疑问问题

今天继续研究池化技术。昨天学习用定制化参数来影响池化技术的行为，那么在面对多线程从资源池中获取资源，怎么能够做到：

1.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

因此我想了解Druid是怎么决定锁的粒度（即临界区）、客户线程和创建线程之间如何进行通信配合等。

复习前面的内容，知道Druid在DruidDataSource中用一个可重入锁来保护核心资源。即每当需要修改核心资源前必须要先获取锁。这里“修改核心资源”的操作包括了从资源池获取连接、将连接放入到资源池中、在资源池中移除无效连接等等。根据第一天的代码运行Druid，可知Druid在运行期间有一个创建连接的线程。

CreateConnectionThread

代码步骤分析

```
public class CreateConnectionThread extends Thread {
    public void run() {
        // 代码里面只讨论创建连接和上锁的位置，忽略了其他代码
         for (;;) {
            try {
                lock.lockInterruptibly(); // 第一次获取锁
            } catch (InterruptedException e2) {
                break;
            }
            try {
                if (emptyWait) {
                    // 必须存在线程等待，才创建连接
                    empty.await(); // 进入等待。即只有当连接池无可用连接时才创建
                }
            } finally {
               lock.unlock(); // 不管是遇到异常还是要准备创建物理连接，都释放掉锁。
            }

            // 这里是真正数据库连接的地方
            PhysicalConnectionInfo connection = createPhysicalConnection()

            put(connection);// 将数据库连接放入到资源池中
            /* 以下是放入到资源池的部分代码
               lock.lock(); // 先获取锁
               connections[poolingCount] = holder; // 将数据库连接放入池子
               incrementPoolingCount(); // 可用连接数+1

               notEmpty.signal(); // 通知在notEmpty上等待的线程，即告诉客户线程“有可用连接了”
            /*
        }
    }
}
```

FSM

下面是创建连接线程在上述代码中的有限状态机：