摘要

随着企业信息化的发展，各种基于不同平台的应用系统随之产生。但这些应用系统彼此之间相互独立数据分散，很难协作，这就往往形成一个个的信息孤岛。在这些应用系统复杂度不断提高的同时，用户对信息获取的便利程度要求也在不断地提高，这样的发展趋势使得企业应用集成研究成为一个具有很强现实意义的研究课题。企业门户的产生，为企业应用集成提供了重要的解决方案。企业门户负责将不同的应用和数据集成到一个统一的信息管理平台，同时提供统一的入口和个性化的配置，被公认为是下一代企业应用桌面。

鉴于企业门户存在着巨大的潜在商业价值，各大厂商纷纷开始生产自己的企业门户产品。由于早期企业门户产品的开发没有统一的标准，使得企业门户的发展受到严重地制约。JSR168 和 WSRP 两大规范的相继推出，JSR168 规范规定了标准的 Portlet 接口和运行机制，WSRP 规范规定了使用远程 Portlet 的机制。这两大规范使得企业门户的发展逐步趋于规范化，同时也促进了企业门户的发展。

本论文的主要工作包括：简要的介绍了企业应用集成和企业门户，较为全面的分析了 JSR168 和 WSRP 规范。基于 Portal 技术将 Struts 2 Web 应用集成到支持 JSR168 规范的 Jetspeed-2 企业门户中，同时实现了 Web 应用的单点登录功能；优化了 Portlet 的通信机制，提出了一种自适应的机制来改善用户访问 Portal 的体验；最后将 WSRP4J 项目的模块集成到企业门户中实现应用的共享。

关键词：企业应用集成；企业门户；Portlet；Jetspeed；WSRP

Research and Implementation of Web Application Integration Based on Portal Technology

Abstract

With the development of enterprise information, all kinds of different applications based on different platforms are producing. However, these applications are independent and make of decentralized data, and difficult to collaborated, which forms information isolated islands. These applications become more and more complexity, at the same time users' convenience requirements of access to information are constantly improved. According to such development trend, the enterprise application integration becomes a practical study for us. The formation of enterprise portal provides an important solution for enterprise application integration. Enterprise Portal will be responsible for different applications and data and integrate then into a unified information management platform, provide the unified entrance and the personalized configuration. It is generally acknowledged to be the next generation enterprise application desktop.

In view of enterprise portal exist a huge potential commercial value, the major manufacturers have begun to produce its own enterprise portal products. As there is no uniform standard for enterprise portal product development at early time, enterprise portal development was seriously restricted. JSR168 and WSRP specification have launched one after the other. JSR168 specification specifies the standards portlet APIs and portlet runtime mechanism, while WSRP specification provides using remote portlet mechanism. These two specifications make the development of enterprise portal gradually become standardized, while also promoting the development of the enterprise portal.

The main task of this thesis include: introduce to the enterprise application integration and enterprise portals briefly and analyze the JSR168 and WSRP

specification roundly. Based on portal technical integrate the struts 2 web application into enterprise portal—Jetspeed-2 which supported JSR168 specification, as well as implement single sign-on of web application; optimize the portlet communication mechanism, and propose an adaptive mechanism to improve user access portal experience. Finally, integrate module of WSRP4J project into enterprise portal, so that can share the application which integrates to enterprise portal.

Keywords: Enterprise Application Integration, Enterprise Portal, Portlet, Jetspeed, WSRP

西北大学学位论文知识产权声明书

本人完全了解西北大学关于收集、保存、使用学位论文的规定。学校有权保留并向国家有关部门或机构送交论文的复印件和电子版。本人允许论文被查阅和借阅。本人授权西北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，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。同时授权中国科学技术信息研究所等机构将本学位论文收录到《中国学位论文全文数据库》或其它相关数据库。

保密论文待解密后适用本声明。

学位论文作者签名：杨青林 指导教师签名：

2008年6月19日　　2008年6月19日

西北大学学位论文独创性声明

本人声明：所呈交的学位论文是本人在导师指导下进行的研究工作及取得的研究成果。据我所知，除了文中特别加以标注和致谢的地方外，本论文不包含其他人已经发表或撰写过的研究成果，也不包含为获得西北大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

学位论文作者签名：杨青林

2008年6月19日

第一章 绪论

1.1 研究背景与意义

随着企业信息化的发展，各种基于不同平台的应用系统随之产生。但这些应用系统彼此之间相互独立，各成独立的系统，这就形成一个个的信息孤岛。而且这些不同的应用系统往往都有自己的认证机制，用户需要多次登录才能得到所需的信息，从而增加了使用的复杂度。如何将企业中一个个的信息孤岛集成到一起，使得适当的人(Who)在适当的时间(When)可以获取适当的信息(What)，这就提出了企业应用集成的课题。企业应用集成能够将企业的业务流程、应用软件、硬件和各种标准联合起来，在两个或更多的企业应用之间实现无缝集成，使它们像一个整体或一个系统一样处理企业业务过程[1]。

传统的企业应用集成方案由于缺乏统一的协议，使得彼此之间没有良好的互通性。同时传统的企业应用集成方案属于高度耦合的集成，客户端和服务器相互的依赖程度较高。企业门户的产生为企业应用集成提供了重要的途径。

企业门户能够把多种来源不同的信息整合到一个门户页面中，并且能够提供统一的身份验证和个性化配置。Portlet 是门户的重要组成部分，是基于 Web 组件的 Java 技术，处理来自门户页面上的请求以及动态的产生内容。早期关于 Portlet 和 Portlet 容器没有一个统一的标准，不同的企业门户提供商有自己专用的 Portlet API，这就导致开发出的 Portlet 没有通用性，很难在不同的 Portlet 容器中运行，严重的制约着企业门户的发展。在 2003，Java Portlet Specification v1.0 作为 JSR(Java Specification Request)168[4]发布，它定义了 Portlet 容器和 Portlet 之间的协议并且提供了一个方便的设计模型。随后，各企业纷纷将自己的 Portlet 容器转化为基于 JSR168 的 Portlet 容器，JSR168 得到很快的、广泛的支持。

本论文的研究课题源于加拿大 Platform 公司的关于适应企业门户的调研项目。VMO[5](Virtual Machine Orchestrate)系统提供了一个基于 Struts 2 的 Web 应用简称 VMO GUI，使得用户可以通过 GUI 界面来管理、监控、配置计算机资源等。随着企业门户技术的日趋成熟，企业开始意识到独立的 Web 应用很难适应客户和社会的需求。所以企业在开始启动项目的时候，需要考虑如何使开发

出的产品即能满足普通客户的需求(即对独立 Web 应用的需求)，又能很容易的集成到客户拥有的企业门户中。这样即可以消除信息孤岛，也可以减少用户在不同系统之间频繁切换带来的麻烦。

1.2 研究的现状

1.2.1 企业应用集成的现状

随着企业应用系统的增多，企业应用集成逐步被重视起来。传统的点到点的集成方案，使得应用的接口复杂，通信状况混乱，而且也不利于整个应用系统的扩充。为了解决传统的点到点集成方案的所面临的问题，学术界和工业界都提出了自己的解决方案。

在工业界方面：Sybase 公司是具有代表性的企业应用集成供应商之一。Sybase 公司的电子商务集成架构[6](e-Business Integration Architecture)使用一个集成服务器来负责和不同的应用进行交互，它实际上是一种基于中间件的企业应用集成方案。这种方式可有效地帮助客户构建灵活、可扩展以及高性能的集成应用，它为用户提供一系列产品，使企业能够按照一个完整、灵活的工作流程成功地将新的和现有的信息系统有效集成，从而有助于企业更轻松地开发、修改它们的电子商务功能。

在学术界方面：基于 Web 服务的企业应用集成[7]，是通过分析遗留系统，将需要暴露出来的功能封装成 Web 服务，通过调用 Web 服务来实现应用之间的协同工作；基于 ESB(企业服务总线)的企业应用集成[8]，是从 SOA 发展而来的，是一种为进行连接服务提供的标准化的通信基础结构，基于开放的标准，为应用提供一个可靠的、可度量的和高度安全的环境，并可帮助企业对业务流程进行设计和模拟。

1.2.2 企业门户的现状

1998 年 11 月美国美林投资集团(Merrill Lynch)发表了一份报告。该报告预测，到 2002 年，美国公司在企业信息门户上的投资将从 1998 年的 44 亿美元增长到 148 亿美元。面对如此巨大的潜在市场，各大软件企业纷纷投入到企业门户产品的开发中，不断推出功能强大的企业门户产品。

1.2.2.1 商业的企业 Portal 产品

面对巨大的商业价值，BEA、IBM、ORACLE、SUN 和 MICROSOFT 等 IT 企业推出各具特点的 Portal 产品：

1．Weblogic Portal

BEA 公司的主要产品，提供了灵活的、功能强大的框架，生命周期管理工具和业务逻辑等功能使得开发者能很快的开发出自己的 Portal 产品。BEA Weblogic Portal 是使用 J2EE 体系结构实现的，为企业的集成提供了完整的门户解决方案。

2．WebSphere Portal Server

IBM 的 Portal 产品，支持 J2EE 标准。提供单点登录、Web 内容发布和搜索、个性化配置、企业应用集成、业务过程集成、文档管理和编辑、内容管理、知识管理、商业智能、移动设备支持等功能。

3．Oracle9iAS Portal

Oracle 的 Portal 产品，同样也支持 J2EE 标准，提供企业应用集成、内建商务智能、对移动设备和多语言支持、通过本地缓存改善 Web 性能、内建多种用途的 Portlet 等功能。可以非常方便地和 Oracle 公司的应用产品集成使用，也可以和基于 Oracle 数据库的应用集成[13]。

4．IPlanet Portal Server

支持 J2EE 标准，提供社区的创建和管理、多层次个性化配置、内容集成、搜索和索引服务等功能，为门户集成提供了全面的、完整的解决方案。

5．SharePoint Portal Server

Microsoft 的产品，支持.Net 平台，提供在 Microsoft 环境下的文档管理、搜索、协作，E-Mail、日历、联系方式管理等功能。使得公司跨越团队、部门来共享信息和文档更加的方便和可靠。可以与 Microsoft 自身知识管理产品配套使用，但是与主流的企业应用软件不易集成。

1.2.2.2 开源的企业 Portal 产品

开源的企业 Portal 产品也纷纷涌现，由于免费、公开，这些开源产品也倍受人们的关注。下面介绍一些比较流行的开源的企业 Portal 产品：

1．Jetspeed[9]

Jetspeed 是 Apache Jakarta Project 项目组的一个子项目，是一个开放源码的企业门户的实现，主要使用了 Java 和 XML 技术。通过健全的门户安全政策管理着所有访问 Jetspeed 的用户。集成到 Jetspeed 中的 Portlets，是独立的应用，扮演着中心路由器的作用，将来源不同的信息以一种方便的形式展现给用户。

通过 Jetspeed 表现的数据是独立于内容类型的。也就是说，来自 XML，RSS(Rich Site Summary)或者 SMTP 的内容都可以整合到 Jetspeed 中，数据的表现经过 ATEX XSL 进行处理，然后通过 JSP 和 HTML 传递给用户。Jetspeed 提供像 Cocoon、WebMacro 和 Velocity 之类的模板技术和内容发布框架。

基于 Jetspeed 的门户可以通过一个网站提供给用户一些应用、数据库信息和其它可得到的数据源。Jetspeed 提供一种安全架构，用户可以根据他所拥有的角色配置 Jetspeed 提供的信息和功能。用户可以通过 Web 浏览器、手机上的 WAP 浏览器、寻呼机或者其他 Servlet 引擎支持的终端设备访问 Portal 页面。

2．Liferay[10]

Liferay 代表了完整的 J2EE 应用，使用了 Web Service、EJB、JMS 以及 AOP 等多种技术，基于 XML 的 Portlet 配置文件可以灵活动态的扩展，使用 Web Service 来支持一些远程信息的获取，使用 Apahce Lucene 实现全文检索功能。Liferay 有一个清晰的架构，允许使用各种各样的容器。这些容器即可以包括轻量级的 Servlet 容器比如 Tomcat，又可以包括完全支持 J2EE 的容器比如 JBoss、Weblogic 等。事实上，Liferay 是仅有的一个几乎支持所有常用的开源和商业 Java 服务器的开源 Portlet 容器。Liferay 门户提供了可以聚集、共享、协作的虚拟空间和流线型的业务流程，减少了操作的费用，增进了客户的满意度，满足了现代企业的需要。

3．eXo platform[11]

eXo Platform 是一个开源企业门户解决方案，提供基于 Java(tm) Server Faces (JSF) 的框架，同时遵循 JSR 168 规范。eXo Platform 所有的业务逻辑都被封装成相互依赖的服务，由于使用了控制反转技术(IoC)使得原本相互依赖的服务联系变得松散起来。

主要优点包括：由 AOP(AspectJ)实现的内容管理系统，极大提高了内容管

理性能； Portlet 容器是基于著名的 IoC3 轻量级容器 PicoContainer；同时也实现了上下文共享，二次开发的流程比较清晰；提供工作流技术服务；通过使用 XML 技术可以为结构化的信息轻松地创建视图。

4. uPortal[12]

uPortal 是 JASIG(Java in Administrator Special Interest Group)组织主持开发的基于 Java, XML, JSP 和 J2EE 的开源门户项目，是在学术研究界使用最广泛的 Portal 框架。uPortal 提供健全的基础门户架构服务，使用开放、易扩展的软件结构，同时支持 JSR168 和 WSRP 标准。uPortal 着眼于高等教育机构，允许每个人定义独一无二的、个性化的校园网视图。一些商业组织也用它来作为构建协同社区的框架。

1.3 本文的主要工作

按照基于企业门户来完成企业应用集成的想法，将基于 Struts 2 的 VMO GUI 应用部署到企业门户中。本文使用 Jetspeed-2 作为企业门户来具体论述将 VMO GUI 应用部署到 Jetspeed-2 中的步骤。同时实现了单点登录功能来彻底地将 VMO GUI 应用整合到 Jetspeed-2 中，优化现有的 Portlet 通信机制，提出一种自适应的机制来改善用户访问 Portal 页面的体验。

深入的分析 WSRP 的体系结构和工作原理，在此基础上将集成到企业门户中的 VMO GUI 应用通过 WSRP 协议实现 Portlet 的共享，即本地部署的 Portlet 在远程的企业门户中可以使用，从而极大的方便了用户的使用。

1.4 论文的结构框架

第一章 介绍了论文的研究背景、意义以及研究的现状，论述了本文要完成的具体工作。

第二章 简要的介绍了企业应用集成和企业门户，说明了基于 Jetspeed 企业门户来完成 VMO GUI 应用集成的有效性。

第三章 介绍 Portal 技术的基本概念包括 JSR168 规范、Portlet 容器以及 Portal 的工作原理。同时还介绍 WSRP 规范，以及规范中定义的 WSRP 的组成成分和工作原理等。

第四章 Jetspeed门户框架的介绍。

第五章 应用系统的集成。介绍了将VMO GUI应用集成到Jetspeed-2中的具体步骤，同时实现了单点登录，优化了现有Portlet的通信机制，提出了一种自适应的机制。最后通过WSRP协议实现远程Portlet。

第六章 集成到Jetspeed-2中VMO GUI的演示结果。

第七章 论文的总结与展望。

1.5 本章小结

本章分析了企业遇到的应用集成问题，使用基于企业门户来完成企业应用集成的方案。总结了现阶段在企业应用集成和企业门户方面的研究成果。具体说明了本文要完成的工作以及论文的主要结构框架。

第二章 企业应用集成

在20世纪60~70年代，企业应用的唯一目标就是通过计算机代替一些孤立的、重复性的工作。到了20世纪80~90年代，随着企业应用的增多，一些企业开始意识到企业应用集成的价值和必要性。一些企业开始对现有的应用进行整合，以便让它们可以集成到一起，这就引入了企业应用集成。

企业应用集成(Enterprise Application Integration, EAI)能够将业务流程、应用软件、硬件和各种标准联合起来，在两个或更多的企业应用系统之间实现无缝集成，使它们像一个整体一样进行业务处理和信息共享。EAI不仅包括企业内部的应用系统和组织的集成，还包括企业与企业之间的集成，以实现企业与企业之间信息交换、商务协同、过程集成和组建虚拟企业和动态联盟等[1][2]。

2.1 企业应用集成分类

EAI从不同的角度有不同的分类方法。根据应用集成的对象来划分，可以分为面向数据和面向过程的集成；根据应用集成所使用的工具和技术来划分，可以分为平台集成、数据集成、构件集成、应用集成和业务集成；根据企业组织角度来划分，可以分为水平的组织内的集成、垂直的组织内的集成和不同组织间的集成[1][3]。

这些不同的分类又可以从广度和深度来进行概括。从深度而言，EAI可以分为表示层、数据层和功能层的集成；从广度而言，EAI可以分为部门内的、部门间的、企业内的、企业间的集成。

2.2 企业应用集成的方案

2.2.1 点对点的集成

点对点的集成就是为了使两个系统能相互协作而开发出彼此连接的接口。这种集成方案的想法很简单，而且对于较少的系统相互协作又是非常有效的。同时允许分布式应用并发的进行数据交互。但是随着应用系统的增多，这种点对点集成的弊端就暴露无遗，从点到点集成的结构图可以看出：

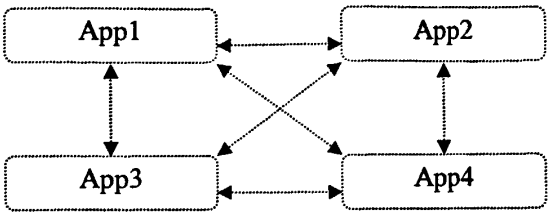

图 2.1 点对点的集成

随着应用系统的增多，需要开发的彼此连接的接口将以 n*(n-1) (n 为应用的个数)增加，而且如果任何一个应用发生改变或者是需要集成新的应用，整个工作将变得非常复杂。显然这种方案不能很好的满足较多应用集成的需要。

2.2.2 基于中间件的集成

基于中间件的集成方案是在所有应用之间增加中间件层，由它负责提供通用的接口，所有集成的应用通过它传递信息。下图为这种方案的结构图：

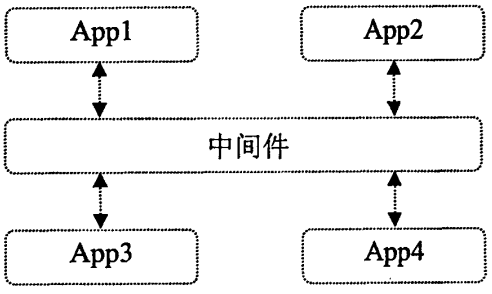

图 2.2 基于中间件的集成

这种基于中间件的集成方案有效的克服了点对点集成方案的缺点，可以支持更多应用的集成，而且也易于维护。常用的中间件的产品有 OMG 的 CORBA、Microsoft 的 COM+以及 SUN 公司的 Java2 企业版。

2.3 基于企业门户的企业应用集成

2.3.1 企业门户

企业门户(EP, Enterprise Portal)提供一个框架，用来越过组织界限集成信息、人和过程。它提供基于 Web 的统一的、安全的用户入口点，个性化的配置和整合不同信息和应用的功能。

企业门户的发展经历了四个阶段，其发展阶段图如下：

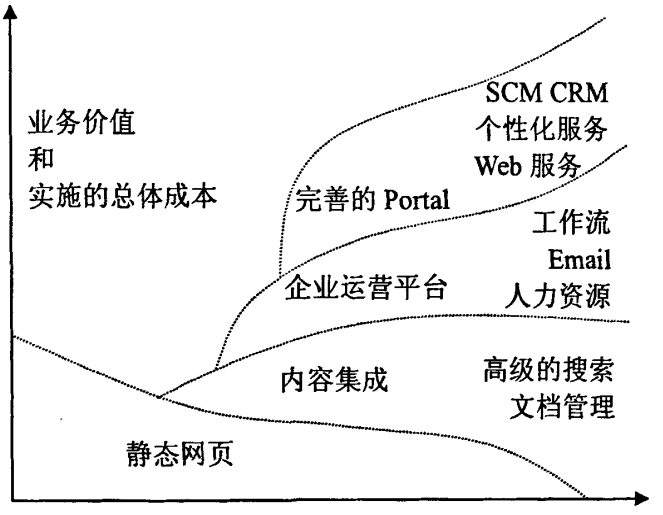

图 2.3 企业 Portal 的发展阶段[13][14]

第一个阶段就是门户中集成一些静态的网页，企业通过门户将相关的信息提供给用户。

第二个阶段就是企业将大家共享的文档集成到门户中，统一进行管理、分类并提供高级的搜索功能。用户可以通过门户轻松的获取到相关的文档。

第三个阶段就是企业门户将一些工作流、电子邮件、人力资源管理、后勤管理的功能集成进来，形成了功能比较完善的企业运营平台。

第四个阶段就是企业将 ERP、CRM、SCM 集成到门户中，同时提供高级的个性化服务和 Web 服务，这时的企业门户已逐渐成为完善的门户。

完善的企业门户所具有的特征：统一的入口、单点登录、统一的用户管理、统一的权限控制、统一的界面、强大的内容管理功能、个性化的服务、整合不同的系统以及协作和共享等特征。

企业 Portal 可以分为企业信息 Portal、企业知识 Portal 和企业应用 Portal。随着 Portal 技术的不断发展和企业应用集成的需求，使得这三种不同的企业 Portal 之间的界限逐渐消失，企业更需要三种企业 Portal 的有机结合体。

2.3.2 基于 Jetspeed 企业门户集成 VMO GUI 应用

企业门户的产生为企业应用集成提供了重要的途径。基于企业门户的企业应用集成是一种基于用户界面的集成，一般称其为肤浅集成(integration at the glass[15])。人们可以创建一个 Portal 页面，它包含的内容来源于不同地方，而不需要去做任何

后台的集成，所以从这个角度而言可以把这种集成称为肤浅的集成。但是，如果这种肤浅的集成可以利用20%的工作量完成80%的功能[37]，则称之为是一种有效的集成方案。

基于企业门户的集成技术实际上是一种基于中间件的集成技术。企业门户提供统一的平台，规定了符合某些标准的组件可以以一种可插拔的方式集成到平台中，通过平台将这些组件所要显示的内容呈现给用户。其工作原理如下：

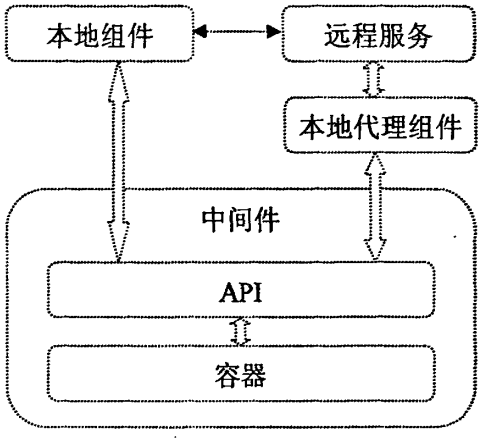

图2.4 基于企业门户集成技术的工作原理

从上图可以看出，基于企业门户的集成技术可以支持本地组件和远程服务两种组件。本地组件是直接部署到本地企业门户中，直接与企业门户提供的API进行交互。远程服务有两种形式，一种是只提供功能的Web服务或者说只返回数据的Web服务，本地的组件通过SOAP协议调用该Web服务；一种是提供功能和表现的服务，它实际上是部署到远程企业门户中的组件，并且实现了WSRP规范，使得部署到本地企业门户中的代理组件可以通过SOAP协议直接使用这些远程服务。

企业门户提供一个统一的平台，同时也规定了和平台进行交互的组件应该遵循的规范，本地组件需要遵循JSR168规范，远程既提供功能又提供表现的服务需要遵循JSR168和WSRP规范，此两个规范将在后续章节详细介绍。

本文通过Jetspeed这个企业门户集成现有的基于Struts 2的VMO GUI应用。集成的框架为：

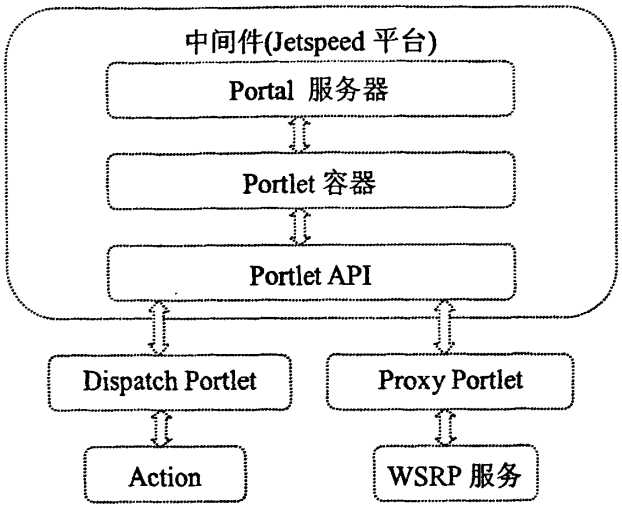

图 2.5 集成的框架

从上图可以看出,在该平台中部署了两个组件,分别为 Dispather Portlet 和 Proxy Portlet。Dispather Portlet 处理来自 Portal 页面的请求，将请求发送给相应的 Action 处理。它提供分发的功能，同时负责和 Portlet 容器进行交互。从而隐藏了 Action 对 Portlet 容器的依赖，也避免了原有的 Action 进行较大的修改，降低了应用集成所需要花费的工作量，同时也确保了原有的应用仍然可以正常的运行。Proxy Portlet 负责将远程的 WSRP 服务代理到本地。

2.4 本章小结

本章主要介绍了企业应用集成的概念、分类和企业应用集成的方案，简要的论述了企业门户的相关内容，说明了基于 Jetspeed 企业门户完成企业应用集成的可行性和有效性。

第三章 Portal 和 WSRP 技术概述

3.1 Portal 关键技术介绍

Portal 即门户，早期指的是 yahoo，google 这些著名的网站，随着 Internet 的发展，Portal 的概念也变得越来越广泛，它指任何可以提供某些信息或服务的网站。由于 Portal 存在着巨大的市场，各大公司纷纷开始开发自己的 Portal 产品。由于早期没有统一的标准，各大公司使用各自的 Portlet API 进行开发，这就使得 Portlet 的移植性、重用性很差，严重地阻碍了 Portal 技术的发展。随着 JSR168 和 WSRP 两大标准的发布，Portal 技术也日趋成熟起来。

3.1.1 Portal 的概念

Portal 是门户网站，是一个基于 Web 的应用，提供一个可以集成来自 Internet 或者 Intranet 的信息和应用程序的框架。同时 Portal 为这些来源不同的内容提供一个统一的入口，用户通过 Portal 这个门户网站便可以访问集成在 Portal 中的所有信息。由于 Portal 提供统一的框架，所有集成在 Portal 中的内容都可以有统一的外观。用户还可以根据个人喜好定制不同的内容。

3.1.2 Portlet 概述

Portlet 是一个基于 Java 技术的 Web 组件，专门负责处理来自客户端的请求以及产生动态的内容。Portlet 的生命周期由 Portlet 容器管理。Portlet 是 Portal 用于提供信息系统表示层的可插入的用户界面组件。可以被动态地加载和管理。例如，当 Portal 运行时，可以安装和除去组成 Portal 页面的 Portlets。Portlet 的设置和访问都可以在 Portal 运行时由管理员更改。

由 Portlet 产生的内容称之为片段(fragment)，是由具有一些规则的 Markup(如：HTML、XHTML、WML)组成的，而且还可以和其他的片段组合而成一个复杂的文件。Portal 页面是由 Portlet 产生的内容聚合而成的。

客户端和 Portlet 的交互是由 Portal 页面通过请求/响应的方式实现，一般来说，用户是与 Portlet 产生的内容进行交互的。这些 Portlet 产生的内容会根据用户的不同而有所不同，这完全依赖于客户对这个 Portlet 的设置。

3.1.2.1 Portlet 与 Servlet 之间的关系[4]

Servlet 规范[16]中是这样定义 Servlet 的："Servlet 是基于 Java 技术的 Web 组件，产生动态内容，并为容器所管理。与其他 Java 的组件一样，Servlet 也是与平台无关的，可以在任何支持 Java 的 Web 服务器中运行。Servlet 与客户端通过 Servlet 容器所实现的请求/响应机制进行交互的"。这样看来 Servlet 与 Portlet 是有很多的相同之处，但研究小组将新的 Web 组件定义为 Portlet，这就说明它们之间还是有所不同。首先对 Portlet 与 Servlet 的相同点作一介绍：

- 都是基于 Java 技术的 Web 组件
- 都能动态的产生内容
- 生命周期都由特定的容器所管理
- 和客户端的交互都是通过请求/响应的机制来完成的

Portlet 与 Servlet 的不同点：

- Portlet 产生的只是一些 Markup 片段，而不是完整的文档
- Portlet 不能直接和 URL 绑定
- 客户端和 Portlet 的交互必须通过 Portal 平台
- Portlet 有更加精细的请求处理：action 请求和 render 请求
- Portlet 有预先定义好的指示 Portlet 正在执行何种功能的 Portlet 模式和反映 Portal 页面中窗口实际状态的窗口状态
- 一个 Portal 页面可以存在多个同样的 Portlet 窗口

Portlet 还具有以下额外的 Servlet 所没有提供的功能：

- Portlet 存在访问和存储持久化配置和个性化数据的方法
- Portlet 可以访问用户配置信息
- Portlet 为在内容中创建超级链接而提供 URL 重写的功能，允许 Portal 服务器对链接和动作的创建保持透明
- Portlet 可以在两种不同的作用域内储存临时数据：应用程序全局作用域和 Portlet 私有作用域

Servlet 具有以下特点是 Portlet 所不具有的：

- 设置响应的字符编码

- 设置响应的 HTTP 头
- 访问客户端请求 Portal 的 URL

为了尽可能重用 Servlet 的架构，Portlet 吸收了 Servlet 中所有可以重用的成分，包括：部署、类载入、Web 应用、Web 应用生命周期管理、会话管理和请求分发。Portlet 中的许多概念和部分 Porlet API 都是模拟 Servlet 来完成的。

3.1.2.2 Portlet 与 Servlet/JSP 的桥梁

Portlet 能够利用 Servlet、JSP 和 JSP 标签库(Taglib)产生内容。Portlet 调用 Servlet 和 JSP 就如同 Servlet 利用请求分发器来激活 Servlet 和 JSP。当 Servlet 或 JSP 在 Portlet 中被调用时，传给 Servlet 或 JSP 的请求是以 Portlet Request 为基础的。同样，传给 Servlet 或 JSP 的 Response 是以 Portlet Response 为基础的。

3.1.3 Portlet 容器的概念

Portlet 容器是 Portlet 的运行环境，管理着 Portlet 的整个生命周期。同时还提供 Portlet 参数持久化存贮的功能。Portlet 容器接收来自 Portal 页面的请求，并将请求传递给 Portal 页面所包含的 Portlet 执行。

Portlet 容器不是类似 Servlet 容器的独立容器，它是在 Servlet 容器上通过扩展方式实现的，并重用 Servlet 容器所提供的功能。Portlet 容器并不负责集成 Portlet 产生的内容，这是 Portal 服务器的职责。Portal 和 Portlet 容器可以作为单一的应用组件组合起来，也可以作为是 Portal 应用的两个独立的组件。

3.1.4 Portal 页面

3.1.4.1 组成 Portal 页面的元素

Portal 页面由 Portlets、Themes 和 Skins 三个组件组成，核心组件是 Portlet。Portal 页面为：

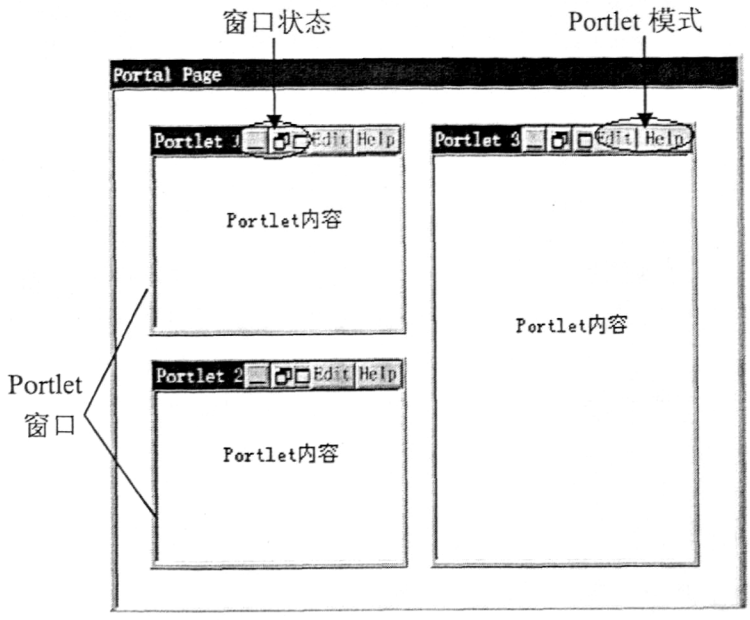

图 3.1 Portal 页面[4]

3.1.4.2 Portal 页面的产生

Portlet 运行在 Portlet 容器中，当一个来自客户端的请求到达 Portlet 容器后，Portlet 容器调用相应的 Portlet 处理该请求，随后将 Portlet 产生的内容传递给 Portal 服务器。Portal 服务器将所有 Portlets 产生的片段聚集成 Portal 页面传递给客户端设备(浏览器)，呈现给用户。

Portal 页面的事件响应序列如下：

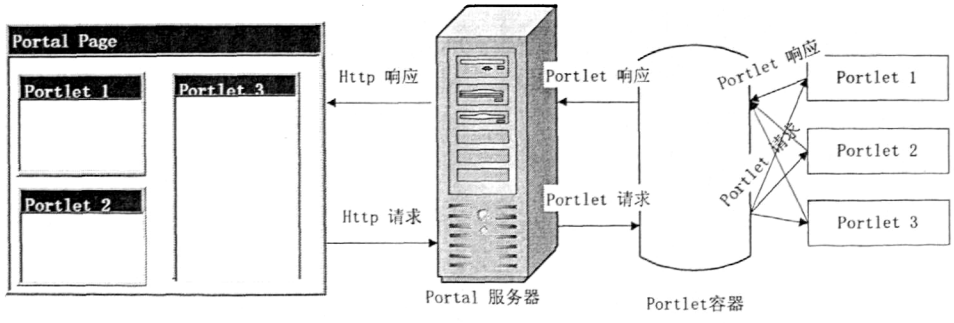

图 3.2 Portal 的事件序列

- 用户通过客户端设备进入 Portal 页面，然后发送请求。
- Portal 接收到请求后，判断此请求是否与组成 Portal 页面的 Portlet 有关。
- 如果存在与某个 Portlet 相关的动作，Portal 通过 Portlet 容器调用相应的 Portlet 处理请求。
- Portal 获得 Portlet 产生的片段，并通过这些片段聚集成 Portal 页面。

- 将 Portal 页面返回给客户端设备。

3.1.5 Portlet 模式

Portlet 模式指示当前 Portlet 正在执行的功能。通常，Portlet 依据当前的模式来执行不同的任务和创建不同的内容。Portlet 模式规定 Portlet 应该执行什么任务以及产生什么内容。当 Portlet 被调用时，Portlet 容器将当前 Portlet 的模式提供给 Portlet。在 Action 请求阶段，Portlet 可以通过程序改变 Portlet 的模式。

Portlet 规范定义了三种 Portlet 模式：View、Edit 和 Help。用户也可以定义自己的 Portlet 模式。不同的用户根据自己的角色来决定自己可以得到何种 Portlet 模式。

3.1.5.1 View 模式

每个 Portlet 必须支持的模式，也是 Portlet 默认的模式。Portlet 开发人员可以通过重写 GenericPortlet 类(实现 Portlet 接口并且提供一些默认功能的抽象类，开发人员只需要直接地或间接地继承这个类来实现自己的 Portlet 即可)的 doView 方法来实现 View 模式的功能。

3.1.5.2 Edit 模式

Edit 模式不是每个 Portlet 都必须支持的模式，如需支持该模式，开发人员则需重写 GenericPortlet 类的 doEdit 方法来实现 Edit 模式的功能。通常在 Edit 模式下，Portlet 会设置或更新 Portlet 的部分参数。

3.1.5.3 Help 模式

Help 模式同样也不是 Portlet 必须支持的模式，如需支持该模式，则需要重写 GenericPortlet 类的 doHelp 方法来实现 Help 模式的功能。通常在 Help 模式下提供 Portlet 的帮助信息。

3.1.5.4 用户自定义 Portlet 模式

JSR168 规范中允许用户自定义 Portlet 模式来实现特殊的功能。JSR168 规范规定在部署描述符中使用 custom-portlet-mode 元素来实现自定义 Portlet 模式。

自定义 Portlet 模式需要和具体支持该模式的实现相对应。如果没有支持该自定义模式的具体实现，Portlet 将不能在该模式下被调用。例如：支持剪贴板的自定义 Portlet 模式如下：

```
<portlet-app>
    ...
    <custom-portlet-mode>
        <description>Creates content for cut and paste</description>
        <name>clipboard</name>
    </custom-portlet-mode>
    ...
</portlet-app>
```

3.1.5.5 实现 Portlet 模式

如何具体地实现 Portlet 模式？需要在部署描述符中声明该 Portlet 支持的模式，例如：

```
<portlet-app>
    <portlet id="HostList">
        ...
        <supports>
            <mime-type>text/html</mime-type>
            <portlet-mode>view</portlet-mode>
            <portlet-mode>edit</portlet-mode>
            <portlet-mode>help</portlet-mode>
        </supports>
        ...
    </portlet>
</portlet-app>
```

从上述例子可以看出该 Portlet 支持三种模式，View、Edit 和 Help 模式，同时需要在对应的实现中增加实现 Portlet 模式的功能：

```
public class HostList extends GenericPortlet{
    protected void doView (RenderRequest request,RenderResponse response){
        ... }
    protected void doEdit (RenderRequest request,RenderResponse response){
        ... }
    protected void doHelp (RenderRequest request,RenderResponse response){
        ... }
}
```

3.1.6 窗口状态

窗口状态是一个 Portal 页面可分配 Portlet 产生内容的空间数量指示器。当 Portlet 被调用时，Portlet 容器提供给 Portlet 当前的窗口状态。Portlet 可以使用窗口状态决定需要产生多少信息。类似 Portlet 模式，窗口状态也可以通过程序在 Action 请求阶段进行修改。

Portlet 规范中定义了三种窗口状态：Normal、Maximized 和 Minimized。Normal 和 Minimized 状态指示当前 Portlet 可以和其他 Portlet 共享 Portal 页面，而 Maximized 状态则指示 Portal 页面由当前的 Portlet 独占。当窗口状态是 Maximized 时，说明 Portlet 可以产生丰富的内容，而窗口状态是 Minimized 时，说明 Portlet 应该输出最少的内容或者什么都不输出。

用户也可以根据自己的喜好，定义自己的窗口状态。在部署描述符中使用 custom-window-state 元素来定义自己的窗口状态。

3.1.7 Portlet 的生命周期

正如上文提到的 Portlet 是由 Portlet 容器管理，那么 Portlet 容器可以创建多少个 Portlet 的实例？在默认的情况下，也就是非分布式环境下，Portlet 容器只能将一个 Portlet 实例化。在分布式环境下，Portlet 容器在每一个虚拟机中可以实例化一个 Portlet。

Portlet 由定义良好的生命周期管理，该生命周期定义了如何加载、实例化和初始化 Portlet，以及如何处理来自客户端的请求和如何销毁该 Portlet 对象。

Portlet 完整的生命周期是由 Portlet 接口中定义的 init，processAction，render 和 destroy 方法构成。

3.1.7.1 加载实例化

Portlet 容器负责加载和实例化 Portlet。当 Portlet 容器启动 Portlet 应用时，或是 Portlet 容器决定该 Portlet 需要提供服务时，Portlet 容器将加载和实例化该 Portlet。Portlet 容器必须使用作为 Portlet 应用一部分的 Servlet 应用中用来加载 Servlet 的类加载器来加载 Portlet。当加载完成后，Portlet 容器实例化 Portlet 以备使用。

3.1.7.2 初始化

当 Portlet 对象被实例化后，Portlet 容器必须初始化 Portlet 以便处理请求。Portlet 容器通过调用 Portlet 接口中提供的初始化方法初始化 Portlet。在初始化过程中，Portlet 对象可能抛出 UnavailableException 或者 PortletException 异常。在这种情况下，Portlet 容器不能将 Portlet 对象作为存活的服务，必须释放该 Portlet 对象。若在初始化过程中出现错误，Portlet 容器将随时尝试重新实例化和初始化 Portlet。直到开发人员调用初始化方法成功后，方可认定该 Portlet 是一个存活的服务，才能和它进行交互。若 Portlet 初始化没有获得成功，则该 Portlet 的 destory 方法不会被调用。

3.1.7.3 请求处理

Portlet 对象被正确的初始化后，Portlet 容器可以调用它处理来自客户端的请求。在 Portlet 环境中请求分为两个阶段：ActionRequest 和 RenderRequest。同样在 Portlet 接口中定义了两个不同的方法处理这两种不同的请求。processAction 方法负责处理 ActionRequest，接收 ActionRequest 和 ActionResponse 两个参数。render 方法负责处理 RenderRequest，接收 RenderRequest 和 RenderResponse 两个参数。

通常，客户端的请求是通过 URL 来触发的。Portlet 请求是通过 Portlet URL 触发的。针对 Portlet 这个特殊的环境，Portlet URL 也分为 action URL 和 render URL。一般情况下，通过 action URL 触发的客户端请求被解释为一个 action 请

求和多个 render 请求，Portal 页面的每一个 Portlet 对应一个请求。而通过 render URL 触发的客户端请求被解释为多个 render 请求，同样也是和 Portal 页面中的 Portlet 一一对应。

来自客户端的请求处理的顺序为：

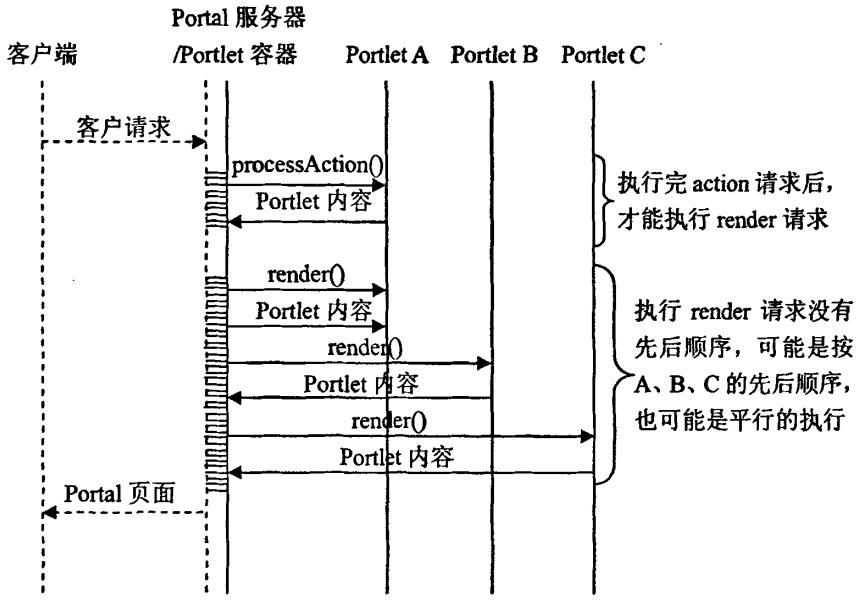

图 3.3 请求处理顺序图[4]

假如客户端的请求是通过 action URL 触发，Portlet 容器将首先调用目标 Portlet 的 processAction 方法来处理 action 请求。一个客户端请求只能触发一个 action 请求。在 action 请求过程中，Portlet 可以发出一个重定向请求，改变 Portlet 的模式、窗口状态和 Portlet 的参数数据，同时还可以给 actionResponse 对象设定 render 参数，以便在 render 阶段使用。当 action 请求处理完成后，Portlet 容器将调用 Portal 页面所包含的所有 Portlets 的 render 方法来处理 render 请求，这里存在一个例外，就是如果某个 Portlet 使用了缓存机制，将不会调用该 Portlet 的 render 方法处理 render 请求，而是直接从缓存中得到 Portlet 的内容。render 请求的触发没有固定的顺序。

如果客户端的请求是通过 render URL 触发，Portlet 容器将会调用 Portal 页面中包含的所有没有使用缓存机制的 Portlets 的 render 方法来处理 render 请求。在这个阶段，任何 Portal 页面包含的 Portlet 的 processAction 方法都不会被调用。

3.1.7.4 销毁

Portlet 容器不会永远加载一个 Portlet 的对象，当 Portlet 容器决定要终止服务时，将会调用 Portlet 接口的 destory 方法释放正在使用的资源和保存持久化的数据。在调用 destory 方法之前，允许和当前 Portlet 相关的正在执行的请求彻底执行完毕。为了避免永久的等待，Portlet 容器等待预先定义的时间后销毁这个对象。如果 destory 方法被调用，Portlet 容器将不能路由任何请求到该 Portlet 对象。如需重新使用该 Portlet，必须重新实例化 Portlet 以产生新的对象。下面给出完整的 Portlet 接口的程序：

```
public interface Portlet{
    public void init(PortletConfig config) throws PortletException;
    public void processAction (ActionRequest request, ActionResponse
        response) throws PortletException, java.io.IOException;
    public void render (RenderRequest request, RenderResponse response)
        throws PortletException, java.io.IOException;
    public void destroy();
}
```

3.1.7.5 Portlet 的数据模型

Portal 提供个性化的配置，也就是说不同的用户可以根据自己的角色得到不同的内容。由于 Portlet 设计模型使用轻量级模型，所以 java 对象本身不保存任何状态，当处理一个请求时，Portlet 将从外界得到个性化的数据。这些数据包括临时数据和持久化数据。Portlet 的数据模型图如下：

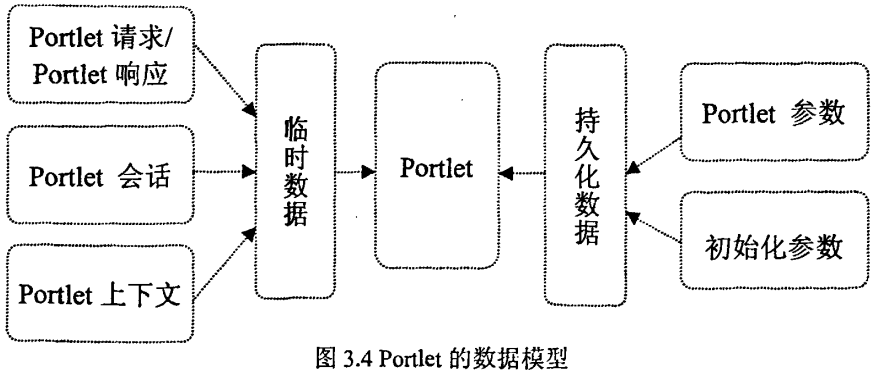

图 3.4 Portlet 的数据模型

从图 3.4 可以看出，临时数据由三种数据组成：

1. Portlet 请求和 Portlet 响应的数据：Portlet 请求的数据包括 Portlet 模式、窗口状态、表单信息等。Portlet 响应的数据包括 render 的参数等。
2. Portlet 会话：保存在应用程序全局作用域和 Portlet 作用域中的数据。
3. Portlet 上下文：包括 MimeType 数据、资源路径等。

持久化数据包括初始化参数和 Portlet 参数。初始化参数是在 Portlet 部署描述符中指定的只读的数据，包括 Portlet 的基本参数等，例如 render 输出的 JSP 的名字。Portlet 参数包括两种类型的参数：可修改的和只读的参数。可修改的参数在任何标准的 Portlet 模式下均可以修改。只读的参数是在 Portlet 部署描述符中声明只读的，不能被普通用户修改，只能由管理员修改。Portlet 参数只能在 action 阶段被读写，在 render 阶段只能读不能写。

3.2 WSRP 技术

3.2.1 WSRP 介绍

WSRP(Web Service for Remote Portlets)[17][18]规范是 OASIS 组织在 2003 年 9 月发布，定义了与交互式面向表现的 Web 服务进行交互的接口，也可以说是定义了门户和 Portlet 容器服务之间标准化接口的一个 Web 服务标准。它继承了面向服务的架构和 Web 服务的思想，它的推出规范了互联网门户之间的信息交换。

WSRP 规范主要包括两部分：一是定义了即插即用、可视化的面向表现的 Web 服务，二是定义了用于发布、查找、关联此类服务的接口。面向表现的 Web 服务通常即包含业务逻辑又包含表现逻辑。而标准的面向数据的 Web 服务只包含业务逻辑。图 3.5 描述了面向表现的 Web 服务和传统的 Web 服务(面向数据的 Web 服务)之间的区别[20]。

面向数据的 Web 服务提供的只是一些数据，客户需要实现自己的表现逻辑。作为消费者的每个应用通过独一无二的接口和 Web 服务进行交互，如果服务的接口发生了变化或是服务提供的数据对象发生改变，消费者需要修改自己的表现逻辑。显然这种方式不能很好的满足动态集成的要求。

面向表现的 Web 服务本身提供表现的内容，不需要消费者进一步的开发。只要遵循统一的规范，就可以直接将服务提供的内容集成到自己的门户产品中。

这种面向表现的 Web 服务具有很强的灵活性，可以很好的满足动态集成的要求。

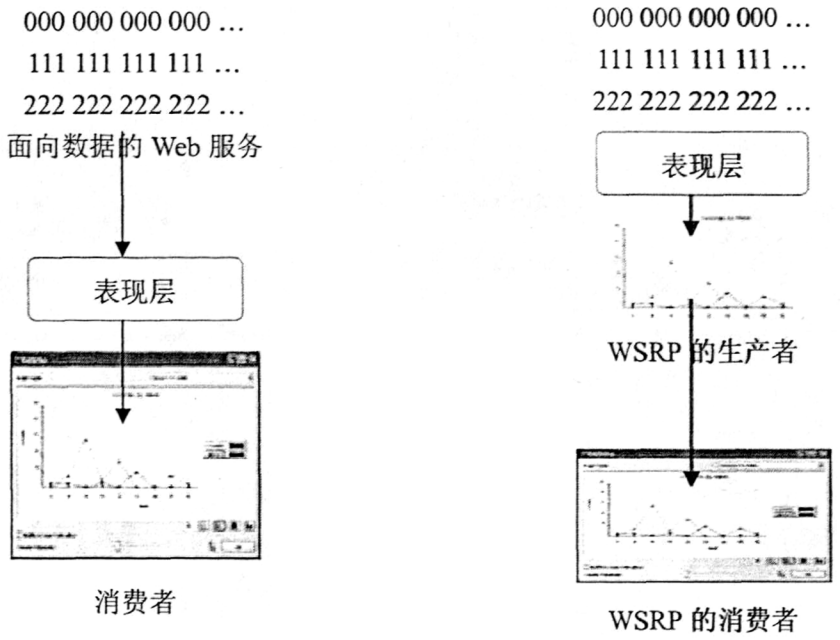

图 3.5 传统的面向数据的 Web 服务(左图)和面向表现的 Web 服务(右图)[20]

WSRP 就是使用面向表现的 Web 服务，生产者提供 Web 服务(Portlet)，消费者通过注册后便可以直接将这些 Web 服务集成到自己的 Portal 平台中，无需开发人员作任何编程。对最终用户而言使用远程的 Portlet 和本地部署的 Portlet 是一样的。这就满足了门户对不同来源的内容可以快速的集成起来，呈现给最终用户的需求。

3.2.2 WSRP 的参与者

WSRP 的参与者包括生产者、消费者、Portlet 和最终用户。生产者是面向表现的 Web 服务提供者，提供可以处理用户交互请求的 Portlet。生产者被作为 Portlet 的容器来建模的。为使生产者和消费者可以交互，WSRP 规范定义了四个 Web Service 接口：

- 服务描述接口：一个必须的接口。消费者可以通过它来查询生产者的性能、需求以及提供的 Portlets。
- Markup 接口：一个必须的、最重要的接口，用来和 Markup 片段进行交互。通过使用 Markup 接口，最终用户可以和生产者进行交互。
- 注册接口：一个可选的接口，用来确立生产者和消费者之间的关系。
- Portlet 管理接口：一个可选的接口，授权消费者访问生产者提供的 Portlet

的生命周期。该接口包含属性管理，它能使消费者通过程序访问 Portlet 的持久化状态。

消费者是一个中间系统，代表最终用户和面向表现的 Web 服务进行交互。消费者负责集成远程Portlet产生的Markup并呈现给最终的用户，管理和Markup的交互。

Portlet 是由生产者提供，负责处理用户的请求和产生动态的 Markup。

最终用户就是通过消费者的站点来使用生产者提供的服务的人。最终用户初始化和生产者之间的交互。

3.2.3 WSRP 的工作流程

WSRP 的工作流程为：

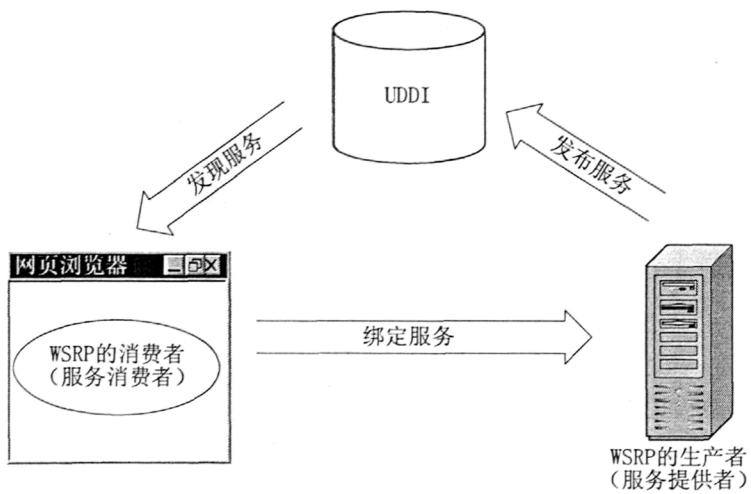

图 3.6 WSRP 的工作流程[20][33]

1. WSRP 生产者发布一个服务到类似 UDDI 的目录。
2. 消费者通过生产者提供的自我描述等信息进入类似 UDDI 的目录中查找并发现生产者。
3. 消费者绑定到生产者，在这一过程中可能会进行一些关于性能、安全需求和业务技术方面的信息互换。
4. 消费者和生产者绑定后，最终用户通过认证后便可以使用生产者所提供的聚合页面，同时消费者还可以进行个性化的配置。
5. 终端用户通过浏览器将请求发送给消费者，消费者接收最终用户的请

求，直接和生产者进行交互，将请求发送给目标 Portlet。

6. 生产者和消费者可以随时选择结束它们之间的关系。

3.2.4 WSRP 的生命周期

生命周期是指某项功能在怎样的状态下可以被使用、交互，直到最终被销毁的整个过程。WSRP 规范中规定了两种生命周期：

持久化的生命周期：这个生命周期由一个显式的创建操作开始，结束于显式的销毁操作。例如：在具体的 WSRP4J[19]项目中，如果显式的创建生产者和消费者的链接，在生产者的持久化数据文件中增加这样的数据，例如：

```
<consumer-portlet-registration-impl ···>
    <registration-handle>192.168.0.9_1196131901781_0</registration-handle>
    <portlet-handles>vmgui.192.168.0.9_1196688889648_0</portlet-handles>
</consumer-portlet-registration-impl>
```

其中<portlet-handles>元素的数据是在另外的持久化文件中定义的：

```
<ConsumerConfiguredPortlet>
    <ParentHandle>vmgui.HostList</ParentHandle>
    <PortletHandle>vmgui.192.168.0.9_1196688889648_0</PortletHandle>
</ConsumerConfiguredPortlet>
```

上述例子说明消费者和生产者所提供的 vmgui.HostList Portlet 进行交互。

临时的生命周期：这个生命周期从一个隐式的或显式的操作开始，创建的数据是临时的，不需要显式的销毁操作，通过一个过期元素来标识这个生命周期将何时结束。

3.2.5 WSRP 的作用域

作用域是信息共享的范围，用来描述信息何时是合法的。WSRP 规范中涉及到三种不同的作用域：

注册作用域：这个作用域在消费者向生产者注册时被初始化，结束于这个注册被释放。从生产者的角度看，这个作用域具有持久化的生命周期。

Portlet 作用域：当 Portlet 被克隆后作为消费者可配置的 Portlet 时，Portlet

作用域被初始化，该作用域被注册作用域所封装。当消费者可配置的 Portlet 所参考的 Portlet 被明确的释放后，该作用域结束。

会话作用域：当 Portlet 需要在生产者一端储存临时状态时，会话作用域被初始化。它总是被 Portlet 作用域所封装，即 Portlet 作用域结束后，会话作用域将自动结束。通过一种显式的操作或者过期机制，会话控制的状态被释放，标志着会话作用域的结束。

3.2.6 WSRP 的优点

通过 WSRP 服务，使得 Portal 管理变的更加容易、有效，Portal 管理员不在需要管理大量部署到本地的应用，只需通过查找得到需要的 WSRP 服务，集成到自己的 Portal 中即可。

通过 WSRP 服务，可以使得本地的 Portlets 得以共享，提高了分布式资源的利用率。

通过 WSRP 服务，将一些不可能在本地部署的应用，成功的集成到本地。

3.3 本章小结

本章主要介绍了与 Portlet 相关的两个标准 JSR168 和 WSRP 规范，JSR168 指定了本地 Portlet 部署到 Portlet 容器应该遵循的接口，而 WSRP 则指定了远程进入 Portlet 遵循的接口。具体介绍了 JSR168 规范中定义的 Portal、Portlet 以及 Portlet 容器的基本概念和它们之间的关系。由于 Portlet 是 Portal 中的重要组成部分，所以着重的介绍了 Portlet 的概念、模式、生命周期和工作原理等。对 Portlet、Portlet 容器有了基本的了解后，接下来介绍了另一个标准 WSRP 规范，它是继承 Web 服务的思想，提出面向表现的 Web 服务，使得遵循 WSRP 规范发布的 WSRP 服务，无需编写任何代码即可集成到远程的门户中。WSRP 规范的推出规范了互联网门户之间的信息交换，进一步推动了 Portal 的发展。

第四章 门户框架—Jetspeed 介绍

4.1 Jetspeed 概述

本文第一章已经初步介绍了 Jetspeed，它是 Apache Jakarta 项目中的开放源码门户系统，主要是使用 Java 和 XML，是企业级信息门户的原型实现。Jetspeed 有两个版本 Jetspeed 1.X 和 Jetspeed 2.X 可供选择。早期的 IBM 的 WebSphere Portal Server 就是在 Jetspeed1.X 的基础上进行二次开发的。Jetspeed1.X 都不完全支持 JSR168。当 JSR168 作为 Portlet 的规范在 2003 年发布后，Apache 便着手开始开发 Jetspeed-2，它提供对 JSR168 的全面支持。

Jetspeed-1[21]是基于 Turbine 框架开发的 Web 应用引擎，同时使用了 Velocity 和 ECS 技术。Turbine、Velocity 和 ECS 都是 Apache Jakarta 的子项目。Turbine 是一个基于 Servlet 的 MVC-2[22]框架，使用了面向服务的架构，允许 Java 程序员快速地创建 Web 应用。Turbine 提供了一些功能，包括安全管理系统、行程安排的服务、XML 校验服务和针对 Web 服务的 XML-RPC 服务，使得在 Web 应用中创建新的服务变得更加容易。Velocity 是一种高度实用的、基于 Java 的开发源码模版引擎，允许使用 Java 代码中定义的方法，从而将网页设计和 Java 代码分开，使得网页设计者和 Java 程序员可以并行工作。Velocity 是 Web 开发中的 Model 2 阶段的代表技术之一。ECS(Element Construction Set)是一个 Java API，为各种标记语言生成元素。它的工作原理是利用 Java 代码产生 HTML，然后将 HTML 储存到临时的 Java 对象中，最后将 HTML 发送到 Servlet 输出流中。

Jetspeed-2 完全不同于 Jetspeed-1，是基于 Spring 框架的，完全支持 JSR168 规范的新一代企业门户。Jetspeed-2 使用标准的 Portlet 容器的实现，即 Pluto[27](Portlet 的运行环境)。

Jetspeed-2 的主要新特征[9]如下：

- 完全兼容 Java Portlet API 标准，Jetspeed-2 是 Java Portlet API 的完整实现。
- Portlet 应用的自动部署：将开发好的 Portlet 应用拷贝到 Jetspeed-2 应

用下的 WEB-INF\deploy 目录中，Jetspeed-2 将自动部署该应用。

- 多线程的 Portlet 聚合引擎
- 基于 Spring 的基于组件的架构
- 基于管道的请求处理
- JAAS 数据库安全组件
- 可升级的架构：Jetspeed-2 将 Spring 框架作为默认的组件框架，用户也可以根据自己的需要替换 Spring 框架。
- Apache 门户桥接组件：使 Jetspeed-2 可以集成基于 Struts、JSF、WebWork、Perl、PHP 等开发的应用。
- 基于 CMS(Content Manage System)的网站导航
- 单点登录功能：用户只需登录一次，即可访问 Jetspeed-2 提供的所有服务。
- Web 内容和 Web 服务组件

本文的 Web 应用集成主要是在 Jetspeed-2 的基础上进行开发的。下面主要介绍 Jetspeed-2 的架构和工作流程。

4.2 Jetspeed-2 的架构

Jetspeed-2 在默认状态下使用 Spring 框架，主要使用 Spring 框架最核心的 IoC(Inversion of Control)[24]引擎 BeanFactory 和 ApplicationContext，由它负责管理所有 Jetspeed 组件的依赖关系和生命周期[25]。所有这些组件的 Spring 声明都是以 XML 文件的形式定义 jetspeed\WEB-INF\assembly 文件夹中。

Jetspeed-2 框架的加载是通过 JetspeedServlet 类来完成。JetspeedServlet 类在 Jetspeed-2 应用包含的 web.xml 文件中声明。当 Tomcat 启动时，初始化 Web 应用。当初始化 Jetspeed-2 应用时，首先读取 Jetspeed-2 应用下的 web.xml，相应的初始化该文件中包含的 Servlet。初始化 JetspeedServlet 类的具体过程如下图 4.1 所示：

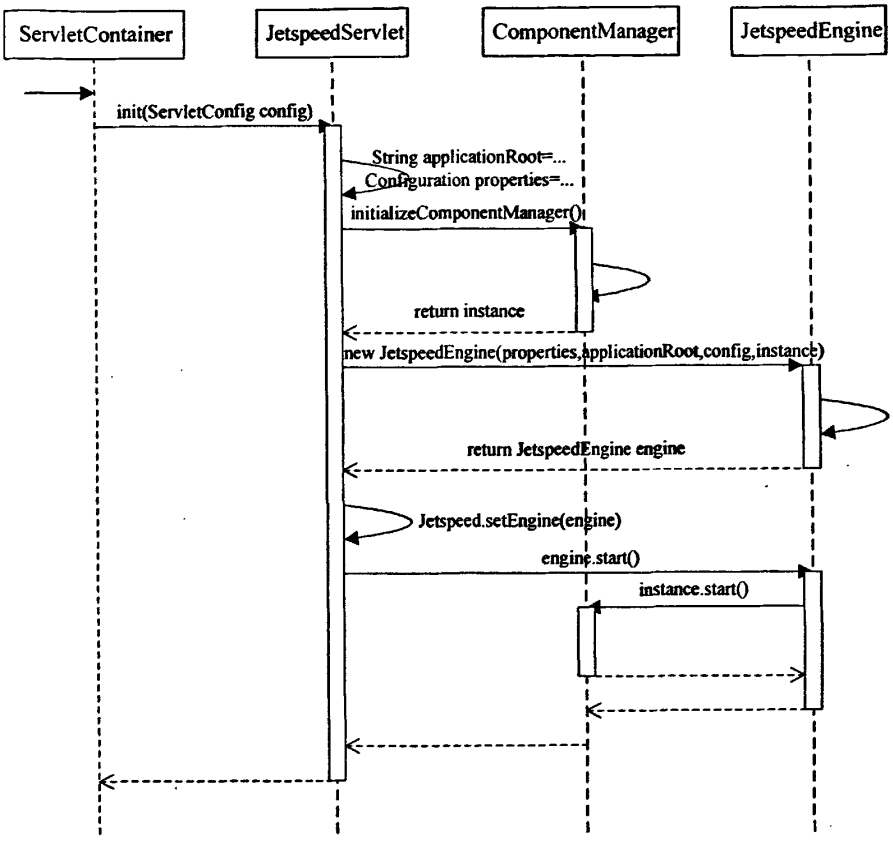

图 4.1 Jetspeed-2 启动时序图[25]

Servlet 容器调用 JetspeedServlet 类的 init()方法，该方法负责创建 JetspeedEngine 对象，该对象的创建需要传入具体的 ComponentManager 对象，默认状态下将 SpringComponentManager 对象传给 JetspeedEngine 对象。如需用其它的框架来管理 Jetspeed 的组件，则将相应的对象传给 JetspeedEngine 对象替换 SpringComponentManager 对象即可。JetspeedEngine 对象创建成功后，将保存到 Jetspeed 的环境变量中。然后启动 JetspeedEngine 对象和 ComponentManager 对象。JetspeedServlet 类和 JetspeedEngine 类的 UML 图如下：

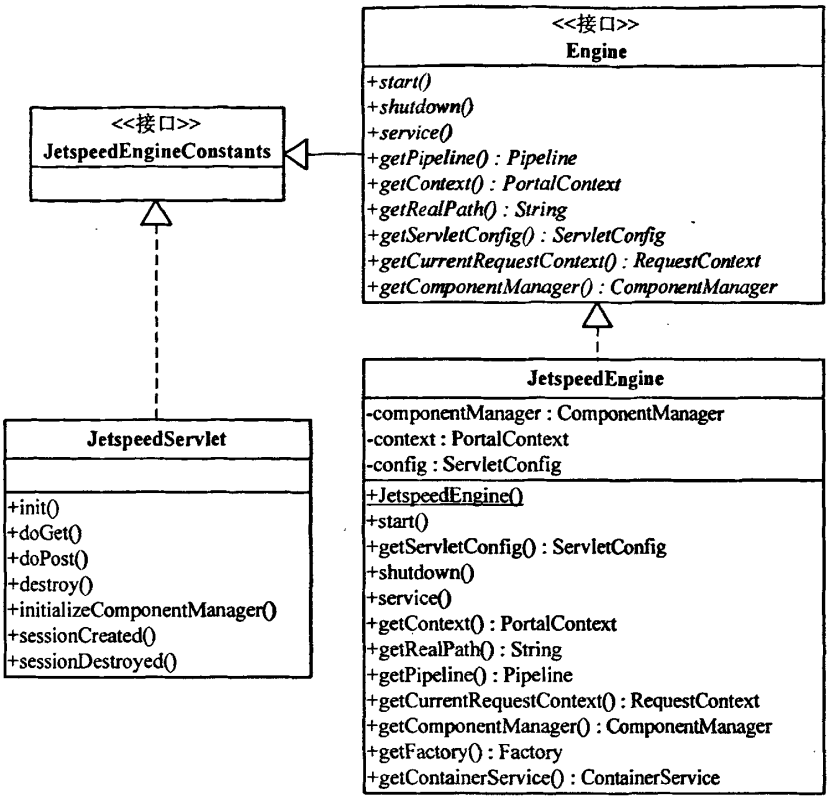

图 4.2 JetspeedServlet 和 JetspeedEngine 的 UML 图

从上述类图可以看出，JetspeedEngine 类是实现 Engine 接口的，而 Engine 接口又是继承 JetspeedEngineConstants 接口。JetspeedServlet 直接实现 JetspeedEngineConstants 接口。

4.3 Jetspeed-2 的工作流程

图 4.3 描述了 Jetspeed-2 的工作流程：

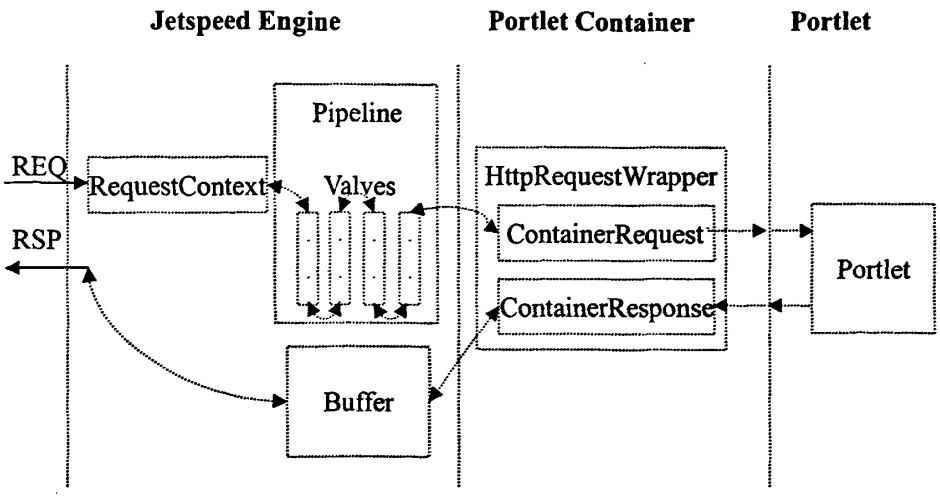

图 4.3 Jetspeed-2 工作流程[26]

(1) 来自 Portal 页面的请求到达 Servlet 服务器后，由 JetspeedServlet 对象负责接收。JetspeedServlet 对象从 Jetspeed 环境变量中得到 JetspeedEngine 对象，通过 JetspeedEngine 对象获得 ComponentManager 对象，然后通过 ComponentManager 对象获得 RequestContextComponent 对象。

(2) RequestContextComponent 对象会针对该请求建立一个相互可以映射的 RequestContext 对象。

(3) 将 RequestContext 对象作为参数传入 JetspeedEngine 对象的 service()方法中， JetspeedEngine 对象从传入的 RequestContext 对象中取得目标管道(Pipeline)来处理。Pipeline 使用了职责链(chain of responsibility)[28]模式，是由一堆 Valve 像链条串在一起组合而成的。

(4) 各个 Valve 依次执行，ActionValve 对象和 Portlet 容器交互，通过 Portlet 容器调用目标 Portlet 来处理这个请求。AggregateValve 对象负责集成 Portlet 产生的内容。CleanupValve 对象负责清空 renderStack 中保存的内容。还有 ContentVavle、SecurityValve、PageProfilerValve 对象等，这里不再赘述。

(5) 将集成的内容输出到 Portal 页面，呈现给用户，整个流程结束。

4.4 Jetspeed-2 的页面布局

Jetspeed-2 页面是由 Layout，Fragment 和 Decoration 组成的，其页面布局为：

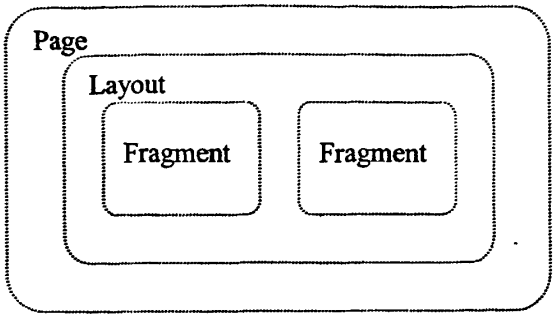

图 4.4 Jetspeed-2 的页面布局

一个 Portal 页面可以由多个片段(Fragment)和多种 Layout 组成。Layout 决定 Portal 页面的样式，例如 VelocityTwoColumns 决定 Portal 页面由两列组成。Decoration 是围绕在 Fragment 周围的任何静态的或半静态的 Markup，它不能访问或改变 Fragment 产生的内容。Page、Layout、Fragment 都可以有自己的 Decoration。

4.5 PSML 文件介绍

PSML(Portal Structure Markup Language)是门户结构描述语言，定义了 Portal 页面中的 Portlet 是怎样集成、布局以及装饰的。它不是 Java Portlet 标准 API 的一部分，是 Jetspeed-2 所支持的文件。在 Jetspeed-2 中每一个 Portal 页面将对应一个 PSML 文件，由它来决定 Portal 页面的布局、装饰以及由哪些 Portlets 组成。下面列举一个简单的 PSML 文件：

```
<?xml version="1.0" encoding="UTF-8"?>
<page id="example">
    <title>Example</title>
    <short-title>Example</short-title>
    <defaults skin="orange" layout-decorator="tigris" portlet-decorator="tigris"/>
    <fragment id="jsabout-10" type="layout" name="jetspeed-layouts::
                                                velocityOneColumns">
        <fragment id="jsabout-11" type="portlet" name="j2-admin::
                                                    WelcomeToJetspeed">
            <propertyname="row" value="0" />
```

```
        </fragment>
    </fragment>
</page>
```

该文件中定义了 Portal 页面的标题、皮肤、装饰以及包含的一个 Portlet 即 WelcomeToJetspeed，该页面使用的布局是 jetspeed-layouts::VelocityOneColumns，表明该 Portal 页面只能有一列。

4.6 本章小结

本章主要介绍了 Apache 的开源门户产品 Jetspeed，介绍了由起初的基于 Turbine 框架的 Jetspeed-1 到现在的基于 Spring 的 Jetspeed-2，着重的介绍了 Jespeed-2 的架构，工作流程以及页面布局等内容。

第五章 用 Portal 技术整合 VMO GUI 应用

5.1 VMO GUI 应用介绍

VMO GUI 是 Platform VM Orchestrator(简称 VMO)的一个子模块，负责给用户提供一个可视化的界面，使用户可以方便地进行操作。图 5.1 描述了 VMO 的结构：

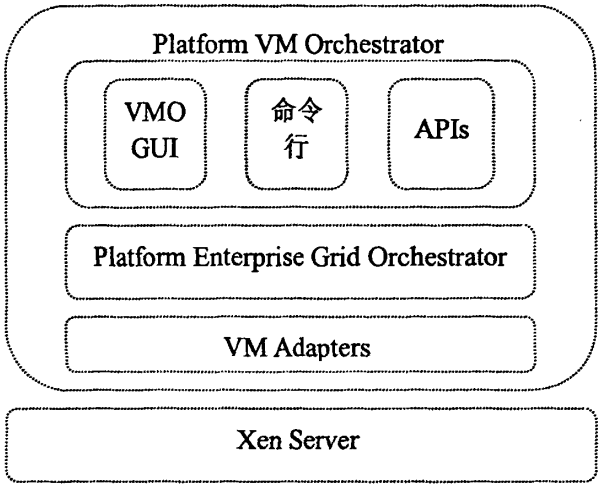

图 5.1 VMO 的结构

VMO 是一个自动的、驱动策略的虚拟环境管理者[5]。它通过动态平衡和实时控制分配虚拟服务的资源来实现资源利用率的最优化，以确保关键应用始终处于运行状态。VMO 制作最好的资源分配策略确保提供高质量的服务。同时提供“failover”机制来使得在某一个服务器上运行的虚拟机，当此服务器不再可达或者不能正常运行时，自动选择合适的地方，重新启动虚拟机。Platform Enterprise Grid Orchestrator(简称 EGO)是对资源进行实时分配与重新分配的底层应用。Xen Server 是一个虚拟机容器。它们之间通过 VM Adapter 联系在一起。

VMO GUI 是一个基于 Web 的应用，提供配置、管理和控制访问所有物理的和虚拟的资源。包括配置机器的使用策略，监控机器的运行状态，管理机器的启动、关闭、克隆等操作。本文主要是对 VMO GUI 进行改造，使得 VMO GUI 既可以被看作是一个 Web 应用，也可以被视为一个 Portlet 应用。

5.1.1 VMO GUI 的页面布局与流程

改造前的 VMO GUI(以下简称为 v1)作为 Web 应用，它的页面由四部分组成：商标信息区、导航区、桌面区、系统消息区。v1 界面的布局为：

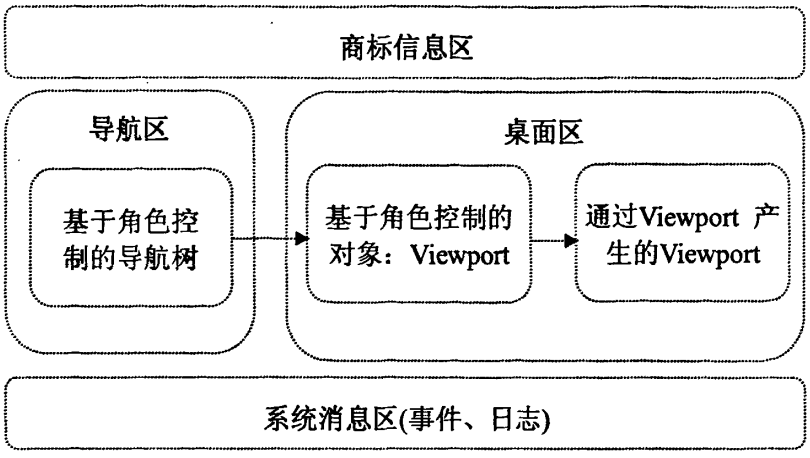

图 5.2 v1 的页面布局

从上图可以看出，桌面区主要是由多个 Viewports 组成。Viewport 不是一个业务对象，而是一个可视化的对象，该对象用来显示和对象的某一类型相关的视图的内容。Viewport 也可以用来显示多种视图，每一个视图是通过点击导航树上的对象被打开的。Viewport 的窗口状态包括最大化、最小化和正常三种状态。这些 Viewports 的状态可以由每个 Viewport 独立控制，也可以由桌面区来统一控制。Viewport 在很大程度上吸取了 Portlet 的概念，这也就简化了 VMO GUI 应用到 Portlet 应用的过渡过程。商标信息区显示公司的商标，以及登录用户的名字和角色等信息。系统消息区显示系统的消息，比如执行事件的消息和日志等。

v1 的页面流程图如下：

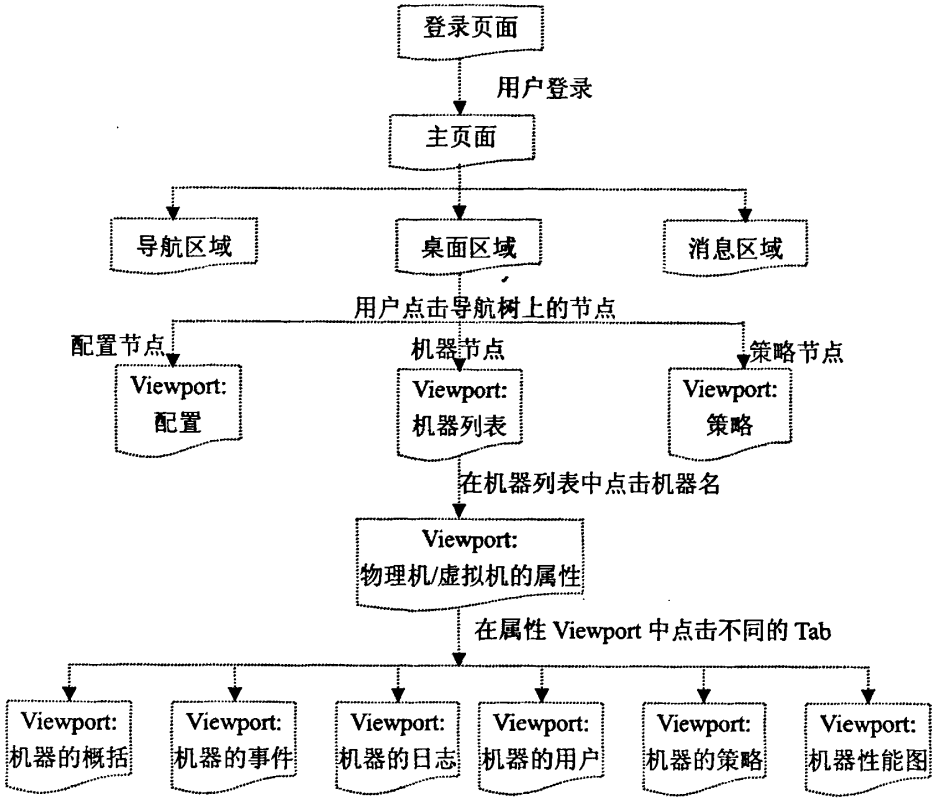

图 5.3 v1 的页面流程图

5.1.2 VMO GUI 的框架

v1 是一个标准的基于 Struts 2 的 Web 应用，运行在 Servlet 容器中。图 5.4 为 v1 的框架图，用户发出的请求通过 Servlet 容器到达相应的 Action，这些 Action 调用 GUI Java API，由 GUI Java API 来调用后台应用程序所提供的 Java API、JNI 或是 Web 服务得到所需的信息。

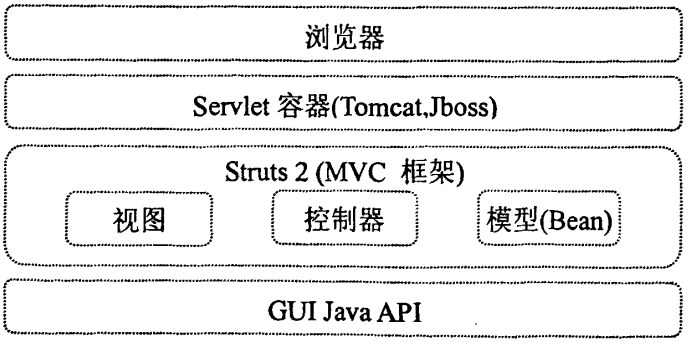

图 5.4 v1 的框架

由于 v1 使用的是 Struts 2 的框架，所以这里首先对 Struts 作简要的介绍。Struts 是基于 Model 2 之上的，是经典的 MVC 框架。它将 MVC 模式“分离显示逻辑和业务逻辑”的能力发挥得淋漓尽致[35]。Struts 的体系结构包括模型、视图和控制器三部分。下面主要介绍 Struts 2 的工作原理，如下图 5.5 所示：

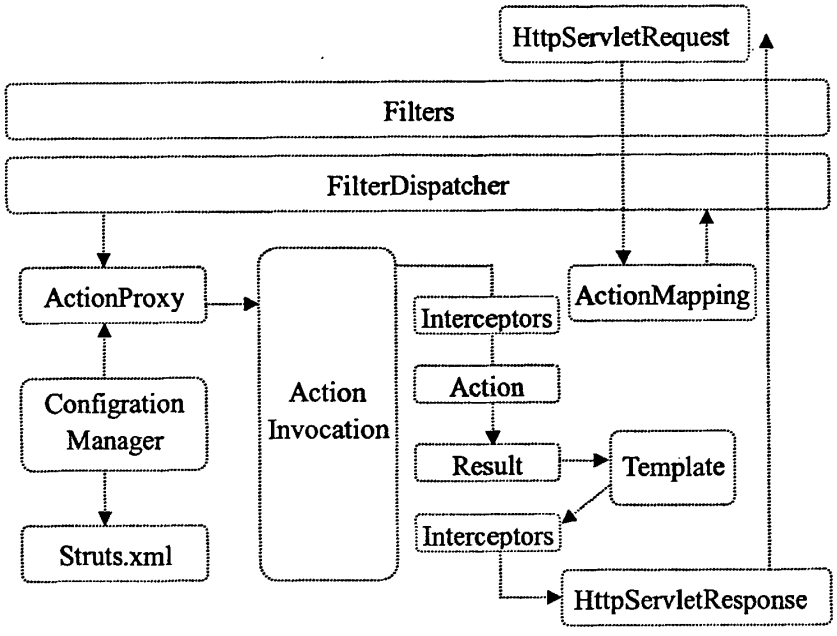

图 5.5 Struts 2 的工作原理[23]

一个请求在 Struts 2 框架中的处理大概分为以下几个步骤：

(1) 客户端初始化一个指向 Servlet 容器的请求。

(2) 这个请求经过一系列的过滤器，最后到达 FilterDispatcher。

(3) 接着 FilterDispatcher 被调用，FilterDispatcher 询问 ActionMapper 来决定这个请求是否需要调用某个 Action 来处理该请求。

(4) 如果 ActionMapper 决定需要调用某个 Action，FilterDispatcher 把该请求交给 ActionProxy 来处理。

(5) ActionProxy 通过 Configuration Manager 查询框架的配置文件，找到需要调用的 Action 类后，创建一个 ActionInvocation 的实例。

(6) ActionInvocation 实例使用命名模式来调用，在调用 Action 的过程前后，涉及到相关拦截器的调用。

(7) 当 Action 执行完毕后，ActionInvocation 负责根据 struts.xml 中的配置，

将处理结果返回。

5.2 Web 应用集成

由于 Struts 2 框架本身提供对 JSR168 的支持，v1 是基于 Struts 2 的 Web 应用，所以可以比较容易的在 v1 的基础上做一些改变来使得 VMO GUI 支持 JSR168 规范。改变后的 VMO GUI(以下简称 v2)既是一个 Web 应用，同时也可以被看作是一个 Portlet 应用。v2 的整体框架为：

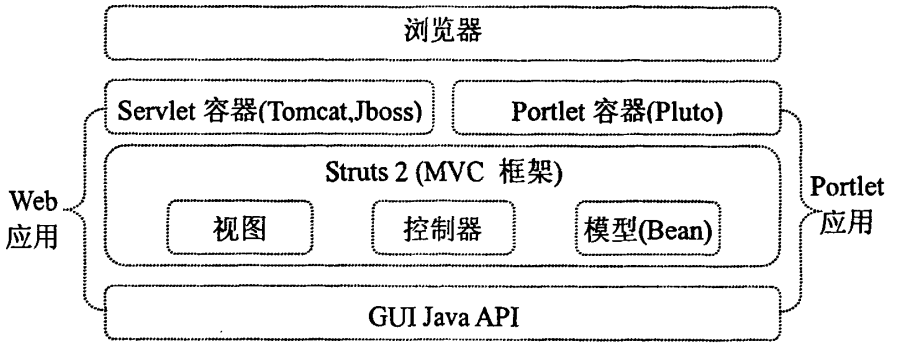

图 5.6 v2 的整体框架

v1 和 v2 的区别在于：v2 中增加了 Portlet 容器，它是 v2 作为 Portlet 应用运行的环境。本文以 Jetspeed-2 作为工作平台，它提供一个标准的 Portlet Container(Pluto[27])的实现。v2 中的每一个 Portlet 对应 v1 中的一个 Viewport，下面以 HostList 这个 Viewport 为例来说明实现 v2 的具体步骤：

(1) 使 Portlet 应用能在 Jetspeed-2 中运行，首先必须在 Web 部署描述符(web.xml)里声明 JetspeedContainerServlet 类。可以看出 Jetspeed-2 是运行在 Servlet 容器中的，同时提供 Portlet 容器的实现。通过 JetspeedContainerServlet 类，最终将控制权交给 Portlet 的具体实现类。web.xml 文件的部分如下所示：

```
<servlet>
  <servlet-name>JetspeedContainer</servlet-name>
  <description>Servlet for Jetspeed Portlet App</description>
  <servlet-class>org.apache.jetspeed.container.JetspeedContainerServlet</servl
   et-class>
  <init-param>
```

```
        <param-name>contextName</param-name>
        <param-value>vmgui</param-value>
      </init-param>
      <load-on-startup>2</load-on-startup>
    </servlet>
    <servlet-mapping>
      <servlet-name>JetspeedContainer</servlet-name>
      <url-pattern>/container/*</url-pattern>
    </servlet-mapping>
```

JetspeedContainerServlet 类根据 contextName 这个参数的值来尝试启动相对应的 Portlet 应用。

(2) 一个 Portlet 应用和一个 Web 应用的区别就在于 Portlet 应用需要增加 Portlets 和 Portlet 部署描述符。Portlet 部署描述符是一个 XML 文件，它的根元素是<portlet-app>元素。<portlet-app>元素包括 Portlet 定义，自定义 Portlet 模式和窗口状态，Portlet 应用支持的用户属性和安全约束等子元素。Portlet 子元素包括 Portlet 的描述，Portlet 的名字，Portlet 的实现类，初始化参数等等。v2 的 portlet.xml 文件的部分内容如下所示：

```
<?xml version="1.0" encoding="UTF-8"?>
<portlet-app … id="vmgui">
    <portlet id="HostList">
        <portlet-name>HostList</portlet-name>
        <display-name">Host list portlet </display-name>
        <portlet-class>org.apache.struts2.portlet.dispatcher.Jsr168Dispatcher</portlet
        -class>
        <init-param>
            <name>viewNamespace</name>
            <value>/host</value>
        </init-param>
        <init-param>
```

```
            <name>defaultViewAction</name>
            <value>hostList</value>
        </init-param>
        ...
    </portlet>
</portlet-app>
```

其中<portlet-app>和<portlet>的 id 属性是独一无二的，它们的值将决定下面将要介绍的 PSML 文件中的值。<portlet-class>的值是具体的一个 Portlet 实现类，该类应该实现 Portlet 这个接口，由于 Portlet API 中提供一个默认的实现了 Portlet 接口的抽象类 GenericPortlet，其它的类只需继承这个抽象类即可。Struts 2 中的 Jsr168Dispatcher 就是继承 GenericPortlet 这个抽象类。当该 Portlet 的请求到达时，Jsr168Dispatcher 会负责处理这个请求。在 Portal 中将 request 分为 actionRequest 和 renderRequest 两个阶段，但作为普通的 Web 应用，只能处理 HttpServletRequest，那么如何处理来自 Portal 页面上的请求呢？基本的思路就是在 actionRequest 阶段不执行任何操作，只是将 action 请求的参数保存起来，在 renderRequest 阶段得到 actionRequest 阶段保存下来的参数进行处理。通过对 Jsr168Dispatcher 进行修改实现上述想法，具体实现如下：

```
...
ActionProxy proxy = factory.createActionProxy(namespace, actionName,
    extraContext);
proxy.setMethod(mapping.getMethod());
request.setAttribute("struts.valueStack",proxy.getInvocation().getStack());
if(PortletActionConstansts.RENDER_PHASE.equals(phase))
    proxy.execute();
if(PortletActionConstants.EVENT_PHASE.equals(phase)) {
    ActionResponse actionResp = (ActionResponse) response;
    actionResp.setRenderParameters((HashMap)extraContext.get("parameters"));
}
...
```

(3) 接下来就是要修改具体的实现类，由于 v2 是一个基于 Struts 2 的 Web 应用，在 Action 阶段所处理的请求都是 ServletRequest，而无法去处理 PortletRequest。所以需要在 Action 中将 PortletRequest 转化为 ServletRequest。具体的 Action 通过实现 ServletRequestAware 接口，利用接口中的 setServletRequest 方法来重写 ServletRequest，该方法的具体实现就是通过 renderRequest 得到 ServletRequest。获得 ServletResponse 做同样的处理即可。具体实现如下：

```
public class BaseAction extends ActionSupport implements ServletRequestAware,
ServletResponseAware {
    protected HttpServletRequest request;
    public void setServletRequest(HttpServletRequest request) {
        if(request != null){  this.request=request; return;}
        Object obj = getContext().get("struts.portlet.request");
        if (obj != null){
            if( obj instanceof RenderRequestImpl) {
                // RenderRequestImpl 是 Pluto 提供的实现类
                RenderRequestImpl renderRequest=(RenderRequestImpl) obj;
                this.request=(HttpServletRequest)renderRequest.getRequest();
            }
        }
    }
```

setServletResponse()方法的实现类似 setServletRequest()方法，这里不做赘述。从上述代码中可以看出，如果是来自 Portal 页面的 PortletRequest，将依赖于 Pluto 提供的实现类来获得 ServletRequest，这就使得 v2 只能在 Pluto 容器中执行，而不能在其它的容器中执行，导致 v2 的移植性很差。如何使 v2 能在所有支持 JSR168 的 Portlet 容器中运行，这是一个迫切需要解决的问题。通过对不同 Portlet 容器的调研，发现大部分 Portlet 容器都会提供获取 ServletRequest 的方法，所以可以使用反射机制来得到 ServletRequest。可以用如下的代码对 setServletRequest 方法进行改进：

```
...
    if (obj != null){
        try{
            Method method = obj.getClass().getMethod(Constants.getRequestString);
            request=(HttpServletRequest)method.invoke(obj);
        } catch(Exception e){
            throw new GuiException("guiException.NoSuchMethod",null);
        }
    }
...
```

Constants.getRequestString 的值是定义在配置文件中的。用户可以根据不同的容器赋予不同的值。例如在 Pluto 容器中 Constants.getRequestString＝getRequest。而在 liferay 中 Constants.getRequestString＝getHttpServletRequest。由于利用反射机制从配置文件中可以获取 Constants.getRequestString 的值，从而提高了该方法的灵活性，也满足 v2 在不同的 Portlet 容器中运行的需求。

(4) JSP 页面的改造。首先，由于 Portal 页面集成 Portlet 产生的片段，所以 Portlet 产生的片段就会有一定的规则。Portlet 产生的 HTML 片段不能使用以下标签[4]：base, body, iframe, frame, frameset, head, html 和 title。所以应该避免 JSP 页面中出现上述标签。其次，v1 中使用自己的 js 和 css 文件来达到某些效果，要想在 v2 中可以沿用 v1 中的这些效果，需要使用动态加载的方法来实现。因为<link>这个标签是在<head>标签中使用的，由于 v1 中的 JSP 在 v2 中不能出现<head>标签，所以只能使用动态加载的方法。例如动态加载 css 的方法如下：

```
var newSS=document.createElement('link');
newSS.rel='stylesheet';
newSS.type='text/css' ;
newSS.href="/vmgui/common/common.css";
document.getElementsByTagName("head")[0].appendChild(newSS);
```

再次，由于一个 Portal 页面可能包含多个 Portlets，这些 Portlets 中可能会

使用名字相同的元素或方法，为了避免由此产生的错误，Portlet 中引入了名字空间这个概念。在元素名或方法名前增加<portlet:namespace/>这个标签来确保不同的 Portlet 使用当前 Portlet 的元素和方法。

(5) 最后通过将具体的 Portlet 加入到 Portal 页面中，这个工作可以通过 Jetspeed-2 提供的可视化工具完成，也可以手工完成。主要是创建 Portal 页面对应的 PSML 文件，在该文件中声明将想要显示在 Portal 页面中的 Portlets。具体实例如下：

```
<page id="/vmgui.psml" hidden="false">
 <title>Virtual Machine Orchestrate Application</title>
 <defaults skin="orange" layout-decorator="tigris" portlet-decorator="tigris"/>
 <fragment id = "t1" type="layout" name="jetspeed-layouts::VelocityTwoColum
   ns">
      <fragment id="v1" type="portlet" name="vmgui::HostList">
            <property name="row" value="0"/>
            <property name="column" value="0"/>
        </fragment>
 </fragment>
</page>
```

该文件中定义了 Portal 页面的标题、皮肤、装饰、布局以及要显示的 Portlet。这里的 vmgui:HostList 就是由<portlet-app>的 id 和<portlet>的 id 组合而成的。通过以上五步就可以成功的将 v2 部署到 Jetspeed-2 中。来自 Portal 页面的请求执行顺序图如下：

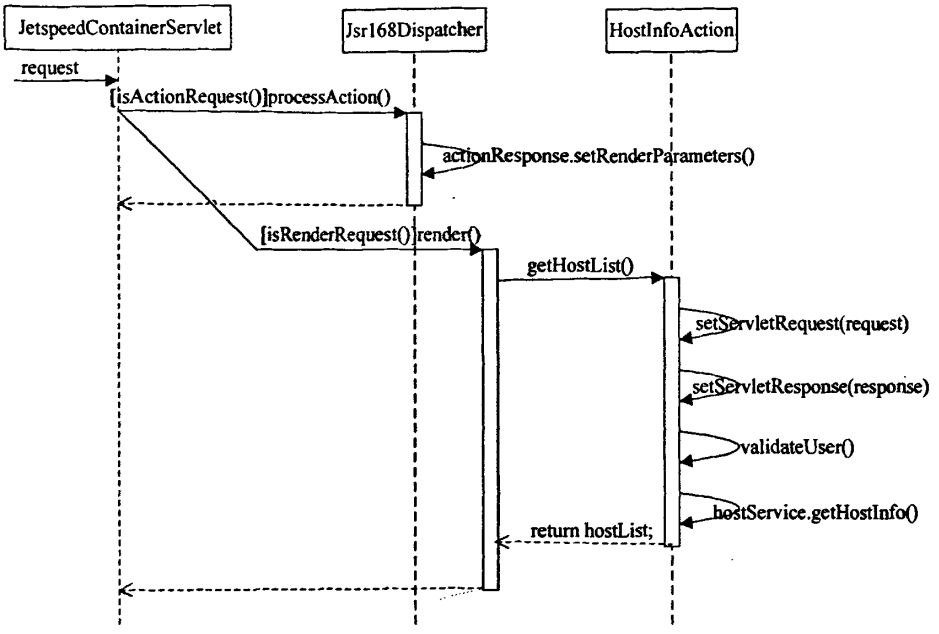

图 5.7 Portal 页面的请求执行顺序图

5.3 实现模拟数据

在实际开发中，由于前台和后台的实现由不同的组遵照预先定义好的接口并行开发，这样前台可以通过模拟后台产生的数据来进行单元测试。其优点有：

1. 彻底将前台实现和后台实现分离开来；

2. 提高了工作效率，由于使用这样的虚拟数据进行测试，确保前台的程序没有问题，这样在与后台程序进行集成，就会减少不必要的时间花费；

3. 通过这样的虚拟数据，可以轻松的给用户演示，而不需要依赖后台的实现。

使用工厂模式[28]来实现这种需求，设计的 UML 类图如下：

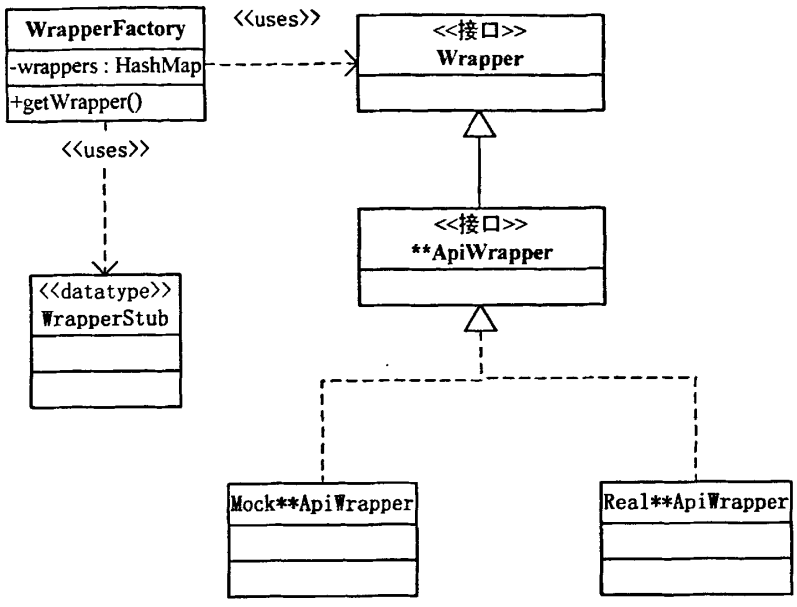

图 5.8 WrapperFactory 的 UML 类图

WrapperStub 类型的定义如下：

```
@Retention(RetentionPolicy.RUNTIME)
public @interface WrapperStub {
    String stub();
}
```

当请求后台的数据时，首先通过调用 WrapperFactory 的 getWrapper()方法来获得相应的 ApiWrapper。在 ApiWrapper 接口中需要声明 WrapperStub 类型，例如@WrapperStub(stub="VMO_STUB_DATA")。VMO_STUB_DATA 的值被设到系统的环境变量中。getWrapper()方法通过传入的类名得到 WrapperStub 类型。然后从环境变量中取得 VMO_STUB_DATA 的值来决定当前需要的是真实数据还是模拟数据，最后通过反射机制将该对象返回。具体的 WrapperFactory 的代码如下：

```
private static HashMap<Class, Object> wrappers = new HashMap<Class, Object>();
public static <T extends Wrapper> T getWrapper(Class clazz)
        throws GuiException {
    T wrapper = (T) wrappers.get(clazz);
    if (wrapper == null) {
```

```
    try {
        ClassLoader cl=clazz.getClassLoader();
        String class_name = "";
        WrapperStub ws=(WrapperStub)clazz.getAnnotation(WrapperStub.class);
        String stub_value=System.getenv(ws.stub());
        String name=clazz.getSimpleName();
        if(stub_value!=null && stub_value.equalsIgnoreCase("yes")){
            class_name=clazz.getName().replaceFirst(name, "Mock".concat(name));
        }else{
            class_name=clazz.getName().replaceFirst(name, "Real".concat(name));
        }
        wrapper=(T)cl.loadClass(class_name).newInstance();
        wrappers.put(clazz,wrapper);
    }catch (java.lang.Throwable e) {
        e.printStackTrace();
        throw new GuiException(null,"System has error");
    }
}
return wrapper;
}
```

从上述代码中可以看出，每一个 Wrapper 只会被实例化一次，第一次实例化后，就将该对象保存到 HashMap 对象中，该对象扮演的一个缓存的角色，当下次调用时，直接从这个缓存里读取即可，这样就避免了类被多次实例化。MockApiWrapper 产生的数据就是模拟的数据，它可以从文件中读取，也可以直接写到程序中。RealApiWrapper 就是真实的从后台获得的数据。

5.4 单点登录

在本文 5.2 节中已经论述了 Viewport 到 Portlet 的转变，接下来则需要解决认证问题。Jetspeed 使用基于角色的方法来管理访问门户网站资源的用户授权。

一个角色把一组许可权与特定的 Portal 资源组合在一起。此组许可权称为角色类型。通过将角色指定到用户或组，来将资源许可权指定给组中的用户。Portal 的资源是层次结构的，该层次结构中每个资源都继承其父资源的角色指定，这种继承可以减少管理的开销[36]。

由于 v1 本身存在认证系统，只有通过认证后才能访问 v1。Portal 系统也有自己的认证系统，所以用户访问 v2 时，首先需要登录 Portal 系统，然后再登录 v2，这样就给用户带来了很多的不便。如何实现只在 Portal 系统登录一次就可以访问 Portal 中的所有应用，这就是通常所说的单点登录。

该系统的单点登录使用本地 Cookie 保存 Token 来完成。如果本地禁用了 Cookie，可以通过 URL 重写的方式传递 Token 的值。Cookie[29]是服务暂时保存到客户端的信息，以便服务器来辨认当前会话。Cookie 提供给浏览器一个空间，以便在一个页面中保存的数据，在下一个页面中的脚本和服务器端程序仍然可以使用。Cookie 中的数据在服务器和客户端自动传递，所以服务器端的程序可以读写保存在客户端的 Cookie 值。Cookie 是由名字、值、生命周期、可见性和安全性这些属性组成的。默认情况下 Cookie 是临时的数据，当其生命周期结束后，Cookie 即随之消失。

Token 是由认证系统返回的，它是一个字符串，主要包括 SessionID、加密后的用户名和密码、认证时间等。SessionID 是由服务器产生，用来唯一标识用户的身份。

v2 的单点登录的具体实现细节为：

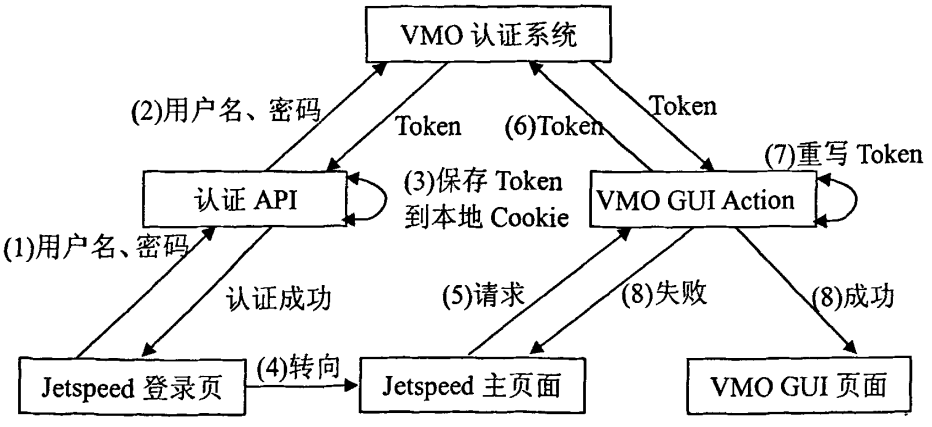

图 5.9 单点登录

在 Jetspeed-2 登录页面中输入用户名和密码，首先 Jetspeed-2 门户进行认证，如果认证成功后，就自动转入 VMO 认证系统来执行认证操作，如果认证成功，则将 VMO 认证系统返回的 Token 保存到当前的 Cookie 中。当用户访问 v2 时，需要到 VMO 认证系统验证当前的 Token 是否有效合法。如果验证通过，则返回一个新的 Token，新的 Token 和过期的 Token 的区别在于更新了访问的时间，这样就避免了用户在访问页面时出现 Token 过期的错误。将新的 Token 保存到 Cookie 中，最后转向当前用户的 v2 页面。

具体的单点登录执行顺序图如下：

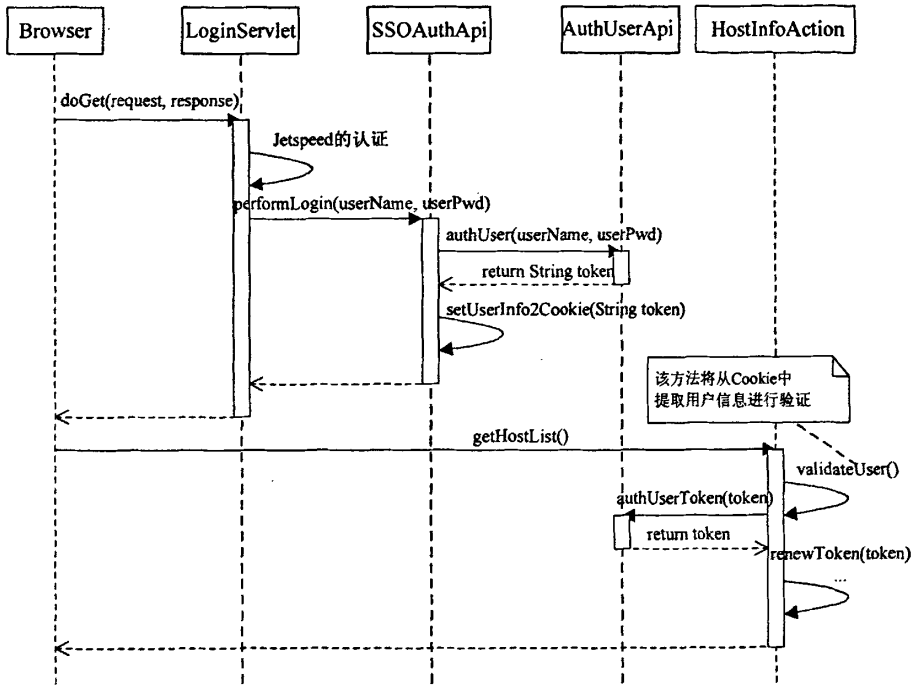

图 5.10 单点登录的顺序图

5.5 Portlet 的通信机制

Portal 页面可以包含多个 Portlet，每一个 Portlet 的请求，在默认没有设置缓存的情况下，整个 Portal 页面的所有 Portlets 都将被刷新。如果一个页面包括大量的数据，每一次请求都要刷新整个页面，这样的响应将极大地影响用户的使用体验。本文尝试通过以下两种方法来改进 Portal 页面的响应速度：1. 改进

Portlet 容器当前的缓存机制；2. 使用 Ajax 技术来提高 Portal 页面的响应速度。

5.5.1 改进 Portlet 容器的缓存机制

Portlet 规范定义了在 Portlet 部署描述符中可以声明一个基于缓存机制的过期值[4]。例如在 portlet.xml 中声明<expiration-cache>60</expiration-cache>，定义了该 Portlet 将使用 Portlet 容器提供的缓存机制，缓存的有效时间为 1 分钟。这种缓存的机制针对不同的用户和不同的 Portlet 提供不同的缓存，也就是说不同的用户不能共享同一个 Portlet 的缓存。

Jetspeed-2 提供了对文件和页面内容的缓存机制。本文主要针对页面内容的缓存机制进行改进。Jetspeed-2 页面内容缓存机制的工作原理是：当用户在 Portlet 部署描述符中声明使用 Jetspeed-2 提供的缓存机制，Portal 页面每加载一个 Portlet 成功后将页面的内容添加到页面内容缓存中，它在缓存中存放的 key 值是由当前登录的用户名和该 Portlet 窗口在 PSML 文件中对应的 Id 值组成。当该页面的某一个 Portlet 发送 renderRequest 时，Portlet 容器认为只需要刷新当前 Portlet 窗口中的数据，同时将该 Portlet 相对应的缓存清空，执行完 render 方法后将新的内容重新添加到缓存中。页面中其它的 Portlet 将根据 key 值从页面内容缓存中获取。当 Portlet 页面的某一个 Portlet 发送一个 actionRequest 时，Portlet 容器认为 Portal 页面的所有 Portlet 都应该被刷新，所以将页面内容缓存中相对应的内容全部清空，待 Portlet 的 render 方法执行完毕后，再将新的内容添加到页面内容缓存中。如果一个页面包含多个 Portlets，且当某个 Portlet 产生 actionRequest，并非其它所有的 Portlet 都需要响应，而是部分需要响应，部分无需响应。为了解决这样的问题，本文提出一种可配置的缓存机制，就是说用户可以配置当某个 Portlet 发送 actionRequest，其它哪些 Portlet 需要响应。由于 v2 系统中的请求大部分属于 actionRequest，所以通过这种机制可以加快页面的响应速度，同时降低服务器的负载量。

可配置的缓存机制通过配置文件来决定哪些 Portlet 需要响应。配置文件的格式如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<Application xmlns="http://www.myPaper.com/2008/portlet_execute_reference"
```

```
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.myPaper.com/2008/portlet_execute_refer
ence portlet_execute_reference.xsd">
    <Action PortletId="HostList" SelfRefresh="false" OtherRefreshPortletId=
        "HostEventList, HostLogList, HostOverview" />
    ...
</Application>
```

从上述文件可以看出 Action 这个元素包含三个属性，其中 PortletId 和 SelfRefresh 在 Schema 文件中规定是必须的，值分别为字符串型和 boolean 型。OtherRefreshPortletId 是可选属性，值为字符串型。PortletId 属性表示当前发送请求的 Portlet 窗口在 PSML 文件中对应的 Id 值。SelfRefresh 属性表示当前 Portlet 是否需要刷新。OtherRefreshPortletId 属性表示当前的 Portlet 发送请求后，哪些 Portlet 需要响应。

使用可配置的缓存机制，当页面上的某个 Portlet 发送 actionRequest 时，Portlet 容器不是将 Portal 页面的所有 Portlet 对应的缓存全部清空，而是从配置文件中获取相关的信息来决定哪些 Portlet 需要响应该请求。如果配置文件中不存在与该 Portlet 相关的信息，则说明该 Portlet 仍然使用默认的缓存机制。使用可配置缓存机制响应用户请求的示例图如下：

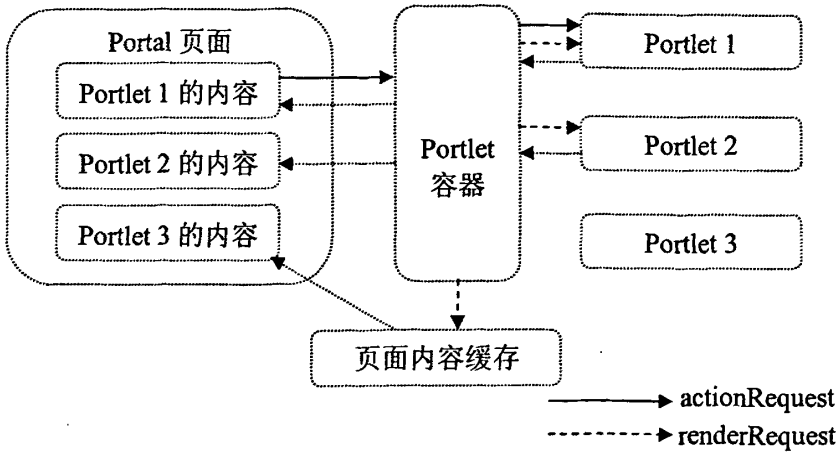

图 5.11 使用可配置的缓存机制响应用户的请求

从上图可以看出，当 Portlet 1 产生 actionRequest，Portlet 1 和 Portlet 2 响应

这次请求，刷新页面的数据，而 Portlet 3 不需要刷新数据，直接从页面缓存中获取数据即可。这样就提高了缓存机制使用的灵活性，同时可以更好地满足用户的需求。

5.5.2 使用 Ajax 技术提高 Portal 页面的响应速度

Ajax [30] (Asynchronous JavaScript and XML) 并不是一项崭新的技术，它是由一些已经存在的技术整合起来的，包括 HTML/XHTML、Dom、JavaScript、CSS、XML、XSLT 和 XMLHTTP。在 Ajax 的解决方案中，所有的这些技术都是可用的，不过只有 HTML/XHTML、Dom 和 JavaScript 是必需的。

下面对传统的 Web 应用程序模型和使用 Ajax 的 Web 应用程序模型进行比较：

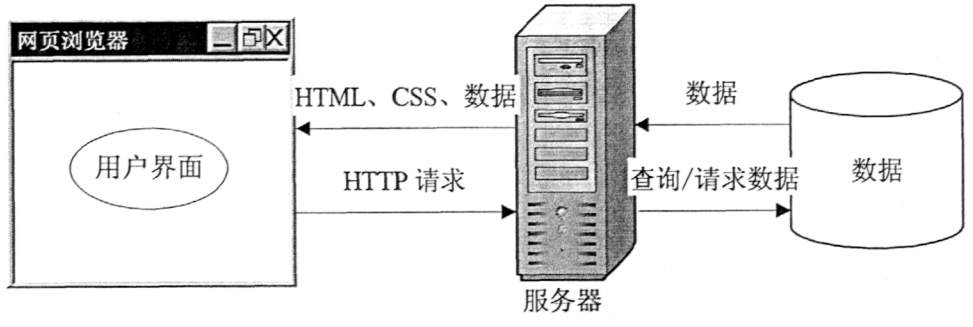

图 5.12 传统的 Web 应用程序模型

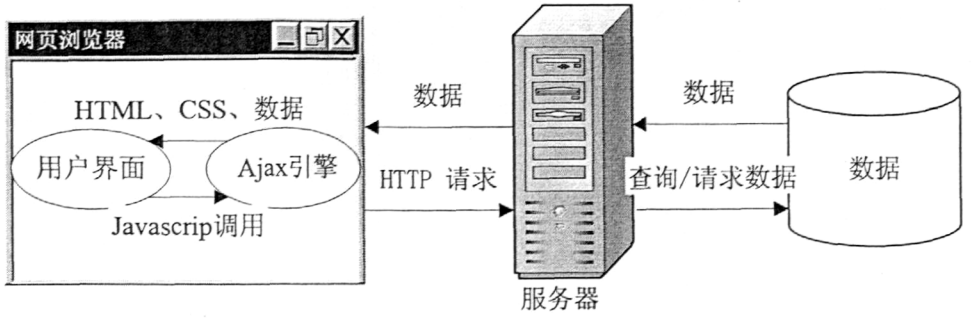

图 5.13 使用 Ajax 技术的 Web 应用程序模型

在传统的 Web 应用程序中，浏览器负责初始化指向服务器的请求，以及处理服务器的响应[31]。所以当用户发送一个请求时，浏览器负责初始化并将该请求发送给服务器，用户只能在一个空白的浏览器窗体和沙漏图标前等待服务器的响应。这就是传统的“请求－等待”的工作模式。使用 Ajax 技术的 Web 应用程序中，在传统的浏览器和服务器中间增加了 Ajax 引擎层。Ajax 引擎负责

初始化请求以及处理响应。此时浏览器得以空闲，可以处理其他的事情。Ajax 的工作原理：

(1) 浏览器捕获用户请求事件。

(2) 通过 Ajax 引擎提交 HTTP 请求到服务器端。

(3) Ajax 引擎处理服务器端的响应。

(4) 部分页面刷新，将服务器的响应返回到页面上，呈现给用户。

Ajax 的核心是 XMLHttp 对象，但是不同的浏览器 XMLHttp 对象是不同的，这就要求用户针对不同的浏览器来创建不同的 XMLHttp 对象。DWR[34]是一个开源的 Java 库，它封装了 Ajax 引擎，支持不同浏览器，使得用户无需费心针对不同的浏览器来创建不同的 XMLHttp 对象。DWR 使用反射机制解析其配置文件中指定的类，然后以 JavaScript 形式分发给客户端。在 JavaScript 中针对类中的每个函数都有一个相似的函数，但增加了一个新的参数，该参数用来指定数据返回时调用的函数，具体的使用本文将在下一节详细介绍。

将 Ajax 技术引入 Portal 中，使得用户可以和服务器异步地进行交互，实现部分页面的刷新，从而提高页面的响应速度，改善了用户的体验。

5.6 Portal 页面的自适应机制

Portal 提供个性化的配置，针对不同的用户可以配置不同的显示内容。对于 Jetspeed-2 的普通用户而言，其配置信息由管理员来管理，普通用户无权编辑和管理 Portal 页面。如果普通用户所在的 Portal 页面包含多个 Portlet，而用户所感兴趣的 Portlet 窗口只有通过拖动页面的滚动条才能获得，这样每次页面刷新，用户都需重复该工作，给用户的使用带来不便。为了使用户更方便地使用 Portlet，本文提出一种自适应的机制。该机制根据用户对 Portlet 的感兴趣程度来决定 Portlet 窗口在页面中的位置，增加了 v2 的友好性。

自适应机制是通过收集用户鼠标在 Portlet 窗口内停留的时间来决定 Portlet 窗口的位置。具体的工作原理是：鼠标移出当前的 Portlet 窗口后，通过 DWR 将当前 Portlet 窗口在 PSML 文件中对应的 Id 值、SessionId、访问时间、停留时间写入文件中。当同一用户下次登录时，读取该文件中的信息来决定页面中 Portlet 窗口的位置。

当用户鼠标停留在某一个 Portlet 窗口中时，监听该窗口的 onmouseover 事件被激活，记录鼠标当前的坐标，每隔一分钟调用 recordStatus()方法来判断当前鼠标是否移动，如果超过某一值(默认是 30 分钟)鼠标位置仍然没有改变，则认为该次操作已经无效。如果鼠标移出当前的 Portlet 窗口，则通过 DWR 将相关信息写入文件。DWR 调用后台写入文件的方法的 Javascript 代码如下：

```
var infoArray = new Array();//储存Portlet窗口的Id值，停留时间和访问时间。
infoArray.push(thisElement.id);
infoArray.push(Math.round(timeInterval/minCountUnit));
infoArray.push(beginTime);
AdaptiveDWRUtil.writeStatus(infoArray, function(data) {
    if( !data ) {// if failed, do something    }
});
```

其中 AdaptiveDWRUtil 是后台的 Java 类，writeStatus 是该 Java 类中的方法。该 Java 类的代码如下：

```
public class AdaptiveDWRUtil extends DWRUtil{
    public boolean writeStatus(String[] infos) {
        HttpSession session = this.getSession();
        String userName = this.getUserName();
        boolean result = true;
        try {
            AdaptiveFileManager.storeHistoryInfo(userName, infos[0], infos[1],
                    infos[2], session.getId(), this.getRequest());
        } catch (Exception e) {
            result = false;
            e.printStackTrace();
        }
        return result;
    }
}
```

其中 Javascript 中 AdaptiveDWRUtil.writeStatus()方法的参数比 Java 类中的 writeStatus()方法的参数多一个，最后一个参数指定处理 Ajax 请求返回时调用的函数。函数的参数 data 对应 Java 类中 result 的值。AdaptiveFileManager. storeHistoryInfo()方法负责将数据写入文件，不同的用户将产生不同的文件。例如用户名为 clusterAdmin 的用户产生的文件名为 History_record_adaptive_cluster Admin.xml。该文件的内容如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<Page xmlns="http://www.myPaper.com/2008/portlet_execute_reference" xmlns:xsi
="http://www.w3.org/2001/XMLSchema-instance"xsi:schemaLocation="http://www
.myPaper.com/2008/portlet_execute_reference portlet_execute_reference.xsd" >
    <Div Id="js_P_1173a1d4a5d_10003_" MaxLength="10">
      <Item SessionId="DBED99769B28EF6F68B089DEA3C65F50"
            TimeDuration="30" AccessDate="2008-01-15 05:40:21"/>
    </Div>
    <Div Id="js_P_11739c2ca17_10000_" MaxLength="10">
      <Item SessionId="DBED99769B28EF6F68B089DEA3C65F50"
          TimeDuration="4" AccessDate="2008-01-15 05:41:14"/>
    </Div>
</Page>
```

Div 元素默认允许最多包含十个 Item 子元素，超过该值将自动删除最旧的数据。将上述文件的内容用一表格来描述可以更清楚地说明文件中保存的信息。

Portlet 窗口对应的 Id	SessionId	停留的时间(分钟)	访问的时间
js_P_1173a1d4a5d_10003_	DBED99769B28EF6F68B089DEA3C65F50	30	2008-01-15 05:40:21
js_P_11739c2ca17_10000_	DBED99769B28EF6F68B089DEA3C65F50	4	2008-01-15 05:41:14

表 5.1 上述文件的内容

从上述表格中可以看出同一个 session 中，用户在两个 Portlet 窗口中停留的时间分别为 30 分钟和 4 分钟。

当用户登录时读取该文件的内容，根据下面的算法[32]改写相应的 PSML 文件。算法的数学表达式如下：

$$V_i = \sum_{j=1}^{n} W_{ij} D_{ij} \tag{1}$$

其中 V_i 代表页面中第 i 个 Div 通过表达式得到的值，由该值来决定相对应的 Portlet 窗口在 Portal 页面中的位置。D_{ij} 是上述文件中第 i 个 Div 中第 j 个 Item 的 TimeDuration 的值，W_{ij} 是第 i 个 Div 中第 j 个 Item 的加权值，它是通过下面的公式得到的：

$$W_s = \frac{1 + Timegap_{oldest} - Timegap_s}{Timegap_{oldest}} \tag{2}$$

$Timegap_{oldest}$ 是当前操作的时间和第 i 个 Div 中所有 Item 的 AccessDate 值的差的最大值。$Timegap_s$ 是当前操作的时间和第 i 个 Div 中第 s 个 Item 的 AccessDate 值的差。显然权值为零是不符合实际情况的，所以在表达式(2)中加 1 来避免出现这种情况。从表达式(2)中可以看出访问时间越久的，它的权值越小，也就是说对当前操作的影响越小。通过表达式(1)计算出来的值来决定 Portlet 窗口在 Portal 页面中的位置，值越大的说明用户对该 Portlet 窗口的感兴趣程度越高，所以显示在 Portal 页面更靠前的位置。

通过这种自适应的机制来改善用户的体验，如果用户经常在某一个 Portlet 中进行操作，即使该 Portlet 窗口被放置到 Portal 页面靠后的位置，它将会在用户登录时自动的移动到靠前的位置。Portlet 窗口的位置将随着用户兴趣度的不同被摆放到不同的位置，而不需要用户手动摆放位置。

5.7 发布本地的 Portlets 供远程用户使用

主要使用 WSRP4J 中提供的模块来完成将本地部署的 v2 应用中的 Portlets 作为 WSRP 服务发布，通过 WSRP 提供的代理模块将本地发布的 WSRP 服务提供给远程的用户。从而实现了部署到本地的 v2 应用可以在远程使用，实现了 v2 在不同 Portlet 容器中的共享。

5.7.1 WSRP4J 项目介绍

WSRP4J 是 Apache 的一个开源项目，实现了 WSRP 规范。WSRP4J 定义了一个开放的架构，提供基于 Apache Tomcat、Apache Axis 和 Apache Pluto 的 WSRP 服务。WSRP4J 是基于 Apache Pluto Portlet 容器的，提供一个 WSRP 生产者的实现和两个消费者的实现。本文主要负责集成 WSRP 生产者和名为 Pluto Proxy Portlet 的消费者。

WSRP 生产者负责将 v2 中的 Portlets 作为 WSRP 服务发布。Proxy Portlet 是一个符合 JSR168 规范的 Portlet，负责将 WSRP 生产者发布的 WSRP 服务代理到本地。

WSRP4J 的工作原理：通过 Proxy Portlet 将远程的 Portlet 代理到本地，本地 Portal 服务器负责将代理到本地的 Portlet 集成到 Portal 页面中。当用户和这些远程的 Portlet 通信时，Proxy Portlet 将所有的参数组成请求包发送到远程的 WSRP 服务。WSRP 服务拆解请求包中的所有信息并调用相应的远程 Portlet 进行处理。将处理的结果组成响应包返回给 Proxy Portlet。

WSRP4J 所依赖的 Web 服务技术如下：

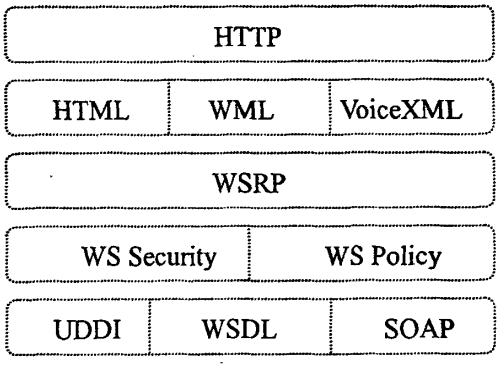

图 5.14 WSRP4J 依赖的 Web 服务技术

VoiceXML(Voice eXtensible Markup Language)是由 VoiceXML 论坛制定的通过电话访问 Internet 网络的标准，是 W3C 定义的可扩展标记语言(XML)的一种扩展，根据播放的提示信息、口述的命令、要记录和识别的语音或按键音输入，实现人和计算机之间的交互对话。WML(无线标记语言)，由它写出来的文件专门用来在手机等一些无线终端显示屏上显示，供人们阅读，并且同样也可以向使用者提供人机交互界面，接受使用者输入查询内容等信息，然后向使用

者返回想要获得的信息。WS Security 和 WS Policy 分别代表 Web 服务安全和 Web 服务策略。UDDI（统一描述、发现和集成协议）用来发布、发现和绑定 WSRP 服务。WSDL(Web 服务描述语言)用来描述 WSRP 服务。SOAP(简单对象访问协议)，通过 SOAP 协议来调用 WSRP 服务。

5.7.2 v2 远程共享的设计与实现

将 v2 中的 Portlets 作为 WSRP 服务发布，供远程的用户使用。实现这一功能的设计图如下：

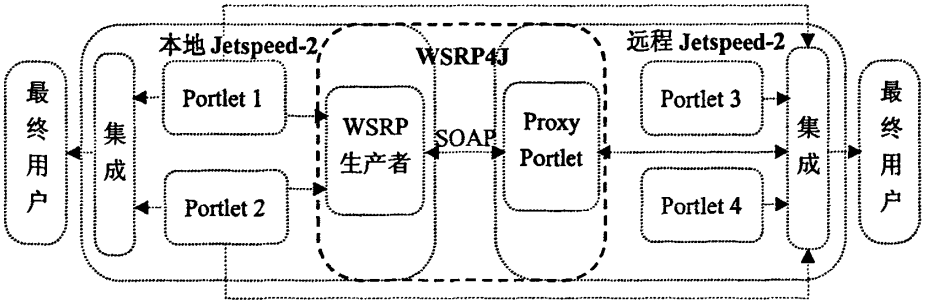

图 5.15 v2 远程共享的设计图

从上图可以看出发布 WSRP 服务供远程使用分为两个阶段：1. 本地的 Jetspeed-2 中集成 WSRP 生产者，负责将 v2 的 Portlets 作为 WSRP 服务发布；2. 远程的 Jetspeed-2 中集成 Pluto Proxy Portlet，负责将本地的 WSRP 服务代理到远程，供远程用户使用。

这里将首先陈述如何将本地的 Portlet 作为 WSRP 服务发布：1. 将 WSRP 生产者模块 wsrp4j-producer 部署到 Jetspeed-2 容器中；2. 将需要发布的 Portlets 配置到 wsrp-producer 的配置文件中：1).在 wsrp4j-config.properties 文件中声明作为 WSRP 服务的应用所在的目录 wsrp4j.producer.configapp.webapps.dir=D:/Apache/Jetspeed-2.1/webapps。2).在 portletcontexts.txt 文件中指定具体的应用名，例如 /vmgui。3). 具体发布为 WSRP 服务的 Portlet 配置：

```
<?xml version="1.0" encoding="UTF-8"?>
<portlet-entity-registry>
    <application id="vmgui">
        <definition-id>vmgui</definition-id>
```

```
        <portlet id="192.168.0.10_1196688889648_0">
            <definition-id>vmgui.HostList</definition-id>
        </portlet>
        ……
    </application>
</portlet-entity-registry>
```

其中<application>的元素<definition-id>是 Portlet 所在的 Web 应用名。<portlet>的元素<definition-id>是由 Web 应用名和 Portlet 的名字组成。3. 声明 WSRP 规范中定义的四个 Web 服务的接口：

```
<wsdl:service name="WSRPService">
    <wsdl:port binding="bind:WSRP_v1_Markup_Binding_SOAP"
name="WSRPBaseService">
            <soap:address location="http://localhost:8080/wsrp4j-producer/WSRP4J
                    Producer/WSRPBaseService"/>
    </wsdl:port>
    <wsdl:port binding="bind:WSRP_v1_ServiceDescription_Binding_SOAP"
        name="WSRPServiceDescriptionService">
        <soap:address location="http://localhost:8080/wsrp4j-producer/
            WSRP4JProducer/WSRPServiceDescriptionService"/>
     </wsdl:port>
    <wsdl:port binding="bind:WSRP_v1_Registration_Binding_SOAP"
        name="WSRPRegistrationService">
        <soap:address location="http://localhost:8080/wsrp4j-producer/
            WSRP4JProducer/WSRPRegistrationService"/>
    </wsdl:port>
    <wsdl:port binding="bind:WSRP_v1_PortletManagement_Binding_SOAP"
        name="WSRPPortletManagementService">
        <soap:address location="http://localhost:8080/wsrp4j-producer/
            WSRP4JProducer/WSRPPortletManagementService"/>
```

</wsdl:port>

</wsdl:service>

通过上面的配置就可以将HostList这个Portlet发布成WSRP服务。

接下来阐述如何在本地使用远程生产者发布的WSRP服务：1. 将WSRP4J的模块wsrp4j-proxyportlet部署到本地Jetspeed-2容器中。2. 在wsrp4j-proxyportlet模块中声明与WSRP生产者相关的参数：<markup-interface-url> <service-description-interface-url> <registration-interface-url> <portlet-management-interface-url>的值分别对应远程WSRP生产者声明的WSRP规范中定义的四个接口的<soap:address>的location值。通过这些配置，本地的用户就可以在自己的Portal页面中添加远程的Portlet，此时本地Portal通过proxyportlet向远程WSRP服务发出复制远程Portlet的请求，WSRP服务器将分配新的Portlet实例到Portal系统中，这样本地的用户就可以对远程Portlet的实例进行个性化定制。

通过将WSRP4J的生产者和代理模块集成到Jetspeed-2中，完成了发布本地的Portlets为WSRP服务供远程用户使用的功能。

5.8 本章小结

本章首先介绍了VMO GUI这个Web应用，包括它使用的技术、提供的功能以及GUI界面的布局等内容。接下来说明如何将VMO GUI应用集成到Jetspeed-2中，具体介绍了集成的步骤以及在实际集成过程中遇到的问题。第三节通过使用工厂模式和环境变量来实现虚拟数据和真实数据并存，这个策略可以有效的提高工作效率。在第四节中介绍了如何使用Cookie技术实现单点登录，避免多次登录给用户使用带来的不便。接下来的两节缓存机制的优化和Portal页面的自适应机制的提出，改善了用户访问Portal的体验。最后通过集成WSRP4J，使本地的Portlet可以作为WSRP服务发布，远程用户可以轻松的使用这些WSRP服务，而不需要在远程的Portlet容器中部署，从而实现了Portlet的共享。

第六章 系统的测试与结果

6.1 系统的功能测试

v2 作为一个普通的 Web 应用，可以独立的部署到 Tomcat 容器中，正常启动 Tomcat 后，通过浏览器访问该应用首先需要用户登录，登录页面如下：

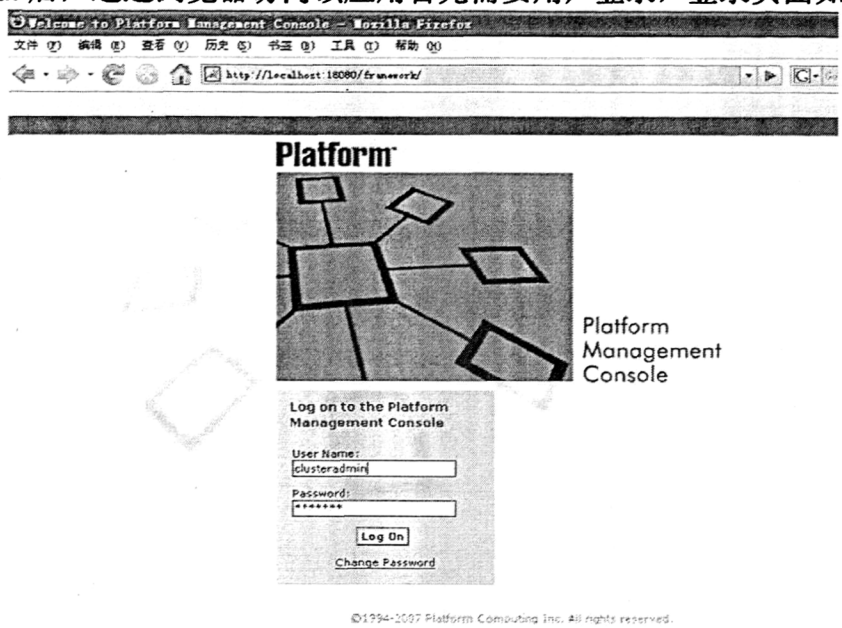

图 6.1 v2 作为独立的 Web 应用的登录页面

正确登录后，将进入 v2 的主页面：

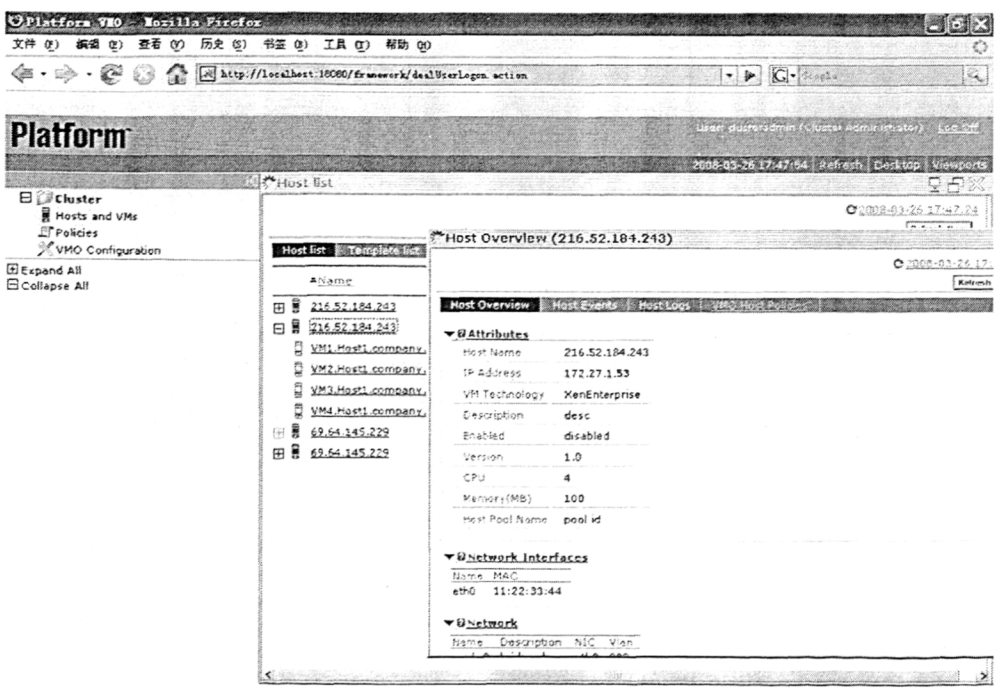

图 6.2 v2 的主页面

从图 6.2 中可以看出，Web 页面分为三部分，上文介绍的页面分为四部分，这里最小化系统消息区。所以直观的可以看出页面分商标信息区、导航区和桌面区三部分。桌面区显示的两个 Viewport，一个为 HostList Viewport，是通过导航区产生的。另一个是 Host Overview Viewport，是通过 HostList Viewport 产生的。

将 v2 部署到 Jetspeed-2 中，成功启动后，可以看到 Jetspeed-2 的登录页面：

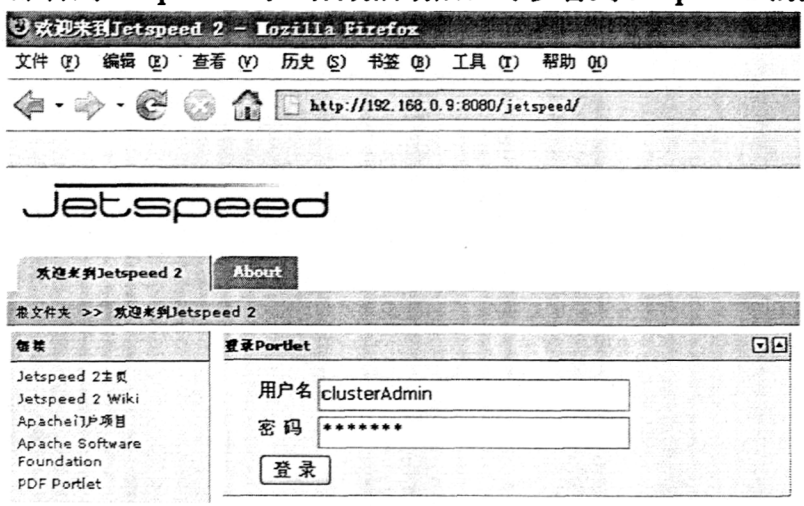

图 6.3 Jetspeed-2 的登录页面

正确的登录后，将 Token 保存到浏览器的 Cookie 中，页面执行跳转，这时用户可以看到 Virtual Machine Orchestarte Application 这个选项卡。该选项卡显示所有 v2 集成到 Jetspeed-2 中的页面：

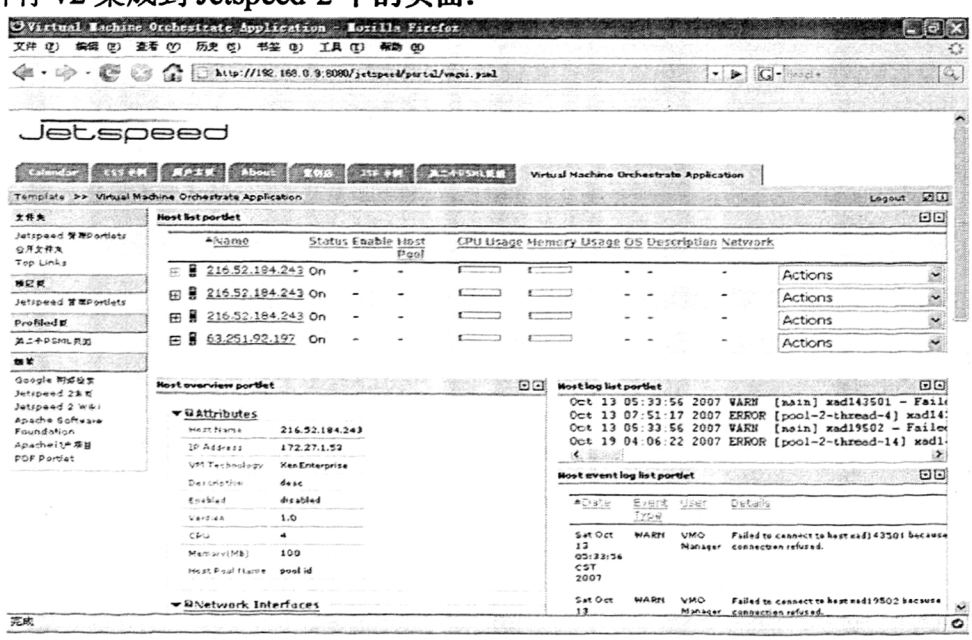

图 6.4 v2 整合到到 Jetspeed-2 中的结果

从上图可以看出，已经将 v2 成功的部署到 Jetspeed-2 中，这就实现了 Web 应用的整合，在某种程度上消除了信息孤岛。

通过改写配置文件中 Constants.getRequestString＝getHttpServletRequest，然后修改 web.xml 来声明在 Liferay 中运行所需要的 Servlet，就可以在 Liferay 中得到 ServletRequest，也就是说 v2 可以部署到 Liferay 中，部署到 Liferay 中的 v2 的页面如下：

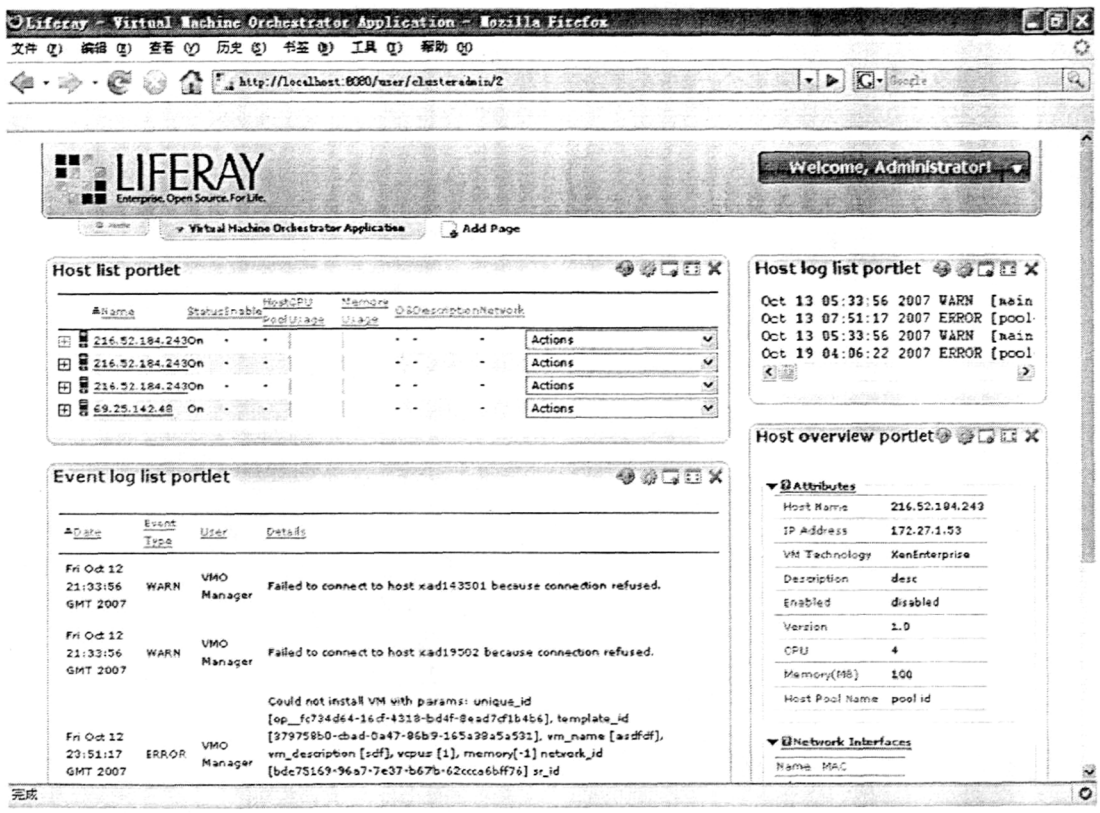

图 6.5 v2 部署到 Liferay 中的结果

将 WSRP 生产者模块部署到 Jetspeed-2 中，来负责将 v2 中的 Portlets 发布成 WSRP 服务，供远程的门户访问。正确部署 WSRP 生产者模块到 Jetspeed-2 中，访问 WSRP 生产者提供的 WSRP 服务描述为：

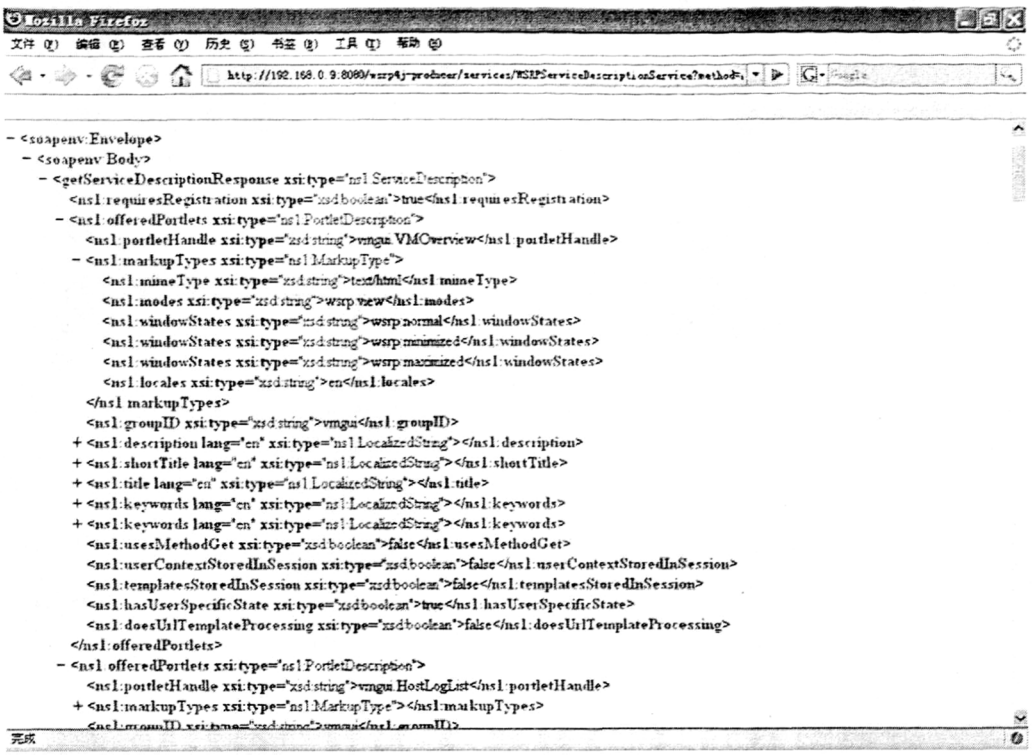

图 6.6 WSRP 服务描述

该图中显示 WSRP 服务的描述，包括是否要求注册、提供的 Portlets 描述、允许的 Portlet 模式、窗口状态等等。消费者通过这些信息来得到远程的目标 Portlets。

将 WSRP 的代理 Portlet 模块正确的配置、部署到远程的 Jetspeed-2 中。下图 6.7 显示的是 WSRP 代理模块提供的两个 Portlet，分别为代理的生产者列表和已经代理的 Portlet。

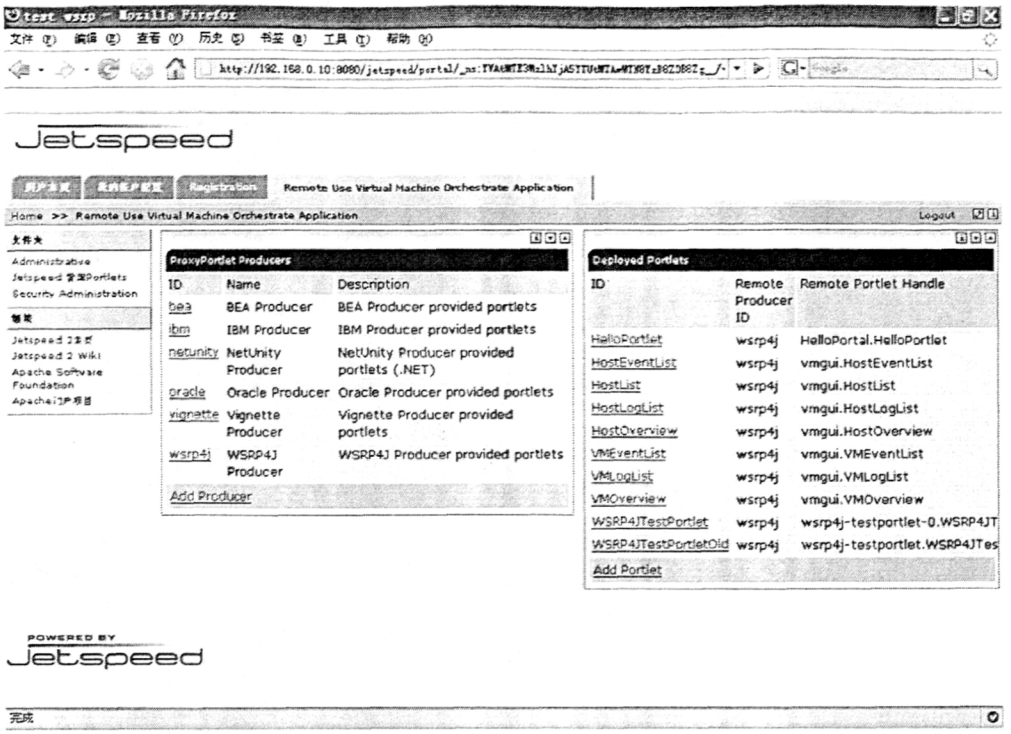

图 6.7 WSRP 模块提供的两个 Portlet

上图中代理生产者 Portlet 窗口中列出了可以代理的生产者有 BEA、IBM、WSRP4J 等生产者。代理 Portlet 窗口中列出了已经代理的 Portlet 的信息，可以看出这些 Portlet 的生产者都是 WSRP4J，也就是生产者提供的 WSRP 服务。

图 6.8 显示 WSRP4J 的信息，包括服务描述、具体服务的 URL 等信息。

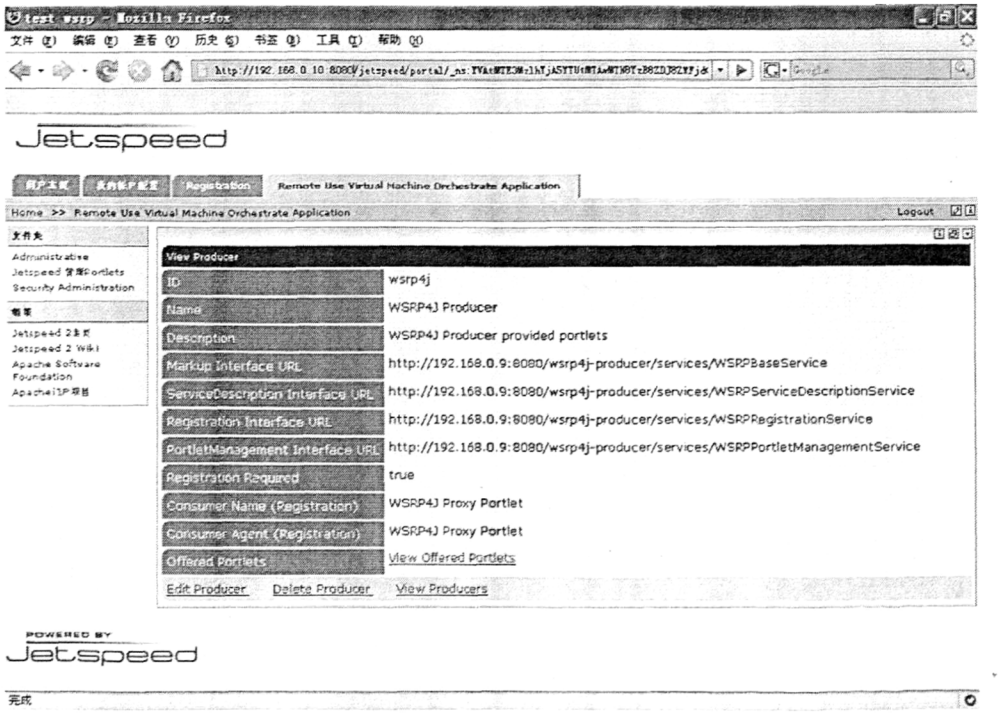

图 6.8 WSRP4J 的信息

图 6.9 显示本地提供的 WSRP 服务：

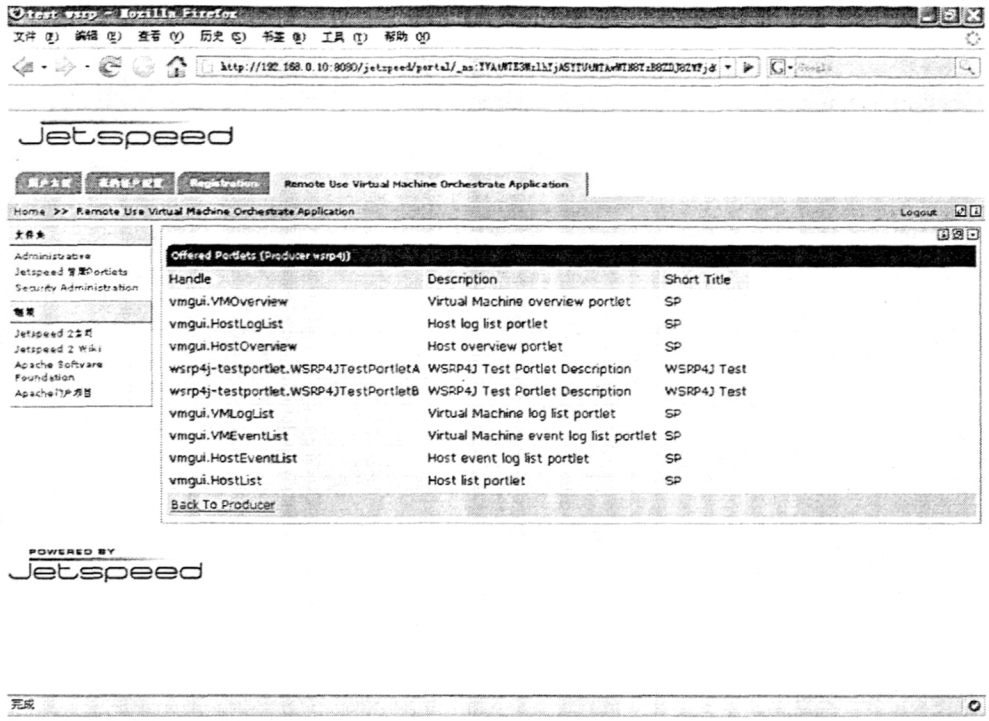

图 6.9 本地提供的 WSRP 服务

将 WSRP 代理模块已经代理到本地的 Portlet 添加到 Portal 页面中，正确登录远程 Jetspeed-2 后，在远程 Jetspeed-2 中显示的结果如下：

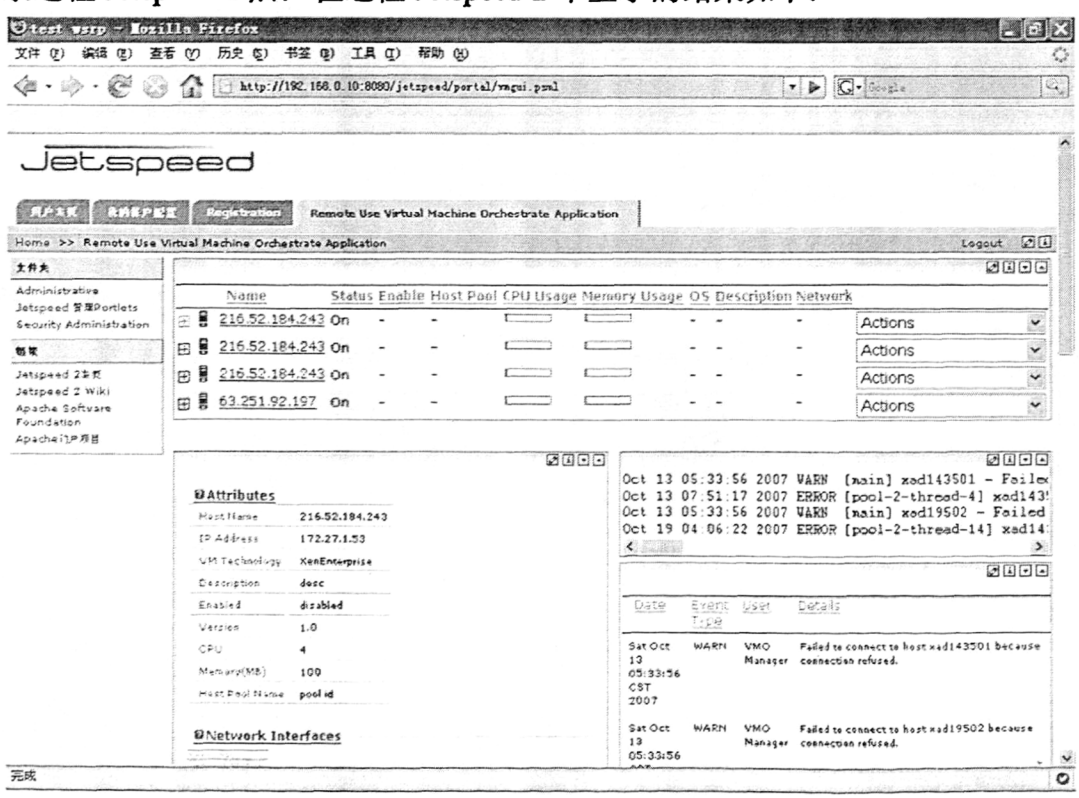

图 6.10 远程 Jetspeed-2 中集成本地提供的 WSRP 服务

这样就实现了远程 Jetspeed-2 中无需部署 v2 应用，而是通过本地 Jetspeed-2 中发布的 WSRP 服务，使得远程 Jetspeed-2 中可以直接使用这些 Portlet。因此也就实现了共享 Portlet 的功能。

6.2 系统的性能测试

本节主要从页面的响应时间来说明将 v1 中的多个 Viewport 同时添加到一个 Portal 页面中，其响应时间是否会增加。

测试的环境：双核 1.73GHz 内存 1024M

测试所使用的浏览器：Firefox 2.0

测试所使用的工具(用来计算页面的响应时间)：FasterFox 2.0.0

测试的结果如下：

第n次响应时间 / 测试页面	1	2	3	4	5
Hostlist	0.999s	0.983s	0.951s	0.967s	0.951s
HostOverview	0.499s	0.468s	0.530s	0.468s	0.484s
HostEventList	0.546s	0.484s	0.530s	0.562s	0.577s
HostLogList	0.680s	0.593s	0.515s	0.499s	0.530s
VMOverview	0.546s	0.577s	0.546s	0.593s	0.562s
VMEventList	0.515s	0.500s	0.561s	0.514s	0.578s
VMLogList	0.484s	0.500s	0.562s	0.499s	0.530s
上述所有页面响应时间的总和	4.269s	4.105s	4.195s	4.102s	4.212s
Portal 页面(整合了上述所有页面)	2.481s	2.246s	2.527s	2.386s	2.293s

表 6.1 性能测试结果

从测试结果可以看出，Portal 页面中包含多个 Viewport，但作为一次请求所需要花费的时间比每个 Viewport 单独发送请求所花费的时间的总和少很多。这可以说明这种基于 Portal 技术的集成方法是有效的。

第七章 论文总结与展望

7.1 论文总结

本论文分析了企业遇到的应用集成问题，使用基于企业门户来完成企业应用集成的方案。重点介绍了和企业门户相关的两个规范—JSR168 和 WSRP，并在基于这两个标准的基础上将现有的基于 Struts 2 的 Web 应用集成到基于 JSR168 规范的 Portlet 容器中。简要的介绍了基于 JSR168 规范的企业 Portal 平台—Jetspeed-2，并在 Jetspeed-2 的基础上，提出了应用集成的具体方法，阐述了集成的关键技术。在将现有系统集成到 Jetspeed-2 的基础上，初步实现了单点登录功能。探索 Portlet 的通信机制，通过优化现有的缓存机制和使用 Ajax 技术来改善现有 Portlet 的通信机制。提出一种自适应的机制，根据用户的兴趣排列 Portlet 窗口在 Portal 页面中的位置，使得用户在一段时间内感兴趣的 Portlet 窗口永远处于用户最方便获取的位置。根据 UCD(user-centered design)的观点来说，这种方法可以给用户带来良好的体验。

将 Struts 2 应用集成到 Jetspeed-2 的过程中，提出了结合工厂模式和环境变量来实现虚拟和真实数据并存，有效地提高工作效率。

本文的最后介绍了一个 Apache 组织开发的符合 WSRP 规范的开源的 WSRP4J 项目，将该项目中的两个模块分别集成到本地和远程的 Jetspeed-2 中，在本地实现了将本地部署的 Portlet 发布成 WSRP 服务的功能，在远程可以使用这些本地发布的 WSRP 服务，用户可以将本地的 WSRP 服务添加到远程的 Portal 页面中。真正实现了部署到 Portlet 容器中的应用的共享，从而使得一些以前不能通过部署而添加到 Portal 页面中的应用，可以通过 WSRP 来添加到 Portal 页面中。

7.2 展望

自从 Portal 概念的提出以后，各大厂商就纷纷推出自己的 Portal 产品。JSR168 和 WSRP 规范的推出使得 Portal 技术的发展趋于规范化，而作为 Portlet 规范的下一个版本 JSR286 也即将推出，将加快 Portal 技术的发展步伐。随着

Portal 技术的发展，企业应用集成也将得到进一步的发展。现在基于 Java 技术的 Portal 产品无法整合.Net 的应用，如何实现真正的跨平台、跨语言的集成，将成为 Portal 产品和应用集成研究发展的新的方向。

下一步的工作：

1. 从目前来看，Portal 技术还是一项比较新的技术，尽管有两个标准已经推出，但不是所有的 Portal 产品都能完整地实现这些规范。对这些规范进行深入的研究以便更好的应用到企业应用集成中。

2. 深入地研究应用集成，寻求一种通用的方法可以将基于不能语言的系统集成到 Portal 产品中，从而实现真正的跨平台工作。

3. 并非所有的 Portlet 容器都提供缓存机制，所以需要在现有的 Web 应用中增加缓存机制，作为一个独立的 Web 应用，缓存机制仍然是非常有用的，它可以提高响应的速度，降低服务器的负荷。

4. 更多 Portal 产品的调研，包括 Weblogic 和 Websphere，将现有基于 Struts 2 的 Web 应用部署到这些产品中。

5. 寻找一种更加安全、可靠的方法来实现单点登录功能，本文使用的基于 Cookie 的方法并非最佳选择，因为在处理远程应用认证问题上存在局限性。

参考文献

[1] 谢小轩，张浩，企业应用集成综述[J]，计算机工程与应用，2002，22：1-5.

[2] 吴敏萍，企业应用集成和企业门户技术研究[J]，电信网技术，2006，1：59-62.

[3] B.Wangler，S.J Paheerathan．Horizontal and Vertical Integration of Organization IT Systems[EB/OL]，http://www.dsv.su.se/~perjons/newhv2.pdf.

[4] Java Portlet Specification 1.0[S]，http://jcp.org/en/jsr/detail?id= 168.

[5] Platform VMO[EB/OL]，http://www.platform.com/Products/platform-vm-orchestrator.

[6] Sybase企业应用集成解决方案[EB/OL]，http://www.mie168.com/EAI.

[7] 周宏生，基于Web服务的企业应用集成的研究与应用[D]，大连理工大学，2005.3.

[8] 黄安安，王丽芳等，基于ESB的企业应用集成研究[J]，微计算机信息，2007，28(9)：965-970.

[9] Jetspeed-2 官方网站[EB/OL]，http://portals.apache.org/jetspeed-2/.

[10]Liferay 官方网站[EB/OL]，http://www.liferay.com.

[11]eXo Platform 官方网站[EB/OL]，http://www.exoplatform.com/.

[12]uPortal 官方网站[EB/OL]，http://uportal.org/.

[13]徐家俊，万涛，推开“门户”看究竟[J]，IT 经理世界，2002，19.

[14]郭丹，基于 Ajax 的企业信息门户系统的设计与实现[D]，中山大学，2007.5.

[15]Eric Knorr，The New Enterprise Portal[EB/OL]，http://www.infoworld.com/article/04/01/09/02FEportal _1.html，2004，1.

[16]Servlet Specification v2.3[S]，http://jcp.org/aboutJava/communityprocess/first/jsr053/index.html.

[17]WSRP 1.0 stamdard specification document[S]，http://www.oasis-open.org/committees/download.php/3343/oasis-200304-wsrp-specification-1.0.pdf.

[18]WSRP Primer 1.0[EB/OL]，http://www.oasis-open.org/committees/download.php/10539/wsrp-primer-1.0.html.

[19]WSRP4J 官方网站[EB/OL]，http://portals.apache.org/wsrp4j.

[20]Stefan Hepper，Peter Fischer，Stephan Hesmer，Portlets and Apache Portals[M]. http://www.manning.com.

[21]Jetspeed-1 官方网站[EB/OL]，http://portals.apache.org/jetspeed-1/.

[22]Designing Enterprise Applications with the J2EE™ Platform[EB/OL]，4.4 Web-Tier Application Framework Design，http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/web-tier/web-tier5.html#1080752.

[23]Apache Software Foundation--Struts 2[EB/OL]，http://struts.apache.org/2.x/.

[24]林信良著，Spring 技术手册[M]，北京：电子工业出版社，2006.6.

[25]廖健，Apache 门户项目组介绍[EB/OL]，http://www.ibm.com/developerworks/cn/opensource/os-apache-portal/index.html，2006.11.

[26]Jetspeed-2 的工作流程[EB/OL]，http://portals.apache.org/jetspeed-2/guides/guide-pipeline.html.

[27]Pluto 官方站[EB/OL]，http://portals.apache.org/pluto.

[28]Erich Gamma，Richard Helm，Ralph Johnson，John Vlissides，Design Patterns：Elements of Reusable Object-Oriented Software[M]，北京：机械工业出版社，2002.3.

[29]David Flanagan 著，李强等译，Javascript 权威指南[M]，北京：机械工业出版社，2007，8.

[30]Ryan Asleson, Nathaniel T.Schutta，Ajax基础教程[M]，北京：人民邮电出版社，2006.

[31]Nicholas C.Zakas，Jeremy McPeak，Joe Fawcett，Ajax高级程序设计[M]，北京：人民邮电出版社，2006.

[32]Jon T.s Quah，Vincent L.H. Seet，Adaptive WAP portals [J]，Electornic Commerce Research and Applications，2007，9.

[33]徐碧云，王志坚，张少柏，企业信息门户关键技术研究[J]，计算机应用研究，2005：142-144.

[34]DWR 官方网站[EB/OL]，http://getahead.ltd.uk/dwr/.

[35]胡一栋，陈福生，朱炜，基于Struts框架的Portal研究和应用[J]，计算机工程与科学，2006(8)：19-21.

[36]代翀，基于Portal的企业信息集成[D]，北京交通大学，2006.12.

[37]Integration at the Glass and the 80/20 Point[EB/OL]，http://www.manageability.org/blog/stuff/integration-at-the-glass-and-80-20-point.

致 谢

论文完成之际，也是我即将离开学校的日子。回顾这三年的学习与生活，我不禁感慨万千。在这里，有收获的喜悦，有对校园的眷恋，也有对未来专业知识的渴求，但更多的是对给予我无私帮助的老师和同学的无限感激之情。

首先，我要感谢我的导师葛玮副教授。葛老师治学严谨，思维敏捷，三年来无论是从学习上还是生活上都给了我们无微不至的关心。他不仅教了我许多专业知识，还教我做人的道理。尤其在论文写作过程中，他提出了许多中肯的建议，给予了无微不至的指导，才使得论文的顺利完成。

我要感谢郝克刚教授。郝老师渊博的学术知识让我敬仰，每当我遇到问题的时候，郝老师简短的几句话常常使我茅塞顿开。他不仅使我懂得如何进行学术研究，还使我领悟到做人的道理。郝老师严谨的治学态度、勤勉不息的工作精神将使我终生受益。

感谢我的家人在学习期间对我的支持、帮助。特别要感谢我的女友贾媛在这段繁忙的日子里给予我极大的帮助。感谢我的同学，能够与你们一起度过三年的学习生活，我感到十分快乐和荣幸。

感谢北京 Platform 西安分公司给我提供的社会实践机会。

最后，对评审论文的各位专家、老师表示衷心的感谢！

攻读硕士学位期间发表的论文

1. 《基于Jetspeed实现Web应用到Portlet应用的转变》，杨春林 葛玮 郝克刚，计算机应用与软件，2008.9