作者:Wilson

Redis数据类型

Redis支持五种数据类型：string（字符串），hash（哈希），list（列表），set（集合）及zset(sorted set：有序集合)等

List类型

简介

List类型是一个链表结构的集合，其主要功能有push、pop、获取元素等。更详细的说，List类型是一个双端链表的节后，我们可以通过相关的操作进行集合的头部或者尾部添加和删除元素，List的设计非常简单精巧，即可以作为栈，又可以作为队列，满足绝大多数的需求。

> 按照插入顺序排序。你可以添加一个元素到列表的头部（左边）或者尾部（右边）一个列表最多可以包含 232 - 1 个元素 (4294967295, 每个列表超过40亿个元素)类似JAVA中的LinkedList

常用命令

赋值

赋值语法：

```
LPUSH key value1 [value2]   :将一个或多个值插入到列表头部(从左侧添加)
RPUSH key value1 [value2]   :在列表中添加一个或多个值(从右侧添加)
LPUSHX key value    :将一个值插入到已存在的列表头部。如果列表不在，操作无效
RPUSHX key value    :一个值插入已存在的列表尾部(最右边)。如果列表不在，操作无效。
```

取值

取值语法：

```
LLEN key       :获取列表长度
LINDEX key index    :通过索引获取列表中的元素
LRANGE key start stop   :获取列表指定范围内的元素
```

> 描述： 返回列表中指定区间内的元素，区间以偏移量 START 和 END 指定。
>
> 其中 0 表示列表的第一个元素， 1 表示列表的第二个元素，以此类推。

也可以使用负数下标，以 -1 表示列表的最后一个元素， -2 表示列表的倒数第二个元素，以此类推。 start: 页大小 *(页数-1)*
stop : (页大小 页数)-1

```
当前是第1页， 每页显示3条数据
start :0  stop:2
start :3  stop:5
start :6  stop: 8
```

删除

删除语法：

```
    LPOP key  移出并获取列表的第一个元素(从左侧删除)
    RPOP key  移除列表的最后一个元素，返回值为移除的元素(从右侧删除)

    BLPOP key1 [key2 ] timeout 移出并获取列表的第一个元素， 如果列表没有元素会阻塞列表直到等待超时或发现可弹出元素为止。

实例：
redis 127.0.0.1:6379> BLPOP list1 100
在以上实例中，操作会被阻塞，如果指定的列表 key list1 存在数据则会返回第一个元素，否则在等待100秒后会返回 nil

    BRPOP key1 [key2 ] timeout :移出并获取列表的最后一个元素， 如果列表没有元素会阻塞列表直到等待超时或发现可弹出元素为止。

    LTRIM key start stop  :对一个列表进行修剪(trim)，就是说，让列表只保留指定区间内的元素，不在指定区间之内的元素都将被删除。
```

修改

修改语法：

```
    LSET key index value  :通过索引设置列表元素的值
    LINSERT key BEFORE|AFTER world value :在列表的元素前或者后插入元素 描述：将值 value 插入到列表 key 当中，位于值 world 之前或之后。
```

高级命令

高级语法：

```
    RPOPLPUSH source destination :移除列表的最后一个元素，并将该元素添加到另一个列表并返回
    示例描述：
        RPOPLPUSH a1  a2   :a1的最后元素移到a2的左侧
        RPOPLPUSH a1  a1   :循环列表，将最后元素移到最左侧
    BRPOPLPUSH source destination timeout  :从列表中弹出一个值，将弹出的元素插入到另外一个列表中并返回它； 如果列表没有元素会阻塞列表直到等待超时或发现可弹出元素为止。
```

代码示例

如下演示了List类型所有命令的调用示例

```
/**
 * @ClassName ListCacheServiceImpl
 * @Description TODO
 * @Author guoweixin
 * @Version 1.0
 */
@Service("listCacheService")
public class ListCacheServiceImpl implements ListCacheService {
    private final static Logger log =
LoggerFactory.getLogger(ListCacheServiceImpl.class);

    @Autowired
    private RedisTemplate<String, Object> redisTemplate;


    /**
     * 将list放入缓存
     * @param key   键
     * @param value 值
     * @return  true 成功 false 失败
     */
    public boolean lpushAll(String key, List <Object> value) {
        try {
            redisTemplate.opsForList().leftPushAll(key, value);
            return true;
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }

    /**
     * 将list放入缓存
     * @param key   键
     * @param value 值
     * @param time  时间(秒)
     * @return  true 成功 false 失败
     */
    public boolean lpushAll(String key, List <Object> value, long time) {
        try {
            redisTemplate.opsForList().leftPushAll(key, value);
            if (time > 0) {
                expire(key, time);
            }
            return true;
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }

    /**
     * 将list放入缓存
```

```
     * @param key   键
     * @param value 值
     * @return true 成功 false 失败
     */
    public boolean rpushAll(String key, List <Object> value) {
        try {
            redisTemplate.opsForList().rightPushAll(key, value);
            return true;
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }

    /**
     * 将list放入缓存
     * @param key   键
     * @param value 值
     * @param time  时间(秒)
     * @return true 成功 false 失败
     */
    public boolean rpushAll(String key, List <Object> value, long time) {
        try {
            redisTemplate.opsForList().rightPushAll(key, value);
            if (time > 0)
                expire(key, time);
            return true;
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }

    /**
     * 在变量左边添加元素值。
     * @param key 键
     * @param object 值
     * @return  true 成功 false 失败
     */
    @Override
    public Boolean lpush(String key, Object object) {
        try {
            redisTemplate.opsForList().leftPush(key, object);
            return true;
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }

    /**
     * 把最后一个参数值放到指定集合的第一个出现中间参数的前面，如果中间参数值存在的话。
     * @param key 键
     * @param pivot 中间参数
     * @param object 要放的值
     * @return 成功 true 失败 false
     */
    @Override
```

```
    public Boolean lpush(String key, Object pivot, Object object) {
        try {
            redisTemplate.opsForList().leftPush(key,pivot,object);
            return true;
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }

    /**
     * 集合中第一次出现第二个参数变量元素的右边添加第三个参数变量的元素值。
     * @param key 键
     * @param pivot 中间参数
     * @param object 要放的值
     * @return 成功 true 失败 false
     */
    @Override
    public Boolean rpush(String key, Object pivot, Object object) {
        try {
            redisTemplate.opsForList().rightPush(key,pivot,object);
            return true;
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }

    /**
     * 向集合最右边添加元素。
     * @param key 键
     * @param object 值
     * @return 成功 true 失败 false
     */
    @Override
    public Boolean rpush(String key, Object object) {
        try {
            redisTemplate.opsForList().rightPush(key, object);
            return true;
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }

    /**
     *  在变量左边添加元素值。
     * @param key 键
     * @param expireTime 超时时间
     * @param objects 值
     * @return 成功 true 失败 false
     */
    @Override
    public Boolean lpush(String key, int expireTime, Object... objects) {
        try {
            redisTemplate.opsForList().leftPush(key,objects);
            if (expireTime > 0) {
                expire(key, expireTime);
```

```
            }
            return true;
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }

    /**
     *  在变量右边添加元素值。
     * @param key 键
     * @param expireTime 超时时间
     * @param objects 值
     * @return 成功 true 失败 false
     */
    @Override
    public Boolean rpush(String key, int expireTime, Object... objects) {
        try {
            redisTemplate.opsForList().rightPush(key,objects);
            if (expireTime > 0) {
                expire(key, expireTime);
            }
            return true;
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }

    /**
     * 如果存在集合则向左边添加元素，不存在不加
     * @param key 键
     * @param object 值
     * @return 成功 true 失败 false
     */
    public boolean lPushIfPresent(String key, Object object){
        try {
            redisTemplate.opsForList().leftPushIfPresent(key,object);
            return true;
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }

    /**
     * 如果存在集合则向右边添加元素，不存在不加
     * @param key  键
     * @param object 返回
     * @return 成功 true 失败 false
     */
    public boolean rPushIfPresent(String key, Object object){
        try {
            redisTemplate.opsForList().rightPushIfPresent(key,object);
            return true;
        } catch (Exception e) {
            e.printStackTrace();
            return false;
```

```
        }
    }

    /**
     * 移除集合中的左边第一个元素
     * @param key 键
     * @return 返回右边的第一个元素
     */
    @Override
    public Object lpop(String key) {
        return redisTemplate.opsForList().leftPop(key);
    }

    /**
     * 移除集合中右边的元素。一般用在队列取值
     * @param key 键
     * @return 返回右边的元素
     */
    @Override
    public Object rpop(String key) {
        return redisTemplate.opsForList().rightPop(key);
    }

    /**
     * 移除集合中左边的元素在等待的时间里，如果超过等待的时间仍没有元素则退出。一般用在队列取值
     * @param key 键
     * @param time 时间
     * @return 左边的元素
     */
    @Override
    public Object lpop(String key,long time) {
        return redisTemplate.opsForList().leftPop(key,time,TimeUnit.MILLISECONDS);
    }

    /**
     * 移除集合中右边的元素在等待的时间里，如果超过等待的时间仍没有元素则退出。一般用在队列取值
     * @param key 键
     * @param time 时间
     * @return 返回右边元素
     */
    @Override
    public Object rpop(String key,long time) {
        return redisTemplate.opsForList().rightPop(key,time,TimeUnit.MILLISECONDS);
    }

    /**
     *获取指定区间的值。
     * @param key 键
     * @param start 开始位置
     * @param end   结束位置，为-1指结尾的位置， start 0，end -1取所有
     * @return
     */
    @Override
    public List<Object> lrange(String key, long start, long end) {
        return redisTemplate.opsForList().range(key,start,end);
    }

    /**
```

```
     * 获取集合长度
     * @param key 键
     * @return 返回长度
     */
    @Override
    public Long llen(String key) {
        return redisTemplate.opsForList().size(key);
    }

    /**
     * 在集合的指定位置插入元素,如果指定位置已有元素，则覆盖，没有则新增，超过集合下标+n则会报错。
     * @param key 键
     * @param index 位置
     * @param value 值
     */
    @Override
    public void set(String key, Long index, Object value) {
        redisTemplate.opsForList().set(key,index,value);
    }

    /**
     * 获取集合指定位置的值
     * @param key 键
     * @param index 位置
     * @return 返回值
     */
    @Override
    public Object lindex(String key, Long index) {
        return redisTemplate.opsForList().index(key,index);
    }

    /**
     *  从存储在键中的列表中删除等于值的元素的第一个计数事件。count> 0：
     *  删除等于从左到右移动的值的第一个元素；count< 0：删除等于从右到左移动的值的第一个元素；count
= 0：删除等于value的所有元素。
     * @param key 键
     * @param count
     * @param object
     * @return
     */
    @Override
    public long remove(String key,long count,Object object) {
       return  redisTemplate.opsForList().remove(key, count ,object);
    }

    /**
     * // 截取集合元素长度，保留长度内的数据。
     * @param key 键
     * @param start 开始位置
     * @param end   结束位置
     */
    @Override
    public void trim(String key,long start,long end) {
        redisTemplate.opsForList().trim(key, start, end);
    }

    /**
     * 除集合中右边的元素，同时在左边加入一个元素。
```

```
     * @param key 键
     * @param str 加入的元素
     * @return 返回右边的元素
     */
    @Override
    public Object rightPopAndLeftPush(String key,String str){
        return redisTemplate.opsForList().rightPopAndLeftPush(key,str);
    }

    /**
     * 移除集合中右边的元素在等待的时间里，同时在左边添加元素，如果超过等待的时间仍没有元素则退出。
     * @param key 键
     * @param str 左边增中的值
     * @param timeout 超时时间
     * @return 返回移除右边的元素
     */
    @Override
    public Object rightPopAndLeftPush(String key,String str, long timeout){
        return
redisTemplate.opsForList().rightPopAndLeftPush(key,str,timeout,TimeUnit.MILLISECONDS);
    }

    /**
     * 删除
     * @param keys 键
     */
    @Override
    public void del(String... keys) {
        if (keys != null && keys.length > 0) {
            if (keys.length == 1) {
                redisTemplate.delete(keys[0]);
            } else {
                redisTemplate.delete(CollectionUtils.arrayToList(keys));
            }
        }
    }

    /**
     * 设置过期时间
     * @param key 键
     * @param seconds 超时时间
     * @return 成功 true 失败 false
     */
    @Override
    public boolean expire(String key, long seconds) {
        return redisTemplate.expire(key,seconds,TimeUnit.SECONDS);
    }
}
```


应用场景

项目常应用于：1、对数据量大的集合数据删减 2、任务队列

1、对数据量大的集合数据删减 列表数据显示、关注列表、粉丝列表、留言评价等...分页、热点新闻（Top5)等 利用LRANGE还可以很方便的实现分页的功能,在博客系统中，每片博文的评论也可以存入一个单独的list中。

2、任务队列 (list通常用来实现一个消息队列，而且可以确保先后顺序，不必像MySQL那样还需要通过ORDER BY来进行排序)

```
任务队列介绍(生产者和消费者模式)：
    在处理Web客户端发送的命令请求时，某些操作的执行时间可能会比我们预期的更长一些，通过将待执行任务的相关信息放入队列里面，并在之后对队列进行处理，用户可以推迟执行那些需要一段时间才能能完成的操作，这种将工作交给任务处理器来执行的做法被称为任务队列（task queue）。

RPOPLPUSH source destination
移除列表的最后一个元素，并将该元素添加到另一个列表并返回
```

代码案例

案例1 热点新闻列表

```
比如:获取最新5条首页新闻,获取最新的评论列表，获取最后登录10个用户，获取最近7天的活跃用户数等或做为队列来使用。
需求：  获取最新5条首页新闻。
```

案例2 任务队列(商城)

```
1:用户系统登录注册短信发送认证等
2:订单系统的下单流程等
```

Set类型

简介

Redis 的 Set 是 String 类型的无序集合。集合成员是唯一的，这就意味着集合中不能出现重复的数据。Redis 中集合是通过哈希表实现的，set是通过hashtable实现的 集合中最大的成员数为 2次方32 - 1 (4294967295, 每个集合可存储40多亿个成员)。 类似于JAVA中的 Hashtable集合

命令

```
赋值语法：
    SADD key member1 [member2] :向集合添加一个或多个成员

取值语法：
    SCARD key :获取集合的成员数
    SMEMBERS key  :返回集合中的所有成员
    SISMEMBER key member  :判断 member 元素是否是集合 key 的成员(开发中：验证是否存在判断）
```

```
    SRANDMEMBER key [count] :返回集合中一个或多个随机数

删除语法:
    SREM key member1 [member2] :移除集合中一个或多个成员
    SPOP key [count]  :移除并返回集合中的一个随机元素
    SMOVE source destination member :将 member 元素从 source 集合移动到 destination 集合

差集语法:
    SDIFF key1  [key2]   :返回给定所有集合的差集(左侧)
    SDIFFSTORE destination key1 [key2]  :返回给定所有集合的差集并存储在 destination 中
交集语法:
    SINTER key1 [key2]  :返回给定所有集合的交集(共有数据)
    SINTERSTORE destination key1 [key2]  :返回给定所有集合的交集并存储在 destination 中
并集语法:
    SUNION key1 [key2] :返回所有给定集合的并集
    SUNIONSTORE destination key1 [key2]  :所有给定集合的并集存储在 destination 集合中
```

代码示例

如下演示了Set类型所有命令的调用示例

```
/**
 * @ClassName SetcacheServiceImpl
 * @Description TODO
 * @Author guoweixin
 * @Version 1.0
 */
@Service("setCacheService")
public class SetCacheServiceImpl implements SetCacheService {
    private final static Logger log =
LoggerFactory.getLogger(SetCacheServiceImpl.class);

    @Autowired
    private RedisTemplate<String, Object> redisTemplate;

    /**
     *  向变量中批量添加值。
     * @param key 键
     * @param objects 值
     * @return true成功 false失败
     */
    public boolean add(String key, Object...objects){
        try {
            redisTemplate.opsForSet().add(key,objects);
            return true;
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }

    /**
     *  向变量中批量添加值。
     * @param key 键
```

```
     * @param expireTime 值
     * @param values 值
     * @return true成功 false失败
     */
    @Override
    public Boolean add(String key, int expireTime, Object... values) {
        try {
            redisTemplate.opsForSet().add(key,values);
            if (expireTime > 0)
                expire(key, expireTime);
            return true;
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }

    /**
     * members(K key)获取变量中的值。
     * @param key 键
     * @return 返回Set对象
     */
    public Set<Object> members(String key) {
        return redisTemplate.opsForSet().members(key);
    }

    /**
     * 获取变量中值的长度。
     * @param key 键
     * @return 返回SET的长度
     */
    @Override
    public long size(String key) {
        return redisTemplate.opsForSet().size(key);
    }

    /**
     * 检查给定的元素是否在变量中。
     * @param key 键
     * @param o 要检查的变量
     * @return true存在 false不存在
     */
    @Override
    public boolean isMember(String key, Object o) {
        return redisTemplate.opsForSet().isMember(key,o);
    }

    /**
     * 转移变量的元素值到目的变量。
     * @param key 键
     * @param value 要转移的元素
     * @param destValue 目标键
     * @return true 成功 false 失败
     */
    @Override
    public boolean move(String key, Object value, String destValue) {
        return redisTemplate.opsForSet().move(key,value,destValue);
    }
```

```
/**
 * 弹出变量中的元素。
 * @param key 键
 * @return 返回弹出的元素
 */
@Override
public Object pop(String key) {
    return redisTemplate.opsForSet().pop(key);
}

/**
 * 批量移除变量中的元素。
 * @param key 键
 * @param values 要移除的元素
 * @return 返回移除元素个数
 */
@Override
public long remove(String key, Object... values) {
    return redisTemplate.opsForSet().remove(key,values);
}

/**
 * 匹配获取键值对
 * @param key 键
 * @param options 选项
 * @return 返回键值对
 */
@Override
public Cursor<Object> scan(String key, ScanOptions options) {
    return redisTemplate.opsForSet().scan(key,options);
}

/**
 * 通过集合求差值。
 * @param key 键
 * @param list LIST中的对象是要比较缓存的KEY
 * @return 返回差差值
 */
@Override
public Set<Object> difference(String key, List list) {
    return redisTemplate.opsForSet().difference(key,list);
}

@Override
public Set<Object> difference(String key, String otherKeys) {
    return  redisTemplate.opsForSet().difference(key,otherKeys);
}

/**
 * 将求出来的差值元素保存。
 * @param key 键
 * @param otherKey 要比较的缓存键
 * @param destKey 要保存差值的缓存键
 */
@Override
public void differenceAndStore(String key, String otherKey, String destKey) {
    redisTemplate.opsForSet().differenceAndStore(key,otherKey,destKey);
```

```
    }

    /**
     * 将求出来的差值元素保存。
     * @param key 键
     * @param otherKeys 要比较的多个缓存键
     * @param destKey 要保存差值的缓存键
     */
    @Override
    public void differenceAndStore(String key, List otherKeys, String destKey) {
        redisTemplate.opsForSet().differenceAndStore(key,otherKeys,destKey);
    }

    /**
     * 获取去重的随机元素。
     * @param key 键
     * @param count 数量
     * @return 返回随机元素
     */
    @Override
    public Set<Object> distinctRandomMembers(String key, long count) {
        return redisTemplate.opsForSet().distinctRandomMembers(key,count);
    }

    /**
     * 获取2个变量中的交集。
     * @param key 键
     * @param otherKey 比较的缓存键
     * @return 返回交集
     */
    @Override
    public Set<Object> intersect(String key, String otherKey) {
        return redisTemplate.opsForSet().intersect(key,otherKey);
    }

    @Override
    public Set<Object> intersect(String key, List list) {
        return redisTemplate.opsForSet().intersect(key,list);
    }

    /**
     * 获取2个变量交集后保存到最后一个参数上
     * @param key 键
     * @param otherKey 其它的缓存键
     * @param destKey 交集键
     */
    @Override
    public void intersectAndStore(String key, String otherKey, String destKey) {
        redisTemplate.opsForSet().intersectAndStore(key, otherKey, destKey);
    }


    /**
     * 获取2个变量交集后保存到最后一个参数上
     * @param key 键
     * @param otherKey 其它的缓存键列表
     * @param destKey 交集键
     */
```

```
    @Override
    public void intersectAndStore(String key, List otherKey, String destKey) {
        redisTemplate.opsForSet().intersectAndStore(key, otherKey, destKey);
    }

    /**
     * 获取2个变量的合集。
     * @param key 键
     * @param otherKey 要合的键
     * @return 返回合并后的SET
     */
    @Override
    public Set<Object> union(String key, String otherKey) {
        return redisTemplate.opsForSet().union(key,otherKey);
    }

    @Override
    public Set<Object> union(String key, Set set) {
        return redisTemplate.opsForSet().union(key,set);
    }

    /**
     * 获取2个变量合集后保存到最后一个参数上。
     * @param key 键
     * @param otherKey 要合的键
     * @param destKey 合并后的键
     */
    @Override
    public void unionAndStore(String key, String otherKey, String destKey) {
        redisTemplate.opsForSet().unionAndStore(key, otherKey, destKey);
    }

    /**获取2个变量合集后保存到最后一个参数上。
     *
     * @param key 键
     * @param list 要合的键列表
     * @param destKey 合并后的键
     */
    @Override
    public void unionAndStore(String key, List list, String destKey) {
        redisTemplate.opsForSet().unionAndStore(key, list, destKey);
    }

    /**
     *  随机获取变量中的元素。
     * @param key 键
     * @return 返回其中一个随机元素
     */
    @Override
    public Object randomMember(String key){
        return redisTemplate.opsForSet().randomMember(key);
    }

    /**
     * 随机获取变量中指定个数的元素
     * @param key 键
     * @param count 取随机数的个数
     * @return 返回随机数LIST
```

```
     */
    @Override
    public List<Object> randomMembers(String key, long count){
        return redisTemplate.opsForSet().randomMembers(key,count);
    }

    @Override
    public void del(String... keys) {
        if (keys != null && keys.length > 0) {
            if (keys.length == 1) {
                redisTemplate.delete(keys[0]);
            } else {
                redisTemplate.delete(CollectionUtils.arrayToList(keys));
            }
        }
    }
}
```

应用场景

常应用于：对两个集合间的数据[计算]进行交集、并集、差集运算

1、利用集合操作，可以取不同兴趣圈子的交集,以非常方便的实现如共同关注、共同喜好、二度好友等功能。对上面的所有集合操作，你还可以使用不同的命令选择将结果返回给客户端还是存储到一个新的集合中。

2、利用唯一性，可以统计访问网站的所有独立 IP、存取当天[或某天]的活跃用户列表。

代码案例

案例1 是否存在校验

```
判断用户名/手机号/邮箱 是否存在
```

案例2 数据统计（抽奖等）

```
抽奖活动。
现有员工10个。1等奖1名。2等奖2名。3等奖3名。 用Redis实现
```

案例3 结果集运算

```
有两组数据，求两组数据的 交集、差集、并集
```

ZSet

有序集合(sorted set)

简介

1、 Redis 有序集合和集合一样也是string类型元素的集合,且不允许重复的成员。 2、不同的是每个元素都会关联一个double类型的分数。redis正是通过分数来为集合中的成员进行从小到大的排序。 3、有序集合的成员是唯一的,但分数(score)却可以重复。 4、集合是通过哈希表实现的。 集合中最大的成员数为2次方32 - 1 (4294967295, 每个集合可存储40多亿个成员)。Redis的ZSet是有序、且不重复 （很多时候，我们都将redis中的有序集合叫做zsets，这是因为在redis中，有序集合相关的操作指令都是以z开头的）

命令

```
赋值语法:
    ZADD key score1 member1 [score2 member2]  :向有序集合添加一个或多个成员，或者更新已存在成员的分数

取值语法:
    ZCARD key  :获取有序集合的成员数
    ZCOUNT key min max :计算在有序集合中指定区间分数的成员数
    ZRANK key member :返回有序集合中指定成员的索引
    ZRANGE key start stop [WITHSCORES] :通过索引区间返回有序集合成指定区间内的成员(低到高)
    ZRANGEBYSCORE key min max [WITHSCORES] [LIMIT] :通过分数返回有序集合指定区间内的成员
    ZREVRANGE key start stop [WITHSCORES] :返回有序集中指定区间内的成员，通过索引，分数从高到底
    ZREVRANGEBYSCORE key max min [WITHSCORES] :返回有序集中指定分数区间内的成员，分数从高到低排序
删除语法:
    DEL key   :移除集合
    ZREM key member [member ...] :移除有序集合中的一个或多个成员
    ZREMRANGEBYRANK key start stop :移除有序集合中给定的排名区间的所有成员(第一名是0)(低到高排序)
    ZREMRANGEBYSCORE key min max :移除有序集合中给定的分数区间的所有成员

    ZINCRBY  key increment member   :增加mememberer元素的分数increment，返回值是更改后的分数
```

代码示例

如下演示了ZSet类型所有命令的调用示例

```
/**
 * @ClassName ZSetCacheServiceImpl
 * @Description TODO
 * @Author guoweixin
 * @Version 1.0
 */
@Service("zsetCacheService")
public class ZSetCacheServiceImpl implements ZSetCacheService {
    private final static Logger log =
LoggerFactory.getLogger(ZSetCacheServiceImpl.class);

    @Autowired
```

```
    private RedisTemplate<String, Object> redisTemplate;

    /**
     * 增添加元素到变量中同时指定元素的分值。
     * @param key 键
     * @param value 值
     * @param score 分值
     * @return true 成功 false 失败
     */
    public boolean add(String key, Object value, double score){
        try {
            redisTemplate.opsForZSet().add(key,value,score);
            return true;
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }

    /**
     * 获取变量指定区间的元素。START为0,END为-1代表取全部
     * @param key 键
     * @param start 开始位置
     * @param end 结束位置
     * @return 返回SET
     */
    @Override
    public Set<Object> range(String key, long start, long end) {
        return redisTemplate.opsForZSet().range(key,start,end);
    }

    /**
     * 用于获取满足非score的排序取值。这个排序只有在有相同分数的情况下才能使用，如果有不同的分数则返回值不确定。
     * @param key 键
     * @param range
     * @return 返回SET
     */
    public Set<Object> rangeByLex(String key, RedisZSetCommands.Range range){
        return redisTemplate.opsForZSet().rangeByLex(key,range);
    }

    /**
     * 获取变量中元素的个数
     * @param key 键
     * @return 返回个数
     */
    public long zCard(String key){
        return redisTemplate.opsForZSet().zCard(key);
    }

    /**
     * 获取区间值的个数。
     * @param key 键
     * @param min 最小SCORE
     * @param max 最大SCORE
     * @return 返回数量
     */
```

```
    @Override
    public long count(String key, double min, double max) {
        return redisTemplate.opsForZSet().count(key,min,max);
    }

    /**
     * 修改变量中的元素的分值。
     * @param key
     * @param value
     * @param delta
     * @return
     */
    @Override
    public double incrementScore(String key, Object value, double delta) {
        return redisTemplate.opsForZSet().incrementScore(key,value,delta);
    }

    /**
     * 获取元素的分值
     * @param key 键
     * @param o 要查找的值
     * @return 返回分值
     */
    public double score(String key, Object o){
        return redisTemplate.opsForZSet().score(key,o);
    }

    /**
     * 用于获取满足非score的设置下标开始的长度排序取值。
     * @param key 键
     * @param range 范围
     * @param limit 限制区域
     * @return 返回SET
     */
    @Override
    public Set<Object> rangeByLex(String key, RedisZSetCommands.Range range,
RedisZSetCommands.Limit limit) {
        return redisTemplate.opsForZSet().rangeByLex(key, range,limit);
    }

    /**
     * 通过TypedTuple方式新增数据。
     * @param key 键
     * @param tuples 元组
     */
    @Override
    public void add(String key, Set<ZSetOperations.TypedTuple<Object>> tuples) {
        redisTemplate.opsForZSet().add(key,tuples);
    }

    /**
     * 根据设置的score获取区间值
     * @param key 键
     * @param min 最小值
     * @param max 最大值
     * @return 返回SET
     */
    @Override
```

```
    public Set<Object> rangeByScore(String key, double min, double max) {
        return redisTemplate.opsForZSet().rangeByScore(key,min,max);
    }

    /**
     * 根据设置的score获取区间值从给定下标和给定长度获取最终值。
     * @param key 键
     * @param min 最小值
     * @param max 最大值
     * @param offset 偏移时
     * @param count 取的长度
     * @return 返回SET
     */
    @Override
    public Set<Object> rangeByScore(String key, double min, double max, long offset,
long count) {
        return redisTemplate.opsForZSet().rangeByScore(key,min,max,offset,count);
    }

    /**
     * 获取RedisZSetCommands.Tuples的区间值。
     * @param key 键
     * @param start 开始SCORE值
     * @param end   结束SCORE值
     * @return 返回区间值
     */
    @Override
    public Set<ZSetOperations.TypedTuple<Object>> rangeWithScores(String key, long
start, long end) {
        return redisTemplate.opsForZSet().rangeWithScores(key,start,end);
    }
/**
TypedTuple是ZSetOperations 接口的内部接口，它定义了两个方法
getValue()是获取值，getScore()是获取分数，只是一个接口。
spring-data-redis提供了一个默认的实现类-- DefaultTypedTuple，实现TypeTuple接口，
在默认的情况下Spring就会把带有分数的有序集合的值和分数封装到这个类中，可以通过这个类读取对应的值和分
数。
*/
   public void test19(){
        Set<ZSetOperations.TypedTuple<Object>> set1=
 redisTemplate.opsForZSet().reverseRangeWithScores("z1",0,-1);
        Iterator<ZSetOperations.TypedTuple<Object>> it= set1.iterator();
        while(it.hasNext()){
            ZSetOperations.TypedTuple<Object> obj= it.next();
            System.out.println(obj.getValue()+"\t"+obj.getScore());
        }



    /**
     * 获取RedisZSetCommands.Tuples的区间值通过分值。
     * @param key 键
     * @param min 最小分值
     * @param max 最大分值
     * @return 返回SET
     */
    @Override
```

```
    public Set<ZSetOperations.TypedTuple<Object>> rangeByScoreWithScores(String key,
double min, double max) {
        return redisTemplate.opsForZSet().rangeByScoreWithScores(key, min, max);
    }

    /**
     * 获取RedisZSetCommands.Tuples的区间值从给定下标和给定长度获取最终值通过分值。
     * @param key 键
     * @param min 最小分值
     * @param max 最大分值
     * @param offset 偏移量
     * @param count 总数
     * @return 返回SET
     */
    @Override
    public Set<ZSetOperations.TypedTuple<Object>> rangeByScoreWithScores(String key,
double min, double max, long offset, long count) {
        return redisTemplate.opsForZSet().rangeByScoreWithScores(key, min,
max,offset,count);
    }

    /**
     * 获取变量中元素的索引,下标开始位置为
     * @param key 键
     * @param o 要查找的值
     * @return 返回下标
     */
    @Override
    public long rank(String key, Object o) {
        return redisTemplate.opsForZSet().rank(key,o);
    }

    /**
     * 匹配获取键值对，ScanOptions.NONE为获取全部键值对；
ScanOptions.scanOptions().match("C*").build()匹配获取键位map1的键值对,不能模糊匹配。
     * @param key 键
     * @param options 选项
     * @return 返回键值对
     */
    @Override
    public Cursor<ZSetOperations.TypedTuple<Object>> scan(String key, ScanOptions
options) {
        return redisTemplate.opsForZSet().scan(key, options);
    }

    /**
     * 索引倒序排列指定区间元素。
     * @param key 键
     * @param start 开始位置
     * @param end 结束位置
     * @return 返回倒排后的结果
     */
    @Override
    public Set<Object> reverseRange(String key, long start, long end) {
        return redisTemplate.opsForZSet().reverseRange(key,start,end);
    }

    /**
```

```
     * 倒序排列指定分值区间元素。
     * @param key 键
     * @param min 最小SCORE
     * @param max 最大SCORE
     * @return 返回区间元素
     */
    @Override
    public Set<Object> reverseRangeByScore(String key, double min, double max) {
        return redisTemplate.opsForZSet().reverseRangeByScore(key,min,max);
    }

    /**
     * 倒序排列从给定下标和给定长度分值区间元素。
     * @param key 键
     * @param min 最小SCORE
     * @param max 最大SCORE
     * @param offset 偏移量
     * @param count  数量
     * @return 返回列表
     */
    @Override
    public Set<Object> reverseRangeByScore(String key, double min, double max, long
offset, long count) {
        return
redisTemplate.opsForZSet().reverseRangeByScore(key,min,max,offset,count);
    }

    /**
     * 倒序排序获取RedisZSetCommands.Tuples的分值区间值
     * @param key 键
     * @param min 最小SCORE
     * @param max 最大SCORE
     * @return 返回SET集合
     */
    @Override
    public Set<ZSetOperations.TypedTuple<Object>> reverseRangeByScoreWithScores(String
key, double min, double max) {
        return redisTemplate.opsForZSet().reverseRangeByScoreWithScores(key,min,max);
    }

    /**
     * 序排序获取RedisZSetCommands.Tuples的从给定下标和给定长度分值区间值
     * @param key 键
     * @param min 最小SCORE
     * @param max 最大SCORE
     * @param offset 偏移量
     * @param count 总数
     * @return 返回SET
     */
    @Override
    public Set<ZSetOperations.TypedTuple<Object>> reverseRangeByScoreWithScores(String
key, double min, double max, long offset, long count) {
        return
redisTemplate.opsForZSet().reverseRangeByScoreWithScores(key,min,max,offset,count);
    }

    /**
     * 索引倒序排列区间值。
```

```
     * @param key 键
     * @param start 开始Score
     * @param end   结束SCORE
     * @return      返回列表
     */
    @Override
    public Set<ZSetOperations.TypedTuple<Object>> reverseRangeWithScores(String key,
long start, long end) {
        return redisTemplate.opsForZSet().reverseRangeWithScores(key,start,end);
    }

    /**
     * 获取倒序排列的索引值。
     * @param key 键
     * @param o   值
     * @return    返回倒序排列的索引值
     */
    @Override
    public long reverseRank(String key, Object o) {
        return redisTemplate.opsForZSet().reverseRank(key,o);
    }

    /**
     * 获取2个变量的交集存放到第3个变量里面。
     * @param key 键
     * @param otherKey 要交集的键
     * @param destKey 目标键
     * @return 返回交集长度
     */
    @Override
    public long intersectAndStore(String key, String otherKey, String destKey) {
        return redisTemplate.opsForZSet().intersectAndStore(key,otherKey,destKey);
    }

    /**
     * 获取多个变量的交集存放到第3个变量里面。
     * @param key 键
     * @param list 多个要交集的KEY
     * @param destKey 要存入的KEY
     * @return 返回数量
     */
    @Override
    public long intersectAndStore(String key, List list, String destKey) {
        return redisTemplate.opsForZSet().intersectAndStore(key,list,destKey);
    }

    /**
     * 获取2个变量的合集存放到第3个变量里面。
     * @param key 键
     * @param otherKey 要合并的KEY
     * @param destKey  共同的并集元素存到destK
     * @return 返回元素个数
     */
    @Override
    public long unionAndStore(String key, String otherKey, String destKey) {
        return redisTemplate.opsForZSet().unionAndStore(key,otherKey,destKey);
    }
```

```
    /**
     * 获取多个变量的合集存放到第3个变量里面。
     * @param key 键
     * @param list 要合的集合KEY
     * @param destKey 目票集合KEY
     * @return 返回合集长度
     */
    @Override
    public long unionAndStore(String key, List list, String destKey) {
        return redisTemplate.opsForZSet().unionAndStore(key,list,destKey);
    }

    /**
     * 批量移除元素根据元素值。
     * @param key 键
     * @param values 要删除的元素
     * @return 返回删除的数量
     */
    @Override
    public long remove(String key, Object... values) {
        return redisTemplate.opsForZSet().remove(key,values);
    }

    /**
     * 根据分值移除区间元素。
     * @param key 键
     * @param min 最小的SCORE
     * @param max 最大的SCORE
     * @return 返回移除的元素数量
     */
    @Override
    public long removeRangeByScore(String key, double min, double max) {
        return redisTemplate.opsForZSet().removeRangeByScore(key,min,max);
    }

    /**
     * 根据索引值移除区间元素。
     * @param key 键
     * @param start 索引开始
     * @param end   索引结束
     * @return 返回移除的数量
     */
    @Override
    public long removeRange(String key, long start, long end) {
        return redisTemplate.opsForZSet().removeRange(key,start,end);
    }

    /**
     * 删除指定的KEY的缓存
     * @param keys
     */
    @Override
    public void del(String... keys) {
        if (keys != null && keys.length > 0) {
            if (keys.length == 1) {
                redisTemplate.delete(keys[0]);
            } else {
                redisTemplate.delete(CollectionUtils.arrayToList(keys));
```

```
            }
        }
    }

}
```

应用场景

常应用于：排行榜

销量排名，积分排名等

> 1比如twitter 的public timeline可以以发表时间作为score来存储，这样获取时就是自动按时间排好序的。
>
> 2比如一个存储全班同学成绩的Sorted Set，其集合value可以是同学的学号，而score就可以是其考试得分，这样在数据插入集合的时候，就已经进行了天然的排序。
>
> 3还可以用Sorted Set来做带权重的队列，比如普通消息的score为1，重要消息的score为2，然后工作线程可以选择按score的倒序来获取工作任务。让重要的任务优先执行。

```
登录网站：
  1、实名认证      1
  2、收货地址      3
  3、绑定银行卡    2
```

代码案例

案例1 排行榜

```
积分、成绩、等等排行榜
```

学员成绩排行榜：

```
需求1：在zset中插入10名同学成绩
需求2：按成绩由高到低，查出前3名同学成绩信息
需求3：查询成绩在60- 80分之间 的同学成绩信息
```

案例2 奥运金牌排行榜

实体Bean

```
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Rank {

    private String country;
    private int score;
    private int goldCount;
    private int silverCount;
    private int copperCount;
}
```

Service业务逻辑层

```
@Service
@Slf4j
public class RankingServiceImpl implements RankingService {

    @Autowired
    private RedisTemplate<String,String> redisTemplate;

    @Override
    public List<Rank> ranking(String country,Double score) {

        if(country==null||country.equals("")){
        }else{
            redisTemplate.opsForZSet().incrementScore("rank",country,score);
        }


        Set<String> set = redisTemplate.opsForZSet().reverseRange("rank", 0, -1);
        List<Rank> list=new ArrayList<>();

        for (String s : set) {

            Rank rank1 = new Rank();
            Double rank = redisTemplate.opsForZSet().score("rank", s);
            int gold= (int) (rank/10000);
            int silver= (int) (rank/100%100);
            int copper= (int) (rank%100);
            int count=gold+silver+copper;
            rank1.setCountry(s);
            rank1.setGoldCount(gold);
            rank1.setSilverCount(silver);
            rank1.setCopperCount(copper);
            rank1.setScore(count);
            list.add(rank1);
        }
        return list;
    }
}
```

controller层

```
@Controller
public class RankingController {

    @Autowired
```

```
    private RankingService rankingService;

    @RequestMapping("index")
    public String index(){
        return "index";
    }

    @RequestMapping("ranking")
    public String ranking(@Param("country") String country,@Param("score")  Double
score, Model model){
        List<Rank> rankList = rankingService.ranking(country, score);
        model.addAttribute("rankList",rankList);

        return "forward:index";
    }

}
```

前端页面

```
<!DOCTYPE html>
<html  xmlns:th="http://www.thymeleaf.org">
<head>
    <title>奖牌排行榜</title>
    <style>
        table tr th{
            text-align: center;
        }
    </style>
</head>
<link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@3.3.7/dist/css/bootstrap.min.css">
<script type="text/javascript" src="http://libs.baidu.com/jquery/2.0.0/jquery.min.js">
</script>
<script src="https://cdn.jsdelivr.net/npm/bootstrap@3.3.7/dist/js/bootstrap.min.js">
</script>
<body>


<div >
    <table class="table table-hover" style="text-align: center">
        <tr>
            <th>排序</th>
            <th>国家</th>
            <th>金牌</th>
            <th>银牌</th>
            <th>铜牌</th>
            <th>奖牌总数</th>
            <th width="200px">操作</th>
        </tr>
        <tr th:each="r:${rankList}">
            <td th:text="${rStat.count}">序号</td>
            <td th:text="${r.country}">国家</td>
            <td th:text="${r.goldCount}">金牌</td>
            <td th:text="${r.silverCount}">银牌</td>
            <td th:text="${r.copperCount}">铜牌</td>
            <td th:text="${r.score}">奖牌总数</td>
```

```
            <td>金:<a href="javascript:goldin()"
th:href="|javascript:goldin('${r.country}')|" class="glyphicon glyphicon-plus">
</a>  银:<a href="javascript:silverin()"
th:href="|javascript:silverin('${r.country}')|" class="glyphicon glyphicon-plus">
</a>  铜:<a href="javascript:copperin()"
th:href="|javascript:copperin('${r.country}')|" class="glyphicon glyphicon-plus"></a>
</td>
        </tr>

    </table>
</div>

</body>
<script type="text/javascript" src="js/jquery.min.js"></script>
<script type="text/javascript" src="js/bootstrap.min.js"></script>
<script type="text/javascript">

    function goldin(country) {
        var score=10000.0;
        $.ajax({
            type:"post",
            url:"ranking",
            data:{"country":country,"score":score},
            success:function () {
                    location.href="ranking"
            }
        })
    }
    function silverin(country) {
        var score=100.0;
        $.ajax({
            type:"post",
            url:"ranking",
            data:{"country":country,"score":score},
            success:function (){
                location.href="ranking"
            }
        })
    }
    function copperin(country) {
        var score=1.0;
        $.ajax({
            type:"post",
            url:"ranking",
            data:{"country":country,"score":score},
            success:function () {
                location.href="ranking"
            }
        })
    }
</script>
</html>
```

案例3 Redis限流

HyperLogLog

简介

Redis 在 2.8.9 版本添加了 HyperLogLog 结构。
Redis HyperLogLog 是用来做基数统计的算法，HyperLogLog 的优点是，在输入元素的数量或者体积非常非常大时，计算基数所需的空间总是固定的、并且是很小的。

在 Redis 里面，每个 HyperLogLog 键只需要花费 12 KB 内存，就可以计算接近 2^64 个不同元素的基数。这和计算基数时，元素越多耗费内存就越多的集合形成鲜明对比。
但是，因为 HyperLogLog 只会根据输入元素来计算基数，而不会储存输入元素本身，所以 HyperLogLog 不能像集合那样，返回输入的各个元素。

小知识：
什么是基数?
比如数据集 {1, 3, 5, 7, 5, 7, 8}， 那么这个数据集的基数集为 {1, 3, 5 ,7, 8}, 基数(不重复元素)为5。 基数估计就是在误差可接受的范围内，快速计算基数。

为什么需要HyperLoglog

如果要统计1亿个数据的基数值，大约需要内存100000000/8/1024/1024 ≈ 12M，内存减少占用的效果显著。

然而统计一个对象的基数值需要12M，如果统计10000个对象，就需要将近120G，同样不能广泛用于大数据场景。
因此，元素的数量和内存的消耗量是成正比的。

常用命令

```
PFADD key element [element ...] :添加指定元素到 HyperLogLog 中
PFCOUNT key [key ...] :返回给定 HyperLogLog 的基数估算值
PFMERGE destkey sourcekey [sourcekey ...] :将多个 HyperLogLog 合并为一个 HyperLogLog
```

应用场景 数据统计

基数不大，数据量不大就用不上，会有点大材小用浪费空间 有局限性，就是只能统计基数数量，而没办法去知道具体的内容是什么

示例：在网站分析方面有几个统计指标，统计总访问量、统计访问人数。
统计总访问量只需要每次来+1即可。但统计访问人数需要不断的去重判断。某个用户某天访问100次，但在统计人数中仅算1次，此时 Hyperlog是最优的方案。

统计每天访问人数
统计注册 IP 数
统计每日总访问 IP 数
统计页面实时 UV 数
统计在线用户数
统计用户每天搜索不同词条的个数
统计真实文章阅读数

总结

HyperLogLog是一种算法，并非redis独有
目的是做基数统计，故不是集合，不会保存元数据，只记录数量而不是数值。
耗空间极小，支持输入非常体积的数据量

核心是基数估算算法，主要表现为计算时内存的使用和数据合并的处理，最终数值存在一定误差
redis中每个hyperloglog key占用了12K的内存用于标记基数（官方文档）
pfadd命令并不会一次性分配12k内存，而是随着基数的增加而逐渐增加内存分配，而pfmerge操作则会将sourcekey合并后存储在12k大小的key中，这由hyperloglog合并操作的原理（两个hyperloglog合并时需要单独比较每个桶的值）可以很容易理解。

误差说明：基数估计的结果是一个带有 0.81% 标准错误（standard error）的近似值。是可接受的范围
Redis 对 HyperLogLog 的存储进行了优化，在计数比较小时，它的存储空间采用稀疏矩阵存储，空间占用很小，仅仅在计数慢慢变大，稀疏矩阵占用空间渐渐超过了阈值时才会一次性转变成稠密矩阵，才会占用 12k 的空间