# MQ client使用指导

MQ Client封装了Kafka和JetMQ的差异,屏蔽了客户端对MQ类型的依赖. 客户端使用时,需要将 [commons.mq.kafka.impl-dist.tar.gz](https://szxsvn01-in:3690/svn/ASSIDSVN/ereading/SDP DEP/DOC/发布包/MQ/installPkg/commons.mq.kafka.impl-${commons.mq.version}-dist.tar.gz, "Kafka Client API") 下载到本地,将所有jar包解压后放入应用的*classpath,将conf*目录下的两个文件放入应用的*conf*目录.

## 设计思路

kafka支持通过topic进行消息传输和定义，为简化生产者和消费者对kafka底层特性的隔离，同时具体kafka的高性能和高可靠性。

- 系统启动时通过start方法启动生产者和消费者；
- 生产者可以选择多种方法进行消息发送，包括同步，异步，异步并且对结果进行回调处理；
- 生产者发送消息的消息体包含字段，***topic***, ***tags数组***，traceuniqid消息跟踪id,seqid消息序列, body消息对象, type消息javabean对象类型，仅有topic和tags必填，tags可以支持多个tag；
- 消费者启动前需要订阅对应的主题和过滤字段tag，topic与tag唯一指定一个对应的消息处理器；
- 系统关闭前调用生产者和消费者的shutdown方法；
- 多个消费者可以通过合理同一个群组id进行集群消费；
- 每个消费者对象可以通过设置coreSize指定消费线程个数，每个线程可以连接同一个topic下个多个partition，每个topic个每个partition仅能被一个消费者线程连接；
- 当kafka节点个数或者消费者连接个数变更时会自动触发消费集群的负载均衡；
- 由于kafka本身不支持服务端字段过滤，所以这里实现的tag过滤实在消费端实现的，当同一组内不同的消费者处理逻辑不同时，消息拉取到本地而不处理时，其他需要该消息的消费者再也无法拉取该消息进行处理。

## 客户端开发

客户端和服务端如果通过maven开发,可以通过pom中加入下面依赖进行开发

```
<dependency>
    <groupId>com.huawei.jaguar</groupId>
    <artifactId>commons.mq</artifactId>
    <version>${commons.mq.version}</version>
</dependency>

<dependency>
    <groupId>com.huawei.jaguar</groupId>
    <artifactId>commons.mq.kafka.impl</artifactId>
    <version>${commons.mq.version}</version>
    <scope>runtime</scope>
</dependency>
```

### 消费者代码样例

消费者启动后注册一个订阅一个topic,注册一个监听器(用于收到消息后的逻辑处理,类似servlet).样例代码如下:

```
//声明一个bean引用,指定消费消息的对象类型,如Date
@Autowired
private MQPushConsumer<Date> pushConsumer;

初始化方法中初始化调用下面代码一次
    try
    {
        //订阅主题,第二个参数是tags,支持字符串包含，*，或操作，如abc,ab||cd,*
        pushConsumer.subscribe("topic", "*", new MessageListener<Date>()
        {
            //消息处理逻辑,传入的类型为业务自定义的类型,要求和服务端的类型相同,
            //其中每个属性名称和类型也要对应匹配,不要求package完全相同
            @Override
            public void processMsg(Date t)
            {
                System.out.println(t);
```

```
            }

            //提供消息体的具体类型,供框架反序列使用,不得为空
            @Override
            public Class<Date> getType()
            {
                return Date.class;
            }
        });

        pushConsumer.start();
    }
    catch (MQClientException e)
    {
        e.printStackTrace();
        fail(e.getMessage());
    }

    //系统关闭前调用方法
    pushConsumer.shutdown();
```

**复杂消费者对象解析样例**

对于生产者对象包含多层抽象类，消费者无法通过topic准确定义java对象类型并通过fastjson直接映射解析的时候，可以通过下面样例按照JSon格式接口说明进行解析处理：

```
......
MessageListener<String> messageListener = new MessageListener<String>()
{
    //接受json格式原始字符串，按照接口说明书解析所需字段后处理
    public void processMsg(String t)
    {
        //TODO: 解析json格式的消息体t;
        //TODO: 调用消费者业务代码处理;
    }

    //不需要fastjson自行解析时直接返回String.class
    @Override
    public Class<String> getType()
    {
         return String.class;
    }
};
......
```

## 生产者代码样例

生产者启动时需要连接broker,连接后需要的时候进行消息发送,系统关闭时记得关闭连接.样例代码如下:

```
    //声明beanid
    @Autowired
    private MQProducer producer;

    //系统启动时保证调用启动方法
    producer.start();

    //消息组装并发送
    Message msg = new Message();
    String topic = "ghz";
    msg.setTopic(topic);
    class Person{
        private String name;
        private boolean sex;
        private int age;

        public void setName(String name)
```

```
        {
            this.name = name;
        }
        public void setSex(boolean sex)
        {
            this.sex = sex;
        }
        public void setAge(int age)
        {
            this.age = age;
        }
        @Override
        public String toString()
        {
            return "Person [name=" + name + ", sex=" + sex + ", age=" + age + "]";
        }
    }
    Person p=new Person();
    p.setAge(15);
    p.setName("Same");
    p.setSex(false);

    msg.setBody(p);
    msg.setTags(new String[] {"abc","def"});
    try
    {
        producer.sendOneway(msg);
    }
    catch Exception e)
    {
        fail("Not yet implemented");
    }

    //系统关闭时记得关闭方法
    producer.shutdown();
```

## 配置文件说明

### 生产者客户端配置文件

生产段客户端配置文件mqproducerconfig.properties位于conf目录下,可以根据官方参数指导文档添加，默认包含如下参数

```
#This is for bootstrapping and the producer will only use it for getting metadata (topics, partitions and replicas).
#The socket connections for sending the actual data will be established based on the broker information returned in the metadata.
#The format is host1:port1,host2:port2, and the list can be a subset of brokers or a VIP pointing to a subset of brokers.
metadata.broker.list=nkgciscslx00109:9092

#The serializer class for messages. The default encoder 'kafka.serializer.DefaultEncoder' takes a byte[] and returns the same byte
serializer.class=kafka.serializer.StringEncoder

#This parameter specifies whether the messages are sent asynchronously in a background thread.
#Valid values are (1) async for asynchronous send and (2) sync for synchronous send.
#By setting the producer to async we allow batching together of requests (which is great for throughput)
#but open the possibility of a failure of the client machine dropping unsent data.
producer.type=async
```

### 消费者客户端配置文件

消费者客户端配置文件mqpushconsumerconfig.properties位于conf目录下,可以根据官方参数指导文档添加，默认包含如下参数

```
#Specifies the ZooKeeper connection string in the form hostname:port where host and port are the host and port of a ZooKeeper serv
#To allow connecting through other ZooKeeper nodes when that ZooKeeper machine is down you can also specify multiple hosts in the
#   hostname1:port1,hostname2:port2,hostname3:port3.
zookeeper.connect=nkgciscslx00109:2181
```

```
#A string that uniquely identifies the group of consumer processes to which this consumer belongs.
#By setting the same group id multiple processes indicate that they are all part of the same consumer group.
group.id=c1

#The frequency in ms that the consumer offsets are committed to zookeeper.
auto.commit.interval.ms=60000

#If true, periodically commit to ZooKeeper the offset of messages already fetched by the consumer.
#This committed offset will be used when the process fails as the position from which the new consumer will begin.
auto.commit.enable=true
```

# 注意：

- 生产端和客户端完全解耦，双方通过Json格式进行消息通讯；
- 生产端和消费端可以通过String格式进行消息通讯；
- 为节省生产端和服务端的工作量，可以通过自定义的javabean进行处理，底层默认通过`fastjson`进行编解码；
- 生产端默认调用`com.alibaba.fastjson.JSON.toJSONString(Object)`进行编码；
- 消费者通过调用消费者提供的`com.huawei.jaguar.commons.mqclient.consumer.MessageListener.getType()`获取到对象类型后通过`com.huawei.jaguar.commons.mqclient.consumer.MessageListener.processMsg(T)`回调业务api进行处理；
- 鉴于`fastjson`的约束，消息中没有保存原始的消息体对象类型，所以双方需要遵循标准`json`格式的标准. 定义的对象类型需要提供get set方法；
- 当消费者定义的对象通过topic无法准确定义对象类型时将无法正常解析消息，这时消费者仅能
  在`com.huawei.jaguar.commons.mqclient.consumer.MessageListener.getType()`通过返回String类型，
  在`com.huawei.jaguar.commons.mqclient.consumer.MessageListener.processMsg(T)`中对json格式进行自动解析处理。