1. 等待/通知机制

等待/通知机制能够实现线程间的通信，当线程间进行通信后，系统之间的交互性更强，提高了CPU的利用率。

1. 等待/通知机制的实现

wait()和notify()都是Object类的方法，Java为每个Object都实现了这两个方法。

1. wait()

等待是通过wait()方法实现的，它能够使当前执行代码的线程进入等待状态，并且在wait()所在的代码行处停止执行，直到接到通知或被中断。在调用wait()方法前，线程必须获得该对象的对象级别锁，因此只能在同步方法或者同步代码块中调用wait()方法，如果调用wait()方法时没有持有锁，则抛出IllegalMonitorStateException异常。在调用wait()方法后，当前线程会立即释放锁。同时，在执行同步代码块的过程中，如果遇到了异常导致线程终止，那么对象锁也会释放。

1. wait(long)

带参数的wait(long)方法的功能是等待某一时间内是否有线程对锁进行唤醒，如果超过这个时间则自动唤醒。

1. notify()

通知是通过notify()方法实现的，同样的，它也要在同步方法或者同步代码块中调用，该方法用于通知等待该对象的对象锁的线程，如果有多个线程在等待，则由线程规划器随机选取一个线程，使它获取该对象的对象锁。注意，notify()方法不会立刻释放对象锁，而是要等到notify()方法的线程将同步方法或者同步代码块中的代码执行完。

1. notifyAll()

notifyAll()方法可以使所有处于等待队列中等待同一共享资源的线程从等待状态中退出，进入可运行状态，即唤醒

1. 线程状态切换
2. 当新建一个线程，并调用该线程的start()后，该线程就处于Runnable（可运行）状态，这表示着该线程准备运行。但是还需要等待CPU资源的分配，得到资源后就进入Running(运行)状态。
3. 如果线程被其他高优先级的线程抢占了CPU资源，那么该线程就会从Running状态转变为Runnable状态。进入Runnable状态的4种情况：
4. 调用sleep()方法后，线程进入了阻塞状态，但是如果时间超过了指定的休眠时间，那么就会进入Runnable状态，等待CPU资源。
5. 线程调用的阻塞方法执行完毕，则该线程进入Runnable状态。
6. 线程获得了同步对象监视器。
7. 处于挂起状态的线程调用了resume()恢复方法。
8. Blocked是阻塞的状态，当Blocked状态结束后，进入Runnable状态，等待CPU资源。线程的阻塞的状态有下列情况：
9. 调用sleep()方法，主动放弃CPU资源。
10. 调用阻塞方法还未执行完。
11. 线程试图获取一个对象监视器，但是该监视器被其他线程所持有的。
12. 线程等待某个通知。
13. 调用suspend()方法将该线程挂起，但该方法容易导致死锁，尽量不用。
14. 当线程的run()方法执行结束或者调用stop()方法时，进入销毁状态，整个线程执行完毕。

每个锁对象都有两个队列，一个是就绪队列，存储着将要获得锁的线程；一个是阻塞队列，存储着被阻塞的线程。

1. 生产者/消费者模式
2. 概述

等待/通知模式最经典的案例就是生产者/消费者模式，生产者生产完后通知消费者来消费，消费者消费完后通知生产者继续生产。只有在未消费完的情况下消费者才能消费，同理只有在还未生产的情况下生产者才能继续生产。通过一个变量或者其他条件来判断是否可以继续生产，还是继续等待；是否可以继续消费，或是继续等待。

1. 假死现象

假死现象其实就是线程进入了等待状态，当所有线程都进入等待时，那么程序就无法继续执行任何业务了。这在多生产者，多消费者的情况下会出现，正常情况是生产者唤醒消费者，消费者唤醒生产者。但是notify不能保证唤醒的一定是异类，也可能是同类，当生产者a生产完产品，发出通知唤醒另一个生产者b后，可能b会因为产品还未消费，因此呈等待状态；当消费者c消费完产品发出通知，唤醒消费者d后，d可能会因为还没有产品能够消费而进入等待状态。这样，会让很多的线程不断进入等待状态，导致假死现象。

假死现象产生的原因是连续唤醒同类线程，因此只要我们将其他的异类线程一同唤醒就可以解决这个问题，将notify()方法改为notifyAll()方法，就可以通知到所有的同步线程，包括同类和异类的线程。

1. 线程通过管道进行通信

管道流(pipeStream)可用于在不同线程之间直接传送数据，共有4个类：

PipedInputStream, PipedOutputStream 用于字节流

PipedReader, PipedWriter 用于字符流

1. 字节流

就是使用PipedInputStream作为字节输入流读数据，使用PipedOutputStream作为字节输出流写数据。使用方法与一般的字节输入流，输出流一致，不过在初始化管道输入输出流后，需要通过connect函数连接两个流，产生通信连接，这样才可以将数据通过管道传输。

1. 字符流

字符流只是换了个输入输出函数而已，传输的数据变为了字符数据，而不是字节数据。