图文 116 我们系统中的Consumer作为消费者是如何创建出来的?

我们系统中的Consumer作为消费者是如何创建出来的?

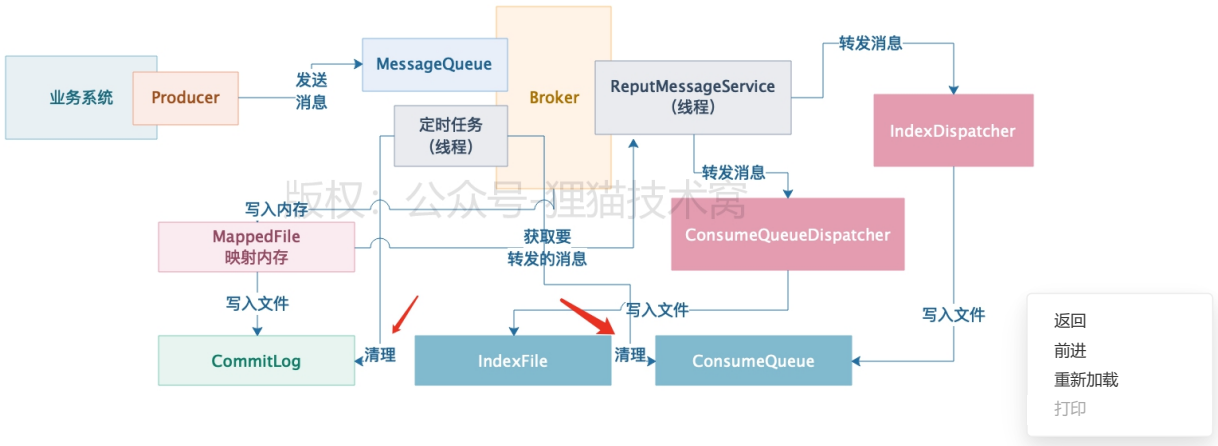
继《从零开始带你成为JVM实战高手》后，阿里资深技术专家携新作再度出山，重磅推荐：

（点击下方蓝字试听）

《从零开始带你成为MySQL实战优化高手》

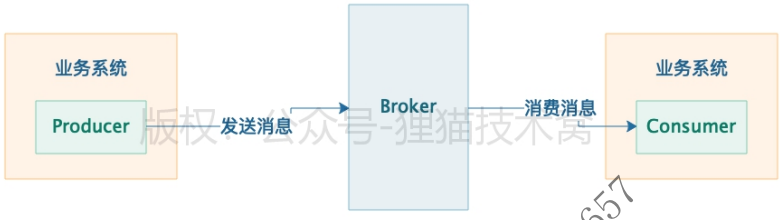
之前我们已经讲完了RocketMQ的Broker这块的一些源码和原理，源码没讲的太细，因为源码量实在是太多了，所以我们只能讲一些重点的片段

但是起码我们现在已经知道了，我们平时把消息写入到Broker去，他会把消息写入到CommitLog、ConsumeQueue、IndexFile里去，如下图。



那么现在Broker上有了数据了，接着当然是某个业务系统里会启动一个Consumer，指定自己要消费哪个Topic的数据

接着Consumer就会从指定的Topic上消费数据过来了，然后消息交给你的业务代码来处理，如下图。



那么这次我们来看看这个业务系统里的Cosumer是如何创建和启动的呢？

其实我们平时创建的一般都是DefaultMQPushConsumerImpl，然后会调用他的start()方法来启动他，那么今天我们就来看看启动Consumer的时候都会干什么。

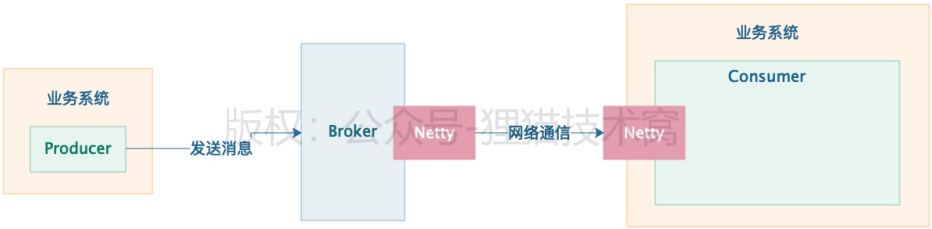
首先在启动的时候，会看到如下一行源码片段：

```
this.mQClientFactory = MQClientManager.getInstance()
        .getOrCreateMQClientInstance(this.defaultMQPushConsumer, this.rpcHook);
```

不知道大家对这个MQClientFactory有没有什么感觉？

说实话，你可以想一下，这个Consumer一旦启动，必然是要跟Broker去建立长连接的，底层绝对也是基于Netty去做的，建立长连接之后，才能不停的通信拉取消息

所以这个MQClientFactory底层直觉上就应该封装了Netty网络通信的东西，如下图所示。



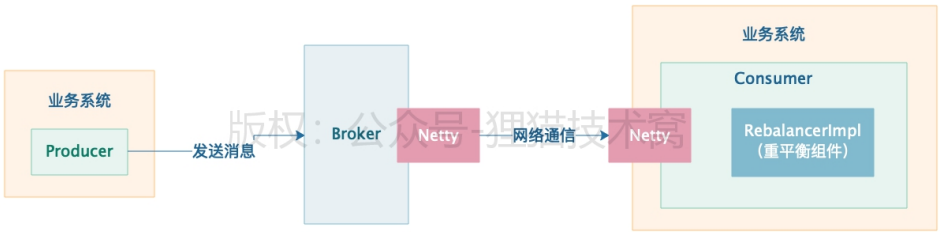
接着我们会看到如下的一些源码片段。

```
this.rebalanceImpl.setConsumerGroup(
        this.defaultMQPushConsumer.getConsumerGroup());
this.rebalanceImpl.setMessageModel(
        this.defaultMQPushConsumer.getMessageModel());
this.rebalanceImpl.setAllocateMessageQueueStrategy(
        this.defaultMQPushConsumer.getAllocateMessageQueueStrategy());
this.rebalanceImpl.setmQClientFactory(this.mQClientFactory);
```

大家看到上述源码有什么感触，是不是发现似乎在搞一个叫做RebalanceImpl的东西，还给他设置了Consumer分组，还有MQClientFactory在里面

那么这个东西，其实大家一看名字就应该知道了，他就是专门负责Consumer重平衡的。

假设你的ConsumerGroup里加入了一个新的Consumer，那么就会重新分配每个Consumer消费的MessageQueue，如果ConsumerGroup里某个Consumer宕机了，也会重新分配MessageQueue，这就是所谓的重平衡，如下图。

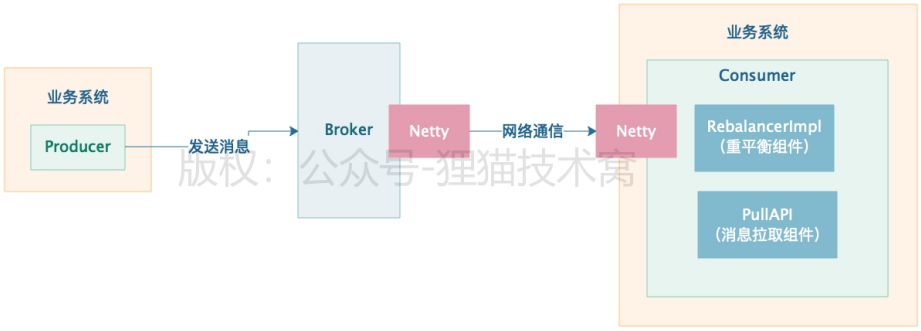


接着我们看如下源码片段。

```
this.pullAPIWrapper = new PullAPIWrapper(
   mQClientFactory,
   this.defaultMQPushConsumer.getConsumerGroup(), isUnitMode());
this.pullAPIWrapper.registerFilterMessageHook(filterMessageHookList);
```


这个PullAPIWrapper大家觉得是什么呢？看起来是不是很像是专门用来拉取消息的API组件？

对的，其实这个一看就是用来拉取消息的，如下图。

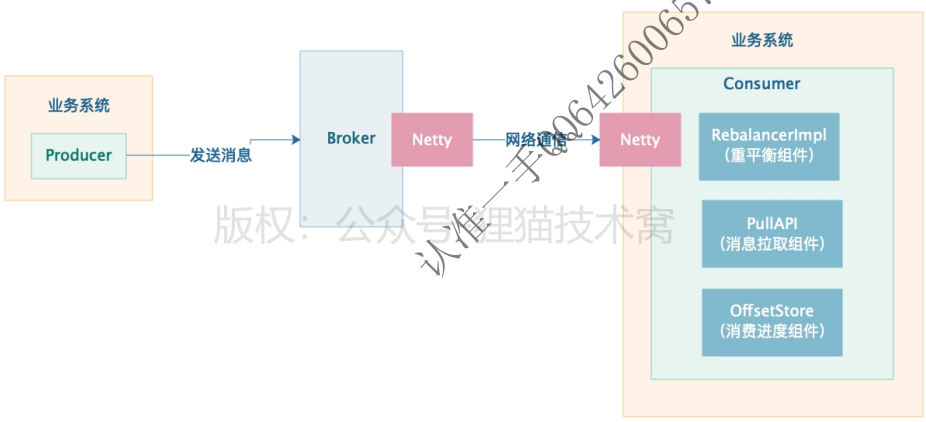


接着大家看如下的源码片段。

```
if (this.defaultMQPushConsumer.getOffsetStore() != null) {
    this.offsetStore =
        this.defaultMQPushConsumer.getOffsetStore();
} else {
    switch (this.defaultMQPushConsumer.getMessageModel()) {
        case BROADCASTING:
            this.offsetStore =
                new LocalFileOffsetStore(
                    this.mQClientFactory,
                    this.defaultMQPushConsumer.getConsumerGroup());
            break;
        case CLUSTERING:
            this.offsetStore =
                new RemoteBrokerOffsetStore(
                    this.mQClientFactory,
                    this.defaultMQPushConsumer.getConsumerGroup());
            break;
        default:
            break;
    }
    this.defaultMQPushConsumer.setOffsetStore(this.offsetStore);
}
this.offsetStore.load();
```


有没有发现他在弄一个叫做OffsetStore的东西呢?

这个东西一看，顾名思义，就是用来存储和管理Consumer消费进度offset的一个组件，如下图。



接下来源码里还有一些东西，其实都不是太核心的了，最核心的无非就是这三个组件，首先Consumer刚启动，必须依托Rebalancer组件，去进行一下重平衡，自己要分配一些MessageQueue去拉取消息。

接着拉取消息，必须要依托PullAPI组件通过底层网络通信去拉取。在拉取的过程中，必然要维护offset消费进度，此时就需要OffsetStore组件。万一要是ConsumerGroup里多了Consumer或者少了Consumer，又要依托Rebalancer组件进行重平衡了。

基本就是这样一个思路，下一次我们继续分析，接下来几讲我们分析完Consumer的一些源码实现，那么对RocketMQ的核心源码的一些思路，我们就理解的差不多了。

End

狸猫技术窝精品专栏及课程推荐：

- 《从零开始带你成为JVM实战高手》

- 《21天互联网Java进阶面试训练营》（分布式篇）
- 《互联网Java工程师面试突击》（第1季）
- 《互联网Java工程师面试突击》（第3季）

重要说明：

- 如何提问：每篇文章都有评论区，大家可以尽情留言提问，我会逐一答疑
- 如何加群：购买狸猫技术窝专栏的小伙伴都可以加入狸猫技术交流群，一个非常纯粹的技术交流的地方

  具体加群方式，请参见目录菜单下的文档：《付费用户如何加群》（**购买后可见**）