

GBA培训项目课程材料-Spring MVC

国际商业机器（中国）有限公司

2020年8月

目 录

- **Spring MVC框架介绍**
- Spring MVC快速入门
- SpringMVC基于Maven构建
- SpringMVC基于注解开发
- SpringMVC获取请求中的参数
- SpringMVC中响应json
- SpringMVC实现文件上传
- SpringMVC拦截器
- SpringMVC异常处理

Spring MVC框架介绍



Spring MVC是Spring提供的一个强大而灵活的web框架。SpringMVC框架是以**请求**为驱动，围绕**Servlet**设计，将请求发给控制器，然后通过模型对象，分派器来展示请求结果视图。其中核心类是DispatcherServlet，它是一个Servlet，顶层是实现的Servlet接口。

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。**Spring MVC的两个核心**是：

处理器映射：选择使用哪个控制器来处理请求

视图解析器：选择结果应该如何渲染

为什么要使用SpringMVC?

很多应用程序的问题在于处理业务数据的对象和显示业务数据的视图之间存在紧密耦合，通常，更新业务对象的命令都是从视图本身发起的，使视图对任何业务对象更改都有高度敏感性。而且，当多个视图依赖于同一个业务对象时是没有灵活性的。

SpringMVC是一种基于Java，实现了Web MVC设计模式，请求驱动类型的轻量级Web框架，即使用了MVC架构模式的思想，将Web层进行**职责解耦**。基于请求驱动指的就是使用请求-响应模型，框架的目的就是帮助我们**简化开发**，SpringMVC也是要简化我们日常Web开发。

SpringMVC接口解释

(1) **DispatcherServlet**接口： Spring提供的前端控制器，所有的请求都有经过它来统一分发。在DispatcherServlet将请求分发给Spring Controller之前，需要借助于Spring提供的HandlerMapping定位到具体的Controller。

(2) **HandlerMapping**接口： 能够完成客户请求到Controller映射。

(3) **Controller**接口： 需要为并发用户处理请求，因此实现Controller接口时，必须保证线程安全并且可重用。 Controller将处理用户请求，一旦Controller处理完用户请求，则返回 ModelAndView对象给DispatcherServlet前端控制器， ModelAndView中包含了模型 (Model) 和视图 (View) 。

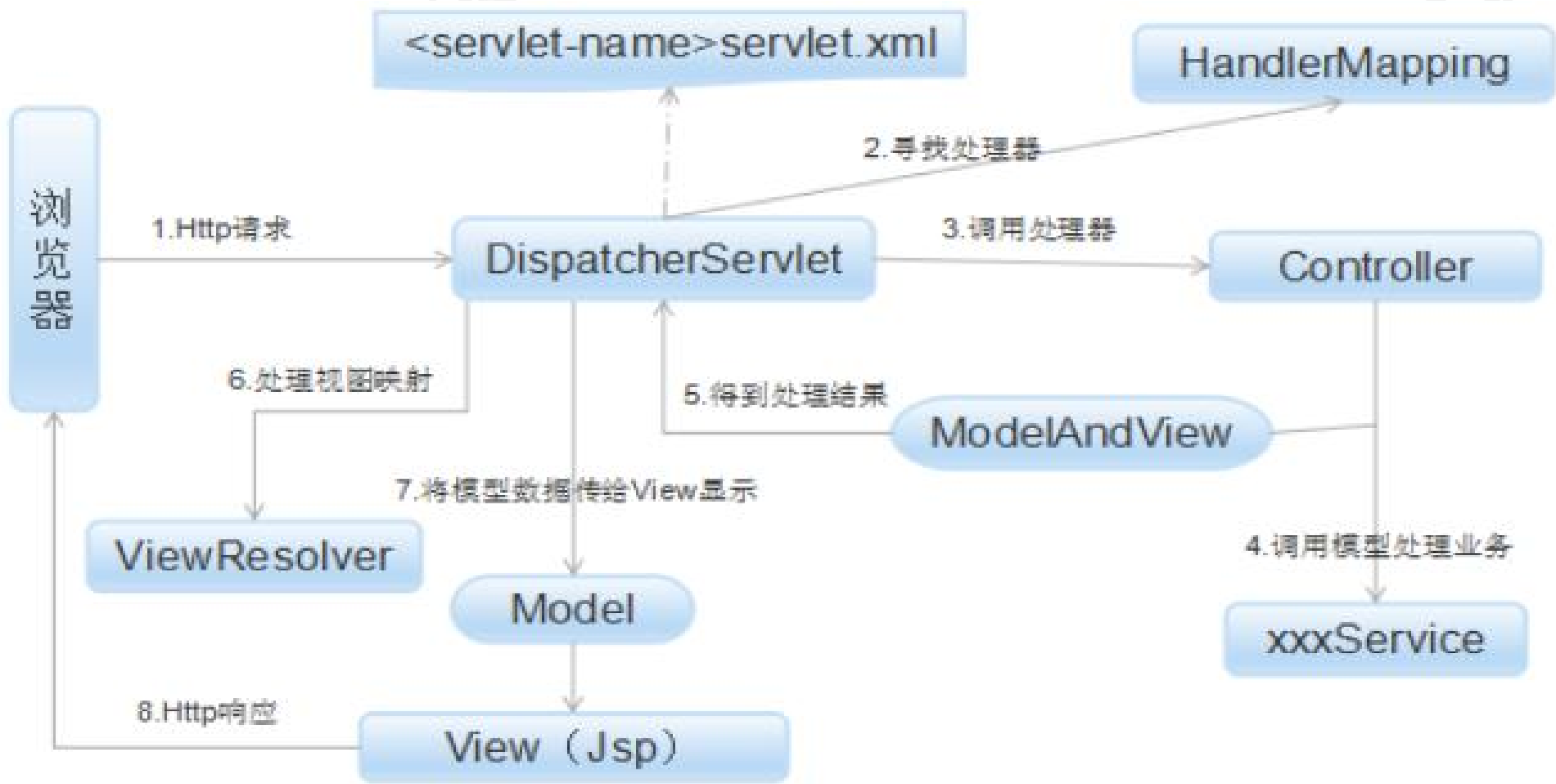
从宏观考虑， DispatcherServlet是整个Web应用控制器；从微观考虑， Controller是单Http请求处理过程中控制器， ModelAndView是Http请求过程中返回模型(Model)和视图(View)。

(4) **ViewResolver**接口： Spring提供的视图解析器 (ViewResolver) 在Web应用中查找 View对象，从而将相应结果渲染给客户。

Spring MVC框架介绍



Spring MVC运行原理



Spring MVC框架介绍

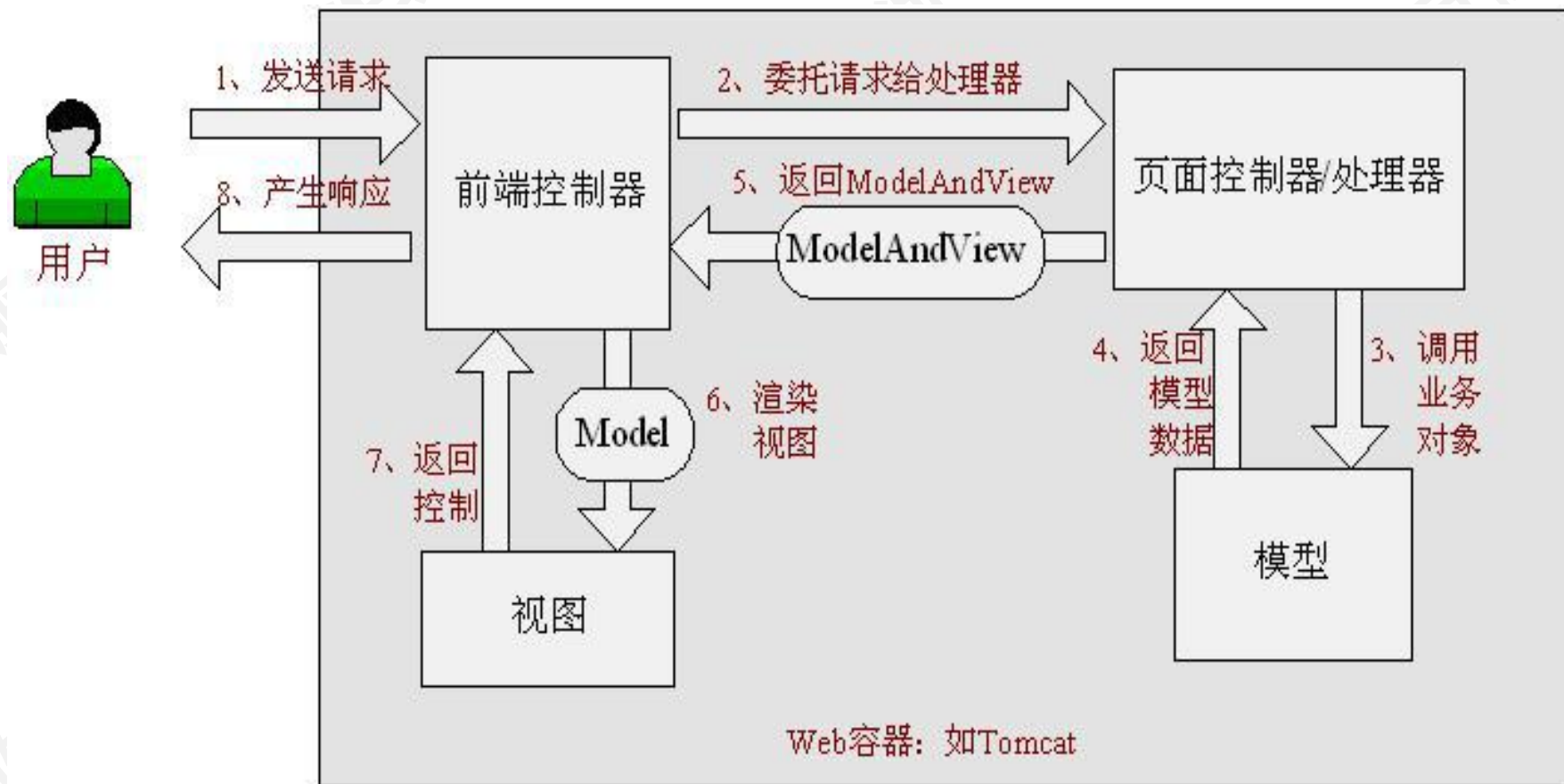


Spring MVC运行原理 **可结合上页图**

- (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ViewResolver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框架介绍

Spring MVC处理流程 (备注: 重点掌握)



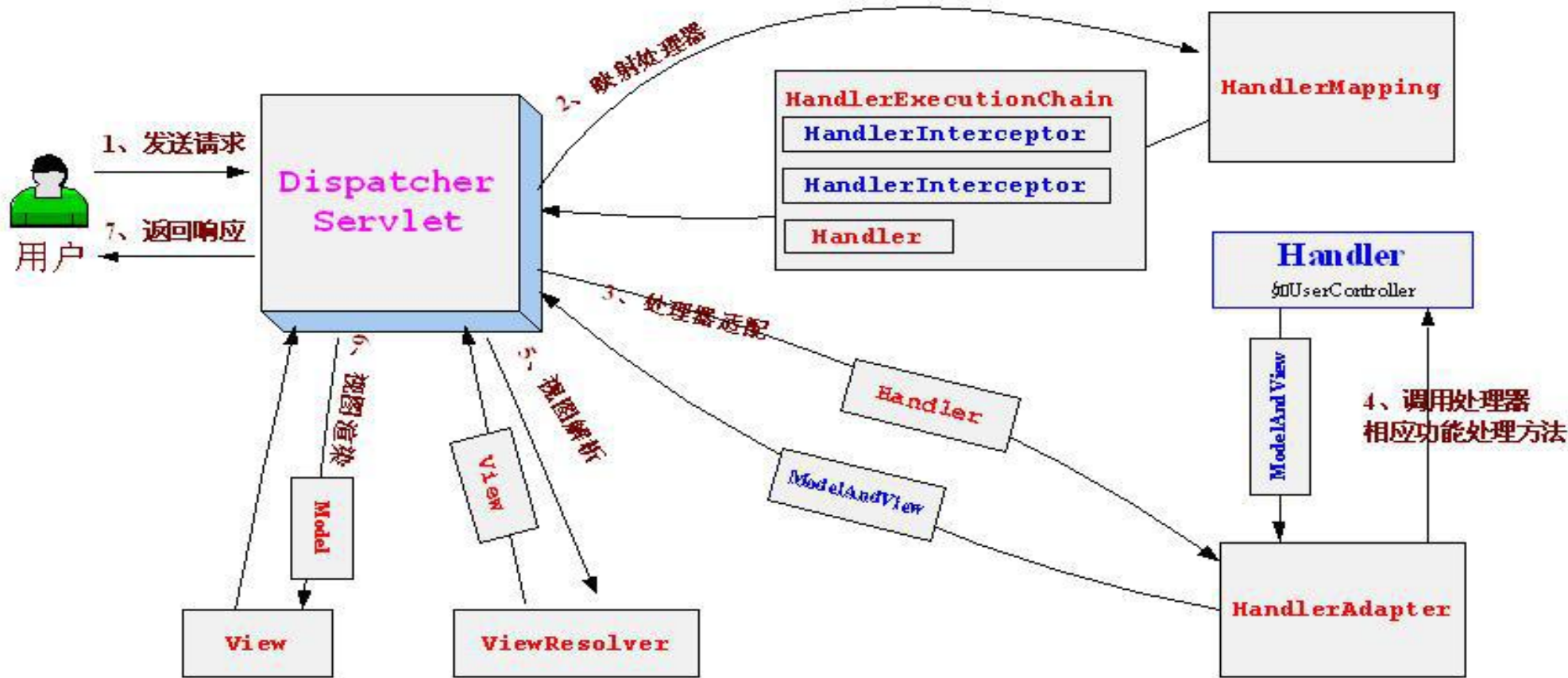
具体执行步骤如下： 可结合[上页图](#)

1. 首先用户发送请求——>前端控制器，前端控制器根据请求信息（如URL）来决定选择哪一个页面控制器进行处理并把请求委托给它，即以前的控制器的控制逻辑部分；图中的1、2步骤；
2. 页面控制器接收到请求后，进行功能处理，首先需要收集和绑定请求参数到一个对象，这个对象在Spring Web MVC中叫命令对象，并进行验证，然后将命令对象委托给业务对象进行处理；处理完毕后返回一个ModelAndView（模型数据和逻辑视图名）；图的3、4、5步骤；
3. 前端控制器收回控制权，然后根据返回的逻辑视图名，选择相应的视图进行渲染，并把模型数据传入以便视图渲染；图的步骤6、7；
4. 前端控制器再次收回控制权，将响应返回给用户，图的步骤8；至此整个结束。

Spring MVC框架介绍



Spring MVC核心架构图 (备注：重要的事情说三遍)



DispatcherServlet是整个Spring MVC的核心

DispatcherServlet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的各个组成组件，并装配到DispatcherServlet中。

备注：**DispatcherServlet是前置控制器**，配置在**web.xml文件**中的。拦截匹配的请求，Servlet拦截匹配规则要自己定义，把拦截下来的请求，依据相应的规则分发到目标Controller来处理。

目 录

- Spring MVC框架介绍
- **Spring MVC快速入门**
- SpringMVC基于Maven构建
- SpringMVC基于注解开发
- SpringMVC获取请求中的参数
- SpringMVC中响应json
- SpringMVC实现文件上传
- SpringMVC拦截器
- SpringMVC异常处理

Spring MVC快速入门



Spring MVC是java开源框架Spring Framework的一个独立的模块

MVC(Model View Controller)框架,在项目中开辟MVC层次架构,Spring MVC框架主要是操作MVC层面中的C,也就是控制。

在工程项目中的MVC

模型:即业务模型,负责完成业务中的数据通信处理,对应项目中的service和dao

视图:渲染数据,生成页面.对应项目中的JSP等

控制器:直接对接请求,控制MVC流程,调度模型,选择视图.对应项目中的controller等

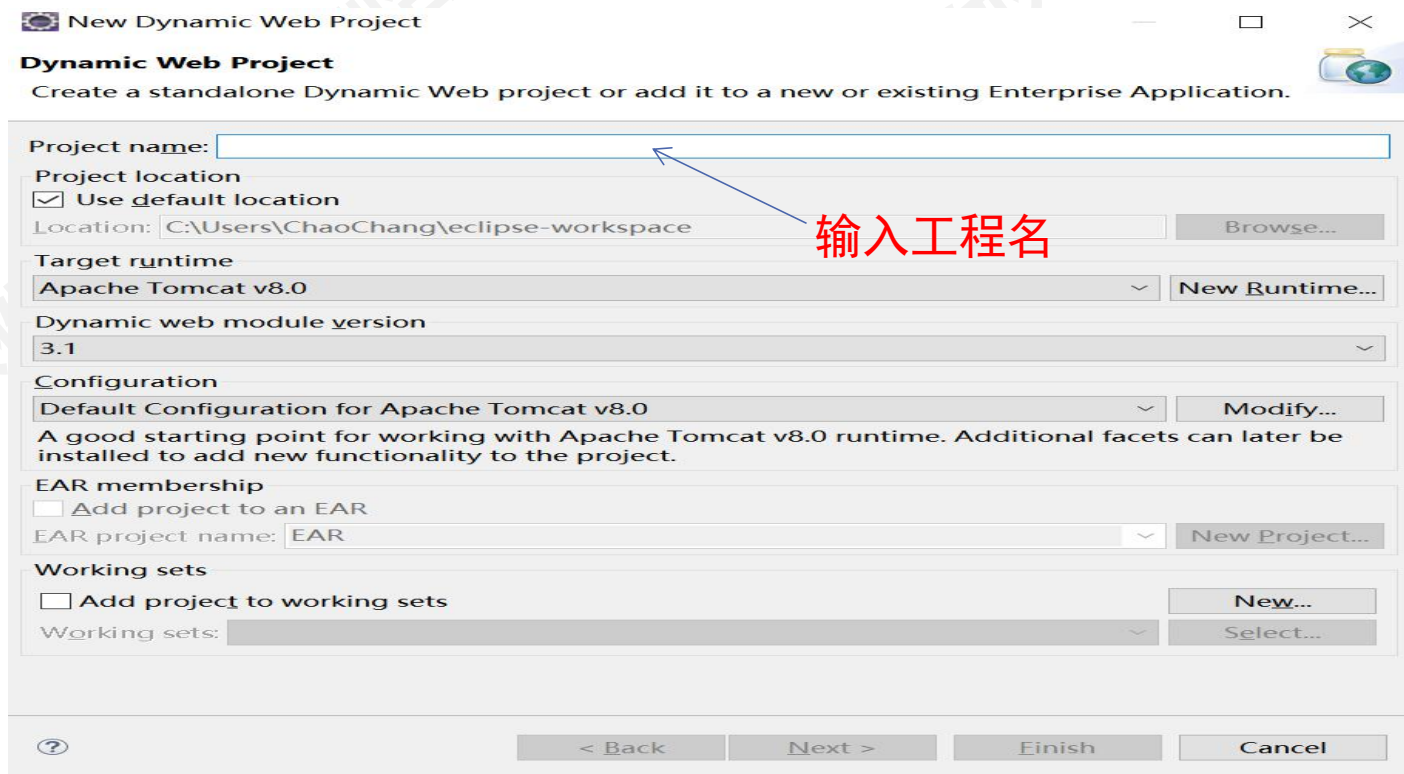
备注: MVC是线下软件开发中最流行的代码结构形态,人们根据负责的不同逻辑,将项目中的代码分成M V C3个层次,层次内部职责单一,层次之间耦合度低,符合低耦合 高内聚的设计理念.也实际有利于项目的长期维护。

Spring MVC快速入门

SpringMVC—入门的简单HelloSpringMVC实例

1.先用eclipse开发工具创建HelloSpringMVCweb工程

打开eclipse工具，点击左上角“File”按钮，选择“New”，再点击“Dynamic Web project”按钮到以下对话框，输入完工程名后点击“Finish”按钮完成工程创建。



SpringMVC—入门的简单HelloSpringMVC实例

2. 配置web.xml文件

用Eclipse新建一个动态的web工程，在WEB-INF文件下的web.xml中配置

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd" id="WebApp_ID" version="3.1">
  <servlet>
    <servlet-name>springmvc</servlet-name>
    <servlet-class>
      org.springframework.web.servlet.DispatcherServlet//前端控制器
    </servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
```

SpringMVC—入门的简单HelloSpringMVC实例

2. 配置web.xml文件

```
<servlet-mapping>  
  <servlet-name>springmvc</servlet-name>  
  <url-pattern>/</url-pattern>  
</servlet-mapping>  
</web-app>
```

备注：Spring MVC的入口 DispatcherServlet，把所有的请求都提交到该Servlet。

Spring MVC快速入门



SpringMVC—入门的简单HelloSpringMVC实例

3. 在WEB-INF文件夹下创建springmvc-servlet.xml文件

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">
<beans>
  <bean id="simpleUrlHandlerMapping"
    class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
    <property name="mappings">
      <props>
        <prop key="/index">helloWorldController</prop>
      </props>
    </property>
  </bean>
  <bean id="helloWorldController"
    class="com.ibm.controller.HelloWorldController"></bean>
</beans>
```

Spring MVC快速入门



SpringMVC—入门的简单HelloSpringMVC实例

4. 创建控制类

```
package com.ibm.controller;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.ModelAndView;
@Controller
public class HelloWorldController {
    @RequestMapping("/index") //结合以上springmvc-servlet.xml文件对应讲解
    public ModelAndView handleRequest(HttpServletRequest request,
        HttpServletResponse response) throws Exception {
        ModelAndView mav = new ModelAndView("index"); //index.jsp
        mav.addObject("message", "Hello Spring MVC");//返回message的消息内容
        return mav;
    }
}
```

Spring MVC快速入门

SpringMVC—入门的简单HelloSpringMVC实例

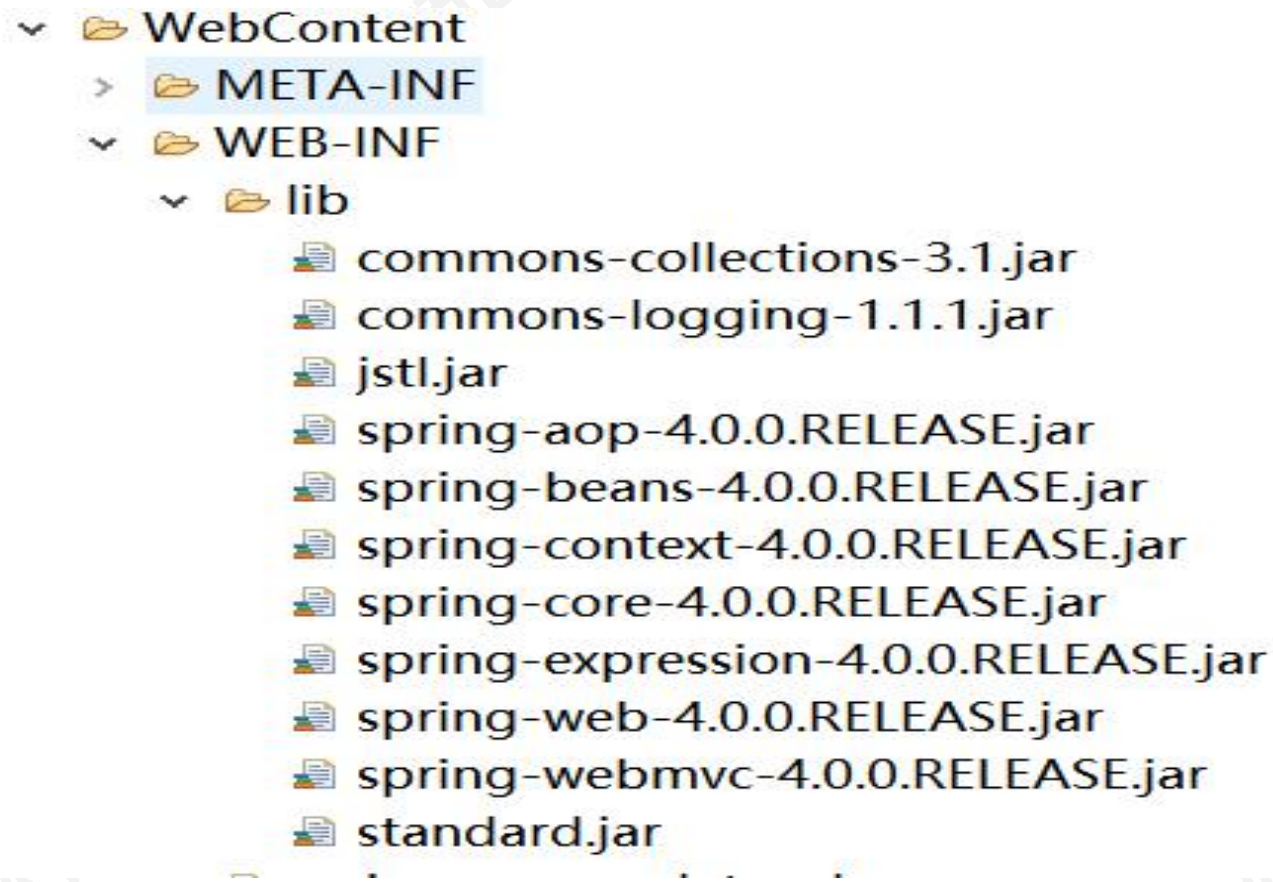
5. 在WebContent目录中创建index.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!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=ISO-8859-1">
<title>index page</title>
</head>
<body>
<h1>Hello SpringMVC</h1> //是首页展示的内容
</body>
</html>
```

Spring MVC快速入门

SpringMVC—入门的简单HelloSpringMVC实例

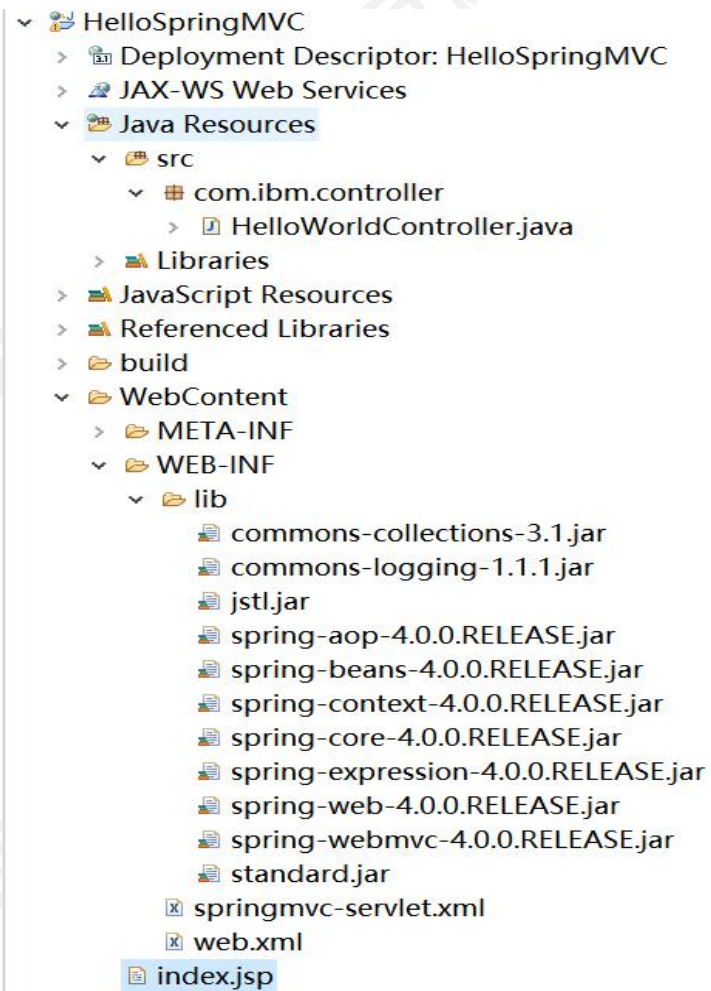
6. 在WebContent目录下的lib目录中导入SpringMVC必要的jar包



Spring MVC快速入门

SpringMVC—入门的简单HelloSpringMVC实例

6. HelloSpringMVC工程代码结构图如下:

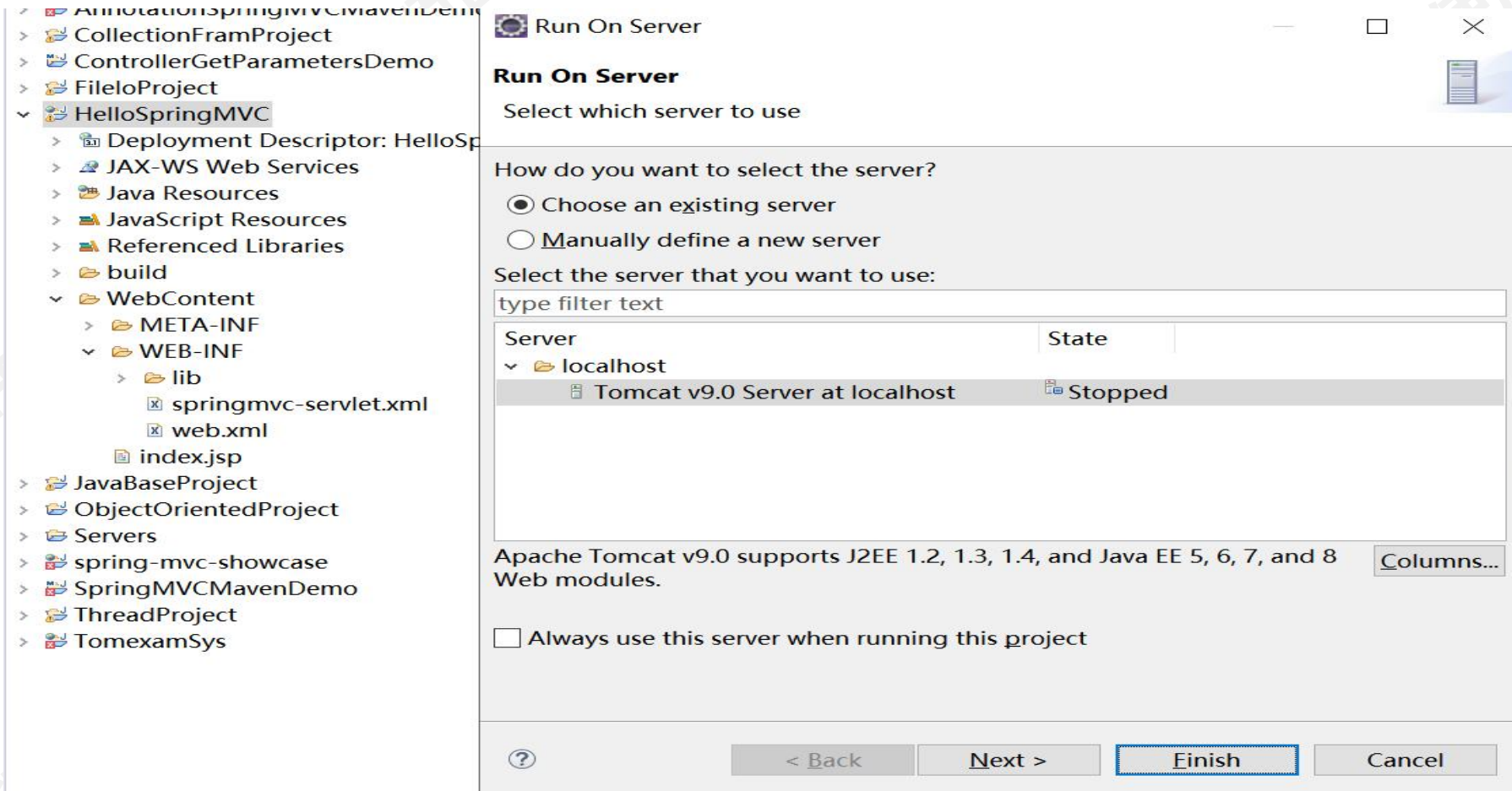


Spring MVC快速入门

SpringMVC—入门的简单HelloSpringMVC实例

7. HelloSpringMVC工程用tomcat部署

选中工程名，右击鼠标，选“Run As”后选“Run Server”，最后点击“Finish”完成。



Spring MVC快速入门

SpringMVC—入门的简单HelloSpringMVC实例

7. HelloSpringMVC工程启动后，打开浏览器输入：<http://localhost:8080/HelloSpringMVC/>后，结果如下：



Spring MVC快速入门

SpringMVC—入门的简单HelloSpringMVC实例 总结

- 1.用户访问 /index
- 2.根据web.xml中的配置 所有的访问都会经过DispatcherServlet
- 3.根据 根据配置文件springmvc-servlet.xml , 访问路径/index
- 4.会进入HelloWorldController类
- 5.在HelloWorldController中指定跳转到页面index.jsp
- 6.在index.jsp中显示内容信息

随堂练习作业:

根据以上示例，搭建项目名为**SpringMVCDemo**的SpringMVC项目，
展示**HelloSpringMVC**的jsp页面。

目 录

- Spring MVC框架介绍
- Spring MVC快速入门
- **SpringMVC基于Maven构建**
- SpringMVC基于注解开发
- SpringMVC获取请求中的参数
- SpringMVC中响应json
- SpringMVC实现文件上传
- SpringMVC拦截器
- SpringMVC异常处理

Spring MVC基于Maven构建

使用**maven**搭建简单的SpringMVC框架 工程实例名: **SpringMVCMavenDemo**

提问: 为啥要引入Maven工具来搭建SpringMVC框架工程啊?

传统的方式做SpringMVC项目要导入很多有关SpringMVC的jar包, 还需要提前官网上下好再导入工程, 这样既麻烦又浪费时间。

而我们引入maven构建项目的话, 就不要我们在下载jar包和导包那么麻烦了。

只要我们在**Maven的pom.xml中做简单配置**, maven在构建项目时候自动下载我们配置好的包了。

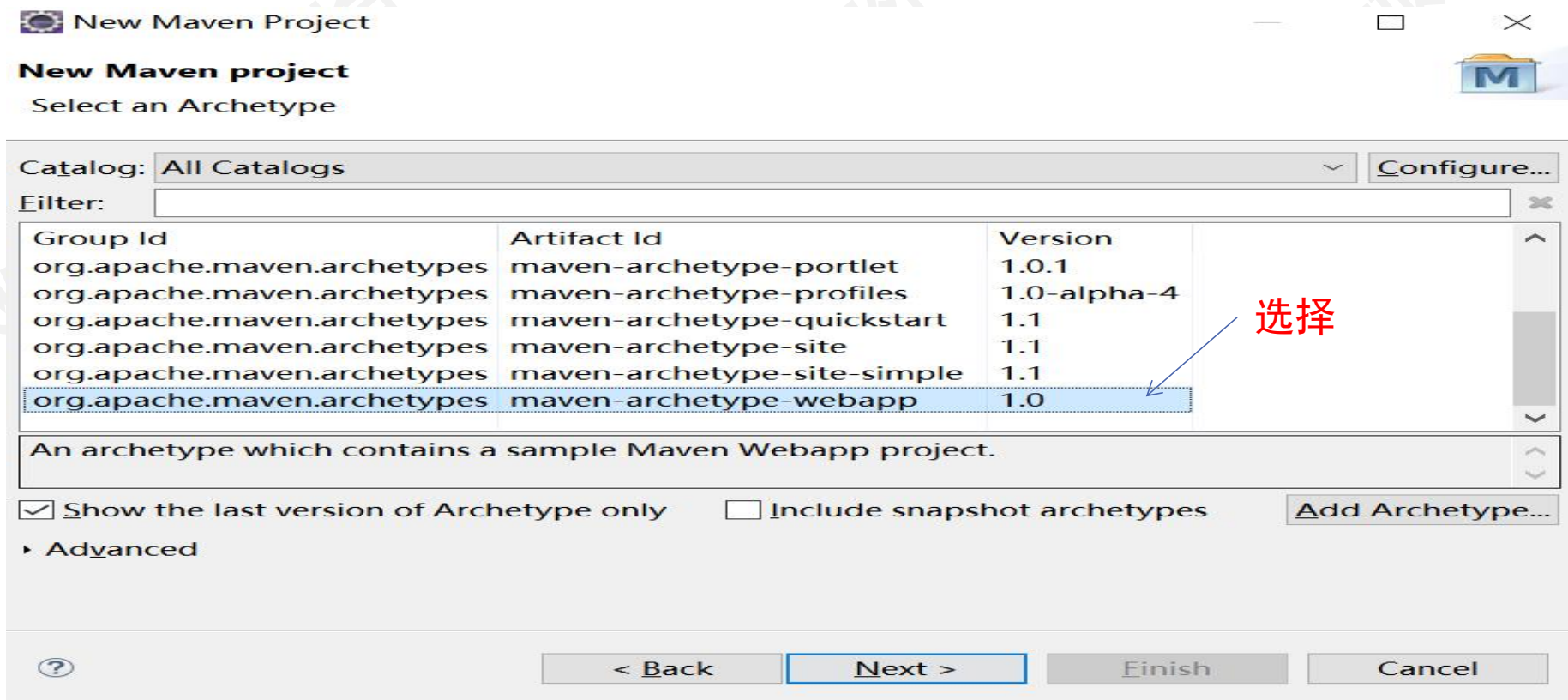
注意: 提前需配置好Maven的环境变量。

备注: 目前做项目大家都用的是基于Maven工具来构建项目的。

Spring MVC基于Maven构建

使用maven搭建简单的SpringMVC框架 工程实例名: **SpringMVCMavenDemo**

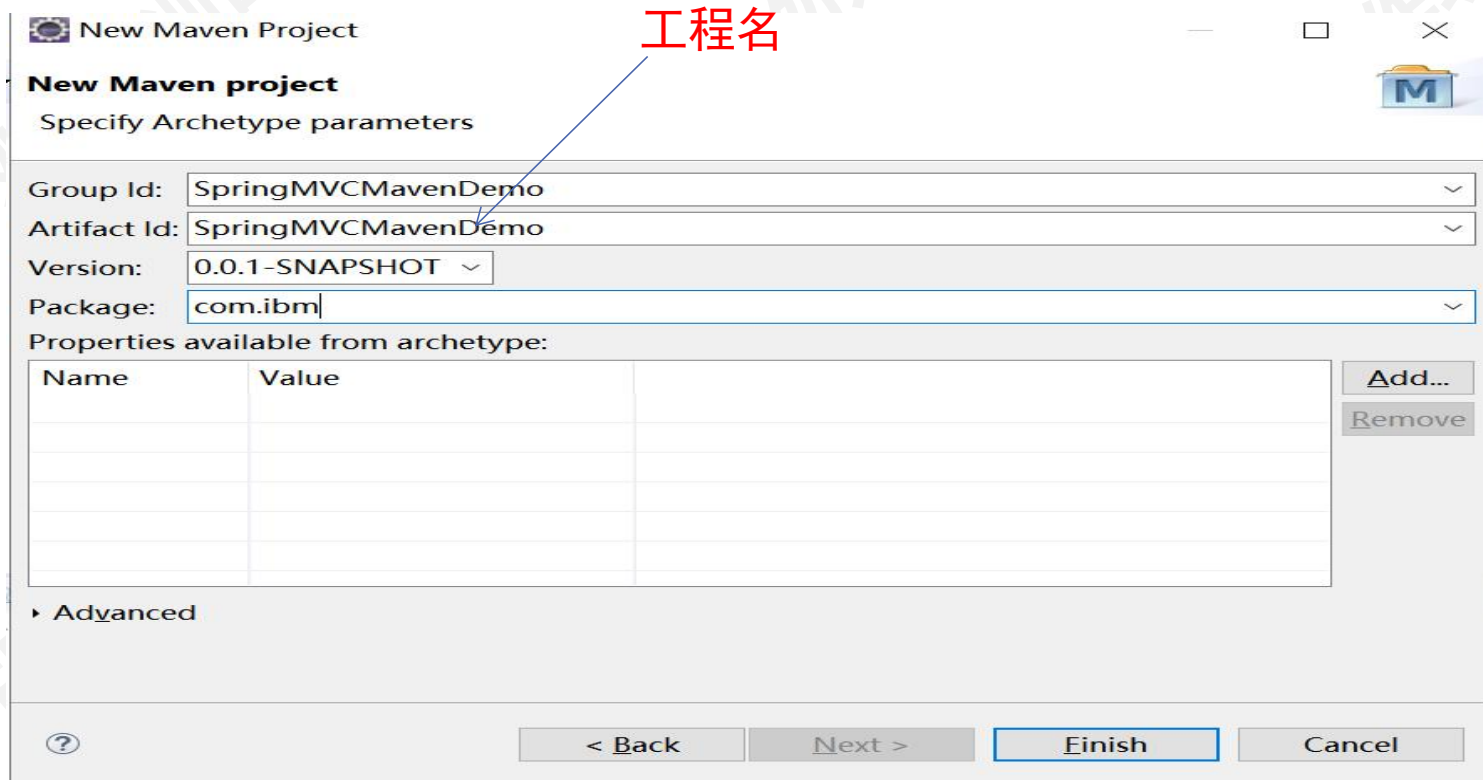
1.打开eclipse开发工具，在左上角点击“File”按钮，然后选“New”按钮，选择“Maven Project”按钮，弹出以下图所示的对话框。



Spring MVC基于Maven构建

使用maven搭建简单的SpringMVC框架实例

2.输入工程名称，如下图所示：



New Maven Project

New Maven project

Specify Archetype parameters

Group Id: SpringMVCMavenDemo

Artifact Id: SpringMVCMavenDemo

Version: 0.0.1-SNAPSHOT

Package: com.ibm

Properties available from archetype:

Name	Value

Advanced

< Back Next > Finish Cancel

Spring MVC基于Maven构建



使用maven搭建简单的SpringMVC框架实例

3.修改WEB-INF目录下的web.xml文件

```
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xmlns="http://java.sun.com/xml/ns/javaee"
          xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
          xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
          version="3.0">
  <display-name>Archetype Created Web Application</display-name>
  <!-- spring MVC的核心就是DispatcherServlet，使用springMVC的第一步就是将下面的servlet
放入web.xml的servlet-name属性非常重要，默认情况下，DispatchServlet会加载这个名字-
servlet.xml的文件，如下，就会加载dispatcher-servlet.xml，也是在WEB-INF目录下。 -->
  <servlet>
    <servlet-name>dispatcher</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
```

Spring MVC基于Maven构建

使用maven搭建简单的SpringMVC框架实例

3.修改WEB-INF目录下的web.xml文件

<!-- 设置dispatchservlet的匹配模式，通过把dispatchservlet映射到/，默认servlet会处理所有的请求，包括静态资源 -->

```
<servlet-mapping>
```

```
  <servlet-name>dispatcher</servlet-name>
```

```
  <url-pattern>/</url-pattern>
```

```
</servlet-mapping>
```

```
<!-- 字符集过滤器 -->
```

```
<filter>
```

```
  <filter-name>encodingFilter</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>
```

Spring MVC基于Maven构建

使用maven搭建简单的SpringMVC框架实例

3.修改WEB-INF目录下的web.xml文件

```
<init-param>  
    <param-name>forceEncoding</param-name>  
    <param-value>true</param-value>  
</init-param>  
</filter>  
<filter-mapping>  
    <filter-name>encodingFilter</filter-name>  
    <url-pattern>/*</url-pattern>  
</filter-mapping>  
  
</web-app>
```

说明：其中字符集过滤器部分不是必须的，但是如果要处理中文的话，最好还是加上。

Spring MVC基于Maven构建



使用maven搭建简单的SpringMVC框架实例

3.在WEN-INF目录下创建dispatcher-servlet.xml(web.xml中servlet-name对应的名称)

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:context="http://www.springframework.org/schema/context"
  xmlns:mvc="http://www.springframework.org/schema/mvc"
  xmlns:p="http://www.springframework.org/schema/p"
  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"
  default-lazy-init="true">
```

Spring MVC基于Maven构建



使用maven搭建简单的SpringMVC框架实例

3.在WEN-INF目录下创建dispatcher-servlet.xml(web.xml中servlet-name对应的名称)

<!-- 通过mvc:resources设置静态资源，这样servlet就会处理这些静态资源，而不通过控制器 -->

<!-- 设置不过滤内容，比如:css,jquery,img 等资源文件 -->

<mvc:resources location="/*.html" mapping="/**/*.html" />

<mvc:resources location="/css/*" mapping="/css/**" />

<mvc:resources location="/js/*" mapping="/js/**" />

<mvc:resources location="/images/*" mapping="/images/**" />

<!-- 添加注解驱动 -->

<mvc:annotation-driven />

<!-- 默认扫描的包路径 -->

<context:component-scan base-package="**com.ibm.controller**" />

Spring MVC基于Maven构建

使用maven搭建简单的SpringMVC框架实例

3.在WEN-INF目录下创建dispatcher-servlet.xml(web.xml中servlet-name对应的名称)

```
<mvc:view-controller path="/" view-name="index" />
<bean class="org.springframework.web.servlet.view.UrlBasedViewResolver">
    <property name="viewClass"
value="org.springframework.web.servlet.view.JstlView"> </property>
    <!-- 配置jsp路径前缀 -->
    <property name="prefix" value="/"> </property>
    <!-- 配置URI后缀 -->
    <property name="suffix" value=".jsp"> </property>
</bean>
</beans>
```

说明： 其中<!-- 默认扫描的包路径 -->

<**context:component-scan** base-package="com.ibm.controller" />中的路径, com.ibm.controller, 是需要在src/main/java中创建的包, 用来放Java代码。

Spring MVC基于Maven构建

使用maven搭建简单的SpringMVC框架实例

4.使用maven引入SpringMVC所依赖的jar包，修改pom.xml文件

4.1 添加属性，在<project>标签中

```
<properties>
```

```
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
```

```
    <spring.version>3.1.2.RELEASE</spring.version>
```

```
</properties>
```

4.2添加依赖，在<dependencies>标签中

```
<dependency>
```

```
    <groupId>org.springframework</groupId>
```

```
    <artifactId>spring-webmvc</artifactId>
```

```
    <version>${spring.version}</version>
```

```
</dependency>
```

```
<dependency>
```

```
    <groupId>org.springframework</groupId>
```

```
    <artifactId>spring-jdbc</artifactId>
```

```
    <version>${spring.version}</version>
```

```
</dependency>
```

Spring MVC基于Maven构建

使用maven搭建简单的SpringMVC框架实例

4.使用maven引入SpringMVC所依赖的jar包, 修改pom.xml文件

4.2添加依赖, 在<dependencies>标签中

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-aop</artifactId>
    <version>${spring.version}</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>${spring.version}</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-test</artifactId>
    <version>${spring.version}</version>
</dependency> 备注: 添加完成之后, 通过更新工程, 就会自动引入所依赖的jar包
```

Spring MVC基于Maven构建

使用maven搭建简单的SpringMVC框架实例

5. 前台页面

在maven工程中生成的有一个index.jsp，将其修改成一下内容，很简单，只是一个登陆框，提交一个form表单，中的用户名和密码。

```
<%@ 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">
```

```
</head>
```

```
<body>
```

```
<h2>Hello World!</h2>
```

```
<form action="login">
```

```
    用户名: <input id="username" name="username" type="text"> </input> <br>
```

```
    密 码: <input id="username" name="password" type="password"> </input> <br>
```

```
    <input type="submit">
```

```
</form>
```

```
</body>
```

```
</html>
```

Spring MVC基于Maven构建

使用**maven**搭建简单的SpringMVC框架实例

6. 写**controller**层代码，用来响应前台请求

```
package com.ibm.controller;  
import org.springframework.stereotype.Controller;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RequestParam;  
import org.springframework.web.bind.annotation.ResponseBody;  
@Controller  
public class LoginController {  
    @RequestMapping("login") //用来处理前台的login请求  
    private @ResponseBody String hello(  
        @RequestParam(value = "username", required = false)String username,  
        @RequestParam(value = "password", required = false)String password){  
        return "Hello "+username+",Your password is: "+password; //返回值  
    }  
}
```


Spring MVC基于Maven构建

使用maven搭建简单的SpringMVC框架实例

7. 部署到tomcat上之后，浏览器输入<http://localhost:8080/SpringMVCMavenDemo/> 运行效果如下：



点击“提交”按钮后结果如下：



Spring MVC基于Maven构建

本节总结

使用maven搭建简单的SpringMVC框架实例

1. 配置好Maven环境变量，创建Maven工程
2. 项目工程web.xml配置
3. pom.xml文件引入jar依赖
4. 用Maven构建项目
5. 通过tomcat发布项目

随堂练习作业:

根据以上示例，用Maven工具构建项目名为MavenSpringMVCDemo的SpringMVC项目，展示HelloSpringMVC的jsp页面。

目 录

- Spring MVC框架介绍
- Spring MVC快速入门
- SpringMVC基于Maven构建
- **SpringMVC基于注解开发**
- SpringMVC获取请求中的参数
- SpringMVC中响应json
- SpringMVC实现文件上传
- SpringMVC拦截器
- SpringMVC异常处理

注解的介绍

注解：是一种分散式的元数据，与源代码紧绑定。

注解的好处：数据绑定用注解，很少改变的用注解，类型安全。

- 1、XML配置起来有时候冗长，此时注解可能是更好的选择，如jpa的实体映射；注解在处理一些不变的元数据时有时候比XML方便的多，比如springmvc的数据绑定，如果用xml写的代码会多的多；
- 2、**注解**最大的好处就是**简化了XML配置**；其实大部分注解一定确定后很少会改变，所以在一些中小项目中使用注解反而提供了开发效率；
- 3、**注解**相对于XML的另一个好处是**类型安全**的，XML只能在运行期才能发现问题。

备注：不管是约定大于配置、注解还是XML配置也好，没有哪个是最优的，在合适的场景选择合适的解决方案这才是重要的

Spring MVC基于注解开发

Spring MVC的常用注解

@Controller

负责注册一个bean 到spring 上下文中

@RequestMapping

注解为控制器指定可以处理哪些 URL 请求

@ResponseBody

该注解用于将Controller的方法返回的对象，通过适当的HttpMessageConverter转换为指定格式后，写入到Response对象的body数据区。

@Autowired 为Spring提供的注解，需要导入包

org.springframework.beans.factory.annotation.Autowired;只按照byType注入。

Spring MVC基于注解开发



Spring MVC的常用注解

@ModelAttribute

在方法定义上使用 @ModelAttribute 注解：Spring MVC 在调用目标处理方法前，会先逐个调用在方法级上标注了 @ModelAttribute 的方法

在方法的入参前使用 @ModelAttribute 注解：可以从隐含对象中获取隐含的模型数据中获取对象，再将请求参数 - 绑定到对象中，再传入入参将方法入参对象添加到模型中。

@RequestParam

在处理方法入参处使用 @RequestParam 可以把请求参数传递给请求方法。

@PathVariable

绑定 URL 占位符到入参

@ExceptionHandler

注解到方法上，出现异常时会执行该方法。

Spring MVC基于注解开发



SpringMVC基于注解开发 示例 不用配置web.xml，也不要再在XML文件中配置，全用注解实现

1. **pom.xml**文件中进行配置,这是Maven工具避免导jar包的，pom.xml是项目级别的配置文件。

```
<properties>
  <spring.version>4.3.5.RELEASE</spring.version>
</properties>
<dependencies>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>${spring.version}</version>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>${spring.version}</version>
  </dependency>
```

Spring MVC基于注解开发

1.pom.xml文件中配置

<!-- 安全起见, 添加, 这个对任务调度, jml、ftl等提供支持 -->

```
<dependency>
```

```
    <groupId>org.springframework</groupId>
```

```
    <artifactId>spring-context-support</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>
```

Spring MVC基于注解开发

1.pom.xml文件中配置

```
<!-- jsp和servlet -->
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>javax.servlet-api</artifactId>
  <version>3.0.1</version>
  <scope>provided</scope>
</dependency>
<dependency>
  <groupId>javax.servlet.jsp</groupId>
  <artifactId>jsp-api</artifactId>
  <version>2.2</version>
  <scope>provided</scope>
</dependency>
```


Spring MVC基于注解开发



1.pom.xml文件中配置

<!-- 为@Response等的json数据绑定提供支持 -->

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-databind</artifactId>

<version>2.9.6</version>

</dependency>

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-core</artifactId>

<version>2.9.6</version>

</dependency>

</dependencies>

Spring MVC基于注解开发



1.pom.xml文件中配置

```
<build>
  <plugins>
    <!-- 编译插件 -->
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.5.1</version>
      <configuration>
        <source>1.7</source>
        <target>1.7</target>
        <encoding>UTF-8</encoding>
      </configuration>
    </plugin>
```

Spring MVC基于注解开发



1.pom.xml文件中进行配置

```
<!-- tomcat 插件 -->
<plugin>
  <groupId>org.eclipse.jetty</groupId>
  <artifactId>jetty-maven-plugin</artifactId>
  <version>9.3.7.v20160115</version>
  <configuration>
    <httpConnector>
      <port>8080</port>
      <host>localhost</host>
    </httpConnector>
    <webApp>
      <contextPath>/spring4webSocket</contextPath>
    </webApp>
  </configuration>
</plugin>
</plugins>
</build>
```

Spring MVC基于注解开发



SpringMVC基于注解开发 示例 不用配置web.xml，也不要再在XML文件中配置，全用注解实现

2. **SpringMVConfig配置类** 作用相当于dispatcher-servlet.xml

package com.ibm.config;

import org.springframework.context.annotation.Configuration;

import org.springframework.http.converter.HttpMessageConverter;

import org.springframework.http.converter.json.Jackson2ObjectMapperBuilder;

import org.springframework.http.converter.json.MappingJackson2HttpMessageConverter;

import org.springframework.web.servlet.config.annotation.*;

import java.text.SimpleDateFormat;

import java.util.List;

Spring MVC基于注解开发



SpringMVC基于注解开发 示例 不用配置web.xml，也不要再在XML文件中配置，全用注解实现

2. **SpringMVCConfig配置类** 作用相当于**dispatcher-servlet.xml**

@Configuration

@EnableWebMvc //这个注解类似于 <mvc:annotation-driven/> (前例**dispatcher-servlet.xml**中)

```
public class SpringMVCConfig extends WebMvcConfigurerAdapter {  
    @Override  
    public void addViewControllers(ViewControllerRegistry registry){  
        registry.addViewController("/").setViewName("/index");  
    }  
    @Override  
    public void configureViewResolvers(ViewResolverRegistry registry) {  
        //spring4 默认不是jsp，而是Thymeleaf  
        registry.jsp().prefix("/jsp").suffix(".jsp");  
    }  
}
```


Spring MVC基于注解开发



2. SpringMVCConfig配置类 作用相当于dispatcher-servlet.xml

@Override

```
public void addResourceHandlers(ResourceHandlerRegistry registry) {  
    //静态资源处理  
    registry.addResourceHandler("/static/**").addResourceLocations("/static/");  
}
```

@Override

```
public void configureMessageConverters(List<HttpMessageConverter<?>> converters)  
{  
    //增加jackson 的支持, json转换器, 支持@RequestBody等  
    Jackson2ObjectMapperBuilder builder = new Jackson2ObjectMapperBuilder()  
        .dateFormat(new SimpleDateFormat("yyyy-MM-dd"));  
    converters.add(new MappingJackson2HttpMessageConverter(builder.build()));  
}  
}
```

Spring MVC基于注解开发



SpringMVC基于注解开发 示例 不用配置web.xml，也不要XML文件中配置，全用注解实现

3. SpringMVCConfig配置类 扫描包

```
package com.ibm.config;
```

```
import org.springframework.context.annotation.ComponentScan;
```

```
import org.springframework.context.annotation.Configuration;
```

```
import org.springframework.context.annotation.Import;
```

```
@Configuration
```

```
@ComponentScan("com.ibm.controller")
```

```
@Import(SpringMVCConfig.class)//关联前面的SpringMVCConfig类
```

```
public class ProductConfig {
```

```
}
```

Spring MVC基于注解开发



SpringMVC基于注解开发 示例 不用配置web.xml，也不要再在XML文件中配置，全用注解实现

4. **MyWebApplicationInitializer**类作用相当于web.xml

package com.ibm.config;

import org.springframework.web.WebApplicationInitializer;

import org.springframework.web.context.support.AnnotationConfigWebApplicationContext;

import org.springframework.web.servlet.DispatcherServlet;

import javax.servlet.ServletContext;

import javax.servlet.ServletException;

import javax.servlet.ServletRegistration;

Spring MVC基于注解开发

4. **MyWebApplicationInitializer**类作用相当于web.xml

```
public class MyWebApplicationInitializer implements WebApplicationInitializer {  
    @Override  
    public void onStartup(ServletContext servletContext) throws ServletException {  
        //Spring  
        AnnotationConfigWebApplicationContext ctx = new  
        AnnotationConfigWebApplicationContext();  
        ctx.register(ProductConfig.class);  
        ctx.setServletContext(servletContext);  
        //Spring MVC  
        ServletRegistration.Dynamic registration =  
        servletContext.addServlet("servletDispatcher", new DispatcherServlet(ctx));  
        registration.setLoadOnStartup(1);  
        registration.addMapping("/");  
    }  
}
```

Spring MVC基于注解开发

SpringMVC基于注解开发 示例 不用配置web.xml，也不要再在XML文件中配置，全用注解实现

5. 创建TestController类 进行与前端交互 此处可在浏览器测试restful数据

package com.ibm.controller;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.ResponseBody;

@Controller

@RequestMapping("/test")

public class TestController {

 @RequestMapping("/do")

 @ResponseBody

 public String test(){

 System.out.println("This is test AnnotationSpringMVC restful ");

 return "ok!";//浏览器测试http://localhost:8080/AnnotationSpringMVCMavenDemo/test/do

 }

}

Spring MVC基于注解开发

SpringMVC基于注解开发 示例 不用配置web.xml，也不要再在XML文件中配置，全用注解实现

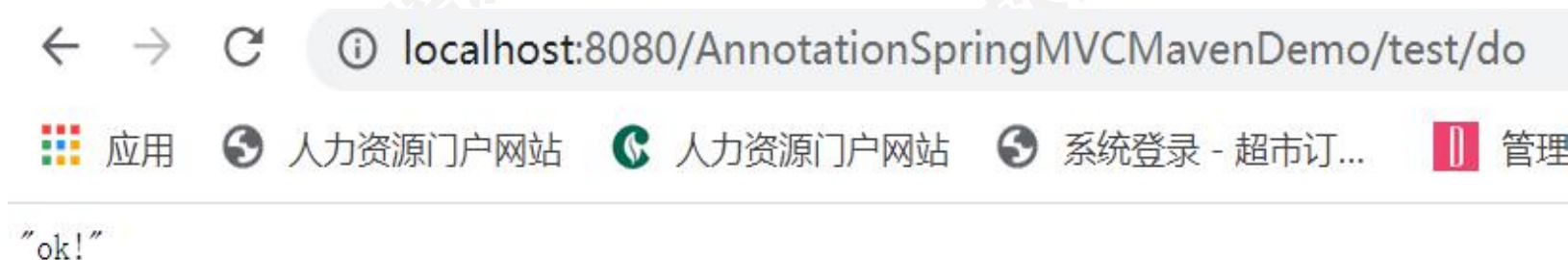
6. 部署到tomcat上之后，运行效果如下：

在浏览器输入：<http://localhost:8080/AnnotationSpringMVCMavenDemo/> 对应index.jsp内容



浏览器输入：<http://localhost:8080/AnnotationSpringMVCMavenDemo/test/do>

后台代码TestController中返回内容如下：



SpringMVC基于注解开发 示例总结

- 1、@Configuration表明配置类
- 2、@EnableWebMvc启用SpringMVCConfig类相当于<mvc:annotation-driven/>
- 3、@ComponentScan扫描识别我们的controller，返回响应前端请求。

在SpringMVC 中，控制器Controller 负责处理由DispatcherServlet 分发的请求，它把用户请求的数据经过业务处理层处理之后封装成一个Model，然后再把该Model 返回给对应的View 进行展示。在SpringMVC 中提供了一个非常简便的定义Controller 的方法，你无需继承特定的类或实现特定的接口，只需使用@Controller 标记一个类是Controller，然后使用@RequestMapping 和@RequestParam 等一些注解用以定义URL 请求和Controller 方法之间的映射，这样的Controller 就能被外界访问到。此外Controller 不会直接依赖于HttpServletRequest 和HttpServletResponse 等HttpServletRequest 对象，它们可以通过Controller 的方法参数灵活的获取到。

@Controller 用于标记在一个类上，使用它标记的类就是一个SpringMVC Controller 对象。

分发处理器将会扫描使用了该注解的类的方法，并检测该方法是否使用了 **@RequestMapping** 注解。@Controller 只是定义了一个控制器类，而使用@RequestMapping 注解的方法才是真正处理请求的处理器。单单使用@Controller 标记在一个类上还不能真正意义上的说它就是SpringMVC 的一个控制器类，因为这个时候Spring 还不认识它。那么要如何做Spring 才能认识它呢？这个时候就需要我们把这个控制器类交给Spring 来管理。有两种方式：

(1) 在SpringMVC 的配置文件中定义MyController 的bean 对象。

(2) 在SpringMVC 的配置文件中告诉Spring 该到哪里去找标记为@Controller 的Controller 控制器。（包扫描）

@RequestMapping是一个用来处理请求地址映射的注解，可用于类或方法上。用于类上，表示类中的所有响应请求的方法都是以该地址作为父路径。

@RequestMapping注解有六个属性，下面我们分成三类进行说明：

1、value, method;

value: 指定请求的实际地址，指定的地址可以是URI Template 模式；

method: 指定请求的method类型， GET、POST、PUT、DELETE等；

2、consumes, produces

consumes: 指定处理请求的提交内容类型（Content-Type），例如
application/json, text/html;

produces: 指定返回的内容类型，仅当request请求头中的(Accept)类型中包含该指定类型才返回；

3、params, headers

params: 指定request中必须包含某些参数值是，才让该方法处理。

headers: 指定request中必须包含某些指定的header值，才能让该方法处理请求。

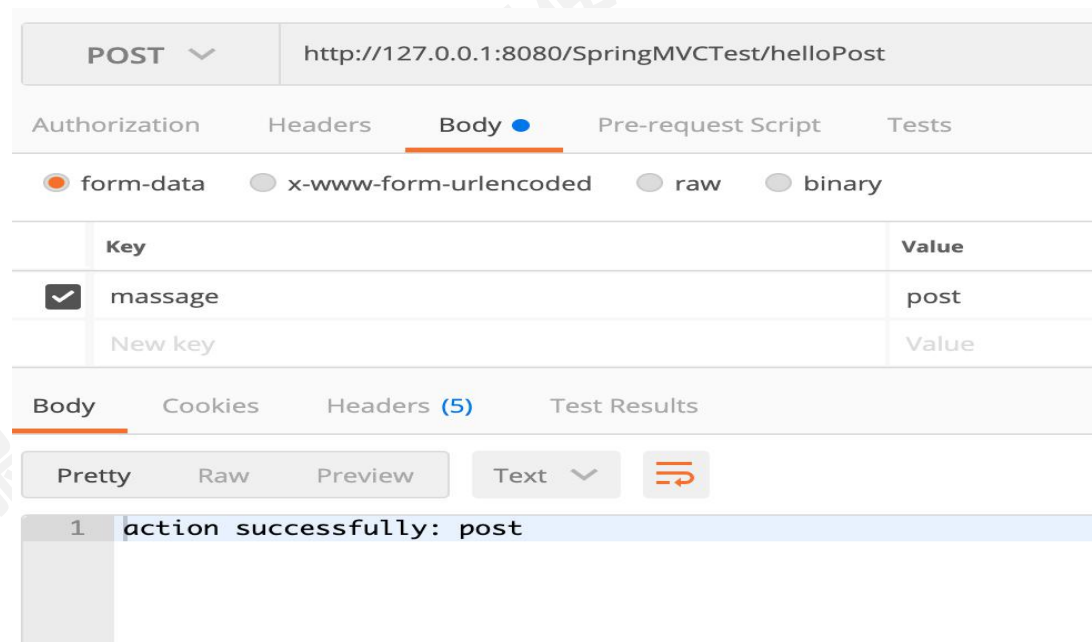
SpringMVC注解小结

@RequestMapping举例说明

```
@RequestMapping(value="/helloPost", method=RequestMethod.POST)
@ResponseBody
public String helloPost(HttpServletRequest request, Model model) {
    String message = request.getParameter("message");
    return "action successfully: " + message;
}
```

客户端通过:

http://ip:port/上下文/web.xml中配置的servlet-mapping路径/user/sayhello来访问say方法, 且仅支持客户端的post请求。



SpringMVC注解小结

@ModelAttribute应用于方法的参数中，定义该参数是一个实体参数。

```
@RequestMapping("/student/add")  
public String add(@ModelAttribute Student stu, Model model) {  
    model.addAttribute("message", stu);  
    return "hello";  
}
```

SpringMVC注解小结



@PathVariable用于将请求URL中的模板变量映射到功能处理方法的参数上，即取出uri模板中的变量作为参数。

```
@RequestMapping("/hello5/{message}")
public String hello5(@PathVariable String message, Model model) {
    model.addAttribute("message", message);
    return "hello";
}
```

SpringMVC注解小结

@RequestParam主要用于在SpringMVC后台控制层获取参数，类似一种是 `request.getParameter("name")`，它有三个常用参数：`defaultValue = "0"`，`required = false`，`value = "isApp"`；`defaultValue` 表示设置默认值，`required` 通过boolean设置是否是必须要传入的参数，`value` 值表示接受的传入的参数类型。

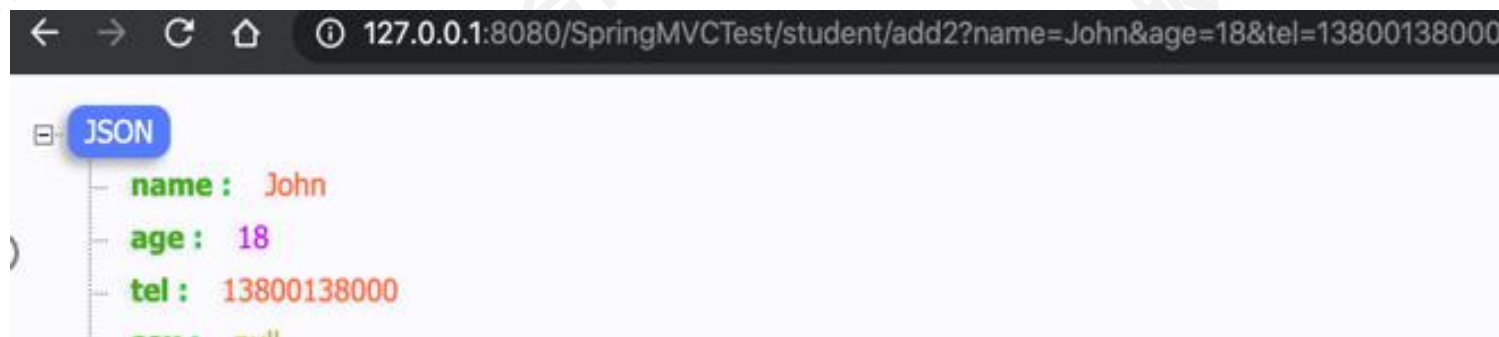
```
@RequestMapping("/hello3")
public String hello3(@RequestParam(required = false, value="message") String message, Model model) {
    model.addAttribute("message", message);
    return "hello";
}
```

SpringMVC注解小结

@ResponseBody作用：该注解用于将Controller的方法返回的对象，通过适当的HttpMessageConverter转换为指定格式后，写入到Response对象的body数据区。

使用时机：返回的数据不是html标签的页面，而是其他某种格式的数据时（如json、xml等）使用；

```
@RequestMapping("/student/add2")
@ResponseBody
public Student add2(Student stu, Model model) {
    return stu;
}
```



SpringMVC注解小结

@DateTimeFormat

用于定义请求参数的日期格式，通过pattern属性指定客户端的日期格式。用于Controller方法的日期参数前以及若是对象参数的属性的话，用在对应的属性前。

```
@RequestMapping("/hello7/{message}/{id}")
public String hello7(@PathVariable String message,@PathVariable int id,
    @RequestParam @DateTimeFormat(pattern="yyyy-MM-dd HH:mm:ss") Date birthday ,Model model) {
    model.addAttribute("message", message + id + " birthday is :" + birthday);
    return "hello";
}

private String name;
private Integer age;
private String tel;
private String sex;
@DateTimeFormat(pattern="yyyy-MM-dd")
private Date birthday;
```

备注：这些注解是我们项目上常用到的，希望大家能理解掌握。

随堂练习作业:

根据以上示例，用Maven工具构建SpringMVC项目，不用配置web.xml，也不要再在XML文件中配置，全用注解实现
浏览器输入：<http://localhost:8080/AnnotationSpringMVCMavenDemo/test/do>
后台代码TestController中返回内容“ success”。

目 录

- Spring MVC框架介绍
- Spring MVC快速入门
- SpringMVC基于Maven构建
- SpringMVC基于注解开发
- **SpringMVC获取请求中的参数**
- SpringMVC中响应json
- SpringMVC实现文件上传
- SpringMVC拦截器
- SpringMVC异常处理

获取请求中的参数

提问： SpringMVC中的Controller如何获取请求中的参数？

SpringMVC中的Controller获取请求中的参数方式（基于XML配置讲解）：

1. 可以从 HttpServletRequest、HttpServletResponse、HttpSession 中获取请求中的参数。
2. 通过简单的数据类型来接收参数值。
3. 通过简单的pojo类来接收参数。
4. 通过包装类来接收参数。
5. 通过集合类型获取。

获取请求中的参数

获取请求中的参数1-从 HttpServletRequest、HttpServletResponse 中获取请求中的参数

//**ControllerGetParametersDemo**工程中测试类LoginController类的getParamByReq方法

```
package com.ibm.controller;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
@Controller
public class LoginController {
    // 通过 HttpServletRequest 和 HttpServletResponse 以及 HttpSession 来获取请求中的参数
    @RequestMapping("/login")
    public void getParamByReq(HttpServletRequest request, HttpServletResponse response) {
        String username = request.getParameter("username");
        String password = request.getParameter("password");
        System.out.println(username);
        System.out.println(password);
    }
}
```

获取请求中的参数

获取请求中的参数1-从 HttpServletRequest、HttpServletResponse 中获取请求中的参数
//ControllerGetParametersDemo工程中测试类LoginController类的getParamByReq方法
通过HttpServletRequest、HttpServletResponse、HttpSession 中获取请求中的参数，在项目正常启动后，打开浏览器输入以下地址：

http://localhost:8080/ControllerGetParametersDemo/login?username=zhangsan&password=123

查看控制台打印结果：—————→

```
method1-----  
zhangsan  
123
```

备注：地址栏中：username=zhangsan 最好username传英文参数，也是按正规的项目要求来传参。如果真想传汉字，需要配置tomcat以及https与http的对应。这里就不详细介绍了。

获取请求中的参数

获取请求中的参数2-通过简单的数据类型来接收参数值

例如：int，string，double，float等这些简单的数据类型参数来获取请求中的参数。

//ControllerGetParametersDemo工程中测试类LoginController类的getParamBySimple方法

// 通过简单的数据类型来获取请求中的参数

```
@RequestMapping("/testBySimple")
```

```
public void getParamBySimple(String username, String password) {
```

```
    System.out.println("method2-----");
```

```
    System.out.println(username);
```

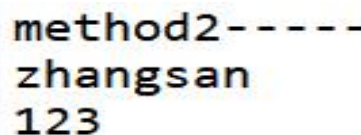
```
    System.out.println(password);
```

```
}
```

备注： 打开浏览器输入以下地址

http://localhost:8080/ControllerGetParametersDemo/testBySimple?username=zhangsan&password=123

eclipse控制台打印结果:



```
method2-----  
zhangsan  
123
```


获取请求中的参数

获取请求中的参数3-通过简单的pojo类来接收参数

//ControllerGetParametersDemo工程中测试类LoginController类的getParamByPojo方法

//通过简单的pojo类来接收参数

@RequestMapping("/testByPojo")

public void getParamByPojo(User user) { //com.ibm.pojo包下的实体User类

System.out.println("method3-----");

System.out.println(user.getUsername());

System.out.println(user.getPassword());

}

备注: 打开浏览器输入以下地址

http://localhost:8080/ControllerGetParametersDemo/testByPojo?username=zhangsan&password=123

eclipse控制台打印结果: →

```
method3-----  
zhangsan  
123
```

获取请求中的参数

获取请求中的参数4-通过包装类来接收参数

//ControllerGetParametersDemo工程中com.ibm.vo包下UserVO类

//包装类代码如下

```
package com.ibm.vo;  
import com.ibm.pojo.User;  
public class UserVO {  
    private User user; //com.ibm.pojo.User  
    public User getUser() {  
        return user;  
    }  
    public void setUser(User user) {  
        this.user = user;  
    }  
    @Override  
    public String toString() {  
        return "UserVO [user=" + user.toString() + "];"  
    }  
}
```

获取请求中的参数

获取请求中的参数4-通过**包装类**来接收参数

//**ControllerGetParametersDemo**工程中**com.ibm.pojo**包下**User**类

```
package com.ibm.pojo;  
public class User { //User类  
    private int userId;  
    private String username;  
    private String password;  
    private int age;  
    public int getUserId() {  
        return userId;  
    }  
    public void setUserId(int userId) {  
        this.userId = userId;  
    }  
    public String getUsername() {  
        return username;  
    }  
}
```

获取请求中的参数

获取请求中的参数4-通过**包装类**来接收参数

//**ControllerGetParametersDemo**工程中**com.ibm.pojo**包下**User**类

//package **com.ibm.pojo**包下**User**类

```
public void setUsername(String username) {  
    this.username = username;  
}  
public String getPassword() {  
    return password;  
}  
public void setPassword(String password) {  
    this.password = password;  
}  
public int getAge() {  
    return age;  
}  
public void setAge(int age) {  
    this.age = age;  
}
```

获取请求中的参数



获取请求中的参数4-通过包装类来接收参数

//ControllerGetParametersDemo工程中测试类LoginController类中saveUserVO方法

// 通过包装类来获取页面信息

```
@RequestMapping("/saveUserVO")
```

```
public String saveUserVO(UserVO userVO) {
```

```
    System.out.println("method4-----");
```

```
    System.out.println("name:" + userVO.getUser().getUsername());
```

```
    System.out.println("age:" + userVO.getUser().getAge());
```

```
    return "userEdit";
```

```
}
```

备注: ControllerGetParametersDemo工程项目正常启动后, 打开浏览器输入以下地址:

<http://localhost:8080/ControllerGetParametersDemo/>

获取请求中的参数

获取请求中的参数4-通过包装类来接收参数

备注: ControllerGetParametersDemo 工程项目正常启动后, 打开浏览器输入以下地址:

http://localhost:8080/ControllerGetParametersDemo/ 然后正常跳转到以下页面



用户信息

用户名称:

用户密码:

用户年龄:

保存

获取请求中的参数

获取请求中的参数4-通过包装类来接收参数

备注: ControllerGetParametersDemo 工程项目正常启动后, 打开浏览器输入以下地址:

http://localhost:8080/ControllerGetParametersDemo/ 然后正常跳转到以下页面输入信息后, 点击“保存”按钮, 然后查看eclipse控制台打印信息:

```
method4-----  
name: zhangsan  
age: 23
```

同时页面成功跳转如下页面:



用户新增/修改页面,通过包装类来获取提交的参数

用户名称:

用户密码:

用户年龄:

获取请求中的参数

获取请求中的参数5-通过集合类型获取

如果前台传过来的是一组数据该怎么接受呢?

针对单个属性的话, 可以使用数组。(针对批量删除)

```
// 根据用户id数组批量删除用户
```

```
@RequestMapping("/batchDelUser")
```

```
public String batchDelUser(Integer[] userId) {
```

```
    for (int i = 0; i < userId.length; i++) {
```

```
        System.out.println(userId[i]);
```

```
    }
```

```
    return "userList";
```

```
}
```

备注:此例和前面讲到获取参数一样, 只是换成数组而已, 这里就不举例讲了

获取请求中的参数

获取请求中的参数5-通过集合类型获取

针对多个属性的话，仍需要使用包装类。（针对批量修改）

// 根据用户id数组批量修改用户

```
@RequestMapping("/batchUpdateUser")
```

```
public String batchUpdateUser(UserVO userVO) {
```

```
    System.out.println("得到的用户数: " + userVO.getUserList().size());
```

```
    for (int i = 0; i < userVO.getUserList().size(); i++) {
```

```
        System.out.println(userVO.getUserList().get(i));
```

```
    }
```

```
    return "userList";
```

```
}
```

```
}
```

备注:此例和前面讲到通过包装类获取参数一样。

获取请求中的参数

获取请求中的参数方式（基于XML配置讲解） 小结

SpringMVC中的Controller获取请求中的参数方式（基于XML配置讲解）：

1. 可以从 HttpServletRequest、HttpServletResponse、HttpSession 中获取请求中的参数。
2. 通过简单的数据类型来接收参数值。
3. 通过简单的pojo类来接收参数。
4. 通过包装类来接收参数。
5. 通过集合类型获取。

备注：具体使用哪种获取请求的参数主要看前台页面是怎样传参而定。

提问：那么 SpringMVC中的Controller基于注解怎样获取获取请求中的参数？

请接着往下看

获取请求中的参数

SpringMVC中的Controller获取请求中的参数方式（基于注解）：

- 1.request获取值。
- 2.使用路径变量@PathVariable绑定页面url路径的参数，用于进行页面跳转。
- 3.通过@RequestParam绑定页面传来的参数。
- 4.自动注入，实体类属性有setter，getter方法，前端form表单的name对应实体的属性名，后台直接可以通过该实体类自动把参数绑定到类的属性。
- 5.使用RequestBody接受前端传来的json数组，对象。ResponseBody把数据返回。

获取请求中的参数

SpringMVC中的Controller获取请求中的参数方式（基于注解）：

1.request获取值。

```
@RequestMapping("/request.action")
```

```
public String request(HttpServletRequest request){
```

```
    String value= (String) request.getAttribute("value");
```

```
    String val=request.getParameter("value");
```

```
    return "index";
```

```
}
```

request的getAttribute和getParameter有什么区别呢？

获取请求中的参数



SpringMVC中的Controller获取请求中的参数方式（基于注解）：

1.request获取值。

request的getAttribute和getParameter有什么区别呢？

getAttribute:取得是setAttribute设定的值，session范围的值，可以设置为object，对象，字符串；getAttribute获取的值是web容器内部的，是具有转发关系的web组件之间共享的值；用于服务端重定向

getParameter:取得是从web的form表单的post/get，或者url传过来的值，只能是String字符串；getParameter获取的值是web端传到服务端的，是获取http提交过来的数据；用于客户端重定向。

获取请求中的参数

SpringMVC中的Controller获取请求中的参数方式（基于注解）：

2.通过@RequestParam绑定页面传来的参数

@Controller

```
public class BaseController {  
    @RequestMapping("/goUrl/{folder}/{file}")  
    public String goUrl(@PathVariable String folder,@PathVariable String file){  
        return folder+"/"+file;  
    }  
}
```

获取请求中的参数

SpringMVC中的Controller获取请求中的参数方式（基于注解）：

3.通过@RequestParam绑定页面传来的参数

效果跟String id=request.getParameter（"id"）是一样的

```
@RequestMapping("/test.action")
```

```
public void test(@RequestParam("id") String id){
```

```
    System.out.println("id:"+id);
```

```
}
```

获取请求中的参数

SpringMVC中的Controller获取请求中的参数方式（基于注解）：

4.自动注入，实体类属性有setter，getter方法，前端form表单的name对应实体的属性名，后台直接可以通过该实体类自动把参数绑定到类的属性

比如:实体类

```
public class Content {  
    private String content;//属性  
    public String getContent() {  
        return content;  
    }  
    public void setContent(String content) {  
        this.content = content;  
    }  
}
```

获取请求中的参数

比如:form表单

```
<form action="<%=request.getContextPath()%>/content" method="post"
```

```
enctype="multipart/form-data">
```

```
    商品描述: <textarea name="content" rows="2" cols="20"></textarea><br>
```

```
    <input type="submit" value="提交"/>
```

```
</form>
```

后台接收数据:

```
@RequestMapping("/content")
```

```
public void contetn(Content content){//实体类
```

```
    System.out.println("content:"+content.getContent());
```

```
}
```

获取请求中的参数

SpringMVC中的Controller获取请求中的参数方式（基于注解）：

5.使用RequestBody接受前端传来的json数组，对象。ResponseBody把数据返回

```
@RequestMapping("/test.action")
```

```
@ResponseBody
```

```
public void test(@RequestBody List<Content> list){
```

```
    for (Content content:list){
```

```
        System.out.println(content.toString());
```

```
    }
```

```
}
```


随堂练习作业:

根据以上示例，用Maven工具构建SpringMVC项目,模拟登陆页面将前端用户名和密码参数传到后台TestController层进行打印验证。

目 录

- Spring MVC框架介绍
- Spring MVC快速入门
- SpringMVC基于Maven构建
- SpringMVC基于注解开发
- SpringMVC获取请求中的参数
- **SpringMVC中响应json**
- SpringMVC实现文件上传
- SpringMVC拦截器
- SpringMVC异常处理

SpringMVC框架中响应Json数据



SpringMVC框架中响应返回Json数据 `@ResponseBody`实现

`@ResponseBody`作用:

该注解用于将Controller的方法返回的对象, 根据HTTP Request Header的Accept的内容, 通过适当的HttpMessageConverter转换为指定格式后, 写入到Response对象的body数据区。

配置返回JSON数据 以项目名**JsonDataSpringMVCMavenDemo**为例

1.在pom.xml文件中添加**jackson**依赖

```
<dependency>
  <groupId>com.fasterxml.jackson.core</groupId>
  <artifactId>jackson-databind</artifactId>
  <version>2.8.7</version>
</dependency>
```

2. 开启<mvc:annotation-driven /> //dispatcher-servlet.xml文件中

SpringMVC框架中响应Json数据

SpringMVC框架中响应返回Json数据 @ResponseBody实现

3.代码举例 以项目名**JsonDataSpringMVC MavenDemo**为例

```
package com.ibm.controller;  
import java.util.ArrayList;  
import java.util.HashMap;  
import java.util.List;  
import java.util.Map;  
import org.springframework.stereotype.Controller;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.ResponseBody;  
import com.ibm.model.User;
```

SpringMVC框架中响应Json数据

SpringMVC框架中响应返回Json数据 `@ResponseBody`实现

3.代码举例 以项目名**JsonDataSpringMVCMavenDemo**为例

`@Controller`

`@RequestMapping("/user")`

```
public class UserController {
```

```
    //测试map集合数据返回Json
```

```
    @RequestMapping("/test")
```

```
    @ResponseBody
```

```
    public Map<String, Object> test(){
```

```
        Map<String , Object> map = new HashMap<String ,Object> ();
```

```
        map.put("id", "ibm");
```

```
        map.put("name", "CC");
```

```
        return map;
```

```
    }
```

SpringMVC框架中响应Json数据

SpringMVC框架中响应返回Json数据 `@ResponseBody`实现

3.代码举例 以项目名**JsonDataSpringMVCMavenDemo**为例

//测试list集合数据返回Json

```
@RequestMapping("/test1")
```

```
@ResponseBody
```

```
public List<String> test1(){
```

```
    List<String> list = new ArrayList<>();
```

```
    list.add("aaa");
```

```
    list.add("bbb");
```

```
    list.add("ccc");
```

```
    return list;
```

```
}
```


SpringMVC框架中响应Json数据

SpringMVC框架中响应返回Json数据 `@ResponseBody`实现

3.代码举例 以项目名**JsonDataSpringMVCMavenDemo**为例

//测试对象格式成Json返回

`@RequestMapping("/test2")`

`@ResponseBody`

`public User test2(){`

`User user = new User(); //package com.ibm.model包下的User类`

`user.setUsername("JiangMenTrain");`

`user.setAge(25);`

`return user;`

`}`

SpringMVC框架中响应Json数据

3.代码举例 以项目名JsonDataSpringMVCMavenDemo为例

//对象列表返回Json

@RequestMapping("/test3")

@ResponseBody

public List<User> test3(){

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

 User user1 = new User();

 user1.setId(1);

 user1.setUsername("CC");

 user1.setAge(21);

 list.add(user1);

 User user2 = new User();

 user2.setId(2);

 user2.setUsername("CSX");

 user2.setAge(21);

 list.add(user2);

 return list;

}

SpringMVC框架中响应Json数据

SpringMVC框架中响应返回Json数据 @ResponseBody实现

3.代码举例 以项目名JsonDataSpringMVCMavenDemo为例

```
package com.ibm.model;  
public class User { //实体类  
    private int id;  
    private String username;  
    private int age;  
    public int getId() {  
        return id;  
    }  
    public void setId(int id) {  
        this.id = id;  
    }  
    public String getUsername() {  
        return username;  
    }  
}
```

SpringMVC框架中响应Json数据

SpringMVC框架中响应返回Json数据 @ResponseBody实现

3.代码举例 以项目名**JsonDataSpringMVC MavenDemo**为例

```
public void setUsername(String username) {  
    this.username = username;  
}  
public int getAge() {  
    return age;  
}  
public void setAge(int age) {  
    this.age = age;  
}  
}
```

SpringMVC框架中响应Json数据

SpringMVC框架中响应返回Json数据 @ResponseBody实现

4. 测试返回浏览器的Json数据 以项目名**JsonDataSpringMVCMavenDemo**为例
项目**JsonDataSpringMVCMavenDemo**正常启动后,

(1)打开浏览器输入如下地址:

http://localhost:8080/JsonDataSpringMVCMavenDemo/user/test 测试map集合返回json

(2)打开浏览器输入如下地址:

http://localhost:8080/JsonDataSpringMVCMavenDemo/user/test1 测试list集合返回json

(3)打开浏览器输入如下地址:

http://localhost:8080/JsonDataSpringMVCMavenDemo/user/test2 测试对象返回json

(4)打开浏览器输入如下地址:

http://localhost:8080/JsonDataSpringMVCMavenDemo/user/test3 测试对象列表返回json数据。

随堂练习作业:

根据以上示例，用Maven工具构建项目JsonDataSpringMVCMavenDemo正常启动后，UserController类上的@Controller注解换成@RestController注解，去掉test()方法上的@ResponseBody启动服务，练习测试，
打开浏览器输入如下地址：
<http://localhost:8080/JsonDataSpringMVCMavenDemo/user/test> 测试map集合返回json

目 录

- Spring MVC框架介绍
- Spring MVC快速入门
- SpringMVC基于Maven构建
- SpringMVC基于注解开发
- SpringMVC获取请求中的参数
- SpringMVC中响应json
- **SpringMVC实现文件上传**
- SpringMVC拦截器
- SpringMVC异常处理

文件上传

Spring MVC 为文件上传提供了直接的支持，这种支持是通过的 **MultipartResolver** 实现的。

Spring 用 **Jakarta Commons FileUpload** 技术实现了一个 MultipartResolver 实现类：

CommonsMultipartResovler

Spring MVC 上下文中默认没有装配MultipartResovler，因此默认情况下不能处理文件的上传工作，如果想使用Spring 的文件上传功能，需在上下文中配置MultipartResolver

1. springmvc的文件上传依赖于commons-fileupload和commons-io环境，故若需要使用springmvc的上传功能，需要添加以上jar包的依赖。

```
<dependency>
  <groupId>commons-fileupload</groupId>
  <artifactId>commons-fileupload</artifactId>
  <version>LATEST</version>
</dependency>
<dependency>
  <groupId>commons-io</groupId>
  <artifactId>commons-io</artifactId>
  <version>LATEST</version>
</dependency>
```

文件上传

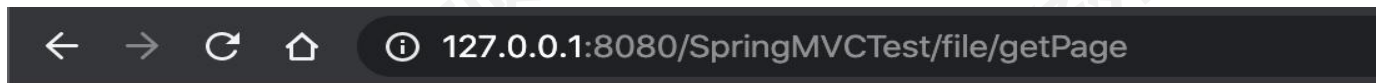
2. 添加文件上传支持的bean，需在配置文件中配置

```
<!-- 用于文件上传 upload file, dependency commons-fileupload.jar -->  
<bean id="multipartResolver"  
class="org.springframework.web.multipart.commons.CommonsMultipartResolver">  
  <!-- 单上传行为的最大上传文件大小，单位为字节 -->  
  <property name="maxUploadSize" value="52428800"/>  
  <!-- 允许上传的单个文件的最大大小，单位为字节 -->  
  <property name="maxUploadSizePerFile" value="52428800"/>  
</bean>
```

3. 跳转上传页面:

```
@Controller
@RequestMapping("/file")
public class UploadController {

    @RequestMapping("/getPage")
    public String uploadFile() {
        return "uploadFile";
    }
}
```



使用spring mvc上传文件

choose file to upload No file chosen

4. 表单提交(uploadFile.jsp)

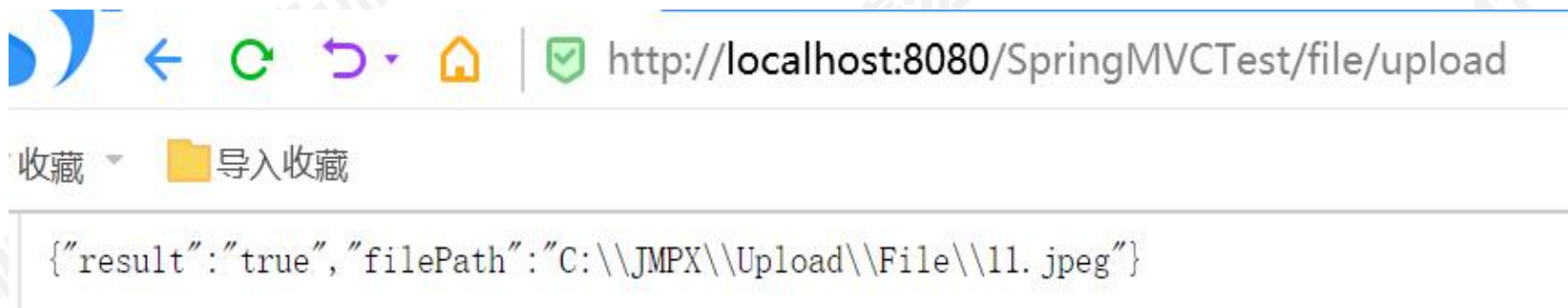
```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"
isELIgnored="false"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Spring 4 MVC -HelloWorld</title>
</head>
<body>
<h1>使用spring mvc上传文件</h1>
  <form action="upload" method="post" enctype="multipart/form-data">
    choose file to upload<input type="file" name="myFile"/>
    <input type="submit"/>
  </form>
</body>
</html>
```


5. Controller后台上传功能

```
@RequestMapping("/upload")
@ResponseBody
public Map uploadFile(@RequestParam("myFile") MultipartFile file, HttpServletRequest req)
    throws IllegalStateException, IOException {
    Map<String, String> result = new HashMap<String, String>();
    // 判断文件是否为空，空则返回失败页面
    if (file.isEmpty()) {
        result.put("result", "false");
        return result;
    }
    // 获取文件存储路径（绝对路径）
    String path = req.getServletContext().getRealPath("/WEB-INF/file");
    // 获取原文件名
    String fileName = file.getOriginalFilename();
    // 创建文件实例
    File filePath = new File(path, fileName);
    // 如果文件目录不存在，创建目录
    if (!filePath.getParentFile().exists()) {
        filePath.getParentFile().mkdirs();
    }
    System.out.println("文件地址: " + filePath);
    result.put("filePath", filePath.toString());
    // 写入文件
    file.transferTo(filePath);
    result.put("result", "true");
    return result;
}
```

6. 测试上传结果

启动服务，浏览器输入：`http://localhost:8080/SpringMVCTest/file/getPage`进行文件上传测试。如果成功上传，会出现以下结果。



练习作业：

根据以上示例，用Maven工具构建SpringMVC项目实现上传本地文件的功能。
实现多个文件上传功能。

目 录

- Spring MVC框架介绍
- Spring MVC快速入门
- SpringMVC基于Maven构建
- SpringMVC基于注解开发
- SpringMVC获取请求中的参数
- SpringMVC中响应json
- SpringMVC实现文件上传
- **SpringMVC拦截器**
- SpringMVC异常处理

什么是拦截器？

Spring MVC中的拦截器（Interceptor）类似于Servlet中的过滤器（Filter），它主要用于拦截用户请求并作相应的处理。例如通过拦截器可以进行权限验证、记录请求信息的日志、判断用户是否登录等。

要使用Spring MVC中的拦截器，就需要对拦截器类进行定义和配置。通常拦截器类可以通过两种方式来定义。

- 1.通过实现**HandlerInterceptor**接口，或继承HandlerInterceptor接口的实现类（如HandlerInterceptorAdapter）来定义。
- 2.通过实现**WebRequestInterceptor**接口，或继承WebRequestInterceptor接口的实现类来定义。

下面以实现HandlerInterceptor接口方式为例，HandlerInterceptor 接口中定义了三个方法，我们就是通过这三个方法来对用户的请求进行拦截处理的。

a). 定义一个实现 HandlerInterceptor 接口 的类



```
4 import javax.servlet.http.HttpServletResponse;
5
6 import org.springframework.web.servlet.HandlerInterceptor;
7 import org.springframework.web.servlet.ModelAndView;
8
9 public class MyInterceptor implements HandlerInterceptor{
10     /**
```


b). preHandle()方法

```
/**
 * 在处理方法之前执行，一般用来做一些准备工作：比如日志，权限检查
 * 如果返回false 表示被拦截，将不会执行处理方法
 * 返回true继续执行处理方法
 */
@Override
public boolean preHandle(HttpServletRequest req, HttpServletResponse resp, Object handler) throws Exception {
    System.out.println("执行preHandler-----"+req.getRemoteHost());
    resp.sendRedirect("index.jsp");
    return false;
}
```

拦截器

c). postHandle()方法

```
/**
 * 在处理方法执行之后，在渲染视图执行之前执行，一般用来做一些清理工作
 */
@Override
public void postHandle(HttpServletRequest req, HttpServletResponse resp, Object handler, ModelAndView mv)
    throws Exception {
    System.out.println("执行postHandler");
}
```

d). postHandle()方法

```
/**
 * 在视图渲染后执行 一般用来释放资源
 */
@Override
public void afterCompletion(HttpServletRequest arg0, HttpServletResponse arg1, Object arg2, Exception arg3)
    throws Exception {
    System.out.println("执行afterCompletion");
}
```

e). 在 springmvc 的配置文件中，添加拦截器配置

```
<mvc:interceptors>
  <!-- 定义一个拦截器的配置 -->
  <mvc:interceptor>
    <!-- mapping 指定哪些url被拦截
    /*表示根路径下的所有请求被拦截-/hello.do
    /**表示根路径及其子路径下的所有请求被拦截/user/add.do
    -->
    <mvc:mapping path="/*"/>
    <!-- 配置拦截器的路径 -->
    <bean class="com.ibm.interceptor.MyInterceptor"/></bean>
  </mvc:interceptor>
</mvc:interceptors>
```

f). 测试

访问 `127.0.0.1:8080/SpringMVCTest/hello` 后, 跳转index.jsp

① `127.0.0.1:8080/SpringMVCTest/index.jsp`

Hello World

[点击跳转](#)

后台日志

```
INFO: Server startup in [5,803] milliseconds  
执行preHandler-----127.0.0.1
```

g). 修改preHandle()方法返回值为true。

@Override

```
public boolean preHandle(HttpServletRequest req, HttpServletResponse resp, Object handler) throws Exception {  
    System.out.println("执行preHandler-----"+req.getRemoteHost());  
    //resp.sendRedirect("index.jsp");  
    return true;  
}
```

127.0.0.1:8080/SpringMVCTest/hello

前端页面:

Hello World: spring MVC

后台日志:

INFO: Server startup in [7,082] milliseconds

执行preHandler-----127.0.0.1

执行postHandler

执行afterCompletion

拦截器



mvc:exclude-mapping 不拦截某个请求

```
<mvc:interceptors>
  <!-- 定义一个拦截器的配置 -->
  <mvc:interceptor>
    <!-- mapping 指定哪些url被拦截
      /*表示根路径下的所有请求被拦截-/hello.do
      /**表示根路径及其子路径下的所有请求被拦截/user/add.do
    -->
    <mvc:mapping path="/**"/>
    <mvc:exclude-mapping path="/hello" /><!-- 不匹配的 -->
    <!-- 配置拦截器的路径 -->
    <bean class="com.ibm.interceptor.MyInterceptor"></bean>
  </mvc:interceptor>
```

随堂练习作业：

根据以上示例，用Maven工具构建SpringMVC项目，练习配置多个拦截器，测试preHandle()、postHandle()、afterCompletion的执行顺序。

目 录

- Spring MVC框架介绍
- Spring MVC快速入门
- SpringMVC基于Maven构建
- SpringMVC基于注解开发
- SpringMVC获取请求中的参数
- SpringMVC中响应json
- SpringMVC实现文件上传
- SpringMVC拦截器
- **SpringMVC异常处理**

SpringMVC通过HandlerExceptionResolver处理程序的异常，包括Handler映射、数据绑定以及目标方法执行时发生的异常

SpringMVC提供的HandlerExceptionResolver的实现类包括:

Type hierarchy of 'org.springframework.web.servlet.HandlerExceptionResolver':



Spring MVC处理异常有3种方式:

- (1) 使用Spring MVC提供的简单异常处理器SimpleMappingExceptionHandler;
- (2) 实现Spring的异常处理接口HandlerExceptionHandler 自定义自己的异常处理器;
- (3) 使用@ExceptionHandler注解实现异常处理。

SpringMVC异常处理



第一种 通过SimpleMappingExceptionHandler类来定义,springmvc.xml内容
自定义的异常可以继承Exception类

```
<bean class="org.springframework.web.servlet.handler.SimpleMappingExceptionHandler">
  <!-- 遇到异常时默认跳转的视图 error其实是error.jsp是视图-->
  <property name="defaultErrorView" value="error"/>
  <!-- 抛出异常后在页面可用 ${errMsg.message} 读出异常信息 -->
  <property name="exceptionAttribute" value="errMsg"/>
  <!-- 自定义的异常转向页面可自定义多个而实现不同异常转向不同页面 -->
  <property name="exceptionMappings">
    <props>
      <prop key="com.ibm.exception.CustomException">error1 </prop>
    </props>
  </property>
</bean>
```


SpringMVC异常处理



第二种 通过实现HandlerExceptionResolver接口来定义 springmvc.xml内容

```
<!-- 全局异常处理器 -->
```

```
<bean class="com.ibm.exception.CustomExceptionHandler"> </bean>
```

接口实现 CustomExceptionHandler是继承了ExceptionHandler类的自定义异常

```
public class CustomExceptionHandler implements HandlerExceptionResolver {
```

```
    @Override
```

```
    public ModelAndView resolveException(HttpServletRequest request, HttpServletResponse  
response, Object handler,Exception ex) {
```

```
        ModelAndView modelAndView = new ModelAndView();
```

```
        modelAndView.addObject("error", errMsg);
```

```
        modelAndView.setViewName("error");
```

```
        return modelAndView;
```

```
    }
```

```
}
```

SpringMVC异常处理



第三种 使用@ExceptionHandler 注解定义

这个更简单了直接在控制器里加入方法,但是必须跟抛异常的方法在同一个控制器里。

@Controller

```
public class LoginAction {  
    @ExceptionHandler(CustomException.class)  
    public String catchException(CustomException e, HttpServletRequest request) {  
        request.setAttribute("errMsg", "注解的异常");  
        return "error";  
    }  
    @RequestMapping("/login.do")  
    public ModelAndView login(int id) throws Exception {  
        if (id == 1) {  
            throw new CustomException("自定义的异常");  
        }  
        ModelAndView modelAndView = new ModelAndView();  
        modelAndView.setViewName("LoginSuccess");  
        return modelAndView;  
    }  
}
```

ExceptionHandlerExceptionHandlerResolver

ExceptionHandlerExceptionHandlerResolver主要处理Handler中，用@ExceptionHandler注解定义的方法

1. @ExceptionHandler注解

可以将@ExceptionHandler添加到任何控制器中，以专门处理由同一控制器中的请求处理（@RequestMapping）方法抛出的异常

在@ExceptionHandler方法的入参中可以加入Exception类型的参数，该参数即对应发生的异常对象

@ExceptionHandler方法的入参不能传入Map。若希望把异常信息传导到页面上，需要使用 ModelAndView 作为返回值

@ExceptionHandler方法标记的异常有优先级的问題，例如发生的是NullPointerException，但是声明的异常有 RuntimeException 和 Exception，此时会根据异常的最近继承关系找到继承深度最浅的那个 @ExceptionHandler注解方法，即标明了RuntimeException的方法

ExceptionHandlerExceptionHandlerResolver内部若找不到@ExceptionHandler注解的话，会找@ControllerAdvice中的@ExceptionHandler注解方法。

2. @ControllerAdvice

@ExceptionHandler只能处理当前handler的控制器中方法抛出的异常，而不是一个更全局的异常处理

@ControllerAdvice:如果当前Handler中找不到@ExceptionHandler方法来处理当前方法出现的异常，则将去@ControllerAdvice标记处来处理异常。

ResponseStatusExceptionHandler

ResponseStatusExceptionHandler处理添加@ResponseStatus注解的异常

@ResponseStatus注解可以标记一个方法，也可以标记一个异常类

它有两个属性：

(1)value属性是http状态码，比如404，500等

(2)reason是错误信息

SpringMVC异常处理



```
/**
 * 异常处理示例
 * @return
 */
@RequestMapping("/testExceptionHandlerExceptionResolver")
public String testExceptionHandlerExceptionResolver(@RequestParam("i") int i){
    System.out.println("result: " + (10 / i));
    return "success";
}
```

备注: 如果url中传递的*i*的值为0, 这会抛出异常。

测试异常抛出如下图:

HTTP Status 500 - Request processing failed; nested exception is java.lang.ArithmeticException: / by zero

type Exception report

message Request processing failed; nested exception is java.lang.ArithmeticException: / by zero

description The server encountered an internal error that prevented it from fulfilling this request.

exception

```
org.springframework.web.util.NestedServletException: Request processing failed; nested exception is java.lang.ArithmeticException: / by zero
    org.springframework.web.servlet.FrameworkServlet.processRequest(FrameworkServlet.java:982)
    org.springframework.web.servlet.FrameworkServlet.doGet(FrameworkServlet.java:861)
    javax.servlet.http.HttpServlet.service(HttpServlet.java:622)
    org.springframework.web.servlet.FrameworkServlet.service(FrameworkServlet.java:846)
    javax.servlet.http.HttpServlet.service(HttpServlet.java:729)
    org.apache.tomcat.websocket.server.WsFilter.doFilter(WsFilter.java:52)
    org.springframework.web.filter.CharacterEncodingFilter.doFilterInternal(CharacterEncodingFilter.java:197)
    org.springframework.web.filter.OncePerRequestFilter.doFilter(OncePerRequestFilter.java:107)
```

root cause

```
java.lang.ArithmeticException: / by zero
    com.springmvc.handlers.SpringMVCTest.testExceptionHandlerExceptionResolver(SpringMVCTest.java:33)
```


随堂练习作业：

实现简单的运行时的异常抛出，进行练习。

SpringMVC异常处理 常见项目异常处理总结

1. HTTP Status 500 - Error instantiating servlet class XXX类

再次刷新 则出现 404 错误

解决办法:

- 1.检查部署的 Tomcat 下的 webapps 下的相应项目里 /WEB-INF/lib 下包是否存在, 没有的话添加即可;
- 2.检查 url 对应的配置文件 (web.xml, applicationContext.xml...) 中对应 servlet-class 标签 或 bean 标签中对应 class 属性配置的 XXX类 是否引用 (检查是否引用用 ctrl + 单击, 能点进去说明能够引用)
- 3.检查部署的 Tomcat 下的 webapps 下的相应项目里 /WEB-INF/classes 是否存在该文件, 应该把 classes 作为编译输出目录;
- 4.检查 Tomcat 是否部署成功;

SpringMVC异常处理



SpringMVC异常处理 常见项目异常处理总结

2. HTTP Status 404 -

type Status report

message

description The requested resource is not available.

解决办法:

检查访问的 URI 地址与 Controller 配置的 URL 地址是否一致;

SpringMVC异常处理



SpringMVC异常处理 常见项目异常处理总结

3. HTTP Status 404 - /XXX/XXX.jsp

type Status report

message /XXX/XXX.jsp

description The requested resource is not available.

解决办法:

检查访问的 URI 地址对应的 Controller 中 ModelAndView.setViewName() 配置的视图资源是否存在;

SpringMVC异常处理 常见项目异常处理总结

4. javax.validation.UnexpectedTypeException: HV000030: No validator could be found for constraint 'javax.validation.constraints.Size' validating type 'java.util.Date' .

产生原因：在相关的 POJO 的校验中加入的校验注解与其被校验的成员变量的类型不统一；

解决方法：

@NotEmpty 只能用于对 String 、 Collection 或 array 字段的注解，其他的就不行；可以换为@NotNull，因为其可以用于任意类型。

SpringMVC异常处理



SpringMVC异常处理 常见项目异常处理总结

5. java.lang.AbstractMethodError: org.mybatis.spring.

transaction.SpringManagedTransaction.getTimeout()Ljava/lang/Integer;

产生原因：错误原因是 mybatis-spring 包版本问题，因为我的Spring 是 4.X 版本的，Mybatis 是 3.X，整合包是1.2.5 版本；

解决方法：解决方案有三种，第一种是更换 Spring 版本，第二种是更换 Mybatis 版本，第三种是升级整合包，建议采用第三种方法。

备注：以上就是SpringMVC项目中常遇到的异常处理。

思无极 行有方 达天下

Thank You!

