java 多线程心得

篇一： java 多线程总结

代码改变世界

【运行结果】：

A 运行 0

A 运行 1

然后运行程序，输出的可能的结果如下：

A 运行 0

b 运行 0

b 运行 1

b 运行 2

b 运行 3

b 运行 4

A 运行 1

A 运行 2

A 运行 3

A 运行 4

因为需要用到 cpu 的资源，所以每次的运行结果基本是都不一样的，呵呵。

注意：虽然我们在这里调用的是 start （）方法，但是实际上调用的还是 run （）方法的主体。那么：为什么我们不能直接调用 run （）方法呢？

我的理解是：线程的运行需要本地操作系统的支持。

如果你查看 start 的源代码的时候，会发现：

【可能的运行结果】 ：

线程 A 运行 0

线程 B 运行 0

线程 B 运行 1

线程 B 运行 2

线程 B 运行 3

线程 B 运行 4

线程 A 运行 1

篇二： java 多线程总结

引

如果对什么是线程、什么是进程仍存有疑惑，请先 google 之，因为这两个概念不在本文的范围之内。

用多线程只有一个目的， 那就是更好的利用 cpu 的资源，因为所有的多线程代码都可以用单线程来实现。说这个话其实只有一半对，因为反应“多角色”的程序代码，最起码每

个角色要给他一个线程吧，否则连实际场景都无法模拟，当然也没法说能用单线程来实现：比如最常见的“生产者，消费者模型”。

很多人都对其中的一些概念不够明确，如同步、并发等等，让我们先建立一个数据字典，以免产生误会。

多线程：指的是这个程序 （一个进程）运行时产生了不止一个线程

并行与并发：

并行：多个 cpu 实例或者多台机器同时执行一段处理逻辑，是真正的同时。

并发：通过 cpu 调度算法，让用户看上去同时执行，实际上从 cpu 操作层面不是真正的同时。并发往往在场景中有公用的资源，那么针对这个公用的资源往往产生瓶颈，我们会用 Tps 或者 Qps 来反应这个系统的处理能力。

并发与并行

线程安全：经常用来描绘一段代码。指在并发的情况之下，该代码经过多线程使用，线程的调度顺序不影响任何结果。这个时候使用多线程，我们只需要关注系统的内存，cpu 是不是够用即可。反过来，线程不安全就意味着线程的调度顺序会影响最终结果，如不加事务的转账代码：

```
voidtransfermoney(userfrom,userto,floatamount){

to.setmoney(to.getbalance()+amount);
```

from.setmoney(from.getbalance()-amount);

}

同步： Java 中的同步指的是通过人为的控制和调度，保证共享资源的多线程访问成为线程安全，来保证结果的准确。

如上面的代码简单加入 @synchronized 关键字。在保证结果准确的同时，提高性能，才是优秀的程序。线程安全的优先级高于性能。

好了，让我们开始吧。我准备分成几部分来总结涉及到多线程的内容：

扎好马步：线程的状态

内功心法：每个对象都有的方法 (机制)

太祖长拳：基本线程类

九阴真经：高级多线程控制类

扎好马步：线程的状态

(:java 多线程心得) 先来两张图：

线程状态

线程状态转换

各种状态一目了然，值得一提的是 "blocked" 这个状态：

线程在 Running 的过程中可能会遇到阻塞 (blocked) 情况

调用 join() 和 sleep() 方法， sleep() 时间结束或被打断， join() 中断 ,Io 完成都会回到 Runnable 状态，等待 JVm

的调度。

调用 wait() ，使该线程处于等待池 (waitblockedpool), 直到 notify()/notifyAll() ，线程被唤醒被放到锁定池 (lockblockedpool) ，释放同步锁使线程回到可运行状态 (Runnable)

对 Running 状态的线程加同步锁 (synchronized) 使其进入(lockblockedpool), 同步锁被释放进入可运行状态 (Runnable) 。

此外，在 runnable 状态的线程是处于被调度的线程, 此时的调度顺序是不一定的。 Thread 类中的 yield 方法可以让一个 running 状态的线程转入 runnable 。

内功心法：每个对象都有的方法 (机制)

synchronized,wait,notify 是任何对象都具有的同步工具。让我们先来了解他们 monitor

他们是应用于同步问题的人工线程调度工具。讲其本质, 首先就要明确 monitor 的概念， Java 中的每个对象都有一个监视器，来监测并发代码的重入。在非多线程编码时该监视器不发挥作用，反之如果在 synchronized 范围内，监视器发挥作用。

wait/notify 必须存在于 synchronized 块中。并且， 这三个关键字针对的是同一个监视器 (某对象的监视器)。这意味着 wait 之后，其他线程可以进入同步块执行。