Lock

Quluf

dasturlash.uz

Lock 1

- The problem with traditional synchronized keyword:
  - We are not having any flexibility to try for a lock without waiting
  - There is no way to specify maximum waiting time for a thread to lock
  - So that thread will wait until getting the lock, which may creates performance problems which may cause deadlock.
  - If thread releases the lock then which waiting thread will get that lock we not having any control on this.

- Odatiy synchronized kalit so’zi muommolari
  - Biz kutmasdan qulfni olish imkoniyatiga ega emasmiz.
  - Buyerda Thread qulufni olishi uchun maxsimum kutush vaqtini tayinlash iloji yo’q.
  - Shunday qilib Thread qulufni olmagunucha kutadi. Bu esa samaralik muommosini kelib chiqarib deadlock ga olib kelishi mumkin.
  - Agar Thread qulufni bo’shatsa qaysi kutayotgan Thread lock/qulufni olishini biz boshqara olmaymiz.

dasturlash.uz

Lock 2

- The problem with traditional synchronized keyword:
  - There is no API to list out all waiting thread for a lock
  - Synchronized keyword compulsory we have to use either at method level or within a method, and it is not possible to use across multiple methods.
  - To overcome these problems sun people introduced java.util.concurrent.lock package in 1.5 version.

- Odatiy synchronized kalit so’zi muommolari
  - Buyerda qulufni olish uchun kutayotgan Thread larni ro’yhatini olish uchun API yo’q.
  - Synchronized kalit so’zini biz method darajasida yoki method ichida ishlatishimiz kerak.
  - Shu muommolarni hal qilish uchun sun odamlari javani 1.5 versiyasida java.util.concurrent.lock package ni taqdim etdi.

Lock 3

- Thus, the Java Lock interface provides a more flexible alternative to a Java synchronized block.
- Javada Lock interface synchronized block ga kushli alternative sifatida ishlatiladi.

Lock vs Synchronized Block

- The main differences between a Lock and a synchronized block are:
  - A synchronized block makes no guarantees about the sequence in which threads waiting to entering it are granted access.
  - You cannot pass any parameters to the entry of a synchronized block. Thus, having a timeout trying to get access to a synchronized block is not possible.
  - The synchronized block must be fully contained within a single method. A Lock can have it's calls to lock() and unlock() in separate methods.
- Lock va synchronized block larning asosiy farqi:
  - synchronized blok ga kirishni kutayotgan Thread lar haqida hech qanday malumot bermaydi.
  - Synchronized block da lock ni olish davomida timeout qo’yish iloji yo’q.
  - Synchronized block hardoim bitta methodda bo’lishi kerak. Lock da esa lock ni olish va uni qo’yib yuborish alohida methodlarda bo’lishi mumkin.

dasturlash.uz

Lock Interface

- Since Java Lock is an interface, you cannot create an instance of Lock directly. You must create an instance of a class that implements the Lock interface. The java.util.concurrent.locks package has the following implementations of the Lock interface:
  - java.util.concurrent.locks.ReentrantLock

- Javada Lock bu interface bo’lgani uchun uni o’zini ishlata olmaymiz. Biz Lock interface dan implements olgan qaysidir class ni ishlatishimiz kerak. java.util.concurrent.lock package da Lock interface ni implements qilgan class larni taqdim etgan:
  - java.util.concurrent.locks.ReentrantLock

Lock Interface Methods 1

- void lock();
  - We can user this method to acquire a lock. If lock is already available then immediately current thread will get that lock.
  - If the lock is not available then it will wait until getting the lock. It is exactly same behavior of traditional synchronized key work.
  - Bu method lock/quluf ni olish uchun ishlatiladi. Agar lock/quluf mavjut bo'lsa hozirgi thread lock/quluf ni oladi.
  - Agar lock/quluf mavjut bo'lmasa Thread uni olguncha kutib turadi. Bu odatiy synchronized ning ishlashi bilna bir xil.

```
Lock lock = new ReentrantLock();

lock.lock();

    //critical section

lock.unlock();
```

dasturlash.uz

Lock Interface Methods 2

- boolean trylock()
  - To acquire the lock not waiting.
  - If the lock is available then the thread acquire that lock and returns true. If lock is no available then method returns false and can continue executing with out waiting.
  - In these case thread never be enter into waiting state.

  - Kutmasdan lock/quluf ni olish uchun ishlatiladi.
  - Agar lock/quluf bo’lsa Thread uni oladi/egallaydi va method true return qiladi. Agar lock/quluf bo’lmasa method false return qiladi va Thread qukufsiz ishlashni davom ettiradi.
  - Bu holatda Thread hechqachon waiting state ga kirmaydi.

trylock() method example

```
try(t.trylock()){
   // perform safe operation
}else {
   // Perform Alternative operation
}
```

Lock Interface Methods 3

- boolean trylock(long time, TimeUnit unit)
  - If lock is available then the tread will get the lock and continue its execution
  - If the lock is not available then thread will wait until specified amount of time.
  - Still if the lock is not available then thread con continue its execution.
  - TimeUnit is a Enum present in java.util.concurrent package

  - Agar lock/quluf bo'lsa Thread uni oladi va ishlashni davom ettiradi.
  - Agar lock/quluf mavjut bo'lmasa Thread berilgan vaqt maboynida kutib turadi.
  - Malum bir vaqt o'tganidan keyinham lock/quluf mavjut bo'lmasa Thread lock/quluf siz o'z ishini davom ettiradi.
  - TimeUnit bu java.util.concurrent package da joylashgan Enum dir.

dasturlash.uz

trylock(long time, TimeUnit unit) example

```
if(t.trylock(1000,TimeUnit.MILLISECOND)){


}
```

Lock Interface Methods 4

- void lockInterruptibly()
  - Acquired the lock . If it is available and returns immediately.
  - If the lock is not available then it will wait. While waiting if the thread is interrupted then thread will not get the lock.
  - Lokc/quluf ni oladi/egallaydi. Agar bo’lsa eegallaydi va darho return qiladi.
  - Agar lock/quluf bo’lmasa Thread kutib turadi. Kutush jarayonida thread interrupt qilinga kutishni to’xtatadi.

dasturlash.uz

Lock Interface Methods 5

- void unlock()
  - To release the lock.
  - If at the beginning we call unlock() method we get RE: IllegalMonitorStateException
  - To call these methods compulsory Current thread should be owner of the lock, other wise we will get RE: IllegalMonitorStateException
  - Lock/quluf ni qo'yib yuborish uchun ishlatiladi.
  - Agar unlock() methodni boshida chaqirilsa IllegalMonitorStateException degan sodir bo'ladi.
  - Bu methodni chaqirish uchun Hozirgi Thread lock/quluf ni egasi bo'lishi kerak bo'lmasa IllegalMonitorStateException sodir bo'ladi.

```
l.lock()
.
.
.
l.unclock()
```

dasturlash.uz

Mazgiation Charchamadizlarmi?

dasturlash.uz

ReentrantLock

- Qayta kiruvchi Lock

Reentrantlock 1

- It is a implementation class of Lock Interface and It is a direct child class of Object
- Bu Lock interface dan implementatsiya olgan class dir.

Reentrantlock 2

- Reentant means a thread can require same lock multiple times without any issue.
- Internally reentant lock increments threads personal count when ever we call lock method and decrements when ever thread calls unlock method and lack releases when ever count reaches 0.
- Reentant degani, mavzu hech qanday muammosiz bir xil qulfni bir necha marta talab qilishi mumkin.
- Ichki qayta kirish blokirovkasi biz bloklash usulini chaqirganimizda mavzular shaxsiy sonini oshiradi va har doim blokni ochish usulini chaqirganda kamayadi va soni 0 ga yetganda reentrantlar yo'q bo'ladi.

Reentrantlock Constructors

- r = new Reentarentlock();
  - Creates an instance of reentreredLock.
- r = new Reentrentlock(boolean fairness)
  - Created reentrantlock with a given fairness policy.
  - If fairness is a true then longest waiting thread can acquire the lock if it is available. That is it follows FCFS policy.
  - If fairness is false then which waiting thread will get chance e can not expect.
  - Default value for fairness is false.
  - Berilgan adolat siyosati bilan reentrantlock yaratildi.
  - Agar fairness true bo’lsa eng ko’p kutayotgan thread lock/quluf ni egallaydi. Bu FCFS siyosatiga amal qiladi.
  - Agar fairness false bo’lsa qaysi qaysi Thread lock/qulufni olishini bilmaymiz.
  - Default holatda fiarness false bo’ladi.

dasturlash.uz

ReentrantLock() Methods 1

- void lock()
- boolean trylock()
- boolean trylock(long l, TimeUnit unit)
- void lockInterruptibly()
- void unlock()
- int getHoldCount() - returnning number of hold on these lock by thread. (Thread tomonidan berilgan resourceni nechta lock/qulub borligini return qiladi).
- boolean isHeldByCurrentThread() - returns true id only if lock is hold by current thread. Method true return qiladi agar lock/quluf faqat hozirgi Thread tomonidan ushlangan bo'lsa.
- int getQueuelenght() - returns number of Thread waiting for the lock. Lock ni kutib turgan Thread larni sonini return qiladi.

dasturlash.uz

ReentrantLock() Methods 2

- boolean hasQueuedThreads() – returns true If any thread waiting to get the lock() . true return qiladi agar lokc/quluf ni kutayotgn bironta thread bo'lsa.
- boolean islocked()- returns true if lock is acquired by some thread. True return qiladi gar lock/quluf qaysidir thread tomonidan olingan bo'lsa.
- boolean isFair() – returns true if a fairness policy is set true. True return qiladi agar fairness o'zgaruvchisi true bo'lsa.
- Thread getOwner() – return a thread which acquires a luck(). Hozirda Lock/quluf ni ushlab turgan Thread ni return qiladi.

ReentrantLock example 1

```
public static void main(String[] args) {
        ReentrantLock l = new ReentrantLock(true);
        System.out.println(l.isLocked());
        l.lock();
        System.out.println(l.isFair());
        System.out.println(l.isLocked());
        System.out.println(l.getHoldCount());
        l.lock();
        System.out.println(l.getHoldCount());
        System.out.println(l.isHeldByCurrentThread());
        System.out.println(l.getQueueLength());
        l.unlock();
        System.out.println(l.getHoldCount());
        l.unlock();
        System.out.println(l.getHoldCount());
        System.out.println(l.isLocked());
        System.out.println(l.isFair());
   }
```

ReentrantLock example 2

- Resource Class

```
public class Display {
    public ReentrantLock r = new ReentrantLock();

    public void show(String name) {
        r.lock();
        try {
            for (int i = 0; i < 5; i++) {
                System.out.println("Display " + name);
                Thread.sleep(2000);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        r.unlock();
    }
}
```

ReentrantLock example 3

```
public class MyThread extends Thread{
    private Display display;
    private String name;

    public MyThread(Display display, String name) {
        this.display = display;
        this.name = name;
    }

    @Override
    public void run() {
        this.display.show(this.name);
    }
}
```

- Thread Class

ReentrantLock example 4

- Main Class

```
public class MyMain {
    public static void main(String[] args) {
        Display d = new Display();

        MyThread t1 = new MyThread(d, "Ali");
        MyThread t2 = new MyThread(d, "Vali");

        t1.start();
        t2.start();
    }
}
```

Static ReentrantLock example 1

- If it is static then we get class level lock
- Thread Class and Resource

```
public class MyThread extends Thread {
    static ReentrantLock r = new ReentrantLock();
    public MyThread(String name) {
        super(name);
    }

    @Override
    public void run() {
        if (r.tryLock()) {
            System.out.println(Thread.currentThread().getName() + "..... got lock and performing safe operation ");
            try {
                Thread.sleep(2000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            r.unlock();
        } else {
            System.out.println(Thread.currentThread().getName() + ".... unable to get lock hence performing alternati
operation ");
        }
    }
}
```

dasturlash.uz

Static ReentrantLock example 2

```
public class MyMain {
    public static void main(String[] args) {

        MyThread t1 = new MyThread("Ali");
        MyThread t2 = new MyThread("Vali");

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

    }
}
```

- Main Class
- Thread Class and Resource

static ReentrantLock with time exp 1

```
public class MyThread extends Thread {
    static ReentrantLock r = new ReentrantLock();
    @Override
    public void run() {
        try {
            do {
                if (r.tryLock(5000, TimeUnit.MILLISECONDS)) {
                      Thread.sleep(30000);
                    r.unlock();
                    break;
                } else {
                    ///
                }
            } while (true);

        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

- Thread Class and Resource
- Wait for a Lock particular amount of time

static ReentrantLock with time exp 2

- Main Class

```
public class MyMain {
    public static void main(String[] args) {


        MyThread t1 = new MyThread("Ali");
        MyThread t2 = new MyThread("Vali");


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

    }
}
```

dasturlash.uz

Other reentrantLock classes

- *ReentrantReadWriteLock*
- *StampedLock*
-

ReentrantLock Links

- https://www.javatpoint.com/java-reentrantlock
- https://www.geeksforgeeks.org/reentrant-lock-java/
- https://www.baeldung.com/java-concurrent-locks
- https://jenkov.com/tutorials/java-util-concurrent/lock.html

ReadWriteLock class

- A java.util.concurrent.locks.ReadWriteLock is an advanced thread lock mechanism. It allows multiple threads to read a certain resource, but only one to write it, at a time.
- Java.util.concurrent.locks.ReadWriteLock Thread larda mukammal blocklar mexanizmidir. U umumiy manbani birnechta Thread larga bir vaqtni o'zida o'qish imkonini beradi, ammo qiymatni o'zgartirish uchun bir vaqtni o'zida bitta Thread ga ruxsat beradi.

ReadWriteLock Links

- https://jenkov.com/tutorials/java-util-concurrent/readwritelock.html#read-write-lock-locking-rules