

# 课程目标

- **SpringMVC的各种参数绑定方式：本章节重点掌握不使用注解的参数绑定方式和类型**
- **SpringMVC里的Model、Map、ModelMap以及ModelAndView**
- **SpringMVC Controller方法支持的返回值类型**

# SpringMVC的各种参数绑定方式

# 参数绑定

- SpringMVC中Controller的方法参数可以是简单数据类型（String和基本数据类型），包装类，自定义对象，ServletRequest，ServletResponse，ModelAndView 等等，非常灵活。
- 当View提交的参数名和对应Controller方法参数名对应时，可以不使用注解，Spring会自动进行绑定。

# 简单数据类型

## □ Login.jsp

```
<body>
  <form action="test3/testSimple" method="post">
    <label id="namelabel" for="name">请输入姓名</label>
    <input name="name" id="name">
    <label id="psdlabel" for="age">请输入年龄</label>
    <input name="age" id="age">
    <input type="submit" value="登陆">
  </form>

</body>
```

## □Controller:

```
@RequestMapping(value="/testSimple")
public String testSimple(String name,int age){

    System.out.println("入参--姓名: "+name);
    System.out.println("入参--年龄: "+age);
    return "success";
}
```

- 注意：View层和Controller方法参数必须一致，并且View层输入age参数值必须符合整型数字格式且不能为空（Spring类型转换后面详解），否则浏览器会报400错误。

# 包装类

- 使用包装类的好处在于，View参数值为null或空串的情况下，不会报400错误，Controller可以接受，值为null的参数...

```
@RequestMapping(value="/testSimple")
public String testSimple(String name,Integer age){

    System.out.println("入参--姓名: "+name);
    System.out.println("入参--年龄: "+age);
    return "success";
}
```

# 简单对象类型

- 与基本类型相似,只不过绑定到对象上更加简洁.(类似struts的ActionForm); 对应类型中要有对应的属性和正确的setter方法。

```
public class Student {  
    private String name;  
    private Integer age;  
  
    //省略getter/setter方法
```

```
@RequestMapping(value="/testSimple")  
public String testSimple(Student s) {  
    System.out.println(s);  
    return "success";  
}
```

# List和Set类型

- List需要绑定在对象(ActionForm),直接写在request-mapping函数的参数是不行的,更重要的一点是要创建对象(ArrayList); Set同List.



# List和Set类型

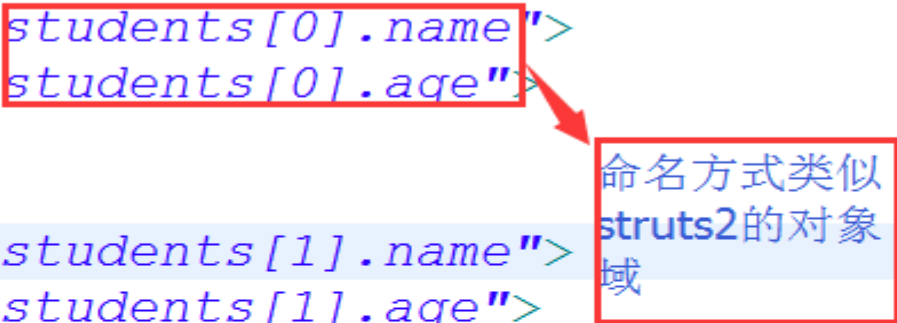
## □添加类Club

```
public class Club {  
    List<Student> students = new ArrayList<Student>();  
  
    public List<Student> getStudents() {  
        return students;  
    }  
  
    public void setStudents(List<Student> students) {  
        this.students = students;  
    }  
}
```

# List和Set类型

## □jsp

```
<form action="test3/testConnection" method="post">
学生1:
    姓名: <input name="students[0].name">
    年龄: <input name="students[0].age">
<br /><br />
学生2:
    姓名: <input name="students[1].name">
    年龄: <input name="students[1].age">
<br /><br />
学生3:
    姓名: <input name="students[2].name">
    年龄: <input name="students[2].age">
<br /><br />
<input type="submit" value="提交">
```



# List和Set类型

□Controller: 入参为Club类型

```
@RequestMapping(value="/testConnection")  
public String testSimple(Club club) {  
    for(Student s:club.getStudents()) {  
        System.out.println(s);  
    }  
    return "success";  
}
```

# Map类型绑定

□最灵活的一种方式,可无限绑定未定义的参数,注意必须绑定在对象下(ActionForm).

□Club中添加studentsMap及对应的getter/setter

```
Map<String, Student> studentsMap = new LinkedHashMap<String, Student>();  
//省略getter/setter方法
```

# Map类型绑定

## □Jsp

```
<form action="test3/testMap" method="post">
学生1:
    姓名: <input name="studentsMap['s1'].name">
    年龄: <input name="studentsMap['s1'].age">
<br /><br />
学生2:
    姓名: <input name="studentsMap['s2'].name">
    年龄: <input name="studentsMap['s2'].age">
<br /><br />
学生3:
    姓名: <input name="studentsMap['s3'].name">
    年龄: <input name="studentsMap['s3'].age">
<br /><br />
<input type="submit" value="提交">
```

# Map类型绑定

## □Controller:

```
@RequestMapping(value="/testMap")
public String testSimple(Club club){
    for(String key:club.getStudentsMap().keySet()){
        System.out.println(key+"---->" +club.getStudentsMap().get(key));
    }
    return "success";
}
```

# **Model、Map、ModelMap 以及ModelAndView**

# Spring模型数据的存储容器

- Spring Web MVC 提供Model、Map或ModelMap让我们能去暴露渲染视图需要的模型数据（向视图层传递参数）。
- SpringMVC在调用方法前会创建一个隐含的数据模型，作为模型数据的存储容器，成为“隐含模型”。
- 如果处理方法入参为Map或者Model类型，SpringMVC会将隐含模型的引用传递给这些入参。
- 注意：以上存储容器在spring中的作用域范围为request级。



# Spring模型数据的存储容器

- SpringMVC内部使用一个  
`org.springframework.ui.Model`接口存储的数据模型，  
它的功能类似于`java.util.Map`,但是比`Map`更好用，其实  
现类为`ExtendedModelMap`，继承了`ModelMap`类。

# Spring模型数据的存储容器

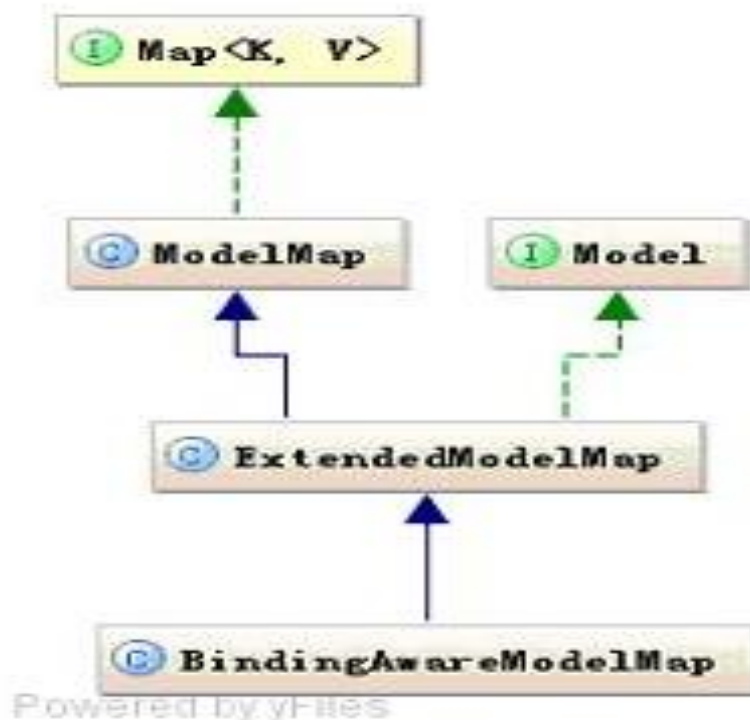
□ **ModelMap**对象主要用于传递控制方法处理数据到结果页面，也就是说我们把结果页面上需要的数据放到ModelMap对象中即可，他的作用类似于request对象的setAttribute方法的作用，用来在一个请求过程中传递处理的数据。通过以下方法向页面传递参数：

`addAttribute(String key, Object value);`

在页面上可以通过el变量方式\$`key`获取并展示modelmap中的数据。

modelmap本身不能设置页面跳转的url地址别名或者物理跳转地址，那么我们可以通过控制器方法的返回值来设置跳转url地址别名或者物理跳转地址。

# Model、Map或ModelMap关系图



# Model、Map或ModelMap实例

## □Controller

```
@Controller
@RequestMapping(value="/test3")
public class HelloWorldController{

    @RequestMapping(value="/testModel")
    public String testSimple(Map<String,Object> map,Model model,ModelMap modelMap){

        map.put("mapKey", "mapValue");
        model.addAttribute("modelKey","modelValue");
        modelMap.addAttribute("modelMapKey","modelMapValue");
        return "success";
    }

}
```

# Model、Map或ModelMap实例

## □view

```
<body>
    跳转成功,
    <p> Map中的值为${requestScope.mapKey}</p>
    <p> Model中的值为${requestScope.modelKey}</p>
    <p> ModelMap中的值为${requestScope.modelMapKey}</p>
    <%=request.getAttribute("mapKey") %>
</body>
```

# ModelAndView

- ModelAndView: 是包含ModelMap 和视图对象的容器。  
正如名字暗示的一样既包含模型也包含视图。

```
@Controller
@RequestMapping(value="/test3")
public class HelloWorldController{

    @RequestMapping(value="/testModelAndView")
    public ModelAndView testModelAndView(){
        ModelAndView mdv =new ModelAndView();
        //设置视图名
        mdv.setViewName("success");

        //作用域为request级的数据，数据名为testModelAndView
        mdv.addObject("testModelAndView", "test");
        return mdv;
    }
}
```

# ModelAndView

□ ModelAndView对象有两个作用：

➤ 作用一：设置转向地址,如下所示（这也是ModelAndView和ModelMap的主要区别）

```
ModelAndView view = new ModelAndView("path:ok");
```

➤ 作用二：用于传递控制方法处理结果数据到结果页面，也就是说我们把需要在结果页面上需要的数据放到ModelAndView对象中即可，他的作用类似于request对象的setAttribute方法的作用，用来在一个请求过程中传递处理的数据。通过以下方法向页面传递参数：**addObject(String key,Object value);**

# SpringMVC和ServletAPI

□Spring中获取ServletAPI对象，有3种方法。

- 1. 注解法自动注入
- 2. 使用RequestContextHolder获取（麻烦）
- 3. ServletAPI对象直接作为Controller方法的入参（最直接）



# 注解法自动注入

```
@RequestMapping(value="/test4")  
public class HelloWorldController{
```

```
    @Autowired  
    HttpServletRequest request;  
    @Autowired  
    HttpServletResponse response;  
    @Autowired  
    HttpSession session;
```

```
    @RequestMapping(value="/testServlet")
```

```
    public String testServlet(){  
        request.setAttribute("requestTest", "requestValue");  
        System.out.println(response.getLocale());  
        session.setAttribute("sessionTest", "sessionValue");  
        return "success";  
    }
```

# 使用RequestContextHolder获取

a. 在web.xml中配置一个监听

```
<listener>
    <listener-class>
        org.springframework.web.context.request.RequestContextListener
    </listener-class>
</listener>
```

b. 之后在程序里可以用

```
HttpServletRequest request = ((ServletRequestAttributes)
RequestContextHolder.getRequestAttributes()).getRequest();
```

# 直接作为Controller方法的入参

```
@RequestMapping(value="/testServlet")
public String testServlet(HttpServletRequest request,
    HttpServletResponse response, HttpSession session) {
    request.setAttribute("requestTest", "requestValue");
    System.out.println(response.getLocale());
    session.setAttribute("sessionTest", "sessionValue");
    return "success";
}
```

# SpringMVC Controller方法 支持的返回值类型

# SpringMVC支持的返回方式

□ Spring MVC 支持如下的返回方式：

- ModelAndView,
- Model,
- ModelMap,
- Map,
- void,
- View,
- String,

# 返回ModelAndView对象

- 通过ModelAndView构造方法可以指定返回的页面名称,也可以通过setViewName()方法跳转到指定的页面

```
@RequestMapping(value="/testRetuen")
```

```
public ModelAndView testServlet() {
```

```
    String msg = "msg";
```

```
    return new ModelAndView("success", "msg", msg);
```

```
}
```

## 返回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
- 即@RequestMapping的value属性值同样为视图名，下例最终的请求页面为success.jsp.

# 返回Model

```
@Controller
public class HelloWorldController{

    @RequestMapping(value="/success")
    public Model testServlet(Model model){
        model.addAttribute("msg", "HelloWorld!");
        System.out.println(111);
        return model;
    }
}
```



# 返回ModelMap对象

□同Model作用一样

# ■ 返回Map对象

□同Model作用一样

# 返回void

- @RequestMapping的value属性值同样为视图名，下例最终的请求页面为success.jsp.

```
@Controller
public class HelloWorldController{

    @RequestMapping(value="/success")
    public void testServlet(Model modelMap){
        modelMap.addAttribute("msg", "void, HelloWorld!");
    }
}
```

# 返回view对象

- 可用于返回EXECL表格、PDF文档等。
- @RequestMapping的value属性值同样为视图名

# 返回String对象

- 返回字符串表示一个视图名称，SpringMVC中使用最多的返回类型。

# 请求转发至视图

□ 下例当发起/testString请求，最终会以请求转发的方式跳转至success.jsp

```
@RequestMapping(value="/testString")  
public String testString(Model modelMap) {  
    modelMap.addAttribute("msg", "void,HelloWorld!");  
    return "success";  
}
```

# 重定向至视图

- 使用重定向至视图，必须以 “redirect:/path” 的方式，并且需自行加上后缀。

```
@RequestMapping(value="/testString")  
public String testString() {  
  
    return "redirect:/success.jsp";  
}
```

# 请求转发至另一Controller方法

- 使用重定向至另一Controller，必须以“forward:/url”的方式

```
@RequestMapping(value="/testString")  
public String testString(){  
    return "forward:/hello";  
}
```

```
@RequestMapping(value="/hello")  
public String test(){  
  
    return "success";  
}
```



# 重定向至另一Controller方法

- 使用重定向至另一Controller，必须以“redirect:/path”的方式

```
@RequestMapping(value="/testString")  
public String testString(){  
    return "redirect:/hello";  
}
```

```
@RequestMapping(value="/hello")  
public String test(){  
  
    return "success";  
}
```

**谢谢！**