

# Redis + Redisson + Cache

## Redis

### 一、依赖

```
1 <!-- redis: 2.3.12.RELEASE & redis连接池: 2.6.2 -->
2 <dependency>
3     <groupId>org.springframework.boot</groupId>
4     <artifactId>spring-boot-starter-data-redis</artifactId>
5 </dependency>
6 <dependency>
7     <groupId>org.apache.commons</groupId>
8     <artifactId>commons-pool2</artifactId>
9 </dependency>
10
11 <!-- fastjson: 1.2.79 -->
12 <dependency>
13     <groupId>com.alibaba</groupId>
14     <artifactId>fastjson</artifactId>
15 </dependency>
16
17 <!-- test: 2.3.12.RELEASE -->
18 <dependency>
19     <groupId>org.springframework.boot</groupId>
20     <artifactId>spring-boot-starter-test</artifactId>
21     <scope>test</scope>
22 </dependency>
```

### 二、yml配置

```
1 # spring
2 spring:
3     redis:
4         timeout: 6000ms    # 连接超时时长
5         password:          # 密码（默认为空）
6         # lettuce连接池
7         # SpringBoot2.0默认采用Lettuce客户端来连接Redis服务端，替换为jedis使用jedis
8         # Lettuce基于Netty线程安全，Jedis直接连接redis server非线程安全
9         lettuce:
```

```

10     pool:
11         max-active: 1000 # 连接池最大连接数（使用负值表示没有限制）
12         max-idle: 10     # 连接池中的最大空闲连接
13         min-idle: 5      # 连接池中的最小空闲连接
14         max-wait: -1      # 连接池最大阻塞等待时间（使用负值表示没有限制）
15     # 单机配置
16     host: 10.207.0.169   # 默认port=6379, database=0
17     port: 6379
18     # 集群配置
19     # cluster:
20     #     nodes: # Redis的各个端口号
21     #         - 10.207.0.169:6379
22     #         - 10.207.0.169:6380
23     #     max-redirects: 3 # 获取失败 最大重定向次数
24
25
26 # 注解方式redisson初始化策略：只能有一个Y，其余可不写
27 annotation:
28     redisson:
29         strategy:
30             single: Y
31             cluster:
32             sentinel:

```

### 三、配置类

```

1  @Configuration
2  @AutoConfigureAfter(RedisAutoConfiguration.class) //根据yaml配置，自动选择注入
3  public class RedisConfig {
4
5      /**
6       * Jackson2JsonRedisSerializer & GenericJackson2JsonRedisSerializer
7       * 1.序列化带泛型的数据时，会以map的结构进行存储，反序列化是不能将map解析成对
8       * 2.使用Jackson2JsonRedisSerializer需要指明序列化的类Class，可以使用Object
9       * 3.使用GenericJacksonRedisSerializer比Jackson2JsonRedisSerializer效率低
10      * 4.Jackson2JsonRedisSerializer反序列化带泛型的数组类会报转换异常，解决办法
11      */
12      @Bean
13      public RedisTemplate<String, Object> redisCacheTemplate(LettuceConnectionFactory
redisConnectionFactory) {
14          RedisTemplate<String, Object> redisTemplate = new RedisTemplate<>()
15
16          redisTemplate.setKeySerializer(new StringRedisSerializer());

```

```
16         redisTemplate.setValueSerializer(new GenericJackson2JsonRedisSerial:
17         redisTemplate.setConnectionFactory(redisConnectionFactory);
18         return redisTemplate;
19     }
20 }
```

## 四、工具类

```
1  @Component
2  public class RedisService {
3
4      @Autowired
5      private RedisTemplate<String, Object> redisTemplate;
6
7      /**
8       * 设置指定 key 的值
9       */
10     public void set(String key, Object value) {
11         if (null == key || null == value) {
12             return;
13         }
14         redisTemplate.opsForValue().set(key, value);
15     }
16
17     /**
18      * 设置key 的值 并设置过期时间
19      */
20     public void set(String key, Object value, long time, TimeUnit unit) {
21         if (null == key || null == value || null == unit) {
22             return;
23         }
24         redisTemplate.opsForValue().set(key, value, time, unit);
25     }
26
27     /**
28      * 获取指定Key的Value。如果与该Key关联的Value不是string类型，Redis将抛出异常
29      * 因为GET命令只能用于获取string Value，如果该Key不存在，返回null
30      */
31     public Object get(String key) {
32         if (null == key) {
33             return null;
34         }
35     }
```

```

35         return redisTemplate.opsForValue().get(key);
36     }
37
38     /**
39      * 设置过期时间
40      */
41     public Boolean expire(String key, long timeout, TimeUnit unit) {
42         if (null == key || null == unit) {
43             return false;
44         }
45         return redisTemplate.expire(key, timeout, unit);
46     }
47 }

```

## 五、测试

```

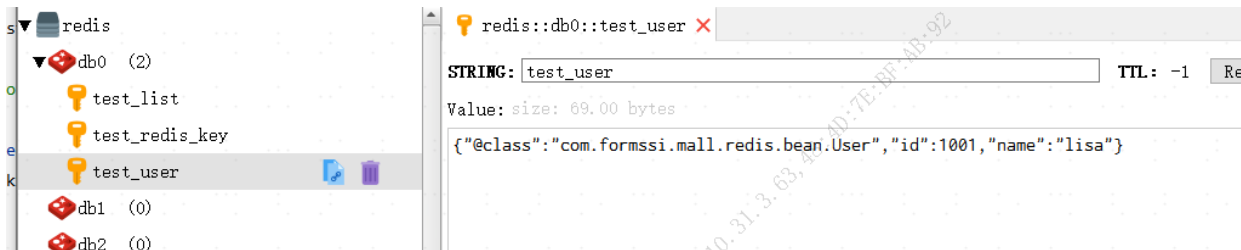
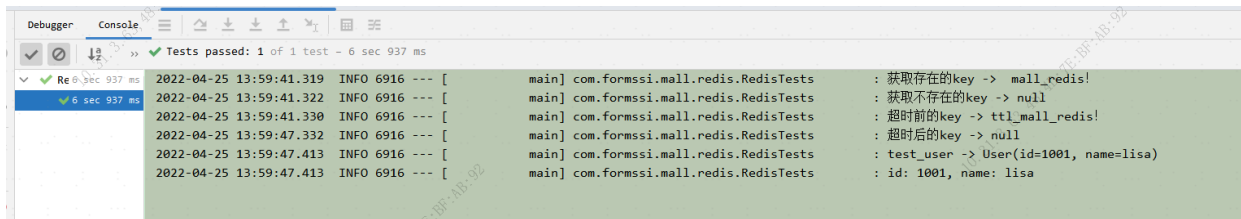
1  @Slf4j
2  @RunWith(SpringRunner.class)
3  @SpringBootTest
4  public class RedisTests {
5
6      @Autowired
7      private RedisService redisService;
8
9      @Test
10     public void testSet() throws Exception{
11         //set
12         redisService.set("test_redis_key", "mall_redis! ");
13         log.info("获取存在的key -> {}", redisService.get("test_redis_key"));
14         log.info("获取不存在的key -> {}", redisService.get("my_key"));
15
16         //set + ttl
17         redisService.set("test_redis_ttl_key", "ttl_mall_redis! ", 5, TimeUnit.SECONDS);
18         log.info("超时前的key -> {}", redisService.get("test_redis_ttl_key"));
19         TimeUnit.SECONDS.sleep(6);
20         log.info("超时后的key -> {}", redisService.get("test_redis_ttl_key"));
21
22         //object
23         User user = new User(1001, "lisa");
24         redisService.set("test_user", user);
25         //{"@class":"com.formssi.mall.redis.bean.User","id":1001,"name":"lisa"}
26
27         Object objUser = redisService.get("test_user");

```

```

26     log.info("test_user -> {}", objUser);
27     User user2 = (User)objUser;
28     log.info("id: {}, name: {}", user2.getId(), user2.getName());
29 }
30 }

```



## Redisson

### 定义

Redisson是一个高级的分布式协调Redis客户端，底层使用netty框架，并提供了与java对象相对应的分布式对象、分布式集合、分布式锁和同步器、分布式服务等一系列的Redisson的分布式对象。Redisson、Jedis、Lettuce 是三个不同的操作 Redis 的客户端，Jedis、Lettuce 的 API 更侧重对 Redis 数据库的 CRUD（增删改查），而 Redisson API 侧重于分布式开发

### 一、依赖

```

1 <!-- redisson -->
2 <dependency>
3     <groupId>org.redisson</groupId>
4     <artifactId>redisson</artifactId>
5     <version>3.17.0</version>
6 </dependency>

```

### 二、配置 - 基于工厂 - 方式一

#### 1.config三种策略：单机、集群、哨兵

```
1 @Getter
2 @AllArgsConstructor
3 public enum RedissonEnum {
4
5     //前缀
6     URL_PREFIX("redis://", "Redis地址配置前缀"),
7     SSL_URL_PREFIX("rediss://", "Redis地址配置SSL前缀"),
8
9     //策略
10    SINGLE_STRATEGY("SINGLE_STRATEGY", "单机模式"),
11    CLUSTER_STRATEGY("CLUSTER_STRATEGY", "集群模式"),
12    SENTINE_STRATEGY("SENTINE_STRATEGY", "哨兵模式");
13
14
15    private final String value;
16    private final String desc;
17 }
18
19 //定义策略接口
20 public interface RedissonConfigStrategy {
21     Config createRedissonConfig(RedisProperties redisProperties);
22 }
23
24 @Slf4j
25 public class SingleConfigImpl implements RedissonConfigStrategy {
26     // 单机模式
27     @Override
28     public Config createRedissonConfig(RedisProperties redisProperties) {
29         Config config = new Config();
30         try {
31             String address =
redisProperties.getHost().concat(":").concat(redisProperties.getPort()+"");
32             String password = redisProperties.getPassword();
33             int database = redisProperties.getDatabase();
34             String redisAddr = redisProperties.isSsl() ? RedissonEnum.SSL_URL
RedissonEnum.URL_PREFIX.getValue();
35             redisAddr += address;
36             config.useSingleServer().setAddress(redisAddr);
37             config.useSingleServer().setDatabase(database);
38             //密码可以为空
39             if (StringUtils.isNotBlank(password)) {
40                 config.useSingleServer().setPassword(password);
41             }
42         } catch (Exception e) {
43             log.error("RedissonConfigImpl createRedissonConfig error: {}", e);
44         }
45     }
46 }
```

```

42         log.info("部署策略[单机模式] -> redisAddress:{})", address);
43     } catch (Exception e) {
44         log.error("部署策略[单机模式] redisson init error!", e);
45         e.printStackTrace();
46     }
47     return config;
48 }
49 }
50
51 @Slf4j
52 public class ClusterConfigImpl implements RedissonConfigStrategy {
53     // 集群模式
54     @Override
55     public Config createRedissonConfig(RedisProperties redisProperties) {
56         Config config = new Config();
57         try {
58             List<String> address = redisProperties.getCluster().getNodes();
59             String password = redisProperties.getPassword();
60             //String[] addrTokens = address.split(",");
61             //设置cluster节点的服务IP和端口
62             String redisAddr = redisProperties.isSsl() ? RedissonEnum.SSL_URL_PREFIX.getValue() :
RedissonEnum.URL_PREFIX.getValue();
63             for (String address : address) {
64                 config.useClusterServers()
65                     .addNodeAddress(redisAddr + address);
66                 if (StringUtils.isNotBlank(password)) {
67                     config.useClusterServers().setPassword(password);
68                 }
69             }
70             log.info("部署策略[集群模式] -> redisAddress:{})", address);
71         } catch (Exception e) {
72             log.error("部署策略[集群模式] redisson init error!", e);
73             e.printStackTrace();
74         }
75         return config;
76     }
77 }
78
79 @Slf4j
80 public class SentinelConfigImpl implements RedissonConfigStrategy {
81     // 哨兵模式
82     @Override
83     public Config createRedissonConfig(RedisProperties redisProperties) {

```

```

84         Config config = new Config();
85         try {
86             String masterAddress = redisProperties.getSentinel().getMaster(
87                 String password = redisProperties.getPassword();
88                 int database = redisProperties.getDatabase();
89                 List<String> nodeAddress = redisProperties.getSentinel().getNode
90                 String sentinelAliasName = Optional.ofNullable(nodeAddress)
91                     .map(u-> nodeAddress.get(0))
92                     .orElseThrow(()->new
RuntimeException("redis.sentinel.nodes is null, please set"));
93                 //设置redis配置文件sentinel.conf配置的sentinel别名
94                 config.useSentinelServers().setMasterName(sentinelAliasName);
95                 config.useSentinelServers().setDatabase(database);
96                 if (StringUtils.isNotBlank(password)) {
97                     config.useSentinelServers().setPassword(password);
98                 }
99                 //设置sentinel节点的服务IP和端口
100                 String redisAddr = redisProperties.isSsl() ? RedissonEnum.SSL_URL
RedissonEnum.URL_PREFIX.getValue();
101                 for (String nodeAddress : nodeAddress) {
102                     config.useSentinelServers().addSentinelAddress(redisAddr +
103                 }
104                 log.info("部署策略[哨兵模式] -> masterAddress:{}, nodeAddress:{}",
nodeAddress);
105             } catch (Exception e) {
106                 log.error("部署策略[哨兵模式] redisson init error!", e);
107                 e.printStackTrace();
108             }
109             return config;
110         }
111     }
112
113     //策略工厂
114     @Component
115     public class RedissonStrategyFactory {
116
117         /**
118          * 基于策略模式，获取不同的连接
119          */
120         public Config createConfig(RedisProperties redisProperties, RedissonEnum
Objects.requireNonNull(redisProperties, "redisProperties 不能为空!")
121             if (strategyEnum == null) {
122                 return new SingleConfigImpl().createRedissonConfig(redisPropert:

```



```

124     }
125     //默认单机
126     RedissonConfigStrategy redissonConfigStrategy;
127     switch (strategyEnum){
128         case CLUSTER_STRATEGY:
129             redissonConfigStrategy = new ClusterConfigImpl();
130             break;
131         case SENTINE_STRATEGY:
132             redissonConfigStrategy = new SentineConfigImpl();
133             break;
134         default:
135             redissonConfigStrategy = new SingleConfigImpl();
136             break;
137     }
138     return redissonConfigStrategy.createRedissonConfig(redisProperties);
139 }
140 }

```

## 2.配置类

```

1  @Configuration
2  @ConditionalOnClass(Redisson.class)
3  @EnableConfigurationProperties(RedisProperties.class)
4  public class RedissonConfig {
5      private static final Logger LOGGER = LoggerFactory.getLogger(RedissonConfig.class);
6
7      @Autowired
8      RedissonStrategyFactory redissonStrategyFactory;
9
10     @Bean
11     @ConditionalOnMissingBean
12     public Redisson redisson(RedisProperties redisProperties) {
13         Config config = redissonStrategyFactory.createConfig(redisProperties);
14         RedissonEnum.SINGLE_STRATEGY); //单机模式
15         Redisson redisson = (Redisson) Redisson.create(config);
16         LOGGER.info("redisson 初始化完成.");
17         return redisson;
18     }
19 }

```

## 三、配置 - 注解Bean - 方式二

```

1 public class RedisConstant {
2     public static final String CONDITION_PREFIX = "annotation.redisson.stra
3     public static final String CONDITION_YES = "Y";
4     public static final String REDISSON_STRATEGY_SINGLE = "single";
5     public static final String REDISSON_STRATEGY_CLUSTER = "cluster";
6     public static final String REDISSON_STRATEGY_SENTINEL = "sentinel";
7 }
8
9 @Configuration
10 @ConditionalOnClass(Redisson.class)
11 public class RedissonConfig {
12     private static final Logger LOGGGER = LoggerFactory.getLogger(RedissonCo
13
14     /**
15      * 策略 - 注解方式
16      */
17     @Autowired
18     RedisProperties redisProperties;
19
20     @Bean
21     @ConditionalOnProperty(prefix = RedisConstant.CONDITION_PREFIX, name =
RedisConstant.REDISSON_STRATEGY_SINGLE, havingValue = RedisConstant.CONDITION
22     public Redisson singleRedisson() {
23         return initRedisson((config, redisProperties) -> {
24             String address =
redisProperties.getHost().concat(":").concat(redisProperties.getPort()+"");
25             String password = redisProperties.getPassword();
26             int database = redisProperties.getDatabase();
27             String redisAddr = redisProperties.isSsl() ? RedissonEnum.SSL_U
RedissonEnum.URL_PREFIX.getValue();
28             redisAddr += address;
29             config.useSingleServer().setAddress(redisAddr);
30             config.useSingleServer().setDatabase(database);
31             //密码可以为空
32             if (StringUtils.isNotBlank(password)) {
33                 config.useSingleServer().setPassword(password);
34             }
35             LOGGGER.info("部署策略[单机模式] -> redisAddress: {}", address);
36             return (Redisson) Redisson.create(config);
37         });
38     }
39
40     @Bean
41     @ConditionalOnProperty(prefix = RedisConstant.CONDITION_PREFIX, name =

```

```

RedisConstant.REDISSON_STRATEGY_CLUSTER, havingValue = RedisConstant.CONDIT
42     public Redisson clusterRedisson() {
43         return initRedisson((config, redisProperties) -> {
44             List<String> address = redisProperties.getCluster().getNodes();
45             String password = redisProperties.getPassword();
46             //String[] addrTokens = address.split(",");
47             //设置cluster节点的服务IP和端口
48             String redisAddr = redisProperties.isSsl() ? RedissonEnum.SSL_UI
RedissonEnum.URL_PREFIX.getValue();
49             for (String address : address) {
50                 config.useClusterServers()
51                     .addNodeAddress(redisAddr + address);
52                 if (StringUtils.isNotBlank(password)) {
53                     config.useClusterServers().setPassword(password);
54                 }
55             }
56             LOGGER.info("部署策略[集群模式] -> redisAddress: {}", address);
57             return (Redisson) Redisson.create(config);
58         });
59     }
60
61     @Bean
62     @ConditionalOnProperty(prefix = RedisConstant.CONDITION_PREFIX, name =
RedisConstant.REDISSON_STRATEGY_SENTINEL, havingValue = RedisConstant.CONDI
63     public Redisson sentinelRedisson() {
64         return initRedisson((config, redisProperties) -> {
65             String masterAddress = redisProperties.getSentinel().getMaster(
66             String password = redisProperties.getPassword();
67             int database = redisProperties.getDatabase();
68             List<String> nodeAddress = redisProperties.getSentinel().getNode
69             String sentinelAliasName = Optional.ofNullable(nodeAddress)
70                 .map(u-> nodeAddress.get(0))
71                 .orElseThrow(() -> new RuntimeException("redis.sentinel
please set"));
72             //设置redis配置文件sentinel.conf配置的sentinel别名
73             config.useSentinelServers().setMasterName(sentinelAliasName);
74             config.useSentinelServers().setDatabase(database);
75             if (StringUtils.isNotBlank(password)) {
76                 config.useSentinelServers().setPassword(password);
77             }
78             //设置sentinel节点的服务IP和端口
79             String redisAddr = redisProperties.isSsl() ? RedissonEnum.SSL_UI
RedissonEnum.URL_PREFIX.getValue();
80             for (String nodeAddress : nodeAddress) {

```

```

81         config.useSentinelServers().addSentinelAddress(redisAddr +
82     }
83     LOGGER.info("部署策略[哨兵模式] -> masterAddress: {}, nodeAddress
nodeAddress);
84     return (Redisson) Redisson.create(config);
85 });
86 }
87
88 private Redisson initRedisson(BiFunction<Config, RedisProperties, Redis
Redisson redisson = null;
89     Config config = new Config();
90     try {
91         LOGGER.info("redisson init start...");
92         redisson = func.apply(config, redisProperties);
93         LOGGER.info("redisson init complete.");
94     } catch (Exception e) {
95         LOGGER.error("redisson init error!", e);
96         e.printStackTrace();
97     }
98     return redisson;
99 }
100 }
101 }

```

## 四、工具类

```

1  @Slf4j
2  @Data
3  @Component
4  @ConditionalOnBean(Redisson.class)
5  public class RedissonLockService {
6
7      @Autowired
8      private Redisson redisson;
9
10     /**
11      * 加锁操作 （设置锁的有效时间）
12      * @param lockName 锁名称
13      * @param leaseTime 锁有效时间
14      */
15     public void lock(String lockName, long leaseTime) {
16         RLock rLock = redisson.getLock(lockName);
17
18         rLock.lock(leaseTime, TimeUnit.SECONDS);

```

```
18     }
19
20     /**
21      * 加锁操作（锁有效时间采用默认时间30秒）
22      * @param lockName 锁名称
23      */
24     public void lock(String lockName) {
25         RLock rLock = redisson.getLock(lockName);
26         rLock.lock();
27     }
28
29     /**
30      * 加锁操作(tryLock锁，没有等待时间)
31      * @param lockName 锁名称
32      * @param leaseTime 锁有效时间
33      */
34     public boolean tryLock(String lockName, long leaseTime) {
35
36         RLock rLock = redisson.getLock(lockName);
37         boolean getLock = false;
38         try {
39             getLock = rLock.tryLock( leaseTime, TimeUnit.SECONDS);
40         } catch (InterruptedException e) {
41             log.error("获取Redisson分布式锁[异常], lockName=" + lockName, e);
42             e.printStackTrace();
43             return false;
44         }
45         return getLock;
46     }
47
48     /**
49      * 加锁操作(tryLock锁，有等待时间)
50      * @param lockName 锁名称
51      * @param leaseTime 锁有效时间
52      * @param waitTime 等待时间
53      */
54     public boolean tryLock(String lockName, long leaseTime, long waitTime)
55
56         RLock rLock = redisson.getLock(lockName);
57         boolean getLock = false;
58         try {
59             getLock = rLock.tryLock( waitTime, leaseTime, TimeUnit.SECONDS);
60         } catch (InterruptedException e) {
```

```
61         log.error("获取Redisson分布式锁[异常], lockName=" + lockName, e);
62         e.printStackTrace();
63         return false;
64     }
65     return getLock;
66 }
67
68 /**
69  * 解锁
70  * @param lockName 锁名称
71  */
72 public void unlock(String lockName) {
73     redisson.getLock(lockName).unlock();
74 }
75
76 /**
77  * 判断该锁是否已经被线程持有
78  * @param lockName 锁名称
79  */
80 public boolean isLock(String lockName) {
81     RLock rLock = redisson.getLock(lockName);
82     return rLock.isLocked();
83 }
84
85 /**
86  * 判断该线程是否持有当前锁
87  * @param lockName 锁名称
88  */
89 public boolean isHeldByCurrentThread(String lockName) {
90     RLock rLock = redisson.getLock(lockName);
91     return rLock.isHeldByCurrentThread();
92 }
93
94 /**
95  * 存储缓存
96  */
97 public <T> void setBucket(String key, T value) {
98     RBucket<T> bucket = redisson.getBucket(key, StringCodec.INSTANCE);
99     bucket.set(value);
100 }
101
102 /**
103  * 存储过期缓存
```

```

104     */
105     public <T> void setBucket(String key, T value, long expired) {
106         RBucket<T> bucket = redisson.getBucket(key, JsonJacksonCodec.INSTANCE);
107         bucket.set(value, expired, TimeUnit.SECONDS);
108     }
109
110     /**
111      * 读取缓存
112      */
113     public <T> T getBucket(String key, Class<T> clazz) {
114         RBucket<T> bucket = redisson.getBucket(key, JsonJacksonCodec.INSTANCE);
115         return bucket.get();
116     }
117
118     /**
119      * 发布消息
120      */
121     public <T> long publishMessage(String topic, T message) {
122         //T要实现序列化接口
123         RTopic rTopic = redisson.getTopic(topic, new SerializationCodec());
124         long l = rTopic.publish(message);
125         log.info("message push success.");
126         return l;
127     }
128
129     /**
130      * 订阅消息
131      */
132     public <T> void receiveMessage(String topic, Class<T> clazz, BiConsumer<
133     T> consumer) {
134         RTopic rTopic = redisson.getTopic(topic, new SerializationCodec());
135         rTopic.addListenerAsync(clazz, (charSequence, msg) -> {
136             //消费的具体内容
137             consumer.accept(charSequence, msg);
138             log.info("message consumer complete.");
139         });
140     }
141 }

```

## 五、测试

```
1 //注解方式加锁
2 @Target({ElementType.TYPE, ElementType.METHOD})
3 @Retention(RetentionPolicy.RUNTIME)
4 @Documented
5 @Inherited
6 public @interface DistributedLock {
7     /**
8      * 锁的名称
9      */
10    String value() default "redisson";
11
12    /**
13     * 锁的有效时间
14     */
15    int leaseTime() default 10;
16 }
17
18 @Slf4j
19 @Aspect
20 @Component
21 public class DistributedLockAspect {
22
23     @Autowired
24     RedissonLockService redissonLockService;
25
26     @Around("@annotation(distributedLock)")
27     public void around(ProceedingJoinPoint joinPoint, DistributedLock distributedLock) throws Throwable {
28         log.info("分布式锁-lock");
29         //获取锁名称
30         String lockName = distributedLock.value();
31         //获取超时时间，默认10秒
32         int leaseTime = distributedLock.leaseTime();
33         boolean isLock = redissonLockService.tryLock(lockName, leaseTime);
34         try {
35             if (isLock){
36                 log.info("加锁成功，开始执行业务...");
37                 joinPoint.proceed();
38                 log.info("业务执行完成.");
39             }else {
40                 log.info("尝试获取锁失败，稍后重试!");
41                 TimeUnit.MILLISECONDS.sleep(200);
42             }
43         } catch (Throwable throwable) {
```



```

44         log.error("加锁失败: ", throwable);
45         throwable.printStackTrace();
46     } finally {
47         //如果该线程还持有该锁，那么释放该锁。如果该线程不持有该锁，说明该线程
        解锁
48         if (redissonLockService.isHeldByCurrentThread(lockName)) {
49             redissonLockService.unlock(lockName);
50         }
51     }
52     log.info("分布式锁-unlock");
53 }
54 }
55
56 //测试类
57 @Slf4j
58 @RunWith(SpringRunner.class)
59 @SpringBootTest
60 public class RedissonTests {
61
62     @Autowired
63     private RedissonLockService redissonLockService;
64
65     private int goodTotal = 100;
66     private final String LOCK_KEY = "test_redisson_lock";
67
68     /**
69      * 手动加锁
70      */
71     @Test
72     public void testLock() throws Exception{
73         for (int i = 0; i < 20; i++) {
74             new Thread(() -> {
75                 while (true){
76                     try {
77                         boolean isLock = redissonLockService.tryLock(LOCK_KI
78                         if (isLock){
79                             if (goodTotal > 0){
80                                 goodTotal-=1;
81                                 log.info("{}抢购成功，库存剩余: {}",
Thread.currentThread().getName(), goodTotal);
82                             }else {
83                                 log.info("库存不足!");

```

```

84                                 break;

```

```

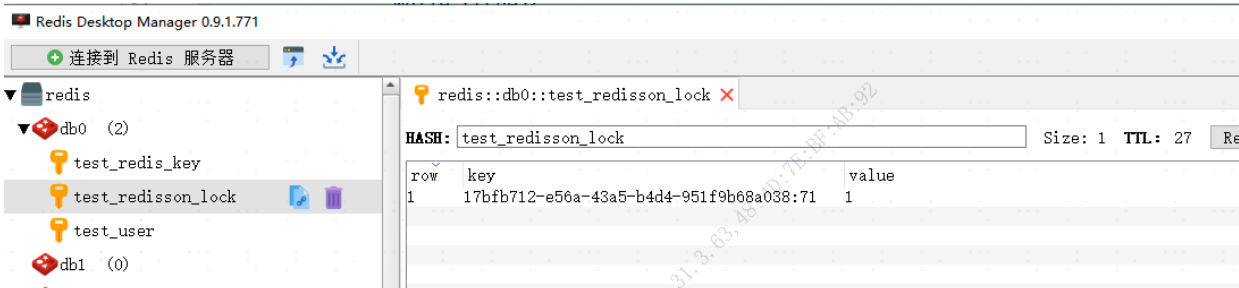
85         }
86     }else{
87         log.info("尝试获取锁失败!");
88         throw new RuntimeException("尝试获取锁失败!");
89     }
90     }finally {
91         redissonLockService.unlock(LOCK_KEY);
92     }
93 }
94 }, Thread.currentThread().getName() + "-" + i).start();
95 }
96 }
97
98
99 /**
100  * 注解-自动加锁
101  */
102 @Test
103 public void testAnnoLock() throws Exception{
104     for (int i = 0; i < 20; i++) {
105         new Thread(() -> {
106             while (true){
107                 if(!getGoods()){
108                     break;
109                 };
110             }
111             }, Thread.currentThread().getName() + "-" + i).start();
112     }
113 }
114
115 @DistributedLock(LOCK_KEY)
116 boolean getGoods(){
117     if (goodTotal > 0){
118         goodTotal-=1;
119         log.info("{}抢购成功，库存剩余：{}", Thread.currentThread().getNa
120         return true;
121     }else {
122         log.info("库存不足!");
123         return false;
124     }
125 }

```

```
127     @Test
128     public void testBucket() throws Exception{
129         String userKey = "bucket_user_key";
130         //设置
131         redissonLockService.setBucket(userKey, new User(2022, "路飞"), 5*60L);
132         //取值
133         User user = redissonLockService.getBucket(userKey, User.class);
134         log.info(user.getName());
135         //是否存在
136         boolean isExist = redissonLockService.existsBucket(userKey);
137         log.info(Boolean.toString(isExist));
138         //删除
139         //redissonLockService.removeBucket(userKey);
140     }
141
142     final String topic = "redisson_topic_test";
143     @Test
144     public void testPush() throws Exception{
145         //发布
146         long rs = redissonLockService.publishMessage(topic, new User(2022, "路飞"));
147         redissonLockService.publishMessage(topic, new User(2023, "乔巴"));
148         log.info(Long.toString(rs));
149     }
150     @Test
151     public void testSubscribe() throws Exception{
152         //订阅
153         redissonLockService.receiveMessage(topic, User.class, (charSequence, msg) -> {
154             log.info("主题: {}, 消息: {}", charSequence, msg);
155         });
156     }
157     @Test
158     public void test() throws Exception{
159         //先执行消费者，再执行生产者
160         testSubscribe();
161         testPush();
162     }
163
164 }
```

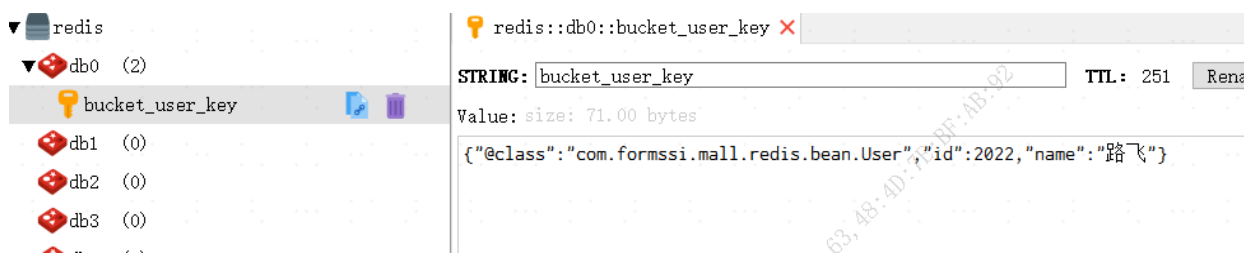
## Lock锁测试

2022-04-25 16:24:00.859	INFO	16200	---	[	main-1]	com.formssi.mall.redis.RedissonTests	:	main-1抢购成功, 库存剩余: 9
2022-04-25 16:24:00.861	INFO	16200	---	[	main-8]	com.formssi.mall.redis.RedissonTests	:	main-8抢购成功, 库存剩余: 8
2022-04-25 16:24:00.864	INFO	16200	---	[	main-1]	com.formssi.mall.redis.RedissonTests	:	main-1抢购成功, 库存剩余: 7
2022-04-25 16:24:00.866	INFO	16200	---	[	main-1]	com.formssi.mall.redis.RedissonTests	:	main-1抢购成功, 库存剩余: 6
2022-04-25 16:24:00.869	INFO	16200	---	[	main-4]	com.formssi.mall.redis.RedissonTests	:	main-4抢购成功, 库存剩余: 5
2022-04-25 16:24:00.893	INFO	16200	---	[	main-4]	com.formssi.mall.redis.RedissonTests	:	main-4抢购成功, 库存剩余: 4
2022-04-25 16:24:00.896	INFO	16200	---	[	main-4]	com.formssi.mall.redis.RedissonTests	:	main-4抢购成功, 库存剩余: 3
2022-04-25 16:24:00.925	INFO	16200	---	[	main-19]	com.formssi.mall.redis.RedissonTests	:	main-19抢购成功, 库存剩余: 2
2022-04-25 16:24:00.941	INFO	16200	---	[	main-19]	com.formssi.mall.redis.RedissonTests	:	main-19抢购成功, 库存剩余: 1
2022-04-25 16:24:00.971	INFO	16200	---	[	main-10]	com.formssi.mall.redis.RedissonTests	:	main-10抢购成功, 库存剩余: 0
2022-04-25 16:24:00.974	INFO	16200	---	[	main-10]	com.formssi.mall.redis.RedissonTests	:	库存不足!
2022-04-25 16:24:01.002	INFO	16200	---	[	main-12]	com.formssi.mall.redis.RedissonTests	:	库存不足!
2022-04-25 16:24:01.033	INFO	16200	---	[	main-16]	com.formssi.mall.redis.RedissonTests	:	库存不足!
2022-04-25 16:24:01.066	INFO	16200	---	[	main-3]	com.formssi.mall.redis.RedissonTests	:	库存不足!
2022-04-25 16:24:01.096	INFO	16200	---	[	main-6]	com.formssi.mall.redis.RedissonTests	:	库存不足!
2022-04-25 16:24:01.112	INFO	16200	---	[	main-18]	com.formssi.mall.redis.RedissonTests	:	库存不足!
2022-04-25 16:24:01.128	INFO	16200	---	[	main-0]	com.formssi.mall.redis.RedissonTests	:	库存不足!
2022-04-25 16:24:01.160	INFO	16200	---	[	main-15]	com.formssi.mall.redis.RedissonTests	:	库存不足!
2022-04-25 16:24:01.163	INFO	16200	---	[	main-7]	com.formssi.mall.redis.RedissonTests	:	库存不足!



## Bucket桶测试

>> ✓ Tests passed: 1 of 1 test - 983 ms									
3 ms	2022-04-28 16:43:06.470	INFO	8844	---	[	main]	com.formssi.mall.redis.RedissonTests	:	路飞
3 ms	2022-04-28 16:43:06.474	INFO	8844	---	[	main]	com.formssi.mall.redis.RedissonTests	:	true



## 发布-订阅测试

>> Tests passed: 1 of 1 test - 1 min 3 sec								
2022-04-28 17:25:45.059	INFO	9508	---	[	main]	c.f.m.redis.service.RedissonLockService	:	message push success.
2022-04-28 17:25:45.064	INFO	9508	---	[	redisson-3-2]	com.formssi.mall.redis.RedissonTests	:	主题: redisson_topic_test, 消息: User(id=2022, name=索隆)
2022-04-28 17:25:45.064	INFO	9508	---	[	redisson-3-2]	c.f.m.redis.service.RedissonLockService	:	message consumer complete.
2022-04-28 17:26:47.166	INFO	9508	---	[	main]	c.f.m.redis.service.RedissonLockService	:	message push success.
2022-04-28 17:26:47.168	INFO	9508	---	[	redisson-3-3]	com.formssi.mall.redis.RedissonTests	:	主题: redisson_topic_test, 消息: User(id=2023, name=乔巴)
2022-04-28 17:26:47.168	INFO	9508	---	[	main]	com.formssi.mall.redis.RedissonTests	:	1
2022-04-28 17:26:47.170	INFO	9508	---	[	redisson-3-3]	c.f.m.redis.service.RedissonLockService	:	message consumer complete.

# Spring Cache

## 为什么使用?

- 1.redis缓存操作和业务逻辑之间的代码耦合度高, 对业务逻辑有较强的侵入性
- 2.每次操作都需要定义缓存Key, 调用缓存命令的API, 产生较多的重复代码
- 3.某些场景下, 需要更换缓存组件, 每个缓存组件有自己的API, 更换成本颇高
- 4.频繁的查询redis, 也会有明显的网络IO上的消耗。对于热点key可以采用应用缓存(一级缓存), redis做二级缓存

## 一、依赖

```
1 <dependency>
2     <groupId>org.springframework.boot</groupId>
3     <artifactId>spring-boot-starter-cache</artifactId>
4 </dependency>
```

## 二、yml配置

```
1 spring:
2     cache:
3         type: redis           #缓存类型
4         redis:
5             cache-null-values: false #不缓存null数据，解决[缓存穿透]可设置为true
6             time-to-live: 120       #缓存数据过期时间，单位s
7             use-key-prefix: false   #不适用前缀
```

## 三、config配置

```
1 @Configuration
2 @AutoConfigureAfter(RedisAutoConfiguration.class)
3 @EnableCaching //开启缓存(必须))
4 public class RedisConfig {
5
6     private final Duration timeToLive = Duration.ofDays(7); //redis ttl过期
7     private final StringRedisSerializer keySerializer = new StringRedisSerializer();
8     private final GenericJackson2JsonRedisSerializer valueSerializer = new
9         GenericJackson2JsonRedisSerializer();
10
11     @Bean
12     public RedisTemplate<String, Object> redisCacheTemplate(LettuceConnectionFactory
13         redisConnectionFactory) {
14         RedisTemplate<String, Object> redisTemplate = new RedisTemplate<>();
15         redisTemplate.setKeySerializer(keySerializer);
16         redisTemplate.setValueSerializer(valueSerializer);
17         redisTemplate.setHashKeySerializer(keySerializer);
18         redisTemplate.setHashValueSerializer(valueSerializer);
19         redisTemplate.setConnectionFactory(redisConnectionFactory);
20         return redisTemplate;
21     }
22 }
```

```

20
21  /**
22   * Cache缓存管理器
23   */
24  @Bean
25  public RedisCacheManager cacheManager(RedisConnectionFactory redisConne
26      // 生成一个默认配置，通过config对象即可对缓存进行自定义配置
27      RedisCacheConfiguration config = RedisCacheConfiguration.defaultCac
28      // 设置缓存的默认过期时间，也是使用Duration设置
29      config = config
30          //不设置默认-1，永不过期
31          .entryTtl(timeToLive)
32          // 设置 key为string序列化
33      .serializeKeysWith(RedisSerializationContext.SerializationPair.fromSerializ
34          // 设置value为json序列化
35      .serializeValuesWith(RedisSerializationContext.SerializationPair.fromSerial:
36          // 不缓存空值
37          .disableCachingNullValues()
38          // 覆盖默认的构造key，否则会多出一个冒号 原规则: cacheName::key
39          .computePrefixWith(name -> name + ":")
40          ;
41
42      // 对每个缓存空间(cacheNames属性)应用不同的配置
43      Map<String, RedisCacheConfiguration> configMap = new HashMap<>();
44      configMap.put("MallCache", config.entryTtl(Duration.ofMinutes(30)));
45
46      // 使用自定义的缓存配置初始化一个cacheManager
47      RedisCacheManager cacheManager = RedisCacheManager.builder(redisCon
48          //默认配置
49          .cacheDefaults(config)
50          // 特殊配置（一定要先调用该方法设置初始化的缓存名，再初始化相关的
51          .initialCacheNames(configMap.keySet())
52          .withInitialCacheConfigurations(configMap)
53          //开启redis事务(和jdbc事务相关联)
54          .transactionAware()
55          .build();
56
57      return cacheManager;
58  }
59
60  /**

```

```

61     * 自定义json序列化器
62     */
63     private RedisSerializer<Object> jackson2JsonRedisSerializer() {
64         //使用Jackson2JsonRedisSerializer来序列化和反序列化redis的value值
65         Jackson2JsonRedisSerializer<Object> serializer = new Jackson2JsonRedisSerializer<Object>(Object.class);
66         //json转对象类，不设置默认的会将json转成hashmap
67         ObjectMapper mapper = new ObjectMapper();
68         mapper.setVisibility(PropertyAccessor.ALL, JsonAutoDetect.Visibility.ANY);
69         mapper.enableDefaultTyping(ObjectMapper.DefaultTyping.NON_FINAL);
70         serializer.setObjectMapper(mapper);
71         return serializer;
72     }
73 }

```

## 四、测试

```

1  @Slf4j
2  @CacheConfig(cacheNames = "userCache", cacheManager = "cacheManager") //配置缓存
3  @Service
4  public class UserService {
5
6      //key中的: 为目录分割符
7      //sync同步阻塞，解决[缓存击穿]
8      @Cacheable(key = "'userId:' + #id", sync = true)
9      public User getUserById(Integer id) {
10         log.info("getUserById -> Cache no data, Query DB & Add Cache");
11         //返回值进缓存
12         return new User(id, "查询不到时，加入的缓存");
13     }
14
15     //插入符合条件（1005 < id <= 1010）的，不符合的不插入cache，但会执行方法体
16     //已经存在cache则不执行方法体
17     @Cacheable(key = "'userId:' + #user.id", condition = "#p0.id > 1005", unless = "#p0.id <= 1010")
18     public User addUserByCondition(User user) {
19         log.info("addUserByCondition -> Insert DB");
20         return user;
21     }
22
23     //存在cache也会执行方法体，然后更新Cache

```

```

24     @CachePut(key = "'userId:' + #user.id")

```

```

25     public User addUserByPut(User user) {
26         log.info("addUserByPut -> Insert DB");
27         return user;
28     }
29
30     //userCache下全被删除，删除db也执行了
31     //无cache，只执行方法体
32     @CacheEvict(key = "'userId:' + #user.id", beforeInvocation = true , all
33     public void delUserById(User user) {
34         log.info("delUserById -> Delete DB");
35     }
36
37     //批量执行多个@CacheXXX
38     @Caching(
39         cacheable = {@Cacheable(key = "'userId:' + #user.id")}
40         , put = {@CachePut(key = "'userName:' + #user.name")}
41         , evict = {@CacheEvict(key = "'userId:' + #user.id")}
42     )
43     public User addUserByCaches(User user) {
44         log.info("addUserByCaches -> Insert DB");
45         return user;
46     }
47 }

```

```

1  /**
2   * @author jp
3   * @version 1.0
4   * @Description: SpringCache测试类
5   * @date 2022/4/25 17:25
6   *
7   * @CacheConfig: 在类级别共享缓存的相同配置
8   * @Cacheable: 方法返回值加入缓存。同时在查询时，会先从缓存中取，若不存在才再发起
9   *      value、cacheNames: 两个等同的参数（cacheNames为Spring 4新增，作为value
      储的集合名
10  *      key: 缓存对象存储在Map集合中的key值，非必需，缺省按照函数的所有参数组合作
      用SpEL表达式，比如：@Cacheable(key = "#p0")：使用函数第一个参数作为缓存的key值
11  *      condition: 缓存对象的条件，非必需，也需使用SpEL表达式，只有满足表达式条件
      @Cacheable(key = "#p0", condition = "#p0.length() < 3")，表示只有当第一个参数
      缓存
12  *      unless: 另外一个缓存条件参数，非必需，需使用SpEL表达式。它不同于conditio
      断时机，该条件是在函数被调用之后才做判断的，所以它可以通过对result进行判断。
13  *      keyGenerator: 用于指定key生成器，非必需。若需要指定一个自定义的key生成器
      org.springframework.cache.interceptor.KeyGenerator接口，并使用该参数来指定。需
      是互斥的

```



```

14 *      cacheManager: 用于指定使用哪个缓存管理器，非必需。只有当有多个时才需要使用
15 *      cacheResolver: 用于指定使用那个缓存解析器，非必需。需通过
      org.springframework.cache.interceptor.CacheResolver接口来实现自己的缓存解析器
16 * @CachePut: 不影响方法执行更新缓存
17 * @CacheEvict: 从缓存删除
18 * @Caching: 组合多个Cache注解使用
19 */

```

## @Cacheable

```

1
2
3 //key中的: 为目录分割符
4 //同步阻塞
5 @Cacheable(key = "'userId:' + #id", sync = true)
6 public User getUserById(Integer id) {
7     log.info("getUserById -> Cache no data, Query DB & Add Cache");
8     //返回值进缓存
9     return new User(id, "查询不到时, 加入的缓存");
10 }
11
12 @Test
13 public void getUserCache() throws Exception{
14     User user1 = userService.getUserById(1001);
15     User user2 = userService.getUserById(1001);
16     log.info(JSON.toJSONString("User1 Query Cache -> " + user1));
17     log.info(JSON.toJSONString("User2 Query Cache -> " + user2));
18     User user3 = userService.getUserById(1005);
19 }

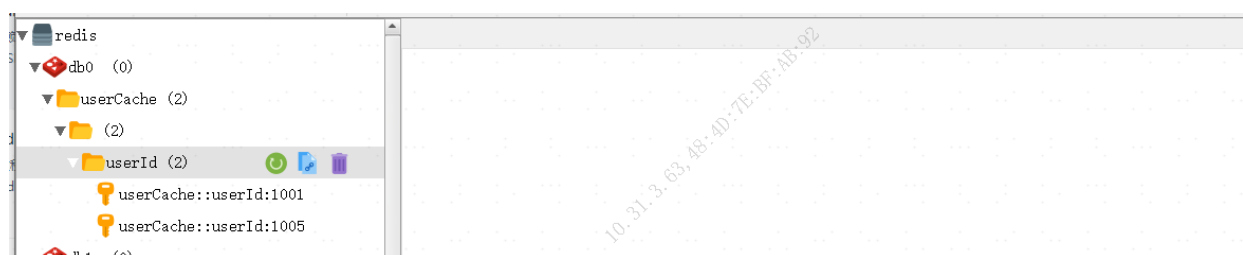
```

```

2022-04-27 14:53:25.855 INFO 11924 --- [main] c.f.mall.redis.service.UserService : getUserById -> Cache no data, Query DB & Add Cache
2022-04-27 14:53:26.079 INFO 11924 --- [main] com.formssi.mall.redis.SpringCacheTests : "User1 Query Cache -> User(id=1001, name=查询不到时, 加入的缓存)"
2022-04-27 14:53:26.079 INFO 11924 --- [main] com.formssi.mall.redis.SpringCacheTests : "User2 Query Cache -> User(id=1001, name=查询不到时, 加入的缓存)"
2022-04-27 14:53:26.081 INFO 11924 --- [main] c.f.mall.redis.service.UserService : getUserById -> Cache no data, Query DB & Add Cache

```

user2查的user1的缓存



```

1 //插入符合条件 (1005 < id <= 1010) 的, 不符合的不插入cache, 但会执行方法体
2 //已经存在cache则不执行方法体
3 @Cacheable(key = "'userId:' + #user.id", condition = "#p0.id > 1005", unless:

```

```

4 public User addUserByCondition(User user) {
5     log.info("addUserByCondition -> Insert DB");
6     return user;
7 }
8
9 @Test
10 public void addUserByCondition() throws Exception{
11     //condition = "#p0.id > 1005    不符合
12     userService.addUserByCondition(new User(1004, "李四"));
13     //condition = "#p0.id > 1005    符合
14     userService.addUserByCondition(new User(1006, "赵六"));
15     //unless = "#user.id > 1010"    不符合
16     userService.addUserByCondition(new User(1011, "王十一"));
17 }

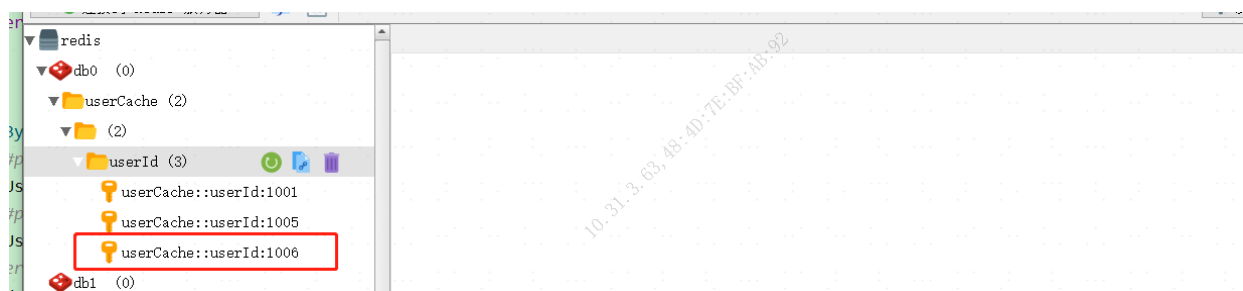
```

```

3 ms 2022-04-27 14:58:20.749 INFO 8496 --- [main] c.f.mall.redis.service.UserService : addUserByCondition -> Insert DB
5 ms 2022-04-27 14:58:21.914 INFO 8496 --- [main] c.f.mall.redis.service.UserService : addUserByCondition -> Insert DB
2022-04-27 14:58:22.028 INFO 8496 --- [main] c.f.mall.redis.service.UserService : addUserByCondition -> Insert DB

```

都执行了方法体，但只有1006符合条件，加入Cache



## @CachePut

```

1 //存在cache也会执行方法体，然后更新Cache
2 //result返回对象，result.name
3 @CachePut(key = "'userId:' + #user.id")
4 public User addUserByPut(User user) {
5     log.info("addUserByPut -> Insert DB");
6     return user;
7 }
8
9 @Test
10 public void addUserByPut() throws Exception{
11     //已经存在的数据
12     userService.addUserByPut(new User(1006, "赵六2"));
13 }

```

Tests passed: 1 of 1 test - 3 sec 206 ms

2022-04-27 15:02:15.690 INFO 4036 --- [main] c.f.mall.redis.service.UserService : addUserByPut -> Insert DB

redis

- db0 (0)
  - userCache (2)
    - (2)
      - userId (3)
        - userCache::userId:1001
        - userCache:: Copy Key Name (undefined)
        - userCache::userId:1006
- db1 (0)
- db2 (0)

redis::db0::use...he::userId:1006

STRING: userCache::userId:1006 TTL: 604692 Rename 删除

Value: size: 72.00 bytes

View

{"@class":"com.formssi.mall.redis.bean.User","id":1006,"name":"赵六2"}

由“赵六”更新为“赵六2”

## @CacheEvict

```

1 //userCache下全被删除，删除db也执行了
2 //无cache，只执行方法体
3 @CacheEvict(key = "'userId:' + #user.id", beforeInvocation = true , allEntries = true)
4 public void delUserById(User user) {
5     log.info("delUserById -> Delete DB");
6 }
7
8 @Test
9 public void delUserById() throws Exception{
10     //beforeInvocation =true 先删除cache，再删db，allEntries = true 删除所有cache
11     userService.delUserById(new User(1001, null));
12 }

```

Tests passed: 1 of 1 test - 1 sec 14 ms

2022-04-27 15:07:31.735 INFO 15768 --- [main] c.f.mall.redis.service.UserService : delUserById -> Delete DB

redis

- db0 (0)
  - userCache (0)
- db1 (0)
- db2 (0)

## @Caching

```

1 //批量执行多个@CacheXXX
2 @Caching(
3     cacheable = {@Cacheable(key = "'userId:' + #user.id")}
4     , put = {@CachePut(key = "'userName:' + #a0.name")}

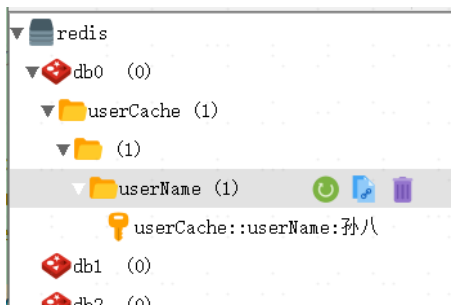
```

```

5      , evict = {@CacheEvict(key = "'userId:' + #user.id")}]
6  )
7  public User addUserByCaches(User user) {
8      log.info("addUserByCaches -> Insert DB");
9      return user;
10 }
11
12 @Test
13 public void addUserByCaches() throws Exception{
14     //已经存在的数据
15     userService.addUserByCaches(new User(1007, "孙八"));
16 }

```

2022-04-27 15:09:37.659 INFO 4004 --- [main] c.f.mall.redis.service.UserService : addUserByCaches -> Insert DB



cacheable生成了userId  
put生成了userName  
evict删除了userId