**Spring MVC**

# 1 Spring MVC简介

## Spring3 MVC VS Struts2

* 官方的下载网址是：<http://www.springsource.org/download>
* 我们用Struts2时采用的传统的配置文件的方式,并没有使用传说中的0配置Spring3 mvc可以认为已经100%零配置了
* Spring会比Struts快,在Struts中默认配置了很多缺省的拦截器,在给开发者提供便利的同时,失去了灵活性和效率.
* Spring mvc是基于方法的设计, 而Sturts是基于类,每次发一次请求都会实例一个action，每个action都会被注入属性，而spring基于方法，粒度更细,可控制更强
* 设计思想上：Struts更加符合oop的编程思想,Spring是在在Servlet上扩展，使用AOP实现。
* Intercepter的实现机制：Struts有以自己的interceptor机制，Spring mvc用的是独立的AOP方式. 需要的自己配置,比较麻烦当性能较优,灵活性高
* 提供约定大于配置的契约式编程支持

# 2 环境与入门案例

## 2.1使用示例

### 2.1.1创建项目并导入相关jar包

mvc/aop/core相关包。

### 2.1.2创建配置文件

新建spring-mvc.xml文件

<?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"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.1.xsd*

*"*>

</beans>

### 2.1.3 配置前端过滤器

org.springframework.web.servlet.DispatcherServlet，继承HttpServlet，需要在Web.xml文件中定义

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<web-app version=*"3.0"*

xmlns=*"http://java.sun.com/xml/ns/javaee"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee*

*http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd"*>

<display-name></display-name>

<welcome-file-list>

<welcome-file>index.jsp</welcome-file>

</welcome-file-list>

<servlet>

<servlet-name>mvc</servlet-name> <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:spring-mvc.xml</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>mvc</servlet-name>

<url-pattern>\*.action</url-pattern>

</servlet-mapping>

</web-app>

### 2.1.4创建控制器

创建控制器FirstAction.java，实现Controller接口

// action实现Controller接口，并实现handleRequest方法（类似service方法），与JSP内置对象偶合

**public** **class** FirstAction **implements** Controller {

@Override

**public** ModelAndView handleRequest(HttpServletRequest request,

HttpServletResponse response) **throws** Exception {

// **TODO** Auto-generated method stub

System.*out*.println("传入的数据为");

String userName=request.getParameter("userName");

String pwd=request.getParameter("pwd");

System.*out*.println("userName:"+userName);

System.*out*.println("这里可以调用业务层处理业务。");

//封装数据，可以直接使用request对象，也可以使用封装等方式，真正使用时可以选择一种

request.setAttribute("rUserName", userName);

Map<String, String > map=**new** HashMap<String, String>();

map.put("mUserName",userName);

//返回视图层，如果使用map封装数据，需要作为（第二个）参数传递，也是request作用域。

**return** **new** ModelAndView("/jsp/first.jsp",map);

//返回视图层,不传递map。

//return new ModelAndView("/jsp/first.jsp");

//也可以使用如下方式传递，不使用Map，数据一样是request作用域

//return new ModelAndView("/jsp/first.jsp","mUserName",userName);

}

}

### 2.1.5修改配置文件，添加控制器信息

修改spring-mvc.xm.文件

<?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"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.1.xsd*

*"*>

<!--

1。配置action，实现controler接口

2。配置映射处理器，用来处理请求与action的映射，可以不用写id,

3。配置视图解析器：完成ModelAndView的解析

缺点：

1。与JSP偶合

2。只支持属性的注入，不支持封闭后对象注入

-->

<!-- 声明bean的name，因为使用了BeanNameUrlHandlerMapping，所以不是定义id,用户调用的URL将通过bean的name匹配 -->

<bean name=*"/first.action"* class=*"cn.itcast.action.FirstAction"* />

<!-- 声明 BeanNameUrlHandlerMapping，使用名称映射-->

<bean class=*"org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping"* />

<!-- 支持servlet与jsp视图解析，可进行进一步处理，此步可省略， -->

<!-- InternalResourceViewResolver支持servlet与jsp视图解析，没有配置时，默认使用它，此步可省略， -->

<bean class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>

<!-- 可以加前缀或后缀 -->

<!--

<property name="prefix" value="/jsp/"/>

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

-->

</bean>

### 2.1.6创建结果展现页面

新建目录jsp及目录下新建first.jsp，用来展现访问结果。

<%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"utf-8"*%>

<html>

<head>

<title>My JSP 'index.jsp' starting page</title>

</head>

<body>

这是/jsp/first.jsp页面.<br/>

<!-- 获取并展现控制层传递过来的值 -->

直接通过request传递的值：${requestScope.rUserName}<br/>通过Map传递的值：${requestScope.mUserName}

</body>

</html>

### 2.1.7编写测试(访问）页面

编写index.jsp用来访问控制器

<%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"utf-8"*%>

<html>

<head>

<title>My JSP 'index.jsp' starting page</title>

</head>

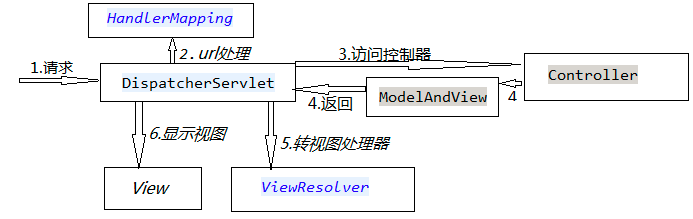
<body>

<a href=*"first.action?userName=zcf&pwd=admin"*>firstMVC</a>

</body>

</html>

## 2.2流程分析



# 3 URL处理器

## 3.1 BeanNameUrlHandlerMapping

BeanNameUrlHandlerMapping：它将收到的HTTP请求映射到bean的名称(这些bean需要在web应用上下文中定义)

<!-- 声明bean的name，因为使用了BeanNameUrlHandlerMapping，所以不是定义id,用户调用的URL将通过bean的name匹配 -->

<bean name=*"/first.action"* class=*"cn.itcast.action.FirstAction"* />

<!--

声明 BeanNameUrlHandlerMapping，使用名称映射

-->

<bean class=*"org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping"* />

## 3.2 SimpleUrlHandlerMapping

SimpleUrlHandlerMapping：它将收到的HTTP请求映射到bean的ID(这些bean需要在web应用上下文中定义)

<!-- 配置URL与ACTION对象ID进行映射 ,<prop key="second.action">second</prop>,其中key匹配url信息,value为action的ID

-->

<bean id=*"first"* class=*"cn.itcast.action.FirstAction"* />

<bean class=*"org.springframework.web.servlet.handler.SimpleUrlHandlerMapping"*>

<property name=*"mappings"*>

<props>

<prop key=*"first.action"*>first</prop>

</props>

</property>

</bean>

# 4 视图解析器

## 4.1 UrlBasedViewResolver

<!-- 支持servlet与jsp视图解析1，可进行进一步处理，此步可省略， -->

<!-- viewClass不同的配置,可以解析不同的资源-->

<bean class="org.springframework.web.servlet.view.UrlBasedViewResolver">

<property name="viewClass" value="org.springframework.web.servlet.view.JstlView"/>

<!--

<property name="prefix" value="/jsp/"/>

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

-->

</bean>

## 4.2 InternalResourceViewResolver

作为UrlBasedViewResolver的子类， 它支持页面jstl处理.

<!-- 支持servlet与jsp视图解析，可进行进一步处理，此步可省略， -->

<bean class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>

<!-- 可以加前缀或后缀 -->

<property name=*"prefix"* value=*"/jsp/"*/>

<property name=*"suffix"* value=*".jsp"*/>

</bean>

## 4.3 forward:前缀

forward使用转发方式：

**return** **new** ModelAndView("/jsp/first.jsp",map);

**return** **new** ModelAndView("forward:/jsp/first.jsp",map);

//控制器采用注解，方法在返回字符串时，可以使用：

**return** "forward:/jsp/first.jsp";

## 4.4 Redirect:前缀

redirect重定向方式

**return** **new** ModelAndView("redirect:/jsp/first.jsp",map);

//控制器采用注解，方法在返回字符串时，可以使用

**return** "redirect:/jsp/first.jsp";

# 5 控制器

## 5.1 controller接口

在spring mvc中控制对象要实现此接口，并且必须实现handRequest方法。此控制器在接收到DispatcherServlet分配置 的请求时，执行handRequest方法，并 返回ModelAndView实例。

**public** **class** FirstAction **implements** Controller {

@Override

**public** ModelAndView handleRequest(HttpServletRequest request,

HttpServletResponse response) **throws** Exception {

...

}

## 5.2 AbstractCommandController

可以将请求参数值自动设置到command对象中，便于后继的使用。

### 5.2.1添加student实体类

**public** **class** Student **implements** Serializable {

@Override

**public** String toString() {

**return** "Student [stuId=" + stuId + ", stuName=" + stuName + ", stuPwd="

+ stuPwd + ", stuAge=" + stuAge + "]";

}

/\*\*

\*

\*/

**private** **static** **final** **long** *serialVersionUID* = 1785249781500211272L;

**private** Integer stuId;

**private** String stuName;

**private** String stuPwd;

**private** Integer stuAge;

**public** Integer getStuId() {

**return** stuId;

}

**public** **void** setStuId(Integer stuId) {

**this**.stuId = stuId;

}

**public** String getStuName() {

**return** stuName;

}

**public** **void** setStuName(String stuName) {

**this**.stuName = stuName;

}

**public** String getStuPwd() {

**return** stuPwd;

}

**public** **void** setStuPwd(String stuPwd) {

**this**.stuPwd = stuPwd;

}

**public** Integer getStuAge() {

**return** stuAge;

}

**public** **void** setStuAge(Integer stuAge) {

**this**.stuAge = stuAge;

}

}

### 5.2.2创建或修改控制器类

**public** **class** StudentAction **extends** ~~AbstractCommandController~~ {

**public** StudentAction(){

//配置student对象可以注入

setCommandClass(Student.**class**);

}

@Override

**protected** ModelAndView handle(HttpServletRequest arg0,

HttpServletResponse arg1, Object arg2, BindException arg3)

**throws** Exception {

// **TODO** Auto-generated method stub

System.*out*.println("---接收数据---");

//方式1接收数据，只能每个属性都分开接书

String stuName=arg0.getParameter("stuName");

String stuPwd=arg0.getParameter("stuPwd");

System.*out*.println("方式1接收的数据为："+stuName+","+stuPwd);

//方式2接收数据，实现对象属性注入

Student student = (Student)arg2;

System.*out*.println("方式2接收的数据为："+student);

System.*out*.println("---调用业务层，进行业务处理，略---");

//封装视图数据，有多种方式 ，这里列表方式 一和方式 二，可以任选一种：

//方式一,直接采用request对象封装

arg0.setAttribute("rStudent", student);

//封装视图数据，方式二,直接采用Map封装，默认作用域是request，需要在return的时候作为参数传递。

Map<String ,Student > map=**new** HashMap<String, Student>();

map.put("mStudent", student);

//默认为forward方式

**return** **new** ModelAndView("/jsp/main.jsp",map);

}

}

### 5.2.3添加或修改spring-mvc.xml文件

<?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"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.1.xsd*

*"*>

<bean id=*"student"* class=*"cn.itcast.action.StudentAction"*></bean>

<bean class=*"org.springframework.web.servlet.handler.SimpleUrlHandlerMapping"*>

<property name=*"mappings"*>

<props>

<prop key=*"student.action"*>student</prop>

</props>

</property>

</bean>

</beans>

### 5.2.4添加跳转页面

/jsp/main.jsp

<%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"utf-8"*%>

<html>

<head>

<title>My JSP 'main.jsp' starting page</title>

</head>

<body>

这是/jsp/main.jsp页面.<br/>

<!-- 获取并展现控制层传递过来的值 -->

直接通过request传递的值：${requestScope.rStudent}<br/>通过Map传递的值：${requestScope.mStudent}

</body>

</html>

添加登陆测试页面

index.jsp

<%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"utf-8"*%>

<html>

<head>

<title>My JSP 'index.jsp' starting page</title>

</head>

<body>

<a href=*"student.action?stuName=zcf&stuPwd=admin"*>test student</a>

</body>

</html>

## 5.3 MultiActionController

### 5.3.1准备工作

除action类以外，其它继续使用上一节代码

### 5.3.2添加StudentMultiAction.java类

**public** **class** StudentMultiAction **extends** MultiActionController {

//定义方法时，参数规则：(HttpServletRequest request, HttpServletResponse response, [,HttpSession session] [,MyObject]);

**public** ModelAndView add(HttpServletRequest request,HttpServletResponse response,Student student){

System.*out*.println("add.student:"+student);

student.setStuName("rename");

**return** **new** ModelAndView("jsp/main","student",student);

}

//定义方法时，参数规则：(HttpServletRequest request, HttpServletResponse response, [,HttpSession session] [,MyObject]);

**public** ModelAndView update(HttpServletRequest request,HttpServletResponse response,Student student){

System.*out*.println("update.student:"+student);

student.setStuName("rename");

**return** **new** ModelAndView("jsp/main","student",student);

}

//定义方法时，参数规则：(HttpServletRequest request, HttpServletResponse response, [,HttpSession session] [,MyObject]);

**public** ModelAndView list(HttpServletRequest request,HttpServletResponse response,Student student){

System.*out*.println("list.student:"+student);

student.setStuName("updateName");

**return** **new** ModelAndView("jsp/main");

}

}

### 5.3.3修改spring-mvc.xml文件

<?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"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.1.xsd*

*"*>

<!-- 配置控制器，并指定通过methodNameResolver方法名调用相关方法处理相关逻辑-->

<bean id=*"studentMultiAction"* class=*"cn.itcast.action.StudentMultiAction"*>

<property name=*"methodNameResolver"* ref=*"parameterMethodNameResolver"*></property>

</bean>

<!-- 定义通过方法名调用控制器相关方法的规则 -->

<bean id=*"parameterMethodNameResolver"*

class=*"org.springframework.web.servlet.mvc.multiaction.ParameterMethodNameResolver"*>

<!-- 在url中使用do=方法名方式识别相关方法，例如：studentMulti.action?do=add，将调用add方法；这里的do不是固定的，可以改为其它 -->

<property name=*"paramName"* value=*"do"* />

<!-- 如果没有指定方法名时，默认 调用控制器的list方法 -->

<property name=*"defaultMethodName"* value=*"list"* />

</bean>

<bean class=*"org.springframework.web.servlet.handler.SimpleUrlHandlerMapping"*>

<property name=*"mappings"*>

<props>

<prop key=*"/studentMulti.action"*>studentMultiAction</prop>

</props>

</property>

</bean>

<bean class=*"org.springframework.web.servlet.view.UrlBasedViewResolver"*>

<property name=*"viewClass"*

value=*"org.springframework.web.servlet.view.JstlView"*></property>

<property name=*"prefix"* value=*"/WEB-INF/"* />

<property name=*"suffix"* value=*".jsp"* />

</bean>

</beans>

### 5.3.4测试页面

index.jsp关键代码

<body>

<form action=*"studentMulti.action?do=add"* method=*"post"*>

<input type=*"text"* name=*"stuName"*><br> <input

type=*"password"* name=*"stuPwd"*><br> <input type=*"submit"*

value=*"student\_add"*>

</form>

<a href=*"studentMulti.action?do=update&stuPwd=testpwd&stuName=testName"*>调用修改方法</a>

<a href=*"studentMulti.action?&stuPwd=testpwd&stuName=testName"*>调用默认方法</a>

</body>

### 5.3.5结果显示页面

/WEB-INF/jsp/main.jsp关键代码

<body>

this is WEB-INF/JSP main jsp<br>

studentName:${requestScope.student.stuName}<br>

</body>

# 6 基于注解的MVC实现

## 6.1示例1

继续使用上一章节的代码（注意新建项目记得重新配置web.xml文件）

### 6.1.1修改spring-mvc.xml文件

添加DefaultAnnotationHandlerMapping，AnnotationMethodHandlerAdapter等相关信息。其中

DefaultAnnotationHandlerMapping：支持通过url找到相关的action

AnnotationMethodHandlerAdapter：支持通过url匹配action定义方法

base-package：定 义扫描的范围,spring可以自动去扫描base-pack下面或者子包下面的java文件，如果扫描到有@Component @Controller@Service等这些注解的类，则把这些类注册为bean

<?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"*

xmlns:p=*"http://www.springframework.org/schema/p"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.1.xsd*

*http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-3.1.xsd*

*"*>

<!--

DefaultAnnotationHandlerMapping：支持通过url找到相关的action

AnnotationMethodHandlerAdapter：支持通过url匹配action定义方法

base-package：定 义扫描的范围,spring可以自动去扫描base-pack下面或者子包下面的java文件，如果扫描到有@Component @Controller@Service等这些注解的类，则把这些类注册为bean

-->

<bean class=*"org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping"*></bean>

<bean class=*"org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter"*></bean>

<context:component-scan base-package=*"\*"*></context:component-scan>

</beans>

### 6.1.2添加或修改控制类

加入@Controller，@RequestMapping注解信息

@Controller //用来声明控制器

@RequestMapping("/student")

**public** **class** StudentAction {

**public** StudentAction(){

System.*out*.println("---StudentAction构造方法被调用---");

}

//访问可用student/save.action,save后边的action是根据web.xml配置来的

//如果要添加其它的数据到最后跳转过去的页面，可以在方法中添加ModelMap的参数，例如 : public String save(Student student,ModelMap map){

//...,通过map再存放其它的数据

@RequestMapping(value="/save")

**public** ModelAndView save(Student student){

System.*out*.println("save方法注入的student对象："+student);

System.*out*.println("---调用业务逻辑进行业务处理---");

//修改学生名字，跳转到下一页面时看能否显示修改后的名字

student.setStuName("rename");

//直接使用字符串，返回视图，进行结果展现等

**return** **new** ModelAndView("forward:/jsp/main.jsp");

}

//同一个action中可以定义多个方法,方法的返回类型也可以用String

@RequestMapping(value="/update")

**public** String update(Student student,ModelMap paramMap){

System.*out*.println("update方法已注入student对象："+student);

System.*out*.println("---调用业务逻辑进行业务处理---");

paramMap.put("other","testOtherValue");

//直接使用字符串，返回视图，进行结果展现等

**return** "forward:/jsp/main.jsp";

}

}

### 6.1.3添加或修改跳转页面

<%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"utf-8"*%>

<html>

<head>

<title>My JSP 'main.jsp' starting page</title>

</head>

<body>

这是/jsp/main.jsp页面.<br/>

<!-- 获取并展现控制层传递过来的值 -->

默认通过request传递的值：${requestScope.student}<br/>

</body>

</html>

### 6.1.4添加或修改测试页面

<%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"utf-8"*%>

<html>

<head>

<title>My JSP 'index.jsp' starting page</title>

</head>

<body>

<a href=*"student/save.action?stuName=zcf&stuPwd=admin"*>调用save方法</a>

<a href=*"student/update.action?stuName=zcf&stuPwd=admin"*>调用update方法</a>

</body>

</html>

## 6.2示例2（基于annotation-driven的注解）

基于上面的示例，在spring3中可以进一步简化配置，取代上面的注解方式.

步骤如下

1.使用上面的action类，仍然给类及方法添加@Controller（类）、@RequestMapping（类及方法）注解

2.本文件顶部添加spring mvc 命名空间的信息（可以参考org.springframework.web.servlet.config包）

3.添加下面注解驱动<mvc:annotation-driven>，取代了上面的DefaultAnnotationHandlerMapping，AnnotationMethodHandlerAdapter，并启动了json的注解

-->

修改内容如下：

### 6.2.1修改配置文件

修改spring-mvc.xml文件， 红色部分：

<?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:mvc=*"http://www.springframework.org/schema/mvc"*

xmlns:context=*"http://www.springframework.org/schema/context"*

xmlns:p=*"http://www.springframework.org/schema/p"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.1.xsd*

*http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-3.1.xsd*

*http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-3.1.xsd*

*"*>

<!--

<bean class="org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping"></bean>

<bean class="org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter"></bean>

<context:component-scan base-package="\*"></context:component-scan>

-->

<!-- mvc:annotation-driven，取代了上面的DefaultAnnotationHandlerMapping，AnnotationMethodHandlerAdapter

两个Bean的配置 -->

<mvc:annotation-driven></mvc:annotation-driven>

<context:component-scan base-package=*"\*"*/>

</beans>

## 6.3 @SessionAttributes与model.addAttribute使用

Spring 2.0 定义了一个 org.springframework.ui.ModelMap 类，它作为通用的模型数据承载对象，传递数据供视图所用。我们可以在请求处理方法中声明一个 ModelMap 类型的入参，Spring 会将本次请求模型对象引用通过该入参传递进来，这样就可以在请求处理方法内部访问模型对象了在默认情况下，ModelMap 中的属性作用域是 request 级别是，也就是说，当本次请求结束后，ModelMap 中的属性将销毁，但实际上有时候需要把ModelMap值存放于session中或有时候也可以从Session中获取对象的值注入到ModelMap中。

继续使用上一节代码

### 6.3.1 modelMap属性注入到Session

如果希望在多个请求中共享 ModelMap 中的属性，必须将其属性转存到 session 中，这样 ModelMap 的属性才可以被跨请求访问；

可以在定义 类时使用@SessionAttributes("属性名")或@SessionAttributes({“attr1”,”attr2”})等方式将尝试从modelMap中寻找相同属性名相应的value.

#### 修改StudentAction.java类,

@Controller

@RequestMapping("/student")

//下边如有多个属性可以用 @SessionAttributes({“attr1”,”attr2”})。

@SessionAttributes("user")

**public** **class** StudentAction {

**public** StudentAction(){

System.*out*.println("---StudentAction构造方法被调用---");

}

@RequestMapping(value="/save")

**public** String save(Student student,ModelMap map){

System.*out*.println("---调用业务逻辑进行业务处理---");

Student s2=**new** Student();

s2.setStuAge(11);

s2.setStuId(11111);

map.addAttribute("user", s2);//属性名必须与session一致

//map.addAttribute("stu", student);

//直接使用字符串，返回视图，进行结果展现等

**return** "forward:/jsp/main.jsp";

}

//同一个action中可以定义多个方法

@RequestMapping(value="/update")

**public** String update(Student student){

System.*out*.println("update方法已注入student对象："+student);

System.*out*.println("---调用业务逻辑进行业务处理---");

paramMap.put("student",student);

//直接使用字符串，返回视图，进行结果展现等

**return** "forward:/jsp/main.jsp";

}

}

#### 修改/jsp/main.jsp

<%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"utf-8"*%>

<html>

<head>

<title>My JSP 'main.jsp' starting page</title>

</head>

<body>

这是/jsp/main.jsp页面.<br/>

<!-- 获取并展现控制层传递过来的值 -->

默认直接通过request传递的值：${requestScope.student}<br/>

<!-- 默认直接通过session传递的值stu：${sessionScope.stu}<br/> -->

默认直接通过session传递user值：${sessionScope.user}<br/>

<!--下边的代码给下一示例使用:调用update方法测试把session的值注入到map中，此时session已经有user相关信息-->

<a href="../student/update.action">session的值注入到map中</a>

</body>

</html>

### 6.3.2 session属性注入到ModelMap

在参数中使用@ModelAttribute("user")，可以获取@SessionAttributes("user")值

继续使用上节代码

#### 修改StudentAction.java类,

定义类时继续使用@SessionAttributes("user")，并修改update方法，在参数中添加@ModelAttribute("user")：参数中的student的对象将由session中获取。

@Controller

@RequestMapping("/student")

//下边如有多个属性可以用 @SessionAttributes({“attr1”,”attr2”})。

@SessionAttributes("user")

**public** **class** StudentAction {

**public** StudentAction(){

System.*out*.println("---StudentAction构造方法被调用---");

}

@RequestMapping(value="/save")

**public** String save(Student student,ModelMap map){

System.*out*.println("---调用业务逻辑进行业务处理---");

Student s2=**new** Student();

s2.setStuAge(11);

s2.setStuId(11111);

s2.setStuName("testname");

map.addAttribute("user", s2);

//map.addAttribute("stu", student);

//直接使用字符串，返回视图，进行结果展现等

**return** "forward:/jsp/main.jsp";

}

//同一个action中可以定义多个方法

@RequestMapping(value="/update")

**public** String update(@ModelAttribute("user")Student student){

System.*out*.println("update方法已注入student对象："+student);

System.*out*.println("---调用业务逻辑进行业务处理---");

//直接使用字符串，返回视图，进行结果展现等

**return** "forward:/jsp/main.jsp";

}

}

# 课堂与课后练习

1。完成spring mvc基于配置方式开发环境搭建，并完成测试功能

2。完成spring mvc注解方式环境搭建，并完成以下场景：

add.jsp：提供学生登陆信息输入，并可以提交登陆

StudentAction：完成信息输出（控制台），并校验信息（校验条件自己设定），符合条件，把相关信息保存到session，并进入主页（main.jsp),显示登陆人员的信息；不符合登陆条件，则返回登陆页面。

# 7综合示例（springmvc文件上传）

## 7.1 multipartResolver使用

spring-mvc.xml文件添加如下内容：

<!--文件上传使用， 配置multipartResolver，id名为约定好的 -->

<bean id=*"multipartResolver"* class=*"org.springframework.web.multipart.commons.CommonsMultipartResolver"*>

<!-- 配置文件（每次上传的所有文件总大小)大小，单位为b, *1024000表示1000kb* -->

<property name=*"maxUploadSize"* value=*"1024000"* />

</bean>

## 7.2中文乱码处理

web.xml文件添加如下内容：

<filter>

<filter-name>encodingFilter</filter-name>

<filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>

<init-param>

<param-name>encoding</param-name>

<param-value>UTF-8</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>encodingFilter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

如果上边的方式设置后，仍然有乱码，请尝试修改tomcat安装目录下的

apache-tomcat安装目录\conf\server.xml文件，修改Connector元素内容,添加URIEncoding="UTF-8" ,修改后内容 如下：

<Connector port="8080" protocol="HTTP/1.1"

connectionTimeout="20000"

redirectPort="8443" URIEncoding="UTF-8"/>

## 7.3 properties文件信息注入

PropertiesFactoryBean：用来注入properties类型的配置文件信息

<!--PropertiesFactoryBean对properties文件可用 ，可以用来注入properties配置文件的信息 -->

<bean id=*"uploadProperties"* class=*"org.springframework.beans.factory.config.PropertiesFactoryBean"*>

<property name=*"location"* value=*"classpath:xxxxx.properties"*></property>

</bean>

## 7.4文件上传示例

### 7.4.1导入包

继续使用上一章节代码，并导入文件上传需要的jar包：commons-fileupload-1.2.2.jar, commons-io-2.0.1.jar

### 7.4.2修改student实体类，添加文件类型属性

**public** **class** Student **implements** Serializable {

**private** **static** **final** **long** *serialVersionUID* = -5304386891883937131L;

**private** Integer stuId;

**private** String stuName;

**private** String stuPwd;

**private** Integer stuAge;

**private** MultipartFile[] files;

**public** MultipartFile[] getFiles() {

**return** files;

}

**public** **void** setFiles(MultipartFile[] files) {

**this**.files = files;

}

**public** Integer getStuId() {

**return** stuId;

}

**public** **void** setStuId(Integer stuId) {

**this**.stuId = stuId;

}

**public** String getStuName() {

**return** stuName;

}

**public** **void** setStuName(String stuName) {

**this**.stuName = stuName;

}

**public** String getStuPwd() {

**return** stuPwd;

}

**public** **void** setStuPwd(String stuPwd) {

**this**.stuPwd = stuPwd;

}

**public** Integer getStuAge() {

**return** stuAge;

}

**public** **void** setStuAge(Integer stuAge) {

**this**.stuAge = stuAge;

}

@Override

**public** String toString() {

**return** "Student [stuId=" + stuId + ", stuName=" + stuName + ", stuPwd="

+ stuPwd + ", stuAge=" + stuAge + "]";

}

}

### 7.4.3编写上传页面

<%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"utf-8"*%>

<html>

<head>

<title>My JSP 'index.jsp' starting page</title>

</head>

<body>

<form action=*"student/save.action"* method=*"post"* enctype=*"multipart/form-data"*>

姓名：<input type=*"text"* name=*"stuName"*><br/>

密码<input type=*"password"* name=*"stuPwd"*><br>

请选择文件：<br/><input type=*"file"* name=*"files"*><br/>

<input type=*"file"* name=*"files"*><br/>

<input type=*"submit"* value=*"文件上传测试"*>

</form>

</body>

</html>

### 7.4.4编写控制器

StudentAction.java

@Controller

@RequestMapping("/student")

**public** **class** StudentAction {

**public** StudentAction(){

System.*out*.println("---StudentAction构造方法被调用---");

}

@RequestMapping("/save")

**public** String save(Student student) {

System.*out*.println("save方法已注入student对象："+student);

MultipartFile[] files=student.getFiles();

**for**(MultipartFile file:files){

**if**(file.isEmpty()){

System.*out*.println("文件为空");

}**else**{

System.*out*.println("文件不为空！");

System.*out*.println("格式:" + file.getContentType());

System.*out*.println("原名:" + file.getOriginalFilename());

System.*out*.println("大小:" + file.getSize());

System.*out*.println("表单控件的名称" + file.getName());

**try** {

FileUtils.*copyInputStreamToFile*(file.getInputStream(), **new** File("e:/testupload/"+file.getOriginalFilename()));

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

System.*out*.println("---调用业务逻辑进行业务处理---");

//直接使用字符串，返回视图，进行结果展现等

**return** "forward:/jsp/main.jsp";

}

}

### 7.4.5修改配置文件

添加文件处理器CommonsMultipartResolver配置

<?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:mvc=*"http://www.springframework.org/schema/mvc"*

xmlns:context=*"http://www.springframework.org/schema/context"*

xmlns:p=*"http://www.springframework.org/schema/p"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.1.xsd*

*http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-3.1.xsd*

*http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-3.1.xsd*

*"*>

<mvc:annotation-driven></mvc:annotation-driven>

<context:component-scan base-package=*"\*"*/>

<!--文件上传使用， 配置multipartResolver，id名称为约定好的 -->

<bean id=*"multipartResolver"* class=*"org.springframework.web.multipart.commons.CommonsMultipartResolver"*>

<!-- 配置文件（每次上传的所有文件总大小)大小,单位为b, 1024000表示1000kb -->

<property name=*"maxUploadSize"* value=*"1024000"* />

</bean>

</beans>

### 7.4.6编写处理完后跳转的页面

main.jsp

<%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"utf-8"*%>

<html>

<head>

<title>main.jsp</title>

</head>

<body>

/jsp/main.jsp页面

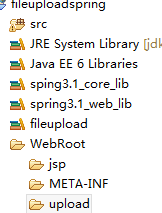
student： ${requestScope.student}

</body>

</html>

### 7.4.7文件存放于tomcat目录下处理方式

* 在项目目录下新建upload文件夹



* 修改StudentAction.java。

@Controller

@RequestMapping("/student")

**public** **class** StudentAction {

**public** StudentAction(){

System.*out*.println("---StudentAction构造方法被调用---");

}

@Resource

ServletContext application;

@RequestMapping("/save")

**public** String save(Student student) {

System.*out*.println("save方法已注入student对象："+student);

MultipartFile[] files=student.getFiles();

System.*out*.println("真实路径："+application.getRealPath("/"));

**for**(MultipartFile file:files){

**if**(file.isEmpty()){

System.*out*.println("文件为空");

}**else**{

System.*out*.println("文件不为空！");

System.*out*.println("格式:" + file.getContentType());

System.*out*.println("原名:" + file.getOriginalFilename());

System.*out*.println("大小:" + file.getSize());

System.*out*.println("表单控件的名称" + file.getName());

**try** {

FileUtils.*copyInputStreamToFile*(file.getInputStream(), **new** File(application.getRealPath("/")+"upload/"+file.getOriginalFilename()));

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

System.*out*.println("---调用业务逻辑进行业务处理---");

//直接使用字符串，返回视图，进行结果展现等

**return** "forward:/jsp/main.jsp";

}

}

其它代码同上一章节，可以在application.getRealPath("/")+"upload/"目录下查看到文件，例如 :

E:\Users\chufeng\Workspaces\MyEclipse10\itcast\.metadata\.me\_tcat\webapps\fileuploadspring\upload查看到上传的文件。

## 7.5 文件上传优化

### 7.5.1编写文件上传工具类

FileUploadUtil.java

@Component(value="fileUploadUtils") //普通的bean注入

**public** **class** FileUploadUtils {

/\*

\* 注入字符串,#{}为spel语言，其中uploadProperties，是xml配置文件中注入properties文件的bean id，

\* path为properties文件的其中一个key ，也可以通过下边的set方法注入

\*/

@Value("#{uploadProperties.path}")

**private** String path;

//private String path="e:/testupload";

//path也可以通过set方法注入

// @Value("#{uploadProperties.path}")

// public void setPath(String path) {

// this.path = path;

// }

**private** String getExtName(MultipartFile file){

**return** FilenameUtils.*getExtension*(file.getOriginalFilename());

}

**private** String createNewName(MultipartFile file){

**return** UUID.*randomUUID*().toString()+"."+getExtName(file);

}

**public** String uploadFile(MultipartFile file){

**try** {

String newName=createNewName(file);

FileUtils.*copyInputStreamToFile*(file.getInputStream(), **new** File(path,newName ));

**return** newName;

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

**throw** **new** RuntimeException(e);

}

}

}

### 7.5.2修改StudentAction.java

主要修改save方法，使用自已的文件上传工具类进行文件上传。

@Controller

@RequestMapping(value="/student")

**public** **class** StudentAction {

@Resource

**private** ServletContext application;

@Resource

**private** FileUploadUtils fileUploadUtils;

**public** StudentAction(){

System.*out*.println("---StudentAction构造方法被调用---");

}

@RequestMapping(value="/save")

**public** String save(Student student,Map<String, Student> paramMap) {

System.*out*.println("save方法已注入student对象："+student);

MultipartFile[] files=student.getFiles();

**for**(MultipartFile file:files){

**if**(file.isEmpty()){

System.*out*.println("文件为空");

}**else**{

System.*out*.println("文件不为空！");

fileUploadUtils.uploadFile(file);

}

}

System.*out*.println("---调用业务逻辑进行业务处理---");

paramMap.put("student",student);

//直接使用字符串，返回视图，进行结果展现等

**return** "forward:/jsp/main.jsp";

}

}

### 7.5.3添加upload.properties文件

配置文件上传后的存放目录

path=e\:\\testdir\\upload\\

### 7.5.4修改spring-mvc.xml配置文件

注入配置文件的信息

<!--PropertiesFactoryBean对properties文件可用 ，可以用来注入properties配置文件的信息 -->

<bean id=*"uploadProperties"* class=*"org.springframework.beans.factory.config.PropertiesFactoryBean"*>

<property name=*"location"* value=*"classpath:upload.properties"*></property>

</bean>

# 8 综合示例（登陆）

在综合示例1上继续。

## 8.1拦截器使用

### 8.1.1编写拦截器类

LoginInterceptor.java，需要实现HandlerInterceptor接口

**public** **class** LoginInterceptor **implements** HandlerInterceptor {

@Override

**public** **void** afterCompletion(HttpServletRequest arg0,

HttpServletResponse arg1, Object arg2, Exception arg3)

**throws** Exception {

// **TODO** Auto-generated method stub

System.*out*.println("---访问请求资源后不管理有没有异常都一定执行此方法---");

}

@Override

**public** **void** postHandle(HttpServletRequest arg0, HttpServletResponse arg1,

Object arg2, ModelAndView arg3) **throws** Exception {

// **TODO** Auto-generated method stub

System.*out*.println("---访问请求资源后，如果没有异常，将执行此方法---");

}

@Override

**public** **boolean** preHandle(HttpServletRequest arg0, HttpServletResponse arg1,

Object arg2) **throws** Exception {

// **TODO** Auto-generated method stub

System.*out*.println("---访问请求资源前执行，如果此方法返回false，将不能访问请求资源---");

**return** **true**;

}

}

### 8.1.2配置文件中添加拦截器

<!-- 配置spring mvc拦截器 -->

<mvc:interceptors>

<!-- 默认拦截DispatcherServlet指定的后缀（这里是.action） -->

<bean class=*"cn.itcast.interceptor.LoginInterceptor"*/>

</mvc:interceptors>

## 8.3登陆示例

### 8.3.1编写及配置拦截器

添加拦截器类及拦截器配置信息，如上面。

### 8.3.2修改拦截器类preHandle方法

@Override

**public** **boolean** preHandle(HttpServletRequest arg0, HttpServletResponse arg1,

Object arg2) **throws** Exception {

// **TODO** Auto-generated method stub

System.*out*.println("---访问请求资源前执行，如果此方法返回false，将不能访问请求资源---");

**if**(arg0.getSession().getAttribute("user")==**null**){

arg1.sendRedirect(arg0.getContextPath()+"/login.jsp");

**return** **false**;

}

**return** **true**;

}

### 8.3.3编写登陆页面

login.jsp,本页面已模仿了登陆

<%@page import=*"cn.itcast.entity.Student"*%>

<%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"utf-8"*%>

<html>

<head>

<title>My JSP 'index.jsp' starting page</title>

</head>

<body>

<%

session.setAttribute("user", **new** Student(1001,"zcf","admin",20));

%>

<!-- 这里正常应该跳到action再到页面 ,为了演示，这里简略-->

<a href=*"index.jsp"*>已登陆，返回首页</a>

</body>

</html>

# 9 json交互

使用上面的源码，暂时去掉拦截器的登陆权限处理

## 9.1导入json包及jquery的js文件

## 9.2修改action文件

@Controller

@RequestMapping(value="/student")

**public** **class** StudentAction {

**public** StudentAction(){

System.*out*.println("---StudentAction构造方法被调用---");

}

@RequestMapping("/doAjax")

@ResponseBody //如果返回json格式，需要这个注解

**public** Object doAjax(Student student){

System.*out*.println("---doAjax.student:"+student);

student.setStuName("1001name");

**return** student;

}

}

## 9.3修改访问页面

<%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"utf-8"*%>

<html>

<head>

<title>My JSP 'index.jsp' starting page</title>

<script type=*"text/javascript"* src=*"*${pageContext.request.contextPath}*/jquery-1.3.js"*></script>

<script type=*"text/javascript"*>

$(

**function**(){

$("#bt1").click(

**function**(){

$.post(

"student/doAjax.action",

{stuId:1001,stuName:"name1001",stuPwd:"pwd1001",stuAge:20},

**function**(json){alert(json.stuName+"||"+json.stuPwd);},

"json"

);

}

);

}

);

</script>

</head>

<body>

<button id=*"bt1"* >testajax</button>

</body>

</html>

# 10 ssi整合示例

换成新的版本

# 课后练习

完成ssm整合