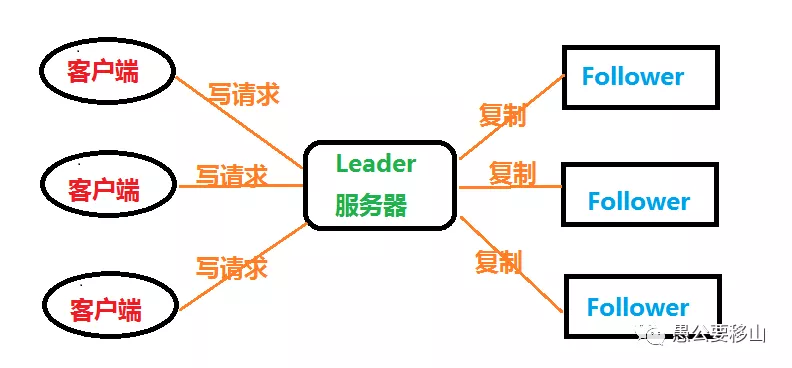
<https://mp.weixin.qq.com/s/Zl-hvWEKZ96hDwEhftX93A>

# Zookeeper中的ZAB协议

ZAB 协议是为分布式协调服务ZooKeeper专门设计的一种支持崩溃恢复的一致性协议。基于该协议，ZooKeeper 实现了一种主从模式的系统架构来保持集群中各个副本之间的数据一致性。今天主要看看这个zab协议的工作原理。

## 一、什么是ZAB协议

话说在分布式系统中一般都要使用主从系统架构模型，指的是一台leader服务器负责外部客户端的写请求。然后其他的都是follower服务器负责读。leader服务器将客户端的写操作数据同步到所有的follower节点中。



就这样，客户端发送来的写请求，全部给Leader，然后leader再转给Follower。这时候需要解决两个问题：

（1）leader服务器是如何把数据更新到所有的Follower的。

（2）Leader服务器突然间失效了，怎么办？

因此ZAB协议为了解决上面两个问题，设计了两种模式：

（1）消息广播模式：把数据更新到所有的Follower

（2）崩溃恢复模式：Leader发生崩溃时，如何恢复

OK。现在带着这两个问题，我们来详细的看一下：

## 二、ZAB协议工作原理

### 1、消息广播模式

如果你了解过2PC协议的话，理解起来就简单很多了，消息广播的过程实际上是一个简化版本的二阶段提交过程。我们来看一下这个过程：

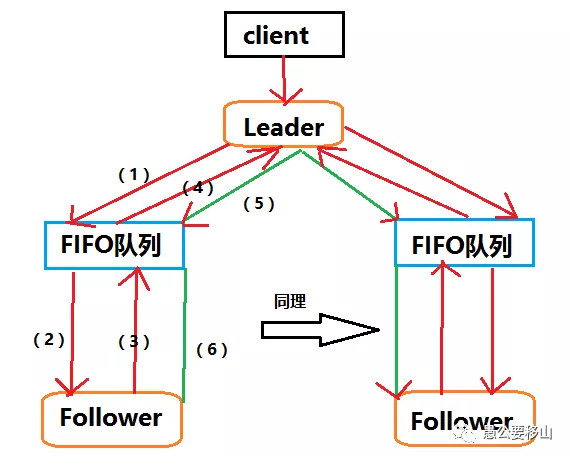
（1）Leader将客户端的request转化成一个Proposal（提议）

（2）Leader为每一个Follower准备了一个FIFO队列，并把Proposal发送到队列上。‘

（3）leader若收到follower的半数以上ACK反馈

（4）Leader向所有的follower发送commit。

其实通俗地理解就比较简单了，我是领导，我要向各位传达指令，不过传达之前我先问一下大家支不支持我，若有一半以上的人支持我，那我就向各位传达指令了。



（1）leader首先把proposal发送到FIFO队列里

（2）FIFO取出队头proposal给Follower

（3）Follower反馈一个ACK给队列

（4）队列把ACK交给leader

**（5）leader收到半数以上ACK，就会发送commit指令给FIFO队列**

（6）FIFO队列把commit给Follower。

这就是整个消息广播模式。下面我们开始看一下，如果这个leader节点崩溃了，怎么办？也就是第二种模式：崩溃回复模式。

### 2、崩溃恢复模式

leader就是一个领导，既然领导挂了，整个组织肯定不会散架，毕竟离开谁都能活下去是不是，这时候我们只需要选举一个新的领导即可，而且还要把前leader还未完成的工作做完，也就是说不仅要进行leader服务器选取，而且还要进行崩溃恢复。我们一个一个来解决。

（1）leader服务器选取

话说江湖上有一个神秘组织，这个组织分工明确，各司其职，平时这个组织的成员有三种状态：

**looking状态：也就是观望状态，这时候是由于组织出现内部问题，那就停下来，做一些其他的事。**

**following状态：自身是一个组织成员，做自己的事。**

**leading状态：自身是一个组织老大，做自己的事。**

但是这个组织只有一个老大。突然有一天，老大挂掉了，于是每一个成员的状态变成了looking状态。于是成员宣布要选举新的leader。

既然是选老大，每个人都想做，于是成员ABC开始了公平选举的过程。但是为了方便，每个人都有一个记录表，来记录当前的信息。

**第一步：成员A告诉BC说我要成为老大，BC记录下来。（A成员广播）**

**第二步：B回复可以，C回复不可以。（B成员广播）**

**第三步：A和C收到B的消息，更新自己的记录表。**

**此时A：2票，B：0票，C：0票。**

**第四步：C这时候不满意了，也要选举成为老大。而且还给自己投了一票。**

**第五步：A回复可以，B回复可以。更新自己的记录表。**

**第六步：C收到AB的回复，更新。**

**此时A：0票，B：0，C：3票。于是确定C就是下一届组织老大了。**

这就是整个选举的过程。并且每个人的选举，都代表了一个事件，为了保证分布式系统的时间有序性，因此给每一个事件都分配了一个Zxid。相当于编了一个号。低32位是按照数字递增，即每次客户端发起一个proposal,低32位的数字简单加1。高32位是leader周期的epoch编号。

每当选举出一个新的leader时，新的leader就从本地事务日志中取出ZXID,然后解析出高32位的epoch编号，进行加1，再将低32位的全部设置为0。这样就保证了每次新选举的leader后，保证了ZXID的唯一性而且是保证递增的。

OK，老大选举完了，这时候前老大遗留下来的事还没完成呢，此时就要开始恢复了。

（2）崩溃恢复

既然要恢复，有些场景是不能恢复的，ZAB协议崩溃恢复要求满足如下2个要求：第一：确保已经被leader提交的proposal必须最终被所有的follower服务器提交。第二：确保丢弃已经被leader出的但是没有被提交的proposal。

好了，现在开始进行恢复。

**第一步：选取当前取出最大的ZXID，代表当前的事件是最新的。**

**第二步：新leader把这个事件proposal提交给其他的follower节点**

**第三步：follower节点会根据leader的消息进行回退或者是数据同步操作。最终目的要保证集群中所有节点的数据副本保持一致。**

这就是整个恢复的过程，其实就是相当于有个日志一样的东西，记录每一次操作，然后把出事前的最新操作恢复，然后进行同步即可。

OK。这个就是ZAB协议的整个过程。

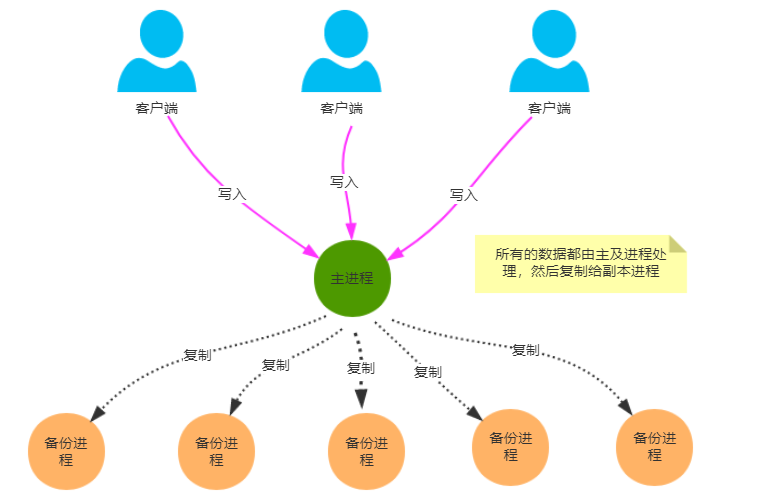
完全没看懂，淦

<https://www.cnblogs.com/stateis0/p/9062133.html>

# 补课

## 一、**什么是 ZAB 协议？ ZAB 协议介绍**

1. ZAB 协议全称：Zookeeper Atomic Broadcast（Zookeeper 原子广播协议）。
2. Zookeeper 是一个为分布式应用提供高效且可靠的分布式协调服务。在解决分布式一致性方面，Zookeeper 并没有使用 Paxos ，而是采用了 ZAB 协议。
3. ZAB 协议定义：ZAB 协议是为分布式协调服务 Zookeeper 专门设计的一种**支持 崩溃恢复 和 原子广播 协议**。下面我们会重点讲这两个东西。
4. 基于该协议，Zookeeper 实现了一种 主备模式 的系统架构来保持集群中各个副本之间数据一致性。具体如下图所示：



上图显示了 Zookeeper 如何处理集群中的数据。所有客户端写入数据都是写入到 主进程（称为 Leader）中，然后，由 Leader 复制到备份进程（称为 Follower）中。从而保证数据一致性。从设计上看，和 Raft 类似。

那么复制过程又是如何的呢？复制过程类似 2PC，ZAB 只需要 Follower 有一半以上返回 Ack 信息就可以执行提交，大大减小了同步阻塞。也提高了可用性。

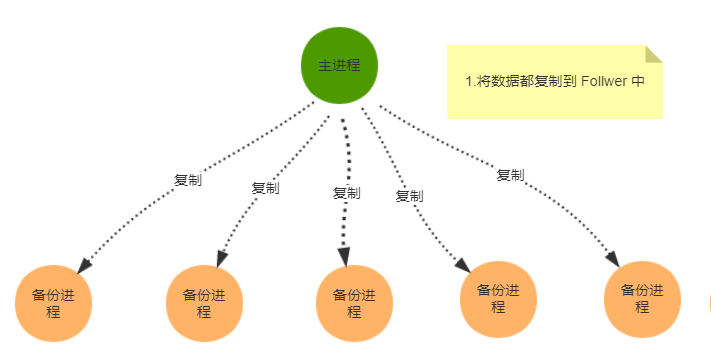
简单介绍完，开始重点介绍 消息广播 和 崩溃恢复。**整个 Zookeeper 就是在这两个模式之间切换。** 简而言之，当 Leader 服务可以正常使用，就进入消息广播模式，当 Leader 不可用时，则进入崩溃恢复模式。

## 二、消息广播

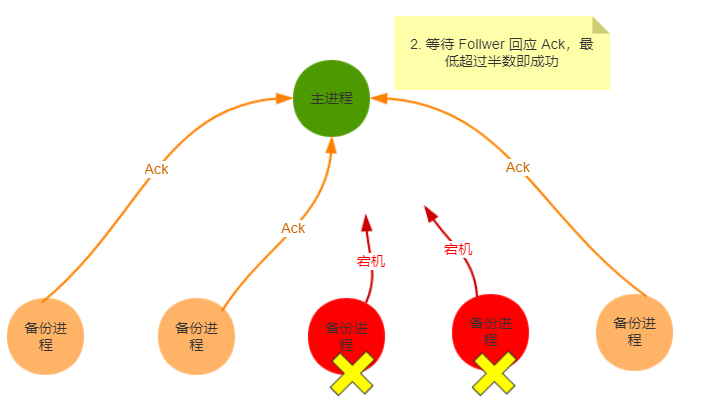
ZAB 协议的消息广播过程使用的是一个原子广播协议，类似一个 二阶段提交过程。对于客户端发送的写请求，全部由 Leader 接收，Leader 将请求封装成一个事务 Proposal，将其发送给所有 Follwer ，然后，根据所有 Follwer 的反馈，如果超过半数成功响应，则执行 commit 操作（先提交自己，再发送 commit 给所有 Follwer）。

基本上，整个广播流程分为 3 步骤：

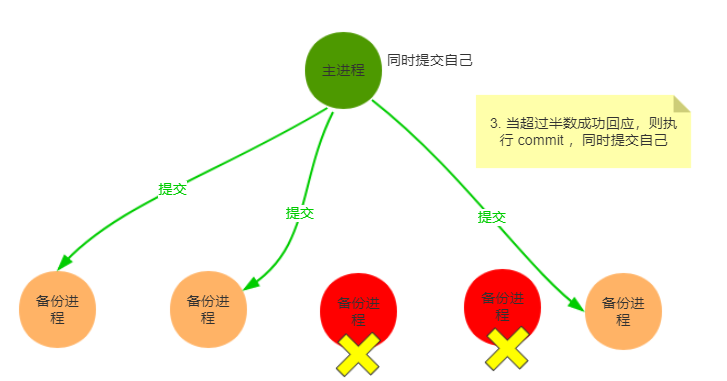
1、将数据都复制到 Follwer 中



2、等待 Follwer 回应 Ack，最低超过半数即成功



3、当超过半数成功回应，则执行 commit ，同时提交自己



通过以上 3 个步骤，就能够保持集群之间数据的一致性。实际上，在 Leader 和 Follwer 之间还有一个消息队列，用来解耦他们之间的耦合，避免同步，实现异步解耦。

还有一些细节：

* Leader 在收到客户端请求之后，会将这个请求封装成一个事务，并给这个事务分配一个全局递增的唯一 ID，称为事务ID（ZXID），ZAB 协议需要保证事务的顺序，因此必须将每一个事务按照 ZXID 进行先后排序然后处理。
* 在 Leader 和 Follwer 之间还有一个消息队列，用来解耦他们之间的耦合，解除同步阻塞。
* zookeeper集群中为保证任何所有进程能够有序的顺序执行，只能是 Leader 服务器接受写请求，即使是 Follower 服务器接受到客户端的请求，也会转发到 Leader 服务器进行处理。
* 实际上，这是一种简化版本的 2PC，不能解决单点问题。等会我们会讲述 ZAB 如何解决单点问题（即 Leader 崩溃问题）。

## 三、崩溃恢复

刚刚我们说消息广播过程中，Leader 崩溃怎么办？还能保证数据一致吗？如果 Leader 先本地提交了，然后 commit 请求没有发送出去，怎么办？

实际上，当 Leader 崩溃，即进入我们开头所说的崩溃恢复模式（崩溃即：Leader 失去与过半 Follwer 的联系）。下面来详细讲述。

假设1：Leader 在复制数据给所有 Follwer 之后崩溃，怎么办？

假设2：Leader 在收到 Ack 并提交了自己，同时发送了部分 commit 出去之后崩溃怎么办？

针对这些问题，ZAB 定义了 2 个原则：

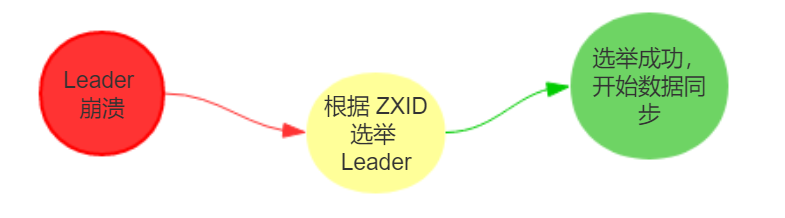
* ZAB 协议确保那些已经在 Leader 提交的事务最终会被所有服务器提交。
* ZAB 协议确保丢弃那些只在 Leader 提出/复制，但没有提交的事务。

所以，ZAB 设计了下面这样一个选举算法：

能够确保提交已经被 Leader 提交的事务，同时丢弃已经被跳过的事务。

针对这个要求，如果让 Leader 选举算法能够保证新选举出来的 Leader 服务器拥有集群中所有机器编号（即 ZXID 最大）的事务，那么就能够保证这个新选举出来的 Leader 一定具有所有已经提交的提案。

而且这么做有一个好处是：可以省去 Leader 服务器检查事务的提交和丢弃工作的这一步操作。



这样，我们刚刚假设的两个问题便能够解决。假设 1 最终会丢弃调用没有提交的数据，假设 2 最终会同步所有服务器的数据。这个时候，就引出了一个问题，如何同步？

## 数据同步

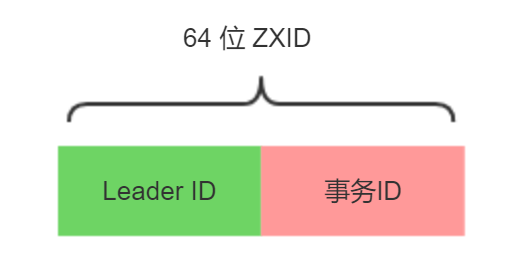
当崩溃恢复之后，需要在正式工作之前（接收客户端请求），Leader 服务器首先确认事务是否都已经被过半的 Follwer 提交了，即是否完成了数据同步。目的是为了保持数据一致。

当所有的 Follwer 服务器都成功同步之后，Leader 会将这些服务器加入到可用服务器列表中。

实际上，Leader 服务器处理或丢弃事务都是依赖着 ZXID 的，那么这个 ZXID 如何生成呢？

答：在 ZAB 协议的事务编号 ZXID 设计中，ZXID 是一个 64 位的数字，其中低 32 位可以看作是一个简单的递增的计数器，针对客户端的每一个事务请求，Leader 都会产生一个新的事务 Proposal 并对该计数器进行 + 1 操作。

而高 32 位则代表了 Leader 服务器上取出本地日志中最大事务 Proposal 的 ZXID，并从该 ZXID 中解析出对应的 epoch 值，然后再对这个值加一。



高 32 位代表了每代 Leader 的唯一性，低 32 代表了每代 Leader 中事务的唯一性。同时，也能让 Follwer 通过高 32 位识别不同的 Leader。简化了数据恢复流程。

基于这样的策略：当 Follower 链接上 Leader 之后，Leader 服务器会根据自己服务器上最后被提交的 ZXID 和 Follower 上的 ZXID 进行比对，比对结果要么回滚，要么和 Leader 同步。

## 总结

ZAB 协议和我们之前看的 Raft 协议实际上是有相似之处的，比如都有一个 Leader，用来保证一致性（Paxos 并没有使用 Leader 机制保证一致性）。再有采取过半即成功的机制保证服务可用（实际上 Paxos 和 Raft 都是这么做的）。

ZAB 让整个 Zookeeper 集群在两个模式之间转换，消息广播和崩溃恢复，消息广播可以说是一个简化版本的 2PC，通过崩溃恢复解决了 2PC 的单点问题，通过队列解决了 2PC 的同步阻塞问题。

而支持崩溃恢复后数据准确性的就是数据同步了，数据同步基于事务的 ZXID 的唯一性来保证。通过 + 1 操作可以辨别事务的先后顺序。