Migrando do JUnit 4 para o JUnit 5

https://www.baeldung.com/junit-5-migration

1. Visão Geral

Neste artigo, veremos como podemos migrar do JUnit 4 para a versão mais recente do JUnit 5 - com uma visão geral das diferenças entre as duas versões da biblioteca.

Para obter as diretrizes gerais sobre o uso do JUnit 5, consulte nosso artigo aqui .

2. Vantagens do JUnit 5

Vamos começar com a versão anterior - JUnit 4 tem algumas limitações claras:

- Toda a estrutura estava contida em uma única biblioteca jar. Toda a biblioteca precisa ser importada, mesmo quando apenas um determinado recurso é necessário. **Na JUnit 5, obtemos mais granularidade e podemos importar apenas o necessário**
- Um executor de teste só pode executar testes no JUnit 4 por vez (por exemplo, *SpringJUnit4ClassRunner* ou *Parametrizado*). **JUnit 5 permite que vários corredores trabalhem simultaneamente**
- O JUnit 4 nunca avançou além do Java 7, perdendo muitos recursos do Java 8. O **JUnit 5 faz bom uso dos recursos do Java 8**

A ideia por trás do JUnit 5 era reescrever completamente o JUnit 4 para resolver a maioria dessas desvantagens.

3. Diferenças

A JUnit 4 foi dividida em módulos que abrangem a JUnit 5:

- **Plataforma JUnit -** este módulo abrange todas as estruturas de extensão em que podemos estar interessados na execução de teste, descoberta e relatório
- **JUnit Vintage -** este módulo permite compatibilidade retroativa com JUnit 4 ou mesmo JUnit 3

3.1. Anotações

JUnit 5 vem com mudanças importantes em suas anotações. **O mais importante é que não podemos mais usar a anotação *@Test* para especificar expectativas.**

O parâmetro *esperado* no JUnit 4:

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

Agora, podemos usar um método *assertThrows* :

```
public void shouldRaiseAnException() throws Exception
{
    Assertions.assertThrows(Exception.class, () ->
    {
        //...
    });
}
```

O atributo de *tempo limite* no JUnit 4:

```
@Test(timeout = 1)
public void shouldFailBecauseTimeout() throws InterruptedException
{
       Thread.sleep(10);
}
```

Agora, o método *assertTimeout* na JUnit 5:

```
@Test
public void shouldFailBecauseTimeout() throws InterruptedException
{
       Assertions.assertTimeout(Duration.ofMillis(1), () -> Thread.sleep(10));
}
```

Outras anotações que foram alteradas na JUnit 5:

- *A* anotação *@Before* foi renomeada para *@BeforeEach*
- *@After* annotation é renomeado para *@AfterEach*
- *A* anotação *@BeforeClass* foi renomeada para *@BeforeAll*
- *A* anotação *@AfterClass* foi renomeada para *@AfterAll*
- *A* anotação *@Ignore* foi renomeada para *@Disabled*

3.2. Afirmações

Agora podemos escrever mensagens de asserção em um lambda no JUnit 5, permitindo que a avaliação preguiçosa ignore a construção de mensagens complexas até que seja necessário:

```
@Test
public void shouldFailBecauseTheNumbersAreNotEqual_lazyEvaluation()
{
   Assertions.assertTrue(2 == 3, () -> "Numbers " + 2 + " and " + 3 + " are not equal!");
}
```

Também podemos agrupar afirmações na JUnit 5:

```
@Test
public void shouldAssertAllTheGroup()
{
       List<Integer> list = Arrays.asList(1, 2, 4);
       Assertions.assertAll("List is not incremental", () -> Assertions.assertEquals(list.get(0).intValue(), 1),
                    () -> Assertions.assertEquals(list.get(1).intValue(), 2),
                    () -> Assertions.assertEquals(list.get(2).intValue(), 3));
}
```

3.3. Premissas

A nova classe *Assumptions* está agora em *org.junit.jupiter.api.Assumptions* . A JUnit 5 oferece suporte total aos métodos de suposições existentes na JUnit 4 e também adiciona um conjunto de novos métodos para permitir a execução de algumas afirmações apenas em cenários específicos:

```
@Test
public void whenEnvironmentIsWeb_thenUrlsShouldStartWithHttp()
{
	assumingThat("WEB".equals(System.getenv("ENV")), () ->
	{
		assertTrue("http".startsWith(address));
	});
}
```

3.4. Marcação e filtragem

No JUnit 4, podemos agrupar os testes usando a anotação *@Category* . Com o JUnit 5, a anotação *@Category* é substituída pela anotação *@Tag* :

```
@Tag("annotations")
@Tag("junit5")
@RunWith(JUnitPlatform.class)
public class AnnotationTestExampleTest
{
	/* ... */
}
```

Podemos incluir / excluir tags específicas usando o *plug-in maven-surefire* :

```
<build>
    <plugins>
        <plugin>
            <artifactId>maven-surefire-plugin</artifactId>
            <configuration>
                <properties>
                    <includeTags>junit5</includeTags>
                </properties>
            </configuration>
        </plugin>
    </plugins>
</build>
```

3,5. Novas anotações para testes em execução

O *@RunWith* foi usado para integrar o contexto de teste com outras estruturas ou para alterar o fluxo de execução geral nos casos de teste no JUnit 4.

Com o JUnit 5, agora podemos usar a anotação *@ExtendWith* para fornecer funcionalidade semelhante.

Por exemplo, para usar os recursos do Spring na JUnit 4:

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration({ "/app-config.xml", "/test-data-access-config.xml" })
public class SpringExtensionTest
{
	/* ... */
}
```

Agora, no JUnit 5, é uma extensão simples:

```
@ExtendWith(SpringExtension.class)
@ContextConfiguration({ "/app-config.xml", "/test-data-access-config.xml" })
public class SpringExtensionTest
{
	/* ... */
}
```

3,6. Novas anotações de regras de teste

Na JUnit 4, as *anotações @Rule* e *@ ClassRule* foram usadas para adicionar funcionalidade especial aos testes.

No JUnit 5. podemos reproduzir a mesma lógica usando a anotação *@ExtendWith* .

Por exemplo, digamos que temos uma regra personalizada no JUnit 4 para gravar rastreamentos de log antes e depois de um teste:

```
public class TraceUnitTestRule implements TestRule
{

	@Override
    public Statement apply(Statement base, Description description) {
        return new Statement() {
            @Override
            public void evaluate() throws Throwable {
                // Before and after an evaluation tracing here
                ...
            }
        };
    }
}
```

E nós o implementamos em um conjunto de testes:

```
@Rule
public TraceUnitTestRule traceRuleTests = new TraceUnitTestRule();
```

No JUnit 5, podemos escrever o mesmo de uma maneira muito mais intuitiva:

```
public class TraceUnitExtension implements AfterEachCallback, BeforeEachCallback
{

	@Override
	public void beforeEach(TestExtensionContext context) throws Exception
	{
		// ...
	}

	@Override
	public void afterEach(TestExtensionContext context) throws Exception
	{
		// ...
	}
}
```

Usando as interfaces *AfterEachCallback* e *BeforeEachCallback* do JUnit 5 disponíveis no pacote *org.junit.jupiter.api.extension*, implementamos facilmente esta regra no conjunto de testes:

```
@RunWith(JUnitPlatform.class)
@ExtendWith(TraceUnitExtension.class)
public class RuleExampleTest
{

	@Test
	public void whenTracingTests()
	{
		/* ... */
	}
}
```

3,7. JUnit 5 Vintage

O JUnit Vintage auxilia na migração de testes JUnit executando os testes JUnit 3 ou JUnit 4 dentro do contexto JUnit 5.

Podemos usá-lo importando o JUnit Vintage Engine:

```
<dependency>
	<groupId>org.junit.vintage</groupId>
	<artifactId>junit-vintage-engine</artifactId>
	<version>${junit5.vintage.version}</version>
	<scope>test</scope>
</dependency>
```

4. Conclusão

Como vimos neste artigo, JUnit 5 é uma abordagem modular e moderna da estrutura JUnit 4. Apresentamos as principais diferenças entre essas duas versões e sugerimos como migrar de uma para a outra.

A implementação completa deste tutorial pode ser encontrada no GitHub .