# TES总结

## 布局模板页Layout.cshtml

* 1. 我/你是如何避免前端代码重复的？我/你是如何设计前端的？

在项目中，大多数页面都具有相同的布局，样式，如果简单的复制粘贴，那么当需要更改布局样式时，改动是空前的，这会给项目后期维护带来很多的困难，时间与经济对于企业是非常宝贵的，在ASP.NET MVC中提供了模板页，通过将反复出现需要重用的公共页面提取为布局模板页，再通过在razor视图页更改Layout属性来使用布局模板页，这样就重用了视图，只需更改布局模板页，就可以将布局样式更改，有效的减少了后期维护的困难，并且布局模板页是支持反复嵌套的，一个视图使用的布局页又可以再次标记使用另一个布局页，然而这在过去WebForms所支持的ASPX视图引擎的母版页所不支持的，然而，这还不算什么，在razor中提供了部分视图Html.Partical()和Html.RenderPartical(),出于性能考虑我选择了后者，因为它直接将页面内容输出到HttpContext上下文，而Html.Partical()会将内容编码为MvcHtmlString，性能上有所损耗，尽管它使用起来简单一点，毕竟他不需要花括号，通过Html.RenderPartical()，我将经常常见的导航栏，菜单，消息框等控件封装，实现了前端模块化，这使得代码结构清晰，提高了易读性，重用率

## 基于数据槽（CallContext）：维护对象线程内唯一

HttpOneRequestFactory.cs

我们在一次请求中常常需要访问很多次数据库，而每次请求都要创建一个数据库上下文实例，交互数据库多次，效率会低一些，这里首先想到单例模式，不过在这里，很显然不适合，原因是使用单例模式，会使数据上下文对象得不到及时的资源释放，想象一下，无数个请求对数据库的访问，数据上下文容器无数次增加对Model对象的Attach监控，内存就爆了。

## 全局过滤器

AuthFilterAttribute.cs

IoC容器-Autofac

AutoMapperStartupTask.cs

使用视图模型隔离开数据库域模型，部分使用了AutoMapper将域模型转换为DTO（数据传输对象）,DTO更注重数据，对域模型进行合理封装，从而不会让域模型的数据过分暴露给表现层

通常在一个应用程序中，我们开发人员会在两个不同的类型对象之间传输数据，例如将数据库实体转换为DTO（数据传输对象），ViewModel（视图模型），这时，我们就需要将属于一个对象的某些属性值赋值给另一个对象的某些属性值，但是，这两个对象可能并不是完全匹配的，比如，两者之间的属性类型，名称等，如果我们采用手动映射转换，你或许可以想到定义显式类型转换，在方法内完成对象转换，或则在需要的地方，手动为每一个属性赋值，很显然，这是非常繁琐的，而指定域模型到指定视图模型的转换方法总是相同的，你可能会将这些代码反复复制，又或者使用了定义显式转换将域模型与视图模型捆绑在了一起，现在使用，AutoMapper 解决了这一问题，你只需在一处定义如何映射，再在应用程序启动时完成注册，这时无论在何处，你只需调用其AutoMapper带来的泛型扩展方法，即可轻松完成对象转换

错误日志

ELMAH

分部开发+集中部署：Areas

为什么需要分离？

在实际的开发中，随着项目规模的不断扩大，Controller控制器也随之不断增多。如果在Controllers文件夹下面有超过两位数controller，即便采用良好的命名规范，或者用子文件夹的形式区分不同功能的控制器，还是会影响项目的可阅读性和可维护性。因此，在一些场景下，如果能把与某功能相关的文件分离到一个独立的项目中是非常有用的

通过注册区域的方式，；两个开发者就能独立的开发后端的服务，将编译好的dll丢入bin目录中，按约定建立视图，这样后端即实现了服务独立，又能满足集中部署的需求