一 KV操作&分布式锁

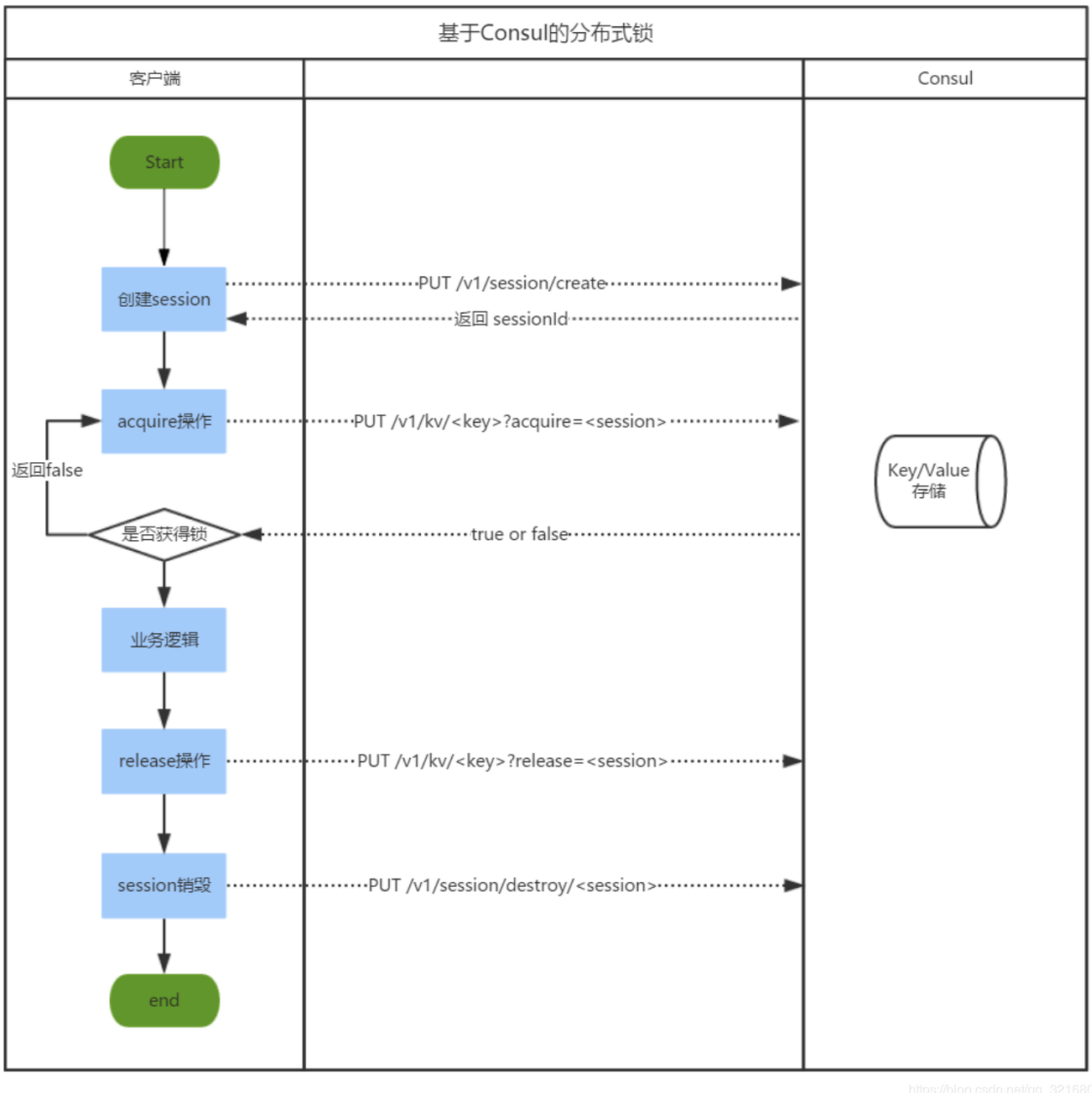
1 环境启动测试

dotnet Zhaoxi.Microservice.TestProject.dll --port=8500

dotnet Zhaoxi.Microservice.TestProject.dll --port=9500

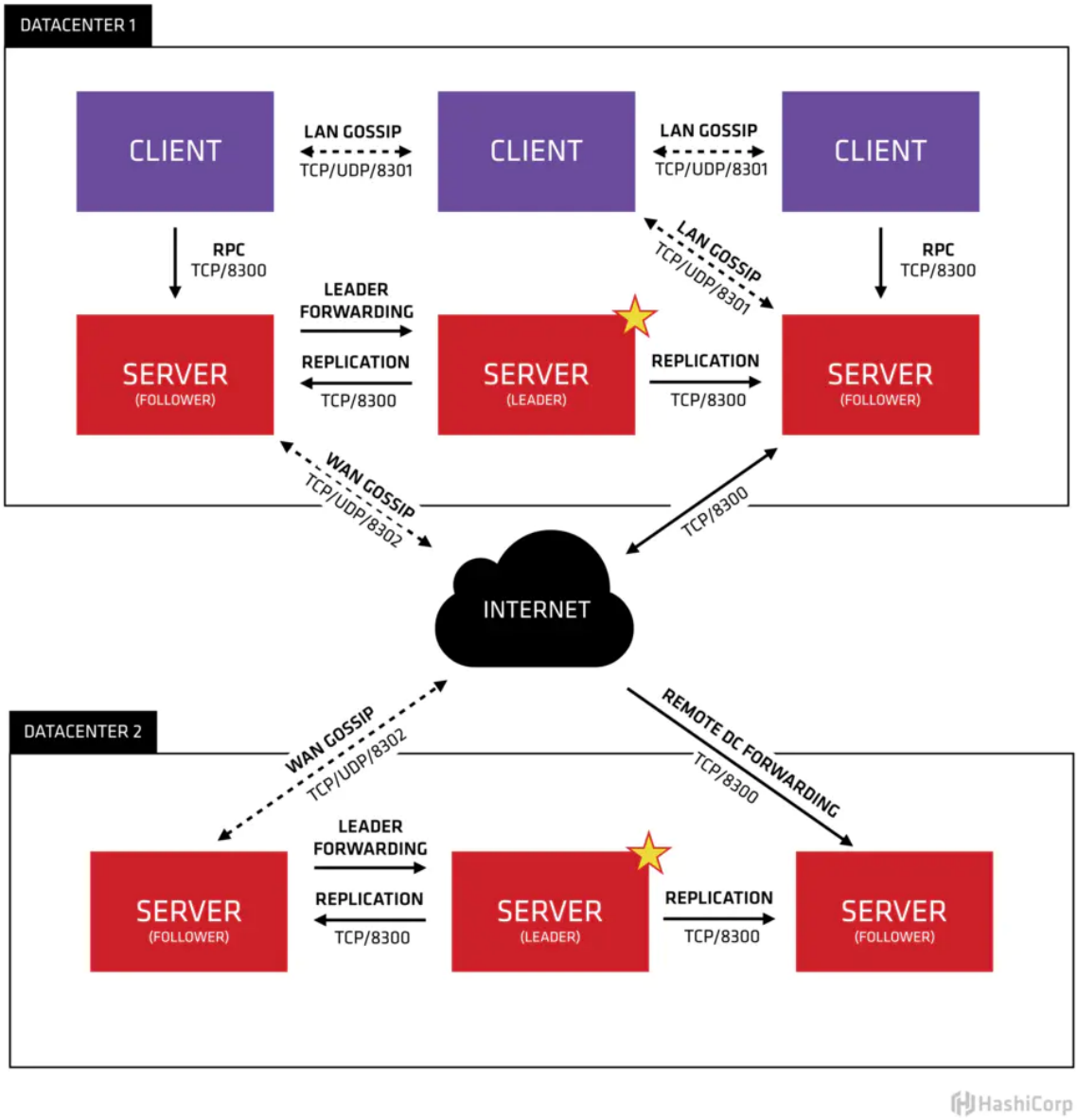
2 NConsul源码---Lock类里面---推荐用Action那种

3 大概流程图



二 Consul集群

架构图



图中包含两个`Consul`数据中心，每个数据中心都是一个`consul`的集群。在数据中心1中，可以看出`consul`的集群是由`N`个`SERVER`，加上`M`个`CLIENT`组成的。而不管是`SERVER`还是`CLIENT`，都是`consul`集群的一个节点。所有的服务都可以注册到这些节点上，正是通过这些节点实现服务注册信息的共享。

- **CLIENT**

`CLIENT`表示`consul`的`client`模式，就是**客户端模式。**是`consul`节点的一种模式，这种模式下，所有注册到当前节点的服务会被**转发**到`SERVER`节点，本身是**不持久化**这些信息。

- **SERVER**

`SERVER`表示`consul`的`server`模式，表明这个`consul`是个`server`节点。这种模式下，功能和`CLIENT`都一样，唯一不同的是，它会把所有的信息**持久化**的本地。这样遇到故障，信息是可以被保留的。

- **SERVER-LEADER**

中间那个`SERVER`下面有`LEADER`的描述，表明这个`SERVER`节点是它们的老大。和其它`SERVER`不一样的一点是，它需要负责**同步注册信息**给其它的`SERVER`，同时也要负责**各个节点**的**健康监测。**

- **其它信息**

其它信息包括各个节点之间的**通信方式**，还有**一些协议信息**、**算法。**它们是用于保证节点之间的**数据同步**、**实时性要求**等等一系列集群问题的解决。这些有兴趣的自己看看官方文档。

Docker安装Consul集群

1. 拉取consul官方镜像

docker pull consul:latest

环节架构：

容器名称	容器IP地址	映射端口号	宿主机IP地址	服务运行模式
node1	172.18.0.1	8500 -> 8500	39.96.82.51	Server Master
node2	172.18.0.3	9500 -> 8500	39.96.82.51	Server
node3	172.18.0.4	10500 -> 8500	39.96.82.51	Server
node4	172.18.0.5	11500 -> 8500	39.96.82.51	Client

`Consul`的配置参数信息说明：

参数列表	参数的含义和使用场景说明
advertise	通知展现地址用来改变我们给集群中的其他节点展现的地址，一般情况下-bind地址就是展现地址
bootstrap	用来控制一个server是否在bootstrap模式，在一个datacenter中只能有一个server处于bootstrap模式，当一个server处于bootstrap模式时，可以自己选举为raft leader
bootstrap-expect	在一个datacenter中期望提供的server节点数目，当该值提供的时候，consul一直等到达到指定sever数目的时候才会引导整个集群，该标记不能和bootstrap共用
bind	该地址用来在集群内部的通讯IP地址，集群内的所有节点到地址都必须是可达的，默认是0.0.0.0
client	consul绑定在哪个client地址上，这个地址提供HTTP、DNS、RPC等服务，默认是127.0.0.1
config-file	明确的指定要加载哪个配置文件
config-dir	配置文件目录，里面所有以.json结尾的文件都会被加载
data-dir	提供一个目录用来存放agent的状态，所有的agent允许都需要该目录，该目录必须是稳定的，系统重启后都继续存在
dc	该标记控制agent允许的datacenter的名称，默认是dc1
encrypt	指定secret key，使consul在通讯时进行加密，key可以通过consul keygen生成，同一个集群中的节点必须使用相同的key
join	加入一个已经启动的agent的ip地址，可以多次指定多个agent的地址。如果consul不能加入任何指定的地址中，则agent会启动失败，默认agent启动时不会加入任何节点
retry-interval	两次join之间的时间间隔，默认是30s
retry-max	尝试重复join的次数，默认是0，也就是无限次尝试
log-level	consul agent启动后显示的日志信息级别。默认是info，可选：trace、debug、info、warn、err
node	节点在集群中的名称，在一个集群中必须是唯一的，默认是该节点的主机名
protocol	consul使用的协议版本
rejoin	使consul忽略先前的离开，在再次启动后仍旧尝试加入集群中
server	定义agent运行在server模式，每个集群至少有一个server，建议每个集群的server不要超过5个
syslog	开启系统日志功能，只在linux/osx上生效
pid-file	提供一个路径来存放pid文件，可以使用该文件进行SIGINT/SIGHUP(关闭/更新)agent

2 启动docker-consul Server实例

运行`consul`镜像，启动`Server Master`节点`node1`：

```
docker run -d --name=node1 --restart=always \
        -e 'CONSUL_LOCAL_CONFIG={"skip_leave_on_interrupt": true}' \
        -p 8300:8300 \
        -p 8301:8301 \
        -p 8301:8301/udp \
        -p 8302:8302/udp \
        -p 8302:8302 \
        -p 8400:8400 \
        -p 8500:8500 \
        -p 8600:8600 \
        -h node1 \
        consul agent -server -bind=0.0.0.0 -bootstrap-expect=3 -node=node1 \
        -data-dir=/tmp/data-dir -client 0.0.0.0 -ui
```

查看node1情况

```
docker logs -f node1
```

http://39.96.82.51:8500/ui/dc1/nodes

node2:

```
docker run -d --name=node2 --restart=always \
        -e 'CONSUL_LOCAL_CONFIG={"skip_leave_on_interrupt": true}' \
        -p 9300:8300  \
        -p 9301:8301 \
        -p 9301:8301/udp \
        -p 9302:8302/udp \
        -p 9302:8302 \
        -p 9400:8400 \
        -p 9500:8500 \
        -p 9600:8600 \
        -h node2 \
        consul agent -server -bind=0.0.0.0 \
        -join=39.96.82.51 -node-id=$(uuidgen | awk '{print tolower($0)}') \
        -node=node2 \
        -data-dir=/tmp/data-dir -client 0.0.0.0 -ui
```

```
docker logs -f node2
```

node3:

```
docker run -d --name=node3 --restart=always \
        -e 'CONSUL_LOCAL_CONFIG={"skip_leave_on_interrupt": true}' \
        -p 10300:8300  \
        -p 10301:8301 \
        -p 10301:8301/udp \
        -p 10302:8302/udp \
        -p 10302:8302 \
        -p 10400:8400 \
        -p 10500:8500 \
```

```
-p 10600:8600 \
-h node2 \
consul agent -server -bind=0.0.0.0 \
-join=39.96.82.51 -node-id=$(uuidgen | awk '{print tolower($0)}') \
-node=node3 \
-data-dir=/tmp/data-dir -client 0.0.0.0 -ui
```

当3个`Server`节点都启动并正常运行时，观察`node2`和`node3`的进程日志，可以发现`node1`被选举为`leader`节点，也就是这个**数据中心**的`Server Master`。

观察日志发现，`node2`和`node3`都成功join到了`node1`所在的数据中心`dc1`。当集群中有3台`Consul Server`启动时，`node1`被选举为`dc1`中的主节点。然后，`node1`会通过心跳检查的方式，不断地对`node2`和`node3`进行健康检查。

3 启动Client节点

node4:

```
docker run -d --name=node4  --restart=always \
    -e 'CONSUL_LOCAL_CONFIG={"leave_on_terminate": true}' \
    -p 11300:8300 \
    -p 11301:8301 \
    -p 11301:8301/udp \
    -p 11302:8302/udp \
    -p 11302:8302 \
    -p 11400:8400 \
    -p 11500:8500 \
    -p 11600:8600 \
    -h node4 \
    consul agent -bind=0.0.0.0 -retry-join=39.96.82.51  \
    -node-id=$(uuidgen | awk '{print tolower($0)}') \
    -node=node4 -client 0.0.0.0 -ui
```

可以发现：`node4`是以`Client`模式启动运行的。启动后完成后，把`dc1`数据中心中的以`Server`模式启动的节点`node1`、`node2`和`node3`都添加到**本地缓存列表**中。当客户端向`node4`发起服务发现的请求后，`node4`会通过`RPC`将请求转发给`Server`节点中的其中一台做处理。

http://39.96.82.51:8500/

http://39.96.82.51:9500/

http://39.96.82.51:10500/

http://39.96.82.51:11500/

查看集群状态

--清理全部容器实例

docker stop $(docker ps -q) & docker rm $(docker ps -aq)

--查看节点

docker exec -t node1 consul members

--查看主从信息

docker exec -t node1 consul operator raft list-peers

--集群参数get/set测试

docker exec -t node1 consul kv put eleven 11

docker exec -t node2 consul kv get eleven

docker exec -t node4 consul kv get eleven

docker exec -t node4 consul kv put seven 7

--关闭leader

docker stop id

查看状态

docker logs -f node2

kill还有2个时可以选举，只剩1个就集群失败，数据不丢失的，可以重新挨个儿启动

三 Docker部署实例集群

1 Dockerfile部署

1 dockfile在项目文件夹一层，copy依赖类库，配置文件增加

2 copy代码---dockfile

3 build镜像

docker build -t api31v2.730 -f Dockerfile .

4 run

docker run -itd -p 5726:80 api31v2.730

5 常用操作：

docker logs id

docker exec -it id /bin/bash

ls cd

docker stop id

docker start id

docker rm id

docker rmi id

2 Docker-Compose部署

1 准备yml文件，cd到目录

2 docker-compose up –d

3 docker-compose stop

4 需要挂载文件

```
version: '3.3'
services:
  service1:
    build:
      context: /elevenmicro/webapidemo
    image: compose31v5.730
    ports:
     - 5727:80/tcp
       command: ["dotnet", "/app/Zhaoxi.Microservice.WebApiDemo"]
       volumes:
         - /elevenmicro/webapidemo/appsettings/appsettings5727.json:/app/appsettings.json
     service2:
         image: compose31v5.730
         ports:
              - 5728:80/tcp
           command: ["dotnet", "/app/Zhaoxi.Microservice.WebApiDemo","--test=5758"]
           volumes:
                -
/elevenmicro/webapidemo/appsettings/appsettings5728.json:/app/appsettings.json
```

四 Docker安装Registrator

1、拉取Registrator的镜像

```
docker pull gliderlabs/registrator:latest
```

2、启动Registrator节点

```
docker run -d --name=registrator \
          -v /var/run/docker.sock:/tmp/docker.sock \
          --net=host \
          gliderlabs/registrator -ip="39.96.82.51" consul://39.96.82.51:8500
```

--net指定为host表明使用主机模式。 -ip用于指定宿主机的IP地址，用于健康检查的通信地址。consul://39.96.82.51:8500: 使用Consul作为服务注册表，指定具体的Consul通信地址进行服务注册和注销（注意：8500是Consul对外暴露的HTTP通信端口）。

查看`Registrator`的容器进程启动日志：

```
2017/11/14 02:00:35 consul: current leader  172.17.0.2:8300
2017/11/14 02:00:35 Listening for Docker events ...
2017/11/14 02:00:35 Syncing services on 5 containers
2017/11/14 02:00:35 ignored: ddbbfa22a1a3 no published ports
2017/11/14 02:00:35 added: 3f79e23999bf ubuntu:node4:8302:udp
2017/11/14 02:00:35 added: 3f79e23999bf ubuntu:node4:8400
2017/11/14 02:00:35 added: 3f79e23999bf ubuntu:node4:8500
2017/11/14 02:00:36 added: 3f79e23999bf ubuntu:node4:8600
2017/11/14 02:00:36 added: 3f79e23999bf ubuntu:node4:8300
2017/11/14 02:00:36 added: 3f79e23999bf ubuntu:node4:8301
2017/11/14 02:00:36 added: 3f79e23999bf ubuntu:node4:8301:udp
2017/11/14 02:00:36 added: 3f79e23999bf ubuntu:node4:8302
2017/11/14 02:00:36 ignored: 3f79e23999bf port 8600 not published on host
2017/11/14 02:00:36 added: 8bb13ba449d0 ubuntu:node3:8301:udp
2017/11/14 02:00:36 added: 8bb13ba449d0 ubuntu:node3:8500
2017/11/14 02:00:36 added: 8bb13ba449d0 ubuntu:node3:8300
2017/11/14 02:00:36 added: 8bb13ba449d0 ubuntu:node3:8301
2017/11/14 02:00:36 added: 8bb13ba449d0 ubuntu:node3:8302
2017/11/14 02:00:36 added: 8bb13ba449d0 ubuntu:node3:8302:udp
2017/11/14 02:00:36 added: 8bb13ba449d0 ubuntu:node3:8400
2017/11/14 02:00:36 ignored: 8bb13ba449d0 port 8600 not published on host
2017/11/14 02:00:36 added: 8bb13ba449d0 ubuntu:node3:8600
2017/11/14 02:00:36 added: 02c1b5eb183a ubuntu:node2:8400
2017/11/14 02:00:36 added: 02c1b5eb183a ubuntu:node2:8600
2017/11/14 02:00:36 ignored: 02c1b5eb183a port 8600 not published on host
2017/11/14 02:00:36 added: 02c1b5eb183a ubuntu:node2:8301
2017/11/14 02:00:36 added: 02c1b5eb183a ubuntu:node2:8301:udp
2017/11/14 02:00:36 added: 02c1b5eb183a ubuntu:node2:8500
2017/11/14 02:00:36 added: 02c1b5eb183a ubuntu:node2:8300
2017/11/14 02:00:36 added: 02c1b5eb183a ubuntu:node2:8302
2017/11/14 02:00:36 added: 02c1b5eb183a ubuntu:node2:8302:udp
2017/11/14 02:00:36 added: 5be7ff12e62f ubuntu:node1:8600
2017/11/14 02:00:36 added: 5be7ff12e62f ubuntu:node1:8300
2017/11/14 02:00:36 added: 5be7ff12e62f ubuntu:node1:8301
2017/11/14 02:00:36 added: 5be7ff12e62f ubuntu:node1:8302:udp
2017/11/14 02:00:36 added: 5be7ff12e62f ubuntu:node1:8400
2017/11/14 02:00:36 added: 5be7ff12e62f ubuntu:node1:8500
```

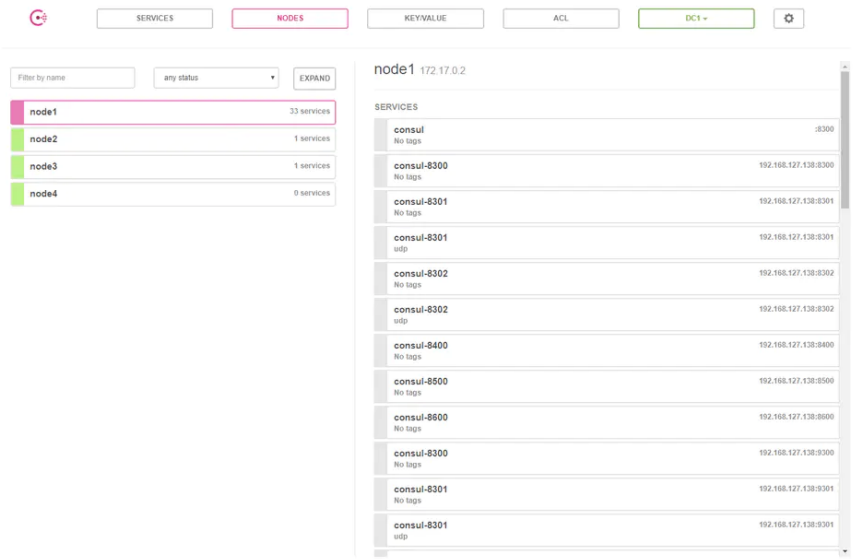
`Registrator`在启动过程完成了以下几步操作：

1. 查看Consul数据中心的leader节点，作为服务注册表；
2. 同步当前宿主机的启用容器，以及所有的服务端口；
3. 分别将各个容器发布的服务地址/端口注册到Consul的服务注册列表。

3、 查看Consul的注册状态

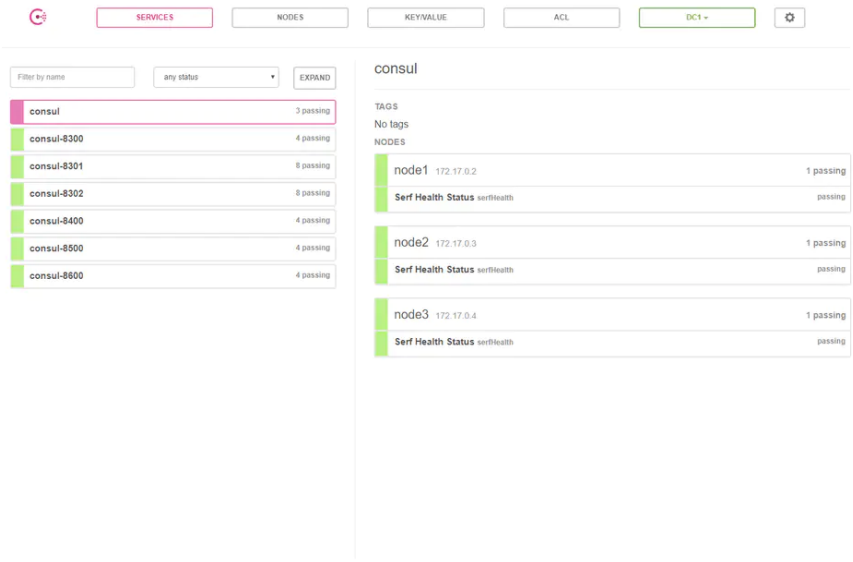
`Consul`提供了一个`Web UI`来可视化**服务注册列表**、**通信节点**、**数据中心**和**键/值存储**等，直接访问宿主机的`8500`端口。

服务注册列表：



`NODES`节点下挂载着`dc1`数据中心中的所有的`Consul`节点，包括`Consul Server`和`Client`。

通信节点列表：



启动 `Registrator` 以后，宿主机中的所有容器把服务都注册到 `Consul` 的 `SERVICES` 上，测试完成！本身没啥别的价值

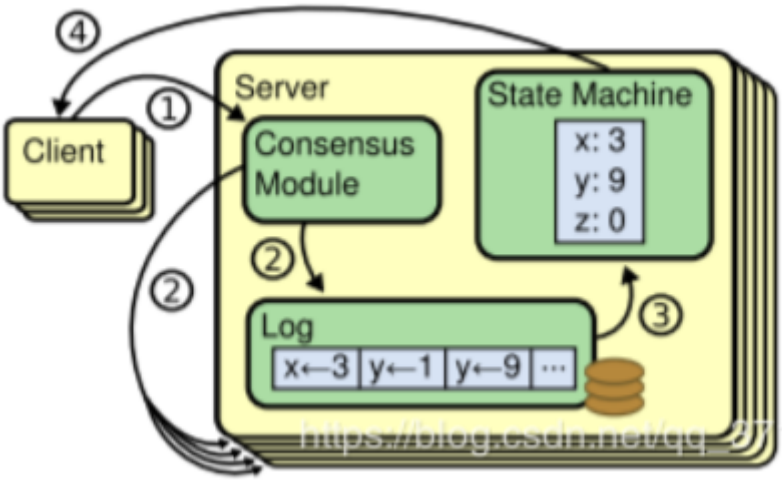
五 Raft 一致性协议简介

比如ETCD、TiDB还有各种Docker容器的编排系统 Consul用的也是

为了保证一致性（Consensus），是构建具有容错性（fault-tolerant）的分布式系统的基础。 在一个具有一致性的性质的集群里面，同一时刻所有的结点对存储在其中的某个值都有相同的结果，即对其共享的存储保持一致。

集群具有自动恢复的性质，当少数结点失效的时候不影响集群的正常工作，当大多数集群中的结点失效的时候，集群则会停止服务（不会返回一个错误的结果）。

在一个典型的一致性架构中，整个集群系统是由多个副本状态机（replicated state mathines)组成，对外部客户端的访问，整个系统就想一台非常可靠的状态机。



系统中每个结点，有三个组件*日志（Log）*、*一致性模块（Concensus Module）* 、*状态机（State Machine）* 三个部分组成。

状态机：当我们说一致性的时候，实际就是在说要保证这个状态机的一致性。状态机会从log里面取出所有的命令，然后执行一遍，得到的结果就是我们对外提供的保证了一致性的数据。

Log：Log中保存了所有的修改记录，比如如果是个hash表的话，就会依次记录各种set、delete信息。

一致性模块：　一致性模块算法就是用来保证写入的log的命令的一致性，这里可以是paxos算法也可以是raft算法。

Raft　算法的总览

Raft算法主要有三个方面，即leader的选举、日志的复制、和安全机制。

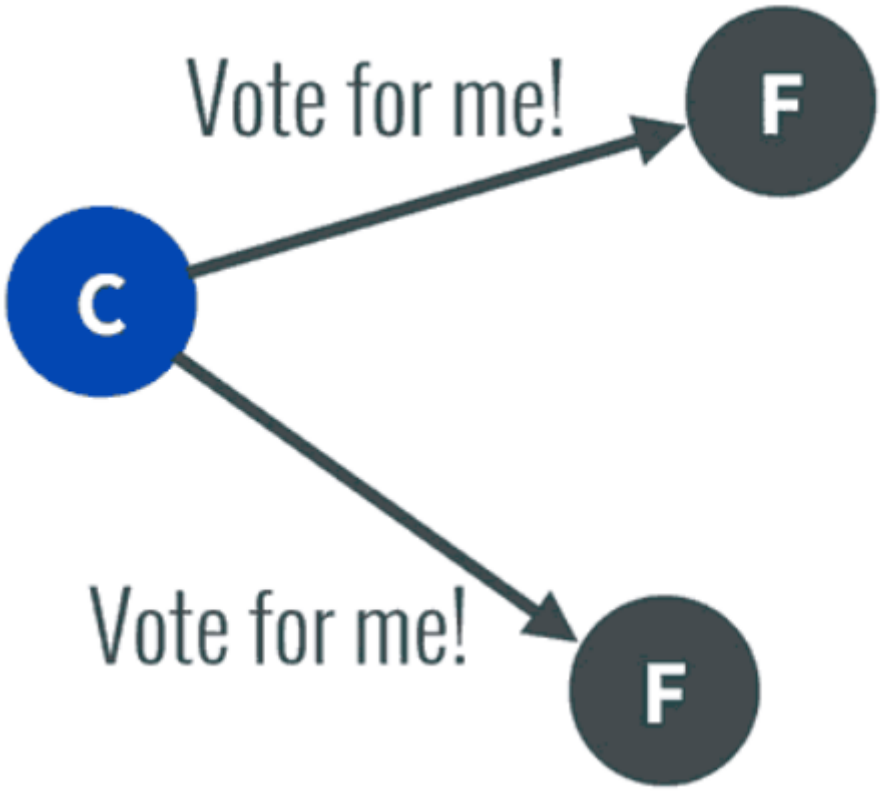
1. Leader选举主要的行为是在一个集群中选择一个leader；以及在leader挂了以后重新选举新的leader
2. 日志复制，主要行为是 leader从客户端获得命令，然后将其写到自己的log中；以及将自己的log副本分发给其他的服务器
3. 安全机制，主要是为了保证只有log为最新的结点才能成为leader。

在Raft中，任何时候一个服务器可以扮演下面角色之一：

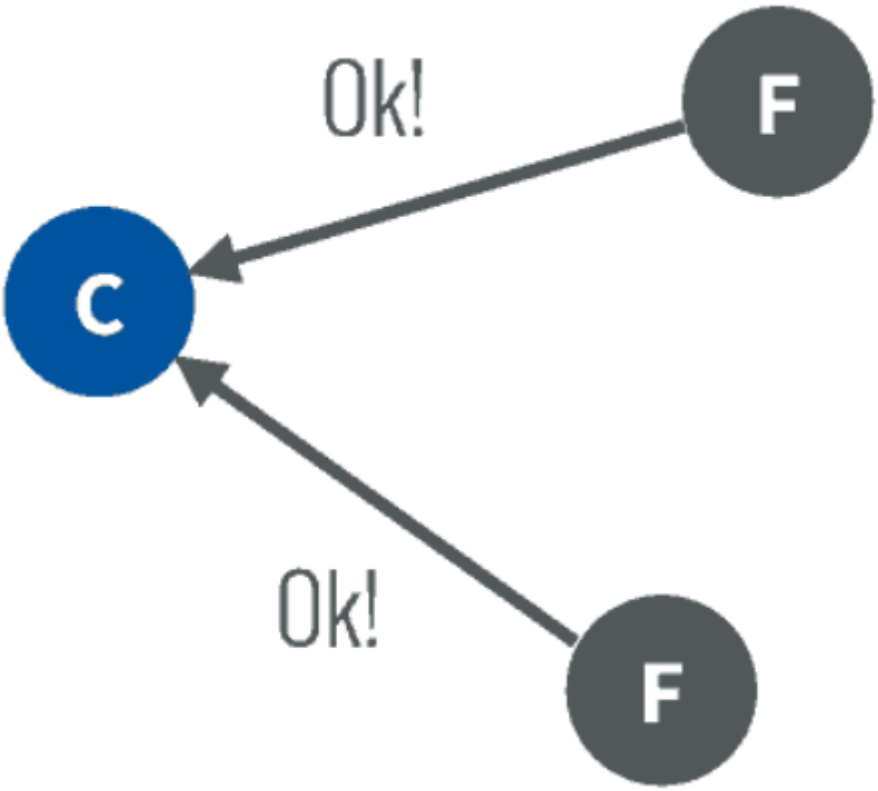
1. Leader: 处理所有客户端交互，日志复制等，一般一次只有一个Leader.
2. Follower: 类似选民，完全被动
3. Candidate候选人: 类似Proposer律师，可以被选为一个新的领导人。

Raft阶段分为两个，首先是选举过程，然后在选举出来的领导人带领进行正常操作，比如日志复制等。下面用图示展示这个过程：

\1. 任何一个服务器都可以成为一个候选者Candidate，它向其他服务器Follower发出要求选举自己的请求：

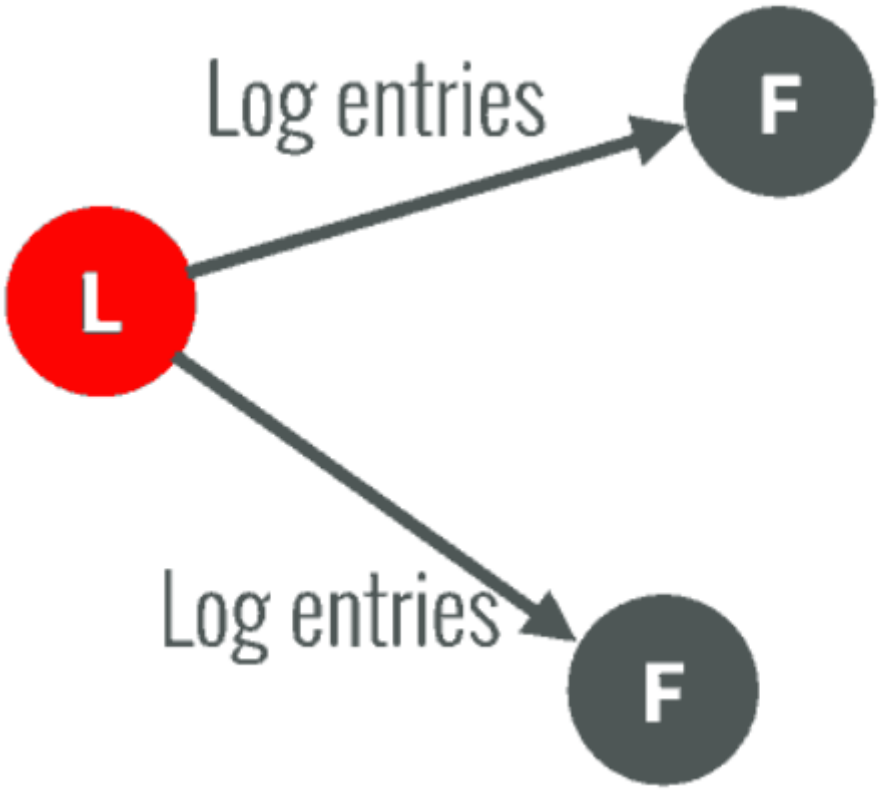


\2. 其他服务器同意了，发出OK。

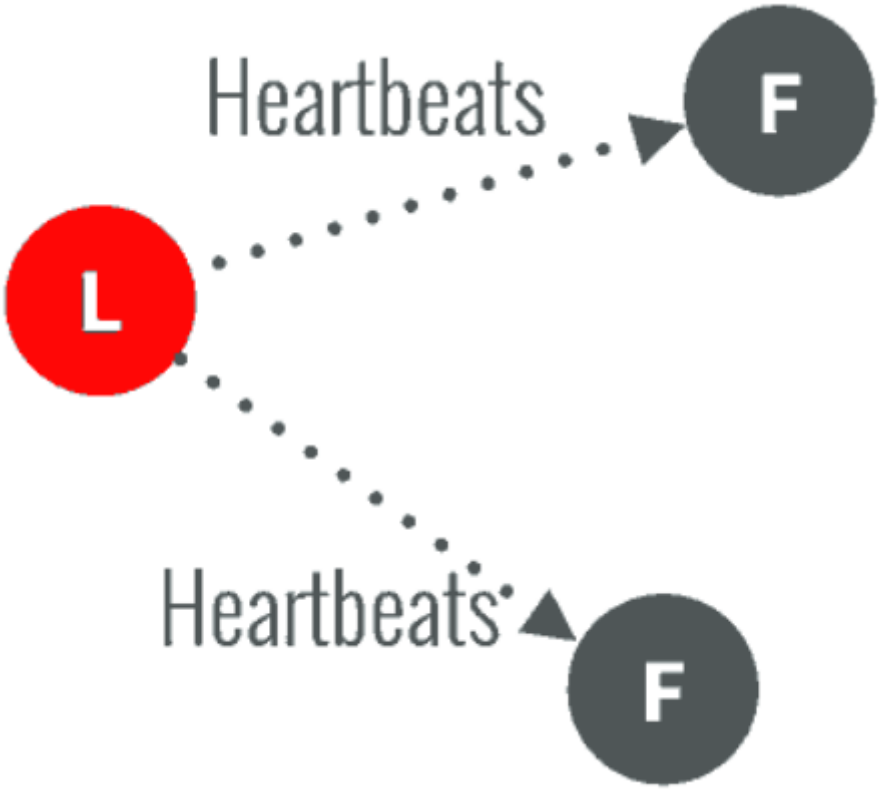


注意如果在这个过程中，有一个Follower当机，没有收到请求选举的要求，因此候选者可以自己选自己，只要达到N/2 + 1 的大多数票，候选人还是可以成为Leader的。

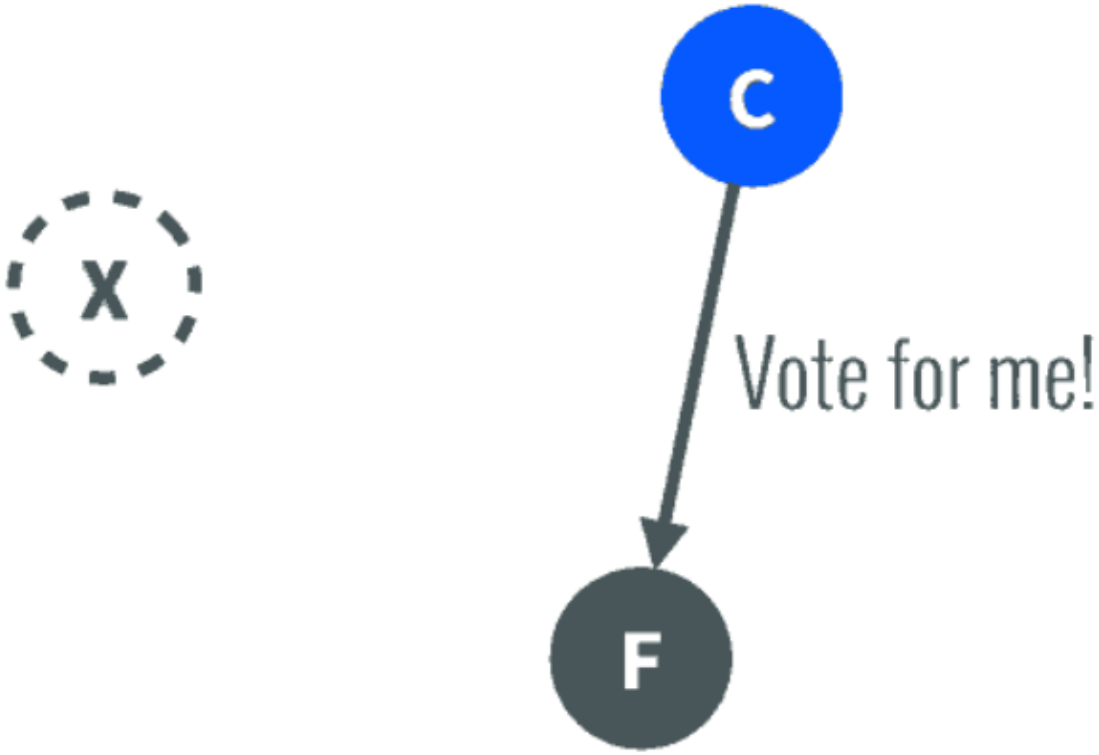
\3. 这样这个候选者就成为了Leader领导人，它可以向选民也就是Follower们发出指令，比如进行日志复制。



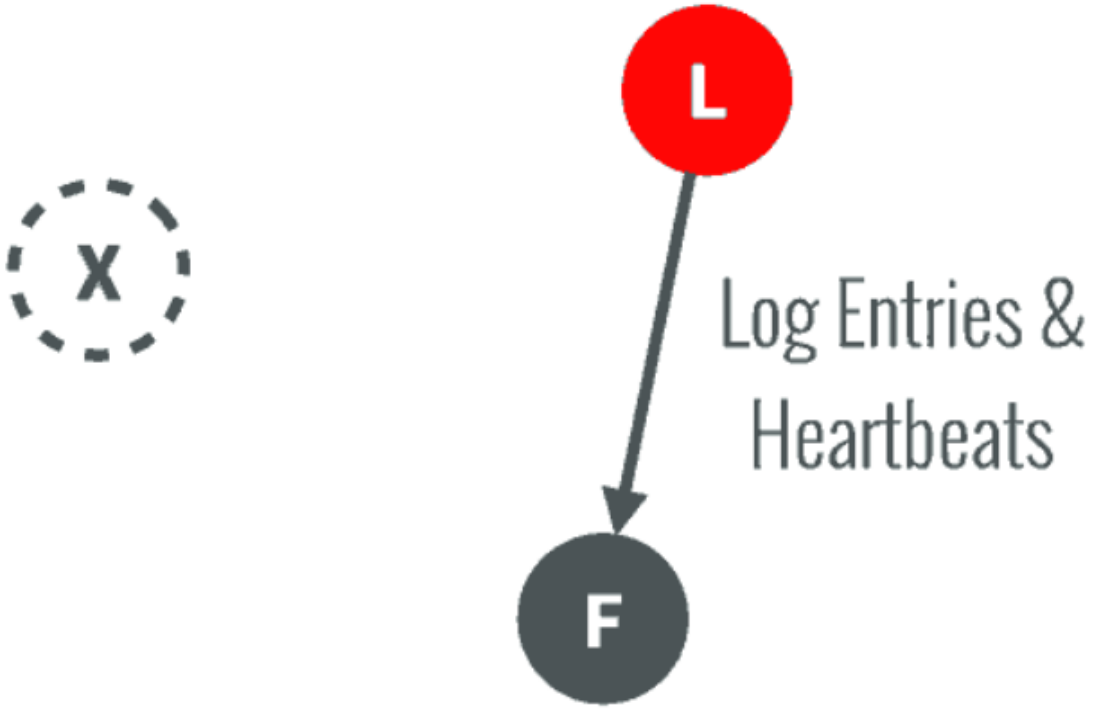
\4. 以后通过心跳进行日志复制的通知



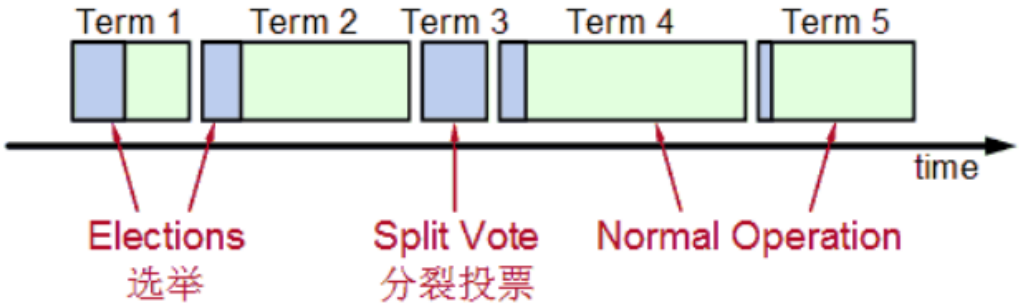
\5. 如果一旦这个Leader当机崩溃了，那么Follower中有一个成为候选者，发出邀票选举。



\6. Follower同意后，其成为Leader，继续承担日志复制等指导工作：



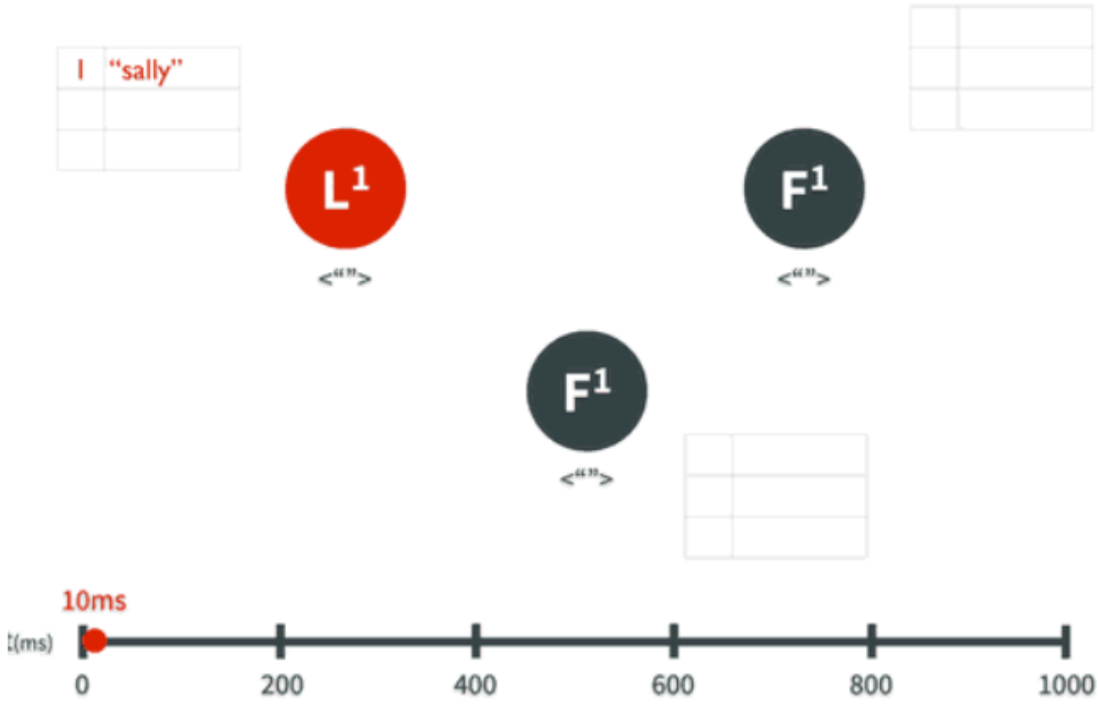
值得注意的是，整个选举过程是有一个时间限制的，如下图：



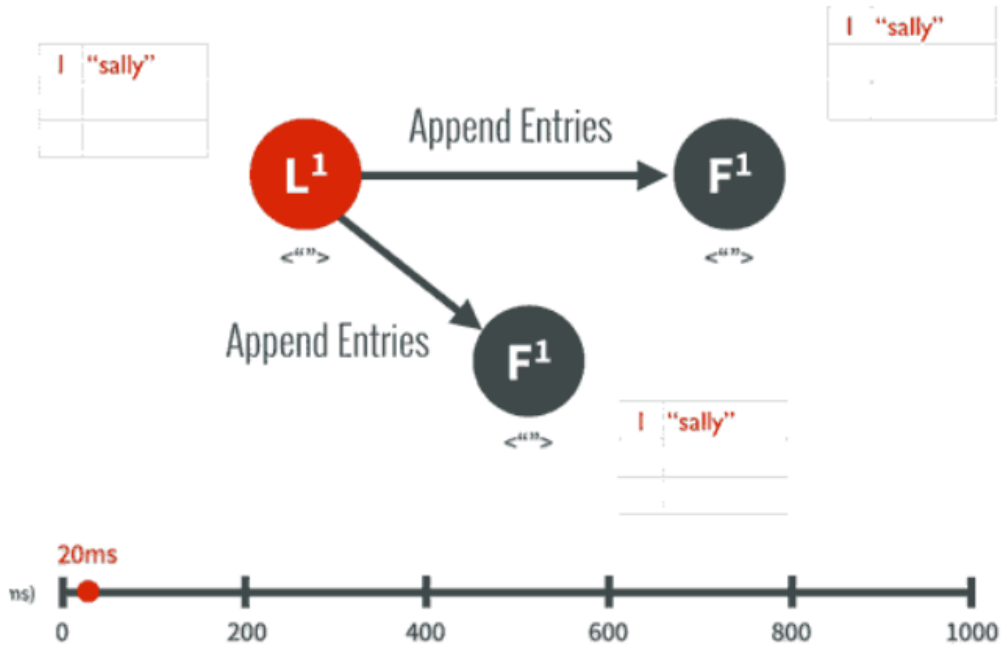
Splite Vote是因为如果同时有两个候选人向大家邀票，这时通过类似加时赛来解决，两个候选者在一段timeout比如300ms互相不服气的等待以后，因为双方得到的票数是一样的，一半对一半，那么在300ms以后，再由这两个候选者发出邀票，这时同时的概率大大降低，那么首先发出邀票的的候选者得到了大多数同意，成为领导者Leader，而另外一个候选者后来发出邀票时，那些Follower选民已经投票给第一个候选者，不能再投票给它，它就成为落选者了，最后这个落选者也成为普通Follower一员了。

日志复制

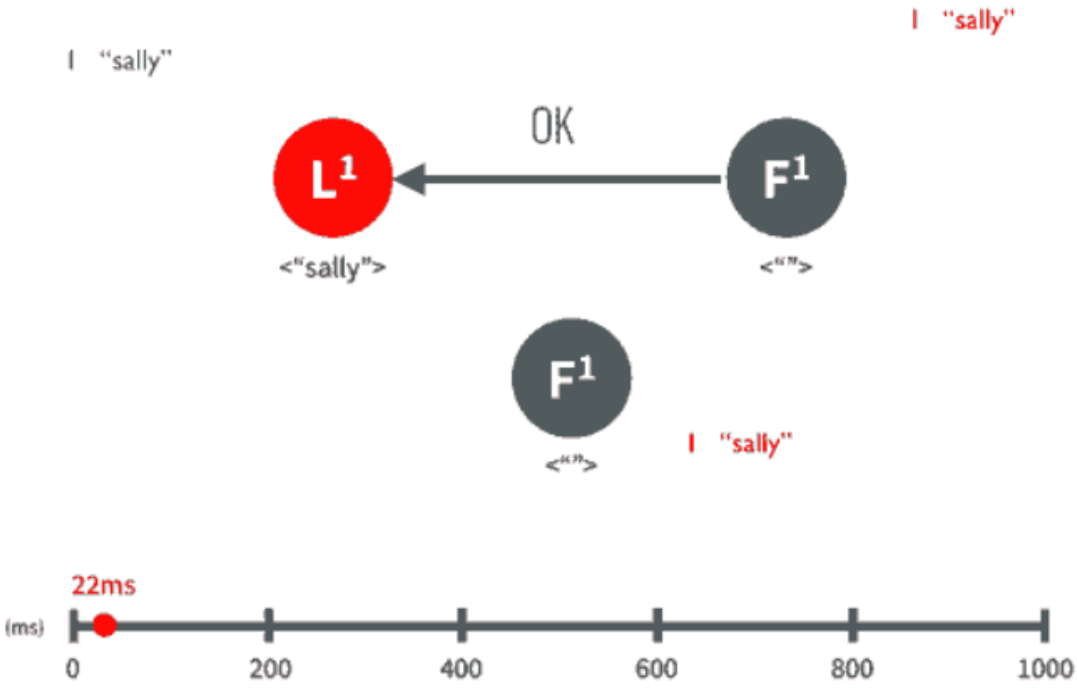
下面以日志复制为例子说明Raft算法，假设Leader领导人已经选出，这时客户端发出增加一个日志的要求，比如日志是"sally"：



\2. Leader要求Followe遵从他的指令，都将这个新的日志内容追加到他们各自日志中：



3.大多数follower服务器将日志写入磁盘文件后，确认追加成功，发出Commited Ok:



\4. 在下一个心跳heartbeat中，Leader会通知所有Follwer更新commited 项目。

对于每个新的日志记录，重复上述过程。

如果在这一过程中，发生了网络分区或者网络通信故障，使得Leader不能访问大多数Follwers了，那么Leader只能正常更新它能访问的那些Follower服务器，而大多数的服务器Follower因为没有了Leader，他们重新选举一个候选者作为Leader，然后这个Leader作为代表于外界打交道，如果外界要求其添加新的日志，这个新的Leader就按上述步骤通知大多数Followers，如果这时网络故障修复了，那么原先的Leader就变成Follower，在失联阶段这个老Leader的任何更新都不能算commit，都回滚，接受新的Leader的新的更新。

六 Gossip 协议

Gossip是什么

Gossip协议是一个通信协议，一种传播消息的方式，灵感来自于：瘟疫、社交网络等。使用Gossip协议的有：Redis Cluster、Consul、Apache Cassandra等。

六度分隔理论

说到社交网络，就不得不提著名的**六度分隔理论**。1967年，哈佛大学的心理学教授Stanley Milgram想要描绘一个连结人与社区的人际连系网。做过一次连锁信实验，结果发现了"六度分隔"现象。简单地说："你和任何一个陌生人之间所间隔的人不会超过六个，也就是说，最多通过六个人你就能够认识任何一个陌生人。

数学解释该理论：若每个人平均认识260人，其六度就是260↑6 =1,188,137,600,000。消除一些节点重复，那也几乎**覆盖**了整个地球人口若干多多倍，这也是Gossip协议的雏形。

原理

Gossip协议基本思想就是：一个节点想要分享一些信息给网络中的其他的一些节点。于是，它**周期性**的**随机**选择一些节点，并把信息传递给这些节点。这些收到信息的节点接下来会做同样的事情，即把这些信息传递给其他一些随机选择的节点。一般而言，信息会周期性的传递给N个目标节点，而不只是一个。这个N被称为**fanout**（这个单词的本意是扇出）。

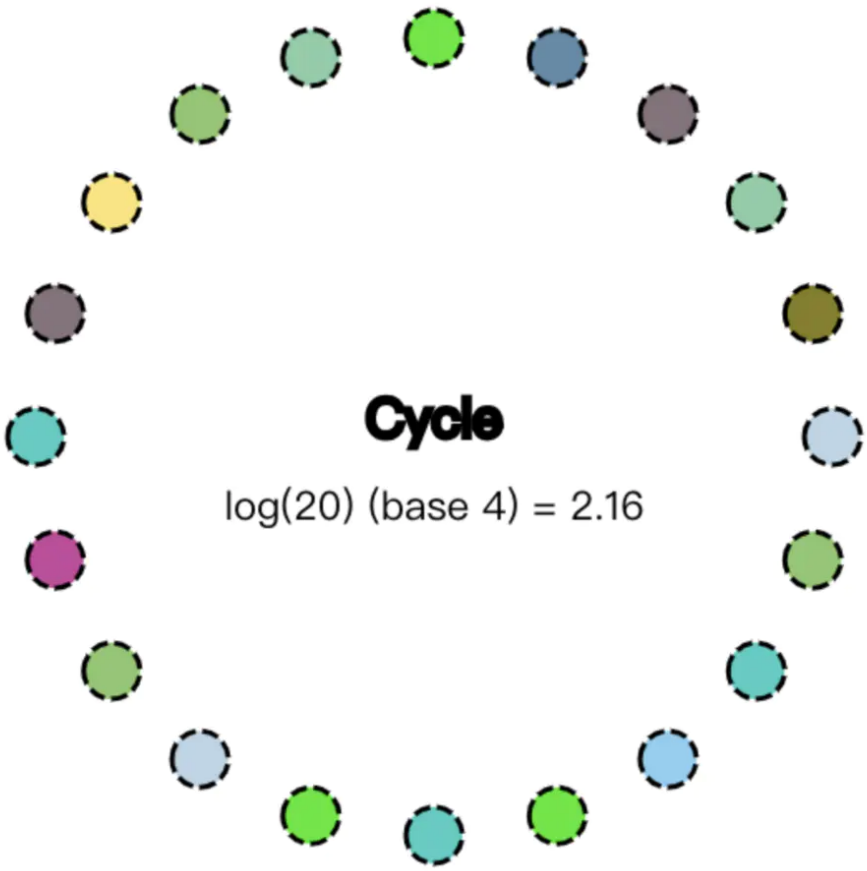
用途

Gossip协议的主要用途就是**信息传播和扩散**：即把一些发生的事件传播到全世界。它们也被用于数据库复制，信息扩散，集群成员身份确认，故障探测等。

基于Gossip协议的一些有名的系统：Apache Cassandra，Redis（Cluster模式），Consul等。

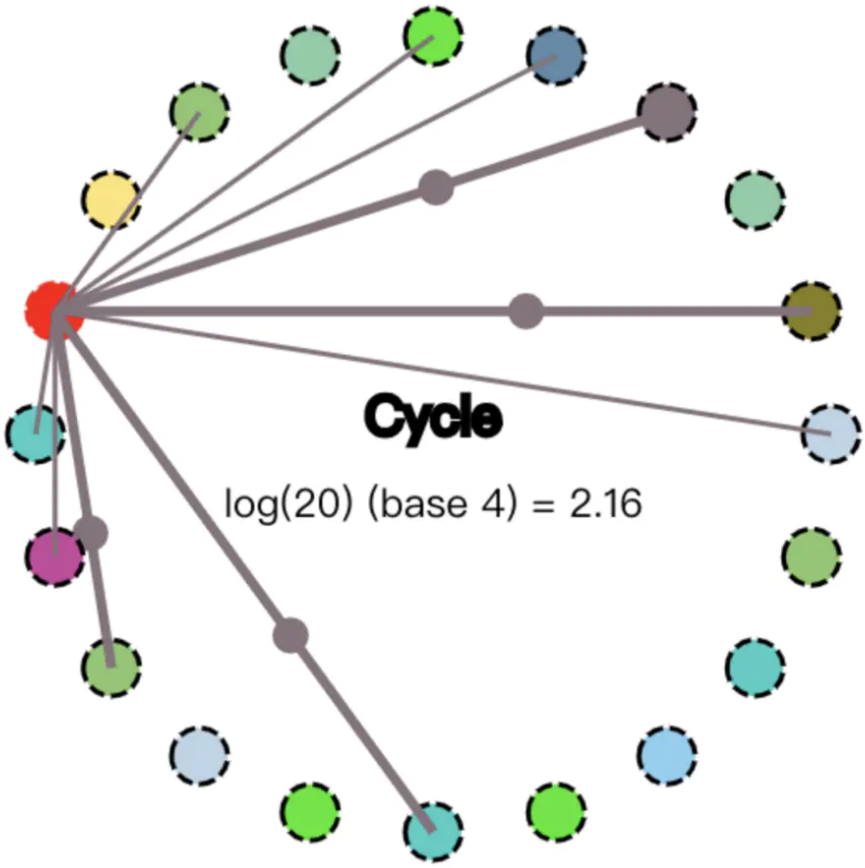
图解

接下来通过多张图片剖析Gossip协议是如何运行的。如下图所示，Gossip协议是周期循环执行的。图中的公式表示Gossip协议把信息传播到每一个节点需要多少次循环动作，需要说明的是，公式中的20表示整个集群有20个节点，4表示某个节点会向4个目标节点传播消息：



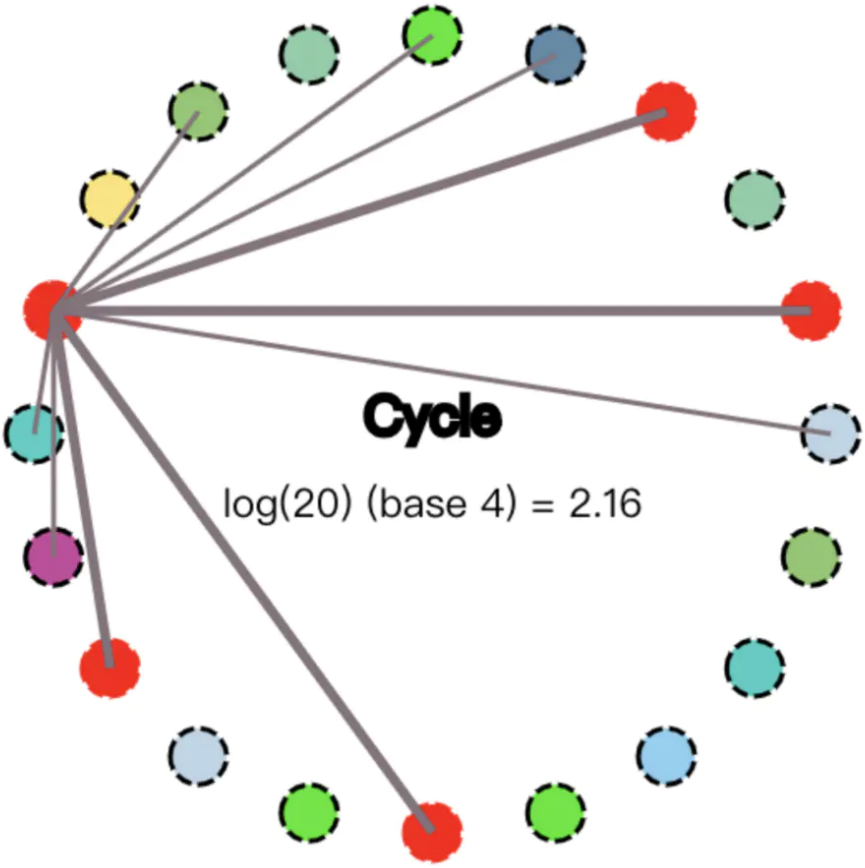
Gossip Protocol

如下图所示，红色的节点表示其已经“受到感染”，即接下来要传播信息的源头，连线表示这个初始化感染的节点能正常连接的节点（其不能连接的节点只能靠接下来感染的节点向其传播消息）。并且N等于4，我们假设4根较粗的线路，就是它第一次传播消息的线路：



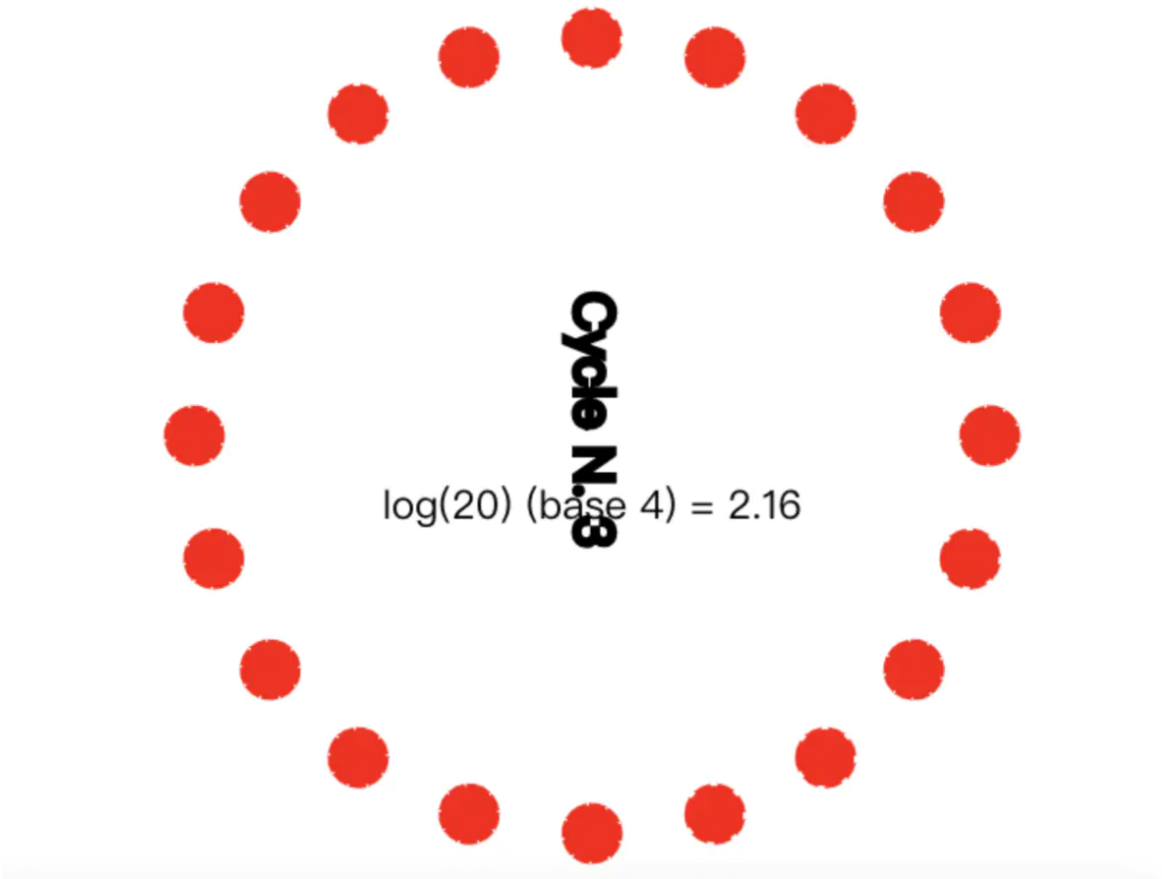
first infected node

第一次消息完成传播后，新增了4个节点会被“感染”，即这4个节点也收到了消息。这时候，总计有5个节点变成红色：



infected nodes

那么在下一次传播周期时，总计有5个节点，且这5个节点每个节点都会向4个节点传播消息。最后，经过3次循环，20个节点全部被感染（都变成红色节点），即说明需要传播的消息已经传播给了所有节点：



infected all nodes

需要说明的是，20个节点且设置fanout=4，公式结果是2.16，这只是个近似值。真实传递时，可能需要3次甚至4次循环才能让所有节点收到消息。这是因为每个节点在传播消息的时候，是随机选择N个节点的，这样的话，就有可能某个节点会被选中2次甚至更多次。

发送消息

由前面对Gossip协议图解分析可知，节点传播消息是周期性的，并且**每个节点有它自己的周期**。另外，节点发送消息时的**目标节点数**由参数fanout决定。至于往哪些目标节点发送，则是**随机**的。

一旦消息被发送到目标节点，那么目标节点也会被感染。一旦某个节点被感染，那么它也会向其他节点传播消息，试图感染更多的节点。最终，每一个节点都会被感染，即消息被同步给了所有节点：

可扩展性

Gossip协议是可扩展的，因为它只需要O(logN) 个周期就能把消息传播给所有节点。某个节点在往固定数量节点传播消息过程中，并不需要等待确认（ack），并且，即使某条消息传播过程中丢失，它也不需要做任何补偿措施。大哥比方，某个节点本来需要将消息传播给4个节点，但是由于网络或者其他原因，只有3个消息接收到消息，即使这样，这对最终所有节点接收到消息是没有任何影响的。

如下表格所示，假定fanout=4，那么在节点数分别是20、40、80、160时，消息传播到所有节点需要的循环次数对比，在节点成倍扩大的情况下，循环次数并没有增加很多。所以，Gossip协议具备可扩展性：

节点数	20	40	80	160	320
循环次数	2.16	2.66	3.16	3.44	4.16

可扩展性

失败容错

Gossip也具备失败容错的能力，即使网络故障等一些问题，Gossip协议依然能很好的运行。因为一个节点会**多次**分享某个需要传播的信息，即使不能连通某个节点，其他被感染的节点也会尝试向这个节点传播信息。

健壮性

Gossip协议下，没有任何扮演特殊角色的节点（比如leader等）。任何一个节点无论什么时候下线或者加入，并不会破坏整个系统的服务质量。

然而，Gossip协议也有不完美的地方，例如，**拜占庭**问题（Byzantine）。即，如果有一个恶意传播消息的节点，Gossip协议的分布式系统就会出问题。