**[synchronized与Lock的区别](https://www.cnblogs.com/iyyy/p/7993788.html)**

两者区别：

1.首先synchronized是java内置关键字，在jvm层面，Lock是个java类；

2.synchronized无法判断是否获取锁的状态，Lock可以判断是否获取到锁；

3.synchronized会自动释放锁(a 线程执行完同步代码会释放锁 ；b 线程执行过程中发生异常会释放锁)，Lock需在finally中手工释放锁（unlock()方法释放锁），否则容易造成线程死锁；

4.用synchronized关键字的两个线程1和线程2，如果当前线程1获得锁，线程2线程等待。如果线程1阻塞，线程2则会一直等待下去，而Lock锁就不一定会等待下去，如果尝试获取不到锁，线程可以不用一直等待就结束了；

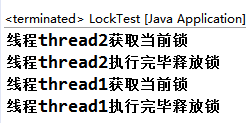
5.synchronized的锁可重入、不可中断、非公平，而Lock锁可重入、可判断、可公平（两者皆可）

6.Lock锁适合大量同步的代码的同步问题，synchronized锁适合代码少量的同步问题。

小例子：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46 | package com.cn.test.thread.lock;    import java.util.concurrent.locks.Lock;  import java.util.concurrent.locks.ReentrantLock;    public class LockTest {      private Lock lock = new ReentrantLock();      /\*       \* 使用完毕释放后其他线程才能获取锁       \*/      public void lockTest(Thread thread) {          lock.lock();//获取锁          try {              System.out.println("线程"+thread.getName() + "获取当前锁"); //打印当前锁的名称              Thread.sleep(2000);//为看出执行效果，是线程此处休眠2秒          } catch (Exception e) {              System.out.println("线程"+thread.getName() + "发生了异常释放锁");          }finally {              System.out.println("线程"+thread.getName() + "执行完毕释放锁");              lock.unlock(); //释放锁          }      }        public static void main(String[] args) {          LockTest lockTest = new LockTest();          //声明一个线程 “线程一”          Thread thread1 = new Thread(new Runnable() {              @Override              public void run() {                  lockTest.lockTest(Thread.currentThread());              }          }, "thread1");          //声明一个线程 “线程二”          Thread thread2 = new Thread(new Runnable() {                @Override              public void run() {                  lockTest.lockTest(Thread.currentThread());              }          }, "thread2");          // 启动2个线程          thread2.start();          thread1.start();        }  } |

执行结果：



[复制代码](javascript:void(0);)

package com.cn.test.thread.lock;

import java.util.concurrent.locks.Lock;

import java.util.concurrent.locks.ReentrantLock;

public class LockTest {

private Lock lock = new ReentrantLock();

/\*

\* 尝试获取锁 tryLock() 它表示用来尝试获取锁，如果获取成功，则返回true，如果获取失败（即锁已被其他线程获取），则返回false

\*/

public void tryLockTest(Thread thread) {

if(lock.tryLock()) { //尝试获取锁

try {

System.out.println("线程"+thread.getName() + "获取当前锁"); //打印当前锁的名称

Thread.sleep(2000);//为看出执行效果，是线程此处休眠2秒

} catch (Exception e) {

System.out.println("线程"+thread.getName() + "发生了异常释放锁");

}finally {

System.out.println("线程"+thread.getName() + "执行完毕释放锁");

lock.unlock(); //释放锁

}

}else{

System.out.println("我是线程"+Thread.currentThread().getName()+"当前锁被别人占用，我无法获取");

}

}

public static void main(String[] args) {

LockTest lockTest = new LockTest();

Thread thread1 = new Thread(new Runnable() {

@Override

public void run() {

lockTest.tryLockTest(Thread.currentThread());

}

}, "thread1");

//声明一个线程 “线程二”

Thread thread2 = new Thread(new Runnable() {

@Override

public void run() {

lockTest.tryLockTest(Thread.currentThread());

}

}, "thread2");

// 启动2个线程

thread2.start();

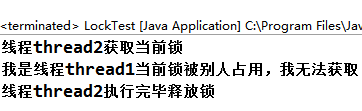
thread1.start();

}

}

[复制代码](javascript:void(0);)

执行结果：



[复制代码](javascript:void(0);)

package com.cn.test.thread.lock;

import java.util.concurrent.TimeUnit;

import java.util.concurrent.locks.Lock;

import java.util.concurrent.locks.ReentrantLock;

public class LockTest {

private Lock lock = new ReentrantLock();

public void tryLockParamTest(Thread thread) throws InterruptedException {

if(lock.tryLock(3000, TimeUnit.MILLISECONDS)) { //尝试获取锁 获取不到锁，就等3秒，如果3秒后还是获取不到就返回false

try {

System.out.println("线程"+thread.getName() + "获取当前锁"); //打印当前锁的名称

Thread.sleep(4000);//为看出执行效果，是线程此处休眠2秒

} catch (Exception e) {

System.out.println("线程"+thread.getName() + "发生了异常释放锁");

}finally {

System.out.println("线程"+thread.getName() + "执行完毕释放锁");

lock.unlock(); //释放锁

}

}else{

System.out.println("我是线程"+Thread.currentThread().getName()+"当前锁被别人占用,等待3s后仍无法获取,放弃");

}

}

public static void main(String[] args) {

LockTest lockTest = new LockTest();

Thread thread1 = new Thread(new Runnable() {

@Override

public void run() {

try {

lockTest.tryLockParamTest(Thread.currentThread());

} catch (InterruptedException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

}, "thread1");

//声明一个线程 “线程二”

Thread thread2 = new Thread(new Runnable() {

@Override

public void run() {

try {

lockTest.tryLockParamTest(Thread.currentThread());

} catch (InterruptedException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

}, "thread2");

// 启动2个线程

thread2.start();

thread1.start();

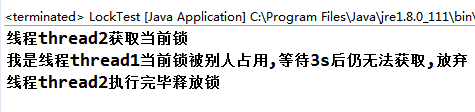
}

}

[复制代码](javascript:void(0);)

执行结果：

因为此时线程1休眠了4秒，线程2等待了3秒还没有获取到就放弃获取锁了，执行结束



将方法中的 Thread.sleep(4000)改为Thread.sleep(2000)执行结果如下：

因为此时线程1休眠了2秒，线程2等待了3秒的期间线程1释放了锁，此时线程2获取到锁，线程2就可以执行了

