/\*

进程：是一个正在执行中的程序。

每一个进程执行都有一个执行顺序。该顺序是一个执行路径，或者叫一个控制单元。

线程：就是进程中的一个独立的控制单元。

线程在控制着进程的执行。

一个进程中至少有一个线程。

Java VM 启动的时候会有一个进程java.exe.

该进程中至少一个线程负责java程序的执行。

而且这个线程运行的代码存在于main方法中。

该线程称之为主线程。

扩展：其实更细节说明jvm，jvm启动不止一个线程，还有负责垃圾回收机制的线程。

1,如何在自定义的代码中，自定义一个线程呢？

通过对api的查找，java已经提供了对线程这类事物的描述。就Thread类。

创建线程的第一种方式：继承Thread类。

步骤：

1，定义类继承Thread。

2，复写Thread类中的run方法。

目的：将自定义代码存储在run方法。让线程运行。

3，调用线程的start方法，

该方法两个作用：启动线程，调用run方法。

发现运行结果每一次都不同。

因为多个线程都获取cpu的执行权。cpu执行到谁，谁就运行。

明确一点，在某一个时刻，只能有一个程序在运行。(多核除外)

cpu在做着快速的切换，以达到看上去是同时运行的效果。

我们可以形象把多线程的运行行为在互相抢夺cpu的执行权。

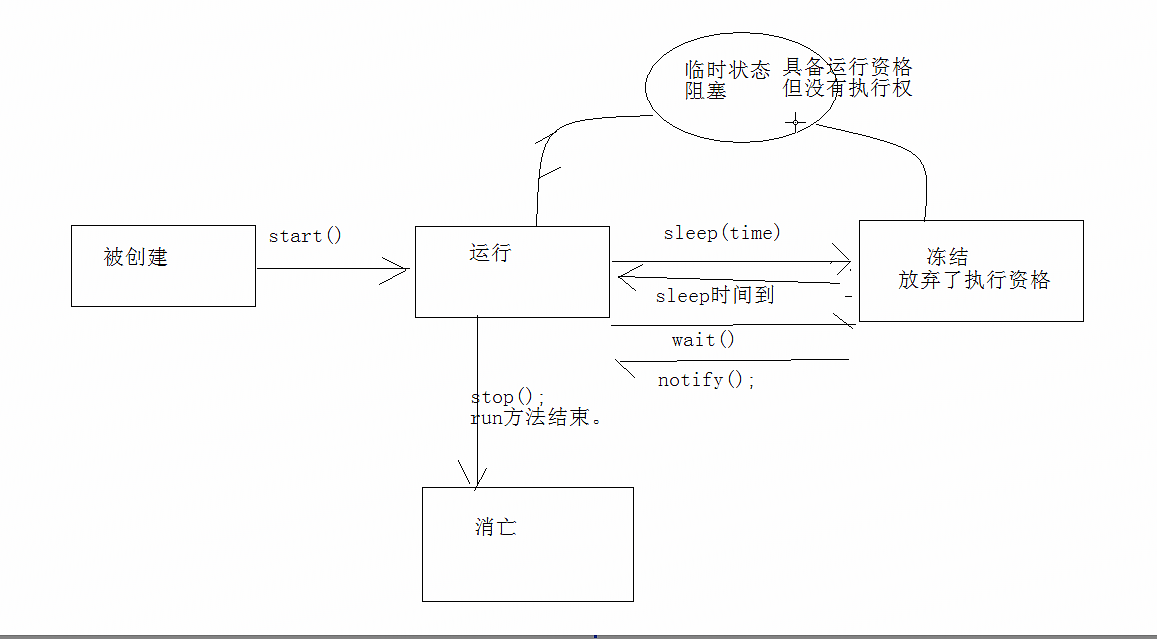
这就是多线程的一个特性：随机性。谁抢到谁执行，至于执行多长，cpu说的算。

为什么要覆盖run方法呢？

Thread类用于描述线程。

该类就定义了一个功能，用于存储线程要运行的代码。该存储功能就是run方法。

也就是说Thread类中的run方法，用于存储线程要运行的代码。



/\*

练习：

创建两个线程，和主线程交替运行。

原来线程都有自己默认的名称。

Thread-编号 该编号从0开始。

static Thread currentThread():获取当前线程对象。

getName(): 获取线程名称。

设置线程名称：setName或者构造函数。

\*/

创建线程的第二种方式：实现Runable接口

步骤：

1，定义类实现Runnable接口

2，覆盖Runnable接口中的run方法。

将线程要运行的代码存放在该run方法中。

3，通过Thread类建立线程对象。

4，将Runnable接口的子类对象作为实际参数传递给Thread类的构造函数。

为什么要将Runnable接口的子类对象传递给Thread的构造函数。

因为，自定义的run方法所属的对象是Runnable接口的子类对象。

所以要让线程去指定指定对象的run方法。就必须明确该run方法所属对象。

5，调用Thread类的start方法开启线程并调用Runnable接口子类的run方法。

实现方式和继承方式有什么区别呢？

实现方式好处：避免了单继承的局限性。

在定义线程时，建立使用实现方式。

两种方式区别：

继承Thread:线程代码存放Thread子类run方法中。

实现Runnable，线程代码存在接口的子类的run方法。

问题的原因：

当多条语句在操作同一个线程共享数据时，一个线程对多条语句只执行了一部分，还没有执行完，

另一个线程参与进来执行。导致共享数据的错误。

解决办法：

对多条操作共享数据的语句，只能让一个线程都执行完。在执行过程中，其他线程不可以参与执行。

Java对于多线程的安全问题提供了专业的解决方式。

就是同步代码块。

synchronized(对象)

{

需要被同步的代码

}

同步（synchronized）

格式：

synchronized（对象）

{

需要同步的代码；

}

同步可以解决安全问题的根本原因就在那个对象上。

该对象如同锁的功能。

对象如同锁。持有锁的线程可以在同步中执行。

没有持有锁的线程即使获取cpu的执行权，也进不去，因为没有获取锁。

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。

必须保证同步中只能有一个线程在运行。

好处：解决了多线程的安全问题。

弊端：多个线程需要判断锁，较为消耗资源，