Redis学习

1、redisObject对象机制

Q：为什么要构建redis对象机制?

redis对键进行处理的命令占了很大一部分，对于键所保存的值的类型，键能执行的命令又各不相同。

有些命令可以用于任何类型的键：del,ttl,type

要正确实现折现命令，就必须为不同类型的键设置不同的处理方法

此外redis的相同type的底层数据结构也有可能不同，例如string:int,embstr,str

那么对string的命令还需根据底层数据结构的不同编码进行多态处理

说到底 redis通过对象机制，配置了key的一些信息，用于

- 类型检查
- 显式多态函数
- 分配、共享和销毁

2、redisObject包含哪些属性?

- type
- encoding
- lru
- refcount
- *ptr

其中 type、encoding和*ptr描述了一个真正的key

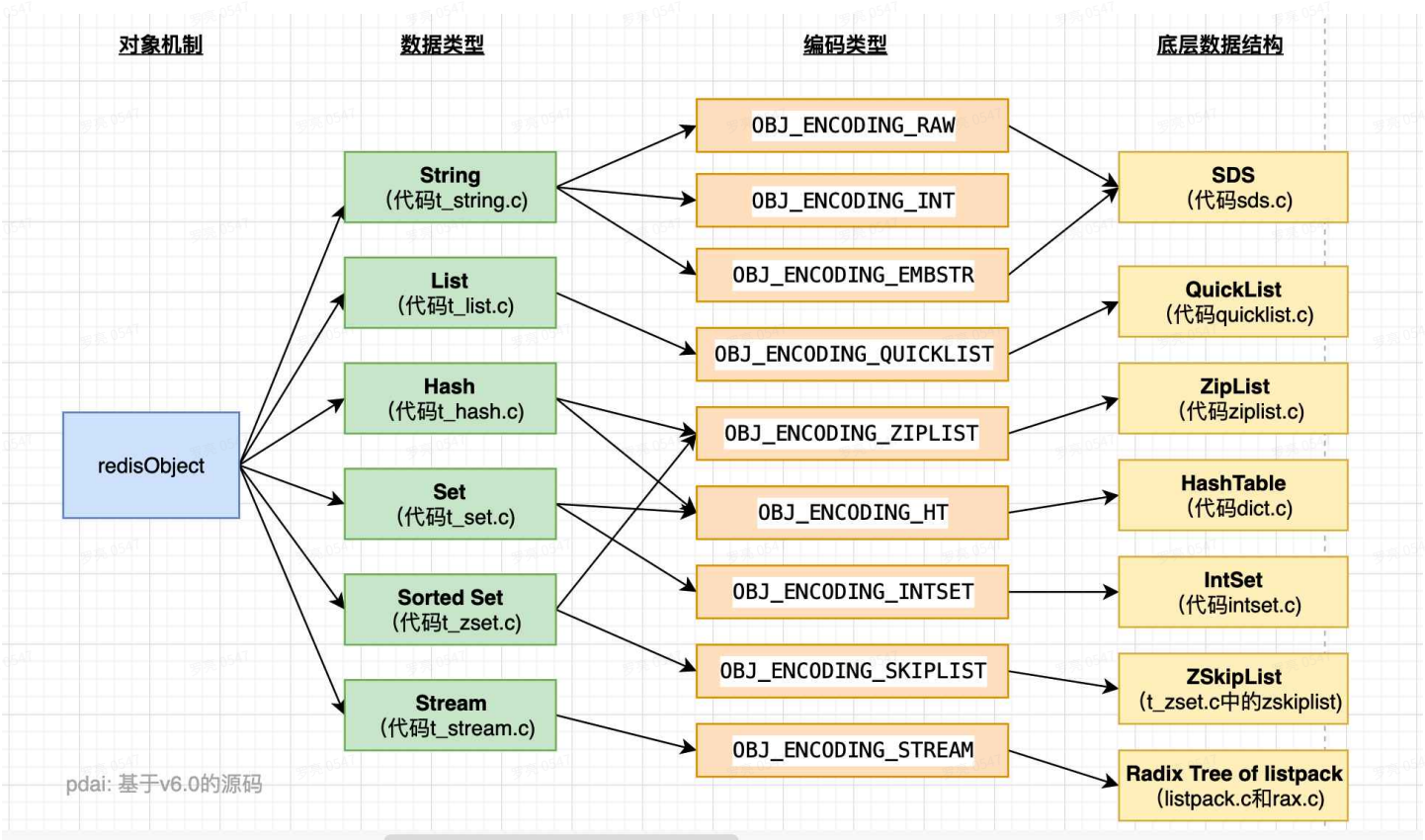
Type:

  - string
  - list
  - set
  - hash

- zset

encoding：

橙色列

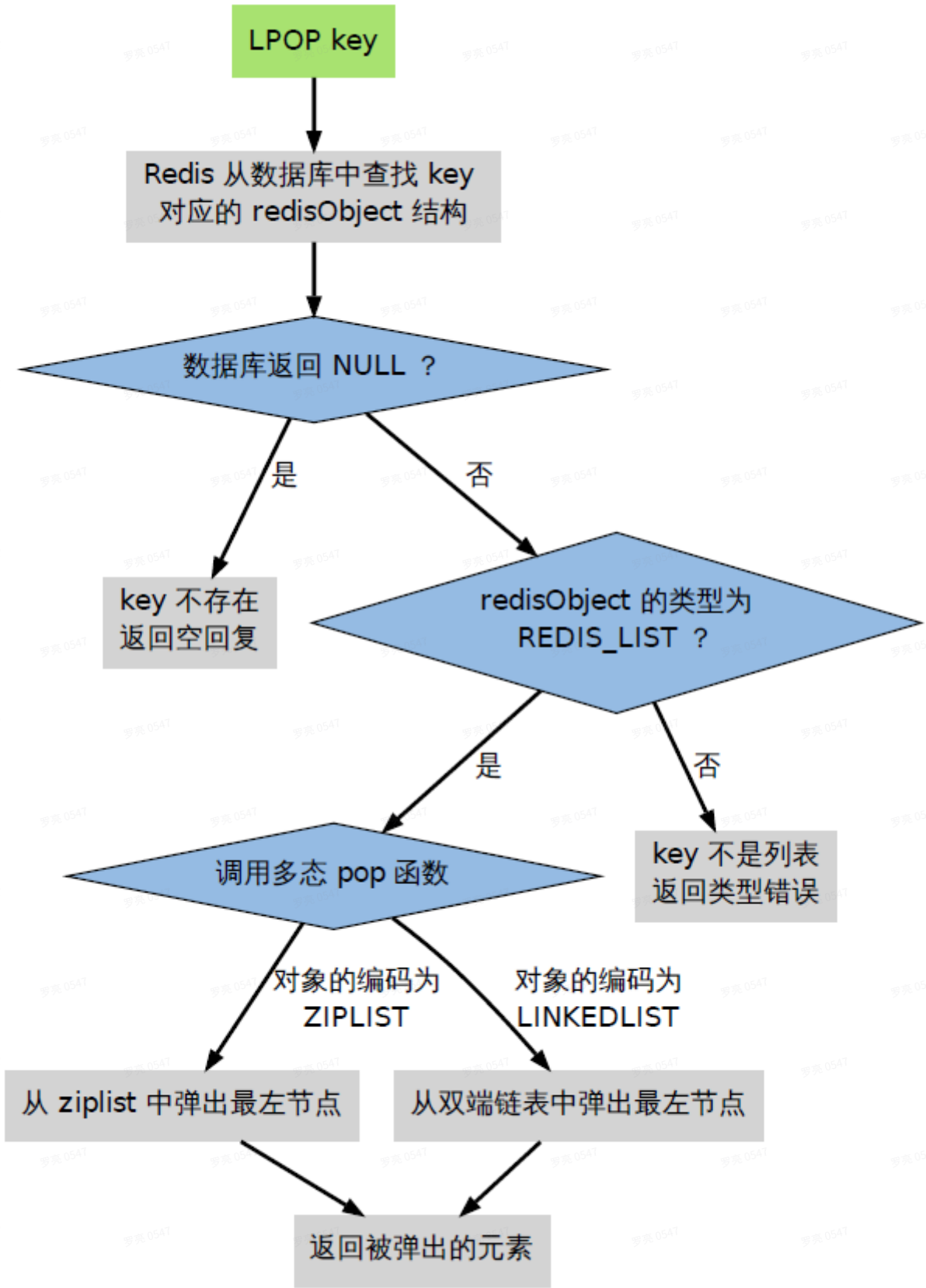


*ptr 指针，指向实际保存值的底层数据结构

lru:记录了对象最后一次被命令程序访问的时间,用于缓存淘汰和过期计时

refcount:饮用计数，为0则释放

- 类型检查和多态函数调用



- 对象共享
  - 命令的返回值：OK,ERROR等
  - 包括0 在内，小于REDIS_SHARED_INTEGERS的所有整数（REDIS_SHARED_INTEGERS的默认值是10000）

3、基本数据类型、编码类型和底层数据结构

string	int	sds
	embstr	
	raw	

如果一个字符串可转化成long,就转化成long类型，encoding设为**int**，64字节即可，不需要额外开辟空间存长度等信息，算是一个存储优化

如果一个字符串小于44字节，那么encoding为**embstr**，因为内存行为64字节，小于44的话可以一次性分配redisObject和底层数据结构的内存空间，减少io次数

- 快表
- 跳表

4、持久化

RDB和AOF可以同类比mysql的bin log和redo log（功能上）

4.1、RDB

- 优势
  - 数据恢复比aof快
  - 备份和主从复制
- 劣势
  - 实时性不够
  - fock子进程刷盘
- rdb能做到秒级快照吗?

可以，但会给磁盘和cpu造成巨大的压力

- 写快照过程中，有新的写入该怎么办?

RDB中的核心思路是Copy-on-Write，来保证在进行快照操作的这段时间，需要压缩写入磁盘上的数据在内存中不会发生变化。在正常的快照操作中，一方面Redis主进程会fork一个新的快照进程专门来做这个事情，这样保证了Redis服务不会停止对客户端包括写请求在内的任何响应。另一方面这段时间发生的数据变化会以副本的方式存放在另一个新的内存区域，待快照操作结束后才会同步到原来的内存区域。

4.2、AOF

Append only file

配置项	写回时机	优点	缺点
Always	同步写回	可靠性高，数据基本不丢失	每个写命令都要落盘，性能影响较大
Everysec	每秒写回	性能适中	宕机时丢失1秒内的数据
No	操作系统控制的写回	性能好	宕机时丢失数据较多

- aof重写机制

  AOF会记录每个写命令到AOF文件，随着时间越来越长，AOF文件会变得越来越大。如果不加以控制，会对Redis服务器，甚至对操作系统造成影响，而且AOF文件越大，数据恢复也越慢。为了解决AOF文件体积膨胀的问题，Redis提供AOF文件重写机制来对AOF文件进行“瘦身”。

- 优势
  - 可以做到不丢失数据，或者最多丢失一秒内的数据
- 劣势
  - fork子进程重写
  - 恢复数据的过程需要一步一步重新执行命令，效率较低
- 写日志的过程中，有数据写入咋办?

  重写过程总结为：“一个拷贝，两处日志”。在fork出子进程时的拷贝，以及在重写时，如果有新数据写入，主线程就会将命令记录到两个aof日志内存缓冲区中。如果AOF写回策略配置的是always，则直接将命令写回旧的日志文件，并且保存一份命令至AOF重写缓冲区，这些操作对新的日志文件是不存在影响的。

  而在bgrewriteaof子进程完成会日志文件的重写操作后，会提示主线程已经完成重写操作，主线程会将AOF重写缓冲中的命令追加到新的日志文件后面。

5、redis的事务

- multi
- exec
- descard
- watch
- unwatch

- watch如何监控?

watch监控的key,表示在下一个exec执行之前，key的value都不允许发生变化，如果执行exec之前发生变化了，则exec事务包裹的命令全部执行失败。

6、主从复制

- 全量复制和增量复制

增量复制是为了避免全量复制过程中，出现断连后，再次触发全量复制，开销非常大，如果合理断连时间内重连，可以采用增量复制继续。

- 为什么全量复制采用rbd文件为不是aof文件?
  - rdb文件小，aof文件大，冗余多，且执行命令慢，总之aof全量复制效率低
  - 在缓存场景（丢失数据不敏感），一般情况下是不开启aof的，那为了全量复制，在很多不需要开启aof的场景也开启了，会造成更多负担
- 主-从-从

主redis实例一对多主从复制，会给主节点造成巨大的压力，通过一个从节点中转，可以缓解主节点压力

- 读写分离
  - 延迟与数据不一致问题

7、哨兵和分片

8、缓存问题

- 缓存穿透

  缓存没有、数据库也没有

  有人用不存在的key恶意攻击

  解决办法：布隆过滤器

  - 布隆过滤器存在的问题
- 缓存击穿

  缓存没有、db有

  一个key突然过期，此时恰好有多个请求访问key，以至于所有请求都打到db上去了，数据库压力瞬增

  解决办法：热点数据永不过期，限流、熔断和过载控制

- 缓存雪崩

  缓存击穿的多key版本

  db压力更大了
- 缓存污染

  缓存淘汰策略：8种
- db和缓存数据一致性

一般情况：最终一致性就行

  - 延迟双删
  - 写后删，队列重试机制保证删操作成功

  写后删+队列重试的缺点就是对业务代码侵入性太强

  - 订阅binlog删除缓存

如果非得强一致性：应该挺难的，可以看看分布式事务的策略

9、redis快的原因

- 内存
- 数据结构
- io多路复用(网络io多线程)
- 主线程是单线程，避免上下文切换

https://segmentfault.com/a/1190000041488709

深入理解redis——Redis快的原因和IO多路复用深度解析

当我们在学习和使用redis的时候，大家口耳相传的，说的都是redis是单线程的。其实这种说法并不严谨，Redis的版本有非常多，3.x、4.x、6.x，版本不同架构也是...