首先是明确一些基础概念：

- 一个钱包地址，在一个区块链上发起一笔交易，都是有一个nonce编号，这个nonce编号是递增的，不能跳过某一个值。
- 一个区块中，可以打包一个钱包的多个交易，但所有nonce都是递增且不同的，同样不能跳过某个值。多个交易的最小nonce必须是基于用户在区块链上的最后一次交易的nonce加1。

当前问题：

在当前集群或者单机多实例的node服务部署下，同一时间发起的交易的nonce出现了重复的值，因此出错。

- 难点1，不能是简单的靠一个单机或主从的数据库或redis维护nonce。并发的请求递增这个nonce，因为某一请求有可能失败而链上没有递增。
- 难点2，数据库或redis里保存的nonce，维护入口有两个，一个入口是每个请求来递增nonce，另一个是要同步链上最新的nonce。这两个入口维护的值是冲突的。

目前想到了几种解决方案：

最简单的办法是单机单进程去跑Node服务，只在这一个进程里解决多个请求的并发问题。——稳定性差，并发性差，cpu使用率低。

第二种方案是集群+单一任务队列+单机单进程执行的形式，集群往队列里赛，单机单进程的服务去批量执行。——交互实现上比较繁琐，且单机单进程去执行。一旦单机服务出问题，会很严重。

第三个方案是集群多进程但每个进程对应一个钱包地址，去执行合约，这样只在当前进程中管理多个请求的并发处理即可——这个方案要配置的钱包地址太多了，且无法不能支持伸缩扩容。

第四个方案：集群+单机或主从redis，这个方案要解决的难点就是上面的难点1和难点2。还需要进一步研究。——比第二种方案并发数更高，交互不会太复杂。比第三种方案无须多钱包，可支持扩容。但一旦有失败，就可能导致同一时间的所有需求都失败。

redis key：
nonce：数字编号，记录当前
transactionLock：交易锁
syncLock：同步锁

加锁逻辑：

锁1：交易锁

来自多个集群或进程的多个请求，每个请求都尝试获取锁，

如果获取到锁，那么

判断链同步锁是否处于锁定状态，如果处于锁定状态，那么循环等待同步锁解锁。

内部逻辑，执行nonce+1，并存入nonce，解锁。

如果没抢到锁，那么循环等待并重新抢锁。

锁2：同步锁

如果任意一笔交易执行失败（TIMEOUT 或 NONCE_EXPIRED），那么获取同步锁，如果获取到锁，那么从链上获取最新的nonce，存入nonce，解锁。

如果没抢到锁，那么取消。多个失败时，同一时间只需要有一个请求同步链即可。

合约执行错误的情形：

重复的nonce 报错：

e.code === "NONCE_EXPIRED"

nonce 过大：

e.code === "TIMEOUT"

合约执行出错：会增加nonce，但合约方法无法成功，也不应该重试。

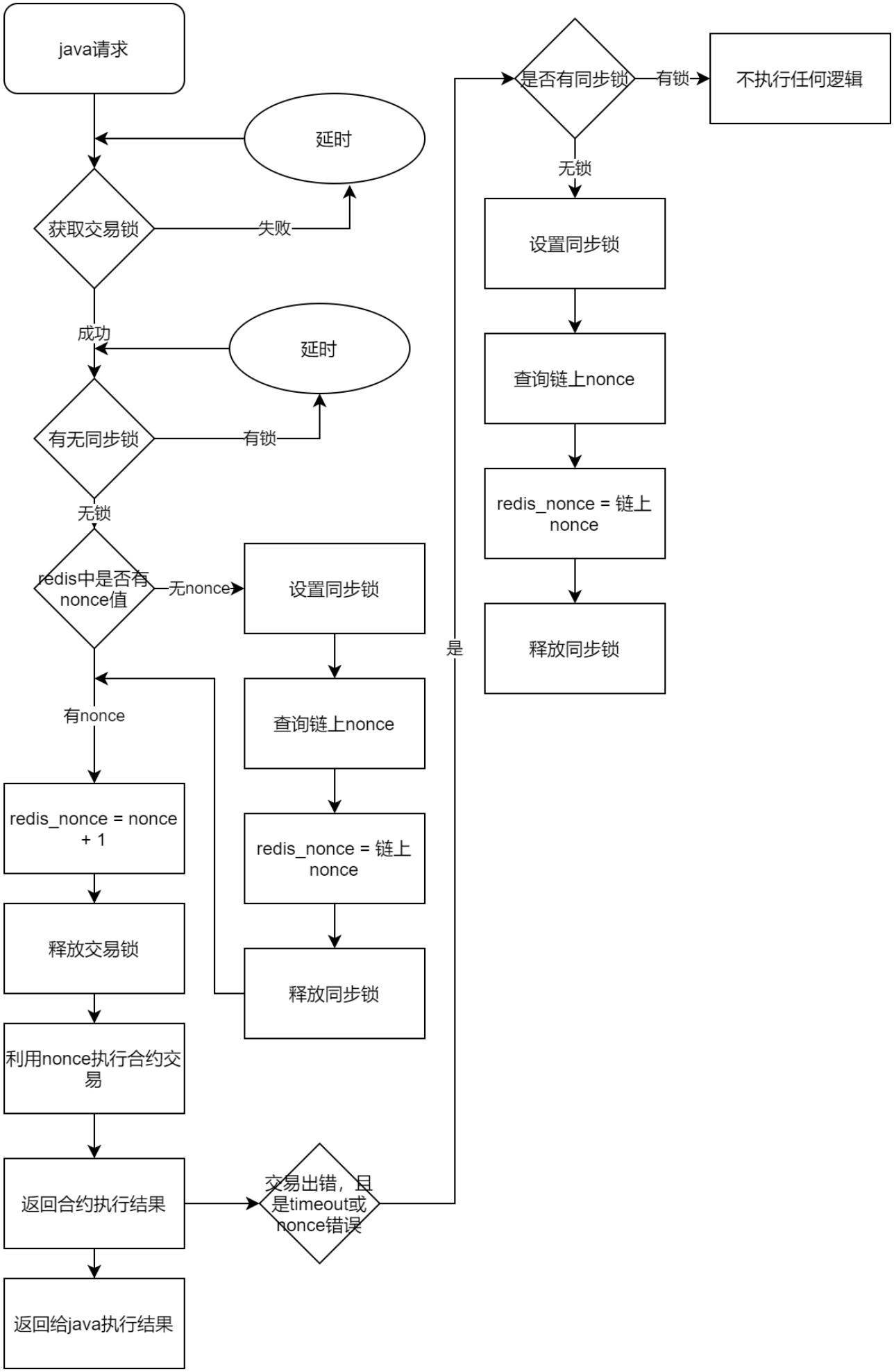
e.code === "CONTRACT_TRANSACTION_ERROR" // 自定义的

e.code === "REPLACEMENT_UNDERPRICED"

e.code === "SERVER_ERROR"

目前设计的方案的流程图：

注：redis里有nonce、transactionLock、syncLock 3个值存储。


java请求
延时
获取交易锁
失败
成功
延时
有无同步锁
有锁
无锁
redis中是否有nonce值
无nonce
设置同步锁
查询链上nonce
redis_nonce = 链上nonce
释放同步锁
有nonce
redis_nonce = nonce + 1
释放交易锁
利用nonce执行合约交易
返回合约执行结果
交易出错，且是timeout或nonce错误
是
返回给java执行结果
是否有同步锁
有锁
不执行任何逻辑
无锁
设置同步锁
查询链上nonce
redis_nonce = 链上nonce
释放同步锁