


理解锁以及分布式锁

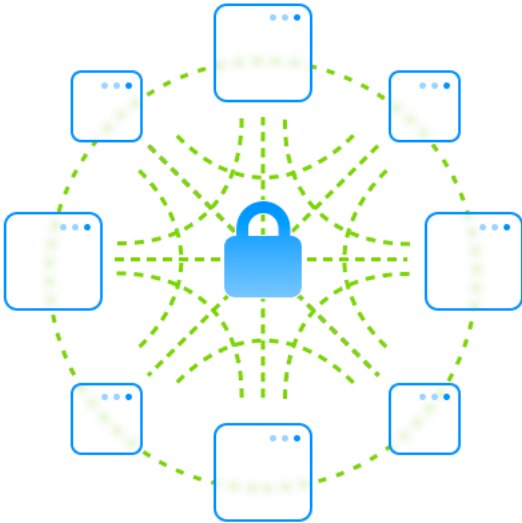


曲高和寡_健 (/u/614a0923878c) [+ 关注](#)

2017.02.02 16:54* 字数 3290 阅读 1522 评论 5 喜欢 25

(/u/614a0923878c)

锁 分布式锁 distributed locks



distributedlocks.png

资源有限，争抢难免，最简单粗暴的办法是谁的拳头大谁就可以抢到最好的资源。那在计算机世界里是如何抢夺资源(cpu, 内存, 网络等)的呢？

锁

在多线程的软件世界里，对共享资源的争抢过程(Data Race)就是**并发**，而对共享资源数据进行访问保护的最直接办法就是引入**锁**！。

无锁编程也是一种办法，但它不在本文的讨论范围，并发多线程转为单线程(Disruptor)，函数式编程，锁粒度控制(ConcurrentHashMap桶)，信号量(Semaphore)等手段都可以实现无锁或锁优化。

技术上来说，锁也可以理解成将大量并发请求**串行化**，但请注意**串行化**不能简单等同为**排队**，因为这里和现实世界没什么不同，排队意味着大家是**公平Fair**的领到资源，先到先得，然而很多情况下为了性能考量多线程之间还是会**不公平Unfair**的去抢。Java中ReentrantLock可重入锁，提供了公平锁和非公平锁两种实现



再注意一点，串行也不是意味着只有一个排队的队伍，每次只能进一个。当然可以好多队伍，每次进入多个。比如餐馆一共10个餐桌，服务员可能一次放行最多10个人进去，有人出来再放行同数量的人进去。Java中Semaphore信号量，相当于同时管理一批锁

锁的类型

1 自旋锁 (Spin Lock)

自旋锁如果已经被别的线程获取，调用者就一直循环在那里看是否该自旋锁的保持者已经释放了锁，“自旋”一词就是因此而得名。

自旋锁是一种**非阻塞**锁，也就是说，如果某线程需要获取自旋锁，但该锁已经被其他线程占用时，该线程不会被挂起，而是在不断的消耗CPU的时间，不停的试图获取自旋锁。

Java没有默认的自旋锁实现，示例代码如下：

```
public class SpinLock {  
    private AtomicReference<Thread> sign =new AtomicReference<>();  
    public void lock(){  
        Thread current = Thread.currentThread();  
        while(!sign .compareAndSet(null, current)){  
        }  
    }  
    public void unlock (){  
        Thread current = Thread.currentThread();  
        sign .compareAndSet(current, null);  
    }  
}
```

通过示例，可以看到CAS原子操作将sign从期望的null设置为当前线程，线程A第一次调用lock()可以获取锁，第二次调用将进入循环等待，因为sign已经被设置为了current。简单加个当前锁的owner比对判断和锁计数器，即可实现**重入**。

2 互斥锁 (Mutex Lock)

互斥锁是阻塞锁，当某线程无法获取互斥锁时，该线程会被直接挂起，不再消耗CPU时间，当其他线程释放互斥锁后，操作系统会唤醒那个被挂起的线程。

阻塞锁可以说是让线程进入阻塞状态进行等待，当获得相应的信号（唤醒，时间）时，才可以进入线程的准备就绪状态，准备就绪状态的所有线程，通过竞争进入运行状态。它的优势在于，阻塞的线程不会占用 CPU 时间，不会导致 CPU 占用率过高，但进入时间以及恢复时间都要比自旋锁略慢。在竞争激烈的情况下阻塞锁的性能要明显高于自旋锁。

JAVA中，能够进入/退出、阻塞状态或包含阻塞锁的方法有：
synchronized
ReentrantLock
Object.wait()/notify()
LockSupport.park()/unpark()(j.u.c经常使用)



自旋锁 VS 互斥锁**两种锁适用于不同场景：**

如果是多核处理器，预计线程等待锁的时间很短，短到比线程两次上下文切换时间要少的情况下，使用**自旋锁**是划算的。

如果是多核处理器，如果预计线程等待锁的时间较长，至少比两次线程上下文切换的时间要长，建议使用**互斥锁**。

如果是单核处理器，一般建议**不要使用自旋锁**。因为，在同一时间只有一个线程是处在运行状态，那如果运行线程发现无法获取锁，只能等待解锁，但因为自身不挂起，所以那个获取到锁的线程没有办法进入运行状态，只能等到运行线程把操作系统分给它的时间片用完，才有机会被调度。这种情况下使用自旋锁的代价很高。

如果加锁的代码经常被调用，但竞争情况很少发生时，应该优先考虑使用**自旋锁**，自旋锁的开销比较小，互斥量的开销较大。

3 可重入锁 (Reentrant Lock)

可重入锁是一种特殊的互斥锁，它可以被同一个线程多次获取，而不会产生死锁。

1. 首先它是互斥锁：任意时刻，只有一个线程锁。即假设A线程已经获取了锁，在A线程释放这个锁之前，B线程是无法获取到这个锁的，B要获取这个锁就会进入阻塞状态。
2. 其次，它可以被同一个线程多次持有。即，假设A线程已经获取了这个锁，如果A线程在释放锁之前又一次请求获取这个锁，那么是能够获取成功的。

Java中的synchronized，ReentrantLock都是可重入锁。

4 轻量级锁(Lightweight Lock) & 偏向锁(Biased Lock)

首先互斥是一种会导致线程挂起，并在较短时间内又需要重新调度回原线程的，较为消耗资源的操作。

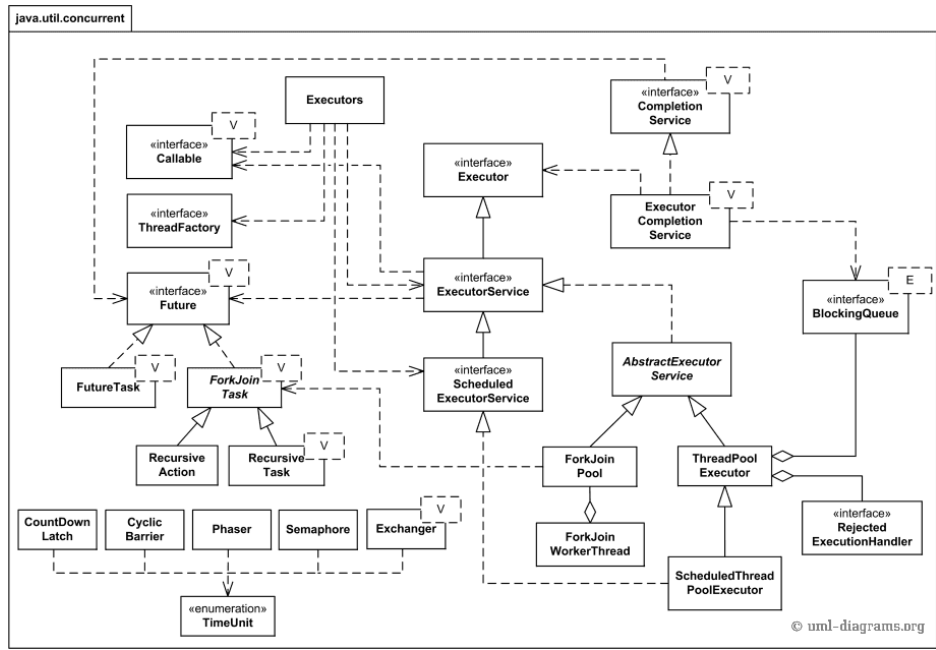
Java6为了减少获得锁和释放锁所带来的性能消耗，引入了“偏向锁”和“轻量级锁”，所以在Java6里锁一共有四种状态，无锁状态，偏向锁状态，轻量级锁状态和重量级锁状态，它会随着竞争情况逐渐升级。锁可以升级但不能降级，意味着偏向锁升级成轻量级锁后不能降级成偏向锁。这种锁升级却不能降级的策略，目的是为了提高获得锁和释放锁的效率。

数据库中针对不同的锁层级(Lock Hierarchy, 表/页/行等)，也有类似锁升级(Lock Escalations)的理念。

5 JUC

并发大师Doug Lea在JUC包中实现了大量的并发工具类，并发思想在源码中得到了很好的体现。比如Semaphore, CountdownLatch, CyclicBarrier都是特定场景下的经典实现，大家有兴趣可以自行研究，最终一叹：锁原来可以玩出这么多花样来。





java-7-concurrent-executors-uml-class-diagram-example.png

锁的后遗症

在并发世界里，锁扮演了一个个亦正亦邪的角色，甚至很多时候是个大反派。锁的后遗症包括：死锁，饥饿，活锁，Lock Convoying(多个同优先级的线程重复竞争同一把锁，此时大量虽然被唤醒而得不到锁的线程被迫进行调度切换，这种频繁的调度切换相当影响系统性能)，优先级反转，不公平和低效率等。而这些问题都是在实现锁的过程中普遍存在而又不得不面对的。

这里只抛出问题让读者了解，具体解决方案不在本文范畴。

活锁和死锁的区别在于，处于活锁的实体是在不断的改变状态，所谓之“活”，而处于死锁的实体表现为等待；活锁有可能自行解开，死锁则不能。

分布式锁

相对于单机应用设定的单机锁，为分布式应用各节点对共享资源的排他式访问而设定的锁就是分布式锁。在分布式场景下，有很多种情况都需要实现多节点的最终一致性。比如全局发号器，分布式事务等等。

传统实现分布式锁的方案一般是利用持久化数据库（如利用InnoDB行锁，或事务，或version乐观锁），当然大部分时候可以满足大部分人的需求。而如今互联网应用的量级已经几何级别的爆发，利用诸如zookeeper，redis等更高效的分布式组件来实现分布式锁，可以提供高可用的更强壮的锁特性，并且支持丰富化的使用场景。

开源实现已有不少比如Redis作者基于Redis设计的Redlock，Redisson等。

小插曲：
锁存在的地方就有争议，Redlock也不例外。有一位分布式专家曾经发表过一篇文章<How to do distributed locking> (<http://martin.kleppmann.com/2016/02/08/how-to-do-distributed-locking.html>)，质疑Redlock的正确性，Redis作者则在<Is Redlock safe?> (<http://antirez.com/news/101>)中给予了回应，交锋相对精彩无比，有兴趣的读者可以自行前往。



前人栽树后人乘凉，当下各种的锁实现已经给我们提供了很多优雅的设计范本，我们具体来分析下分布式锁到底应该怎么设计呢？

分布式锁的设计要点

我们以Redis为例，简单思考下这个锁的实现。

似乎加锁的时候只要一个 **SETNX** 命令就搞定了，返回1代表加锁成功，返回0 表示锁被占用着。然后再用 **DEL** 命令解锁，返回1表示解锁成功，0表示已经被解锁过。

接着问题就来了：

SETNX会存在锁竞争，如果在执行过程中客户端宕机，会引起死锁问题，也就是锁资源无法释放。解决死锁的问题其实可以向Mysql的死锁检测学习，设置一个失效时间，通过key的时间戳来判断是否需要强制解锁。

但是强制解锁也存在问题，一个就是时间差问题，不同的机器的本地时间可能也存在时间差，在很小事务粒度的高并发场景下还是会存在问题，比如删除锁的时候，会判断时间戳已经超过时效，有可能删除其他已经获取锁的客户端的锁。

另外，如果设置了一个超时时间，若程序执行时间超过了超时时间，那么还没执行完锁会被自动释放，原来持锁的客户端再次解锁的时候会出现问题，而且最为严重的还是一致性没有得到保障。如何合理的设置这个超时时间可能是一个观测并不断调整的过程。

那么，总结下设计的几个要点：

- 锁的时效。避免单点故障造成死锁，影响其他客户端获取锁。但是也要保证一旦一个客户端持锁，在客户端可用时不会被其他客户端解锁。
- 持锁期间的check。尽量在关键节点检查锁的状态，所以要设计成可重入锁。
- 减少获取锁的操作，尽量减少redis压力。所以需要让客户端的申请锁有一个等待时间，而不是所有申请锁的请求线程不断的循环申请锁。
- 加锁的事务或者操作尽量粒度小，减少其他客户端申请锁的等待时间，提高处理效率和并发性。
- 持锁的客户端解锁后，要能通知到其他等待锁的节点，否则其他节点只能一直等待一个预计的时间再触发申请锁。类似线程的notifyAll,要能同步锁状态给其他客户端，并且是分布式消息。
- 考虑任何执行句柄中可能出现的异常，状态的正确流转和处理。比如，不能因为一个节点解锁失败，或者锁查询失败（redis 超时或者其他运行时异常），影响整个等待的任务队列，或者任务池。
- 若Redis服务器宕机或者网络异常，要有其他备份方案，比如单机锁限流+最终数据库的持久化锁来做好最终一致性控制。

如果大家想自己实现分布式锁的话，建议先把开源的一些实现先读一读，拓展下思路。还是那句话，如非必要，勿增实体！

参考文章：

基于Redis实现分布式锁，Redisson使用及源码分析

(<http://blog.jobbole.com/99751/>)

聊一聊分布式锁的设计

(<http://www.weizijun.cn/2016/03/17/%E8%81%8A%E4%B8%80%E8%81%8A%E5%88%86%E5%B8%83%E5%BC%8F%E9%94%81%E7%9A%84%E8%AE%BE%E8%AE%A1/>)



曲高和寡_健 (/u/614a0923878c) ♂


写了 48901 字，被 68 人关注，获得了 152 个喜欢 (/u/614a0923878c)

+ 关注

欢迎关注我的订阅号 [geniusiandev](#) 曲水流觞TechRill，互相交流

如果觉得我的文章对您有用，请随意赞赏。您的支持将鼓励我继续创作！

赞赏支持

 喜欢 (/sign_in?utm_source=desktop&utm_medium=not-signed-in-like-button) | 25



更多分享

(<http://cwb.assets.jianshu.io/notes/images/7751970/w>




登录 (/sign_in?utm_source=desktop&utm_medium=not-signed-in-comment-form)



5条评论


只看作者

按喜欢排序 按时间正序 按时间倒序




羿良帝 (/u/1d70aa5e915b)
2楼 · 2017.02.02 01:02
(/u/1d70aa5e915b)
菜鸟表示看不懂！



 赞  回复


曲高和寡_健 (/u/614a0923878c): anyway, 多谢支持
2017.02.03 00:05  回复


添加新评论



景b (/u/721cce1d494b)
3楼 · 2017.06.09 21:55
(/u/721cce1d494b)
楼主你好，提个不成熟的小建议，希望可以把你写这篇文章引用的资料和链接加在最后面。
另外问个问题，像小插曲redis作者同分布式专家的辩论那段，这样的资料您是什么渠道获取的。是简单的变换各种Google关键字浏览资料，还是有什么其他的正确打开方式？

 赞  回复




曲高和寡_健 (/u/614a0923878c): 谢谢建议，以后会考虑加上。辩论的那个真的是在某个角落不小心发现然后记录下来，忘记出处了。
2017.06.13 04:21  回复

曲高和寡_健 (/u/614a0923878c): 已更新引用。
2017.06.14 01:04  回复

添加新评论



被以下专题收入，发现更多相似内容

-  @IT-互联网 (/c/V2CqjW?utm_source=desktop&utm_medium=notes-included-collection)
-  架构算法设计模... (/c/c568ddab391a?utm_source=desktop&utm_medium=notes-included-collection)
-  技术干货 (/c/38d96caffb2f?utm_source=desktop&utm_medium=notes-included-collection)
-  程序员 (/c/NEt52a?utm_source=desktop&utm_medium=notes-included-collection)
-  Java学习笔记 (/c/04cb7410c597?utm_source=desktop&utm_medium=notes-included-collection)

推荐阅读

更多精彩内容 > (/)

短网址服务系统如何设计 (/p/d1cb7a51e7e5?utm_campaign=maleskine&utm_content=note&utm_medium=pc_all_hots&utm_source=recommendation) (/p/d1cb7a51e7e5?utm_campaign=maleskine&utm_content=note&utm_medium=pc_all_hots&utm_source=recommendation)

短网址 顾名思义，就是将长网址缩短到一个很短的网址，用户访问这个短网址可以重定向到原本的长网址（还原）。这样可以达到易于记忆、转换的目的， ...

曲高和寡_健 (/u/614a0923878c?utm_campaign=maleskine&utm_content=user&utm_medium=pc_all_hots&utm_source=recommendation)

2015.10.1起空间搬到这里 (/p/9a25ee5cdb80?utm_campaign=maleskine&utm_content=note&utm_medium=pc_all_hots&utm_source=recommendation) (/p/9a25ee5cdb80?utm_campaign=maleskine&utm_content=note&utm_medium=pc_all_hots&utm_source=recommendation)

之前过于迷信Baidu空间的稳定性，现在它关闭了，会陆续将一些文章搬到这里来。

曲高和寡_健 (/u/614a0923878c?utm_campaign=maleskine&utm_content=user&utm_medium=pc_all_hots&utm_source=recommendation)

给未来女婿的一封信 (/p/8e0d3e43922f?utm_campaign=maleskine&utm_content=note&utm_medium=pc_all_hots&utm_source=recommendation) (/p/8e0d3e43922f?utm_campaign=maleskine&utm_content=note&utm_medium=pc_all_hots&utm_source=recommendation)

亲爱的某先生：因为未知贵姓，所以暂且这么称呼你，请原谅！你的姓应该就是百家姓里的一种，当然，也不排除可能是斯密斯、布兰特或者道格拉斯等等...

断鹞 (/u/65096740cbfc?utm_campaign=maleskine&utm_content=user&utm_medium=pc_all_hots&utm_source=recommendation)

谢谢你，给我18厘米的爱情 (/p/bac3b1020838?utm_campaign=maleskine&utm_content=note&utm_medium=pc_all_hots&utm_source=recommendation) (/p/bac3b1020838?utm_campaign=maleskine&utm_content=note&utm_medium=pc_all_hots&utm_source=recommendation)

-1- 身边曾经有朋友拍着胸膛预言，高璐可能会做一辈子的老姑娘。结果气得高璐当场哭着跑出去。可没过几天，她拉着新男友出现在我们面前时，所有人...

共央君 (/u/8c84a932666e?utm_campaign=maleskine&utm_content=user&utm_medium=pc_all_hots&utm_source=recommendation)

对啊，我就是一杯12块钱的奶茶也买不起 (/p/fcee5596e865?utm_campaign=maleskine&utm_content=note&utm_medium=pc_all_hots&utm_source=recommendation) (/p/fcee5596e865?utm_campaign=maleskine&utm_content=note&utm_medium=pc_all_hots&utm_source=recommendation)

1 入职时，听到要去上海培训，我的内心是拒绝的。试用期工资低，上海消费高，自给自足都成问题。好在公司很人性化，给我们提供了住宿和食堂就餐。...

学飞的猪女侠 (/u/df1d8b26f22f?utm_campaign=maleskine&utm_content=user&utm_medium=pc_all_hots&utm_source=recommendation)

