

<<
21 从零实现 KV 存储—Hash 结构支持
HSET
HGET
HDEL

21 从零实现 KV 存储—Hash 结构支持

理论讲解

21 从零实现 KV 存储—Hash 结构支持

127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379> HSET myset a 100
(integer) 1
127.0.0.1:6379> HGET myset a
"100"
127.0.0.1:6379>
127.0.0.1:6379>

元数据

key =>	type	expire	version	size
1	(1byte)	(8byte)	(8byte)	(8byte)

数据部分

key version field =>	value
1	

这一节将会实现 Hash 数据结构中，最常用的三个命令，分别是 HSet、HGet、HDel。

Go 编码

Go 编码

127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379> HSET myset a 100
(integer) 1
127.0.0.1:6379> HGET myset a
"100"
127.0.0.1:6379>
127.0.0.1:6379>

元数据

key =>	type	expire	version	size
1	(1byte)	(8byte)	(8byte)	(8byte)

数据部分

key version field =>	value
1	

这一节将会实现 Hash 数据结构中，最常用的三个命令，分别是 HSet、HGet、HDel。

Rust 编码

Rust 编码

127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379> HSET myset a 100
(integer) 1
127.0.0.1:6379> HGET myset a
"100"
127.0.0.1:6379>
127.0.0.1:6379>

元数据

key =>	type	expire	version	size
1	(1byte)	(8byte)	(8byte)	(8byte)

数据部分

key version field =>	value
1	

这一节将会实现 Hash 数据结构中，最常用的三个命令，分别是 HSet、HGet、HDel。

```
127.0.0.1:6379>  
127.0.0.1:6379>  
127.0.0.1:6379> HSET myset a 100  
(integer) 1  
127.0.0.1:6379> HGET myset a  
"100"  
127.0.0.1:6379>  
127.0.0.1:6379>
```

在前面的 Redis 数据结构总体设计中，我们设计的 Hash 数据结构的编码如下：

元数据

1	key =>	type	expire	version	size
3		(1byte)	(8byte)	(8byte)	(8byte)
4					

在前面的讲述中已经解释了 version 这个字段的作用，主要用于快速删除一个 key。

数据部分

1	key version field =>	value
3		

这一节将会实现 Hash 数据结构中，最常用的三个命令，分别是 HSet、HGet、HDel。

HSET

首先获取元数据，如果元数据不存在的话，则说明是一个新的 Key 被添加了，需要初始化一个新的元数据结构体，元数据的定义如下：

```
1 type metadata struct {  
2     dataType byte  
3     expire    int64  
4     version   int64  
5     size      uint32  
6     head      uint64  
7     tail      uint64  
8 }
```

```
0 }
```

```
1 pub(crate) struct Metadata {  
2     pub(crate) data_type: RedisDataType,  
3     pub(crate) expire: u128,  
4     pub(crate) version: u128,  
5     pub(crate) size: u32,  
6     pub(crate) head: u64,  
7     pub(crate) tail: u64,  
8 }
```

其中 `dataType`、`expire`、`version`、`size` 是四个通用的属性，`head` 和 `tail` 是为 `List` 专用的。

数据部分编码后的 `key` 是 `key+version+field`，如果这个 `key` 不存在的话，则说明是新增的一个 `field`，那么元数据的 `size` 需要加上 1，然后再更新元数据。

如果存在，则说明 `field` 属于的 `value` 是存在的，则不需要更新元数据。

在讲总体设计的时候我也提到过，就是我们需要在更新元数据和数据部分的时候，使用 `WriteBatch`，保证更新的原子性。

HGET

首先根据 `key` 查询元数据，如果不存在的话则说明 `key` 不存在，否则判断 `value` 的类型，如果 `value` 的类型不是 `Hash`，则直接返回对应的错误。

然后我们根据 `key` 和元数据中的 `version` 字段，以及 `field` 编码出一个 `key`，然后根据这个 `key` 去获取实际的 `value`。

HDEL

这个命令是删除一个指定的 `field`，我们还是会首先查询元数据，并且判断类型，和 `HGet` 比较类似。

然后再更新元数据，主要是将 `size` 递减，然后删除实际的数据部分。



—— 真诚赞赏，手留余香 ——

输入评论

