高并发程序设计

<http://tutorials.jenkov.com/java-concurrency/concurrency-vs-parallelism.html>

<http://blog.csdn.net/net19880504/article/details/20807403>

# 一、 明确概念

## 1 并发与并行

1 并发（Concurrency）其反义词是sequential(顺序执行)

并行(Parallelism) ,其反义词是Serial（串行）

2 并发：一个cpu交替处理任务的能力,cpu对任务进行切换

并行：多个cpu（前提是多个），每个cpu各处理一个任务，同时执行

如果按照任务的角度：

并发：交替执行的是不同的任务，（顺序执行是一个执行完，再执行另外一个）

并行：任务分解，同时执行的是一个任务的子任务。（串行是任务不能分解）

参考：<http://tutorials.jenkov.com/java-concurrency/concurrency-vs-parallelism.html>

## 2 同步和异步

同步（Synchronous） 异步(Asynchronous)

二者形容的一次方法调用。

同步：调用一个方法，等待结果返回。函数的调用和执行是同一个线程。

异步：调用一个方法，不等待结果返回，自己接着其他的工作。在函数内部会**重启一个线程**来执行任务，当任务结束后，使用回调(callback)**通知**调用者或者Future （java中）或者A Message（Akka）

## 3 阻塞和非阻塞

通常形容**多线程**间访问共享资源的相互影响

阻塞：如访问临界区：锁、IO，一个线程在使用，另外一个线程**等待资源（阻塞）**，此时线程**被挂起**，放弃cpu。

非阻塞：一个线程在使用临界区，另外一个线程也来使用，当看到临界区被占用后，不是傻傻的阻塞等待，而是**立刻返回**，做其他事情。

## 4 同步IO和异步IO

我的理解：把IO当成了方法调用。

## 5阻塞IO和非阻塞IO

我的理解：多线程同时访问IO

# 二、java并行程序基础

# 三、JDK并发包

解决的问题是：

1 多线程对临界区资源的访问：

a同一时间只允许一个线程进行访问

b 对临界区的修改，其他线程可见。

2 多线程之前的通信：等待和通知

## 1 ReentrantLock重入锁

与Synchronized相比，有哪些改进思想？

1 具有Synchronized相同的功能，多线程访问临界区资源互斥

2 中断响应：

**阻塞的线程(等待获取资源的线程)**能被中断（Synchronized只能傻傻的等待）

api: lockInterruptibly

3 有限时间等待

Synchronized只会无限期等待

Api： tryLock() / tryLock(long time, TimeUnit unit)

4 公平锁

## 2 Condition

1 Condition代替的Object，实现的线程间的互相通信（等待->通知）

2 在await方法调用后，会释放已经获得的锁

## 3 Semaphore信号量

锁只允许一个线程访问资源；信号量为多个线程

## 4 LockSupport

随时随地挂起一个线程，内部是许可的机制。

LockSupport.park() 和LockSupport.unpark() 次序可以不分先后。

# 四、线程池

解决问题： 线程复用的问题