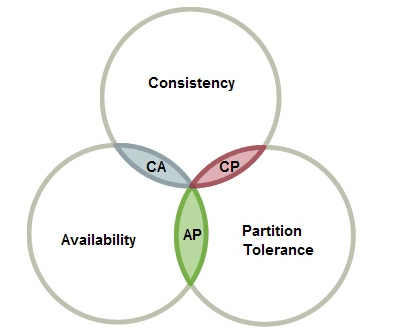
## 理论基础

### CAP理论

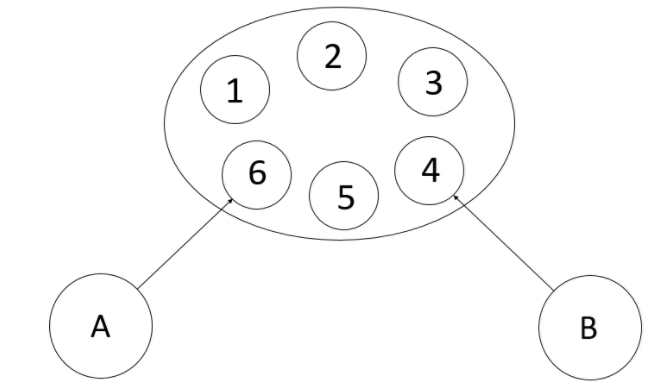
#### 简介



指的是在一个分布式系统中，一致性（Consistency）、可用性（Availability）、分区容错性（Partition tolerance）。CAP 原则指的是，这三个要素最多只能同时实现两点，不可能三者兼顾。

1. 一致性(C)：在分布式系统中的所有数据备份，在同一时刻是否同样的值
2. 可用性(A)：在集群中一部分节点故障后，集群整体是否还能响应客户端的读写请求
3. 分区容错性(P)：单台服务器或多台服务器出现问题(主要是网络问题)后，正常的服务器依然能正常提供服务，并保证前面设计号的一致性或可用性

#### 为什么不能三者共存?



假设上面的1-6是一个分布式系统，假设在网络正常的情况下，A往1里面写数据，B读取4的数据，当A往1写数据时，1同步给4，B能读取到A提交的数据。

但当出现网络问题时，假如1和4的通信挂了，意味着A提交给1的数据没法被4同步

假设为了保证可用性，我们使B能继续访问4的数据，但此时4已经不与1进行同步，所以失去了一致性

假设为了保证一致性，我们只能报错返回当前数据不一致，这样就失去了可用性。

因为是分布式系统，所以必须保证一台机器挂掉后，整个系统还能对外提供，所以分区容错性是必须的。所以我们只能在C跟A之间做抉择

### BASE理论

#### 简介

BASE：全称：Basically Available(基本可用)，Soft state（软状态）,和 Eventually consistent（最终一致性）三个短语的缩写。BASE理论是对CAP中一致性和可用性权衡的结果，其核心思想是：即使无法做到强一致性，但每个应用都可以根据自身业务的特点，采用适当的方法来使系统达到最终一致性。

1. 基本可用：假设系统，出现了不可预知的故障，但还是能用，相比较正常的系统而言可以允许响应时间上的损失或功能上的损失
2. Soft state（软状态）: 允许系统中的数据存在中间状态，并认为该状态不影响系统的整体可用性，即允许系统在多个不同节点的数据副本存在数据延时
3. Eventually consistent（最终一致性）：这个应该都懂

### 一致性协议

#### 2PC

二阶段协议是将事务的提交过程分成两个阶段：提交事务请求阶段，执行事务提交阶段

#### 3PC

略

#### Paxos算法

Paxos主要用于保证分布式存储中副本（或者状态）的一致性。副本要保持一致， 那么，所有副本的更新序列就要保持一致。

Paxos解决一致性的方法是利用选举，采用少数服从多数的思想，只要2N+1个节点中，有N个以上同意了某个决定，则认为系统达到了一致

用大白话解释这个算法：<https://www.cnblogs.com/xuwc/p/9029882.html>

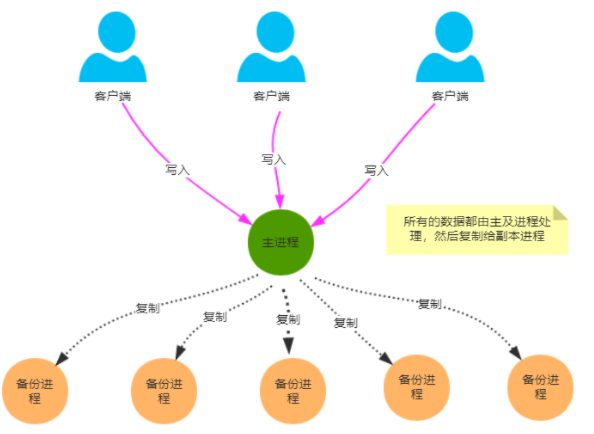
## Zookeeper

### ZAB协议

#### 简介

zookeeper采用ZAB协议来解决数据一致性问题。ZAB 协议全称：Zookeeper Atomic Broadcast（Zookeeper 原子广播协议）。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 和 原子广播 协议。

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的数据一致性



客户端写入数据都是写入到Leader，然后由Leader复制到Follower中，从而保证数据一致性。

#### 协议介绍

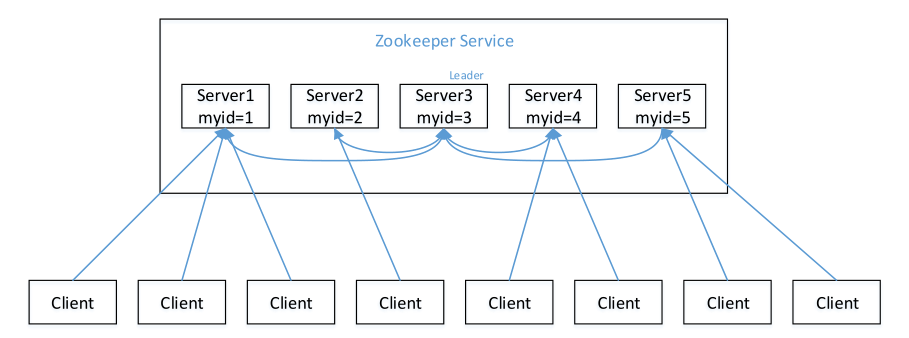
看《从Paxos到Zookeeper 分布式一致性原理与实践》的ZAB协议部分

看了一个总结的过程是这样的:

1. 当Leader接受到客户的写请求后(如果是Follower收到的会发给Leader)
2. 整个分布式系统进入广播模式，Leader服务器会为每个事务请求生成对应的Proposal进行广播，这个事务请求会分配一个全局单调递增的唯一ID（ZXID）后面用来保证先后顺序
3. Leader为每个Follower分配一个单独的队列，并将事务请求放到里面去并采用FIFO策略发送
4. 每一个Follower收到事务请求后将其写到本地磁盘中，写入成功后反馈给Leader。当Leader收到超过半数Follower的ACK响应后，就会广播一个Commit消息给所有的Follower通知其进行事务提交，同时Leader也自己进行事务提交

### Zookeeper核心概念

#### Zookeeper选举流程



（1）服务器 1 启动，此时只有它一台服务器启动了，它发出去的报没有任何响应，所

以它的选举状态一直是 LOOKING 状态。

（2）服务器 2 启动，它与最开始启动的服务器 1 进行通信，互相交换自己的选举结果，

由于两者都没有历史数据，所以 id 值较大的服务器 2 胜出，但是由于没有达到超过半数以

上的服务器都同意选举它(这个例子中的半数以上是 3)，所以服务器 1、2 还是继续保持

LOOKING 状态。

（3）服务器 3 启动，根据前面的理论分析，服务器 3 成为服务器 1、2、3 中的老大，

而与上面不同的是，此时有三台服务器选举了它，所以它成为了这次选举的 leader。

（4）服务器 4 启动，根据前面的分析，理论上服务器 4 应该是服务器 1、2、3、4 中最

大的，但是由于前面已经有半数以上的服务器选举了服务器 3，所以它只能接收当小弟的命

了。

1. 服务器 5 启动，同 4 一样当小弟。

#### Zookeeper三种角色

Leader：主要负责数据的写。所有的写操作都需要通过Leader完成再有Leader将写操作广播给其他服务器。只要有超过半数的节点写入成功，则该请求就会提交

Follower：可以直接处理客户端的读请求，但是不能处理写请求，写请求会转发给Leader处理

Observer：角色与 Follower 类似，但是无投票权。Zookeeper 需保证高可用和强一致性，为了支持更多的客户端，需要增加更多 Server；Server 增多，投票阶段延迟增大，影响性能；引入 Observer，Observer 不参与投票； Observers 接受客户端的连接，并将写请求转发给 leader 节点； 加入更多 Observer 节点，提高伸缩性，同时不影响吞吐率

### 注意点

1. zookeeper中Leader和Follower之间有一个消息队列，用来解耦他们直接的耦合，解除同步阻塞，因为Leader是需要将事务请求发给所有的Follower，如果有一台比较慢则会影响全局
2. Zookeeper保证的是最终一致性而不是强制性。