# 背景

分布式一致性算法最著名的应该是 Paxos，1990年提出，google的Chubby Lock服务就是使用的Paxos。之后的一些一致性算法基本都是在Paxos思路上的调整，例如 ZooKeeper的 ZAB。但Paxos算法一直被认为比较繁杂，很不好理解，大家对其调整优化，就是因为他的复杂。

2013年，斯坦福的两个人以易懂为目标，设计了一致性算法 Raft，现在已经被广泛应用，比较有名的是etcd，Google的Kubernetes就使用了etcd作为他的服务发现框架。

**分布式一致：**

在单节点环境中，client向node发送一个值，很容易就达成一致了



但当我们有多个node时，我们应该如何做，才能实现一致性呢？

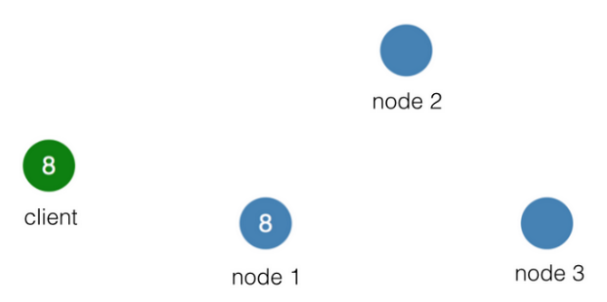
这就是分布式一致性问题，Raft就是用来解决此问题的。

# 原理

Paxos难以理解，raft应运而生，可以理解为简单的Paxos，它划分为三个子问题：

1. Leader Election
2. Log Replication
3. Safety

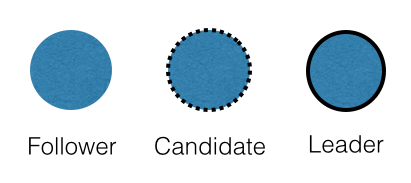
每个node都会处于以下3个状态之一：



（1）Follower 跟随者

（2）Candidate 候选人

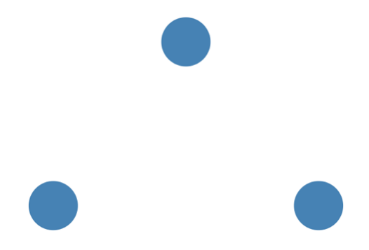
（3）Leader 领导人



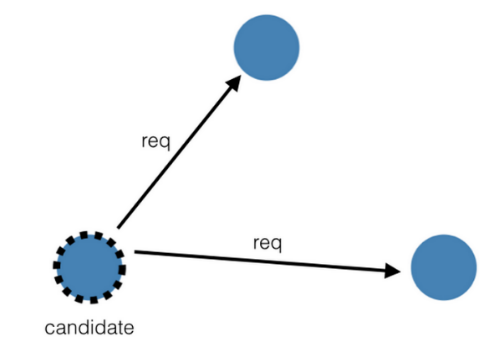
所有node开始时都是follower：

当follower没有收到leader的心跳时，他就会申请成为candidate，然后向其他node发送请求，说“我要成为Leader，请给我投票”

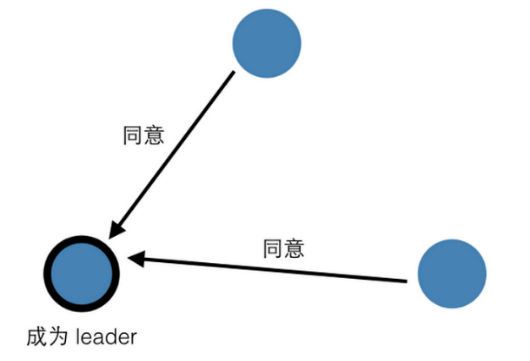
当candidate收到大多数node的同意后，就变为了Leader，以后对于系统的修改操作，都必须经过Leader

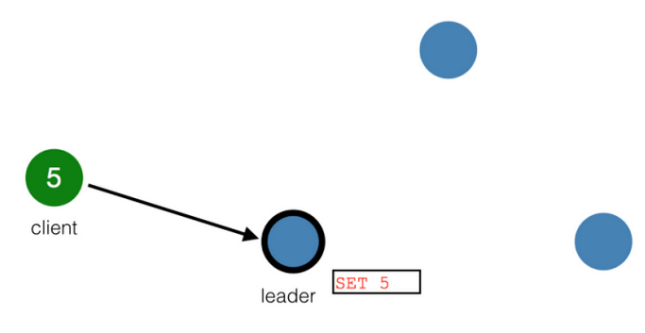


例如client要发送消息，会先发给leader，leader会把这个操作记录到自己的日志

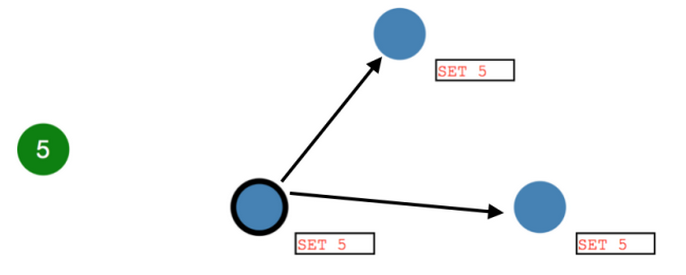


注意，是记录到日志，并没有实际修改node中的值

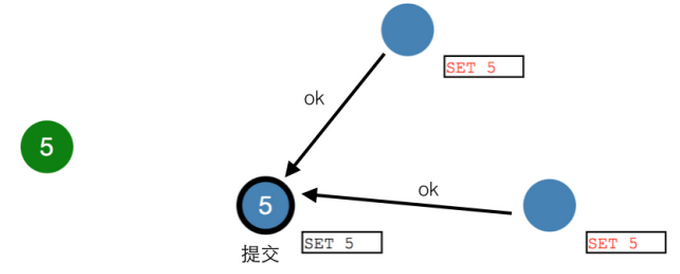




leader把这条操作记录发送给各个follower，follower收到后，也保存到自己的日志中

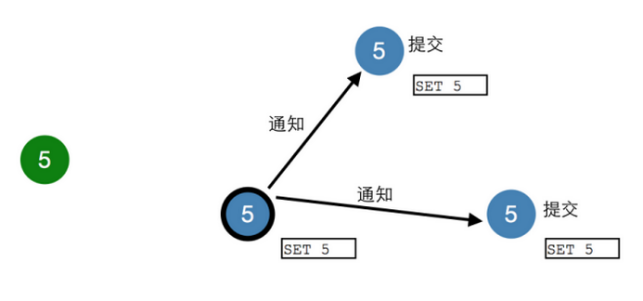


follower收到操作记录后，向leader发送消息，说自己安排好了



leader收到大多数的回馈后，就把这条记录进行提交，真正修改了node中的值

leader执行提交以后，就通知各个follower，“我已经提交了，你们可以更新了”



现在，系统就达成了一致的状态

这个过程叫做 Log Replication 日志复制，是Raft的核心之一，还有选举leader过程也是核心，就不细说了。

# 文档

如果对Raft算法有兴趣，强烈建议看一下他的动态演示

地址 http://thesecretlivesofdata.com/raft/

非常易懂，上面介绍的日志复制过程就是整理自这个演示，里面还有很多其他内容，看过后就会对Raft有了整体认识。

还有Raft的详细说明文档，中文的，很好的资料，地址：

https://github.com/maemual/raft-zh\_cn/blob/master/raft-zh\_cn.md

# 源码

## raft\_new

### log\_new

#### log\_alloc

##### log\_clear

#### raft\_set\_snapshot\_metadata

## ----

## raft\_recv\_requestvote\_response

### raft\_get\_nvotes\_for\_me

#### raft\_node\_has\_vote\_for\_me

### raft\_votes\_is\_majority

## raft\_recv\_requestsvote

### \_\_should\_grant\_vote

#### raft\_already\_voted

## ----

## raft\_recv\_entry

## raft\_msg\_entry\_response\_commited

## raft\_get\_first\_entry\_idx

## raft\_get\_last\_applied\_entry

## ----

## raft\_set\_election\_timeout

## raft\_set\_request\_timeout

## raft\_get\_election\_timeout

## raft\_get\_request\_timeout

## raft\_get\_timeout\_elapsed

## ----

## raft\_recv\_appendentries

### raft\_is\_candidate

### raft\_become\_follower

### raft\_delete\_entry\_from\_idx

#### log\_delete

##### raft\_node\_set\_voting

##### raft\_node\_set\_active

### raft\_append\_entry

#### raft\_entry\_is\_voting\_cfg\_change

#### log\_append\_entry

##### \_\_ensurecapacity

##### raft\_offer\_log

###### raft\_add\_non\_voting\_node

raft\_add\_node

raft\_node\_new

## raft\_recv\_appendentries\_response

### raft\_node\_get\_match\_idx

### raft\_voting\_change\_is\_progress

### raft\_node\_is\_voting\_commited

### raft\_node\_has\_suffient\_logs

## ----

## raft\_begin\_snapshot

## raft\_begin\_load\_snapshot

### log\_load\_from\_snapshot

## raft\_end\_snapshot

### raft\_get\_num\_snapshottable\_logs

#### raft\_get\_log\_count

#### log\_get\_base

### raft\_poll\_entry

#### log\_poll

### raft\_end\_load\_snapshot

### raft\_get\_snapshot\_last\_idx

### raft\_get\_snapshot\_last\_term

## ----

## raft\_periodic

### raft\_get\_num\_voting\_nodes

#### raft\_node\_is\_active

#### raft\_node\_is\_voting

### raft\_node\_is\_viting

### raft\_get\_my\_node

#### raft\_get\_nodeid

#### raft\_node\_get\_id

### raft\_is\_leader

### raft\_become\_leader

#### raft\_set\_state

#### raft\_node\_set\_next\_idx

#### raft\_get\_current\_idx

#### raft\_node\_set\_match\_idx

#### raft\_send\_appendentries

### raft\_send\_appendentries\_all

### raft\_snapshot\_is\_in\_progress

### raft\_get\_num\_voting\_nodes

### raft\_get\_commit\_idx

### raft\_apply\_all

## ----

## raft\_free

## raft\_clear

## raft\_set\_callbacks

## ----

## raft\_get\_num\_nodes

## raft\_get\_voted\_for

## raft\_get\_node\_from\_idx

## raft\_get\_current\_leader

## raft\_get\_current\_leader\_node

### log\_get\_at\_idx

## raft\_set\_last\_applied\_idx

## raft\_is\_follower

## raft\_is\_connected

## ----

## raft\_node\_get\_udata

## raft\_node\_set\_udata

## raft\_node\_has\_vote\_for\_me