并发文章

<https://dayarch.top>

## java队列

Java 并发队列

<https://dayarch.top/p/java-concurrency-queue.html>

Java 阻塞队列--BlockingQueue

<https://www.cnblogs.com/bjxq-cs88/p/9759571.html>

一文秒懂java线程

<https://www.twle.cn/c/yufei/javatm/javatm-basic-executorservice.html>

【并发重要原则】happens-before理解和应用

<https://www.jianshu.com/p/b9186dbebe8e>

## 锁

[不可不说的Java“锁”事 - 美团技术团队](https://tech.meituan.com/2018/11/15/java-lock.html)

偏向锁、轻量锁、重量锁

<https://cloud.tencent.com/developer/article/1036756>

## java并发代码案例

<https://www.zhihu.com/people/wang-du-du-43-1/posts?page=2>

## happens-before是什么？JMM最最核心的概念，看完你就懂了

[happens-before是什么？JMM最最核心的概念，看完你就懂了 - 知乎 (zhihu.com)](https://zhuanlan.zhihu.com/p/126275344)

## java 内存模型

Java 内存模型（必看）

<https://blog.csdn.net/IT__learning/article/details/117457613>

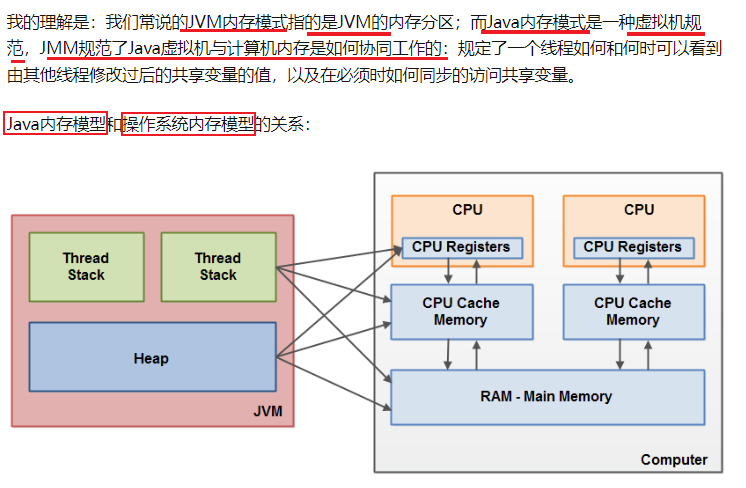
JMM(java内存模型)与 JVM(java虚拟机) 的区别（辅助）

<https://www.zhihu.com/question/45903673/answer/156666045>

java模型介绍（参考）

<https://jenkov.com/tutorials/java-concurrency/java-memory-model.html>

结论：jvm是对java 内存模型的实现





## CAS机制

Java并发系列：CAS操作

<https://blog.csdn.net/m0_46672151/article/details/125274328>

Java的Unsafe类

<https://www.jianshu.com/p/cda24891f9e4>

[Java的CAS乐观锁原理解析 - 公众号-JavaEdge - 博客园 (cnblogs.com)](https://www.cnblogs.com/JavaEdge/p/12159069.html)

<https://www.cnblogs.com/JavaEdge/p/12159069.html>

final的使用

<https://www.cnblogs.com/ywjcqq/p/15850729.html>

field.setAccessible(true)的作用

<https://blog.csdn.net/csdm_admin/article/details/117328519>

Java并发系列：CAS操作

<https://blog.csdn.net/m0_46672151/article/details/125274328>

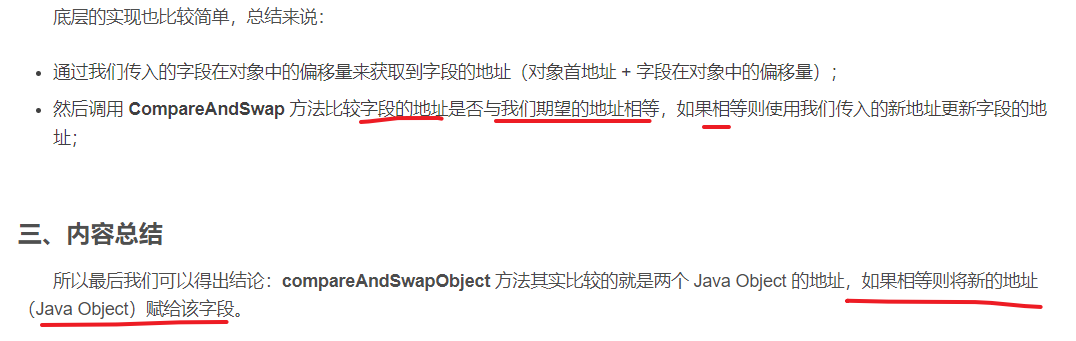
Java的Unsafe类

<https://www.jianshu.com/p/cda24891f9e4>

<https://www.cnblogs.com/JavaEdge/p/12159069.html>

## compareAndSwapObject到底比较什么





文章参考

[源码解析 Java 的 compareAndSwapObject 到底比较的是什么？](https://blog.csdn.net/qq_40697071/article/details/103374783)

## Monitor监视器锁原理

[理解Monitor监视器锁原理](https://blog.csdn.net/qq_43783527/article/details/114669174)

[synchronized底层原理](https://blog.csdn.net/qq_43783527/article/details/114667772)

[Monitor对象是什么？](https://zhuanlan.zhihu.com/p/356010805)

[Monitor机制介绍](https://zhuanlan.zhihu.com/p/88541755)

[java并发系列-monitor机制实现](https://www.cnblogs.com/qingshan-tang/p/12698705.html)

[Java精通并发-自旋对于synchronized关键字的底层意义与价值分析以及互斥锁属性详解与Monitor对象特性解说](https://www.cnblogs.com/webor2006/p/11441679.html)

JVM中的同步是基于进入与退出监视器对象（Monitor，也叫管程对象）来实现的，每个对象实例都会有一个Monitor对象，Monitor对象会和Java对象一同创建并销毁。Monitor对象是由C++来实现的【未来会通过openjdk来分析C++的底层实现的】。

当多个线程同时访问一段同步代码时，这些线程会被放到一个EntrySet集合中，处于阻塞状态的线程都会被放到该列表当中。接下来，当线程获取到对象的Monitor时，Monitor是依赖于底层操作系统的mutex lock来实现互斥的，线程获取mutex成功，则会持有该mutex，这时其它线程就无法再获取到该mutex。

如果线程调用了wait()方法，那么该线程就会释放掉所持有的mutex，并且该线程会进入到WaitSet集合（等待集合）中，等待下一次被其他线程调用notify/notifyAll唤醒。如果当前线程顺利执行完毕方法，那么它也会释放掉所持有的mutex。

总结一下：同步锁在这种实现方式当中，因为Monitor是依赖于底层的操作系统实现，这样就存在用户态和内核态之间的切换，所以会增加性能开销。

通过对象互斥锁的概念来保证共享数据操作的完整性。每个对象都对应于一个可称为“互斥锁”的标记，这个标记用于保证在任何时刻，只能有一个线程访问该对象。

那些处于EntrySet和WaitSet中的线程均处于阻塞状态，阻塞操作是由操作系统来完成的，在linux下是通过pthread\_mutex\_lock函数实现的。线程被阻塞后便会进入到内核调度状态，这会导致系统在用户态和内核态之间切换，严重影响锁的性能。

解决上述问题的办法便是自旋【Spin】。其原理是：当发生对Monitor的争用时，若owner能够在很短的时间内释放掉锁，则那些正在争用的线程就可以稍微等待一下（既所谓的自旋），在Owner线程释放锁之后，争用线程可能会立刻获取到锁，从而避免了系统阻塞。不过，当Owner运行的时间超过了临界值后，争用线程自旋一段时间后依然无法获取到锁，这时争用线程则会停止自旋而进入到阻塞状态。所以总体的思想是：先自旋，不成功再进行阻塞，尽量降低阻塞的可能性，这对那些执行时间很短的代码来说有极大的性能提升。显然，自旋在多处理器（多核心）上才有意义 。

## ThreadLocal

[ThreadLocal，一篇文章就够了 - 知乎 (zhihu.com)](https://zhuanlan.zhihu.com/p/102744180)

[Java中ThreadLocal的实际用途是啥？ - 知乎 (zhihu.com)](https://www.zhihu.com/question/341005993)