MVC 模式在 Web 应用中的一种实现*)

谢 珩 吴多益 卢显良 宋 杰
(电子科技大学计算机科学与工程学院 成都 610054)

摘 要 MVC(Model-View-Controller)是 Web 应用开发中常用到的一种设计模式,通常,由 Model 和 Controller 在服务器端生成 View,浏览器端只是简单地对 View 进行显示。在这种实现方式下,几乎所有的处理都集中在了服务器端,浏览器端的空闲处理能力被白白闲置。同时,频繁传送包含大量数据的页面,对网络带宽也有很高的要求,这已经不能满足用户对 Web 应用越来越高的交互需求。为此,本论文基于 JavaScript、XMLHttp、DOM 和 Template Toolkit 等几项技术,提出了以浏览器为主的 MVC 设计模式的一种新实现,降低了服务器的处理负荷,减轻了网络负担,有效地解决了传统方式的不足,同时也有利于传统 GUI 应用开发人员掌握和使用。

关键词 MVC 模式, JavaScript, XMLHttp, DOM, Template toolkit, Web 应用

A MVC Pattern Implementation on Web Application

XIE Heng WU Duo-Yi LU Xian-Liang SONG Jie
(School of Computer Science and Engineering, UEST of China, Chengdu 610054)

Abstract MVC(Model-View-Controller) is a traditional design pattern, which is commonly used in the Web application development. Usually, browser simply answers for the display of View generated by Model and Controller on server side. According to this traditional implementation of MVC, most work is so concentrated on server that the idle capability on client side is left unused wastefully, and frequent transmission of pages containing many data requires a higher bandwidth. Aiming at these drawbacks, this paper introduces a novel implementation of MVC mainly working on browser by using JavaScript, XMLHttp, DOM and Template Toolkit technology etc, which alleviates the server's burden and reduces network traffic greatly. As a result, this new method effectively solves the disadvantages of traditional implementation, and is easy for traditional GUI Web application developer to master and use.

Keywords MVC pattern, JavaScript, XMLHttp, DOM, Template toolkit, Web application

1 引言

随着 Web 技术的飞速发展,Web 应用已从简单的页面展示发展到了系统全方面功能的实现,使得 Web 应用开发也越来越复杂,如何快速开发出稳定健壮的 Web 应用程序,已成为当前面临的巨大挑战。为了适应这一挑战,开发者们提出了一种 Web 组件的开发技术,并引入了许多基于 Web Component 的开发模式,MVC(Model-View-Controller)模式就是其典型代表[1]。

由于 MVC 模式在 Web 应用中组件化开发的优势十分明显,MVC 模式近几年得到了广泛深入的研究,并且产生了许多成熟的基于 MVC 模式的 Web 框架,其中,Struts[2]就是目前用户群最大、开发厂商支持最多的开源 Web Framework[3]。但这些 Web 框架往往都侧重于服务器端 MVC 模式的实现,忽略了客户端闲置的处理能力,对强大的客户端技术如 Flash、Javascript、Applet 和 Activex(特别是微软 XMLHttp)等技术也注重不多,从而把大部分的工作都集中在了服务器端处理,使得服务器负荷很重,同时,在网络上频繁传送包含大量数据的页面,对带宽也有很高的要求,这两个缺陷造成页面刷新很慢(白屏)。

针对 MVC 模式传统实现方式的以上缺点,本文基于 JavaScript、XMLHttp、DOM 和 Template Toolkit[4]等几项技术,提出了一种以浏览器为主的 MVC 设计模式的新实现方式。在这种实现方式下,浏览器显示的页面中,加入了由 JavaScript 编写的 Controller、Model 引擎和控件库;服务器则主要扮演一个存储数据和通过 View 模板、Controller 和 Model 脚本与浏览器合作的角色。浏览器首先必须通过服务器脚本和 View 模板取得 Controller、Model 引擎和控件库,然后 Controller 引擎再根据需要,通过 Model 引擎从服务器直接取得原始数据,并把数据交给 JavaScript 控件以生成或无刷新更新页面。

和传统实现方式相比,本文的实现方式具有以下几个优点。首先,页面的生成不再由服务器来完成,而是在浏览器上完成,有效地减轻了服务器的负荷;其次,从服务器送到浏览器的主要是原始数据,从而减少了网络通信量,也有利于 Model 引擎对它们进行缓存或再处理,甚至把同一组数据提供给不同框架中的视图使用;再次,Controller 引擎从 Model 引擎取得的数据,结合 DOM 技术,还可以实现无刷新地更新视图,用户几乎感觉不到白屏现象;另外,JavaScirpt 控件的引入,不仅提高了界面的交互性能,而且它的开发比较接近 WINDOWS 本机 GUI 的开发模式,这便于这类程序员转移到 Web 应用开发的领域。

本文第 2 节介绍系统的工作原理及其实现细节;第 3 节讨论当前 MVC 模式的实现模型,并与类似的 Ajax 技术进行了一些比较;最后是结论和未来工作。

2 工作原理及其实现细节

2.1 浏览器端 MVC 引擎模型描述

*)基金项目:国家信息产业部电子发展基金(信部运[2004])。谢 珩 硕士研究生,主要研究方向:计算机网络、操作系统;吴多益 本科生,主要研究方向:Web 应用技术、电子商务技术;卢显良 教授,博士生导师,主要研究方向:计算机网络、操作系统;宋 杰 博士研究生,主要研究方向:计算机网络、操作系统。

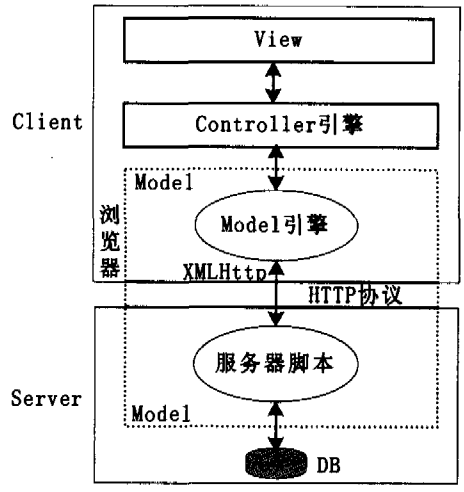

图 1　浏览器端 MVC 引擎模型

浏览器端 MVC 引擎模型，是以浏览器引擎为主、服务器脚本与其合作的 MVC 实现模型，该模型将 MVC 模式的 Model 分成了浏览器 Model 引擎和服务器脚本两部分，其模型结构如图 1 所示。

该模型的特点是，Controller 引擎根据 View 需求，调用 Model 引擎的 Modify 方法，Model 引擎则通过 XMLHttp 组件请求服务器脚本修改数据，由服务器脚本配合完成服务器端的数据变更，并将处理结果通知给 Model 引擎。然后，Controller 引擎根据此结果，使用 Model 引擎的 Get 方法，再次访问服务器脚本将数据 Pull 到浏览器端，以生成或无刷新更新页面。这个模型的优点在于，Model 引擎利用客户端技术封装了数据和访问方法，提供了数据的易操作性和可复用性，同时，原始数据的直接交换，减轻了服务器的负担，降低了网络流量，从而加快了响应速度，缩短了用户的等待时间。

2.2　系统工作原理

本系统利用浏览器端 MVC 引擎模型的上述优点，结合服务器端 MVC 实现模型的优势，采用浏览器端 MVC 引擎模型为主，服务器 MVC 的 View 模板、Controller 和 Model 脚本为辅的方法，使 MVC 模式在浏览器端和服务器端发挥出双重功效，整个系统工作原理如图 2 所示。

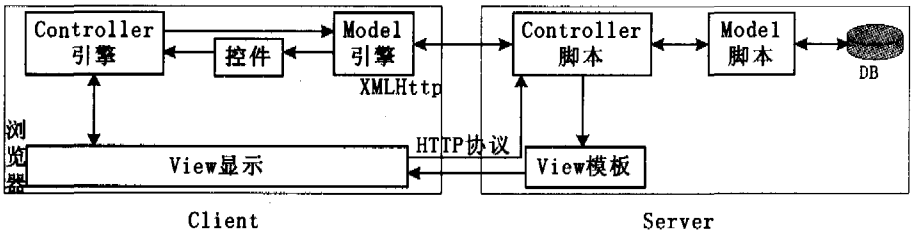

图 2　系统工作原理

浏览器首先要通过服务器脚本和 View 模板从服务器取得 Controller、Model 引擎和客户端控件库，在取得这些代码之后，浏览器就直接使用 Controller 引擎访问 Model 引擎，Model 引擎再操作 XMLHttp 组件从服务器取得数据，交给 JavaScript 控件来生成或刷新页面。对服务器而言，在通过脚本和 View 模板送出 Controller、Model 引擎和控件库后，其工作就主要是用 Controller 脚本处理 XMLHttp 组件的请求，访问 Model 脚本修改和获取数据库中的数据，最后将原始数据返回给浏览器。

2.3　系统实现细节

2.3.1　*系统实现*　在浏览器端，JavaScript 技术具有接收事件和生成较为复杂页面的能力，用 XMLHttp 组件直接传送数据能减轻服务器负担，两者再结合 DOM 技术，不仅能加快响应速度、缩短用户等待时间，而且还可以动态无刷新地更新页面，从而使浏览器与用户的交互能力大大加强。因此，对于浏览器端的 MVC 引擎实现，本系统采用 JavasScript 设计 Controller 引擎，依靠 JavaScript 面向对象方法编写可复用的 Web 控件库，结合 XMLHttp 组件提供 Model 引擎，其中，XMLHttp 组件主要负责与服务器端 Controller 脚本的交互，以获取原始数据。

在服务器端，系统使用 Perl 编写 Controller 脚本作 CGI 程序，采用 Perl 面向对象方法提供封装了数据及其操作的 Model 脚本，在 View 模板中嵌入浏览器需求的 JavasScript 的 Model 引擎、控件库以及 Controller 引擎，并使用 Template Toolkit 技术，在 View 模板再加入其它动态参数返回给浏览器。为进一步提升服务器性能，系统在 Apache 服务器中嵌入了 Mod_perl[5] 方式来取代 CGI 方式，大大降低了执行 CGI 程序的开销，提升了 Perl 脚本的工作效率；同时，系统采用 BerkeleyDB[6] 替代了传统的关系数据库，避免了数据库连接和 SQL 语句分析的开销，提高了数据操作的效率。

2.3.2　*主/子框架机制*　其优点在于：首先，利用主框架页面变动小的特性，置入 JavaScript 的 Model 引擎和控件库，使得浏览器一次性获取这些代码后就可长久使用；其次，可以在主框架内使用多个子框架，并根据每个子框架中页面的不同功能，分别嵌入不同的 Controller 引擎，同时，这些不同功能的 Controller 引擎，都可以很方便地使用驻留在主框架内的 Model 引擎与服务器交互，并用 JavaScript 控件库来生成交互良好的 Web 界面。另外，这种机制还可以使用同一组数据为不同框架提供不同的视图，当 Model 引擎从服务器取得新的数据后，利用 MVC 采用的 Observer 模式，就能同时更新各个框架内的不同视图。

2.3.3　*参数传递机制*　每个子框架中的页面，往往需要把当前页面中的一些参数，传递到下一个页面。因此，该机制使用页面中的 Controller 引擎，搜集页面中的相关参数，然后通过 Model 引擎送往服务器，由服务器脚本在 View 模板中加入这些参数，返回给浏览器。在多子框架的结构中，甚至可以使用这个机制来实现一个子框架页面的 Controller 引擎去控制另一个子框架内不同页面间的参数传递。

2.3.4　*客户端 Cache 机制*　这种机制使得浏览器缓存已请求到的原始数据和 JavaScript 代码，从而浏览器第一次获取 JavaScript 代码后就可长久使用，数据也只有在发生改变时才会重新去服务器获取，这节省了传送冗余数据和代码的时间，降低了网络流量，提高了系统的响应速度。其中对于数据的更新，系统采用了主动更新和定时查询相结合的办法。主动更新，是指 Model 引擎通过服务器脚本修改数据成功后，再次去请求服务器脚本将数据 Pull 到浏览器端，由 Controller 引擎生成或更新页面的方法；定时查询，则是浏览器端定时用 XMLHttp 方式查询服务器端相应的状态信息，然后

与本地的状态信息相比较，如果不同，则说明服务器端的数据发生了改变，此时再由 Model 引擎用 XMLHttp 方式去获取数据，用 Controller 引擎生成或更新页面。

3 相关工作

3.1 MVC 模式的实现模型分析

由于 MVC 模式在 Web 应用开发中的灵活性、可复用性和易维护性等一系列特点，现在已有了很多成熟的开源 MVC Web 框架，如 Apache 项目 Struts、OpenSymphony 项目 WebWork、Enhydra 项目 Barracuda、Sun 公司的 JSF(JavaServer Faces)，以及 Spring MVC 和 Maverick 等[3]。按照网络架构，现有 MVC 模式的主要实现，可总结为 B-MVC-S(Brower-MVC-Server)和 SMVC (Server-MVC) 两种实现模型，其模型结构分别如图 3 和图 4 所示。

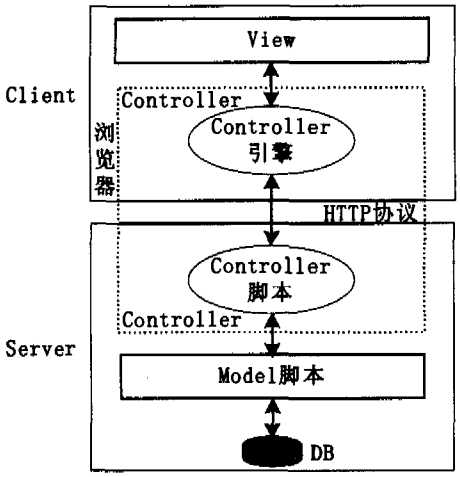

图 3 B-MVC-S 模型

B-MVC-S 模型[7]，优点是易于替换浏览器端与服务器的通信方法，缺点是原始数据及其访问方法在浏览器端没有很好地封装，数据可重用性差。本系统的实现中，在浏览器端采用 Model 引擎对数据进行了缓存和封装，便于数据的再使用。

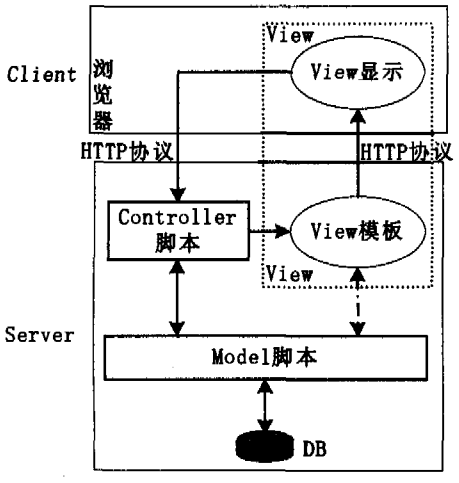

图 4 SMVC 模型

SMVC 模型[3]，是目前最为广泛的实现方式，其优点在于浏览器端是 100%的瘦客户，缺点是页面生成完全集中在服务器端，使得服务器负担很重，同时，频繁传送包含大量数据的页面，大大加重了网络流量，用户也能明显感觉到页面的重载过程(白屏)。本系统的实现方式，由浏览器分担页面生成工作，网络上频繁传送的也主要是原始数据，弥补了上述两个不足。

3.2 Ajax 技术分析

Jesse James Garrett 提出的 Ajax[8] (Asynchronous JavaScript and XML)技术，其实质是 JavaScript、DOM、XMLHttpRequest、XML 等各项成熟技术的综合应用，目前已有几十个网站[9]使用了 Ajax 技术，如：Gmail、Google Groups、Google Suggest、Google Maps 等[8,9]。Ajax 的优势在于，数据与呈现分离，利用了客户端闲置的处理能力，按需取数据，减轻了服务器和带宽的负担，实现了无刷新更新页面[10]。

当前的 Ajax 框架[11]，也提供类似浏览器端 MVC 引擎的基础框架，但这些 Ajax 框架只注重 MVC 引擎在浏览器端的实现。随着 Ajax 的进一步发展，也开始出现了 Ajax 与服务器端 MVC 框架的集成，如 Ajax 与 JSF 合成的 AjaxFaces[12]、Ajax 结合 Struts 的 xhrstruts[13]等，这些集成，在浏览器端采用 Ajax 引擎，服务器端使用 MVC 框架，使得浏览器端和服务器端一样采用了 MVC 模式。与这些集成方式相比，本系统实现的不同之处，在于多视图的主/子框架机制、依靠 View 模板的参数传递机制，以及主动更新和定时查询相结合的数据缓存机制，提高了系统的交互性能，保证了数据的及时更新。

结论 本文基于 JavaScript、XMLHttp、DOM 和 Template Toolkit 等几项技术，提出了一种以浏览器为主的 MVC 设计模式的新实现。在这种实现方式下，浏览器首先通过服务器脚本和 View 模板取得 JavaScript 的 Controller、Model 引擎和控件库后，就由 Model 引擎去请求原始数据，Controller 引擎使用 JavaScript 控件生成或无刷新更新页面。该实现有效地减轻了服务器负担，降低了网络流量，可以无刷新地更新页面，也易于传统 GUI 应用开发人员掌握和使用，将其应用到本项目的远程技术支持系统产品中，已取得了良好的效果。

未来的工作是：1. 采用 Web 标准化规范，使用 XML 表示和传送原始数据，将 XSLT 替换 Template Toolkit 模板技术来传递参数，以及返回 JavaScript 的 Controller、Model 引擎和控件库；2. 在使用 XML 的基础上，采用 BerkeleyDB XML 数据库，以避免数据库存取时 XML 转换的开销，提高数据操作效率；3. 引入 Command、Strategy 等设计模式，以进一步提高 Web 应用中组件化开发的复用性和高效性。

参考文献

1 林仪明. Web 开发技术史话[EB/OL]. http://blog.csdn.net/emag_java/archive/2005/01/12/250060.aspx, 2005
2 何成万，余秋惠. MVC 模型 2 及软件框架 Struts 的研究[J]. 计算机工程，2002，28(6)：274～275，281
3 王海龙. Java Web Framework 综述[EB/OL]. http://blog.csdn.net/emag_java/archive/2005/01/13/252039.aspx, 2005
4 Chamberlain D, Cross D, Wardley A. Perl Template Toolkit [M]. California: O'Reilly & Associate Inc., 2003. 10～124
5 Bekman S, Cholet E. Practical Mod_perl [M]. California: O'Reilly & Associate Inc., 2003. 11～303
6 Olson M A, Bostic K, Seltzer M. Berkeley DB [A]. In: Proc. of 1999 USENIX Annual Technical Conference [C], Monterey, California USA, June 1999
7 何成万，余秋惠. JnetMVC——一个新的 Java 网络编程 MVC 模型[J]. 计算机工程，2001，27(8)：74～75
8 Garrett J J. Ajax: A New Approach to Web Applications [EB/OL]. http://www.adaptivepath.com/publications/essays/archives/000385.php, 2005
9 Witbeck S. Sites Using AJAX [EB/OL]. http://www.ajaxmatters.com/, 2005
10 GUO A. Ajax 内部交流文档[EB/OL]. http://www.dragonson.com/doc/ajax.html, 2005
11 Mahemoff M. Ajax Frameworks [EB/OL]. http://ajaxpatterns.org/Ajax_Frameworks, 2005
12 CyberXP. NET Inc. CyberXP. NET AjaxFacesTM Solutions and Components [EB/OL]. http://cyberxp.net/, 2005
13 Zammetti F W. Ajax using XMLHttpRequest and Struts [EB/OL]. http://www.omnytex.com/articles/xhrstruts, 2005