6/23/2018 11:57:31 PM

（一）JUnit5 介绍与安装

官方网址：http://junit.org/junit5/

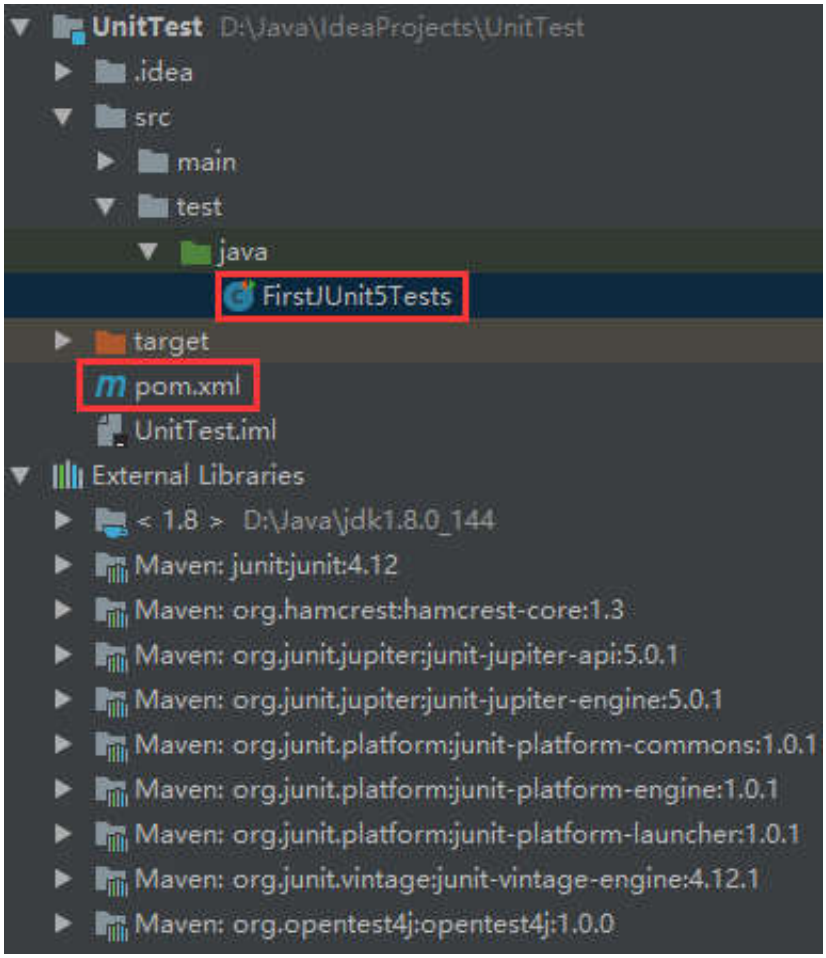
JUnit 5 = JUnit Platform + JUnit Jupiter + JUnit Vintage

> **JUnit Platform** 是在JVM上启动测试框架的基础。
>
> **JUnit Jupiter** 是JUnit5扩展的新的编程模型和扩展模型，用来编写测试用例。
>
> **JUnit Vintage** 提供了一个在平台上运行 **JUnit3** 和 **JUnit4** 的 **TestEngine**。

创建maven项目，在pom.xml添加如下的配置

```
<dependencies>

    <dependency>
        <groupId>org.junit.platform</groupId>
        <artifactId>junit-platform-launcher</artifactId>
        <version>1.2.0</version>
        <scope>test</scope>
    </dependency>

    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter-engine</artifactId>
        <version>5.2.0</version>
        <scope>test</scope>
    </dependency>

    <dependency>
        <groupId>org.junit.vintage</groupId>
        <artifactId>junit-vintage-engine</artifactId>
        <version>5.2.0</version>
        <scope>test</scope>
    </dependency>

</dependencies>
```

（二）JUnit5 创建测试




```
//导入测试测试注解（@Test）和断言方法（assertEquals）的路径不同。
import static org.junit.jupiter.api.Assertions.assertEquals;

import org.junit.jupiter.api.Test;

//测试类不需要声明为public
class FirstJUnit5Tests {

    @Test
    ////测试方法不需要声明为public
    void myFirstTest() {
        assertEquals(2, 1 + 1);
    }

}
```

（三）JUnit5 新的用法

```
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertTrue;
import static org.junit.jupiter.api.Assertions.assertAll;

import org.junit.jupiter.api.*;
//创建 JUnit5NewTests 测试类
```

```
class JUnit5NewTests {

    //循环之前
    @BeforeEach
    @DisplayName("每条用例开始时执行")
    void start(){

    }

    //循环之后
    @AfterEach
    @DisplayName("每条用例结束时执行")
    void end(){

    }

    @Test
    void myFirstTest() {
        assertEquals(2, 1 + 1);
    }

    @Test
    @DisplayName("描述测试用例")
    void testWithDisplayName() {

    }

    @Test
    @Disabled("这条用例暂时跑不过，忽略!")
    void myFailTest(){
        assertEquals(1,2);
    }

    @Test
    @DisplayName("运行一组断言")
    public void assertAllCase() {
        assertAll("groupAssert",
                () -> assertEquals(2, 1 + 1),
                () -> assertTrue(1 > 0)
        );
    }

    @Test
    @DisplayName("依赖注入1")
    public void testInfo(final TestInfo testInfo) {
        System.out.println(testInfo.getDisplayName());
    }

    @Test
```

```
    @DisplayName("依赖注入2")
    public void testReporter(final TestReporter testReporter) {
        testReporter.publishEntry("name", "Alex");
    }

}
```

（四）JUnit 注解之Rule

一个JUnit Rule就是一个实现了TestRule的类，作用类似于 @Before、@After

可以避免代码重复，可以动态获取将要运行的测试类、测试方法的信息

除了增加Rule特性，新版JUnit还添加了很多核心Rule

- TemporaryFolder：提前创建文件与目录并且会在测试运行结束后将其删除。
- ExternalResource：提前建立好资源并且会在测试结束后将其销毁。
- ErrorCollector：在失败后继续运行并在测试结束时报告所有错误。
- ExpectedException：指定期望的异常类型与消息。
- Timeout：为类中的所有测试应用相同的超时时间。

  import org.junit.Rule; import org.junit.Test; import org.junit.rules.Timeout;

  public class RuleTestDemo {

```
//使用Timeout这个Rule
@Rule
public Timeout timeout = new Timeout(1000);

@Test
public void testMethod1() throws Exception {
    Thread.sleep(1001);
}

@Test
public void testMethod2() throws Exception {
    Thread.sleep(999);
}
```

  }

自定义的Rule

implement一个TestRule 接口，并实现apply()方法。该方法需要返回一个Statement对象

```
import org.junit.rules.TestRule;
import org.junit.runner.Description;
import org.junit.runners.model.Statement;
```

```
//自定义一个rule类
public class MethodNameRule implements TestRule {

    public Statement apply(final Statement base, final Description description) {
        return new Statement() {
            @Override
            public void evaluate() throws Throwable {
                //在测试方法运行之前做一些事情，在base.evaluate()之前
                String className = description.getClassName();
                String methodName = description.getMethodName();

                base.evaluate();  //运行测试方法

                //在测试方法运行之后做一些事情，在base.evaluate()之后
                System.out.println("Class name:"+className+",
                        method name: "+methodName);
            }
        };
    }
}
```

```
public class RuleTestDemo {

    //使用Timeout这个Rule
    @Rule
    public Timeout timeout = new Timeout(1000);

    //使用自定义Rule
    @Rule
    //MethodNameRule作用 打印当前测试用例的类名和方法名
    public MethodNameRule methodNameRule = new MethodNameRule();

}
```