基于数据库实现排他锁

- 方案1.利用数据库的唯一性索引

```
INSERT INTO method_lock (method_name, desc) VALUES ('methodName', 'methodName');
# 对method_name做了唯一性约束，这里如果有多个请求同时提交到数据库的话，数据库会保证只有一个操作可以成功
```

- 方案2.

```
# 先获取锁的信息
     select id, method_name, state,version from method_lock where state=1 and method_name='methodName';

# 占有锁
       update t_resoure set state=2, version=2, update_time=now() where method_name='methodName' and state=1 and version=2;

# 如果没有更新影响到一行数据，则说明这个资源已经被别人占位了。
```

- 方案3.

```
select 语句 + for update   # 加锁
```

基于redis实现-redisson

```
# SET resource_name my_random_value NX PX 30000
# 利用nx的原子性
 @RequestMapping("redisLock")
    public String redisLock() {
        String key = "redisKey";
        String value = UUID.randomUUID().toString();
        RedisCallback<Boolean> redisCallback = redisConnection -> {
            // 设置nx
            RedisStringCommands.SetOption setOption = RedisStringCommands.SetOption.ifAbsent();
            // 设置过期时间
            Expiration seconds = Expiration.seconds(30);
            // 序列化
            byte[] redisKey = redisTemplate.getKeySerializer().serialize(key);
            byte[] redisValue = redisTemplate.getValueSerializer().serialize(value);
```

```
            Boolean result = redisConnection.set(redisKey, redisValue, seconds,
setOption);
            return result;

        };
        Boolean lock = (boolean)redisTemplate.execute(redisCallback);

        if (lock) {
            log.info("我进入了锁！");
            try {
                Thread.sleep(15000);

            } catch (InterruptedException e) {
                e.printStackTrace();
            } finally {
                String script = "if redis.call(\"get\",KEYS[1])== ARGV[1]
then\n" +
                        " return redis.call(\"del\", KEYS[1])\n" +
                        "else\n" +
                        " return 0\n" +
                        "end";
                RedisScript<Object> redisScript = RedisScript.of(script,
Boolean.class);
                List<String> keys = Arrays.asList(key);
                Boolean result = (Boolean) redisTemplate.execute(redisScript,
keys, value);

                log.info("释放锁的结果：" + result);
            }
        }
        return "";
```

基于zookeeper-curator

```
# zookeeper分布式锁恰恰应用了临时顺序节点，瞬时有序。
# 实现等待队列
```

三种方案的比较

- 从理解的难易程度角度（从低到高）：数据库 > 缓存 > Zookeeper
- 从实现的复杂性角度（从低到高）：Zookeeper >= 缓存 > 数据库
- 从性能角度（从高到低）：缓存 > Zookeeper >= 数据库
- 从可靠性角度（从高到低）：Zookeeper > 缓存 > 数据库