# 控制器可扩展性

这部分主要研究一下如何配置控制器工厂和动作调用器，以便了解如何控制它们的行为；另外，也看看如何能够替代这些组件，并使用自己的逻辑。

下图是一个请求经过各组件时的基本流程，它演示了请求的处理过程：



调用一个动作方法

（一个请求在调用动作方法时的处理管道）

## 准备示例项目

为了解控制器的可扩展性，需要创建一个新项目。

项目名称：ControllerExtensibility

项目模板：Empty

基本结构：

* Models文件夹：Result.cs类——定义Result Model对象
* Views/ Shared文件夹：Result.cshtml视图——一个渲染控制器类中所有动作方法的视图（以Result类作为其模型，简单地显示ControllerName和ActionName）
* 控制器：Product、Customer

下面看看上述基本结构中各项的代码示例：

1、Result Model对象：

namespace ControllerExtensibility.Models

{

public class Result

{

public string ControllerName { get; set; }

public string ActionName { get; set; }

}

}

2、Result.cshtml视图：

@model ControllerExtensibility.Models.Result

@{

Layout = null;

}

<!DOCTYPE html>

<html>

<head>

<meta name="viewport" content="width=device-width" />

<title>Result</title>

</head>

<body>

<div>Controller: @Model.ControllerName</div>

<div>Action: @Model.ActionName</div>

</body>

</html>

3、Product控制器：

using ControllerExtensibility.Models;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

namespace ControllerExtensibility.Controllers

{

public class ProductController : Controller

{

public ViewResult Index()

{

return View("Result", new Result { ControllerName = "Product", ActionName = "Index" });

}

public ViewResult List()

{

return View("Result", new Result { ControllerName = "Product", ActionName = "List" });

}

}

}

4、Customer控制器

using ControllerExtensibility.Models;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

namespace ControllerExtensibility.Controllers

{

public class CustomerController : Controller

{

public ViewResult Index()

{

return View("Result", new Result { ControllerName = "Customer", ActionName = "Index" });

}

public ViewResult List()

{

return View("Result", new Result { ControllerName = "Customer", ActionName = "List" });

}

}

}

主要目的：研究MVC框架提供的对控制器和动作的管理进行定制的方式。

## 创建自定义控制器工厂

虽然实际项目中建议通过对内建的控制器工厂进行扩展，但现在需要理解其工作原理，所以，我们通过创建一个自定义控制器工厂进行探究。

控制器工厂是由IControllerFactory接口定义的，下面是接口的定义：

using System;

using System.Web.Routing;

using System.Web.SessionState;

namespace System.Web.Mvc

{

public interface IControllerFactory

{

IController CreateController(RequestContext requestContext, string controllerName);

SessionStateBehavior GetControllerSessionBehavior(RequestContext requestContext, string controllerName);

void ReleaseController(IController controller);

}

}

现在创建一个Infrastructure文件夹，并在其中建立一个简单的控制器工厂CustomControllerFactory：

using ControllerExtensibility.Controllers;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

using System.Web.Routing;

using System.Web.SessionState;

namespace ControllerExtensibility.Infrastructure

{

public class CustomControllerFactory : IControllerFactory

{

public IController CreateController(RequestContext requestContext, string controllerName)

{

Type targetType = null;

switch (controllerName)

{

case "Product":

targetType = typeof(ProductController);

break;

case "Customer":

targetType = typeof(CustomerController);

break;

default:

requestContext.RouteData.Values["controller"] = "Product";

targetType = typeof(ProductController);

break;

}

return targetType == null ? null : (IController)DependencyResolver.Current.GetService(targetType);

}

public SessionStateBehavior GetControllerSessionBehavior(RequestContext requestContext, string controllerName)

{

return SessionStateBehavior.Default;

}

public void ReleaseController(IController controller)

{

IDisposable disposable = controller as IDisposable;

if (controller != null)

{

disposable.Dispose();

}

}

}

}