**Правила и Запускалки**

Правила

Кроме всего вышеперечисленного есть довольно интересная вещь — правила. Правила это некое подобие утилит для тестов, которые добавляют функционал до и после выполнения теста.

В Junit 5 @Rule и @ClassRule больше не существует; их заменено @ExtendWith и @RegisterExtension

Например, есть встроенные правила для задания таймаута для теста(*Timeout*), для задания ожидаемых исключений(*ExpectedException*), для работы с временными файлами(*TemporaryFolder*) и д.р. Для объявления правила необходимо создать *public* не *static* поле типа производного от *MethodRule* и зааннотировать его с помощью [*Rule*](https://habrahabr.ru/users/rule/).  
public class OtherJUnit4Test {

@Rule

public final TemporaryFolder folder = new TemporaryFolder();

@Rule

public final Timeout timeout = new Timeout(1000);

@Rule

public final ExpectedException thrown = ExpectedException.none();

@Ignore

@Test

public void anotherInfinity() {

while (true);

}

@Test

public void testFileWriting() throws IOException {

final File log = folder.newFile("debug.log");

final FileWriter logWriter = new FileWriter(log);

logWriter.append("Hello, ");

logWriter.append("World!!!");

logWriter.flush();

logWriter.close();

}

@Test

public void testExpectedException() throws IOException {

thrown.expect(NullPointerException.class);

StringUtils.toHexString(null);

}

}

Также в сети можно найти и другие варианты использования. Например, [здесь](http://blog.mycila.com/2009/11/writing-your-own-junit-extensions-using.html) рассмотрена возможность параллельного запуска теста.

Запускалки

Но и на этом возможности фреймворка не заканчиваются. То, как запускается тест, тоже может быть сконфигурировано с помощью *@RunWith*.

В Junit 5 @RunWith более не существует; заменено на @ExtendWith.

*JUnit4* — запускалка по умолчанию, как понятно из названия, предназначена для запуска JUnit 4 тестов.  
  
*JUnit38ClassRunner* предназначен для запуска тестов, написанных с использованием JUnit 3.  
  
*SuiteMethod*  либо  *AllTests*  тоже предназначены для запуска JUnit 3 тестов. В отличие от предыдущей запускалки, в эту передается класс со статическим методом suite возвращающим тест(последовательность всех тестов).  
  
*Suite* — эквивалент предыдущего, только для JUnit 4 тестов. Для настройки запускаемых тестов используется аннотация *@SuiteClasses*.

@Suite.SuiteClasses( { OtherJUnit4Test.class, StringUtilsJUnit4Test.class })

@RunWith(Suite.class)

public class JUnit4TestSuite {

}

*Enclosed* — то же, что и предыдущий вариант, но вместо настройки с помощью аннотации используются все внутренние классы.  
  
*Categories* — попытка организовать тесты в категории(группы). Для этого тестам задается категория с помощью *@Category*, затем настраиваются запускаемые категории тестов в сюите. Это может выглядеть так:  
public class StringUtilsJUnit4CategoriesTest extends Assert {

//...

@Category(Unit.class)

@Test

public void testIsEmpty() {

//...

}

//...

}

@RunWith(Categories.class)

@Categories.IncludeCategory(Unit.class)

@Suite.SuiteClasses( { OtherJUnit4Test.class, StringUtilsJUnit4CategoriesTest.class })

public class JUnit4TestSuite {

}

*Parameterized* — довольно интересная запускалка, позволяет писать параметризированные тесты. Для этого в тест-классе объявляется статический метод возвращающий список данных, которые затем будут использованы в качестве аргументов конструктора класса.

@RunWith(Parameterized.class)

public class StringUtilsJUnit4ParameterizedTest extends Assert {

private final CharSequence testData;

private final boolean expected;

public StringUtilsJUnit4ParameterizedTest(final CharSequence testData, final boolean expected) {

this.testData = testData;

this.expected = expected;

}

@Test

public void testIsEmpty() {

final boolean actual = StringUtils.isEmpty(testData);

assertEquals(expected, actual);

}

@Parameterized.Parameters

public static List<Object[]> isEmptyData() {

return Arrays.asList(new Object[][] {

{ null, true },

{ "", true },

{ " ", false },

{ "some string", false },

});

}

}

*Theories* — чем-то схожа с предыдущей, но параметризирует тестовый метод, а не конструктор. Данные помечаются с помощью *@DataPoints* и *@DataPoint*, тестовый метод — с помощью *[Theory](https://habrahabr.ru/users/theory/)*. Тест использующий этот функционал будет выглядеть примерно так:

@RunWith(Theories.class)

public class StringUtilsJUnit4TheoryTest extends Assert {

@DataPoints

public static Object[][] isEmptyData = new Object[][] {

{ "", true },

{ " ", false },

{ "some string", false },

};

@DataPoint

public static Object[] nullData = new Object[] { null, true };

@Theory

public void testEmpty(final Object... testData) {

final boolean actual = StringUtils.isEmpty((CharSequence) testData[0]);

assertEquals(testData[1], actual);

}

}

Как и в случае с правилами, в сети можно найти и другие варианты использования. Например, [здесь](http://blog.mycila.com/2009/11/writing-your-own-junit-extensions-using.html) рассмотрена та же возможность паралельного запуска теста, но с использованием запускалок.