



PDB数据同步与数据库设计学习分享

2016.07.14

涂仲秋

目录



- PDB(Paxos Data Base)模块概述
- PDB模块架构设计和系统组成
- PDB模块数据同步过程
- PDB模块数据库设计

PDB(Paxos Data Base)模块概述



• 系统功能

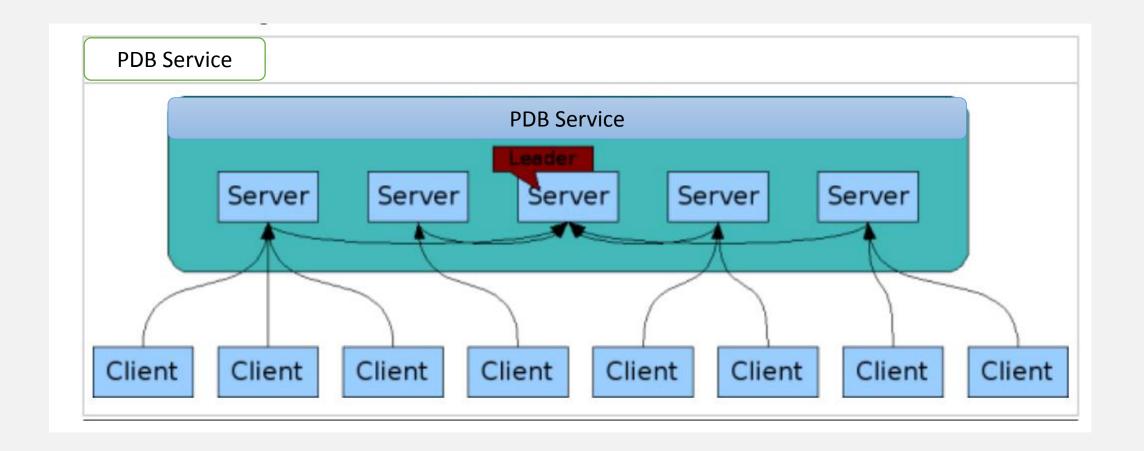
- PDB是一个分布式数据库,由一个集群组成,集群中的每个节点保存数据库的一个完整副本,为分布式应用提供协调服务(Coordination Service):
 - 配置信息管理和维护(maintain configuration information)
 - 进程运行信息管理和维护
 - 组管理
 - 实现同步原语(synchronization primitive)
 - 节点失效恢复管理

• 系统性能

- 高可用(High Availability)
- 高可靠(High Reliability)
- 高吞吐率
- 读操作高性能
- 容灾容错(2n+1个节点的集群,最多可容忍n个节点失效,集群节点数目最好为 奇数个)

PDB模块架构设计和系统组成







- Leader: 系统中的主进程(Primary Process),通过选举算法选出,除选举阶段外,系统中任何时刻有且仅有一个Leader节点。处理来自客户端的写请求,主导集群的数据同步。
- Follower: 系统集群中除Leader外的任何节点都是Follower节点,跟随 Leader节点做数据同步,与Leader的数据库保持一致。
- Client:连接到集群的客户端,可以与集群中任意一个节点建立连接,向系统发出读写请求。

PDB系统核心概念



• 事务(Transaction)

- 是数据库管理系统执行过程中的一个逻辑单位,由一个有限的数据库操作序列构成。
- 客户端对PDB数据库的一个写操作,表示数据库的状态要发生变化,由主进程(Leader)产生, 并广播到Follower节点。系统每收到一个写请求,就会产生一个事务。

• 事务ID (Transaction ID)

• 每个事务的全局唯一编号,由Leader节点产生,用以区分各个不同事务,事务ID以递增的方式生成,用以标识各个事务产生的先后顺序,事务ID越大,表示事务越新。

事务日志(Transaction Log)

• 记录每个事务内容、事务编号等信息的日志,Leader产生一个事务后会将事务日志持久化记录到本地,用于数据同步。

Epoch

• 字面意思为时期、时代、纪元。用以表示某个Leader节点管理集群的时期,每次发生Leader 选举后,新选举出的Leader会增加Epoch,用以区分不同Leader管理集群的时期。



Quorum

• 表示集群中一个由大多数Follower节点所组成的集合,节点数目多于总结点数目的一半,在 Leader选举以及Leader与Follower之间数据同步时,都需要经过一个Quorum中所有节点的通 过才能进行下一步操作。

Proposal

• 提案,当Leader节点收到一个写请求时,先生成一个事务,然后会产生一个提案,将这个提案广播给所有Follower节点,等待Follower节点的ACK回应。提案包括这个事务的所有信息。

Commit

• 往本地数据库里提交请求,修改数据库中的数据。当Leader发送的Proposal消息得到大多数 Follower返回的ACK信息之后,Leader就向所有Follower节点发送Commit消息,要求各个 Follower节点修改本地数据库,与Leader节点的数据库保持一致。

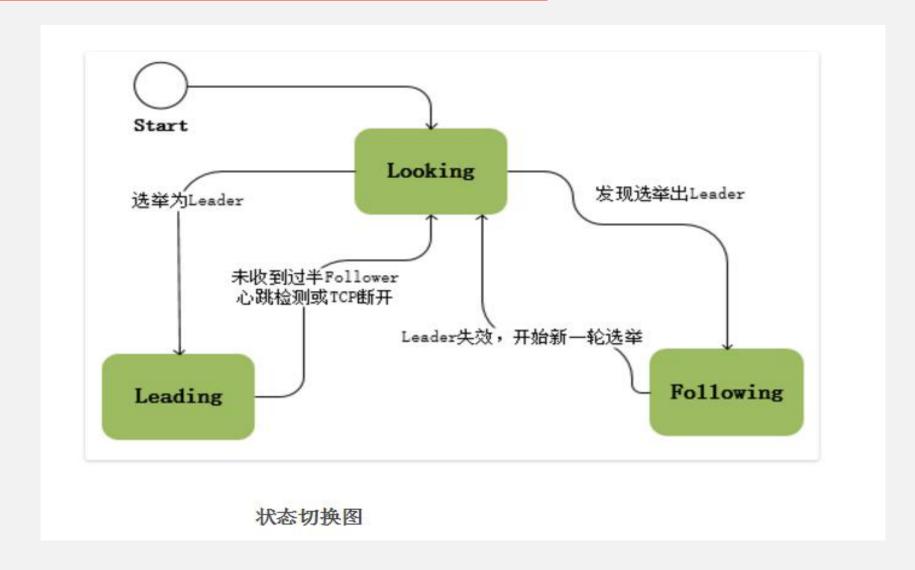
PDB系统中节点状态切换策略



- PDB系统中的节点共有3种状态:
 - Looking:表示系统正在进行Leader选举,发生在系统刚启动或老Leader崩溃选举新 Leader过程中;
 - Following:表示Follower节点所处的状态,Follower与Leader处于数据同步状态;
 - Leading: 表示Leader节点所处的状态,当前集群中有且仅有一个Leader为主进程;

节点状态切换图



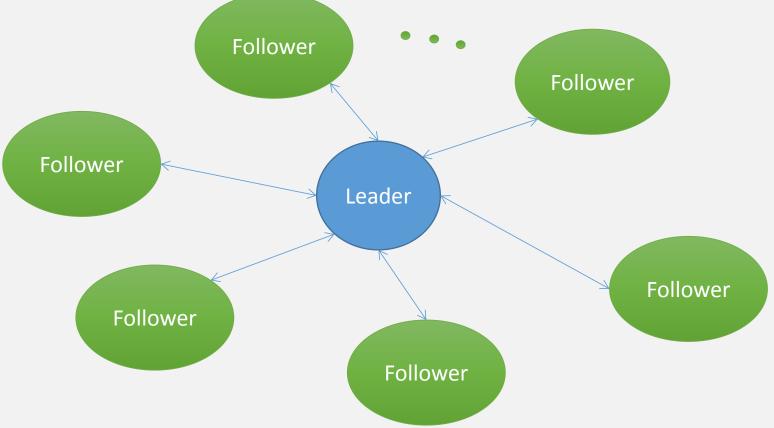




• 网络拓扑结构

• 在Leader选举完成之后,Leader与所有Follower建立连接,系统网络呈星型网络拓

扑结构



PDB集群数据同步过程



- 定义
 - 集群中的Follower节点同步Leader节点的数据库,与Leader节点的数据保持一致。同步方向严格限制为 Leader到Follower。
- 目的
 - 使集群中所有节点维护的本地数据库保持一致,对外提供高可用、容灾、容错的服务。
- 同步方式
 - 事务日志
 - ①如果Leader节点的事务日志更新,则根据Leader本地保存的事务日志同步Follower的数据库; (DIFF)
 - ②如果Follower节点的事务日志比Leader节点的日志更新,则Follower节点需要截断本地保存的事务日志,对本地数据库的操作做回滚; (TRUNC)
 - 快照
 - 如果Follower节点本地保存的事务日志太旧,则直接用Leader节点的数据库快照同步数据库; (SNAP)



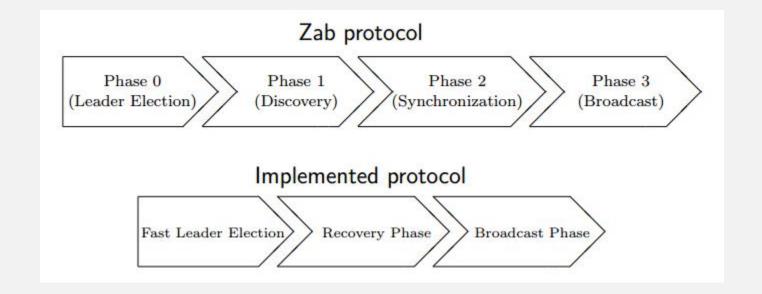
• 数据同步分四个阶段:

• PhaseO: Leader选举

• Phasel: 发现(Discovery)

• Phase2: 数据同步(Synchronization)

• Phase3: 广播(Broadcast)



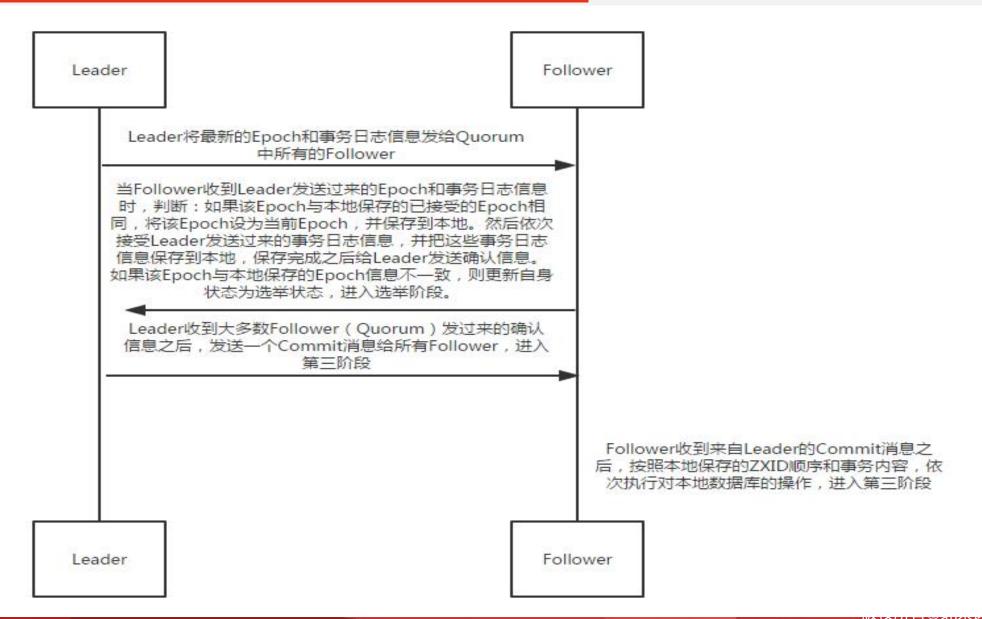
第一阶段(Discovery)交互时序图



Follower Leader 所有Follower将当前已知的Epoch发给新选举出的 Leader Leader收到大多数Follower (Quorum)发过来的 Epoch信息之后,选择一个新的Epoch大于所有这些收 到的Epoch, 然后将最新的Epoch发送给这Quorum个 Follower Follower收到Leader发送过来的最新Epoch信息之后,判 断:如果该Epoch比本地保存的最新Epoch大,将本地保 存的最新Epoch更新为该Epoch, 然后给Leader发送 Epoch确认信息,并包括最新Epoch、事务日志历史信 息、最大ZXID,进入第二阶段 Leader收到Quorum个Follower返回的 Epoch确认信息之后,找到所有这些 Follower中ZXID最大的一个,将这个 Follower中保存的历史事务日志信息作为 自己的事务日志信息,保存到本地,进入 第二阶段。 Leader Follower

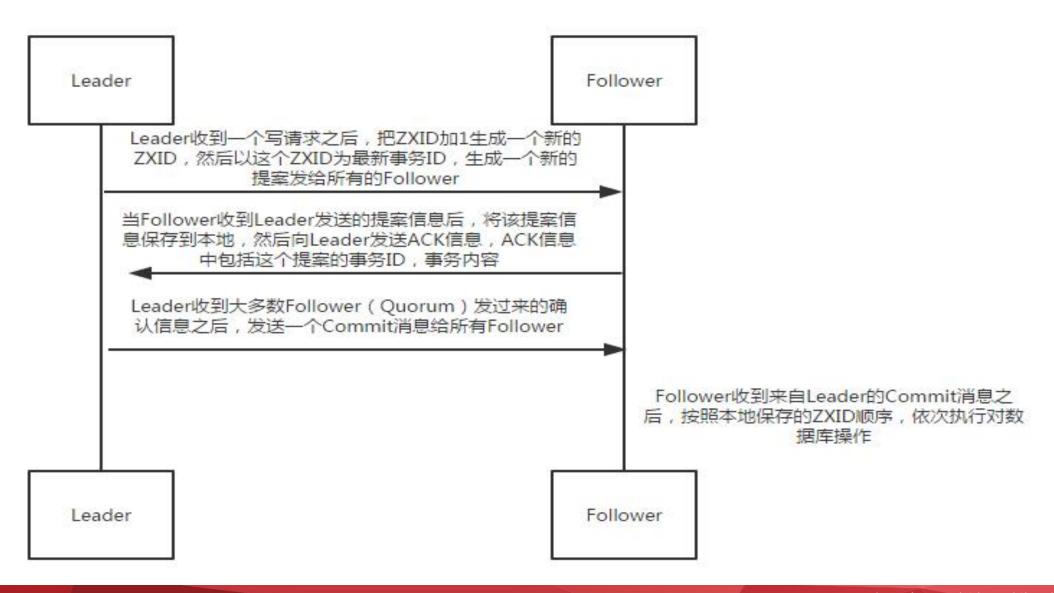
第二阶段(synchronization)交互时序图





第三阶段(Broadcast)交互时序图





第四阶段交互时序图(与选举算法FLE强相关)



Leader

Follower

当新加入一个Follower,与Leader建立连接之后,将本地保存的最大ZXID发送给Leader

当Leader收到来自于Follower的ZXID信息之后,将自己本地保存的最大ZXID发送给这个Follower,然后比较Follower本地最大ZXID与Leader上次提交的ZXID:

- 1、如果Follower的ZXID太小(依据门限),则给该 Follower发送一个数据库的快照(SNAP)
- 2、如果Follower的ZXID小于Leader上次提交的ZXID,则发送Follower缺少的事务日志信息(DIFF)
- 3、如果Follower本地最大ZXID大于Leader上次提交ZXID, 发送一个截断信息给Follower,要求Follower将超过Leader 上次提交ZXID之外的日志信息截断(TRUNC)

当Follower收到来自Leader的最新ZXID时,取出ZXID中的Epoch信息,如果该Epoch小于本地保存的最新Epoch信息,修改自身状态为选举状态,进入选举阶段。如果该Epoch合法,检查所收到的数据同步信息:

- 1、如果收到快照信息(SNAP),就使用该快照来同步本地数据库;
- 2、如果收到DIFF信息,则依据收到的事务日志信息同步本 地数据库:
- 3、如果收到TRUNC信息,就按照Leader发送所来的上次提交ZXID信息截断本地数据库;

最后,给Leader发送ACK信息,回到阶段3。

Leader收到Quorum个Follower返回的 ACK信息之后,也回到阶段3.

Leader

Follower

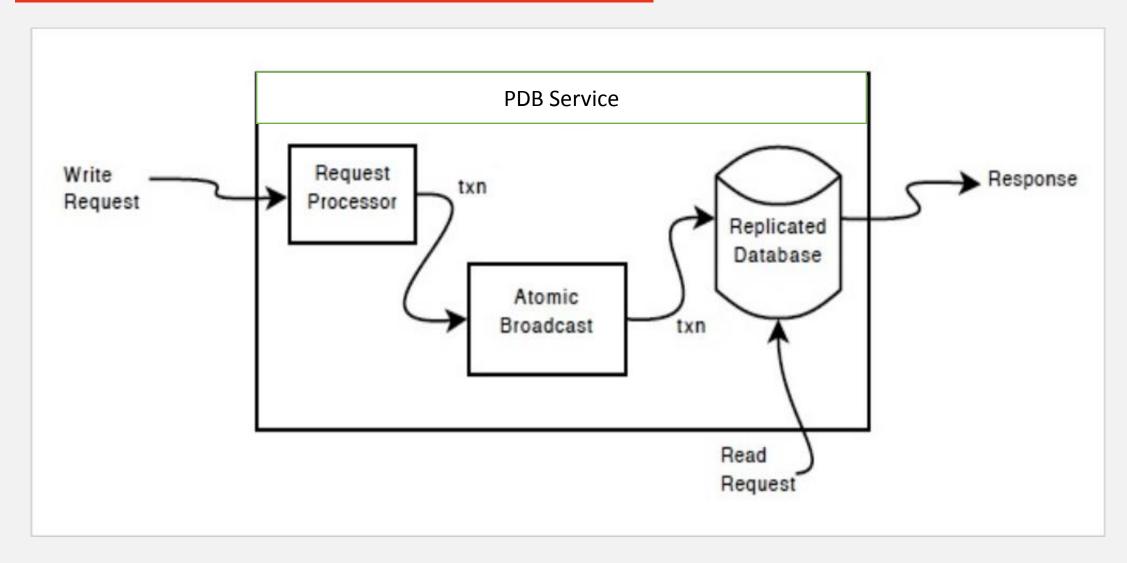
读写请求响应方式



- 读请求
 - 读请求直接有接收请求的节点响应,无论该节点是Leader还是Follower;
- 写请求
 - 写请求涉及到对数据库的修改和集群数据同步的过程,只能由Leader节点来处理。如果接收到写请求的是Follower节点,则该节点必须将该请求转发给Leader节点来处理。多个写请求按照到来的先后顺序放入队列(FIFO),依次处理。

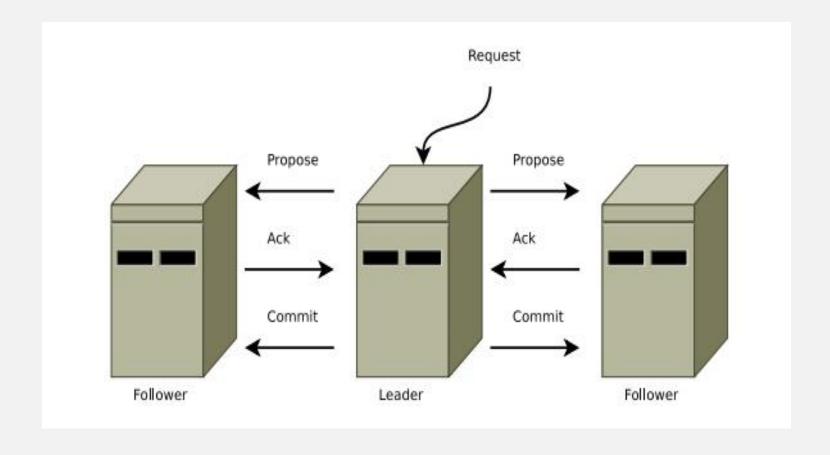
读写请求响应示意图





写请求处理流程







• 关键数据结构设计

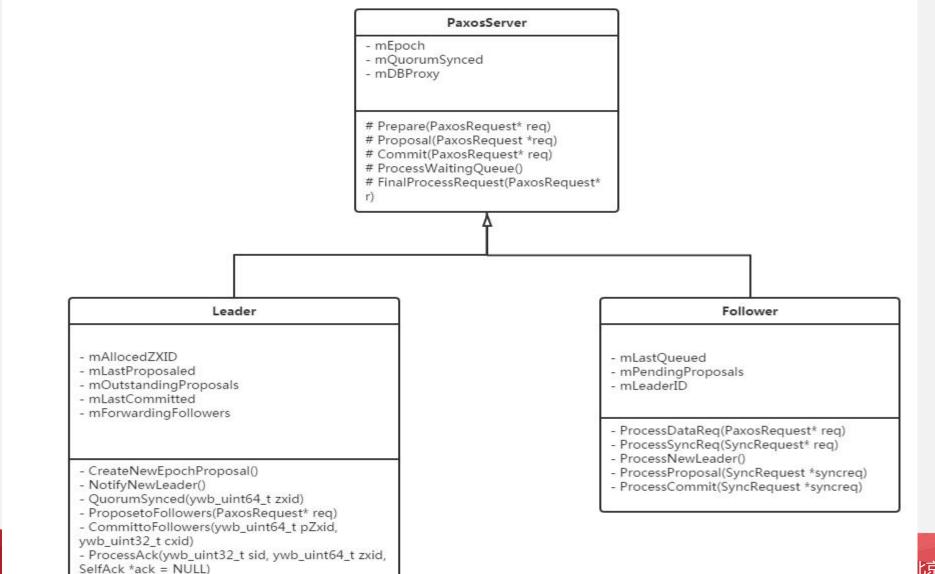
- epoch ywb_uint32_t 无符号32位整型变量,从0开始递增,每次发生Leader选举之后,新 Leader会把上次的epoch加一生成新的epoch;
- ZXID (ZooKeeper Transaction ID) ywb_uint64_t 无符号64位整型变量,表示事务ID,高32 位标识epoch,低32位是一个计数器,表示在这个epoch内的事务编号,计数器的初始值也设置为0,每次生成新的事务ID时把计数器的值加一,发生Leader选举后,新Leader把计数器清零重新开始计数;
- sid(server ID) yfs_sid_t server ID,用以标识集群中各个节点的ID,每个节点拥有全局唯一的server ID,将所有节点的server ID 保存在一个std::set容器中,在Leader节点广播请求时依据sid 对集群中的所有节点发送消息;
- t_PaxosReqQue mWaitingQueue 请求等待队列,当Leader收到写请求后,就把这个请求放入这个等待队列,然后按照事务ID依次处理所有收到的请求;

具体代码实现介绍

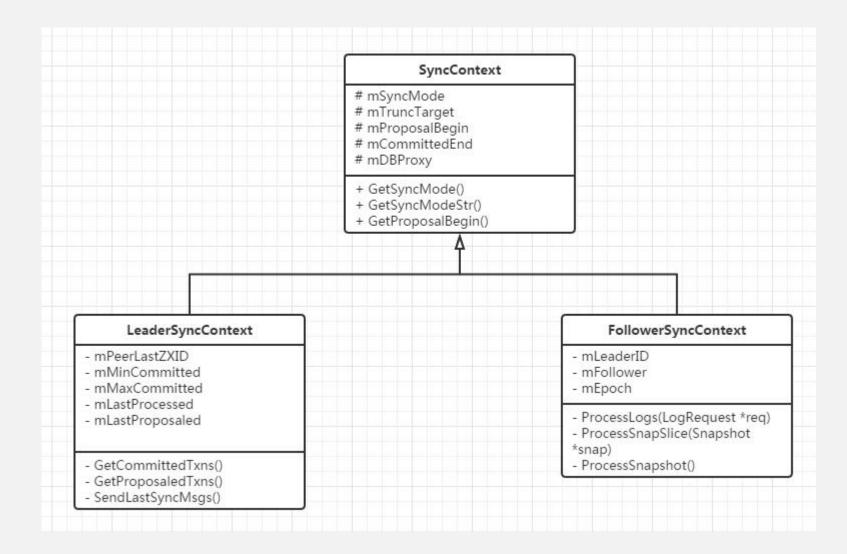


• 关键数据结构(类)设计

- BroadcastPacket(NMsqblock* block)







PDB数据库设计与实现



- Shard数据库概念
 - PDB是一个分布式数据库,每个节点保持着数据库的一个完整副本,通过数据多副本 冗余来达到高可用、高可靠、容灾、容错。在PDB中,每个节点维护的数据库副本称 为shard。
- 两种类型
 - Key/Value类型数据库
 - SQL类型数据库(使用SQLite实现)

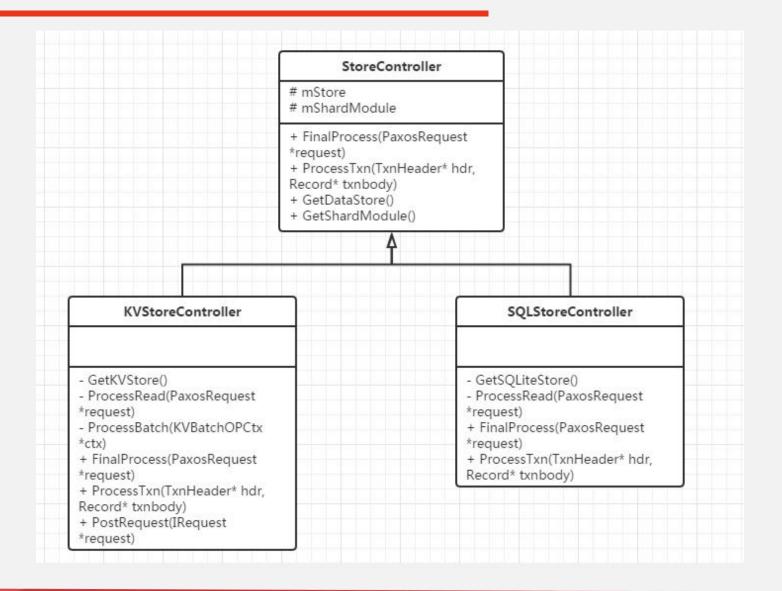
t KVResult& items)



StoreModel # mStoreCtrl # mSnapMgr + TakeSnapshot(ywb_uint64_t lastprocessed) + FinishSnapshot(SnapRequest* snapred + GetLatestSnapshot(ywb uint64 t lastprocessed, Snapshot **snap) + SerializeToSnapshot(Snapshot + DeserializeFromSnapshot(Snapshot *snap) **KVStore** SQLiteStore - mKVTables - mDataDir - mDeletedTables mSQLiteDB - mFreezingZXID - mFrozenTableCount + TakeSnapshot(ywb_uint64_t lastprocessed) - GetTable(const std::string& tablename) + FinishSnapshot(SnapRequest* snapreq) RemoveTable(const std::string& tablename) + SerializeToSnapshot(Snapshot *snap) - AllocTable(const std::string& name) + DeserializeFromSnapshot(Snapshot - Serialize(std::ostream &fout) *snap) - Deserialize(ywb_uint64_t fileid, std::istream &fin) + GetLatestSnapshot(ywb uint64 t + TakeSnapshot(ywb_uint64_t lastprocessed) lastprocessed, Snapshot **snap) + FinishSnapshot(SnapRequest* snapreg) + ExcuteSQL(const string& sqlstr, + SerializeToSnapshot(Snapshot *snap) ywb_bool_t istransaction,ywb_uint64_t zxid) + DeserializeFromSnapshot(Snapshot *snap) + ExcuteQuery(const string& salstr. + CreateTable(const std::string& name) t_SQLTable& result) + DeleteTable(const std::string& name, ywb_uint32_t flags) + Put(const std::string& tablename, const std::string& key, DataEntry* value, ywb uint32 t flags) + Del(const std::string& tablename, const std::string& key) + Get(const std::string& tablename, const std::string& key, string& value) + List(const std::string& tablename,

存储控制器





KV数据库分类



KVTable

- # mTableName
- # mFreezingZXID
- # mCapacity
- + GetTableName()
- + GetCapacity()
- + Create()
- + Delete()
- + Put(const std::string& key, DataEntry *data, ywb_uint32_t flags)
- + Get(const std::string& key)
- + Del(const std::string& key)
- + List(const vector<std::string>& keys,
- t KVResult& items)
- + List(t_KVResult& items)
- + Serialize(std::ostream &fout)
- + Deserialize(ywb_uint64_t fileid, std::istream &fin)

KVTable Persist

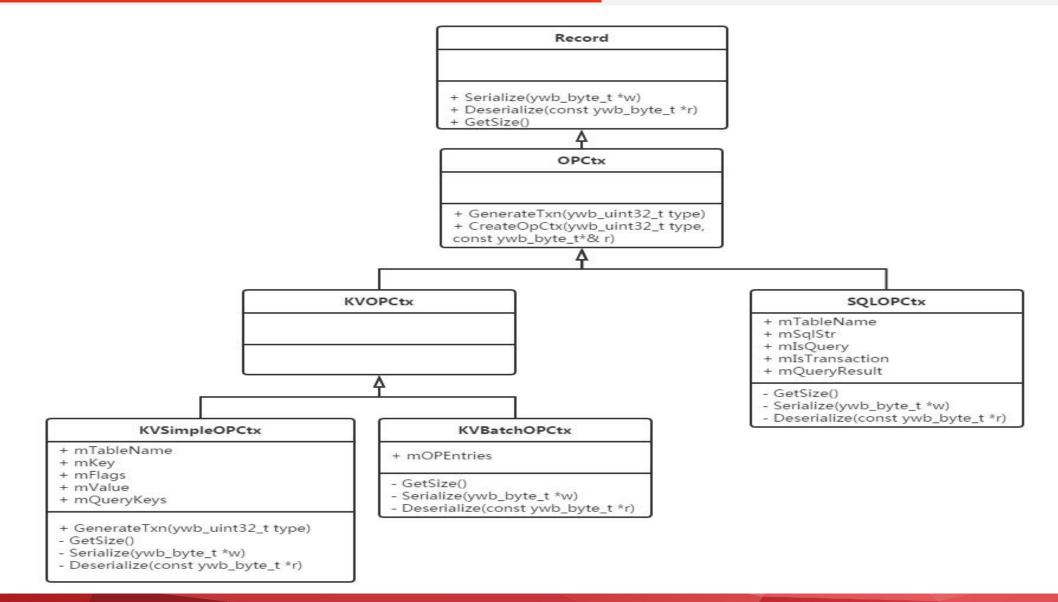
- mKValues
- mIsRecreated
- mIsRecreated
- mDeletedKeys
- + Create()
- + Delete()
- + Put(const std::string& key, DataEntry *data, ywb_uint32_t flags)
- + Get(const std::string& key)
- + List(const vector<std::string>& keys, t_KVResult& items)
- + List(t_KVResult& items)
- + Del(const std::string& key)

KVTable_SkipList

- mKValues
- mIsRecreated
- mTempKValues
- mDeletedKeys
- + Create()
- + Delete()
- + Put(const std::string& key, DataEntry *data, ywb_uint32_t flags)
- + Get(const std::string& key)
- + List(const vector<std::string>& keys, t_KVResult& items)
- + List(t_KVResult& items)
- + Del(const std::string& key)

数据库操作类图





Thanks!

北京市

北京市朝阳区安定路一号国家奥林匹克体育中心体

育场东南门2层2160室

邮编:100029

北京总部

电话: +8610 8843 7583

传真:+8610 8437 6490

武汉分公司

武汉市洪山区珞瑜路33号中部创意大厦1907、1908

邮编:430074

电话: +8627 8786 2881

传真: +8627 8786 2567

联系我们