尚马教育 JAVA 课程

springMVC 基础

文档编号：C05

创建日期： 2017-07-07

最后修改日期：2021-01-21

版 本 号：V3.5

电子版文件名：尚马教育-第三阶段-5.springMVC 基础.docx

文档修改记录：

更新日期	更新作者	更新说明	版本号
2017-07-30	张元林	初始版本	V1.0
2018-08-01	王绍成	Mybatis 版本更新	V2.0
2019-11-06	冯勇涛	课件格式以及课程深度加深	V3.0
2021-01-21	冯勇涛	优化课程深度，优化案例，调整顺序	V3.5

目录

知识点：

掌握 SpringMVC 的核心组件结构；
掌握 SpringMVC 环境的搭建方法；
掌握 SpringMVC 的处理流程；

1. springMVC 介绍

Spring Web MVC 是基于 Servlet API 构建的原始 Web 框架，并从一开始就包含在 Spring Framework 中。正式名称“Spring Web MVC”来自其源模块 spring-webmvc 的名称， 但它通常被称为“Spring MVC”。简而言之，springMVC 是对 servlet 的封装框架，避免了 servlet 的代码弊端，提高开发效率。

Spring 为展现层提供的基于 MVC 设计理念的优秀的 Web 框架，是目前最主流的 MVC 框架之一：

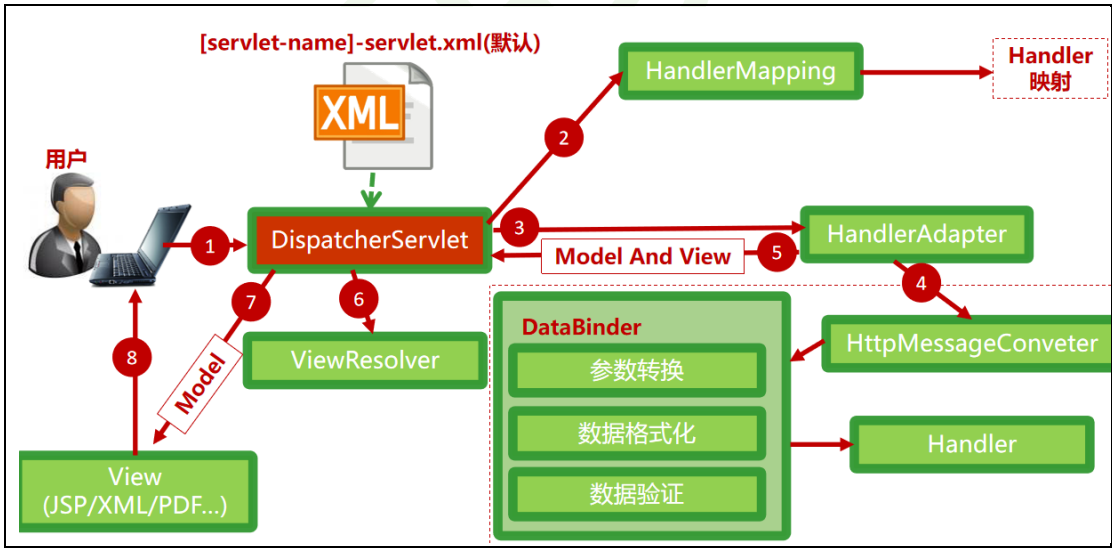
Spring3.0 后全面超越 Struts2，成为最优秀的 MVC 框架。

Spring MVC 通过一套 MVC 注解，让 POJO 成为处理请求的控制器，而无须实现任何接口。

支持 REST 风格的 URL 请求。

采用了松散耦合可插拔组件结构，比其他 MVC 框架更具扩展性和灵活性。

2. springMVC 执行原理图



3. 原理解析

1.客户端向服务器发送请求，请求被 SpringMVC 前端控制器 DispatcherServlet 捕获；

2.DispatcherServlet 对请求 URL 进行解析，得到请求资源标识符 URI，然后根据 URI 去调用 HandlerMapping 获得该 URI 对应的相关对象（包括 Handler 对象以及 Handler 对象对应的拦

截器），最后以 HandlerExecutionChain 对象返回；

3.DispatcherServlet 根据获得的 Handler 处理器，选择一个合适的 HandlerAdapter 处理器适配器，该适配器对象执行 Handler（获得 HandlerAdapter 后，将开始执行拦截器的前置拦截方法）；

4.提取 Request 中的请求数据，填充 Handler 入参，开始执行 Handler。在填充 Handler 的入参过程中，SpringMVC 将帮我们做一些很棒的工作：如通过 HttpMessageConveter 消息转化器对象生效：

- 将请求数据（如 Json 字符串）转换成对象，将响应对象转换为指定的数据格式。
- 数据转换：对请求数据进行格式转换。如 String 转换成 Integer、Double 等
- 数据格式化：对请求数据进行格式化。如将字符串转换成格式化数字或格式化日期等。
- 数据验证：验证数据的有效性（长度、格式等），验证结果存储 BindingResult 或 Error。

5. Handler 执行完成后，向 DispatcherServlet 返回一个 ModelAndView 对象；

6. 根据返回的 ModelAndView，选择一个适合的 ViewResolver（已注册到 SpringMVC 容器中的 ViewResolver)返回给 DispatcherServlet；

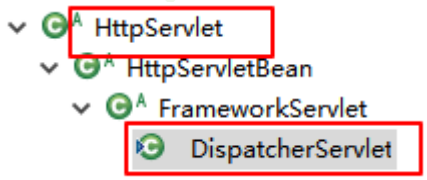
7. ViewResolver 结合 Model 和 View，来渲染视图；

8. 将渲染结果返回给客户端；

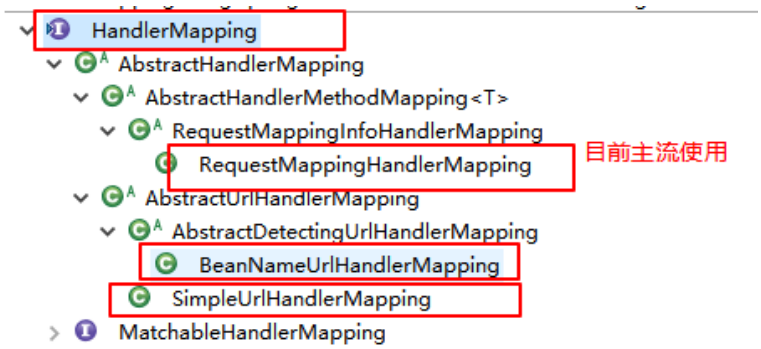
9. 响应结束。

4. 核心类与接口

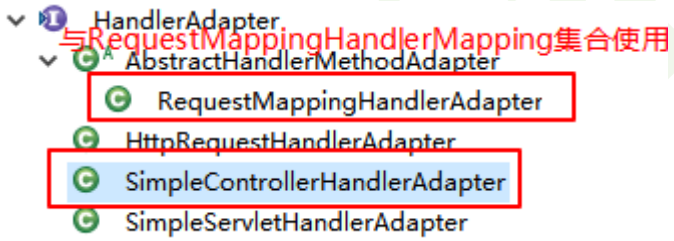
4.1. DispatcherServlet 前端控制器

类图： 
作用：接收请求，响应结果，相当于转发器，是 springMVC 的中央处理器。

4.2. HandlerMapping 处理器映射器

类图： 
作用：HandlerMapping 负责查找 Handler 处理器对象与对应的拦截器对象

4.3. HandlerAdapter 处理器适配器

类图： 
作用：找到 Handler 处理器对象后，由 DispatcherServlet 前端控制器调用 HandlerAdapter 来执行 HandlerMethod,Handler 处理器方法返回给适配器 ModelAndView 对象

4.4. Handler 处理器对象

需要开发人员自己编写，不需要继承任何接口。
作用：相当于原来的 Servlet 控制器

4.5. HttpMessageConverter 消息转换器对象

消息转换器主要两个作用，1.接收请求参数时，把请求参数转换成不同类型的对象（实体对象，Integer，Double 等）；2.返回响应数据时，把返回数据转成指定的数据格式（json 字符串）

4.6. ModelAndView 模型与视图对象

作用：是 springmvc 框架的一个底层对象，包括 Model 和 View，代表数据与视图部分。Handler 执行完成后，返回给 Adapter 的是 ModelAndView 对象。Adapter 再把该对象返回给 DispatherServlet 前端控制器。

4.7. ViewResolver 视图解析器对象

作用：前端控制器请求视图解析器去进行视图解析，根据逻辑视图名解析成真正的视图(jsp)，视图解析器向前端控制器返回 View，DispatcherServlet 负责渲染视图，将模型数据(在 ModelAndView 对象中)填充到 request 作用域，便于显示数据,最终响应用户。

5. springMVC 入门

5.1. 创建 web 项目，导入 springMVC 依赖

5.2. 创建 springmvc.xml 配置

在类路径下创建 springmvc.xml 配置文件，该文件时 spring 风格的配置，在该文件中配置处理器映射器，处理器适配器，视图解析器对象等。内容如下：

```
<!--开启 ioc 与 di 注解的识别-->
<context:component-scan base-package="com.javasm"></context:component-scan>
<!--处理器映射器对象：解析 bean 对象上的 RequestMapping 注解进行 url 映射-->
<bean class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping"></bean>
<!--处理器适配器对象：执行处理器方法-->
<bean class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter"></bean>
<!--视图解析器对象-->
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix" value="/page/"></property>
    <property name="suffix" value=".jsp"></property>
</bean>
```

注意点 1：必须配置<context:component-scan base-package="com.javasm"></context:component-scan>，用来识别基础的 spring 注解(Controller,Service,Repository,Component,Resource,Autowired 等)。

注意点 2：一般 HandlerMapping 与 HandlerAdapter 不手工配置，使用<mvc:annotation-driven></mvc:annotation-driven>标签替代，作用一致。如下：

```
<mvc:annotation-driven></mvc:annotation-driven>

<!--视图解析器对象-->

<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">

    <property name="prefix" value="/page/"></property>

    <property name="suffix" value=".jsp"></property>

</bean>
```

5.3. 配置前端控制器

在 Web.xml 中配置 DispatcherServlet，映射路径：/

```
<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:springMVC-config.xml</param-value>

    </init-param>

    <load-on-startup>1</load-on-startup>

  </servlet>

  <servlet-mapping>

    <servlet-name>dispatcherServlet</servlet-name>

    <url-pattern>/</url-pattern>

  </servlet-mapping>
```

5.4. 创建 page 目录

在 web 目录按照视图解析器路径创建 page 文件夹，并创建 main.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
    <head>
        <title>Title</title>
    </head>
    <body>
        hello springMVC!
    </body>
</html>
```

5.5. 编写 Handler 处理器类

```
@Controller
@RequestMapping("user")
public class SysuserHandler {
    @RequestMapping("main")
    public String main(){
        System.out.println("main 方法");
        return "main";
    }
```

注意点：这里的返回值字符串 main 是 jsp 视图的名称。

5.6. 部署项目启动 tomcat

启动 tomcat 后，浏览器访问 http://localhost:8080/user/main。

5.7. 入门总结

① @Controller

负责注册一个 bean 到 springMVC 上下文中。

② @RequestMapping

注解到类与方法上，springMVC 解析 requestMapping 注解，进行 uri 映射，一个 uri 映射到一个 HandlerMethod。

③ 返回值

Main 方法返回的”main”字符串，与视图解析器对象中的视图前缀：/page/，视图后缀：.jsp，组合完整视图路径:/page/main.jsp。

6. Kv 键值对数据处理

6.1. 接收 key=value 数据

6.1.1. 定义 login.jsp 与 main.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>
这里是 login.jsp
    <form action="user/login" method="post">
        用户名:<input type="text" name="uname">
        密码：<input type="password" name="upwd">
        <button type="submit">登录</button>
    </form>
</body>
</html>
```

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

6.1.2. springmvc 定义登录服务

```
@Controller
@RequestMapping("user")
public class SysuserHandler {
    //处理器方法添加形参，形参名与表单参数名一致。
    @RequestMapping(path = "login", method = RequestMethod.POST)
    public String dologin(String uname, String upwd) {
        System.out.println("用户名：" + uname + ",密码：" + upwd);
        if ("admin".equals(uname) && "upwd".equals(upwd)) {
            Sysuser u = new Sysuser();
            u.setUname(uname);
            u.setUpwd(upwd);
            //登录成功，进入 main 页面
            return "main";
        } else {
            //登录失败，进入 login 页面
            return "login";
        }
    }

@RequestMapping(path = "userlist",method = RequestMethod.GET)
public String userlist(Sysuser u, @RequestParam(defaultValue ="1") Integer pageNum,@RequestParam(defaultValue
= "10") Integer pageSize){
    System.out.println("userlist 方法"+u+"--"+pageNum+"--"+pageSize);
    return "main";
}
}
```

注意点：

注意点 1：RequestMapping 注解通过 method 属性限定请求方法；

注意点 2：表单参数名一定要与方法形参名一致；

注意点 3：登录服务的完整 uri：/user/login；

注意点 4：在 page 目录下要有对应的 main.jsp 与 login.jsp 视图。

注意点 5：可以通过 RequestParam 注解为形参指定默认值。但该注解只能用在简单类型的形参（String，Double，Integer 等），不能注解复杂对象类型。

6.1.3. 测试登录服务

启动 tomcat，浏览器访问：http://localhost:8080/login.jsp， 输入用户名与密码后点击登录按钮进行测试

6.2. 返回视图数据

如果后端服务需要返回数据给客户端进行展示，需要使用 Model 或 ModelAndView 对象，以下分别展示两个对象的使用方法：

```
//处理器方法添加形参，形参名与表单参数名一致。
@RequestMapping(path = "login", method = RequestMethod.POST)
public String dologin(String uname, String upwd, Model model) {
    System.out.println("用户名：" + uname + ",密码：" + upwd);
    if ("admin".equals(uname) && "upwd".equals(upwd)) {
        Sysuser u = new Sysuser();
        u.setUname(uname);
        u.setUpwd(upwd);
        //向 model 中加入数据等价于 request.setAttribute
        model.addAttribute("msg", "suc");
        model.addAttribute("loginuser", u);
        //登录成功，进入 main 页面
        return "main";
    } else {
        //登录失败，进入 login 页面
```

```
        model.addAttribute("msg", "error");
        return "login";
    }
}
```

```
//处理器方法添加形参，形参名与表单参数名一致。
@RequestMapping(path = "login", method = RequestMethod.POST)
public ModelAndView dologin(String uname, String upwd) {
    System.out.println("用户名：" + uname + ",密码：" + upwd);
    ModelAndView modelAndView = new ModelAndView();

    if ("admin".equals(uname) && "upwd".equals(upwd)) {
        Sysuser u = new Sysuser();
        u.setUname(uname);
        u.setUpwd(upwd);
        //向 model 中加入数据等价于 request.setAttribute
        modelAndView.addObject("msg", "suc");
        modelAndView.addObject("loginuser", u);
      //登录成功，进入 main 页面
        modelAndView.setViewName("main");
    } else {
        //登录失败，进入 login 页面
        modelAndView.setViewName("login");
        modelAndView.addObject("msg", "error");
    }
    return modelAndView;
}
```

6.3. 注意点

6.3.1. 注意点 1

一般后端服务接收表单参数，属于同一个实体类的数据，在方法的形参直接添加对应的形参对象，方法改造如下：

```
//处理器方法添加形参，形参名与表单参数名一致。
@RequestMapping(path = "login", method = RequestMethod.POST)
public ModelAndView dologin(Sysuser user) {
    System.out.println("用户名：" + user.getUname() + ",密码：" + user.getUpwd());
    ModelAndView modelAndView = new ModelAndView();

    if ("admin".equals(user.getUname()) && "upwd".equals(user.getUpwd())) {
        //向 model 中加入数据等价于 request.setAttribute
        modelAndView.addObject("msg", "suc");
        modelAndView.addObject("loginuser",user);
        //登录成功，进入 main 页面
        modelAndView.setViewName("main");
    } else {
        //登录失败，进入 login 页面
        modelAndView.setViewName("login");
        modelAndView.addObject("msg", "error");
    }
    return modelAndView;
}
```

6.3.2. 注意点 2

Springmvc 会对表单参数进行自动转型，不需要进行繁琐的 String 转 Double 等操作。

代码如下：

```
@GetMapping("test")
public String testCast(String uname,Double uprice){
    System.out.println(uname);
    System.out.println(uprice);
    return "main";
}
```

浏览器输入 url:http://localhost:8080/user/test?uname=fyt&uprice=100.12

6.3.3. 注意点 3

后端服务方法的形参类型不允许出现基本类型（byte,short,int,long 等），必须指定为包装类对象，因为包装类对象可以有默认值 null，基本类型无法指定默认值。

7. json 数据处理

目前前端主流的异步提交请求组件 ajax 与 axios。

- Ajax 提交默认的 ContentType 的值为:application/x-www-form-urlencoded; charset=UTF-8

  此格式为表单提交格式，数据为 key1=value1&key2=value2 的格式，以上章节后端服务获取数据方式适合前端使用 ajax 组件的情况。

- axios 提交默认的 ContentType 的值为:application/json;charset=utf-8，

  此格式为 json 提交格式，数据为{key1:value1,key2:value2}的格式，以上章节代码写法无法获取 json 格式数据。具体代码如下：

7.1. 加入 json 转换器

Springmvc 默认使用 jackson 组件作为 json 转换器。因此需要加入 jackson 相关的核心包。

7.2. 接收 json 数据

7.2.1. 定义响应枚举类与响应体对象

```
package com.javasm.sys.entity;

public enum Status {

    SUC(20000,"成功"),

    ERROR(50000,"失败"),

    ;

    private Integer code;

    private String msg;

    Status(Integer code, String msg) {

        this.code = code;

        this.msg = msg;

    }

    public Integer getCode() {

        return code;

    }



    public void setCode(Integer code) {

        this.code = code;

    }

    public String getMsg() {

        return msg;

    }

    public void setMsg(String msg) {

        this.msg = msg;

    }

}
```

```java
package com.javasm.sys.entity;


import com.fasterxml.jackson.annotation.JsonInclude;


//null 值不返回
@JsonInclude(JsonInclude.Include.NON_NULL)
public class ResultBean {
    private Integer code;
    private String msg;
    private Object data;
    public ResultBean(Status s, Object data) {
        this.code = s.getCode();
        this.msg = s.getMsg();
        this.data = data;
    }
    public ResultBean(Status s) {
        this.code = s.getCode();
        this.msg = s.getMsg();
    }
    public static ResultBean data(Status s,Object data){
        return new ResultBean(s,data);
    }
    public static ResultBean status(Status s){
        return new ResultBean(s);
    }
    public Integer getCode() {
        return code;
    }
    public void setCode(Integer code) {
```

```
        this.code = code;
    }
    public String getMsg() {
        return msg;
    }
    public void setMsg(String msg) {
        this.msg = msg;
    }
    public Object getData() {
        return data;
    }
    public void setData(Object data) {
        this.data = data;
    }
}
```

注意点：

注意点 1：@JsonInclude(JsonInclude.Include.*NON_NULL*)表示 null 值不返回前端；

7.2.2. 重新定义登录接口

由于异步提交，后端接口不需要进行视图的转发，仅需要返回 json 数据。因此不再执行视图解析器步骤，通过 json 消息转换器把方法的返回值转 json 字符串返回前端，springMVC 三种方法支持返回数据转 json 字符串。

7.2.2.1. 使用 ResponseBody 注解

```
//ResponseBody 表示当前方法返回值通过 json 转换器，转 json 字符串返回前端
@RequestMapping(path = "login", method = RequestMethod.POST)
@ResponseBody
public ResultBean dologin(@RequestBody Sysuser user) {
```

```
    System.out.println("收到数据："+user);
    if("admin".equals(user.getUname())&&"admin".equals(user.getUpwd())){
        user.setUid(100);
        return ResultBean.data(Status.SUC,user);
    }else{
        return ResultBean.status(Status.ERROR);
    }
}
```

注意点：RepsosneBody 注解还可以注解到类上，表示该类中所有方法返回值通过 json 转换器转字符串。

7.2.2.2. 使用 ResponseEntity 作为返回值类型

```
@RequestMapping(path = "login", method = RequestMethod.POST)
public ResponseEntity dologin(@RequestBody Sysuser user) {
    System.out.println("收到数据："+user);
    if("admin".equals(user.getUname())&&"admin".equals(user.getUpwd())){
        user.setUid(100);
        ResultBean data = ResultBean.data(Status.SUC,user);
        HttpHeaders headers = new HttpHeaders();
        headers.add("testHeader","自定义响应头");
        return new ResponseEntity(data,headers,HttpStatus.OK);
    }else{
        return ResponseEntity.ok(ResultBean.status(Status.ERROR));
    }
}
```

注意点： ResponseEntity 相比 ReposneBody 更加强大，能够灵活的指定响应头，响应体，状态行。

7.2.2.3. 使用 RestController 注解控制层 bean

RestController 表示当前类中所有方法的返回值通过 json 转换器转字符串。

```
@RestController

@RequestMapping("user")

public class SysuserHandler {}
```

注意点：用过 RestController 后，方法不再需要加 ResponseBody 注解。

7.2.3. 测试登录服务

以下所有接口均使用 postman 测试.

7.2.4. 注意点

前端传递数据包含日期格式，后端接口可以使用 String 或 Date 接收，String 接收则正常接收前端数据，Date 接收需要使用 JsonFormat 注解指定日期格式

```
public class Sysuser {

    private Integer uid;

    private String uname;

    private String upwd;

    private Integer uage;

    private String uphone;

    private Double uweight;

    @JsonFormat(pattern = "yyyy-MM-dd",timezone = "GMT+8")

    private Date udate;

}
```

8. 服务的转发与重定向

视图解析器默认是做服务端跳转到指定视图。

forward:跳转到指定 handlerMethod

redirect:重定向到指定 handlerMethod。

处理器结果返回字符串以 forward 或 redirect 结尾

```
    //处理器方法添加形参，形参名与表单参数名一致。
    @RequestMapping(path = "login", method = RequestMethod.POST)
    public String dologin(Sysuser user,Model model) {
        System.out.println("用户名：" + user.getUname() + ",密码：" + user.getUpwd());

        if ("admin".equals(user.getUname()) && "upwd".equals(user.getUpwd())) {
            //向 model 中加入数据等价于 request.setAttribute
            model.addAttribute("msg", "suc");
            model.addAttribute("loginuser",user);
            //登录成功，进入 main 页面
//            return "forward:main";//视图名是 forward:开头，表示转发到/user/main 服务中
            return "redirect:main";//视图名是 redirect:开头，表示重定向到/user/main
        } else {
            //登录失败，进入 login 页面
            model.addAttribute("msg", "error");
            return "login";
        }
    }

@RequestMapping("main")
public String main(){
    System.out.println("main 方法");
```

```
    return "main";

}
```

9. 数据乱码

9.1. get 提交

tomcat7 以下需要在 tomcat/conf/server.xml 中的 connector 标签下配置属性 URIEncoding="UTF-8"。

tomcat7 及以上不需要配置，url 默认支持中文数据,不会出现乱码。

9.2. post 提交

9.2.1. 提交 kv 键值对数据

Springmvc 对键值对数据的处理会产生乱码问题，需要指定编码格式。

在 web.xml 中配置 CharacterEncodingFilter 过滤器进行编码统一过滤，配置如下：

```
<filter>
  <filter-name>encodingFilter</filter-name>
  <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
  <init-param>
      <param-name>encoding</param-name>
      <param-value>UTF-8</param-value><!-- 与项目编码一致-->
  </init-param>
  <init-param>
      <param-name>forceResponseEncoding</param-name>
      <param-value>true</param-value>
  </init-param>
</filter>
<filter-mapping>
  <filter-name>encodingFilter</filter-name>
  <url-pattern>/*</url-pattern>
```

```
</filter-mapping>
```

9.2.2. 提交 json 格式数据

Springmvc 的提交的 json 数据处理通过 jackson 转换器，该转换器对象底层编码格式默认是 UTF-8，默认支持中文，不会出现乱码问题。因此建议使用 json 格式进行数据传输。

10. Servlet 对象的获取

springMVC 达到与 servlet 核心对象解耦，便于测试。

获取方式：以处理器方法形参方式进行传入，便于获取 HttpServletRequest，HttpServletResponse，HttpSession。

```
@RequestMapping("/dologin")
public String login(Sysuser u,@DateTimeFormat(pattern="yyyy-MM-dd")Date cd,HttpSession session) {
    session.setAttribute("sss", "*********session中的数据1******");
    System.out.println("------处理器方法login------"+u);
    return "main";
}
```

能够得到Request，Response，Session

11. Rest 风格 url

REST 风格：Representational State Transfer 表述性状态转移。

传统的查、改、删的 URL 与 REST 风格的增删改 URL 对比

传统 url	Rest 风格 URL	提交方法
查询操作 /user/getUserById?id=12	/user/12	Get，数据跟在 url 后面
删除操作 /user/deleteUserById?id=12	/user/12	DELETE，数据一般跟在 url 后，也可
添加操作 /user/addUser	/user	POST，数据在请求体
编辑操作 /user/updateUser?id=12	/user/12	PUT，数据再请求体或 url 后

请求方式：GET 负责查询、POST 负责添加、DELETE 负责删除、PUT 负责更新

他强调的是一个 url 资源可以对应多种视图

11.1. 常用注解

GetMapping、PostMapping、PutMapping、DeleteMapping

四种注解对应四种请求方式。

```
@RestController
@RequestMapping("/goods")
public class GoodsHandler {
   //@RequestMapping(path="{goodsId}",method=RequestMethod.GET)
   @GetMapping("{goodsId}")
   public String getGoods(@PathVariable("goodsId")int gid) {
      System.out.println("查询商品："+gid);
      return "查询商品："+gid;
   }
   @PostMapping
   public void addGoods(Goods g) {
      System.out.println("添加商品："+g);
```

```
    }

    @DeleteMapping("{goodsId}")

    public void delGoodsById(@PathVariable("goodsId")int gid) {

        System.out.println("删除商品：  "+gid);

    }

    @PutMapping("{goodsId}")

    public void updateGoodsById(@PathVariable("goodsId")int gid,Goods g) {

        System.out.println("更新商品：  "+g);

    }

}
```