本章主要对Semaphore进行学习。

1.Semaphore简介

Semaphore，是JDK1.5的java.util.concurrent并发包中提供的一个并发工具类。

所谓Semaphore即 信号量 的意思。

这个叫法并不能很好地表示它的作用，更形象的说法应该是许可证管理器。

其作用在JDK注释中是这样描述的：

A counting semaphore.

Conceptually, a semaphore maintains a set of permits.

Each {@link #acquire} blocks if necessary until a permit is available, and then takes it.

Each {@link #release} adds a permit, potentially releasing a blocking acquirer.

However, no actual permit objects are used; the {@code Semaphore} just keeps a count of the number available and acts accordingly.

翻译过来，就是：

Semaphore是一个计数信号量。

从概念上将，Semaphore包含一组许可证。

如果有需要的话，每个acquire()方法都会阻塞，直到获取一个可用的许可证。

每个release()方法都会释放持有许可证的线程，并且归还Semaphore一个可用的许可证。

然而，实际上并没有真实的许可证对象供线程使用，Semaphore只是对可用的数量进行管理维护。

2.Semaphore方法说明

Semaphore的方法如下：

——Semaphore(permits)

初始化许可证数量的构造函数

——Semaphore(permits,fair)

初始化许可证数量和是否公平模式的构造函数

——isFair()

是否公平模式FIFO

——availablePermits()

获取当前可用的许可证数量

——acquire()

当前线程尝试去阻塞的获取1个许可证。

此过程是阻塞的，它会一直等待许可证，直到发生以下任意一件事：

当前线程获取了1个可用的许可证，则会停止等待，继续执行。

当前线程被中断，则会抛出InterruptedException异常，并停止等待，继续执行。

——acquire(permits)

当前线程尝试去阻塞的获取permits个许可证。

此过程是阻塞的，它会一直等待许可证，直到发生以下任意一件事：

当前线程获取了n个可用的许可证，则会停止等待，继续执行。

当前线程被中断，则会抛出InterruptedException异常，并停止等待，继续执行。

——acquierUninterruptibly()

当前线程尝试去阻塞的获取1个许可证(不可中断的)。

此过程是阻塞的，它会一直等待许可证，直到发生以下任意一件事：

当前线程获取了1个可用的许可证，则会停止等待，继续执行。

——acquireUninterruptibly(permits)

当前线程尝试去阻塞的获取permits个许可证。

此过程是阻塞的，它会一直等待许可证，直到发生以下任意一件事：

当前线程获取了n个可用的许可证，则会停止等待，继续执行。

——tryAcquire()

当前线程尝试去获取1个许可证。

此过程是非阻塞的，它只是在方法调用时进行一次尝试。

如果当前线程获取了1个可用的许可证，则会停止等待，继续执行，并返回true。

如果当前线程没有获得这个许可证，也会停止等待，继续执行，并返回false。

——tryAcquire(permits)

当前线程尝试去获取permits个许可证。

此过程是非阻塞的，它只是在方法调用时进行一次尝试。

如果当前线程获取了permits个可用的许可证，则会停止等待，继续执行，并返回true。

如果当前线程没有获得permits个许可证，也会停止等待，继续执行，并返回false。

——tryAcquire(timeout,TimeUnit)

当前线程在限定时间内，阻塞的尝试去获取1个许可证。

此过程是阻塞的，它会一直等待许可证，直到发生以下任意一件事：

当前线程获取了可用的许可证，则会停止等待，继续执行，并返回true。

当前线程等待时间timeout超时，则会停止等待，继续执行，并返回false。

当前线程在timeout时间内被中断，则会抛出InterruptedException一次，并停止等待，继续执行。

——tryAcquire(permits,timeout,TimeUnit)

当前线程在限定时间内，阻塞的尝试去获取permits个许可证。

此过程是阻塞的，它会一直等待许可证，直到发生以下任意一件事：

当前线程获取了可用的permits个许可证，则会停止等待，继续执行，并返回true。

当前线程等待时间timeout超时，则会停止等待，继续执行，并返回false。

当前线程在timeout时间内被中断，则会抛出InterruptedException一次，并停止等待，继续执行。

——release()

当前线程释放1个可用的许可证。

——release(permits)

当前线程释放permits个可用的许可证。

——drainPermits()

当前线程获得剩余的所有可用许可证。

——hasQueuedThreads()

判断当前Semaphore对象上是否存在正在等待许可证的线程。

——getQueueLength()

获取当前Semaphore对象上是正在等待许可证的线程数量。

3.Semaphore方法练习

练习目的：熟悉Semaphore的各类方法的用法。

实例代码：

//new Semaphore(permits)：初始化许可证数量的构造函数

Semaphore semaphore = new Semaphore(5);

//new Semaphore(permits,fair):初始化许可证数量和是否公平模式的构造函数

semaphore = new Semaphore(5, true);

//isFair()：是否公平模式FIFO

System.out.println("是否公平FIFO：" + semaphore.isFair());

//availablePermits():获取当前可用的许可证数量

System.out.println("获取当前可用的许可证数量：开始---" + semaphore.availablePermits());

//acquire():获取1个许可证

//---此线程会一直阻塞，直到获取这个许可证，或者被中断(抛出InterruptedException异常)。

semaphore.acquire();

System.out.println("获取当前可用的许可证数量：acquire 1 个---" + semaphore.availablePermits());

//release()：释放1个许可证

semaphore.release();

System.out.println("获取当前可用的许可证数量：release 1 个---" + semaphore.availablePermits());

//acquire(permits):获取n个许可证

//---此线程会一直阻塞，直到获取全部n个许可证,或者被中断(抛出InterruptedException异常)。

semaphore.acquire(2);

System.out.println("获取当前可用的许可证数量：acquire 2 个---" + semaphore.availablePermits());

//release(permits):释放n个许可证

semaphore.release(2);

System.out.println("获取当前可用的许可证数量：release 1 个---" + semaphore.availablePermits());

//hasQueuedThreads():是否有正在等待许可证的线程

System.out.println("是否有正在等待许可证的线程：" + semaphore.hasQueuedThreads());

//getQueueLength():正在等待许可证的队列长度(线程数量)

System.out.println("正在等待许可证的队列长度(线程数量)：" + semaphore.getQueueLength());

Thread.sleep(10);

System.out.println();

//定义final的信号量

Semaphore finalSemaphore = semaphore;

new Thread(() -> {

//drainPermits():获取剩余的所有的许可证

int permits = finalSemaphore.drainPermits();

System.out.println(Thread.currentThread().getName() + "获取了剩余的全部" + permits + "个许可证.");

try {

Thread.sleep(2000);

} catch (InterruptedException e) {

e.printStackTrace();

}

//释放所有的许可证

finalSemaphore.release(permits);

System.out.println(Thread.currentThread().getName() + "释放了" + permits + "个许可证.");

}).start();

Thread.sleep(10);

new Thread(() -> {

try {

//有一个线程正在等待获取1个许可证

finalSemaphore.acquire();

System.out.println(Thread.currentThread().getName() + "获取了1个许可证.");

} catch (InterruptedException e) {

e.printStackTrace();

}

//释放1个许可证

finalSemaphore.release();

System.out.println(Thread.currentThread().getName() + "释放了1个许可证.");

}).start();

Thread.sleep(10);

System.out.println();

System.out.println("获取当前可用的许可证数量：drain 剩余的---" + finalSemaphore.availablePermits());

System.out.println("是否有正在等待许可证的线程：" + finalSemaphore.hasQueuedThreads());

System.out.println("正在等待许可证的队列长度(线程数量)：" + finalSemaphore.getQueueLength());

System.out.println();

Thread.sleep(10);

new Thread(() -> {

try {

//有一个线程正在等待获取2个许可证

finalSemaphore.acquire(2);

System.out.println(Thread.currentThread().getName() + "获取了2个许可证.");

} catch (InterruptedException e) {

e.printStackTrace();

}

//释放两个许可证

finalSemaphore.release(2);

System.out.println(Thread.currentThread().getName() + "释放了2个许可证.");

}).start();

Thread.sleep(10);

System.out.println();

System.out.println("获取当前可用的许可证数量：drain 剩余的---" + finalSemaphore.availablePermits());

System.out.println("是否有正在等待许可证的线程：" + finalSemaphore.hasQueuedThreads());

System.out.println("正在等待许可证的队列长度(线程数量)：" + finalSemaphore.getQueueLength());

System.out.println();

Thread.sleep(5000);

System.out.println();

System.out.println("获取当前可用的许可证数量：---" + finalSemaphore.availablePermits());

System.out.println("是否有正在等待许可证的线程：" + finalSemaphore.hasQueuedThreads());

System.out.println("正在等待许可证的队列长度(线程数量)：" + finalSemaphore.getQueueLength());

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

90

91

92

93

94

95

96

97

98

99

100

运行结果：

是否公平FIFO：true

获取当前可用的许可证数量：开始---5

获取当前可用的许可证数量：acquire 1 个---4

获取当前可用的许可证数量：release 1 个---5

获取当前可用的许可证数量：acquire 2 个---3

获取当前可用的许可证数量：release 1 个---5

是否有正在等待许可证的线程：false

正在等待许可证的队列长度(线程数量)：0

Thread-0获取了剩余的全部5个许可证.

获取当前可用的许可证数量：drain 剩余的---0

是否有正在等待许可证的线程：true

正在等待许可证的队列长度(线程数量)：1

获取当前可用的许可证数量：drain 剩余的---0

是否有正在等待许可证的线程：true

正在等待许可证的队列长度(线程数量)：2

Thread-0释放了5个许可证.

Thread-2获取了2个许可证.

Thread-1获取了1个许可证.

Thread-1释放了1个许可证.

Thread-2释放了2个许可证.

获取当前可用的许可证数量：---5

是否有正在等待许可证的线程：false

正在等待许可证的队列长度(线程数量)：0

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

4.Semaphore应用场景-实例

Semaphore经常用于限制获取某种资源的线程数量。

场景说明：

模拟学校食堂的窗口打饭过程

学校食堂有2个打饭窗口

学校中午有20个学生 按次序 排队打饭

每个人打饭时耗费时间不一样

有的学生耐心很好，他们会一直等待直到打到饭

有的学生耐心不好，他们等待时间超过了心里预期，就不再排队，而是回宿舍吃泡面了

有的学生耐心很好，但是突然接到通知，说是全班聚餐，所以也不用再排队，而是去吃大餐了

重点分析

食堂有2个打饭窗口：需要定义一个permits=2的Semaphore对象。

学生 按次序 排队打饭：此Semaphore对象是公平的。

有20个学生：定义20个学生线程。

打到饭的学生：调用了acquireUninterruptibly()方法，无法被中断

吃泡面的学生：调用了tryAcquire(timeout,TimeUnit)方法，并且等待时间超时了

吃大餐的学生：调用了acquire()方法，并且被中断了

实例代码：

定义2个窗口的食堂

/\*\*

\* 打饭窗口

\* 2： 2个打饭窗口

\* true：公平队列-FIFO

\*/

static Semaphore semaphore = new Semaphore(2, true);

1

2

3

4

5

6

定义打饭学生

/\*\*

\* <p>打饭学生</p>

\*

\* @author hanchao 2018/3/31 19:45

\*\*/

static class Student implements Runnable {

private static final Logger LOGGER = Logger.getLogger(Student.class);

//学生姓名

private String name;

//打饭许可

private Semaphore semaphore;

/\*\*

\* 打饭方式

\* 0 一直等待直到打到饭

\* 1 等了一会不耐烦了，回宿舍吃泡面了

\* 2 打饭中途被其他同学叫走了，不再等待

\*/

private int type;

public Student(String name, Semaphore semaphore, int type) {

this.name = name;

this.semaphore = semaphore;

this.type = type;

}

/\*\*

\* <p>打饭</p>

\*

\* @author hanchao 2018/3/31 19:49

\*\*/

@Override

public void run() {

//根据打饭情形分别进行不同的处理

switch (type) {

//打饭时间

//这个学生很有耐心，它会一直排队直到打到饭

case 0:

//排队

semaphore.acquireUninterruptibly();

//进行打饭

try {

Thread.sleep(RandomUtils.nextLong(1000, 3000));

} catch (InterruptedException e) {

e.printStackTrace();

}

//将打饭机会让后后面的同学

semaphore.release();

//打到了饭

LOGGER.info(name + " 终于打到了饭.");

break;

//这个学生没有耐心，等了1000毫秒没打到饭，就回宿舍泡面了

case 1:

//排队

try {

//如果等待超时，则不再等待，回宿舍吃泡面

if (semaphore.tryAcquire(RandomUtils.nextInt(6000, 16000), TimeUnit.MILLISECONDS)) {

//进行打饭

try {

Thread.sleep(RandomUtils.nextLong(1000, 3000));

} catch (InterruptedException e) {

e.printStackTrace();

}

//将打饭机会让后后面的同学

semaphore.release();

//打到了饭

LOGGER.info(name + " 终于打到了饭.");

} else {

//回宿舍吃泡面

LOGGER.info(name + " 回宿舍吃泡面.");

}

} catch (InterruptedException e) {

//e.printStackTrace();

}

break;

//这个学生也很有耐心，但是他们班突然宣布聚餐，它只能放弃打饭了

case 2:

//排队

try {

semaphore.acquire();

//进行打饭

try {

Thread.sleep(RandomUtils.nextLong(1000, 3000));

} catch (InterruptedException e) {

//e.printStackTrace();

}

//将打饭机会让后后面的同学

semaphore.release();

//打到了饭

LOGGER.info(name + " 终于打到了饭.");

} catch (InterruptedException e) {

//e.printStackTrace();

//被叫去聚餐，不再打饭

LOGGER.info(name + " 全部聚餐，不再打饭.");

}

break;

default:

break;

}

}

}

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

90

91

92

93

94

95

96

97

98

99

100

101

102

编写食堂打饭过程：

/\*\*

\* <p>食堂打饭</p>

\*

\* @author hanchao 2018/3/31 21:13

\*\*/

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

//101班的学生

Thread[] students101 = new Thread[5];

for (int i = 0; i < 20; i++) {

//前10个同学都在耐心的等待打饭

if (i < 10) {

new Thread(new Student("打饭学生" + i, SemaphoreDemo.semaphore, 0)).start();

} else if (i >= 10 && i < 15) {//这5个学生没有耐心打饭，只会等1000毫秒

new Thread(new Student("泡面学生" + i, SemaphoreDemo.semaphore, 1)).start();

} else {//这5个学生没有耐心打饭

students101[i - 15] = new Thread(new Student("聚餐学生" + i, SemaphoreDemo.semaphore, 2));

students101[i - 15].start();

}

}

//

Thread.sleep(5000);

for (int i = 0; i < 5; i++) {

students101[i].interrupt();

}

}

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

运行结果：

2018-04-01 21:13:16 INFO - 打饭学生1 终于打到了饭.

2018-04-01 21:13:16 INFO - 打饭学生0 终于打到了饭.

2018-04-01 21:13:18 INFO - 打饭学生2 终于打到了饭.

2018-04-01 21:13:18 INFO - 打饭学生3 终于打到了饭.

2018-04-01 21:13:19 INFO - 聚餐学生15 全部聚餐，不再打饭.

2018-04-01 21:13:19 INFO - 聚餐学生19 全部聚餐，不再打饭.

2018-04-01 21:13:19 INFO - 聚餐学生17 全部聚餐，不再打饭.

2018-04-01 21:13:19 INFO - 聚餐学生18 全部聚餐，不再打饭.

2018-04-01 21:13:19 INFO - 聚餐学生16 全部聚餐，不再打饭.

2018-04-01 21:13:19 INFO - 打饭学生4 终于打到了饭.

2018-04-01 21:13:20 INFO - 打饭学生5 终于打到了饭.

2018-04-01 21:13:21 INFO - 泡面学生13 回宿舍吃泡面.

2018-04-01 21:13:21 INFO - 泡面学生11 回宿舍吃泡面.

2018-04-01 21:13:21 INFO - 打饭学生7 终于打到了饭.

2018-04-01 21:13:22 INFO - 打饭学生6 终于打到了饭.

2018-04-01 21:13:23 INFO - 打饭学生9 终于打到了饭.

2018-04-01 21:13:24 INFO - 打饭学生8 终于打到了饭.

2018-04-01 21:13:24 INFO - 泡面学生10 终于打到了饭.

2018-04-01 21:13:26 INFO - 泡面学生14 终于打到了饭.

2018-04-01 21:13:26 INFO - 泡面学生12 终于打到了饭.

————————————————

版权声明：本文为CSDN博主「hanchao5272」的原创文章，遵循CC 4.0 BY-SA版权协议，转载请附上原文出处链接及本声明。

原文链接：https://blog.csdn.net/hanchao5272/article/details/79780045