



下载APP



09 | 事务：如何安全地实现多key操作？

2021-02-08 唐聪

etcd实战课

[进入课程 >](#)**讲述：王超凡**

时长 19:42 大小 18.04M



你好，我是唐聪。

在软件开发过程中，我们经常会遇到需要批量执行多个 key 操作的业务场景，比如转账案例中，Alice 给 Bob 转账 100 元，Alice 账号减少 100，Bob 账号增加 100，这涉及到多个 key 的原子更新。

无论发生任何故障，我们应用层期望的结果是，要么两个操作一起成功，要么两个一起失败。我们无法容忍出现一个成功，一个失败的情况。那么 etcd 是如何解决多 key 原子更新问题呢？



这正是我今天要和你分享的主题——事务，它就是为了**简化应用层的编程模型**而诞生的。我将通过转账案例为你剖析 etcd 事务实现，让你了解 etcd 如何实现事务 ACID 特性的，


以及 MVCC 版本号在事务中的重要作用。希望通过本节课，帮助你在业务开发中正确使用事务，保证软件代码的正确性。

事务特性初体验及 API

如何使用 etcd 实现 Alice 向 Bob 转账功能呢？

在 etcd v2 的时候，etcd 提供了 CAS (Compare and swap)，然而其只支持单 key，不支持多 key，因此无法满足类似转账场景的需求。严格意义上说 CAS 称不上事务，无法实现事务的各个隔离级别。

etcd v3 为了解决多 key 的原子操作问题，提供了全新迷你事务 API，同时基于 MVCC 版本号，它可以实现各种隔离级别的事务。它的基本结构如下：

 复制代码

```
1 client.Txn(ctx).If(cmp1, cmp2, ...).Then(op1, op2, ...).Else(op1, op2, ...)
```

从上面结构中你可以看到，**事务 API 由 If 语句、Then 语句、Else 语句组成**，这与我们平时常见的 MySQL 事务完全不一样。

它的基本原理是，在 If 语句中，你可以添加一系列的条件表达式，若条件表达式全部通过检查，则执行 Then 语句的 get/put/delete 等操作，否则执行 Else 的 get/put/delete 等操作。

那么 If 语句支持哪些检查项呢？

首先是 **key 的最近一次修改版本号 mod_revision**，简称 mod。你可以通过它检查 key 最近一次被修改时的版本号是否符合你的预期。比如当你查询到 Alice 账号资金为 100 元时，它的 mod_revision 是 v1，当你发起转账操作时，你得确保 Alice 账号上的 100 元未被挪用，这就可以通过 `mod("Alice") = "v1"` 条件表达式来保障转账安全性。

其次是 **key 的创建版本号 create_revision**，简称 create。你可以通过它检查 key 是否已存在。比如在分布式锁场景里，只有分布式锁 key(lock) 不存在的时候，你才能发起 put

操作创建锁，这时你可以通过 `create("lock") = "0"` 来判断，因为一个 key 不存在的话它的 `create_revision` 版本号就是 0。

接着是 **key 的修改次数 version**。你可以通过它检查 key 的修改次数是否符合预期。比如你期望 key 在修改次数小于 3 时，才能发起某些操作时，可以通过 `version("key") < "3"` 来判断。

最后是 **key 的 value 值**。你可以通过检查 key 的 value 值是否符合预期，然后发起某些操作。比如期望 Alice 的账号资金为 200, `value("Alice") = "200"`。

If 语句通过以上 MVCC 版本号、value 值、各种比较运算符 (等于、大于、小于、不等于)，实现了灵活的比较的功能，满足你各类业务场景诉求。

下面我给出了一个使用 `etcdctl` 的 `txn` 事务命令，基于以上介绍的特性，初步实现的一个 Alice 向 Bob 转账 100 元的事务。

Alice 和 Bob 初始账上资金分别都为 200 元，事务首先判断 Alice 账号资金是否为 200，若是则执行转账操作，不是则返回最新资金。etcd 是如何执行这个事务的呢？**这个事务实现上有哪些问题呢？**

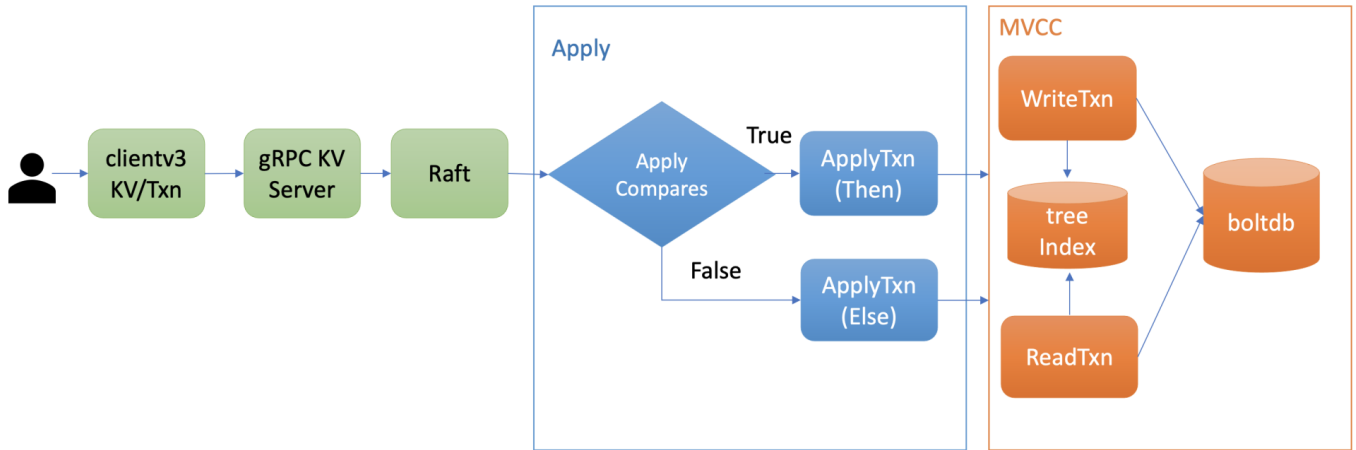
[复制代码](#)

```
1 $ etcdctl txn -i
2 compares: //对应If语句
3 value("Alice") = "200" //判断Alice账号资金是否为200
4
5
6 success requests (get, put, del): //对应Then语句
7 put Alice 100 //Alice账号初始资金200减100
8 put Bob 300 //Bob账号初始资金200加100
9
10
11 failure requests (get, put, del): //对应Else语句
12 get Alice
13 get Bob
14
15
16 SUCCESS
17
18
19 OK
20
```

21 OK

22

整体流程



在和你介绍上面案例中的 etcd 事务原理和问题前，我先给你介绍下事务的整体流程，为我们后面介绍 etcd 事务 ACID 特性的实现做准备。

上图是 etcd 事务的执行流程，当你通过 client 发起一个 txn 转账事务操作时，通过 gRPC KV Server、Raft 模块处理后，在 Apply 模块执行此事务的时候，它首先对你的事务的 If 语句进行检查，也就是 ApplyCompares 操作，如果通过此操作，则执行 ApplyTxn/Then 语句，否则执行 ApplyTxn/Else 语句。

在执行以上操作过程中，它会根据事务是否只读、可写，通过 MVCC 层的读写事务对象，执行事务中的 get/put/delete 各操作，也就是我们上一节课介绍的 MVCC 对 key 的读写原理。

事务 ACID 特性

了解完事务的整体执行流程后，那么 etcd 应该如何正确实现上面案例中 Alice 向 Bob 转账的事务呢？别着急，我们先来了解一下事务的 ACID 特性。在你了解了 etcd 事务 ACID 特性实现后，这个转账事务案例的正确解决方案也就简单了。

ACID 是衡量事务的四个特性，由原子性 (Atomicity)、一致性 (Consistency)、隔离性 (Isolation)、持久性 (Durability) 组成。接下来我就为你分析 ACID 特性在 etcd 中

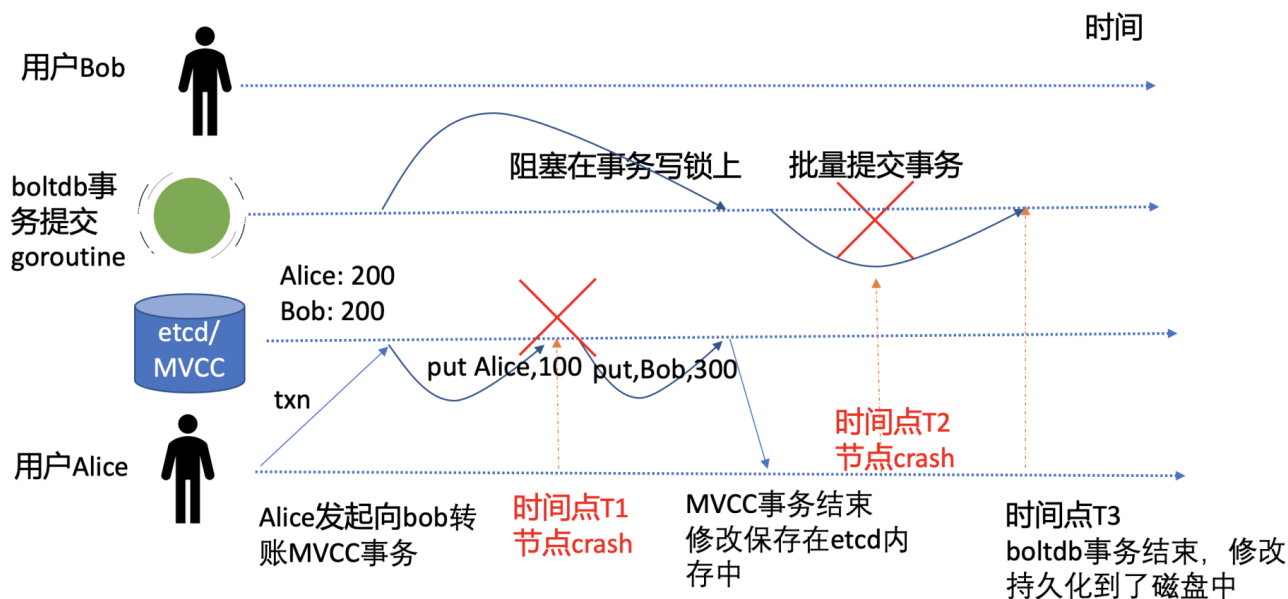
的实现。

原子性与持久性

事务的原子性（Atomicity）是指在一个事务中，所有请求要么同时成功，要么同时失败。比如在我们的转账案例中，是绝对无法容忍 Alice 账号扣款成功，但是 Bob 账号资金到账失败的场景。

持久性（Durability）是指事务一旦提交，其所做的修改会永久保存在数据库。

软件系统在运行过程中会遇到各种各样的软硬件故障，如果 etcd 在执行上面事务过程中，刚执行完扣款命令（put Alice 100）就突然 crash 了，它是如何保证转账事务的原子性与持久性的呢？



如上图转账事务流程图所示，etcd 在执行一个事务过程中，任何时间点都可能会出现节点 crash 等异常问题。我在图中给你标注了两个关键的异常时间点，它们分别是 T1 和 T2。接下来我分别为你分析一下 etcd 在这两个关键时间点异常后，是如何保证事务的原子性和持久性的。

T1 时间点

T1 时间点是在 Alice 账号扣款 100 元完成时，Bob 账号资金还未成功增加时突然发生了 crash。

从前面介绍的 etcd 写原理和上面流程图我们可知，此时 MVCC 写事务持有 boltdb 写锁，仅是将修改提交到了内存中，保证幂等性、防止日志条目重复执行的一致性索引 consistent index 也并未更新。同时，负责 boltdb 事务提交的 goroutine 因无法持有写锁，也并未将事务提交到持久化存储中。

因此，T1 时间点发生 crash 异常后，事务并未成功执行和持久化任意数据到磁盘上。在节点重启时，etcd server 会重放 WAL 中的已提交日志条目，再次执行以上转账事务。因此不会出现 Alice 扣款成功、Bob 到帐失败等严重 Bug，极大简化了业务的编程复杂度。

T2 时间点

T2 时间点是在 MVCC 写事务完成转账，server 返回给 client 转账成功后，boltdb 的事务提交 goroutine，批量将事务持久化到磁盘中时发生了 crash。这时 etcd 又是如何保证原子性和持久性的呢？

我们知道一致性索引 consistent index 字段值是和 key-value 数据在一个 boltdb 事务里同时持久化到磁盘中的。若在 boltdb 事务提交过程中发生 crash 了，简单情况是 consistent index 和 key-value 数据都更新失败。那么当节点重启，etcd server 重放 WAL 中已提交日志条目时，同样会再次应用转账事务到状态机中，因此事务的原子性和持久化依然能得到保证。

更复杂的情况是，当 boltdb 提交事务的时候，会不会部分数据提交成功，部分数据提交失败呢？这个问题，我将在下一节课通过深入介绍 boltdb 为你解答。

了解完 etcd 事务的原子性和持久性后，那一一致性又是怎么回事呢？事务的一致性难道是指各个节点数据一致性吗？

一致性

在软件系统中，到处可见一致性（Consistency）的表述，其实在不同场景下，它的含义是不一样的。

首先分布式系统中多副本数据一致性，它是指各个副本之间的数据是否一致，比如 Redis 的主备是异步复制的，那么它的一致性是最最终一致性的。

其次是 CAP 原理中的一致性是指可线性化。核心原理是虽然整个系统是由多副本组成，但是通过线性化能力支持，对 client 而言就如一个副本，应用程序无需关心系统有多少个副本。

然后是一致性哈希，它是一种分布式系统中的数据分片算法，具备良好的分散性、平衡性。

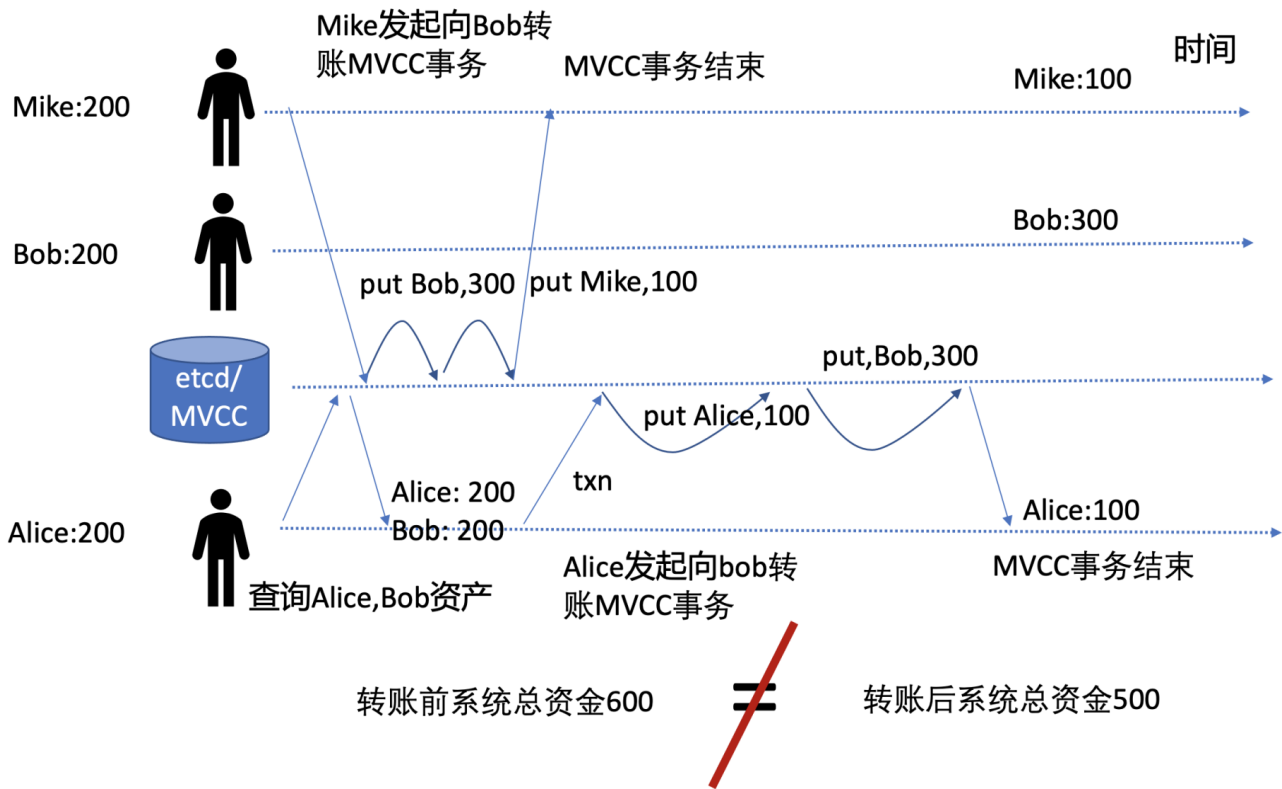
最后是事务中的一致性，它是指事务变更前后，数据库必须满足若干恒等条件的状态约束，**一致性往往是由数据库和业务程序两方面来保障的。**

在 Alice 向 Bob 转账的案例中有哪些恒等状态呢？

很明显，转账系统内的各账号资金总额，在转账前后应该一致，同时各账号资产不能小于 0。

为了帮助你更好地理解前面转账事务实现的问题，下面我给你画了幅两个并发转账事务的流程图。

图中有两个并发的转账事务，Mike 向 Bob 转账 100 元，Alice 也向 Bob 转账 100 元，按照我们上面的事务实现，从下图可知转账前系统总资金是 600 元，转账后却只有 500 元了，因此它无法保证转账前后账号系统内的资产一致性，导致了资产凭空消失，破坏了事务的一致性。



事务一致性被破坏的根本原因是，事务中缺少对 Bob 账号资产是否发生变化的判断，这就导致账号资金被覆盖。

为了确保事务的一致性，一方面，业务程序在转账逻辑里面，需检查转账者资产大于等于转账金额。在事务提交时，通过账号资产的版本号，确保双方账号资产未被其他事务修改。若双方账号资产被其他事务修改，账号资产版本号会检查失败，这时业务可以通过获取最新的资产和版本号，发起新的转账事务流程解决。

另一方面，etcd 会通过 WAL 日志和 consistent index、boltdb 事务特性，去确保事务的原子性，因此不会有部分成功部分失败的操作，导致资金凭空消失、新增。

介绍完事务的原子性和持久化、一致性后，我们再看看 etcd 又是如何提供各种隔离级别的事务，在转账过程中，其他 client 能看到转账的中间状态吗 (如 Alice 扣款成功，Bob 还未增加时)？

隔离性

ACID 中的 I 是指 Isolation，也就是事务的隔离性，它是指事务在执行过程中的可见性。常见的事务隔离级别有以下四种。

首先是**未提交读**（Read UnCommitted），也就是一个 client 能读取到未提交的事务。比如转账事务过程中，Alice 账号资金扣除后，Bob 账号上资金还未增加，这时如果其他 client 读取到这种中间状态，它会发现系统总金额钱减少了，破坏了事务一致性的约束。

其次是**已提交读**（Read Committed），指的是只能读取到已经提交的事务数据，但是存在不可重复读的问题。比如事务开始时，你读取了 Alice 和 Bob 资金，这时其他事务修改 Alice 和 Bob 账号上的资金，你在事务中再次读取时会读取到最新资金，导致两次读取结果不一样。

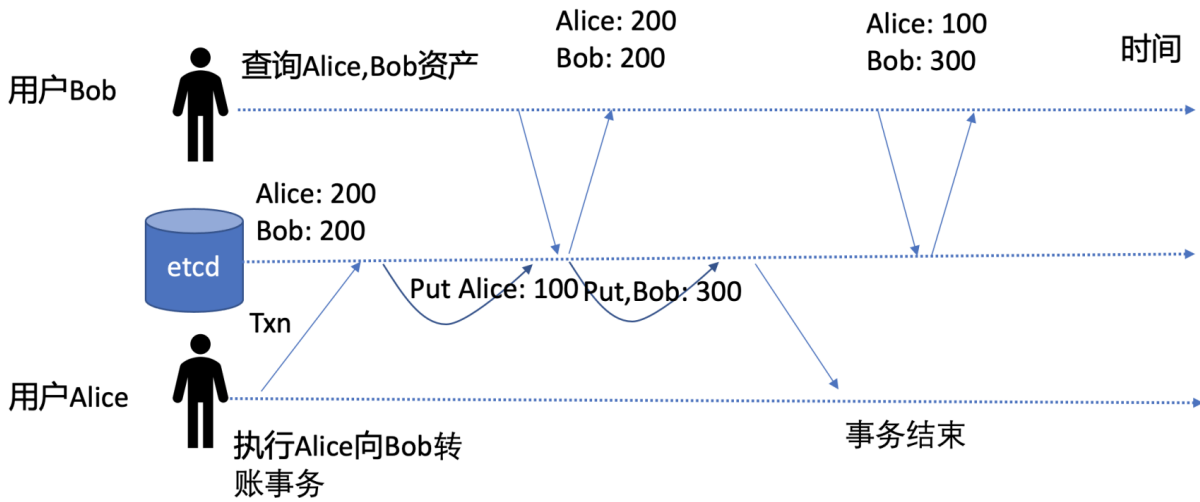
接着是**可重复读**（Repeated Read），它是指在一个事务中，同一个读操作 get Alice/Bob 在事务的任意时刻都能得到同样的结果，其他修改事务提交后也不会影响你本事务所看到的结果。

最后是**串行化**（Serializable），它是最高的事务隔离级别，读写相互阻塞，通过牺牲并发能力、串行化来解决事务并发更新过程中的隔离问题。对于串行化我要和你特别补充一点，很多人认为它都是通过读写锁，来实现事务一个个串行提交的，其实这只是在基于锁的并发控制数据库系统实现而已。**为了优化性能，在基于 MVCC 机制实现的各个数据库系统中，提供了一个名为“可串行化的快照隔离”级别，相比悲观锁而言，它是一种乐观并发控制，通过快照技术实现的类似串行化的效果，事务提交时能检查是否冲突。**

下面我重点和你介绍下未提交读、已提交读、可重复读、串行化快照隔离。

未提交读

首先是最低的事务隔离级别，未提交读。我们通过如下一个转账事务时间序列图，来分析下一个 client 能否读取到未提交事务修改的数据，是否存在脏读。



图中有两个事务，一个是用户查询 Alice 和 Bob 资产的事务，一个是我们执行 Alice 向 Bob 转账的事务。

如图中所示，若在 Alice 向 Bob 转账事务执行过程中，etcd server 收到了 client 查询 Alice 和 Bob 资产的读请求，显然此时我们无法接受 client 能读取到一个未提交的事务，因为这对应用程序而言会产生严重的 BUG。那么 etcd 是如何保证不出现这种场景呢？

我们知道 etcd 基于 boltdb 实现读写操作的，读请求由 boltdb 的读事务处理，你可以理解为快照读。写请求由 boltdb 写事务处理，etcd 定时将一批写操作提交到 boltdb 并清空 buffer。

由于 etcd 是批量提交写事务的，而读事务又是快照读，因此当 MVCC 写事务完成时，它需要更新 buffer，这样下一个读请求到达时，才能从 buffer 中获取到最新数据。

在我们的场景中，转账事务并未结束，执行 put Alice 为 100 的操作不会回写 buffer，因此避免了脏读的可能性。用户此刻从 boltdb 快照读事务中查询到的 Alice 和 Bob 资产都为 200。

从以上分析可知，etcd 并未使用悲观锁来解决脏读的问题，而是通过 MVCC 机制来实现读写不阻塞，并解决脏读的问题。

已提交读、可重复读

比未提交读隔离级别更高的是已提交读，它是指在事务中能读取到已提交数据，但是存在不可重复读的问题。已提交读，也就是说你每次读操作，若未增加任何版本号限制，默认

都是当前读，etcd 会返回最新已提交的事务结果给你。

如何理解不可重复读呢？

在上面用户查询 Alice 和 Bob 事务的案例中，第一次查出来资产都是 200，第二次是 Alice 为 100，Bob 为 300，通过读已提交模式，你能及时获取到 etcd 最新已提交的事务结果，但是出现了不可重复读，两次读出来的 Alice 和 Bob 资产不一致。

那么如何实现可重复读呢？

你可以通过 MVCC 快照读，或者参考 etcd 的事务框架 STM 实现，它在事务中维护一个读缓存，优先从读缓存中查找，不存在则从 etcd 查询并更新到缓存中，这样事务中后续读请求都可从缓存中查找，确保了可重复读。

最后我们再来重点介绍下什么是串行化快照隔离。

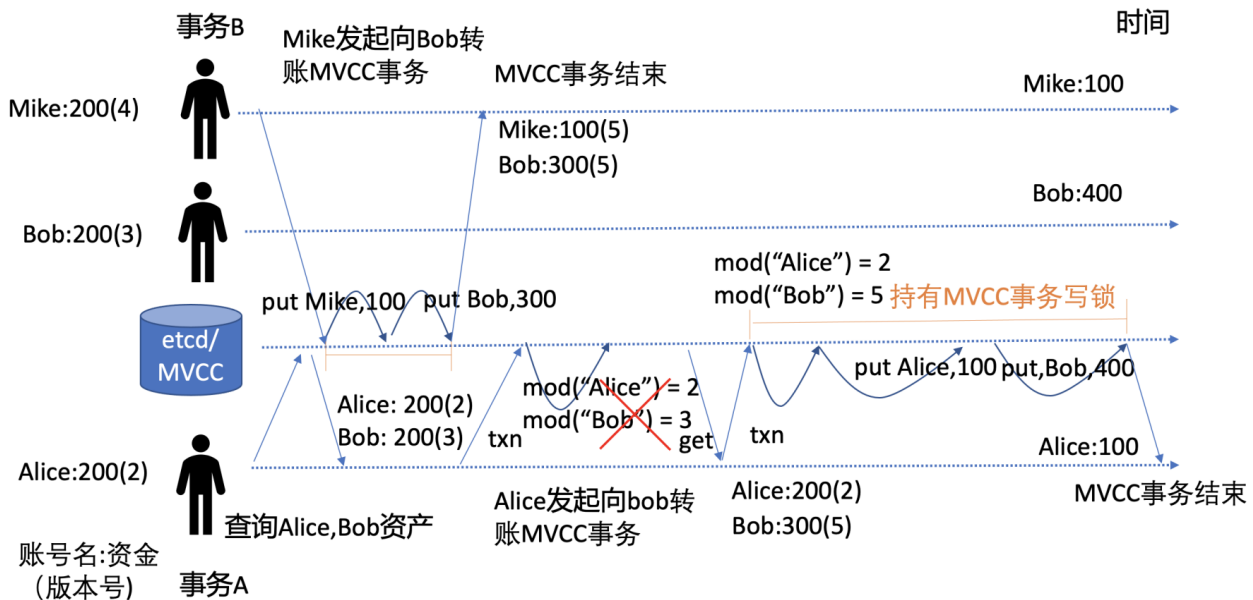
串行化快照隔离

串行化快照隔离是最严格的事务隔离级别，它是指在在事务刚开始时，首先获取 etcd 当前的版本号 rev，事务中后续发出的读请求都带上这个版本号 rev，告诉 etcd 你需要获取那个时间点的快照数据，etcd 的 MVCC 机制就能确保事务中能读取到同一时刻的数据。

同时，它还要确保事务提交时，你读写的数据都是最新的，未被其他人修改，也就是要增加冲突检测机制。当事务提交出现冲突的时候依赖 client 重试解决，安全地实现多 key 原子更新。

那么我们应该如何为上面一致性案例中，两个并发转账的事务，增加冲突检测机制呢？

核心就是我们前面介绍 MVCC 的版本号，我通过下面的并发转账事务流程图为你解释它是如何工作的。



如上图所示，事务 A，Alice 向 Bob 转账 100 元，事务 B，Mike 向 Bob 转账 100 元，两个事务同时发起转账操作。

一开始时，Mike 的版本号 (指 `mod_revision`) 是 4，Bob 版本号是 3，Alice 版本号是 2，资产各自 200。为了防止并发写事务冲突，etcd 在一个写事务开始时，会独占一个 MVCC 大写锁。

事务 A 会先去 etcd 查询当前 Alice 和 Bob 的资产版本号，用于在事务提交时做冲突检测。在事务 A 查询后，事务 B 获得 MVCC 写锁并完成转账事务，Mike 和 Bob 账号资产分别为 100，300，版本号都为 5。

事务 B 完成后，事务 A 获得写锁，开始执行事务。

为了解决并发事务冲突问题，事务 A 中增加了冲突检测，期望的 Alice 版本号应为 2，Bob 为 3。结果事务 B 的修改导致 Bob 版本号变成了 5，因此此事务会执行失败分支，再次查询 Alice 和 Bob 版本号和资产，发起新的转账事务，成功通过 MVCC 冲突检测规则 `mod("Alice") = 2` 和 `mod("Bob") = 5` 后，更新 Alice 账号资产为 100，Bob 资产为 400，完成转账操作。

通过上面介绍的快照读和 MVCC 冲突检测检测机制，etcd 就可实现串行化快照隔离能力。

转账案例应用

介绍完 etcd 事务 ACID 特性实现后，你很容易发现事务特性初体验中的案例问题了，它缺少了完整事务的冲突检测机制。


首先你可通过一个事务获取 Alice 和 Bob 账号的上资金和版本号，用以判断 Alice 是否有足够的金额转账给 Bob 和事务提交时做冲突检测。你可通过如下 `etcdctl txn` 命令，获取 Alice 和 Bob 账号的资产和最后一次修改时的版本号 (`mod_revision`):

[复制代码](#)

```
1 $ etcdctl txn -i -w=json
2 compares:
3
4
5 success requests (get, put, del):
6 get Alice
7 get Bob
8
9
10 failure requests (get, put, del):
11
12
13 {
14   "kvs":[
15     {
16       "key":"QWxpY2U=",
17       "create_revision":2,
18       "mod_revision":2,
19       "version":1,
20       "value":"MjAw"
21     }
22   ],
23   .....
24   "kvs":[
25     {
26       "key":"Qm9i",
27       "create_revision":3,
28       "mod_revision":3,
29       "version":1,
30       "value":"MzAw"
31     }
32   ],
33 }
```

其次发起资金转账操作，Alice 账号减去 100，Bob 账号增加 100。为了保证转账事务的准确性、一致性，提交事务的时候需检查 Alice 和 Bob 账号最新修改版本号与读取资金时的一致 (compares 操作中增加版本号检测)，以保证其他事务未修改两个账号的资金。

若 compares 操作通过检查，则执行转账操作，否则执行查询 Alice 和 Bob 账号资金操作，命令如下：

 复制代码

```
1 $ etcdctl txn -i
2 compares:
3 mod("Alice") = "2"
4 mod("Bob") = "3"
5
6
7 success requests (get, put, del):
8 put Alice 100
9 put Bob 300
10
11
12 failure requests (get, put, del):
13 get Alice
14 get Bob
15
16
17 SUCCESS
18
19
20 OK
21
22 OK
```

到这里我们就完成了一个安全的转账事务操作，从以上流程中你可以发现，自己从 0 到 1 实现一个完整的事务还是比较繁琐的，幸运的是，etcd 社区基于以上介绍的事务特性，提供了一个简单的事务框架 [STM](#)，构建了各个事务隔离级别类，帮助你进一步简化应用编程复杂度。

小结

最后我们来小结下今天的内容。首先我给你介绍了事务 API 的基本结构，它由 If、Then、Else 语句组成。

其中 If 支持多个比较规则，它是用于事务提交时的冲突检测，比较的对象支持 key 的 **mod_revision**、**create_revision**、**version**、**value 值**。随后我给你介绍了整个事务执行的基本流程，Apply 模块首先执行 If 的比较规则，为真则执行 Then 语句，否则执行 Else 语句。

接着通过转账案例，四幅转账事务时间序列图，我为你分析了事务的 ACID 特性，剖析了在 etcd 中事务的 ACID 特性的实现。

原子性是指一个事务要么全部成功要么全部失败，etcd 基于 WAL 日志、consistent index、boltdb 的事务能力提供支持。

一致性是指事务转账前后的，数据库和应用程序期望的恒等状态应该保持不变，这通过数据库和业务应用程序相互协作完成。

持久性是指事务提交后，数据不丢失，

隔离性是指事务提交过程中的可见性，etcd 不存在脏读，基于 MVCC 机制、boltdb 事务你可以实现可重复读、串行化快照隔离级别的事务，保障并发事务场景中你的数据安全性。

思考题

在数据库事务中，有各种各样的概念，比如脏读、脏写、不可重复读与读倾斜、幻读与写倾斜、更新丢失、快照隔离、可串行化快照隔离？你知道它们的含义吗？

感谢你的阅读，如果你认为这节课的内容有收获，也欢迎把它分享给你的朋友，谢谢。

提建议

12.12 大促

每日一课 VIP 年卡

10分钟，解决你的技术难题

¥159/年 ¥365/年

每日一课
VIP 年卡

仅3天，【点击】图片，立即抢购 >>>

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 08 | Watch：如何高效获取数据变化通知？

下一篇 10 | boltdb：如何持久化存储你的key-value数据？

精选留言 (1)

写留言



云原生工程师

2021-02-10

老师分析得透彻，学习了，原来自己平时使用的事务过程中，无意间已经使用了串行化快照隔离级别，还请问一下事务性能相比put接口差异大吗

展开 ∨

作者回复：嗯，肯定有不少差异的，事务执行逻辑更复杂，影响性能的因素比较多，实践篇我有介绍，建议根据自己实际业务场景，部署环境，使用etcd自带的benchmark工具压测对比下差异。



2

