分享点Java编程的东东

《java.util.concurrent 包源码阅读》16 一种特别的BlockingQueue：SynchronousQueue

SynchronousQueue是一种很特别的BlockingQueue，任何一个添加元素的操作都必须等到另外一个线程拿走元素才会结束。也就是SynchronousQueue本身不会存储任何元素，相当于生产者和消费者手递手直接交易。

SynchronousQueue有一个fair选项，如果fair为true，称为fair模式，否则就是unfair模式。

在fair模式下，所有等待的生产者线程或者消费者线程会按照开始等待时间依次排队，然后按照等待先后顺序进行匹配交易。这种情况用队列实现。

在unfair模式下，则刚好相反，后来先匹配，这种情况用栈实现。

```
     */
    public SynchronousQueue(boolean fair) {
        transferer = fair ? new TransferQueue() : new TransferStack();
    }
```

因为添加元素和拿走元素是类似手递手交易的，所以对于拿走元素和添加元素操作，SynchronousQueue调用的是Transferer同一个方法transfer。

当object为null时表示是拿走元素，用于消费者线程，否则则是添加元素，用于生产者线程。因此transfer方法是分析的重点。

```
abstract Object transfer(Object e, boolean timed, long nanos);
```

首先来看用于fair模式的TransferQueue的transfer方法：

看代码之前，来理一下逻辑：

1. 开始队列肯定是空。

2. 线程进入队列，如果队列是空的，那么就添加该线程进入队列，然后进行等待（要么有匹配线程出现，要么就是该请求超时取消）

3. 第二个线程进入，如果前面一个线程跟它属于不同类型，也就是说两者是可以匹配的，那么就从队列删除第一个线程。

    如果是相同的线程，那么做法参照2。

理清了基本逻辑，也就是会有两种情况：

1. 队列为空或者队列中的等待线程是相同类型

2. 队列中的等待线程是匹配的类型

```
        Object transfer(Object e, boolean timed, long nanos) {

            QNode s = null;
            // e不是null表示是生成者线程，e就是产品，反之就是消费者线程
            boolean isData = (e != null);

            for (;;) {
                QNode t = tail;
                QNode h = head;
                // tail和head在队列创建时会被初始化成一个虚拟节点
                // 因此发现没有初始化，重新循环等待直到初始化完成
```

```
            if (t == null || h == null)
                continue;

            // 队列为空或等待线程类型相同（不同类型才能匹配）
            // 这两种情况都要把当前线程加入到等待队列中
            if (h == t || t.isData == isData) {
                QNode tn = t.next;
                // tail对象已经被更新，出现不一致读的现象，重新循环
                if (t != tail)
                    continue;
                // 添加线程到等待队列时会先更新当前tail的next，然后
                // 更新tail本身，因此出现只有next被更新的情况，应该
                // 更新tail，然后重新循环
                if (tn != null) {
                    advanceTail(t, tn);
                    continue;
                }
                // 设定了超时，剩余等待时间耗尽的时候，就无需再等待
                if (timed && nanos <= 0)
                    return null;
                // 首次使用s的时候，新建一个节点保存当前线程和数据来初始化s
                if (s == null)
                    s = new QNode(e, isData);
                // 尝试更新tail的next，把新建节点添加到tail的后面，如果失败了，就重新循环
                if (!t.casNext(null, s))
                    continue;
                // 把新建的节点设置为tail
                advanceTail(t, s);
                // 等待匹配线程，成功匹配则返回的匹配的值
                // 否则返回当前节点，因此s和x相同表示请求被取消
                Object x = awaitFulfill(s, e, timed, nanos);
                if (x == s) {
                    clean(t, s);
                    return null;
                }

                // 这个时候已经匹配成功了，s应该是排在第一个的等待线程
                // 如果s依然在队列中，那么需要更新head。
                // 更新head的方法是把s这个排在第一位的节点作为新的head
                // 因此需要重置一些属性使它变成虚拟节点
                if (!s.isOffList()) {
                    advanceHead(t, s);
                    if (x != null)
                        s.item = s;
                    s.waiter = null;
                }
                // x不为null表示拿到匹配线程的数据（消费者拿到生产者的数据），
                // 因此返回该数据，否则返回本身的数据（生成者返回自己的数据）
                return (x != null) ? x : e;

            } else { // 线程可以匹配
                // 因为是队列，因此匹配的是第一个节点
                QNode m = h.next;
                // 同样需要检查不一致读的情况
                if (t != tail || m == null || h != head)
                    continue;

                Object x = m.item;
                // 匹配失败时，把m从队列中移走，重新循环
                if (isData == (x != null) ||    // m已经被匹配了
                    x == m ||                   // m已经被取消了
                    !m.casItem(x, e)) {         // 用CAS设置m的数据为null
                    advanceHead(h, m);
                    continue;
                }
```

Java通用服务器程序(3)
深入实践Java并发包

随笔档案

2014年10月 (3)
2014年9月 (10)
2014年8月 (17)
2014年7月 (2)
2013年6月 (1)
2012年12月 (1)

积分与排名

积分 - 18024
排名 - 13847

最新评论

1. Re:《用Java写一个通用的服务器程序》03 处理新socket
我想问一下为什么服务器启动失败
--网易吗

2. Re:《用Java写一个通用的服务器程序》01 综述
@archy_yu你有源代码吗？先看看是不是代码的问题。还有看看阿里云是不是对于连接有没有限制？...
--梧留柒

3. Re:《用Java写一个通用的服务器程序》01 综述
问博主一个http服务器的例子！为什么我用mina或者Netty写的httpServer的例子，qps都很差，在阿里云上测试每秒也就处理 30个左右的http请求！用的ab 测试的？求指导...
--archy_yu

4. Re:《用Java写一个通用的服务器程序》02 监听器
@CloudHi有啊，在github上:...
--梧留柒

5. Re:《用Java写一个通用的服务器程序》02 监听器
有没有完整的demo、源码
--CloudHi

阅读排行榜

1. 《java.util.concurrent 包源码阅读》01 源码包的结构(1442)
2. 《java.util.concurrent 包源码阅读》 结束语(1233)
3. 《java.util.concurrent 包源码阅读》02 关于java.util.concurrent.atomic包(989)
4. 《用Java写一个通用的服务器程序》01 综述(975)
5. 《java.util.concurrent 包源码阅读》22 Fork/Join框架的初体验(965)

评论排行榜

1. 《用Java写一个通用的服务器程序》01 综述(6)
2. 《java.util.concurrent 包源码阅读》 结束语(6)
3. 《用Java写一个通用的服务器程序》02 监听器(3)
4. 《java.util.concurrent 包源码阅读》01 源码包的结构(2)
5. 《java.util.concurrent 包源码阅读》04 ConcurrentMap(2)

推荐排行榜

1. 《用Java写一个通用的服务器程序》02 监听器(4)
2. 《java.util.concurrent 包源码阅读》 结束语(3)

3. 《java.util.concurrent 包源码阅读》27 Phaser 第一部分(2)
4. 《java.util.concurrent 包源码阅读》24 Fork/Join框架之Work-Stealing(1)
5. 《java.util.concurrent 包源码阅读》22 Fork/Join框架的初体验(1)

```
                    // 匹配成功，更新head
                    advanceHead(h, m);
                    // 解除m的线程等待状态
                    LockSupport.unpark(m.waiter);
                    // 返回匹配的数据
                    return (x != null) ? x : e;
                }
            }
        }
```

接着来用于Unfair模式的TransferStack的transfer方法

大体逻辑应该是一样的，不同就是队列的入队和出队操作对应到栈时就是入栈和出栈的操作。

```
        Object transfer(Object e, boolean timed, long nanos) {
            SNode s = null;
            int mode = (e == null) ? REQUEST : DATA;

            for (;;) {
                SNode h = head;
                // 栈为空或者节点类型相同的情况
                if (h == null || h.mode == mode) {
                    if (timed && nanos <= 0) {
                        // 检查栈顶节点是否已经取消，如果已经取消，弹出节点
                        // 重新循环，接着检查新的栈顶节点
                        if (h != null && h.isCancelled())
                            casHead(h, h.next);
                        else
                            return null;
                    // 新建节点，并且尝试把新节点入栈
                    } else if (casHead(h, s = snode(s, e, h, mode))) {
                        // 等待匹配，如果发现是被取消的情况，则释放节点，返回null
                        SNode m = awaitFulfill(s, timed, nanos);
                        if (m == s) {
                            clean(s);
                            return null;
                        }
                        // 如果匹配的成功两个节点是栈顶的两个节点
                        // 把这两个节点都弹出
                        if ((h = head) != null && h.next == s)
                            casHead(h, s.next);     // help s's fulfiller
                        return (mode == REQUEST) ? m.item : s.item;
                    }
                } else if (!isFulfilling(h.mode)) { // 栈顶节点没有和其他线程在匹配，可以匹配
                    if (h.isCancelled())            // 栈顶节点的请求已经被取消
                        casHead(h, h.next);         // 移除栈顶元素重新循环
                    // 尝试把该节点也入栈，该节点设置为正在匹配的状态
                    // 也就是isFulfilling返回true
                    else if (casHead(h, s=snode(s, e, h, FULFILLING|mode))) {
                        for (;;) {
                            // 栈顶节点（当前线程的节点）和它的下一个节点进行匹配，m为null意味着
                            // 栈里没有其他节点了，因为前面该节点入栈了，需要弹出这个节点重新循环
                            SNode m = s.next;
                            if (m == null) {
                                casHead(s, null);
                                s = null;
                                break;
                            }

                            // 这个时候是有节点可以匹配的，尝试为这两个节点做匹配
                            SNode mn = m.next;
```

```
                        // m和s匹配成功，弹出这两个节点，返回数据；匹配失败，把m移除
                        if (m.tryMatch(s)) {
                            casHead(s, mn);
                            return (mode == REQUEST) ? m.item : s.item;
                        } else
                            s.casNext(m, mn);
                    }
                }
            // 栈顶正在匹配，参见代码：
            // else if (casHead(h, s=snode(s, e, h, FULFILLING|mode))) {
            // 做法基本类似，只是这里帮助其他线程匹配，无论成功与否
            // 都要重新循环
            } else {
                SNode m = h.next;
                if (m == null)
                    casHead(h, null);
                else {
                    SNode mn = m.next;
                    if (m.tryMatch(h))
                        casHead(h, mn);
                    else
                        h.casNext(m, mn);
                }
            }
        }
    }
```

TransferQueue和TransferStack的算法实现可以参考 这里

分类: java.util.concurrent包源码分析

标签: java concurrent

梧留柒
关注 - 0
粉丝 - 66
+加关注

1 0

« 上一篇：《java.util.concurrent 包源码阅读》15 线程池系列之ScheduledThreadPoolExecutor 第二部分
» 下一篇：《java.util.concurrent 包源码阅读》17 信号量 Semaphore

posted on 2014-08-25 15:21 梧留柒 阅读(759) 评论(0) 编辑 收藏