链滴

Redis 基础入门

作者：Wuhon

原文链接：https://ld246.com/article/1615987400568

来源网站：链滴

非关系型数据库redis

1. 基本命令

1. set key value # 设置键 值
2. flushall # 清除所有数据库的值
3. keys * # 查看所有的键
4. exists key # 判断是否存在这个key
5. select index # 根据index切换数据库 从0到15 默认
6. move key index # 将key移动到数据库index中
7. expire key seconds # 设置key在多少秒之后过期
8. flushdb # 清空当前数据库的值

2. 五大数据类型

Redis-Key

String

```
127.0.0.1:6379[1]> set name wuhobin
OK
127.0.0.1:6379[1]> append name "hello"  # 向key中追加 如果追加的key不存在就相当于set
(integer) 12
127.0.0.1:6379[1]> get name
"wuhobinhello"

127.0.0.1:6379[1]> expire name 10   # 设置key过期的时间
(integer) 1
```

```
127.0.0.1:6379[1]> strlen name   #  获取key的字符串的长度
(integer) 7

127.0.0.1:6379[1]> get views
"0"
127.0.0.1:6379[1]> incr views   #  让当前key加1
(integer) 1
127.0.0.1:6379[1]> get views
"1"

127.0.0.1:6379[1]> decr views  # 让当前key减去1
(integer) 0
127.0.0.1:6379[1]> get views
"0"

127.0.0.1:6379[1]> incrby views 10  # 让当前key自增 每次增加10
(integer) 10
127.0.0.1:6379[1]> get views
"10"

127.0.0.1:6379[1]> decrby views 10  #让当前key自减 每次减去10
(integer) 0
127.0.0.1:6379[1]> get views
"0"


127.0.0.1:6379[1]> set name helloword
OK
127.0.0.1:6379[1]> getrange name 0 5    #  截取字符串 [0,5]
"hellow"

127.0.0.1:6379[1]> setrange name 0 xxx  # 替换 把之前的字符串的0的位置替换为xxx
(integer) 9
127.0.0.1:6379[1]> get name
"xxxloword"

127.0.0.1:6379[1]> setex key 30 hello  #  设置key过期时间为30秒 值为hello
OK
127.0.0.1:6379[1]> ttl key   #  查看key还有多少时间过去
(integer) 27
127.0.0.1:6379[1]> get key
"hello"

127.0.0.1:6379[1]> setnx mykey hello  # set if not exist 当mykey不存在时设置mykey的值为hello
如果成功 返回1
(integer) 1

127.0.0.1:6379[1]> mset k1 v1 k2 v2 k3 v3   #  同时设置多个值
OK
127.0.0.1:6379[1]> keys *
1) "k3"
2) "k1"
3) "k2"
```

```
127.0.0.1:6379[1]> set user:1 {name:wuhobin,age:20,girFirend:sunsi} # 设置一个user:1对象,值为json字符来保存一个对象
OK
127.0.0.1:6379[1]> get user:1
"{name:wuhobin,age:20,girFirend:sunsi}"
127.0.0.1:6379[1]>
```

List

```
# redis的列表相当于是一个双端队列 lpush从上边入栈 rpush从下边入栈
127.0.0.1:6379[1]> lpush list one # 向列表list里面添加one元素
(integer) 1
127.0.0.1:6379[1]> lpush list two # 向列表list里面添加two元素
(integer) 2
127.0.0.1:6379[1]> lpush list three # 向列表list里面添加three元素
(integer) 3

127.0.0.1:6379[1]> lrange list 0 4
1) "three"
2) "three"
3) "two"
4) "one"
127.0.0.1:6379[1]> lrem list 1 three # 从列表中移除一个值为three的元素
(integer) 1
127.0.0.1:6379[1]> lrange list 0 4
1) "three"
2) "two"
3) "one"
```

Set

```
127.0.0.1:6379[1]> sadd list hello  # 向集合中添加元素
(integer) 1
127.0.0.1:6379[1]> sadd list hello
(integer) 0
127.0.0.1:6379[1]> sadd list wuhobin
(integer) 1
127.0.0.1:6379[1]> smembers list   #  查看集合里有哪些元素
1) "hello"
2) "wuhobin"
127.0.0.1:6379[1]>
127.0.0.1:6379[1]>
127.0.0.1:6379[1]> sismember list hello   # 看集合里面是否有hello 如果有返回1 否则返回0
(integer) 1
127.0.0.1:6379[1]> sismember list sss
(integer) 0

127.0.0.1:6379[1]> scard list   # 获取集合里面的个数值
(integer) 2

127.0.0.1:6379[1]> srem list hello # 从集合里移除某个元素
```

```
(integer) 1

127.0.0.1:6379[1]> srandmember list   # 从集合中随机获取一个元素
"wuhobin"

127.0.0.1:6379[1]> spop list  # 从集合里随机弹出一个元素
"wuhobin"
```

Hash

key-Map key里面存的是map集合

```
127.0.0.1:6379[1]> hset myhash name wuhobin  # 向hash中存值 以键值对的方式
(integer) 1
127.0.0.1:6379[1]> hset myhash age 21
(integer) 1

127.0.0.1:6379[1]> hget myhash name  # 从hash中取值
"wuhobin"
```

Zset

```
127.0.0.1:6379[1]> zadd salary 2500 wuhobin  #  向有序集合中添加元素
(integer) 1
127.0.0.1:6379[1]> zadd salary 500 sunsi
(integer) 1

127.0.0.1:6379[1]> ZRANGEBYSCORE salary -inf +inf  # 根据添加时的标识从小到大排序 -inf负
穷
1) "sunsi"
2) "cy"
3) "wuhobin"

127.0.0.1:6379[1]> ZRANGEBYSCORE salary -inf +inf withscores # 排序并且附带标识
1) "sunsi"
2) "500"
3) "cy"
4) "2000"
5) "wuhobin"
6) "2500"

127.0.0.1:6379[1]> zrange salary 0 -1
1) "sunsi"
2) "cy"
3) "wuhobin"
127.0.0.1:6379[1]> zrem salary sunsi   # 移除某一元素
(integer) 1
127.0.0.1:6379[1]> zrange salary 0 -1
1) "cy"
2) "wuhobin"
```

3. 三种特殊数据类型

geospatial

地理空间,地理位置,朋友圈,附近的人,距离,方圆半径的人

```
127.0.0.1:6379> geoadd ching:city 114.05 22.52 shenzhen  # 添加地理位置 参数 key 经度 纬度 城市
(integer) 1
127.0.0.1:6379> geoadd china:city 160.16 30.24 hangzhou1
(integer) 1
127.0.0.1:6379> geoadd china:city 108.96 34.26 xian
(integer) 1

127.0.0.1:6379> geopos china:city shanghai  #  查看某个城市的地理位置
1) 1) "116.00000113248825073"
   2) "45.00000100864581043"

127.0.0.1:6379> zrange china:city 0 -1  # 查看所有的元素
1) "shanghai"
2) "shanghai1"
3) "shanghai2"

127.0.0.1:6379> GEORADIUS china:city 114 66 10000 km withcoord  # 返回114 66半径1000k之内的地理位置的集合
1) 1) "shanghai"
   2) 1) "116.00000113248825073"
      2) "77.99999963605176845"
2) 1) "shanghai1"
   2) 1) "116.00000113248825073"
      2) "77.99999963605176845"
3) 1) "shanghai2"
   2) 1) "116.00000113248825073"
      2) "77.99999963605176845"
127.0.0.1:6379>
```

Hyperglog

基数统计的算法, 可以统计一个key里面不重复元素的个数

```
127.0.0.1:6379> PFADD ket1 1 122 2 3 4 4 1 14 4 5 3 # 添加元素
(integer) 1
127.0.0.1:6379> PFCOUNT ket1 # 查看集合的不重复元素的个数
(integer) 7
127.0.0.1:6379> PFMERGE ket3 ket1 ket2  # 合并两个key 求并集
OK
127.0.0.1:6379> PFCOUNT ket3
(integer) 14
```

Bitmaps

统计用户信息 活跃 不活跃 登陆 未登陆 365天打卡

位图 数据结构 都是操作二进制数来进行记录 就只有0 和1 两个状态

4. 事务

原子性: 要么同时成功, 要么同时失败

Redis事务本质 : 一组命令的集合! 一个事物中的所有命令都会被序列化, 在事务执行的过程中,会按照序执行

Redis单条命令是保证原子性的,事务是不保证原子性的

```
127.0.0.1:6379> multi  # 开启事务
OK
127.0.0.1:6379(TX)>
127.0.0.1:6379(TX)>
127.0.0.1:6379(TX)> set k1 v1  # 命令入队
QUEUED
127.0.0.1:6379(TX)> set k2 v2
QUEUED
127.0.0.1:6379(TX)> set k3 v3
QUEUED
127.0.0.1:6379(TX)> exec   # 执行事务
1) OK
2) OK
3) OK
```

编译有错误 代码错误 事务中所有的命令都不会执行

5. 悲观锁 乐观锁

悲观锁 : 认为无论干什么都会出问题, 无论做什么都会加锁

乐观锁 : 认为做什么都不会出现问题, 所以不会加锁

redis的监测测试

```
127.0.0.1:6379> set money 100
OK
127.0.0.1:6379> set out 0
OK
127.0.0.1:6379> watch money # 监视money
OK
127.0.0.1:6379> multi  # 开启事务
OK
127.0.0.1:6379(TX)> decrby money 20
QUEUED
127.0.0.1:6379(TX)> incrby out 20
QUEUED
127.0.0.1:6379(TX)> exec
1) (integer) 80
2) (integer) 20
```

当前开了两个线程, 一个线程监视一个线程改动

```
# 改动money
127.0.0.1:6379> get money
```

```
"80"
127.0.0.1:6379> set money 1000
OK
```

使用watch在这里相当于乐观锁的操作

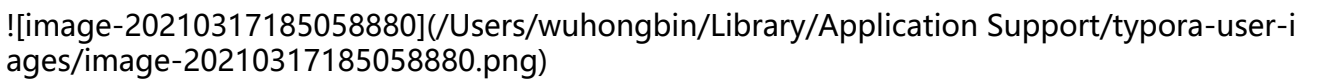
```
# 当前线程实现监控
127.0.0.1:6379> watch money
OK
127.0.0.1:6379> multi
OK
127.0.0.1:6379(TX)> DECRby money 10
QUEUED
127.0.0.1:6379(TX)> incrby out 10
QUEUED
127.0.0.1:6379(TX)> exec   # 当监视到另外一个线程导致了money的值发生了改变时, 再执行事务会执行失败
(nil)
```

6. Springboot整合redis

1. 添加springboot关于redis的依赖

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-redis</artifactId>
</dependency>
```

2. 在真实java开发中,我们一般都以json的格式来传递对象,所以在项目中都会让实体类来实现Serializable接口实现序列化,默认的redisTemplate的序列化是使用的jdk序列化, 获得的key名会乱码



这里我们需要自定义redisTemplate,设置不同的序列化方式

```
@Configuration
public class RedisConfig {
    @Bean
    public RedisTemplate<String, Object> redisTemplate(RedisConnectionFactory factory) {
        RedisTemplate<String, Object> template = new RedisTemplate<String, Object>();
        template.setConnectionFactory(factory);
        // 序列化配置
        Jackson2JsonRedisSerializer<Object> objectJackson2JsonRedisSerializer = new Jackson2JsonRedisSerializer(Object.class);
        ObjectMapper om = new ObjectMapper();
        om.setVisibility(PropertyAccessor.ALL, JsonAutoDetect.Visibility.ANY);
        om.enableDefaultTyping(ObjectMapper.DefaultTyping.NON_FINAL);
        objectJackson2JsonRedisSerializer.setObjectMapper(om);
        // String 序列化
        StringRedisSerializer stringRedisSerializer = new StringRedisSerializer();
        // key采用String的序列化方式
        template.setKeySerializer(stringRedisSerializer);
        // hash的key也采用string的序列化方式
```

```
        template.setHashKeySerializer(stringRedisSerializer);
        // value序列化方式采用jackson
        template.setValueSerializer(objectJackson2JsonRedisSerializer);
        // hash的value采用jackson
        template.setHashValueSerializer(objectJackson2JsonRedisSerializer);
        template.afterPropertiesSet();
        return template;
    }
}
```

自定义redisUtils工具类

```
@Component
public class RedisUtils {
    @Autowired
    private RedisTemplate<String, Object> redisTemplate;


    // ================common====================
    /**
     * 指定缓存的失效时间
     * @param key
     * @param time
     * @return
     */

    public boolean expire(String key,long time){
        try {
            if(time > 0 ){
                redisTemplate.expire(key,time, TimeUnit.SECONDS);
            }
            return true;
        }catch (Exception e){
            e.printStackTrace();
            return false;
        }
    }
    /**
     * 根据key 获取过期时间
     * @param key 键 不能为null
     * @return 时间(秒) 返回0代表为永久有效
     */
    public long getExpire(String key) {
        return redisTemplate.getExpire(key, TimeUnit.SECONDS);
    }

    /**
     * 判断key是否存在
     * @param key 键
     * @return true 存在 false不存在
     */
    public boolean hasKey(String key) {
        try {
            return redisTemplate.hasKey(key);
```

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

    /**
     * 删除缓存
     * @param key 可以传一个值 或多个
     */
    @SuppressWarnings("unchecked")
    public void del(String... key) {
        if (key != null && key.length > 0) {
            if (key.length == 1) {
                redisTemplate.delete(key[0]);
            } else {
                redisTemplate.delete((Collection<String>) CollectionUtils.arrayToList(key));
            }
        }
    }


    // ============================String=============================
    /**
     * 普通缓存获取
     * @param key 键
     * @return 值
     */
    public Object get(String key) {
        return key == null ? null : redisTemplate.opsForValue().get(key);
    }

    /**
     * 普通缓存放入
     * @param key 键
     * @param value 值
     * @return true成功 false失败
     */
    public boolean set(String key, Object value) {
        try {
            redisTemplate.opsForValue().set(key, value);
            return true;
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }

    /**
     * 普通缓存放入并设置时间
     * @param key 键
     * @param value 值
     * @param time 时间(秒) time要大于0 如果time小于等于0 将设置无限期
```

```
     * @return true成功 false 失败
     */
    public boolean set(String key, Object value, long time) {
        try {
            if (time > 0) {
                redisTemplate.opsForValue().set(key, value, time, TimeUnit.SECONDS);
            } else {
                set(key, value);
            }
            return true;
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }

    /**
     * 递增
     * @param key 键
     * @param delta 要增加几(大于0)
     * @return
     */
    public long incr(String key, long delta) {
        if (delta < 0) {
            throw new RuntimeException("递增因子必须大于0");
        }
        return redisTemplate.opsForValue().increment(key, delta);
    }



    /**
     * 递减
     * @param key 键
     * @param delta 要减少几(大于0)
     * @return
     */
    public long decr(String key, long delta) {
        if (delta < 0) {
            throw new RuntimeException("递减因子必须大于0");
        }
        return redisTemplate.opsForValue().increment(key, -delta);
    }

    // ================================Map=================================
    /**
     * HashGet
     * @param key 键 不能为null
     * @param item 项 不能为null
     * @return 值
     */
    public Object hget(String key, String item) {
        return redisTemplate.opsForHash().get(key, item);
```

```
}


    /**
     * 获取hashKey对应的所有键值
     * @param key 键
     * @return 对应的多个键值
     */
public Map<Object, Object> hmget(String key) {
    return redisTemplate.opsForHash().entries(key);
}

/**
 * HashSet
 * @param key 键
 * @param map 对应多个键值
 * @return true 成功 false 失败
 */
public boolean hmset(String key, Map<String, Object> map) {
    try {
        redisTemplate.opsForHash().putAll(key, map);
        return true;
    } catch (Exception e) {
        e.printStackTrace();
        return false;
    }
}

/**
 * HashSet 并设置时间
 * @param key 键
 * @param map 对应多个键值
 * @param time 时间(秒)
 * @return true成功 false失败
 */

public boolean hmset(String key, Map<String, Object> map, long time) {
    try {
        redisTemplate.opsForHash().putAll(key, map);
        if (time > 0) {
            expire(key, time);
        }
        return true;
    } catch (Exception e) {
        e.printStackTrace();
        return false;
    }
}


/**
 * 向一张hash表中放入数据,如果不存在将创建
 * @param key 键
 * @param item 项
```

```
     * @param value 值
     * @return true 成功 false失败
     */
    public boolean hset(String key, String item, Object value) {
        try {
            redisTemplate.opsForHash().put(key, item, value);
            return true;
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }

    /**
     * 向一张hash表中放入数据,如果不存在将创建
     * @param key 键
     * @param item 项
     * @param value 值
     * @param time 时间(秒) 注意:如果已存在的hash表有时间,这里将会替换原有的时间
     * @return true 成功 false失败
     */
    public boolean hset(String key, String item, Object value, long time) {
        try {
            redisTemplate.opsForHash().put(key, item, value);
            if (time > 0) {
                expire(key, time);
            }
            return true;
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }

    /**
     * 删除hash表中的值
     * @param key 键 不能为null
     * @param item 项 可以使多个 不能为null
     */
    public void hdel(String key, Object... item) {
        redisTemplate.opsForHash().delete(key, item);
    }

    /**
     * 判断hash表中是否有该项的值
     * @param key 键 不能为null
     * @param item 项 不能为null
     * @return true 存在 false不存在
     */
    public boolean hHasKey(String key, String item) {
        return redisTemplate.opsForHash().hasKey(key, item);
    }

    /**
```

```
     * hash递增 如果不存在,就会创建一个 并把新增后的值返回
     * @param key 键
     * @param item 项
     * @param by 要增加几(大于0)
     * @return
     */
    public double hincr(String key, String item, double by) {
        return redisTemplate.opsForHash().increment(key, item, by);
    }

        /**
         * hash递减
         * @param key 键
         * @param item 项
         * @param by 要减少记(小于0)
         * @return
        285
         */
    public double hdecr(String key, String item, double by) {
        return redisTemplate.opsForHash().increment(key, item, -by);
    }



    // =============================set=============================
=

        /**
         * 根据key获取Set中的所有值
         * @param key 键
         * @return
         */
    public Set<Object> sGet(String key) {
        try {
            return redisTemplate.opsForSet().members(key);
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }

        /**
         * 根据value从一个set中查询,是否存在
         * @param key 键
         * @param value 值
         * @return true 存在 false不存在
         */
    public boolean sHasKey(String key, Object value) {
        try {
            return redisTemplate.opsForSet().isMember(key, value);
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
```

```
}

    /**
     * 将数据放入set缓存
     * @param key 键
     * @param values 值 可以是多个
     * @return 成功个数
     */
public long sSet(String key, Object... values) {
    try {
        return redisTemplate.opsForSet().add(key, values);
    } catch (Exception e) {
        e.printStackTrace();
        return 0;
    }
}

    /**
     * 将set数据放入缓存
     * @param key 键
     * @param time 时间(秒)
     * @param values 值 可以是多个
     * @return 成功个数
     */
public long sSetAndTime(String key, long time, Object... values) {
    try {
        Long count = redisTemplate.opsForSet().add(key, values);
        if (time > 0)
        expire(key, time);
        return count;
    } catch (Exception e) {
        e.printStackTrace();
        return 0;
    }
}


    /**
     * 获取set缓存的长度
     * @param key 键
     * @return
     */
public long sGetSetSize(String key) {
    try {
        return redisTemplate.opsForSet().size(key);
    } catch (Exception e) {
        e.printStackTrace();
        return 0;
    }
}


    /**
     * 移除值为value的
```

```
     * @param key 键
     * @param values 值 可以是多个
     * @return 移除的个数
     */
    public long setRemove(String key, Object... values) {
        try {
            Long count = redisTemplate.opsForSet().remove(key, values);
            return count;
        } catch (Exception e) {
            e.printStackTrace();
            return 0;
        }
    }


    // ================================list=============================
========

    /**
     * 获取list缓存的内容
     * @param key 键
     * @param start 开始
     * @param end 结束 0 到 -1代表所有值
     * @return
     */
    public List<Object> lGet(String key, long start, long end) {
        try {
            return redisTemplate.opsForList().range(key, start, end);
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }



    /**
     * 获取list缓存的长度
     * @param key 键
     * @return
     */
    public long lGetListSize(String key) {
        try {
            return redisTemplate.opsForList().size(key);
        } catch (Exception e) {
            e.printStackTrace();
            return 0;
        }
    }



    /**
     * 通过索引 获取list中的值
```

```
     * @param key 键
     * @param index 索引 index>=0时， 0 表头，1 第二个元素，依次类推；index<0时，-1，
尾，-2倒数第二个元素，依次类推
     * @return
     */
    public Object lGetIndex(String key, long index) {
        try {
            return redisTemplate.opsForList().index(key, index);
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }

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


    /*** 将list放入缓
     * @param key 键
     * @param value 值
```

```
     * @return
     */
    public boolean lSet(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
     * @param key 键
     * @param value 值
     * @param time 时间(秒)
     * @return
     */
    public boolean lSet(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;
        }
    }


    /**
     * 根据索引修改list中的某条数据
     * @param key 键
     * @param index 索引
     * @param value 值
     * @return
    509
     */
    public boolean lUpdateIndex(String key, long index, Object value) {
        try {
            redisTemplate.opsForList().set(key, index, value);
            return true;
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }


    /**
     * 移除N个值为value
     * @param key 键
```

```
     * @param count 移除多少个
     * @param value 值
     * @return 移除的个数
     */
    public long lRemove(String key, long count, Object value) {
        try {
            Long remove = redisTemplate.opsForList().remove(key, count, value);
            return remove;
        } catch (Exception e) {
            e.printStackTrace();
            return 0;
        }
    }
}
```