开发一个使用SpringMVC注解web项目实例

创建一个空工程idea-maven-springmvc，在该工程中创建web项目

需求：
    用户在页面发起一个请求，请求间接交给SpringMVC的控制器对象，并显示请求的处理结果（转到一个欢迎页面）

步骤：
    1、--新建一个maven工程，模板选择webapp

    2、--加入依赖：
    spring-webmvc依赖，当加入该依赖时，会间接的把Spring的依赖都加入到该项目中
    jsp和servlet依赖

    3、--重点：
    在web.xml配置文件中注册SpringMVC框架的核心对象DispatherServlet，就像我们之前在该xml文件中注册Servlet实
    现类对象一样，但是这个对象跟Spring其他相关操作是有关联的，很重要。
    1）、DispatherServlet叫做中央调度器，是一个servlet对象，其父类是HttpServlet
    2）、DispatherServlet也叫做前端控制器（front controller）
    3）、负责接收用户请求，根据请求的要求来调用其他相应的控制器对象，并把请求的处理结果显示给用户
    --该对象是SpringMVC框架做web开发的标准

    4、--创建请求页面index.jsp

    5、创建控制器类：
        1）、在类的上面添加@Controller（一般不需要给注解的value属性赋值，用默认的），让SpringMVC容器创建对
        象，存放到SpringMVC容器中
        2）、在类中的方法上面添加注解@RequestMapping。该注解是调用该方法的资源路径映射注解，注解中的属性是字
        符串，值填的是路径，请求发送过来，这边处理，选择用控制器中的什么方法处理，要看请求路径对应的是控制器对
        象中方法上@RequestMapping中填写的路径

    6、创建一个jsp页面作为请求的处理结果显示

    7、创建SpringMVC配置文件（和Spring配置文件一样）
        1）、声明组件扫描器，指定@Controller注解所在包名，如果还有@Component、@Service等注解也要指定
        2）、声明视图解析器。帮助处理视图的

1）、创建web项目a-springmvc-web-01，其中创建java源文件目录和resources资源目录。在pom文件中添加依赖

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

<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.studymyself</groupId>
  <artifactId>a-springmvc-web-01</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>war</packaging>


  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
  </properties>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.11</version>
      <scope>test</scope>
    </dependency>

<!--    servlet依赖-->
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>javax.servlet-api</artifactId>
      <version>4.0.1</version>
      <scope>provided</scope>
    </dependency>

<!--    SpringMVC依赖-->
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-webmvc</artifactId>
      <version>5.2.5.RELEASE</version>
    </dependency>

  </dependencies>

  <build>

  </build>
</project>
```

2）、在web.xml文件中声明注册中央调度器DispatcherServlet

```
<?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">

    <!--注册SpringMVC的核心对象DispatcherServlet
```

```
        这样web容器启动后就会创建这里声明的所有Servlet对象，这里是创建DispatcherServlet对象实例
        原因：
            DispatcherServlet对象在被服务器创建的过程中，同时创建SpringMVC容器对象，spring容器对象的创建
            会读取SpringMVC的配置文件，同时就把文件中声明的bean全部创建好了，到时用户发送请求就可以直接调用
            容器中的对象就好了。就跟前面的监听器一样。
        原理：
        servlet初始化执行init()方法。DispatcherServlet类中的init方法有如下内容{
            //创建web项目的Spring容器
            WebApplicationContext wac = new ClassPathXmlApplicationContext("springmvc.xml");
            //把容器对象放到ServletContext对象中
            getServletContext().setAttribute(key,wac);
        }
        -->
    <servlet>
        <servlet-name>springMVC</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

        <!--跟Spring中的监听器一样，在创建中央调度器对象的时候为我们创建SpringMVC
            容器对象时读取的spring配置文件默认读取路径是/WEB-INF/springMVC-springmvc.xml
            所以我们要修改读取路径,记得配置文件中这些标签的添加顺序，不然会报错-->
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>classpath:springmvc.xml</param-value>
        </init-param>

        <!--我们希望在Tomcat启动后创建Servlet对象
            使用load-on-startup标签：表示Tomcat服务器启动后创建Servlet的对象的顺序
            其中的值是整数，单位越小越快创建这个声明的Servlet对象，大于等于0的整数
        -->
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>springMVC</servlet-name>
        <!--在使用框架时，url-pattern标签中可以使用两种值

            1、使用扩展名方式，语法*.xxx，其中xxx是自定义的扩展名。常用*.do,*.action,*.mvc等
            意思是以.do结尾的请求路径的请求都交给名称为<servlet-name>springMVC</servlet-name>
            这个servlet来处理,如
            http://localhost:8080/JD/some.do
            http://localhost:8080/JD/other.do

            2、第二种就是使用斜杠"/"
            -->
        <url-pattern>*.do</url-pattern>
    </servlet-mapping>
</web-app>
```

3）、创建一个欢迎页面index.jsp

```
<%--
  Created by IntelliJ IDEA.
  User: 钟荣杰
  Date: 2021/2/14
  Time: 19:23
  To change this template use File | Settings | File Templates.
--%>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Welcome Page</title>
</head>
<body>
    <P>SpringMVC的一个项目</P>
    <!--这里添加一个超链接，转到web中声明的接收.do结尾的中央调度器，
        然后调度器根据some.do和扫描@Controller中的值，调用与值对应控制器对象，完成处理-->
    <a href="some.do">发起some.do请求</a>
</body>
</html>
```

4）、创建控制器类

```
package com.studymyself.controller;

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

/**
 * @Controller:创建处理器对象，然后放到SpringMVC容器中
 */
@Controller
public class MyController {

    /**
     * 里面的方法处理用户提交的请求
     * SpringMVC中的方法是自定义的，可以有多种返回值，多种参数，自定义名称
     */

    /**
     * 这里使用doSome方法处理xxx/some.do请求
     * 如何定位到这个方法呢？
     * @RequestMapping：请求映射，作用是吧一个请求的URL地址和一个方法绑定到一块
     *                      一个请求指定一个方法来处理请求
     *         属性：
     *         value：是String类型的数组，让多个请求共用一个处理方法
     *                填的值是请求的URL地址（这里是some.do）
     *                该值必须是唯一的，不能重复。使用时推荐以"/"开头的地址
     *         在方法上面使用，也可以在类上面使用
     *  使用@RequestMapping修饰的方法叫做处理器方法或者控制器方法，当于我们在doGet或者
doPost方法
     *  中编写处理代码
     *
     *  返回值：ModelAndView类型，表示本次请求的处理结果，是Spring框架中的一个类
     *  Model：表示可以保存数据，请求处理完成后，显示给用户看的
     *  View：视图，如jsp等
```

```
     *
     */
    //@RequestMapping(value = {"/some.do","first.do"})
    @RequestMapping(value = "/some.do")
    public ModelAndView doSome(){   //相当于doGet/doPost方法

        //这里语句相当于调用service中的方法

        ModelAndView modelAndView = new ModelAndView();
        //然后把返回结果存储到ModelAndView对象中,该对象是键值对的map集合
        modelAndView.addObject("msg","SpringMVC的第一个实例项目");
        modelAndView.addObject("fun","非常高兴认识您");

        //下面指定视图。指定的是完整的路径，框架底层会对其进行forward的操作
//        modelAndView.setViewName("WEB-INF/view/show.jsp");
//        modelAndView.setViewName("WEB-INF/view/other.jsp");

        //当声明配置视图解析器后，直接使用文件逻辑名指定视图
        //原理是框架会使用配置的视图解析器的前缀属性值+逻辑名称+后缀属性值，字符串拼接组成路径
        //  /WEB-INF/view/+show+.jsp
        modelAndView.setViewName("show");
        modelAndView.setViewName("other");

        return modelAndView;
        //这里执行 return modelAndView; 语句后，SpringMVC框架底层执行如下：
        /*
            把执行addObject方法中的得到的数据放入到Request作用域中
            request.setAttribute("msg","SpringMVC的第一个实例项目");
            request.setAttribute("fun","非常高兴认识您");

            把执行setViewName的方法在底层执行如下重定向语句：
            request.getRequestDispatcher("/show.jsp").forward(request,response);
         */

    }
}
```

5）、创建show.jsp，作为结果展示页面

```
<%--
  Created by IntelliJ IDEA.
  User: 钟荣杰
  Date: 2021/2/14
  Time: 21:45
  To change this template use File | Settings | File Templates.
--%>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Result Page</title>
</head>
<body>
    <h3>结果展示</h3>
    <!--把结果取出来，通过EL表达式${key值}来取值-->
    <h3>msg数据：${msg}</h3>
```

```
    <h3>fun数据：${fun}</h3>
</body>
</html>
```

6）、在SpringMVC中配置组件扫描器，由于没有声明组件扫描器就开始启动服务器，导致控制器类中的注解没有扫描到，没有创建控制器对象和请求路径映射。使得点击超链接一直访问不到资源。

```
<?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"
       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">
    <!--声明组件扫描器-->
    <context:component-scan base-package="com.studymyself.controller"/>
</beans>
```

启动服务器，结果如下：


localhost:8080/a_springmvc_web_01/
应用
百度
淘宝
京东
天猫
苏宁易购
杂七杂八
tensorflow
SpringMVC的一个项目
发起some.do请求
点击，超链接地址如下
localhost:8080/a_springmvc_web_01/some.do

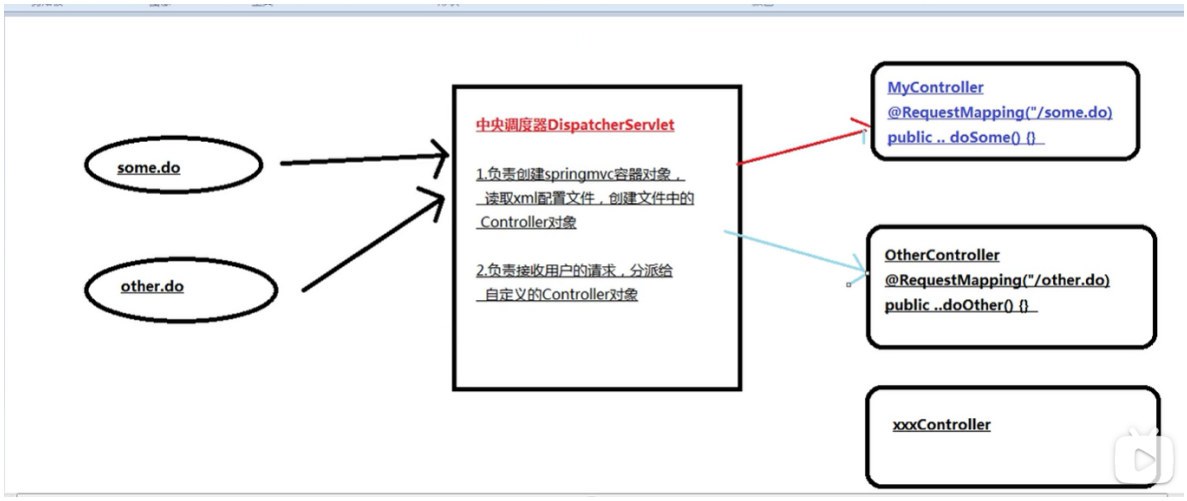
结果展示

msg数据：SpringMVC的第一个实例项目

fun数据：非常高兴认识您

整个执行流程如下：
1、用户发起some.do的请求给Tomcat
2、Tomcat接收请求（通过web.xml文件中的url-pattern标签知道*.do的请求交给DispatcherServlet来接收处理）
3、DispatherServlet接收请求（因为DispatherServlet对象创建的时候同时帮助创建SpringMVC容器和容器中的对象，读取了springmvc.xml，而配置文件中声明了组件扫描器，扫描到了指定路径包下的对象中的注解，所以DispatherServlet接收请求后知道some.do对应doSome()方法上面的@RequestMapping中的值对应，直接获取MyController对象调用doSome方法处理请求）
4、DispatherServlet根据some.do调用MyController.doSome()
5、框架执行doSome方法得到ModelAndView对象，把该对象进行处理，最后转发到show.jsp

中央调度器把Tomcat发给它的请求通过(@Controller和@RequestMapping)路径映射，来判断这个请求应该调用哪个Controller类中的哪个处理器方法



--SpringMVC执行过程源代码分析
1、Tomcat启动，创建容器的过程
Tomcat启动后通过标签load-on-startup指定的1值，创建了DispatherServlet对象。DispatherServlet的父类FrameworkServlet继承了HttpServlet类，所以其本质是个servlet，所以被创建时执行了init()。
而在init方法中有创建SpringMVC容器对象的代码：

```
//创建web项目的Spring容器
WebApplicationContext wac = new ClassPathXmlApplicationContext("springmvc.xml");
//把容器对象放到ServletContext对象中
getServletContext().setAttribute(key,wac);
```

而上面创建SpringMVC容器对象时启动组件扫描器：创建@Controller标注的的类对象，创建了MyController对象，然后把这个对象放入到SpringMVC容器中，容器是map，底层执行map.put("myController",MyController对象)

2、请求处理过程分析

7）、补充的内容：配置视图解析器

当用户知道我们结果要跳转到show.jsp页面，他直接输入请求
http://localhost:8080/a_springmvc_web_01/show.jsp
这样显示的结果是没有任何数据的，因为没有经过Controller中的方法进行处理

那么我们要让用户按照我们的规则进行请求路径的跳转，即：
先到欢迎页面：
http://localhost:8080/a_springmvc_web_01/或
http://localhost:8080/a_springmvc_web_01/index.jsp
然后点击其中的超链接，路径为：http://localhost:8080/a_springmvc_web_01/some.do
最后跳转到：http://localhost:8080/a_springmvc_web_01/show.jsp

设计规则，不按照规定直接到某一页面就不显示，我们可以把webapp目录下的show.jsp放到该目录下的WEB-INF目录下，页面可能很多，那么可以再创建一个目录view专门存放页面，因为WEB-INF目录下的资源是不对外直接开放的，所以直接访问是报404的。

此时控制器对象中的方法中需要转发到该页面的路径就需要改为：

modelAndView.setViewName("WEB-INF/view/show.jsp");

当页面很多时，路径和后缀都是相同的，只有名字不同，这样一个一个写就很繁琐，所以框架可以帮助我们简化写法，需要在SpringMVC配置文件中声明配置视图解析器

```
<?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"
       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">
    <!--声明组件扫描器-->
    <context:component-scan base-package="com.studymyself.controller"/>

    <!--声明配置视图解析器,框架提供的一个类-->
    <bean
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <!--前缀属性：视图文件的路径，前后要加/，前一个/表示项目根路径：
http://localhost:8080/a_springmvc_web_01-->
        <property name="prefix" value="/WEB-INF/view/"/>
        <!--后缀属性：视图文件的类型-->
        <property name="suffix" value=".jsp"/>
    </bean>
</beans>
```

8）、当我们的Cotroller类中方法不止一个，需要在映射路径中添加某个模块的名称，需要我们在这个类中每一个方法的映射路径都添加同一个模块名，很繁琐，所以我们可以在Controller类的类名上面添加@RequestMapping，其中value的值填的是模块名字符串，作为该类所有方法的公共路径。

新建一个OtherController类，如下

```
package com.studymyself.controller;

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

@Controller
/**
 * @RequestMapping注解添加到类上面：
 * 作用是定义Controller类中的所有方法的公共映射路径，项目的模块名
 */
@RequestMapping(value = "/test")
public class OtherController {

    @RequestMapping(value = "/otherOne.do")
    public ModelAndView doOtherOne() {   //相当于doGet/doPost方法
```

```
        //这里语句相当于调用service中的方法

        ModelAndView modelAndView = new ModelAndView();
        //然后把返回结果存储到ModelAndView对象中,该对象是键值对的map集合
        modelAndView.addObject("msg", "SpringMVC的第一个实例项目");
        modelAndView.addObject("fun", "非常高兴认识您");

        modelAndView.setViewName("show");
        modelAndView.setViewName("other");

        return modelAndView;

    }

    @RequestMapping(value = "/otherTwo.do")
    public ModelAndView doOtherTwo(){   //相当于doGet/doPost方法

        //这里语句相当于调用service中的方法

        ModelAndView modelAndView = new ModelAndView();
        //然后把返回结果存储到ModelAndView对象中,该对象是键值对的map集合
        modelAndView.addObject("msg","SpringMVC的第一个实例项目");
        modelAndView.addObject("fun","非常高兴认识您");

        modelAndView.setViewName("show");
        modelAndView.setViewName("other");

        return modelAndView;

    }

}
```

9）、设置方法需要的请求方式

需要设置的是@RequestMapping注解中的method属性，具体如下

```
package com.studymyself.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.servlet.ModelAndView;

@Controller
/**
 * @RequestMapping注解添加到类上面：
 * 作用是定义Controller类中的所有方法的公共映射路径，项目的模块名
 */
@RequestMapping(value = "/test")
public class OtherController {

    /**
```

```
     * @RequestMapping的method属性：
     *       表示的是设置请求的方式，一旦前端请求方式不符合，就无法调用该方法，报405错误
     *       值是RequestMethod枚举类，如要设置为GET请求访问，RequestMethod.GET,post就是
     *       RequestMethod.POST，具体的可以自己到该枚举类中查看
     *       不设置就可以是处理任意类型的请求
     */

    //指定doOtherOne()只能处理GET请求
    @RequestMapping(value = "/otherOne.do",method = RequestMethod.GET)
    public ModelAndView doOtherOne() {   //相当于doGet/doPost方法

        //这里语句相当于调用service中的方法

        ModelAndView modelAndView = new ModelAndView();
        //然后把返回结果存储到ModelAndView对象中,该对象是键值对的map集合
        modelAndView.addObject("msg", "SpringMVC的第一个实例项目");
        modelAndView.addObject("fun", "非常高兴认识您");

        modelAndView.setViewName("show");
        modelAndView.setViewName("other");

        return modelAndView;

    }

    //指定doOtherTwo()只能处理POST请求
    @RequestMapping(value = "/otherTwo.do",method = RequestMethod.POST)
    public ModelAndView doOtherTwo(){   //相当于doGet/doPost方法

        //这里语句相当于调用service中的方法

        ModelAndView modelAndView = new ModelAndView();
        //然后把返回结果存储到ModelAndView对象中,该对象是键值对的map集合
        modelAndView.addObject("msg","SpringMVC的第一个实例项目");
        modelAndView.addObject("fun","非常高兴认识您");

        modelAndView.setViewName("show");
        modelAndView.setViewName("other");

        return modelAndView;

    }

}
```