并发

Java Web 开发中，最常见的是 Servlet 就是多线程的一个例子。

实现并发的方式有两种：一种是进程（操作系统级别），内存和 IO 是独立、互不干涉的；另一种是线程，会共享内存和 I/O 等资源，对操作系统是透明性的，可以“编写一次，到处运行”，同时实现“松散耦合”的设计。

关于 CPU 核数问题：单核 CPU 机器编写多任务的程序任意时刻只能执行一项任务。多核 CPU 机器是处理多任务和多线程最合理的方式。

Executor 管理 Thread 对象：单个的 Executor 被用来创建和管理系统中所有的任务。常见用法：

```
ExecutorService exec = new Executors.newCachedThreadPool();
for i in rang(n):
    exec.execute(Runnable 接口)
exec.shutdown();
```

shutdown()方法可以防止新任务提交给 Executor。

exec.isTerminated()判断线程是否中止。

多线程类的实现方式： (1)、实现 Runnable 接口并编写 run()方法；(2)、直接继承 Thread 类，覆盖 run()方法；(3)、实现 Callable 接口，并编写 call()方法（从任务中产生返回值）。(1)、(2)是通过 exec.execute(Runnable 接口)，(3)是通过 exec.submit(Future 对象)

JAVA 并发的类库：

java.util.concurrent.*

concurrent:adj 并发的　concurrency:n 并发

常用的时间类：

休眠 1s　TimeUnit.MILLISECONDS.sleep(1000)　MILLISECONDS 毫秒；

TimeUnit.SECONDS.sleep(1)

java.lang.Thread 类（ 注：JAVA 中 java.lang 类库会自动导入到每个 java 中）

Thread.currentThread()的 toString → Thread[pool-1-thread-5,10,main]，其中

pool-1-thread-5：是线程的名称，1 是线程池的编号，5 是线程的 id 号

10：线程的优先级

main：所属线程租

线程的优先级：设置为 MAX_PRIORITY 的线程被线程调度器优先选择，优先级较低的线程的执行频率较低。JDK 提供 10 个优先级，唯一可移植的方法是当调整优先级，只使用 MAX_PRIORITY、NORM_PRIORITY 和 MIN_PRIORITY（实际中作用感觉不大？？）

daemon 线程：后台线程或者叫守护线程

1、当所有的非后台线程结束时，程序也就终止了，并且杀死进程中所有后台线程。

2、基于 daemon 线程，创建的所有子线程，默认会被设置为后台线程。

join 用法：加入一个线程

一个线程上调用 join（）方法，比如：主线程调用 t.join()，则要等到 t 线程执行完毕，

主线程才会继续执行。

线程租：*"最好把线程租看成是一次不成功的尝试，你只要忽略它就好"*

异常：默认情况下，线程中的异常一旦逃出 run()方法，就会传播到控制台。处理器处理线程的异常的顺序是：先检查线程专有版本有没有设置 uncaughtException()，再检查线程组是否有专有的 uncaughtException()，最后，调用默认的 defaultUncaughtExceptionHandler。

解决共享资源竞争：关键词：synchronized、lock、volatile

synchronized:将要同步的方法标记为 synchronized 可防止资源冲突。另一种是，建立分离出来的代码段称为临界区。

```
synchronized(syncObject){
    //同步控制模块
}
```

syncObject：如果只有一个同步模块，则直接用 this，多个需要在类中创建多个对象。

lock:Lock 对象必须被显示地创建、锁定和释放。

```
private Lock lock = new ReentrantLock();
//临界资源开始
lock.lock();
...
Lock.unlock();
//临界资源结束
```

synchronized 和 lock 对比：lock 可以更细粒度的控制临界资源，比如：当临界资源被占有时，synchronized 如果要访问该资源，必须要等待直至这个锁释放。而 lock 允许尝试获取该锁而未获取，则可以离开去执行其他事情。

```
Private ReentrantLock lock = new ReentrantLock();
Boolean captured = lock.trylock();
...
lock.unlock();
```

long 和 double 的非原子性:JVM 将 64 位（long 和 double）的读取和写入操作当作两个分离的 32 位操作来执行，产生一个读取和写入操作间发生上下文切换，导致不同任务看到不同的结果的可能性。

volatile:将写操作立即写入到主内存中，而读操作就发生在主存中。