**Java Concurrency / Multithreading Tutorial**

**Java并发/多线程教程**

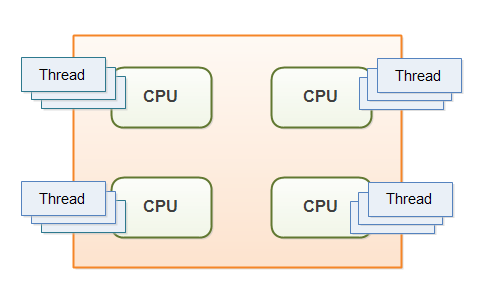
* [Multithreading and Concurrency in Java](http://tutorials.jenkov.com/java-concurrency/index.html#multithreading-and-concurrency-in-java)
* [Java Concurrency in 2015 and Forward](http://tutorials.jenkov.com/java-concurrency/index.html#java-concurrency-in-2015-and-forward)

在过去一台电脑只有一个CPU，并且一次只能够执行一个单一程序。随后而来的多任务技术，意思是计算机可以同时执行多个程序(也叫任务或进程)。然而它并不是真的“同一时间”。一片CPU在程序之间共享。操作系统在程序运行之间切换，执行每一个程序一小段时间然后切换。

随着多任务而来的是对于软件开发者新的挑战。程序不再能够假设全部的CPU时间可用，内存或其他电脑资源也是如此。一个“良好公民”程序应当在不再使用后释放所有资源，以便其他程序使用。

稍后到来的多线程技术，意思是你能够在同一个程序中拥有多个执行线程。一个执行线程可以被视作一个CPU执行程序。当你有多个线程执行同一个程序时，就好像多个CPU在同一个程序内执行。

多线程技术可以是一种很伟大的方式提高某些类型的程序的性能。然而，多线程技术比多任务技术更具有挑战性。多个线程在同一个程序中执行，因此也同时读写相同的内存。这可能导致单线程程序中见不到的错误。这些错误中的某些可能在单CPU机器上见不到，因为两个线程从不真正“同时”执行在现代计算机上，然而，在多核CPU甚至多个CPU上是同时的。这意味着独立的线程可能被独立的核心或者CPU同时执行。



如果一个线程读取一个内存地址同时另一个线程写入它，第一个线程最终将会读取出什么值？老的值？被第二个线程写入的值？或者一个介于二者之间的混合？或者，如果两个线程同时写入相同一个内存地址，当他们完成后最终留下的是什么值？第一个线程写入的值？第二个线程写入的值？抑或写入的两个值的混合？

如果没有合适的预防措施任何输出皆有可能。行为甚至是不可预测的。输出可能随着时间改变。因此对于开发者来说，知道如何采取正确的预防措施 – 意味着学会控制线程如何访问共享资源如内存、文件、数据库等，是很重要的。那就是这个Java并发教程的主题之一。

## Multithreading and Concurrency in Java

Java是第一批使多线程技术简单提供给开发者的语言之一。从一开始Java就拥有多线程能力。因此，Java开发者经常面对上述描述到的问题。那就是为什么我写这篇基于Java并发的教程。作为笔记给我自己，和可能从它收益的全部同仁Java开发者。

教程将主要涉及Java中的多线程技术，但一些发生在多线程部分的问题与发生在多任务和分布式系统中的问题类似。因此多任务和分布式系统的参考文献也可能出现在这篇教程中。因此单词“并发”而非“多线程”更贴切。

## Java Concurrency in 2015 and Forward

从第一批Java并发书籍编写开始，甚至从Java5并发工具包放出开始算，并发体系和设计的世界发生了很多改变。

新的、异步的“无共享”平台和API例如Vert.x和Play，Akka和Qbit已经出现。这些平台使用的并发模型不同于标准Java/JEE的线程、共享内存和锁机制的并发模型。新的非阻塞并发算法已经发布，新的非阻塞工具如Lmax干扰器加入到我们的工具包中。新的函数式编程并行性引入到Java7的Fork与Join框架，集合流API引入Java8。

伴随着所有这些新的发展，是时候我更新下这篇Java并发教程了。因此，这篇教程再一次**开始更新**。新的教程无论何时当能够写他们的时候将会发布。