长春理工大学
Changchun University of Science and Technology

硕士学位论文

基于 Ajax 的新 MVC 框架的研究与实现

研究生姓名：王　佳
学科、专业：计算机软件与理论

二〇〇九年三月

摘　　要

本文将 Ajax 技术与 MVC 模式紧密结合，设计了一个新的 MVC 框架。应用该框架后，客户端请求基本都是 Ajax 请求。本文首次采用了 Container 这一概念，并提出了 Container 树结构信息的表示法，设计了针对该表示法的解析及生成算法。与使用传统的 XML 格式或者 JSON 格式相比，新的树结构信息表示法将占用更少的存储空间，获得更高的解析效率。新框架的请求处理流程借鉴了 JSF 的请求生命周期，但对它进行了改造，从而简化了流程。本文还将 Java 反射机制与 Velocity 技术相结合，实现了在 VTL 文件中直接存取 Container 的相关属性。由于引入了 jQuery，极大简化了 DOM 以及 Ajax 编程。与 Struts 相比，应用新框架使开发人员不再需要编写配置文件，也不用考虑复杂的页面跳转；相比于 Brasato，则定制页面样式更加灵活。此外，经过改进后的新框架还支持集群环境。

关键字：Ajax　MVC　框架　Container　集群

ABSTRACT

In this paper, Ajax technology and MVC pattern are closely combined, a new MVC framework is designed. After using this framework, the client requests of a web application are almost Ajax requests. The concept of Container is proposed for the first time in this paper, the representation of Container tree structure is put forward and the analytical and generation algorithm for this representation is designed. Compared to the traditional XML format or JSON format, the new information representation of the tree structure occupys less storage space and obtains higher analysis of efficiency. In this paper, The request process of the new framework is designed by learnning JSF lifecycle which is more complex. In this paper, accessing related properties of Container directly from the VTL file is realized by combining Java reflection mechanism with Velocity. The new framework greatly simplifies the DOM and Ajax programming by drawing jQuery into it. Compared with Struts, developers don't need to write a profile and do not have to consider the complexity of the page jump when apply the new framework; compared to Brasato, customizing page styles are more flexible. In addition, being improved, the new framework also supports the cluster environment.

Key words: Ajax MVC framework Container cluster

长春理工大学硕士（或博士）学位论文原创性声明

本人郑重声明：所呈交的硕士（或博士）学位论文，《基于 Ajax 的新 MVC 框架的研究与实现》是本人在指导教师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的作品成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

作者签名：王佐 09 年 3 月 20 日

长春理工大学学位论文版权使用授权书

本学位论文作者及指导教师完全了解“长春理工大学硕士、博士学位论文版权使用规定”，同意长春理工大学保留并向中国科学信息研究所、中国优秀博硕士学位论文全文数据库和CNKI系列数据库及其它国家有关部门或机构送交学位论文的复印件和电子版，允许论文被查阅和借阅。本人授权长春理工大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，也可采用影印、缩印或扫描等复制手段保存和汇编学位论文。

作者签名：王佐 09 年 3 月 20 日

指导导师签名：[illegible] 2009 年 3 月 20 日

第一章 绪 论

1.1 本课题研究的目的和意义

回顾多个基于 J2EE 技术的项目，所使用的 MVC 框架都是清一色的 Struts，Struts 可以说是 MVC 经典设计模式中的一个经典产品，一度被认为是最广泛、最流行 Java 的 WEB 应用框架。但是随着 MVC 框架的广泛使用，Struts 的缺陷也越来越明显。每写一个 Action 类就要相应的在配置文件中写一个 action，还得事先规划好页面的跳转，之后还要在配置文件中为需要跳转的页面定义 forward。随着项目的不断推进，页面不断增加，Action 类也不断增加，配置文件也就越来越复杂，这对于后期的维护是十分不利的。通常为了追踪一次请求调用，需要在配置文件中不断定位 action、forward、actionform，由于配置文件的复杂，定位也就十分不便。事实上，Struts 就是将原本应该写在 web.xml 中的配置信息写在了 Struts 的配置文件中并提供了很多便于开发的工具。

偶然的机会，需要在项目中集成 OLAT（由瑞士苏黎世大学开发的一个基于 Web 的开源学习管理系统）。通过对 OLAT 的深入研究，发现 OLAT 所使用的 MVC 框架是苏黎世大学自己开发的，是基于组件的 MVC 框架，该框架目前已经是一个独立的开源 MVC 框架，即 Brasato。使用 Brasato 来开发 J2EE 项目可以像使用 Swing/AWT 来开发桌面应用程序一样，使用组件来构建界面，通过事件监听的方式来处理客户端请求。不需要使用配置文件，没有复杂的页面跳转。使用组件有一个很大的好处就是组件是可重用的，一个组件可以在多个页面中重复使用。Brasato 也有缺点，Brasato 的组件都是用 Java 类来编写的，在 Java 类中定义 HTML 标记，因此编写组件非常困难，通常只能使用框架自带的组件，因此，界面的样式也受到了限制。

综上，决定开发一个新的 MVC 框架，取两者之精华。此框架既不需要配置文件也不是完全的基于组件，它主要使用到两项技术：Ajax 与 Velocity。

Ajax 是一种创建互动式网页应用的网页开发技术[1]。这种技术的最大优点就是能在不更新整个页面的前提下维护数据。使用新的 MVC 框架所开发的应用，请求几乎都是 Ajax 请求，这使得 Web 应用程序能够更为迅捷的回应用户的动作。由于页面都是动态刷新的，会造成浏览器的后退按钮失效，这也正是新的 MVC 框架所带来的好处。在开发项目的过程中，发现用户经常会使用后退按钮来回到前一个状态，这种操作容易打乱正常的操作流程，造成莫名其妙的错误。使浏览器后退按钮失效能够有效避免这种错误的发生。

使用 Ajax 技术的另一个好处就是能够在客户端保存应用程序的状态。使用 Struts 开发的应用，通常需要将请求中的参数存储到请求对象中，然后再将请求对象中存储

的参数写回到页面上以便下一次请求时使用。这种操作方式不利于快速开发 Web 应用。如果使用 AJAX 就可以有效避免进行这种操作，只需要将请求中的参数保存到 Javascript 的变量中，后台程序就不需再重复保存这些参数。

Velocity 是一个基于 Java 的模版引擎，可以方便地集成到客户端或服务器端应用程序中。他允许使用者通过简单而强大的模版语言得到 Java 代码中定义的对象。当 Velocity 应用于 Web 开发，在 MVC 的开发模式下，Web 设计者和 Java 程序员可以并行的开发 web 站点。这意味着 Web 设计者可以将精力放在好的 Web 站点设计上，而 Java 程序开发者可以将精力放在编写高质量的代码上。Velocity 将 Java 代码从 Web 页面中分离出来，确保了站点在整个生命周期中更容易维护。相比于 JSP，提供了一种更有生命力的开发方式。

1.2 国内外研究现状

1.2.1 MVC 模式与 MVC 框架

MVC 模式最早是 Smalltalk 语言研究团提出的，应用于用户交互应用程序中。Smalltalk 语言和 Java 语言有很多相似性,都是面向对象语言。很自然地，SUN 在 petstore(宠物店)事例应用程序中就推荐 MVC 模式作为开发 Web 应用的架构模式。MVC 模式是一种架构模式，其实需要其他模式协作完成。在 J2EE 模式目录中，通常采用 service to worker 模式实现，而 service to worker 模式可由集中控制器模式、派遣器模式和 Page Helper 模式组成。Struts 框架只实现了 MVC 的 View 和 Controller 两个部分，Model 部分需要开发者自己来实现[2]。

国内也有不少文献对 MVC 模式进行了研究。其中多数讨论的是如何将 MVC 模式更好地应用于实际应用开发。也有文献提出了对 MVC 模式的改进。有的提出利用分层模型低耦合的特性改进 MVC 各个层之间的模糊分割，并提出将 MVC 模式分为五层：UI、Service、BO、DO、DA[3]。经过这种改进，不但实现了用户接口和功能模块之间的分离，而且各个部分之间的耦合度也进一步降低。

MVC 框架是对 MVC 模式的实现，并且这种实现的结果是一个中间件。如 Struts，它可以说是目前 J2EE 平台上最流行的 MVC 框架。

在 Struts 之后也出现了很多的 MVC 框架，如 WebWork、JSF、Tapestry 等。WebWork 与 Struts 相似，也是基于 service to worker 模式。WebWork 比 Struts 更加先进。WebWork 引入了前端拦截机（interceptor）以及 IoC（Inversion of Control 倒置控制）[4]容器等功能。目前 WebWork 已与 Struts 整合成立了一个新的项目 Struts2。JSF 和 Tapestry 都是基于组件和事件驱动开发模型的 MVC 框架。JSF 在很大程度上类似 Struts，而不是类似 Tapestry。JSF 只有在组件和事件机制这个概念上类似 Tapestry，Tapestry 则是一个完全组件的框架。

目前国内也出现了开源 MVC 框架，如 EasyJWeb。EasyJWeb 基于 Java 技术，框架充分借鉴了当前主要流行的开源 Web 框架（Struts、JSF、Tapestry 、Webwork），吸取了其优点及精华，利用 Velocity 作为模板页面引擎，是一个实现了页面及代码完全分离的 MVC 开发框架。

1.2.2 Ajax

Ajax 技术是目前在浏览器中通过 JavaScript 脚本可以使用的所有技术的集合。Ajax 并没有创造出某种具体的新技术，它所使用的大多数技术都是在很多年以前就已经存在了，然而 Ajax 以一种崭新的方式来使用所有的这些技术，使得古老的 B/S 方式的 Web 开发焕发了新的活力。

在 Ajax 出现以前，JacaScript、HTML/XHTML 和 CSS、DOM、XML 和 XSTL、XMLHttpRequest 基本上都是各自为政的，Ajax 让这些技术第一次有了交集，并且组成一个整体[5]。

Ajax 目前正处于一个快速的成长期，国内外很多 Web 应用开发都加入了 Ajax 的设计思想。现在，几乎所有的浏览器都提供了 Ajax 所需的技术，使用这种模式的富客户端应用程序也不断出现。而且相关 Ajax 的开源项目也有很快的成长，先后有 Rico，qooxdoo，AMOWA 等项目的兴起。

随着 Ajax 逐渐成为 Web 应用的主流开发技术，大量的业界巨头也已经采纳并且在大力推动这个技术的发展。如 IBM、Oracle、Yahoo!、BEA、RedHat、Novell 等业界领先的公司启动了 Open Ajax 项目，致力于为 Ajax 开发建造先进强大的的开发工具。IBM 已经发布了 Open Ajax 项目的 Ajax Toolkit Framework(ATF)，它是一个基于 Eclipse IDE 的 Ajax 开发工具。微软已经在 IE 中加入了 Ajax 所需的所有技术——DHTML、JScript 和 XMLHttp。在 ASP.NET 2.0 中，微软使用异步回调及舒适的 Ajax 风格使应用程序的编写更加简单，并且，微软为此提供了内建的控件。此外，微软还开发了 Ajax 框架 Altas，不过主要是和服务器端的 ASP.NET 框架配合工作。Sun 虽然行动迟缓，但是也将 Ajax 技术列入了 J2EE 的 blueprint(蓝图)中，作为 J2EE 技术的有益的补充。Google 在这方面的应用相对多一些，如 Google 讨论组、Google 地图、Google 搜索建议、Gmail 等。其实早在 1998 年该技术就得到了应用，但当时这项技术还不叫 Ajax。直到 2005 年初，许多事件使得 Ajax 被大众所接受。Ajax 这个词由《Ajax: A New Approach to Web Applications》一文所创，该文的迅速流传提高了人们使用该项技术的意识[6]。另外，对 Mozilla/Gecko 的支持，使得该技术更加成熟，更易于使用。

1.3 本文的主要工作和组织结构

由于经常在项目中使用 Struts 框架，对 Struts 的优劣有非常充分的了解。新的 MVC

框架也借鉴了 Struts 的一些优秀设计，如 DispatchAction 等。

新 MVC 框架的设计灵感主要来自 Brasato 开源框架。主要借鉴了应用该框架所开发的 Web 应用的文件组织结构，该框架对 Velocity 的应用以及对国际化和本地化的支持，等等。

新 MVC 框架首次采用了 Container 这一概念。在新 MVC 框架的设计及实现过程中一项十分重要的工做就是设计 Container 树结构信息的表示法，并基于该表示法设计 Container 树结构信息解析及生成算法。Container 树结构信息的表示法并没有采用传统的 XML 格式或者 JSON 格式，而是重新设计了一种信息表示格式。该格式相比于 XML 和 JSON 都更加高效。但是，该格式只适用于树结构。

此外，设计新 MVC 框架处理请求的流程也是一项非常重要的工作。新 MVC 框架处理请求的流程参考了 JSF 的请求生命周期，但对它进行了改造，简化了流程。这主要考虑到新 MVC 框架的架构设计以及处理请求的效率。

一个框架的好坏还有一个重要的评判标准就是它的易用性，新 MVC 框架在这方面也做了大量工作。

首先，引入了 Velocity 技术，并参考了 Brasato 的一些设计。当然，并没有采用将 Velocity 引擎初始化参数直接写入 Java 代码的方式，而是通过读取 Velocity 引擎的配置文件来初始化 Velocity 引擎。从而避免了更改 Velocity 引擎初始化参数后需要重新编译代码的弊端。

其次，将 Container 也加入 Velocity 的上下文中，并通过运用 Java 反射机制实现在页面文件中直接存取 Container 的相关属性。

最后，由于引入了 jQuery 开源 JavaScript 库，从而极大地简化了 DOM 编程以及 Ajax 编程。

本论文共分六章，各章组织如下：

第一章：阐述了本课题研究的目的、意义、研究现状，以及论文的组织结构。

第二章：对 Struts、Brasato 等 MVC 框架做了比较研究。

第三章：主要从使用者的角度阐述新 MVC 框架的结构，包括静态结构以及动态结构，提出了 Container 这一概念并设计了请求生命周期。

第四章：详述了新 MVC 框架的设计与实现。按照请求生命周期的顺序分阶段进行论述，详细讨论了视图层，对新 MVC 框架的性能做了较为全面的评价。

第五章：对 MVC 框架进行了扩展，使其能够支持国际化以及集群环境。

第六章：对全文工作进行总结，并提出了下一步需要解决的问题。

第二章 MVC 框架比较研究

2.1 MVC 模式在 J2EE 中的应用

MVC 模式并不能自动保证一个结构设计是正确的，如何在一个系统的设计中正确地使用 MVC 架构模式与系统所使用的技术有密切的关系。一般而言，一个 J2EE 系统应当适当地划分接收请求，根据请求采取行动，并将结果显示给用户等责任。流行的划分方式有 JSP 模型一和 JSP 模型二两种。这两种模型也是针对早期在 Web 开发中引入 MVC 时遇到的 Java 程序和 HTML 代码强耦合在一起，调试困难等问题所产生的两种解决方案。

JSP 模型一和 JSP 模型二本质区别在于处理用户请求的位置不同。

JSP 模型一又称做以 JSP 为中心的设计模型[7]。如图 2.1 所示，JSP 负责响应用户请求并将处理结果返回给用户。JSP 既要负责业务流程控制，又要负责提供表示层数据，同时充当视图和控制器，未能实现两个模块之间的独立和分离。但是，显示数据的逻辑和数据在这个模型中已有了一定程度的区分，所以说 JSP 模型一在一定程度上实现了 MVC。

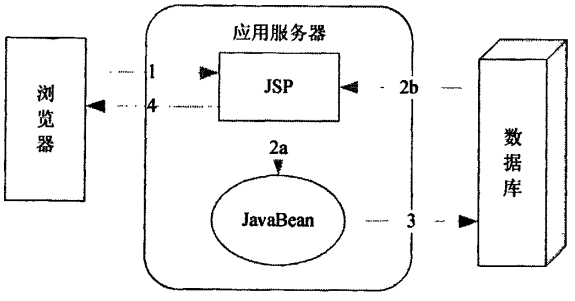


图 2.1 JSP 模型一

JSP 模型二又称做以 Servlet 为中心的设计模型[7]。如图 2.2 所示，JSP 模型二是一种联合使用 JSP 与 Servlet 来提供动态内容服务的方法。它吸取了 JSP 和 Servlet 两种技术各自的突出优点，用 JSP 生成表示层的内容，让 Servlet 完成深层次的处理任务。这里，Servlet 充当控制器的角色，负责处理用户请求，创建 JSP 页需要使用的 JavaBean 对象，根据用户请求选择合适的 JSP 页返回给用户。在 JSP 页内没有处理逻辑，它仅负责检索原来有 Servlet 创建的 JavaBean 对象，从 Servlet 中提取动态内容插入到静态模板。它清晰地分离了表达和内容，将显示数据的逻辑与商务逻辑分割开来，从而使系统的层次更加清楚，使视图和控制器两个模块可以独立演化。

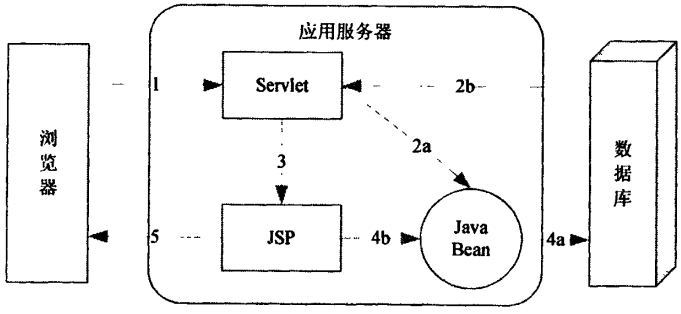


图 2.2 JSP 模型二

2.2 Struts

Struts 实质上就是在 JSP 模型二的基础上实现的一个 MVC 框架[8]。图 2.3 显示了 Struts 实现的 MVC 框架。

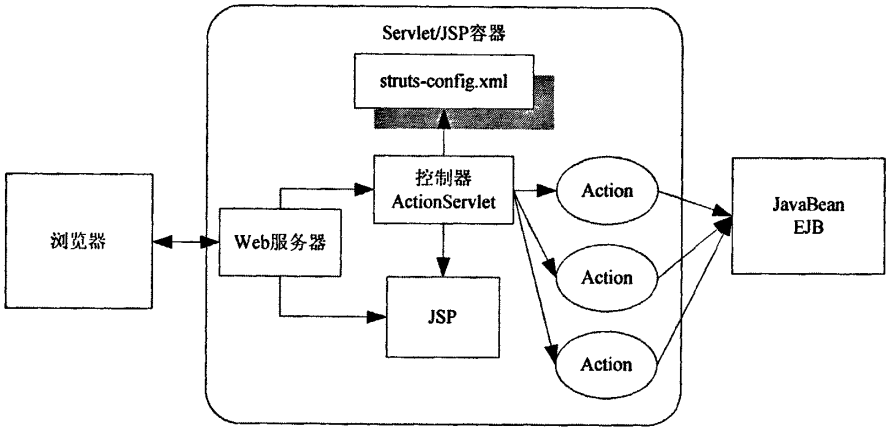


图 2.3 Struts 实现的 MVC 框架

在 Struts 框架中，视图由一组 JSP 文件构成，在这些 JSP 文件中没有业务逻辑，也没有模型信息，只有标签，这些标签可以是标准的 JSP 标签或客户化标签。此外，通常把 Struts 框架中的 ActionForm Bean 也划分到视图模块中。Struts 框架利用 ActionForm Bean 来进行视图和控制器之间表单数据的传递。

模型表示应用程序的状态和业务逻辑。对于大型应用，业务逻辑通常由 JavaBean 或 EJB 组件来实现。

控制器由 ActionServlet 类和 Action 类来实现。ActionServlet 类是 Struts 框架中的核心组件。ActionServlet 继承了 javax.http.HttpServlet 类，它在 MVC 模型中扮演中央控制器的角色。ActionServlet 主要负责接收 HTTP 请求信息，根据配置文件 struts-config.xml 的配置信息，把请求转发给适当的 Action 对象，如果该 Action 对象不存在，ActionServlet 会先创建这个 Action 对象。

Action 类负责调用模型的方法，更新模型的状态，并帮助控制应用程序的流程。对于小型简单的应用，Action 类本身也可以完成一些实际的业务逻辑。对于大型应用，Action 充当用户请求和业务逻辑处理之间的适配器，其功能就是将请求与业务逻辑分开，Action 根据用户请求调用相关的业务逻辑组件。业务逻辑由 Java Bean 或 EJB 来完成，Action 类侧重于控制应用程序的流程，而不是实现应用程序的逻辑。通过将业务逻辑放在单独的 Java 包或 EJB 中，可以提高应用程序的灵活性和可重用性。

2.3 Brasato

Brasato 框架主要是对两个设计模式的实现：MVC 模式和合成模式。

合成模式用于实现基于组件的架构。在 Brasato 中，组件非常类似 Java Swing 中的组件。实际上，整个 GUI 布局也借鉴了 Swing 架构的思想。在这样一个基于组件的架构中，控制器和视图元素都被分解到由组件和组件的容器（容器其实也是组件）对象组成的层次结构中。

组件是显示在界面中的视图元素。它由控制器生成，控制器负责视图的特定部分并且需要对组件产生的时间进行响应。组件是一个可重用的 GUI 元素。Brasato 提供的组件有表格、表单、树形菜单、链接等等。所有这些组件都是独立于业务的。它们的目的就是提供某种显示和操作数据的功能。

由于组件本身并不知道也无法告诉框架在页面的哪个位置显示自己，因此需要有容器。容器提供了容纳组件并确定它们在容器中的位置的功能。但容器也不知道它自己的位置因为它也是个组件。为此，Brasato 提供了一个非常重要的容器“Window”，它代表整个浏览器窗口。

Brasato 框架为开发人员处理了请求分派，提供了某种方式将业务过程分解成可易管理且易重用的控制器以及视图部分。使用 Brasato 框架，开发人员不再写 Servlet，而是写控制器（Controller）。一个控制器具有如下特性：

1. 实现业务逻辑。控制器需要对用户交互过程中发生的事件做出响应。如某个用户点击了一个链接，那么负责响应的控制器需要决定用户动作的意图并为这个事件决定触发哪个业务流程。
2. 视图的构建者。每个控制器都拥有一个视图，它就是一个用于显示控制器所关心的数据的组件。通过在控制器的构造方法中调用 setInitialComponent 方法来构建一个视图。
3. 实现了合成模式。一个控制器也可以有多个子控制器。
4. 管理数据模型。数据模型由控制器初始化并传给组件用于显示数据。
5. 可重用性。由于使用了合成模式，控制器被组合成一个层次结构，越高层的控制器越难重用，而越低层的控制器重用性越好，因为它们的业务相关性越弱。在 Brasato 中，GroupController、UserSearchController、TableController 就经常被重用。

控制器与视图之间的关系如图 2.4 所示。

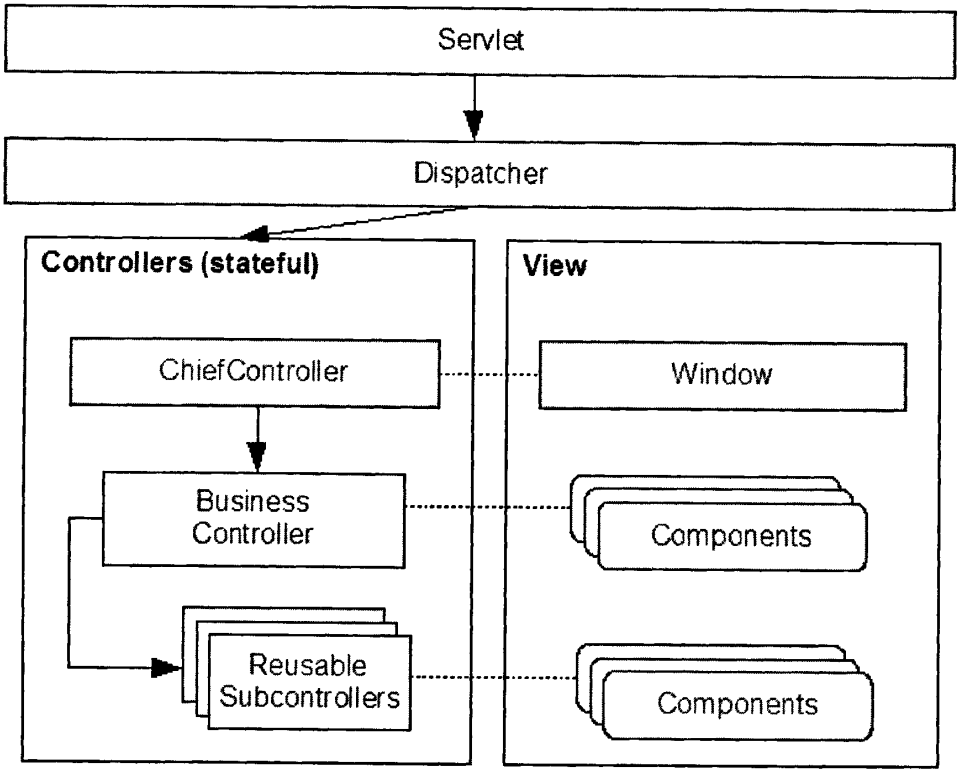


图 2.4 控制器与视图关系图

第三章 新 MVC 框架的结构

本章主要从使用者的角度论述了新 MVC 框架的静态结构和动态结构。静态结构包括页面结构、文件结构；动态结构主要是指请求生命周期。

3.1 页面结构

3.1.1 常用的页面的布局方式与实现方法

在实际的应用开发中，页面通常会被分成多个区域。如图 3.1 所示，这是一种比较常见的页面区域划分方式。Header 部分显示的可能是系统的名称，还可能包括使用系统的用户名以及退出系统的链接等。Menu 部分显示的是系统的菜单，系统的功能将在 Menu 部分得到体现。Content 部分是主要操作区，系统的绝大部分功能将在 Content 部分实现。

以上 3 个部分中，Header 与 Menu 部分的内容相对稳定，即在整个系统的运行过程中，Header 与 Menu 的内容很少发生变化。而 Content 部分的内容则经常改变。

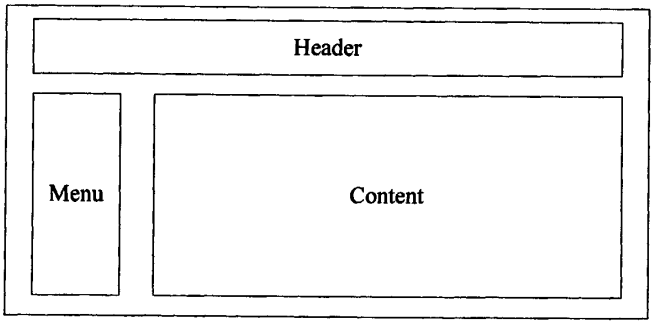


图 3.1 页面区域图

目前有多种途径可以实现图 3.1 所示的页面区域划分方式。如应用 JSP 的<jsp:include>标签、HTML 的<frameset>与<frame>标签、Ajax 等。

如果使用的是 JSP，通常 Header 部分用一个 JSP 文件，Menu 部分用一个 JSP 文件，Content 部分则需要多个 JSP 文件，在 Content 的每个 JSP 文件中使用<jsp:include>标签来引用 Header 的 JSP 文件与 Menu 的 JSP 文件。

如果使用<frameset>与<frame>标签，则需要构建一个框架页面。框架页面将一个浏览器窗口分割成多个小窗口，每个小窗口就是一个 frame，frame 引用一个独立的页面。<frameset>标签用于对 frame 布局。图 3.1 的框架页面代码类似如下：

```
<frameset rows="160,*">
  <frame src="..." name="Header" scrolling="yes" />
```

```
<frameset cols="160,*">
  <frame src="..." name="Menu" scrolling="yes"/>
  <frame src="..." name="Content" scrolling="yes"/>
</frameset>
</frameset>
```

<frameset>的 rows、cols 属性用于定义框架的布局方式，rows 为横向布局、cols 为纵向布局。<frameset>中还可以嵌套<frameset>，这样可以支持复杂的框架布局。<frame>标签的 src 属性定义了所引用文件的 URI。

将 Content 部分置入一个 frame 中可以使 Content 改变时不用重新加载 Menu 与 Header 部分，减少了网络传输的数据量。但这种方式使得当某个 frame 的内容改变时，浏览器地址栏的值不会发生变化。因此，如果 frame 中有元素使用相对 URI 来引用资源，则其 URI 不能参照浏览器地址栏中的地址，这有时会给定义页面的跳转带来一些麻烦。

Ajax 方式则是一种比较新的方式，尽管 Ajax 的相关技术基本都是老技术，但应用 Ajax 还是比较新的。此时，主页面只需要做成普通的 HTML 页面，可以使用 或者等标签来标记 Header、Menu 以及 Content 部分的位置。由于 Header 与 Menu 部分相对稳定，因此在主页面中也可以直接写入这两部分的内容，而只对 Content 部分使用特殊标签标记。

应用 Ajax 后，页面内容的改变不需要重传整个页面，因此减少了网络传输的数据量。页面内容的改变同样不会导致浏览器地址栏的值发生变化，但与使用框架页面不同的是，页面中任何元素的相对 URI 可以参考浏览器地址栏中的地址。此外，页面内容的改变会使浏览器的前进后退按钮失效，要想回到之前某个状态，必须编写相关的链接或按钮。这样做看似会增加开发的工作量，但实际上这是一个好的现象，会强制开发人员不依赖浏览器自带的导航功能——此功能有时会打乱正常的操作流程，从而为最终用户带来更好的操作体验。

3.1.2 新 MVC 框架的页面布局方式与实现方法

新 MVC 框架可以支持 3.1.1 小节介绍的常用页面布局方式，而且主要是用 Ajax 技术实现。

新 MVC 框架中有一个非常重要的概念——Container——它实际上是指 Container 类型的对象。页面是由 Container 组织起来的。如果用 Container 来构建 3.1.1 小节的图 3.1 所示的页面，则 Header、Menu 以及 Content 可以各用一个 Container 来表示，这三个 Container 需要被组织到一个大的 Container 中。Container 被组织成一个树形结构，最上层的 Container（即根 Container）代表整个页面。一个页面只有一个根 Container，每个 Container 都可以包含任意数量的 Container。Container 的组织结构图如图 3.2 所示。

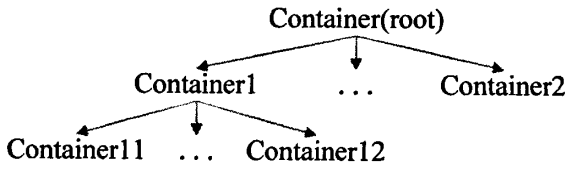


图 3.2 Container 组织结构图

Container 的组织结构与页面区域划分的方式是对应的。一个页面可以被划分成若干个区域，每个区域又可以再被分成若干子区域，可以不断重复这一划分过程。如果最终用户点击页面中的某个链接或者某个按钮，则会改变页面中某些区域的内容，反映到 Container，则是替换 Container 树中的某些 Container，有时甚至是替换根 Container。

页面内容是 Container 树在某一时刻的快照，Container 树结构改变，页面内容也跟着变化。还是以 3.1.1 小节的图 3.1 为例。假设代表整个页面的 Container 为 main；代表 Header 部分的 Container 为 header；代表 Menu 部分的 Container 为 menu；代表 Content 部分的 Container 为 form 与 table。form 是一个显示表单的 Container，table 是一个显示表格的 Container。

在时刻 A，Container 树的结构如图 3.3 所示，此时页面的 Content 部分显示的是一个表单。在时刻 B，假设用户点击了某个链接（这个链接可能位于 Header、Menu 或者 Content 部分中）。此时，浏览器向服务器发送一个请求，导致 Container 树结构发生变化。新的 Container 树的结构如图 3.4 所示。Container 树结构的改变将导致服务器向浏览器发送响应，响应的内容是 table 所表示的信息，即一个表格。反映到页面上，其结果是 A 时刻页面中 Content 部分的表单被表格所替换。上述过程中，浏览器向服务器发送请求与页面内容的改变均有 Ajax 技术实现。

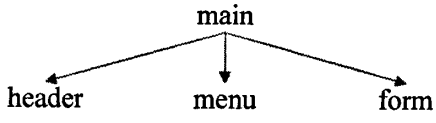


图 3.3 时刻 A 的 Container 树结构图

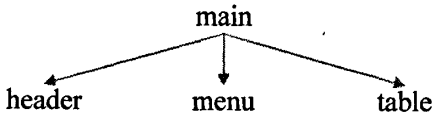


图 3.4 时刻 B 的 Container 树结构图

3.1.3 改变 Container 树结构

新 MVC 框架使用合成（Composite）模式来设计 Container。合成模式属于对象的结构模式。合成模式将对象组织到树结构中，可以用来描述整体与部分的关系[9]。合成模式可以使客户端将单纯元素与复合元素等同看待。合成模式是一个处理对象的树结构模式，因此适合用于设计 Container。

树结构被分为三种：从上向下、从下向上和双向的[10]。在三种有向树图中，树的节点和它们的相互关系都是一样的，但是连接它们的关系的方向却很不一样。

由上向下的树的树图如图 3.5 所示。每一个树枝节点都有箭头指向它的所有子节点，从而一个客户端可以要求一个树枝节点给出所有的子节点，而一个节点却并不知道它的父节点。在这样的树结构上，信息可以按照箭头所指的方向自上向下传播。

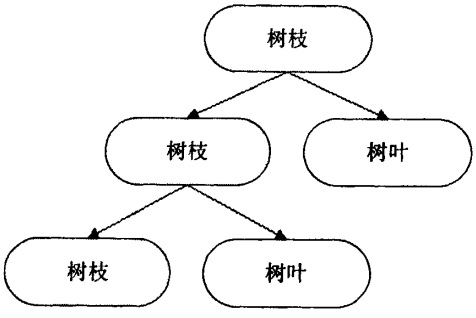


图 3.5 有上向下的树图

由下向上的树的树图如图 3.6 所示。每一个节点都有箭头指向它的父节点，但是一个父节点却不知道其子节点。信息可以按照箭头所指的方向自下向上传播。

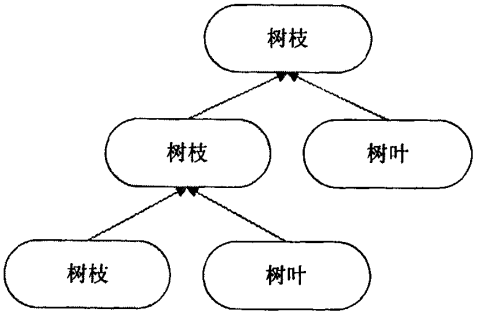


图 3.6 有下向上的树图

双向树的树图如图 3.7 所示。每一个节点都同时知道它的父节点和所有的子节点。在这样的树结构上，信息可以按照箭头所指的方向向两个方向传播。

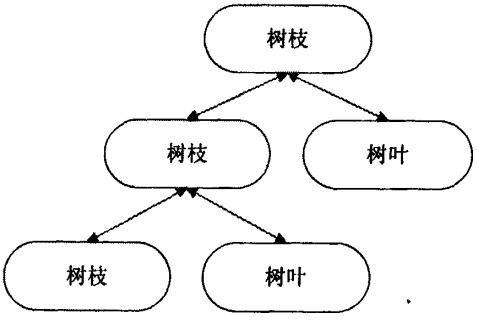


图 3.7 双向树图

新 MVC 框架使用的是双向树结构来组织 Container。因为使用双向树结构，获取每个节点的子节点与父节点都同样容易。尽管双向树结构需要更多的存储空间，但是

访问子节点与父节点的简便性更重要。因为使用 MVC 框架的目的就是要简化开发，如果访问子节点或父节点很复杂，则改变 Container 树结构就会很困难，这是不利于提高开发效率的，而且容易出错。

合成模式的类图如图 3.8 所示。

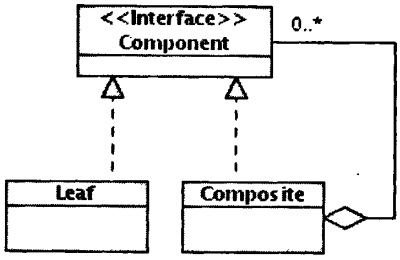


图 3.8 合成模式类图

可以看出合成模式的类图结构涉及到三个角色：抽象构件（Component）、树叶构件（Leaf）和树枝构件（Composite）。Container 扮演的正是树枝构件的角色，实际上，并不严格区分树叶构件或树枝构件。Container 类的实例在某一时刻是树叶构件，在另一时刻则可能是树枝构件，反过来也是成立的。简单起见，可以不需要抽象构件角色。经过改造后的 Container 类的类图如图 3.9 所示。

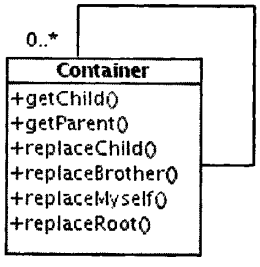


图 3.9 Container 类的类图

在图 3.9 中，没有了抽象构件和树叶构件，只有一个树枝构件 Container。在新 MVC 框架中，Container 是一个抽象类。Container 拥有获得某个子节点的 getChild 方法，获得父节点的 getParent 方法，修改某个子节点的 replaceChild 方法，修改某个兄弟节点的 replaceBrother 方法，修改自身的 replaceMyself 方法以及修改根节点的 replaceRoot 方法。应用这些方法就可以很方便地修改 Container 树结构。

3.2 文件结构

本节主要介绍应用新 MVC 框架后，模型层、视图层、控制器层的代码文件都是如何组织的。

3.2.1 符合 J2EE 规范的 Web 应用程序的文件结构

图 3.10 所示的是一个符合 J2EE 规范的 Web 应用程序的文件结构。

图 3.10 文件结构图

上图中的 WebContent 文件夹是 Eclipse 集成开发环境为 Web 应用自动生成的。该文件夹中的文件（夹）都是 Web 应用最终需要的。其中有两个非常关键的文件夹：META-INF 与 WEB-INF。

META-INF 虽然关键，但一般不用开发人员关心，它通常可以由工具自动生成，如 Ant 等。

WEB-INF 文件夹中的 classes 文件夹用于存储当前 Web 应用需要的 class 文件，即由 Java 源代码编译后的文件。lib 文件夹存储的是一些 jar 文件，这些 jar 文件相当于当前 Web 应用的类库。web.xml 文件用于配置当前 Web 应用所用到的 Filter、Servlet 等构件。

除了 classes、lib、web.xml 这 3 个文件（夹）外，WEB-INF 文件夹中还可以包含其他的文件（夹），许多 MVC 框架通常会选择在 WEB-INF 文件夹中建立配置文件，如 Struts 的 struts-config.xml 文件。

WebContent 文件夹中与上述两个文件夹处在相同层次的文件（夹）通常是当前 Web 应用的页面文件。如 css 文件夹中存储的是 CSS 文件，login.jsp 则是一个 JSP 文件。

3.2.2 应用新 MVC 框架的 Web 应用的文件结构

应用新 MVC 框架的 Web 应用的文件结构与 3.2.1 小节的图 3.10 所示的文件结构最大区别在于 login.jsp 之类的页面文件已不在 WebContent 文件夹中。当然，这种区别不是绝对的。因为一个 Web 应用可以使用不止一个 MVC 框架，就算只使用新的 MVC

框架，如果有需要，也可以在 WebContent 文件夹中添加 JSP 之类的页面文件。因此，只能说应用新 MVC 框架可以实现不在 WebContent 文件夹中建立页面文件。

应用新 MVC 框架的 Web 应用的页面文件通常与 Java 类文件存储在一起，即存储在 classes 文件夹中。classes 文件夹的文件结构如图 3.11 所示。

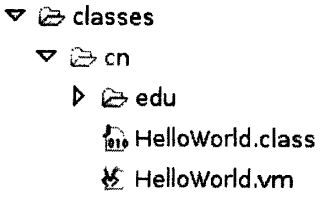


图 3.11 文件结构图

上图中 HelloWorld.class 是 Container 子类的类文件。一个 Container 子类一般对应一个页面文件。页面文件名与 Container 子类的类名相同。页面文件必须同与其对应的 Container 子类的类文件在同一个文件夹中。上图中 HelloWorld.vm 文件就是页面文件，它所对应的 Container 子类为 HelloWorld。从页面文件的扩展名不难看出，页面文件是用 Velocity 模板语言编写的。

新 MVC 框架要求所有与 Container 子类对应的页面都必须使用 Velocity 模板语言编写。之所以选择 Velocity，首先是因为它简单。Velocity 的语法很简单，学习起来比较容易。用 Velocity 模板语言编写的页面也很简洁，例如“$!test”这一语句，如果用 Java 代码来表达，则要写成“test==null?””:test.toString()”。

其次，Velocity 完全实现了 Java 代码与页面代码分离。当然，在 JSP 页面中也能不出现 Java 代码，但 JSP 并没有从技术上保证不能出现 Java 代码。

Velocity 还支持在集群环境上发布应用。这一点已经过实践检验。笔者所参与开发的某个系统，其页面基本使用 Velocity 模板语言编写，目前在 IBM Websphere 集群环境上运行良好。

3.3 请求生命周期

新 MVC 框架分 4 个阶段来处理一次客户端请求：重建 Container 树、应用请求值、调用应用、呈现响应。

3.3.1 重建 Container 树

一棵 Container 树对应一个页面，重建 Container 树的目的就是要了解用户当前所操作的页面的内容和结构。

这是一个非常重要的步骤。因为如果没有 Container，则后续的处理都无法进行。此外，Container 的信息并不保存在服务器端——这样会占用大量的服务器内存的存储

空间。因为，每个用户所操作页面的内容和结构可能都不相同，因此，在服务器端保存每个用户所操作的页面的内容和结构是不现实的。

新 MVC 框架是将用户的 Container 树结构的信息存储在客户端，当客户端发送请求时，必须将 Container 树结构的信息发往服务器，然后服务器根据 Container 树结构的信息在内存中为当前请求重新构建 Container 对象树。

新 MVC 框架为每个 Container 子类定义了一个 ID，它是一个唯一的标识码，是一个数字。客户端存储的 Container 树结构的信息是一个字符串，该字符串中记录的是 Container 树中每个 Container 子类的 ID。

以 3.1.1 小节的图 3.1 所示的页面为例。假设代表整个页面的 Container 子类的 ID 为 1，代表 Header 部分的 Container 子类的 ID 为 2，代表 Menu 部分的 Container 子类的 ID 为 3，代表 Content 部分的 Container 子类的 ID 为 4。则客户端存储的 Container 树结构的信息为“1(2,3,4)”，该字符串体现了 Container 树结构中节点（Container 子类）之间的父子关系。如果某个 ID 后面紧跟着“(”，则说明该 ID 所代表的节点含有子节点，子节点为“(”和与其配对的“)”之间的所有 ID 所代表的节点。如字符串“1(2,3,4)”中“1”后面紧跟着“(”，且“（”和与其配对的“)”之间的所有 ID 为“2”、“3”、“4”，则表明“1”所代表的节点是“2”、“3”、“4”所代表的节点的父节点，“2”、“3”、“4”所代表的节点是“1”所代表的节点的子节点。

仍以 3.1.1 小节的图 3.1 所示的页面为例。假设用户点击了 Menu 部分的某个链接，则浏览器需要将 Container 树结构的信息作为请求参数发送给服务器，参数值为“1(2,3,4)”。服务器获取 Container 树结构的信息后，为 ID 为 1 的 Container 子类新建一个实例 main，为 ID 为 2 的 Container 子类新建一个实例 header，为 ID 为 3 的 Container 子类新建一个实例 menu，为 ID 为 4 的 Container 子类新建一个实例 content。然后将 header、menu、content 加入 main 的集合属性中。至此，在内存中就构建了一棵 Container 的对象树，重建 Container 树完毕。

3.3.2 应用请求值

该阶段是将请求中的参数复制到某个 Container 子类的实例中。因此，该阶段分为两步，首先是获取某个 Container 子类的实例，然后再复制请求参数。

由此提出一个新的问题，即获取哪个 Container 子类的实例。首先，获取的 Contianer 子类的实例必须是重建的 Container 树中的某个节点。其次，该实例必须与发起请求的页面区域有关。综上，该实例就是与发起请求的页面区域直接对应的 Container 子类的实例。

以 3.1.1 小节的图 3.1 所示的页面为例。假设某用户点击了 Menu 区域的某个链接。然后服务器端为这一请求重建 Container 树，树结构如图 3.12 所示。

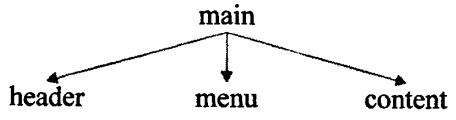


图 3.12 Container 树结构图

上图中 main 为代表整个页面的 Container 子类的实例，header 为代表 Header 部分的 Container 子类的实例，menu 为代表 Menu 部分的 Container 子类的实例，content 为代表 Content 部分的 Container 子类的实例。

接着，获取的 Container 子类的实例是 menu。因为，menu 是重建的 Container 树中的某个节点，而且，发起请求的页面区域是 Menu——请求是由用户点击 Menu 区域的某个链接产生的，而 Menu 区域对应的 Container 子类的实例是 menu。

服务器端已经知道获取的 Container 子类的实例必须对应于发起请求的页面区域，那么服务器端又是如何知道发起请求的页面区域呢？这是客户端告诉服务器端的。由于每个 Container 子类都有一个唯一编号 ID，则页面上每个区域也都有一个编号 ID，当用户点击页面某个区域的链接或者按钮时，客户端会向服务器端发送该页面区域的 ID，服务器端根据该 ID 就可以确定发起请求的页面区域。

有了 Container 子类的实例，下一步就是将请求中的参数复制到实例中。这一步相对比较简单，就是一个同名拷贝。假设实例中有个名为 test 的属性，请求中有个名为 test 的请求参数，则将 test 请求参数的值作为实例中 test 属性的值。这一复制过程有一个关键的问题需要解决，就是类型转换。因为请求参数的值都是字符串，而实例属性的类型则有多种，包括字符串、整数、浮点数、日期等数据类型。因此需要将字符串转换成相应的数据类型，这将在第四章中做具体论述。

这一阶段的主要作用也是为了简化开发，将请求参数直接复制到 Container 子类实例中可以使开发人员不需再操作 javax.servlet.http.HttpServletRequest 类型的对象，转而直接操作 Container 子类实例的属性，而且这些属性的值基本不再需要进行数据类型转换。

3.3.3 调用应用

此阶段需要完成对系统业务层的访问。几乎每一次用户请求都是为了完成一项业务，因此 MVC 框架必须提供某种访问业务层的机制。新 MVC 框架通过 Container 子类的实例来访问业务层。

在请求生命周期的第二阶段已经获取了某个 Container 子类的实例，此实例对应于发起请求的页面区域。在调用应用阶段，需要调用此实例的 event 方法来完成对业务层的访问，event 方法有一个参数用于标识事件源。由于一个页面区域可能包含多个链接和按钮，每个链接或按钮要完成的业务通常是不同的，而且它们都在同一个页面区域中，无论点击哪个链接或按钮，都将调用同一个 Container 子类实例的 event 方法，因

此必须有一种方法能够知道用户点击的是哪个链接或按钮，以便能够访问业务层的不同业务。通过在 event 方法中包含一个标识事件源的参数就能够确定用户点击的是哪个链接或按钮。事件源参数实际上是一个字符串，它是链接或按钮的名称。新 MVC 框架要求为页面中的每个链接或按钮命名。

访问业务层的机制借鉴了观察者模式，观察者模式定义了一种一对多的依赖关系，让多个观察者对象同时监听某一个主题对象[12]。这个主题对象在状态上发生变化时，会通知所有观察者对象，使它们能够自动更新自己。观察者模式的类图如图 3.13 所示。

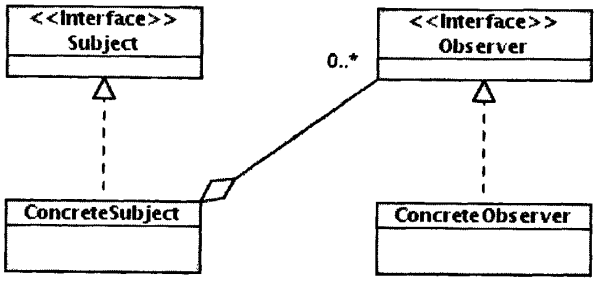


图 3.13 观察者模式结构图

在新 MVC 框架中，对观察者模式做了简化，即删除了抽象主题（Subject 接口）。具体主题（ConcreteSubject 类）角色由 LevinServlet 类扮演。抽象观察者（Observer 接口）是 Container 类，它是一个抽象类。具体观察者（ConcreteObserver 类）是 Container 类的子类，由应用开发人员编写。具体主题与抽象观察者之间并不是一对多的关系，而是一对一的关系，因为用户只可能点击一个链接或按钮。经过改造后的类图如图 3.14 所示。

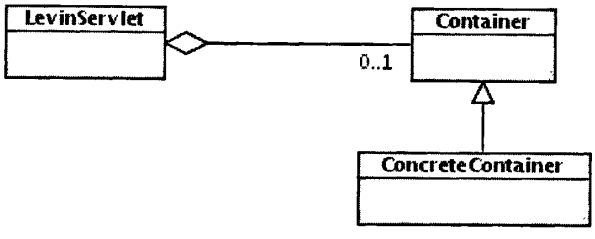


图 3.14 经过改造后的类图

3.3.4 呈现响应

执行完某项业务之后，通常需要修改 Container 树结构。这一阶段的主要任务就是将新的 Container 树结构所对应的页面信息返回给客户端。最终，用户将看到页面被更新。

现在需要做一个选择，是将整个新 Container 树结构所对应的页面信息都返回到客户端还是只将原 Container 树结构中被更新的节点所对应的页面信息返回客户端。答案是后者。因为前者通常会传输更多的信息，页面更新的时间更长，也没能充分体现 Ajax

局部刷新页面的优势。

那么，是否需要知道原 Container 树结构中所有被更新的节点呢？答案是否定的。实际上，所有被更新的节点会组成一个森林，因此，只需要知道这个森林中每一棵树的根节点即可。如图 3.15 所示是一个 Container 树。

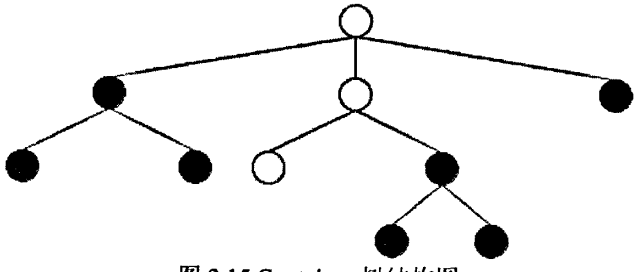


图 3.15 Container 树结构图

在上图中每一个圆圈代表 Container 树中的一个节点，实心圆表示的是被更新的节点，所有实心圆组成一个森林，如图 3.16 所示。

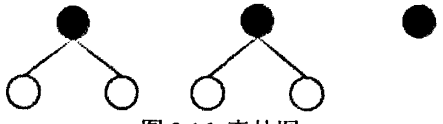


图 3.16 森林图

上图中森林的每棵树的根节点用实心圆表示。在呈现响应过程中只需要知道图 3.16 中每个实心圆所代表的 Container 子类的实例。然后，生成这些实例所对应的页面信息，最后将页面信息返回客户端浏览器。返回客户端的信息中，除了要包含页面信息外，还得包含页面信息所对应的 Container 子类的 ID，只有这样，客户端才能知道新的页面信息需要替换哪些旧的页面信息。至此，整个请求生命周期结束。

第四章 新 MVC 框架的设计与实现

本章主要论述新 MVC 框架中的模型层、视图层以及控制器层的设计与实现。主要涉及的类有 LevinServlet（新 MVC 框架的控制器）、ContainerManager（负责管理 Container 树）、Container（新 MVC 框架的核心，属于模型层）、RendererManager（负责管理 Renderer，属于视图层）以及 Renderer（负责呈现响应，是视图层的核心）。以上 5 个类之间的关系如图 4.1 所示。

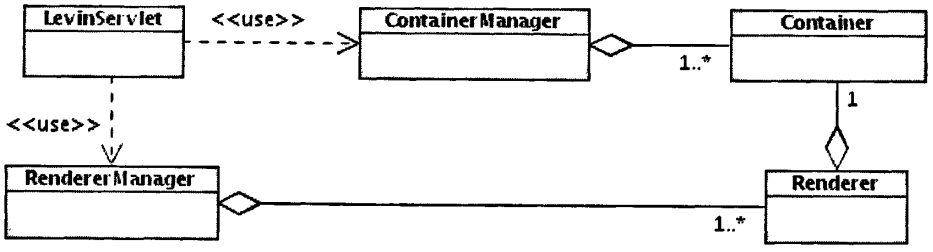


图 4.1 主要类关系图

上图中 LevinServlet 与 ContainerManager、RendererManager 之间都是依赖关系。ContainerManager、RendererManager 这两个类型的对象都是临时构建的，即每一次请求都要新建这两个类的实例，在请求结束后由 Java 虚拟机负责回收这两个类的实例。由于 ContainerManager 与 Container 之间是聚合关系，RendererManager 与 Renderer 之间也是聚合关系，因此 Container、Renderer 这两个类型的对象的生命周期将分别由 ContainerManager、RendererManager 这两个类型的对象决定。

4.1 请求生命周期

本节的结构与 3.3 节的结构类似。论述了请求生命周期各个阶段的设计与实现。请求生命周期各个阶段的流程是由 LevinServlet 来控制。其流程如下：

Step1：从请求中获取 Container 树结构信息。

Step2：如果不存在 Container 树结构信息，则新建事先定义好的主 Container 类的实例，然后执行 **Step8**。

Step3：否则，利用 Container 树结构信息重建 Container 树。

Step4：从请求中获取发起请求的页面区域的 ID。

Step5：如果该 ID 不存在，则向客户端输出错误信息，结束请求生命周期。

Step6：否则，利用该 ID 获取 Contaienr 树中的某个 Container 子类的实例，将请求参数复制到该实例的与参数名同名的属性中（应用请求值）。

Step7：向上一步获取的实例发送消息来调用应用。

Step8：利用新的 Container 树来呈现响应，结束请求生命周期。

上述流程的第二步说明用户是第一次访问某 Web 应用，请求并不是由在页面上点击某个链接或按钮触发。在这种情况下，系统会构建一个默认的 Container 树，然后跳过重建 Container 树、应用请求值、调用应用这 3 个阶段，直接将默认的 Container 树作为新 Container 树来呈现响应。

4.1.1 重建 Container 树

客户端传给服务器端的 Container 树结构信息是一个包含 Container 子类 ID 的字符串。因此需要一种能够将 Container 子类 ID 映射成 Contaienr 子类的机制。新 MVC 框架提供了一个类——ContainerPool 来负责从 ID 到 Container 子类的映射。

ContainerPool 类的实例是一个全局对象，它是 LevinServlet 的一个属性。ContainerPool 类有一个名为 getContainer 的实例方法，其参数为 Container 子类的 ID，返回值是 Container 子类的对象。ContainerPool 类还有一个名为 containers 的实例属性，它是 java.util.ArrayList 类型的对象，它存储的是 Container 子类的类全名（包名加上类名）。ArrayList 类型的对象是一个可变数组，里面的每个元素通过索引值来存取，实际上 Container 子类的 ID 就是数组元素的索引值。

ContainerPool 类的 getContainer 方法所执行的操作可以描述如下：

Step1：判断 ID 值是否小于 contaienrs 中的元素个数。

Step2：如果不是则抛出不存在 Container 子类的异常。

Step3：否则，调用 containers 的 get 方法，将 ID 值作为参数。get 方法返回的将是 Container 子类的类全名。

Step4：利用 Container 子类的类全名来构建 Container 子类的实例，并将该实例赋给变量 container。

Step5：将 ID 值复制到 container 的 id 属性中。

Step6：将 container 作为方法的返回值返回。

如果执行了上述步骤中的 Step2，则会抛出 Java 异常，此时后续的步骤都不再执行，getContainer 方法结束。Step4 利用了 Java 的反射机制来构建对象——Java 反射机制允许使用 Java 类的类名来新建对象[16]。ContainerPool 类的实例除了负责将 Container 子类 ID 映射到 Container 子类外，还有一个责任，即生成 Container 子类的 ID。这部分内容将在 4.1.4 小节介绍。

至此，已经有方法通过 ID 值来获取 Container 子类的实例，下一步就是解析 Container 树结构信息的字符串，并生成 Container 对象树。

通过对 Container 树结构信息的字符串深入分析发现，如果从左至右逐个字符地扫描一遍字符串，相当于对 Container 对象树做了一次先根序遍历。如“1(2(3),4)”这一 Container 树结构信息的字符串，其树结构如图 4.2 所示。如果对其从左至右逐个字符地扫描一遍，将得到的顺序是 1、2、3、4，这是采用先根序遍历的结果。

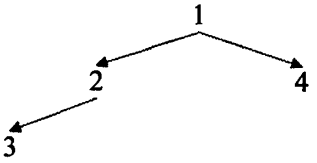


图 4.2 Container 树结构图

先根序遍历有一个特点，就是父节点总是出现在子节点的前面，某个节点的任意子孙节点不会出现在它兄弟节点的后面[17]。也就是说，对任意的先根序顺序，如$(a_0,a_1,\ldots,a_n)$，假设 a_i 是含有子节点的节点，则 a_{i+1} 必为子节点，且 a_i 的所有子节点是$(a_{i+1},a_{i+2},\ldots,a_k)$，其中 k<=n。

现在，关键的问题就是如何确定 a_i 和 a_k。通过对字符串的分析不难看出，“(”之前紧挨着“(”的 ID 必是 a_i，即含有子节点的节点。“(”同与其配对的“)”之间的所有 ID 必是 a_i 的子节点，此外不再有 ID 会是 a_i 的子节点。

至此，已经可以很容易地实现在从左至右扫描字符串的同时重建 Container 对象树，字符串扫描完毕，Container 对象树构建完毕。具体步骤如下：

Step1：设置三个变量，branch（java.util.Stack 类型的对象，是一个堆栈，用于存储所有带有子节点的 Container 类型的对象）、id（int 数据类型，用于记录 ID，初始值为-1）、root（Container 类型的对象，是方法的返回值，初始值为 null）。

Step2：依次取出字符串中的每个字符赋给变量 c。

Step3：如果 c 是数字（0 到 9 之间的字符），执行 **Step31**。

Step4：否则，如果 c 是左括号（字符“(”），执行 **Step41**。

Step5：否则，如果 c 是右括号（字符“)”），执行 **Step51**。

Step6：否则，如果 c 是逗号（字符“,”），执行 **Step61**。

Step7：否则，抛出 Java 异常，声明出现非法字符。

Step8：如果 root 为空，则抛出 Java 异常，声明字符串中左右括号个数不等。

Step9：否则，返回 root，方法结束。

Step31：c -= '0'。执行这一步主要是因为 c 是 ASCII 编码，需要将其转换为数字。

Step32：如果 id 等于-1，则将 c 值赋给 id。

Step33：否则，先将 id 乘 10，再执行 id += c。之所以这么做，是因为有的 ID 可能大于等于 10，如 12。那么在扫描字符串的过程中是先得到 1 再得到 2，为了最终得到 12，需将 1 乘以 10 再加上 2。

Step34：执行 Step2。

Step41：如果 id 等于-1，抛出 Java 异常，声明“(”之前无数字。

Step42：利用 id 值构建 Container 类型的对象，赋给变量 container。

Step43：如果 branch 不为空，则获取栈顶元素（但不删除栈顶元素），赋给变量 parent，并将 container 作为 parent 的子节点。

Step44：将 container 压入 branch。

Step45：id = -1。执行 **Step2**。

Step51：获取栈顶元素（删除栈顶元素），赋给变量 parent。

Step52：如果 id 不等于-1，则利用 id 值构建 Container 类型的对象并赋给变量 container。将 container 作为 parent 的子节点。执行 id = -1。

Step53：如果 branch 是空，则将 parent 赋给变量 root，执行 **Step8**。

Step54：否则，执行 **Step2**。

Step61：如果 id 等于-1，则执行 **Step2**。

Step62：获取栈顶元素（但不删除栈顶元素），赋给变量 parent。利用 id 值构建 Container 类型的对象，赋给变量 container。将 container 作为 parent 的子节点。执行 id = -1。

Step63：执行 Step2。

上述步骤即 ContainerManager 类的实例方法 buildTree 的执行流程。该方法接收一个参数 tree，它是 String 类型的对象，即 Container 树结构信息的字符串。该方法的返回值是 Container 树结构的根节点，是 Container 类型的对象。

4.1.2 应用请求值

3.3.2 小节中提到，应用请求值的第一步是获取 Container 树中的某个 Container，该 Container 的 ID 也是以请求参数的形式传给服务器端。

在重建 Container 树阶段返回的是 Container 树的根节点。因此，可以通过遍历这棵树的每一个节点，比较节点的 ID 值与请求参数中的 ID 值，就可以确定当前需要操作的节点。但新 MVC 框架并没有这么做，因为通过研究发现，完全可以在重建组件树的过程中就确定要操作的节点，所以没有必要再重新遍历一次 Container 树。

为此，需要对 ContainerManager 类的实例方法 buildTree 做一下改造，即为方法添加一个参数 cid，它是 int 类型的参数，代表了请求参数中的 ID 值。此外，还需要在 ContainerManager 类中添加一个实例属性 current，该属性是 Container 类型，用于引用 Container 树中 ID 为请求参数中的 ID 的节点。

buildTree 方法中被改造的步骤有 Step42、Step52 以及 Step62。在“利用 id 值构建 Container 类型的对象并赋给变量 container”操作之后，需要比较 id 值与 cid 的值。如果两个值相等，则将 container 值赋给 current 属性。

经过这些改造，在重建 Container 树之后就可以调用 ContainerManager 的实例方法 getCurrentContainer 来获取需要操作的 Container，该方法只是将 current 属性值返回。

下一步，就是将请求中的其他参数设置到需要操作的 Container 中。这一过程比较简单，主要使用了 Apache 的 commons-beanutils 工具，该工具提供一种方法，能够将 Map 对象中的值设置到 JavaBean 中。

Map 中存储的元素都会有一个 key，通过 key 可以很方便的查找 Map 中的元素。JavaBean 的代码通常有这么一个规律，就是对于 JavaBean 中的某个私有实例属性，如

果想要对其存取，则需要定义一对方法分别用于设置属性值与读取属性值。假设有一个名为 bean 的 JavaBean，其代码如下：

```
public class BasicInfo {
    private java.lang.String name;// 私有属性

    public java.lang.String getName() { return name; }// 获取属性值
    public void setName(String name) { this.name = name; }// 设置属性值
}
```

初始状态下，bean 的 name 属性值为 null。现在假设有一个名为 map 的 Map，其中有个 String 类型的元素，值为“test”，对应的 key 为 String 类型的对象，其值是“name”。当调用 BeanUtils 的 populate 方法（BeanUtils 是 Apache 的 commons-beanutils 工具中的一个类，populate 方法接收两个参数：一个是 Map 类型，一个是 Object 类型）后，bean 的 name 属性值为“test”，这样就实现了将 Map 中的值设置到 JavaBean 中。

BeanUtils 的功能不止这些。它不但可以复制普通类性的属性，实际上任何类型的属性都可以复制。此外 BeanUtils 还可以自动进行类型转换（仅限于普通类型）。具体方法可以参考 commons-beanutils 工具的相关文档。

有了 BeanUtils 这样一个方便的类，复制请求参数就很简单了。只要调用如下代码就可以将请求参数复制到 Container 中：

```
BeanUtils.populate(container, request.getParameterMap());
```

以上代码的第一个参数是要操作的 Container，第二个参数中的 request 是 javax.servlet.http.HttpServletRequest 类型的对象，该对象提供一个实例方法 getParameterMap 用于将请求参数以 Map 的形式返回——HTTP 请求中的每个参数实际都是 key/value 对，这与 Map 存储数据的形式是极其相似的。

4.1.3 调用应用

调用应用实际上就是调用要操作的 Container 的 event 方法，该方法有两个参数：LevinContext 类型的对象、event 事件源。

LevinContext 类型的对象对以下 2 个取自 Web 容器的对象进行了封装：javax.servlet.ServletRequest、javax.servlet.ServletContext。

选用 Struts 作为 Web 应用的 MVC 框架时，如果需要操作 HttpSession 对象，首先要调用 HttpServletRequest 对象的 getSession 方法获得 HttpSession 对象，然后再调用 HttpSession 对象的 getAttribute 方法或 setAttribute 方法。

选用新 MVC 框架时，操作 HttpSession 对象就没这么麻烦，只要直接调用 LevinContext 对象的 setSessionAttr 方法或 getSessionAttr 方法即可，省去了获取 HttpSession 对象这一步骤。此外，如果要操作 HttpServletRequest 对象，则可直接调用 LevinContext 对象的 setReqAttr 方法（调用此方法相当于调用 HttpServletRequest 对象

的 setAttribute 方法)、getReqAttr 方法(相当于调用 getAttribute 方法)、getPara 方法(相当于调用 getParameter 方法)以及 getParaValues(相当于调用 getParameterValues 方法)方法。如果要操作 ServletContext 对象，则可直接调用 LevinContext 对象的 setAppAttr 方法(调用此方法相当于调用 ServletContext 对象的 setAttribute 方法)与 getAppAttr 方法(相当于调用 getAttribute 方法)。

可见，LevinContext 可以使得开发人员不用直接操作 HttpServletRequest、ServletContext 以及 HttpSession 这三个对象，这种间接操作的方式更加简便。

event 事件源实际上就是一个字符串，它标识了发起请求的页面区域的某个链接或者按钮。当用户点击某个链接或按钮时，会向服务器端发送该链接或按钮的标识——它是一个字符串，由开发人员为链接或按钮定义的。

如果某个页面区域的链接或按钮的数量比较多，则可能会在 event 方法中写入较多的判断语句，这是非常繁琐的。新 MVC 框架借鉴了 Struts 框架的 DispatchAction，引入了 DispatchContainer。假设某个页面区域有两个链接，分别被标识为"link1"和"link2"。那么只需要在 DispatchContainer 的子类中添加名为 link1 的方法和 link2 的方法，而不再需要重写 event 方法，link1 与 link2 方法的参数列表与 event 方法的参数列表相同。当用户点击标识为"link1"的链接时，将执行 link1 方法；点击标识为"link2"的链接时，将执行 link2 方法。DispatchContainer 应用了 Java 的反射机制——允许在运行时通过方法名来调用对象的某个方法。

4.1.4 呈现响应

这一阶段需要解决的一个重点问题是，如何确定 Container 树中哪些节点被更新了。这其实是两个问题，即如何判定某个节点是否被更新以及哪些节点被更新。第一个问题比较好解决。因为每个 Container 都有 id 属性，在第一阶段结束时，Container 树中的每个节点的 id 属性值必定是大于等于 0 的整数。那么只要规定在新建 Container 时，将 id 属性值置为-1，则如果某个节点的 id 属性值为-1，就可以认定该节点是新的 Container。该方案的实现方法是在 Container 的初始化方法中将 id 置为-1，具体代码如下：

```
public abstract class Container {
    int id;
    …
    public Container() { id = -1; }
    …
}
```

解决第二个问题需要对 Container 树结构做更深入的分析。在 3.3.4 小节曾介绍过 Container 树结构中被更新的节点会组成一个森林。之所以会有这么一个判断，是因为如果 Container 树结构中某个节点被更新了，那么它自身以及它的所有子节点的 id 值肯

定为-1，它自身和它的所有子节点组成了一棵树。因此，所有 id 值为-1 的节点要么是组成一棵树，要么就是多棵树。不论是一棵树还是多棵树都可以被称作森林。

通过分析还发现，并不需要找出所有被更新的节点，而只需要找出所有被更新的节点组成的森林中每棵树的根节点。因为，实际上在呈现响应时只需要呈现根节点所代表的页面区域，这个页面区域包含了所有子节点的页面区域。寻找所有根节点的步骤如下：

Step1：设置一个集合变量 roots（java.util.ArrayList 类型的对象，是一个可变数组，用于存储所有被更新的节点的根节点）作为方法的返回值。

Step2：取出 Container 树的根节点赋给变量 node。

Step3：执行 **Step31**。

Step31：如果 node 的 id 属性值为-1，则将 node 加入 roots。

Step32：否则，取出 node 的所有孩子节点（这些节点的深度均为 node 的深度加 1），对每个孩子节点，执行 Step31。

上述步骤即 ContainerManager 类的实例方法 getNewRoots 的执行流程。Step31、Step32 可以写入另一个实例方法中，该方法会被递归调用。可以肯定，递归过程必定会结束。因为递归调用的方法要么结束在 Step31，要么因为 node 是叶子节点而结束。如果判断某个节点被更新了，则不再搜索其所有子节点。这样递归过程最终找到的将是所有被更新节点的根节点。

下一步，要构建新 Contaienr 树结构信息的字符串，该字符串将在下一次用户请求时被用于重建 Container 树。

在构建之前需要设置所有被更新节点的 id 值，即将 id 值为-1 的节点的 id 值置为大于等于 0 的某个整数。在 4.1.1 小节曾介绍过 ContainerPool 类的实例有一个责任是生成 Container 子类的 ID。也就是说，ContainerPool 类有一个实例方法 getID，它接收 Container 子类的类名作为参数，返回 Container 子类的 ID，这个 ID 就是节点的 id 值。ContainerPool 类有一个名为 containerMap 的实例属性，它是 java.util.HashMap 类型的对象，它存储的值是 Container 子类的 ID，关键字是 Container 子类的类名。getID 方法的执行步骤如下（设方法的参数名为 classname）：

Step1：设置变量 id，它是 Integer 类型。

Step2：为 containerMap 对象加锁。

Step3：调用 containerMap 对象的 get 方法，参数为 classname，将返回值赋给变量 id，id 是 Integer 类型。

Step4：如果 id 不为空，则执行 **Step8**。

Step5：否则，调用 containers（ContainerPool 类的实例属性，在 5.1.1 小节有介绍）的 size 方法，利用返回值构建 Integer 类型的对象并赋给变量 id。

Step6：将 id 加入 containerMap，关键字为 classname。

Step7：将 classname 加入 containers。

Step8：释放 containerMap 对象的锁。

Step9：调用 id 的 intValue 方法，并将返回值作为 getID 方法的返回值返回。

上述步骤中之所以要对 containerMap 对象加锁，是因为该对象与 containers 对象都是共享资源（一个 Web 应用就一个 ContainerPool 类的实例），各个线程需要互斥地访问共享资源。Step5 说明 Container 子类的 ID 实际上就是数组元素的索引值。

现在可以开始构建新 Container 树结构信息的字符串了。具体步骤如下：

Step1：设置一个变量 result（java.lang.StringBuffer 类型的对象，用于存储新 Container 树结构信息的字符串）作为方法的返回值。

Step2：取出新 Container 树的根节点赋给变量 node。

Step3：执行 **Step31**。

Step4：如果 result 中最后一个字符为“,”，则将其删除。

Step31：如果 node 的 id 属性值为-1，则调用 ContainerPool 类的实例的 getID 方法，参数为 node 的类名，将返回值置为 id 属性值。

Step32：将 node 的 id 属性值加入 result。

Step33：如果 node 是叶子节点，则往 result 中加入字符“,”。

Step34：否则，往 result 中加入字符“(”。取出 node 的所有孩子节点（这些节点的深度均为 node 的深度加 1），对每个孩子节点，执行 **Step31**。将 result 中最后一个字符（该字符必为“,”）改为“)”。

上述步骤即 ContainerManager 类的实例方法 buildTreeInfo 的执行流程。Step31 至 Step34 可以写入另一个实例方法中，该方法会被递归调用。可以肯定，递归过程必定会结束。因为递归调用的方法必定结束在 Step33，即因为 node 是叶子节点而结束。

至此，最后的工作就是呈现响应。这个过程需要将所有被更新节点的每个根节点封装成 Renderer 类型的对象，将所有 Renderer 对象加入一个集合由 RendererManager 对象管理。RendererManager 会调用每个 Renderer 的 render 方法来生成页面信息。最终返回客户端的信息除了每个 Renderer 的信息外还有一些附加信息。呈现响应的具体过程将在 4.2 节做详细介绍。

4.2 视图层

视图层负责呈现响应。呈现响应分为两个阶段：服务器端生成页面信息并传到客户端，客户端刷新页面。第一个阶段由 RendererManager 与 Renderer 共同完成。第二个阶段则需要用 Javascript 实现。

4.2.1 Velocity

应用新 MVC 框架开发的 Web 应用的页面都是使用 VTL（Velocity 模板语言）编

写的。VTL 使用起来非常简单，使用 VTL 编写的页面代码也很简洁。如下是一个用 VTL 编写的页面代码：

```
#set($hello = "您好")
<html>
<body>
$hello $!{customer.name}!##实际上调用了 customer 的 getName 方法
以下是您所购买的产品：
<table>
#foreach($product in $products)
#if($customer.hasPurchased($product))
<tr><td>$product</td></tr>
#end
#end
</table>
</body>
</html>
```

上述代码基本包含了 VTL 的常用元素，如注释、设置变量语句、引用变量、引用属性、引用方法、循环语句、判断语句等。使用 VTL 最重要的是记住以下几条规则：

1. 以“##”开头的是单行注释；多行注释则是以“#*”开头，以“*#”结尾。
2. 以“#”开头的是语句，如设置变量语句“set($变量名 = 值)”、循环语句“foreach($集合中每个元素的变量名 in $集合变量名)”、判断语句“if(...)”等。
3. 以“$”开头的是引用。引用分为变量引用、属性引用、方法引用。变量引用可以指向 Java 代码中的定义内容，如上述代码中的“customer”、“products”、“product”，或者由 Web 页面中的 VTL 语句来获得值，如“hello”，它是由“set($变量名 = 值)”语句定义的。属性引用的格式类似“$变量名.属性名”，如“$!{customer.name}”，它实际上通过调用 customer 的 getName 方法来获取 name 属性值。方法引用的格式类似“$变量名.方法名(参数列表)”，参数列表可以是变量引用、属性引用或方法引用等，如“$customer.hasPurchased($product)”。

使用 Velocity，只有 VTL 文件是不够的，必须得有 Java 代码。Java 代码负责初始化 Velocity 引擎，设置 Velocity 上下文（类似 HashMap），将 VTL 文件、Velocity 上下文以及某个输出流连接。配合上述 VTL 的 Java 代码如下：

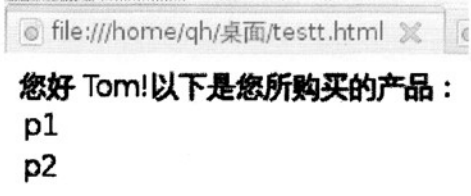
```
public class Customer {
    private String name = "Tom";

    public String getName() { return name; }
    public boolean hasPurchased(String product) {
        if (product.equals("p1") || product.equals("p2")) { return true; }
        return false;
    }
    public static void main(String[] args) throws Exception {
```

```
        // 初始化 Velocity 引擎
        Properties p = new Properties();
        p.load(new FileInputStream("/home/qh/velocity.properties"));
        VelocityEngine ve = new VelocityEngine();
        ve.init(p);
        // 设置 Velocity 上下文
        VelocityContext vc = new VelocityContext();
        vc.put("products", new String[] { "p1", "p2", "p3" });
        vc.put("customer", new Customer());
        StringWriter sw = new StringWriter();
        // 合并 VTL 文件与 Velocity 上下文
        ve.mergeTemplate("cn/Customer.vm", vc, sw);
        System.out.println(sw.toString());
    }
}
```

执行上述 Java 代码，可以得到如下输出：

```
<html>
<body>
您好 Tom!以下是您所购买的产品：
<table>
<tr><td>p1</td></tr>
<tr><td>p2</td></tr>
</table>
</body>
</html>
```

将以上输出保存成 HTML 文件，用浏览器打开，可以得到如图 4.3 所示的效果：



4.3 浏览器显示效果图

4.2.2 应用 Velocity

本小节主要涉及三个类：VelocityHelper、ChainedContext、VelocityWriter。

4.2.2.1 VelocityHelper

VelocityHelper 的责任就两个：初始化 Velocity 引擎，将 VTL 文件、Velocity 上下文以及某个输出流连接。

VelocityHelper 类的实例将在 LevinServlet 初始化时被构建，并且 LevinServlet 的一

个实例属性会引用该实例，这样可以保证 Web 应用运行过程中只初始化一次 Velocity 引擎。

4.2.2.2 ChainedContext

ChainedContext 是 VelocityContext 的子类，引入该类的主要目的是要改写 VelocityContext 的 internalGet 方法，以便开发人员能够在 VTL 页面中直接使用未被置入 VelocityContext 的对象，这些对象包括 Container、HttpServletRequest、HttpServletResponse、HttpSession 以及 ServletContext。

ChainedContext 类有六个实例属性，除了 Container、HttpServletRequest、HttpServletResponse、HttpSession、ServletContext 以外，还有一个 HashSet 类型的实例属性，其作用是存储 Container 子类中的所有实例属性。ChainedContext 类的实例方法 internalGet 的代码如下：

```
public class ChainedContext extends VelocityContext {
    public Object internalGet(String key) {
        Object o = null;
        // 判断是否引用了 Contaienr 的实例属性
        if(fields.contains(key)){
            String method = "get"+key.substring(0,1).toUpperCase()
                            +key.substring(1);
            try{
                Method m = container.getClass()
                    .getMethod(method, new Class[]{});
                return m.invoke(container, new Object[]{});
            }catch(Exception ex){
            }
        }
        // 判断是否引用了 request、response、session、application
        if (key.equals(REQUEST)) { return request; }
        else if (key.equals(RESPONSE)) { return response; }
        else if (key.equals(SESSION)) { return session; }
        else if (key.equals(APPLICATION)) { return application; }

        o = super.internalGet(key);
        if (o != null) { return o; }
        return getAttribute(key);
    }
    ...
}
```

上述代码中的第一个判断语句表示如果引用了 Contaienr 的实例属性，则直接调用

Container 的获取实例属性值的方法将实例属性值返回。第二个至第五个判断语句比较简单，就是用于返回 request 或 response 或 session 或 application（ServletContext 类型的对象）。

有了 ChainedContext，开发人员如果想在某 Container 子类所对应的页面中引用 Container 子类的实例属性，则直接使用"$实例属性名"即可。如 Container 子类中有一个名为 username 的实例属性，并且有一个 getUsername 的不带参数的实例方法，则在该 Container 子类所对应的页面中用"$username"就可以使用 username 属性。此外，开发人员可以在页面中用"$request"来使用 HttpServletRequest，用"$response"来使用 HttpServletResponse，用"$session"来使用 HttpSession，用"$application"来使用 ServletContext。

4.2.2.3 VelocityWriter

呈现响应有时不止呈现一个页面区域，有可能会有多个页面区域，因为所有被更新的 Container 可能组成多棵树（每棵树的根节点对应一个页面区域）。服务器端是将所要呈现的所有页面区域的信息一次性发给客户端的。因此必需得有一种机制能够从页面信息中区分不同的页面区域。一种比较好的方法是在发送的页面信息中附加一些信息，附加信息包括每个页面区域的字符个数。附加信息的格式为"页面区域 1 的字符个数:页面区域 2 的字符个数:..."，如"1000:500:3000"，该附加信息表示有 3 个页面区域，每个页面区域的长度分别为 1000 个字符、500 个字符和 3000 个字符。

页面区域的个数比较好统计，它就是调用 ContainerManager 的 getNewRoots 方法所返回的集合中元素的个数。比较难统计的是页面区域的字符个数。当然，有一种方法，就是生成页面区域的信息，它通常是一个字符串，然后调用字符串对象的 length 方法。但这么做会占用大量的系统存储空间，不可取。因此，还是需要从生成页面信息的过程中寻找解决方案。由于使用的是 Veloctiy 来生成页面信息，而 Velocity 生成页面信息的方式就是将 VTL 文件、Velocity 上下文以及某个输出流连接，写成 Java 代码，就是调用如下的代码：

```
ve.mergeTemplate(template, ctx, out);
```

上述代码中的 ve 是 Velocity 引擎，template 是 VTL 的文件名，ctx 是 Velocity 上下文，out 是某个输出流。至于 mergeTemplate 方法是如何将 template、ctx、out 三者连接不用关心，关键是 out 为最终结果。out 是 java.io.Writer 类型的对象，通常就是往客户端传输信息的输出流。因此可以构建一个 Writer 类的子类，将输出流包装到该类的对象中，然后将这个对象作为 mergeTemplate 方法的输出流。Velocity 引擎必定调用 Writer 的 write 方法、或 flush 方法。因此 Writer 的子类只要重写这两个方法，只要 Velocity 引擎调用它们，就记录输出的字符个数。

VelocityWriter 就是 Writer 的子类，该类除了记录输出字符的个数外，还充当了输出流缓冲器的作用。该类有几个实例属性：writer（被包装的输出流）、bufferSize（缓冲区大小）、cb（字符数组，即缓冲区）、nextChar（缓冲区中的字符个数）、totalChar（输出的字符个数）。该类的 write 实例方法如下：

```
public final void write(int c) throws IOException {
    if (bufferSize == 0) {// 如果没有设置缓冲区
        writer.write(c);
        totalChar++;// 输出字符个数自增
    } else {
        if (nextChar >= bufferSize) { /*刷新缓冲区*/}
        cb[nextChar++] = (char) c;// 将字符加入缓冲区
        totalChar++;// 输出字符个数自增
    }
}
```

上述代码中的注释部分已经对该方法做了很详细的说明。总之，可以通过访问 VelocityWriter 的 totalChar 实例属性来获取输出的字符个数。

4.2.3 Renderer

Renderer 负责构建某个页面区域。由于一个页面区域对应一个 Container，因此该类需要有一个属性引用 Container。Renderer 的构造方法如下：

```
public class Renderer {
    private Container container;

    Renderer(Container container){
        this.container = container;
    }
    ...
}
```

构造方法非常简单，就是设置 container 实例属性的值。Renderer 有 2 个名为 render 的实例方法。以下将对这 2 个方法分别加以介绍，第一个方法的代码如下：

```
public void render(VelocityWriter writer, HttpServletRequest request,
        HttpServletResponse response, ServletContext application) {
    this.request = request;
    this.response = response;
    this.application = application;
    this.writer = writer;
    ChainedContext cc = new ChainedContext(container, request, response,
            application);
    cc.put("r", this);
    VelocityHelper vh = VelocityHelper.getInstance();
```

```
        vh.merge(Util.getClassVelocityPath(container.getClass()), cc,
                writer);
}
```

该方法是 Renderer 最关键的一个实例方法，该方法实现了对页面区域的输出。该方法的前 4 条语句用于将方法中的参数保存到实例属性中。第 5 条语句用于构建 ChainedContext，ChainedContext 的作用在 4.2.2.2 已做了介绍。第 6 条语句非常关键，有了这条语句就可以在 VTL 文件中引用 Renderer（提供这个功能的原因将在下文介绍）。之后的两条语句就是将 VTL 文件、Velocity 上下文以及某个输出流连接。

提供在 VTL 文件中引用 Renderer 的功能是因为一个 VTL 文件可能会引用多个 VTL 文件，这和一个 Container 可能包含多个 Container 是对应的，反映到页面上则是一个页面区域可能由多个页面区域组成。例如，有一个名为 parent 的 Container 包含一个名为 child 的子 Container，parent 所对应的 VTL 文件代码如下：

```
<table>
<tr><th>#*这是表头*#</th></tr>
<tr><td>$r.render('child')</td></tr>
</table>
```

上述代码中通过调用 Renderer 的 render 方法引用 child 所对应的 VTL 文件，该 render 方法只接收一个字符串类型的参数，方法的代码如下：

```
public void render(String name) {
    try {
        writer.write("<span id=\"" + name + "\">");
        Container child = container.getChild(name);
        Renderer r = new Renderer(child);
        r.render(writer, request, response, application);
        writer.write("</span>");
    } catch (Exception ex) { }
}
```

上述代码的第一条和最后一个条语句将 child 所对应的页面区域用标签包围，该标签有一个 id 属性，其值是 child，即 Container 的名称。该标签的作用是用于标记一个页面区域，浏览器不会对该标签应用任何样式。第 2 条语句用于取出 Container 中的某个 Container，在本例中是 child。第 3 步是新建一个 Renderer，接着调用新 Renderer 的 render 方法，并将当前 Renderer 的 writer、request、response、application 作为方法的参数。至此，就将 child 所对应的页面区域引入 parent 所对应的页面区域。如果 child 里还有 Container，假设该 Container 名为 grandson，则在 child 所对应的 VTL 文件中还可以继续使用“$r.render('grandson')”。

4.2.4 RendererManager

该类负责管理一个由 Renderer 组成的集合并依次调用每个 Renderer 的 render 方法

（参数最多的那个方法）。当然，RendererManager 并不是简单地将所有的 Renderer 的 render 方法调用一遍就结束了，而是在每执行完一次 render 方法之后都要获取输出流已输出的字符个数，用于在输出信息的最后输出附加信息（附加信息在 5.2.2.3 有介绍）。为此，RendererManager 需要一个实例属性 totalCharArray，用于存储每个 Renderer 所输出的字符个数，该属性是整型数组类型。RendererManager 输出信息的过程如下（假设 Renderer 的集合名为 roots，它是 java.util.ArrayList 类型的对象）：

Step1：设置变量 i，它是 int 类型，初始值为 0。构建一个整型数组，数组长度为 roots 中元素的个数，将该数组赋给实例属性 totalCharArray。

Step2：调用 HttpServletResponse 对象的 getWriter 方法获取输出流，将该输出流包装成 VelocityWriter 类型的对象并将该对象赋给变量 writer。

Step3：如果 i 大于等于 roots 中元素的个数，执行 **Step10**。

Step4：否则，调用 roots 的 get 方法，参数为 i，将返回值赋给变量 root（它是一个 Renderer）。

Step5：往 writer 中输出 root 所包装的 Container 的名称。

Step6：往 writer 中输出一个字符":"该字符用于分割 Container 的名称，和 Container 所对应页面区域的信息。

Step7：调用 root 的 render 方法，参数为 writer、request、response、application。这一步实际上就是往 writer 中输出 Container 所对应页面区域的信息。

Step8：调用 writer 的 getTotalChar 方法获取已输出的字符个数，将该值赋给 totalCharArray[i]。

Step9：调用 writer 的 setTotalChar，参数为"0"，将已输出的字符个数清零。将 i 值自增，执行 **Step3**。

Step10：设置变量 i，它是 int 类型，初始值为 totalCharArray 中元素个数减一，即 totalCharArray.length-1。

Step11：如果 i 小于 0，则执行 **Step14**。

Step12：否则，往 writer 中输出字符":"，再输出 totalCharArray[i]。

Step13：将 i 值减 1，执行 **Step10**。

Step14：结束。

以上步骤从 Step10 开始是输出附加信息。最终输出到客户端的信息类似"Container1 名称:...Container2 名称:...:Container2 信息字符个数:Container1 信息字符个数"，其中的"..."表示 Container 所对应页面区域的信息，在 Container1 所对应页面区域信息与 Container2 名称之间并没有任何分隔符，也不需要任何分隔符，因为在附加信息里标明了每部分的字符个数。需要注意的是，附加信息与主信息（附加信息之前的信息）之间的分隔符是":"，附加信息中 Container 信息字符个数的顺序是主信息中 Container 顺序的逆序。

4.2.5 jQuery

新 MVC 框架的请求基本都是 Ajax 请求，因此发送请求，刷新页面几乎都会使用到 JavaScript 语言。为了使 DOM 和 Ajax 编程更加简单，新 MVC 框架应用了 jQuery。jQuery 是一款免费且开放源代码的 JavaScript 代码库，它有助于简化 DOM 以及 Ajax 编程，并能帮助开发人员保证代码简洁易读。

假设要获得 id 属性为“test”的文档元素（文档元素在 HTML 文件中就是指某个标签，每个文档元素通过 id 属性值来唯一标识）。如果使用传统的 JavaScript，代码如下：

```
var test = document.getElementById("test");
```

如果使用 jQuery，则代码如下：

```
var test = $("#test");
```

对比不难看出，使用 jQuery 代码简单了许多。其实，不单是代码简单了，效率也提高了。首先是传输效率提高，因为更少的代码量将减少传输的信息量，最终获得更短的响应延迟。其次，jQuery 对 DOM 解析做了很好的优化，使得通过 jQuery 来访 DOM 元素的速度要比使用传统的 JavaScript 方式快许多。

使用 JavaScript 的时候，大部分时间都是对 DOM 元素进行操作。除了通过元素的 id 属性值来查找元素外，还可通过元素的类型来查找元素。如要查找所有的链接元素，然后为它们绑定一个单击事件，在该事件中执行相同的方法。如果使用传统的 JavaScript，必须先获得所有的链接类型的元素，然后挨个为元素绑定事件，而使用 jQuery，则代码如下：

```
$("a").click(function() {
    //方法体
});
```

上述代码通过“$("a")”就可以获得所有链接类型的元素，然后调用 click 方法，参数是一个函数，这样就将某个函数绑定到了所有的链接元素的单击事件。

使用 jQuery 将使得 Ajax 变得极其简单。如果需要把一些参数传递给服务器中的某个页面。可以使用“$.post()”或者“$.get()”，前者对应 POST 请求，后者对应 GET 请求。如果要编写复杂的 Ajax 脚本，那么需要用到“$.ajax()”函数，具体用法如下：

```
$.ajax({
    url: 'test.jsp',
    type: 'POST',
    data: 'key1=value1&key2=value2',
    timeout: 1000,
    error: function(){
        alert('载入失败！');
    },
    success: function(xml){
        // 执行相关处理
    }
```

});

上述代码中 ajax 方法的参数是一个对象。url 属性指定了请求的地址；type 属性定义了请求的类型；data 属性则是请求参数，请求参数的格式与 URL 中请求参数的格式相同；timeout 属性设置了超时时限，单位是毫秒；error 与 success 均绑定了一个函数，前者负责处理异常情况，可能是超时或者服务器端返回了异常消息；后者负责处理正常情况，在新 MVC 框架中则是对页面进行更新。

4.2.6 刷新页面

服务器端返回的信息内容分为 3 个部分，首先是 Container 所对应的页面区域的信息，其次是附加信息，最后是 Container 树结构信息。这 3 个部分之间使用“|”进行分割。

对返回信息的处理需要在 success 所绑定的函数中进行，该函数有一个参数，此参数就是返回的信息，是一个字符串，假设参数名为 txt。处理的过程如下：

Step1：设置变量 i，初始值为 txt 的字符个数减 1，即 txt.length-1。

Step2：如果 i 小于 0，则结束。

Step3：否则，调用 txt 的 charAt 方法，参数是 i，将返回值赋给变量 c。

Step4：如果 c 为“|”，则执行 **Step6**。

Step5：否则，将 i 值减 1，执行 **Step2**。

Step6：调用 txt 的 substring 方法，参数为 i+1，返回值即为 Container 树结构信息的字符串。再调用 txt 的 substring 方法，第一个参数为 0，第二个参数为 i，将返回值赋给变量 txt，这一步的作用是更新 txt，将 txt 的第三部分信息删除。执行 break 跳出循环。

Step7：将 i 减 1。

Step8：如果 i 小于 0，则执行 **Step14**。

Step9：否则，调用 txt 的 charAt 方法，参数是 i，将返回值赋给变量 c。

Step10：如果 c 不为“:”，则执行 **Step7**。

Step11：否则，调用 txt 的 substring 方法，参数为 i+1，返回值即为某个 Renderer 所输出的信息的字符个数，将返回值转换为整数类型并赋给变量 temp。再调用 txt 的 substring 方法，第一个参数为 0，第二个参数为 temp，返回值为该 Renderer 所输出的信息，将该信息赋给变量 content。取出 content 的第一部分，即某个 Container 的名称，然后调用“$("#Container 的名称").html("content 的第二部分")”，这样就刷新了一个页面区域。

Step12：调用 txt 的 substring 方法，参数为 temp，将返回值赋给变量 txt，这一步的作用是删除 txt 中已经被使用的页面区域的信息。

Step13：执行 Step7。

Step14：结束。

4.3 系统性能评价

笔者最近参与开发了一个管理信息系统——长春理工大学学工办奖助管理系统，该系统应用了新的 MVC 框架。目前还在进行系统的测试，就测试的结果来看，新 MVC 框架运行比较稳定，没有出现因为框架原因导致页面更新错误的情况。因此，新 MVC 框架的可靠性还是得到了保证。

本章主要介绍的是新 MVC 框架在平台独立性、运行效率以及安全性等方面的评价。

4.3.1 客户端性能

客户端性能主要包括两个方面，平台独立性与运行效率。平台独立性主要是指对客户端浏览器的独立性，即所能支持的浏览器。通过测试，可以确定新 MVC 框架支持 IE 6、IE 7、Firefox 2 以及 Firefox 3。当然，这并不代表该框架只支持这四种浏览器，而是只测试了这 4 种浏览器，其他的浏览器还没有测试过。

运行效率主要考察在不断刷新页面的过程中，系统的内存占用率以及 CPU 的资源占用率。为此，选择了 Web 应用中常见的翻页功能来进行这项测试。因为该项功能需要传输的数据量大——每一页通常都有 20 条的记录数，而且该功能可以不断进行测试，即通过上一页、下一页、再上一页不断往复进行。该测试过程是一个人工的过程。选择用于测试的操作系统平台是 Windows XP Home sp2 以及 Ubuntu Linux 8.10。在 Windows 下使用了 IE 6 以及 Firefox 3 浏览器进行测试，测试结果如图 4.4、4.5 所示。

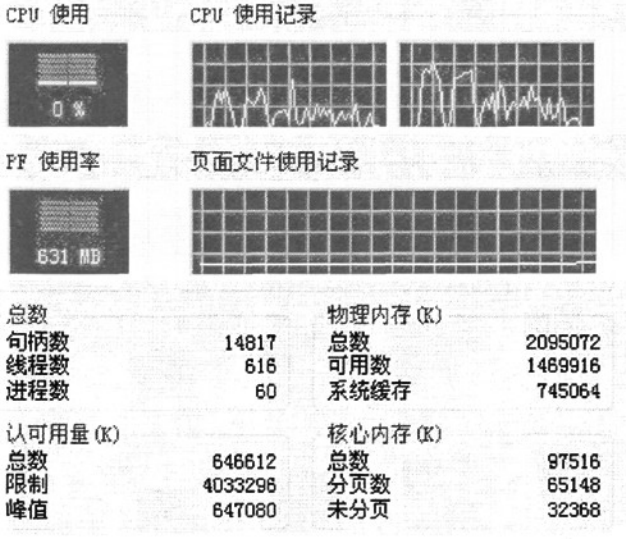


图 4.4 IE 6 测试结果图

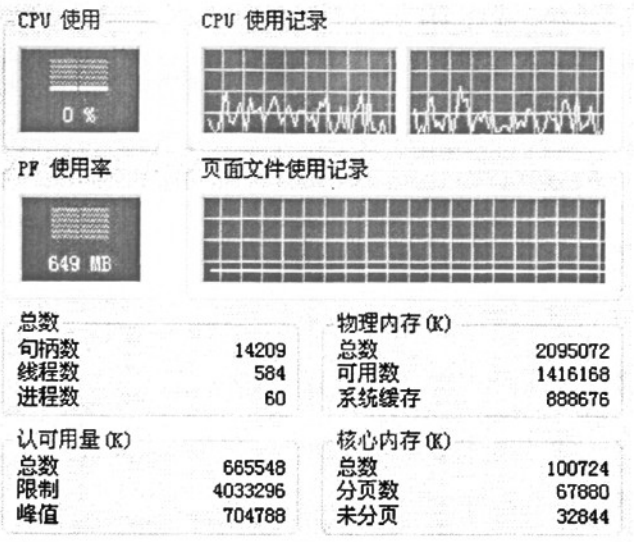


图 4.5 Firefox 3 测试结果图

从测试结果来看，不论是使用 IE 还是 Firefox 浏览器，系统的内存占用率一直都很稳定，从图中的“页面文件使用记录”可以看到，内存占用率十分平稳。“CPU”使用记录则显示十分不稳定，但这正是符合要求的效果。证明 CPU 并不一直处在繁忙状态，而是繁忙与空闲交替进行。CPU 忙是因为浏览器正在呈现响应，空闲则是因为浏览器正在等待服务器响应。

在 Ubuntu 下使用了 Firefox3 浏览器测试，测试结果如图 4.6 所示

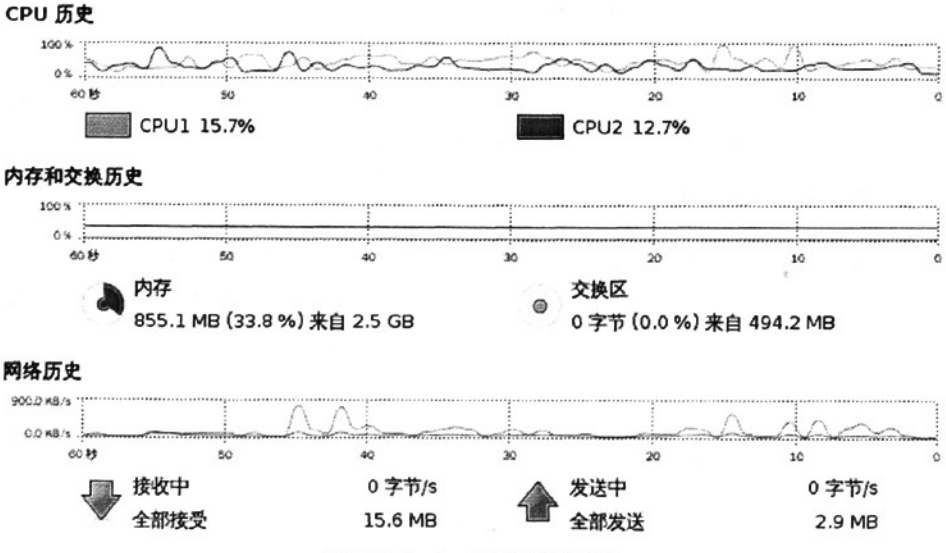


图 4.6 Firefox 3 测试结果图

测试结果和在 Windows XP Home sp2 下类似，也是符合要求的。

4.3.2 服务器端性能

服务器端性能也包括两个方面，平台独立性与运行效率。新 MVC 框架基于 Java 语言开发，众所周知，Java 语言具有跨平台的特性，因此新 MVC 框架所支持的服务器端平台与 Java 语言所支持的服务器端平台是一致的。

运行效率主要考察系统运行过程中服务器的内存占用率以及 CPU 资源占用率。选择用于测试的操作系统平台是 Ubuntu Linux 8.04 Server，应用服务器软件平台是 IBM Websphere Application Server 6.1。测试的过程中使用了 Websphere 自带的“性能查看器”进行实时监控。测试结果如图 4.7 所示。

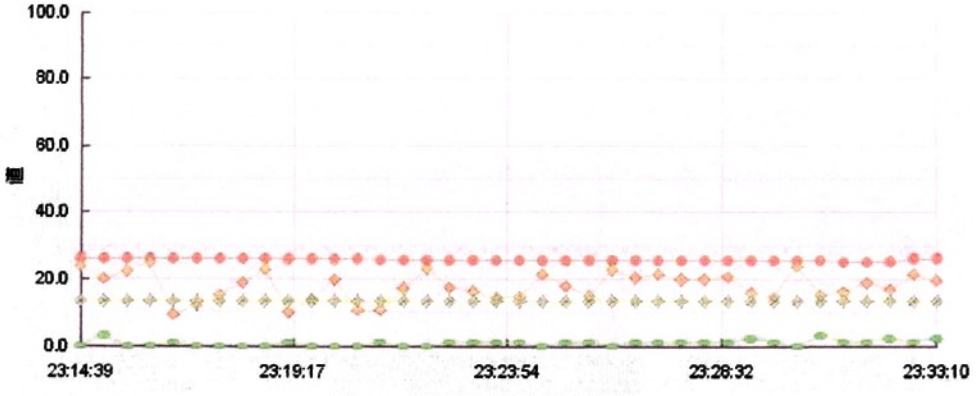


图 4.7 服务器端测试结果图

图 4.7 中的红线表示最大的堆值，黄线表示被使用的内存大小。可以看出，黄线呈现锯齿状并没有超过红线，红线也十分稳定。因此，系统的内存占用率是符合要求的。当内存占用高时，表示服务器正在响应用户请求，而内存占用低时表明服务器正在释放内存。

图 4.7 中的绿线表示 CPU 占用率。可以看出，CPU 的占用率很稳定，这可能和服务器处理器有较好的性能有关。

4.3.3 安全性

安全性主要是从理论上进行分析。首先是 Web 资源的访问安全。在这一点上，新的 MVC 框架是很安全的。因为在 3.2 节介绍文件结构时已经介绍了新的 MVC 框架的页面文件是和类文件存放在一起，而类文件在 WEB-INF 文件夹中。该文件夹有个特点，即无法通过 URL 来访问。这是 J2EE 规定的，由各个应用服务器软件厂商实现。因此，只要操作系统，应用服务器软件是安全的，则 Web 资源必定是安全的。任何客户端都无法跳过 Web 应用直接访问某个页面。这与 Struts、JSF 等 MVC 框架截然不同，应用这些框架开发的 Web 应用，其页面文件是可以直接通过 URL 访问的。

其次是 URL 的隐蔽性。这一点主要得益于 Ajax 的应用。由于是 Ajax 请求，因此

发送请求时，在浏览器地址栏上无法看到请求目标和参数。又由于响应是用 JavaScript 处理，因此无法通过浏览器的查看源文件的方式看到更新后的页面源代码。对于攻击者来说，请求信息和响应信息都是很难获取的。因此，较好地隐藏了应用资源。

第五章 系统的扩展

新 MVC 框架的可靠性、运行效率以及安全性是符合要求的。但是，新 MVC 框架在功能上还是显得太单薄。随着互联网的发展以及增强世界各国人民之间相互交流的需要，Web 应用支持本地化已经成为一种趋势。此外，对 Web 应用的负载能力也提出了更高的要求。因此，是否支持国际化以及集群环境也就成了评价一个 MVC 框架好坏的两个重要指标。

5.1 对国际化的支持

5.1.1 国际化和本地化

国际化就是设计软件应用，从而在不改变它们程序逻辑的前提下支持各种语言和区域。本地化就是设计软件应用支持特定地区。

一个设计如果用于国际化的应用，它的界面元素（例如错误消息和 GUI 组件标签）应该以运行应用的计算机的本地语言来显示。也就是说，界面内容不是硬编码的，而通常是从文本文件中动态提取的。另外，在显示和文化相关的数据（例如日期或货币）时，格式应该遵从用户的语言和区域。同时，当增加一种新的语言时，应用不需要重新编译。

国际化简称为 i18n 因为国际化的英文单词（internationalization）以 i 开头，以 n 结尾，并且 i 和 n 之间有 18 个字母。

地区（locale）是一个特定的地理区域、特定的行政区域或特定的文化 区域。如果一个操作所执行的任务是地区相关的，则称这个操作是区分地区的（locale-sensitive）。例如，显示日期就是一个区分地区的操作，因为日期必须是国家或区域所接受的格式。2008 年 8 月 24 日在中国写为 2008/8/24，在北美则写为 8/24/2008。

本地化简称 l10n，因为本地化的英文单词（localization）以 l 开头，以 n 结尾，并且 l 和 n 之间有 10 个字母。

5.1.2 Java 语言对本地化和国际化的支持

地区由 java.util.Locale 类表示，它有 3 个主要部分：language（语言）、country（国家）和 variant（变量）。很明显，语言是最重要的部分。然而，有时语言并不能充分表达一个地区。例如，英国和美国都说英语，但是英国说的英语和美国说的英语并不相同。因此需要指定语言的国家。

变量参数是厂商特定的或浏览器特定的代码。例如，Windows 使用 WIN、Macintosh 使用 MAC 等。

在 JDK1.4 中，java.util.Locale 类有 3 个构造方法：

```
public Locale(String language);
public Locale(String language, String country);
public Locale(String language, String country, String variant);
```

对于文本内容，一个支持国际化的应用针对每个地区都会有一个独立的属性文件。每个文件都包含一些键/值对，每个键唯一地标识一个地区特定的对象。键一般是一个字符串，它的值可以是字符串或任何其他的对象类型。

一个英文版本的属性文件如下，它有两个键：greetings 和 farewell。

```
greetings=Hello
farewell=Goodbye
```

一个中文版本的属性文件如下：

```
greetings=你好
farewell=再见
```

现在只需要使用 java.util.ResourceBundle 类就可以很容易地选择和读取用户地区特定的属性文件并查找值。ResourceBundle 类有一个基名，它可以是任何名称。为了使 ResourceBundle 类可以获得一个属性文件，属性文件的命名格式为：ResourceBundle 基名的后面跟上一个下划线，然后跟上一个语言代码。例如，假设基名为 MyResources，则对应美国和中国的属性文件分别是：

```
MyResources_en_US.properties
MyResources_zh_CN.properties
```

通过调用 ResourceBundle 的静态方法 getBundle 可以获得 ResourceBundle 对象，这个方法需要的参数是基名和地区：

```
ResourceBundle msgs = ResourceBundle.getBundle("MyResources", locale);
```

上述代码将载入 ResourceBundle 对象，这个对象含有相应属性文件中的值。然后调用该对象的 getString 方法取得值，参数为键。代码如下：

```
msgs.getString("greetings");
```

上述代码返回的值将由 locale 决定，如果 locale 是中国，则返回“你好”；如果是美国，则返回“Hello”。

5.1.3 新 MVC 框架对国际化的支持

从用户的角度来看，使用新的 MVC 框架不可避免需要编写 Container 子类。通常用户会将相关的 Container 子类组织到一个包中。如果这些 Container 子类所对应的 VTL 文件中含有需要被本地化信息，则要在这些 Container 所在的包下建立一个文件夹，名为“i18n”，在该文件夹下建立属性文件，属性文件的基名为 LocalStrings。该属性文件中的值可以被它所在的包中的所有 Container 子类所对应的 VTL 文件引用。假设“cn”

包下有两个 Container 子类，类名分别为 Container1 和 Container2，这两个类所对应的 VTL 文件中都含有需要被本地化的信息。则类文件、VTL 文件以及属性文件的组织结构如图 5.1 所示：

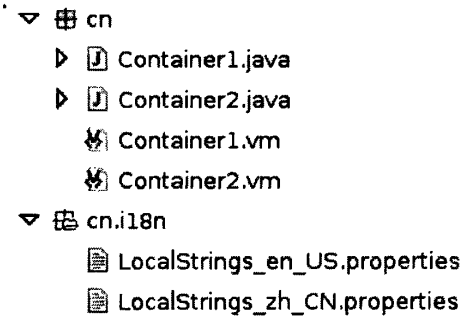


图 5.1 文件结构图

如果在属性文件中有个键“greetings”，则在 VTL 文件中可以使用“$msg.get('greetings')”来引用 greetings 所对应的值。

为了支持国际化，新 MVC 框架需要引入一个新的类 Message，该类的构造方法有两个参数：Container 和 Locale。构造方法利用这两个参数构建 ResourceBundle 对象，Container 的作用是确定属性文件的位置，Locale 则用于根据地区选择正确的属性文件。Message 的 get 方法十分简单，就是调用 ResourceBundle 对象的 getString 方法。

为了能够在 VTL 文件中使用“$msg”，需要对 Renderer 的 render 方法做下改造，只需添加一条语句：

```
cc.put("msg", new Message(container,request.getLocale()));
```

上述代码中的 cc 指的是 ChainedContext 类型的对象，即 Velocity 的上下文。

5.2 对集群环境的支持

5.2.1 集群环境与非集群环境

非集群环境系统结构如图 5.2 所示。这是一个非常典型的单服务器环境，一台 Web 服务器，一台应用服务器，一台数据库服务器。Web 应用主要被部署到应用服务器上。

图 5.2 非集群环境系统结构图

集群环境通常有多台 Web 服务器，多台应用服务器，多台数据库服务器。集群环境能够提供更高的性能，更可靠的服务。Web 应用主要部署在应用服务器集群上，每台应用服务器上都有一个独立的 Web 应用。集群环境系统结构如图 5.3 所示。

图 5.3 集群环境系统结构图

集群环境与非集群环境的最大区别在于，在非集群环境下，某个客户端在任何时刻发送的请求都是由同一台应用服务器为其提供服务，而在集群环境下，某个客户端在不同时刻发送的请求很有可能是由不同的应用服务器为其提供服务。如图 5.3 所示，假设集群环境下有 A、B 两台应用服务器，客户端在 a、b 两个不同时刻均发送请求。a 时刻请求由 A 服务器响应，而 b 时刻请求却由 B 服务器响应。当然，b 时刻请求也可能由 A 服务器响应，具体由哪台服务器响应要取决于管理服务器的调度算法。

5.2.2 新 MVC 框架在集群环境下遇到的问题

应用新 MVC 框架后，任何请求都要经过 4 个阶段的处理：重建 Container 树、应用请求值、调用应用以及呈现响应。在这 4 个阶段中，第一个阶段和最后一个阶段容易在集群环境下出问题，而中间两个阶段一般不会出问题，因为它们都是对第一个阶段所确定的 Container 进行操作。

第一阶段和最后一个阶段出问题主要是由于 ContainerPool 的设计不适合集群环境。ContainerPool 的 containers 属性用于将 ID 映射到 Container 子类。由此产生一个问题，即在集群环境的各台应用服务器上，相同的 ID 是否都映射到相同的 Container 子类。例如，假设集群环境中有两台应用服务器 A、B，Web 应用有一个 Container 子类，名为 Container1，在 A 服务器上，其 ID 是 5，那么在 B 服务器上，ID 是否也为 5。答案是不确定的。之所以如此，是因为 Container 子类是在 Web 应用的运行过程中逐渐被加入 containers 的，Container 子类的 ID 是它在 containers 中的索引值，在不同的应用服务器上，Container 子类在被加入到 containers 时，containers 中元素的个数有可能不同，因此 Container 子类的索引值就有可能不同。例如，某名为 Container1 的 Container 子类，当它被加入到 A 服务器的 containers 时，containers 中已有 4 个元素，那么 Container1 的 ID 在 A 服务器上的值为 4。而当 Container1 被加入到 B 服务器的 containers 时，containers 中已有 5 个元素，那么 Container1 的 ID 在 B 服务器上的值为 5。

5.2.3 对新 MVC 框架的改进

改进的目标是要实现任何一个 Container 子类在集群环境中的任何一台服务器上的 ID 值都相同。因此 ContainerPool 的 containers 属性不能动态添加元素，必须在 Web 应用初始化时将所有的 Container 子类加入 containers 属性中。为此，需要使用配置文件，

该配置文件中存储的是 Web 应用中的所有 Container 子类的类名，格式如下：

cn.Container1

cn.Container2

cn.Container3

...

配置文件中每个 Container 子类的类名占据一行。配置文件就是一个普通的文本文件。新 MVC 框架在解析配置文件时，按照从上往下的顺序依次获取每个 Container 子类的类名并将其加入 containers。这样保证 Container1 的 ID 在任何服务器上的值都是 0，Container2 的 ID 在任何服务器上的值都是 1，依次往下。

由于一个 Web 应用可能包含很多 Container 子类。由开发人员手动将所有 Container 子类的类名写入配置文件的工作量很大。为了简化开发，新 MVC 框架提供了一个自动生成配置文件的工具，开发人员只需要指定 Web 应用源代码所在的根文件夹，则该工具就可以自动生成配置文件。具体生成步骤如下：

Step1：获取文件夹中的所有文件，置入集合 files。

Step2：从 files 中依次取出每一个 file。

Step3：如果 file 的扩展名为 vm，则检查 files 中是否有主文件名与 file 主文件名相同且扩展名为 java 的文件，如果有则将该 java 文件对应的类名加入配置文件。

Step4：否则，执行 **Step2**。

Step5：获取文件夹中的所有文件夹，置入集合 directories。

Step6：取出 directories 中的每一个 directory，对其执行 **Step1**。

第六章 结束语

6.1 总结

新的 MVC 框架的设计思想主要借鉴了 Struts 和 Brasato 这两个优秀的 MVC 框架，力求使开发更加简单，更加灵活。

简单性主要体现在无配置文件（尽管在集群环境下需要使用配置文件，但配置文件是可以自动生成的）。所有的页面跳转都在 Java 代码中完成。实际上新 MVC 框架没有页面跳转这个概念，页面跳转被理解为对页面某个区域的更新。更新页面的某个区域反映到 Java 代码就是对 Container 树的重新组织。

灵活性主要体现在页面风格可以随意定制。新 MVC 框架并没有定义页面的样式，开发页面就和编辑普通的 HTML 页面类似。例如一个表单，开发人员可以任意安排表单中输入文本框的位置，可以将所有的输入文本框用表格来布局。使用 Brasato 则不行，因为表单组件只能将输入文本框横向排列。如果开发人员想要自定义样式，则需要修改组件代码，这是很困难的。

新的 MVC 框架的核心概念是 Container，这个概念是在新的 MVC 框架中首次采用。它类似于容器，可以层层嵌套；也类似于组件，拥有值并可以响应事件。

通过测试，新的 MVC 框架的可靠性和执行效率是符合要求的。其安全性也是有保障的。此外，新的 MVC 框架还支持集群环境，这进一步扩展了它的适用范围。

6.2 展望

在现有工作的基础上，还需要从以下方面展开进一步的研究工作：

1. 进一步提高新框架的可靠性。这需要通过在实际项目中多次应用来验证，是一个耗时的过程。本课题由于时间关系，只在一个项目中应用了新框架，在后续的工作中，需要更多地应用新框架，对每一项功能都要做相关测试。
2. 进一步对新框架的执行效率进行调优。这需要通过进行大量的压力测试来检验。压力测试除了使用软件测试外，还可以在项目的实际应用中进行测试。
3. 增强新 MVC 框架的功能。主要是方便页面开发。需要提供相关的 JavaScript 方法等。

致　谢

本论文是我攻读硕士期间的工作总结，在此谨向所有帮助、关心、爱护过我的老师、同学、朋友和亲人致以诚挚的谢意。

我由衷的感谢我的导师杨华民教授两年来对我的指导和教育，使自己在理论知识、科研能力和个人素质等方面都取得了可喜的进步和提高。本论文从选题、方案制定、寻找资料以及论文撰写等各个方面都得到了导师的悉心指导和帮助。杨老师有着深厚的理论造诣、丰富的实践经验，对前沿科学有敏锐的洞察力。他不但具有渊博的学识，严谨的治学作风、勇于创新的科研精神、对人对事的态度更是我学习的榜样，这必将使我终生受益。

感谢实验室的底晓强老师。他开阔了我考虑问题的思路，给了我全新的角度，不时给我一些建议要求，让我总是能够朝着正确的方向努力，使我受益良多。也感谢计算机学院的各位老师，他们在这两年多的时间里给予我许多关心和帮助，使我愉快地度过了研究生阶段。

感谢祁晖、王洪斌、梁田、于光等同学和实验室的师弟师妹们，他们让我处在一个团结、合作、友好、上进的集体中，在学习上、生活上给了我无私的帮助。能和他们相识、相处是我的荣幸，谢谢他们对我的关心和帮助。

感谢我的家人以及亲戚朋友多年来对我的支持、鼓励和帮助，使我顺利地完成了学业。

在此，一并向所有帮助我的人表示衷心的感谢。

参考文献

[1] 赵永屹,宿红毅,胡韶辉. 基于 AJAX 与 J2EE 的新型 Web 应用的设计与实现. 计算机工程与设计, 2007(01): 189～192

[2] 孙凌燕,陆保岚,孙健. 基于 Struts 的 Web 应用框架设计与研究. 计算机工程,2005(8): 57～60

[3] 李春红,高建华. 使用分层模型改进 MVC 设计架构. 计算机工程与设计,2007(04): 766～769

[4] 娄锋,孙涌. 轻量级 IoC 容器的研究与设计. 计算机技术与发展, 2007(01): 91～93,97

[5] GREY, MURRAY. Asynchronous Javascript Technology and XML (Ajax) with Java 2 Platform, Enterp rise Edition [EB/OL]. 2005-06-09. http://java.sun.com/developer/technicalArticles/J2EE/AJAX

[6] JESSE JAMES GARRETT. Ajax: A New Approach to Web Applications [EB/OL]. 2005-02-18. http://www.adaptivepath.com/ideas/essays/archives/000385.php

[7] 何成万,余秋惠. MVC 模型 2 及软件框架 Struts 的研究. 计算机工程,2002(6): 271～274

[8] 甘早斌,彭彬,李志欣. 基于集中控制的 MVC 模型. 计算机工程与设计,2005(2): 454～455

[9] 阎宏. Java 与模式. 北京：电子工业出版社. 2005

[10] Cooper B F, Sample N. A fast index for semi-structured data. VLDB, September 2005: 341～350

[11] CRANE D, PASCARELLO E, JAMES D. Ajax in action. America: Manning, 2005: 246-278.

[12] 付登科,郝克刚,葛玮. AOP 改进观察者模式——实现关注点的分离. 计算机应用, 2005(S1): 410～412

[13] HAKANSSON A . UML as an app roach to modeling knowledge in rule-based systems. In Proceedings of ES 2001, the Twenty-first SGA I International Conference on Knowledge Based Systems and Applied Artificial Intelligent, 2001: 187 - 200

[14] 周世兵,刘渊.运用 UML 对基于 J2EE 的 Web 应用系统建模研究. 计算机工程与设计, 2007(02):368-370

[15] CHUNGW WC, PAK JF. A case study: using UML to develop a knowledge-based system for supporting business in a small financial institute. International Journal of Computer IntegratedManufac-turing, 2006, 19 (1) : 59 - 68

[16] 孙巍,徐学东,徐学军. Java 反射机制在可重构 Web 框架中的应用. 计算机工程与应用, 2005(36), 92～94

[17] 胡东东,孟小峰. 一种基于树结构的 Web 数据自动抽取方法. 计算机研究与发展,2004(10): 1607～1613

[18] 王甲民,杨子翔,沈均毅.用 UML 设计 XML 文档模式.计算机工程与应用, 2002(22) : 131～133

[19] E.Gamma, R.helm, R.Johnson, and J.Vlissides. Design Patterns-Elements of Reusable Object-oriented Software. Addison Wesley, 1995

[20] 刘芳,肖铁军.XML 应用的基石:XML 解析技术.计算机工程与设计, 2005, 26(10): 2823～2824

[21] RAMLJAK D, PUKSEC J, HULJEN IC D, et al. Building enterprise information system using model driven architecture on J2EE platform 7th International Conference on Telecommunications-ConTEL 2003

[22] ARTHUR J,AZADEGAN S. Spring framework for rapid open source J2EE Web application development—a case study Proceedings of the Sixth International Conference on Software Engineering, 2005

[23] 王东,孙彬.基于 Ajax 的 MVC 框架的改造分析.计算机应用, 2007(S1); 293～295

[24] 米海波,吴照林.JSP 与 Ajax 在 Web 系统视图层上的性能对比分析.计算机应用, 2007(S1):281～282,285

[25] Dustin R.Callaway.Inside Servlets:Server Side Programming for the Java™ Platform.Addison Wesley/Pearson 2002.

[26] Gusfield D.Algorithms on string, tree and sequences. Cambridge University Press, 2004.

[27] 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):96～97

[28] 牛纪桢,陆坤,宋丹.以 XML 扩充的 MVC 设计模式. 计算机工程与设计, 2005,26(12):3372～3374

[29] 房丽娜,唐胜群,曾奕,等.基于 Web 应用的 MVC 架构实现——AWDF.计算机工程, 2005,31(10):89～90

[30] FOWLER M. Patterns of Enterprise Application Architecture. USA: Person Edition Inc, 2003

[31] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal. Pattern-Oriented Software Architecture-A System of patterns. Wiley,2000

[32] A Arasu , H Garcia-Molina. Extracting structured data from Web pages. In : Proc of ACM SIGMOD'031 New York : ACM Press ,2003. 337～348

[33] 许劲松,石磊.基于递归 MVC 结构的 Web 应用软件分析模式.计算机工程与设计, 2005,26(12):3417～3419

[34] 王林章, 李宣东, 郑国梁.一个基于 UML 协作图的集成测试用例生成方法.电子学报, 2004, 32(8): 1290～ 1296

[35] 虞蕾,赵宗涛,李刚.基于 UML 和组件的应用软件开发技术研究. 计算机应用与软件, 2007(03):34～36,71

[36] Mark Grand. Patterns in Java. Vol.3. John Wiley & Sons, Inc., 1998

[37] Bill Shannon, Mark Hapner, Vlada Matena, James Davidson, Eduardo Pelegri-Llopart, Larry Cable the Enterprise Team. Java 2 Platform, Enterprise Edition: Platform and Component Specifications. Addison Wesley,2000

[38] 汪广怡,龙源,张士峰.一个面向组件的软件分发框架.计算机工程, 2000(7):72～73

[39] 钟晖云,徐海水,廖志坚,黄常青,李锦棠. 基于 Ajax 的轻量级身份认证. 计算机应用研究, 2007(07):135～137

[40] Li Wenjie, Li Mingshu, Wang Qing,et al. VICOS: A framework for session management in network computing infrastruce. Proceedings of the 2004 IEEE International Conference on Services Computing (SCC'04), 2004.674～650

[41] Ma Bo,Zhang Yi,Shi Xingguo. Liquid meta-services:A component-based operating system layer for pervasive computing. Advances in Embedded Software and System, 2004, (11): 317～322