jedis采用的是直连redis server，在多个线程之间共用一个jedis实例时，是线程不安全的。如果想避免线程不安全，可以使用连接池pool，这样每个线程单独使用一个jedis实例。由此带来的问题时，如果线程数过多，带来redis server的负载加大。有点类似于BIO的模式。

一、使用Jedis

1.1 pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.3.0.RELEASE</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.wl.redis</groupId>
    <artifactId>jedis</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>jedis</name>
    <description>SpringBoot整合redis之Jedis</description>

    <properties>
        <java.version>1.8</java.version>
    </properties>

    <dependencies>
        <!-- 引入jedis依赖包 -->
        <dependency>
            <groupId>redis.clients</groupId>
            <artifactId>jedis</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
```

```
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

1.2 key+五大数据类型操作

```
package com.wl.redis.jedis;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.junit4.SpringRunner;

@RunWith(SpringRunner.class)
@SpringBootTest
public class JedisApplicationTests {

    @Test
    public void contextLoads() {
    }

}
```

```
package com.wl.redis.jedis.jedis;

import com.wl.redis.jedis.JedisApplicationTests;
import lombok.extern.slf4j.Slf4j;
import org.junit.Test;
import redis.clients.jedis.Jedis;

import java.util.*;

/**
 * @author 王柳
 * @date 2020/5/18 17:12
 */
@Slf4j
public class TestJedis extends JedisApplicationTests {

    /**
     * 测试Jedis连接
     */
    @Test
    public void test01() {
```

```
        Jedis jedis = new Jedis("127.0.0.1", 6379);
        jedis.auth("ahhs2019");
        log.info("connection is OK==========>" + jedis.ping());
    }

    /**
     * 测试 一个key + 五大数据类型
     */
    @Test
    public void test02() {

        Jedis jedis = new Jedis("127.0.0.1", 6379);
        jedis.auth("ahhs2019");

        //key
        log.info("key----------------------------------------");
        Set<String> keys = jedis.keys("*");
        for (Iterator iterator = keys.iterator(); iterator.hasNext(); ) {
            String key = (String) iterator.next();
            log.info(key);
        }
        log.info("判断某个key是否存在====>" + jedis.exists("k2"));

        log.info("查看还有多少秒过期，-1表示永不过期，-2表示已过期=======>" +
jedis.ttl("k1"));

        //String
        log.info("String----------------------------------------");
        log.info("append之前：" + jedis.get("k1"));
        jedis.append("k1","myreids");
        log.info("append之后：" + jedis.get("k1"));

        // 设置k-v
        jedis.set("k4", "k4_redis");

        // 设置多个k-v
        jedis.mset("str1", "v1", "str2", "v2", "str3", "v3");
        log.info("获取多个k-v：" + jedis.mget("str1", "str2", "str3"));

        //list
        log.info("list----------------------------------------");
        // 从左边添加list
        jedis.lpush("mylist","v1","v2","v3","v4","v5");
        // lrange 0, -1 获取list所有
        List<String> list = jedis.lrange("mylist", 0, -1);
        for (String element : list) {
            log.info(element);
        }

        //set
        log.info("set----------------------------------------");
        // 添加set
        jedis.sadd("orders", "jd001");
        jedis.sadd("orders", "jd002");
        jedis.sadd("orders", "jd003");
        Set<String> set1 = jedis.smembers("orders");
        for (Iterator iterator = set1.iterator(); iterator.hasNext(); ) {
            String string = (String) iterator.next();
```

```
            log.info(string);
        }
        // 删除集合中元素
        jedis.srem("orders", "jd002");
        log.info("获取指定key中的所有元素集合" + jedis.smembers("orders").size());

        //hash
        log.info("hash----------------------------------------");
        // 添加hash
        jedis.hset("hash1", "userName", "lisi");
        // 获取hash中的指定字段的值
        log.info(jedis.hget("hash1", "userName"));
        Map<String, String> map = new HashMap<String, String>();
        map.put("telphone", "13811814763");
        map.put("address", "atguigu");
        map.put("email", "abc@163.com");
        // 批量添加hash
        jedis.hmset("hash2", map);
        // 批量获取hash
        List<String> result = jedis.hmget("hash2", "telphone", "email");
        for (String element : result) {
            log.info(element);
        }

        //zset
        log.info("zset----------------------------------------");
        // 添加zset
        jedis.zadd("zset01", 60d, "v1");
        jedis.zadd("zset01", 70d, "v2");
        jedis.zadd("zset01", 80d, "v3");
        jedis.zadd("zset01", 90d, "v4");

        // 获取指定key中的所有元素集合
        Set<String> s1 = jedis.zrange("zset01", 0, -1);
        for (Iterator iterator = s1.iterator(); iterator.hasNext(); ) {
            String string = (String) iterator.next();
            log.info(string);
        }


    }
}
```

1.3 事务提交

1、日常

```
/**
     * 事务提交：日常
     */
    @Test
```

```
    public void test03() {
        Jedis jedis = new Jedis("127.0.0.1", 6379);
        jedis.auth("ahhs2019");

        //监控key，如果改动了事务就被放弃
//        jedis.watch("serialNum");
//        jedis.set("serialNum", "s#####################");
//        jedis.unwatch();

        //被当作一个命令进行执行，标记一个事务的开始
        Transaction transaction = jedis.multi();
        Response<String> response = transaction.get("serialNum");
        transaction.set("serialNum", "s002");
        response = transaction.get("serialNum");
        transaction.lpush("list3", "a");
        transaction.lpush("list3", "b");
        transaction.lpush("list3", "c");

        Response<List<String>> response2 = transaction.lrange("list3",0,-1);

        // 执行事务块所有命令
        transaction.exec();
        // 取消事务
//        transaction.discard();
        log.info("serialNum***********" + response.get());
        log.info("list3***********" + response2.get());
    }
```

2、加锁

```
    /**
     * 通俗点讲，watch命令就是标记一个键，如果标记了一个键， 在提交事务前如果该键被别人修改过，那事务就会失败，这种情况通常可以在程序中
     * 重新再尝试一次。
     * 首先标记了键balance，然后检查余额是否足够，不足就取消标记，并不做扣减； 足够的话，就启动事务进行更新操作，
     * 如果在此期间键balance被其它人修改， 那在提交事务（执行exec）时就会报错， 程序中通常可以捕获这类错误再重新执行一次，直到成功。
     */
    /**
     * 事务提交：加锁
     */
    @Test
    public void Test04() {
        Jedis jedis = new Jedis("127.0.0.1", 6379);
        jedis.auth("ahhs2019");
        int balance;// 可用余额
        int debt;// 欠额
        int amtToSubtract = 10;// 实刷额度

        jedis.set("balance","100");
        jedis.set("debt","0");
        jedis.watch("balance");
```

```
//        jedis.set("balance","5");//此句不该出现，讲课方便。模拟其他程序已经修改了该条目
        balance = Integer.parseInt(jedis.get("balance"));
        log.info("balance===> " + balance);
        if (balance < amtToSubtract) {
            jedis.unwatch();
            log.info("modify");
        } else {
            log.info("***********transaction");
            Transaction transaction = jedis.multi();
            transaction.decrBy("balance", amtToSubtract);
            transaction.incrBy("debt", amtToSubtract);
            transaction.exec();
            balance = Integer.parseInt(jedis.get("balance"));
            debt = Integer.parseInt(jedis.get("debt"));

            log.info("*******" + balance);
            log.info("*******" + debt);
        }
    }
```

1.4 主从复制

```
    /**
     * 6379,6380启动，先各自先独立
     * 主写
     * 从读
     */
    @Test
    public void Test05() throws InterruptedException {
        Jedis jedis_M = new Jedis("192.168.253.131",6379);
        jedis_M.auth("123456");
        Jedis jedis_S = new Jedis("192.168.253.131",6380);
        jedis_S.auth("123456");

        log.info(jedis_S.slaveof("192.168.253.131",6379));

        jedis_M.set("k6","v6");
        Thread.sleep(500);
        log.info(jedis_S.get("k6"));
    }
```

二、使用JedisPool

2.1 pom.xml

需要多引入下面依赖：

```
        <dependency>
            <groupId>org.apache.commons</groupId>
            <artifactId>commons-pool2</artifactId>
        </dependency>
```

fastjson依赖用于对象转换：

```
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>fastjson</artifactId>
            <version>1.2.68</version>
        </dependency>
```

2.2 application.yml

```
#redis
redis:
  host: 127.0.0.1
  port: 6379
  password: ahhs2019
  timeout: 2000
  poolMaxTotal: 10
  poolMaxIdle: 10
  poolMaxWait: 3
  db: 0
```

2.3 JedisConfig

```
package com.wl.redis.jedis.config;

/**
 * @author 王柳
 * @date 2020/5/19 11:34
 */

import lombok.Data;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.stereotype.Component;

/**
 * Redis的配置类
 */
@Component
@ConfigurationProperties(prefix = "redis")
@Data
```

```
public class JedisConfig {
    private String host;
    private int port;
    private String password;
    private int timeout;
    private int poolMaxTotal;
    private int poolMaxIdle;
    private int poolMaxWait;
    private int db;
}
```

2.4 JedisPoolFactory

```
package com.wl.redis.jedis.config;

import org.springframework.context.annotation.Bean;
import org.springframework.stereotype.Component;
import redis.clients.jedis.JedisPool;
import redis.clients.jedis.JedisPoolConfig;

import javax.annotation.Resource;

/**
 * 创建edisPool对象
 */
@Component
public class JedisPoolFactory {

    @Resource
    private JedisConfig jedisConfig;

    @Bean
    public JedisPool getJedisPool() {
        JedisPoolConfig jedisPoolConfig = new JedisPoolConfig();
        // 控制一个pool最多有多少个状态为idle(空闲)的jedis实例；
        jedisPoolConfig.setMaxIdle(jedisConfig.getPoolMaxIdle());
        // 表示当borrow一个jedis实例时，最大的等待时间，如果超过等待时间，则直接抛
JedisConnectionException；
        jedisPoolConfig.setMaxWaitMillis(jedisConfig.getPoolMaxWait());
        // 在指定时刻通过pool能够获取到的最大的连接的jedis个数
        jedisPoolConfig.setMaxTotal(jedisConfig.getPoolMaxTotal());
        JedisPool jp = new JedisPool(jedisPoolConfig, jedisConfig.getHost(),
jedisConfig.getPort(),
                jedisConfig.getTimeout(), jedisConfig.getPassword(),
jedisConfig.getDb());
        return jp;
    }
}
```

2.5 JedisService

```
package com.wl.redis.jedis.service;

import com.alibaba.fastjson.JSON;
import org.springframework.stereotype.Service;
import redis.clients.jedis.Jedis;
import redis.clients.jedis.JedisPool;

import javax.annotation.Resource;

/**
 * @author 王柳
 * @date 2020/5/19 11:38
 */
@Service
public class JedisService {

    @Resource
    JedisPool jedisPool;

    /**
     * 获取对象
     *
     * @param key
     * @param clazz
     * @param <T>
     * @return
     */
    public <T> T get(String key, Class<T> clazz) {
        Jedis jedis = null;
        try {
            jedis = jedisPool.getResource();
            String str = jedis.get(key);
            T t = stringToBean(str, clazz);
            return t;
        } finally {
            closeJedis(jedis);
        }
    }

    /**
     * 设置对象
     *
     * @param key
     * @param value
     * @param <T>
     * @return
     */
    public <T> boolean set(String key, T value) {
        Jedis jedis = null;
        try {
            jedis = jedisPool.getResource();
            String str = beanToString(value);
            if (str == null || str.length() <= 0) {
```

```
                return false;
            }
            jedis.set(key, str);
            return true;
        } finally {
            closeJedis(jedis);
        }
    }

    /**
     * 删除
     *
     * @param key
     * @return
     */
    public boolean delete(String key) {
        Jedis jedis = null;
        try {
            jedis = jedisPool.getResource();
            long ret = jedis.del(key);
            return ret > 0;
        } finally {
            closeJedis(jedis);
        }
    }

    /**
     * 判断key是否存在
     *
     * @param key
     * @param <T>
     * @return
     */
    public <T> boolean exists(String key) {
        Jedis jedis = null;
        try {
            jedis = jedisPool.getResource();
            return jedis.exists(key);
        } finally {
            closeJedis(jedis);
        }
    }

    /**
     * 增加值
     *
     * @param key
     * @param <T>
     * @return
     */
    public <T> Long incr(String key) {
        Jedis jedis = null;
        try {
            jedis = jedisPool.getResource();
            return jedis.incr(key);
        } finally {
            closeJedis(jedis);
        }
```

```
    }

    /**
     * 减少值
     *
     * @param key
     * @param <T>
     * @return
     */
    public <T> Long decr(String key) {
        Jedis jedis = null;
        try {
            jedis = jedisPool.getResource();
            return jedis.decr(key);
        } finally {
            closeJedis(jedis);
        }
    }

    /**
     * 关闭jedis
     *
     * @param jedis
     */
    private void closeJedis(Jedis jedis) {
        if (jedis != null) {
            jedis.close();
        }
    }

    /**
     * 将String转换为其他对象
     *
     * @param str
     * @param clazz
     * @param <T>
     * @return
     */
    private <T> T stringToBean(String str, Class<T> clazz) {
        if (str == null || str.length() <= 0 || clazz == null) {
            return null;
        }
        if (clazz == int.class || clazz == Integer.class) {
            return (T) Integer.valueOf(str);
        } else if (clazz == String.class) {
            return (T) str;
        } else if (clazz == long.class || clazz == Long.class) {
            return (T) Long.valueOf(str);
        } else {
            return JSON.toJavaObject(JSON.parseObject(str), clazz);
        }
    }

    /**
     * 将其他对象转换为String
     *
     * @param value
     * @param <T>
```

```
     * @return
     */
    private <T> String beanToString(T value) {
        if (value == null) {
            return null;
        }
        Class<?> clazz = value.getClass();
        if (clazz == int.class || clazz == Integer.class) {
            return "" + value;
        } else if (clazz == String.class) {
            return (String) value;
        } else if (clazz == long.class || clazz == Long.class) {
            return "" + value;
        } else {
            return JSON.toJSONString(value);
        }


    }
}
```

2.6 测试

```
package com.wl.redis.jedis.jedisPool;

import com.wl.redis.jedis.JedisApplicationTests;
import com.wl.redis.jedis.service.JedisService;
import lombok.extern.slf4j.Slf4j;
import org.junit.Test;
import org.springframework.beans.factory.annotation.Autowired;
import redis.clients.jedis.Jedis;
import redis.clients.jedis.Response;
import redis.clients.jedis.Transaction;

import java.util.*;

/**
 * @author 王柳
 * @date 2020/5/18 17:12
 */
@Slf4j
public class TestJedisPool extends JedisApplicationTests {

    @Autowired
    JedisService jedisService;

    @Test
    public void Test01() {
        boolean isSet = jedisService.set("girlfriend", "sakura");
        log.info("girlfriend============>set: " + isSet);

    }
```

```
    @Test
    public void Test02() {
        String str = jedisService.get("girlfriend", String.class);
        log.info("girlfriend============>get: " + str);
    }
}
```

2.7 jedisPoolConfig部分配置

资源数控制参数:

参数名	含义	默认值	使用建议
maxTotal	资源池最大连接数	8	
maxIdle	资源池允许最大空闲连接数	8	建议=maxTotal
minIdle	资源池确保最少空闲连接数	0	预热minIdle
jmxEnabled	是否开启jmx监控，可用于监控	true	建议开启

借还参数:

参数名	含义	默认值	使用建议
blockWhenExhausted	当资源池用尽后，调用者是否要等待。只有当为true时，下面的maxWaitMillis才会生效	true	建议使用默认值
maxWaitMillis	当资源池连接用尽后，调用者的最大等待时间（单位毫秒）	-1：表示永不超时	不建议使用默认值
testOnBorrow	向资源池借用连接时是否做连接有效性检查(ping)，无效连接会被移除	false	建议false
testOnReturn	向资源池归还连接时是否做连接有效性测试(ping)，无效连接会被移除	false	建议false

三、使用RedisTemplate

3.1 pom.xml

需要多引入下面依赖：

```
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-redis</artifactId>
            <exclusions>
                <exclusion>
                    <groupId>io.lettuce</groupId>
                    <artifactId>lettuce-core</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
```

3.2 application.yml

```
spring:
  redis:
    host: 127.0.0.1
    port: 6379
    password: ahhs2019
    timeout: 2000
    jedis:
      pool:
        # 控制一个pool可分配多少个jedis实例，通过pool.getResource()来获取；如果赋值为-1，则表示不限制；如果pool已经分配了maxActive个jedis实例，则此时pool的状态为exhausted。
        max-active: 10
        # 控制一个pool最多有多少个状态为idle(空闲)的jedis实例；
        max-idle: 10
        # 控制一个pool最少有多少个状态为idle(空闲)的jedis实例
        min-idle: 1
        # 表示当borrow一个jedis实例时，最大的等待时间，如果超过等待时间，则直接抛JedisConnectionException；
        max-wait: 3
        # 表示idle object evitor两次扫描之间要sleep的毫秒数；
        time-between-eviction-runs: 30000

    database: 0
```

3.3 RedisConfig

```
package com.wl.redis.jedis.config;

import com.fasterxml.jackson.annotation.JsonAutoDetect;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
```

```
import com.fasterxml.jackson.annotation.PropertyAccessor;
import com.fasterxml.jackson.databind.ObjectMapper;
import
com.fasterxml.jackson.databind.jsontype.impl.LaissezFaireSubTypeValidator;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.redis.connection.RedisConnectionFactory;
import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.data.redis.core.StringRedisTemplate;
import org.springframework.data.redis.serializer.Jackson2JsonRedisSerializer;
import org.springframework.data.redis.serializer.StringRedisSerializer;

/**
 * StringRedisTemplate与RedisTemplate区别点
 * 两者的关系是StringRedisTemplate继承RedisTemplate。
 * <p>
 * 两者的数据是不共通的；也就是说StringRedisTemplate只能管理StringRedisTemplate里面的数
据，RedisTemplate只能管理RedisTemplate中的数据。
 * <p>
 * 其实他们两者之间的区别主要在于他们使用的序列化类:
 *       RedisTemplate使用的是JdkSerializationRedisSerializer     存入数据会将数据先
序列化成字节数组然后在存入Redis数据库。
 * <p>
 *         StringRedisTemplate使用的是StringRedisSerializer
 * <p>
 * 使用时注意事项:
 *       当你的redis数据库里面本来存的是字符串数据或者你要存取的数据就是字符串类型数据的时候，
那么你就使用StringRedisTemplate即可。
 *       但是如果你的数据是复杂的对象类型，而取出的时候又不想做任何的数据转换，直接从Redis里面
取出一个对象，那么使用RedisTemplate是更好的选择。
 */
@Configuration
public class RedisConfig {

    /**
     * RedisTemplate配置
     */
    @Bean
    public RedisTemplate<String, Object> redisTemplate(RedisConnectionFactory
factory) {
        RedisTemplate<String, Object> template = new RedisTemplate<>();

        Jackson2JsonRedisSerializer<Object> jackson2JsonRedisSerializer = new
Jackson2JsonRedisSerializer<>(Object.class);

        ObjectMapper om = new ObjectMapper();
        om.setVisibility(PropertyAccessor.ALL, JsonAutoDetect.Visibility.ANY);
        // 方法enableDefaultTyping过期，使用activateDefaultTyping替换
//        om.enableDefaultTyping(ObjectMapper.DefaultTyping.NON_FINAL);
        om.activateDefaultTyping(LaissezFaireSubTypeValidator.instance,
                ObjectMapper.DefaultTyping.NON_FINAL, JsonTypeInfo.As.PROPERTY);
        jackson2JsonRedisSerializer.setObjectMapper(om);

        template.setConnectionFactory(factory);
        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(jackson2JsonRedisSerializer);
        // hash的value序列化方式采用jackson
        template.setHashValueSerializer(jackson2JsonRedisSerializer);
        template.afterPropertiesSet();
        return template;
    }

    @Bean
    public StringRedisTemplate stringRedisTemplate(RedisConnectionFactory
factory) {
        StringRedisTemplate stringRedisTemplate = new StringRedisTemplate();
        stringRedisTemplate.setConnectionFactory(factory);
        return stringRedisTemplate;
    }

}
```

3.4 RedisUtil

```
package com.wl.redis.jedis.util;

import org.springframework.data.redis.connection.DataType;
import org.springframework.data.redis.core.Cursor;
import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.data.redis.core.ScanOptions;
import org.springframework.data.redis.core.ZSetOperations;
import org.springframework.stereotype.Component;

import javax.annotation.Resource;
import java.util.*;
import java.util.concurrent.TimeUnit;

/**
 * @author 王柳
 * @date 2020/5/18 13:52
 */
@Component
public class RedisUtil {


    @Resource
    private RedisTemplate<String, Object> redisTemplate;


    /**
     * --------------------key相关操作---------------------
     */
```

```
    /**
     * 删除key
     *
     * @param key
     */
    public void delete(String key) {
        redisTemplate.delete(key);
    }

    /**
     * 批量删除key
     *
     * @param keys
     */
    public void delete(Collection<String> keys) {
        redisTemplate.delete(keys);
    }

    /**
     * 序列化key
     *
     * @param key
     * @return
     */
    public byte[] dump(String key) {
        return redisTemplate.dump(key);
    }

    /**
     * 是否存在key
     *
     * @param key
     * @return
     */
    public Boolean hasKey(String key) {
        return redisTemplate.hasKey(key);
    }

    /**
     * 设置过期时间
     *
     * @param key
     * @param timeout
     * @param unit
     * @return
     */
    public Boolean expire(String key, long timeout, TimeUnit unit) {
        return redisTemplate.expire(key, timeout, unit);
    }

    /**
     * 设置过期时间
     *
     * @param key
     * @param date
     * @return
     */
    public Boolean expireAt(String key, Date date) {
```

```
        return redisTemplate.expireAt(key, date);
    }

    /**
     * 查找匹配的key
     *
     * @param pattern
     * @return
     */
    public Set<String> keys(String pattern) {
        return redisTemplate.keys(pattern);
    }

    /**
     * 将当前数据库的 key
     * 移动到给定的数据库 db
     * 当中
     *
     * @param key
     * @param dbIndex
     * @return
     */
    public Boolean move(String key, int dbIndex) {
        return redisTemplate.move(key, dbIndex);
    }

    /**
     * 移除 key
     * 的过期时间，
     * key 将持久保持
     *
     * @param key
     * @return
     */
    public Boolean persist(String key) {
        return redisTemplate.persist(key);
    }

    /**
     * 返回 key
     * 的剩余的过期时间
     *
     * @param key
     * @param unit
     * @return
     */
    public Long getExpire(String key, TimeUnit unit) {
        return redisTemplate.getExpire(key, unit);
    }

    /**
     * 返回 key
     * 的剩余的过期时间
     *
     * @param key
     * @return
     */
    public Long getExpire(String key) {
```

```
        return redisTemplate.getExpire(key);
    }

    /**
     * 从当前数据库中随机返回一个 key
     *
     * @return
     */
    public String randomKey() {
        return redisTemplate.randomKey();
    }

    /**
     * 修改 key
     * 的名称
     *
     * @param oldKey
     * @param newKey
     */
    public void rename(String oldKey, String newKey) {
        redisTemplate.rename(oldKey, newKey);
    }

    /**
     * 仅当 newkey
     * 不存在时，
     * 将 oldKey
     * 改名为 newkey
     *
     * @param oldKey
     * @param newKey
     * @return
     */
    public Boolean renameIfAbsent(String oldKey, String newKey) {
        return redisTemplate.renameIfAbsent(oldKey, newKey);
    }

    /**
     * 返回 key
     * 所储存的值的类型
     *
     * @param key
     * @return
     */
    public DataType type(String key) {
        return redisTemplate.type(key);
    }

    /** -------------------string相关操作---------------------- */

    /**
     * 设置指定 key
     * 的值
     *
     * @param key
     * @param value
     */
    public void set(String key, String value) {
```

```
        redisTemplate.opsForValue().set(key, value);
    }

    /**
     * 获取指定 key
     * 的值
     *
     * @param key
     * @return
     */
    public Object get(String key) {
        return redisTemplate.opsForValue().get(key);
    }

    /**
     * 返回 key
     * 中字符串值的子字符
     *
     * @param key
     * @param start
     * @param end
     * @return
     */
    public String getRange(String key, long start, long end) {
        return redisTemplate.opsForValue().get(key, start, end);
    }

    /**
     * 将给定 key
     * 的值设为 value ，
     * 并返回 key
     * <p>
     * 的旧值(old value)
     *
     * @param key
     * @param value
     * @return
     */
    public Object getAndSet(String key, String value) {
        return redisTemplate.opsForValue().getAndSet(key, value);
    }

    /**
     * 对 key
     * 所储存的字符串值，
     * <p>
     * 获取指定偏移量上的位(bit)
     *
     * @param key
     * @param offset
     * @return
     */
    public Boolean getBit(String key, long offset) {
        return redisTemplate.opsForValue().getBit(key, offset);
    }

    /**
     * 批量获取
```

```
     *
     * @param keys
     * @return
     */
    public List<Object> multiGet(Collection<String> keys) {
        return redisTemplate.opsForValue().multiGet(keys);
    }

    /**
     * 设置ASCII码,字符串'a'的ASCII码是97,转为二进制是'01100001',此方法是将二进制第
offset位值变为value
     *
     * @param key   位置
     * @param value 值，true为1，false为0
     * @return
     */
    public boolean setBit(String key, long offset, boolean value) {
        return redisTemplate.opsForValue().setBit(key, offset, value);
    }

    /**
     * 将值 value
     * 关联到 key ，
     * 并将 key
     * 的过期时间设为 timeout
     *
     * @param key
     * @param value
     * @param timeout 过期时间
     * @param unit    时间单位，天:
     *                TimeUnit.DAYS 小时:
     *                TimeUnit.HOURS 分钟:TimeUnit.MINUTES
     *                秒:
     *                TimeUnit.SECONDS 毫秒:TimeUnit.MILLISECONDS
     */
    public void setEx(String key, String value, long timeout, TimeUnit unit) {
        redisTemplate.opsForValue().set(key, value, timeout, unit);
    }

    /**
     * 只有在 key
     * 不存在时设置 key
     * 的值
     *
     * @param key
     * @param value
     * @return 之前已经存在返回false，不存在返回true
     */
    public boolean setIfAbsent(String key, String value) {
        return redisTemplate.opsForValue().setIfAbsent(key, value);
    }

    /**
     * 用 value
     * 参数覆写给定 key
     * 所储存的字符串值，
     * 从偏移量 offset
     * 开始
```

```
     *
     * @param key
     * @param value
     * @param offset 从指定位置开始覆写
     */
    public void setRange(String key, String value, long offset) {
        redisTemplate.opsForValue().set(key, value, offset);
    }

    /**
     * 获取字符串的长度
     *
     * @param key
     * @return
     */
    public Long size(String key) {
        return redisTemplate.opsForValue().size(key);
    }

    /**
     * 批量添加
     *
     * @param maps
     */
    public void multiSet(Map<String, String> maps) {
        redisTemplate.opsForValue().multiSet(maps);
    }

    /**
     * 同时设置一个或多个 key-
     * value 对，
     * 当且仅当所有给定 key
     * 都不存在
     *
     * @param maps
     * @return 之前已经存在返回false, 不存在返回true
     */
    public boolean multiSetIfAbsent(Map<String, String> maps) {
        return redisTemplate.opsForValue().multiSetIfAbsent(maps);
    }

    /**
     * 增加(自增长),负数则为自减
     *
     * @param key
     * @return
     */
    public Long incrBy(String key, long increment) {
        return redisTemplate.opsForValue().increment(key, increment);
    }

    /**
     * 增加(自增长), 负数则为自减
     *
     * @param key
     * @return
     */
    public Double incrByFloat(String key, double increment) {
```

```
        return redisTemplate.opsForValue().increment(key, increment);
    }

    /**
     * 追加到末尾
     *
     * @param key
     * @param value
     * @return
     */
    public Integer append(String key, String value) {
        return redisTemplate.opsForValue().append(key, value);
    }

    /** -------------------hash相关操作------------------------- */

    /**
     * 获取存储在哈希表中指定字段的值
     *
     * @param key
     * @param field
     * @return
     */
    public Object hGet(String key, String field) {
        return redisTemplate.opsForHash().get(key, field);
    }

    /**
     * 获取所有给定字段的值
     *
     * @param key
     * @return
     */
    public Map<Object, Object> hGetAll(String key) {
        return redisTemplate.opsForHash().entries(key);
    }

    /**
     * 获取所有给定字段的值
     *
     * @param key
     * @param fields
     * @return
     */
    public List<Object> hMultiGet(String key, Collection<Object> fields) {
        return redisTemplate.opsForHash().multiGet(key, fields);
    }

    /**
     * 添加字段
     *
     * @param key
     * @param hashKey
     * @param value
     */
    public void hPut(String key, String hashKey, String value) {
        redisTemplate.opsForHash().put(key, hashKey, value);
    }
```

```java
/**
 * 添加多个字段
 *
 * @param key
 * @param maps
 */
public void hPutAll(String key, Map<String, String> maps) {
    redisTemplate.opsForHash().putAll(key, maps);
}

/**
 * 仅当hashKey不存在时才设置
 *
 * @param key
 * @param hashKey
 * @param value
 * @return
 */
public Boolean hPutIfAbsent(String key, String hashKey, String value) {
    return redisTemplate.opsForHash().putIfAbsent(key, hashKey, value);
}

/**
 * 删除一个或多个哈希表字段
 *
 * @param key
 * @param fields
 * @return
 */
public Long hDelete(String key, Object... fields) {
    return redisTemplate.opsForHash().delete(key, fields);
}

/**
 * 查看哈希表 key
 * 中，指定的字段是否存在
 *
 * @param key
 * @param field
 * @return
 */
public boolean hExists(String key, String field) {
    return redisTemplate.opsForHash().hasKey(key, field);
}

/**
 * 为哈希表 key
 * 中的指定字段的整数值加上增量 increment
 *
 * @param key
 * @param field
 * @param increment
 * @return
 */
public Long hIncrBy(String key, Object field, long increment) {
    return redisTemplate.opsForHash().increment(key, field, increment);
}
```

```
    /**
     * 为哈希表 key
     * 中的指定字段的整数值加上增量 increment
     *
     * @param key
     * @param field
     * @param delta
     * @return
     */
    public Double hIncrByFloat(String key, Object field, double delta) {
        return redisTemplate.opsForHash().increment(key, field, delta);
    }

    /**
     * 获取所有哈希表中的字段
     *
     * @param key
     * @return
     */
    public Set<Object> hKeys(String key) {
        return redisTemplate.opsForHash().keys(key);
    }

    /**
     * 获取哈希表中字段的数量
     *
     * @param key
     * @return
     */
    public Long hSize(String key) {
        return redisTemplate.opsForHash().size(key);
    }

    /**
     * 获取哈希表中所有值
     *
     * @param key
     * @return
     */
    public List<Object> hValues(String key) {
        return redisTemplate.opsForHash().values(key);
    }

    /**
     * 迭代哈希表中的键值对
     *
     * @param key
     * @param options
     * @return
     */
    public Cursor<Map.Entry<Object, Object>> hScan(String key, ScanOptions
options) {
        return redisTemplate.opsForHash().scan(key, options);
    }

    /** ------------------------list相关操作---------------------------- */
```

```java
    /**
     * 通过索引获取列表中的元素
     *
     * @param key
     * @param index
     * @return
     */
    public Object lIndex(String key, long index) {
        return redisTemplate.opsForList().index(key, index);
    }

    /**
     * 获取列表指定范围内的元素
     *
     * @param key
     * @param start 开始位置, 0是开始位置
     * @param end   结束位置, -1返回所有
     * @return
     */
    public List<Object> lRange(String key, long start, long end) {
        return redisTemplate.opsForList().range(key, start, end);
    }

    /**
     * 存储在list头部
     *
     * @param key
     * @param value
     * @return
     */
    public Long lLeftPush(String key, String value) {
        return redisTemplate.opsForList().leftPush(key, value);
    }

    /**
     * 存储在list头部
     *
     * @param key
     * @param value
     * @return
     */
    public Long lLeftPushAll(String key, String... value) {
        return redisTemplate.opsForList().leftPushAll(key, value);
    }

    /**
     * 存储在list头部
     *
     * @param key
     * @param value
     * @return
     */
    public Long lLeftPushAll(String key, Collection<String> value) {
        return redisTemplate.opsForList().leftPushAll(key, value);
    }

    /**
     * 当list存在的时候才加入
```

```
     *
     * @param key
     * @param value
     * @return
     */
    public Long lLeftPushIfPresent(String key, String value) {
        return redisTemplate.opsForList().leftPushIfPresent(key, value);
    }

    /**
     * 如果pivot存在,再pivot前面添加
     *
     * @param key
     * @param pivot
     * @param value
     * @return
     */
    public Long lLeftPush(String key, String pivot, String value) {
        return redisTemplate.opsForList().leftPush(key, pivot, value);
    }

    /**
     * 存储在list尾部
     *
     * @param key
     * @param value
     * @return
     */
    public Long lRightPush(String key, String value) {
        return redisTemplate.opsForList().rightPush(key, value);
    }

    /**
     * 存储在list尾部
     *
     * @param key
     * @param value
     * @return
     */
    public Long lRightPushAll(String key, String... value) {
        return redisTemplate.opsForList().rightPushAll(key, value);
    }

    /**
     * 存储在list尾部
     *
     * @param key
     * @param value
     * @return
     */
    public Long lRightPushAll(String key, Collection<String> value) {
        return redisTemplate.opsForList().rightPushAll(key, value);
    }

    /**
     * 为已存在的列表添加值
     *
     * @param key
```

```
     * @param value
     * @return
     */
    public Long lRightPushIfPresent(String key, String value) {
        return redisTemplate.opsForList().rightPushIfPresent(key, value);
    }

    /**
     * 在pivot元素的右边添加值
     *
     * @param key
     * @param pivot
     * @param value
     * @return
     */
    public Long lRightPush(String key, String pivot, String value) {
        return redisTemplate.opsForList().rightPush(key, pivot, value);
    }

    /**
     * 通过索引设置列表元素的值
     *
     * @param key
     * @param index 位置
     * @param value
     */
    public void lSet(String key, long index, String value) {
        redisTemplate.opsForList().set(key, index, value);
    }

    /**
     * 移出并获取列表的第一个元素
     *
     * @param key
     * @return 删除的元素
     */
    public Object lLeftPop(String key) {
        return redisTemplate.opsForList().leftPop(key);
    }

    /**
     * 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
     *
     * @param key
     * @param timeout 等待时间
     * @param unit    时间单位
     * @return
     */
    public Object lBLeftPop(String key, long timeout, TimeUnit unit) {
        return redisTemplate.opsForList().leftPop(key, timeout, unit);
    }

    /**
     * 移除并获取列表最后一个元素
     *
     * @param key
     * @return 删除的元素
     */
```

```
    public Object lRightPop(String key) {
        return redisTemplate.opsForList().rightPop(key);
    }

    /**
     * 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
     *
     * @param key
     * @param timeout 等待时间
     * @param unit    时间单位
     * @return
     */
    public Object lBRightPop(String key, long timeout, TimeUnit unit) {
        return redisTemplate.opsForList().rightPop(key, timeout, unit);
    }

    /**
     * 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
     *
     * @param sourceKey
     * @param destinationKey
     * @return
     */
    public Object lRightPopAndLeftPush(String sourceKey, String destinationKey)
{
        return redisTemplate.opsForList().rightPopAndLeftPush(sourceKey,
                destinationKey);
    }

    /**
     * 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
     *
     * @param sourceKey
     * @param destinationKey
     * @param timeout
     * @param unit
     * @return
     */
    public Object lBRightPopAndLeftPush(String sourceKey, String destinationKey,
                                        long timeout, TimeUnit unit) {
        return redisTemplate.opsForList().rightPopAndLeftPush(sourceKey,
                destinationKey, timeout, unit);
    }

    /**
     * 删除集合中值等于value得元素
     *
     * @param key
     * @param index index = 0, 删除所有值等于value的元素;
     *              index>0,从头部开始删除第一个值等于value的元素;
     *              index<0,从尾部开始删除第一个值等于value的元素;
     * @param value
     * @return
     */
    public Long lRemove(String key, long index, String value) {
        return redisTemplate.opsForList().remove(key, index, value);
    }
```

```
    /**
     * 裁剪list
     *
     * @param key
     * @param start
     * @param end
     */
    public void lTrim(String key, long start, long end) {
        redisTemplate.opsForList().trim(key, start, end);
    }

    /**
     * 获取列表长度
     *
     * @param key
     * @return
     */
    public Long lLen(String key) {
        return redisTemplate.opsForList().size(key);
    }

    /** --------------------set相关操作-------------------------- */

    /**
     * set添加元素
     *
     * @param key
     * @param values
     * @return
     */
    public Long sAdd(String key, String... values) {
        return redisTemplate.opsForSet().add(key, values);
    }

    /**
     * set移除元素
     *
     * @param key
     * @param values
     * @return
     */
    public Long sRemove(String key, Object... values) {
        return redisTemplate.opsForSet().remove(key, values);
    }

    /**
     * 移除并返回集合的一个随机元素
     *
     * @param key
     * @return
     */
    public Object sPop(String key) {
        return redisTemplate.opsForSet().pop(key);
    }

    /**
     * 将元素value从一个集合移到另一个集合
```

```
     *
     * @param key
     * @param value
     * @param destKey
     * @return
     */
    public Boolean sMove(String key, String value, String destKey) {
        return redisTemplate.opsForSet().move(key, value, destKey);
    }

    /**
     * 获取集合的大小
     *
     * @param key
     * @return
     */
    public Long sSize(String key) {
        return redisTemplate.opsForSet().size(key);
    }

    /**
     * 判断集合是否包含value
     *
     * @param key
     * @param value
     * @return
     */
    public Boolean sIsMember(String key, Object value) {
        return redisTemplate.opsForSet().isMember(key, value);
    }

    /**
     * 获取两个集合的交集
     *
     * @param key
     * @param otherKey
     * @return
     */
    public Set<Object> sIntersect(String key, String otherKey) {
        return redisTemplate.opsForSet().intersect(key, otherKey);
    }

    /**
     * 获取key集合与多个集合的交集
     *
     * @param key
     * @param otherKeys
     * @return
     */
    public Set<Object> sIntersect(String key, Collection<String> otherKeys) {
        return redisTemplate.opsForSet().intersect(key, otherKeys);
    }

    /**
     * key集合与otherKey集合的交集存储到destKey集合中
     *
     * @param key
     * @param otherKey
```

```
     * @param destKey
     * @return
     */
    public Long sIntersectAndStore(String key, String otherKey, String destKey)
{
        return redisTemplate.opsForSet().intersectAndStore(key, otherKey,
                destKey);
    }

    /**
     * key集合与多个集合的交集存储到destKey集合中
     *
     * @param key
     * @param otherKeys
     * @param destKey
     * @return
     */
    public Long sIntersectAndStore(String key, Collection<String> otherKeys,
                                   String destKey) {
        return redisTemplate.opsForSet().intersectAndStore(key, otherKeys,
                destKey);
    }

    /**
     * 获取两个集合的并集
     *
     * @param key
     * @param otherKeys
     * @return
     */
    public Set<Object> sUnion(String key, String otherKeys) {
        return redisTemplate.opsForSet().union(key, otherKeys);
    }

    /**
     * 获取key集合与多个集合的并集
     *
     * @param key
     * @param otherKeys
     * @return
     */
    public Set<Object> sUnion(String key, Collection<String> otherKeys) {
        return redisTemplate.opsForSet().union(key, otherKeys);
    }

    /**
     * key集合与otherKey集合的并集存储到destKey中
     *
     * @param key
     * @param otherKey
     * @param destKey
     * @return
     */
    public Long sUnionAndStore(String key, String otherKey, String destKey) {
        return redisTemplate.opsForSet().unionAndStore(key, otherKey, destKey);
    }

    /**
```

```
     * key集合与多个集合的并集存储到destKey中
     *
     * @param key
     * @param otherKeys
     * @param destKey
     * @return
     */
    public Long sUnionAndStore(String key, Collection<String> otherKeys,
                               String destKey) {
        return redisTemplate.opsForSet().unionAndStore(key, otherKeys, destKey);
    }

    /**
     * 获取两个集合的差集
     *
     * @param key
     * @param otherKey
     * @return
     */
    public Set<Object> sDifference(String key, String otherKey) {
        return redisTemplate.opsForSet().difference(key, otherKey);
    }

    /**
     * 获取key集合与多个集合的差集
     *
     * @param key
     * @param otherKeys
     * @return
     */
    public Set<Object> sDifference(String key, Collection<String> otherKeys) {
        return redisTemplate.opsForSet().difference(key, otherKeys);
    }

    /**
     * key集合与otherKey集合的差集存储到destKey中
     *
     * @param key
     * @param otherKey
     * @param destKey
     * @return
     */
    public Long sDifference(String key, String otherKey, String destKey) {
        return redisTemplate.opsForSet().differenceAndStore(key, otherKey,
                destKey);
    }

    /**
     * key集合与多个集合的差集存储到destKey中
     *
     * @param key
     * @param otherKeys
     * @param destKey
     * @return
     */
    public Long sDifference(String key, Collection<String> otherKeys,
                            String destKey) {
        return redisTemplate.opsForSet().differenceAndStore(key, otherKeys,
```

```
                destKey);
    }

    /**
     * 获取集合所有元素
     *
     * @param key
     * @return
     */
    public Set<Object> setMembers(String key) {
        return redisTemplate.opsForSet().members(key);
    }

    /**
     * 随机获取集合中的一个元素
     *
     * @param key
     * @return
     */
    public Object sRandomMember(String key) {
        return redisTemplate.opsForSet().randomMember(key);
    }

    /**
     * 随机获取集合中count个元素
     *
     * @param key
     * @param count
     * @return
     */
    public List<Object> sRandomMembers(String key, long count) {
        return redisTemplate.opsForSet().randomMembers(key, count);
    }

    /**
     * 随机获取集合中count个元素并且去除重复的
     *
     * @param key
     * @param count
     * @return
     */
    public Set<Object> sDistinctRandomMembers(String key, long count) {
        return redisTemplate.opsForSet().distinctRandomMembers(key, count);
    }

    /**
     * 使用迭代器获取元素
     *
     * @param key
     * @param options
     * @return
     */
    public Cursor<Object> sScan(String key, ScanOptions options) {
        return redisTemplate.opsForSet().scan(key, options);
    }

    /**------------------zSet相关操作--------------------------------*/
```

```
    /**
     * 添加元素,有序集合是按照元素的score值由小到大排列
     *
     * @param key
     * @param value
     * @param score
     * @return
     */
    public Boolean zAdd(String key, String value, double score) {
        return redisTemplate.opsForZSet().add(key, value, score);
    }

    /**
     * 批量添加
     *
     * @param key
     * @param values
     * @return
     */
    public Long zAdd(String key, Set<ZSetOperations.TypedTuple<Object>> values)
{
        return redisTemplate.opsForZSet().add(key, values);
    }

    /**
     * 移除
     *
     * @param key
     * @param values
     * @return
     */
    public Long zRemove(String key, Object... values) {
        return redisTemplate.opsForZSet().remove(key, values);
    }

    /**
     * 增加元素的score值，并返回增加后的值
     *
     * @param key
     * @param value
     * @param delta
     * @return
     */
    public Double zIncrementScore(String key, String value, double delta) {
        return redisTemplate.opsForZSet().incrementScore(key, value, delta);
    }

    /**
     * 返回元素在集合的排名,有序集合是按照元素的score值由小到大排列
     *
     * @param key
     * @param value
     * @return 0表示第一位
     */
    public Long zRank(String key, Object value) {
        return redisTemplate.opsForZSet().rank(key, value);
    }
```

```
    /**
     * 返回元素在集合的排名,按元素的score值由大到小排列
     *
     * @param key
     * @param value
     * @return
     */
    public Long zReverseRank(String key, Object value) {
        return redisTemplate.opsForZSet().reverseRank(key, value);
    }

    /**
     * 获取集合的元素,从小到大排序
     *
     * @param key
     * @param start 开始位置
     * @param end   结束位置, -1查询所有
     * @return
     */
    public Set<Object> zRange(String key, long start, long end) {
        return redisTemplate.opsForZSet().range(key, start, end);
    }

    /**
     * 获取集合元素,并且把score值也获取
     *
     * @param key
     * @param start
     * @param end
     * @return
     */
    public Set<ZSetOperations.TypedTuple<Object>> zRangeWithScores(String key,
long start,
                                                                   long end) {
        return redisTemplate.opsForZSet().rangeWithScores(key, start, end);
    }

    /**
     * 根据Score值查询集合元素
     *
     * @param key
     * @param min 最小值
     * @param max 最大值
     * @return
     */
    public Set<Object> zRangeByScore(String key, double min, double max) {
        return redisTemplate.opsForZSet().rangeByScore(key, min, max);
    }

    /**
     * 根据Score值查询集合元素,从小到大排序
     *
     * @param key
     * @param min 最小值
     * @param max 最大值
     * @return
     */
```

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

    /**
     * 根据Score值查询集合元素, 从小到大排序
     *
     * @param key
     * @param min
     * @param max
     * @param start
     * @param end
     * @return
     */
    public Set<ZSetOperations.TypedTuple<Object>> zRangeByScoreWithScores(String
key,
                                                                           double
min, double max, long start, long end) {
        return redisTemplate.opsForZSet().rangeByScoreWithScores(key, min, max,
                start, end);
    }

    /**
     * 获取集合的元素,从大到小排序
     *
     * @param key
     * @param start
     * @param end
     * @return
     */
    public Set<Object> zReverseRange(String key, long start, long end) {
        return redisTemplate.opsForZSet().reverseRange(key, start, end);
    }

    /**
     * 获取集合的元素,从大到小排序,并返回score值
     *
     * @param key
     * @param start
     * @param end
     * @return
     */
    public Set<ZSetOperations.TypedTuple<Object>> zReverseRangeWithScores(String
key,
                                                                           long
start, long end) {
        return redisTemplate.opsForZSet().reverseRangeWithScores(key, start,
                end);
    }

    /**
     * 根据Score值查询集合元素,从大到小排序
     *
     * @param key
     * @param min
```

```
     * @param max
     * @return
     */
    public Set<Object> zReverseRangeByScore(String key, double min,
                                            double max) {
        return redisTemplate.opsForZSet().reverseRangeByScore(key, min, max);
    }

    /**
     * 根据Score值查询集合元素,从大到小排序
     *
     * @param key
     * @param min
     * @param max
     * @return
     */
    public Set<ZSetOperations.TypedTuple<Object>>
zReverseRangeByScoreWithScores(
            String key, double min, double max) {
        return redisTemplate.opsForZSet().reverseRangeByScoreWithScores(key,
                min, max);
    }

    /**
     * @param key
     * @param min
     * @param max
     * @param start
     * @param end
     * @return
     */
    public Set<Object> zReverseRangeByScore(String key, double min,
                                            double max, long start, long end) {
        return redisTemplate.opsForZSet().reverseRangeByScore(key, min, max,
                start, end);
    }

    /**
     * 根据score值获取集合元素数量
     *
     * @param key
     * @param min
     * @param max
     * @return
     */
    public Long zCount(String key, double min, double max) {
        return redisTemplate.opsForZSet().count(key, min, max);
    }

    /**
     * 获取集合大小
     *
     * @param key
     * @return
     */
    public Long zSize(String key) {
        return redisTemplate.opsForZSet().size(key);
    }
```

```java
/**
 * 获取集合大小
 *
 * @param key
 * @return
 */
public Long zZCard(String key) {
    return redisTemplate.opsForZSet().zCard(key);
}

/**
 * 获取集合中value元素的score值
 *
 * @param key
 * @param value
 * @return
 */
public Double zScore(String key, Object value) {
    return redisTemplate.opsForZSet().score(key, value);
}

/**
 * 移除指定索引位置的成员
 *
 * @param key
 * @param start
 * @param end
 * @return
 */
public Long zRemoveRange(String key, long start, long end) {
    return redisTemplate.opsForZSet().removeRange(key, start, end);
}

/**
 * 根据指定的score值的范围来移除成员
 *
 * @param key
 * @param min
 * @param max
 * @return
 */
public Long zRemoveRangeByScore(String key, double min, double max) {
    return redisTemplate.opsForZSet().removeRangeByScore(key, min, max);
}

/**
 * 获取key和otherKey的并集并存储在destKey中
 *
 * @param key
 * @param otherKey
 * @param destKey
 * @return
 */
public Long zUnionAndStore(String key, String otherKey, String destKey) {
    return redisTemplate.opsForZSet().unionAndStore(key, otherKey, destKey);
}
```

```
    /**
     * 获取key和多个集合的并集并存储在destKey中
     *
     * @param key
     * @param otherKeys
     * @param destKey
     * @return
     */
    public Long zUnionAndStore(String key, Collection<String> otherKeys,
                               String destKey) {
        return redisTemplate.opsForZSet()
                .unionAndStore(key, otherKeys, destKey);
    }

    /**
     * 交集 获取key和otherKey的交集并存储在destKey中
     *
     * @param key
     * @param otherKey
     * @param destKey
     * @return
     */
    public Long zIntersectAndStore(String key, String otherKey,
                                   String destKey) {
        return redisTemplate.opsForZSet().intersectAndStore(key, otherKey,
                destKey);
    }

    /**
     * 交集 获取key和多个集合的交集并存储在destKey中
     *
     * @param key
     * @param otherKeys
     * @param destKey
     * @return
     */
    public Long zIntersectAndStore(String key, Collection<String> otherKeys,
                                   String destKey) {
        return redisTemplate.opsForZSet().intersectAndStore(key, otherKeys,
                destKey);
    }

    /**
     * 使用迭代器获取
     *
     * @param key
     * @param options
     * @return
     */
    public Cursor<ZSetOperations.TypedTuple<Object>> zScan(String key,
ScanOptions options) {
        return redisTemplate.opsForZSet().scan(key, options);
    }
}
```

3.5 RedisService

```
package com.wl.redis.jedis.service;

import org.springframework.data.redis.core.StringRedisTemplate;
import org.springframework.stereotype.Service;

import javax.annotation.Resource;

/**
 * @author 王柳
 * @date 2020/5/19 12:20
 */
@Service
public class RedisService {

    @Resource
    private StringRedisTemplate stringRedisTemplate;

    public void set(String key, String value) {
        stringRedisTemplate.opsForValue().set(key, value);
    }

    public String get(String key) {
        return stringRedisTemplate.opsForValue().get(key);
    }
}
```

3.6 测试

```
package com.wl.redis.jedis.restTemplate;

import com.wl.redis.jedis.JedisApplicationTests;
import com.wl.redis.jedis.service.RedisService;
import lombok.extern.slf4j.Slf4j;
import org.junit.Test;
import org.springframework.beans.factory.annotation.Autowired;

/**
 * @author 王柳
 * @date 2020/5/18 17:12
 */
@Slf4j
public class TestRedisTemplate extends JedisApplicationTests {

    @Autowired
    RedisService redisService;

    @Test
    public void Test01() {
```

```
        redisService.set("boyfriend", "selenium");
        String str = redisService.get("boyfriend");
        log.info("girlfriend============>get: " + str);

    }

}
```