疑问问题

1.如何保证资源池线程安全

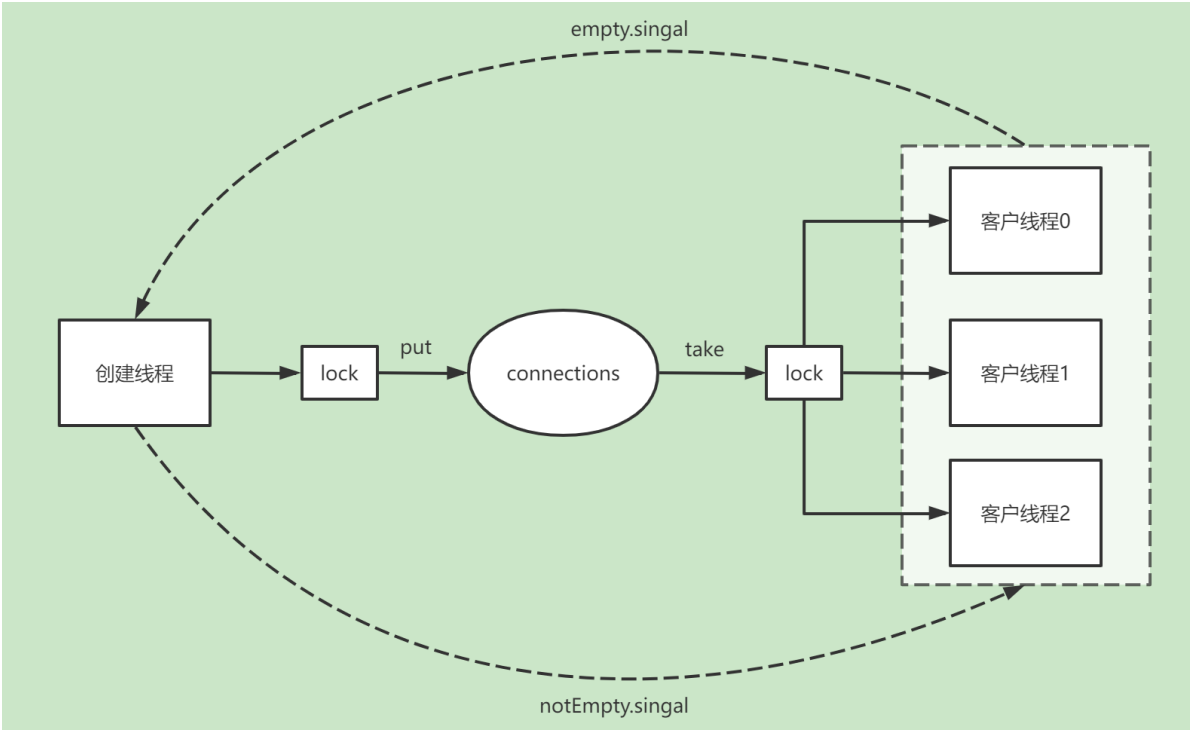
2.如何让资源池能够应付高并发场景

通信模型

前面两篇分析了createThread与客户线程的代码步骤和FSM，现在可以来分析这两者之间的通信模型了（注：为了方便分析，仅摘取对本次分析有用的内容）。这里按照调用顺序给出两者主要的通信过程。

CreateThread(循环以下过程)	客户线程（讨论takeLast()）
当符合条件时等待通知（`empty.await()`）	无连接可用，通知创建线程（`emptySignal()`）
收到通知被唤醒（其他线程调用`empty.signal()` or `empty.signalAll()`）	进入wait状态（`notEmpty.await()`）
当符合条件时创建连接（否则从头循环）	收到通知被唤醒（其他线程调用`notEmpty.signal()` or `notEmpty.signalAll()`）
创建连接，将连接放入到池子中，唤醒客户线程（`notEmpty.signal()`）	获取连接

不难看出这两个线程之间依赖于Lock实例lock及其两个Condition实例（empty、notEmpty）来通信。更加准确来说，通过lock实例来保证连接池（即资源）能够被安全地修改数据，其次是通过【资源池正确的信息和两个信号量（empty、notEmpty）】来达到线程之间传递信息和作出相应行为的目的。这样子的通信模型我联想到得是【生产者消费者模型】。如下图



通过上面的信息，我可以认为这个【生产者消费者模型】是基于信号量来实现的。

一个额外的知识点是，在createThread在创建物理线程的时候，这个时候createThread是异步的，createThread不会响应客户线程发来的通知（即不响应`empty.signal()`）。所以我当时有个疑问：“createThread创建物理线程时只会创建一个，如果此时有多个客户线程发来通知，岂不是会让这些通知失效并且创建的连接数与等待的客户线程数匹配不上吗？”。这个疑问在以下代码中得到了回复：

```
public class CreateConnectionThread extends Thread {
    public void run() {
        for (;;) {
            /*省略部分代码*/
            if (emptyWait) {
                // 必须存在线程等待，才创建连接
                if (poolingCount >= notEmptyWaitThreadCount // 每次循环到此判断，根据notEmptyWaitThreadCount与poolingCount就可以判断
                                                            // 可用连接数与当前等待连接的客户线程数。因此也解决了上面所说的疑问。
                    && (!(keepAlive && activeCount + poolingCount < minIdle))
                    && !isFailContinuous()
                   ) {
                    empty.await();
                }

                // 防止创建超过maxActive数量的连接
                if (activeCount + poolingCount >= maxActive) {
                    empty.await();
                    continue;
                }
            }
            /*省略部分代码*/
        }
    }
}
```

阶段总结

通过这几篇的学习可以回答疑问1“如何保证资源池线程安全”。至于疑问2“如何让资源池能够应付高并发场景”，目前只能回答部分“是因为利用了【生产者消费者模型】的优点来提高并发性”，由于对lock和condition没有很深入的了解，猜测“或许采用lock和condition可以在JVM内部来管理线程，无需陷入到内核来进行上下文切换”。往后有机会会将这一块空缺补上。

新的疑问

在Demo（ConcurrentTest2）中，作者写了个测试类，该类大概的行为是：使用50个线程，每个线程轮询10000次。轮询时执行的任务是从Druid中获取连接然后马上释放掉，然后最长等待10秒。当等待的线程数目达到10个时，通知Druid调用shrink()方法。

每次执行的结果是总是存在等待超时的现象。

而我所疑问的是，仅仅是通过这个测试类（无对比类，无量化标准），如何判断Druid才算是“高性能”呢？判断的标准是什么呢？