课程介绍

- 了解什么RocketMQ
- 了解RocketMQ的核心概念
- 动手安装RocketMQ服务
- 快速入门，掌握RocketMQ的api使用
- 对producer、consumer进行详解
- 了解RocketMQ的存储特点

1、RocketMQ简介与安装

1.1、RocketMQ简介

Apache RocketMQ是一个采用Java语言开发的分布式的消息系统，由阿里巴巴团队开发，与2016年底贡献给Apache，成为了Apache的一个顶级项目。

在阿里内部，RocketMQ 很好地服务了 集 团大大小小上千个应用，在每年的双十一当天，更有不可思议的万亿级消息通过 RocketMQ 流转(在 2017 年的双十一当天，整个阿里巴巴集团通过 RocketMQ 流转的线上消息达到了 万亿级，峰值 TPS 达到 5600 万)，在阿里大中台策略上发挥着举足轻重的作用 。

地址：http://rocketmq.apache.org/

1.2、RocketMQ的历史发展

- 阿里巴巴消息中间件起源 于 2001 年的五彩石项目， Notify 在这期间应运而生，用于交易核心消息的流转 。
- 2010 年， B2B 开始大规模使用 ActiveMQ 作为消息内核，随着阿里业务 的快速发展，急需一款支持顺序消息，拥有海量消息堆积能力的消息中间件， MetaQ 1.0 在 2011 年诞生 。
- 2012年， MetaQ已经发展到了3.0版本，并抽象出了通用的消息引擎 RocketMQ。 随后，对 RocketMQ 进行了开源 ， 阿里的消息中间件正式走人了 公众视野 。
- 2015年， RocketMQ已经经历了多年双十一的洗礼，在可用性、 可靠性以 及稳定性等方面都有出色的表现。与此同时 ，云计算大行其道， 阿里消息中间 件基于 RocketMQ推出了 Aliware MQ 1.0，开始为阿里云上成千上万家企业提 供消息服务 。

- 2016 年， MetaQ 在双十一期间承载了万亿级消息的流转，跨越了一个新的里程碑 ，同时 RocketMQ 进入 Apache 孵化 。

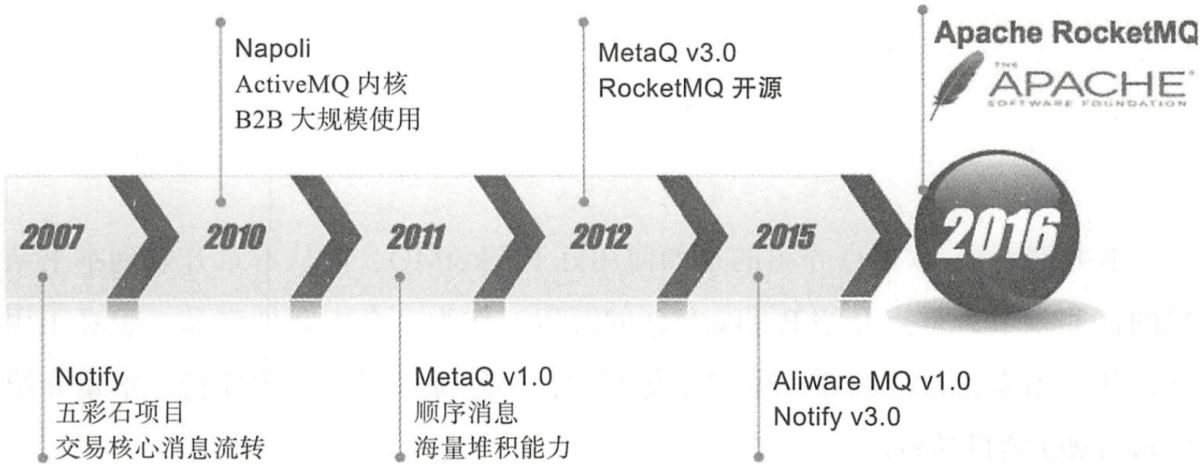
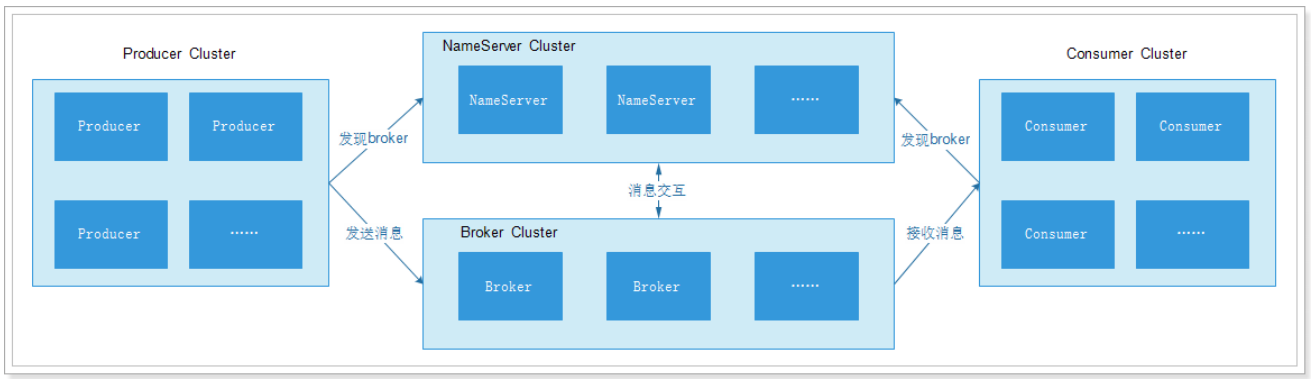


图 9-1　RocketMQ 演进历史

> 内容以及图片来源于《RocketMQ实战与原理解析》

1.3、核心概念说明



- Producer
  - 消息生产者，负责产生消息，一般由业务系统负责产生消息。
  - Producer Group
    - 一类 Producer 的集合名称，这类 Producer 通常发送一类消息，且发送逻辑一致。
- Consumer
  - 消息费者，负责消费消息，一般是后台系统负责异步消费。
  - Push Consumer
    - 服务端向消费者端推送消息
  - Pull Consumer
    - 消费者端向服务定时拉取消息
  - Consumer Group
    - 一类 Consumer 的集合名称，这类 Consumer 通常消费一类消息，且消费逻辑一致。
- NameServer
  - 集群架构中的组织协调员

    - 收集broker的工作情况
    - 不负责消息的处理
- Broker
    - 是RocketMQ的核心负责消息的发送、接收、高可用等（真正干活的）
    - 需要定时发送自身情况到NameServer，默认10秒发送一次，超时2分钟会认为该broker失效。
- Topic
    - 不同类型的消息以不同的Topic名称进行区分，如User、Order等
    - 是逻辑概念
    - Message Queue
        - 消息队列，用于存储消息

1.4、部署安装

1.4.1、下载

下载地址：https://www.apache.org/dyn/closer.cgi?path=rocketmq/4.3.2/rocketmq-all-4.3.2-bin-release.zip

版本使用目前最新版：4.3.2

1.4.2、非Docker安装

```
cd /haoke
unzip rocketmq-all-4.3.2-bin-release.zip
cd rocketmq-all-4.3.2-bin-release

#启动nameserver
bin/mqnamesrv
# The Name Server boot success. serializeType=JSON 看到这个表示已经提供成功
#启动broker
bin/mqbroker -n 172.16.185.55:9876  #-n 指定nameserver地址和端口
#启动出错
Java HotSpot(TM) 64-Bit Server VM warning: INFO:
os::commit_memory(0x00000005c0000000, 8589934592, 0) failed; error='Cannot allocate
memory' (errno=12)
.....
```

启动错误，是因为内存不够，导致启动失败，原因：RocketMQ的配置默认是生产环境的配置，设置的jvm的内存大小值比较大，对于学习而言没有必要设置这么大，测试环境的内存往往都不是很大，所以需要调整默认值。

```
#调整默认的内存大小参数
cd bin/
vim runserver.sh
JAVA_OPT="${JAVA_OPT} -server -Xms128m -Xmx128m -Xmn128m -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=128m"

cd bin/
vim runbroker.sh
JAVA_OPT="${JAVA_OPT} -server -Xms128m -Xmx128m -Xmn128m"

#从新启动测试
bin/mqbroker -n 172.16.55.185:9876
The broker[itcast, 172.17.0.1:10911] boot success. serializeType=JSON and name server is 172.16.185.55:9876
```

下面进行发送消息测试：

```
export NAMESRV_ADDR=127.0.0.1:9876
cd bin
sh tools.sh org.apache.rocketmq.example.quickstart.Producer

#测试结果
SendResult [sendStatus=SEND_OK, msgId=AC110001473C7D4991AD336AEA5703E0, offsetMsgId=AC11000100002A9F00000000000E8580, messageQueue=MessageQueue [topic=TopicTest, brokerName=itcast, queueId=3], queueOffset=1323]
SendResult [sendStatus=SEND_OK, msgId=AC110001473C7D4991AD336AEA5903E1, offsetMsgId=AC11000100002A9F00000000000E8634, messageQueue=MessageQueue [topic=TopicTest, brokerName=itcast, queueId=0], queueOffset=1323]
SendResult [sendStatus=SEND_OK, msgId=AC110001473C7D4991AD336AEA5F03E2, offsetMsgId=AC11000100002A9F00000000000E86E8, messageQueue=MessageQueue [topic=TopicTest, brokerName=itcast, queueId=1], queueOffset=1323]
SendResult [sendStatus=SEND_OK, msgId=AC110001473C7D4991AD336AEA6103E3, offsetMsgId=AC11000100002A9F00000000000E879C, messageQueue=MessageQueue [topic=TopicTest, brokerName=itcast, queueId=2], queueOffset=1323]

#可以正常发送消息
```

测试接收消息：

```
sh tools.sh org.apache.rocketmq.example.quickstart.Consumer

#测试结果
ConsumeMessageThread_7 Receive New Messages: [MessageExt [queueId=2, storeSize=180,
queueOffset=1322, sysFlag=0, bornTimestamp=1544456244818,
bornHost=/172.16.55.185:33702, storeTimestamp=1544456244819,
storeHost=/172.17.0.1:10911, msgId=AC11000100002A9F00000000000E84CC,
commitLogOffset=951500, bodyCRC=684865321, reconsumeTimes=0,
preparedTransactionOffset=0, toString()=Message{topic='TopicTest', flag=0,
properties={MIN_OFFSET=0, MAX_OFFSET=1325, CONSUME_START_TIME=1544456445397,
UNIQ_KEY=AC110001473C7D4991AD336AEA5203DF, WAIT=true, TAGS=TagA}, body=[72, 101, 108,
108, 111, 32, 82, 111, 99, 107, 101, 116, 77, 81, 32, 57, 57, 49],
transactionId='null'}]]
ConsumeMessageThread_6 Receive New Messages: [MessageExt [queueId=2, storeSize=180,
queueOffset=1323, sysFlag=0, bornTimestamp=1544456244833,
bornHost=/172.16.55.185:33702, storeTimestamp=1544456244835,
storeHost=/172.17.0.1:10911, msgId=AC11000100002A9F00000000000E879C,
commitLogOffset=952220, bodyCRC=801108784, reconsumeTimes=0,
preparedTransactionOffset=0, toString()=Message{topic='TopicTest', flag=0,
properties={MIN_OFFSET=0, MAX_OFFSET=1325, CONSUME_START_TIME=1544456445397,
UNIQ_KEY=AC110001473C7D4991AD336AEA6103E3, WAIT=true, TAGS=TagA}, body=[72, 101, 108,
108, 111, 32, 82, 111, 99, 107, 101, 116, 77, 81, 32, 57, 57, 53],
transactionId='null'}]]

#从结果中，可以看出，接收消息正常
```

1.4.3、编写Java代码进行测试

第一步，创建itcast-rocketmq工程

第二步，导入依赖

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>cn.itcast.rocketmq</groupId>
    <artifactId>itcast-rocketmq</artifactId>
    <version>1.0-SNAPSHOT</version>

    <dependencies>
        <dependency>
            <groupId>org.apache.rocketmq</groupId>
            <artifactId>rocketmq-client</artifactId>
            <version>4.3.2</version>
        </dependency>
    </dependencies>

    <build>
        <plugins>
```

```
            <!-- java编译插件 -->
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.2</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                    <encoding>UTF-8</encoding>
                </configuration>
            </plugin>
        </plugins>
    </build>


</project>
```

第三步，编写代码

```
package cn.itcast.rocketmq;

import org.apache.rocketmq.client.producer.DefaultMQProducer;
import org.apache.rocketmq.client.producer.SendResult;
import org.apache.rocketmq.common.message.Message;
import org.apache.rocketmq.remoting.common.RemotingHelper;

public class SyncProducer {
    public static void main(String[] args) throws Exception {
        //Instantiate with a producer group name.
        DefaultMQProducer producer = new
            DefaultMQProducer("test-group");
        // Specify name server addresses.
        producer.setNamesrvAddr("172.16.55.185:9876");
        //Launch the instance.
        producer.start();

        for (int i = 0; i < 100; i++) {
            //Create a message instance, specifying topic, tag and message body.
            Message msg = new Message("TopicTest11" /* Topic */,
                "TagA" /* Tag */,
                ("Hello RocketMQ " +
                    i).getBytes(RemotingHelper.DEFAULT_CHARSET) /* Message body */
            );
            //Call send message to deliver message to one of brokers.
            SendResult sendResult = producer.send(msg);
            System.out.printf("%s%n", sendResult);
        }
        //Shut down once the producer instance is not longer in use.
        producer.shutdown();
    }
}
```

测试：

```
Exception in thread "main"
org.apache.rocketmq.remoting.exception.RemotingTooMuchRequestException:
sendDefaultImpl call timeout
    at
org.apache.rocketmq.client.impl.producer.DefaultMQProducerImpl.sendDefaultImpl(Defaul
tMQProducerImpl.java:612)
    at
org.apache.rocketmq.client.impl.producer.DefaultMQProducerImpl.send(DefaultMQProducer
Impl.java:1253)
    at
org.apache.rocketmq.client.impl.producer.DefaultMQProducerImpl.send(DefaultMQProducer
Impl.java:1203)
    at
org.apache.rocketmq.client.producer.DefaultMQProducer.send(DefaultMQProducer.java:214
)
    at cn.itcast.rocketmq.SyncProducer.main(SyncProducer.java:26)
```

测试结果会发现，发送消息会报错。

原因是什么呢？

仔细观察broker启动的信息：

```
The broker[itcast, 172.17.0.1:10911] boot success. serializeType=JSON and name server
is 172.16.185.55:9876
```

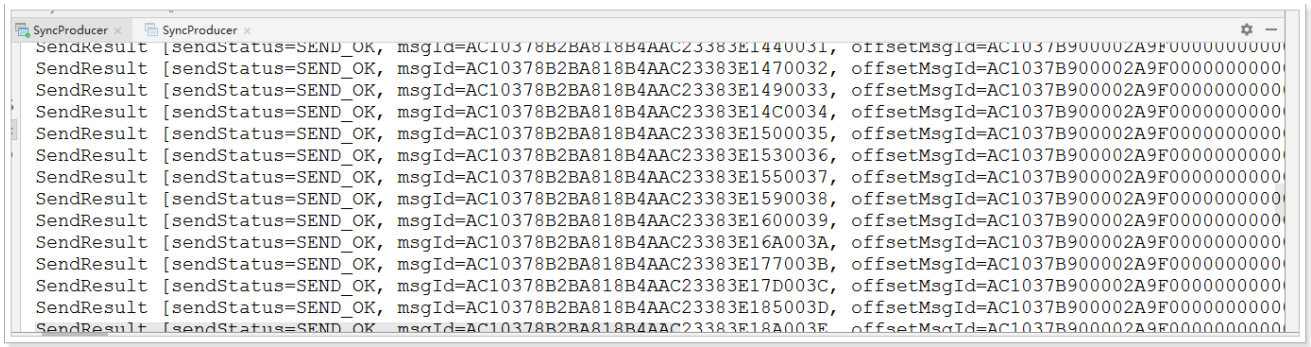
会发现，broker的ip地址是172.17.0.1，那么在开发机上是不可能访问到的。

所以，需要指定broker的ip地址。

```
#创建broker配置文件
vim /haoke/rmq/rmqbroker/conf/broker.conf
brokerIP1=172.16.55.185
namesrvAddr=172.16.55.185:9876
brokerName=broker_haoke_im

#启动broker，通过 -c 指定配置文件
bin/mqbroker -c /haoke/rmq/rmqbroker/conf/broker.conf
The broker[itcast, 172.16.55.185:10911] boot success. serializeType=JSON and name
server is 172.16.55.185:9876  #这样就可以进行访问了
```

测试：



看到，可以发送消息了。问题得到解决。

1.4.3、通过docker安装

```
#拉取镜像
docker pull foxiswho/rocketmq:server-4.3.2
docker pull foxiswho/rocketmq:broker-4.3.2

#创建nameserver容器
docker create -p 9876:9876 --name rmqserver \
-e "JAVA_OPT_EXT=-server -Xms128m -Xmx128m -Xmn128m" \
-e "JAVA_OPTS=-Duser.home=/opt"  \
-v /haoke/rmq/rmqserver/logs:/opt/logs \
-v /haoke/rmq/rmqserver/store:/opt/store \
foxiswho/rocketmq:server-4.3.2

#创建broker容器
docker create -p 10911:10911 -p 10909:10909 --name rmqbroker \
-e "JAVA_OPTS=-Duser.home=/opt"  \
-e "JAVA_OPT_EXT=-server -Xms128m -Xmx128m -Xmn128m" \
-v /haoke/rmq/rmqbroker/conf/broker.conf:/etc/rocketmq/broker.conf \
-v /haoke/rmq/rmqbroker/logs:/opt/logs \
-v /haoke/rmq/rmqbroker/store:/opt/store \
foxiswho/rocketmq:broker-4.3.2

#启动容器
docker start rmqserver rmqbroker

#停止删除容器
docker stop rmqbroker rmqserver
docker rm rmqbroker rmqserver
```

经测试，可以正常发送、接收消息。

1.4.4、部署RocketMQ的管理工具

RocketMQ提供了UI管理工具，名为rocketmq-console，项目地址：https://github.com/apache/rocketmq-externals/tree/master/rocketmq-console

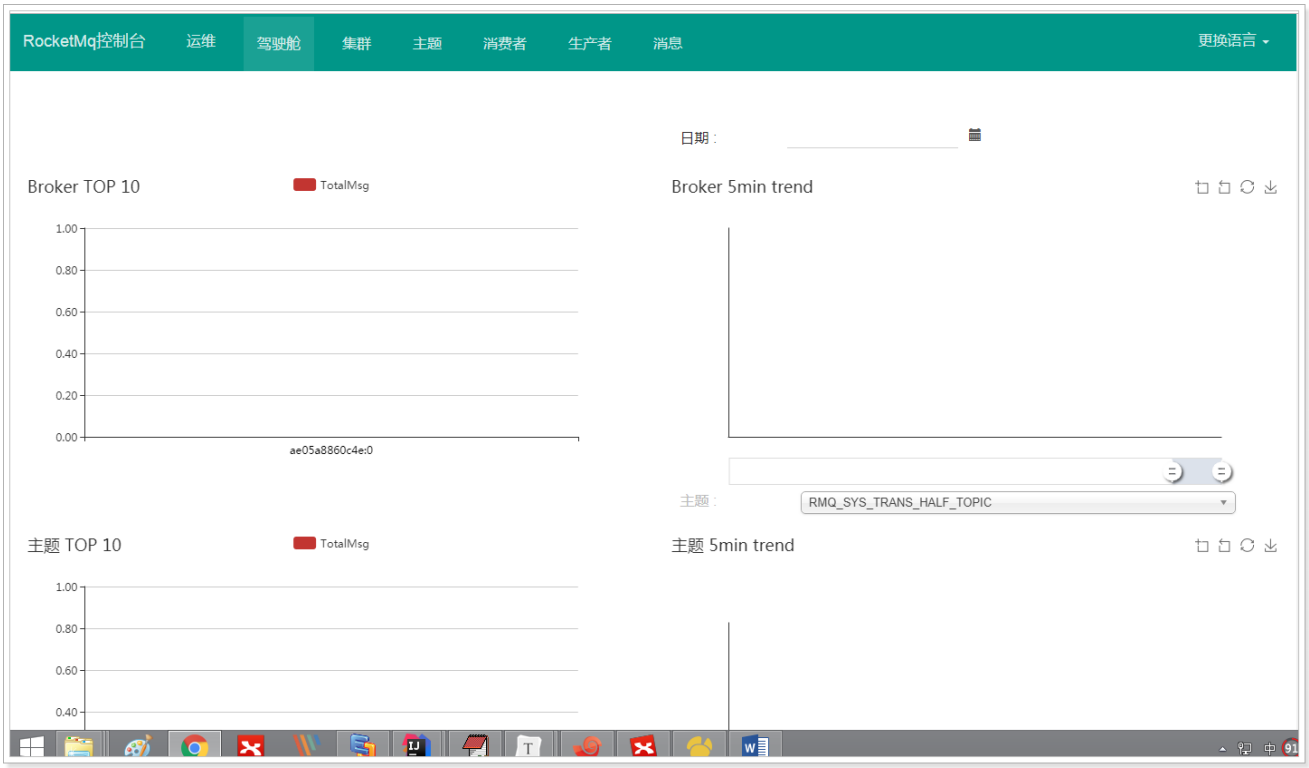
该工具支持docker以及非docker安装，这里我们选择使用docker安装

```
#拉取镜像
docker pull styletang/rocketmq-console-ng:1.0.0

#创建并启动容器
docker run -e "JAVA_OPTS=-Drocketmq.namesrv.addr=172.16.55.185:9876 -Dcom.rocketmq.sendMessageWithVIPChannel=false" -p 8082:8080 -t styletang/rocketmq-console-ng:1.0.0
```

通过浏览器进行访问：http://172.16.55.185:8082/#/

切换语言到中文，所有的功能就一目了然了。



2、快速入门

2.1、创建topic

```
package cn.itcast.rocketmq;

import org.apache.rocketmq.client.exception.MQClientException;
import org.apache.rocketmq.client.producer.DefaultMQProducer;
import org.apache.rocketmq.client.producer.MQProducer;

public class TopicDemo {

    public static void main(String[] args) throws MQClientException {
        DefaultMQProducer producer = new DefaultMQProducer("HAOKE_IM");
        producer.setNamesrvAddr("172.16.55.185:9876");

        producer.start();

        /**
```

```
         * key：broker名称
         * newTopic：topic名称
         * queueNum：队列数（分区）
         */
        producer.createTopic("broker_haoke_im", "haoke_im_topic", 8);

        System.out.println("创建topic成功");

        producer.shutdown();
    }

}

```

创建成功：

2.2、发送消息（同步）

```
package cn.itcast.rocketmq;

import org.apache.rocketmq.client.producer.DefaultMQProducer;
import org.apache.rocketmq.client.producer.SendResult;
import org.apache.rocketmq.common.message.Message;
import org.apache.rocketmq.remoting.common.RemotingHelper;

public class SyncProducer {
    public static void main(String[] args) throws Exception {
        DefaultMQProducer producer = new DefaultMQProducer("HAOKE_IM");
        producer.setNamesrvAddr("172.16.55.185:9876");
        producer.start();

        String msgStr = "用户A发送消息给用户B";
        Message msg = new Message("haoke_im_topic","SEND_MSG",
                msgStr.getBytes(RemotingHelper.DEFAULT_CHARSET));

        // 发送消息
        SendResult sendResult = producer.send(msg);

        System.out.println("消息状态：" + sendResult.getSendStatus());
        System.out.println("消息id：" + sendResult.getMsgId());
        System.out.println("消息queue：" + sendResult.getMessageQueue());
        System.out.println("消息offset：" + sendResult.getQueueOffset());

        producer.shutdown();
    }
}

```

```
打印结果：
消息状态：SEND_OK
消息id：AC1037A0307418B4AAC2374062400000
消息queue：MessageQueue [topic=haoke_im_topic, brokerName=broker_haoke_im, queueId=6]
消息offset：0
```

2.2.1、Message数据结构

字段名	默认值	说明
Topic	null	必填，线下环境不需要申请，线上环境需要申请后才能使用
Body	null	必填，二进制形式，序列化由应用决定，Producer 与 Consumer 要协商好序列化形式。
Tags	null	选填，类似于 Gmail 为每封邮件设置的标签，方便服务器过滤使用。目前只支持每个消息设置一个 tag，所以也可以类比为 Notify 的 MessageType 概念
Keys	null	选填，代表这条消息的业务关键词，服务器会根据 keys 创建哈希索引，设置后，可以在 Console 系统根据 Topic、Keys 来查询消息，由于是哈希索引，请尽可能保证 key 唯一，例如订单号，商品 Id 等。
Flag	0	选填，完全由应用来设置，RocketMQ 不做干预
DelayTimeLevel	0	选填，消息延时级别，0 表示不延时，大于 0 会延时特定的时间才会被消费
WaitStoreMsgOK	TRUE	选填，表示消息是否在服务器落盘后才返回应答。

2.3、发送消息（异步）

```
package cn.itcast.rocketmq;

import org.apache.rocketmq.client.producer.DefaultMQProducer;
import org.apache.rocketmq.client.producer.SendCallback;
import org.apache.rocketmq.client.producer.SendResult;
import org.apache.rocketmq.common.message.Message;
import org.apache.rocketmq.remoting.common.RemotingHelper;

public class AsyncProducer {
    public static void main(String[] args) throws Exception {
        DefaultMQProducer producer = new DefaultMQProducer("HAOKE_IM");
        producer.setNamesrvAddr("172.16.55.185:9876");
        // 发送失败的重试次数
        producer.setRetryTimesWhenSendAsyncFailed(0);

        producer.start();

        String msgStr = "用户A发送消息给用户B";
        Message msg = new Message("haoke_im_topic","SEND_MSG",
                msgStr.getBytes(RemotingHelper.DEFAULT_CHARSET));

        // 异步发送消息
        producer.send(msg, new SendCallback() {
            @Override
            public void onSuccess(SendResult sendResult) {
                System.out.println("消息状态：" + sendResult.getSendStatus());
                System.out.println("消息id：" + sendResult.getMsgId());
                System.out.println("消息queue：" + sendResult.getMessageQueue());
                System.out.println("消息offset：" + sendResult.getQueueOffset());
            }

            @Override
            public void onException(Throwable e) {
                System.out.println("发送失败！" + e);
            }
        });

        System.out.println("发送成功!");

//        producer.shutdown();
    }
}

```

> 注意： producer.shutdown()要注释掉，否则发送失败。原因是，异步发送，还未来得及发送就被关闭了。

2.4、消费消息

```
package cn.itcast.rocketmq;

```

```
import org.apache.rocketmq.client.consumer.DefaultMQPushConsumer;
import org.apache.rocketmq.client.consumer.listener.ConsumeConcurrentlyContext;
import org.apache.rocketmq.client.consumer.listener.ConsumeConcurrentlyStatus;
import org.apache.rocketmq.client.consumer.listener.MessageListenerConcurrently;
import org.apache.rocketmq.client.exception.MQClientException;
import org.apache.rocketmq.common.message.MessageExt;

import java.io.UnsupportedEncodingException;
import java.util.List;

public class ConsumerDemo {

    public static void main(String[] args) throws Exception {
        DefaultMQPushConsumer consumer = new DefaultMQPushConsumer("HAOKE_IM");
        consumer.setNamesrvAddr("172.16.55.185:9876");

        // 订阅topic，接收此Topic下的所有消息
        consumer.subscribe("haoke_im_topic", "*");

        consumer.registerMessageListener(new MessageListenerConcurrently() {
            @Override
            public ConsumeConcurrentlyStatus consumeMessage(List<MessageExt> msgs,
ConsumeConcurrentlyContext context) {

                for (MessageExt msg : msgs) {
                    try {
                        System.out.println(new String(msg.getBody(), "UTF-8"));
                    } catch (UnsupportedEncodingException e) {
                        e.printStackTrace();
                    }
                }
                System.out.println("收到消息->" + msgs);

                return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
            }
        });

        consumer.start();


    }
}

```

测试：

```
用户A发送消息给用户B
收到消息->[MessageExt [queueId=7, storeSize=200, queueOffset=1, sysFlag=0,
bornTimestamp=1544521864503, bornHost=/172.16.55.160:3460,
storeTimestamp=1544521864456, storeHost=/172.16.55.185:10911,
msgId=AC1037B900002A9F0000000000011F58, commitLogOffset=73560, bodyCRC=203638610,
reconsumeTimes=0, preparedTransactionOffset=0,
toString()=Message{topic='haoke_im_topic', flag=0, properties={MIN_OFFSET=0,
MAX_OFFSET=2, CONSUME_START_TIME=1544521864541,
UNIQ_KEY=AC1037A02F5018B4AAC2375431360000, WAIT=true, TAGS=SEND_MSG}, body=[-25,
-108, -88, -26, -120, -73, 65, -27, -113, -111, -23, -128, -127, -26, -74, -120, -26,
-127, -81, -25, -69, -103, -25, -108, -88, -26, -120, -73, 66],
transactionId='null'}]]

```

其它订阅方式：

```
//完整匹配
consumer.subscribe("haoke_im_topic", "SEND_MSG");

//或匹配
consumer.subscribe("haoke_im_topic", "SEND_MSG || SEND_MSG1");
```

2.5、消息过滤器

RocketMQ支持根据用户自定义属性进行过滤，过滤表达式类似于SQL的where，如：a> 5 AND b ='abc'

发送消息：

```
package cn.itcast.rocketmq;

import org.apache.rocketmq.client.producer.DefaultMQProducer;
import org.apache.rocketmq.client.producer.SendResult;
import org.apache.rocketmq.common.message.Message;
import org.apache.rocketmq.remoting.common.RemotingHelper;

public class SyncProducerFilter {
    public static void main(String[] args) throws Exception {
        DefaultMQProducer producer = new DefaultMQProducer("HAOKE_IM");
        producer.setNamesrvAddr("172.16.55.185:9876");
        producer.start();

        String msgStr = "美女001";
        Message msg = new Message("haoke_meinv_topic","SEND_MSG",
                msgStr.getBytes(RemotingHelper.DEFAULT_CHARSET));
        msg.putUserProperty("age", "18");
        msg.putUserProperty("sex", "女");

        // 发送消息
        SendResult sendResult = producer.send(msg);

        System.out.println("消息状态：" + sendResult.getSendStatus());
        System.out.println("消息id：" + sendResult.getMsgId());
```

```
        System.out.println("消息queue：" + sendResult.getMessageQueue());
        System.out.println("消息offset：" + sendResult.getQueueOffset());

        System.out.println(sendResult);

        producer.shutdown();
    }
}

```

接收消息：

```
package cn.itcast.rocketmq;

import org.apache.rocketmq.client.consumer.DefaultMQPushConsumer;
import org.apache.rocketmq.client.consumer.MessageSelector;
import org.apache.rocketmq.client.consumer.listener.ConsumeConcurrentlyContext;
import org.apache.rocketmq.client.consumer.listener.ConsumeConcurrentlyStatus;
import org.apache.rocketmq.client.consumer.listener.MessageListenerConcurrently;
import org.apache.rocketmq.common.message.MessageExt;

import java.io.UnsupportedEncodingException;
import java.util.List;

public class ConsumerFilterDemo {

    public static void main(String[] args) throws Exception {
        DefaultMQPushConsumer consumer = new DefaultMQPushConsumer("HAOKE_IM");
        consumer.setNamesrvAddr("172.16.55.185:9876");

        // 订阅topic，接收此Topic下的所有消息
        consumer.subscribe("haoke_meinv_topic", MessageSelector.bySql("age>=20 AND sex='女'"));

        consumer.registerMessageListener(new MessageListenerConcurrently() {
            @Override
            public ConsumeConcurrentlyStatus consumeMessage(List<MessageExt> msgs, ConsumeConcurrentlyContext context) {

                for (MessageExt msg : msgs) {
                    try {
                        System.out.println(new String(msg.getBody(), "UTF-8"));
                    } catch (UnsupportedEncodingException e) {
                        e.printStackTrace();
                    }
                }
                System.out.println("收到消息->" + msgs);

                return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
            }
        });

        consumer.start();
```

```
40
41
42      }
43  }
```

测试，报错：

```
1  Exception in thread "main" org.apache.rocketmq.client.exception.MQClientException:
   CODE: 1  DESC: The broker does not support consumer to filter message by SQL92
2  For more information, please visit the url, http://rocketmq.apache.org/docs/faq/
3      at
   org.apache.rocketmq.client.impl.MQClientAPIImpl.checkClientInBroker(MQClientAPIImpl.j
   ava:2089)
4      at
   org.apache.rocketmq.client.impl.factory.MQClientInstance.checkClientInBroker(MQClient
   Instance.java:432)
5      at
   org.apache.rocketmq.client.impl.consumer.DefaultMQPushConsumerImpl.start(DefaultMQPus
   hConsumerImpl.java:633)
6      at
   org.apache.rocketmq.client.consumer.DefaultMQPushConsumer.start(DefaultMQPushConsumer
   .java:520)
7      at cn.itcast.rocketmq.ConsumerFilterDemo.main(ConsumerFilterDemo.java:39)
```

原因是默认配置下，不支持自定义属性，需要设置开启：

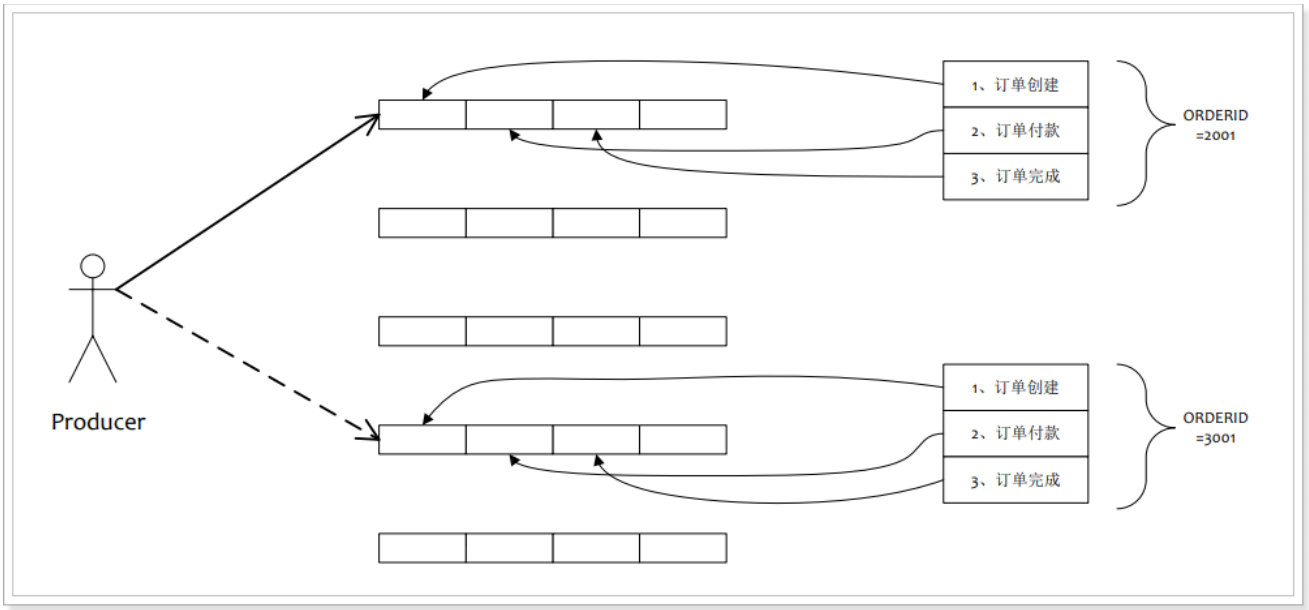
```
1  #加入到broker的配置文件中
2  enablePropertyFilter=true
```

重启broker进行测试，发现已经可以正常发送、接收消息了。

3、producer详解

3.1、顺序消息

在某些业务中，consumer在消费消息时，是需要按照生产者发送消息的顺序进行消费的，比如在电商系统中，订单的消息，会有创建订单、订单支付、订单完成，如果消息的顺序发生改变，那么这样的消息就没有意义了。



3.1.1、生产者

```
package cn.itcast.rocketmq.order;

import org.apache.rocketmq.client.producer.DefaultMQProducer;
import org.apache.rocketmq.client.producer.SendResult;
import org.apache.rocketmq.common.message.Message;
import org.apache.rocketmq.remoting.common.RemotingHelper;

public class OrderProducer {

    public static void main(String[] args) throws Exception{
        DefaultMQProducer producer = new DefaultMQProducer("HAOKE_ORDER_PRODUCER");
        producer.setNamesrvAddr("172.16.55.185:9876");
        producer.start();

        for (int i = 0; i < 100; i++) {
            String msgStr = "order --> " + i;
            int orderId = i % 10; // 模拟生成订单id
            Message message = new Message("haoke_order_topic","ORDER_MSG",
                    msgStr.getBytes(RemotingHelper.DEFAULT_CHARSET));

            SendResult sendResult = producer.send(message, (mqs, msg, arg) -> {
                Integer id = (Integer) arg;
                int index = id % mqs.size();
                return mqs.get(index);
            }, orderId);

            System.out.println(sendResult);
        }

        producer.shutdown();
    }
}

```

3.1.2、消费者

```
package cn.itcast.rocketmq.order;

import org.apache.rocketmq.client.consumer.DefaultMQPushConsumer;
import org.apache.rocketmq.client.consumer.listener.ConsumeOrderlyContext;
import org.apache.rocketmq.client.consumer.listener.ConsumeOrderlyStatus;
import org.apache.rocketmq.client.consumer.listener.MessageListenerOrderly;
import org.apache.rocketmq.common.message.MessageExt;

import java.util.List;

public class OrderConsumer {

    public static void main(String[] args) throws Exception{
        DefaultMQPushConsumer consumer = new
DefaultMQPushConsumer("HAOKE_ORDER_CONSUMER");
        consumer.setNamesrvAddr("172.16.55.185:9876");
        consumer.subscribe("haoke_order_topic", "*");

        consumer.registerMessageListener(new MessageListenerOrderly() {
            @Override
            public ConsumeOrderlyStatus consumeMessage(List<MessageExt> msgs,
ConsumeOrderlyContext context) {

                System.out.println(Thread.currentThread().getName() + " Receive New
Messages: " + msgs);

                return ConsumeOrderlyStatus.SUCCESS;
            }
        });

        consumer.start();
    }
}

```

3.1.3、测试

测试结果：相同订单id的消息会落到同一个queue中，一个消费者线程会顺序消费queue，从而实现顺序消费消息。

3.2、分布式事务消息

3.2.1、回顾什么事务

聊什么是事务，最经典的例子就是转账操作，用户A转账给用户B1000元的过程如下：

- 用户A发起转账请求，用户A账户减去1000元
- 用户B的账户增加1000元

如果，用户A账户减去1000元后，出现了故障（如网络故障），那么需要将该操作回滚，用户A账户增加1000元。

这就是事务。

3.2.2、分布式事务

随着项目越来越复杂，越来越服务化，就会导致系统间的事务问题，这个就是分布式事务问题。

分布式事务分类有这几种：

- 基于单个JVM，数据库分库分表了（跨多个数据库）。
- 基于多JVM，服务拆分了（不跨数据库）。
- 基于多JVM，服务拆分了 并且数据库分库分表了。

解决分布式事务问题的方案有很多，使用消息实现只是其中的一种。

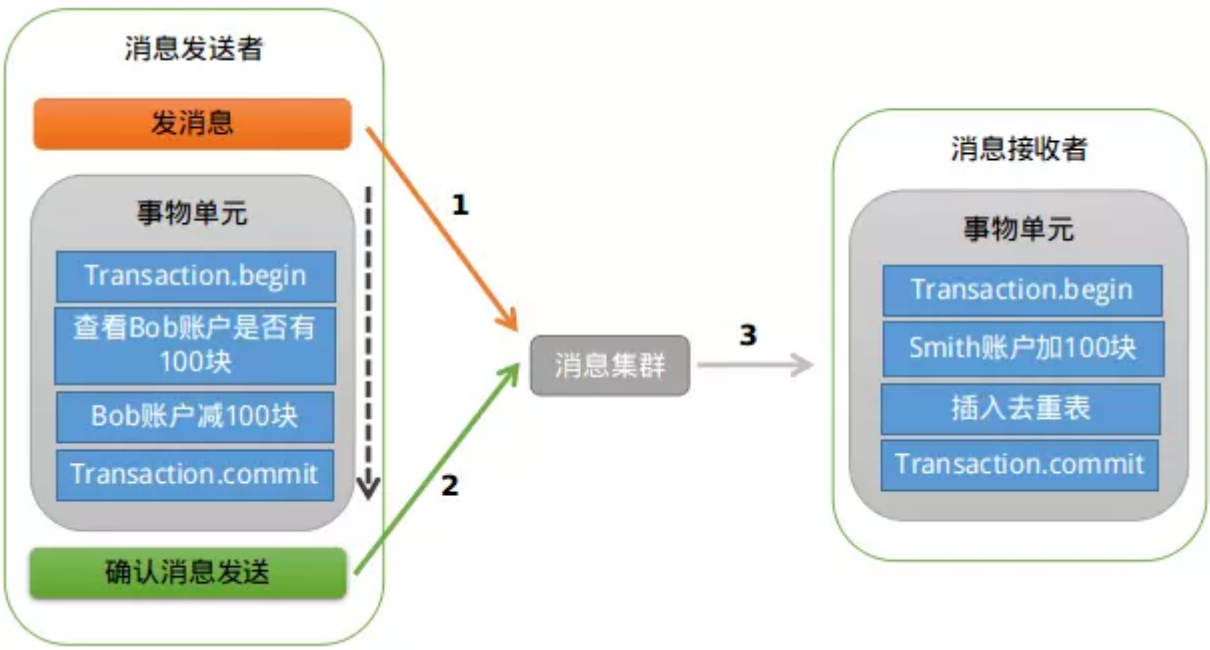
3.2.3、原理

> Half(Prepare) Message

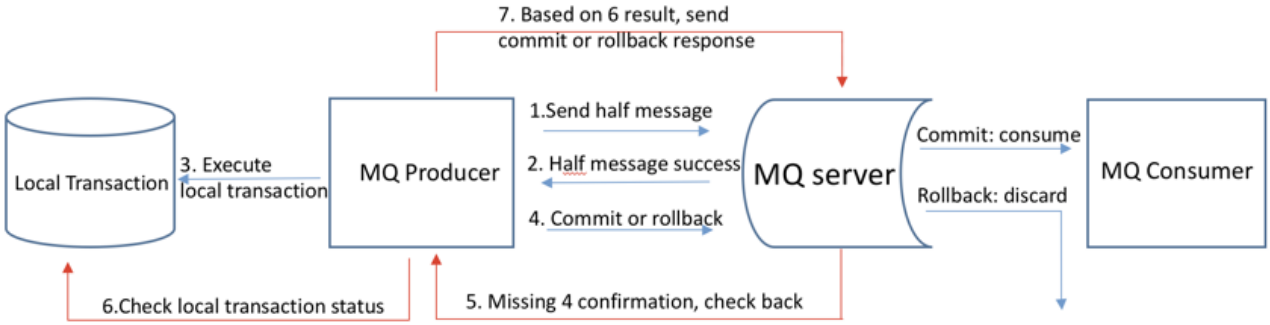
指的是暂不能投递的消息，发送方已经将消息成功发送到了 MQ 服务端，但是服务端未收到生产者对该消息的二次确认，此时该消息被标记成“暂不能投递”状态，处于该种状态下的消息即半消息。

> Message Status Check

由于网络闪断、生产者应用重启等原因，导致某条事务消息的二次确认丢失，MQ 服务端通过扫描发现某条消息长期处于“半消息”时，需要主动向消息生产者询问该消息的最终状态（Commit 或是 Rollback），该过程即消息回查。



3.3.4、执行流程



1. 发送方向 MQ 服务端发送消息。
2. MQ Server 将消息持久化成功之后，向发送方 ACK 确认消息已经发送成功，此时消息为半消息。
3. 发送方开始执行本地事务逻辑。
4. 发送方根据本地事务执行结果向 MQ Server 提交二次确认（Commit 或是 Rollback），MQ Server 收到 Commit 状态则将半消息标记为可投递，订阅方最终将收到该消息；MQ Server 收到 Rollback 状态则删除半消息，订阅方将不会接受该消息。
5. 在断网或者是应用重启的特殊情况下，上述步骤4提交的二次确认最终未到达 MQ Server，经过固定时间后 MQ Server 将对该消息发起消息回查。
6. 发送方收到消息回查后，需要检查对应消息的本地事务执行的最终结果。
7. 发送方根据检查得到的本地事务的最终状态再次提交二次确认，MQ Server 仍按照步骤4对半消息进行操作。

3.3.5、生产者

```
package cn.itcast.rocketmq.transaction;

import org.apache.rocketmq.client.producer.TransactionMQProducer;
import org.apache.rocketmq.common.message.Message;

public class TransactionProducer {

    public static void main(String[] args) throws Exception {
        TransactionMQProducer producer = new
TransactionMQProducer("transaction_producer");
        producer.setNamesrvAddr("172.16.55.185:9876");

        // 设置事务监听器
        producer.setTransactionListener(new TransactionListenerImpl());

        producer.start();

        // 发送消息
        Message message = new Message("pay_topic", "用户A给用户B转账500
元".getBytes("UTF-8"));
        producer.sendMessageInTransaction(message, null);

        Thread.sleep(999999);
        producer.shutdown();

    }
```

```
}

```

注意：发送消息使用的是TransactionMQProducer

3.3.6、本地事务处理

```
package cn.itcast.rocketmq.transaction;

import org.apache.rocketmq.client.producer.LocalTransactionState;
import org.apache.rocketmq.client.producer.TransactionListener;
import org.apache.rocketmq.common.message.Message;
import org.apache.rocketmq.common.message.MessageExt;

import java.util.HashMap;
import java.util.Map;

public class TransactionListenerImpl implements TransactionListener {

    private static Map<String, LocalTransactionState> STATE_MAP = new HashMap<>();

    /**
     * 执行具体的业务逻辑
     *
     * @param msg 发送的消息对象
     * @param arg
     * @return
     */
    @Override
    public LocalTransactionState executeLocalTransaction(Message msg, Object arg) {

        try {
            System.out.println("用户A账户减500元.");
            Thread.sleep(500); //模拟调用服务

//            System.out.println(1/0);

            System.out.println("用户B账户加500元.");
            Thread.sleep(800);

            STATE_MAP.put(msg.getTransactionId(), LocalTransactionState.COMMIT_MESSAGE);

            // 二次提交确认
            return LocalTransactionState.COMMIT_MESSAGE;
        } catch (Exception e) {
            e.printStackTrace();
        }

        STATE_MAP.put(msg.getTransactionId(), LocalTransactionState.ROLLBACK_MESSAGE);
        // 回滚
        return LocalTransactionState.ROLLBACK_MESSAGE;
```

```
    }

    /**
     * 消息回查
     *
     * @param msg
     * @return
     */
    @Override
    public LocalTransactionState checkLocalTransaction(MessageExt msg) {
        return STATE_MAP.get(msg.getTransactionId());
    }
}

```

3.3.7、消费者

```
package cn.itcast.rocketmq.transaction;

import org.apache.rocketmq.client.consumer.DefaultMQPushConsumer;
import org.apache.rocketmq.client.consumer.listener.ConsumeConcurrentlyContext;
import org.apache.rocketmq.client.consumer.listener.ConsumeConcurrentlyStatus;
import org.apache.rocketmq.client.consumer.listener.MessageListenerConcurrently;
import org.apache.rocketmq.common.consumer.ConsumeFromWhere;
import org.apache.rocketmq.common.message.MessageExt;

import java.io.UnsupportedEncodingException;
import java.util.List;

public class TransactionConsumer {

    public static void main(String[] args) throws Exception {
        DefaultMQPushConsumer consumer = new
DefaultMQPushConsumer("HAOKE_CONSUMER");
        consumer.setNamesrvAddr("172.16.55.185:9876");

        // 订阅topic，接收此Topic下的所有消息
        consumer.subscribe("pay_topic", "*");

        consumer.registerMessageListener(new MessageListenerConcurrently() {
            @Override
            public ConsumeConcurrentlyStatus consumeMessage(List<MessageExt> msgs,
ConsumeConcurrentlyContext context) {

                for (MessageExt msg : msgs) {
                    try {
                        System.out.println(new String(msg.getBody(), "UTF-8"));
                    } catch (UnsupportedEncodingException e) {
                        e.printStackTrace();
                    }
                }

                return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
```

```
            }
        });

        consumer.start();


    }

}

```

3.3.8、测试

测试结果：返回commit状态时，消费者能够接收到消息，返回rollback状态时，消费者接受不到消息。

4、consumer详解

4.1、push和pull模式

在RocketMQ中，消费者有两种模式，一种是push模式，另一种是pull模式。

push模式：客户端与服务端建立连接后，当服务端有消息时，将消息推送到客户端。

pull模式：客户端不断的轮询请求服务端，来获取新的消息。

但在具体实现时，Push和Pull模式都是采用消费端主动拉取的方式，即consumer轮询从broker拉取消息。

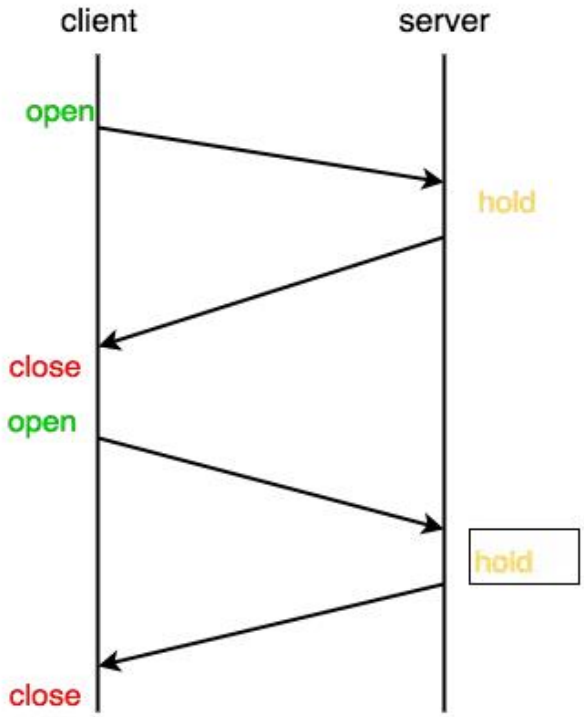
> 区别：
>
> Push方式里，consumer把轮询过程封装了，并注册MessageListener监听器，取到消息后，唤醒MessageListener的consumeMessage()来消费，对用户而言，感觉消息是被推送过来的。
>
> Pull方式里，取消息的过程需要用户自己写，首先通过打算消费的Topic拿到MessageQueue的集合，遍历MessageQueue集合，然后针对每个MessageQueue批量取消息，一次取完后，记录该队列下一次要取的开始offset，直到取完了，再换另一个MessageQueue。

疑问：既然是采用pull方式实现，RocketMQ如何保证消息的实时性呢？

4.1.1、长轮询

RocketMQ中采用了长轮询的方式实现，什么是长轮询呢？

> 长轮询即是在请求的过程中，若是服务器端数据并没有更新，那么则将这个连接挂起，直到服务器推送新的数据，再返回，然后进入循环周期。
>
> 客户端像传统轮询一样从服务端请求数据，服务端会阻塞请求不会立刻返回，直到有数据或超时才返回给客户端，然后关闭连接，客户端处理完响应信息后再向服务器发送新的请求。



(轮询示意图)

4.2、消息模式

DefaultMQPushConsumer实现了自动保存offset值以及实现多个consumer的负载均衡。

```
//设置组名
DefaultMQPushConsumer consumer = new DefaultMQPushConsumer("HAOKE_IM");
```

通过groupname将多个consumer组合在一起，那么就会存在一个问题，消息发送到这个组后，消息怎么分配呢？

这个时候，就需要指定消息模式，分别有集群和广播模式。

- 集群模式
  - 同一个 ConsumerGroup(GroupName相同) 里的每 个 Consumer 只消费所订阅消息的一部分内容， 同一个 ConsumerGroup 里所有的 Consumer消费的内容合起来才是所订阅 Topic 内容的整体， 从而达到负载均衡的目的 。
- 广播模式
  - 同一个 ConsumerGroup里的每个 Consumer都 能消费到所订阅 Topic 的全部消息，也就是一个消息会被多次分发，被多个 Consumer消费。

```
// 集群模式
consumer.setMessageModel(MessageModel.CLUSTERING);
// 广播模式
consumer.setMessageModel(MessageModel.BROADCASTING);
```

4.3、重复消息的解决方案

造成消息重复的根本原因是：网络不可达。只要通过网络交换数据，就无法避免这个问题。所以解决这个问题的办法就是绕过这个问题。那么问题就变成了：如果消费端收到两条一样的消息，应该怎样处理？

1. 消费端处理消息的业务逻辑保持幂等性
2. 保证每条消息都有唯一编号且保证消息处理成功与去重表的日志同时出现

第1条很好理解，只要保持幂等性，不管来多少条重复消息，最后处理的结果都一样。第2条原理就是利用一张日志表来记录已经处理成功的消息的ID，如果新到的消息ID已经在日志表中，那么就不再处理这条消息。

第1条解决方案，很明显应该在消费端实现，不属于消息系统要实现的功能。第2条可以消息系统实现，也可以业务端实现。正常情况下出现重复消息的概率其实很小，如果由消息系统来实现的话，肯定会对消息系统的吞吐量和高可用有影响，所以最好还是由业务端自己处理消息重复的问题，这也是RocketMQ不解决消息重复的问题的原因。

RocketMQ不保证消息不重复，如果你的业务需要保证严格的不重复消息，需要你自己在业务端去重。

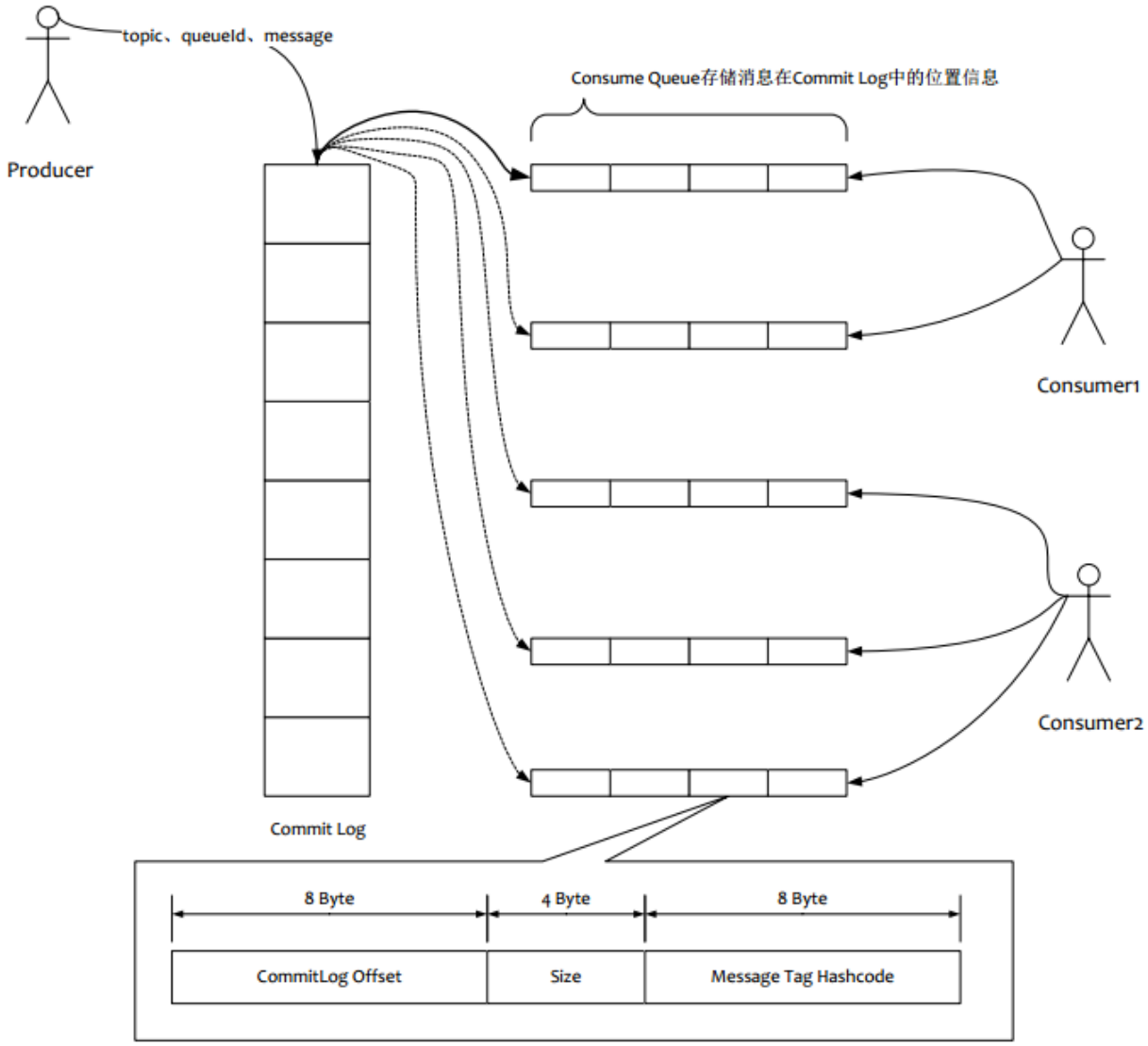
5、RocketMQ存储

RocketMQ中的消息数据存储，采用了零拷贝技术（使用 mmap + write 方式），文件系统采用 Linux Ext4 文件系统进行存储。

5.1、消息数据的存储

在RocketMQ中，消息数据是保存在磁盘文件中，为了保证写入的性能，RocketMQ尽可能保证顺序写入，顺序写入的效率比随机写入的效率高很多。

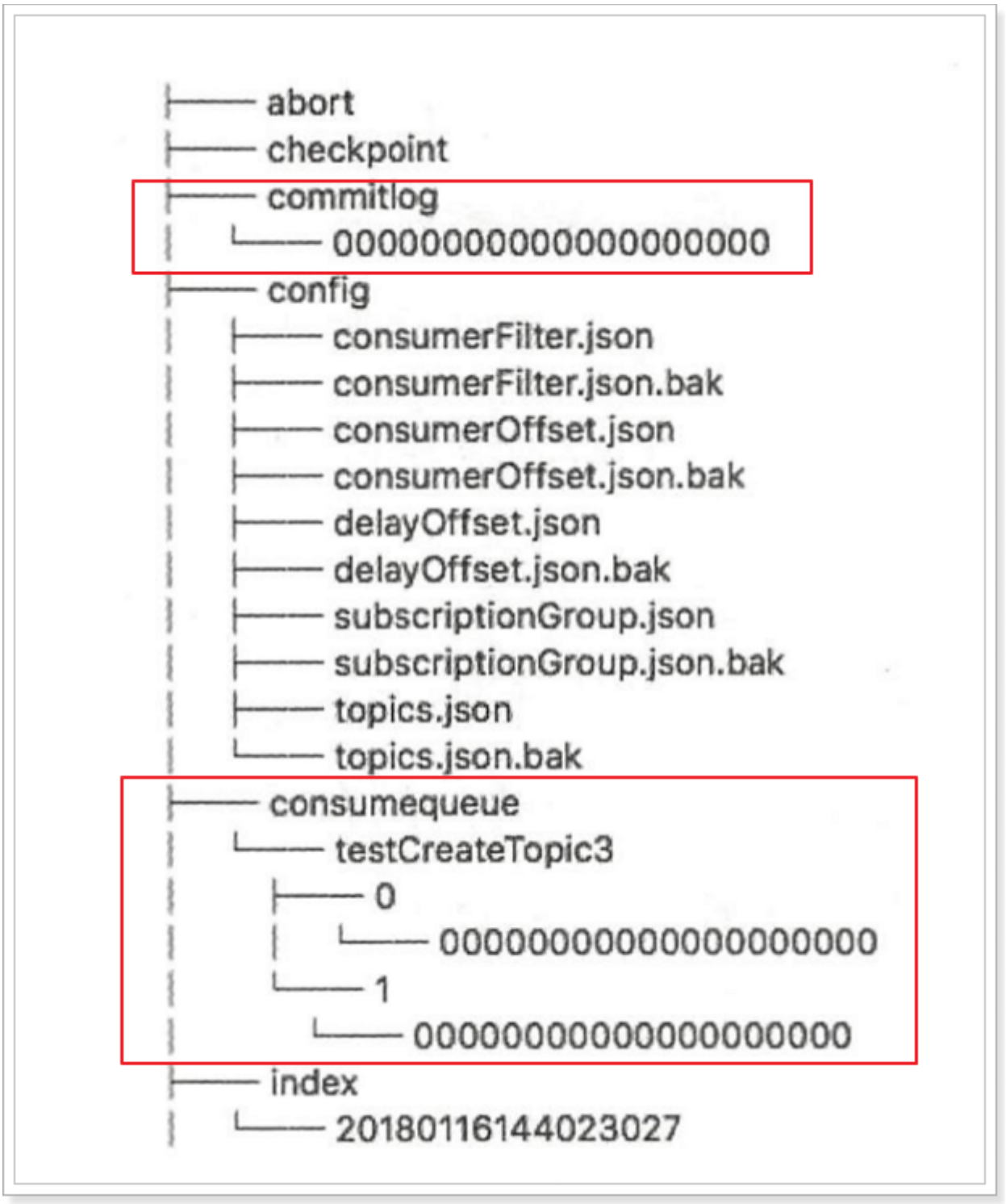
RocketMQ消息的存储是由ConsumeQueue和CommitLog配合完成的，CommitLog是真正存储数据的文件，ConsumeQueue是索引文件，存储数据指向到物理文件的配置。



如上图所示：

- 消息主体以及元数据都存储在CommitLog当中
- Consume Queue相当于kafka中的partition，是一个逻辑队列，存储了这个Queue在CommiLog中的起始offset，log大小和MessageTag的hashCode。
- 每次读取消息队列先读取consumerQueue,然后再通过consumerQueue去commitLog中拿到消息主体。

文件位置：



5.2、同步刷盘与异步刷盘

RocketMQ 为了提高性能，会尽可能地保证 磁盘的顺序写。消息在通过 Producer 写入 RocketMQ 的时候，有两种写磁盘方式，分别是同步刷盘与异步刷盘。

- 同步刷盘
  - 在返回写成功状态时，消息已经被写入磁盘 。
  - 具体流程是：消息写入内存的 PAGECACHE 后，立刻通知刷盘线程刷盘，然后等待刷盘完成，刷盘线程执行完成后唤醒等待的线程，返回消息写成功的状态 。
- 异步刷盘

- 在返回写成功状态时，消息可能只是被写入了内存的 PAGECACHE，写操作的返回快，吞吐量大
- 当内存里的消息量积累到一定程度时，统一触发写磁盘动作，快速写入。

- broker配置文件中指定刷盘方式
  - flushDiskType=ASYNC_FLUSH -- 异步
  - flushDiskType=SYNC_FLUSH -- 同步