数据与存储

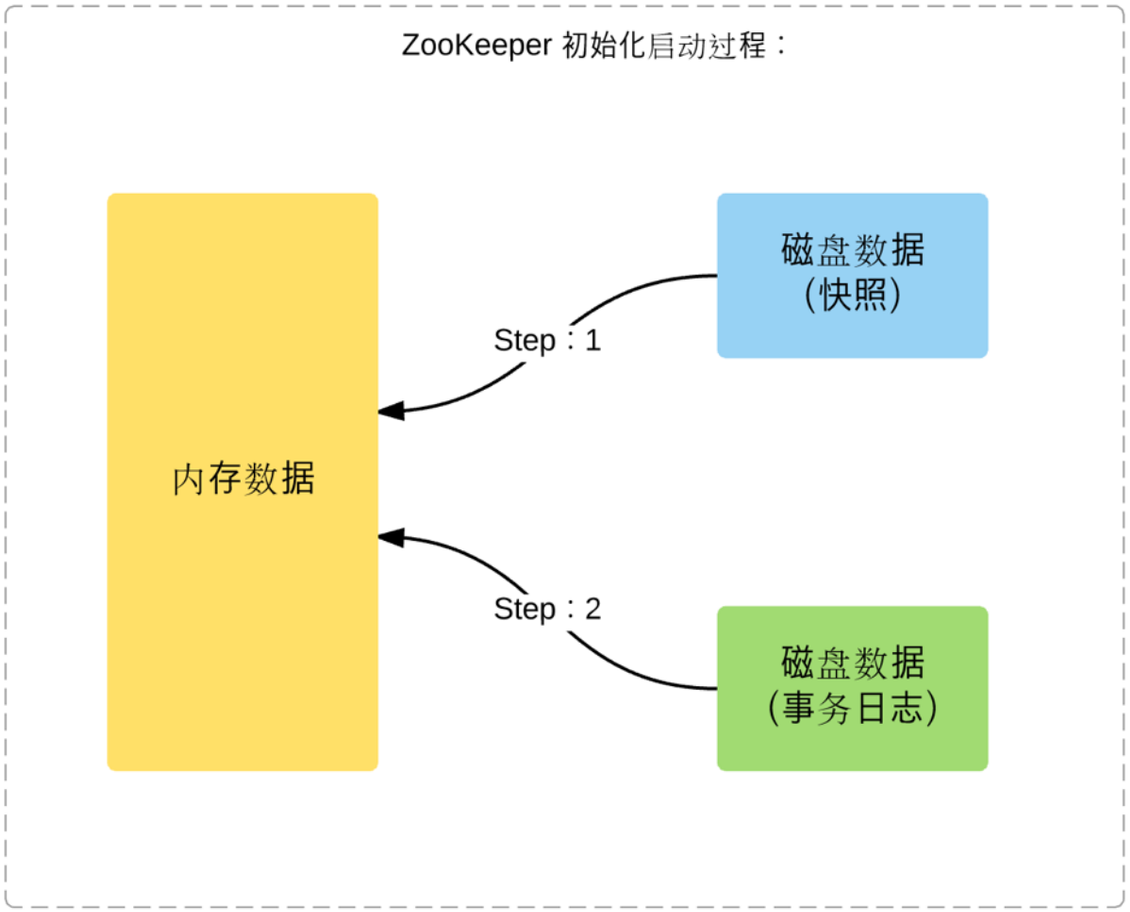
—— ZooKeeper 技术内幕

@NingG http://ningg.top/

2016/05/12

目录

- 数据分类：
  - 内存数据
  - 磁盘数据：
    - 快照
    - 事务日志
- 数据存储的相关过程：
  - 初始化：ZK 服务器启动
  - 数据同步：leader 与 learner



内存数据

- 关键点：
  - ZK 的数据模型：树
  - 树，包含：
    - 节点数据
    - 节点 ACL 信息
    - 节点的路径
- 具体实现：DataTree 和 DataNode

内存数据


DataTree
– nodes： ConcurrentHashMap<String, DataNode>
– ephemerals：ConcurrentHashMap<Long, HashSet<String>>
– dataWatches：WatchManager
– childWatches： WatchManager
1
n
DataNode
– parent： DataNode
– children： Set<String>
– data： byte[]
– acl： long
– stat： StatPersisted
关键点：
1. DataTree，内存数据存储的核心，代表完整的数据
2. 不包含：任何与网络、客户端连接、请求处理的业务逻辑
3. 单独保存一份临时节点，方便实时访问和清理

快照数据

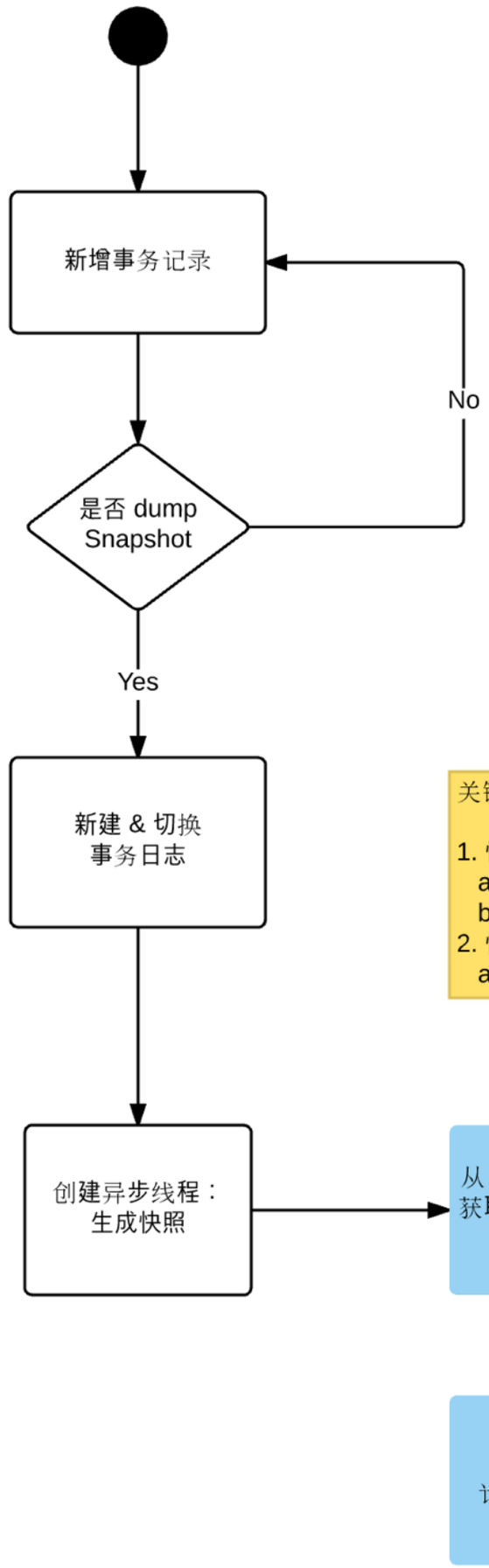
- 简介：
  - 某一时刻，全量的内存数据，存储到磁盘中，形成快照（Snapshot）
- 如何确定时刻？
  - ZXID，事务 ID，标记在快照文件名中
  - ZXID，最后一次成功执行的事务 ID
  - logCount > snapCount： 累计新增的事务记录数量，超过阈值
- 关键点：
  - 快照文件大小，能够反应当时内存中全量数据的大小
  - 快照数据，没有采用「预分配」策略，没有填充 0

生成快照的触发条件：

logCount > (snapCount/2 + randRoll)

注：

1. logCount：新增事务日志数量
2. snapCount：配置的阈值，用于触发 dump，默认 10w
3. randRoll：1～snapCount/2 之间的随机数
4. 添加随机因素，避免所有 ZK 节点在同一时刻触发 dump



关键点：

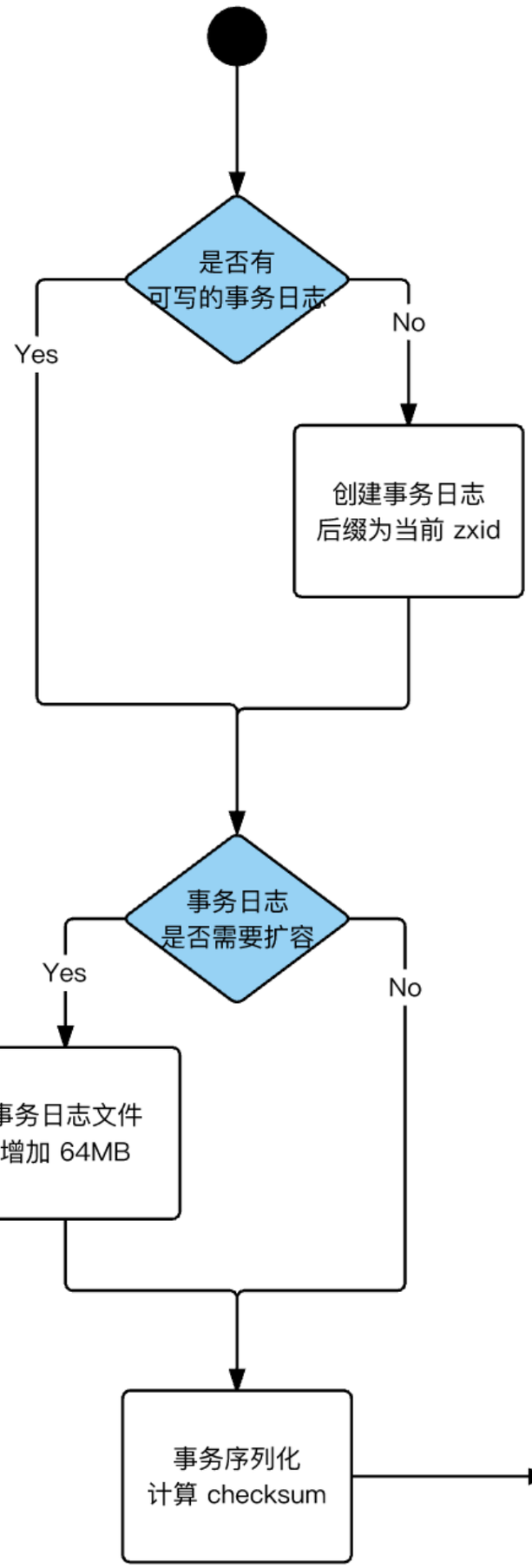
1. 快照中包含的内容：
   a. 所有数据节点信息（DataTree）
   b. 会话信息
2. 快照文件名：
   a. 包含最后一次执行成功的事务id， zxid

快照数据

- 关键点：
  - 异步：异步线程，生成快照文件
  - Fuzzy 快照：
    - 快照文件生成过程中，仍然有新的事务提交
    - 快照文件，不是精确到某一时刻的快照文件，而是模糊的
    - 要求事务操作是幂等的，否则产生不一致
- 疑问：
  - 是否每次生成快照文件，都会认为「事务日志已经写满」，并切换一次事务日志文件?
  - 切换事务日志文件的时机，实际是生成快照的时机。

事务日志

- 简介：
  - 事务日志文件：大小一致，64M 的倍数
  - 事务日志文件命名：log.***，后缀是 zxid，当前的zxid
    - zxid 中包含 Leader 周期（epoch），高 32 位
- 关键点：
  - 事务日志，频繁 flush 到磁盘，消耗大量磁盘 IO
  - 磁盘空间**预分配**：事务日志剩余空间 < 4KB 时，将文件大小增加 64 MB
  - 磁盘预分配的目标：减少磁盘 seek 次数
  - 建议：事务日志，采用独立磁盘单独存放



缺少可写事务日志的情况：

1. 第一次写入事务日志
2. 上一个事务日志写满

需要扩容的时机：

1. 事务日志文件，剩余大小 < 4KB
2. 文件大小增加 64MB，使用 \0 填充

事务日志

- 日志截断：
  - 现象：Learner 的机器上记录的 zxid 比 Leader 机器上的 zxid 大，这是非法状态；
  - 原则：只要集群中存在 Leader，所有机器都必须与 Leader 的数据保持同步
  - 处理细节：遇到非法状态，Leader 发送 TRUNC 命令给特定机器，要求进行日志截断，Learner 机器收到命令，会删除非法的事务日志

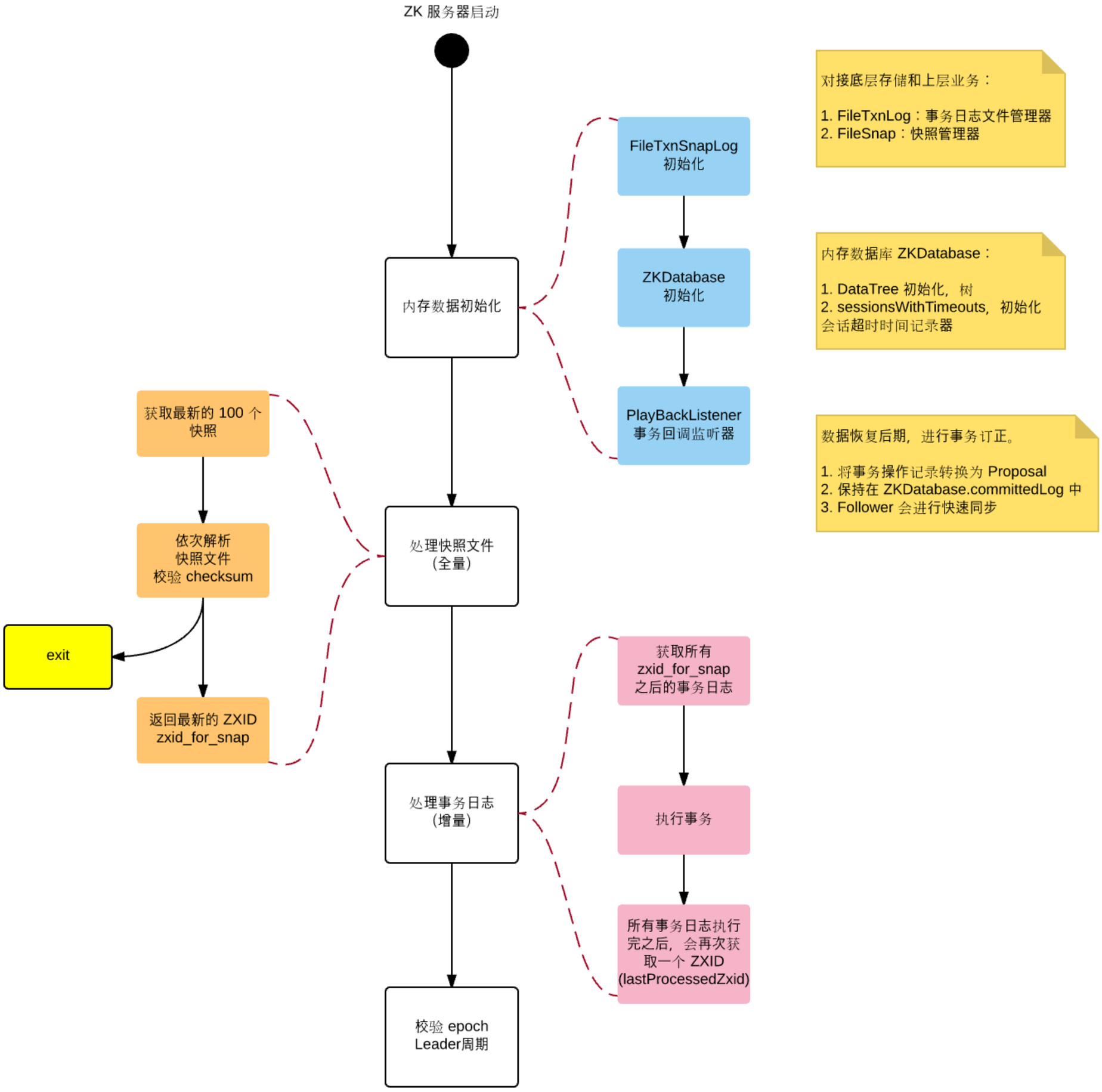
小结

- 上一部分：ZK 中数据和存储：
  - 内存数据：DataTree，数据节点的存储、临时节点的存储
  - 磁盘数据：
    - 快照：生成的时机
    - 事务日志：日志截断

- 下一步：数据存储相关过程：
  - 初始化：ZK 服务器启动过程
  - 数据同步：ZK 服务器启动后，Leader 与 Learner 之间数据同步

初始化

- ZK 服务器初始化的目标：
- ZK 服务器启动时，首先会进行数据初始化，将磁盘中数据，加载到内存中，恢复现场。


ZK 服务器启动
内存数据初始化
FileTxnSnapLog
初始化
ZKDatabase
初始化
PlayBackListener
事务回调监听器
对接底层存储和上层业务：
1. FileTxnLog：事务日志文件管理器
2. FileSnap：快照管理器
内存数据库 ZKDatabase：
1. DataTree 初始化，树
2. sessionsWithTimeouts，初始化
会话超时时间记录器
数据恢复后期，进行事务订正。
1. 将事务操作记录转换为 Proposal
2. 保持在 ZKDatabase.committedLog 中
3. Follower 会进行快速同步
获取最新的 100 个
快照
依次解析
快照文件
校验 checksum
exit
返回最新的 ZXID
zxid_for_snap
处理快照文件
(全量)
处理事务日志
(增量)
获取所有
zxid_for_snap
之后的事务日志
执行事务
所有事务日志执行
完之后，会再次获
取一个 ZXID
(lastProcessedZxid)
校验 epoch
Leader周期

初始化

- 疑问：初始化最后
- 为什么要校验 Epoch?
- 如何判断校验成功失败?
- 如果失败，如何处理?

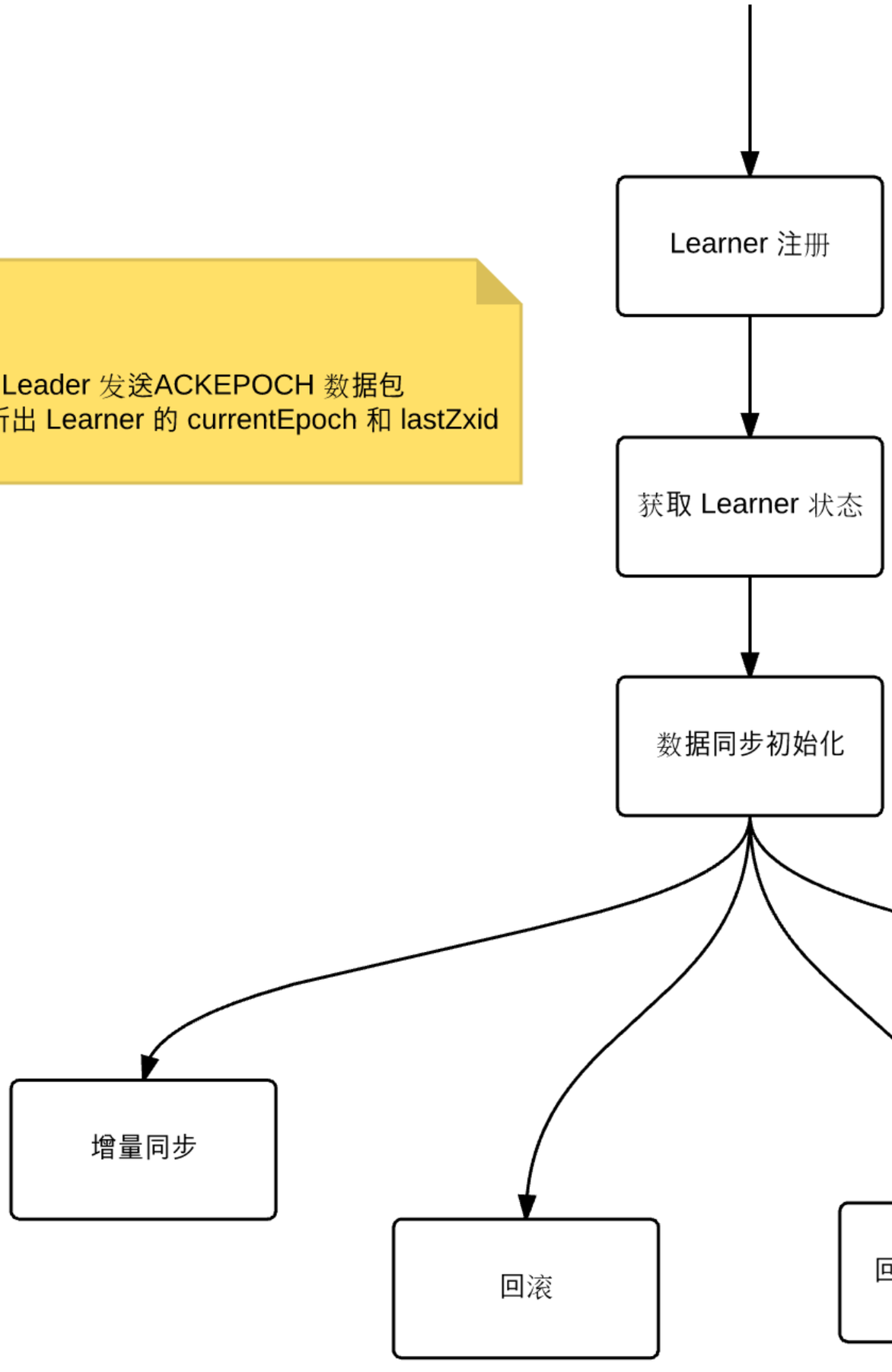
数据同步

- ZK 集群服务器启动之后，会进行 2 个动作：
  - 选举 Leader：分配角色
  - Learner 向 Leader 服务器注册：数据同步

- 数据同步，本质：
  - 将没有在 Learner 上执行的事务，同步给 Learner。

基本过程：

1. Learner 向 Leader 发送ACKEPOCH 数据包
2. Leader 解析出 Learner 的 currentEpoch 和 lastZxid



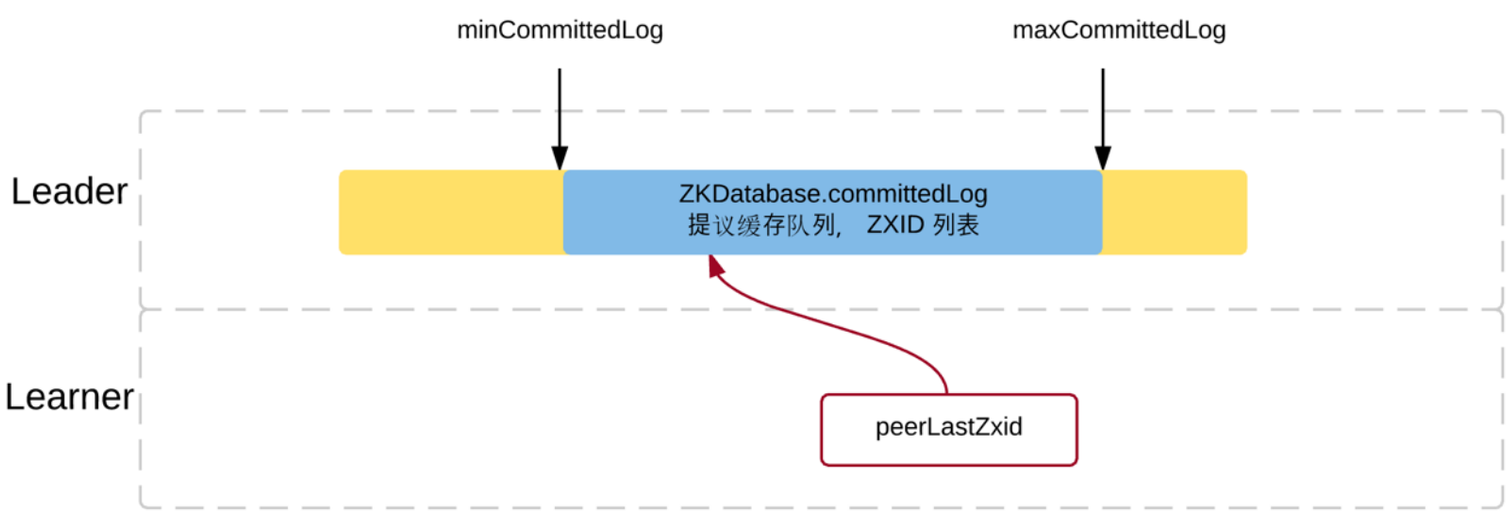
基本过程：

1. Leader 读取 Proposals（提议缓存队列）
2. 完成 3 个 zxid 的初始化：
   a. peerLastZxid：Learner 最后处理的事务 Id
   b. minCommittedLog： Proposals 中最小 ZXID
   c. maxCommittedLog： Proposals 中最大 ZXID

数据同步

- 关键点：
  - 集群启动后，什么时候能够对外提供服务？需要等所有 Learner 都完成数据同步吗?
  - 过半策略：只需要半数 Learner 完成数据同步，Learder 向所有已经完成数据同步的 Learner 发送 UPTODATE 命令，表示集群具备了对外服务能力

- 几种同步：
  - 增量同步
  - 回滚
  - 回滚，增量同步
  - 全量同步



小结

- 数据存储相关的过程：
  - 初始化：
    - 场景：ZK 服务器启动过程
    - 关键点：100 个快照，依次遍历（校验）
  - 数据同步：
    - 场景：ZK 服务器启动后，Leader 与 Learner 之间数据同步
    - 关键点：获取 learner 上 zxid 与 Proposals 中 min 和 max 的关系