# SpringMVC框架

## SpringMVC的基本概念

### 三层架构和MVC

#### 三层架构

开发服务器端程序有两种形式：C/S架构.B/S架构

java语言基本上都是开发B/S架构的程序，B/S架构有分成了三层架构

1. 表现层：WEB层，用来和客户端进行数据交互。表现层一般会采用MVC的设计模型
2. 业务层：处理公司具体的业务逻辑的
3. 持久层：用来操作数据库的

#### MVC模型

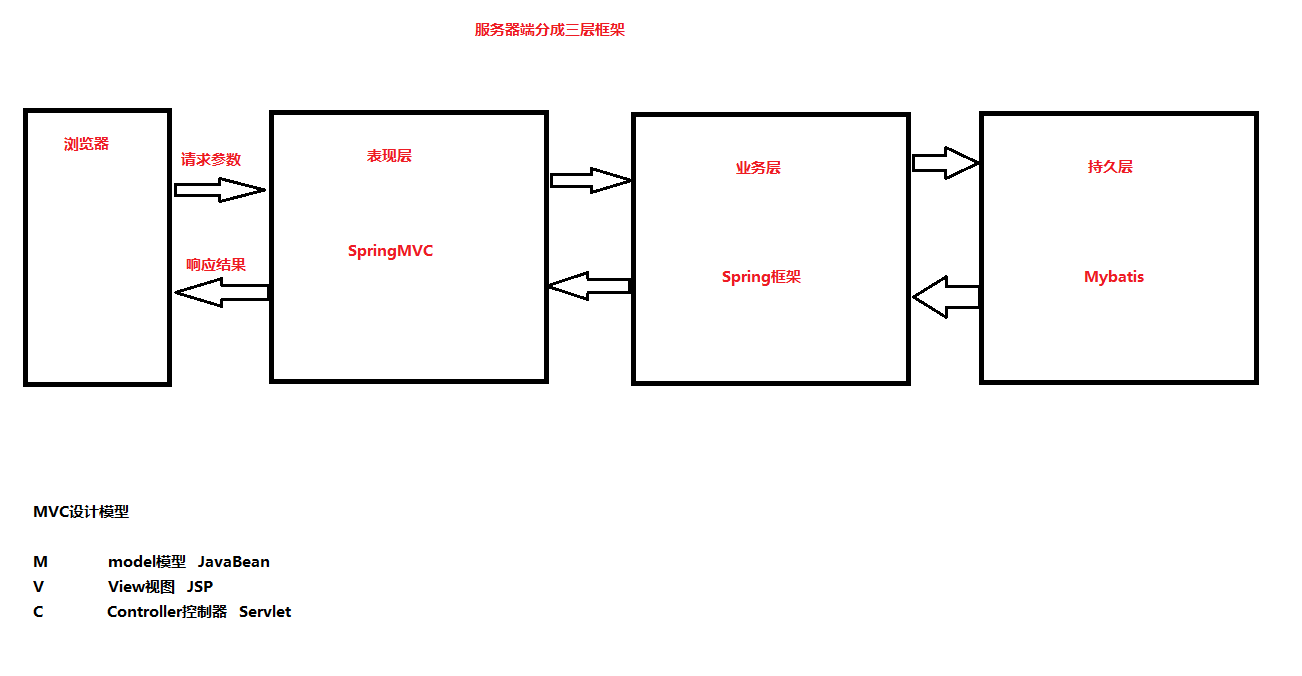
Model View Controller

模型 视图 控制器

Model：数据模型，javabean的类，用来进行数据封装

View：指JSP、HTML用来展示数据给用户

Controller：用来接收用户的请求，整个流程的控制器。用来进行数据校验等。

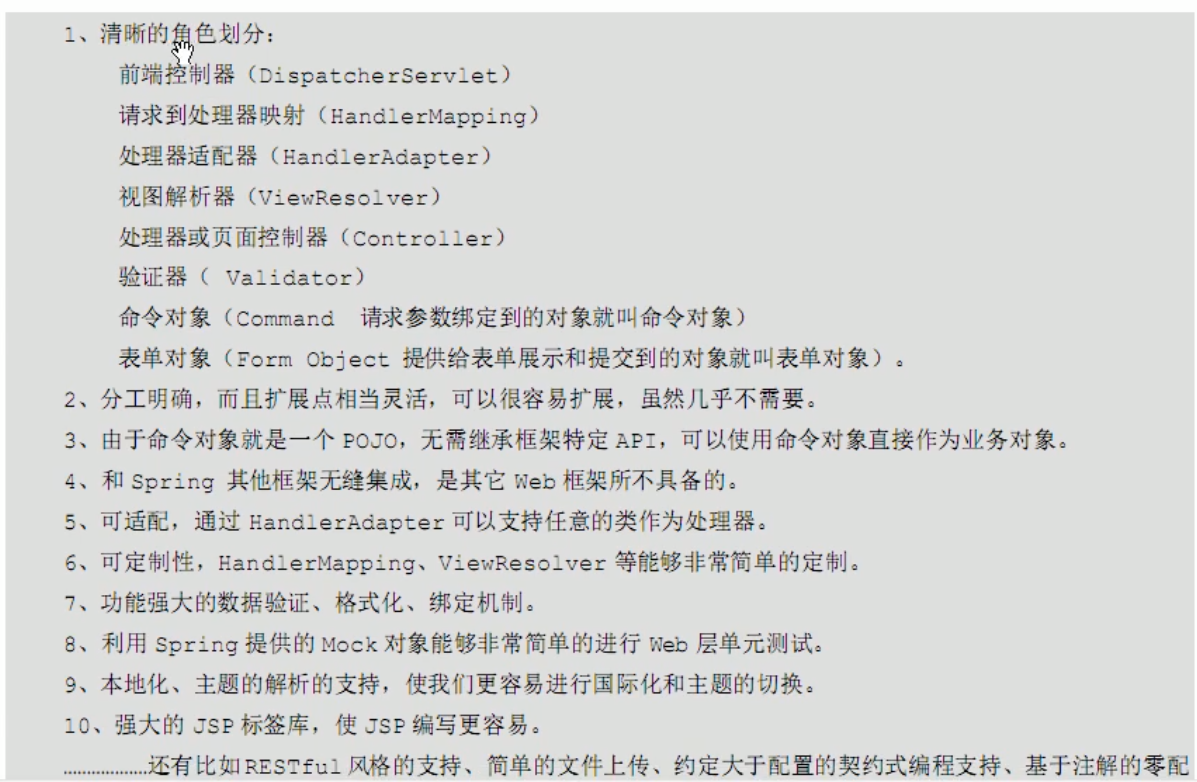


### SpringMVC概述

#### **什么是SpringMVC**

SpringMVC是一种基于Java的实现MVC设计模型的请求驱动类型的轻量级Web框架，属于SpringFrameWork的后续产品，已经融合在Spring WebFlow里面。Spring框架提供了构建web应用程序的全功能MVC模块，使用Spring可插入的MVC架构，从而在使用Spring进行WEB开发时，可选择使用Spring的SpringMVC框架或继承其他MVC开发框架，如Struts1，Struts2等。SpringMVC全面超越Struts2.支持RESTful编程风格

#### SpringMVC的优势

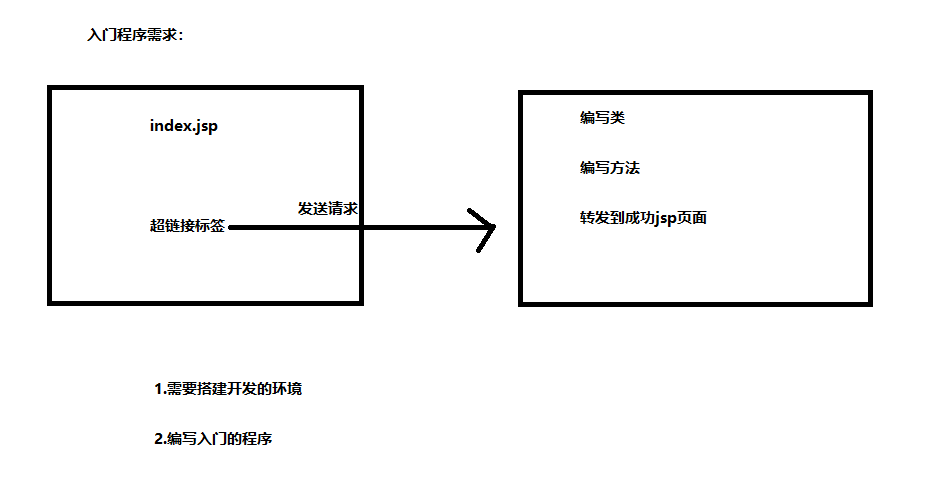


#### 比较SpringMVC和Struts2

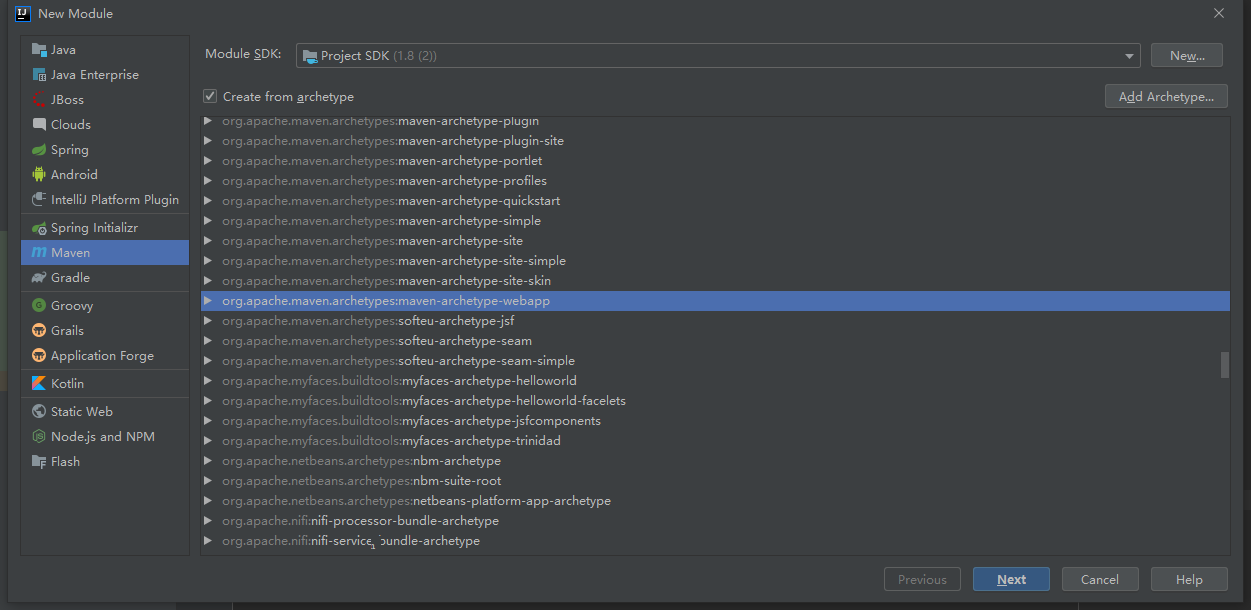


## SpringMVC的入门

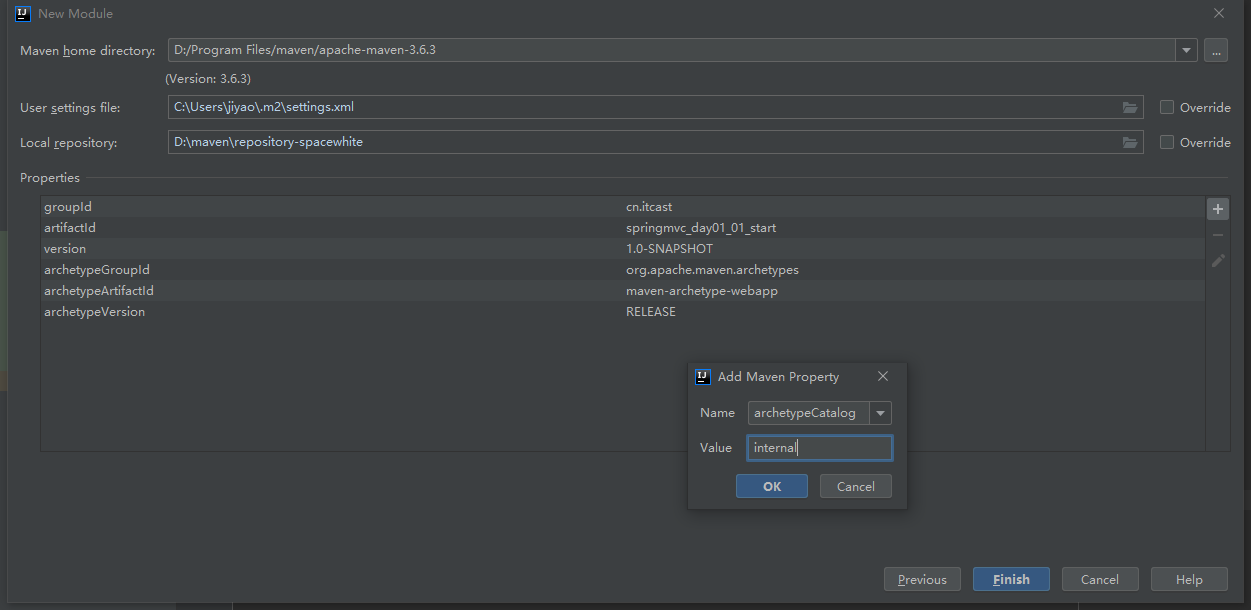
### SpringMVC的入门案例



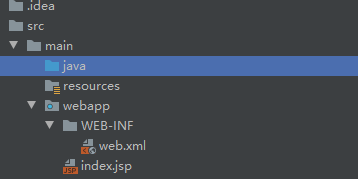
#### 搭建环境

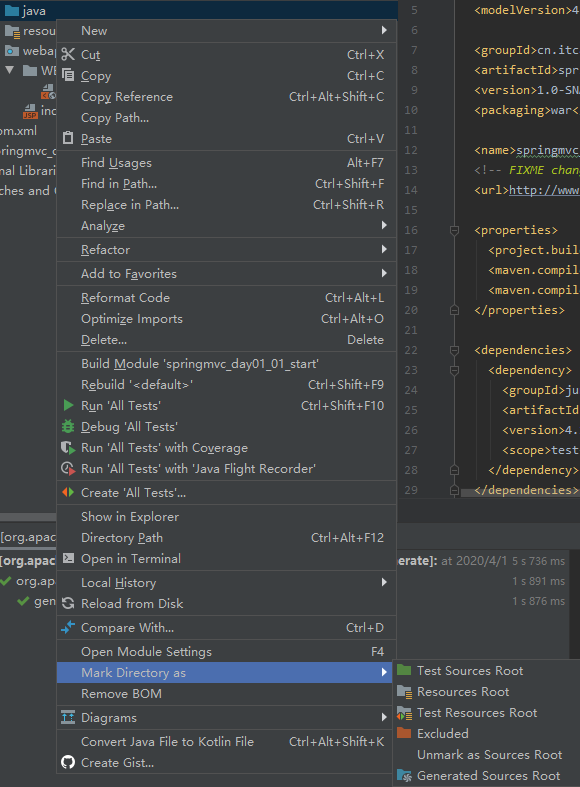


##### 解决项目构建过慢



##### 完善项目结构





##### 配置文件pom.xml

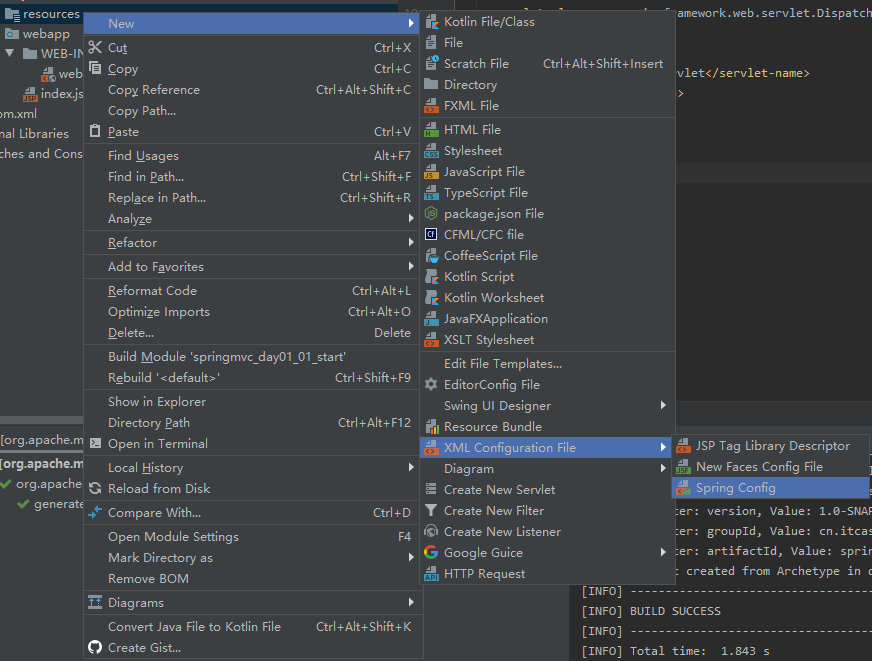
<?xml version="1.0" encoding="UTF-8"?>  
  
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
  
 <groupId>cn.itcast</groupId>  
 <artifactId>springmvc\_day01\_01\_start</artifactId>  
 <version>1.0-SNAPSHOT</version>  
 <packaging>war</packaging>  
  
 <name>springmvc\_day01\_01\_start Maven Webapp</name>  
 <!-- *FIXME change it to the project's website* -->  
 <url>http://www.example.com</url>  
  
 <properties>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 <maven.compiler.source>1.8</maven.compiler.source>  
 <maven.compiler.target>1.8</maven.compiler.target>  
 <spring.version>5.0.2.RELEASE</spring.version>  
 </properties>  
  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>${spring.version}</version>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-web</artifactId>  
 <version>${spring.version}</version>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-webmvc</artifactId>  
 <version>${spring.version}</version>  
 </dependency>  
  
 <dependency>  
 <groupId>javax.servlet</groupId>  
 <artifactId>servlet-api</artifactId>  
 <version>2.5</version>  
 <scope>provided</scope>  
 </dependency>  
  
 <dependency>  
 <groupId>javax.servlet.jsp</groupId>  
 <artifactId>jsp-api</artifactId>  
 <version>2.2</version>  
 <scope>provided</scope>  
 </dependency>  
 </dependencies>  
  
 <build>  
 <finalName>springmvc\_day01\_01\_start</finalName>  
 <pluginManagement><!-- lock down plugins versions to avoid using Maven defaults (may be moved to parent pom) -->  
 <plugins>  
 <plugin>  
 <artifactId>maven-clean-plugin</artifactId>  
 <version>3.1.0</version>  
 </plugin>  
 <!-- see http://maven.apache.org/ref/current/maven-core/default-bindings.html#Plugin\_bindings\_for\_war\_packaging -->  
 <plugin>  
 <artifactId>maven-resources-plugin</artifactId>  
 <version>3.0.2</version>  
 </plugin>  
 <plugin>  
 <artifactId>maven-compiler-plugin</artifactId>  
 <version>3.8.0</version>  
 </plugin>  
 <plugin>  
 <artifactId>maven-surefire-plugin</artifactId>  
 <version>2.22.1</version>  
 </plugin>  
 <plugin>  
 <artifactId>maven-war-plugin</artifactId>  
 <version>3.2.2</version>  
 </plugin>  
 <plugin>  
 <artifactId>maven-install-plugin</artifactId>  
 <version>2.5.2</version>  
 </plugin>  
 <plugin>  
 <artifactId>maven-deploy-plugin</artifactId>  
 <version>2.8.2</version>  
 </plugin>  
 </plugins>  
 </pluginManagement>  
 </build>  
</project>

##### web.xml配置servlet前端控制器及加载springmvc.xml配置文件

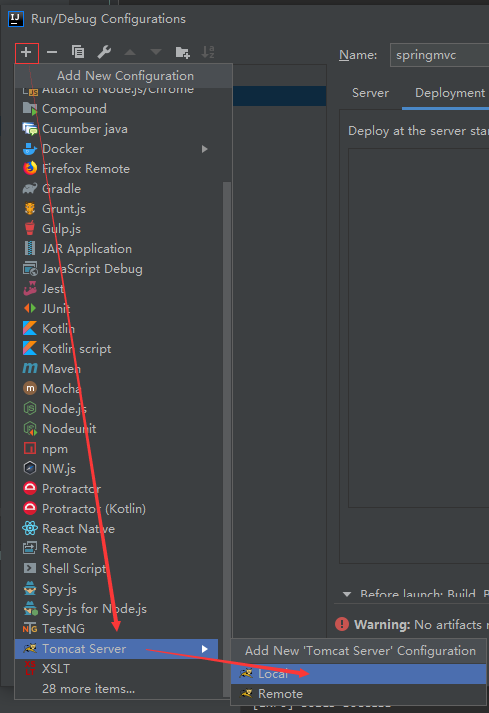
<!DOCTYPE web-app PUBLIC  
 "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"  
 "http://java.sun.com/dtd/web-app\_2\_3.dtd" >  
  
<web-app>  
 <display-name>Archetype Created Web Application</display-name>  
   
 <servlet>  
 <servlet-name>dispatcherServlet</servlet-name>  
 <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>  
 </servlet>

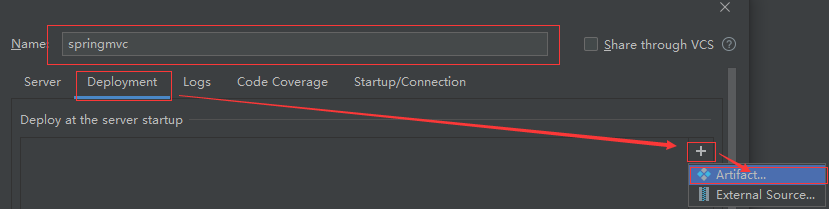
<init-param>  
 <param-name>contextConfigLocation</param-name>  
 <param-value>classpath:springmvc.xml</param-value>  
</init-param>  
 <servlet-mapping>  
 <servlet-name>dispatcherServlet</servlet-name>  
 <url-pattern>/</url-pattern>  
 </servlet-mapping>  
   
</web-app>

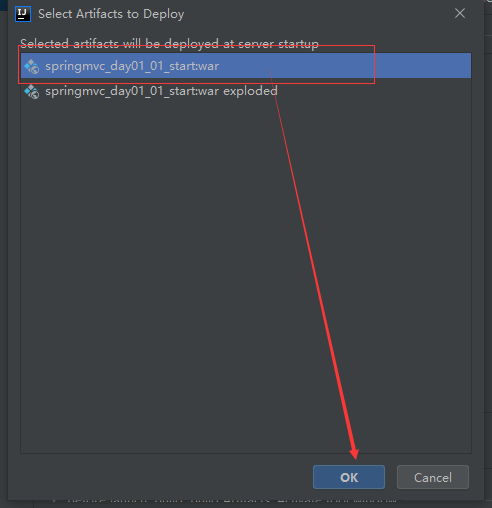
##### 在Resource目录下创建Spring Config配置文件

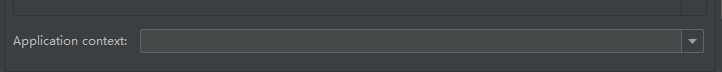


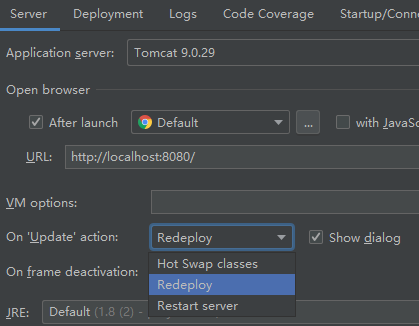
##### 配置服务器Tomcat



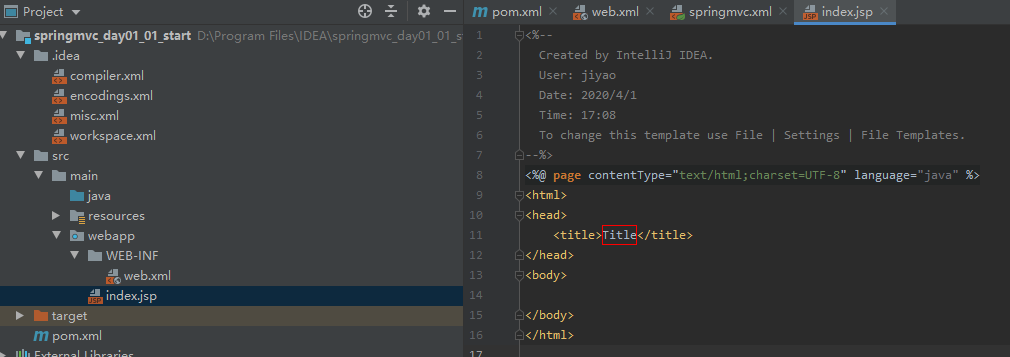








##### 重新创建index.jsp（为了有支持字节码头文件）



<%--  
 Created by IntelliJ IDEA.  
 User: jiyao  
 Date: 2020/4/1  
 Time: 17:08  
 To change this template use File | Settings | File Templates.  
--%>  
<%@ **page** contentType="text/html;charset=UTF-8" language="java" %>  
<html>  
<head>  
 <title>Title</title>  
</head>  
<body>  
  
 <h3>入门程序</h3>  
  
 <a href="">入门程序</a>  
  
</body>  
</html>

##### 配置springmvc.xml

<?xml version="1.0" encoding="UTF-8"?>  
<beans xmlns="http://www.springframework.org/schema/beans"  
 xmlns:mvc="http://www.springframework.org/schema/mvc"  
 xmlns:context="http://www.springframework.org/schema/context"  
 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.xsd  
 http://www.springframework.org/schema/mvc  
 http://www.springframework.org/schema/mvc/spring-mvc.xsd  
 http://www.springframework.org/schema/context  
 http://www.springframework.org/schema/context/spring-context.xsd">  
 <!--开启器注解扫描-->  
 <context:component-scan base-package="cn.itcast"/>

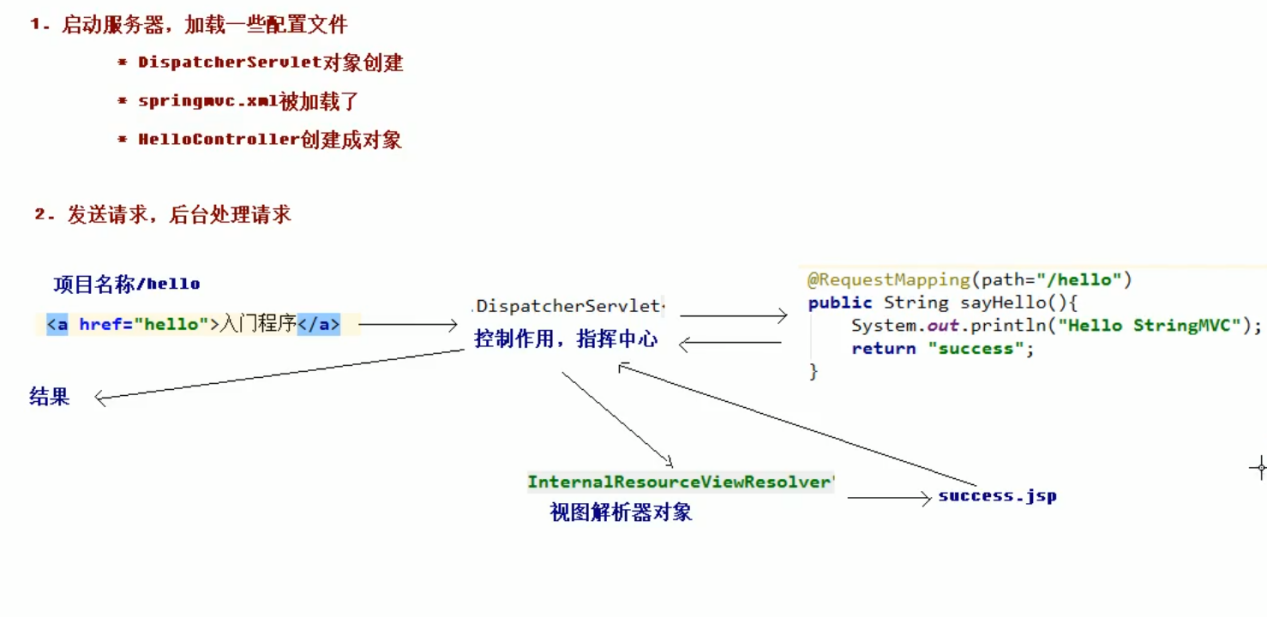
<!--视图解析器-->  
<bean name="internalResourceViewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver">  
 <property name="prefix" value="/WEB-INF/pages/"/>  
 <property name="suffix" value=".jsp"/>  
</bean>  
<!--开启SpringMVC框架注解的支持-->  
<mvc:annotation-driven/>  
</beans>

##### 在java下新建类

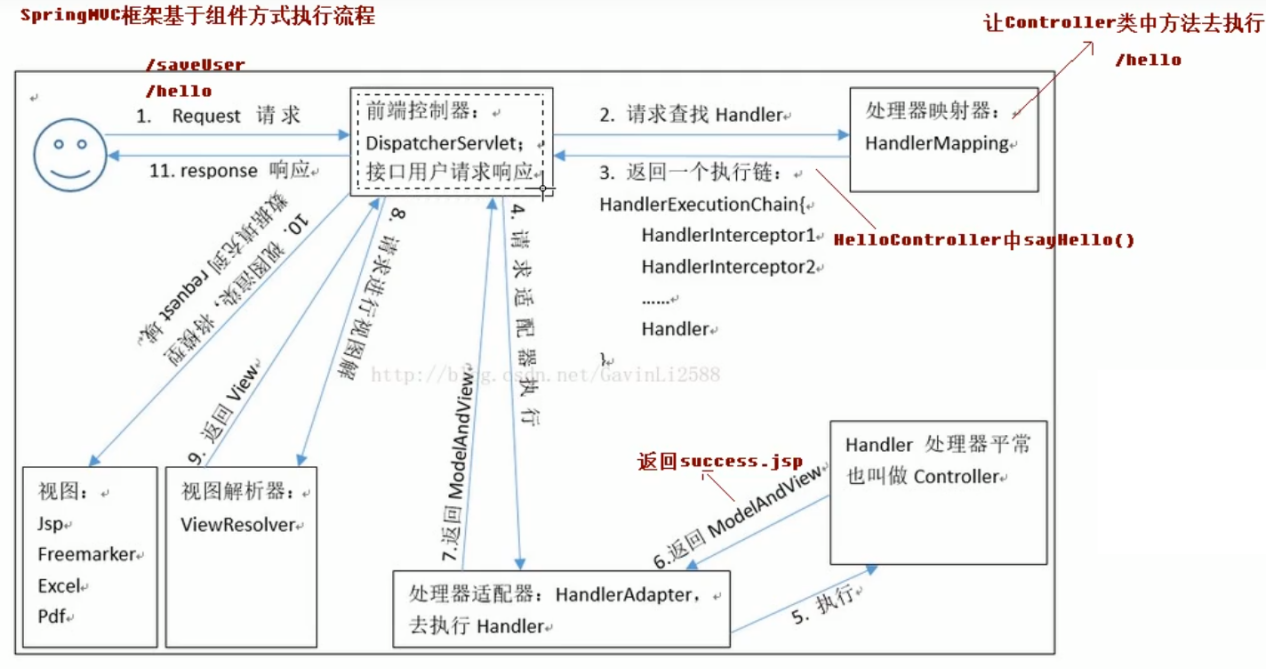
package cn.itcast.controller;  
  
import org.springframework.stereotype.Controller;  
import org.springframework.web.bind.annotation.RequestMapping;  
  
*/\*\*  
 \* 控制器类  
 \*/*@Controller  
public class HelloController {  
  
  
 @RequestMapping(path = "/hello")  
 public String sayHello(){  
 System.*out*.println("Hello StringMVC");  
 return "success";  
 }  
}

### 入门案例的执行过程及原理

#### 简易原理



#### 详细原理



### 入门案例中涉及的组件

#### DisptcherServlet:前端控制器

用户请求叨叨前端控制器，它就相当于mvc模式中的c，dispatcherServlet是整个流程控制的中心，由它调用其他组件用户的请求，dispatcherServlet的存在降低了组件之间的耦合性。

#### HandlerMappeing:处理器映射器

HandlerMapping负责根据用户请求找到Handler即处理器。SpringMVC提供了不同的映射器实现不同的映射方式，例如：配置文件方式，实现接口方式，注解方式等。

#### Handler

它就是我们开发中药编写的具体业务控制器。由DispatcherServlet把用户请求转发到Handler。由Handler对具体用户请求进行处理。

#### HandlerAdapter

通过HandlerAdapter对处理器进行执行，这是适配器模式的应用，通过扩展适配器可以对更多类型的处理器进行执行。

#### ViewResolver：视图解析器

View Resolver负责将处理结果生产View视图，View Resolver首先根据逻辑视图名解析成物理视图名即具体的页面地址，再生成View视图对象，最后对View进行渲染将处理结果通过页面展示给用户。

#### View：视图

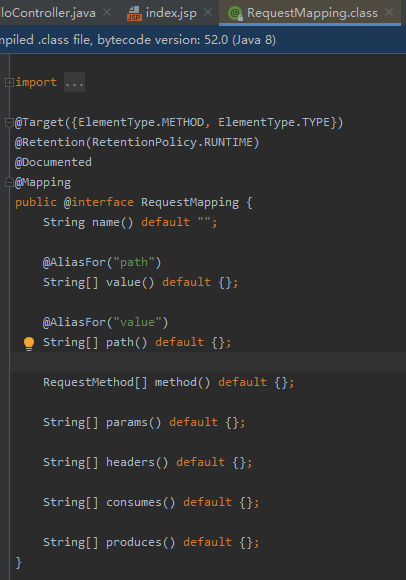
SpringMVC框架提供了很多的View视图类型的支持，包括：jstlView，freemarkerView，pdfView等。我们常用的视图就是jsp。

一般情况下需要通过页面标签或页面模板技术将模型数据通过页面展示给用户，需要由程序员根据业务需求开发具体的页面。

### RequestMapping注解

用于建立请求URL和处理请求方法之间的对应关系。

#### 用法



查看源码发现注解还可以用在类上

*/\*\*  
 \* 控制器类  
 \*/*@Controller  
@RequestMapping(path = "/user")  
public class HelloController {  
 */\*\*  
 \* 入门案例  
 \** ***@return*** *\*/* @RequestMapping(path = "/hello")  
 public String sayHello(){  
 System.*out*.println("Hello StringMVC");  
 return "success";  
 }  
  
 */\*\*  
 \* RequestMapping注解  
 \** ***@return*** *\*/* @RequestMapping(path = "/testRequestMapping")  
 public String testRequestMapping(){  
 System.*out*.println("测试RequestMapping注解。。。");  
 return "success";  
 }  
}

用在类上相当于加了一个目录

所以要在jsp页面中引用

<a href="user/testRequestMapping">RequestMapping注解</a>

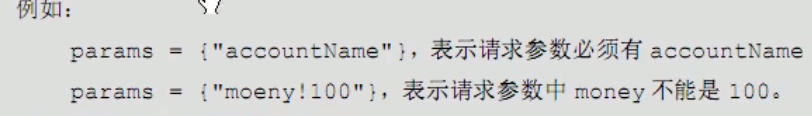
#### 属性

##### value=path

用于指定请求的url。也可以不写，默认为value=path。

##### params

用于指定限制参数的条件。它支持简单的表达式。请求要求参数的key和value必须和配置的一模一样。



用法需要传参

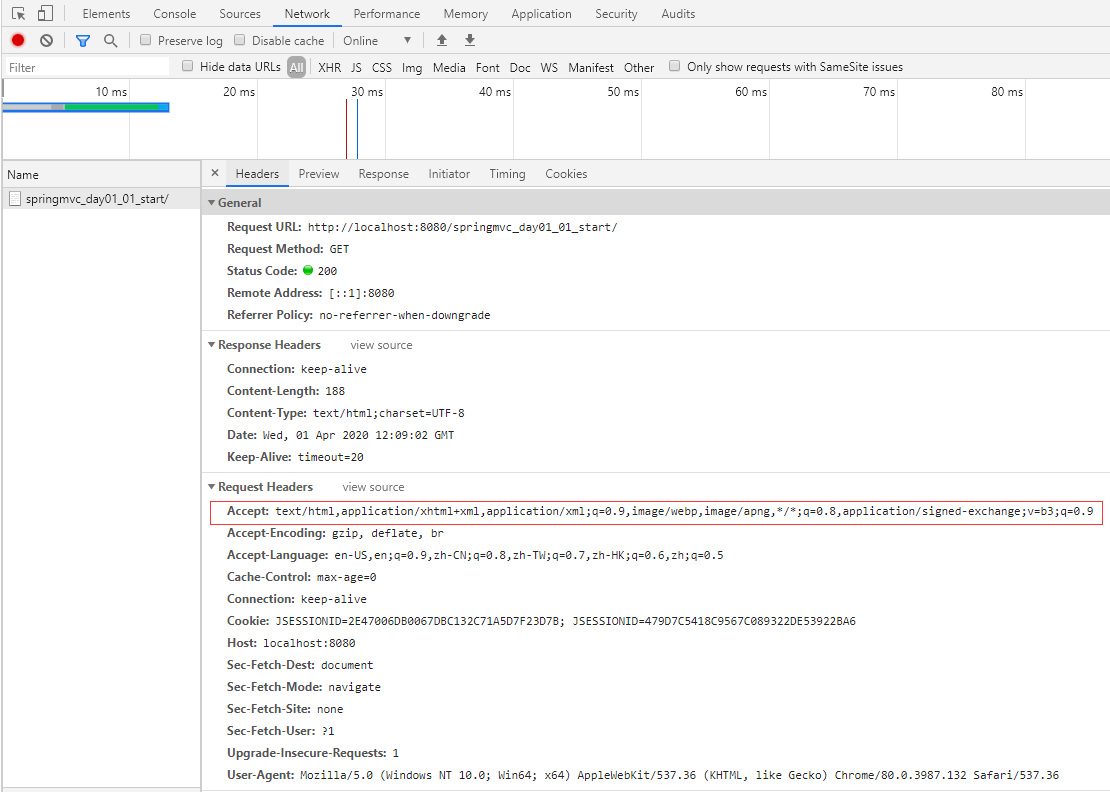
@RequestMapping(value = "/testRequestMapping",params = {"username=hehe"})

<a href="user/testRequestMapping?username=hehe">RequestMapping注解</a>

##### headers

用于指定限制请求消息头的条件

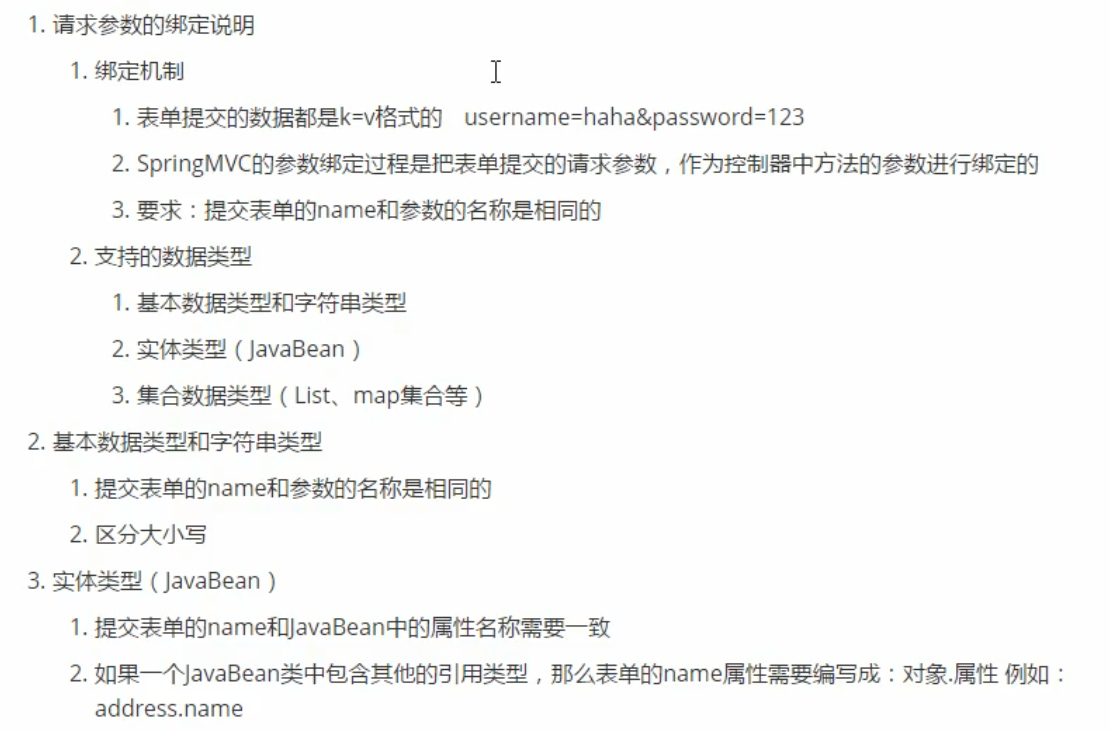
@RequestMapping(value = "/testRequestMapping",params = {"username=hehe"},headers = {"Accept"})



注意：

以上四个属性只要出现两个或者两个以上，他们的关系是与的关系

## 请求参数的绑定



##### method

当前方法可以接受什么样的请求方式，该方法为enum枚举类型

GET,  
HEAD,  
POST,  
PUT,  
PATCH,  
DELETE,  
OPTIONS,  
TRACE;

写法：

@RequestMapping(value = "/testRequestMapping",method = {RequestMethod.*POST*})

### 绑定机制示例

#### 新建类ParamController

package cn.itcast.controller;  
  
import org.springframework.stereotype.Controller;  
import org.springframework.web.bind.annotation.RequestMapping;  
  
@Controller  
@RequestMapping("/param")  
public class ParamController {  
  
 */\*\*  
 \* 请求参数绑定  
 \** ***@return*** *\*/* @RequestMapping("/testParam")  
 public String testParam(String username,String password){  
 System.*out*.println("执行了。");  
 System.*out*.println("用户名："+username);  
 System.*out*.println("密码："+password);  
 return "success";  
 }  
  
}

#### 创建param.jsp文件

<%--  
 Created by IntelliJ IDEA.  
 User: jiyao  
 Date: 2020/4/1  
 Time: 20:16  
 To change this template use File | Settings | File Templates.  
--%>  
<%@ **page** contentType="text/html;charset=UTF-8" language="java" %>  
<html>  
<head>  
 <title>Title</title>  
</head>  
<body>  
<%-- 请求参数绑定--%>  
 <a href="param/testParam?username=hehe&password=123456">请求参数绑定</a>  
  
</body>  
</html>

### 封装javabean的参数绑定

#### 新建ParamController控制器

package cn.itcast.controller;  
  
 import cn.itcast.domain.Account;  
 import org.springframework.stereotype.Controller;  
 import org.springframework.web.bind.annotation.RequestMapping;  
  
@Controller  
@RequestMapping("/param")  
public class ParamController {  
  
 */\*\*  
 \* 请求参数绑定  
 \** ***@return*** *\*/* @RequestMapping("/testParam")  
 public String testParam(String username,String password){  
 System.*out*.println("执行了。");  
 System.*out*.println("用户名："+username);  
 System.*out*.println("密码："+password);  
 return "success";  
 }  
 */\*\*  
 \* 请求参数绑定把数据封装到javabean的类中  
 \** ***@return*** *\*/* @RequestMapping("/saveAccount")  
 public String saveAccount(Account account){  
 System.*out*.println("执行了。");  
 System.*out*.println(account);  
 return "success";  
 }  
  
}

#### 新建User类

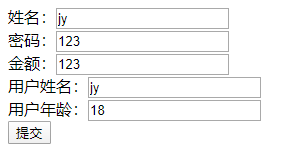
package cn.itcast.domain;  
  
import java.io.Serializable;  
  
public class User implements Serializable {  
 private String uname;  
 private Integer age;  
  
 public String getUname() {  
 return uname;  
 }  
  
 public void setUname(String uname) {  
 this.uname = uname;  
 }  
  
 public Integer getAge() {  
 return age;  
 }  
  
 public void setAge(Integer age) {  
 this.age = age;  
 }  
  
 @Override  
 public String toString() {  
 return "User{" +  
 "uname='" + uname + '\'' +  
 ", age=" + age +  
 '}';  
 }  
}

#### 新建Account类

package cn.itcast.domain;  
  
import javax.jws.soap.SOAPBinding;  
import java.io.Serializable;  
  
public class Account implements Serializable {  
 private String username;  
 private String password;  
 private Double money;  
  
 private User user;  
  
 public User getUser() {  
 return user;  
 }  
  
 public void setUser(User user) {  
 this.user = user;  
 }  
  
 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 Double getMoney() {  
 return money;  
 }  
  
 public void setMoney(Double money) {  
 this.money = money;  
 }  
  
 @Override  
 public String toString() {  
 return "Account{" +  
 "username='" + username + '\'' +  
 ", password='" + password + '\'' +  
 ", money=" + money +  
 ", user=" + user +  
 '}';  
 }  
}

#### 修改param.jsp

<%--  
 Created by IntelliJ IDEA.  
 User: jiyao  
 Date: 2020/4/1  
 Time: 20:16  
 To change this template use File | Settings | File Templates.  
--%>  
<%@ **page** contentType="text/html;charset=UTF-8" language="java" %>  
<html>  
<head>  
 <title>Title</title>  
</head>  
<body>  
<%-- 请求参数绑定--%>  
<%-- <a href="param/testParam?username=hehe&password=123456">请求参数绑定</a>--%>  
 <form action="param/saveAccount" method="post">  
 姓名：<input type="text" name="username"/><br/>  
 密码：<input type="text" name="password"/><br/>  
 金额：<input type="text" name="money"/><br/>  
 用户姓名：<input type="text" name="user.uname"/><br/>  
 用户年龄：<input type="text" name="user.age"/><br/>  
 <input type="submit" value="提交"/><br/>  
 </form>  
</body>  
</html>



### 解决中文乱码

在web.xml中Servlet标签上面加入

<!-- 配置解决中文乱码的过滤器-->  
<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>  
</filter>  
<filter-mapping>  
 <filter-name>characterEncodingFilter</filter-name>  
 <url-pattern>/\*</url-pattern>  
</filter-mapping>

### 封装集合

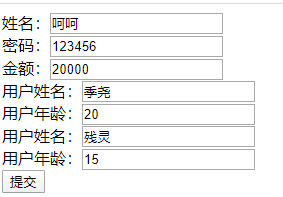
#### 在Account中添加List集合和Map集合并重构toString

private List<User> list;  
private Map<String,User> map;  
  
public List<User> getList() {  
 return list;  
}  
  
public void setList(List<User> list) {  
 this.list = list;  
}  
  
public Map<String, User> getMap() {  
 return map;  
}  
  
public void setMap(Map<String, User> map) {  
 this.map = map;  
}

#### 在jsp文件中加入

<form action="param/saveAccount" method="post">  
 姓名：<input type="text" name="username"/><br/>  
 密码：<input type="text" name="password"/><br/>  
 金额：<input type="text" name="money"/><br/>  
  
 用户姓名：<input type="text" name="list[0].uname"/><br/>  
 用户年龄：<input type="text" name="list[0].age"/><br/>  
  
 用户姓名：<input type="text" name="map['one'].uname"/><br/>  
 用户年龄：<input type="text" name="map['one'].age"/><br/>  
 <input type="submit" value="提交"/><br/>  
</form>





### 自定义类型转换器

问题如果数据类型转换出错如何解决

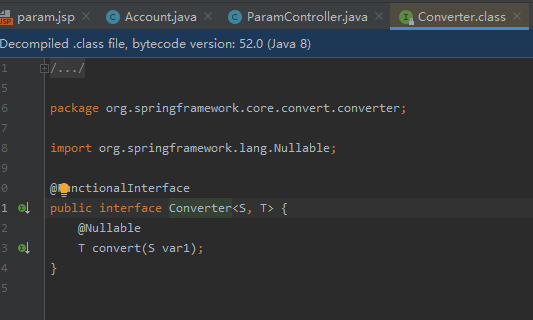
例如date类型数据2020/04/01

和2020-04-01（不被识别）

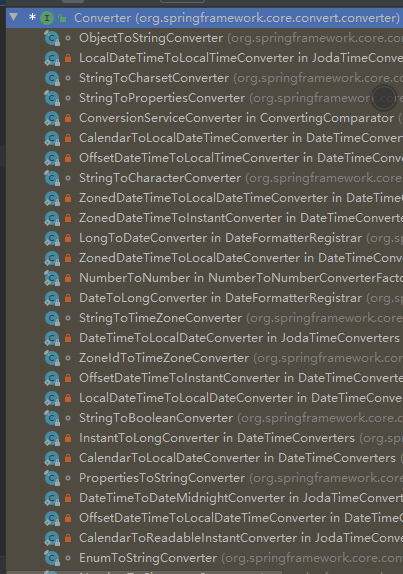
则需要自定义类型转换器

#### 使用步骤

找到源码



发现有很多类实现这个接口



##### 自建工具配置类StringToDateConverter继承Converter<String,Date>

package cn.itcast.utils;  
  
import com.sun.xml.internal.ws.policy.privateutil.RuntimePolicyUtilsException;  
import org.springframework.core.convert.converter.Converter;  
  
import java.text.DateFormat;  
import java.text.ParseException;  
import java.text.SimpleDateFormat;  
import java.util.Date;  
  
*/\*\*  
 \* 把字符串转换日期  
 \*/*public class StringToDateConverter implements Converter<String, Date> {  
 */\*\*  
 \*  
 \** ***@param*** *s 传入进来的字符串  
 \** ***@return*** *\*/* @Override  
 public Date convert(String s) {  
 //判断  
 if(s == null){  
 throw new RuntimeException("请您传入数据");  
 }  
 DateFormat df = new SimpleDateFormat("yyyy-MM-dd");  
 //把字符串转换日期  
 try {  
 return df.parse(s);  
 } catch (Exception e) {  
 throw new RuntimeException("数据类型转换出现错误");  
 }  
 }  
}

##### 在springmvc.xml中配置自定义类型转化器

<!-- 配置自定义类型转化器-->  
 <bean id="conversionService2" class="org.springframework.context.support.ConversionServiceFactoryBean">  
 <property name="converters">  
 <set>  
 <bean class="cn.itcast.utils.StringToDateConverter"/>  
 </set>  
 </property>  
 </bean>  
  
 <!--开启SpringMVC框架注解的支持-->  
 <mvc:annotation-driven conversion-service="conversionService2"/>

##### 配置控制器ParamController类

*/\*\*  
 \* 自定义类型转换器  
 \** ***@param*** *user  
 \** ***@return*** *\*/*@RequestMapping("/saveUser")  
public String saveUser(User user){  
 System.*out*.println("执行了。");  
 System.*out*.println(user);  
 return "success";  
}

##### jsp表单提交

<form action="param/saveUser" method="post">  
 用户姓名：<input type="text" name="uname"/><br/>  
 用户年龄：<input type="text" name="age"/><br/>  
 用户生日：<input type="text" name="date"/><br/>  
 <input type="submit" value="提交"/><br/>  
</form>

### 原生ServletAPI

*/\*\*  
 \* 原生的API获取  
 \** ***@return*** *\*/*@RequestMapping("/testServlet")  
public String testServlet(HttpServletRequest request, HttpServletResponse response){  
 System.*out*.println("执行了。");  
 System.*out*.println(request);  
 HttpSession session =request.getSession();  
 System.*out*.println(session);  
 ServletContext servletContext = session.getServletContext();  
 System.*out*.println(servletContext);  
 System.*out*.println(response);  
 return "success";  
}

## 常用注解

### RequestParam

把请求中指定名称的参数给控制器中形参赋值

value：请求参数中的名称

required：请求参数中是否必须提供此参数。默认值：true。表示必须提供，如果不提供将报错

#### 示例

##### jsp

<a href="anno/testRequestParam?name=哈哈">RequestParam</a>

##### AnnoController类

package cn.itcast.controller;  
import org.springframework.stereotype.Controller;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RequestParam;  
*/\*\*  
 \* 常用的注解  
 \*/*@Controller  
@RequestMapping("/anno")  
public class AnnoController {  
 @RequestMapping("/testRequestParam")  
 public String testRequestParam(@RequestParam(name = "name") String username){  
 System.*out*.println("执行了");  
 System.*out*.println(username);  
 return "success";  
 }  
}

### RequestBody

用于获取请求体内容。直接使用得到是key=value&key=value...结构的数据

get请求方法不适用

属性：

required:是否必须有请求体。默认值是true

当取值为true时get方法会报错

get请求得到是null

#### 示例

*/\*\*  
 \* 获取到请求体的内容  
 \** ***@param*** *body  
 \** ***@return*** *\*/*@RequestMapping("/testRequestBody")  
public String testRequestBody(@RequestBody String body){  
 System.*out*.println("执行了");  
 System.*out*.println(body);  
 return "success";  
}

##### jsp

<form action="anno/testRequestBody" method="post">  
 用户姓名：<input type="text" name="username"/><br/>  
 用户年龄：<input type="text" name="age"/><br/>  
 <input type="submit" value="提交"/><br/>  
</form>

### PathVariable

用于绑定url的占位符，例如：请求url中/delete/{id},这个{id}就是占位符。

url支持占位符是spring3.0之后加入的。是springmvc支持rest风格url的一个重要标志

属性：

value:用于指定url中占位符名称。

required:是否必须提供占位符。

#### 示例：

jsp

<a href="anno/testPathVariable/10">testPathVariable</a>

Controller方法

*/\*\*  
 \* PathVariable注解  
 \** ***@param*** *id  
 \** ***@return*** *\*/*@RequestMapping(value = "/testPathVariable/{sid}",method = RequestMethod.*PUT*)  
public String testPathVariable(@PathVariable(name = "sid") String id){  
 System.*out*.println("执行了");  
 System.*out*.println(id);  
 return "success";  
}

### RequestHeader

用于获取请求消息头

属性：

value：提供消息头名称

required：是否必须有此消息头

实际开发中一般不怎么用

示例：

jsp

<a href="anno/testRequestHeader">testRequestHeader</a>

Controller方法

*/\*\*  
 \* 获取请求头的值  
 \** ***@param*** *header  
 \** ***@return*** *\*/*@RequestMapping("/testRequestHeader")  
public String testRequestHeader(@RequestHeader(value = "Accept") String header){  
 System.*out*.println("执行了");  
 System.*out*.println(header);  
 return "success";  
}

### CookieValue

用于把指定cookie名称的值传入控制器方法参数

属性：

value:指定cookie的名称

required:是否必须有此cookie。

#### jsp

<a href="anno/testCookieValue">testCookieValue</a>

#### Controller方法

*/\*\*  
 \* 获取Cookie的值  
 \** ***@param*** *cookieValue  
 \** ***@return*** *\*/*@RequestMapping("/testCookieValue")  
public String testCookieValue(@CookieValue(value = "JSESSIONID") String cookieValue){  
 System.*out*.println("执行了");  
 System.*out*.println(cookieValue);  
 return "success";  
}

### ModelAttribute

该注解是SpringMVC4.3版本以后新加入的。它可以用于修饰方法和参数

出现在方法上，表示当前方法会在控制器的方法执行之前，先执行，它可以修饰没有返回值的方法，也可以修饰有具体返回值的方法。

出现在参数上，获取指定的数据给参数赋值。

属性

Value：用于获取数据的key。key可以是POJO的属性名称，也可以是map结构的key

应用：

当表单提交数据不是完整的实体类数据时，保证没有提交数据的字段使用数据库对象原来的数据。

例如：

我们在编辑一个用户时，用户有一个创建信息字段，该字段的值是不允许被修改的。在提交表单数据是肯定没有此字段的内容，一旦更新就会把该字段内容置为null，此时就可以使用此注解解决问题。

#### 有返回值

*/\*\*  
 \* ModelAttribute  
 \** ***@param*** *user  
 \** ***@return*** *\*/*@RequestMapping("/testModelAttribute")  
public String testModelAttribute(User user){  
 System.*out*.println("testModelAttribute执行了");  
 System.*out*.println(user);  
 return "success";  
}

*/\*\*  
 \*有返回值  
 \** ***@param*** *uname  
 \** ***@return*** *\*/*@ModelAttribute  
public User showUser(String uname){  
 System.*out*.println("showUser执行了...");  
 //通过用户查询数据库(模拟)  
 User user = new User();  
 user.setUname(uname);  
 user.setAge(20);  
 user.setDate(new Date());  
 return user;  
}

#### 无返回值传入参数Map

*/\*\*  
 \* ModelAttribute  
 \** ***@param*** *user  
 \** ***@return*** *\*/*@RequestMapping("/testModelAttribute")  
public String testModelAttribute(@ModelAttribute("abc") User user){  
 System.*out*.println("testModelAttribute执行了");  
 System.*out*.println(user);  
 return "success";  
}  
  
*/\*\*  
 \* 无返回值传入Map  
 \** ***@param*** *uname  
 \** ***@param*** *map  
 \*/*@ModelAttribute  
public void showUser(String uname, Map<String,User> map){  
 System.*out*.println("showUser执行了...");  
 //通过用户查询数据库(模拟)  
 User user = new User();  
 user.setUname(uname);  
 user.setAge(20);  
 user.setDate(new Date());  
 map.put("abc", user);  
}

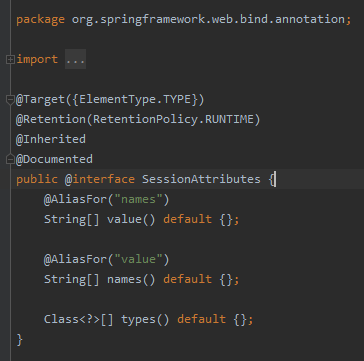
### SessionAttribute

用于多次执行控制器方法间参数共享

value:用于指定存入的属性名称

type:用于指定存入的数据类型

#### 源码



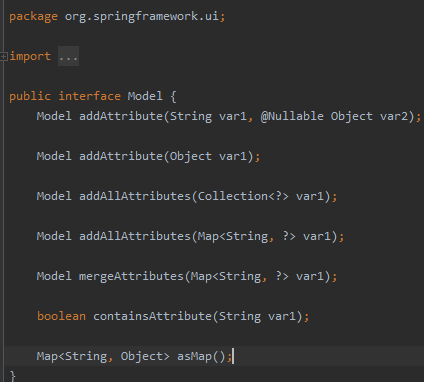
作用在类中

#### Controller类方法

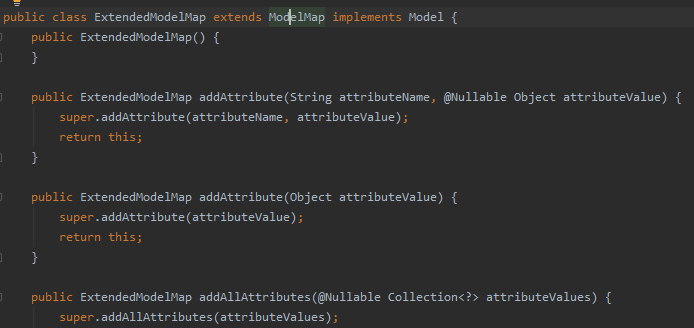
@SessionAttributes(value = {"msg"})  
public class AnnoController {

*/\*\*  
 \* SessionAttributes的注解  
 \** ***@param*** *model  
 \** ***@return*** *\*/*@RequestMapping("/testSessionAttributes")  
public String testSessionAttributes(Model model){  
 System.*out*.println("testSessionAttributes执行了");  
 //底层会存储到request域对象当中  
 model.addAttribute("msg", "美美");  
 return "success";  
}

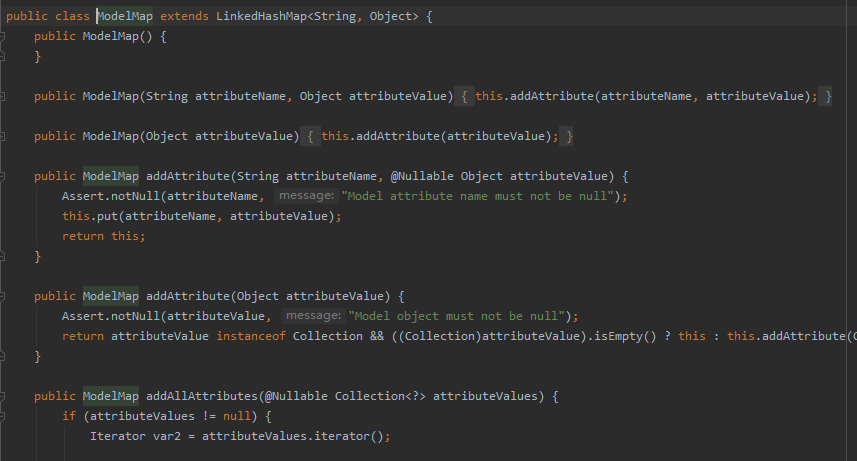
#### 查看Model类中的用法



ExtendedModelMap为实现Model接口的类



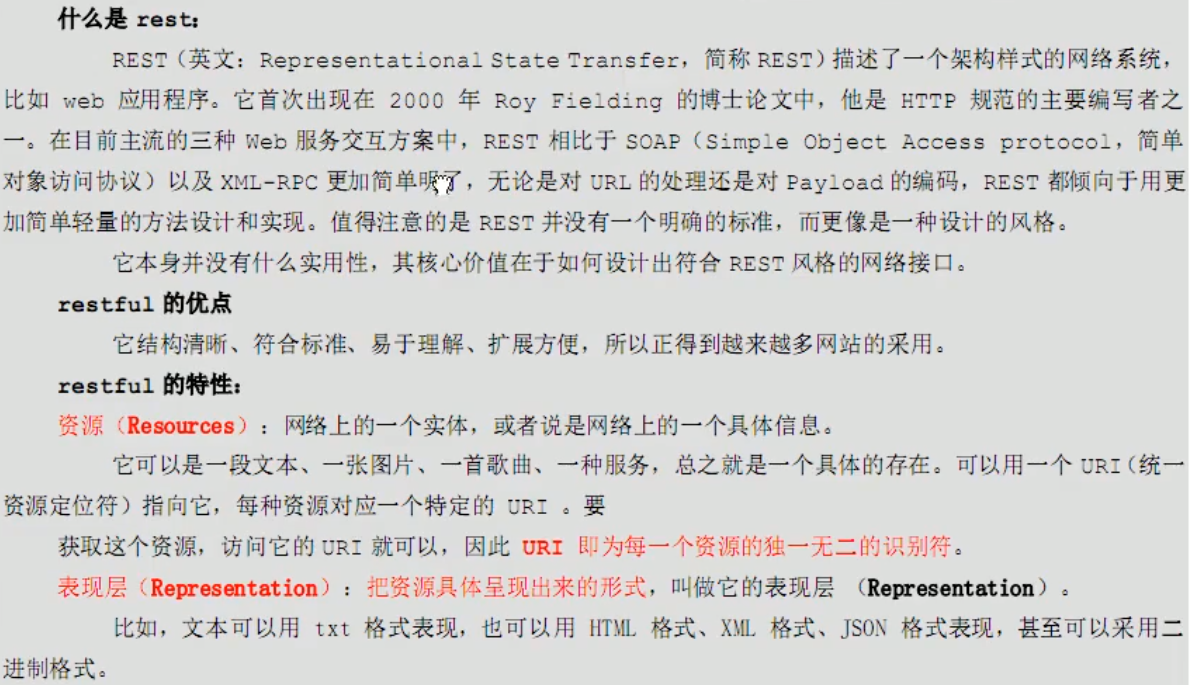
找到其继承的Modelmap类

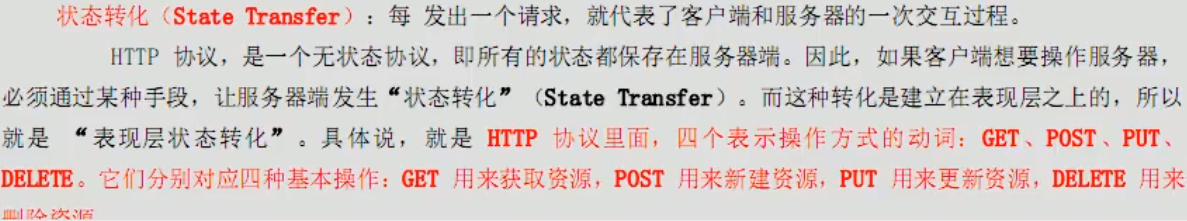


#### 获取session域中值与清除域中值方法

*/\*\*  
 \* 获取session域中值  
 \** ***@param*** *modelMap  
 \** ***@return*** *\*/*@RequestMapping("/getSessionAttributes")  
public String getSessionAttributes(ModelMap modelMap){  
 System.*out*.println("getSessionAttributes执行了");  
 //底层会存储到request域对象当中  
 String msg = (String) modelMap.get("msg");  
 System.*out*.println(msg);  
 return "success";  
}  
  
*/\*\*  
 \* 清除  
 \** ***@param*** *status  
 \** ***@return*** *\*/*@RequestMapping("/deleteSessionAttributes")  
public String deleteSessionAttributes(SessionStatus status){  
 System.*out*.println("deleteSessionAttributes执行了");  
 status.setComplete();  
 return "success";  
}

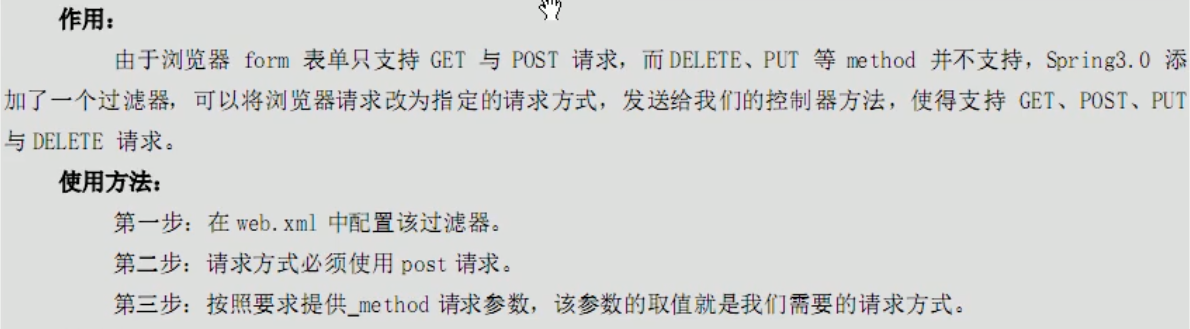
### REST风格URL







### 基于HiddentHttpMethodFilter的示例



## 响应数据和结果视图

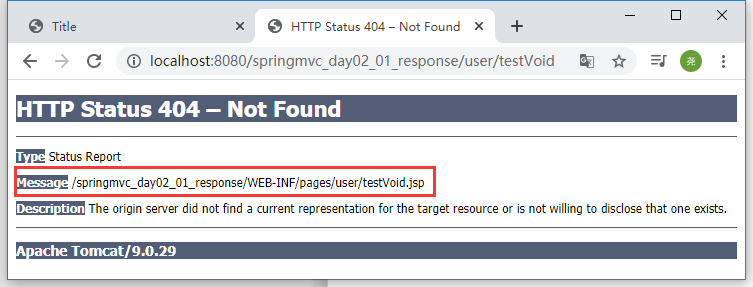
### 返回值分类

#### 字符串

*/\*\*  
 \* 返回字符串  
 \** ***@param*** *model  
 \** ***@return*** *\*/*@RequestMapping(value = "/testString")  
public String testString(Model model){  
 System.*out*.println("testString方法执行了");  
 //模拟从数据库中查询出User对象  
 User user = new User();  
 user.setUsername("jy");  
 user.setPassword("123");  
 user.setAge(20);  
 //model对象  
 model.addAttribute("user", user);  
 return "success";  
}

#### void

默认情况



*/\*\*  
 \* void  
 \* 请求转发一次请求，请求路径不用编写项目名称  
 \*/*@RequestMapping(value = "/testVoid")  
public void testVoid(HttpServletRequest request, HttpServletResponse response) throws Exception{  
 System.*out*.println("testVoid方法执行了");  
 //编写请求转发的程序  
 //request.getRequestDispatcher("/WEB-INF/pages/success.jsp").forward(request,response);  
  
 //重定向  
 //response.sendRedirect(request.getContextPath()+"/index.jsp");  
  
 //设置中文乱码  
 response.setCharacterEncoding("UTF-8");  
 response.setContentType("text/html;charset=UTF-8");  
  
 //直接响应  
 response.getWriter().print("你好");  
  
 return;  
}

##### 三种方式

//编写请求转发的程序  
//request.getRequestDispatcher("/WEB-INF/pages/success.jsp").forward(request,response);

//重定向  
//response.sendRedirect(request.getContextPath()+"/index.jsp");

//设置中文乱码  
response.setCharacterEncoding("UTF-8");  
response.setContentType("text/html;charset=UTF-8");  
  
//直接响应  
response.getWriter().print("你好");

#### ModelAndView

*/\*\*  
 \* ModelAndView  
 \** ***@return*** *\*/*@RequestMapping(value = "/testModelAndView")  
public ModelAndView testModelAndView(){  
 //创建一个ModelAndView对象  
 ModelAndView mv = new ModelAndView();  
 System.*out*.println("testModelAndView方法执行了");  
 //模拟从数据库中查询出User对象  
 User user = new User();  
 user.setUsername("cl");  
 user.setPassword("007");  
 user.setAge(20);  
 //把User对象存储到mv对象中，并且会吧user对象存入到request对象  
 mv.addObject("user",user);  
 //跳转到哪个页面  
 mv.setViewName("success");  
 return mv;  
}

### 转发和重定向（用得少）

*/\*\*  
 \* 使用关键字的方式进行转发或者重定向  
 \** ***@param*** *\** ***@return*** *\*/*@RequestMapping(value = "/testForwardOrRedirect")  
public String testForwardOrRedirect(){  
 System.*out*.println("testForwardOrRedirect方法执行了");  
 //请求转发  
 //return "forward:/WEB-INF/pages/success.jsp";  
  
 //重定向  
 return "redirect:/index.jsp";  
}

### ResponseBody响应json数据





#### 添加pom.xml依赖坐标

<dependency>  
 <groupId>com.fasterxml.jackson.core</groupId>  
 <artifactId>jackson-databind</artifactId>  
 <version>2.9.10</version>  
</dependency>  
  
<dependency>  
 <groupId>com.fasterxml.jackson.core</groupId>  
 <artifactId>jackson-core</artifactId>  
 <version>2.9.10</version>  
</dependency>  
  
<dependency>  
 <groupId>com.fasterxml.jackson.core</groupId>  
 <artifactId>jackson-annotations</artifactId>  
 <version>2.9.10</version>  
</dependency>

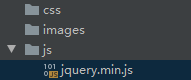
#### 使用Ajax需要下载jquery.min.js

下载地址

<https://code.jquery.com/jquery-3.4.1.min.js>

中文开发文档

<http://jquery.cuishifeng.cn/>



#### 配置拦截器不拦截静态资源

<!--前端控制器，哪些静态资源不拦截-->  
<mvc:resources location="/css/" mapping="/css/\*\*" />  
<mvc:resources location="/images/" mapping="/images/\*\*"/>  
<mvc:resources location="/js/" mapping="/js/\*\*" />

#### jsp页面

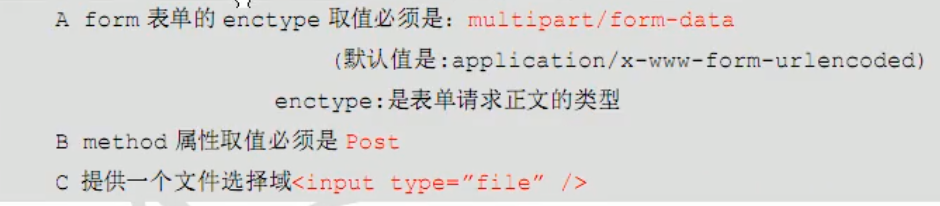
<script src="js/jquery.min.js"></script>  
<script>  
 //页面加载，绑定点击事件  
 $(function () {  
 $("#btn").click(function () {  
 $.ajax({  
 //编写json格式 设置属性和值  
 url:"user/testAjax",  
 contentType:"application/json;charset=UTF-8",  
 data:'{"username":"hehe","password":"123","age":30}',  
 dataType:"json",  
 type:"post",  
 success:function (data) {  
 //data服务器端响应的json的数据，进行解析  
 alert(data);  
 alert(data.username);  
 alert(data.age);  
 alert(data.password);  
 }  
 });  
 });  
 });  
</script>

<button id="btn">发送ajax的请求</button>

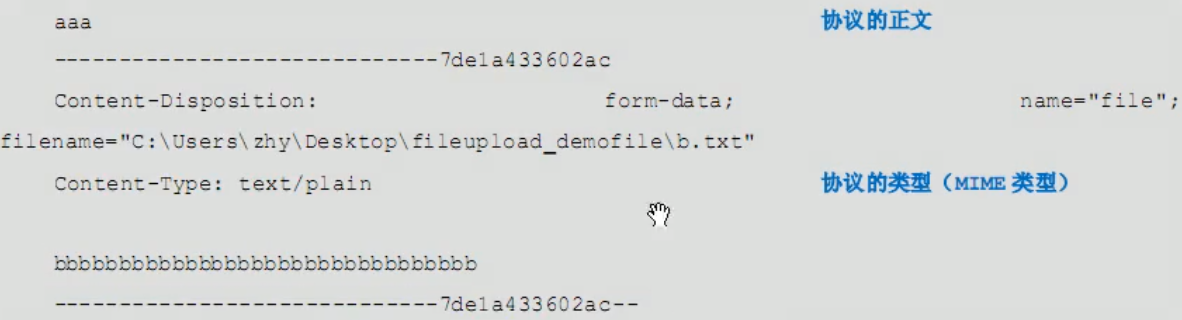
Controller类方法

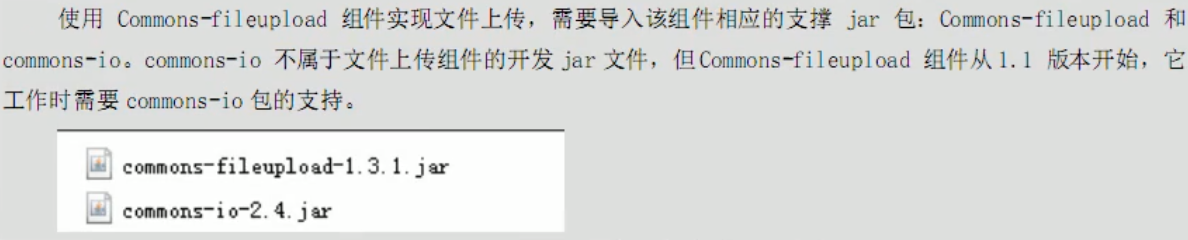
*/\*\*  
 \* 模拟异步请求响应  
 \** ***@return*** *\*/*@RequestMapping(value = "/testAjax")  
public @ResponseBody User testAjax(@RequestBody User user){  
 System.*out*.println("testAjax方法执行了。。。");  
 //客户端发送ajax的请求，传的是json字符串，后端把json字符串封装到user对象中  
 System.*out*.println(user);  
 //做响应,模拟查询数据库  
 user.setUsername("haha");  
 user.setAge(40);  
 //做响应  
 return user;  
}

## SpringMVC实现文件上传









### 传统上传方式

index.jsp

<h3>传统文件上传</h3>  
<form action="/user/fileUpLoad1" method="post" enctype="multipart/form-data">  
 选择文件:<input type="file" name="upload"/><br>  
 <input type="submit" value="上传">  
</form>

pom.xml导入坐标

<dependency>  
 <groupId>commons-fileupload</groupId>  
 <artifactId>commons-fileupload</artifactId>  
 <version>1.4</version>  
</dependency>  
  
<dependency>  
 <groupId>commons-io</groupId>  
 <artifactId>commons-io</artifactId>  
 <version>2.6</version>  
</dependency>

UserController.java

*/\*\*  
 \*传统文件上传  
 \** ***@return*** *\*/*@RequestMapping(value = "/fileUpLoad1")  
public String fileUpLoad1(HttpServletRequest request) throws Exception {  
 System.*out*.println("文件上传..");  
 //使用fileupload组件完成文件上传  
 //上传的位置  
 String path = request.getSession().getServletContext().getRealPath("/uploads/");  
 //判断，该路径是否存在  
 File file = new File(path);  
 if (!file.exists()){  
 //创建该文件夹  
 file.mkdirs();  
 }  
 //解析request对象，获取上传文件项  
 DiskFileItemFactory factory = new DiskFileItemFactory();  
 ServletFileUpload upload = new ServletFileUpload(factory);  
 //解析request  
 List<FileItem> items = upload.parseRequest(request);  
 for (FileItem item : items) {  
 //进行判断，当前item对象是否是上传文件项  
 if (item.isFormField()){  
 //说明是普通表单项  
 }else {  
 //说明是上传文件项  
 //获取上传文件的名称  
 String filename = item.getName();  
 //把文件的名称设置唯一值，uuid  
 String uuid = UUID.*randomUUID*().toString().replace("-", "");  
 filename = uuid + "\_" + filename;  
 //完成文件上传  
 item.write(new File(path,filename));  
 //删除临时文件  
 item.delete();  
 }  
  
 }

### SpringMVC文件上传

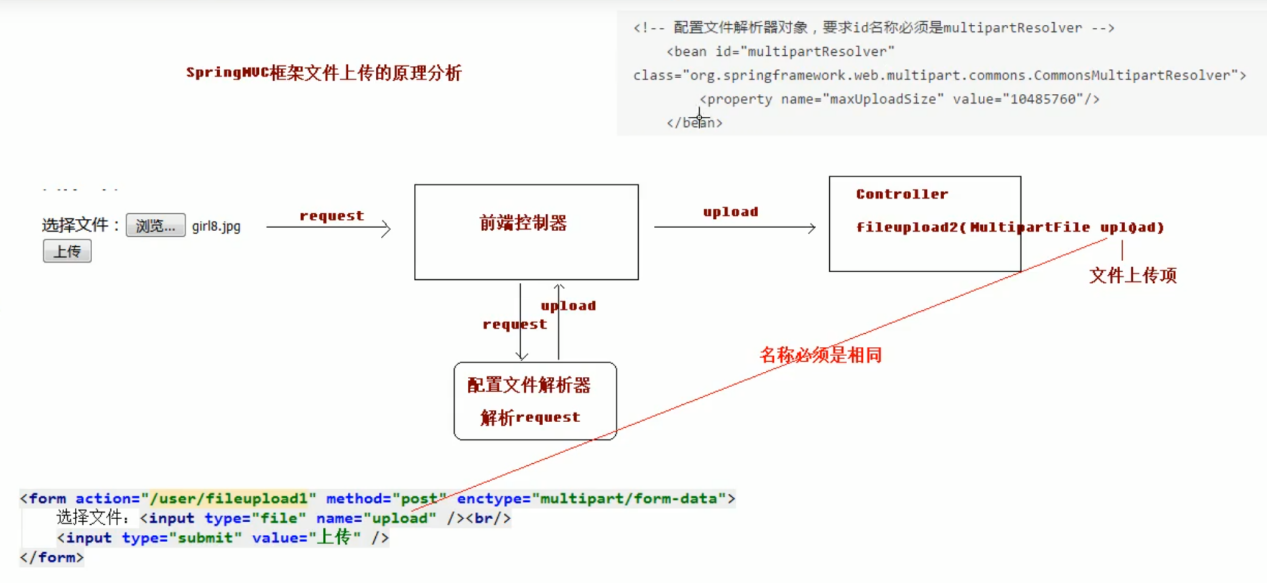
index.jsp

<h3>SpringMVC文件上传</h3>  
<form action="/user/fileUpLoad2" method="post" enctype="multipart/form-data">  
 选择文件:<input type="file" name="upload"/><br>  
 <input type="submit" value="上传">  
</form>

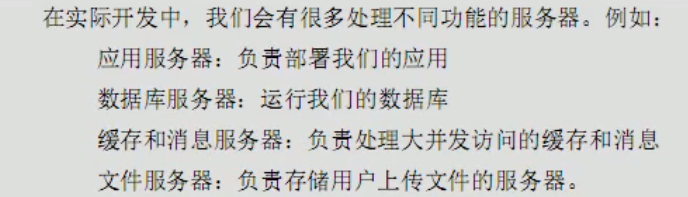
UserController.java

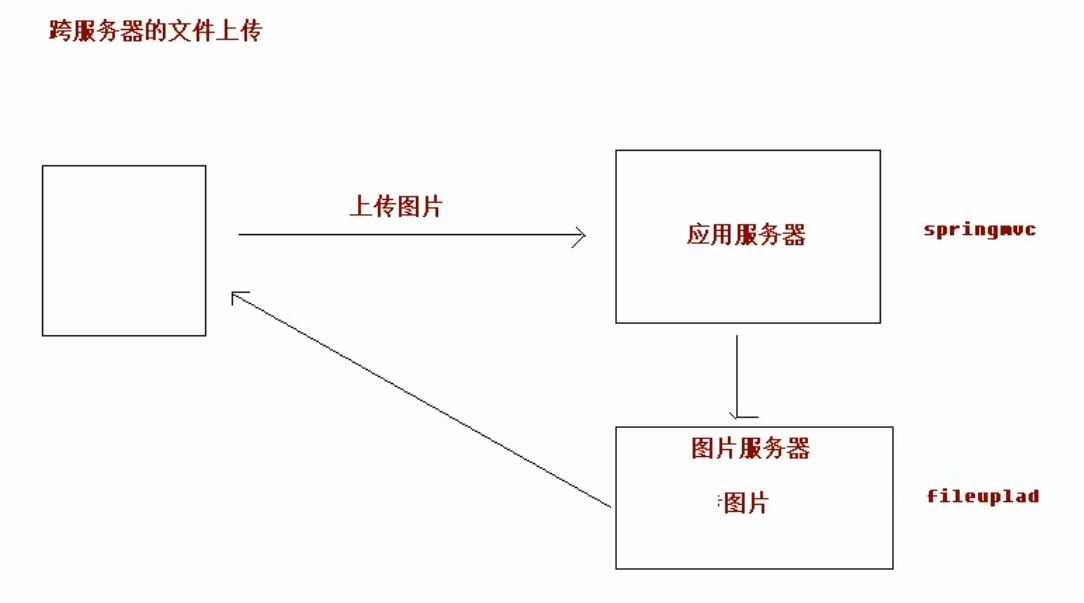
*/\*\*  
 \*SpringMVC文件上传  
 \** ***@return*** *\*/*@RequestMapping(value = "/fileUpLoad2")  
public String fileUpLoad2(HttpServletRequest request, MultipartFile upload) throws Exception {  
 System.*out*.println("springmvc文件2上传..");  
 //使用fileupload组件完成文件上传  
 //上传的位置  
 String path = request.getSession().getServletContext().getRealPath("/uploads/");  
 //判断，该路径是否存在  
 File file = new File(path);  
 if (!file.exists()){  
 //创建该文件夹  
 file.mkdirs();  
 }  
 //说明是上传文件项  
 //获取上传文件的名称  
 String filename = upload.getOriginalFilename();  
 //把文件的名称设置唯一值，uuid  
 String uuid = UUID.*randomUUID*().toString().replace("-", "");  
 filename = uuid + "\_" + filename;  
 //完成文件上传  
 upload.transferTo(new File(path,filename));  
 return "success";  
}

#### 上传原理



### SpringMVC跨服务器方式上传

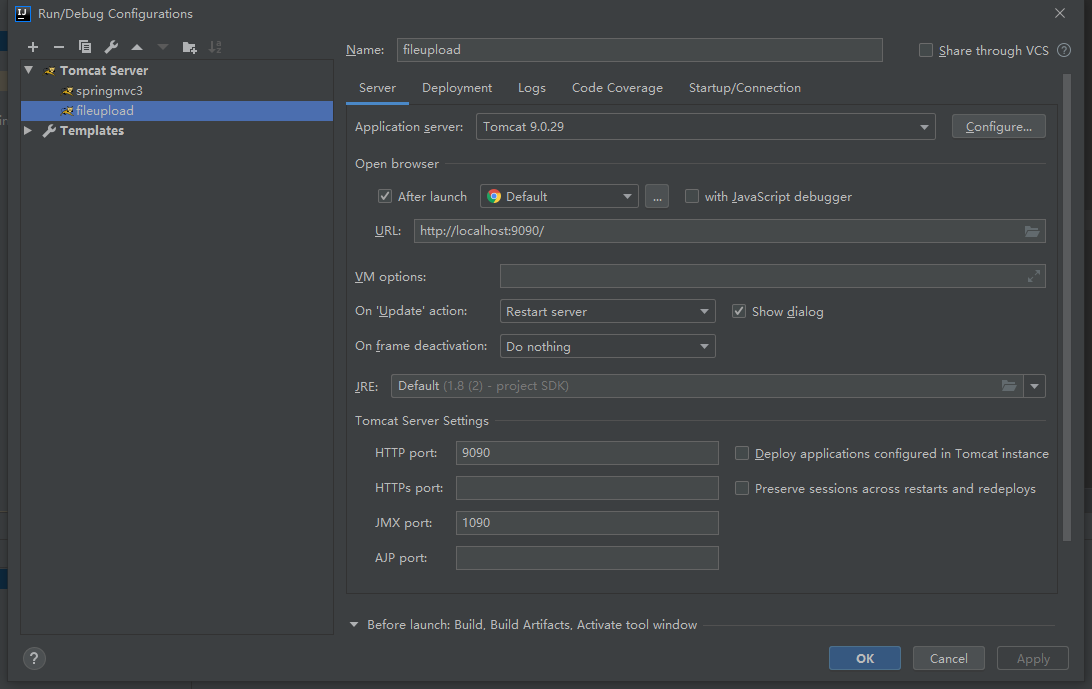




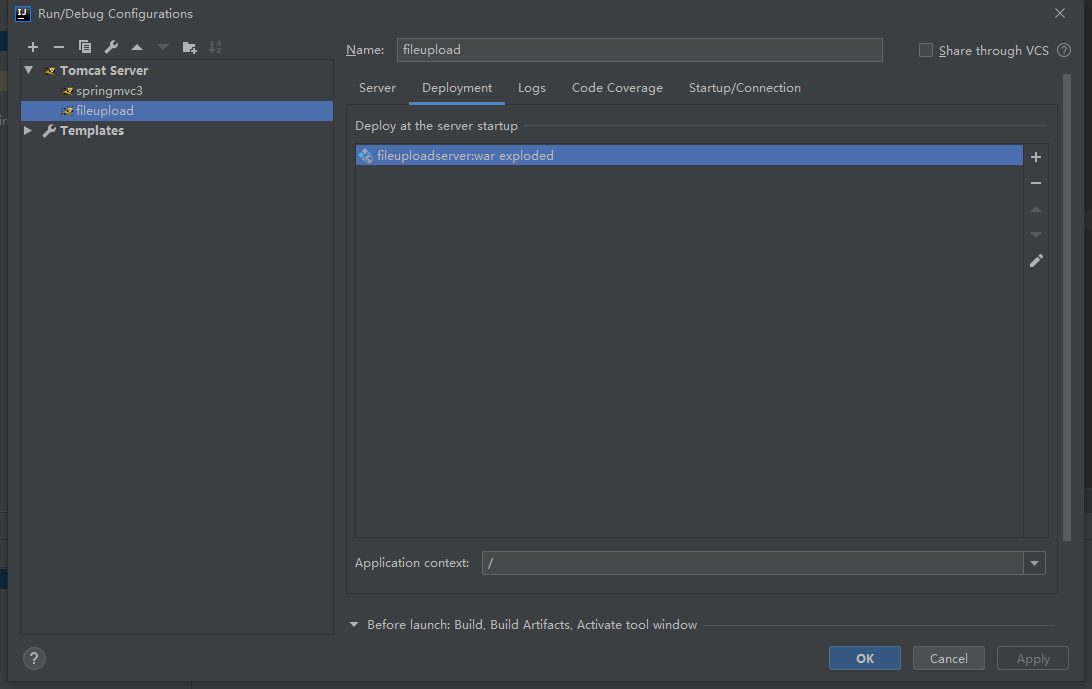
#### 建立图片服务器

新建maven WebApp项目

配置tomcat服务器端口号，不与springmvc相同，且未被占用的端口号



添加artifact



##### 配置web.xml去掉Tomcat只读属性

<servlet>  
 <servlet-name>default</servlet-name>  
 <servlet-class>org.apache.catalina.servlets.DefaultServlet</servlet-class>  
 <init-param>  
 <param-name>debug</param-name>  
 <param-value>0</param-value>  
 </init-param>  
 <init-param>  
 <param-name>readonly</param-name>  
 <param-value>false</param-value>  
 </init-param>  
 <init-param>  
 <param-name>listings</param-name>  
 <param-value>false</param-value>  
 </init-param>  
 <load-on-startup>1</load-on-startup>  
</servlet>

##### 启动服务器

#### 建立springmvc项目服务器

index.jsp

<h3>SpringMVC跨服务器方式文件上传</h3>  
<form action="/user/fileUpLoad3" method="post" enctype="multipart/form-data">  
 选择文件:<input type="file" name="upload"/><br>  
 <input type="submit" value="上传">  
</form>

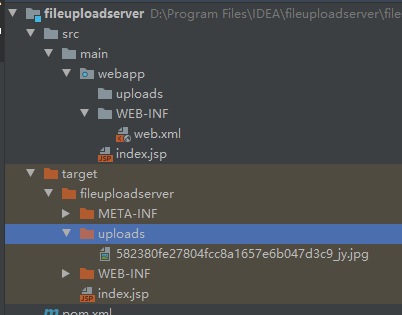
springmvc.xml配置文件解析最大值

<!-- 配置文件解析器对象 -->  
<bean id="multipartResolver" class="org.springframework.web.multipart.commons.CommonsMultipartResolver">  
 <property name="maxUploadSize" value="10485760"/>  
</bean>

UserController.java

*/\*\*  
 \*SpringMVC跨服务器文件文件上传  
 \** ***@return*** *\*/*@RequestMapping(value = "/fileUpLoad3")  
public String fileUpLoad3(MultipartFile upload) throws Exception {  
 System.*out*.println("springmvc跨服务器文件3上传..");  
 //定义上传文件服务器路径  
 String path = "http://localhost:9090/uploads/";  
 //说明是上传文件项  
 //获取上传文件的名称  
 String filename = upload.getOriginalFilename();  
 //把文件的名称设置唯一值，uuid  
 String uuid = UUID.*randomUUID*().toString().replace("-", "");  
 filename = uuid + "\_" + filename;  
 //完成文件上传,跨服务器上传  
 //创建客户端的对象  
 Client client = Client.*create*();  
 //和图片服务器进行连接  
 WebResource webResource = client.resource(path + filename);  
 //上传图片文件  
 webResource .put(upload.getBytes());  
 return "success";  
}

#### 最后在target中上传目录文件中寻找图片

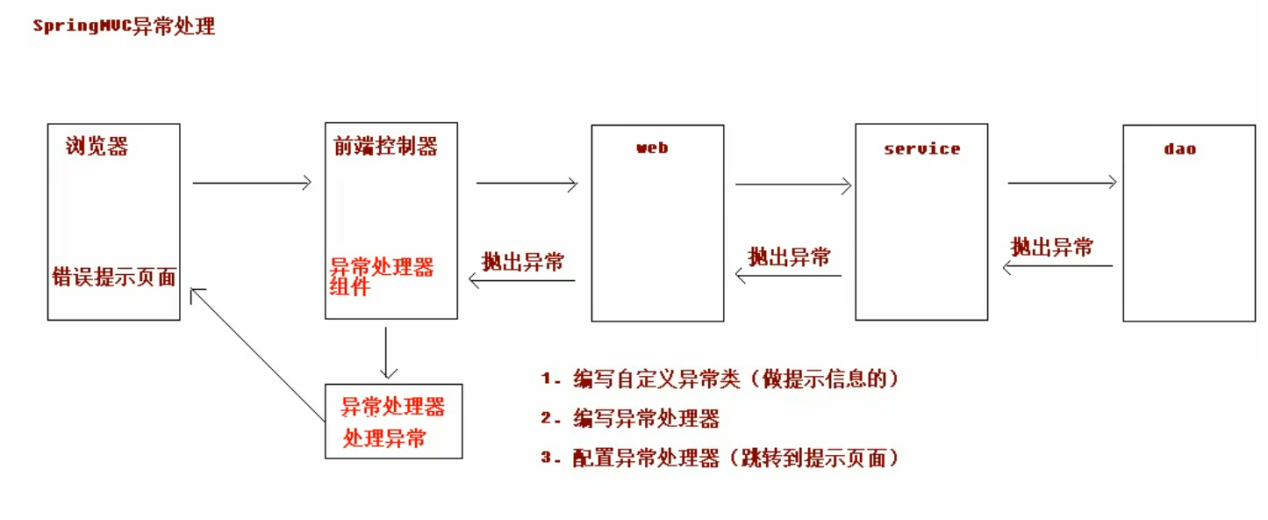


## SpringMVC的异常处理

### 异常处理思路

Controller调用service，service调用dao，异常都是向上抛出的，最终有DispatcherServlet找异常处理器进行异常的处理。

### SpringMVC的异常处理



#### 创造异常，模拟异常

@RequestMapping("/testException")  
public String testException() throws SysException{  
 System.*out*.println("testException执行了");  
  
 try {  
 //模拟异常  
 int a = 10/0;  
 } catch (Exception e) {  
 //打印异常信息  
 e.printStackTrace();  
 throw new SysException("查询所有用户出现错误了...");  
 }  
 return "success";  
}

#### 自定义异常类

package cn.itcast.exception;  
  
*/\*\*  
 \* 自定义异常类  
 \*/*public class SysException extends Exception{  
  
 //存储提示信息的  
 private String message;  
  
 @Override  
 public String getMessage() {  
 return message;  
 }  
  
 public void setMessage(String message) {  
 this.message = message;  
 }  
  
 public SysException(String message) {  
 this.message = message;  
 }  
}

#### 配置异常处理器

package cn.itcast.exception;  
  
import org.springframework.web.servlet.HandlerExceptionResolver;  
import org.springframework.web.servlet.ModelAndView;  
  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
  
*/\*\*  
 \* 异常处理器  
 \*/*public class SysExceptionResolver implements HandlerExceptionResolver {  
  
 */\*\*  
 \*  
 \** ***@param*** *httpServletRequest  
 \** ***@param*** *httpServletResponse  
 \** ***@param*** *o  
 \** ***@param*** *e  
 \** ***@return*** *\*/* @Override  
 public ModelAndView resolveException(HttpServletRequest httpServletRequest, HttpServletResponse httpServletResponse, Object o, Exception e) {  
 //获取异常对象  
 SysException ex = null;  
 if (e instanceof SysException){  
 ex = (SysException) e;  
 }else {  
 ex = new SysException("系统正在维护....");  
 }  
 //创建ModelAndView对象  
 ModelAndView mv = new ModelAndView();  
 mv.addObject("errorMsg",ex.getMessage());  
 mv.setViewName("error");  
 return mv;  
 }  
}

#### springmvc.xml

<!--配置异常处理器-->  
<bean id="sysExceptionResolver" class="cn.itcast.exception.SysExceptionResolver"/>

#### error.jsp跳转页

<%@ **page** contentType="text/html;charset=UTF-8" language="java" isELIgnored="false" %>  
<html>  
<head>  
 <title>Title</title>  
</head>  
<body>  
 ${errorMsg}  
</body>  
</html>

## SpirngMVC中的拦截器

### springmvc.xml文件配置拦截器

<!-- 配置拦截器-->  
<mvc:interceptors>  
 <!--配置拦截器-->  
 <mvc:interceptor>  
 <!--要拦截的具体的方法-->  
 <mvc:mapping path="/user/\*"/>  
 <!--不要拦截的方法  
 <mvc:exclude-mapping path=""/>-->  
 <!--配置拦截器对象-->  
 <bean class="cn.itcast.interceptor.MyInterceptor1"/>  
 </mvc:interceptor>  
 <!--配置第二个拦截器-->  
 <mvc:interceptor>  
 <!--要拦截的具体的方法-->  
 <mvc:mapping path="/\*\*"/>  
 <!--不要拦截的方法  
 <mvc:exclude-mapping path=""/>-->  
 <!--配置拦截器对象-->  
 <bean class="cn.itcast.interceptor.MyInterceptor2"/>  
 </mvc:interceptor>  
</mvc:interceptors>

### 自定义拦截器类，实现HandlerInterceptor接口

package cn.itcast.interceptor;  
  
import org.springframework.web.servlet.HandlerInterceptor;  
import org.springframework.web.servlet.ModelAndView;  
  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
  
*/\*\*  
 \* 自定义拦截器  
 \*/*public class MyInterceptor1 implements HandlerInterceptor {  
  
 */\*\*  
 \* 预处理，controller方法执行前  
 \* return true 放行,执行下一个拦截器，如果没有，执行Controller中的方法  
 \* return false不放行  
 \** ***@param*** *request  
 \** ***@param*** *response  
 \** ***@param*** *handler  
 \** ***@return*** *\** ***@throws*** *Exception  
 \*/* @Override  
 public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) throws Exception {  
 System.*out*.println("MyInterceptor1执行了...前1111");  
// request.getRequestDispatcher("/WEB-INF/pages/error.jsp").forward(request, response);  
 return true;  
 }  
  
 */\*\*  
 \* 后处理方法，controller方法执行后，success.jsp执行之前  
 \** ***@param*** *request  
 \** ***@param*** *response  
 \** ***@param*** *handler  
 \** ***@param*** *modelAndView  
 \** ***@throws*** *Exception  
 \*/* @Override  
 public void postHandle(HttpServletRequest request, HttpServletResponse response, Object handler, ModelAndView modelAndView) throws Exception {  
 System.*out*.println("MyInterceptor1执行了...后1111");  
// request.getRequestDispatcher("/WEB-INF/pages/error.jsp").forward(request, response);  
 }  
  
 */\*\*  
 \* success.jsp页面执行后，该方法会执行  
 \** ***@param*** *request  
 \** ***@param*** *response  
 \** ***@param*** *handler  
 \** ***@param*** *ex  
 \** ***@throws*** *Exception  
 \*/* @Override  
 public void afterCompletion(HttpServletRequest request, HttpServletResponse response, Object handler, Exception ex) throws Exception {  
 System.*out*.println("MyInterceptor1执行了...最后1111");  
 }  
}