# 控制器和动作

到达应用程序的每一个请求都是由控制器处理的。但要注意，不要把事务或数据存储逻辑放到控制器中，也不要生成用户界面。

在ASP.NET MVC框架中，控制器是含有请求处理逻辑的.NET类。其作用是封装应用程序逻辑。也就是说，控制器要负责处理输入请求、执行域模型上的操作，并选择渲染给用户的视图。

## 控制器的介绍

为了能够详细的说明控制器和动作的功能，这里使用“空（Empty）”模板创建一个名为“ControllersAndActions”的新的MVC项目（记得选择“创建单元测试项目（Create a unit test project）”）。

在MVC框架中，必须实现System.Web.Mvc命名空间的IController接口。这个接口很简单，只有唯一的一个方法：Execute，其在请求以控制器类为目标时被调用。MVC框架通过读取路由数据生成的controller属性值，便会指定请求的目标是哪一个控制器。

由于IController接口是一个相当低级的接口，因此必须做大量的工作才能达到预期效果。如下面所示的一个相当简单的用于演示的控制器类：

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

using System.Web.Routing;

namespace ControllersAndActions.Controllers

{

public class BasicController : IController

{

public void Execute(RequestContext requestContext)

{

string controller = (string)requestContext.RouteData.Values["controller"];

string action = (string)requestContext.RouteData.Values["action"];

requestContext.HttpContext.Response.Write(string.Format("Controller: {0},Action: {1}", controller, action));

}

}

}

上面代码仅仅演示了通过与请求相关联的RouteData对象读取controller和action变量的值，并将其显示出来。MVC框架并未指出控制器应该如何处理请求，也就是说可以采用任何方式来处理。需要注意的是，MVC框架并未在这个Basic控制器上强加视图引擎。如何产生响应是控制器本身要做的事，MVC框架不会对生成响应所用的技术做任何假设。

MVC框架可以无限定制和扩展。我们可以通过实现IController接口，来创建自己的控制器类，根据自己的需求来决定该如何处理请求。也可以通过System.Web.Mvc.Controller类来派生控制器。

该类提供了三个关键特性：

* **动作方法**（Action Method）：一个控制器的行为被分解为多个方法（而不是只有单一的Execute方法）。每个动作方法被暴露给不同的URL，并通过输入请求提取的参数进行调用。
* **动作结果**（Action Result）：可以返回一个描述动作结果的对象（如，渲染一个视图，或重定向到一个不同的URL或动作方法），然后通过该对象实现目的。这种指定结果和执行它们之间的分离简化了单元测试。
* **过滤器**（Filter）：可以把可重用的行为（如认证）封装成过滤器，然后通过在源代码中放置一个[Attribute]（注解属性）的办法，把这种行为标注到一个或多个控制器或动作方法上。

除非已经有了一个非常明确的需求，否则创建控制器最好的办法是通过Controller类进行派生，这也是Visual Studio创建控制器的默认方式。如下面通过这种方式创建的一个简单控制器：

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

namespace ControllersAndActions.Controllers

{

public class DerivedController : Controller

{

public ActionResult Index()

{

ViewBag.Message = "Hello from the DerivedController Index method";

return View("MyView");

}

}

}

Controller基类会实现Execute方法并负责调用动作方法，动作方法名与路由数据中的action的值匹配。

Controller类也连接到Razor视图系统。上面代码中返回的View方法的结果，在其中传递了希望渲染给客户端的视图名。下面是该视图的内容：

@{

ViewBag.Title = "MyView";

}

<h2>MyView</h2>

Message:@ViewBag.Message