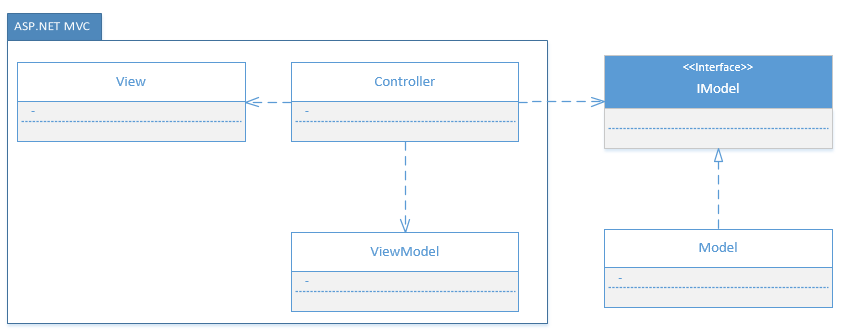
Asp.net MVC也接触好久了，但由于自己一直主要负责后台，尤其是数据库方面的工作对于该框架并没有一个很好的了解，尤其是蒋金楠大师的ASP.NET MVC4框架剖析一书都买了2年多了，真正认真看过并实践过的也就第一章而已。最开始的时候还老认为这本书不好，又厚又晦涩，不容易坚持读完，现在回想起来都觉得羞愧。标准的徒弟不好怪师傅的心态，当然学习的确需要一个过程，从浅入深会越来越喜欢。到目前为止，我最喜欢的书籍系列依然是入门经典系列，内容简单有习题，方便记忆与实践。最近有一段学习JAVA的过程，让我慢慢开始学着阅读源码，这才知道蒋大师是源码级别的剖析，并且比很多相关书籍好的地方时，基本上每一个复杂的知识点他都会根据自己的理解创建一个相对简单的小例子促进读者的理解，真心是棒棒哒。闲扯了变天，进入今天的正题—ASP.NET MVC与IOC的相关实践。

首先由于IOC是在Controller这章介绍，所以先来说说Controller方面的相关知识。ASP.NET MVC框架的主要的处理过程就是通过解析URL信息获取Controller的名称和Action的名称，同时通过Model的绑定机制将请求上下文所包含的请求数据转化为Action方法调用的参数列表。这儿想提及的一点就是关于Controller的执行是同步的还是异步的，实际上默认情况下是**异步**的，其中DisableAsyncSupport属性负责控制，默认为false，我之前为这块也困惑了很久。接下来提及的是ControllerFactory类中的SessionStateBehavior属性，有Default、Required、ReadOnly、Disable，到目前为止，我对于该框架中的~~会话~~仍然不是特别的清楚，之后弄清后回过来修改。

接下来用简图表示Controller激活的基本过程，请求到来后，HttpModule进行拦截，通过MvcRouteHandler调用GetHttpHandler方法找到对应HttpHandler处理类，即MvcHandler。然后调用该类的异步处理方法，注意这儿会始终使用异步处理方式，该方法中首先通过ControllerBuilder对象获取当前的ControllerFactory，然后通过控制器的名称找到并创建该控制器，让后调用[Begin]Execute执行，可以异步也可以是同步执行，之前有提及。Controller在执行完后，ControllerFactory会调用Release将其释放。

再则说说Controller类型的缓存，为了提高解析Controller的效率，框架对其进行了缓存，是一种XML文件形式的缓存，可以%Windir%\Micrisoft.NET的相关子目录中找到，与AreaRegistration的缓存方式类似，需要时将该XML文件反序列化为一个List<Type>对象即可。

最后介绍，本文的重点Controller中IOC的应用。在此之前，需要介绍一下应用IOC的原因，在实际的开发过程中，用户的请求会直接发送到Controller，这是如果是针对某项业务功能的调用，Controller会直接调用Model（也可以是外部的Service），当需要呈现数据时，回见Model中获得数据转化为ViewModel的形式用于呈现。但不管怎样，Controller都会与业务层的Model（领域模型）产生直接的依赖，不复合程序设计中依赖倒置的原则，因此引入IOC，降低耦合性。



在IOC的应用上，我们选择框架默认的DependencyResolver组件与Unity组件的组合，当前Unity组件已从codeplex搬到了github。有一个UnityMvc包原生支持IOC都不需要代码，直接配置即可，但由于版本兼容原因，附实现的代码如下，相关内容比较简单，就不介绍了。这儿的重点是相关整合方案有很多，这种是被使用最多的，当然蒋老师介绍的与Ninject的组合也很好。

|  |
| --- |
| public class UnityDependencyResolver : IDependencyResolver  {  IUnityContainer \_container;  public UnityDependencyResolver(IUnityContainer container)  {  \_container = container;  }  public object GetService(Type serviceType)  {  object instance;  try  {  instance = \_container.Resolve(serviceType);  if (serviceType.IsAbstract || serviceType.IsInterface)  {  return null;  }  return instance;  }  catch (Exception e)  {  return null;  }  }  public IEnumerable<object> GetServices(Type serviceType)  {  return \_container.ResolveAll(serviceType);  }  } |

参考：

[1]蒋金楠. ASP.NET MVC4框架揭秘[M]. 上海:电子工业出版社, 2012. 86-122