可重入锁详解

概述

什么是 “可重入”，可重入就是说某个线程已经获得某个锁，可以再次获取锁而不会出现死锁。例如

package com.test.reen;

// 演示可重入锁是什么意思，可重入，就是可以重复获取相同的锁，synchronized和ReentrantLock都是可重入的

// 可重入降低了编程复杂性

public class WhatReentrant {

public static void main(String[] args) {

new Thread(new Runnable() {

@Override

public void run() {

synchronized (this) {

System.out.println("第1次获取锁，这个锁是：" + this);

int index = 1;

while (true) {

synchronized (this) {

System.out.println("第" + (++index) + "次获取锁，这个锁是：" + this);

}

if (index == 10) {

break;

}

}

}

}

}).start();

}

}

package com.test.reen;

import java.util.Random;

import java.util.concurrent.locks.ReentrantLock;

// 演示可重入锁是什么意思

public class WhatReentrant2 {

public static void main(String[] args) {

ReentrantLock lock = new ReentrantLock();

new Thread(new Runnable() {

@Override

public void run() {

try {

lock.lock();

System.out.println("第1次获取锁，这个锁是：" + lock);

int index = 1;

while (true) {

try {

lock.lock();

System.out.println("第" + (++index) + "次获取锁，这个锁是：" + lock);

try {

Thread.sleep(new Random().nextInt(200));

} catch (InterruptedException e) {

e.printStackTrace();

}

if (index == 10) {

break;

}

} finally {

lock.unlock();

}

}

} finally {

lock.unlock();

}

}

}).start();

}

}

可以发现没发生死锁，可以多次获取相同的锁

可重入锁有

synchronized

ReentrantLock

使用ReentrantLock的注意点

ReentrantLock 和 synchronized 不一样，需要手动释放锁，所以使用 ReentrantLock的时候一定要手动释放锁，并且加锁次数和释放次数要一样

以下代码演示，加锁和释放次数不一样导致的死锁

package com.test.reen;

import java.util.Random;

import java.util.concurrent.locks.ReentrantLock;

public class WhatReentrant3 {

public static void main(String[] args) {

ReentrantLock lock = new ReentrantLock();

new Thread(new Runnable() {

@Override

public void run() {

try {

lock.lock();

System.out.println("第1次获取锁，这个锁是：" + lock);

int index = 1;

while (true) {

try {

lock.lock();

System.out.println("第" + (++index) + "次获取锁，这个锁是：" + lock);

try {

Thread.sleep(new Random().nextInt(200));

} catch (InterruptedException e) {

e.printStackTrace();

}

if (index == 10) {

break;

}

} finally {

// lock.unlock();// 这里故意注释，实现加锁次数和释放次数不一样

}

}

} finally {

lock.unlock();

}

}

}).start();

new Thread(new Runnable() {

@Override

public void run() {

try {

lock.lock();

for (int i = 0; i < 20; i++) {

System.out.println("threadName:" + Thread.currentThread().getName());

try {

Thread.sleep(new Random().nextInt(200));

} catch (InterruptedException e) {

e.printStackTrace();

}

}

} finally {

lock.unlock();

}

}

}).start();

}

}

由于加锁次数和释放次数不一样，第二个线程始终无法获取到锁，导致一直在等待。

稍微改一下，在外层的finally里头释放9次，让加锁和释放次数一样，就没问题了

try {

lock.lock();

System.out.println("第1次获取锁，这个锁是：" + lock);

int index = 1;

while (true) {

try {

lock.lock();

System.out.println("第" + (++index) + "次获取锁，这个锁是：" + lock);

... 代码省略节省篇幅...

} finally {

// lock.unlock();// 这里故意注释，实现加锁次数和释放次数不一样

}

}

} finally {

lock.unlock();

// 在外层的finally里头释放9次，让加锁和释放次数一样，就没问题了

for (int i = 0; i < 9; i++) {

lock.unlock();

}

}

————————————————

版权声明：本文为CSDN博主「石头StoneWang」的原创文章，遵循CC 4.0 BY-SA版权协议，转载请附上原文出处链接及本声明。

原文链接：https://blog.csdn.net/w8y56f/article/details/89554060