Spring MVC

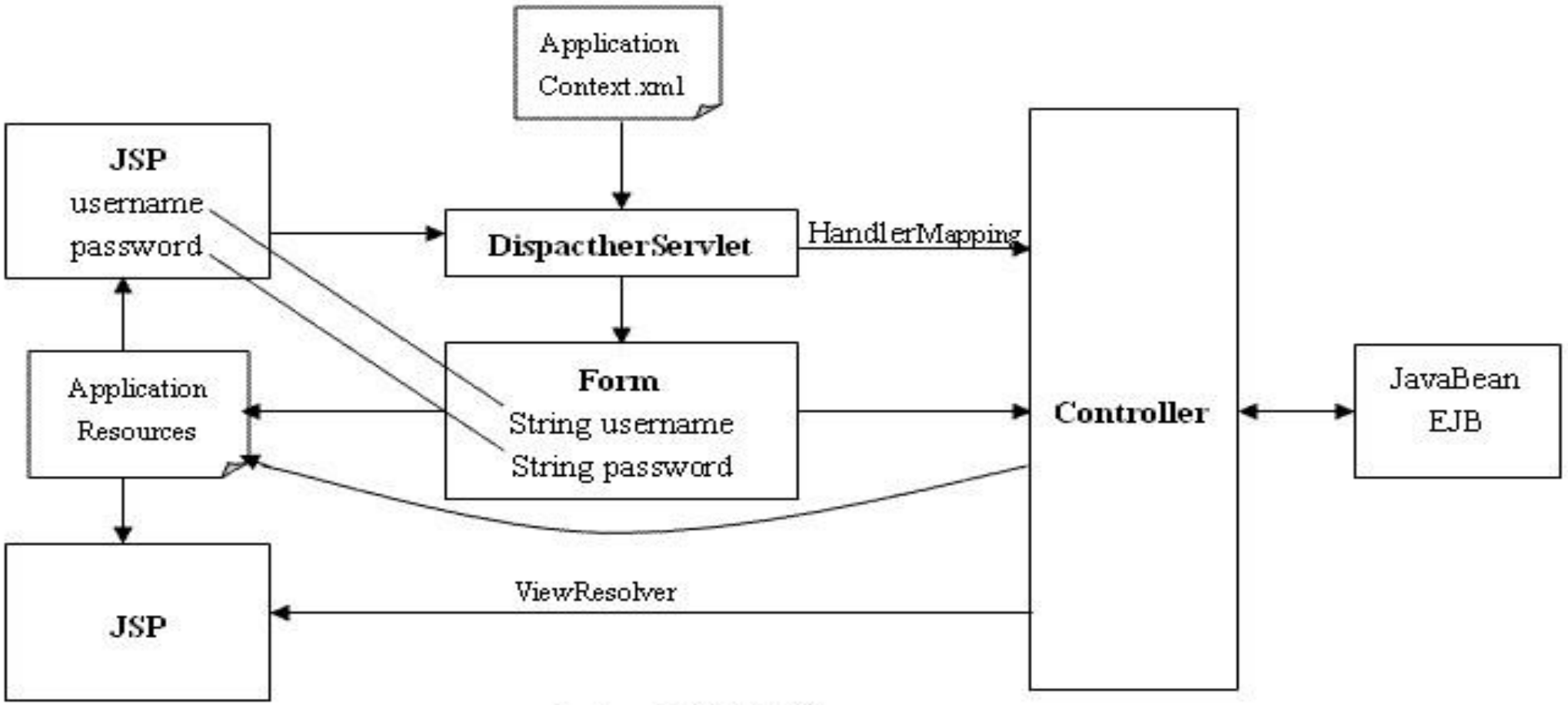
王新阳
wxyyuppie@bjfu.edu.cn

主要内容

- 开发过程
- 简单示例
- 架构与工作原理
- 核心组件
- MVC对比分析
- SpringMVC主要功能及原理

Spring MVC简介

Spring MVC开发流程



Spring 开发流程图

Spring MVC简单示例

见工程ch9

1 创建Web应用并引入JAR包

2 在web.xml文件中部署DispatcherServlet

3 创建Web应用首页

4 创建Controller类

5 创建Spring MVC配置文件并配置Controller映射信息

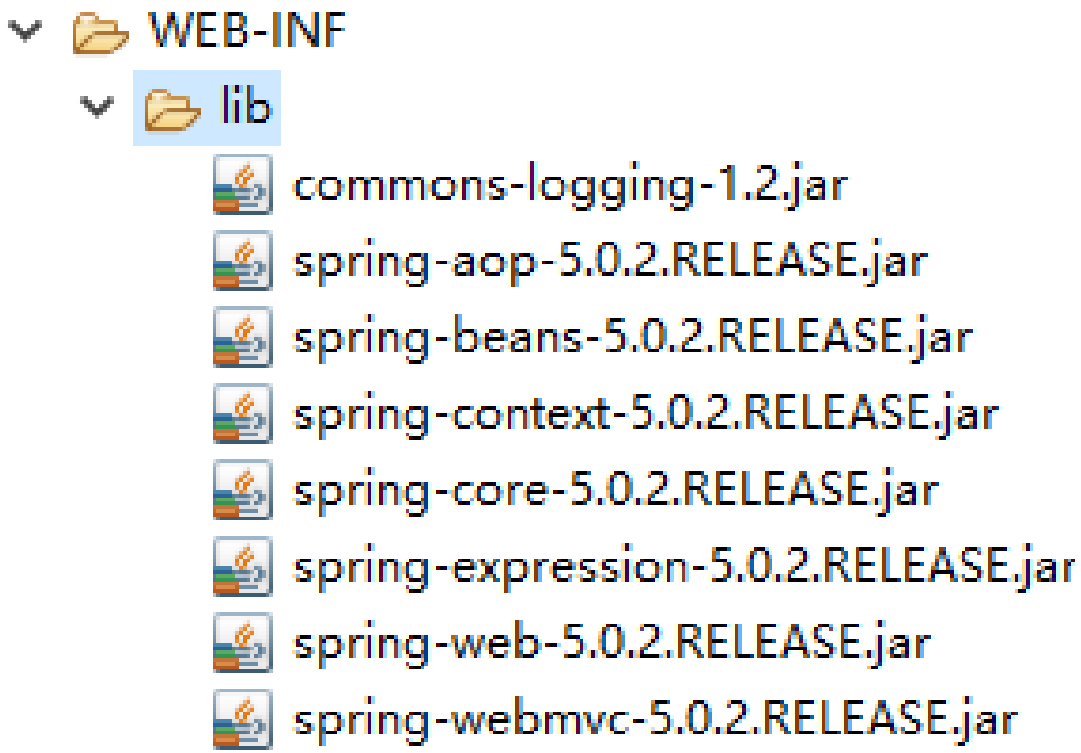
6 应用的其他页面

7 发布并运行Spring MVC应用

Spring MVC简单示例

1 创建Web应用并引入JAR包

添加Spring MVC程序所需要的JAR包，包括Spring的4个核心JAR包、commons-logging的JAR包以及两个Web相关的JAR包（spring-web-5.0.2.RELEASE.jar和spring-webmvc-5.0.2.RELEASE.jar）。另外，在Spring MVC应用中使用注解时，要添加spring-aop-5.0.2.RELEASE.jar包。



Spring MVC简单示例

2 在web.xml文件中部署DispatcherServlet

在开发Spring MVC应用时，需要在web.xml中部署DispatcherServlet

```
<!--部署DispatcherServlet-->
<servlet>
   <servlet-name>springmvc</servlet-name>
   <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
   <!-- 表示容器在启动时立即加载servlet -->
   <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
        <servlet-name>springmvc</servlet-name>
        <!-- 处理所有URL-->
   <url-pattern>/</url-pattern>
</servlet-mapping>
```

上述DispatcherServlet的servlet对象springmvc初始化时，将在应用程序的WEB-INF目录下查找一个配置文件，该配置文件的命名规则是“servletName-servlet.xml”，如：springmvc-servlet.xml。

Spring MVC简单示例

3 创建Web应用首页

与springmvc-servlet.xml配置文件中的bean名称一致

```
<body>
 没注册的用户，请<a href="${pageContext.request.contextPath }/register1">注册</a>!
<br>
 已注册的用户，去<a href="${pageContext.request.contextPath }/login">登录</a>!
</body>
```

- ✓ ${pageContext.request.contextPath}：等价于<%=request.getContextPath()%>，用于取出部署的应用程序名或者是当前的项目名称
- ✓ 比如项目名称是demo1在浏览器中输入为http://localhost:8080/demo1/a.jsp ${pageContext.request.contextPath}或<%=request.getContextPath()%>取出来的就是/demo1,而"/"代表的含义就是http://localhost:8080

Spring MVC简单示例

4 创建Controller类

在src目录下，创建包controller，并在该包中创建“RegisterController”和“LoginController”两个传统风格的控制器类（实现了Controller接口），分别处理首页中“注册”和“登录”超链接请求。

Spring MVC简单示例

5 创建Spring MVC配置文件并配置Controller映射信息

传统风格的控制器定义后，需要在Spring MVC配置文件中部署它们（学习基于注解的控制器后，不再需要部署控制器）。在WEB-INF目录下，创建名为springmvc-servlet.xml的配置文件

与web.xml中配置的servelet名称一致

控制器的访问路径放在bean的name属性中，不是id属性

```
<!--LoginController控制器类，映射到“/login” -->
 <bean name="/login" class="controller.LoginController"/>
 <!--RegisterController控制器类，映射到“/register” -->
 <bean name="/register1" class="controller.RegisterController"/>
```

Spring MVC简单示例

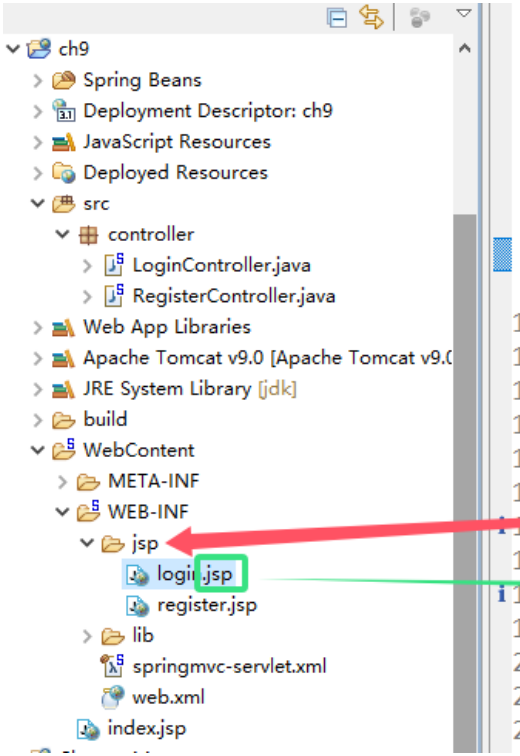
6 应用的其他页面

RegisterController控制器处理成功后，跳转到“/WEB-INF/jsp/register.jsp”视图；LoginController控制器处理成功后，跳转到“/WEB-INF/jsp/login.jsp”视图。因此，应用的“/WEB-INF/jsp”目录下应有“register.jsp”和“login.jsp”页面。

Spring MVC简单示例

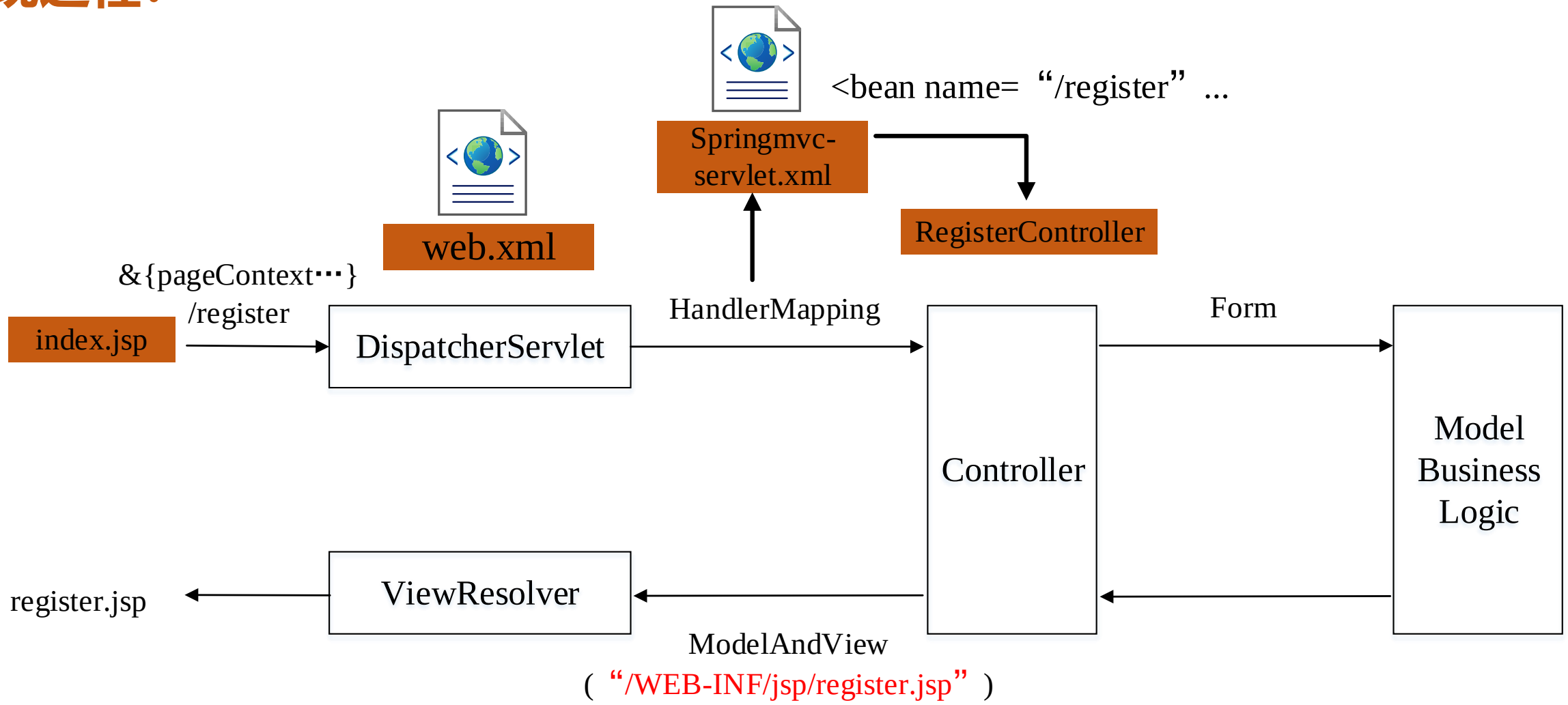
7 视图解析器

```
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver"
        id="internalResourceViewResolver">
    <!-- 前缀 -->
    <property name="prefix" value="/WEB-INF/jsp/" />
    <!-- 后缀 -->
    <property name="suffix" value=".jsp" />
</bean>
```

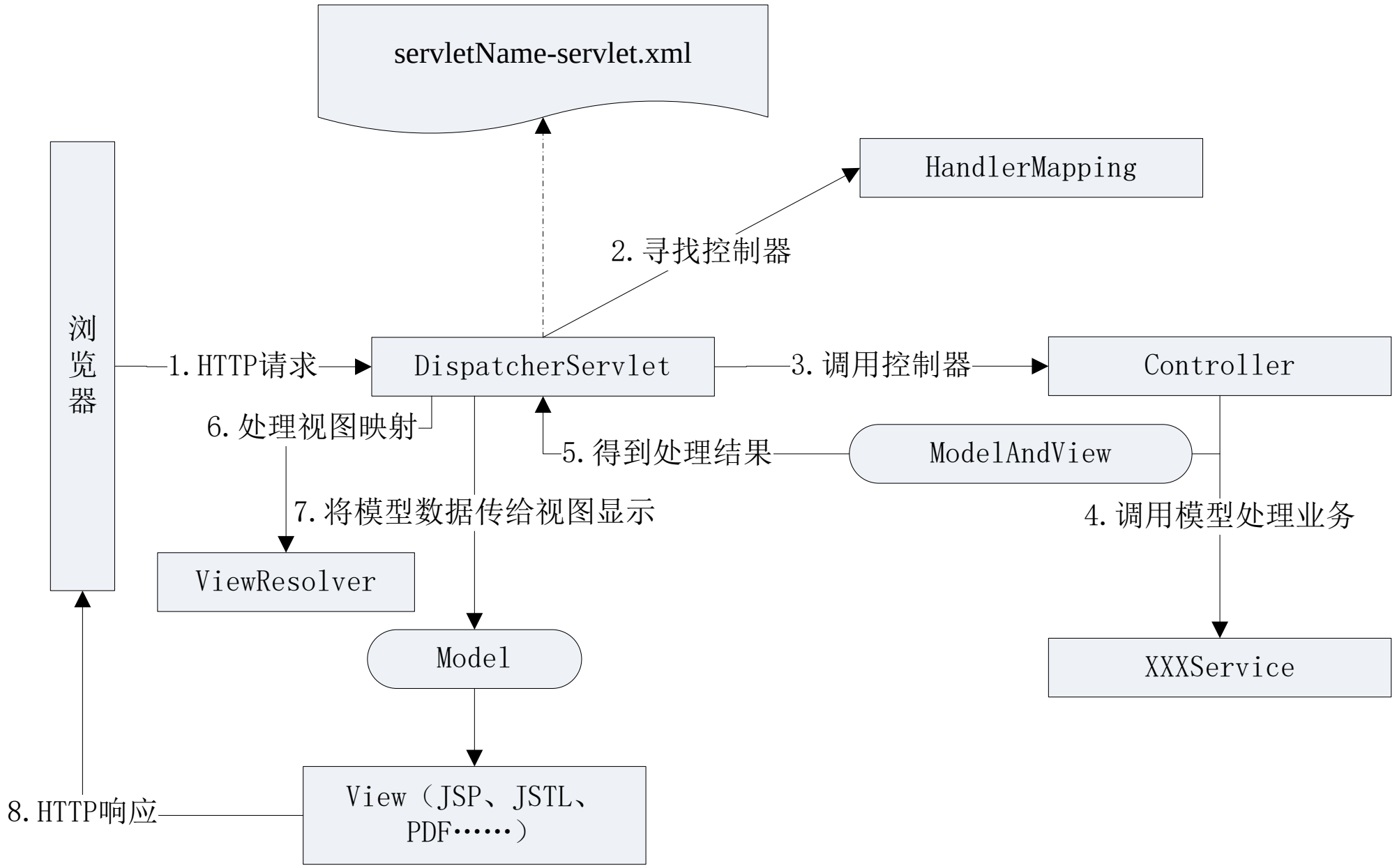



```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <beans xmlns="http://www.springframework.org/schema/beans"
3      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4      xsi:schemaLocation="
5              http://www.springframework.org/schema/beans
6              http://www.springframework.org/schema/beans/spring-beans.xsd">
7    <!--LoginController控制器类，映射到“/login”  -->
8    <bean name="/login" class="controller.LoginController"/>
9    <!--RegisterController控制器类，映射到“/register”  -->
10   <bean name="/register1" class="controller.RegisterController"/>
11
12   <!--配置视图解析器 -->
13   <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver"
14               id="internalResourceViewResolver">
15         <!--前缀 -->
16         <property name="prefix" value="/WEB-INF/jsp/" />
17         <!-- 后缀 -->
18         <property name="suffix" value=".jsp" />
19   </bean>
20
21 </beans>
22
```

实现过程：


<bean name=“/register” ...
Springmvc-servlet.xml
web.xml
RegisterController
&{pageContext…}
/register
index.jsp
DispatcherServlet
HandlerMapping
Form
Controller
Model Business Logic
register.jsp
ViewResolver
ModelAndView
(“/WEB-INF/jsp/register.jsp”)

Spring MVC架构与工作原理



Spring MVC架构与工作原理

- (1) Http请求：客户端请求提交到DispatcherServlet。
- (2) 寻找处理器：由DispatcherServlet控制器查询一个或多个HandlerMapping，找到处理请求的Controller。
- (3) 调用处理器：DispatcherServlet将请求提交到Controller。
- (4)(5)调用业务处理和返回结果：Controller调用业务逻辑处理后，返回ModelAndView。
- (6)(7)处理视图映射并返回模型：DispatcherServlet查询一个或多个ViewResoler视图解析器，找到ModelAndView指定的视图。
- (8) Http响应：视图负责将结果显示到客户端。

Spring MVC核心组件

- Spring MVC主要由DispatcherServlet、处理器映射、处理器(控制器)、视图解析器、视图组成。
- 两个核心：
  - 处理器映射（HandlerMapping）：选择使用哪个控制器来处理请求 ——在web.xml中声明
  - 视图解析器（ViewResolver）：选择结果应该如何渲染——默认在XXX-servlet.xml中声明

通过以上两点，Spring MVC保证了如何选择控制处理请求和如何选择视图展现输出之间的松耦合。

DispatcherServlet

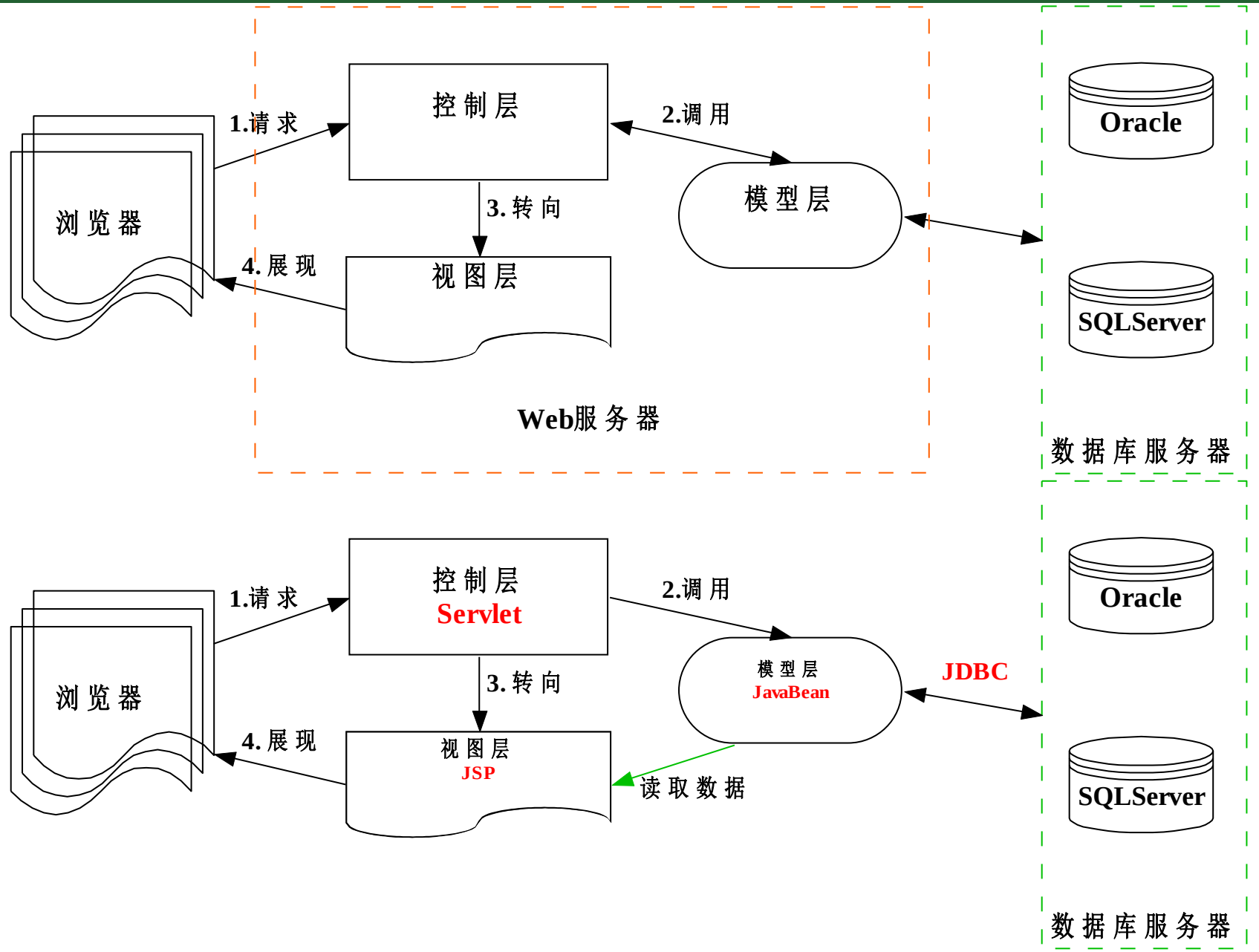
- DispatcherServlet是整个Spring MVC的核心。它负责接收HTTP请求组织协调Spring MVC的各个组成部分。其主要工作有以下三项：

  （1）截获符合特定格式的URL请求。

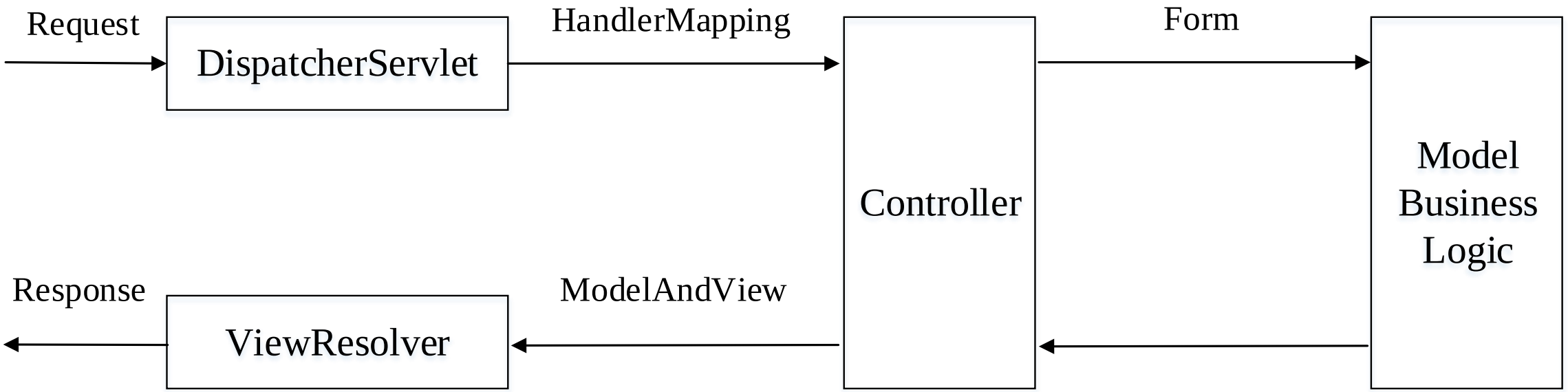
  （2）初始化DispatcherServlet上下文对应WebApplicationContext，并将其与业务层、持久化层的WebApplicationContext建立关联。

  （3）初始化Spring MVC的各个组件，并装配到DispatcherServlet中。

MVC与JSP MVC


控制层
1.请求
2.调用
浏览器
3.转向
模型层
4.展现
视图层
Oracle
SQLServer
Web服务器
数据库服务器
控制层
Servlet
1.请求
2.调用
浏览器
3.转向
模型层
JavaBean
JDBC
4.展现
视图层
JSP
读取数据
Oracle
SQLServer
数据库服务器

Spring MVC架构



Spring MVC架构特点

- 由专门的控制器接受请求
- 但控制器一般不直接处理请求，而是将其委托给Spring上下文中的其他bean，通过Spring的依赖注入功能，这些bean被注入到控制器中。

Spring MVC主要功能及原理

- 1　如何实现URL请求响应
- 2　如何实现请求表单参数传递
- 3　如何实现结果页面映射和跳转
- 4　如何实现向结果页面传递数据

（1）如何实现URL请求响应

请求表单的请求URL→借助Controller类的注解声明，跳转到具体的Controller类或者方法中，三种方法：

方法1：在类的层面上声明URL对应；

方法2：在方法层面上声明URL映射；

方法3：一个 Controller 对应一个 URL，由请求参数决定请求处理方法

（1）如何实现URL请求响应

方法1：在类的层面上声明URL对应

SpringMVCTest工程中的Login1.jsp
<form action="login1.do

①在类声明之前加上如下注解

- @Controller
- @RequestMapping（“/login1.do”）//表明该类对应这个URL请求

②在对应的Controller方法声明之前加上如下注解

- @RequestMapping //<——② //表明该方法为默认的处理URL请求的方法

UserController.java

```
@Controller
@RequestMapping("/login1.do")//@RequestMapping 可以标注在类定义处,
public class UserController {
    @RequestMapping //<——②真正让ForumController 具备 Spring MVC C
    public String logining() {
        System.out.println("正在登陆ing11111。。。。。");
        return "logining1";//返回的为视图名称，即页面logining1.jsp
    }
```

（1）如何实现URL请求响应

方法2：在方法层面上声明URL映射

SpringMVCTest工程
UserController、
Login2.jsp <form action= *“login2.do”*

①在类声明之前加上如下注解

- @Controller//表明该类归Spring框架掌控

②直接在在对应的Controller方法声明之前加上如下注解

- @RequestMapping(“/login2.do”)//表明该类对应这个URL请求

UserController.java

```
 //注意：只有测试下面这个函数时，要注释掉类定义之前的@RequestMapping
@RequestMapping("/login2.do") ///login2.do请求则由 下面这个 方法处理。
public String logining2() {
    System.out.println("正在登陆ing2222222。。。。。。");
    return "logining2";
}
```

（1）如何实现URL请求响应

方法3：一个 Controller 对应一个 URL，由请求参数决定请求处理方法

①在类声明之前加上如下注解

- @Controller
- @RequestMapping（“/login1.do”）//表明该类对应这个URL请求

②直接在对应的Controller方法声明之前加上如下注解

- @RequestMapping(params = "method=login3") // 如果URL请求中包括"method=login3"的参数，由本方法进行处理

```
<body>
    <form action="login1.do" method="post"
        <fieldset>
        <legend>Login3</legend>
        <input type="text" name="method" value="login3"
        <INPUT TYPE="SUBMIT"> <!
        </fieldset>
    </form>
</body>
```

参见SpringMVCTest工程中的Login3.jsp
URL处以 HTTP POST 方式提交login1.do?method=login3

```
<form action= "login1.do "
<input type= "text" name="method" value="login3">
```

将表单中login3标签处前端发送的参数传递给方法logining3（）处理

```
@RequestMapping(params = "method=login3") ///以 HT
public String logining3() {
    System.out.println("正在登陆ing33333。。。。。。");
    return "logining3";
}
```

UserController.java

（2）如何实现请求表单参数传递

请求表单的请求URL→借助Controller类的注解声明，按参数名匹配进行绑定到对应的函数上，三种方法：

方法1：按参数名匹配的原则绑定到 URL 请求参数；

方法2：使用JavaBean传递数据；

方法3：使用注解@RequestParam(“...”) 传递参数到函数形参上去

（2）如何实现请求表单参数传递

方法1：请求处理方法入参的类型可以是 Java 基本数据类型或 String 类型，这时方法入参按参数名匹配的原则绑定到 URL 请求参数，同时还自动完成 String 类型的 URL 请求参数到请求处理方法参数类型的转换

①保证表单参数名称和函数形参名称一致即可

- **@RequestMapping(params = “method=login4”) // 如果URL请求中包括“method=login4”的参数，由本方法进行处理，并自动匹配函数的形参**
- **public String logining4(String username, String pwd) {...}**
- **注意：如果入参是基本数据类型（如 int、long、float 等），URL 请求参数中一定要有对应的参数，否则将抛出 TypeMismatchException 异常，提示无法将 null 转换为基本数据类型。**

参见SpringMVCTest工程中的Login4.jsp
<form action="login1.do?method=login4"

```java
@RequestMapping(params = "method=login4") ///以 HTTP PO
public String logining4(String username, String pwd) {
    System.out.println("正在登陆ing4444。。。。。。");
    System.out.println("用户名："+username);
    System.out.println("密码："+pwd);
    return "logining4";
}
```

UserController.java

```html
<form action="login1.do?method=login4" method="post">
    <fieldset>
    <legend>Login4</legend>
    <input type="text" name="username" value="王菲444"/>
    <input type="password" name="pwd" value="abc123"/>
    <INPUT TYPE="SUBMIT"> <!-- Press this to submit fo
    </fieldset>
</form>
```

（2）如何实现请求表单参数传递

方法2：使用JavaBean传递数据

① 保证表单参数名称和JavaBean的私有属性名称保持一致即可

- @RequestMapping(params = “method=login5”) // 如果URL请求中包括“method=login5”的参数，由本方法进行处理，并自动匹配函数的形参

```
<form action="login1.do?method=login5" method="post">
    <fieldset>
    <legend>Login5</legend>
    <input type="text" name="username" value="王菲555"/>
    <input type="password" name="pwd" value="abc123"/>
    <INPUT TYPE="SUBMIT"> <!-- Press this to submit form -->
    </fieldset>
</form>
```

SpringMVCTest工程中的Login5.jsp
<form action="login1.do?method=login5"

UserController.java

```
@RequestMapping(params = "method=login5")///以 HTTP
public String logining5( User user) {
    System.out.println("正在登陆ing5555。。。。。。");
    System.out.println("用户名："+user.getUsername());
    System.out.println("User："+user);
    return "logining5";
}
```

```
public class User {
private String username;
private String pwd;
public String getUsername() {
    return username;
}
```

（2）如何实现请求表单参数传递

方法3：使用注解@RequestParam(“...”）传递参数到函数形参上去

① 在函数形参名称前使用@RequestParam(“...”）,表示将请求表单中的username666变量传给本方法的username

- @RequestMapping(params = "method=login6")
- public String logining6(@RequestParam("username66")String username)

```
<form action="login1.do?method=login6" method="post">
    <fieldset>
    <legend>Login6</legend>
    <input type="text" name="username66" value="王菲6666"/>
    <input type="password" name="pwd" value="abc123"/>
    <INPUT TYPE="SUBMIT"> <!-- Press this to submit form
    </fieldset>
</form>
```

参见SpringMVCTest工程中的Login6.jsp

<form action="login1.do?method=login6"

UserController.java

```
@RequestMapping(params = "method=login6") //以 HTTP POST 方式提交logi
public String logining6(@RequestParam("username66") String username)
    System.out.println("正在登陆ing666.....");
    System.out.println("用户名："+username);
    return "logining6";
}
```

（3）如何实现结果页面映射和跳转

- 将Controller函数**返回的字符串**直接拼接为JSP结果地址
- 通过appliationContext.xml文件的以下设置实现：
  - <bean
    class= *"org.springframework.web.servlet.view.InternalResourceViewResolver"*
    p:prefix= *"/WEB-INF/resultjsp/"*
    p:suffix=".jsp"/>
  - ***结果页面地址为：前缀+返回字符串+后缀***

```
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver"
    p:prefix="/resultjsp/" p:suffix=".jsp"/>
```

```
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver"
    p:prefix="/resultjsp/" p:suffix=".jsp"/>
```

/resultjsp/logining6.jsp=

```
@RequestMapping(params = "method=login6"  HTTP POST 方式提交login
public String logining6(@RequestParam("user me66")String username) {
    System.out.println("正在登陆ing666。。。。。。");
    System.out.println("用户名："+username);
    return "logining6";
}
```

UserController.java

（4）如何实现向结果页面传递数据

三种方法：

- **借助Spring框架提供的ModelAndView对象（request范围）**
- **借助Model接口或者Map对象（request范围）**
- **使用@SessionAttributes注解（session范围）**

（4）如何实现向结果页面传递数据

方法1：借助Spring框架提供的ModelAndView对象，将数据放入request范围共享

- **原理：** 在控制器中处理完用户请求后，可以把结果数据存储在该对象中，把要返回的视图名称存储在该对象的view属性中，然后把ModelAndView对象返回给SpringMVC框架。框架则会通过调用Spring配置文件中定义的视图解析器，对该对象进行解析，最后把结果数据传递到指定的结果视图上。
- **基本方法：**
  - 在处理函数的形参中添加ModelAndView对象并将该对象作为返回值
  - 将结果数据放入ModelAndView对象的model属性
  - 将结果页面放入ModelAndView对象的view属性

参见SpringMVCTest工程中的Login7.jsp

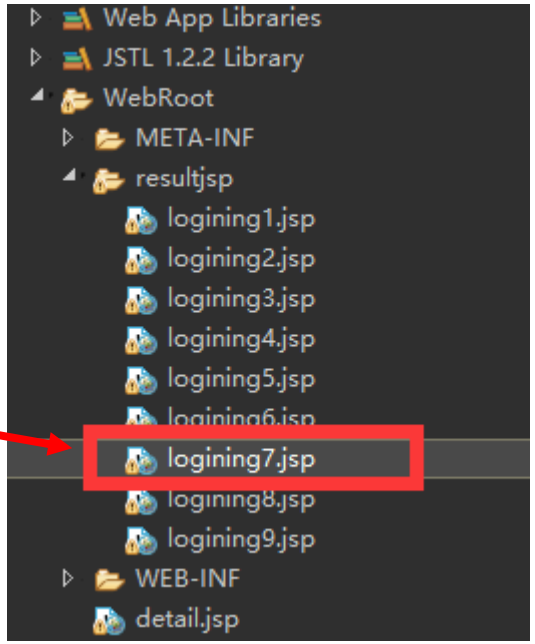
```
<form action="login1.do?method=login7"
```

（4）如何实现向结果页面传递数据

UserController.java

```
<form action="login1.do?method=login7" method="post">
    <fieldset>
    <legend>Login7</legend>
    <input type="text" name="username" value="王菲777"/>
    <input type="password" name="pwd" value="abc123"/>
    <INPUT TYPE="SUBMIT"> <!-- Press this to submit fo
    </fieldset>
</form>
```

```
@RequestMapping(params = "method=login7") //以 HTTP POST 方式提
public ModelAndView  logining7(User user, ModelAndView  mav) {
    System.out.println("正在登陆ing777。。。。。。");
 // 设置视图名称
    mav.setViewName("logining7");
    // 添加数据
    mav.addObject("userinfo",user);
    return mav;
}
```



resultjsp/logining7.jsp

```
<H>Logining7777...</H>
<div>
    <p>name::
        <span>${requestScope.userinfo.username}</span>
    </p>
    <p>pwd::
        <span>${requestScope.userinfo.pwd}</span>
    </p>
</div>
</body>
```

（4）如何实现向结果页面传递数据

方法2：借助Spring框架提供的Model接口/Map对象，实现request范围共享

- **基本方法：**
  - **model.addAttribute（key,value）或者map.put(key,value)**

```
@RequestMapping(params = "method=login8") ///以 HTTP
public String  logining8(User user, Model  model) {
    System.out.println("正在登陆ing8888。。。。。。");
    model.addAttribute("userinfo88",user);
    return  "logining8" ;
}
```

UserController.java

resultjsp/logining7.jsp

```
<body>
<H>Logining88....</H>
<div>
    <p>name:
        <span>${requestScope.userinfo88.username}</span>
    </p>
    <p>pwd:
        <span>${requestScope.userinfo88.pwd}</span>
    </p>
</div>
</body>
```

SpringMVCTest工程中的Login8.jsp

```
<form action= "login1.do?method=login8"
```

（4）如何实现向结果页面传递数据

方法3： @SessionAttributes类注解，声明session范围共享的数据

- 基本方法：将Model接口共享的数据变为Session范围的共享数据
  - ① 通过注解直接指明session共享的变量名（类定义处）

    例如 @SessionAttributes(value = {"name", "age", "address"})
  - ② 或者通过注解直接指明共享的某种数据类型

    例如 @SessionAttributes(types = User.class)//表明*只要是User类型的数据就存储一份到session中*

注意：@SessionAttribute属于类注解，在类的定义开始处进行注解

参见SpringMVCTest工程中的Login9.jsp

<form action= *"login1.do?method=login9 “*

@SessionAttributes(value = {"userinfo99"})

（4）如何实现向结果页面传递数据

```java
//使用@SessionAttributes实现Session，在类的开始定义处使用。
@RequestMapping(params = "method=login9") ///以 HTTP
public String  logining9(User user, Model  model) {
    System.out.println("正在登陆ing999。。。。。。");
    model.addAttribute("userinfo99",user);
    return "logining9";
}
```

UserController.java

UserController.java

```java
@Controller
//@SessionAttributes(value = {"userinfo99"})
//<——①将一个类成为 Spring 容器的 Bean, @Controller、
@RequestMapping("/login1.do")//@RequestMapping
public class UserController {
```

```
<H>Logining88....</H>
<div>
    <p>Request name:
        <span>${requestScope.userinfo99.username}</span>
    </p>
    <p>Request pwd:
        <span>${requestScope.userinfo99.pwd}</span>
    </p>
    ----------------------------------------
     <p>Session name :
        <span>${sessionScope.userinfo99.username}</span>
    </p>
    <p>Session pwd:
        <span>${sessionScope.userinfo99.pwd}</span>
    </p>
```

localhost:8080/SpringMVCTest/login1.do?method=login9

Logining88....

Request name: 王菲999

Request pwd: abc123

Session name :

Session pwd:

@SessionAttributes
被注释掉

（4）如何实现向结果页面传递数据

```java
//使用@SessionAttributes实现Session，在类的开始定义处使用。
@RequestMapping(params = "method=login9") ///以 HTTP
public String  logining9(User user, Model  model) {
    System.out.println("正在登陆ing999。。。。。。");
    model.addAttribute("userinfo99",user);
    return "logining9";
}
```

UserController.java

```jsp
<H>Logining88....</H>
<div>
    <p>Request name:
        <span>${requestScope.userinfo99.username}</span>
    </p>
    <p>Request pwd:
        <span>${requestScope.userinfo99.pwd}</span>
    </p>
    ----------------------------------------
     <p>Session name :
        <span>${sessionScope.userinfo99.username}</span>
    </p>
    <p>Session pwd:
        <span>${sessionScope.userinfo99.pwd}</span>
    </p>
```

```java
@Controller
@SessionAttributes(value = {"userinfo99"})
//@SessionAttributes(types = User.class)//
//<——①将一个类成为 Spring 容器的Bean, @Controlle
```

localhost:8080/SpringMVCTest/login1.do?method=login9

Logining88....

Request name: 王菲999

Request pwd: abc123

Session name : 王菲999

Session pwd: abc123

使用@SessionAttributes

感谢聆听

Thanks For Your Listening!