# 分布式锁

## 方案

针对分布式锁的实现，目前比较常用的方案：

1. 基于数据库实现分布式锁
2. 基于缓存（redis、memcache、tair）实现分布式锁
3. 基于Zookeeper实现分布式锁

### DB锁

### Redis锁

### Zookeeper分布式锁

## 特点

分布式锁的实现由多种方式，但是不管怎样，分布式锁一般要有以下特点：

排他性：任意时刻，只能有一个client能获取到锁

容错性：分布式锁服务一般要满足AP，也就是说，只要分布式锁服务集群节点大部分存活，client就可以进行加锁解锁操作

避免死锁：分布式锁一定能得到释放，即使client在释放之前崩溃或者网络不可达

除了以上特点之外，分布式锁最好也能满足可重入、高性能、阻塞锁特性（AQS这种，能够及时从阻塞状态唤醒）等。

# MySQL数据库锁

锁(INNODB引擎)、表级锁(MYISAM引擎)和页级锁(BDB引擎 )。

InnoDB存储引擎既支持行级锁（row-level locking），也支持表级锁，但默认情况下采用行级锁。

## 行锁

### 概述

行级锁是Mysql中锁定粒度最细的一种锁，表示只针对当前操作的行进行加锁。行级锁能大大减少数据库操作的冲突。其加锁粒度最小，但加锁的开销也最大。

行级锁分为共享锁和排他锁。

### 特点

开销大，加锁慢；会出现死锁；锁定粒度最小，发生锁冲突的概率最低，并发度也最高。

### 分类

InnoDB实现了两种类型的行锁：

共享锁（S）：又称读锁。允许一个事务去读一行，阻止其他事务获得相同数据集的排他锁。若事务T对数据对象A加上S锁，则事务T可以读A但不能修改A，其他事务只能再对A加S锁，而不能加X锁，直到T释放A上的S锁。这保证了其他事务可以读A，但在T释放A上的S锁之前不能对A做任何修改。

所谓的共享锁，就是多个事务只能读数据不能修改数据。加共享锁可以使用select…lock in share mode语句。

排他锁（X）：又称写锁。允许获取排他锁的事务更新数据，阻止其他事务获取相同的数据集共享读锁和排他写锁。若事务T对数据对象A加上X锁，事务T可以读A也可以修改A，其他事务不能再对A加任何锁，知道T释放A上的锁。

排他锁指的是一个事务在一行数据加上排他锁后，其他事务不能再在其上加上其他的锁。InnoDB引擎默认的修改数据语句：update、delete、insert都会自动给涉及到的数据加上排他锁。所以加过排他锁的数据行在其他事务中是不能修改数据的，也不能通过for update和lock in share mode锁的方式查询数据，但是可以直接通过select…from…查询数据，因为普通查询没有任何锁限制。

另外，为了允许行锁和表锁共存，实现多粒度的锁机制，InnoDB还有两种内部使用的意向锁（Intention Locks），这两种意向锁都是表锁。

意向共享锁（IS）：事务打算给数据行共享锁，事务在给一个数据行加共享锁前必须先取得该表的IS锁。

意向排他锁（IX）：事务打算给数据行加排他锁，事务在给一个数据行加排他锁前必须先取得该表的IX锁。

### 死锁

MyISAM中是不会产生死锁的，因为MyISAM总是一次性获得所需的全部锁，要么全部满足，要么全部等待。而在InnoDB中，锁是逐步获得的，就造成了死锁的可能。

在MySQL中，行级锁并不是直接锁记录，而是锁索引。索引分为主键索引和非主键索引两种，如果一条sql语句操作了主键索引，MySQL就会锁定这条主键索引；如果一条语句操作了非主键索引，MySQL会先锁定该非主键索引，再锁定相关的主键索引。在UPDATE、DELETE操作时，MySQL不仅锁定WHERE条件扫描过的所有索引记录，而且会锁定相邻的键值，即所谓的next-key locking。

当两个事务同时执行，一个锁住了主键索引，在等待其他相关索引。另一个锁定了非主键索引，在等待主键索引。这样就会发生死锁。发生死锁后，InnoDB一般都可以检测到，并使一个事务释放锁回退，另一个获取锁完成事务。

有多种方法可以避免死锁，这里只介绍常见的三种：

1、如果不同程序会并发存取多个表，尽量约定以相同的顺序访问表，可以大大降低死锁机会。

2、在同一个事务中，尽可能做到一次锁定所需要的所有资源，减少死锁产生概率；

3、对于非常容易产生死锁的业务部分，可以尝试使用升级锁定颗粒度，通过表级锁定来减少死锁产生的概率；

### 对比

什么时候使用行锁和表锁？

InnoDB行锁是通过给索引上的索引项加锁来实现的，这一点MySQL与Oracle不同，后者是通过在数据块中对相应数据行加锁来实现的。InnoDB这种行锁实现特点意味着：只有通过索引条件检索数据，InnoDB才使用行级锁，否则，InnoDB将使用表锁！

在实际应用中，要特别注意InnoDB行锁的这一特性，不然的话，可能导致大量的锁冲突，从而影响并发性能。

在不通过索引条件查询的时候,InnoDB 确实使用的是表锁,而不是行锁。

由于 MySQL 的行锁是针对索引加的锁,不是针对记录加的锁,所以虽然是访问不同行的记录,但是如果是使用相同的索引键,是会出现锁冲突的。应用设计的时候要注意这一点。

当表有多个索引的时候,不同的事务可以使用不同的索引锁定不同的行,另外,不论 是使用主键索引、唯一索引或普通索引,InnoDB 都会使用行锁来对数据加锁。

即便在条件中使用了索引字段,但是否使用索引来检索数据是由 MySQL 通过判断不同 执行计划的代价来决定的,如果 MySQL 认为全表扫效率更高,比如对一些很小的表,它就不会使用索引,这种情况下 InnoDB 将使用表锁,而不是行锁。因此,在分析锁冲突时, 别忘了检查 SQL 的执行计划,以确认是否真正使用了索引。

## 表锁

### 概述

表级锁是MySQL中锁定粒度最大的一种锁，表示对当前操作的整张表加锁，它实现简单，资源消耗较少，被大部分MySQL引擎支持。

最常使用的MYISAM与INNODB都支持表级锁定。

表级锁定分为表共享读锁与表独占写锁。

### 特点

开销小，加锁快；不会出现死锁；锁定粒度大，发出锁冲突的概率最高，并发度最低。

### 分类

MySQL的表级锁有两种模式：表共享读锁（Table Read Lock）和表独占写锁（Table Write Lock）。

使用lock table/unlock table进行加锁/解锁。

## 页锁

### 概述

页级锁是MySQL中锁定粒度介于行级锁和表级锁中间的一种锁。表级锁速度快，但冲突多，行级冲突少，但速度慢。所以取了折衷的页级，一次锁定相邻的一组记录。

BDB支持页级锁。

### 特点

开销和加锁时间界于表锁和行锁之间；会出现死锁；锁定粒度界于表锁和行锁之间，并发度一般。

### 分类

## 总结

MyISAM和MEMORY采用表级锁(table-level locking)。

BDB采用页面锁(page-level locking)或表级锁，默认为页面锁。

InnoDB支持行级锁(row-level locking)和表级锁,默认为行级锁。

行级锁：开销大，加锁慢，会出现死锁，锁粒度小，发生锁冲突的概率最低，并发度最高。

表级锁：开销小，加锁快，不会产生死锁，锁粒度大，发生锁冲突的概率最高，并发度最低。

页面锁：开销和加锁时间介于表锁和行锁之间，会产生死锁，锁粒度介于表锁和行锁之间，并发度一般。

综上所述，很难笼统地说哪种锁更好，只能就具体应用的特点选择合适的锁！仅从锁的角度来看，表级锁更适合查询为主，只有少量按索引更新数据的应用场景，如web应用；而行级锁则更适合于有大量按照索引条件并发更新少量不同数据，同时又有并发查询的应用，如一些在线事务处理（OLAP）系统。

## ----

## 乐观锁

### 概述

乐观锁的特点是先进行业务操作，不到万不得已不去拿锁，即“乐观”的认为拿锁多半是成功的，因此在进行完业务操作需要实际更新数据的最后一步再去拿一下锁就好。

### 实现

乐观锁在数据库上的实现完全是逻辑的，不需要数据库提供特殊的支持。

乐观锁采用版本号（或者时间戳）的方式，即当前版本号如果对应上了就可以写入数据，如果判断当前版本号不一致，那么就不会更新成功。

比如：

update table

set

column = value

where

version=${version}

and

otherKey = ${otherKey}

实现步骤：

1. select data as lod\_data,version as old\_version from…
2. 根据获取的数据进行业务操作，得到new\_data和new\_version
3. update set data=new\_data,version=new\_version where version=old\_version

if(update row > 0){

//乐观锁获取成功，操作完成

}else{

//乐观锁获取失败，回滚并重试

}

乐观锁是否在事务中其实都是无所谓的，其底层机制是这样的：在数据库内部update同一行的时候是不允许并发的，即数据库每次执行一条update语句的时候会获取被update行的写锁，直到这一行被成功更新后才释放。因此在业务操作进行前获取需要锁的数据的当前版本号，然后实际更新数据时再次对比版本号确认与之前获取的时间，并更新版本号，即可确认这之间没有发生并发的修改。如果更新失败即可认为老版本的数据已经被并发修改掉而不存在了，此时认为获取锁失败，需要回滚整个业务操作并可根据需要重试整个过程。

### 适用场景

乐观锁适用于写少读多的情景，因为这种乐观锁相当于JAVA的CAS，所以多条数据同时过来的时候，不用等待，可以立即进行返回。

悲观锁适用于写多读少的情景，这种情况也相当于JAVA的synchronized，reentrantLock等，大量数据过来的时候，只有一条数据可以被写入，其他的数据需要等待。执行完成后下一条数据可以继续。

## 悲观锁

### 概述

总是假设最坏的情况，每次取数据时都认为其他线程会修改，所以都会加锁（读锁、写锁、行锁等），当其他线程想要访问数据时，都需要阻塞挂起。可以根据数据库实现，如行锁、读锁和写锁等，都是在操作之前加锁。在Java中，synchronize的思想也是悲观锁。

### 实现

悲观锁实现的机制一般是在执行更新语句的时候采用for update方式。

比如：

update table

set

column=

'value'

for

update

这种情况where条件一定要涉及到数据库对应的索引字段，这样才会是行级锁，否则会是表锁，这样执行速度会变慢。

### 适用场景

综上所述，乐观锁在不发生取锁失败的情况下开销比悲观锁小，但是一旦发生失败回滚开销则比较大，因此适合用在取锁失败概率比较小的场景，可以提升系统并发性能。乐观锁还适用于一些比较特殊的场景，例如在业务操作过程中无法和数据库保持连接等悲观锁无法适用的地方。