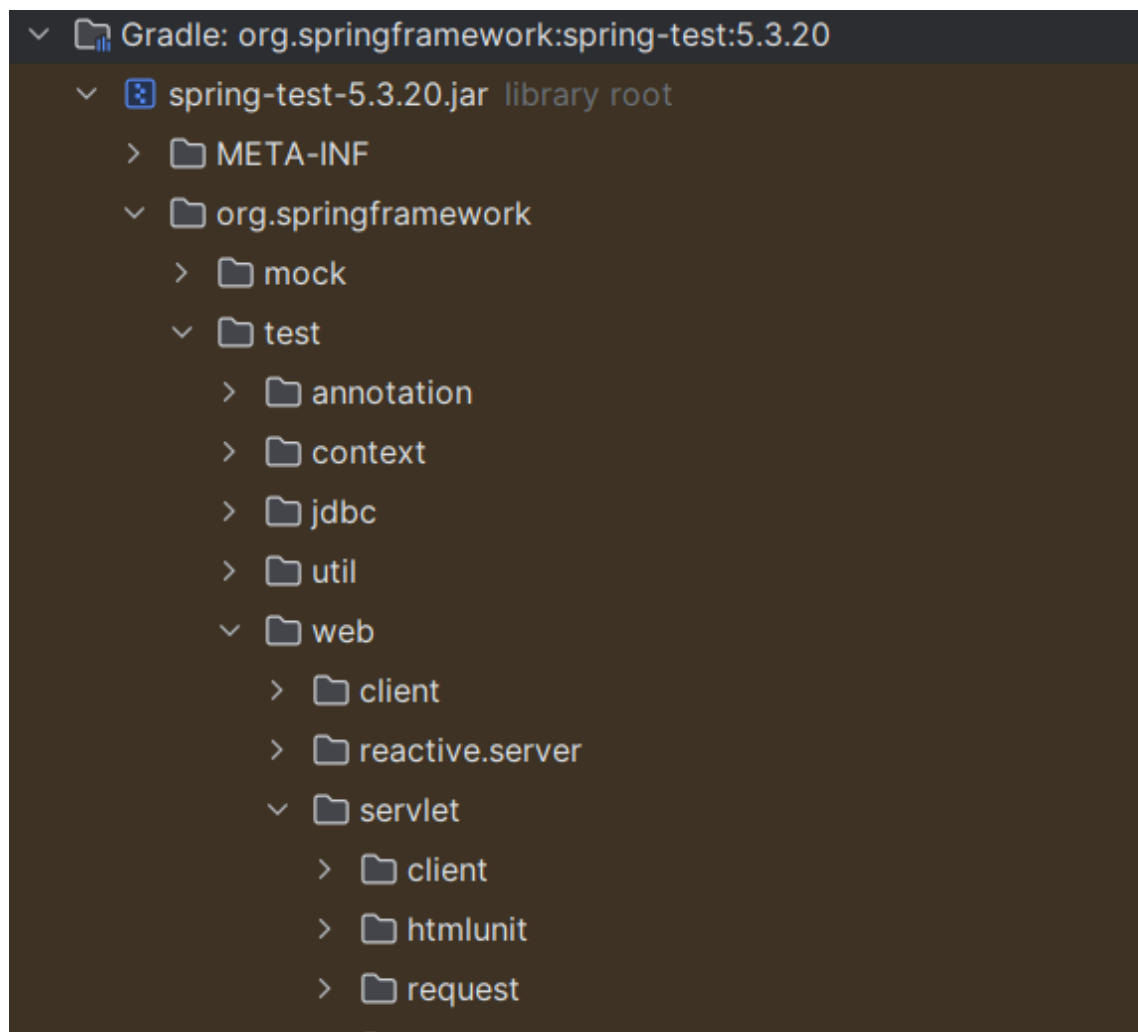


1、MockMvc进行单元测试

(0) 参考

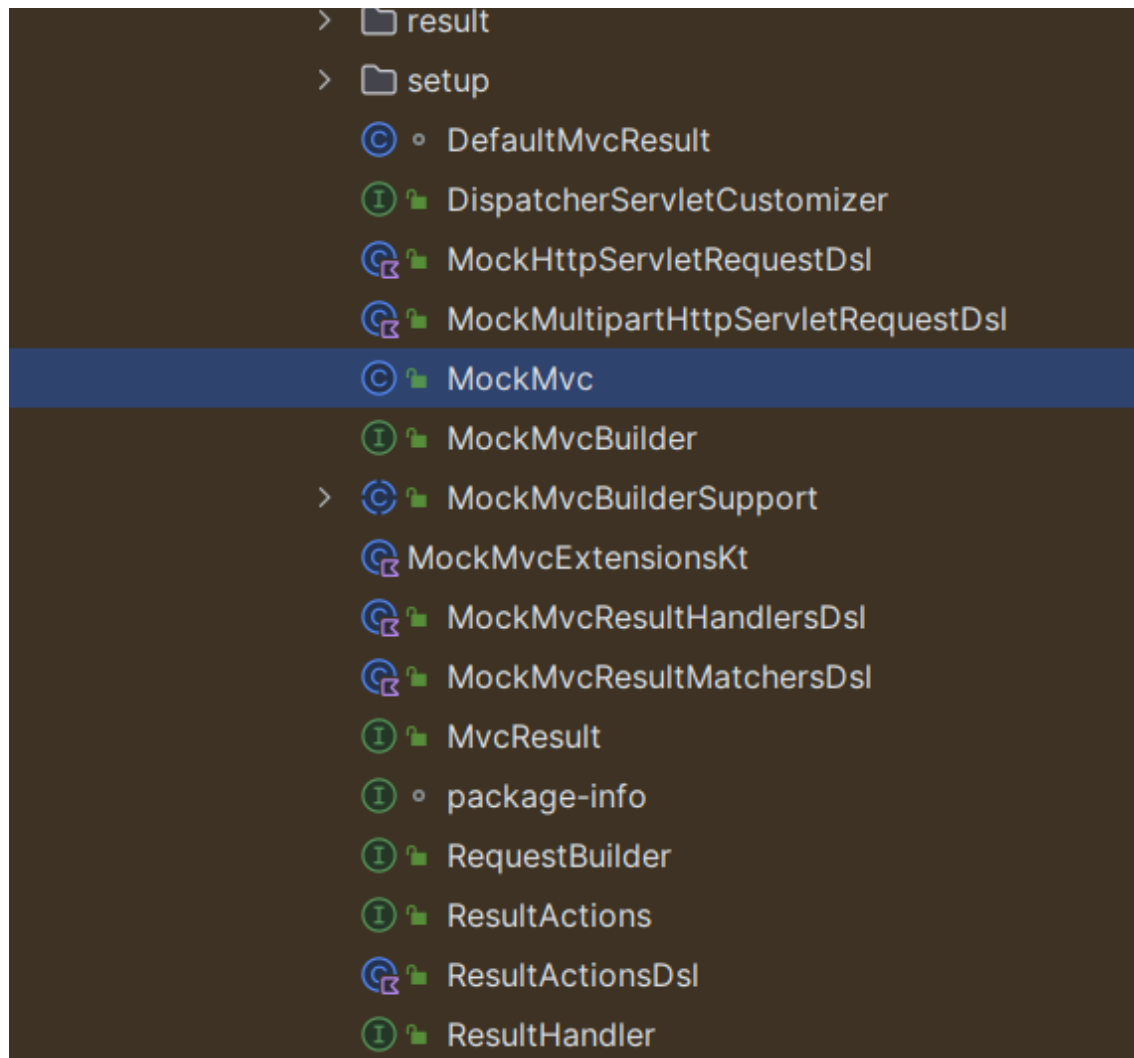
- <https://blog.csdn.net/yangshengwei230612/article/details/104811671>
- [Spring MVC测试框架详解——服务端测试 - 《亿级流量网站架构核心技术》~- ITeye博客](#)
- spring-test包提供的，针对restful风格接口（servlet）的单元测试



笔记目录



- 1、MockMvc进行单元测试
 - (0) 参考
 - (1) 简单介绍
 - (2) 使用原因
 - (3) 依赖
 - (4) 底层源码逻辑梳理
 - (4-1) MockMvcBuilde...
 - (4-2) MockMvcBuilde...
 - (4-1) 和 (4-2) 的总结
 - (4-3) MockMvcReque...
 - (4-4) MockHttpServle...
 - (4-5) 执行MockMvc对...
 - (5) 通用测试案例
 - (6) MvcResult接口



(1) 简单介绍

- MockMvc，顾名思义，mock一个mvc，mock又叫模拟，所以是对springmvc的模拟，从URL请求到springmvc的controller控制器处理，再到view视图渲染都可以模拟测试
- 一句话：基于RESTful风格（GET/POST/PUT/DELETE/OPTIONS/HEAD）的SpringMVC的模拟测试

(2) 使用原因

- 一般我们对Controller的方法进行测试，需要启动服务器（主应用程序，tomcat服务器启动，然后构建spring的上下文环境，也叫web环境，注册一堆组件到web环境），然后客户端http请求（借助HttpClient、apifox、postman等工具）服务器进行测试，这样测试比较麻烦，速度慢，测试验证不方便，依赖网络环境等，所以才引入了MockMvc，MockMvc是对http请求的模拟，无需启动服务器，不依赖网络环境，能够直接调用到Controller，测试快，还提供结果验证工具，使得测试很方便。
- 所以MockMvc是模拟出一个SpringMVC的运行环境（模拟了一个上下文环境），并没有真正地运行一个完整的servlet容器

（3）依赖

复制代码

```
1 testImplementation("org.springframework.boot:spring-boot-starter-  
test")  
2 内部是spring-test-6.0.5.jar包
```

java >

（4）底层源码逻辑梳理

（4-1）MockMvcBuilders工具类，用来构建MockMvcBuilder对象

- 用来构建一个含有WebApplicationContext上下文数据（模拟web环境）的MockMvcBuilder对象，所以还是集成了一个模拟的web环境来测试
- 也可以用来构建一个含有一组控制器（不需要上下文环境了，因为已经拿到需要的组件对象了）的MockMvcBuilder对象，后面就不用上下文了，以及从上下文中获取注册到上下文的控制器的操作了。独立测试，不需要web环境。

复制代码

```
1  
2 public final class MockMvcBuilders {  
3  
4     private MockMvcBuilders() {}
```

```

5
6     public static DefaultMockMvcBuilder
      webApplicationContextSetup(WebApplicationContext context) {
7         return new DefaultMockMvcBuilder(context);
8     }
9
10    public static StandaloneMockMvcBuilder
      standaloneSetup(Object... controllers){
11        return new StandaloneMockMvcBuilder(controllers);
12    }
13 }

```

java >

(4-2) MockMvcBuilder对象，用来构建一个MockMvc对象

```

1 public interface MockMvcBuilder {
2     MockMvc build();
3 }

```

复制代码

java >

(4-1) 和 (4-2) 的总结

```

1 this.mockMvc = MockMvcBuilders
2     //模拟创建一个含有上下文的DefaultMockMvcBuilder
3     .webApplicationContextSetup(webApplicationContext)
4     //模拟创建含有指定的控制器的StandaloneMockMvcBuilder
5     .standaloneSetup(this.controller)
6     //上述两种创建方式选一种即可

```

复制代码

```
7     .build();
```

java >

(4-3) MockMvcRequestBuilders工具类，提供一些静态方法来构建一个MockHttpServletRequestBuilder对象

- 此时MockHttpServletRequestBuilder对象就可以当成含有一个空的请求对象的容器（空表单，请求对象中只是包含了请求方式post/get/put/delete/options/head请求方式），这一步没有构建出请求对象中的一些数据（请求头，请求cookie、请求内容（content、param），内容格式类型等）

复制代码

```
1 public abstract class MockMvcRequestBuilders {
2
3     //post请求
4     public static MockHttpServletRequestBuilder post(URI uri) {
5         return new MockHttpServletRequestBuilder(HttpMethod.POST,
6 uri);
7     }
8     //get请求
9     public static MockHttpServletRequestBuilder get(URI uri) {
10        return new MockHttpServletRequestBuilder(HttpMethod.GET,
11 uri);
12    }
13    //put请求
14    public static MockHttpServletRequestBuilder put(URI uri) {
15        return new MockHttpServletRequestBuilder(HttpMethod.PUT,
16 uri);
17    }
18    //delete请求
19    public static MockHttpServletRequestBuilder delete(URI uri) {
20        return new
21 MockHttpServletRequestBuilder(HttpMethod.DELETE, uri);
```

```

18     }
19     //options请求
20     public static MockHttpServletRequestBuilder options(Uri uri)
21     {
22         return new
23         MockHttpServletRequestBuilder(HttpMethod.OPTIONS, uri);
24     }
25     //head请求
26     return new MockHttpServletRequestBuilder(httpMethod,
27     uri);
28 }

```

java >

(4-4) MockHttpServletRequestBuilder对象的相关方法，来构建出请求对象中的相关数据（请求头，请求cookie、请求内容（content、param），内容格式类型等）

- 是使用的建造者模式来构建填充请求的相关数据的，每次调用相关方法后，都是返回MockHttpServletRequestBuilder对象本身
- MockHttpServletRequestBuilder对象的buildRequest(ServletContext)方法，创建出完整的请求对象

复制代码

```

1 public class MockHttpServletRequestBuilder{
2     private MockHttpSession session;
3     private String characterEncoding;
4     private byte[] content;
5     private String contentType;

```

```
6     private final MultiValueMap<String, Object> headers =
7         new LinkedMultiValueMap<>();
8     private final MultiValueMap<String, String> parameters =
9         new LinkedMultiValueMap<>();
10    private final MultiValueMap<String, String> queryParams =
11        new LinkedMultiValueMap<>();
12    private final List<Cookie> cookies = new ArrayList<>();
13    private final List<Locale> locales = new ArrayList<>();
14    private final Map<String, Object> requestAttributes =
15        new LinkedHashMap<>();
16    private final Map<String, Object> sessionAttributes =
17        new LinkedHashMap<>();
18
19    public MockHttpServletRequestBuilder
characterEncoding(Charset encoding){
20        this.characterEncoding = encoding
21    }
22    public MockHttpServletRequestBuilder content(byte[] content)
    {}
23    public MockHttpServletRequestBuilder content(String content)
    {}
24    public MockHttpServletRequestBuilder contentType(MediaType
contentType){}
25    public MockHttpServletRequestBuilder contentType(String
contentType){}
26    public MockHttpServletRequestBuilder header(String name,
Object... values){}
27    public MockHttpServletRequestBuilder headers(HttpHeaders
httpHeaders){}
28    public MockHttpServletRequestBuilder param(String name,
String... values){}
```

```
29     public MockHttpServletRequestBuilder
        params(MultiValueMap<String, String> params){}
30     public MockHttpServletRequestBuilder queryParam(String name,
        String... values){}
31     public MockHttpServletRequestBuilder
        queryParams(MultiValueMap<String, String> params){}
32     public MockHttpServletRequestBuilder cookie(Cookie...
        cookies){}
33     public MockHttpServletRequestBuilder requestAttr(String name,
        Object value) {}
34     public MockHttpServletRequestBuilder sessionAttr(String name,
        Object value) {}
35     public MockHttpServletRequestBuilder locale(@Nullable Locale
        locale){}
36
37     @Override
38     public final MockHttpServletRequest buildRequest(ServletContext
        servletContext) {
39
40         MockHttpServletRequest request =
        createServletRequest(servletContext);
41         if (this.session != null) {
42             request.setSession(this.session);
43         }
44         request.setCharacterEncoding(this.characterEncoding);
45         request.setContent(this.content);
46         request.setContentType(this.contentType);
47
48         ...
49         return request;
50     }
51
```


(4-5) 执行MockMvc对象的perform方法 (传入一个RequestBuilder对象参数) , 此时perform方法内部会调用RequestBuilder对象的buildRequest方法来创建出完整请求对象, 发出的请求会调用到controller方法的逻辑代码。

- perform方法返回了ResultActions类型的对象, 可以通过再调用ResultActions类型的对象的andExpect、andDo、andReturn方法对返回结果做验证, 即提供了统一的验证方式。
 - **ResultActions.andExpect方法**: 执行完成后添加断言。添加ResultMatcher结果匹配器验证规则, 验证控制器执行完成后结果是否正确。
 - **ResultActions.andDo方法**: 添加一个结果处理器。比如此处使用.andDo(MockMvcResultHandlers.print())输出整个响应结果信息, 可以在调试的时候使用。
 - ResultHandler是用于对处理的结果进行相应处理的, 比如输出整个请求/响应等信息方便调试, Spring mvc测试框架提供了MockMvcResultHandlers静态工厂方法, 该工厂提供了ResultHandler print()返回一个输出MvcResult详细信息到控制台的ResultHandler实现。
 - **ResultActions.andReturn方法**: 执行完成后返回相应的结果。
- 使用ResultMatchers结果匹配器对象 (携带期望值) 来验证结果对象中的内容
 - **MockMvcResultMatchers的API (静态方法)**, 来获取不同类型的ResultMatchers结果匹配器对象, 来验证返回的结果。
 - content()静态方法: 拿到ContentResultMatchers结果验证器对象 (含有结果的内容) , 使用ContentResultMatchers对请求返回的内容进行验证。

- `status()`静态方法：拿到`StatusResultMatchers`结果验证器对象（含有结果的状态），使用`StatusResultMatchers`对请求返回的结果状态进行验证。
- `handler()`静态方法：拿到`HandlerResultMatchers`结果验证器对象（含有处理请求的`Handler`处理器，比如验证处理器类型/方法名），使用`HandlerResultMatchers`对请求的`Handler`处理器进行验证。
- `request()`静态方法：拿到`RequestResultMatchers`结果验证器对象（含有`request`请求），使用`RequestResultMatchers`结果验证器对象对请求进行验证。
- `model()`静态方法：得到`ModelResultMatchers`模型验证器对象（含有`Model`数据），使用`ModelResultMatchers`对象对`Model`数据进行验证
- `view()`静态方法：得到`ViewResultMatchers`视图验证器；
- `flash()`静态方法：得到`FlashAttributeResultMatchers`Flash属性验证；
- `header()`静态方法：得到`HeaderResultMatchers`，响应Header验证器对象；
- `cookie()`静态方法：得到`CookieResultMatchers`，响应Cookie验证器；
- `content()`静态方法：得到`ContentResultMatchers`，响应内容验证器；
- `JsonPathResultMatchers` `jsonPath(String expression, Object ... args)`
`ResultMatcher jsonPath(String expression, Matcher matcher)`：得到Json表达式验证器；
- `XpathResultMatchers` `xpath(String expression, Object... args)`
`XpathResultMatchers xpath(String expression, Map<string, string=""> namespaces, Object... args)`：得到Xpath表达式验证器；
- `ResultMatcher forwardedUrl(final String expectedUrl)`：验证处理完请求后转发的url（绝对匹配）；
- `ResultMatcher forwardedUrlPattern(final String urlPattern)`：验证处理完请求后转发的url（Ant风格模式匹配，@since spring4）；
- `ResultMatcher redirectedUrl(final String expectedUrl)`：验证处理完请求后重定向的url（绝对匹配）；

- ResultMatcher redirectedUriPattern(final String expectedUri): 验证处理完请求后重定向的url (Ant风格模式匹配, @since spring4);

复制代码

```
1 public final class MockMvc {
2
3     public ResultActions perform(RequestBuilder requestBuilder) {
4         MockHttpServletRequest mvcRequest =
5             requestBuilder.buildRequest(this.servletContext);
6
7         MvcResult mvcResult =
8             new DefaultMvcResult(request, new
9             MockHttpServletRequest());
10
11         return new ResultActions() {
12             @Override
13             public ResultActions andExpect(ResultMatcher matcher)
14             {
15                 matcher.match(mvcResult);
16                 return this;
17             }
18             @Override
19             public ResultActions andDo(ResultHandler handler){
20                 handler.handle(mvcResult);
21                 return this;
22             }
23             @Override
24             public MvcResult andReturn() {
25                 return mvcResult;
26             }
27         };
28     }
29 }
```

(5) 通用测试案例

- 引入spring-test依赖包，在测试类上加入@AutoConfigureMockMvc，然后依赖注入MockMvc对象即可。
 - 准备测试环境
 - 通过MockMvc对象的perform方法执行请求，方法返回MvcResult对象
 - 添加验证断言
 - 添加结果处理器
 - 得到MvcResult，进行自定义断言/下一步异步请求
 - 卸载测试环境

[复制代码](#)

```
1 import com.fasterxml.jackson.databind.ObjectMapper
2 import org.junit.jupiter.api.Assertions
3 import org.junit.jupiter.api.BeforeEach
4 import org.junit.jupiter.api.Test
5 import org.springframework.beans.factory.annotation.Autowired
6 import
    org.springframework.boot.test.autoconfigure.web.servlet.AutoConf
    igureMockMvc
7 import org.springframework.boot.test.context.SpringBootTest
8 import org.springframework.http.MediaType
9 import org.springframework.test.web.servlet.MockMvc
10 import
    org.springframework.test.web.servlet.request.MockMvcRequestBuild
```

```
ers
11 import
    org.springframework.test.web.servlet.result.MockMvcResultMatchers
    s
12 import
    org.springframework.test.web.servlet.setup.DefaultMockMvcBuilder
13 import
    org.springframework.test.web.servlet.setup.MockMvcBuilders
14 import org.springframework.web.context.WebApplicationContext
15
16 @SpringBootTest(webEnvironment =
    SpringBootTest.WebEnvironment.RANDOM_PORT)
17 @AutoConfigureMockMvc
18 class xxxTest {
19
20     @Autowired
21     lateinit var webApplicationContext: WebApplicationContext
22
23     @Autowired
24     lateinit var objectMapper: ObjectMapper
25
26
27     lateinit var mvc: MockMvc
28
29
30     @BeforeEach
31     fun before() {
32         // 所有测试方法执行之前，都会先构建一个含有上下文环境的MockMvc对象
33         mvc = MockMvcBuilders
34             .webAppContextSetup(this.webApplicationContext)
35             .build()
36     }
```

```

37
38
39     private fun buildHeaders(httpHeaders: HttpHeaders) {
40         httpHeaders.contentType =
41             MediaType.APPLICATION_JSON
42         httpHeaders.add("name1", "value1")
43     }
44
45     @Test
46     fun xxxTest() {
47         //1、请求参数: 对象、集合
48         val param = xxx;
49
50         ///////////////////////////////////////////////////
51         ///////////////////////////////////////////////////
52         //2、模拟http请求
53         mvc.perform(MockMvcRequestBuilders.get("/user/{id}",
54             1));
55
56         //POST请求的参数也可以是post("/user?name=zhang"))
57         mvc.perform(
58             MockMvcRequestBuilders.post("/user")
59                 .param("name", "zhang")
60         );
61
62         mvc.perform(
63             MockMvcRequestBuilders.post("/user")
64                 .contentType(MediaType.APPLICATION_JSON)
65                 .content(
66                     objectMapper.writeValueAsString(param)
67                     /JSON.toJSONString(param)
68                     /jsonStr

```

```

64         )
65     );
66
67     val file =
68         MockMultipartFile("filename",FileInputStream(File("d:/xxx")))
69     mvc.perform(
70         MockMvcRequestBuilders
71             .multipart(HttpMethod.POST,"/upload")
72             .file(file)
73             .param("paramName", "paramValue")
74             .header("KEY", "VALUE")
75             .contentType(MediaType.MULTIPART_FORM_DATA_VALUE)
76     )
77     //////////////////////////////////////
78     //////////////////////////////////////
79     mvc.perform(
80         MockMvcRequestBuilders.post("/request/ip")
81             .contentType(MediaType.APPLICATION_JSON_VALUE) //
            请求数据格式
82             .accept(MediaType.APPLICATION_JSON_VALUE) //返回数
            据格式
83             .header("X-Forwarded-For", "127.0.0.1")
84             headers {
85                 //lambda表达式写法, 这个this代表一个内部创建好的
            HttpHeaders对象
86                 buildHeaders(this)
87             }
88
89             .content(objectMapper.writeValueAsString(param)) //请求内容
90     )

```

```
90
91      //3、请求返回
92      //③①andExpect的写法一(lamada表达式), 可以直接操作MvcResult结
      果对象中的方法来获取返回的数据, 通过断言, 将期望值与结果对象中的返回数据做比
      对
93      .andExpect { mvcResult →
94          assertEquals(HttpStatus.OK.value(),
      mvcResult.response.status)
95          assertEquals("xxx",
      mvcResult.response.contentAsString)
96      }
97
98      //③②andExpect的写法二, MockMvcResultMatchers.content()会
      获取一个ContentResultMatchers结果匹配器对象, .string("xxx")会携带期望
      值, string()内部会使用assertEquals断言, 将携带的期望值和内部的MvcResult
      结果对象中的值比对
99
      .andExpect(MockMvcResultMatchers.content().string("expectValue")
      )
100
      .andExpect(MockMvcResultMatchers.content().contentType("applicat
      ion/json;charset=UTF-8"))
101      // 验证响应头中是否有参数名有token, 值为aa
102
      .andExpect(MockMvcResultMatchers.header().string("token", "aa"))
103      // 验证id字段是否为1, jsonPath的使用
104
      .andExpect(MockMvcResultMatchers.jsonPath("$.id").value(1))
105      // 验证{"id":1}是否符合表达式规则
106      .andExpect(MockMvcResultMatchers.jsonPath("表达式规则", "
      {"id":1}").isBoolean)
107      // 验证正确
```



```
108     .andExpect(MockMvcResultMatchers.status().isOk)
109         // 验证参数错误 (请求400了)
110     .andExpect(MockMvcResultMatchers.status().isBadRequest)
111         // 验证控制器不存在
112     .andExpect(MockMvcResultMatchers.status().isNotFound)
113         // 验证网关错误
114     .andExpect(MockMvcResultMatchers.status().isBadGateway)
115         // 验证服务器内部错误
116
117     .andExpect(MockMvcResultMatchers.status().isInternalServerError)
118
119     // 验证执行的控制器类型
120     .andExpect(handler().handlerType(UserController.class))
121     // 验证执行的控制器方法名
122     .andExpect(handler().methodName("create"))
123     // 验证存储模型数据
124
125     .andExpect(MockMvcResultMatchers.model().attributeExists("user"))
126 )
127     // 验证viewName视图名
128
129     .andExpect(MockMvcResultMatchers.view().name("user/view"))
130
131     .andExpect(MockMvcResultMatchers.view().name("redirect:/user"));
132
133     // 验证视图渲染时forward到的jsp
134     .andExpect(MockMvcResultMatchers.forwardedUrl("/WEB-INF/jsp/user/view.jsp"))
135
136     // 4、输出MvcResult结果对象到控制台，方便调试
137     .andDo(print());
138
139 }
```

```
131         //5、也可以将perform的执行返回的MvcResult结果对象返回出去赋值给
        response, 在外部通过自定义断言Assertions将期望值和结果对象进行比对
132         .andReturn();
133
134         val resultObj = objectMapper.readValue(response,XXX对
        象::class.java)
135         assertEquals("参数对象属性值", resultObj.xxx属性)
136     }
137 }
```

(6) MvcResult接口

- 接口中的方法，是用来获取执行完控制器后的整个结果，并不仅仅是返回值，其包含了测试时需要的所有信息
- 通过实现类的对象的方法来获取返回的数据。

复制代码

```
1 public interface MvcResult {
2     MockHttpServletRequest getRequest():得到执行的请求;
3     MockHttpServletResponse getResponse():得到执行后的响应;
4     Object getHandler():得到执行的处理器，一般就是控制器;
5     HandlerInterceptor[] getInterceptors(): 得到对处理器进行拦截的拦截器;
6     ModelAndView getModelAndView(): 得到执行后的ModelAndView;
7     Exception getResolvedException(): 得到HandlerExceptionResolver解析后的异常;
8     FlashMap getFlashMap(): 得到FlashMap;
9     Object getAsyncResult()/Object getAsyncResult(long timeout):
    得到异步执行的结果;
10 }
```

java >