进程：两个相互独立的盒子

线程的生命周期

start() cpu dispatch调用 可能会blocked

new （创建）- ------------>runnable------------------->Running --------------------->blocked

| |

----------------------------------------------------------------|

runnable ，blocked和running都可能进入terminated状态

一个线程不能启动多次，如果线程已经启动，再次启动会抛出异常

1java应用程序的main函数是一个线程，被JVM启动的时候调用，线程的名字叫main

2实现一个线程必须创建Thread实例，override run方法，并且调用start方法

3在JVM启动后，实际上有多个线程，但至少有一个是非守护线程

4当你调用一个线程的start方法时，此时至少有两个线程，一个是调用你的线程，还有一个执行run方法的线程

5 线程的生命周期，分为new，runnable ，running，block，terminate

多个窗口同时工作，但是操作一份数据

可以使用一个类集成Runnable接口，Runnable接口存储实际逻辑，然后将这些类引用这个Runnable接口，然后实现这些接口

创建线程对象默认有一个线程名，以Thread-开头，0开始计数的线程名

☐ Thread-0

☐ Thread-1

☐ Thread-2

如果在构造thread的时候没有传递Runnable或者没有复写Thread的run方法，该Thread将不会调用任何东西，如果传递了Runnable接口或者复写了Thread 的run方法，则执行该方法的逻辑单元（逻辑代码）

如果构造线程对象时没有传入threadGroup，则该Thread的ThreadGroup会默认是父线程的ThreadGroup

JVM

栈 ：非堆

虚拟机栈：每一个方法被执行的时候都会创建，线程私有，StackSize

本地方法区：jni的地方

程序计数器：类似于硬件的缓存，pc缓存器

方法区：存放一些方法的入口和出口，所有线程共享的东西

堆内存：4.构造Thread的时候传入stacksize代表该线程占用的stacksize的大小，如果没有指定stacksize的大小，默认是0,0代表忽略改参数，改参数会被JNI函数调用使用，需要注意：该参数在一些平台有用，在一些平台无效

t.setDaemon(true)

创建守护线程，当同一个ThreadGroup的其余线程被关掉了，守护线程会强制结束

t1.setPriority(Thread.***MAX\_PRIORITY***);

线程可以设置不同的优先级，在不同的优先级，10最大，1最小，优先级只能尽量的使得线程先被执行，不能解决同步问题。

t1.join()

等待当前线程运行结束，等待t1线程技术再调用当前线程,join()必须放在start()以后，并且当前线程与t1不是一个线程，让t1和t2交替进行

Thread.*currentThread*().join();让线程等待自己结束，但是线程一直没有结束，所以程序一直在线，不会退掉

Thread中断interrupt()方法的详细讲解

如果一个线程被wait(),join(),sleep(),中断状态不会被捕获。

强制关闭线程

**public class** ThreadService {

**private** Thread **executeThread**;

**private boolean finished** = **false**;

**public void** execute(Runnable task){

**executeThread** = **new** Thread(){

@Override

**public void** run() {

Thread runner = **new** Thread(task);

runner.setDaemon(**true**);

runner.start();

**try** {

runner.join();

**finished** = **true**;

} **catch** (InterruptedException e) {

}

}

};

**executeThread**.start();

}

**public void** shutDown(**long** mills) {

**long** currentTime = System.*currentTimeMillis*();

**while** (!**finished**){

**if** (System.*currentTimeMillis*() - currentTime >= mills) {

System.***out***.println(**"任务超时，任务结束"**);

**executeThread**.interrupt();

**break**;

}

**try** {

**executeThread**.*sleep*(1);

} **catch** (InterruptedException e) {

System.***out***.println(**"执行线程被打断."**);

**break**;

}

}

}

}

java线程同步

**synchronized（） 类似于加上一个栅栏，让程序串行执行，被sychronized之后的代码没有了多线程并行功能**

被synchronized修饰的方法使用的是This锁，也就是同一个对象中只用同时只能由一个被synchronized修饰的方法可以被调用

死锁

线程sleep和wait的区别

1.sleep是Thread的方法，wait是Object的方法

2.sleep will not release the object monitor，but the wait will release the monitor and add to the Object monitor waiting queue.

3.use sleep not depend on the monitor ,but wait need;

4.the sleep method not need be wakeup,but wait(0) need;

定义锁的时候定义成final类型的

性能低下会不会是开辟线程过多

自定义锁