Java 核心技术(进阶)

第五章 Java多线程和并发编程

第八节 Java并发协作控制

华东师范大学 陈良育

线程协作

- Thread/Executor/Fork-Join
  - 线程启动，运行，结束
  - 线程之间缺少协作

- synchronized 同步
  - 限定只有一个线程才能进入关键区
  - 简单粗暴，性能损失有点大☹

Lock

- Lock也可以实现同步的效果
  - 实现更复杂的临界区结构
  - tryLock方法可以预判锁是否空闲
  - 允许分离读写的操作，多个读，一个写
  - 性能更好
- ReentrantLock类，可重入的互斥锁
- ReentrantReadWriteLock类，可重入的读写锁
- lock和unlock函数

Semaphore

- 信号量，由1965年Dijkstra提出的
- 信号量：本质上是一个计数器
- 计数器大于0，可以使用，等于0不能使用
- 可以设置多个并发量，例如限制10个访问
- Semaphore
  - acquire获取
  - release释放
- 比Lock更进一步，可以控制多个同时访问关键区

Latch

- 等待锁，是一个同步辅助类
- 用来同步执行任务的一个或者多个线程
- 不是用来保护临界区或者共享资源
- CountDownLatch
  - countDown() 计数减1
  - await() 等待latch变成0

Barrier

- 集合点，也是一个同步辅助类
- 允许多个线程在某一个点上进行同步
- CyclicBarrier
  - 构造函数是需要同步的线程数量
  - await等待其他线程，到达数量后，就放行

Phaser

- 允许执行并发多阶段任务，同步辅助类
- 在每一个阶段结束的位置对线程进行同步，当所有的线程都到达这步，再进行下一步
- Phaser
  - arrive()
  - arriveAndAwaitAdvance()

Exchanger

- 允许在并发线程中互相交换消息
- 允许在2个线程中定义同步点，当两个线程都到达同步点，它们交换数据结构
- Exchanger
  - exchange(), 线程双方互相交互数据
  - 交换数据是双向的

总结

- java.util.concurrent包提供了很多并发编程的控制协作类
- 根据业务特点，使用正确的线程并发控制协作

代码(1) LockExample.java


```java
public class LockExample {

    private static final ReentrantLock queueLock = new ReentrantLock(); //可重入锁
    private static final ReentrantReadWriteLock orderLock = new ReentrantReadWriteLock(); //可重入读写锁

    /**
     * 有家奶茶店，点单有时需要排队
     * 假设想买奶茶的人如果看到需要排队，就决定不买
     * 又假设奶茶店有老板和多名员工，记单方式比较原始，只有一个订单本
     * 老板负责写新订单，员工不断地查看订单本得到信息来制作奶茶，在老板写新订单时员工不能看订单本
     * 多个员工可同时看订单本，在员工看时老板不能写新订单
     * @param args
     * @throws InterruptedException
     */
    public static void main(String[] args) throws InterruptedException {
        //buyMilkTea();
        handleOrder(); //需手动关闭
    }
```

代码(2) LockExample.java


```
public void tryToBuyMilkTea() throws InterruptedException {
    boolean flag = true;
    while(flag)
    {
        if (queueLock.tryLock()) {
            //queueLock.lock();
            long thinkingTime = (long) (Math.random() * 500);
            Thread.sleep(thinkingTime);
            System.out.println(Thread.currentThread().getName() + ": 来一杯珍珠奶茶，不要珍珠");
            flag = false;
            queueLock.unlock();
        } else {
            //System.out.println(Thread.currentThread().getName() + ": " + queueLock.getQue
            System.out.println(Thread.currentThread().getName() + ": 再等等");
        }
        if(flag)
        {
            Thread.sleep(1000);
        }
    }

}
```

代码(3) LockExample.java


```
public void addOrder() throws InterruptedException {
    orderLock.writeLock().lock();
    long writingTime = (long) (Math.random() * 1000);
    Thread.sleep(writingTime);
    System.out.println("老板新加一笔订单");
    orderLock.writeLock().unlock();
}

public void viewOrder() throws InterruptedException {
    orderLock.readLock().lock();

    long readingTime = (long) (Math.random() * 500);
    Thread.sleep(readingTime);
    System.out.println(Thread.currentThread().getName() + ": 查看订单本");
    orderLock.readLock().unlock();

}
```

代码(4) LockExample.java


```
public static void buyMilkTea() throws InterruptedException {
    LockExample lockExample = new LockExample();
    int STUDENTS_CNT = 10;

    Thread[] students = new Thread[STUDENTS_CNT];
    for (int i = 0; i < STUDENTS_CNT; i++) {
        students[i] = new Thread(new Runnable() {

            @Override
            public void run() {
                try {
                    long walkingTime = (long) (Math.random() * 1000);
                    Thread.sleep(walkingTime);
                    lockExample.tryToBuyMilkTea();
                } catch(InterruptedException e) {
                    System.out.println(e.getMessage());
                }
            }

        }
        );

        students[i].start();
    }

    for (int i = 0; i < STUDENTS_CNT; i++)
        students[i].join();

}
```

代码(5) LockExample.java


```
public static void handleOrder() throws InterruptedException {
    LockExample lockExample = new LockExample();


    Thread boss = new Thread(new Runnable() {

        @Override
        public void run() {
            while (true) {
                try {
                    lockExample.addOrder();
                    long waitingTime = (long) (Math.random() * 1000);
                    Thread.sleep(waitingTime);
                } catch (InterruptedException e) {
                    System.out.println(e.getMessage());
                }
            }
        }
    });
    boss.start();
```

代码(6) LockExample.java


```
        int workerCnt = 3;
        Thread[] workers = new Thread[workerCnt];
        for (int i = 0; i < workerCnt; i++)
        {
            workers[i] = new Thread(new Runnable() {

                @Override
                public void run() {
                    while (true) {
                        try {
                            lockExample.viewOrder();
                            long workingTime = (long) (Math.random() * 5000);
                            Thread.sleep(workingTime);
                        } catch (InterruptedException e) {
                            System.out.println(e.getMessage());
                        }
                    }
                }

            });

            workers[i].start();
        }

    }
}
```

代码(7) SemaphoreExample.java


```
public class SemaphoreExample {

    private final Semaphore placeSemaphore = new Semaphore(5);

    public boolean parking() throws InterruptedException {
        if (placeSemaphore.tryAcquire()) {
            System.out.println(Thread.currentThread().getName() + ": 停车成功");
            return true;
        } else {
            System.out.println(Thread.currentThread().getName() + ": 没有空位");
            return false;
        }

    }

    public void leaving() throws InterruptedException {
        placeSemaphore.release();
        System.out.println(Thread.currentThread().getName() + ": 开走");
    }
```

代码(8) SemaphoreExample.java


```
/**
 * 现有一地下车库，共有车位5个，由10辆车需要停放，每次停放时，去申请信号量
 * @param args
 * @throws InterruptedException
 */
public static void main(String[] args) throws InterruptedException {
    int tryToParkCnt = 10;

    SemaphoreExample semaphoreExample = new SemaphoreExample();

    Thread[] parkers = new Thread[tryToParkCnt];

    for (int i = 0; i < tryToParkCnt; i++) {
        parkers[i] = new Thread(new Runnable() {
```

代码(9) SemaphoreExample.java


```
                @Override
                public void run() {
                    try {
                        long randomTime = (long) (Math.random() * 1000);
                        Thread.sleep(randomTime);
                        if (semaphoreExample.parking()) {
                            long parkingTime = (long) (Math.random() * 1200);
                            Thread.sleep(parkingTime);
                            semaphoreExample.leaving();
                        }
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
            });

            parkers[i].start();
        }

        for (int i = 0; i < tryToParkCnt; i++) {
            parkers[i].join();
        }
    }
}
```

代码(10) CountDownLatchExample.java


```
public class CountDownLatchExample {

    /**
     * 设想百米赛跑比赛 发令枪发出信号后选手开始跑，全部选手跑到终点后比赛结束
     *
     * @param args
     * @throws InterruptedException
     */
    public static void main(String[] args) throws InterruptedException {
        int runnerCnt = 10;
        CountDownLatch startSignal = new CountDownLatch(1);
        CountDownLatch doneSignal = new CountDownLatch(runnerCnt);

        for (int i = 0; i < runnerCnt; ++i) // create and start threads
            new Thread(new Worker(startSignal, doneSignal)).start();

        System.out.println("准备工作...");
        System.out.println("准备工作就绪");
        startSignal.countDown(); // let all threads proceed
        System.out.println("比赛开始");
        doneSignal.await(); // wait for all to finish
        System.out.println("比赛结束");
    }
```

代码(11) CountDownLatchExample.java


```
    static class Worker implements Runnable {
        private final CountDownLatch startSignal;
        private final CountDownLatch doneSignal;

        Worker(CountDownLatch startSignal, CountDownLatch doneSignal) {
            this.startSignal = startSignal;
            this.doneSignal = doneSignal;
        }

        public void run() {
            try {
                startSignal.await();
                doWork();
                doneSignal.countDown();
            } catch (InterruptedException ex) {
            } // return;
        }

        void doWork() {
            System.out.println(Thread.currentThread().getName() + ": 跑完全程");
        }
    }
}
```

代码(12) CyclicBarrierExample.java


```
public class CyclicBarrierExample {

    /**
     * 假定有三行数，用三个线程分别计算每一行的和，最终计算总和
     * @param args
     */
    public static void main(String[] args) {
        final int[][] numbers = new int[3][5];
        final int[] results = new int[3];
        int[] row1 = new int[]{1, 2, 3, 4, 5};
        int[] row2 = new int[]{6, 7, 8, 9, 10};
        int[] row3 = new int[]{11, 12, 13, 14, 15};
        numbers[0] = row1;
        numbers[1] = row2;
        numbers[2] = row3;

        CalculateFinalResult finalResultCalculator = new CalculateFinalResult(results);
        CyclicBarrier barrier = new CyclicBarrier(3, finalResultCalculator);
        //当有3个线程在barrier上await，就执行finalResultCalculator

        for(int i = 0; i < 3; i++) {
            CalculateEachRow rowCalculator = new CalculateEachRow(barrier, numbers, i, results);
            new Thread(rowCalculator).start();
        }
    }
}
```

代码(13) CyclicBarrierExample.java


```
class CalculateEachRow implements Runnable {

    final int[][] numbers;
    final int rowNumber;
    final int[] res;
    final CyclicBarrier barrier;

    CalculateEachRow(CyclicBarrier barrier, int[][] numbers, int rowNumber, int[] res) {
        this.barrier = barrier;
        this.numbers = numbers;
        this.rowNumber = rowNumber;
        this.res = res;
    }
```

代码(14) CyclicBarrierExample.java


```
    @Override
    public void run() {
        int[] row = numbers[rowNumber];
        int sum = 0;
        for (int data : row) {
            sum += data;
            res[rowNumber] = sum;
        }
        try {
            System.out.println(Thread.currentThread().getName() + ": 计算第" + (rowNumber + 1) + "行结束，结果为: " + sum);
            barrier.await(); //等待！只要超过3个(Barrier的构造参数)，就放行。
        } catch (InterruptedException | BrokenBarrierException e) {
            e.printStackTrace();
        }
    }

}
```

代码(15) CyclicBarrierExample.java


```
class CalculateFinalResult implements Runnable {
    final int[] eachRowRes;
    int finalRes;
    public int getFinalResult() {
        return finalRes;
    }

    CalculateFinalResult(int[] eachRowRes) {
        this.eachRowRes = eachRowRes;
    }

    @Override
    public void run() {
        int sum = 0;
        for(int data : eachRowRes) {
            sum += data;
        }
        finalRes = sum;
        System.out.println("最终结果为: " + finalRes);
    }

}
```

代码(16) PhaserExample.java


```
public class PhaserExample {

    /**
     * 假设举行考试，总共三道大题，每次下发一道题目，等所有学生完成后再进行下一道
     *
     * @param args
     */
    public static void main(String[] args) {

        int studentsCnt = 5;
        Phaser phaser = new Phaser(studentsCnt);

        for (int i = 0; i < studentsCnt; i++) {
            new Thread(new Student(phaser)).start();
        }
    }
}
```

代码(17) PhaserExample.java


```
class Student implements Runnable {

    private final Phaser phaser;

    public Student(Phaser phaser) {
        this.phaser = phaser;
    }

    @Override
    public void run() {
        try {
            doTesting(1);
            phaser.arriveAndAwaitAdvance(); //等到5个线程都到了，才放行
            doTesting(2);
            phaser.arriveAndAwaitAdvance();
            doTesting(3);
            phaser.arriveAndAwaitAdvance();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
```

代码(18) PhaserExample.java


```
    private void doTesting(int i) throws InterruptedException {
        String name = Thread.currentThread().getName();
        System.out.println(name + "开始答第" + i + "题");
        long thinkingTime = (long) (Math.random() * 1000);
        Thread.sleep(thinkingTime);
        System.out.println(name + "第" + i + "道题答题结束");
    }
}
```

代码(19) ExchangerExample.java


```
public class ExchangerExample {

    /**
     * 本例通过Exchanger实现学生成绩查询，简单线程间数据的交换
     * @param args
     * @throws InterruptedException
     */
    public static void main(String[] args) throws InterruptedException {
        Exchanger<String> exchanger = new Exchanger<String>();
        BackgroundWorker worker = new BackgroundWorker(exchanger);
        new Thread(worker).start();

        Scanner scanner = new Scanner(System.in);
        while(true) {
            System.out.println("输入要查询的属性学生姓名：");
            String input = scanner.nextLine().trim();
            exchanger.exchange(input); //把用户输入传递给线程
            String value = exchanger.exchange(null); //拿到线程反馈结果
            if ("exit".equals(value)) {
                break;
            }
            System.out.println("查询结果：" + value);
        }
        scanner.close();
    }
}
```

截图(Alt + A)

代码(20) ExchangerExample.java


```
class BackgroundWorker implements Runnable {

    final Exchanger<String> exchanger;
    BackgroundWorker(Exchanger<String> exchanger) {
        this.exchanger = exchanger;
    }
    @Override
    public void run() {
        while (true) {
            try {
```

代码(21) ExchangerExample.java


```
                String item = exchanger.exchange(null);
                switch (item) {
                case "zhangsan":
                    exchanger.exchange("90");
                    break;
                case "lisi":
                    exchanger.exchange("80");
                    break;
                case "wangwu":
                    exchanger.exchange("70");
                    break;
                case "exit":
                    exchanger.exchange("exit");
                    return;
                default:
                    exchanger.exchange("查无此人");
                }
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

谢 谢！