MySQL 中的悲观锁和乐观锁

在分布式系统的大环境下，各种锁机制扮演着重要角色：从宏观的分布式锁到微观的代码锁，再到数据库层面的锁。尽管锁的类型繁多且复杂，但它们的核心功能是一致的：确保在某一时刻，仅有一个线程能够执行操作，其他线程则需等待锁的释放才能继续运行。

锁的作用是针对写入操作而设计的，它并不适用于读取操作，因为通常读取数据并不需要加锁。我们今天要探讨的是 MySQL 中的锁机制，以及它在众多锁类型中的位置。理解 MySQL 锁的特性对于我们更好地应用它们至关重要。让我们从应用层的锁开始讨论。

应用的锁和数据库的锁

在任何编程语言中，都存在锁的实现方式，例如 Java 中的 Synchronized 关键字或 Go 语言中的 Mutex。在业务场景中，即使数据库不提供锁机制，我们也可以通过代码实现线程安全。更常见的做法是在应用程序中使用锁，因为开发者通常更熟悉自己的代码，而将数据库视为外部存储，只需支持基本的增删改查功能。

以唯一性校验为例，这一需求既可以通过应用程序中的逻辑实现，也可以利用数据库的唯一性约束来完成，两者都能达到唯一性约束的效果。

在分布式系统中，为了实现对锁的集中管理，我们通常需要引入分布式锁。Redis 常被用作锁的中心节点。然而，除了 Redis 之外，MySQL 本身也可以作为锁的中心节点，这似乎有点本末倒置。如果能够利用 MySQL 的锁机制，那么复杂的分布式架构就可以简化为传统的应用加数据库模型。

悲观锁

接下来，我们来讨论 MySQL 中的悲观锁。悲观锁是一种显式锁，其语法清晰可见，并且需要依赖于 MySQL 的 InnoDB 存储引擎和事务机制才能生效。

悲观锁的实现通常与”select for update”语句相关，但这并不完全准确。实际

上，MySQL 悲观锁的核心语法是”update”。当两条 update 语句试图同时操作同一行数据时，只有一条语句能够执行，另一条则需等待。

后来，人们意识到在某些情况下，select 语句也需要加锁，以保持事务的排他性。因此，”select for update”语句应运而生，它允许 **select 语句像 update 一样可以锁定行，直到事务执行完才能释放**，我们来看下具体例子：

```
begin;
--A操作
select name from user where id=1 for update;
--B操作
commit;
```

这段事务中 select for update 夹在 A 操作和 B 操作之间，当执行到 select for update 时会对 ID 为 1 的行加锁，B 操作执行时其他线程不能操作当前的行，直到 commit 执行才能释放锁。我们再来看一段代码：

```
begin;
--A操作
update user set name=’yafeng’ where id=1;
--B操作
commit;
```

这次我们把 select for update 语句换成 update 语句，当执行到 update 时会对 ID 为 1 的行加锁，B 操作执行时其他线程不能操作当前的行，直到 commit 执行才能释放锁。

可见，上面两段代码在锁的效果上完全一样。

通过 MySQL 的 sleep 函数，我们可以模拟事务执行过程中的延迟，从而观察到锁的持有和释放过程。这有助于理解在实际应用中，锁是如何影响并发事务的执行。

```
begin;
--A操作
update user set name='yafeng' where id=1;
select sleep(20);
commit;
```

上面执行完 update 语句以后将 sleep 20 秒再 commit，这时如果你打开另一个窗口执行其他 update 操作，那么你的行为将被阻塞。

值得注意的是，MySQL 的悲观锁默认作用于具有唯一索引的数据行。如果查询条件不涉及唯一索引，MySQL 可能会升级锁的范围，从行级锁变为表级锁，这在某些情况下可能会影响数据库的性能。

```
SELECT * FROM user WHERE id=1 FOR UPDATE;
```

比如上面，ID 为 1 的行确实存在，并且 ID 是唯一索引，因此会锁定这一行，这就是我们常说的行级锁。如果 ID 对应的行不存在，则不会产生任何锁。

```
SELECT * FROM user WHERE id>1 FOR UPDATE;
```

我们稍微改一下这个 SQL 条件，此时 ID 指向一个不明确的，或者是无限的范围，MySQL 找不到具体的行就会给整个表加锁。

```
SELECT * FROM user WHERE name='yafeng' FOR UPDATE;
```

我们再改一下这个 SQL，name 列没有唯一索引，MySQL 依然会给整个表加锁。MySQL 行级锁锁的是有限的唯一索引，找不到有限的唯一索引，就会锁表。

总的来说，悲观锁是 MySQL 中一种重要的数据一致性保障机制，通过显式的锁定操作，它能够有效地处理并发事务中的冲突问题。然而，开发者在使用时也需要考虑到锁的粒度和性能影响，以确保系统的高效运行。

乐观锁

乐观锁是一种与悲观锁相对的数据一致性保障机制，它基于一种乐观的假设：在大多数情况下，数据的并发冲突是罕见的。这种锁的实现通常不依赖于数据库的显式锁定，而是通过应用逻辑来确保数据的一致性。

在乐观锁的实现中，通常会引入一个额外的字段，如版本号 version，来跟踪数据的变更。每次数据更新时，版本号都会递增，这样在提交更新之前，系统会检查版本号是否与读取时的值一致，如果不一致，说明数据在读取后已被其他事务修改。

使用版本号时，可以在数据初始化时指定一个版本号，在每次数据更新时都对版本号进行+1 操作。

```
select * from user where id=1
update user set name=#{name},version=version+1 where id=1 and versio
n=0;
```

假设 2 个线程分别执行上面的语句，线程 A 和线程 B 为 name 传入不同的值，其中 select 语句是可以并行执行的，假设 select 语句执行以后返回的 version 字段值为 0，那么下面我们就该执行 update 语句。而因为 MySQL 悲观锁的特性，两个线程不可能同时 update 一条数据，所以在 update 同一条数据的时候，是有先后顺序的，只有在第一个线程执行完 update，才能释放行锁，让第二个线程继续进行 update。

第一个线程执行完成后，version 字段值将变成 1，所以第二个线程修改失败，实现了乐观锁控制。

这种机制的优势在于它允许多个事务并发执行，减少了锁的争用，从而提高了系统的吞吐量。然而，它也要求开发者在应用层面上实现额外的逻辑来处理可能的冲突。

总结

今天我们深入探讨了 MySQL 中的两种锁机制：悲观锁和乐观锁。实际上，MySQL 原生只支持悲观锁，而乐观锁是通过巧妙地利用现有机制实现的。

通过使用 MySQL 的锁，我们可以在分布式架构中减少对外部锁机制的依赖，简化系统设计。这使得传统的应用与数据库模型更加高效地协同工作。尽管 MySQL 的锁机制在许多场景下都非常有用，但它们并不适用于所有情况。例如，在面对高并发的秒杀活动时，可能会更倾向于使用性能更优的 NoSQL 解决方案，如 Redis，来处理分布式锁的需求。

然而，在大多数常规业务场景中，MySQL 的锁已经足够应对需求。开发者可以根据具体的业务需求和性能考量，选择最合适的锁策略。

原文链接: https://juejin.cn/post/7387700215657021455