大话Redis设计实现

“峰云就她了”

- xiaorui.cc

v 2016.6.5

redis协议

```
set name fengyun
```

```
*3\r\n
$3\r\n
SET\r\n
$4\r\n
name\r\n
$7\r\n
fengyun\r\n
```

```
if OK:
    +OK\r\n
else:
    -1\r\n
```

消息体数

```
get name
```

```
*2\r\n
$3\r\n
GET\r\n
$4\r\n
name\r\n
```

字节数

```
if OK:
    $7\r\n
    fengyun\r\n
else:
    $-1\r\n
```

数据结构

- string (sds)
- list (deque)
- hash (hash table)
- set (intset + dict)
- zset (skip list + hash table)

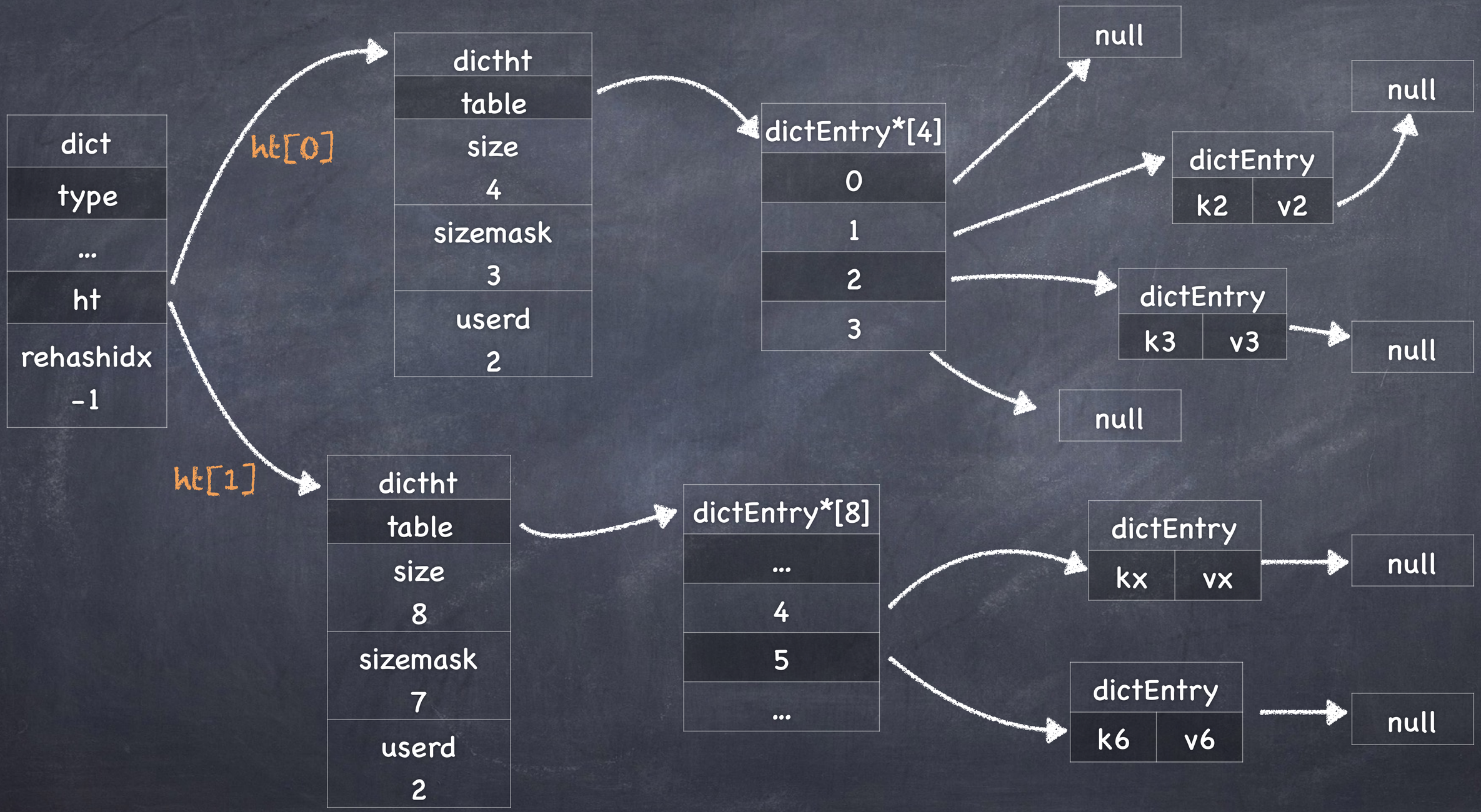
String (sds)

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

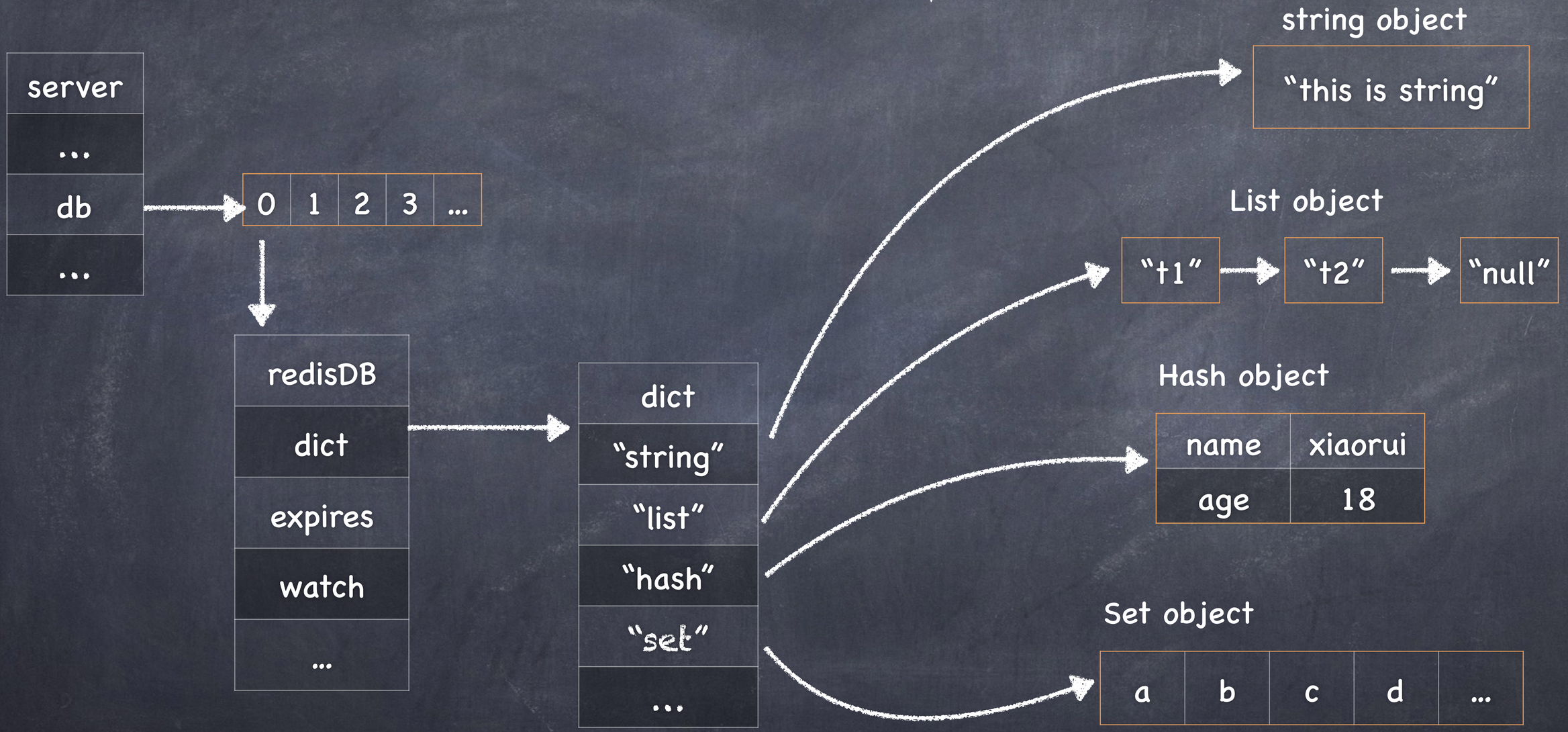
- 获取字符串长度，常数复杂度
- 增长时空间预分配
- 惰性回收

Hash table

- 通过渐进式rehash解决kv过多造成服务阻塞
- 流程:
  - 给ht[1]分配至少2倍于ht[0]的空间
  - 将ht[0]数据分批迁移到ht[1]
  - 清空ht[0], 将ht[0]指针指向ht[1],ht[1]指针指向ht[0]
- 每次rehash 100个桶
- 渐进式rehash
  - del \ find \ update in ht[0] and ht[1]
  - insert in ht[1]

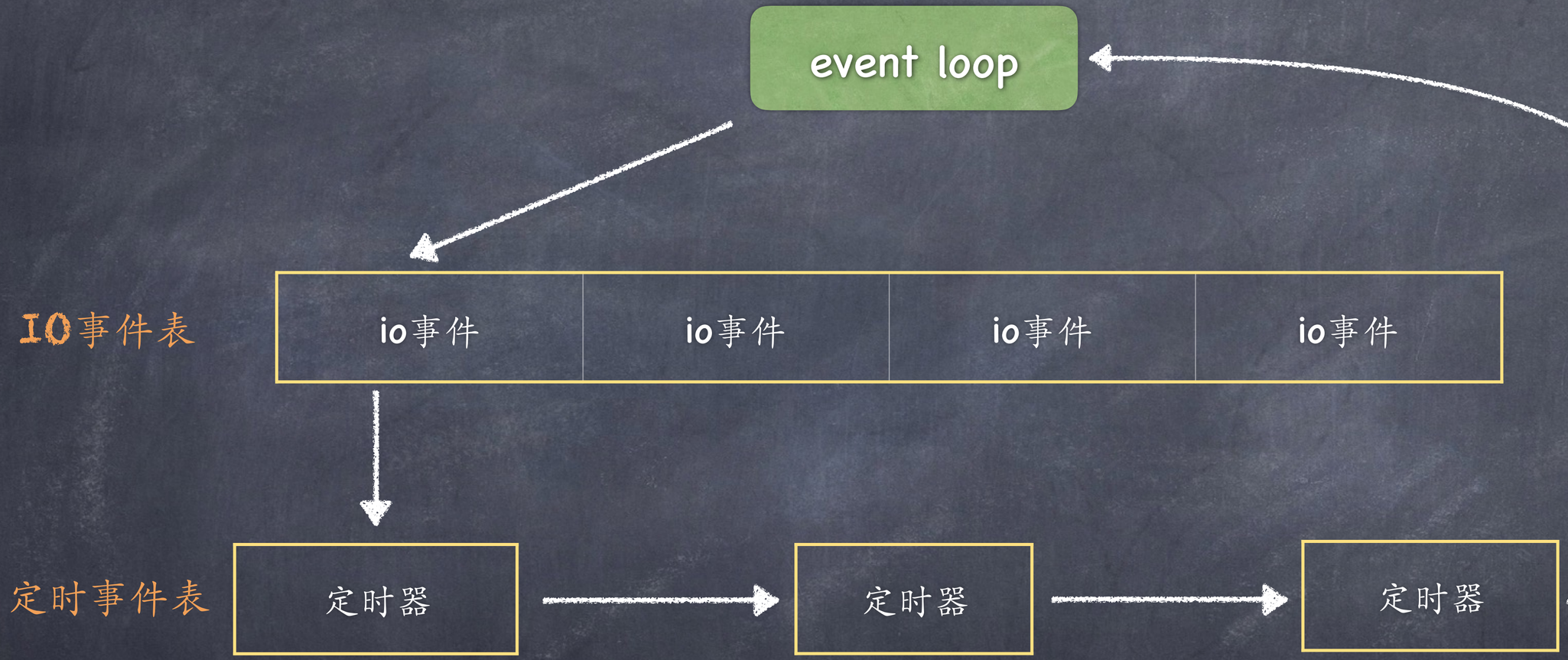

dict
type
...
ht
rehashidx
-1
ht[0]
ht[1]
dictht
table
size
4
sizemask
3
userd
2
dictEntry*[4]
0
1
2
3
null
null
dictEntry
k2
v2
null
dictEntry
k3
v3
null
dictht
table
size
8
sizemask
7
userd
2
dictEntry*[8]
...
4
5
...
dictEntry
kx
vx
null
dictEntry
k6
v6
null

RedisDB 内部结构



Redis event

- 事件类型
  - 时间类型　(serverCron \ aof \ rdb \ expires)
  - 文件类型　(client reuqest \ repl)
  - 串行
- epoll_create , epoll_ctl , epoll_wait
- 先文件类型, 再时间类型　(100ms cs)


event loop
IO事件表
io事件
io事件
io事件
io事件
定时事件表
定时器
定时器
定时器

Redis Rdb

- RDB是Redis物化数据，保证宕机恢复的一个
- 手段（会丢一部分最新数据）
  - 每个Redis实例只会存一份rdb文件
  - 可以通过Save以及BGSAVE 来调用
  - 二进制文件，lzf

RDB Format

"redis rdb"	"version"	"Data"	"Data"
"Data"	...	EOF	CHECKSUM

"expireTime"	"value type"	"key"	"value"

rdbSave

- 遍历每个db, 遍历每个db的dict, 获取每一个dictEntry
  - 获取Key之后查询expire, 如过期, 就舍弃
  - 将数据key, value, type, expiretime 等写入文件
  - 计算checksum, 通过rename交换旧的RDB文件

rdbLoad

- 遍历RDB文件的每一条数据
- 读取key, value, expiretime等信息，插入dict字典以及expire字典
- 校验checksum

Redis Aof

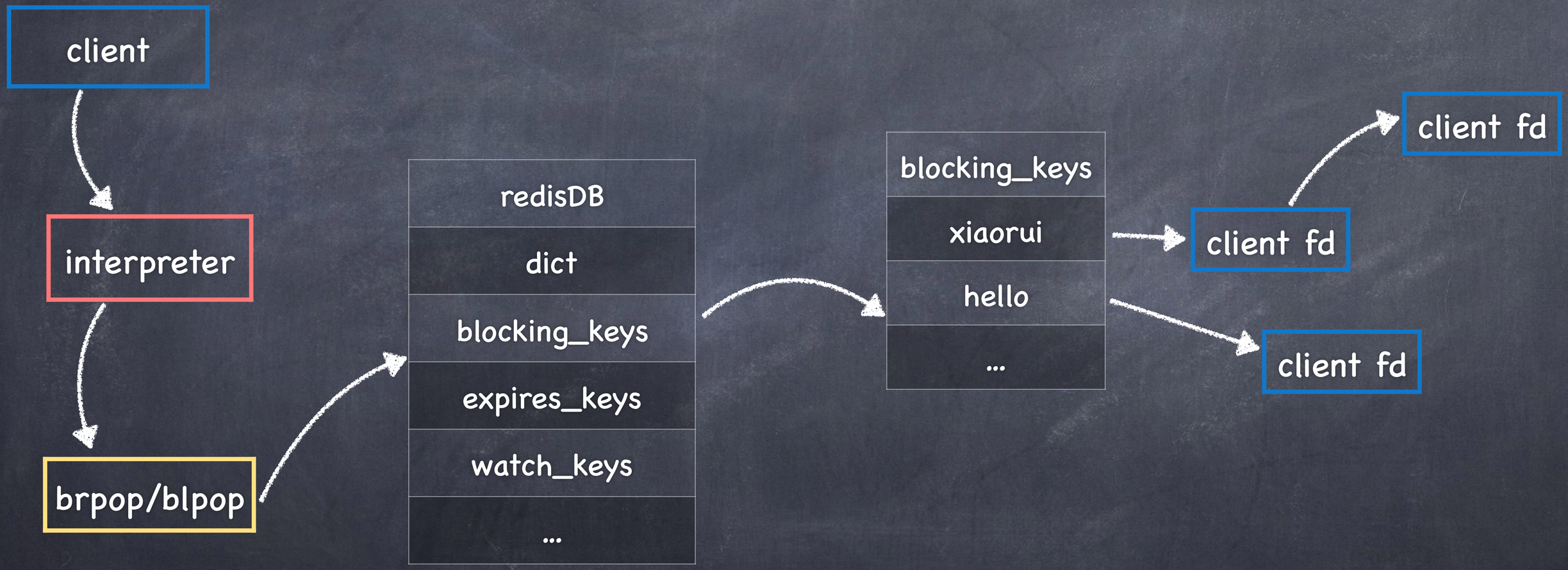
- 类似于BINLOG机制，可以做到不丢数据
- how ?
  - 每次数据操作都会调用flushAppendOnlyFile来刷新aof
- 每次操作都需要fsync，前台线程阻塞
- aof的内容就是redis标准协议
  - *3\r\n$5\r\nHMGET\r\n$5\r\nHENRY\r\n$10\r\nxiaorui.cc\r\n

Aof

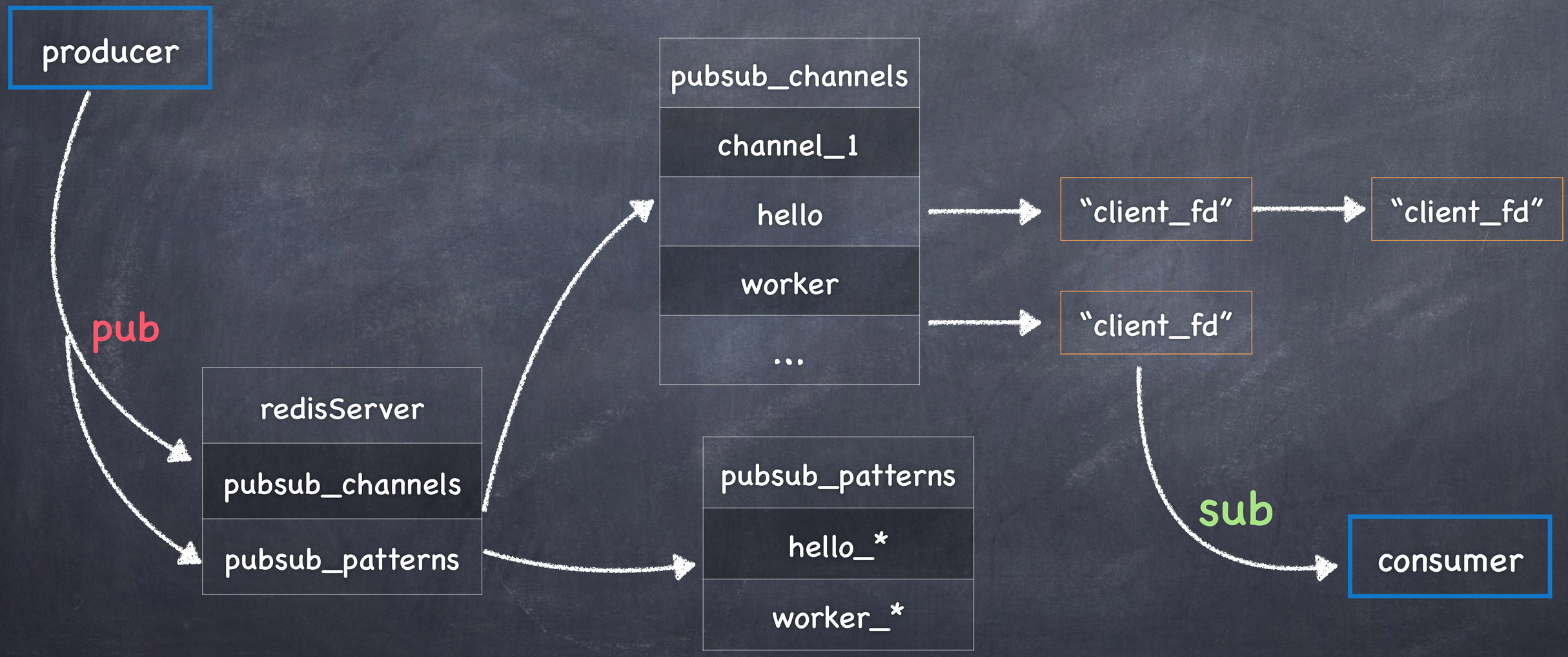
- 意义？
  - 将同一个key的反复操作，全部转为最后的值或multi集合（< 64）
- no-appendfsync-on-rewrite yes
  - 正导出rdb快照的过程中,要不要停止同步fsync
- auto-aof-rewrite-min-size 3000mb
  - aof文件,至少超过3000M时，再执行aof重写
- auto-aof-rewrite-percentage 80
  - aof文件大小比起上次重写时的大小,增长率80%时,执行aof重写

- AOF ReWrite的实现：
  - Fork一个子进程进行重写
  - 打开一个tmp aof文件准备写入
  - 遍历每个DB 及DICT中的每对key value，同时忽略过期键
  - 以协议命令的方式写入tmp aof文件中
  - 主进程中新的操作内容写到aof rewrite buffer中
  - 子进程重写完后，向主进程发送信号，主进程serverCron中将buffer的内容刷到新的aof文件中
  - 最后以rename的方式替换旧的aof文件

brpop/blpop


client
interpreter
brpop/blpop
redisDB
dict
blocking_keys
expires_keys
watch_keys
...
blocking_keys
xiaorui
hello
...
client fd
client fd
client fd

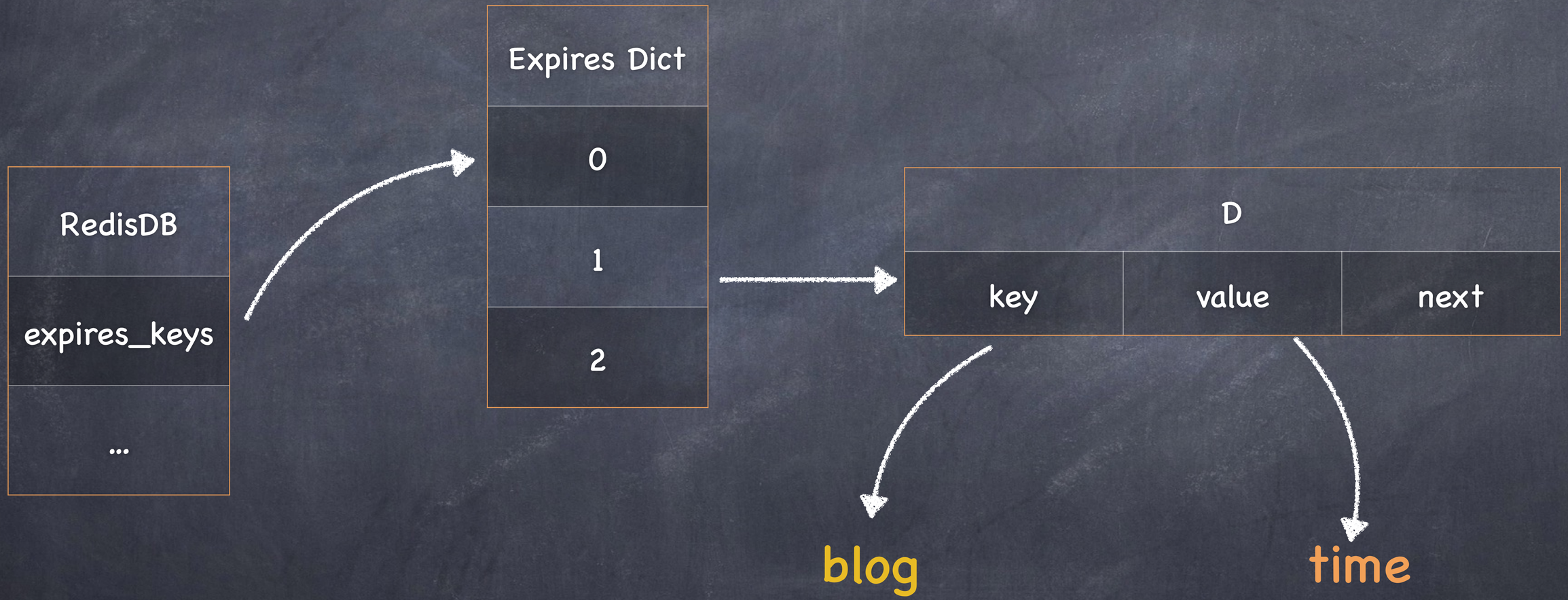
pubsub


producer
pub
redisServer
pubsub_channels
pubsub_patterns
pubsub_channels
channel_1
hello
worker
...
"client_fd"
"client_fd"
"client_fd"
pubsub_patterns
hello_*
worker_*
sub
consumer

expire realize

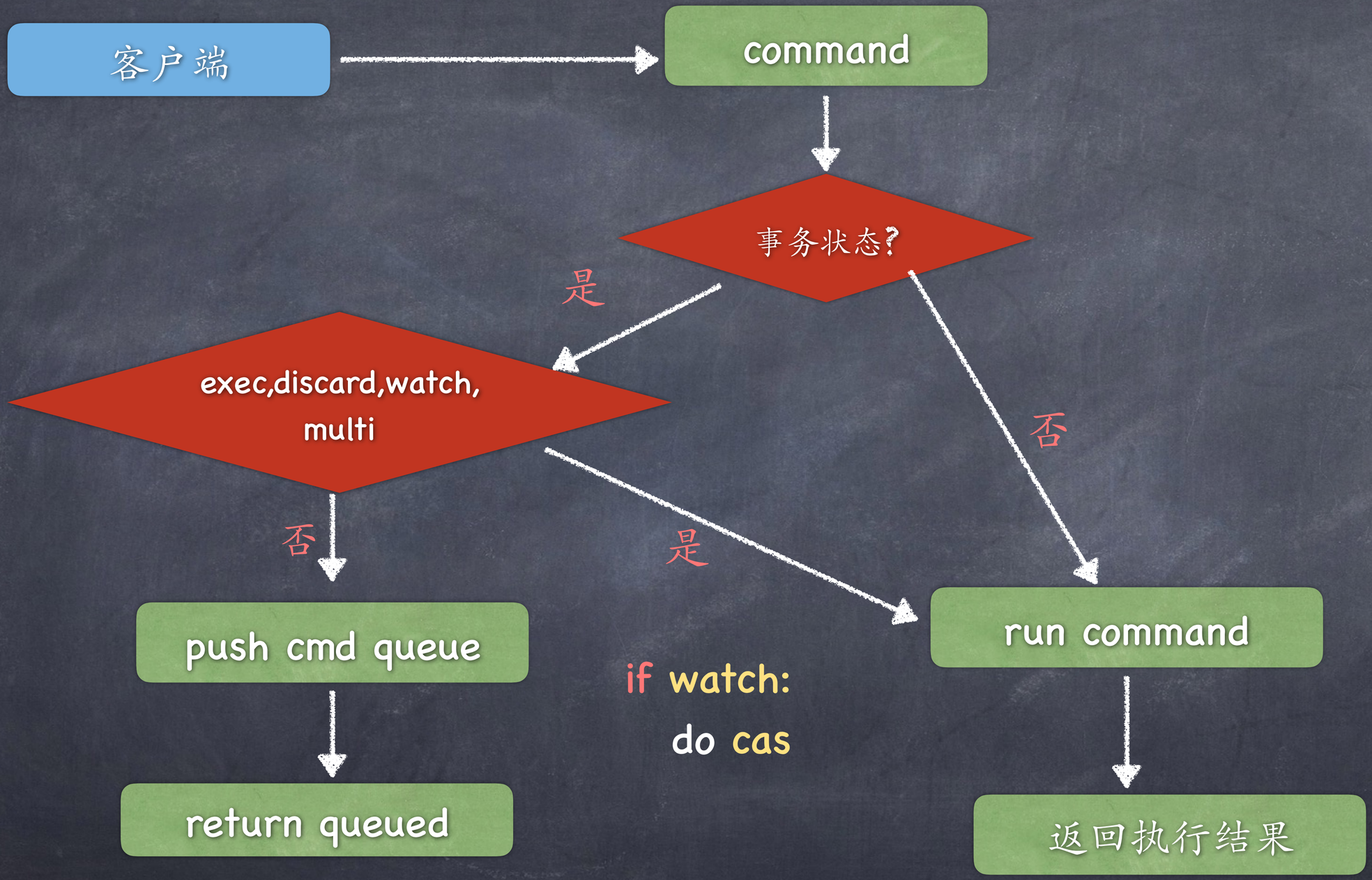
- 懒惰清理
  - 当发生读写的时候，会检查是否过期
- 主动清理
  - 定期轮询expire dict中的键是否过期
- slave屏蔽懒惰和主动清理，只等待psync .

expire realize



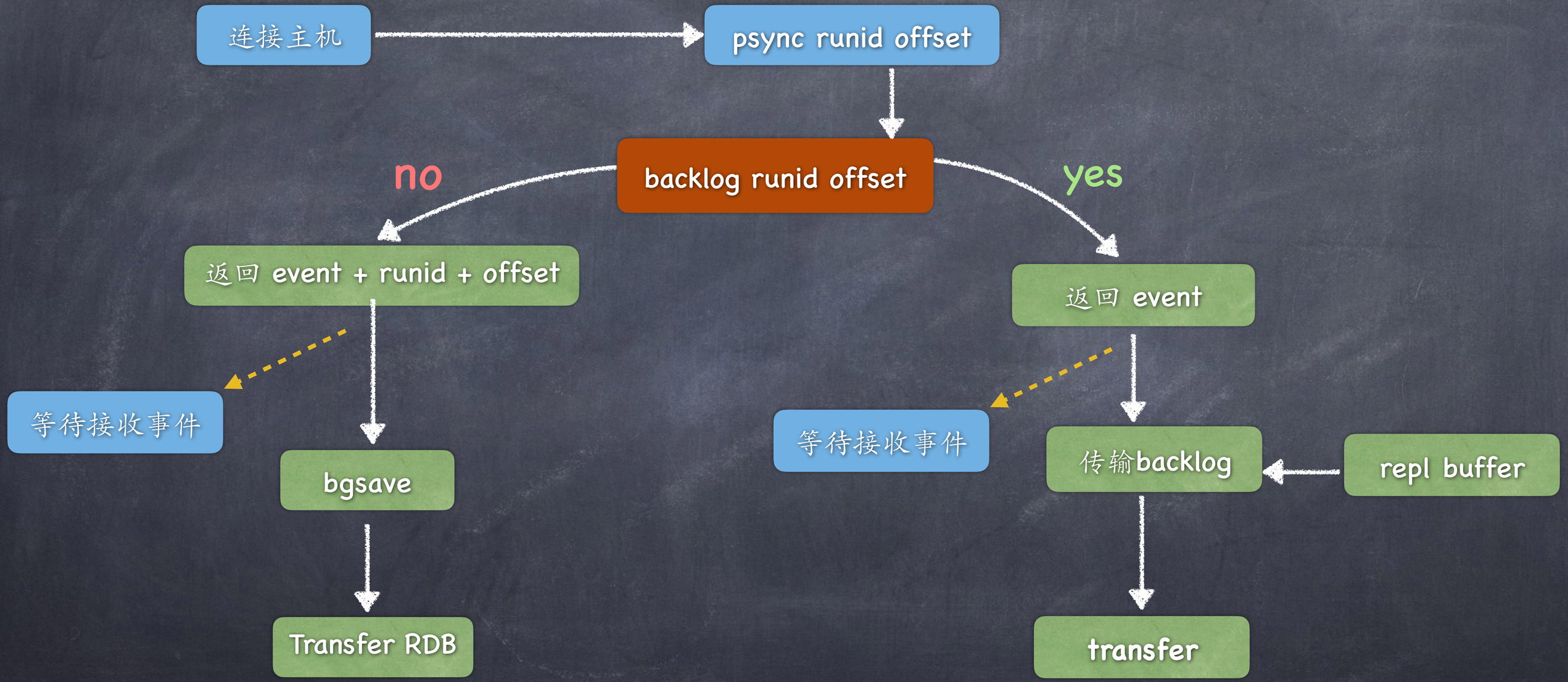
watch multi

- watch 乐观锁
- multi 原子性
- 流程
  - 事务标记开始
  - 命令入队
  - 事务执行
- 无rallback 、 无Lock


客户端
command
事务状态?
是
否
exec,discard,watch,
multi
否
是
push cmd queue
run command
if watch:
do cas
return queued
返回执行结果

主从同步

- fork bgsave ?
- fork child block ? bgsave block ?
- psync vs fullsync 区别 ?
- run_id标识slave, 为毛不适用fd ?
- offset
- replication backlog vs replication buffer
- 两个slave同时fullsync，共用一个RDB及buffer


连接主机
psync runid offset
backlog runid offset
no
yes
返回 event + runid + offset
返回 event
等待接收事件
bgsave
等待接收事件
传输backlog
repl buffer
Transfer RDB
transfer

改进

- transfer rdb, copy效率| 断连 | 落地
  - sendfile | 断点 | 内存
- expires keyes, random check
  - 二叉堆？

"Q & A"

- 2016. 06. 05