**Параллельное выполнение тестов Junit5**

[**https://junit.org/junit5/docs/snapshot/user-guide/#writing-tests-parallel-execution**](https://junit.org/junit5/docs/snapshot/user-guide/#writing-tests-parallel-execution)

**п.2.20**

По умолчанию тесты JUnit Jupiter выполняются последовательно в одном потоке. Параллельное выполнение тестов — например, для ускорения выполнения — доступно в качестве дополнительной функции начиная с версии 5.3. Чтобы включить параллельное выполнение, установите для параметра конфигурации ***junit.jupiter.execution.parallel.enabled*** значение ***true*** — например, в *junit-platform.properties* (другие параметры см. в разделе «[Configuration Parameters](https://junit.org/junit5/docs/snapshot/user-guide/" \l "running-tests-config-params)»).

Обратите внимание, что включение этого свойства — это только первый шаг, необходимый для параллельного выполнения тестов. Если этот параметр включен, тестовые классы и методы по-прежнему будут выполняться последовательно по умолчанию. Будет ли узел в дереве тестов выполняться одновременно, зависит от его режима выполнения. Доступны следующие два режима.

SAME\_THREAD

Принудительное выполнение в том же потоке, который используется родительским элементом. Например, при использовании в тестовом методе он будет выполняться в том же потоке, что и любые методы @BeforeAll или @AfterAll.

CONCURRENT

Выполнять одновременно, если блокировка ресурса не приводит к принудительному выполнению в том же потоке.

По умолчанию узлы в дереве тестов используют режим выполнения SAME\_THREAD. Для изменения этого поведения вы можете:

1. Изменить значение по умолчанию, установив параметр конфигурации ***junit.jupiter.execution.parallel.mode.default***.
2. Альтернативно вы можете использовать аннотацию ***@Execution(CONCURRENT)*** для изменения режима выполнения аннотированного элемента и его подэлементов (если таковые имеются), что позволяет активировать параллельное выполнение для отдельных классов тестов, один за другим.

Параметры конфигурации для параллельного выполнения всех тестов

junit.jupiter.execution.parallel.enabled = true junit.jupiter.execution.parallel.mode.default = concurrent

Режим выполнения по умолчанию применяется ко всем узлам дерева тестов за несколькими заметными исключениями, а именно к классам тестов, использующим режим ***Lifecycle.PER\_CLASS*** в аннотации **@TestInstance** (*позволяющий нам попросить JUnit создать только один экземпляр тестового класса для всех его методов*  *и повторно использовать его между тестами*, *а не как делается по умолчанию создавать для каждого метода каждого класса отдельный объект класса*) или [MethodOrderer](https://junit.org/junit5/docs/snapshot/api/org.junit.jupiter.api/org/junit/jupiter/api/MethodOrderer.html) (за исключением [MethodOrderer.Random](https://junit.org/junit5/docs/snapshot/api/org.junit.jupiter.api/org/junit/jupiter/api/MethodOrderer.Random.html)). В первом случае авторы тестов должны гарантировать, что тестовый класс является потокобезопасным; во втором случае одновременное выполнение может конфликтовать с настроенным порядком выполнения. Таким образом, в обоих случаях тестовые методы в таких тестовых классах выполняются одновременно только в том случае, если аннотация ***@Execution(CONCURRENT)*** присутствует в тестовом классе или методе.

Если параллельное выполнение включено и [ClassOrderer](https://junit.org/junit5/docs/snapshot/api/org.junit.jupiter.api/org/junit/jupiter/api/ClassOrderer.html) по умолчанию зарегистрирован (подробнее см. в разделе [Class Order](https://junit.org/junit5/docs/snapshot/user-guide/#writing-tests-test-execution-order-classes) ), тестовые классы верхнего уровня сначала будут соответствующим образом отсортированы и запланированы в этом порядке. Однако их запуск не гарантируется именно в этом порядке, поскольку потоки, в которых они выполняются, не контролируются непосредственно JUnit.

Все узлы дерева тестов, настроенные в режиме выполнения CONCURRENT, будут выполняться полностью параллельно согласно предоставленной [configuration](https://junit.org/junit5/docs/snapshot/user-guide/#writing-tests-parallel-execution-config) с соблюдением механизма декларативной [synchronization](https://junit.org/junit5/docs/snapshot/user-guide/#writing-tests-parallel-execution-synchronization). Обратите внимание, что захват стандартного вывода/ошибок ([Capturing Standard Output/Error](https://junit.org/junit5/docs/snapshot/user-guide/" \l "running-tests-capturing-output)) необходимо включать отдельно.

Кроме того, вы можете настроить режим выполнения по умолчанию*(в один поток)* для классов верхнего уровня, установив параметр конфигурации ***junit.jupiter.execution.parallel.mode.classes.default***. Объединив оба параметра конфигурации, вы можете настроить классы для параллельного выполнения, но их методы в одном потоке *(при предыдущей настройке каждый метод каждого класса выполнялся в своем потоке)*:

Параметры конфигурации для параллельного выполнения классов верхнего уровня, но методов в одном потоке

junit.jupiter.execution.parallel.enabled = true junit.jupiter.execution.parallel.mode.default = same\_thread junit.jupiter.execution.parallel.mode.classes.default = concurrent

Противоположная комбинация приведет к параллельному запуску всех методов внутри одного класса, но классы верхнего уровня будут выполняться последовательно:

Параметры конфигурации для последовательного выполнения классов верхнего уровня, но их методов параллельно

junit.jupiter.execution.parallel.enabled = true junit.jupiter.execution.parallel.mode.default = concurrent junit.jupiter.execution.parallel.mode.classes.default = same\_thread

Если параметр конфигурации junit.jupiter.execution.parallel.mode.classes.default не задан явно, вместо него будет использоваться значение junit.jupiter.execution.parallel.mode.default.

**1. Конфигурация**

Такие свойства, как желаемый параллелизм и максимальный размер пула, можно настроить с помощью ***ParallelExecutionConfigurationStrategy***. Платформа JUnit предоставляет:

две готовые реализации: 1. динамическую и 2. фиксированную.

3. Альтернативно вы можете реализовать собственную стратегию.

Чтобы выбрать стратегию, установите для параметра конфигурации ***junit.jupiter.execution.parallel.config.strategy*** одно из следующих значений.

dynamic *(установлено по умолчанию)*

Вычисляет желаемый параллелизм на основе количества доступных процессоров/ядер, умноженного на параметр конфигурации ***junit.jupiter.execution.parallel.config.dynamic.factor*** *(по умолчанию = 1)*. Необязательный параметр конфигурации ***junit.jupiter.execution.parallel.config.dynamic.max-pool-size-factor*** можно использовать для ограничения максимального количества потоков.

fixed

Использует обязательный параметр конфигурации ***junit.jupiter.execution.parallel.config.fixed.parallelism*** в качестве желаемого параллелизма. Необязательный параметр конфигурации ***junit.jupiter.execution.parallel.config.fixed.max-pool-size*** можно использовать для ограничения максимального количества потоков.

custom

Позволяет указать пользовательскую реализацию [ParallelExecutionConfigurationStrategy](https://junit.org/junit5/docs/snapshot/api/org.junit.platform.engine/org/junit/platform/engine/support/hierarchical/ParallelExecutionConfigurationStrategy.html) с помощью обязательного параметра конфигурации ***junit.jupiter.execution.parallel.config.custom.class*** для определения желаемой конфигурации.

Если стратегия конфигурации не задана, JUnit Jupiter использует стратегию динамической конфигурации с коэффициентом 1. Следовательно, желаемый параллелизм будет равен количеству доступных процессоров/ядер.

Сам по себе параллелизм не подразумевает максимальное количество параллельных потоков.

По умолчанию JUnit Jupiter не гарантирует, что количество одновременно выполняемых тестов не превысит настроенный параллелизм. Например, при использовании одного из механизмов синхронизации, описанных в следующем разделе, ForkJoinPool, который используется «за кулисами», может создавать дополнительные потоки, чтобы гарантировать продолжение выполнения с достаточным параллелизмом. Если вам требуются такие гарантии, в Java 9+ можно ограничить максимальное количество одновременных потоков, контролируя максимальный размер пула динамических, фиксированных и пользовательских стратегий.

Соответствующие свойства

В следующей таблице перечислены соответствующие свойства для настройки параллельного выполнения. См. «Параметры конфигурации» для получения подробной информации о том, как установить такие свойства.

| **Property** | **Description** | **Supported Values** | **Default Value** |
| --- | --- | --- | --- |
| junit.jupiter.execution.parallel.enabled | Включить параллельное выполнение тестов | * true * false | false |
| junit.jupiter.execution.parallel.mode.default | Режим выполнения узлов по умолчанию в дереве тестов | * concurrent * same\_thread | same\_thread |
| junit.jupiter.execution.parallel.mode.classes.default | Режим выполнения по умолчанию для классов верхнего уровня | * concurrent * same\_thread | same\_thread |
| junit.jupiter.execution.parallel.config.strategy | Стратегия выполнения для желаемого параллелизма и максимального размера пула | * dynamic * fixed * custom | dynamic |
| junit.jupiter.execution.parallel.config.dynamic.factor | Коэффициент, который необходимо умножить на количество доступных процессоров/ядер, чтобы определить желаемый параллелизм для стратегии динамической конфигурации. | a positive decimal number | 1.0 |
| junit.jupiter.execution.parallel.config.dynamic.max-pool-size-factor | для ограничения максимального количества потоков в режиме dinamic | положительное десятичное (decimal) число, должно быть больше или равно 1,0 | 256 + the value of junit.jupiter.execution.parallel.config.dynamic.factor multiplied by the number of available processors/cores |
| junit.jupiter.execution.parallel.config.dynamic.saturate | Отключить насыщение базового пула fork-join для стратегии динамической конфигурации. | * true * false | true |
| junit.jupiter.execution.parallel.config.fixed.parallelism | Желаемый параллелизм для стратегии фиксированной конфигурации | a positive integer | no default value |
| junit.jupiter.execution.parallel.config.fixed.max-pool-size | для ограничения максимального количества потоков в режиме fixed | положительное integer, должно быть больше или равно junit.jupiter.execution.parallel.config.fixed.parallelism | 256 + the value of junit.jupiter.execution.parallel.config.fixed.parallelism |
| junit.jupiter.execution.parallel.config.fixed.saturate | Отключить насыщение базового пула fork-join для стратегии фиксированной конфигурации. | * true * false | true |
| junit.jupiter.execution.parallel.config.custom.class | Полное имя класса ParallelExecutionConfigurationStrategy, которое будет использоваться для пользовательской стратегии конфигурации. | for example, *org.example.CustomStrategy* | no default value |

**2. Запуск тестов изолированно**

Если большинство ваших тестовых классов можно запускать параллельно без какой-либо синхронизации, но у вас есть некоторые тестовые классы, которые необходимо запускать изолированно, вы можете пометить последние аннотацией **@Isolated**. Тесты в таких классах выполняются последовательно без одновременного запуска других тестов.

**3. Синхронизация**

Помимо управления режимом выполнения с помощью аннотации ***@Execution***, JUnit Jupiter предоставляет еще один механизм декларативной синхронизации на основе аннотаций. Аннотация ***@ResourceLock*** позволяет объявить, что тестовый класс или метод использует определенный общий ресурс, к которому требуется синхронизированный доступ для обеспечения надежного выполнения теста. Общий ресурс идентифицируется уникальным именем, которое представляет собой строку. Имя может быть задано пользователем или одной из предопределенных констант в ресурсах: SYSTEM\_PROPERTIES, SYSTEM\_OUT, SYSTEM\_ERR, LOCALE или TIME\_ZONE.

Если бы тесты в следующем примере выполнялись параллельно без использования @ResourceLock, они были бы нестабильными. Иногда они проходили успешно, а иногда терпели неудачу из-за состояния гонки, присущего записи, а затем чтению одного и того же системного свойства JVM.

Когда доступ к общим ресурсам объявляется с помощью аннотации @ResourceLock, механизм JUnit Jupiter использует эту информацию, чтобы гарантировать, что никакие конфликтующие тесты не выполняются параллельно.

Помимо строки, которая однозначно идентифицирует общий ресурс, вы можете указать режим доступа. Два теста, требующие доступа *READ* к общему ресурсу, могут выполняться параллельно друг с другом, но не во время выполнения любого другого теста, требующего доступа *READ\_WRITE* к тому же общему ресурсу.

@Execution(CONCURRENT)

class SharedResourcesDemo {

private Properties backup;

@BeforeEach

void backup() {

backup = new Properties();

backup.putAll(System.getProperties());

}

@AfterEach

void restore() {

System.setProperties(backup);

}

@Test

@ResourceLock(value = SYSTEM\_PROPERTIES, mode = READ)

void customPropertyIsNotSetByDefault() { assertNull(System.getProperty("my.prop"));

}

@Test

@ResourceLock(value = SYSTEM\_PROPERTIES, mode = READ\_WRITE)

void canSetCustomPropertyToApple() {

System.**set**Property("my.prop", "apple");

assertEquals("apple", System.getProperty("my.prop"));

}

@Test

@ResourceLock(value = SYSTEM\_PROPERTIES, mode = READ\_WRITE)

void canSetCustomPropertyToBanana() {

System.**set**Property("my.prop", "banana");

assertEquals("banana", System.getProperty("my.prop"));

}

}