# 美团一面，面试官让介绍AQS原理并手写一个同步器，直接凉了

## 写在开头

今天在牛客上看到了一个帖子，一个网友吐槽美团一面上来就让手撕同步器，没整出来，结果面试直接凉凉。

就此联想到一周前写的一篇关于AQS知识点解析的博文，当时也曾埋下伏笔说后面会根据AQS的原理实现一个自定义的同步器，那今天就来把这个坑给填上哈。

## 常用的AQS架构同步器类

### 自定义同步器实现步骤

在上一篇文章中我们就已经提过了AQS是基于 **`模版方法模式`** 的，我们基于此的自定义同步器设计一般需要如下两步：

**1. 使用者继承 AbstractQueuedSynchronizer 并重写指定的方法；**
**2. 将 AQS 组合在自定义同步组件的实现中，并调用其模板方法，而这些模板方法会调用使用者重写的方法。**

在模版方法模式下，有个很重要的东西，那就是`“钩子方法”`，这是一种抽象类中的方法，一般使用 protected 关键字修饰，可以给与默认实现，空方法居多，其内容逻辑由子类实现，为什么不适用抽象方法呢？因为，抽象方法需要子类全部实现，增加大量代码冗余！

Ok，有了这层理论知识，我们去看看Java中根据AQS实现的同步工具类有哪些吧

### Semaphore(信号量)

在前面我们讲过的synchronized 和 ReentrantLock 都是一次只允许一个线程访问某个资源，而Semaphore(信号量)可以用来控制同时访问特定资源的线程数量，它并不能保证线程安全。

我们下面来看一个关于Semaphore的使用示例：

**【代码示例1】**

```
public class Test {
    private final Semaphore semaphore;

    /**
     * 构造方法初始化信号量
     * @param limit
     */
    public Test(int limit) {
        this.semaphore = new Semaphore(limit);
    }

    public void useResource() {
        try {
            semaphore.acquire();
            // 使用资源
            System.out.println("资源use:" +
Thread.currentThread().getName());
            Thread.sleep(1000); // 模拟资源使用时间
        } catch (InterruptedException e) {
            e.printStackTrace();
        } finally {
            semaphore.release();
            System.out.println("资源release:" +
Thread.currentThread().getName());
        }
    }

    public static void main(String[] args) {
        // 限制3个线程同时访问资源
        Test pool = new Test(3);

        for (int i = 0; i < 4; i++) {
            new Thread(pool::useResource).start();
```

```
        }
    }
}
```

**输出：**

```
资源use:Thread-1
资源use:Thread-0
资源use:Thread-2
资源release:Thread-0
资源release:Thread-1
资源release:Thread-2
资源use:Thread-3
资源release:Thread-3
```

由此结果可看出，我们成功的将同时访问共享资源的线程数限制在了不超过3个的级别，这里面涉及到了Semaphore的两个主要方法：`acquire()和release()`

**① acquire()**:获取许可
跟进这个方法后，我们会发现其内部调用了AQS的一个final 方法acquireSharedInterruptibly()，这个方法中又调用了tryAcquireShared(arg)放，作为AQS中的钩子方法，这个方法的实现在Semaphore的两个静态内部类 **FairSync（公平模式）** 和 **NonfairSync（非公平模式）** 中。


(https://p3-juejin.byteimg.com/tos-cn-i-k3u1fbpfcp/812923e4f34845eaa44035328ca4d070~tplv-k3u1fbpfcp-jj-mark:3024:0:0:0:q75.awebp#?w=901&h=228&s=33248&e=png&b=4d4841)

**② release()**:释放许可
同样跟入这个方法，里面用了AQS的releaseShared()，而在这个方法内也毫无疑问的用了tryReleaseShared(int arg)这个钩子方法，原理同上，不再冗释。

**【补充】**
此外，在Semaphore中还有一个Sync的内部类，提供nonfairTryAcquireShared()自旋获取资源，以及tryReleaseShared(int releases)，共享方式尝试释放资源。

除了Semaphore(信号量)外，基于AQS实现的还有CountDownLatch （倒计时器）、CyclicBarrier(循环栅栏)，本来想在一篇文章中讲完的，但感觉篇幅上会非常长，遂放弃，后面分篇学习吧。

手写一个同步器!
---

好了，有了上面的一系列学习，我们现在来手撕一个自定义的同步器吧，原理都一样滴，开始前，先贴上AQS中的几个钩子方法，防止待会忘记，哈哈!

**【钩子方法】**

```
//独占方式。尝试获取资源，成功则返回true，失败则返回false。
protected boolean tryAcquire(int)
//独占方式。尝试释放资源，成功则返回true，失败则返回false。
protected boolean tryRelease(int)
//共享方式。尝试获取资源。负数表示失败；0表示成功，但没有剩余可用资源；正数表示成功，且有剩余资源。
protected int tryAcquireShared(int)
//共享方式。尝试释放资源，成功则返回true，失败则返回false。
protected boolean tryReleaseShared(int)
//该线程是否正在独占资源。只有用到condition才需要去实现它。
protected boolean isHeldExclusively()
```

> ***写一个基于AQS的互斥锁，同一时刻只允许一个线程获取资源。***

**步骤一：**

首先，我们在第一步，我们定义一个互斥锁类OnlySyncByAQS，在类中我们同样写一个静态内部类去继承AbstractQueuedSynchronizer，在内部类中，我们重写AQS的tryAcquire方法，独占方式，尝试获取资源；重写tryRelease()尝试释放资源，这俩为主要方法!

然后我们再进一步封装成lock()与unlock()的上锁与解锁方法，并在里面通过模版方法模式，去调用AQS中的acquire()和release()，从而去调到我们对模版方法的实现。

**【代码示例2】**

```
public class OnlySyncByAQS {

    private final Sync sync = new Sync();

    /**
     * 获取许可，给资源上锁
     */
    public void lock() {
        sync.acquire(1);
    }

    /**
     * 释放许可，解锁
     */
    public void unlock() {
        sync.release(1);
    }

    /**
     * 判断是否独占
     * @return
     */
    public boolean isLocked() {
        return sync.isHeldExclusively();
    }

    /**
     * 静态内部类，继承AQS，重写钩子方法
     */
    private static class Sync extends AbstractQueuedSynchronizer {

        /**
         * 重写AQS的tryAcquire方法，独占方式，尝试获取资源。
         */
        @Override
        protected boolean tryAcquire(int arg) {
            //CAS 尝试更改状态
            if (compareAndSetState(0, 1)) {
                //独占模式下，设置锁的持有者为当前线程，来自于AOS
                setExclusiveOwnerThread(Thread.currentThread());
                System.out.println(Thread.currentThread().getName()+”获取锁
```

```
成功”);
            return true;
        }
        System.out.println(Thread.currentThread().getName()+”获取锁失
败”);
        return false;
    }

    /**
     * 独占方式。尝试释放资源，成功则返回true，失败则返回false。
     * @param arg
     * @return
     */
    @Override
    protected boolean tryRelease(int arg) {
        if (getState() == 0) {
            throw new IllegalMonitorStateException();
        }
        //置空锁的持有者
        setExclusiveOwnerThread(null);
        //改状态为0，未锁定状态
        setState(0);
        System.out.println(Thread.currentThread().getName()+”释放锁成
功！”);
        return true;
    }

    /**
     * 判断该线程是否正在独占资源，返回state=1
     * @return
     */
    @Override
    protected boolean isHeldExclusively() {
        return getState() == 1;
    }
}

}
```

**步骤二：**

第二步，我们写一个测试类去调用这个自定义的互斥锁。

**【代码示例2】**

```
public class Test {

    private OnlySyncByAQS onlySyncByAQS = new OnlySyncByAQS();

    public void use(){
        onlySyncByAQS.lock();
        try {
            //休眠1秒获取使用共享资源
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        } finally {
            onlySyncByAQS.unlock();
        }
    }

    public static void main(String[] args) throws InterruptedException {
        Test test = new Test();
        //多线程竞争资源，每次仅一个线程拿到锁
        for (int i = 0; i < 3; i++) {
            new Thread(()->{
                test.use();
            }).start();
        }
    }
}

```

**输出：**

```
Thread-0获取锁成功
Thread-1获取锁失败
Thread-2获取锁失败
Thread-1获取锁失败
Thread-1获取锁失败
Thread-0释放锁成功!
Thread-1获取锁成功
Thread-1释放锁成功!
Thread-2获取锁成功
Thread-2释放锁成功!
```

```

由输出结果可以看出作为互斥锁，每次仅一个线程可以获取到锁资源，其他线程会不断尝试获取并失败，直至该线程释放锁资源！

结尾彩蛋
---

如果本篇博客对您有一定的帮助，大家记得**留言+点赞+收藏**呀。原创不易，转载请联系Build哥！



(https://p3-juejin.byteimg.com/tos-cn-i-k3u1fbpfcp/a1e71931afbd454ba334d3eb1f391f80~tplv-k3u1fbpfcp-jj-mark:3024:0:0:0:q75.awebp#?w=260&h=202&s=37960&e=png&b=fcfafa)
如果您想与Build哥的关系更近一步，还可以“JavaBuild888”，在这里除了看到《Java成长计划》系列博文，还有提升工作效率的小笔记、读书心得、大厂面经、人生感悟等等，欢迎您的加入！



(https://p3-juejin.byteimg.com/tos-cn-i-k3u1fbpfcp/5a5663a4f69e42b4a3146dc419c4d786~tplv-k3u1fbpfcp-jj-mark:3024:0:0:0:q75.awebp#?w=2394&h=259&s=56024&e=png&b=fefefe)

原文链接: https://juejin.cn/post/7356055073585774643
```