第2天: SpringMVC

一、目标

1. 掌握自定义类型转换器
2. 了解高级参数的绑定
3. 了解方法返回值的用法
4. 掌握json数据交互
5. 了解RESTful风格支持

二、高级参数绑定

3.1 自定义参数类型

> 在实际项目中，比如日期类型或者货币数据，由于格式多不固定。springmvc框架不知道我们需要的格式，只能我们根据业务需求来转换。

```
/**
 * 特殊类型
 *  请求的时间类型转换失败，如：2019-10-30
 *  原因是国际的写法与国内有差异：10/30/2019
 * @param date 请求时间
 * @return
 */
@RequestMapping("hello")
public String hello(Date date){
    System.out.println(date);
    // 返回的页面路径
    return "success";
}
```

3.1.1 自定义转换器

- org.springframework.core.convert.converter.Converter

```
// 实现该接口可自定义转换类型
public interface Converter<S, T> {

    /**
     * @param source: 源参数
     * @return target: 转换后的目标参数
     */
    T convert(S source);

}
```

```java
package com.itheima.web;


import org.springframework.core.convert.converter.Converter;

import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;

/**
 * 自定义转换器 (实现转换接口).
 *  S: source,源
 *  T: target,目标
 *  (将s转换成t)
 */
public class DateConverter implements Converter<String, Date> {


    /**
     * 时间格式转换器
     * @param s 源参数
     * @return 目标类型
     */
    @Override
    public Date convert(String s) {
        SimpleDateFormat format = new SimpleDateFormat("yyyy-DD-mm");
        try {
            return format.parse(s);
        } catch (ParseException e) {
            e.printStackTrace();
        }
        return null;
    }
}
```

3.1.2 转换器配置

```xml
<!-- 1. 开启SpringMVC注解驱动 -->
<mvc:annotation-driven conversion-service="conversionServiceFactoryBean" />

<!-- 2. 注册转换工厂服务 -->
<bean id="conversionServiceFactoryBean"
class="org.springframework.format.support.FormattingConversionServiceFactoryBean">
    <property name="converters">
        <set>
            <!-- 时间转换器 -->
            <bean class="com.itheima.web.DateConverter"/>
        </set>
    </property>
</bean>
```

3.2.1 数组类型

- com.itheima.controller.ParamController

```java
package com.itheima.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

import java.util.Arrays;

/**
 * 高级参数类型的绑定案例
 */
@Controller
public class ParamController {


    /**
     * 绑定数组类型的参数
     * @param id 数组参数
     *     请求示例: http://localhost:8080/hello.do?id=1&id=2&id=3
     */
    @RequestMapping("hello")
    public String hello(Integer[] id){
        System.out.println(Arrays.toString(id));
        return "success";
    }
}
```

- index.jsp

```jsp
<%--
  Created by IntelliJ IDEA.
  User: Jason
  Date: 2019/7/24
  Time: 18:01
  To change this template use File | Settings | File Templates.
--%>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>测试页面</title>
</head>
<body>
    <h3>提交数组参数</h3>
    <form action="/hello.do" method="post">
        <input name="id" value="1"/>
        <input name="id" value="2"/>
        <input name="id" value="3"/>
        <input type="submit"/>
    </form>
</body>
</html>
```

直接绑定

- com.itheima.controller.ParamController

```
/**
 * 绑定数组类型的参数
 * @param id 数组参数
 *     请求示例: http://localhost:8080/hello.do?id=1&id=2&id=3
 *
 * 提示: SpringMVC绑定集合参数需要使用@RequestParam修饰, 详见源码(5.1.7.RELEASE)
 *      126:
org.springframework.web.method.annotation.RequestParamMethodArgumentResolver
 */
@RequestMapping("hello")
public String hello(@RequestParam ArrayList<Integer> id){
    System.out.println(id);
    return "success";
}
```

- index.jsp

```
<h3>提交集合参数</h3>
<%-- SpringMVC绑定集合属性需要使用@RequestParam修饰, 详见源码(5.1.7.RELEASE) --%>
<%-- 126:
org.springframework.web.method.annotation.RequestParamMethodArgumentResolver
--%>
<form action="/hello.do" method="post">
    <input name="id" value="1"/>
    <input name="id" value="2"/>
    <input name="id" value="3"/>
    <input type="submit"/>
</form>
```

包装绑定

- com.itheima.controller.User

```
package com.itheima.controller;

import java.util.Date;
import java.util.List;

/**
 * User.
 *
 * @author : Jason.lee
 * @version : 1.0
 */
public class User {
    private Integer id;
    private String username;
    private Date birthday;
    private String sex;
```

```java
    // jsp的valueName写法: order[0].id, order[0].name, order[1].id
    private List<Order> order;

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public Date getBirthday() {
        return birthday;
    }

    public void setBirthday(Date birthday) {
        this.birthday = birthday;
    }

    public String getSex() {
        return sex;
    }

    public void setSex(String sex) {
        this.sex = sex;
    }

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    public List<String> getOrders() {
        return orders;
    }

    public void setOrders(List<String> orders) {
        this.orders = orders;
    }

    public List<Order> getOrder() {
        return order;
    }

```

```
78
79      @Override
80      public String toString() {
81          return "User{" +
82                  "id=" + id +
83                  ", username='" + username + '\'' +
84                  ", birthday=" + birthday +
85                  ", sex='" + sex + '\'' +
86                  ", address='" + address + '\'' +
87                  '}';
88      }
89  }
```

- com.itheima.controller.ParamController

```
1       /**
2        * 绑定对象包装集合参数 【扩展1】
3        * @param user 对象
4        *     请求示例: http://localhost:8080/hello.do?orders=1&orders=2&orders=3
5        * 提示: 包装类(List<String>)中的泛型是基本类型直接对属性名赋值
6        */
7   //    @RequestMapping("hello")
8       public String hello(User user){
9           System.out.println(user.getOrders());
10          return "success";
11      }
12
13      /**
14       * 绑定对象包装集合参数 【扩展2】
15       * @param user 对象
16       *     请求示例: Tomcat不支持[]符号
17       * 提示: 包装类(List<Order>)中的泛型是对象类型需要指定下标对对象属性赋值
18       */
19  //    @RequestMapping("hello")
20      public String hello2(User user){
21          System.out.println(user.getOrder());
22          return "success";
23      }
```

- index.jsp

```
1   <h3>提交对象包装集合参数</h3>
2   <%-- 包装类(List<String>)中的泛型是基本类型直接对属性名赋值 --%>
3   <form action="/hello.do" method="post">
4       <input name="orders" value="1"/>
5       <input name="orders" value="2"/>
6       <input name="orders" value="3"/>
7       <input type="submit"/>
8   </form>
9   <h3>提交对象包装集合参数</h3>
10  <%-- 包装类(List<Order>)中的泛型是对象类型需要指定下标对对象属性赋值 --%>
11  <form action="/hello.do" method="post">
12      <input name="order[0].id" value="1"/>
13      <input name="order[2].name" value="2"/>
```

```
</form>
```

三、控制器方法的返回值

- com.itheima.controller.ReturnController

3.1 void

3.1.1 转发

- com.itheima.controller.ReturnController

```
/**
 * 接收ServletAPI参数, 调用原始API实现转发
 * @param request 请求对象
 * @param response 响应对象
 * @throws Exception
 */
@RequestMapping("hello")
public void hello(HttpServletRequest request, HttpServletResponse response)
throws Exception {
    // 获取请求调度器
    RequestDispatcher rd =
request.getRequestDispatcher("/pages/success.jsp");
    // 转发处理
    rd.forward(request, response);
}
```

3.1.2 重定向

- com.itheima.controller.ReturnController

```
/**
 * 接收ServletAPI参数, 调用原始API实现转发
 * @param request 请求对象
 * @param response 响应对象
 * @throws Exception
 */
@RequestMapping("hello")
public void hello(HttpServletRequest request, HttpServletResponse response)
throws Exception {
    // 获取请求调度器
    RequestDispatcher rd =
request.getRequestDispatcher("/pages/success.jsp");
    // 响应转发的资源
    // (HTTP是问答式协议, 一个请求只能一个响应)
    //        rd.forward(request, response);
    // 将响应重定向给其他资源响应
    response.sendRedirect("/index.jsp");
}
```

3.1.3 响应数据

```java
/**
 * 接收ServletAPI参数，调用原始API实现转发
 * @param request 请求对象
 * @param response 响应对象
 * @throws Exception
 */
@RequestMapping("hello")
public void hello(HttpServletRequest request, HttpServletResponse response)
throws Exception {
    // 获取请求调度器
    RequestDispatcher rd =
request.getRequestDispatcher("/pages/success.jsp");
    // 响应转发的资源
    // (HTTP是问答式协议，一个请求只能一个响应)
    //        rd.forward(request, response);
    // 将响应重定向给其他资源响应
    //        response.sendRedirect("/index.jsp");

    // 响应中文需要设置字符集编码
    response.setCharacterEncoding("utf-8");
    response.setContentType("application/json;charset=utf-8");
    // 模拟向浏览器输出的json字符串
    String str = "{\"id\":1,\"name\":\"小明\"}";
    response.getWriter().write(str);
}
```

3.2 String

3.2.1 转发

```java
/**
     * 接收SpringMVC数据模型对象
     * @return 响应的视图
     */
@RequestMapping("hello")
public String hello(){
    return "forward:/pages/success.jsp";
}
```

3.2.2 重定向

```java
/**
     * 接收SpringMVC数据模型对象
     * @return 响应的视图
     */
@RequestMapping("hello")
public String hello(Model model){
    //        return "forward:/pages/success.jsp";
    return "redirect:/pages/success.jsp";
}
```

3.2.3 响应数据

```
     *
     * @param model 数据模型
     * @return 响应的视图
     */
@RequestMapping("hello")
public String hello(Model model){
    // 往数据模型中存储数据
    // 最终调用的HttpServletRequest的setAttribute存储的数据
    model.addAttribute("name", "Jason");
    return "success";
}
```

3.3 ModelAndView

```
/**
     * ModelAndView:  模型与视图对象
     * （代替String与Model的写法）
     */
@RequestMapping("hello")
public ModelAndView hello(){
    // 创建模型与视图对象
    ModelAndView mv = new ModelAndView();
    // 设置视图名称
    mv.setViewName("success");
    // 设置响应数据
    mv.addObject("name", "Jason");
    return mv;
}
```

四、参数的相关注解【扩展】

- com.itheima.web.OtherController

4.1 @RequestParam

```
package com.itheima.web;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.CookieValue;
import org.springframework.web.bind.annotation.RequestHeader;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
@RequestMapping("other")
public class OtherController {

    /**
     * 基本数据类型: 参数不能为空(null)
     * 包装数据类型: 参数可以是空
     *
     * @RequestParam: 绑定指定名称的简单类型参数, 名称区分大小写
```

```
     */
    @RequestMapping("hello")
    public String hello(@RequestParam("id") Integer id
                , @RequestParam(defaultValue = "Jason") String name
                , @RequestParam(required = false) Integer sex
    ) {
            System.out.println(id);
            System.out.println(name);
            System.out.println(sex);
            // 返回的页面路径
            return "success";
    }
}
```

4.2 @RequestBody

```
@Controller
@RequestMapping("other")
public class OtherController {

    /**
     * @RequestBody: 修饰参数，提取请求体内容
     *  required: 默认值true，表示必须携带请求体
     *  【Json请求】
     *  请求体的的参数将会使用&串联成字符串: id=1&name=Jason
     *  配合Jackson组件使用可以转换成对象: User user (需要引入Jackson依赖)
     *
     */
    @RequestMapping("hello")
    public String hello(@RequestBody String body/*User user*/){
        System.out.println(body); //id=1&name=Jason
        return "success";
    }
}
```

- pages/commit.jsp

```
<%--
  Created by IntelliJ IDEA.
  User: Jason
  Date: 2019/7/18
  Time: 15:55
  To change this template use File | Settings | File Templates.
--%>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>COMMIT</title>
</head>
<body>
    <h4>POST方法</h4>
    <%-- 限定GET方法后,其他方法不能访问 --%>
    <form action="/account/hello.do" method="post">
        <input type="submit"/>
```

```
    <%-- get方法tomcat8以上没有乱码 --%>
    <%-- post方法又乱码问题 --%>
    <form action="/param/hello.do" method="post">
        <input name="id" value="1"/>
        <input name="name" value="中文"/>
        <input type="submit"/>
    </form>
    <h4>提交请求体内容</h4>
    <%-- 请求体包括所有name不为空的值 --%>
    <form action="/other/hello.do" method="post">
        <input name="id" value="1"/>
        <input name="name" value="Jason"/>
        <input type="submit"/>
    </form>
</body>
</html>
```

4.3 @RequestHeader

```
package com.itheima.web;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.CookieValue;
import org.springframework.web.bind.annotation.RequestHeader;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
@RequestMapping("other")
public class OtherController {


    /**
     * @RequestHeader: 提取请求中的头部信息, 名称不区分大小写
     *  value: 属性名称
     *  required: 是否必传参数 true默认值表示必须携带
     *  defaultValue: 默认值 当未传递此参数将采用默认值
     */
    @RequestMapping("/hello.do")
    public String hello(@RequestHeader(value = "cookie") String cookie
            , @RequestHeader(required = false) String host
            , @RequestHeader(defaultValue = "aaa") String a
    ){
        System.out.println(cookie);
        System.out.println(host);
        System.out.println(a);
        System.out.println(JSESSIONID);
        // 返回的页面路径
        return "success";
    }
}
```

4.4 @CookieValue

```java
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.CookieValue;
import org.springframework.web.bind.annotation.RequestHeader;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
@RequestMapping("other")
public class OtherController {


    /**
     * @RequestHeader: 提取请求中的头部信息, 名称不区分大小写
     * @CookieValue: 提取请求中的cookie信息, 名称区分大小写
     *  value: 属性名称
     *  required: 是否必传参数 true默认值表示必须携带
     *  defaultValue: 默认值 当未传递此参数将采用默认值
     */
    @RequestMapping("/hello.do")
    public String hello(@RequestHeader(value = "cookie") String cookie
            , @RequestHeader(required = false) String host
            , @RequestHeader(defaultValue = "aaa") String a
            , @CookieValue("JSESSIONID") String JSESSIONID
    ){
        System.out.println(cookie);
        System.out.println(host);
        System.out.println(a);
        System.out.println(JSESSIONID);
        // 返回的页面路径
        return "success";
    }
}
```

五、json数据交互

5.1 注解说明

5.1.1 @RequestBody

- 位置: 参数
- 作用: 将请求的json数据转换成对象

5.1.2 @ResponseBody

- 位置: 方法
- 作用: 将返回的对象转换成json数据

5.2 案例演示

- 添加依赖: pom.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>

```

```
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">
      <parent>
        <artifactId>mvc2</artifactId>
        <groupId>com.itheima</groupId>
        <version>1.0-SNAPSHOT</version>
      </parent>
      <modelVersion>4.0.0</modelVersion>
      <artifactId>mvc-day02-controller</artifactId>
      <packaging>war</packaging>
      <dependencies>
        <dependency>
          <groupId>org.springframework</groupId>
          <artifactId>spring-webmvc</artifactId>
          <version>5.1.7.RELEASE</version>
        </dependency>
        <dependency>
          <groupId>javax.servlet</groupId>
          <artifactId>servlet-api</artifactId>
          <version>2.5</version>
        </dependency>

        <!-- Json转换由jackson完成 -->
        <dependency>
          <groupId>com.fasterxml.jackson.core</groupId>
          <artifactId>jackson-databind</artifactId>
          <version>2.9.9</version>
        </dependency>
      </dependencies>
    </project>
```

- com.itheima.controller.JsonController

```
package com.itheima.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;

/**
 * Json格式数据交互案例代码
 * @RestController: 等于@Controller+@ResponseBody
 */
@Controller
public class JsonController {


    /**
     * @RequestBody: 将请求的json数据解析成对象 (默认由jackson提供支持)
     * @ResponseBody: 将返回的对象转换成json数据 (默认由jackson提供支持)
     *
     */
    @ResponseBody
```

```
        return order;
    }
}

```

- index.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>测试页面</title>
</head>
<body>
    <h3>提交Json数据请求</h3>
    <form id="jsonForm">
        <input name="id" value="1"/>
        <input name="name" value="2"/>
        <button type="button" onclick="submitJson()">提交</button>
    </form>
    <div id="jsonRes">

    </div>
</body>
<script>
    function submitJson(){
        // 1. 创建异步请求对象
        let req = new XMLHttpRequest();
        // 2. 打开请求
        req.open("POST", "/hello.do", true);
        // 3. 设置请求的数据类型(必须)
        req.setRequestHeader("Content-Type", "application/json;charset=UTF-8");
        // 4. 异步读取响应
        req.onreadystatechange = function(){
            // 2.1 判断响应状态
            if(req.readyState==4 || req.status==200){
                let data = req.responseText;
                document.getElementById("jsonRes").innerHTML=data;
            }
        };
        // 5. 封装数据
        let es = document.getElementById("jsonForm").getElementsByTagName("input");
        let json = {};
        for (let i=0; i<=es.length; i++){
            if(es[i]!=undefined){
                let name = es[i].name;
                let value = es[i].value;
                json[name] = value;
            }
        }
        // 6. 发送数据
        let body = JSON.stringify(json);
        req.send(body);
    }
```

六、RESTful支持

6.1 RESTful介绍

6.1.1 基本介绍

> REST全称是Representational State Transfer，中文意思是表述（编者注：通常译为表征）性状态转移。 它首次出现在2000年Roy Fielding的博士论文中，Roy Fielding是HTTP规范的主要编写者之一。 他在论文中提到："我这篇文章的写作目的，就是想在符合架构原理的前提下，理解和评估以网络为基础的应用软件的架构设计，得到一个功能强、性能好、适宜通信的架构。REST指的是一组架构约束条件和原则。" 如果一个架构符合REST的约束条件和原则，我们就称它为RESTful架构。

> REST本身并没有创造新的技术、组件或服务，而隐藏在RESTful背后的理念就是使用Web的现有特征和能力， 更好地使用现有Web标准中的一些准则和约束。虽然REST本身受Web技术的影响很深， 但是理论上REST架构风格并不是绑定在HTTP上，只不过目前HTTP是唯一与REST相关的实例。 所以我们这里描述的REST也是通过HTTP实现的REST。

6.1.2 RESTful优点

- **结构清晰**
- 符合标准
- 易于理解
- 扩展方便
- 地址简洁
- **应用广泛**

6.1.3 RESTful示例

请求地址(URI)	请求方法	请求含义
/user/1	GET	获取编号1的用户
/user/1	PUT	修改编号1的用户
/user/1	DELETE	删除编号1的用户
/user	POST	添新增用户

6.2 URL风格对比

	增	删	查	改
传统	/user/add	/user/delete?id=1	/user/get?id=1	/user/update?id=1
RESTful	/user	/user/1	/user/1	/user/1

6.3 案例演示

6.3.1 特点一: 方法区分

> 请求，该过滤器为HiddenHttpMethodFilter。

HiddenHttpMethodFilter

- web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
    version="2.5">


    <!-- 1. 使用统一的Servlet处理请求 -->
    <servlet-mapping>
        <servlet-name>DispatcherServlet</servlet-name>
        <url-pattern>*.do</url-pattern>
    </servlet-mapping>

    <!-- 2. 定义前端控制器统一处理请求 -->
    <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.xml</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>


    <!-- 1. 配置需要过滤器的资源 -->
    <filter-mapping>
        <filter-name>hiddenHttpMethodFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

    <!-- 2. 使用filter实现对PUT,DELETE的支持 -->
    <filter>
        <filter-name>hiddenHttpMethodFilter</filter-name>
        <filter-
class>org.springframework.web.filter.HiddenHttpMethodFilter</filter-class>
    </filter>
</web-app>
```

- index.jsp

```
<%-- 使用HiddenHttpMethodFilter过滤器实现PUT,DELETE,POST,GET方法 --%>
<%-- form标签属性method必须设置为post --%>
<form action="/hello.do" method="post">
    <%-- 表示提交方法的方式:  --%>
    <%-- 1. 设置隐藏域name=_method 2. 设置值为真实的提交方法PUT/DELETE --%>
    <input type="hidden" name="_method" value="PUT"/>
    <input name="id" value="1"/>
    <input name="name" value="2"/>
    <input type="submit"/>
</form>
```

- com.itheima.controller.RestController

```
package com.itheima.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.ResponseBody;

/**
 *
 */
@Controller
public class RestController {


    /**
     * 提交PUT方法
     *  注意1: 需要使用HiddenHttpMethodFilter过滤请求
     *  注意2: 只支持返回json类型的数据
     */
    @RequestMapping(value = "hello", method = RequestMethod.PUT)
    @ResponseBody
    public String hello(Integer id, String name){
        System.out.println(id);
        System.out.println(name);
        return "success";
    }

    /**
     * 提交DELETE方法
     *  注意1: 需要使用HiddenHttpMethodFilter过滤请求
     *  注意2: 只支持返回json类型的数据
     */
    @RequestMapping(value = "hello", method = RequestMethod.DELETE)
    @ResponseBody
    public String hello2(Integer id, String name){
        System.out.println(id);
        System.out.println(name);
        return "success";
    }
}
```

6.3.2 @PathVariable

- com.itheima.controller.RestController

```
/**
     * @PathVariable: 绑定URL地址中的参数
     *  value: 参数名称
     *  required: 默认true,必传;
     *
     *  请求示例: http://localhost:8080/hello/1/Jason
     */
@RequestMapping(value = "hello/{id}/{name}")
@ResponseBody
public String hello3(@PathVariable Integer id, @PathVariable("name") String
name){
    System.out.println(id);
    System.out.println(name);
    return "success";
}
```