浪淘金
LANG TAO JIN CO.,LTD.

只为有效果的广告付费

深入了解Redis

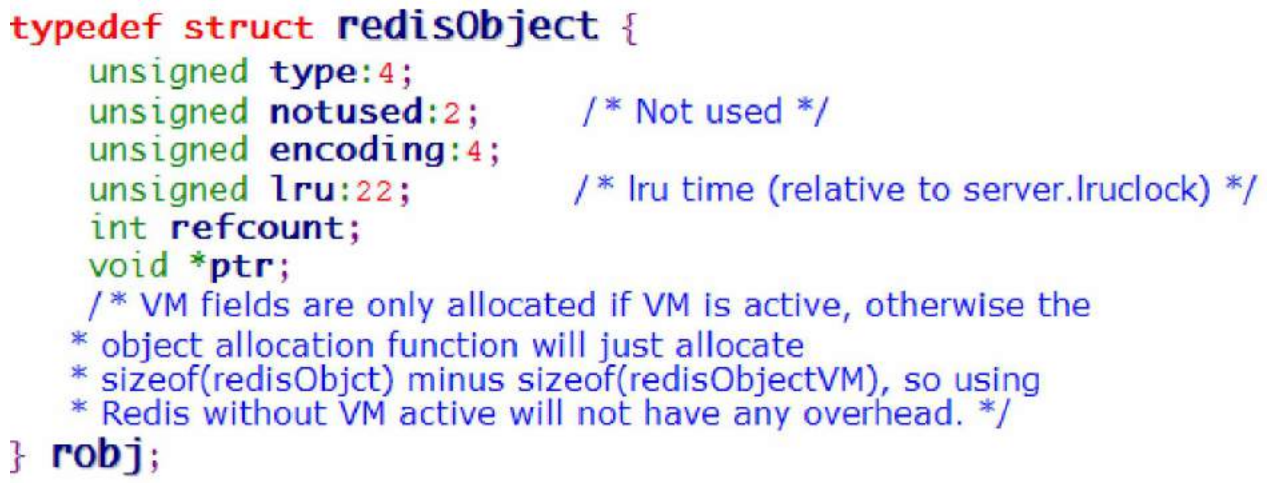
宋传胜
Chuansheng.song@langtaojin.com

简单介绍

- 文本协议 memcached类似
- KEY 可打印字符
- VALUE支持的类型
  - STRINGS
  - LIST
  - SET
  - SORTED SET
  - HASH
- 高性能 (100k+ SET/80k+ GET)/s
- 序列化支持
- 主从同步支持
- 客户端自己实现sharding

基本数据结构

- RedisObject (Redis.h)
  - key/value对象的基础



```
typedef struct redisObject {
    unsigned type:4;
    unsigned notused:2;     /* Not used */
    unsigned encoding:4;
    unsigned lru:22;        /* lru time (relative to server.lruclock) */
    int refcount;
    void *ptr;
    /* VM fields are only allocated if VM is active, otherwise the
   * object allocation function will just allocate
   * sizeof(redisObjct) minus sizeof(redisObjectVM), so using
   * Redis without VM active will not have any overhead. */
} robj;
```

KEY

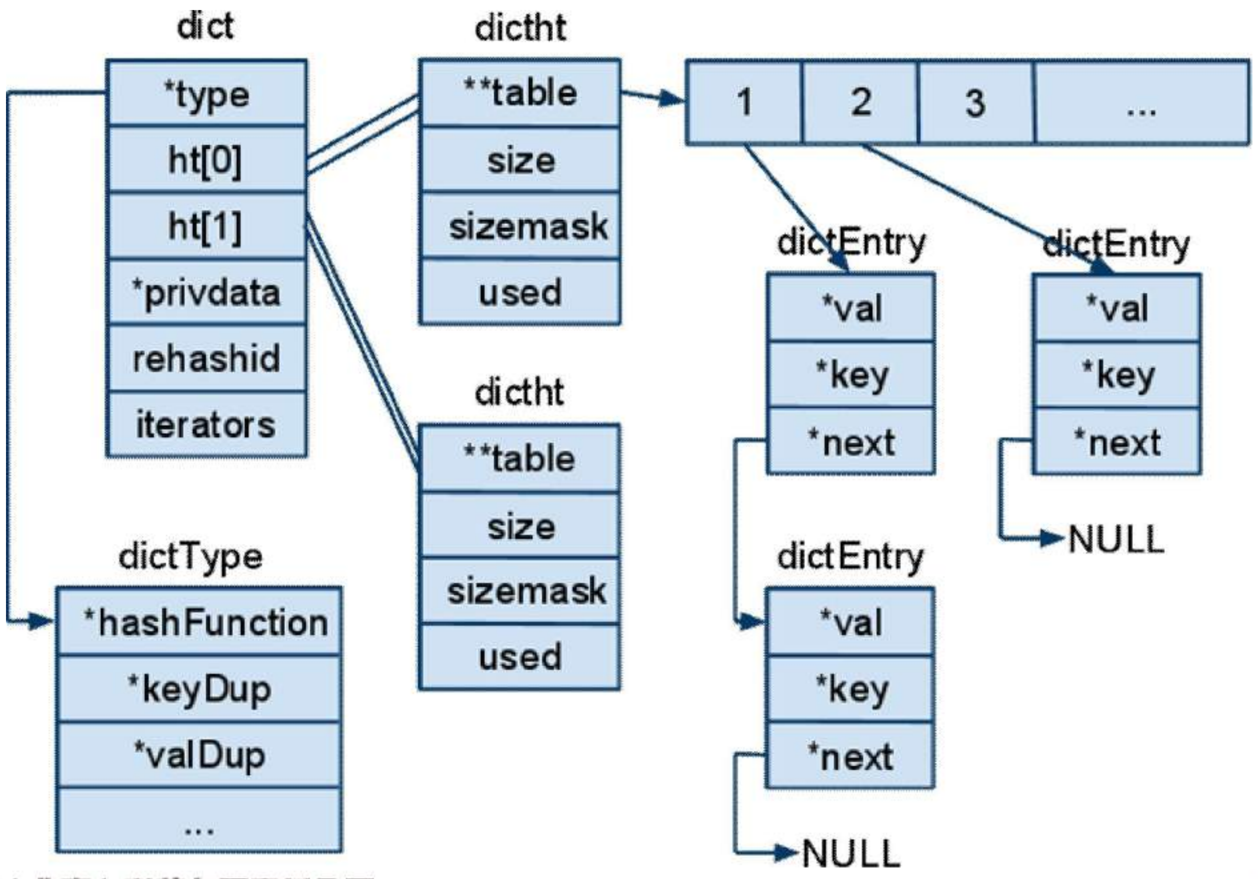
- 基本命令
  - get/set/exists O(1)
  - setnx/randomkey/rename/renamenx/dbsize/type
  - keys pattern
  - del 一次删除多个key
  - expire key seconds 指定过期时间
  - ttl key 剩余过期时间
  - select db
  - move key newdb
  - flush db/flushall

EXPIRE AND LRU CACHE

- Volatile Key (Db.c)
  - When the key is set to a new value using the SET command, or when a key is destroyed via DEL, the timeout is removed from the key.
- Enhanced Lazy Expiration algorithm
  - Redis does not constantly monitor keys that are going to be expired. Keys are expired simply when some client tries to access a key, and the key is found to be timed out.
- Work as LRU cache (memcached)
  - Maxmemory/maxmemory-policy
  - When memory limit was already reached, server will remove some old data deleting a *volatile key*, even if the key is still far from expiring automatically.
  - Random get keys, delete by lru rules.

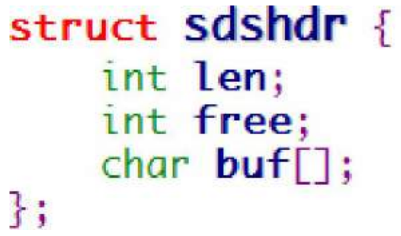
基本数据结构

- Hashtable(Dict.c)



基本Value数据结构

- STRING（sds.c）



```
struct sdshdr {
    int len;
    int free;
    char buf[];
};
```

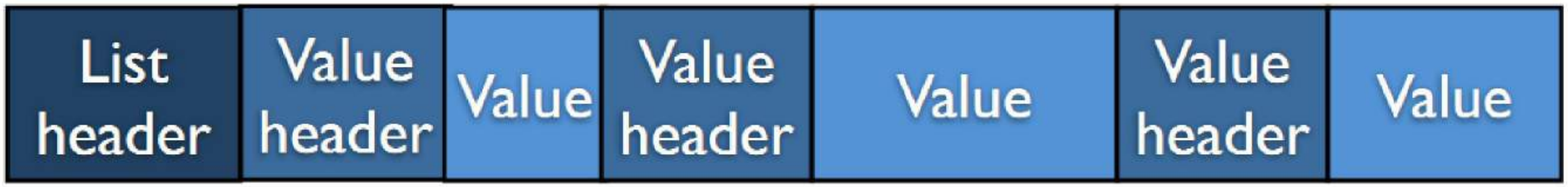
  - set/setnx/get/getset/mget/mset/msetnx
    - nx – not exists
    - m -- multiple
  - incr/decr/incrby/decrby
  - getrange/append

基本Value数据结构

- LIST (T_list.c)
  - REDIS_LIST 类型, 如果其 entry 小于配置值: list-max-ziplist-entries 或 value字符串的长度小于 list-max-ziplist-value， 使用ziplist数据结构， 否则使用标准的Doubly linked list
  - (l/r)push/(l/r)pop/llen O(1)
  - b(l/r)pop支持阻塞等待， 避免了轮循
  - lrange/ltrim/lrem/lset/rpoplpush

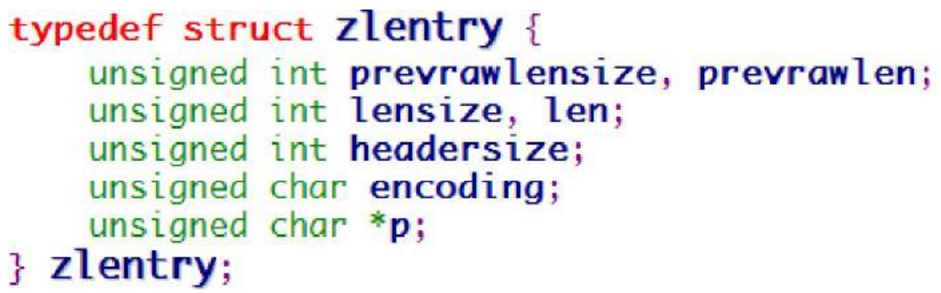
LIST

- Ziplist (Ziplist.c)
  - O(mem_size) add/delete
  - list-max-ziplist-entries (default: 1024)
  - list-max-ziplist-value (default: 32)



LIST

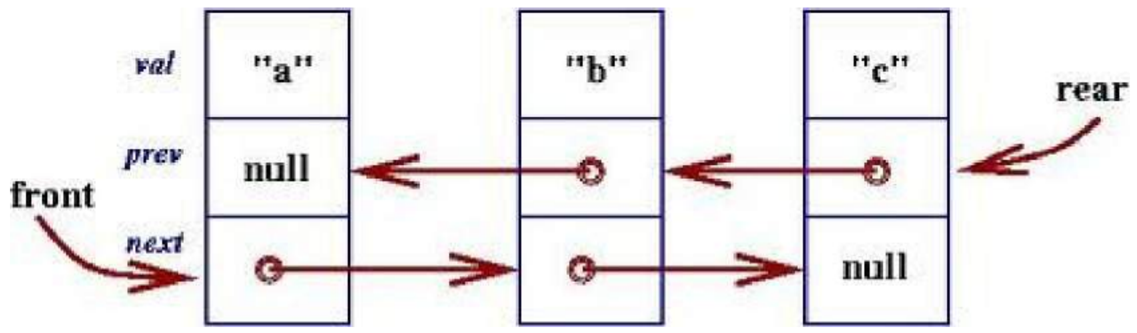
- Ziplist （continue）
  - zlentry (unencode structure)



```c
typedef struct zlentry {
    unsigned int prevrawlensize, prevrawlen;
    unsigned int lensize, len;
    unsigned int headersize;
    unsigned char encoding;
    unsigned char *p;
} zlentry;
```

LIST

- Doubly linked list (Adlist.c)



基本Value数据结构

- SET (T_set.c) – hashtable + Intset
  - String的无序集合
  - sadd/srem/sismember/scard O(1)
  - spop/srandmember
  - smove
  - smembers
  - sinter(store) O(C)
  - sunion(store)/sdiff(store) O(N)

INTSET

- INTSET (Intset.c)
  - 都是整型时数据结构退化成排序的intset
  - Good fit for size up to 20-50K
  - set-max-intset-entries (default: 4096)
  - O(n) search
  - O(log(n) + mem_size) add/del

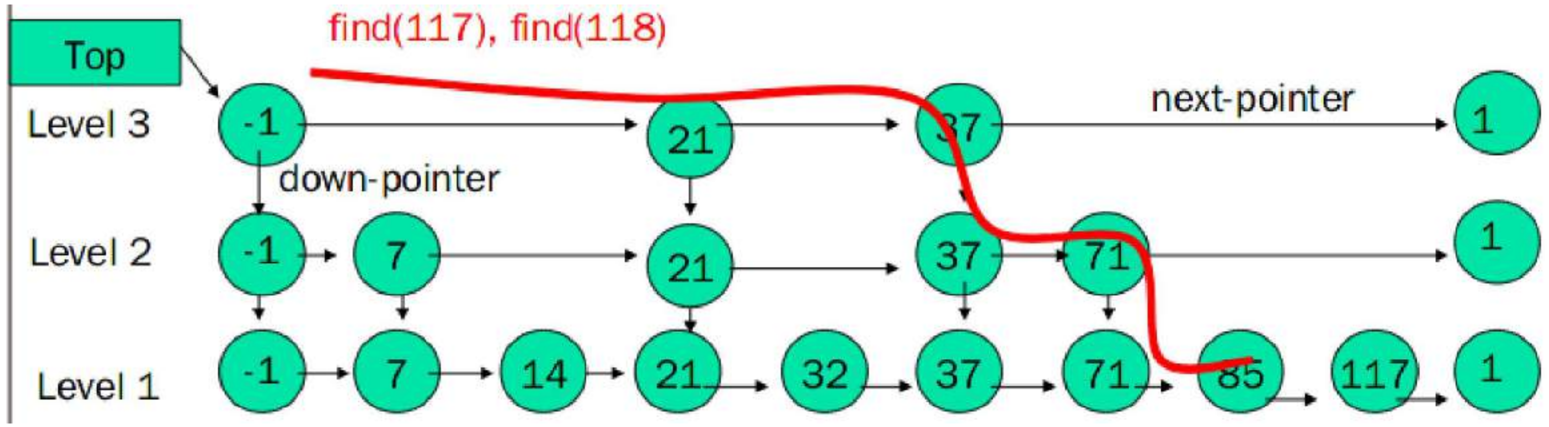
```
typedef struct intset {
    uint32_t encoding;
    uint32_t length;
    int8_t contents[];
} intset;
```

基本Value数据结构

- SORTED SET(T_zset.c) hashtable + skip list
  - 按照key的score排序
  - zadd/zrem/zrank/zrevrank O(log(n))
  - zcard O(1)
  - zincrby 修改score
  - zrange/zrevrange/zrangebyscore/zscore O(log(N)+M)
  - zremrangebyrank/zremrangebyscore O(log(N)+M)

SKIP LIST

- Skip List (T_zset.c)
  - skip list → redblack tree → AVL tree (more balanced)
  - lockfree/concurrency

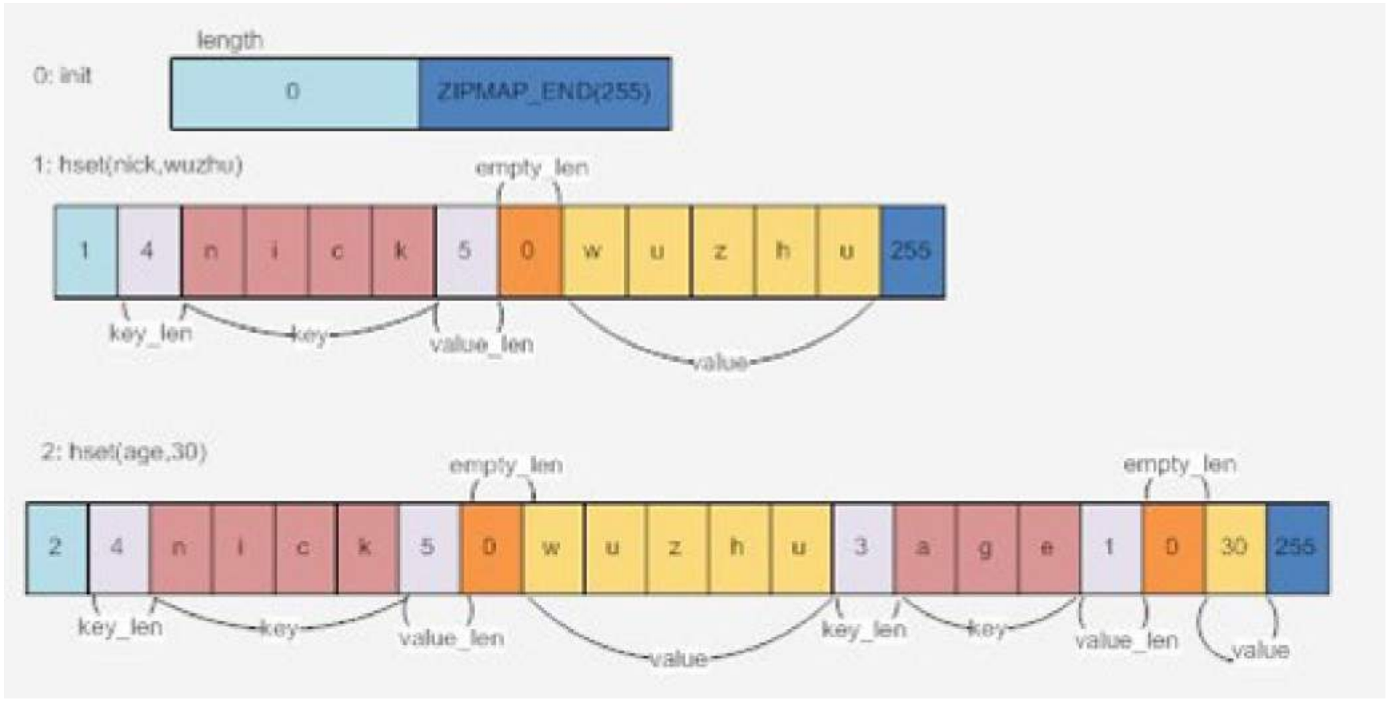


基本Value数据结构

- HASH (T_hash.c) zipmap + hashtable
  - 如果其 entry 小于配置值: hash-max-zipmap-entries 或 value字符串的长度小于 hash-max-zipmap-value， 使用zipmap数据结构
  - Value只能是string类型
  - hset/hget/hexists O(1)
  - hmset/hexists/hincrby/hdel/hlen
  - hmget O(N)
  - hkeys/hvals/hgetall

ZIPMAP

- zipmap
  - O(n) search O(mem_size) add/delete



内存管理

- zmalloc.c
  - 简单内存管理
  - 支持tcmalloc USE_TCMALLOC
  - zmalloc(size_t size) → malloc(size+PREFIX_SIZE)
  - fragmentation_ratio = zmalloc_get_rss/zmalloc_used_memory

PUBSUB

- SUBSCRIBE/UNSUBSCRIBE/PSUBSCRIBE/PUNSUBSCRIBE
- PUBLISH
- 连接断开后信息丢失

TRANSACTION

- MULTI （不支持CAS）
  - MULTI
  - SET foo 0
  - INCR foo
  - EXEC
- WATCH （Check and Set ）
  - WATCH theKey
  - v = GET theKey
  - MULTI
  - SET theKey v+1
  - EXEC
- UNWATCH/DISCARD

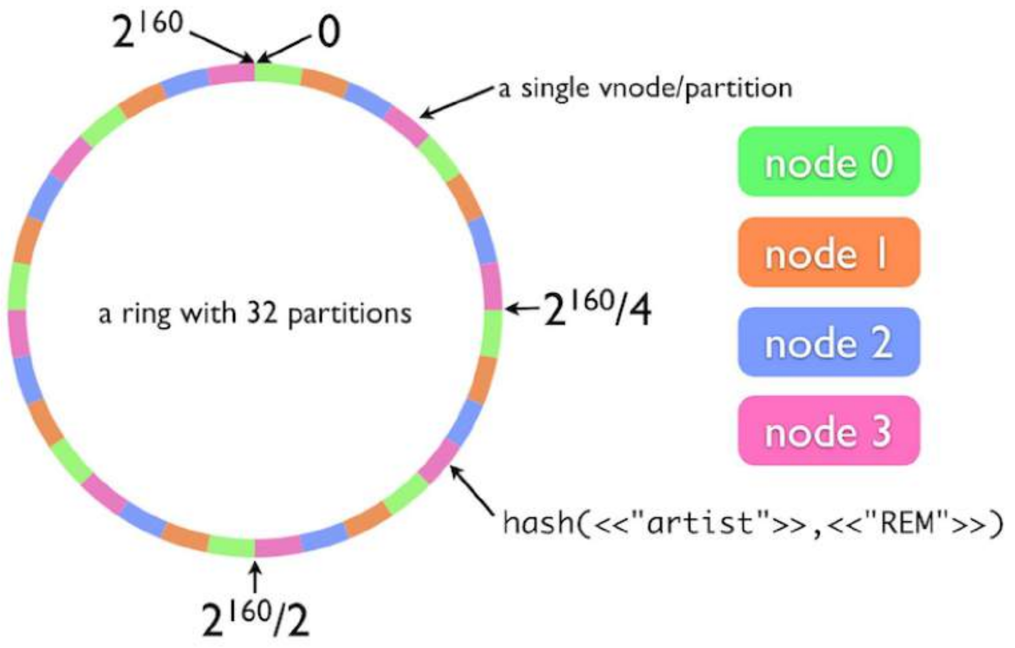
提升网络效率、减少内存占用

- MGET/MSET
- PIPELINING

  -
    ```
    $ (echo -en "PING\r\nPING\r\nPING\r\n"; sleep 1) | nc localhost 6379
    +PONG
    +PONG
    +PONG
    ```
- 压缩数据，触发ziplist/zipmap/intset
- 使用 GETBIT/SETBIT/GETRANGE/SETRANGE 压缩多个属性
- 使用HASH
- 使用32位redis + sharding

Client Side Sharding

- Consistent Hashing
  - *Redis Sharding* at Craigslist

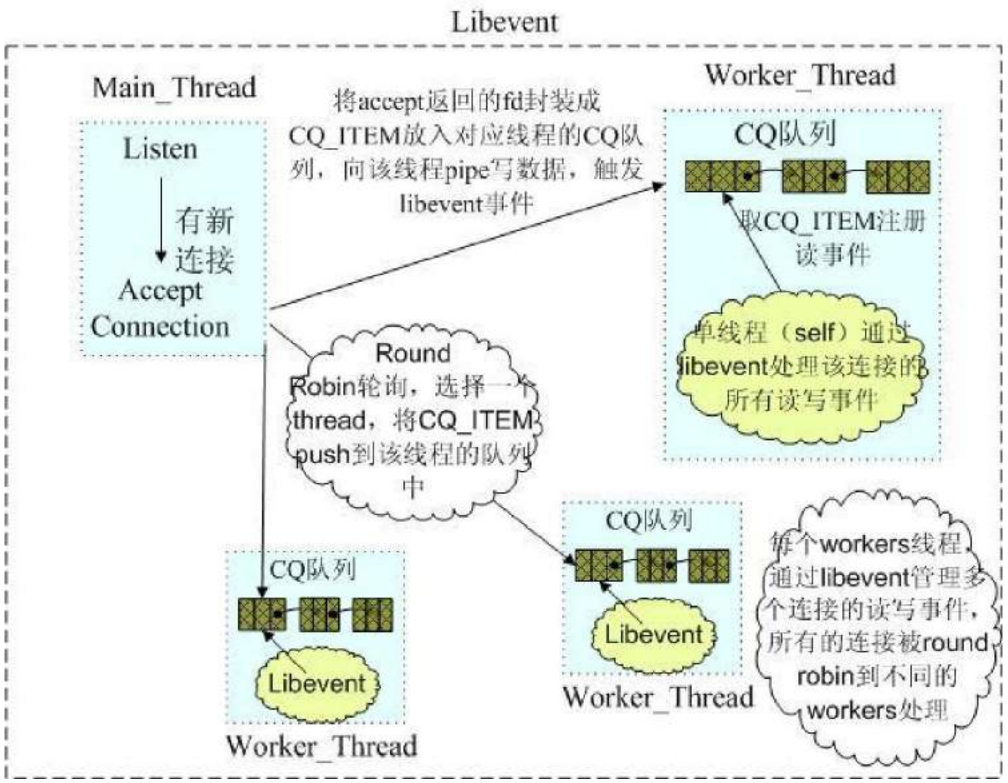


网络层

- Anet.c
- 事件驱动 ae.c → ae_epoll.c/ae_kqueue.c/ae_select.c
- 非阻塞单线程
- 支持timer
- 在VM场景下有可能多线程

网络层对比

- memcached/varnish/scribed的网络IO模型
  - 多线程，非阻塞IO复用


Libevent
Main_Thread
Listen
有新连接
Accept Connection
将accept返回的fd封装成CQ_ITEM放入对应线程的CQ队列，向该线程pipe写数据，触发libevent事件
Worker_Thread
CQ队列
取CQ_ITEM注册读事件
单线程（self）通过libevent处理该连接的所有读写事件
Round Robin轮询，选择一个thread，将CQ_ITEM push到该线程的队列中
CQ队列
Libevent
Worker_Thread
CQ队列
Libevent
Worker_Thread
每个workers线程，通过libevent管理多个连接的读写事件，所有的连接被round robin到不同的workers处理

网络层对比

- lighttpd/nginx的网络IO模型
  - 多进程，单线程非阻塞IO
- apache的MPM模型
  - 多进程prefork，多线程

PERSISTENCE

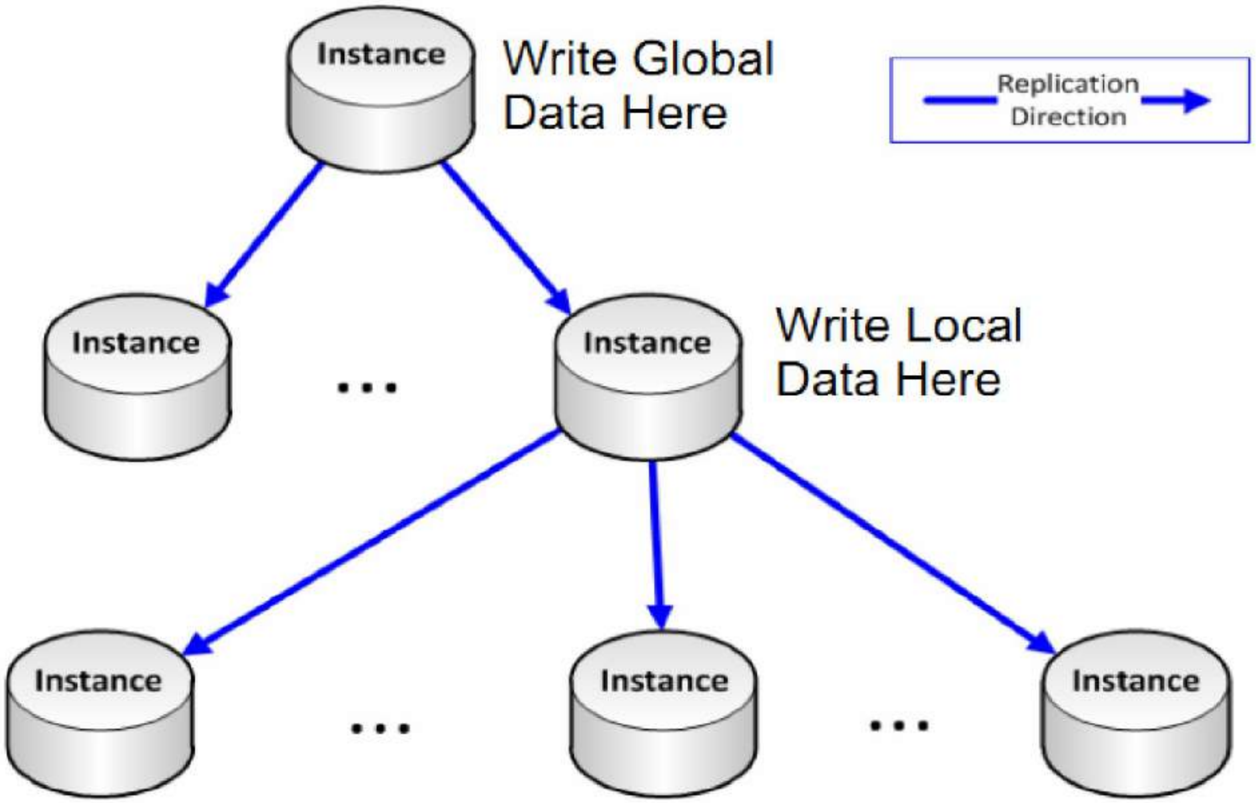
- SNAPSHOT (Rdb.c)
- 可以关闭
- BGSAVE or save xseconds ychanges
- fork
  - parent - disable rehash, periodic check child on serverCron (100ms)
  - child - copy-on-write, save whole db, exit
- Need to turn on the overcommit_memory setting if you want to deal with a dataset more than 1/2 of RAM, which contradicts our habit to battle against OOM killer as a database administrator.

Append Only File

- Append to log file on every change (Aof.c)
- fsync() policy
  - always/os decide/every second
  - Always with battery backed raid controllers
  - Every second by default (innodb_flush_log_at_trx_commit=2 in mysql)
- Compact aof file
  - BGREWRITEAOF/REWRITEAOF
- Fork
  - child – write new aof in temporary file
  - parent – write change in both old aof file and memory buffer, append memory buffer to temporary file when child done, rename temporary to aof file.
- redis-check-aof --fix <filename>

Replication

- 支持多级同步 (Replication.c)



Replication

- 支持AUTH
- 起动过程
  - slave
    - SYNC
    - Wait
  - master
    - Issue a BGSAVE, or wait if BGSAVE in progress

Virtual Memory

- Redis VM is now deprecated. Redis 2.4 will be the latest Redis version featuring Virtual Memory!
- Why not OS VM
  - Varnish What's wrong with 1975 programming ?
  - Redis What's wrong with 2006 programming?
    - Blocking when page fault
    - 4k pages granularity
    - Optimal disk persistent format, 10:1
- Keys can't be swapped out
- vm is read only when BGSAVE/BGREWRITEAOF in progress
- vm-max-memory/vm-pages/vm-page-size

Virtual Memory

- Blocking vm
- Threaded vm
  - Before command is executed, check the value
  - Block client and load, then signal by unix pipe
  - Continue
- Recommended filesystem – ext3 or any other file system with good support for *sparse files*
- redis-stat vmstat
- InnoDB innodb_buffer_pool_size – swap and durability

Future

- Diskstore (diskstore.c dscache.c)
  - All data on disk, B+tree in future
  - Swaps data into memory when needed
- Leveldb/InnoDB Change Buffer
  - LSM tree

其他相关的代码

- quicksort (pqsort.c)
- lzf compression (lzf_c.c lzf_d.c)
- Sha1 (Sha1.c)
- Syncio.c
- SORT (Sort.c)

线上Redis集群情况

- 16台服务器， 兼做后端服务器
- 45个Redis Instance进程， 三组服务
- 总内存 77G VIRT，71G RSS
- 使用了主从复制、snapshot持久化，VM支持， 客户端sharding、pipelining、hash优化

Redis应用

- Resque – message queue of github (celery)
- Nginx HttpRedis module
- Redweb - Web administrative and query UI
- A fast, fuzzy, full-text index using Redis
- Redis-backed BloomFilter(s) in Ruby
- Soulmate : Auto Complete with Redis
- Rate limiting with Redis
- Redis Sharding
- Locking with setnx
- A Collection Of Redis Use Cases
- Radishapp Redis monitoring
- openredis.com hosting service

Contribute to Redis

- Fork redis on github
  - https://github.com/antirez/redis
- Choose branch and commit your changes
- Create a pull request
- Wait for approval
  - https://github.com/antirez/redis/pulls
- My contribute for read only slave
  - https://github.com/antirez/redis/pull/47

Resources

- DOCUMENATION
  - http://redis.io/documentation
  - http://antirez.com/
  - http://groups.google.com/group/redis-db
- 本文参考的Redis资料集合
  - http://www.delicious.com/fakechris/redis
  - http://www.delicious.com/tag/redis