链滴

Junit5 介绍，Junit4 升级 Junit5 注意事项

作者：JayGao

原文链接：https://ld246.com/article/1651893769281

来源网站：链滴

前言

由于项目中经常使用Junit4和Junit5，最近想详细的总结一下区别和使用，特此写下文章记录一下。

参考文章：

- 知乎： 从JUnit 4迁移到JUnit 5：重要的区别和好处 - 知乎 (zhihu.com)
- 博客园： Springboot集成JUnit5优雅进行单元测试 - 海向 - 博客园 (cnblogs.com)
- 官网： https://junit.org/junit5/docs/current/user-guide/
- blogs-oracle： Migrating from JUnit 4 to JUnit 5: Important Differences and Benefits (oracle.com)

什么是Junit5

首先就得聊下 Java 单元测试框架 JUnit，它与另一个框架 TestNG 占据了 Java领域里单元测试框架主要市场，其中 JUnit 有着较长的发展历史和不断演进的丰富功能，备受大多数 Java 开发者的青睐。而说到 JUnit 的历史，JUnit 起源于 1997年，最初版本是由两位编程大师 Kent Beck 和 Erich Gamma 的一次飞机之旅上完成的，由于当时 Java 测试过程中缺乏成熟的工具，两人在飞机上就合作设计实了 JUnit 雏形，旨在成为更好用的 Java 测试框架。如今二十多年过去了，JUnit 经过各个版本迭代演，已经发展到了 5.x 版本，为 JDK 8以及更高的版本上提供更好的支持 (如支持 Lambda) 和更丰富测试形式 (如重复测试，参数化测试)。

改进和新功能

JUnit 5 是对 JUnit 框架的强大而灵活的更新，它提供了各种改进和新功能来组织和描述测试用例，且有助于理解测试结果。升级到 JUnit 5 非常简单快捷：只需更新项目依赖项，就可以开始使用新功。

如果您已经使用 JUnit 4 一段时间了，那么迁移测试看起来是一项艰巨的任务。有个好消息，您可能需要转换任何测试；JUnit 5 可以使用Vintage库运行 JUnit 4 测试。

话虽如此，以下是开始使用 JUnit 5 编写新测试的四个强有力的理由：

- JUnit 5 利用 Java 8 或更高版本的特性（如 lambda 函数），使测试更强大、更易于维护。
- JUnit 5 添加了一些非常有用的新功能，用于描述、组织和执行测试。例如，测试获得更好的显示称，并且可以分层组织。
- JUnit 5 被组织到多个库中，所以只有您需要的功能才会导入到项目中。使用Maven和Gradle等构系统，很容易包含正确的库。
- JUnit 5 可以同时使用多个扩展，而 JUnit 4 不能（一次只能使用一个运行器runner）。这意味着可以轻松地将Spring扩展与其他扩展（例如您自己的自定义扩展）组合在一起。

从 JUnit 4 切换到 JUnit 5 非常简单，即使您有现有的 JUnit 4 测试。除非需要新功能，否则大多数织不需要将旧的 JUnit 测试转换为 JUnit 5。在这种情况下，请使用以下步骤：

1. 更新您的库并将系统从 JUnit 4 构建为 JUnit 5。请务必在测试运行时路径中包含 junit-vintage 引项目，才可以允许执行现有测试。
2. 使用新的 JUnit 5 构造开始构建新的测试。
3. （可选）将 JUnit 测试转换为 JUnit 5。

为什么使用Junit5

- JUnit4被广泛使用，但是许多场景下使用起来语法较为繁琐，JUnit5中支持lambda表达式，语法单且代码不冗余。
- JUnit5易扩展，包容性强，可以接入其他的测试引擎。
- 功能更强大提供了新的断言机制、参数化测试、重复性测试等新功能。
- ps：开发人员为什么还要测试，单测写这么规范有必要吗？其实单测是开发人员必备技能，只不过多开发人员开发任务太重导致调试完就不管了，没有系统化得单元测试，单元测试在系统重构时能发巨大的作用，可以在重构后快速测试新的接口是否与重构前有出入。

JUnit5结构

JUnit 5可以说是 JUnit 单元测试框架的一次重大升级，首先需要 Java 8 以上的运行环境，虽然在旧本 JDK 也能编译运行，但要完全使用 JUnit 5 功能， JDK 8 环境是必不可少的。

JUnit 5 与以前版本的 JUnit 不同，拆分成由三个不同子项目的几个不同模块组成：

JUnit 5 = JUnit Platform + JUnit Jupiter + JUnit Vintage

- **JUnit Platform**： 这是Junit提供的平台功能模块，通过它，其它的测试引擎都可以接入Junit实接口和执行。
- **JUnit JUpiter**：这是JUnit5的核心，是一个基于JUnit Platform的引擎实现，它包含许多丰富的特性来使得自动化测试更加方便和强大。
- **JUnit Vintage**：这个模块是兼容JUnit3、JUnit4版本的测试引擎，使得旧版本的自动化测试也可在JUnit5下正常运行。

SpringBoot中使用Junit5

maven中pom.xml中添加依赖spring-boot-starter-test，说明如下：

```
<!-- spring boot 测试支持 -->
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-test</artifactId>
   <scope>test</scope>
   <!-- 排除Junit4、Junit3的单元测试引擎，新版本SpringBoot默认使用JUnit5的引擎 -->
   <!-- junit-jupiter-engine 是 JUnit5 中默认使用的测试引擎 -->
   <!-- junit-vintage-engine 是 Junit5 中用于支持和适配需要使用JUnit 4、3 的测试引擎 -->
   <exclusions>
      <exclusion>
         <groupId>org.junit.vintage</groupId>
         <artifactId>junit-vintage-engine</artifactId>
      </exclusion>
   </exclusions>
</dependency>
```

常用注解

- @BeforeEach：在每个单元测试方法执行前都执行一遍
- @BeforeAll：在每个单元测试方法执行前执行一遍（只执行一次）
- @DisplayName("商品入库测试")：用于指定单元测试的名称
- @Disabled：当前单元测试置为无效，即单元测试时跳过该测试
- @RepeatedTest(n)：重复性测试，即执行n次
- @ParameterizedTest：参数化测试，
- @ValueSource(ints = {1, 2, 3})：参数化测试提供数据

断言

JUnit Jupiter提供了强大的断言方法用以验证结果，在使用时需要借助java8的新特性lambda表达式
均是来自org.junit.jupiter.api.Assertions包的static方法。

assertTrue与assertFalse用来判断条件是否为true或fals

<pre highlighted="true"> Copy <code class="language-java hljs" id="copy_target_1">
@Test
    @DisplayName("测试断言equals")
    void testEquals() {
        assertTrue(3 < 4);
    }</code></pre>

assertNull与assertNotNull用来判断条件是否为·null

```
Copy
    @Test
    @DisplayName("测试断言NotNull")
```

原文链接：Junit5 介绍，Junit4 升级 Junit5 注意事项

```
    void testNotNull() {
        assertNotNull(new Object());
    }
```

assertThrows用来判断执行抛出的异常是否符合预期，并可以使用异常类型接收返回值进行其他操作

```
Copy
    @Test
    @DisplayName("测试断言抛异常")
    void testThrows() {
        ArithmeticException arithExcep = assertThrows(ArithmeticException.class, () -> {
            int m = 5/0;
        });
        assertEquals("/ by zero", arithExcep.getMessage());
    }
```

assertTimeout用来判断执行过程是否超时

```
<pre highlighted="true"><div class="esa-clipboard-button" data-clipboard-target="#copy_t
rget_4" title="复制代码">Copy</div><code class="language-java hljs" id="copy_target_4">
<span class="hljs-meta">@Test</span>
    <span class="hljs-meta">@DisplayName("测试断言超时")</span>
    <span class="hljs-keyword">void</span> <span class="hljs-title function_">testTimeOut<
span><span class="hljs-params">()</span> {
        <span class="hljs-type">String</span> <span class="hljs-variable">actualResult</spa
> <span class="hljs-operator">=</span> assertTimeout(ofSeconds(<span class="hljs-numbe
">2</span>), () -> {
            Thread.sleep(<span class="hljs-number">1000</span>);
            <span class="hljs-keyword">return</span> <span class="hljs-string">"a result"</sp
n>;
        });
        System.out.println(actualResult);
    }
</code></pre>
```

assertAll是组合断言，当它内部所有断言正确执行完才算通过

```
<pre highlighted="true"><div class="esa-clipboard-button" data-clipboard-target="#copy_t
rget_5" title="复制代码">Copy</div><code class="language-java hljs" id="copy_target_5">
<span class="hljs-meta">@Test</span>
    <span class="hljs-meta">@DisplayName("测试组合断言")</span>
    <span class="hljs-keyword">void</span> <span class="hljs-title function_">testAll</spa
><span class="hljs-params">()</span> {
        assertAll(<span class="hljs-string">"测试item商品下单"</span>,
                () -> {
                    <span class="hljs-comment">//模拟用户余额扣减</span>
                    assertTrue(<span class="hljs-number">1</span> < <span class="hljs-number">
</span>, <span class="hljs-string">"余额不足"</span>);
                },
                () -> {
                    <span class="hljs-comment">//模拟item数据库扣减库存</span>
                    assertTrue(<span class="hljs-number">3</span> < <span class="hljs-number">
</span>);
                },
```

```
            () -> {
                <span class="hljs-comment">//模拟交易流水落库</span>
                assertNotNull(<span class="hljs-keyword">new</span> <span class="hljs-title c
ass_">Object</span>());
            }
        );
    }
</code></pre>
```

重复性测试

在许多场景中我们需要对同一个接口方法进行重复测试，例如对幂等性接口的测试。

JUnit Jupiter通过使用@RepeatedTest(n)指定需要重复的次数

```
<pre highlighted="true"><div class="esa-clipboard-button" data-clipboard-target="#copy_t
rget_6" title="复制代码">Copy</div><code class="language-java hljs" id="copy_target_6">
<span class="hljs-meta">@RepeatedTest(3)</span>
    <span class="hljs-meta">@DisplayName("重复测试")</span>
    <span class="hljs-keyword">void</span> <span class="hljs-title function_">repeatedTest
/span><span class="hljs-params">()</span> {
        System.out.println(<span class="hljs-string">"调用"</span>);
    }
</code></pre>
```

参数化测试

参数化测试可以按照多个参数分别运行多次单元测试这里有点类似于重复性测试，只不过每次运行传的参数不用。需要使用到@ParameterizedTest，同时也需要@ValueSource提供一组数据，它支持种基本类型以及String和自定义对象类型，使用极其方便。

```
<pre highlighted="true"><div class="esa-clipboard-button" data-clipboard-target="#copy_t
rget_7" title="复制代码">Copy</div><code class="language-java hljs" id="copy_target_7">
<span class="hljs-meta">@ParameterizedTest</span>
    <span class="hljs-meta">@ValueSource(ints = {1, 2, 3})</span>
    <span class="hljs-meta">@DisplayName("参数化测试")</span>
    <span class="hljs-keyword">void</span> <span class="hljs-title function_">paramTest</
pan><span class="hljs-params">(<span class="hljs-type">int</span> a)</span> {
        assertTrue(a > <span class="hljs-number">0</span> && a < <span class="hljs-number
>4</span>);
    }
</code></pre>
```

内嵌测试

原文链接：Junit5 介绍，Junit4 升级 Junit5 注意事项

JUnit5提供了嵌套单元测试的功能，可以更好展示测试类之间的业务逻辑关系，我们通常是一个业务应一个测试类，有业务关系的类其实可以写在一起。这样有利于进行测试。而且内联的写法可以大大少不必要的类，精简项目，防止类爆炸等一系列问题。

```
<pre highlighted="true"><div class="esa-clipboard-button" data-clipboard-target="#copy_t
rget_8" title="复制代码">Copy</div><code class="language-java hljs" id="copy_target_8"><
pan class="hljs-meta">@SpringBootTest</span>
<span class="hljs-meta">@AutoConfigureMockMvc</span>
<span class="hljs-meta">@DisplayName("Junit5单元测试")</span>
<span class="hljs-keyword">public</span> <span class="hljs-keyword">class</span> <spa
 class="hljs-title class_">MockTest</span> {
    <span class="hljs-comment">//....</span>
    <span class="hljs-meta">@Nested</span>
    <span class="hljs-meta">@DisplayName("内嵌订单测试")</span>
    <span class="hljs-keyword">class</span> <span class="hljs-title class_">OrderTestClas</
pan> {
        <span class="hljs-meta">@Test</span>
        <span class="hljs-meta">@DisplayName("取消订单")</span>
        <span class="hljs-keyword">void</span> <span class="hljs-title function_">cancelOrde
</span><span class="hljs-params">()</span> {
            <span class="hljs-type">int</span> <span class="hljs-variable">status</span> <spa
 class="hljs-operator">=</span> -<span class="hljs-number">1</span>;
            System.out.println(<span class="hljs-string">"取消订单成功,订单状态为:"</span>+statu
);
        }
    }
}

</code></pre>
```

Junit5/Junit4重要的区别

JUnit 5的测试看起来和JUnit 4的测试基本相同，但有几个不同之处你应该注意。

导入。 JUnit 5 使用新的 org.junit.jupiter 包。例如，org.junit.junit.Test变成了org.junit.jupiter.api.est。

注解。 @Test 注解不再有参数，每个参数都被移到了一个函数中。例如，下面是如何在JUnit 4中表预计一个测试会抛出异常的方法：

```
@Test(expected = Exception.class)
public void testThrowsException() throws Exception {
    // ...
}
```

在Junit5中，这个写法被改成了如下：

```
@Test
void testThrowsException() throws Exception {
    Assertions.assertThrows(Exception.class, () -> {
        //...
    });
}
```

原文链接：Junit5 介绍，Junit4 升级 Junit5 注意事项

同样，超时也发生了变化。下面是JUnit 4中的一个例子：

```
@Test(timeout = 10)
public void testFailWithTimeout() throws InterruptedException {
    Thread.sleep(100);
}
```

在JUnit5中，它被改成了如下：

```
@Test
void testFailWithTimeout() throws InterruptedException {
    Assertions.assertTimeout(Duration.ofMillis(10), () -> Thread.sleep(100));
}
```

以下是其他的注解的变化：

- @Before变成了@BeforeEach。
- @After变成了@AfterEach。
- @BeforeClass变成了@BeforeAll。
- @AfterClass变成了@AfterAll。
- @Ignore变成了@Disabled。
- @Category变成了@Tag。
- @Rule和@ClassRule没有了，用@ExtendWith和@RegisterExtension代替。

断言。 JUnit 5断言现在在org.junit.jupiter.api.Assertions中。大多数常见的断言，如assertEquals() assertNotNull()，看起来和以前一样，但有一些不同。

- 错误信息现在是最后一个参数，例如：assertEquals（“my message”，1，2)现在是assertEquals 1，2，” my message”）。
- 大多数断言现在接受一个构造错误信息的lambda，只有当断言失败时才会被调用。
- assertTimeout()和 assertTimeoutPreemptively()取代了 @Timeout 注释（在 JUnit 5 中有一个 Timeout 注释，但它的工作方式与 JUnit 4 中不同）。
- 有几个新的断言，如下文所述。

请注意，如果你愿意，你可以在JUnit 5测试中继续使用JUnit 4中的断言。

假设。 假设已被移至 org.junit.jupiter.api.Assumptions。

同样的假设也存在，但它们现在支持BooleanSupplier以及Hamcrest匹配器（**http://****hamcrest.rg/**）来匹配条件。Lambdas（类型为 Executable）可以用来在条件满足时执行代码。

例如，这里是JUnit 4中的一个例子：

```
@Test
public void testNothingInParticular() throws Exception {
    Assume.assumeThat("foo", is("bar"));
    assertEquals(...);
}
```

在Junit5中，它变成了这样：

```
@Test
void testNothingInParticular() throws Exception {
    Assumptions.assumingThat("foo".equals(" bar"), () -> {
        assertEquals(...);
    });
}
```

扩展JUnit

在JUnit 4中，自定义框架通常意味着使用@RunWith注释来指定一个自定义的运行器。使用多个运器是有问题的，通常需要链式或使用@Rule。在JUnit 5中，这一点已经得到了简化和改进。

例如，在JUnit 4中，使用Spring框架构建测试看起来是这样的：

```
@RunWith(SpringJUnit4ClassRunner.class)
public class MyControllerTest {
    // ...
}
```

在JUnit 5中，你可以用Spring扩展来代替：

```
@ExtendWith(SpringExtension.class)
class MyControllerTest {
    // ...
}
```

@ExtendWith 注解是可重复的，这意味着多个扩展可以很容易地组合在一起。

你也可以通过创建一个类来实现org.junit.jupiter.api.extendWith中的一个或多个接口，然后用@ExtndWith将其添加到你的测试中，从而轻松定义你自己的自定义扩展。

将Junit4转换到JUnit 5

要将现有的JUnit 4测试转换为JUnit 5，请使用以下步骤，这应该对大多数测试都有效。

1. 更新导入，删除JUnit 4并添加JUnit 5。例如，更新@Test注解的包名，更新断言的包名和类名（Asserts到Assertions）。如果有编译错误也不要担心，因为完成下面的步骤应该可以解决。

2. 在全局中用新的注解和类名替换旧的注解和类名。例如，将所有的@Before替换为@BeforeEach将所有的Asserts替换为Assertions。

3. 更新断言；任何提供消息的断言都需要将消息参数移到最后（当三个参数都是字符串时要特别注意）。另外，更新超时和预期异常（见上面的例子）。

4. 更新假设，如果你正在使用它们。

5. 用适当的 @ExtendWith 注释替换 @RunWith、@Rule 或 @ClassRule 的任何实例。你可能需要网上找到你所使用的扩展实例的更新文档。

注意，迁移参数化测试需要更多的重构，特别是如果你一直在使用JUnit 4参数化测试（JUnit 5参数测试的格式更接近于JUnitParams），那么迁移参数化测试需要更多的重构。

新功能

到目前为止，我只讨论了现有的功能以及它的变化。但JUnit 5提供了大量的新功能，让你的测试更具

描述性和可维护性。

显示名称。 使用JUnit 5，你可以在类和方法中添加@DisplayName注释。这个名称在生成报告时使，这使得描述测试的目的和追踪失败更容易，比如说：

```
@DisplayName("Test MyClass")
class MyClassTest {
    @Test
    @DisplayName("Verify MyClass.myMethod returns true")
    void testMyMethod() throws Exception {
        // ...
    }
}
```

你也可以使用显示名称生成器来处理你的测试类或方法，以你喜欢的任何格式生成测试名称。请参阅Junit文档中的具体内容和示例。

断言。 JUnit 5引入了一些新的断言，比如以下这些：

- assertIterableEquals()使用equals()对两个迭代项进行深度验证。
- assertLinesMatch()验证两个字符串列表是否匹配；它接受期望参数中的正则表达式。
- assertAll() 将多个断言分组在一起。附加的好处是所有的断言都会被执行，即使单个断言失败。
- assertThrows()和 assertDoesNotThrow()取代了 @Test 注释中的预期属性。

嵌套测试。 JUnit 4中的测试套件是很有用的，但JUnit 5中的嵌套测试更容易设置和维护，它们能更地描述测试组之间的关系，比如说：

```
@DisplayName("Verify MyClass")
class MyClassTest {
    MyClass underTest;

    @Test
    @DisplayName("can be instantiated")
    public void testConstructor() throws Exception {
        new MyClass();
    }
    @Nested
    @DisplayName("with initialization")
    class WithInitialization {
        @BeforeEach
        void setup() {
            underTest = new MyClass();
            underTest.init("foo");
        }

        @Test
        @DisplayName("myMethod returns true")
        void testMyMethod() {
            assertTrue(underTest.myMethod());
        }
    }
}
```

在上面的例子中，你可以看到，我对所有与MyClass相关的测试都使用一个类。我可以验证该类在外测试类中是可以实例化的，而我对所有MyClass被实例化和初始化的测试都使用嵌套的内部类。@BeforeEach方法只适用于嵌套类中的测试。

测试和类的@DisplayNames注解表示了测试的目的和组织方式。这有助于理解测试报告，因为你可看到测试是在什么条件下执行的（初始化后验证MyClass），以及测试要验证什么（myMethod返回true）。这是JUnit 5的一个很好的测试设计模式。

测试的参数化。 测试参数化在JUnit 4中就已经存在，有内置的库如JUnit4Parameterized或第三方如JUnitParams等。在JUnit 5中，参数化测试完全内置，并采用了JUnit4Parameterized和JUnitParams等一些最好的特性。例子：

```
@ParameterizedTest
@ValueSource(strings = {"foo", "bar"})
@NullAndEmptySource
void myParameterizedTest(String arg) {
    underTest.performAction(arg);
}
```

其格式看起来像JUnitParams，其中参数直接传递给测试方法。注意，要测试的值可以来自多个不同来源。这里，我只用了一个参数，所以使用@ValueSource很方便。@EmptySource和@NullSourc分别表示你要在要运行的值列表中添加一个空字符串和一个空值（如果你使用这两个值，你可以把它组合在一起，如上所示）。还有其他多个值源，比如@EnumSource和@ArgumentsSource（一种定义值提供者）。如果你需要一个以上的参数，也可以使用@MethodSource或@CsvSource。

在JUnit 5中添加的另一个测试类型是@RepeatedTest，在这里，一个测试被重复指定次数的测试。

测试执行条件。 JUnit 5 提供了 ExecutionCondition 扩展 API 来有条件地启用或禁用一个测试或容（测试类）。这就像在测试上使用@Disabled一样，但它可以定义自定义条件。有多个内置的条件，如说这些条件：

- @EnabledOnOs和@DisabledOnOs。仅在指定的操作系统上启用或禁用测试。
- @EnabledOnJre和@DisabledOnJre。指定特定版本的Java测试应该启用或禁用。
- @EnabledIfSystemProperty: 启用基于JVM系统属性值的测试。
- @EnabledIf: 使用脚本逻辑，在满足脚本条件的情况下，使用脚本逻辑启用测试。

测试模板。 测试模板不是常规测试；它们定义了一组要执行的步骤，然后可以在其他地方使用特定的用上下文执行。这意味着你可以定义一次测试模板，然后在运行时建立一个调用上下文列表来运行该试。有关详细信息和示例，请参阅文档。

动态测试。 动态测试就像测试模板，要运行的测试是在运行时生成的。然而，测试模板是用一组特定步骤来定义并运行多次，而动态测试使用相同的调用上下文，但可以执行不同的逻辑。动态测试的一用途是将一个抽象对象的列表流化，并根据它们的具体类型为每个对象执行一组单独的断言。文档中一些很好的例子。

结论

尽管你可能不需要将旧的JUnit 4测试转换为JUnit 5，除非你想使用新的JUnit 5功能，但也有令人信的理由让你切换到JUnit 5。例如，JUnit 5的测试功能更强大，更容易维护。此外，JUnit 5提供了许有用的新特性，只有你使用的特性才会被导入，你可以使用多个扩展，甚至可以创建自己的自定义扩。这些变化和新功能一起，为JUnit框架提供了一个强大而灵活的更新。