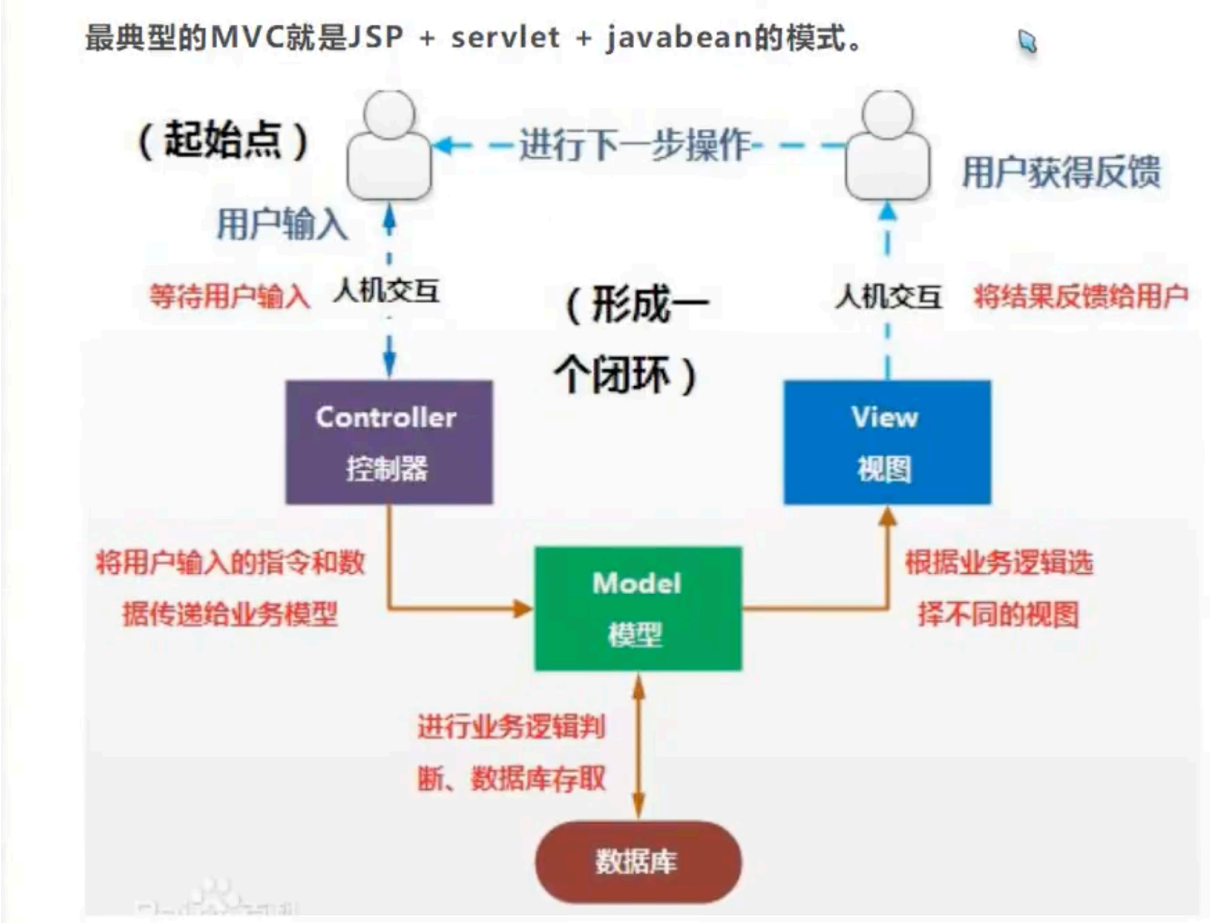
Spring MVC

1、MVC:模型(dao,service)、视图(jsp)、控制器(Servlet)

Model（模型）：数据模型，提供要展示的数据，因此包含数据和行为，可以认为是领域模型或JavaBean组件（包含数据和行为），不过现在一般都分离开来：Value Object（数据Dao）和服务层（行为Service）。也就是模型提供了模型数据查询和模型数据状态更新等功能，包括数据和业务。

View（视图）：负责进行模型的展示，一般就是我们见到的用户界面，客户想看到的东西。

Controller（控制器）：接收用户的请求，委托给模型进行处理（状态改变），处理完毕后把返回的模型数据返回给视图，由视图负责展示。也就是说控制器做了个**调度员**的工作



职责分析：

1、Controller：控制器

- 取得表单数据
- 调用业务逻辑
- 转向指定的页面

2、Model：模型

- 实现业务逻辑
- 保存数据状态

3、View：视图

- 显示页面

2、回顾Servlet

1、编写Servlet类

```
public class HelloServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        //1、获取前端参数；
        String method = req.getParameter("method");
        if (method.equals("add")){
            req.getSession().setAttribute("msg","执行了add方法");
        }
        if (method.equals("delete")){
            req.getSession().setAttribute("msg","执行了delete方法");
        }
        // 2、调用业务逻辑；

        // 3、视图转发或重定向
        req.getRequestDispatcher("/WEB-INF/test.jsp").forward(req,resp);
    }

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        this.doGet(req,resp);
    }
}
```

2、配置web.xml

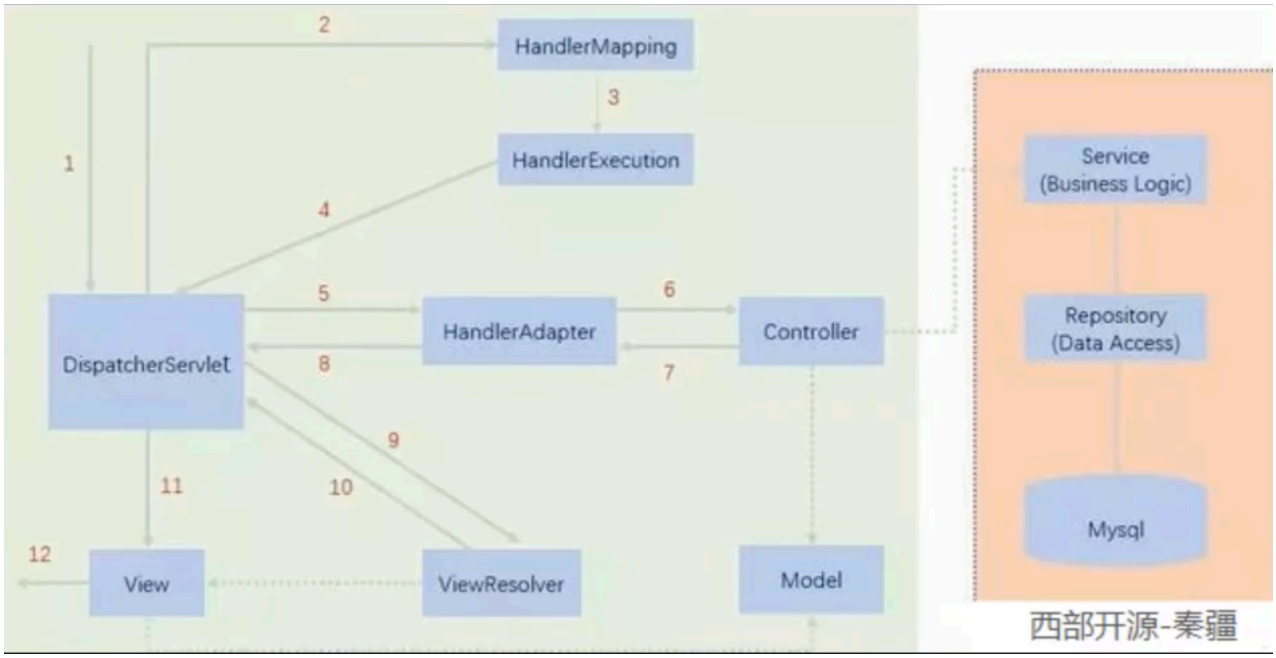
```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
         version="4.0">
    <servlet>
        <servlet-name>helloServlet</servlet-name>
        <servlet-class>com.chamfers.servlet.HelloServlet</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>helloServlet</servlet-name>
        <url-pattern>/hello</url-pattern>
```

```
    </servlet-mapping>
</web-app>
```

3、编写test.jsp

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

3、初识SpringMVC



1、编写springmvc-servlet.xml配置

```
<?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
        https://www.springframework.org/schema/beans/spring-beans.xsd">
    <!--处理器映射器-->
    <bean
class="org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping"/
>
    <!--处理器适配器-->
```

```
    <bean class="org.springframework.web.servlet.mvc.SimpleControllerHandlerAdapter" />
    <!--视图解析器-->
    <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver" id="internalResourceViewResolver">
      <!--在WEB-INF文件夹下新建jsp文件夹，存放视图.jsp-->
        <property name="prefix" value="/WEB-INF/jsp/"/>
        <property name="suffix" value=".jsp"/>
    </bean>

    <!--Handler-->
    <bean id="/hello" class="com.chamfers.controller.HelloController"/>

</beans>

```

2、在web.xml中进行配置

```
<?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">
    <!--1、注册 DispatcherServlet：是SpringMVC的核心：请求分发器、前端控制器-->
    <servlet>
        <servlet-name>springmvc</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <!--DispatcherServlet要绑定Spring的配置文件-->
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>classpath:springmvc-servlet.xml</param-value>
        </init-param>
        <!--启动级别-->
        <load-on-startup>1</load-on-startup>
    </servlet>

    <!--/  只匹配所有的请求（不包括.jsp页面）-->
    <!--/* 匹配所有的请求（包括.jsp页面）-->
    <!--https://localhost:8080/hello.jsp-->
    <servlet-mapping>
        <servlet-name>springmvc</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>

```

```
</web-app>
```

3、编写Servlet或者Controller类

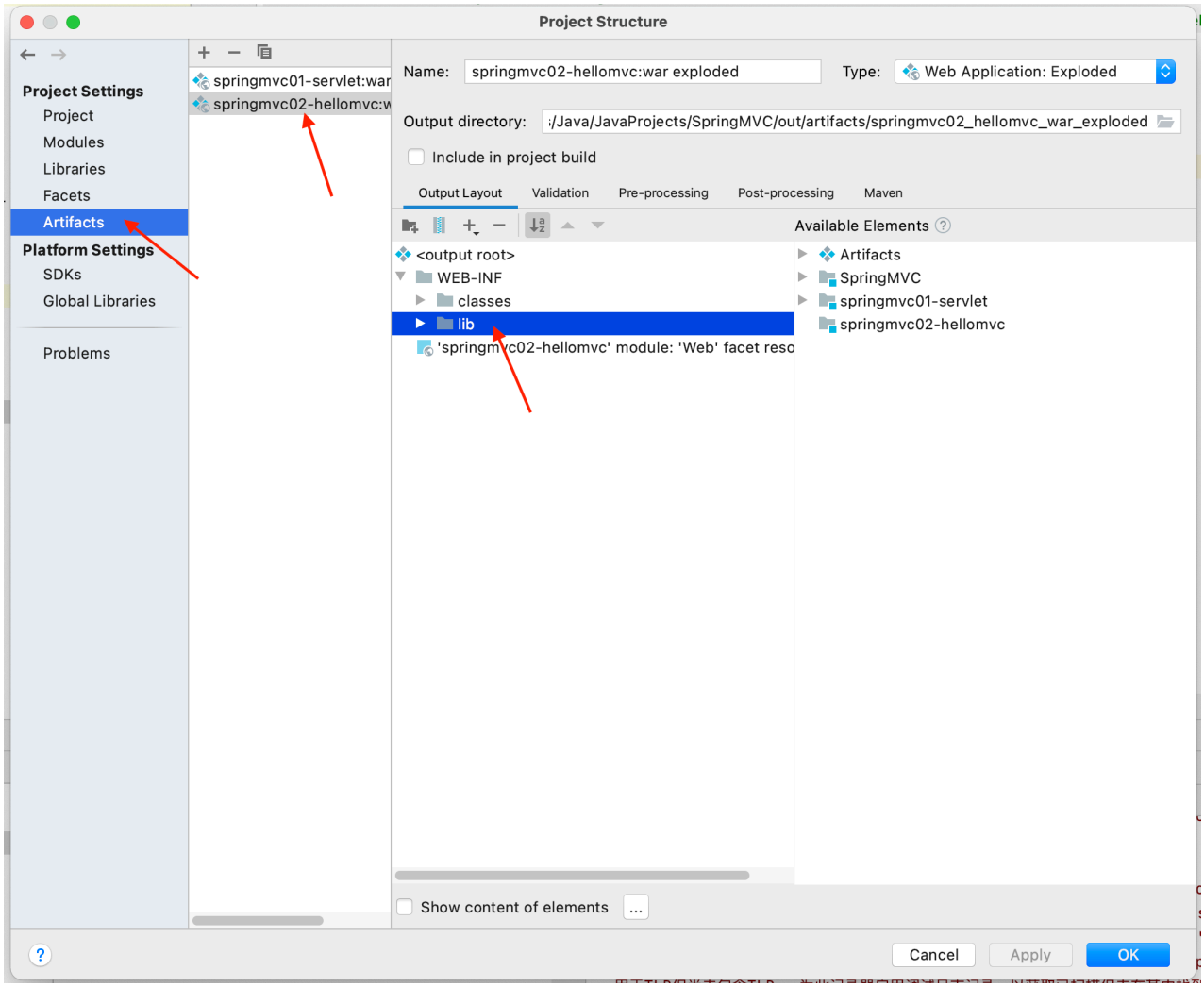
```
//先导入Controller接口
public class HelloController implements Controller {
    public ModelAndView handleRequest(HttpServletRequest
httpServletRequest, HttpServletResponse httpServletResponse) throws
Exception {
        //ModelAndView：模型和视图
        ModelAndView modelAndView = new ModelAndView();

        //调用业务层

        //封装对象，放在ModelAndView中：Model
        modelAndView.addObject("msg","Hello Spring MVC");
        //封装要跳转的视图，放在ModelAndView中：View
        modelAndView.setViewName("hello");//通过springmvc-servlet.xml中的前
缀和后缀：/WEB-INF/jsp/hello.jsp
        return modelAndView;
    }
}
```

注意点：

1、查看控制台输出，看一下是不是少了什么jar包。

2、如果jar包存在，显示无法输出，就在IDEA的项目发布中，添加lib依赖！！

3、重启Tomcat即可解决！



4、注解开发SpringMVC

1、配置web.xml文件

```
<?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">
    <!--DispatcherServlet-->
    <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:springmvc-servlet.xml</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
```

```
16
17      <servlet-mapping>
18          <servlet-name>springmvc</servlet-name>
19          <!--/: 只匹配所有的请求（不包含.jsp请求）-->
20          <!--/*: 匹配所有的请求（包含.jsp请求）-->
21          <url-pattern>/</url-pattern>
22      </servlet-mapping>
23  </web-app>
```

2、配置springmvc-servlet.xml文件

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <beans xmlns="http://www.springframework.org/schema/beans"
3          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4          xmlns:context="http://www.springframework.org/schema/context"
5          xmlns:mvc="http://www.springframework.org/schema/mvc"
6          xsi:schemaLocation="http://www.springframework.org/schema/beans
7           https://www.springframework.org/schema/beans/spring-beans.xsd
8           http://www.springframework.org/schema/context
9           https://www.springframework.org/schema/context/spring-context.xsd
10          http://www.springframework.org/schema/mvc
11          https://www.springframework.org/schema/mvc/spring-mvc.xsd">
12
13      <context:component-scan base-package="com.chamfers.controller"/>
14      <mvc:default-servlet-handler/>
15      <!--处理器映射器和处理器适配器-->
16      <!--HandlerMapping和HandlerAdapter-->
17      <mvc:annotation-driven/>
18
19      <!--视图解析器-->
20      <bean
    class="org.springframework.web.servlet.view.InternalResourceViewResolver">
21          <property name="prefix" value="/WEB-INF/jsp/"/>
22          <property name="suffix" value=".jsp"/>
23      </bean>
24  </beans>
```

3、编写Controller类

```java
@Controller
public class HelloController {
    <!--网络请求中的https请求-->
    @RequestMapping("hello")
    public String hello(Model model){
        //封装数据
        model.addAttribute("msg","Hello,SpringMVC Annotation");
        return "test";//会被视图解释器处理 ViewResolver
    }
}
```

4.1、@Controller

控制器Controller

- 控制器复杂提供访问应用程序的行为，通常通过接口定义或者注解定义两种方法实现。
- 控制器负责解析用户的请求并将其转换为一个模型。
- 在Spring MVC中一个控制器类可以包含多个方法。
- 在Spring MVC中，对于Controller的配置方式有很多种。

这样处理之后，我们的两个请求都可以指向一个视图，但是页面结果是不一样的，从而这里可以看出视图是被复用的，而控制器和视图之间是弱耦和关系。

4.2、@RequestMapping

- @RequestMapping注解用于映射url到控制器类或一个特定的处理程序方法。可用于类或方法上。用于类上，表示类中的所有响应的方法都是以该地址作为**父路径**
- 为了测试结论更加准确，我们可以加上一个项目名测试myweb
- 只注解在方法上面

```java
@Controller
public class TestController{
  @RequestMapping("/h1")
  public String test(){
    return "test";
  }
}
```

访问路径：http://localhost:8080/项目名/h1

- 同时注解类和方法

```java
@Controller
@RequestMapping("/admin")
public class TestController{
  @RequestMapping("/h1")
  public String test(){
    return "test";
  }
}
```

访问路径：http://localhost:8080/项目名/admin/h1,需要先指定类的路径在指定方法的路径。

建议：

```java
@Controller
public class TestController{
  //建议用这种直接写在方法中的类型
  @RequestMapping("/admin/h1")
  public String test(){
    return "test";
  }
}
```

5、RestFul风格

概念

RestFul就是一个资源定位及资源操作的风格。不是标准也不是协议，只是一种风格。基于这种风格设计的软件可以更简洁，更有层次，更便于实现缓存等机制

功能

- 资源：互联网所有的事务都可以被抽象为资源
- 资源操作：使用POST、DELETE、PUT、GET，使用不同方式对资源进行操作
- 分别对应添加、删除、修改、查询

传统方式操作资源：通过不同的参数来实现不同的效果！方法单一：post和get

- http://127.0.0.1/item/queryItem.action?id=1 查询，GET
- http://127.0.0.1/item/saveItem.action 新增，POST
- http://127.0.0.1/item/updateItem.action 更新，POST
- http://127.0.0.1/item/queryItem.action?id=1 删除，GET或POST

使用ResrFul操作资源：可以通过不同的请求方式和来实现不同的效果！如下：请求地址一样，但是功能可能不同

- http://127.0.0.1/item/1 查询，GET
- http://127.0.0.1/item 新增，POST

- http://127.0.0.1/item 更新，PUT
- http://127.0.0.1/item/1 删除，DELETE

代码测试

1、编写Controller类

```
@Controller
public class RestFulController {
}
```

2、在Spring MVC中可以使用 @PathVariable注解，让方法参数的值对应绑定到一个URL模版变量上。

```
@Controller
public class RestFulController {
    //之前的URL：http://localhost:8080/test? a=1&b=4
    //RestFul URL:http://localhost:8080/test/1/4
    @RequestMapping("/test/{a}/{b}")
    public String test1(Model model, @PathVariable int a,@PathVariable int b){
        int res = a+b;
        model.addAttribute("msg","结果是："+res);
        return "test";
    }
}
```

注意：@PathVariable有类型限制

使用method属性指定请求类型

用于约束请求类型，可以收窄请求范围。指定请求谓词的类型入GET、POST、HEAD、OPTIONS、PUT、PATCH、DELETE、TRACE、等

```java
@Controller
public class RestFulController {
    //之前的URL：http://localhost:8080/test? a=1&b=4
    //RestFul URL:http://localhost:8080/test/1/4

    //映射访问路径，必须是POST请求
    @RequestMapping(value = "/test/{a}/{b}",method = RequestMethod.POST)
    public String test1(Model model, @PathVariable int a,@PathVariable int b){
        int res = a+b;
        model.addAttribute("msg","结果是："+res);
        return "test";
    }
}
```

方法级别的注视变体有如下几个：

```
@GetMapping
@PostMapping
@PutMapping
@DeleteMapping
@PatchMapping
```

```java
@Controller
public class RestFulController {
    //之前的URL：http://localhost:8080/test? a=1&b=4
    //RestFul URL:http://localhost:8080/test/1/4

    //映射访问路径，必须是POST请求
    //@RequestMapping(value = "/test/{a}/{b}",method = RequestMethod.POST)

    @GetMapping("/test/{a}/{b}")
    public String test1(Model model, @PathVariable int a,@PathVariable int b){
        int res = a+b;
        model.addAttribute("msg","结果是："+res);
        return "test";
    }
}
```

简洁、高效、安全

6、SpringMVC：结果跳转方式

6.1、ModelAndView

设置ModelAndView对象，根据view的名称和视图解析器跳转到指定的页面。

页面：[视图解析器前缀]+viewName+[视图解析器后缀]

```
<!--视图解析器：DispatcherServlet给他的ModelAndView
1、获得了ModelAndView的数据
2、获得了ModelAndView的视图名
3、拼接视图名，找到对应的视图（/WEB-INF/jsp/hello.jsp）
4、将数据渲染到视图上
-->
<bean
class="org.springframework.web.servlet.view.InternalResourceViewResolver"
id="internalResourceViewResolver">
    <!--前缀-->
    <property name="prefix" value="/WEB-INF/jsp/"/>
    <!--后缀-->
    <property name="suffix" value=".jsp"/>
</bean>
```

对应的controller类

```
//先导入Controller接口
public class HelloController implements Controller {
    public ModelAndView handleRequest(HttpServletRequest
httpServletRequest, HttpServletResponse httpServletResponse) throws
Exception {
        //ModelAndView：模型和视图
        ModelAndView modelAndView = new ModelAndView();

        //调用业务层

        //封装对象，放在ModelAndView中：Model
        modelAndView.addObject("msg","Hello Spring MVC");
        //封装要跳转的视图，放在ModelAndView中：View
        modelAndView.setViewName("hello");//通过springmvc-servlet.xml中的前
缀和后缀：/WEB-INF/jsp/hello.jsp
        return modelAndView;
    }
}
```

6.2、通过Spring MVC来实现转发和重定向（无需视图解析器）

```java
@Controller
public class ResultSpringMVC{
  @ResultMapping("/rsm/t1")
  public String test1(){
    //转发
    return "/index.jsp";
  }
  @ResultMapping("/rsm/t2")
  public String test2(){
    //转发二:
    return "forward:/index.jsp";
  }
    @ResultMapping("/rsm/t3")
  public String test3(){
    //重定向
    return "redirect:/index.jsp";
  }
}
```

6.3、通过SpringMVC来实现转发和重定向（有视图解析器）

重定向：不需要视图解析器，本质就是重新请求一个新的地方，所以注意路径问题。

可以重定向到另一个请求实现。

```java
@Controller
public class ResultSpringMVC2{
  @ResultMapping("/rsm2/t1")
  public String test1(){
    //转发
    return "test";
  }
  @ResultMapping("/rsm2/t2")
  public String test3(){
    //重定向
    return "redirect:/index.jsp";
    //return "redirect:hello.do";//hello.do为另一个请求
  }
}
```

7、SpringMVC：数据处理

7.1、处理提交数据

1、提交的域名称和处理方法的参数名一致

提交数据：http://localhost:8080/hello?name=chamfers

处理方法：

```
@RequestMapping("/hello")
public String hello(String name){
  System.out.println(name);
  return "hello";
}
```

2、提交的域名称和处理方法的参数名不一致

提交数据：http://localhost:8080/hello?username=chamfers

处理方法：

```
//@RequestParam("username"):username提交的域名称
@RequestMapping("/hello")
public String hello(@RequestParam("username")String name){
  System.out.println(name);
  return "hello";
}
```

建议：前端接收的数据无论如何都要加上@RequestParam("username")注解

3、提交的是一个对象

要求提交的表单域和对象的属性名一致，参数使用对象即可

1、实体类

```
public class User{
  private int id;
  private String name;
  private int age;
  //构造
  //get/set
  //toString()
}
```

2、提交数据：http://localhost:8080/mvc04/user?name=chamfers&id=1&age=15

3、处理方法

```java
@RequestMapping("/user")
public String hello(User user){
  System.out.println(user);
  return "hello";
}
```

后台输出：User(id=1,name='chamfers',age=15)

说明：如果使用对象的话，前端传递的参数名和对象属性名必须一致，否则就是null

7.2、数据显示到前端

方式1、通过Model And View

方式2、通过ModelMap

方式3、通过Model

对比

- Model：只有寥寥几个方法，只适用于存储数据，简化了新手对于Model对象的操作和理解
- ModelMap：继承了LinkedMap，除了实现了自身的一些方法，永扬继承了LinkedMap的方法和特性
- ModelAndView：可以在存储数据的同时，可以进行设置返回的逻辑视图，进行控制展示层的跳转。

8、乱码问题解决

测试步骤：

1、在首页编写一个提交的表单

```html
<form action="/e/t" method="post">
  <input type="text" name="name">
  <input type="submit">
</form>
```

2、后台编写对应的处理类

```java
@Controller
public class Encoding{
  @RequestMapping("/e/t")
  public String test(Model model,String name){
    model.addAttribute("msg",name)//获得表单提交的值
    return "test";//跳转懂啊test页面显示输入的值
  }
}
```

3、可能乱码

乱码问题在我们开发中十分常见的问题。

Spring MVC给我们提供了一个过滤器，可以配置在web.xml中

修改web.xml文件需要重启服务器

```
<!--2、配置乱码过滤器-->
<filter>
    <filter-name>encoding</filter-name>
    <filter-
class>org.springframework.web.filter.CharacterEncodingFilter</filter-
class>
</filter>

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

9、SpringMVC：JSON讲解

9.1、什么是JSON?

- JSON(JavaScript Object Notation,JS对象标记)是一种轻量级的数据交换格式，目前使用特别广泛。
- 采用完全独立于编程语言的**文本格式**来存储和使用数据
- 简洁和清晰的层次结构使得JSON成为理想的数据交换语言。

在JavaScript语言中，一切都是对象。因此，任何JavaScript支持的类型都可以通过JSON来表示，例如字符串、数字、对象、数组等。其语法格式和要求：

- 对象表示为键值对，数据由逗号分隔
- 花括号保存对象
- 方括号保存数组

JSON键值对是用来保存JavaScript对象的一种方式，和JavaScript对象的写法也大同小异，键/值对组合中的键名写在前面并用双引号“”包裹，使用冒号分隔，然后紧接着值：

```
{"name":"chamfers"}
{"age":"18"}
{"sex":"男"}
```

9.2、JSON和JavaScript对象互转

- 使用JSON.parse()方法，实现从JSON自负床转换为JavaScript对象

```
var obj = JSON.parse('{"name":"chamfers","sex":"男"}')
//结果是：{name:'chamfers',sex:'男'}
```

- 使用JSON.stringify()方法，实现从JavaScript对象转换为JSON字符串

```
var json = JSON.stringify({name:'chamfers',sex:'男'})
//结果是：'{"name":"chamfers","sex":"男"}'}
```

10、Controller返回JSON数据

- Jackson应该是目前比较好的json解析工具了
- 当然工具不止一个，比如阿里巴巴的fastjson等
- 我们这里使用Jackson，使用它需要导入它的jar包

```
<dependency>
  <groupId>com.fasterxml.jackson.core</groupId>
  <artifactId>jackson-databind</artifactId>
  <version>2.11.3</version>
</dependency>
```

- 配置SpringMVC需要的配置

web.xml

```
<?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">
    <!--1、DispatcherServlet-->
    <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:springmvc-servlet.xml</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>springmvc</servlet-name>
        <!--/: 只匹配所有的请求（不包含.jsp请求）-->
        <!--/*: 匹配所有的请求（包含.jsp请求）-->
        <url-pattern>/</url-pattern>
    </servlet-mapping>


```

```
    <!--2、配置乱码过滤器-->
    <filter>
        <filter-name>encoding</filter-name>
        <filter-
class>org.springframework.web.filter.CharacterEncodingFilter</filter-
class>
    </filter>

    <filter-mapping>
        <filter-name>encoding</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
</web-app>
```

- Springmvc-servlet.xml

```
<?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
        https://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.chamfers.controller"/>
    <mvc:default-servlet-handler/>
    <!--处理器映射器和处理器适配器-->
    <!--HandlerMapping和HandlerAdapter-->
    <mvc:annotation-driven/>

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

- 编写一个User的实体类，然后我们去编写我们的测试Controller

```java
@Data
@AllArgsConstructor
@NoArgsConstructor
public class User{
  private String name;
  private int age;
  private String sex;
}
```

- 这里我们需要两个新东西，@ResponseBody，ObjectMapper对象，其具体用法如下：

```java
@Controller
public class UserController{
  @RequestMapping("/j1")
  @ResponseBody//加上这个注解，就不会走视图解析器，会直接返回一个字符串
  public String json1(){
    //jackson,ObjectMapper
    ObjectMapper mapper = new ObjectMapper();
    //使用ObjectMapper 来格式化输出；不使用时间戳的方式

 mapper.configure(SerializationFeature.WRITE_DATES_AS_TIMESTAMPS,false);
    //创建一个对象
    User user = new User("陈",1,"男");
    String str = mapper.writeValueAsString(user);
    return str;
    //return new ObjectMapper().writeValueAsString(user);
  }
}
```

- 解决乱码问题
    - 方法一：在@RequestMapping("/j1")中添加"produces = "application/json;charset=utf-8""

```java
@RequestMapping(value = "/j1",produces =
"application/json;charset=utf-8")
```

    - 方式2:在springmvc-servlet.xml中配置：

```xml
<!--JSON乱码问题配置-->
<mvc:annotation-driven>
    <mvc:message-converters register-defaults="true">
        <bean
class="org.springframework.http.converter.StringHttpMessageConverte
r">
            <constructor-arg value="utf-8"/>
        </bean>
```

```
        <bean class="org.springframework.http.converter.json.MappingJackson2HttpMessageConverter">
            <property name="objectMapper">
                <bean class="org.springframework.http.converter.json.Jackson2ObjectMapperFactoryBean">
                    <property name="failOnEmptyBeans" value="false"/>
                </bean>
            </property>
        </bean>
    </mvc:message-converters>
</mvc:annotation-driven>
```