(143 条消息) 关于 CompletableFuture 因为拒绝策略无限等待的解决思路_Dongguabai 的博客 - CSDN 博客_completedfuture 存在问题

先简答说明一下上下文：

1. 我就是想实现等待多个异步任务都执行完成的操作，同时搜集到每个任务的执行结果，当然可以使用线程池 + `CountDownLauncher` ，但是我个人更倾向于使用 `CompletableFuture` 来实现；
2. 很明显要使用 `CompletableFuture.allOf().join()` 来做；
3. `CompletableFuture` 默认使用的是 ForkJoin 线程池，我个人倾向于自定义线程池；
4. 自定义线程池拒绝策略是肯定要考虑的；

今天这个问题就出在拒绝策略上。这里偷个懒，直接将我之前一篇博客（https://dongguabai.blog.csdn.net/article/details/101145256）中的代码改下模拟我的项目代码：

```
package dongguabai.test.weixin;

import java.util.Date;
import java.util.HashMap;
import java.util.Map;
import java.util.concurrent.ArrayBlockingQueue;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ThreadLocalRandom;
import java.util.concurrent.ThreadPoolExecutor;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.atomic.AtomicInteger;

/**
 * @author Dongguabai
 * @description
 * @date 2021-08-07 13:00
 */
public class Tests2 {
```

```
    private static final AtomicInteger SEQ = new AtomicInteger();

    private static final ThreadPoolExecutor EXECUTOR = new ThreadPoolExecutor(1, 1, 1,
            TimeUnit.MINUTES, new ArrayBlockingQueue<>(1), r -> new Thread(r, "my-thread-" + SEQ.getAr
            , (r, executor) -> {
        throw new RuntimeException("REJECTED.......");
    });

    public static void main(String[] args) {

       try{
           Map<Integer, String> works = new HashMap<>(6);
           works.put(0, "a");
           works.put(1, "b");
           works.put(2, "c");
           works.put(3, "d");
           works.put(4, "e");
           works.put(5, "f");

           //Boolean标识成功或者失败
           Map<Integer, Boolean> resultMapv = new HashMap<>(6);

           System.out.println("Strat." + new Date().toLocaleString());

           CompletableFuture[] array = works.entrySet().stream().map(integerStringEntry ->
                   CompletableFuture
                           .supplyAsync(() -> process(integerStringEntry),EXECUTOR)
                           .whenComplete((result, e) -> {
                               resultMapv.put(result.getKey(),true);
                           })
           ).toArray(CompletableFuture[]::new);
           CompletableFuture.allOf(array).join();
           System.out.println("End."+new Date().toLocaleString());
           System.out.println(resultMapv);
       }finally {
           EXECUTOR.shutdownNow();
       }
    }

    private static Map.Entry<Integer, String> process(Map.Entry<Integer, String> entry) {
        int workingTime = ThreadLocalRandom.current().nextInt(1, 10);
        workTime(workingTime*1000);
        System.out.println(Thread.currentThread().getName() + "完成工作，用时：" + workingTime);
        entry.setValue(entry.getValue() + "_finished");
        return entry;
    }

    private static void workTime(long ms) {
        final long l = System.currentTimeMillis();
        while (System.currentTimeMillis() <= l + ms) {
        }
    }
}
```

我这里线程池线程数设置的是 1，队列也是 1，也就是说同时最多只能进 2 个任务。我这里直接会丢 6 个任务进去。执行一下上面的代码，输出：

```
Strat.2021-8-7 14:22:58
Exception in thread "main" java.lang.RuntimeException: REJECTED.......
        at dongguabai.test.weixin.Tests2.lambda$static$1(Tests2.java:25)
        at java.util.concurrent.ThreadPoolExecutor.reject(ThreadPoolExecutor.java:830)
        at java.util.concurrent.ThreadPoolExecutor.execute(ThreadPoolExecutor.java:1379)
        at java.util.concurrent.CompletableFuture.asyncSupplyStage(CompletableFuture.java:1604)
        at java.util.concurrent.CompletableFuture.supplyAsync(CompletableFuture.java:1830)
        at dongguabai.test.weixin.Tests2.lambda$main$4(Tests2.java:45)
        at java.util.stream.ReferencePipeline$3$1.accept(ReferencePipeline.java:193)
        at java.util.HashMap$EntrySpliterator.forEachRemaining(HashMap.java:1699)
        at java.util.stream.AbstractPipeline.copyInto(AbstractPipeline.java:481)
        at java.util.stream.AbstractPipeline.wrapAndCopyInto(AbstractPipeline.java:471)
        at java.util.stream.AbstractPipeline.evaluate(AbstractPipeline.java:545)
        at java.util.stream.AbstractPipeline.evaluateToArrayNode(AbstractPipeline.java:260)
        at java.util.stream.ReferencePipeline.toArray(ReferencePipeline.java:438)
        at dongguabai.test.weixin.Tests2.main(Tests2.java:49)
my-thread-0完成工作，用时：2
```

同时进程也终止了。

输出结果并不符合我的预期，因为并没有打印出每个任务的执行结果。很明显，不符合预期的原因是因为执行了线程池的拒绝策略，任务都是串行丢给线程池执行的，造成一个任务被拒绝后直接影响了后续任务的执行。

那么就改一下，拒绝的时候不抛出异常：

```
    private static final ThreadPoolExecutor EXECUTOR = new ThreadPoolExecutor(1, 1, 1,
            TimeUnit.MINUTES, new ArrayBlockingQueue<>(1), r -> new Thread(r, "my-thread-" + SEQ.getAr
            , (r, executor) -> {
        //throw new RuntimeException("REJECTED.......");
        System.out.println("REJECTED.......");
    });
```

再执行一遍，结果发现整个主线程直接就卡主了：

这个就很诡异了。要想知道为啥卡主了，就要解决两个问题：

- 线程池的拒绝策略是怎么执行的？
- 线程池的拒绝策略跟 `CompletableFuture` 的执行有什么联系？

先解决第一个问题，查看 `java.util.concurrent.ThreadPoolExecutor#execute` 方法：

```
public void execute(Runnable command) {
        if (command == null)
            throw new NullPointerException();
        int c = ctl.get();
        if (workerCountOf(c) < corePoolSize) {
            if (addWorker(command, true))
                return;
            c = ctl.get();
        }
        if (isRunning(c) && workQueue.offer(command)) {
            int recheck = ctl.get();
            if (! isRunning(recheck) && remove(command))
                reject(command); //执行拒绝策略
            else if (workerCountOf(recheck) == 0)
                addWorker(null, false);
        }
        else if (!addWorker(command, false))
            reject(command); //执行拒绝策略
    }
```

也就是说调用链路就是 `ThreadPoolExecutor#execute` -> `ThreadPoolExecutor#reject` 。如果拒绝策略抛出了异常，会直接导致往线程池提交任务的时候出现异常，这也就解释了最开始为啥当拒绝策略是抛出异常的时候

会导致后续任务无法提交。

再解决第二个问题，线程池的拒绝策略跟 `CompletableFuture` 的执行是怎么联系起来的呢？`CompletableFuture` 的源码非常复杂，抓大放小在，直奔主题，查看 `java.util.concurrent.CompletableFuture#asyncSupplyStage` 方法：

```
    static <U> CompletableFuture<U> asyncSupplyStage(Executor e,
                                                     Supplier<U> f) {
        if (f == null) throw new NullPointerException();
        CompletableFuture<U> d = new CompletableFuture<U>();
        e.execute(new AsyncSupply<U>(d, f));
        return d;
    }
```

即调用链路是 `CompletableFuture#supplyAsync` -> `CompletableFuture#asyncSupplyStage` -> `CompletableFuture#supplyAsync` -> `ThreadPoolExecutor#execute` -> `ThreadPoolExecutor#reject` 。到这里，虽然整个调用链路非常明了了，但还是无法解释为什么会卡主。

发现遗漏了一个细节，程序卡主，是卡在哪了，从第二段代码的输出结果中可以得到两个结论：

- 6 个任务的确是都丢到线程池中执行了；
- 没有打印出"End"；
- 程序卡着，连 `finally` 逗没进去；

也就是说明程序是卡在了 `CompletableFuture.allOf(array).join();` 上。那么就再看看 `CompletableFuture#join` 是怎么玩的，看 `java.util.concurrent.CompletableFuture#waitingGet` 方法，那些细节就不看， 直接看最主要的：

```
private Object waitingGet(boolean interruptible) {
      Signaller q = null;
      boolean queued = false;
      int spins = -1;
      Object r;
      while ((r = result) == null) {
                        ......
         ......
      }
}
```

也就是说 `CompletableFuture` 是根据 `result` 作为标识符来判断任务是否执行完。

到这里，其实为什么程序会" 卡主 "无限等待，就是因为线程池执行了拒绝策略后造成 `CompletableFuture` 无法更新结束标识，从而导致无限 `join` 。

结论：

- 线程池的拒绝策略抛出异常可以阻断无限 `join` ，但是会造成后续串行提交的任务提交也被阻断；
- 像我这样的场景，多个任务串行提交的情况下可以将 `CompletableFuture#supplyAsync` 方法 `catch` 一下；

以这个例子为例，可以这样改造：

```
package dongguabai.test.weixin;

import java.util.Date;
import java.util.HashMap;
import java.util.Map;
import java.util.concurrent.ArrayBlockingQueue;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ThreadLocalRandom;
import java.util.concurrent.ThreadPoolExecutor;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.atomic.AtomicInteger;

/**
 * @author Dongguabai
 * @description
 * @date 2021-08-07 13:00
 */
public class Tests2 {

    private static final AtomicInteger SEQ = new AtomicInteger();

    private static final ThreadPoolExecutor EXECUTOR = new ThreadPoolExecutor(1, 1, 1,
            TimeUnit.MINUTES, new ArrayBlockingQueue<>(1), r -> new Thread(r, "my-thread-" + SEQ.getAr
            , (r, executor) -> {
        throw new RuntimeException("REJECTED.......");
        //System.out.println("REJECTED.......");
    });

    public static void main(String[] args) {

        try {
            Map<Integer, String> works = new HashMap<>(6);
            works.put(0, "a");
            works.put(1, "b");
            works.put(2, "c");
            works.put(3, "d");
            works.put(4, "e");
            works.put(5, "f");

            //Boolean标识成功或者失败
            Map<Integer, Boolean> resultMapv = new HashMap<>(6);

            System.out.println("Strat." + new Date().toLocaleString());

            CompletableFuture[] array = works.entrySet().stream().map(integerStringEntry -> {
```

```
                        try {
                            return CompletableFuture
                                    .supplyAsync(() -> process(integerStringEntry), EXECUTOR)
                                    .whenComplete((result, e) -> {
                                        resultMapv.put(result.getKey(), true);
                                    });
                        } catch (Exception e) {
                            resultMapv.put(integerStringEntry.getKey(), false);
                            return CompletableFuture.completedFuture(new Object());
                        }
                    }
            ).toArray(CompletableFuture[]::new);
            CompletableFuture.allOf(array).join();
            System.out.println("End." + new Date().toLocaleString());
            System.out.println(resultMapv);
        } finally {
            EXECUTOR.shutdownNow();
        }
    }

    private static Map.Entry<Integer, String> process(Map.Entry<Integer, String> entry) {
        int workingTime = ThreadLocalRandom.current().nextInt(1, 10);
        workTime(workingTime * 1000);
        System.out.println(Thread.currentThread().getName() + "完成工作，用时：" + workingTime);
        entry.setValue(entry.getValue() + "_finished");
        return entry;
    }

    private static void workTime(long ms) {
        final long l = System.currentTimeMillis();
        while (System.currentTimeMillis() <= l + ms) {
        }
    }
}
```

输出结果：

```
Strat.2021-8-7 15:07:10
my-thread-0完成工作，用时：1
my-thread-0完成工作，用时：9
End.2021-8-7 15:07:20
{0=true, 1=true, 2=false, 3=false, 4=false, 5=false}
```

References

- https://dongguabai.blog.csdn.net/article/details/101145256
- https://blog.csdn.net/Dongguabai/article/details/110338023

欢迎关注公众号：

全文完