链滴

Java 里的阻塞队列之 DelayQueue

作者：jsy

原文链接：https://ld246.com/article/1482657836086

来源网站：链滴

使用场景

DelayQueue是一个支持延时获取元素的无界阻塞队列。队列使用PriorityQueue来实现。队列中的素必须实现Delayed接口，在创建元素时可以指定多久才能从队列中获取当前元素。只有在延迟期满才能从队列中提取元素。我们可以将DelayQueue运用在以下应用场景：

- 缓存系统的设计：可以用DelayQueue保存缓存元素的有效期，使用一个线程循环查询DelayQueu，一旦能从DelayQueue中获取元素时，表示缓存有效期到了。
- 定时任务调度。使用DelayQueue保存当天将会执行的任务和执行时间，一旦从DelayQueue中获到任务就开始执行，比如TimerQueue就是使用DelayQueue实现的。

- 其中DelayQueue的take方法，该方法获取并移除此队列的头部，在可从此队列获得延迟到期的元之前会一直等待(即阻塞)
- poll()：获取并移除此队列的头，如果此队列不包含具有已到期延迟时间的元素，则 **返回 null**。(阻塞)
- peek()：获取但不移除此队列的头部；如果此队列为空，则返回 null。与poll不同，如果队列中**有到期元素**可用，则此方法**返回下一个将到期的元素**（如果存在一个这样的元素）。
- getDelay()，它可以用来告知延迟到期还有多长时间，或者延迟在多长时间之前已经到期;
- compareTo来指定元素的顺序, 比如让延时时间最长的放在队列的末尾;

```
public int compareTo(Delayed other) {
      if (other == this) // compare zero ONLY if same object
          return 0;
       if (other instanceof ScheduledFutureTask) {
          ScheduledFutureTask x = (ScheduledFutureTask)other;
          long diff = time - x.time;
          if (diff < 0)
              return -1;
          else if (diff > 0)
              return 1;
    else if (sequenceNumber < x.sequenceNumber)
              return -1;
          else
              return 1;
      }
       long d = (getDelay(TimeUnit.NANOSECONDS) -
                 other.getDelay(TimeUnit.NANOSECONDS));
       return (d == 0) ? 0 : ((d < 0) ? -1 : 1);
   }
```

Example

- DelayedTask Code

```
class DelayedTask implements Runnable, Delayed {

    private static int counter = 0;
    private final int id = counter++;
    private final int delayTime;
    private final long triggerTime;
```

```
    public DelayedTask(int delayInMillis) {
        delayTime = delayInMillis;
        triggerTime = System.nanoTime() + NANOSECONDS.convert(delayTime, MILLISECONDS)

    }

    @Override
    public int compareTo(Delayed o) {
        DelayedTask that = (DelayedTask)o;
        if (triggerTime < that.triggerTime) return -1;
        if (triggerTime > that.triggerTime) return 1;
        return 0;
    }

    /**
     * 剩余的延迟时间 */  @Override
    public long getDelay(TimeUnit unit) {
        return unit.convert(triggerTime - System.nanoTime(), NANOSECONDS);
    }

    @Override
    public void run() {
        System.out.println(this + " ");
    }

    @Override
    public String toString() {
        return String.format("[%1$-4d]", delayTime) + " Task " + id;
    }

    public static class EndSentinel extends DelayedTask {
        private ExecutorService exec;
        public EndSentinel(int delay, ExecutorService exec) {
            super(delay);
            this.exec = exec;
        }
        @Override
        public void run() {
            System.out.println(this + " calling shutDownNow()");
            exec.shutdownNow();
        }
    }
}
```

- DelayedTaskConsumer Code

```
class DelayedTaskConsumer implements Runnable {
    private DelayQueue tasks;
    public DelayedTaskConsumer(DelayQueue tasks) {
        this.tasks = tasks;
    }
    @Override
    public void run() {
        try {
```

```
            while(!Thread.interrupted()) {
                tasks.take().run();//run tasks with current thread.
            }
        } catch (InterruptedException e) {
            // TODO: handle exception
    }
        System.out.println("Finished DelaytedTaskConsumer.");
    }
}
```

● Main code

```
public class DelayQueueDemo {

    public static void main(String[] args) {
        int maxDelayTime = 5000;//milliseconds
        Random random = new Random(47);
        ExecutorService exec = Executors.newCachedThreadPool();
        DelayQueue<DelayedTask> queue = new DelayQueue<DelayedTask>();
        //填充10个休眠时间随机的任务
        for (int i = 0; i < 10; i++) {
            queue.put(new DelayedTask(random.nextInt(maxDelayTime)));
        }
        //设置结束的时候。
        queue.add(new DelayedTask.EndSentinel(maxDelayTime, exec));
        exec.execute(new DelayedTaskConsumer(queue));
    }
}
```

- 输出结果

```
[200 ] Task 7
[429 ] Task 5
[555 ] Task 1
[961 ] Task 4
[1207] Task 9
[1693] Task 2
[1861] Task 3
[4258] Task 0
[4522] Task 8
[4868] Task 6
[5000] Task 10 calling shutDownNow()
```

Example2 :模拟一个考试的日子，考试时间为120分钟，30分钟后才可交卷，当时间到了，或学生都交完卷了考试结束。

● 这个场景中几个点需要注意：

1. 考试时间为120分钟，30分钟后才可交卷，初始化考生完成试卷时间最小应为30分钟
2. 对于能够在120分钟内交卷的考生，如何实现这些考生交卷
3. 对于120分钟内没有完成考试的考生，在120分钟考试时间到后需要让他们强制交卷

4. 在所有的考生都交完卷后，需要将控制线程关闭

● 实现思想：用DelayQueue存储考生（Student类），每一个考生都有自己的名字和完成试卷的时，Teacher线程对DelayQueue进行监控，收取完成试卷小于120分钟的学生的试卷。当考试时间120钟到时，先关闭Teacher线程，然后强制DelayQueue中还存在的考生交卷。每一个考生交卷都会进一次countDownLatch.countDown()，当countDownLatch.await()不再阻塞说明所有考生都交完卷，而后结束考试。

● Student code

```
class Student implements Runnable,Delayed {

    private String name;
    private long workTime;
    private long submitTime;
    private boolean isForce = false;
    private CountDownLatch countDownLatch;

    public Student(){}

    public Student(String name,long workTime,CountDownLatch countDownLatch){
        this.name = name;
        this.workTime = workTime;
        this.submitTime = TimeUnit.NANOSECONDS.convert(workTime, TimeUnit.NANOSECON
S)+System.nanoTime();
        this.countDownLatch = countDownLatch;
    }

    @Override
    public int compareTo(Delayed o) {
        // TODO Auto-generated method stub
      if(o == null || ! (o instanceof Student)) return 1;
        if(o == this) return 0;
        Student s = (Student)o;
        if (this.workTime > s.workTime) {
            return 1;
        }else if (this.workTime == s.workTime) {
            return 0;
        }else {
            return -1;
        }
    }

    @Override
    public long getDelay(TimeUnit unit) {
        // TODO Auto-generated method stub
      return unit.convert(submitTime - System.nanoTime(),  TimeUnit.NANOSECONDS);
    }

    @Override
    public void run() {
      // TODO Auto-generated method stub
      if (isForce) {
            System.out.println(name + " 交卷, 希望用时" + workTime + "分钟"+" ,实际用时 120分钟"
```

```
;
        }else {
            System.out.println(name + " 交卷, 希望用时" + workTime + "分钟"+" ,实际用时 "+workT
me +" 分钟");
        }
        countDownLatch.countDown();
    }

    public boolean isForce() {
        return isForce;
    }

    public void setForce(boolean isForce) {
        this.isForce = isForce;
    }

}
```

● Teacher Code

```
class Teacher implements Runnable{

    private DelayQueue students;
    public Teacher(DelayQueue students){
        this.students = students;
    }

    @Override
    public void run() {
        // TODO Auto-generated method stub
        try {
            System.out.println(" test start");
            while(!Thread.interrupted()){
                students.take().run();
            }
        } catch (Exception e) {
            // TODO: handle exception
          e.printStackTrace();
        }
    }

}
```

● EndExam Code

```
class EndExam extends Student{

    private DelayQueue<Student> students;
    private CountDownLatch countDownLatch;
    private Thread teacherThread;

    public EndExam(DelayQueue<Student> students, long workTime, CountDownLatch count
ownLatch,Thread teacherThread) {
        super("强制收卷", workTime,countDownLatch);
```

```
        this.students = students;
        this.countDownLatch = countDownLatch;
        this.teacherThread = teacherThread;
    }

    @Override
    public void run() {
        // TODO Auto-generated method stub

        teacherThread.interrupt();
        Student tmpStudent;
        for (Iterator<Student> iterator2 = students.iterator(); iterator2.hasNext();) {
            tmpStudent = iterator2.next();
            tmpStudent.setForce(true);
            tmpStudent.run();
        }
        countDownLatch.countDown();
    }

}
```

- Main Code

```
public class Exam {

    public static void main(String[] args) throws InterruptedException {
        // TODO Auto-generated method stub
        int studentNumber = 20;
        CountDownLatch countDownLatch = new CountDownLatch(studentNumber+1);
        DelayQueue< Student> students = new DelayQueue();
        Random random = new Random();
        for (int i = 0; i < studentNumber; i++) {
            students.put(new Student("student"+(i+1), 30+random.nextInt(120),countDownLatch))

        }
        Thread teacherThread =new Thread(new Teacher(students));
        students.put(new EndExam(students, 120,countDownLatch,teacherThread));
        teacherThread.start();
        countDownLatch.await();
        System.out.println(" 考试时间到，全部交卷！");
    }

}
```

- 输出结果

```
 test start
student16 交卷, 希望用时37分钟 ,实际用时 37 分钟
student2 交卷, 希望用时39分钟 ,实际用时 39 分钟
student12 交卷, 希望用时40分钟 ,实际用时 40 分钟
student11 交卷, 希望用时47分钟 ,实际用时 47 分钟
student4 交卷, 希望用时53分钟 ,实际用时 53 分钟
student6 交卷, 希望用时55分钟 ,实际用时 55 分钟
student20 交卷, 希望用时74分钟 ,实际用时 74 分钟
```

```
student19 交卷, 希望用时85分钟 ,实际用时 85 分钟
student15 交卷, 希望用时85分钟 ,实际用时 85 分钟
student3 交卷, 希望用时88分钟 ,实际用时 88 分钟
student9 交卷, 希望用时93分钟 ,实际用时 93 分钟
student14 交卷, 希望用时114分钟 ,实际用时 114 分钟
student18 交卷, 希望用时117分钟 ,实际用时 117 分钟
student5 交卷, 希望用时121分钟 ,实际用时 120分钟
student1 交卷, 希望用时122分钟 ,实际用时 120分钟
student13 交卷, 希望用时123分钟 ,实际用时 120分钟
student10 交卷, 希望用时133分钟 ,实际用时 120分钟
student17 交卷, 希望用时148分钟 ,实际用时 120分钟
student8 交卷, 希望用时135分钟 ,实际用时 120分钟
student7 交卷, 希望用时138分钟 ,实际用时 120分钟
 考试时间到，全部交卷!
```

Example 3: 清理过期时间的缓存：向缓存添加内容时，每一个key设定过期时间，系统自动将超过过期时间的ke清除。

● 这个场景中几个点需要注意：

1. 当向缓存中添加key-value对时，如果这个key在缓存中存在并且还没有过期，需要用这个key对应新过期时间

2. 为了能够让DelayQueue将其已保存的key删除，需要重写实现Delayed接口添加到DelayQueue的elayedItem的hashCode函数和equals函数

3. 当缓存关闭，监控程序也应关闭，因而监控线程应当用守护线程

● 具体实现如下：

```
class DelayedItem<T> implements Delayed {

    private******** T t;
    private long liveTime ;
    private long removeTime;

    public DelayedItem(T t,long liveTime){
        this.setT(t);
        this.liveTime = liveTime;
        this.removeTime = TimeUnit.NANOSECONDS.convert(liveTime, TimeUnit.NANOSECOND
) + System.nanoTime();
    }

    @Override
    public int compareTo(Delayed o) {
        if (o == null) return 1;
        if (o == this) return 0;
        if (o instanceof DelayedItem){
            DelayedItem<T> tmpDelayedItem = (DelayedItem<T>)o;
            if (liveTime > tmpDelayedItem.liveTime ) {
                return 1;
            }else if (liveTime == tmpDelayedItem.liveTime) {
```

```
            return 0;
        }else {
            return -1;
        }
    }
    long diff = getDelay(TimeUnit.NANOSECONDS) - o.getDelay(TimeUnit.NANOSECONDS);
    return diff > 0 ? 1:diff == 0? 0:-1;
}

@Override
public long getDelay(TimeUnit unit) {
    return unit.convert(removeTime - System.nanoTime(), unit);
}

public T getT() {
    return t;
}

public void setT(T t) {
    this.t = t;
}

@Override
public int hashCode(){
    return t.hashCode();
}

@Override
public boolean equals(Object object){
    if (object instanceof DelayedItem) {
        return object.hashCode() == hashCode() ?true:false;
    }
    return false;
}

}


public class Cache<K, V> {

    public ConcurrentHashMap<K, V> map = new ConcurrentHashMap<K, V>();
    public DelayQueueK>> queue = new DelayQueueK>>();

    public void put(K k,V v,long liveTime){
        V v2 = map.put(k, v);
        DelayedItem<K> tmpItem = new DelayedItem<K>(k, liveTime);
        if (v2 != null) {
            queue.remove(tmpItem);
        }
        queue.put(tmpItem);
    }

    public Cache(){
        Thread t = new Thread(){
```

```
            @Override
            public void run(){
                dameonCheckOverdueKey();
            }
        };
        t.setDaemon(true);
        t.start();
    }

    public void dameonCheckOverdueKey(){
        while (true) {
            DelayedItem<K> delayedItem = queue.poll();
            if (delayedItem != null) {
                map.remove(delayedItem.getT());
                System.out.println(System.nanoTime()+" remove "+delayedItem.getT() +" from cac
e");
            }
            try {
                Thread.sleep(300);
            } catch (Exception e) {
                // TODO: handle exception
            }
        }
    }

    public static void main(String[] args) throws InterruptedException {
        Random random = new Random();
        int cacheNumber = 10;
        int liveTime = 0;
        Cache cache = new Cache();

        for (int i = 0; i < cacheNumber; i++) {
            liveTime = random.nextInt(3000);
            System.out.println(i+"  "+liveTime);
            cache.put(i+"", i, random.nextInt(liveTime));
            if (random.nextInt(cacheNumber) > 7) {
                liveTime = random.nextInt(3000);
                System.out.println(i+"  "+liveTime);
                cache.put(i+"", i, random.nextInt(liveTime));
            }
        }

        Thread.sleep(3000);
        System.out.println();
    }

}
```

- 输出结果：

```
0  1427
0  2382
1  1520
```

```
2  990
3  2109
3  2726
4  306
5  2166
5  2473
6  231
7  672
8  485
9  2127
1362291709176994 remove 1 from cache
1362292012899275 remove 4 from cache
1362292314446757 remove 6 from cache
1362292619465248 remove 7 from cache
1362292924508940 remove 8 from cache
1362293229379108 remove 9 from cache
1362293533474043 remove 3 from cache
1362293836015945 remove 2 from cache
1362294140880042 remove 0 from cache
```

参考http://www.cnblogs.com/sunzhenchao/p/351508.html

参考https://my.oschina.net/itblog/blog/516670