第5章 软件实现

在完成系统分析和设计之后，进入软件实现环节。软件实现的目标是：利用已有的资产和构件，遵循程序开发规范，按照系统详细设计说明书中的数据结构、算法和模块实现等方面的设计，用面向对象技术实现目标系统的功能、性能、接口、界面等要求。

在 eGov 电子政务系统中，基于 Struts-Spring-Hibernate 架构完成了软件实现步骤。

5.1 Struts-Spring-Hibernate 架构概述

目前，国内外信息化建设已经进入以 Web 应用为核心的阶段。Java 语言是开发 Web 应用的最佳语言之一。然而，就算用 Java 建造一个不是很复杂的 Web 应用系统，也不是一件轻松的事情。有很多东西需要仔细考虑，例如，要考虑怎样建立用户接口，在哪里处理业务逻辑，怎样持久化数据。幸运的是，Web 应用面临的一些问题已经由曾遇到过这类问题的开发者通过建立相应的框架解决了。事实上，企业开发中直接采用的往往并不是某些具体的技术，例如大家熟悉的 Core Java、JDBC、Servlet、JSP 等，而是基于这些技术的应用框架，Struts、Spring 和 Hibernate 就是其中最常用的几种。

这里讨论一个使用 3 种开源框架的策略：表示层用 Struts，业务层用 Spring，持久层用 Hibernate。

接下来介绍这些技术。

5.2 Struts 技术

在 eGov 电子政务系统中，使用 Struts 框架来实现用户接口（User Interface，UI）层及其与后端应用层之间的交互。

5.2.1 Struts 概述

Struts 是由 Craig McClanahan 在 2001 年发布的 Web 框架。经过多年的改进，Struts 越来越稳定和成熟。Struts 1.x 版本被统称为 Struts 1。2006 年，Struts 推出全新框架，命名为 Struts 2，它改进了 Struts 1 的一些主要不足。

Struts 1 的主要缺点如下：

(1) 支持的表示层技术单一。Struts 1 只支持 JSP 视图技术，当然，可以部分支持 Velocity 等技术。

(2) Struts 1 与 Servlet API 严重耦合，难于测试。

例如，Struts 1 的 Action 的 execute 方法有 4 个参数：ActionMapping、ActionForm、HttpServletRequest 和 HttpServletResponse，初始化这 4 个参数比较困难，尤其是 HttpServletRequest 和 HttpServletResponse 两个参数，因为这两个参数通常是由容器进行注入的。如果脱离 Web 服务器，Action 的测试是很困难的。

(3) Struts 1 的侵入性太大。一个 Action 中包含了大量的 Struts API，例如 ActionMapping、ActionForm、ActionForward 等。这种侵入式设计最大的弱点在于切换框架相当困难，代码复用率较低，不利于重构。

Struts 2 在另一个 MVC 框架——WebWork 的优良基础设计之上进行了一次巨大的升级。注意，Struts 2 不是基于 Struts 1，而是基于 WebWork 的。Struts 2 针对 Struts 1 的不足，提出了全新的解决方案。

5.2.2 MVC 与 Struts 映射

Struts 的体系结构实现了 MVC 设计模式的概念，它将这些概念映射到 Web 应用的组件和概念中。

1. 控制器层

与 Struts 1 使用 ActionServlet 作为控制器不同，Struts 2 使用了 Filter 技术，FilterDispatcher 是 Struts 2 框架的核心控制器，该控制器负责拦截和过滤所有的用户请求。如果用户请求以 Action 结尾，该请求将被转入 Struts 框架来进行处理。Struts 2 框架获得了 *.action 请求后，将根据该请求前面的名称部分决定调用哪个业务控制 Action 类。例如，对于 test.action 请求，Struts 2 框架调用名为 test 的 Action 来处理该请求。

Struts 2 应用中的 Action 都被定义在 struts.xml 文件中。在该文件中配置 Action 时，主要定义了该 Action 的 name 属性和 class 属性，其中，name 属性决定了该 Action 处理哪个用户请求，而 class 属性决定了该 Action 的实现类。例如，<action name="registAction" class="com.ascent.action.RegistAction">。

用于处理用户请求的 Action 实例并没有与 Servlet API 耦合，所以无法直接处理用户请求。为此，Struts 2 框架提供了系列拦截器，该系列拦截器负责将 HttpServletRequest 请求中的请求参数解析出来，传入 Action，并回调 Action 的 execute 方法来处理用户请求。

2. 显示层

Struts 2 框架改进了 Struts 1 只能使用 JSP 作为视图技术的缺点(当然，Struts 1 可以

部分支持 Velocity 等技术)。Struts 2 框架允许使用其他视图技术(如 FreeMarker 等)作为显示层。

当 Struts 2 的控制器调用业务逻辑组件处理完用户请求后,会返回一个字符串,该字符串代表逻辑视图,它并未与任何视图技术关联。

当在 struts.xml 文件中配置 Action 时,还要为 Action 元素指定系列 result 子元素,每个 result 子元素定义上述逻辑视图和物理视图之间的映射。一般情况下,使用 JSP 技术作为视图,故配置 result 子元素时没有 type 属性,默认使用 JSP 作为视图资源。例如,<result name="error">/product/register.jsp</result>。

如果需要在 Struts 2 中使用其他视图技术,则可以在配置 result 子元素时指定相应的 type 属性。例如,为 type 属性指定的值可以是 velocity。

3. 模型层

模型层指的是后端业务逻辑处理,Action 调用它来处理用户请求。当控制器需要获得业务逻辑组件实例时,通常并不会直接获得,而是通过工厂模式来获得,或者利用其他 IoC 容器(如 Spring 容器)来管理业务逻辑组件的实例。在后面会详细展开这些技术。

基于 MVC 的系统中的业务逻辑组件还可以细分为两个概念:系统的内部状态以及能够改变状态的行为。可以把内部状态当作名词(事物),把行为当作动词(对事物状态的改变),它们使用 JavaBean、EJB 或 Web Service 实现。

5.2.3 Struts 2 的工作流程和配置文件

1. Struts 2 的工作流程

Struts 2 的工作流程是 WebWork 的升级,而不是 Struts 1 的升级。Struts 2 的体系如图 5-1 所示。

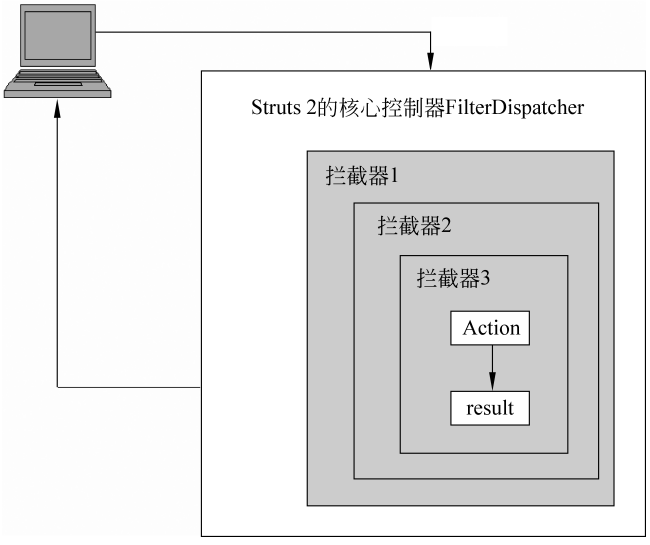

图 5-1 Struts 2 的体系

Struts 2 的工作流程如下:

(1) 浏览器发送请求,例如请求/regist.action、/reports/myreport.pdf 等。

(2) 核心控制器 FilterDispatcher 根据请求调用合适的 Action。

(3) WebWork 的拦截器链自动对请求应用通用功能,例如验证、工作流或文件上传等功能。

(4) 回调 Action 的 execute 方法,该方法先获取用户请求参数,然后执行某种业务操作,既可以将数据保存到数据库,也可以从数据库中检索信息。实际上,因为 Action 只是一个控制器,它会调用业务逻辑组件(即模型层)来处理用户的请求。

(5) Action 的 execute 方法的处理结果信息将被输出到浏览器中,可以是 HTML 页面或者图像,也可以是 PDF 文档或者其他文档。Struts 2 支持的视图技术非常多,既支持 JSP,也支持 Velocity、FreeMarker 等模板技术。

要想更深入地掌握 Struts 的核心技术和流程,就要先理解 Struts 的配置文件。

2. Struts 2 的配置文件

Struts 2 的配置文件有两个,包括配置 Action 的 struts.xml 文件和配置 Struts 2 全局属性的 struts.properties 文件。接下来分别对它们进行讨论。

1) struts.xml 文件

Struts 框架的核心配置文件就是 struts.xml。在默认情况下,Struts 2 框架将自动加载放在 WEB-INF/classes 路径下的 struts.xml 文件。该文件主要负责管理应用中的 Action 映射、该 Action 包含的 result 定义等以及其他相关配置。

以下是 eGov 电子政务系统中的 struts.xml 内容:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC "-//Apache Software Foundation//
    DTD Struts Configuration 2.0//EN"
"http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>

        <constantname="struts.i18n.encoding" value="GBK"/>
        <package name="struts2" extends="json-default">
            <action name="*authAction" class="com.ascent.action.AuthAction"
                method="{1}">
                <result>/jsp/authplot.jsp</result>
                <result name="qxx">/jsp/manager.jsp</result>
                <result name="de" type="json">/jsp/authplot.jsp</result>
                <result name="am">/jsp/authmanager.jsp</result>
            </action>
            <action name="exitAction" class="com.ascent.action.ExitAction">
                <result>/index.jsp</result>
            </action>
            <action name="loginAction" class="com.ascent.action.LoginAction">
                <result>/index.jsp</result>
            </action>
```

```
            <action name="*newsAction" class="com.ascent.action.NewsAction"
                method="{1}">
            <result>/jsp/manager.jsp</result>
            <result name="auditing">/jsp/auditing.jsp </result>
            <result name="auditings">/jsp/auditings.jsp </result>
            <result name="myissue">/jsp/myissue.jsp</result>
            <result name="view">/jsp/view.jsp</result>
            <result name="list">/jsp/list.jsp</result>
            <result name="newsselect">/jsp/updatenews.jsp</result>
        </action>
    </package>
</struts>
```

2）struts.properties 文件

除了 struts.xml 文件外，Struts 2 框架还包含 struts.properties 文件，该文件通常放在 Web 应用的 WEB-INF/classes 路径下。它定义了 Struts 2 框架的大量属性，开发者可以通过改变这些属性来满足个性化应用的需求。

在有些时候，开发者不喜欢使用额外的 struts.properties 文件。前面提到，Struts 2 允许在 struts.xml 文件中管理 Struts 2 属性，在该文件中通过配置 constant 元素，一样可以配置这些属性。建议尽量在 struts.xml 文件中配置 Struts 2 常量。

5.2.4 创建 Controller 组件

Struts 的核心是 Controller 组件。它是连接 Model 组件和 View 组件的桥梁，也是理解 Struts 2 架构的关键。Struts 2 的控制器由两个部分组成：FilterDispatcher 和 Action。

1. FilterDispatcher

任何 MVC 框架都需要与 Web 应用整合，这就离不开 web.xml 文件，只有配置在 web.xml 文件中 Filter/Servlet 才会被 Web 应用加载。对于 Struts 2 框架而言，需要加载 FilterDispatcher。因为 Struts 2 将核心控制器设计成 Filter，而不是一个 Servlet。因此，为了让 Web 应用加载 FilterDispatcher，需要在 web.xml 文件中配置 FilterDispatcher。

配置 FilterDispatcher 的代码片段如下：

```
<!--配置 Struts 2 框架的核心 Filter-->
<filter>
        <!--配置 Struts 2 核心 Filter 的名字-->
        <filter-name>struts</filter-name>
        <!--配置 Struts 2 核心 Filter 的实现类-->
        <filter-class>org.apache.struts2.dispatcher.Filter Dispatcher
          </filter-class>
        <init-param>
            <!--配置 Struts 2 框架默认加载的 Action 包结构-->
            <param-name>actionpackages</param-name>
            <param-value>org.apache.struts2.showcase.person</param-value>
```

```
        </init-param>
        <!--配置 Struts 2 框架的配置提供者类-->
        <init-param>
            <param-name>configProviders</param-name>
            <param-value>lee.MyConfigurationProvider</param-value>
        </init-param>
    </filter>
```

正如上面所示，当配置 Struts 2 的 FilterDispatcher 类时，可以指定一系列的初始化参数。为该 Filter 配置初始化参数时，其中的 3 个初始化参数有特殊意义：

(1) Config。该参数的值是一系列以英文逗号(,)隔开的字符串，每个字符串都有一个 XML 配置文件的位置。Struts 2 框架将自动加载该属性指定的配置文件。

(2) ActionPackages。该参数的值也是一系列以英文逗号隔开的字符串，每个字符串都是一个包空间，Struts 2 框架将扫描这些包空间下的 Action 类。

(3) ConfigurationProviders。如果用户需要实现自己的 ConfigurationProvider 类，则可以提供一个或多个实现了 ConfigurationProvider 接口的类，然后将这些类的类名设置成该属性的值，类名之间以英文逗号隔开。

除此之外，还可在此处配置 Struts 2 常量，每个＜init-param＞元素配置一个 Struts 2 常量，其中＜param-name＞子元素指定了常量 name，而＜param-value＞子元素指定了常量 value。

在 web.xml 文件中配置该 Filter，还需要配置该 Filter 拦截的 URL。通常，让该 Filter 拦截所有的用户请求，因此使用通配符来配置该 Filter 拦截的 URL。

下面是配置该 Filter 拦截 URL 的代码片段：

```
<!--配置 Filter 拦截的 URL-->
<filter-mapping>
    <!--配置 Struts 2 的核心 FilterDispatcher 以拦截所有用户请求-->
    <filter-name>struts</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

配置了 Struts 2 的核心 FilterDispatcher 后，就基本完成了 Struts 2 在 web.xml 文件中的配置了。

2. Action 的开发

对于 Struts 2 应用而言，Action 是应用系统的核心，也称 Action 为业务控制器。开发者需要提供大量的 Action 类，并在 struts.xml 文件中配置这个 Action 类。

1) 实现 Action 类

相对于 Struts 1 而言，Struts 2 采用了低侵入式的设计。Struts 2 的 Action 类是普通的 POJO(通常应该包含一个无参数的 execute 方法)，从而具有很好的代码复用性。

例如，用户登录模块 LoginAction 类的代码如下：

```
package com.ascent.action;
import com.ascent.po.Usr;
```

```
import com.opensymphony.xwork2.ActionContext;
public class LoginAction extends BaseAction {
    public String username;
    public String password;
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password=password;
    }
    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username=username;
    }
    public String execute(){
        if(this.getUsername()!=null && this.getPassword()!=null){
            Usr user=userService.userLogin(this.getUsername(),
                this.getPassword());
            if(user!=null){
                ActionContext.getContext().getSession().put("user",user);
            } else {
                ActionContext.getContext().put("tip","账号或密码不正确");
                return SUCCESS;
            }
        } else {
            ActionContext.getContext().put("tip","账号或密码不正确");
            return SUCCESS;
        }
        return SUCCESS;
    }
}
```

注意：上面的 LoginAction 类继承了 BaseAction 类，它是本书作者开发的一个帮助类，是为了对 Spring 集成提供支持的工具类。它可以帮助读者更好地理解 Spring，有兴趣的读者可参考配套电子资源中的源代码。

上面的 Action 类只是一个普通类，这个 Java 类提供了两个属性：username 和 password，这两个属性分别对应两个 HTTP 请求参数。

为了让用户开发的 Action 类更规范，Struts 2 提供了一个 Action 接口。该接口定义了 Struts 2 的 Action 处理类应该实现的规范。它只定义了一个 execute 方法，该接口的规范规定了 Action 类应该包含能够返回一个字符串的方法。除此之外，该接口还定义了 5 个字符串常量，分别是 ERROR、INPUT、LOGIN、NONE 和 SUCCESS，它们的作用是统一 execute 方法的返回值。例如，当 Action 类处理用户请求成功后，有人喜欢返回

WELCOME 字符串，有人喜欢返回 SUCCESS 字符串，这样不利于项目的统一管理。Struts 2 的 Action 类定义了上面的 5 个字符串，分别代表了统一的特定含义。

另外，Struts 2 还提供了 Action 类的一个实现类——ActionSupport。它是一个默认的 Action 类，该类里已经提供了许多默认方法，这些默认方法包括获取国际化信息的方法、数据校验的方法、默认的处理用户请求的方法等。实际上，ActionSupport 类是 Struts 2 默认的 Action 处理类，如果让开发者的 Action 类继承该 Action 类，则会大大简化 Action 的开发。

2）Action 访问 Servlet API

Struts 2 的 Action 并未直接与任何 Servlet API 耦合，这是 Struts 2 相对于 Struts 1 的一个改进之处，因为这样的 Action 类具有更好的重用性，并且能更轻松地测试 Action。

然而，对于 Web 应用的控制器而言，不访问 Servlet API 几乎是不可能的，例如获得 HTTP Request 参数、跟踪 HTTP Session 状态等。为此，Struts 2 提供了 ActionContext 类，Struts 2 的 Action 可以通过该类来访问 Servlet API，包括 HttpServletRequest、HttpSession 和 ServletContext 这 3 个类，它们分别代表 JSP 内置对象中的 request、session 和 application。

表 5-1 是 ActionContext 类中包含的常用方法。

表 5-1 ActionContext 类中包含的常用方法

方　法	描　述
Object get(Object key)	该方法类似于调用 HttpServletRequest 的 getAttribute(stringname)方法
Map getApplication	返回一个 Map 对象，该对象模拟了该应用的 ServletContext 实例
Static ActionContext getContext	静态方法，获取系统的 ActionContext 实例
Map getParameters	获取所有的请求参数。类似于调用 HttpServletRequest 对象的 getparameter Map 方法
Map getSession	返回一个 Map 对象，该对象模拟了 HttpSession 实例
Void setApplication(Map application)	直接传入一个 Map 实例，将该 Map 实例里的键-值(key-value)对转换成 session 的属性名和属性值
Void setSession(Map session)	直接传入一个 Map 实例，将该 Map 实例里的键-值对转换成 session 的属性名和属性值

虽然 Struts 2 提供了 ActionContext 来访问 Servlet API，但这种访问毕竟不能直接获取 Servlet API 实例。为了在 Action 中直接访问 Servlet API，还提供了一些接口，如表 5-2 所示。

另外，为了直接访问 Servlet API，Struts 2 提供了 ServletActionContext 类。借助于这个类的帮助，开发者也能够在 Action 中直接访问 Servlet API，同时无须在 Action 类中实现上面的接口。这个类包含的静态方法如表 5-3 所示。

表 5-2　直接访问 Servlet API 的接口

接　口	描　述
ServletContextAware	实现该接口的 Action 可以直接访问应用的 ServletContext 实例
ServletRequestAware	实现该接口的 Action 可以直接访问用户请求的 HttpServletRequest 实例
ServletResponseAware	实现该接口的 Action 可以直接访问服务器响应的 HttpServletResponse 实例

表 5-3　ServletActionContext 类包含的静态方法

方　法	描　述
Static PageContext getPageContext()	取得 Web 应用的 PageContext 对象
Static HttpServletRequest getRequest()	取得 Web 应用的 HttpServletRequest 对象
Static HttpServletResponse getResponse()	取得 Web 应用的 HttpServletResponse 对象
Static ServletContext getServletContext()	取得 Web 应用的 ServletContext 对象

在 eGov 电子政务系统中，AuthAction 就使用了 Servlet API，其代码如下：

```
package com.ascent.action;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import com.ascent.po.Authorization;
import com.ascent.po.Department;
import com.ascent.po.News;
import com.ascent.po.Userauth;
import com.ascent.po.Usr;
import com.opensymphony.xwork2.ActionContext;
public class AuthAction extends BaseAction {
    public String typ;
    public String pars;
    public String id;
    public String name;
    public String gid;
    public String sid;
    public String gname;
    public String sname;
    public String dept;
    public String getGname() {
        return gname;
    }
    public void setGname(String gname) {
        this.gname=gname;
    }
    public String getSname() {
```

```
    return sname;
}
public void setSname(String sname) {
    this.sname=sname;
}
public String getGid() {
    return gid;
}
public void setGid(String gid) {
    this.gid=gid;
}
public String getSid() {
    return sid;
}
public void setSid(String sid) {
    this.sid=sid;
}
public String getId() {
    return id;
}
public void setId(String id) {
    this.id=id;
}
public String getName() {
    return name;
}
public void setName(String name) {
    this.name=name;
}
public String getPars() {
    return pars;
}
public void setPars(String pars) {
    this.pars=pars;
}
public String getTyp() {
    return typ;
}
public void setTyp(String typ) {
    this.typ=typ;
}
public String getDept() {
    return dept;
}
public void setDept(String dept) {
```