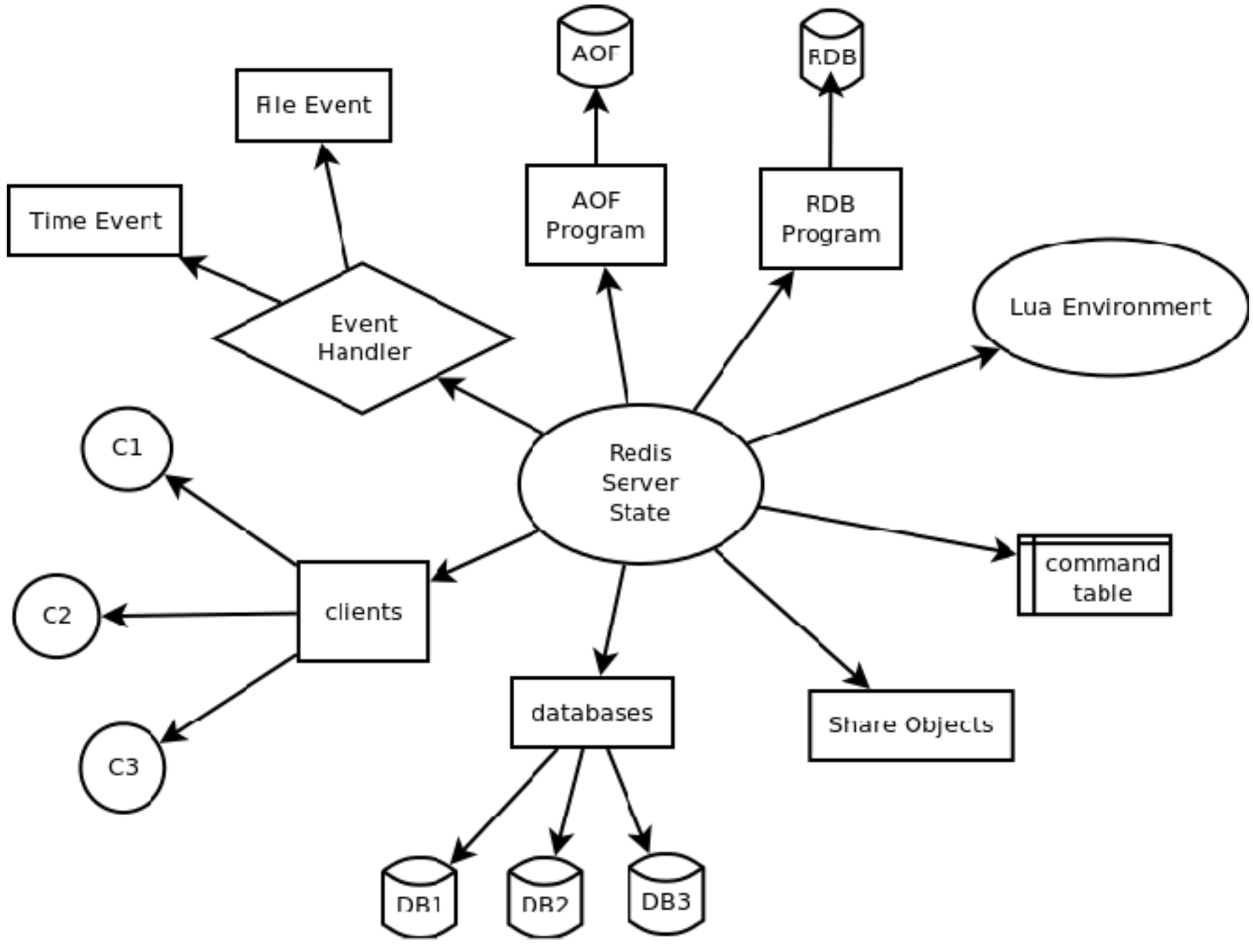
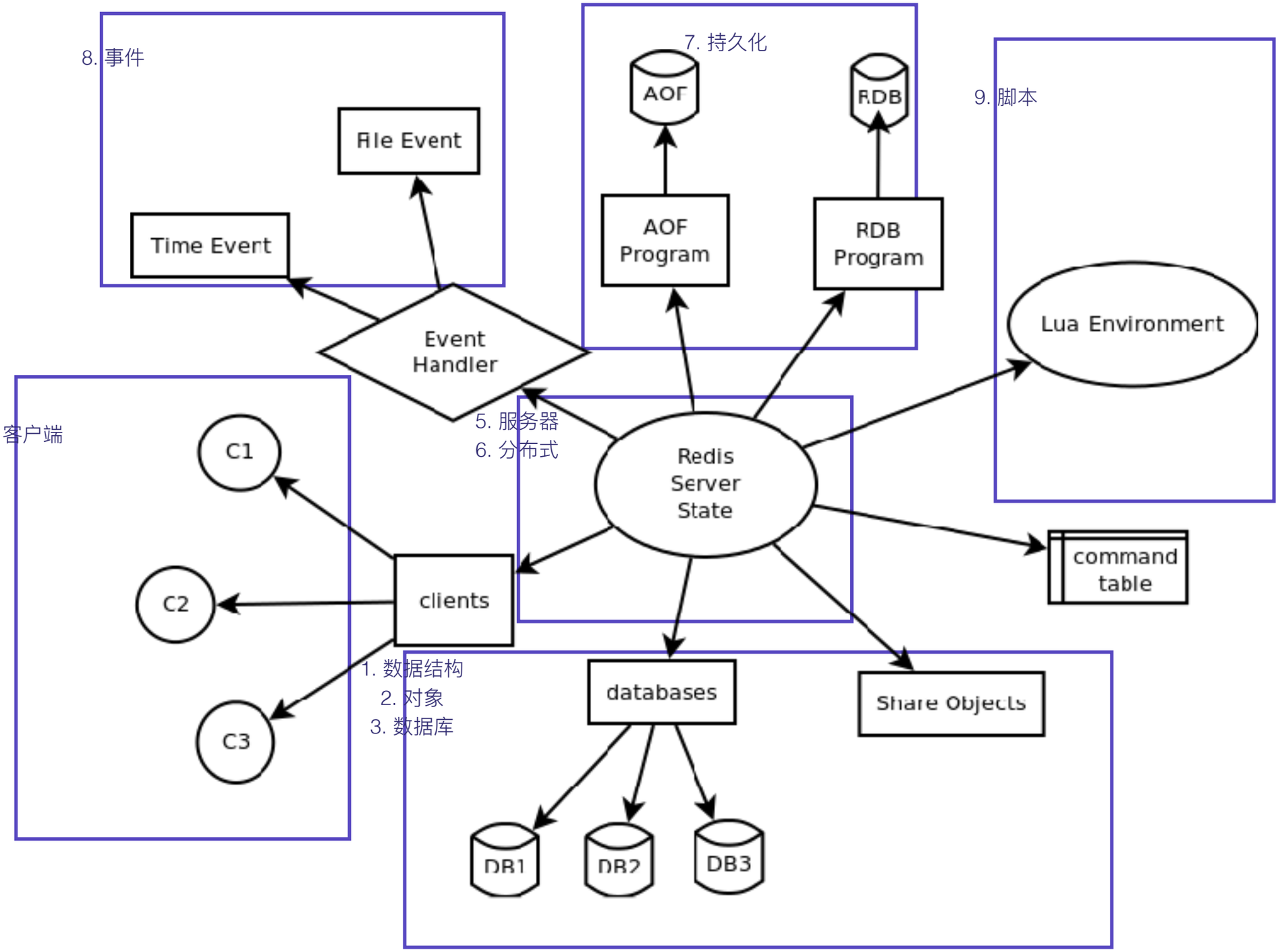
Redis设计与实现

sunkai@51haoyayi.com

REmote **D**ictionary **S**erver



REmote **D**ictionary **S**erver



基本数据结构

- SDS
- Linkedlist
- Skiplist
- Dict
- Intset
- Ziplist

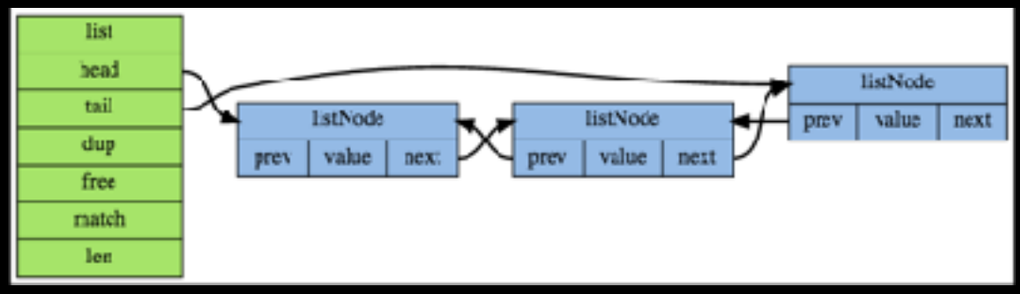
基本数据结构

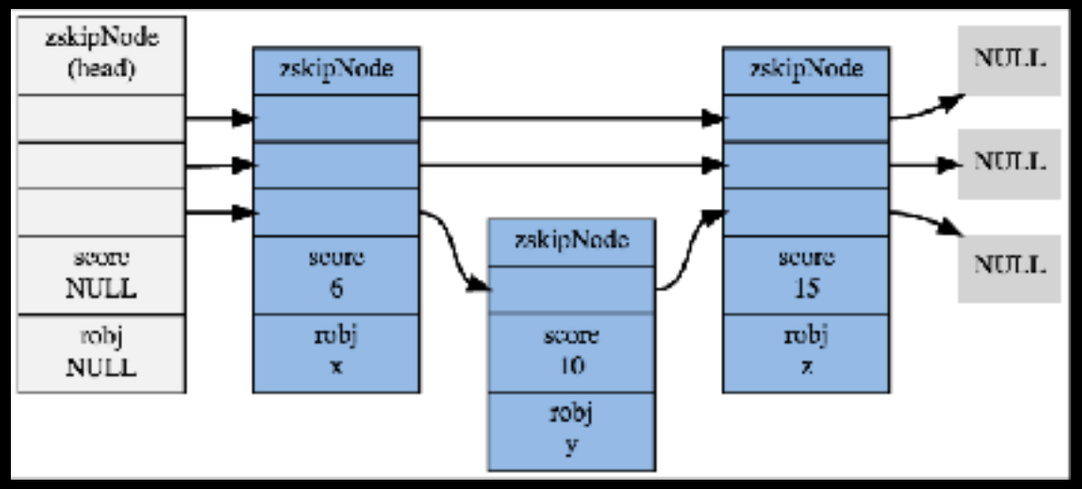
- SDS **Simple Dynamic String**
  - 实现redis的StringObject
  - 内部代替char *
  - strlen
  - append　　SDS_MAX_PREALLOC 1M
  - 二进制安全

```
typedef char *sds;

struct sdshdr {

    // buf 已占用长度
    int len;

    // buf 剩余可用长度
    int free;

    // 实际保存字符串数据的地方
    char buf[];
};
```

基本数据结构

- **Linkedlist**
  - 双向链表
  - void*
- **Skiplist**
  - 跳表
  - score
  - levels
    - forward pt + span
    - power law.
  - backward pt





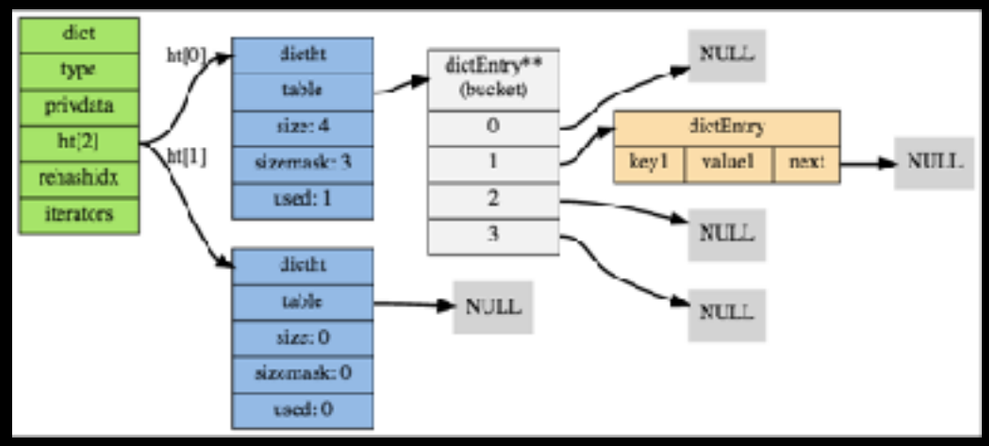
基本数据结构

- **Dict**
  - 使用hashtable作为底层实现
  - 两个hashtable

```
/*
 * 字典
 *
 * 每个字典使用两个哈希表，用于实现渐进式 rehash
 */
typedef struct dict {

    // 特定于类型的处理函数
    dictType *type;

    // 类型处理函数的私有数据
    void *privdata;

    // 哈希表（2 个）
    dictht ht[2];

    // 记录 rehash 进度的标志，值为 -1 表示 rehash 未进行
    int rehashidx;

    // 当前正在运作的安全迭代器数量
    int iterators;

} dict;
```

基本数据结构

- **Dict**
  - Hashtable
    - Hash算法：MurmurHash2
    - Hash Collision：Seperate Chain
  - Rehash
    - ratio=used/size
      - 1 & dict_can_resize
      - dict_force_resize_ratio=5
    - rehashidx
    - incremental rehash
  - expand & shrink



基本数据结构

- **Intset**
  - 无重复、有序整数
  - 元素不多
  - Encoding
    - INTSET_ENC_INT16
    - INTSET_ENC_INT32
    - INTSET_ENC_INT64
    - Upgrade
    - 不支持降级

```
typedef struct intset {

    // 保存元素所使用的类型的长度
    uint32_t encoding;

    // 元素个数
    uint32_t length;

    // 保存元素的数组
    int8_t contents[];

} intset;
```

基本数据结构

- **Ziplist**
  - 连续内存空间
  - <zlbytes><zltail><zllen><entry>...<entry><zlend>
  - Entry
    - TLV (Type-Length-Value)
    - <prelen><<encoding+lensize><len>><data>

对象

- **RedisObject**
  - type
  - encoding
  - ptr
  - refcount
  - lru

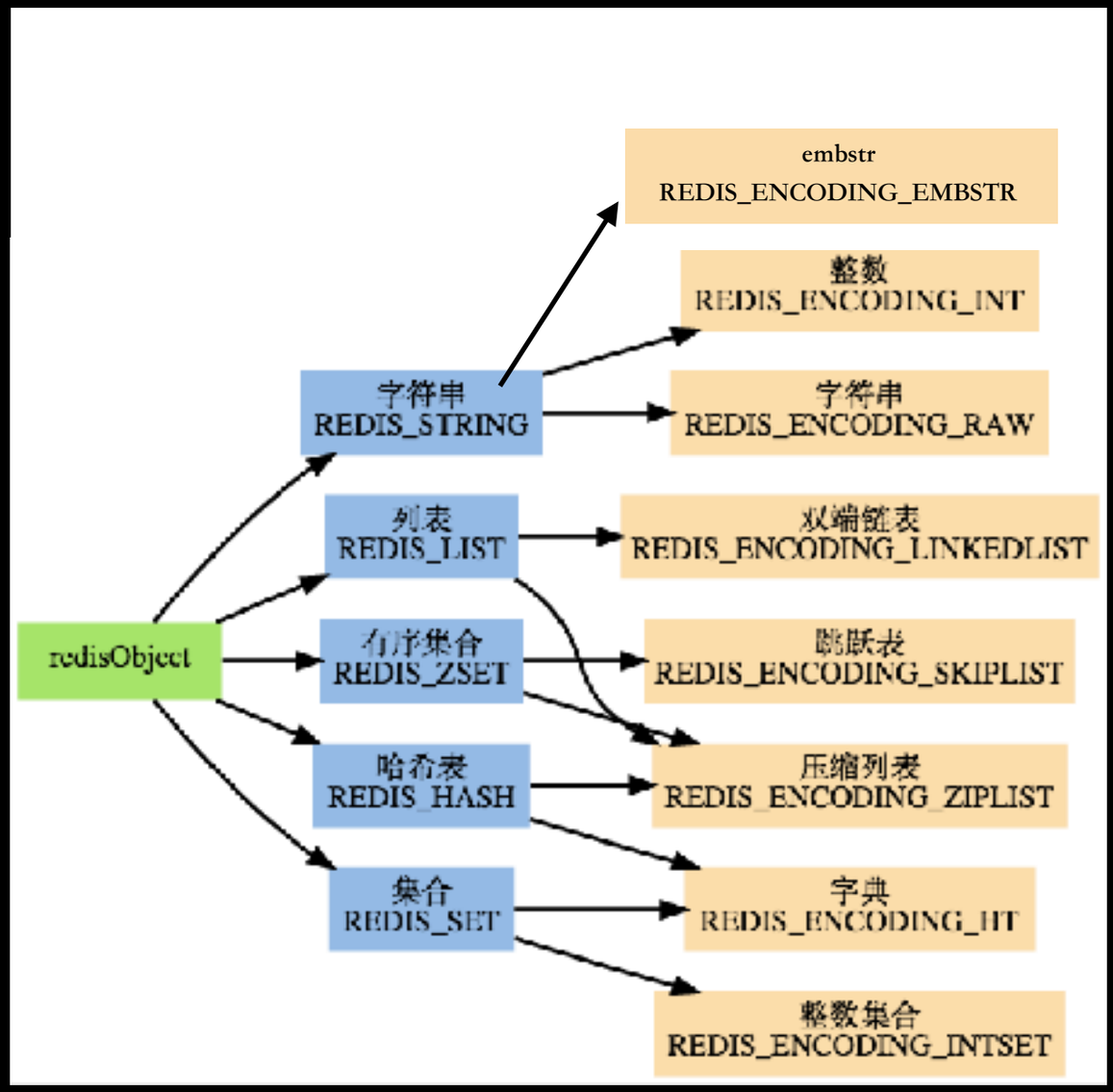
```
/*
 * Redis 对象
 */
typedef struct redisObject {

    // 类型
    unsigned type:4;

    // 对齐位
    unsigned notused:2;

    // 编码方式
    unsigned encoding:4;

    // LRU 时间（相对于 server.lruclock）
    unsigned lru:22;

    // 引用计数
    int refcount;

    // 指向对象的值
    void *ptr;

} robj;
```

对象

- **RedisObject**
  - type
  - encoding
  - ptr

Type check

Polymorphism

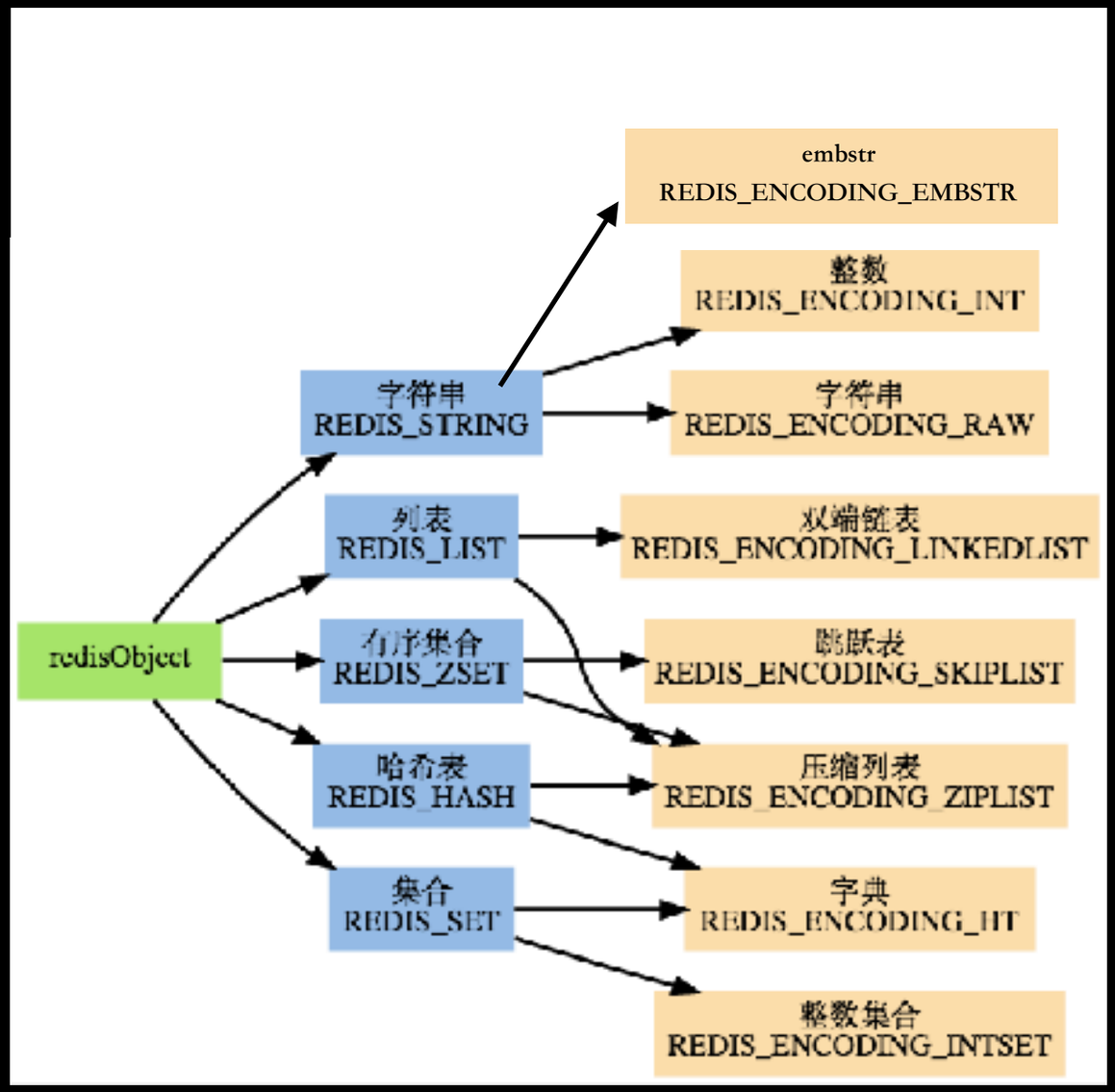


对象

- **RedisObject**
  - type
  - encoding
  - ptr

内存共享

预分配
  - 命令返回
  - REDIS_SHARED_INTEGERS



对象

- **RedisObject**
  - refcount
  - GC

```
/*
 * Redis 对象
 */
typedef struct redisObject {

    // 类型
    unsigned type:4;

    // 对齐位
    unsigned notused:2;

    // 编码方式
    unsigned encoding:4;

    // LRU 时间（相对于 server.lruclock）
    unsigned lru:22;

    // 引用计数
    int refcount;

    // 指向对象的值
    void *ptr;

} robj;
```

对象

- **RedisObject**
  - lru
  - MAXMEMORY

```
/*
 * Redis 对象
 */
typedef struct redisObject {

    // 类型
    unsigned type:4;

    // 对齐位
    unsigned notused:2;

    // 编码方式
    unsigned encoding:4;

    // LRU 时间（相对于 server.lruclock）
    unsigned lru:22;

    // 引用计数
    int refcount;

    // 指向对象的值
    void *ptr;

} robj;
```

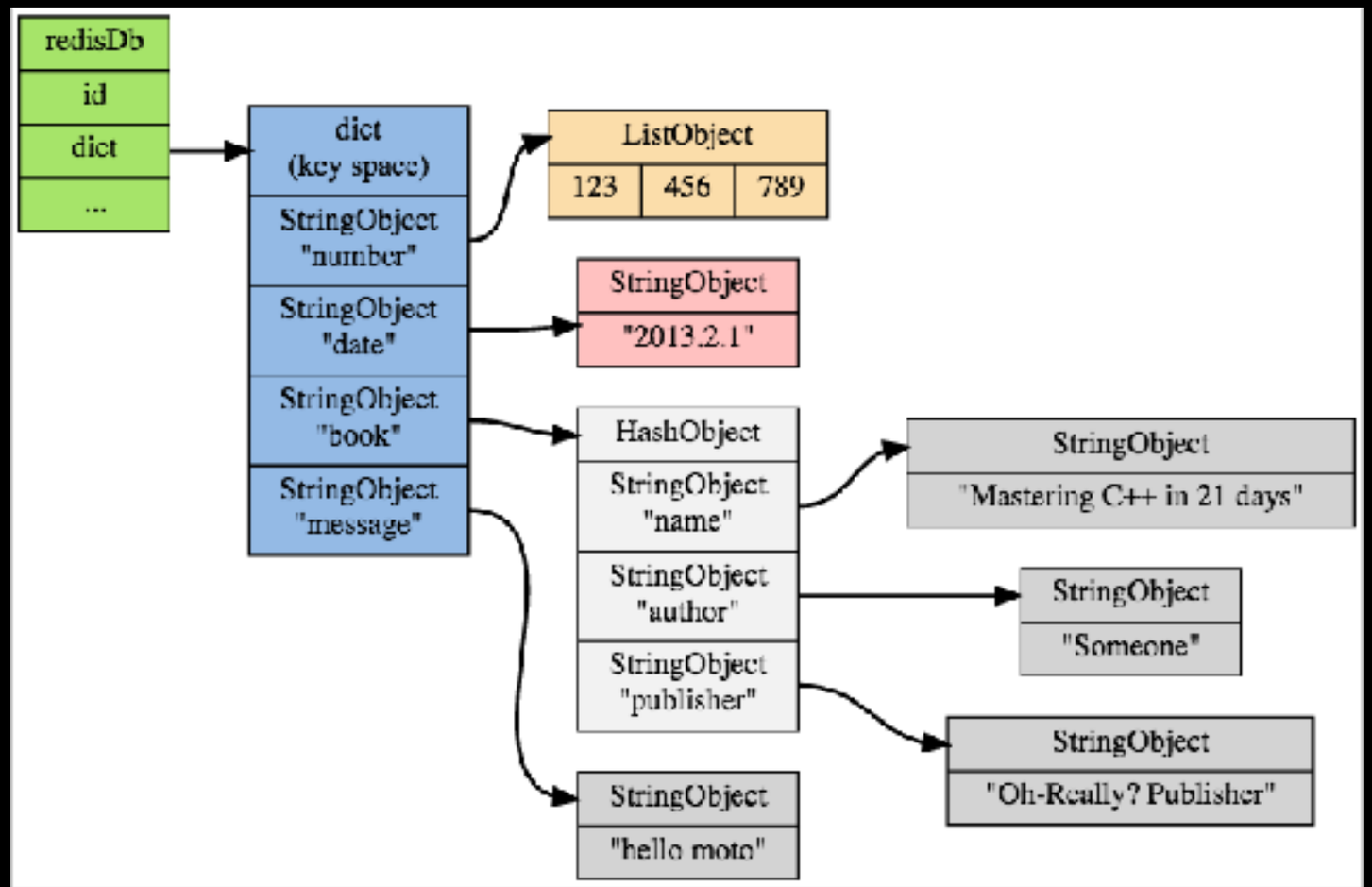
数据库

- Database
  - select
  - key space
    - key -> string object
    - value -> reds object

```
typedef struct redisDb {

    // 保存着数据库以整数表示的号码
    int id;

    // 保存着数据库中的所有键值对数据
    // 这个属性也被称为键空间（key space）
    dict *dict;

    // 保存着键的过期信息
    dict *expires;

    // 实现列表阻塞原语，如 BLPOP
    // 在列表类型一章有详细的讨论
    dict *blocking_keys;
    dict *ready_keys;

    // 用于实现 WATCH 命令
    // 在事务章节有详细的讨论
    dict *watched_keys;

} redisDb;
```

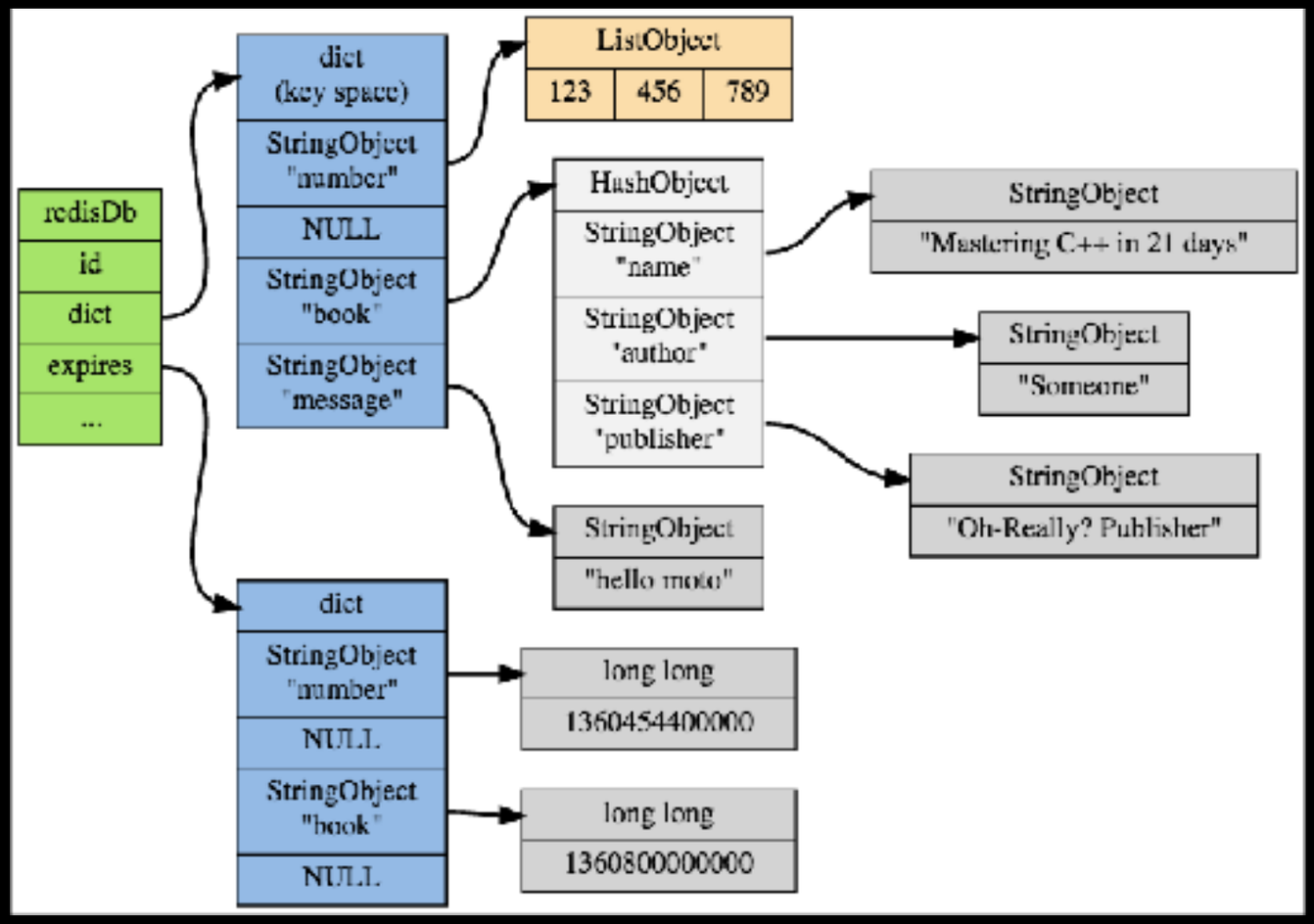
数据库

- Database



数据库

- Database
  - EXPIRE
  - PEXPIRE
  - PEXPIREAT
  - TTL
  - PTTL



数据库

- Database
  - 惰性删除
    - expireIfNeeded
  - 定期删除
    - activeExpireCycle
    - reach_time_limit()

数据库

- 事务
  - MULTI　client.flags　REDIS_MULTI
  - Command Queue
  - EXEC
  - DISCARD
  - WATCH optimistic locking
    - redisDb.watched_keys
    - touchWatchKey　-> REDIS_DIRTY_CAS

数据库

- 事务
  - A 不支持事务回滚
  - C
  - I 单线程
  - D AOF appendfsync=always
    - MULTI -> SAVE -> EXEC 效率低

客户端/服务端

- **RedisClient**
  - fd
  - name
  - flags 二进制
    - REDIS_MASTER,REDIS_SLAVE,REDIS_LUA_CLIENT……
  - 输入
    - querybuf <1G
    - argv argc
    - redisCommand
  - 输出
    - buf bufpos 定长输出buf
    - reply 变长输出buf
  - aunthenticated 0 NO PASS 1 PASS

```
struct redisServer {
    list *clients;
}


typedef struct redisClient {
    int fd;
    robj *name;
    int flags;
    sds querybuf;
    robj **argv;
    int argc;
    struct redisCommand *cmd;
    char buf[REDIS_REPLAY_CHUNK_BYTES]; // 16kb
    int bufpos;
    list *reply
    int authenticated;
}
```

客户端/服务端

- 服务器初始化
  - 初始化服务器全局状态。
  - 载入配置文件。
  - 创建 daemon 进程。
  - 初始化服务器功能模块。
  - 载入数据。
  - 开始事件循环

多机分布式

- Replication
- Sentinel
- Cluster

多机分布式

- Replication
  - SLAVEOF <IP> <PORT>
  - SYNC
  - PSYNC

多机分布式

- Replication
- SLAVE -> SYNC ->MASTER
- MASTER
  - BGSAVE -> RDB
  - Commands -> Buffer
  - SEND RDB
  - SEND Commands

多机分布式

- Replication
- 一致性问题
  - Command Propagate
  - 最终一致性
- PSYNC　Partial resync

多机分布式

- **Sentinel**
  - Monitoring
  - Notification
  - Automatic failover
  - Configuration Provider

多机分布式

- **Cluster**
  - Node　Master/Slave
  - Sharding slot　2^14(16384)
    - CRC16(key) & 16383
    - slot table
- SLAVEOF
- Failover 类似sentinel

持久化

- **RDB**
  - rdbSave
    - SAVE 主进程block
    - BGSAVE fork子进程，主进程不block
  - rdbLoad
    - 读取rdb文件
    - 除PUB/SUB之外所有命令，其他命令需要等load完成。
    - 优先AOF

持久化

- RDB
  - Binary

  - REDIS Magic Number
  - VERSION 006
  - 多个数据库+kv数据
  - 8B校验和

持久化

- **AOF**
  - 文本协议
  - 写入命令+参数
  - 过程
    - 命令传播
    - 缓存追加
      - 生成协议文本　追加aof_buf

```
if (execRedisCommand(cmd, argv, argc) ==
EXEC_SUCCESS):

    if aof_is_turn_on():
        # 传播命令到 AOF 程序
        propagate_aof(cmd, argv, argc)

    if replication_is_turn_on():
        # 传播命令到 REPLICATION 程序
        propagate_replication(cmd, argv, argc)
```

持久化

- **AOF**
    - 文本协议
    - 写入命令+参数
    - 过程
        - 文件写入和保存
            - flushAppendOnlyFile
                - WRITE aof_buf写aof文件
                - SAVE fsync/fdatasync aof文件写磁盘
            - 模式:
                - AOF_FSYNC_NO： 不保存。WRITE NO SAVE
                - AOF_FSYNC_EVERYSEC： 每一秒钟保存一次。
                - AOF_FSYNC_ALWAYS： 每执行一个命令保存一次。

持久化

- AOF
- 还原
    - 创建一个不带网络连接的伪客户端（fake client）。
    - 读取 AOF 所保存的文本，并根据内容还原出命令、命令的参数以及命令的个数。
    - 根据命令、命令的参数和命令的个数，使用伪客户端执行该命令。
    - 执行 2 和 3 ， 直到 AOF 文件中的所有命令执行完毕。
- Rewrite
    - BGREWRITEAOF

事件

- File event
- Time event

事件

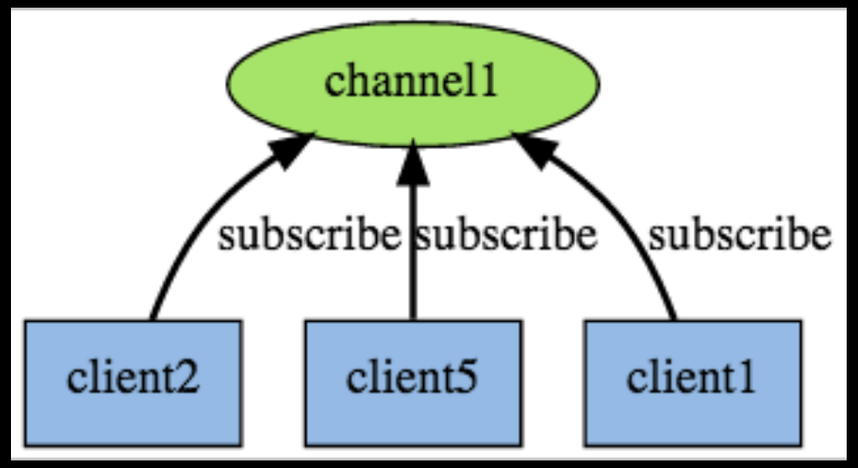
- File event
  - socket操作抽象
  - Reactor
    - multiplexing
      - select epoll evport kqueue
    - Event
      - AE_READABLE
    - AE_WRITEABLE

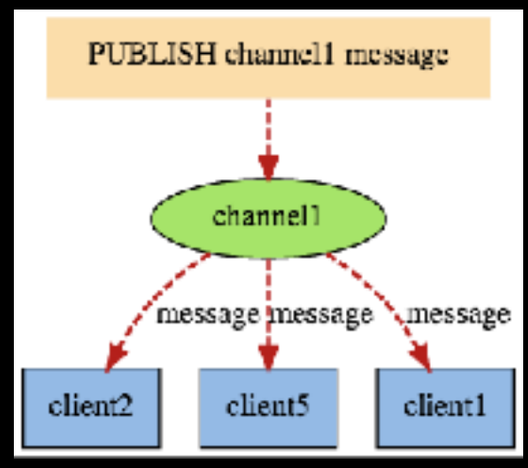
事件

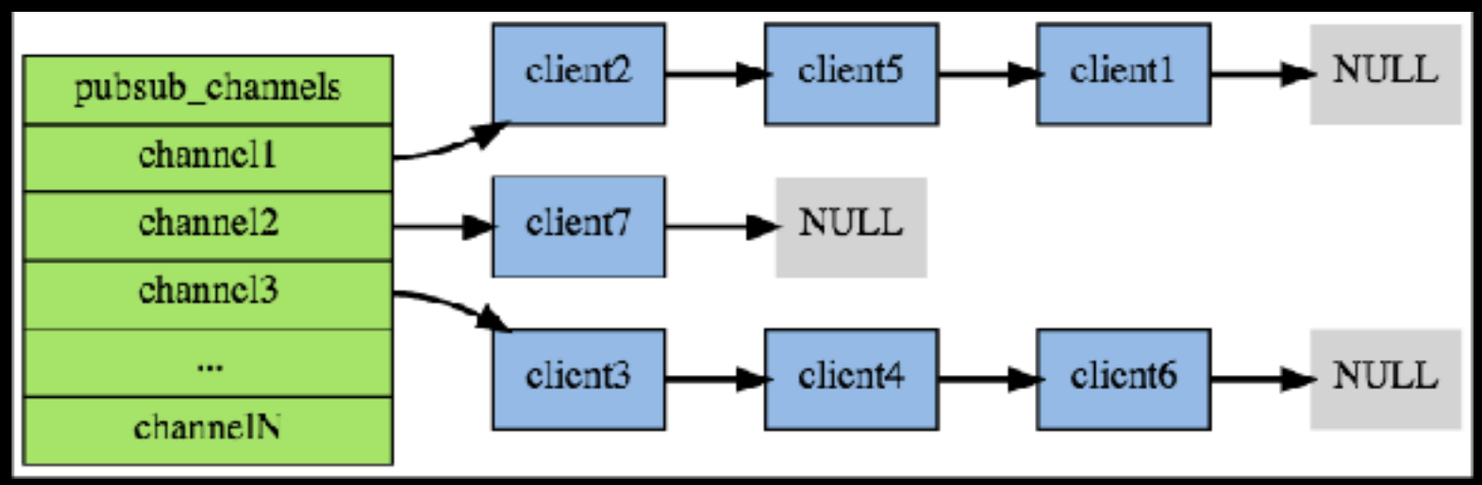
- Time event
  - 定时事件
  - 周期性事件
    - 结构
      - id
      - when
      - timeProc
- serverCron

事件

- Pub/Sub
  - PUBLISH
  - SUBSCRIBE
  - PSUBSCRIBE

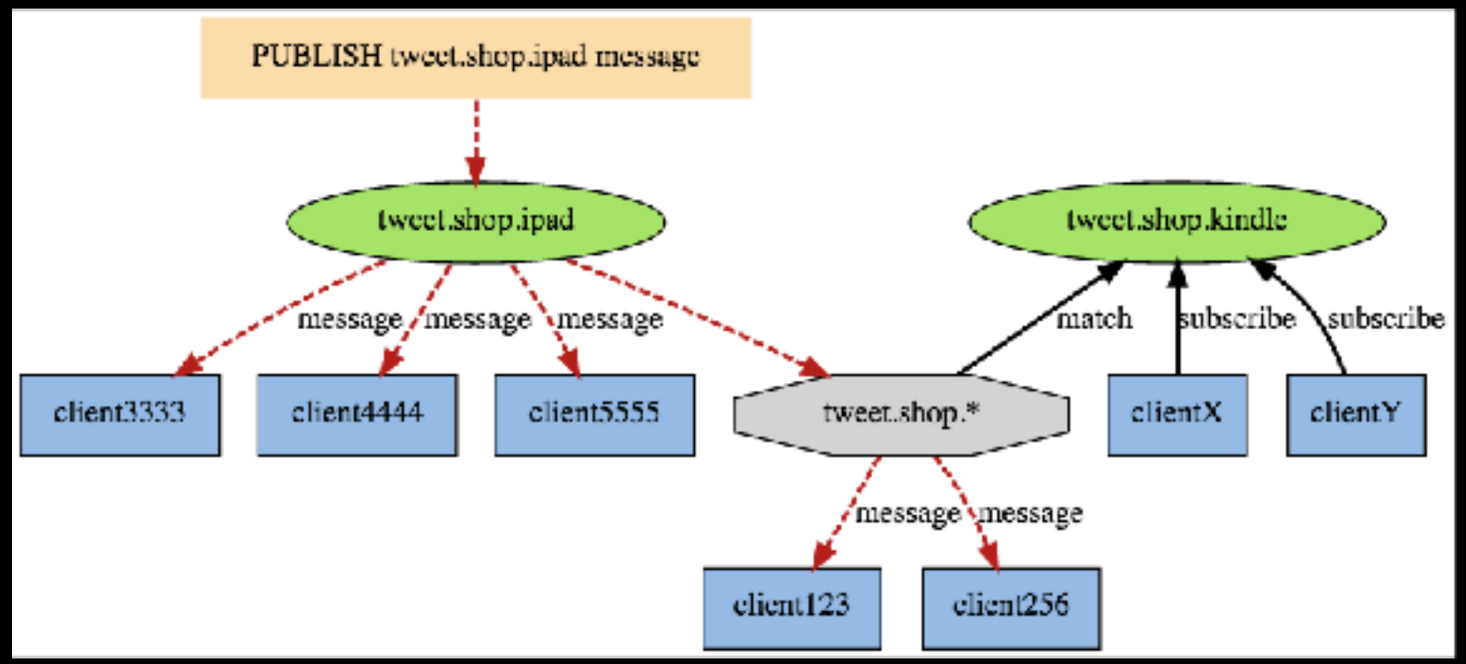






事件

- Pub/Sub
  - PSUBSCRIBE



脚本

- Redis 2.6+
  - 支持Lua
    - redis> EVAL “return ‘haoyayi’” 0