微服务架构

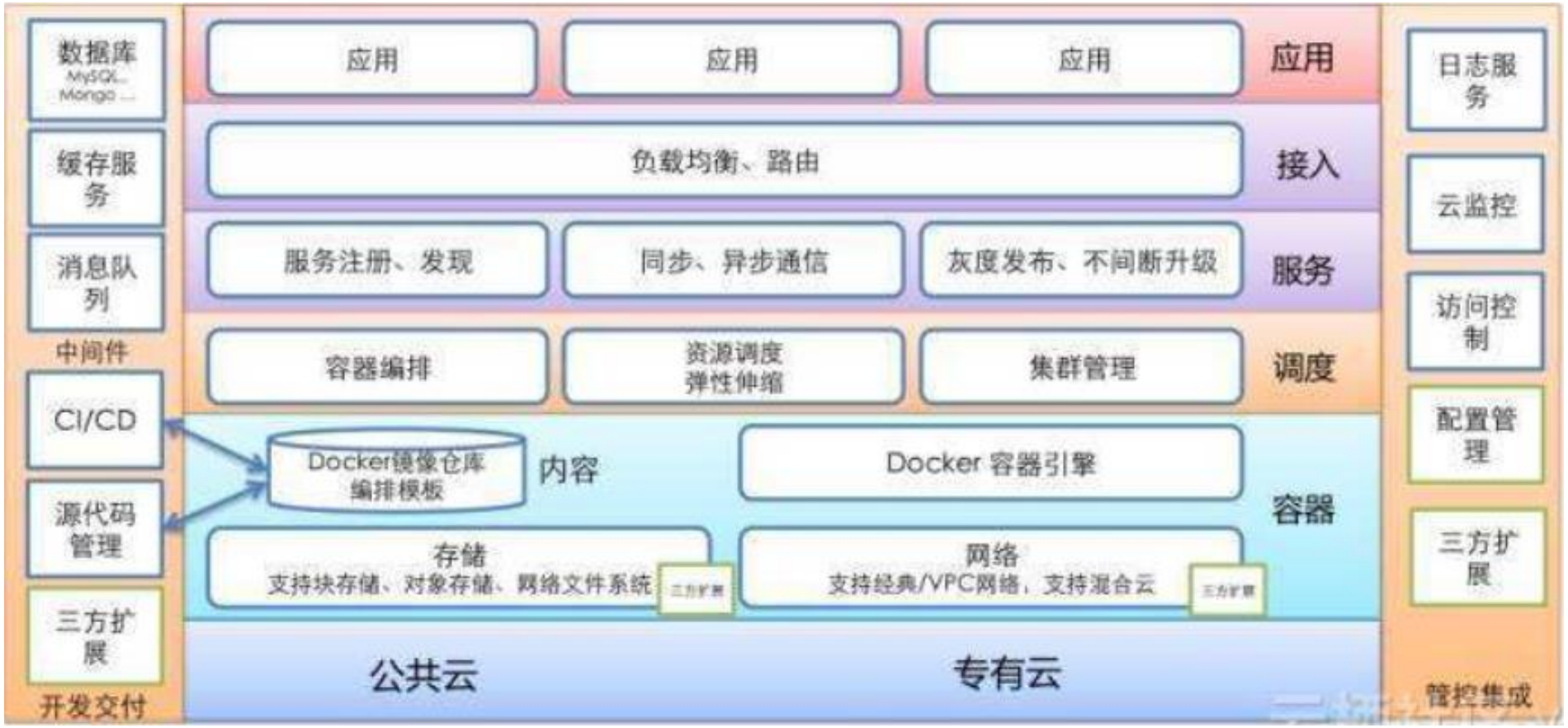
联系QQ：2816010068，加入会员群

二叉树出品

目录

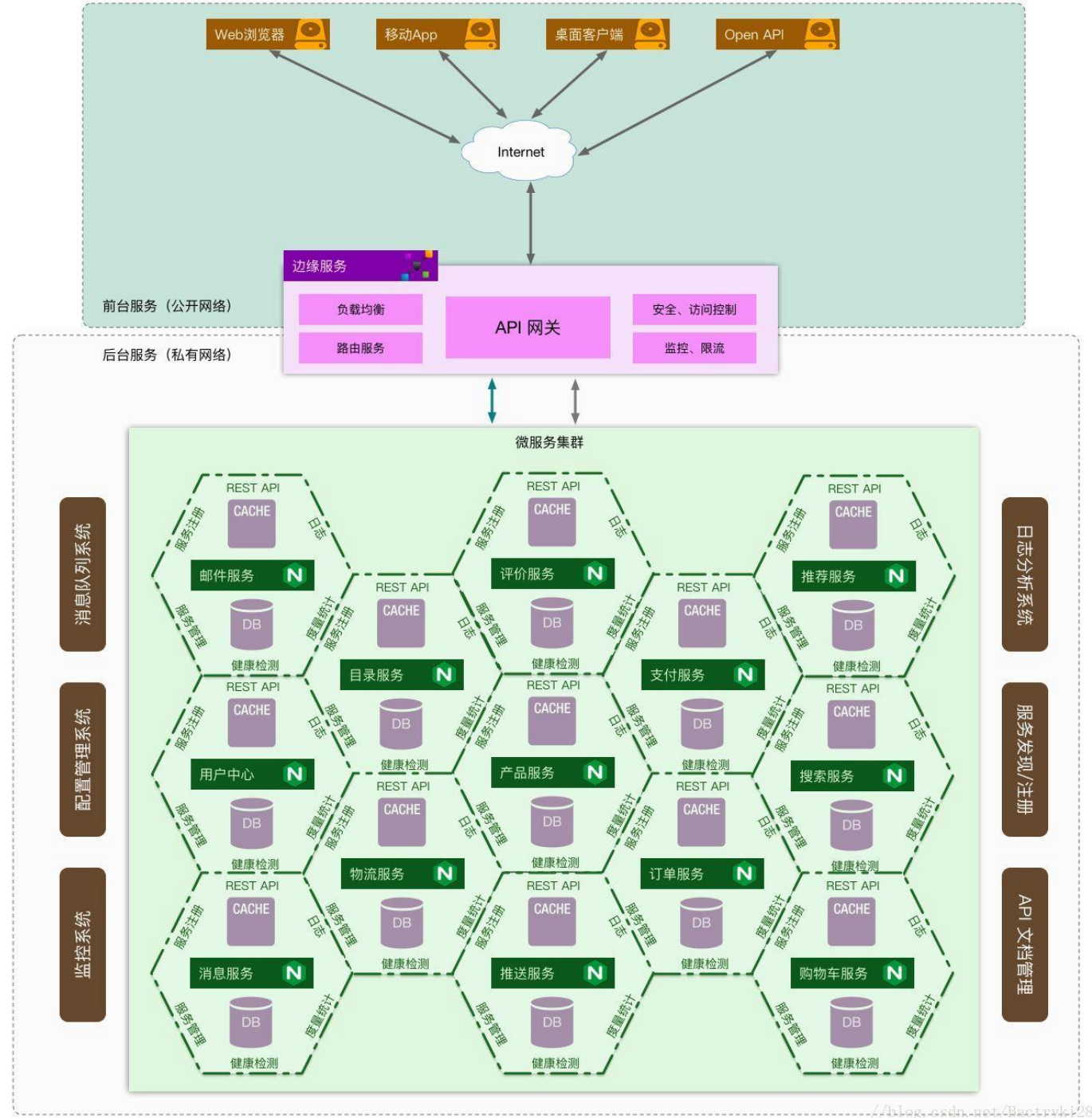
- 微服务架构全景图
- 服务注册和发现
- Raft协议详解
- Rpc调用
- 服务监控和追踪

微服务架构全景图



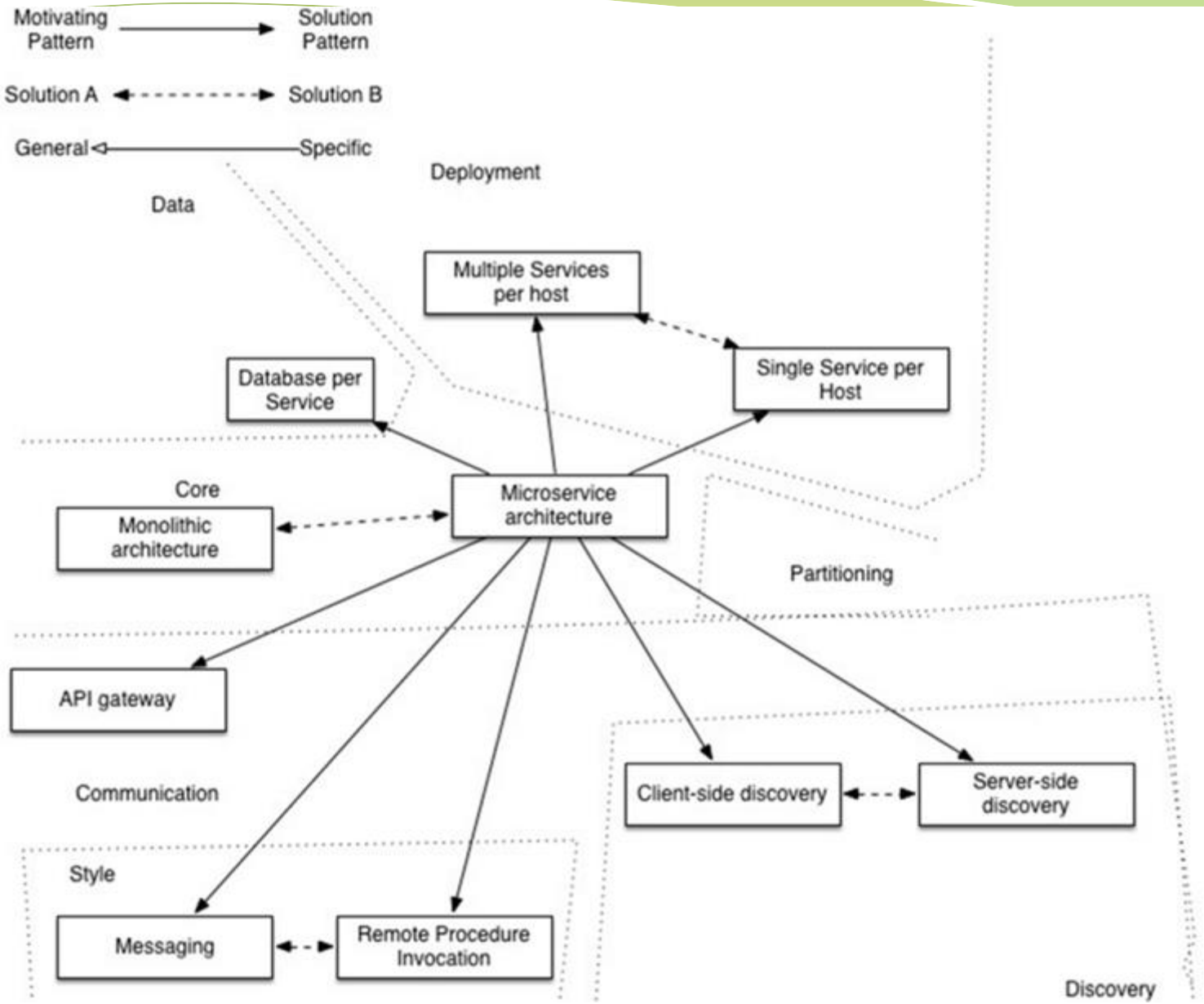
2019年5月12日

二叉树出品



2019年5月12日

二叉树出品

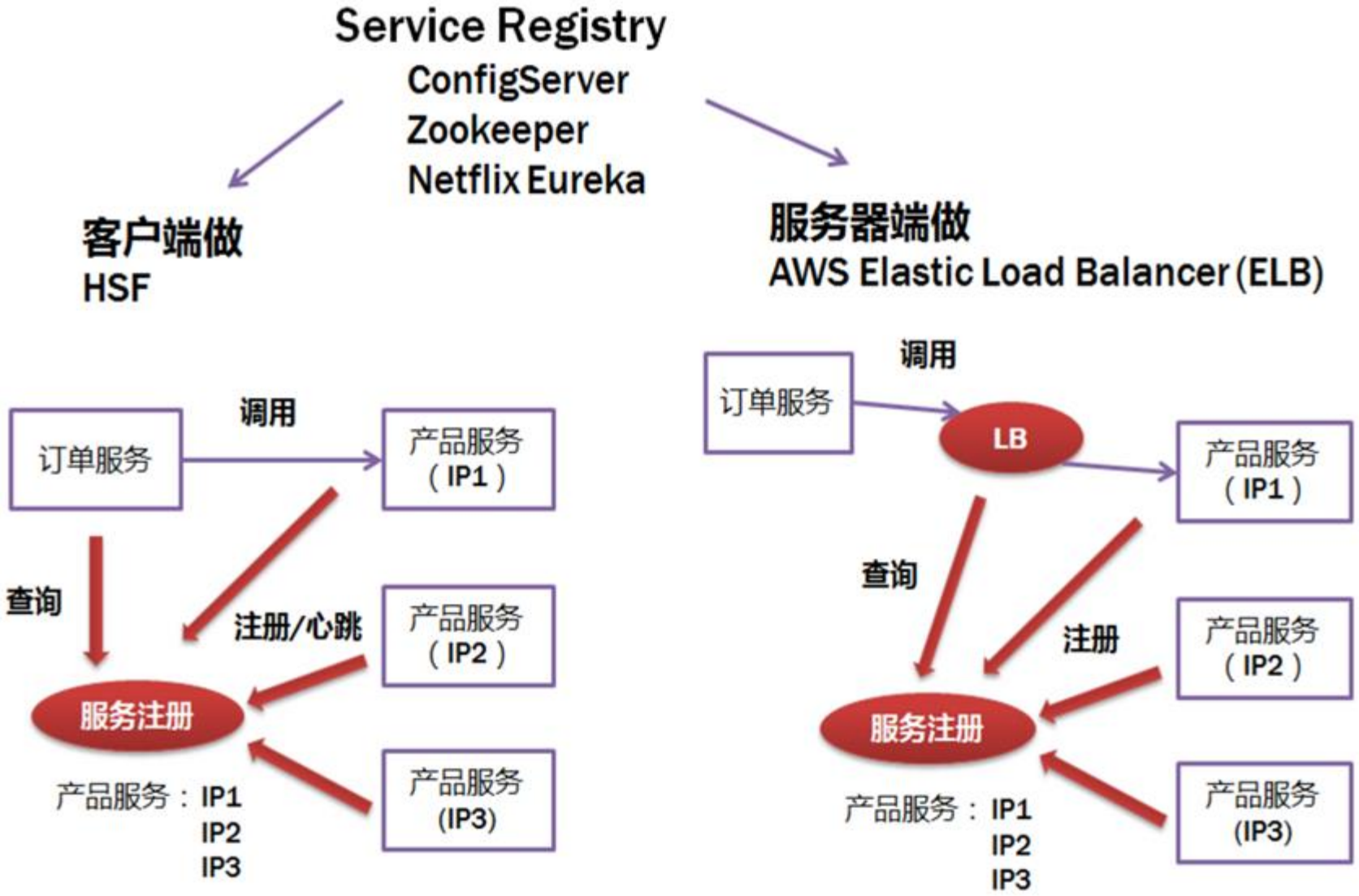

Motivating Pattern
Solution Pattern
Solution A
Solution B
General
Specific
Deployment
Data
Multiple Services per host
Single Service per Host
Database per Service
Core
Monolithic architecture
Microservice architecture
Partitioning
API gateway
Communication
Client-side discovery
Server-side discovery
Style
Messaging
Remote Procedure Invocation
Discovery

服务注册和发现

- Client side implement
  - 调用需要维护所有调用服务的地址
  - 有一定的技术难度，需要rpc框架支持

- Server side implement
  - 架构简单
  - 有单点故障

- **问题：在微服务架构中，为什么不选择传统的LVS方案？**


Service Registry
ConfigServer
Zookeeper
Netflix Eureka
客户端做
HSF
服务器端做
AWS Elastic Load Balancer (ELB)
订单服务
调用
产品服务
(IP1)
查询
注册/心跳
产品服务
(IP2)
服务注册
产品服务
(IP3)
产品服务： IP1
IP2
IP3
订单服务
调用
LB
产品服务
(IP1)
查询
注册
产品服务
(IP2)
服务注册
产品服务
(IP3)
产品服务： IP1
IP2
IP3

注册中心

- etcd注册中心
  - 分布式一致性系统
  - 基于raft一致性协议
- etcd使用场景
  - 服务注册和发现
  - 共享配置
  - 分布式锁
  - Leader选举
- 服务注册和发现的思考
  - 一定需要一致性么？
  - Eureka 是一个AP系统，netflix开发，用来做高可用的注册中心

Raft协议详解

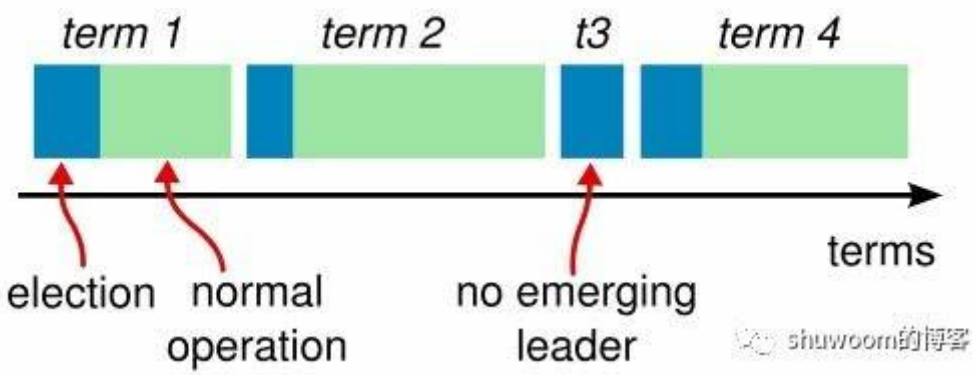
- 应用场景
  - 解决分布式系统一致性的问题
  - 基于复制的

- 工作机制
  - leader选举
  - 日志复制
  - 安全性

基本概念

- 角色
  - Follower角色
  - Leader角色
  - Candicate角色

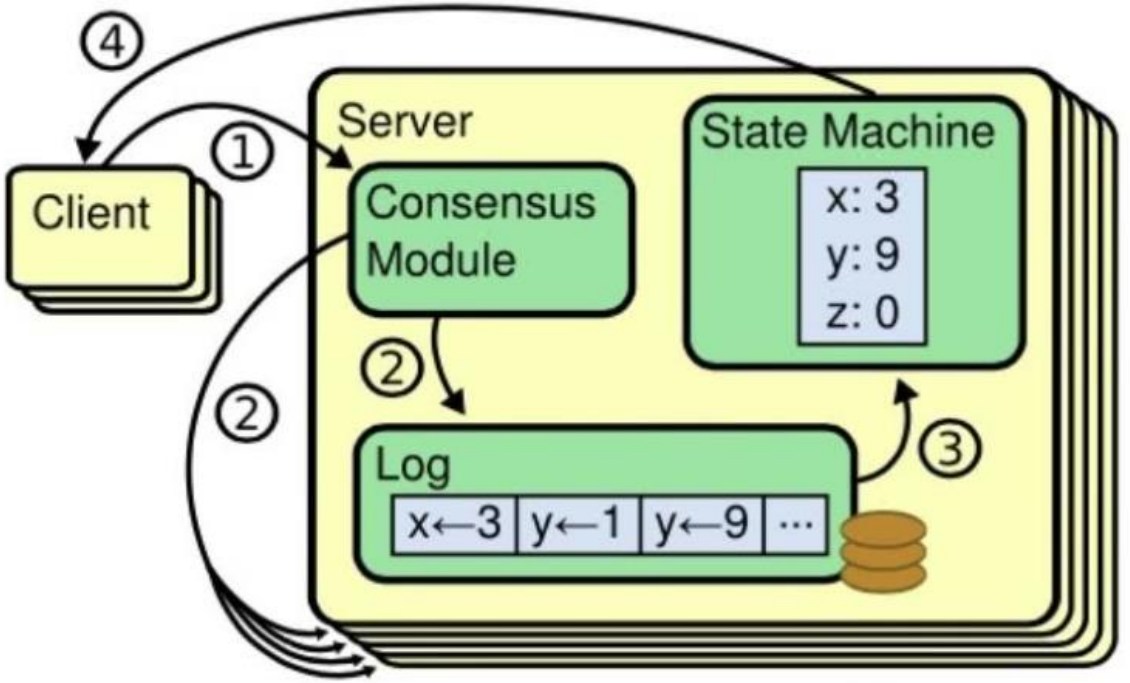
- Term（任期）概念
  - 在raft协议中，将时间分成一个个term（任期）



基本概念

- 复制状态机

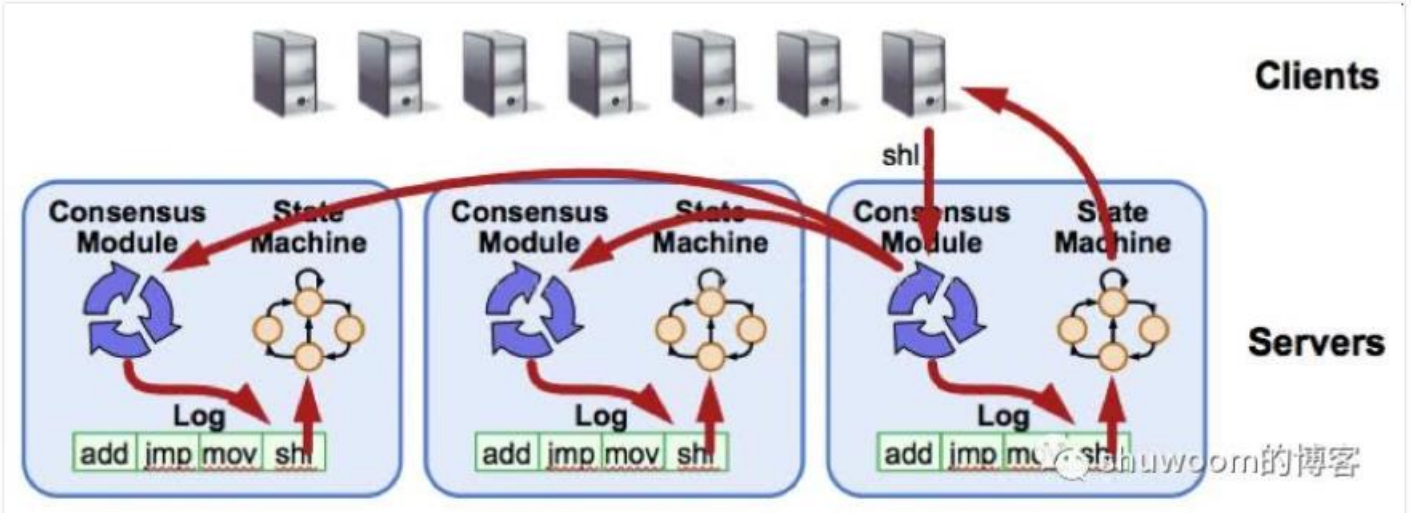
我们知道，在一个分布式系统数据库中，如果每个节点的状态一致，每个节点都执行相同的命令序列，那么最终他们会得到一个一致的状态。也就是和说，为了保证整个分布式系统的一致性，我们需要保证每个节点执行相同的命令序列，也就是说每个节点的日志要保持一样。所以说，保证日志复制一致就是Raft等一致性算法的工作了。



基本概念

- 复制状态机

这里就涉及Replicated State Machine（复制状态机），如上图所示。在一个节点上，一致性模块（Consensus Module，也就是分布式共识算法）接收到了来自客户端的命令。然后把接收到的命令写入到日志中，该节点和其他节点通过一致性模块进行通信确保每个日志最终包含相同的命令序列。一旦这些日志的命令被正确复制，每个节点的状态机（State Machine）都会按照相同的序列去执行他们，从而最终得到一致的状态。然后将达成共识的结果返回给客户端，如下图所示。



基本概念

- 心跳（heartbeats）和超时机制（timeout）

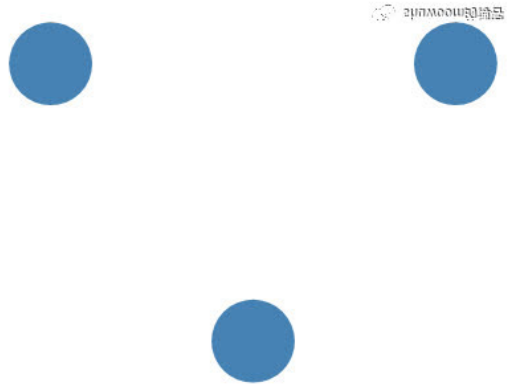
在Raft算法中，有两个timeout机制来控制领导人选举：

一个是选举定时器（eletion timeout）：即Follower等待成为Candidate状态的等待时间，这个时间被随机设定为150ms~300ms之间

另一个是headrbeat timeout：在某个节点成为Leader以后，它会发送Append Entries消息给其他节点，这些消息就是通过heartbeat timeout来传送，Follower接收到Leader的心跳包的同时也重置选举定时器。

Leader选举

- **触发条件：**
  - 一般情况下，追随者接到领导者的心跳时，把选举定时器清零，不会触发
  - 追随者的选举定时器超时发生时（比如leader故障了），会变成候选者，触发领导人选取
- 选举过程
  - 一开始，所有节点都是以Follower角色启动，同时启动选举定时器（时间随机，降低冲突概率）



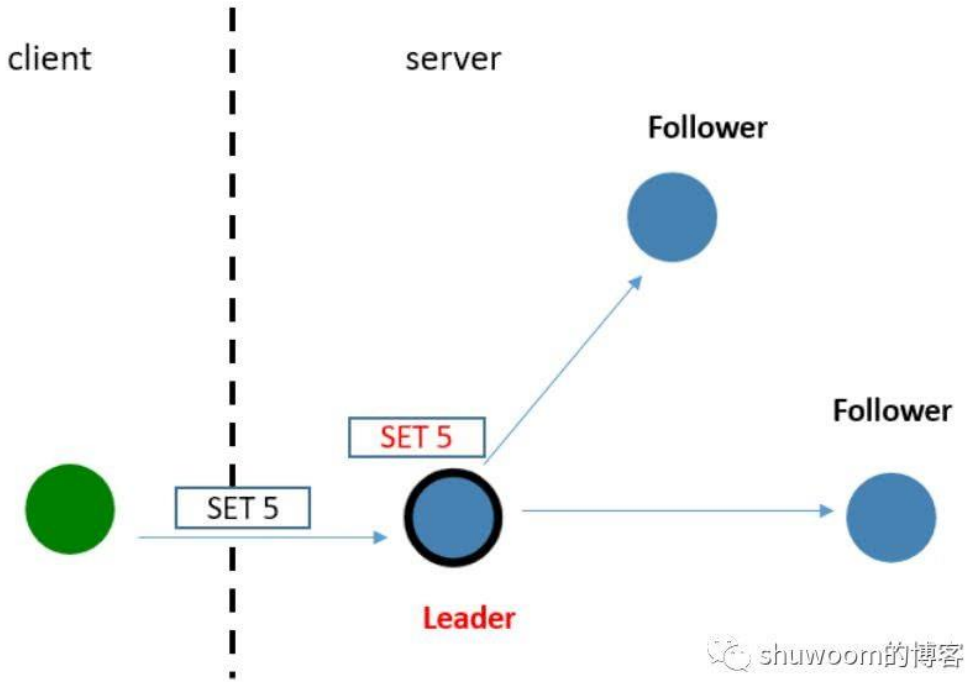
leader选举

- 选举过程
  - 当定时器到期时，转为candidate角色
  - 把当前任期加+1，并为自己进行投票
  - 发起RequestVote的RPC请求，要求其他的节点为自己投票
  - 如果得到半数以上节点的同意，就成为Leader（Leader）。
  - 如果选举超时，还没有Leader选出，则进入下一任期，重新选举。

- 限制条件
  - 每个节点在一个任期内，最多能给一个候选人投票，采用先到先服务原则
  - 如果没有投过票， 则对比candidate的log和当前节点的log哪个更新，比较方式为***谁的lastLog的term越大谁越新，如果term相同，谁的lastLog的index越大谁越新。如果当前节点更新，则拒绝投票。***

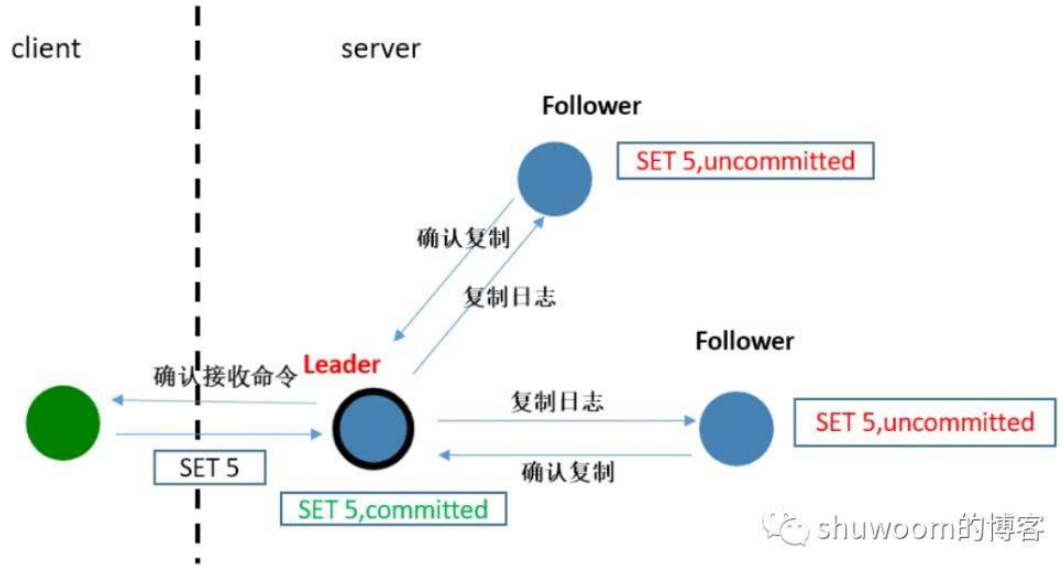
日志复制

- Client向Leader提交指令（如：SET 5），Leader收到命令后，将命令追加到本地日志中。此时，这个命令处于“uncomitted”状态，复制状态机不会执行该命令。



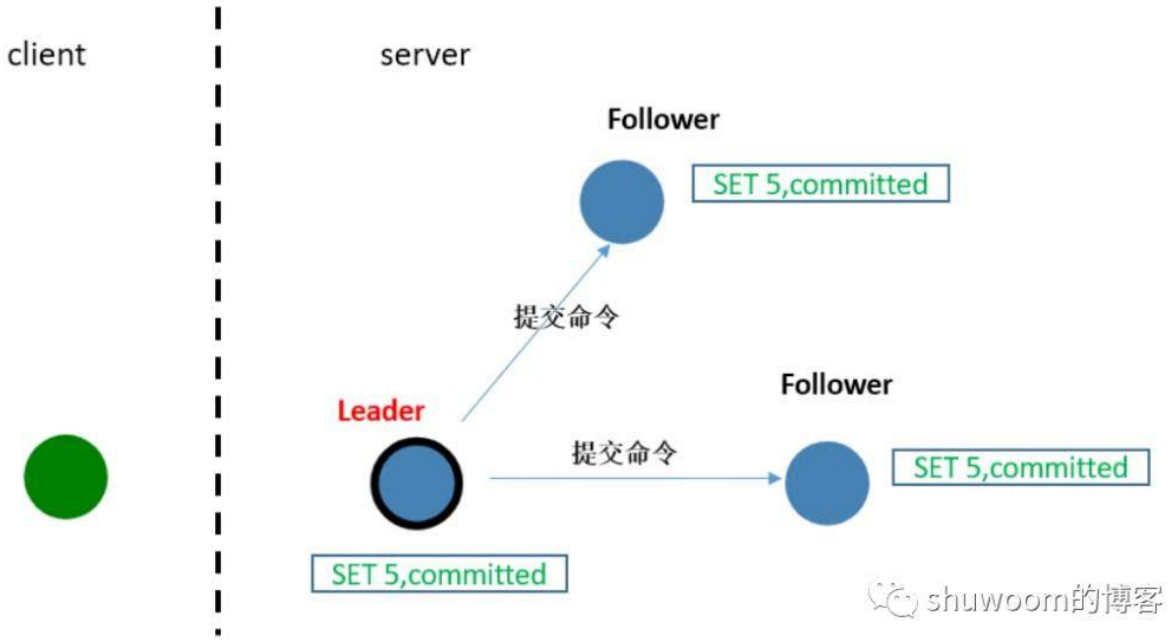
日志复制

- Leader将命令（SET 5）并发复制给其他节点，并等待其他其他节点将命令写入到日志中，如果此时有些节点失败或者比较慢，Leader节点会一直重试，知道所有节点都保存了命令到日志中。之后Leader节点就提交命令（即被状态机执行命令，这里是：SET 5），并将结果返回给Client节点。



日志复制

- Leader节点在提交命令后，下一次的心跳包中就带有通知其他节点提交命令的消息，其他节点收到Leader的消息后，就将命令应用到状态机中（State Machine），最终每个节点的日志都保持了一致性。



安全性

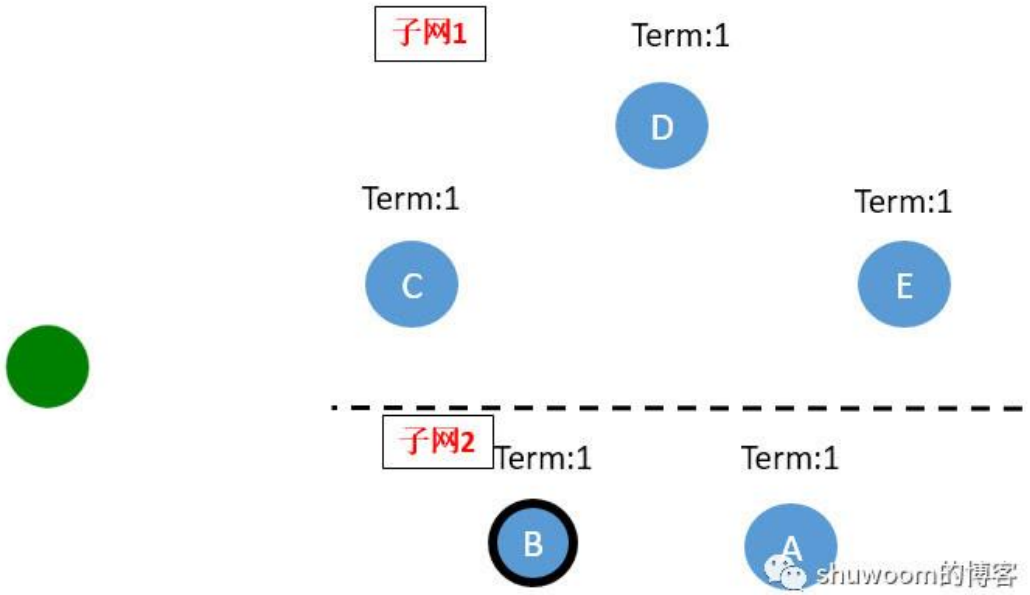
- 一个Candidate节点要成为赢得选举，就需要跟网络中大部分节点进行通信，这就意味着每一条已经提交的日志条目最少在其中一台服务器上出现。如果候选人的日志至少和大多数服务器上的日志一样新，那么它一定包含有全部的已经提交的日志条目。RequestVote RPC 实现了这个限制：这个 RPC包括候选人的日志信息，如果它自己的日志比候选人的日志要新，那么它会拒绝候选人的投票请求。

安全性

- 最新判断标准
  - 如果两个日志的任期号不同，任期号大的日志内容更新
  - 如果任期号一样大，则根据日志中最后一个命令的索引（index），谁大谁最新

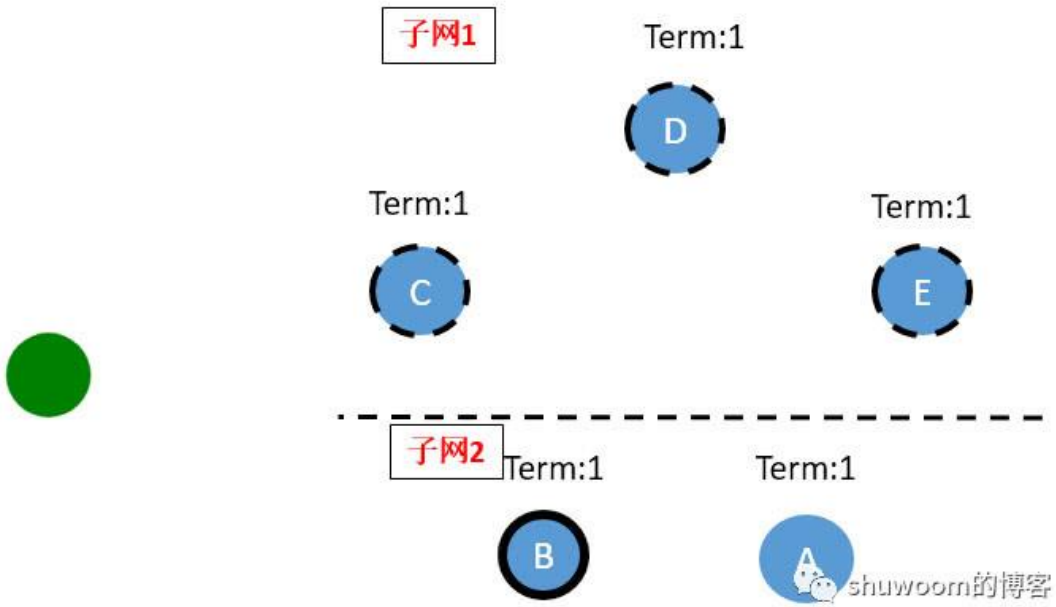
安全性

- 实例分析



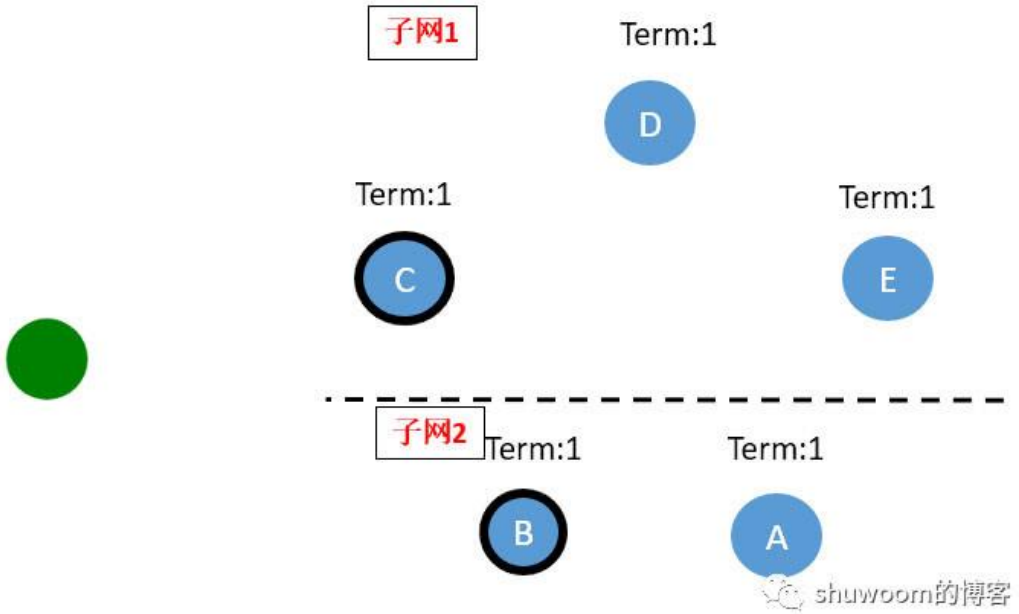
安全性

- 实例分析



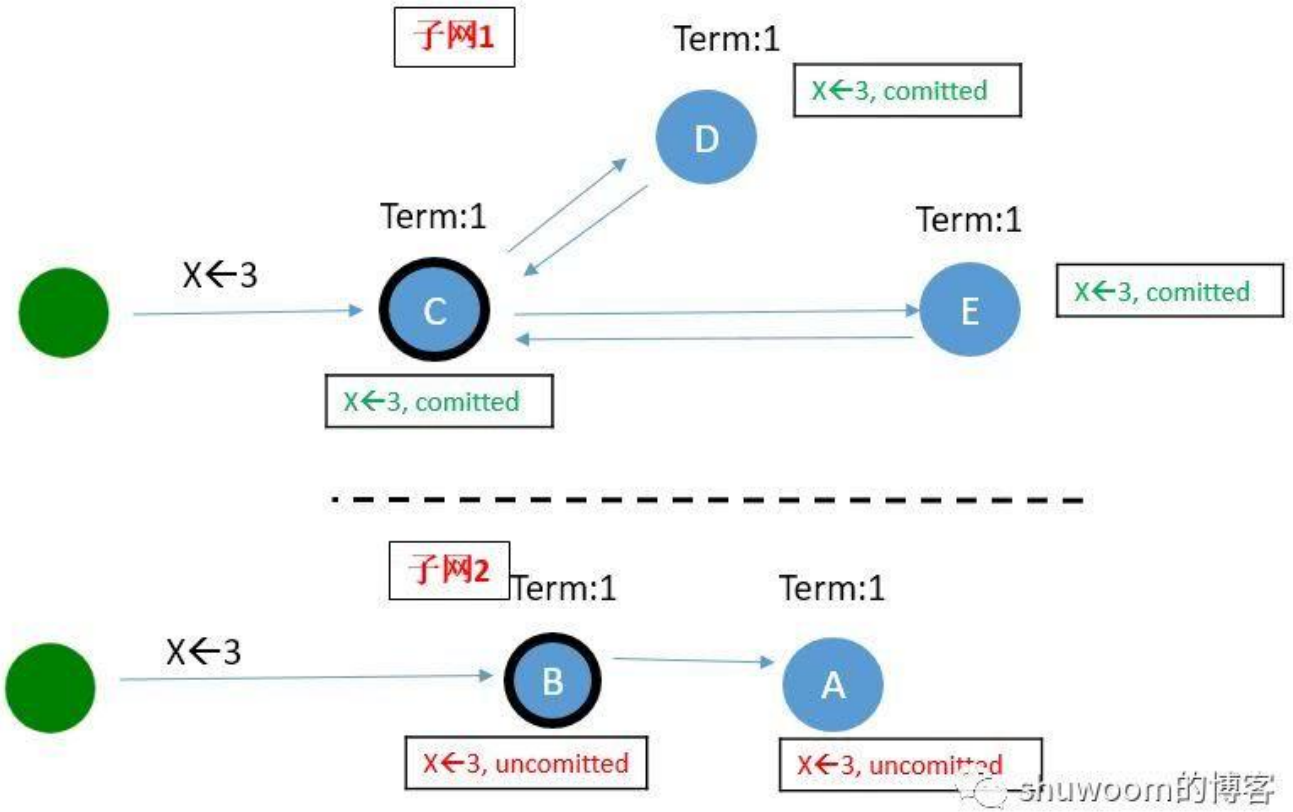
安全性

- 实例分析



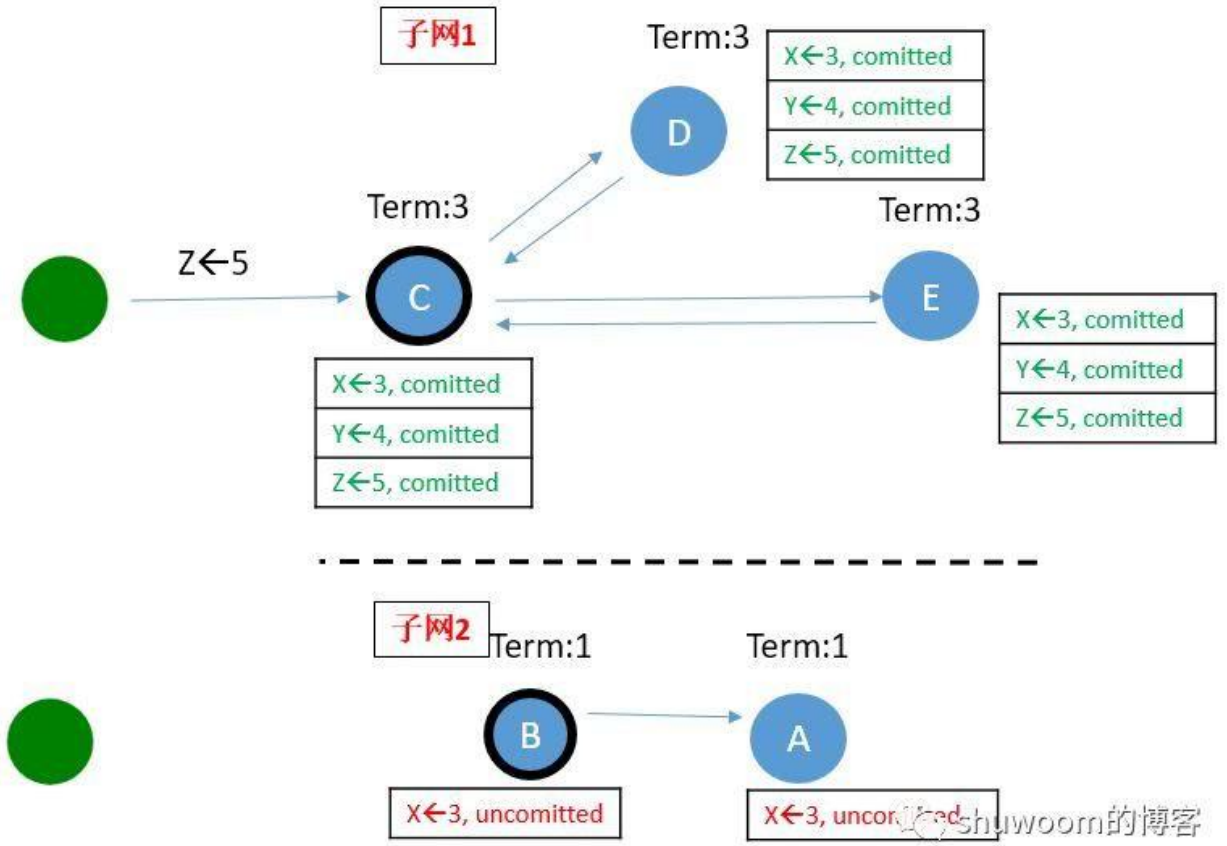
安全性

- 实例分析


子网1
Term:1
X←3, comitted
D
Term:1
Term:1
X←3
C
E
X←3, comitted
X←3, comitted
子网2
Term:1
Term:1
X←3
B
A
X←3, uncomitted
X←3, uncomitted
shuwoom的博客

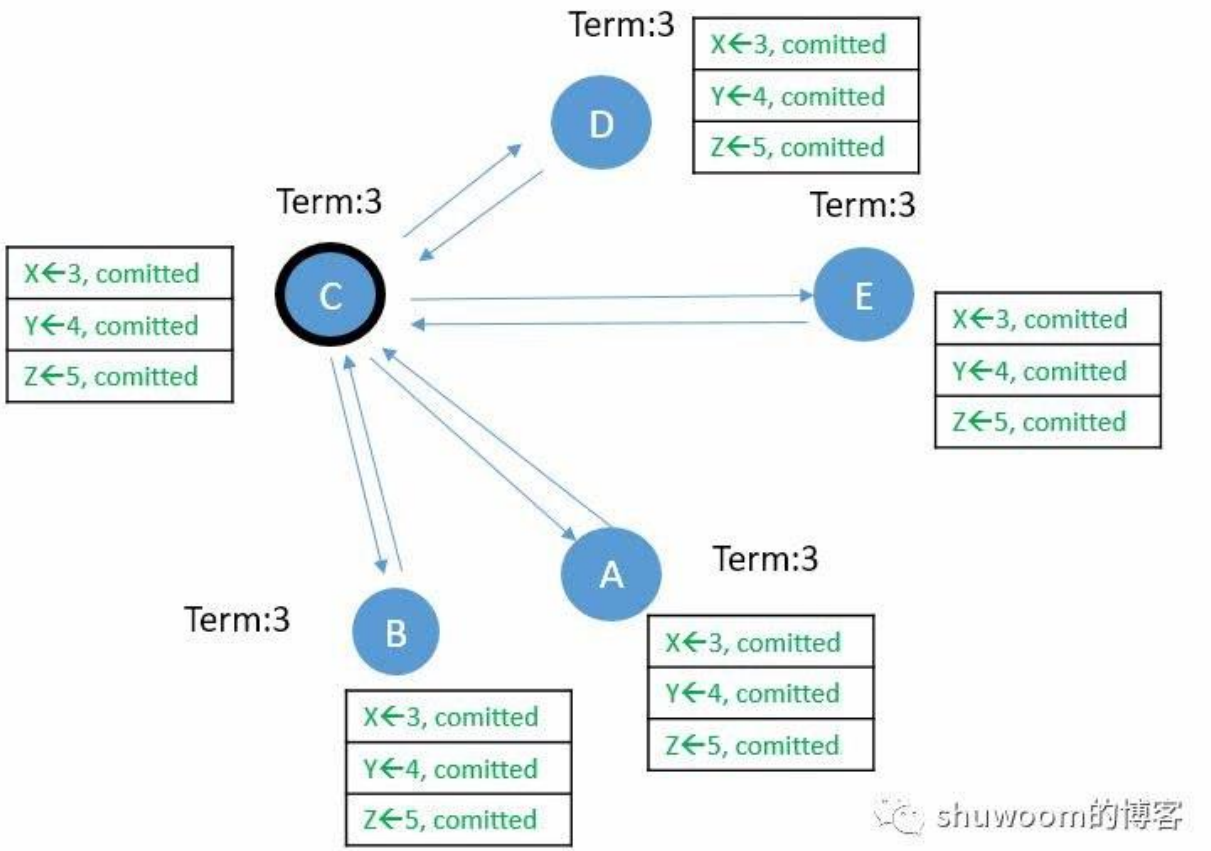
安全性

- 实例分析



安全性

- 实例分析


Term:3
X←3, comitted
Y←4, comitted
Z←5, comitted
D
Term:3
C
Term:3
E
X←3, comitted
Y←4, comitted
Z←5, comitted
X←3, comitted
Y←4, comitted
Z←5, comitted
A
Term:3
X←3, comitted
Y←4, comitted
Z←5, comitted
Term:3
B
X←3, comitted
Y←4, comitted
Z←5, comitted
shuwoom的博客

参考资料

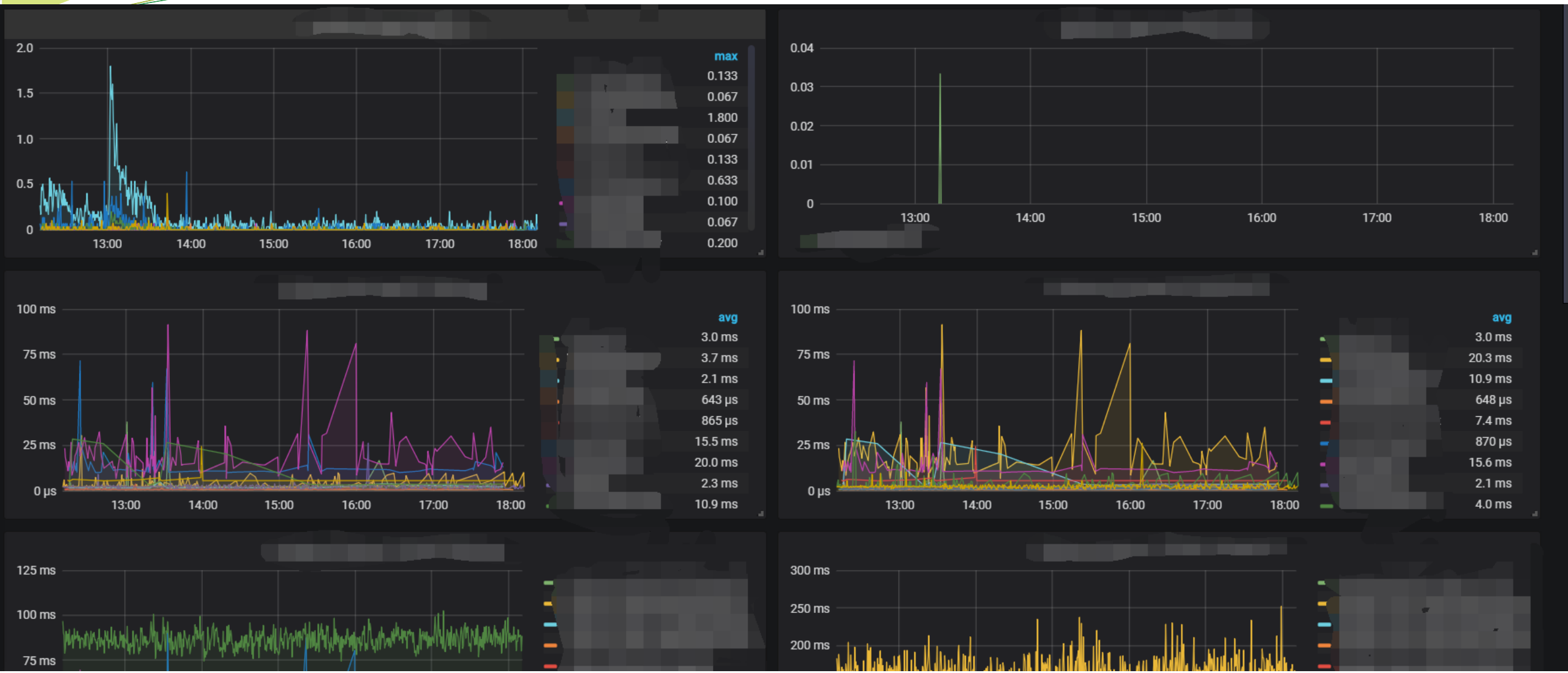
- 参考资料
  - https://www.jianshu.com/p/aa77c8f4cb5c
  - https://container-solutions.com/raft-explained-part-33-safety-liveness-guarantees-conclusion/
  - **http://thesecretlivesofdata.com/raft/ 强烈推荐**
  - https://raft.github.io/raft.pdf
  - https://github.com/maemual/raft-zh_cn/ 中文翻译
  - http://www.yuxiumin.com/2017/08/12/raft-protocol-intro/ 异常情况

RPC调用

- 数据传输
  - Thrift协议
  - Protobuf协议
  - Json协议
  - msgpack协议
- 负载均衡
  - 随机算法
  - 轮询
  - 一致性hash
- 异常容错
  - 健康检查
  - 熔断
  - 限流

服务监控

- 日志收集
  - 日志收集端→ kafka集群→数据处理→日志查询和报警
- Metrics打点
  - 实时采样服务的运行状态
  - 直观的报表展示


2.0
1.5
1.0
0.5
0
13:00
14:00
15:00
16:00
17:00
18:00
max
0.133
0.067
1.800
0.067
0.133
0.633
0.100
0.067
0.200
0.04
0.03
0.02
0.01
0
13:00
14:00
15:00
16:00
17:00
18:00
100 ms
75 ms
50 ms
25 ms
0 µs
13:00
14:00
15:00
16:00
17:00
18:00
avg
3.0 ms
3.7 ms
2.1 ms
643 µs
865 µs
15.5 ms
20.0 ms
2.3 ms
10.9 ms
100 ms
75 ms
50 ms
25 ms
0 µs
13:00
14:00
15:00
16:00
17:00
18:00
avg
3.0 ms
20.3 ms
10.9 ms
648 µs
7.4 ms
870 µs
15.6 ms
2.1 ms
4.0 ms
125 ms
100 ms
75 ms
300 ms
250 ms
200 ms

总结

- 微服务的架构
- 服务注册、发现机制
- raft协议基本了解
- RPC相关概念
- 监控和报警相关概念