1、MockMvc进行单元测试

（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，构建上下文环境，web环境，注册一堆组件），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。

（3）依赖

复制代码

```java
testImplementation("org.springframework.boot:spring-boot-starter-test")
  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

```java

public final class MockMvcBuilders {

  private MockMvcBuilders() {}

  public static DefaultMockMvcBuilder webAppContextSetup(WebApplicationContext context) {
   return new DefaultMockMvcBuilder(context);
  }

  public static StandaloneMockMvcBuilder standaloneSetup(Object... controllers){
    return new StandaloneMockMvcBuilder(controllers);
  }
}
```

java

（4-2）MockMvcBuilder对象，用来构建一个MockMvc对象

复制代码

```java
public interface MockMvcBuilder {
    MockMvc build();
}
```

java

（4-1）和（4-2）的总结

```java
this.mockMvc = MockMvcBuilders
    //创建含有上下文的DefaultMockMvcBuilder
    .webAppContextSetup(webApplicationContext)
    //创建含有指定的控制器的StandaloneMockMvcBuilder
    .standaloneSetup(this.controller)
    //上述两种创建方式选一种即可
    .build();
```

（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）

```java
public abstract class MockMvcRequestBuilders {

    //post请求
    public static MockHttpServletRequestBuilder post(URI uri) {
        return new MockHttpServletRequestBuilder(HttpMethod.POST,
uri);
    }
    //get请求
    public static MockHttpServletRequestBuilder get(URI uri) {
        return new MockHttpServletRequestBuilder(HttpMethod.GET,
uri);
    }
    //put请求
```

```java
    public static MockHttpServletRequestBuilder put(URI uri) {
        return new MockHttpServletRequestBuilder(HttpMethod.PUT,
uri);
    }
    //delete请求
    public static MockHttpServletRequestBuilder delete(URI uri) {
        return new MockHttpServletRequestBuilder(HttpMethod.DELETE,
uri);
    }
    //options请求
    public static MockHttpServletRequestBuilder options(URI uri) {
        return new MockHttpServletRequestBuilder(HttpMethod.OPTIONS,
uri);
    }
    //head请求
        return new MockHttpServletRequestBuilder(httpMethod, uri);
    }
}
```

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

```java
public class MockHttpServletRequestBuilder{
    private MockHttpSession session;
    private String characterEncoding;
```

```
    private byte[] content;
    private String contentType;
    private final MultiValueMap<String, Object> headers =
      new LinkedMultiValueMap<>();
    private final MultiValueMap<String, String> parameters =
      new LinkedMultiValueMap<>();
    private final MultiValueMap<String, String> queryParams =
      new LinkedMultiValueMap<>();
    private final List<Cookie> cookies = new ArrayList<>();
    private final List<Locale> locales = new ArrayList<>();
    private final Map<String, Object> requestAttributes =
      new LinkedHashMap<>();
    private final Map<String, Object> sessionAttributes =
      new LinkedHashMap<>();

    public MockHttpServletRequestBuilder characterEncoding(Charset
encoding){
        this.characterEncoding = encoding
    }
    public MockHttpServletRequestBuilder content(byte[] content){}
    public MockHttpServletRequestBuilder content(String content) {}
    public MockHttpServletRequestBuilder contentType(MediaType
contentType){}
    public MockHttpServletRequestBuilder contentType(String
contentType){}
    public MockHttpServletRequestBuilder header(String name,
Object... values){}
    public MockHttpServletRequestBuilder headers(HttpHeaders
httpHeaders){}
    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
52  }
```

（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FlashAttributeResultMatchersFlash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 redirectedUrlPattern(final String expectedUrl)：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

```
public final class MockMvc {

    public ResultActions perform(RequestBuilder requestBuilder) {
        MockHttpServletRequest mvcRequest =
            requestBuilder.buildRequest(this.servletContext);

```

```java
        MvcResult mvcResult =
          new DefaultMvcResult(request, new
MockHttpServletResponse());

        return new ResultActions() {
            @Override
            public ResultActions andExpect(ResultMatcher matcher) {
                matcher.match(mvcResult);
                return this;
            }
            @Override
            public ResultActions andDo(ResultHandler handler){
                handler.handle(mvcResult);
                return this;
            }
            @Override
            public MvcResult andReturn() {
                return mvcResult;
            }
        };
  }
```

java

（5）通用测试案例

- 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

```
import com.fasterxml.jackson.databind.ObjectMapper
import org.junit.jupiter.api.Assertions
import org.junit.jupiter.api.BeforeEach
import org.junit.jupiter.api.Test
import org.springframework.beans.factory.annotation.Autowired
import
org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigur
eMockMvc
import org.springframework.boot.test.context.SpringBootTest
import org.springframework.http.MediaType
import org.springframework.test.web.servlet.MockMvc
import
org.springframework.test.web.servlet.request.MockMvcRequestBuilders
import
org.springframework.test.web.servlet.result.MockMvcResultMatchers
import
org.springframework.test.web.servlet.setup.DefaultMockMvcBuilder
import org.springframework.test.web.servlet.setup.MockMvcBuilders
import org.springframework.web.context.WebApplicationContext

@SpringBootTest(webEnvironment =
SpringBootTest.WebEnvironment.RANDOM_PORT)
@AutoConfigureMockMvc
class xxxTest {

    @Autowired
    lateinit var webApplicationContext: WebApplicationContext

    @Autowired
```

```
    lateinit var objectMapper: ObjectMapper


    lateinit var mvc: MockMvc


    @BeforeEach
    fun before() {
        //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
        mvc = MockMvcBuilders
            .webAppContextSetup(this.webApplicationContext)
            .build()
    }


   private fun buildHeaders(httpHeaders: HttpHeaders) {
                httpHeaders.contentType =
MediaType.APPLICATION_JSON
httpHeaders.add("name1","value1")
   }

    @Test
    fun xxxTest() {
        //1、请求参数：对象、集合
        val param = xxx;

////////////////////////////////////////////////////////////////////
////////
        //2、模拟http请求
        mvc.perform(MockMvcRequestBuilders.get("/user/{id}", 1));

        //POST请求的参数也可以是post("/user?name=zhang"))
```

```
52
mvc.perform(MockMvcRequestBuilders.post("/user").param("name",
"zhang"));
53
54          mvc.perform(
55             MockMvcRequestBuilders.post("/user")
56            .contentType(MediaType.APPLICATION_JSON)
57            .content(
58                objectMapper.writeValueAsString(param)
59                /JSON.toJSONString(param)
60                /jsonStr
61             )
62          );
63
64        val file =
65
MockMultipartFile("filename",FileInputStream(File("d:/xxx")))
66        mvc.perform(
67            MockMvcRequestBuilders
68                .multipart(HttpMethod.POST,"/upload")
69                .file(file)
70                .contentType(MediaType.MULTIPART_FORM_DATA_VALUE)
71        )
72 ////////////////////////////////////////////////////////////////
///////
73
74          mvc.perform(
75              MockMvcRequestBuilders.post("/request/ip")
76                  .contentType(MediaType.APPLICATION_JSON_VALUE)
77                  .header("X-Forwarded-For",  "127.0.0.1")
78                  headers {
```

```
                    //lambda表达式写法，这个this代表一个内部创建好的
HttpHeaders对象
                    buildHeaders(this)
                }
                .content(objectMapper.writeValueAsString(param))
        )

         //3、请求返回
        //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
        .andExpect { mvcResult ->
            assertEquals(HttpStatus.OK.value(),
mvcResult.response.status)
            assertEquals("xxx", mvcResult.response.contentAsString)
        }

        //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

.andExpect(MockMvcResultMatchers.content().string("expectValue"))

.andExpect(MockMvcResultMatchers.content().contentType("application/
json;charset=UTF-8"))
          //验证响应头中是否有参数名有token，值为aa

.andExpect(MockMvcResultMatchers.header().string("token","aa"))
          //验证id字段是否为1，jsonPath的使用
        .andExpect(MockMvcResultMatchers.jsonPath("$id").value(1))
          //验证{\"id\":1}是否符合表达式规则
```

```
100         .andExpect(MockMvcResultMatchers.jsonPath("表达式规则","
    {\"id\":1}").isBoolean)
101           //验证正确
102         .andExpect(MockMvcResultMatchers.status().isOk)
103           //验证参数错误（请求400了）
104         .andExpect(MockMvcResultMatchers.status().isBadRequest)
105           //验证控制器不存在
106         .andExpect(MockMvcResultMatchers.status().isNotFound)
107           //验证网关错误
108         .andExpect(MockMvcResultMatchers.status().isBadGateway)
109           //验证服务器内部错误
110
    .andExpect(MockMvcResultMatchers.status().isInternalServerError)
111           //验证执行的控制器类型
112         .andExpect(handler().handlerType(UserController.class))
113           //验证执行的控制器方法名
114         .andExpect(handler().methodName("create"))
115           //验证存储模型数据
116
    .andExpect(MockMvcResultMatchers.model().attributeExists("user"))
117           //验证viewName视图名
118         .andExpect(MockMvcResultMatchers.view().name("user/view"))


119
    .andExpect(MockMvcResultMatchers.view().name("redirect:/user"));
120           //验证视图渲染时forward到的jsp
121         .andExpect(MockMvcResultMatchers.forwardedUrl("/WEB-
    INF/jsp/user/view.jsp"))
122         //4、输出MvcResult结果对象到控制台，方便调试
123         .andDo(print());
124
```

```java
        //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
        .andReturn();

        val resultObj = objectMapper.readValue(response,XXX对象::class.java)
        assertEquals("参数对象属性值", resultObj.xxx属性)
    }
}
```

java

（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

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

java