Redis 位图与 HyperLogLog

作者：DongXiaokai0819
原文链接：https://ld246.com/article/1570687216410
来源网站：链滴

《Redis深度历险》读书笔记 -- 之位图与 HyperLogLog

位图

1、什么是位图?

位图并不是什么特殊的数据结构，而是定义在String类型上的一个面向字节操作的集合，也就是byte组，位图的最小单位是bit，每个bit的取值只能是0或者1。

2、位图的基本用法

● 位图数据结构是可以自扩展的，如果设置了某个偏移位置超出了现有的内容范围，就会自动将位数进行零扩充。

零存整取

首先我们先用python获取 ‘hello’ 的ASCII码的二进制值

```
>>> bin(ord('h'))
'0b1101000'
>>> bin(ord('e'))
'0b1100101'
>>> bin(ord('l'))
'0b1101100'
>>> bin(ord('o'))
'0b1101111'
```

由于位数组的顺序和字符的位顺序是相反的

所以hello所对应二进制位表示为为

01101000 01100101 01101100 01101100 01101111

12345678 9....

我们只需要在为1的地方设置为1即可

```
127.0.0.1:6379> setbit s 1 1
(integer) 0
127.0.0.1:6379> setbit s 2 1
(integer) 0
127.0.0.1:6379> setbit s 4 1
(integer) 0
127.0.0.1:6379> setbit s 9 1
(integer) 0
127.0.0.1:6379> setbit s 10 1
(integer) 0
127.0.0.1:6379> setbit s 13 1
(integer) 0
127.0.0.1:6379> setbit s 15 1
(integer) 0
127.0.0.1:6379> get s
```

"he"

零存零取

- 使用单个位操作设置位值，使用单个位操作获取具体位值

```
127.0.0.1:6379> setbit w 1 1
(integer) 0
127.0.0.1:6379> setbit w 4 1
(integer) 0
127.0.0.1:6379> getbit w 1
(integer) 1
127.0.0.1:6379> getbit w 2
(integer) 0
```

整存零取

- 使用字符串操作批量设置位值， 使用单个位操作获取具体位值

```
127.0.0.1:6379> set w h
OK
127.0.0.1:6379> getbit w 1
(integer) 1
127.0.0.1:6379> getbit w 2
(integer) 1
127.0.0.1:6379> getbit w 3
(integer) 0
127.0.0.1:6379> getbit w 4
(integer) 1
```

如果对应位的字节是不可打印字符，客户端会显示该字符的十六进制形式

```
127.0.0.1:6379> setbit x 0 1
(integer) 0
127.0.0.1:6379> setbit x 1 1
(integer) 0
127.0.0.1:6379> get x
"\xc0"
```

3、位图的统计和查找

- bitcount key [start end]统计指令，统计指定位置范围内1的个数， 可用来统计用户一共可签到多天。
- bitpos key [start end]查找指令，查找指定范围内出现的第一个0 或者 1， 可用来查找用户从哪天开始签到的。
- 但是start和end参数是字节索引，也就是说指定的范围必须是8的倍数，而不能任意指定。

```
127.0.0.1:6379> set w hello
OK
127.0.0.1:6379> bitcount w
(integer) 21
```

```
127.0.0.1:6379> bitcount w 0 0 #第一个字符中1 的位数
(integer) 3
127.0.0.1:6379> bitcount w 0 1
(integer) 7
127.0.0.1:6379> bitpos w 0 #第一个0 位
(integer) 0
127.0.0.1:6379> bitpos w 1
(integer) 1
127.0.0.1:6379> bitpos w 1 1 1 #从第二个字符算起，第一个1位
(integer) 9
127.0.0.1:6379> bitpos w 1 2 2
(integer) 17
```

4、位图的bitfield指令

- setbit与getbit 指定位的值都是单个位的，如果要操作多个位就需要用到bitfield指令了
- bitfield指令有三个子指令，分别为 ger 、 set 、 incrby，都可以对指定位片段进行读写，但是最只能处理64个连续的位，如果超出，就得使用多个子指令。
- bitfield 的incrby子指令，可能会产生溢出，bitfield对此有自己的溢出策略overflow子指令，默认wrap（折返）、fail（失败）——报错不执行 、sat（截断）——如果溢出就停留在最大值或者最小。overflow指令只会影响接下来的第一条指令。

不知道为什么我的客户端，在敲这条命令的时候报错，所以代码就先不贴了，待以后再试试

```
127.0.0.1:6379> set w hello
OK
127.0.0.1:6379> bitfield w get u4 0
(error) ERR unknown command 'bitfield'
127.0.0.1:6379> bitfield w get u4 0
```

HyperLogLog

Set的高级数据结构，HyperLogLog，它提供了不精确的去重计数方案，标准误差是0.81%。

应用场景：某个网页的每天的UV数据

HyperLogLog 使用方法

- pfadd 增加计数 ， 使用方法同set的sadd
- pfcount 获取计数 ， 使用方法同set的scard，直接获取计数值
- pfmerge 用于将多个pf计数值合并起来形成一个新的pf值，应用：多个页面数据进行合并时使用

```
127.0.0.1:6379> pfadd codehole user1
(integer) 1
127.0.0.1:6379> pfcount codehole
(integer) 1
127.0.0.1:6379> pfadd codehole user2
(integer) 1
127.0.0.1:6379> pfadd codehole user3
(integer) 1
127.0.0.1:6379> pfadd codehole user4
```

```
(integer) 1
127.0.0.1:6379> pfadd codehole user5
(integer) 1
127.0.0.1:6379> pfcount codehole
(integer) 5
```

接下来增大数据量

```
public class PfTest {
    public static void main(String[] args) {
        Jedis jedis = new Jedis("127.0.0.1");
        for (int i = 0; i < 100000; i++){
            jedis.pfadd("codehole","user"+i);
        }
        long total = jedis.pfcount("codehole");
        System.out.println(total);
        jedis.close();
    }
}

---------
99715

Process finished with exit code 0
```

将上述程序执行俩次，会发现俩次的结果一样，这说明pf确实有去重功能，而且误差率也不是很高。