**Fork Опции и параллельное выполнение тестов**

Выбор правильной стратегии разветвления и настроек параллельного выполнения может оказать существенное влияние на требования к памяти и время выполнения вашей системы сборки.

Surefire предлагает различные варианты параллельного выполнения тестов, что позволяет максимально эффективно использовать имеющееся в вашем распоряжении оборудование. Но, в частности, разветвление также может помочь снизить требования к памяти.

На этой странице вы найдете некоторые идеи о том, как настроить выполнение теста так, чтобы оно лучше всего подходило для вашей среды.

**Параллельное выполнение тестов**

По сути, в плагине maven-surefire-plugin есть два способа добиться параллельного выполнения тестов:

**1.** Самый очевидный из них — использование параметра parallel. Возможные значения зависят от используемого поставщика тестов. Для JUnit 4.7 и более поздних версий это могут быть methods, classes, both, suites, suitesAndClasses, suitesAndMethods, classesAndMethods or  all. В качестве предварительного условия в тестах JUnit исполнитель JUnit должен расширять org.junit.runners.ParentRunner. Если в аннотации @org.junit.runner.RunWith не указан ранер, необходимое условие выполнено.

Начиная с maven-surefire-plugin:2.16, значение «both» устарело, но его все еще можно использовать и оно ведет себя так же, как и classAndMethods.

Подробности смотрите на страницах примеров для JUnit и TestNG.

Расширение параллелизма настраивается с использованием следующих параметров. Параметр useUnlimitedThreads допускает неограниченное количество потоков. Если useUnlimitedThreads=true, параметр threadCount можно использовать с необязательным параметром perCoreThreadCount=true (по умолчанию true). Параметры useUnlimitedThreads и threadCount следует интерпретировать в контексте значения, указанного для параллельного параметра.

Начиная с maven-surefire-plugin:2.16, можно накладывать ограничения на количество потоков для наборов, классов или методов, используя один или несколько параметров threadCountSuites, threadCountClasses и threadCountMethods. Если указан только threadCount, плагин maven-surefire пытается оптимизировать количество потоков для наборов, классов и методов и повторно использует потоки в пользу **leaf**, например. параллельные методы (возможно увеличение количества параллельных методов).

В первом примере с неограниченным количеством потоков можно использовать максимум три параллельных потока для выполнения наборов: Parallel=all, useUnlimitedThreads=true, threadCountSuites=3.

Во втором примере количество параллельных методов строго не ограничено:parallel=classesAndMethods, threadCount=8, threadCountClasses=3. Здесь количество параллельных методов варьируется от 5 до 7. Соответственно Parallel=all, но сумма threadCountSuites и threadCountClasses не должна превышать определенную (threadCount - 1). Возможны другие комбинации с **leaf** с неуказанным числом потоков. Убедитесь, что **leaf** является последним из параллельно упорядоченных наборов-классов-методов.

В третьем примере количество потоков представляет собой соотношение, например. для parallel=all, threadCount=16, threadCountSuites=2, threadCountClasses=3, threadCountMethods=5. Таким образом, параллельные наборы составят 20%, параллельные классы — 30% и параллельные методы — 50%.

Наконец, threadCount и useUnlimitedThreads не обязательно настраиваются, если для значения parallel указано эквивалентное количество потоков.

Плагин maven-surefire пытается повторно использовать потоки, тем самым оптимизируя количество потоков и предпочитая справедливость потоков. Оптимизация parallelOptimized числа потоков включена по умолчанию, например. количество запускающих Suite не обязательно должно тратить ресурсы Threads Suite. Если используется threadCount, то **leaf** с неограниченным количеством потоков может ускориться, особенно в конце фазы тестирования.

Параметры ParallelTestsTimeoutInSeconds и ParallelTestsTimeoutForcedInSeconds используются для указания дополнительного времени ожидания при параллельном выполнении. Если время ожидания истекло, плагин печатает сводный журнал со строками ОШИБКИ: «Эти тесты были выполнены до операции выключения» и «Эти тесты не завершены», если работающие потоки были прерваны.

Примечание. По задумке исполнителей JUnit, статические методы, помеченные, например, @Parameters, @BeforeClass и @AfterClass вызываются в родительском потоке. Ради видимости памяти между потоками синхронизируйте методы. См. ключевые слова: *olatile*, *synchronized*, **immutable** and *final* в [Java Memory Model - JSR-133](https://jcp.org/en/jsr/detail?id=133).

При использовании параллельного варианта важно помнить: параллелизм происходит внутри одного и того же процесса JVM. Это эффективно с точки зрения памяти и времени выполнения, но вы можете быть более уязвимы к условиям гонки или другому неожиданному и трудно воспроизводимому поведению.

**2.** Другая возможность параллельного выполнения теста — установка для параметра forkCount значения выше 1. В следующем разделе описаны подробности об этом и связанном с ним параметре reuseForks. Использование reuseForks=true (по умолчанию) и разветвление тестовых классов в многоразовых JVM может привести к той же проблеме с общим статическим кодом в инициализаторах классов @BeforeClass при использовании parallel без разветвления. Поэтому установка reuseForks=false может помочь, однако не гарантирует правильную работу некоторых функций, например. skipAfterFailureCount.

**Параллельное выполнение тестов и выполнение одного потока**

Как упоминалось выше, выполнение теста parallel используется с определенным количеством потоков. Начиная с maven-surefire-plugin:2.18, вы можете применить аннотацию JCIP @net.jcip.annotations.NotThreadSafe к классу Java теста JUnit (чистый тестовый класс, набор, параметризованный и т. д.), чтобы выполнить его в одном Экземпляре потока. Поток имеет имя *maven-surefire-plugin@NotThreadSafe* и выполняется в конце тестового запуска.

Просто используйте зависимость проекта *net.jcip:jcip-annotations:1.0* или другой артефакт *com.github.stephenc.jcip:jcip-annotations:1.0-1* с Apache License 2.0.

1. <dependencies>
   1. [...]
   2. <dependency>
      1. <groupId>junit</groupId>
      2. <artifactId>junit</artifactId>
      3. <!-- 4.7 or higher -->
      4. <version>4.7</version>
      5. <scope>test</scope>
   3. </dependency>
   4. <dependency>
      1. <groupId>com.github.stephenc.jcip</groupId>
      2. <artifactId>jcip-annotations</artifactId>
      3. <version>1.0-1</version>
      4. <scope>test</scope>
   5. </dependency>
   6. [...]
2. </dependencies>

Таким образом, параллельное выполнение классов тестов, аннотированных @NotThreadSafe, разветвляется в экземпляре одного потока (не имеется в виду разветвленный процесс JVM).

Если *Suite* или *Parameterized* аннотирован с помощью @NotThreadSafe, классы набора выполняются в одном потоке. Вы также можете аннотировать отдельный тестовый класс, на который ссылается Suite, а другие неаннотированные тестовые классы в Suite могут запускаться параллельно. Таким образом, вы можете изолировать конфликтующие группы тестов и при этом запускать отдельные тесты параллельно.

Примечание. По задумке исполнителей JUnit, статические методы, помеченные, например, @Parameters, @BeforeClass и @AfterClass вызываются в родительском потоке. Назначьте классы @NotThreadSafe Suite, чтобы избежать этой проблемы. Если вы не хотите менять иерархию своих тестовых классов, вы можете синхронизировать такие методы ради улучшения видимости памяти в качестве упрощенного решения. См. ключевые слова: *volatile*, *synchronized*, **immutable** and *final* в [Java Memory Model - JSR-133](https://jcp.org/en/jsr/detail?id=133).

**Параллельное выполнение набора тестов (Test-Suite)**

Например, предположим, что ваши тесты расположены следующим образом:

1. src/test/java
2. +-features.areaA
3. | +-SomeTest.java
4. | +-AnotherTest.java
5. +-features.areaB
6. | +-DifferentTest.java
7. | +-OtherTest.java

Вы должны добавить TestSuite.java для каждого пакета, который будет выглядеть примерно так:

1. package features.areaA;
3. import org.junit.runner.RunWith;
4. import org.junit.runners.Suite;
6. @RunWith(Suite.class)
7. @Suite.SuiteClasses({SomeTest.class,AnotherTest.class})
8. public class TestSuite {
9. }

и аналогично для пакета Features.areaB.

Затем вы настроите плагин maven-surefire следующим образом:

1. <plugin>
   1. <groupId>org.apache.maven.plugins</groupId>
   2. <artifactId>maven-surefire-plugin</artifactId>
   3. <version>3.2.5</version>
   4. <configuration>
      1. <includes>
         1. <include>features.\*.TestSuite</include>
      2. </includes>
      3. <parallel>suites</parallel>
      4. <threadCountSuites>2</threadCountSuites>
      5. <perCoreThreadCount>false</perCoreThreadCount>
   5. </configuration>
2. </plugin>