创建线程的两种方法：

1. 继承 Thread类 并实现run方法

class A extends Thread{

public void run() { }

}

new A().start();

1. 实现接口Runnable 并实现run方法

class A extends impelements Runnable{

public void run() { }

}

new A().start();

其他方法创建线程方法：

new Thread()

{

public void run()

{

for(int x=0; x<100; x++)

{

System.out.println(Thread.currentThread().getName()+"....."+x);

}

}

}.start();

Runnable r = new Runnable()

{

public void run()

{

for(int x=0; x<100; x++)

{

System.out.println(Thread.currentThread().getName()+"....."+x);

}

}

};

new Thread(r).start();

直接调用run方法和 线程.start()的区别：

调用run()不会启动该线程,作为一般的函数调用，此时线程不会增加

调用start() 启动线程并且 该线程内调用run方法

常用方法：查文档

线程同步问题：

保证操作的数据是同一个

所以可以在 实例化对象的时候 将数据传入

同步的

优势：解决了线程的安全问题。

弊端：

较为消耗资源。

同步嵌套后，容易死锁。

同步的前提：

1，必须要有两个或者两个以上的线程。

2，必须是多个线程使用同一个锁。

此处让我想到了操作系统课里面的诸多问题：

读者写者问题

哲学家用餐问题

过桥问题

生产者消费者问题

突然发现，这个课里面有很多经典算法啊 找时间来啃一啃

PV操作，信号量？

临界区如何设置：一定设置在操作临界资源的两端 一旦不使用资源，立即释放

否则影响其他线程对资源的使用，造成效率降低

同步方法和同步代码块：

Class A{

Synchronized void fun(){ } //synchronized 关键字放在返回值类型前

} //该同步方法的锁 使用的是当前实例对象锁 即this.class锁

Object obj;可以传入任意对象,如果需要实现同步，则需要保证使用同一个obj作为锁

Synchronized(obj){

}

锁(对象监视器)的方法： 使用这些方法 就可以实现线程间通信

wait() ;阻塞线程

notify();唤醒由 锁.wait()阻塞的线程池中 的一个线程，无法指定唤醒某一线程

notifyAll();只能唤醒由 锁.wait()所阻塞的 线程池中的 所有线程

如果 一把锁 调用wait 阻塞了多个线程

Notify无法唤醒某一指定线程

敲黑板，划重点：

线程在进入阻塞状态后(调用了wait方法)，是会释放锁的

如果不释放锁的话,那这个线程就无法被唤醒了

而1.5版本新特性Condition可以 新版本特性如下：

Jdk1.5以后 出现的新特性：Lock类 和synchronized功能相似，但更强大

Lock lo=new ReenTrantLock();

Lo.lock();

Try

{同步代码；}

Finally

{ lo.unlock() }

Lock代替了synchronized的使用

condition代替了对象监视器方法（wait，notify）的上使用

调用Condition的await()和signal()方法，都必须在lock保护之内，就是说必须在lock.lock()和lock.unlock之间才可以使用

await()理解为正在等待signal信号，拿到信号就往下执行，也就被唤醒了

signal理解为发送信号 唤醒线程

同一把锁 拥有可以拥有多个Condition对象：

Condition c1=l.newCondition();

Condition c2=l.newCondition();

如果线程A被c1.wait()阻塞，那么只能通过c1来唤醒

如果线程B被c2.wait()阻塞，那么只能通过c2来唤醒

这样 就可以把使用同一把锁的线程 分割成几个部分

每一部分使用各自的condition来进行 进程状态转换

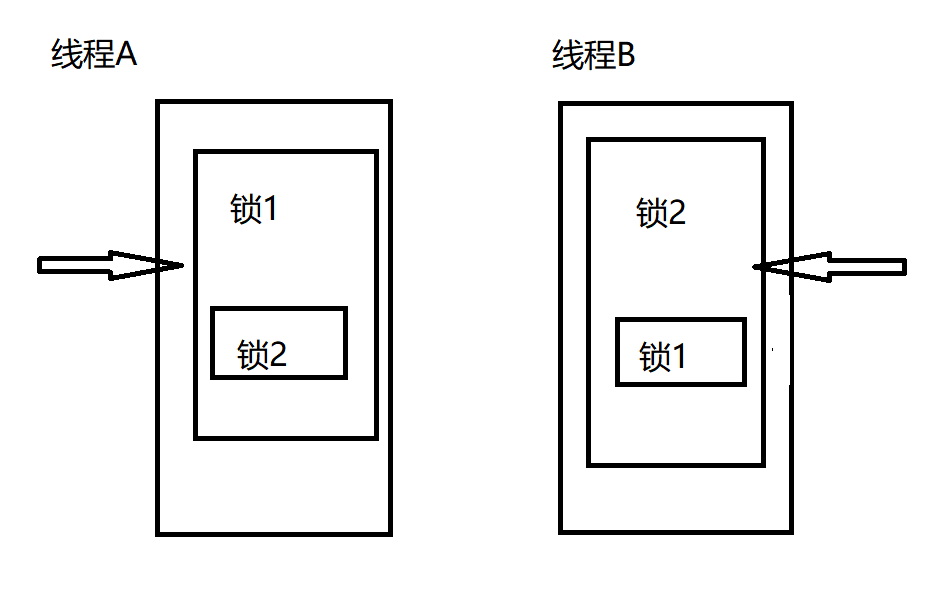
如何终止线程？stop方法已经过时了

此时可以使用信号量 一般run方法中 都使用的循环

如果使用信号量作为循环条件，就可以通过控制信号量来控制循环，从而控制线程

但是如果 线程进入了阻塞状态（即调用了wait方法），此时线程阻塞，无法进行循环判定，无法控制线程 此时调用线程方法interrupt方法 中断线程的阻塞状态（wait和sleep都是阻塞状态，sleep会超时会自动唤醒，wait不会自动唤醒）

死锁：



线程A已经获取锁1，等待锁2释放

此时B已经获取锁2，等待锁1释放

形成僵局，互相等待对方的锁，形成死锁

死锁举例：

