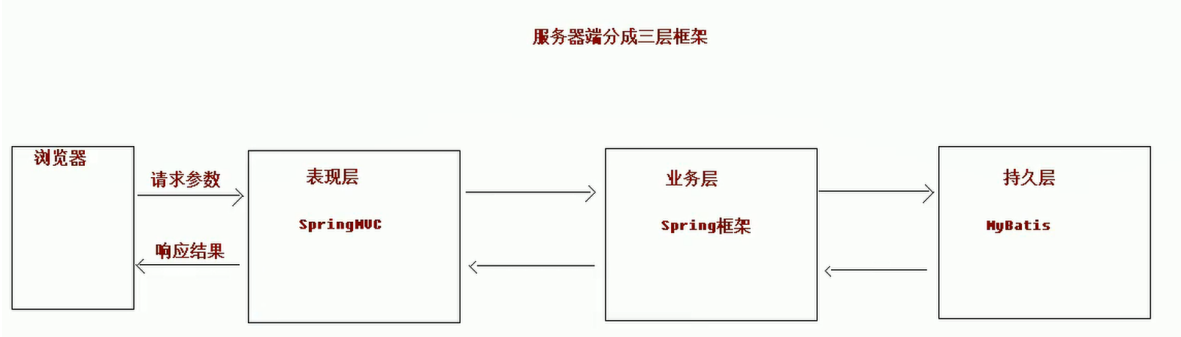
基本概念

1. 三层架构

1. 咱们开发服务器端程序，一般都基于两种形式，一种 C/S 架构程序，一种 B/S 架构程序
2. 使用 Java 语言基本上都是开发 B/S 架构的程序， B/S 架构又分成了三层架构
3. 三层架构
    1. 表现层： WEB 层，用来和客户端进行数据交互的。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

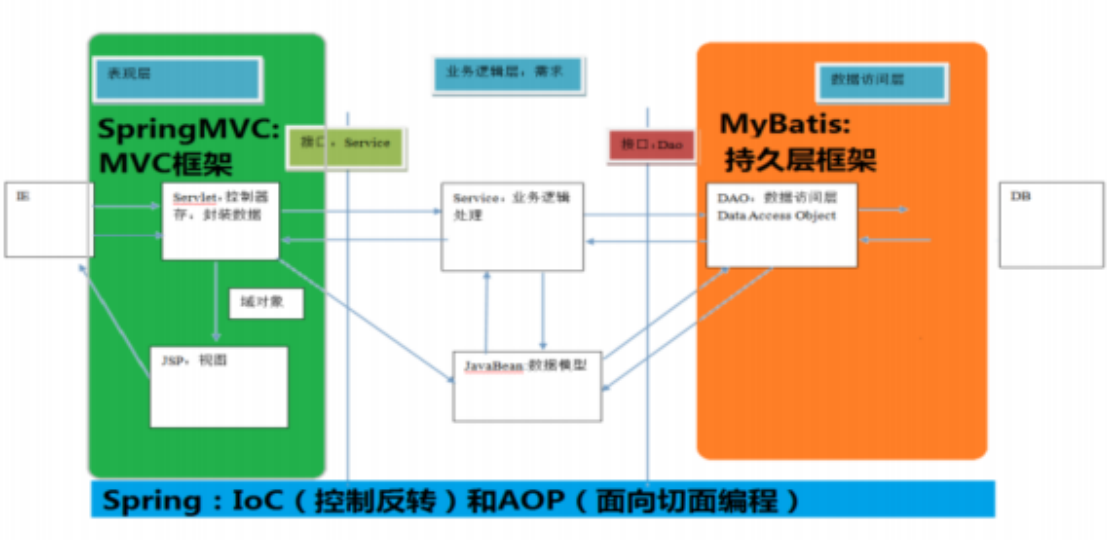


2. MVC模型

1. MVC 全名是 Model View Controller 模型视图控制器，每个部分各司其职。
2. Model ：数据模型， JavaBean 的类，用来进行数据封装。
3. View ：指 JSP 、 HTML 用来展示数据给用户
4. 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。

3. SpringMVC 的概述

1. SpringMVC 的概述
    1. 是一种基于 Java 实现的 MVC 设计模型的请求驱动类型的轻量级 WEB 框架。
    2. Spring MVC 属于 SpringFrameWork 的后续产品，已经融合在 Spring Web Flow 里面。 Spring 框架提供了构建 Web 应用程序的全功能 MVC 模块。
    3. 使用 Spring 可插入的 MVC 架构，从而在使用 Spring 进行 WEB 开发时，可以选择使用 Spring 的 SpringMVC 框架或集成其他 MVC 开发框架，如 Struts1 (现在一般不用)，Struts2 等。
2. SpringMVC 在三层架构中的位置
    - 表现层框架



3. `SpringMVC` 的优势

> 1、清晰的角色划分：
>
> 前端控制器（`DispatcherServlet`）
>
> 请求到处理器映射（`HandlerMapping`）
>
> 处理器适配器（`HandlerAdapter`）
>
> 视图解析器（`ViewResolver`）
>
> 处理器或页面控制器（`Controller`）
>
> 验证器（`Validator`）
>
> 命令对象（`Command` 请求参数绑定到的对象就叫命令对象）
>
> 表单对象（`Form Object` 提供给表单展示和提交到的对象就叫表单对象）。
>
> 2、分工明确，而且扩展点相当灵活，可以很容易扩展，虽然几乎不需要。
>
> 3、由于命令对象就是一个 `POJO`，无需继承框架特定 `API`，可以使用命令对象直接作为业务对象。
>
> 4、和 Spring 其他框架无缝集成，是其它 Web 框架所不具备的。
>
> 5、可适配，通过 `HandlerAdapter` 可以支持任意的类作为处理器。
>
> 6、可定制性，`HandlerMapping`、`ViewResolver` 等能够非常简单的定制。
>
> 7、功能强大的数据验证、格式化、绑定机制。
>
> 8、利用 `Spring` 提供的 `Mock` 对象能够非常简单的进行 Web 层单元测试。
>
> 9、本地化、主题的解析的支持，使我们更容易进行国际化和主题的切换。
>
> 10、强大的 `JSP` 标签库，使 `JSP` 编写更容易。
>
> ……还有比如RESTful风格的支持、简单的文件上传、约定大于配置的契约式编程支持、基于注解的零配
>
> 置支持等等。

4. `SpringMVC` 和 `Struts2` 框架的对比

> 共同点：
>
> - 它们都是表现层框架，都是基于 MVC 模型编写的。
> - 它们的底层都离不开原始 `ServletAPI`。

- 它们处理请求的机制都是一个核心控制器。

区别：

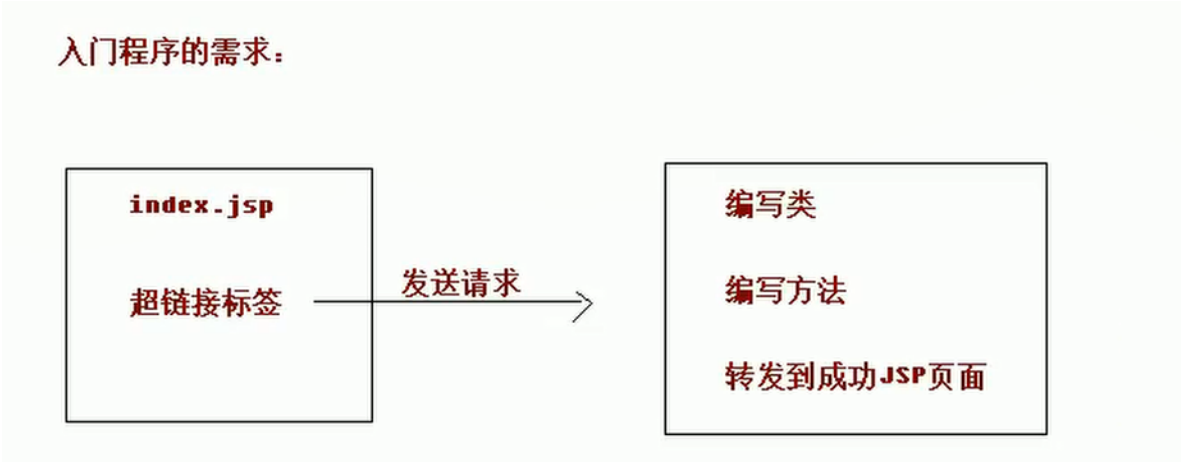
- `Spring MVC` 的入口是 `Servlet`，而 `Struts2` 是 `Filter`
- `Spring MVC` 是基于方法设计的，而 `Struts2` 是基于类，`Struts2` 每次执行都会创建一个动作类。所
- 以 `Spring MVC` 会稍微比 `Struts2` 快些。
- `Spring MVC` 使用更加简洁,同时还支持 `JSR303`，处理 `ajax` 的请求更方便
  - `JSR303` 是一套 `JavaBean` 参数校验的标准，它定义了很多常用的校验注解
  - 我们可以直接将这些注解加在我们 `JavaBean` 的属性上面，就可以在需要校验的时候进行校验了。
- `Struts2` 的 `OGNL` 表达式使页面的开发效率相比 `Spring MVC` 更高些，但执行效率并没有比 `JSTL` 提

升，尤其是 `struts2` 的表单标签，远没有 `html` 执行效率高。

入门

入门案例

- 入门案例的需求

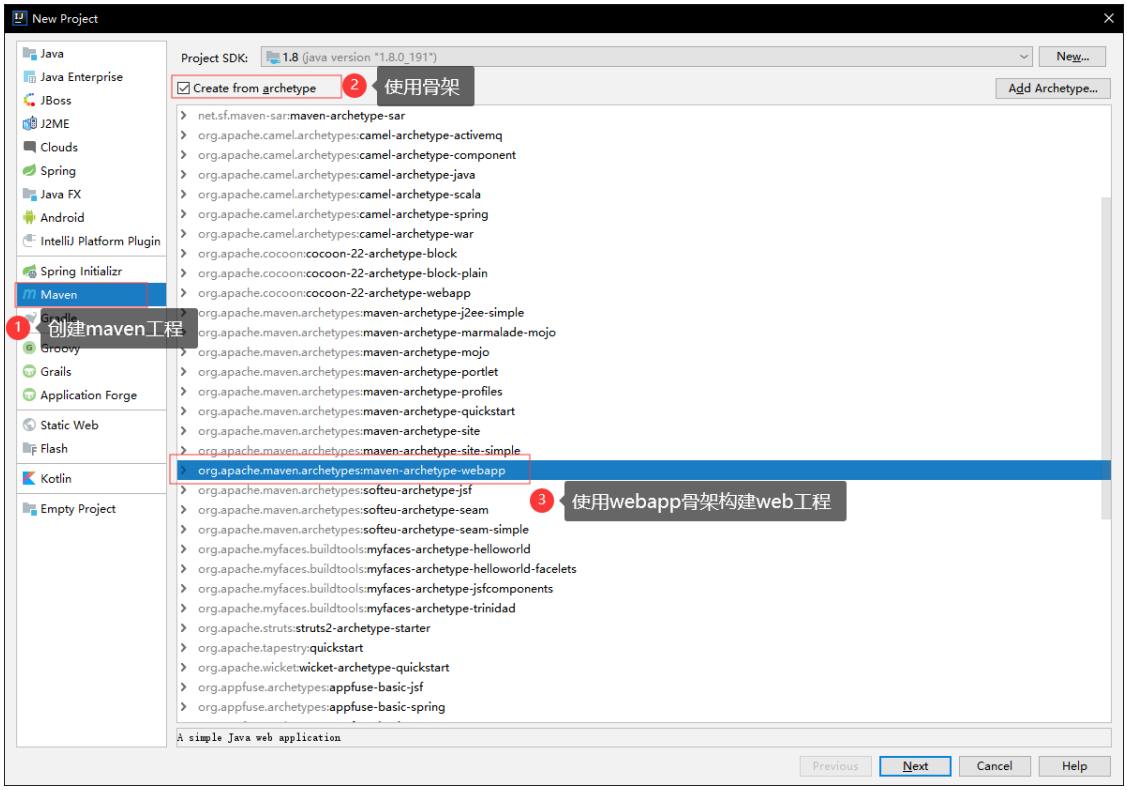


1. 搭建开发环境
2. 编写入门程序

搭建开发环境

创建项目

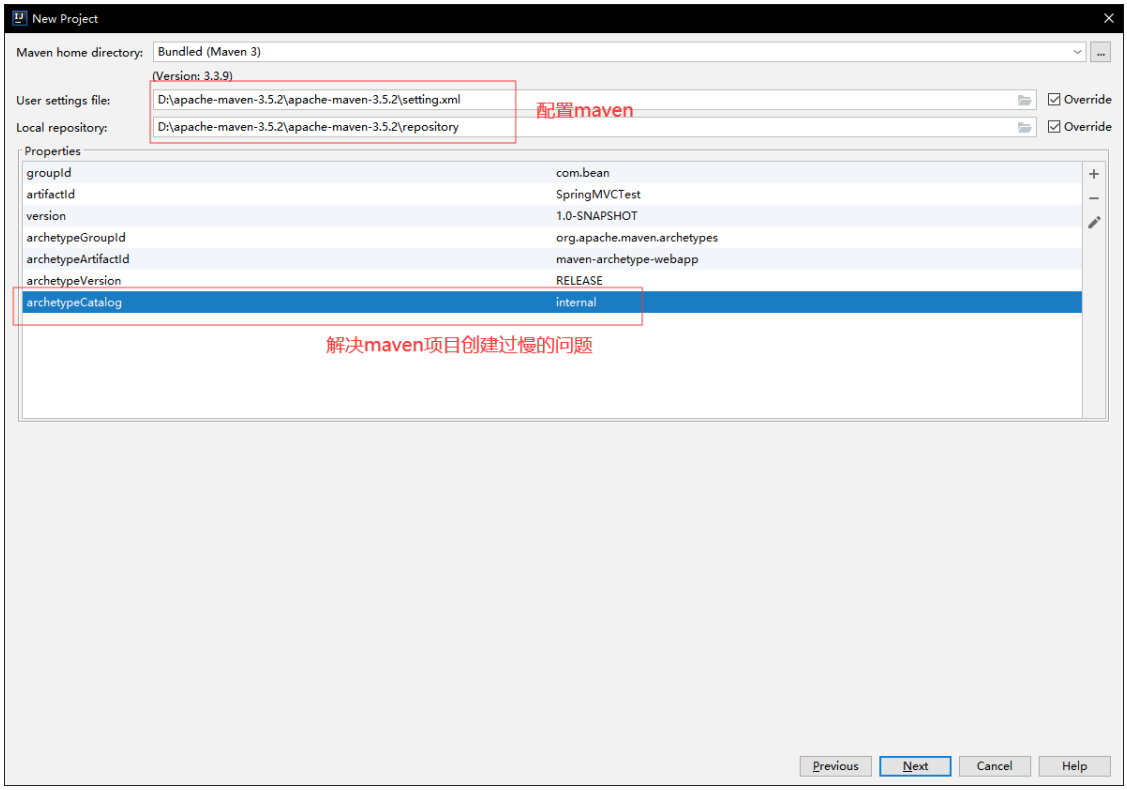
1. 使用骨架



2. 起名

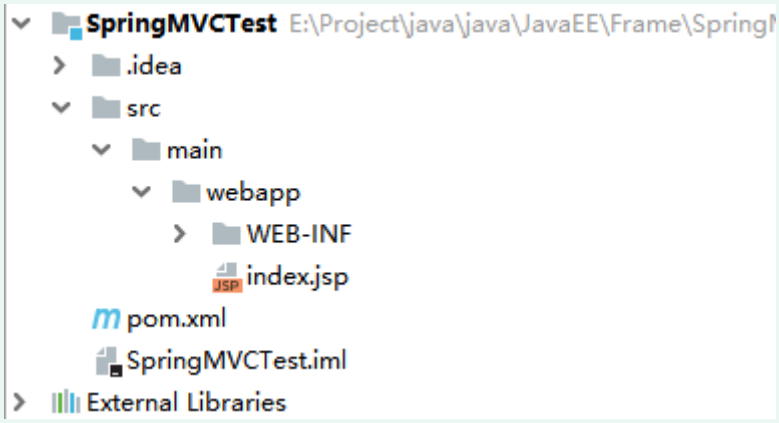
3. 配置 `maven` ，解决 `maven` 项目配置过慢的问题

> archetypeCatalog
>
> internal

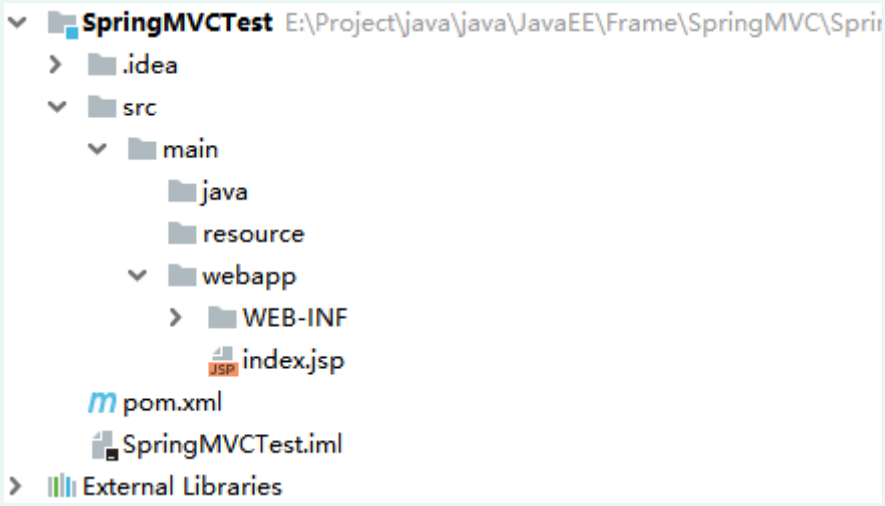


补全 maven 的目录结构

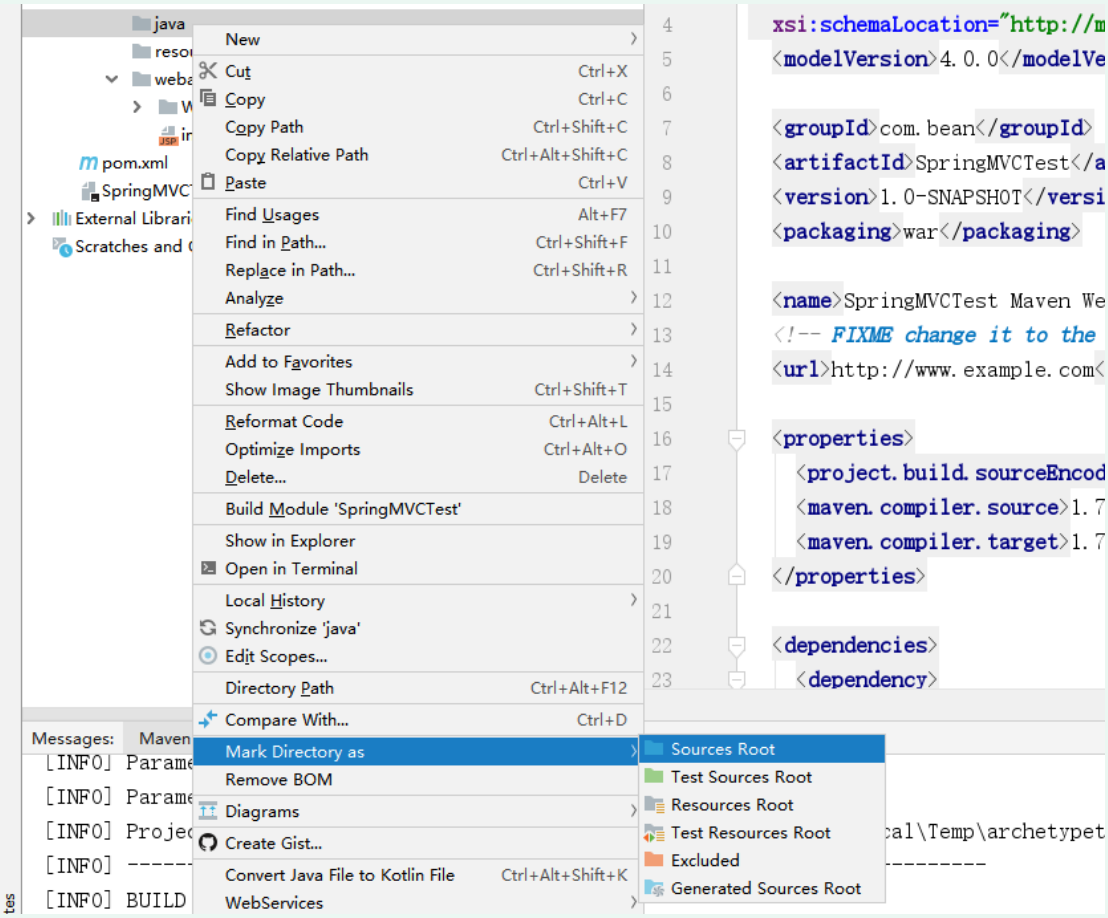
- 之前是这样的：



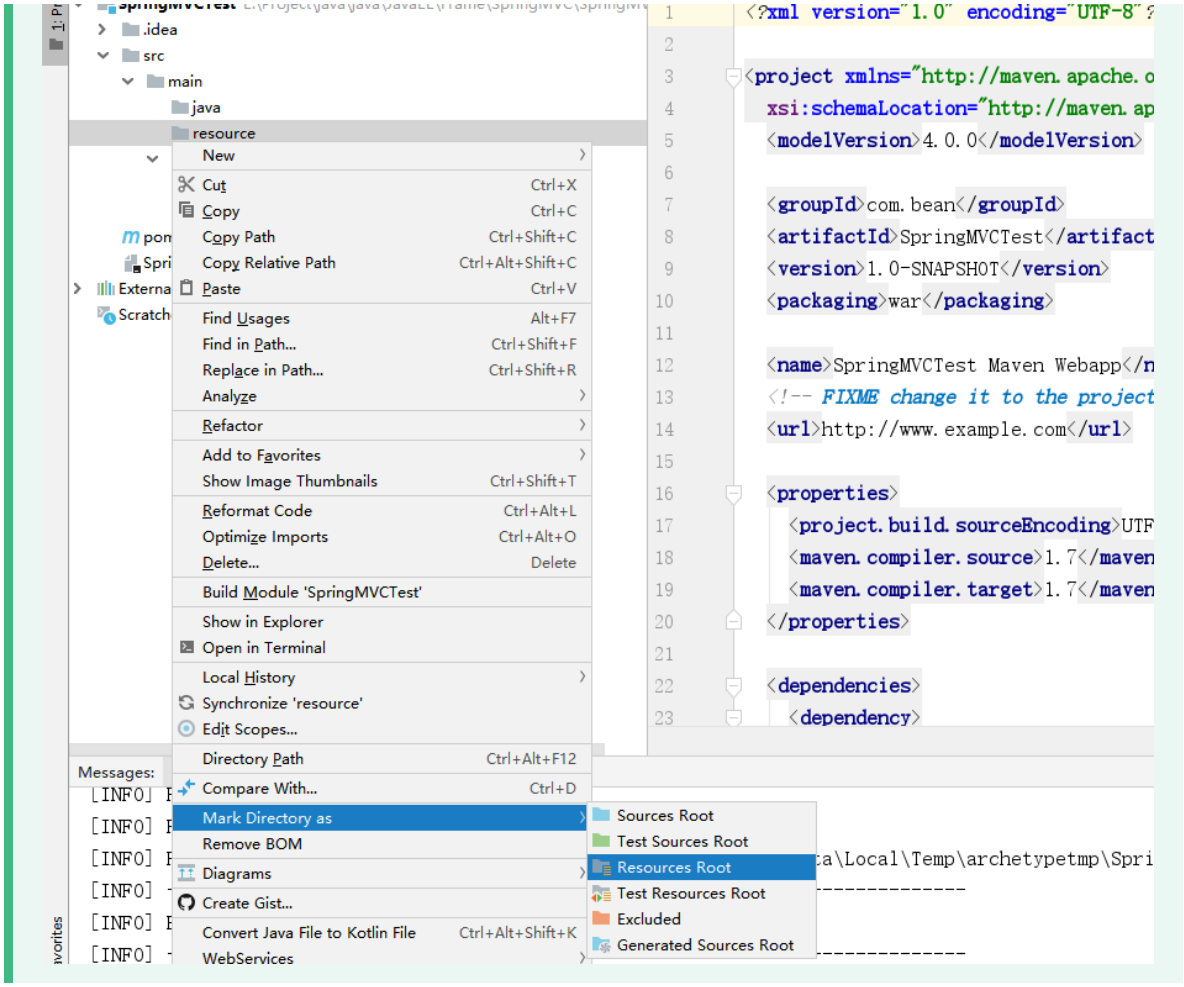
- 补全之后是这样的：



- 将 `java` 配置为源码文件夹



- 将 `resource` 配置为资源文件夹



导入 jar 包，导入依赖

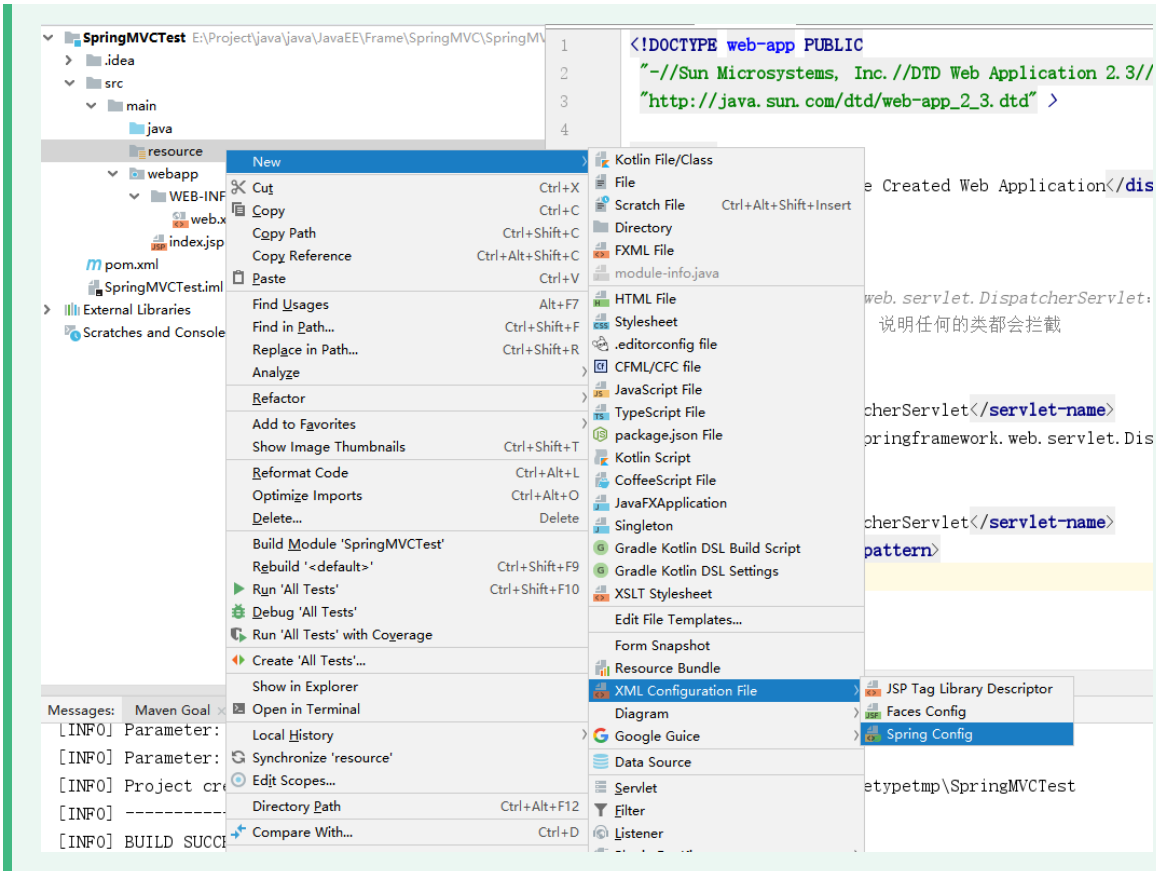
```
<!-- 版本锁定 -->
<properties>
    <spring.version>5.0.2.RELEASE</spring.version>
</properties>

<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>2.0</version>
        <scope>provided</scope>
    </dependency>
</dependencies>
```

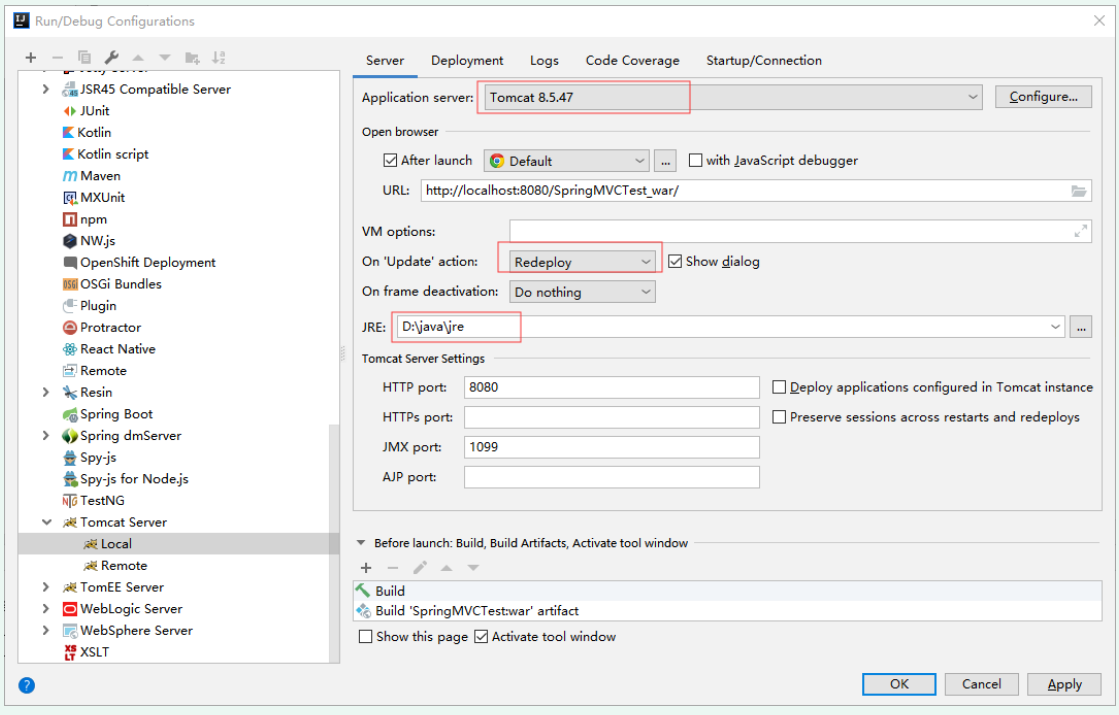
在 `web.xml` 中配置前端控制器

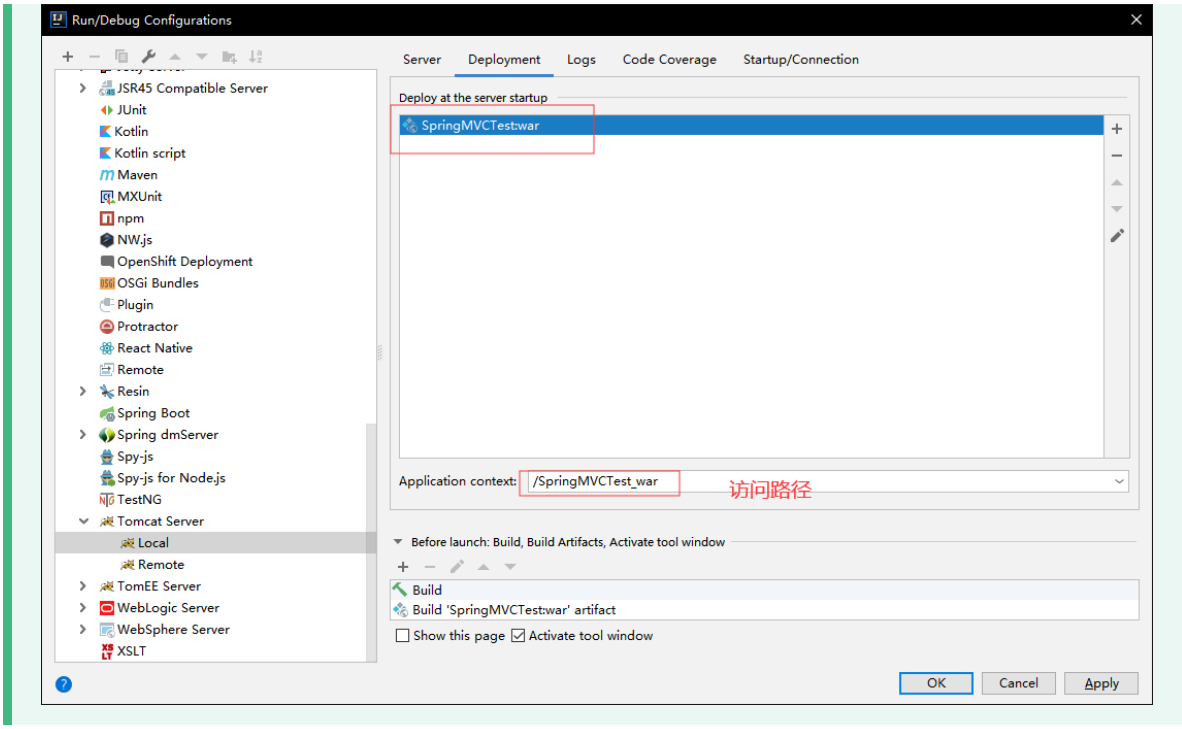
```
<!DOCTYPE web-app PUBLIC
 "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
 "http://java.sun.com/dtd/web-app_2_3.dtd" >

<web-app>
  <display-name>Archetype Created Web Application</display-name>


  <!--配置前端控制器
    org.springframework.web.servlet.DispatcherServlet：前端控制器，这个类是固定的
    url-pattern：值为"/"，说明任何的类都会拦截
  -->
  <servlet>
    <servlet-name>dispatcherServlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>dispatcherServlet</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>
</web-app>
```

在 `resource` 下创建配置文件 `springmvc.xml`


SpringMVCTest E:\Project\java\java\JavaEE\Frame\SpringMVC\SpringMV
.idea
src
main
java
resource
webapp
WEB-INF
web.x
index.jsp
pom.xml
SpringMVCTest.iml
External Libraries
Scratches and Console
<!DOCTYPE web-app PUBLIC
"-//Sun Microsystems, Inc.//DTD Web Application 2.3//
"http://java.sun.com/dtd/web-app_2_3.dtd" >
e Created Web Application</dis
web.servlet.DispatcherServlet:
说明任何的类都会拦截
cherServlet</servlet-name>
pringframework.web.servlet.Dis
cherServlet</servlet-name>
pattern>
etypetmp\SpringMVCTest
New
Cut Ctrl+X
Copy Ctrl+C
Copy Path Ctrl+Shift+C
Copy Reference Ctrl+Alt+Shift+C
Paste Ctrl+V
Find Usages Alt+F7
Find in Path... Ctrl+Shift+F
Replace in Path... Ctrl+Shift+R
Analyze
Refactor
Add to Favorites
Show Image Thumbnails Ctrl+Shift+T
Reformat Code Ctrl+Alt+L
Optimize Imports Ctrl+Alt+O
Delete... Delete
Build Module 'SpringMVCTest'
Rebuild '<default>' Ctrl+Shift+F9
Run 'All Tests' Ctrl+Shift+F10
Debug 'All Tests'
Run 'All Tests' with Coverage
Create 'All Tests'...
Show in Explorer
Open in Terminal
Local History
Synchronize 'resource'
Edit Scopes...
Directory Path Ctrl+Alt+F12
Compare With... Ctrl+D
Kotlin File/Class
File
Scratch File Ctrl+Alt+Shift+Insert
Directory
FXML File
module-info.java
HTML File
Stylesheet
.editorconfig file
CFML/CFC file
JavaScript File
TypeScript File
package.json File
Kotlin Script
CoffeeScript File
JavaFXApplication
Singleton
Gradle Kotlin DSL Build Script
Gradle Kotlin DSL Settings
XSLT Stylesheet
Edit File Templates...
Form Snapshot
Resource Bundle
XML Configuration File
Diagram
Google Guice
Data Source
Servlet
Filter
Listener
JSP Tag Library Descriptor
Faces Config
Spring Config
Messages: Maven Goal
[INFO] Parameter:
[INFO] Parameter:
[INFO] Project cr
[INFO] ----------
[INFO] BUILD SUCC

配置服务器

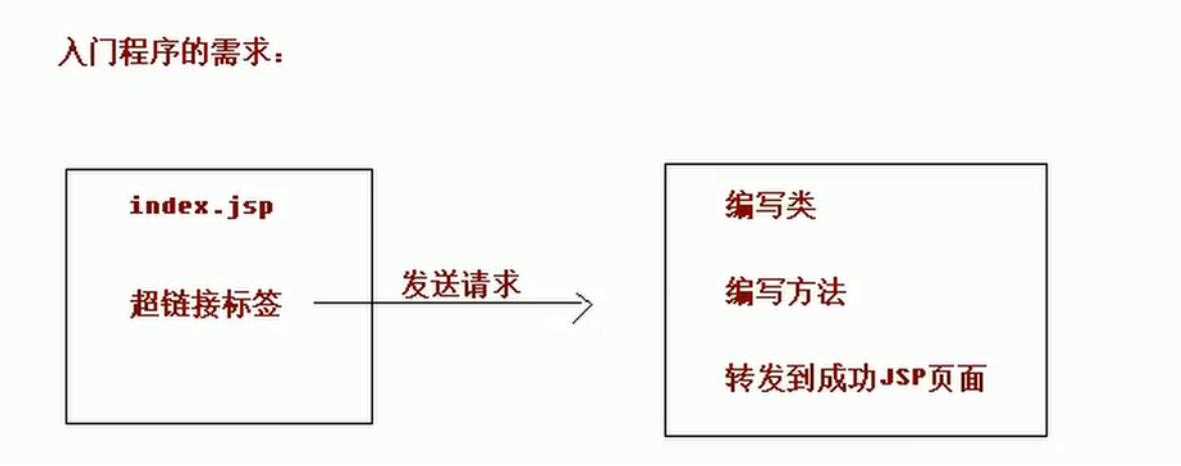

Run/Debug Configurations
JSR45 Compatible Server
JUnit
Kotlin
Kotlin script
Maven
MXUnit
npm
NW.js
OpenShift Deployment
OSGi Bundles
Plugin
Protractor
React Native
Remote
Resin
Spring Boot
Spring dmServer
Spy-js
Spy-js for Node.js
TestNG
Tomcat Server
Local
Remote
TomEE Server
WebLogic Server
WebSphere Server
XSLT
Server
Deployment
Logs
Code Coverage
Startup/Connection
Application server: Tomcat 8.5.47
Configure...
Open browser
After launch
Default
with JavaScript debugger
URL: http://localhost:8080/SpringMVCTest_war/
VM options:
On 'Update' action: Redeploy
Show dialog
On frame deactivation: Do nothing
JRE: D:\java\jre
Tomcat Server Settings
HTTP port: 8080
HTTPs port:
JMX port: 1099
AJP port:
Deploy applications configured in Tomcat instance
Preserve sessions across restarts and redeploys
Before launch: Build, Build Artifacts, Activate tool window
Build
Build 'SpringMVCTest:war' artifact
Show this page
Activate tool window
OK
Cancel
Apply



编写入门程序

需求

- 开发环境搭建完成之后，确认一件事：我们的需求



我们需要一个 `index.jsp` 页面，使用超链接发送请求，通过前端控制器找到类中的方法，获得返回的页面，然后跳转到该页面

过程

1. 编写开始界面 `index.jsp`
2. 编写 `web.xml` 配置
3. 编写 `springmvc.xml` 配置
4. 编写控制器类 `HelloController` 与方法 `sayHello()`
5. 编写跳转界面 `success.jsp`

编写

1. 编写开始界面 `jsp`

```jsp
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>
    <h3>SpringMVC入门程序</h3>

    <%--注意，这里配置的路径是hello，意思是点击这个超链接之后，跳转到"/hello"的路径--%>
    <a href="hello">入门程序</a>
</body>
</html>

```

2. 编写 `web.xml` 核心控制器

```xml
<!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>


  <!--配置前端控制器 org.springframework.web.servlet.DispatcherServlet：前端控制器，这个类是固定的-->
  <servlet>
    <servlet-name>dispatcherServlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

    <!--配置读取springmvc.xml这个配置文件-->
    <init-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>classpath:springmvc.xml</param-value>
    </init-param>

    <!-- 一般来说DispatcherServlet是在请求的时候才创建的，但是这里配置的是服务器启动的时候就要创建 -->
    <load-on-startup>1</load-on-startup>
  </servlet>

  <!--url-pattern：值为"/"，说明任何的类都会拦截-->
  <servlet-mapping>
    <servlet-name>dispatcherServlet</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>

</web-app>
```

3. 编写 `springmvc.xml`

```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">

       <!--这里开启注解扫描-->
       <context:component-scan base-package="com.bean"></context:component-scan>

       <!--配置视图解析器，只要有人告诉视图解析器我要找什么界面，就去找
           prefix：配置界面的路径，去这个路径下找界面
           suffix：配置后缀名，注意"."
       -->
       <bean id="internalResourceViewResolver"
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
           <property name="prefix" value="/WEB-INF/pages"></property>
           <property name="suffix" value=".jsp"></property>
       </bean>

       <!--开启SpringMVC框架注解的支持-->
       <mvc:annotation-driven/>
   </beans>
```

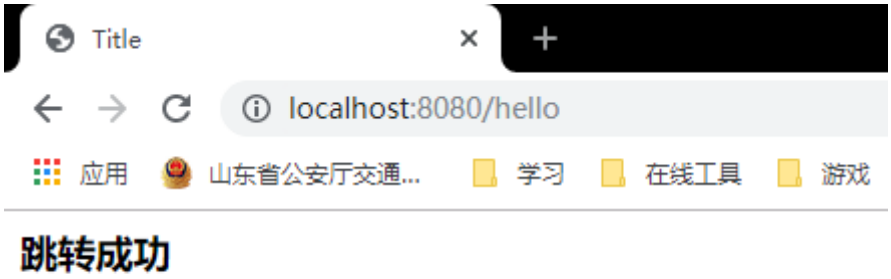
4. 配置控制器 `HelloController` 和方法

```
package com.bean.controller;


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

//这里是控制器类，@Controller指的是配置的控制器类
@Controller
public class HelloController {
    /*
    * 这里配置的是请求映射
    * 这个方法的映射就是路径"/hello"，也就是index.jsp超链接的访问路径
    * @RequestMapping属于SpringMVC注解
    * */
    @RequestMapping(path = "/hello")
    String sayHello(){
        System.out.println("Hello SpringMVC");
        return "/success";//返回值方法不是随便返回的，这个返回值说明告诉视图解析器，我要找
success这个界面，注意斜杠
    }
}
```

5. 配置最后的界面

```jsp
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>

    <h3>跳转成功</h3>
</body>
</html>
```

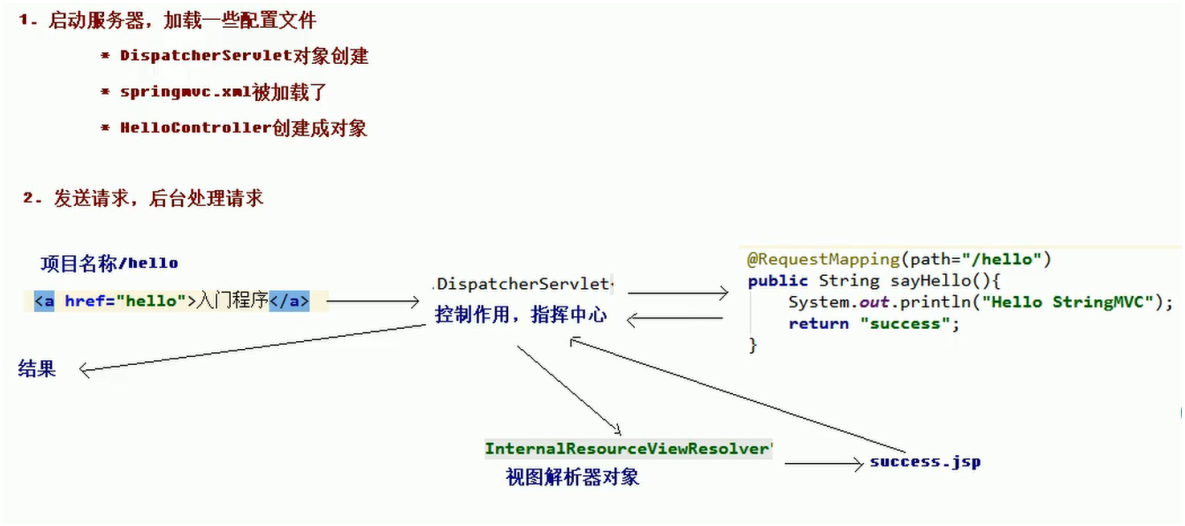
测试



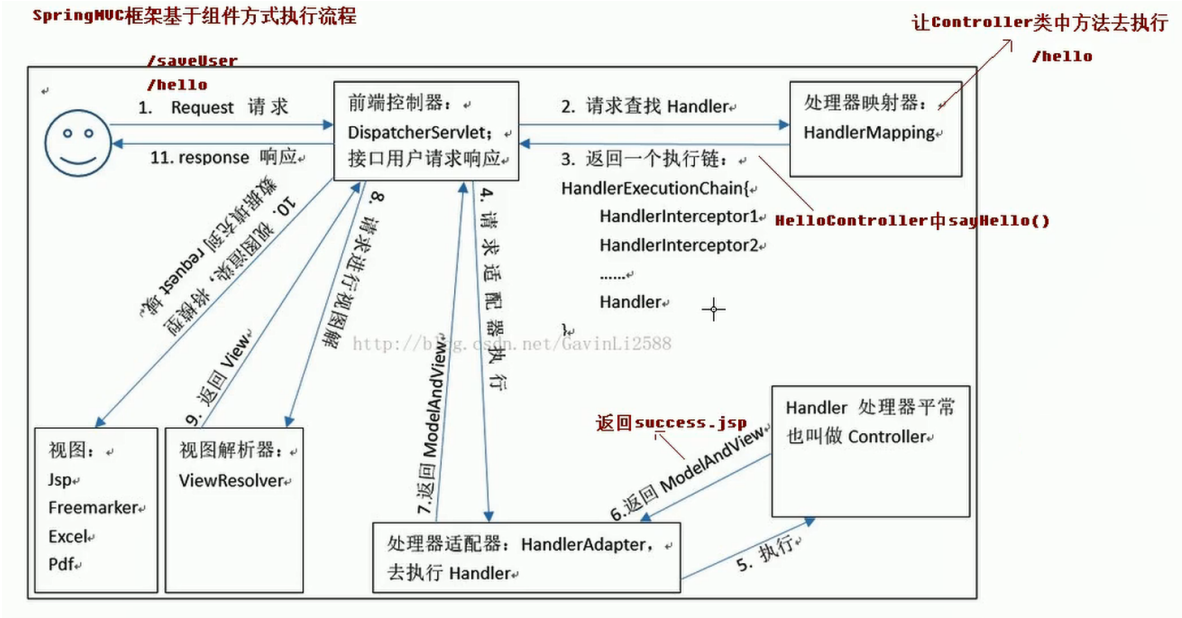
流程总结

1. 启动服务器，加载配置文件
    - 访问 `web.xml`
        - 因为配置，所以在启动服务器的时候创建前端控制器 `DispactherServlet`
        - 读取配置文件 `springmvc.xml`
    - 访问 `springmvc.xml`
        - 开启注解扫描
        - 配置了视图解析器
        - 开启了 `SpringMVC` 框架注解的支持
2. 发送请求，后台处理请求
    - 在 `index.jsp` 的超链接中访问 `/hello` 路径
    - 请求被 `web.xml` 中的 `servlet` 配置拦截，然后交由前端控制器 `DispactherServlet`
    - 根据访问路径调用方法 `sayHello()` ，返回了 `success`
    - 返回的 `success` 交给了前端控制器
    - 前端控制器交给 `springvc.xml` 中的视图解析器 `InternalResourceViewResolver`
    - 在视图解析器发现路径和后缀名称，发现文件
    - 视图解析器再交给前端控制器，由前端控制器返回结果

> 在上面，我们发现前端控制器可以说是起到了一个总控的作用



更加详细的流程



`<mvc:annotation-driven>`

处理器映射器，处理器适配器，视图解析器被称为 `SpringMVC` 的三大组件

- 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。

- 处理器适配器

  > 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。

- 视图解析器

> `View Resolver` 负责将处理结果生成 `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。

> 我们在之前配置 `springmvc.xml` 的时候配置了这个
>
> 这个不仅有支持 `SpringMVC` 注解的作用，也有让其自动配置
>
> - 处理器映射器
> - 处理器适配器
>
> 的作用

`RequestMapping` 注解

- 作用：建立请求 `URL` 与请求方法之间的关系
- 出现位置
  - 类上：声明一级目录
  - 方法上：声明二级目录，假如类上没有目录就是一级目录
- 属性
  - `path` ：路径
  - `value` ：别名是 `path`
  - `method` ：指定该方法的请求方式
    - `RequestMethod.GET`
    - `RequestMethod.POST`
    - `RequestMethod.PUT`
    - `RequestMethod.HEAD`
    - `RequestMethod.DELETE`
    - `RequestMethod.PATCH`
    - `RequestMethod.OPTIONS`
    - `RequestMethod.TRACE`
  - `params` ：指定限制请求参数的条件，比如必须要有 `usernmae` 才可以访问，甚至限制 `username=hehe` 等才可以访问
  - `headers` ：请求参数的请求头必须包含所指定的

例子

- 配置目录

```jsp
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>
    <h3>SpringMVC入门程序</h3>

    <%--注意，这里配置的路径是hello，意思是点击这个超链接之后，跳转到"/hello"的路径--%>
    <a href="test/hello">入门程序</a>
</body>
</html>
```

```java
package com.bean.controller;


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

@Controller
@RequestMapping(path = "/test") //一级目录
public class HelloController {
    @RequestMapping(path = "/hello")    //二级目录，访问时就是：http://localhost:8080/test/hello
    String sayHello(){
        System.out.println("Hello SpringMVC");
        return "/success";
    }

    @RequestMapping(path = "/mapping")//二级目录，访问时就是：http://localhost:8080/test/mapping
    String testMapping(){
        System.out.println("testMapping...");
        return "/success";
    }
}
```

- 配置请求方式

```jsp
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>
    <h3>SpringMVC入门程序</h3>

    <%--注意，这里配置的路径是hello，意思是点击这个超链接之后，跳转到"/hello"的路径--%>
    <a href="test/hello">入门程序</a>
</body>
</html>
```

```java
package com.bean.controller;

```

```
3
4   import org.springframework.stereotype.Controller;
5   import org.springframework.web.bind.annotation.RequestMapping;
6   import org.springframework.web.bind.annotation.RequestMethod;
7
8   @Controller
9   public class HelloController {
10      @RequestMapping(path = "/hello",method = RequestMethod.GET)//配置访问方式必须为GET
11      String sayHello(){
12          System.out.println("Hello SpringMVC");
13          return "/success";
14      }
15  }
```

- 配置请求参数

```
1   <%@ page contentType="text/html;charset=UTF-8" language="java" %>
2   <html>
3   <head>
4       <title>Title</title>
5   </head>
6   <body>
7       <h3>SpringMVC入门程序</h3>
8
9       <%--注意，这里配置的路径是hello，意思是点击这个超链接之后，跳转到"/hello"的路径--%>
10      <a href="test/requestMapping">入门程序</a>
11
12      <a href="test/requestMapping?username=haha">测试是否有所规定的username</a>
13
14      <%-- 不是hehe，凉了 --%>
15      <a href="test/requestMappingTest?username=heihei">测试是否有所规定的username，且username是否
    为hehe</a>
16
17
18  </body>
19  </html>
```

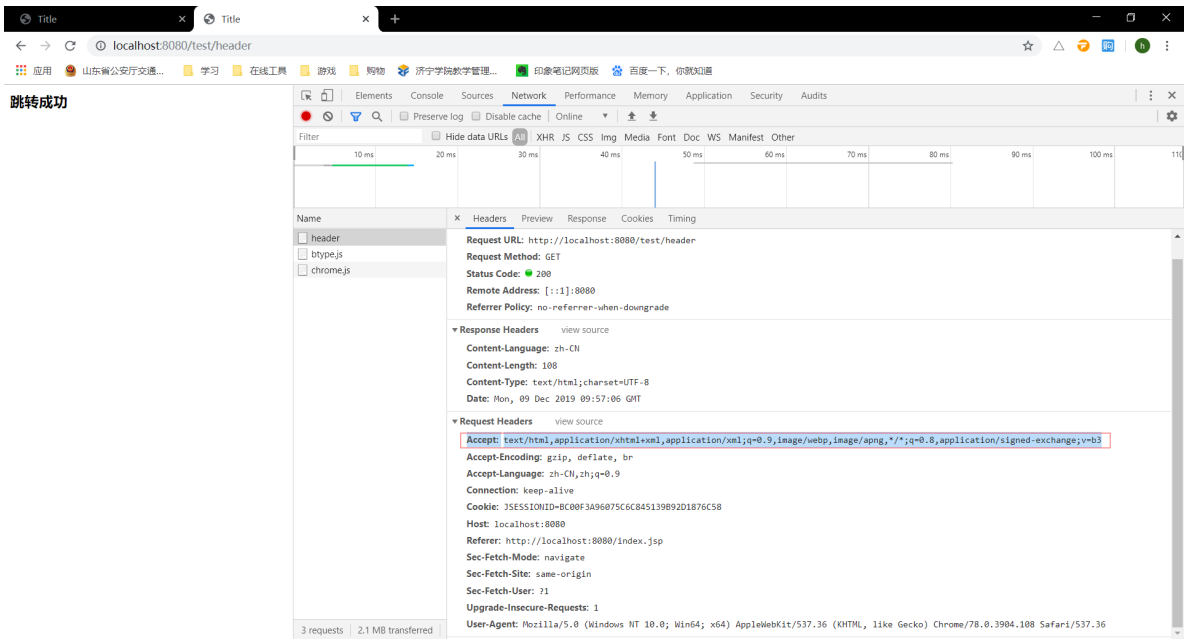
```
1   package com.bean.controller;
2
3
4   import org.springframework.stereotype.Controller;
5   import org.springframework.web.bind.annotation.RequestMapping;
6   import org.springframework.web.bind.annotation.RequestMethod;
7
8   @Controller
9   @RequestMapping(path = "/test")
10  public class HelloController {
11      @RequestMapping(path = "/hello")
12      String sayHello(){
13          System.out.println("Hello SpringMVC");
14          return "/success";
15      }
16
17      @RequestMapping(path = "/requestMapping",params = "username")//配置请求参数中必须有username
18      String requestMapping(){
19          return "/success";
20      }
21
```

```
22      @RequestMapping(path = "/requestMappingTest",params = "username=hehe")//配置请求参数中必须有username，且username必须为hehe
23      String requestMappingTest(){
24          return "/success";
25      }
26  }
```

- 是否有所规定的请求头

```
1   <%@ page contentType="text/html;charset=UTF-8" language="java" %>
2   <html>
3   <head>
4       <title>Title</title>
5   </head>
6   <body>
7       <h3>SpringMVC入门程序</h3>
8
9       <a href="test/hello">入门程序</a>
10
11      <a href="test/header">检查是否有请求头</a>
12  </body>
13  </html>
```

```
1   package com.bean.controller;
2
3
4   import org.springframework.stereotype.Controller;
5   import org.springframework.web.bind.annotation.RequestMapping;
6   import org.springframework.web.bind.annotation.RequestMethod;
7
8   @Controller
9   @RequestMapping(path = "/test")
10  public class HelloController {
11  //    没有Application，凉了
12      @RequestMapping(path = "/hello",headers = "Application")
13      String sayHello(){
14          return "/success";
15      }
16
17      @RequestMapping(path = "/header",headers = "Accept")
18      String requestHeaders(){
19          return "/success";
20      }
21  }
```



请求参数的绑定

请求参数绑定说明

- 绑定机制

> 表单提交的数据都是 `key=value` 形式的，且都是字符串，比如：`username=hehe&password=123`

> `springMVC` 的参数绑定过程是将表单提交的请求参数作为控制器进行绑定的，底层使用反射的方式拿到值并且赋值给参数

要求：提交表单的 `name` 属性与参数的名称是相同的

基本数据类型和字符串

- 只要方法中参数名字和表单中的值相同即可
- 例如

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>
    <h3>SpringMVC入门程序</h3>

    <form action="test/hello">
        用户名：<input type="text" name="username"> <%--注意这里，username和后面参数的username相同--%>
        密码：<input type="text" name="password">  <%--注意这里，password和后面的password相同，--%>
        <input type="submit" value="提交">
    </form>
```

```
14
15  </body>
16  </html>
```

```
1   package com.bean.controller;
2
3
4   import org.springframework.stereotype.Controller;
5   import org.springframework.web.bind.annotation.RequestMapping;
6   import org.springframework.web.bind.annotation.RequestMethod;
7
8   @Controller
9   @RequestMapping(path = "/test")
10  public class HelloController {
11      //使用参数绑定获得值
12      @RequestMapping(path = "/hello")
13      String sayHello(String username,Integer password){//这里的username和password和表单的name属
    性名称相同，所以才可以获取值
14          System.out.println(username);
15          System.out.println(password);
16          return "/success";
17      }
18
19  }
```

JAVABEAN 实体

简单实体

- 表单中的 `name` 属性要写为： `username` 的形式， `username` 和 `domain` 中的 `username` 相同
- 参数列表中直接写 `domain` 的形式

```
1   package com.bean.domain;
2
3   import java.util.Date;
4
5   public class User {
6
7       private String username;
8       private String password;
9       private Date birthday;
10
11      public User() {
12      }
13
14      public User(String username, String password, Date birthday) {
15          this.username = username;
16          this.password = password;
17          this.birthday = birthday;
18      }
19
20      public String getUsername() {
21          return username;
22      }
23
24      public void setUsername(String username) {
25          this.username = username;
```

```
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public Date getBirthday() {
        return birthday;
    }

    public void setBirthday(Date birthday) {
        this.birthday = birthday;
    }

    @Override
    public String toString() {
        return "User{" +
                "username='" + username + '\'' +
                ", password='" + password + '\'' +
                ", birthday=" + birthday +
                '}';
    }
}
```

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>
    <h3>SpringMVC入门程序</h3>

    <form action="test/hello">
        用户名：<input type="text" name="username">
        密码：<input type="text" name="password">
        日期：<input type="text" name="birthday">
        <%--这里的格式先写为yyyy/MM/dd的形式，因为这里涉及到一个类型转换的问题--%>

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

</body>
</html>
```

```
package com.bean.controller;


import com.bean.domain.User;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

@Controller
```

```java
@RequestMapping(path = "/test")
public class HelloController {
    //使用参数绑定获得值
    @RequestMapping(path = "/hello")
    String sayHello(User user){//这里的user和前面表单上的是一个值
        System.out.println(user.getUsername());
        System.out.println(user.getPassword());
        System.out.println(user.getBirthday());
        return "/success";
    }
}
```

带有其他 JAVABEAN 的实体

```java
package com.bean.domain;

public class Address {
    private String address;

    public Address() {
    }

    public Address(String address) {
        this.address = address;
    }

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    @Override
    public String toString() {
        return "Address{" +
                "address='" + address + '\'' +
                '}';
    }
}
```

```java
package com.bean.domain;

import java.util.Date;

public class User {

    private String username;
    private String password;
    private Date birthday;
    private Address address;
    public User() {
    }

    public User(String username, String password, Date birthday, Address address) {
```

```
        this.username = username;
        this.password = password;
        this.birthday = birthday;
        this.address = address;
    }

    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 Date getBirthday() {
        return birthday;
    }

    public void setBirthday(Date birthday) {
        this.birthday = birthday;
    }

    public Address getAddress() {
        return address;
    }

    public void setAddress(Address address) {
        this.address = address;
    }

    @Override
    public String toString() {
        return "User{" +
                "username='" + username + '\'' +
                ", password='" + password + '\'' +
                ", birthday=" + birthday +
                ", address=" + address +
                '}';
    }
}
```

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>
    <h3>SpringMVC入门程序</h3>

```

```
    <form action="test/hello">
        用户名：<input type="text" name="username">
        密码：<input type="text" name="password">
        日期：<input type="text" name="birthday">
        <%--这里的格式先写为yyyy/MM/dd的形式，因为这里涉及到一个类型转换的问题--%>

        其他实体（Address）<input type="text" name="address.address">
        <%--这里注意，name中的前一个address指的是address实体类，但是后面的是实体类中的参数，是将address封装成了一个address对象--%>

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


</body>
</html>
```

```
package com.bean.controller;


import com.bean.domain.Address;
import com.bean.domain.User;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

@Controller
@RequestMapping(path = "/test")
public class HelloController {
    //使用参数绑定获得值
    @RequestMapping(path = "/hello")
    String sayHello(User user){//这里的user和前面表单上的是一个值
        System.out.println(user.getUsername());
        System.out.println(user.getPassword());
        System.out.println(user.getBirthday());
        System.out.println(user.getAddress());
        return "/success";
    }
}
```

集合

```
package com.bean.domain;

public class Address {
    private String address;

    public Address() {
    }

    public Address(String address) {
        this.address = address;
    }

    public String getAddress() {
        return address;
```

```
    }

    public void setAddress(String address) {
        this.address = address;
    }

    @Override
    public String toString() {
        return "Address{" +
                "address='" + address + '\'' +
                '}';
    }
}
```

```
package com.bean.domain;

import java.util.Date;
import java.util.List;
import java.util.Map;

public class User {

    private String username;

    private List<Address> addresseList;

    private Map<String,Address> addressMap;

    public User() {
    }

    public User(String username, List<Address> addresseList, Map<String, Address>
addressMap) {
        this.username = username;
        this.addresseList = addresseList;
        this.addressMap = addressMap;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public List<Address> getAddresseList() {
        return addresseList;
    }

    public void setAddresseList(List<Address> addresseList) {
        this.addresseList = addresseList;
    }

    public Map<String, Address> getAddressMap() {
        return addressMap;
```

```
    }

    public void setAddressMap(Map<String, Address> addressMap) {
        this.addressMap = addressMap;
    }

    @Override
    public String toString() {
        return "User{" +
                "username='" + username + '\'' +
                ", addresseList=" + addresseList +
                ", addressMap=" + addressMap +
                '}';
    }
}
```

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>
    <h3>SpringMVC入门程序</h3>

    <form action="test/hello">
        用户名：<input type="text" name="username">
        List：<input type="text" name="addresseList[0].address">
        <%--这里直接将address封装进了addressList，这里的addressList和实体类中的是一个--%>

        Map：<input type="text" name="addressMap['one'].address">
        <%--这里直接将address封装进了addressMap，这里的addressMap和实体类中的是一个--%>

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


</body>
</html>
```

```
package com.bean.controller;


import com.bean.domain.Address;
import com.bean.domain.User;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
        import org.springframework.web.bind.annotation.RequestMethod;

@Controller
@RequestMapping(path = "/test")
public class HelloController {
    //使用参数绑定获得值
    @RequestMapping(path = "/hello")
    String sayHello(User user){//这里的user和前面表单上的是一个值
        System.out.println(user.getUsername());
        System.out.println("----------------------------");
        System.out.println(user.getAddresseList());
```

```
        System.out.println("----------------------------");
        System.out.println(user.getAddressMap());
        return "/success";
    }
}
```

```
wangwu
----------------------------
[Address{address='wangwu'}]
----------------------------
{one=Address{address='wangwu'}}
```

中文乱码的解决方案

> 在书写中会存在中文乱码的情况，`GET`请求不会出现，但是`POST`请求会出现这种情况，所以我们应该配置一个过滤器来解决这种情况

- 在`web.xml`中配置一个过滤

```
<!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>

  <!--配置过滤器 filter-class：这个类是spring提供的类 init-param：配置编码，设置为UTF-8-->
  <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>
  <!--配置拦截，拦截所有请求-->
  <filter-mapping>
    <filter-name>characterEncodingFilter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>

  <!--配置前端控制器 org.springframework.web.servlet.DispatcherServlet：前端控制器，这个类是固定的-->
  <servlet>
    <servlet-name>dispatcherServlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

    <!--配置读取springmvc.xml这个配置文件-->
    <init-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>classpath:springmvc.xml</param-value>
    </init-param>

    <!-- 一般来说DispatcherServlet是在请求的时候才创建的，但是这里配置的是服务器启动的时候就要创建 -->
    <load-on-startup>1</load-on-startup>
```

```
  </servlet>

  <!--url-pattern：值为"/"，说明任何的类都会拦截-->
  <servlet-mapping>
    <servlet-name>dispatcherServlet</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>

</web-app>
```

类型转换与自定义类型转换器

> 首先一定要了解到，前端给后端数据的时候给的都是字符串，是由 `spring` 帮我们把这些字符串转为的各种类型

> 在之前讲解 `JAVABEAN` 实体绑定的时候曾经使用过 `Date` 数据类型，当时说明的是日期要以 `yyyy/MM/dd` 的形式写，这是因为 `spring` 底层在帮我们做类型转换的时候就是以这种形式转换的

> 我们不想以这种方式进行转换，而想用一个我们比较熟悉的格式： `yyyy-MM-dd` 的形式做一个类型转换

> `spring` 不提供这种格式，我们自己写一个

> 注意：这里只是以 `Date` 为例做一个转换器，当我们发现其他类型也不好使的时候，应该自己会写

1. 自定义一个类，这个类要实现 `Converter<S,T>` 接口，该接口有两个泛型
   - `S` ：表示接收到的，我们这里当然就是字符串了
   - `T` ：表示要转换的类型
2. 在 `Spring` 的配置文件中配置类型转换器
3. 在 `annotation-driven` 标签中引用配置的类型转换服务

- `util`

```
package com.bean.utils;

import org.springframework.core.convert.converter.Converter;
import org.springframework.util.StringUtils;

import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;

public class StringToDateConverter implements Converter<String, Date> {//注意别导错包
    @Override
    public Date convert(String s) {
        SimpleDateFormat simpleDateFormat;
```

```
        try{
            if (StringUtils.isEmpty(s)){
                throw new NullPointerException("请输入要转换的日期");
            }
            simpleDateFormat = new SimpleDateFormat("yyyy-MM-dd");
            Date date = simpleDateFormat.parse(s);
            return date;
        } catch (ParseException e) {
            throw new RuntimeException("转换异常");
        }
    }
}
```

- 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">

    <!--这里开启注解扫描-->
    <context:component-scan base-package="com.bean"></context:component-scan>

    <!--配置视图解析器-->
    <bean id="internalResourceViewResolver"
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/pages"></property>
        <property name="suffix" value=".jsp"></property>
    </bean>


    <!--配置类型转换器工厂，通过源码我们可以看到这是一个类型转换器的工厂-->
    <bean id="conversionServiceFactoryBean"
class="org.springframework.context.support.ConversionServiceFactoryBean">
        <!--配置一个类型转化器，通过源码我们可以看到这里需要一个set，那么就配置一个set，指向我们的方法-->
        <property name="converters">
            <set>
                <!--配置自定义类型转换器-->
                <bean class="com.bean.utils.StringToDateConverter"></bean>
            </set>
        </property>
    </bean>

    <!--开启SpringMVC框架注解的支持，我们知道，这个注解已经默认配置了处理器映射器，处理器配置器，
    但是在spring中的类型转换器工厂中并没有我们想要的，所以我们自己加一个-->
    <mvc:annotation-driven conversion-service="conversionServiceFactoryBean"/>
</beans>
```

- jsp

```jsp
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>
    <h3>SpringMVC入门程序</h3>

    <form action="test/hello" method="post">
        Date:<input type="date" name="date"/>
        <input type="submit" value="提交">
    </form>


</body>
</html>
```

```java
package com.bean.controller;


import com.bean.domain.Address;
import com.bean.domain.User;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
        import org.springframework.web.bind.annotation.RequestMethod;

import java.util.Date;

@Controller
@RequestMapping(path = "/test")
public class HelloController {
    //使用参数绑定获得值
    @RequestMapping(path = "/hello")
    String sayHello(Date date){//这里的user和前面表单上的是一个值
        System.out.println(date.toString());
        return "/success";
    }
}
```

常用注解

@RequestParam

作用：

- 把请求中指定名称的参数给控制器中的形参赋值。

属性：

- 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。

```
package com.bean.controller;


import com.bean.domain.Address;
import com.bean.domain.User;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
        import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;

import java.util.Date;

@Controller
@RequestMapping(path = "/test")
public class HelloController {

    @RequestMapping(path = "/hello")
    String sayHello(
            @RequestParam(value = "time",required = true) Date date,
            @RequestParam(value = "username",required = false) String username
    ){
        /*
        * 这里注意了，既然value中指定的就是time，那么在jsp中就不可以写date了，而是写time，然后time的值将会赋给date
        * 然后后面的username当然就是username，required指定了fase，说明不写也不会报错
        * */

        return "/success";
    }
}
```

@RequestBody

作用：

- 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。
- 必须是 `post`，get 请求方式不适用

属性：

- required：是否必须有请求体。默认值是:true。当取值为 true 时，get 请求方式会报错。如果取值为 false，get 请求得到是 null。

> 这里获取请求体的内容的时候，在控制器中的参数列表里直接写一个 `String body` 就行了，获取的内容是 `key=value&key=value` 的形式

```
package com.bean.controller;


import com.bean.domain.Address;
import com.bean.domain.User;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
        import org.springframework.web.bind.annotation.RequestMethod;
```

```java
import org.springframework.web.bind.annotation.RequestParam;

import java.util.Date;

@Controller
@RequestMapping(path = "/test")
public class HelloController {

    @RequestMapping(path = "/hello")
    String sayHello(@RequestBody String body){
        /*
        * 获取请求体内容，get没有请求体，不适用
        *   required
        *       true：请求值必须有请求体
        *       false：请求可以没有请求体
        * */
        System.out.println(body);

        String[] split = body.split("&");
        for (String s : split) {
            System.out.println(s);
        }

        return "/success";
    }
}
```

```jsp
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>
    <h3>SpringMVC入门程序</h3>

    <form action="test/hello" method="post">

        Date:<input type="date" name="time"/>
        username:<input type="text" name="username"/>

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

    </form>


</body>
</html>
```

```
time=2019-12-11&username=zhangsan
time=2019-12-11          这里直接就是字符串形式的
username=zhangsan
```

@PathVaribale

作用：

- 用于绑定 `url` 中的占位符。

  > 例如：请求 `url` 中 `/delete/{id}`，这个{id}就是 `url` 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。

> `Rest` 风格 `URL`
>
> 简单来说就是使用同一个地址和不同的网络访问进行不同的操作
>
> 例子：
>
> > /account/1　HTTP **GET**　　得到id=1的account
> >
> > /account/1　HTTP **DELETE**　删除id=1的account
> >
> > /account/1　HTTP **PUT**　　更新id=1的account

> 为什么要讲 `REST` 风格呢，因为我们使用 `@PathVaribale` 和 `@RequestMapping` 可以自己形成这种风格

```java
package com.bean.controller;


import com.bean.domain.Address;
import com.bean.domain.User;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.*;

import java.util.Date;

@Controller
@RequestMapping(path = "/test")
public class HelloController {

    /*因为浏览器只能模拟get和post两种请求，所以只写这两种*/

    /*这里是访问的find方法，通过POST请求*/
    @RequestMapping(value = "/find",method = RequestMethod.POST)
    String find(){
        System.out.println("POST方式查询所有用户");
        return "success";
    }

    @RequestMapping(value = "/find",method = RequestMethod.GET)
    String findAll(){
        System.out.println("GET方式查询所有用户");
```

```
        return "success";
    }

    /*访问的findById方法，通过占位符来表示查询的哪个id，uid就是绑定了参数列表中的uid*/
    @RequestMapping(value = "/find/{uid}",method = RequestMethod.GET)
    String findByID(@PathVariable(value = "uid") Integer id){
        System.out.println("根据id查询用户");
        return "seccess";
    }

    /*访问的findById方法，通过占位符来表示查询的哪个id，uid就是绑定了参数列表中的uid*/
    @RequestMapping(value = "/find/{uid}",method = RequestMethod.POST)
    String findByUID(@PathVariable(value = "uid") Integer id){
        System.out.println("根据id查询用户");
        return "seccess";
    }



}
```

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>
    <h3>SpringMVC入门程序</h3>

    <%--使用get方式来查询所有用户--%>
    <a name="date" href="test/find">sayHelloGet</a>

    <%--使用post方法来访问所有用户--%>
    <form action="test/find" method="post"></form>

    <%--使用GET方式来查询特定用户，注意这种带有占位符的是直接/xxx，而不是?xxx=xxx的形式了，因为这是一个路径，不是参数--%>
    <a href="test/find/100"></a>

    <%--同样的，使用post来标定占位符的时候，也是要手动添加/xxx，因为这是路径不是参数--%>
    <form action="test/find/100"></form>

</body>
</html>
```

@RequestHeader

作用：

- 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

注：

- 在实际开发中一般不怎么用。

```
<a href="springmvc/useRequestHeader">获取请求消息头</a>
```

```
@RequestMapping("/useRequestHeader")
public String useRequestHeader(@RequestHeader(value="Accept-Language",required=false)String
requestHeader){
    System.out.println(requestHeader);
    return "success";
}
```

@CookieValue

作用：

- 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。

```
package com.bean.controller;


import com.bean.domain.Address;
import com.bean.domain.User;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.*;

import java.util.Date;

@Controller
@RequestMapping(path = "/test")
public class HelloController {

    /*把名称为JSESSIONID的cookie值传给cookieValue*/
    @RequestMapping(value = "/hello")
    String sayHello(@CookieValue(value = "JSESSIONID",required = false) String cookieValue){
        System.out.println(cookieValue);
        return "/success";
    }
}
```

@ModelAttribute

作用：

- 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。

应用场景：

- 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，一旦更新会把该字段内容置为 null，此时就可以使用此注解解决问题。

> 因为被 `@ModleAttribute` 修饰的方法会先执行，所以首先解决一些问题，比如用户传过来的值不够完整，或者解决乱码错误等

> 现在我们的情景就是从表单提交过来的不完整，一共有数据：`username,password,date。`
>
> 但是只提交了两个数据 `username` 和 `password`
>
> 现在我们做的工作就是把没有提交的数据默认就是数据库中原来的数据

方法一：有返回值的

```java
package com.bean.domain;

import java.util.Date;
import java.util.List;
import java.util.Map;

public class User {

    private String username;

    private String password;

    private Date date;

    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 Date getDate() {
        return date;
    }

    public void setDate(Date date) {
        this.date = date;
    }

    @Override
    public String toString() {
        return "User{" +
                "username='" + username + '\'' +
                ", password='" + password + '\'' +
                ", date=" + date +
                '}';
    }
}
```

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>
    <h3>SpringMVC入门程序</h3>

    <a href="test/hello?username=123&password=123">hello</a>

</body>
</html>
```

```
package com.bean.controller;


import com.bean.domain.Address;
import com.bean.domain.User;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.*;

import java.util.Date;

@Controller
@RequestMapping(path = "/test")
public class HelloController {

    @RequestMapping(value = "/hello")
```

```
    String sayHello(User user){
        System.out.println(user);
        return "/success";
    }

    //这个方法先执行，假装查询了数据库
    @ModelAttribute
    User model(User user){
        System.out.println("model执行了："+user);
        user.setUsername(user.getUsername());
        user.setPassword(user.getPassword());
        user.setDate(new Date());
        return user;
    }

}
```

方法二：没有返回值的

```
package com.bean.controller;


import com.bean.domain.Address;
import com.bean.domain.User;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.*;

import java.util.Date;
import java.util.Map;

@Controller
@RequestMapping(path = "/test")
public class HelloController {

    @RequestMapping(value = "/hello")
    String sayHello(@ModelAttribute(value = "userMap") User user){//获取了map集合里的对象
        System.out.println(user);
        return "/success";
    }

    //这个方法先执行，假装查询了数据库，放到了Map集合里
    @ModelAttribute
    void model(User user, Map<String,User> map){
        System.out.println("model执行了："+user);
        user.setUsername(user.getUsername());
        user.setPassword(user.getPassword());
        user.setDate(new Date());
        map.put("userMap",user);
    }

}
```

SessionAttribute

作用：

- 用于多次执行控制器方法间的参数共享。
- 只能放到类上

属性：

- value：用于指定存入的属性名称
- type：用于指定存入的数据类型。

向Session域中存值

- `index.jsp`

```jsp
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>
    <h3>SpringMVC入门程序</h3>

    <a href="test/hello">SessionAttributes</a>

</body>
</html>
```

- 控制器

```java
package com.bean.controller;


import com.bean.domain.Address;
import com.bean.domain.User;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;

import java.util.Date;
import java.util.Map;

@Controller
@RequestMapping(path = "/test")
@SessionAttributes(value = {"message"})//这里是将request域中的message再放到session中，注意，此注解只能加到类上
public class HelloController {

    @RequestMapping(value = "/hello")
    String sayHello(Model model){
        System.out.println("控制器方法执行了，底层会放到Request域中");

        //底层会放到Request域中，但是这个要配合直接@SessionAttribute才可以放到session域中
        model.addAttribute("message","消息内容");

        return "/success";
    }

}
```

- `success.jsp`

```jsp
<%@ page contentType="text/html;charset=UTF-8" language="java" isELIgnored="false" %>
<%--注意这里isELIgnored="false"，表示不忽略EL表达式--%>

<html>
<head>
    <title>Title</title>
</head>
<body>

    <h3>跳转成功</h3>
    <%--获取值--%>
    ${sessionScope.get("message")}

</body>
</html>
```

从Session域中取值

> 刚才我们所讲的Model其实是一个接口，这个接口没有取值的方法，但是他的实现类有，比如一个：`ModelMap`

```jsp
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>
    <h3>SpringMVC入门程序</h3>

    <a href="test/hello">SessionAttributes</a>

</body>
</html>
```

```java
package com.bean.controller;


import com.bean.domain.Address;
import com.bean.domain.User;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.ui.ModelMap;
import org.springframework.web.bind.annotation.*;

import java.util.Date;
import java.util.Map;

@Controller
@RequestMapping(path = "/test")
@SessionAttributes(value = {"message"})//这里是将request域中的message再放到session中，注意，此注解只能加到类上
public class HelloController {
```

```
18
19      @RequestMapping(value = "/hello")
20      String sayHello(ModelMap model){
21
22          /*从Session中取值，严格来说并不是session，而是request域*/
23          System.out.println(model.get("message"));
24
25          return "/success";
26      }
27
28      @ModelAttribute
29      void addValue(ModelMap model){
30          /*存值到request域中*/
31          model.addAttribute("message","消息内容");
32      }
33
34  }
```

相应视图和结果视图

返回值分类

字符串

`controller` 方法返回字符串可以采用指定逻辑视图名，使用视图解析器解析地址

```
1   //指定逻辑视图名，经过视图解析器解析为 jsp 物理路径：/WEB-INF/pages/success.jsp
2   @RequestMapping("/testReturnString")
3   public String testReturnString() {
4    System.out.println("AccountController 的 testReturnString 方法执行了。。。。");
5    return "success";
6   }
```

void

使用 `HttpServletRequest` 和 `HttpServletResponse`

```
1    @RequestMapping(value = "/hello")
2    void sayHello(HttpServletRequest request, HttpServletResponse response) throws
     IOException {
3
4        //1. 转发，请求一次，因为不使用视图解析器了，所以要手动指定文件和文件名
5        request.getRequestDispatcher("/WEB-INF/pages/success.jsp");
6
7        //2. 重定向，请求多次，重定向不可以直接访问到WEB-INF下的内容
8        response.sendRedirect("test");
9
10       //3. 通过response直接输出结果
11       response.setCharacterEncoding("utf-8");
```

```
        response.setContentType("application/json;charset=utf-8");
        response.getWriter().write("直接输出结果");

    }
```

ModleAndView

```
    @RequestMapping(value = "/hello")
    ModelAndView sayHello(){
        //SpringMVC为我们提供的这个方法，可以存值（底层向request域），可以跳转界面
        ModelAndView modelAndView = new ModelAndView();

        //存值
        modelAndView.addObject("key","value");

        //跳转界面
        modelAndView.setViewName("success");

        return modelAndView;
    }
```

转发和重定向

forward

```
    @RequestMapping(value = "/hello")
    String sayHello(){

        //在使用了字符串的情况下，默认就是请求转发
//        return "success";

        //或者我们也可以写为：
        return "forward:/WEB-INF/pages/success.jsp";
    }
```

注意在使用：`forward:xxx`这种写法的时候，必须要采用实际的运行路径

这种方法相当于：
`request.getRequestDispatcher("**url**").forward(request,response)`

可以写页面，也可以写其他的控制器方法

Redirect

```java
@RequestMapping(value = "/hello")
String sayHello(){

    //重定向，注意/WEB-INF下的文件重定向是找不到的，只有请求转发能找到
    return "redirect:test";
}
```

ResponseBody 响应JSON

- 使用说明

> **作用：**
>
> 该注解用于将 `Controller` 的方法返回的对象，通过 `HttpMessageConverter` 接口转换为指定格式的数据，
>
> 如： `json,xml` 等通过 Response 响应给客户端

- 使用示例

> **需求：**
>
> - 使用 `@ResponseBody` 注解实现将 `controller` 方法返回对象转换为 `json` 响应给客户端。
>
> **前置知识点：**
>
> - `Springmvc` 默认用 `MappingJacksonHttpMessageConverter` 对 `json` 数据进行转换，需要加入 `jackson` 的包。

```xml
<!--Jackson required包-->
<dependency>
  <groupId>com.fasterxml.jackson.core</groupId>
  <artifactId>jackson-databind</artifactId>
  <version>2.9.0</version>
</dependency>

<dependency>
  <groupId>com.fasterxml.jackson.core</groupId>
  <artifactId>jackson-core</artifactId>
  <version>2.9.0</version>
</dependency>

<dependency>
  <groupId>com.fasterxml.jackson.core</groupId>
  <artifactId>jackson-annotations</artifactId>
  <version>2.9.0</version>
</dependency>
```

在开始之前，说明一下几个问题：

1. 首先在 `webapp` 下面建立一个 `js` 的文件夹，里面放 `jq` 文件，然后引入到 `jsp` 中

2. 配置前端控制器

> 我们之前配置的前端控制器都是拦截所有的请求，也包括拦截 `jq` 的请求，拦截之后就不会响应了
>
> 所以我们应该在前端控制器中配置好，让他不要拦截
>
> 不仅仅是配置 `js` ，而且应该配置所有的静态资源

> 在 `springmvc.xml` 告诉前端控制器，哪些文件不要拦截

```
<?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">

 <!--这里开启注解扫描-->
 <context:component-scan base-package="com.bean"></context:component-scan>

 <!--配置视图解析器-->
 <bean id="internalResourceViewResolver"
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
     <property name="prefix" value="/WEB-INF/pages"></property>
     <property name="suffix" value=".jsp"></property>
 </bean>

 <!--告诉前端控制器，什么内容不要拦截，这里暂时是js下的所有文件
     血泪的错误：location在前边，mapping在后面，location后面不加**
     如果还不行，把target删了重新运行一遍
 -->
 <mvc:resources location="/js/" mapping="/js/**"></mvc:resources>


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

- 前端控制器再看一遍

```
<!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>

<!--配置过滤器 filter-class：这个类是spring提供的类 init-param：配置编码，设置为UTF-8-->
 <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>
  <!--配置拦截，拦截所有请求-->
  <filter-mapping>
    <filter-name>characterEncodingFilter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>

  <!--配置前端控制器 org.springframework.web.servlet.DispatcherServlet：前端控制器，这个类是固定的-->
  <servlet>
    <servlet-name>dispatcherServlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

    <!--配置读取springmvc.xml这个配置文件-->
    <init-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>classpath:springmvc.xml</param-value>
    </init-param>

    <!-- 一般来说DispatcherServlet是在请求的时候才创建的，但是这里配置的是服务器启动的时候就要创建 -->
    <load-on-startup>1</load-on-startup>
  </servlet>

  <!--url-pattern：值为"/"，说明任何的类都会拦截-->
  <servlet-mapping>
    <servlet-name>dispatcherServlet</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>

  </web-app>
```

JSON

- JSP

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
    <script src="js/jquery.min.js"></script>

</head>
<body>
    <h3>SpringMVC入门程序</h3>

    <input id="test" type="button" value="value"/>
```

```
    <script>

        $("#test").click(function(){
            $.ajax({
                type:"post",
                url:"test/hello",
                contentType:"application/json;charset=utf-8",
                data:'{"username":1,"age":"20"}',
                dataType:"json",
                success:function(data){
                    alert(data);
                }
            });
        });
    </script>

</body>
</html>
```

- JavaBean

```
package com.bean.domain;

import java.io.Serializable;

public class User implements Serializable {

    private String username;
    private Integer age;

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public Integer getAge() {
        return age;
    }

    public void setAge(Integer age) {
        this.age = age;
    }

    @Override
    public String toString() {
        return "User{" +
                "username='" + username + '\'' +
                ", age=" + age +
                '}';
    }
}
```

- Controller

```java
package com.bean.controller;


import com.bean.domain.User;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.ui.ModelMap;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.servlet.ModelAndView;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.util.Date;
import java.util.Map;

@Controller
@RequestMapping(path = "/test")
public class HelloController {

    @RequestMapping(value = "/hello")
    public @ResponseBody User sayHello(@RequestBody User user){//@ReponseBody会帮我们转为json数据
        System.out.println(user);//输出了前端传过来的json，但是后端使用jackson把json字符串封装到了user对象中

        User returnUser = new User();
        returnUser.setUsername("测试");
        returnUser.setAge(20);

        //返回的是user对象，但是前端接收到的仍然是json数据，使用的@ReopnseBody
        return returnUser;
    }

}
```

Spring实现文件上传

必要前提

1. `form` 表单的 `enctype` 取值必须是：`multipart/form-data`

   > - 默认值是 `application/x-www-form-urlencoded`
   > - `enctype` 是表单请求正文的类型

2. `method` 属性取值必须是 `Post`
3. 提供一个文件选择域 `<input type="file" />`

借助第三方组件实现文件上传

> 使用 `Commons-fileupload` 组件实现文件上传，需要导入该组件相应的支撑 `jar` 包

> - `Commons-fileupload`
> - `commons-io`
>
> `commons-io` 不属于文件上传组件的开发 `jar` 文件，但 `Commons-fileupload` 组件从 `1.1` 版本开始，它
> 工作时需要 `commons-io` 包的支持。

文件上传回顾

1. 导入 `jar` 包
2. 编写 `jsp`
3. 编写 `controller`

---

1. 导入 `jar` 包

```xml
<dependency>
    <groupId>commons-fileupload</groupId>
    <artifactId>commons-fileupload</artifactId>
    <version>1.3.1</version>
</dependency>

<dependency>
    <groupId>commons-io</groupId>
    <artifactId>commons-io</artifactId>
    <version>2.4</version>
</dependency>
```

2. 编写 `jsp`

```jsp
   <form action="user/fileupload" method="post" enctype="multipart/form-data">
       选择文件：<input type="file" name="upload"/><br/>
       <input type="submit" value="上传文件"/>
   </form>
```

3. 编写控制器

```java
package com.bean.controller;


import com.bean.domain.User;
import org.apache.commons.fileupload.FileItem;
import org.apache.commons.fileupload.FileUploadException;
import org.apache.commons.fileupload.disk.DiskFileItemFactory;
import org.apache.commons.fileupload.servlet.ServletFileUpload;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.ui.ModelMap;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.servlet.ModelAndView;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.File;
import java.io.IOException;
import java.util.Date;
```

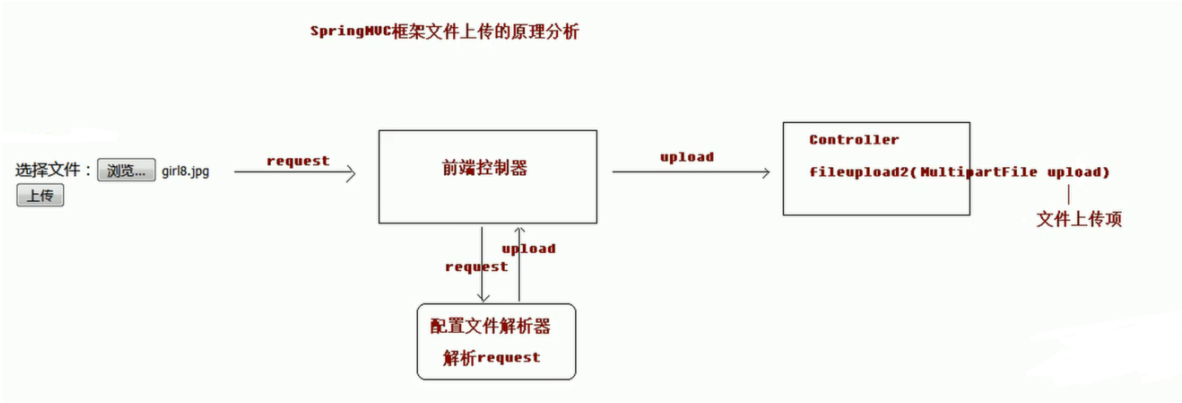
```java
import java.util.List;
import java.util.Map;
import java.util.UUID;

@Controller
@RequestMapping(path = "/user")
public class HelloController {

    @RequestMapping(value = "/fileupload")
    public String sayHello(HttpServletRequest request) throws Exception {

        //因为上传的文件都解析到了request中，所以获取request

        //1. 获取到文件上传的目录，如果没有就创造一个
        String path = request.getSession().getServletContext().getRealPath("/uploads");
            //1.1 创建File对象，获取到文件路径
        File file = new File(path);
            //1. 判断此路径是否存在，不存在就创建
        if (!file.exists()){
            file.mkdirs();
        }

        //2. 创建磁盘文件项工厂
        DiskFileItemFactory factory = new DiskFileItemFactory();
        ServletFileUpload fileUpload = new ServletFileUpload(factory);

        //3. 解析request对象
        List<FileItem> items = fileUpload.parseRequest(request);
            //3.1 遍历
        for (FileItem item : items) {
            if (item.isFormField()){
                //3.2 假如文件项是普通字段，那么不做处理
            }else {
                //3.2 假如文项是上传文件，那么上传文件
                    //3.2.1 获取到上传文件的名称
                String fileName = item.getName();
                    //3.2.2 给文件改个名字，避免重复覆盖，使用UUID，UUID生成的有-，替换为空字符串
                String replaceName = UUID.randomUUID().toString().replace("-", "");
                //3.2.3 上传文件
                item.write(new File(path,replaceName+"_"+fileName));
                //3.3 删除临时文件：文件大小小于10kb，则放到内存里，大于10kb，则会生成临时文件，所以清除临时文件
                item.delete();

            }
        }
        return "/success";
    }

}
```

`SpringMVCTest\target\SpringMVCTest\uploads`

放到了`target`包下

springMVC 使用传统的方式上传

说明



> 注意在 `jsp` 中的上传文件的 `name` 和 `Controller` 中参数名字是相同的

> 配制文件解析器的时候， `id` 必须为 `multipartResolver` ，这次这个不能随便改

实现步骤

1. 配置文件解析器
2. 编写 `jsp`
3. 编写 `Controller`

```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">

    <!--这里开启注解扫描-->
    <context:component-scan base-package="com.bean"></context:component-scan>

    <!--配置视图解析器-->
    <bean id="internalResourceViewResolver"
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/pages"></property>
        <property name="suffix" value=".jsp"></property>
    </bean>

    <!--告诉前端控制器，什么内容不要拦截-->
    <mvc:resources location="/js/" mapping="/js/**"/>

    <!--配置文件解析器
        注意id必须叫做multipartResolver,这个不准配置
```

```
    -->
    <bean id="multipartResolver"
class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
        <!--配置文件的最大值为10*1024*1024=10M-->
        <property name="maxUploadSize" value="10485760"/>
    </bean>


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

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
    <script src="js/jquery.min.js"></script>

</head>
<body>
    <h3>SpringMVC入门程序</h3>

    <form action="user/fileupload" method="post" enctype="multipart/form-data">
        选择文件：<input type="file" name="upload"/><br/><%--注意，这里叫upload，那么Controller中的参数也叫upload--%>
        <input type="submit" value="上传文件"/>
    </form>

</body>
</html>
```

```
package com.bean.controller;


import com.bean.domain.User;
import org.apache.commons.fileupload.FileItem;
import org.apache.commons.fileupload.FileUploadException;
import org.apache.commons.fileupload.disk.DiskFileItemFactory;
import org.apache.commons.fileupload.servlet.ServletFileUpload;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.ui.ModelMap;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.multipart.MultipartFile;
import org.springframework.web.servlet.ModelAndView;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.File;
import java.io.IOException;
import java.util.Date;
import java.util.List;
import java.util.Map;
import java.util.UUID;

@Controller
```

```java
@RequestMapping(path = "/user")
public class HelloController {

    @RequestMapping(value = "/fileupload")
    public String sayHello(HttpServletRequest request, MultipartFile upload) throws
Exception {

        //这里注意了，因为jsp中的name为upload，所以这里也应该是upload

        //获取到文件上传的目录，如果没有就创造一个
        String path = request.getSession().getServletContext().getRealPath("/uploads");
        File file = new File(path);
        if (!file.exists()){
            file.mkdirs();
        }

        //获取到上传文件的文件名称
        String name = upload.getOriginalFilename();
        //给文件名称唯一化
        String replaceName = UUID.randomUUID().toString().replace("-", "");

        //上传文件，不用做删除工作了，它做好了
        upload.transferTo(new File(path,replaceName+"_"+name));
        return "/success";
    }

}
```

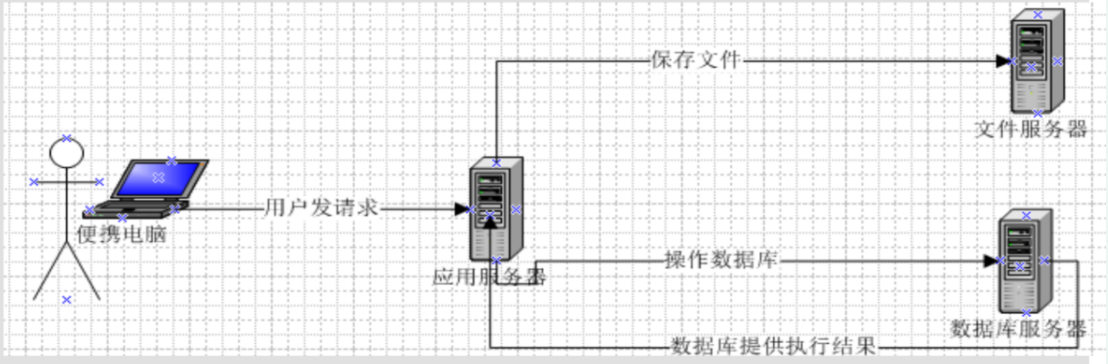
springMVC 跨服务器实现文件上传

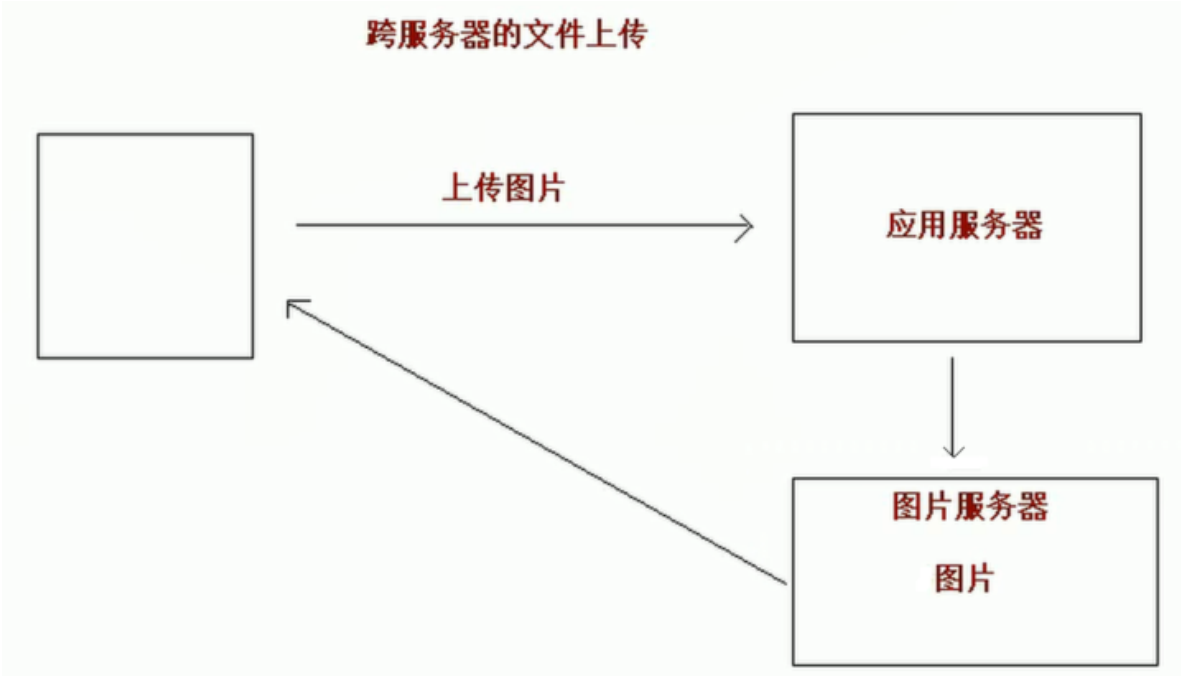
在实际开发中，我们会有很多处理不同功能的服务器。例如：

- 应用服务器：负责部署我们的应用
- 数据库服务器：运行我们的数据库
- 缓存和消息服务器：负责处理大并发访问的缓存和消息
- 文件服务器：负责存储用户上传文件的服务器。

(注意：此处说的不是服务器集群)

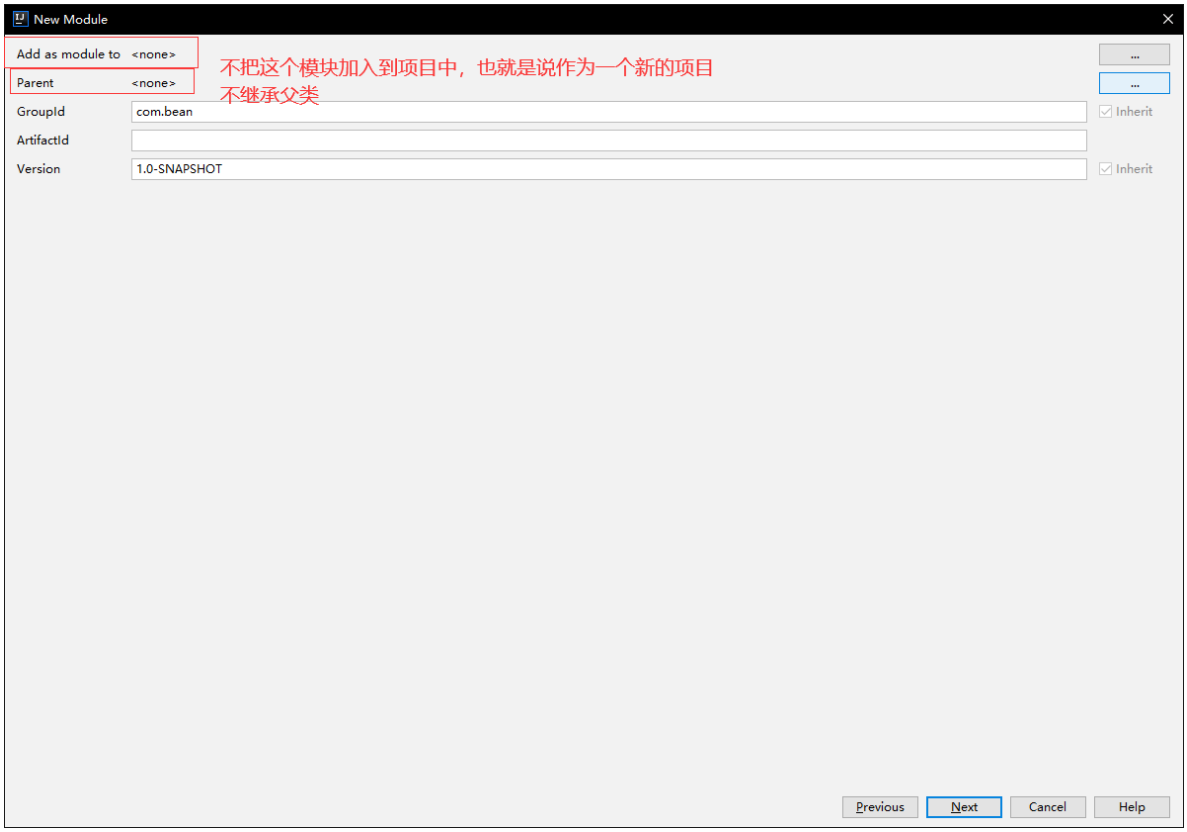
分服务器处理的目的是让服务器各司其职，从而提高我们项目的运行效率。



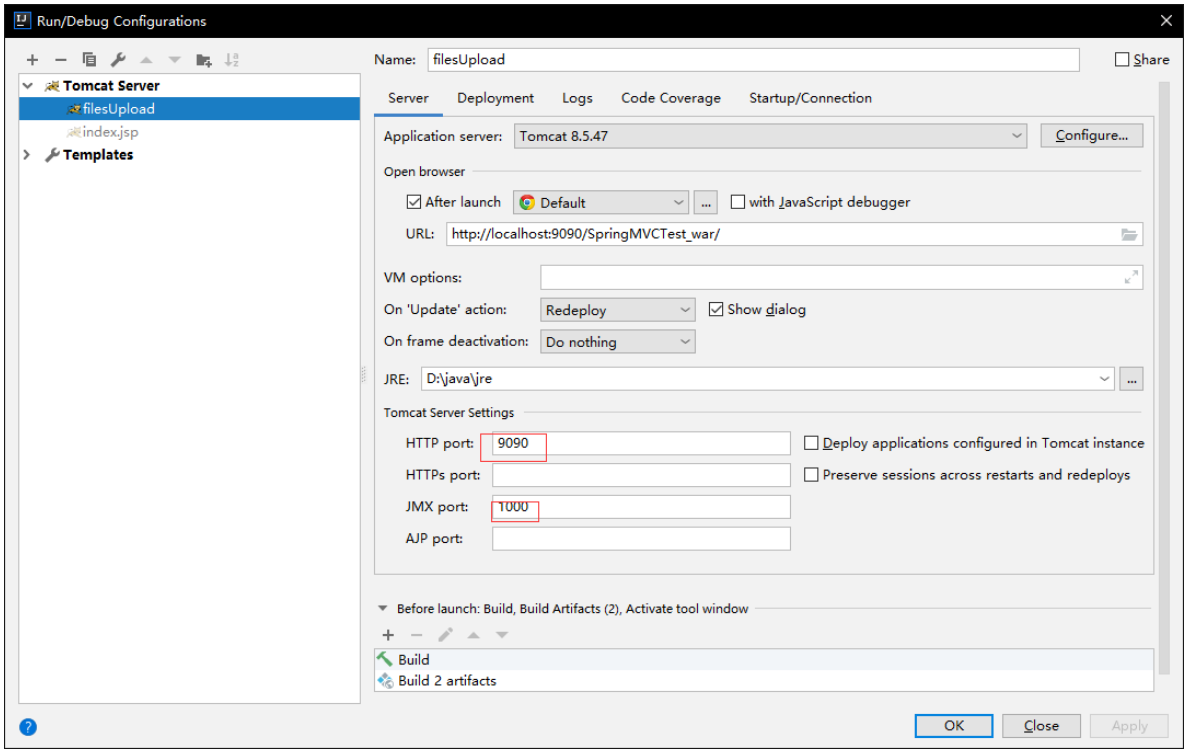


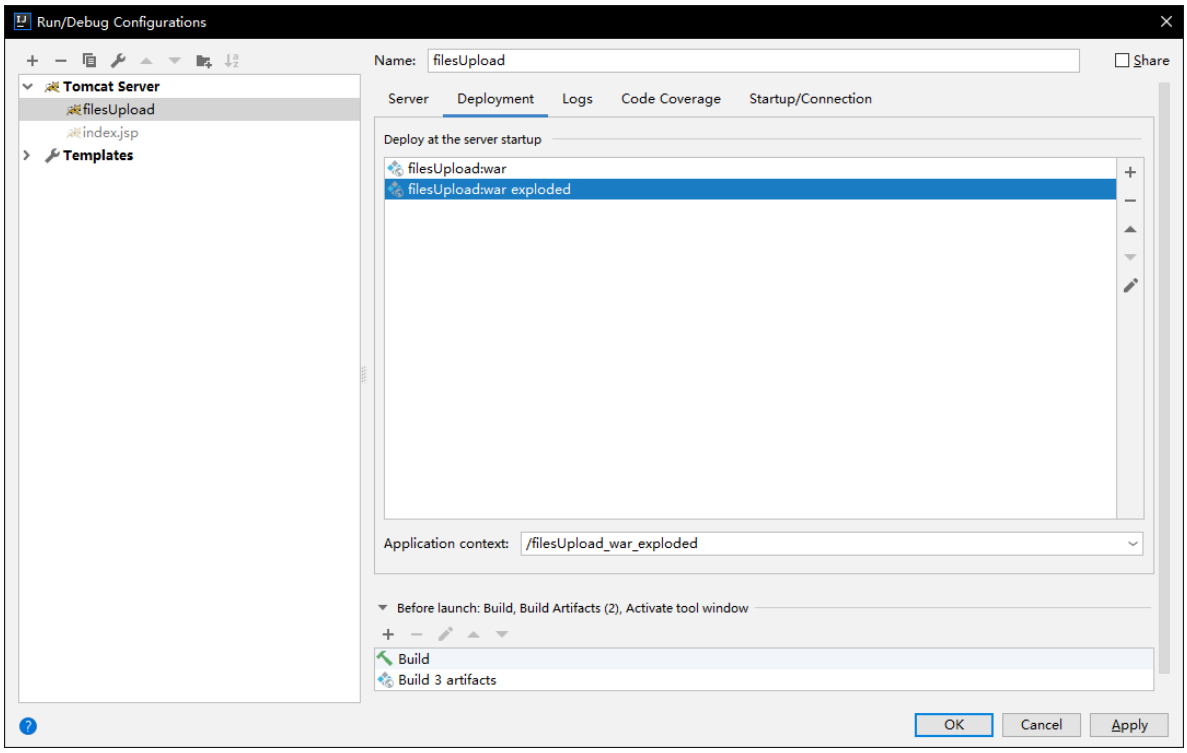
1. 准备两个 `tomcat` 服务器，并创建一个用于存放图片的 `web` 工程
2. 拷贝 `jar` 包（跨服务器开发的时候导入的 `jar` 包）
3. 编写控制器实现上传图片
4. 编写 `jsp` 页面
5. 配置解析器

> 注意了，新的 `tomcat` 在这里意味着新的项目，所以要加一个新的项目作为新的 `tomcat` 的运行



- 准备存储图片的 `tomcat`





还有一个最容易忘记的事情：建立上传文件夹，在这里我们建立 `uploads` 文件夹

在 `target` 下面建立一个 `uploads`

`filesUpload\target\filesUpload\uploads`

出错的原因：

- 服务器报405：去 `tomcat` 下面找到： `apache-tomcat-8.5.47\conf\web.xml`

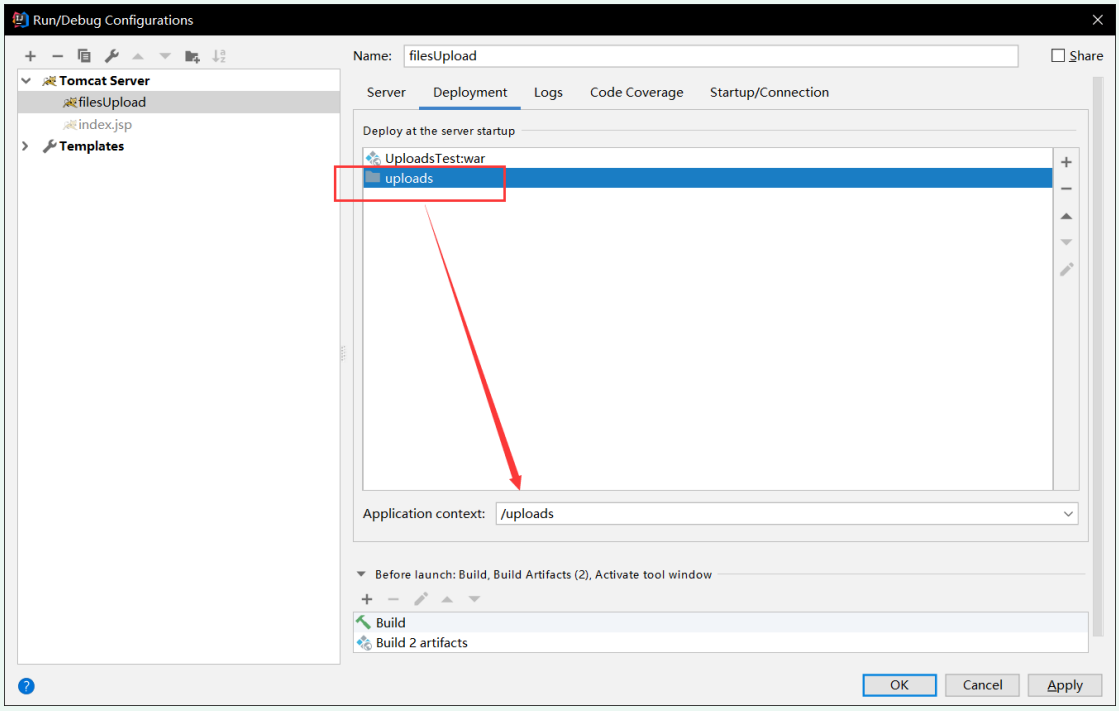
```
        <init-param>
            <param-name>readonly</param-name>
            <param-value>false</param-value>
        </init-param>
```

- 服务器报404：原因是`Controller`和`本地路径没有`没有进行关联起来，解决方法：把路径给`idea托管`

`UploadsTest\target\UploadsTest\uploads`

把这个文件放到`tomcat`服务器上进行托管

在配置第二个`tomcat`即配置存储图片服务器的时候，在idea下点开`tomcat`设置（就是配置路径的界面），手动在`` `Deployment ``中点击加号-->`ExternalSource`-->指定`uploads`文件夹，点击ok，重新启动服务器



- 拷贝`jar`包

```
        <!--跨服务器上传的jarb包-->
        <dependency>
          <groupId>com.sun.jersey</groupId>
          <artifactId>jersey-core</artifactId>
          <version>1.18.1</version>
        </dependency>

        <dependency>
          <groupId>com.sun.jersey</groupId>
          <artifactId>jersey-client</artifactId>
          <version>1.18.1</version>
        </dependency>
```

- 编写 `jsp`

1. 发送的 `jsp`

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
    <script src="js/jquery.min.js"></script>

</head>
<body>
    <h3>SpringMVC入门程序</h3>

    <form action="user/fileupload" method="post" enctype="multipart/form-data">
        选择文件：<input type="file" name="upload"/><br/><%--注意，这里叫upload，那么Controller中的参数也叫upload--%>
        <input type="submit" value="上传文件"/>
    </form>

</body>
</html>
```

2. 接受的 `jsp`

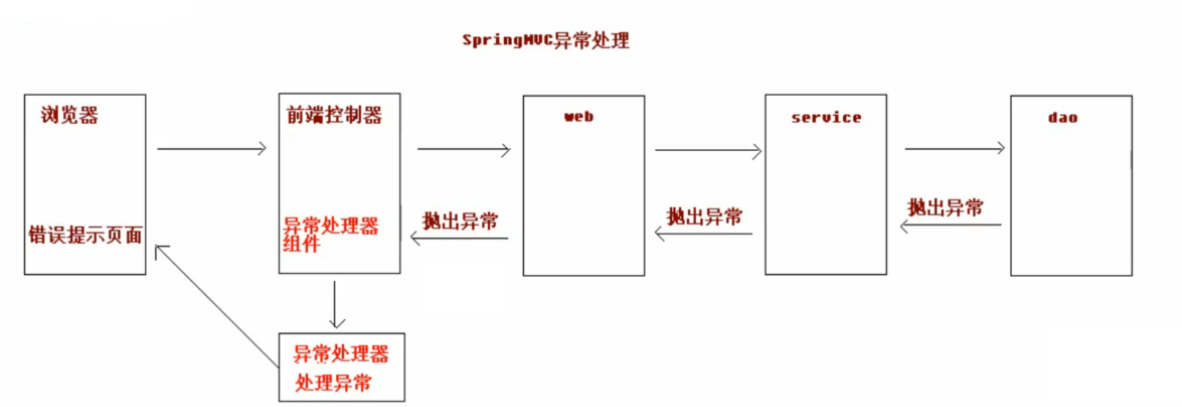
```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>filesUpload</title>
</head>
<body>
    <h3>文件上传</h3>
</body>
</html>
```

- 编写 `Controller`

```
package com.bean.controller;


import com.bean.domain.User;
import com.sun.jersey.api.client.Client;
import com.sun.jersey.api.client.WebResource;
import org.apache.commons.fileupload.FileItem;
import org.apache.commons.fileupload.FileUploadException;
import org.apache.commons.fileupload.disk.DiskFileItemFactory;
import org.apache.commons.fileupload.servlet.ServletFileUpload;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.ui.ModelMap;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.multipart.MultipartFile;
import org.springframework.web.servlet.ModelAndView;
```

```java

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.File;
import java.io.IOException;
import java.util.Date;
import java.util.List;
import java.util.Map;
import java.util.UUID;

@Controller
@RequestMapping(path = "/user")
public class HelloController {

    @RequestMapping(value = "/fileupload")
    public String sayHello( MultipartFile upload) throws Exception {

        //因为我们是跨服务器上传文件，所以我们就不需要Request了

        //1. 获取到上传文件的文件名称
        String name = upload.getOriginalFilename();
        //2. 给文件名称唯一化
        String replaceName = UUID.randomUUID().toString().replace("-", "")+"_"+name;
        //3. 上传文件，当然需要链接服务器
            //3.1 定义图片服务器的请求路径，因为我们启动的另一个服务器为：http://localhost:9090，并且要向文件夹uploads下传文件，所以这么定义
        String path = "http://localhost:9090/uploads/";
            //3.2 创建客户端对象
        Client client = Client.create();//注意导包import com.sun.jersey.api.client.Client;
            /*
            * 3.3 链接图片服务器，注意这里还有一个小细节，这里的总路径是：http://localhost:9090/uploads/文件名称
            * 其中"uploads/文件名称"中间的斜杠必须要有，要么在这里拼接，要么直接在path面前定义
            * */
        WebResource webResource = client.resource(path + replaceName);
        //3.4 上传文件，以字节形式上传
        webResource.put(upload.getBytes());
        return "/success";
    }

}
```

Spring异常处理



> 从上面的图我们可以看到处理异常的时候应该是在 `Controller` --> `前端控制器` 这方面进行的，所以应该配置一个异常处理器来进行异常的处理

使用异常处理

1. 编写自定义异常类（做提示信息）和错误页面
2. 在 `Controller` 中捕获异常，并抛出自己自定义的异常
3. 编写异常处理器类
4. 配置异常处理器（跳转到异常界面去）

---

1. 编写异常类和错误页面

- `com.exception.SysException`

```java
package com.bean.exception;

public class SysException extends Exception {
    private String message;

    public SysException(String message) {
        this.message = message;
    }

    public SysException() {
    }

    @Override
    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }
}
```

- `exception.jsp`

```jsp
<%@ page contentType="text/html;charset=UTF-8" language="java" isELIgnored="false" %>
<html>
<head>
    <title>Error</title>
</head>
<body>
    <h3>错误界面</h3>
    ${message}
</body>
</html>
```

- `index.jsp`

```jsp
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
    <script src="js/jquery.min.js"></script>

</head>
<body>
    <h3>SpringMVC入门程序</h3>

    <a href="user/hello">异常处理器</a>

</body>
</html>

```

---

2. 捕获异常并抛出异常

```java
package com.bean.controller;


import com.bean.exception.SysException;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.*;

@Controller
@RequestMapping(path = "/user")
public class HelloController {

    @RequestMapping(value = "/hello")
    public String sayHello() throws SysException {

        try {
            int i = 1/0;
        } catch (Exception e) {
            e.printStackTrace();
            throw new SysException("异常");
        }

        return "/success";
```

```
    }

}
```

3. 编写异常处理器类（实现 `HanderExceptionResolver` ）

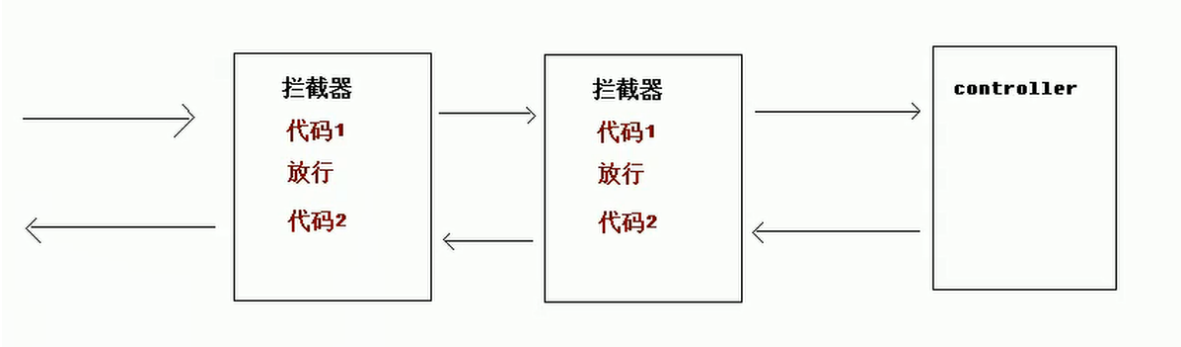
```
package com.bean.exception;

import org.springframework.web.servlet.HandlerExceptionResolver;
import org.springframework.web.servlet.ModelAndView;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;


public class SysExceptionResolver implements HandlerExceptionResolver {

    /**
     * 编写异常处理器
     * @param httpServletRequest
     * @param httpServletResponse
     * @param o
     * @param e 捕获之后被抛出的异常
     * @return
     */
    public ModelAndView resolveException(HttpServletRequest httpServletRequest,
HttpServletResponse httpServletResponse, Object o, Exception e) {

        SysException exception = null;

        if (e instanceof SysException){
            exception = (SysException) e;
        }else {
            exception = new SysException("系统维护");
        }

        ModelAndView modelAndView = new ModelAndView();
        //获取到信息，存入request数据域中，message:exception.getMessage()
        modelAndView.addObject("message",exception.getMessage());
        //跳转到exception界面
        modelAndView.setViewName("/exception");

        return modelAndView;
    }
}
```

4. 配置异常处理器（这个当作正常的 `bean` 对象配置就好了）

```
<?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">

    <!--这里开启注解扫描-->
    <context:component-scan base-package="com.bean"></context:component-scan>

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

    <!--配置视图解析器-->
    <bean id="exceptionResolver"
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/pages"></property>
        <property name="suffix" value=".jsp"></property>
    </bean>

    <!--告诉前端控制器，什么内容不要拦截-->
    <mvc:resources location="/js/" mapping="/js/**"/>

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

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

Spring拦截器



实现拦截器

1. 编写 `jsp`

2. 编写 `Controller`
3. 编写拦截器（实现 `HandlerInterceptor` ）
4. 配置拦截器

---

1. 编写 `jsp`

- `index.jsp`

```jsp
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
    <script src="js/jquery.min.js"></script>

</head>
<body>
    <h3>SpringMVC入门程序</h3>

    <a href="user/hello">拦截器</a>

</body>
</html>
```

- `success.jsp`

```jsp
<%@ page contentType="text/html;charset=UTF-8" language="java" isELIgnored="false" %>
<%--注意这里isELIgnored="false"，表示不忽略EL表达式--%>

<html>
<head>
    <title>Title</title>
</head>
<body>

    <h3>跳转成功</h3>
    <%System.out.println("success.jsp页面跳转成功了");%>
</body>
</html>
```

---

2. 编写 `Controller`

```java
package com.bean.controller;


import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.*;

@Controller
@RequestMapping(path = "/user")
public class HelloController {

    @RequestMapping(value = "/hello")
    public String sayHello(){
        return "/success";
    }
```

```
15
16  }
```

3. 编写拦截器

- 第一个拦截器

```
1   package com.bean.Interceptor;
2
3   import org.springframework.web.servlet.HandlerInterceptor;
4   import org.springframework.web.servlet.ModelAndView;
5
6   import javax.servlet.http.HttpServletRequest;
7   import javax.servlet.http.HttpServletResponse;
8
9
10  /**
11   * 这里是第一个拦截器
12   * 在实现方法没有发现报错的时候不要慌，因为jdk1.8新特性允许接口自己实现
13   * 要实现自己的功能还是要重写方法
14   */
15  public class BeforeInterceptor implements HandlerInterceptor {
16
17      /**
18       * 在Controller之前执行
19       * @param request
20       * @param response
21       * @param handler
22       * @return  (true：放行，进入下一个拦截器或者Controller)
23       * (false：不放行，可以利用request和response直接跳转错误界面)
24       * @throws Exception
25       */
26      public boolean preHandle(HttpServletRequest request, HttpServletResponse response,
    Object handler) throws Exception {
27
28          //可以跳转界面，但是别忘记返回false
29          //request.getRequestDispatcher("/WEB-
    INF/pages/exception.jsp").forward(request,response);
30          //return false;
31
32          System.out.println("BeforeInterceptor中的preHandle方法执行了");
33          return true;
34      }
35
36      /**
37       * 在Controller之后执行，这个也可以利用request和response跳转界面
38       * @param request
39       * @param response
40       * @param handler
41       * @param modelAndView
42       * @throws Exception
43       */
44      public void postHandle(HttpServletRequest request, HttpServletResponse response, Object
    handler, ModelAndView modelAndView) throws Exception {
45          //可以跳转界面
46          //request.getRequestDispatcher("/WEB-
    INF/pages/exception.jsp").forward(request,response);
```

```
47
48          System.out.println("BeforeInterceptor中的postHandle方法执行了");
49      }
50
51
52      /**
53       * 在所有事情（包括跳转jsp）都做完之后执行
54       * 所以注意了，既然所有的事情都做完了，在这里不能够跳转界面了
55       * 也就是说，不可以在这里跳转界面
56       * @param request
57       * @param response
58       * @param handler
59       * @param ex
60       * @throws Exception
61       */
62      public void afterCompletion(HttpServletRequest request, HttpServletResponse response,
    Object handler, Exception ex) throws Exception {
63          System.out.println("BeforeInterceptor中的afterCompletion方法执行了");
64      }
65  }
```

- 第二个拦截器

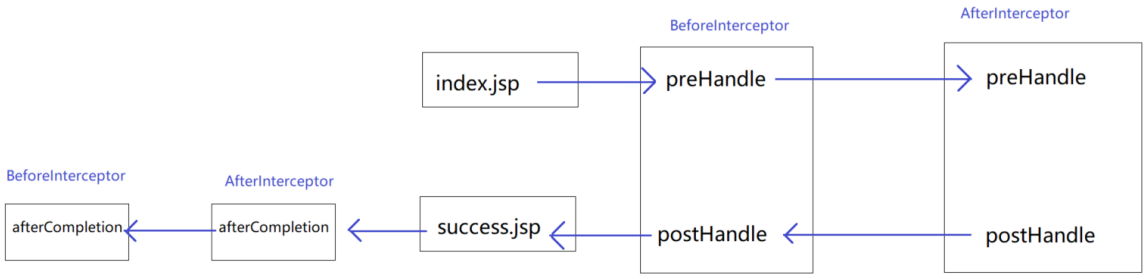
```
1   package com.bean.Interceptor;
2
3   import org.springframework.web.servlet.HandlerInterceptor;
4   import org.springframework.web.servlet.ModelAndView;
5
6   import javax.servlet.http.HttpServletRequest;
7   import javax.servlet.http.HttpServletResponse;
8
9   /**
10   * 这里是第二个拦截器
11   */
12  public class AfterInterceptor implements HandlerInterceptor {
13
14      /**
15       * 在Controller之前执行
16       * @param request
17       * @param response
18       * @param handler
19       * @return  (true：放行，进入下一个拦截器或者Controller)
20       * (false：不放行，可以利用request和response直接跳转错误界面)
21       * @throws Exception
22       */
23      public boolean preHandle(HttpServletRequest request, HttpServletResponse response,
    Object handler) throws Exception {
24
25          System.out.println("AfterInterceptor中的preHandle方法执行了");
26
27          return true;
28      }
29
30      /**
31       * 在Controller之后执行，这个也可以利用request和response跳转界面
32       * @param request
33       * @param response
34       * @param handler
```

```
     * @param modelAndView
     * @throws Exception
     */
    public void postHandle(HttpServletRequest request, HttpServletResponse response, Object handler, ModelAndView modelAndView) throws Exception {

        System.out.println("AfterInterceptor中的postHandle方法执行了");
    }


    /**
     * 在所有事情（包括跳转jsp）都做完之后执行
     * 所以注意了，既然所有的事情都做完了，在这里不能够跳转界面了
     * 也就是说，不可以在这里跳转界面
     * @param request
     * @param response
     * @param handler
     * @param ex
     * @throws Exception
     */
    public void afterCompletion(HttpServletRequest request, HttpServletResponse response, Object handler, Exception ex) throws Exception {
        System.out.println("AfterInterceptor中的afterCompletion方法执行了");
    }
}
```

4. 配置拦截器

```
<?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">

    <!--这里开启注解扫描-->
    <context:component-scan base-package="com.bean"></context:component-scan>

    <!--配置拦截器们-->
    <mvc:interceptors>
        <!--配置第一个拦截器-->
        <mvc:interceptor>
            <!--配置要拦截的内容，这里配置的是访问user路径下的所有文件-->
            <mvc:mapping path="/user/*"/>
            <!--配置不要拦截的内容-->
            <!--<mvc:exclude-mapping path=""/>-->
            <!--配置拦截器，就像配置普通bean对象即可-->
            <bean id="beforeInterceptor" class="com.bean.Interceptor.BeforeInterceptor"/>
        </mvc:interceptor>

```

```
        <mvc:interceptor>
            <mvc:mapping path="/user/*"/>
            <!--<mvc:exclude-mapping path=""/>-->
            <bean id="" class="com.bean.Interceptor.AfterInterceptor"/>
        </mvc:interceptor>
    </mvc:interceptors>

    <!--配置视图解析器-->
    <bean id="exceptionResolver"
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/pages"></property>
        <property name="suffix" value=".jsp"></property>
    </bean>

    <!--告诉前端控制器，什么内容不要拦截-->
    <mvc:resources location="/js/" mapping="/js/**"/>

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

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




```
BeforeInterceptor中的preHandle方法执行了
AfterInterceptor中的preHandle方法执行了
AfterInterceptor中的postHandle方法执行了
BeforeInterceptor中的postHandle方法执行了
success.jsp页面跳转成功了
AfterInterceptor中的afterCompletion方法执行了
BeforeInterceptor中的afterCompletion方法执行了
```