一、SpringMVC Introduction

1、What is MVC ?

MVC是一种软件架构的思想，将软件按照模型、视图、控制器来划分

MVC is a kind of software architecture idea, the software is divided according to model, view, controller

M：Model，模型层，指工程中的JavaBean，作用是处理数据

M: Model, a JavaBean in engineering that handles data

Javabeans divided into two categories:

- 一类称为实体类Bean：专门存储业务数据的，如 Student、User 等
  One category is called entity beans: that specialize in storing business data, such as Student, User, and so on
- 一类称为业务处理 Bean：指 Service 或 Dao 对象，专门用于处理业务逻辑和数据访问
  Another category is called business processing beans: like Service or Dao objects , that specialize in handling business logic and data access

V：View，视图层，指工程中的html或jsp等页面，作用是与用户进行交互，展示数据

V: View, view layer, refers to the html or jsp and other pages in the project, which is used to interact with the user and display data

C：Controller，控制层，指工程中的servlet，作用是接收请求和响应浏览器

C: Controller, the control layer, refers to the servlet in the project, which is responsible for receiving requests and responding to the browser

MVC的工作流程：

How MVC works:

用户通过视图层发送请求到服务器，在服务器中请求被Controller接收，Controller调用相应的Model层处理请求，处理完毕将结果返回到Controller，Controller再根据请求处理的结果找到相应的View视图，渲染数据后最终响应给浏览器
The user sends a request to the server through the View layer, and the request is received by the Controller in the server. The Controller invokes the corresponding Model layer to process the request, and returns the result to the Controller, and the Controller finds the corresponding view according to the request processing result. The data is rendered and the final response is sent back to the browser

2、What is SpringMVC?

SpringMVC是Spring的一个后续产品，是Spring的一个子项目
SpringMVC is a successor to Spring, a subproject of Spring

SpringMVC 是 Spring 为表述层开发提供的一整套完备的解决方案。在表述层框架历经 Strust、WebWork、Strust2 等诸多产品的历代更迭之后，目前业界普遍选择了 SpringMVC 作为 Java EE 项目表述层开发的**首选方案**。
SpringMVC is a complete set of solutions provided by Spring for the development of presentation layer. After the evolution of presentation layer framework, such as Strust, WebWork, Strust2 and

so on, SpringMVC is generally chosen as the first choice for presentation layer development in Java EE projects.

> 注：三层架构分为表述层（或表示层）、业务逻辑层、数据访问层，表述层表示前台页面和后台servlet
> Note: The three-tier architecture is divided into presentation layer (or presentation layer), business logic layer and data access layer. The expression layer represents the front page and the background servlet

3、SpringMVC的特点

Features of SpringMVC

- **Spring 家族的原生产品**，与 IOC 容器等基础设施无缝对接
  Spring MVC is a native product of the spring family and works seamlessly with the IOC container
- **基于原生的Servlet**，通过了功能强大的**前端控制器DispatcherServlet**，对请求和响应进行统一处理
  Based on native Servlet, through the powerful **DispatcherServlet** front-end controller, the request and response are handled in a unified way
- 提供表述层全面解决方案
  Provides a comprehensive solution to the presentation layer
- **代码清新简洁**，大幅度提升开发效率
  Clean and concise code, which greatly improves development productivity
- 内部组件化程度高，可插拔式组件**即插即用**，想要什么功能配置相应组件即可
  High degree of internal components, pluggable components **plug and play**, what function you want to configure the corresponding component
- **性能卓著**，尤其适合现代大型、超大型互联网项目要求
  Excellent performance, especially suitable for modern large, very large scale Internet projects

二、入门案例 Introductory Case

1、开发环境 development environment

IDE：idea 2019.2

Project bundler：maven3.5.4

Web Server：tomcat8

Spring version：5.3.1

2、创建maven工程 Create maven project named springMVC-demo1

a>packaging：war

b>引入依赖 Import dependencies

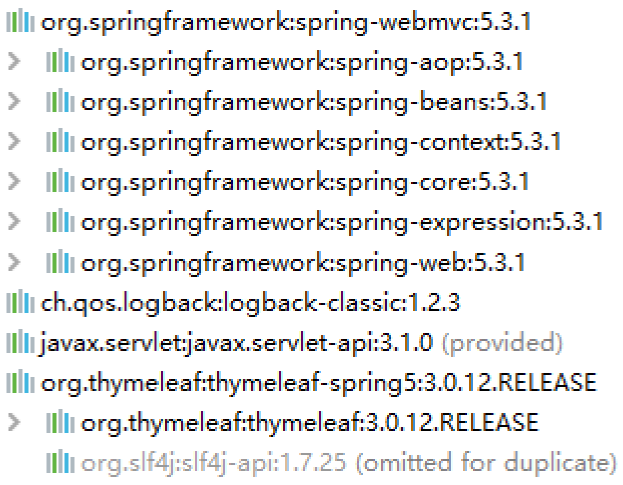
https://mvnrepository.com/

```
<dependencies>
    <!-- SpringMVC -->
    <dependency>
```

```
        <groupId>org.springframework</groupId>
        <artifactId>spring-webmvc</artifactId>
        <version>5.3.1</version>
    </dependency>

    <!-- logback -->
    <dependency>
        <groupId>ch.qos.logback</groupId>
        <artifactId>logback-classic</artifactId>
        <version>1.2.3</version>
    </dependency>

    <!-- ServletAPI -->
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>javax.servlet-api</artifactId>
        <version>3.1.0</version>
        <scope>provided</scope>
    </dependency>
    <!-- jsp-api -->
        <dependency>
            <groupId>javax.servlet.jsp</groupId>
            <artifactId>javax.servlet.jsp-api</artifactId>
            <version>2.3.3</version>
            <scope>provided</scope>
        </dependency>
</dependencies>
```

注：由于 Maven 的传递性，我们不必将所有需要的包全部配置依赖，而是配置最顶端的依赖，其他靠传递性导入。

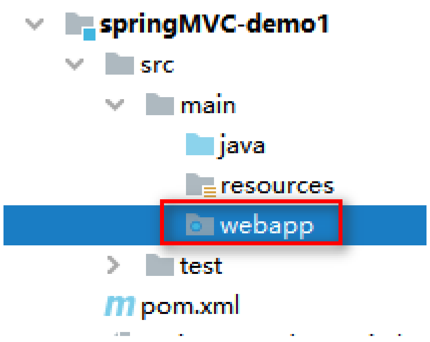
Although we only introduced one dependency, due to the transitivity of maven, it can automatically import all other related dependencies.



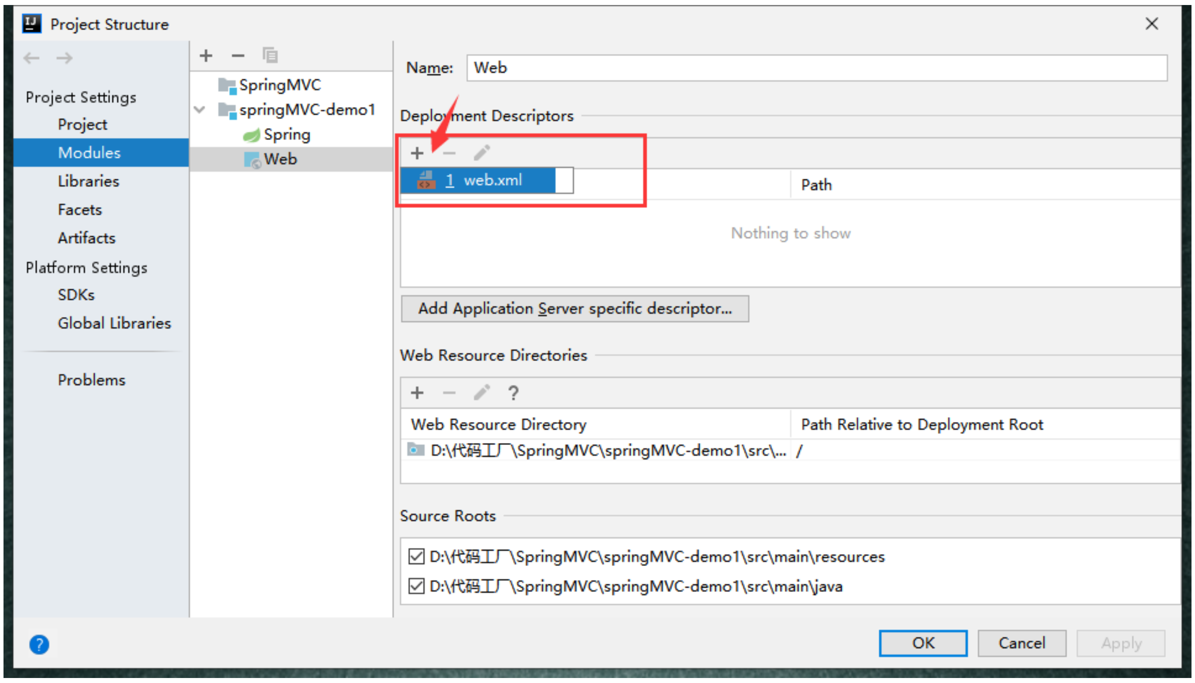
```
org.springframework:spring-webmvc:5.3.1
  > org.springframework:spring-aop:5.3.1
  > org.springframework:spring-beans:5.3.1
  > org.springframework:spring-context:5.3.1
  > org.springframework:spring-core:5.3.1
  > org.springframework:spring-expression:5.3.1
  > org.springframework:spring-web:5.3.1
ch.qos.logback:logback-classic:1.2.3
javax.servlet:javax.servlet-api:3.1.0 (provided)
org.thymeleaf:thymeleaf-spring5:3.0.12.RELEASE
  > org.thymeleaf:thymeleaf:3.0.12.RELEASE
    org.slf4j:slf4j-api:1.7.25 (omitted for duplicate)
```

c>添加web模块 Adding web module

create webapp folder



choose Project Structure，in Deployment Descriptors, click + plus sign，adding web.xml file



Name: Web

Deployment Descriptors

1 web.xml | Path

Nothing to show

Add Application Server specific descriptor...

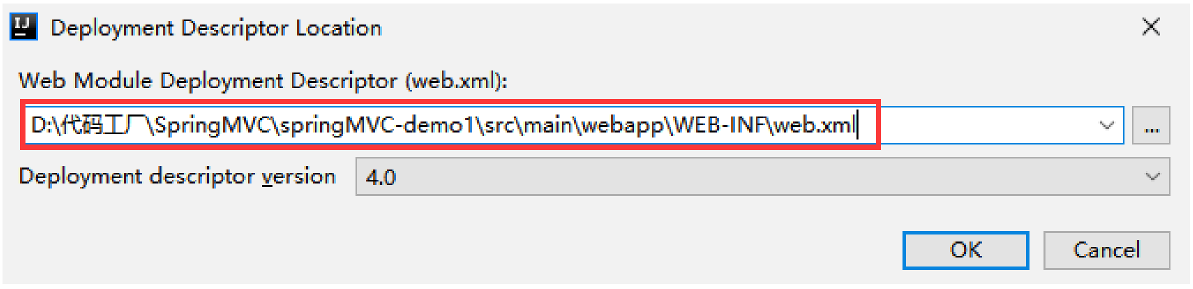
Web Resource Directories

Web Resource Directory	Path Relative to Deployment Root
D:\代码工厂\SpringMVC\springMVC-demo1\src\...	/

Source Roots

- D:\代码工厂\SpringMVC\springMVC-demo1\src\main\resources
- D:\代码工厂\SpringMVC\springMVC-demo1\src\main\java

Notice：The path needs to be complete



Web Module Deployment Descriptor (web.xml):

D:\代码工厂\SpringMVC\springMVC-demo1\src\main\webapp\WEB-INF\web.xml

Deployment descriptor version 4.0

3、Configure web.xml

注册SpringMVC的前端控制器DispatcherServlet
Register SpringMVC's DispatcherServlet front-end controller

a>默认配置方式 Default configuration

此配置作用下，SpringMVC的配置文件默认位于WEB-INF下，默认名称为<servlet-name>-servlet.xml，例如，以下配置所对应SpringMVC的配置文件位于WEB-INF下，文件名为springMVC-servlet.xml
With this configuration, the SpringMVC configuration file is located in the WEB-INF directory by default with the default name `<servlet-name>-servlet.xml`. For example, the SpringMVC

configuration file corresponding to the following configuration is located in the WEB-INF directory. The file is called springMVC-servlet.xml

```
<!-- 配置SpringMVC的前端控制器，对浏览器发送的请求统一进行处理 The front-end controller
of SpringMVC is configured to handle the requests sent by the browser uniformly -
->
<servlet>
    <servlet-name>springMVC</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-
class>
</servlet>
<servlet-mapping>
    <servlet-name>springMVC</servlet-name>
    <!--
        设置springMVC的核心控制器所能处理的请求的请求路径
        Sets the request path that DispatcherServlet can be handled
    -->
    <url-pattern>/</url-pattern>
</servlet-mapping>
```

b>扩展配置方式 Extended Configuration

可通过init-param标签设置SpringMVC配置文件的位置和名称，通过load-on-startup标签设置SpringMVC前端控制器DispatcherServlet的初始化时间

You can use the init-param tag to set the location and name of SpringMVC configuration file, and the load-on-startup tag to set the initialization time of DispatcherServlet for SpringMVC controller

```
<!-- 配置SpringMVC的前端控制器，对浏览器发送的请求统一进行处理 The front-end controller
of SpringMVC is configured to handle the requests sent by the browser uniformly -
->
<servlet>
    <servlet-name>springMVC</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-
class>
    <!-- 通过初始化参数指定SpringMVC配置文件的位置和名称 Specify the location and name
of the SpringMVC configuration file through the initialization parameters -->
    <init-param>
        <!-- contextConfigLocation is fixed value -->
        <param-name>contextConfigLocation</param-name>
        <!-- 使用classpa表示从类路径查找配置文件，例如maven工程中的src/main/resources --
>
        <!-- Use the classpa prefix to find the configuration file from the
classpath
, For example, src/main/resources in the maven project -->
        <param-value>classpaspringMVC.xml</param-value>
    </init-param>
    <!--
        作为框架的核心组件，在启动过程中有大量的初始化操作要做
        而这些操作放在第一次请求时才执行会严重影响访问速度
        因此需要通过此标签将启动控制DispatcherServlet的初始化时间提前到服务器启动时
As the core component of the framework, DispatcherServlet has a lot of
initialization operations to do during the startup process, and these operations
will seriously affect the access speed when the first request is executed.
Therefore, it is necessary to advance the initialization time of
DispatcherServlet to the startup of the server through this label.
    -->
```

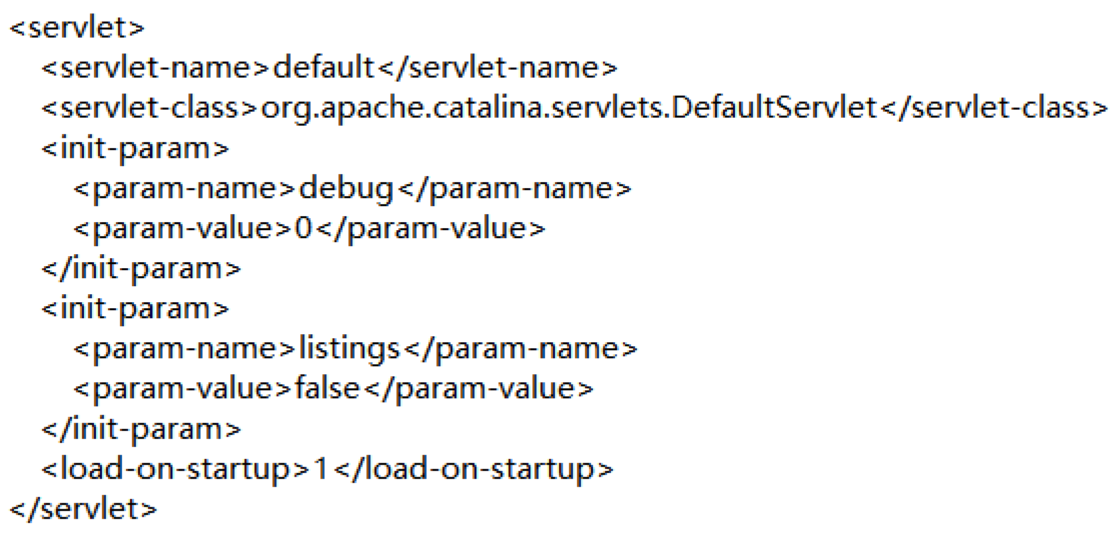
```
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>springMVC</servlet-name>
        <!--
            设置springMVC的核心控制器所能处理的请求的请求路径
            Sets the request path that DispatcherServlet can be handled
        -->
        <url-pattern>/</url-pattern>
    </servlet-mapping>
```

Note:

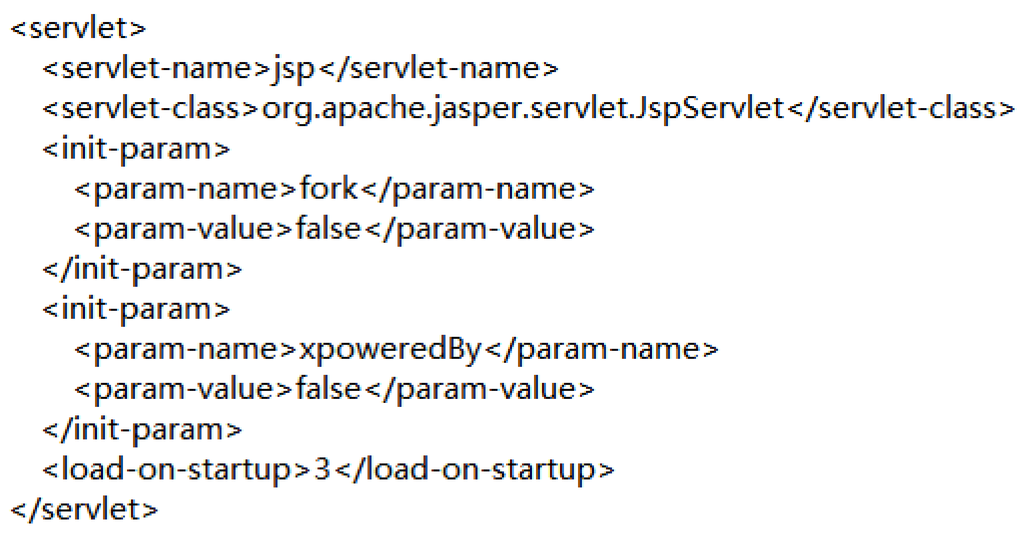
The difference between using/and /*(asterisk) in tags:

/代表除了.jsp以外的所有请求
/ represents all requests except.jsp

当我们请求页面的静态资源，如.jpg图片、.png图片、.js文件、.css文件等时，前端控制器会如何处理这些请求呢?
When we request static resources for the page, such as. jpg images, .png images, .js files, .css files, etc., what does the front-end controller do with these requests?



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



```
<servlet>
  <servlet-name>jsp</servlet-name>
  <servlet-class>org.apache.jasper.servlet.JspServlet</servlet-class>
  <init-param>
    <param-name>fork</param-name>
    <param-value>false</param-value>
  </init-param>
  <init-param>
    <param-name>xpoweredBy</param-name>
    <param-value>false</param-value>
  </init-param>
  <load-on-startup>3</load-on-startup>
</servlet>
```

we can find the code in web.xml in tomcat server conf folder.

/*则能够匹配所有请求，例如在使用过滤器时，若需要对所有请求进行过滤，就需要使用/*的写法
/* matches all requests. For example, when using a filter, if you want to filter all requests, you need to use /*

4、创建请求控制器 Creating Request Controller

由于前端控制器对浏览器发送的请求进行了统一的处理，但是具体的请求有不同的处理过程，因此需要创建处理具体请求的类，即请求控制器
Requests sent by the browser are uniformly processed by the front-end controller, but specific requests have different processing processes. Therefore, you need to create a class to process specific requests, which is the request controller

请求控制器中每一个处理请求的方法称为控制器方法
Each method in the request controller that handles the request is called the controller method

因为SpringMVC的控制器由一个POJO（普通的Java类）担任，因此需要通过@Controller注解将其标识为一个控制层组件，交给Spring的IoC容器管理，此时SpringMVC才能够识别控制器的存在
Because SpringMVC's controller is a POJO (plain Java class), it needs to be identified as a control-layer component through the @Controller annotation and handed over to Spring's IoC container to manage so that SpringMVC can recognize the controller's existence

```
@Controller
public class HelloController {

}
```

5、创建springMVC的配置文件 Create springMVC configuration file

```
<!-- 自动扫描包 automatic scanning package -->
<context:component-scan base-package="com.qixin.mvc.controller"/>

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

<!--
处理静态资源，例如html、js、css、jpg。在springMVC核心配置文件中配置<mvc:default-servlet-
handler />后，会在Spring MVC上下文中定义一个
org.springframework.web.servlet.resource.DefaultServletHttpRequestHandler，它会像
一个检查员，对进入DispatcherServlet的URL进行筛查，如果发现是静态资源的请求，就将该请求转由
Web应用服务器默认的Servlet处理，如果不是静态资源的请求，才由DispatcherServlet继续处理。
Used to process static resources such as html, js, css, jpg. After configuring
<mvc:default-servlet-handler /> in the springMVC core configuration file, In the
Spring MVC context defines a org. Springframework. Web. Servlet. Resource.
DefaultServletHttpRequestHandler, it will be like an inspector, The URL entering
DispatcherServlet is screened. If it is found to be a request for static
resources, the request is transferred to the default Servlet of the Web
application server for processing. If it is not a request for static resources,
the DispatcherServlet continues to process it.
 -->
<mvc:default-servlet-handler/>
```

```
<!-- 开启mvc注解驱动 enable mvc annotation driven -->
<mvc:annotation-driven/>
```

6、Testing HelloWorld

a>实现对首页的访问 visit the home page

在请求控制器中创建处理请求的方法
In the request controller, create a method to handle the request

```
// @RequestMapping：处理请求和控制器方法之间的映射关系 Handles the mapping between
requests and controller methods
// @RequestMapping注解的value属性可以通过请求地址匹配请求，/表示的当前工程的上下文路径
// The @RequestMapping annotation has a value attribute that matches the request
by the request address, and / represents the context path of the current project
// localhost:8080/springMVC/
@RequestMapping("/")
public String index() {
    //set the view name
    return "index";
}
```

b>编写视图页面 Writing the view page index.jsp (/WEB-INF/views/)

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Index</title>
</head>
<body>
<h2>Index</h2>
<a href="<%=request.getContextPath()%>/hello">hello</a>
</body>
</html>
```

在请求控制器中创建处理请求的方法 In the request controller, create a method to handle the request

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

install jRebel plugins

https://blog.csdn.net/weixin_44233253/article/details/118788185

三、@RequestMapping

1、@RequestMapping注解的功能

@RequestMapping注解的作用就是将请求和处理请求的控制器方法关联起来，建立映射关系。

The @RequestMapping annotation establishes a mapping between a request and the controller methods that handle it

SpringMVC 接收到指定的请求，就会来找到在映射关系中对应的控制器方法来处理这个请求。

SpringMVC receives a specific request and finds the corresponding controller method in the mapping to handle the request.

2、@RequestMapping注解的位置 annotation location

@RequestMapping标识一个类时：设置请求路径的初始信息

When @RequestMapping identifies a class: Sets initial information about the request path

@RequestMapping标识一个方法时：设置请求路径的具体信息

When @RequestMapping identifies a method: Sets specific information about the request path

```
@Controller
@RequestMapping("/test")
public class RequestMappingController {

    //at this time the request path is: /test/testRequestMapping
    @RequestMapping("/testRequestMapping")
    public String testRequestMapping(){
        return "success";
    }

}
```

3、@RequestMapping注解的value属性 The value proerty of the @RequestMapping annotation

@RequestMapping注解的value属性通过请求的请求地址匹配请求映射

The @RequestMapping annotation has a value property that matches the request mapping with the request address

@RequestMapping注解的value属性是一个字符串类型的数组，表示该请求映射能够匹配多个请求地址所对应的请求

The @RequestMapping annotation's value property is an array of strings, indicating that the request map can match multiple request addresses

@RequestMapping注解的value属性必须设置 The value property of the @RequestMapping annotation must be set

```
<a href="<%=request.getContextPath()%>/testRequestMapping">测试@RequestMapping的value属性-->/testRequestMapping</a><br>
<a href="<%=request.getContextPath()%>/test">测试@RequestMapping的value属性-->/test</a><br>
```

```
    @RequestMapping(value = {"/testRequestMapping", "/test"})
    public String testRequestMapping(){
        return "success";
    }
```

4、@RequestMapping注解的method属性 method property of @RequestMapping annotation

@RequestMapping注解的method属性通过请求的请求方式（get或post）匹配请求映射

The method property of the @RequestMapping annotation matches the request mapping by the request method (get or post)

@RequestMapping注解的method属性是一个RequestMethod类型的数组，表示该请求映射能够匹配多种请求方式

The method property of the @RequestMapping annotation is an array of RequestMethod types, indicating that the request map can match multiple request modes

若当前请求的请求地址满足请求映射的value属性，但是请求方式不满足method属性，则浏览器报错405：Request method 'POST' not supported

If the current Request satisfies the value property of the request map, but the request method does not satisfy the method property, then the browser will report an error 405: Request method 'POST' not supported

```
<a href="<%=request.getContextPath()%>/test">测试@RequestMapping的value属性-->/test</a><br>
<form action="<%=request.getContextPath()%>/test" method="post">
    <input type="submit">
</form>
```

```
@RequestMapping(
        value = {"/testRequestMapping", "/test"},
        method = {RequestMethod.GET, RequestMethod.POST}
)
public String testRequestMapping(){
    return "success";
}
```

> 注：
>
> 1、对于处理指定请求的控制器方法，SpringMVC中提供了@RequestMapping的派生注解 SpringMVC provides a @RequestMapping derived annotation for controller methods that handle specific requests
>
> 处理get请求的映射-->@GetMapping
>
> 处理post请求的映射-->@PostMapping
>
> 处理put请求的映射-->@PutMapping
>
> 处理delete请求的映射-->@DeleteMapping
>
> 2、Common request methods are get，post，put，delete

> 但是目前浏览器只支持get和post，若在form表单提交时，为method设置了其他请求方式的字符串（put或delete），则按照默认的请求方式get处理
> However, the browser currently only supports get and post, and if the form is submitted with another request method (put or delete), the default request method is get
>
> 若要发送put和delete请求，则需要通过spring提供的过滤器HiddenHttpMethodFilter，在RESTful部分会讲到
> If we want to send put and delete requests, we need using spring's HiddenHttpMethodFilter, which we'll cover in the RESTful section later

8、SpringMVC路径中的占位符（重点）Placeholders in the SpringMVC path (important)

The original way：/deleteUser?id=1

```
<a href="<%=request.getContextPath()%>/testRest1?id=1&username=qixin">Testing placeholders in the path-->/testRest1</a><br>
```

```
@RequestMapping("/testRest1")
public String testRest1(String id, String username){
    System.out.println("id:"+id+",username:"+username);
    return "success";
}
```

RESTful style：/deleteUser/1

SpringMVC路径中的占位符常用于RESTful风格中，当请求路径中将某些数据通过路径的方式传输到服务器中，就可以在@RequestMapping注解的value属性中通过占位符{xxx}表示传输的数据，在通过@PathVariable注解，将占位符所表示的数据赋值给控制器方法的形参
Placeholders in SpringMVC paths are often used in RESTful style. When the request path sends some data to the server by the path, you can use the {xxx} placeholder in the value property of the @RequestMapping annotation to represent the transferred data. Through @PathVariable annotation, the data assigned to the parameter of controller method by placeholder .

```
<a href="<%=request.getContextPath()%>/testRest2/1/admin">Placeholders in the test path-->/testRest2</a><br>
```

```
@RequestMapping("/testRest2/{id}/{username}")
public String testRest2(@PathVariable("id") String id, @PathVariable("username") String username){
    System.out.println("id:"+id+",username:"+username);
    return "success";
}
//output-->id:1,username:admin
```

四、SpringMVC获取请求参数 gets the request parameters

创建新的param.jsp页面和页面跳转控制器ParamController，在控制器中添加页面跳转代码
Create a new param.jsp page and a new page-hopping controller named ParamController

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>param</title>
</head>
<body>
<h2>param</h2>
</body>
</html>
```

```
@Controller
public class ParamController {

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

1、通过原生ServletAPI获取 Using the native ServletAPI

将HttpServletRequest作为控制器方法的形参，此时HttpServletRequest类型的参数表示封装了当前请求报文的对象
When you pass HttpServletRequest as a parameter to the controller method, the HttpServletRequest type parameter represents the object that encapsulates the current request.

```
<a href="<%=request.getContextPath()%>/testParamServletAPI?
username=admin&password=123456">Test gets the request parameters--
>/testServletAPI</a><br>
```

```
@RequestMapping("/testParamServletAPI")
public String testParamServletAPI(HttpServletRequest request){
    String username = request.getParameter("username");
    String password = request.getParameter("password");
    System.out.println("username:"+username+",password:"+password);
    return "success";
}
```

2、通过控制器方法的形参获取请求参数 Get the request parameters through the parameters of the controller method

在控制器方法的形参位置，设置和请求参数同名的形参，当浏览器发送请求，匹配到请求映射时，在DispatcherServlet中就会将请求参数赋值给相应的形参
In the formal parameter position of the controller method, set the parameter with the same name as the request parameter. When the browser sends a request and matches the request mapping, the DispatcherServlet will assign the request parameter to the corresponding formal parameters

```
<a href="<%=request.getContextPath()%>/testParam1?
username=admin&password=123456">Test gets the request parameters--
>/testParam1</a><br>
```

```java
@RequestMapping("/testParam1")
public String testParam1(String username, String password){
    System.out.println("username:"+username+",password:"+password);
    return "success";
}
```

> Note：
>
> 若请求所传输的请求参数中有多个同名的请求参数，此时可以在控制器方法的形参中设置字符串数组或者字符串类型的形参接收此请求参数
> If there are multiple request parameters with the same name in the request parameter transmitted by the request, you can set the string array or string type parameter in the formal parameter of the controller method to accept the request parameter
>
> 若使用字符串数组类型的形参，此参数的数组中包含了每一个数据
>
> If you use an array of strings as a parameter, then the array contains each piece of data
>
> 若使用字符串类型的形参，此参数的值为每个数据中间使用逗号拼接的结果
> If a string is used as a parameter, the value of the parameter is the result of a comma concatenation between each data

```html
<form action="<%=request.getContextPath()%>/testParam2" method="post">
    username:<input type="text" name="username"/><br>
    password:<input type="password" name="password"/><br>
    hobby:<input type="checkbox" name="hobby" value="a"/>a
    <input type="checkbox" name="hobby" value="b"/>b
    <input type="checkbox" name="hobby" value="c"/>c<br>
    <input type="submit" value="Test gets the request parameters">
</form>
```

```java
@RequestMapping("/testParam2")
public String testParam2(String username, String password, String hobby){

 System.out.println("username:"+username+",password:"+password+",hobby:"+hobby);
    return "success";
}
or
@RequestMapping("/testParam2")
public String testParam2(String username, String password, String[] hobby){
    System.out.println("username:"+username+",password:"+password+",hobby:"+
Arrays.toString(hobby));
    return "success";
}
```

3、@RequestParam annotation

@RequestParam是将请求参数和控制器方法的形参创建映射关系
@RequestParam annotation creates a mapping between request parameters and controller method parameters

The @RequestParam annotation has three properties:

- value：指定为形参赋值的请求参数的参数名
  Specifies the parameter name of the request parameter assigned to the formal parameter

- required：设置是否必须传输此请求参数，默认值为true
  Sets whether this request must be transmitted. The default value is true

若设置为true时，则当前请求必须传输value所指定的请求参数，若没有传输该请求参数，且没有设置defaultValue属性，则页面报错400：Required String parameter 'xxx' is not present；

If this parameter is set to true, the request parameter specified in value property must be transmitted. If the request parameter is not transmitted and the defaultValue property is not set, an error 400 is displayed on the page: Required String parameter 'xxx' is not present;

若设置为false，则当前请求不是必须传输value所指定的请求参数，若没有传输，则注解所标识的形参的值为null

If set to false, the current request does not have to transmit the request parameters specified by value. If not, the value of the parameter identified by the annotation is null.

- defaultValue：不管required属性值为true或false，当value所指定的请求参数没有传输或传输的值为""时，则使用默认值为形参赋值
  Regardless of whether the required property value is true or false, the default value is assigned to the parameter when the request parameter specified by value is not transmitted or when the value is blank value

```
<form action="<%=request.getContextPath()%>/testParam" method="post">
    username:<input type="text" name="user_name"/><br>
    password:<input type="password" name="password"/><br>
    hobby:<input type="checkbox" name="hobby" value="a"/>a
    <input type="checkbox" name="hobby" value="b"/>b
    <input type="checkbox" name="hobby" value="c"/>c<br>
    <input type="submit" value="Test gets the request parameters">
</form>
```

```
@RequestMapping(value = "/testParam", method = RequestMethod.POST)
public String testParam(@RequestParam("user_name") String username, String
password, String hobby){

 System.out.println("username:"+username+",password:"+password+",hobby:"+hobby);
    return "success";
}
```

4、通过POJO获取请求参数 Get the request parameters through the POJO(Plain Ordinary Java Object)

如果获取的请求参数比较多，则可以通过POJO方式获取请求参数。
If we receive more parameters, we can retrieve the request parameters via the POJO.

可以在控制器方法的形参位置设置一个实体类型的形参，此时若浏览器传输的请求参数的参数名和实体类中的属性名一致，那么请求参数就会为此属性赋值。
A parameter of the entity type can be set in the parameter position of the controller method, and if the parameter name passed by the browser matches the property name in the entity class, the request parameter will be assigned to that property.

```
public class User {
    private String username;
    private String password;
    private String gender;
    private int age;
    private String email;
    ......
}
```

```
<form action="<%=request.getContextPath()%>/testPojo" method="post">
    username: <input type="text" name="username"><br>
    password: <input type="password" name="password"><br>
    gender: <input type="radio" name="gender" value="male">male<input
type="radio" name="gender" value="female">female<br>
    age: <input type="text" name="age"><br>
    email: <input type="text" name="email"><br>
    <input type="submit" value="Use an entity class to receive the request
parameters">
</form>
```

```
@RequestMapping("/testPojo")
public String testPOJO(User user) throws IOException {
    System.out.println(user);
    return "success";
}
//result-->User{username='é?????', password='123', gender='male', age=45,
email='qixin622@163.com'}}
```

5、解决获取请求参数的乱码问题 Solve the garbled code problem of getting request parameters

解决获取请求参数的乱码问题，可以使用SpringMVC提供的编码过滤器CharacterEncodingFilter，但是必须在web.xml中进行注册
To solve the problem of getting the request parameters, you can use the CharacterEncodingFilter, this filter provided by SpringMVC, but it must be registered in web.xml

```
<!--config springMVC character encoding filter-->
<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>
    <init-param>
        <param-name>forceRequestEncoding</param-name>
        <param-value>true</param-value>
    </init-param>
    <init-param>
        <param-name>forceResponseEncoding</param-name>
        <param-value>true</param-value>
    </init-param>
</filter>
```

```
<filter-mapping>
    <filter-name>CharacterEncodingFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

> Note：
>
> SpringMVC中处理编码的过滤器一定要配置到其他过滤器之前，否则无效
> CharacterEncodingFilter must be configured before other filters, or they won't work

五、域对象共享数据 Scope objects share data

建立新的SpringMVC项目springMVC-demo2

1、使用ServletAPI向request域对象共享数据 Use the ServletAPI to share data with the request scope object

```
scope.jsp
<a href="<%=request.getContextPath()%>/testScopeServletAPI">Use the ServletAPI
to share data with the request domain object</a><br>
```

```
ScopeController.java
@Controller
public class ScopeController {
    @RequestMapping("/scope")
    public String scope() {
        return "scope";
    }
}
@RequestMapping("/testScopeServletAPI")
public String testScopeServletAPI(HttpServletRequest request){
    request.setAttribute("testScope", "hello,ScopeServletAPI");
    return "success";
}
```

```
success.jsp
<p>${testScope}</p>
success
```

2、使用ModelAndView向request域对象共享数据 Use ModelAndView to share data with the request scope object

```
scope.jsp
<a href="<%=request.getContextPath()%>/testModelAndView">Use ModelAndView to
share data with the request domain object</a><br>
```

```
ScopeController.java
@RequestMapping("/testModelAndView")
public ModelAndView testModelAndView(){
    /**
```

```
     * ModelAndView有Model和View的功能 ModelAndView has the functionality of Model
and View
     * Model主要用于向请求域共享数据 The Model is mainly used to share data to the
requesting domain
     * View主要用于设置视图，实现页面跳转 View is mainly used to set the view and
realize the page jump
     */
    ModelAndView mav = new ModelAndView();
    //向请求域共享数据 Share data to the request domain
    mav.addObject("testScope", "hello,ModelAndView");
    //设置视图，实现页面跳转 Set the view to achieve page jump
    mav.setViewName("success");
    return mav;
}
```

3、使用Model向request域对象共享数据 Use the Model to share data with the request domain object

```
scope.jsp
<a href="<%=request.getContextPath()%>/testModel">Use the Model to share data
with the request domain object</a><br>
```

```
ScopeController.java
@RequestMapping("/testModel")
public String testModel(Model model){
    model.addAttribute("testScope", "hello,Model");
    return "success";
}
```

4、使用map向request域对象共享数据 Use map object to share data with the request domain object

```
scope.jsp
<a href="<%=request.getContextPath()%>/testMap">Use map object to share data
with the request domain object</a><br>
```

```
ScopeController.java
@RequestMapping("/testMap")
public String testMap(Map<String, Object> map){
    map.put("testScope", "hello,Map");
    return "success";
}
```

5、使用ModelMap向request域对象共享数据 Use ModelMap object to share data with the request domain object

```
scope.jsp
<a href="<%=request.getContextPath()%>/testModelMap">Use ModelMap object to
share data with the request domain object</a><br>
```

```
ScopeController.java
@RequestMapping("/testModelMap")
public String testModelMap(ModelMap modelMap){
    modelMap.addAttribute("testScope", "hello,ModelMap");
    return "success";
}
```

6、向session域共享数据 The session domain object shares data

```
scope.jsp
<a href="<%=request.getContextPath()%>/testSession">use session domain object
shares data</a><br>
```

```
@RequestMapping("/testSession")
public String testSession(HttpSession session){
    session.setAttribute("testSessionScope", "hello,session");
    return "success";
}
```

```
success.jsp
<p>${testSessionScope}</p>
```

7、使用application域共享数据 Use the application domain to share data

```
scope.jsp
<a href="<%=request.getContextPath()%>/testApplication">Use the application
domain to share data</a><br>
```

```
@RequestMapping("/testApplication")
public String testApplication(HttpSession session){
    ServletContext application = session.getServletContext();
    application.setAttribute("testApplicationScope", "hello,application");
    return "success";
}
```

```
success.jsp
<p>${testApplicationScope}</p>
```

六、SpringMVC的视图 VIEW

SpringMVC中的视图是View接口，视图的作用渲染数据，将模型Model中的数据展示给用户
The View in SpringMVC is the implementation class of the View interface. The role of the view is to render data and show the data in the Model to the user

SpringMVC视图的种类很多，默认有转发视图和重定向视图
There are many kinds of SpringMVC views.By default, there are forward views and redirect views

当控制器方法中所设置的视图名称没有任何前缀时，此时的视图名称会被SpringMVC配置文件中所配置的视图解析器解析，视图名称拼接视图前缀和视图后缀所得到的最终路径，会通过转发的方式实现跳转
When the view name is set in the controller method without any prefix, the view name will be resolved by the view resolver configured in the SpringMVC configuration file, and the final path obtained by concatenate the view name with the view prefix and the view suffix will be forwarded.

1、转发视图 forward view

SpringMVC中默认的转发视图是InternalResourceView
The default view is InternalResourceView in SpringMVC

SpringMVC中创建转发视图的情况：
How to create a forwarding view in SpringMVC:

当控制器方法中所设置的视图名称以"forward:"为前缀时，创建InternalResourceView视图，此时的视图名称不会被SpringMVC配置文件中所配置的视图解析器解析，而是会将前缀"forward:"去掉，剩余部分作为最终路径通过转发的方式实现跳转
When the view name set in the controller method is prefixed with "forward:", the InternalResourceView is created. The view name will not be resolved by the view resolver configured in the SpringMVC configuration file, but will remove the prefix "forward:". The remaining part is used as the final path to realize the jump by forwarding

For example "forward:/"，"forward:/employee"

```
view.jsp
<a href="<%=request.getContextPath()%>/testForward">Test
InternalResourceView</a><br>
```

```
ViewController.java
@Controller
public class ViewController {

    @RequestMapping("/view")
    public String view() {
        return "view";
    }

    @RequestMapping("/testForward")
    public String testForward(){
        return "forward:/";
    }
}
```

2、重定向视图 Redirect View

SpringMVC中默认的重定向视图是RedirectView
The default redirection view in SpringMVC is the RedirectView

当控制器方法中所设置的视图名称以"redirect:"为前缀时，创建RedirectView视图，此时的视图名称不会被SpringMVC配置文件中所配置的视图解析器解析，而是会将前缀"redirect:"去掉，剩余部分作为最终路径通过重定向的方式实现跳转
The RedirectView is created when the view name set in the controller method is prefixed with "redirect:". Instead of being resolved by the view resolver configured in the SpringMVC configuration file, the "redirect:" prefix is removed. The remaining part is used as the final path to realize the jump by redirection.

For example "redirect:/"，"redirect:/employee"

```
view.jsp
<a href="<%=request.getContextPath()%>/testRedirect">Test RedirectView</a><br>
```

```
ViewController.java
@RequestMapping("/testRedirect")
public String testRedirect(){
    return "redirect:/";
}
```

4、view-controller

如果控制器方法仅仅用来实现页面跳转，即只需要设置视图名称时，可以将处理器方法使用view-controller标签来进行表示
When the controller method is only used to navigate to the page, that is, when you just need to set the view name, you can use the view-controller tag to represent the handler method

```
<!--
    path：设置处理的请求地址 Sets the address of the request to be processed
    view-name：设置请求地址所对应的视图名称 Sets the view name for the request address
-->
<mvc:view-controller path="/testView" view-name="success"></mvc:view-controller>
```

> Note：
>
> 当SpringMVC中设置任何一个view-controller时，其他控制器中的请求映射将全部失效，此时需要在SpringMVC的核心配置文件中设置开启mvc注解驱动的标签：
> When any of the View-controllers are set in SpringMVC, the request mappings in the other controllers will not work. In this case, we need to set a tag in the SpringMVC core configuration file to enable mvc annotation driven:
>
> <mvc:annotation-driven />

七、RESTful

1、RESTful Introduction

RESTFUL是一种网络应用程序的设计风格和开发方式
RESTFUL is a design style and development method of network applications

REST：**Re**presentational **S**tate **T**ransfer，It is the presentation layer resource state transfer

表现层资源状态转移。

资源是一种看待服务器的方式，即，将服务器看作是由很多离散的资源组成。每个资源是服务器上一个可命名的抽象概念。它能代表服务器文件系统中的一个文件、数据库中的一张表等等具体的东西。一个资源可以由一个或多个URI来标识。URI既是资源的名称，也是资源在Web上的地址。对某个资源感兴趣的客户端应用，可以通过资源的URI与其进行交互。

Resources is a way of looking at a server as a collection of discrete resources. Each resource is a namable abstraction on the server. It can represent a file in a server's filesystem, a table in a database, and so on. A resource can be identified by one or more URIs. A URI is both the name of a resource and its address on the Web. Client applications interested in a resource can interact with it via the resource URI.

2、RESTful implementation

具体说，就是 HTTP 协议里面，四个表示操作方式的动词：GET、POST、PUT、DELETE。

Specifically, the four verbs that represent actions in HTTP: GET, POST, PUT, and DELETE.

它们分别对应四种基本操作：GET 用来获取资源，POST 用来新建资源，PUT 用来更新资源，DELETE 用来删除资源。

They correspond to four basic operations: GET to retrieve a resource, POST to create a new resource, PUT to update a resource, and DELETE to delete a resource.

REST 风格提倡 URL 地址使用统一的风格设计，从前到后各个单词使用斜杠分开，不使用问号键值对方式携带请求参数，而是将要发送给服务器的数据作为 URL 地址的一部分，以保证整体风格的一致性。

The REST style encourages urls to be styled in a consistent manner, with each word separated by a slash from the beginning to the end, and instead of using question mark key-value pairs to carry request parameters, the data to be sent to the server should be part of the URL address to ensure consistency.

operation	traditional way	REST style
query operation	getUserById?id=1	user/1-->get request method
save operation	saveUser	user-->post request method
delete operation	deleteUser?id=1	user/1-->delete request method
update operation	updateUser	user-->put request method

Example：

a>create UserController.java controller in springMVC-demo3 project

b>config in web.xml (write by yourself)

c>config in springMVC.xml (write by yourself)

d>create UserController

```
@Controller
public class UserController {
    /**
     * Using RESTFul simulation of user resources to add, delete, update and
query
     * /user    GET     query all the users information
     * /user/1  GET     query user information by user id
     * /user    POST    add user information
     * /user/1  DELETE  delete user information
     * /user/1  PUT     update user information
     */
    @RequestMapping(value = "/user", method = RequestMethod.GET)
    public String getAllUser(Model model) {
        System.out.println("query all the users information");
        return "success";
    }

    @RequestMapping(value = "/user/{id}", method = RequestMethod.GET)
    public String getUserById(@PathVariable("id") Integer id) {
        System.out.println("query user information with id: " + id);
        return "success";
    }

    @RequestMapping(value = "/user", method = RequestMethod.POST)
    public String addUser(String username, String password) {
        System.out.println("add user information: " + username + "," + password);
        return "redirect:/";
    }
}
```

e>在views文件夹中创建index.jsp页面

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Index</title>
</head>
<body>
<h2>Index</h2>
<a href="<%=request.getContextPath()%>/user">query all users information</a><br>
<a href="<%=request.getContextPath()%>/user/1">query user information by id</a>
<br>
<form action="<%=request.getContextPath()%>/user" method="post">
  username:<input type="text" name="username"><br>
  password:<input type="password" name="password"><br>
  <input type="submit" value="add user">
</form>
</body>
</html>
```

f>create success.jsp in views directory

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>success</title>
</head>
<body>
<h2>success</h2>
</body>
</html>
```

g>create view controller in springMVC.xml configuration file

```
<mvc:view-controller path="/" view-name="index"></mvc:view-controller>
```

h>next step, send a put request

```
@RequestMapping(value = "/user", method = RequestMethod.PUT)
public String updateUser(String username, String password) {
    System.out.println("update user information: " + username + "," + password);
    return "redirect:/";
}
```

```
<form action="<%=request.getContextPath()%>/user" method="put">
    username:<input type="text" name="username"><br>
    password:<input type="password" name="password"><br>
    <input type="submit" value="update user">
</form>
```

If you look at the results, it's still send a get request.

i>register `HiddenHttpMethodFilter` filter in web.xml configuration file.

```
<filter>
    <filter-name>HiddenHttpMethodFilter</filter-name>
    <filter-class>org.springframework.web.filter.HiddenHttpMethodFilter</filter-
class>
</filter>
<filter-mapping>
    <filter-name>HiddenHttpMethodFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

j>modify the form code，a put request will be performed.

```
<form action="<%=request.getContextPath()%>/user" method="post">
    <input type="hidden" name="_method" value="put">
    username:<input type="text" name="username"><br>
    password:<input type="password" name="password"><br>
    <input type="submit" value="update user">
</form>
```

3、HiddenHttpMethodFilter

由于浏览器只支持发送get和post方式的请求，那么该如何发送put和delete请求呢?
Because browsers only support get and post requests, what about send put or delete requests?

SpringMVC 提供了 **HiddenHttpMethodFilter** 帮助我们**将 POST 请求转换为 DELETE 或 PUT 请求**

SpringMVC provides **HiddenHttpMethodFilter** to help us convert a **POST** request into a **DELETE** or **PUT** request

HiddenHttpMethodFilter 处理put和delete请求的条件:
HiddenHttpMethodFilter handles conditions for put and delete requests:

a>The request method for the current request must be **POST**

b>The request parameter **_method** must be transmitted for the current request

满足以上条件，**HiddenHttpMethodFilter** 过滤器就会将当前请求的请求方式转换为请求参数*method*
的值，因此请求参数_*method*的值才是最终的请求方式
*If the above conditions are met, the **HiddenHttpMethodFilter** will convert the request mode of the current request to the value of the request parameter* **method**, so the _method** parameter is the final request mode.

> Notice:
>
> 目前为止，SpringMVC中提供了两个过滤器：CharacterEncodingFilter和HiddenHttpMethodFilter

> 在web.xml中注册时，必须先注册CharacterEncodingFilter，再注册HiddenHttpMethodFilter
>
> So far, we have used two filters: CharacterEncodingFilter and HiddenHttpMethodFilter
>
> When registering with web.xml, you must first register the CharacterEncodingFilter and then the HiddenHttpMethodFilter
>
> Cause：
>
> - Through the request in the CharacterEncodingFilter. SetCharacterEncoding (encoding) method to set the character set
> - request.setCharacterEncoding(encoding) require this method does not have any access to and operation of the request parameters
> - And the HiddenHttpMethodFilter has exactly one operation to get the request method:
>
>   `String paramValue = request.getParameter(param);`

八、RESTful examples

1、准备工作 Preparation

Function: Realize the CRUD of employee information

- Set up the enviroment

  creating springMVC-rest project

- Import dependencies

```
<packaging>war</packaging>
......
<dependencies>
        <!-- SpringMVC -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
            <version>5.3.1</version>
        </dependency>

        <!-- logback -->
        <dependency>
            <groupId>ch.qos.logback</groupId>
            <artifactId>logback-classic</artifactId>
            <version>1.2.3</version>
        </dependency>

        <!-- ServletAPI -->
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>javax.servlet-api</artifactId>
            <version>3.1.0</version>
            <scope>provided</scope>
        </dependency>
        <!-- jsp-api -->
        <dependency>
            <groupId>javax.servlet.jsp</groupId>
            <artifactId>javax.servlet.jsp-api</artifactId>
            <version>2.3.3</version>
            <scope>provided</scope>
        </dependency>
        <!--jstl-->
        <dependency>
            <groupId>jstl</groupId>
            <artifactId>jstl</artifactId>
            <version>1.2</version>
        </dependency>
    </dependencies>
```

- create web.xml configuration file

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
         version="4.0">
    <servlet>
        <servlet-name>DispatcherServlet</servlet-name>
        <servlet-
class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>classpath:springMVC.xml</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>DispatcherServlet</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
</web-app>
```

- create springMVC.xml configuration file

```
<?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"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context
https://www.springframework.org/schema/context/spring-context.xsd
http://www.springframework.org/schema/mvc
https://www.springframework.org/schema/mvc/spring-mvc.xsd">

    <context:component-scan base-package="com.huat"></context:component-scan>

    <bean
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/views/"/>
        <property name="suffix" value=".jsp"/>
    </bean>
    <mvc:view-controller path="/" view-name="index"/>

    <mvc:default-servlet-handler/>
    <mvc:annotation-driven/>
</beans>
```

- Create entity class Employee.java

```
package com.huat.bean;

public class Employee {

   private Integer id;
   private String lastName;

   private String email;
   //1 male, 0 female
   private Integer gender;

   public Integer getId() {
      return id;
   }

   public void setId(Integer id) {
      this.id = id;
   }

   public String getLastName() {
      return lastName;
   }

   public void setLastName(String lastName) {
      this.lastName = lastName;
   }

   public String getEmail() {
      return email;
   }

   public void setEmail(String email) {
      this.email = email;
   }

   public Integer getGender() {
      return gender;
   }

   public void setGender(Integer gender) {
      this.gender = gender;
   }

   public Employee(Integer id, String lastName, String email, Integer gender) {
      super();
      this.id = id;
      this.lastName = lastName;
      this.email = email;
```

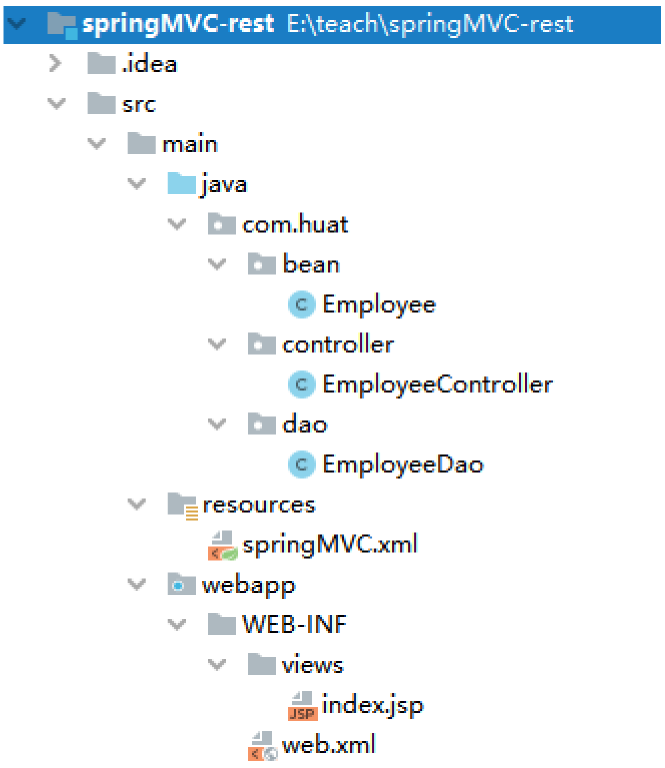
```
        this.gender = gender;
    }

    public Employee() {
    }
}
```

- Create EmployeeDao.java

```
package com.huat.dao;

import java.util.Collection;
import java.util.HashMap;
import java.util.Map;

import com.qixin.mvc.bean.Employee;
import org.springframework.stereotype.Repository;

@Repository
public class EmployeeDao {

    //A map object is used to simulate a database to store employee data
    private static Map<Integer, Employee> employees = null;

    static{
        employees = new HashMap<Integer, Employee>();

        employees.put(1001, new Employee(1001, "E-AA", "aa@163.com", 1));
        employees.put(1002, new Employee(1002, "E-BB", "bb@163.com", 1));
        employees.put(1003, new Employee(1003, "E-CC", "cc@163.com", 0));
        employees.put(1004, new Employee(1004, "E-DD", "dd@163.com", 0));
        employees.put(1005, new Employee(1005, "E-EE", "ee@163.com", 1));
    }

    private static Integer initId = 1006;

    //add employee information
    public void save(Employee employee){
        if(employee.getId() == null){
            employee.setId(initId++);
        }
        employees.put(employee.getId(), employee);
    }

    // get all employees information
    public Collection<Employee> getAll(){
        return employees.values();
    }

    // get one employee by id
    public Employee get(Integer id){
        return employees.get(id);
    }

    // delete one employee by id
    public void delete(Integer id){
        employees.remove(id);
    }
}
```

- Create EmployeeController

```
package com.huat.controller;
@Controller
public class EmployeeController {

    @Autowired
    private EmployeeDao employeeDao;

}
```



```
springMVC-rest E:\teach\springMVC-rest
  .idea
  src
    main
      java
        com.huat
          bean
            Employee
          controller
            EmployeeController
          dao
            EmployeeDao
      resources
        springMVC.xml
      webapp
        WEB-INF
          views
            index.jsp
          web.xml
```

2、Function List

Function	URL	Request method
Visit Home Page √	/	GET
Query all employee data √	/employee	GET
Delete employee by id √	/employee/2	DELETE
jump to add employee page √	/toAdd	GET
save employee data √	/employee	POST
jump to update employee page√	/employee/2	GET
update employee data√	/employee	PUT

3、Specific functions: Visit the home page

a>config view-controller

```
<mvc:view-controller path="/" view-name="index"/>
```

b>create index.jsp page

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Index</title>
</head>
<body>
<h2>Employee Index Page</h2>
<a href="<%=request.getContextPath()%>/employee">query all employees
information</a>
</body>
</html>
```

4、Specific functions: Query all employees information

a>controller method

```
@RequestMapping(value = "/employee", method = RequestMethod.GET)
public String getEmployeeList(Model model){
    Collection<Employee> employeeList = employeeDao.getAll();
    model.addAttribute("employeeList", employeeList);
    return "employee_list";
}
```

b>创建employee_list.html

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
<head>
    <title>Employee Info</title>
</head>
<body>
<table border="1" cellpadding="0" cellspacing="0" style="text-align: center;"
id="dataTable">
    <tr>
        <th colspan="5">Employee Info</th>
    </tr>
    <tr>
        <th>id</th>
        <th>lastName</th>
        <th>email</th>
        <th>gender</th>
        <th>options</th>
    </tr>
    <c:forEach items="${employeeList}" var="employee">
        <tr>
            <td>${employee.id}</td>
            <td>${employee.lastName}</td>
            <td>${employee.email}</td>
            <td>
                <c:if test="${employee.gender==1}">male</c:if>
                <c:if test="${employee.gender==0}">female</c:if>
            </td>
            <td>
                <a href="<%=request.getContextPath()%>/employee/${employee.id}"
class="deleteEmp">delete</a>
                <a href="
<%=request.getContextPath()%>/employee/${employee.id}">update</a>
            </td>
        </tr>
    </c:forEach>
    <tr>
        <td colspan="5">
            <a href="<%=request.getContextPath()%>/toAdd">Add Employee</a>
        </td>
    </tr>
</table>
</body>
</html>
```

5、Specific functions: jump to add employee page

a>config view-controller in springMVC.xml

```
<mvc:view-controller path="/toAdd" view-name="employee_add"></mvc:view-
controller>
```

b>创建employee_add.html

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

<form action="<%=request.getContextPath()%>/employee" method="post">
    lastName:<input type="text" name="lastName"><br>
    email:<input type="text" name="email"><br>
    gender:<input type="radio" name="gender" value="1">male
    <input type="radio" name="gender" value="0">female<br>
    <input type="submit" value="add"><br>
</form>
</body>
</html>
```

6、Specific functions: add employee data

a>controller method

```
@RequestMapping(value = "/employee", method = RequestMethod.POST)
public String addEmployee(Employee employee){
    employeeDao.save(employee);
    return "redirect:/employee";
}
```

7、Specific functions: jump to update employee page

a>modify hyperlink address

```
<a href="<%=request.getContextPath()%>/employee/${employee.id}">update</a>
```

b>controller method

```
@RequestMapping(value = "/employee/{id}", method = RequestMethod.GET)
public String getEmployeeById(@PathVariable("id") Integer id, Model model){
    Employee employee = employeeDao.get(id);
    model.addAttribute("employee", employee);
    return "employee_update";
}
```

c>create employee_update.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
<head>
    <title>Update Employee</title>
</head>
<body>

<form action="<%=request.getContextPath()%>/employee" method="post">
    <input type="hidden" name="_method" value="put">
    <input type="hidden" name="id" value="${employee.id}">
    lastName:<input type="text" name="lastName" value="${employee.lastName}">
<br>
    email:<input type="text" name="email" value="${employee.email}"><br>
    gender:<input name="gender" value="1" <c:if
test="${employee.gender==1}">checked</c:if> type="radio">male
    <input name="gender" value="0" <c:if
test="${employee.gender==0}">checked</c:if> type="radio">female<br>
    <input type="submit" value="update"><br>
</form>
</body>
</html>
```

8、Specific functions: update employee data

a>controller method

```
@RequestMapping(value = "/employee", method = RequestMethod.PUT)
public String updateEmployee(Employee employee){
    employeeDao.save(employee);
    return "redirect:/employee";
}
```

9、Specific functions: Delete employee information by id

a>register HiddenHttpMethodFilter in web.xml configuration file

```
<filter>
    <filter-name>HiddenHttpMethodFilter</filter-name>
    <filter-class>org.springframework.web.filter.HiddenHttpMethodFilter</filter-
class>
</filter>
<filter-mapping>
    <filter-name>HiddenHttpMethodFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

b>Create the form to handle delete requests

```
employee_list.jsp
<!-- What forms do: It uses a hyperlink to control form submission and convert
post requests to delete requests -->
<form id="delete_form" method="post">
    <!-- The HiddenHttpMethodFilter requires that the _method request parameter
must be passed and that its value is the final request method -->
    <input type="hidden" name="_method" value="delete"/>
</form>
```

b>删除超链接绑定点击事件 Delete the hyperlink binding click event

Delete hyperlink

```
<a href="<%=request.getContextPath()%>/employee/${employee.id}"
class="deleteEmp">delete</a>
```

在webapp目录下创建js目录，引入jquery.js, 通过js处理点击事件

create js folder in webapp folder, import jquery.js , handle click events with js

```
<script type="text/javascript" src="<%=request.getContextPath()%>/js/jquery.js">
</script>
```

```
<script type="text/javascript">
    //delete operation
    $(document).on("click", ".deleteEmp", function () {
        var tip = "Are you confirm want to delete the current employee's data?";
        if (confirm(tip)) {
            var href = $(this).attr("href");
            $("#delete_form").attr("action", href).submit();
        }
        //prevent the default behavior of hyperlinks
        return false;
    });
</script>
```

c>add controller method

```
@RequestMapping(value = "/employee/{id}", method = RequestMethod.DELETE)
public String deleteEmployee(@PathVariable("id") Integer id){
    employeeDao.delete(id);
    return "redirect:/employee";
}
```

八、HttpMessageConverter

HttpMessageConverter的作用是将请求报文转换为Java对象，或将Java对象转换为响应报文
HttpMessageConverter is used to convert request packets into Java objects or Java objects into response packets

HttpMessageConverter provide two annotations and two entity class：@RequestBody，@ResponseBody，RequestEntity，ResponseEntity

create new project named springMVC-demo4

1、@RequestBody

@RequestBody可以获取请求体，需要在控制器方法设置一个形参，使用@RequestBody进行标识，当前请求的请求体就会为当前注解所标识的形参赋值
The @RequestBody annotation can be used to retrieve the request body by setting a parameter with @RequestBody in the controller method, and the current request body will be assigned to the parameter identified by the annotation

```
index.jsp
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Index</title>
</head>
<body>
<h2>Index</h2>
<form action="<%=request.getContextPath()%>/testRequestBody" method="post">
  username<input type="text" name="username">
  password<input type="password" name="password">
  <input type="submit" value="Test @RequestBody">
</form>
```

```
</body>
</html>
```

```
package com.qixin.mvc.controller;

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

@Controller
public class HttpController {
    @PostMapping("/testRequestBody")
    public String testRequestBody(@RequestBody String requestBody){
        System.out.println("requestBody:"+requestBody);
        return "success";
    }
}
```

输出结果：

requestBody:username=admin&password=123456

2、RequestEntity

RequestEntity封装请求报文的一种类型，需要在控制器方法的形参中设置该类型的形参，当前请求的请求报文就会赋值给该形参，可以通过getHeaders()获取请求头信息，通过getBody()获取请求体信息

RequestEntity encapsulates a type of request packet, which is set as a parameter in the controller method and will be used for the current request.You can get the header information from getHeaders() and the body information from getBody()

```
index.jsp
<form action="<%=request.getContextPath()%>/testRequestEntity" method="post">
    username<input type="text" name="username">
    password<input type="password" name="password">
    <input type="submit" value="Test RequestEntity">
</form>
```

```
@PostMapping("/testRequestEntity")
public String testRequestEntity(RequestEntity<String> requestEntity){
    System.out.println("requestHeader:"+requestEntity.getHeaders());
    System.out.println("requestBody:"+requestEntity.getBody());
    System.out.println("requestUrl:" + requestEntity.getUrl());
    System.out.println("requestMethod:" + requestEntity.getMethod());
    return "success";
}
```

输出结果：
requestHeader:[host:"localhost", connection:"keep-alive", content-leng"27", cache-control:"max-age=0", sec-ch-ua:""Google Chrome";v="105", "Not)A;Brand";v="8", "Chromium";v="105"", sec-ch-ua-mobile:"?0", sec-ch-ua-platform:""Windows"", upgrade-insecure-requests:"1", origin:"http://localhost", user-agent:"Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/105.0.0.0 Safari/537.36", accept:"text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,/;q=0.8,application/signed-exchange;v=b3;q=0.9", sec-fetch-site:"same-origin", sec-fetch-

mode:"navigate", sec-fetch-user:"?1", sec-fetch-dest:"document", referer:"http://localhost/springMVC/", accept-encoding:"gzip, deflate, br", accept-language:"zh-CN,zh;q=0.9", cookie:"JSESSIONID=89CD5420911D58724120F9D485C6279F", Content-Type:"application/x-www-form-urlencoded;charset=UTF-8"]
requestBody:username=admin&password=123

3、@ResponseBody (*)

通过servletAPI的response对象响应浏览器数据
Before we learn @ResponseBody annotation , let's see how to use servletAPI to response data to browser .

```
<a href="<%=request.getContextPath()%>/testResponse">response to browser data via servletAPI's Response object</a><br>
```

```
@RequestMapping("/testResponse")
public void testResponse(HttpServletResponse response) throws IOException {
    response.getWriter().write("hello,response");
}
```

@ResponseBody用于标识一个控制器方法，可以将该方法的返回值直接作为响应报文的响应体响应到浏览器
@ResponseBody is used to identify a controller method that can be sent return value to the browser as the response body.

```
<a href="<%=request.getContextPath()%>/testResponseBody">Responding to browser data with @ResponseBody annotation</a><br>
```

```
@RequestMapping("/testResponseBody")
@ResponseBody
public String testResponseBody(){
    return "testResponseBody";
}
```

Result: The browser page shows testResponseBody

4、SpringMVC process json

JSON（JavaScript Object Notation）是一种轻量级的数据交换格式。
JSON is a lightweight data interchange form.

define a User class

```
package com.huat.bean;
public class User {
    private Integer id;
    private String username;
    private String password;
    private Integer age;
    private String sex;
    ...constructor method
    ...getter and settter method
}
```

Steps to process json：

a>import jackson dependencies

```
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.12.1</version>
</dependency>
```

b>enable annotation driven

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

在SpringMVC的核心配置文件中开启mvc的注解驱动，此时在HandlerAdaptor中会自动装配一个消息转换器：MappingJackson2HttpMessageConverter，可以将响应到浏览器的Java对象转换为Json格式的字符串
When you enable annotation driven in the SpringMVC configuration file, MappingJackson2HttpMessageConverter message transformer is automatically assembled in HandlerAdaptor, Java object can be converted to Json format string and response to the browser .

c>将Java对象直接作为控制器方法的返回值返回，就会自动转换为Json格式的字符串
When a Java object as the return value of a controller method, it is automatically converted to a Json string

```
index.jsp
<a href="<%=request.getContextPath()%>/testResponseUser">SpringMVC process
json</a><br>
```

```
@RequestMapping("/testResponseUser")
@ResponseBody
public User testResponseUser(){
    return new User(1001,"admin","123456",23,"male");
}
```

The browser page shows the result:

{"id":1001,"username":"admin","password":"123456","age":23,"sex":"男"}

5、@RestController

@RestController注解是springMVC提供的一个复合注解，标识在控制器的类上，就相当于为类添加了@Controller注解，并且为其中的每个方法添加了@ResponseBody注解
The @RestController annotation is a composite annotation provided by springMVC that identifies the controller class, equals to annotating the class with @Controller and each method with @ResponseBody

6、ResponseEntity

ResponseEntity用于控制器方法的返回值类型，该控制器方法的返回值就是响应到浏览器的响应报文
ResponseEntity is used as the return type of the controller method that returns the response packet to the browser

九、File download and upload

1、File download

add view-controller in springMVC.xml configuration file

```
<mvc:view-controller path="/file" view-name="file"></mvc:view-controller>
```

file download page file.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Test file download and upload</title>
</head>
<body>
<a href="<%=request.getContextPath()%>/testDown">download default.png</a>
</body>
</html>
```

Use ResponseEntity to download files

```
@Controller
public class FileUpAndDownloadContorller {
    @RequestMapping("/testDown")
    public ResponseEntity<byte[]> testResponseEntity(HttpSession session) throws
IOException {
        // get ServletContext object
        ServletContext servletContext = session.getServletContext();
        // gets the real path of the file on the server
        String realPath = servletContext.getRealPath("/img/default.png");
        //create inputstream
        InputStream is = new FileInputStream(realPath);
        // Creating byte arrays
        byte[] bytes = new byte[is.available()];
        // Read stream into byte array
        is.read(bytes);
        // Create an HttpHeaders object to set the response headers
        MultiValueMap<String, String> headers = new HttpHeaders();
        //Set the method to download and the name of the file to download
        headers.add("Content-Disposition", "attachment;filename=default.png");
        //Sets the http response status code
        HttpStatus statusCode = HttpStatus.OK;
        //create ResponseEntity object
        ResponseEntity<byte[]> responseEntity = new ResponseEntity<>(bytes,
headers, statusCode);
        //close inputstream
        is.close();
        return responseEntity;
    }
}
```

2、File upload

文件上传要求form表单的请求方式必须为post，并且添加属性enctype="multipart/form-data"
The form request must be post and add the enctype property with value "multipart/form-data".

SpringMVC中将上传的文件封装到MultipartFile对象中，通过此对象可以获取文件的相关信息
SpringMVC encapsulates the uploaded file in a MultipartFile object, which can be used to get information about the file

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
  <title>Test file download and upload</title>
</head>
<body>
<a href="<%=request.getContextPath()%>/testDown">download default.png</a>
<form action="<%=request.getContextPath()%>/testUp" method="post"
enctype="multipart/form-data">
  photo:<input type="file" name="photo"><br>
  <input type="submit" value="upload">
</form>
</body>
</html>
```

the step of upload:

a>import dependencies:

```
<!-- https://mvnrepository.com/artifact/commons-fileupload/commons-fileupload --
>
<dependency>
    <groupId>commons-fileupload</groupId>
    <artifactId>commons-fileupload</artifactId>
    <version>1.3.1</version>
</dependency>
```

b>Add the configuration in SpringMVC.xml

```
<!--The file must be parsed by the file parser to convert the file into a
MultipartFile object-->
<bean id="multipartResolver"
class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
</bean>
```

> 注意：id属性非常重要，必须设置为multipartResolver，否则，springMVC无法加载识别bean。
> Note: The id property is very important and must be set to multipartResolver; otherwise, springMVC won't be able to load this bean.

c>controller method:

```
@PostMapping("/testUp")
    public String testUp(MultipartFile photo, HttpSession session, Model model)
throws IOException {
        //Gets the filename of the uploaded file
        String fileName = photo.getOriginalFilename();
        //Handle filename duplication issues
        String suffix = fileName.substring(fileName.lastIndexOf("."));
        fileName = UUID.randomUUID().toString() + suffix;
        //Gets the path to the server directory where the images are stored
        ServletContext servletContext = session.getServletContext();
        String photoPath = servletContext.getRealPath("img");
        File file = new File(photoPath);
        if(!file.exists()){
            file.mkdir();
        }
        String finalPath = photoPath + File.separator + fileName;
        model.addAttribute("path", "img/" + fileName);
        //upload pictures
        photo.transferTo(new File(finalPath));
        return "success";
    }
```

d>modify success.jsp to display the upload image.

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>success</title>
</head>
<body>
<h2>success</h2>
<img src="${path}" alt="photo">
</body>
</html>
```

十、拦截器 Interceptor

1、拦截器的配置 Interceptor configuration

SpringMVC中的拦截器用于拦截控制器方法的执行
Interceptors in SpringMVC are used to intercept the execution of controller methods

SpringMVC中的拦截器需要实现HandlerInterceptor接口
Interceptors in SpringMVC need to implement the HandlerInterceptor interface

```
public class FirstInterceptor implements HandlerInterceptor {
    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse
response, Object handler) throws Exception {
        System.out.println("FirstInterceptor-->preHandle");
        //false表示拦截 false indicates Intercept the request.
        //true表示放行 true indicates release the request
        return false;
    }

    @Override
    public void postHandle(HttpServletRequest request, HttpServletResponse
response, Object handler, ModelAndView modelAndView) throws Exception {
        System.out.println("FirstInterceptor-->postHandle");
    }

    @Override
    public void afterCompletion(HttpServletRequest request, HttpServletResponse
response, Object handler, Exception ex) throws Exception {
        System.out.println("FirstInterceptor-->afterCompletion");
    }
}
```

SpringMVC的拦截器必须在SpringMVC的配置文件中进行配置:
SpringMVC interceptors must be configured in the SpringMVC.xml:

```
<mvc:interceptors>
    <mvc:interceptor>
        <mvc:mapping path="/*"/>
        <mvc:exclude-mapping path="/"/>
        <bean class="com.huat.interceptor.FirstInterceptor"/>
    </mvc:interceptor>
</mvc:interceptors>
<!--
    以上配置方式可以通过ref或bean标签设置拦截器，通过mvc:mapping设置需要拦截的请求，通过
mvc:exclude-mapping设置需要排除的请求，即不需要拦截的请求
This configuration allows you to set the interceptor via ref or bean tags,
mvc:mapping sets the requests to intercept
mvc:exclude-mapping sets requests to be excluded, that is, requests that do not
need to be intercepted
-->
```

Create new InterceptorController

```
@Controller
public class InterceptorController {

    @RequestMapping("/testInterceptor")
    public String testInterceptor() {
        return "success";
    }
}
```

2、拦截器的三个抽象方法 The three abstract methods in the interceptor

SpringMVC中的拦截器有三个抽象方法:
Interceptors in SpringMVC have three abstract methods:

- preHandle：控制器方法执行之前执行preHandle()，其boolean类型的返回值表示是否拦截或放行，返回true为放行，即调用控制器方法；返回false表示拦截，即不调用控制器方法
  The preHandle() method is executed before the controller method is executed, and its boolean return value indicates whether to intercept or release, returning true to release the request, that is, calling the controller method. Returning false indicates intercept the request, that is, not calling the controller method
- postHandle：控制器方法执行之后执行postHandle()
  The postHandle() method is executed after the controller method executed.

- afterComplation：处理完视图和模型数据，渲染视图完毕之后执行afterComplation()
  afterComplation: After processing view and model data, execute afterComplation() method after rendering the view page.

十一、异常处理器 Exception handler

SpringMVC提供了一个处理控制器方法执行过程中所出现的异常的接口：HandlerExceptionResolver
SpringMVC provides an interface to handle exceptions that occur during the execution of controller methods: HandlerExceptionResolver

HandlerExceptionResolver接口的实现类有：DefaultHandlerExceptionResolver和SimpleMappingExceptionResolver
HandlerExceptionResolver interface has two implementation class: DefaultHandlerExceptionResolver and SimpleMappingExceptionResolver

SpringMVC提供了自定义的异常处理器SimpleMappingExceptionResolver，使用方式有以下两种：
For SpringMVC provides a custom exception handler SimpleMappingExceptionResolver, There are two ways to use it:

1、基于配置的异常处理 exception handling baseon configuration file

```xml
<bean
class="org.springframework.web.servlet.handler.SimpleMappingExceptionResolver">
        <property name="exceptionMappings">
            <props>
                <!--
                    key Indicates the type of the exception
                    value indicates that if the specified exception occurs, the
system switches to the specified page
                -->
                <prop key="java.lang.ArithmeticException">error</prop>
                <prop key="java.lang.NullPointerException">error</prop>
            </props>
        </property>
        <!--
            exceptionAttribute属性设置一个属性名，将出现的异常信息在请求域中进行共享
            exceptionAttribute Sets an attribute name to share exception
information in the request domain
        -->
        <property name="exceptionAttribute" value="ex"></property>
    </bean>
```

```java
@Controller
public class TestController {
    @RequestMapping("/testExceptionHandler")
    public String testExceptionHandler() {
        int n = (int)(Math.random()*2+1);
        System.out.println(n);
        if(n==1) {
            String s = null;
            System.out.println(s.length());
        } else {
            System.out.println(1/0);
        }
        return "success";
    }
}
```

index.jsp

```
<a href="<%=request.getContextPath()%>/testExceptionHandler">Test
ExceptionHandler</a>
```

error.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>error</title>
</head>
<body>
<h2>error</h2>
<p>${ex}</p>
</body>
</html>
```

When an exception occurs, it is forward to the exception handling page error.jsp

2、基于注解的异常处理 exception handling baseon annotation

```
package com.huat.controller;
//@ControllerAdvice将当前类标识为异常处理的组件
//@ControllerAdvice identifies the current class as the exception handling
component
@ControllerAdvice
public class ExceptionController {

    //@ExceptionHandler用于设置所标识方法处理的异常
    //The @ExceptionHandler is used to set the exception handled by the
identified method
    @ExceptionHandler(value = {ArithmeticException.class,
NullPointerException.class})
    //ex表示当前请求处理中出现的异常对象
    //ex represents the exception object present in the current request
processing
    public String handleException(Exception ex, Model model){
        model.addAttribute("ex", ex);
        return "error";
    }

}
```