



数据部分

```
1                                     +-----+
2 key[version]index -> |   value   |
3                                     +-----+
```

index 会根据 head 或者 tail 的值来确定, 当在左边 Push 的时候, index 的值是 `head - 1`, 当在右边 push 的时候, index 的值就是 `tail`。

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 uint64
4     index  uint64
5 }
```

```
1 pub(crate) struct ListInternalKey {
2     pub(crate) key: Vec,
3     pub(crate) version: u128,
4     pub(crate) index: u64,
5 }
```

构造数据部分的 key, 其中 index 的值就是 `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 tail 递减, `meta.tail -= 1`。



真诚点赞, 手留余香

输入评论