Lesson1 Redis入门

数据结构与内存管理

Bo.Xiao

Agenda

- Reids简介
- Redis架构
- Redis数据结构
- Redis内存管理

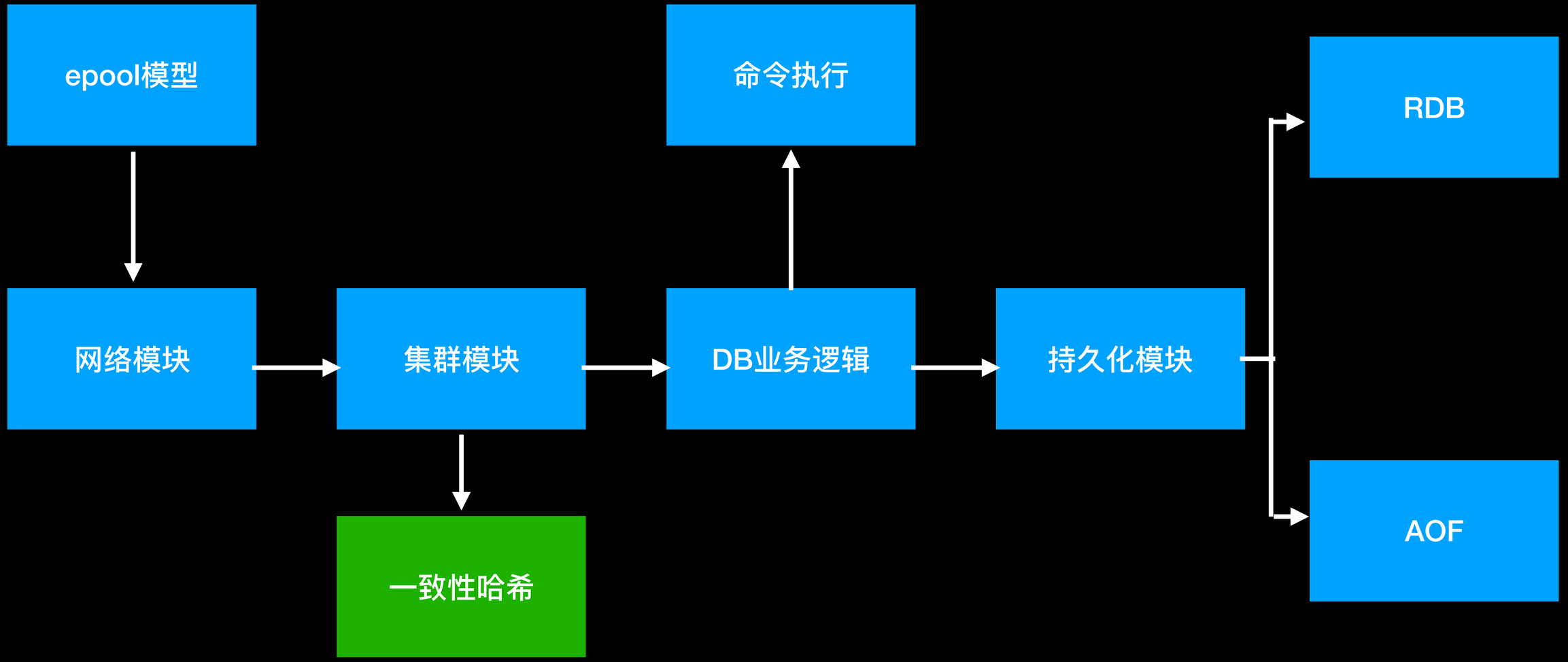
Redis架构

Redis简介

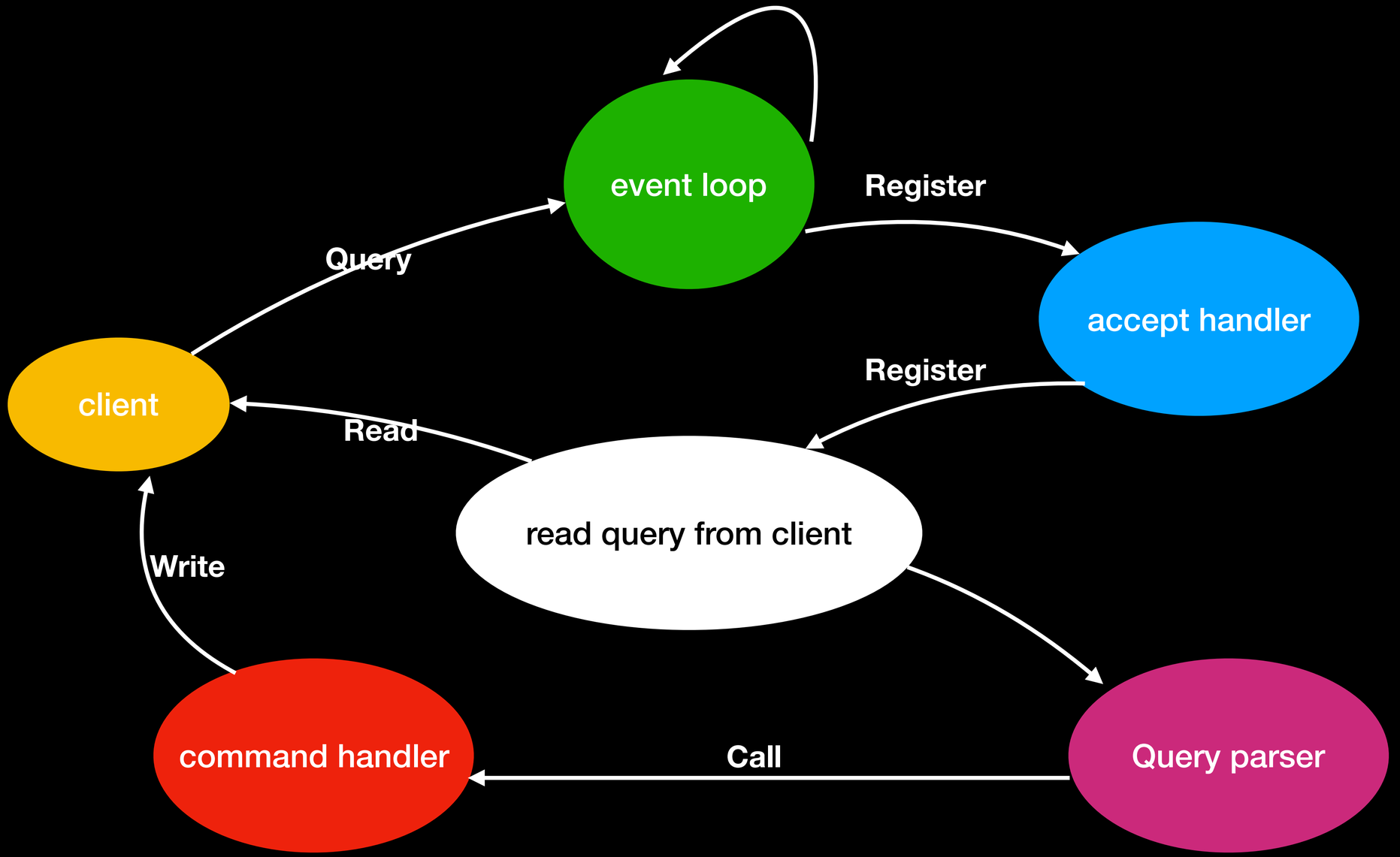
- In Memory
- Data structure store
- database,cache ,message broker

redis

Redis架构



Redis架构


event loop
Register
accept handler
Query
client
Register
Read
read query from client
Write
command handler
Call
Query parser

Redis数据结构

Redis数据结构

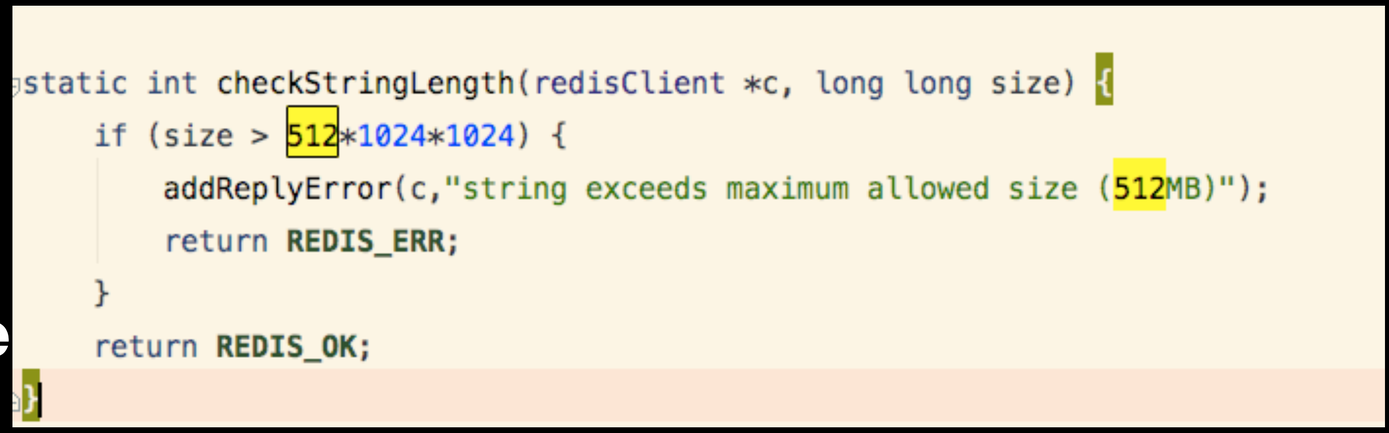
- String
- Hash
- List
- Set
- ZSet

Redis keys

- Strings & binary safe
- Very long keys are not a good idea
- Very short keys are often not a good idea
- Try to stick with a schem
- The maximum allowed key size is 512 MB

String

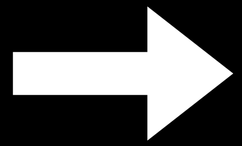
- Strings & binary safe
- The maximum allowed key size is 512 MB



```
static int checkStringLength(redisClient *c, long long size) {
    if (size > 512*1024*1024) {
        addReplyError(c,"string exceeds maximum allowed size (512MB)");
        return REDIS_ERR;
    }
    return REDIS_OK;
}
```

Hash

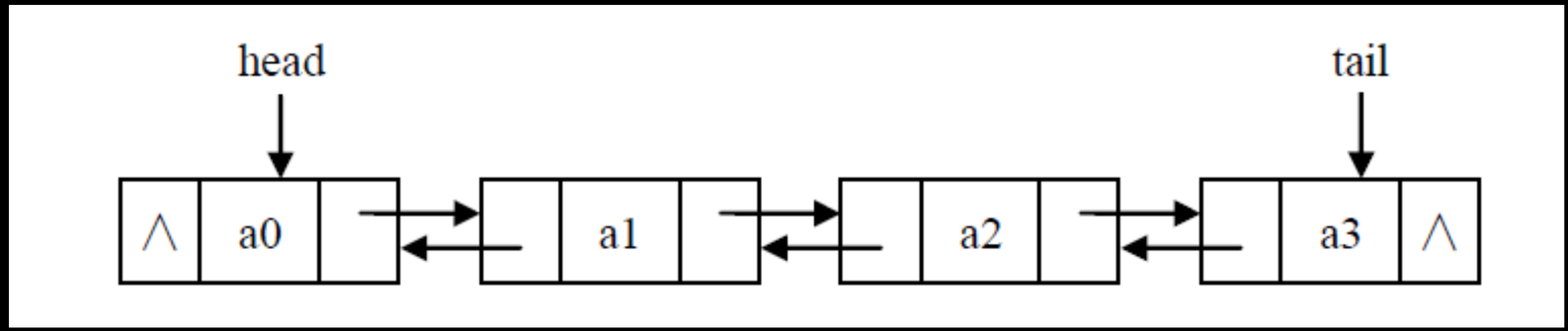
- ZipMap
- o(n)



- **HashMap**
- **O(1)**

List

- 反向查找
- 遍历



Set & Zset

- 有序集合
- 无序集合

内存管理

SDS

- 获取字符串长度更容易
- 减少修改字符串时带来的内存重分配次数
- 二进制安全
- 兼容部分C字符串函数

```c
struct sdshdr {

    // 记录 buf 数组中已使用字节的数量
    // 等于 SDS 所保存字符串的长度
    int len;

    // 记录 buf 数组中未使用字节的数量
    int free;

    // 字节数组，用于保存字符串
    char buf[];

};
```

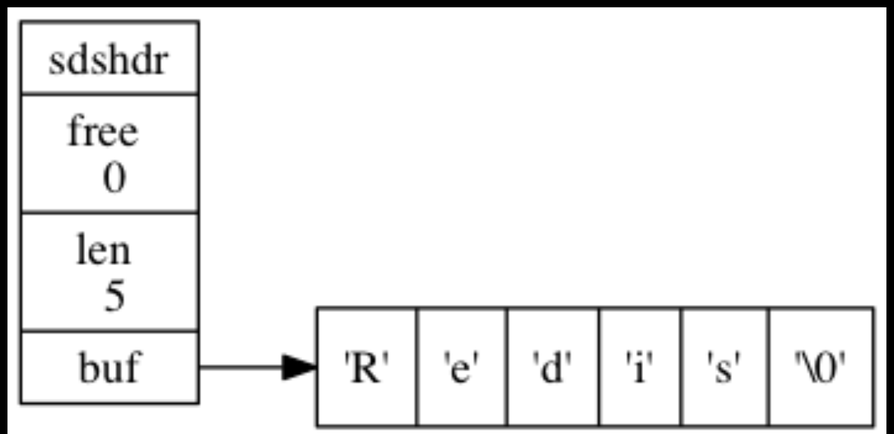


图 2-1　SDS 示例

Maxmemory

To store user keys, Redis allocates at most as much memory as the maxmemory setting enables (however there are small extra allocations possible).

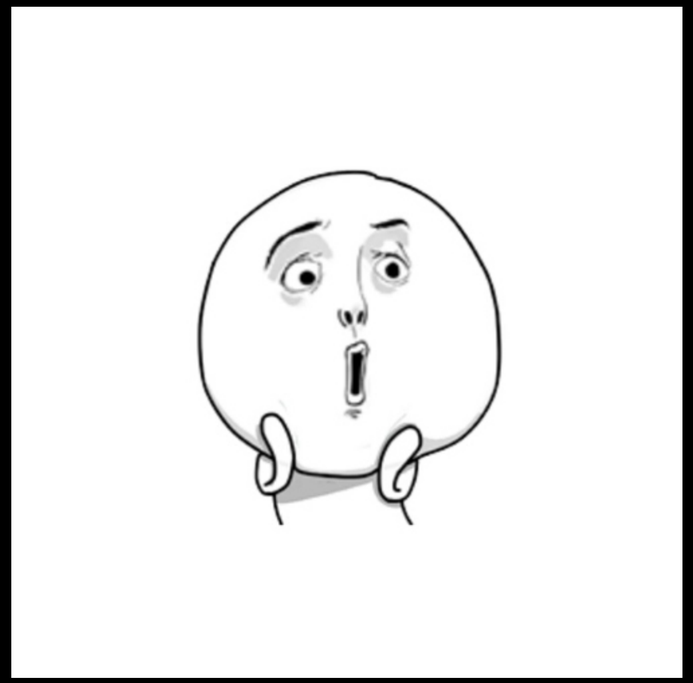
```
int freeMemoryIfNeeded(void) {
    size_t mem_used, mem_tofree, mem_freed;
    int slaves = listLength(server.slaves);
    mstime_t latency, eviction_latency;

    /* Remove the size of slaves output buffers and AOF buffer from the
     * count of used memory. */
    mem_used = zmalloc_used_memory();
    if (slaves) {
        listIter li;
        listNode *ln;

        listRewind(server.slaves,&li);
        while((ln = listNext(&li))) {
            client *slave = listNodeValue(ln);
            unsigned long obuf_bytes = getClientOutputBufferMemoryUsage(slave);
            if (obuf_bytes > mem_used)
                mem_used = 0;
            else
                mem_used -= obuf_bytes;
        }
    }
    if (server.aof_state != AOF_OFF) {
        mem_used -= sdslen(server.aof_buf);
        mem_used -= aofRewriteBufferSize();
    }
```

Maxmemory

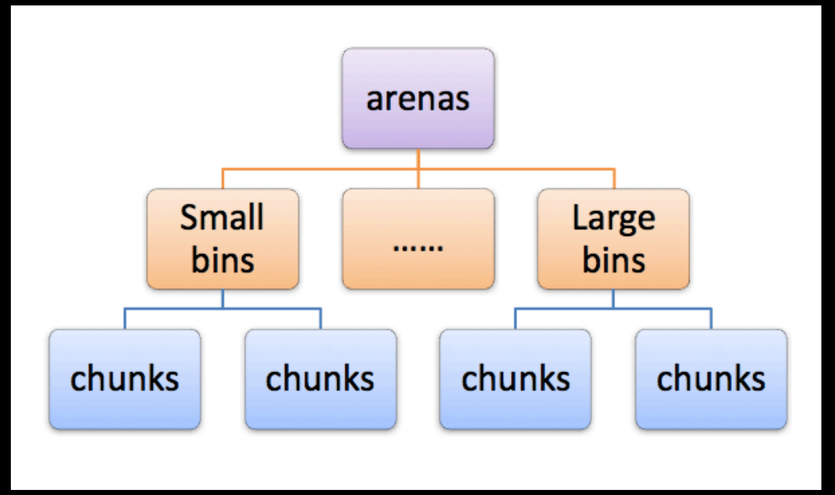
used_memory > maxmemory ?



Maxmemory

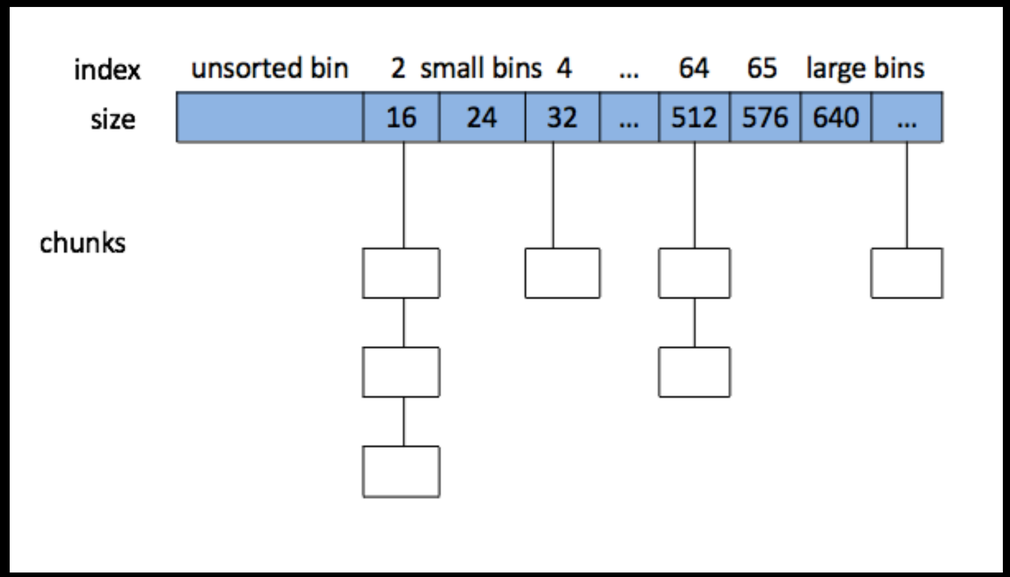
mem_used - obuf_bytes - aofRewriteBufferSize() - repl-backlog-size - RDB-COW - LUA - monitors = maxmemory

内存碎片率



内存碎片率=

(float) rss/zmalloc_used_memory()



当内存不足时

- Random
- LRU
- TTL

LRU? Sample LRU……

```
/* volatile-lru and allkeys-lru policy */
else if (server.maxmemory_policy == REDIS_MAXMEMORY_ALLKEYS_LRU ||
    server.maxmemory_policy == REDIS_MAXMEMORY_VOLATILE_LRU)
{
    for (k = 0; k < server.maxmemory_samples; k++) {
        sds thiskey;
        long thisval;
        robj *o;

        de = dictGetRandomKey(dict);
        thiskey = dictGetKey(de);
        /* When policy is volatile-lru we need an additional lookup
         * to locate the real key, as dict is set to db->expires. */
        if (server.maxmemory_policy == REDIS_MAXMEMORY_VOLATILE_LRU)
            de = dictFind(db->dict, thiskey);
        o = dictGetVal(de);
        thisval = estimateObjectIdleTime(o);

        /* Higher idle time is better candidate for deletion */
        if (bestkey == NULL || thisval > bestval) {
            bestkey = thiskey;
            bestval = thisval;
        }
    }
}
```

overcommit_memory

- 0 ->表示内核将检查是否有足够的可用内存供应用进程使用
- 1 ->表示内核允许分配所有的物理内存，而不管当前的内存状态如何
- 2 ->表示内核允许分配超过所有物理内存和交换空间总和的内存


Q/A