文章编号:1672 - 058X(2011)04 - 0394 - 04

WebForms、MVC 和 MVP 在 ASP. NET 开发中的对比分析

顾明霞, 蔡长安
(盐城师范学院 信息科学与技术学院, 江苏 盐城 224002)

摘 要:介绍了. NET 平台下 WebForms、MVC 和 MVP 架构的工作原理,对三种结构的特点进行了分析和比较,特别对 WebForms 的 ViewState、性能和客户端 ID 污染等问题进行了阐述,并给出了解决方案;最后结合各自优势,指出三种架构适合的应用场景,旨在为 Web 应用系统开发提供参考。

关键词:软件架构;Web 系统;MVC;MVP;ASP. NET
中图分类号:TP311 **文献标志码**:A

1 架构概述

(1) WebForms。WebForms 是 ASP. NET 最初采用的 Web 开发的架构,它创造性的把 Windows 桌面应用中的表单模型引入到 Web 应用程序的开发中,同时采用了事件驱动的方法,将前端静态代码和后端程序完全隔离在两个文件里。通过控件模型,前端页面开发人员可以轻松地拖放控件, 后端用户可以使用. NET 平台上任意一种语言进行后端编程来响应页面和控件的各种事件来快速开发 Web 应用。

(2) ASP. NET MVC。MVC(Model-View-Controller)设计模式,把软件系统分为 3 个基本部分:模型(Model),视图(View)和控制器(Controller),适合于 Web 开发[1]。ASP. NET MVC 是微软官方提供的 MVC 模式编写 Web 应用程序的一个架构,目前最新版本 2.0。ASP. NET MVC 架构有助于以松耦合方式开发程序。视图部分负责生成应用程序的用户接口,仅仅是填充来自控制器部分传递而来的应用程序数据的 HTML 模板;模型部分则负责实现应用程序的数据逻辑,它所描述的是应用程序的业务对象;控制器部分对应一组处理函数,由控制器来响应用户的输入与交互情况。Web 请求都将由控制器来处理,控制器会决定使用哪些模型以及生成哪些视图,工作原理如图 1 所示。

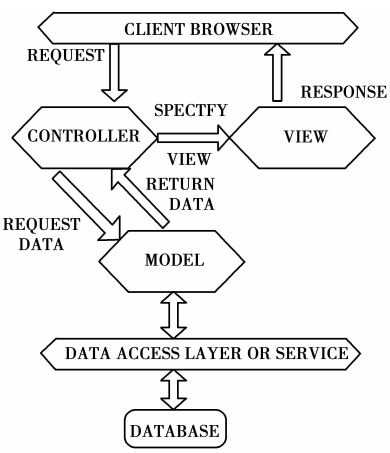

图 1 ASP. NET MVC

(3) ASP. NET MVP。系统开发时用户接口层经常沦为一个包含实际上应属于程序其他层的逻辑的容器。使得 UI 层中的代码非常难于调试。另外,在应用程序的公共视图之间会有大量的重复代码,很难找到好的可选重构方法。MVP(Model-View-Presenter)设计模式可将 UI 层中的逻辑和静态代码分离。MVP 最早由 Taligent 的 Mike Potel 提出。主要是为了解决 MVC 模式中结构过于复杂和模型-视图耦合性过高的问题。

表示层可以分为 UI(User Interface)和 P Logic(Presentation Logic)。UI 职责是接受用户的输入和向用户展示输出,UI 本身不应包含任何逻辑。P Logic 是表示层应有的逻辑性内容。例如,某个文本内容不能为空,

收稿日期:2011 - 03 - 10;**修回日期**:2011 - 04 - 21.
作者简介:顾明霞(1977 -), 女,江苏阜宁人, 讲师,从事 Web 应用系统、网络体系结构研究.

当某个事件发生时获取界面上哪些内容,这都属于 P Logic。P Logic 是抽象于具体 UI 的,它的本质是逻辑,可以复用到任何与此逻辑相符的 UI。UI 与 P Logic 之间的联系是事件,UI 可以根据用户的动作触发各种事件,P Logic 响应事件并执行相应的逻辑[4]。UI 与 P Logic 的结构及交互原理如图 2 所示。

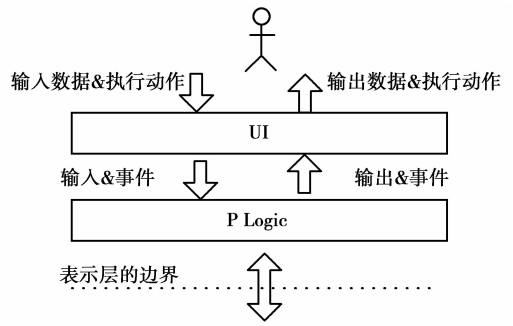

图 2　UI 与 P Logic

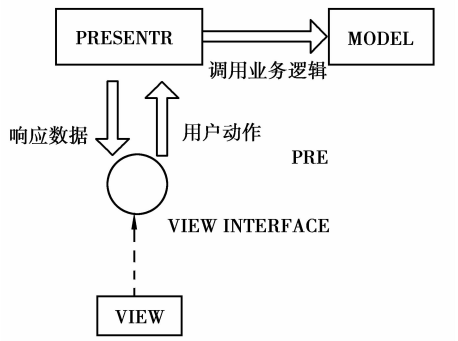

图 3　MVP 模式

MVP 核心思想是将 UI 分离成 View,将 P Logic 分离成 Presenter,而业务逻辑和领域相关逻辑都分离到 Model 中。View 和 Model 完全解除耦合,不再像 MVC 中实现一个 Observer 模式,两者的通信则依靠 Presenter 进行。Presenter 响应 View 接获的用户动作,并调用 Model 中的业务逻辑,最后将用户需要的信息返回给 View,MVP 工作原理如图 3 所示。

2　性能对比分析

2.1　WebForms 特点

(1) ViewState。由于 HTTP 协议是无状态协议,在到服务器的每个往返过程中页面被实例化、执行、呈现和处理。为了维护页面的 UI 状态,WebForms 中提出了 ViewState 的概念,数据被重新 Post 回页面时,UI 的状态就能恢复,从而支持一些复杂的控件事件。但是 ViewState 如果使用不当,页面体积就会增加许多,网络传输时影响性能。因此从当前页面跳转到到一个和当前页面完全无关的新页面,可以关闭 ViewState。

(2) 性能影响。WebForms 的组件模型具有生命周期,有时生命周期是“无谓”地一遍遍执行,影响了性能。但大多数的 Web 应用中,性能瓶颈大都是在数据库访问上,多执行一次数据库查询操作可能就能抵得上内存中 1 亿次引用拷贝[5]。对于此瓶颈通过优化数据库的查询,采用缓存等解决。

(3) 混乱的 HTML。WebFroms 中一些服务器控件,会生成丑陋的 HTML,难以进行样式控制。比如 GridView 不会生成 < th / > ,带来一些混乱,但是 WebForms 最关键的是 Control 模型,如 < div >　< uc1: DemoControl ID = " DemoControl1" runat = " server" / > < /div > ,生成的 html < div >　Hello World! < /div > ,非常干净,还有 Master Page, < asp:ContentPlaceHolder / > 、基础控件、显示批量数据 Repeater 等也不会产生任何多余的代码。

(4) 客户端 ID 污染。为了使组件内部的服务器控件最终生成的客户端 ID 能够在页面上唯一,WebForms 引入了 NamingContainer 这个概念。在 NamingContainer 中的服务器端控件最终在客户端生成的 ID,会使用 NamingContainer 的“客户端 ID”作为前缀。如此“递归”的做法保证了服务器控件在客户端的 ID 唯一。另外由于 AJAX 技术的应用,使用 JavaScript 操作 DOM 元素的情况非常多见。为了能在客户端获得 html 元素,服务器端的控件模型提供了一个 ClientID 属性,通过这个属性,就可以在服务器端得到控件最终在客户端的 ID。例如,如果在客户端访问 TextBox 对应的 HTML 元素,编写如下的代码:

```
< % @ Control Language = " C#" AutoEventWireup = " true" % >
< asp:TextBox runat = " server" ID = " textBox" / >
< script language = " javascript" type = " text/javascript" >
    document. getElementById( " < % = this. textBox. ClientID % > " ). value = " Hello World!" ;
< /script >
```

此时,当控件被放到页面上之后,它在客户端生成的代码则会是:

```
<input name = "DemoControl1 MYMtextBox" type = "text" id = "DemoControl1_textBox" />
<script language = "javascript" type = "text/javascript" >
    document. getElementById( "DemoControl1_textBox" ). value = "Hello World" !;
</script >
```

注意 <input /> 元素的 name 和 id,它们都留下了 NamingContainer 的痕迹。由于在页面上使用了 <%=% > 标记直接输出了服务器控件的 ID,这样在客户端的 JavaScript 代码也就可以正确访问到服务器端 <asp:TextBox /> 对应的客户端 <input /> 元素了。这种在设计器很难预测的客户端 ID,就是使用 WebForms 时的"客户端 ID 污染"问题,使得 Web 表单与 JavaScript 框架的集成比较困难。随着项目的发展,页面上复杂的 JavaScript 代码会越来越多,为了加快页面加载的速度,提高性能,通常会想办法将其转移到 js 文件中并且在页面上引用它们。问题是为了能够正确引用到页面上的某个服务器控件生成的 DOM 元素,必须在页面中使用 <%=% > 标记来输出控件的 ClientID,但是 <%=% > 无法写在 js 文件中,于是"客户端 ID 污染"也就成了一个使用 WebForms 时非常严重的问题。

(5) 对单元测试支持不好。整个 ASP. NET 框架中所有内容都是紧耦合型的并且仅使用一个类来负责显示输出和处理用户输入。因此,单元测试几乎是一项不可能的任务。但是当遵循敏捷软件方法论及相应惯例开发软件时,单元测试却是很重要的。

2.2 ASP. NET MVC 特点

MVC 模型使用其特定的控制器动作(Action)来代替 Web 表单事件,利用了基于 REST(Representational state transfer,表述性状态转移)的 URL 取代 Web 表单模型中所使用的文件名扩展方法,从而构造出更为符合搜索引擎优化(SEO)标准的 URL。WebForms 和 ASP. NET MVC 的 URL 生成方式对比如图 4 所示。

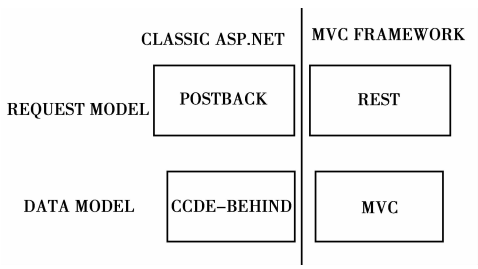

图 4 URL 构成区别

以利用 VS 2008 开发为例。新建一个 MVC 项目,即可看到自动生成的文件夹结构。默认情况下,ASP. NET MVC 项目有 6 个顶级目录。主要包括 Controls-放置 Controller 类,处理 URL 请求;Models-放置业务实体类,表示和操作数据;Views-放置 UI 模板文件,负责展示输出结果。

ASP. NET MVC 包含强大的 URL 路由引擎,灵活控制 URL 如何映射到控制器类。默认情况下,新的 ASP. NET MVC 项目已经注册了预配置的 URL 路由规则,这样允许轻松启动应用程序,而不需要配置任何东西。可以在项目中的 Application 类中看到默认的路由规则注册方法 RegisterRoutes()。

```
public static void RegisterRoutes( RouteCollection routes)
{   routes. IgnoreRoute( "{resource}. axd/{ * pathInfo}" );
    routes. MapRoute(
        "Default",                                          // Route name
        "{controller}/{action}/{id}",                       // URL with parameters
        new { controller = "Home", action = "Index", id = "" } // Parameter defaults
    );  }
```

上述调用的 routes. MapRoute() 方法注册了一个默认的路由规则,映射请求的 URL 到 controller 类。使用的 URL 格式为:/{controller}/{action}/{id},这里 controller 是指需要实例化的类名,action 是将调用的公共方法的名称,id 是一个可选的参数,嵌入在 URL 地址中,用来传递参数给方法。传递给 MapRoute() 方法的第三个参数是一组 controller/action/id 默认值,在 URL 没有指定时,Controller = Home、Action = Index、Id = ""。利用 ASP. NET MVC 架构开发 Web 应用程序时,没有数据回传,没有在页面中保存视图状态,开发者可以完全掌控页面的呈现过程,同时易于单元测试,进行测试驱动开发。ASP. NET MVC 架构仍然支持 WebForm 中的有关特性,如:用户控件、母版页、数据绑定、本地化等,不足的是需要开发人员具备更多的 html、css 和 javascript 知识;在服务器端和客户端需要编写更多的代码。

2.3　ASP. NET MVP 特点

MVP 模式将 View 和 Model 完全解耦,两者不发生直接关联,通过 Presenter 进行通信。Presenter 并不是与具体的 View 耦合,而是和一个抽象的 View Interface 耦合,View Interface 相当于一个契约,抽象出了对应 View 应实现的方法。只要实现了这个接口,任何 View 都可以与指定 Presenter 兼容,从而实现了 P Logic 的复用性和视图的无缝替换。View 在 MVP 里是一个“极瘦”的概念,最多也只能包含维护自身状态的逻辑,而其他逻辑都应实现在 Presenter 中。使用 MVP 模式将 UI 和 P Logic 两个关注点分离,得到更干净和单一的代码结构。实现了 P Logic 的复用以及 View 的无缝替换。而且实现起来比 MVC 在结构上要简单很多。WebForms、MVC 和 MVP 特点比较如表 1 所示。

表 1　WebForms、MVC 和 MVP 特点对比

	是否有客户端 ID 污染	View 和 Model 是否耦合	是否支持单元测试	表示层逻辑是否支持复用	复杂度
WebForms	是	是	否	否	低
MVC	否	是	是	否	高
MVP	否	否	是	是	较高

3　结　论

WebForms、MVC 和 MVP 3 种架构的选择根据实际开发需求进行定夺,如果要更多控制 HTML,关注 Web 标准,可访问性,构建 SEO 友好的 URL,希望集成进 jQuery 以及其他 JavaScript 开发框架,并且要进行测试驱动开发的话,可以选择 MVC 模型。如果要实现对系统的更好的控制以及想实现面向状态的事件驱动的 Web 开发的话,可以选择 Web 表单模型。MVC 模型应用有一定的技术难度,没有 WebForms 简单。通过禁用 ViewState,WebForms 模型能达到 MVC 的页面干净、传输数据小的效果,另外通过设置一个短暂页面级缓存的话,也可达到 MVC 没有页面生命周期,创建速度快的特点。如果不希望 UI 具有过多的逻辑,显的臃肿不堪,系统有 C/S 到 B/S 等的扩展需求时可以采用 MVP 架构。

参考文献:

[1] 林庆. 基于 ASP. NET 的 MVC 设计模式的研究[J]. 计算机工程与设计,2008(1):167-169
[2] 谢珩. MVC 模式在 Web 应用中的一种实现[J]. 计算机科学,2006(5):150-153
[3] WALKTHROUGH. Creating a Basic MVC Project with Unit Tests in Visual Studio [EB/OL]. [2008-10-15]. http://quickstarts.asp. net/previews/mvc/mvc_CreateMvcProjectWalkthrough. htm
[4] . NET 平台上的 Model-View-Presenter 模式实践[EB/ OL] . 张洋博客:http://www. cnblogs. com/leoo2sk/archive/2010/01/28/mvp-in-practice-based-on-dot-net. html
[5] ASP. NET 开发经验[EB/ OL] . 赵劼博客:http://www. cnblogs. com/JeffreyZhao/archive/2007/12/22/Experience-for-Asp-dot-net-and-WebForms. html
[6] 张恩慧. MVC 模式在 Asp. net 下的分析[J]. 硅谷,2008(18):148
[7] PETER D. Blackburn William(Bill) Vaughn. ADO. NET Examples and Best Practices for C# Programmers. New York:Apress, 2006:210-213
[8] 何玲娟, 蚁龙, 刘连臣. 一种松耦合高复用 MVC 模式的 Web 分页实现. [J]. 计算机工程与应用, 2007, 43(15).
[9] MICHAEL KIRCHER, PRASHANT JAIN. Pattern-oriented software architecture: Patterns for resource management, volume-3 [M]. Hoboken, USA:John Wiley & Sons 2004:163-168
[10] MARTIN F. Patterns of enterprise application architecture[M]. Boston,USA:Addison-Wesley,2002:170-176

(下转第 409 页)