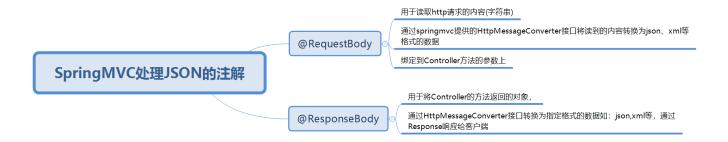
- \* 学习目标
- \*能够掌握@RequestBody和@ResponseBody
  - \* 转换器: Gson
  - \* <mvc:anotation-driven>
    - \* RequestAnnotationHandlerMapping
    - \* RequestAnnotationHandlerAdapter
- \*能够掌握SpringMVC请求参数的绑定
  - \*基本数据类型和String
  - \* POJO
  - \*集合
  - \* JSON
  - \* 自定义类型转换器
  - \* PostMan
- \*能够MockMVC对Web层进行单元测试
  - \* 模拟请求,调用控制层的方法
- \*能够掌握SpringMVC常用的注解
  - \* @RequestBody , @ResponseBody , @RestController
  - $^{\ast}$  @RequestParam , @RequestHeader , @RequestAttribute
  - \* @SessionAttribute , @SessionAttributes , @CookieValue
  - \* @GetMapping , @PostMapping , @RequestMapping

-----

----

- \* 回顾
- \*能够掌握@RequestBody和@ResponseBody
  - \* 复习JSON: <u>02-json</u>



# \* HttpMessageConverter可以发现很多解析器,其中有Gson解析

```
븚
→ HttpMessageConverter (org.springframework.http.converter)
➤ G = FormHttpMessageConverter (org.springframework.http.converter)
                                                                                                                Ā

    Q a AbstractHttpMessageConverter (org.springframework.http.converter)

                                                                                                                8: Hierarchy

    BufferedImageHttpMessageConverter (org.springframework.http.converter)

▼ 1 GenericHttpMessageConverter (org.springframework.http.converter)

    Jaxb2CollectionHttpMessageConverter (org.springframework.http.converter.xml)

   ✓ Q n AbstractGenericHttpMessageConverter (org.springframework.http.converter)
                                                                                                                ▼ ( a) AbstractJsonHttpMessageConverter (org.springframework.http.converter.json)
                                                                                                                Database
             GsonHttpMessageConverter (org.springframework.http.converter.json)

    IsonbHttpMessageConverter (org.springframework.http.converter.json)

    Q = AbstractJackson2HttpMessageConverter (org.springframework.http.converter.json)

                                                                                                                E Waven

    Q n MappingJackson2SmileHttpMessageConverter (org.springframework.http.converter.smile)

    Q m MappingJackson2HttpMessageConverter (org.springframework.http.converter.json)

    Q m MappingJackson2CborHttpMessageConverter (org.springframework.http.converter.cbor)

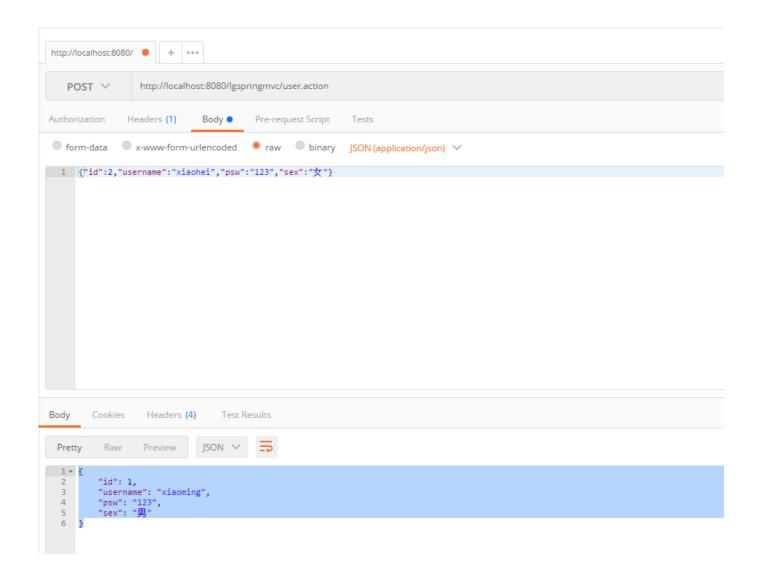
    Q m MappingJackson2XmlHttpMessageConverter (org.springframework.http.converter.xml)

    ResourceRegionHttpMessageConverter (org.springframework.http.converter)
```

```
* 案例一
 1
 2
    * 添加gson依赖
 3
     <dependency>
 4
       <groupId>com.google.code.gson
 5
       <artifactId>gson</artifactId>
       <version>2.8.5
 6
 7
     </dependency>
    * 配置
8
9
     <bean class="org.springframework.web.servlet.mvc.method.annotation.RequestMap</pre>
10
           cproperty name="messageConverters">
               t>
11
12
                   <bean class="org.springframework.http.converter.json.GsonHttpMe</pre>
               </list>
13
14
           </property>
15
     </bean>
```

```
16 * 代码
17 @Controller
18 public class HelloController3 {
      @RequestMapping(value = "/user.action", method = RequestMethod.POST)
19
      @ResponseBody
20
      public User getUser(@RequestBody User user){
21
           System.out.println(user);
22
23
          User u=new User();
          u.setId(1);
24
          u.setUsername("xiaoming");
25
          u.setPsw("123");
26
          u.setSex("男");
27
28
          return u;
29
      }
30 }
31
32 * 案例二: (<mvc:annotation-driven/>)
   * <mvc:annotation-driven>
33
    * 自动加载RequestMappingHandlerMapping,RequestMappingHandlerAdapter
34
    * 默认加载了JSON数据格式转换器。
35
   * 配置
36
     * 按照以前的规则添加命名空间
37
38
    xmlns:mvc="http://www.springframework.org/schema/mvc"
39
    http://www.springframework.org/schema/mvc
    http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd
40
    * 在springmvc.xml添加
41
      <mvc:annotation-driven/>
42
   * 继续接口测试
43
```

## \*接口测试



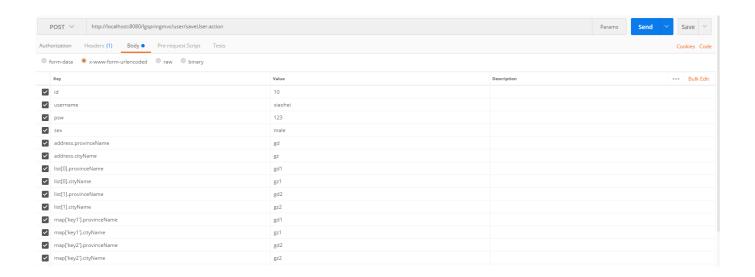
# \*能够掌握SpringMVC请求参数的绑定



```
2 @Data
 3 @AllArgsConstructor
4 @NoArgsConstructor
 5 public class Address {
       private String provinceName;
6
 7
       private String cityName;
8 }
9 @Data
10 @AllArgsConstructor
11 @NoArgsConstructor
12 public class User implements Serializable {
       private int id;
13
       private String username;
14
15
       private String psw;
       private String sex;
16
       private Address address;
17
       private List<Address> list;
18
19
       private Map<String,Address> map;
20 }
21
22 * 案例
23 @Controller
24 @RequestMapping("/user")
25 public class UserController {
       @RequestMapping(value = "/findUser", method = RequestMethod.POST)
26
       public void findUser(int id,String username){
27
           System.out.println("用户: "+id+","+username);
28
29
       }
       @RequestMapping(value = "/saveUser", method = RequestMethod.POST)
30
       public void saveUser(User user){
31
           System.out.println(user);
32
33
       }
34 }
35 测试一: (基本数据类型和String)
36 * 在postman访问: http://localhost:8080/lgspringmvc/user/findUser.action
     * 表单提交方式,请求方式为post
37
     * 参数key=id, value=10, key=username, value=xiaohei
38
39 * 在控制台观察效果
40 测试二: (pojo和集合类型)
41 * 在postman访问: http://localhost:8080/lgspringmvc/user/saveUser.action
```

```
42
    * 表单提交方式,请求方式为post
    * 参数pojo和集合类型
43
44 * 在控制台观察效果
45
  测试三:传递中文,出现中文乱码
47 * 在web.xml 添加编码过滤去
    <filter>
48
49
      <filter-name>CharacterEncodingFilter</filter-name>
50
      <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filte</pre>
      <init-param>
51
52
        <param-name>encoding</param-name>
        <param-value>UTF-8</param-value>
53
      </init-param>
54
    </filter>
55
56
    <filter-mapping>
      <filter-name>CharacterEncodingFilter</filter-name>
57
58
      <url-pattern>/*</url-pattern>
    </filter-mapping>
59
60
```





## \* 自定义类型转换器

## 实现Converter接口

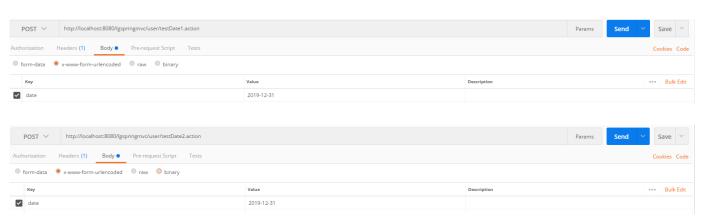
# 自定义类型转换器

在T convert(S source)完成转换功能

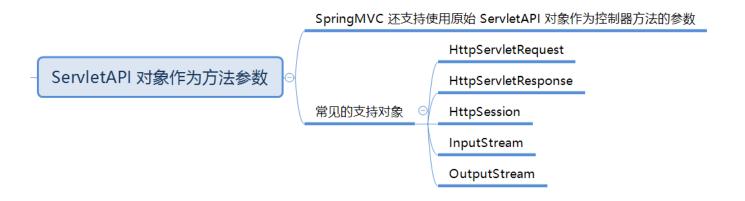
在xml配置

```
* 案例
 1
    @RequestMapping(value = "testDate1", method = RequestMethod.POST)
 2
     public void testDate(String date){
 3
          System.out.println(date);
 4
 5
     }
    @RequestMapping(value = "testDate2", method = RequestMethod.POST)
6
7
     public void testDate(Date date){
          System.out.println(date);
8
9
     }
10
11 * 测试
     * 访问http://localhost:8080/lgspringmvc/user/testDate1.action
12
      * 提交date=2019-12-31,可以测试成功
13
     * 访问http://localhost:8080/lgspringmvc/user/testDate2.action
14
      * 提交date=2019-12-31, 不可以测试成功
15
      * 字符串转换成日期类型需要自定义类型转换器
16
17
18 * 案例二: 字符串转换成日期类型
  public class StringToDateConverter implements Converter<String, Date> {
19
      @Override
20
      public Date convert(String source) {
21
22
          SimpleDateFormat sdf=new SimpleDateFormat("yyyy-MM-dd");
23
          try {
              Date date = sdf.parse(source);
24
              return date;
25
          } catch (ParseException e) {
26
              e.printStackTrace();
27
          }
28
```

```
29
           return null;
       }
30
31 }
    * 配置
32
    <bean id="conversionService1" class="org.springframework.context.support.Conve</pre>
33
34
           cproperty name="converters">
35
               <array>
                   <bean class="com.lg.converter.StringToDateConverter"></bean>
36
37
               </array>
           </property>
38
39
     </bean>
     <mvc:annotation-driven conversion-service="conversionService1"/>
40
41 * 访问http://localhost:8080/lgspringmvc/user/testDate2.action
     * 提交date=2019-12-31,可以测试成功
42
43
```



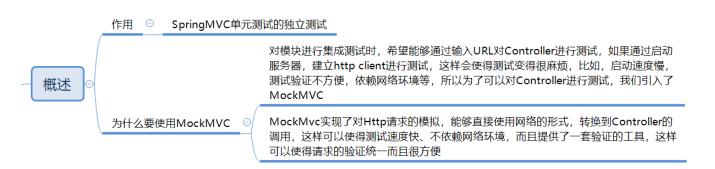
\* ServletAPI对象作为方法参数



```
* 案例
1
  @RequestMapping(value = "testsa", method = RequestMethod.POST)
       public void testServletApi(HttpServletRequest request,
 3
 4
                                  HttpServletResponse response,
 5
                                  HttpSession session){
           System.out.println(request);
 6
 7
           System.out.println(response);
           System.out.println(session);
8
9
    }
    * 访问进行http://localhost:8080/lgspringmvc/user/testsa.action测试
10
```



# \*能够MockMVC对Web层进行单元测试



```
1 * 添加依赖
   <dependency>
 3
     <groupId>junit
     <artifactId>junit</artifactId>
 4
     <version>4.12</version>
 5
 6
     <scope>test</scope>
   </dependency>
8
   <dependency>
9
     <groupId>org.springframework</groupId>
     <artifactId>spring-test</artifactId>
10
     <version>5.2.2.RELEASE
11
```

```
12
      <scope>test</scope>
   </dependency>
13
14
15 * 代码
    * 温馨提醒用静态导入,编写代码方便
16
17 import static org.springframework.test.web.servlet.request.MockMvcRequestBuilde
18 import static org.springframework.test.web.servlet.result.MockMvcResultHandlers
  import static org.springframework.test.web.servlet.result.MockMvcResultMatchers
19
20
21 @RunWith(SpringJUnit4ClassRunner.class)
22 @ContextConfiguration("classpath:springmvc.xml")
23 @WebAppConfiguration
24 public class AppTest
25 {
26
       @Autowired
27
       private WebApplicationContext applicationContext;
28
29
       private MockMvc mockMvc;
30
31
       @Before
32
33
       public void before(){
           this.mockMvc= MockMvcBuilders.webAppContextSetup(applicationContext).bu
34
35
       }
       @Test
36
       public void test1() throws Exception {
37
           MockHttpServletRequestBuilder builder = MockMvcRequestBuilders.post("/L
38
           mockMvc.perform(builder);
39
       }
40
41
       @Test
42
       public void test2() throws Exception {
43
           MockHttpServletRequestBuilder builder = post("/user/testDate1").param('
44
           mockMvc.perform(builder);
45
       }
46
47
48
       @Test
49
       public void test3() throws Exception {
           MockHttpServletRequestBuilder builder = post("/user/testDate2").param('
50
           mockMvc.perform(builder);
51
```

```
52
       }
53
54
       @Test
       public void test4() throws Exception {
55
           MockHttpServletRequestBuilder builder = post("/user/findUser").
56
                   param("id","20").
57
                   param("username", "xiaohei");
58
           mockMvc.perform(builder);
59
       }
60
61
       @Test
62
       public void test5() throws Exception {
63
           mockMvc.perform(post("/user/saveUser")
64
                   .param("id","2")
65
                   .param("username", "xiaohei")
66
                   .param("psw","123")
67
                   .param("sex","男")
68
                   .param("address.provinceName","广东")
69
                   .param("address.cityName","广州")
70
                   .param("list[0].provinceName","广东")
71
                   .param("list[0].cityName","深圳")
72
                   .param("list[1].provinceName","广东")
73
74
                   .param("list[1].cityName","佛山")
                   .param("map['key1'].provinceName","广东")
75
                   .param("map['key1'].cityName","佛山")
76
                   .param("map['key2'].provinceName","广东")
77
                   .param("map['key2'].cityName","佛山")
78
               );
79
80
       }
81
       @Test
82
       public void test6() throws Exception {
83
           //{"id":2,"username":"xiaohei","psw":"123","sex":"女"}
84
           String json="{\"id\":2,\"username\":\"xiaohei\",\"psw\":\"123\",\"sex\'
85
             ResultActions res = mockMvc.perform(post("/user.action")
86 //
                     .characterEncoding("UTF-8")
87 //
                     .contentType(MediaType.APPLICATION JSON)
88 //
89 //
                     .content(json)
90 //
             );
             ResultActions res1 = res.andExpect(status().isOk()).andDo(print());
91 //
```

```
92 //
              String result = res1.andReturn().getResponse().getContentAsString();
 93 //
              System.out.println(result);
            String result = mockMvc.perform(post("/user.action").
 94
                    characterEncoding("UTF-8").
 95
                    contentType(MediaType.APPLICATION_JSON).
 96
                    accept(MediaType.APPLICATION_JSON).
 97
                    content(json)
 98
 99
            ).andExpect(status().isOk()).andDo(print()).andReturn().getResponse().g
            System.out.println(result);
100
        }
101
102 }
103
```

\*能够掌握SpringMVC常用的注解

SpringMVC常见的注解

RequestBody

ResponseBody

RestController

RequestParam

RequestHeader

CookieValue

RequestAttribute

SessionAttribute

SessionAttributes

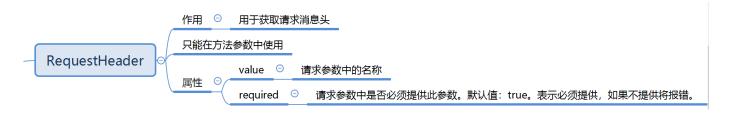
ModelAttribute

**PathVariable** 

<sup>\*</sup> RequestParam

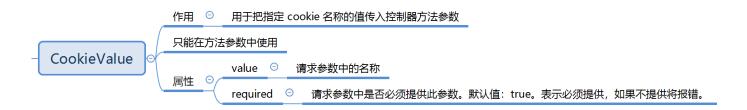
```
1 * 案例
2 * 接口方法
3 @Controller
4 @RequestMapping("/test")
5 public class TestController {
     @RequestMapping("/test1")
6
7
     public void test1(@RequestParam(value = "name") String username,
                     @RequestParam(value ="mage" ,required = false) Integer ag
8
         System.out.println(username+":"+age);
9
     }
10
11 }
12 * 单元测试
13 @Test
   public void test8() throws Exception {
14
      mockMvc.perform(post("/test/test1.action").
15
            param("name","xiaohei").param("mage","20"));
16
17
   }
   * 测试
18
    * 先不用RequestParam注解进行测试,客户端传递名字参数和控制器方法参数一致
19
     * 结果: 可以获取客户端提交的数据
20
    * 先不用RequestParam注解进行测试和少传age参数进行测试
21
     * 结果: 可以获取客户端提交的数据,只是不传,获得参数为null
22
    * 使用RequestParam的属性required进行测试
23
     * 假如为true,没有传递age参数,会报错
24
      * 假如为false,没有传递age参数,可以获取客户端提交的数据,只是不传,获得参数为nul
25
    * 先不用RequestParam注解进行测试,
26
     客户端传递参数名字和控制器方法参数不一致
27
     * 结果: 没有获取客户端提交的数据
28
    * 使用RequestParam的属性value,名字和客户端传递参数名字曲一致
29
     * 结果: 可以获取客户端提交的数据
30
31
```

## \* RequestHeader



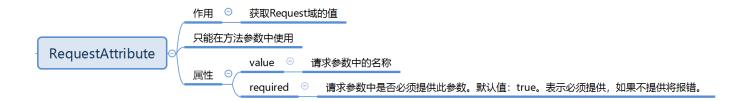
```
1 * 案例
2 @RequestMapping("/test2")
3 public void test2(
4 @RequestHeader(value = "Accept-Language",required = false) String requestHead
5 System.out.println(requestHeader);
6 }
7 @Test
8 public void test9() throws Exception {
9 mockMvc.perform(post("/test/test2.action").header("Accept-Language","zh"));
10 }
```

#### \* CookieValue



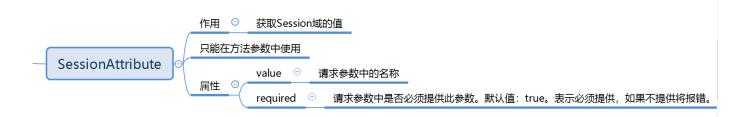
```
1 * 案例
2 @RequestMapping("/test3")
3 public void test3(@CookieValue(value = "JSESSIONID",required = false) String co
4    System.out.println(cookieValue);
5 }
6 @Test
7 public void test10() throws Exception {
8    mockMvc.perform(post("/test/test3.action").cookie(new Cookie("JSESSIONID","a
9 }
```

## \* RequestAttribute



```
1 * 案例
2 @RequestMapping("/test4")
3 public void test4(@RequestAttribute(value = "username", required = false) String
4    System.out.println(username);
5 }
6 @Test
7 public void test11() throws Exception {
    mockMvc.perform(post("/test/test4.action").requestAttr("username", "xiaoming)
9 }
```

#### \* SessionAttribute



```
1 * 案例
2 @RequestMapping("/test5")
3 public void test5(@SessionAttribute(value = "username",required = false) String
4    System.out.println(username);
5 }
6 @Test
7 public void test12() throws Exception {
    mockMvc.perform(post("/test/test5.action").sessionAttr("username","xiaoming")
9 }
```

#### \* SessionAttributes

## 作用 🖯 用于多次执行控制器方法间的参数共享

### 在类中使用

 $\Theta$ 

SessionAttributes

value 🖯 用于指定存入的属性名称

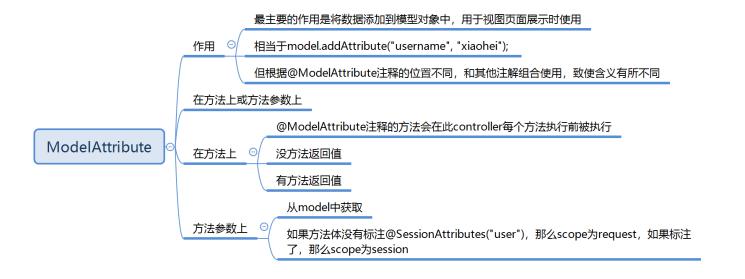
属性

types 🖯 用于指定存入的数据类型

```
1 * 案例
 2 * 代码
 3 @Controller
 4 @RequestMapping("/test")
 5 @SessionAttributes(value = {"username", "psw"}, types = Integer.class)
 6 public class TestController {
 7
       @RequestMapping("/test6")
       public String test6(Model model){
 8
           model.addAttribute("username", "xiaohei");
 9
10
           model.addAttribute("psw","123");
           model.addAttribute("age",88);
11
           return "success";
12
13
       }
       @RequestMapping("/test7")
14
       public String test7(Model model, HttpSession session){
15
           System.out.println(model.getAttribute("username")+":"
16
                   +model.getAttribute("psw")+":"
17
18
                   +model.getAttribute("age"));
           System.out.println(session.getAttribute("username")+":"
19
                   +session.getAttribute("psw")+":"
20
                   +session.getAttribute("age"));
21
22
           return "success";
       }
23
       @RequestMapping("/test8")
24
       public String test8(SessionStatus sessionStatus){
25
26
           sessionStatus.setComplete();
           return "success";
27
       }
28
29 }
30
   * 在webapp创建test.jsp
31
32 | 32 | | contentType="text/html;charset=UTF-8" language="java" isELIgnored="fal
```

```
<html>
33
  <head>
34
      <title>测试</title>
35
36 </head>
  <body>
37
38
      39
      <a href="${pageContext.request.contextPath}/test/test7.action">取出 Session/
40
      <hr/>
41
      <a href="${pageContext.request.contextPath}/test/test8.action">清除 Session/
42
      <hr/>
43
44 </body>
45 </html>
46
47 * 在WEB-INF/jsps/下新建success.jsp
48 < m page contentType="text/html;charset=UTF-8" language="java" isELIgnored="fal
49 
49 
taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
50 <html>
51 <body>
52 <h2>Hello World:${username}</h2>
53 </body>
54 </html>
55
56 * 访问:http://localhost:8080/lgspringmvc/test.jsp 进行测试
```

### \* ModelAttribute



```
1 * 案例一: 声明在方法前,没有返回值
 2 @ModelAttribute
 3 public void populateModel(String username, Model model){
      model.addAttribute("username", username);
4
 5 }
6 @RequestMapping("/test9")
7 public String test9(){
      return "success";
8
9 }
10 * 测试: http://localhost:8080/lgspringmvc/test/test9.action?username=xiaohei
11 * 案例二:声明在方法前,有返回值
   @ModelAttribute
12
   public User showUser(){
13
          User user1=new User();
14
          user1.setUsername("xiaobai");
15
          user1.setPsw("123");
16
          return user1;
17
18
    * 相当于: model.addAttribute("user",user1);
19
20 * 在success.jsp添加
    <h2>Hello World:${user.username}</h2>
21
22
23 * 案例三: 声明在参数上
   @RequestMapping("/test9")
24
25 public String test9(@ModelAttribute("user") User u){
      System.out.println(u);
26
      return "success";
27
28
   }
29
   * 测试结果: 不谢@ModelAttribute也可以获取值
```