

# SpringMVC 学习

忘了很多，还需要再次学习和复习：<http://c.biancheng.net/view/4396.html>

书籍：Spring MVC 学习指南

## 前言

- 域名：[www.baidu.com](http://www.baidu.com)，不同域名可以指向同一个 ip
- Ip：一串数字
- URL 的 host：www 是默认的 host，有无主机名的域名是不同的。但是通常 <http://www.domainName> 会被映射到 <http://domainName>
- http 的默认端口是 80 端口，对于采用 80 端口的 WEB 服务器，无需输入端口号
- localhost：保留关键字，指向本机

# Servlet

Java Servlet 是 Java 体系中用于开发 WEB 应用的底层技术。1996 年 sun 公司发布了 Servlet 和 JSP，以代替 CGI。

Servlet 是一个 Java 程序，一个 Servlet 应用包含一个或多个 Servlet，一个 JSP 页面会被翻译并编译成一个 Servlet。

一个 Servlet 运行在一个 Servlet 容器中，它无法运行，该容器将用户请求传递给 Servlet，并将 Servlet 的响应返还给用户。

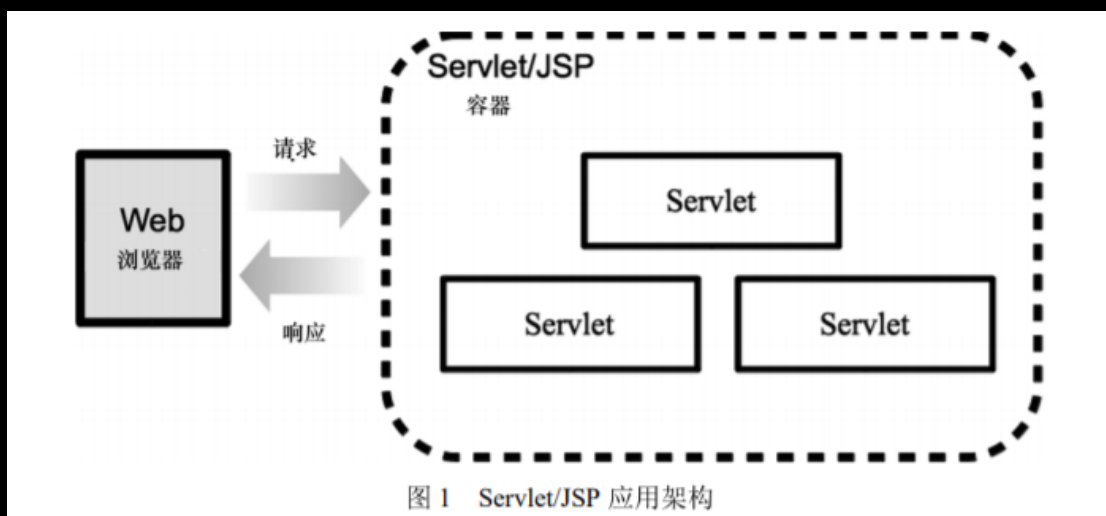


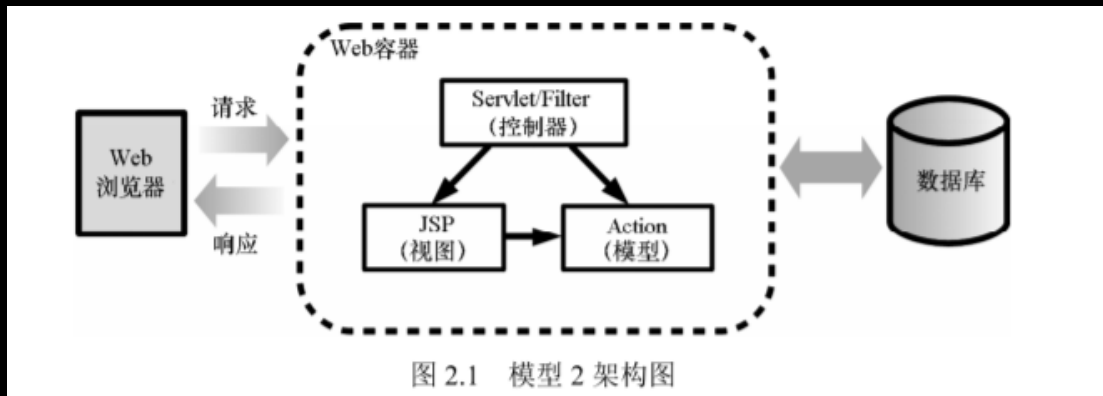
图 1 Servlet/JSP 应用架构

目前更加成熟的 Servlet/jsp 容器：Apache Tomcat、Jetty

# 第一章 · Spring 框架

Spring 管理的对象称为 bean

## 第二章·模型 2 和 MVC 模式



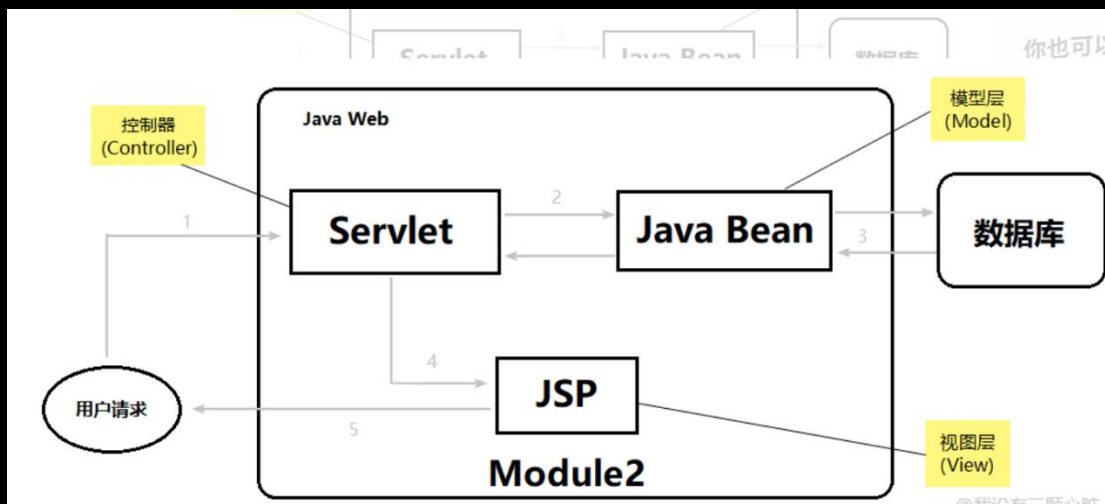
## 第三章·SpringMVC 介绍

SpringMVC 是一个包含了 DispatcherServlet 的 MVC 框架

- Spring MVC 提供了一个 Dispatcher Servlet，无需额外开发。
- Spring MVC 使用基于 XML 的配置文件，可以编辑，而无需重新编译应用程序。
- Spring MVC 实例化控制器，并根据用户输入来构造 bean。
- Spring MVC 可以自动绑定用户输入，并正确地转换数据类型。例如，Spring MVC 能自动解析字符串，并设置 float 或 decimal 类型的属性。
- Spring MVC 可以校验用户输入，若校验不通过，则重定向回输入表单。输入校验是可选的，支持编程方式以及声明方式。关于这一点，Spring MVC 内置了常见的校验器。
- Spring MVC 是 Spring 框架的一部分，可以利用 Spring 提供的其他能力。
- Spring MVC 支持国际化和本地化，支持根据用户区域显示多国语言。
- Spring MVC 支持多种视图技术。最常见的 JSP 技术以及其他技术包括 Velocity 和 FreeMarker。

### 概述

早期的 MVC 模型：



用户请求到达

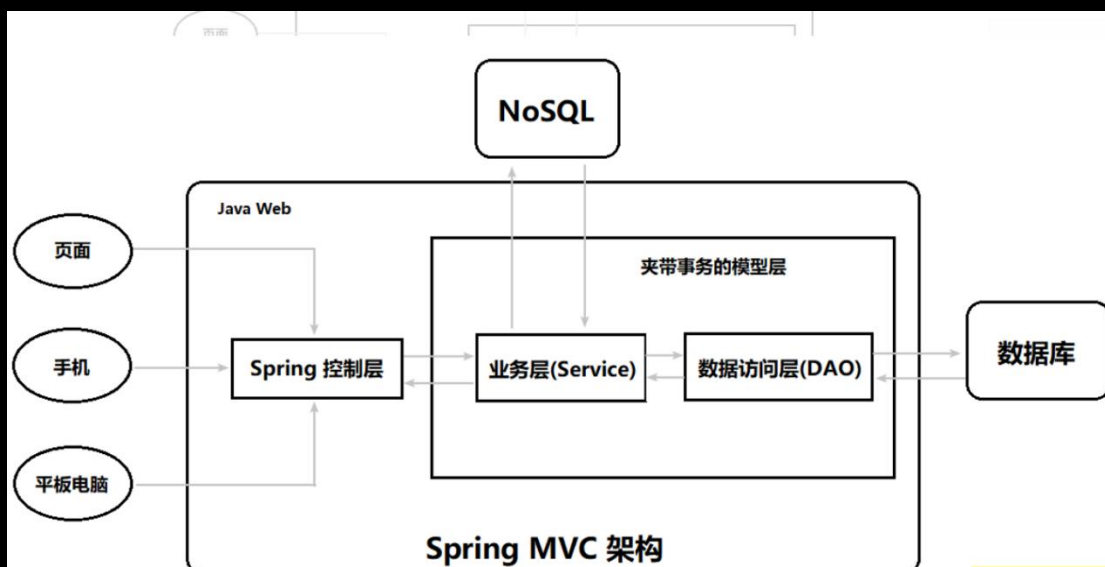
Servlet，然后根据请求调用相应的 Java bean，并把所有的 显示结果交给 jsp 去完成——这种模式就是 mvc

M: model 模型

V: view 视图

C: 控制器 controller

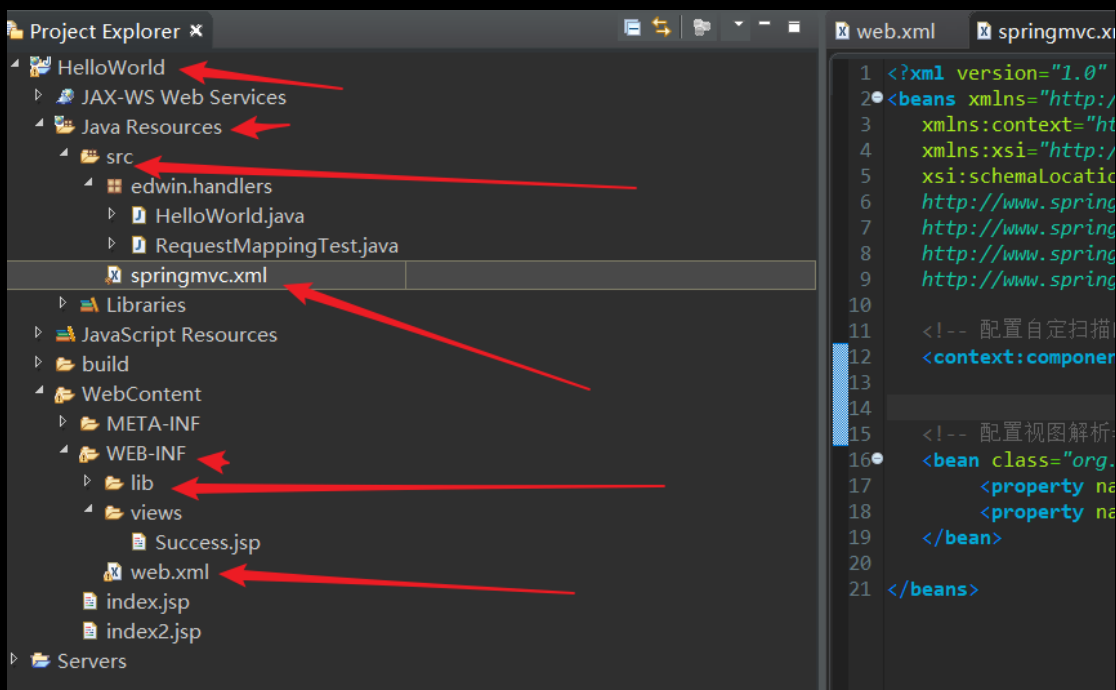
MVC 新模型：



特点：

- 结构松散，可以使用各种视图
- 松耦合，模块分离
- 与 Spring 无缝集成

项目文件结构



## DispatcherServlet

全称：org.springframework.web.servlet.DispatcherServlet

使用这个 Servlet，必须要在部署描述符即 web.xml 中使用 servlet 和 servlet-mapping 来配置它

```

<servlet>
  <servlet-name>springmvc</servlet-name>
  <servlet-class>
    org.springframework.web.servlet.DispatcherServlet
  </servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
  <servlet-name>springmvc</servlet-name>
  <url-pattern>/</url-pattern>
</servlet-mapping>

```

- Load-on-startup: 可选的，有则会在程序启动时装载 Servlet 并调用 init。不存在则会在第一个请求时装载
- SpringMVC 的 Servlet 配置文件：有 2 种方式：
  1. 在 WEB-INF 下，规定名字的 servletName-servlet.xml，如此在该文件中就不需要在 web.xml 声明 init-param 来说明地址
  2. 该配置文件可以放在任何地方，只是需要在 web.xml 中使用 init-param 标签来声明文件地址

```

<init-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>/WEB-INF/config/simple-config.xml</param-value>
</init-param>

```

(技巧：在 web.xml 中输入 dispatcherservlet 会有自动提示弹出框架)

## 基于注解的控制器

传统的控制器风格：实现 Controller 接口

## Controller 注解

Org.springframework.stereotype.Controller

Spring 使用扫描机制来找到注解的类，所以需要在 SpringMVC 配置文件中声明 spring-context:

```

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

```

然后还需要使用<component-scan/>

```

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

```

Base-package 包及其子包都会被扫描

# RequestMapping 注解

需要在控制类内部为每一个动作开发相应的处理方法

Org.springframework.web.bind.annotation.RequestMapping

- Value 属性：默认，唯一属性时可以省略
- Method 属性：指示该方法仅处理那些 HTTP 方法，使用 RequestMethod 静态类可以访问 HTTP 方法：同时可以有多个方法，使用 {}：

```
@RequestMapping(value="/process-order",  
                method={RequestMethod.POST, RequestMethod.PUT})
```

注解类时，对所有的方法都映射为该请求

- Params 参数：  
通过 “para=value” ”para!=value”等方式指定参数值  
通过 ! name 指定参数中不能有 Name 的

## 编写处理请求的方法

可以为处理方法添加不同的 HTTP 请求参数

```
@RequestMapping("/uri")  
public String myOtherMethod(HttpServletRequest request, HttpSR  
    Locale locale) {  
    ...  
    // access Locale and HttpServletRequest here  
    ...  
}
```

 **客户端语言环境**

一些参数：

- javax.servlet.**ServletRequest** 或 javax.servlet.http.HttpServletRequest。
- javax.servlet.ServletResponse 或 javax.servlet.http.**HttpServletResponse**。
- javax.servlet.http.**HttpSession**。
- org.springframework.web.context.request.**WebRequest** 或 org.springframework.web.context.request.NativeWebRequest。

- java.util.Locale。
- java.io.InputStream 或 java.io.Reader。
- java.io.OutputStream 或 java.io.Writer。
- java.security.Principal。
- HttpEntity<?>parameters
- java.util.Map/org.springframework.ui.Model /。
- org.springframework.ui.ModelMap。
- org.springframework.web.servlet.mvc.support.RedirectAttributes。
- org.springframework.validation.Errors /。
- org.springframework.validation.BindingResult。
- 命令或表单对象。
- org.springframework.web.bind.support.SessionStatus。
- org.springframework.web.util.UriComponentsBuilder。
- 带@PathVariable、@MatrixVariable、@RequestParam、@RequestHeader、@RequestBody 或@RequestPart 注释的对象。

### 返回值：

- 提供对 Servlet 的访问，以响应 HTTP 头部和内容 HttpEntity 或 ResponseEntity 对象。
- Callable。
- DeferredResult。
- 其他任意类型，Spring 将其视作输出给 View 的对象模型。

- ModelAndView。
- Model。
- 包含模型的属性的 Map。
- View。
- 代表逻辑视图名的 String。
- void。

## 请求参数 路径变量

请求参数与路径变量用于发送值给服务器，他们是 URL 的一部分

### 请求参数

格式为 key=value，使用&分隔

获取方法：



- 传统：使用 `HttpServletRequest` 的 `getParameter` 方法
- SpringMVC：  
使用 `org.springframework.web.bind.annotation.RequestParam` 注解类型来注解这个方法参数

```
public void sendProduct(@RequestParam int productId)
```

## 路径变量

为了使用路径变量，首先需要在 `RequestMapping` 注解值中添加一个变量，且把它放到 `{}` 中：

```
@RequestMapping(value = "/view-product/{id}")
```

然后在方法签名中添加一个同名变量，使用 `@PathVariable` 注解

```
@RequestMapping(value = "/view-product/{id}")
public String viewProduct(@PathVariable Long id, Model model) {
    Product product = productService.get(id);
    model.addAttribute("product", product);
    return "ProductView";
}
```

## @ModelAttribute

每次调用请求处理方法时，都会创建 `Model` 类型的一个实例。若打算使用该实例，则可以在方法中添加一个 `Model` 类型的参数，或添加注解。

使用 `@ModelAttribute` 注解的方法会将其输入或创建的参数对象添加到 `Model` 对象中。

## Ant 风格

\*: 表示任意一个目录    eg: `FolderA/*/file`

\*\*：表示多个目录      eg:

? : 任意单个字符

## @PathVariable 动态获取参数

```
@RequestMapping(value="welcome5/{name}")
public String welcome5(@PathVariable("name") String name ) {
    System.out.println(name);
}
```

# 项目理解

构建项目注意事项————五个文件的关系及内容

## 1.request.jsp:

触发请求的 JSP 文件在 WebContent 下,不能在 WebContent 的子文件夹或者是父文件夹,不知道为什么

另外里面应该有请求发出,即应该有超链接、表单提交等

超链接: href="RequestName",

表单提交请求: action = "\*.jsp" 实际上也可以是一个 RequestName,同超链接

## 2.web.xml:

必须在 WebContent/WEB-INF 下,是默认配置文件

作用:配置 Servlet 容器:声明 Servlet 名称、初始化、应答的请求类型等

```
<web-app config-info>
    <servlet>
        <servlet-name>name</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-
class>

        <load-on-startup>1</load-on-startup>
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>classpath:or other</param-name>
        </init-param>
    </servlet>
    <servlet-mapping>
        <servlet-name>name<servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
</web-app>
```

## 3.主要类

使用@Controller 和@RequestMapping 注解

/\*

\* 1.使用@RequestMapping 注解来映射请求的 URL

\* 2.返回值会通过视图解析器解析为实际的物理视图,对于

InternalResourceViewResolver 视图解析器，会做如下解析：

\* 通过 prefix+ReturnValue+suffix 这样的方式得到实际的物理视图，然后做转发操作

\* /WEB-INF/views/ReturnValue.jsp

\*/

@Controller

public class ClassName {

@RequestMapping("/RequestName") //这个和请求 jsp 中的 href 是一致的，无/也行

public type method() {

---

return "ReturnValue";

}

#### 4.Spring 配置 XML

<!-- 配置自定扫描的包 -->

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

<!-- 配置视图解析器：如何把方法返回值解析为实际的物理视图 -->

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

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

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

</bean>

#### 5.物理视图 JSP:

视图解析器已经说明了物理视图表现的 JSP 文件地址：prefix+ReturnValue+suffix, 即/WEB-INF/views/ReturnValue.jsp

在该文件中声明即可

web.xml 快捷键：

Alt+/  
d

即可生成 DispatcherServlet

# <url-pattern>/</url-pattern>

表示拦截所有

如果是/user

那么就拦截 user 开头的，eg: /user/a

使用.:

.a: 只拦截以.a 结尾的

## 第五章 • 数据绑定和表单标签库

### 数据绑定

数据绑定：将用户输入绑定到领域模型的一种特性。

HTTP 请求参数的类型均为字符串,所以需要将这类型转化为需要的类型。

数据绑定的一个好处是：当验证失败时，会重新生成一个 HTML 表单。

### 表单标签库

表单标签库包含了可以用在 jsp 页面渲染 HTML 元素的标签，在 jsp 文件开头声明 taglib 指令：

```
<%@taglib prefix="form"
    uri="http://www.springframework.org/tags/form" %>
```

表 5.1 表单标签库中的标签

标签	描述
form	渲染表单元素
input	渲染<input type="text"/>元素
password	渲染<input type="password"/>元素
hidden	渲染<input type="hidden"/>元素
textarea	渲染 textarea 元素
checkbox	渲染一个<input type="checkbox"/>元素
checkboxes	渲染多个<input type="checkbox"/>元素
radiobutton	渲染一个<input type="radio"/>元素
radiobuttons	渲染多个<input type="radio"/>元素
select	渲染一个选择元素
option	渲染一个可选元素
options	渲染一个可选元素列表
errors	在 span 元素中渲染字段错误

表 5.2 表单标签的属性

属性	描述
acceptCharset	定义服务器接受的字符编码列表
commandName	暴露表单对象之模型属性的名称，默认为 command
cssClass	定义要应用到被渲染 form 元素的 CSS 类
cssStyle	定义要应用到被渲染 form 元素的 CSS 样式
htmlEscape	接受 true 或者 false，表示被渲染的值是否应该进行 HTML 转义
modelAttribute	暴露表单支持对象的模型属性名称，默认为 command

注意：

要使用渲染一个表单输入字段的任何其他标签属性，必须有一个 form 标签：

```
<form:form commandName="book" action="save-book" method="post">
    ...
</form:form>
```

表单标签的属性：

表 5.2 表单标签的属性

属性	描述
acceptCharset	定义服务器接受的字符编码列表
commandName	暴露表单对象之模型属性的名称，默认为 command
cssClass	定义要应用到被渲染 form 元素的 CSS 类
cssStyle	定义要应用到被渲染 form 元素的 CSS 样式
htmlEscape	接受 true 或者 false，表示被渲染的值是否应该进行 HTML 转义
modelAttribute	暴露表单支持对象的模型属性名称，默认为 command

Input 标签

Input 标签渲染<input type="text"/>元素，这个标签最重要的属性是 path，它将这个输入字段绑定到表单支持对象的一个属性。如：

如果<form/>标签的 commandName 属性值为 book, 并且input 标签的 path 值为 ISBN, 那么 input 标签将被绑定到 Book 对象的 ISBN 属性。

表 5.3 input 标签的属性

属性	描述
cssClass	定义要应用到被渲染 input 元素的 CSS 类
cssStyle	定义要应用到被渲染 input 元素的 CSS 样式
cssErrorClass	定义要应用到被渲染 input 元素的 CSS 类，如果 bound 属性中包含错误，则覆盖 cssClass 属性值
htmlEscape	接受 true 或者 false，表示是否应该对被渲染的值进行 HTML 转义
path	要绑定的属性路径

在 web.xml 中添加过滤器可以防止中文乱码问题:

```
<!-- 避免中文乱码 -->
<filter>
    <filter-name>characterEncodingFilter</filter-name>
    <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
    <init-param>
        <param-name>encoding</param-name>
        <param-value>UTF-8</param-value>
    </init-param>
    <init-param>
        <param-name>forceEncoding</param-name>
        <param-value>true</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>characterEncodingFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
</web-app>
```

## 绑定表单数据

JSP 页面使用 Spring 提供的标签需要引入相关 taglib:

<%@taglib prefix="form" uri="http://www.springframework.org/tags/form" %>

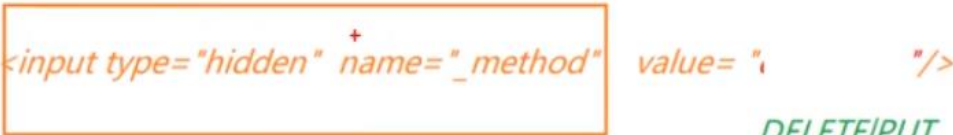
## REST 风格

Springmvc:

- GET: 查
- POST: 增
- DELETE: 删
- PUT: 改

普通浏览器只支持 GET、PUT，其他两个是通过过滤器（HiddenHttpMethodFilter）实现的

过滤的条件:

1.   
`<input type="hidden" name="_method" value="DELETE/PUT" />`
2. 请求方式为post

实现步骤:

- 增加过滤器:

```
<!-- 增加HiddenHttpMethodFilter过滤器: 目的是给普通浏览器 增加 put|delete请求方式 -->
<filter>
    <filter-name>HiddenHttpMethodFilter</filter-name>
    <filter-class>org.springframework.web.filter.HiddenHttpMethodFilter</filter-class>
</filter>

<filter-mapping>
    <filter-name>HiddenHttpMethodFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

- 表单

```
<form action="handler/testRest/1234" method="post">
    <input type="hidden" name="_method" value="DELETE"/>
    <input type="submit" value="删">
</form>
```

- 控制器

通过 `method=RequestMethod.DELETE` 匹配具体的请求方式

## @RequestParam

使用在方法的参数中

```
@RequestParam("uname") String name, @RequestParam(value="uage", required=false, defaultValue="23")
@RequestParam("uname"): 接受前台传递的值, 等价于 request.getParameter("uname");
required=false: 该属性 不是必须的。
defaultValue="23": 默认值23
```

## @RequestHeader

获取 HTTP 的请求头中的变量

```
获取请求头信息 @RequestHeader
public String testRequestHeader(@RequestHeader("Accept-Language") String al ) {
    通过@RequestHeader("Accept-Language") String al 获取请求头中的Accept-Language值, 并将值保存再al
}
```

## @CookieValue

获取 cookie 数据

```
通过mvc获取cookie值 (JSESSIONID)
@CookieValue
(前置知识: 服务端在接受客户端第一次请求时, 会给该客户端分配一个session (该session包含一个sessionID))
```

# Springmvc 处理参数流程



- 请求：前端发送请求 req ——> @RequestMapping("req")
- 处理参数中的请求：

```
@RequestMapping("a")
public String aa(@Xxx注解("xyz") xyz)
{
}
```

注解后面的变量用于存储注解获取的值

## 使用对象接收请求参数

Spring 提交表单后自动传递给一个实例，  
对象的属性必须与表单中的 Name 值一致

```
@RequestMapping("submitIt")
//这里必须要使用Student实例，但是仍可以添加Model.
//特别注意：填写的表单类型一定要符合类的属性类型，比如id必须填写数字，否则报错
public String print(Student student, Model model) {
    model.addAttribute("s", student);
    System.out.println("ID: "+student.getId() + "    "+"NAME: "+student.getName());
    return "Mapping";
}
```

当然，也可以使用原生态的 HttpServletRequest 类

```
public String testServletAPI(HttpServletRequest request, HttpServletResponse res)
```

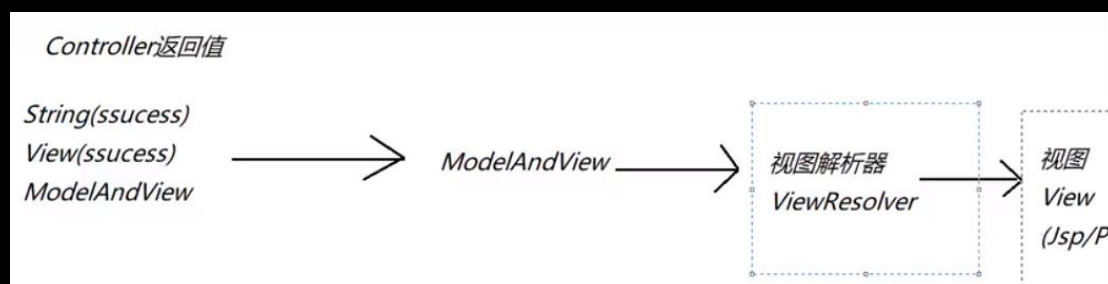
## Controller 到 view

Model：模型，往往带有许多的数据

从 Controller 到 View，往往需要带着数据显示出来。

需要连带数据，则可以使用 ModelAndView, ModelMap, Map, Model  
这些方式最终都会转化为 ModelAndView

— 这些数据是放在 **request 作用域**



ModelAndView

可以将数据绑定到 ModelAndView，然后返回到 Request 域。使用 JSP 等可以取用。



```

@RequestMapping("/request-1")
public ModelAndView method1(){
    // 参数仍然是文件名。
    ModelAndView modelAndView = new ModelAndView("response-1");
    Student edwin = new Student();
    edwin.setId(171250579);
    edwin.setName("Edwin Xu");

    modelAndView.addObject("edwin", edwin);
    // 这条语句会将一个学生数据添加到modelAndView, 并添加到request作用域
    // 其实就相当于Request.setAttribute()
    return modelAndView;
}

```

使用 EL 可以输出数据信息:

```

<h2>${requestScope.edwin.id}---${requestScope.edwin.name}</h2>

```

## ModelMap

```

@RequestMapping(value="testModelAndView")
public String testModelAndView(ModelMap mm) { //success

    Student student = new Student();
    student.setId(2);
    student.setName("zs");

    mm.put("student", student); //request域

    return "success"; //view
}

```

Model 和 Map 也是一样的，都会自动放到 request 域里。

## 加入 Session

使用注解: @SessionAttributes(value=属性名)

这个属性名是传到 request 的属性名。

@SessionAttributes(types=ClassName.class)——使用一个类型

## @ModelAttribute

经常用于更新

修改之前先查询。

```

java 8 index.jsp Student.java
// 快速构建二、三层网站的设计
Student student = new Student();
student.setId(31);
student.setName("zs");
student.setAge(23);
map.put("student", student); // 约定: map的key 就是方法参数 类型的首字母大写
map.put("stu", student); // 约定: map的key 就是方法参数 类型的首字母小写
}

// 修改: Zs-ls
@RequestMapping(value="testModelAttribute")
public String testModelAttribute(@ModelAttribute("stu") Student student)
{
    student.setName("ls"); // 将名字修改为ls
    System.out.println(student.getId()+" "+student.getName()+" "+student.getName());
    return "success";
}

```

@ModelAttribute 经常在 更新时使用  
 通过@ModelAttribute修饰的方法，会在每次请求前先执行；  
 并且该方法参数map.put()可以将 对象 放入 即将查询的参数中；  
 必须满足的约定：  
 map.put(k, v) 其中的k 必须是即将查询的方法参数 的首字母小写  
 testModelAttribute(Student xxx)，即student；  
 如果不一致，需要通过@ModelAttribute声明

慎用，易出错，其他方法易受影响。

```

@ModelAttribute//在任何一次请求前，都会先执行@ModelAttribute修饰的方法
//@ModelAttribute 在请求 该类的各个方法前 均被执行的设置是基于
public void queryStudentById(Map<String, Object> map) {

```

每个方法执行之前都会执行该方法。

```

@ModelAttribute
public void method_modelAttribute(Map<String, Object> map){
    Student yuanteng = new Student();
    yuanteng.setId(2);
    yuanteng.setName("YuanTeng Xu");
    map.put("xuyuanteng", yuanteng); // ModelAttribute注解的方法不需要返回值，不过需要提前加入你后面需要更新的信息。
}

@RequestMapping("/changeInfo")
public String method_modelAttribute_test(@ModelAttribute("xuyuanteng") Student teng) {
    // 使用@ModelAttribute来查询需要修改的值，然后取出更新。
    teng.setId(222222);
    teng.setName("I'am XuYuanTeng");
    return "response=1"; // 更新后的返回给视图解析器
}

```

## 视图&视图解析

视图类型		简介
URL 视图资源图	<a href="#">InternalResourceView</a>	将 JSP 或其他资源封装成一个视图。被视图解析器 <a href="#">InternalResourceViewResolver</a> 默认使用。
	<a href="#">JstlView</a>	<a href="#">InternalResourceView</a> 的子类。 如果 JSP 中使用了 JSTL 的国际化标签，就需要使用该视图类。
文档视图	<a href="#">AbstractExcelView</a>	Excel 文档视图的抽象类。
	<a href="#">AbstractPdfView</a>	PDF 文档视图的抽象类
报表视图	<a href="#">ConfigurableJasperReportsView</a>	常用的 <a href="#">JasperReports</a> 报表视图
	<a href="#">JasperReportsHtmlView</a>	
	<a href="#">JasperReportsPdfView</a>	
	<a href="#">JasperReportsXlsView</a>	
JSON 视图	<a href="#">MappingJackson2JsonView</a>	将数据通过 Jackson 框架的 <a href="#">ObjectMapper</a> 对象，以 JSON 方式输出

视图顶级接口：View

常见视图：InternalResourceView

视图解析器类型		简介
解析为 bean	<a href="#">BeanNameViewResolver</a>	将视图解析后，映射成一个 bean，视图的名字就是 bean 的 id。
解析为映射文件	<a href="#">InternalResourceViewResolver</a>	将视图解析后，映射成一个资源文件。例如将一个视图名为字符串“success.jsp”的视图解析后，映射成一个名为 success 的 JSP 文件。
	<a href="#">JasperReportsViewResolver</a>	将视图解析后，映射成一个报表文件。
解析为模板文件	<a href="#">FreeMarkerViewResolver</a>	将视图解析后，映射成一个 <a href="#">FreeMarker</a> 模板文件。 <a href="#">FreeMarker</a> 是一款模板引擎，用法可以参见 <a href="http://jiangsha.iteye.com/blog/372307">http://jiangsha.iteye.com/blog/372307</a> 。
	<a href="#">VelocityViewResolver</a>	将视图解析后，映射成一个 Velocity 模板文件。
	<a href="#">VelocityLayoutViewResolver</a>	

视图解析器的顶级接口：ViewResolver

常见视图解析器：InternalResourceViewResolver

JstlView

JstlView 是 InternalResourceView 的子类，可以解析 jstl（JavaServer Pages Standard Tag Library，JSP 标准标签库），实现国际化操作。

SpringMVC 一般会使用父类 InternalResourceView，但是如果语言中包含 JSTL 语言，会自动使用子类 JstlView

## 国际化

实现步骤：

- 第一步：创建资源文件

## 常见的资源文件命名

资源文件名	简介
基名_en.properties	所有英文语言的资源
基名_en_US.properties	针对美国地区、英文语言的资源
基名_zh.properties	所有的中文语言的资源
基名_zh_CN.properties	针对中国大陆的、中文语言的资源
基名_zh_HK.properties	针对中国香港的、中文语言的资源
基名.properties	默认资源文件。如果请求相应语言的资源文件不存在，将使用此资源文件。例如，若是中国大陆地区用户，应该访问“基名_zh_CN.properties”，而如果不存在此文件，就会去访问默认的“基名.properties”。

### ■ 资源文件格式：

基名\_语言\_地区.properties

基名\_语言.properties

### ■ 补充：

i18n, internationalization 的首末字符 i 和 n, 18 为中间的字符数, “国际化”的简称

L10n: localization 的缩写形式, 意即在 l 和 n 之间有 10 个字母, 本意是指软件的“本地化”)

■ 可以创建一个 i18n.properties, 如果其他资源文件中找不到, 则在该资源文件中找。

■ 资源文件全是 Unicode: \F34F

■ i18n\_en\_US.properties

■ i18n\_zh\_CN.properties

```
1 resource.welcome=\u6B22\u8FCE
2 resource.hello=\u4F60\u597D
3
```

```
1 resource.welcome=WELCOME
2 resource.hello=HELLO
```

## ● 第二步：配置 springmvc.xml

```
<!-- 加载国际化资源文件:
1.ResourceBundleMessageSource会在程序启动时自动加载到SpringMVC: SpringMVC在启动时会自动查找
id="messageSource",如果有, 自动加入
-->
<bean id="messageSource" class="org.springframework.context.support.ResourceBundleMessageSource">
  <!-- id必须是messageSource -->
  <property name="basename" value="classpath:i18n"></property>
  <!-- 自动寻找基名为i18n的文件 -->
</bean>
```

## ● 第三步：使用

■ 添加包: jstl.jar standard.jar

■ 在视图 jsp 中引入库:

<%@taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt"%>

```
<fmt:message key = "resource.welcome"></fmt:message>
<br>
<fmt:message key = "resource.hello"></fmt:message>
```

## request→response

最常用的 mvc 模式：request→controller→response

但是还可以使用：request→response

需要在 SpringMVC 的配置文件中使用：

```
<mvc:view-controller path="" view-name="">
```

Path:请求的路径，相当于 RequestMapping 中的 value

View-name: 视图文件或路径

同时还需要加入名称空间：

```
xmlns:mvc="http://www.springframework.org/schema/mvc"
```

```
xsi:schemaLocation=" http://www.springframework.org/schema/mvc
```

```
http://www.springframework.org/schema/mvc/spring-mvc.xsd"
```

注意：

- 如果使用了 mvc:controller，则会忽略掉所有的@Controller 注解
- 如果要想@Controller 与 mvc 共存，则需要在 springmvc 配置文件中添加一个 mvc 基础配置，极其强大：

```
<!--此配置是SpringMVC的基础配置，很功能都需要通过该注解来协调 -->
<mvc:annotation-driven></mvc:annotation-driven>
```

## 指定跳转方式

### 请求转发 forward

服务器请求资源，访问目标 URL 后，读取内容，然后把内容发给浏览器显示出来，浏览器不知道内容来自哪里，客服端的地址栏地址还是原来的地址。（服务器行为）

### 重定向 redirect

服务器发送一个状态码给浏览器，浏览器去请求那个地址，所以地址栏是变化了的。（客服端行为）

Springmvc 中也可以使用，在 @RequestMapping 中返回的视图渲染文件名中添加 forward:、redirect:。

注意：使用重定向或请求转发会导致 springmvc 配置文件的前后缀 prefix、suffix 失效。所以这里地址一定要全面。



```
return "forward:/views/success.jsp";
```

## 处理静态资源

静态资源：HTML、css、js、图片等不会因为时间地点改变的资源。

在 springmvc 中，如果直接在地址栏访问静态资源，会 404，因为请求会被 DispatcherServlet 拦截

解决方式：

1. 需要 mvc: 使用 @RequestMapping 处理
2. 不需要 mvc: 使用 Tomcat 默认的 Servlet 处理
  - a) 如果配置了对应的 servlet，那么使用其处理

```
<servlet>
    <servlet-name>abc</servlet-name>
    <servlet-class>xxx.xxx.xx.ABCServlet</servlet-class>
</servlet>

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

- b) 否则，默认直接访问。

即只需要在 springmvc 配置文件文件中添加 2 句：

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

```
<mvc:default-servlet-handler> </mvc:default-servlet-handler>
```

所以以后都把<mvc:annotation-driven>（协调作用）  
和<mvc:default-servlet-handler>添加进去吧。

## 类型转换

Springmvc 自带一些基本的类型转换器如 @PathVariable

自定义类型转换器：

- 编写类：必须实现 Converter 接口
- 实现转换方法。
- 使用泛型: <Source, Target>

```

public class MyConverter implements Converter<String,Student>{

    @Override
    public Student convert(String source) { //source:2-zs-23
        //source接受前端传来的String:2-zs-23
        String[] studentStrArr = source.split("-");
        Student student = new Student();
        student.setId( Integer.parseInt( studentStrArr[0]) );
        student.setName(studentStrArr[1]);
        student.setAge(Integer.parseInt(studentStrArr[2] ));
        return student;
    }
}

```

- 讲将编写的类加入 springmvc (配置文件)
  - a) 将自定义转换器纳入 SpringIOC 容器
  - b) 将转换器再纳入 springmvc 提供的转换器 Bean
  - c) 注册到 annotation-driven

```

<!-- 1将自定义转换器 纳入SpringIOC容器 -->
<bean id="myConverter" class="org.lanqiao.converter.MyConverter"></bean>

<!-- 2将myConverter再纳入 SpringMVC提供的转换器Bean -->
<bean id="conversionService" class="org.springframework.context.support.Conv
    <property name="converters">
        <set>
            <ref bean="myConverter"/>
        </set>
    </property>
</bean>

<!-- 3将conversionService注册到annotation-driven中 -->
<!--此配置是SpringMVC的基础配置，很功能都需要通过该注解来协调 -->
<mvc:annotation-driven conversion-service="conversionService"></mvc:annotation-driven>

```

```

@RequestMapping(value="testConverter")
public String testConverter(@RequestParam("studentInfo") Student student) { //
    System.out.println(student.getId()+","+student.getName()+","+student.ge
    return "success";
}

```

## 数据格式化

Springmvc 提供注解来实现数据格式化

- 配置数据格式化所需要的 bean

```

<bean id="conversionService" class="org.springframework.format.support.FormattingConverters
</bean>

```

- 注解需要格式化的属性: @DateTimeFormat pattern 表示形式

```
public class Student {
    private int id;
    private String name;
    private int age;

    @DateTimeFormat(pattern="yyyy-MM-dd")
    private Date birthday ;// 2018-12-13
}
```

## 数字格式化

@NumberFormat(pattern = "###.###")

```
public class Student {
    @NumberFormat(pattern="###, #")
    private int id;
}
```

拓展:

```
<!-- 配置 数据格式化 注解 所依赖的bean
FormattingConversionServiceFactoryBean:既可以实现格式化、又可以实现类型转换
-->
<bean id="conversionService" class="org.springframework.format.support.Format
    <property name="converters">
        <set>
            <ref bean="myConverter"/>
        </set>
    </property>
</bean>
```

## 打印错误

```
@RequestMapping(value="testDateTimeFormat")//如果Student格式化出错,会将错
public String testDateTimeFormat(Student student, BindingResult result)
    如果第一个参数发生错误,会错误传给BindingResult
```

可以再添加 Map, 放到 request 域中, 前段再取出显示。

BindingResult 必须紧跟校验项的后面。

前段若取到的是集合, 需要使用 JSTL 中的

<c:forEach>

```
<c:forEach items="${requestScope.errors}" var="error">
    ${error.getDefaultMessage()}、|
</c:forEach>
```

需要加顿号



# 其他校验方法

## JSR 303——专业校验

<code>@Null</code>	被注释的元素必须为 <code>null</code> 。
<code>@NotNull</code>	被注释的元素必须不为 <code>null</code> 。
<code>@AssertTrue</code>	被注释的元素必须为 <code>true</code> 。
<code>@AssertFalse</code>	被注释的元素必须为 <code>false</code> 。
<code>@Min(value)</code>	被注释的元素必须是一个数字，其值必须大于或等于 <code>value</code> 。
<code>@Max(value)</code>	被注释的元素必须是一个数字，其值必须小于或等于 <code>value</code> 。
<code>@DecimalMin(value)</code>	被注释的元素必须是一个数字，其值必须大于或等于 <code>value</code> 。
<code>@DecimalMax(value)</code>	被注释的元素必须是一个数字，其值必须小于或等于 <code>value</code> 。

<code>@Size(max, min)</code>	被注释的元素的取值范围必须是介于 <code>min</code> 和 <code>max</code> 之间。
<code>@Digits(integer, fraction)</code>	被注释的元素必须是一个数字，其值必须在可接受的范围内。
<code>@Past</code>	被注释的元素必须是一个过去的日期。
<code>@Future</code>	被注释的元素必须是一个将来的日期。
<code>@Pattern(value)</code>	被注释的元素必须符合指定的正则表达式。

## Hibernate Validator

### JSR 的扩展

<code>@Email</code>	被注释的元素值必须是合法的电子邮箱地址。
<code>@Length</code>	被注释的字符串的长度必须在指定的范围内。
<code>@NotEmpty</code>	被注释的字符串的必须非空。

<code>@Range</code>	被注释的元素必须在合适的范围内。
---------------------	------------------

以下是对hibernate-validator中部分注解进行描述:

@AssertTrue 用于boolean字段, 该字段只能为true  
@AssertFalse 该字段的值只能为false  
@CreditCardNumber 对信用卡号进行一个大致的验证  
@DecimalMax 只能小于或等于该值  
@DecimalMin 只能大于或等于该值  
@Digits(integer=,fraction=) 检查是否是一种数字的整数、分数, 小数位数的数字  
@Email 检查是否是一个有效的email地址  
@Future 检查该字段的日期是否是属于将来的日期  
@Length(min=,max=) 检查所属的字段长度是否在min和max之间, 只能用于字符串  
@Max 该字段的值只能小于或等于该值  
@Min 该字段的值只能大于或等于该值  
@NotNull 不能为null  
@NotBlank 不能为空, 检查时会忽略空格  
@NotEmpty 不能为空, 这里的空是指空字符串  
@Null 检查该字段为空  
@Past 检查该字段的日期是否是在过去  
@Pattern(regex=,flag=) 被注释的元素必须符合指定的正则表达式  
@Range(min=,max=,message=) 被注释的元素必须在合适的范围内  
@Size(min=, max=) 检查该字段的size是否在min和max之间, 可以是字符串、数组、集合、Map等  
@URL(protocol=,host,port) 检查是否是一个有效的URL, 如果提供了protocol, host等, 则该URL还需满足提供的条件  
@Valid 该注解主要用于字段为一个包含其他对象的集合或map或数组的字段, 或该字段直接为一个其他对象的引用, 这样在检查当前对象的同时也会检查该字段所引用的对象

使用步骤:

1. +包:

```
hibernate-validator.jar  
classmate.jar  
job-logging.jar  
validation-api.jar  
hibernate-validator-annotation-processor.jar
```

2. <mvc:annotation-driven>

(如果自定义校验类, 需要实现 ValidatorFactory, 不过 springmvc 已经帮我们实现了一个: LocalValidatorFactoryBean)

<mvc:annotation-driven>的作用就是自动加载

LocalValidatorFactoryBean

3. 使用注解

注意: 对于需要校验的对象, 使用注解: @Valid

```
@RequestMapping(value="testDateTimeFormat")//如果Student格式化出错, 会将错  
public String testDateTimeFormat(@Valid Student student, BindingResult r
```

使用注解 Valid 后, 就会自动去校验注解后面的那个类型。

# Spring 视频学习

## HelloWorld

步骤：

1. 加入 jar 包
2. 在啊 Web.xml 中配置 DispatcherServlet
3. 加入 SpringMVC 配置文件
4. 编写处理请求的处理器，并标识为处理器
5. 编写视图

具体步骤（Eclipse）：

1. 如果新建工程里没有 web 选项：help-install new software – Web tool
2. 新建一个 web-Dynamic web project
3. 在 WEB-INF 下的 lib 下添加包
4. WEB-INF 下配置 web.xml

## 使用 @RequestMapping 映射请求

SpringMVC 使用 @RequestMapping 注解为控制器指定可以处理那些 URL 请求：

@RequestMapping（URL）

在控制器的类定义 及 方法定义处都可以标注

- 类定义处：提供初步的请求映射信息。相对于 WEB 应用的根目录
- 方法处：提供进一步的细分映射信息。相对于类定义处的 URL。若类定义处未标注 @RequestMapping，则方法处标记的 URL 相对于 WEB 应用的根目录

DispatcherServlet 截获请求后，通过控制器上 @RequestMapping 提供的映射信息确定求情所对应的处理方法

HTTP 请求

## • 标准的 HTTP 请求报头



@RequestMapping 还可以使用:

Value: 请求 URL, 必须包含

Method: 请求方法

Params: 请求参数

Heads: 请求头

Params 和 heads 支持简单的表达式:

1. Params\_1: 必须包含名为 Params\_1 的参数
2. ! Params\_1: 不能包含名为 Params\_1 的参数
3. Params\_1 != value-1: 包含名为 Params\_1 的参数, 但是其值不为 value-1
4. {"para1=value1","para2"}: 包含两个参数, 但是值有限制

```
@RequestMapping(value = "/rm2", method = RequestMethod.POST)
public void test2() {
    System.out.println("mapping with paras...");
}
```

## 使用 @RequestMapping 映射请求

### • Ant 风格资源地址支持 3 种匹配符:

- ?: 匹配文件名中的一个字符
- \*: 匹配文件名中的任意字符
- \*\*: \*\* 匹配多层路径

### • @RequestMapping 还支持 Ant 风格的 URL:

- /user/\*/createUser: 匹配 /user/aaa/createUser、/user/bbb/createUser 等 URL
- /user/\*\*/createUser: 匹配 /user/createUser、/user/aaa/bbb/createUser 等 URL
- /user/createUser?: 匹配 /user/createUseraa、/user/createUserbb 等 URL

## @PathVariable 映射 URL 绑定的占位符

- 带占位符的 URL 是 Spring3.0 新增的功能，该功能在 SpringMVC 向 REST 目标挺进发展过程中具有里程碑的意义
- 通过 @PathVariable 可以将 URL 中占位符参数绑定到控制器处理方法的入参中：URL 中的 {xxx} 占位符可以通过 @PathVariable("xxx") 绑定到操作方法的入参中。

```
@RequestMapping("/delete/{id}")
public String delete(@PathVariable("id") Integer id){
    UserDao.delete(id);
    return "redirect:/user/list.action";
}
```

# 孙浩 springMVC

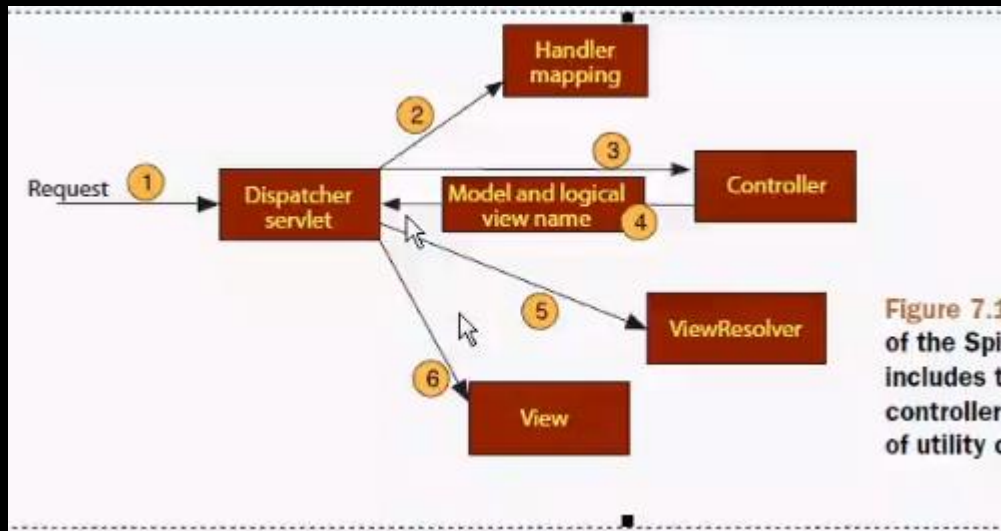
## MVC 学习步骤

- 1.搭建环境
- 2.如何完成 Controller 和 view 的映射
- 3.如何把值传给 Controller
- 4.Controller 如何把值传递给 view
- 5.异常处理
- 6.页面标签



- 7.文件上传
- 8.深入源代码

## 原理



1. 在 web.xml 中启动 DispatcherServlet
2. 在 WEB-INF 中创建 servletName-servlet.xml, 配置 Spring

构建项目注意事项——五个文件的关系及内容

### 1.request.jsp:

触发请求的JSP文件在WebContent下,不能在WebContent的子文件夹或者是父文件夹,不知道为什么  
另外里面应该有请求发出,即应该有超链接、表单提交等  
超链接: href="RequestName",  
表单提交请求: action = "\*.jsp" 实际上也可以是一个RequestName,同超链接

### 2.web.xml:

必须在WebContent/WEB-INF下,是默认 配置文件  
作用: 配置Servlet容器: 声明Servlet名称、初始化、应答的请求类型等

```
<web-app config-info>
  <servlet>
    <servlet-name>name</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
    <init-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>classpath:or other</param-value>
    </init-param>
  </servlet>
</web-app>
```

### 3.主要类

使用@Controller和@RequestMapping注解

/\*

\* 1.使用@RequestMapping注解来映射请求的URL

\* 2.返回值会通过视图解析器解析为实际的物理视图,对于InternalResourceViewResolver视图解析器,会做如下解析:

\* 通过prefix+ReturnValue+suffix这样的方式得到实际的物理视图,然后做转发操作

\* /WEB-INF/views/ReturnValue.jsp

\*/

@Controller

public class ClassName {

@RequestMapping("/RequestName") //这个和请求jsp中的href是一致的,无/也行

public type method() {

---

return "ReturnValue";

}

```

4.Spring配置XML
<!-- 配置自定义扫描的包 -->
<context:component-scan base-package="edwin" />

<!-- 配置视图解析器：如何把方法返回值解析为实际的物理视图 -->
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix" value="/WEB-INF/views/"></property>
    <property name="suffix" value=".jsp"></property>
</bean>

5.物理视图JSP：
视图解析器已经说明了物理视图表现的JSP文件地址：prefix+ReturnValue+suffix，即/WEB-INF/views/ReturnValue.jsp
在该文件中声明即可

```

注意:触发请求的条件可以是一个文件,如 HTML、JSP,或者是一个文件中的触发 URL,在 RequestMapping 中 value 需要写入这个 URL,文件的话名字需要全。

## 函数传值

在 @RequestMapping 注释的方法中  
可以添加一个参数:

```

public String map(String name){
    System.out.println(name);
    return "Mapping";
}

```

这样的话如果 URL 的参数中有该参数,则使用,否则为 null

方法 2:

参数使用 @RequestParam("name") String name

这样的话 URL 参数必须涵该参数,否则报错。

使用 Model 添加属性:

```

@RequestMapping("/{hello}"/)
public String hello(String username, Model model) {
    System.out.println("hello");
    model.addAttribute("username", username);
    System.out.println(username);
    return "hello";
}

```

然后在视图文件中可以调用这个属性:以 \${attributeName} 形式

```

<h1>Hello ${name}</h1>
</body>

```

关于 addAttribute()单参数:

```

model.addAttribute("username", username);
//此时用那个作为key?它默认是使用对象的类型作为key-->model.addAttribute("string", username)
model.addAttribute(username);

```

对于对象,使用其类的小写:如 User——>user

```

model.addAttribute("username", username);
//此时用那个作为key?它默认是使用对象的类型作为key-->model.addAttribute("string", username)
//model.addAttribute(new User());-->addAtt("user", new User());

```

# 错误日志

## 包不能解析

需要导包：project – properties--java build path –libraries—add

```
import javax.servlet.http.HttpServletRequest;
```

```
import javax.servlet.http.HttpServletResponse;
```

这两个类需要导入：servlet-api.jar

[org.springframework.web.servlet.DispatcherServlet](#)

在写 springMVC 时，导入所有需要的包后，运行程序，控制台报以下错误：

java.lang.ClassNotFoundException: org.springframework.web.servlet.DispatcherServlet

解决方案：

要把依赖都加进 classpath 下。

步骤：

项目右击-->properties-->Deployment Assembly-->add-->Java Build Path Entries-->导入所有依赖的 Jar 包，重新 start tomcat 即可。

(**com.mysql.cj.jdbc.Driver** 无法找到也是这个处理方法)

## Run As 无 on servlet

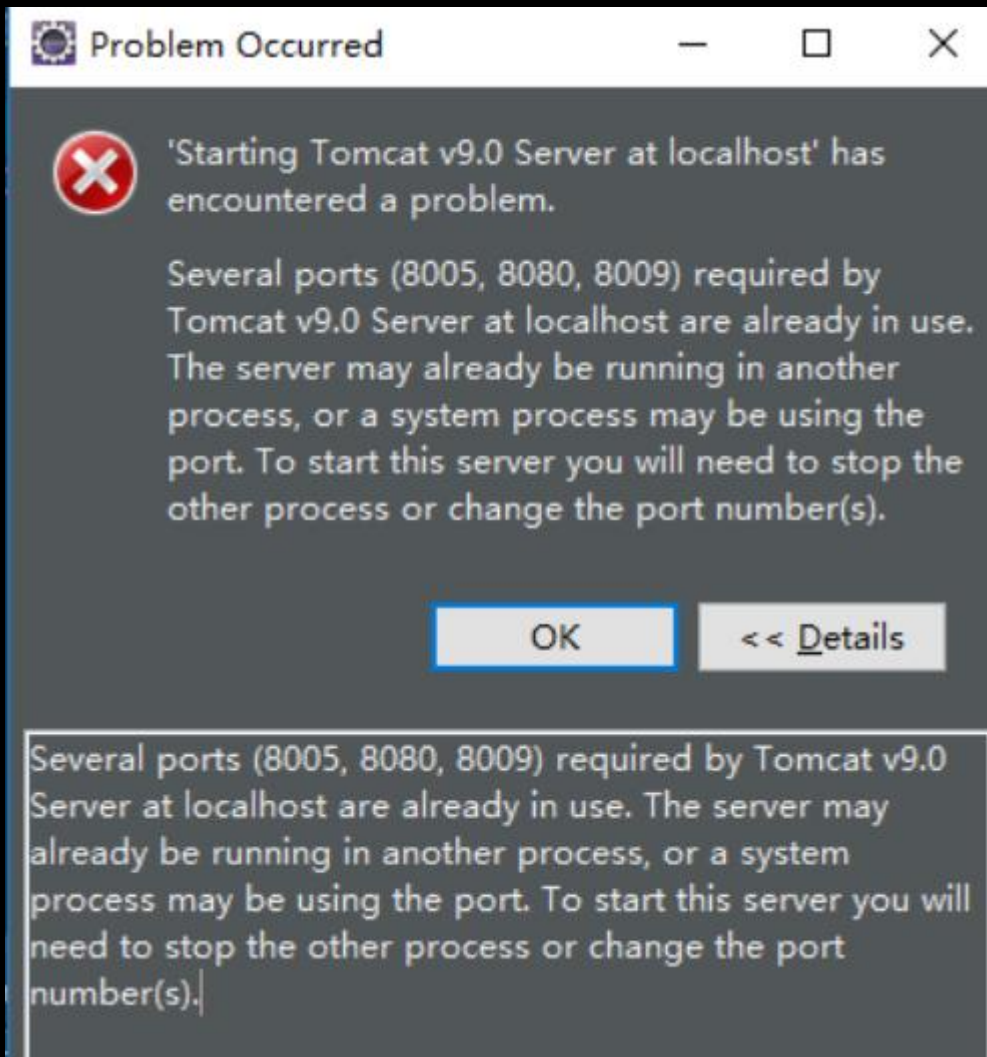
<https://blog.csdn.net/S852509769/article/details/79753070>

### <default-servlet-handler>报错

org.springframework.beans.factory.xml.XmlBeanDefinitionStoreException: Line 17 in XML document from class path resource [springmvc.xml] is invalid; nested exception is org.xml.sax.SAXParseException; lineNumber: 17; columnNumber: 29; cvc-complex-type.2.4.a: 发现了以元素 '{"http://www.springframework.org/schema/beans":default-servlet-handler}' 开头的无效内容。应以 '{"http://www.springframework.org/schema/mvc":path-matching, "http://www.springframework.org/schema/mvc":message-converters, "http://www.springframework.org/schema/mvc":argument-resolvers, "http://www.springframework.org/schema/mvc":return-value-handlers, "http://www.springframework.org/schema/mvc":async-support}' 之一开头。

Several ports (8005, 8080, 8009)





由于你已经在外面通过 startup.bat 打开了 tomcat,而在 eclipse 中运行时会再一次通过 server 打开 Tomcat,这就造成了上面端口被占用的发中生,解决方法就是找到 Tomcat 的文件夹下 bin 文件夹然后双击 shutdown.bat 关闭 Tomcat 后再在 eclipse 中运行即可

Unknown column 'something' in 'field list'

insert into users values("+username+", "+password+")

其中 username 需要”

关于 JSP 运行显示为下载

这是 jsp 页面<%@>的配置错误,导致浏览器或 servlet 不知道怎么解析正确格式:

```
1<%@ page language="java" contentType="text/html;UTF-8" pageEncoding="UTF-8"%>
```

错误:

1. 差 type:

```
<%@ page language="java" contentType="UTF-8" pageEncoding="UTF-8"%>
```

2. 差分号:

```
1<%@ page language="java" contentType="text/html UTF-8" pageEncoding="UTF-8"%>
```

