基于 JavaScript MVC 模式的 Wiki 设计与实现①

刘烨辉，王加阳，王安莉

(中南大学 信息科学与工程学院，长沙 410083)

摘 要：MVC 模式常被应用于大型的 B/S 应用程序开发，它提供了一个结构化的模型，实现了软件开发的分工和应用程序的模块化。深入研究了一个 Wiki 系统在客户端浏览器中的设计与实现，在实现过程中完全使用 JavaScript 并结合 MVC 模式，使得整个应用程序具有较高的系统性，由于使用了模块分割提高了代码复用率，降低了维护成本。

关键词：MVC 模式；Wiki 系统；浏览器；JavaScript

Wiki Design and Realization Based on JavaScript Pattern

LIU Ye-Hui, WANG Jia-Yang, WANG An-Li

(College of Information Science and Engineering, Central South University, Changsha 410083, China)

Abstract: MVC pattern is usually used for large-scale B/S application development. It supplies a structured module, and realizes software development's job-share as well as application's modularization. This paper in depth studies a WIKI system's design and realization in client's browser, in the process of realization. It fully uses JavaScript and combines with MVC pattern. So it makes all the application program higher systematic. At the same time, it reduces the maintenance cost because of using divided template and increasing code reusability.

Key words: MVC pattern; WIKI system; browser; JavaScript

1 概述

在基于 Web 应用的软件开发中，MVC 模式被广泛应用。该模式的核心思想[1,2]是将程序分成相对独立，而又能协同工作的 3 个部分：模型、视图、控制器。降低模块之间的耦合，为大型可扩展的 Web 应用开发提供一个结构化的模型[3]，非常适合多类型用户，可扩展、可维护的系统。

JavaScript 是一种工作在客服端的脚本语言。Web2.0 时代的到来，Ajax 技术使得客户端可以直接连接到服务器且不用刷新整个页面。而 JavaScript 是这门技术的重要组成部分，因此它也从以往的“玩具”语言一跃成为 B/S 架构中客服端的主角，而利用 JavaScript 实现的 MVC 模式使得用户界面、用户操作和具体的业务逻辑(后台的数据库)分离，能够实现软件开发的分工和应用程序的模块化。

Wiki 一词来源于夏威夷语的“wee kee wee kee”，它指一种超文本实现的技术，用以实现超文本系统。这种超文本系统支持面向社群的协作式写作，同时也包括一组支持这种写作的辅助工具[4]。与其它超文本系统相比，Wiki 有使用方便及开放的特点，这种方便快捷的操作使得 Wiki 成为一种团体协作书写的高效工具。

2 MVC在ASP.NET平台的实现

应用程序可分为三个部分：客户端、服务器端和数据库。客户端负责接受用户操作并处理界面，服务器端维护客户端和数据库之间的通信，其中客户端采用 JavaScript 向服务器端发送 Ajax 请求。本文将着重描述应用程序的客户端实现。整个系统的架构如图 1 所示：

① 基金项目:长沙市节能专项资金

收稿时间:2010-10-29;收到修改稿时间:2010-12-06

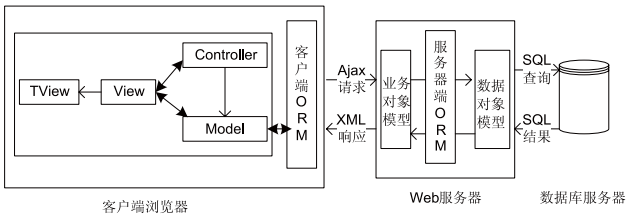

图 1　应用程序系统架构

该应用的独特之处在于：客户端 Wiki 引擎的核心完全用 JavaScript 编写，同时借鉴了 Prototype JS 类库，以便支持面向对象的编程风格，并结合 MVC 模式，使之在客户端实现。

2.1 模型的设计与实现

在传统的 Web 应用程序中，所有的智能处理都位于服务器端，客户端仅仅是显示服务器处理后的结果，所以无论使用何种语言，业务模型也位于那里[5]。在 Ajax 应用中，我们依然需要在服务器端建立模型，因为一些资源只能在服务器端获得(如数据库资源)。然而，为了使客户端具有快速的响应性，我们希望客户端浏览器也能维护一部分业务模型。另一方面，如果客户端在处理用户交互的时候仅仅做出非常简单的决定，我们可以编写少量代码来处理，但是不可能充分利用 Ajax 应用中智能客户端的长处，系统仍然会反应迟钝。所以，如果需要客户端针对自身做出更加重要的决定，那么它就有必要知道一些业务领域的事情。从这个角度上看，除了在服务器端建立一个模型,我们还需要在客户端也实现一个业务领域的模型。

在服务器端，已经有很多框架来实现对象-关系映射(ORM)，它们能够很方便地使得服务器端的对象模型能够与数据库之间双向交互。为了在客户端为业务领域建模，我们仍然可以利用服务器端的 ORM 框架，通过在服务器端定义细粒度的、容易处理的对象，为从数据库到客户端传播数据定义一条清晰的路径。

首先，从数据库创建了一个服务器端对象模型。这样，就可以将数据读入对象，修改它，然后保存数据。然后，序列化数据库字段生成对象模型的 XML 数据流。最后，在客户端解析该数据流，从而在 JavaScript 层创建对象，生成客户端业务模型。在该应用程序中，我们也实现了客户端解析 XML 从而生成对象的 JS 类库。如代码 1 所示:

在 B/S 应用程序中，由于不同的浏览器之间标准的差异，我们往往不得不针对特定的浏览器编写脚本语言。但是在编写客户端模型的时候，我们纯粹与 JavaScript 打交道，而与特定于浏览器的功能毫无关系。

```
function wiki_Receive(wikiData)
{
//解析来自服务器端的 XML 字符串 wikiData, 返回一个 JavaScript 对象 desc
return desc
}
Wiki.Model.KImage.prototype =
{
_title:getFromObject(desc, "title",""),
_type:getFromObject(desc, "type",""),
getId:function(){return this._id;},
getType:function(){return this._type;},
_load:function(desc){this._appx = desc;},
getPrefix:function(type){
return Wiki.Config.getComponentPrefix(type);
},
    getPureId:function(){return this._id.substring(this.getPrefix(this._type).length);},
getUrl:function(){return this._appx;},
    getTitle:function(){return this._title.replaceAll("<","&lt;").replaceAll(">","&gt;")},
callServer: wiki_post(postXml)
}
function getFromObject(obj, key, def){return (obj && (key in obj)&& obj[key])?obj[key]:def;}
function wiki_post(xmlData)
{
//Ajax 调用服务器端方法，并指定回调函数，用于更新返回后的用户界面更新
}
```

代码 1 客户端 KImage 对象的 JavaScript 模型

getFromObject(obj, key, def)是我们定义的一个辅助函数，它的作用是从客户端反序列化之后的业务对象 obj 得到 key 属性值，若 obj 没有该属性则返回 def。

在上面的实例代码中，我们利用 JavaScript 声明了一个 KImage，并从、经过客户端 ORM 映射的 desc 中生成了 KImage 各个属性的值。

通过上述方式，我们可以把服务器端的业务模型都转化为客户端的业务模型。

2.2 视图和控制器的设计与实现

上面我们已经在客户端得到了底层的业务模型现在需要做的是，从底层的模型自动生成用户界面，或者至少自动生成用户界面的一部分。在这个应用程序中，我们采取一种独特的方式来生成模型的视图表示，并给视图添加事件，从而使 View 和 Controller 关联起来。我们需要考虑的是：Web 应用中，控制器层由事件处理函数组成，但为了保持 Controller 和 View 分离，可以采用编程方式添加事件。除了使用嵌入的事件处理函数，我们还可以制定某种类型的记号，它随后会被代码获得，我们可以给元素附加唯一的 ID，以每个元素为基础制定事件处理函数。例如：

```
Wiki.TView. KImage ={
tmpl:'<div class="wiki_kimg" id="wiki_#VID#" onclick="#FUNC1#" >#IMG#</div>',
_paramList:{
tmpl:["IMG", "VID", "FUNC1"]
}
}
```

代码 2 客户端 KImage 模型的 JavaScript 模板视图

在上面的代码中，我们设计实现了一个 TView. KImage 模板视图对象，并用 JavaScript 简单对象表示(JSON)方式描述了该对象在显示时呈现的 DOM 元素以及绑定在该 DOM 元素上的事件。需要注意的是，tmpl 属性对应的 DOM 元素是字符串且带有参数的，_paramList 属性是一个参数的数组，它记录了 TView.KImage 对象里 tmpl 所对应的 DOM 元素所需要的参数。用这种方法，在我们真正生成该 TView. KImage 对象所对应的视图时，只需要逐个的给该模型的模板视图(TView)对象传递所需的参数即可。当 Controller 调用 KImage.View 的 show()方法时，最终会被传递这些参数值，那么该对象所需呈现的 DOM 元素就显示出来了。例如，KImage 对象的 show 属性就是给 TView.KImage 的_paramList 属性传递参数

```
Wiki.View.KImage = Class.create();
Wiki.View.KImage.prototype = {
show:function(){return this.pf("KImage.tmpl", [this.__renderIcon(this._model.getUrl()),this._id,fun("show")]);},
initialize:function(model){
    this._init_View(model);
},
_init_View:function(model) {
    this._model =model;
}
};
__IconTmp: '<img src="#_IMG#" title="#_ALT#" class="#_CLASS#" #_PROPS#/>',
__renderAIcon:function(img,alt, clz, propsStr) {return this.__IconTmp.loopReplace([["#_IMG#", img], ["#_ALT#", alt],["#_CLASS#", clz], ["#_PROPS#", propsStr?propsStr:""]]);},
__renderIcon:function(img, alt, clz){return this.__renderAIcon(img,alt, clz, "");},
```

代码 3 客户端 KImage 模型真正的视图模型

上面是 KImage 模型所对应真正的视图。其中 pf 函数接收两个参数：一个是 Tview 模板中需要生成参数(KImage.tmpl)，第二个是数组([this.__renderIcon (this._model.getUrl()),this._id,fun("show")])，该数组中的值最终会替代第一个参数中的值。

2.3 两个模型的同步问题的解决

上面我们简单介绍了一个在客户端用 JavaScript 实现 MVC 的方法，并没有考虑与服务器交互的问题。在实际的应用程序中，由于业务流程的需要，我们需要在客户端编辑这些数据，即修改业务模型，然后就这些修改与服务器进行通信。但是，客户端和服务器端存在着的是两个业务模型的副本，它们可能失去同步，客户端浏览器并没有能力独立解决这个问题。这里，我们利用 Web Service，它使用 XML 作为通信语言，强调的是数据而不是内容，这对于 Ajax 应用是非常适合的，因为 Ajax 在更新页面时，不是向服务器发送整个页面进行更新。当 Ajax 客户端解析 XML 数据流的时候，根据 JS 类库里 ORM 的定义，生成客户端对象；而在向服务器提交新的请求时，我们也能够把需要修改的客户端需要传输的数据转化为 XML 数据流在网络上传输，那么在服务器端接收到 XML 字符串的时候会根据服务器端的 ORM 模板再反序列化为业务模型，使得两者之间保持了同步。

通过使用 XML 传输数据，使得服务器层和客户层都能理解传输数据格式，并且客户和服务器端的代

码就可以独立于对方而修改，做到了客户端模型可以和服务器端模型可以配合工作。

但是，该方法的主要局限是它将解析数据的负担完全交给了客户端。因此客户层的代码会变得更加复杂，如果在大型的应用中采用这种方法，则可以通过重用解析器代码或将一些功能抽象为库的方法来解决。

2.4 用 Ajax 与服务器交互

当需要保存用户在 wiki 页面上的一次修改或者提交新的请求时候触发 Ajax 请求，从客户端 JavaScript 代码角度来看，读取和更新的区别是细微的，我们仅仅需要指定使用 POST 方法，并且传入模型所对应的 XML 字符串就可以了。

Ajax 请求发生在需要和数据库(因此也跟服务器)通信的时候。如果 Controller 监听到绑定在 View 上的事件已触发，Controller 就开始与服务器交互，但是根据 MVC 法则，Controller 是通过 Model 与服务器通信的。如代码 4 所示：

```
Wiki.Controller.prototype = {
callServer:function(xmlData){this.model.callServer(
xmlData);},
}
```

代码 4 Controller 通过 Model 向服务器发送请求

Model 传递给服务器端的是一个 XML 字符串，这样在服务器接受到这个字符串之后可以直接根据定义的 ORM 模板反序列为服务器端的业务模型对象，实现了两个模型之间的同步通信。在服务器跟数据库通信完成之后，再次返回一个 XML 字符串用来更新客户端的界面。Ajax 让我们能够在发送请求的时候就指定回调函数，这样在服务器端进行处理的时候，不会阻止用户的下一步操作。当服务器返回处理之后的结果时，指定的回调函数会自动处理数据，更新用户界面。

3 结论

在上面的文章中，我们利用了 MVC 模式在客户端把用户界面、业务模型和事件处理分离成 3 个不同的部分，以降低应用程序模块之间的耦合性。但在某些情况下，分离可能是不需要的，甚至某些情况下，分离会造成很多不必要的程序冗长。而当我们的应用程序变得越来越复杂，需要 JavaScript 在客服端的多数部分的交互操作的时候，我们把 JavaScript 分离进入 MVC 模式能够产生出更多元化，更重复利用的代码。

参考文献

1 孙卫琴.精通 Structs 基于 MVC 的 Web 设计与开发.北京:电子工业出版社, 2004.

2 黎永良,崔杜武.MVC 设计模式的改进与应用.计算机工程,2005,31(9):96–98.

3 李园,陈世平.MVC 设计模式在 ASP.NET 平台中的应用.计算机工程与设计, 2009,30(13):3180–3184.

4 李学俊,李龙澍,徐怡.新一代网络语言 Wiki.计算机技术与发展, 2007,17(1):85–87.

5 Resion J.陈贤安,江疆译.精通 JavaScript.北京:人民邮件出版社, 2008.

(上接第 191 页)

the 10th International Conference on World Wide Web. New York: ACM Press, 2001.285–295.

6 Sarwar B. Sparsity, scalability and distribution in recommender systems [Ph.D. Thesis]. Minneapolis: University of Minnesota, 2001.

7 汪静,印鉴,郑利荣,黄创光.基于共同评分和相似性权重的协同过滤推荐算法.计算机科学,2010,37(2):99–103.

8 Linden G, Smith B, York J. Amazon.com recommendations: item-to-item collaborative filtering. IEEE Internet Computing, 2003,7(1):76–80.

9 张海鹏,李列彪,李仙等. 基于项目分类预测的协同过滤推荐算法.情报学报,2008,27(2):218–223.

10 龚瑞君,王佳,戴珺等.基于两阶段聚类的协作过滤推荐算法.郑州大学学报(理学版),2010,42(1):14–16.

11 李聪,梁昌勇,董珂.基于项目类别相似性的协同过滤推荐算法.合肥工业大学学报(自然科学版),2008,31(3):360–363.

12 邵伟,袁方,张瑜.融入项目类别信息的协同过滤推荐算法.数学的实践与认识,2010,40(6):108–112.

13 赵宏霞,杨皎平,陈宗娇.面向客户需求的神经网络挖掘方法.管理评论,2005,17(11):53–57.

14 马庆国.管理统计:数据获取、统计原理.SPSS 工具与应用研究.北京:科学出版社,2002.315–326.