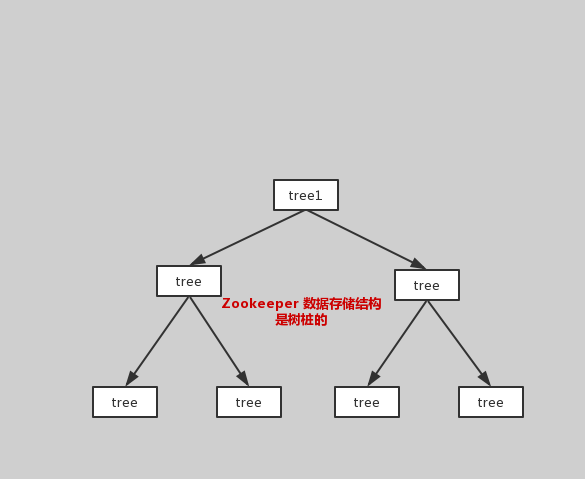
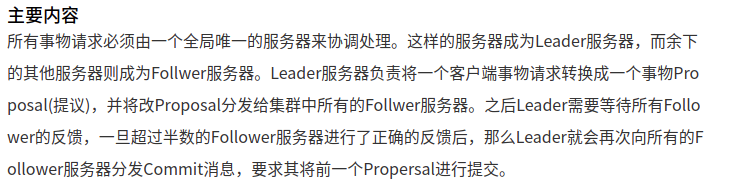
### Zookeeper 数据结构



### Zookeeper 协议？

1.0.1图



1.0.1 图主要讲解了zookeeper的两个主要内容点：第一zookeeper 集群时如何保证数据的一致性；第二个讲解了数据保证一致性的条件下，有一个两阶段提交的过程。

Zookeeper 的 ZAB（原子广播）协议两种基本的模式为崩溃恢复和消息广播:

崩溃恢复：说白了，就是当 Leader 挂时，Follower 会自动通过选举投票的策略，在选举出一个新的 Leader（第三题有详情解释），这种行为称为“崩溃恢复”。

消息广播：当一个写请求发到 Follower 服务时，Follower 服务会把这个请求，转发到 Leader 服务，这时会生成一个全局并且唯一、保证顺序的 ZXID，Leader 操作完后 commit 到其他的 Follower 服务。只要一个主节点，N个从节点，保证了数据的一致性。

### Zookeeper 部署设备为什么是奇数？

Zookeeper 部署集群设备一般而言是至少 3 台服务的，因为 zookeeper 服务中必须有一台 Leader 服务，多台 Follower 服务，所有的 Follower 必须连接 Leader 服务，这样子是为了保证数据的一致性。

为什么这么说呢？

因为所有的事务操作都必须通过 Leader 服务来执行的，Follower 服务是没有写的权限，Follower 服务只是做了一个转发的作用。

最重要的一点：zookeeper 集群有一个特性，这个特性有一个公式：当 Leader 挂时，Follower 会进行投票选举，而这个投票选举的公式是：选票数要大于集群总机器数的 一半，如果是 3 台服务，挂了一台，那就是 “2 > 3 / 2”, 这个条件成立，机器还能正常工作，如果是 2 台服务，挂了一台，那么整个集群就死了。

### Zookeeper的选举策略和投票策略？

当 Leader 挂了的时候，Follower之间会发起相互投票的消息，发起的格式如下 {myid，ZXID}

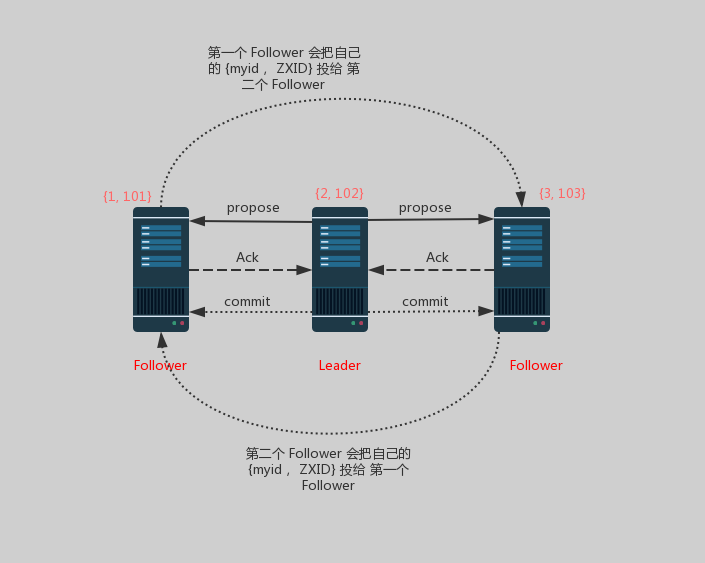
**选举策略：**

所有的　Follower 会在 propose 中发起一个自己的 id 和最新的ZXID， 给予原来没有挂的 Leader，比如1.1.0 图，{3, 103} 对应的是{myid， ZXID}， 第一台 Follower 会把自己的 {myid, ZXID}, 传递给 Leader， 同样第二台 Follower 也会一样的事情，在挂的时候 Leader 会比较第一台 Follower 和第二台 Follower的myid，ZXID谁的大，就 commit 给最大的那台 Follower，成为新的 Leader

**投票策略：**

所有的 Follower 会把自己的 id 和 最新的 ZXID 投出去，然后做一个 PK，PK 的原则是：先比较事务id 谁的比较大，就推举谁出去，比如 第二台 Follower 会把自己的 {myid， ZXID} 投给 其他的 Follower ，当第二台 Follower 投到了 第一台 Follower 会做一个比较，第一台 Follower 拿出本身最新的事务 id 进行比较 ，如果是 第二台 Follower 比第一台 Follower 大，就会推举 第二台 Follower 出去，Follower 成为 Leader 是有一个必要的条件的，如图 1.1.0 所示整个集群有 3 台机器，中间的 Leader 已经挂了，有两台 Follower 还在工作， zookeeper 公式成立的条件下，会推举新的 Follower 成为 Leader，如果公式不成立，整个集群就会崩掉，保正这个集群存活的公式就是: 选票数要大于集群总机器数的一半即可。“ 2 > 3 / 2 ”条件成立，机器就会判断这个 Follower 成为一个新的 Leader。

1.1.0 图



### Zookeeper 如何实现分布式事务 id 的唯一性？

Zookeeper 中所有写的操作，统一交给了 Leader 服务， 读的操作给予了 Follower 服务，这种主从模式保证了，数据的同步，因为所有写的操作都是在主节点，从节点如果接收到写的操作时，会转发到主节点中去做，主节点写完后，会同步到各个节点中。

所有说，一般而言，从节点在传递数据的时，会生成全局并且唯一、保证顺序的 ZXID ；

1.1.2 分布式图

