[学习笔记]基于注解的 spring 3.0.x MVC 学习笔记(一)

学习 spring3.0.x(以下简称 spring 3)已经一段日子了,新特性也接触不少,比较感兴趣还是 spring mvc 这一块3.0的 mvc 变化太大了,跟2.5基本上是两个样子,至于详细的区别可以参考以下文章

领略 Spring 3.x 时代的 Spring MVC

spring3 mvc 变化比较大,但是还是有些2.5的影子的,首先来个2.5也可以使用的注解版本的 mvc 入门例子.例子采用 maven2管理,所以必须要安装 m2eclipse 插件或者使用 maven2进行管理.

本文基于 eclipse3.5 Galileo-sr2Javaee 版本跟 m2eclipse 插件进行管理.

m2eclipse 在线安装地址如下:

http://m2eclipse.sonatype.org/sites/m2e

如何安装请参考

Eclipse 中安装 SpringSource Tools Suite 插件

使用 eclipse 创建一个 maven project,选择 webapp Javaee5.0的模板项目或者 simpleproject,对于如何创建可以参考

[使用心得]maven2之 m2eclipse 使用手册之三第一个 Simple MavenProject 与 Pom.xml 配置说明

[使用心得]maven2之 m2eclipse 使用手册之六使用 Maven2插件创建一个简单的 SSH2项目之 jetty 篇(一)

对于 pom.xml 如下:

```
<build>
        <plugins>
            <plugin>
                <groupId>org.mortbay.jetty</groupId>
                <artifactId>maven-jetty-plugin</artifactId>
                <version>6.1.24</version>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>2.3.1</version>
                <configuration>
                    <source>1.6</source>
                    <target>1.6</target>
                    <encoding>utf-8</encoding>
                </configuration>
            </plugin>
        </plugins>
    </build>
    <dependencies>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
            <version>3.0.3.RELEASE</version>
            <type>jar</type>
            <scope>compile</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>3.0.3.RELEASE</version>
            <type>jar</type>
            <scope>compile</scope>
        </dependency>
        <dependency>
            <groupId>javax.servlet</groupId>
```

```
        <artifactId>servlet-api</artifactId>
        <version>2.5</version>
        <type>jar</type>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>javax.servlet.jsp</groupId>
        <artifactId>jsp-api</artifactId>
        <version>2.1</version>
        <type>jar</type>
        <scope>provided</scope>
    </dependency>

    <!-- Logging -->
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-api</artifactId>
        <version>${org.slf4j.version}</version>
    </dependency>
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>jcl-over-slf4j</artifactId>
        <version>${org.slf4j.version}</version>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-log4j12</artifactId>
        <version>${org.slf4j.version}</version>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>log4j</groupId>
        <artifactId>log4j</artifactId>
        <version>1.2.15</version>
        <exclusions>
            <exclusion>
                <groupId>javax.mail</groupId>
                <artifactId>mail</artifactId>
            </exclusion>
            <exclusion>
                <groupId>javax.jms</groupId>
                <artifactId>jms</artifactId>
            </exclusion>
            <exclusion>
                <groupId>com.sun.jdmk</groupId>
                <artifactId>jmxtools</artifactId>
            </exclusion>
            <exclusion>
                <groupId>com.sun.jmx</groupId>
                <artifactId>jmxri</artifactId>
            </exclusion>
        </exclusions>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>junit</groupId>
```

```
            <artifactId>junit</artifactId>
            <version>4.8.1</version>
            <type>jar</type>
            <scope>test</scope>
        </dependency>
    </dependencies>
    <properties>
        <org.slf4j.version>1.5.10</org.slf4j.version>
    </properties>
```

spring 的 mvc 本身就是一个 servlet,所以在 web.xml 加入以下内容

```
  <servlet>
        <servlet-name>StudySpringMVC</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <init-param>
            <description>springmvc.xml</description>
            <param-name>contextConfigLocation</param-name>
            <param-value>/WEB-INF/spring/*.xml</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>StudySpringMVC</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
```

其中 contextConfigLocation 是读取 spring 的配置文件进行解释,

springmvc.xml 如下:

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:aop="http://www.springframework.org/schema/aop"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
xmlns:tx="http://www.springframework.org/schema/tx"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
        http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-3.0.xsd
        http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-3.0.xsd
        http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd
        http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-3.0.xsd">


    <!-- 对 spring org.lxh 包下所有注解扫描 -->
    <context:component-scan base-package="org.lxh"></context:component-scan>
    <!-- 支持 spring mvc 新的注解类型 详细 spring3.0手册 15.12.1 mvc:annotation-driven-->
    <mvc:annotation-driven />


    <!--  对模型视图名称的解析，即在模型视图名称添加前后缀,在 requestmapping 输入的地址后自动调用该类进
行视图解析-->
    <bean id="viewResolver"
```

```
        class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="viewClass"
            value="org.springframework.web.servlet.view.JstlView" />
        <property name="prefix" value="/WEB-INF/jsp/" />
        <property name="suffix" value=".jsp"></property>
    </bean>

</beans>
```

UserController.Java 类:

```
@Controller
public class UserController {
    private Logger logger = org.slf4j.LoggerFactory
           .getLogger(UserController.class);

    @RequestMapping(value="/welcome.do",method=RequestMethod.GET)
    public void welcome() {
        logger.info("Welcome!");
        System.out.println("helloworld");
    }
}
```

@Controller:将一个类成为 Spring 容器的 Bean

@RequestMapping:指定该方法处理的 URL 请求,当在浏览器中输入该 url 地址的时候则会执行会执行指定的方法.

-----value:要运行的 url 地址

-----method:该方法在什么才执行.默认 GET

运行 jetty 插件命令 jetty:run

看到如下图:

```
2010-07-21 21:51:05.890:INFO::jetty-6.1.24
2010-07-21 21:51:06.201:INFO::No Transaction manager found - if your webapp requires one, please configure one.
2010-07-21 21:51:06.607:INFO:/StudySpringMVC:Initializing Spring FrameworkServlet 'StudySpringMVC'
```

看到该信息后则证明 spring 的 mvc 已经正常执行了

输入以下地址:http://localhost:8080/StudySpringMVC/welcome.do

看到如下图的内容:

注意:调用该方法的时候会自动搜索 bean id="viewResolver"的设置,然后查找该墓里下的 jsp 名字构成是所调用的方法名+.jsp,如果没有的话会出现后台 NOT_FOUND 的提示信息.

[学习笔记]基于注解的 spring3.0.x MVC 学习笔记(二)

用惯了 maven 管理项目之后会发现自己懒惰了很多,所以决定放弃使用 maven 去学习 spring3的 mvc,采用传统的 web project 进行学习,好了闲话不说,首先我们需要知道 spring mvc 需要加什么包.

使用到 spring mvc 的需要加入以下依赖包:

org.springframework.aop-3.0.3.RELEASE.jar----------Spring 的切面编程

org.springframework.asm-3.0.3.RELEASE.jar----------Spring 独立的 asm 程序

org.springframework.beans-3.0.3.RELEASE.jar----------SpringIoC(依赖注入)的基础实现

org.springframework.context-3.0.3.RELEASE.jar----------Spring 提供在基础 IoC 功能上的扩展服务

org.springframework.core-3.0.3.RELEASE.jar----------Spring 的核心包

org.springframework.expression-3.0.3.RELEASE.jar------Spring 表达式语言

org.springframework.web-3.0.3.RELEASE.jar----------SpringWeb 下的工具包

org.springframework.web.servlet-3.0.3.RELEASE.jar-----SpringMVC 工具包,并且对 JEE6.0 Servlet3.0的支持

除了 spring 的包之外还需要加入以下几个依赖包:

com.springsource.org.aopalliance-1.0.0.jar----------aop 的工具包

com.springsource.org.apache.commons.logging-1.1.1.jar----------commons 的日志管理

本文使用了 slf4j 日志管理,所以需要加入以下包

slf4j-api-1.5.10.jar----------日志管理的增强

slf4j-log4j12-1.5.10.jar----------日志包的连接层

所有包如图1springmvc 需要的包:

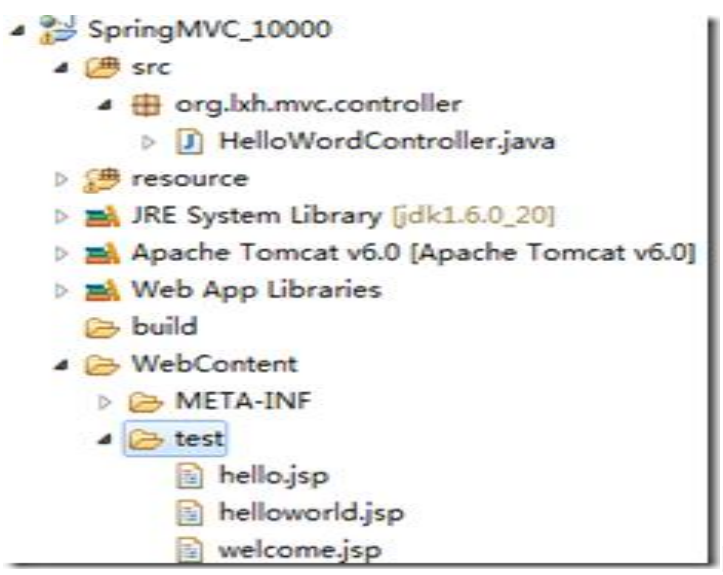


图1:springmvc 需要的包

实现的例子还是以 helloworld 的简单输出,以后再逐步加入数据库的操作,新建一个 resource 的 source folder,用来存放 spring 的配置文件跟 log4j.xml 的日志文件

使用日志管理就必须要加入 log4j.xml 或者是 log4j.properties 进行配置,这里采用了 Log4j.mxl 进行配置

Log4j.xml:

⊞

springmvc.xml

```
<!-- 对 spring org.lxh 包下所有注解扫描 -->
    <context:component-scan base-package="org.lxh"></context:component-scan>
    <!-- 支持 spring mvc 新的注解类型 详细 spring3.0手册 15.12.1 mvc:annotation-driven-->
    <mvc:annotation-driven />


    <!-- 对模型视图名称的解析，即在模型视图名称添加前后缀,在 requestmapping 输入的地址后自动调用该类进行视图解析-->
    <bean id="viewResolver"
        class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/" />
        <property name="suffix" value=".jsp"></property>
    </bean>
```

如果需要使用 jstl 的话则需要在<property>标签中加入

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

prefix:则为前缀,也就是目录的地址, 一般以"WEB-INF/xx"为主,若为 "/" 则为全局使用

suffix:则为后缀,也就是文件名的后缀.使用 InternalResourceViewResolver 类是只支持 jsp,不支持 html 等其他后缀,如果强制加入其他后缀的话胡出现死循环,至于其他视图的话则以后在使用的时候再介绍

web.xml

```
  <servlet>
```

```
        <description>This is Spring MVC DispatcherServlet</description>
        <servlet-name>SpringMVC DispatchServlet</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <init-param>
            <description>SpringContext</description>
            <param-name>contextConfigLocation</param-name>
            <param-value>classpath*:springmvc.xml</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>SpringMVC DispatchServlet</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
```

注意:配置 spring 的 DispatcherServlet 的时候,如果需要使用自定义名字的 spring 配置文件的话,需要在 <serlvet> 中加入 <init-param> 这个参数,否则的话 spring 会默认查找 web-inf/classes 下面以 [servlet-name]-servlet.xml 的文件,会出现以下错误:

ERROR: org.springframework.web.servlet.DispatcherServlet - Context initialization failed org.springframework.beans.factory.BeanDefinitionStoreException: IOException parsing XML document from ServletContext resource [/WEB-INF/SpringMVC DispatchServlet-servlet.xml]; nested exception is java.io.FileNotFoundException: Could not open ServletContext resource [/WEB-INF/SpringMVC DispatchServlet-servlet.xml]

HelloWordController.java:

```
package org.lxh.mvc.controller;


import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.ModelAndView;

@Controller
@RequestMapping("/test")
public class HelloWordController {
    private Logger logger = LoggerFactory.getLogger(HelloWordController.class);

    @RequestMapping("/hello")
    public void Hello() {
        logger.info("The hello() method is use");
    }

    @RequestMapping("/welcome")
    public String welcome() {
        logger.info("The welcome() method is use ");
//        return "/welcome";
         return "/WEB-INF/jsp/welcome";
    }

    @RequestMapping("/helloWorld")
    public ModelAndView helloWorld() {
        logger.info("The helloWorld() method is use");
        ModelAndView view = new ModelAndView();
        // view.setViewName("/helloworld");
        // view.setViewName("/WEB-INF/JSP/helloworld");
        view.setViewName("/hello");
        return view;
```

```
    }
}
```

可以发现上面为什么有那么多注释,

先一个一个来说明,spring mvc 可以通过以下三种方式链接到视图解析器中,每种方法以 requestMapping 注解作为跳转.

第一种,使用无返回方法跳转,如果使用返回方法进行跳转的话,则会通过视图解析器进行以 prefix(前缀)+方法名+suffix(后缀)组成的页面文件名称.这样的话可能需要用 url writer 框架进行重向地址更改,保证安全性.

第二种,使用一个返回的字符串方法作为跳转,使用字符串跳转的话好处就是在 return 的时候可以自己指定返回的名字,JSP 组成是 prefix(前缀)+返回的字符串+suffix(后缀),这样的话跳转的时候外围广很多而且,安全性相对高点

第三种,返回一个 ModelAndView 类型,ModelAndView 是 spring2.5经常使用的方式,使用 setViewName 方法则可以跳转到指定的页面.

在类中还还发现一个 Contorller 的注解,该注解是用于把当前 Java 类变成一个 Spring 里面的一个 beans,如果在 Class 上面加入 requestMapping 的话,而在方法上还拥有 requestMapping,则在浏览器输入的地址为 http://访问地址:端口/contextroot/要调用类的 requestMapping 跳转值/该类中存在的 requestMapping/,而在 Class 上标记了 requestMapping 的话,则需要在指定的位置创建一个文件夹作为存放.

例如:

输入 http://localhost:8080/SpringMVC_10000/test/hello 的话则会查找根目录下面的 test 文件夹下面的 hello.jsp.组成是 prefix(前缀)+/test/+方法名+suffix(后缀).目录结构如图2:

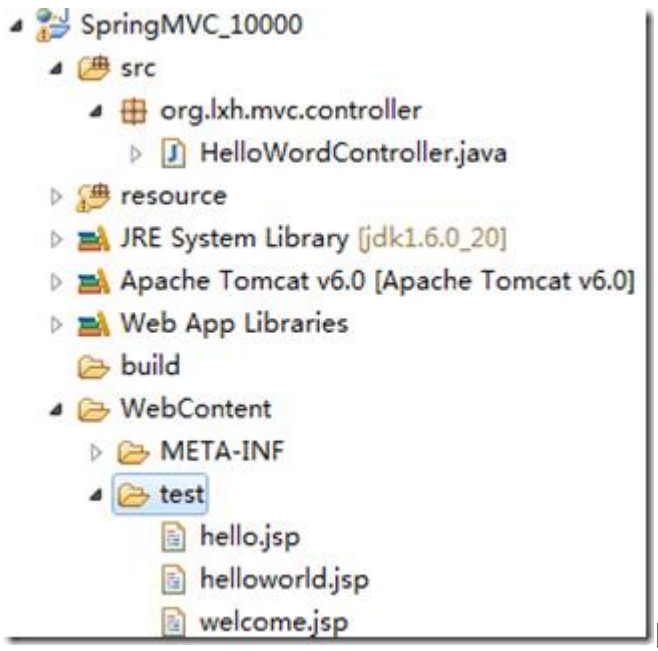


图2:跳转目录

[学习笔记]基于注解的 spring3.0.x MVC 学习笔记(三)

这篇不会大量的张贴代码,毕竟是自己对 springMVC 的学习,而不是对某一种东西作为详细解析的,好了话不多说了,近期学习 springMVC 换了不少东西,连日志工具也换了,采用了 slf4j+logback 进行日志管理,至于好处,请自行 Google

如何把主流的 log4j+commons-loggin 更换为 slf4j+logback 呢,由于 spring 里面采用了 commons-logging 日志监控,所以我们需要准备以下几个包:

logback-classic.jar:改善了 log4j 并且实现了 slf4j-log4j 的代码

logback-core.jar:logback 的核心代码

slf4j-api.jar:slf4j 实现代码

jcl-over-slf4j.jar:代替 commons-loggin 类

log4j-over-slf4j.jar(可选):替换 log4j。原有的 log4j.properites 将失效，需要转换为 logback.xml

jul-to-slf4j.jar:(可选)替换 jdk logging。需要在程序开始时调用 SLF4JBridgeHandler.install()来注册 listener。

LogBack.xml 配置文件跟 Log4j.xml 的配置文件非常相似:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <appender class="ch.qos.logback.core.ConsoleAppender" name="STDOUT">
```

```
    <encoder>
        <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{100} - %msg%n
        </pattern>
    </encoder>
</appender>
<logger name="org.lxh.mvc.controller" level="INFO" />

<!-- 3rdparty Loggers -->
<logger name="org.springframework.core">
    <level value="info" />
</logger>

<logger name="org.springframework.beans">
    <level value="info" />
</logger>

<logger name="org.springframework.context">
    <level value="info" />
</logger>

<logger name="org.springframework.web">
    <level value="info" />
</logger>

<root>
    <level value="debug" />
    <appender-ref ref="STDOUT" />
</root>
</configuration>
```

logger{100} :是限制当前日志显示内容的长度,最大为100个字符

如何使用 logback+slf4j?

参考代码:

```
private Logger logger = LoggerFactory.getLogger(RequestMappingStudy1.class);
```

同以前使用 log4j+commons-logging 是一样的方式,只是换了使用 slf4j 的工厂类而已

用到 spring 配置如下:

```
  <!-- 对 spring org.lxh 包下所有注解扫描 -->
    <context:component-scan base-package="org.lxh"></context:component-scan>
    <!-- 支持 spring mvc 新的注解类型 详细 spring3.0手册 15.12.1 mvc:annotation-driven-->
    <mvc:annotation-driven />


    <!--  对模型视图名称的解析，即在模型视图名称添加前后缀,在 requestmapping 输入的地址后自动调用该类进行视图解析-->
    <bean id="viewResolver"
        class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="viewClass"
            value="org.springframework.web.servlet.view.JstlView"></property>
        <property name="prefix" value="/" />
        <property name="suffix" value=".jsp"></property>
    </bean>
```

这次开始主要对 spring 的每个注解用法进行详细的介绍,

@RequestMapping 的参数如下

```
/**
* @see RequestMapping 参数
*
* @param value
*           需要跳转的地址
* @param mehtod
*           基于 RestFul 的跳转参数,有 RequestMethod.get post,put 等
* @param params
*           符合某个参数的时候才调用该方法
* @param headers
*           符合头信息的时候才调用
* */
```

SpringMVC 中大部分请求都是由 RequestMapping 提交的,而且提交的类型有很多种,以3.0来讲一般的请求方式有以下几种

第一种:以无参的形式返回:

```
/**
 * 无参数返回的是根据 prefix 前缀+@RequestMapping value +suffix 后缀组成
 *
 * */
 @RequestMapping("/novoid")
 public void novoid() {
     logger.info(this.getClass().getSimpleName() + "novoid 方法被调用");
 }
```

返回的地址是 http://访问地址/项目名称/spring 配置文件 bean 为 viewResolver 的 prefix 的值+requestMapping 返回的值+suffix 的值

第二种:返回一个 String 类型:

```
/**
    * 根据 String 字符串返回对应的地址 prefix 前缀+返回值+suffix 后缀组成
    * */
    @RequestMapping("/string")
    public String string() {
        logger.info("String 方法调用");
        return "WEB-INF/jsp/success";
    }
```

第三种:返回一个 ModelAndView 对象

```
/**
* spring2.5的方法,返回一个 ModelAndView 对象,通过 setViewName 方法跳转到指定的页面 调用 addObject
* 相当于调用 request.setAttribute 方法
* */
@RequestMapping("/modelview")
public ModelAndView view(Model model) {
    logger.info("view 方法调用");
    ModelAndView andView = new ModelAndView();
    andView.setViewName("WEB-INF/jsp/success");
    return andView;
}
```

第四种返回一个 Map 集合

```
/**
    * @see 使用 map 作为返回值的时候 是以 prefix 前缀+requestMapping 的 value+suffix 后缀组成 返回一个 map
    *      ,map 的 put 方法调用相当于 request.setAttribute 方法
    * */
    @RequestMapping("/mapa")
    public Map<String, Object> mapa(ModelMap map1) {

        Map<String, Object> map = new HashMap<String, Object>();
        UserBean bean = new UserBean();
        bean.setId(1);
        bean.setUsername("Edward Lau");
        bean.setPassword("edward");
        map.put("hello", "world key");
        map.put("user", bean);
        return map;
    }
```

使用第四种方法,可以在页面中通过调用 JSTL 进行取值,如下面 jsp 代码

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Mapa</title>
</head>
<body>
Map a
姓名:${user.username }
密码:${user.password }
hello:${hello }
</body>
</html>
```

第五种返回一个 ModelMap 类型:

```
/**
     * @see 返回一个 ModelMap 类型,返回地址根据以 prefix 前缀+requestMapping 的 value+suffix 后缀组成
     *      ModelMap 本身也拥有 hashmap 的方法,也可以使用 addAllAttributes 对一个 map 添加到 attribute 里面
     * */
    @RequestMapping("/map")
    public ModelMap map() {
        ModelMap map = new ModelMap();
        map.addAttribute("aa", "bb");
        map.addAllAttributes(temp());
        return map;
    }

    /**
     *@see 临时类
     *@return 返回一个 map 类型
     * */
    public Map<String, UserBean> temp() {
        Map<String, UserBean> map1 = new HashMap<String, UserBean>();
        UserBean bean = new UserBean();
```

```
        bean.setId(1);
        bean.setUsername("Edward Lau");
        bean.setPassword("edward");
        map1.put("user", bean);
        // map1.put("hello", "world key");
        UserBean bean1 = new UserBean();
        bean1.setId(2);
        bean1.setUsername("Edward Lau2");
        bean1.setPassword("edward");
        map1.put("user1", bean1);
        System.out.println(map1);
        return map1;
    }
```

使用 ModelMap 可以把一个多个集合存到一个属性中,可以直接在页面调用 EL 语言进行读取,jsp 代码如下:

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
   pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Mapa</title>
</head>
<body>
Map a
姓名:${user.username }
密码:${user.password }
hello:${hello }
aa:${aa }
</body>
</html>
```

[学习笔记]基于注解的 spring3.0.x MVC 学习笔记(四)

好久没有更新关于 spring3.0.x 的学习笔记了,这笔记除了平时工作上用的代码之外还有部分代码是根据我的老师学习下来后我再重新理解所整理的,接着上期第三部分的内容:

上次写到了 requestMapping 对返回集合的用法,使用集合返回的话则需要根据 requestmapping 中 values 的值进行命名 jsp 页面,而整个路径地址是根据视图模型的返回地址来确定.这次主要介绍的是返回类型为 Model,List.Collection,Set,Object,先看返回类型为 Model,代码如下:

```
1/** 2 * @see 返回一个 Model 接口.通过创建 ExtendedModelMap 可以使用,方法同 modelMap 一样  3 * */ 4 @RequestMapping("/model")  5 public Model model() {  6 Model model = new ExtendedModelMap();  7 model.addAllAttributes(temp());  8 return model;  910 }
```

temp 方法代码如下:

```
 1:  /**
 2:       * @see 临时类
 3:       * @return 返回一个 map 类型
 4:       * */
 5:      public Map<String, UserBean> temp() {
 6:          Map<String, UserBean> map1 = new HashMap<String, UserBean>();
 7:          UserBean bean = new UserBean();
 8:          bean.setId(1);
 9:          bean.setUsername("Edward Lau");
10:          bean.setPassword("edward");
11:          map1.put("user", bean);
```

```
12:         // map1.put("hello", "world key");
13:         UserBean bean1 = new UserBean();
14:         bean1.setId(2);
15:         bean1.setUsername("Edward Lau2");
16:         bean1.setPassword("edward");
17:         map1.put("user1", bean1);
18:         System.out.println(map1);
19:         return map1;
20:     }
```

通过 model 的 addAllAttributes 可以把一个 map 集合的内容放到 model 对象中,通过 jsp 中读取指定的数据,相当于读取 request.getAttributes 一样

jsp 代码如下图:

```
<title>/model例子</title>
</head>
<body>

姓名:${user1.username }
密码:${user1.password }
hello:${hello }
aa:${aa }
</body>
```

效果如下图:

姓名:Edward Lau2 密码:edward hello: aa:

对于 List,Collection,Set 这3种类型返回的方式都是根据 requestmapping 的 value 标记进行读取指定页面,但是在 jsp 页面中读取方式值的方式相对有些不同,首先是对 List:

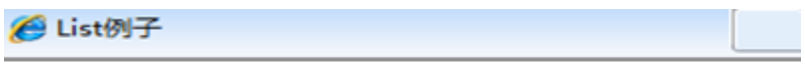
对于 List 代码只是把返回类型改为了 List 之外,在 jsp 中我们需要根据代码第一个 add 的数据类型进行保存,代码如下:

```
1:  /**
2:          * @see 返回集合类型的默认名称 默认值为: 第一个存入对象的
Object.getClass().getSimpleName()+List 组成
3:       * @see 返回 list 集合的话可以使用 request.getAttribute(默认值)取出值,或者使
用 el 表达式${默认值}
4:       *       <b>如果需要取某一个序列的话可以使用${默认值[下标]}</b>
5:       * */
6:      @RequestMapping("/list")
7:      public List listReturn() {
8:          List toys = new ArrayList();
9:          toys.add("gg5555555");
10:         toys.add(new UserBean(1, "a", "b"));
11:         toys.add(new UserBean(2, "PSP", "2100"));
12:
13:         return toys;
14:     }
```

对应的 jsp 如下图:

```
<%@ page language="java" contentType="text/html;
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Tr
<html>
<head>
<meta http-equiv="Content-Type" content="text/h
<title>List例子</title>
</head>
<body>
1:${stringList[1].username }
<br/>
2:${stringList[0] }
</body>
</html>
```

其中1显示的是集合第1个元素的内容,从代码上看下标元素为1的对象是 UserBean,但是在前台调用 request.getAttribute 方法是根据 list 第一个 add 元素的 clasas 类型的 simplename+上集合类型名字组成的,效果如下图:



对于 Set 来说,由于 Set 也是属于集合的一种所以读取方式也是根据 List 一样.

对于 Collection 他则有点特别他是根据返回所属 Collection 的类型来决定的,代码如下:

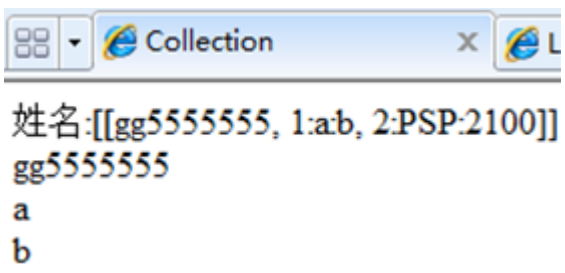
```
1:  @RequestMapping("/collection")
2:      public Collection collection() {
3:          logger.info("collection is runing");
4:  //        return temp().values();
5:          List toys = new ArrayList();
6:          toys.add("gg5555555");
7:          toys.add(new UserBean(1, "a", "b"));
8:          toys.add(new UserBean(2, "PSP", "2100"));
9:
10:         return Collections.singletonList(toys);
11:     }
```

如果返回的集合类型是 list 的话该 Collection 而没有定义泛型的话则以返回的 list 的实例对象作为命名的 model type 则 arrayList+List,对应的 jsp 如下:

```
1:  <%@ page language="java" contentType="text/html; charset=UTF-8"
2:      pageEncoding="UTF-8"%>
3:     <!DOCTYPE  html  PUBLIC  "-//W3C//DTD  HTML  4.01  Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
4:  <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
5:
6:  <%@page import="java.util.List"%><html>
7:  <head>
8:  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
9:  <title>Collection</title>
10: </head>
11: <body>
12:
13: 姓名:${arrayListList}
14: <c:forEach var="user" items="${arrayListList}"><br>
15: ${user[0] }<br>
16: ${user[1].username }<br>
17: ${user[1].password }<br>
18: </c:forEach>
19:
20:
21: </body>
```

```
22:  </html>
```

这样可以得到结果如下图:



如果返回类型是 List 并且有定义泛型的话则以泛型名字+List 进行返回,Set 集合效果同 List 一样.

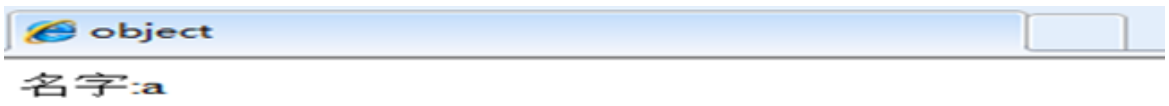
对于返回 Object 类型的话则按照创新的对象类型返回指定的 model type 代码如下:

```
1:  /**
2:       * @see 返回对象在页面中以${返回对象名称}取值,返回
3:       * @return Object 返回对象的类型,如返回 Userbean 则输入返回的对象为 Userbean
4:       * */
5:      @RequestMapping("/object")
6:      public Object object() {
7:          return new UserBean(1, "a", "b");
8:      }
```

对应的 jsp 与效果如下图:

```
<%@ page language="java" contentType="text/html; c
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Trans
<html>
<head>
<meta http-equiv="Content-Type" content="text/html
<title>object</title>
</head>
<body>

名字:${userBean.username }
</body>
</html>
```



[学习笔记]基于注解的 spring3.0.x MVC 学习笔记(五)

在通过 requestmapping 中返回中,我们可以通过 forward 还有 redirect 方式进行页面跳转而避开视图模型的控制,这次先讲 forward 的功能,forward 有 2 种表示模式,**forward 功能的功能同 request 中的 request.getRequestDispatcher("目标地址").forward(request,response)功能是一样的(spring mvc 默认这种)**先看第一种:代码如下:

```
1:  @RequestMapping("/forward")
2:      public String testForword(ModelMap map){
3:  //        Map<String,Object> map = new HashMap<String, Object>();
4:          map.put("testdata", "hello world!");
5:          return "forward:forwardlist.jsp";
6:      }
```

通过 forward 我们可以访问特定的页面,而不需要经过视图模型进行监控返回页面一定是 jsp 的页面,如下代码示例:

```
1:  @RequestMapping("/forward.do")
2:      public String testForword(ModelMap map){
3:  //        Map<String,Object> map = new HashMap<String, Object>();
4:          map.put("testdata", "hello world!");
5:          return "forward:forwardlist.html";
6:      }
```

两个对比一下可以发现上面跟下面的 requestmapping 方法跳转是否有些不同,第二种跳转的时候多了个.do,原因是我把 web.xml 中 Spring 的 servlet-mapping 的 <url-pattern></url-pattern>, 改写成了 <url-pattern>*.do</url-pattern> 让方法只对 *.do 的跳转有效,改成这个原因很简单如果 **<url-pattern>/</url-pattern>使用这种的话对所有/后面所有链接都进行拦截,但是不会拦截视图模型下对应的后缀名称**.我的视图模型对应的配置文件如下:

```
 1:  <bean id="viewResolver"
 2:
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
 3:          <property name="viewClass"
 4:
value="org.springframework.web.servlet.view.JstlView"></property>
 5:          <property name="prefix" value="/" />
 6:          <property name="suffix" value=".jsp"></property>
 7:      </bean>
```

spring 会自动过滤视图模型中的 suffix 对应的后缀,可以使第一个例子正常访问.

第二种的 forward 方式是可以跳转到指定的另外一个方法体中,代码如下:

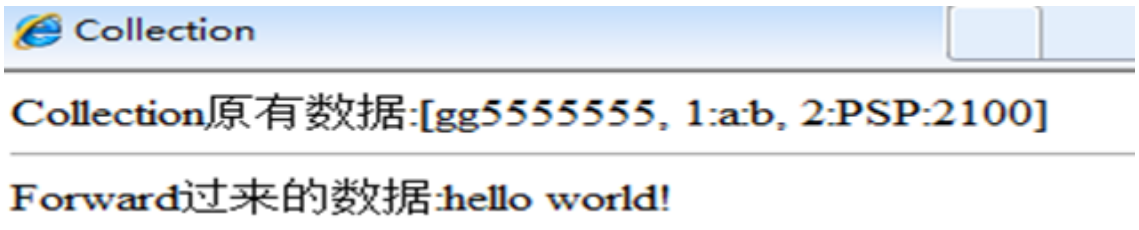
forward 的代码:

```
 1:  @RequestMapping("/forward")
 2:      public String testForword(ModelMap map){
 3:  //        Map<String,Object> map = new HashMap<String, Object>();
 4:          map.put("testdata", "hello world!");
 5:          return "forward:collection1";
 6:      }
```

collection 代码:

```
 1:  @RequestMapping("/collection1")
 2:      public Collection collection1() {
 3:          logger.info("collection is runing1");
 4:          List toys = new ArrayList();
 5:          toys.add("gg5555555");
 6:          toys.add(new UserBean(1, "a", "b"));
 7:          toys.add(new UserBean(2, "PSP", "2100"));
 8:
 9:          return toys;
10:      }
```

最后显示的效果:

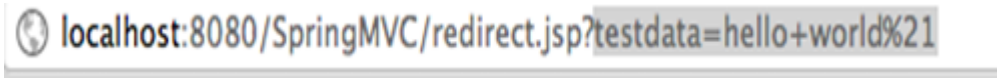


[学习笔记]基于注解的 spring3.0.x MVC 学习笔记(六)

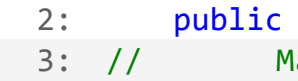
近期心情不太好，想起当初有些事情，继续 springmvc 的返回类型中带有 redirect 方式的使用，使用 redirect 方式返回跟有3种方式首先介绍第一种，代码如下：

```
 1:  @RequestMapping("/redirect")
 2:      public String testRedirect(ModelMap map){
 3:  //        Map<String,Object> map = new HashMap<String, Object>();
 4:          map.put("testdata", "hello world!");
 5:          return "redirect:redirect.jsp";
 6:      }
```

使用 redirect 后他不会对 modelmap 中的数据进行调用 request.setAttribute 而采用带参数的形式进行传值,如下图:



再页面中调用 request.getParameter("testdata")后得到以下结果:



jsp 中的代码如下:

```
 1:  <%@ page language="java" contentType="text/html; charset=GB18030"
 2:      pageEncoding="GB18030"%>
 3:     <!DOCTYPE  html  PUBLIC  "-//W3C//DTD  HTML  4.01  Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
 4:  <html>
 5:  <head>
 6:  <meta http-equiv="Content-Type" content="text/html; charset=GB18030">
 7:  <title>跳转过来的 jsp</title>
 8:  </head>
 9:  <body>
10:  ${testdata}
11:  :
12:  <%=request.getParameter("testdata")+"----" %>
13:  </body>
14:  </html>
```

事实证明使用了 redirect 后就变成了带参传值.

另外一种则是可以返回一个链接代码如下:

片段代码1:

```
 1:  @RequestMapping(value="/redirect",method=RequestMethod.GET)
 2:      public String testRedirect(ModelMap map){
 3:  //        Map<String,Object> map = new HashMap<String, Object>();
 4:          map.put("testdata", "redirect:hello world!");
 5:          return "redirect:forward";
 6:      }
```

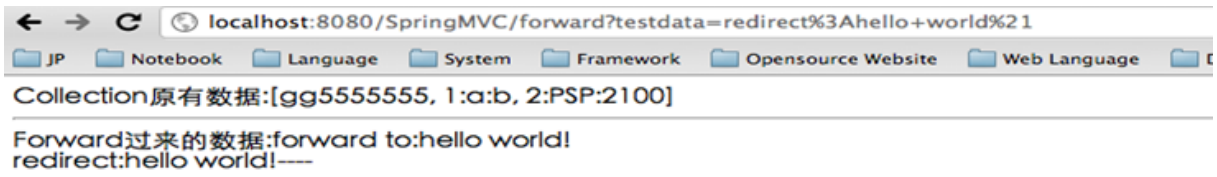
片段代码2:

```
 1:  @RequestMapping("/forward")
 2:      public String testForword(ModelMap map){
 3:  //        Map<String,Object> map = new HashMap<String, Object>();
 4:          map.put("testdata", "forward to:hello world!");
 5:          return "forward:collection1";
 6:      }
```

片段代码3

```
 1:  @RequestMapping("/collection1")
 2:      public Collection collection1() {
 3:          logger.info("collection is runing1");
 4:          List toys = new ArrayList();
 5:          toys.add("gg5555555");
 6:          toys.add(new UserBean(1, "a", "b"));
 7:          toys.add(new UserBean(2, "PSP", "2100"));
 8:
 9:          return toys;
10:      }
```

在浏览器上运行/redirect 得到结果如下:



很明显运行/redirect 跳转到 forward 中然后 forward 中又引用了 collection1所以才变成这样.

最终一种则是跳转到外部链接:代码如下:

```
1:  @RequestMapping(value="/redirect",method=RequestMethod.GET)
2:      public String testRedirect(ModelMap map){
3:  //        Map<String,Object> map = new HashMap<String, Object>();
4:          map.put("testdata", "redirect:hello world!");
5:          return "redirect:http://blog.163.com/edwardlauxh";
6:      }
```

可以直接跳转到外部链接中,跳转到外部链接,在 modelmap 中所保存的值也会一直带过去所对应的网站.

[学习笔记]基于注解的 spring3.0.x MVC 学习笔记(七)

本次介绍的是@ModelAttribute 注解，这个注解可以跟@SessionAttributes 配合在一起用，或者单独使用，首先先介绍@ModelAttribute 注解，他跟 mvc 中的 modelMap 很相似，可以把它当作一个 modelmap 的简易版，使用@ModelAttribute 方法很简单只是@ModelAttribute("保存的名字")就可以了,他只支持以下几种方式:（如果有还有其他方式请留言，谢谢）

@ModelAttribute

第一种：在返回值中使用：

代码如下：

```
1:  @RequestMapping("get")
2:      public @ModelAttribute("user") UserBean getUser(){
3:          UserBean bean = new UserBean();
4:          bean.setId(2);
5:          bean.setUsername("EdwardLau1");
6:          bean.setPassword("edward1");
7:
8:
9:
logger.debug("userbean-------------------------"+bean.getId()+":"+bean.getUsern
ame());
10:         //map.put("user", bean);
11:         return bean;
12:     }
```

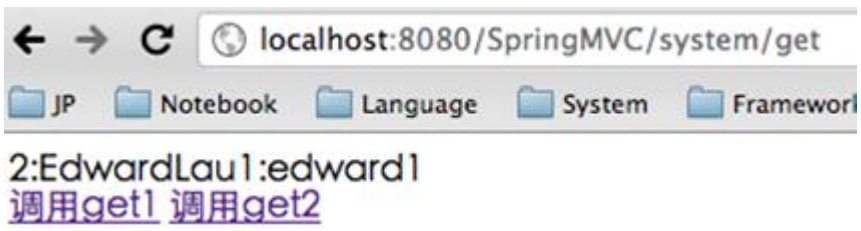
平常我们都是使用 ModelMap 或者 ModelAndView 进行保存值传到前台，当如果你需要保存值比较少的时候可以采用这种方式进行保存值并且保存到前台显示。

JSP 代码如下：

```
1:  <%@ page language="java" contentType="text/html; charset=GB18030"
2:      pageEncoding="GB18030"%>
3:     <!DOCTYPE  html  PUBLIC  "-//W3C//DTD  HTML  4.01  Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
4:  <html>
5:  <head>
6:  <meta http-equiv="Content-Type" content="text/html; charset=GB18030">
7:  <title>调用 get 方法</title>
8:  </head>
9:  <body>
10: ${user}
11: <br/>
```

```
12:  <a href="get1">调用 get1</a>
13:  <a href="get2">调用 get2</a>
14:  </body>
15:  </html>
```

如图所示：



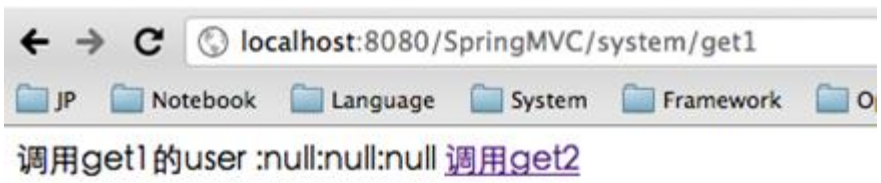
在 userbean 中我重写了 toString（）方法所以会显示上述内容。

注意：

ModelAttribute 的有效范围，相当于 HttpServletRequest 中的 request.setAttribute 一样,如果不配合 @SessionAttributes 注解的话则不能全局读取的,如首先运行上述代码然后运行如下代码：

```
1:  @RequestMapping("get1")
2:      public  UserBean getUser1(@ModelAttribute("user") UserBean bean,ModelMap
map){
3:
4:
logger.debug("userbean1------------------------"+bean.getId()+":"+bean.getUser
name());
5:          //map.put("user", bean);
6:          return bean;
7:      }
```

点击上图调用 get1链接看到如下图结果



事实证明 **ModelAttribute 只能保存当前值,传到另外一个地方后没有对获取的值进行再次保存,直接再调用获取之前的值会获取不到,变成 null.**

而**调用 ModelAttribute 注解的话可以放在方法体上面,或者传参的值里面(定义类型需要跟 ModelAttribute 的保存的值类型一致,因为 ModelAttribute 会自动转成该参数的)**

[学习笔记]基于注解的 spring3.0.x MVC 学习笔记(八)

原本打算将@ModelAttribute 跟@SessionAttributes 一起写的发现有些例子不可以重复使用，所以决定再开一篇文章写

顾名思义 SessionAttributes 就是保存 session 的，SessionAttributes 使用方法很简单，@SessionAttributes 允许指定多个属性。你可以通过字符串数组的方式指定多个属性，如 @SessionAttributes({“attr1”,”attr2”})。此外，@SessionAttributes 还可以通过属性类型指定要 session 化的 ModelMap 属性，如 @SessionAttributes(types = User.class)，当然也可以指定多个类，如 @SessionAttributes(types = {User.class,Dept.class})，还可以联合使用属性名和属性类型指定：@SessionAttributes(types = {User.class,Dept.class},value={“attr1”,”attr2”})。

以下例子使用：@SessionAttribute 必须配合@ModelAttribute 一起使用，

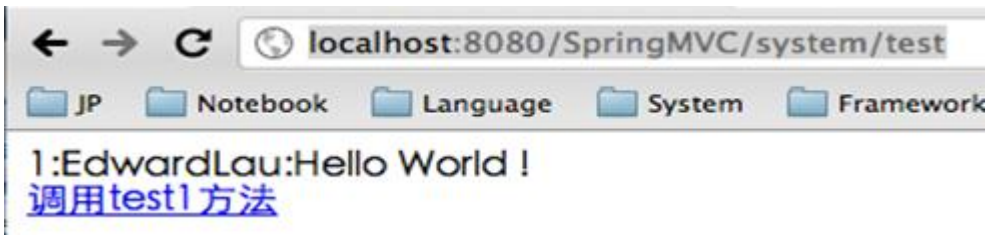
第一种方式，采用@SessionAttributes(“变量名”)

代码如下

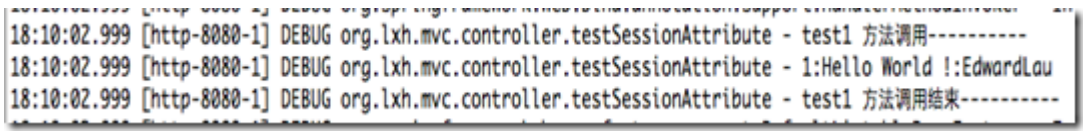
```
1:  @Controller
2:  @SessionAttributes("user")
3:  public class testSessionAttribute {
4:
```

```
 5:      private Logger logger = LoggerFactory.getLogger(testSessionAttribute.class);
 6:
 7:      @RequestMapping("test")
 8:      public String test(ModelMap map){
 9:          UserBean userBean = new UserBean();
10:          userBean.setId(1);
11:          userBean.setUsername("EdwardLau");
12:          userBean.setPassword("Hello World !");
13:          map.put("user", userBean);
14:          return "sessionattribute";
15:      }
16:
17:
18:      @RequestMapping("test1")
19:      public String test(@ModelAttribute("user") UserBean userBean){
20:          logger.debug("test1 方法调用----------");
21:
logger.debug(userBean.getId()+":"+userBean.getPassword()+":"+userBean.getUsername());
22:          logger.debug("test1 方法调用结束----------");
23:          return "users";
24:      }
25:
26:  }
```

在浏览器运行 http://localhost:8080/SpringMVC/system/test 得到如下图所示：



点击调用 test1方法后，看后台如下：

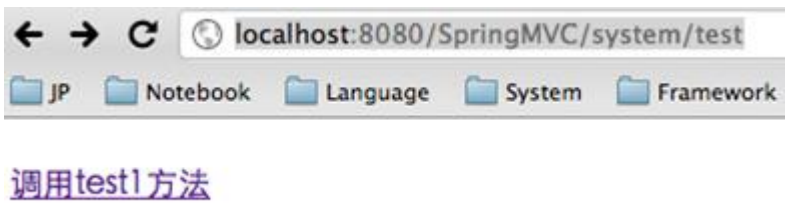


通过注解@ModelAttribute 可以获得之前保存的值。

第二种方式，采用@SessionAttributes(types={指定 class})

采用这种方式的话系统会默认保存指定 class 的名字（头字母小写），如 UserBean.class，modelmap 会仅仅会保存名字为 userBean 或者 userbean 的对应的 class，其余则不会保存，代码如下：

当在在页面上输入 http://localhost:8080/SpringMVC/system/test 时候，会把对应的内容保存到 modelmap 中，但后点击如下图，调用 test1方法的时候，



得到如下图结果：

事实证明：调用 sessionAttriubte 只适用 types 参数的时候只是，默认保存以类名为名字的值，其中类名第一个字母为小写，而他只是能保存当前对象的值，而对象中存在了那个外一个对象的时候则不能保存

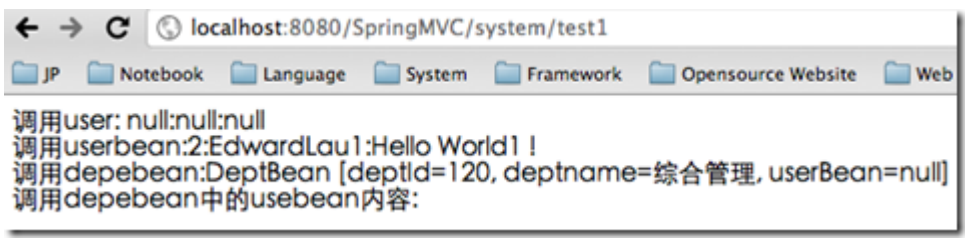
第三种：采用@SessionAttributes(value={指定名称},types={指定 class})

使用这种方法则对指定的类，进行特定名称保存，一对一的形式进行保存，代码如下：

```
 1:  @Controller
 2:  @SessionAttributes(value={"test","dept"}, types={UserBean.class,DeptBean.class})
 3:  public class testSessionAttribute {
 4:
 5:      private Logger logger = LoggerFactory.getLogger(testSessionAttribute.class);
 6:
 7:      @RequestMapping("test")
 8:      public String test(ModelMap map){
 9:          UserBean userBean = new UserBean();
10:          userBean.setId(1);
11:          userBean.setUsername("EdwardLau");
12:          userBean.setPassword("Hello World !");
13:          map.put("aaa", "aaa");
14:          map.put("test", userBean);
15:          UserBean userBean1 = new UserBean();
16:          userBean.setId(2);
17:          userBean.setUsername("EdwardLau1");
18:          userBean.setPassword("Hello World1 !");
19:          map.put("user", userBean1);
20:          DeptBean deptBean = new DeptBean();
21:          deptBean.setDeptId(120);
22:          deptBean.setDeptname("综合管理");
23:          deptBean.setUserBean(userBean1);
24:          map.put("dept", deptBean);
25:          return "sessionattribute";
26:      }
27:
28:
29:      @RequestMapping("test1")
30:      public String test(@ModelAttribute("test") UserBean userBean, @ModelAttribute("aaa") String
aaa
31:                         ,@ModelAttribute("user")   UserBean  bean,@ModelAttribute("dept")  DeptBean
deptBean){
32:          logger.debug("test1 方法调用----------");
33:
logger.debug(userBean.getId()+":"+userBean.getPassword()+":"+userBean.getUsername());
34:
logger.debug("aaaa:"+aaa+"bean:"+bean.getId()+":"+bean.getUsername()+":"+bean.getPassword())
;
35:
logger.debug("deptBean:"+deptBean.getDeptId()+":"+deptBean.getUserBean().getUsername()+":"+d
eptBean.getDeptname());
36:          logger.debug("test1 方法调用结束----------");
37:          return "testsession";
38:      }
```

```
39:
40:  }
```

同样运行第二种方式的地址，得到的结果如下图：



采用这种方式只是把要保存的类定义到固定的名字而已，用法与第二种一样

[学习笔记]基于注解的 spring3.0.x MVC 学习笔记(九)

本章节，仅为@SessionAttributes 的功能扩展介绍介绍，结合@requestparam 注解进行简易无数据库分页。

之前介绍了@SessionAttributes 的功能，他可以保存指定的值在 modelnap 中，不会因为 request 提交后则消失，我们利用这个特性，可以利用 SessionAttributes 注解进行简易的分页。

代码如下：

```
 1:  public class  testservice<T>{
 2:
 3:      public List<UserBean> list ;
 4:
 5:      public Page<T> getuserlist(int page,int size){
 6:          List<UserBean> temp = new ArrayList<UserBean>();
 7:
 8:                      for  (int  i  =  Page.getStartOfPage(page,  size);  i  <
Page.getLastOfPage(page, size); i++) {
 9:              temp.add(this.list.get(i));
10:          }
11:          return (Page<T>) new Page<T>(Page.getStartOfPage(page, size), size,
list.size(), (List<T>) temp);
12:      }
13:
14:      public int getsize(){
15:          return this.list.size();
16:      }
17:
18:      public testservice(){
19:          list =new ArrayList<UserBean>();
20:          for (int i = 0; i < 20; i++) {
21:              UserBean bean = new UserBean();
22:              bean.setId(i);
23:              bean.setUsername(String.valueOf(i));
24:              bean.setPassword(String.valueOf(i));
25:              this.list.add(bean);
26:          }
27:      }
28:  }
```

由于采取了不带数据库进行分页，此类作为仅作假设从数据库取回数据用。

Page 类是属于个人封装分页类，支持 spring struts2 jstl 标签（还在优化阶段中暂不公开）

controller 代码如下：

```
 1:  @Controller
 2:  @SessionAttributes(value={"pagesize","number"})
 3:  @RequestMapping("/system")
 4:  public class testSessionAttribute {
 5:
```

```
  6:                                          private      Logger      logger      =
LoggerFactory.getLogger(testSessionAttribute.class);
  7:
  8:      @RequestMapping("listindex")
  9:      public String test(ModelMap map ,@RequestParam(defaultValue="10") Integer
pagesize,@RequestParam(defaultValue="1") Integer number){
 10:
 11:          Page<UserBean> users = new testservice<UserBean>().getuserlist(number,
pagesize);
 12:          /**
 13:           * 如果获取的数据集为空与当前页不为1的话,如-1 则调用
 14:           * */
 15:          if(users.getResultSize()==0&&users.getPageNumber()!=1){
 16:              number = users.getLastPage();
 17:                     users  =  new  testservice<UserBean>().getuserlist(number,
pagesize);
 18:          }
 19:          map.put("number", number);
 20:          map.put("pagesize", pagesize);
 21:          map.put("page", users);
 22:          return "pagelist";
 23:      }
 24:
 25:
 26:  }
```

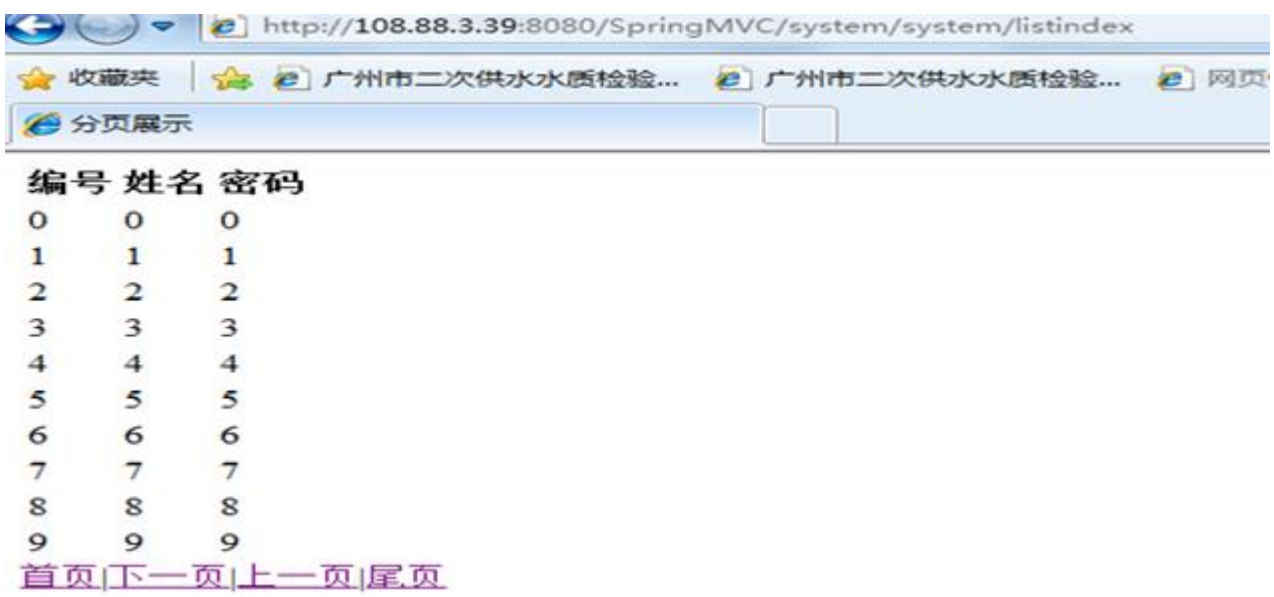
代码说明：利用@SessionAttributes 分别定义了每页显示数据 pagesize 与下一页页码 nubmer，这样可以保证在其他地方可以得到他当前页与每页显示数。

jsp 如下：

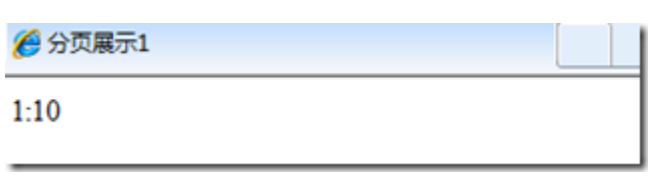
```
  1:  <form action="list" >
  2:
  3:  <table>
  4:      <thead>
  5:          <tr>
  6:              <th>编号</th>
  7:              <th>姓名</th>
  8:              <th>密码</th>
  9:          </tr>
 10:      </thead>
 11:      <tbody>
 12:      <c:forEach items="${page }" var="data">
 13:          <tr>
 14:
<td>${data.id }</td><td>${data.username }</td><td>${data.password }</td>
 15:          </tr>
 16:      </c:forEach>
 17:      </tbody>
 18:  </table>
 19:       <a    href="listindex?number=${page.firstPage    }">   首   页   </a>|<a
href="listindex?number=${page.nextPage      }">     下     一     页     </a>|<a
```

```
href="listindex?number=${page.previousPage     }">     上     一     页     </a>|<a
href="listindex?number=${page.lastPage }">尾页</a>
 20:  </form>
```

效果如下图：



使用了 SessionAttributes 后可以在其他页面调用得到所需要的值，如下图



总结：

我们使用@SessionAttributes 可以将常用的东西参数封装到 ModelMap 中，这样可以高效利用，如分页，配置文件名称等。。