## zookeeper 的选举算法：

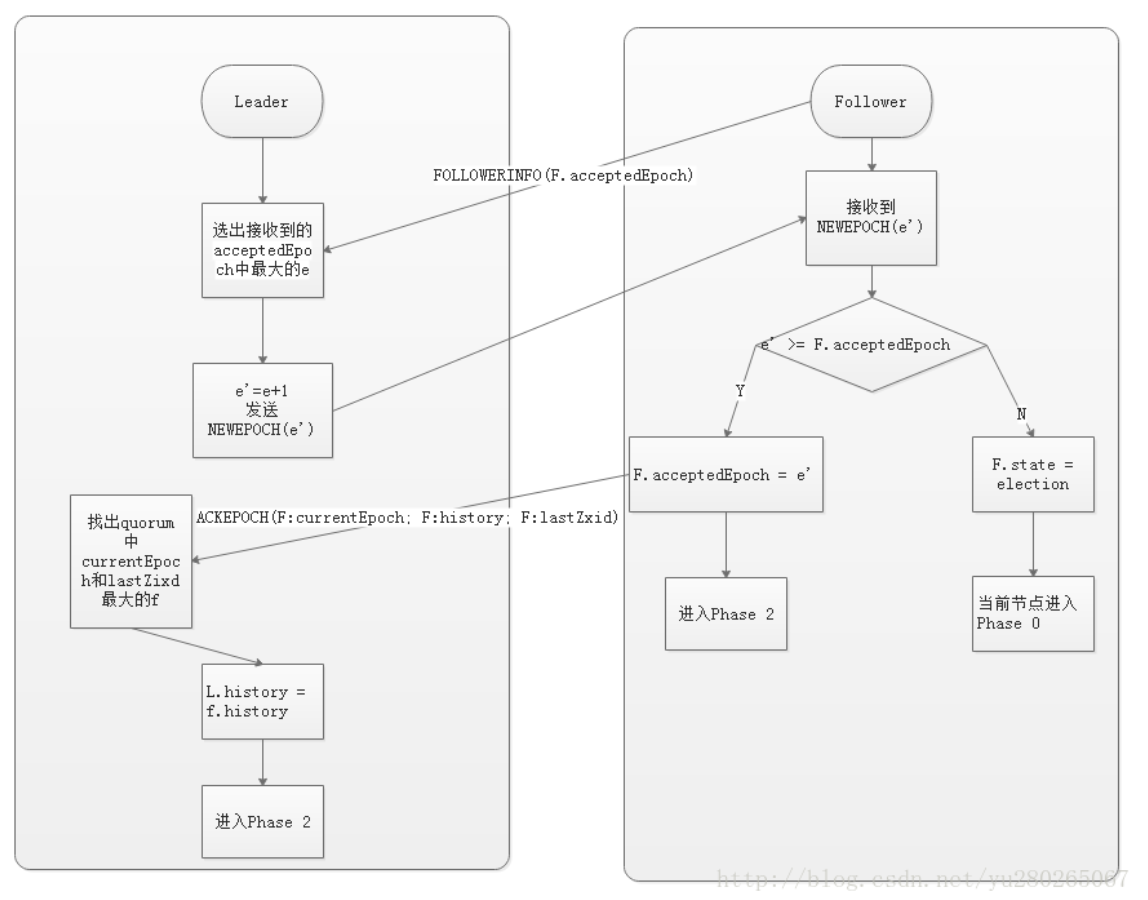
Phase 0: Leader election（选举阶段）：

节点在一开始都处于选举阶段，只要有一个节点得到超半数节点的票数，它就可以当选准 leader。只有到达 Phase 3 准 leader 才会成为真正的 leader。这一阶段的目的是就是为了选出一个准 leader，然后进入下一个阶段。   
协议并没有规定详细的选举算法，默认使用的 Fast Leader Election。   
Phase 1: Discovery（发现阶段）

在这个阶段，followers 跟准 leader 进行通信，同步 followers 最近接收的事务提议。这个一阶段的主要是三个目的：   
1.是发现当前大多数节点接收的最新提议，并且准 leader 生成新的 epoch，让 followers 接受，更新它们的 acceptedEpoch。   
2.当Follower 接受到来自准Leader 的 newEpoch 消息后，会检查当前的epoch 是否小于newEpoch，如果是就会重新赋值自己的epoch，并且向leader反馈当前的Follower 的epoch，以及钙Follower的历史事务集合。   
3.准 leader 接受到来自过半Follower的确认消息ack之后，准leader 就会从这些过半的服务器中选取一个Follower 集合，并使用该集合作为初始化集合，这个集合满足的最大epoch 与 zxid 都是所有集合中最大的（这一步骤最重要，用于同步阶段使用，同时也是zookeeper 的 leader挂机以后，新任leader 不会丢失事务的保证，也是ZAB算法与paxos算法的不同之处）

Phase 2: Synchronization（同步阶段）

同步阶段主要是利用 leader 前一阶段获得的最新提议历史，同步集群中所有的副本。只有当 quorum 都同步完成，准 leader 才会成为真正的 leader。follower 只会接收 zxid 比自己的 lastZxid 大的提议。



## kafka 选举算法：

Kafka的核心是日志文件，日志文件在集群中的同步是分布式数据系统最基础的要素。   
一旦leader down掉了，需要在followers中选择一个新的leader.但是followers本身有可能延时太久或者crash，所以必须选择高质量的follower作为leader。必须保证，一旦一个消息被提交了，但是leader down掉了，新选出的leader必须可以提供这条消息。大部分的分布式系统采用了多数投票法则选择新的leader,对于多数投票法则，就是根据所有副本节点的状况动态的选择最适合的作为leader。Kafka并不是使用这种方法。   
 Kafaka动态维护了一个同步状态的副本的集合（a set of in-sync replicas），简称ISR，在这个集合中的节点都是和leader保持高度一致的，任何一条消息必须被这个集合中的每个节点读取并追加到日志中了，才回通知外部这个消息已经被提交了。因此这个集合中的任何一个节点随时都可以被选为leader。ISR在ZooKeeper中维护。ISR中有f+1个节点，就可以允许在f个节点down掉的情况下不会丢失消息并正常提供服。ISR的成员是动态的，如果一个节点被淘汰了，当它重新达到“同步中”的状态时，他可以重新加入ISR。因此如果leader宕了，直接从ISR中选择一个follower就行。   
 那么如果所有节点都down掉了怎么办？Kafka对于数据不会丢失的保证，是基于至少一个节点是存活的，一旦所有节点都down了，这个就不能保证了。   
实际应用中，当所有的副本都down掉时，必须及时作出反应。可以有以下两种选择:   
1.等待ISR中的任何一个节点恢复并担任leader。   
2.选择所有节点中（不只是ISR）第一个恢复的节点作为leader。   
 这是一个在可用性和连续性之间的权衡。如果等待ISR中的节点恢复，一旦ISR中的节点起不起来或者数据都是了，那集群就永远恢复不了了。如果等待ISR意外的节点恢复，这个节点的数据就会被作为线上数据，有可能和真实的数据有所出入，因为有些数据它可能还没同步到。   
 Kafka目前选择了第二种策略，在未来的版本中将使这个策略的选择可配置，可以根据场景灵活的选择。   
 这种窘境不只Kafka会遇到，几乎所有的分布式数据系统都会遇到。   
以上仅仅以一个topic一个分区为例子进行了讨论，但实际上一个Kafka将会管理成千上万的topic分区.Kafka尽量的使所有分区均匀的分布到集群所有的节点上而不是集中在某些节点上，另外主从关系也尽量均衡这样每个几点都会担任一定比例的分区的leader。   
 优化leader的选择过程也是很重要的，它决定了系统发生故障时的空窗期有多久。Kafka选择一个节点作为“controller”,当发现有节点down掉的时候它负责在游泳分区的所有节点中选择新的leader,这使得Kafka可以批量的高效的管理所有分区节点的主从关系。如果controller down掉了，活着的节点中的一个会备切换为新的controller。