一，线程间通信

1.两个线程交替打印

题目:

`i=0,a:i++,b:i--,交替打印10次`

1）使用synchronized

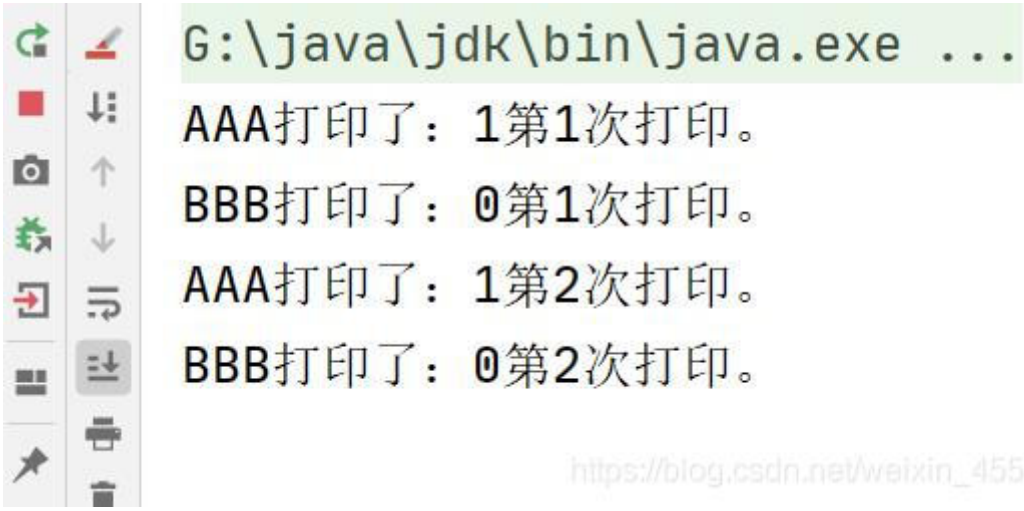
```
public class JUCNote3 {
    /**
     * i=0
     * a:i++
     * b:i-
     * 交替10次
     */
    public static void main(String[] args) {
        Share share = new Share();
        new Thread(() -> {
            for (int i = 0; i < 10; i++)
                try {
                    share.add();
                } catch (Exception e) {
                    e.printStackTrace();
                }
        }, "AAA").start();
        new Thread(() -> {
            for (int i = 0; i < 10; i++)
                try {
                    share.sub();
                } catch (Exception e) {
                    e.printStackTrace();
                }
        }, "BBB").start();
    }
}

class Share {
    private int num = 0;

    public synchronized void add()throws Exception{
        //判断
        if (num!=0)
            this.wait();

        System.out.println(Thread.currentThread().getName()+"执行了："+ ++num);
        this.notify();
    }
    public synchronized void sub()throws Exception{
        if (num==0)
            this.wait();

        System.out.println(Thread.currentThread().getName()+"执行了："+ --num);
```

```
        this.notify();
    }

}
```

2）两个线程可以正常执行，现在增加到四个

```
public class JUCNote3 {
    public static void main(String[] args) {
        Share share = new Share();
        new Thread(() -> {
            for (int i = 0; i < 10; i++)
                try {
                    share.add();
                } catch (Exception e) {
                    e.printStackTrace();
                }
        }, "AAA").start();
        new Thread(() -> {
            for (int i = 0; i < 10; i++)
                try {
                    share.sub();
                } catch (Exception e) {
                    e.printStackTrace();
                }
        }, "BBB").start();
        new Thread(() -> {
            for (int i = 0; i < 10; i++)
                try {
                    share.add();
                } catch (Exception e) {
                    e.printStackTrace();
                }
        }, "CCC").start();
        new Thread(() -> {
            for (int i = 0; i < 10; i++)
                try {
                    share.sub();
                } catch (Exception e) {
                    e.printStackTrace();
                }
        }, "DDD").start();
    }
}

class Share {
    private int num = 0;

    public synchronized void add()throws Exception{
        //判断
        if (num!=0)
            this.wait();
        Thread.sleep(500);
        System.out.println(Thread.currentThread().getName()+"执行了："+ ++num);
        this.notify();
    }
    public synchronized void sub()throws Exception{
```

```
        if (num==0)
            this.wait();
        Thread.sleep(500);
        System.out.println(Thread.currentThread().getName()+"执行了："+ --num);
        this.notify();
    }

}
```



3）线程间调用化定制通信

查看`jdkAPI  wait()`;

像在一个参数版本中，中断和虚假唤醒是可能的，并且该方法应该始终在循环中使用：

```
synchronized (obj) {
       while (<condition does not hold>)
           obj.wait();
       ... // Perform action appropriate to condition
   }
```

该方法只能由作为该对象的监视器的所有者的线程调用。 有关线程可以成为监视器所有者的方式的说明，请参阅notify方法。

注意：判断一定要while循环判断，不能用if，防止多线程虚假唤醒。

```
public synchronized void add()throws Exception{
    //判断
    while (num!=0)
        this.wait();

    System.out.println(Thread.currentThread().getName()+"执行了："+ ++num);
    this.notify();
}
```



```
public class Demo2 {
    public static void main(String[] args)throws Exception {
        Share s = new Share();
        new Thread(()->{
            try {
                for (int i = 0; i <10 ; i++)
                    s.add();
            } catch (Exception e) {
```

```
                e.printStackTrace();
            }
        },"AA").start();
        new Thread(()->{
            try {
                for (int i = 0; i <10 ; i++)
                    s.del();
            } catch (Exception e) {
                e.printStackTrace();
            }
        },"BB").start();
        new Thread(()->{
            try {
                for (int i = 0; i <10 ; i++)
                    s.add();
            } catch (Exception e) {
                e.printStackTrace();
            }
        },"CC").start();
        new Thread(()->{
            try {
                for (int i = 0; i <10 ; i++)
                    s.del();
            } catch (Exception e) {
                e.printStackTrace();
            }
        },"DD").start();
    }
}
class Share{
    private int num=0;
    public synchronized void add()throws Exception{
        while (num!=0)
            this.wait();
        System.out.println(Thread.currentThread().getName()+ ++num);
        notifyAll();
    }
    public synchronized void del()throws Exception{
        while (num!=1)
            this.wait();
        System.out.println(Thread.currentThread().getName()+ --num);
        notifyAll();
    }
}
```

4）使用lock

```
/**
 * @author yinhuidong
 * @createTime 2020-07-18-17:43
 */
public class Demo3 {
    public static void main(String[] args) {
        ShareDate date = new ShareDate();
        new Thread(() -> {
            for (int i = 0; i < 10; i++) date.add(i + 1);
        }, "AAA").start();
```

```
        new Thread(() -> {
            for (int i = 0; i < 10; i++) date.sub(i + 1);
        }, "BBB").start();
        new Thread(() -> {
            for (int i = 0; i < 10; i++) date.add(i + 1);
        }, "CCC").start();
        new Thread(() -> {
            for (int i = 0; i < 10; i++) date.sub(i + 1);
        }, "DDD").start();
    }
}

class ShareDate {
    private int i = 0;
    private ReentrantLock lock = new ReentrantLock();
    private Condition cd = lock.newCondition();

    public void add(int total) {
        try {
            lock.lock();
            while (i != 0)
                cd.await();
            Thread.sleep(500);
            System.out.println(Thread.currentThread().getName() + "打印了：" + ++i
+ "第" + total + "次打印。");
            cd.signalAll();
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            lock.unlock();
        }
    }

    public void sub(int total) {
        try {
            lock.lock();
            while (i != 1)
                cd.await();
            Thread.sleep(500);
            System.out.println(Thread.currentThread().getName() + "打印了：" + --i
+ "第" + total + "次打印。");
            cd.signalAll();
        } catch (InterruptedException e) {
            e.printStackTrace();
        } finally {
            lock.unlock();
        }
    }
}
```

2.多个线程按顺序打印

题目：

多线程之间按照顺序调用，实现A-B-C

三个县城启动，要求如下：

AA打印5次，BB打印10次，CC打印15次

接着循环10轮

分析：

1.有顺序通知，需要有标识位
2.有一个锁，lock，3把钥匙condition
3.判断标识位
4.输出线程名+第几次+第几轮
5.修改标识位，通知下一个

```
/**
 * @author yinhuidong
 * @createTime 2020-07-18-18:31
 */
public class Demo4 {
    public static void main(String[] args) {
        ShareData data = new ShareData();
        new Thread(()->{
            for (int i = 0; i < 10; i++)
            data.print5(i+1);

        },"AA").start();
        new Thread(()->{
            for (int i = 0; i < 10; i++)
                data.print10(i+1);

        },"BB").start();
        new Thread(()->{
            for (int i = 0; i < 10; i++)
                data.print15(i+1);

        },"CC").start();
    }
}

class ShareData {
    /**
     * 标识位
     * 1：A
     * 2：B
     * 3：C
     */
    private int num = 1;
    //锁
    private ReentrantLock lock = new ReentrantLock();
    //钥匙
    private Condition cdA = lock.newCondition();
    private Condition cdB = lock.newCondition();
    private Condition cdC = lock.newCondition();

    public void print5(int total) {
        try {
            lock.lock();
            while (num != 1)
                cdA.await();
```

```
            for (int i = 0; i < 5; i++)
                System.out.println(Thread.currentThread().getName() + "打印:" + i
+ "第" + total + "轮");
            num = 2;
            cdB.signal();
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            lock.unlock();
        }
    }
    public void print10(int total) {
        try {
            lock.lock();
            while (num != 2)
                cdB.await();
            for (int i = 0; i < 10; i++)
                System.out.println(Thread.currentThread().getName() + "打印:" + i
+ "第" + total + "轮");
            num = 3;
            cdC.signal();
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            lock.unlock();
        }
    }
    public void print15(int total) {
        try {
            lock.lock();
            while (num != 3)
                cdC.await();
            for (int i = 0; i < 15; i++)
                System.out.println(Thread.currentThread().getName() + "打印:" + i
+ "第" + total + "轮");
            num = 1;
            cdA.signal();
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            lock.unlock();
        }
    }
}
```

二，线程和集合

1.如何证明集合是线程不安全的

```
public class Demo5 {
    public static void main(String[] args) {
        List<String> list=new ArrayList<>();
        for (int i = 0; i < 30; i++) {
        new Thread(()->{
                list.add(UUID.randomUUID().toString().substring(8));
                System.out.println(list);
        },String.valueOf(i)).start();
        }
    }
}
//并发修改异常：java.util.ConcurrentModificationException
```

或者就是查看ArrayList源码：

```
public boolean add(E e) {
    ensureCapacityInternal( minCapacity: size + 1);
    elementData[size++] = e;
    return true;
}
```

2.如何让集合变得安全

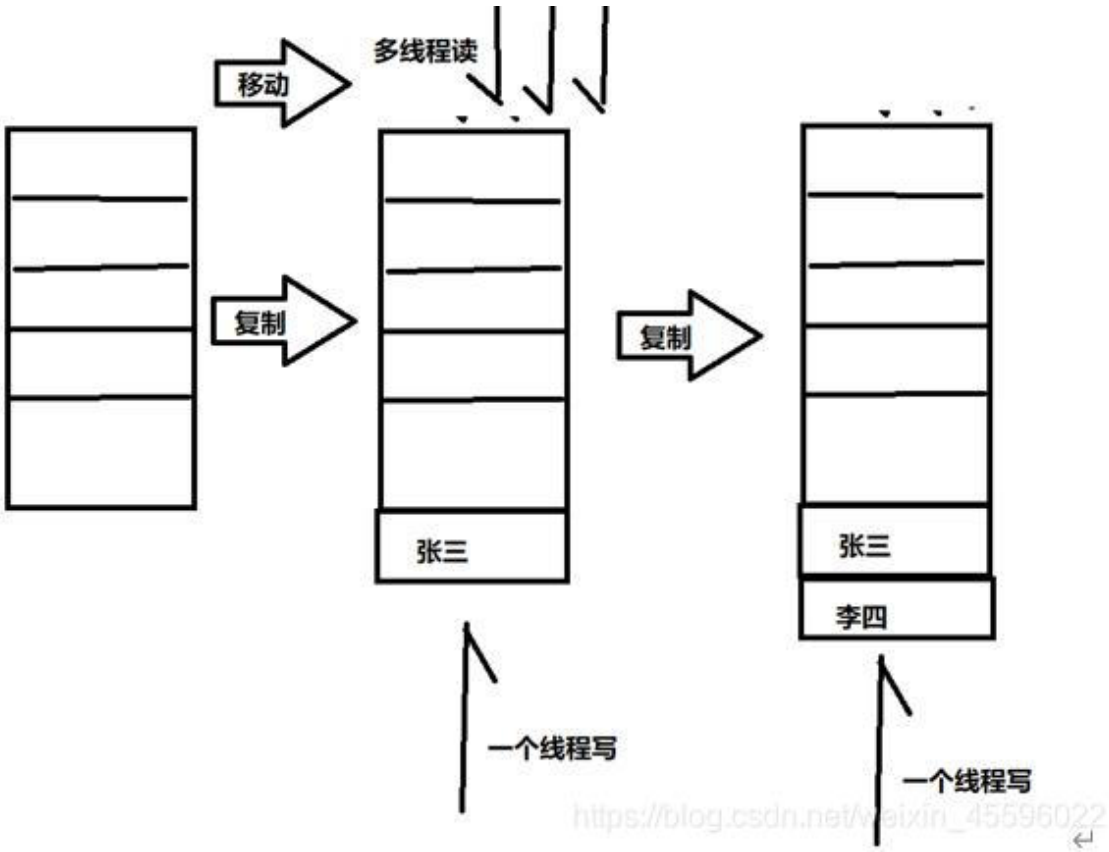
1）调用工具类

```
        List<String> list= Arrays.asList("a","b","c","d");
        List<String> list2= Collections.synchronizedList(list);
```

2）使用JUC

写时复制技术

`CopyOnWriteArrayList`




```
/**
 * @author yinhuidong
 * @createTime 2020-07-18-16:37
 * juc写时复制技术
 * CopyOnWriteArrayList
 */
public class JUCNote7 {
    public static void main(String[] args) {
        CopyOnWriteArrayList<String> list = new CopyOnWriteArrayList<>();
        for (int i = 0; i < 30; i++) {
            new Thread(()->{
                list.add(UUID.randomUUID().toString().substring(8));
                System.out.println(list);
            },String.valueOf(i)).start();
        }
    }
}
```

查看源码

```
    public boolean add(E e) {
        final ReentrantLock lock = this.lock;
        lock.lock();
        try {
            Object[] elements = getArray();
            int len = elements.length;
            Object[] newElements = Arrays.copyOf(elements, len + 1);
            newElements[len] = e;
            setArray(newElements);
            return true;
        } finally {
            lock.unlock();
```

```
        }
    }
```

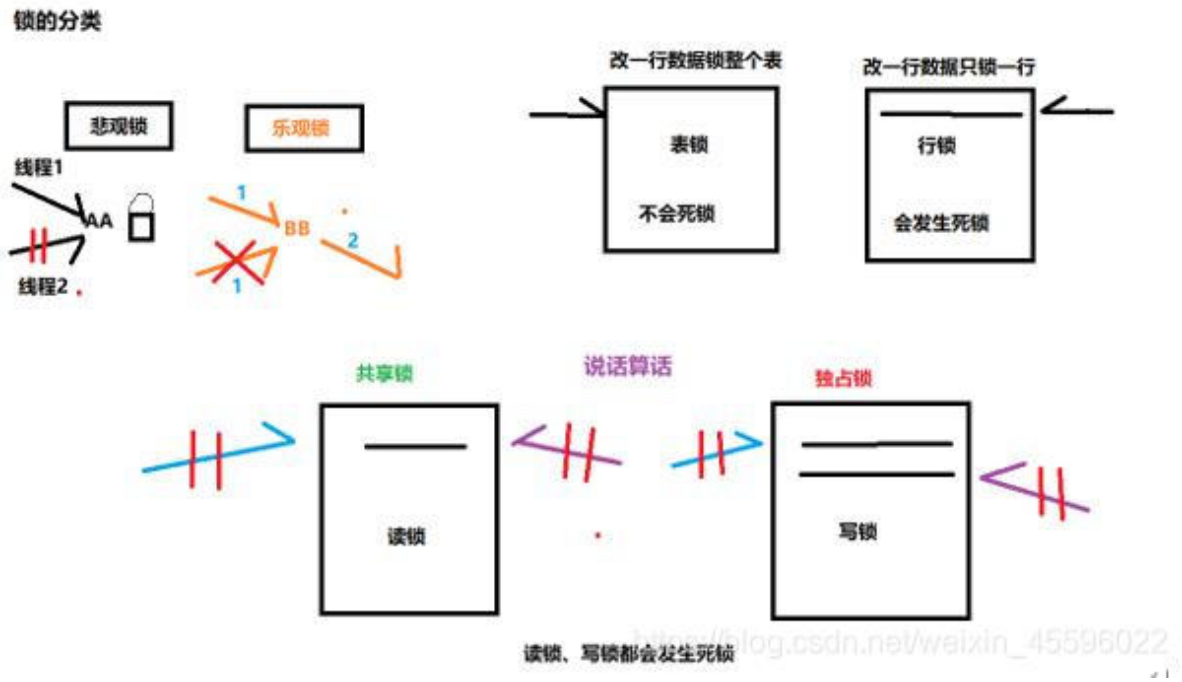
3）面试题

```
/**
 * @author yinhuidong
 * @createTime 2020-07-18-13:45
 * 两个线程，一个县城打印1-52，另一个打印字母A-Z打印顺序为12A34B。。。
 * 65-90
 */
public class JUCNote2 {
    public static void main(String[] args) {
        Printers printers = new Printers();
        new Thread(()->{
           for (int i = 1; i <= 26; i++)
               printers.printNum();

       },"数字打印线程").start();
        new Thread(()->{
            for (int i = 1; i <= 26; i++)
                printers.printLetter(i+64);

        },"字母打印线程").start();
    }
}
class Printers {
    private int num=1;
    private int a=0;
    private ReentrantLock lock=new ReentrantLock();
    private Condition cd1=lock.newCondition();
    private Condition cd2=lock.newCondition();
    public void printNum(){
        try {
            lock.lock();
            while (num!=1)
                cd1.await();
            for (int i = 0; i < 2; i++) {
                System.out.println(Thread.currentThread().getName()+"打印了："+
++a);
            }
            num++;
            cd2.signal();
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            lock.unlock();
        }
    }
    public void printLetter(int aa){
        try {
            lock.lock();
            while (num!=2)
                cd2.await();
                System.out.println(Thread.currentThread().getName()+"打印了："+
(char) aa);
            num--;
```

```
            cd1.signal();
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            lock.unlock();
        }
    }
}
```

三，读写锁

1）锁的分类



2）读写锁

读锁：共享锁

写锁：独占锁

读锁和写锁都会发生死锁

3）案例：ReentrantReadWriteLock

```
public class Demo7 {
    public static void main(String[] args)throws Exception {
        MyCache myCache=new MyCache();
        for (int i = 1; i <=5; i++) {
            int num = i ;
            new Thread(()->{
                myCache.put(String.valueOf(num),String.valueOf(num));
            },String.valueOf(i)).start();
        }
        TimeUnit.SECONDS.sleep(3);
        for (int i = 1; i <=5; i++) {
            int num = i ;
            new Thread(()->{
```

```
                myCache.get(String.valueOf(num));
            },String.valueOf(i)).start();
        }
    }
}
class MyCache{
    //volatile:表示经常变化的
    private volatile Map<String,Object> map=new HashMap<>();
    private ReadWriteLock lock=new ReentrantReadWriteLock();
    public void put(String key,Object val){
        try {
            lock.writeLock().lock();
            System.out.println(Thread.currentThread().getName()+"\t 开始写入数
据"+key+"!!!!!!!!!");
            TimeUnit.SECONDS.sleep(1);
            map.put(key,val);
            System.out.println(Thread.currentThread().getName()+"\t 完成写入数
据"+key+"----------");

        } catch (InterruptedException e) {
            e.printStackTrace();
        }finally {
            lock.writeLock().unlock();
        }
    }
    public void get(String key){
        try {
            lock.readLock().lock();
            System.out.println(Thread.currentThread().getName()+"\t 开始读取数
据"+key+"!!!!!!!!!");
            TimeUnit.SECONDS.sleep(1);
            Object result = map.get(key);
            System.out.println(Thread.currentThread().getName()+"\t 完成读取数
据"+result+"----------");

        } catch (InterruptedException e) {
            e.printStackTrace();
        }finally {
            lock.readLock().unlock();
        }
    }
}
```