《我想进大厂》之Zookeeper夺命连环9问

五分钟学算法 6/7

The following article is from 艾小仙 Author 艾小仙

艾小仙

一个愤世嫉俗，脱离低级趣味的人

谈谈你对Zookeeper的理解?

Zookeeper是一个开源的分布式协调服务，由雅虎公司创建，由于最初雅虎公司的内部研究小组的项目大多以动物的名字命名，所以后来就以Zookeeper(动物管理员)来命名了，而就是由Zookeeper来负责这些分布式组件环境的协调工作。

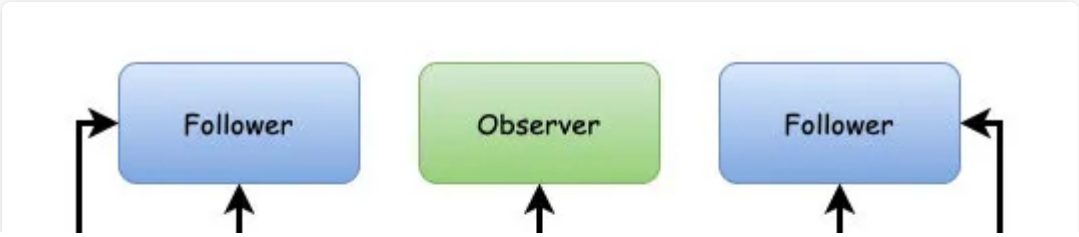
他的目标是可以提供高性能、高可用和顺序访问控制的能力，同时也是为了解决分布式环境下数据一致性的问题。

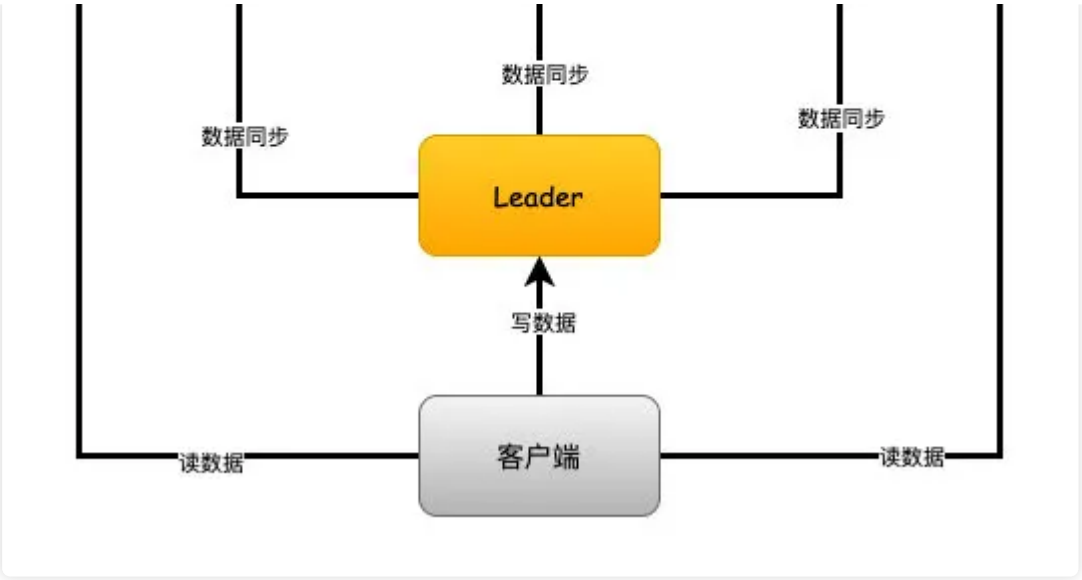
集群

首先，Zookeeper集群中有几个关键的概念，Leader、Follower和Observer，Zookeeper中通常只有Leader节点可以写入，Follower和Observer都只是负责读，但是Follower会参与节点的选举和**过半写成功**，Observer则不会，他只是单纯的提供读取数据的功能。

通常这样设置的话，是为了避免太多的从节点参与过半写的过程，导致影响性能，这样Zookeeper只要使用一个几台机器的小集群就可以实现高性能了，如果要横向扩展的话，只需要增加Observer节点即可。

Zookeeper建议集群节点个数为奇数，只要超过一半的机器能够正常提供服务，那么整个集群都是可用的状态。





数据节点Znode

Zookeeper中数据存储于内存之中，这个数据节点就叫做Znode，他是一个树形结构，比如/a/b/c类似。

而Znode又分为持久节点、临时节点、顺序节点三大类。

持久节点是指只要被创建，除非主动移除，否则都应该一直保存在Zookeeper中。

临时节点不同的是，他的生命周期和客户端Session会话一样，会话失效，那么临时节点就会被移除。

还有就是临时顺序节点和持久顺序节点，除了基本的特性之外，子节点的名称还具有有序性。

会话Session

会话自然就是指Zookeeper客户端和服务端之间的通信，他们使用TCP长连接的方式保持通信，通常，肯定会有心跳检测的机制，同时他可以接受来自服务器的Watch事件通知。

事件监听器Wather

用户可以在指定的节点上注册Wather，这样在事件触发的时候，客户端就会收到来自服务端的通知。

权限控制ACL

Zookeeper使用ACL来进行权限的控制，包含以下5种：

1. CREATE，创建子节点权限

2. DELETE，删除子节点权限
3. READ，获取节点数据和子节点列表权限
4. WRITE，更新节点权限
5. ADMIN，设置节点ACL权限

所以，Zookeeper通过集群的方式来做到高可用，通过内存数据节点Znode来达到高性能，但是存储的数据量不能太大，通常适用于读多写少的场景。

Zookeeper有哪些应用场景?

1. 命名服务Name Service，依赖Zookeeper可以生成全局唯一的节点ID，来对分布式系统中的资源进行管理。
2. 分布式协调，这是Zookeeper的核心使用了。利用Wather的监听机制，一个系统的某个节点状态发生改变，另外系统可以得到通知。
3. 集群管理，分布式集群中状态的监控和管理，使用Zookeeper来存储。
4. Master选举，利用Zookeeper节点的全局唯一性，同时只有一个客户端能够创建成功的特点，可以作为Master选举使用，创建成功的则作为Master。
5. 分布式锁，利用Zookeeper创建临时顺序节点的特性。

说说Wather监听机制和它的原理?

Zookeeper可以提供分布式数据的发布/订阅功能，依赖的就是Wather监听机制。

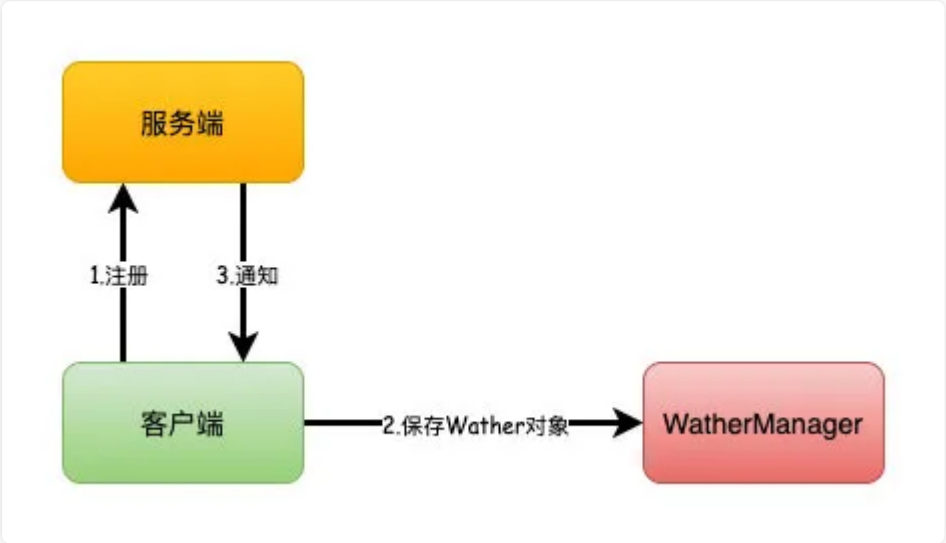
客户端可以向服务端注册Wather监听，服务端的指定事件触发之后，就会向客户端发送一个事件通知。

他有几个特性：

1. 一次性：一旦一个Wather触发之后，Zookeeper就会将它从存储中移除
2. 客户端串行：客户端的Wather回调处理是串行同步的过程，不要因为一个Wather的逻辑阻塞整个客户端
3. 轻量：Wather通知的单位是WathedEvent，只包含通知状态、事件类型和节点路径，不包含具体的事件内容，具体的时间内容需要客户端主动去重新获取数据

主要流程如下：

1. 客户端向服务端注册Wather监听
2. 保存Wather对象到客户端本地的WatherManager中
3. 服务端Wather事件触发后，客户端收到服务端通知，从WatherManager中取出对应Wather对象执行回调逻辑



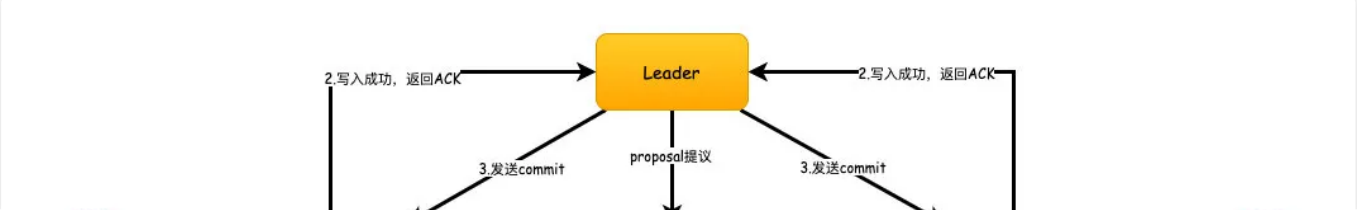
Zookeeper是如何保证数据一致性的?

Zookeeper通过ZAB原子广播协议来实现数据的最终顺序一致性，他是一个类似2PC两阶段提交的过程。

由于Zookeeper只有Leader节点可以写入数据，如果是其他节点收到写入数据的请求，则会将之转发给Leader节点。

主要流程如下：

1. Leader收到请求之后，将它转换为一个proposal提议，并且为每个提议分配一个全局唯一递增的事务ID：zxid，然后把提议放入到一个FIFO的队列中，按照FIFO的策略发送给所有的Follower
2. Follower收到提议之后，以事务日志的形式写入到本地磁盘中，写入成功后返回ACK给Leader
3. Leader在收到超过半数的Follower的ACK之后，即可认为数据写入成功，就会发送commit命令给Follower告诉他们可以提交proposal了



ZAB包含两种基本模式，崩溃恢复和消息广播。

整个集群服务在启动、网络中断或者重启等异常情况的时候，首先会进入到崩溃恢复状态，此时会通过选举产生Leader节点，当集群过半的节点都和Leader状态同步之后，ZAB就会退出恢复模式。之后，就会进入消息广播的模式。

那么，Zookeeper如何进行Leader选举的?

Leader的选举可以分为两个方面，同时选举主要包含事务zxid和myid，节点主要包含LEADING\FOLLOWING\LOOKING3个状态。

1. 服务启动期间的选举
2. 服务运行期间的选举

服务启动期间的选举

1. 首先，每个节点都会对自己进行投票，然后把投票信息广播给集群中的其他节点
2. 节点接收到其他节点的投票信息，然后和自己的投票进行比较，首先zxid较大的优先，如果zxid相同那么则会去选择myid更大者，此时大家都是LOOKING的状态
3. 投票完成之后，开始统计投票信息，如果集群中过半的机器都选择了某个节点机器作为leader，那么选举结束
4. 最后，更新各个节点的状态，leader改为LEADING状态，follower改为FOLLOWING状态

服务运行期间的选举

如果开始选举出来的leader节点宕机了，那么运行期间就会重新进行leader的选举。

1. leader宕机之后，非observer节点都会把自己的状态修改为LOOKING状态，然后重新进入选举流程
2. 生成投票信息(myid,zxid)，同样，第一轮的投票大家都会把票投给自己，然后把投票信息广播出去
3. 接下来的流程和上面的选举是一样的，都会优先以zxid，然后选择myid，最后统计投票信息，修改节点状态，选举结束

那选举之后又是怎样进行数据同步的?

那实际上Zookeeper在选举之后，Follower和Observer（统称为Learner）就会去向Leader注册，然后就会开始数据同步的过程。

数据同步包含3个主要值和4种形式。

PeerLastZxid：Learner服务器最后处理的ZXID

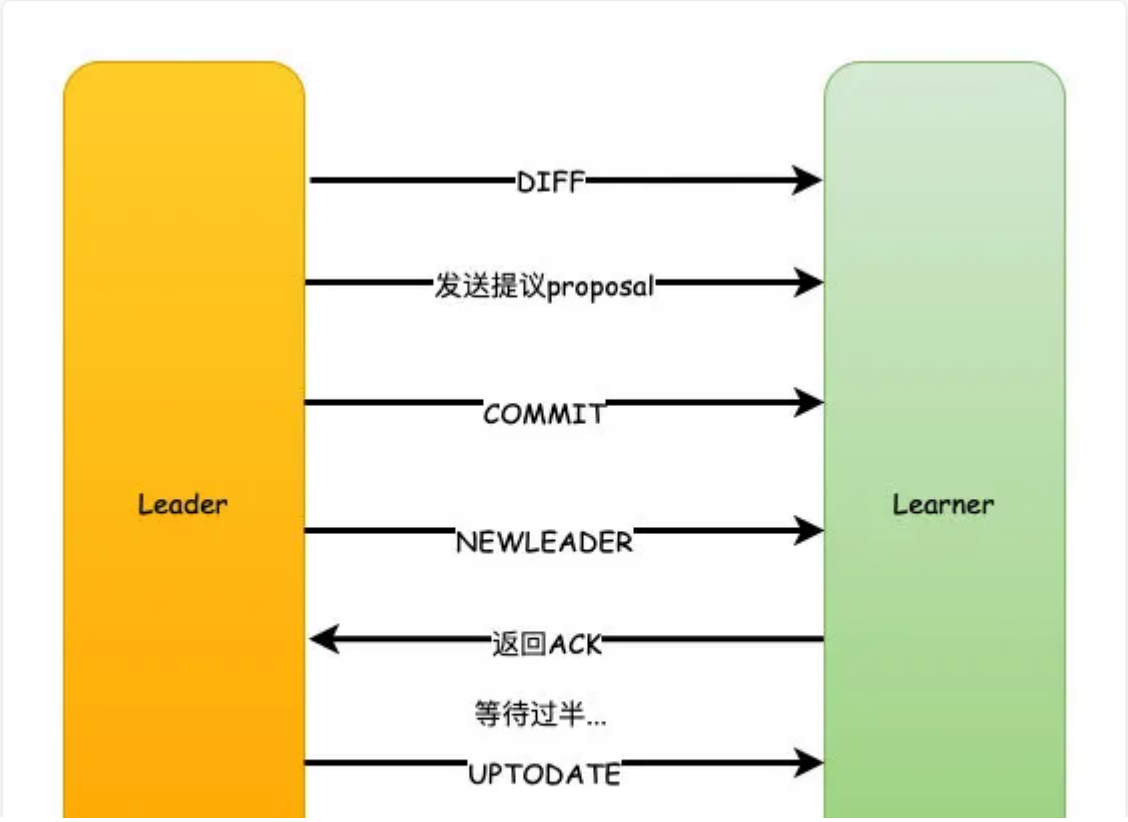
minCommittedLog：Leader提议缓存队列中最小ZXID

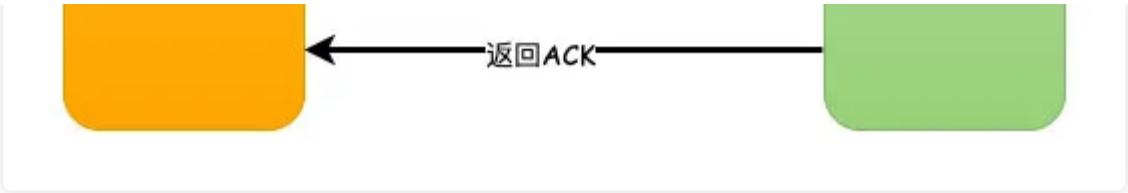
maxCommittedLog：Leader提议缓存队列中最大ZXID

直接差异化同步 DIFF同步

如果PeerLastZxid在minCommittedLog和maxCommittedLog之间，那么则说明Learner服务器还没有完全同步最新的数据。

1. 首先Leader向Learner发送DIFF指令，代表开始差异化同步，然后把差异数据（从PeerLastZxid到maxCommittedLog之间的数据）提议proposal发送给Learner
2. 发送完成之后发送一个NEWLEADER命令给Learner，同时Learner返回ACK表示已经完成了同步
3. 接着等待集群中过半的Learner响应了ACK之后，就发送一个UPTODATE命令，Learner返回ACK，同步流程结束





先回滚再差异化同步 TRUNC+DIFF同步

这个设置针对的是一个异常的场景。

如果Leader刚生成一个proposal，还没有来得及发送出去，此时Leader宕机，重新选举之后作为Follower，但是新的Leader没有这个proposal数据。

举个栗子：

假设现在的Leader是A，minCommittedLog=1，maxCommittedLog=3，刚好生成的一个proposal的ZXID=4，然后挂了。

重新选举出来的Leader是B，B之后又处理了2个提议，然后minCommittedLog=1，maxCommittedLog=5。

这时候A的PeerLastZxid=4，在(1,5)之间。

那么这一条只存在于A的提议怎么处理？

A要进行事务回滚，相当于抛弃这条数据，并且回滚到最接近于PeerLastZxid的事务，对于A来说，也就是PeerLastZxid=3。

流程和DIFF一致，只是会先发送一个TRUNC命令，然后再执行差异化DIFF同步。

仅回滚同步 TRUNC同步

针对PeerLastZxid大于maxCommittedLog的场景，流程和上述一致，事务将会被回滚到maxCommittedLog的记录。

这个其实就更简单了，也就是你可以认为TRUNC+DIFF中的例子，新的Leader B没有处理提议，所以B中minCommittedLog=1，maxCommittedLog=3。

所以A的PeerLastZxid=4就会大于maxCommittedLog了，也就是A只需要回滚就行了，不需要执行差异化同步DIFF了。

全量同步 SNAP同步

适用于两个场景：

1. PeerLastZxid小于minCommittedLog

2. Leader服务器上没有提议缓存队列，并且PeerLastZxid不等于Leader的最大ZXID

这两种场景下，Leader将会发送SNAP命令，把全量的数据都发送给Learner进行同步。

有可能会出现数据不一致的问题吗?

还是会存在的，我们可以分成3个场景来描述这个问题。

查询不一致

因为Zookeeper是过半成功即代表成功，假设我们有5个节点，如果123节点写入成功，如果这时候请求访问到4或者5节点，那么有可能读取不到数据，因为可能数据还没有同步到4、5节点中，也可以认为这算是数据不一致的问题。

解决方案可以在读取前使用sync命令。

leader未发送proposal宕机

这也就是数据同步说过的问题。

leader刚生成一个proposal，还没有来得及发送出去，此时leader宕机，重新选举之后作为follower，但是新的leader没有这个proposal。

这种场景下的日志将会被丢弃。

leader发送proposal成功，发送commit前宕机

如果发送proposal成功了，但是在将要发送commit命令前宕机了，如果重新进行选举，还是会选择zxid最大的节点作为leader，因此，这个日志并不会被丢弃，会在选举出leader之后重新同步到其他节点当中。

如果作为注册中心，Zookeeper 和Eureka、Consul、Nacos有什么区别?

	Nacos	**Eureka**	**Consul**	**Zookeeper**
一致性协议	CP+AP	AP	CP	CP
健康检查	TCP/HTTP/MYSQL/Client Beat	Client Beat	TCP/HTTP/gRPC/Cmd	Keep Alive

	Nacos	Eureka	Consul	Zookeeper
负载均衡策略	权重/ metadata/Selector	Ribbon	Fabio	—
雪崩保护	有	有	无	无
自动注销实例	支持	支持	不支持	支持
访问协议	HTTP/DNS	HTTP	HTTP/DNS	TCP
监听支持	支持	支持	支持	支持
多数据中心	支持	支持	支持	不支持
跨注册中心同步	支持	不支持	支持	不支持
SpringCloud集成	支持	支持	支持	不支持
Dubbo集成	支持	不支持	不支持	支持
K8S集成	支持	不支持	支持	不支持

最后，你对于CAP理论怎么理解?

CAP是一个分布式系统设计的定理，他包含3个部分，并且最多只能同时满足其中两个。

1. Consistency一致性，因为在一个分布式系统中，数据肯定需要在不同的节点之间进行同步，就比如Zookeeper，所以一致性就是指的是数据在不同的节点之间怎样保证一致性，对于纯理论的C而言，默认的规则是忽略掉延迟的，因为如果考虑延迟的话，因为数据同步的过程无论如何都会有延迟的，延迟的过程必然会带来数据的不一致。
2. Availability可用性，这个指的是对于每一个请求，节点总是可以在合理的时间返回合理的响应，比如Zookeeper在进行数据同步时，无法对外提供读写服务，不满足可用性要求。这里常有的一个例子是说Zookeeper选举期间无法提供服务不满足A，这个说法并不准确，因为CAP关注的是数据的读写，选举可以认为不在考虑范围之内。所以，可以认为对于数据的读写，无论响应超时还是返回异常都可以认为是不满足A。
3. Partition-tolerance分区容错性，因为在一个分布式系统当中，很有可能由于部分节点的网络问题导致整个集群之间的网络不连通，所以就产生了网络分区，整个集群的环境被分隔成不同的的子网，所以，一般说网络不可能100%的不产生问题，所以P一定会存在。

为什么只能同时满足CAP中的两个呢?

以A\B两个节点同步数据举例，由于P的存在，那么可能AB同步数据出现问题。

如果选择AP，由于A的数据未能正确同步到B，所以AB数据不一致，无法满足C。

如果选择CP，那么B就不能提供服务，就无法满足A。

> 巨人的肩膀：
>
> https://my.oschina.net/yunqi/blog/3040280
>
> 《从Paxos到Zookeeper分布式一致性原理与实践》