# SpringMVC框架(web工程)

## 1.springmvc优点

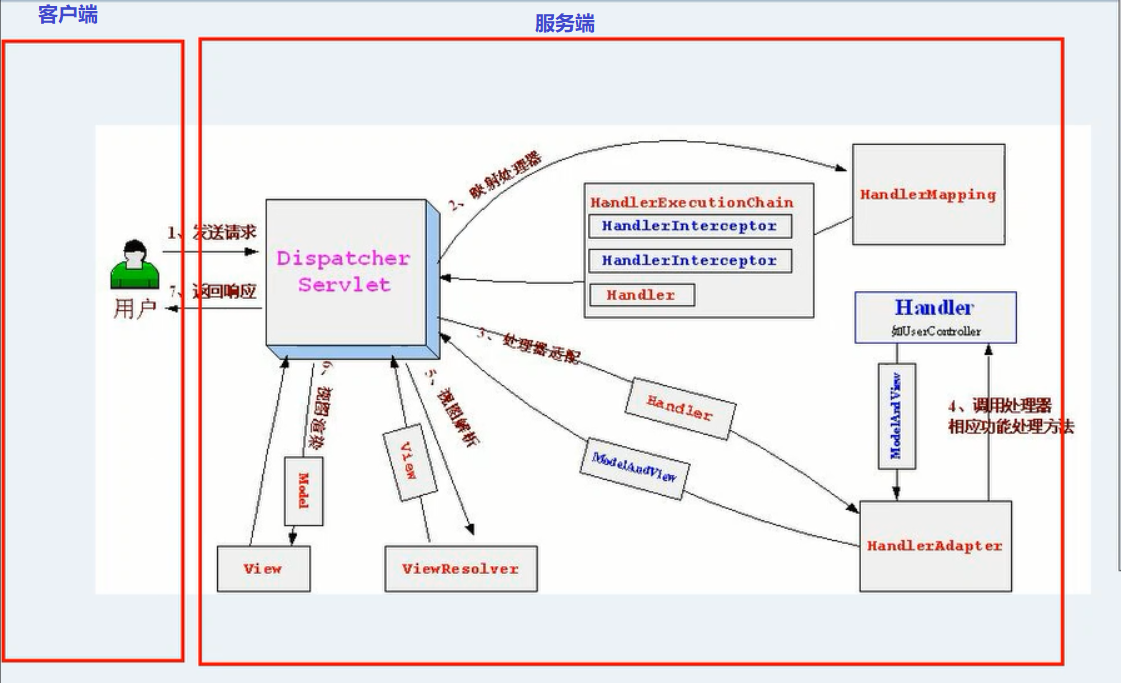
### 1.配置比较简单

### 2.效率还不错

### 3.SpringMVC提供大量扩展点，方便我们自定义一些功能

1. **Spring框架无缝对接**

## springmvc工作原理



### 1.DispatcherServlet(心脏) web.xml中要配置此servlet

### 分发处理器 根据接受到请求对象的不同 分发给不同处理者

### 2.HandlerMapping 映射处理器 根据请求URL映射到handler(action/controller/handler)上

### 3.HandlerAdapter 适配器 负责适配不同处理 根据handler的实现方式(实现接口/注解)不一样 去适配不同的handler

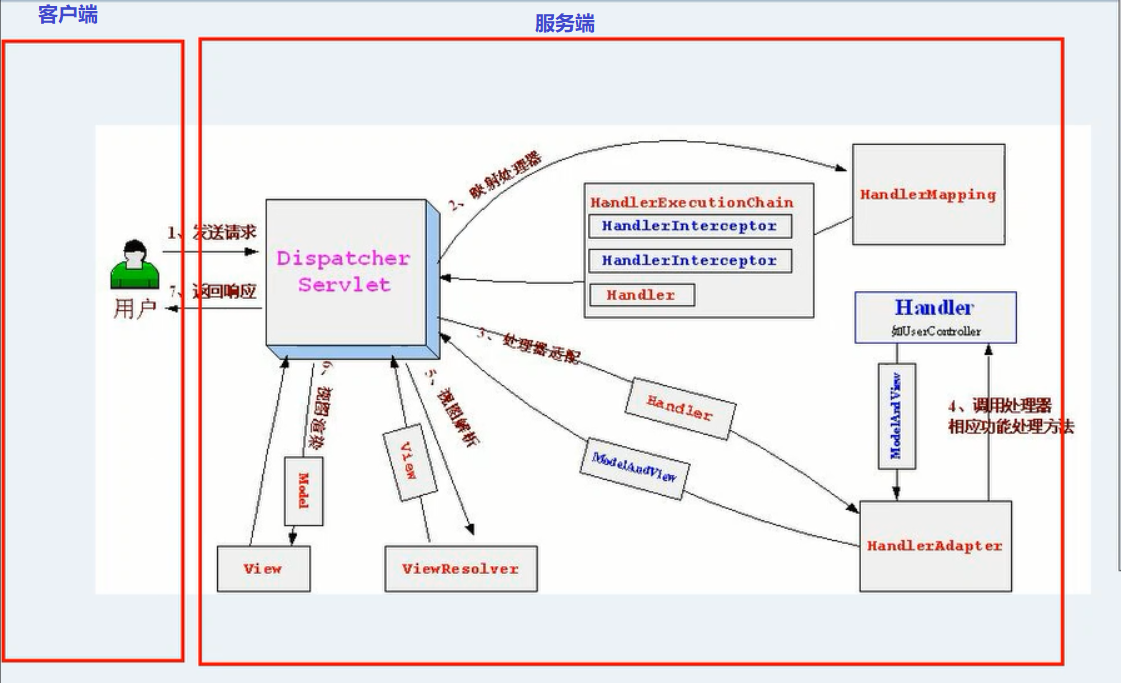
### 4.handler 业务处理器 UserController 处理核心业务逻辑，并返回对应的视图与数据模型(ModelAndView)

### 5.ViewResolver 视图解析器

### Java 中常见的视图解析技术: jsp(默认) freemarker......

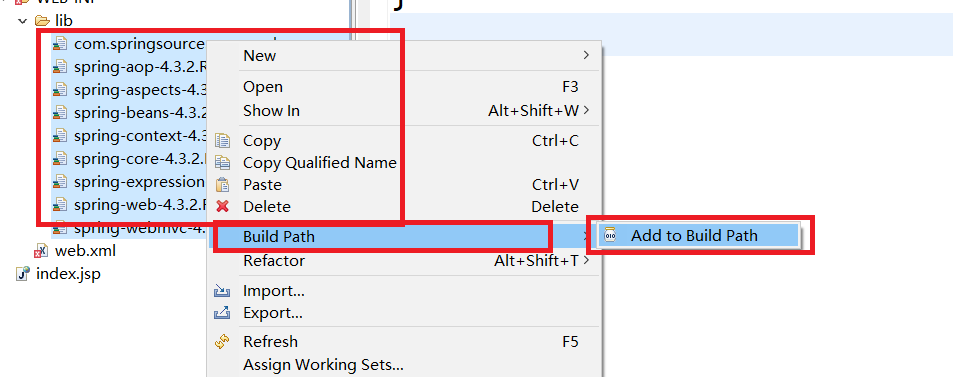
### 负责将视图名解析为具体的视图技术

### 6.View 具体的视图 负责处理视图技术， 最终响应给客户端

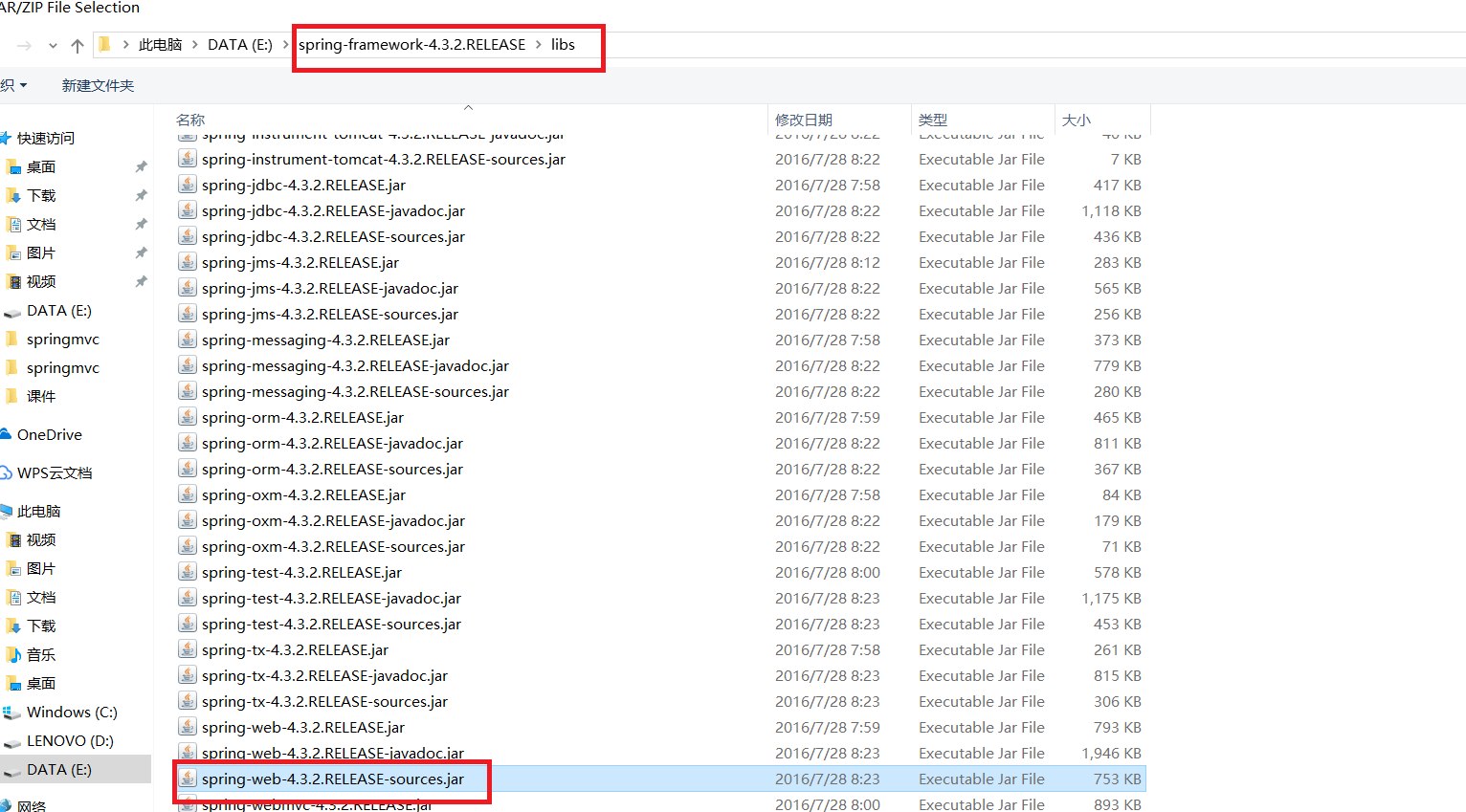


## 3.springmvc开发步骤

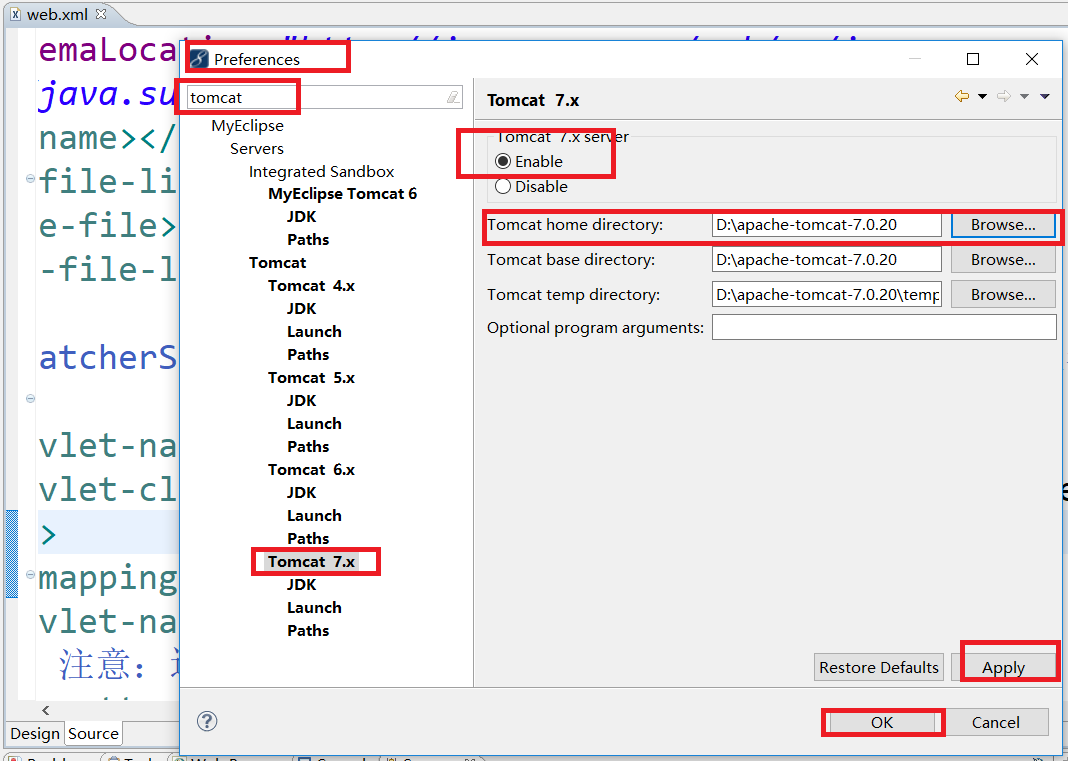
### 1.添加jar包

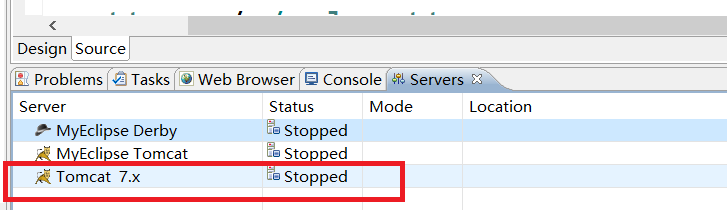


添加源码:



在IDE myEclipse 中配置tomcat



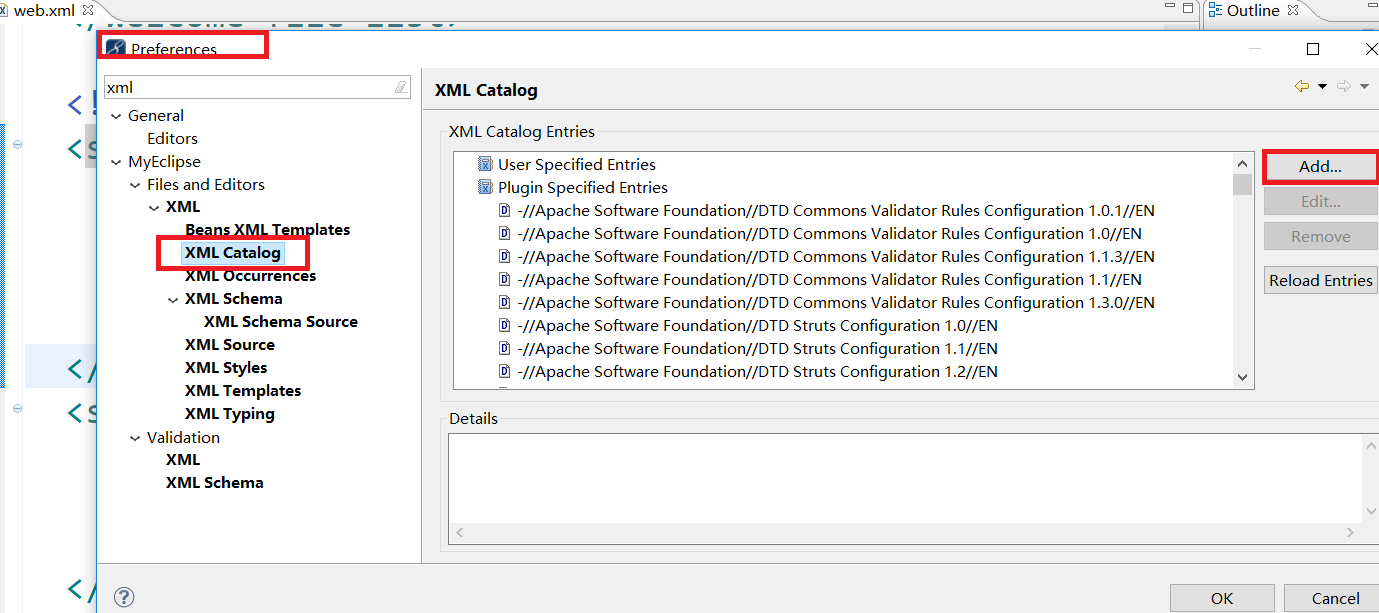


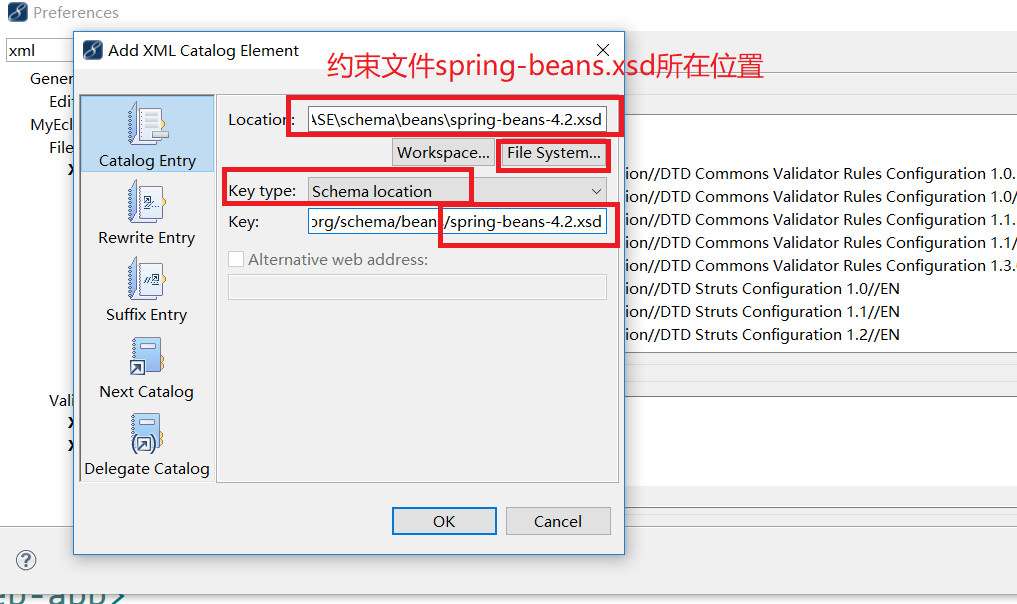
### 2.配置springmvc核心控制器DispatcherServlet(WEB-INF/web.xml)

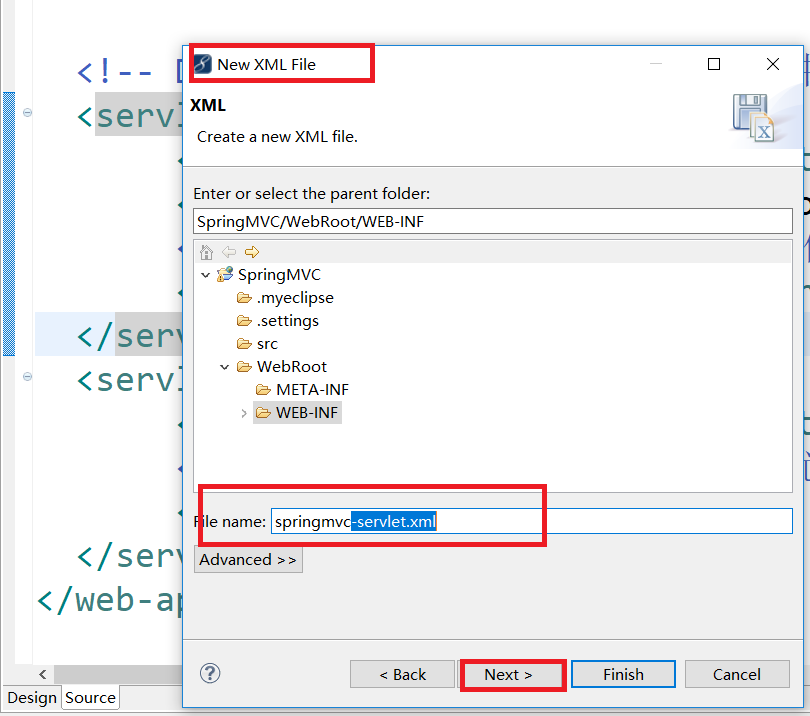
|  |
| --- |
| <!-- DispatcherServlet(心脏) web.xml中要配置此servlet 分发处理器 根据接受到请求对象的不同 分发给不同处理者 -->  <servlet>  <servlet-name>springmvc</servlet-name>  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>  <!-- 我们希望在tomcat容器启动的时候就初始化DispatcherServlet,加载springmvc的核心配置文件/WEB-INF/springmvc-servlet.xml -->  <load-on-startup>1</load-on-startup>  </servlet>  <servlet-mapping>  <servlet-name>springmvc</servlet-name>  <!-- 注意：这里不能配置成/\* 否则就映射不到请求 -->  <url-pattern>/</url-pattern>  </servlet-mapping> |

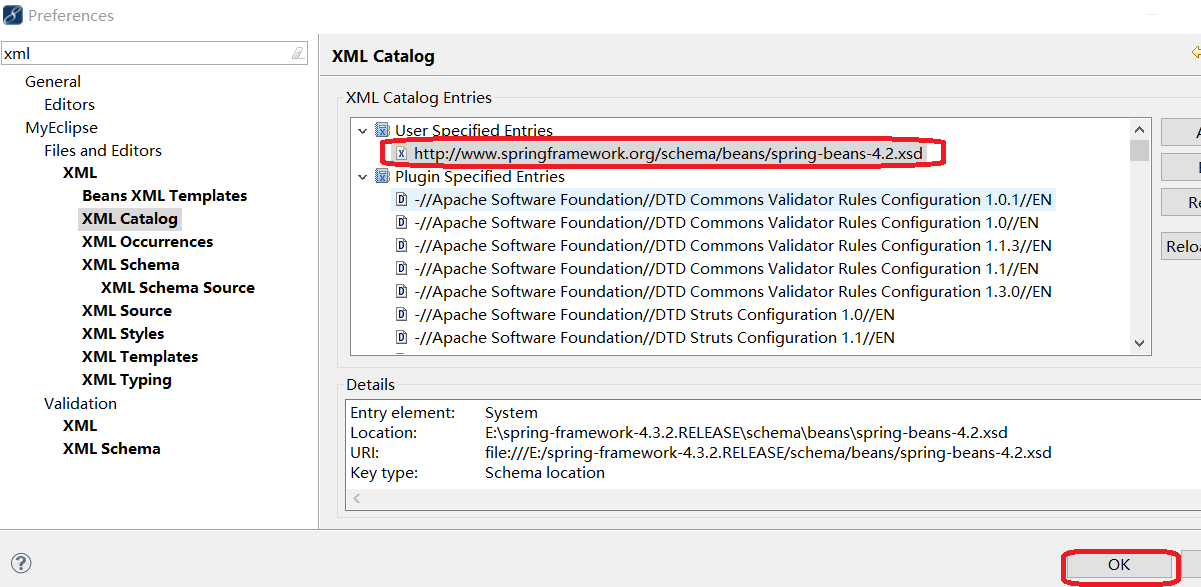
### 3.在/WEB-INF/springmvc-servlet.xml文件中配置

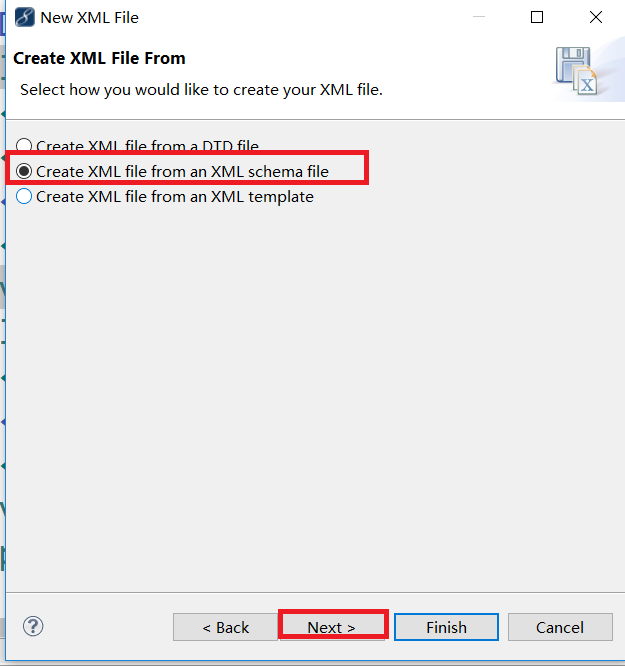
#### 1.添加约束文件spring-beans.xsd

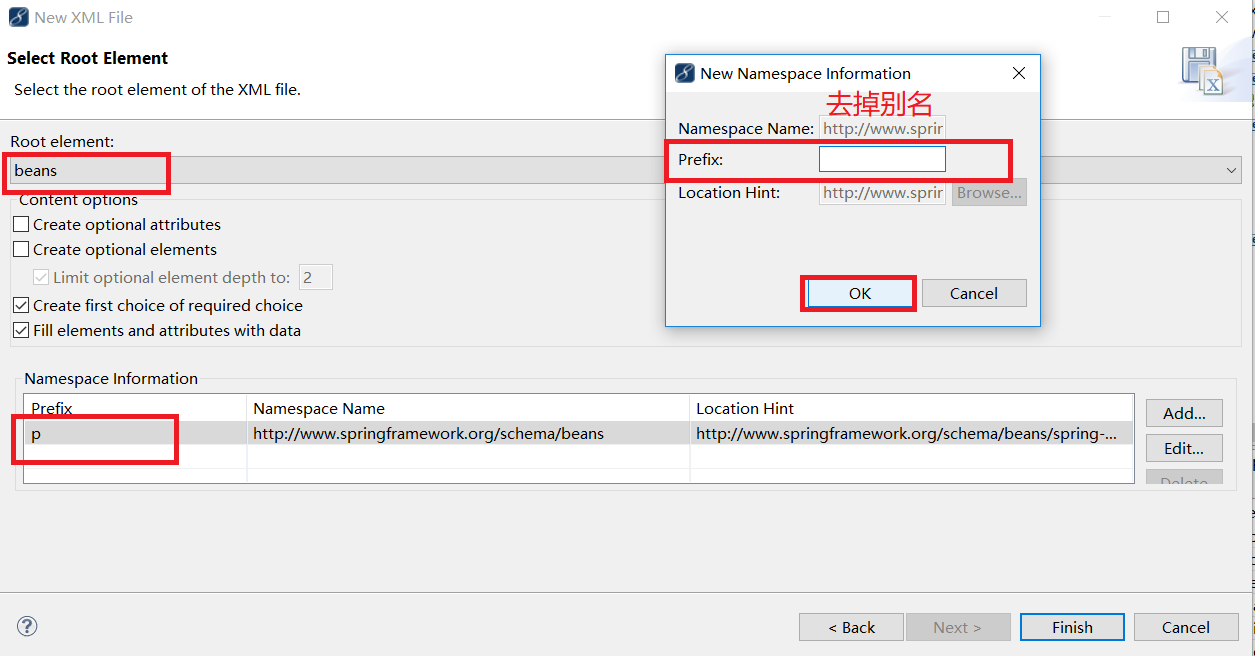




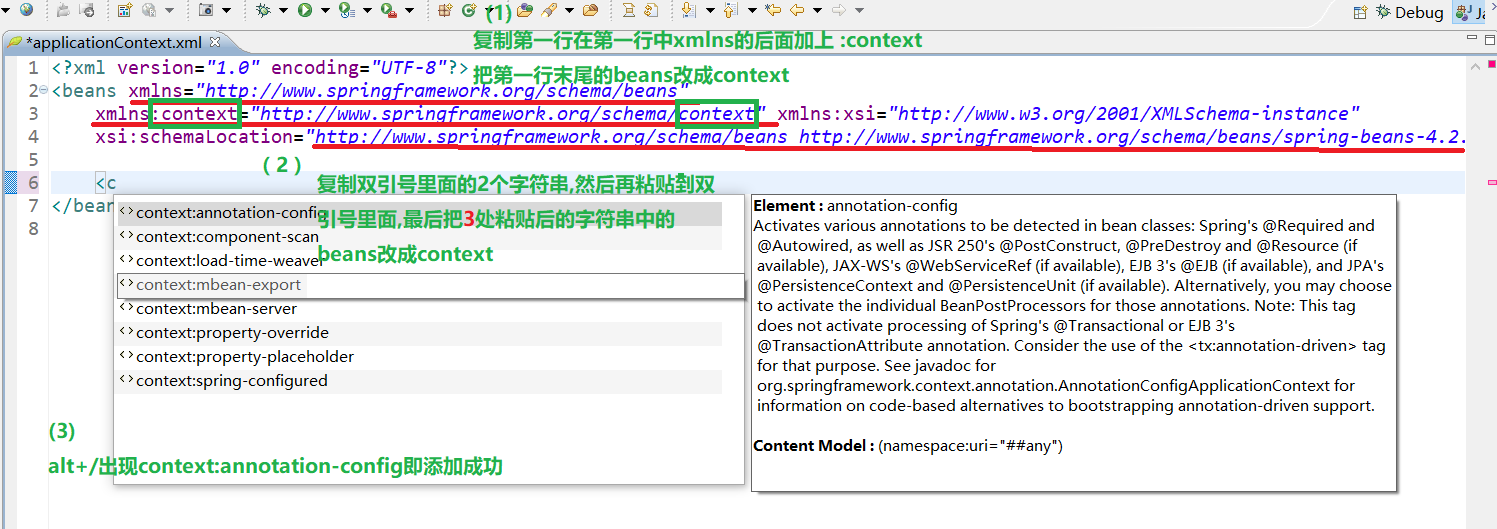






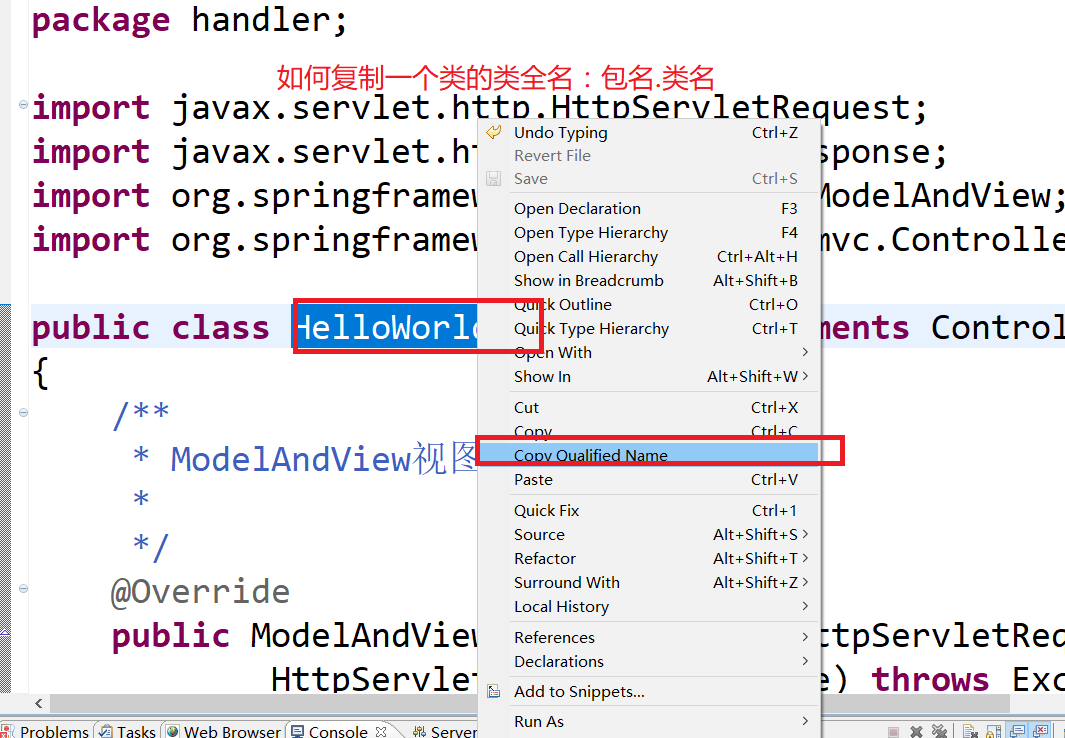


#### 2.手动加入context命名空间

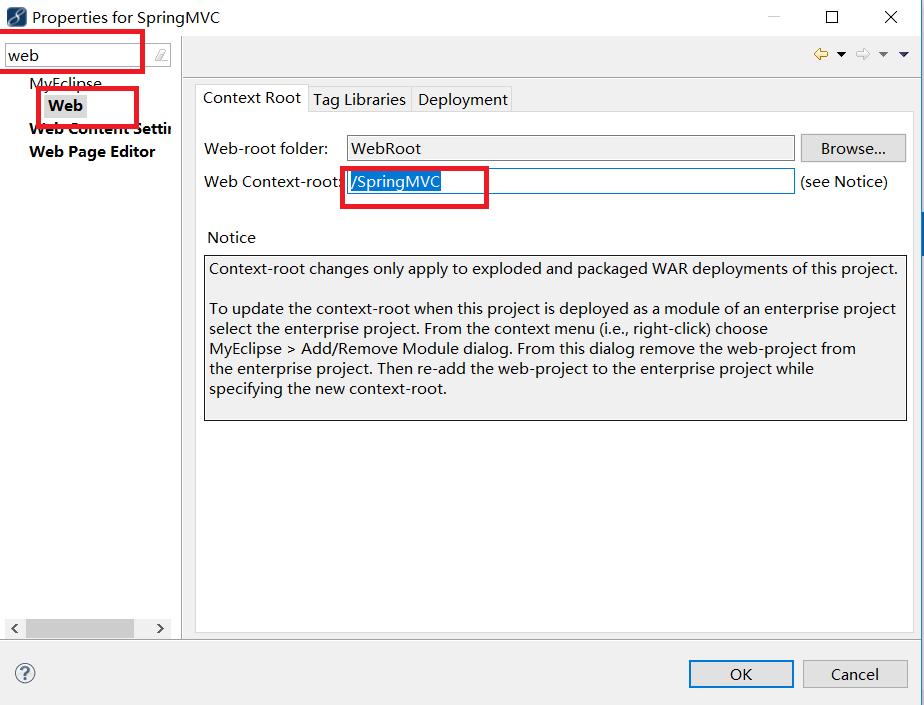


#### 3.配置springmvc-servlet.xml

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"*  *http://www.springframework.org/schema/beans*  *http://www.springframework.org/schema/beans/spring-beans-4.2.xsd*  *"*>  <!-- 2.HandlerMapping 映射处理器 根据请求URL映射到handler(action/controller/handler)上 -->  <bean class=*"org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping"*/>    <!-- 3.HandlerAdapter 适配器 负责适配不同处理 根据handler的实现方式(实现接口/注解)不一样 去适配不同的handler -->  <bean class=*"org.springframework.web.servlet.mvc.SimpleControllerHandlerAdapter"*/>    <!-- 4.handler 业务处理器 UserController 处理核心业务逻辑，并返回对应的视图与数据模型(ModelAndView) -->  <!-- http://localhost:8080/springmvc/helloWord -->  <bean name=*"*/helloWord*"* class=*"handler.HelloWorldController"*/>    <!-- 5.ViewResolver 视图解析器(类名要求记住:InternalResourceViewResolver) -->  <bean class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>  <!-- 配置视图的位置 prefix:前缀 -->  <property name=*"prefix"* value=*"/WEB-INF/view/"*/>  <property name=*"suffix"* value=*".jsp"*/>    <!-- 6.View 具体的视图 -->  <property name=*"viewClass"* value=*" org.springframework.web.servlet.view.JstlView"*/>  </bean>    </beans> |



查看web工程的应用名：



#### 4.新建hanlder(controller/HelloWorldController)

|  |
| --- |
| **package** handler;  **import** javax.servlet.http.HttpServletRequest;  **import** javax.servlet.http.HttpServletResponse;  **import** org.springframework.web.servlet.ModelAndView;  **import** org.springframework.web.servlet.mvc.Controller;  /\*\*  \* http://localhost:8080/SpringMVC/helloWord?username=liyan  \* **@author** liyan  \*  \*/  **public** **class** HelloWorldController **implements** Controller  {  /\*\*  \* ModelAndView视图和模型  \*  \*/  @Override  **public** ModelAndView handleRequest(HttpServletRequest request,  HttpServletResponse response) **throws** Exception  {  String name = request.getParameter("username");  ModelAndView modelAndView = **new** ModelAndView();    // request.setAttriubte("username", name);  modelAndView.addObject("username", name); // 设置数据模型  modelAndView.setViewName("hello"); // 设置视图名称hello    **return** modelAndView;  }  } |

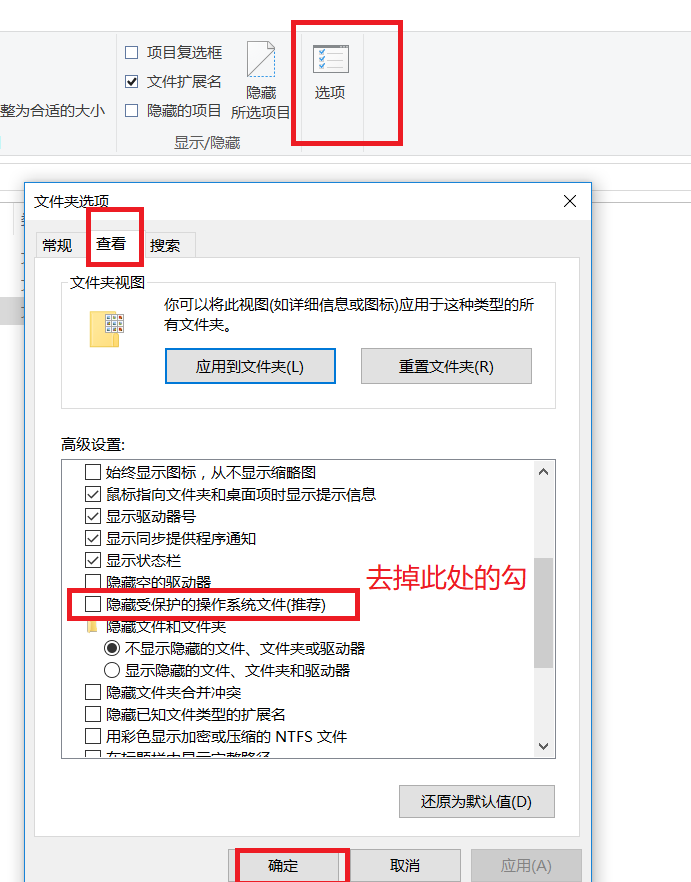
#### 5.新建WEB-INF/view/hello.jsp

|  |
| --- |
| <%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"ISO-8859-1"*%>  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  <html>  <head>  </head>    <body>  hello ${username} !!!  </body>  </html> |

#### 6.访问

|  |
| --- |
| http://localhost:8080/SpringMVC/helloWord?username=liyan |

显示出OS(操作系统)默认隐藏的文件



手动加入命名空间

## 4.springmvc核心配置文件

默认是在WEB-INF/ DispatcherServlet的名字-servlet.xml

springmvc-servlet.xml

自定义SpringMVC的核心配置文件

|  |
| --- |
| <!-- 自定义springmvc的核心配置文件 -->  <init-param>  <param-name>contextConfigLocation</param-name>  <!-- 告诉springmvc框架我们的核心配置文件在哪 -->  <param-value>classpath:springmvc.xml</param-value>  </init-param> |

## 5.springmvc中静态资源的访问

静态资源的访问: 图片/js/css

在springmvc框架中静态资源不能直接访问

### 1.访问WebRoot下静态资源

|  |
| --- |
| http://localhost:8080/SpringMVC/images/1.png  <!-- 访问WebRoot下静态资源 -->  <mvc:default-servlet-handler/>  <mvc:resources location=*"/"* mapping=*"/\*\*"*/> |

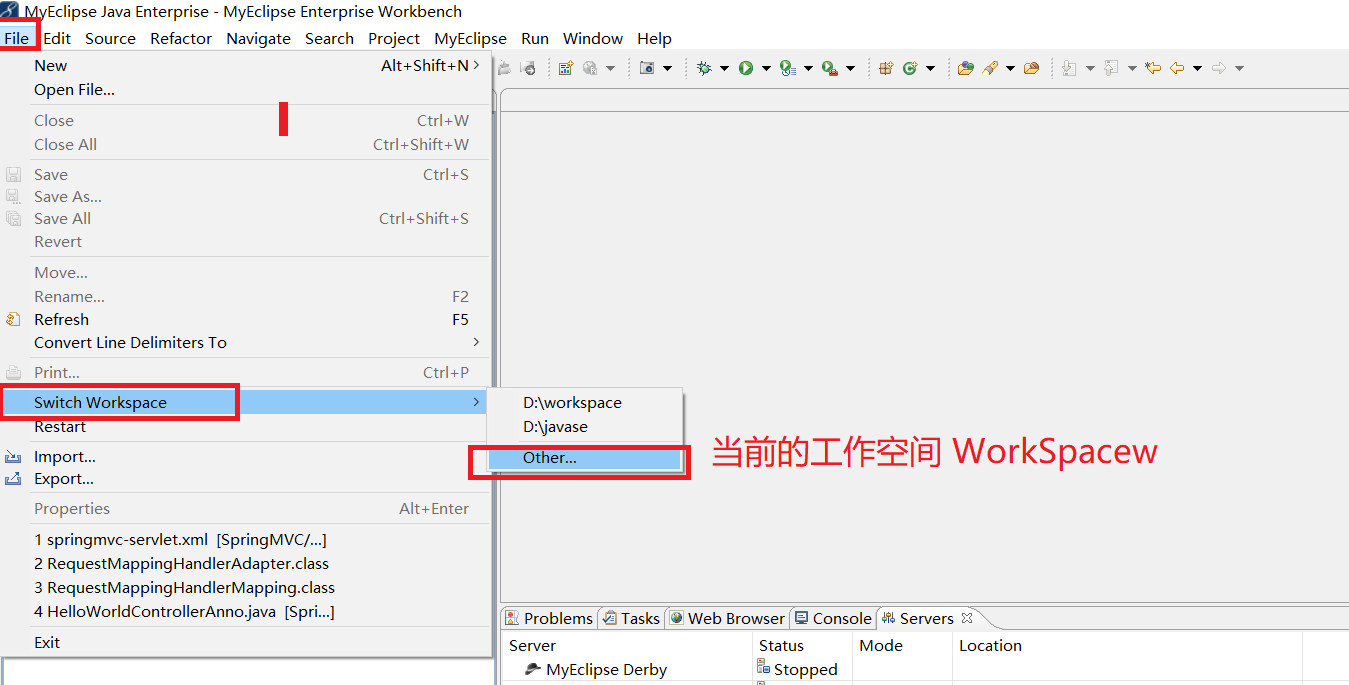
### 2.访问WEB-INF下静态资源

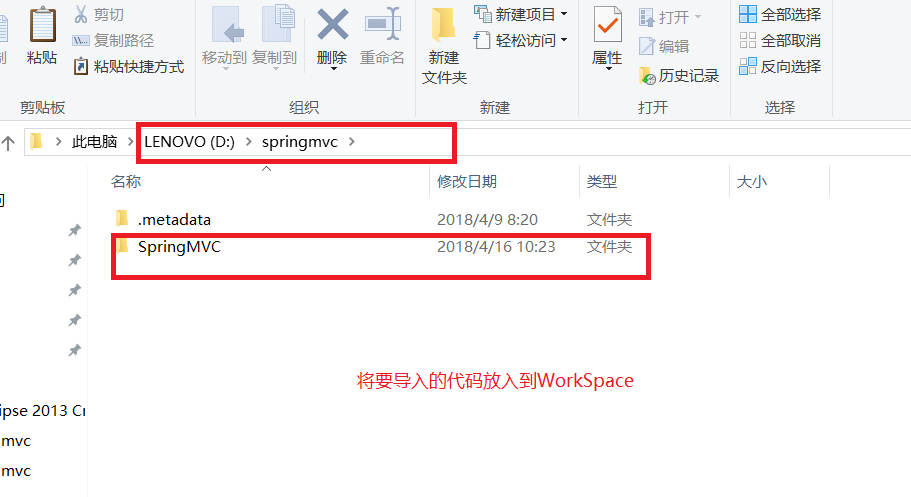
|  |
| --- |
| <!-- 访问WEB-INF下静态资源 --> |

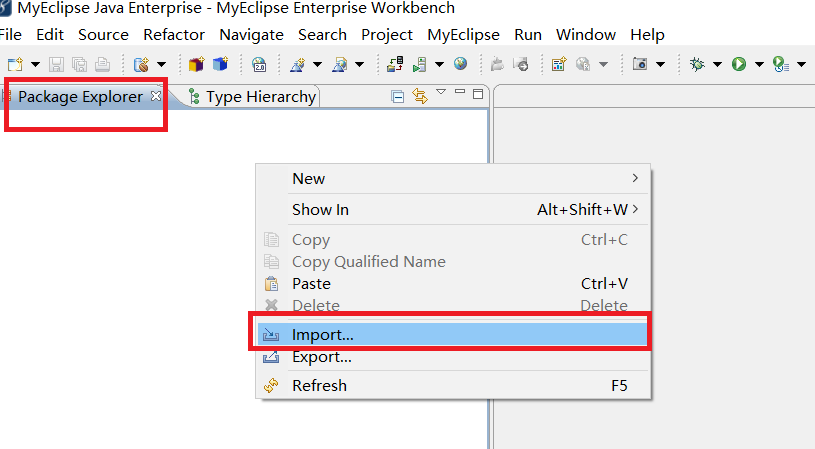
假设我们在WEB-INF/images/a/b/1.jpg

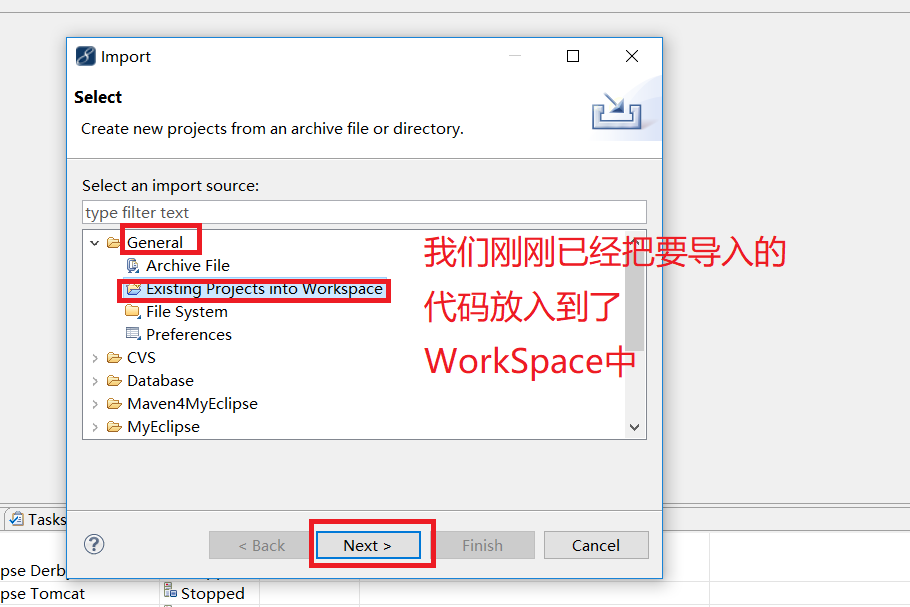
|  |
| --- |
| <mvc:resources location=*"/WEB-INF/images/"* mapping=*"/image/\*\*"*></mvc:resources> |

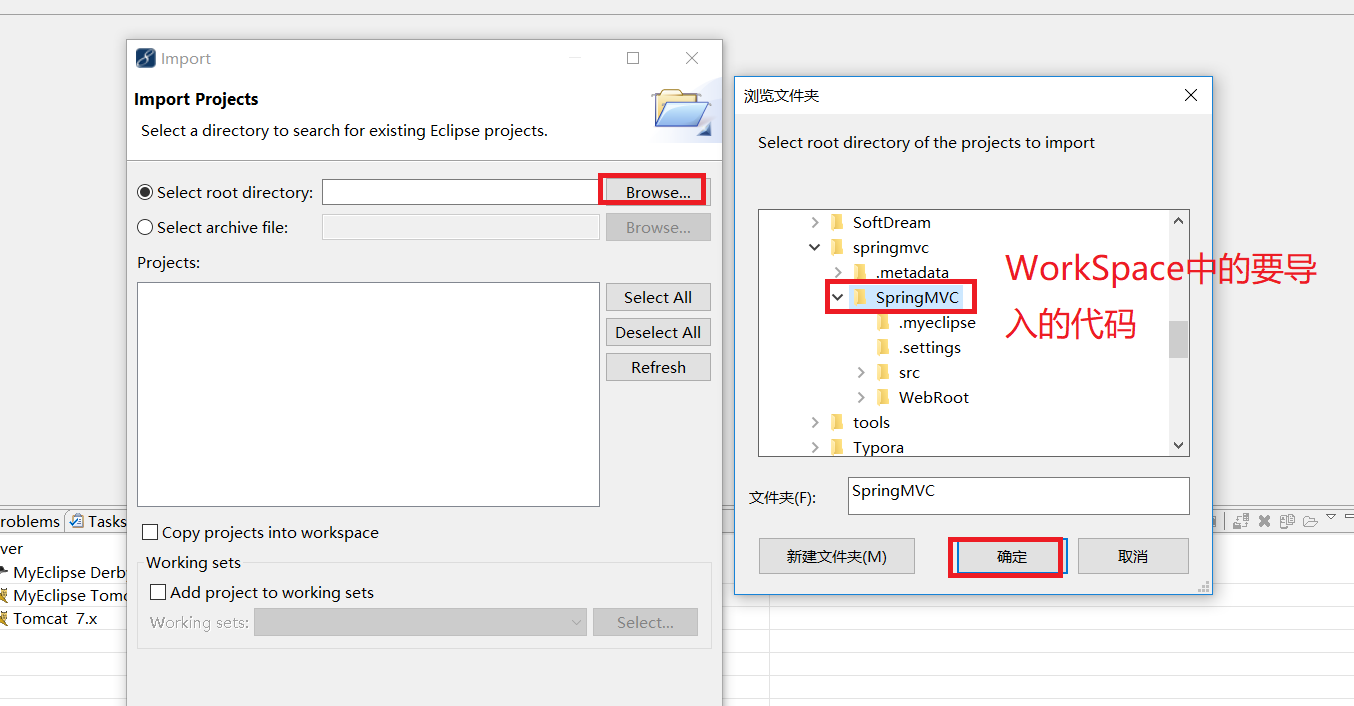
## 6.如何把本地工程导入到myeclipse中











直接点击Finsh即可

## 7.mvc中常用注解(重点)

### 1.@Controller ---> 标注此类是一个Handler

### 2.@RestController---> 和Controller一致(手机APP开发 ajax)

### 3.@RequestMapping

用在类上：相当于包的概念

java.util.Date

java.sql.Date

|  |
| --- |
| @Controller // 标注此类是一个Handler  @RequestMapping("/aaa")  **public** **class** HelloWorldControllerAnno |

用在方法上：就是要访问此方法的路径

|  |
| --- |
| // http://应用名/aaa/helloWord  @RequestMapping("/helloWord")  **public** ModelAndView sayHello(String username)  {  ModelAndView modelAndView = **new** ModelAndView();        **return** modelAndView;  } |

### 4.@RequestParam

表示方法中的参数来源于请求

### 5.@PathVariable

表示方法中的参数来源于URL

|  |
| --- |
| <http://localhost:8080/SpringMVC/deleteUser?id=1>  http://localhost:8080/SpringMVC/deleteUser/1 |

### 6.@ModelAttribute

表示将请求的数据映射到一个Javabean类(User)中

User

{

private String username;

private String password;

}

public void addUser(User user)

{

}

### 7.@ResponseBody

返回json数据 ajax

{

“username” : “liyan”,

“password” : “123456”

}

### 8.@ExceptionHandler

当系统出现某个异常时跳转到异常统一界面中去

### 9.@InitBinder(了解) 类型转换 spring 2.5 转换器(在高版本的Spring中用配置转换器(Conversion)的方式实现类型转换 Date ---> String)

### 10.@SessionAttributes

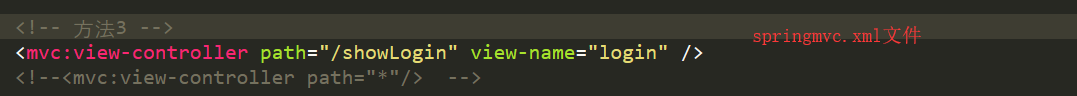
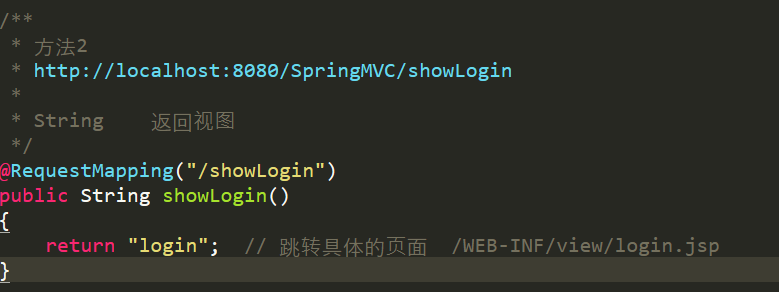
对象在Session作用域中

## 8.Handler中方法的返回值(4种返回值)

ModelAndView 返回数据模型与视图



String 返回视图

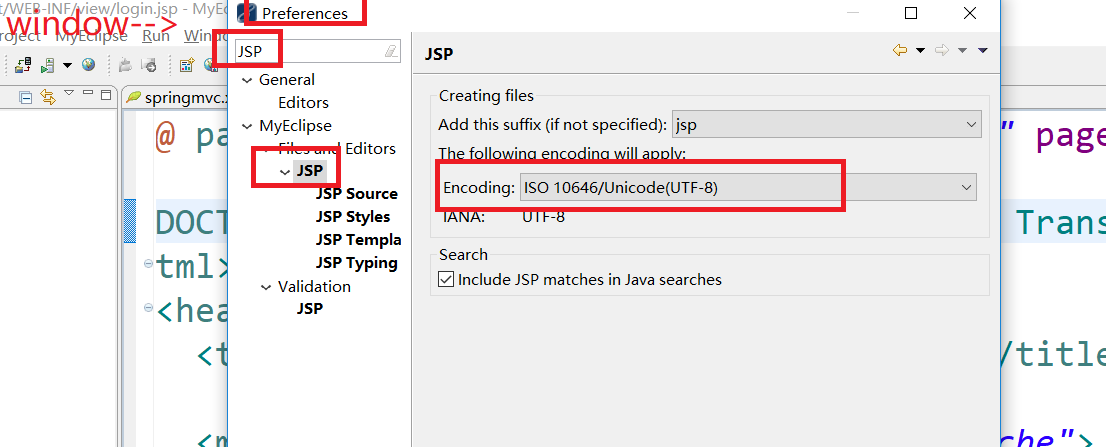


|  |
| --- |
| 重定向  转发 |

void 返回视图(把url当成视图名称 工作中很少用)

Object 返回数据模型(一般在发送ajax请求时使用)

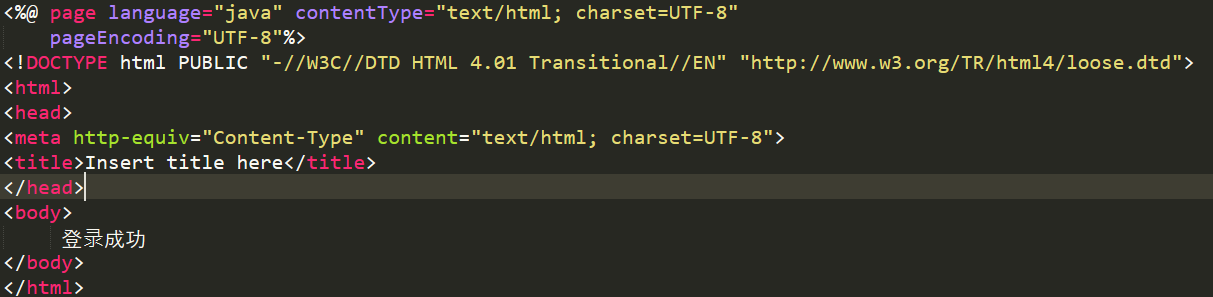
修改新建的JSP页面的默认编码值:



## 9.直接跳转页面中的方法

|  |
| --- |
| **package** handler;  **import** org.springframework.stereotype.Controller;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** org.springframework.web.servlet.ModelAndView;  /\*\*  \* 直接跳转页面的三种方法  \* **@author** liyan  \*  \*/  @Controller  **public** **class** LoginController  {  /\*\*  \* 方法1  \* http://localhost:8080/SpringMVC/showLogin  \* ModelAndView 返回数据模型与视图  \*/  /\*@RequestMapping("/showLogin")  public ModelAndView showLogin()  {  ModelAndView modelAndView = new ModelAndView();    modelAndView.setViewName("login"); // 设置视图名称    return modelAndView;  }\*/    /\*\*  \* 方法2  \* http://localhost:8080/SpringMVC/showLogin  \*  \* String 返回视图  \*/  /\*@RequestMapping("/showLogin")  public String showLogin()  {  return "login"; // 跳转具体的页面 /WEB-INF/view/login.jsp  }\*/    /\*\*  \* 方法3  \* http://localhost:8080/SpringMVC/showLogin  \*/  // <mvc:view-controller path="/showLogin" view-name="login" />  // 或者  // <mvc:view-controller path="\*"/>    /\*\*  \* void 返回视图(把url当成视图名称 工作中很少用)  \*/  /\* @RequestMapping("/showLogin")  public void showLogin()  {  System.out.println("1111111111");  }\*/ |

## 10.登录页面中的登录方法示例代码



## 11.2种风格的URL

### 1.ant风格的url

\* 星号表示单层目录

\*\* 2个星号 表示单层或者多层目录

? 单个字符的目录(不能没有)

### 2.Restful风格的url(企业中现在用的多) 结合@PathVariable(value=”restful风格的变量名”)(此处为重点)

{变量名}

|  |
| --- |
| **package** handler;  **import** org.springframework.stereotype.Controller;  **import** org.springframework.web.bind.annotation.PathVariable;  **import** org.springframework.web.bind.annotation.RequestMapping;  /\*\*  \* http://localhost:8080/SpringMVC/path  \*  \* ant风格的url  \* \* 星号表示单层目录  \* \*\* 2个星号 表示单层或者多层目录  \* ? 单个字符的目录(不能没有)  \*  \* Restful风格的url(企业中现在用的多)  \* {变量名}  \*  \* **@author** liyan  \*  \*/  @Controller  @RequestMapping("/path")  **public** **class** PathController  {  // http://localhost:8080/SpringMVC/path/m1/  // http://localhost:8080/SpringMVC/path/m1/a  // http://localhost:8080/SpringMVC/path/m1/bbbbb  @RequestMapping("/m1/\*")  **public** **void** m1()  {  System.*out*.println("m1()......");  }    // http://localhost:8080/SpringMVC/path/m2/  // http://localhost:8080/SpringMVC/path/m2/a  // http://localhost:8080/SpringMVC/path/m2/a/b/c  @RequestMapping("/m2/\*\*")  **public** **void** m2()  {  System.*out*.println("m2()......");  }    // http://localhost:8080/SpringMVC/path/m3/a  @RequestMapping("/m3/?")  **public** **void** m3()  {  System.*out*.println("m3()......");  }    // 当方法中的形参的名字 和 {变量名} 不一致时要用@PathVariable(value="restful中的变量名")  // http://localhost:8080/SpringMVC/path/m4/admin  @RequestMapping("/m4/{username}")  **public** **void** m4(@PathVariable String name) //String username来源于URL的{username}位置  {  System.*out*.println("m4()......" + name);  }    // http://localhost:8080/SpringMVC/path/m5/1/liyan  @RequestMapping("/m5/{id}/{name}")  **public** **void** m5(@PathVariable(value="id") String aa, @PathVariable(value="name") String bb) //String username来源于URL的{username}位置  {  System.*out*.println("m5()...... id:" + aa + " name:"+ bb);  }    // 3种ant风格混用  // http://localhost:8080/SpringMVC/path/m6/a/kky/aaa/bbb/ccc/abc  @RequestMapping("/m6/\*/kk?/\*\*/abc")  **public** **void** m6()  {  System.*out*.println("m6()......");  }    // ant风格和restful风格混用  // http://localhost:8080/SpringMVC/path/m7/aaa/liyan  @RequestMapping("/m7/\*/{username}")  **public** **void** m7(@PathVariable String username)  {  System.*out*.println("m7()......" + username);  }    } |

## 12.ajax

**A*synchronous*J*avascript And*X*ML 异步的 js 和xml***

**理解同步和异步**

**同步: 同一个时间点只有一个线程操作 共享资源**

### 1.在springmvc框架中如果使用ajax 异步发送请求

#### 1. 返回普通字符串 dataType : 'text'(了解)

##### 1.新建一张表 t\_user

|  |
| --- |
| C:\Users\Jack\AppData\Roaming\Tencent\Users\983702292\QQ\WinTemp\RichOle\SE%$H~JTA}H{G19I%]9Q~RL.png |

构建jsp页面：表单+jQuery事件

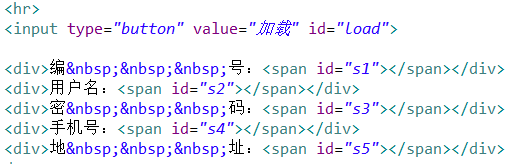
C:\Users\Jack\AppData\Roaming\Tencent\Users\983702292\QQ\WinTemp\RichOle\K6T4_`(N__99803[$BUFOUV.png

·

Controller逻辑：



#### 2. 返回json dataType : 'json'(掌握)

1. jsp页面

2.jQuery代码逻辑

3.User javaBean(见User.java)

**import** java.io.Serializable;

/\*\*

\* 实体bean

\* DB中的一张表 ---> Java实体bean（pojo/JavaBean）

\* **@author** Jack

\*

\*/

**public** **class** User **implements** Serializable{

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**private** Integer id;

**private** String username;

**private** String password;

**private** String phone;

**private** String address;

//alt + shift + s 方法构造器

**public** User() {}

**public** User(Integer id, String username, String password, String phone, String address) {

**super**();

**this**.id = id;

**this**.username = username;

**this**.password = password;

**this**.phone = phone;

**this**.address = address;

}

**public** Integer getId() {

**return** id;

}

**public** **void** setId(Integer id) {

**this**.id = id;

}

**public** String getUsername() {

**return** username;

}

**public** **void** setUsername(String username) {

**this**.username = username;

}

**public** String getPassword() {

**return** password;

}

**public** **void** setPassword(String password) {

**this**.password = password;

}

**public** String getPhone() {

**return** phone;

}

**public** **void** setPhone(String phone) {

**this**.phone = phone;

}

**public** String getAddress() {

**return** address;

}

**public** **void** setAddress(String address) {

**this**.address = address;

}

@Override

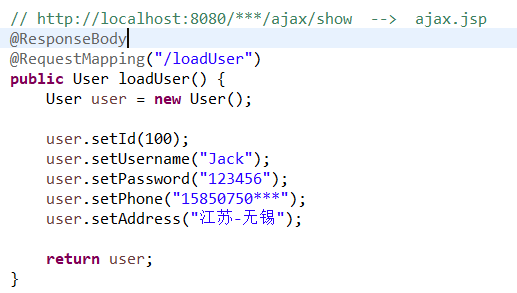
**public** String toString() {

**return** "User [id=" + id + ", username=" + username + ", password=" + password + ", phone=" + phone

+ ", address=" + address + "]";

}

}

4.Controller内部逻辑

\*\*注：

1. 更改jsp页面不用重启服务器
2. 加入jar、更改Controller之后一定要重新部署环境

SpringMVC标签库：

<!-- 引入标签库: Spring form标签库 -->

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

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

<title>Spring form标签库演示</title>

</head>

<body>

<!--

<form method="post" action="">

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

</form>

-->

<!-- html form表单的默认提交方式 method="get" 实际开发中99.9 是用post method="post"-->

<f:form commandName="formCommand" action="${pageContext.request.contextPath }/form/submit">

<!--

java.lang.IllegalStateException:

Neither BindingResult nor plain target object for

bean name 'command' available as request attribute -->

<!--

path:属性即表示name属性,也表示value属性

name属性: 表单提交数据的key 后台取值服务

value属性: 设置此标签的默认值

-->

用户名:<f:input path="name" /> <br/>

<!-- showPassword="true" 显示密码属性 -->

密&nbsp;&nbsp;&nbsp;码:<f:password path="password" showPassword="true" /> <br/>

<!-- 修改/删除用户信息用 -->

<f:hidden path="id" />

爱好:

<!--

<f:checkbox path="favor" value="1" label="吃饭" />

<f:checkbox path="favor" value="2" label="打游戏" />

<f:checkbox path="favor" value="3" label="敲代码" /> <br/>

-->

<!-- f:checkboxes：数据要求动态加载出来 -->

<f:checkboxes items="${favorsMap }" path="favor"/> <br/>

性别: <!-- f:radiobuttons：数据要求动态加载出来 -->

<f:radiobuttons items="${userList }" path="gender" itemValue="id" itemLabel="username"/> <br/>

月份: <f:select path="month">

<!-- <f:option value="--请选择--"></f:option> -->

<f:options items="${monthList }" />

</f:select> <br/>

<f:button>提交</f:button>

</f:form>

</body>

</html>

需要定义FormCommand对象：

public class FormCommand

{

// form.jsp页面中的用户名:<f:input path="name" />相对应

private String name;

private String password;

private Integer id; // int(默认值:0) VS Integer(工作中 默认值:null)

private List<String> favor;

private String gender;

private String month;

//…以下皆为get/set方法

}

FormController控制器：

@Controller

@RequestMapping("/form")

public class FormController

{

@RequestMapping("/show")

public String show(FormCommand fromCommand)

{

// 设置 form.jsp页面中的所有默认值

fromCommand.setName("xiaofeng");

fromCommand.setPassword("123456");

fromCommand.setId(100);

// 爱好有多个: 数组(不能动态的添加元素) 或者 集合(可以动态的添加数据)

List<String> favorList = new ArrayList<String>();

// 往List中添加值

favorList.add("1");

favorList.add("2");

favorList.add("3");

favorList.add("4");

fromCommand.setFavor(favorList);

fromCommand.setGender("2"); // 默认男 和 form.jsp页面中的 <f:radiobutton path="gender" value="1" label="男" /> 相对应

fromCommand.setMonth("2");

return "form";

}

// @ModelAttribute 在Controller中的所有方法执行之前调用 加载数据用

// @ModelAttribute(key) 和 form.jsp中的<f:checkboxes items="${key }" path="favors"/> items属性值相对应

@ModelAttribute("favorsMap")

public Map<String, String> loadFavors()

{

Map<String, String> favors = new TreeMap<String, String>();

// 往Map中添加数据用put

favors.put("1", "吃");

favors.put("2", "喝");

favors.put("3", "嫖");

favors.put("4", "赌");

return favors;

}

@ModelAttribute("userList")

public List<User> LoadUser()

{

List<User> userList = new ArrayList<User>();

User user1 = new User();

user1.setId(1);

user1.setUsername("admin");

User user2 = new User();

user2.setId(2);

user2.setUsername("alice");

userList.add(user1);

userList.add(user2);

return userList;

}

@ModelAttribute("monthList")

public List<String> loadMonth()

{

List<String> mList = new ArrayList<String>();

mList.add("1");

mList.add("2");

mList.add("3");

mList.add("4");

return mList;

}

@RequestMapping("/submit")

public String submit(FormCommand fromCommand)

{

System.out.println("name:" + fromCommand.getName());

System.out.println("password:" + fromCommand.getPassword());

System.out.println("hidden:" + fromCommand.getId());

System.out.println("favorList:" + fromCommand.getFavor());

System.out.println("gender:" + fromCommand.getGender());

System.out.println("month:" + fromCommand.getMonth());

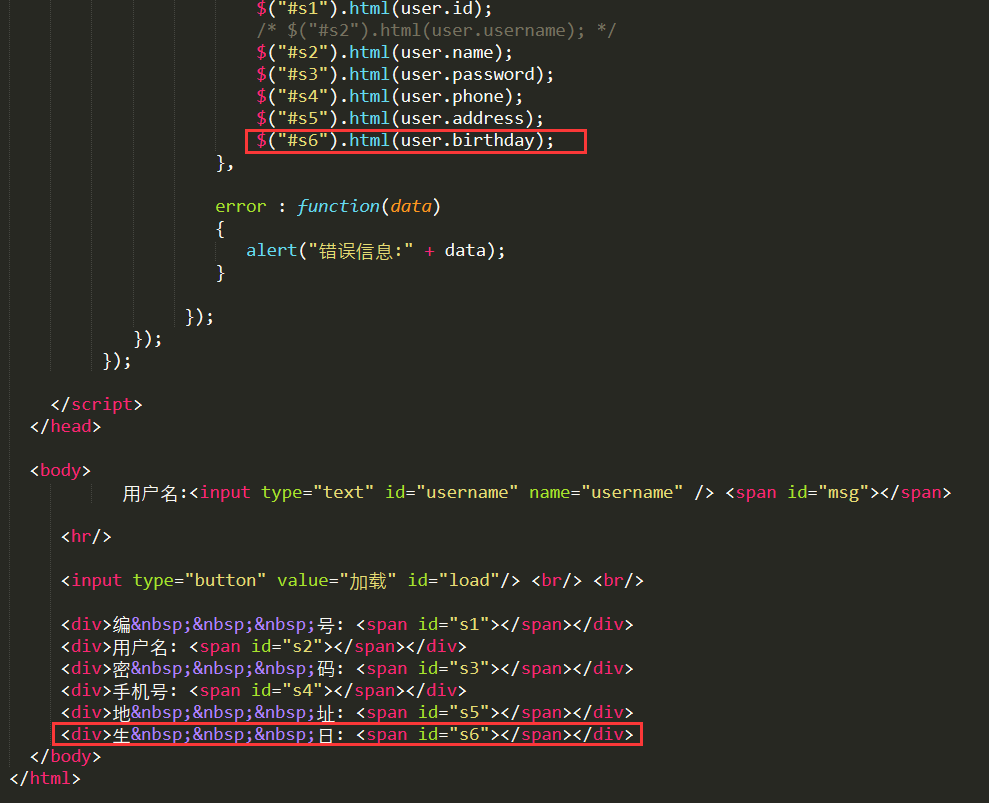
return "redirect:show"; // 重定向到form.jsp页面

}

}

使用Ajax和Json注释显示日期：

在ajax.jsp内添加标签：



Bean类User内添加注释：

public class User implements Serializable // 序列化接口

{

//…

private Integer id;

// 表示在返回json数据时,将 private String username;字段去一个别名 name

@JsonProperty("name")

private String username;

// 表示返回json数据时,忽略此字段 起到密码保护作用

@JsonIgnore

private String password;

private String phone;

private String address;

@JsonFormat(pattern="yyyy-MM-dd HH:mm:ss", timezone="GMT+8")

private Date birthday;

//…以下是构造函数和get&set函数

}

Date-pattern选择：



# 进行异常处理：

1. 异常处理Controller：ExceptionController.java

/\*\*

\* Throwable

\* Exception 异常

\* Exception 编译器异常

\* RuntimeException 运行时异常（避免）

\* Error 错误（一般不需要关心）

\* 处理异常的2种方案

\* throws IOException 抛异常 throws

\* try...catch...finally 捕获异常

\*/

@Controller

@RequestMapping("/exception")

public class ExceptionController {

//.../exception/test

@RequestMapping("/test")

public void test() throws Exception{

System.out.println("ExceptionController类中的test()...");

//throw new 新的自定义异常

throw new StockDeficiencyException("库存不足。");

}

/\*

//局部异常处理：@ExceptionHandler注解（要处理的而异常类.class）

//用ExceptionHandler注解来标识异常方法 StockDeficiencyException

//@ExceptionHandler默认处理的是java.lang.Exception

//对应的value应该与抛出的异常类一致

@ExceptionHandler(value=Exception.class)

public ModelAndView exceptionHandler(Exception e) {

System.out.println("ExceptionController类中的 exceptionHandler()......");

//第一个参数：标识视图名称

//第二个参数：存储异常数据模型key

return new ModelAndView("exception", "ex", e);

}\*/

}

1. 自定义异常：StockDeficiencyException.java

/\*\*

\* 库存不足 自定义异常

\* 1.自定义异常类名继承：Exception或RuntimeException

\* **@author** Jack

\*

\*/

**public** **class** StockDeficiencyException **extends** Exception{

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**public** StockDeficiencyException() {

**super**();

// **TODO** Auto-generated constructor stub

}

**public** StockDeficiencyException(String message, Throwable cause, **boolean** enableSuppression,

**boolean** writableStackTrace) {

**super**(message, cause, enableSuppression, writableStackTrace);

// **TODO** Auto-generated constructor stub

}

**public** StockDeficiencyException(String message, Throwable cause) {

**super**(message, cause);

// **TODO** Auto-generated constructor stub

}

**public** StockDeficiencyException(String message) {

**super**(message);

// **TODO** Auto-generated constructor stub

}

**public** StockDeficiencyException(Throwable cause) {

**super**(cause);

// **TODO** Auto-generated constructor stub

}

}

1. 构建跳转页面：excep.jsp

<%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"*

pageEncoding=*"UTF-8"*%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"*>

<title>Insert title here</title>

</head>

<body>

<h1>异常信息：${ex}</h1>

</body>

</html>

1. 用全局异常处理代替异常处理：GlobalExceptionHandler.java

/\*\*

\* 全局异常系统类（整个SpringMVC系统都能用）

\* (1) **@ControllerAdvice** 来标识此类是用来处理全局异常的

\* (2) 并且想要使用全局异常处理 在SpringMVC的核心配置文件中必须要有<mvc:annotion-driver>

\* (3) 在SpringMVC的核心配置文件中配置扫描包：<context:component-scan base-package="globalexceptionhandler" />

\* **@author** Jack

\*

\*/

@ControllerAdvice

**public** **class** GlobalExceptionHandler {

@ExceptionHandler

**public** String exceptionHandler(Exception e) {

System.***out***.println("GlobalException类中的exceptionHandler()......");

**return** "error";

}

}

5.构建error.jsp页面：

<%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"*

pageEncoding=*"UTF-8"*%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"*>

<title>Insert title here</title>

</head>

<body>

<h1>服务器忙，请稍后重试...</h1>

</body>

</html>

6.更改dispatcher-servlet.xml静态配置：

<mvc:annotation-driven/>

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

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

\*注意：配置全局异常处理器之后异常将不会被内部异常处理器拦截。