

## 什么是MVC

MVC是一种web的框架，主要是一种把界面和业务的设计思想

## MVC的参数传递的几种方法

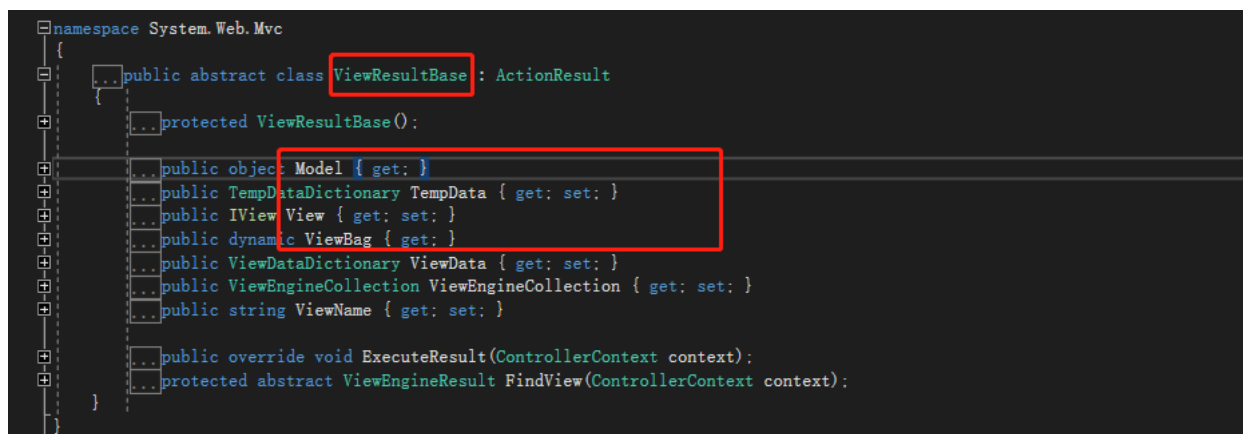
ViewData

ViewBag

view

model

传递值得几种方式都是由return view带过去的，view中是由急匆匆的ViewResultBase中的父类自动把相应的几种传值方式携带过去的。



```
namespace System.Web.Mvc
{
    ...public abstract class ViewResultBase : ActionResult
    {
        ...protected ViewResultBase();

        ...public object Model { get; }
        ...public TempDataDictionary TempData { get; set; }
        ...public IView View { get; set; }
        ...public dynamic ViewBag { get; }
        ...public ViewDataDictionary ViewData { get; set; }
        ...public ViewEngineCollection ViewEngineCollection { get; set; }
        ...public string ViewName { get; set; }

        ...public override void ExecuteResult(ControllerContext context);
        ...protected abstract ViewEngineResult FindView(ControllerContext context);
    }
}
```

## Layout

相当于模板文件，master

Global.asax

## 1.global.asax文件概述

global.asax这个文件包含全局应用程序事件的事件处理程序。它响应应用程序级别和会话级别事件的代码。

运行时，Global.asax 将被编译成一个动态生成的 .NET Framework 类，该类是从HttpApplication基类派生的。

因此在global.asax中的代码可以访问HttpApplication类中所有的public或者protected的成员

global.asax不被用户直接请求，但global.asax中的代码会被自动执行来响应特定的应用程序事件。

global.asax是可选的，而且在一个web项目中是唯一的，它应该处于网站的根目录。

## 2.一个请求的完整处理过程

以下过程由Internet Information Service (inetinfo.exe) (IIS) 执行

- 1.客户端发出请求
- 2.验证请求
- 3.给请求授权
- 4.确定请求的缓存

- 5.获取缓存状态
- 6.在请求的处理程序执行前
- 7.http处理程序执行请求（asp.net页面由aspnet\_wp.exe执行）
- 8.在请求的处理程序执行后
- 9.释放请求状态
- 10.更新请求缓存
- 11.请求结束

## 3.global.asax中的事件

global.asax中的所有事件可以分成两种，一种是满足特定事件时才会被触发，一种是每次请求都会被按照顺序执行的事件。



```
public class MvcApplication : System.Web.HttpApplication { protected void Application_Start(object sender, EventArgs e) { //不是每次请求都调用 //在Web应用程序的生命周期里就执行一次 //在应用程序第一次启动和应用程序域创建时被调用 //适合处理应用程序范围的初始化代码 } void Application_End(object sender, EventArgs e) { //不是每次请求都调用 //在应用程序关闭时运行的代码，在最后一个HttpApplication销毁之后执行 //比如IIS重启，文件更新，进程回收导致应用程序转换到另一个应用程序域 } void Session_Start(object sender, EventArgs e) { //不是每次请求都调用 //会话开始时执行 } void Session_End(object sender, EventArgs e) { //不是每次请求都调用 //会话结束或过期时执行 //不管在代码中显式的清空Session或者Session超时自动过期，此方法都将被调用 } void Application_Init(object sender, EventArgs e) { //不是每次请求都调用 //在每一个HttpApplication实例初始化的时候执行 } void Application_Disposed(object sender, EventArgs e) { //不是每次请求都调用 //在应用程序被关闭一段时间之后，在.net垃圾回收器准备回收它占用的内存的时候被调用。 //在每一个HttpApplication实例被销毁之前执行 } void Application_Error(object sender, EventArgs e) { //不是每次请求都调用 //所有没有处理的错误都会导致这个方法的执行 } /***** //每次请求都会按照顺序执行以下事件 *****/ void Application_BeginRequest(object sender, EventArgs e) { //每次请求时第一个出发的事件，这个方法第一个执行 } void Application_AuthenticateRequest(object sender, EventArgs e) { //在执行验证前发生，这是创建验证逻辑的起点 } void Application_AuthorizeRequest(object sender, EventArgs e) { //当安全模块已经验证了当前用户的授权时执行 } void Application_ResolveRequestCache(object sender, EventArgs e) { //当ASP.NET完成授权事件以使缓存模块从缓存中为请求提供服务时发生，从而跳过处理程序（页面或者是WebService）的执行。 //这样做可以改善网站的性能，这个事件还可以用来判断正文是不是从Cache中得到的。 } //----- //在这个时候，请求将被转交给合适程序。例如：web窗体将被编译并完成实例化 //----- void Application_AcquireRequestState(object sender, EventArgs e) { //读取了Session所需的特定信息并且在这些信息填充到Session之前执行 } void Application_PreRequestHandlerExecute(object sender, EventArgs e) { //在合适的处理程序执行请求前调用 //这个时候，Session就可以用了 } //----- //在这个时候，页面代码将会被执行，页面呈现为HTML //----- void Application_PostRequestHandlerExecute(object sender, EventArgs e) { //当处理程序完成对请求的处理后被调用。 } void Application_ReleaseRequestState(object sender, EventArgs e) { //释放请求状态 } void Application_UpdateRequestCache(object sender, EventArgs e) { //为了后续的请求，更新响应缓存时被调用 } void Application_EndRequest(object sender, EventArgs e) { //EndRequest是在响应Request时最后一个触发的事件 //但在对象被释放或者从新建立以前，适合在这个时候清理代码 } void Application_PreSendRequestHeaders(object sender, EventArgs e) { //向客户端发送Http标头之前被调用 } void Application_PreSendRequestContent(object sender, EventArgs e) { //向客户端发送Http正文之前被调用 } }
```

