22 优秀的基数统计算法——HyperLogLog

为什么要使用 HyperLogLog?

在我们实际开发的过程中，可能会遇到这样一个问题，当我们需要统计一个大型网站的独立访问次数时，该用什么的类型来统计?

如果我们使用 Redis 中的集合来统计，当它每天有数千万级别的访问时，将会是一个巨大的问题。因为这些访问量不能被清空，我们运营人员可能会随时查看这些信息，那么随着时间的推移，这些统计数据所占用的空间会越来越大，逐渐超出我们能承载最大空间。

例如，我们用 IP 来作为独立访问的判断依据，那么我们就要把每个独立 IP 进行存储，以 IP4 来计算，IP4 最多需要 15 个字节来存储信息，例如：110.110.110.110。当有一千万个独立 IP 时，所占用的空间就是 15 bit*10000000 约定于 143MB，但这只是一个页面的统计信息，假如我们有 1 万个这样的页面，那我们就需要 1T 以上的空间来存储这些数据，而且随着 IP6 的普及，这个存储数字会越来越大，那我们就不能用集合的方式来存储了，这个时候我们需要开发新的数据类型 HyperLogLog 来做这件事了。

HyperLogLog 介绍

HyperLogLog（下文简称为 HLL）是 Redis 2.8.9 版本添加的数据结构，它用于高性能的基数（去重）统计功能，它的缺点就是存在极低的误差率。

HLL 具有以下几个特点：

- 能够使用极少的内存来统计巨量的数据，它只需要 12K 空间就能统计 2^64 的数据；
- 统计存在一定的误差，误差率整体较低，标准误差为 0.81%；
- 误差可以被设置辅助计算因子进行降低。

基础使用

HLL 的命令只有 3 个，但都非常的实用，下面分别来看。

添加元素

```
127.0.0.1:6379> pfadd key "redis"
(integer) 1
127.0.0.1:6379> pfadd key "java" "sql"
(integer) 1
```

相关语法：

```
pfadd key element [element ...]
```

此命令支持添加一个或多个元素至 HLL 结构中。

统计不重复的元素

```
127.0.0.1:6379> pfadd key "redis"
(integer) 1
127.0.0.1:6379> pfadd key "sql"
(integer) 1
127.0.0.1:6379> pfadd key "redis"
(integer) 0
127.0.0.1:6379> pfcount key
(integer) 2
```

从 pfcount 的结果可以看出，在 HLL 结构中键值为 key 的元素，有 2 个不重复的值：redis 和 sql，可以看出结果还是挺准的。

相关语法：

```
pfcount key [key ...]
```

此命令支持统计一个或多个 HLL 结构。

合并一个或多个 HLL 至新结构

新增 k 和 k2 合并至新结构 k3 中，代码如下：

```
127.0.0.1:6379> pfadd k "java" "sql"
(integer) 1
```

```
127.0.0.1:6379> pfadd k2 "redis" "sql"
(integer) 1
127.0.0.1:6379> pfmerge k3 k k2
OK
127.0.0.1:6379> pfcount k3
(integer) 3
```

相关语法：

```
pfmerge destkey sourcekey [sourcekey ...]
```

pfmerge 使用场景

当我们需要合并两个或多个同类页面的访问数据时，我们可以使用 pfmerge 来操作。

代码实战

接下来我们使用 Java 代码来实现 HLL 的三个基础功能，代码如下：

```
import redis.clients.jedis.Jedis;

public class HyperLogLogExample {
    public static void main(String[] args) {
        Jedis jedis = new Jedis("127.0.0.1", 6379);
        // 添加元素
        jedis.pfadd("k", "redis", "sql");
        jedis.pfadd("k", "redis");
        // 统计元素
        long count = jedis.pfcount("k");
        // 打印统计元素
        System.out.println("k: " + count);
        // 合并 HLL
        jedis.pfmerge("k2", "k");
        // 打印新 HLL
        System.out.println("k2: " + jedis.pfcount("k2"));
    }
}
```

以上代码执行结果如下：

```
k: 2
k2: 2
```

HLL 算法原理

HyperLogLog 算法来源于论文 *HyperLogLog the analysis of a near-optimal cardinality estimation algorithm*，想要了解 HLL 的原理，先要从伯努利试验说起，伯努利实验说的是抛硬币的事。一次伯努利实验相当于抛硬币，不管抛多少次只要出现一个正面，就称为一次伯努利实验。

我们用 k 来表示每次抛硬币的次数，n 表示第几次抛的硬币，用 k_max 来表示抛硬币的最高次数，最终根据估算发现 n 和 k_max 存在的关系是 n=2^(k_max)，但同时我们也发现了另一个问题当试验次数很小的时候，这种估算方法的误差会很大，例如我们进行以下 3 次实验：

- 第 1 次试验：抛 3 次出现正面，此时 k=3，n=1；
- 第 2 次试验：抛 2 次出现正面，此时 k=2，n=2；
- 第 3 次试验：抛 6 次出现正面，此时 k=6，n=3。

对于这三组实验来说，k_max=6，n=3，但放入估算公式明显 3≠2^6。为了解决这个问题 HLL 引入了分桶算法和调和平均数来使这个算法更接近真实情况。

分桶算法是指把原来的数据平均分为 m 份，在每段中求平均数在乘以 m，以此来消减因偶然性带来的误差，提高预估的准确性，简单来说就是把一份数据分为多份，把一轮计算，分为多轮计算。

而调和平均数指的是使用平均数的优化算法，而非直接使用平均数。

> 例如小明的月工资是 1000 元，而小王的月工资是 100000 元，如果直接取平均数，那小明的平均工资就变成了 (1000+100000)/2=50500 元，这显然是不准确的，而使用调和平均数算法计算的结果是 2/(1/1000+1/100000)≈1998 元，显然此算法更符合实际平均数。

所以综合以上情况，在 Redis 中使用 HLL 插入数据，相当于把存储的值经过 hash 之后，再将 hash 值转换为二进制，存入到不同的桶中，这样就可以用很小的空间存储很多的数据，统计时再去相应的位置进行对比很快就能得出结论，这就是 HLL 算法的基本原理，想要更深入的了解算法及其推理过程，可以看去原版的论文，链接地址在文末。

小结

当需要做大量数据统计时，普通的集合类型已经不能满足我们的需求了，这个时候我们可以借助 Redis 2.8.9 中提供的 HyperLogLog 来统计，它的优点是只需要使用 12k 的空间就能

统计 2^64 的数据，但它的缺点是存在 0.81% 的误差，HyperLogLog 提供了三个操作方法 pfadd 添加元素、pfcount 统计元素和 pfmerge 合并元素。

参考文献

- 论文 *HyperLogLog: the analysis of a near-optimal cardinality estimation algorithm*

上一页 下一页