分布式锁的特性是排他、避免死锁、高可用。分布式锁的实现可以通过数据库的乐观锁(通过版本号)或者悲观锁(通过for update)、Redis的setnx()命令、Zookeeper（在某个持久节点添加临时有序节点，判断当前节点是否是序列中最小的节点，如果不是则监听比当前节点还要小的节点。如果是，获取锁成功。当被监听的节点释放了锁(也就是被删除)，会通知当前节点。然后当前节点再尝试获取锁，如此反复）

一、Redis的形式实现分布式锁

#### 1、Redis配置

spring.redis.host=

spring.redis.port=6379

#reids超时连接时间

spring.redis.timeout=100000

spring.redis.password=

#连接池最大连接数

spring.redis.pool.max-active=10000

#连接池最大空闲数

spring.redis.pool.max-idle=1000

#连接池最大等待时间

spring.redis.pool.max-wait=10000

@Configuration

@ConfigurationProperties(prefix = "spring.redis")

public class RedisConfig {

@Value("${spring.redis.host}")

private String host;

@Value("${spring.redis.port}")

private int port;

@Value("${spring.redis.password}")

private String password;

@Value("${spring.redis.timeout}")

private int timeout;

@Value("${spring.redis.pool.max-active}")

private int poolMaxActive;

@Value("${spring.redis.pool.max-idle}")

private int poolMaxIdle;

@Value("${spring.redis.pool.max-wait}")

private int poolMaxWait;

}

JedisPoolConfig连接池配置：

@Configuration

public class RedisPoolFactory {

@Autowired

private RedisConfig redisConfig;

@Bean

public JedisPool jedisPoolFactory() {

JedisPoolConfig poolConfig = new JedisPoolConfig();

poolConfig.setMaxIdle(redisConfig.getPoolMaxIdle());

poolConfig.setMaxTotal(redisConfig.getPoolMaxActive());

poolConfig.setTestOnBorrow(true);

poolConfig.setMaxWaitMillis(redisConfig.getPoolMaxWait());

JedisPool jp = new JedisPool(poolConfig, redisConfig.getHost(),

redisConfig.getPort(),

redisConfig.getTimeout(),

redisConfig.getPassword(), 0);

return jp;

}

}

#### 2、区分不同模块的key，抽象出一个KeyPrefix接口和BasePrefix类

public interface KeyPrefix {

int expireSeconds();

String getPrefix();

}

public abstract class BasePrefix implements KeyPrefix {

private int expireSeconds;

private String prefix;

public BasePrefix(int expireSeconds, String prefix) {

this.expireSeconds = expireSeconds;

this.prefix = prefix;

}

public BasePrefix(String prefix) {

this(0, prefix);

}

@Override

public int expireSeconds() {

return expireSeconds;

}

@Override

public String getPrefix() {

String className = getClass().getSimpleName();

return className + ":" + prefix;

}

}

#### 3、分布式锁分析与编码

因为分布式系统之间是不同进程的，单机版的锁无法满足要求。所以我们可以借助中间件Redis的setnx()命令实现分布式锁。setnx()命令只会对不存在的key设值，返回1代表获取锁成功。对存在的key设值，会返回0代表取锁失败。这里的value是System.currentTimeMillis() (获取锁的时间)+锁持有的时间。

我这里设置锁持有的时间是200ms，实际业务执行的时间远比这200ms要多的多，持有锁的客户端应该检查锁是否过期，保证锁在释放之前不会过期。因为客户端故障的情况可能是很复杂的。比如现在有A，B两个客户端。A客户端获取了锁，执行业务中做了骚操作导致阻塞了很久，时间应该远远超过200ms，当A客户端从阻塞状态下恢复继续执行业务代码时，A客户端持有的锁由于过期已经被其他客户端占有。这时候A客户端执行释放锁的操作，那么有可能释放掉其他客户端的锁。

我这里设置的客户端等待锁的时间是200ms。这里通过轮询的方式去让客户端获取锁。如果客户端在200ms之内没有锁的话，直接返回false。实际场景要设置合适的客户端等待锁的时间，避免消耗CPU资源。

如果获取锁的逻辑只有这三行代码的话，会造成死循环，明显不符合分布式锁的特性。