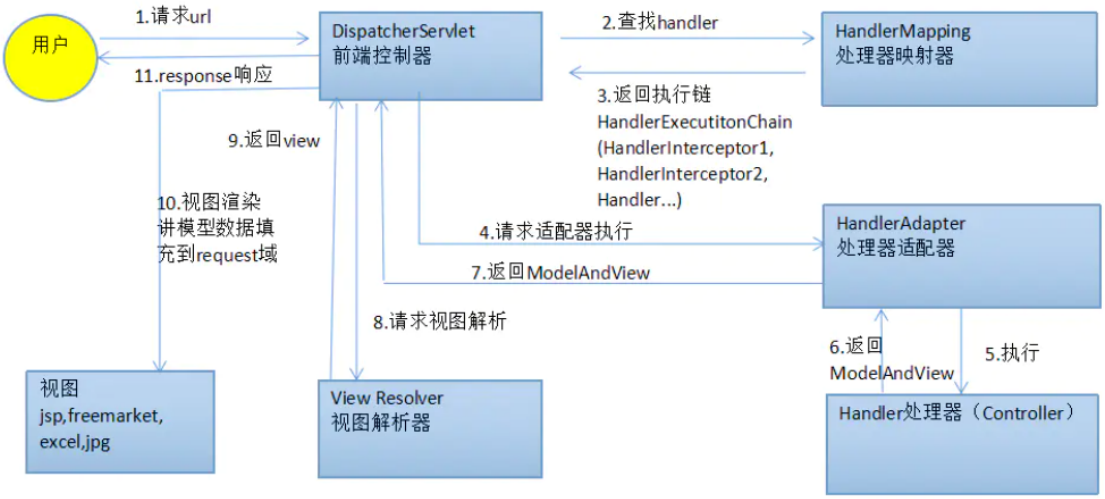
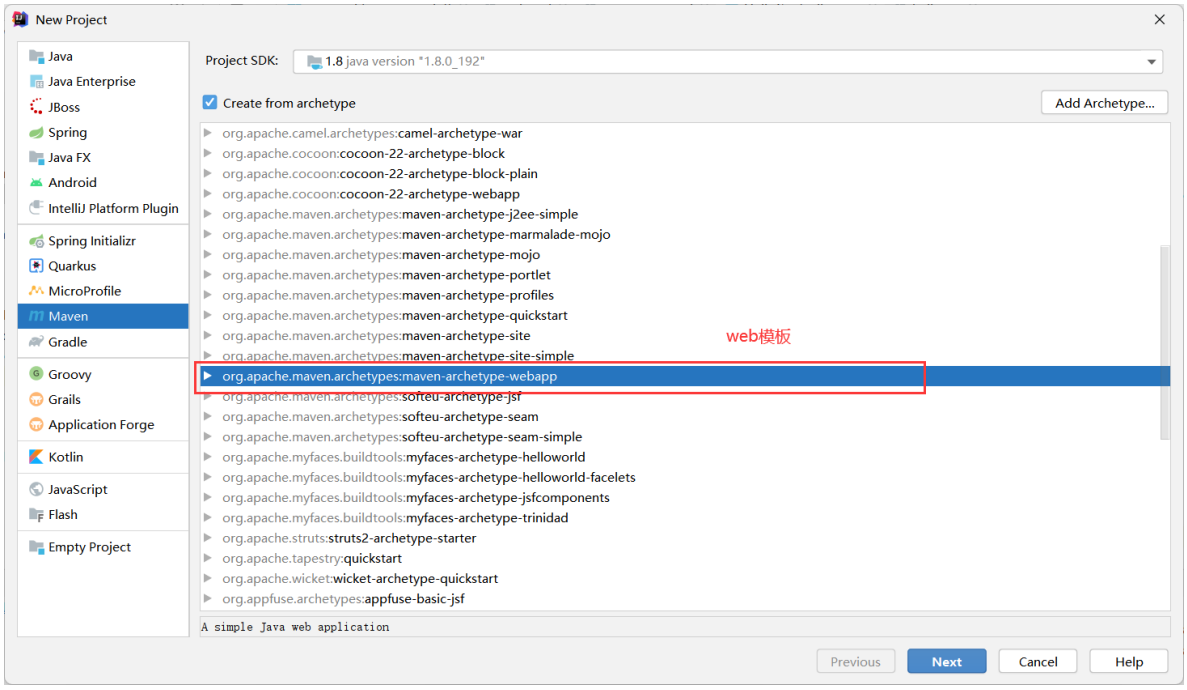
Maven 搭建原生 SpringMVC 的方式

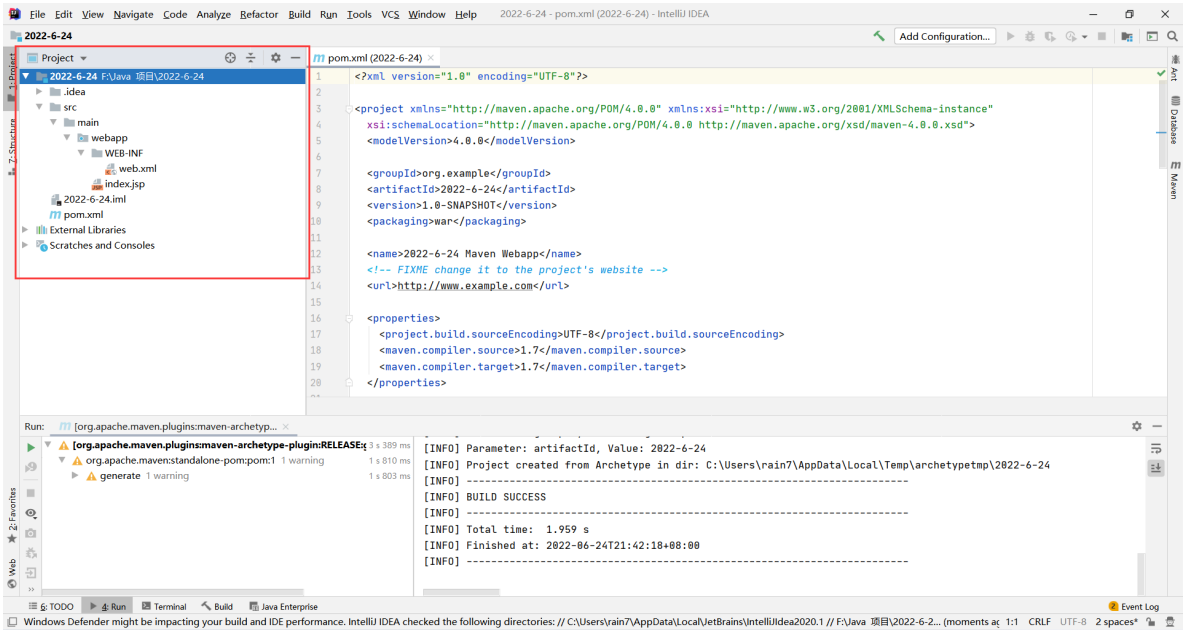
使用原生的方式，更加熟悉具体的MVC框架的执行流程



（1）使用maven创建web项目（使用web模板）



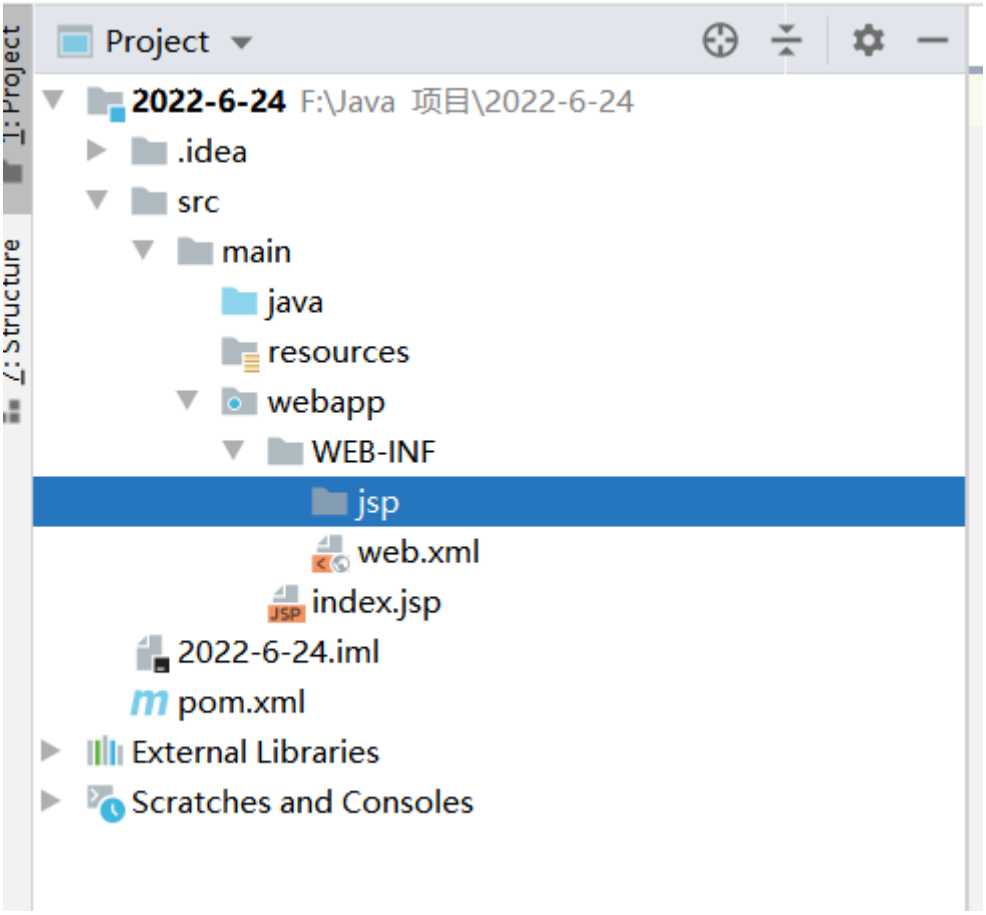
创建好项目的显示界面



（2）创建基本的目录结构

main下面创建java、resource目录，

如果使用页面渲染的话，在WEB-INF下创建jsp目录



（3）导入servlet、spring-webmvc依赖，部署好tomcat

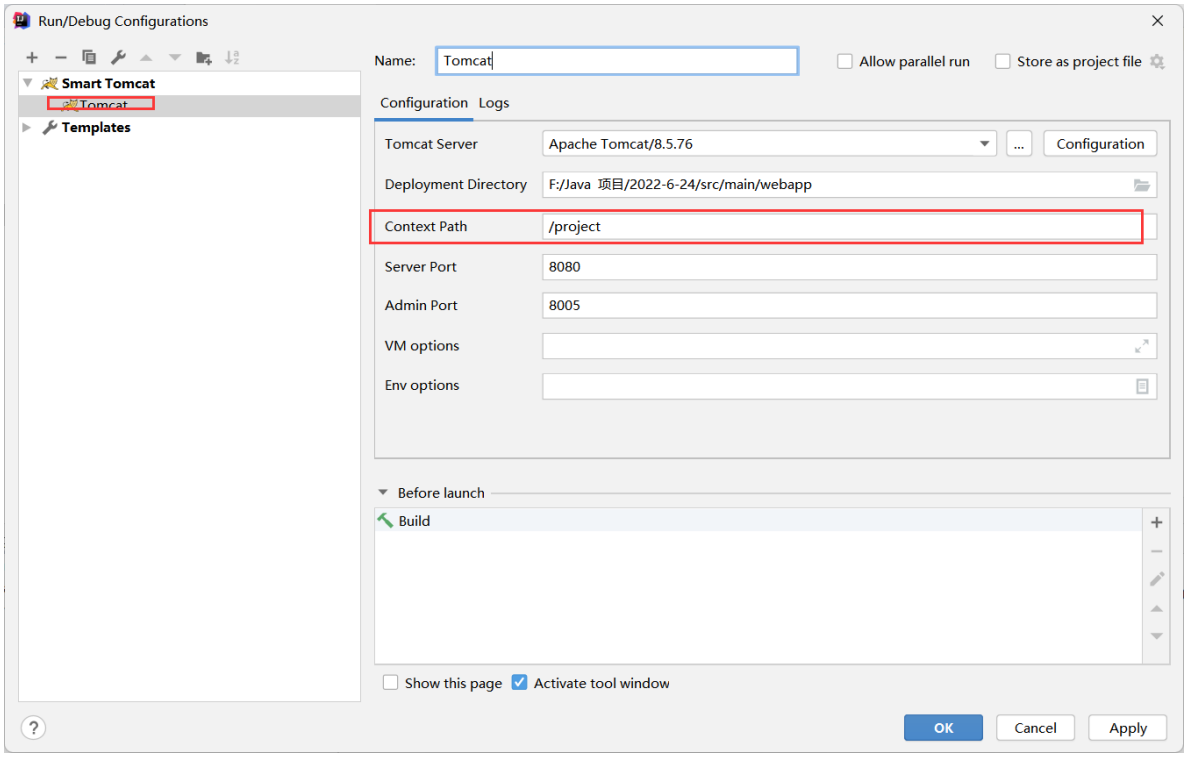
servelt-API的依赖

```
    <!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -
->
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>javax.servlet-api</artifactId>
      <version>3.1.0</version>
      <scope>provided</scope>
    </dependency>
```

下面是maven仓库中最新的spring-webmvc依赖

```
    <!-- https://mvnrepository.com/artifact/org.springframework/spring-webmvc
-->
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-webmvc</artifactId>
      <version>5.3.21</version>
    </dependency>
```

tomcat使用工具设置好



(4) 配置web.xml, 注册DispatcherServlet

SpringMVC框架是围绕 DispatcherServelt 调度器进行设计的

DispatcherServlet 就是前端控制器，对前端的各种请求进行调度，最终返回一个渲染的视图或者数据

在web.xml中注册为 DispathcherServlet 接口设计 servlet 标签及映射 servletMapper

```
<?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>springmvc</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
  <init-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>classpath:spring-mvc.xml</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>

  <servlet-mapping>
    <servlet-name>springmvc</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>

</web-app>
```

Dispatcher同时与springMVC的配置文件绑定，同时设置启动级别为1，随着服务器的启动而加载。

映射的路径为/，处理/的所有请求

/ 与 /* 的区别

> / 匹配所有的请求，但是不包括 不包括 .jsp

> /* 匹配所有的请求，也包括 .jsp

因为我们想要返回的视图都是jsp文件，所有在后面的视图解析器配置中会加上jsp文件的前缀与后缀，如果是jsp的请求的话，那么达到视图解析器就会嵌套名字了，不符合要求了。

（5）编写SpingMVC 的配置文件

使用原生的底层处理的方式使用mvc，在resource目录下创建spring-mvc.xml 配置文件

处理器映射器（HandlerMapping）

使用spingMVC框架提供的 **BeanNameUrlHandlerMapping**，记住得在下面注册bean使得id与class进行对应，这样才能找到url与接口的映射关系。

处理器适配器（HandlerAdapter）

使用SpringMVC框架提供的 **SimpleControllerHandlerAdapter**，根据映射关系，执行controller的代码，调用service层，返回数据（Json）或者模型视图（ModelAndView）。

视图解析器（ViewResolver）

使用SpringMVC框架提供的 **InternalResourceViewResolver**，并加上指定id方便调用，渲染视图界面，返回给DispatcherServlet。

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       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 ">


<!--    处理器映射器：在上下文中找到url与具体控制器（接口或者类）的关系-->
<bean
class="org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping"/>

<!--    处理器适配器：找到映射关系之后，让控制器执行之后返回数据或者视图 -->
    <bean
class="org.springframework.web.servlet.mvc.SimpleControllerHandlerAdapter"/
>

<!--    视图解析器：  如果返回数据（JSON、HTML），不需要经过该步骤-->
<!--    如果返回的是视图的话，那么原来的视图经过 之前的数据进行渲染之后 返回给
Dispatcher，在展示给前端-->
    <bean
class="org.springframework.web.servlet.view.InternalResourceViewResolver"
id="internalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/jsp/"/>
        <property name="suffix" value=".jsp"/>
    </bean>


<!--    因为前面是url是根据bean名字进行映射的，所以需要注册bean，id与class符合映射关
系-->
    <bean id="/hello" class="com.bit.controller.HelloController"/>

</beans>
```

(6) 创建具体的jsp页面

在jsp目录下写一个具体的hello.jsp 为渲染的页面，等待后端返回数据渲染

```jsp
<%--
  Created by IntelliJ IDEA.
  User: rain7
  Date: 2022/6/24
  Time: 19:50
  To change this template use File | Settings | File Templates.
--%>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>111</title>
</head>
<body>

${msg}

</body>
</html>
```

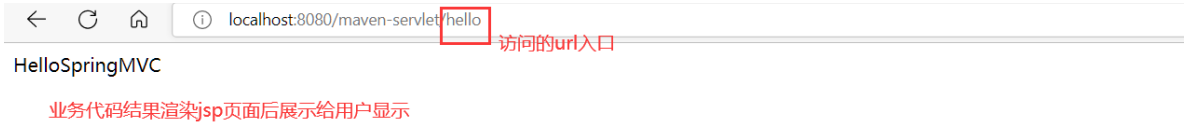
（7）写一个后端的控制器代码（Controller），部署项目进行访问

使用原生的方式，继承Controller（别导错了，导入springweb框架下的包），重写方法，给视图中的部分进行设置返回的业务数据，返回具体的视图模型。

```java
package com.bit.controller;

import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.Controller;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class HelloController implements Controller {

    @Override
    public ModelAndView handleRequest(HttpServletRequest request,
HttpServletResponse response) throws Exception {
        ModelAndView mv = new ModelAndView();

        // 给视图设置业务数据
        mv.addObject("msg","HelloSpringMVC");

        // 返回一个具体的视图
        mv.setViewName("hello");

        return mv;
    }
```

```
22 }
```

(8)运行项目，前端进行访问



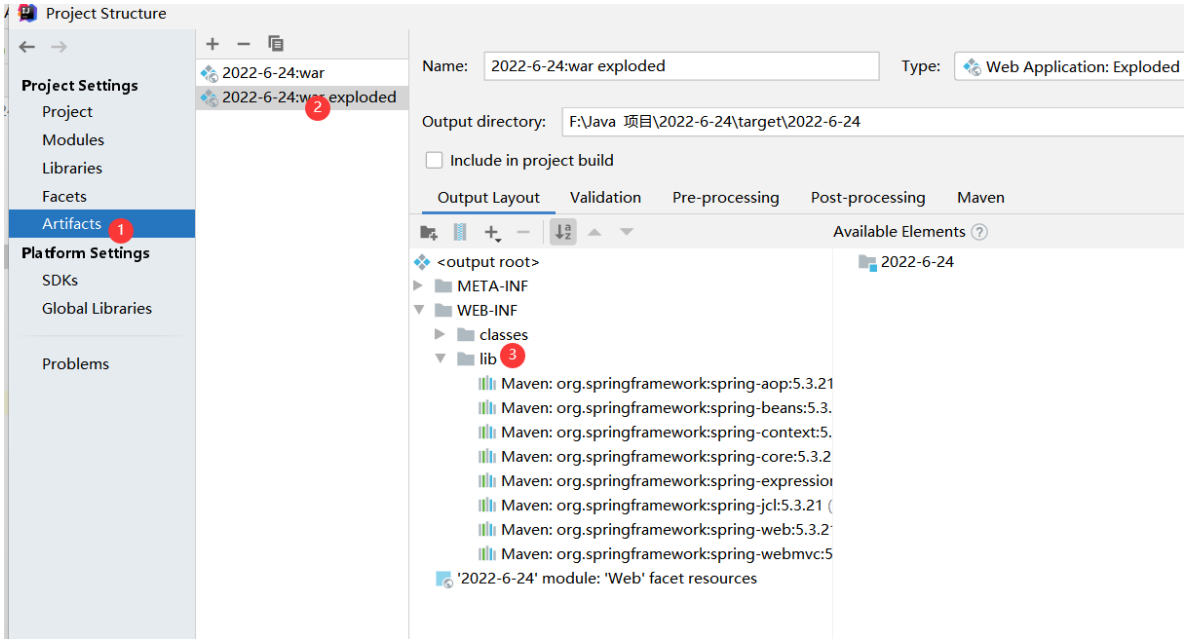
可能存在的问题

如果遇到404问题的话，那么排查步骤：

1、查看控制台输出，看一下是否是缺少了jar包

2、如果jar包存在的话，那么查看IEAD的发布项目中，是否添加了依赖

3、如果不存在依赖的话，那么在WEB-INF目录下建立lib文件夹，导入所有的依赖即可



Spring MVC执行流程

1. 用户发送请求至前端控制器DispatcherServlet。

2. DispatcherServlet收到请求调用HandlerMapping处理器映射器。

3. 处理器映射器找到具体的处理器（controller或者handle）(可以根据xml配置、注解进行查找)，生成处理器对象及处理器拦截器(如果有则生成)一并返回给DispatcherServlet。

4. DispatcherServlet调用HandlerAdapter处理器适配器。

5. HandlerAdapter经过适配调用具体的处理器(Controller，也叫后端控制器)。

6. Controller执行完成返回ModelAndView。

7. HandlerAdapter将controller执行结果ModelAndView返回给DispatcherServlet。

8. DispatcherServlet将ModelAndView传给ViewReslover视图解析器。

9. ViewReslover解析后返回具体View。

10. DispatcherServlet根据View进行渲染视图（即将模型数据填充至视图中）。

11. DispatcherServlet响应用户。

什么是handler?

Handle是什么?

- Handler是一个Controller的对象和请求方式的组合的一个**Object对象**
- **HandleExcutionChains是HandleMapping返回的一个处理执行链，它是对Handle的二次封装，将拦截器关联到一起。**然后，在DispatcherServlert中完成了拦截器链对handler的过滤。
- **DispatcherServlet要将一个请求交给哪个特定的Controller，它需要咨询一个Bean——这个Bean的名字为"HandlerMapping"。HandlerMapping是把一个URL指定到一个Controller上，（就像应用系统的web.xml文件使用<servlet-mapping>将URL映射到servlet）。**

什么是拦截器?

2.1 HandlerInterceptor拦截器

HandlerInterceptor是springMVC项目中的拦截器，它拦截的目标是请求的地址，比MethodInterceptor先执行。

实现一个HandlerInterceptor拦截器可以直接实现HandlerInterceptor接口，也可以继承HandlerInterceptorAdapter类。

这两种方法殊途同归，其实HandlerInterceptorAdapter也就是声明了HandlerInterceptor接口中所有方法的默认实现，而我们在继承他之后只需要重写必要的方法。

核心架构的具体流程步骤如下：

1.首先用户发送请求——>DispatcherServlet，前端控制器收到请求后自己不进行处理，而是委托给其他的解析器进行处理，作为统一访问点，进行全局的流程控制；

2.DispatcherServlet——>HandlerMapping， HandlerMapping 将会把请求映射为 HandlerExecutionChain 对象（包含一个Handler 处理器（Controller）对象、多个 HandlerInterceptor 拦截器）对象，通过这种策略模式，很容易添加新的映射策略；

3.DispatcherServlet——>HandlerAdapter，HandlerAdapter 将会把处理器包装为适配器，从而支持多种类型的处理器，即适配器设计模式的应用，从而很容易支持很多类型的处理器；

4.HandlerAdapter——>处理器功能处理方法的调用，HandlerAdapter 将会根据适配的结果调用真正的处理器的功能处理方法，完成功能处理；并返回一个ModelAndView 对象（包含模型数据、逻辑视图名）；

5.ModelAndView的逻辑视图名——> ViewResolver， ViewResolver 将把逻辑视图名解析为具体的 View，通过这种策略模式，很容易更换其他视图技术；

6.View——>渲染，View会根据传进来的Model模型数据进行渲染，此处的Model实际是一个Map数据结构，因此很容易支持其他视图技术；

7返回控制权给DispatcherServlet，由DispatcherServlet返回响应给用户，到此一个流程结束。