RedisProxy 方案概要设计

目录

1.需求背景

- 背景
  - 存用户画像，总量 800G 内存，数据量 14 亿， DSP 侧高性能查询 15W TPS，低于 10ms 时延，数据过期时间是 30 天，全内存数据，数据全是热点数据
  - 用户点击相关数据（用户画像外的其他数据），要求性能数据 5wtps，低于 10ms 延时，总量低于 100G 内存
  - Redis 数据备份依赖 AOF/Hadoop/Leveldb

- 痛点
  - 业务服务和数据缓存混合部署，CPU/IO/网络资源争抢，导致互相影响不可用
  - 安全能力弱，实例和服务模块权限不明确
  - 扩展能力弱，扩容时需要停机使用，影响业务正常使用
  - 数据挤兑，同个实例不同业务场景写入数据导致内存资源争抢(超内存)
  - 是否存在超大 Key 和慢查询、热点 Key 情况？

- 一期业务目标
  - 高可用 99.95%，流控
  - 高性能 TPS15+w/s(用户画像 15w/s，其他集群 5w/s)，低延时<10ms
  - 数据安全，鉴权避免数据泄漏，操作记录
  - 低成本(易接入、易迁移、易维护、低存储)
- 二期业务目标
  - 高可用 99.95%，负载均衡，平滑扩容，自动恢复
  - 易运维，集群管理，可告警可监控，分片迁移

- 注意点
  - 每台机器要预留 16G 以上内存，用于 AOF 时 rewrite
  - 字段合法性(长度/类型)校验，Pipeline 的限制

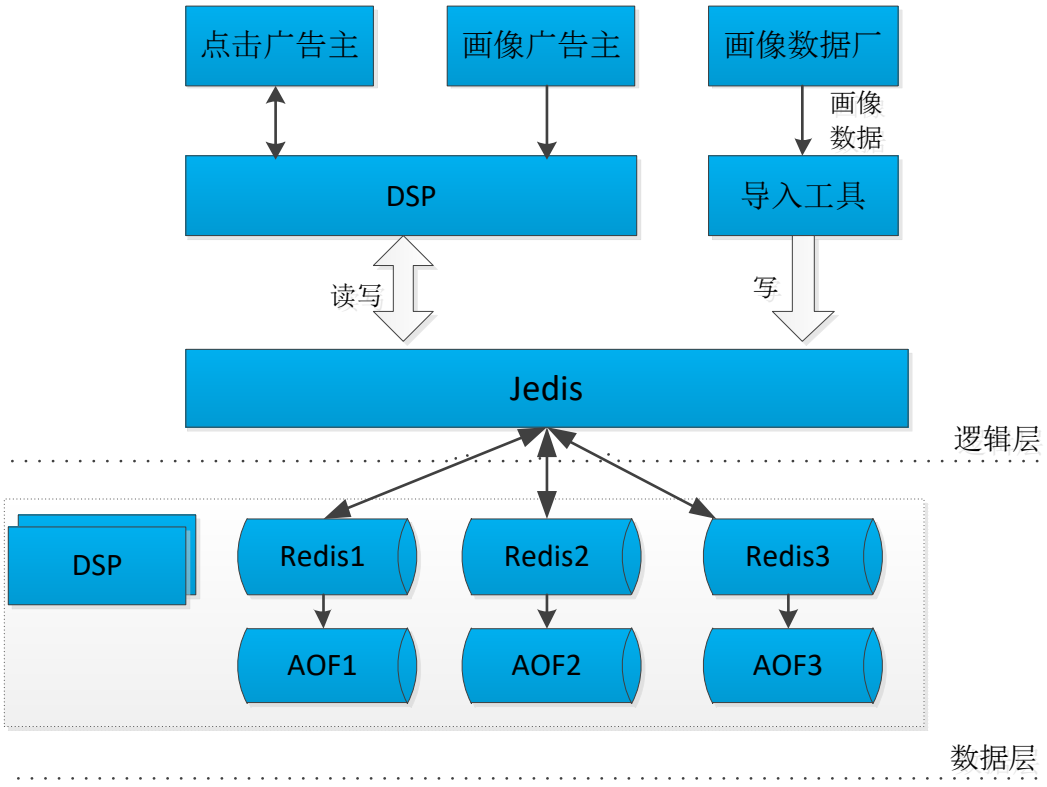
2. 系统关键要素

2.1 关键 key

用户画像：Key：UserID Value：标签数据\属性　　平均记录大小 428 字节
用户点击：Key：UserID Value：点击数据
其他产品：Key：ID Value：其他数据

3. RedisProxy 体系

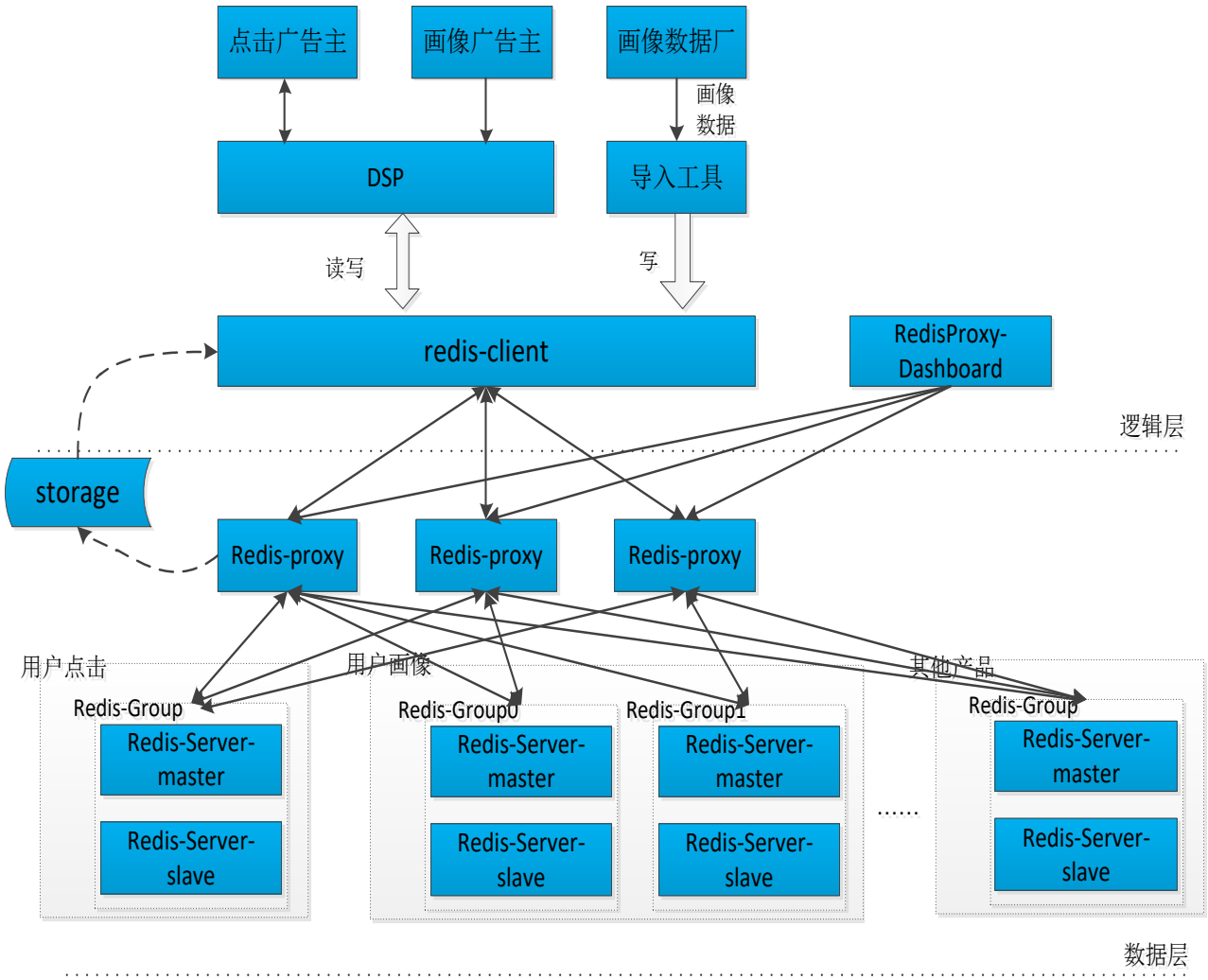
3.0 Redis 旧部署架构图



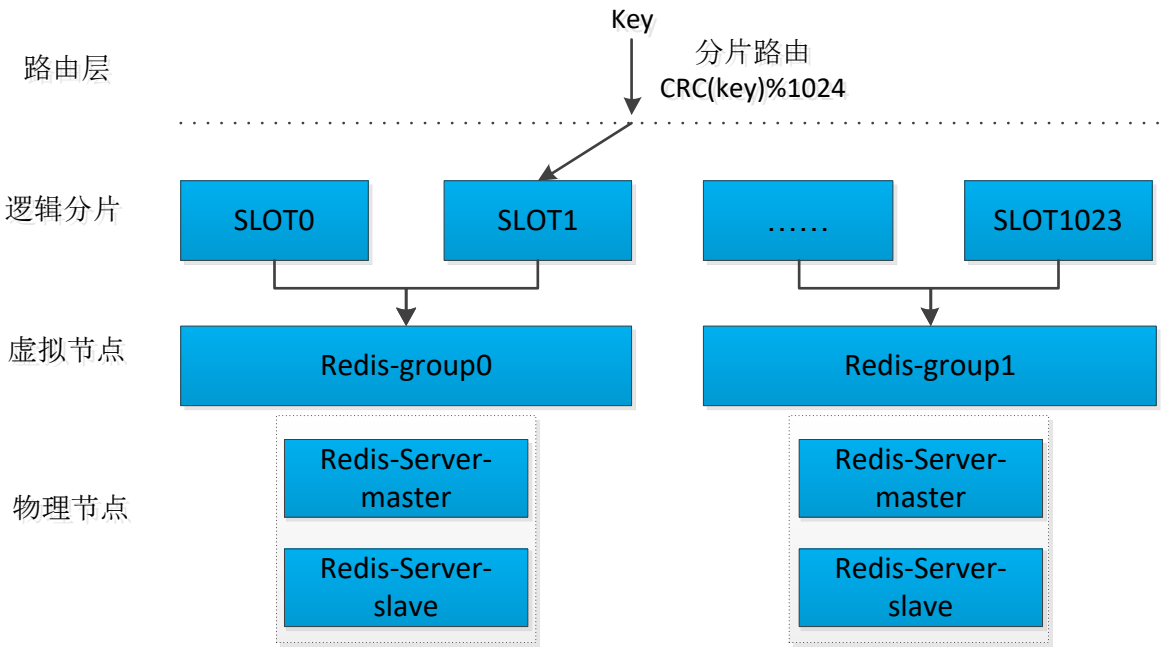
3.0.1 方案对比

Feature	Codis	Redis Cluster	TwemProxy	RedisProxy(自研)	WeRedis
存储引擎	基于原生Redis扩展增加迁移相关指令	原生Redis	原生Redis	原生Redis	原生Redis
Redis版本	3.2.8	跟随版本	略	跟随版本	3.X
数据分布算法	哈希槽crc16(key) % 1024	哈希槽crc16(key) % 16384	ketama/modula/random	一致性hash+路由(自研)	crc(key)%s2048
平滑扩缩容	支持	客户端路由	不支持	支持(自研)，二期	不详
Value大小限制	无	无	无	无	无
开发语言(Proxy)	Go	采用无中心节点设计,无Proxy	C	C++/C	Java
单线程/多线程/多进程(Proxy)	多线程	无	单线程	单线程/多线程(自研)	多线程
鉴权	原生	原生		AppID+AppKey	支持
流控	不支持	不支持	不支持	组内组件	不支持
原数据迁移	需要	需要	需要	需要/待定	需要
proxy负载均衡/容灾	zk	gossip	不支持	组内组件，容灾二期	obServer组件
机型	内存型	内存型	内存型	内存型	内存型
Client	jodis/redis-client	需要支持cluster语义	任意	提供SDK((自研)grpc+pb)	C++/Java
Pipeline	支持	不支持	支持	支持/(自研)	支持
机器成本	机器本身	机器本身	略	机器本身	高
运维成本	中	高	高	低	托管
开发成本	低	高	高	高	低
社区活跃程度	中	高	低	无	行内
单机单核读写性能数据qps(w)	6	15	较低	待	10

3.1 RedisProxy 优化架构图



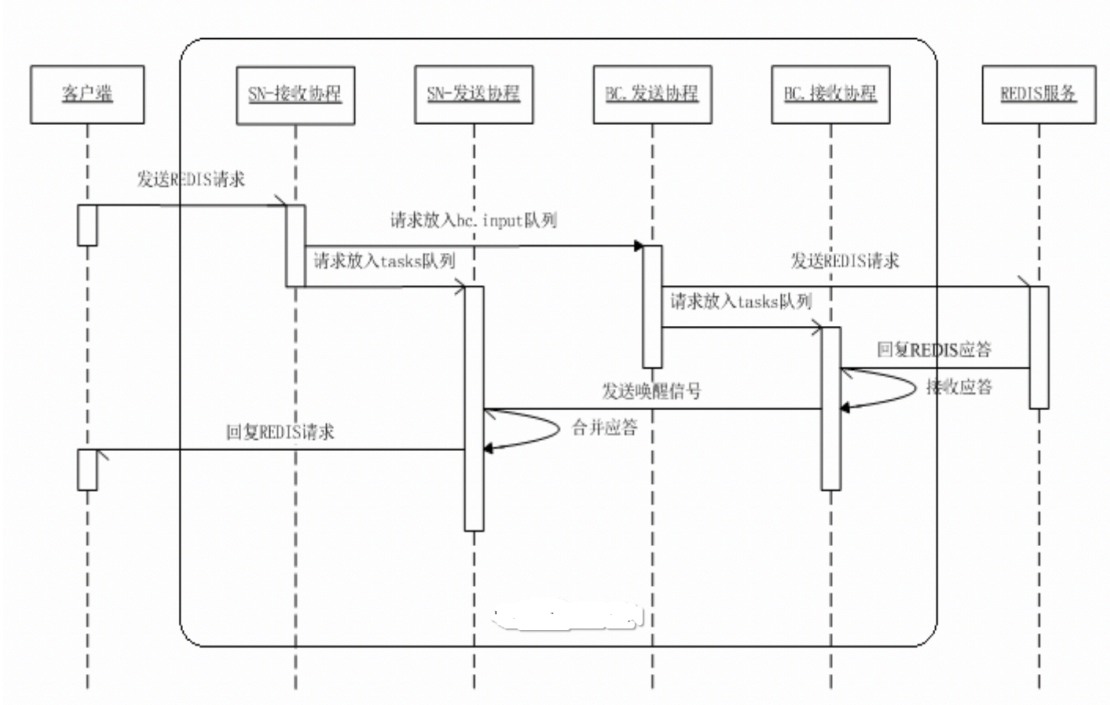
3.2 RedisProxy 路由映射流程



slot：分片信息，在 redis 当中仅仅表示一个数字，代表分片索引。每个分片会归属于具体的 redis 实例。

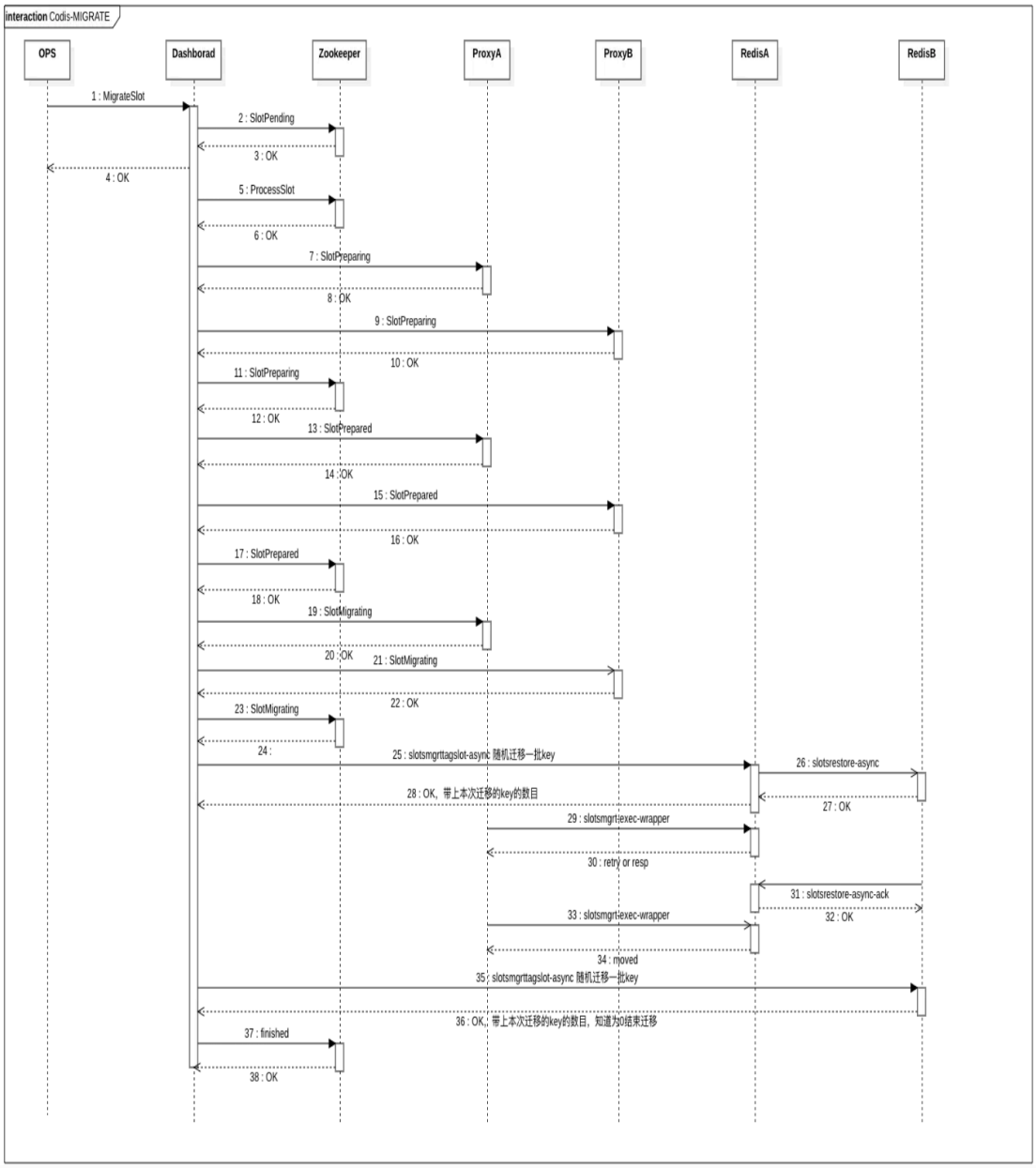
group:主要是虚拟结点，由多台 redis 机器组成，形成一主多从的模式，是逻辑意义上的结点。

3.3 Proxy 请求处理序列图

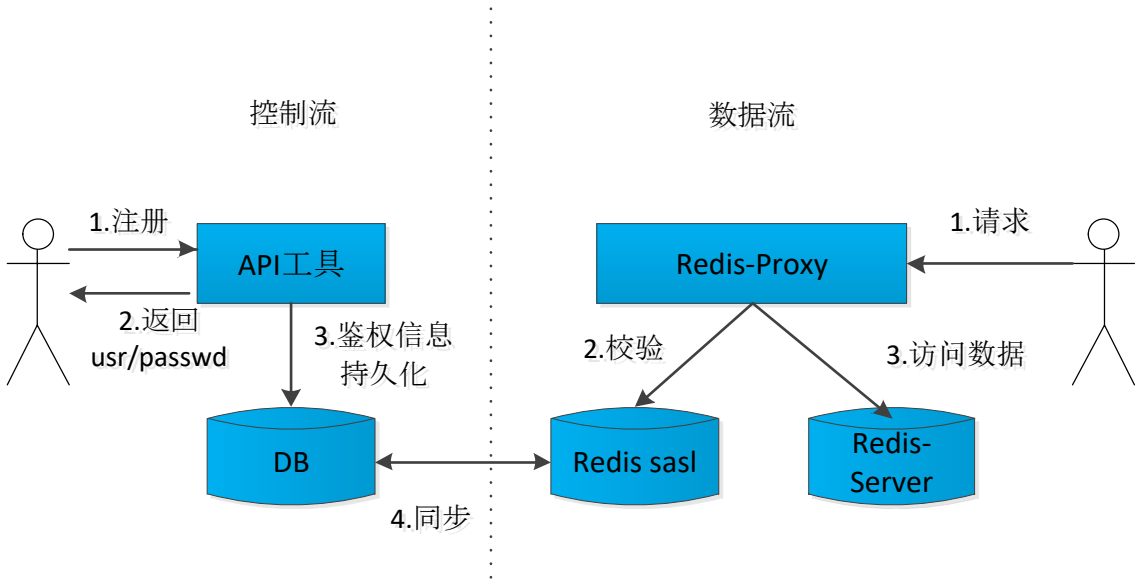


1.客户端：Redis 命令的发起方
2.SN-接收协程：用于接收来自客户端的请求
3.SN-发送协程：用于发送请求处理结果至客户端
4.BC-发送协程：用于将客户端的请求转发到后端某 redis-server 服务
5.BC-接收协程：用于接收来自某 redis-server 服务的应答
6.REDIS 服务：某 redis-server 服务

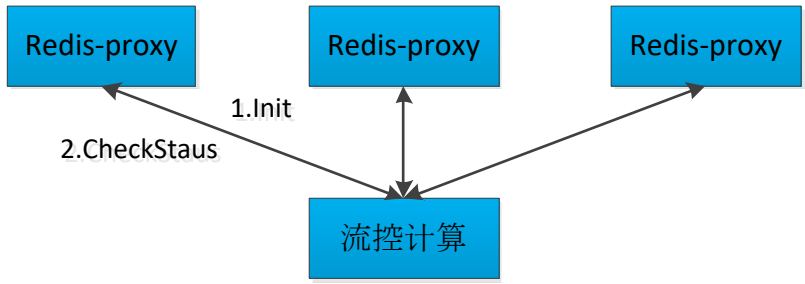
3.4 slot 迁移序列图



3.5 鉴权流程



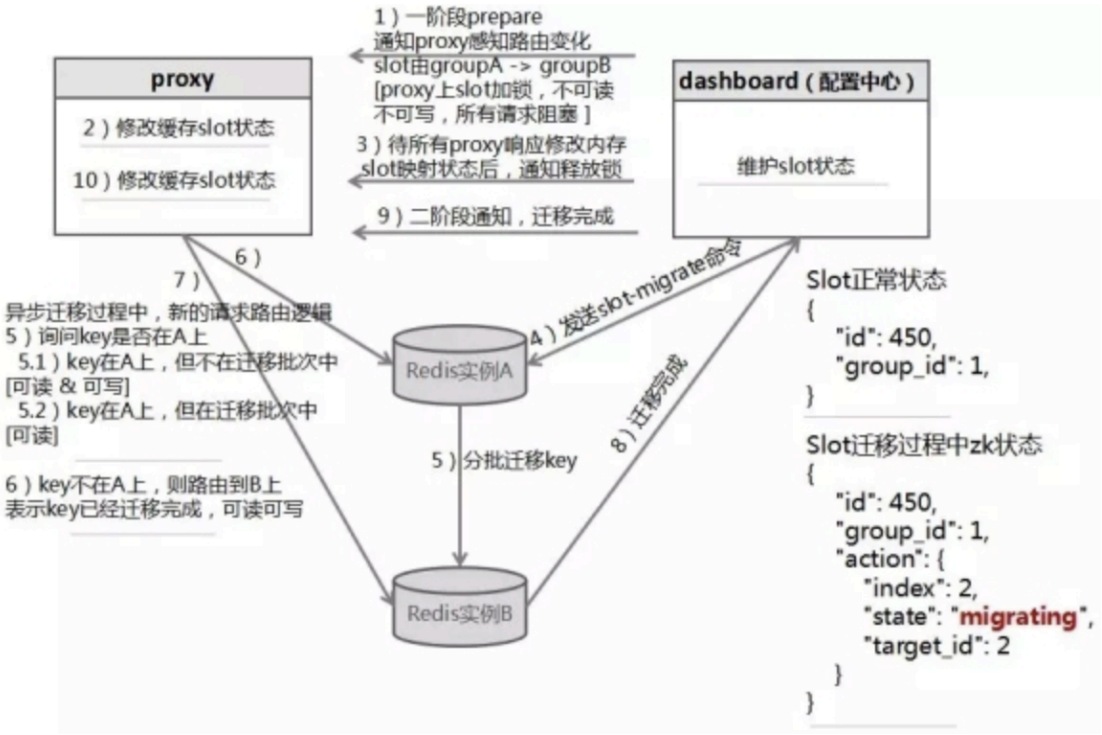
3.6 流控流程



3.7 告警和监控流程图

通过 Dashboard 的能力

3.8 扩容/缩容流程图



3.9 redis->redisProxy 数据迁移流程

参考 4.8 逻辑方案

4. 方案逻辑概述/技术要点

- 4.1 Proxy 路由策略
  1） 业务侧的表名规则：AppName.TableName 对映着一个实例，静态配置着多个 slot（app.table->slot），不同 slot 也映射到对应的 group 虚拟节点上(slot->group)关系，虚拟节点 group 对应着一主多从实例(物理 ip:port)，三者关系保存在 storage 上面
  2） 对 Redis 进行数据读写时，通过 AppName.TablName 得到所有的 slot 关系
  3） 基于 crc32(key)%1024(可配置)得到落在的 slotid
  4） 查找 slotid 对应虚拟节点 groupid 上，并且得到 group 所对应的实例信息，目前只有 master
  5） 通过 4.2 的方式进行 redis 操作

- 4.2 Proxy 处理逻辑(重要)
  0) 初始化加载鉴权配置和流控阀值到本地内存
  1) 启动专门监听端口，接收客户端链接
  2) 监听到客户端链接后，接收客户端请求，创建 Sessoin。单 key 命令请求转发到后端服务；如果是 MGET、MSET、DEL 多 Key 命令，需要拆分和分发对应的 redis-server 服务 4.1 内容；如果是 PipeLine 的模式，则需要拆分和分发对应的 redis-server 服务
  3) 每次完成应答接收，会发送完成信号，告知一次应答完成
  4) 阻塞等待应答，等到所有应答收到后合并结果，并发送结果给客户端
  5) 支持 Pipeline 的使用方式，二期再支持事务命令(但理论只是调用 redis 接口)
  6) 实现同步调用的方式

- 4.3 平滑 Slot 扩展逻辑(难点跟 4.7 过程类似)
  1) 迁移过程
  2) key 迁移原子性保证
  3) 迁移过程中的读写冲突

- 4.4 鉴权逻辑(重要)
  1) 通过配置文件进行鉴权方式，Iptable 设置 Ip 限制规则；redis.conf 设置 passwd 规则和集群名称；实现简单快捷，但对于限流方式不友好，并且不好维护
  2) 通过 DB 方式去写入不同的 APPID(账号)和 APPKey(密码)，实现多租户方式，RedisProxy 进行校验策略，并且可以支持限流方式实现；支持 IP 白名单设置。

- 4.5 告警逻辑(重要)
  1) 实时数据监控：通过自研 RedisProxy 控制台即可查看集群的实时数据，包括 key 数量、占用内存大小、实时请求数等
  2) 历史数据访问：可以通过运维监控来实现，包括内存、CPU、IO、Load、请求数等的历史曲线，并可以设置监控报警

- 4.6 流控逻辑
  1) 接入项目组内限流分布式令牌桶限流方案

2) 应用侧初始化限流配置(令牌速、容量、本地容量)
3) 每次请求时校验是否被限流
4) 需要提供方支持 go 语言

- 4.7 平滑扩容/缩容逻辑

1) 一阶段，Dashboard 通知 proxy 告知路由变化，slot 由 groupA->groupB[proxy 上 slot 加锁，不可读，不可写，所有请求阻塞]
2) zk 修改缓存 slot 状态，待所有 proxy 响应修改内存 slot 映射状态后，通知释放锁
3) 发送 sbt-migrate 命令，分批迁移 Key
   3.1 询问 Key 是否在 A 上，若在但不在迁移批次中[可读&可写]
   3.2 若 Key 在 A 上， 但在迁移批次中[可读]
   3.3 Key 不在 A 上，则路由到 B 上表示 Key 已经迁移完成[可读可写]
4) 迁移过程中，新的请求路由逻辑
5) 二阶段通知，迁移完成(大概 10s)，修改缓存 slot 状态
6) 如果迁移过程中出现失败、超时，会先删除目标节点 Key，然后进行重试
7) 可以使用自动均衡策略保证大量新增节点自动进行迁移(待验证)

- 4.8 redis 迁移 redisproxy 逻辑

1) 搭建生产环境下的 redisporxy 集群，建议至少包括 2 个 proxy、3 个 redis-server
2) 上线前：迁移 redis 数据到 redisproxy 集群。数据迁移工具我们采用的是 redis-port，对每一个实例运行一个 redis-port 来向 redisProxy 导入数据
   需要注意的是：redis-port 的 sync 操作，同步完成以后可以持续同步新写入的数据，切记不要关闭 redis-port 的 sync 进程，以保证 redisproxy 集群的数据能够包括所有 redis 服务器的数据
3) 上线时，修改各个系统的 redis 连接信息，指向新 redisproxy 集群，并重启服务
4) 观察各个系统是否运行正常
5) 观察原 redis 服务是否已经无请求接入

- 4.8.1 低成本平滑迁移逻辑

1) 难点：机器资源少(1040g)，业务数据量大(800g)；1. 不能再申请资源进行全量迁移；2. 迁移过程中如何保证数据读取的完整性

2) 增加 SDK，支持 Jedis 和 RedisProxy 接口，读写情况下灰度(按 key%hash)配置指定接口两个重要功能
3) 非用户画像数据迁移(100g)：
4) 双写逻辑，验证数据；两周后迁移到 RedisProxy 接口
5) 用户画像数据迁移(800g)：
6) 全量 30%通过 SDK 导入到 RedisProxy 接口
7) 灰度双写逻辑—灰度流量(key%hash_写=模值)，全量通过 Jedis 接口写；灰度流量通过 RedisProxy 接口写
8) 灰度双读逻辑—灰度流量(key%hash_读=模值)，非灰度流量通过 Jedis 接口读，灰度流量通过 RedisProxy 接口读，业务侧配合校验数据准确性
9) 运行 3～5 天后，Jedis 接口设置为非灰度流量写入，该状态运行 2 周(旧 redis 自然淘汰 100g 数据)；

10) 2 周后，灰度双写流量 42.5%，灰度双读 42.5%；该状态运行 1 周； 1 周后，查看集群容量情况评估是否能直接切换剩余流量

11) 写灰度情况(每天增量 set5 亿数据，总存量 800g 数据)

灰度值(key%hash)	Jedis(g)	RedisProxy(g)	运行时间	备注
30%	800	240	2 周时间	补全
42.5%	700	340		补全
待定				补全
100%	剩余	800	1 个月	补全

Ps：补全=导全量数据，保证灰度的数据都落到 RedisProxy

读灰度情况(>30%后跟写流量保持一致)

灰度值(key%hash)	Jedis(g)	RedisProxy(g)	运行时间
1%	99%	1%	30min
5%~29%	95%~71%	5%~29%	3~5day
30%	70%	30%	2week
42.5%	57.5%	42.5%	
待定			
100%			1mon

问：如果全上后大量失败了哪些情况，怎么回退？

答：1. 每一个灰度过程中保证顺利，全上之后就没有该问题；2. 回退 SDK 的 Key%hash 配置，同时导入全量数据

问：写灰度 42.5%过程中，灰度数据仅单写到 RedisProxy，而这时读还停留在 30%，会出现读取不到新数据？

答：确实短时间内会读到部分旧数据，但不影响服务正常运行和不会出现数据丢失；当读也切换过来后就会恢复到正常读写；

问：在灰度读的过程中，命中灰度读的数据在旧的 proxy，这种情况下如何处理？

答：1. SDK/Proxy 判断重读，增加延迟；SDK/Proxy 双读，会增加集群的压力；

2. 增加迁移数据缓存，如果命中则读新的，否则读旧的；

3. 灰度数据补全，即灰度到一定值时进行全量导入，缺点：人工需要介入

- 4.9 部署逻辑

1) 存用户画像，总量 800G 内存，我们搭建 16G*5*10 个 Redis 实例来支持，其中：16G 是单个实例大小，10 台机器，每台机器起 5 个 Redis 实例。数据是 DMP 批量导入，DSP 侧高性能查询 15W TPS，低于 10ms 时延。数据过期时间是 30 天。

2) 存用户点击相关数据（用户画像外的其他数据），总量低于 100G 内存，我们搭建 4G*4*10 个 Redis 实例来支持，其中：4G 是单个实例大小，10 台机器，每台机器起 4 个 Redis 实例；

- 4.10 组件引入

1) 支持 CRC/唯一 id/config 配置/日志等组件，提高开发效率

5. 模块和功能、接口定义等

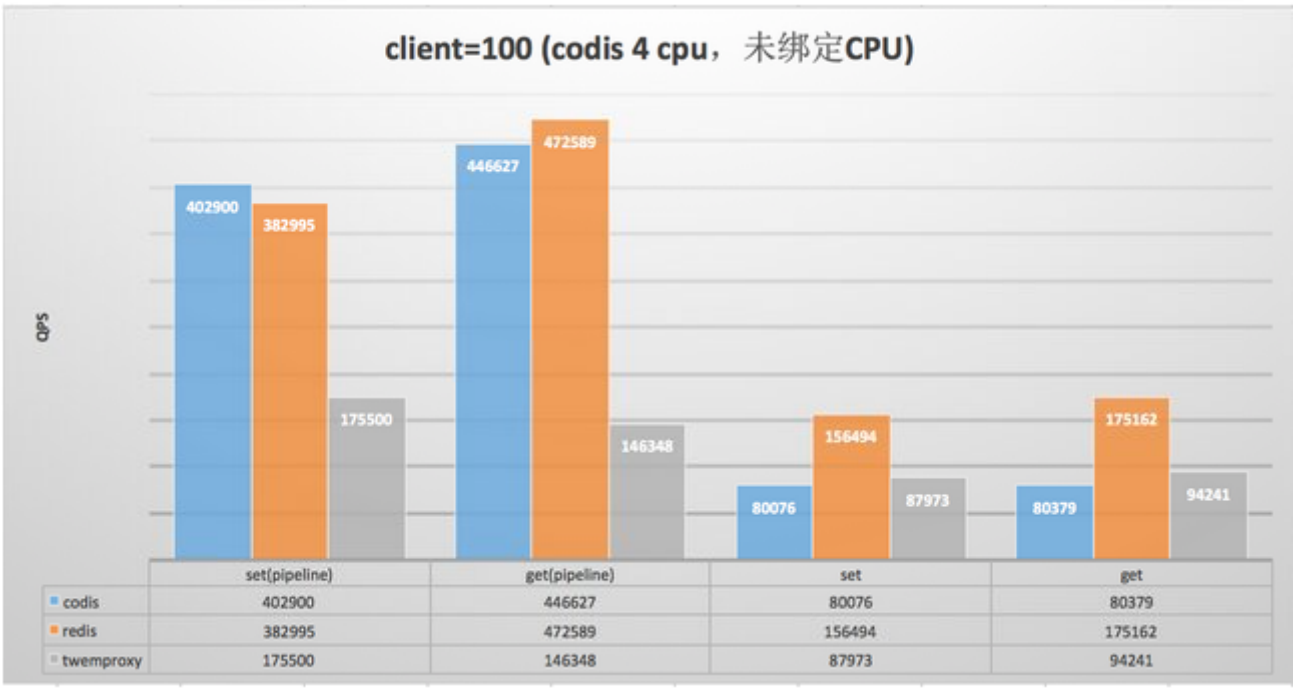
模块名	功能	备注
RedisProxy	客户端连接 Redis 代理服务，支持 Redis 协议，增加鉴权和流控，一致性 Hash、路由	无状态 可部署多个
Redis-Dashboard	管理工具，支持增删 proxy/redis 节点，发起数据迁移，在浏览器观察 redis 集群状态，可监控	
Storage	为集群状态提供外部存储 1. 提供 NameSpace 区分不同的产品，拥有不同的 productname，各项配置不冲突；2. 存放数据路由表和 proxy 节点原信息	实现 zk/etcd/consul
Redis-Server	支持原生 Redis。增加额外的数据结构，以支持 slot 操作以及数据迁移指令	兼容原生 Redis
Redis-Group	一个主多个从，设定为一个组	多机房多活部署
Redis-Client	提供 SDK，封装 Redis 命令接口，但功能支持	通用协议 grpc+pb
RedisProxy -Admin	集群管理的命令行工具	
RedisProxy-FE	集群管理界面，1. 多个集群实例共享可以共享同一个前端展示页面；2. 通过配置文件管理后段 redis-dashboard 列表，配置文件可自动更新	

服务或者代理	Api	功能描述
Redis-Client	TODO	1. 读写集群资源数据 2. 服务发现 3. 确认是否支持域名的方式
Redis-Proxy	RedisProxyCmd	1. 操作 Redis 数据操作函数

6. 存储/性能数据预估

6.1 同类组件 git 测试数据：

1 台 proxy 服务，8 线程，4 核，value：2；测试数据 1 亿，普通性能数据读写 8w/s，pipeline=500 读 44.6w/s,写 40w/s



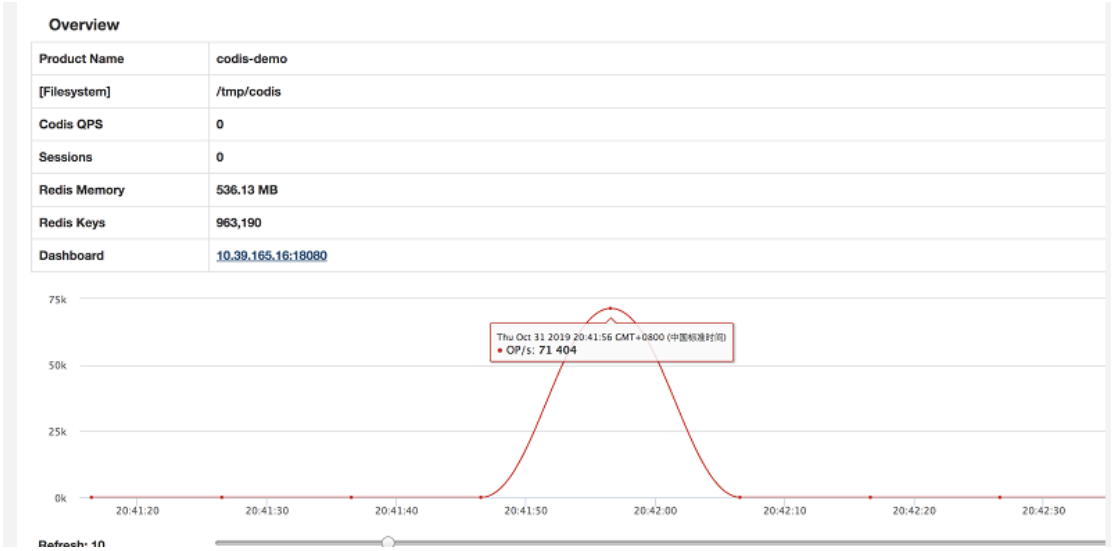
6.2 同类组件个人压测数据：

1 台 proxy 服务，2 个 group， 4 线程，6 核，value：430 机器测试 3 次 10w，普通性能读写 7w/s，qps 基本是 71000 左右

```
ACANDYSUI-MB0:bin luoshaohua$ ./redis-benchmark -p 19000 -h 10.39.165.16 -n 100000 -r 1000000 -c 50 -d 430 -t get,set,mset -q
SET: 68212.83 requests per second
GET: 73583.52 requests per second
MSET (10 keys): 48496.61 requests per second

ACANDYSUI-MB0:bin luoshaohua$ ./redis-benchmark -p 19000 -h 10.39.165.16 -n 100000 -r 1000000 -c 50 -d 430 -t get,set,mset -q
SET: 62814.07 requests per second
GET: 70721.36 requests per second
MSET (10 keys): 58207.21 requests per second

ACANDYSUI-MB0:bin luoshaohua$ ./redis-benchmark -p 19000 -h 10.39.165.16 -n 100000 -r 1000000 -c 50 -d 430 -t get,set,mset -q
SET: 67842.61 requests per second
GET: 68870.52 requests per second
MSET (10 keys): 58105.75 requests per second
```



6.3 同类组件腾讯单点压测数据：

4 核 8G 内存 100G 磁盘(其他部署不明确)

命令	redis-最佳性能	proxy-最佳性能
PING	988467.88(6ms-99.79) 直连-pipeline(100) 50client	550458(7ms-99.1) proxy-pipeline(1000)10client
SET	**352941.19(4ms-99.84)** **直连-pipeline(100) 10client**	**315855.97(9ms-99.2)** **proxy-pipeline(1000) 10client**
GET	**917431.19(10.0ms-99.69)** **直连-pipeline(50)150client**	**468384(8ms-99.4)** **proxy-pipeline(1000)5client**
INCR	430601.41(13ms-99.64) 直连-pipeline(100)50client	408329.94(8ms-99.4) proxy-pipeline(1000)10client
LPUSH	291686.94(4ms-99.45) 直连-pipeline(100)10client	243783.5(11ms-99.8) proxy-pipeline(1000)10client
RPUSH	360620.25(2ms-99.29) 直连-pipeline(50)10client	333111.25(8ms-99.2) proxy-pipeline(1000)10client
LPOP	339866.34 (3ms-99.92) 直连-pipeline(50)10client	327546.69(9ms-99.6) proxy-pipeline(1000)10client
RPOP	442673.75(4ms-99.99) 直连-pipeline(100) 10client	407332.00(8ms-99.7) proxy-pipeline(1000)10client
SADD	**668300.31(4ms-99.41)** **直连-pipeline(100)10client**	**419639.12(8ms-99.3)** **proxy-pipeline(1000) 10client**
SPOP	1126972.25(8.0ms-99.11) 直连-pipeline(50)150client	501504.53(7ms-99.1) proxy-pipeline(1000)10client
LPUSH	289519.38(4ms-99.75) 直连-pipeline(100)10client	244140.61(12ms-99.9) proxy-pipeline(1000)10client
LRANGE_100	41879.55(6.0ms-99.85) 直连(150client)	25028.78(12.0ms-99.56) proxy-1 连接(150client)
LRANGE_300	18162.67(9.0ms-99.54) 直连(150client)	8289.81(29.0ms-99.37) proxy-1 连接(150client)
LRANGE_500	12254.3(13.0ms-99.3) 直连(150client)	5588.5(42.0ms-99.1) proxy-1 连接(150client)
LRANGE_600	10399.55(17.0ms-99.36) 直连(150client)	4213.45(54.0ms-99.15) proxy-1 连接(150client)
ZADD	**164203.61(5.0ms-99.83)** **"直连-pipeline(50)10client"**	**113520.27(3.0ms-99.9)** **"proxy-pipeline(1000)1client"**
ZRANGE_1	637348(16.0ms-99.45) "直连-pipeline(50)150client"	376364(12.0ms-99.3) "proxy-pipeline(1000)5client"
ZRANGE_WITHSCORE_1	260824(34.0ms-99.48) "直连-pipeline(50)150client"	203748(14.0ms-99.23) "proxy-pipeline(1000)5client"
ZRANGE_50	64968.81(3.0ms-99.92)	45993.93(7.0ms-99.35)
ZRANGE_WITHSCORE_50	16359.12(11.0ms-99.7)	13784.93(15.0ms-99.36)
ZRANGE_100	41687.51(5.0ms-99.41)	23559.9(13.0ms-99.4)
ZRANGE_WITHSCORE_100	8931.92(20.0ms-99.17)	7003.98(30.0ms-99.04)
ZRANGE_300	17480.38(11.0ms-99.56)	7345.54(33.0ms-99.38)
ZRANGE_WITH_SCORE_300	3019.11(58.0ms-99.62)	2313.19(96.0ms-99.08)
ZRANGE_500	10486.91(21.0ms-99.33)	4251.1(56.0ms-99.27)
ZRANGE_WITH_SCORE_500	1796.42(91.0ms-99.27)	1350.8(171.0ms-99.02)

7. 日志设计

略

8. 容灾设计

Proxy 负载均衡
宕机时通过 AOF 的方式恢复，未来再考虑主从机制

9. 监控报表设计

- 监控维度

接口调用次数、写入数据量、读出数据量、调用耗时、数据淘汰

10. 部署方案

模块	配置	内存要求	节点数量	备注
RedisProxy	16 核/128G 内存	单机 2g	5	机器共用(可扩容/缩)
RedisServer	16 核/128G 内存	单机 80g	10	机器共用
Storage	16 核/128G 内存	待评估	3	机器共用

11. 工期计划表

13. 效果

略