链滴

SpringBoot 整合 Redis 实现分布式锁

作者：NekoChips

原文链接：https://ld246.com/article/1573630456746

来源网站：链滴

本文主要为了回顾一下 Java 对 Redis 的操作，对 SpringBoot 自带的 Redis 组件进行简单的封装，现一个基础版的分布式锁。后续会进行扩展。话不多说，直接进入正题。

1.配置文件

POM 文件：

```
<dependencies>
        <!-- cache -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-cache</artifactId>
        </dependency>

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

</dependencies>
```

application.yml 文件：

```
spring:
  ## redis 配置
  redis:
    host: 127.0.0.1
    port: 6379
    password: 123456
    ## 连接 redis 超时时间
```

```
    timeout: 3000ms
    jedis:
      ## redis 连接池配置
      pool:
        ## 最大空闲连接数
        max-idle: 20
        ## 最小空闲连接数
        min-idle: 0
        ## 最长阻塞等待时间
        max-wait: 30
        ## 最大活跃连接数
        max-active: 15
```

2.配置RedisTemplate

```
@Configuration
public class RedisConfig
{
    /**
     * 配置RedisTemplate,设置key和value的序列化方式
     *
     * @return RedisTemplate
     */
    @Bean
    public RedisTemplate<String, Object> redisTemplate(RedisConnectionFactory connectionF
ctory) {
        RedisTemplate<String, Object> redisTemplate = new RedisTemplate<>();
        redisTemplate.setConnectionFactory(connectionFactory);

        // 配置序列化
        setSerializer(redisTemplate);
        redisTemplate.afterPropertiesSet();

        return redisTemplate;
    }

    /**
     * 配置 RedisTemplate 序列化
     *
     * @param redisTemplate redisTemplate
     */
    @SuppressWarnings("unchecked")
    private void setSerializer(@SuppressWarnings("rawtypes") RedisTemplate redisTemplate) {

        StringRedisSerializer stringRedisSerializer = new StringRedisSerializer();

        //noinspection rawtypes,unchecked
        Jackson2JsonRedisSerializer jsonRedisSerializer = new Jackson2JsonRedisSerializer(Objec
.class);
        ObjectMapper om = new ObjectMapper();
        om.setVisibility(PropertyAccessor.ALL, JsonAutoDetect.Visibility.ANY);
        om.enableDefaultTyping(ObjectMapper.DefaultTyping.NON_FINAL);
```

```
        jsonRedisSerializer.setObjectMapper(om);

        redisTemplate.setKeySerializer(stringRedisSerializer);
        redisTemplate.setValueSerializer(jsonRedisSerializer);
        redisTemplate.setHashKeySerializer(stringRedisSerializer);
        redisTemplate.setHashValueSerializer(jsonRedisSerializer);
    }
}
```

这里如果不设置 RedisTemplate的序列化方式话，默认会使用 JdkSerializationRedisSerializer，虽对使用上没什么影响，但数据的可读性会比较差。（可以使用默认的序列化器体验一下）

3.简单地封装常用功能

```
@SuppressWarnings("unchecked")
@Component
public class RedisTool {

    private Logger logger = LoggerFactory.getLogger(RedisTool.class);

    @Autowired
    @SuppressWarnings("rawtypes")
    private RedisTemplate redisTemplate;

    @Autowired
    private ObjectMapper objectMapper;

    /**
     * 添加缓存
     *
     * @param key   key
     * @param value 缓存内容
     */
    public void put(Object key, Object value) {
        if (verifyParam(key, value)) {
            redisTemplate.opsForValue().set(key, value);;
        }
    }

    /**
     * 添加缓存
     *
     * @param key        key
     * @param value      缓存内容
     * @param expireTime 过期时间
     */
    public void put(Object key, Object value, Long expireTime) {
        if (verifyParam(key, value)) {
            redisTemplate.opsForValue().set(key, value, Duration.ofSeconds(expireTime));
        }
    }
```

```
/**
 * 添加缓存
 *
 * @param key        key
 * @param value      value
 * @param expireTime 过期时间
 * @param timeUnit   时间单位
 */
public void put(Object key, Object value, Long expireTime, TimeUnit timeUnit) {
    if (verifyParam(key, value)) {
        redisTemplate.opsForValue().set(key, value, expireTime, timeUnit);
    }
}

/**
 * 获取 对象
 *
 * @param key   key
 * @param clazz 对象class
 * @param <T>   对象泛型
 * @return 对象
 */
public <T> T get(Object key, Class<T> clazz) {
    if (verifyParam(key)) {
        Object value = redisTemplate.opsForValue().get(key);
        if (value != null) {
            try {
                String jsonStr = objectMapper.writeValueAsString(value);
                return objectMapper.readValue(jsonStr, clazz);
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
    return null;
}

/**
 * 获取 字符串
 *
 * @param key key
 * @return 字符串
 */
public String get(Object key) {
    if (verifyParam(key)) {
        Object o = redisTemplate.opsForValue().get(key);
        return o == null ? null : String.valueOf(o);
    }
    return null;
}

/**
 * 删除缓存
 *
```

```
     * @param key key
     * @return 是否删除成功
     */
    public Boolean delete(Object key) {
        if (verifyParam(key)) {
            return redisTemplate.delete(key);
        }
        return Boolean.FALSE;
    }

    /**
     * 如果缓存不存在 添加
     *
     * @param key   key
     * @param value value
     */
    public void setIfAbsent(Object key, Object value) {
        if (verifyParam(key, value)) {
            redisTemplate.opsForValue().setIfAbsent(key, value);
        }
    }

    /**
     * 如果缓存不存在 添加
     *
     * @param key        key
     * @param value      value
     * @param expireTime 过期时间
     */
    public void setIfAbsent(Object key, Object value, long expireTime) {
        if (!verifyParam(key, value)) {
            return;
        }
        redisTemplate.opsForValue().setIfAbsent(key, value, Duration.ofSeconds(expireTime));
    }

    /**
     * 检查 键值对合法性
     *
     * @param key   key
     * @param value value
     * @return 是否合法
     */
    private boolean verifyParam(Object key, Object value) {
        if (key == null || value == null) {
            logger.warn("Illegal param, null key or null value.");
            return Boolean.FALSE;
        }
        return Boolean.TRUE;
    }

    /**
     * 检查 key 合法性
     *
```

```
     * @param key key
     * @return 是否合法
     */
    private boolean verifyParam(Object key) {
        if (key == null) {
            logger.warn("Illegal param, null key.");
            return Boolean.FALSE;
        }
        return Boolean.TRUE;
    }
}
```

---

4. 锁的实现

抽象锁 RedisLock，尝试获取锁和释放锁具体由其子类实现。

```
public abstract class RedisLock {

    public RedisTool redisTool;

    private String lockUUId;

    public RedisLock(RedisTool redisTool) {
        this.redisTool = redisTool;
    }

    public String getLockUUId() {
        return lockUUId;
    }

    public void setLockUUId(String lockUUId) {
        this.lockUUId = lockUUId;
    }

    /**
     * 加锁
     */
    public void acquire() {
        if (tryAcquire()) {
            redisTool.put(RedisTool.LOCK_NAME, lockUUId);
        }
    }

    /**
     * 加锁一定时间后自动释放锁
     *
     * @param leaseTime 超过该时间，自动释放锁
     * @param timeUnit  时间单位
     */
    public void acquire(Long leaseTime, TimeUnit timeUnit) {
        if (tryAcquire()) {
            redisTool.put(RedisTool.LOCK_NAME, lockUUId, leaseTime, timeUnit);
```

```
        }
    }

    /**
     * 尝试加锁
     *
     * @return 是否可以加锁
     */
    public abstract Boolean tryAcquire();

    /**
     * 释放锁
     */
    public void release() {
        if (tryRelease()) {
            redisTool.delete(RedisTool.LOCK_NAME);
        }
    }

    /**
     * 尝试释放锁
     *
     * @return 是否可以释放锁
     */
    public abstract Boolean tryRelease();
}
```

SimpleRedisLock：锁的简单实现

```
public class SimpleRedisLock extends RedisLock {
    private Logger logger = LoggerFactory.getLogger(SimpleRedisLock.class);

    private Long waitTime = 3000L;

    public SimpleRedisLock(RedisTool redisTool) {
        super(redisTool);
    }

    @Override
    public Boolean tryAcquire() {
        long startMills = System.currentTimeMillis();
        String lockUUID = getLockUUId();
        logger.debug("{} try to acquire the lock {}", Thread.currentThread().getName(), lockUUID);
        while (System.currentTimeMillis() - startMills < waitTime) {
            String existLock = redisTool.get(RedisTool.LOCK_NAME);
            if (!(StringUtils.isNotBlank(existLock) && StringUtils.equals(lockUUID, existLock))) {
                logger.debug("lock {} can be acquired", lockUUID);
                return Boolean.TRUE;
            }
        }
        logger.debug("{} try to acquire lock {} failed", Thread.currentThread().getName(), lockUU
D);
        return Boolean.FALSE;
    }
```

```
    @Override
    public Boolean tryRelease() {
        String lockUUID = getLockUUId();
        String existLock = redisTool.get(RedisTool.LOCK_NAME);
        if (StringUtils.isBlank(existLock) || !StringUtils.equals(lockUUID, existLock)) {
            logger.debug("lock {} has been released", lockUUID);
            return Boolean.TRUE;
        }
        if (StringUtils.equals(lockUUID, existLock)) {
            logger.debug("lock {} can be released", lockUUID);
            return Boolean.TRUE;
        }
        return Boolean.FALSE;
    }
}
```

6.测试锁的功能

测试类：

```
@RunWith(SpringRunner.class)
@SpringBootTest(classes = RedisApplication.class)
public class TestSimpleRedisLock
{
    private Logger logger = LoggerFactory.getLogger(TestSimpleRedisLock.class);

    @Autowired
    private RedisTool redisTool;

    @Test
    public void testSimpleRedisLock() throws InterruptedException
    {
        RedisLock lock = new SimpleRedisLock(redisTool);
        lock.setLockUUId(UUID.randomUUID().toString());

        // 2秒后自动释放锁
        lock.acquire(2L, TimeUnit.SECONDS);
        logger.info("main acquired lock {}", lock.getLockUUId());

        Thread thread = new Thread(() -> {
            lock.acquire();
            logger.info("thread acquire lock {}", lock.getLockUUId());
            lock.release();
            logger.info("thread release lock {}", lock.getLockUUId());
        });

        thread.start();
        // 手动释放锁，
        lock.release();
        logger.info("main release lock {}", lock.getLockUUId());
        thread.join();
    }
}
```

测试结果：

```
20:08:10.047  INFO 11020 --- [           main] com.demo.redis.core.TestSimpleRedisLock  : main cquired lock 78cd8137-90f1-4830-9509-ff69b718083f
20:08:12.085  INFO 11020 --- [       Thread-3] com.demo.redis.core.TestSimpleRedisLock  : thre d acquire lock 78cd8137-90f1-4830-9509-ff69b718083f
20:08:12.157  INFO 11020 --- [       Thread-3] com.demo.redis.core.TestSimpleRedisLock  : thre d release lock 78cd8137-90f1-4830-9509-ff69b718083f
```

后记

代码中还有很多地方不够严谨，例如获取锁超时，失败重试等机制的实现，本文主要讨论的是分布式实现的思想，如有不当欢迎指正。

源码地址：https://github.com/NekoChips/SpringDemo/09.springboot-redis