# 千变万化的锁

# Lock接口

## 简介、地位、作用

### 1.作用

1. 锁是一种工具，用于控制对共享资源的访问。
2. Lock和Synchronized，这两种是最常见的锁，他们都可以达到线程安全的目的，但是在使用上和功能上又有较大的不同。
3. Lock并不是用来替代synchronized的，而是当使用synchronized不合适或不足以满足要求的时间，来提供高级功能的。
4. lock接口最常见的实现类是ReentrantLock。
5. 通常情况下，Lock只允许一个线程来访问这个共享资源，不过有的时候，一些特殊的实现也可允许并发访问，比如ReadWriteLock里面的ReadLock。

## 为什么synchronized不够用？为什么需要Lock？

1. 为什么Synchronized不够用？
   1. 效率低：锁的释放情况少、试图获得锁时不能设定超时、不能中断一个正在试图获得锁的线程
   2. 不够灵活(读写锁更灵活)：加锁和释放的时机单一，每个锁仅有单一的条件(某个对象)，可能是不够的
   3. 无法知道是否成功获取到锁

## 方法介绍

1.在Lock中声明了四个方法来获取锁

2.lock()、tryLock()、tryLock(long time,TimeUnit unit)和lockInterruptibly()

3.那么这四个方法有什么区别呢？

1.lock()

1.lock()就是最普通的获取锁。如果锁已被其他线程获取，则进行等待。

2.Lock不会像synchronized一样在异常时自动释放锁。

3.因此最佳实践是，在finally中释放锁，以保证发生异常时锁一定被释放

4.lock()方法不能被中断，这会带来很大的隐患：一旦陷入死锁，lock()就会陷入永久等待

2.tryLock()

1.tryLock()用来尝试获取锁，如果当前锁没有被其他线程占用，则获取成功。则返回true，否则返回false，代表获 取锁失败

2.相比于lock，这样的方法显然功能更强大了，我们可以根据是否能获取到锁来决定后续程序的行为。

3.该方法会立刻返回，即便在拿不到锁时不会一直在那等

3.tryLock(long time,TimeUnit unit)：超时就放弃(代码演示，代码在ThreadLock.src.lock.lock.TryLockDeadLock)。

4.lockInterruptibly()；相当于tryLock(long time,TimeUnit unit)把超时时间设置为无限。在等待死锁的过程中，线程可以被中断

5.unlock()：解锁

## 可见性保证

### 可见性

可见性是java内存模型里面最重要的一个部分，他指的是我们线程与线程之间，他们并不是随时可以看到对方最新的动态的，比如说我第一个线程修改了a =1，第二个线程可能看到还是刚才的那个值，可能刚才是0，那么他看到可能就是0。也可能是1。这就是可见性问题

### happens-before

Happens-before原则指的就是说我们这件事发生了，如果其他线程一定能看到我之前所做的修改的话，就代表她们拥有happens-before。

### 3.Lock的加解锁和synchronized有同样的内存语义，也就是说，下一个线程加锁后可以看到所有前一个线程解锁前发生的所有操作

# 锁的分类

# 乐观锁和悲观锁

# 可重入锁和非可重入锁，以ReentrantLock为例(重点)

# 公平锁和非公平锁

# 共享锁和排它锁：以ReentrantReadWriteLock读写锁为例(重点)

# 自旋锁和阻塞锁

# 可中断锁：顾名思义，就是可以响应中断的锁

# 9.锁优化