(/)

Semaphores in Java

Last modified: April 27, 2020

by baeldung (https://www.baeldung.com/author/baeldung/)

Java (https://www.baeldung.com/category/java/) +

Java Concurrency (https://www.baeldung.com/tag/java-concurrency/)

1. Overview

In this qui

2. *Semaphore*

We'll start with *java.util.concurrent.Semaphore*. We can use semaphores to limit the number of concurrent threads accessing a specific resource.

In the following example, we will implement a simple login queue to limit the number of users in the system:

```
class LoginQueueUsingSemaphore {

    private Semaphore semaphore;

    public LoginQueueUsingSemaphore(int slotLimit) {
        semaphore = new Semaphore(slotLimit);
    }

    boolean tryLogin() {
        return semaphore.tryAcquire();
    }

    void logout() {
        semaphore.release();
    }

    int availableSlots() {
        return semaphore.availablePermits();
    }

}
```

Notice how we used the following methods:

- *tryAcquire()* – return true if a permit is available immediately and acquire it otherwise return false, but *acquire()* acquires a permit and blocking until one is available
- *release() – release a permit*
- *availablePermits()* – return number of current permits available

To test our login queue, we will first try to reach the limit and check if the next login attempt will be blocked:

```
@Test
public void givenLoginQueue whenReachLimit thenBlocked() {
    executorService.shutdown();

    assertEquals(0, loginQueue.availableSlots());
    assertFalse(loginQueue.tryLogin());
}
```

Next, we will see if any slots are available after a logout:

```
@Test
public void givenLoginQueue_whenLogout_thenSlotsAvailable() {
    int slots = 10;
    ExecutorService executorService = Executors.newFixedThreadPool(slots);
    LoginQueueUsingSemaphore loginQueue = new LoginQueueUsingSemaphore(slots);
    IntStream.range(0, slots)
      .forEach(user -> executorService.execute(loginQueue::tryLogin));
    executorService.shutdown();
    assertEquals(0, loginQueue.availableSlots());
    loginQueue.logout();

    assertTrue(loginQueue.availableSlots() > 0);
    assertTrue(loginQueue.tryLogin());
}
```

3. Timed *Semaphore*

Next, we will discuss Apache Commons *TimedSemaphore*. *TimedSemaphore* allows a number of permits as a simple Semaphore but in a given period of time, after this period the time reset and all permits are released.

We can use *TimedSemaphore* to build a simple delay queue as follows:

```
class DelayQueueUsingTimedSemaphore {

    private TimedSemaphore semaphore;

    DelayQueueUsingTimedSemaphore(long period, int slotLimit) {
        semaphore = new TimedSemaphore(period, TimeUnit.SECONDS, slotLimit);
    }

    boolean tryAdd() {
        return semaphore.tryAcquire();
    }

    int availableSlots() {
        return semaphore.getAvailablePermits();
    }

}
```

When we use a delay queue with one second as time period and after using all the slots within one second, none should be available:

```
public void givenDelayQueue_whenReachLimit_thenBlocked() {
    int slots = 50;
    ExecutorService executorService = Executors.newFixedThreadPool(slots);
    DelayQueueUsingTimedSemaphore delayQueue
      = new DelayQueueUsingTimedSemaphore(1, slots);

    IntStream.range(0, slots)
      .forEach(user -> executorService.execute(delayQueue::tryAdd));
    executorService.shutdown();

    assertEquals(0, delayQueue.availableSlots());

}
```

But after s

```
@Test
public void givenDelayQueue_whenTimePass_thenSlotsAvailable() throws InterruptedException {
    int slots = 50;
    ExecutorService executorService = Executors.newFixedThreadPool(slots);
    DelayQueueUsingTimedSemaphore delayQueue = new DelayQueueUsingTimedSemaphore(1, slots);
    IntStream.range(0, slots)
      .forEach(user -> executorService.execute(delayQueue::tryAdd));
    executorService.shutdown();

    assertEquals(0, delayQueue.availableSlots());
    Thread.sleep(1000);
    assertTrue(delayQueue.availableSlots() > 0);
    assertTrue(delayQueue.tryAdd());
}
```

4. Semaphore vs. Mutex

Mutex acts similarly to a binary semaphore, we can use it to implement mutual exclusion.

In the following example, we'll use a simple binary semaphore to build a counter:

```
class CounterUsingMutex {

    private Semaphore mutex;
    private int count;

    CounterUsingMutex() {
        mutex = new Semaphore(1);
        count = 0;
    }

    void increase() throws InterruptedException {
        mutex.acquire();
        this.count = this.count + 1;
        Thread.sleep(1000);
        mutex.release();

    }

    int getCount() {
        return this.count;
    }



}
```

When a lot of threads try to access the counter at once, **they'll simply be blocked in a queue**:

```
@Test
public void whenMutexAndMultipleThreads_thenBlocked()
 throws InterruptedException {
    int count = 5;
    ExecutorService executorService
     = Executors.newFixedThreadPool(count);
    CounterUsingMutex counter = new CounterUsingMutex();
    IntStream.range(0, count)
      .forEach(user -> executorService.execute(() -> {
          try {
              counter.increase();
          } catch (InterruptedException e) {
              e.printStackTrace();
          }
      }));
    executorService.shutdown();

    assertTrue(counter.hasQueuedThreads());
}
```

When we wait, all threads will access the counter and no threads left in the queue:

```
@Test
public void givenMutexAndMultipleThreads_ThenDelay_thenCorrectCount()
 throws InterruptedException {
    int count = 5;
    ExecutorService executorService
     = Executors.newFixedThreadPool(count);
    CounterUsingMutex counter = new CounterUsingMutex();
    IntStream.range(0, count)
      .forEach(user -> executorService.execute(() -> {
          try {
              counter.increase();
          } catch (InterruptedException e) {
              e.printStackTrace();
          }
      }));
    executorService.shutdown();

    assertTrue(counter.hasQueuedThreads());
    Thread.sleep(5000);
    assertFalse(counter.hasQueuedThreads());
    assertEquals(count, counter.getCount());
}
```

5. Conclusion

In this article, we explored the basics of semaphores in Java.

As always, the full source code is available over on GitHub (https://github.com/eugenp/tutorials/tree/master/core-java-modules/core-java-concurrency-advanced-2).

Comments are closed on this article!

CATEGORIES

SPRING (HTTPS://WWW.BAELDUNG.COM/CATEGORY/SPRING/)
REST (HTTPS://WWW.BAELDUNG.COM/CATEGORY/REST/)
JAVA (HTTPS://WWW.BAELDUNG.COM/CATEGORY/JAVA/)
SECURITY (HTTPS://WWW.BAELDUNG.COM/CATEGORY/SECURITY-2/)
PERSISTENCE (HTTPS://WWW.BAELDUNG.COM/CATEGORY/PERSISTENCE/)
JACKSON (HTTPS://WWW.BAELDUNG.COM/CATEGORY/JSON/JACKSON/)
HTTP CLIENT-SIDE (HTTPS://WWW.BAELDUNG.COM/CATEGORY/HTTP/)
KOTLIN (HTTPS://WWW.BAELDUNG.COM/CATEGORY/KOTLIN/)

SERIES

JAVA "BACK TO BASICS" TUTORIAL (/JAVA-TUTORIAL)
JACKSON JSON TUTORIAL (/JACKSON)
HTTPCLIENT 4 TUTORIAL (/HTTPCLIENT-GUIDE)
REST WITH SPRING TUTORIAL (/REST-WITH-SPRING-SERIES)
SPRING PERSISTENCE TUTORIAL (/PERSISTENCE-WITH-SPRING-SERIES)
SECURITY WITH SPRING (/SECURITY-SPRING)