* 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

    * @RequestParam，@RequestHeader，@RequestAttribute

    * @SessionAttribute，@SessionAttributes，@CookieValue

    * @GetMapping，@PostMapping，@RequestMapping

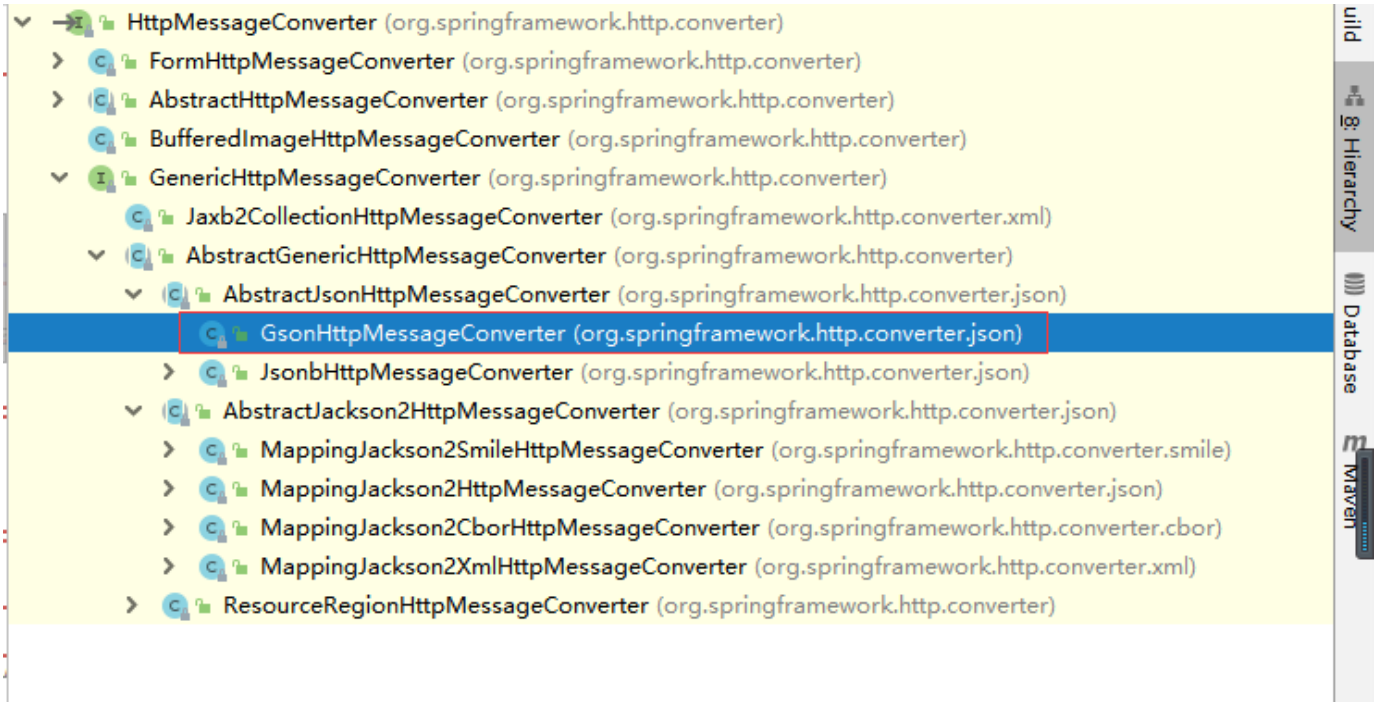
---

 * 回顾

 * 能够掌握@RequestBody和@ResponseBody

    * 复习JSON：02-json

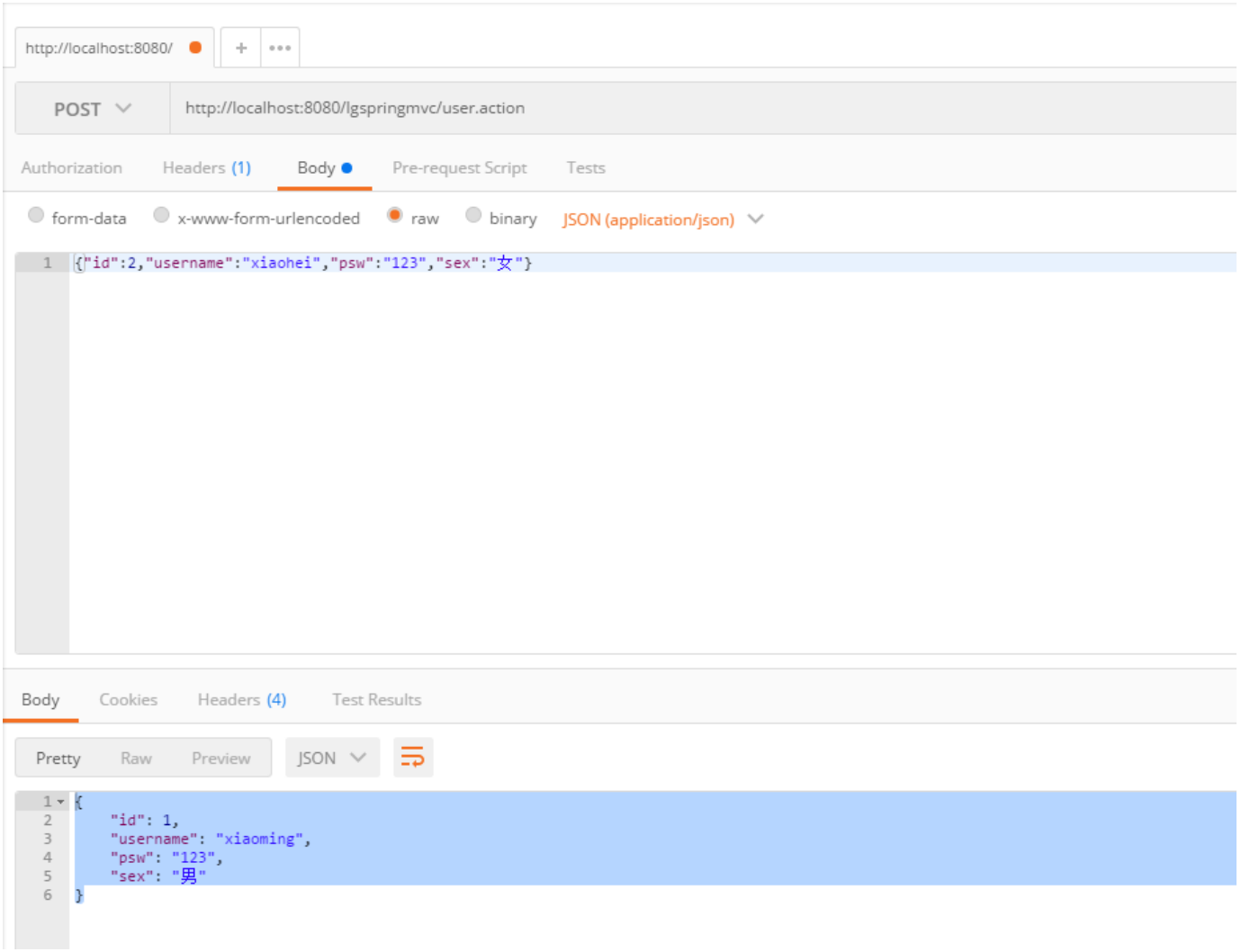
* HttpMessageConverter可以发现很多解析器，其中有Gson解析




```
* 案例一
 * 添加gson依赖
  <dependency>
    <groupId>com.google.code.gson</groupId>
    <artifactId>gson</artifactId>
    <version>2.8.5</version>
  </dependency>
 * 配置
  <bean class="org.springframework.web.servlet.mvc.method.annotation.RequestMap
        <property name="messageConverters">
            <list>
                <bean class="org.springframework.http.converter.json.GsonHttpMe
            </list>
        </property>
  </bean>
```

```
 * 代码
@Controller
public class HelloController3 {
    @RequestMapping(value = "/user.action",method = RequestMethod.POST)
    @ResponseBody
    public User getUser(@RequestBody User user){
        System.out.println(user);
        User u=new User();
        u.setId(1);
        u.setUsername("xiaoming");
        u.setPsw("123");
        u.setSex("男");
        return u;
    }
}

* 案例二:（<mvc:annotation-driven/>）
 * <mvc:annotation-driven>
  * 自动加载RequestMappingHandlerMapping，RequestMappingHandlerAdapter
  * 默认加载了JSON数据格式转换器。
 * 配置
  * 按照以前的规则添加命名空间
  xmlns:mvc="http://www.springframework.org/schema/mvc"
  http://www.springframework.org/schema/mvc
  http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd
 * 在springmvc.xml添加
   <mvc:annotation-driven/>
 * 继续接口测试
```

* 接口测试

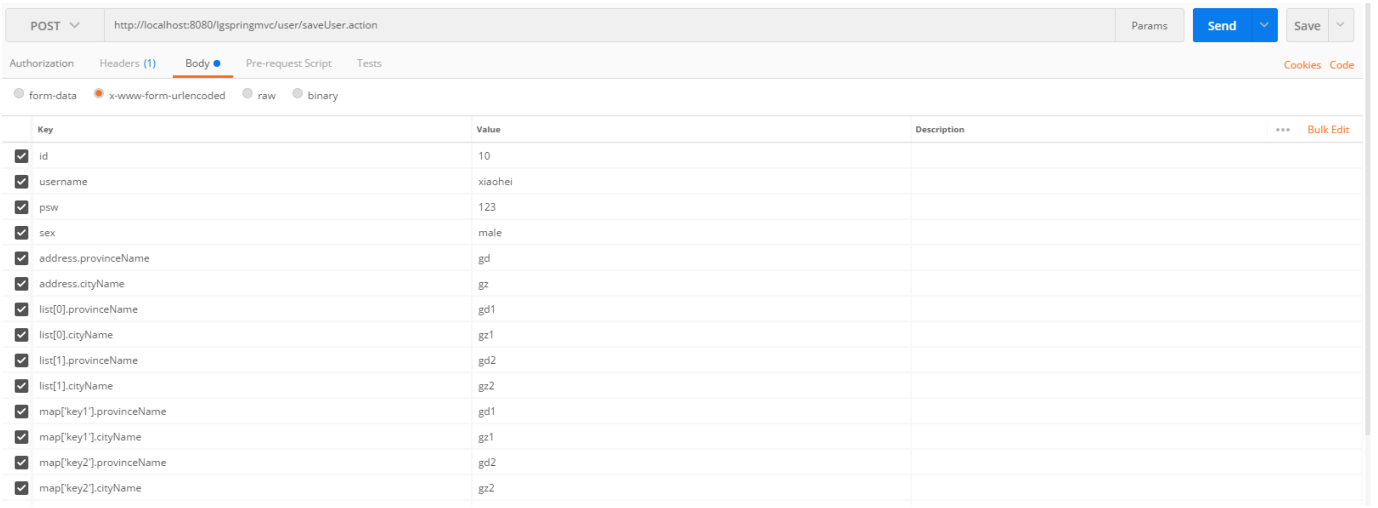


* 能够掌握SpringMVC请求参数的绑定

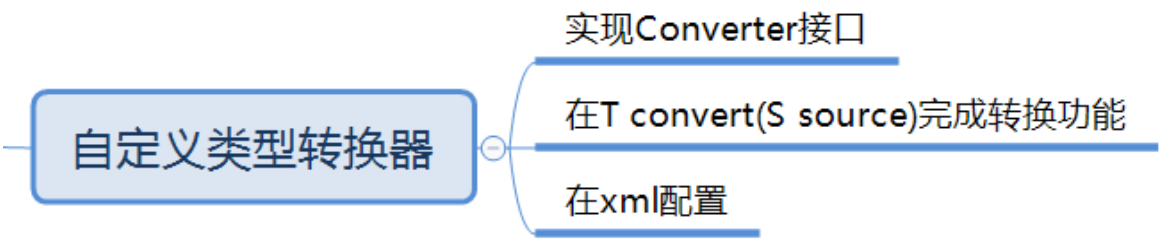

```
1 * 前期准备
```

```
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Address {
    private String provinceName;
    private String cityName;
}
@Data
@AllArgsConstructor
@NoArgsConstructor
public class User implements Serializable {
    private int id;
    private String username;
    private String psw;
    private String sex;
    private Address address;
    private List<Address> list;
    private Map<String,Address> map;
}

* 案例
@Controller
@RequestMapping("/user")
public class UserController {
    @RequestMapping(value = "/findUser",method = RequestMethod.POST)
    public void findUser(int id,String username){
        System.out.println("用户: "+id+","+username);
    }
    @RequestMapping(value = "/saveUser",method = RequestMethod.POST)
    public void saveUser(User user){
        System.out.println(user);
    }
}
测试一：（基本数据类型和String）
* 在postman访问：http://localhost:8080/lgspringmvc/user/findUser.action
  * 表单提交方式，请求方式为post
  * 参数key=id，value=10，key=username，value=xiaohei
* 在控制台观察效果
测试二：（pojo和集合类型）
* 在postman访问：http://localhost:8080/lgspringmvc/user/saveUser.action
```

```
   * 表单提交方式，请求方式为post
   * 参数pojo和集合类型
* 在控制台观察效果

测试三：传递中文，出现中文乱码
* 在web.xml 添加编码过滤去
 <filter>
    <filter-name>CharacterEncodingFilter</filter-name>
    <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filte
    <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>

```



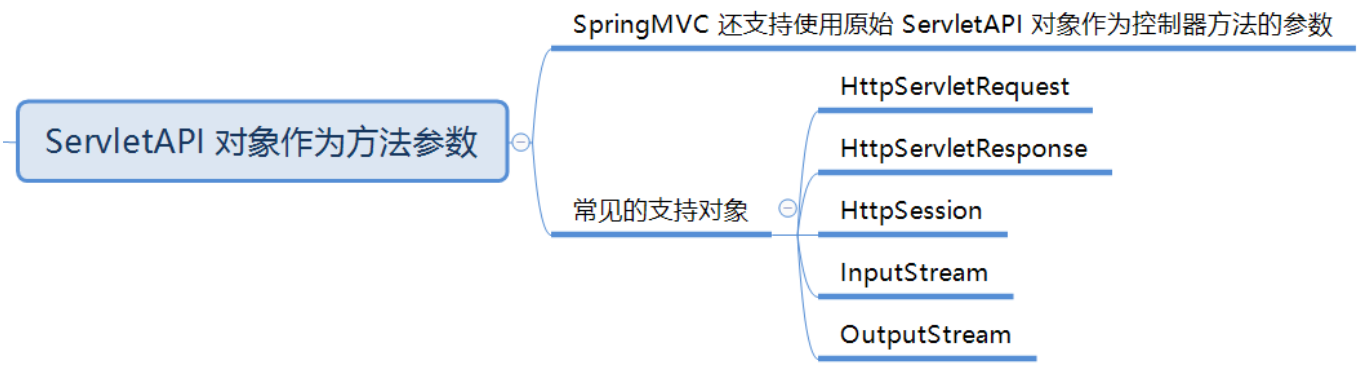
* 自定义类型转换器




```
* 案例
  @RequestMapping(value = "testDate1",method = RequestMethod.POST)
  public void testDate(String date){
        System.out.println(date);
  }
  @RequestMapping(value = "testDate2",method = RequestMethod.POST)
  public void testDate(Date date){
        System.out.println(date);
   }

* 测试
  * 访问http://localhost:8080/lgspringmvc/user/testDate1.action
    * 提交date=2019-12-31，可以测试成功
  * 访问http://localhost:8080/lgspringmvc/user/testDate2.action
    * 提交date=2019-12-31，不可以测试成功
    * 字符串转换成日期类型需要自定义类型转换器

* 案例二：字符串转换成日期类型
public class StringToDateConverter implements Converter<String, Date> {
    @Override
    public Date convert(String source) {
        SimpleDateFormat sdf=new SimpleDateFormat("yyyy-MM-dd");
        try {
            Date date = sdf.parse(source);
            return  date;
        } catch (ParseException e) {
            e.printStackTrace();
        }
```

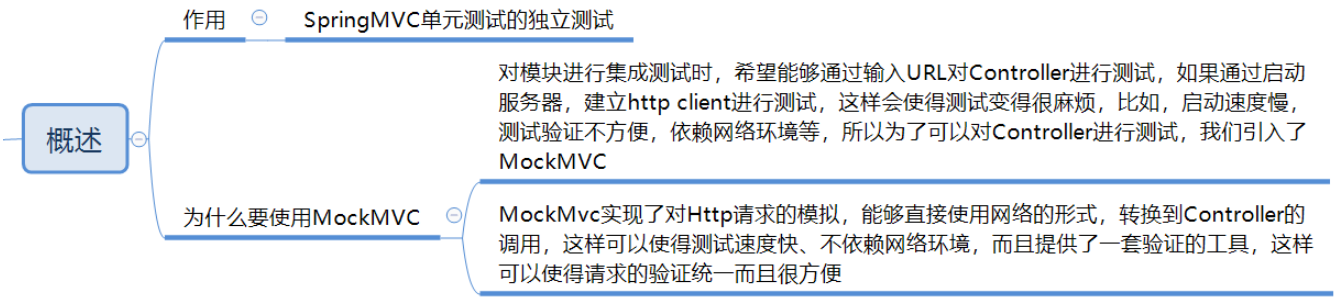
```
        return null;
    }
}
 * 配置
 <bean id="conversionService1" class="org.springframework.context.support.Conve
        <property name="converters">
            <array>
                <bean class="com.lg.converter.StringToDateConverter"></bean>
            </array>
        </property>
  </bean>
  <mvc:annotation-driven conversion-service="conversionService1"/>
* 访问http://localhost:8080/lgspringmvc/user/testDate2.action
  * 提交date=2019-12-31，可以测试成功

```

* ServletAPI对象作为方法参数



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

* 能够MockMVC对 Web 层进行单元测试



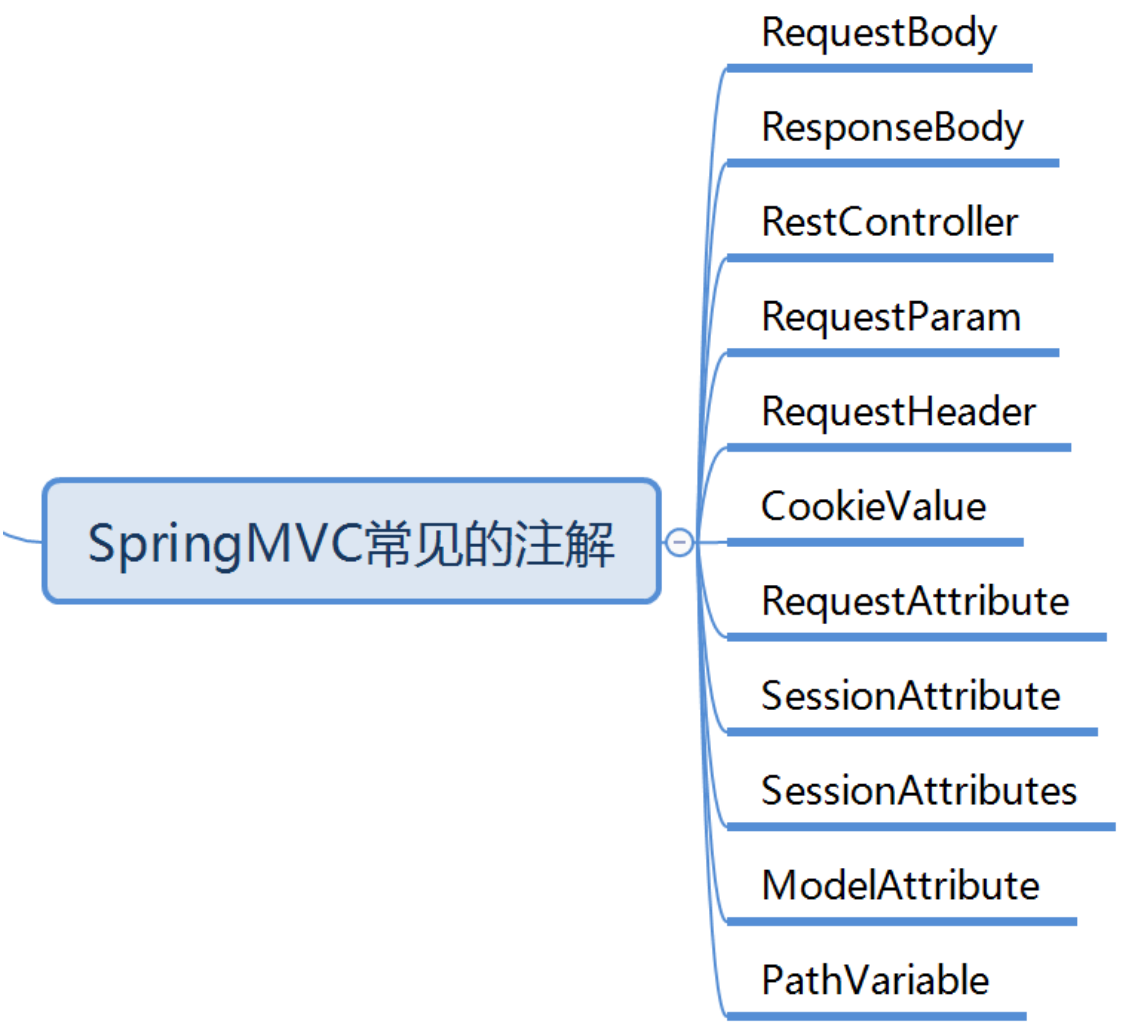

```
* 添加依赖
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-test</artifactId>
    <version>5.2.2.RELEASE</version>
```

```
   <scope>test</scope>
</dependency>

* 代码
 * 温馨提醒用静态导入，编写代码方便
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilde
import static org.springframework.test.web.servlet.result.MockMvcResultHandlers
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("classpath:springmvc.xml")
@WebAppConfiguration
public class AppTest
{

    @Autowired
    private WebApplicationContext applicationContext;

    private MockMvc mockMvc;

    @Before
    public void before(){
        this.mockMvc= MockMvcBuilders.webAppContextSetup(applicationContext).bu
    }
    @Test
    public void test1() throws Exception {
        MockHttpServletRequestBuilder builder = MockMvcRequestBuilders.post("/u
        mockMvc.perform(builder);
    }

    @Test
    public void test2() throws Exception {
        MockHttpServletRequestBuilder builder = post("/user/testDate1").param('
        mockMvc.perform(builder);
    }

    @Test
    public void test3() throws Exception {
        MockHttpServletRequestBuilder builder = post("/user/testDate2").param('
        mockMvc.perform(builder);
```

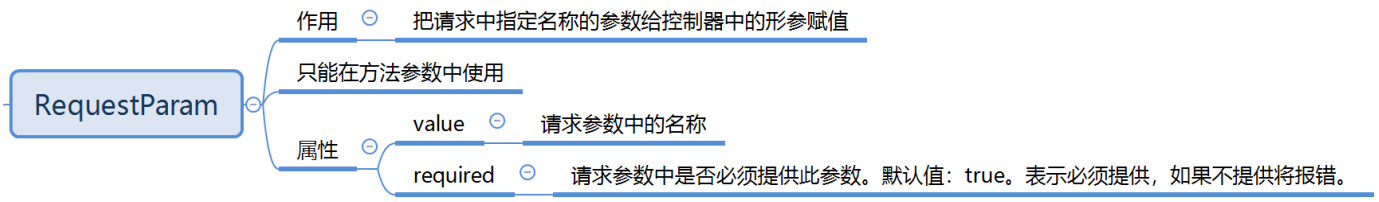
```
52      }
53
54      @Test
55      public void test4() throws Exception {
56          MockHttpServletRequestBuilder builder = post("/user/findUser").
57                  param("id","20").
58                  param("username","xiaohei");
59          mockMvc.perform(builder);
60      }
61
62      @Test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

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

* 能够掌握SpringMVC常用的注解

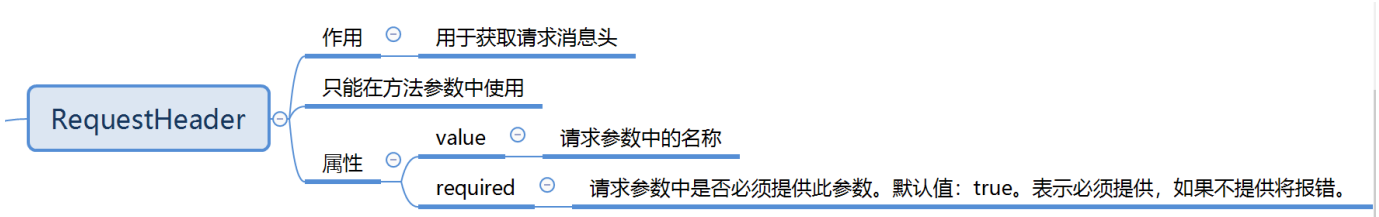


* RequestParam



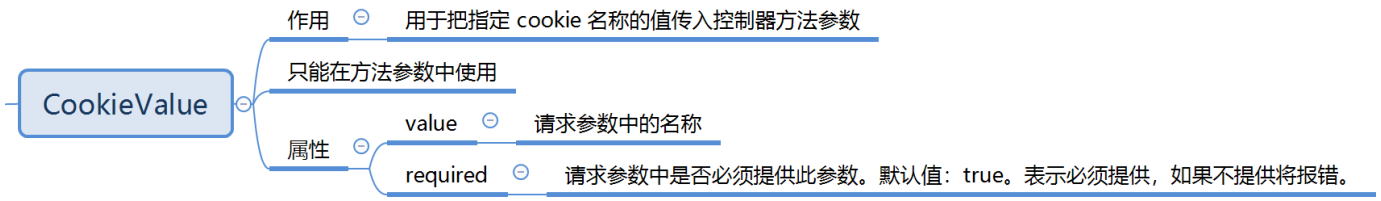

```
* 案例
* 接口方法
@Controller
@RequestMapping("/test")
public class TestController {
    @RequestMapping("/test1")
    public void test1(@RequestParam(value = "name") String username,
                      @RequestParam(value ="mage" ,required = false) Integer ag
        System.out.println(username+":"+age);
    }
}
* 单元测试
@Test
 public void test8() throws Exception {
     mockMvc.perform(post("/test/test1.action").
             param("name","xiaohei").param("mage","20"));
 }
 * 测试
  * 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
    * 结果：可以获取客户端提交的数据
  * 先不用RequestParam注解进行测试和少传age参数进行测试
    * 结果：可以获取客户端提交的数据，只是不传，获得参数为null
  * 使用RequestParam的属性required进行测试
    * 假如为true，没有传递age参数，会报错
    * 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
  * 先不用RequestParam注解进行测试，
    客户端传递参数名字和控制器方法参数不一致
    * 结果：没有获取客户端提交的数据
  * 使用RequestParam的属性value，名字和客户端传递参数名字曲一致
    * 结果：可以获取客户端提交的数据

```

* RequestHeader



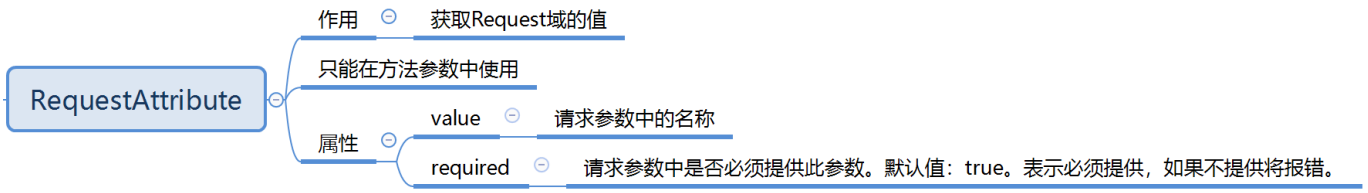

```
* 案例
@RequestMapping("/test2")
public void test2(
  @RequestHeader(value = "Accept-Language",required = false) String requestHead
        System.out.println(requestHeader);
 }
@Test
public void test9() throws Exception {
  mockMvc.perform(post("/test/test2.action").header("Accept-Language","zh"));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

* CookieValue



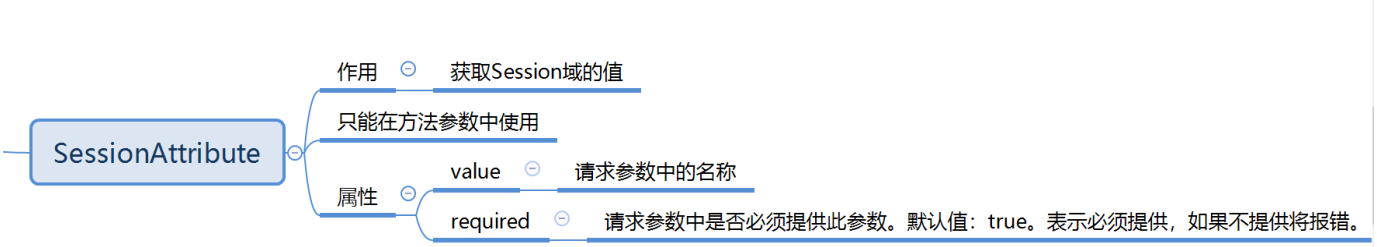

```
* 案例
@RequestMapping("/test3")
public void test3(@CookieValue(value = "JSESSIONID",required = false) String co
    System.out.println(cookieValue);
}
@Test
public void test10() throws Exception {
   mockMvc.perform(post("/test/test3.action").cookie(new Cookie("JSESSIONID","a
}
```

* RequestAttribute



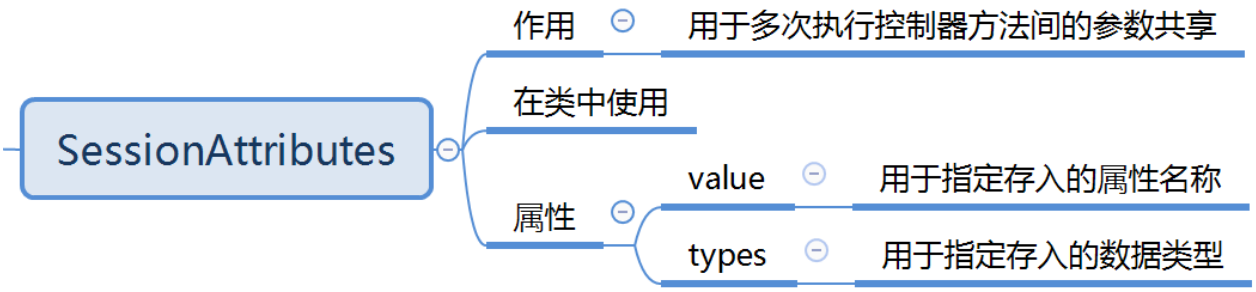

```
* 案例
@RequestMapping("/test4")
public void test4(@RequestAttribute(value = "username",required = false) String
    System.out.println(username);
}
@Test
public void test11() throws Exception {
    mockMvc.perform(post("/test/test4.action").requestAttr("username","xiaoming
}
```

* SessionAttribute




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

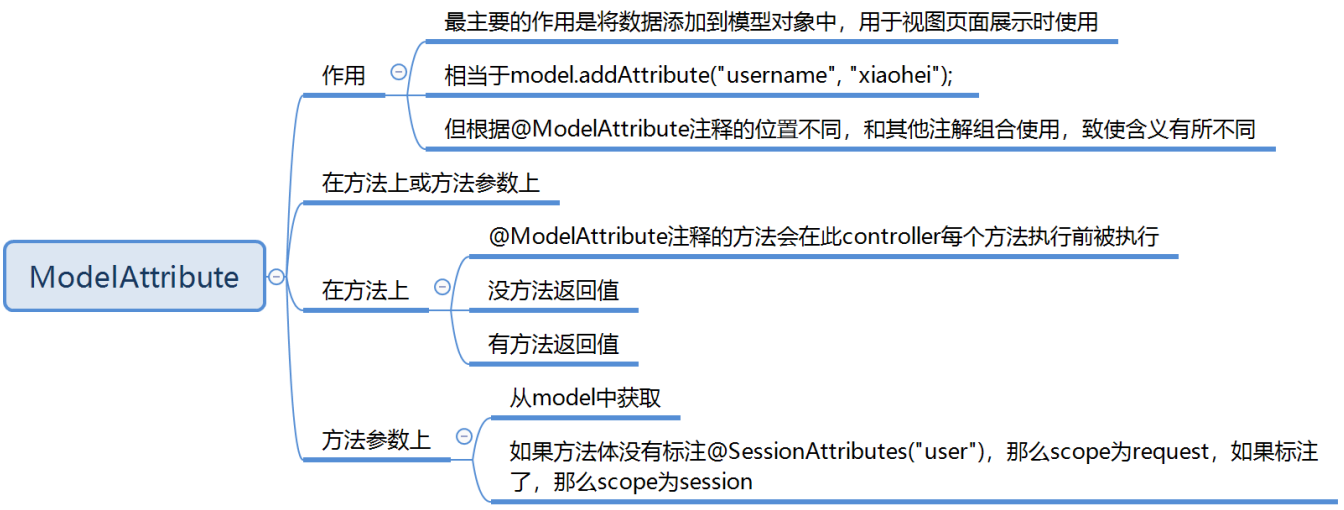
* SessionAttributes




```
* 案例
* 代码
@Controller
@RequestMapping("/test")
@SessionAttributes(value = {"username","psw"},types = Integer.class)
public class TestController {
    @RequestMapping("/test6")
    public String test6(Model model){
        model.addAttribute("username","xiaohei");
        model.addAttribute("psw","123");
        model.addAttribute("age",88);
        return "success";
    }
    @RequestMapping("/test7")
    public String test7(Model model, HttpSession session){
        System.out.println(model.getAttribute("username")+":"
                +model.getAttribute("psw")+":"
                +model.getAttribute("age"));
        System.out.println(session.getAttribute("username")+":"
                +session.getAttribute("psw")+":"
                +session.getAttribute("age"));
        return "success";
    }
    @RequestMapping("/test8")
    public String test8(SessionStatus sessionStatus){
        sessionStatus.setComplete();
        return "success";
    }
}

* 在webapp创建test.jsp
<%@ page contentType="text/html;charset=UTF-8" language="java" isELIgnored="fal
```

```
<html>
<head>
    <title>测试</title>
</head>
<body>
    <a href="${pageContext.request.contextPath}/test/test6.action">存入 Session
    <hr/>
    <a href="${pageContext.request.contextPath}/test/test7.action">取出 Session
    <hr/>
    <a href="${pageContext.request.contextPath}/test/test8.action">清除 Session
    <hr/>
</body>
</html>

* 在WEB-INF/jsps/下新建success.jsp
<%@ page contentType="text/html;charset=UTF-8" language="java" isELIgnored="fal
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
<body>
<h2>Hello World:${username}</h2>
</body>
</html>

* 访问:http://localhost:8080/lgspringmvc/test.jsp 进行测试
```

* ModelAttribute



```
* 案例一：声明在方法前，没有返回值
 @ModelAttribute
public void populateModel(String username,Model model){
    model.addAttribute("username",username);
}
@RequestMapping("/test9")
public String test9(){
    return "success";
}
* 测试：http://localhost:8080/lgspringmvc/test/test9.action?username=xiaohei
* 案例二:声明在方法前，有返回值
 @ModelAttribute
 public User showUser(){
        User user1=new User();
        user1.setUsername("xiaobai");
        user1.setPsw("123");
        return user1;
  }
  * 相当于：model.addAttribute("user",user1);
* 在success.jsp添加
 <h2>Hello World:${user.username}</h2>

* 案例三：声明在参数上
 @RequestMapping("/test9")
public String test9(@ModelAttribute("user") User u){
    System.out.println(u);
    return "success";
 }
 * 测试结果：不谢@ModelAttribute也可以获取值
```