# SpringBoot整合Junit单元测试

公共数据平台-梁晓发

## pom中引入springboot的junit依赖

<!-- spring-boot-starter-test 模块 -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

## 单元测试代码

### 2.1编写单元测试类

**一个单元测试类测试一个Controller,命令规则为Controller名称后面加Test。比如UserController，单元测试类名为UserControllerTest。**

**一个完整的单元测试类代码如下。**

@RunWith(SpringJUnit4ClassRunner.**class**) // SpringJUnit的支持

@SpringBootTest(classes = { VQAMain.**class**, MockServletContext.**class** }) // VQAMain.class是springboot的启动类.

@WebAppConfiguration // web项目，让Junit模拟ServletContext

**public** **class** DemoControllerTest {

/\*\* 注入需要测试的Controller实例. \*/

@Autowired

**private** DemoController demoController;

/\*\* SpringBoot的Web应用上下文. \*/

@Autowired

**private** WebApplicationContext webApplicationContext;

/\*\* MockMvc对象 \*/

**private** MockMvc mockMvc;

/\*\*

\* 在单元测试方法(声明注解@Test的方法)之前执行.用于初始化资源

\*/

@Before

**public** **void** setup() {

mockMvc = MockMvcBuilders.*standaloneSetup*(**this**.demoController).build();

}

/\*\*

\* 在所有单元测试方法完毕之后执行.用于释放资源.

\*/

@After

**public** **void** tearDown() {

// ....

}

/\*\*

\* 模拟GET请求并传递请求参数.

\*/

@Test

**public** **void** testDemo01() **throws** Exception {

**final** String uri = "/demo01";// 请求路径

// 模拟Request的get请求

MockHttpServletRequestBuilder requestBuilder = MockMvcRequestBuilders.*get*(uri);

// 设置请求参数

requestBuilder.param("name", "Jack");

requestBuilder.param("age", "59");

MvcResult mvcResult = mockMvc.perform(requestBuilder).andReturn();

MockHttpServletResponse response = mvcResult.getResponse();

// 获取Response的响应状态码

**int** status = response.getStatus();

// 断言，如果失败则抛出异常,表示单元测试不通过

Assert.*assertTrue*(status == 200);

// 响应内容,即在Controller中调用方法完毕之后的返回值.返回json格式字符串

String content = response.getContentAsString();

JSONObject jsonObject = JSONObject.*parseObject*(content);

**int** responseStatus = jsonObject.getIntValue("responseStatus");

Assert.*assertTrue*(responseStatus == 0);

// 这里根据业务写更多的断言

}

/\*\*

\* 模拟post请求,传递请求参数和请求头参数

\*/

@Test

**public** **void** testDemo02() **throws** Exception {

**final** String uri = "/demo02";// 请求路径

// 模拟Request的post请求

MockHttpServletRequestBuilder requestBuilder = MockMvcRequestBuilders.*post*(uri);

// 设置Content-Type

requestBuilder.contentType(MediaType.***APPLICATION\_FORM\_URLENCODED***);

// 设置请求参数

requestBuilder.param("userName", "Jack");

requestBuilder.param("address", "59");

requestBuilder.param("num", "800");

requestBuilder.header("token", "AABBCCDDEE11223344");

MvcResult mvcResult = mockMvc.perform(requestBuilder).andReturn();

MockHttpServletResponse response = mvcResult.getResponse();

// 获取Response的响应状态码

**int** status = response.getStatus();

// 断言，如果失败则抛出异常,表示单元测试不通过

Assert.*assertTrue*(status == 200);

// 这里根据业务写更多的断言

}

/\*\*

\* 模拟post请求,传递请求参数和请求头参数

\*/

@Test

**public** **void** testDemo03() **throws** Exception {

**final** String uri = "/demo03";// 请求路径

// 模拟Request的post请求

MockHttpServletRequestBuilder requestBuilder = MockMvcRequestBuilders.*post*(uri);

// 设置Content-Type为application/json。

requestBuilder.contentType(MediaType.***APPLICATION\_JSON***);

// 设置请求参数

requestBuilder.param("desc", "ABCD");

// 设置请求头

requestBuilder.header("token", "AABBCCDDEE11223344");

// 设置Cookie.

requestBuilder.cookie(**new** Cookie("bbs\_token", "ODOAGDOFOUAGOFUA9ww36973469"));

// 设置RequestBody，只在Content-Type设置了APPLICATION\_JSON时有效,Controller中方法使用@RequestBody接收

requestBuilder.content("{\"tt\":\"123456\"}");

MvcResult mvcResult = mockMvc.perform(requestBuilder).andReturn();

MockHttpServletResponse response = mvcResult.getResponse();

// 获取Response的响应状态码

**int** status = response.getStatus();

// 断言，如果失败则抛出异常,表示单元测试不通过

Assert.*assertTrue*(status == 200);

// 这里根据业务写更多的断言

}

}

**创建单元测试类，运行单元测试时，会先启动SpringBoot的MVC容器，然后创建模拟Request的请求对象，调用Controller的相关方法执行。执行之后返回一个MockHttpServletResponse 对象，可以通过此对象获取响应状态码，以及响应数据包的内容，然后解析内容，根据业务逻辑编写断言。**

**相关注解：**

**@Before ：单元测试启动时调用，一般用于初始化资源**

**@After ：单元测试完毕之后调用**

**@Test : 单元测试的方法**

### 2.2单元测试Suite

可以把项目中所有的单元测试类整合起来，使用Suite包装，跑这个Suite，即可把项目中所有的单元测试类统一运行。

@RunWith(Suite.**class**)

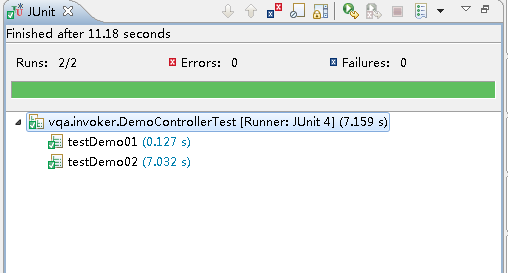
@SuiteClasses({ DemoControllerTest.**class**, UserControllerTest.**class** })

**public** **class** TestSuite {

}

### 2.3运行单元测试

**右键点击单元测试类，Run As-->JUnit Test**



### 2.4 规则

**2.3.1命名规则：**

**(1)单元测试类名，即为Controller类名后面加上Test，如UserControllerTest**

**(2) 单元测试方法名，根据Controller内部待测试的方法名，前面加上test。UserController内部有方法queryUser，那么单元测试的方法命名为testQueryUser()。**

**2.3.2 测试规则**

1. **单元测试方法内部必须设置断言。**
2. **单元测试的参数请求除了要走正确的流程，还需要走异常流程，也就是根据业务逻辑构造不同的参数，不断测试，每一次返回都需要断言。**
3. **单元测试内部的一个方法，对应Controller内部的一个方法，但是单元测试方法内部，可以构造不同的参数调用多次Controler内部的方法。**
4. **单元测试的成功与否在于测试方法有无抛出异常，因此，所有单元测试的方法必须throws Exception。**
5. **需要关注代码覆盖率（代码覆盖率的生成参见第4节）。原则上讲，代码覆盖率并不需要全体代码100%覆盖，但是待测试的Controller内部的代码覆盖率必须超过90%。**

**代码覆盖率越高，测试越全面。**

## 单元测试报告

### 3.1引入依赖

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-plugin</artifactId>

<configuration>

<skip>false</skip>

</configuration>

</plugin>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-site-plugin</artifactId>

</plugin>

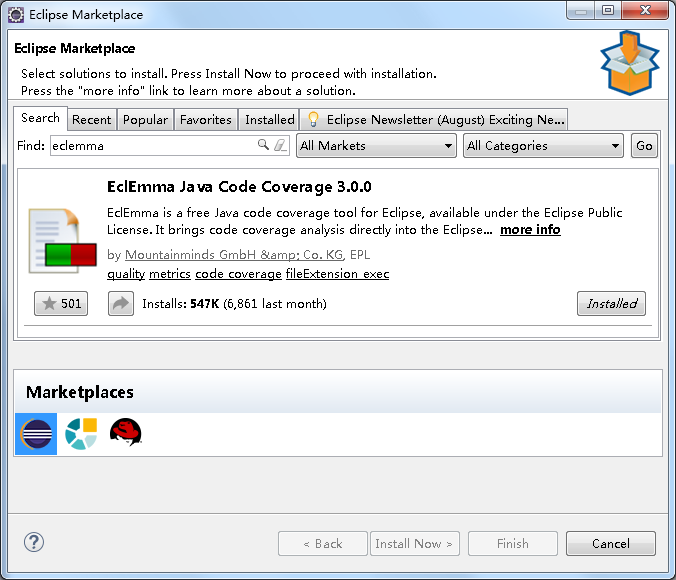
### 3.2执行命令

在工程根目录中执行命令：mvn surefire-report:report

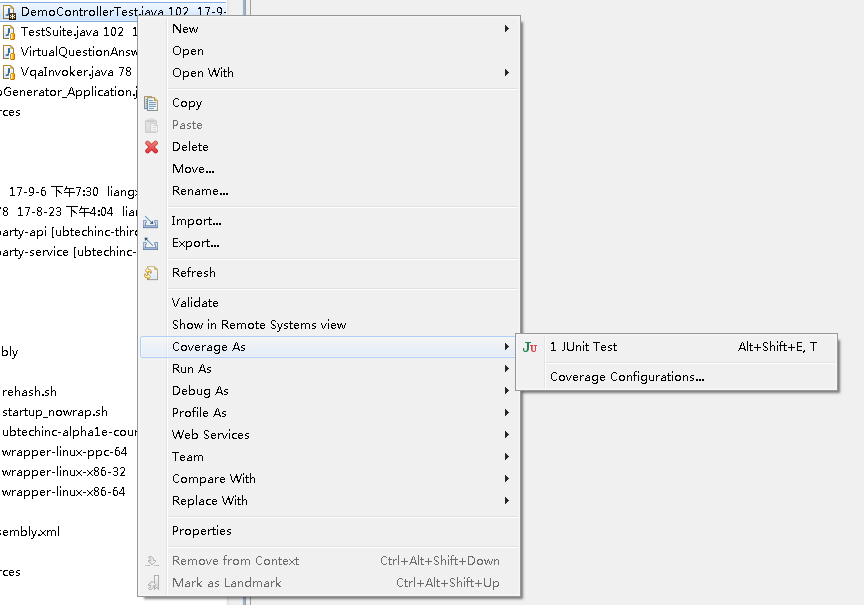
报告生成在${projectDir}/target/site/surefire-report.html

## 单元测试覆盖率报告

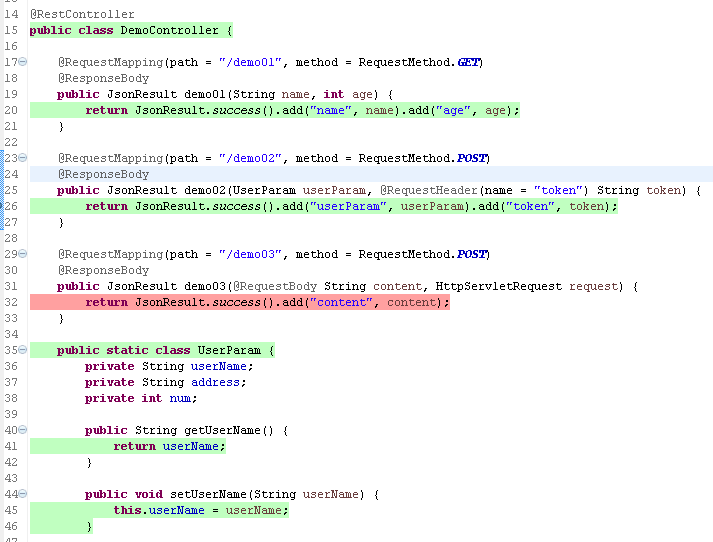
安装插件eclemma



安装成功后，在单元测试类中点击右键，Coverage As -->JUnit Test



单元测试完毕之后，会在代码中标注覆盖率颜色，如下图所示：



绿色是在单元测试中跑过的代码流程，红色表示还未覆盖。

覆盖率报告视图.

点击右键导出HTML报告

