21 游标迭代器（过滤器）——Scan

一个问题引发的「血案」

曾经发生过这样一件事，我们的 Redis 服务器存储了海量的数据，其中登录用户信息是以 user_token_id 的形式存储的。运营人员想要当前所有的用户登录信息，然后悲剧就发生了：因为我们的工程师使用了 `keys user_token_*` 来查询对应的用户，结果导致 Redis 假死不可用，以至于影响到线上的其他业务接连发生问题，然后就收到了一堆的系统预警短信。并且这个假死的时间是和存储的数据成正比的，数据量越大假死的时间就越长，导致的故障时间也越长。

那如何避免这个问题呢？

问题的解决方案

在 Redis 2.8 之前，我们只能使用 keys 命令来查询我们想要的数据，但这个命令存在两个缺点：

1. 此命令没有分页功能，我们只能一次性查询出所有符合条件的 key 值，如果查询结果非常巨大，那么得到的输出信息也会非常多；
2. keys 命令是遍历查询，因此它的查询时间复杂度是 o(n)，所以数据量越大查询时间就越长。

然而，比较幸运的是在 Redis 2.8 时推出了 Scan，解决了我们这些问题，下面来看 Scan 的具体使用。

Scan 命令使用

我们先来模拟海量数据，使用 Pipeline 添加 10w 条数据，Java 代码实现如下：

```
import redis.clients.jedis.Jedis;
import redis.clients.jedis.Pipeline;
import utils.JedisUtils;
```

```
public class ScanExample {
    public static void main(String[] args) {
        // 添加 10w 条数据
        initData();
    }
    public static void initData(){
        Jedis jedis = JedisUtils.getJedis();
        Pipeline pipe = jedis.pipelined();
        for (int i = 1; i < 100001; i++) {
            pipe.set("user_token_" + i, "id" + i);
        }
        // 执行命令
        pipe.sync();
        System.out.println("数据插入完成");
    }
}
```

我们来查询用户 id 为 9999* 的数据，Scan 命令使用如下：

```
127.0.0.1:6379> scan 0 match user_token_9999* count 10000
1) "127064"
2) 1) "user_token_99997"
127.0.0.1:6379> scan 127064 match user_token_9999* count 10000
1) "1740"
2) 1) "user_token_9999"
127.0.0.1:6379> scan 1740 match user_token_9999* count 10000
1) "21298"
2) 1) "user_token_99996"
127.0.0.1:6379> scan 21298 match user_token_9999* count 10000
1) "65382"
2) (empty list or set)
127.0.0.1:6379> scan 65382 match user_token_9999* count 10000
1) "78081"
2) 1) "user_token_99998"
   2) "user_token_99992"
127.0.0.1:6379> scan 78081 match user_token_9999* count 10000
1) "3993"
2) 1) "user_token_99994"
   2) "user_token_99993"
127.0.0.1:6379> scan 3993 match user_token_9999* count 10000
1) "13773"
2) 1) "user_token_99995"
127.0.0.1:6379> scan 13773 match user_token_9999* count 10000
1) "47923"
2) (empty list or set)
127.0.0.1:6379> scan 47923 match user_token_9999* count 10000
1) "59751"
2) 1) "user_token_99990"
   2) "user_token_99991"
   3) "user_token_99999"
127.0.0.1:6379> scan 59751 match user_token_9999* count 10000
1) "0"
2) (empty list or set)
```

从以上的执行结果，我们看出两个问题：

1. 查询的结果为空，但游标值不为 0，表示遍历还没结束；
2. 设置的是 count 10000，但每次返回的数量都不是 10000，且不固定，这是因为 count 只是限定服务器单次遍历的字典槽位数量（约等于），而不是规定返回结果的 count 值。

相关语法：

```
scan cursor [MATCH pattern] [COUNT count]
```

其中：

- cursor：光标位置，整数值，从 0 开始，到 0 结束，查询结果是空，但游标值不为 0，表示遍历还没结束；
- match pattern：正则匹配字段；
- count：限定服务器单次遍历的字典槽位数量（约等于），只是对增量式迭代命令的一种提示（hint），并不是查询结果返回的最大数量，它的默认值是 10。

代码实战

本文我们使用 Java 代码来实现 Scan 的查询功能，代码如下：

```
import redis.clients.jedis.Jedis;
import redis.clients.jedis.Pipeline;
import redis.clients.jedis.ScanParams;
import redis.clients.jedis.ScanResult;
import utils.JedisUtils;

public class ScanExample {
    public static void main(String[] args) {
        Jedis jedis = JedisUtils.getJedis();
        // 定义 match 和 count 参数
        ScanParams params = new ScanParams();
        params.count(10000);
        params.match("user_token_9999*");
        // 游标
        String cursor = "0";
        while (true) {
            ScanResult<String> res = jedis.scan(cursor, params);
            if (res.getCursor().equals("0")) {
```

```
                // 表示最后一条
                break;
            }
            cursor = res.getCursor(); // 设置游标
            for (String item : res.getResult()) {
                // 打印查询结果
                System.out.println("查询结果：" + item);
            }
        }
    }
}
```

以上程序执行结果如下：

```
查询结果：user_token_99997
查询结果：user_token_9999
查询结果：user_token_99996
查询结果：user_token_99998
查询结果：user_token_99992
查询结果：user_token_99994
查询结果：user_token_99993
查询结果：user_token_99995
查询结果：user_token_99990
查询结果：user_token_99991
查询结果：user_token_99999
```

Scan 相关命令

Scan 是一个系列指令，除了 Scan 之外，还有以下 3 个命令：

1. HScan 遍历字典游标迭代器
2. SScan 遍历集合的游标迭代器
3. ZScan 遍历有序集合的游标迭代器

来看这些命令的具体使用。

HScan 使用

```
127.0.0.1:6379> hscan myhash 0 match k2* count 10
1) "0"
2) 1) "k2"
   2) "v2"
```

相关语法：

```
hscan key cursor [MATCH pattern] [COUNT count]
```

SScan 使用

```
127.0.0.1:6379> sscan myset 0 match v2* count 20
1) "0"
2) 1) "v2"
```

相关语法：

```
sscan key cursor [MATCH pattern] [COUNT count]
```

ZScan 使用

```
127.0.0.1:6379> zscan zset 0 match red* count 20
1) "0"
2) 1) "redis"
   2) "10"
```

相关语法：

```
zscan key cursor [MATCH pattern] [COUNT count]
```

Scan 说明

官方对 Scan 命令的描述信息如下。

Scan guarantees

> The SCAN command, and the other commands in the SCAN family, are able to provide to the user a set of guarantees associated to full iterations.
>
> - A full iteration always retrieves all the elements that were present in the

> collection from the start to the end of a full iteration. This means that if a given element is inside the collection when an iteration is started, and is still there when an iteration terminates, then at some point SCANreturned it to the user.
>
> - A full iteration never returns any element that was NOT present in the collection from the start to the end of a full iteration. So if an element was removed before the start of an iteration, and is never added back to the collection for all the time an iteration lasts, SCAN ensures that this element will never be returned.
>
> However because SCAN has very little state associated (just the cursor) it has the following drawbacks:
>
> - A given element may be returned multiple times. It is up to the application to handle the case of duplicated elements, for example only using the returned elements in order to perform operations that are safe when re-applied multiple times.
> - Elements that were not constantly present in the collection during a full iteration, may be returned or not: it is undefined.

官方文档地址：

> https://redis.io/commands/scan

翻译为中文的含义是：Scan 及它的相关命令可以保证以下查询规则。

- 它可以完整返回开始到结束检索集合中出现的所有元素，也就是在整个查询过程中如果这些元素没有被删除，且符合检索条件，则一定会被查询出来；
- 它可以保证不会查询出，在开始检索之前删除的那些元素。

然后，Scan 命令包含以下缺点：

- 一个元素可能被返回多次，需要客户端来实现去重；
- 在迭代过程中如果有元素被修改，那么修改的元素能不能被遍历到不确定。

小结

通过本文我们可以知道 Scan 包含以下四个指令：

1. Scan：用于检索当前数据库中所有数据；
2. HScan：用于检索哈希类型的数据；

3. SScan：用于检索集合类型中的数据；
4. ZScan：由于检索有序集合中的数据。

Scan 具备以下几个特点：

1. Scan 可以实现 keys 的匹配功能；
2. Scan 是通过游标进行查询的不会导致 Redis 假死；
3. Scan 提供了 count 参数，可以规定遍历的数量；
4. Scan 会把游标返回给客户端，用户客户端继续遍历查询；
5. Scan 返回的结果可能会有重复数据，需要客户端去重；
6. 单次返回空值且游标不为 0，说明遍历还没结束；
7. Scan 可以保证在开始检索之前，被删除的元素一定不会被查询出来；
8. 在迭代过程中如果有元素被修改， Scan 不保证能查询出相关的元素。

上一页 下一页