彻底解密企业级分布式事务设计与实践（上）
打造哲学本质篇

主讲人：孙玄

2020.07.07

我的职业成长路线

奈学教育

奈学教育

创始人&CEO

转 转

首席架构师
技术委员会主席
大中台技术负责人

58集团

技术委员会主席
高级系统架构师

百度

资深研发工程师

毕业

浙江大学

擅长领域

架构设计、大数据
机器学习、技术管理等

对外分享

业界顶级大会
百万年薪架构
直播大课品牌创始人

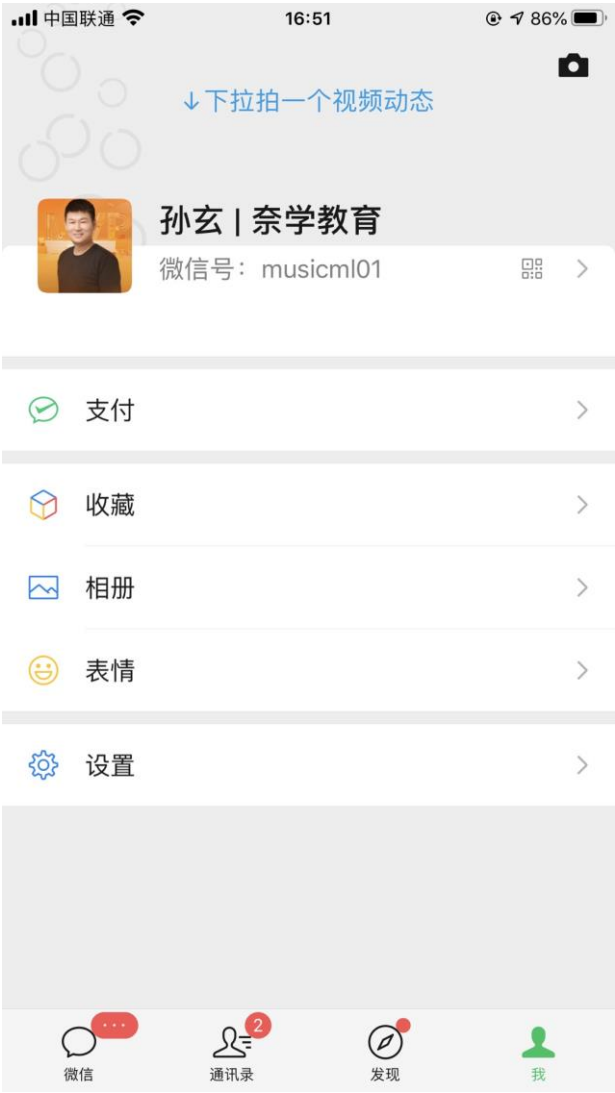
目录

NX 奈学教育

01.企业级分布式事务的本质（同步/异步业务场景）

企业级分布式事务的本质

- 一次请求涉及数据分布多个存储系统
  - 多个DB
  - DB和Redis
  - DB和MQ
  - ……
- 业务场景容忍度
  - 拼多多购买商品
    - 商品、订单、支付
  - 朋友圈发布信息
    - 发布信息、朋友圈的小红点提升



企业级分布式事务的本质

- 业务场景分类
  - 拼多多购买商品
    - 同步场景
      - 减商品库存、建立订单
      - 前台支付
    - 异步场景
      - 前台超时未支付

NX 奈学教育

02.企业级分布式事务设计普适方法论（DB/MQ/Redis等实践方法）

企业级分布式事务设计普适方法论（DB/MQ/Redis等实践方法）

- 分布式事务普适方法论
  - 拆分
    - 分布式事务->长事务
    - 本地事务->短事务
    - 长事务**拆分**多个短事务
  - 补偿
    - A->B->C
      - A、B成功，C失败
        - C不需要补偿
        - 补偿B和A（**逆向**）

企业级分布式事务设计普适方法论（DB/MQ/Redis等实践方法）

- 分布式事务普适方法论

```
public boolean lock(int ttl)
    throws RuntimeException {
    this.ttl = ttl;
    // 锁不存在时：上锁并过期时间，最后跳出。
    Long result = redisTemplate.execute(new RedisScript<Long>() {
        @Override
        public String getSha1() {
            return SCRIPT_LOCK_SHA1;
        }

        @Override
        public Class<Long> getResultType() {
            return Long.class;
        }

        @Override
        public String getScriptAsString() {
            return SCRIPT_LOCK;
        }

    }, Collections.singletonList(lockKey), "1", String.valueOf(ttl));
    if (SUCCESS.equals(result)) {
        return true;
    }
    return false;
}
```

NX 奈学教育

03.企业级异步业务场景分布式事务设计与实践（上）【异步/事务/本地/2PC/3PC】

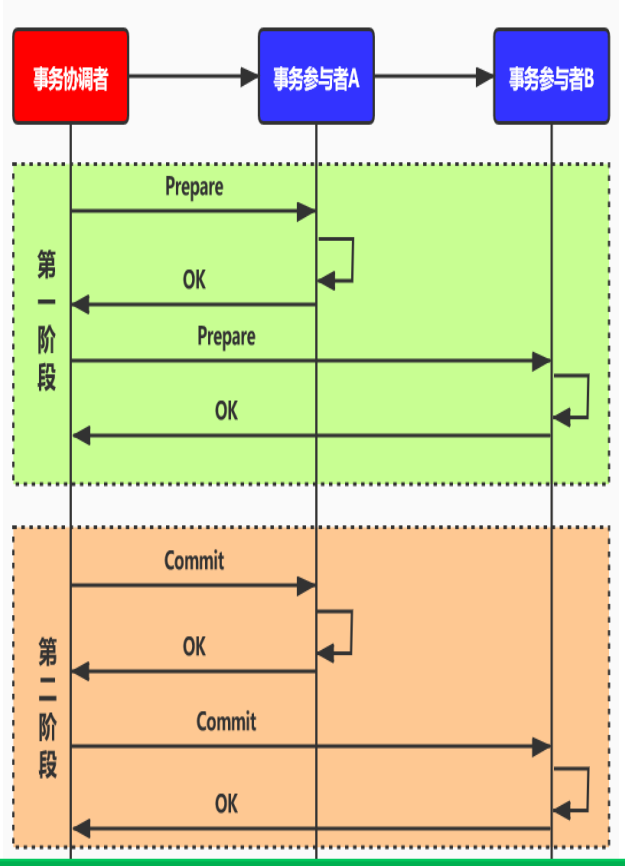
企业级企业级异步业务场景分布式事务设计与实践（上）

- 分布式事务
  - 拼多多购买商品
    - 同步场景
      - 减商品库存、建立订单
      - 前台支付
    - 异步场景
      - 前台超时未支付

- 分布式事务
  - 异步场景
    - 步骤一：下订单
    - 步骤二：发送订单未支付延时消息
  - 常见做法
    - 做法一：本地事务
      - OK?
    - 做法二：时序调整
      - OK?

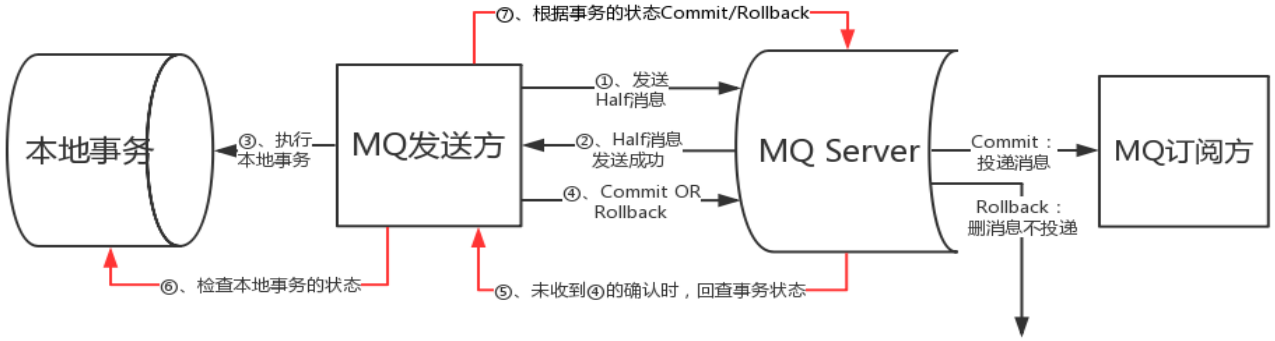
如何设计和落地？

- 分布式事务
  - 2PC
    - 案例
      - 组织爬山
    - 过程
      - **应用程序**发起事务commit请求
      - **事务协调者**发起prepare投票
      - **事务参与者**都同意后，事务协调者再发起commit
      - commit过程出现宕机等异常，服务重启后，再次进行commit



上述问题如何实践？

- 分布式事务
  - 2PC落地实践
    - 业务处理回查
      - 业务方提供下单成功的回查接口
    - 处理过程



存在什么问题？如何实施通用的2PC？

企业级企业级异步业务场景分布式事务设计与实践（上）

奈学教育

- 分布式事务

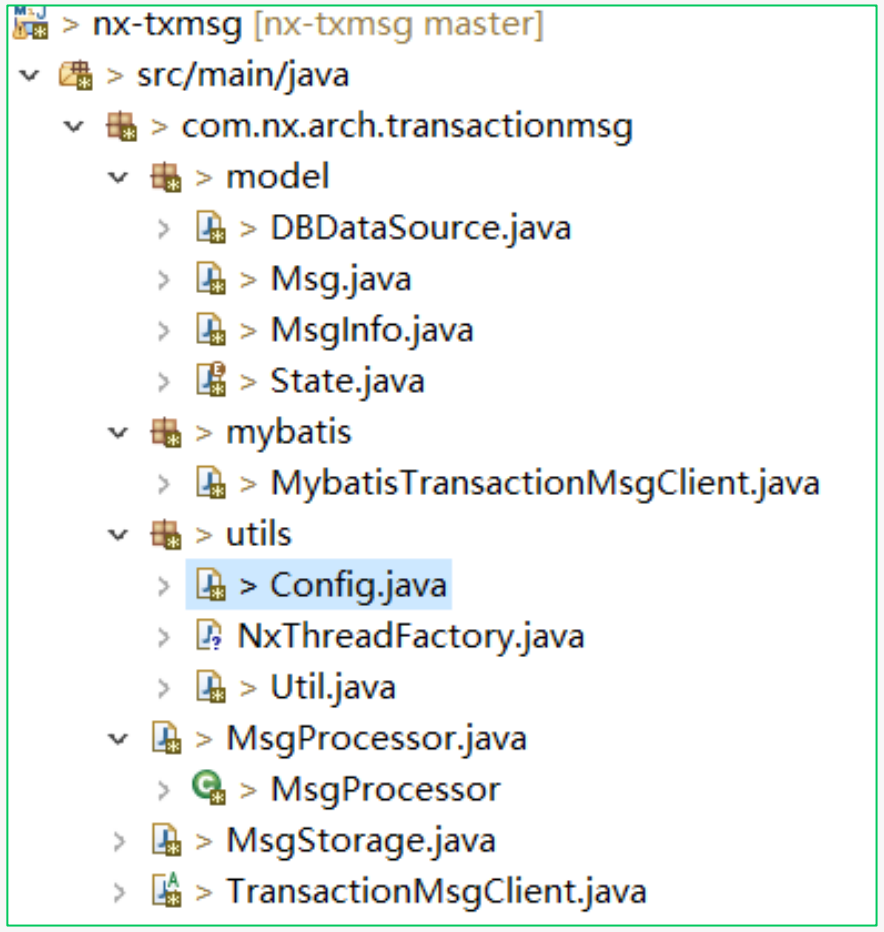
- 分布式事务设计
  - 第二阶段
    - 发送端消息不幂等
      - At least once
    - 订单业务逻辑服务冗余部署
      - 重复提交

一切脱离代码谈架构都是要流氓，代码如何落地？

企业级企业级异步业务场景分布式事务设计与实践（上）

奈学教育

项目结构



事务相关表

```
CREATE TABLE `msg_
  `id` bigi
  `content`
  `topic` c
  `tag` cha
  `status`
  `create_t
  `delay` i
  PRIMARY K
) ENGINE=In
```

```
public class MsgInfo {
```

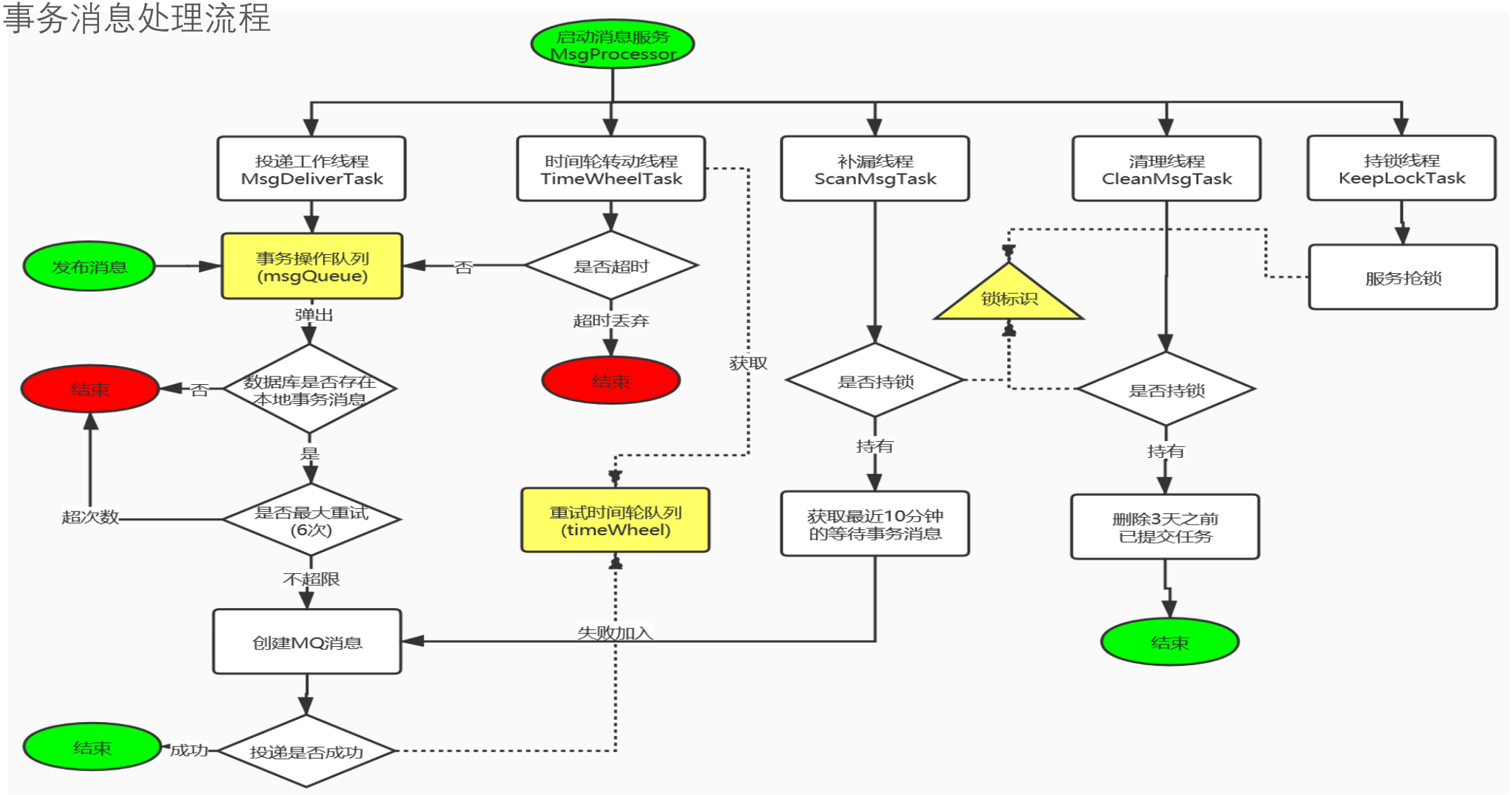
```
 * @类描述 事务操作实体
public class Msg {
     * 主键
    private Long id;

     * mysql url 分库分表，所以需要id和url映射
    private String url;

     * 已经处理次数
    private int haveDealedTimes;

     * 创建时间
    private long createTime;

     * 下次超时时间
    private long nextExpireTime;
```

```
     * 延迟时间(单位：s)
    private int delay;
```

企业级企业级异步业务场景分布式事务设计与实践（上）

奈学教育

- 事务消息处理流程



- 事务投递任务：将已提交的事务消息投递到MQ

```
class MsgDeliverTask implements Runnable {
    // 3.2、投递事务消息
    Message mqMsg = buildMsg(msgInfo);
    log.debug("will sendMsg {}", mqMsg);
    SendResult result = producer.send(mqMsg);
    log.info("msgId {} topic {} tag {} sendMsg result {}", msgInfo.getId(), mqMsg.getTopic(), mqMsg.getTags(), result);
    if (null == result || result.getSendStatus() != SendStatus.SEND_OK) {
        // 投递失败，重入时间轮
        if (dealedTime < maxDealTime) {
            long nextExpireTime = System.currentTimeMillis() + timeOutData[dealedTime];
            msg.setNextExpireTime(nextExpireTime);
            timeWheel.put(msg);
            // 这里可以优化，因为已经确认事务提交了，可以从DB中拿到了
            log.debug("put msg in timeWhellQueue {} ", msg);
        }
    } else if (result.getSendStatus() == SendStatus.SEND_OK) {
        // 投递成功，修改数据库的状态(标识已提交)
        int res = msgStorage.updateSendMsg(msg);
        log.debug("msgId {} updateMsgStatus success res {}", msgInfo.getId(), res);
    }
}
                        // 3.1、加入时间轮转动队列:重试投递
                        long nextExpireTime = System.currentTimeMillis() + timeOutData[dealedTime];
                        msg.setNextExpireTime(nextExpireTime);
                        timeWheel.put(msg);
                        log.debug("put msg in timeWhellQueue {} ", msg);
                    }
                } else {
```

- 时间轮转动任务：重试投递失败未超时事务消息

```
 * @类描述 时间轮转动线程：重试投递失败的事务操作(未超时)
class TimeWheelTask implements Runnable {
    @Override
    public void run() {
        try {
            if (state.get().equals(State.RUNNING)) {
                long cruTime = System.currentTimeMillis();
                Msg msg = timeWheel.peek();
                // 拿出来的时候有可能还没有超时
                while (msg != null && msg.getNextExpireTime() <= cruTime) {
                    msg = timeWheel.poll();
                    log.debug("timeWheel poll msg ,return to msgQueue {}", msg);
                    // 重新放进去
                    msgQueue.put(msg);
                    msg = timeWheel.peek();
                }
            }
        } catch (Exception ex) {
            log.error("pool timequeue error", ex);
        }
    }
}
```

- 事务消息清理任务：删除三天前发送成功的消息

```
 * @类描述 事务消息删除线程：删除三天之前的发送成功的消息
class CleanMsgTask implements Runnable {
    @Override
    public void run() {
        if (state.get().equals(State.RUNNING)) {
            log.debug("DeleteMsg start run");
            try {
                Iterator<DataSource> it = msgStorage.getDataSourcesMap().values().iterator();
                while (it.hasNext()) {
                    DataSource dataSrc = it.next();
                    boolean canExe = holdLock;
                    if (canExe) {
                        log.info("DeleteMsgRunnable run ");
                        int count = 0;
                        int num = config.deleteMsgOneTimeNum;
                        while (num == config.deleteMsgOneTimeNum && count < maxDealNumOneTime) {
                            try {
                                num = msgStorage.deleteSendedMsg(dataSrc, config.deleteMsgOneTimeNum);
                                count += num;
                            } catch (SQLException e) {
                                log.error("deleteSendedMsg fail ", e);
                            }
                        }
                    }
                }
            } catch (Exception ex) {
                log.error("delete Run error ", ex);
            }
        }
```

企业级企业级异步业务场景分布式事务设计与实践（上）

奈学教育

- 事务消息补漏任务：扫描最近10分钟未提交事务消息，防止异常场景消息丢失

```
public void run() {
    if (state.get().equals(State.RUNNING)) {
        log.debug("SchedScanMsg start run");
        Iterator<DataSource> it = msgStorage.getDataSourcesMap().values().iterator();
        while (it.hasNext()) {
            DataSource dataSrc = it.next();
            boolean canExe = holdLock;
            if (canExe) {
                log.info("SchedScanMsgRunnable run");
                int num = limitNum;
                int count = 0;
                while (num == limitNum && count < maxDealNumOneTime) {
                    try {
                        List<MsgInfo> list = msgStorage.getWaitingMsg(dataSrc, limitNum);
                        num = list.size();
                        if (num > 0) {
                            log.debug("scan db get msg size {} ", num);
                        }
                        count += num;
                        for (MsgInfo msgInfo : list) {
                            try {
                                Message mqMsg = buildMsg(msgInfo);
                                SendResult result = producer.send(mqMsg);
                                log.info("msgId {} topic {} tag {} sendMsg result {}", msgInfo.getId(), mqMsg.getTopic(), mqMsg.getTags(), result);
                                if (result != null && result.getSendStatus() == SendStatus.SEND_OK) {
                                    // 修改数据库的状态
                                    int res = msgStorage.updateMsgStatus(dataSrc, msgInfo.getId());
                                    log.debug("msgId {} updateMsgStatus success res {}", msgInfo.getId(), res);
                                }
                            } catch (Exception e) {
                                log.error("SchedScanMsg deal fail", e);
                            }
                        }
                    } catch (SQLException e) {
                        log.error("getWaitMsg fail", e);
                    }
                }
```

百万架构师VIP学员获取全部企业级可运行代码！！

- 思考

  - MQ下游消费服务如何保证事务的成功处理?
  - 异步场景分布式事务存在性能瓶颈吗？如何提升性能?

明天直播课程继续讲解！！！

要点回顾

奈学教育

01 企业级分布式事务的本质（同步/异步业务场景）

02 企业级分布式事务设计普适方法论（DB/MQ/Redis等实践方法）

03 企业级异步业务场景分布式事务设计与实践（上）【异步/事务/本地/2PC/3PC】

NX 奈学教育

欢迎关注本人公众号
“架构之美”