二

25 读写锁 ReadWriteLock 获取锁有哪些规则?

在本课时我们主要讲解读写锁 ReadWriteLock 获取锁有哪些规则呢?

在没有读写锁之前，我们假设使用普通的 ReentrantLock，那么虽然我们保证了线程安全，但是也浪费了一定的资源，因为如果多个读操作同时进行，其实并没有线程安全问题，我们可以允许让多个读操作并行，以便提高程序效率。

但是写操作不是线程安全的，如果多个线程同时写，或者在写的同时进行读操作，便会造成线程安全问题。

我们的读写锁就解决了这样的问题，它设定了一套规则，既可以保证多个线程同时读的效率，同时又可以保证有写入操作时的线程安全。

整体思路是它有两把锁，第 1 把锁是写锁，获得写锁之后，既可以读数据又可以修改数据，而第 2 把锁是读锁，获得读锁之后，只能查看数据，不能修改数据。读锁可以被多个线程同时持有，所以多个线程可以同时查看数据。

在读的地方合理使用读锁，在写的地方合理使用写锁，灵活控制，可以提高程序的执行效率。

读写锁的获取规则

我们在使用读写锁时遵守下面的获取规则：

1. 如果有一个线程已经占用了读锁，则此时其他线程如果要申请读锁，可以申请成功。
2. 如果有一个线程已经占用了读锁，则此时其他线程如果要申请写锁，则申请写锁的线程会一直等待释放读锁，因为读写不能同时操作。
3. 如果有一个线程已经占用了写锁，则此时其他线程如果申请写锁或者读锁，都必须等待之前的线程释放写锁，同样也因为读写不能同时，并且两个线程不应该同时写。

所以我们用一句话总结：要么是一个或多个线程同时有读锁，要么是一个线程有写锁，但是两者不会同时出现。也可以总结为：读读共享、其他都互斥（写写互斥、读写互斥、写读互斥）。

使用案例

下面我们举个例子来应用读写锁，ReentrantReadWriteLock 是 ReadWriteLock 的实现类，最主要的有两个方法：readLock() 和 writeLock() 用来获取读锁和写锁。

代码如下：

```
/**
 * 描述：        演示读写锁用法
 */
public class ReadWriteLockDemo {
    private static final ReentrantReadWriteLock reentrantReadWriteLock = new Reentr
            false);
    private static final ReentrantReadWriteLock.ReadLock readLock = reentrantReadWr
            .readLock();
    private static final ReentrantReadWriteLock.WriteLock writeLock = reentrantRead
            .writeLock();
    private static void read() {
        readLock.lock();
        try {
            System.out.println(Thread.currentThread().getName() + "得到读锁，正在读取
            Thread.sleep(500);
        } catch (InterruptedException e) {
            e.printStackTrace();
        } finally {
            System.out.println(Thread.currentThread().getName() + "释放读锁");
            readLock.unlock();
        }
    }
    private static void write() {
```

```
            writeLock.lock();
            try {
                System.out.println(Thread.currentThread().getName() + "得到写锁，正在写入
                Thread.sleep(500);
            } catch (InterruptedException e) {
                e.printStackTrace();
            } finally {
                System.out.println(Thread.currentThread().getName() + "释放写锁");
                writeLock.unlock();
            }
        }
        public static void main(String[] args) throws InterruptedException {
            new Thread(() -> read()).start();
            new Thread(() -> read()).start();
            new Thread(() -> write()).start();
            new Thread(() -> write()).start();
        }
    }
```

程序的运行结果是：

```
Thread-0得到读锁，正在读取
Thread-1得到读锁，正在读取
Thread-0释放读锁
Thread-1释放读锁
Thread-2得到写锁，正在写入
Thread-2释放写锁
Thread-3得到写锁，正在写入
Thread-3释放写锁
```

可以看出，读锁可以同时被多个线程获得，而写锁不能。

读写锁适用场合

最后我们来看下读写锁的适用场合，相比于 ReentrantLock 适用于一般场合，ReadWriteLock 适用于读多写少的情况，合理使用可以进一步提高并发效率。

上一页 下一页