正在加载一言...

Redis分布式锁

1、常见分布式锁实现原理

- setnx key value

> 通过setnx命令，当且仅当key不存在时，将key的值设置为value；当key已经存在时，不执行set操作
>
> SETNX是【SET If Not Exists】(如果不存在，则set)的简写；
>
> 返回值：
>
> 1 设置key成功
> 0 设置key失败
>
> 使用：
>
> 使用setnx，加锁成功后，通过expire给锁增加过期时间，使用完解锁
>
> 优点：简单，在并发不高的场景下，可以使用
>
> 缺点：加锁和设置过期时间非原子操作，可能出现加锁之后，程序异常，没有设置过期时间，导致程序阻塞

- SET实现的分布式锁

> 语法：
>
> SET key value [EX seconds|PX milliseconds] [NX|XX]
>
> 将键`key`设定为指定的"字符串"值。如果 `key` 已经保存了一个值，那么这个操作会直接覆盖原来的值，并且忽略原始类型。当`set`命令执行成功之后，之前设置的过期时间都将失效。
>
> 使用：通过命令加锁，加锁的同时会设置过期时间，用完之后进行解锁
>
> 优点：加锁和设置过期时间是原子操作，不会出现程序一直阻塞的问题
>
> 缺点：过期时间内，线程A加锁的业务还没有执行完，可能其它线程 线程B会获取锁，然后之前的线程A执行完进行解锁，导致线程B的锁被释放了

2、Redisson实现的分布式锁

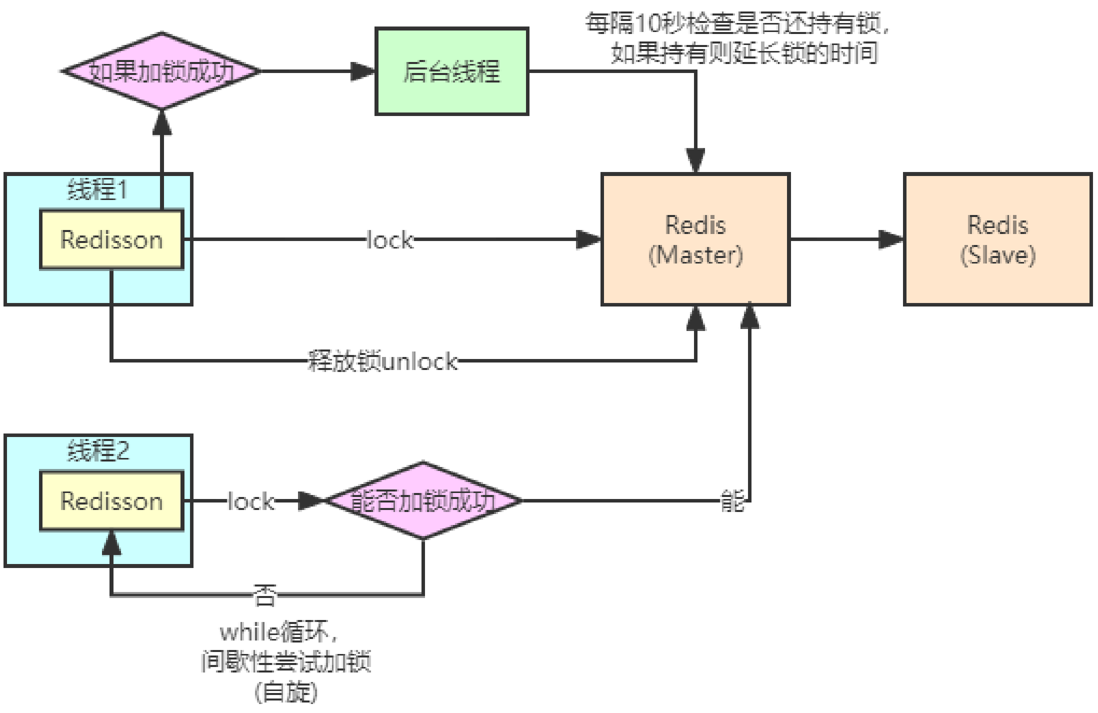
示例：

```
@Bean
    public Redisson redisson() {
        // 此为单机模式
        Config config = new Config();
 config.useSingleServer().setAddress("redis://localhost:6379").setDatabase(0);
        return (Redisson) Redisson.create(config);
    }


//获取锁对象
        RLock redissonLock = redisson.getLock(lockKey);
        //加分布式锁
        redissonLock.lock();  //  .setIfAbsent(lockKey, clientId, 30, TimeUnit.SECONDS);
        try {
            int stock = Integer.parseInt(stringRedisTemplate.opsForValue().get("stock")); // jedis.get("stock")
            if (stock > 0) {
                int realStock = stock - 1;
                stringRedisTemplate.opsForValue().set("stock", realStock + ""); // jedis.set(key,value)
                System.out.println("扣减成功，剩余库存:" + realStock);
            } else {
                System.out.println("扣减失败，库存不足");
            }
        } finally {
            //解锁
            redissonLock.unlock();
        }
```

底层逻辑:

- 锁续命
- 锁重入



加锁逻辑:

```
//加分布式锁
redissonLock.lock();
------>加锁
public void lock() {
        try {
            lockInterruptibly();/**【0】*/
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
    }
------>加锁【0】
public void lockInterruptibly() throws InterruptedException {
        lockInterruptibly(-1, null);/**【1】*/
    }
------>加锁【1】
public void lockInterruptibly(long leaseTime, TimeUnit unit) throws
InterruptedException {
        long threadId = Thread.currentThread().getId(); // 线程id
        Long ttl = tryAcquire(leaseTime, unit, threadId);/**【2】*/
        // lock acquired
        if (ttl == null) {  // 加锁成功
            return;
        }
        // redis的发布/订阅  抢锁失败 订阅channel 【redisson_lock__channel】
        RFuture<RedissonLockEntry> future = subscribe(threadId);
        commandExecutor.syncSubscription(future);

        try {
            while (true) { // 没加锁成功 自旋  非公平锁实现
                ttl = tryAcquire(leaseTime, unit, threadId); // 又尝试加一次
锁
                // lock acquired
                if (ttl == null) { // 加锁成功 跳出循环
                    break;
                }

                // waiting for message
                if (ttl >= 0) {
                    getEntry(threadId).getLatch().tryAcquire(ttl,
TimeUnit.MILLISECONDS); // 调用Semaphore信号量 许可数量为0 获取许可逻辑 获取不到
阻塞ttl s  阻塞等待 让出cpu  不会占用cpu  ----->唤醒 见  【7】
                } else {
                    getEntry(threadId).getLatch().acquire();
                }
            }
        } finally {
            unsubscribe(future, threadId);
        }
//        get(lockAsync(leaseTime, unit));
    }
------>加锁【2】
private Long tryAcquire(long leaseTime, TimeUnit unit, long threadId) {
        return get(tryAcquireAsync(leaseTime, unit, threadId)/**【3】*/);
    }
------>加锁【3】
private <T> RFuture<Long> tryAcquireAsync(long leaseTime, TimeUnit unit,
final long threadId) {
        if (leaseTime != -1) { // 默认-1 会走下面逻辑
            return tryLockInnerAsync(leaseTime, unit, threadId,
RedisCommands.EVAL_LONG);
        }
        RFuture<Long> ttlRemainingFuture =
tryLockInnerAsync(commandExecutor.getConnectionManager().getCfg().getLockW
atchdogTimeout(), TimeUnit.MILLISECONDS, threadId,
RedisCommands.EVAL_LONG);/**【4】*/
        ttlRemainingFuture.addListener(new FutureListener<Long>() {
```

```java
            @Override
            public void operationComplete(Future<Long> future) throws Exception {
                if (!future.isSuccess()) {
                    return;
                }

                Long ttlRemaining = future.getNow();
                // lock acquired
                if (ttlRemaining == null) {  // 加锁成功
                    scheduleExpirationRenewal(threadId);  /**【5】*/
                }
            }
        });
        return ttlRemainingFuture;
    }
------>加锁【4】
<T> RFuture<T> tryLockInnerAsync(long leaseTime, TimeUnit unit, long threadId, RedisStrictCommand<T> command) {
        internalLockLeaseTime = unit.toMillis(leaseTime);
        // 通过lua脚本加锁
        return commandExecutor.evalWriteAsync(getName(), LongCodec.INSTANCE, command,
                  "if (redis.call('exists', KEYS[1]) == 0) then " +   // key不存在 往hash中添加数据 同时设置过期时间
                      "redis.call('hset', KEYS[1], ARGV[2], 1); " +
                      "redis.call('pexpire', KEYS[1], ARGV[1]); " +
                      "return nil; " +      // 加锁成功 ，返回null
                  "end; " +
                  "if (redis.call('hexists', KEYS[1], ARGV[2]) == 1) then " + // key 和filed存在 value值自增 重新设置过期时间【锁重入逻辑】
                      "redis.call('hincrby', KEYS[1], ARGV[2], 1); " +
                      "redis.call('pexpire', KEYS[1], ARGV[1]); " +
                      "return nil; " +
                  "end; " +
                  "return redis.call('pttl', KEYS[1]);", // 没有加锁成功 返回这个key剩余的过期时间
                    Collections.<Object>singletonList(getName()/**redis key getLock()方法传入*/), internalLockLeaseTime/**过期时间，默认30s*/, getLockName(threadId)/**设置一个值   */);
    }

------>加锁【5】
private void scheduleExpirationRenewal(final long threadId) {
        if (expirationRenewalMap.containsKey(getEntryName())) {
            return;
        }

        Timeout task = commandExecutor.getConnectionManager().newTimeout(new TimerTask() {
            @Override
            public void run(Timeout timeout) throws Exception {

                RFuture<Boolean> future = commandExecutor.evalWriteAsync(getName(), LongCodec.INSTANCE, RedisCommands.EVAL_BOOLEAN,
                        "if (redis.call('hexists', KEYS[1], ARGV[2]) == 1) then " +  // 判断hash的key 对应的field是否存在[存在表明没有执行完]，存在则设置key过期
                            "redis.call('pexpire', KEYS[1], ARGV[1]); " +
                            "return 1; " +
                        "end; " +
                        "return 0;",
                          Collections.<Object>singletonList(getName()), internalLockLeaseTime, getLockName(threadId));

                future.addListener(new FutureListener<Boolean>() { // 延时任务 延时执行
                    @Override
                    public void operationComplete(Future<Boolean> future) throws Exception {
                        expirationRenewalMap.remove(getEntryName());
                        if (!future.isSuccess()) {
                            log.error("Can't update lock " + getName() + " expiration", future.cause());
                            return;
                        }

                        if (future.getNow()) {
                            // reschedule itself
                            scheduleExpirationRenewal(threadId);  // 此处自我调用 达到了类似定时任务执行的效果 值得借鉴 好处是时间间隔可控，这里是续命逻辑，通过类似定时任务逻辑达到续命的目的
                        }
                    }
                });
            }
        }, internalLockLeaseTime / 3 【值为10】, TimeUnit.MILLISECONDS);

        if (expirationRenewalMap.putIfAbsent(getEntryName(), task) != null) {
            task.cancel();
        }
    }


```

解锁逻辑：

```
 //解锁
redissonLock.unlock();
-----> 【0】
   public void unlock() {
        Boolean opStatus = get(unlockInnerAsync(Thread.currentThread().getId())/**【1】*/);
        if (opStatus == null) {
            throw new IllegalMonitorStateException("attempt to unlock lock, not locked by current thread by node id: "
                    + id + " thread-id: " + Thread.currentThread().getId());
        }
        if (opStatus) {
            cancelExpirationRenewal();
        }

//        Future<Void> future = unlockAsync();
//        future.awaitUninterruptibly();
//        if (future.isSuccess()) {
//            return;
//        }
//        if (future.cause() instanceof IllegalMonitorStateException) {
//            throw (IllegalMonitorStateException)future.cause();
//        }
//        throw commandExecutor.convertException(future);
    }
----->【1】
protected RFuture<Boolean> unlockInnerAsync(long threadId) {
        return commandExecutor.evalWriteAsync(getName(), LongCodec.INSTANCE, RedisCommands.EVAL_BOOLEAN,
                "if (redis.call('exists', KEYS[1]) == 0) then " +  // key不存在，已经解锁了
                    "redis.call('publish', KEYS[2], ARGV[1]); " +  // 发布消息，通知解锁，【解锁消息】
                    "return 1; " +
                "end;" +
                "if (redis.call('hexists', KEYS[1], ARGV[3]) == 0) then " + // 这个锁是不是当前线程加的 不是 返回nil
                    "return nil;" +
                "end; " +
                "local counter = redis.call('hincrby', KEYS[1], ARGV[3], -1); " +  // 是当前线程  这个是重入逻辑，重入次数减1
                "if (counter > 0) then " +                  // 重入续命
                    "redis.call('pexpire', KEYS[1], ARGV[2]); " +
                    "return 0; " +
                "else " +                           // counter为0 删除key，发布消息
                    "redis.call('del', KEYS[1]); " +
                    "redis.call('publish', KEYS[2], ARGV[1]); " +
                    "return 1; "+
                "end; " +
                "return nil;",
                Arrays.<Object>asList(getName(), getChannelName()), LockPubSub.unlockMessage, internalLockLeaseTime, getLockName(threadId));

    }


----->订阅监听【7】
public class LockPubSub extends PublishSubscribe<RedissonLockEntry> {

    public static final Long unlockMessage = 0L;

    @Override
    protected RedissonLockEntry createEntry(RPromise<RedissonLockEntry> newPromise) {
        return new RedissonLockEntry(newPromise);
    }
    // 订阅的消息 消费逻辑
    @Override
    protected void onMessage(RedissonLockEntry value, Long message) {
        if (message.equals(unlockMessage)) {
```

```java
                value.getLatch().release();  // 信号量解锁唤醒

                while (true) {
                    Runnable runnableToExecute = null;
                    synchronized (value) {
                        Runnable runnable = value.getListeners().poll();
                        if (runnable != null) {
                            if (value.getLatch().tryAcquire()) {
                                runnableToExecute = runnable;
                            } else {
                                value.addListener(runnable);
                            }
                        }
                    }

                    if (runnableToExecute != null) {
                        runnableToExecute.run();
                    } else {
                        return;
                    }
                }
            }
        }

}
```