1. Zookeeper是什么框架

分布式开源框架，提供分布式协调服务，解决了分布式一致性。

# 1.1作用有：

1. 数据的发布与订阅
2. 负载均衡（软）
3. 统一命名服务器。
4. 分布式协调、通知
5. 集群管理
6. Master选举
7. 分布式锁
8. 分布式队列

# 1.2.保证了哪些一致性：

1. 顺序执行
2. 原子性
3. 单一视图（客户端连接到任意节点，数据模型是一致的）
4. 可靠性
5. 实时性 （最终一致）

2. Paxos算法

是分布式选举算法

3. ZAB协议

崩溃恢复和消息广播系统。只有唯一的leader能够进行事物处理，非leader服务器接受客户端的请求，必须将请求转发给leader服务器。

# 3.1 消息广播模式

原子广播协议，类似一个2阶段提交过程。针对客户端的事物发送请求时，leader服务器生成事物（分配事物ID，ZXID），对于leader，leader会为每一个follower生成一个独立的队列，将广播的事物顺序的发到出去。发送给其余节点。Follower节点接受事物后，将事物写入事物日志（处于硬盘上），（成功正常返回leader提交的事（ACK），若失败直接抛弃，无中断流程）。收集一半以上ACK（不必所有都返回），commit自身，继续发送广播commit消息给follower，follower收到消息后提交事物（会有不一致问题，数据会同步）。

发送广播消息，使用FIFO特性的TCP，保证执行顺序一致性。

# 3.2 崩溃回复模式

Leader选举成功后，整个集群都知道新的leader产生。Leader（服务器service1）崩溃：

1. 一段崩溃即刚提出事物，还未发送给follower。其他服务器节点被选为leader后，service1回复正常，事物应该被抛弃。
2. 二段崩溃即发送各个follower的commit消息前崩溃。其他follower必须全部成功提交事物。

解决：让最大编号的ZXID的节点成为leader，这两个问题直接解决了（对于问题1，自己肯定拥有最多，最完整已经提交的事物。问题2，新leader根本没收到事物请求，需让原来的leader回退掉这个commit）。

# 3.3 数据同步

新的leader产生后，正式开始工作之前，检查自己日志的事物都被半数的follower提交的。如果没有开始同步。

1.正常情况下，leader会给follower单独建立队列，发送事物和commit。完成同步后加入follower列表。

2.非正常，ZXID由64位组成，leader每产生一个事物，低32递增。高32位代表了leader的epoch编号，区分leader的变化周期。·······················································································新leader产生，该编号递增，低32清零。解决3.2问题二。

# 3.4模式切换

由广播模式转为崩溃恢复模式：

1. leader崩溃或退出
2. 集群中不存在一半的follower处于通讯状态。

由崩溃恢复转为广播模模式：新的leader选举出来

4. Paxos算法与ZAB比较

# 4.1.相同点：

1. 比如都有一个Leader，用来协调N个Follower的运行；
2. Leader要等待超半数的Follower做出正确反馈之后才进行提案
3. 每个proposal都有编号值，zab中epoch，pasxos叫ballot。

# 4.2.不同点：

1. Paxos分为读和写两个阶段，读用来收集其他主进程的提案将他们提交，写用来提出自己的提案。Zab多了一个同步阶段。ZAB中的发现相当于paxos里面的读。
2. 设计目标不一样，ZAB用于做一个高可用的分布式数据主备系统。Paxos用于一致性状态机系统。

5.持久节点和临时节点

znode节点可以是持久（persistent）节点，还可以是临时(ephemeral)节点。持久节点node，如/path 只能通过delete命令进行删除，而临时节点相反，当创建临时节点的客户端崩溃或者关闭了与Zookeeper的连接时，这个节点就会被删除

持久znode是一种很有用的znode，可以通过持久znode保存一些应用数据，即使znode的创建者崩溃或者断开连接，数据也不会丢失。例如在主-从模式中，需要保存从节点的任务分配情况，即使分配任务的主节点已经崩溃。

临时zonde传达了应用某些方面的信息，仅当创建者会话有效时信息才得以保存。例如在主-从模式中，当主节点创建得zonde为临时节点时，该临时节点存在意味着主节点存在，主节点状态处于正常运行中。如果主znode(临时节点)消失，则证明主节点崩溃。

结合实际工作中，Zookeeper主要是用于dubbo框架的注册中心。Dubbo框架的提供者会向Zookeeper下的provider目录注册自己的URL。消费者订阅提供者的注册URL，并在consumer下注册自己的URL,以便在后续执行中调用提供者。消费者获取到URL之后，netty调用提供者提供的服务。提供者若发生了变化会主动通过zookeeper推送给消费者。

**选举算法和流程**

**FastLeaderElection是zookeeper默认提供的选举算法**

选举流程

目前有5台服务器，每台服务器均没有数据，它们的编号分别是1,2,3,4,5,按编号依次启动，它们的选择举过程如下：

1. 服务器1启动，给自己投票，然后发投票信息，由于其它机器还没有启动所以它收不到反馈信息，服务器1的状态一直属于Looking。

2. 服务器2启动，给自己投票，同时与之前启动的服务器1交换结果，由于服务器2的编号大所以服务器2胜出，但此时投票数没有大于半数，所以两个服务器的状态依然是LOOKING。

3. 服务器3启动，给自己投票，同时与之前启动的服务器1,2交换信息，由于服务器3的编号最大所以服务器3胜出，此时投票数正好大于半数，所以服务器3成为leader，服务器1,2成为follower。

4. 服务器4启动，给自己投票，同时与之前启动的服务器1,2,3交换信息，尽管服务器4的编号大，但之前服务器3已经胜出，所以服务器4只能成为follower。

5. 服务器5启动，后面的逻辑同服务器4成为follower。

**Zookeeper有哪几种节点类型**  
持久：创建之后一直存在，除非有删除操作，创建节点的客户端会话失效也不影响此节点。  
持久顺序：持久节点名后缀加上一个10位数字。  
临时：创建客户端会话失效（注意是会话失效，不是连接断了），节点也就没了。不能建子节点。  
临时顺序：临时节点名后缀加上一个10位数字。  
**Zookeeper对节点的watch监听通知是永久的吗？**  
不是。官方声明：一个Watch事件是一个一次性的触发器，当被设置了Watch的数据发生了改变的时候，则服务器将这个改变发送给设置了Watch的客户端，以便通知它们。  
为什么不是永久的，举个例子，如果服务端变动频繁，而监听的客户端很多情况下，每次变动都要通知到所有的客户端，这太消耗性能了。  
一般是客户端执行getData(“/节点A”,true)，如果节点A发生了变更或删除，客户端会得到它的watch事件，但是在之后节点A又发生了变更，而客户端又没有设置watch事件，就不再给客户端发送。  
在实际应用中，很多情况下，我们的客户端不需要知道服务端的每一次变动，我只要最新的数据即可。  
**部署方式？集群中的机器角色都有哪些？集群最少要几台机器**

单机，集群，伪集群。Leader、Follower、Observer。集群最低3（2N+1）台，保证奇数，主要是为了选举算法。

**集群如果有3台机器，挂掉一台集群还能工作吗？挂掉两台呢？**  
过半存活即可用。  
**集群支持动态添加机器吗？**  
其实就是水平扩容了，Zookeeper在这方面不太好。两种方式：  
全部重启：关闭所有Zookeeper服务，修改配置之后启动。不影响之前客户端的会话。  
逐个重启：这是比较常用的方式。