

# Codis Design & Implementation

姓名：刘奇

微博：@goroutine

Storage-基础架构部-豌豆荚

# 典型业务

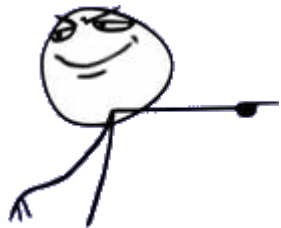


image-service (单个 value 较大)

好友关系(set, key count 4亿+)

applist (720G, 32 instances)

.....

# 业务特点

持久存储: MySQL 或者 HBase

缓存: memcached + Redis

# 豌豆荚的 Redis 历程

- 单实例
- 多实例, 业务代码中做 Sharding
- 单个 Twemproxy
- 多个 Twemproxy
- Codis 豌豆荚自己开发的分布式 Redis 服务

# 使用 Redis 的一些痛点

1. 单机内存有限
2. 带宽压力
3. 单点问题
4. 不能动态扩容
5. 磁盘损坏时数据抢救

# How to scale Redis ?

- 客户端静态分片(Consistent Hash)
- 通过 Proxy 分片
  - Twemproxy
- 官方的 Redis Cluster

# Why not Twemproxy

- 最大痛点：无法平滑的扩/缩容 (Scale!!!!)
  - 甚至修改个配置都需要重启服务。。。
- 不可运维，甚至没有 Dashboard

# Why not Redis Cluster (official)

- 无中心的设计, 难把程序写对
- 代码有点吓人, clusterProcessPacket 函数有426行, 大脑难处理到所有的状态切换
- 迟迟没有正式版本, 等了4年之久
- 目前还缺乏 Best Practice, 还没有人写一个 Redis Cluster 若干条注意事项
- 整个系统高度耦合, 升级比较困难




# 为什么一定要redis

关于 Tair, Couchbase...

良好的数据结构支持

# What's Codis?

- Distributed Redis Proxy
- Redis Cluster 又一个解决方案
- HA
- Design for Scalability
- No SPOF
- Powered by  and C

# Codis 解决了哪些问题？

- 自动扩容, 缩容
- 单点故障
- 单点带宽不足
- 高可用
- 轻松将数据从 Twemproxy 迁移到 Codis
- 轻松运维, 监控

# Codis 整体设计

- Pre-sharding
  - Slot => [0, 1023]
- Zookeeper
- Proxy 无状态
- 平滑扩容/缩容
- 扩容对用户透明

# 设计考量

- 分布式系统是复杂的
- 开发人员不足
- 尽量拆分，简化每个模块，同时易于升级
- 每个组件只负责自己的事情
- Redis 只作为存储引擎
- Proxy 状态

# 设计考量

- Redis 是否挂掉的判定放到外部, 因为分布式系统存活的判定是复杂的
- 提供 API 让外部调用, 当 Redis master 挂掉的时候, 提升 slave 为 master

# 设计考量

- graph everything
  - slot status
  - proxy status
  - group status
  - lock
  - action

# proxy vs smart client

## **proxy:**

更好的监控, 控制

后端信息不暴露, 易于升级

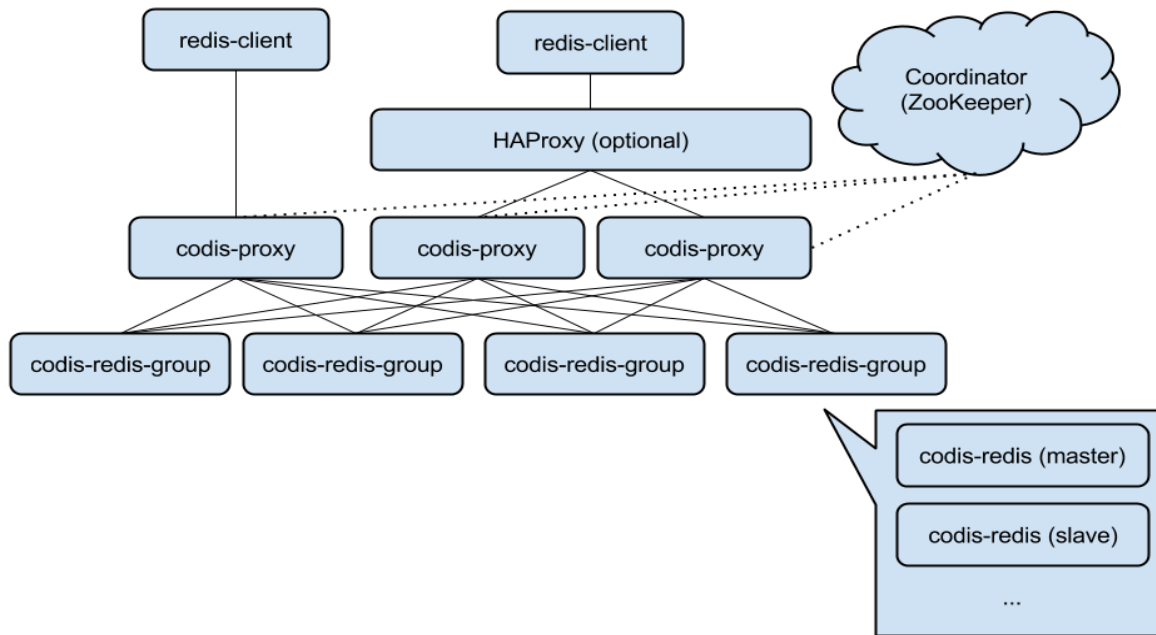
## **smart client:**

更好的性能

更低的延迟, 升级比较麻烦



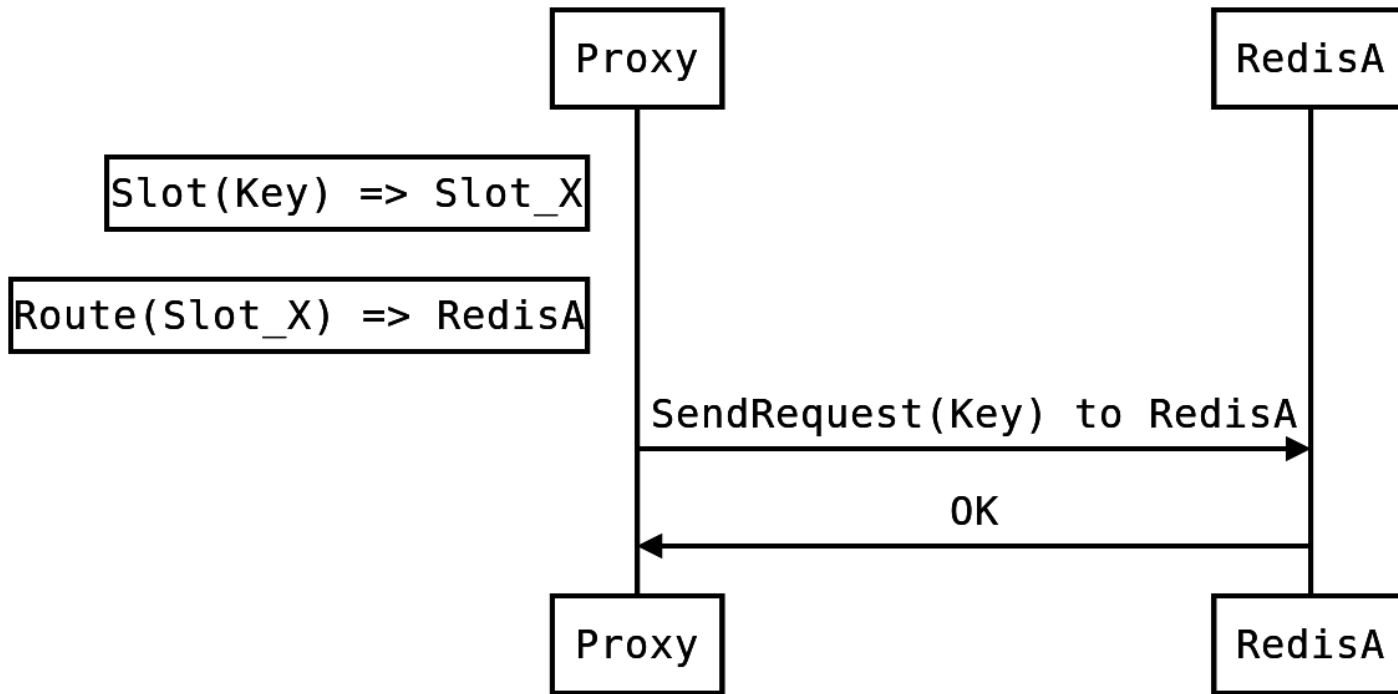
# Codis Architecture



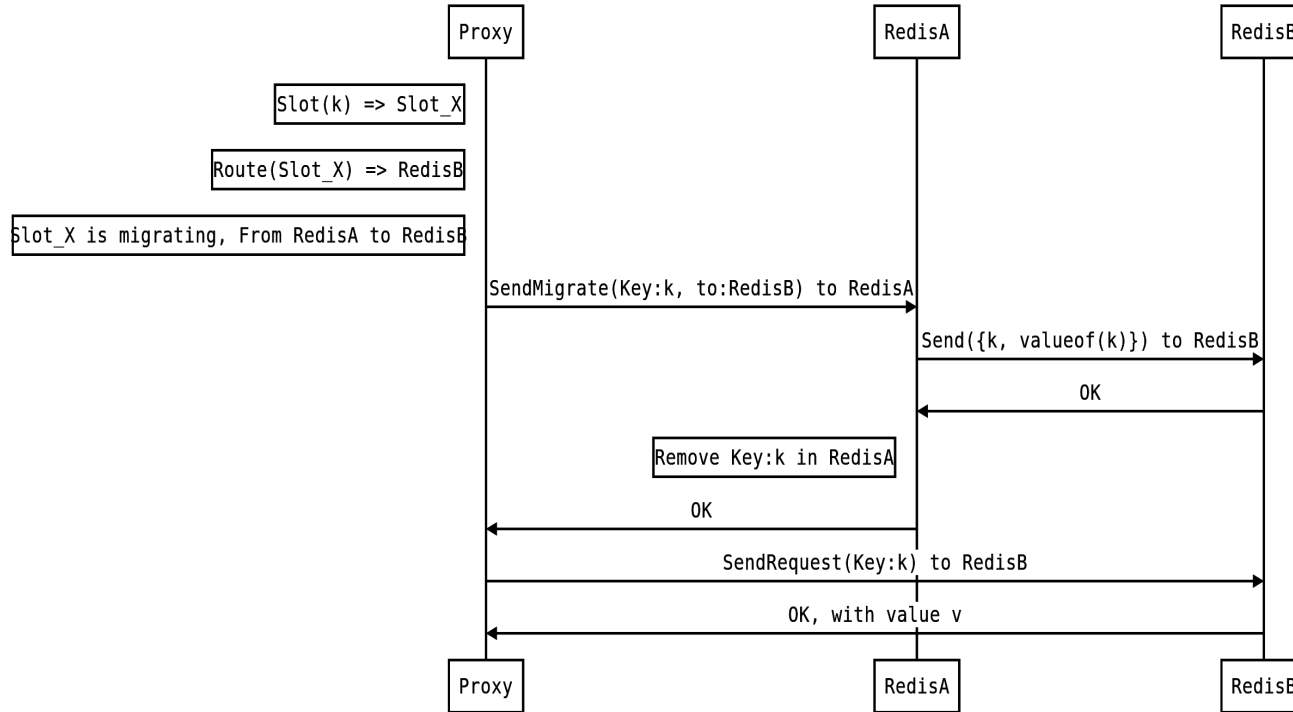
# Zookeeper

- Distributed Lock service
- Router Table
  - Pre-sharding: 1024 slots
- Server Groups
  - Server Groups
  - groupX: [master, slave]

# Read/Write Flow (Normal)



# Read/Write Flow (Migrating)



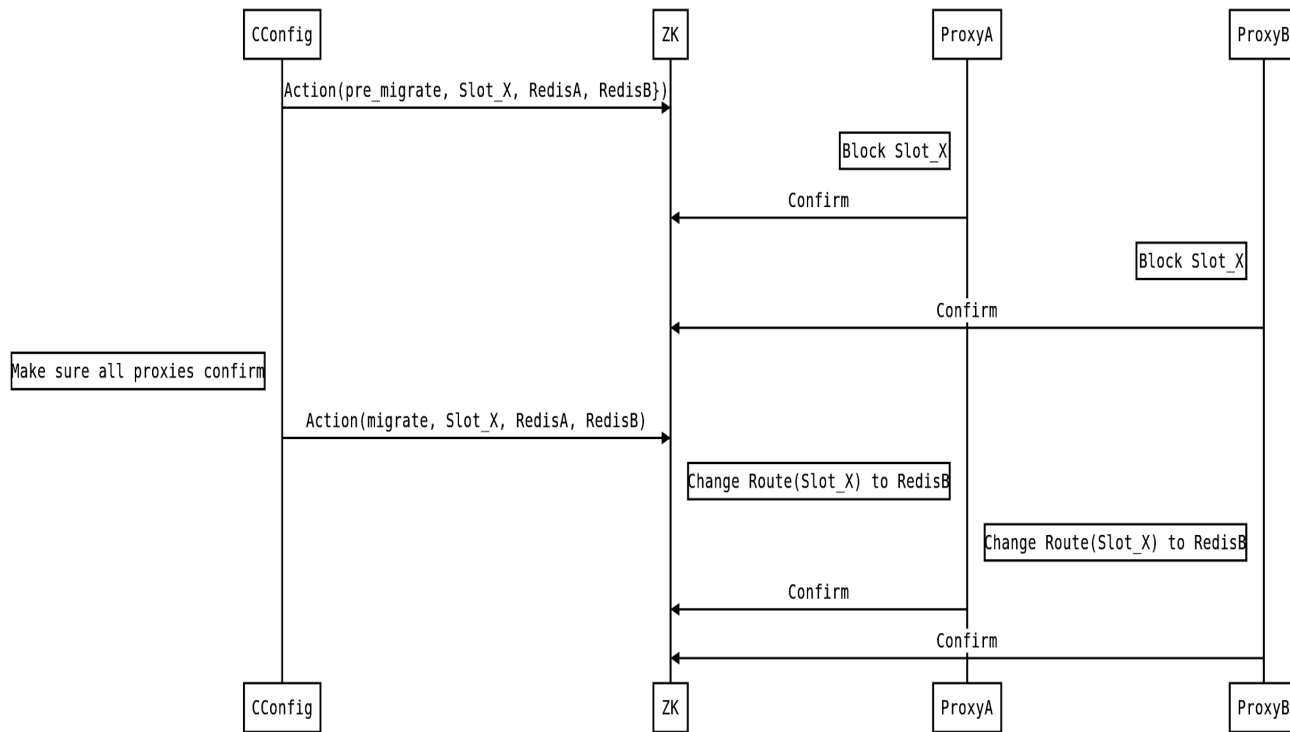
# Migration

1. 将 slot\_X 状态标记为 'pre\_migrate'
2. 等待所有的 proxy 确认
3. 将 slot\_X 状态标记为 'migrating'
4. 不断发送 SLOTSMGRT 命令给 source redis instance 直到 slot\_X 所有的 key 迁移完成
5. 将 slot\_X 状态标记为 'online'

# How it works?

- make sure all proxies updated its routing table
  - 2-phase commit
  - Zookeeper
- Atomic migration
  - Redis is single threaded
  - little C patch...

# How it works(do migrate)

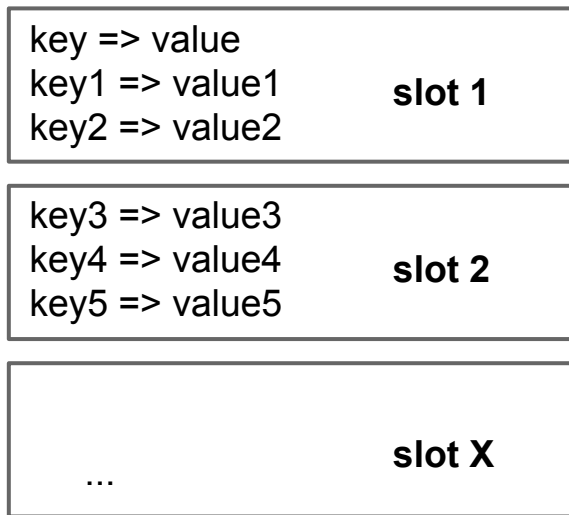


# How it works(codis-redis)

## Redis

key => value  
key1 => value1  
key2 => value2  
key3 => value3  
key4 => value4  
key5 => value5  
...  
keyN => valueN

## Codis Redis





# Router Table

## ZooKeeper ZNode:

`/codis/db_{xx}`

`{xx}` means 产品名, 如: `/codis/db_sync`, `/codis/db_applist`

`/codis/db_{xx}/servers/group_{N}/{ server addr (e.g. 127.0.0.1:6379) }`

存储真实的 redis 组 (主master、从slave), N为一个自定义的整数编号, in JSON, 内容包括服务器地址, 角色 (master or slave) 等信息

# Router Table

// 存储 key slot 的分布信息, N 为 hash(key)

/codis/db\_{xx}/slots/slot\_{N} ( 0 < N < 1024) =>

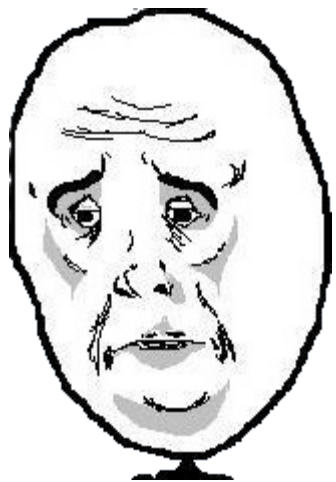
```
{
  'group' : 'group_znode_name',
  'state' : {
    'status' : ' ONLINE| OFFLINE | MIGRATING | PRE_MIGRATE',
    'op_ts' : '123123123123',
  }
}
```

# Codis' Redis

- Slot support
- Atomic migration
  - 每个操作迁移一个key

# Codis Config Tool (cconfig)

- Rebalance
- Manage Server Groups
- Dashboard (HTTP)
- RESTful APIs

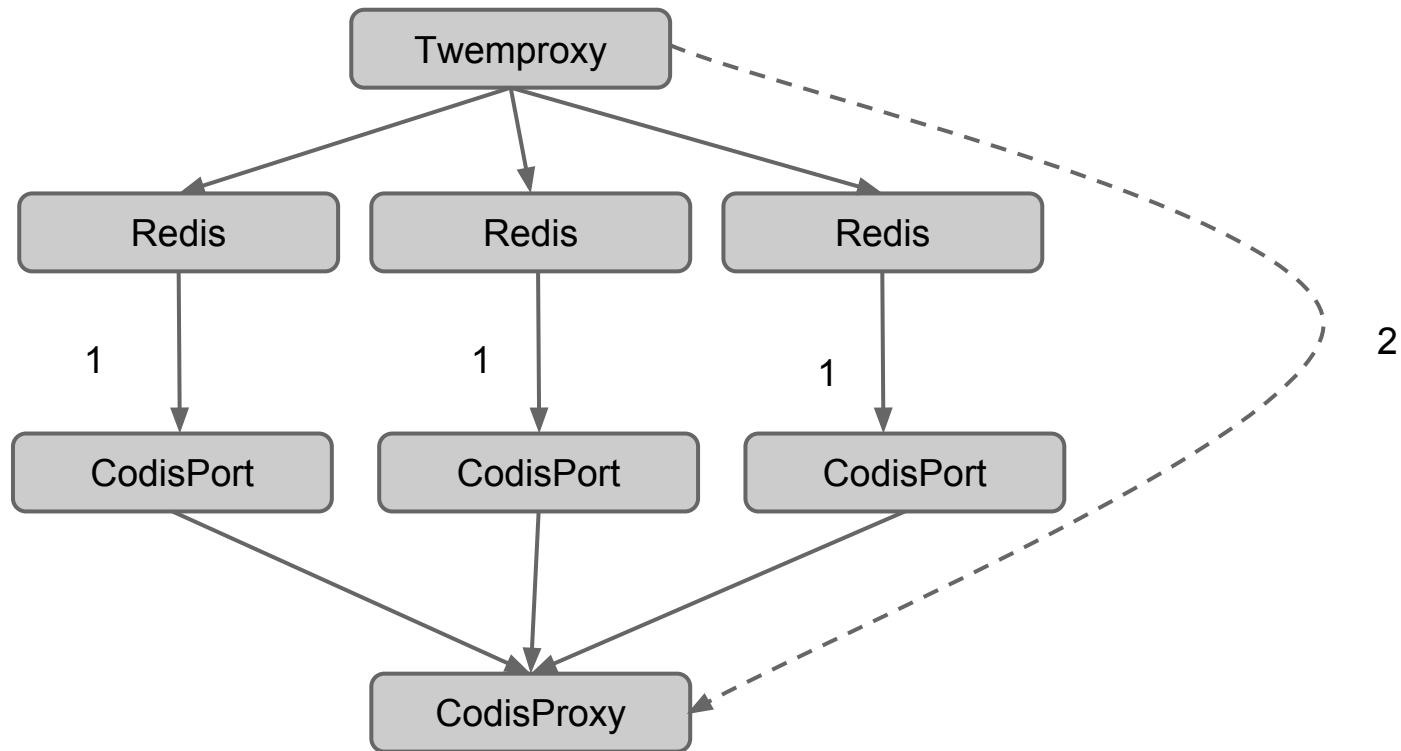


可是我的数据已经在  
Twemproxy 上了,太大了,  
还没法重建,我还懒得  
写迁移脚本,没法迁移!!!  
肿么办!!!

# CodisPort

- 通过replication的方式将twemproxy/redis的数据迁移到codis
- Sync协议
- Fake Slave

# CodisPort



## 使用注意事项

1. 单key的value别太大了(< 1 MB)
  - list别太大...



# TODO

- Persistent storage support
  - LevelDB, Rocksdb .....
- Redis HA module

有些公司是单独运行codis的,  
没有和运维系统结合,也不想自己调用  
codis API

# Performance

Codis会比Twemproxy慢20%左右

# open-source

[github.com/wandoulabs/codis](https://github.com/wandoulabs/codis)



# Thanks

QA