1. 理解线程的常用方法

1.1 理解设置守护线程

线程对象.setDaemon(true);

2.理解线程池的概念(掌握)

2.1 利用Excecutes创建线程池

2.2 利用ThreadPoolExector类创建线程，重点理构造方法的参数

核心线程数 最大线程数 排队的线程数 拒绝策略

3.理解锁

3.1 synchronized(悲观锁)和CAS(乐观锁)

3.2 乐观锁底层基于CAS算法(理解算法)

3.3 AtomicInteger类 实现cas算法

代码:示例一:没有解决并发

public class MyThread implements Runnable{

private int count = 0; //送冰淇淋的数量

@Override

public void run() {

for (int i = 0; i < 10; i++) {

count++;

System.out.println(Thread.currentThread().getName()+"已经送了" + count + "个冰淇淋"); //1

try {

Thread.sleep(200);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

}

主函数:

//3.1创建线程池对象

ThreadPoolExecutor pool=new ThreadPoolExecutor(3,

5, 5, TimeUnit.SECONDS,

new ArrayBlockingQueue<>(3),

Executors.defaultThreadFactory(),new ThreadPoolExecutor.AbortPolicy());

//3.2指定线程任务

pool.submit(myThread);

pool.submit(myThread);

pool.submit(myThread);

//3.3关闭

pool.shutdown();

代码二：修改线程任务----使用atomicInteger解决并发问题 数字自增

public class MyThread implements Runnable{

//private int count = 0; //送冰淇淋的数量

private AtomicInteger atom=new AtomicInteger();

@Override

public void run() {

for (int i = 0; i < 10; i++) {

//count++;

int count = atom.incrementAndGet(); //实现相加取值

System.out.println(Thread.currentThread().getName()+"已经送了" + count + "个冰淇淋"); //1

try {

Thread.sleep(200);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

}