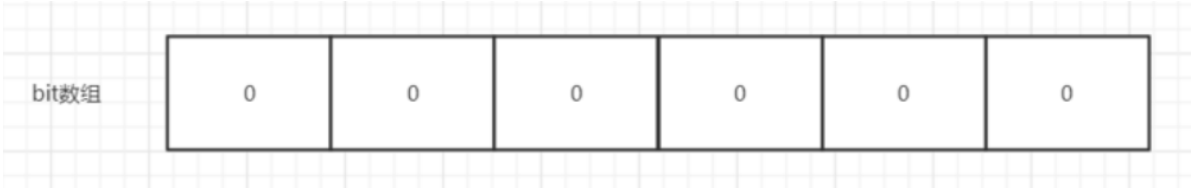
# 布隆过滤器

## 什么是布隆过滤器?

布隆过滤器（Bloom Filter）是一个叫做 Bloom 的老哥于 1970 年提出的。我们可以把它看作由二进制向量（或者说位数组）和一系列随机映射函数（哈希函数）两部分组成的数据结构。相比于我们平时常用的的 List、Map 、Set 等数据结构，它占用空间更少并且效率更高，但是缺点是其返回的结果是概率性的，而不是非常准确的。理论情况下添加到集合中的元素越多，误报的可能性就越大。并且，存放在布隆过滤器的数据不容易删除。




位数组中的每个元素都只占用 1 bit ，并且每个元素只能是 0 或者 1。这样申请一个 100w 个元素的位数组只占用 1000000Bit / 8 = 125000 Byte = 125000/1024 kb ≈ 122kb 的空间。

总结：**一个名叫 Bloom 的人提出了一种来检索元素是否在给定大集合中的数据结构，这种数据结构是高效且性能很好的，但缺点是具有一定的错误识别率和删除难度。并且，理论情况下，添加到集合中的元素越多，误报的可能性就越大。**

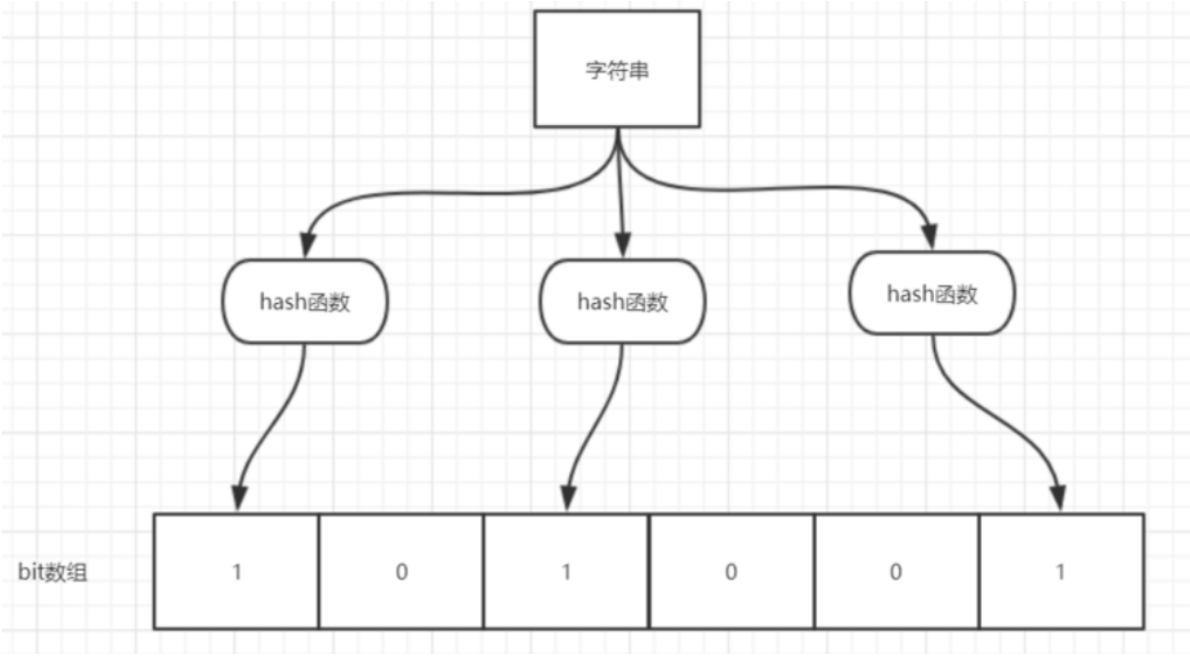
# 原理介绍

**当一个元素加入布隆过滤器中的时候，会进行如下操作：**

1. 使用布隆过滤器中的哈希函数对元素值进行计算，得到哈希值（有几个哈希函数得到几个哈希值）。
2. 根据得到的哈希值，在位数组中把对应下标的值置为 1。

**当我们需要判断一个元素是否存在于布隆过滤器的时候，会进行如下操作：**

1. 对给定元素再次进行相同的哈希计算；
2. 得到值之后判断位数组中的每个元素是否都为 1，如果值都为 1，那么说明这个值在布隆过滤器中，如果存在一个值不为 1，说明该元素不在布隆过滤器中。




当字符串存储要加入到布隆过滤器中时，该字符串首先由多个哈希函数生成不同的哈希值，然后将对应的位数组的下标设置为 1（当位数组初始化时，所有位置均为 0）。当第二次存储相同字符串时，因为先前的对应位置已设置为 1，所以很容易知道此值已经存在（去重非常方便）。

如果我们需要判断某个字符串是否在布隆过滤器中时，只需要对给定字符串再次进行相同的哈希计算，得到值之后判断位数组中的每个元素是否都为 1，如果值都为 1，那么说明这个值在布隆过滤器中，如果存在一个值不为 1，说明该元素不在布隆过滤器中。

**不同的字符串可能哈希出来的位置相同，这种情况我们可以适当增加位数组大小或者调整我们的哈希函数。**

综上，我们可以得出：**布隆过滤器说某个元素存在，小概率会误判。布隆过滤器说某个元素不在，那么这个元素一定不在。**

# 布隆过滤器使用场景

1. 判断给定数据是否存在：比如判断一个数字是否存在于包含大量数字的数字集中（数字集很大，5 亿以上！）、 防止缓存穿透（判断请求的数据是否有效避免直接绕过缓存请求数据库）等等、邮箱的垃圾邮件过滤、黑名单功能等等。
2. 去重：比如爬给定网址的时候对已经爬取过的 URL 去重。

# Guava 布隆过滤器

引入 Guava 的依赖：

```xml
        <dependency>
            <groupId>com.google.guava</groupId>
            <artifactId>guava</artifactId>
            <version>28.0-jre</version>
        </dependency>
```

```java
package mao;

import com.google.common.hash.BloomFilter;
import com.google.common.hash.Funnels;

/**
 * Project name(项目名称)：布隆过滤器
 * Package(包名): mao
 * Class(类名): GuavaBloomFilter
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2023/2/27
 * Time(创建时间)： 20:01
 * Version(版本): 1.0
 * Description(描述)： 无
 */

public class GuavaBloomFilter
{
    public static void main(String[] args)
    {
        //布隆过滤器对象，创建最多存放最多2000个整数的布隆过滤器，误判率为0.01
        BloomFilter<Integer> bloomFilter = BloomFilter.create(
                Funnels.integerFunnel(), 2000, 0.01);

        //判断是否存在
        System.out.println(bloomFilter.mightContain(100));
        System.out.println(bloomFilter.mightContain(101));
        System.out.println(bloomFilter.mightContain(102));

        //设置值
        bloomFilter.put(100);
        bloomFilter.put(101);
        bloomFilter.put(102);

        //判断是否存在
        System.out.println(bloomFilter.mightContain(100));
        System.out.println(bloomFilter.mightContain(101));
        System.out.println(bloomFilter.mightContain(102));
        System.out.println(bloomFilter.mightContain(103));
    }
}
```

运行结果：

```
false
false
false
true
true
true
false
```

```
package mao;

import com.google.common.hash.BloomFilter;
import com.google.common.hash.Funnels;

/**
 * Project name(项目名称)：布隆过滤器
 * Package(包名): mao
 * Class(类名): GuavaBloomFilter2
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2023/2/27
 * Time(创建时间)： 20:12
 * Version(版本): 1.0
 * Description(描述)： 测试布隆过滤器误差
 */

public class GuavaBloomFilter2
{
    public static void main(String[] args)
    {
        testGuavaBloomFilter(10000, 0.01);
        testGuavaBloomFilter(10000, 0.02);
        testGuavaBloomFilter(10000, 0.05);
        testGuavaBloomFilter(10000, 0.1);
        testGuavaBloomFilter(10000, 0.5);

        System.out.println("------");
        testGuavaBloomFilter(3000, 0.01);
        testGuavaBloomFilter(3000, 0.02);
        testGuavaBloomFilter(3000, 0.05);
        testGuavaBloomFilter(3000, 0.1);
        testGuavaBloomFilter(3000, 0.5);

        System.out.println("------");
        testGuavaBloomFilter(2000, 0.01);
```

```
        testGuavaBloomFilter(2000, 0.02);
        testGuavaBloomFilter(2000, 0.05);
        testGuavaBloomFilter(2000, 0.1);
        testGuavaBloomFilter(2000, 0.5);

        System.out.println("------");
        testGuavaBloomFilter(1000, 0.01);
        testGuavaBloomFilter(1000, 0.02);
        testGuavaBloomFilter(1000, 0.05);
        testGuavaBloomFilter(1000, 0.1);
        testGuavaBloomFilter(1000, 0.5);
        testGuavaBloomFilter(1000, 0.005);
        testGuavaBloomFilter(1000, 0.001);
        testGuavaBloomFilter(1000, 0.0001);

        System.out.println("------");
        testGuavaBloomFilter(500, 0.01);
        testGuavaBloomFilter(500, 0.02);
        testGuavaBloomFilter(500, 0.05);
        testGuavaBloomFilter(500, 0.1);
        testGuavaBloomFilter(500, 0.5);
        testGuavaBloomFilter(500, 0.005);
        testGuavaBloomFilter(500, 0.001);
        testGuavaBloomFilter(500, 0.0001);

        System.out.println("------");
        testGuavaBloomFilter(200, 0.01);
        testGuavaBloomFilter(200, 0.02);
        testGuavaBloomFilter(200, 0.05);
        testGuavaBloomFilter(200, 0.1);
        testGuavaBloomFilter(200, 0.5);
        testGuavaBloomFilter(200, 0.005);
        testGuavaBloomFilter(200, 0.001);
        testGuavaBloomFilter(200, 0.0001);

        System.out.println("------");
        testGuavaBloomFilter(100, 0.01);
        testGuavaBloomFilter(100, 0.02);
        testGuavaBloomFilter(100, 0.05);
        testGuavaBloomFilter(100, 0.1);
        testGuavaBloomFilter(100, 0.5);
        testGuavaBloomFilter(100, 0.005);
        testGuavaBloomFilter(100, 0.001);
        testGuavaBloomFilter(100, 0.0001);
    }


    /**
     * 测试Guava的布隆过滤器
     *
     * @param expectedInsertions 布隆过滤器最多存放的数量
     * @param fpp                误差
     */
    public static void testGuavaBloomFilter(int expectedInsertions, double
fpp)
```

```
    {
        //布隆过滤器对象，创建最多存放最多2000个整数的布隆过滤器，误判率为0.01
        BloomFilter<Integer> bloomFilter = BloomFilter.create(
                Funnels.integerFunnel(), expectedInsertions, fpp);

        //1500次循环
        for (int i = 0; i < 1500; i++)
        {
            if (i % 3 == 0)
            {
                continue;
            }
            //将i的值% 3 不等于 0 的值放进去
            bloomFilter.put(i);
        }

        //存在的计数
        int count = 0;
        //统计
        for (int i = 0; i < 1500; i++)
        {
            //判断是否存在
            boolean b = bloomFilter.mightContain(i);
            //System.out.println(i + " --> " + b);
            //可能存在
            if (b)
            {
                count++;
            }
        }
        System.out.println("最大数量：" + expectedInsertions + "  误差：" +
fpp);
        System.out.println("预期结果：1000，最终结果：" + count);
        System.out.println();
    }
}
```

运行结果：

```
最大数量：10000  误差：0.01
预期结果：1000，最终结果：1000

最大数量：10000  误差：0.02
预期结果：1000，最终结果：1000

最大数量：10000  误差：0.05
预期结果：1000，最终结果：1000

最大数量：10000  误差：0.1
预期结果：1000，最终结果：1000

最大数量：10000  误差：0.5
```

```
预期结果：1000，最终结果：1030

------
最大数量：3000  误差：0.01
预期结果：1000，最终结果：1000

最大数量：3000  误差：0.02
预期结果：1000，最终结果：1000

最大数量：3000  误差：0.05
预期结果：1000，最终结果：1001

最大数量：3000  误差：0.1
预期结果：1000，最终结果：1001

最大数量：3000  误差：0.5
预期结果：1000，最终结果：1095

------
最大数量：2000  误差：0.01
预期结果：1000，最终结果：1000

最大数量：2000  误差：0.02
预期结果：1000，最终结果：1000

最大数量：2000  误差：0.05
预期结果：1000，最终结果：1002

最大数量：2000  误差：0.1
预期结果：1000，最终结果：1013

最大数量：2000  误差：0.5
预期结果：1000，最终结果：1137

------
最大数量：1000  误差：0.01
预期结果：1000，最终结果：1006

最大数量：1000  误差：0.02
预期结果：1000，最终结果：1011

最大数量：1000  误差：0.05
预期结果：1000，最终结果：1021

最大数量：1000  误差：0.1
预期结果：1000，最终结果：1053

最大数量：1000  误差：0.5
预期结果：1000，最终结果：1230

最大数量：1000  误差：0.005
预期结果：1000，最终结果：1000

最大数量：1000  误差：0.001
预期结果：1000，最终结果：1001
```

```
69
70  最大数量：1000  误差：1.0E-4
71  预期结果：1000，最终结果：1000
72
73  ------
74  最大数量：500  误差：0.01
75  预期结果：1000，最终结果：1075
76
77  最大数量：500  误差：0.02
78  预期结果：1000，最终结果：1106
79
80  最大数量：500  误差：0.05
81  预期结果：1000，最终结果：1128
82
83  最大数量：500  误差：0.1
84  预期结果：1000，最终结果：1175
85
86  最大数量：500  误差：0.5
87  预期结果：1000，最终结果：1375
88
89  最大数量：500  误差：0.005
90  预期结果：1000，最终结果：1060
91
92  最大数量：500  误差：0.001
93  预期结果：1000，最终结果：1037
94
95  最大数量：500  误差：1.0E-4
96  预期结果：1000，最终结果：1011
97
98  ------
99  最大数量：200  误差：0.01
100 预期结果：1000，最终结果：1423
101
102 最大数量：200  误差：0.02
103 预期结果：1000，最终结果：1438
104
105 最大数量：200  误差：0.05
106 预期结果：1000，最终结果：1421
107
108 最大数量：200  误差：0.1
109 预期结果：1000，最终结果：1442
110
111 最大数量：200  误差：0.5
112 预期结果：1000，最终结果：1479
113
114 最大数量：200  误差：0.005
115 预期结果：1000，最终结果：1399
116
117 最大数量：200  误差：0.001
118 预期结果：1000，最终结果：1375
119
120 最大数量：200  误差：1.0E-4
121 预期结果：1000，最终结果：1293
122
123 ------
```

```
最大数量：100  误差：0.01
预期结果：1000，最终结果：1497

最大数量：100  误差：0.02
预期结果：1000，最终结果：1500

最大数量：100  误差：0.05
预期结果：1000，最终结果：1497

最大数量：100  误差：0.1
预期结果：1000，最终结果：1497

最大数量：100  误差：0.5
预期结果：1000，最终结果：1494

最大数量：100  误差：0.005
预期结果：1000，最终结果：1498

最大数量：100  误差：0.001
预期结果：1000，最终结果：1498

最大数量：100  误差：1.0E-4
预期结果：1000，最终结果：1481
```

当 `mightContain()` 方法返回 *true* 时，我们可以 99%确定该元素在过滤器中，当过滤器返回 *false* 时，我们可以 100%确定该元素不存在于过滤器中。

**Guava 提供的布隆过滤器的实现还是很不错的，但是它有一个重大的缺陷就是只能单机使用（另外，容量扩展也不容易），而现在互联网一般都是分布式的场景。为了解决这个问题，我们就需要用到 Redis 中的布隆过滤器了。**

# Redis 中的布隆过滤器

## 介绍

Redis v4.0 之后有了 Module（模块/插件） 功能，Redis Modules 让 Redis 可以使用外部模块扩展其功能 。布隆过滤器就是其中的 Module。

https://redis.io/resources/modules/

另外，官网推荐了一个 RedisBloom 作为 Redis 布隆过滤器的 Module，地址：https://github.com/RedisBloom/RedisBloom
其他还有：

- redis-lua-scaling-bloom-filter（lua 脚本实现）：https://github.com/erikdubbelboer/redis-lua-scaling-bloom-filter
- pyreBloom（Python 中的快速 Redis 布隆过滤器） ：https://github.com/seomoz/pyreBloom

# 使用 Docker 安装

库：[redislabs/rebloom - Docker Image | Docker Hub]

## 第一步：搜索

命令：**docker search redislabs/rebloom**

```
PS C:\Users\mao\Desktop> docker search redislabs/rebloom
NAME                DESCRIPTION                                     STARS
OFFICIAL   AUTOMATED
redislabs/rebloom   A probablistic datatypes module for Redis   22
        [OK]
PS C:\Users\mao\Desktop>
```

## 第二步：拉取镜像

命令：**docker pull redislabs/rebloom**

```
PS C:\Users\mao\Desktop> docker pull redislabs/rebloom
Using default tag: latest
latest: Pulling from redislabs/rebloom
284055322776: Pull complete
b7286e96019c: Pull complete
8d789d3c47a4: Pull complete
3806fdc99c73: Pull complete
e751a0826f38: Pull complete
ef5499a774c1: Pull complete
c72d4acf12b3: Pull complete
38d98e95a943: Pull complete
4a50810bd519: Pull complete
209cee60b260: Pull complete
605e8f893c4b: Pull complete
Digest:
sha256:182ba5c55c8b2b5758edda4682f8f7a3711efe025d593583ecf1bf3ed9b4f55e
Status: Downloaded newer image for redislabs/rebloom:latest
docker.io/redislabs/rebloom:latest
PS C:\Users\mao\Desktop>
```

### 第三步：查看镜像是否拉取成功

命令：**docker images**

```
PS C:\Users\mao\Desktop> docker images
REPOSITORY          TAG       IMAGE ID       CREATED         SIZE
redislabs/rebloom   latest    66d626dc1387   15 months ago   147MB
PS C:\Users\mao\Desktop>
```

### 第四步：运行

命令：docker run -p 16379:6379  -d --name redis-redisbloom redislabs/rebloom

```
PS C:\Users\mao> docker run -p 16379:6379  -d --name redis-redisbloom
redislabs/rebloom
0a3197e86f21545c46e581d1b64aca3aa6a641144b9e38518fc50f8cedca1917
PS C:\Users\mao>
```

### 第五步：查看是否运行成功

命令：**docker ps**

```
PS C:\Users\mao> docker ps
CONTAINER ID   IMAGE               COMMAND                  CREATED
STATUS          PORTS                     NAMES
0a3197e86f21   redislabs/rebloom   "docker-entrypoint.s…"   14 seconds ago
Up 13 seconds   0.0.0.0:16379->6379/tcp   redis-redisbloom
PS C:\Users\mao>
```

### 第六步：进入容器

命令：**docker exec -it redis-redisbloom /bin/bash**

```
PS C:\Users\mao> docker exec -it redis-redisbloom /bin/bash
root@0a3197e86f21:/data# ls -l
total 0
root@0a3197e86f21:/data# pwd
/data
root@0a3197e86f21:/data# cd ..
root@0a3197e86f21:/#
```

### 第七步：使用redis-cli链接redis

命令：**redis-cli**

```
root@0a3197e86f21:/# redis-cli
127.0.0.1:6379> ping
PONG
127.0.0.1:6379>
```

如果是外部机，则需要使用命令：**redis-cli -p 16379**

```
PS C:\Users\mao> redis-cli -p 16379
127.0.0.1:16379> ping
PONG
127.0.0.1:16379>
```

## 常用命令

key : 布隆过滤器的名称，item : 添加的元素

### BF.ADD

将元素添加到布隆过滤器中，如果该过滤器尚不存在，则创建该过滤器。格式：`BF.ADD {key} {item}`

```
127.0.0.1:6379> BF.ADD filter 1
(integer) 1
127.0.0.1:6379> BF.ADD filter 2
(integer) 1
127.0.0.1:6379> BF.ADD filter 3
(integer) 1
127.0.0.1:6379> BF.ADD filter 5
(integer) 1
127.0.0.1:6379>
```

## BF.MADD

将一个或多个元素添加到"布隆过滤器"中，并创建一个尚不存在的过滤器。该命令的操作方式 `BF.ADD` 与之相同，只不过它允许多个输入并返回多个值。格式：`BF.MADD {key} {item} [item ...]`

```
127.0.0.1:6379> BF.MADD filter2 1 2 7 9 10
1) (integer) 1
2) (integer) 1
3) (integer) 1
4) (integer) 1
5) (integer) 1
127.0.0.1:6379> BF.MADD filter2 2 8
1) (integer) 0
2) (integer) 1
127.0.0.1:6379>
```

## BF.EXISTS

确定元素是否在布隆过滤器中存在。格式：`BF.EXISTS {key} {item}`

```
127.0.0.1:6379> BF.EXISTS filter 1
(integer) 1
127.0.0.1:6379> BF.EXISTS filter 2
(integer) 1
127.0.0.1:6379> BF.EXISTS filter 3
(integer) 1
127.0.0.1:6379> BF.EXISTS filter 4
(integer) 0
127.0.0.1:6379> BF.EXISTS filter 5
(integer) 1
127.0.0.1:6379> BF.EXISTS filter2 5
(integer) 0
127.0.0.1:6379> BF.EXISTS filter2 7
```

```
(integer) 1
127.0.0.1:6379> BF.EXISTS filter2 8
(integer) 1
127.0.0.1:6379> BF.EXISTS filter2 3
(integer) 0
127.0.0.1:6379>
```

## BF.MEXISTS

确定一个或者多个元素是否在布隆过滤器中存在。格式：`BF.MEXISTS {key} {item} [item ...]`

```
127.0.0.1:6379> BF.MEXISTS filter 1 2 3 4 5 6 7 8 9 10 11
 1) (integer) 1
 2) (integer) 1
 3) (integer) 1
 4) (integer) 0
 5) (integer) 1
 6) (integer) 0
 7) (integer) 0
 8) (integer) 0
 9) (integer) 0
10) (integer) 0
11) (integer) 0
127.0.0.1:6379>
```

```
127.0.0.1:6379> BF.MEXISTS filter2 1 2 3 4 5 6 7 8 9 10 11
 1) (integer) 1
 2) (integer) 1
 3) (integer) 0
 4) (integer) 0
 5) (integer) 0
 6) (integer) 0
 7) (integer) 1
 8) (integer) 1
 9) (integer) 1
10) (integer) 1
11) (integer) 0
127.0.0.1:6379>
```

## BF. RESERVE

命令的格式如下：

`BF. RESERVE {key} {error_rate} {capacity} [EXPANSION expansion]`

参数的具体含义：

1. key：布隆过滤器的名称
2. error_rate : 期望的误报率。该值必须介于 0 到 1 之间。例如，对于期望的误报率 0.1%（1000 中为 1），error_rate 应该设置为 0.001。该数字越接近零，则每个项目的内存消耗越大，并且每个操作的 CPU 使用率越高。
3. capacity: 过滤器的容量。当实际存储的元素个数超过这个值之后，性能将开始下降。实际的降级将取决于超出限制的程度。随着过滤器元素数量呈指数增长，性能将线性下降。

可选参数：

- expansion：如果创建了一个新的子过滤器，则其大小将是当前过滤器的大小乘以`expansion`。默认扩展值为 2。这意味着每个后续子过滤器将是前一个子过滤器的两倍。

```
127.0.0.1:6379> BF.RESERVE filter3 0.001 500
OK
127.0.0.1:6379>
```

# Java操作Redis布隆过滤器

使用RESP2.0协议来操作redis服务端

## RedisInformation类

```
package mao;

import java.io.InputStream;
import java.util.Properties;

/**
 * Project name(项目名称)：布隆过滤器
 * Package(包名): mao
 * Class(类名): RedisInformation
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2023/2/27
 * Time(创建时间)： 21:59
 * Version(版本): 1.0
 * Description(描述)： redis服务的信息对象
 */

public class RedisInformation
{

```

```
22      /**
23       * redis服务的ip
24       */
25      private static final String host;
26      /**
27       * redis服务的端口号
28       */
29      private static final int port;
30      /**
31       * redis服务的密码
32       */
33      private static final String password;
34
35      /**
36       * 单行字符串
37       */
38      public static final char SINGLE_LINE_STRING = '+';
39      /**
40       * 异常或者错误
41       */
42      public static final char ERROR = '-';
43      /**
44       * 数值
45       */
46      public static final char NUMBER = ':';
47      /**
48       * 多行字符串
49       */
50      public static final char MULTILINE_STRING = '$';
51      /**
52       * 数组
53       */
54      public static final char ARRAY = '*';
55
56
57      static
58      {
59          try
60          {
61              //从类路径里获取配置信息
62              InputStream inputStream =
RedisInformation.class.getClassLoader().getResourceAsStream("redis.propertie
s");
63              Properties properties = new Properties();
64              //加载配置到properties
65              properties.load(inputStream);
66              //ip地址
67              host = properties.getProperty("redis.host");
68              //端口号
69              port = Integer.parseInt(properties.getProperty("redis.port"));
70              //密码
71              password = properties.getProperty("redis.password");
72          }
73          catch (Exception e)
74          {
```

```
            throw new RuntimeException(e);
        }
    }


    public static String getHost()
    {
        return host;
    }

    public static int getPort()
    {
        return port;
    }

    public static String getPassword()
    {
        return password;
    }
}
```

## 定义接口RedisBloomFilter

```
package mao;

import java.util.List;

/**
 * Project name(项目名称)：布隆过滤器
 * Package(包名): mao
 * Interface(接口名): RedisBloomFilter
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2023/2/27
 * Time(创建时间)： 22:05
 * Version(版本): 1.0
 * Description(描述)： redis布隆过滤器客户端
 */

public interface RedisBloomFilter
{
    /**
     * 关闭链接
     */
    void close();

    /**
```

```
     * 将元素添加到布隆过滤器中，如果该过滤器尚不存在，则创建该过滤器。
     *
     * @param filterKey 布隆过滤器的名称
     * @param item      添加的元素
     * @return boolean
     */
    boolean add(String filterKey, String item);

    /**
     * 将一个或多个元素添加到“布隆过滤器”中，并创建一个尚不存在的过滤器。
     * 该命令的操作方式`BF.ADD`与之相同，只不过它允许多个输入并返回多个值。
     *
     * @param filterKey 布隆过滤器的名称
     * @param items     添加的元素
     * @return {@link List}<{@link Boolean}>
     */
    List<Boolean> mAdd(String filterKey, String... items);


    /**
     * 确定元素是否在布隆过滤器中存在
     *
     * @param filterKey 布隆过滤器的名称
     * @param item      添加的元素
     * @return boolean 如果存在，则为true，反之为false
     */
    boolean exists(String filterKey, String item);


    /**
     * 确定一个或者多个元素是否在布隆过滤器中存在
     * 该命令的操作方式`BF.EXISTS`与之相同，只不过它允许多个输入并返回多个值。
     *
     * @param filterKey 布隆过滤器的名称
     * @param items     添加的元素
     * @return {@link List}<{@link Boolean}>
     */
    List<Boolean> mExists(String filterKey, String... items);


    /**
     * 储备
     *
     * @param filterKey  布隆过滤器的名称
     * @param error_rate 期望的误报率。该值必须介于 0 到 1 之间。例如，对于期望的误报率 0.1%（1000 中为 1），
     *                   error_rate 应该设置为 0.001。
     *                   该数字越接近零，则每个项目的内存消耗越大，并且每个操作的 CPU 使用率越高。
     * @param capacity   过滤器的容量。当实际存储的元素个数超过这个值之后，性能将开始下降。
     *                   实际的降级将取决于超出限制的程度。随着过滤器元素数量呈指数增长，性能将线性下降。
     * @return boolean
     */
```

```
    boolean reserve(String filterKey, float error_rate, int capacity);
}
```

## 实现接口

```
package mao;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;
import java.nio.charset.StandardCharsets;
import java.util.ArrayList;
import java.util.List;
import java.util.Objects;

/**
 * Project name(项目名称)：布隆过滤器
 * Package(包名): mao
 * Class(类名): RedisBloomFilterImpl
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2023/2/27
 * Time(创建时间)： 22:05
 * Version(版本): 1.0
 * Description(描述)： redis布隆过滤器客户端接口实现类
 */

public class RedisBloomFilterImpl implements RedisBloomFilter
{
    private final Socket socket;
    private final PrintWriter printWriter;
    private final BufferedReader bufferedReader;


    /**
     * Instantiates a new Redis client.
     */
    public RedisBloomFilterImpl()
    {
        try
        {
            //连接redis
            socket = new Socket(RedisInformation.getHost(),
RedisInformation.getPort());
            //获取输入流
            bufferedReader = new BufferedReader(new
InputStreamReader(socket.getInputStream(), StandardCharsets.UTF_8));
            //获取输出流
```

```
            printWriter = new PrintWriter(socket.getOutputStream());
            //身份认证
            if (RedisInformation.getPassword() != null)
            {
                sendRequest("auth", RedisInformation.getPassword());
                Object response = getResponse();
                System.out.println("密码验证成功");
            }
        }
        catch (IOException e)
        {
            throw new RuntimeException(e);
        }
    }

    /**
     * Instantiates a new Redis client.
     *
     * @param host     the host
     * @param port     the port
     * @param password the password
     */
    public RedisBloomFilterImpl(String host, int port, String password)
    {
        try
        {
            //连接redis
            socket = new Socket(host, port);
            //获取输入流
            bufferedReader = new BufferedReader(new
InputStreamReader(socket.getInputStream(), StandardCharsets.UTF_8));
            //获取输出流
            printWriter = new PrintWriter(socket.getOutputStream());
            //身份认证
            if (password != null)
            {
                sendRequest("auth", password);
                Object response = getResponse();
                System.out.println("密码验证成功：" + response);
            }
        }
        catch (IOException e)
        {
            throw new RuntimeException(e);
        }
    }

    /**
     * Instantiates a new Redis client.
     *
     * @param host the host
     * @param port the port
     */
    public RedisBloomFilterImpl(String host, int port)
    {
```

```
        try
        {
            //连接redis
            socket = new Socket(host, port);
            //获取输入流
            bufferedReader = new BufferedReader(new
InputStreamReader(socket.getInputStream(), StandardCharsets.UTF_8));
            //获取输出流
            printWriter = new PrintWriter(socket.getOutputStream());
        }
        catch (IOException e)
        {
            throw new RuntimeException(e);
        }
    }


    /**
     * 关闭redis客户端与redis服务端的连接
     */
    public void close()
    {
        try
        {
            if (printWriter != null)
            {
                printWriter.close();
            }
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
        try
        {
            if (bufferedReader != null)
            {
                bufferedReader.close();
            }
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
        try
        {
            if (socket != null)
            {
                socket.close();
            }
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
```

```
153
154     }
155
156     /**
157      * 发送请求
158      *
159      * @param args 发起请求的参数，参数数量不一定
160      */
161     private void sendRequest(String... args)
162     {
163         //先写入元素个数，数组，换行
164         printWriter.println("*" + args.length);
165         //剩余的都是数组，遍历添加
166         for (String arg : args)
167         {
168             //$为多行字符串，长度
169             printWriter.println("$" +
arg.getBytes(StandardCharsets.UTF_8).length);
170             printWriter.println(arg);
171         }
172         //刷新
173         printWriter.flush();
174     }
175
176     /**
177      * 获取发送请求后的响应
178      *
179      * @return Object对象
180      */
181     private Object getResponse()
182     {
183         try
184         {
185             //获取当前前缀，因为要判断是什么类型
186             char prefix = (char) bufferedReader.read();
187             //判断是什么类型
188             if (prefix == RedisInformation.SINGLE_LINE_STRING)
189             {
190                 //单行字符串
191                 //直接读一行，读到换行符
192                 return bufferedReader.readLine();
193             }
194             if (prefix == RedisInformation.ERROR)
195             {
196                 //错误
197                 //抛出运行时异常
198                 throw new RuntimeException(bufferedReader.readLine());
199             }
200             if (prefix == RedisInformation.NUMBER)
201             {
202                 //数值
203                 //转数字
204                 return Integer.valueOf(bufferedReader.readLine());
205             }
206             if (prefix == RedisInformation.MULTILINE_STRING)
```

```
            {
                //多行字符串
                //先获取长度
                int length = Integer.parseInt(bufferedReader.readLine());
                //判断数组是否为空
                if (length == -1 || length == 0)
                {
                    //不存在或者数组为空
                    //返回空字符串
                    return "";
                }
                //不为空，读取
                return bufferedReader.readLine();
            }
            if (prefix == RedisInformation.ARRAY)
            {
                //数组
                return readBulkString();
            }
        }
        catch (Exception e)
        {
            throw new RuntimeException(e);
        }
        return null;
    }

    /**
     * 读取数组
     *
     * @return List<Object>
     * @throws IOException IOException
     */
    private List<Object> readBulkString() throws IOException
    {
        //获取当前数组的大小
        int size = Integer.parseInt(bufferedReader.readLine());
        //判断数组大小
        if (size == 0 || size == -1)
        {
            //返回null
            return null;
        }
        //数组有值
        //构建集合
        List<Object> list = new ArrayList<>(3);
        //遍历获取
        for (int i = 0; i < size; i++)
        {
            try
            {
                //递归获取
                list.add(getResponse());
            }
            catch (Exception e)
```

```
            {
                //异常加入到集合
                list.add(e);
            }
        }
        //返回
        return list;
    }

    /**
     * 将元素添加到布隆过滤器中，如果该过滤器尚不存在，则创建该过滤器。
     *
     * @param filterKey 布隆过滤器的名称
     * @param item      添加的元素
     * @return boolean
     */
    public boolean add(String filterKey, String item)
    {
        //发送命令
        this.sendRequest("BF.ADD", filterKey, item);
        //读取结果
        String response =
Objects.requireNonNull(this.getResponse()).toString();
        if (Objects.equals(response, "1"))
        {
            return true;
        }
        if (Objects.equals(response, "0"))
        {
            return false;
        }
        throw new
RuntimeException(Objects.requireNonNull(response).toString());
    }


    /**
     * 将一个或多个元素添加到“布隆过滤器”中，并创建一个尚不存在的过滤器。
     * 该命令的操作方式`BF.ADD`与之相同，只不过它允许多个输入并返回多个值。
     *
     * @param filterKey 布隆过滤器的名称
     * @param items     添加的元素
     * @return {@link List}<{@link Boolean}>
     */
    public List<Boolean> mAdd(String filterKey, String... items)
    {
        if (filterKey == null)
        {
            return null;
        }

        String[] args = new String[items.length + 2];
        args[0] = "BF.MADD";
        args[1] = filterKey;
        System.arraycopy(items, 0, args, 2, items.length);
```

```
315         sendRequest(args);
316         String r = getResponse().toString();
317         r = r.substring(1, r.length() - 1);
318         String[] split = r.split(", ");
319         List<Boolean> list = new ArrayList<>(items.length);
320         for (String s : split)
321         {
322             if (Objects.equals(s, "1"))
323             {
324                 list.add(true);
325                 continue;
326             }
327             if (Objects.equals(s, "0"))
328             {
329                 list.add(false);
330                 continue;
331             }
332             throw new RuntimeException(Objects.requireNonNull(s));
333         }
334         return list;
335     }
336 
337 
338     /**
339      * 确定元素是否在布隆过滤器中存在
340      *
341      * @param filterKey 布隆过滤器的名称
342      * @param item      添加的元素
343      * @return boolean 如果存在，则为true，反之为false
344      */
345     public boolean exists(String filterKey, String item)
346     {
347         //发送命令
348         this.sendRequest("BF.EXISTS", filterKey, item);
349         //读取结果
350         String response =
Objects.requireNonNull(this.getResponse()).toString();
351         if (Objects.equals(response, "1"))
352         {
353             return true;
354         }
355         if (Objects.equals(response, "0"))
356         {
357             return false;
358         }
359         throw new RuntimeException(Objects.requireNonNull(response));
360     }
361 
362 
363     /**
364      * 确定一个或者多个元素是否在布隆过滤器中存在
365      * 该命令的操作方式`BF.EXISTS`与之相同，只不过它允许多个输入并返回多个值。
366      *
367      * @param filterKey 布隆过滤器的名称
368      * @param items     添加的元素
```

```
     * @return {@link List}<{@link Boolean}>
     */
    public List<Boolean> mExists(String filterKey, String... items)
    {
        if (filterKey == null)
        {
            return null;
        }

        String[] args = new String[items.length + 2];
        args[0] = "BF.MEXISTS";
        args[1] = filterKey;
        System.arraycopy(items, 0, args, 2, items.length);
        sendRequest(args);
        String r = getResponse().toString();
        r = r.substring(1, r.length() - 1);
        String[] split = r.split(", ");
        List<Boolean> list = new ArrayList<>(items.length);
        for (String s : split)
        {
            if (Objects.equals(s, "1"))
            {
                list.add(true);
                continue;
            }
            if (Objects.equals(s, "0"))
            {
                list.add(false);
                continue;
            }
            throw new RuntimeException(Objects.requireNonNull(s));
        }
        return list;
    }


    /**
     * 储备
     *
     * @param filterKey  布隆过滤器的名称
     * @param error_rate 期望的误报率。该值必须介于 0 到 1 之间。例如，对于期望的误报率 0.1%（1000 中为 1），
     *                   error_rate 应该设置为 0.001。
     *                   该数字越接近零，则每个项目的内存消耗越大，并且每个操作的 CPU 使用率越高。
     * @param capacity   过滤器的容量。当实际存储的元素个数超过这个值之后，性能将开始下降。
     *                   实际的降级将取决于超出限制的程度。随着过滤器元素数量呈指数增长，性能将线性下降。
     * @return boolean
     */
    public boolean reserve(String filterKey, float error_rate, int capacity)
    {
        if (error_rate > 1 || error_rate < 0)
```

```
        {
            throw new RuntimeException("期望的误报率应该在0到1之间");
        }
        if (capacity <= 0)
        {
            throw new RuntimeException("过滤器的容量必须要大于0");
        }
        //发送命令
        this.sendRequest("BF.RESERVE", filterKey,
String.valueOf(error_rate), String.valueOf(capacity));
        //读取结果
        String response =
Objects.requireNonNull(this.getResponse()).toString();
        if (Objects.equals(response, "OK"))
        {
            return true;
        }
        throw new RuntimeException(Objects.requireNonNull(response));
    }


    public static void main(String[] args)
    {
        RedisBloomFilterImpl bloomFilter = new
RedisBloomFilterImpl("127.0.0.1", 16379);
        bloomFilter.sendRequest("BF.ADD", "filter", "1");
        System.out.println(bloomFilter.getResponse());
        bloomFilter.sendRequest("BF.MADD", "filter", "1", "2", "3");
        String s = bloomFilter.getResponse().toString();
        s = s.substring(1, s.length() - 1);
        System.out.println(s);
        String[] split = s.split(", ");
        for (String s1 : split)
        {
            System.out.println(s1);
        }

        bloomFilter.sendRequest("BF.EXISTS", "filter", "1");
        System.out.println(bloomFilter.getResponse());
    }
}
```

## 单元测试

```
package mao;

import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.Test;

```

```java
import java.util.List;

import static org.junit.jupiter.api.Assertions.*;

/**
 * Project name(项目名称)：布隆过滤器
 * Package(包名): mao
 * Class(测试类名): RedisBloomFilterImplTest
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2023/2/27
 * Time(创建时间)： 22:36
 * Version(版本): 1.0
 * Description(描述)： 测试类
 */

class RedisBloomFilterImplTest
{

    private static RedisBloomFilterImpl redisBloomFilter;

    @BeforeAll
    static void beforeAll()
    {
        redisBloomFilter = new RedisBloomFilterImpl("127.0.0.1", 16379);
    }

    @AfterAll
    static void afterAll()
    {
        redisBloomFilter.close();
    }

    @Test
    void add()
    {
        System.out.println(redisBloomFilter.add("filter3", "1"));
        System.out.println(redisBloomFilter.add("filter3", "2"));
        System.out.println(redisBloomFilter.add("filter3", "3"));
        System.out.println(redisBloomFilter.add("filter3", "4"));
        System.out.println(redisBloomFilter.add("filter3", "5"));

    }

    @Test
    void mAdd()
    {
        List<Boolean> filter4 = redisBloomFilter.mAdd("filter4", "1", "3",
"4", "6");
        System.out.println(filter4);
        List<Boolean> filter44 = redisBloomFilter.mAdd("filter4", "1", "3",
"4", "7");
        System.out.println(filter44);
    }
```

```
60
61      @Test
62      void exists()
63      {
64          System.out.println(redisBloomFilter.exists("filter3", "1"));
65          System.out.println(redisBloomFilter.exists("filter3", "2"));
66          System.out.println(redisBloomFilter.exists("filter3", "4"));
67          System.out.println(redisBloomFilter.exists("filter3", "5"));
68          System.out.println(redisBloomFilter.exists("filter3", "6"));
69          System.out.println(redisBloomFilter.exists("filter3", "9"));
70
71          System.out.println(redisBloomFilter.exists("filter4", "6"));
72          System.out.println(redisBloomFilter.exists("filter4", "7"));
73          System.out.println(redisBloomFilter.exists("filter4", "9"));
74      }
75
76      @Test
77      void mExists()
78      {
79          List<Boolean> filter3 = redisBloomFilter.mExists
80                  ("filter3", "1", "2", "3", "4", "5", "6", "7", "8");
81          System.out.println(filter3);
82
83          List<Boolean> filter4 = redisBloomFilter.mExists(
84                  "filter4", "1", "2", "3", "4", "5", "6", "7", "8", "9", "10"
85          );
86          System.out.println(filter4);
87      }
88
89
90      @Test
91      void reserve()
92      {
93          System.out.println(redisBloomFilter.reserve("filter5", 0.002f,
    2000));
94          System.out.println(redisBloomFilter.reserve("filter6", 0.002f,
    2000));
95      }
96  }
```

## 测试误差

```
1  package mao;
2
3  import com.google.common.hash.BloomFilter;
4  import com.google.common.hash.Funnels;
5
6  /**
7   * Project name(项目名称)：布隆过滤器
8   * Package(包名): mao
9   * Class(类名): RedisBloomFilterTest
```

```
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2023/2/27
 * Time(创建时间)： 23:13
 * Version(版本): 1.0
 * Description(描述)： 测试redis的布隆过滤器，因为有网络io，所以很慢
 */

public class RedisBloomFilterTest
{
    public static void main(String[] args)
    {
        //redis的布隆过滤器
        RedisBloomFilter redisBloomFilter = new
RedisBloomFilterImpl("127.0.0.1", 16379);

        //1500次循环
        for (int i = 0; i < 1500; i++)
        {
            if (i % 3 == 0)
            {
                continue;
            }
            //将i的值% 3 不等于 0 的值放进去
            redisBloomFilter.add("filter11", String.valueOf(i));
        }

        //存在的计数
        int count = 0;
        //统计
        for (int i = 0; i < 1500; i++)
        {
            //判断是否存在
            boolean b = redisBloomFilter.exists("filter11",
String.valueOf(i));
            //System.out.println(i + " --> " + b);
            //可能存在
            if (b)
            {
                count++;
            }
        }

        System.out.println("预期结果：1000，最终结果：" + count);
        System.out.println();

        redisBloomFilter.close();
    }
}
```

end

by mao
2023 02 27