一、SpringMVC的各种参数绑定方式（后端controller获取前端数据）

**1. 基本数据类型(以int为例，其他类似)：**  
Controller代码：

@RequestMapping("saysth.do")

public void test(int count) {

}

表单代码：

<form action="saysth.do" method="post">

<input name="count" value="10" type="text"/>

......

</form>

表单中input的name值和Controller的参数变量名保持一致，就能完成数据绑定，如果不一致可以使用@RequestParam注解。需要注意的是，如果Controller方法参数中定义的是基本数据类型，但是从页面提交过来的数据为null或者”"的话，会出现数据转换的异常。也就是必须保证表单传递过来的数据不能为null或”"，所以，在开发过程中，对可能为空的数据，最好将参数数据类型定义成包装类型，具体参见下面的例子。

**2. 包装类型(以Integer为例，其他类似)：**  
Controller代码：

@RequestMapping("saysth.do")

public void test(Integer count) {

}

表单代码：

<form action="saysth.do" method="post">

<input name="count" value="10" type="text"/>

......

</form>

和基本数据类型基本一样，不同之处在于，表单传递过来的数据可以为null或”"，以上面代码为例，如果表单中num为”"或者表单中无num这个input，那么，Controller方法参数中的num值则为null。

**3. 自定义对象类型：**  
Model代码：

[复制代码](javascript:void(0);)

public class User {

private String firstName;

private String lastName;

public String getFirstName() {

return firstName;

}

public void setFirstName(String firstName) {

this.firstName = firstName;

}

public String getLastName() {

return lastName;

}

public void setLastName(String lastName) {

this.lastName = lastName;

}

}

[复制代码](javascript:void(0);)

Controller代码：

@RequestMapping("saysth.do")

public void test(User user) {

}

表单代码：

<form action="saysth.do" method="post">

<input name="firstName" value="张" type="text"/>

<input name="lastName" value="三" type="text"/>

......

</form>

非常简单，只需将对象的属性名和input的name值一一匹配即可。

**4. 自定义复合对象类型：**  
Model代码：

[复制代码](javascript:void(0);)

public class ContactInfo {

private String tel;

private String address;

public String getTel() {

return tel;

}

public void setTel(String tel) {

this.tel = tel;

}

public String getAddress() {

return address;

}

public void setAddress(String address) {

this.address = address;

}

}

public class User {

private String firstName;

private String lastName;

private ContactInfo contactInfo;

public String getFirstName() {

return firstName;

}

public void setFirstName(String firstName) {

this.firstName = firstName;

}

public String getLastName() {

return lastName;

}

public void setLastName(String lastName) {

this.lastName = lastName;

}

public ContactInfo getContactInfo() {

return contactInfo;

}

public void setContactInfo(ContactInfo contactInfo) {

this.contactInfo = contactInfo;

}

}

[复制代码](javascript:void(0);)

Controller代码：

[复制代码](javascript:void(0);)

@RequestMapping("saysth.do")

public void test(User user) {

System.out.println(user.getFirstName());

System.out.println(user.getLastName());

System.out.println(user.getContactInfo().getTel());

System.out.println(user.getContactInfo().getAddress());

}

[复制代码](javascript:void(0);)

表单代码：

[复制代码](javascript:void(0);)

<form action="saysth.do" method="post">

<input name="firstName" value="张" /><br>

<input name="lastName" value="三" /><br>

<input name="contactInfo.tel" value="13809908909" /><br>

<input name="contactInfo.address" value="北京海淀" /><br>

<input type="submit" value="Save" />

</form>

[复制代码](javascript:void(0);)

User对象中有ContactInfo属性，Controller中的代码和第3点说的一致，但是，在表单代码中，需要使用“属性名(对象类型的属性).属性名”来命名input的name。

**5. List绑定：**  
List需要绑定在对象上，而不能直接写在Controller方法的参数中。  
Model代码：

[复制代码](javascript:void(0);)

public class User {

private String firstName;

private String lastName;

public String getFirstName() {

return firstName;

}

public void setFirstName(String firstName) {

this.firstName = firstName;

}

public String getLastName() {

return lastName;

}

public void setLastName(String lastName) {

this.lastName = lastName;

}

}

public class UserListForm {

private List<User> users;

public List<User> getUsers() {

return users;

}

public void setUsers(List<User> users) {

this.users = users;

}

}

[复制代码](javascript:void(0);)

Controller代码：

@RequestMapping("saysth.do")

public void test(UserListForm userForm) {

for (User user : userForm.getUsers()) {

System.out.println(user.getFirstName() + " - " + user.getLastName());

}

}

表单代码：

[复制代码](javascript:void(0);)

<form action="saysth.do" method="post">

<table>

<thead>

<tr>

<th>First Name</th>

<th>Last Name</th>

</tr>

</thead>

<tfoot>

<tr>

<td colspan="2"><input type="submit" value="Save" /></td>

</tr>

</tfoot>

<tbody>

<tr>

<td><input name="users[0].firstName" value="aaa" /></td>

<td><input name="users[0].lastName" value="bbb" /></td>

</tr>

<tr>

<td><input name="users[1].firstName" value="ccc" /></td>

<td><input name="users[1].lastName" value="ddd" /></td>

</tr>

<tr>

<td><input name="users[2].firstName" value="eee" /></td>

<td><input name="users[2].lastName" value="fff" /></td>

</tr>

</tbody>

</table>

</form>

[复制代码](javascript:void(0);)

其实，这和第4点User对象中的contantInfo数据的绑定有点类似，但是这里的UserListForm对象里面的属性被定义成List，而不是普通自定义对象。所以，在表单中需要指定List的下标。值得一提的是，Spring会创建一个以最大下标值为size的List对象，所以，如果表单中有动态添加行、删除行的情况，就需要特别注意，譬如一个表格，用户在使用过程中经过多次删除行、增加行的操作之后，下标值就会与实际大小不一致，这时候，List中的对象，只有在表单中对应有下标的那些才会有值，否则会为null，看个例子：

表单代码：

[复制代码](javascript:void(0);)

<form action="saysth.do" method="post">

<table>

<thead>

<tr>

<th>First Name</th>

<th>Last Name</th>

</tr>

</thead>

<tfoot>

<tr>

<td colspan="2"><input type="submit" value="Save" /></td>

</tr>

</tfoot>

<tbody>

<tr>

<td><input name="users[0].firstName" value="aaa" /></td>

<td><input name="users[0].lastName" value="bbb" /></td>

</tr>

<tr>

<td><input name="users[1].firstName" value="ccc" /></td>

<td><input name="users[1].lastName" value="ddd" /></td>

</tr>

<tr>

<td><input name="users[20].firstName" value="eee" /></td>

<td><input name="users[20].lastName" value="fff" /></td>

</tr>

</tbody>

</table>

</form>

[复制代码](javascript:void(0);)

这个时候，Controller中的userForm.getUsers()获取到List的size为21，而且这21个User对象都不会为null，但是，第2到第19的User对象中的firstName和lastName都为null。打印结果：

[复制代码](javascript:void(0);)

aaa - bbb

ccc - ddd

null - null

null - null

null - null

null - null

null - null

null - null

null - null

null - null

null - null

null - null

null - null

null - null

null - null

null - null

null - null

null - null

null - null

null - null

eee - fff

[复制代码](javascript:void(0);)

**6. Set绑定：**  
Set和List类似，也需要绑定在对象上，而不能直接写在Controller方法的参数中。但是，绑定Set数据时，必须先在Set对象中add相应的数量的模型对象。  
Model代码：

[复制代码](javascript:void(0);)

public class User {

private String firstName;

private String lastName;

public String getFirstName() {

return firstName;

}

public void setFirstName(String firstName) {

this.firstName = firstName;

}

public String getLastName() {

return lastName;

}

public void setLastName(String lastName) {

this.lastName = lastName;

}

}

public class UserSetForm {

private Set<User> users = new HashSet<User>();

public UserSetForm() {

users.add(new User());

users.add(new User());

users.add(new User());

}

public Set<User> getUsers() {

return users;

}

public void setUsers(Set<User> users) {

this.users = users;

}

}

[复制代码](javascript:void(0);)

Controller代码：

@RequestMapping("saysth.do")

public void test(UserSetForm userForm) {

for (User user : userForm.getUsers()) {

System.out.println(user.getFirstName() + " - " + user.getLastName());

}

}

表单代码：

[复制代码](javascript:void(0);)

<form action="saysth.do" method="post">

<table>

<thead>

<tr>

<th>First Name</th>

<th>Last Name</th>

</tr>

</thead>

<tfoot>

<tr>

<td colspan="2"><input type="submit" value="Save" /></td>

</tr>

</tfoot>

<tbody>

<tr>

<td><input name="users[0].firstName" value="aaa" /></td>

<td><input name="users[0].lastName" value="bbb" /></td>

</tr>

<tr>

<td><input name="users[1].firstName" value="ccc" /></td>

<td><input name="users[1].lastName" value="ddd" /></td>

</tr>

<tr>

<td><input name="users[2].firstName" value="eee" /></td>

<td><input name="users[2].lastName" value="fff" /></td>

</tr>

</tbody>

</table>

</form>

[复制代码](javascript:void(0);)

基本和List绑定类似。  
需要特别提醒的是，如果最大下标值大于Set的size，则会抛出org.springframework.beans.InvalidPropertyException异常。所以，在使用时有些不便。

**7. Map绑定：**  
Map最为灵活，它也需要绑定在对象上，而不能直接写在Controller方法的参数中。  
Model代码：

[复制代码](javascript:void(0);)

public class User {

private String firstName;

private String lastName;

public String getFirstName() {

return firstName;

}

public void setFirstName(String firstName) {

this.firstName = firstName;

}

public String getLastName() {

return lastName;

}

public void setLastName(String lastName) {

this.lastName = lastName;

}

}

public class UserMapForm {

private Map<String, User> users;

public Map<String, User> getUsers() {

return users;

}

public void setUsers(Map<String, User> users) {

this.users = users;

}

}

[复制代码](javascript:void(0);)

Controller代码：

[复制代码](javascript:void(0);)

@RequestMapping("saysth.do")

public void test(UserMapForm userForm) {

for (Map.Entry<String, User> entry : userForm.getUsers().entrySet()) {

System.out.println(entry.getKey() + ": " + entry.getValue().getFirstName() + " - " +

entry.getValue().getLastName());

}

}

[复制代码](javascript:void(0);)

表单代码：

[复制代码](javascript:void(0);)

<form action="saysth.do" method="post">

<table>

<thead>

<tr>

<th>First Name</th>

<th>Last Name</th>

</tr>

</thead>

<tfoot>

<tr>

<td colspan="2"><input type="submit" value="Save" /></td>

</tr>

</tfoot>

<tbody>

<tr>

<td><input name="users['x'].firstName" value="aaa" /></td>

<td><input name="users['x'].lastName" value="bbb" /></td>

</tr>

<tr>

<td><input name="users['y'].firstName" value="ccc" /></td>

<td><input name="users['y'].lastName" value="ddd" /></td>

</tr>

<tr>

<td><input name="users['z'].firstName" value="eee" /></td>

<td><input name="users['z'].lastName" value="fff" /></td>

</tr>

</tbody>

</table>

</form>

[复制代码](javascript:void(0);)

打印结果：

x: aaa - bbb

y: ccc - ddd

z: eee - fff

二、前端jsp获取后端controller数据

1、 @RequestMapping("/")

**public** ModelAndView getIndex(){

ModelAndView mav = **new** ModelAndView("index");

User user = userDao.selectUserById(1);

mav.addObject("user", user);

**return** mav;

}

返回ModeAndView对象中addObject方法将对象直接绑定返回index（在spring-mvc.xml中配置对模型视图添加前后缀）视图。Jsp中value="${user.userId}"使用el表达式直接输出。

## 2、向前台传递参数

[复制代码](javascript:void(0);)

//pass the parameters to front-end

@RequestMapping("/show")

public String showPerson(Map<String,Object> map){

Person p =new Person();

map.put("p", p);

p.setAge(20);

p.setName("jayjay");

return "show";

}

[复制代码](javascript:void(0);)

前台可在Request域中取到"p"

二、使用ajax传值例如：

在jsp页面中ajax如下

$.ajax({

type:"POST",

url:"test.do", /\* 这里就是action名+要执行的action中的函数 \*/

contentType: "application/json; charset=utf-8",

data:JSON.stringify({name:$("#userName").val()}), //url后面要传送的参数

async: **true**,

dataType : "json",

success:**function**(data){

alert(data.name);}

});

**后台controller中如下：**

@RequestMapping("/test.do")

@ResponseBody //在springMVC中提供了JSON响应的支持

**public** Map<String,Object> test2(@RequestBody Map<String, String> map){

String name;

**if**(map.containsKey("name")){

name=map.get("name");

}**else**{

name=**null**;

}

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

mapout.put("id", 16);

mapout.put("name",name.trim());

**return** mapout;

}

当然因为springmvc会自动绑定参数名，以及类级别的属性所以这里可以直接获取

**public** Map<String,Object> test2(@RequestBody String name){

。。。。省略

使用ajax提交后controller返回的Response中有视图，实现跳转如下：

以为ajax提交后宾部能够在controller中实现跳转（可以返回数据在页面，刷新），只能通过其他途径来实现。

$(**function**(){

$("#reg01").click(**function**(){

$.get("showUser.do",**function**(data){

window.location.href = "showUser.do";

});

});

});

数据提交后，跳转到提交返回的页面。

Spring mvc cotnroller几种返回类型：cotnrolle处理方法支持如下的返回方式：ModelAndView, Model, ModelMap, Map,View, String, void。

1. ModelAndView  返回的是一个包含模型和视图的ModelAndView对象；
2. //对于ModelAndView构造函数可以指定返回页面的名称，也可以通过setViewName方法来设置所需要跳转的页面；
3. @RequestMapping(value="/index2",method=RequestMethod.GET)
4. **public** ModelAndView index2(){
5. ModelAndView modelAndView = **new** ModelAndView();
6. modelAndView.addObject("name", "xxx");
7. modelAndView.setViewName("/user/index");
8. **return** modelAndView;
9. }
10. //返回的是一个包含模型和视图的ModelAndView对象；
11. Model   Model一个模型对象主要包含spring封装好的model和modelMap,以及java.util.Map， 当没有视图返回的时候视图名称将由requestToViewNameTranslator决定；
12. Map
13. @RequestMapping(value="/index3",method=RequestMethod.GET)
14. **public** Map<String, String> index3(){
15. Map<String, String> map = **new** HashMap<String, String>();
16. map.put("1", "1");
17. //map.put相当于request.setAttribute方法
18. **return** map;
19. }
20. //响应的view应该也是该请求的view。等同于void返回。
21. 返回String
22. //通过model进行封装数据产地
23. @RequestMapping(value="/index4",method = RequestMethod.GET)
24. **public** String index(Model model) {
25. String retVal = "user/index";
26. User user = **new** User();
27. user.setName("XXX");
28. model.addAttribute("user", user);
29. **return** retVal;
30. }
31. String 返回，使用json处理
32. //通过配合@ResponseBody来将内容或者对象作为HTTP响应正文返回（适合做即时校验）；
33. @RequestMapping(value = "/valid", method = RequestMethod.GET)
34. @ResponseBody
35. **public** String valid(@RequestParam(value = "userId", required = **false**) Integer userId,
36. @RequestParam(value = "name") String name) {
37. **return** String.valueOf(**true**);
38. }
39. //返回字符串表示一个视图名称，这个时候如果需要在渲染视图的过程中需要模型的话，就可以给处理器添加一个模型参数，然后在方法体往模型添加值就可以了，
40. void
41. @RequestMapping(method=RequestMethod.GET)
42. **public** **void** index5(){
43. ModelAndView modelAndView = **new** ModelAndView();
44. modelAndView.addObject("xxx", "xxx");
45. }
46. //返回的结果页面还是：/type
47. //这个时候我们一般是将返回结果写在了HttpServletResponse 中了，如果没写的话，
48. //spring就会利用RequestToViewNameTranslator 来返回一个对应的视图名称。如果这个时候需要模型的话，处理方法和返回字符串的情况是相同的。
50. }

访问web\_inf下的资源的一种方式

引文web\_inf下的资源是安全的禁止直接访问，所以一般通过后台的spring mvc 中的配置映射到controller内转发。以下是集中方式：

方法一

本来WEB-INF中的jsp就是无法通过地址栏访问的，所以安全。

如果说你要访问这个文件夹中的jsp文件需要在项目的web.xml文件中去配置servlet格式差不多的配置就ok了  
      如下:

**[html]** [view plain](http://blog.csdn.net/liu578182160/article/details/50685770) [copy](http://blog.csdn.net/liu578182160/article/details/50685770)

[在CODE上查看代码片](https://code.csdn.net/snippets/1581451)

1. **<servlet>**
2. **<servlet-name>**runtain**</servlet-name>**
3. **<jsp-file>**/WEB-INF/INF.jsp**</jsp-file>**
4. **</servlet>**
5. **<servlet-mapping>**
6. **<servlet-name>**runtain**</servlet-name>**
7. **<url-pattern>**/XXX**</url-pattern>**

访问地址:http://localhost:8080/runtain/xxx  
即可访问jsp页面内容  
方法二

<jsp:forward page = "/WEB-INF/jsp/test/test.jsp" />  
方法三

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

怎么样让servlet访问web-inf下的网页或jsp文件呢？

因为web-inf下,应用服务器把它指为禁访目录，即直接在浏览器里是不能访问到的。  
因些，可以让servlet进行访问，如web-inf下有a.jsp，则可以用request.getRequestDispatcher("/WEB-INF/a.jsp").forward(request,response);进行派遣访问<jsp:forward page = "/WEB-INF/leaveoff/leavestart.jsp" />同理

但如果web-inf下有a.htm,则用request.getRequestDispatcher("/WEB-INF/a.htm").forward(request,response);则不能访问。

一开始想不通，觉得怪。后来想想，jsp其实也是servlet,会自动编译的，于是work目录下会有/web-inf/a$jsp.class类型，于是有头绪了,让应用服务器能够编译.htm,如a$htm.class.抱有这个想法，开始动手

在tomcat下的conf/web.xml，找到jsp的访问方式

**[html]** [view plain](http://blog.csdn.net/liu578182160/article/details/50685770) [copy](http://blog.csdn.net/liu578182160/article/details/50685770)

[在CODE上查看代码片](https://code.csdn.net/snippets/1581451)

1. **<servlet-mapping>**
2. **<servlet-name>**jsp**</servlet-name>**
3. **<url-pattern>**\*.jsp**</url-pattern>**
4. **</servlet-mapping>**

于是在下面添加

**[html]** [view plain](http://blog.csdn.net/liu578182160/article/details/50685770) [copy](http://blog.csdn.net/liu578182160/article/details/50685770)

[在CODE上查看代码片](https://code.csdn.net/snippets/1581451)

1. **<servlet-mapping>**
2. **<servlet-name>**jsp**</servlet-name>**
3. **<url-pattern>**\*.htm**</url-pattern>**
4. **</servlet-mapping>**
5. **<servlet-mapping>**
6. **<servlet-name>**jsp**</servlet-name>**
7. **<url-pattern>**\*.html**</url-pattern>**
8. **</servlet-mapping>**

随后，一切OK，此时可访问a.htm。

1. html在work/web-inf/下者有a$htm.class,a$html.class生成

**jquery ajax向spring mvc controller中传值并接受并返回**

### 第一种传值：

controller中是几个单独的基本类型参数

[**spring**](http://lib.csdn.net/base/javaee) MVC-controller

1. @RequestMapping("update")
2. @ResponseBody//此注解不能省略 否则ajax无法接受返回值
3. **public** Map<String,Object> update(Long num, Long id, BigDecimal amount){
4. Map<String,Object> resultMap = **new** HashMap<String, Object>();
5. **if**(num == **null** || agentId == **null** || amount == **null**){
6. resultMap.put("result", "参数不合法！");
7. **return** resultMap;
8. }
9. //xxx逻辑处理
10. resultMap.put("result", result);
11. **return** resultMap;
12. }

[**jQuery**](http://lib.csdn.net/base/jquery) ajax

1. var params = {};
2. //params.XX必须与Spring Mvc controller中的参数名称一致
3. //否则在controller中使用@RequestParam绑定
4. params.num = num;
5. params.id = id;
6. params.amount = amount;
7. $.ajax({
8. async:**false**,
9. type: "POST",
10. url: "price/update",//注意路径
11. data:params,
12. dataType:"json",
13. success:function(data){
14. **if**(data.result=='SUCCESS'){
15. alert("修改成功");
16. }**else**{
17. alert("修改失败，失败原因【" + data + "】");
18. }
19. },
20. error:function(data){
21. alert(data.result);
22. }
23. });

### 第二种传值：

controller中是参数是实体bean，bean中属性都是基本数据类型

Spring MVC-controller

1. @RequestMapping("add")
2. @ResponseBody//此处不能省略 否则ajax无法解析返回值
3. **public** Map<String,Object> add(DataVo dataVo){
4. Map<String, Object> result = **null**;
5. **if**(dataVo.getNum() == **null** || StringUtils.isBlank(dataVo.geId())){
6. result = **new** HashMap<String, Object>();
7. result.put("msg", "参数不合法！");
8. **return** result;
9. }
10. //xxx业务逻辑处理
11. **return** result;
12. }

实体bean DataVo

1. **public** **class** DataVo {
2. /\*\*
3. \* 编号
4. \*/
5. **private** Long num;
6. /\*\*
7. \* id
8. \*/
9. **private** String id;
11. **public** Long getNum() {
12. **return** num;
13. }
14. **public** **void** setNum(Long num) {
15. **this**.num = num;
16. }
17. **public** String getId() {
18. **return** id;
19. }
20. **public** **void** setId(String id) {
21. **this**.id = id;
22. }
23. }

jquery ajax

1. var params = {};
2. params.num = $("#num").val();
3. params.id = $("#id").val();//注意params.名称  名称与实体bean中名称一致
4. $.ajax({
5. type: "POST",
6. url: "price/add",
7. data:params,
8. dataType:"json",
9. //                     contentType: "application/json; charset=utf-8",//此处不能设置，否则后台无法接值
10. success:function(data){
11. **if**(data.msg != ""){
12. alert( data.msg );
13. }
14. },
15. error:function(data){
16. alert("出现异常，异常原因【" + data + "】!");
17. }
18. });

### 第三种传值：

controller中是参数是实体bean，bean中属性有数组

Spring MVC-controller

1. @RequestMapping("add")
2. @ResponseBody//此处不能省略 否则ajax无法解析返回值
3. **public** Map<String,Object> add(@RequestBody DataVo dataVo){//@RequestBody注解不能省略，否则无法接值
4. Map<String,Object> resultMap = **new** HashMap<String, Object>();
5. //业务逻辑处理
6. **return** resultMap;
7. }

实体 DataVo

1. **public** **class** DataVo {
3. **private** BigDecimal[] nums;
4. **private** String id;
6. **public** Long getId() {
7. **return** id;
8. }
10. **public** **void** setId(Long id) {
11. **this**.id = id;
12. }
14. **public** BigDecimal[] getNums() {
15. **return** nums;
16. }
18. **public** **void** setNums(BigDecimal[] nums) {
19. **this**.nums = nums;
20. }
22. }

jquery ajax  需要jquery json的插件  进行json序列化，我这里使用了json.js

且配置

datatype:"json",

contentType: "application/json; charset=utf-8",

1. var params = {};
2. params.nums = [];
3. params.id = $("#id").val();//parmas.参数名 注意与实体bean参数名称相同
4. var prices = document.getElementsByName("prices");//prices 是name="prices"一组input标签
5. **for** (var i = 0; i < prices.length; i++) {
6. params.nums[i] =  prices[i].value;
7. }
8. $.ajax({
9. type: "POST",
10. url: "price/add",
11. data:JSON.stringify(params),//json序列化
12. datatype:"json", //此处不能省略
13. contentType: "application/json; charset=utf-8",//此处不能省略
14. success:function(data){
15. alert(data);
16. },
17. error:function(data){
18. alert(data)
19. }
20. });