spring mvc入门

作者: 86asm http://86asm.iteye.com

spring mvc
后端控制器、映射处理器、视图解析器、注解配置

目 录

1. spring mvc

1.1 spring-mvc入门（一）：入门实例

发表时间: 2011-02-27 关键字: MVC, Spring, Servlet, Web, Bean

引言

1.MVC ：Model-View-Control

框架性质的C 层要完成的主要工作：封装web 请求为一个数据对象、调用业务逻辑层来处理数据对象、返回处理数据结果及相应的视图给用户。

2. 简要概述springmvc

Spring C 层框架的核心是 DispatcherServlet，它的作用是将请求分发给不同的后端处理器，也即 使用了一种被称为Front Controller 的模式（后面对此模式有简要说明）。 Spring 的C 层框架使用了后端控制器来、映射处理器和视图解析器来共同完成C 层框架的主要工作。并且spring 的C 层框架还真正地把业务层处理的数据结果和相应的视图拼成一个对象，即我们后面会经常用到的ModelAndView 对象。

一、入门实例

1. 搭建环境

在spring 的官方API 文档中，给出所有包的作用概述，现列举常用的包及相关作用：

org.springframework.aop-3.0.5.RELEASE.jar ：与Aop 编程相关的包

org.springframework.beans-3.0.5.RELEASE.jar ：提供了简捷操作bean 的接口

org.springframework.context-3.0.5.RELEASE.jar ：构建在beans 包基础上，用来处理资源文件及国际化。

org.springframework.core-3.0.5.RELEASE.jar ：spring 核心包

org.springframework.web-3.0.5.RELEASE.jar ：web 核心包，提供了web 层接口

org.springframework.web.servlet-3.0.5.RELEASE.jar ：`web` 层的一个具体实现包，`DispatcherServlet`也位于此包中。

后文全部在`spring3.0` 版本中进行，为了方便，建议在搭建环境中导入`spring3.0` 的所有`jar` 包（所有`jar` 包位于`dist` 目录下）。

2. 编写HelloWorld 实例

步骤一、建立名为springMVC_01_helloword ，并导入上面列出的jar 包。

步骤二、编写web.xml 配置文件，代码如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
        http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">

        <servlet>
                <servlet-name>spmvc</servlet-name>
                <servlet-class>
                        org.springframework.web.servlet.DispatcherServlet
                </servlet-class>
                <load-on-startup>1</load-on-startup>
        </servlet>
        <servlet-mapping>
                <servlet-name>spmvc</servlet-name>
                <url-pattern>*.do</url-pattern>
        </servlet-mapping>
</web-app>
```

简要说明 ：DispatcherServlet 就是一个Servlet ，也是对请求进行转发的核心Servlet 。在这里即所有.do 的请求将首先被DispatcherServlet 处理，而DispatcherServlet 它要作的工作就是对请求进行分发（也即是说把请求转发给具体的Controller ）。可以简单地认为，它就是一个总控处理器，但事实上它除了具备总控处理理器对请求进行分发的能力外，还与spring 的IOC 容器完全集成在一起，从而可以更好地使用spring 的其它功能。在这里还需**留意** < servlet-name > spmvc </ servlet-name > ，下面步骤三会用到。

步骤三、建立 spmvc-servlet.xml 文件，它的命名规则： servlet-name-servlet.xml 。它的主要代码如下：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN" "http://www.springframework.org/dtd/spring-bea
<beans>
```

```
        <bean id="simpleUrlHandlerMapping"           class="org.springframework.web.servlet.har
                <property name="mappings">
                        <props>
                                <prop key="/hello.do">helloControl</prop>
                        </props>
                </property>
        </bean>
        <bean id="helloControl" class="com.asm.HelloWord"></bean>
</beans>
```

说明： hello.do 的请求将给名为 helloControl 的 bean 进行处理。

步骤四、完成 HelloWord.java 的编写，代码如下：

```
package com.asm;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.Controller;
public class HelloWord implements Controller {
        public ModelAndView handleRequest(HttpServletRequest request, HttpServletResponse respo
                        throws Exception {
                ModelAndView mav = new ModelAndView("hello.jsp");
                mav.addObject("message", "Hello World!");
                return mav;
        }
}
```

说明 ：ModelAndView 对象是包含视图和业务数据的混合对象，即是说通过此对象，我们可以知道所返回的相应页面（比如这里返回hello.jsp 页面），也可以在相应的页面中获取此对象所包含的业务数据（比如这里message-hello worrld ）。

步骤五、在当前项目web 根目录下编写hello.jsp ，主要代码如下：

< body >

世界，你好！

获取值： ${message }

</ body >

步骤六：输入 .../hello.do 进行测试。

4. 简析spring mvc 工作原理

（1）启动服务器，根据web.xml 的配置加载前端控制器（也称总控制器） DispatcherServlet 。在加载时、会完成一系列的初始化动作。

（2）根据servlet 的映射请求（上面的helloWorld 实例中针对.do 请求），并参照“控制器配置文件”（即spmvc-servlet.xml 这样的配置）文件，把具体的请求分发给特定的后端控制器进行处理（比如上例会分发给HelloWorld 控制器进行处理）

（3）后端控制器调用相应的逻辑层代码，完成处理并返回视图对象（ ModelAndView ）给前端处理器。

（4）前端控制器根据后端控制器返回的 ModelAndView 对象，并结合一些配置（后面有说明），返回一个相应的页面给客户端。

小结 ：这种Front Controller 模式常应用在主流的web 框架中，比如典型的struts1.x 框架.Front Controller 模式：所有请求先交给一个前端处理器（总控处理器）处理，然后前端处理器会参照一些配置文件再把具体的请求交给相应的后端处理器。后端处理器调用逻辑层代码，并根据逻辑返回相应的视图对象给前端控制器。然后前端控制器再根据视图对象返回具体的页面给客户端（提示：和spring mvc 一样，在struts1.x 中前端控制器是Servlet, 而在struts2 中前端控制器是Filter ）。 **概述** Front Controller 模式：前端控制器预处理并分发请求给后端控制器，后端控制器进行真正的逻辑处理并返回视图对象，前端控器器根据视图对象返回具体页面给客户端。

5. 初识spring mvc 的视图

在前面的HelloWorld 实例中，在HelloWorld.java 中返回 ModelAndView mav = **new** ModelAndView("hello.jsp") 参数为 hello.jsp ，它会对应于当前项目根目录下的 hello.jsp 页面。但 spring mvc 为我们提供了一个特别的视图定位方式，下面改进前面的 HelloWord 实例：

改进一 ：在 spmvc-servlet.xml 中增加如下代码：

```
<bean id="viewResolver"         class="org.springframework.web.servlet.view.InternalResourceVie
                <property name="prefix" value="/WEB-INF/page/" />
```

```
                <property name="suffix" value=".jsp" />
</bean>
```

改进二 ：在HelloWorld.java 重新定义返回的 ModelAndView 对象，即把 ModelAndView mav = **new** ModelAndView("hello.jsp") 改为 ModelAndView mav = **new** ModelAndView("hello")

改进三 ：在/WEB-INF/page 目录下建立hello.jsp 页面

进行上面三个改进操作后，重新访问hello.do 会访问到WEB-INF/page/hello.jsp 页面。
简析视图定位 ：当返回 ModelAndView 对象名称为hello 时，会给hello 加上前后缀变成

/WEB-INF/page/hello.jsp 。因此在给前后缀赋值时，应特别注意它和返回的 ModelAndView 对象能否组成一个正确的文件全路径。在前面的“简析spring mvc 工作原理(4) ” 点中提到在根据ModelAndView 对象返回页面时，会结合一些配置。这里就是结合了视图定位方式，给viewName 加上前后缀进行定位。

1.2 spring-mvc入门（二）：后端控制器（上）

发表时间: 2011-03-07 关键字: MVC, Spring, 应用服务器, Bean, Web

1.概述SpringMVC后端控制器

为了方便开发人员快捷地建立适合特定应用的后端控制器，springMVC实现Controller接口，自定义了许多特定控制器。这些控制器的层次关系如下：

-AbstractController
-AbstractUrlViewController
-UrlFilenameViewController
-BaseCommandController
-AbstractCommandController
-AbstractFormController
-AbstractWizardFormController
-SimpleFormController
-CancellableFormController
-MultiActionController
-ParameterizableViewController
-ServletForwardingController
-ServletWrappingController

下面重点分析两个特色控制器：

2.SimpleFormController控制器

在正式开发前，请先熟悉上前面的HelloWord实例。在保证熟悉前一个实例后，我们建立名为springMVC_02_controllerweb项目，并导入相关的jar包。

步骤一：建立后端控制器RegControl.java代码如下：

```
package com.asm;
//...省略导入的相关类
public class RegControl extends SimpleFormController{
        @SuppressWarnings("deprecation")
        public RegControl() {
                setCommandClass(User.class);
        }

        protected ModelAndView processFormSubmission(HttpServletRequest arg0, HttpServletRespor
                        Object formbean, BindException arg3) throws Exception {
                User user = (User) formbean;
```

```
                ModelAndView mav = new ModelAndView("hello");
                mav.addObject("message", "Hello World!");
                mav.addObject("user", user);
                return mav;
        }

        protected ModelAndView showForm(HttpServletRequest arg0, HttpServletResponse arg1, Binc
                        throws Exception {
                return null;
        }
}
```

User.java，代码如下：

```
package com.asm;
public class User {
        private String username;
        private int age;
        //省略getter/setter方法
}
```

简要说明：如果熟悉struts1.x相信很容易理解Object formbean参数，其实它就是和表单属性打交道的一个对象，也即是说表单参数会依据一定的规则填充给formbean对象。在struts1.x中，如果像把这种与formbean转换成User对象，必须要求User继承自ActionForm类,这样才能把一个表单参数转换成一个具体的formbean对象（所谓具体实质是指参数formbean对象已经能成功地赋值给User对象）并与相应的Action绑定。
但springmvc并不要求这种User一定要继承某个类，既然springmvc对这种User没有要求，那表单参数是怎样与User进行完美匹配的，注意在RegControl构造方法中有如下一句代码：setCommandClass(User.**class**); 这句代码就指明了此控制器绑定User类来和表单进行匹配。如果想验证此句代码的作用，可以注释掉这句代码并查看异常。后面将会分析这种控制器的一个执行过程（包括表单填充及验证过程）

概述此步要点：（1）继承SimpleFormController类（2）构造器中调用setCommandClass方法绑定定命令对象（这里为User类）（3）转换formbean为User类进行业务逻辑操作
步骤二：配置web.xml(和前面HelloWorld实例一样，在此省略)
步骤三：配置spmvc-servlet.xml文件，代码如下：

```
<bean id="viewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolve
        <property name="prefix" value="/WEB-INF/page/" />
        <property name="suffix" value=".jsp" />
```

```
</bean>

<bean id="simpleUrlHandlerMapping"   class="org.springframework.web.servlet.handler.SimpleUrlHa
        <property name="mappings">
                <props>
                        <prop key="/reg.do">regControl</prop>
                </props>
        </property>
</bean>
<bean id="regControl" class="com.asm.RegControl"></bean>
```

步骤四：根据配置文件完善相应页面

在index.jsp设定表单填写页面，主要代码如下：

```
<form action="<%=request.getContextPath()%>/reg.do" method="post">
用户名：<input type="text" name="username"><br/>
年龄：<input type="text" name="age"><br/>
<input type="submit">
</form>
```

/page/hello.jsp，主要代码如下：

```
<body>
世界，你好！(WEB-INF/page)
用户名：${user.username }
年龄：${user.age }
</body>
```

步骤五：启动服务器，访问到首页，填写表单完成测试。

1.3 spring-mvc入门（二）：后端控制器（下）

发表时间: 2011-03-07 关键字: MVC, Spring, Bean, Servlet, JSP

3.细研SimpleController控制器

在RegControl.java中增加如下代码：

```
protected Object formBackingObject(HttpServletRequest request) throws Exception {
                System.out.println("formBackingObject方法执行-->01");
                setCommandClass(User.class); //也可在此处调用setCommandClass方法
                return super.formBackingObject(request);
        }

        protected void initBinder(HttpServletRequest request, ServletRequestDataBinder binder)
                System.out.println("initBinder方法执行-->02");
                super.initBinder(request, binder);
        }

        protected void onBind(HttpServletRequest request, Object command) throws Exception {
                System.out.println("onBind方法执行-->03");
                super.onBind(request, command);
        }

        protected void onBindAndValidate(HttpServletRequest request, Object command, BindExcept
                        throws Exception {
                System.out.println("onBindAndValidate方法执行-->04");
                super.onBindAndValidate(request, command, errors);
        }
```

下面简要分析执行过程：

（1）.当前端控制器把请求转交给此此控制器后，会首先调用formBackingObject方法，此方法的作用就是根据绑定的Command Class来创建一个Command对象，因此除了可以在构造方法中调用setCommandClass方法，也可以在此处调用setCommandClass方法。其实创建这个Command对象很简单，spring通过如下代码完成：

BeanUtils.instantiateClass(this.commandClass);

由于在此处必须根据commandClass来完成Command对象的创建，因此在此方法调用前应保证commandClass设置完成，所以我们可以在formBackingObject方法和构造方法中完成commandClass的设置。

（2）.调用initBinder方法，初始化Command对象，即把表单参数与Command字段按名称进行匹配赋值。

（3）.调用onBind方法，把Command对象和后端控制器绑定。

（4）.调用onBindAndValidate方法，验证用户输入的数据是否合法。如果验证失败，我们可以通过修改errors参数，即新的errors对象将会绑定到ModelAndView上并重新回到表单填写页面。

（5）.执行processFormSubmission方法，主要操作就是把绑定的Command对象转换成一个User这样的表单对象，并调用业务逻辑方法操作User对象，根据不同的逻辑返回不同的ModelAndView对象。

4.MultiActionController控制器

此控制器来将多个请求处理方法合并在一个控制器里，这样可以把相关功能组合在一起（它和struts1.x中的DispatchAction极为相似）。下面通过实例演示此控制器的使用。

步骤一：在springMVC_02_controllerweb项目下，建立后端控制器UserManagerController.java，代码如下：

```
package com.asm;
//...省略导入的相关类
public class UserManagerController extends MultiActionController {
        public ModelAndView list(HttpServletRequest request, HttpServletResponse response) {
                ModelAndView mav = new ModelAndView("list");
                return mav;
        }

        public ModelAndView add(HttpServletRequest request, HttpServletResponse response) {
                ModelAndView mav = new ModelAndView("add");
                return mav;
        }

        public ModelAndView edit(HttpServletRequest request, HttpServletResponse response) {
                ModelAndView mav = new ModelAndView("edit");
                return mav;
        }

}
```

步骤二：配置web.xml（参前面实例），并在spmvc-servlet.xml中增加如下配置：

```
<bean id="springMethodNameResolver"             class="org.springframework.web.servlet.mvc.mult
                <property name="mappings">
```

```
                        <props>
                                <prop key="/list.do">list</prop>
                                <prop key="/add.do">add</prop>
                                <prop key="/edit.do">edit</prop>
                        </props>
                </property>
        </bean>

        <bean id="userManagerController"     class="com.asm.UserManagerController">
                <property name="methodNameResolver"
                        ref="springMethodNameResolver">
                </property>
        </bean>
```

说明：methodNameResolver负责从请求中解析出需要调用的方法名称。Spring本身已经提供了一系列MethodNameResolver的实现，当然也可以编写自己的实现。在这里我们使用了Pro方式来解析，具体表现如下：

<prop key="/list.do">list</prop> 请求list.do时调用list方法

<prop key="/add.do">add</prop> 请求为add.do时调用add方法

<prop key="/edit.do">edit</prop> 请求为edit.do时调用edit方法

然后通过把springMethodNameResolver解析器注入给UserManagerController的methodNameResolver，这样配置后才完成了一个真正的具有请求转发能力的MultiActionController控制器对象——UserManagerController **强调**：此步骤实质完成了一个工作：就是为UserManagerController控制器配置一个方法解析器。

步骤三：配置请求转发的访问路径，在spmvc-servlet.xml中添加如下代码

```
<bean id="simpleUrlHandlerMapping"              class="org.springframework.web.servlet.handler.
                <property name="mappings">
                        <props>
                                <prop key="/list.do">userManagerController</prop>
                                <prop key="/add.do">userManagerController</prop>
                                <prop key="/edit.do">userManagerController</prop>
                        </props>
                </property>
        </bean>
```

步骤四：根据配置文件，完善jsp页面编写。

page/list.jsp，代码如下：

```
<body>
  用户列表页面
</body>
```

page/add.jsp，代码如下：

```
<body>
  用户添加页面
</body>
```

page/edi.jsp，代码如下：

```
<body>
  用户修改页面
</body>
```

步骤五：启动服务器，访问.../list.do将调用到list方法并转向到list.jsp页面。

补充：细说MethodNameResolver解析器

InternalPathMethodNameResolver：默认MethodNameResolver解析器，从请求路径中获取文件名作为方法名。比如，.../list.do的请求会调用list(HttpServletRequest,HttpServletResponse)方法。

ParameterMethodNameResolver：解析请求参数，并将它作为方法名。比如，对应.../userManager.do?method=add的请求，会调用 add(HttpServletRequest, HttpServletResponse)方法。使用paramName属性定义要使用的请求参数名称。

PropertiesMethodNameResolver :使用用户自定义的属性（Properties）对象，将请求的URL映射到方法名，具体可以参见实例。

使用ParameterMethodNameResolver作为MethodNameResolver的解析器时，主要配置代码如下：

```
<bean id="simpleUrlHandlerMapping"              class="org.springframework.web.servlet.handler.
                <property name="mappings">
                        <props>
                                <prop key="/user.do">userManagerController</prop>
                        </props>
                </property>
        </bean>

        <bean id="ParameterMethodNameResolver"              class="org.springframework.web.ser
                <property name="paramName" value="crud"></property>
        </bean>

        <bean id="userManagerController"
```

```
                class="com.asm.UserManagerController">
                <property name="methodNameResolver"
                        ref="ParameterMethodNameResolver">
                </property>
        </bean>
```

访问路径为.../user.do?crud=list(add|edit)

1.4 spring-mvc入门（三）：映射处理器（上）

发表时间: 2011-03-17 关键字: Spring, MVC, Servlet, Bean, Web

三、映射处理器Handler Mapping

1.简析映射处理器

在spring mvc中,使用映射处理器可以把web请求映射到正确的处理器上，spring内置了很多映射处理器，而且我们也可以自定义映射处理器。下面的实例展示spring中最常用的两个映射处理器：BeanNameUrlHandlerMapping和SimpleUrlHandlerMapping。在正式开始前有必要了解以下相关要点：

（1）映射处理器都能把请求传递到处理器执行链接（HandlerExecutionChain）上,并且处理器执行链接必须包含能处理该请求的处理器（实质就是处理器链上动态添加了了此处理器，可以结合filter工作原理理解），而且处理器链接也能包含一系列拦截器。

（2）上面列举的spring最常用的两种处理器都是继承自AbstractHandlerMapping类，因而它们具备父类的属性。

2.实例：BeanNameUrlHandlerMapping

建立springMVC_03_handlerMappingsweb项目，并导入相关jar包。

步骤一：建立后端控制器MessageController.java，代码如下：

```
package com.asm;
//...省略导入的相关类
public class MessageController implements Controller {
        public ModelAndView handleRequest(HttpServletRequest arg0, HttpServletResponse arg1) th
                ModelAndView mav = new ModelAndView("message");
                mav.addObject("message", "您好！spring MVC");
                return mav;
        }
}
```

步骤二：配置web.xml，代码如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
        http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
        <servlet>
                <servlet-name>spmvc</servlet-name>
                <servlet-class>
```

```
                        org.springframework.web.servlet.DispatcherServlet
                </servlet-class>
                <load-on-startup>1</load-on-startup>
        </servlet>
        <servlet-mapping>
                <servlet-name>spmvc</servlet-name>
                <url-pattern>*.do</url-pattern>
        </servlet-mapping>
</web-app>
```

步骤三：配置spmvc-servlet.xml，代码如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:p="http://www.springframework.org/schema/p"
        xmlns:context="http://www.springframework.org/schema/context"
        xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
    http://www.springframework.org/schema/context http://www.springframework.org/schema/context
        <bean id="viewResolver"              class="org.springframework.web.servlet.view.Interr
                <property name="prefix" value="/WEB-INF/page/" />
                <property name="suffix" value=".jsp" />
        </bean>
        <bean name="/message.do" class="com.asm.MessageController"></bean>
</beans>
```

步骤四：在WEB-INF/page目录下建立message.jsp，主要代码如下：

```
<body>
   Message：${message}
</body>
```

步骤五：启动服务器，输入.../message.do访问测试。

简析执行过程

（1）启动服务器后，当我们向服务器发送message.do请求时，首先被在web.xml中配置的前端控制器DispatcherServlet拦截到。

（2）前端控制器把此请求转交给后端控制器，下面分析转交过程：当在spmvc-servlet.xml中查找能执行message.do请求的映射处理器时，发现没有能处理此请求的映射处理器，这时便使用默认的映射处理器BeanNameUrlHandlerMapping：This is the default implementation used by the DispatcherServlet, along with DefaultAnnotationHandlerMapping (on Java 5 and higher). 我们还需**注意**：这种后端控制器的bean Name必须以“/”开头，并且要结合DispatcherServlet的映射配置。同时beanName支持通配符配置。比如如果配置：<bean name="/m*.do" class="com.asm.MessageController" /> 时，当访问messasge.do时也可以成功访问到MessageController类。

（3）BeanNameUrlHandlerMapping处理器,会查找在spring容器中是否在名为“message.do”的bean实例。当查找到此实例后，则把此bean作为处理此请求的后端控制器。同时把自身加到映射处理器链上，并向处理器链传递此请求。

（4）后端控制器进行处理，并返回视图。

1.5 spring-mvc入门（三）：映射处理器（下）

发表时间: 2011-03-17 关键字: Spring, MVC, Bean, 配置管理, JSP

3.实例：SimpleUrlHandlerMapping

步骤一：建立后端控制器UserContrller.java.代码如下：

```
package com.asm;
//...省略导入的相关类
public class UserController extends SimpleFormController {
        protected ModelAndView processFormSubmission(HttpServletRequest request, HttpServletRes
                        Object command, BindException errors) throws Exception {
                System.out.println("调用逻辑层，处理表单");
                ModelAndView mav = new ModelAndView("loginSuc");
                return mav;
        }
}
```

步骤二：在spmvc-servlet.xml中增加如下配置：

```
<bean id="simpleUrlHandlerMapping"              class="org.springframework.web.servlet.handler.
<!-- 为映射处理器引入拦截器bean -->
                <property name="interceptors">
                        <list>
                                <ref bean="workTimeInterceptor" />
                        </list>
                </property>
                <property name="mappings">
                        <props>
                                <prop key="/op/*/login.do">userController</prop>
                        </props>
                </property>
        </bean>

        <bean id="userController" class="com.asm.UserController">
                <property name="commandClass" value="com.asm.User"/>
```

```
        </bean>

        <!-- 拦截器bean -->
        <bean id="workTimeInterceptor"
                class="com.asm.LoginTimeInterceptor">
                <property name="startTime" value="6" />
                <property name="endTime" value="18" />
        </bean>
```

说明：（1）通过前面实例我们可以知道，SimpleController这样的后端控制器必须绑定一个commandClass对象，在这里我们通过配置文件<property name="commandClass" value="com.asm.User"/>绑定。

（2）<prop key="/op/*/login.do">userController</prop>配置说明只要访问是以op开头，中间*可以是任意字符，并以login.do结尾的请求，便能访问到userController 控制器。

（3）SimpleUrlHandlerMapping是一个更强大的映射处理器，它除了支持上面<props>的这种配置，还支持Ant风格的路径匹配。另外也可以进行如下形式的配置：

```
<property name="mappings">
    <value>
      /op/*/login.do=userController
    </value>
</property>
```

（4）拦截器：为了为某些特殊请求提供特殊功能，spring为映射处理器提供了拦截器支持。它的配置文件很简单：一是把拦截器类纳入spring容器管理，二是在映射处理器引入配置的拦截器bean。

步骤三：编写拦截器LoginTimeInterceptor.java，主要代码如下：

```
package com.asm;
//...省略导入的相关类
public class LoginTimeInterceptor extends HandlerInterceptorAdapter {
        private int startTime;
        private int endTime;

        public void setStartTime(int startTime) {
                this.startTime = startTime;
        }
        public void setEndTime(int endTime) {
```

```
                this.endTime = endTime;
        }

        public void afterCompletion(HttpServletRequest request, HttpServletResponse response, C
                        Exception ex) throws Exception {
                System.out.println("执行afterCompletion方法-->03");
                super.afterCompletion(request, response, handler, ex);
        }

        public void postHandle(HttpServletRequest request, HttpServletResponse response, Object
                        ModelAndView modelAndView) throws Exception {
                System.out.println("执行postHandle方法-->02");
                super.postHandle(request, response, handler, modelAndView);
        }

        public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Obje
                        throws Exception {
                System.out.println("执行preHandle方法-->01");
                Calendar cal = Calendar.getInstance();
                int hour = cal.get(Calendar.HOUR_OF_DAY);
                if (startTime <= hour && hour < endTime) {
                        return true;
                } else {
                        response.sendRedirect("http://www.iteye.com");
                        return false;
                }
        }
}
```

说明：此拦截器作用：如果用户没有在6-18点登录，则重定向到javaeye站点（1）拦截器必须HandlerInterceptorAdapter接口（2）preHandle方法在后端控制器执行前被调用，postHandle方法在后端控制器执行后被调用；afterCompletion方法在整个请求处理完成后被调用。(3) preHandle方法：返回true，映射处理器执行链将继续执行；当返回false时，DispatcherServlet处理器认为拦截器已经处理完了请求，而不继续执行执行链中的其它拦截器和处理器。它的API文档解释如下：true if the execution chain should proceed with the next interceptor or the handler itself. Else, DispatcherServlet assumes that this interceptor has already dealt with the response itself.（4）这三个方法都是相同的参数，Object handler

参数可以转化成一个后端控制器对象，比如这里可以转换成UserController对象。

步骤四：完成其它相关代码的编写

User.java代码

```
package com.asm;
public class User {
   private String username;
   private String password;
   //省略getter/setter方法
}
```

WEB-INF/page/loginSuc.jsp，主要代码如下：

```
<body>
   登录成功!欢迎来到后台管理页面
</body>
```

index.jsp代码：

```
 <form action="<%=request.getContextPath()%>/op/luanXie/login.do" method="post">
   用户名：<input type="text" name="username"><br/>
   密　码：<input type="password" name="password"><br/>
   <input type="submit" value="登录">
 </form>
```

步骤五：访问index.jsp页面，完成测试。

分析执行过程：为了清晰体会到整个处理器执行过程，我们首先在UserController.java中增加如下代码：

```
protected Object formBackingObject(HttpServletRequest request) throws Exception {
                System.out.println("formBackingObject方法执行-->01");
                return super.formBackingObject(request);
        }
        protected void initBinder(HttpServletRequest request, ServletRequestDataBinder binder)
                System.out.println("initBinder方法执行-->02");
                super.initBinder(request, binder);
        }
        protected void onBind(HttpServletRequest request, Object command) throws Exception {
                System.out.println("onBind方法执行-->03");
                super.onBind(request, command);
        }
```

```
protected void onBindAndValidate(HttpServletRequest request, Object command, BindExcept
                throws Exception {
        System.out.println("onBindAndValidate方法执行-->04");
        super.onBindAndValidate(request, command, errors);
}
```

（1）当访问.../login.do时，会首先被前端控制器DispatcherServlet拦截到，前端控制器通过查找spmvc-servlet.xml配置文件，并交给后端控制器处理。

(2)执行后，得到如下打印结果，通过打印结果我们知道它的一个大致执行过程。

执行preHandle方法-->01

formBackingObject方法执行-->01

initBinder方法执行-->02

onBind方法执行-->03

onBindAndValidate方法执行-->04

调用逻辑层，处理表单

Admin----123456

执行postHandle方法-->02

执行afterCompletion方法-->03

1.6 spring-mvc入门（四）：视图与视图解析器（上）

发表时间: 2011-04-23 关键字: Spring, MVC, Excel, Bean, Servlet

四、视图与视图解析器

通常像spring mvc 这样的web框架都会有相应的定位视图技术，spring提供了视图解析器来解析ModelAndView模型数据到特定的视图上，spring提供了ViewResolver和View两个特别重要的接口，ViewResolver提供了从视图名称到实际视图的映射，View处理请求的准备的工作，并将该请求提交给某种具体的视图解析器

1.使用Excel作为视图（了解）

步骤一：建立后端控制器ExcelControl.java，主要代码如下：

```
package com.asm;
//...省略导入的相关类
public class ExcelControl extends AbstractController {
        @Override
        protected ModelAndView handleRequestInternal(HttpServletRequest request, HttpServletRes
                Map map = new HashMap();
                List wordList = new ArrayList();

                wordList.add("first");
                wordList.add("second");
                wordList.add("third");

                map.put("wordList", wordList);
                return new ModelAndView("exl", map);
        }
}
```

步骤二：编写web.xml（参前面实例），编写配置spmvc-servlet.xml文件，主要代码如下：

```
<bean id="rbViewResolver"       class="org.springframework.web.servlet.view.ResourceBundleViewR
        <property name="order" value="2" />
        <property name="basename" value="view" />
</bean>

<bean id="simpleUrlHandlerMapping"          class="org.springframework.web.servlet.handler.Sim
        <property name="mappings">
```

```
                <props>
                        <prop key="/getExcel.do">excelControl</prop>
                </props>
        </property>
</bean>

<bean id="excelControl" class="com.asm.ExcelControl"></bean>
```

简析：ResourceBundleViewResolver视图解析器会在classpath中寻找properties属性文件，根据此bean配置的<property name="basename" value="view" />说明寻找的properties文件名为view.properties。<property name="order" value="2" />说明了此解析器在整个解析链接中的顺序。

步骤三：编写view.properties文件，主要代码如下：

exl.(class)=com.asm.ExcelViewBuilder

说明：在控制器中返回的视图名称exl——>return new ModelAndView("exl", map)，所以这里要用exl.(class)。而exl视图会交给com.asm.ExcelViewBuilder进行预处理。

步骤四：编写AbstractExcelView.java，由于此类用到了生成Excel文件相关的jar包，所以应先下载poi-2.5.1.jar并导入。主要代码如下：

```
package com.asm;
//...省略导入的相关类
public class ExcelViewBuilder extends AbstractExcelView {
        protected void buildExcelDocument(Map model, HSSFWorkbook wb, HttpServletRequest req,
                HSSFSheet sheet;
                HSSFCell cell;

                sheet = wb.createSheet("Spring's Excel"); // 创建一张表；
                // sheet = wb.getSheetAt(0);//根据索引，得到第一张存在的表
                sheet.setDefaultColumnWidth((short) 10); // 设置列数为10：即A-J
                cell = getCell(sheet, 0, 0); //第二个参数-->excel表数字序号；第三个参数-->excel表字
                setText(cell, "Spring-Excel test"); // 把内容写入A1
                List words = (List) model.get("wordList");
                for (int i = 0; i < words.size(); i++) {
                        cell = getCell(sheet, 2, i);// A3-B3-C3
                        setText(cell, (String) words.get(i));
                }
        }
```

```
}
```

步骤五：启动服务器，输入.../getExcel.do，将访问到excel文件。

简析执行过程：当后端控制器接受到前端控制派发的请求时，后端控制器会首先准备Model，这里即Map对象，然后返回exl视图。在返回视图前，会查找spmvc-servlet.xml配置文件，当查找名为exl的视图是由ResourceBundleViewResolver视图解析器进行解析时，这时根据此视图解析的解析规则，会对每个待解析的视图，ResourceBundle里（这时即view.properties文件）的[视图名].class所对应的值就是实现该视图的类。 同样，[视图名].url所对应的值是该视图所对应的URL。当视图类成功完成视图的预处理工作后，会把此视图返回给客户端。

1.7 spring-mvc入门（四）：视图与视图解析器（下）

发表时间: 2011-04-23 关键字: Spring, MVC, freemarker, Bean, Excel

2.使用FreeMarker作为视图

步骤一：建立后端控制器FreeMarkerController.java，主要代码如下：

```
package com.asm;
//...省略导入的相关类
@SuppressWarnings("deprecation")
public class FreeMarkerController extends AbstractCommandController {
        @Override
        protected ModelAndView handleRequestInternal(HttpServletRequest reqeust, HttpServletRes
                        throws Exception {
                ModelAndView mav = new ModelAndView("base");
                mav.addObject("username", "张三");
                mav.addObject("time", new Date().toString());
                return mav;
        }

        @Override
        protected ModelAndView handle(HttpServletRequest request, HttpServletResponse response,
                        BindException errors) throws Exception {
                return null;
        }
}
```

步骤二：在spmvc-servlet.xml中配置：

```
<!-- freemarker相关的配置 -->
<bean id="freemarkerConfig"          class="org.springframework.web.servlet.view.freemarker.Fre
        <property name="defaultEncoding" value="UTF-8" />
        <property name="templateLoaderPath" value="/WEB-INF/freemarker/" />
</bean>
```

```
<!—freemarker视图解析器 -->
<bean id="fmViewResolver"            class="org.springframework.web.servlet.view.freemarker.Fre
        <property name="contentType" value="text/html;charset=utf-8" />
        <property name="cache" value="true" />
        <property name="prefix" value="" />
        <property name="suffix" value=".ftl" />
</bean>

<bean id="fmControl" class="com.asm.FreeMarkerController"></bean>
```

并在映射处理器中配置映射路径为：<prop key="/freeMarker.do">fmControl</prop>
步骤三：通过步骤二的配置，我们还需在WEB-INF/freemarker路径下编写base.ftl(base即后端控制器返回的视图名)，主要代码如下：

```
<body>
 欢迎来到：FreeMarker模板页面<br/>
 welcome ${username}<br/>
 当前时间：${time}
</body>
```

步骤四：启动服务器，输入.../freemarker.do完成测试。

3.小结视图技术
（1）ModelAndView所表示的视图名很关键，视图解析链会依此名来选择一个正确的视图。
（2）不同的视图解析器解析视图规则不相同，但是他们实质都是实现了ViewResolver接口，并会依赖于配置View对象来处理请求的准备工作。
（3）spring 内置了多种视图解析器，列表如下：

ViewResolver	描述
AbstractCachingViewResolver	抽象视图解析器实现了对视图的缓存。在视图被使用之前，通常需要进行一些准备工作。从它继承的视图解析器将对要解析的视图进行缓存。
XmlViewResolver	XmlViewResolver实现ViewResolver，支持XML格式的配置文件。 该配置文件必须采用与Spring XML Bean Factory相同的DTD。默认的配置文件是 /WEB-INF/views.xml。

ResourceBundleViewResolver	ResourceBundleViewResolver实现ViewResolver， 在一个ResourceBundle中寻找所需bean的定义。 这个bundle通常定义在一个位于classpath中的属性文件中。默认的属性文件是views.properties。
UrlBasedViewResolver	UrlBasedViewResolver实现ViewResolver， 将视图名直接解析成对应的URL，不需要显式的映射定义。 如果你的视图名和视图资源的名字是一致的，就可使用该解析器，而无需进行映射。
InternalResourceViewResolver	作为UrlBasedViewResolver的子类， 它支持InternalResourceView（对Servlet和JSP的包装）， 以及其子类JstlView和TilesView。 通过setViewClass方法，可以指定用于该解析器生成视图使用的视图类。 更多信息请参考UrlBasedViewResolver的Javadoc。
VelocityViewResolver / FreeMarkerViewResolver	作为UrlBasedViewResolver的子类， 它能支持VelocityView（对Velocity模版的包装）和FreeMarkerView以及它们的子类。

（4）Spring支持多个视图解析器一起使用，即视图解析链。 视图解析链包含一系列视图解析器，更方便开发人员处理某些特殊请求，比如在特定情况下重新定义某些视图（为某个视图解析器使用order，可以改变此视图解析器在整个视图解析链中的解析顺序：order值越大，它在整个视图解析链中的顺序越靠前，即它越会被优先选作为视图解析器）。

4.视图解析链

通过前面的两个实例，在spmvc-servlet.xml配置文件便具备了两个视图解析器，它们共同构成了视图解析链。下面我们将再增加一个视图解析器来解析jsp视图，并编写一个后端控制器来负责调用特定的请求。

步骤一：编写后端控制器.java，主要代码如下：

```
package com.asm;
//...省略导入的相关类
public class SelectViewController implements Controller {
        public ModelAndView handleRequest(HttpServletRequest request, HttpServletResponse respo
                String param = request.getParameter("op");
                ModelAndView mav = new ModelAndView("display");
                if ("fm".equals(param)) {
                        // mav = new ModelAndView(new RedirectView("freeMarker.do"));
                        // mav =new ModelAndView("redirect:freeMarker.do");
                        mav = new ModelAndView("forward:freeMarker.do");
                        return mav;
                } else if ("excel".equals(param)) {
                        mav = new ModelAndView(new RedirectView("getExcel.do"));
                        return mav;
```

```
                } else {
                        return mav;
                }
        }
}
```

简析：如果请求参数op为fm，则调用freemarker.do，如果op为excel，则调用getExcel.do，否则显示display.jsp视图。调用freemarker.do和getExcel.do我们可以使用重定向技术和直接使用forward跳转：使用forward跳转，forward:视图名；使用重定向，redirect:视图名（也可以RedirectView对象实现）。
步骤二：在spmvc-servlet.xml中增加jsp视图解析器（它与前面定义的两个视图解析器共同构成了视图解析链），并配置后端处理器及映射路径。

```
<bean id="irViewResolver"               class="org.springframework.web.servlet.view.InternalRes
        <property name="prefix" value="/WEB-INF/page/" />
        <property name="suffix" value=".jsp" />
</bean>

<bean id="simpleUrlHandlerMapping"
        class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
                <property name="mappings">
                        <props>
                                <prop key="/getExcel.do">excelControl</prop>
                                <prop key="/freeMarker.do">fmControl</prop>
                                <prop key="/sv.do">svControl</prop>
                        </props>
                </property>
</bean>
<bean id="svControl" class="com.asm.SelectViewController"></bean>
```

说明：irViewResolver视图解析器应放在前面两个视图解析器后面，理解视图解析顺序，可以把irViewResolver视图解析器放在视图解析链的最前面试下执行效果，当然也可以为视图解析器定义order值来进一步理解视图解析顺序。
步骤三：在WEB-INF/page目录下编写display.jsp， 主要代码如下：
<body>

欢迎来到display页面，你可以选择如下操作：
<a href="<%=request.getContextPath()%>/sv.do?op=excel">Excel页面
<a href="<%=request.getContextPath()%>/sv.do?op=fm">freeMarker页面
<a href="<%=request.getContextPath()%>/sv.do">jsp页面
</body>
步骤四：index.jsp页面主要代码如下：
<body>
<a href="<%=request.getContextPath()%>/sv.do?op=excel">Excel页面
<a href="<%=request.getContextPath()%>/sv.do?op=fm">freeMarker页面
<a href="<%=request.getContextPath()%>/sv.do">jsp页面
</body>
步骤五：启动服务器，访问首面测试。

1.8 spring-mvc入门（五）：使用注解（上）

发表时间: 2011-07-07

五、使用注解

1.简单实例

建立springMVC_05_annotation web项目，并导入相关的jar包。

步骤一：编写web.xml文件，主要代码如下：<servlet>

```
  <servlet-name>spmvc</servlet-name>
        <servlet-class>
                        org.springframework.web.servlet.DispatcherServlet
        </servlet-class>
        <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
        <servlet-name>spmvc</servlet-name>
        <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

步骤二：编写后端控制器，主要代码如下：package com.asm;

```
//...省略导入的相关类
@Controller
public class SimpleAnnotationControl {
@RequestMapping("/anno.do")
        public ModelAndView show() {
                ModelAndView mav = new ModelAndView("anno");
                mav.addObject("message", "welcome annotation demo");
                return mav;
        }
}
```

步骤三：编写springmvc-servlet.xml配置文件，主要代码如下：<?xml version="1.0" encoding="UTF-8"?>

```
<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:context="http://www.springframework.org/schema/context"
        xsi:schemaLocation="
http://www.springframework.org/schema/beans         http://www.springframework.org/schema/beans
http://www.springframework.org/schema/context         http://www.springframework.org/schema/cor

        <context:component-scan base-package="com.asm" />

<bean id="irViewResolver"            class="org.springframework.web.servlet.view.InternalResour
        <property name="prefix" value="/WEB-INF/page/" />
        <property name="suffix" value=".jsp" />
</bean>
</beans>
```

说明：在spring中，通过配置一个自动扫描器，可以自动扫描到某个范围下的所有组件，这些组件必须是标记有特定注解，比较常用的特定注解有@Component、@Repository、 @Service和@Controller。比如这里就会把SimpleAnnotationControl类纳入spring容器管理。关于spring自动扫描管理bean可以参它的相关文档。

步骤四：在WEB-INF/page目录下编写anno.jsp页面，主要代码如下：

```
<body>

  欢迎来到注解页面<br/>

  ${message}

</body>
```

步骤五：启动服务器，访问.../anno.do完成测试。

简析注解：在本实例中，我们使用了两个注解，一是Controller注解，此注解的作用是把控制器SimpleAnnotationControl纳入spring容器管理；二是@RequestMapping注解。下面我们重点分析此注解：在本例中，我们使用 @RequestMapping("/anno.do")这种简写形式，实际它的完整形式应为：@RequestMapping(value = "/anno.do")。value属性指明了它的映射路径，比如这里的映射路径为anno.do。此注解除了value属性外，还有如下属性：method、headers、params。

```
@RequestMapping(value = "/anno.do", method = RequestMethod.GET)
```

此配置说明只有GET请求才能访问anno.do。可以增加如下POST请求代码测试：

```
<form action="<%=request.getContextPath() %>/anno.do" method="post">
    <input type="submit" value="POST方式">
</form>
```

发现此种POST请求方式是不能访问anno.do得

@RequestMapping(value = "/anno.do", params = { "username=admin", "password=123456" }) 此配置说明只有附带正确的请求参数才能访问，比如在这里只能是.../anno.do?username=admin&password=123456才能正确访问。

1.9 spring-mvc入门（五）：使用注解（下）

发表时间: 2011-07-07

2.类级别的基路径请求

在上面的实例中，我们通过为方法配置请求路径来进行访问，而下面我们将为类配置一个请求实例，这种类似于struts2中package-namespace。

控制器代码如下：package com.asm;

```
//...省略导入的相关类
@Controller
@RequestMapping("/myRoot")
public class AnnotationControl {
        @RequestMapping(value = "/the/{name}.do")
        public String getName(@PathVariable
        String name, Model model) {
                model.addAttribute("message", "名字：" + name);
                return "anno";
        }

        @RequestMapping("/age.do")
        public ModelAndView getAge(@RequestParam
        int age) {
                ModelAndView mav = new ModelAndView("anno");
                mav.addObject("message", "年龄：" + age);
                return mav;
        }
}
```

（1）访问getAge方法

这样如果想访问到上面的两个方法都必须是以myRoot作为开头。下面我们先对getAge方法进行访问测试，在index.jsp页面增加如下代码：

```
<form action="<%=request.getContextPath() %>/myRoot/age.do" method="post">
```

```
<input type="text" name="age">
<input type="submit" value="提交年龄">
</form>
```

这样提交请求时，将会访问到getAge方法，并且age会作为参数传递给此方法，这样我们便可以得到客户端输入的age。 @RequestParam注解：Annotation which indicates that a method parameter should be bound to a web request parameter，意为此注解将会为方法中的参数绑定对应（对应是指名字上，比如这里表单和方法中参数名都用了age）的web请求参数。**强调**：访问路径必须是以myRoot作为一个基路径，因为类上配置了请求基路径。@RequestParam注解实现web请求到方法参数的绑定。

（2）访问getName方法

对于getName方法它的访问路径应该是这样的：.../myRoot/the/jack.do 这样我们得到@PathVariable：Annotation which indicates that a method parameter should be bound to a URI template variable，意思是此注解将会把一个uri路径中对应的变量和方法中的参数绑定，这里我们用jack来代替来{name}，所以实质就是把jack作为参数和方法中name参数绑定，如果想传递其它值，只须把jack换下就可以了，比如：.../myRoot/the/tom.do等。

3.自动表单

在struts框架中，表单参数能成功地交给struts Action来处理，在前面的实例中我们使用springmvc也完成了类似功能，下面我们使用spring mvc注解方式来完成表单参数和业务层实体的绑定。

步骤一、准备业务层实体Bean,User.java主要代码如下：

```
package com.asm;
public class User {
        private String username;
        private String password;
//省略getter/setter方法。
}
```

步骤二：编写控制层代码

```
package com.asm;
//省略导入的相关类
@Controller
public class FormAnnotationControl {
        @ModelAttribute("user")
        public User initUser() {
                User user = new User();
                user.setUsername("在此处填写用户名");
                return user;
        }

        @RequestMapping("reg.do")
        public String addUI() {
                return "reg";
        }

        @RequestMapping("save.do")
        public String add(@ModelAttribute
        User user, Model model) {
                model.addAttribute(user);
                return "userInfo";
        }

        @RequestMapping("login.do")
        public ModelAndView login(@ModelAttribute
        User user) {
                ModelAndView mav = new ModelAndView(new RedirectView("manage.do"));
                if (!"admin".equals(user.getUsername())) {
                        mav = new ModelAndView("error");
                }
                return mav;
        }

        @RequestMapping("manage.do")
        public String manage() {
                return "list";
```

```
        }
}
```

说明：@ModelAttribute ：Annotation that binds a method parameter or method return value to a named model attribute, exposed to a web view. Supported for `RequestMapping` annotated handler classes，此注解可以用于web请求参数和方法参数的绑定，也可用于方法的返回值作为模型数据（这种模型数据在类中全局有效，比如这里initUser方法绑定的模型数据在此类的任何方法中都能访问到这种模型数据）。

步骤三、编写相关的jsp页面

reg.jsp

```
 <form action="<%=request.getContextPath() %>/save.do" method="post">
  用户名：<input type="text" name="username" value="${user.username }"><br/>
  密　码：  <input type="password" name="password" value="${user.password }"><br/>
  <input type="submit" value="保存用户">
 </form>
```

userInfo.jsp

```
 <body>
  保存用户成功，信息如下:<br/>
  用户名：${user.username}<br/>
  密　码：${user.password}
 </body>
```

list.jsp

```
<body>
  列出所有用户
</body>
```

error.jsp

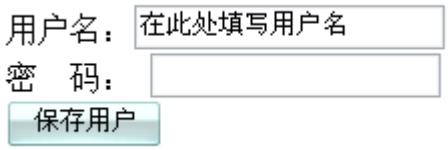
```
<body>

  用户你好，你没权力进入后台

</body>
```

步骤四、输入相关信息进行测试。

reg.do,对应于addUI()方法的访问：我们输入..../reg.do，可以看到页面显示如下内容：



“在此处填写用户名”，这几个字是由于我们在reg.jsp中使用了${user.usernam}，并且在initUser方法中我们通过ModelAttribute注解初始化了一个模型数据，这样我们便可以在此类对应的任何方法映射的视图中访问到此模型数据，这也即是前面提到这种模型数据在类中全局有效。然后填写相关数据，提交表单给sava.do，即是提交给给save方法，在save方法中我们同样使用了ModelAttribute注解，此注解在save方法中实现了web请求参数和方法参数的绑定的功能，也即是说把web中username、password和ModelAttribute指示的user中的username、password进行绑定，这种绑定和struts2中的模型驱动ModerDriven极为相似。

步骤五、访问登录页面，在login方法中我们对登录进行了简单的校验，如果用户名是admin则允许后台进行管理操作，如果不是则提示用户没有权力进行此操作。在前面我们对save方法进行了简要分析，这里的login方法也是类似得。另在此方法中如果是admin登录我们**重定向**到管理页面，如果不是admin登录，我们使用**forward**进行转发。访问login.do的jsp页面代码如下：

```
<form action="<%=request.getContextPath() %>/login.do" method="post">

  <input type="text" name="username">

  <input type="submit" value="登入管理后台">

</form>

<!--[endif]-->
```