cmos-msg接入手册-v2.0.0

1.前言

cmos-msg-2.0.0-SNAPSHOT对原来0.2.3、0.2.4、0.2.5、0.2.6版本进行全面重构，从框架设计、API封装、对象返回、接入方式等进行全面的调整，对原来低效的、不稳定的、有bug的功能进行了修复及优化，同时也新增了一些新的高效API接口，用于提高消息发送、消费的吞吐量。删除了对日志中心、配置中心的强依赖捆绑，避免被迫升级或版本冲突，也删除了老版本繁琐的XML配置，使其接入更简洁、灵活。

新版本使用的RocketMQ核心版本为Apache RocketMQ release v4.1.0-incubating ，该版本是RocketMQ进入Apache基金会孵化的第2个稳定发布版，相对于之前Alibaba RocketMQ的3.5.8版本，在稳定性、性能、功能完善方面都有很大提升。官方要求需64bit JDK 1.8+，但1.7+也可以正常编译及运行，建议使用JDK1.8+。

业务系统集成请参加第3、4章节说明，消息发送推荐使用4.1.1 普通消息异步发送，消息消费推荐使用4.2.1 普通消息集群消费，若这两种方式无法满足，需使用其他消息模式，请联系在线技术组待评估确认后再使用。

技术支撑

姓名	电话	邮箱	职责
许晓东	187 3713 8433	xuxd@asiainfo.com	技术支撑第1接口人
朱桢宏	136 1384 6850	zhuzh@asiainfo.com	技术支撑第2接口人

2.版本说明

2.1 新特性

1. 单条消息异步发送
2. 批量消息异步发送
3. 消息发送后异步回调,onSuccess、onException
4. 事务消息发送
5. 顺序消息发送

6. 消息并行集群消费
7. 定时批量拉取消费
8. 单条、批量消息异步业务处理接口
9. 顺序消息消费
10. 返回RocketMQ的SendResult对象
11. 透传原生RocketMQ的Message、MessageExt对象
12. 提供完善的API使用demo
13. 删除老版本的配置中心、日志中心、Spring强依赖绑定

2.2 提升点

重构第1个版本，后续版本升级时补充

2.3 Bug修复

重构第1个版本，后续版本升级时补充

3.快速集成

3.1 引入依赖包

在业务系统的pom.xml文件中添加jar包依赖，即可完成集成。

```
<dependency>
  <groupId>com.cmos</groupId>
  <artifactId>cmos-msg</artifactId>
  <version>2.0.0-SNAPSHOT</version>
</dependency>
```

3.2 配置集成

cmos-msg只需要配置一个参数:rocketMQ的name server地址,cmos-msg-2.0.0-SNAPSHOT兼容两种配置方式:

1. 配置中心托管,如果项目已接入配置中心,优先使用此方式,请参见3.2.1章节配置中心托管方式
2. 本地配置文件,如果项目没有集成配置中心,请参见3.2.2章节

3.2.1 配置中心托管

配置中心托管方式需要根据项目使用的开发框架进行选择,目前在线提供2个版本开发框架,老开发框架和新开发框架,如何区分?请查看项目pom依赖:

新框架为spring boot工程，且在项目根目录下pom文件内，parent节点下有继承`com.cmos:itframe-boot:pom:*.SNAPSHOT`的配置信息，如下：

```
<parent>
    <groupId>com.cmos</groupId>
    <artifactId>itframe-boot</artifactId>
    <version>1.0.1-SNAPSHOT</version>
</parent>
```

反之没有且不是spring boot工程则为老框架。

1. 新开发框架

   基于javaConfig的spring配置,spring配置类上启用注解`@EnableCfgProperty`（比如spring boot工程也可以直接在入口函数类上启用）

2. 老开发框架

   spring配置文件中注册bean

```
<bean class="com.cmos.msg.common.MsgConfig" />
```

2. 配置中心key值命名支持如下（按以下优先配置顺序读取）：

配置**key**命名规范

遵守配置中心key值命名规范，key值增加前缀`cmos.msg.client.`，即在配置中心配置mq服务器地址属性的key值为：cmos.msg.client.rocketmq.namesrv.addr

注意：如果**cmos-msg**开启了配置中心配置，但是未在配置中心配置相关属性，或属性**key**值不正确，则默认继续从本地配置获取

3.2.2 本地配置文件

根据业务系统的情况不同，可以选用以下**3**种配置方式的某一种，按以下优先顺序选择：

- （优先推荐）有本地配置文件（如system.properties）且该配置文件内的配置是被加载到**VM**环境变量内的，添加如下一行配置：

```
rocketmq.namesrv.addr=192.168.0.1:9876;192.168.0.2:9876
```

- 有工程启动入口的方法，一般这种方法是在工程启动时初始化会被调用，方法体内添加如下配置：

```
System.getProperties().setProperty("rocketmq.namesrv.addr", "192.168.0.1:9876;192.168.0.2:9876");
```

4.快速开发

4.1 消息发送

4.1.1 普通消息异步发送(推荐)

4.1.1.1 场景说明

适用于响应效率要求比较高，事务一致性要求不强的业务逻辑，可以使用异步消息处理，如发短信、记日志等。

参数	说明

异步发送相比同步优势：

- 提高消息发送效率及降低等待响应延迟时间
- 提高消息发送TPS
- 增加消息发送结果异步回调处理机制

4.1.1.2 接口入参说明

```
public void sendDefaultMsg(String topic, String keys, String msg, SendCallback sendCallback)
        throws CmosMsgException;
```

参数	说明
topic	Topic名字
keys	发送消息的key值，该值应为可以标记本次消息的唯一值，用于在mq服务器根据此索引查询消息内容，建议使用消息报文中的id字段，且发送失败后需要把keys的日志打印处理，否则后续无法根据此值查询报文内容
msg	消息内容，如json报文，最大不能超过4M
sendCallback	消息发送后的回调通知接口，便于业务系统对此条消息发送结果进行业务处理

4.1.1.3 demo实例

步骤**1**：实现消息发送结果异步接口(SendCallback)，详细参见SendCallback接口说明，可选。

步骤**2**：直接在需要使用异步消息的业务类中通过spring注入SendCallback接口的实现类。

> 注意：
>
> SendCallback的接口实现类需为单例模式，spring的bean管理默认时单例模式，请不要直接通过new方式实例化SendCallback，避免回调接口实例过多。

步骤**3**：在业务类的方法中直接调用消息发送接口，如：

```
try {
      CmosMsgFactory.getRocketMQProducer().
                    sendDefaultMsg("SEND_SMS",
                            "20171128102945",
                            "{\"id\":\"20171128102945\",\"msg\":\"您已成功充值100元！\"}",
                            sendCallBack);       //禁止在此直接new此接口的实例对象
} catch (CmosMsgException e) {
    //异常处理包含以下 2 部分功能
    //1.记录TOPIC、消息key值、消息内容；
    //2.业务高可用策略，若MQ宕机或连接不上需要业务侧保障业务高可用，如同步接口或服务调用
}
```

4.1.2 Oneway发送

4.1.2.1 场景说明

不关心消息的发送结果，消息发送之后立即返回。

4.1.2.2 接口入参说明

参数	说明

```
public void sendDefaultMsg(String topic, String keys, String msg)
            throws CmosMsgException;
```

参数	说明
topic	Topic名字
keys	发送消息的key值，该值应为可以标记本次消息的唯一值，用于在mq服务器根据此索引查询消息内容，建议使用消息报文中的id字段，且发送失败后需要把keys的日志打印处理，否则后续无法根据此值查询报文内容
msg	消息内容，如json报文，最大不能超过4M

4.1.2.3 demo实例

步骤**1**：在业务类的方法中直接调用消息发送接口，如：

```
try {
      CmosMsgFactory.getRocketMQProducer().
                    sendDefaultMsg("SEND_SMS",
                           "20171128102945",
                           "{\"id\":\"20171128102945\",\"msg\":\"您已成功充值100元！\"}",
 } catch (CmosMsgException e) {
    //异常处理包含以下 2 部分功能
    //1.记录TOPIC、消息key值、消息内容，异常信息；
    //2.业务高可用策略，若MQ宕机或连接不上需要业务侧保障业务高可用，如同步接口或服务调用
 }
```

4.1.3 顺序消息发送

4.1.3.1 场景说明

需要按照消息的发送顺序来消费时使用。例如：一笔订单产生了 3 条消息，分别是订单创建、订单付款、订单完成。消费时，要按照顺序依次消费。同时多笔订单可以并行消费。

> 注意：
>
> **1**、顺序消息发送的效率会低于普通消息的发送效率，顺序消息发送的时候，每个操作的所有消息会发送到同一条队列，如果传入的**orderId**导致分配的队列不均，会导致某条队列消息积压多于其它队列，所以避免连续多次操作使用相同的**orderId**
>
> **2**、**orderId**的值：大于**topic**的队列总数且小于**int**的最大值（若不清楚**topic**的队列总数可咨询运维或找消息管理员在消息**WEB**端查看）

4.1.3.2 接口入参说明

```
public SendResult sendOrderMsg(String topic, String keys, String msg, int orderId)
            throws CmosMsgException;
```

参数	说明
topic	Topic名字
keys	发送消息的key值，该值应为可以标记本次消息的唯一值，用于在mq服务器根据此索引查询消息内容，建议使用消息报文中的id字段，且发送失败后需要把keys的日志打印处理，否则后续无法根据此值查询报文内容
msg	消息内容，如json报文
orderId	一组有序的消息的唯一索引号，mq是通过此字段确定几个消息是否是一组的

4.1.3.3 demo实例

步骤**1**：设置**orderId**

步骤**2**：在业务类的方法中直接调用顺序消息发送接口，如：

```
try {
  int orderId = 2017;
  CmosMsgFactory.getRocketMQProducer().
    sendOrderMsg("SEND_SMS",
                 "20171128102945",
                 "{\"id\":\"20171128102945\",\"msg\":\"用户cmos创建一笔订单\"}",
                 orderId);
    CmosMsgFactory.getRocketMQProducer().
    sendOrderMsg("SEND_SMS",
                 "20171128102945",
                 "{\"id\":\"20171128102945\",\"msg\":\"用户cmos已付款\"}",
                 orderId);
    CmosMsgFactory.getRocketMQProducer().
    sendOrderMsg("SEND_SMS",
                 "20171128102945",
                 "{\"id\":\"20171128102945\",\"msg\":\"用户cmos订单完成\"}",
                 orderId);
} catch (CmosMsgException e) {
    //异常处理包含以下 2 部分功能
    //1.记录TOPIC、消息key值、消息内容，异常信息；
    //2.业务高可用策略，若MQ宕机或连接不上需要业务侧保障业务高可用，如同步接口或服务调用
}
```

4.1.4 事务消息发送

4.1.4.1 场景说明

需要保证事务一致性的场景，比如帐号转账：A->B转账，A本地扣款，发消息给B加钱，A扣款失败，B不应该加钱

> 注意：
> 事务消息的发送效率低于普通消息的发送效率，大概发送一次事务消息可以发送两条普通消息

参数	说明

建议：

优于采用业务代码事务操作+普通消息发送的策略，例如：上面的账号转账例子，可以先判断是否本地扣款成功，再选择是否采用普通消息发送给B加钱的操作来保证事务一致性。如果业务代码事务操作+普通消息发送的策略不适用于当前业务场景，可以采用事物消息发送。

4.1.4.2 接口入参说明

```
public SendResult sendTransactionMsg(String topic, String keys, String msg,
ITransactionMsgExecuter transactionMsgExecuter)
            throws CmosMsgException ;
```

参数	说明
topic	Topic名字
keys	发送消息的key值，该值应为可以标记本次消息的唯一值，用于在mq服务器根据此索引查询消息内容，建议使用消息报文中的id字段，且发送失败后需要把keys的日志打印处理，否则后续无法根据此值查询报文内容
msg	消息内容，如json报文，最大不能超过4M
transactionMsgExecuter	事物回调处理接口

4.1.4.3 demo实例

步骤**1**：实现事务消息发送回调接口(ITransactionMsgExecuter)，详细参见ITransactionMsgExecuter接口说明。

步骤**2**：直接在需要使用异步消息的业务类中通过spring注入ITransactionMsgExecuter接口的实现类。

> 注意：ITransactionMsgExecuter的接口实现类需为单例模式，spring的bean管理默认是单例模式，请不要直接通过new方式实例化ITransactionMsgExecuter，避免回调接口实例过多。

步骤**3**：在业务类的方法中直接调用事务消息发送接口，如：

```
try {
    SendResult sendResult = CmosMsgFactory.getRocketMQProducer()
      .sendTransactionMsg("SEND_SMS",
                          "20171128102945",
                          "{\"id\":\"20171128102945\",\"msg\":\"您已成功充值100元！\"}",
                          transactionMsgExecuter);  //禁止在此直接new此接口的实例对象
  } catch (CmosMsgException e) {
    //异常处理包含以下 2 部分功能
    //1.记录TOPIC、消息key值、消息内容，异常信息；
    //2.业务高可用策略，若MQ宕机或连接不上需要业务侧保障业务高可用，如同步接口或服务调用
}
```

4.1.5 批量消息发送

4.1.5.1 场景说明

在一次操作中发送多条消息大小比较小的时候可以使用批量发送提升效率。总消息大小不应该超过1MB。

4.1.5.2 接口入参说明

参数	说明

```
public SendResult sendBatchMsgs(List<Message> messages) throws CmosMsgException;
```

参数	说明
messages	本次发送的消息集合,单条消息最大不能超过4M

4.1.5.3 demo实例

步骤**1**：将一次操作要发送的消息放入List集合。

步骤**2**：在业务类的方法中直接调用批量消息发送接口如：

```
List<Message> list = new ArrayList<>();
try {
      list.add(new Message("SEND_SMS", "用户cmos充值100元！".getBytes()));
      list.add(new Message("SEND_SMS", "用户itframe充值100元！".getBytes()));
      list.add(new Message("SEND_SMS", "用户msg充值100元！".getBytes()));
    } catch (Exception e) {
        //异常打印
    }
try {
    SendResult result = CmosMsgFactory.getRocketMQProducer().sendBatchMsgs(list);
} catch (CmosMsgException e) {
    //异常处理包含以下 2 部分功能
    //1.记录TOPIC、消息key值、消息内容，异常信息；
    //2.业务高可用策略，若MQ宕机或连接不上需要业务侧保障业务高可用，如同步接口或服务调用
}
```

4.2 消息消费

4.2.1 普通消息集群消费(推荐)

4.2.1.1 场景说明

对接收到的每条消息只需要进行相关业务处理

4.2.1.2 demo实例

步骤**1**：实现消息消费回调接口(IConsumerCallBack)，详细参见IConsumerCallBack接口说明。

步骤**2**：在业务类的方法中直接调用普通消息消费接口，入参传入IConsumerCallBack实现类实例，如：

```
try {
    CmosMsgFactory
      .getRocketMQConsumer()
      .subScribe("SMSNotice", // topic名字
                 new ConsumerCallBackImpl());    //IConsumerCallBack接口实现类实例
    } catch (CmosMsgException e) {
      // 异常处理，记录异常日志
    }
```

4.2.2 顺序消息消费

4.2.2.1 场景说明

订阅顺序消费的消息时，消息的顺序由mq保证，消费端不需要关心，只需要对收到的消息消费即可

4.2.2.2 demo示例

步骤**1**：实现消息消费回调接口(IConsumerCallBack)，详细参见IConsumerCallBack接口说明。

步骤**2**：在业务类的方法中直接调用顺序消息消费接口，入参传入IConsumerCallBack实现类实例，如：

```
try {
      CmosMsgFactory
        .getRocketMQConsumer()
        .subScribeOrder("SMSNotice", // topic名字
                   new ConsumerCallBackImpl());    //IConsumerCallBack接口实现类实例
    } catch (CmosMsgException e) {      // 异常处理，记录异常日志
    }
```

4.2.3 事物消息消费

4.2.3.1 场景说明

事物消息的场景和普通消息的消费场景类似，事物一致性由发送端保证，消费端不需要关心也不需要感知，只需要对收到的消息消费即可

4.2.3.2 demo示例

步骤**1**：实现消息消费回调接口(IConsumerCallBack)，详细参见IConsumerCallBack接口说明。

步骤**2**：在业务类的方法中直接调用普通消息消费接口，入参传入IConsumerCallBack实现类实例，如：

```
try {
    CmosMsgFactory
      .getRocketMQConsumer()
      .subScribe("SMSNotice", // topic名字
                 new ConsumerCallBackImpl());    //IConsumerCallBack接口实现类实例
    } catch (CmosMsgException e) {
      // 异常处理，记录异常日志
    }
```

4.2.4 定时批量消费消息

4.2.4.1 场景说明

对消息响应及时性要求不高，及消息接收频率不高的情况下可以考虑定时批量消费，消息由消费端定时主动拉取

4.2.4.2 demo示例

步骤**1**：实现批量消息消费回调接口(IConsumerBatchCallBack)，详细参见IConsumerBatchCallBack接口说明。

步骤**2**：在业务类的方法中直接调用定时批量消息消费接口，如：

```
try {
      CmosMsgFactory
        .getRocketMQConsumer()
        .pullSchedule("SMSNotice",  // topic名字
                      10,            // 一次拉取消息消费的最大数量
                      10,            // 定时的间隔时间
                      new ConsumerBatchCallBackImpl());// IConsumerBatchCallBack接口实现类实例
    } catch (CmosMsgException e) {
      // 异常处理，记录异常日志
    }
```

4.2.5 广播消费消息

4.2.5.1 场景说明

每一条消息会被多个订阅的消费端消费。每个消费端的业务逻辑可以相同也可以不相同。

4.2.5.2 demo示例

步骤**1**：实现消息消费回调接口(IConsumerCallBack)，详细参见IConsumerCallBack接口说明。

步骤**2**：在业务类的方法中直接调用广播消息消费接口，如：

```
try {
      CmosMsgFactory
        .getRocketMQConsumer()
        .subScribeBroadcast("SMSNotice",  // topic名字
                      new ConsumerCallBackImpl());// IConsumerCallBack接口实现类实例
    } catch (CmosMsgException e) {
      // 异常处理，记录异常日志
    }
```

5.附录

5.1 SendCallback接口说明

5.1.1 使用说明

SendCallback接口为普通消息发送的异步回调接口，在进行消息发送时需要传入该接口的实例类的实例，当消息发送完成后，可以在该接口的onSuccess方法体内对发送结果进行相关处理，或出现异常在onException方法体内进行异常处理。

因为消息发送属于重复操作（不像消息消费订阅一次即可），所以禁止匿名实现。

消息发送完成返回结果SendResult说明请参看SendResult对象说明

5.1.2 demo示例

```
public class SendCallbackImpl implements SendCallback {
    //以下2个方法不允许在往外抛任何异常
    @Override
    public void onSuccess(SendResult sendResult) {
      //TODO: 在此添加对返回结果业务处理代码
      try {

      }catch (Exception e) {

      }
    }

    @Override
    public void onException(Throwable throwable) {
         //TODO：在此添加对异常处理的代码，日志记录消息内容
    }
}
```

5.2 ITransactionMsgExecuter接口说明

5.2.1 使用说明

ITransactionMsgExecuter接口为执行事务消息时的回调接口，在该接口的execute方法内添加相关业务逻辑代码，执行其它操作，如执行失败，为了保持事务一致性，本次发送的消息不会发送给消费端处理。

5.2.2 demo示例

```
public class TransactionExecuterImpl implements ITransactionMsgExecuter {
    @Override
    public void execute(Message message) {
        // TODO：事务逻辑处理，此方法不要吃掉异常，mq是根据此方法是否有异常来决定消息是否发送.
        // 异常有 2 种处理方式：
        // 1.不进行任何异常处理
         // 2.捕捉异常,在catch块中记录异常信息，然后再进行二次抛出
    }
}
```

5.3 IConsumerCallBack接口说明

5.3.1 使用说明

IConsumerCallBack接口为消息消费回调接口，对于接收到的消息在该接口的process方法体内，进行相关业务处理

对于接收到的消息，打印topic的msgId的日志信息，如果在消费过程中发生问题，更加容易定位

5.3.2 demo示例

```
class ConsumerCallBackImpl implements IConsumerCallBack {
   @Override
   public void process(MessageExt msg) {
     // TODO: 这里添加对接收到的消息的业务处理逻辑,此方法内必须全捕获异常，且不允许再往上抛
       try {
         // 打印日志topic和msgId，如果出现问题，根据这条日志信息更加容易定位
         logger.info("topic:" + msg.getTopic() + ",msgId:" + msg.getMsgId());
         // 获得消息体
         String body = new String(msg.getBody(), "utf-8");
         // 获得消息id
         String msgId = msg.getMsgId();
         // 获得keys
         String keys = msg.getKeys();
         // 获得topic
         String topic = msg.getTopic();
       } catch (Exception e) {
         //异常处理，不能再往外抛，打印topic，msgId，body日志,便于故障定位分析
       }
   }
}
```

5.4 IConsumerBatchCallBack接口说明

5.4.1 使用说明

IConsumerBatchCallBack接口为批量消息消费的回调接口，会同时收到消息的一个List集合，对每一条消息进行处理。

对于接收到的消息，打印topic的msgId的日志信息，如果在消费过程中发生问题，更加容易定位。

消费过程中发生的异常必须全部捕捉进行相关处理，禁止向上层抛

5.4.2 demo示例

```
public class ConsumerBatchCallBackImpl implements IConsumerBatchCallBack {
    @Override
    public void process(List<MessageExt> msgs) {
         try {
           for (MessageExt messageInfo : msgs) {
            // 打印日志topic和msgId，如果出现问题，根据这条日志信息更加容易定位
             logger.info("topic:" + msg.getTopic() + ",msgId:" + msg.getMsgId());
              // TODO: 这里添加对接收到的消息的业务处理逻辑

             }
         }catch (Exception e){
          //异常处理，不能再往外抛，打印topic，msgId，body日志,便于故障定位分析
         }
    }
}
```

5.5 SendResult对象说明

SendResult为消息发送完成后的返回结果，各字段列表如下：

字段	说明
sendStatus	发送状态
msgId	消息ID
messageQueue	消息队列
queueOffset	队列偏移

各字段详细说明如下：

- sendStatus

状态	说明
SEND_OK	消息发送成功
FLUSH_DISK_TIMEOUT	消息发送成功，但是服务器刷盘超时，消息已经进入服务器队列，只有此时服务器宕机，消息才会丢失
FLUSH_SLAVE_TIMEOUT	消息发送成功，但是服务器同步到 Slave 时超时，消息已经进入服务器队列，只有此时服务器宕机，消息才会丢失
SLAVE_NOT_AVAILABLE	消息发送成功，但是此时 slave 不可用，消息已经进入服务器队列，只有此时服务器宕机，消息才会丢失

- msgId

消息ID，全局唯一标识符，但是在去重处理上建议使用消息内容的唯一标识字段（比如订单ID等）

- messageQueue

该字段为一个MessageQueue类对象，有3个字段（topic,brokerName,queueId）来确定该条消息在哪条队列

- queueOffset

该条消息在当前队列的偏移值（该队列的第几条消息）

- transactionId

事物ID

- offsetMsgId
- regionId
- traceOn

6. 老版本升级步骤

1. 升级cmos-msg的pom依赖，将版本号替换为2.0.0-SNAPSHOT，并确认工程中是否已正常下载此包
2. 删除原工程中因引导入最新版本所产生的编译报错内容，如:import 错误，方法api错误等
3. 重新按此此文档进行集成，消息发送或订阅参照第4节说明重新接入