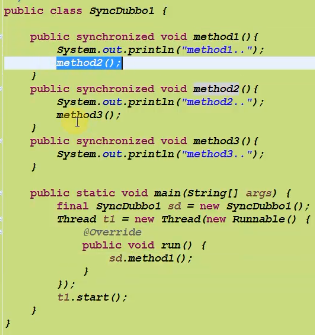
java api中文：<http://tool.oschina.net/uploads/apidocs/jdk-zh/>

1.关键字synchronized拥有锁重入的功能，也就是在使用synchronized时，当一个线程得到了一个对象的锁后，再次请求此对象时是可以再次得到该对象的锁(下例)。出现异常，锁自动释放。



最好不要用字符串去做锁。如果用了，就不要在运行时修改字符串锁。

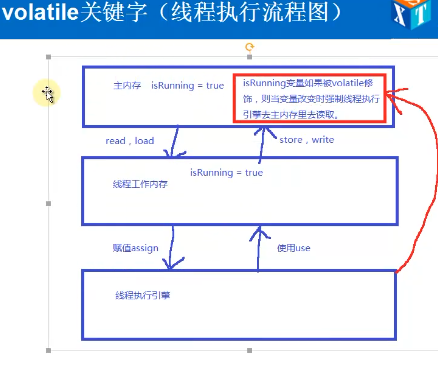
2.volatile关键字的主要作用是使变量在多个线程间可见。

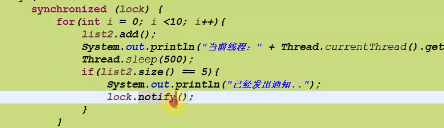


例:AtomicInteger 如果执行多个addAndGet在一个方法内是非原子性的

void xx(){ AtomicInteger.addAndGet(1); AtomicInteger.addAndGet(2);}

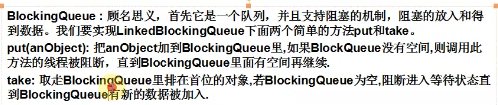
3.线程中有2块空间，假设A,B, A是用来存放主线程中的一些变量，是为了加快效率。B是当前线程的变量存放空间，他会把A空间中主线程的变量copy一份到B.如果A中的变量发生了变化,B是看不见的，只有用关键字修饰变量后B才可见。



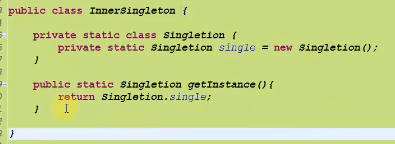


wait()会释放锁。lock.notify()执行后并不会释放锁，它必须要等这段代码块执行完才会释放锁。

工具类: CountDownLatch 用给定的*计数* 初始化 CountDownLatch。由于调用了 [countDown()](http://tool.oschina.net/uploads/apidocs/jdk-zh/java/util/concurrent/CountDownLatch.html#countDown()) 方法，所以在当前计数到达零之前，[await](http://tool.oschina.net/uploads/apidocs/jdk-zh/java/util/concurrent/CountDownLatch.html#await()) 方法会一直受阻塞。之后，会释放所有等待的线程，[await](http://tool.oschina.net/uploads/apidocs/jdk-zh/java/util/concurrent/CountDownLatch.html#await()) 的所有后续调用都将立即返回。这种现象只出现一次——计数无法被重置。如果需要重置计数，请考虑使用 [CyclicBarrier](http://tool.oschina.net/uploads/apidocs/jdk-zh/java/util/concurrent/CyclicBarrier.html)。



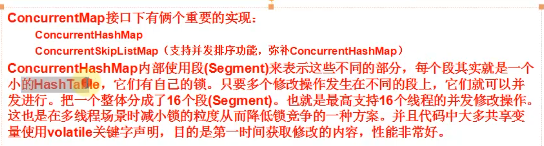
单例模式在多线程中最安全的方式：

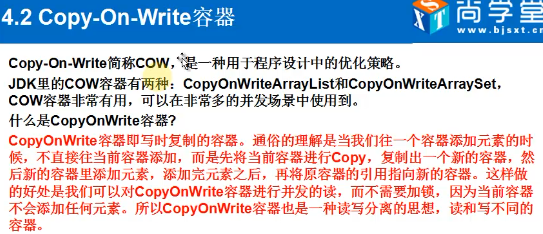


并发容器类支持高并发



ConcurrentHashMap 里有16个segment,每个segment类似一个hashtable,如果线程1访问ConcurrentHashMap 时，它会对第一个segment进行加锁，第二个线程来访问时就不能访问第一个segment,它可以访问第二个segment.增加性能。

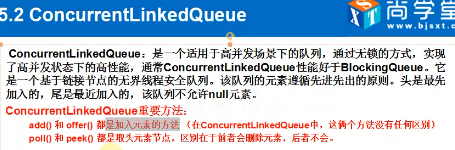


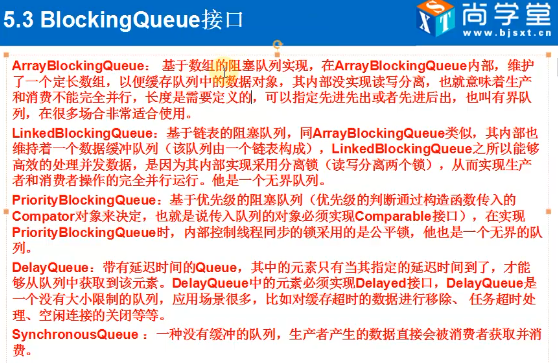


读多写少可以用CopyOnWrite. 高并发读都不用加锁，写场景多的情况下不适用。

并发Queue:

在并发队列上JDK提供了两套实现，一个是以ConcurrentLinkedQueue为代表的高性能队列，一个是以BlcokingQueue接口为代表的阻塞队列，无论哪种都继承自Queue.





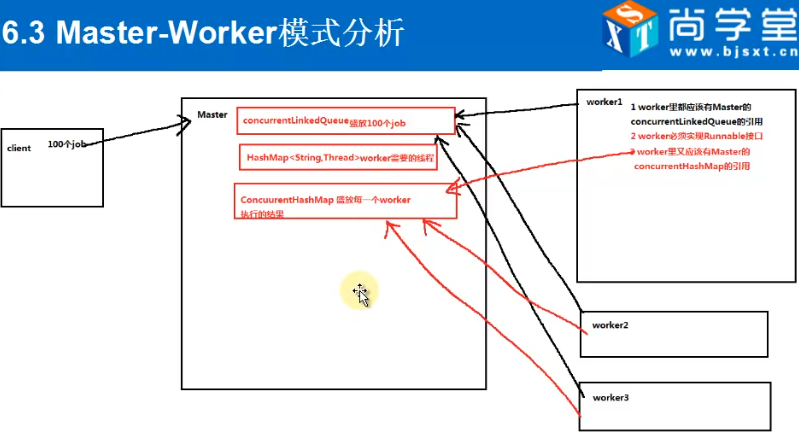
Funter类似于异步调用，它先返回一个空的数据（其实它启动了一个线程T去处理这个任务），用户可以做其它事情，当线程T执行完后就将结果返回。调用顺序如下：

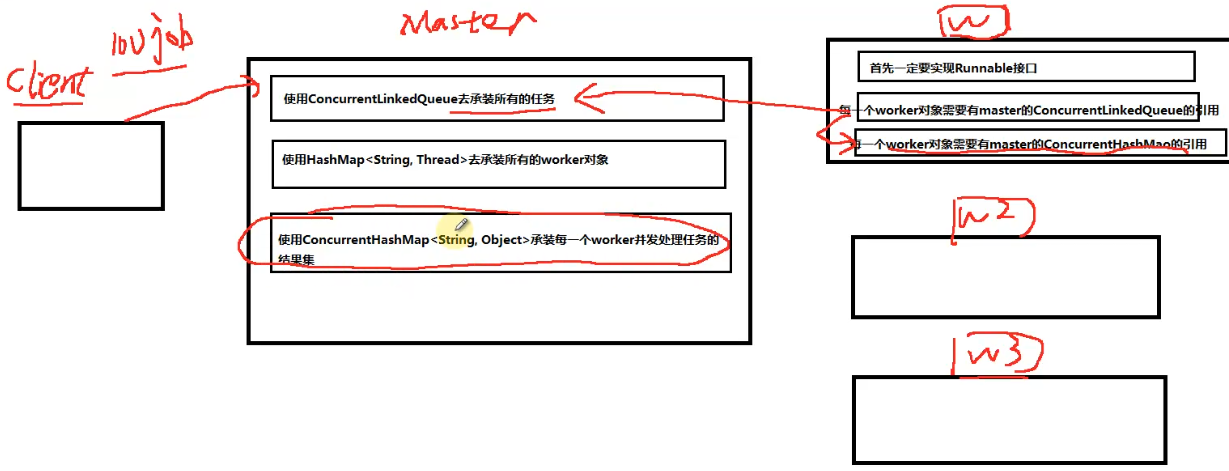
1.返回空数据

2.执行其它事情

3.返回真实数据



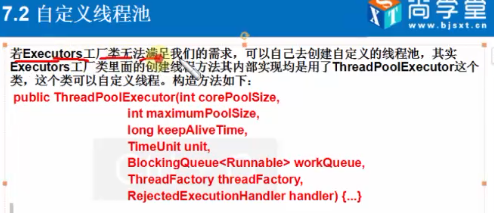


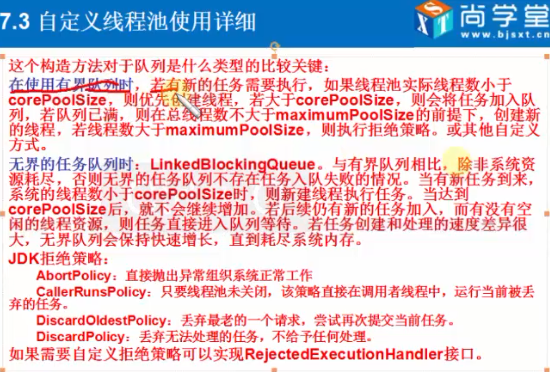


client有一个大的任务，将大的任务分成很多小的任务放在ConcurrentLinkedQueue里，然后创建多个worker去取取这个任务执行多线程处理。将结果存放在ConcurrentHashMap中。每个worker都会有ConcurrentLinkedQueue的引用。Master里的HashMap又保存了worker的引用。Master一启动，worker也会随着启动。







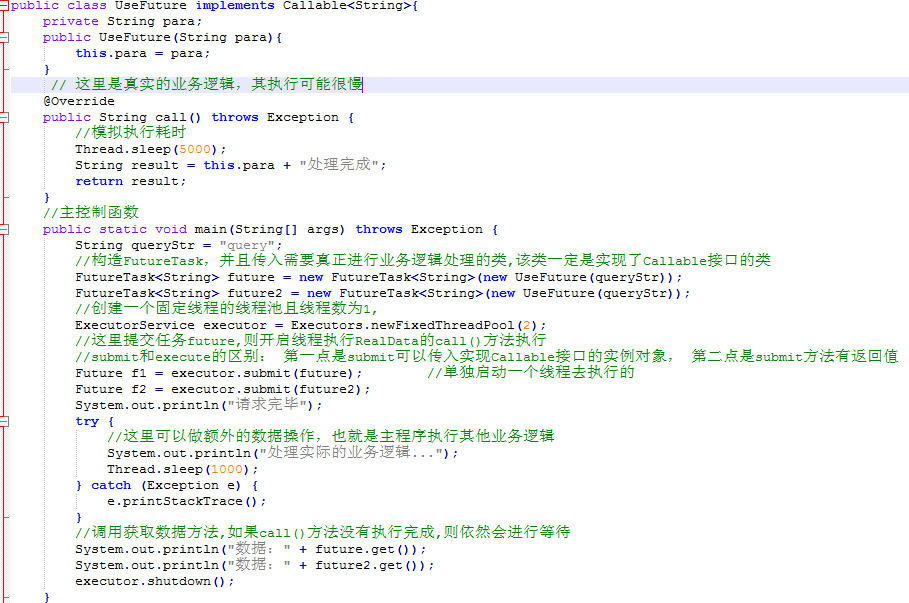


自定义线程时，将 Queue传入进去用来存放任务。

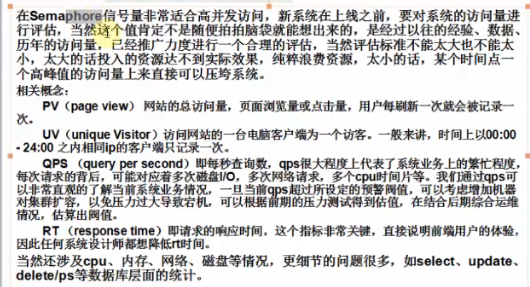


CountDownLatch是争对一个线程，阻塞一个线程。CyclicBarrier是争对多个线程，阻塞多个线程。



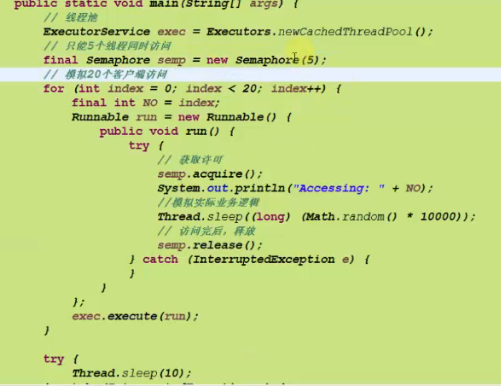


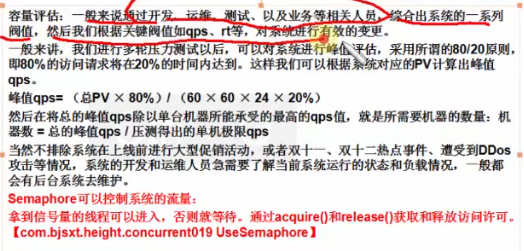
future.get()是会被阻塞的，因为主函数要去取future线程的结果。



如果并发量大，在网络上可能 使用LVS,Nginx进行负载均衡限流，如果还不行，可以套几层LVS或Nginx。在业务上进行限流就要在代码层去限流，在Java里可能使用Semaphore进行处理。

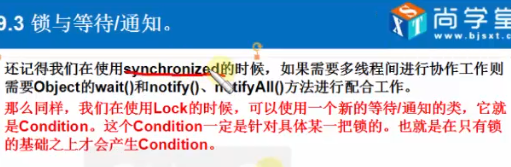
比如 Semaphore seamap = new Semaphore(5);就是我一次只允许5个线程进行处理。



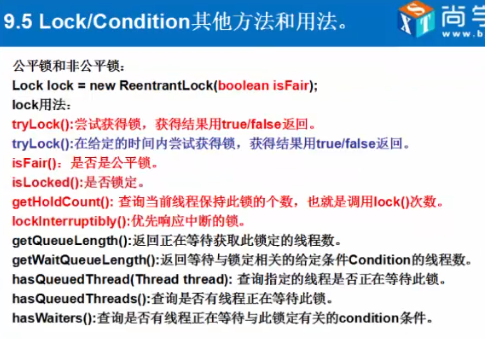


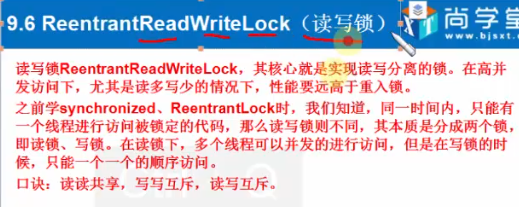
做秒杀的设计思路：独立机器，独立域名。



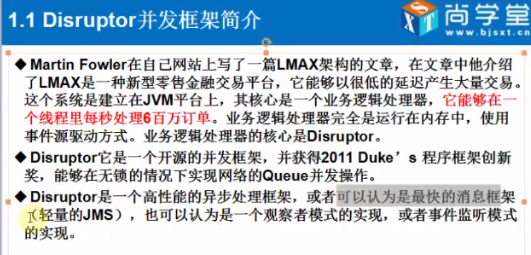








两个读操作是可以并发的







4特性：原子性（Atomicity）、一致性（Consistency）、隔离性（Isolation）、持久性（Durability）.

打个比方，假如我9:00在oracle数据库里执行了一条查询语句查找a的结果，当前a=100，可能要9:10出结果.这时DBA易总在9:05分更新了a并进行了提交(a=200).那么9:10分时我看到的结果应该是什么 呢？其实应该还是100，因为oracle里有一个undo的机制，就是在易总更新提交时，它会把原来的数据100放到undo里，当在9:10分查询到a时，oracle会发现undo发生了变化，所以他不会取现在的值，而是去undo里去取被保存的值100.这就是数据一致性。