Atitit 并发锁机制 艾提拉总结

目录

[1. 2. 用的比较频繁锁 2](#_Toc30342)

[1.1. 语法锁sync api锁 2](#_Toc14264)

[1.2. 数据库标志位锁 2](#_Toc10526)

[1.3. 文件锁 2](#_Toc5033)

[2. 锁得类型 3 2](#_Toc16067)

[2.1. 轻量级锁 vs 重量级锁 2](#_Toc18554)

[2.2. App锁 vm锁 os锁 cpu锁 3](#_Toc26806)

[2.3. 2.1. 排它锁 “互斥锁 共享锁 3](#_Toc29838)

[2.4. 乐观锁与悲观锁 乐观锁: 3 3](#_Toc23890)

[2.5. 2.2. 自旋锁还是信号量 3 3](#_Toc18511)

[2.6. 个监视器锁（monitor） 3](#_Toc30955)

[2.7. 互斥锁 3](#_Toc11790)

[2.8. 2.3. -自释放 vs 手动释放 3 3](#_Toc9906)

[2.9. 2.4. 按照使用者 数据库锁 操作系统锁 app锁 3](#_Toc28111)

[2.10. 2.5. 按照外观 u型锁 一字锁 月牙锁 3](#_Toc22986)

[2.11. 语法锁 api锁 3 3](#_Toc5269)

[2.12. 偏向锁 4](#_Toc24559)

[2.13. 大锁 小锁 细粒度锁 大力度所 4](#_Toc25320)

[2.14. 2.6. 按照工作原理 弹簧锁 弹子锁 3 5](#_Toc25175)

[2.15. 2.8. 行级锁 表级别锁 根据锁的粒度来区分 3 5](#_Toc25769)

[2.16. 2.9.  表级锁:5种 共享锁(SHARE) 排他(EXCLUSIVE) 5](#_Toc20163)

[2.17. 2.11. 按照数据库维度分类 共享锁与排它锁 更新锁  7 5](#_Toc9333)

[2.18. 内存锁 文件锁 数据库锁 标志位锁 5](#_Toc29135)

[2.19. 比特锁 字节锁 5](#_Toc16160)

[2.20. 按照语言分 php锁 java锁 net锁 5](#_Toc5377)

[2.21. 跨语言锁 5](#_Toc18)

[2.22. 分布式锁 非分布式 跨进程锁 5](#_Toc17968)

[2.23. 共享内存空间计数锁 剪贴板锁 6](#_Toc13712)

[2.24. 队列所 6](#_Toc23347)

[2.25. 同步锁 6](#_Toc9263)

[2.26.  读写锁是一种特殊的自旋锁, 6](#_Toc7163)

[3. 4. 锁的实现机制 大多是基于数据版本versin记录机制 9 7](#_Toc31496)

[3.1. 4.1.1. 乐观锁另一种实现方式CAS 10 7](#_Toc3714)

[3.2. 4.2. test and set”指令 10 7](#_Toc30308)

[3.3. 4.3. 锁内存总线 10 7](#_Toc31545)

[3.4. 4.4. 硬件层面，CPU提供了原子操作、关中断、锁内存总线的机制 10 7](#_Toc3325)

[3.5. 内存屏障、 7](#_Toc13783)

[3.6. CAS操作 7](#_Toc9220)

[3.7. 、LOCK指令 7](#_Toc21948)

[3.8. 操作系统提供的互斥锁 互斥锁（Mutex Lock） 7](#_Toc26759)

[4. 常见vm锁机制 7](#_Toc8768)

[4.1. Monitor监视器锁 java c#net的vm锁实现机制 7](#_Toc8437)

[5. reef 8](#_Toc23207)

# 2. 用的比较频繁锁

## 语法锁sync api锁

## 数据库标志位锁

## 文件锁

# 锁得类型 3

## 轻量级锁 vs 重量级锁

Sync是重量级所，Synchronized是通过对象内部的一个叫做监视器锁（monitor）来实现的。

但是监视器锁本质又是依赖于底层的操作系统的互斥锁（Mutex Lock）来实现的

这种依赖于操作系统互斥锁（Mutex Lock）所实现的锁我们称之为“重量级锁”。

轻量级锁依赖于cas机制

## App锁 vm锁 os锁 cpu锁

## 2.1. 排它锁 “互斥锁 共享锁

## 乐观锁与悲观锁 乐观锁: 3

## 2.2. 自旋锁还是信号量 3

## ****个监视器锁（monitor）****

我们直接参考JVM规范中描述：**每个对象有一个监视器锁（monitor）**。

当monitor被占用时就会处于锁定状态，线程执行monitorenter指令时尝试获取monitor的所有权，过程如下：

## 互斥锁

在[编程](https://baike.baidu.com/item/%E7%BC%96%E7%A8%8B/139828" \t "https://baike.baidu.com/item/_blank)中，引入了对象互斥锁的概念，来保证共享数据操作的完整性。每个[对象](https://baike.baidu.com/item/%E5%AF%B9%E8%B1%A1/2331271" \t "https://baike.baidu.com/item/_blank)都对应于一个可称为" 互斥锁" 的标记，这个标记用来保证在任一时刻，只能有一个[线程](https://baike.baidu.com/item/%E7%BA%BF%E7%A8%8B/103101" \t "https://baike.baidu.com/item/_blank)访问该[对象](https://baike.baidu.com/item/%E5%AF%B9%E8%B1%A1/2331271" \t "https://baike.baidu.com/item/_blank)

## 2.3. -自释放 vs 手动释放 3

## 2.4. 按照使用者 数据库锁 操作系统锁 app锁

## 2.5. 按照外观 u型锁 一字锁 月牙锁

## 语法锁 api锁 3

Synchronized Monitor类

C#的lock关键字

2.7. 实现方式，语言方式与库方式 3

Synchronized是通过对象内部的一个叫做监视器锁（monitor）来实现的。

但是监视器锁本质又是依赖于底层的操作系统的互斥锁（Mutex Lock）来实现的。而操作系统实现线程之间的切换这就需要从用户态转换到核心态，这个成本非常高，状态之间的转换需要相对比较长的时间，这就是为什么Synchronized效率低的原因。

因此，这种依赖于操作系统互斥锁（Mutex Lock）所实现的锁我们称之为“重量级锁”。

Synchronized是通过对象内部的一个叫做监视器锁（monitor）来实现的。

但是监视器锁本质又是依赖于底层的操作系统的互斥锁（Mutex Lock）来实现的

这种依赖于操作系统互斥锁（Mutex Lock）所实现的锁我们称之为“重量级锁”。

## ****偏向锁****

根据轻量级锁的实现，我们知道虽然**轻量级锁不支持“并发”**，遇到“并发”就要膨胀为重量级锁，但是轻量级锁可以支持多个线程以串行的方式访问同一个加锁对象。

比如A线程可以先获取对象o的轻量锁，然后A释放了轻量锁，这个时候B线程来获取o的轻量锁，是可以成功获取得，以这种方式可以一直串行下去。

之所以能实现这种串行，是因为有一个释放锁的动作。那么假设有一个加锁的java方法，这个方法在运行的时候其实从始至终只有一个线程在调用，但是每次调用完却也要释放锁，下次调用还要重新获得锁。

那么我们能不能做一个假设：“假设加锁的代码**从始至终就只有一个线程**在调用，如果发现有多于一个线程调用，再膨胀成轻量级锁也不迟”。这个假设，就是偏向锁的核心思想。

   偏向锁依赖了一种叫做CAS(compare and swap)的操作。

## 大锁 小锁 细粒度锁 大力度所

## 2.6. 按照工作原理 弹簧锁 弹子锁 3

## 2.8. 行级锁 表级别锁 根据锁的粒度来区分 3

## 2.9.  表级锁:5种 共享锁(SHARE) 排他(EXCLUSIVE)

悲观锁: 乐观锁: 4

2.10. PS：只有共享锁与共享锁相互兼容，共享锁与排它锁、排它锁之间都互不兼容 7

## 2.11. 按照数据库维度分类 共享锁与排它锁 更新锁  7

2.12. 按照思想维度分类 悲观锁 乐观锁 7

## 内存锁 文件锁 数据库锁 标志位锁

## 比特锁 字节锁

## 按照语言分 php锁 java锁 net锁

## 跨语言锁

## 分布式锁 非分布式 跨进程锁

分布式锁可以使用redis zookeep来实现

## 共享内存空间计数锁 剪贴板锁

## 队列所

## 同步锁

## 读写锁是一种特殊的自旋锁,

它把对共享资源对访问者划分成了读者和写者,读者只对共享资源进行访问,写者则是对共享资源进行写操作

## 全局锁 局部锁

## 显示锁 隐式锁

# 4. 锁的实现机制 大多是基于数据版本versin记录机制 9

## 4.1.1. 乐观锁另一种实现方式CAS 10

## 4.2. test and set”指令 10

## 4.3. 锁内存总线 10

## 4.4. 硬件层面，CPU提供了原子操作、关中断、锁内存总线的机制 10

## 内存屏障、

## CAS操作

## 、LOCK指令

## 操作系统提供的互斥锁 互斥锁（Mutex Lock）

# 常见vm锁机制

## Java中存在两种锁机制：synchronized和Lock

synrhronized关键字简洁、清晰、语义明确，因此即使有了Lock接口，使用的还是非常广泛。其应用层的语义是可以把任何一个非null对象 作为"锁"，当synchronized作用在方法上时，锁住的便是对象实例（this）；当作用在静态方法时锁住的便是对象对应的Class实例，因为 Class数据存在于永久带，因此静态方法锁相当于该类的一个全局锁；当synchronized作用于某一个对象实例时，锁住的便是对应的代码块。在 HotSpot JVM实现中，锁有个专门的名字：对象监视器。

Free的队列AQS，基本思路是自旋后阻塞，竞争切换后继续竞争锁，稍微牺牲了公平性，但获得了高吞吐量。AQS

AQS是AbustactQueuedSynchronizer的简称，它是一个Java提供的底层同步工具类，用一个int类型的变量表示同步状态，并提供了一系列的CAS操作来管理这个同步状态。AQS的主要作用是为Java中的并发同步组件提供统一的底层支持，例如ReentrantLock，CountdowLatch就是基于AQS实现的，用法是通过继承AQS实现其模版方法，然后将子类作为同步组件的内部类。

————————————————

## Monitor监视器锁 java c#net的vm锁实现机制

**每个对象有一个监视器锁（monitor）**。

Monitor.Enter与Monitor.Exit

　　Monitor.Enter和Monitor.Exit这个东西跟lock的作用一样。事实上。lock就是Monitor.Enter和Monitor.Exit的包装。

　　下面用Monitor.Enter与Monitor.Exit来实现相同的代码：

# reef

Atitit 粘贴路径参数法 跨进程通讯法

Atitit.软件与编程语言中的锁机制原理attilax总结

Java并发机制及锁的实现原理 - yuluoxingkong - 博客园

锁机制与原子操作 \_第四篇\_ - 逆心 - 博客园

(9+条消息)JVM底层如何实现synchronized - kobejayandy的专栏 - CSDN博客