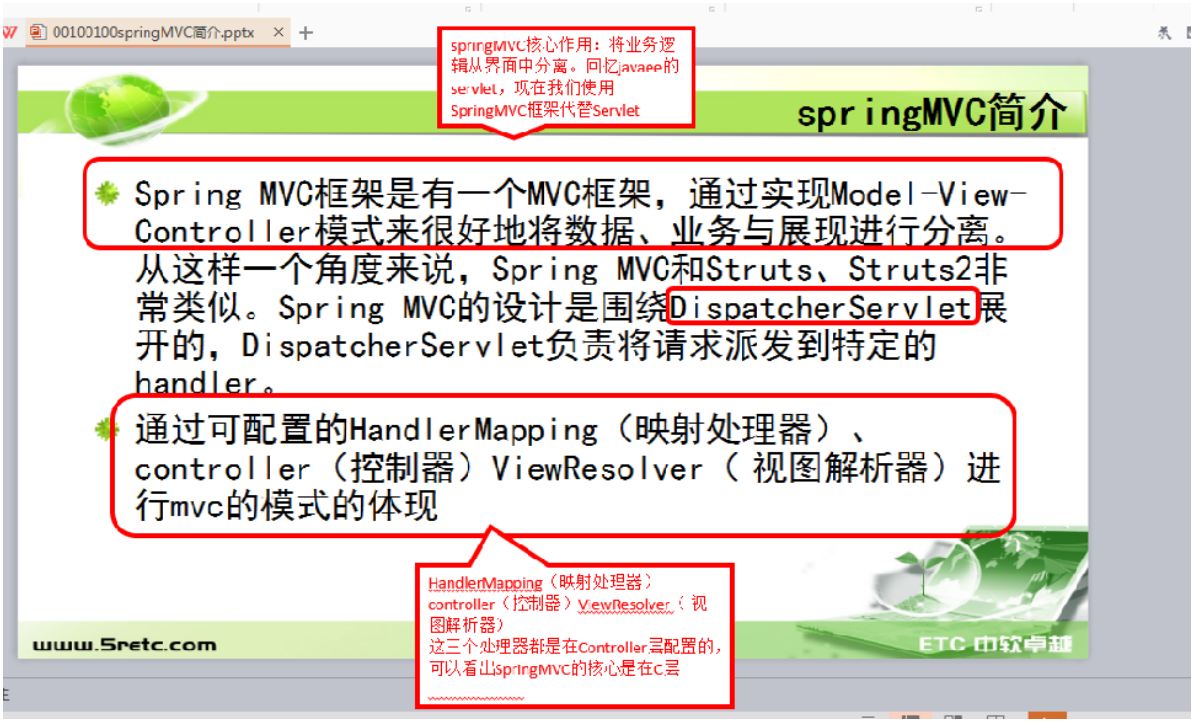
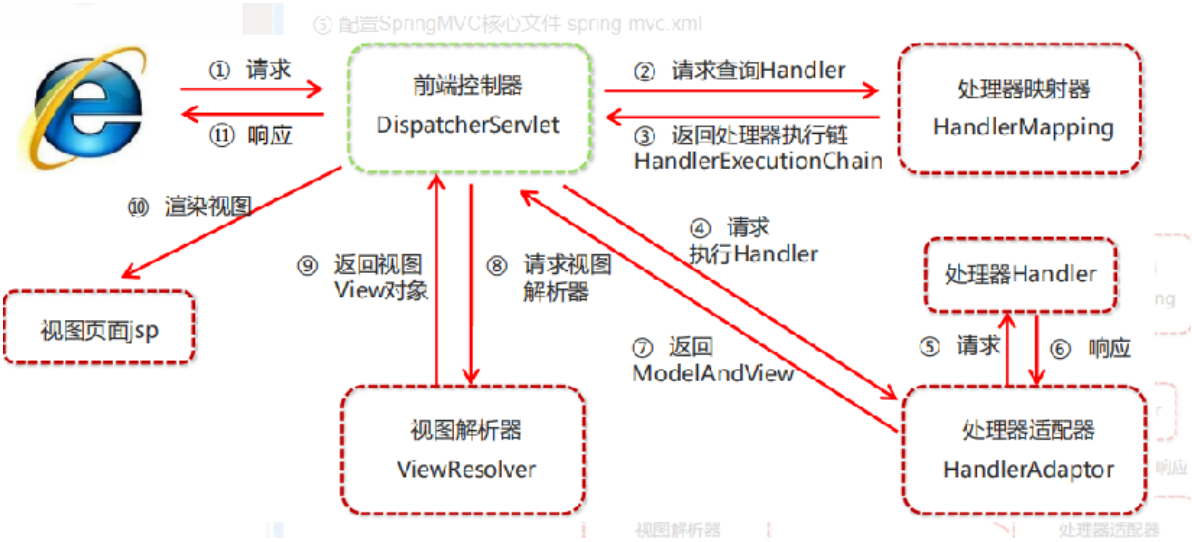
1、什么是SpringMVC



2、SpringMVC 执行的流程



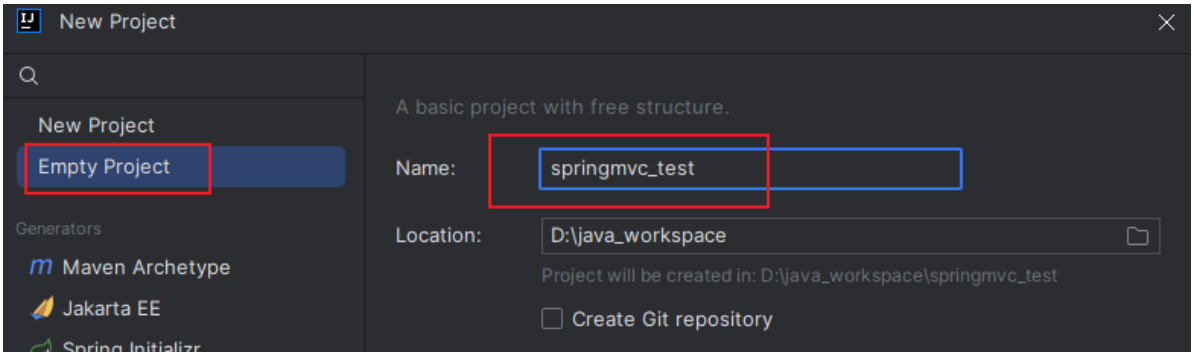
① 用户发送请求至前端控制器DispatcherServlet。
② DispatcherServlet收到请求调用HandlerMapping处理器映射器。
③ 处理器映射器找到具体的处理器(可以根据xml配置、注解进行查找)，生成处理器对象及处理器拦截器(如果有则生成)一并返回给DispatcherServlet。
④ DispatcherServlet调用HandlerAdapter处理器适配器。
⑤ HandlerAdapter经过适配调用具体的处理器(Controller，也叫后端控制器)。
⑥ Controller执行完成返回ModelAndView。
⑦ HandlerAdapter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）。DispatcherServlet响应用户。

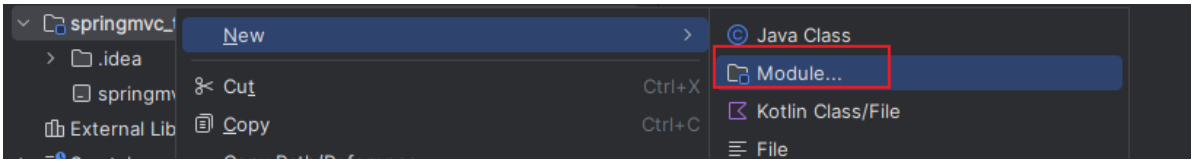
> 简单说：

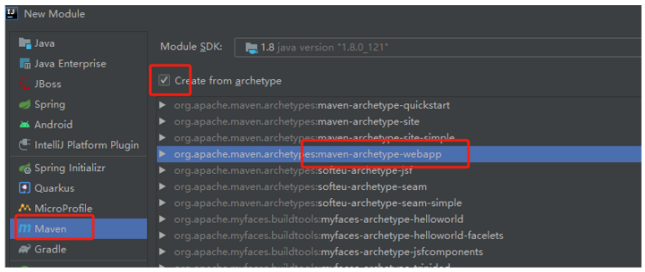
> 用户发送请求，被DispatherServlet 拦截，通过HandlerMapping映射处理器，找到对应的Controller控制器
>
> 在控制器中处理请求与响应结果，将数据封装到ModelAndView中，在通过ViewResolver 指定页面的前缀与后缀，拼接
>
> ModelAndView中的视图名称，找到要渲染的页面，最终将ModelAndView中的数据渲染到视图中。

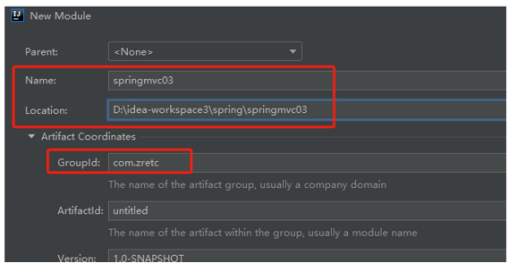
3、项目演示

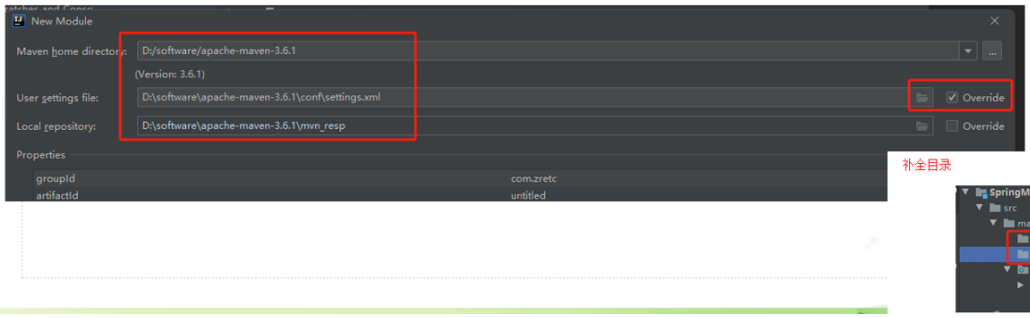
3.1 搭建项目

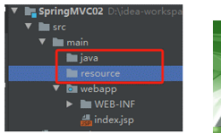










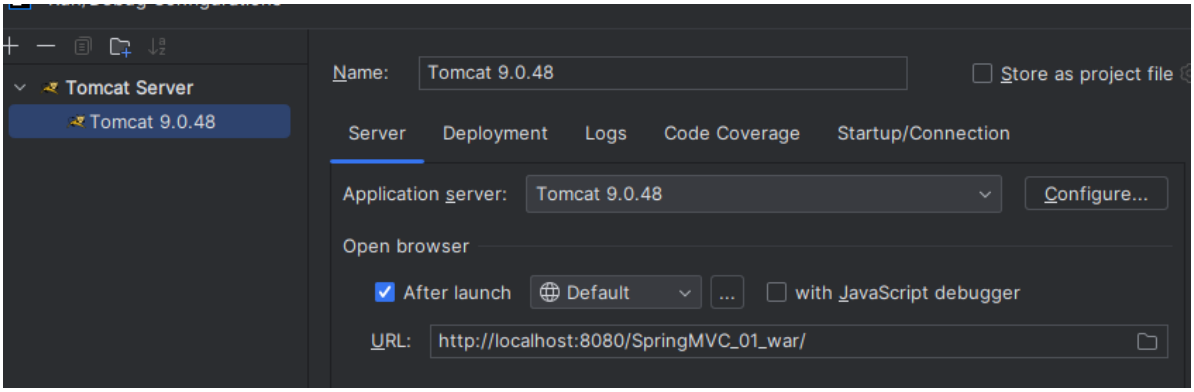


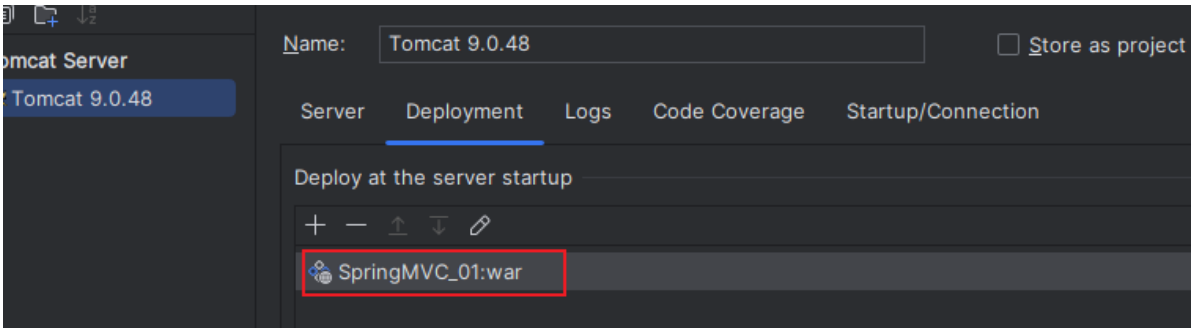
引入依赖：

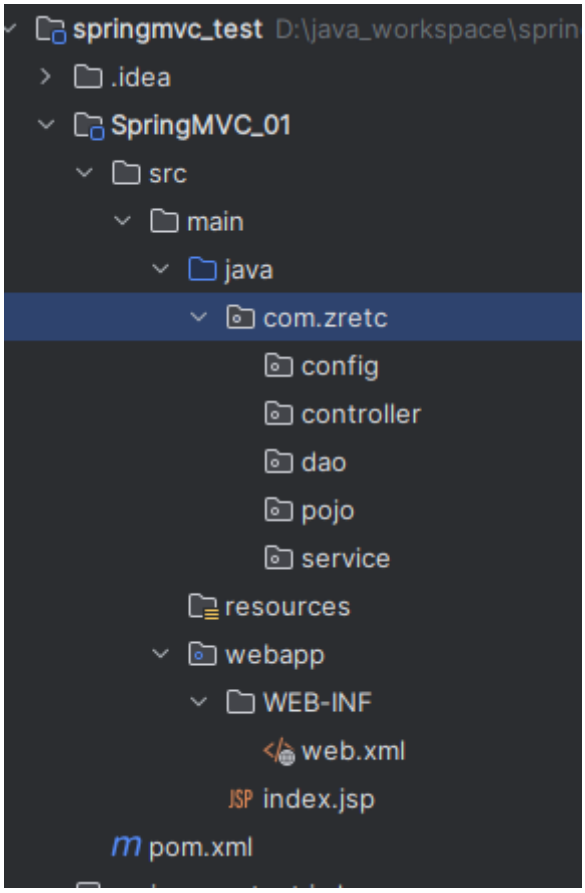
```
<dependencies>
        <!--servlet 依赖-->
```

```
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>javax.servlet-api</artifactId>
            <version>3.1.0</version>
            <scope>provided</scope>
        </dependency>
        <!--webmvc 依赖-->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
            <version>5.0.5.RELEASE</version>
        </dependency>
    </dependencies>
```

tomcat部署项目







springmvc的配置文件

```
// springmvc的配置文件
@Configuration //配置文件
@ComponentScan("com.zretc.controller")
public class SpringMvcConfig {
}
```

核心配置文件，相当于web.xml

```
import
org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServle
tInitializer;
// 相当于 web.xml
public class ServletContainersInitConfig extends
AbstractAnnotationConfigDispatcherServletInitializer {
    @Override
    protected Class<?>[] getRootConfigClasses() {
        return new Class[0]; // spring 配置类
    }

    @Override
    protected Class<?>[] getServletConfigClasses() {
        return new Class[]{SpringMvcConfig.class}; // springmvc配置类
    }

    @Override
    protected String[] getServletMappings() {// 由springmvc控制器处理的请求映射路径
        return new String[]{"/"}; // / 任意请求都会被拦截
    }
```

```
}
```

控制器

```
@Controller
public class UserController {

    @RequestMapping("/selectAll") // 映射该方法的请求路径
    @ResponseBody // 将return的值响应到前端
    public String selectAll(){
        System.out.println("{'msg','selectAll'}");
        return "{'msg','selectAll'}";
    }
}
```

知识点1：@Controller

名称	@Controller
类型	类注解
位置	SpringMVC控制器类定义上方
作用	设定SpringMVC的核心控制器bean

知识点2：@RequestMapping

名称	@RequestMapping
类型	类注解或方法注解
位置	SpringMVC控制器类或方法定义上方
作用	设置当前控制器方法请求访问路径
相关属性	value(默认)，请求访问路径

知识点3：@ResponseBody

名称	@ResponseBody
类型	类注解或方法注解
位置	SpringMVC控制器类或方法定义上方
作用	设置当前控制器方法响应内容为当前返回值，无需解析

名称	@ComponentScan
类型	类注解
位置	类定义上方
作用	设置spring配置类扫描路径，用于加载使用注解格式定义的bean
相关属性	excludeFilters:排除扫描路径中加载的bean,需要指定类别(type)和具体项(classes) includeFilters:加载指定的bean，需要指定类别(type)和具体项(classes)

4、 配置文件

- config目录存入的是配置类，写过的配置类有：
  - ServletContainersInitConfig
  - SpringConfig
  - SpringMvcConfig
  - JdbcConfig
  - MybatisConfig
- controller目录存放的是SpringMVC的controller类
- service目录存放的是service接口和实现类
- dao目录存放的是dao/Mapper接口

controller、service和dao这些类都需要被容器管理成bean对象，那么到底是该让SpringMVC加载还是让Spring加载呢？

- SpringMVC加载其相关bean(表现层bean)，也就是controller包下的类
- Spring控制的bean
  - 业务bean(Service)
  - 功能bean(DataSource,SqlSessionFactoryBean,MapperScannerConfigurer等)

针对上面的问题，解决方案也比较简单，就是：

- 加载Spring控制的bean的时候排除掉SpringMVC控制的备案 具体该如何排除，
- 有两种方式来解决:
  - 方式一:Spring加载的bean设定扫描范围为com.zretc,排除掉controller包中的bean
  - 方式二:Spring加载的bean设定扫描范围为精准范围，例如service包、dao包等
  - 方式三:不区分Spring与SpringMVC的环境，加载到同一个环境中[了解即可]

方式一:修改Spring配置类，设定扫描范围为精准范围。

```
@Configuration
//@ComponentScan("com.zretc")
// 方式一:修改Spring配置类，设定扫描范围为精准范围。
@ComponentScan({"com.zretc.service","com.zretc.dao"})
public class SpringConfig {
}
```

说明：

上述只是通过例子说明可以精确指定让Spring扫描对应的包结构，真正在做开发的时候，因为Dao最终是交给MapperScannerConfigurer对象来进行扫描处理的，我们只需要将其扫描到service包即可。

方式二:修改Spring配置类，设定扫描范围为com.zretc,排除掉controller包中的bean

```java
@Configuration
//@ComponentScan("com.com.zretc")
// 方式一:修改Spring配置类，设定扫描范围为精准范围。
//@ComponentScan({"com.com.zretc.service","com.com.zretc.dao"})
// 方式二:修改Spring配置类，设定扫描范围为com.zretc,排除掉controller包中的bean
@ComponentScan(value="com.zretc",
        excludeFilters=@ComponentScan.Filter(
                type = FilterType.ANNOTATION,
                classes = Controller.class
        )
)
public class SpringConfig {
}
```

- excludeFilters属性：设置扫描加载bean时，排除的过滤规则
- type属性：设置排除规则，当前使用按照bean定义时的注解类型进行排除
  - ANNOTATION：按照注解排除
- classes属性：设置排除的具体注解类，当前设置排除@Controller定义的bean

4.1 项目演示

引入依赖

```xml
<dependencies>
        <!--servlet 依赖-->
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>javax.servlet-api</artifactId>
            <version>4.0.1</version>
            <scope>provided</scope>
        </dependency>
        <!--webmvc 依赖-->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
            <version>5.0.5.RELEASE</version>
        </dependency>
    </dependencies>
```

spring配置文件

```
@Configuration
// 一：精准扫描，service,dao
//@ComponentScan({"com.zretc.service","com.zretc.dao"})
// 二：扫描全包，排除controller
@ComponentScan(value = "com.zretc",
        excludeFilters = @ComponentScan.Filter(
                type = FilterType.ANNOTATION,
                classes = Controller.class
        ))
public class SpringConfig {
}
```

springmvc配置类

```
// springmvc的配置文件
@Configuration //配置文件
@ComponentScan("com.zretc.controller")
public class SpringMvcConfig {
}
```

核心配置类:

```
import
org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServle
tInitializer;
// 相当于 web.xml
public class ServletContainersInitConfig extends
AbstractAnnotationConfigDispatcherServletInitializer {
    @Override
    protected Class<?>[] getRootConfigClasses() {
        return new Class[0]; // spring 配置类
    }

    @Override
    protected Class<?>[] getServletConfigClasses() {
        return new Class[]{SpringMvcConfig.class}; // springmvc配置类
    }

    @Override
    protected String[] getServletMappings() {// 由springmvc控制器处理的请求映射路径
        return new String[]{"/"}; // / 任意请求都会被拦截
    }
}
```

前面我们已经完成了入门案例相关的知识学习，接来了我们就需要针对SpringMVC相关的知识点进行系统的学习，之前我们提到过，SpringMVC是web层的框架，主要的作用是接收请求、接收数据、响应结果，所以这一章节是学习SpringMVC的**重点**内容，我们主要会讲解四部分内容：

- 请求映射路径
- 请求参数
- 日期类型参数传递
- 响应json数据

请求路径设置好后，只要确保页面发送请求地址和后台Controller类中配置的路径一致，就可以接收到前端的请求，接收到请求后，如何接收页面传递的参数？

关于请求参数的传递与接收是和请求方式有关系的，目前比较常见的两种请求方式为：

- GET
- POST

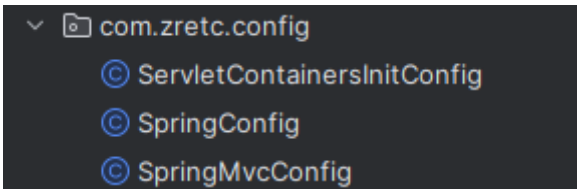
针对于不同的请求前端如何发送，后端如何接收？

5、请求映射路径

导入依赖

```
<dependencies>
        <!--    springmvc    -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
            <version>5.0.5.RELEASE</version>
        </dependency>
        <!--    servlet 请求和响应    -->
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>javax.servlet-api</artifactId>
            <version>4.0.1</version>
            <scope>provided</scope>
        </dependency>
    </dependencies>
```

导入配置类：



```
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.FilterType;
import org.springframework.stereotype.Controller;

@Configuration
// 一：精准扫描，service,dao
//@ComponentScan({"com.zretc.service","com.zretc.dao"})
// 二：扫描全包，排除controller
@ComponentScan(value = "com.zretc",
        excludeFilters = @ComponentScan.Filter(
                type = FilterType.ANNOTATION,
                classes = Controller.class
        ))
public class SpringConfig {
}



// springmvc的配置文件
@Configuration //配置文件
@ComponentScan("com.zretc.controller")
public class SpringMvcConfig {
}


// 相当于 web.xml
public class ServletContainersInitConfig extends
AbstractAnnotationConfigDispatcherServletInitializer {
    @Override
    protected Class<?>[] getRootConfigClasses() {
        return new Class[]{SpringConfig.class}; // spring 配置类
    }

    @Override
    protected Class<?>[] getServletConfigClasses() {
        return new Class[]{SpringMvcConfig.class}; // springmvc配置类
    }

    @Override
    protected String[] getServletMappings() {// 由springmvc控制器处理的请求映射路径
        return new String[]{"/"}; // / 任意请求都会被拦截
    }
}
```

index.jsp中页面跳转，发送请求，跳转到登录页面

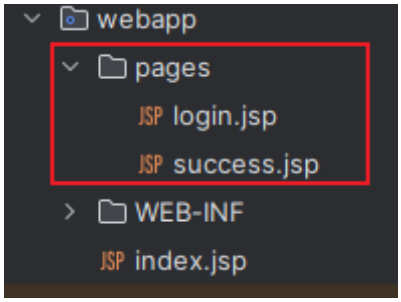
```
<html>
<body>
<%-- 发送请求 --%>
<%
    response.sendRedirect("/hello");
%>
</body>
</html>
```

HelloController

```
@Controller
public class HelloController {
    // 映射请求路径，处理请求
    @RequestMapping("/hello")
    public String hello(){
        System.out.println("hello哈哈哈");
        // 跳转到登录页面
        return "/pages/login.jsp";
    }

}
```

新建pages目录，创建login.jsp 与 success.jsp页面



login.jsp

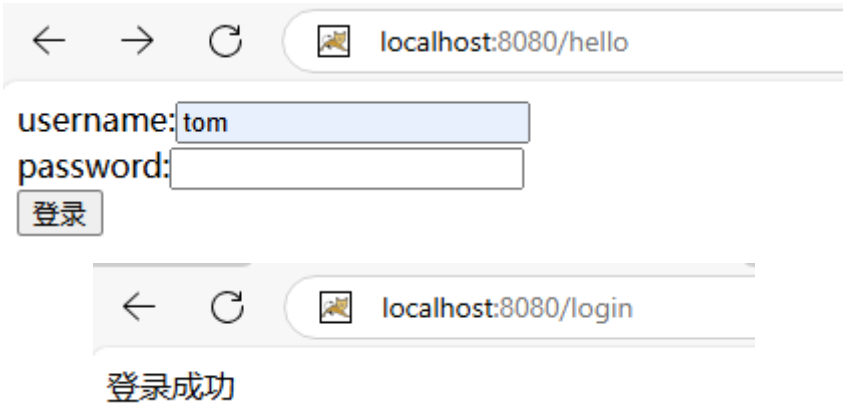
```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>
    <form action="/login" method="post">
        username:<input type="text" name="username"/><br>
        password:<input type="password" name="password"/><br>
        <input type="submit" value="登录">
    </form>
</body>
</html>
```

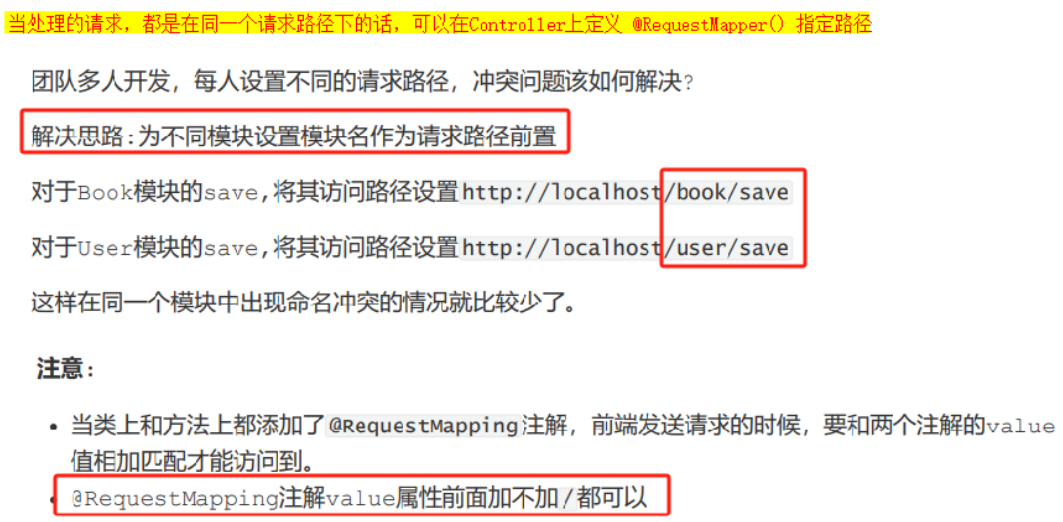
success.jsp

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

```java
@Controller
public class HelloController {
    // 映射请求路径，处理请求
    @RequestMapping("/hello")
    public String hello(){
        System.out.println("hello哈哈哈");
        // 跳转到登录页面
        return "/pages/login.jsp";
    }

    @RequestMapping("/login")
    public String login(HttpServletRequest request){
        String username = request.getParameter("username");
        System.out.println("username = " + username);
        // 获取请求参数
        // 访问service
        return "/pages/success.jsp";
    }
}
```





用户下有登录、注册等功能都是 AdminController 处理的请求，可以加一个通用的请求路径，在控制层上

index.jsp

```
<html>
<body>
<%-- 发送请求 --%>
<%
    response.sendRedirect("/admin/hello");
%>
</body>
</html>
```

login.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>
    <form action="/admin/login" method="post">
        username:<input type="text" name="username"/><br>
        password:<input type="password" name="password"/><br>
        <input type="submit" value="登录">
    </form>
</body>
</html>
```

HelloController

```
@Controller
@RequestMapping("/admin/")
public class HelloController {
    // 映射请求路径，处理请求
    @RequestMapping("hello")
    public String hello(){
        System.out.println("hello哈哈哈");
        // 跳转到登录页面
        return "/pages/login.jsp";
    }

    @RequestMapping("login")
    public String login(HttpServletRequest request){
        String username = request.getParameter("username");
        System.out.println("username = " + username);
        // 获取请求参数
        // 访问service
        return "/pages/success.jsp";
    }
}
```

5.1 视图解析器

指定视图的前缀与后缀，跳转页面时，只需要指定页面的名称

在SpringMvcConfig 中配置视图解析器

```
// springmvc的配置文件
@Configuration //配置文件
@ComponentScan("com.zretc.controller")
@EnableWebMvc
public class SpringMvcConfig implements WebMvcConfigurer {
    // 视图解析器

    @Override
    public void configureViewResolvers(ViewResolverRegistry registry) {
        // 指定页面的前缀和后缀
        registry.jsp("/pages/",".jsp");
    }
}
```

控制器中只需要指定页面的名称，省略前缀与后缀

```
@Controller
@RequestMapping("/admin/")
public class HelloController {
    // 映射请求路径，处理请求
    @RequestMapping("hello")
    public String hello(){
        System.out.println("hello哈哈哈");
        // 跳转到登录页面
//        return "/pages/login.jsp";
        return "login"; // 去掉前缀和后缀
    }

    @RequestMapping("login")
    public String login(HttpServletRequest request){
        String username = request.getParameter("username");
        System.out.println("username = " + username);
        // 获取请求参数
        // 访问service
//        return "/pages/success.jsp";
        return "success";
    }
}
```

5.2 设置静态资源

6、获取请求参数与传递参数

前面我们已经能够使用GET或POST来发送请求和数据，所携带的数据都是比较简单的数据，接下来在这个基础上，我们来研究一些比较负责的参数传递，常见的参数种类有：

- 普通参数
- POJO类型参数
- 嵌套POJO类型参数
- 数组类型参数
- 集合类型参数

6.1 使用request对象传递参数

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

jsp页面的中的el表达式默认情况是关闭的，需要手动打开

success.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ page isELIgnored="false" %>  <%--默认el表达式是关闭状态，需要手动代开--%>
<html>
<head>
    <title>Title</title>
</head>
<body>
登录成功,欢迎您：${username}
</body>
</html>
```

6.2 使用ModelAndView 传递请求参数

```
    // 使用ModelAndView 存储数据，指定跳转的页面
    @RequestMapping("login")
    // 表单中的数据自动赋值给函数的参数上
    public ModelAndView login(String username){
        ModelAndView mav = new ModelAndView("success");// 构造参数是跳转的页面名称
        // 存储数据
        mav.addObject("username",username);
         // 设置跳转的路径
//         mav.setViewName("success");
        return mav;
    }
```

注意：表单中的数据自动赋值给函数的参数上
　　函数的参数名称必须要与表单中的name属性相同，才能自动映射数据。

@RequestParam 注解，获取请求参数

> 当表单中的name属性与方法的形参不同时，可以使用@RequestParam映射

将 login.jsp 中的name属性改了

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>
    <form action="/admin/login" method="get">
        username:<input type="text" name="admin_name"/><br>
        password:<input type="password" name="admin_password"/><br>
        <input type="submit" value="登录">
    </form>
</body>
</html>
```

```
    // 使用ModelAndView 存储数据，指定跳转的页面
    @RequestMapping("login")
    // 当表单中的name属性与方法的形参不同时，可以使用@RequestParam映射
    public ModelAndView login(@RequestParam("admin_name") String username){
        ModelAndView mav = new ModelAndView("success");// 构造参数是跳转的页面名称
        // 存储数据
        mav.addObject("username",username);
        // 设置跳转的路径
//        mav.setViewName("success");
        return mav;
    }
```

6.3 使用Model对象传递请求参数

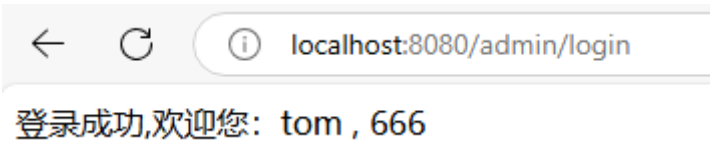
> 将表单中的数据直接映射给实体类中的成员变量
> 前提：表单的name属性必须与实体类的变量名相同

login.jsp 的name属性必须与实体类变量名一致

```
    // 使用Model传递参数
    @RequestMapping("login")
    // 将表单中的数据直接映射给实体类中的成员变量
    // 前提：表单的name属性必须与实体类的变量名相同
    public String login(Admin admin, Model model){
//        model.addAttribute("username",admin.getAdmin_name());
        // 直接将对象添加到model中
        model.addAttribute("admin",admin);
        return "success";
    }
```

success.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ page isELIgnored="false" %>  <%--默认el表达式是关闭状态，需要手动代开--%>
<html>
<head>
    <title>Title</title>
</head>
<body>
<%--登录成功,欢迎您：${username}--%>
登录成功,欢迎您：${admin.admin_name} , ${admin.admin_password}
</body>
</html>
```



请求时的中文乱码;

```
<form action="/admin/login" method="post">
    username:<input type="text" name="admin_name"/><br>
    password:<input type="password" name="admin_password"/><br>
    <input type="submit" value="登录">
</form>
```

在核心配置文件中注册过滤器;

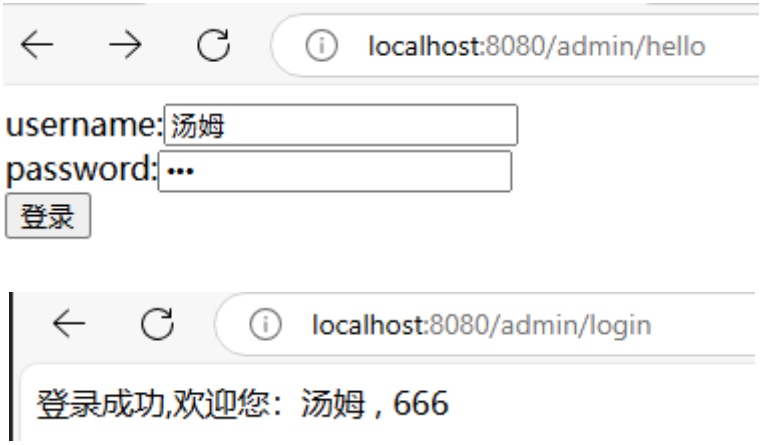
```
// 相当于 web.xml
public class ServletContainersInitConfig extends
AbstractAnnotationConfigDispatcherServletInitializer {
    @Override
    protected Class<?>[] getRootConfigClasses() {
        return new Class[]{SpringConfig.class}; // spring 配置类
    }

    @Override
    protected Class<?>[] getServletConfigClasses() {
```

```
        return new Class[]{SpringMvcConfig.class}; // springmvc配置类
    }

    @Override
    protected String[] getServletMappings() {// 由springmvc控制器处理的请求映射路径
        return new String[]{"/"}; // / 任意请求都会被拦截
    }

    // 注册过滤器--解决post请求的中文乱码问题
    @Override
    protected Filter[] getServletFilters() {
        CharacterEncodingFilter character = new CharacterEncodingFilter();
        character.setEncoding("utf-8");
        return new Filter[]{character};
    }
```



6.4 使用ModelMap传递请求参数

```
    // 使用ModelMap传递参数
    @RequestMapping("login")
    // 将表单中的数据直接映射给实体类中的成员变量
    // 前提：表单的name属性必须与实体类的变量名相同
    public String login(Admin admin, ModelMap model){
//        model.addAttribute("username",admin.getAdmin_name());
        // 直接将对象添加到model中
        model.addAttribute("admin",admin);
        return "success";
    }
```

6.5 直接通过实体类映射

```
    // 使用实体类映射
    @RequestMapping("login")
    // 将表单中的数据直接映射给实体类中的成员变量
    // 前提：表单的name属性必须与实体类的变量名相同
    public String login(Admin admin){
        return "success";
    }
```

@ModelAttribute 给实体类起别名，在前端中使用别名获取请求参数，如果没有起别名，默认就是类名首字母小写。

```
    // 使用实体类映射
    @RequestMapping("login")
    // 将表单中的数据直接映射给实体类中的成员变量
    // 前提：表单的name属性必须与实体类的变量名相同
    public String login(@ModelAttribute("aa") Admin admin){
        return "success";
    }
```

success.jsp

```
<body>
<%--登录成功,欢迎您：${username}--%>
登录成功,欢迎您：${aa.admin_name} , ${aa.admin_password}
</body>
```

6.6 嵌套pojo参数

请求参数名与形参对象属性名相同，按照对象层次结构关系即可接收嵌套POJO属性参数

Address

```
@Data
public class Address {
    private String city;
}
```

Admin

```
@Data
public class Admin {
    private String admin_name;
    private String admin_password;
    // 用户的地址
    private Address address;
}
```

login.jsp

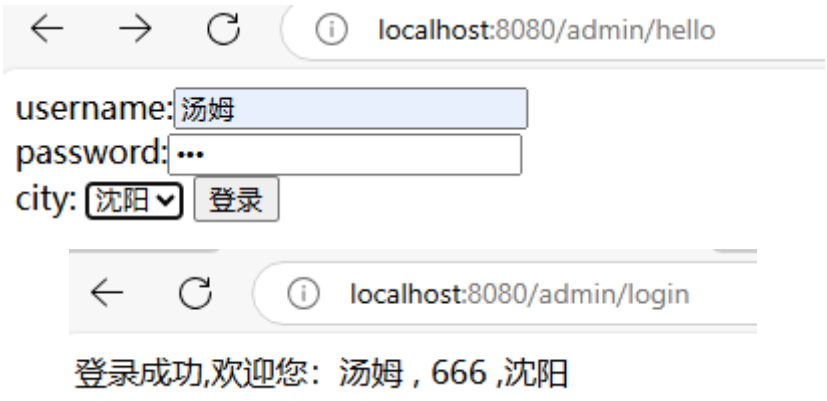
```
<body>
    <form action="/admin/login" method="post">
        username:<input type="text" name="admin_name"/><br>
        password:<input type="password" name="admin_password"/><br>
        city:
        <select name="address.city">
            <option value="大连">大连</option>
            <option value="鞍山">鞍山</option>
            <option value="沈阳">沈阳</option>
        </select>
        <input type="submit" value="登录">
    </form>
</body>
```

HelloController

```
    // 使用实体类映射
    @RequestMapping("login")
    // 将表单中的数据直接映射给实体类中的成员变量
    // 前提：表单的name属性必须与实体类的变量名相同
    public String login(@ModelAttribute("aa") Admin admin){
        return "success";
    }
```

success.jsp

```
<body>
<%--登录成功,欢迎您：${username}--%>
登录成功,欢迎您：${aa.admin_name} , ${aa.admin_password} ,${aa.address.city}
</body>
```



6.7 使用数组类型的参数传递参数

举个简单的例子，如果前端需要获取用户的爱好，爱好绝大多数情况下都是多个，如何发送请求数据和接收数据呢？

- 数组参数：请求参数名与形参对象属性名相同且请求参数为多个，定义数组类型即可接收参数


```
@Data
public class Admin {
    private String admin_name;
    private String admin_password;
    // 用户的地址
    private Address address;
    // 兴趣爱好
    private String[] like;
}
```

login.jsp

```
<body>
    <form action="/admin/login" method="post">
        username:<input type="text" name="admin_name"/><br>
        password:<input type="password" name="admin_password"/><br>
        city:
        <select name="address.city">
            <option value="大连">大连</option>
            <option value="鞍山">鞍山</option>
            <option value="沈阳">沈阳</option>
        </select>
        <br>
        兴趣:
        <input type="checkbox" name="like" value="睡觉">睡觉
        <input type="checkbox" name="like" value="看书">看书
        <input type="checkbox" name="like" value="吃饭">吃饭
        <br>
        <input type="submit" value="登录">
    </form>
</body>
```

success.jsp

```
<body>
<%--登录成功,欢迎您：${username}--%>
登录成功,欢迎您：${aa.admin_name} , ${aa.admin_password} ,
${aa.address.city}, ${aa.like}
<%
    String[] like = request.getParameterValues("like");
    for (String ll : like) {
%>
        <p><%=ll%></p>
<%
    }
%>
</body>
```

登录成功,欢迎您：汤姆 , 666 , 大连, [Ljava.lang.String;@5078f0c8

看书

吃饭

也可以直接映射到数组类型的形参中，前提是形参的名字必须与表单的name一致

```
    // 使用实体类映射
    @RequestMapping("login")
    // 将checkbox中选中的数据直接存储到like这个数组中
    public String login(String[] like){
        return "success";
    }
```

登录成功,欢迎您：,,,

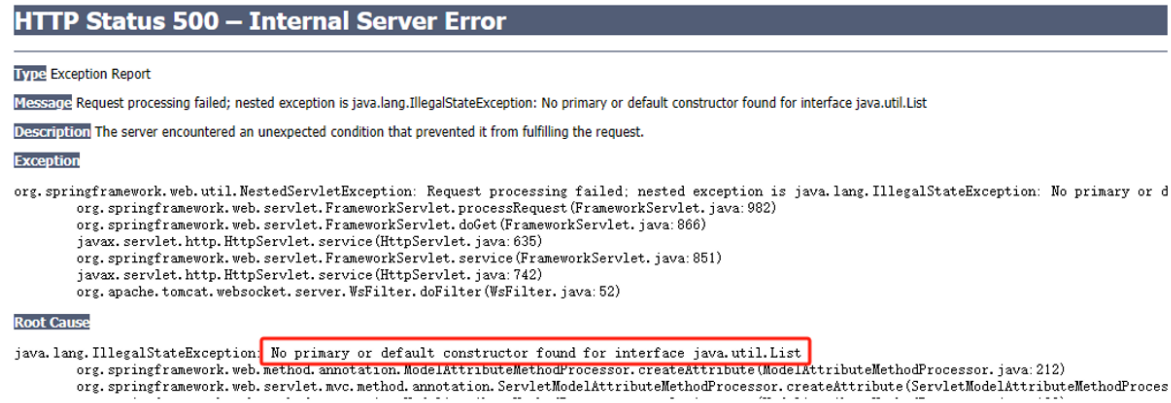
睡觉

看书

对象中就没有数据了，
因为是直接映射给了形参，不是实体类

6.8 使用集合映射数据

```
    //使用集合映射
    @RequestMapping("login")
    // 将checkbox中选中的数据直接存储到like这个数组中
    public String login(List<String> like){
        return "success";
    }
```



错误的原因是：SpringMVC将List看做是一个POJO对象来处理，将其创建一个对象并准备把前端的数据封装到对象中，但是List是一个接口无法创建对象，所以报错。

解决方案：使用 @RequestParam

```
//使用集合映射
@RequestMapping("login")
// 将checkbox中选中的数据直接存储到like这个数组中
public String login(@RequestParam List<String> like){
    return "success";
}
```

登录成功,欢迎您：,,,

睡觉

吃饭

- 集合保存普通参数：请求参数名与形参集合对象名相同且请求参数为多个，@RequestParam绑定参数关系
- 对于简单数据类型使用数组会比集合更简单些。

名称	@RequestParam
类型	形参注解
位置	SpringMVC控制器方法形参定义前面
作用	绑定请求参数与处理器方法形参间的关系
相关参数	required：是否为必传参数 defaultValue：参数默认值