

# 业务需求汇总

## 部分申诉队列归因标签不发信

由于发信涉及到的模块比较多，在后面决定直接新建两个不发信的配置，50002268-申诉成功 50002269-申诉失败 需要对以下内容进行配置

Java

复制代码

```
1 @ApolloJsonValue("${note_feedback_revoke_succ_labels: []}")
   private List<Long> noteFeedbackRevokeSuccLabels;

//申诉失败标签ID
@ApolloJsonValue("${note_feedback_revoke_fail_labels: []}")
private List<Long> noteFeedbackRevokeFailLabels;
```

这个时候又牵扯到另外一个问题，由于queryAffectLabel会直接返回noteFeedbackRevokeFailLabels和noteFeedbackRevokeSuccLabels，就导致新添加一个标签之后处置两遍，这时候就需要把发信和不发信的标签分开，通过从队列配置中读取是否为发信标签来返回对应的申诉成功或申诉失败标签

通过在队列配置中配置enableLabelSending来进行控制

Java

复制代码

```
1 boolean isLabelSent= Boolean.parseBoolean(
    queueConfig.getBackendConfig().getOrDefault("enableLabelSending","true"));
```

Java

复制代码

```
1 private DisposalTaskResponseB0 revokeDisposalAction(DisposalTaskRequestB0 request,
    DisposalTaskContext context) throws Exception {
    DisposalTaskResponseB0 responseB0 = new DisposalTaskResponseB0();
    List<Long> remainLabels =
    JsonSerializer.deserialize(MapUtils.getString(request.getParams(),
    FeedbackConstant.REMAIN_LABELS), new TypeReference<List<Long>>() {
    }); // 保留的标签
    List<Long> revokeLabels =
    JsonSerializer.deserialize(MapUtils.getString(request.getParams(),
    FeedbackConstant.REVOKE_LABELS), new TypeReference<List<Long>>() {
    }); // 撤销的标签

    // 进行拦截校验，撤销标签 和 保留标签 有交集
    boolean isIntersection = !CollectionUtils.isEmpty(remainLabels)
        && !CollectionUtils.isEmpty(revokeLabels)
        && !CollectionUtils.intersection(remainLabels, revokeLabels).isEmpty();
    if (isIntersection) {
        throw new RuntimeException(
            String.format("remainLabels and revokeLabels isIntersection
            remainLabels:%s, revokeLabels:%s",
            remainLabels, revokeLabels));
    }
}
```

```

JSONObject.toJSONString(remainLabels),JSONObject.toJSONString(revokeLabels)));
    }
    // 申诉成功/失败, 兼容原有逻辑, 先根据申诉标签判断
    List<Long> appealLabels = DisposalTaskRequestUtil.filterLabelId(request,
ProcessContentTypeEnum.COMMON_APPEAL.name());

    //从配置中读取是否要发信
    QueueConfig queueConfig =
queueConfigService.getQueueConfigDetail(request.getCategoryType(),
request.getSourceType(), true);
    boolean sendMsgWhenTag = true;
    if (queueConfig.getBackendConfig() != null
        && queueConfig.getBackendConfig().containsKey("FeedBackSendMsgFlag")) {
        sendMsgWhenTag =
Boolean.parseBoolean(queueConfig.getBackendConfig().get("FeedBackSendMsgFlag"));
    }

    // 申诉成功
    if (CollectionUtils.isNotEmpty(appealLabels)
        &&
        (Sets.newHashSet(noteFeedbackSendRevokeSuccLabels).containsAll(appealLabels) ||
        Sets.newHashSet(noteFeedbackNoSendRevokeSuccLabels).containsAll(appealLabels))) {
        // 新逻辑, 根据传参撤销 (即本次处置下所有标签)
        if (CollectionUtils.isNotEmpty(revokeLabels)) {
            CommonProcessResponse revokeResp = revokeCommonNote(request,
revokeLabels, remainLabels);
        } else {
            log.info("NoteRiskNoChangeFeedbackTaskService.revokeDisposalAction:
feedback success but revoke label id is empty");
        }
        //申诉归因处置
        CommonProcessResponse commonResp = processRevokeCommonAppeal(request,
appealLabels,sendMsgWhenTag);
    }
    // 申诉失败
    else if (CollectionUtils.isNotEmpty(appealLabels)
        &&
        (Sets.newHashSet(noteFeedbackSendRevokeFailLabels).containsAll(appealLabels) ||
        Sets.newHashSet(noteFeedbackNoSendRevokeFailLabels).containsAll(appealLabels))) {
        // 保留部分标签
        if (CollectionUtils.isNotEmpty(remainLabels)) {
            // 查处置中心是否发信透传申诉成功/失败
            boolean isLabelSent= Boolean.parseBoolean(
queueConfig.getBackendConfig().getOrDefault("enableLabelSending","true"));
            appealLabels = queryAffectLabel(remainLabels,request,isLabelSent);
            // 存在申诉失败无需撤销的情况
            if (CollectionUtils.isNotEmpty(revokeLabels)){
                // 按传参撤销部分标签
                CommonProcessResponse revokeResp = revokeCommonNote(request,

```

```

revokeLabels, remainLabels));
    }
    } // remainLabels为空说明是原有逻辑, 不做任何撤销
    // 申诉归因处置
    CommonProcessResponse commonResp = processRevokeCommonAppeal(request,
        appealLabels, sendMsgWhenTag);
    }
    else{
        throw new RuntimeException("feedback label id config is error");
    }
    responseB0.setResult(ResultUtils.successHandleMessage());
    return responseB0;
}

```

### 申诉复审队列新增降级按钮需求

#### 降级逻辑

1. 降级归因提交时, 打标申诉成功归因标签撤销进审标签处罚+新降级标签处置
  - a. 【降级30天禁言】: 打标申诉成功归因 (不发信) 50002241+撤销进审标签处罚+50002295-降级30天禁言
  - b. 【降级30天0级】: 打标申诉成功归因 (不发信) 50002241+撤销进审标签处罚+50002296-降级30天0级

UserRiskFeedbackTaskService进行处理

原有逻辑是

获取到申述标签之后, 使用这个申述标签对申述进行归因处置

再进行判断, 如果为申述成功的标签, 进行标签的一个撤销, 申述失败, 则保留标签

现在逻辑:

获取到申述标签如果是50002296/50002295, 则使用50002241进行归因处置

其他逻辑相同

分为申述标签和用户标签, 分为两套逻辑

代码:

Java

复制代码

```

1 private DisposalTaskResponseB0 revokeDisposalAction(DisposalTaskRequestB0 request,
    DisposalTaskContext context) throws Exception {
    DisposalTaskResponseB0 responseB0 = new DisposalTaskResponseB0();
    // 申诉标签
    List<Long> labels = DisposalTaskRequestUtil.filterLabelId(request,
        ProcessContentTypeEnum.COMMON_APPEAL.name());
    // 用户标签
    List<Long> userLabels = DisposalTaskRequestUtil.filterLabelId(request,
        ProcessContentTypeEnum.USER.name());

    // 从配置中读取是否要发信
    QueueConfig queueConfig =

```

```

queueConfigService.getQueueConfigDetail(request.getCategoryType(),
request.getSourceType(), true);
    boolean sendMsgWhenTag = true;
    if (queueConfig.getBackendConfig() != null
        && queueConfig.getBackendConfig().containsKey("FeedBackSendMsgFlag")) {
        sendMsgWhenTag =
Boolean.parseBoolean(queueConfig.getBackendConfig().get("FeedBackSendMsgFlag"));
    }
    //走申诉标签逻辑
    if(CollectionUtils.isNotEmpty(labels)){
        processCommonAppeal(request, sendMsgWhenTag, labels);
        //新逻辑：只要两组标签有交集，就判定为申诉成功
        if (!Collections.disjoint(userFeedbackRevokeSuccLabels, labels)) {
            //撤销标签
            List<Long> revokeLabels=extractRevokeLabels(request);
            //用户撤销
            CommonProcessResponse userResp = revokeCommonUser(request, revokeLabels);
        }
        //走用户标签逻辑
    } else if (CollectionUtils.isNotEmpty(userLabels)) {
        //新逻辑，传入新的不发信标签
        processCommonAppeal(request, sendMsgWhenTag, userNewFeedbackRevokeSuccLabels);
        if (Sets.newHashSet(userFeedbackDemotionSuccLabels).containsAll(userLabels)) {
            //撤销标签
            List<Long> revokeLabels=extractRevokeLabels(request);
            //用户撤销
            CommonProcessResponse userResp = revokeCommonUser(request, revokeLabels);
            //用户处置
            processCommonUser(request, sendMsgWhenTag);
        }
    } else {
        throw new RuntimeException("feedback label id is empty");
    }
    responseB0.setResult(ResultUtils.successHandleMessage());
    return responseB0;
}

```

### 申诉撤销标签发信查询接口更换

目前申诉去掉改标逻辑后，查询的是实时处置的接口判断发申诉成功还是申诉失败的信，该接口已经下线的标签查不到，需要更换查询接口（改成查用户被处置时候身上真实标签）

接口：

替换原有旧接口：

`processingCenterService.estimateRevokeTagAffect`

1;

申诉失败的条件：根据保留的标签，查询新的接口，返回的消息结果中，如果存在 hasMsgInfo 和 status 为 valid，则表明存在有处置力度的动作；维持申诉失败；

2: 如果保留的标签, 返回结果为空 或者 返回的status 为 invalid 则 申诉成功

部分申诉队列归因标签不发信

#### 🔗 客服转复审申诉队列

背景: 新增客服到申诉的人审送审链路, 该链路进申诉的case打成功/失败标签 c端不发信。

需求: 需要针对客服专用人审队列支持打申诉成功/失败后不发信 (目前的常规申诉是写死在代码里的)。

- NoteRiskNoChangeFeedbackTaskService
- UserNewFeedBackTaskService
- UserRiskFeedbackTaskService

对于审核任务, disposalTask只是执行外部逻辑, 真正执行逻辑的是disposalAction

通过从queueConfig.getBackendConfig()读取配置来控制是否发信, key="FeedBackSendMsgFlag", 默认设置为发信,没读取到也是发信, 只有读到false才是不发信

```
Java ▼ 复制代码

1 @Override
   protected DisposalTaskResponseB0 disposalAction(DisposalTaskRequestB0 request,
   DisposalTaskContext context) throws Exception {
       DisposalTaskResponseB0 responseB0 = new DisposalTaskResponseB0();

       //从配置中读取是否要发信
       QueueConfig queueConfig =
       queueConfigService.getQueueConfigDetail(request.getCategoryType(),
       request.getSourceType(), true);
       boolean sendMsgWhenTag = true;
       if (queueConfig.getBackendConfig() != null
           && queueConfig.getBackendConfig().containsKey("FeedBackSendMsgFlag")) {
           sendMsgWhenTag =
           Boolean.parseBoolean(queueConfig.getBackendConfig().get("FeedBackSendMsgFlag"));
       }

       //用户处置
       CommonProcessResponse userResp = processCommonUser(request, sendMsgWhenTag);
       if (Objects.isNull(userResp)) {
           return DataUtils.errorReturn(new DisposalTaskResponseB0(),
           FeedbackErrorInfo.FEEDBACK_USER_PROCESS_ERROR);
       }

       //申诉归因处置
       CommonProcessResponse commonResp = processCommonAppeal(request, sendMsgWhenTag);
       if (Objects.isNull(commonResp)) {
           return DataUtils.errorReturn(new DisposalTaskResponseB0(),
           FeedbackErrorInfo.FEEDBACK_COMMON_PROCESS_ERROR);
       }


       responseB0.setResult(ResultUtils.successHandleMessage());
       return responseB0;
   }
```

举报业务apollo配置兼容队列配置

### 【业务】恶劣评论互动队列处置逻辑修复

在ecology\_user\_reviews队列中实现过滤，过滤逻辑：

- 用户在进入审的时候，判断当前用户“最新一条评论的发布时间”和“这个用户上一次在本队列的处置时间”，如果最新一条评论的发布时间 早于 用户上次处置时间，则该用户不送审

这个用户上一次在本队列的处置时间 接口：`getProcessRecordV2`  David

```
Java   复制代码

1    protected void doConsumeMessage(MessageExt msg, ConsumeContext context) throws
    Exception {
2        log.info("UserMarkListener doConsumeMessage start, msg={}" +
        JSONObject.toJSONString(msg));
3        UserMarkMessage userMarkMessage = null;
4        String bodyStr = "";
5        try {
6            // 这个转化会将字段下划线转化为驼峰形式
7            bodyStr = new String(msg.getBody(), StandardCharsets.UTF_8);
8            userMarkMessage = JSON.parseObject(msg.getBody(), UserMarkMessage.class);
9            log.info("UserMarkListener:get message:user_id={},message={}",
            userMarkMessage.getUserId(),
10                JSONObject.toJSONString(userMarkMessage));
11            // 命中了批量任务
12            if (StringUtils.equalsIgnoreCaseIgnoreCase(userMarkMessage.getTaskType(),
            BATCH_TASK_TYPE)) {
13                this.batchUserConsumer(userMarkMessage);
14                return;
15            }
16            // 新增过滤逻辑
17            if(userMarkFilterQueue.contains(userMarkMessage.getUserQueueType())
            &&!shouldSendToAudit(userMarkMessage.getUserId(), userMarkMessage.getUserQueueType())){
18                log.info("queue not sent to audit because latest comment time is earlier
            than last disposition time, userId:{},queue:{},userMarkMessage.getUserId(),
            userMarkMessage.getUserQueueType());
19                return;
20            }
21
22            saveUserQueueConsumerRecord(userMarkMessage.getUserId(), bodyStr, 1, "",
            userMarkMessage.getType());
23            doConsume(userMarkMessage);
24        } catch (Exception e) {
25            saveUserQueueConsumerRecord(userMarkMessage == null ? "" :
            userMarkMessage.getUserId(), bodyStr, 0,
26                e.getMessage(), userMarkMessage.getType());
27            throw new RuntimeException(e);
28        }
29    }
```

```
30
31 private boolean shouldSendToAudit(String userId, String auditQueueType) {
    log.info("Checking if should send to audit. userId: {}, auditQueueType: {}", userId,
auditQueueType);

    LocalDateTime latestCommentTime = getLatestCommentTime(userId, auditQueueType);
    if (latestCommentTime == null) {
        log.info("No latest comment time found for userId: {}, auditQueueType: {}, not
sending to audit", userId, auditQueueType);
        return false;
    }

    LocalDateTime lastDispositionTime = getLastDispositionTime(userId, auditQueueType);
    if (lastDispositionTime == null) {
        log.info("No last disposition time found for userId: {}, auditQueueType: {},
sending to audit", userId, auditQueueType);
        return true;
    }

    return !latestCommentTime.isBefore(lastDispositionTime);
}
32
```

通过把从Apollo配置中心配置的信息获取优化为从队列配置模版中获取，避免后期新增队列就要新增配置

Java

复制代码

```
1 QueueConfig queueConfig =
    queueConfigService.getQueueConfigDetail(request.getCategoryType(),
    request.getSourceType(), true);
2
3 //categoryType可以通过以下方式获取
4 String categoryType =
    queueConfigV2Helper.getCategoryTypeBySourceType(reportInfo.getQueueType());
```

apollo配置	这4个都通过 queueConfigService.compatibleConfigura tions进行兼容
reportInfoAddTaskOldSourceType	
reportInfoAddTaskSourceType	
report_add_redis_lock_source_type	
adduce_note_rights_center_action_label_mapping_ config	noteInfringement noteNotInfringement
rights_center_action_label_mapping_config	noteNotify noteVerify

adduce_user_rights_center_action_label_mapping_config	这四个都在 ReportNoteRightsCenterTaskService的 buildOtherParams中进行兼容
user_rights_center_action_label_mapping_config	
sourceTypeWhiteList	这4个都通过 queueConfigService.compatibleConfigurations进行兼容
skip_report_disposal_task_validate_queue_config	
query_report_condition_v2_config	
report_page_audit_by_condition_support_queue	

评论队列违规图片标注-存储与查询

评论有多图，评论被标注违规，需要查询到是哪一张图违规了

- 图片url+标签id +评论id落表 参考【机审证据标注】需求的结构

SQL

复制代码

```
1 CREATE TABLE `comment_image_tag_record` (  
2   `id` bigint(20) unsigned NOT NULL AUTO_INCREMENT COMMENT '主键id',  
3   `image_url` varchar(2048) NOT NULL DEFAULT '' COMMENT '图片URL',  
4   `label_id` varchar(255) NOT NULL DEFAULT '' COMMENT '处置标签',  
5   `comment_id` varchar(128) NOT NULL DEFAULT '' COMMENT '图片关联的评论ID',  
6   `create_time` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP COMMENT '创建时间',  
7   `update_time` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP  
   COMMENT '更新时间',  
8   `task_id` varchar(128) NOT NULL DEFAULT '' COMMENT '任务id',  
9   PRIMARY KEY (`id`),  
10  KEY `idx_update_time` (`update_time`),  
11  KEY `idx_comment_id` (`comment_id`)  
12 ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COMMENT='评论队列违规图片标注表'
```

- 与前端约定字段

放在DisposalTaskRequestBO的public Map<String, String> params，只传打勾的图片

约定map字段为imageURLs，直接把全部URL放进image\_url列中

在CommentVerifyService写具体逻辑，参考了 诈骗关联/多图评论标注 的代码

Java

复制代码

```
1 private void saveToManualInfo(DisposalTaskRequestBO request) {  
    try {  
        CommonSingleSqlInsertRequest insertRequest = new CommonSingleSqlInsertRequest();  
        insertRequest.setRecords(Lists.newArrayList());  
        CommonSingleSqlInsertResult insertResult = new CommonSingleSqlInsertResult();  
        insertRequest.getRecords().add(insertResult);  
        insertResult.setTableName(TABLE_COMMENT_IMAGE_TAG_RECORD);  
        Map<String, String> fieldMap = Maps.newHashMap();
```



```
String
labelId=request.getLabelIds().stream().map(String::valueOf).collect(Collectors.joining(",
"));

String imageURLs = request.getParams().get("imageURLs");

if(StringUtils.isBlank(imageURLs)){
    log.error("Parameter 'imageURLs' cannot be empty, request={}",request);
    return;
}

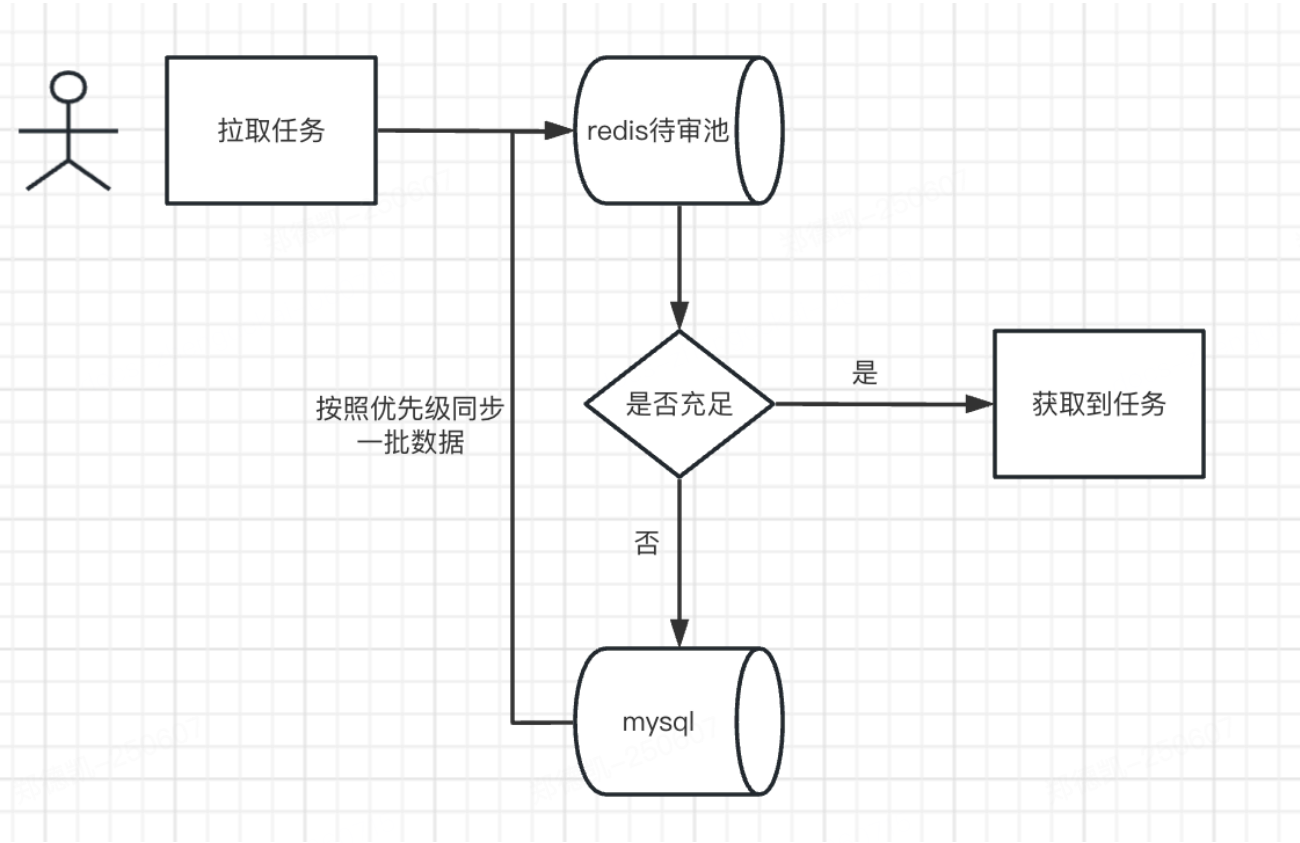
fieldMap.put("image_url", imageURLs);
fieldMap.put("label_id", labelId);
fieldMap.put("comment_id", request.getResourceId());
fieldMap.put("task_id", request.getTaskId());
insertResult.setFieldValueMap(fieldMap);

List<CommonSingleSqlInsertResult> responseResult =
erisClient.batchInsertCommonSingleRecord(insertRequest);
if (org.apache.commons.collections.CollectionUtils.isEmpty(responseResult) ||
Objects.isNull(responseResult.get(0))) {
    log.error("record insert to manual_info failed, request={}, rsp is null.",
insertRequest);
    return;
}
if (!responseResult.get(0).isSuccess()) {
    log.error("record insert to manual_info failed, request={}, errorMsg={}.",
insertRequest,
        responseResult.get(0).getMsg());
}
} catch (Exception e) {
    log.error(String.format("SaveToManualInfo failed, request=%s, errorMsg=%s.",
request,e.getMessage()), e);
}
}
```

## redis任务池重排序

# 1. 项目背景

目前人审队列存在redis缓存池，当更加高优的case进审时，此时是无法立刻拉取到这个高优的case的，需要等待redis待审池中的case全部审出才可以同步一批按照优先级排序的case进到redis中，导致无法保证高优case快速审出。



2. 整体设计

2.1 设计目标

可以根据队列配置完成分钟级别的redis待审池按优先级重排序。

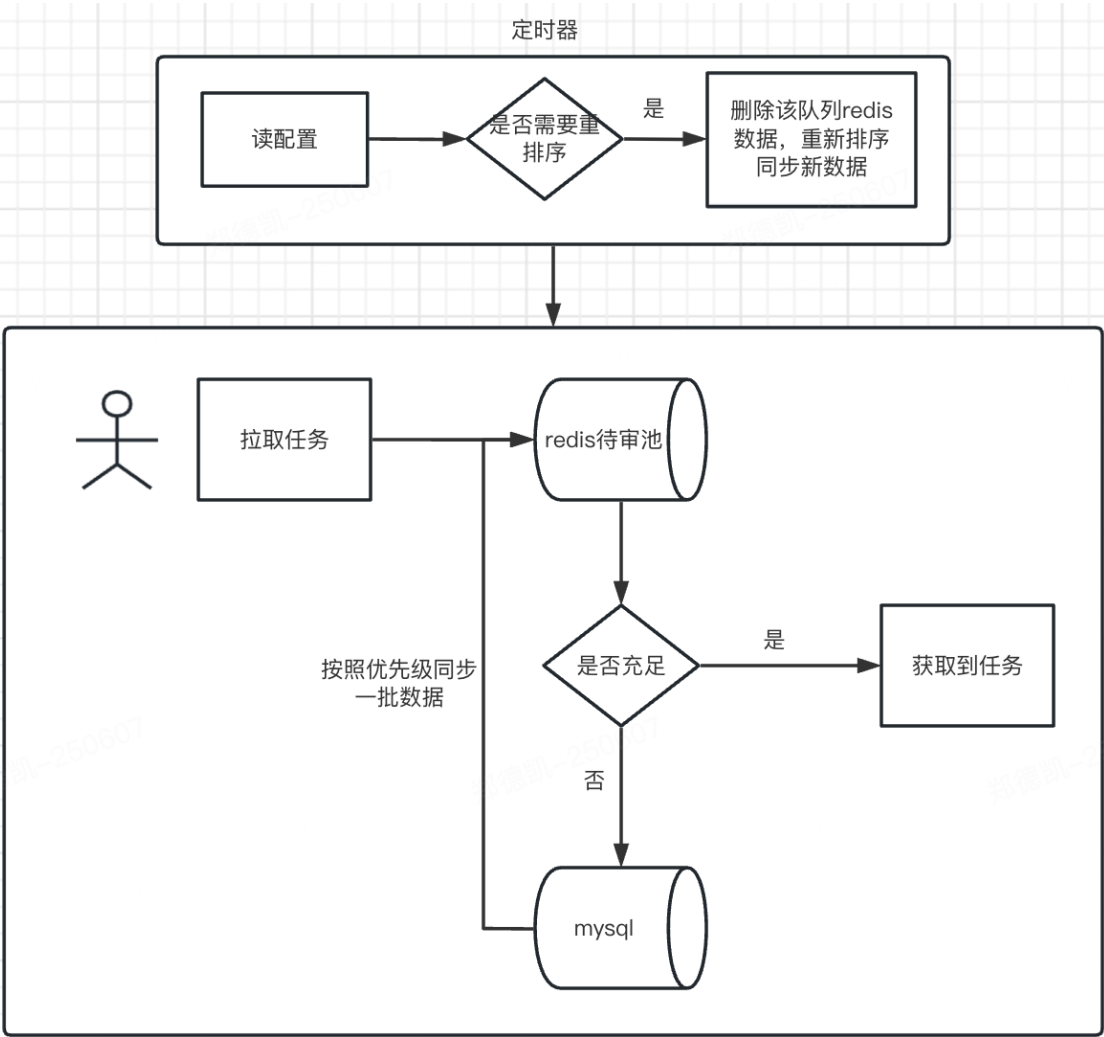
2.2 设计原则 & 思路

尽可能的保证高优先级的case先行审出，用户无感知redis待审池重排序。

2.3 技术选型

依赖公司提供的现有定时任务平台redschedule实现。

2.4 整体架构图



3. 详细设计

3.1 核心流程设计 & 模块拆解

- 1、队列配置表增加字段
  - 1) is\_sort int 判断该队列是否需要重排序
  - 2) interval\_time 重排序间隔时间【设定间隔时间为5的倍数】
- 2、读取队列配置缓存
- 3、新建job，5分钟执行一次，执行前判断前一次执行时间是否符合满足间隔时间，当某个队列进行重排序后将最新的执行时间记录【存redis】

3.2 实现说明

- 1、增加定时器，可按照业务需求配置哪个队列执行，执行间隔时间等
- 2、第一版方案是使用apollo配置维护白名单，后期看业务使用情况（有必要则在队列配置中增加配置【redis重排序】）。
- 3、在页面上增加按钮供研发使用，当遇到紧急情况研发可手动触发队列redis重排序

笔记预审处置动作


 [【技术方案】](#) [预审工具](#)  [云效平台](#)

Java  复制代码

```
1 MQ: note_pre_audit_result
2 tag: NOTE_PRE
3 key: ${task_id}
4
5 {
6     "task_id": "xxxxx", // 送审时的任务id
7     "status": "PASS" "REJECT", // PASS: 通过 REJECT: 不通过
8     "risk_domains": [{
9         "name": "风险域",
10        "id": "风险域ID",
11        "label_list": [{
12            "label_id": 1,
13            "label_name": "标签名称",
14            "cause_msg": "违规原因"
15        }]
16    }]
17 }
```


action：向mq发信，格式如上

- 根据request里面的getInputTagIds从kuiperFacadeService.getTagMultiInfo获取到标签配置信息
- 并通过内部类，构造发信msg，按照risk\_domains进行分组
  - risk\_domains对应TagMultiInfo的二级风险域：H2Domain
  - label\_list就是TagMultiInfo的列表，使用其id，name，reason来填充
  - taskId为request的contentId，status为request的action
- 最后通过EventsMQProducer producer来向mq发信，通过TaskID路由
- 

Java  复制代码

```
1 topic: note_pre_audit_result
2 tag: NOTE_PRE
3 key: taskId
```

测试：

调用david接口触发场景  David

然后在mq里面查消息

返回资源的接口：QueryNoteResourceService

主要思路是返回coverImageOcrInfo里面的wordPosition的width和height

在coverImageOcrInfo里面新增了两个字段width和height

两个值的获取方式如下

```

Java
复制代码

1  List<ImageSizeInfo> noteImageSizeMap = noteImageOcrService.getNoteImageSizeMap(noteId,
    historyId, url);
2      if (!CollectionUtils.isEmpty(noteImageSizeMap)) {
3          Double height = null;
4          Double width = null;
5          for (ImageSizeInfo info : noteImageSizeMap) {
6              if (StringUtils.equalsIgnoreCase(info.getInfo(), "height")) {
7                  height = info.getScore();
8                  noteImageOcrInfo.setHeight(height);
9              }
10             if (StringUtils.equalsIgnoreCase(info.getInfo(), "width")) {
11                 width = info.getScore();
12                 noteImageOcrInfo.setWidth(width);
13             }
14             if (height != null && width != null) {
15                 break; // 已经都填充完, 提前退出
16             }
17         }
18     }

```

#### 【PRD】识别举报人为空账号的机审因子

因子1:

输入: 举报人用户ID

条件: 查询账号历史各维度发布数量, 维度枚举: 笔记&评论&私信&群消息&弹幕) 是否统计为空

查询时间周期: 近1年/6个月

输出: false/true

数据类型: BOOLEAN

交互: 是/否

调用以下因子:

userNoteCountV2 用户笔记数量

userCommentCountTotalIn90Days 用户90天内评论数量

userRecallPrivateMsgInfoList

userNeverPostGroupMessage 用户没有发送群信息

userRecallLikeBarrageInfoList

```

Java
复制代码

1  if (userNoteCountV2.total == 0 && userCommentCountTotalIn90Days == nil &&
    count(userRecallPrivateMsgInfoList) == 0 && userNeverPostGroupMessage &&
    count(userRecallLikeBarrageInfoList) == 0){
2      return true;
3  }
4  return false;

```

诈骗关联，存储与查询

## 📖 诈骗关联/多图评论标注

### 1. sit新建数据表

SQL

复制代码

```
1 CREATE TABLE `user_affect_dispose_record` (  
2   `id` bigint(20) unsigned NOT NULL AUTO_INCREMENT COMMENT '主键id',  
3   `affected_resource_id` varchar(255) NOT NULL DEFAULT '' COMMENT '关联用户id',  
4   `trigger_resource_id` varchar(255) NOT NULL DEFAULT '' COMMENT '主用户id',  
5   `source_type` varchar(255) NOT NULL DEFAULT '' COMMENT '队列类型',  
6   `create_time` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP COMMENT '创建时间',  
7   `update_time` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP  
   COMMENT '更新时间',  
8   PRIMARY KEY (`id`),  
9   KEY `idx_update_time` (`update_time`),  
10  KEY `idx_affected_source` (`affected_resource_id`, `source_type`),  
11  KEY `idx_trigger_resource_id` (`trigger_resource_id`)  
12 ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COMMENT='关联用户处置记录表'
```

### 2. mars UserVerifyService.processDataFieldContent 增加一段落表的逻辑（参考charon里的erisService用法），使用thrift远程调用rpc接口

Java

复制代码

```
1 public void processDataFieldContent(DisposalTaskRequestBO request, String  
   finalControlSendMsgWhenTag, String idempotentId, String remark, String finalTagSource,  
   List<Long> labelIds) {  
   try {  
     //获取dataFiled里的用户，并对其打标  
     if (CollectionUtils.isEmpty(request.getDisposalDataFieldTaskRequestBos())) {  
       // 构造落表数据（主用户、关联用户）  
       CommonSingleSqlInsertRequest insertRequest = new  
CommonSingleSqlInsertRequest();  
       insertRequest.setRecords(Lists.newArrayList());  
       for (FinishDataFieldTaskRequestBo finishDataFieldTaskRequestBo :  
request.getDisposalDataFieldTaskRequestBos()) {  
  
         //与审核主体用户打的同一个标签  
         CommonProcessRequestV2 dataFiledCommonProcessRequest =  
buildProcessRequest(finishDataFieldTaskRequestBo.getContentId(),  
           request.getProcessorId(), request.getSourceType(),  
finalTagSource,  
           labelIds,  
           finalControlSendMsgWhenTag, request.getRiskId(), idempotentId,  
remark);  
         processCommon(request.categoryType, request.sourceType,  
dataFiledCommonProcessRequest);  
       }  
     }  
   }  
}
```

```

        CommonSingleSqlInsertResult insertResult = new
CommonSingleSqlInsertResult();
        insertRequest.getRecords().add(insertResult);
        insertResult.setTableName(TABLE_USER_AFFECT_DISPOSE_RECORD);
        Map<String, String> fieldMap = Maps.newHashMap();
        fieldMap.put("affected_resource_id",
finishDataFieldTaskRequestBo.getContentId());
        fieldMap.put("trigger_resource_id", request.getResourceId());
        fieldMap.put("source_type", request.getSourceType());
        fieldMap.put("task_id", request.getTaskId());
        insertResult.setFieldValueMap(fieldMap);
    }
    // 落表记录
    saveToManualInfo(insertRequest);
}
} catch (Exception e) {
    log.error("processDataFieldContent error", e);
}
}
2
3 private void saveToManualInfo(CommonSingleSqlInsertRequest insertRequest) {
4     try {
5         List<CommonSingleSqlInsertResult> responseResult =
erisClient.batchInsertCommonSingleRecord(insertRequest);
6         if (org.apache.commons.collections.CollectionUtils.isEmpty(responseResult)
|| Objects.isNull(responseResult.get(0))) {
7             log.error("record insert to manual_info failed, request={}, rsp is
null.", insertRequest);
8             return;
9         }
10        if (!responseResult.get(0).isSuccess()) {
11            log.error("record insert to manual_info failed, request={}, errorMsg=
{}.", insertRequest, responseResult.get(0).getMsg());
12        }
13    } catch (Exception e) {
14        log.error(String.format("transToManualInfo failed, request=%s,
errorMsg=%s.", insertRequest, e.getMessage()), e);
15    }
16 }

```

3. vulcan写一个查询关联用户的接口，输入用户id（申诉的targetid），输出一组用户id（包括主用户和一并被关联处置的用户）。by查表

Java ▾

复制代码

```

1 @Slf4j
@Service
public class QueryUserAffectDisposeService extends
AbstractQuery<QueryUserAffectDisposeRequest, QueryUserAffectDisposeResponse> {

```

```

@Resource
private ErisClient erisClient;

@Override
public QueryUserAffectDisposeResponse doQuery(QueryUserAffectDisposeRequest request)
throws Exception {
    //参数校验
    if (StringUtils.isBlank(request.getTargetId())) {
        return ParamCheckUtils.errorReturn(null, new
QueryUserAffectDisposeResponse());
    }
    return queryUserAffectDisposeService(request);
}

private QueryUserAffectDisposeResponse
queryUserAffectDisposeService(QueryUserAffectDisposeRequest serviceReq) {
    QueryUserAffectDisposeResponse serviceResp = new
QueryUserAffectDisposeResponse();
    // 根据传入的用户id, 查询导致该用户被处置的用户id列表。例如用户a处置时关联处置了用户bcd, 输入用
    户b id, 输出用户a id
    CommonSingleSqlSelectRequest request = new CommonSingleSqlSelectRequest();
    request.setTableFields(Lists.newArrayList("trigger_resource_id"));
    request.setTableName("user_affect_dispose_record");
    // 必传被关联处置的用户id affected_resource_id。如果有source_type, 也一并作为查询条件。无
    法区分发起申诉的处置对应哪个主用户, 查到的结果都返回
    request.setSqlSelect(Lists.newArrayList());
    List<SqlSelectBody> selectBodyList = Lists.newArrayList();
    selectBodyList.add(erisClient.buildSqlSelectBody("affected_resource_id",
ConditionEnum.EQUAL, new ConditionValue().setStrArg(serviceReq.getTargetId())));
    if (StringUtils.isNotBlank(serviceReq.getSourceType())) {
        selectBodyList.add(erisClient.buildSqlSelectBody("source_type",
ConditionEnum.EQUAL, new ConditionValue().setStrArg(serviceReq.getSourceType())));
    }
    request.getSqlSelect().add(selectBodyList);
    request.setLimit(100);
    CommonSingleSqlSelectResult result =
erisClient.getCommonSingleSqlSelectResult(request);
    // 构造返回结果
    List<String> triggerUserIdList = new ArrayList<>();
    Optional.ofNullable(result.getRecords()).orElse(new ArrayList<>())
        .forEach(e -> triggerUserIdList.add(e.get("trigger_resource_id")));
    serviceResp.setTriggerUserIdList(triggerUserIdList);
    return serviceResp;
}

@Override
public String requestKey() {
    return "queryUserAffectDisposeService";
}
}

```



郑德凯-250607

郑德凯-250607

郑德凯-250607

郑德凯-250607

郑德凯-250607

郑德凯-250607

郑德凯-250607

郑德凯-250607

郑德凯-250607

郑德凯-250607