Java Semaphore examples

By mkyong | Last updated: December 13, 2018

In Java, we can use `Semaphore` to limit the number of threads to access a certain resource.

1. What is Semaphore?

In short, a semaphore maintains a set of permits (tickets), each `acquire()` will take a permit (ticket) from semaphore, each `release()` will return back the permit (ticket) back to the semaphore. If permits (tickets) are not available, `acquire()` will block until one is available.

```
// 5 tickets
Semaphore semaphore = new Semaphore(5);

// take 1 ticket
semaphore.acquire();

// 4
semaphore.availablePermits();

// return back ticket
semaphore.release();

// 5
semaphore.availablePermits();
```


2. Java Semaphore

A Java Semaphore example to limit the number of tasks running in `ExecutorService`. In this example, 5 `Callable` tasks are submitted to `ExecutorService`, but only 2 tasks are running concurrently.

```
TaskLimitSemaphore.java
```

```
package com.mkyong.concurrency.synchronizer.semaphore;

import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.concurrent.*;

// Throttle task submission
public class TaskLimitSemaphore {

    private final ExecutorService executor;
    private final Semaphore semaphore;

    public TaskLimitSemaphore(ExecutorService executor, int limit) {
        this.executor = executor;
        this.semaphore = new Semaphore(limit);
    }

    public <T> Future<T> submit(final Callable<T> task) throws InterruptedException {

        semaphore.acquire();
        System.out.println("semaphore.acquire()...");

        return executor.submit(() -> {
            try {
                return task.call();
            } finally {
                semaphore.release();
                System.out.println("semaphore.release()...");
            }
        });

    }

    private static final DateFormat sdf = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss");

    public static void main(String[] args) throws InterruptedException {

        ExecutorService executor = Executors.newCachedThreadPool();

        // Only 2 tasks are able to run concurrently
        TaskLimitSemaphore obj = new TaskLimitSemaphore(executor, 2);

        obj.submit(() -> {
            System.out.println(getCurrentDateTime() + " : task1 is running!");
            Thread.sleep(2000);
            System.out.println(getCurrentDateTime() + " : task1 is done!");
            return 1;
        });

        obj.submit(() -> {
            System.out.println(getCurrentDateTime() + " : task2 is running!");
            Thread.sleep(2000);
            System.out.println(getCurrentDateTime() + " task2 is done!");
            return 2;
        });

        obj.submit(() -> {
            System.out.println(getCurrentDateTime() + " task3 is running!");
            Thread.sleep(2000);
            System.out.println(getCurrentDateTime() + " task3 is done!");
            return 3;
        });

        obj.submit(() -> {
            System.out.println(getCurrentDateTime() + " task4 is running!");
            Thread.sleep(2000);
            System.out.println(getCurrentDateTime() + " task4 is done!");
            return 4;
        });
```

```
        obj.submit(() -> {
            System.out.println(getCurrentDateTime() + " task5 is running!");
            Thread.sleep(2000);
            System.out.println(getCurrentDateTime() + " task5 is done!");
            return 5;
        });

        executor.shutdown();
    }

    private static String getCurrentDateTime() {
        return sdf.format(new Date());
    }
}
```

Output

```
semaphore.acquire()...
semaphore.acquire()...
2018/12/06 18:45:22 : task1 is running!
2018/12/06 18:45:22 : task2 is running!
2018/12/06 18:45:24 : task1 is done!
2018/12/06 18:45:24 task2 is done!
semaphore.release()...
semaphore.acquire()...
semaphore.release()...
semaphore.acquire()...
2018/12/06 18:45:24 task3 is running!
2018/12/06 18:45:24 task4 is running!
2018/12/06 18:45:26 task4 is done!
2018/12/06 18:45:26 task3 is done!
semaphore.acquire()...
semaphore.release()...
semaphore.release()...
2018/12/06 18:45:26 task5 is running!
2018/12/06 18:45:28 task5 is done!
semaphore.release()...
```

3. Mutex

In short, always `new Semaphore(1)`, only one thread is allowed to access a certain resource.

```
PrintSequenceCallable.java
```

Copy

```
package com.mkyong.concurrency.synchronizer.semaphore;

import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Semaphore;

public class SshLoginSemaphore {

    private final Semaphore mutex;

    // only 1 user is allow
    public SshLoginSemaphore() {
        this.mutex = new Semaphore(1);
    }

    private void ssh(String user) throws InterruptedException {

        mutex.acquire();
        System.out.println(getCurrentDateTime() + " : " + user + " mutex.acquire()");

        Thread.sleep(2000);

        mutex.release();
        System.out.println(getCurrentDateTime() + " : " + user + " mutex.release()");

    }

    private static final DateFormat sdf = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss");

    public static void main(String[] args) {

        ExecutorService executor = Executors.newFixedThreadPool(5);

        SshLoginSemaphore task = new SshLoginSemaphore();

        // submit 3 tasks
        executor.submit(() -> {

            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        });

        executor.submit(() -> {
            try {
                task.ssh("yflow");
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        });

        executor.submit(() -> {
            try {
                task.ssh("zilap");
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        });

        executor.shutdown();

    }

    private static String getCurrentDateTime() {
        return sdf.format(new Date());
    }
```

```
}
```

Output

Review the time (seconds), only one thread is allowed, one `acquire()` and one `release()`

```
2018/12/06 18:54:25 : mkyong mutex.acquire()
2018/12/06 18:54:27 : yflow mutex.acquire()
2018/12/06 18:54:27 : mkyong mutex.release()
2018/12/06 18:54:29 : zilap mutex.acquire()
2018/12/06 18:54:29 : yflow mutex.release()
2018/12/06 18:54:31 : zilap mutex.release()
```

Download Source Code

> $ git clone *https://github.com/mkyong/java-concurrency.git*

References

1. *Wikipedia – Semaphore (programming)*
2. *Oracle doc – Semaphore*

Related Articles

- Java Thread - Mutex and Semaphore example
- Java BlockingQueue examples
- Review : Java 7 Concurrency Cookbook
- Java ScheduledExecutorService examples
- Java ExecutorService examples

mkyong

Founder of Mkyong.com, love Java and open source stuff. Follow him on Twitter. If you like my tutorials, consider make a donation to these charities. Read all published posts by mkyong

Comments

Sanjoy Roy

Guest

not working semaphore code . getting following error
Exception in thread “main” java.lang.Error: Unresolved compilation problems:
The method submit(() -> {}) is undefined for the type TaskLimitSemaphore
The method submit(() -> {}) is undefined for the type TaskLimitSemaphore
The method submit(() -> {}) is undefined for the type TaskLimitSemaphore
The method submit(() -> {}) is undefined for the type TaskLimitSemaphore
The method submit(() -> {}) is undefined for the type TaskLimitSemaphore

+ 0 - Reply

1 month ago