2024 Q2 OKR 完结 RocketMQ优化Webhook处理等业务场景 (已上线)

1. 背景/历程/收获/展望

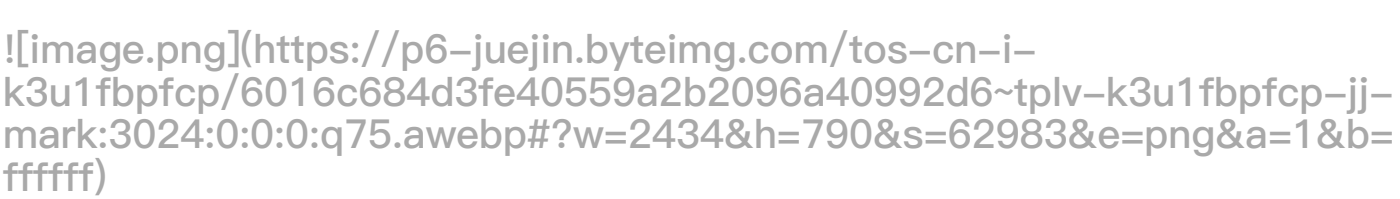
> 今天2024/05/23
>
>
> 背景: 最近组内okr 需要使用MQ优化多数据中心用户数据同步 / webhook处理等业务场景 / App推送提醒，我这边接手了这个优化任务，去年已经优化了一个场景 第三方adjust调用使用异步线程池接收回调，导致系统某个时刻大量的线程thread堆积，出现告警，影响服务的可用性,后面就使用MQ 异步发送 慢慢消费处理 达到削峰填谷的效果，后续问题就没有出现过了，但是监控方面没有做。
>
>
> Q2季度主要对核心业务进行优化，同步多数据中心现在是使用python脚本(千万级别表 慢SQL)，Webhook/推送/站内信 都是使用异步线程池 (危险就是 消息数据不是持久化，服务重启就没/线程池任务堆积 拒绝策略主线程执行 主线程出现阻塞现象/没有调用失败重试机制/没有消息数量,执行状态的监控.....)

* **时间线:** 2024/04-01- 2024/05/23.
* **历程:**

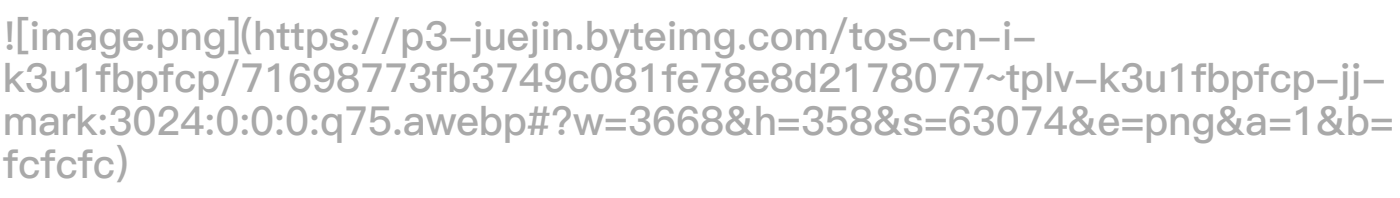
1. 最近主要觉得写业务成长已经到瓶颈了，一个db+redis就够完成版本迭代的业务了，这段时间也想了很多，怎么提升自己的能力,并且能够作为okr去推动，自己就去当前业务里面发掘可以进行优化的点，找了一点并发量大一点的业务进行优化，当然我们还有短信业务(用户使用量较小 优化带来的收益不大)，我们这边商城订单/购物车/发货都是db梭哈的，优化的空间非常大，主要业务不是我写的，现在能跑就行 如果你强行去优化，可能对自我的提升非常大，如果过程中出现bug,带来的损失可不是我能够承担的，风险太大不考虑商城这块了，这个季度主要对三个点Webhook处理等业务场景 / App推送提醒/多数据中心用户数据同步 进行代码重构优化。
2. 工作两年来，虽然头脑中存储了很多MQ相关的知识点，却没有机会落地，这不机会来了 自己创造。整个的过程中，都是自测，预发自测/修复，再自测上线 观察线上功能是否正常，MQ数据的状态，各项指标是否正常。还有一个点我们这边RocketMQ搭建的单机版本的，可用性得不到保证，并且机器的配置很低，现在暂时能扛住当前用户流量并发操作。
3. 其实我们有接入Amazon的SQS进行转发消息到多数据中国中心，leader那边想让我使用这个对部分业务进行改造，部分业务使用RocketMQ, SQS看了一

眼设计和代码，就是一坨屎，那个sqs 感觉没啥用 首先restApi请求 又转发到其他数据中心的restApi请求，并且学习成本太大了,没必要用这个，两层api接口请求 完全保证不了消息发送成功/失败，leader说SQS有死信队列，在面板上可以直接一键重新消费，当时脑子比较懵的，当时对RocketMQ的死信队列没有什么概念，后面我去看了一眼RocketMQ的死信队列，没毛病啊，都能实现，刚才那个点不是能说服我的理由，后面说服leader，全部使用RocketMQ进行优化改造。
* **收获:** 做完了之后 感觉没什么挑战性，主要个人技能提升，深入理解MQ及源码, 各种使用场景。
* **未来展望:** RocketMQ 每个服务需要使用都要集成相关配置/日志监控困难等等，后续进行封装rocketmq-starter,简化代码/流程/简单使用，参与RocketMQ开源社区项目建设。

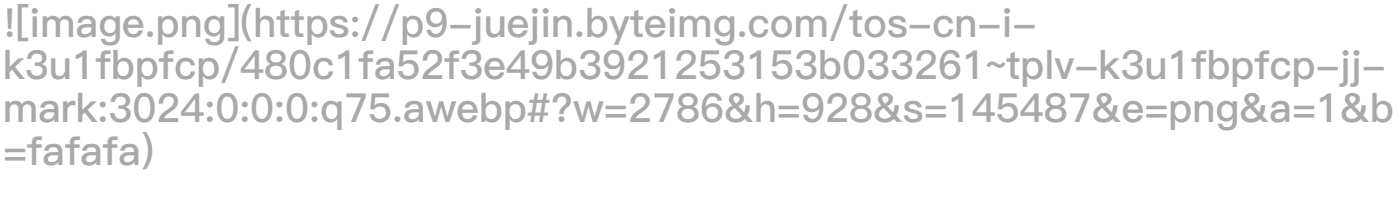
> 从这个季度上线到今天05/23 已经接收了160ws数据了



> 观察一下 每天MQ消息的数量



> Gitlab 分支记录



2. MQ业务场景优化

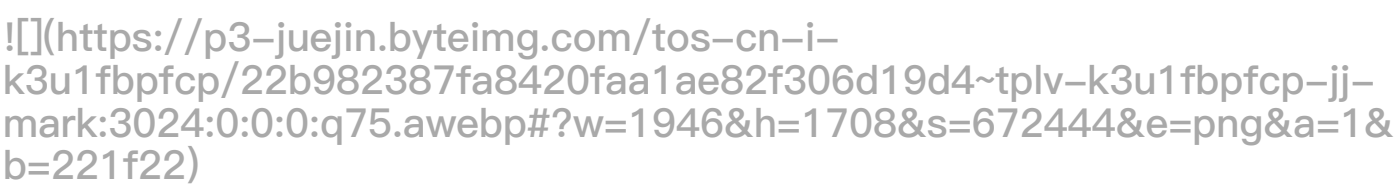
2.1 MQ版本

* [rocketmq.apache.org/zh/docs/4.x…](http://cxyroad.com/”https://rocketmq.apache.org/zh/docs/4.x/”) 4.x 官方相关文档

> Springboot-rocketmq 2.2.2 版本 内置rocketmq 版本是4.9.3

* 看rocketmq 5.0.0+ 版本有很多新特性 比如自定义消息延迟时间(原来最大是2个小时)

```
<!-- rocketmq -->
<dependency>
    <groupId>org.apache.rocketmq</groupId>
    <artifactId>rocketmq-spring-boot-starter</artifactId>
    <version>2.2.2</version>
</dependency>
```



2.2 生产者

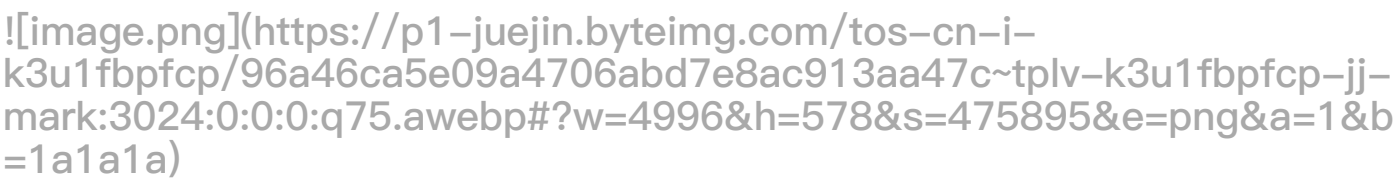
* 生产者这边采用 asyncSend 异步发送，接收回调消息是否成功，依赖参数配置 retry-times-when-send-async-failed: 3# 异步消息发送失败重试次数（其实这个步骤 发送失败重试的流程我不确定他是否会生效，消息发送到broker，我没办法模拟这个环境进行重试，可能这个参数没有起作用，后续再去看看网上博客/源码研究一下）

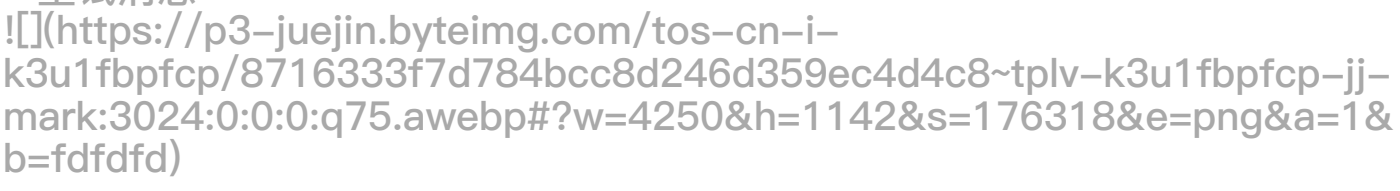
2.3 消费者

* 消费者这边采用集群模式 messageModel = MessageModel.CLUSTERING，设置消费者重试5次 进入死信队列 consumer.setMaxReconsumeTimes(5); 每次提取的最大消息数

consumer.setPullBatchSize(32); 使用并发模式pull消息 consumeMode = ConsumeMode.CONCURRENTLY, 这个重试消费失败5次进入死信队列我测试过了 没问题，我这消费者业务逻辑都是try catch 然后打印错误，抛出throw new RuntimeException(e); 进行重试，但是这个流程其实有很大问题，你消费的业务需要保证事物 要么全部成功/要么全部失败，假如你的消费者有涉及到分布式接口调用场景，可能前面的请求都成功了，到最后一个请求失败了，那算消费失败需要重试呢 还是消费成功，是不是需要上本地消息表或者别的分布式事务来解决相关问题，问题的复杂性又提升了一个档次，我们这边的业务并没有涉及关心事物操作，失败就失败吧，重试就完事了，保证消费者幂等性操作就行。

> 消费失败加上重试操作 测试



* 重试消息


2.4. App推送提醒

* app端顶端消息提醒大家应该都见过吧，我们推送接入的是第三方极光[www.jiguang.cn/accounts/lo...](http://cxyroad.com/”https://www.jiguang.cn/accounts/login/form”) 的推送，并不是自研的推送，成本太大/维护成本太高，直接买就完事了，当大量推送消息到达时，线程池也有顶不住的时候，比如后台对50w用户进行推送活动相关的消息，线程池肯定扛不住啊，到时候不仅推送业务顶不住，还拖垮了这个服务中其他的业务，业务出现瓶颈，优化机会来了，下面都是改造过程中的一些配置(已脱敏)。

> RocketMQConfig 配置

```
@Configuration
```

```
@Import({RocketMQAutoConfiguration.class})
public class RocketMQConfig {
}
```

```
# rocketmq配置
rocketmq:
  name-server:
${rocketmq.server.host:127.0.0.1}:${rocketmq.server.port:9876}
  # 生产者
  producer:
    group: producer_xxxxx
    # 消息发送失败重试次数
    retry-times-when-send-failed: 3
    # 异步消息发送失败重试次数
    retry-times-when-send-async-failed: 3
```

> 生产者

```
@Slf4j
@Component
public class PushMessageProducerService {

  private static final Gson GSON = new Gson();

  @Autowired
  private RocketMQTemplate rocketMQTemplate;

  public void asyncPushToUser(PushContents pushContents, int userId,
      int appFlag, int activityId, boolean passThough, int pushType) {
    PushToUserDto pushMessage = new PushToUserDto();
    pushMessage.setPushContents(pushContents);
    pushMessage.setUserId(userId);
    pushMessage.setAppFlag(appFlag);
    pushMessage.setActivityId(activityId);
    pushMessage.setPassThough(passThough);
    pushMessage.setPushType(pushType);

    String requestId = ThreadContext.get("requestId");
```

```
    pushMessage.setRequestId(requestId);
    // 异步发送消息
    log.info("push message producer send message {} {}", userId,
        GSON.toJson(pushMessage));
rocketMQTemplate.asyncSend(BusinessConstants.TOPIC_PUSH_MESSA
GE,
        pushMessage, new SendCallback() {
          @Override
          public void onSuccess(SendResult sendResult) {
            log.info("push message async message succeeded
sendResult:{}",
                sendResult);
          }

          @Override
          public void onException(Throwable throwable) {
            log.error("push message async message failed: {}",
                throwable.getMessage());
          }
        });
  }
}

```

> 消费者

```
@Slf4j
@Component
@RocketMQMessageListener(topic =
BusinessConstants.TOPIC_PUSH_MESSAGE,
    consumerGroup = BusinessConstants.GROUP_CONSUMER_TOOLS)
public class PushToUserConsumerService implements
    RocketMQListener<PushToUserDto>,
RocketMQPushConsumerLifecycleListener {

  @Override
  public void onMessage(PushToUserDto pushToUserDto) {
    String requestId = pushToUserDto.getRequestId();
    if (!Strings.isNullOrEmpty(requestId)) {
      ThreadContext.put("requestId", requestId);
    }
    log.info("push message consumer receipt message {} {}",
        pushToUserDto.getUserId(),
```

```
        new Gson().toJson(pushToUserDto));
    try {
      //xxx 业务处理
    } catch (Exception e) {
      log.error("push message consumer receipt message fail {} {}",
          pushToUserDto.getUserId(), e.getMessage(), e);
      throw new RuntimeException(e);
    }
  }

  @Override
  public void prepareStart(DefaultMQPushConsumer consumer) {
    // 设置消费者重试次数
    consumer.setMaxReconsumeTimes(3);
    // 每次提取的最大消息数
    consumer.setPullBatchSize(32);
  }

```

2.5 Webhook处理等业务场景

* webhook 回调的场景我们这边特别多，比如appstore购买订阅回调，google/paypal/stripe 购买订阅的回调等等，由于我们这边有多个数据中心，但是上游平台只能配置一个地址，我们也不知道这次的回调数据要回调到哪个数据中心，只能回调到其中一个，再进行转发到其他数据中心，现在转发的策略是通过new Thread() 进行异步转发，不考虑消息是否成功/失败，监听不到消息是否请求，对我们上层来说无感/危险性极大，这不改造机会来了，下面都是改造过程中的一些代码(已脱敏)。

> 生产者

```
@Slf4j
@Component
public class CallBackProducerService {

  private static final Gson GSON = new Gson();

  @Autowired
  private RocketMQTemplate rocketMQTemplate;
```

```
  /**
   * 异步发送 callback消息.
   *
   * @param appStoreCallBackDto
   */
  public void asyncSendAppStoreCallBackMessage(
      AppStoreCallBackDto appStoreCallBackDto) {
    log.info("appstore callback producer send message {}",
        GSON.toJson(appStoreCallBackDto));
rocketMQTemplate.asyncSend(BusinessConstants.TOPIC_APPSTORE_C
ALLBACK,
        appStoreCallBackDto, new SendCallback() {
          @Override
          public void onSuccess(SendResult sendResult) {
            log.info("callback async message succeeded sendResult:{}",
                sendResult);
          }

          @Override
          public void onException(Throwable throwable) {
            log.error("callback async message failed: {}",
                throwable.getMessage());
          }
        });
  }
}

```


> 消费者


```
@Slf4j
@Component
@RocketMQMessageListener(
    topic = BusinessConstants.TOPIC_APPSTORE_CALLBACK,
    consumerGroup =
BusinessConstants.GROUP_APPSTORE_CALLBACK,
    consumeMode = ConsumeMode.CONCURRENTLY,
    messageModel = MessageModel.CLUSTERING)
public class AppStoreCallBackConsumerService implements
    RocketMQListener<AppStoreCallBackDto>,
    RocketMQPushConsumerLifecycleListener {

  @Autowired
```

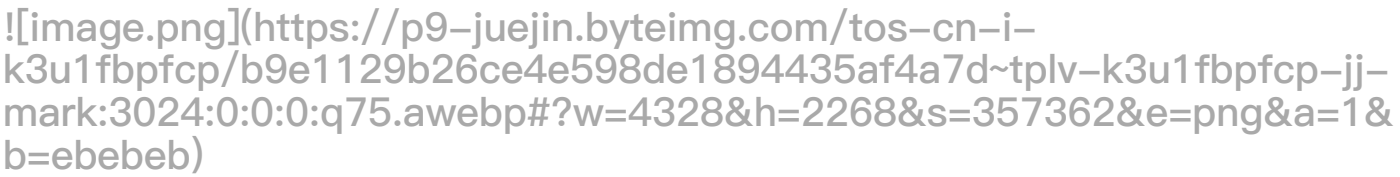
```
  WebClient webClient;

  @Override
  public void onMessage(AppStoreCallBackDto appStoreCallBackDto) {
    String requestId = appStoreCallBackDto.getRequestId();
    if (!Strings.isNullOrEmpty(requestId)) {
      ThreadContext.put("requestId", requestId);
    }

    try {
        // 业务处理xxxxxxxxxxxxx
      } catch (Exception e) {
        log.error("appstore callback consumer message fail {}",
            e.getMessage(), e);
        // 500 抛出异常 进行重试消息
        throw new RuntimeException(e);
      }
    }
  }


  @Override
  public void prepareStart(DefaultMQPushConsumer consumer) {
    // 设置消费者重试5次 进入死信队列
    consumer.setMaxReconsumeTimes(5);
    // 每次提取的最大消息数
    consumer.setPullBatchSize(32);
  }

```

2.6 多数据中心用户数据同步

* 同步多数据中心现在是使用python脚本查从库MySQL将符合条件的用户查询出来，再直接连接其他数据中心的Redis 将数据写入进去，现在出现了慢SQL查询的瓶颈，同步的用户数据出现延迟，数据只会越来越多，带来的危害越来越大，所以才用业务优化的方案，放弃python脚本的同步，在用户登陆注册的时候/用户修改了xxxx将数据同步到其他数据中心，使用MQ异步写入，消费者慢慢消费 转发到其他数据中心，将数据同步到其他数据中心的Redis中。也是成功解决的现在存在的大表问题，主要这个同步方案是2021年设计的,当时的用户量并没有那么多 加上查询的是从库 不影响主库，所有这个设计方案也是成功运行了3年多了，现在也暴露出该有的问题了。下面都是改造过程中的一些代码(已脱敏)。

> 读写队列的大小都是默认值4/4 记住消费者的数量并不是越多越好 而是看队

列的大小来设置的。



> 生产者

```
@Slf4j
@Component
public class AccountUpdateProducerService {

  private static final Gson GSON = new Gson();

  @Autowired
  private RocketMQTemplate rocketMQTemplate;

  /**
   * 异步发送同步到其他数据中心消息.
   *
   * @param accountUpdateDto
   * @param source
   */
  public void asyncSendAccountUpdateMessage(AccountUpdateDto
accountUpdateDto,
      int source) {
    log.info("account update producer send message {} {}", source,
        GSON.toJson(accountUpdateDto));
    if (accountUpdateDto != null && accountUpdateDto.getUserId() == 0) {
      return;
    }
rocketMQTemplate.asyncSend(BusinessConstants.TOPIC_ACCOUNT_UP
DATE,
        accountUpdateDto, new SendCallback() {
          @Override
          public void onSuccess(SendResult sendResult) {
            log.info("account update producer async message succeeded "
+
                "sendResult:{}", sendResult);
          }
```

```
            @Override
            public void onException(Throwable throwable) {
              log.error("account update producer async message failed: {}",
                  throwable.getMessage());
            }
          });
    }
}

```

```
@Slf4j
@Component
@RocketMQMessageListener(
    topic = BusinessConstants.TOPIC_ACCOUNT_UPDATE,
    consumerGroup = BusinessConstants.GROUP_ACCOUNT_UPDATE,
    consumeMode = ConsumeMode.CONCURRENTLY,
    messageModel = MessageModel.CLUSTERING)
public class AccountUpdateConsumerService implements
    RocketMQListener<AccountUpdateDto>,
    RocketMQPushConsumerLifecycleListener {

  @Autowired
  WebClient webClient;

  @Override
  public void onMessage(AccountUpdateDto accountUpdateDto) {
    String requestId = accountUpdateDto.getRequestId();
    if (!Strings.isNullOrEmpty(requestId)) {
      ThreadContext.put("requestId", requestId);
    }
    log.info("account update consumer message {}",
        accountUpdateDto);
    try {
        // 业务处理xxxxxxxxxxx
      } catch (Exception e) {
        log.error("account update consumer message fail {}",
            e.getMessage(), e);
        // 500 抛出异常 进行重试消息
        throw new RuntimeException(e);
      }
    }
  }

  @Override
  public void prepareStart(DefaultMQPushConsumer consumer) {
```

```
    // 设置消费者重试5次 进入死信队列
    consumer.setMaxReconsumeTimes(5);
    // 每次提取的最大消息数
    consumer.setPullBatchSize(32);
  }
}

```

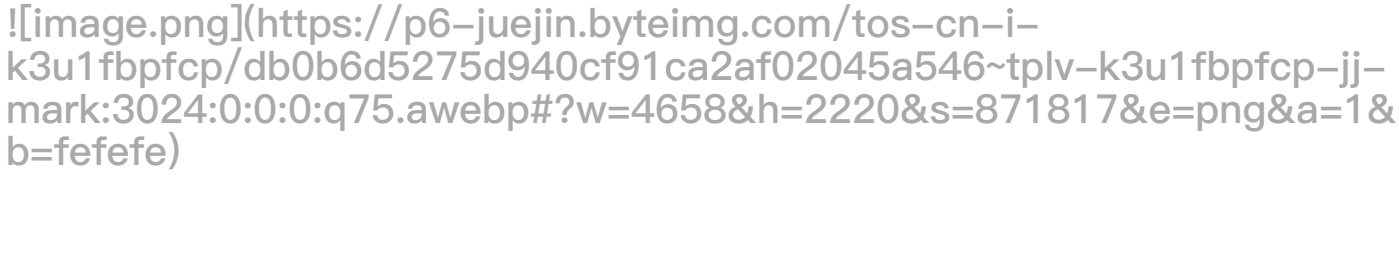
3.告警监控 RocketMQ Exporter+HertzBeat
---

* 当我们加了一层MQ的话 架构的复杂性就增大了,我们需要保证中间件的高可用，以及宕机的时候能够直接告警，自动重启修复，至人工处理，所以我们需要监控的数据越来越多了,我这边主要是监控的数据有以下:

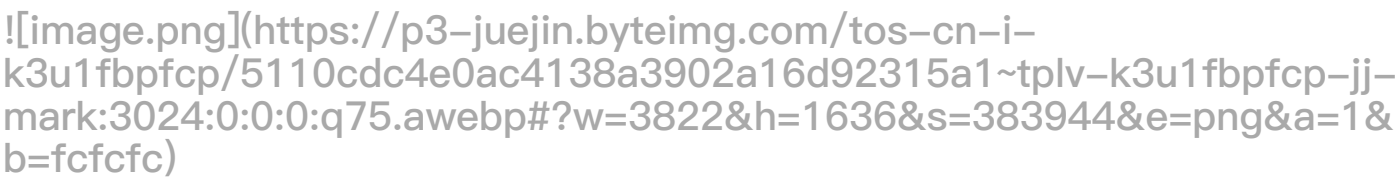
+ commitlog 磁盘使用大小
rocketmq_brokeruntime_commitlog_disk_ratio
rocketmq_brokeruntime_commitlogdir_capacity_free,
+ 消息延迟堆积大小 **rocketmq_group_diff**,
+ 重试消息堆积大小 **rocketmq_group_retrydiff**,
+ 死信队列堆积大小 **rocketmq_group_dlqdiff**
+ .....等等。
* RocketMQ Exporter 安装 使用docker 参考这个作者的安装 juejin.cn/post/726253...
* [rocketmq.apache.org/zh/docs/4.x...](http://cxyroad.com/”https://rocketmq.apache.org/zh/docs/4.x/deployment/04Exporter/”)
监控的指标选择

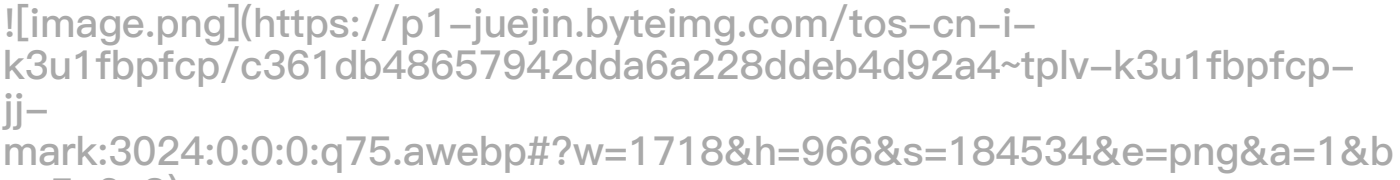
* 启动完 访问 [http://127.0.0.1:9121/metrics](http://cxyroad.com/”http://127.0.0.1:9121/metrics”) 查看相关数据

> hertzBeat MQ性能指标面板观看 绑定暴露出来的 /metrics接口



> 监控阈值设置 告警/恢复发送到飞书



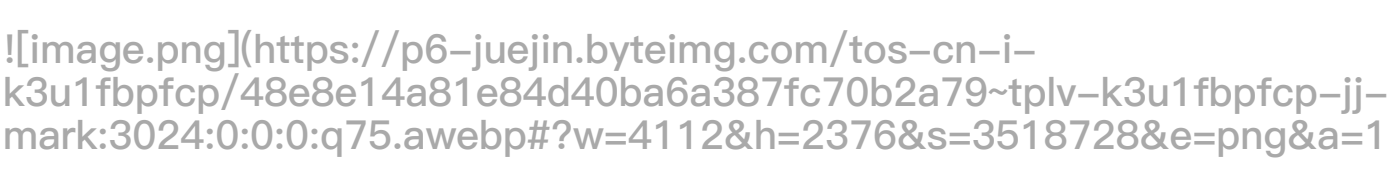


4. 思考
---

* （设计） 现在场景是回调都是打到mall服务上面 在mall服务继承rocketmq依赖 添加配置文件，但是推送/站内信/邮件是 打到tools服务上面，tools服务也继承rokcetmq依赖 添加配置文件，后续如果想在别的服务进行mq操作 又需要走这一套流程，好的方法就是mq抽成一个项目，别的服务器调用mq项目工具类进行生产消息，消费者消费信息的话也是在mq项目里面, 这样的话全都要走rpc调用了(有些直接方法调用)，违背了服务拆分按照业务来。
* （个人）MQ现在只是掌握一点毛皮，需要深入学习相关原理和配置合理的参数，考虑各种异常场景解决的方案，对于出现的异常能够独立有效解决, 继续深入学习吧，熟悉运用到各个场景里面。
* 单机RocketMQ 有点危险，现在的业务量 还能够保证运行，以后就不确定了，业务并发量太小了，时常和朋友交流，他们那么的业务消息堆积起来都是百万级别的，在下游处理的消费者你需要TPS千级别+，听起来就刺激这种业务，可惜没机会遇到 进行发挥一下。

> 贴一张雀魂麻将 2024/05/22 悸动之夏-八木唯（动态）皮肤返场

* [www.bilibili.com/video/BV1b1...](http://cxyroad.com/”https://www.bilibili.com/video/BV1b1421i7ub/?spm_id_from=333.788”)个人记录 嘿嘿 打麻将去了



&b=20181f)

5. 开源项目阅读
---

* 两个rocketmq开源项目 各种文档介绍学习

+ [github.com/apache/rock...](http://cxyroad.com/”https://github.com/apache/rocketmq-spring”)
+ [github.com/apache/rock...](http://cxyroad.com/”https://github.com/apache/rocketmq”)
* 今天看rocketmq-spring消费者消息监听消费者组那边的源码 看到一个文档链接 点进去发现404 又捡个了PR
[github.com/apache/rock...](http://cxyroad.com/”https://github.com/apache/rocketmq-spring/pull/651/files”)

参考
--

* [RocketMQ死信队列_rocketmq 死信队列-CSDN博客](http://cxyroad.com/”https://blog.csdn.net/CSDN877425287/article/details/112786826”)
* [【RocketMQ 二十六】RocketMQ应用之死信队列_rocketmq死信队列使用-CSDN博客](http://cxyroad.com/”https://blog.csdn.net/qq_33333654/article/details/126538081”)
* [RocketMQ(四)：重复消费、消息重试、死信消息的解决方案_springboot中的rocketmq的死信队列如何重新消费?-CSDN博客](http://cxyroad.com/”https://blog.csdn.net/qq_35512802/article/details/134408826”)
* [rocketmq.apache.org/zh/docs/dep...](http://cxyroad.com/”https://rocketmq.apache.org/zh/docs/deploymentOperations/05Exporter”)
* [github.com/apache/rock...](http://cxyroad.com/”https://github.com/apache/rocketmq-exporter”)
* [以rocketmq监控为例，在mac上安装prometheus及grafana_rocketmq接入 grafana-CSDN博客](http://cxyroad.com/”https://blog.csdn.net/x763795151/article/details/122828134”)
* [基于 RocketMQ Prometheus Exporter 打造定制化 DevOps 平台-阿里云开发者社区](http://cxyroad.com/”https://developer.aliyun.com/article/783490”)
* [hub.docker.com/r/apache/ro...](http://cxyroad.com/”https://hub.docker.com/r/apache/rocketmq-exporter”)
* juejin.cn/post/726253...

* [rocketmq客户端发送消息报错和超时问题-阿里云开发者社区](http://cxyroad.com/ "https://developer.aliyun.com/article/1161535")
* [RocketMQ消息消费过后会被清理吗？_rocketmq消息消费后还在吗-CSDN博客](http://cxyroad.com/ "https://blog.csdn.net/qq_15687611/article/details/129029599")
* [RocketMQ——消息文件过期原理 - 薛定谔的风口猪](http://cxyroad.com/ "https://jaskey.github.io/blog/2017/02/16/rocketmq-clean-commitlog/")
* juejin.cn/post/722979...
* [微服务架构中是否应该把诸如redis、mq之类的中间件也封装成一个服务? - 知乎](http://cxyroad.com/ "https://www.zhihu.com/question/306592456/answer/734320458")[SpringBoot整合RocketMQ，老鸟们都是这么玩的！ - JAVA日知录 - 博客园](http://cxyroad.com/ "https://www.cnblogs.com/jianzh5/p/17301690.html")
* RocketMQ与SpringBoot整合进行生产级二次封装 - 掘金
* [github.com/maihaoche/r...](http://cxyroad.com/ "https://github.com/maihaoche/rocketmq-spring-boot-starter")
* [RocketMq生产者组和消费者组_rocketmq的生产者组和消费者组-CSDN博客](http://cxyroad.com/ "https://blog.csdn.net/qq_36559868/article/details/106305164")
* [www.cnblogs.com/CF1314/p/17...](http://cxyroad.com/ "https://www.cnblogs.com/CF1314/p/17681814.html")
* [springboot整合rocketmq，支持多连接生产者和消费者配置。不同topic适配不同业务处理类_rocketmq 多连接-CSDN博客](http://cxyroad.com/ "https://blog.csdn.net/ke7025/article/details/119982155")
* [RocketMQ同一Topic、消费组创建多个消费者失败问题_rocketmq一个topic多个group-CSDN博客](http://cxyroad.com/ "https://blog.csdn.net/qq_43128724/article/details/117324847")
* [RocketMQ 消费者_rocketmq消费者代码-CSDN博客](http://cxyroad.com/ "https://blog.csdn.net/JemeryShen/article/details/126855342")
* [rocketmq-spring的consumer设置消费失败最大重试次数_rocketmq springboot 设置消费失败默认重试次数-CSDN博客](http://cxyroad.com/ "https://blog.csdn.net/x763795151/article/details/118023942")
* [在Apache RocketMQ中这种情况队列数量应该设置多少比较合理呢？_问答-阿里云开发者社区](http://cxyroad.com/ "https://developer.aliyun.com/ask/575677")
* [RocketMQ入坑指南（五）：SpringBoot集成RocketMQ和具体使用方式_rocket mq springboot实际应用-CSDN博客](http://cxyroad.com/ "https://blog.csdn.net/u014374743/article/details/136255200")
* [RocketMQ 添加监控和系统告警通知-腾讯云开发者社区-腾讯云](http://cxyroad.com/ "https://cloud.tencent.com/developer/article/1432219")
* [使用Prometheus监控RocketMQ-CSDN博客](http://cxyroad.com/ "https://blog.csdn.net/sawyerlan/article/details/118152802")
* [RocketMQ的监控和管理工具有哪些_rocketmq管理工具-CSDN博客](http://cxyroad.com/

”https://blog.csdn.net/weixin_41860630/article/details/134984499”)[rocketMQ中，消费者、消费者组、Topic、队列的关系_rocketmq topic和queue关系-CSDN博客](http://cxyroad.com/”https://blog.csdn.net/yuanchangliang/article/details/119190281”)
* [面试官：RocketMQ同一个消费组内的消费者订阅了不同tag，会有问题吗？_rocketmq 订阅多个tag-CSDN博客](http://cxyroad.com/”https://blog.csdn.net/zjj2006/article/details/122014993”)
* [RocketMQ订阅关系一致_rocketmq 消费组订阅同一topic-CSDN博客](http://cxyroad.com/”https://blog.csdn.net/meser88/article/details/122344534?utm_medium=distribute.pc_relevant.none-task-blog-2~default~baidujs_baidulandingword~default-0-122344534-blog-122014993.235%5Ev43%5Epc_blog_bottom_relevance_base6&spm=1001.2101.3001.4242.1&utm_relevant_index=3”)
* [订阅关系一致 | RocketMQ](http://cxyroad.com/”https://rocketmq.apache.org/zh/docs/4.x/bestPractice/07subscribe/”)
* [RocketMQ Promethus Exporter | RocketMQ](http://cxyroad.com/”https://rocketmq.apache.org/zh/docs/4.x/deployment/04Exporter”)
* juejin.cn/post/684490…
* juejin.cn/post/684490…
* [cluster 集群参数](http://cxyroad.com/”https://learn.lianglianglee.com/%E4%B8%93%E6%A0%8F/RocketMQ%20%E5%AE%9E%E6%88%98%E4%B8%8E%E8%BF%9B%E9%98%B6%EF%BC%88%E5%AE%8C%EF%BC%89/15%20RocketMQ%20%E5%B8%B8%E7%94%A8%E5%91%BD%E4%BB%A4%E5%AE%9E%E6%88%98.md”)
* [Rocketmq消息批量发送&消息批量消费-CSDN博客](http://cxyroad.com/”https://blog.csdn.net/l123lgx/article/details/130841226”)
* [Consumer消息拉取和消费流程分析_consumeconcurrentlymaxspan-CSDN博客](http://cxyroad.com/”https://blog.csdn.net/qq_32099833/article/details/120229506”)
原文链接: https://juejin.cn/post/7372020457125052450