任务一： SpringMVC基本应用

课程任务主要内容：

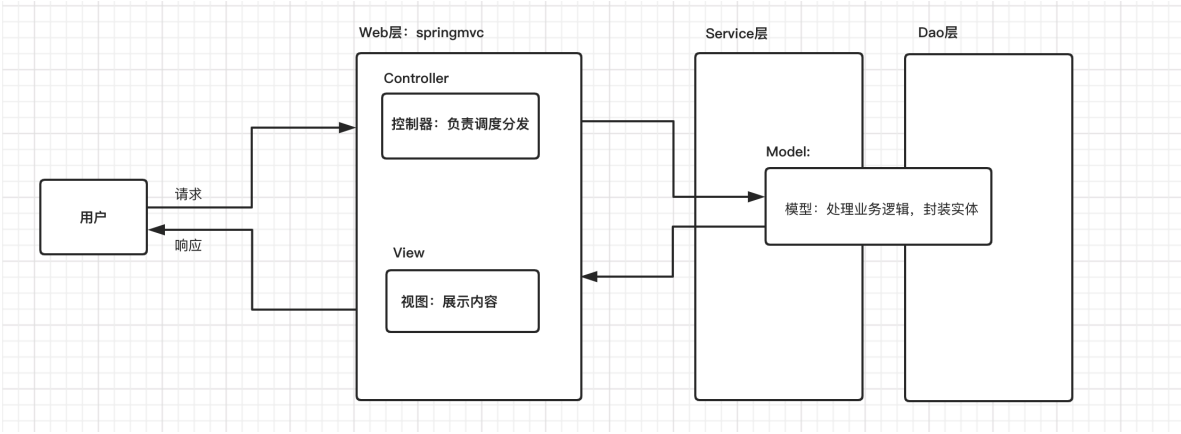
```
* SpringMVC简介
* SpringMVC组件概述
* SpringMVC请求
* SpringMVC响应
* 静态资源开启
```

一 SpringMVC简介

1.1 MVC模式

MVC是软件工程中的一种软件架构模式，它是一种分离业务逻辑与显示界面的开发思想。

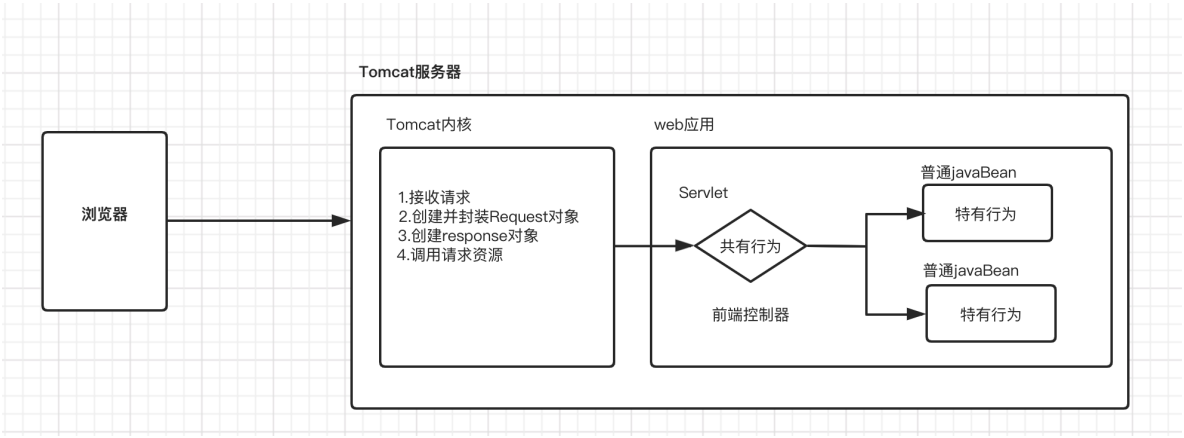
```
* M（model）模型：处理业务逻辑，封装实体

* V（view） 视图：展示内容

* C（controller）控制器：负责调度分发（1.接收请求、2.调用模型、3.转发到视图）
```



1.2 SpringMVC概述

SpringMVC 是一种基于 Java 的实现 MVC 设计模式的轻量级 Web 框架，属于SpringFrameWork 的后续产品，已经融合在 Spring Web Flow 中。

SpringMVC 已经成为目前最主流的MVC框架之一，并且随着Spring3.0 的发布，全面超越 Struts2，成为最优秀的 MVC 框架。它通过一套注解，让一个简单的 Java 类成为处理请求的控制器，而无须实现任何接口。同时它还支持 RESTful 编程风格的请求。



总结

SpringMVC的框架就是封装了原来Servlet中的共有行为；例如：参数封装，视图转发等。

1.3 SpringMVC快速入门

需求

客户端发起请求，服务器接收请求，执行逻辑并进行视图跳转。

步骤分析

1. 创建web项目，导入SpringMVC相关坐标
2. 配置SpringMVC前端控制器 DispathcerServlet
3. 编写Controller类和视图页面
4. 使用注解配置Controller类中业务方法的映射地址
5. 配置SpringMVC核心文件 spring-mvc.xml

1）创建web项目，导入SpringMVC相关坐标

```
<!-- 设置为web工程 -->
<packaging>war</packaging>

<dependencies>
    <!--springMVC坐标-->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-webmvc</artifactId>
        <version>5.1.5.RELEASE</version>
    </dependency>
    <!--servlet坐标-->
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>javax.servlet-api</artifactId>
        <version>3.1.0</version>
        <scope>provided</scope>
```

```
    </dependency>
    <!--jsp坐标-->
    <dependency>
        <groupId>javax.servlet.jsp</groupId>
        <artifactId>jsp-api</artifactId>
        <version>2.2</version>
        <scope>provided</scope>
    </dependency>
</dependencies>
```

2）配置SpringMVC前端控制器DispathcerServlet

```
<?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_3_1.xsd"
         version="3.1">

    <!--前端控制器-->
    <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:spring-mvc.xml</param-value>
        </init-param>
        <load-on-startup>2</load-on-startup>
    </servlet>

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

</web-app>
```

3）编写Controller类和视图页面

UserController.java

```
public class UserController {

    public String quick() {
        System.out.println("quick running.....");
        return "/WEB-INF/pages/success.jsp";
    }
}
```

/WEB-INF/pages/ success.jsp

```jsp
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>success</title>
</head>
<body>
    <h3>请求成功! </h3>
</body>
</html>
```

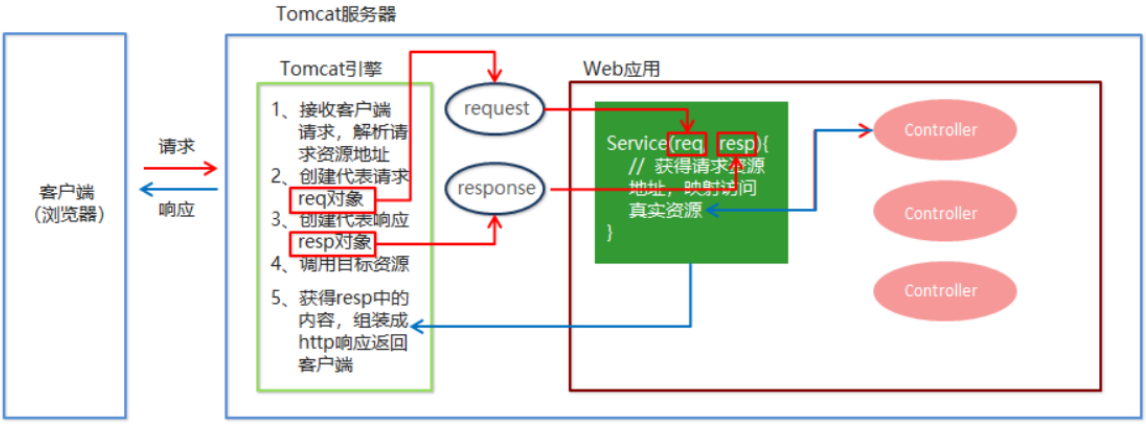
4）使用注解配置Controller类中业务方法的映射地址

```java
@Controller
public class UserController {

    @RequestMapping("/quick")
    public String quick() {
        System.out.println("quick running.....");
        return "/WEB-INF/pages/success.jsp";
    }
}
```

5）配置SpringMVC核心文件spring-mvc.xml

```xml
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xmlns:context="http://www.springframework.org/schema/context"
    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
    http://www.springframework.org/schema/mvc
    http://www.springframework.org/schema/mvc/spring-mvc.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context.xsd">

    <!--配置注解扫描-->
    <context:component-scan base-package="com.lagou.controller"/>

</beans>
```

1.4 web工程执行流程



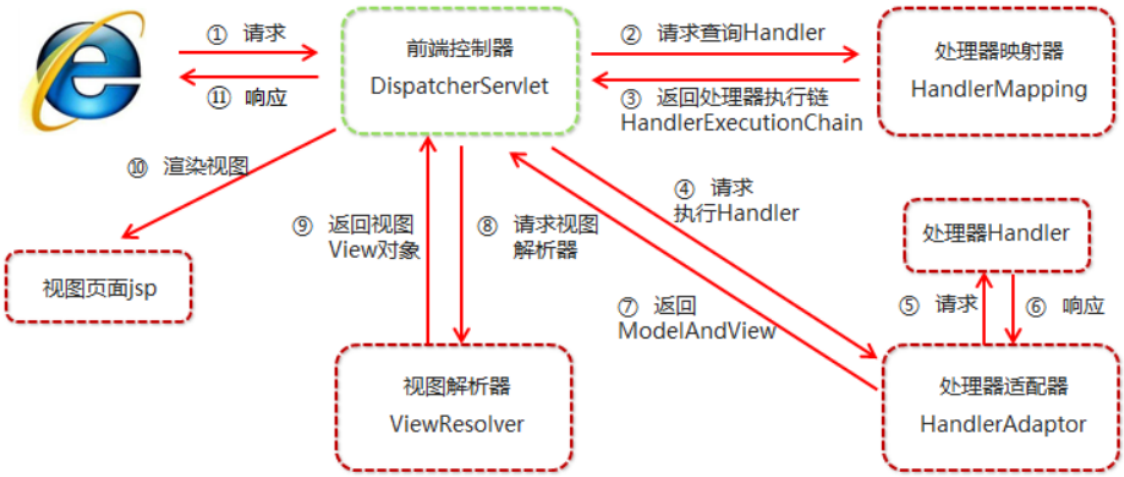
1.5 知识小结

* SpringMVC是对MVC设计模式的一种实现，属于轻量级的WEB框架。

* SpringMVC的开发步骤：
  1.创建web项目，导入SpringMVC相关坐标
  2.配置SpringMVC前端控制器 DispathcerServlet
  3.编写Controller类和视图页面
  4.使用注解配置Controller类中业务方法的映射地址
  5.配置SpringMVC核心文件 spring-mvc.xml

二 SpringMVC组件概述

2.1 SpringMVC的执行流程



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(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将渲染后的视图响应响应用户。

2.2 SpringMVC组件解析

1. 前端控制器：DispatcherServlet
    用户请求到达前端控制器，它就相当于 MVC 模式中的 C，DispatcherServlet 是整个流程控制的中心，由它调用其它组件处理用户的请求，DispatcherServlet 的存在降低了组件之间的耦合性。

2. 处理器映射器：HandlerMapping
    HandlerMapping 负责根据用户请求找到 Handler 即处理器，SpringMVC 提供了不同的映射器实现不同的映射方式，例如：配置文件方式，实现接口方式，注解方式等。

3. 处理器适配器：HandlerAdapter
    通过 HandlerAdapter 对处理器进行执行，这是适配器模式的应用，通过扩展适配器可以对更多类型的处理器进行执行。

4. 处理器：Handler【**开发者编写**】
    它就是我们开发中要编写的具体业务控制器。由 DispatcherServlet 把用户请求转发到Handler。由Handler 对具体的用户请求进行处理。

5. 视图解析器：ViewResolver
    View Resolver 负责将处理结果生成 View 视图，View Resolver 首先根据逻辑视图名解析成物理视图名，即具体的页面地址，再生成 View 视图对象，最后对 View 进行渲染将处理结果通过页面展示给用户。

6. 视图：View 【**开发者编写**】
    SpringMVC 框架提供了很多的 View 视图类型的支持，包括：jstlView、freemarkerView、pdfView等。最常用的视图就是 jsp。一般情况下需要通过页面标签或页面模版技术将模型数据通过页面展示给用户，需要由程序员根据业务需求开发具体的页面。

    ***笔试题：springmvc中的三大组件是什么？*

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xmlns:context="http://www.springframework.org/schema/context"
       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
    http://www.springframework.org/schema/mvc
    http://www.springframework.org/schema/mvc/spring-mvc.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context.xsd">
```

```
    <!--配置注解扫描-->
    <context:component-scan base-package="com.lagou.controller"/>

    <!--处理器映射器和处理器适配器功能增强-->
    <mvc:annotation-driven></mvc:annotation-driven>

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

</beans>
```

2.3 SpringMVC注解解析

@Controller

SpringMVC基于Spring容器，所以在进行SpringMVC操作时，需要将Controller存储到Spring容器中，如果使用@Controller注解标注的话，就需要使用：

```
<!--配置注解扫描-->
<context:component-scan base-package="com.lagou.controller"/>
```

@RequestMapping

```
* 作用：
    用于建立请求 URL 和处理请求方法之间的对应关系

* 位置：
    1.类上：请求URL的第一级访问目录。此处不写的话，就相当于应用的根目录。写的话需要以/开头。
        它出现的目的是为了使我们的URL可以按照模块化管理：
            用户模块
                /user/add
                /user/update
                /user/delete
                ...
            账户模块
                /account/add
                /account/update
                /account/delete
    2.方法上：请求URL的第二级访问目录，和一级目录组成一个完整的 URL 路径。

* 属性：
    1.value：用于指定请求的URL。它和path属性的作用是一样的
    2.method：用来限定请求的方式
    3.params：用来限定请求参数的条件
     例如：params={"accountName"} 表示请求参数中必须有accountName
          pramss={"money!100"} 表示请求参数中money不能是100
```

2.4 知识小结

```
* SpringMVC的三大组件
    处理器映射器：HandlerMapping
    处理器适配器：HandlerAdapter
    视图解析器：View Resolver

* 开发者编写
    处理器：Handler
    视图：View
```

三 SpringMVC的请求

3.1 请求参数类型介绍

客户端请求参数的格式是：`name=value&name=value……`

服务器要获取请求的参数的时候要进行类型转换，有时还需要进行数据的封装

SpringMVC可以接收如下类型的参数：

- 基本类型参数
- 对象类型参数
- 数组类型参数
- 集合类型参数

3.2 获取基本类型参数

Controller中的业务方法的参数名称要与请求参数的name一致，参数值会自动映射匹配。并且能自动做类型转换；自动的类型转换是指从String向其他类型的转换。

```
<a href="${pageContext.request.contextPath}/user/simpleParam?id=1&username=杰克">
    基本类型
</a>
```

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

3.3 获取对象类型参数

Controller中的业务方法参数的POJO属性名与请求参数的name一致，参数值会自动映射匹配。

```
<form action="${pageContext.request.contextPath}/user/pojoParam" method="post">
    编号：<input type="text" name="id"> <br>
    用户名：<input type="text" name="username"> <br>
    <input type="submit" value="对象类型">
</form>
```

```
public class User {

    Integer id;
    String username;
    // setter getter...
}
```

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

3.4 中文乱码过滤器

当post请求时，数据会出现乱码，我们可以设置一个过滤器来进行编码的过滤。

```
<!--配置全局过滤的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>
</filter>
<filter-mapping>
    <filter-name>CharacterEncodingFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

3.5 获取数组类型参数

Controller中的业务方法数组名称与请求参数的name一致，参数值会自动映射匹配。

```jsp
<form action="${pageContext.request.contextPath}/user/arrayParam">
    编号：<br>
    <input type="checkbox" name="ids" value="1">1<br>
    <input type="checkbox" name="ids" value="2">2<br>
    <input type="checkbox" name="ids" value="3">3<br>
    <input type="checkbox" name="ids" value="4">4<br>
    <input type="checkbox" name="ids" value="5">5<br>
    <input type="submit" value="数组类型">
</form>
```

```java
@RequestMapping("/arrayParam")
public String arrayParam(Integer[] ids) {
    System.out.println(Arrays.toString(ids));
    return "success";
}
```

3.6 获取集合（复杂）类型参数

获得集合参数时，要将集合参数包装到一个POJO中才可以。

```jsp
<form action="${pageContext.request.contextPath}/user/queryParam" method="post">
    搜索关键字：
    <input type="text" name="keyword"> <br>
    user对象：
    <input type="text" name="user.id" placeholder="编号">
    <input type="text" name="user.username" placeholder="姓名"><br>
    list集合<br>
    第一个元素：
    <input type="text" name="userList[0].id" placeholder="编号">
    <input type="text" name="userList[0].username" placeholder="姓名"><br>
    第二个元素：
    <input type="text" name="userList[1].id" placeholder="编号">
    <input type="text" name="userList[1].username" placeholder="姓名"><br>
    map集合<br>
    第一个元素：
    <input type="text" name="userMap['u1'].id" placeholder="编号">
    <input type="text" name="userMap['u1'].username" placeholder="姓名"><br>
    第二个元素：
    <input type="text" name="userMap['u2'].id" placeholder="编号">
    <input type="text" name="userMap['u2'].username" placeholder="姓名"><br>
    <input type="submit" value="复杂类型">
</form>
```

```java
public class QueryVo {

    private String keyword;
    private User user;
    private List<User> userList;
    private Map<String, User> userMap;
}
```

```
@RequestMapping("/queryParam")
public String queryParam(QueryVo queryVo) {
    System.out.println(queryVo);
    return "success";
}
```

3.7 自定义类型转换器

SpringMVC 默认已经提供了一些常用的类型转换器，例如：客户端提交的字符串转换成int型进行参数设置，日期格式类型要求为：yyyy/MM/dd 不然的话会报错，对于特有的行为，SpringMVC提供了自定义类型转换器方便开发者自定义处理。

```
<form action="${pageContext.request.contextPath}/user/converterParam">
    生日：<input type="text" name="birthday">
    <input type="submit" value="自定义类型转换器">
</form>
```

```
public class DateConverter implements Converter<String, Date> {
    public Date convert(String dateStr) {
        //将日期字符串转换成日期对象 返回
        SimpleDateFormat format = new SimpleDateFormat("yyyy-MM-dd");
        Date date = null;
        try {
            date = format.parse(dateStr);
        } catch (ParseException e) {
            e.printStackTrace();
        }
        return date;
    }
}
```

```
<!--处理器映射器和适配器增强-->
<mvc:annotation-driven conversion-service="conversionService"></mvc:annotation-
driven>

<!--自定义转换器配置-->
<bean id="conversionService"
class="org.springframework.context.support.ConversionServiceFactoryBean">
    <property name="converters">
        <set>
            <bean class="com.lagou.converter.DateConverter"></bean>
        </set>
    </property>
</bean>
```

```
@RequestMapping("/converterParam")
public String converterParam(Date birthday) {
    System.out.println(birthday);
    return "success";
}
```

3.8 相关注解

@RequestParam

当请求的参数name名称与Controller的业务方法参数名称不一致时，就需要通过@RequestParam注解显示的绑定

```
<a href="${pageContext.request.contextPath}/user/findByPage?pageNo=2">
    分页查询
</a>
```

```
/*
   @RequestParam() 注解
      defaultValue 设置参数默认值
      name  匹配页面传递参数的名称
      required 设置是否必须传递参数，默认值为true；如果设置了默认值，值自动改为false
*/
@RequestMapping("/findByPage")
public String findByPage(@RequestParam(name = "pageNo", defaultValue = "1")
Integer pageNum, @RequestParam(defaultValue = "5") Integer pageSize) {
    System.out.println(pageNum);
    System.out.println(pageSize);
    return "success";
}
```

@RequestHeader

获取请求头的数据。

```
@RequestMapping("/requestHead")
public String requestHead(@RequestHeader("cookie") String cookie) {
    System.out.println(cookie);
    return "success";
}
```

@CookieValue

获取cookie中的数据。

```
@RequestMapping("/cookieValue")
public String cookieValue(@CookieValue("JSESSIONID") String jesessionId) {
    System.out.println(jesessionId);
    return "success";
}
```

3.9 获取Servlet相关API

SpringMVC支持使用原始ServletAPI对象作为控制器方法的参数进行注入，常用的对象如下：

```
@RequestMapping("/servletAPI")
public String servletAPI(HttpServletRequest request, HttpServletResponse
response, HttpSession session) {
    System.out.println(request);
    System.out.println(response);
    System.out.println(session);
    return "success";
}
```

四 SpringMVC的响应

4.1 SpringMVC响应方式介绍

页面跳转

1. 返回字符串逻辑视图
2. void原始ServletAPI
3. ModelAndView

返回数据

1. 直接返回字符串数据
2. 将对象或集合转为json返回（任务二演示）

4.2 返回字符串逻辑视图

直接返回字符串：此种方式会将返回的字符串与视图解析器的前后缀拼接后跳转到指定页面

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

4.3 void原始ServletAPI

我们可以通过request、response对象实现响应

```
@RequestMapping("/returnVoid")
public void returnVoid(HttpServletRequest request, HttpServletResponse response)
throws Exception {
    // 1.通过response直接响应数据
    response.setContentType("text/html;charset=utf-8");
    response.getWriter().write("拉勾网");
    ----------------------------------------
    request.setAttribute("username", "拉勾教育");
    // 2.通过request实现转发
    request.getRequestDispatcher("/WEB-INF/pages/success.jsp").forward(request,
response);
    ----------------------------------------
    // 3.通过response实现重定向
    response.sendRedirect(request.getContextPath() + "/index.jsp");
}
```

4.4 转发和重定向

企业开发我们一般使用返回字符串逻辑视图实现页面的跳转，这种方式其实就是请求转发。

我们也可以写成：forward转发

如果用了forward：则路径必须写成实际视图url，不能写逻辑视图。它相当于：

```
request.getRequestDispatcher("url").forward(request,response)
```

使用请求转发，既可以转发到jsp，也可以转发到其他的控制器方法。

```
@RequestMapping("/forward")
public String forward(Model model) {
    model.addAttribute("username", "拉勾招聘");
    return "forward:/WEB-INF/pages/success.jsp";
}
```

Redirect重定向

我们可以不写虚拟目录，springMVC框架会自动拼接，并且将Model中的数据拼接到url地址上

```
@RequestMapping("/redirect")
public String redirect(Model model) {
    model.addAttribute("username", "拉勾教育");
    return "redirect:/index.jsp";
}
```

4.5 ModelAndView

4.4.1 方式一

在Controller中方法创建并返回ModelAndView对象，并且设置视图名称

```
@RequestMapping("/returnModelAndView1")
public ModelAndView returnModelAndView1() {
    /*
        Model:模型 作用封装数据
        View: 视图 作用展示数据
    */
    ModelAndView modelAndView = new ModelAndView();
    //设置模型数据
    modelAndView.addObject("username", " lagou");
    //设置视图名称
    modelAndView.setViewName("success");
    return modelAndView;
}
```

4.4.2 方式二

在Controller中方法形参上直接声明ModelAndView，无需在方法中自己创建，在方法中直接使用该对象设置视图，同样可以跳转页面

```
@RequestMapping("/returnModelAndView2")
public ModelAndView returnModelAndView2(ModelAndView modelAndView) {
    //设置模型数据
    modelAndView.addObject("username", "itheima");
    //设置视图名称
    modelAndView.setViewName("success");
    return modelAndView;
}
```

4.6 @SessionAttributes

如果在多个请求之间共用数据，则可以在控制器类上标注一个 @SessionAttributes,配置需要在session中存放的数据范围，Spring MVC将存放在model中对应的数据暂存到 HttpSession 中。

注意：@SessionAttributes只能定义在类上

```
@Controller
@SessionAttributes("username") //向request域存入的key为username时，同步到session域中
public class UserController {

    @RequestMapping("/forward")
    public String forward(Model model) {
        model.addAttribute("username", "子慕");
        return "forward:/WEB-INF/pages/success.jsp";
    }

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

```
    }
}
```

4.7 知识小结

```
* 页面跳转采用返回字符串逻辑视图
    1.forward转发
        可以通过Model向request域中设置数据
    2.redirect重定向
        直接写资源路径即可，虚拟目录springMVC框架自动完成拼接

* 数据存储到request域中
    Model model
        model.addAttribute("username", "子慕");
```

五 静态资源访问的开启

当有静态资源需要加载时，比如jquery文件，通过谷歌开发者工具抓包发现，没有加载到jquery文件，原因是SpringMVC的前端控制器DispatcherServlet的url-pattern配置的是 /（缺省）,代表对所有的静态资源都进行处理操作，这样就不会执行Tomcat内置的DefaultServlet处理，我们可以通过以下两种方式指定放行静态资源：

方式一

```
<!--在springmvc配置文件中指定放行资源-->
<mvc:resources mapping="/js/**" location="/js/"/>
<mvc:resources mapping="/css/**" location="/css/"/>
<mvc:resources mapping="/img/**" location="/img/"/>
```

方式二

```
<!--在springmvc配置文件中开启DefaultServlet处理静态资源-->
<mvc:default-servlet-handler/>
```