DTCC

第十一届中国数据库技术大会

DATABASE TECHNOLOGY CONFERENCE CHINA 2020

架构革新 高效可控

IT168.com ChinaUnix.net ITPUB

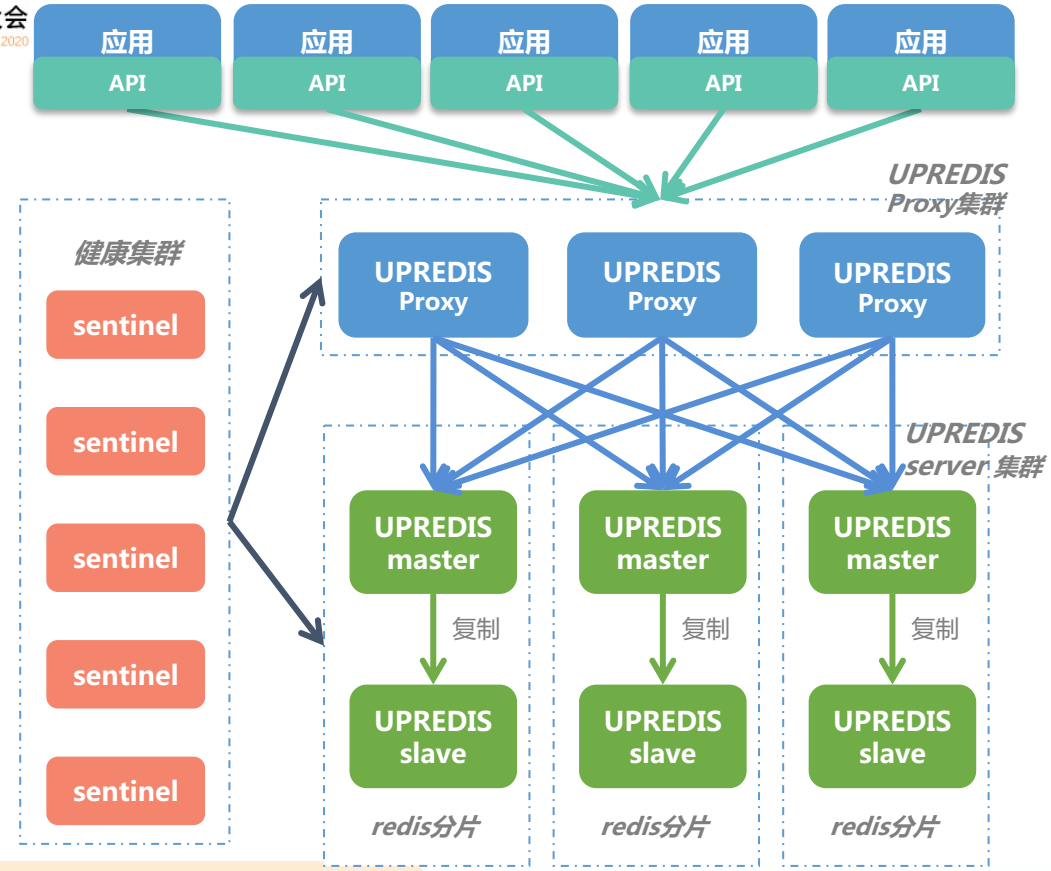
北京国际会议中心 | 2020/12/21-12/23

银联分布式缓存的异地多活实践

中国银联云计算中心 赵仕荣

目录

UPRedis 集群架构



- 连 接 池
- 负 载 均 衡
- 高 可 用
- 数据分片
- 自动主备切换
- 读写分离
- 支持双活和异地容灾方案
- 支持冷热数据分离
- 兼容redis-5.0
- 支持数据加密

WHY多中心？

01.多中心趋势，单数据中心资源有限， 应用逐渐扩展到多中心

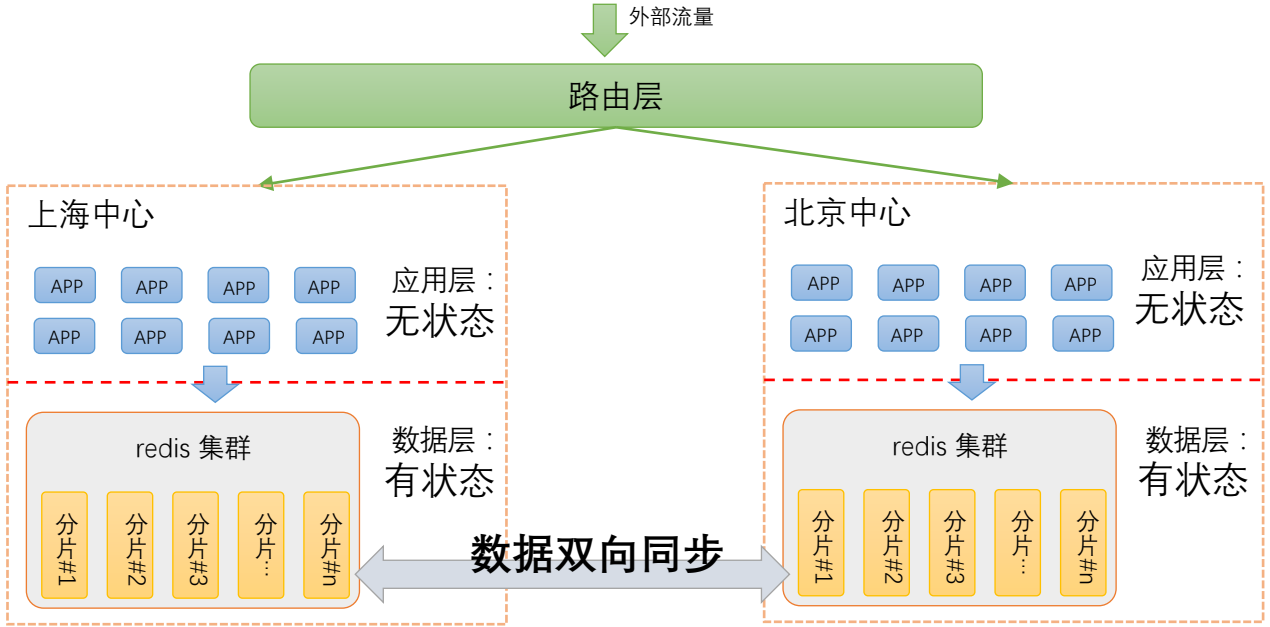
02. 就近原则，请求尽量就近处理，降低延迟

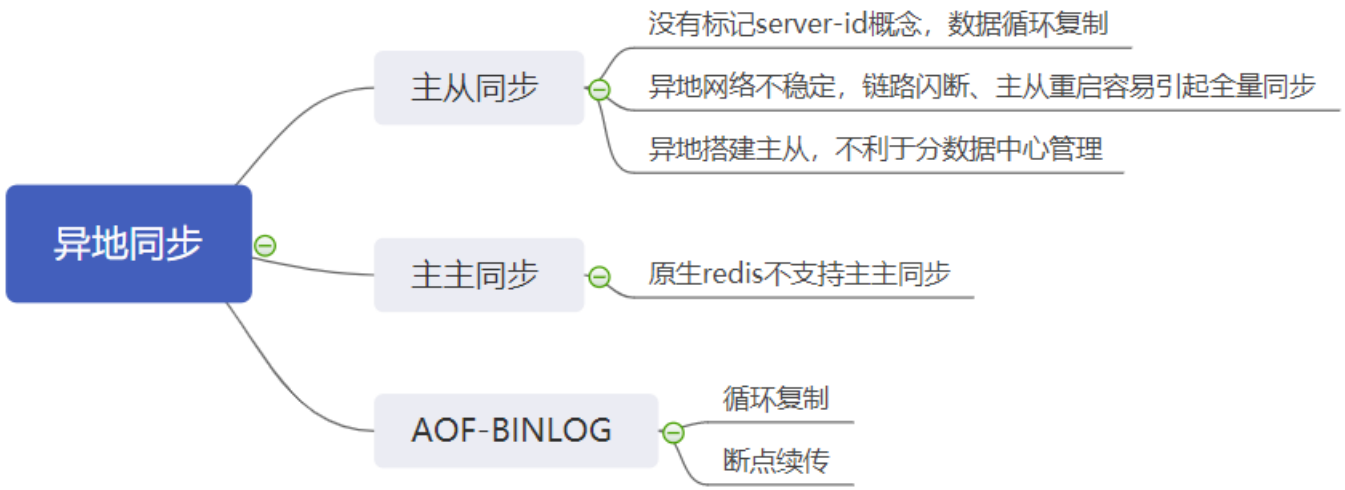
03. 可靠性要求，可靠性要求高的系统，数据需要异地存储

WHY双向同步？

应用层无状态，随时切换；

数据层有状态，异地中心需要有全量数据，因此需要数据异地同步。




异地同步
主从同步
没有标记server-id概念，数据循环复制
异地网络不稳定，链路闪断、主从重启容易引起全量同步
异地搭建主从，不利于分数据中心管理
主主同步
原生redis不支持主主同步
AOF-BINLOG
循环复制
断点续传

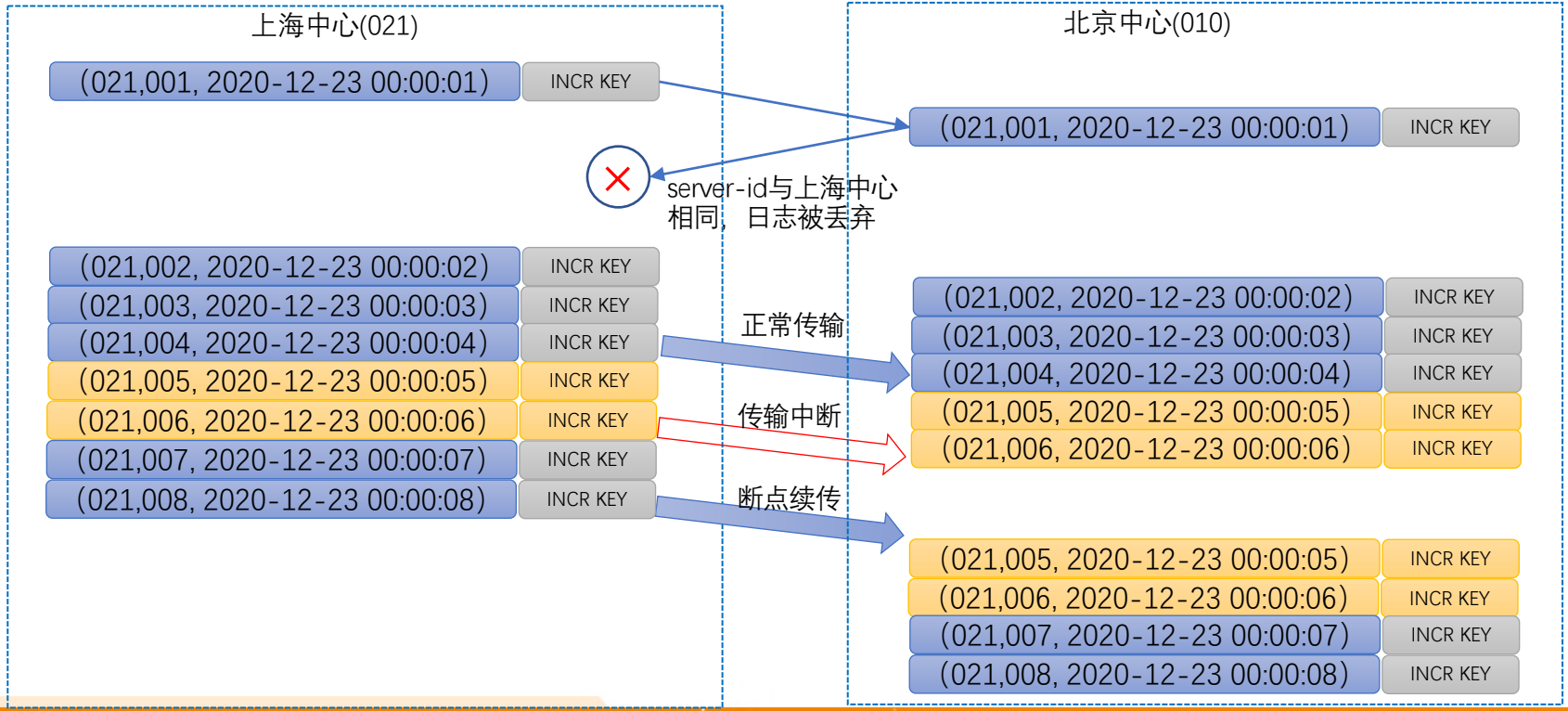
目录

一、循环复制

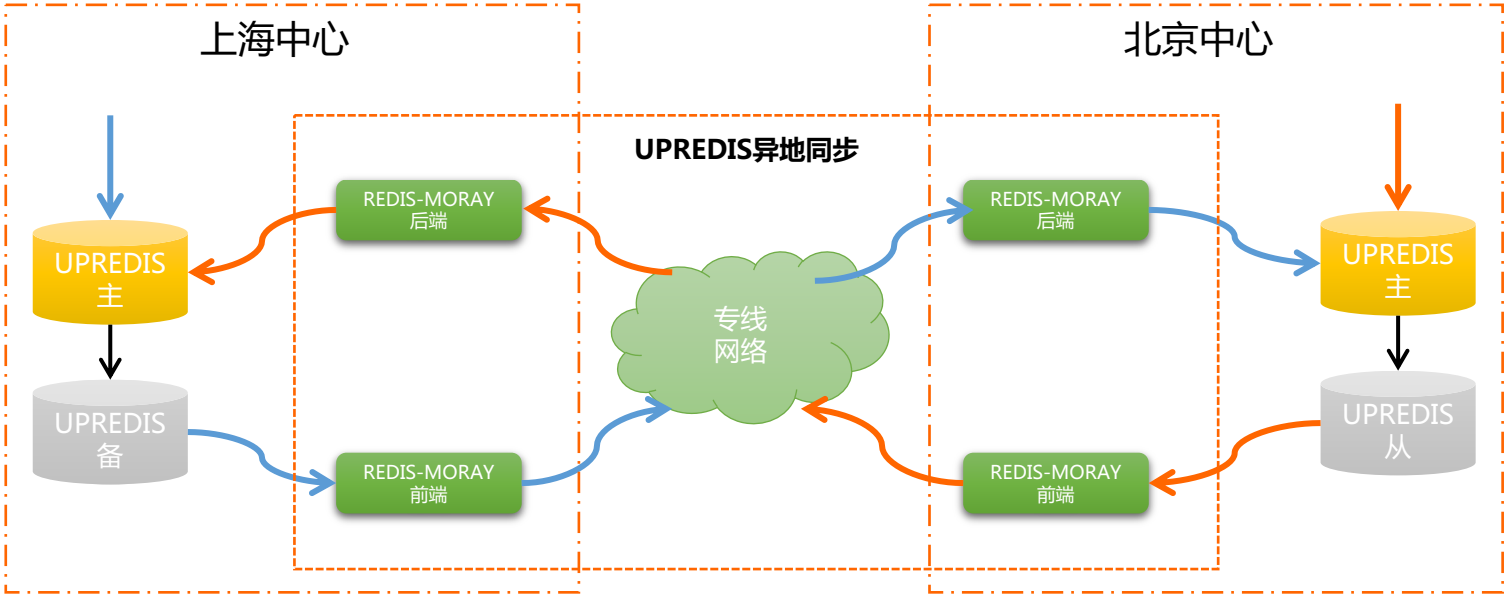
相同server-id的日志被过滤，避免回放自身产生的日志

二、断点续传

已经应用过的日志被过滤，避免回放已经回放过的日志


上海中心(021)
北京中心(010)
(021,001, 2020-12-23 00:00:01)
INCR KEY
(021,001, 2020-12-23 00:00:01)
INCR KEY
server-id与上海中心
相同，日志被丢弃
(021,002, 2020-12-23 00:00:02)
INCR KEY
(021,003, 2020-12-23 00:00:03)
INCR KEY
(021,004, 2020-12-23 00:00:04)
INCR KEY
(021,005, 2020-12-23 00:00:05)
INCR KEY
(021,006, 2020-12-23 00:00:06)
INCR KEY
(021,007, 2020-12-23 00:00:07)
INCR KEY
(021,008, 2020-12-23 00:00:08)
INCR KEY
正常传输
传输中断
断点续传
(021,002, 2020-12-23 00:00:02)
INCR KEY
(021,003, 2020-12-23 00:00:03)
INCR KEY
(021,004, 2020-12-23 00:00:04)
INCR KEY
(021,005, 2020-12-23 00:00:05)
INCR KEY
(021,006, 2020-12-23 00:00:06)
INCR KEY
(021,005, 2020-12-23 00:00:05)
INCR KEY
(021,006, 2020-12-23 00:00:06)
INCR KEY
(021,007, 2020-12-23 00:00:07)
INCR KEY
(021,008, 2020-12-23 00:00:08)
INCR KEY

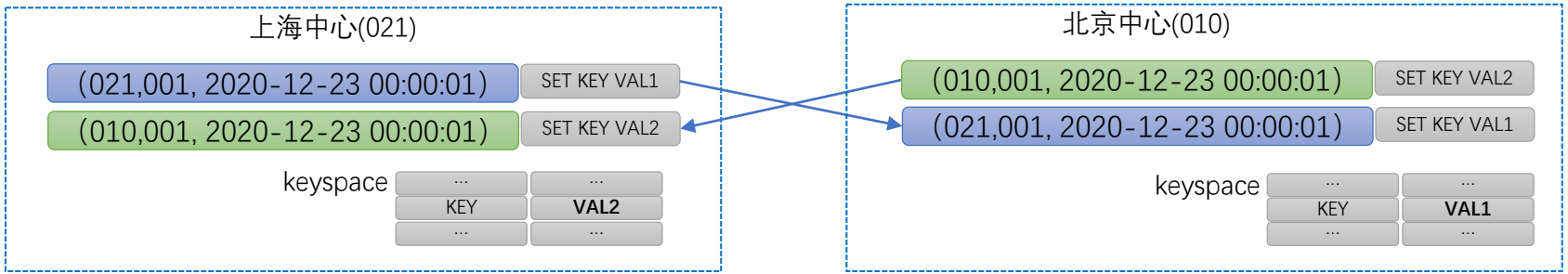
异地同步



主从库开启AOF日志，UPREDIS-MORAY订阅从库AOF数据，传输到异地回放

目录

多写冲突



由于数据多写冲突，以上数据同步方法将导致两个中心的数据不一致。

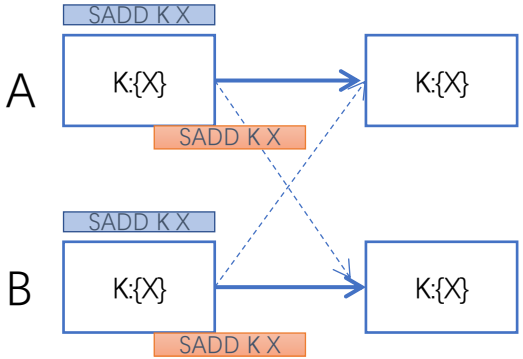
一致性 ✚ 高性能，低延迟 ⇨ CRDT

CRDT是一类实现多中心**无协作**并发更新，能保证**最终一致**的数据结构。

- CRDT
  - op-based(CmRDT)
    - 中心间只传操作日志
    - 典型的比如**op-based counter**：
      - *DECR counter 10*
      - *INCR counter 15*
      - 无论以何种顺序复制，最终*counter*结果为5
      - 但由于日志回放非幂等，必须保证日志传输**不丢不重**
  - state-based(CvRDT)
    - 中心间传操作日志和数据状态元信息
    - 各中心merge本地数据达到最终一致
    - G-Set
    - 2P-Set
    - OR-Set
    - LWW-element-Set

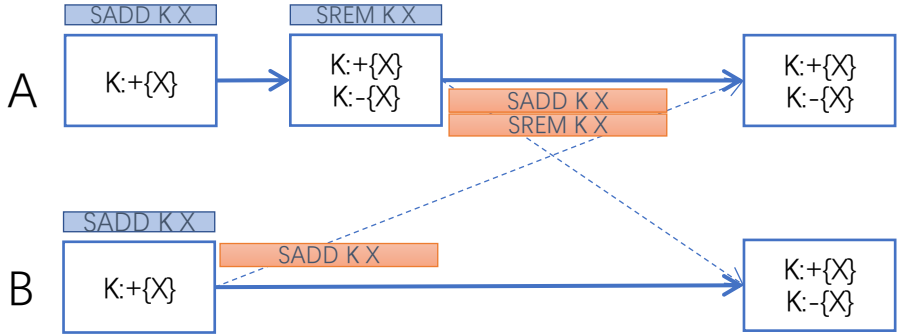
UPRedis最终采用了LWW-Element-Set+ op-based counter方案

G-Set (Grow-only Set)



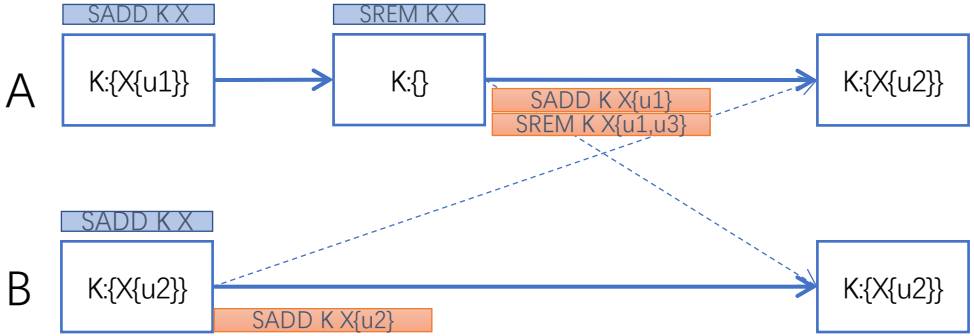
只支持添加元素，不支持删除元素。

2P-Set (Two-Phase Set)



2P-Set采用了remove优先策略，删除的元素不能重新添加。

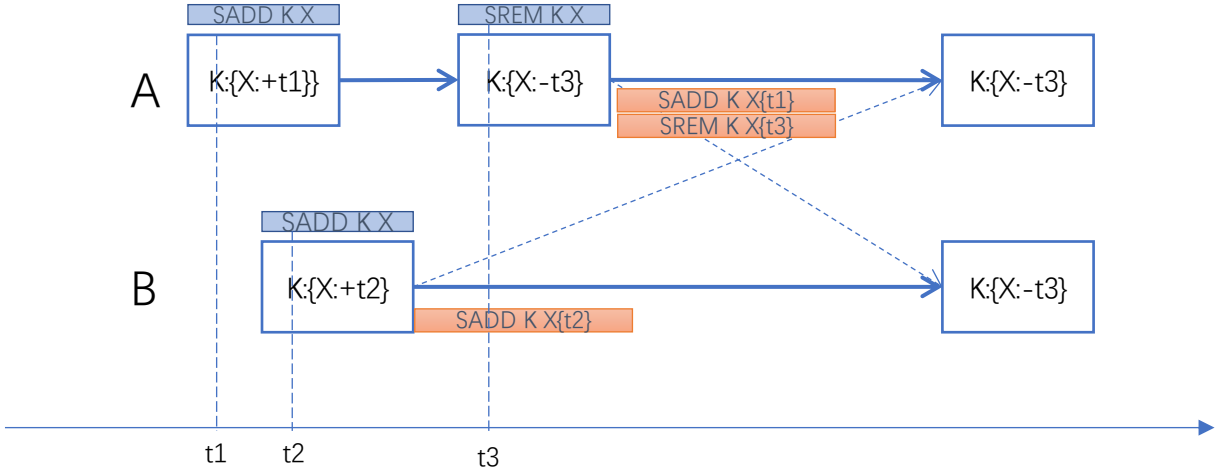
OR-Set (Observed-Remove Set)



添加元素：添加元素和uniqe tag
删除元素：删除元素以及当前该元素所有uniqe tag

与G-Set,2P-Set相比，OR-Set功能无明显短板；阿里云和redis-labs都使用了OR-Set作为最终一致性的算法。

LWW-Element-Set (Last-Write-Wins-Element-Set)



添加元素：添加元素和当前时间戳
删除元素：更新时间戳并标记元素已删除(tombstone)

LWW-Element-Set没有明显短板，UPRedis采用的就是LWW-Element-Set算法。

目录

LWW-Element-Set

- 集合每个元素都带有更新或者删除时间
- 回放异地传输来的日志与时，数据以更新时间最新为准(LWW)
- 保存tombstone主要为了防止已删除数据复活

A中心	B中心
SADD myset e	SADD myset e
SREM myset e	
--sync--	
myset{e}	myset{}

LWW-Element-Set

对于**任意中心x的任意key**的最终合并结果：

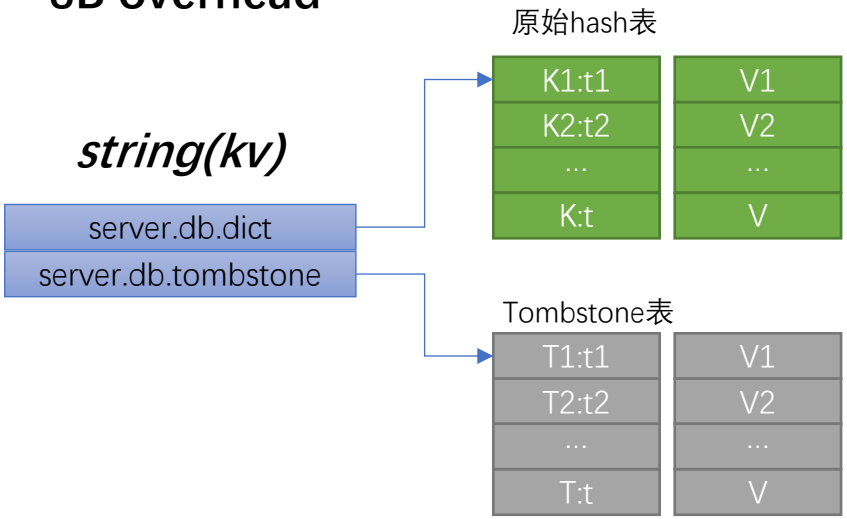
a) 由于LWW的偏序性(semi-lattice)，因此**最后应用到该key的是所有日志里最新的一条**

b) 由于操作O_{xn}的决定性(determitive，i.e. SET/DEL命令不像INCR命令需要基于前置值)

可以保证**任意中心x的任意key**最终值一致。

A中心	B中心	…	D中心
O_{a1}	O_{b1}	O_{c1}	O_{d1}
O_{a2}	O_{b2}	O_{c2}	O_{d2}
…	…	…	…
O_{ai}	O_{bj}	O_{ck}	O_{dl}
-- sync --			
$Permute(O_{xn} \mid x \in \{a, b, \dots\}, n \in \{1,2, \dots\}$			

8B overhead

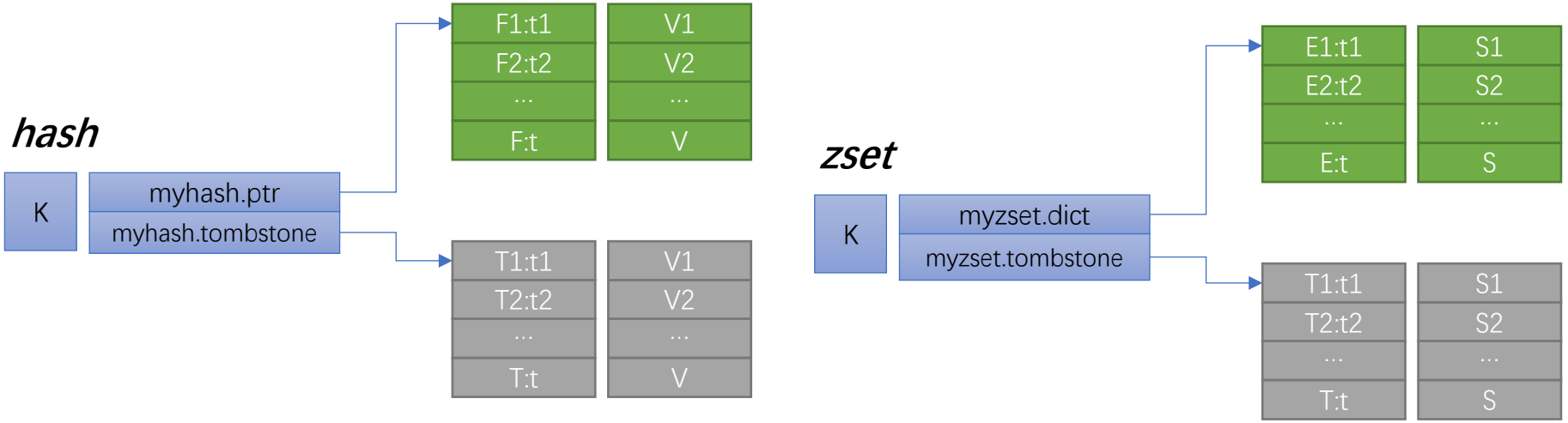



```
typedef struct redisObject {
    unsigned type:4;
    unsigned encoding:4;
    unsigned lru:REDIS_LRU_BITS;
    int refcount;
    union {
        uint64_t ts;
        void *tombstone;
    } crdt; /* crdt元信息 */
    void *ptr;
} robj;
```

原始hash表每个robj多8B overhead（用于存更新时间）

Tombstone表保存该历史删除数据和删除时间

8B overhead



hash/zset与string(kv)的实现类似，都有8B overhead用于存更新时间或者tombstone。不同点仅在于存储tombstone的字典位置不同

数据结构	是否支持	编码	实现算法
string(hyperloglog)	√	int, embstr, raw	op-based
string(kv)	√	int, embstr, raw	LWW-Element-Set
string(bit)	×	int, embstr, raw	-
string(counter)	√	int, embstr, raw	op-based
hash	√	ziplist,hashtable	LWW-Element-Set
set	√	intset, hashtable	LWW-Element-Set
zset	√	ziplist, skiplist	LWW-Element-Set
geo	√	ziplist, skiplist	LWW-Element-Set
list	×	quicklist	-

- UPRedis采用了op-based counter + LWW-Element-Set。
- CDRT不是银弹，并不能完美支持所有Redis命令。
- 比如string命令不能混用counter和kv类型命令，此外支持的编码也有有限的。

限制和注意点

- 跨类别不能保证最终一致

A中心	B中心
INCR counter	INCR counter
DEL counter	
-- sync --	
counter:1	

不同CRDT类别算法可能不一致，不能保证最终一致

- Multi-key命令

A中心	B中心
SET a a	SET b b
DEL a b	
-- sync --	

A中心的DEL a b命令本身删除了B中心的b

- lua、expire、eviction不保证最终一致

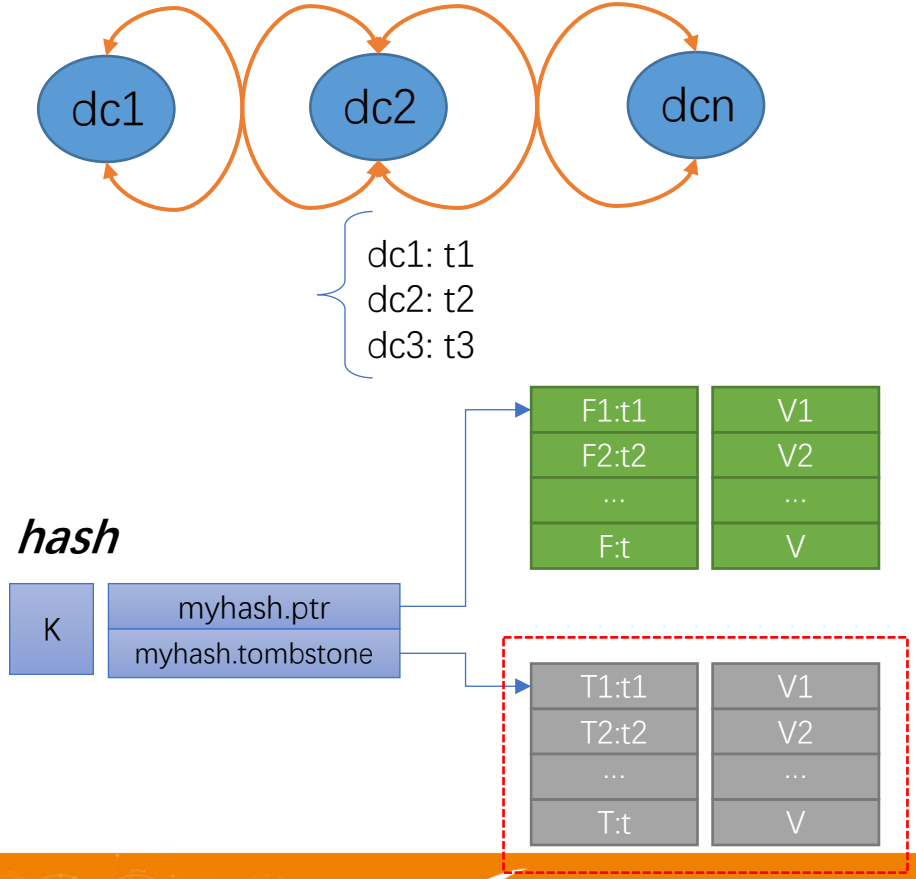
Tombstone的GC策略

barrier=min (dc1.ts,dc2.ts,…dcn.ts)

a) 数据中心的时间单调递增

b) 日志传输通道能保证有序且不丢不重

时间戳小于barrier的tombstone都可以安全GC

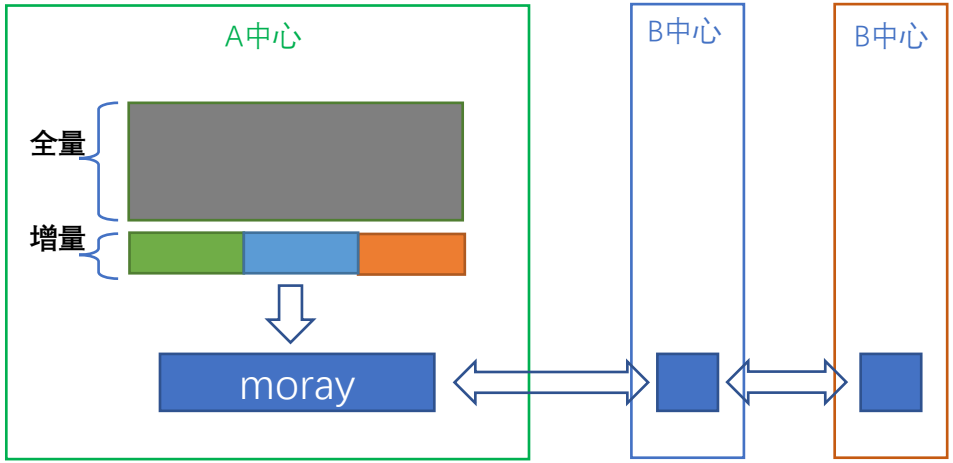


一些优化：

- 和超时机制类似，采用active+inline两种触发策略
- 中心间定时同步时间，防止灾备场景造成无法GC

全量方案

复制协议+增量日志



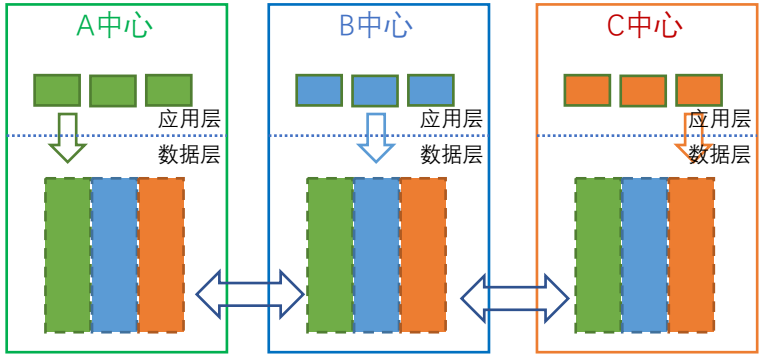
主要问题：

1. 解析全量RDB文件 ⇨ 根据RDB协议解析生成AOF-BINLOG日志

2. 全量和增量的接续 ⇨ 全量数据使用**fake server-id(0)**+自增opid，避免没有server-id造成循环回传

目录

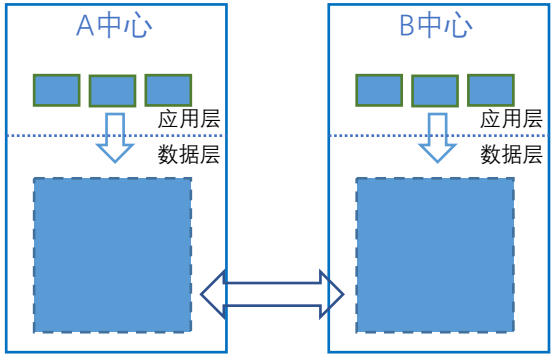
应用场景1：单元化



特点：各中心数据切片，中心间相互备份

场景：业务上可以数据切片，比如云闪付？？

应用场景2：多活



特点：各中心可能写同一数据，CRDT或者业务自行处理

场景：业务上无法数据切片，比如全渠道某系统


银联云，具有四大特色
稳 定 可 靠
Stable and reliable
金 融 级 安 全
Financial security
开 放 共 享
open and share
自 主 可 控
Independently controllable

中国银联期待你的加入

数据库开发方向

开源组件方向

THANKS