Shale 框架构建 Web 应用

张南平，陈大鹏
（武汉理工大学 计算机科学与技术学院，湖北 武汉 430070）

摘 要：Shale 是新型的 Web 应用框架，它将整体框架划分为多个独立的层，各层间松散耦合负责不同的任务，根据需求灵活组合实现具体应用。阐述了 Shale 框架基本思想和主要特性，在重点分析研究 Shale 视图控制器和对话管理器的基础上，通过具体实例详述了 Shale 框架在 Web 应用中的实现，采用代码片断对关键技术点做了明确说明。

关键词：MVC；Shale；框架；生命周期；视图控制器；对话管理器
中图分类号：TP311.1 **文献标识码**：A **文章编号**：1673－629X(2007)04－0247－03

Web Application Implemented by Shale Framework

ZHANG Nan-ping, CHEN Da-peng
(Dept. of Computer Science & Technology, Wuhan University of Technology, Wuhan 430070, China)

Abstract: Shale is a new type of modern Web application framework which based on the principle that a Web application framework should be divided into several individual layers. Each layer focus on the specific requirement and the use of one layer do not require the use of other layers. This paper discusses the fundamental principle and main features of Shale framework. Based on analyzing the mechanism of view controller and dialog manager, it detailed discusses how Shale framework implements Web application by a specific example. It clearly explains these key technologies by pieces of codes.

Key words: MVC; Shale; framework; life cycle; view controller; dialog manager

1 MVC 简介

MVC 设计模式[1]的目的在于把交互式的应用程序化分为三个独立的组成部分：模型、视图和控制。三者各司其职，模型用于处理事务逻辑，是应用程序的核心部分；视图用于实现用户界面，将应用程序的数据显示给用户；控制接收用户输入将模型和视图匹配在一起，共同完成用户请求。Web 应用程序建立在无状态的 HTTP 协议之上，所以在 Web 应用程序中对传统 MVC 设计模式做了一些相应的改变，也就是所谓的 Model 2 体系结构。由一个作为控制的 Servlet 接收用户请求，当收到请求后这个 Servlet 与模型相互作用并且决定由哪一个 JSP 页面作为视图组件产生响应。

2 Shale 框架原理及主要特性

2.1 Shale 框架的设计原则

Shale 是一种新型的 Web 应用程序框架，它包含 Java Web 程序设计中一些最重要的、最新的开发成果，包括 JSTL（JSP Standard Tag Library）和 JSF[2]（JavaServer Faces），并且基于 JSF，它关注于提高 JSF 作为基础技术的易用性。Shale 框架的设计原则是，Web 应用程序框架应该被划分成多个独立的层，框架的每个层仅关注与解决对应于本层的请求，对一个层的使用可以不依赖于其他层。同时，Shale 框架也是一些服务的集合，这些服务可以根据具体需要来组合以适应特殊的应用，每个服务与其他服务之间是松散耦合的，而不是象 Struts 那样的一个整体的不易定制和扩展的请求处理器。

2.2 Shale 的 MVC 策略

Shale 建立在 JSF 技术之上，并对其进行了多方面的扩展，因此采用与 JSF 相同的 MVC 策略[3]，其中包含模型组件（Model）、视图组件（View）和控制器组件（Controller）。图 1 为不同组件之间的相互作用关系。

（1）Model 组件包括了 EJB，JDO，JavaBeans 和 JDBC。

（2）View 组件主要包括组件树，作为组件模型的 JavaBeans、转换器、验证器、自定义的渲染器，以及作为资源的 JavaBeans、属性文件和 XML 文件。

（3）Controller 组件主要由作为前端控制 Servlet 的

收稿日期：2006－06－29
作者简介：张南平（1961－），男，湖南长沙人，教授，硕士研究生导师，研究方向为计算机网络技术、ERP、办公自动化。

FacesServlet、XML 配置文件及一系列 action 处理器组成。FacesServlet 从客户端接收请求,执行生命周期中每个步骤,这些步骤包括:恢复视图、应用请求值、处理验证、更新模型值、调用应用程序和最后呈现响应。

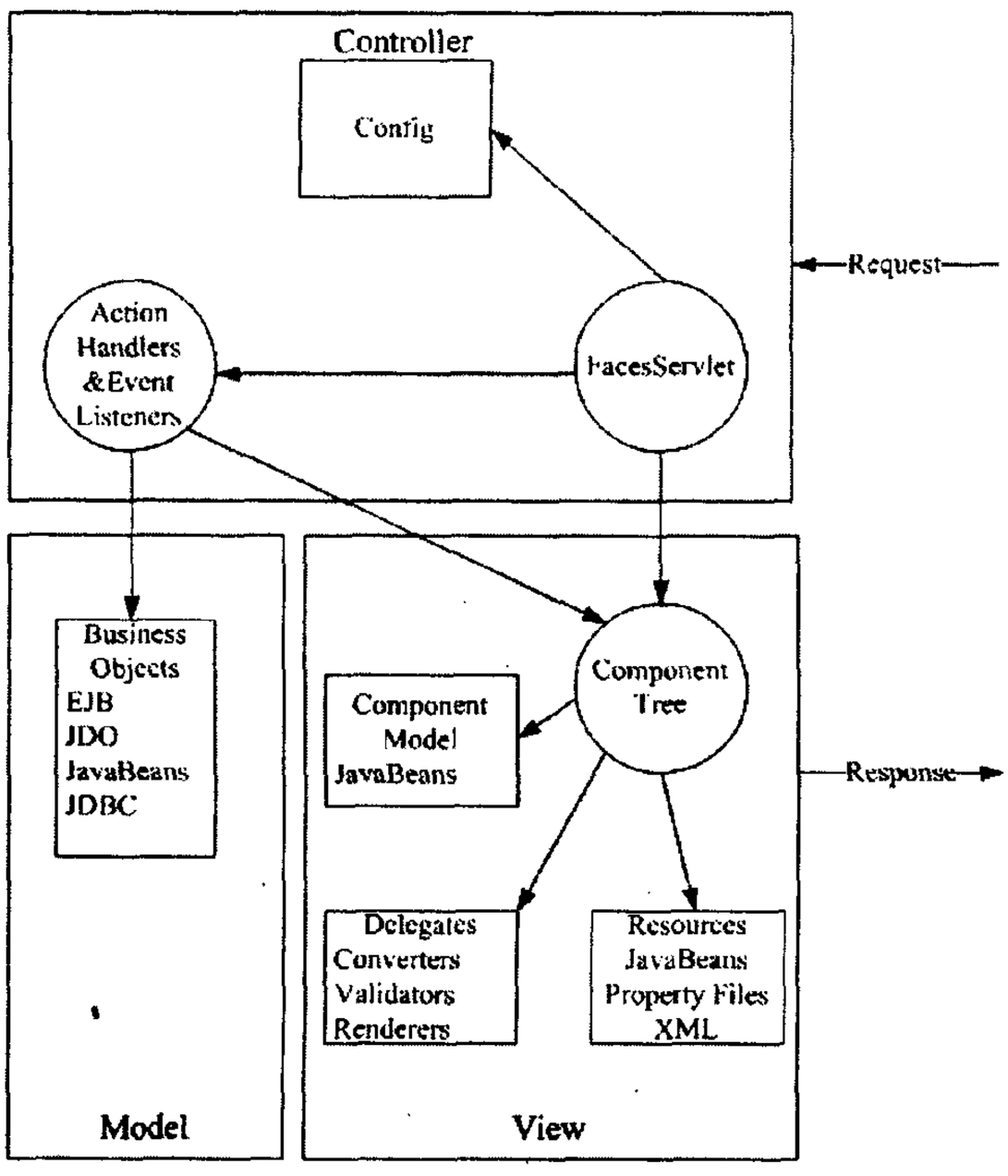

图 1 JSF 的 MVC 策略

在页面第一次显示时,恢复视图阶段会构造组件树然后直接跳到呈现响应阶段显示页面。当页面在浏览器显示后,用户填写表单字段并单击提交按钮,浏览器将表单数据发送给 Web 服务器,此时 FacesServlet 收到一个 POST 请求,进入到应用请求值阶段。在这个阶段,JSF 会遍历组件树中的组件对象,每个组件对象都会检查那些请求值属于它并将其保存。处理验证阶段对提交的数据进行正确性检查,如果发生验证错误则直接调用呈现响应阶段,重新显示页面以便用户更正输入。如果通过验证就进入更新模型值阶段,更新绑定到组件的 bean。在调用应用程序阶段,引起提交表单的按钮或链接组件的 action 方法被执行。此 action 方法返回一个结果字符串,导航处理器根据这个字符串查找下一个页面,最后呈现响应阶段将此页面发送到浏览器。

2.3 Shale 的视图控制器

JSF 的 backing bean[4] 相当于 Struts[5] 框架中 ActionForm 和 Action 类的结合体,它不仅包含应用级事件处理逻辑,还包含呈现响应所用到的数据。Shale 的视图控制器扩展了 backing bean,使其具有对预定义生命周期事件的处理能力。

每一个实现了 ViewController 接口的 backing bean 都会支持一个 boolean 型的属性 postback,如果视图正在处理一个来自同一页面的表单提交,那么与这个视图关联的 postback 属性值被设置为 true,如果是导航到一个新的视图,那么与这个新视图关联的 backing bean 的 postback 属性值被设置为 false。postback 属性值在所有的生命周期方法调用之前被设置,从而决定了随后的生命周期的具体内容,应用逻辑根据这个属性值执行相应的任务。而且每一个实现了 ViewController 接口的 backing bean 都包含如下的方法:

init():在创建与视图关联的 backing bean 时被调用。

preprocess():在恢复视图阶段完成后,应用请求值阶段开始之前被调用。这个方法被调用的前提条件是 postback 属性为真。

prerender():在呈现响应阶段之前被调用。

distroy():在呈现响应阶段之后被调用。

这些方法会在 Shale 生命周期的确定阶段被调用。

视图控制器在请求处理过程中的应用最基本的有如下三种情形:

(1)初次导航到一个新的页面,或者通过 URL 直接访问一个页面时。在这种情况下,恢复视图阶段创建 backing bean 实例,backing bean 调用其 init()方法将 postback 属性设置为 false。因为没有表单提交,所以直接调用 prerender()方法,进入呈现响应阶段向浏览器呈现视图,最后调用 distroy()方法。

(2)页面处理一个表单提交请求,并重新显示此页面。在这种情况下,恢复视图阶段创建 backing bean 实例,backing bean 调用其 init()方法将 postback 属性设置为 true。随后应用请求值、处理验证、更新模型值及调用应用程序阶段被依次执行,然后调用 prerender()方法,进入呈现响应阶段向浏览器呈现视图,最后调用 distroy()方法。

(3)A 页面处理表单提交,然后导航到 B 页面。在这种情况下,首先在恢复视图阶段 A 页面的 backing bean 被实例化,这个 backing bean 调用其 init()方法将 postback 属性设置为 true。之后对应于 A 页面的应用请求值、处理验证、更新模型值及调用应用程序阶段被依次执行,然后导航到 B 页面。B 页面的 backing bean 被创建,在调用 B 页面的 prerender()方法之后,进入呈现响应阶段向浏览器呈现 B 页面,最后同时调用 A、B 页面的 distroy()方法。

2.4 Shale 的对话管理器

在 Web 应用程序中,对话是跨越多重 HTTP 请求的处理过程,Shale 通过对话管理器提供一种机制来实

现对多重 HTTP 请求的处理功能,从而在对标准的导航处理过程之外提供一种新的页面导航机制。在对话管理器中一个对话包括四个状态类型,分别是 Action-State, ViewState, SubdialogState 及 EndState。

(1)ActionState:描述了对一个无参 public 方法的调用,这个方法被执行后返回一个逻辑结果字符串。通过这个字符串驱动对话过渡到后序状态。

(2)ViewState:描述对一个 JSF 视图的呈现。当这个视图中的表单被提交之后,对应的 action 方法返回一个逻辑结果字符串。同样的通过这个字符串驱动对话过渡到后序状态。

(3)SubdialogState:在这个状态中,将父对话的状态推入栈中,并且开始一个特定的子对话的开始状态,当子对话执行完成返回后,父对话从栈中取出继续执行,从子对话返回的逻辑结果字符串驱动父对话过渡到后序状态。

(4)EndState:表示结束当前的对话。此时有两种情况,如果这个 EndState 是属于一个顶级对话,那么必须设置视图标识符,以指明对话结束后要呈现的视图。如果结束的是一个子对话则需要执行出栈操作,并且返回引起这个结束状态的逻辑结果给父对话,由父对话负责呈现当前请求的响应,此时就不需要设置视图标识符。

状态之间的过渡是参考一个 Transition 结构的集合来完成的。每一个 Transition 结构都包含一个字符串和一个目标状态名,当前一状态返回的字符串与某一 Transition 结构中的字符串相匹配时,就过渡到与此字符串对应的目标状态中去。

3　Shale 框架应用

文中通过对用户注册登陆应用程序的实现研究 Shale 框架对话管理器的实现。图 2 显示了注册登陆应用程序的 UML 状态图。

dialog - config. xml 配置文件中指定不同的页面和 Java 类基于其过渡值连接在一起,对应用程序中的对话做出定义。其中包括两个部分,首先作为父对话的 Log on 对话实现了用户登录功能,以下是其代码:

```
<dialog name="Log On" start="Check Cookie">
<action name="Check Cookie" method="#{logon.check}">
<transition outcome="authenticated" target="Exit"/>
<transition outcome="unauthenticated" target="Logon Form"/>
</action>
<view name="Logon Form" viewId="/logon.jsp">
<transition outcome="authenticated" target="Exit"/>
<transition outcome="create" target="Create Profile"/>
<transition outcome="cancel"target="Exit"/>
</view>
<subdialog name="Create Profile" dialogName="Edit Profile">
<transition outcome="success" target="Exit"/>
</subdialog>
<end name="Exit" viewId="/usecases.jsp"/>
</dialog>
```

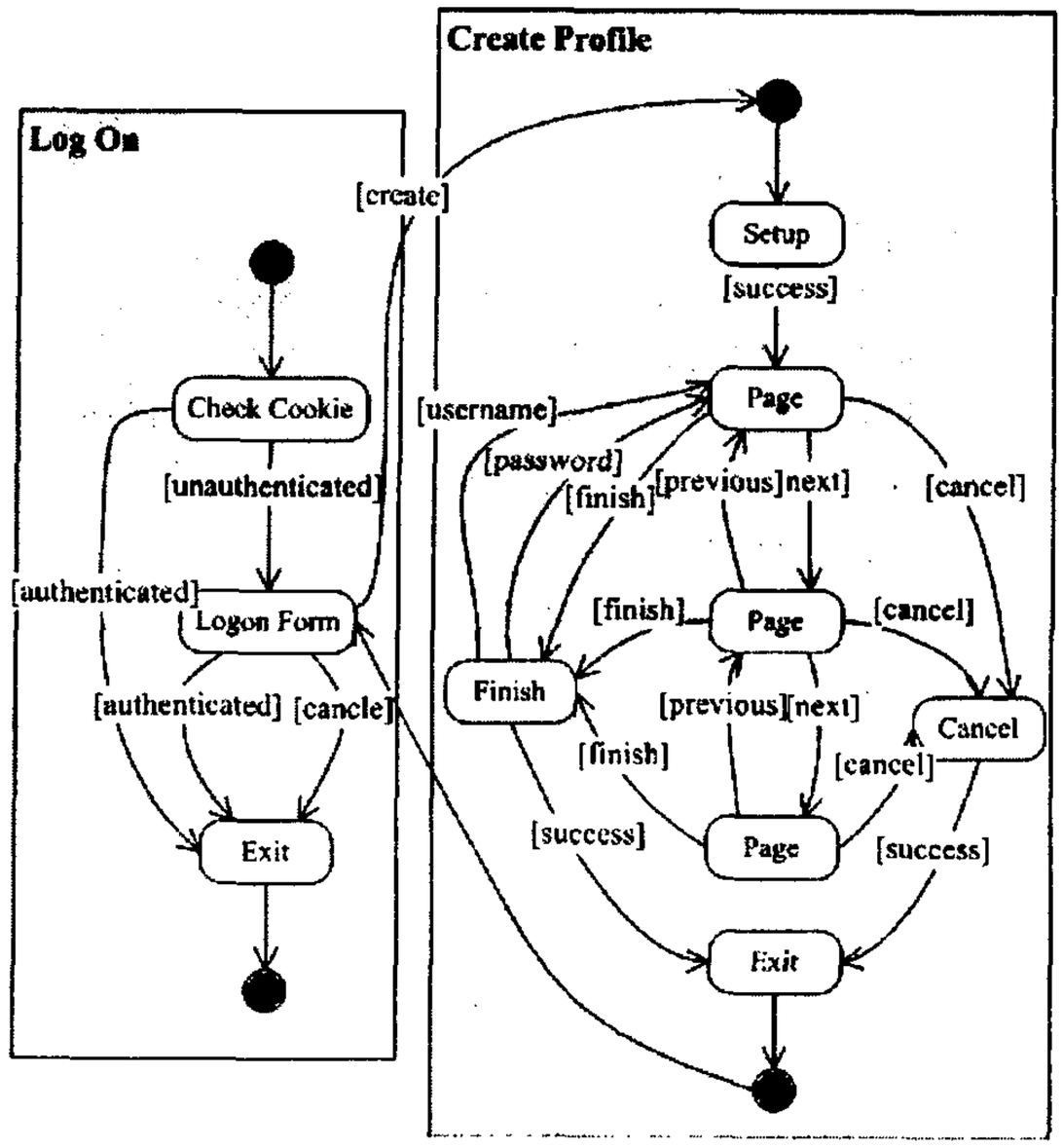

图 2　注册登陆应用程序 UML 状态图

在 Log on 对话的 subdialog 状态中包含一个名为 Edit Profile 的子对话,实现用户注册功能,代码如下:

```
<dialog name="Edit Profile" start="Setup">
<! -- Global transitioh definitions -->
<transition outcome="cancel" target="Cancel"/>
<transition outcome="finish" target="Finish"/>
<action name="Setup" method="#{edit.setup}">
<transition outcome="success" target="Page 1"/>
</action>
<view name="Page 1" viewId="/profile1.jsp">
<transition outcome="next" target="Page 2"/>
</view>
<view name="Page 2" viewId="/profile2.jsp">
<transition outcome="next" target="Page 3"/>
<transition outcome="previous" target="Page 1"/>
</view>
<view name="Page 3" viewId="/profile3.jsp">
<transition outcome="next" target="Exit"/>
<transition outcome="previous" target="Page 2"/>
</view>
<action name="Cancel" method="#{edit.cancel}">
<transition outcome="success" target="Exit"/>
</action>
```

(下转封三)

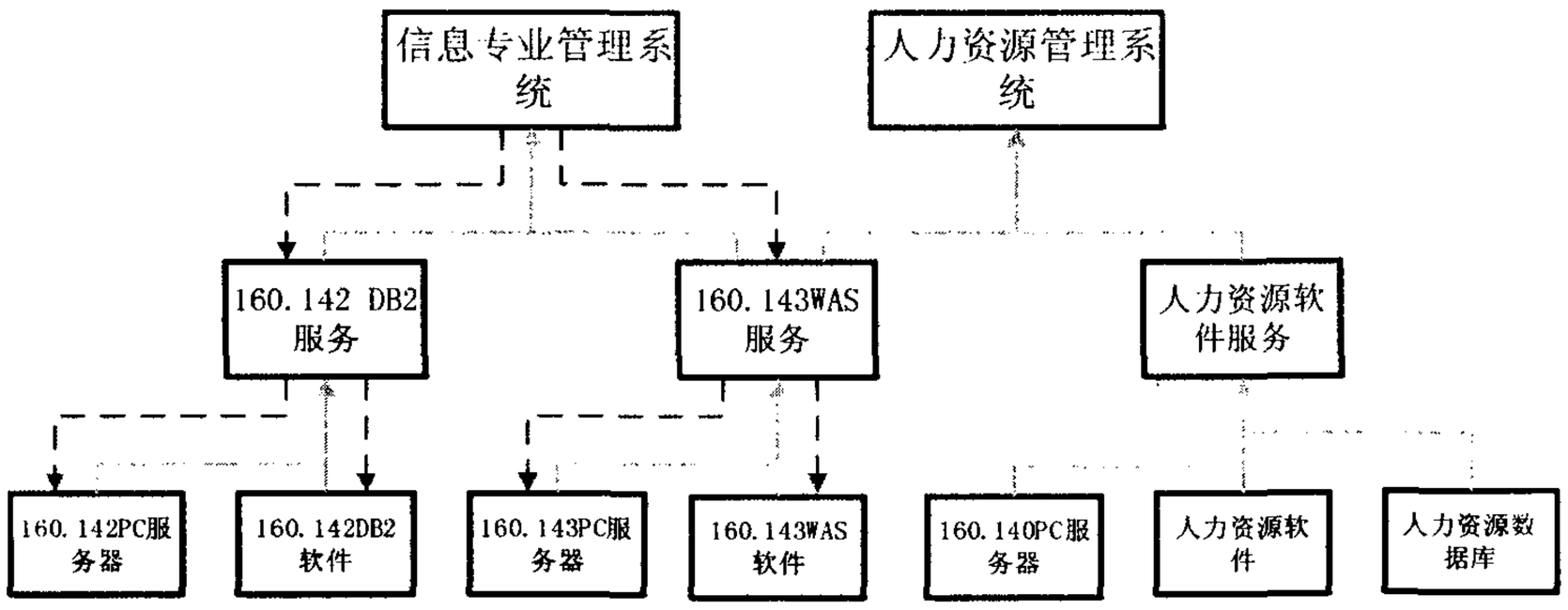

图 5　依赖关系模型

有了支撑和依赖关系模型，IT 管理部门可以对系统之间的影响有很好的前瞻性，防止由于作业前无法预估影响范围导致受影响的范围无法控制。同时可以迅速地定位故障，快速恢复生产，提高 IT 部门的服务质量。

4　总　结

作为 IT 管理的“ERP 解决方案”，ITIL 给实施它的企业带来了丰厚的商业价值。大量的成功实践表明实施 ITIL 可以将企业 IT 部门的运营效率提高 25%～30%。加特纳(Gartner)和国际数据集团(IDC)等研究机构的调查研究也表明，通过在 IT 部门实施 ITIL 的最佳服务管理实践，可以将因重复呼叫、不当的变更等引起的延误时间减少 79%，每年可以为每个终端用户平均节约 800 美元的成本，同时将每项新服务推出的时间缩短一半。目前我国实施 IT 服务管理的企业也取得了良好的经济效益[1]。ITIM 模型的实践只是 ITIL 框架中的一小部分，希望通过 ITIM 模型的实践，能够在提高 IT 部门管理效率和服务质量的同时也能对 ITIL 理论有更进一步的理解，为构建本土化的最佳实践积累经验。

参考文献：

[1] 左天祖，刘　伟. 中国 IT 服务管理指南[M]. 北京：北京大学出版社，2004.

[2] 四　木. ITIL 及其实施步骤. [EB/OL]. 2005－06. http://www2.ccw.com.cn/04/0424/b/0424b04-1.asp.

[3] 孙　强，刘小宁. 实施 IT 服务管理——路在脚下[EB/OL]. 2004－08. http://www.ccidcom.com/weekly/news/39/200483114920.htm.

[4] 佩里切利 G. 服务营销学[M]. 张　密译. 北京：对外经济贸易大学出版社，2000.

[5] 蔺　雷，吴贵生. 服务创新[M]. 北京：清华大学出版社，2003.

(上接第 249 页)

```
<action name="Finish" method="#{edit.finish}">
<transition outcome="password" target="Page 1"/>
<transition outcome="success" target="Exit"/>
<transition outcome="username" target="Page 1"/>
</action>
<end name="Exit"/>
</dialog>
```

对话的 action 状态代表了一个无参方法的调用，以下是 Log on 对话中 logon.check 方法的部分代码：

```
public String check()
{
  ……
  if (user==null) {
  return "unauthenticated";
  }
  //Register the newly authenticated user and return that outcome
  register(user);
  return "authenticated";
}
```

和 action 状态通过方法调用返回一个字符串相似，View 状态对应的 JSP 页面执行相应操作后，以静态或动态方式返回一个字符串。transition 将 action 和 View 状态返回值映射到下一个 action 或 View 状态。

4　结束语

通过对 Shale 框架的分析，探讨了 Shale 的设计思想和运行机制，并通过实例详细描述了基于 Shale 的 Web 应用实现。Shale 基于分层的思想，把整个应用划分成多个独立的服务，服务间松散耦合便于开发者根据需要有选择地组合这些服务，形成满足特定需要的 Web 应用程序，更加符合现代软件工程的设计思想。Shale 是目前最新的 Web 应用开发框架，其一出现就已引起普遍关注，相信在不远的将来一定会为业界广泛应用。

参考文献：

[1] Berry C A, Carnell J. 实用 J2EE 设计模式编程指南[M]. 北京：电子工业出版社，2003.

[2] Kurniawan. Java Server Faces 编程[M]. 北京：清华大学出版社，2005.

[3] Dudney B, Lehr J. Mastering JavaServer Faces(中文版)[M]. 北京：电子工业出版社，2005.

[4] Geary D, Horstmann C. JaveServer Faces 核心编程[M]. 北京：电子工业出版社，2005.

[5] Husted T. 实战 STRUTS[M]. 北京：机械工业出版社，2005.