J.U.C

一. tools (同步工具类)

1. CountDownLatch (倒计时器): 用于一个线程或多个线程等待其他线程完成后再执行。(例如: 火箭发射前,需进行各种检查,检查合格)
用于协调多个线程进行同步。
构造时设定计数值,当计数值归零,所有阻塞线程恢复执行。

2. CyclicBarrier (循环栅栏): 与上面类似,但是比他复杂,功能更多。它允许一组线程相互等待,直到达到某个公共屏障点,而且在线程释放后,可以循环重用。
构造时设定等待线程数,当所有线程到达栅栏后,全部放行。

3. Semaphore: 可以控制多个线程同时访问某个资源。(用于控制共享资源的访问数量)

4. Executors 可用创建线程池. 5. Exchanger: 用于两个线程之间交换数据。

二. locks (锁):

ReentrantLock: 是独占锁(排他),且可重入锁。可实现公平锁,等待可中断,Condition 可把多个条件绑定到一个锁上。
内部维持3个类. Sync, NonfairSync, FairSync。分别实现 tryRelease 和 tryAcquire()

Lock: 是个接口: 常用方法 lock(), unLock(), tryLock() (尝试拿锁) 一定要在 finally 中释放锁,不然可能出现严重问题。

ReentrantReadWriteLock: 允许多个读线程同时访问,不允许多个写线程或读写线程同时访问。相对于排他锁,提高了读并发,但是不能共享写锁。
内部维护了两个锁: 一个读锁,一个写锁。(适用于读操作多的场景) 且可重入锁。

Condition: 通过 Condition condition = lock.newCondition(); 常用方法 await(), Signal();

LockSupport: 提供了线程基本原语,通过调用 Unsafe (操作系统级别) 实现线程阻塞和唤醒 (Park() 和 unPark())。

三. executor (线程池)

Callable: 类似 Runable, 但是有返回值。一般配合 ExecutorService 配合使用。通过 Submit() 方法, 返回 future 对象。

Future: 表示异步任务,用于获取还未完成的任务的结果。可以获取 Callable 任务执行的状态, 或者可以停止该任务, 或获取任务结果 (阻塞直到 callable 任务完成) 方法: Cancel() 取消任务。IsDone() 判断是否完成。get() 获取结果, 阻塞等待任务完毕。

FutureTask: 实现 RunnableFuture 接口, 该接口由 Runable 和 Future 组成。所以 FutureTask 既可以作为 Runable 实现类线程, 又可以作为 Future 获取任务结果。(可以认为是 Runable 和 Future 结合) 所以能和 Callable 组合, 完成线程创建。

ExecutorService: 可通过 Executors. { newFixedThreadPool(), newCachedThreadPool(), newSingleThreadPool(), newScheduledThreadPool(), } 阿里巴巴建议使用: ThreadPoolExecutor 创建线程池。

四：Collections

ArrayList 对应：CopyOnWriteArrayList：是线程安全的ArrayList。为啥安全：内部有个互斥锁Lock，用于COWAL的互斥访问。

(线程安全)(维持一个volatile数组) 为啥叫CopyOnWrite：里面维持了一个volatile数组(实际上)，在进行添加/修改/删除，会 新建一个数组并将更新后的数据拷贝到新数组，最后将该数组赋值给"volatile"数组。

(线程安全)(依赖于Lock保证线程安全)(写时复制) 适用于读多写少，缺点：造成内存占用很大。

HashSet 对应：CopyOnWriteArraySet 是线程安全。但不同的是，HashSet 底层是HashMap。但CopyOnWriteArraySet 底层是 CopyOnWriteArrayList。

ConcurrentHashMap：线程安全Map。用于解决HashMap线程不安全，Hashtable效率不高的问题。

为啥Hashtable效率不高，它是对整个Hash表加锁。使synchronized。

JDK1.7 之前，ConcurrentHashMap 采用分段锁，将整个Hash表，分成 Segment数组，(Segment 存放多个 HashEntry) 指定到哪个 就对 某个Segment加锁。就是采用 ReentrantLock。所以说最大并发受数组大小影响。

(Segment 天生继承 自锁，自带锁 lock)

JDK1.8中：改用Node数组+链表+红黑树实现，加锁改为采用CAS操作和synchronized。

当链表长度超过8，会自动转成红黑树，小于6变回数组。

总结：①用Node数组+链表+红黑树 取代了 segment。

②用 CAS+Synchronized 取代了 ReentrantLock。

ConcurrentSkipListMap：替代TreeMap，TreeMap基于红黑树。而它基于跳表。(支持排序 高并发)

ListSet：也是一样，替代TreeSet。

跳表：是一种随机化数据结构，可进行二分查找的有序链表。通过建立多级索引)，实现快速查找。时间复杂度=红黑树 logn，实现简单相比红黑树。

queue：(非阻塞队列)

ConcurrentLinkedQueue：支持高并发，线程安全，是一个基于链表的无界非阻塞队列。通过CAS保证安全，因为没有加锁 速度奇快。加入队头offer()，取出队尾poll()，维持头和尾两个指针，head和tail。

(阻塞队列)

BlockingQueue:
- LinkedBlockingQueue：基于链表的(无界)阻塞队列。
- ArrayBlockingQueue：基于数组 有界阻塞队列。(因为是Array所以有界)
- PriorityBlockingQueue：支持优先级排序的无界阻塞队列。
- DelayQueue：使用优先级队列实现的无界阻塞队列。
- TransferQueue:
- SynchronousQueue：没有容量的队列，生产出来的东西必须马上消费。

何为阻塞队列：当队列为空不得取元素，当队列满了不得加元素，否则进入等待状态。

五：aotmic (原子类)

线程运行状态：①新创建（新创建一个线程）②可运行（包含运行和等待CPU时间片）③阻塞（等待其他线程释放锁资源）

④等待（可能wait了，等待notify）⑤限时等待（如：sleep）⑥死亡（线程结束）

创建线程四种方式：①实现Runable接口 ②继承Thread类。③实现Callable，创建FutureTask通过new Thread(future)实现。

④通过线程池+Callable实现future。

守护线程：Daemon，让某个线程成为守护：thread.setDaemon(true)。main是主线程，不是守护线程。

sleep()：会抛出InterruptedException.

线程停止方法：①stop 太暴力，不建议使用

②interrupt()：设置线程中断状态标记。isInterrupt()：判断是否中断。isInterrupted()（静态方法）判断是否中断并清除。

在线程中循环判断是否中断：
- ①异常停止。（抛异常）
- ②沉睡中停止。（线程sleep被唤醒时，停止会抛出异常。
- ③直接return。

在线程外设置中断标志。

③

并行：单位时间，并发：同一时间段

上下文切换：CPU是采用轮转法为每个线程分配时间片，当线程执行完时间片，需要给其他线程去执行。而这时需要保存当前状态（任务），从线程保存到加载这一过程，称为上下文切换。

Synchronized和ReentrantLock区别：①Synchronized依赖于jvm，ReentrantLock依赖于API

②①可实现公平锁 ②等待可中断 ③锁条件更细粒度，更加灵活 ④Synchronized是关键字，ReentrantLock是调用方法

相同：①都是用于同步保证线程。②都是可重入锁。

Volatile：①保证变量可见性。（使线程从公共内存取值，不从自己的私有内存取值）

②防止指令重排。（指令流水线，打乱指令顺序）

Synchronized和Volatile区别：①一个是保证同步，一个是保证变量可见。

②一个能作用于变量，一个能作用于同步块或方法等。

③一个只保证了可见性，一个既保证可见性又保证原子性。都保证了有序性。

ThreadLocal：为访问这个变量的每个线程创建变量副本，每个线程变量都是独立的，从而避免线程安全的问题。

线程池作用：①避免频繁创建和销毁线程。②提高线程的管理。

execute()和submit()区别，Runable()和Callable()区别

原子类：AtomicInteger：每个操作都是原子性的。底层使用CAS+Volatile保证原子性

AQS：用来构建锁或者其他同步组件的基础框架。是一个抽象类，通过实现模板方法实现一套锁的模板。
（队列同步器）

内部维护 1.队尾 Node head 2.队尾 Node tail 3. state 锁状态，0表示未锁