**Cucumber все дополнительные функции**

**https://github.com/cucumber/cucumber-jvm/tree/main/cucumber-junit-platform-engine**

**Cucumber JUnit Platform Engine**

Use the JUnit (5) Platform to execute Cucumber scenarios.

Add the cucumber-junit-platform-engine dependency to your pom.xml:

<dependency>

<groupId>io.cucumber</groupId>

<artifactId>cucumber-junit-platform-engine</artifactId>

<version>${cucumber.version}</version>

<scope>test</scope>

</dependency>

This will allow IntelliJ IDEA, Eclipse, Maven, Gradle, etc, to discover, select and execute Cucumber scenarios.

**Surefire and Gradle workarounds**

Maven, Surefire and Gradle do not yet support discovery of non-class based tests (see: [gradle/#4773](https://github.com/gradle/gradle/issues/4773), [SUREFIRE-1724](https://issues.apache.org/jira/browse/SUREFIRE-1724)). As a workaround, you can either use:

* the [JUnit Platform Suite Engine](https://junit.org/junit5/docs/current/user-guide/" \l "junit-platform-suite-engine);
* the [JUnit Platform Console Launcher](https://junit.org/junit5/docs/current/user-guide/" \l "running-tests-console-launcher) or;
* the [Gradle Cucumber-Companion](https://github.com/gradle/cucumber-companion) plugins for Gradle and Maven.
* the [Cucable](https://github.com/trivago/cucable-plugin) plugin for Maven.

**Используйте JUnit Platform Suite Engine**

Движок JUnit Platform Suite можно использовать для запуска Cucumber. Краткое описание того, как это сделать, см. в разделе Пакеты с различными конфигурациями.

Поскольку отчеты Surefire и Gradle предоставляют результаты в формате <Имя класса> — <Имя метода>, сообщаются только имена сценариев или номера примеров. Это может затруднить чтение отчетов.

Чтобы улучшить читаемость отчетов, укажите параметр конфигурации:

cucmber.junit-platform.naming-strategy=long.

это будет включать имя функции как часть имени теста.

**Maven**

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-plugin</artifactId>

<version>3.0.0-M5</version>

<configuration>

<properties>

<configurationParameters>

cucumber.junit-platform.naming-strategy=long

</configurationParameters>

</properties>

</configuration>

</plugin>

**Gradle**

tasks.test {

useJUnitPlatform()

systemProperty("cucumber.junit-platform.naming-strategy", "long")

}

**Use the JUnit Console Launcher**

You can integrate the JUnit Platform Console Launcher in your build by using either the Maven Antrun plugin or the Gradle JavaExec task.

**Use the Maven Antrun plugin**

Add the following to your pom.xml:

<dependencies>

....

<dependency>

<groupId>org.junit.platform</groupId>

<artifactId>junit-platform-console</artifactId>

<version>${junit-platform.version}</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-antrun-plugin</artifactId>

<executions>

<execution>

<!--Work around. Surefire does not use JUnit's Test Engine discovery functionality -->

<id>CLI-test</id>

<phase>integration-test</phase>

<goals>

<goal>run</goal>

</goals>

<configuration>

<target>

<echo message="Running JUnit Platform CLI"/>

<java classname="org.junit.platform.console.ConsoleLauncher"

fork="true"

failonerror="true"

newenvironment="true"

maxmemory="512m"

classpathref="maven.test.classpath">

<arg value="--include-engine"/>

<arg value="cucumber"/>

<arg value="--scan-classpath"/>

<arg value="${project.build.testOutputDirectory}"/>

</java>

</target>

</configuration>

</execution>

</executions>

</plugin>

</plugins>

</build>

**Use the Gradle JavaExec task**

Add the following to your build.gradle.kts:

tasks {

val consoleLauncherTest by registering(JavaExec::class) {

dependsOn(testClasses)

val reportsDir = file("$buildDir/test-results")

outputs.dir(reportsDir)

classpath = sourceSets["test"].runtimeClasspath

main = "org.junit.platform.console.ConsoleLauncher"

args("--scan-classpath")

args("--include-engine", "cucumber")

args("--reports-dir", reportsDir)

}

test {

dependsOn(consoleLauncherTest)

exclude("\*\*/\*")

}

}

**Запуск одного сценария или функции из CLI**

To select a single scenario or feature the cucumber.features property can be used. Because this property will cause Cucumber to ignore any other selectors from JUnit, it is prudent to execute only the Cucumber engine.

**Maven**

To select the scenario on line 10 of the example.feature file use:

mvn test -Dsurefire.includeJUnit5Engines=cucumber -Dcucumber.plugin=pretty -Dcucumber.features=path/to/example.feature:10

**Gradle**

TODO: (Feel free to send a pull request. ;))

**Suites с разными конфигурациями**

The JUnit Platform Suite Engine can be used to run Cucumber multiple times with different configurations. Add the junit-platform-suite dependency:

<dependency>

<groupId>org.junit.platform</groupId>

<artifactId>junit-platform-suite</artifactId>

<version>${junit-platform.version}</version>

<scope>test</scope>

</dependency>

Then define suites as needed using the annotation from the [org.junit.platform.suite.api](https://junit.org/junit5/docs/current/api/org.junit.platform.suite.api/org/junit/platform/suite/api/package-summary.html) package:

package com.example;

import org.junit.platform.suite.api.ConfigurationParameter;

import org.junit.platform.suite.api.IncludeEngines;

import org.junit.platform.suite.api.SelectClasspathResource;

import org.junit.platform.suite.api.Suite;

import static io.cucumber.junit.platform.engine.Constants.GLUE\_PROPERTY\_NAME;

@Suite

@IncludeEngines("cucumber")

@SelectClasspathResource("com/example")

@ConfigurationParameter(key = GLUE\_PROPERTY\_NAME, value = "com.example")

public class RunCucumberTest {

}

**Многопоточка**

По умолчанию Cucumber запускает тесты последовательно в одном потоке. Параллельное выполнение тестов доступно в качестве дополнительной функции. Чтобы включить параллельное выполнение, установите для параметра конфигурации cucumber.execution.parallel.enabled значение true, например, в файле junit-platform.properties.

Для управления такими свойствами, как желаемый и максимальный параллелизм, Cucumber поддерживает JUnit 5s ParallelExecutionConfigurationStrategy. Cucumber предоставляет две реализации: динамическую и фиксированную, которые можно настроить с помощью cucumber.execution.parallel.config.strategy. Вы также можете реализовать собственную стратегию.

динамический: автоматически вычисляет желаемый параллелизм как <доступные ядра> умноженные на установленный нами параметр cucumber.execution.parallel.config.dynamic.factor=2(если у ядер компа по 2 потока).

фиксированный(fixed): установите для cucumber.execution.parallel.config.fixed.parallelism желаемый параллелизм, а для cucumber.execution.parallel.config.fixed.max-pool-size — максимальный размер базового пула ForkJoin.

custom: укажите пользовательскую реализацию ParallelExecutionConfigurationStrategy через cucumber.execution.parallel.config.custom.class.

Если стратегия не указана, Cucumber будет использовать динамическую стратегию с коэффициентом 1.

Примечание. Хотя .fixed.max-pool-size эффективно ограничивает максимальное количество одновременных потоков, Cucumber не гарантирует, что количество одновременно выполняемых сценариев не превысит это значение. Подробности смотрите в junit5/#3108.

**Синхронизация доступа к одному ресурсу**

To avoid flaky tests when multiple scenarios manipulate the same resource, tests can be [synchronized](https://junit.org/junit5/docs/current/user-guide/#writing-tests-parallel-execution-synchronization) on that resource.

To synchronize a scenario on a specific resource, the scenario must be tagged and this tag mapped to a lock for the specific resource. A resource is identified by an arbitrary string and can be either locked with a read-write-lock, or a read-lock.

For example, the following tags:

Feature: Exclusive resources

@reads-and-writes-system-properties

Scenario: first example

Given this reads and writes system properties

When it is executed

Then it will not be executed concurrently with the second example

@reads-system-properties

Scenario: second example

Given this reads system properties

When it is executed

Then it will not be executed concurrently with the first example

with this configuration:

cucumber.execution.exclusive-resources.reads-and-writes-system-properties.read-write=java.lang.System.properties

cucumber.execution.exclusive-resources.reads-system-properties.read=java.lang.System.properties

when executing the first scenario tagged with @reads-and-writes-system-properties will lock the java.lang.System.properties resource with a read-write lock and will not be concurrently executed with the second scenario that locks the same resource with a read lock.

Note: The @ from the tag is not included in the property name. Note: For canonical resource names see [junit5/Resources.java](https://github.com/junit-team/junit5/blob/main/junit-jupiter-api/src/main/java/org/junit/jupiter/api/parallel/Resources.java)

**Запуск изолированного теста**

To ensure that a scenario runs while no other scenarios are running the global resource [org.junit.platform.engine.support.hierarchical.ExclusiveResource.GLOBAL\_KEY](https://github.com/junit-team/junit5/blob/main/junit-platform-engine/src/main/java/org/junit/platform/engine/support/hierarchical/ExclusiveResource.java#L47) can be used.

Feature: Isolated scenarios

@isolated

Scenario: isolated example

Given this scenario runs isolated

When it is executed

Then it will not be executed concurrently with the second or third example

Scenario: second example

When it is executed

Then it will not be executed concurrently with the isolated example

And it will be executed concurrently with the third example

Scenario: third example

When it is executed

Then it will not be executed concurrently with the isolated example

And it will be executed concurrently with the second example

with this configuration:

cucumber.execution.exclusive-resources.isolated.read-write=org.junit.platform.engine.support.hierarchical.ExclusiveResource.GLOBAL\_KEY

**Executing features in parallel**

By default, when parallel execution is enabled, scenarios and examples are executed in parallel. Due to limitations, JUnit 4 could only execute features in parallel. This behaviour can be restored by setting the configuration parameter cucumber.execution.execution-mode.feature to same\_thread.

**Configuration Options**

Cucumber receives its configuration from the JUnit Platform. To see how these can be supplied; see the JUnit documentation [4.5. Configuration Parameters](https://junit.org/junit5/docs/current/user-guide/#running-tests-config-params). For documentation on Cucumber properties, see [Constants](https://github.com/cucumber/cucumber-jvm/blob/main/cucumber-junit-platform-engine/src/main/java/io/cucumber/junit/platform/engine/Constants.java).

cucumber.ansi-colors.disabled= # true or false.

# default: false

cucumber.filter.name= # a regular expression.

# only scenarios with matching names are executed.

# example: ^Hello (World|Cucumber)$

# note: To ensure consistent reports between Cucumber and

# JUnit 5 prefer using JUnit 5s discovery request filters

# or JUnit 5 tag expressions instead.

cucumber.features= # comma separated paths to feature files.

# example: path/to/example.feature, path/to/other.feature

# note: When used any discovery selectors from the JUnit

# Platform will be ignored. This may lead to multiple

# executions of Cucumber. For example when used in

# combination with the JUnit Platform Suite Engine.

# When using Cucumber through the JUnit Platform

# Launcher API or the JUnit Platform Suite Engine, it is

# recommended to respectively use JUnit's

# DiscoverySelectors or equivalent annotations.

cucumber.filter.tags= # a cucumber tag expression.

# only scenarios with matching tags are executed.

# example: @Cucumber and not (@Gherkin or @Zucchini)

# note: To ensure consistent reports between Cucumber and

# JUnit 5 prefer using JUnit 5s discovery request filters

# or JUnit 5 tag expressions instead.

cucumber.glue= # comma separated package names.

# example: com.example.glue

cucumber.junit-platform.naming-strategy= # long or short.

# default: short

# include parent descriptor name in test descriptor.

cucumber.plugin= # comma separated plugin strings.

# example: pretty, json:path/to/report.json

cucumber.object-factory= # object factory class name.

# example: com.example.MyObjectFactory

cucumber.publish.enabled # true or false.

# default: false

# enable publishing of test results

cucumber.publish.quiet # true or false.

# default: false

# suppress publish banner after test execution.

cucumber.publish.token # any string value.

# publish authenticated test results.

cucumber.snippet-type= # underscore or camelcase.

# default: underscore

cucumber.execution.dry-run= # true or false.

# default: false

cucumber.execution.execution-mode.feature= # same\_thread or concurrent

# default: concurrent

# same\_thread - executes scenarios sequentially in the

# same thread as the parent feature

# concurrent - executes scenarios concurrently on any

# available thread

cucumber.execution.parallel.enabled= # true or false.

# default: false

cucumber.execution.parallel.config.strategy= # dynamic, fixed or custom.

# default: dynamic

cucumber.execution.parallel.config.fixed.parallelism= # positive integer.

# example: 4

cucumber.execution.parallel.config.fixed.max-pool-size= # positive integer.

# example: 4

cucumber.execution.parallel.config.dynamic.factor= # positive double.

# default: 1.0

cucumber.execution.parallel.config.custom.class= # class name.

# example: com.example.MyCustomParallelStrategy

cucumber.execution.exclusive-resources.<tag-name>.read-write= # a comma separated list of strings

# example: resource-a, resource-b.

cucumber.execution.exclusive-resources.<tag-name>.read= # a comma separated list of strings

# example: resource-a, resource-b

**Поддерживаемые селекторы и фильтры**

JUnit 5 [introduced a test discovery mechanism](https://junit.org/junit5/docs/current/user-guide/#launcher-api-discovery) as a dedicated feature of the platform itself. This allows IDEs and build tools to identify tests. Supported DiscoverySelectors are:

* ClasspathRootSelector
* ClasspathResourceSelector
* ClassSelector
* PackageSelector
* FileSelector
* DirectorySelector
* UriSelector
* UniqueIdSelector

The only supported DiscoveryFilter is the PackageNameFilter and only when features are selected from the classpath.

**Выбор индивидуальных сценариев, правил и примеров**

The FileSelector and ClasspathResourceSelector support a FilePosition.

* DiscoverySelectors.selectClasspathResource("rule.feature", FilePosition.from(5))
* DiscoverySelectors.selectFile("rule.feature", FilePosition.from(5))

The UriSelector supports URI's with a line query parameter:

* classpath:/com/example/example.feature?line=20
* file:/path/to/com/example/example.feature?line=20

Any TestDescriptor that matches the line *and* its descendants will be included in the discovery result. For example, selecting a Rule will execute all scenarios contained within the Rule.

**Использование Тегов для группировки**

Cucumber tags are mapped to JUnit tags. Note that the @ symbol is not part of the JUnit tag. So the scenarios below are tagged with Smoke and Sanity.

@Smoke

@Ignore

Scenario: A tagged scenario

Given I tag a scenario

When I select tests with that tag for execution

Then my tagged scenario is executed

@Sanity

Scenario: Another tagged scenario

Given I tag a scenario

When I select tests with that tag for execution

Then my tagged scenario is executed

When using Maven, tags can be provided from the CLI using the groups and excludedGroups parameters. These take a [JUnit5 Tag Expression](https://junit.org/junit5/docs/current/user-guide/#running-tests-tag-expressions). The example below will execute Another tagged scenario.

mvn verify -DexcludedGroups="Ignore" -Dgroups="Smoke | Sanity"

For more information on how to select tags, see the relevant documentation:

* [JUnit 5 Suite: @Include Tags](https://junit.org/junit5/docs/current/api/org.junit.platform.suite.api/org/junit/platform/suite/api/IncludeTags.html)
* [JUnit 5 Suite: @Exclude Tags](https://junit.org/junit5/docs/current/api/org.junit.platform.suite.api/org/junit/platform/suite/api/ExcludeTags.html)
* [JUnit 5 Console Launcher: Options](https://junit.org/junit5/docs/current/user-guide/#running-tests-console-launcher-options)
* [JUnit 5 Tag Expression](https://junit.org/junit5/docs/current/user-guide/#running-tests-tag-expressions)
* [Maven: Filtering by Tags](https://maven.apache.org/surefire/maven-surefire-plugin/examples/junit-platform.html)
* [Gradle: Test Grouping](https://docs.gradle.org/current/userguide/java_testing.html#test_grouping)

**@Disabled**

Функциональность @Disabled JUnit Jupiter можно воссоздать, установив свойство огурца.filter.tags=not @Disabled1. Любые сценарии, отмеченные тегом @Disabled, будут пропущены. Дополнительные сведения см. в разделе [Configuration Options](https://github.com/cucumber/cucumber-jvm/tree/main/cucumber-junit-platform-engine#configuration-options). Обратите внимание, что это выражение [Cucumber Tag Expression](https://cucumber.io/docs/cucumber/api/#tags), а не выражение тега JUnit5.

**Прерывание тестов вместо падения**

Cucumber supports [OpenTest4Js](https://github.com/ota4j-team/opentest4j) TestAbortedException. This makes it possible to use JUnit Jupiter's Assumptions to abort rather than fail a scenario.

package com.example;

import io.cucumber.java.Before;

import org.junit.jupiter.api.Assumptions;

import java.util.List;

public class RpnCalculatorSteps {

@Before

public void before() {

boolean condition = // decide if tests should abort

Assumptions.assumeTrue(condition, "Condition not met");

}

}

**Многократный перезапуск упавших тестов**

When using cucumber-junit-platform-engine rerun files are not supported. However, the JUnit Platform allows you to rerun failed tests through its API.

package com.example;

import org.junit.platform.engine.discovery.DiscoverySelectors;

import org.junit.platform.engine.discovery.UniqueIdSelector;

import org.junit.platform.launcher.Launcher;

import org.junit.platform.launcher.LauncherDiscoveryRequest;

import org.junit.platform.launcher.TestIdentifier;

import org.junit.platform.launcher.core.LauncherFactory;

import org.junit.platform.launcher.listeners.SummaryGeneratingListener;

import org.junit.platform.launcher.listeners.TestExecutionSummary;

import org.junit.platform.launcher.listeners.TestExecutionSummary.Failure;

import java.util.List;

import java.util.stream.Collectors;

import static org.junit.platform.engine.discovery.DiscoverySelectors.selectDirectory;

import static org.junit.platform.launcher.core.LauncherDiscoveryRequestBuilder.request;

public class RunCucumber {

public static void main(String[] args) {

LauncherDiscoveryRequest request = request()

.selectors(

selectDirectory("path/to/features")

)

.build();

Launcher launcher = LauncherFactory.create();

SummaryGeneratingListener listener = new SummaryGeneratingListener();

launcher.registerTestExecutionListeners(listener);

launcher.execute(request);

TestExecutionSummary summary = listener.getSummary();

// Do something with summary

List<UniqueIdSelector> failures = summary.getFailures().stream()

.map(Failure::getTestIdentifier)

.filter(TestIdentifier::isTest)

.map(TestIdentifier::getUniqueId)

.map(DiscoverySelectors::selectUniqueId)

.collect(Collectors.toList());

LauncherDiscoveryRequest rerunRequest = request()

.selectors(failures)

.build();

launcher.execute(rerunRequest);

TestExecutionSummary rerunSummary = listener.getSummary();

// Do something with rerunSummary

}

}