并发

66、同步访问共享的可变数据

(1) 关键字 synchronized 可以保证在同一时刻，只有一个线程可以执行某一个方法，或者某一个代码块。

(2) 如果没有同步，一个线程的变化不能被其他线程看到。

(3) Volatile 变量可用于提供线程安全，但是只能应用于非常有限的一组用例：多个变量之间或者某个变量的当前值与修改后值之间没有约束。

(4) 简而言之，当多个线程共享可变数据的时候，每个读或者写数据的线程必须执行同步。

67、避免过度同步

(1) 第 66 条告诫我们缺少同步的危险性，本条目关注相反的问题，根据情况的不同，过度同步可能会导致性能降低、死锁、甚至不确定的行为。

(2)本条建议

a.开放调用：即将外来代码的调用放到同步区域之外，同步区域只执行尽可能少的工作

b.如果一个类不是为了并发而设计的，那么不要使用内部同步（内部类的设计的时候使用同步的方法），而是让客户自己在必要时刻进行外部同步。

68、executor 和 task 优于线程

(1)现在应当优先使用 executor 和 task 而不是 Thread 或者 Runnable，因为在以前 Thread 既是工作单元又是执行单元，但是现在二者分离，executor 是执行单元，Runnable 和 Callable 是工作单元

(2)使用 ScheduledThreadPool 代替 Timer 作为多线程的定时器

(3)对于 ThreadPool：

a.使用 Executor.newCachedThreadPool 来应对一个轻负载的服务器

b.使用 Executor.newFixedThreadPool 来应对一个重负载的服务器

c.为了最大限度控制，直接使用 ThreadPoolExecutor

69、并发工具优先于 wait 和 notify

(1)Java 并发包包含 Executor Framework、并发集合以及同步器三大部分

(2)使用同步器而不是 wait 和 notify

(3)使用 wait 和 notify 的建议：

a.使用 wait 的正确方式：在循环中调用 wait

```
synchronized(obj){
while(<condition does not hold>){
        obj.wait();
    }
}
```

b.使用 notifyAll 而不是 notify 避免恶意访问

70、线程安全性的文档化

(1) 线程安全的级别：

a.不可变：类的实例是不可变的，也就是无需同步，也不会出并发问题

b.无条件的线程安全：类的实例是可变的，但是实现了足够的内部同步，可以并发，例如并发包内的集合类

c.有条件的线程安全：除了有些方法为了进行安全的并发而使用外部同步之外，线程安全级别和无条件的线程同步相同，例如 Collections.synchronized 返回的类，其 iterator（迭代器）需要外部同步

d.非线程安全：这个类的实例是可变的，为了并发使用它们，需要客户代码对其进行外部同步每个方法，例如一般的集合类

e.线程对立：不能安全地被多个线程并发，即所有的方法调用都要被外部同步包围。

(2)每个类都应该说明或者使用线程安全注解，清楚地在文档中说明它的线程安全属性。有条件的线程安全类必须在文档中指明“哪个方法调用系列需要外部同步，以及在执行这些序列的时候获得哪把锁”。如果编写的是无条件的线程安全类，就应该考虑使用私有锁对象来代替同步的方法。

71、慎用延迟初始化

(1)延迟初始化就是延迟到需要域的值是才将它初始化的行为。

(2)对于延迟初始化，最好建议“除非绝对必要，否则就不要那么做”。延迟化降低了初始化类或者创建实例的开销，却增加了访问被延迟初始化的域的开销。

(3)如果域只是在类的实例部分被访问，并且初始化这个域的开销很高，可能就值得进行延迟初始化

(4)简而言之，大多数的域应该正常地进行初始化，而不是延迟初始化。

72、不要依赖于线程调度器

(1) 当有多个线程可以运行时，由线程调度器决定哪些线程将会运行，以及运行多长时间。但是，编写良好的的程序不应该依赖于线程调度器。因为任何依赖于线程调度器来达到正确性和性能要求的程序，很有可能都是不可移植的。

(2) 要编写健壮，响应良好的，可移植的多线程应用程序，最好的办法是确保可运行线程的平均数量不明显多于处理器的数量。这使得线程调度器没有更多的选择：它只需要运行这些可运行的线程，直到他们不再可运行为止。

(3)保持可运行线程数量尽可能少的主要方法是：让每个线程做些有意义的工作，如果线程没有在做有意义的工作，就不应该运行。

(4) 不要依赖 Thread.yield 改变优先级，因为它在不同环境下效果不同，而且实现也不同

(5) 不要使用调整线程优先级的方法，它不具有可移植性

(6)简而言之，不要让应用程序的正确性依赖于线程调度器，否则，结果得到的应用程序既不健壮也不具有可移植性。

73、避免使用线程组

线程组已经是一个废弃的类，在 Java1.5 之后不应当使用