<

23 从零实现 KV 存储--List 结构支持

LPUSH RPUSH

23 从零实现 KV 存储—List 结构支持

理论讲解



Go 编码



Rust 编码



前面的 Redis 数据结构设计中,List 结构的编码设计如下:

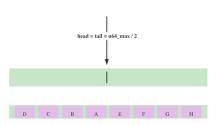
```
127.0.0.1:6379>
127.0.0.1:6379> LPUSH mylist a b c
(integer) 3
127.0.0.1:6379> RPUSH mylist e d f
(integer) 6
127.0.0.1:6379> LPOP mylist
'c"
127.0.0.1:6379> RPOP mylist
'f"
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379>
```

元数据

| | | | | | + | | | + |
|---------|---------|-------------------|-------------------|-----------------------------|---------------------------------------|--|---|---|
| type | expire | 1.4 | /ersion | size | h | ead | tail | 1 |
| (1byte) | (Ebyte) | 1 (| (8byte) | (Sbyte) | (8 | byte) | (8byte) | 1 |
| | (1byte) | (1byte) (Ebyte) | (1byte) (Ebyte) | (1byte) (Ebyte) (8byte) | (1byte) (Ebyte) (8byte) (Sbyte) | (1byte) (Ebyte) (8byte) (Sbyte) (8 | (1byte) (Ebyte) (8byte) (Sbyte) (8byte) | type expire version size head tail (1byte) (Ebyte) (8byte) (8byte) (8byte) (8byte) |

List 结构的元数据部分和 Hash、Set 比较类似,只是多了两个字段 head 和 tail。

List 数据结构可以看做是一个队列,可以在队列的头尾进行 Push、Pop 操作,因此我们可以使用一个标识来表示头 尾,在初始情况下, | head = tail = $U64_MAX$ / 2 ,



6 0

0

②



数据部分

```
1
2 key|version|index -> | value |
3
```

index 会根据 head 或者 tall 的值来确定,当在左边 Push 的时候,index 的值是 head-1,当在右边 head-1 ,当在右边 head-1 ,当在右边 head-1 ,当在右边 head-1 ,当在右边 head-1 ,当在右边 head-1 ,其由 head-1 ,其由

LPUSH

先查找元数据,如果不存在则初始化。

```
1 type listInternalKey struct {
2 key []byte
3 version int64
4 index uint64
5 }

1 pub(crate) struct ListInternalKey {
2 pub(crate) key: Veccub,
3 pub(crate) version: ut28,
4 pub(crate) index: u64,
5 }
```

构造数据部分的 key,其中 index 的值就是 $_{
m meta.head}$ - 1 ,调用存储引擎的接口存储数据,并且更新元数据。

RPUSH

和 LPush 基本类似,只是 Index 的值是 meta.tail 。

LPOP

先查找元数据,如果元数据不存在或者 key 下面没有任何数据,那么直接返回。

否则构造数据部分的 key,并且调用存储引擎的接口获取值。然后需要更新元数据,元数据的 size 需要递减,然后 meta.head += 1。

RPOP

和 LPop 基本类似,只是更新元数据的时候,是将元数据的 tall 递减, meta.tail -= 1。



输入评论

0

0