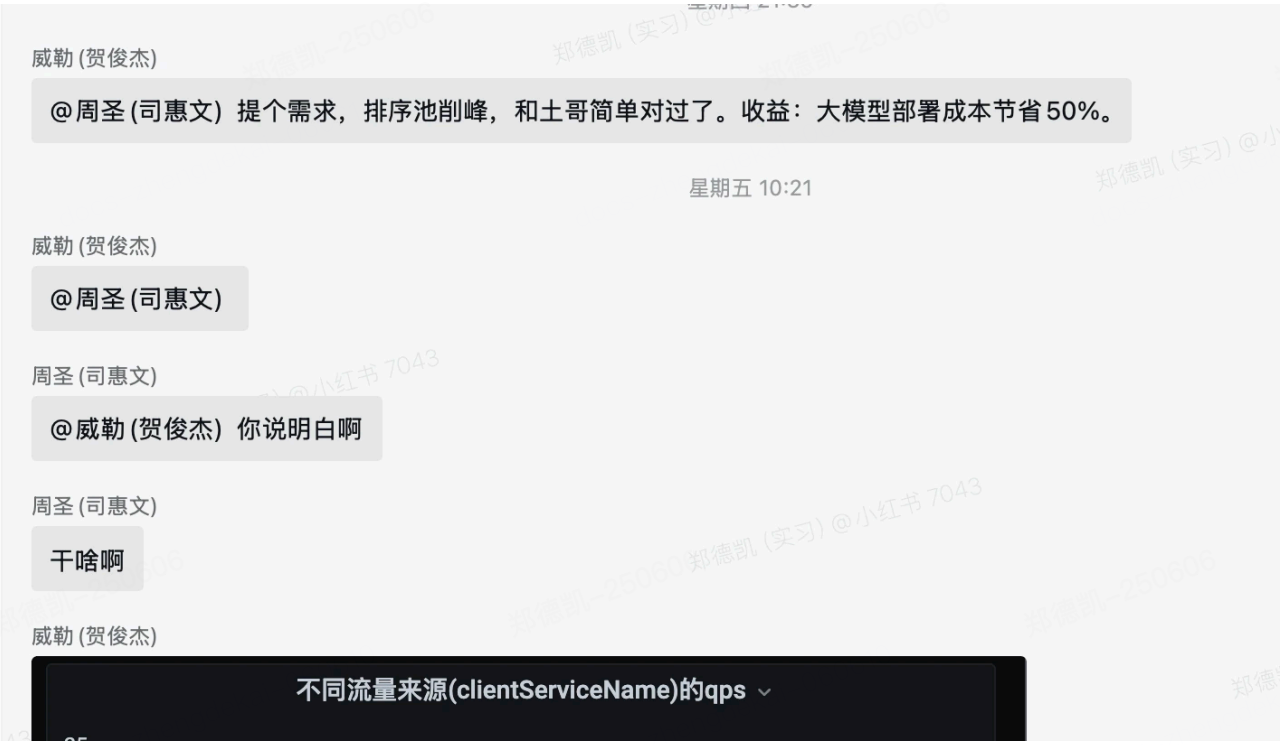
平滑排序池



拉取逻辑：

排序池/机审-》人审数据库

排序池的会自动向mq里发送信息，送入机审进行过滤，之后在送入人审的数据库里面

如果人审数据库不足，就会让排序池拉取数据

具体代码在aphro，逻辑是分批加平滑，前置送审使用了限流器进行限流

Java 复制代码

```java
private boolean sendAuditPreCheck(String queueType, GetPoolDataRequest request) {
    String poolType = this.register();
    if (PriorityPoolType.COMMON_POOL.getPoolType().equals(poolType)) {
        poolType = request.getBizType() + ":" + request.getPoolType();
    }

    String key = poolType + ":" + queueType;
    Integer limitTime = rateLimitMap.getOrDefault(queueType, defaultRateLimit);
    log.info("sendAudit rateLimit queueType {} ,limitTime {}", queueType, limitTime);

    if (RedisUtils.existsLimit(aphroCache, key, PriorityPoolConstant.LOCK)) {
        return false;
    }
    if (RedisUtils.tryLock(aphroCache, key,
            PriorityPoolConstant.LOCK, limitTime)) {
        return true;
```

```
        }
        return false;
    }
```

Java 复制代码

```java
private GetPoolDataResponse doSendAudit(GetPoolDataResponse response, String queueType) {
    if (response != null && response.isSuccess()) {
        List<GetPoolDataResponse.SendToAuditData> data = response.getData();
        List<String> deleteList = new ArrayList<>();

        if (!CollectionUtils.isEmpty(data)) {
            // 分批次处理，每批10个
            int batchSize = sendAuditRateLimitMap.getOrDefault("batchSize",10);
            int delayMillis = sendAuditRateLimitMap.getOrDefault("delayMillis",200);

            List<GetPoolDataResponse.SendToAuditData> sendData = new ArrayList<>();
            for (int i = 0; i < data.size(); i += batchSize) {
                int end = Math.min(i + batchSize, data.size());
                List<GetPoolDataResponse.SendToAuditData> batch = data.subList(i, end);

                for (GetPoolDataResponse.SendToAuditData sendToAuditData : batch) {
                    String noteId = sendToAuditData.getNoteId();
                    Map<String, String> extraInfo =
scenarioInfoMapPrepare(sendToAuditData);

                    String poolType = this.register();
                    if (PriorityPoolType.COMMON_POOL.getPoolType().equals(poolType)) {
                        poolType = response.getPoolType();
                    }
                    if (priorityPoolProducer.triggerScenario(noteId, poolType,
queueType, this.register(), extraInfo)) {
                        deleteList.add(noteId);
                        sendData.add(sendToAuditData);
                    }
                }
                // 延迟：减缓下游压力
                try {
                    Thread.sleep(delayMillis);
                } catch (InterruptedException e) {
                    Thread.currentThread().interrupt();
                    break;
                }
            }
            response.setData(sendData);
        }
        changeDataStatus(deleteList, PriorityPoolConstant.WILL_DELETE, response);
    }
    return response;
}
```

云效平台

Java 复制代码

```
@Component
@Slf4j
public class RedisRateLimiterService {

    @Resource
    private Jedis aphroCache;

    private RateLimiterDelegate componentCommonRateLimiter;

    @Resource
    private AphroMetricsService aphroMetricsService;

    public boolean acquirePerSecondForCommPool(int incNum, String poolType, Integer threshold) {
        String sortKey = String.format(RedisKeys.COMMON_POOL_SAVE, poolType);

        long seconds = System.currentTimeMillis() / 1000;

        return acquire(sortKey, seconds, threshold, incNum);
    }

    /**
     *
     * @param incNum 本次增加计数;
     * @param tagId 本次组件处置标签
     * @param reason  本次组件处置reason
     * @param threshold 限流阈值;
     * @return
     */
    public boolean acquirePerSecondForComponentInit(int incNum, String tagId, String reason, Integer threshold) {
        String sortKey = String.format(RedisKeys.COMMON_COMPONENT_OP, tagId, reason);

        long seconds = System.currentTimeMillis() / 1000;

        boolean limit = acquire(sortKey, seconds, threshold, incNum);

        //1.被redis限流后，降级使用本地rateLimiter阻塞线程；平缓流量;
        if (limit) {
            //1.记录被限流的业务，辅助判断是否调整限流值;
            aphroMetricsService.recordComponentOpRateLimitCount(sortKey);
            //1.尝试阻塞1s;
            componentCommonRateLimiter.tryAcquire(1, 1000, TimeUnit.MILLISECONDS);
        }
        return true;
    }
```

```
    /**
     * 以秒或分为单位，滑动窗口（存储600 *（秒或分），超过删除）
     *
     * @param sortSetKey 滑动窗口数据存储set
     * @param seconds    set的member 当前的秒数
     * @param threshold  限制阈值
     * @param incNum     当次增量
     * @return boolean
     */
    public boolean acquire(String sortSetKey, long seconds, double threshold, double
incNum) {

        String member = String.valueOf(seconds);
        Double zscore = aphroCache.zscore(sortSetKey, member);

        if (zscore != null && Double.compare(zscore, threshold) >= 0) {
            return true;
        }
        aphroCache.zincrby(sortSetKey, incNum, member);
        //删除多余数据
        Set<String> zrange = aphroCache.zrange(sortSetKey, 0, -1);
        if (zrange != null && zrange.size() > 100) {
            String min = Collections.min(zrange);
            aphroCache.zrem(sortSetKey, min);
        }
        return false;
    }

    @PostConstruct
    public void init() {
        this.componentCommonRateLimiter = new RateLimiterDelegate(() ->
ConfigService.getAppConfig().getIntProperty(BusinessAdConstant.COMPONENT_RATE_KEY, 5));
    }
}
```