Junit分享

1. Junit

Junit是一个java编程语言的单元测试框架。 Junit测试是程序员测试，即所谓的白盒测试。

因为程序员知道被测试的软件如何（How）完成功能和完成什么样（What）的功能。Junit是一套框架，继承TestCase类，就可以用Junit进行自动测试了。

Junit的优点

JUnit的好处在于可以对代码进行单元测试。

并且可以根据单元测试生成单元测试报告。  
  
比代码中写main 方法测试的好处：  
  
1. 可以书写一系列的 测试方法，对项目所有的 接口或者方法进行单元测试。   
2.启动后，自动化测试，并判断执行结果, 不需要人为的干预。   
3.只需要查看最后结果，就知道整个项目的方法接口是否通畅。。  
4.每个单元测试用例相对独立， 由Junit 启动，自动调用。 不需要添加额外的调用语句。  
5.添加，删除，屏蔽测试方法，不影响其他的测试方法。 开源框架都对JUnit 有相应的支持。

**注意：编写测试类的原则：**

**①测试方法上必须使用@Test进行修饰**

**②测试方法必须使用public void 进行修饰，不能带任何的参数**

**③新建一个源代码目录来存放我们的测试代码，即将测试代码和项目业务代码分开**

**④测试类所在的包名应该和被测试类所在的包名保持一致**

**⑤测试单元中的每个方法必须可以独立测试，测试方法间不能有任何的依赖**

**⑥测试类使用Test作为类名的后缀（不是必须）**

**⑦测试方法使用test作为方法名的前缀（不是必须）**

1. Junit的基本标签

@RunWith：指定使用的单元测试执行类

@Before：初始化方法

@After：释放资源

@Test：测试方法，在这里可以测试期望异常和超时时间

@Ignore：忽略的测试方法

@BeforeClass：针对所有测试，只执行一次，且必须为static void

@AfterClass：针对所有测试，只执行一次，且必须为static void

| **@BeforeClass and @AfterClass** | **@Before and @After** |
| --- | --- |
| 在一个类中只可以出现一次 | 在一个类中可以出现多次，即可以在多个方法的声明前加上这两个Annotaion标签，执行顺序不确定 |
| 方法名不做限制 | 方法名不做限制 |
| 在类中只运行一次 | 在每个测试方法之前或者之后都会运行一次 |
| @BeforeClass父类中标识了该Annotation的方法将会先于当前类中标识了该Annotation的方法执行。 @AfterClass 父类中标识了该Annotation的方法将会在当前类中标识了该Annotation的方法之后执行 | @Before父类中标识了该Annotation的方法将会先于当前类中标识了该Annotation的方法执行。 @After父类中标识了该Annotation的方法将会在当前类中标识了该Annotation的方法之后执行 |
| 必须声明为public static | 必须声明为public 并且非static |
| 所有标识为@AfterClass的方法都一定会被执行，即使在标识为@BeforeClass的方法抛出异常的的情况下也一样会。 | 所有标识为@After 的方法都一定会被执行，即使在标识为 @Before 或者 @Test 的方法抛出异常的的情况下也一样会。 |

**每一个测试用例执行顺序为:**

@BeforeClass -> @Before -> @Test -> @After -> @AfterClass;   
**每一个测试方法的调用顺序为:**   
@Before -> @Test -> @After;

Demo：Junit4Test

1. assertThat、Hamcrest 、JUnitMatchers、CoreMatchers

JUnit中的部分断言的可读性并不是很好，有时我们不得不自己编写表达式并断言其结果，并且因为我们没有提供失败的信息，当这个断言失败时只会抛出java.lang.AssertionError，无法知道到底是哪一部分出错

Demo : TestClass.test1()

**JUnit4.4**引入了Hamcrest框架，Hamcrest是一个书写匹配器对象时允许直接定义匹配规则的框架，它提供了一套匹配符Matcher，这些匹配符更接近自然语言，可读性高，更加灵活。并且使用全新的断言语法：assertThat，结合Hamcest提供的匹配符，只用这一个方法，就可以实现所有的测试。

assertThat**语法如下：**

* **assertThat(T actual, Matcher matcher);**
* **assertThat(String reason, T actual, Matcher matcher);**

Hamcrest 提供了很强大的一些api 供我们进行测试断言



Demo : TestClass.test1()

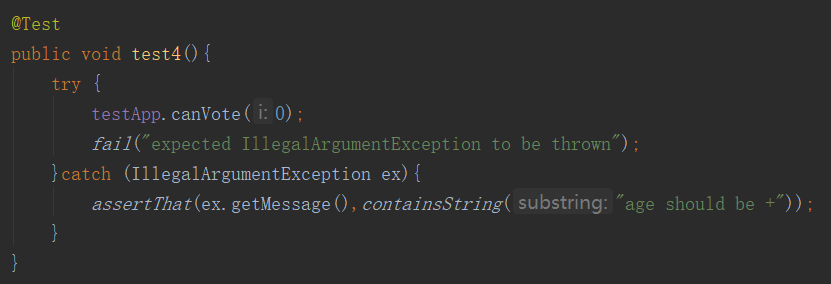
Hamcrest匹配符demo

https://www.cnblogs.com/softidea/p/5011527.html

1. 对异常的测试

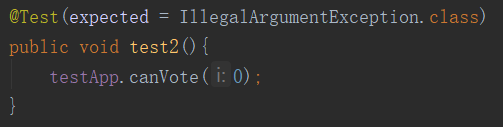
对异常的测试有三种方法

1. **try…fail...catch…**



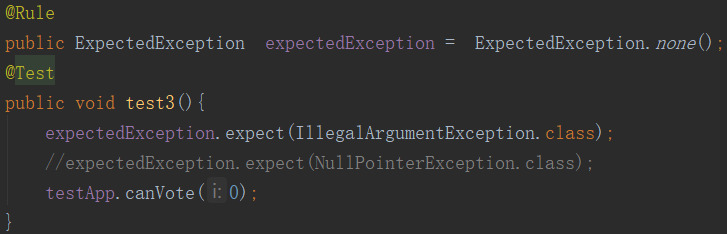
这种写法看上去和实现类的写法很相似，当没有异常被抛出的时候fail方法会被调用，输出测试失败的信息

1. **@Test(expected=xxx)**



这种写法看上去简单了一些，但是它有一个潜在的问题：当被标记的这个测试方法中的任何一个操作抛出了相应的异常时，这个测试就会通过。这就意味着有可能抛出异常的地方并不是我们期望的那个操作。虽然这种情况可以在写test case的时候人为的避免，但是还是有更好的方法来测试异常抛出。

1. **ExpectedException Rule**



这种方法除了可以指定期望抛出的异常类型之外还可以指定在抛出异常时希望同时给出的异常信息。它需要在测试之前使用Rule标记来指定一个ExpectedException，并在测试相应操作之前指定期望的Exception类型（如IllegalArgumentException.class）

@Rule 允许灵活添加或重新定义测试类中的每个测试方法的行为

当我们越来越多需要进行单元测试时，就需要使用 Rule 来帮忙了。其主要目的是针对一个测试类中的每个单元测试方法进行统一添加一些行为。代码则使用 @Rule 注解的形式来添加至类的属性上。

Demo : TestClass.test3()

1. Junit对Springboot工程的测试

因为我们是Web项目，所以在创建的Java类中添加注解：@RunWith(SpringJUnit4ClassRunner.class)

// SpringJUnit支持，由此引入Spring-Test框架支持！

@SpringBootTest(classes = SpringBootSampleApplication.class)

// 指定我们SpringBoot工程的Application启动类

@WebAppConfiguration

// 由于是Web项目，Junit需要模拟ServletContext，因此我们需要给我们的测试类加

1. 对Controller的测试

Mockito  Java 单元测试 Mock 框架

对被测试类的单个方法进行测试，当遇到依赖的外部资源时，对外部资源进行打桩、模拟。从而达到对该方法的逻辑正确性进行测试的目的。

1. Junit的cobertura-maven-plugin Maven插件实现测试报告

配置文件