

目录

SpringMvc 教程.....	1
初识 springMVC.....	1
背景.....	1
常见 MVC 框架比较.....	2
基于 spring3.2 的 采用 annotation 方式搭建 springMVC 环境.....	2
springMVC 的 RequestMapping 的基本设置.....	7
在 controller 中获取前台传递的参数.....	9
在 controller 中获取 web 元素.....	10
将 controller 中数据传递到 jsp 页面.....	11
设置跳转方式为重定向或者转发.....	15
视图解析器的配置和使用.....	15
controller 中方法的返回值类型.....	16
springMVC 的文件上传于下载.....	19
springMVC 和 jQuery 的 Ajax 结合.....	23

SpringMvc 教程

作者：DK

初识 springMVC

背景

Spring 框架提供了构建 Web 应用程序的全功能 MVC 模块。使用 Spring 可插入的 MVC 架构，可以选择是使用内置的 Spring Web 框架还是 Struts 这样的 Web 框架。通过策略接口，Spring 框架是高度可配置的，而且包含多种视图技术，例如 JavaServer Pages (JSP) 技术、Velocity、Tiles、iText 和 POI。Spring MVC 框架并不知道使用的视图，所以不会强迫您只使用 JSP 技术。Spring MVC 分离了控制器、模型对象、分派器以及处理程序对象的角色，这种分离让它们更容易进行定制。

常见 MVC 框架比较


运行性能上:

Jsp+servlet>struts1>spring mvc>struts2+freemarker>>struts2,ognl,值栈。

开发效率上,基本正好相反。值得强调的是, **spring mvc** 开发效率和 **struts2** 不相上下。

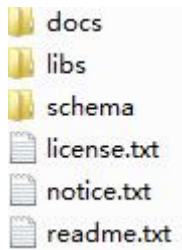
Struts2 的性能低的原因是因为 OGNL 和值栈造成的。所以,如果你的系统并发量高,可以使用 **freemaker** 进行显示,而不是采用 OGNL 和值栈。这样,在性能上会有相当大得提高。

基于 spring3.2 的 采用 annotation 方式搭建 springMVC 环境


























































- 1、 上官网下载对应的 zip 包  **spring-framework-3.2.0.RELEASE-dist.zip** 当然该 zip 并非最新的。

下载地址为: <http://repo.spring.io/webapp/home.html?0>

- 2、 解压之后得到目录:

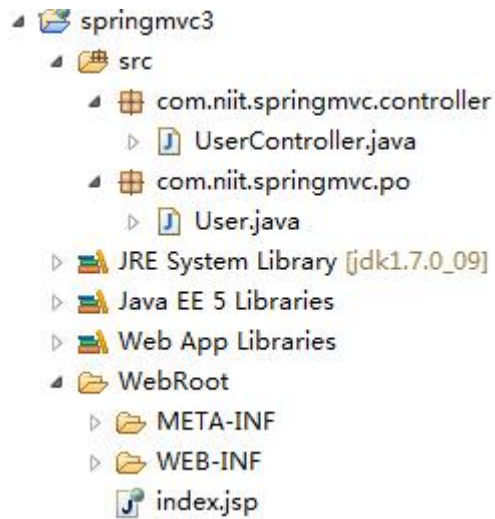


打开 libs 会看到会多 jar

 spring-aop-3.2.0.RELEASE.jar	 spring-aop-3.2.0.RELEASE-javad...
 spring-aop-3.2.0.RELEASE-sourc...	 spring-aspects-3.2.0.RELEASE.jar
 spring-aspects-3.2.0.RELEASE-jav...	 spring-aspects-3.2.0.RELEASE-so...
 spring-beans-3.2.0.RELEASE.jar	 spring-beans-3.2.0.RELEASE-java...
 spring-beans-3.2.0.RELEASE-sour...	 spring-context-3.2.0.RELEASE.jar
 spring-context-3.2.0.RELEASE-jav...	 spring-context-3.2.0.RELEASE-so...
 spring-context-support-3.2.0.REL...	 spring-context-support-3.2.0.REL...
 spring-context-support-3.2.0.REL...	 spring-core-3.2.0.RELEASE.jar
 spring-core-3.2.0.RELEASE-javad...	 spring-core-3.2.0.RELEASE-sourc...
 spring-expression-3.2.0.RELEASE...	 spring-expression-3.2.0.RELEASE...
 spring-expression-3.2.0.RELEASE...	 spring-instrument-3.2.0.RELEASE....
 spring-instrument-3.2.0.RELEASE-...	 spring-instrument-3.2.0.RELEASE-...
 spring-instrument-tomcat-3.2.0.R...	 spring-instrument-tomcat-3.2.0.R...
 spring-instrument-tomcat-3.2.0.R...	 spring-jdbc-3.2.0.RELEASE.jar
 spring-jdbc-3.2.0.RELEASE-javad...	 spring-jdbc-3.2.0.RELEASE-sourc...
 spring-jms-3.2.0.RELEASE.jar	 spring-jms-3.2.0.RELEASE-javado...
 spring-jms-3.2.0.RELEASE-source...	 spring-orm-3.2.0.RELEASE.jar
 spring-orm-3.2.0.RELEASE-javad...	 spring-orm-3.2.0.RELEASE-sourc...
 spring-oxm-3.2.0.RELEASE.jar	 spring-oxm-3.2.0.RELEASE-javad...
 spring-oxm-3.2.0.RELEASE-sourc...	 spring-struts-3.2.0.RELEASE.jar
 spring-struts-3.2.0.RELEASE-java...	 spring-struts-3.2.0.RELEASE-sour...
 spring-test-3.2.0.RELEASE.jar	 spring-test-3.2.0.RELEASE-javado...
 spring-test-3.2.0.RELEASE-source...	 spring-tx-3.2.0.RELEASE.jar
 spring-tx-3.2.0.RELEASE-javadoc.j...	 spring-tx-3.2.0.RELEASE-sources.j...
 spring-web-3.2.0.RELEASE.jar	 spring-web-3.2.0.RELEASE-javad...
 spring-web-3.2.0.RELEASE-sourc...	 spring-webmvc-3.2.0.RELEASE.jar
 spring-webmvc-3.2.0.RELEASE-ja...	 spring-webmvc-3.2.0.RELEASE-so...
 spring-webmvc-portlet-3.2.0.REL...	 spring-webmvc-portlet-3.2.0.REL...
 spring-webmvc-portlet-3.2.0.REL...	

这里边包括了所有的 jar 和 source 和 doc。当然我们只是需要使用 jar 就可以了。

3、创建空的web项目 目录结构如下：



其中user实体类为:

```
public class User {  
    private String name;  
    private Integer age;  
    private Date birth;  
  
    public String getName() {  
        return name;  
    }  
  
    public void setName(String name) {  
        this.name = name;  
    }  
  
    public User(String name, Integer age, Date birth) {  
        super();  
        this.name = name;  
        this.age = age;  
        this.birth = birth;  
    }  
  
    public Integer getAge() {  
        return age;  
    }  
  
    public void setAge(Integer age) {  
        this.age = age;  
    }  
}
```

```

    public Date getBirth() {
        return birth;
    }

    public void setBirth(Date birth) {
        this.birth = birth;
    }

    public User() {
        super();
        // TODO Auto-generated constructor stub
    }

    public User(String name) {
        super();
        this.name = name;
    }

    @Override
    public String toString() {
        // TODO Auto-generated method stub
        return "["+name+";";
    }
}

```

4、在项目中添加如下jar文件：

```

spring-webmvc-3.2.0.RELEASE.jar
spring-core-3.2.0.RELEASE.jar
spring-context-3.2.0.RELEASE.jar
spring-beans-3.2.0.RELEASE.jar
spring-web-3.2.0.RELEASE.jar
commons-logging.jar
spring-expression-3.2.0.RELEASE.jar

```

其中commons-logging.jar请单独下载。

5、在web.xml中添加过滤器的配置。

```

<servlet>
    <servlet-name>example</servlet-name>

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

    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/spring-servlet.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>

```

```

</servlet>
<servlet-mapping>
    <servlet-name>example</servlet-name>
    <url-pattern>*.do</url-pattern>
</servlet-mapping>

```

这个org.springframework.web.servlet.DispatcherServlet即为springMVC的核心控制器。其中init-param中配置的是spring的配置文件的文件路径。

6、在WEB-INF下添加spring的配置文件spring-servlet.xml 文件内容如下: <?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:mvc="http://www.springframework.org/schema/mvc"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:util="http://www.springframework.org/schema/util"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-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/util
        http://www.springframework.org/schema/util/spring-util-3.0.xsd">

```

```

    <!-- 启动spring自动扫描 -->
    <context:component-scan base-package="com.niit.springmvc"/>
    <mvc:annotation-driven /> <!-- 支持spring3.0新的mvc注解 -->
    <!-- 启动Spring MVC的注解功能，完成请求和注解POJO的映射 -->
    <bean
class="org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandle
rAdapter"/>

</beans>

```

其中schemaLocation和xmlns建议直接拷贝。

7、创建UserController如下:

```

@Controller
@RequestMapping("/user.do")
public class UserController {

    @RequestMapping(params="method=add")
    public String addUser(Model model){
        model.addAttribute("message", "添加了一个用户");
        System.out.println("UserController.addUser()");
    }
}

```

```

        return "/WEB-INF/jsp/addsuc.jsp";
    }
}

```

其中@RequestMapping("/user.do")的配置是指：该controller的请求url为：user.do

@RequestMapping(params="method=add")的注解是指：凡是请求的url为:user.do而带了参数method=add的请求会由方法addUser来处理。

addUser的形参model为后续讲解内容。

return "/WEB-INF/jsp/addsuc.jsp"; 是告诉spring完成处理之后直接进入该视图。

8、添加对应的页面，测试请求user.do?method=add

springMVC 的 RequestMapping 的基本设置

1、在类的上面注解@RequestMapping("/ex.do")意思为所有的 ex.do 请求全部进入该类处理。如上一章代码中编写 @RequestMapping("/user.do")。所有的 user.do 请求都会进入该 Controller。

2、在自定义的controller中会调用有@RequestMapping注解字样的方法来处理请求。

```
@Controller
```

```
@RequestMapping("/user.do")
```

```
public class UserController {
```

```
    @RequestMapping
```

```
    public String addUser(Model model){
```

```
        model.addAttribute("message", "添加了一个用户");
```

```
        System.out.println("UserController.addUser()");
```

```
        return "/WEB-INF/jsp/addsuc.jsp";
```

```
    }
```

```
}
```

3、当然可以编写多个处理请求的方法，而这些方法的调用都是通过@RequestMapping的属性类控制调用的。

@RequestMapping属性：

value：指定请求的实际地址，指定的地址可以是URI Template 模式（最终请求的url为类的注解的url+方法注解的url）

value的uri值为以下三类：

A) 可以指定为普通的具体值；

如：

```
@Controller
```

```
@RequestMapping("/user")
```

```
public class UserController{
```

```
    @RequestMapping(value="/some.do")
```

```
    public ModelAndView handleRequest(HttpServletRequest arg0,
```

```
        HttpServletResponse arg1) throws Exception {
```

```
        System.out.println("handleRequest");
```

```

        return new ModelAndView("/WEB-INF/jsp/addsuc.jsp");
    }
}

```

该注解的是说：请求的url为"user/some.do"就会进入该方法（handleRequest）处理。

url: user/some.do

B) 可以指定为含有某变量的一类值 (URI Template Patterns with Path Variables);

如:

```

@RequestMapping(value="/{userId}/delete.do",method=RequestMethod.GET)

```

```

    public String delete(@PathVariable String userId){
        System.out.println("delete:"+userId);
        return "/WEB-INF/jsp/addsuc.jsp";
    }
}

```

这个注解：url中带了参数的数据 userId url: user/1123/delete.do

使用@PathVariable 指定形参接收url中的数据

C) 可以指定为含正则表达式的一类值 (URI Template Patterns with Regular Expressions);

如:

```

@RequestMapping(value="/{userBirth:\\d{4}-\\d{2}-\\d{2}}/update.do")

```

```

    public String update(@PathVariable String userBirth){
        System.out.println("userBirth:"+userBirth);
        return "/WEB-INF/jsp/addsuc.jsp";
    }
}

```

请求的url类似: user/1990-11-11/update.do

使用@PathVariable 指定形参接收url中的数据

method: 指定请求的method类型， GET、POST、PUT、DELETE等；（也就是说只有制定类型的请求才会进入该方法处理）

consumes: 指定处理请求的提交内容类型（Content-Type），例如application/json, text/html;

produces: 指定返回的内容类型，仅当request请求头中的 (Accept) 类型中包含该指定类型才返回;

params: 指定request中必须包含某些参数值是，才让该方法处理。

headers: 指定 request 中必须包含某些指定的 header 值，才能让该方法处理请求。

4、当类没有@RequestMapping 注解时，则直接参考方法的注解匹配对于的

url。如:

```

@Controller
public class UserController{
    @Controller
    @RequestMapping("/user.do")
    public void managerUser(){}
}

```



```
}
```

在这里 url 为 `user.do` 则直接使用 `managerUser` 处理请求。

在 controller 中获取前台传递的参数

将页面数据传递到 controller

页面表单:

```
<form action="user.do" method="post">

    用户名:<input type="text" name="name"/><br/>

    年龄:<input type="text" name="age"/><br/>

    生日:<input type="text" name="birth"/><br/>

    <input type="submit" value="添加"/>

</form>
```

Controller为:

```
/**
 * 1、直接使用形参获取前台传递的参数数据
 * 要注意的是形参的名字必须和页面参数的名字一致
 * @param model
 * @param name
 * @param age
 * @param birth
 * @return
 */
@RequestMapping(method=RequestMethod.POST)
public String addUser(Model model,String name,Integer
age,Date birth){

    model.addAttribute("message", "添加了一个用户");

    System.out.println("name:"+name +
"\tage:"+age+"\tbirht:"+birth);
    System.out.println("UserController.addUser()");
    return "/WEB-INF/jsp/addsuc.jsp";
}
```

```
/**
```

- * 2、使用对象接受前台传递的参数，

- * 要注意的是前台传递的参数名称必须和对象的属性名称一直，如果不

一致则可以使用@ModelAttribute("u")String uname指定

```
*/  
@RequestMapping(method=RequestMethod.POST)  
public String addUser(Model model, User user) {  
    model.addAttribute("message", "添加了一个用户");  
    System.out.println("name:"+user.getName() +  
"\tage:"+user.getAge()+"\tbirth:"+user.getBirth());  
    System.out.println("UserController.addUser()");  
    return "/WEB-INF/jsp/addsuc.jsp";  
}
```

在 controller 中获取 web 元素

当某个方法需要使用web对象时

(request, response, session, application)

可以使用如下方式：

除过application其他的对象都可以直接设为方法的形参即可。spring会自动将对应的对象传递给对应的形参。

而application对象可以使用session对象获取。

当然也可以在方法中使用response对象重定向到其他的url 这时方法最后

return的url则可以视作无效。

同样的也可以使用 request 对象转发到其他的 url。

程序示例：

```
@RequestMapping(value="/web.do")  
public String getWebElement(HttpServletRequest request,  
HttpServletResponse response, HttpSession session)
```

```

throws IOException, ServletException{

    System.out.println("使用request获取的前台参
数:"+request.getParameter("pname"));

    request.setAttribute("message", "这个是request中的数据");

    session.setAttribute("message", "这个是session中的数据");

    session.getServletContext().setAttribute("message",
"这个是application中的数据");

    //response.sendRedirect("http://www.baidu.com");
    //return null;

    //request.getRequestDispatcher("/WEB-INF/jsp/showData.
jsp").forward(request, response);

    return "/WEB-INF/jsp/showData.jsp";
}

```

将 controller 中数据传递到 jsp 页面

- 1、可以在controller中获取request对象，然后将数据设置为request对象的属性，然后使用转发的方式进入jsp即可。这一点不赘述。
- 2、将方法的返回值该为ModelAndView在返回时，将数据存储在

ModelAndView对象中如：

```

new
ModelAndView("/WEB-INF/jsp/showData.jsp","message",message)

```

其中第一个参数为url，第二个参数为要传递的数据的key，第三个参数为数据对象。

在这里要注意的是 数据是默认被存放在request中的。

程序示例：

```
//使用modelAndView对象将数据传递到前台。  
@RequestMapping(value="/mad/showData_1.do")  
public ModelAndView showData_1(){  
    String message = "这个是要传递的数据";  
  
    //其中第一个参数为url,第二个参数为要传递的数据的key,第三个参  
    数为数据对象。  
  
    //在这里要注意的是 数据是默认被存放在request中的。  
  
    return new  
    ModelAndView("/WEB-INF/jsp/showData.jsp","message",messag  
e);  
}
```

前台页面获取方式：

```
request:${requestScope.message }<br/>
```

2.1 、可以在类的前面添加注解

```
@SessionAttributes({"message","user"})
```

这个注解可以设置对应的model中参数也会在session中存储一份。该注解中的参数为一个集合，可以写多个，如上面的例子，其中message和user都是存储的数据的key。

示例程序：

```
@SessionAttributes({"message","user"})    //modelAndView中
```

的对应的数据也会在session中存储一份

页面获取：

```
session:${sessionScope.message }<br/>
```

3、数据modelAndView返回一个集合

该处理方式和上面的处理方式一直，因为modelAndView接受的数据类型是Object的，集合也是一样的处理方式

```
//使用modelAndView对象将数据传递到前台。
```

```

//传递多个参数（不同类型的）

@RequestMapping(value="/mad/showData_2.do")
public ModelAndView showData_2(){
    System.out.println("showData_2");

    String message = "这个是要传递的数据";

    User user = new User("张三", 12, new Date());
    List<User> us= new ArrayList<User>();

    us.add(new User("张三", 12, new Date()));

    us.add(new User("张四", 13, new Date()));

    us.add(new User("张五", 14, new Date()));

    ModelAndView mad = new
ModelAndView("/WEB-INF/jsp/showData.jsp");

    //将数据存入modelMap

    mad.addObject("message", message);

    mad.addObject(user); //默认为类名的首字母小写

    mad.addObject("users", us);
    return mad;
}

```

前台页面获取方式:

```

request:${requestScope.message }<br/>
<c:forEach items="${requestScope.users }" var="u">
    ${u.name }-${u.age }-${u.birth }<br/>
</c:forEach>

```

4、使用modelAndView传递多个参数。

可以通过ModelAndView的mad.addObject("message", message);
方法设置参数。

该方法中第一个参数为数据的key，第二个参数为数据对象。

也可以使用mad.addObject(user);方法

该方法中数据的参数为数据对象，数据的key为该对象的类的类名（其中首字母小写）。

5、使用ModelMap传递多个数据到jsp中。

在方法的参数列表中添加形参 ModelMap map,spring 会自动创建 ModelMap对象。

然后调用map的put(key,value)或者addAttribute(key,value)将数据放入map中，spring会自动将数据存入request。

示例程序：

//使用modelMap对象将数据传递到前台。

//传递多个参数（不同类型的）

```
@RequestMapping(value="/mad/showData_3.do")
public String showData_3(ModelMap map){
    System.out.println("showData_3");

    String message = "这个是要传递的数据";

    User user = new User("张三", 12, new Date());
    List<User> us= new ArrayList<User>();
    us.add(new User("张三", 12, new Date()));
    us.add(new User("张四", 13, new Date()));
    us.add(new User("张五", 14, new Date()));

    //将数据存入modelMap
    map.put("message", message);
    map.addAttribute("user", user);
    map.put("users", us);
    return "/WEB-INF/jsp/showData.jsp";
}
```

页面调用：

```
request:${requestScope.message }<br/>
session:${sessionScope.message }<br/>
application:${applicationScope.message }<br/>
<hr/>

<h1>ModelMap中的数据</h1>

${requestScope.message }<br/>
```

```

    ${requestScope.user.name }<br/>
<p>列表</p>
<c:forEach items="${requestScope.users }" var="u">
    ${u.name }-${u.age }-${u.birth }<br/>
</c:forEach>

```

设置跳转方式为重定向或者转发

- 1、spring默认的跳转方式即为转发，当然转发也可以写作：return "forward:/WEB-INF/jsp/showData.jsp";
- 2、重定向必须写作：return "redirect:http://www.baidu.com";

视图解析器的配置和使用

- 1、在spring-servlet.xml中配置视图解析器

```

<!-- 配置视图解析器 -->
<bean
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="suffix" value=".jsp"/> <!-- 视图后缀,controller中的方法返回的url字符串会添加该后缀 -->
    <property name="prefix" value="/WEB-INF/jsp/" /> <!-- 视图后缀controller中的方法返回的url字符串会添加该前缀 -->
</bean>

```

配置该解析器之后，那么 controller 中的返回的视图 url 就可以改写了。

改写如下：

```

return "showData";          实  际  跳  转  的  url  为  :
/WEB-INF/jsp/showData.jsp

```

controller 中方法的返回值类型

springMvc中controller中方法的返回值除了可以返回String和

ModelAndView两种类型外还有其他类型。

在这里上面用过的两种类型不再赘述。

其他类型还包括: void、ModelMap、Map、Object、map、List、Set。一般建议使用String。

1、void 返回值类型为void 则只是纯粹的执行了方法中的程序，然后响应的url依然为请求的url

例如案例中请求为index.do 则响应的url为 index 在视图解析器解析之后得到的最终的url为 /WEB-INF/jsp/index.jsp

示例:

//返回值为void

```
@RequestMapping(value="/index.do",params="type=void")
public void resultVoid(HttpServletRequest request){
    request.setAttribute("void", "resultVoid");
}
```

2、ModelMap 返回值为modelMap时，响应的url和void一致。

只是存储在ModelMap中的数据可以在jsp页面中取出。

示例:

//返回值为ModelMap

```
@RequestMapping(value="index.do",params="type=modelMap")
public ModelMap resultModelMap(ModelMap map){
    map.put("msg", "这里是modelMap中的数据");
    return map;
}
```

3、Map 和modelMap几乎完全一致。

示例:

//返回值为Map

```
@RequestMapping(value="index.do",params="type=map")
public Map resultMap(){
    Map<String,String> map = new HashMap<String,String>();
    map.put("msg", "这里是Map中的数据");
    return map;
}
```

4、List 返回list是响应的url和void一致。

只是spring会将list对象存储在request中，而该对象的存储的key为：
当list中存储为String类型数据时 key为：stringList，当存储为用户对象时 key为：userList。其他的类型的可以类比。

//返回值为List<String>

```
@RequestMapping(value="index.do",params="type=list_string")
public List resultList_String(){
    List ls = new ArrayList();
    ls.add("list1");ls.add("list2");ls.add("list3");
    return ls;
}
```

//返回值为List<User>

```
@RequestMapping(value="index.do",params="type=list_user")
public List<User> resultList_User(){
    List<User> ls = new ArrayList<User>();
    ls.add(new User("张三"));
    ls.add(new User("张四"));
    return ls;
}
```

5、Set 返回Set类型的数据时和List除了没有顺序之外，其他都一直。

//返回值为Set<User>

```
@RequestMapping(value="index.do",params="type=set_user")
```

```

public Set<User> resultSet_User() {
    Set<User> ls = new HashSet<User>();

    ls.add(new User("张三"));

    ls.add(new User("张四"));

    return ls;
}

```

6、Object 返回object时，响应的url和以上一直，spring也会将返回的对象存储在request中，该对象在request中的key为该对象类型的类名（首字母小写）

//返回值为User

```

@RequestMapping(value="index.do",params="type=user")
public User resultUser() {

    return new User("张四");

}

```

所有的返回值类型页面获取方式为:

<h1>返回值类型</h1>

<p>

<h3>Void</h3>

无返回值类型

`\${requestScope.void }`

</p>

<p>

<h3>ModelMap</h3>

返回ModelMap

`\${requestScope.msg }`

</p>

<p>

<h3>Map</h3>

返回map

`\${requestScope.msg }`

</p>

<p>

<h3>List</h3>

返回

```

List<String></a><br/>
    ${requestScope.stringList }
</p>
<p>
<h3>List</h3>

    <a href="index.do?type=list_user">返回

List<User></a><br/>
    ${requestScope.userList }
</p>
<h3>Set</h3>

    <a href="index.do?type=set_user">返回

Set<User></a><br/>
    ${requestScope.userSet }
</p>

    <h3>User (Object) </h3>

    <a href="index.do?type=user">返回User (Object) </a><br/>
    ${requestScope.user }
</p>

```

springMVC 的文件上传与下载

- 1、springmvc 文件的上传也是借助于两个工具所以需要添加两个jar

```

apache-commons-fileupload.jar
apache-commons-io.jar

```

- 2、在spring-servlet.xml中添加文件上传的处理bean的配置。

```

<bean id="multipartResolver"

    class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
        <property name="defaultEncoding" value="utf-8" /> <!--
默认编码 (ISO-8859-1) -->
        <property name="maxInMemorySize" value="10240" /> <!--

```

最大内存大小 (10240) -->

```
<property name="uploadTempDir" value="/temp/" />
```

<!-- (临时文件存储目录) 上传后的目录名

(WebUtils#TEMP_DIR_CONTEXT_ATTRIBUTE) -->

```
<property name="maxUploadSize" value="-1" /> <!-- 最
```

大文件大小, -1为无限止 (-1) -->

```
</bean>
```

其中属性<property name="uploadTempDir" value="/temp/" />

的配置配置的是临时文件目录, spring会将文件先传到临时文件, 然后我们再用对应的API将临时文件写到目标文件。

3、编写上传文件的controller

3.1上传一个文件

直接在处理的方法中设置形参@RequestParam("file")

```
CommonsMultipartFile file
```

注意这里的参数必须使用@RequestParam指定。

然后调用方法file.getFileItem().write(targetFile);将临时文件写出到目标文件。

示例:

```
/**
```

```
 * 上传一个文件
```

```
 * @param name
```

```
 * @param file
```

```
 * @param session
```

```
 * @return
```

```
 */
```

```
@RequestMapping(value="/upload.do",method=RequestMethod.POST)
```

```
public String fileUpload(String  
name,@RequestParam("file") CommonsMultipartFile  
file,HttpSession session){
```

```

        if(!file.isEmpty()){
            String path =
session.getServletContext().getRealPath("/upload/");
            String fileName = file.getOriginalFilename();
            String fileType =
fileName.substring(fileName.lastIndexOf("."));
            File targetFile = new File(path,new
Date().getTime()+fileType);
            try {
                file.getItem().write(targetFile);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        return "showData";
    }
}

```

3.2上传多个文件

上传多个文件时，其实和上传一个文件一样，只是将形参改为
@RequestParam("file") CommonsMultipartFile[] file

然后我们只需在方法中循环处理这些文件即可。

示例：

```

/**
 * 上传多个文件
 *
 * @param name
 * @param files
 * @param session
 * @return
 */
@RequestMapping(value="/mupload.do",method=RequestMethod.POST)
public String muFileUpload(String
name,@RequestParam("file") CommonsMultipartFile[]
files,HttpSession session){
    if(files!=null && files.length>0){
        String path =
session.getServletContext().getRealPath("/upload/");
        for (CommonsMultipartFile file : files) {
            String fileName = file.getOriginalFilename();
            String fileType =
fileName.substring(fileName.lastIndexOf("."));
            File targetFile = new File(path,new
Date().getTime()+fileType);

```

```

        try {
            file.getFileItem().write(targetFile);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    }

    return "showData";
}

```

4、文件下载

文件下载其实和 spring 没关系，还是使用最普通的方式实现下载即可，在这里不赘述。

示例：

```

/**
 * 文件下载
 * @param session
 * @param response
 * @param fileName
 * @param isOnline
 * @throws Exception
 */
@RequestMapping(value="/download.do",method=RequestMethod.GET)
public void download(HttpSession session,HttpServletResponse response,String fileName,boolean isOnline)throws Exception{
    String path =
session.getServletContext().getRealPath("/upload/")+"\\\\"+
fileName;
    File file = new File(path);
    System.out.println(path);
    if(!file.exists()){
        response.sendError(404, "您要下载的文件没找到");
        return;
    }
    BufferedInputStream bufIn = new
BufferedInputStream(new FileInputStream(file));
    byte [] buff = new byte[1024];
    int len = -1;

```

```

        response.reset();
        if(isOnline){
            URL u = new URL("file:///"+path);

            response.setContentType(u.openConnection().getContentType());
            response.setHeader("Content-Disposition",
"inline;filename="+fileName);

        }else{

            response.setContentType("application/x-msdownload");
            response.setHeader("Content-Disposition",
"attachment;filename="+fileName);
        }
        OutputStream out = response.getOutputStream();
        while((len=bufIn.read(buff))!=-1){
            out.write(buff,0,len);
            out.flush();
        }
        bufIn.close();
        out.close();
    }
}

```

springMVC 和 jQuery 的 Ajax 结合

1、使用@RequestBody 接收前台传递的json 集合数据。

首先:从spring3.1开始只要配置了<mvc:annotation-driven /> 就
不用再配置其他转换器了。

然后添加json的几个jar: jackson-annotations-2.4.0.jar,
jackson-core-2.4.1.jar, jackson-databind-2.4.1.jar

这些jar建议从官网上下载最新的

(<http://wiki.fasterxml.com/JacksonDownload>)

在这里要注意的是, 如果jackson的jar如果和spring的版本不匹配, 可

能会出现响应状态码415.如果出现415响应状态吗,不能解决可以联系此文档
的主人 qq:2780004063 或者发送邮件。

@RequestBody 将HTTP请求正文转换为适合的
HttpMessageConverter对象。

1.1

在前台js中创建JSON字符串。如

```
[{name:'11',age:12},{name:'222',age:15}],
```

然后使用JSON.stringify将该json对象转换为json字符串,因为

@RequestBody只接受json字符串。

示例:

```
$("#saveUser").click(function() {  
    var users = [{  
        name : '张三',  
        age : 18,  
        birth : '2014-11-11'  
    }, {  
        name : '王五',  
        age : 18,  
        birth : '2014-11-11'  
    }, {  
        name : '李四',  
        age : 18,  
        birth : '2014-11-11'  
    } ];  
    $.ajax({  
        type : 'POST',  
        data:JSON.stringify(users),  
        contentType : 'application/json',  
        dataType: 'json',  
        url : 'user/saveJsonUser.do',  
        success : function(data) {  
            alert("OK");  
        },  
        error : function(e) {
```



```

        alert("error");
    }
    });
});

```

1.2

使用Ajax提交数据需要注意的是：contentType：

'application/json', dataType: 'json',
这两个属性必须这样设置。

1.3

Controller中的处理方法的形参需要添加注解@RequestBody 而且形参必须是数组或者list。

如：@RequestBody User[] users

示例：

```

/**
 * 使用@RequestBody接受前台传递的一组json数据
 * @param users
 * @return
 */
@RequestMapping(value="/saveJsonUser.do",method=RequestMethod.POST)
public String saveJsonUser(@RequestBody User[] users){
    for (User user2 : users) {
        System.out.println(user2);
    }
    System.out.println(users);
    return "saveUser";
}

```

2、使用@Response返回指定形式的返回值。

在返回值类型钱添加@Response注解之后，spring不会再对返回的url进行解析，而是直接将返回的对象转化成对应的字符串形式放入respons的流中输出到客户端。

2.1如果返回的为字符串，则直接将该字符串输出到客户端。

示例：

```

/**
 * 使用@ResponseBody返回普通字符串。

```

```

    * @param name
    * @return
    */
@RequestMapping(value="/checkName.do")
public @ResponseBody String checkName(String name){
    name = "userName is:"+name;
    return name;
}

```

2.2 如果是其他形式的对象，则 spring 会自动将这些对象转换为对应的

json形式的字符串，然后将字符串输出到客户端。

示例：

```

/**
 * 使用@ResponseBody返回一个对象集合。
 * @param name
 * @return
 */
@RequestMapping(value="/getUsers.do")
public @ResponseBody List<User> getUsers(){
    List<User> us = new ArrayList<User>();

    us.add(new User("张三", 12, new Date()));

    us.add(new User("张四", 13, new Date()));

    us.add(new User("张五", 14, new Date()));

    return us;
}

/**
 * 使用@ResponseBody返回一个对象。
 * @param name
 * @return
 */
@RequestMapping(value="/getUser.do")
public @ResponseBody User getUser(){
    return new User("老黑", 45, new Date());
}

```

还有 springMVC 和 hibern 的整合尚未整理，待续。