert.AreNotSame`: 断言不指向同一个对象; `Assert.AreSame`: 断言指向同一个对象;
- c) `Assert.Fail()`直接导致断言失败 (当没有合适的断言方法调用的时候)
- d) `IsFalse`: 断言值为 false; `.IsTrue`: 断言值为 true
- e) `IsNull`、`IsNotNull`
- f) `CollectionAssert` 类是和集合相关的断言方法

9、如何查看输出？在测试资源管理器左下角的【输出】。如何获取控制台输入？

不建议在单元测试项目中输出，都为都是给程序自动判断的。特别是自动化测试，谁去看输出。也不要依赖于控制台输入。

10、其他方法的标注：

- a) `[ClassInitialize]` 测试类每次测试类执行前执行一次
- b) `[ClassCleanup]` 测试类每次测试类执行后执行一次
- c) `[TestInitialize]` 每个测试案例方法之前都会执行一次
- d) `[TestCleanup]` 每个测试案例方法之后都会执行一次

测试案例编写的原则：可以重复执行。

11、ASP.NET MVC 的单元测试。

主要难点是对 Request 等的模拟 (Mock)，我不建议对 UI 层做过多的单元测试，因为 UI 层应该是非常轻的，而且很难模拟好。感兴趣的可以网上搜。HttpContext、HttpContextBase

十四、js 模板引擎 artTemplate

使用 js、jquery 动态生成 html 会非常麻烦。现在的模板引擎可以很简单的解决这个问题。比如腾讯出的 artTemplate

添加 artTemplate 的 js

官网：<http://aui.github.io/artTemplate/>

文档 1：http://www.cnblogs.com/Leo_wl/p/5562012.html

文档 2：<http://blog.csdn.net/jiazimo/article/details/39232425>

还有类似的框架 vue.js、angularjs 等

需求分析和数据库设计

一、项目准备

1、项目开发流程（各公司各有不同）（需求工程师，产品经理 PM）需求收集、需求文档编写、需求文档讨论及定稿；原型图编写，原型图讨论及定稿；数据库设计、架构设计、接口设计；美工设计、前端工程师切图（生成图片、CSS、html，填充假数据，并完成部分 js 的编写）；后台开发实现；测试；发版（发布、上线）；bug 修复；功能迭代（迭代周期）。

2、全部程序员都要参与，主要工作精力在“后台开发实现”

3、瀑布式开发过时→敏捷迭代式开发。

4、前端、后端；前台、后台。

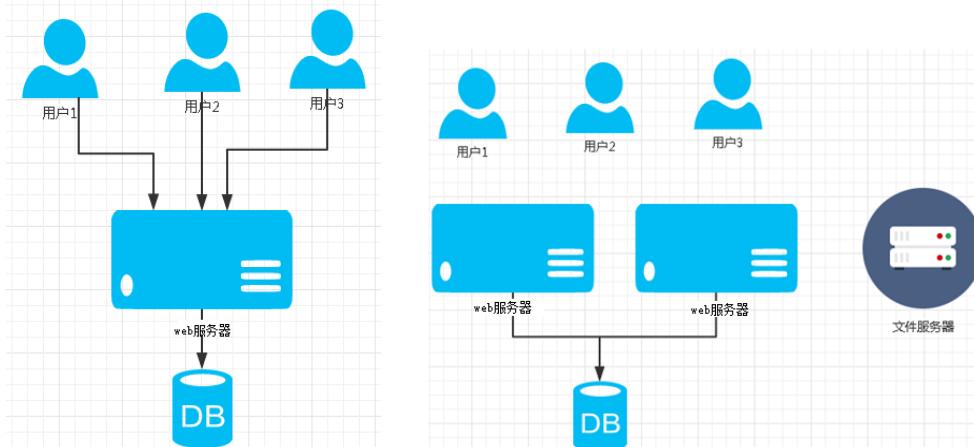
5、前台直接看原型图，然后再看需求文档；后台直接看需求文档

6、切图文件说明：前端工程师提供完成的页面，都是填的假数据，需要后端开发通过程序完成全部功能。“套模板”。模板有问题我们只要反馈即可，不过这次我们是用的从网上下

载的，有问题只能自己改。前台模板是运行在手机上的，不支持 IE 等浏览器，因此必须用 Chrome 测试。

6、第一次自己独立做项目肯定会难。

二、分布式架构（集群）



集群结构对于技术的要求是：单台服务器是随时可以抛弃、有故障的；用户的请求可以在任何一个服务器上处理。

三、数据库设计原则复习

1、一对多、多对多及表设计

2、数据库三范式：字段不能重复；必须有主键；表间通过外键引用；

四、数据的软删除

1、为什么要软删除？

1) 由于外键的存在，在删除一条数据的时候，要把所有指向他的记录都删除，这样很麻烦，而且控制不好容易引起“雪崩”。

2) 数据真正删除后容易造成问题（员工离职，那之前他发布的文章怎么办？）

3) 很多时候用户说“删除”其实只是不想看到了，如果真删了，想看就再也看不到了。

2、给表增加一个 `IsDeleted` 字段，默认值是 `false`。当“删除”的时候只是把 `IsDeleted` 更新成 `true`，再进行查询、显示的时候把 `IsDeleted=false` 的都过滤掉只显示没“删除”的（`where IsDeleted=1`）。这就叫“软删除”。

3、有什么不好？有得必有失！

五、数据库设计规范

表命名 `T_` 开头，复数结尾；主键 `Id` (`bigint` 型。程序对应 `long`)，自动增长；日期时间用 `DateTime`；表之间建外键，外键字段以 `Id` 结尾；软删除（`IsDeleted` 字段）；`CreateDateTime` 字段。

是否可空要设计好；可空字段在 `Model` 中对应包装类

六、系统表设计

所有数据表要自己建，不要管老师吃现成的。（数据不用填）项目中这些表也要开发人员设计，因此要掌握表设计的思想，作业中的表要自己设计。

表结构不是固定的，随着系统的升级而变化。

所有实体类要自己建，不要管老师吃现成的。

初始表用 `CodeFirst` 建立。

1、系统配置表(`T_Settings`):

写到代码中的坏处？配置放到数据库比放到配置文件的好处（有利于分布式部署，而且业务相关的配置便于非程序员调整参数）Value 是 nvarchar 类型，存储数字怎么办？

	列名	数据类型	允许 Null 值
▶	Id	bigint	<input type="checkbox"/>
	Name	nvarchar(250)	<input type="checkbox"/>
	Value	nvarchar(MAX)	<input type="checkbox"/>
	IsDeleted	bit	<input type="checkbox"/>
	CreateDateTime	datetime	<input type="checkbox"/>

	Id	Name	Value	IsDeleted	CreateDateTime
▶	1	名称	如鹏网	False	2017-01-20 22:...
	2	暂停时间	8	False	2017-01-20 22:...
	3	如鹏短信UserN...	rupengtest1	False	2008-08-08 00:...
	4	如鹏短信AppKey	fasdfaf2322888...	False	2008-08-08 00:...
	5	注册短信模板Id	11	False	2008-08-08 00:...

2、城市(T_Cities)、区域(T_Regions)、小区(T_Communities)

T_Cities

Id	bigint
Name	nvarchar(250)
IsDeleted	bit
CreateDateTime	datetime

T_Regions

Id	bigint
Name	nvarchar(250)
CityId	bigint
IsDeleted	bit
CreateDateTime	datetime

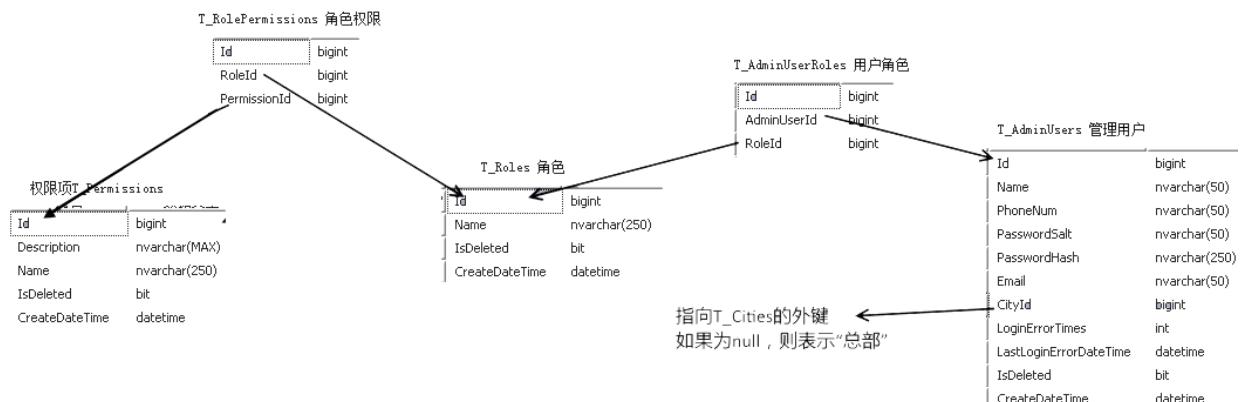
T_Communities

Id	bigint
Name	nvarchar(250)
RegionId	bigint
IsDeleted	bit
Location	nvarchar(MAX)
Traffic	nvarchar(MAX)
BuiltYear	int
CreateDateTime	datetime

3、权限表

比较难的表关系，打开数据库看一下实际数据去理解一下

RBAC: Role-Based Access Control 权限模型



4、数据字典表 T_IdNames

数据库中有很多“***类型”等数据量很小同时不经常变动的数据，比如“民族”、“房屋类型”、“性别”、“房屋状态”等，如果每个都建一张表（也没啥不可以），维护起来比较麻烦，因此有的系统中把这些只有 Id、Name 三个字段的表统一到一张“数据字典表”中。注意“地区”是无法放入“数据字典表”中的，因为它还有 CityId 字段。

通过 TypeName 属性确定这个数据属于哪种类型的数据。

表结构自己建，insert 数据给大家。

ID	TypeName	Name	IsD...	CreateDateT...
7	户型	开间	False	2017-02-06 1...
8	户型	一室一厅	False	2017-02-06 1...
9	户型	两室一厅	False	2017-02-06 1...
10	户型	三室一厅	False	2017-02-06 1...
11	户型	三室两厅	False	2017-02-06 1...
12	户型	别墅	False	2017-02-06 1...
13	户型	其他	False	2017-02-06 1...
14	房屋状态	租房中	False	2017-02-06 1...
15	房屋状态	已出租	False	2017-02-06 1...
16	房屋状态	锁定中	False	2017-02-06 1...
17	房屋类别	短租	False	2017-02-06 1...
18	房屋类别	写字楼	False	2017-02-06 1...
19	房屋类别	合租	False	2017-02-06 1...
20	房屋类别	整租	False	2017-02-06 1...
21	装修状态	精装修	False	2017-02-06 1...
22	装修状态	简装	False	2017-02-06 1...
23	装修状态	毛坯	False	2017-02-06 1...

ID	bigint
TypeName	nvarchar(50)
Name	nvarchar(250)
IsDeleted	bit
CreateDateTime	datetime

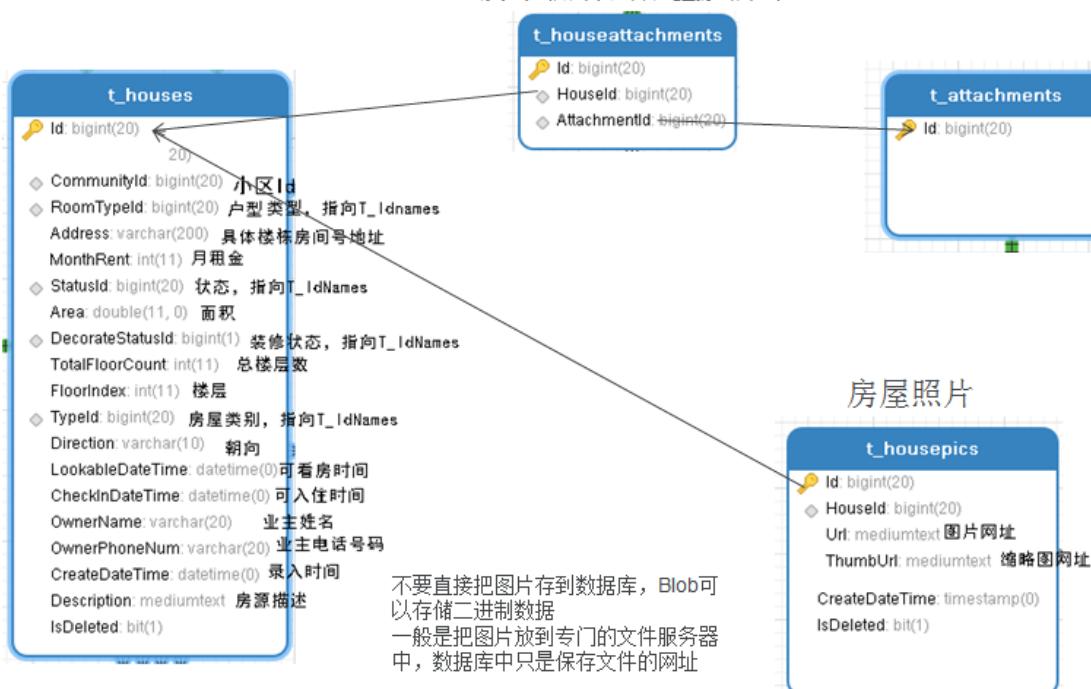
5、房屋配置表 T_Attachments

表结构自己建，insert 数据给大家

	ID	Name	IconName	IsD...	CreateDate
	1	床	icon-chuangdian	False	2008-08-08 00:00:00
	2	衣柜	icon-dpc	False	2008-08-08 00:00:00
	3	沙发	icon-shafa	False	2017-02-06 10:10:10
	4	燃气	icon-7	False	2017-02-06 10:10:10
	5	洗衣机	icon-xiyiji	False	2017-02-06 10:10:10
	6	网络	icon-wifi	False	2017-02-06 10:10:10
	7	冰箱	icon-bingxiang	False	2017-02-06 10:10:10
	8	书桌	icon-bangongzhuo	False	2017-02-06 10:10:10
	9	空调	icon-kongdiao	False	2017-02-06 10:10:10
	10	餐桌	icon-zhuozi	False	2017-02-06 10:10:10

6、房屋配置、房屋图片表

房子拥有的配置关系表



7、普通用户表 T_Users



8、房屋预约 T_HouseAppointments

Id	bigint	<input type="checkbox"/>
UserId	bigint	<input checked="" type="checkbox"/>
Name	nvarchar(50)	<input type="checkbox"/>
PhoneNum	nvarchar(50)	<input type="checkbox"/>
VisitDate	datetime	<input type="checkbox"/>
HouseId	bigint	<input type="checkbox"/>
CreateDateTime	datetime	<input type="checkbox"/>
Status	nvarchar(50)	<input type="checkbox"/>
FollowAdminUserId	bigint	<input checked="" type="checkbox"/>
FollowDateTime	datetime	<input checked="" type="checkbox"/>
IsDeleted	bit	<input type="checkbox"/>

9、管理员操作日志 T_AdminLogs

Id、**CreateDateTime**、**IsDeleted**、**UserId**、**Msg**

学生自己先写，然后再使用老师建完的。

建 ZSZ.Service 项目，安装 EF，配置 UI 项目的 app.config、UI 项目也要安装 EF，编写实体类、FluentAPI 配置和 DbContext。插入初始化数据。

项目架构搭建

- 一、 项目整体采用分层结构，不搞传统的三层架构，直接用 **UI+Service**（只是根据项目的不同的变体而已，思想上没什么不一样），建 **ZSZ.Common**、**ZSZ.Web.Common**、**ZSZ.IService**、**ZSZ.Service**、**ZSZ.DTO**、**ZSZ.Front.Web**、**ZSZ.Admin.Web**、**ZSZ.UnitTesting**
- 二、 配置 Log4Net；搭建 ExceptionFilter 打印到 Log4Net 未处理异常；把 EF 的日志通过 Debug 方式打印到 Log4Net；
开发 EF 实体部分；然后编写单元测试，确保运行没问题；后面用的时候都 Database.SetInitializer<XXXDbContext>(null);
- 三、 配置 EF；编写实体公共父类
- 四、 开发通用的 **BaseService**，提供 **GetAll()**、**MarkDeleted()**、**GetTotalCount**、**GetPagedData()**、 **GetById()** 等通用的方法。
- 五、 配置前后台项目采用进程外 Session(用数据库) **SessionId** 是保存在 Cookie 中的，其 **Expires** (过期时间) 为“空”，也就是浏览器关闭之后 **SessionId** 消失，这时候对应的服务器端的 **Session** 信息其实还会存在一段时间，直到过期，但是由于浏览器的 **SessionId** 已经消失了，所以浏览器端重启之后也找不到之前的 **Session** 了，而是一个新的 **Session**。
进程外 Session 解决的不是浏览器关闭之后取不到之前 Session 数据的问题，而是 web 服务器重启之后服务器端 Session 数据丢失的问题。测试进程外 session 一定注意浏览器不能关。
- 六、 配置 TrimAndDBModelBinder
- 七、 项目前后台都配置 AutoFac(如果连上 nuget 很费劲的话，把 nuget 的源勾选掉就行了，否则即使选择了其他源，还是会连接 nuget)
- 八、 套用后台模板，注意相对路径的问题调整；后台搞 layout 共用通用代码；因为后台模板中用到了 **ueditor**，其中有 cs 代码参与编译，需要添加“newtonsoft.json”的引用，

由于它内置的这个 Newtonsoft.json.dll 太老了，因此我们使用 nuget 安装最新的。

九、在 js 文件夹创建一个内容为 `/// <autosync enabled="false" />` 的 js 文件，文件名是 `_references.js`，然后在文件上点右键执行【更新 JavaScript 引用】。这样就会有 js 的自动提示

Service 的设计

一、说明

有的项目是先开发 **DTO**、**Service**、再开发 **UI** 层

初学者很难一次性把所有 **Service** 设计好，有可能需要随时再添加、调整，这块遇到困难是正常的。

DTO、**Service** 设计的属性、方法是根据要做的功能来设计的，设计完成后可能还会调整。
简单的 **DTO**、**Service** 老师提前写好了，学生要自己写；

学生自己先写，然后再使用老师建完的。

二、**DTO** 设计

DTO 设计的原则：扁平化；POCO（Plain Old C# Object）

设计哪些属性？取决于使用者要用哪些数据？

```
public abstract class BaseDTO
{
    public long Id { get; set; }
    public DateTime CreateDateTime { get; set; }
}

public class SettingDTO:BaseDTO
{
    public String Name { set; get; }
    public String Value { set; get; }
}

public class IdNameDTO:BaseDTO
{
    public String Name { get; set; }
    public String TypeName { get; set; }
}

public class CityDTO:BaseDTO
{
    public String Name { get; set; }
}

public class RegionDTO:BaseDTO
{}
```

```
public String Name { get; set; }
public long CityId { get; set; }
public String CityName{get;set;}
}

public class RoleDTO:BaseDTO
{
    public String Name { get; set; }
}

public class PermissionDTO:BaseDTO
{
    public String Name { get; set; }
    public String Description { get; set; }
}

public class AdminUserDTO:BaseDTO
{
    public String Name { get; set; }
    public String PhoneNum { get; set; }
    public String Email { get; set; }
    public long? CityId { get; set; }
    public String CityName { get; set; }
    public int LoginErrorTimes { get; set; }
    public DateTime? LastLoginErrorDateTime { get; set; }
}

public class AdminLogDTO: BaseDTO
{
    public long AdminUserId { get; set; }
    public String Message { get; set; }
    public String AdminUserName { get; set; }
    public String AdminUserPhoneNum { get; set; }
}

public class AttachmentDTO:BaseDTO
{
    public String Name { get; set; }
    public String IconName { get; set; }
}

public class CommunityDTO:BaseDTO
{
    public String Name { get; set; }
```

```
public long RegionId { get; set; }
public String Location { get; set; }
public String Traffic { get; set; }
public int? BuiltYear { get; set; }

}

public class HouseAppointmentDTO:BaseDTO
{
    public long? UserId { get; set; }
    public String Name { get; set; }
    public String PhoneNum { get; set; }
    public DateTime VisitDate { get; set; }
    public long HouseId { get; set; }
    public String Status { get; set; }
    public long? FollowAdminUserId { get; set; }
    public String FollowAdminUserName { get; set; }
    public DateTime? FollowDateTime { get; set; }
    public String RegionName { get; set; }
    public String CommunityName { get; set; }
}

public class HouseDTO:BaseDTO
{
    public long CityId { get; set; }
    public String CityName { get; set; }
    public long RegionId { get; set; }
    public String RegionName { get; set; }
    public long CommunityId { get; set; }
    public String CommunityName { get; set; }
    public String CommunityLocation { get; set; }
    public String CommunityTraffic { get; set; }
    public int? CommunityBuiltYear { get; set; }

    public long RoomTypeId { get; set; }
    public String RoomTypeName { get; set; }
    public String Address { get; set; }
    public int MonthRent { get; set; }
    public long StatusId { get; set; }
    public String StatusName { get; set; }
    public decimal Area { get; set; }
    public long DecorateStatusId { get; set; }
    public String DecorateStatusName { get; set; }
    public int TotalFloorCount { get; set; }
    public int FloorIndex { get; set; }
```

```
public long TypeId { get; set; }
public String TypeName { get; set; }
public String Direction { get; set; }
public DateTime LookableDateTime { get; set; }
public DateTime CheckInDateTime { get; set; }

public String OwnerName { get; set; }
public String OwnerPhoneNum { get; set; }
public String Description { get; set; }
public long[] AttachmentIds { get; set; }
public String FirstThumbUrl { get; set; }
}

public class HousePicDTO:BaseDTO
{
    public long HouseId { get; set; }
    public String Url { get; set; }
    public String ThumbUrl { get; set; }
}

public class UserDTO:BaseDTO
{
    public String PhoneNum { get; set; }
    public int LoginErrorTimes { get; set; }
    public DateTime? LastLoginErrorDateTime { get; set; }
    public long? CityId { get; set; }
}



### 三、IService 设计



不要忘了 Service 接口都要继承自 IServiceSupport



```
public interface IAdminLogService
{
 //插入一条日志: adminUserId 为操作用户 id, message 为消息
 void AddNew(long adminUserId, String message);
 //以后做: 如果做日志搜索等的话就要增加新的方法
}

public interface IAdminUserService
{
 //加入一个用户, name 用户姓名, phoneNum 手机号, password 密码, email, cityId
 //城市 id (null 表示总部)
 long AddAdminUser(String name, String phoneNum, String password, String email,
 long? cityId);
```


```

```
void UpdateAdminUser(long id, string name, string phoneNum, String password,
string email, long? cityId);

//获取 cityId 这个城市下的管理员
AdminUserDTO[] GetAll(long? cityId);

//获取所有管理员
AdminUserDTO[] GetAll();

//根据 id 获取 DTO
AdminUserDTO GetById(long id);

//根据手机号获取 DTO
AdminUserDTO GetByPhoneNum(String phoneNum);

//检查用户名密码是否正确
bool CheckLogin(String phoneNum, String password);

//软删除
void MarkDeleted(long adminUserId);

//判断 adminUserId 这个用户是否有 permissionName 这个权限项 (举个例子)
//HasPermission(3,"User.Add")
bool HasPermission(long adminUserId, String permissionName);

void RecordLoginError(long id); //记录错误登录一次
void ResetLoginError(long id); //重置登录错误信息
}

public interface IAttachmentService
{
    //获取所有的设施
    AttachmentDTO[] GetAll();

    //获取房子 houseId 有用的设施
    AttachmentDTO[] GetAttachments(long houseId);

}

public interface ICityService
{
    //根据 id 获取城市 DTO
    CityDTO GetById(long id);
```

```
//获取所有城市
CityDTO[] GetAll();

}

public interface ICommunityService
{
    //获取区域 regionId 下的所有小区
    CommunityDTO[] GetByRegionId(long regionId);
}

public interface IHHouseAppointmentService
{
    //新增一个预约:userId 用户 id(可以为 null,表示匿名用户);name 姓名、phoneNum
    手机号、houseId 房间 id、visiteDate 预约看房时间
    long AddNew(long? userId, String name, String phoneNum, long houseId, DateTime
    visitDate);

    bool Follow(long adminUserId, long houseAppointmentId) //使用乐观锁解决并发的
    问题（这里先不实现，先抛个异常，后面再做）

    //根据 id 获取预约
    HouseAppointmentDTO GetById(long id);

    //得到 cityId 这个城市中状态为 status 的预约订单数
    long GetTotalCount(long cityId, String status);

    //分页获取数据
    //limit 后面两个数不能用计算表达式，只能用固定的值，因此只能通过参数传递，
    计算在 java 中完成。
    HouseAppointmentDTO[] GetPagedData(long cityId, String status, int pageSize, int
    currentIndex);

}

public interface IHHouseService
{
    HouseDTO GetById(long id);

    //获取 typeId 这种房源类别下 cityId 这个城市中房源的总数量
    long GetTotalCount(long cityId, long typeId);

    //分页获取 typeId 这种房源类别下 cityId 这个城市中房源
    HouseDTO[] GetPagedData(long cityId, long typeId, int pageSize, int currentIndex);

    //新增房源，返回房源 id
}
```

```
long AddNew(HouseDTO house);

//更新房源，房源的附件先删除再新增
void Update(HouseDTO house);

//软删除
void MarkDeleted(long id);

//得到房源的图片
HousePicDTO[] GetPics(long houseId);

//添加房源图片
long AddNewHousePic(HousePicDTO housePic);

//软删除房源图片
void DeleteHousePic(long housePicId);

//搜索，返回值包含：总条数和 HouseDTO[] 两个属性
HouseSearchResult Search(HouseSearchOptions options);

int GetCount(long cityId, DateTime startTime, DateTime endTime);
}

//一个方法中的参数不要太多（一般不要多于 5 个），否则就封装成一个类
public class HouseSearchOptions
{
    public long CityId{get;set;}//城市 id
    public long TypeId{get;set;}//房源类型，可空
    public long? RegionId{get;set;}//区域，可空
    public int? StartMonthRent{get;set;}//起始月租，可空
    public int? EndMonthRent {get;set;}//结束月租，可空
    public OrderByType OrderByType { get; set; } = OrderByType.MonthRentAsc;//排序
    方式
    public String Keywords{get;set;}//搜索关键字，可空
    public int PageSize{get;set;}//每页数据条数
    public int CurrentIndex{get;set;}//当前页码
}

public enum OrderByType
{
    MonthRentDesc=1, MonthRentAsc=2, AreaDesc=4, AreaAsc=8, CreateDateDesc=16
}

public class HouseSearchResult
```

```
{  
    public HouseDTO[] result { get; set; } //当前页的数据  
    public long totalCount { get; set; } //搜索的结果总条数  
}  
  
public interface IIdNameService  
{  
    //类别名，名字  
    long AddNew(String typeName, String name);  
    IdNameDTO GetById(long id);  
  
    //获取类别下的 IdName (比如所有的民族)  
    IdNameDTO[] GetAll(String typeName);  
}  
  
public interface IPermissionService  
{  
    PermissionDTO GetById(long id);  
    PermissionDTO[] GetAll();  
    PermissionDTO GetByName(String name); //GetByName("User.Add")  
  
    //获取角色的权限  
    PermissionDTO[] GetByRoleId(long roleId);  
  
    //给角色 roleId 增加权限项 id permIds  
    void AddPermIds(long roleId, long[] permIds);  
  
    //更新角色 role 的权限项：先删除再添加  
    void UpdatePermIds(long roleId, long[] permIds);  
}  
  
public interface IRegionService  
{  
    RegionDTO GetById(long id);  
  
    //获取城市下的区域  
    RegionDTO[] GetAll(long cityId);  
}  
  
public interface IRoleService  
{  
    //新增角色  
    long AddNew(String roleName);  
    void Update(long roleId, String roleName);
```

```
void MarkDeleted(long roleId);
RoleDTO GetById(long id);
RoleDTO GetByName(string name);
RoleDTO[] GetAll();
//给用户 adminUserId 增加权限 roleIds
void AddRoleIds(long adminUserId, long[] roleIds);

//更新权限，先删再加
void UpdateRoleIds(long adminUserId, long[] roleIds);

//获取用户的角色
RoleDTO[] GetByAdminUserId(long adminUserId);
}

public interface ISettingService
{
    //设置配置项 name 的值为 value
    void SetValue(String name, String value); //SetValue("SmtpServer", "smtp.qq.com")

    //获取配置项 name 的值
    String GetValue(String name); //GetValue("SmtpServer")

    void SetIntValue(string name, int value); //SetIntValue("秒数", 5);

    int? GetIntValue(string name);

    void SetBoolValue(string name, bool value);

    bool? GetBoolValue(string name);

    SettingDTO[] GetAll();
}

public interface IUserService
{
    long AddNew(String phoneNum, String password);
    UserDTO GetById(long id);
    UserDTO GetByPhoneNum(String phoneNum);

    ////检查用户名密码是否正确（很好体现了分层的思想）
    bool CheckLogin(String phoneNum, String password);
    void UpdatePwd(long userId, String newPassword);

    //设置用户 userId 的城市 id
}
```

```
void SetUserCityId(long userId, long cityId);  
}
```

别忘了接口继承自 **IServiceSupport**, 否则 **AutoFac** 中获取不到对象。实现类不要忘了
public

完成实现类，并且进行充分的单元测试。

上次做完了 **AdminUserService** 的 UT。[HouseAppUnitTest](#) 没有通过
然后把通过的代码给到大家，大家基于这个版本继续向后开发。

后台权限管理模块

一、角色管理

- 1、前后台都配置通用的 Error 视图，配置通用的 return Error 视图的方法
- 2、开发权限项的 CRUD 页面：显示页面和提交动作都用同名的重载方法，通过[**HttpGet**]和 [**HttpPost**]分辨。删除操作等这些“不幂等”的操作最好不要用 **Get** 方式，因为有的浏览器会有“预取”的行为，而且也和“Http 语义”不符。通俗的说：删除请用 **Post**。
Get、Post、Put、Delete
- 3、现在主流的编程风格都不是表单提交，而是使用 **Ajax**，因为 **Ajax** 提交更灵活。
- 4、如果表单元素很多的话，自己拼 **json** 很麻烦；可以用 **jquery** 的 **serializeArray** : **var formData = \$("#form-admin-add").serializeArray()** 来简化，遵循表单提交的原则：有 **name** 的 **input/texarea/select** 的 **value** 被提交；被选中的 **checkbox、radio** 的 **name=value**。
- 5、角色的增删改查。**checked="checked" @Model.RolePerms.Select(p=>p.Id).Contains(perm.Id)"**
- 6、套模板的时候：VS 默认会自动插入关闭标记（也就是输入<div>会自动补齐</div>）给调模板带来多麻烦，因此关掉：选项的文本编辑器→HTML→高级→“自动插入关闭标记”
- 7、这套模板的表单校验用的是 **jquery** 的 **Validform** 插件（<http://validform.rjboy.cn/>），照着文档和例子去学习（工作后用到的各种框架、库很多，很多都是随用随学）。**ready** 的时候 **var validForm = \$("#form1").Validform({tiptype:2});** 初始化校验。//不同 **tiptype** 的意义

```
$("#btnSave").click(function(){//点击保存按钮  
if(validForm.check(false))//校验，返回是否校验成功  
{  
    save();  
}  
});
```

服务器端校验不能省
- 8、管理员管理：和角色管理基本一样，唯一的新的地方就是：没有选择 **CityId** 就是总部；密码要用 **MD5** 散列处理保存。（自己做，下节课讲写好的代码）作业。
- 9、后台管理员登录；**session** 记录用户 **id** 和用户城市；验证码需要下次解决。用户名：**18918911111**，密码 **123**。使用 **TempData** “一次性”的特点避免验证码的漏洞。
- 10、自定义 **IAuthorizeFilter** 实现全局权限验证。自定义一个 [**HasPermission("User.Add")**] 的 **Attribute**。这样每次访问的时候检查当前用户请求的这个 **Action** 上标注的 [**HasPermission("User.Add")**]（当然前提是必须要登录）。根据请求是否 **ajax** 返回不同格式的错误消息。
- 11、因为启用了进程外 **Session**，只要不是总重启浏览器，就不需要频繁的登录。

12、完善已有功能的权限控制。

后台房源管理

1. 房源列表：总部的人不能进行房源管理；只能显示自己所在城市的房源。
2. 表单验证。
3. 房源列表要分页。作业：给后台用户管理加上分页。删除自己做。
4. 新建房源之后再进行配图的管理；
5. 新建房源：城市取当前 session 的 cityId，选择区域，通过 ajax 加载小区
6. 编辑房源：\$("#regionId").change();//页面加载的时候触发一次，让他加载小区
7. 日期编辑器：直接使用 onfocus="WdatePicker({dateFmt:'yyyy-MM-dd HH:mm:ss',maxDate:'#F{\$dp.\$D(\'datemax\')}||\'%y-%M-%d\'}')" 会报错 因为经过猜测，maxDate:....代表设置最大日期为 datemax 这个控件的值。因此不设置 maxDate 即可。onfocus="WdatePicker({dateFmt:'yyyy-MM-dd HH:mm:ss'})" 需要选择时间，因此选择日期后，对话框不会关闭，需要点【确定按钮】。而 onfocus="WdatePicker({ dateFmt: 'yyyy-MM-dd' })" 则会点击日期后自动关闭
8. 完善后台界面，登录流程，登出功能。

房源图片管理

- 1、再次强调：不要把图片内容直接存到数据库中，只能存图片在服务器上的路径；
- 2、配图管理：使用后台模板中的“图片管理”的模板来做，把“图片名称”链接的界面和“添加图片”中下面的批量上传结合到一起。这个模板使用的百度的 WebUploader 来做上传，可以实现批量上传，可以显示上传进度、异步上传，比<input type="file"> 好用。批量上传对于服务器仍然是收到的多个上传请求，而不是一次上传请求。
- 3、lightbox-plus-jquery.min.js 和其他插件有冲突，不好调整，因此不用lightbox-plus-jquery.min.js 的引用，改成图片链接的 target="_blank"
- 4、例子代码中重复的创建了两次 uploader，把第一个 ready 中的所有代码删掉就可以。
- 5、上传地址配置在 WebUploader.create 的 WebUploader.create 中的 server 属性中
- 6、如何响应服务器端上传处理程序的响应，服务器端可以返回 json，然后在代码中添加：

```
uploader.on( 'uploadSuccess', function( file,response ) {  
    if(response.status!="ok")  
    {  
        alert(response.msg);  
        showError( response.msg );  
    }  
});
```

7、webuploader 文件是一个个上传的，表单名是 file

8、过滤非法的后缀名

9、暂时先上传到主站的 upload 目录下，后期改造成云存储。

文件上传到“年/月/日/md5.后缀”，如果使用用户选择图片的文件名会有重名的问题，用 md5 的好处（同一天上传同一个文件不会重复）。

经过 md5 运算后，指针在末尾，再写入就没的可写了，因此用 stream.Position=0 指针归零。

10、生成缩略图、大图加上水印；

上传的图片信息要插入到数据库中

配图的展示和配图的删除

分页!!!!

```
insert into
t_houses (CommunityId, RoomTypeId, Address, MonthRent, StatusId, Area, Decor
ateStatusId, TotalFloorCount, FloorIndex, TypeId, Direction, LookableDateT
ime, CheckInDateTime, OwnerName, OwnerPhoneNum, Description, CreateDateTim
e, IsDeleted)
select
CommunityId, RoomTypeId, Address, MonthRent, StatusId, Area, DecorateStatus
Id, TotalFloorCount, FloorIndex, TypeId, Direction, LookableDateTime, Check
InDateTime, OwnerName, OwnerPhoneNum, Description, CreateDateTime, IsDelet
ed from t_houses
```

前台基本框架

- 1、建前台项目、搭建必要的文件、套模板（模板不支持 ie，用 chrome 没问题；最好缩小到手机大小，根据屏幕大小动态调整布局的技术叫“响应式布局”（Media Query），自适应，是前端技术。bootstrap）
- 2、前端使用的 MUI（<http://dev.dcloud.net.cn/mui/>）这个前端框架
- 3、提取出通用的部分为 layout（参考 sign.html，不要引入 index.html 中的 css/swiper.min.css 等，否则会导致 body 正文部分上错的问题）
- 4、登录页面中<i class="iconfont icon-lock fl"></i>，那还有那些 icon-可以用呢？用 F12 工具查看 icon-lock 定义在哪里，就可以找到其他的了
- 5、script、img、a 等的 src 中写"~/"会自动进行虚拟路径处理
- 6、Web 项目也要在 web.config 中配置 ef，也要添加对 ef 的引用
- 7、注册页面：需要

```
<script type="text/javascript" src="~/js/jquery.leanModal.min.js"></script>
<script type="text/javascript" src="~/js/tchuang.js"></script>
```

并且要放到 jquery 之后
不要使用原始的 tchuang.js、jquery.leanModal.min.js，要改里面的代码，这样才能在发送验证码之后才弹出窗口以及在对话框中验证通过再关闭对话框。
把 tchuang.js、jquery.leanModal.min.js 改成注册页面独有的
- 8、前台的页面中不要写 js，要写到单独的 js 文件中(好处是?)，js 文件要使用 utf8 编码，否则显示会有乱码
- 9、密码为什么“加盐”算散列值更安全：不加盐用户的简单密码很容易被反推出来，固定的盐会更安全一些，加随机盐更安全。
- 10、注册登录：短信验证码调用之前封装好的 RuPengSMSender、appkey、短信模板 id 等存到 T_Settings 中；
- 11、注册发送验证码要先验证图形验证码；注册时候检查短信验证码正确性。用 TempData 保存。
- 12、登录、找回密码；登录的时候把用户 id、城市 id 保存到 session 中

前台首页

- 1、首页城市列表，页面初始显示当前城市名（默认是第一个城市）
- 2、获取当前城市 id：如果用户没登录，则取 Session 中的城市 id；如果用户登录了，则取用户的 CityId，如果当前用户没有 CityId，则认为第一个城市是；
- 3、点击城市列表动态切换显示城市列表

```
$("#btnCities").click(function(){  
    var left = $(this).offset().left;  
    var top = $(this).offset().top;  
    var height = $(this).height();  
    $("#popupCities").toggle("fast").css("top",top+height).css("left",left);  
});
```

- 4、监听城市点击\$(“#ulCities li”)，更新当前用户的 CityId(没有登录则设置 Session)

房源搜索

- 1、房源展示页面（我要看房暂时不实现）
- 2、先编写 HouseService 中的 search 方法，先使用 EF 来进行查询。
- 3、搜索链接的生成。搜索条件要根据用户的选择动态增减 queryString 的选项：

```
public static string ToQueryString(this NameValueCollection queryString)  
{  
    StringBuilder sb = new StringBuilder();  
    for (int i = 0; i < queryString.Keys.Count; i++)  
    {  
        string key = queryString.Keys[i];  
        sb.Append(key).Append("=").Append(Uri.EscapeDataString(queryString[key]));  
        if (i != queryString.Keys.Count - 1)  
        {  
            sb.Append("&");  
        }  
    }  
    return sb.ToString();  
}  
  
/// 修改 QueryString 中 name 的值为 value，如果不存在，则添加  
// NameValueCollection 相当于 Dictionary，存放的是 QueryString 中的键值对  
public static string UpdateQueryString(NameValueCollection queryString, string name, object  
value)  
{  
    //拷贝一份，不影响本来的 QueryString  
    NameValueCollection newNVC = new NameValueCollection(queryString);  
    if (newNVC.AllKeys.Contains(name))  
    {  
        newNVC[name] = Convert.ToString(value);  
    }
```

```
        }
        else
        {
            newNVC.Add(name, Convert.ToString(value));
        }
        return newNVC.ToQueryString();
    }

/// 从 QueryString 中 移 除 name 的 值 public static string
RemoveQueryString(NameValueCollection queryString, string name)
{
    NameValueCollection newNVC = new NameValueCollection(queryString);
    newNVC.Remove(name);
    return newNVC.ToQueryString();
}

使用:
<li><a href='/House/Search?@WebMVCHelper.RemoveQueryString(Request.QueryString,"MonthRent")'>全部</a></li>
<li><a href='/House/Search?@WebMVCHelper.UpdateQueryString(Request.QueryString,"MonthRent","500-1000")'>500-1000</a></li>
<li><a href='/House/Search?@WebMVCHelper.UpdateQueryString(Request.QueryString,"MonthRent","1001-2000")'>1001-2000</a></li>
<li><a href='/House/Search?@WebMVCHelper.UpdateQueryString(Request.QueryString,"MonthRent","2001-3000")'>2001-3000</a></li>
<li><a href='/House/Search?@WebMVCHelper.UpdateQueryString(Request.QueryString,"MonthRent","4001-5000")'>4001-4000</a></li>
<li><a href='/House/Search?@WebMVCHelper.UpdateQueryString(Request.QueryString,"MonthRent","5000-*")'>5000 以上</a></li>
```

房源搜索页面要: <script type="text/javascript" src="~/js/menu.js"></script>

- 4、房屋搜索页面上的链接点不了, F12 一看原来是遮罩了: <div id="oe_overlay" class="oe_overlay"></div>删掉即可
- 5、前台页面是给手机屏幕用的, 不适合使用分页组件, 因为手机操作是上下拉刷新和加载
- 6、参考 MUI 文档的“上拉刷新”这一页。如果出现 Cannot read property 'addEventListener' of undefined, 说明是 container 高度太窄了。container 为待刷新区域标识, querySelector 能定位的 css 选择器均可。在 callback 中 var prThis = this; 然后 prThis.endPullupToRefresh(true); // 数据加载完成。true 表示“没有更多数据”了

```
prThis.endPullupToRefresh(false); //数据加载完成。false 表示“还有更多数据”
```

7、使用 js 模板引擎加载搜索结果和分页加载。

预约和抢单搜索

1、数据库的并发问题

高并发（很多人在操作这个系统。某个操作分为 ABC 三步，在 1 这个用户刚执行到 B 的之后，2 这个用户也开始执行 A。）的系统中会遇到数据并发修改的问题，如下几个场景：

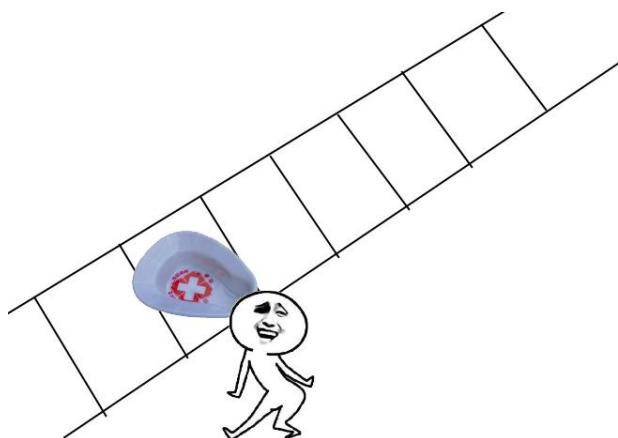
- 1) 抢票：假设有 100 张票 T_Tickets(Id,Name,Amount)，有一行数据 【1,"D168",100】
如果执行如下的抢票操作：首先执行 select Amount from T_Tickets where Name=' D168' 如果查到还有票的话，再 Update T_Tickets Set Amount=Amount-1 Where Name=' D168' 在高并发的场景下会出现“超卖”的情况。
- 2) 打车软件中司机抢一个订单：T_Orders(Id,Customer,DriverId)，有一行数据 【1,"张三",null】
如果执行如下的抢单操作：首先查询一下这个订单是否有被人抢了，select DriverId from where Id=1，如果没有被人抢，则 Update T_Orders set DriverId=@did where Id=1。在高并发场景下就会出现两个人都提示抢单成功，但是稍早抢到单的最后反而发现抢单者不是自己。

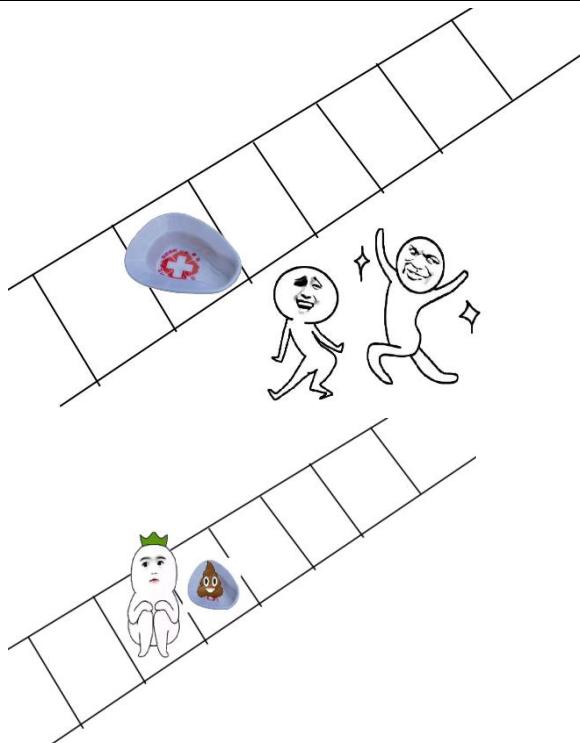
2、数据库的并发控制

悲观锁：很悲观，每次去拿数据的时候都认为别人会修改，所以每次在拿数据的时候都会“上锁”，操作完成之后再“解锁”。在数据加锁期间，其他人如果来拿数据就会等待，直到去掉锁。抢厕所。数据库中的悲观锁有“表锁”、“行锁”等。



乐观锁：很乐观，认为不会占用，自己先占了再说，占完了之后再看看是不是占上了，如果没占上就是失败了。





两种锁各有优缺点。悲观锁使用的体验更好，但是对系统性能的影响大，只适合并发量不大的场合。乐观锁适用于“写少读多”的情况下，加大了系统的整个吞吐量，但是“乐观锁可能失败”给用户的体验不好。

举例：建立表 T_Girls(Id,Name,BF)，争抢成为男朋友。

2、悲观锁的使用

EF 不支持悲观锁，只能写原生 SQL。

一定要在同一个事务中。在查询语句的表名后加上 with (xlock,ROWLOCK)。xlock 表示“排他锁”，一旦加上排他锁，那么其他人再获得这个锁的话就要等待开锁（事务结束）。

ROWLOCK 为行锁，为锁定查询出来的行。

```
Console.WriteLine("输入你的名字");
string bf=Console.ReadLine();
using (MyDbContext ctx = new MyDbContext())
using (var tx = ctx.Database.BeginTransaction())
{
    Console.WriteLine("开始查询");
    //一定要遍历一下 SqlQuery 的返回值才会真正执行 SQL
    var g = ctx.Database.SqlQuery<Girl>("select * from T_Girls with (xlock,ROWLOCK) where id=1 ").Single();
    Console.WriteLine("结束查询");
    Console.WriteLine("开始 Update");
    ctx.Database.ExecuteSqlCommand("Update T_Girls set BF={0} where id=1 ", bf);
    Console.WriteLine("结束 Update");
    Console.WriteLine("按任意键结束事务");
    Console.ReadKey();
```

```
try
{
    tx.Commit();
}
catch(Exception ex)
{
    Console.WriteLine(ex);//别忘了打印异常，否则出错了都不知道
    tx.Rollback();
}
}
```

然后直接运行两次生成的 exe，会发现第一个事务结束后第二个才会执行查询。
等价的纯 ADO.Net 的写法

```
string connstr = ConfigurationManager.ConnectionStrings["connstr"].ConnectionString;
using (SqlConnection conn = new SqlConnection(connstr))
{
    conn.Open();
    using (var tx = conn.BeginTransaction())
    {
        try
        {
            Console.WriteLine("开始查询");
            using (var selectCmd = conn.CreateCommand())
            {
                selectCmd.Transaction = tx;
                selectCmd.CommandText = "select * from T_Girls with(xlock,ROWLOCK) where id=1";
                using (var reader = selectCmd.ExecuteReader())
                {
                    if(!reader.Read())
                    {
                        Console.WriteLine("没有id为1的女孩");
                        return;
                    }
                    string bf = null;
                    if(!reader.IsDBNull(reader.GetOrdinal("BF")))
                    {
                        bf = reader.GetString(reader.GetOrdinal("BF"));
                    }
                    if(!string.IsNullOrEmpty(bf))//已经有男朋友
                    {
                        if(bf==myname)
                        {
                            Console.WriteLine("早已经是我的人了");
                        }
                    }
                }
            }
        }
    }
}
```

```
{  
    Console.WriteLine("早已经被" + bf + "抢走了");  
}  
Console.ReadKey();  
return;  
}  
//如果 bf==null, 则继续向下抢  
}  
Console.WriteLine("查询完成, 开始 update");  
using (var updateCmd = conn.CreateCommand())  
{  
    updateCmd.Transaction = tx;  
    updateCmd.CommandText = "Update T_Girls set BF='aaa' where id=1";  
    updateCmd.ExecuteNonQuery();  
}  
Console.WriteLine("结束 Update");  
Console.WriteLine("按任意键结束事务");  
Console.ReadKey();  
}  
tx.Commit();  
}  
catch (Exception ex)  
{  
    Console.WriteLine(ex);  
    tx.Rollback();  
}  
}  
}  
}
```

(*)锁分为两个维度：锁的等级（共享锁、排他锁等）和锁的范围（行锁、表锁）。主流数据库都支持这些锁，不过语法有差别。

(*)死锁：事务 1 依次锁定 A、B；事务 2 依次锁定 B、A。他们两个如果碰到一起……

(*)参考资料：<http://www.cnblogs.com/knowledgesea/p/3714417.html>

3、乐观锁的使用

数据库中有一个特殊的字段类型 `rowversion`，列名叫什么无所谓，这个字段的值不需要程序员去维护，每次修改这行数据的时候，对应的 `rowversion` 都会自动的变化(一般是增加)。

有的资料中可能提到的是 `timestamp` 类型，在其他数据库中也支持 `timestamp`。`SQLServer` 中也支持 `timestamp`，但是微软更推荐 `SQLServer` 中使用 `rowversion` 类型。

乐观锁“抢女友”的思路很简单：抢之前查一下女友的 `rowversion` 的值，假如查出来是 666。那么就执行 `update T_Girls set BF='yzk' where id=1 and RowVersion=666`。

执行之后如果发现“执行受影响的行数是 0”，就说明在这之前已经有人“抢先了”，因此“抢单失败”。

给表增加一个 `rowversion` 类型的字段，比如名字也叫 `rowversion`。实体类中对应的属性

类型要写成 byte[]。然后 FluentAPI 中配置： Property(e => e.RowVersion).IsRowVersion();

原生 SQL 的使用方法：

```
using (MyDbContext ctx = new MyDbContext())
{
    string bf = Console.ReadLine();
    var g = ctx.Database.SqlQuery<Girl>("select * from T_Girls where id=1").Single();
    if (g.BF != null)
    {
        if (g.BF == bf)
        {
            Console.WriteLine("早已经是你的人了呀，还抢啥？");
            Console.ReadKey();
            return;
        }
        else
        {
            Console.WriteLine("来晚了，早就被别人抢走了");
            Console.ReadKey();
            return;
        }
    }
    Console.WriteLine("点击任意键，开抢（模拟耗时等待并发）");
    Console.ReadKey();
    Thread.Sleep(3000);
    int affectRow = ctx.Database.ExecuteSqlCommand("update T_Girls set BF={0} where id=1
and RowVersion={1}",
        bf, g.RowVersion);
    if (affectRow == 0)
    {
        Console.WriteLine("抢媳妇失败");
    }
    else if (affectRow == 1)
    {
        Console.WriteLine("抢媳妇成功");
    }
    else
    {
        Console.WriteLine("什么鬼");
    }
}
```

纯 ADO.Net 写法的思路……

EF 的用法：

```
string bf = Console.ReadLine();
```

```
using (MyDbContext ctx = new MyDbContext())
{
    ctx.Database.Log = (sql) =>
    {
        Console.WriteLine(sql);
    };
    var g = ctx.Girls.First();
    if (g.BF != null)
    {
        if (g.BF == bf)
        {
            Console.WriteLine("早已经是你的人了呀，还抢啥？");
            Console.ReadKey();
            return;
        }
        else
        {
            Console.WriteLine("来晚了，早就被别人抢走了");
            Console.ReadKey();
            return;
        }
    }
    Console.WriteLine("点击任意键，开抢（模拟耗时等待并发）");
    Console.ReadKey();
    g.BF = bf;
    try
    {
        ctx.SaveChanges();
        Console.WriteLine("抢媳妇成功");
    }
    catch (DbUpdateConcurrencyException)
    {
        Console.WriteLine("抢媳妇失败");
    }
}
Console.ReadKey();
```

EF 会在 SaveChanges() 之后去检查 RowVersion 是否有变化，如果发现和之前查的时候不一致就会抛 DbUpdateConcurrencyException 异常。

(*)参考资料：<http://www.cnblogs.com/Gyoung/archive/2013/01/18/2866649.html>

总结：悲观锁容易引起死锁，而且性能低，大部分系统的特点都是“读多、写少”，因此除非特殊情况，否则都推荐使用乐观锁，当然如果是使用也推荐更傻瓜化的捕捉 DbUpdateConcurrencyException 这种“傻瓜化”的方案。

4、预约对话框的打开。预约界面的日期选择使用 mui.picker，比后台用的日期选择器更适

合手机使用习惯，需要添加对 mui.picker.js、mui.picker.css 添加，参考 <http://dev.dcloud.net.cn/mui/> 官网的代码，需要注意官网例子中的\$不是 jquery 的\$。

- 5、通过 ajax 提交预约
- 6、后台员工抢单：使用乐观锁方案。
- 7、定时发送运营报表给老板
 - 1) 使用 Quartz.NET 做定时；
 - 2) 先编写报表的代码：每个城市昨天的新增房源数量。并且使用 Smtp 邮件发送；

所见即所得编辑器

- 1、展示什么是所见即所得编辑器？通过 F12 查看所见即所得编辑器的本质。
- 2、所见即所得编辑器有很多，常用的有 CKEditor、KindEditor、UEditor、TinyMCE 等，用法大同小异，这里使用百度的 UEditor。
- 3、后台的模板已经内置了 UEditor 的插件(lib 下)，也可以自己去 UEditor 官网下载最新版。用法看官方文档。初始内容放到<script>标签内，提交给服务器的时候服务器会报错“从客户端中检测到有潜在危险的 Request.Form 值”，因为当提交的请求中有 html 标签等数据的时候默认是禁止访问的，可以在 Action 上加上[ValidateInput(false)]禁止这个检测。
- 4、房源管理中使用 UEditor 替代“房源描述”的 textarea。渲染的时候要用@Html.Raw
- 5、一个页面中如何放多个 UEditor 实例；浏览 UEditor 文档，了解他能做什么。
- 6、(*)当内容有太复杂内容的时候，有可能导致 HTML 错乱而显示错乱。那么就 ajax 加载，然后 setContent。
- 7、什么是 XSS (Cross Site Script，跨站脚本) 漏洞？直接切换到 UEditor 的 html 模式，可以直接输入 alert 语句，也可以直接输入 form 标签，这样就给人骗人的机会。因此 asp.net mvc 默认禁止提交内容中带 html 标签。
- 8、不能通过客户端 js 检查来避免 XSS 漏洞，因为用户可以直接提交给服务器。要么让 HTML 默认替我们在服务器端检测拦截，或者不用 Raw 显示到界面上。
- 9、原则：后台管理员等这些可信的使用者使用的功能允许[ValidateInput(false)]，普通用户提交的内容在显示的时候不要@Html.Raw。如果是普通用户提交又想插入链接、图片怎么办？→ MarkDown 编辑器，显示之前再把格式做转换成 html。
- 10、 图片的上传和批量上传：ueditor.config.js 中修改 serverUrl 中的路径，指向 net/controller.ashx，插入图片后发现图片上传到了 ueditor\1.4.3\net\upload 下，但是网页中怎么显示不对呢？根据 html 猜测应该修改 net 下的 config.json 的*** UrlPrefix 成为***。

网站优化

- 1、云存储
 - a) 文件放到主站的缺点？如鹏公开课聊天室的事故及优化；
 - b) 使用云存储服务器的缺点？什么是 CDN？优点：便宜；不用自己维护、不用自己调优；支持 CDN 加速。
 - c) 阿里云、七牛云、又拍云、新浪云、百度云、AWS、Azure 等
 - d) 用七牛云举例，七牛变了怎么办。七牛云的申请，我申请好了（yzk365@qq.com 、123456），对象存储→添加存储资源（Bucket）在“个人面板”→“密钥管理”设置 AccessKey 和 SecretKey。
 - e) 七牛云 SDK 的调通。世上的接口都是坑，腾讯都有坑。有时候文档说得也是错的。

- f) 七牛云替换 UEditor 中默认的文件上传。研究得知处理上传的是 UploadHandler，研究最终搞定上传。解决问题的过程比结论重要，“以不变应万变”。

缓存优化

- 1、缓存
 - a) 复习：为什么用缓存？asp.net 内置缓存怎么用？
 - b) 使用内置缓存优化房源的搜索和房源的展示页面。优化获取配置的代码；
 - c) 什么时候不能用缓存“需要数据及时显示出来的”
- 2、ASP.Net 内置的 Cache 是“内存 Cache”，是放到 Web 服务器中的，在多台 Web 服务器组成的集群中，无法实现共享。而且如果数据量大会占用 Web 服务器内存。
- 3、现在流行的实现“进程外 Cache”的服务器主要有 Redis 和 Memcached。他们都是 NoSQL（Not Only SQL）数据库，不是替代 SQL 数据库的，而是弥补 SQL 数据库缺点的。NOSQL 能做的 SQL 数据库也能做，不过 NoSQL 这些做的效率更高。Redis 可以把数据持久化到硬盘中，而 Memcached 是放到内存中，重启后就消失。如果仅仅是做 KV 键值对缓存使用，用 Memcached 就可以。
- 4、NOSQL 数据库也是单独运行的服务器，我们的程序通过 SDK（类似于 ADO.Net 之于 SQLServer）连接服务器。
- 5、Memcached
 - a) Memcached 服务器的运行：Memcached 推荐安装在 Linux 下，Windows 下仅供测试；安装包：<http://www.runoob.com/memcached/window-install-memcached.html>（因为仅是测试，因此不用安装成服务，直接双击运行 memcached.exe 即可。**不要关**）
 - b) 安装 .Net 的 Memcached 客户端 API：EnyimMemcached（还有其他的 API）：**Install-Package EnyimMemcached**
 - c) Redis 默认是使用二进制序列化，因此对象需要标记为可序列化[Serializable]
 - d) 测试代码：

```
MemcachedClientConfiguration config = new MemcachedClientConfiguration();
config.Servers.Add(new IPEndPoint(IPAddress.Loopback, 11211));
config.Protocol = MemcachedProtocol.Binary;
MemcachedClient client = new MemcachedClient(config);
var p = new Person { Id = 3, Name = "yzk" };
client.Store(Enyim.Caching.Memcached.StoreMode.Set, "p", p); //还可以指定第四个参数指定数据的过期时间。
Person p1 = client.Get<Person>("p");
```
 - e) 为了利用连接池，不要 using MemcachedClient，而是使用一个 static 对象。
- 6、Redis
 - a) Redis 服务器的运行，Redis 推荐安装在 Linux 下，Windows 下仅供测试；安装包见旁边资料（因为仅是测试，因此不用安装成服务，直接双击运行 redis-server.exe 即可，**不要关**）
 - b) 安装 .Net Redis 开发包：Install-Package ServiceStack.Redis 文档：<https://github.com/ServiceStack/ServiceStack.Redis>
 - c) Redis 是使用 Json 序列化，不需要标记为可序列化
 - d) 测试代码

```
PooledRedisClientManager redisMgr = new PooledRedisClientManager("127.0.0.1");
using (IRedisClient redisClient = redisMgr.GetClient())
{
    var p = new Person { Id = 3, Name = "yzk" };
    redisClient.Set("p", p);
    var p1 = redisClient.Get<Person>("p");
    return Content(p1.Name);
}
```

e) 也可以通过 Set 的最后一个参数设定过期时间

7、哪些地方需要缓存优化（经常读取、很少写入）。

2、页面静态化

- a) 即使使用缓存，只是降低数据库服务器压力，web 服务器仍然是“每次来访都要跑一遍代码”，如果所有人访问的结果都一样，就可以直接将响应的内容保存成 html 文件，让用户访问 html 文件。
- b) 之前都是用户请求 Action，获取 html 响应去显示，那么怎么样通过程序去请求 Action 获取响应呢？
- c) 首先定义如下的方法：

```
static string RenderViewToString(ControllerContext context,
                                  string viewPath,
                                  object model = null)
{
    ViewEngineResult viewEngineResult =
        ViewEngines.Engines.FindView(context, viewPath, null);
    if (viewEngineResult == null)
        throw new FileNotFoundException("View" + viewPath + "cannot be found.");
    var view = viewEngineResult.View;
    context.Controller.ViewData.Model = model;
    using (var sw = new StringWriter())
    {
        var ctx = new ViewContext(context, view,
                                 context.Controller.ViewData,
                                 context.Controller.TempData,
                                 sw);
        view.Render(ctx, sw);
        return sw.ToString();
    }
}
```

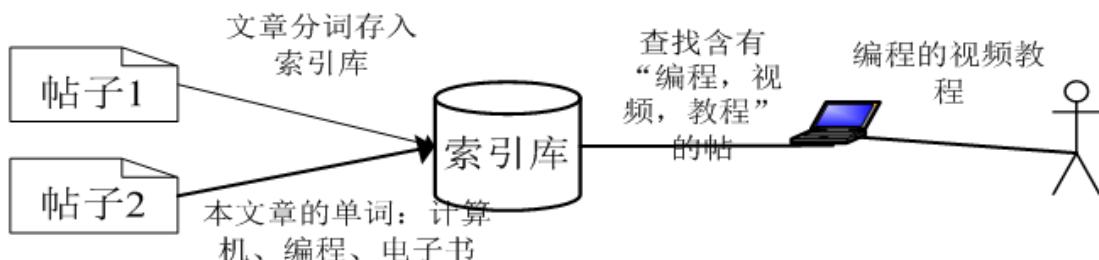
d) 然后如下调用：`string html = RenderViewToString(this.ControllerContext, "~/Views/Default/Test.cshtml", person);`

e) 把房源展示做成静态化页面，对应列表页面和搜索中指向的超链接也改成指向 html 的。也就是普通用户再也不知道还可以通过/House/Index/3 来访问。

- f) 缓存和静态页的区别：静态页的性能比缓存好，能用静态页就用静态页。什么情况下不能用静态页：相同的地址不同的人看的不一样（<http://www.rupeng.com/User/MyInfo>）、有的页面有的人不能看（如鹏的学习卡页面）。
- g) 做一个“全部重新生成”的按钮；

全文检索引擎

- 1、现在主流网站、App 都有“站内搜索”的功能：优酷、爱奇艺、链家、携程……。如果用 SQL 做站内搜索怎么做，有什么问题？全表扫描。
- 2、什么是全文检索，原理是什么？分词。



- 3、Lucene 是 Java 编写的全文检索引擎底层开发包，也有.net 移植版本 Lucene.Net。基于 Lucene 开发难度比较大。因此有人开发出了基于 Lucene 的搜索引擎服务器，主要有 Solr、Elastic Search 等。Solr 的使用可以参考如鹏的公开课：<http://www.rupeng.com/Courses/Chapter/523>。这次我们就讲讲 Elastic Search 的使用。Solr、Elastic Search(简称 ES)都是使用 Java 开发的，因此运行的时候要配置 Java 的运行环境。这些搜索引擎服务器都是“客户端 SDK+服务器”的这种使用方式。

4、elastic search 的安装

- 1) 下载安装 Java 运行环境 JDK1.8；解压 elasticsearch-5.2.0.zip (参考资料区域)；环境变量中配置“JAVA_HOME”指向 JDK 的目录。
- 2) 打开 cmd, cd 到 bin 目录。运行 elasticsearch.bat 如果报错“命令语法不正确”说明 JAVA_HOME 没配置好，然后好之后一定要重启 cmd。
- 3) 如果 elasticsearch 运行报错： Error occurred during initialization of VM Could not reserve enough space for 2097152KB object heap 。那么说明是内存不足，就修改 config/jvm.options 下的
-Xms2g
-Xmx2g
改成
-Xms512m
-Xmx512m
- 4) 浏览器访问 <http://localhost:9200/>，如果不报错就说明成功了
- 5) 运行成功后不要关闭 cmd。

5、SDK 有 ElasticSearch.Net、NEST、PlainElastic.Net 等都可以使用。这里使用 PlainElastic.Net。 安装 .Net SDK: Install-Package PlainElastic.Net

6、插入数据：

存的数据需要是 Json 格式的，搜索的时候会根据属性来搜索

```
Person p1 = new Person();
p1.Id = 2;
p1.Age = 8;
p1.Name = "丑娘娘";
p1.Desc = "二丑家的姑娘，是最美丽的女孩";

ElasticConnection client = new ElasticConnection("localhost", 9200);
var serializer = new JsonNetSerializer();
//第一个参数相当于“数据库”，第二个参数相当于“表”，第三个参数相当于“主键”
IndexCommand cmd = new IndexCommand("zsz","persons",p1.Id.ToString());
//Put()第二个参数是要插入的数据
OperationResult result = client.Put(cmd, serializer.Serialize(p1));
var indexResult = serializer.ToIndexResult(result.Result);
if(indexResult.created)
{
    Console.WriteLine("创建了");
}
else
{
    Console.WriteLine("没创建"+indexResult.error);
}
```

如何知道插入成功？看 `indexResult` 返回值。如果 `id` 已经存在，则不再插入，如果想覆盖 `update` 的话，就要先删再插。

7、查询

```
ElasticConnection client = new ElasticConnection("localhost", 9200);
SearchCommand cmd = new SearchCommand("persons", "1");
var query = new QueryBuilder<Person>()
.Query(b =>
    b.Bool(m =>
        //并且关系
        m.Must(t =>
            //分词的最小单位或关系查询
            t.QueryString(t1 => t1.DefaultField("Name").Query("培训"))
        )
    )
)
//分页
.From(0)
.Size(10)
//排序
// .Sort(c => c.Field("age", SortDirection.desc))
//添加高亮
.Highlight(h => h
.PreTags("<b>")
```

```
.PostTags("</b>")  
.Fields(  
    f => f.FieldName("Name").Order(HighlightOrder.score)  
)  
)  
.Build();  
var result = client.Post(cmd, query);  
var serializer = new JsonNetSerializer();  
var list = serializer.ToSearchResult<Person>(result);  
普通遍历获取 list.Documents 就可以了，如何获取“总搜索结果数”：searchResult.hits.total
```

8、案例 处理 verycd 电影数据

verycd 数据库是 SQLite 文件数据库，可以使用 navicat 打开 sqlite 文件：

首先安装 SQLite 的 ADO.Net SDK: [Install-Package System.Data.SQLite.Core](#)

创建连接: `SQLiteConnection conn = new SQLiteConnection(@"Data Source=d:\verycd.sqlite3.db")`

其他都是标准的ADO.Net用法。

把整个verycd数据导入我的电脑大约用了7个小时。上课测试的时候导入10000条就行了。

9、使用 ES 改造房源搜索。

面试问题“你们的数据量有多大、怎么优化”

```
.Query(b =>  
    b.Bool(m =>  
        //并且关系  
        m.Must(t =>  
            t.QueryString(t1 => t1.DefaultField("content").Query("周星驰"))  
        )  
        .Must(t=>t.Term(t1=>t1.Field("verycdid").Value("2746440")))//精确匹配  
(对于数字等不切词类型)  
    )  
)
```

C:\Users\yzk\Documents\Visual Studio 2015\Projects\ConsoleApplication3\ConsoleApplication3

<http://www.cnblogs.com/eggTwo/p/4425269.html>

说明：咱们做的不是手机 App，是适应手机的“网站”。简单介绍 Android、IOS 的“原生 App”开发技术以及“混合架构”技术：Xamarin、ReactNative。

讲思路如何拓展到其他项目，就业的时候准备自己的项目

配置首页