Notes - Reentrant Locks

Read/Write Lock -

Read/Write Lock gives us more flexibility during locking and unlocking. Based on the type of operation being performed over the object we can segregate the locks into

1) readLock

2) writeLock

readLock allows us to lock the object for read operation, and the interesting point is that the read operation can be shared i.e if two threads are waiting for readLock then both of them can proceed forward with the operation as read operation doesn't change the data.

Where as writeLock is mutually exclusive i.e. if a writeLock is accepted then all the other lock requests should wait till the thread that owns the lock releases it.

For example let us assume the following chronologically ordered lock requests

T1 -> lock.readLock();

T2 -> lock.readLock();

T3 -> lock.readLock();

T4 -> lock.writeLock();

T5 -> lock.readLock();

Here T1, T2, T3 can share the readLock and proceed forward with the operation. Where T4 should wait till T1, T2 and T3 unlocks.

Why T5 is waiting ?

Because writeLock is requested by T4 before its request and hence all subsequent requests to read/write locks should wait.

This is in contrast to synchronized methods/blocks because for synchronized method/block there is no segregation of read and write operations. Object is locked no matter whether it is read or write.

Caution - It is always better to put the unlock operation in finally, as you need to unlock irrespective of exceptions.

Example -

Example is just for demo, hence lock/unlock operations are kept in incr() method itself. They can be added to getX() and setX() operations as well.

```java
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReadWriteLock;
import java.util.concurrent.locks.ReentrantReadWriteLock;

class Sample {

    private int x;

    // ReadWriteLock object for requesting the lock.
    ReadWriteLock rw_lock = new ReentrantReadWriteLock();

    public int getX() {
        return x;
    }

    public void setX(int x) {
     this.x = x;
    }

    public void incr() {

     // Request the write lock as the
     // operation is intended to modify the data.

     Lock lock = rw_lock.writeLock();
     lock.lock();

     try {

         int y = getX();
         y++;

         // Just for simulation
         try { Thread.sleep(1); } catch(Exception e) {}

         setX(y);

     } finally {
         // Unlock
         lock.unlock();
     }
    }

}

class MyThread extends Thread {

    Sample obj;

    public MyThread(Sample obj) {
     this.obj = obj;
    }

    public void run() {
     obj.incr();
    }
}

public class Main {

    public static void main(String[] args) {

     Sample obj = new Sample();
     obj.setX(10);

     MyThread t1 = new MyThread(obj);
     MyThread t2 = new MyThread(obj);

     t1.start();
     t2.start();

```

```
        try {
            t1.join();
            t2.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        System.out.println( obj.getX() );
       }
   }
```

Course content

Progress cannot be changed for this item

31. Need for Synchronization

5min

Progress cannot be changed for this item

32. Synchronized methods

8min

Progress cannot be changed for this item

33. Synchronized in case of static members

3min

Progress cannot be changed for this item

34. The Problem with Synchronized Method and Solution with Synchronized Block

6min

Progress cannot be changed for this item

35. Notes - Thread Synchronization

5min

Progress cannot be changed for this item

36. Deadlocks and solution with lock sequencing

7min

Progress cannot be changed for this item

37. Notes - Deadlocks and solution with lock sequencing

1min

Progress cannot be changed for this item

38. Reentrant Locks

3min

Progress cannot be changed for this item

39. Notes - Reentrant Locks

2min

Progress cannot be changed for this item

40. Problem Set - 2

1min

Progress cannot be changed for this item

Section 4: Mock HttpServer and ThreadLocal Pattern

16min
Resources

Progress cannot be changed for this item
46. ThreadLocal

7min

Progress cannot be changed for this item
47. Notes - ThreadLocal

2min

Section 5: java.util.concurrent package

2 / 10|1hr 13min

Progress cannot be changed for this item
48. BlockingQueue and revised producer and consumer problem

6min

Progress cannot be changed for this item
49. Notes - BlockingQueue and revised producer and consumer problem

2min

Progress cannot be changed for this item

50. PriorityBlockingQueue

8min

Progress cannot be changed for this item

51. Notes - PriorityBlockingQueue

1min

Progress cannot be changed for this item

52. Fork Join Framework

13min

Progress cannot be changed for this item

53. Notes - Fork Join Framework

3min

Progress cannot be changed for this item

54. Semaphore

8min

Progress cannot be changed for this item
55. CountDownLatch

14min

Progress cannot be changed for this item
56. CyclicBarrier

7min

Progress cannot be changed for this item
57. Atomic Types (AtomicInteger, AtomicBoolean....)

12min

Section 6: Kick Start Thinking Distributed

0 / 2|8min

Section 7: Source Code

0 / 1|1min

Section 8: What's next?

0 / 1l1min

Overview
Q&A
Questions and answers
Notes
Announcements

Create a new note at 0:00

All lectures

Sort by most recent

Click the "Create a new note" box, the "+" button, or press "N" to make your first note.

×
Close alert