Redis使用场景与注意事项

> 本文档主要描述了redis的一些典型使用场景，以及注意事项

redis基础配置

硬件配置

> 硬件配置 2c16g
>
> container 1c8g

代码中使用

> 目前系统中默认的redis客户端为redisson。redisson启动默认使用了连接池，24个连接，大部分情况够用了

redisson配置文件

> 默认沿用spring的redis配置，扩展了几个redisson独有的配置点，可以在spring yml配置文件中直接配置

```
#
spring.redis.redisson.connectionPoolSize: 256
spring.redis.redisson.connectionMinimumIdleSize: 24
spring.redis.redisson.pingConnectionInterval: 60000  #
spring.redis.timeout: 3000 #
spring.redis.redisson.retryAttempts: 3 #
```

RedisionClient

```
    @Autowired
private RedissonClient redissonClient;
```

RedisTemplate

```
    @Autowired
private StringRedisTemplate redisTemplate;
```

redision与RedisTemplate

父工程中已添加redision依赖。可以在代码中通过注入RedissonClient使用。

Redisson可以直接存储java对象，使用起来较为方便。并支持多种数据格式。

如有特殊的场景，需要使用原生redis语义，则也可直接注入RedisTemplate使用。

典型业务场景

Key-Value缓存

> 用来做数据库查询的缓存。详情查看缓存

分布式计数器

库存控制

同时有多个并发请求进行库存扣减

```
SET inv:remain "100"
DECR inv:remain
```

计数器

同时有多个并发请求进行计数统计

```
redis> SET page_view
20
OK

redis> INCR page_view
(integer) 21
```

取最新N个值

每条微博下列表显示最新的10条回复，其余的回复通过点击“更多”按钮展示。

```
#
redis> LPUSH
(integer) 1


#  LTRIM
redis> LTRIM  0 9
OK
```

分布式锁

```
        RLock lock = redissonClient.getLock(
                ArafStringUtils.isEmpty(distribtedLock.value()) ? pjp.
getSignature().getName()
                        : distribtedLock.value());
        boolean locked = lock.tryLock(0, distribtedLock.expired(),
TimeUnit.SECONDS);
        try {
            if (locked) {
                return pjp.proceed();
            }

        } finally {
            if (locked) {
                lock.unlock();
            }
        }
```

使用key的过期时间

> 提醒用户更新密码。需要提前N天提醒，每天提醒一次。不能重复提醒

```
redis> SET user1:sendAlert  "yes"
OK

redis> EXPIRE user1:sendAlert  86000  #
(integer) 1

redis> EXISTS user1:sendAlert
(integer) 0
```

发布订阅

> 用户选择感兴趣的频道。当有新消息的时候通知用户。
>
> 聊天系统。

```
redis> psubscribe news.* tweet.*
Reading messages... (press Ctrl-C to quit)
1) "psubscribe"                  #
2) "news.*"                      #
3) (integer) 1                   #


#
redis> publish chat_room "hello~ everyone"
(integer) 3
```

对比两个集合，找不同，对账

两个字符串集合，找到集合中相同与不同的数据。

```
redis> SMEMBERS peter's_movies
1) "bet man"
2) "start war"
3) "2012"

redis> SMEMBERS joe's_movies
1) "hi, lady"
2) "Fast Five"
3) "2012"

redis> SDIFF peter's_movies joe's_movies
1) "bet man"
2) "start war"



redis> SMEMBERS group_1
1) "LI LEI"
2) "TOM"
3) "JACK"

redis> SMEMBERS group_2
1) "HAN MEIMEI"
2) "JACK"

redis> SINTER group_1 group_2
1) "JACK"
```

sadd 存储200000次所花费的时间： 873279ms.

sdiff 对比两个大小为100000的set所花费的时间：237ms.

布隆过滤器

Redis中有一个数据结构叫做Bitmap(下方有官网详解)，它提供一个最大长度为512MB（2^32）的位数组。我们可以把它提供给布隆过滤器做位数组。

根据《数学之美》中给出的数据，在使用8个哈希函数的情况下，512MB大小的位数组在误报率万分之五的情况下可以对约两亿的url去重。而若单纯的使用set()去重的话，以一个url64个字节记，两亿url约需要128GB的内存空间,不敢想象。

```
/*
         *  ""
         */
        static List<String> l = new ArrayList<String>();

        // //ID
        static {
                l.add("201810120001");
                l.add("201810120002");
                l.add("201810120003");
                l.add("201810120004");
        }

        /**
         *
         */
        @Test
        public void TestRedis() {
                Jedis jedis = new Jedis("192.168.1.118", 6379);

                // redis
                double size = Math.pow(2, 32);
                // l redis
                l.forEach(orderId -> {
                        long index = Math.abs((long) (orderId.hashCode() %
size));
                        jedis.setbit("orderId", index, true);
                });



                //
                String orderId = l.get(0);
                //String orderId = "201810120005";
                long index =  Math.abs((long) (orderId.hashCode() % size));
                boolean contain = jedis.getbit("orderId", index);
                // true
                if (contain) {
                        System.out.println(".");
                        //
                } else {
                        System.out.println(".");
                }


                jedis.close();
        }
```

Key命名规则

key命名规则：

```
{}:{}:{key}
```

注意事项

- 必须有有效期
- 不能使用key*做模糊匹配key值。
- 需要组内评审，考虑数量，更新频率等等问题。
- 一般由服务提供方使用redis，管理redis数据。获取数据通过服务调用的方式。
- redis中可以直接放java对象。redision会序列化为二进制格式。
- 不要使用过长的Key。例如使用一个1024字节的key就不是一个好主意，不仅会消耗更多的内存，还会导致查找的效率降低
- Key短到缺失了可读性也是不好的，例如"u1000flw"比起"user:1000:followers"来说，节省了寥寥的存储空间，却引发了可读性和可维护性上的麻烦
- 避免操作大集合的慢命令

特殊场景