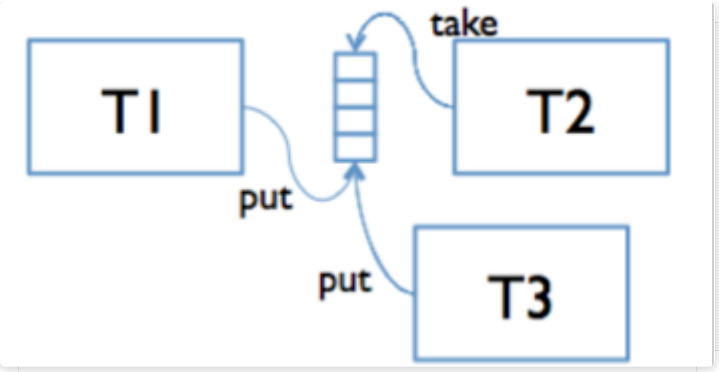

从一个故障说说Java的三个BlockingQueue

最近出了个故障，排查的时候耗费了很长的时间，回顾整个排查过程，经验主义在这里起了不好的作用，直接导致了整个故障排查的时间非常长，这个故障的根本原因在于BlockingQueue用的有问题，顺带展开说说Java中常用的几个BlockingQueue：ArrayBlockingQueue、LinkedBlockingQueue和SynchronousQueue。

收获：仔细看日志信息来反追问题

当时故障的现象是应用处理请求的线程池满了，导致请求处理不了，于是dump线程，看线程都在做什么，结果发现线程都Block在写日志的地方，以前出现过很多次问题，去线程dump的时候看到也是一堆的block在写日志，但通常是别的原因引发的，所以这次也是按照这样的经验，认为肯定不会是写日志这个地方的问题，于是各种排查...折腾了N久后，回过头看发现持有那把日志锁的地方是自己人写的代码，那段代码在拿到了这个日志锁后，从线程堆栈上看，block在了ArrayBlockingQueue.put这个地方，于是翻看这段代码，结果发现这是个1024长度的BlockingQueue，那就意味着如果这个Queue被放了1024个对象的话，put就一定会被block住，而且其实翻代码的时候能看出写代码的同学是考虑到了BlockingQueue如果满了应该要处理的，代码里写着：

经验主义

```
if (blockingQueue.remainingCapacity() < 1) { //todo } blockingQueue.put
```

这里两个悲催的问题，一是这个if判断完还是直接会走到put，而不是else，二是竟然关键的满了后的处理逻辑还在//todo...

另外我觉得这段代码还反应了同学对BlockingQueue的接口不太熟，要达到这个效果，不需要这样先去判断，更合适的做法是用blockingQueue.offer，返回false再做相应的异常处理。

BlockingQueue是在生产/消费者模式下经常会用到的数据结构，通常常用的主要会是ArrayBlockingQueue、LinkedBlockingQueue和SynchronousQueue。

ArrayBlockingQeue/LinkedBlockingQueue两者的最大不同主要在于存放Queue中对象方式，一个是数组，一个是链表，代码注释里也写到了两者的不同：

Linked queues typically have higher throughput than array-based queues but less predictable performance in most concurrent applications.

链表队列，通常有更高的吞吐量；
数组队列，在大多数并发应用中有可预测的性能

同步队列

SynchronousQueue是一个非常特殊的BlockingQueue，它的模式是在offer的时候，如果没有另外一个线程正在take或poll的话，那么offer就会失败；在take的时候，如果没有另外的线程正好并发在offer，也会失败，这种特殊的模式非常适合用来做要求高响应并且线程出不固定的线程池的Queue。

线程数不固定

1. 超时机制，fail fast

对于在线业务场景而言，所有的并发，外部访问阻塞的地方的一个真理就是一定要有超时机制，我不知道见过多少次由于没有超时造成的在线业务的严重故障，在线业务最强调的是快速处理掉一次请求，所以fail fast是在线业务系统设计，代码编写中的最重要原则，按照这个原则上面的代码最起码明显犯的错误就是用put而不是带超时机制的offer，或者说如果是不重要的场景，完全就应该直接用offer，false了直接抛异常或记录下异常即可。

2. 队列长度

对于BlockingQueue这种场景呢，除了超时机制外，还有一个是队列长度一定要做限制，否则默认的是Integer.MAX_VALUE，万一代码出点bug的话，内存就被玩挂了。

说到BlockingQueue，就还是要提下BlockingQueue被用的最多的地方：线程池，Java的ThreadPoolExecutor中有个参数是BlockingQueue，如果这个地方用的是ArrayBlockingQueue或LinkedBlockingQueue，而线程池的coreSize和poolSize不一样的话，在coreSize线程满了后，这个时候线程池首先会做的是offer到BlockingQueue，成功的话就结束，这种场景同样不符合在线业务的需求，在线业务更希望的是快速处理，而不是先排队，而且其实在线业务最好是不要让请求堆在排队队列里，在线业务这样做很容易引发雪崩，超出处理能力范围直接拒绝抛错是相对比较好的做法，至于在前面页面上排队什么这个是可以的，那是另外一种限流机制。

收获

前端页面

限流机制

所以说在写高并发、分布式的代码时，除了系统设计外，代码细节的功力是非常非常重要的。

多看看 JDK、Netty 的相关使用代码

This entry was posted in Java and tagged BlockingQueue, ThreadPoolExecutor on 2016-05-04.

收获：在线业务更希望的是快速处理，而不是先排队，而且其实在线业务最好是不要让请求堆在排队队列里，在线业务这样做很容易引发雪崩，超出处理能力范围直接拒绝抛错是相对比较好的做法。