链滴

JAVA 多线程之线程间的通信方式

作者：zxd

原文链接：https://ld246.com/article/1496557788692

来源网站：链滴

一，介绍

本总结我对于JAVA多线程中线程之间的通信方式的理解，主要以代码结合文字的方式来讨论线程间通信，故摘抄了书中的一些示例代码。

二，线程间的通信方式

①同步

这里讲的同步是指多个线程通过synchronized关键字这种方式来实现线程间的通信。

参考示例：

```
public class MyObject {

synchronized public void methodA() {
    //do something....
}

synchronized public void methodB() {
    //do some other thing
}
}

public class ThreadA extends Thread {

    private MyObject object;
//省略构造方法
    @Override
    public void run() {
        super.run();
        object.methodA();
    }
}

public class ThreadB extends Thread {

    private MyObject object;
//省略构造方法
    @Override
    public void run() {
        super.run();
        object.methodB();
    }
}

public class Run {
    public static void main(String[] args) {
        MyObject object = new MyObject();

        //线程A与线程B 持有的是同一个对象:object
        ThreadA a = new ThreadA(object);
        ThreadB b = new ThreadB(object);
```

```
        a.start();
        b.start();
    }
}
```

由于线程A和线程B持有同一个MyObject类的对象object，尽管这两个线程需要调用不同的方法，但它们是同步执行的，比如：**线程B需要等待线程A执行完了methodA()方法之后，它才能执行methoB()方法。这样，线程A和线程B就实现了 通信。**

这种方式，本质上就是“共享内存”式的通信。多个线程需要访问同一个共享变量，谁拿到了锁（获了访问权限），谁就可以执行。

②while轮询的方式

代码如下：

```
import java.util.ArrayList;
import java.util.List;

public class MyList {

    private List<String> list = new ArrayList<String>();
    public void add() {
        list.add("elements");
    }
    public int size() {
        return list.size();
    }
}

import mylist.MyList;

public class ThreadA extends Thread {

    private MyList list;

    public ThreadA(MyList list) {
        super();
        this.list = list;
    }

    @Override
    public void run() {
        try {
            for (int i = 0; i < 10; i++) {
                list.add();
                System.out.println("添加了" + (i + 1) + "个元素");
                Thread.sleep(1000);
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

```
import mylist.MyList;

public class ThreadB extends Thread {

    private MyList list;

    public ThreadB(MyList list) {
        super();
        this.list = list;
    }

    @Override
    public void run() {
        try {
            while (true) {
                if (list.size() == 5) {
                    System.out.println("==5, 线程b准备退出了");
                    throw new InterruptedException();
                }
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}

import mylist.MyList;
import extthread.ThreadA;
import extthread.ThreadB;

public class Test {

    public static void main(String[] args) {
        MyList service = new MyList();

        ThreadA a = new ThreadA(service);
        a.setName("A");
        a.start();

        ThreadB b = new ThreadB(service);
        b.setName("B");
        b.start();
    }
}
```

在这种方式下，线程A不断地改变条件，线程ThreadB不停地通过while语句检测这个条件(list.size()==5)是否成立 ，从而实现了线程间的通信。但是**这种方式会浪费CPU资源**。之所以说它浪费资源，是因JVM调度器将CPU交给线程B执行时，它没做啥“有用”的工作，只是在不断地测试 某个条件是否成 。_就类似于现实生活中，某个人一直看着手机屏幕是否有电话来了，而不是： 在干别的事情，当有话来时，响铃通知TA电话来了。_关于线程的轮询的影响，可参考：JAVA多线程之当一个线程在执行循环时会影响另外一个线程吗?

这种方式还存在另外一个问题：

轮询的条件的可见性问题，关于内存可见性问题，可参考：JAVA多线程之volatile 与 synchronized比较中的第一点“**一，volatile关键字的可见性**”

线程都是先把变量读取到本地线程栈空间，然后再去再去修改的本地变量。因此，如果线程B每次都取本地的 条件变量，那么尽管另外一个线程已经改变了轮询的条件，它也察觉不到，这样也会造成死环。

③wait/notify机制

代码如下：

```
import java.util.ArrayList;
import java.util.List;

public class MyList {

    private static List<String> list = new ArrayList<String>();

    public static void add() {
        list.add("anyString");
    }

    public static int size() {
        return list.size();
    }
}


public class ThreadA extends Thread {

    private Object lock;

    public ThreadA(Object lock) {
        super();
        this.lock = lock;
    }

    @Override
    public void run() {
        try {
            synchronized (lock) {
                if (MyList.size() != 5) {
                    System.out.println("wait begin "
                            + System.currentTimeMillis());
                    lock.wait();
                    System.out.println("wait end  "
                            + System.currentTimeMillis());
                }
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

```
public class ThreadB extends Thread {
    private Object lock;

    public ThreadB(Object lock) {
        super();
        this.lock = lock;
    }

    @Override
    public void run() {
        try {
            synchronized (lock) {
                for (int i = 0; i < 10; i++) {
                    MyList.add();
                    if (MyList.size() == 5) {
                        lock.notify();
                        System.out.println("已经发出了通知");
                    }
                    System.out.println("添加了" + (i + 1) + "个元素!");
                    Thread.sleep(1000);
                }
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}

public class Run {

    public static void main(String[] args) {

        try {
            Object lock = new Object();

            ThreadA a = new ThreadA(lock);
            a.start();

            Thread.sleep(50);

            ThreadB b = new ThreadB(lock);
            b.start();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

线程A要等待某个条件满足时(list.size()==5)，才执行操作。线程B则向list中添加元素，改变list 的siz
。

A,B之间如何通信的呢？也就是说，线程A如何知道 list.size() 已经为5了呢？

这里用到了Object类的 wait() 和 notify() 方法。

当条件未满足时(list.size() !=5)，线程A调用wait() 放弃CPU，并进入阻塞状态。---不像②while轮询样占用CPU

当条件满足时，线程B调用 notify()通知 线程A，所谓通知线程A，就是唤醒线程A，并让它进入可运状态。

这种方式的一个好处就是CPU的利用率提高了。

但是也有一些缺点：比如，线程B先执行，一下子添加了5个元素并调用了notify()发送了通知，而此线程A还执行；当线程A执行并调用wait()时，那它永远就不可能被唤醒了。因为，线程B已经发了通了，以后不再发通知了。这说明：**通知过早，会打乱程序的执行逻辑。**

④**管道通信**就是使用java.io.PipedInputStream 和 java.io.PipedOutputStream进行通信

具体就不介绍了。分布式系统中说的两种通信机制：共享内存机制和消息通信机制。感觉前面的①中synchronized关键字和②中的while轮询 “属于” 共享内存机制，由于是轮询的条件使用了volatile键字修饰时，这就表示它们通过判断这个“共享的条件变量“是否改变了，来实现进程间的交流。

而管道通信，更像消息传递机制，也就是说：通过管道，将一个线程中的消息发送给另一个。