Database的备份、恢复和同步技术

tianjiqx

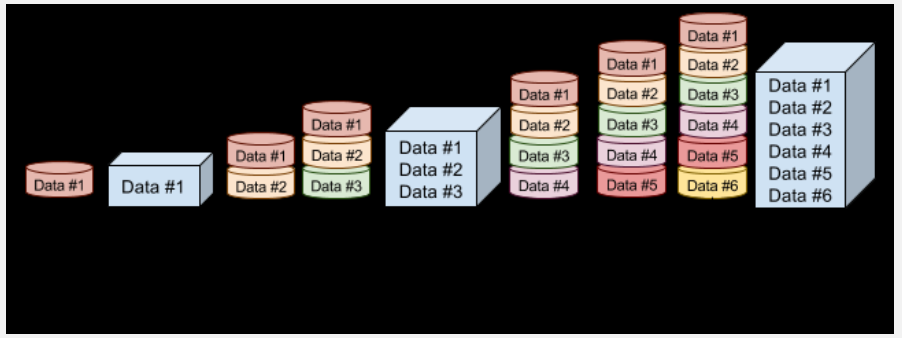
11.10

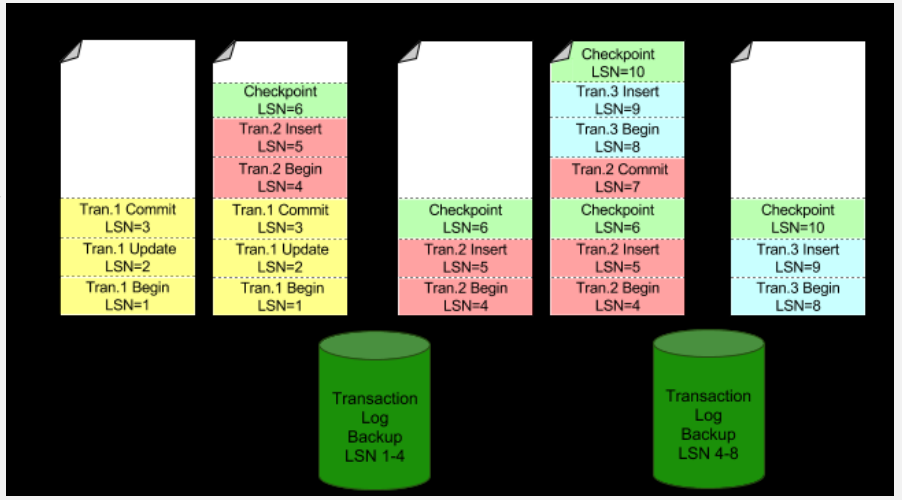
SQL Server备份

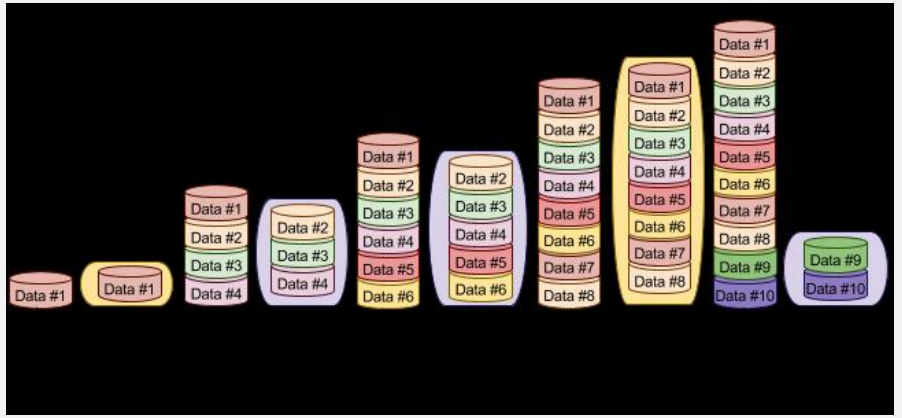
- 完全备份
  - 数据库在备份时间点的完整拷贝

- 事务日志备份
  - 事务日志备份记录了数据库从上一次日志备份到当前时间内的所有事务提交的数据变更

- 差异备份
  - 差异备份是备份至上一次数据库全量备份以来的所有变更的数据页







SQL Server备份

- 灾备策略
  - **每个小时一次完全备份**：备份文件过大，备份还原效率低下，这种方案无法实现任意时间点的还原；
  - **每天一个完全备份 + 每小时一个日志备份**：解决了备份文件过大和效率问题，也可以实现任意时间点还原，但是拉长了日志还原链条；
  - **每天一个完全备份 + 每六个小时一个差异备份 + 每小时一个日志备份**：具备任意时间点还原的能力，综合了备份文件大小、效率和备份链条长度。
- 恢复过程
  - 全备 + 日志备份
  - 全备 + 差备 + 日志备份

SQL Server恢复

- 简单恢复模式(Simple)
  - 数据库事务日志会伴随着Checkpoint或者Backup操作而被清理，最大限度的保证事务日志最小化。
  - 缺点：
    - 无法实现任意时间点恢复
    - 只能恢复到上一次的备份文件（可以是完全或者差异备份），无法恢复到最近可用状态
- 完全恢复模式(Full)
  - Checkpoint without truncate log不主动清日志
- 大容量日志恢复模式(Bulk-logged)
  - Bulk Imports：快速导入数据，记录少量日志
  - 没有Bulk Imports操作的时候，它与完全恢复模式等价，而当存在Bulk Imports操作的时候，它与简单恢复模式等价

SQL Server基于文件组冷热数据隔离备份

- 背景
  - 超大数据量，存在很多历史数据，完全备份和恢复时间过长
- 文件组备份
  - 主文件组（数量1）：存放数据库系统表、视图等对象信息，文件组可读可写
  - 自定义只读文件组（数量N）：存放历史年表的数据及相应索引数据
  - 自定义自定义可读写文件组（数量1）：当前年表数据和相应索引数据
  - 数据库事务日志文件组（数量1）：数据库事务日志

PostgreSQL备份和恢复

- SQL转储
  - 生成构成数据库的SQL文件
- 文件系统级别备份
  - 直接拷贝PostgreSQL数据文件，然后进行还原
- 连续归档和时间点恢复（PITR）
  - 利用WAL的文件系统备份，对这些WAL文件应答进行还原

PostgreSQL Flashback

- 闪回目标
  - DML闪回
  - DDL闪回
- 实现
  - 物理回退，利用物理备份和归档进行时间点恢复，全库恢复到误操作前的状态PITR
  - 当前库回退，使用HOOK，实现DROP和TRUNCATE操作的回收站
  - 使用延迟垃圾回收、脏读、行头事务号、事务提交日志，修改next XID，实现DML操作的闪回

MySQL并行复制

- MySQL知识点
  - 重做日志（redo log）
    - 确保事务的持久性
    - 物理格式的日志，记录的是物理数据页面的修改的信息
  - 回滚日志（undo log）
    - 保存了事务发生之前的数据的一个版本，可以用于回滚，同时可以提供多版本并发控制下的读（MVCC）
    - 逻辑格式的日志
  - 二进制日志（binlog）
    - 用于复制，在主从复制；数据库的基于时间点的还原
    - 逻辑格式的日志，执行过的事务中的sql语句

MySQL并行复制

- MySQL5.6前
  - 主备复制只有两个线程，IO 线程负责从主库接收 binlog 日志，并保存在本地的 relaylog 中，SQL线程负责解析和重放 relaylog 中的 event
  - 缺点：
    - 主库并行写入压力较大时，备库 IO 线程由于 relaylog 是顺序写，能够跟上
    - 但是**SQL线程重放的速度根不上，导致relaylog 一直在备库，磁盘写满**
- MySQL5.6 Schema **级别的并行复制**
  - 启动多个 Worker 线程，原有的 SQL 线程变为 Coordinator 线程
    - 可并行-> Worker线程
    - 不可并行->等待 Worker 线程全部结束后,由 Coordinator 线程执行
    - DDL 语句或者是跨 Schema（database）的语句不能并行执行
      - 适用多个 DB 同时更新
      - 简单改进DB级->Table级，但是无法解决单表热点更新

MySQL并行复制

- MySQL5.7 基于 Group Commit 的并行复制
  - 串行提交MySQL5.7 基于 Group Commit 的并行复制
    - InnoDB prepare
    - write/sync Binlog
    - InnoDB commit
  - Group Commit，分为三个阶段，每个阶段有一个线程操作，三个阶段可以并发执行
    - flush stage：binlog 从 cache 写入文件
    - sync stage： 对 binlog 做 fsync
    - commit stage：引擎层 commit

  InnoDB prepare 成功的事务可以进入队列，每个阶段可以对队列事务统一做操作，提高了并行度

MySQL并行复制

- Group Commit的binlog
  - sequence_number是自增事务ID
  - last_commited代表上一个提交的事务ID

```
$~/mysql_5710/bin/mysqlbinlog binlog.000011|grep last_committed
#180614 20:29:29 server id 2323000081  end_log_pos 219 CRC32 0x8cddd0ba        GTID    last_committed=0        sequence_number=1       rbr_only=no
#180614 20:29:47 server id 2323000081  end_log_pos 384 CRC32 0xcb763cba        GTID    last_committed=1        sequence_number=2       rbr_only=no
#180614 20:29:51 server id 2323000081  end_log_pos 550 CRC32 0x32305dfa        GTID    last_committed=2        sequence_number=3       rbr_only=yes
#180614 20:30:01 server id 2323000081  end_log_pos 807 CRC32 0x7a4b0eee        GTID    last_committed=3        sequence_number=4       rbr_only=yes
#180614 20:30:02 server id 2323000081  end_log_pos 1064 CRC32 0x2b316185       GTID    last_committed=4        sequence_number=5       rbr_only=yes
#180614 20:30:03 server id 2323000081  end_log_pos 1326 CRC32 0xa72b6d36       GTID    last_committed=5        sequence_number=6       rbr_only=yes
#180614 20:30:04 server id 2323000081  end_log_pos 1598 CRC32 0x9ba67045       GTID    last_committed=6        sequence_number=7       rbr_only=yes
#180614 20:30:35 server id 2323000081  end_log_pos 1890 CRC32 0x14f17463       GTID    last_committed=7        sequence_number=8       rbr_only=no
#180614 20:30:40 server id 2323000081  end_log_pos 2056 CRC32 0xff1e26ec       GTID    last_committed=8        sequence_number=9       rbr_only=yes
#180614 20:30:47 server id 2323000081  end_log_pos 2313 CRC32 0x14cf0488       GTID    last_committed=9        sequence_number=10      rbr_only=yes
#180614 20:30:48 server id 2323000081  end_log_pos 2570 CRC32 0x5be488a7       GTID    last_committed=10       sequence_number=11      rbr_only=yes
#180614 20:30:49 server id 2323000081  end_log_pos 2832 CRC32 0x05ccd286       GTID    last_committed=11       sequence_number=12      rbr_only=yes
#180614 20:30:49 server id 2323000081  end_log_pos 3104 CRC32 0xe20fcfdf       GTID    last_committed=12       sequence_number=13      rbr_only=yes
#180614 20:30:50 server id 2323000081  end_log_pos 3396 CRC32 0x59970167       GTID    last_committed=13       sequence_number=14      rbr_only=yes
#180614 20:33:40 server id 2323000081  end_log_pos 3728 CRC32 0xcf3ad103       GTID    last_committed=14       sequence_number=15      rbr_only=yes
#180614 20:33:41 server id 2323000081  end_log_pos 4060 CRC32 0x68e22c37       GTID    last_committed=14       sequence_number=16      rbr_only=yes
```

MySQL并行复制

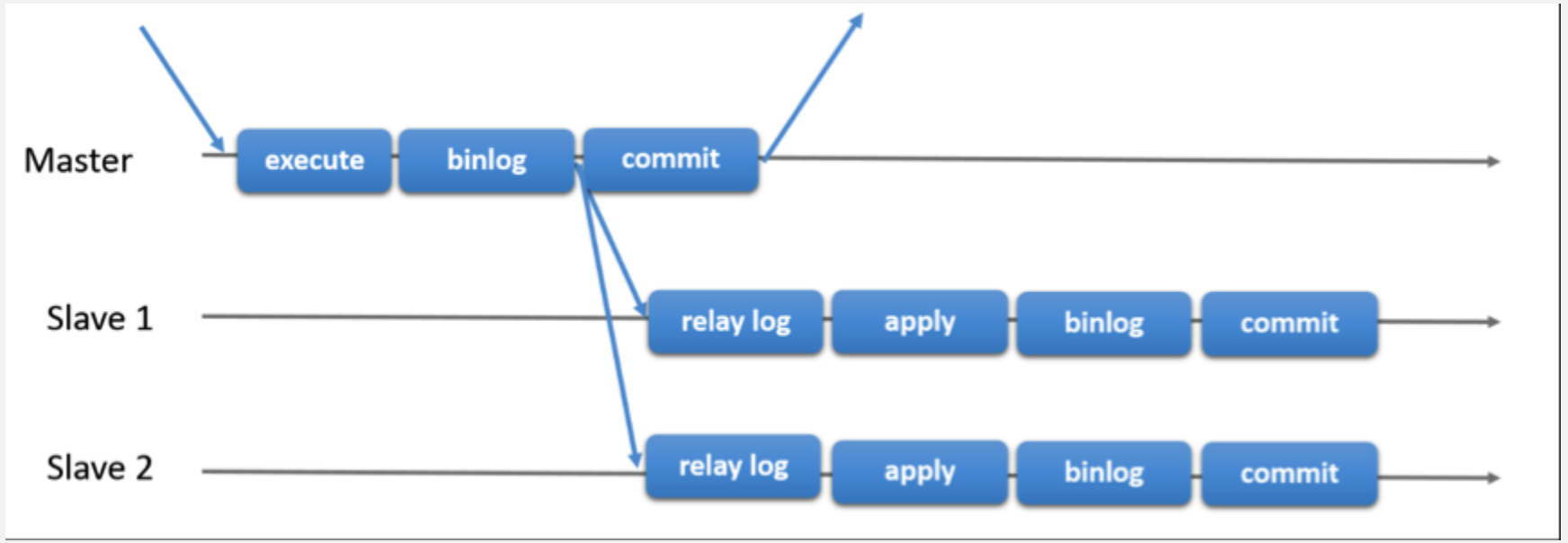
- MySQL5.7 基于 Group Commit 的并行复制
  - **LOGICAL_CLOCK 并行复制**
    - master可以并发的事务，在slave端也可以并发
    - Group Commit 实现了主库事务的并行提交
    - last_commited 相同的事务，可以在备库并行回放
  - 缺点：主库并行度低，那么备库回放时也很难并行
  - 改进：
    - binlog_group_commit_sync_delay：等待延迟提交的时间，binlog提交后等待一段时间再 fsync。让每个 group 的事务更多，人为提高并行度
    - binlog_group_commit_sync_no_delay_count：等待提交的最大事务数，如果等待时间没到，而事务数达到了，就立即 fsync。达到期望的并行度后立即提交，尽量缩小等待延迟
  - 逻辑时钟调度（Logical Clock scheduler）
    - https://dev.mysql.com/worklog/task/?id=9556
    - http://mysql.taobao.org/monthly/2017/12/03/

MySQL并行复制

- **MySQL8.0 基于 WriteSet 的并行复制（5.7.21+）**
  - 参数 binlog_transaction_dependency_tracking 控制事务依赖模式
    - COMMIT_ORDERE：使用 5.7 Group commit 的方式决定事务依赖
    - WRITESET：使用 WriteSet 的方式决定判定事务直接的冲突，发现冲突则依赖冲突事务，否则按照 COMMIT_ORDERE 方式决定依赖
  - WRITESET
    - 一个hash数组（vector<bitset>）
    - 记录了事务的更新行信息
    - https://zhuanlan.zhihu.com/p/37129637
  - 决定commit_parent时，使用事务自己的 session WriteSet 和 history WriteSet 进行比对，找到最近的冲突行，设为 commit_parent。如果 WriteSet 找不到 commit_parent，则还是使用 COMMIT_ORDERE 决定 commit_parent

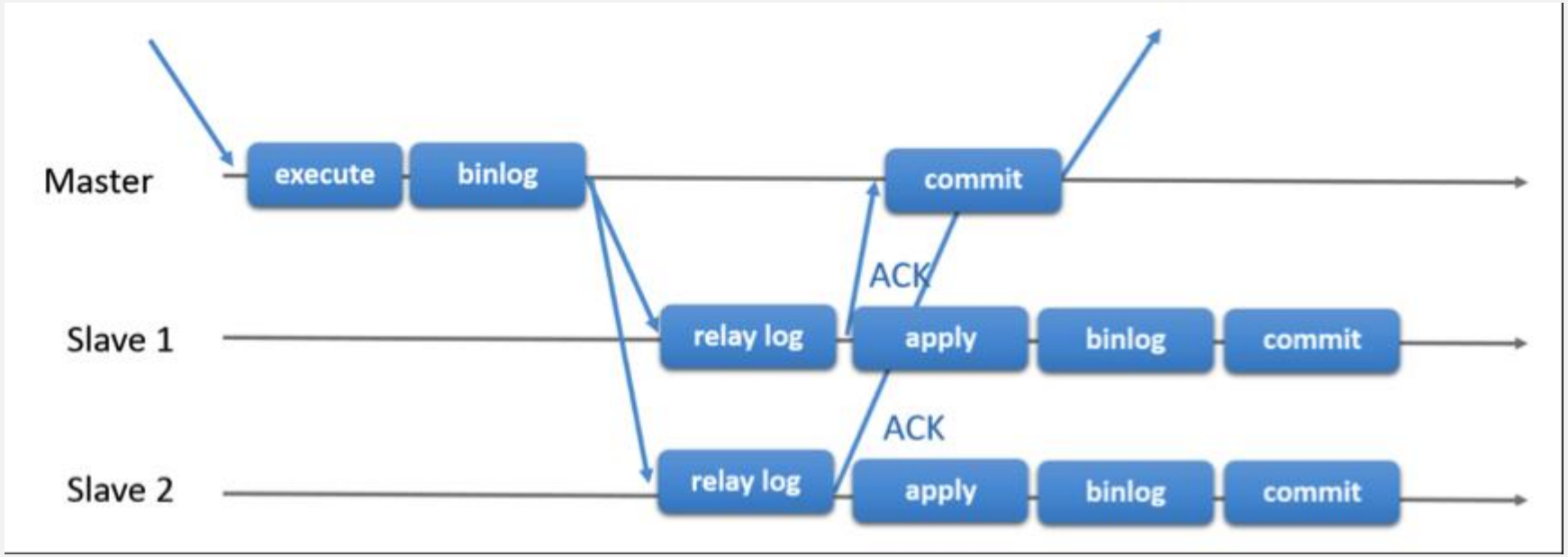
MySQL高可用

- 主备集群方案
  - 主机上的事务提交时，通过两阶段事务将binlog写入到磁盘，然后再将其发送给备机，备机收到后重放日志，以完成事务与主机的同步。



MySQL高可用

- Semi-sync
  - 主备之间会存在一些延迟，当主机已经将事务提交后，备机也许还没有收到这条事务的binlog，此时在备机上这条事务其实的缺失的。
  - 为了尽可能降低主备延迟，使用Semi-sync



MySQL半同步

- AFTER_COMMIT过程：
  - 写master binlog并commit
  - 同步binlog到slave并commit
  - slave返回acknowledgment给master
  - master接收到slave acknowledgment
  - master返回结果给client

- Master commit之后再将日志复制到slave。所有已经复制到slave的事务在master上一定commit了。所有master上commit的事务不一定复制到slave，比如，master commit之后，还没来得及将**日志复制到slave就宕机了，这时无法保证数据的一致性。**

MySQL半同步

- AFTER_SYNC过程：
  - 写master binglog
  - 同步主binlog到slave
  - slave返回acknowledgment给master
  - master接收到acknowledgment并commit
  - master返回结果给client

- 日志复制到slave之后，master再commit。所有在master上commit的事务都已经复制到了slave。所有已经复制到slave的事务在master不一定commit了，比如，master将日志复制到slave之后，master在commit之前宕机了，那么**slave有可能比master执行了更多事**务。

MySQL半同步

- 优点
  - 从库可以提供一定的读的能力，进行架构上的读写分离。
  - AFTER_SYNC模式下，可以保证数据不丢，主库宕机时从库可以升级为主库继续提供服务
- 缺点
  - 开启半同步复制时，Master在返回之前会等待Slave的响应或超时，当Slave超时时，半同步复制退化成异步复制。
  - 当Master宕机时，**数据一致性无法保证，依然存在从节点多执行或从节点少执行的情况，重启时可能需要人工干预**。
  - 网络质量要求高，每次事务处理都需要实时进行远程数据同步，对性能有一定影响

MySQL高可用

- Semi-sync存在的隐患
  - 但当系统意外down机，重新启动时，MySQL首先会进入故障恢复阶段，读取redo日志，做事务恢复，但可能会发现部分事务并没有提交，那么这部分事务是应该回滚吗？
  - MySQL的实现方法是先将这部分事务挂起，暂时既不提交也不回滚，然后读取binlog日志，如果在binlog中发现有此事务的记录，就将事务提交，若未发现此事务的记录，就将事务回滚
  - MySQL为什么这样决策？
    - 备机如果已经回放了binlog日志，确实应当提交
    - 备机如何没有接收到binlog日志，应该回滚
  - 场景：
    - 用户在原主机上插入10条数据，但在新主机上没有发现，因此重新插入，但在原主机成为备机后，却又接收到从主机传来的插入10条数据的binlog，此时何去何从？该不该插入呢？

第3点mysql决策后补：该逻辑是mysql 事务2阶段提交逻辑保证binlog和redo log一致性，与主备回放无关，主机redo log和bin log一致，即认为事务已经完成，只是未更新事务状态，参考：[mysql之 事务prepare 与 commit 阶段分析](#) 和 https://www.jianshu.com/p/0b2301d50351

MySQL高可用

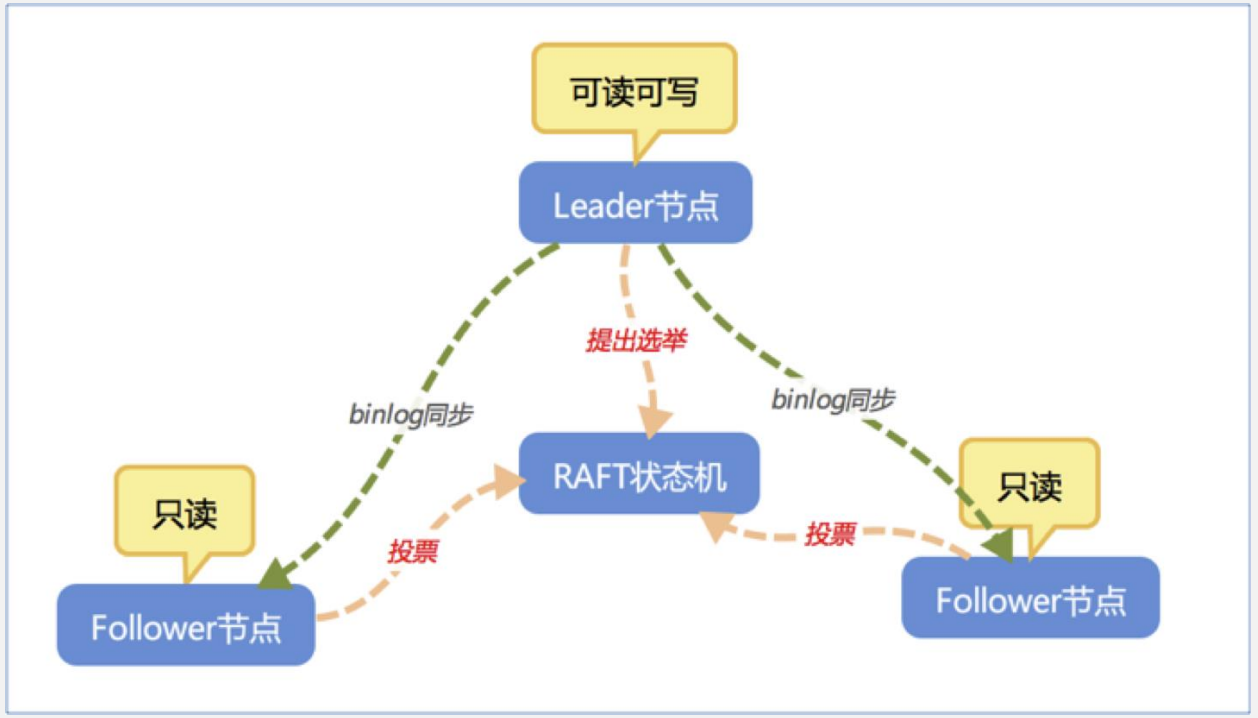
- 半同步复制数据一致性
  - http://mysql.taobao.org/monthly/2017/04/01/
- 异步复制
  - Binlog异步复制，备库落后，主库crash，丢数据
- 半同步复制
  - 主库在应答客户端提交的事务前需要保证至少一个从库接收并写到relay log中
  - 主库在等待备库ack时候，如果超时会退化为异步，（假设设置超时时间很长）
- rpl_semi_sync_master_wait_point
  - WAIT_AFTER_COMMIT：事务提交，虽然未响应客户端，但是其他客户端可读，主崩溃，产生幻读，client仅丢失一个事务，由于未响应客户端，不会开启下一个事务
  - WAIT_AFTER_SYNC：**从binlog同步ack之后，提交事务**

MySQL高可用

- sync_binlog
  - 等于0：binlog sync磁盘由操作系统负责
  - 不等于0：定期sync磁盘的binlog commit group数
  - 大于1：
    - sync binlog操作可能并没有使binlog落盘。如果没有落盘，事务在提交前，Master掉电，然后恢复，那么这个时候该事务被回滚。但是Slave上可能已经收到了该事务的events并且执行，这个时候就会出现Slave事务比Master多的情况，主备同步会失败。
- sync_relay_log
  - **sync_relay_log不是1的时候，semisync返回给Master的position可能没有sync到磁盘**？仅发生Master或Slave的一次Crash并不会发生数据丢失或者主备同步失败
  - 如果发生Slave没有sync relay log，Master端事务提交，客户端观察到事务提交，然后Slave端Crash。Slave端就会丢失掉已经回复Master ACK的事务events。
  - 但当Slave再次启动，如果没有来得及从Master端同步丢失的事务Events，Master就Crash。这个时候，用户访问Slave就会发现数据丢失。
- MySQL semisync如果要保证任意时刻发生一台机器宕机都不丢失数据，需要同时设置sync_relay_log为1。对relay log的sync操作是在queue_event中，对每个event都要sync，所以sync_relay_log设置为1的时候，事务响应时间会受到影响，对于涉及数据比较多的事务延迟会增加很多。

MySQL高可用

- Raft版MySQL集群
  - 主节点可以接受客户的读写事务
  - 备节点只能接受客户的读事务
  - binlog同步大多数节点后才能提交事务

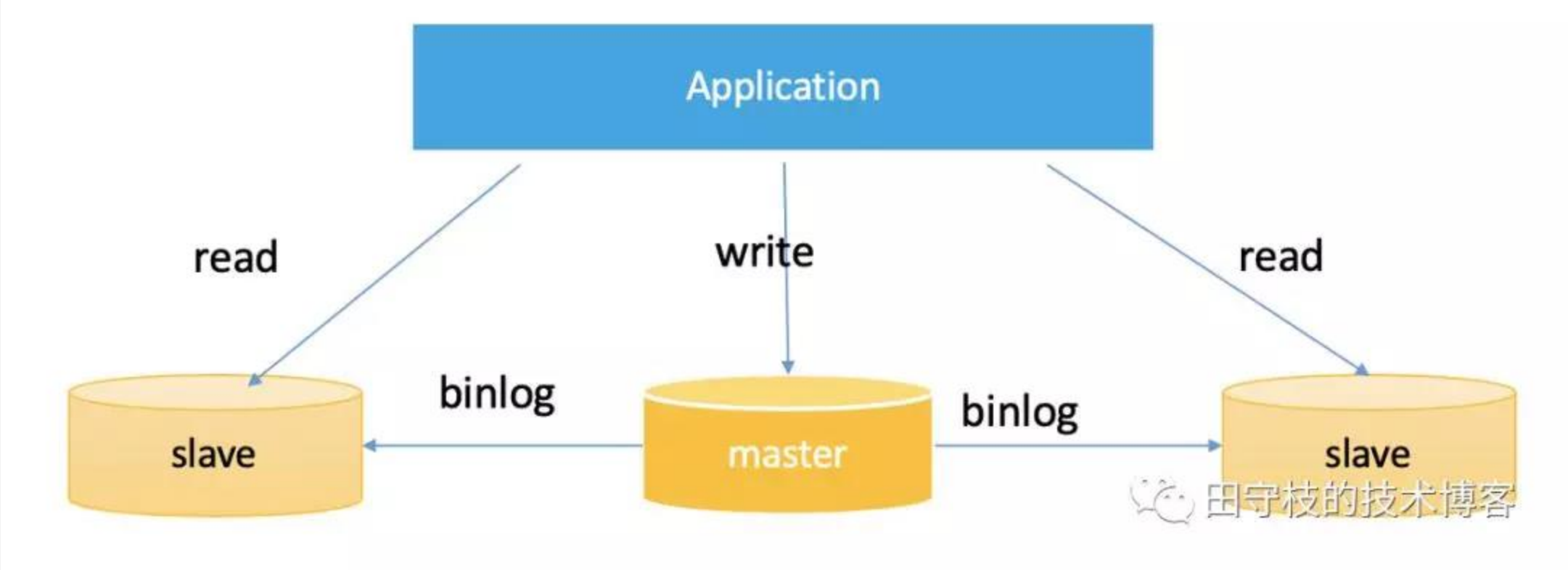


MySQL高可用

- Raft版MySQL集群
  - 识别故障（主机宕机or网络故障）
    - 主备集群依赖于第3方管理监控软件来识别故障和发起主备切换
    - Raft集群：可以多数派确认，选主和切换
  - 选主
    - 新主拥有更多的binlog
  - 在原主库加入集群对外服务之前，回滚那些未传送到其它节点的事务，从而确保其恢复到与新主库被选举出来的那个时刻

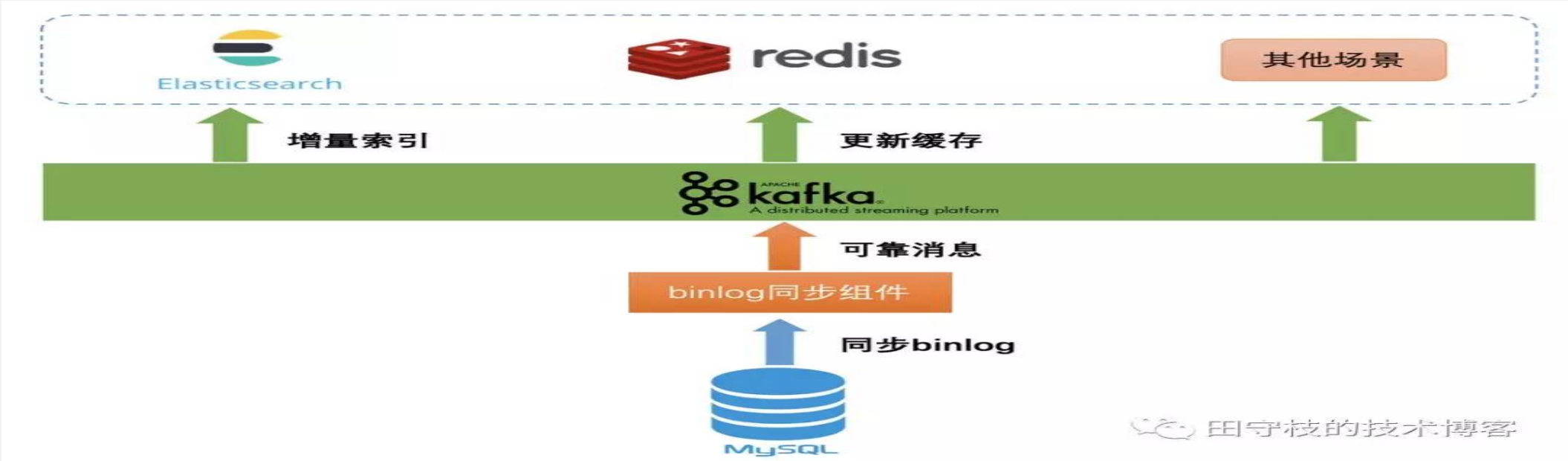
MySQL binlog的应用

- 读写分离，横向扩展
  - 一个主库Master，所有的更新操作都在master上进行
  - 多个Slave，每个Slave都连接到Master上，获取binlog在本地回放，实现数据复制



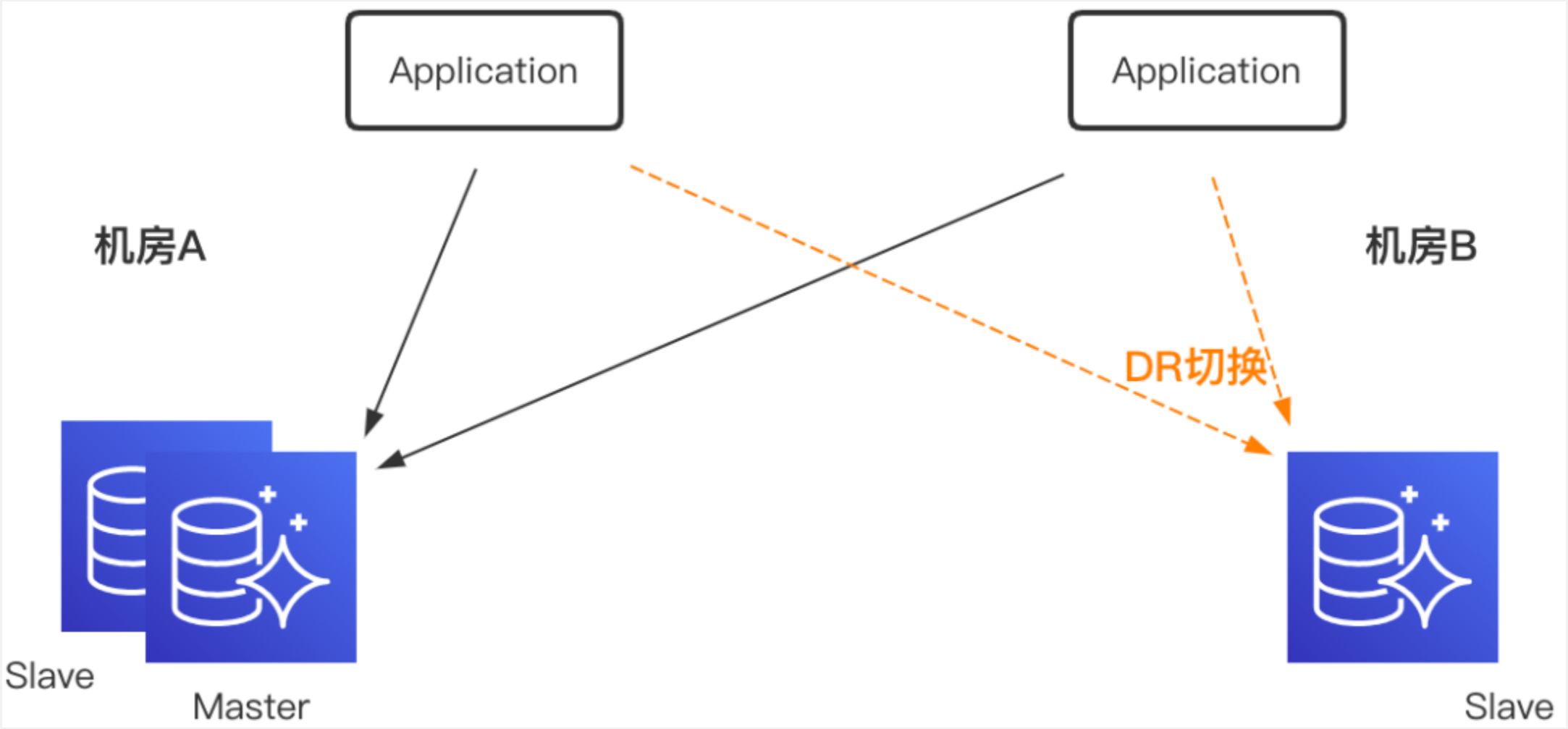
MySQL binlog的应用

- 数据恢复
  - 反解binlog，逆操作
- 数据最终一致性
  - 场景：应用更新多个组件时，消息队列、缓存、索引等的一致性保证
  - 通过解析binlog的信息，去异步的更新缓存、索引或者发送MQ消息，保证数据库与其他组件中数据的最终一致



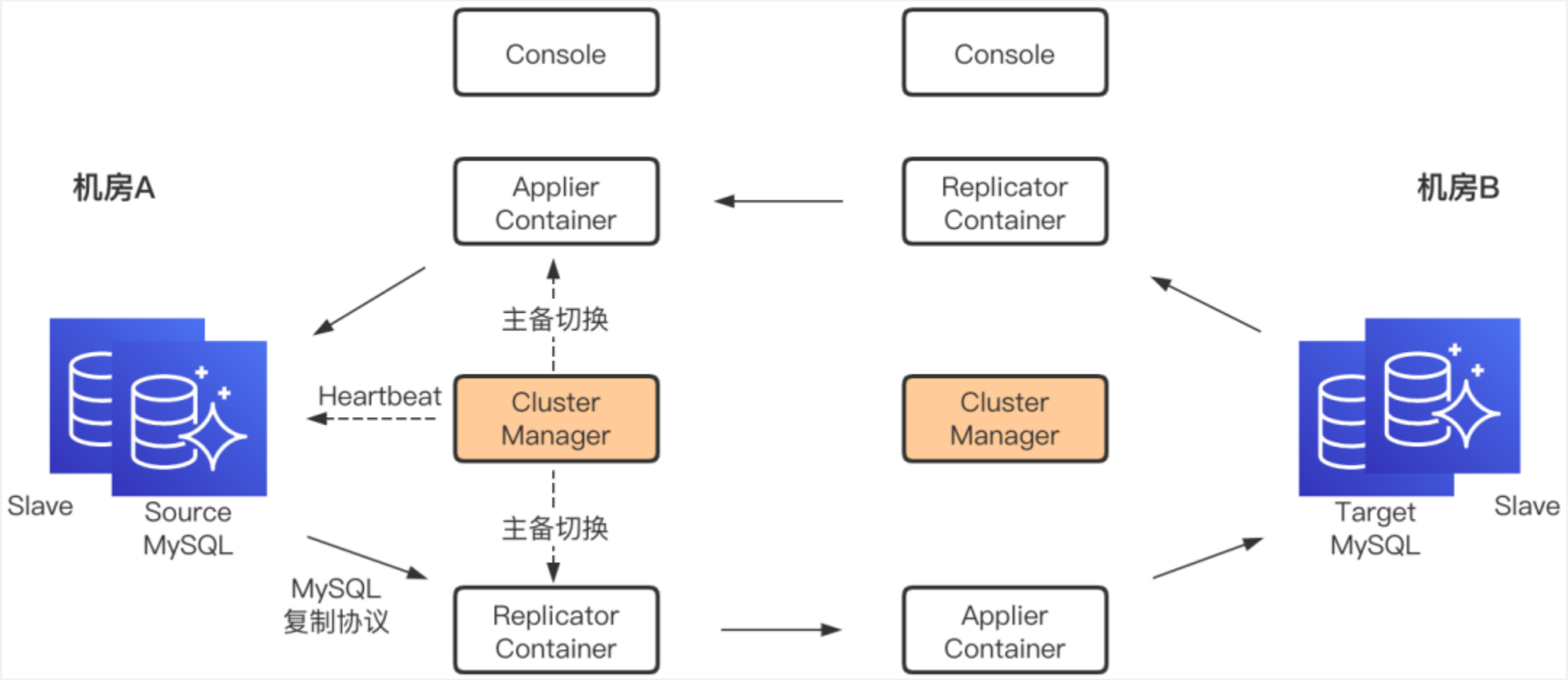
MySQL binlog的应用

- 异地多活（naïve，单向复制）



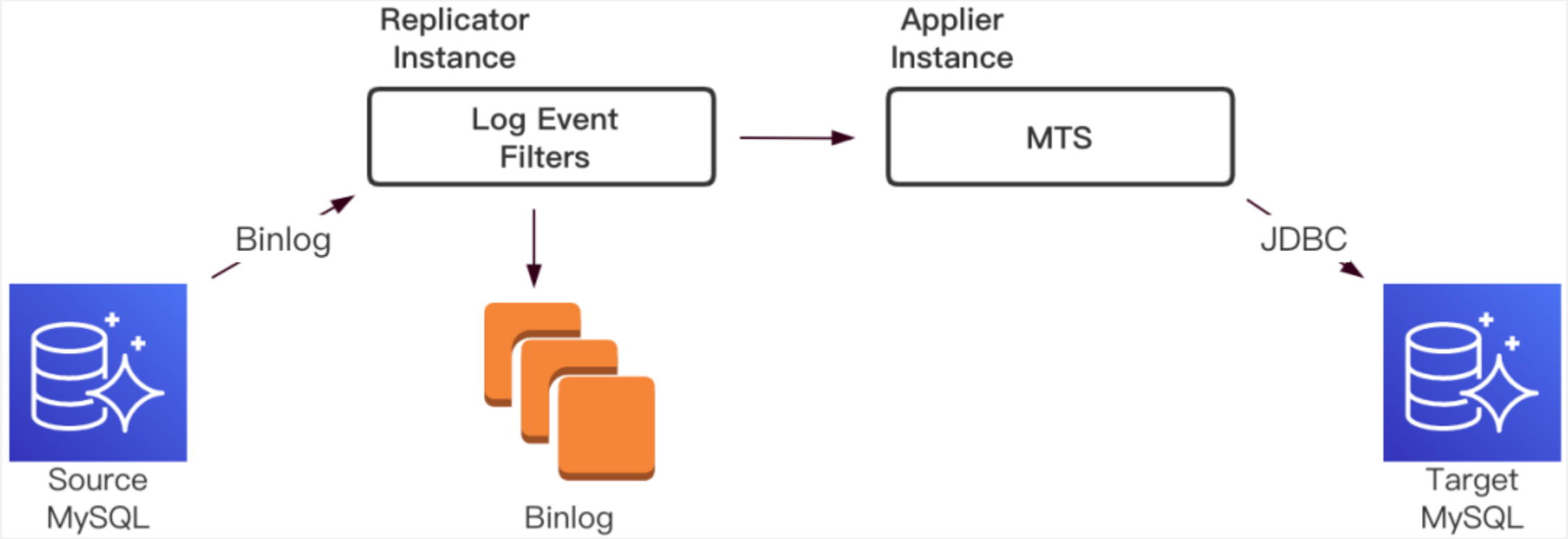
MySQL binlog的应用

- 异地多活（双向复制）



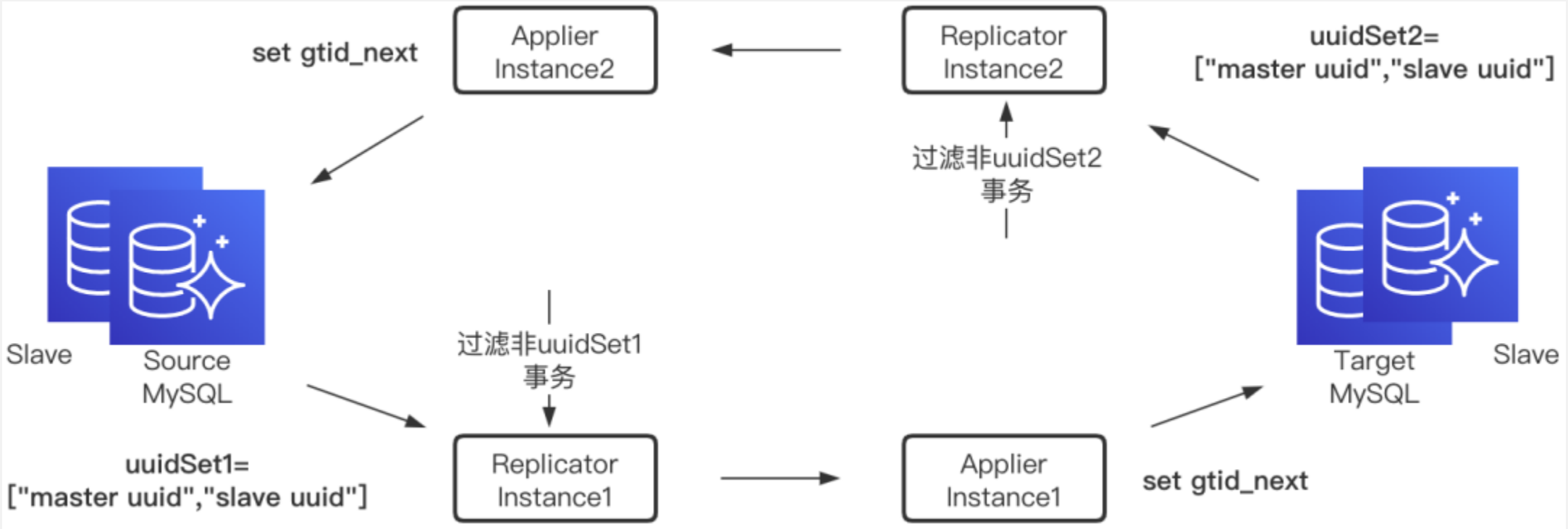
MySQL binlog的应用

- 异地多活



MySQL binlog的应用

- 异地多活一致性保证：GTID
  - 断点重续
  - 循环复制
  - 幂等



异地多活

- 1. 应用双活，数据库A地读写，B地不可读写。这种只有应用多活，数据库是**异地备份容灾**（无并发）。
- 2. 应用双活，数据库A地读写，B地只读。这种也是应用双活，数据库**读写分离**（实例级并发）。
- 3. 应用双活，数据库双活，两地应用同时读写不同表。这种数据库双向同步，应用同时错开写不同的数据（**表级并发**）。
- 4. 应用双活，数据库双活，两地应用同时读写相同表不同记录。这种数据库双向同步，应用同时错开写不同的数据（**行级并发**）。
  - （应用层流量拆分）
- 5. 应用双活，数据库双活，两地应用同时读写相同表相同记录。这种数据库双向同步，应用同时写相同的数据，最终会因为冲突一方事务回滚（**行级并发写冲突**）

参考

- 阿里数据库内核月报http://mysql.taobao.org/monthly/
- PostgreSQL9.6.0备份和恢复 http://www.postgres.cn/docs/9.6/backup.html
- mysql binlog应用场景与原理深度剖析 http://www.jiangxinlingdu.com/mysql/2019/06/07/binlog.html
- 携程异地多活 MySQL实时双向（多向）复制实践 https://www.infoq.cn/article/yvG7ytTjTfXCeUO9X1w9
- 数据库的异地多活分析和方案 https://cloud.tencent.com/developer/article/1421904
- 深入MySQL复制(三)：半同步复制 https://www.cnblogs.com/f-ck-need-u/p/9166452.html

- 数据密集型应用设计第5章，数据复制
- 高可用mysql第2版