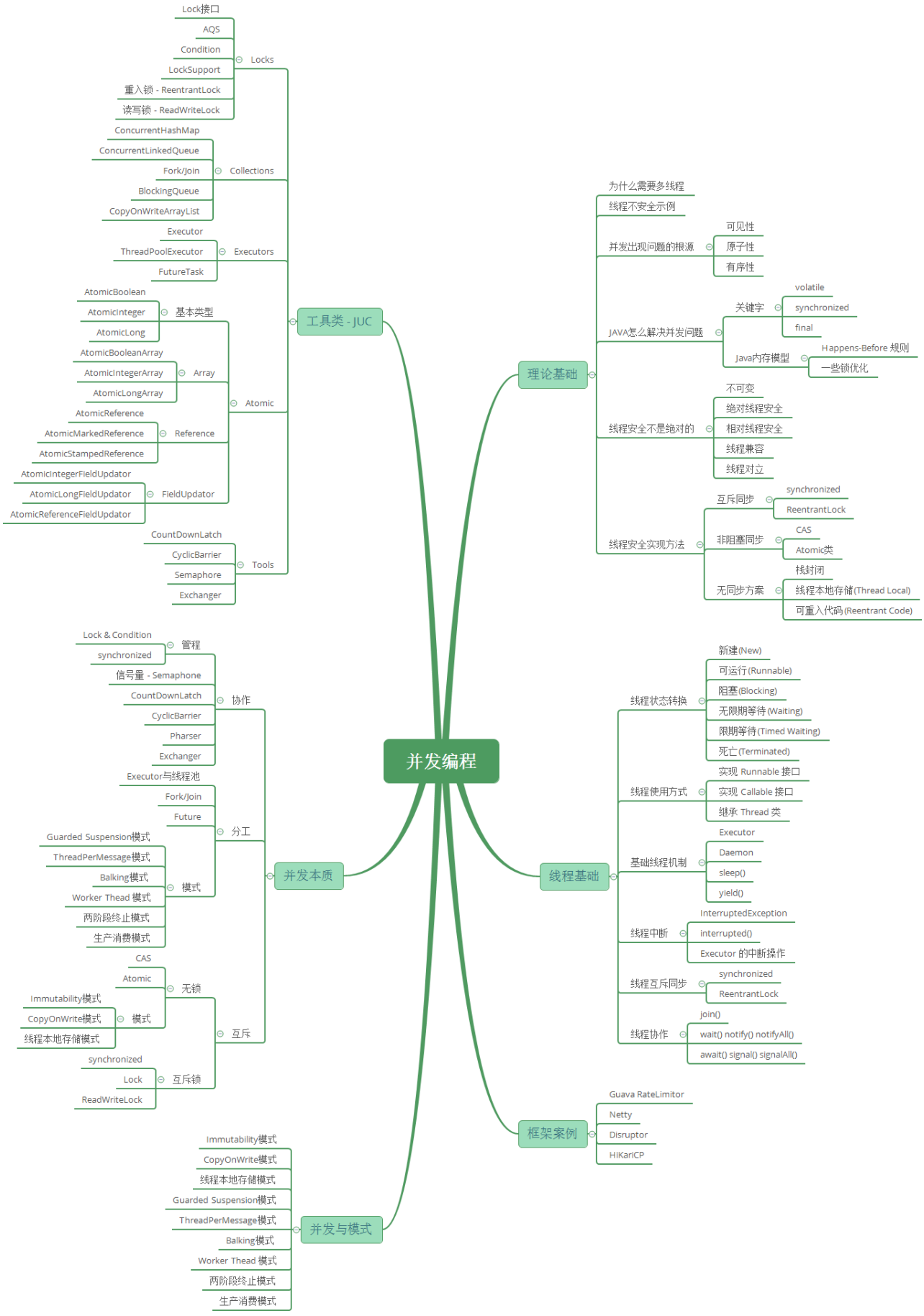
♥Java并发知识体系详解♥

知识体系



相关文章

- Java 并发 - 理论基础

- 多线程的出现是要解决什么问题的?
- 线程不安全是指什么? 举例说明
- 并发出现线程不安全的本质什么? 可见性，原子性和有序性。
- Java是怎么解决并发问题的? 3个关键字，JMM和8个Happens-Before
- 线程安全是不是非真即假? 不是
- 线程安全有哪些实现思路?
- 如何理解并发和并行的区别?

- Java 并发 - 线程基础
  - 线程有哪几种状态? 分别说明从一种状态到另一种状态转变有哪些方式?
  - 通常线程有哪几种使用方式?
  - 基础线程机制有哪些?
  - 线程的中断方式有哪些?
  - 线程的互斥同步方式有哪些? 如何比较和选择?
  - 线程之间有哪些协作方式?
- 关键字: synchronized详解
  - Synchronized可以作用在哪里? 分别通过对象锁和类锁进行举例。
  - Synchronized本质上是通过什么保证线程安全的? 分三个方面回答：加锁和释放锁的原理，可重入原理，保证可见性原理。
  - Synchronized由什么样的缺陷？ Java Lock是怎么弥补这些缺陷的。
  - Synchronized和Lock的对比，和选择?
  - Synchronized在使用时有何注意事项?
  - Synchronized修饰的方法在抛出异常时,会释放锁吗?
  - 多个线程等待同一个snchronized锁的时候，JVM如何选择下一个获取锁的线程?
  - Synchronized使得同时只有一个线程可以执行，性能比较差，有什么提升的方法?
  - 我想更加灵活地控制锁的释放和获取(现在释放锁和获取锁的时机都被规定死了)，怎么办?
  - 什么是锁的升级和降级? 什么是JVM里的偏斜锁、轻量级锁、重量级锁?
  - 不同的JDK中对Synchronized有何优化?
- 关键字: volatile详解
  - volatile关键字的作用是什么?
  - volatile能保证原子性吗?
  - 之前32位机器上共享的long和double变量的为什么要用volatile? 现在64位机器上是否也要设置呢?
  - i++为什么不能保证原子性?
  - volatile是如何实现可见性的？ 内存屏障。
  - volatile是如何实现有序性的？ happens-before等
  - 说下volatile的应用场景?
- 关键字: final详解
  - 所有的final修饰的字段都是编译期常量吗?
  - 如何理解private所修饰的方法是隐式的final?
  - 说说final类型的类如何拓展? 比如String是final类型，我们想写个MyString复用所有String中方法，同时增加一个新的toMyString()的方法，应该如何做?
  - final方法可以被重载吗? 可以
  - 父类的final方法能不能够被子类重写? 不可以
  - 说说final域重排序规则?
  - 说说final的原理?
  - 使用 final 的限制条件和局限性?
  - 看本文最后的一个思考题

- JUC - 类汇总和学习指南
  - JUC框架包含几个部分?
  - 每个部分有哪些核心的类?
  - 最最核心的类有哪些?

- JUC原子类: CAS, Unsafe和原子类详解

- 线程安全的实现方法有哪些?
- 什么是CAS?
- CAS使用示例，结合AtomicInteger给出示例?
- CAS会有哪些问题?
- 针对这这些问题，Java提供了哪几个解决的?
- AtomicInteger底层实现? CAS+volatile
- 请阐述你对Unsafe类的理解?
- 说说你对Java原子类的理解? 包含13个，4组分类，说说作用和使用场景。
- AtomicStampedReference是什么?
- AtomicStampedReference是怎么解决ABA的? 内部使用Pair来存储元素值及其版本号
- java中还有哪些类可以解决ABA的问题? AtomicMarkableReference

- JUC锁: LockSupport详解
  - 为什么LockSupport也是核心基础类? AQS框架借助于两个类： Unsafe(提供CAS操作)和LockSupport(提供park/unpark操作)
  - 写出分别通过wait/notify和LockSupport的park/unpark实现同步?
  - LockSupport.park()会释放锁资源吗? 那么Condition.await()呢?
  - Thread.sleep()、Object.wait()、Condition.await()、LockSupport.park()的区别? 重点
  - 如果在wait()之前执行了notify()会怎样?
  - 如果在park()之前执行了unpark()会怎样?
- JUC锁: 锁核心类AQS详解
  - 什么是AQS? 为什么它是核心?
  - AQS的核心思想是什么? 它是怎么实现的? 底层数据结构等
  - AQS有哪些核心的方法?
  - AQS定义什么样的资源获取方式? AQS定义了两种资源获取方式：独占(只有一个线程能访问执行，又根据是否按队列的顺序分为公平锁和非公平锁，如ReentrantLock) 和共享(多个线程可同时访问执行，如Semaphore、CountDownLatch、 CyclicBarrier)。ReentrantReadWriteLock可以看成是组合式，允许多个线程同时对某一资源进行读。
  - AQS底层使用了什么样的设计模式? 模板
  - AQS的应用示例?
- JUC锁: ReentrantLock详解
  - 什么是可重入，什么是可重入锁? 它用来解决什么问题?
  - ReentrantLock的核心是AQS，那么它怎么来实现的，继承吗? 说说其类内部结构关系。
  - ReentrantLock是如何实现公平锁的?
  - ReentrantLock是如何实现非公平锁的?
  - ReentrantLock默认实现的是公平还是非公平锁?
  - 使用ReentrantLock实现公平和非公平锁的示例?
  - ReentrantLock和Synchronized的对比?
- JUC锁: ReentrantReadWriteLock详解
  - 为了有了ReentrantLock还需要ReentrantReadWriteLock?
  - ReentrantReadWriteLock底层实现原理?
  - ReentrantReadWriteLock底层读写状态如何设计的? 高16位为读锁，低16位为写锁
  - 读锁和写锁的最大数量是多少?
  - 本地线程计数器ThreadLocalHoldCounter是用来做什么的?
  - 缓存计数器HoldCounter是用来做什么的?
  - 写锁的获取与释放是怎么实现的?
  - 读锁的获取与释放是怎么实现的?
  - RentrantReadWriteLock为什么不支持锁升级?
  - 什么是锁的升降级? RentrantReadWriteLock为什么不支持锁升级?

- JUC集合: ConcurrentHashMap详解
  - 为什么HashTable慢? 它的并发度是什么? 那么ConcurrentHashMap并发度是什么?
  - ConcurrentHashMap在JDK1.7和JDK1.8中实现有什么差别? JDK1.8解决了JDK1.7中什么问题

- ConcurrentHashMap JDK1.7实现的原理是什么? 分段锁机制
- ConcurrentHashMap JDK1.8实现的原理是什么? 数组+链表+红黑树，CAS
- ConcurrentHashMap JDK1.7中Segment数(concurrencyLevel)默认值是多少? 为何一旦初始化就不可再扩容?
- ConcurrentHashMap JDK1.7说说其put的机制?
- ConcurrentHashMap JDK1.7是如何扩容的? rehash(注：segment 数组不能扩容，扩容是 segment 数组某个位置内部的数组 HashEntry<K,V>[] 进行扩容)
- ConcurrentHashMap JDK1.8是如何扩容的? tryPresize
- ConcurrentHashMap JDK1.8链表转红黑树的时机是什么? 临界值为什么是8?
- ConcurrentHashMap JDK1.8是如何进行数据迁移的? transfer

- JUC集合: CopyOnWriteArrayList详解
  - 请先说说非并发集合中Fail-fast机制?
  - 再为什么说ArrayList查询快而增删慢?
  - 对比ArrayList说说CopyOnWriteArrayList的增删改查实现原理? COW基于拷贝
  - 再说下弱一致性的迭代器原理是怎么样的? `COWIterator<E>`
  - CopyOnWriteArrayList为什么并发安全且性能比Vector好?
  - CopyOnWriteArrayList有何缺陷，说说其应用场景?
- JUC集合: ConcurrentLinkedQueue详解
  - 要想用线程安全的队列有哪些选择？Vector，`Collections.synchronizedList( List<T> list)`, ConcurrentLinkedQueue等
  - ConcurrentLinkedQueue实现的数据结构?
  - ConcurrentLinkedQueue底层原理? 全程无锁(CAS)
  - ConcurrentLinkedQueue的核心方法有哪些? offer()，poll()，peek()，isEmpty()等队列常用方法
  - 说说ConcurrentLinkedQueue的HOPS(延迟更新的策略)的设计?
  - ConcurrentLinkedQueue适合什么样的使用场景?
- JUC集合: BlockingQueue详解
  - 什么是BlockingDeque?
  - BlockingQueue大家族有哪些? ArrayBlockingQueue, DelayQueue, LinkedBlockingQueue, SynchronousQueue...
  - BlockingQueue适合用在什么样的场景?
  - BlockingQueue常用的方法?
  - BlockingQueue插入方法有哪些? 这些方法(`add(o)`,`offer(o)`,`put(o)`,`offer(o, timeout, timeunit)`)的区别是什么?
  - BlockingDeque 与BlockingQueue有何关系，请对比下它们的方法?
  - BlockingDeque适合用在什么样的场景?
  - BlockingDeque大家族有哪些?
  - BlockingDeque 与BlockingQueue实现例子?

> **B.4 Java进阶 - Java 并发之J.U.C框架【4/5】：线程池**：再者分析JUC中非常常用的线程池等。

- JUC线程池: FutureTask详解
  - FutureTask用来解决什么问题的? 为什么会出现?
  - FutureTask类结构关系怎么样的?
  - FutureTask的线程安全是由什么保证的?
  - FutureTask结果返回机制?
  - FutureTask内部运行状态的转变?
  - FutureTask通常会怎么用? 举例说明。
- JUC线程池: ThreadPoolExecutor详解
  - 为什么要有线程池?
  - Java是实现和管理线程池有哪些方式? 请简单举例如何使用。
  - 为什么很多公司不允许使用Executors去创建线程池? 那么推荐怎么使用呢?
  - ThreadPoolExecutor有哪些核心的配置参数? 请简要说明
  - ThreadPoolExecutor可以创建哪是哪三种线程池呢?
  - 当队列满了并且worker的数量达到maxSize的时候，会怎么样?
  - 说说ThreadPoolExecutor有哪些RejectedExecutionHandler策略? 默认是什么策略?

    - 简要说下线程池的任务执行机制? execute –> addWorker –>runworker (getTask)
    - 线程池中任务是如何提交的?
    - 线程池中任务是如何关闭的?
    - 在配置线程池的时候需要考虑哪些配置因素?
    - 如何监控线程池的状态?
- JUC线程池: ScheduledThreadPool详解
    - ScheduledThreadPoolExecutor要解决什么样的问题?
    - ScheduledThreadPoolExecutor相比ThreadPoolExecutor有哪些特性?
    - ScheduledThreadPoolExecutor有什么样的数据结构，核心内部类和抽象类?
    - ScheduledThreadPoolExecutor有哪两个关闭策略? 区别是什么?
    - ScheduledThreadPoolExecutor中scheduleAtFixedRate 和 scheduleWithFixedDelay区别是什么?
    - 为什么ThreadPoolExecutor 的调整策略却不适用于 ScheduledThreadPoolExecutor?
    - Executors 提供了几种方法来构造 ScheduledThreadPoolExecutor?
- JUC线程池: Fork/Join框架详解
    - Fork/Join主要用来解决什么样的问题?
    - Fork/Join框架是在哪个JDK版本中引入的?
    - Fork/Join框架主要包含哪三个模块? 模块之间的关系是怎么样的?
    - ForkJoinPool类继承关系?
    - ForkJoinTask抽象类继承关系? 在实际运用中，我们一般都会继承 RecursiveTask 、 RecursiveAction 或 CountedCompleter 来实现我们的业务需求，而不会直接继承 ForkJoinTask 类。
    - 整个Fork/Join 框架的执行流程/运行机制是怎么样的?
    - 具体阐述Fork/Join的分治思想和work-stealing 实现方式?
    - 有哪些JDK源码中使用了Fork/Join思想?
    - 如何使用Executors工具类创建ForkJoinPool?
    - 写一个例子: 用ForkJoin方式实现1+2+3+...+100000?
    - Fork/Join在使用时有哪些注意事项? 结合JDK中的斐波那契数列实例具体说明。
- JUC工具类: CountDownLatch详解
    - 什么是CountDownLatch?
    - CountDownLatch底层实现原理?
    - CountDownLatch一次可以唤醒几个任务? 多个
    - CountDownLatch有哪些主要方法? await(),countDown()
    - CountDownLatch适用于什么场景?
    - 写道题：实现一个容器，提供两个方法，add，size 写两个线程，线程1添加10个元素到容器中，线程2实现监控元素的个数，当个数到5个时，线程2给出提示并结束? 使用CountDownLatch 代替wait notify 好处。
- JUC工具类: CyclicBarrier详解
    - 什么是CyclicBarrier?
    - CyclicBarrier底层实现原理?
    - CountDownLatch和CyclicBarrier对比?
    - CyclicBarrier的核心函数有哪些?
    - CyclicBarrier适用于什么场景?
- JUC工具类: Semaphore详解
    - 什么是Semaphore?
    - Semaphore内部原理?
    - Semaphore常用方法有哪些? 如何实现线程同步和互斥的?
    - Semaphore适合用在什么场景?
    - 单独使用Semaphore是不会使用到AQS的条件队列?
    - Semaphore中申请令牌(acquire)、释放令牌(release)的实现?
    - Semaphore初始化有10个令牌，11个线程同时各调用1次acquire方法，会发生什么?
    - Semaphore初始化有10个令牌，一个线程重复调用11次acquire方法，会发生什么?
    - Semaphore初始化有1个令牌，1个线程调用一次acquire方法，然后调用两次release方法，之后另外一个线程调用acquire(2)方法，此线程能够获取到足够的令牌并继续运行吗?
    - Semaphore初始化有2个令牌，一个线程调用1次release方法，然后一次性获取3个令牌，会获取到吗?

- JUC工具类: Phaser详解
  - Phaser主要用来解决什么问题?
  - Phaser与CyclicBarrier和CountDownLatch的区别是什么?
  - 如果用CountDownLatch来实现Phaser的功能应该怎么实现?
  - Phaser运行机制是什么样的?
  - 给一个Phaser使用的示例?
- JUC工具类: Exchanger详解
  - Exchanger主要解决什么问题?
  - 对比SynchronousQueue，为什么说Exchanger可被视为 SynchronousQueue 的双向形式?
  - Exchanger在不同的JDK版本中实现有什么差别?
  - Exchanger实现机制?
  - Exchanger已经有了slot单节点，为什么会加入arena node数组? 什么时候会用到数组?
  - arena可以确保不同的slot在arena中是不会相冲突的，那么是怎么保证的呢?
  - 什么是伪共享，Exchanger中如何体现的?
  - Exchanger实现举例
- Java 并发 - ThreadLocal详解
  - 什么是ThreadLocal? 用来解决什么问题的?
  - 说说你对ThreadLocal的理解
  - ThreadLocal是如何实现线程隔离的?
  - 为什么ThreadLocal会造成内存泄露? 如何解决
  - 还有哪些使用ThreadLocal的应用场景?

- TODO：Java 并发 - 并发的本质：协作,分工和互斥
- TODO：Java 并发 - 并发的模式梳理