如何基于 MySQL 进行数据高可用

这次探讨的话题是数据高可用，首先，我们需要理清楚数据高可用的目标，数据高可用说的是，“即使发生故障，数据也不丢失”，那么就需要明确，什么叫不丢失。

使用者视角下，完美数据库宕机会遇到何种问题？

先假设存在一个完美的数据库系统，我们忽略它是怎么做到的，只把它当成一个黑盒子，则，当承载这个数据库的服务器宕机的时候，从数据库使用者角度分析，会遇到以下几种情况：

情况一.过去执行的事务都已经提交完成，当前没有在执行事务，连接中断。

情况二.正在执行事务，还没有发出提交操作，连接中断。

情况三.正在执行事务，已经发出提交操作，连接中断。

使用者视角下，完美数据库在不同情况下应保证何种结果？

结果一，完美数据库在恢复后应该要保证过去已经提交完成的事务依旧存在，不会丢失。

结果二，完美数据库在恢复后应该要保证这样的事务被正确回滚，不能被提交。

结果三，会出现三种分支情况，**我们逐一展开**

1.提交操作到达数据库前连接就已经中断，这个时候，完美数据库在恢复时应该回滚事务；

2.提交操作已经到达数据库，这个时候，完美数据库在恢复时，应当提交这个事务；

3.数据库已经提交完成，并且发出了提交成功的信息，但是提交成功的信息在到达使用者之前，连接中断了，这个时候，完美数据库在恢复时，也应当保证这个事务依旧已经提交了。

回到使用者的视角，一个完美的数据库，此时应做到 3 保证：

第 1 保证：没有发出提交操作的事务一定被回滚。

第 2 证：保证已经发出提交操作并收到提交成功的事务一定存在，且已提交。

第 3 保证：已经发出提交操作但没有收到提交成功的事务，只存在两种状态：已提交或已回滚。

Tips

在实际生产环境中,针对 应用端“已经发出提交操作但没有收到提交成功的事务”的情形，，一定要应用程序在数据库恢复可访问后，重新检查事务是否存在，再从对应的业务角度做出相应的补偿处理。若只简单的认为这样的事务一定成功或者一定失败，都是不正确的；必须检查事务中的数据才能知道结果；只有业务的操作完全满足幂等，才能无脑重做。并且从数据库的角度，努力的上限也只是使得情况三的

第二种分支的事务可以全被提交，无法实现在情形三下，有确定结果。

现在我们可以定义，只要高可用方案，满足上述三条使用者视角下的“完美数据库保证的结果”，就可以说，高可用方案可以实现“使用者视角等效完美数据库的不丢数据”，本文后续称之为“等效不丢”,顾名思义我们所描述的是使用者视角下，和其等效，并不代表已达到完美数据库的高度。

不同灾难等级下的高可用方案

考虑现实情况，一个高可用方案，明显是不可能面对任意灾难都能保证不丢数据。因为存在不可控情况，当彻底破坏这个方案所涉及的所有存储设备（包含内存）的时候，数据定会全部丢失。因此除了明确什么叫数据不丢失之外，一个真实可操作的高可用方案，还需要定义清楚这个方案能够应对的灾难等级。

先讲最简单的一种，单台服务器因为硬件故障彻底损毁。我们应当怎么样做到“等效不丢”呢？

tips

单台服务器故障后，如果通过人工或自动重启，经过一定的处理使得数据库恢复运行，反而会带来更多的坑点与处理上的麻烦，我们后面再来详细分析。

首先，很明显，我们必须排除任何单服务器的方案。我们知道 MySQL 最常见的是简单的主从复制。

主从复制中，从机损坏显然是安全的，主机可以完整保证“等效不丢”。如果当主机损坏时，立即切换到从机，仔细分析一下，这种最基本的高可用有没有问题，有问题的话可不可以改善和解决。

很明确，等效不丢的第 1 保证明显是可以满足的，没有提交的事务根本就不会发往从机。

再看关键的第 2 保证。在普通的主从复制架构下，主机上成功提交的事务和复制的数据传输过程没有任何关联，不能保证从机收到所有在主机上已经提交的事务，因此无法实现第 2 保证。但是我们可以开启半同步复制，这样，只有从机收到了事务内容，主机才会返回提交成功，这样的方案，就奠定了实现第 2 保证的基础（还要配合后续的等待追上复制才实现）。

在讲述第三保证前，我们先弄清楚不同半同步复制下的“等效不丢”：

1. 关于 5.6 的半同步复制是否

2. 会丢数据，一直是一个争论不休的问题，现在可以得到一个答案：5.6 的半同步复制，满足“等效不丢”。但是 5.6 和 after_commit 方式，确实存在一些小问题，举例：T1 线程执行了一个事务操作，发起了提交，从机网络很慢，没有及时返回 ACK；这时 T2 线程，发起了一个事务，是可以读到 T1 线程的操作的，假设在从机收到事务前发生

故障并切换，则 T2 线程可能观察到“数据丢了”；但是 T1 并没有收到事务提交成功的信息，未来重新判定，这个事务是没有提交成功，从 T1 的视角是没有问题的。那从这个例子来看，问题很明显，5.6 和 after_commit 的方式，存在事务隔离处理上的微小缺陷。

2.半同步复制如果退化，则不能保证“等效不丢”。

3.半同步复制的场景，不应该使用 relay_log_recovery，启用这个参数后，在一些场景会删除 relaylog，这将明显破坏半同步复制的有效运行。

4.上述讨论的半同步复制，完全没有涉及到 sync_binlog、innodb_flush_log_at_trx_commit 参数的配置值，因为这个时候，数据的高可用依赖的是“持久化到网络”而非“持久化到磁盘”，数据是否刷到磁盘，跟数据的高可用完全无关，我们只关注，数据是否到安全达了另一台服务器。并且，我们这次举例的最简单场景，属于主机或者从机完全损坏，数据是否刷到磁盘，反正都是全部没了，不刷盘，数据库还能跑得更欢乐一些。

第 3 保证也是可以满足的，从机的复制机制能够保证单个事务的完整执行，实现要么有要么没有的结果。但是此时我们可以发现一个藏得比较深但很常见的异常问题：复制的 SQL 线程应用数据是有延迟的，切换后，从机可能还在不断追赶复制；从应用程序的角度看，则是出现：看到第 3 保证场景的不知道成功还是失败的事务的操作一开始丢失了，但是过了一会儿又神秘出现了（第 2 保证场景的情形，同样可

能看到这样的情况），这样显然是无法接受的。MySQL 主从复制 IO 线程延迟和 SQL 线程延迟有长有短，但是毫秒、微秒级的延迟，始终是存在的，不能通过假设复制或者要求复制没有延迟的方式去逃避这个问题。这个问题的解决实际上是可以通过等待复制追上，然后再开放从机的访问来解决的。一般高可用方案也都考虑到了这一点。

tips

从机收到的 relaylog，可能存在末尾不是一个完整的事务的现象，因此追复制过程，也需要一些很特别的处理过程，才能做到完美。很少有高可用方案，正确处理了这种情况。

小结：

经过这篇文章的探讨，使用半同步复制，配合良好完善的切换处理，可以在半同步复制没有退化，单一服务器无法恢复的损毁中（不含 MySQL BUG 、服务器后续可以恢复运行与访问、MySQL 实例 hang、以及误判断切换等场景），保证数据在“等效不丢”这个概念下的数据高可用。