课程目标

1、掌握 Kafka 的基本架构原理

2、掌握 Kafka 基本使用；

3、掌握 Kafka 的高级特性；

4、Kafka 和 RabbitMQ 对比分析

1. Kafka 基本介绍

http://kafka.apache.org/documentation.html#introduction

https://github.com/apache/kafka

（中文翻译）https://kafka.apachecn.org/documentation.html

你会直接看官网学习吗？比如 MySQL 的官网，从上往下一页一页地看？这种扫资料的方式效率是低下的，而且有很多是你不需要的内容。建议先翻几本书的目录，建立自己的知识结构之后，再逐个击破。官网只用来参考。

1.1.Kafka 发展历史

第一个问题，Kafka 是一个普通的消息队列吗？它是怎么诞生的呢？是不是跟 TIB 一样主要用在金融领域？

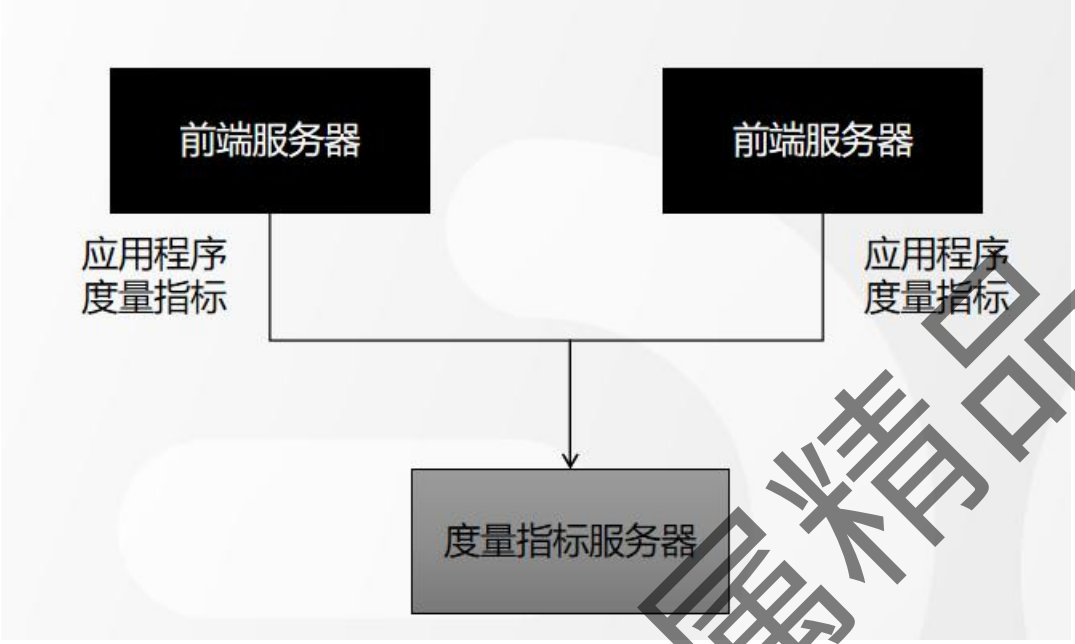
（产品的诞生背景决定了它的特性，特性决定使用场景）

有同学可能注册了 LinkedIn 的账号（国内叫领英）？在芝麻分里面一项就是领英的职业信息，说明它的影响力还是比较大的。

LinkedIn 2003 年在美国创立，号称全球最大的职业社交网站，18 年的时候用户就超过 5.6 亿了，网站所承受的流量也是世界顶级的。

作为一家社交企业，LinkedIn 有非常多的 IT 系统，每天要收集很多实时生成的数

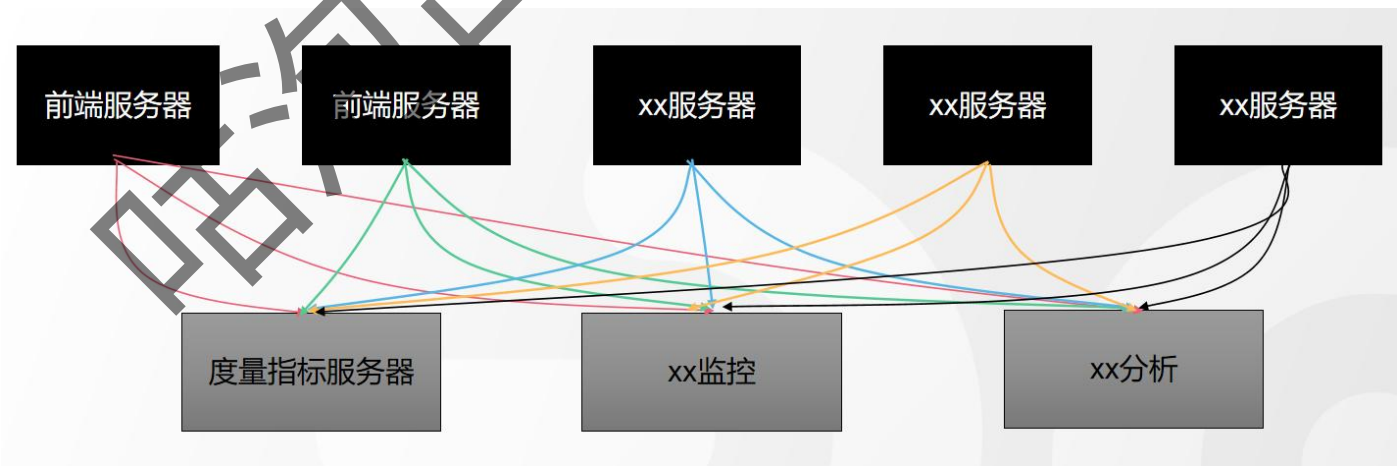
据，比如用户活跃度、用户的页面访问情况，用户搜索内容等等，这些数据被很多系统用到。



最开始的模型

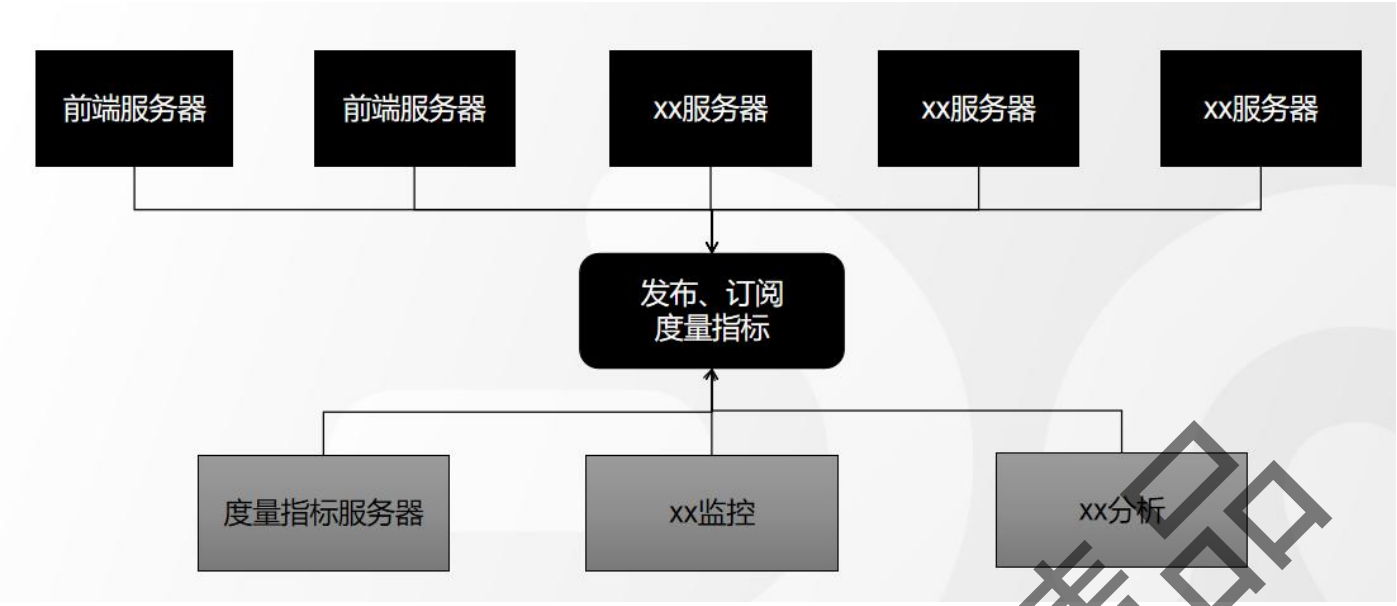
最开始这些数据的传输都是点对点的传输，但是随着应用程序的增多，指标的细分，数据量不断增长，这种方式的效率也越来越差（参考：2019 年 10 月，每天处理的消息是 7 万亿条）。

https://engineering.linkedin.com/blog/2019/apache-kafka-trillion-messages



稍微复杂一点的场景

所以 LinkedIn 设计了一个集中式的数据通道，大家统一通过这个数据通道来交互数据。



最终解决方案

最开始这个数据通道用 ActiveMQ 来实现，但是经常会出现消息阻塞和服务不可用的情况（开始想的是何必重复发明轮子，后来发现这个轮子原来是方的）。所以 Linkedin 自己开发了一个数据引擎，就是 Kafka。

（业务会推动技术的发展，淘宝阿里也是这样。业务越复杂，挑战越大，技术难度越大，技术能力越强。没有业务的驱动，一个公司成不了牛逼的公司）

从这个故事看来，Kafka 解决的是生产环境中数据上下游的耦合问题，所以很多时候不仅仅把它叫做消息中间件，还可以叫做消息引擎，或者分布式实时流处理平台。

那为什么要叫 Kafka 呢?

大家在高中的时候应该都学过一篇课文叫《变形记》，作者叫 Kafka，其实这个正是这个消息引擎的名字的来源。因为 Kafka 的架构师也很喜欢作家卡夫卡。

Kafka 由 Scala 和 Java 写成，Scala 运行在 Java 虚拟机上，所以需要 JDK 环境。

跟 RabbitMQ 一样，Kafka 对主流编程语言都有客户端的支持。

Kafka 2010 年开源，11 年捐献给 Apache。开源以后还产生了很多衍生的版本（包括现在云厂商卖的 Kafka 服务），一般我们说的 Kafka 都是 Apache Kafka。

https://cloud.tencent.com/developer/news/372069

1.2.Kafka 使用场景

http://kafka.apache.org/documentation/ 官网的 documentation 页面列举了一些应用场景，总体上可以分成三类：

首先当然是作为普通的消息队列使用。

1.2.1.消息传递 Messaging

消息传递就是发送数据，作为 TCP HTTP 或者 RPC 的替代方案，可以实现异步、解耦、削峰（RabbitMQ 和 RocketMQ 能做的事情，它也能做）。因为 Kafka 的吞吐量更高，在大规模消息系统中更有优势。

第二个是大数据领域的使用，比如网站行为分析：

1.2.1.1.Website Activity Tracking 网站活动跟踪

把用户活动发布到数据管道中，可以用来做监控、实时处理、报表等等，比如社交网站的行为跟踪，购物网站的行为跟踪，这样可以实现更加精准的内容推荐。

举例：外卖、物流、电力系统的实时信息。

1.2.1.2.Log Aggregation 日志聚合

又比如用 Kafka 来实现日志聚合。这样就不用把日志记录到本地磁盘或者数据库，

实现分布式的日志聚合。

1.2.1.3.Metrics 应用指标监控

还可以用来记录运营监控数据。举例，对于贷款公司，需要监控贷款的业务数据：今天放出去多少笔贷款，放出去的总金额，用户的年龄分布、地区分布、性别分布等等。

或者对于运维数据的监控，CPU、内存、磁盘、网络连接的使用情况，可以实现告警。

1.2.2 数据集成+流计算

数据集成指的是把 Kafka 的数据导入 Hadoop、HBase 等离线数据仓库，实现数据分析。

第三块是流计算。什么是流（Stream）？它不是静态的数据，而是没有边界的、源源不断的产生的数据，就像水流一样。流计算指的就是 Stream 对做实时的计算。

Kafka 在 0.10 版本后，内置了流处理框架 API——Kafka Streams。

所以，它跟 RabbitMQ 的定位差别还是比较大的，不仅仅是一个简单的消息中间件，而且是一个流处理平台。在 Kafka 里面，消息被称为日志。日志就是消息的数据文件。

2. Kafka 安装与命令

2.1.Kafka CentOS 安装

注意：kafka_2.13-3.2.0.tgz 这个名字里面的版本号，前面（2.13）是 Scala 的版本号，后面（3.2.0）才是 Kafka 的版本号。

1、准备一个或者多个 CentOS 虚拟机，可以克隆

2、JDK 环境依赖

3、ZK 依赖或者使用自带 ZK，生产环境需要 ZK 集群，一般是在不同机器

4、命令可以配置别名

类别	位置
CentOS 安装 Zookeeper 单节点	参考预习资料
CentOS 安装 kafka 单机版	参考预习资料
CentOS Kafka 单机集群安装（伪集群）	参考预习资料
kafka 常用命令	参考预习资料
基于 Canal 和 Kafka 实现数据同步	参考预习资料

2.2.目录结构

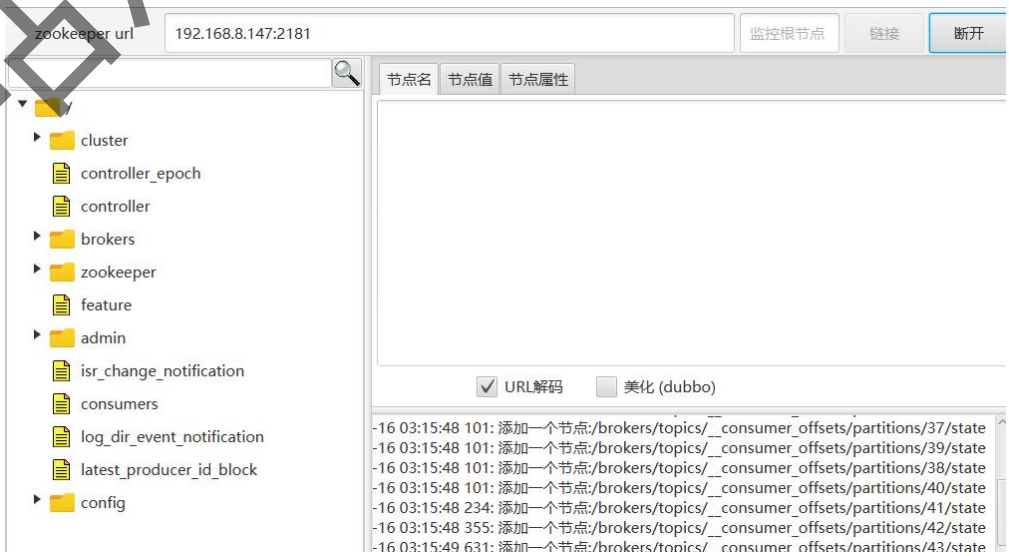
```
drwxr-xr-x. 3 root root   4096 Jul 16 05:53 bin
drwxr-xr-x. 3 root root   4096 Jul 12 16:39 config
drwxr-xr-x. 2 root root   8192 Jul 11 16:39 libs
-rw-r--r--. 1 root root  14640 May  3 20:52 LICENSE
drwxr-xr-x. 2 root root    262 May  3 20:58 licenses
drwxr-xr-x. 2 root root   8192 Jul 16 22:40 logs
-rw-------. 1 root root 339578 Jul 16 22:40 nohup.out
-rw-r--r--. 1 root root  28184 May  3 20:52 NOTICE
drwxr-xr-x. 2 root root     44 May  3 20:58 site-docs
```

2.3.Kafka 与 ZK 关系

在安装 Kafka 的时候我们知道，必须要依赖 ZK 的服务，在生产环境通常是 ZK 的集群，而且 Kafka 还自带了一个 ZK 服务。ZK 做了什么事情呢?

(通过 ZooViewer、或者 zktools 查看)

ZK 中的存储:



总结起来：利用 ZK 的有序节点、临时节点和监听机制，ZK 帮 Kafka 做了这些事情：配置中心（管理 Broker、Topic、Partition、Consumer 的信息，包括元数据的变动）、负载均衡、命名服务、分布式通知、集群管理和选举、分布式锁。

2.4.Kafka 脚本介绍

客户端的脚本是用 Java 编写的，本质上都是执行 Java 方法(kafka-run-class.sh)，做了进一步的封装，需要的参数更少。

脚本	作用
kafka-server-start.sh kafka-server-stop.sh	Kafka 启动停止
kafka-topics.sh	查看创建删除 topic
kafka-console-consumer.sh	消费者操作，例如监听 topic
kafka-consumer-groups.sh	消费者组操作
kafka-console-producer.sh	生产者操作，例如发送消息
zookeeper-server-start.sh	ZK 操作： 启动停止连接 ZK
kafka-reassign-partitions.sh	分区重新分配
kafka-consumer-perf-test.sh	性能测试

服务安装好了，我们先试一下发送消息，理解了 Kafka 的架构之后，我们再来详细分析一下客户端 API。

2.5.Kafka 界面管理工具

Kafka 没有自带管理界面，但是基于 admin 的接口可以开发。目前比较流行的管理界面主要是 kafka-manager 和 kafka-eagle（国产）。

https://github.com/yahoo/kafka-manager/releases

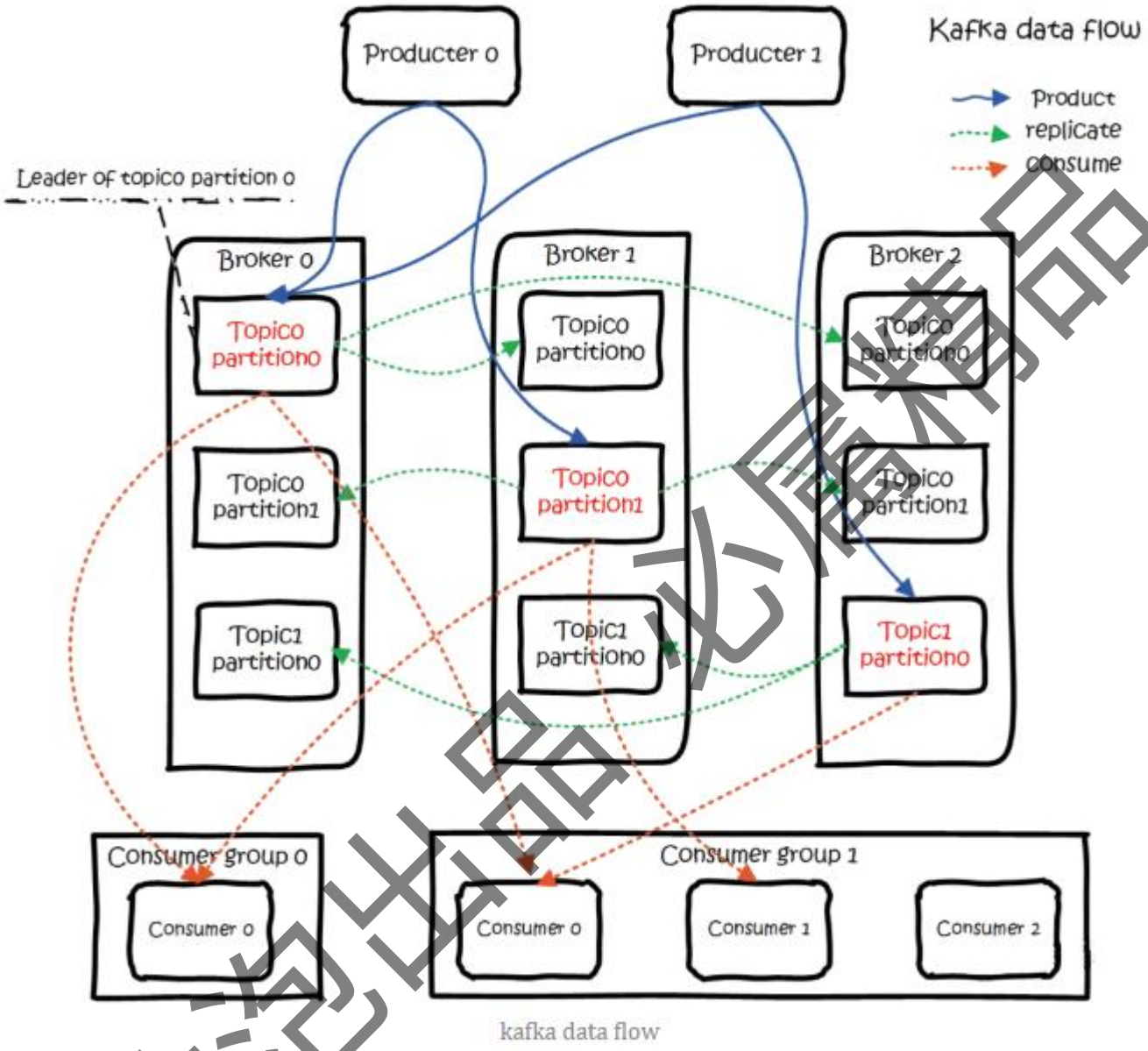
https://github.com/smartloli/kafka-eagle

注意最新版本的 cmak 对 Java 版本要求比较高，最低需要 JDK11。

kafka-eagle 对内存要求比较高，在虚拟机中部署需要修改 JVM 参数才能启动。

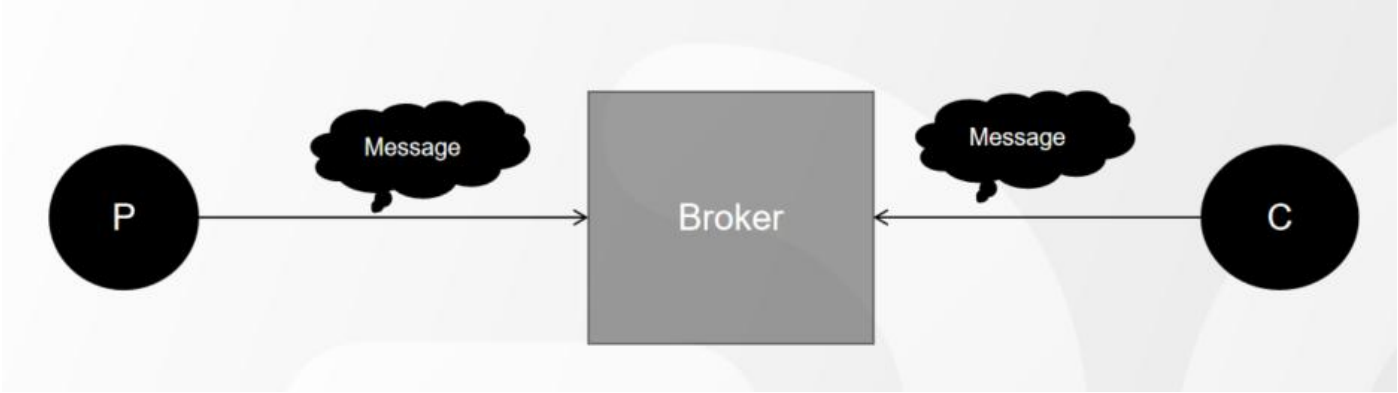
3. Kafka 架构分析

来看这张架构图，本堂课的目的就是看懂这张架构图，包括箭头和颜色。



kafka data flow

3.1.Broker



Broker：Kafka 作为一个中间件，是帮我们存储和转发消息的，它做的事情就像中介，所以 Kafka 的服务也叫做 Broker，默认是 9092 的端口。生产者和消费者都需要跟这个 Broker 建立一个连接，才可以实现消息的收发。

3.2.消息

客户端之间传输的数据叫做消息，或者叫做记录（Record 名词 [ˈrekɔːd]）。

在客户端的代码中，Record 可以是一个 KV 键值对。

生产者对应的封装类是 ProducerRecord，消费者对应的封装类是 ConsumerRecord。

消息在传输的过程中需要序列化，所以代码里面要指定序列化工具。

消息在服务端的存储格式（RecordBatch 和 Record）：

http://kafka.apache.org/documentation/#messageformat

3.3.生产者

发送消息的一方叫做生产者，接收消息的一方叫做消费者。

为了提升消息发送速率，生产者不是逐条发送消息给 Broker，而是批量发送的。

多少条发送一次由一个参数决定。

```
pros.put("batch.size",16384);
```

3.4.消费者

一般来说消费者获取消息有两种模式，一种是 Pull 模式，一种是 Push 模式。

Pull 模式就是消费放在 Broker，消费者自己决定什么时候去获取。Push 模式是消息放在 Consumer，只要有消息到达 Broker，都直接推给消费者。

RabbitMQ Consumer 及支持 Push 又支持 Pull，一般用的是 Push。Kafka 只有 Pull 模式。

为什么消费者用 Pull，官网已经说得很明白了：

http://kafka.apache.org/documentation/#design_pull

在 Push 模式下，如果消息产生速度远远大于消费者消费消息的速率，那消费者就会不堪重负（你已经吃不下了，但是还要不断地往你嘴里塞），直到挂掉。

而且消费者可以自己控制一次到底获取多少条消息：

```
max.poll.records
```

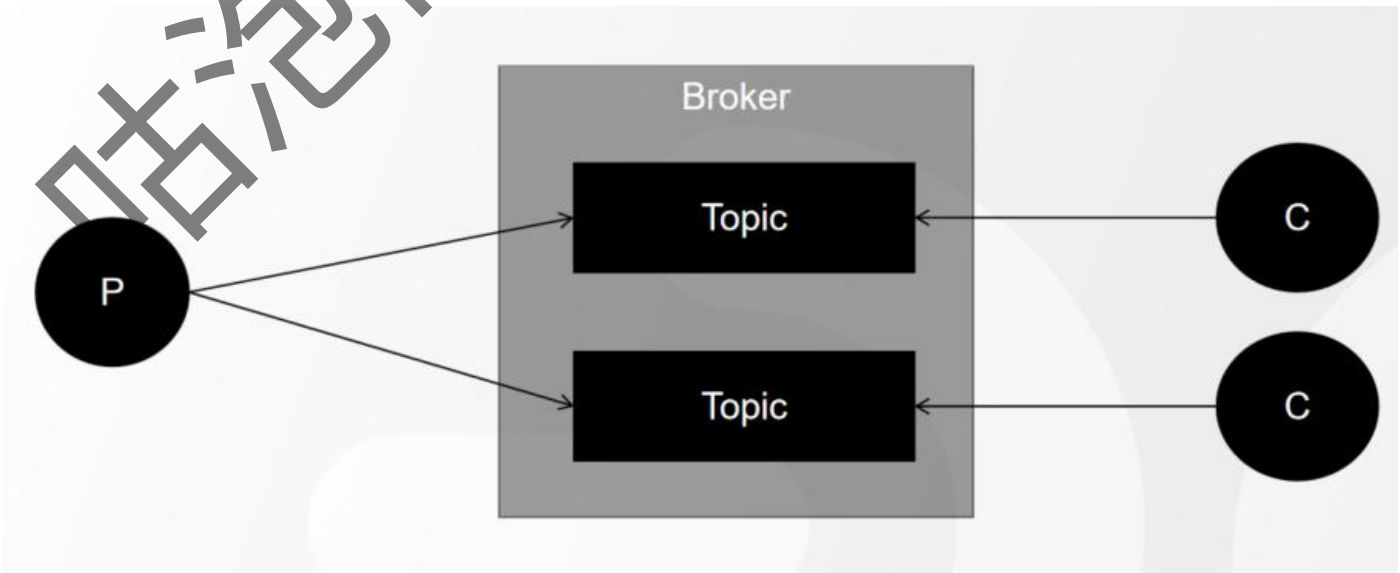
默认 500。在 poll 方法里面可以指定。

3.5.Topic

生产者跟消费者是怎么关联起来的呢？或者说，生产者发送的消息，怎么才能到达某个特定的消费者？他们要通过队列关联起来，也就是说，生产者发送消息，要指定发给哪个队列。消费者接收消息，要指定从哪个队列接收。

在 Kafka 里面，这个队列叫做 Topic，是一个逻辑的概念，可以理解为一组消息的集合（不同业务用途的消息）。

生产者和 Topic 以及 Topic 和消费者的关系都是多对多。一个生产者可以发送消息到多个 Topic，一个消费者也可以从多个 Topic 获取消息（但是不建议这么做）。



注意，生产者发送消息时，如果 Topic 不存在，会自动创建。由一个参数控制。

http://kafka.apache.org/documentation/#auto.create.topics.enable

```
auto.create.topics.enable
```

默认为 true。如果要彻底删掉一个 Topic，这个参数必须改成 false，否则只要有代码使用这个 Topic，它就会自动创建。

3.6.Partition 与 Cluster

如果说一个 Topic 中的消息太多，会带来两个问题：

第一个是不方便横向扩展，比如我想要在集群中把数据分布在不同的机器上实现扩展，而不是通过升级硬件做到，如果一个 Topic 的消息无法在物理上拆分到多台机器的时候，这个是做不到的。

第二个是并发或者负载的问题，所有的客户端操作的都是同一个 Topic，在高并发的场景下性能会大大下降。

怎么解决这个问题呢？我们想到的就是把一个 Topic 进行拆分(分片的思想来啦)。

Kafka 引入了一个分区（Partition）的概念。一个 Topic 可以划分成多个分区。

分区在创建 Topic 的时候指定，每个 Topic 至少有一个分区。

创建 Topic 的命令：

```
sh kafka-topics.sh --create --topic mytopic --bootstrap-server 192.168.8.147:9092 --replication-factor 1 --partitions 2
```

如果没有指定分区数，默认的分区数是一个，这个参数可以修改：

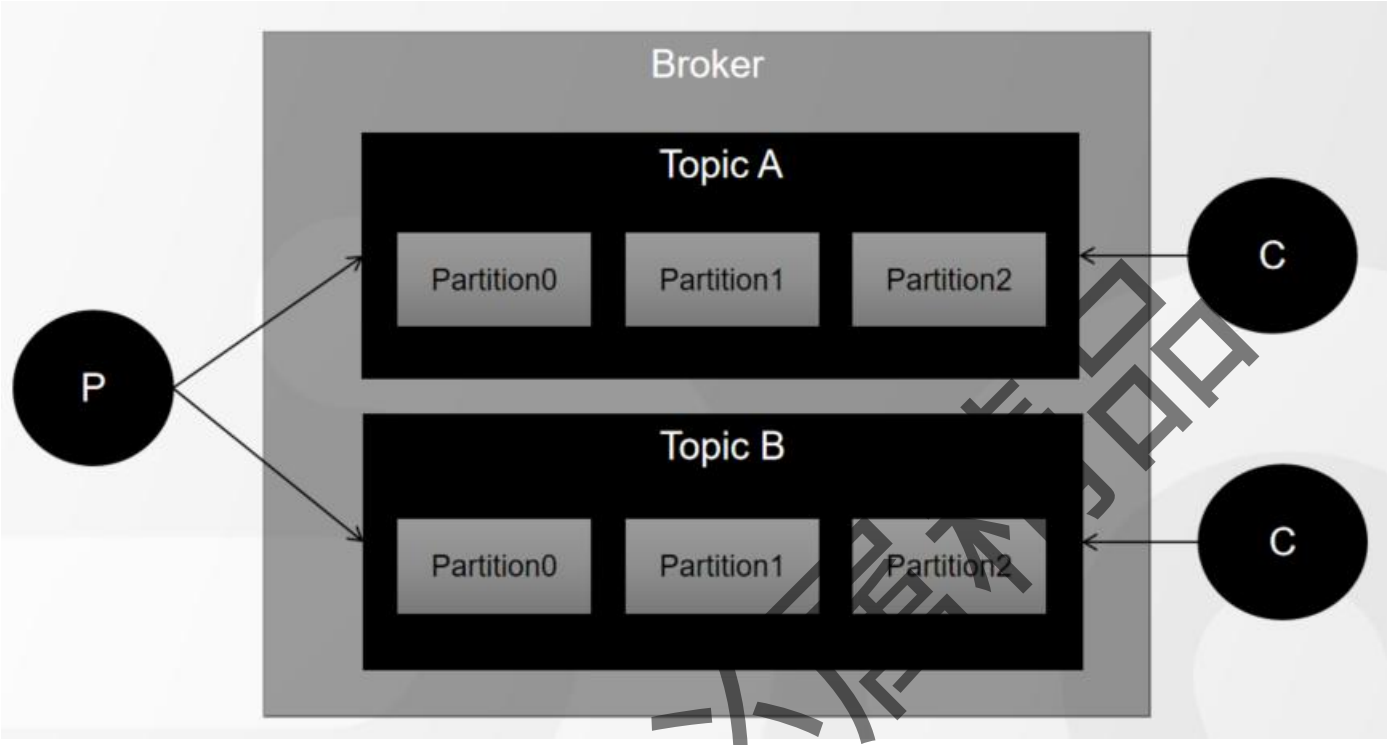
```
num.partitions=1
```

partitions 是分区数，replication-factor 副本因子（相当于是主题的副本数）。

Partition 思想上有点类似于分库分表，实现的也是横向扩展和负载的目的。

举个例子，Topic 有 3 个分区，生产者依次发送 9 条消息，对消息进行编号。

第一个分区存 1 4 7，第二个分区存 2 5 8 ，第三个分区存 3 6 9，这个就实现了负载。



每个 partition 都有一个物理目录。在配置的数据目录下（日志就是数据）：

/tmp/kafka-logs/

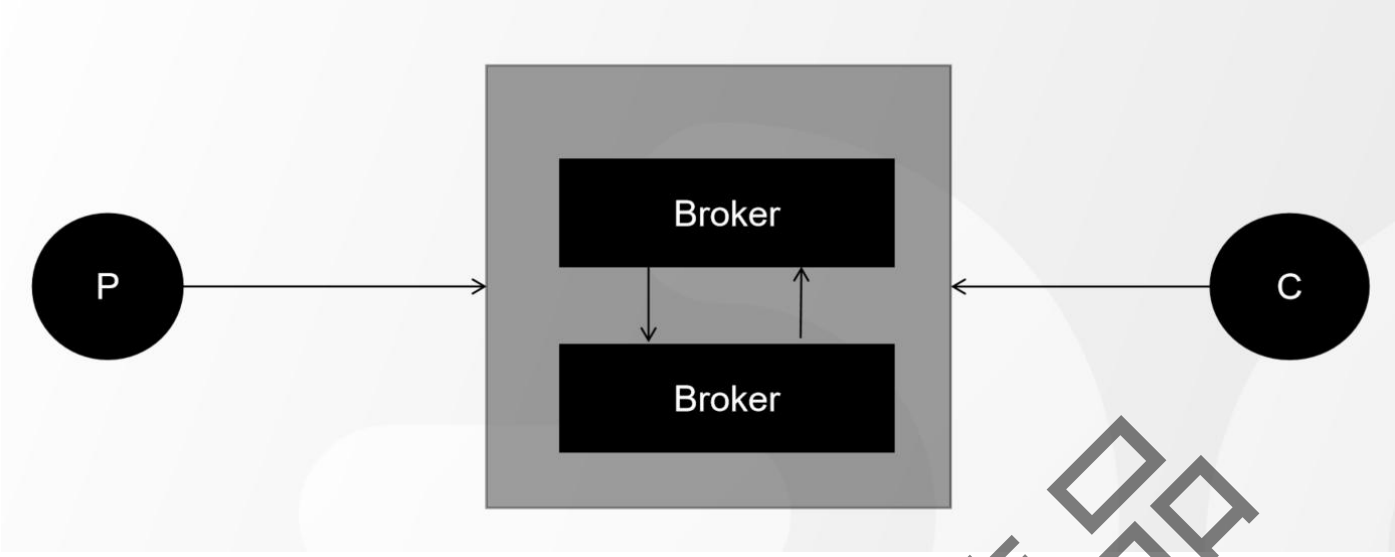
```
tomdemo-0
tomdemo-1
```

跟 RabbitMQ 不一样的地方是，Partition 里面的消息被读取之后不会被删除，所以同一批消息在一个 Partition 里面顺序、追加写入的。这个也是 Kafka 吞吐量大的一个很重要的原因。

分区数量怎么选择呢？是不是分区数越多越好？不一定。不同的机器网络环境，这个答案不尽相同，最好是通过性能测试的脚本验证。

3.7.Partition 副本 Replica 机制

如果 Partition 的数据只存储一份，在发生网络或者硬件故障的时候，该分区的数据就无法访问或者无法恢复了。



Kafka 在 0.8 的版本之后增加了副本机制。

每个 Partition 可以有若干个副本（Replica），副本必须在不同的 Broker 上面。一般我们说的副本包括其中的主节点。

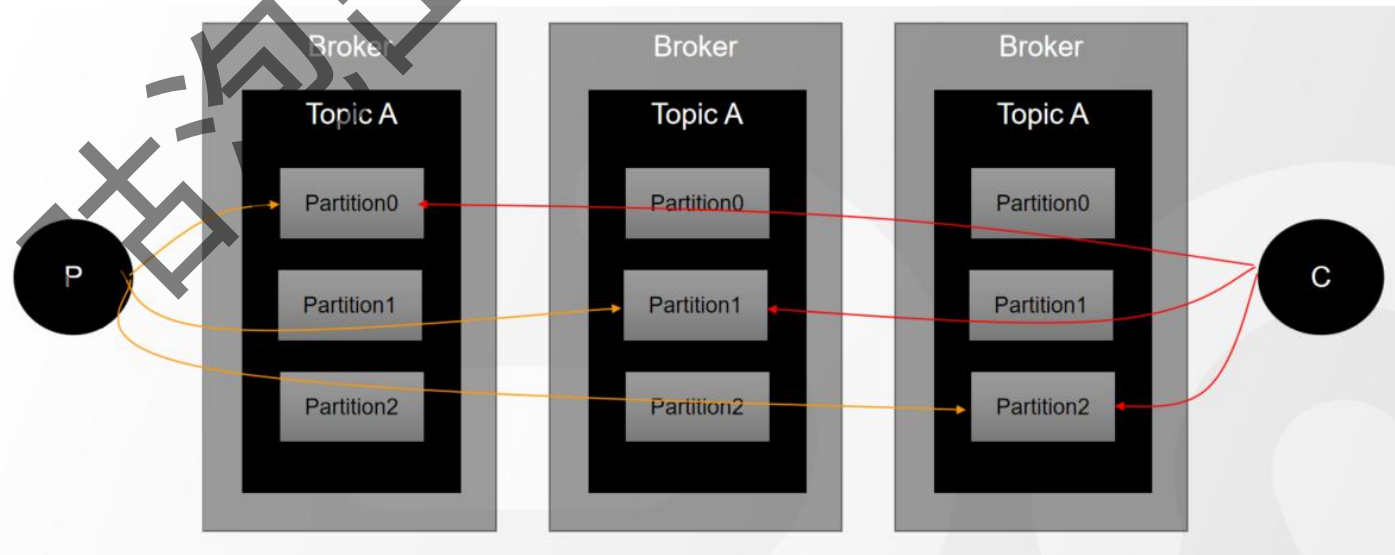
由 replication-factor 指定一个 Topic 的副本数。

```
sh kafka-topics.sh --create --topic test3p3r--bootstrap-server 192.168.8.147:9092 --replication-factor 3 --partitions 3
```

服务端有一个参数控制默认的副本数：

```
offsets.topic.replication.factor
```

举例：部署了 3 个 Broker，该 Topic 有 3 个分区，每个分区一共 3 个副本。



注意：这些存放相同数据的 Partition 副本有 Leader（图中红色）和 Follower（图中绿色）的概念。Leader 在 哪台机器是不一定的，选举出来的。

生产者发消息、消费者读消息都是针对Leader(为什么不让客户端读Leader呢？)。Follower 的数据是从 Leader 同步过来的。

3.8.Segment

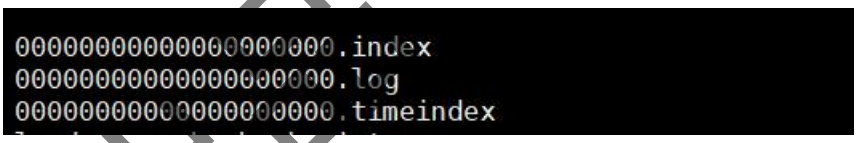
Kafka 的数据是放在后缀.log 的文件里面的。如果一个 Partition 只有一个 log 文件，消息不断地追加，这个 log 文件也会变得越来越大，这个时候要检索数据效率就很低了。

所以干脆把 Partition 再做一个切分，切分出来的单位就叫做段（Segment）。实际上 Kafka 的存储文件是划分成段来存储的。

默认存储路径：/tmp/kafka-logs/

每个 Segment 都有至少有 1 个数据文件和 2 个索引文件，这 3 个文件是成套出现的。

（Partition 一个目录，一个 Segement 一套文件）



Segment 多大一个呢？默认大小是 1073741824 bytes（1G），由这个参数控制：

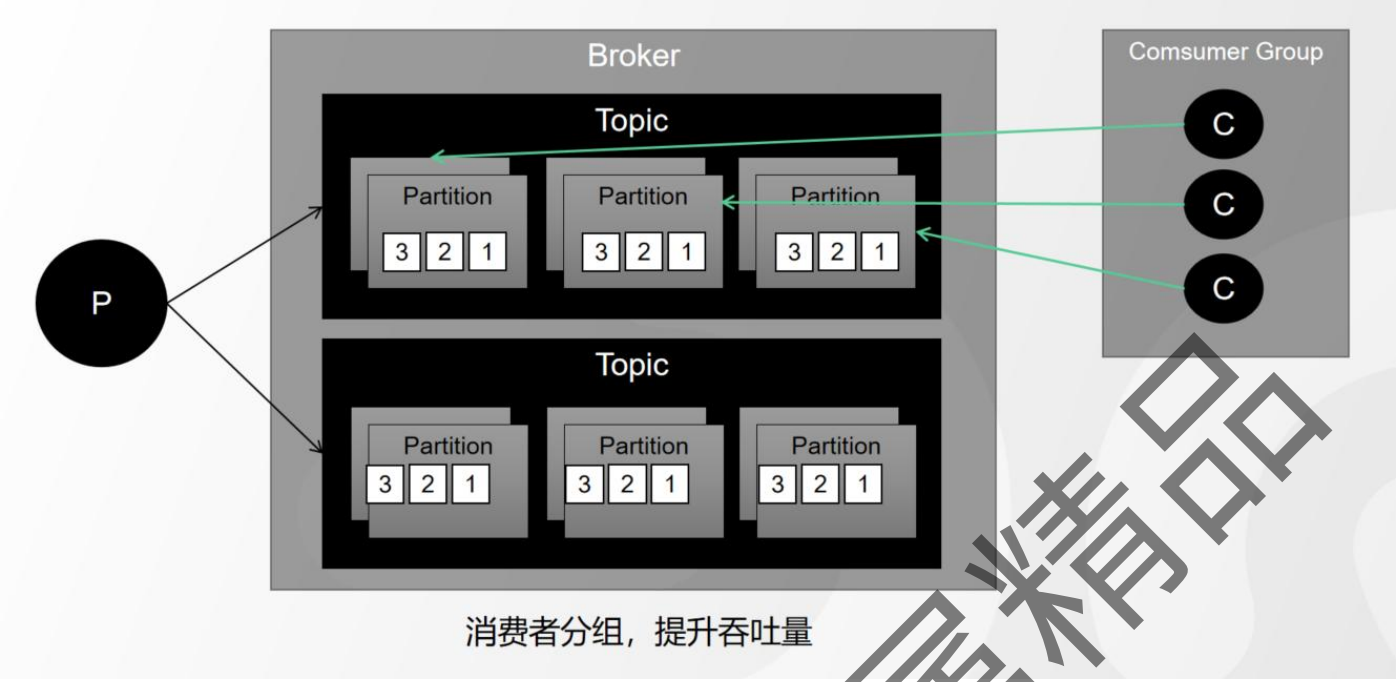
```
log.segment.bytes
```

3.9.Consumer Group

如果生产者生产消息的速率过快，会造成消息在 Broker 的堆积，影响 Broker 的性能。怎么提升消息的消费速率呢？增加消费者的数量。但是这么多消费者，怎么知道大家是不是消费的同一个 Topic 呢?

所以引入了一个 Consumer Group 消费组的概念，在代码中通过 group id 来配置。消费同一个 Topic 的消费者不一定是同一个组，只有 group id 相同的消费者才

是同一个消费者组。



注意：

同一个 Group 中的消费者，不能消费相同的 Partition——Partition 要在消费者之间分配。

怎么理解呢？大家在上大学的时候是没有固定的教室的吧？教室里面的座位可以理解成 Partition，分区。一个教室有很多班级都可以使用，一个班级就可以理解为 Consumer Group。很显然，对于一个班级来说，是不可能两个人坐一张桌子的。但是对于不同的班级，却是可以的，比如编号为 0 的这张桌子，5 班的某个学生用，6 班的某个学生用，7 班的某个学生也用。

那就会有一个问题，如果学生比桌子多，或者学生比桌子少，怎么办？

1、如果消费者比 Partition 少，一个消费者可能消费多个 Partition（两张桌子凑一起，躺在桌子上都可以）。

2、如果消费者比 Partition 多，肯定有消费者没有 Partition 可以消费（两个人不能挤在一张桌子上，有一个人站着上课了）。不会出现一个 Group 里面的消费者消费同一个 Partition 的情况。

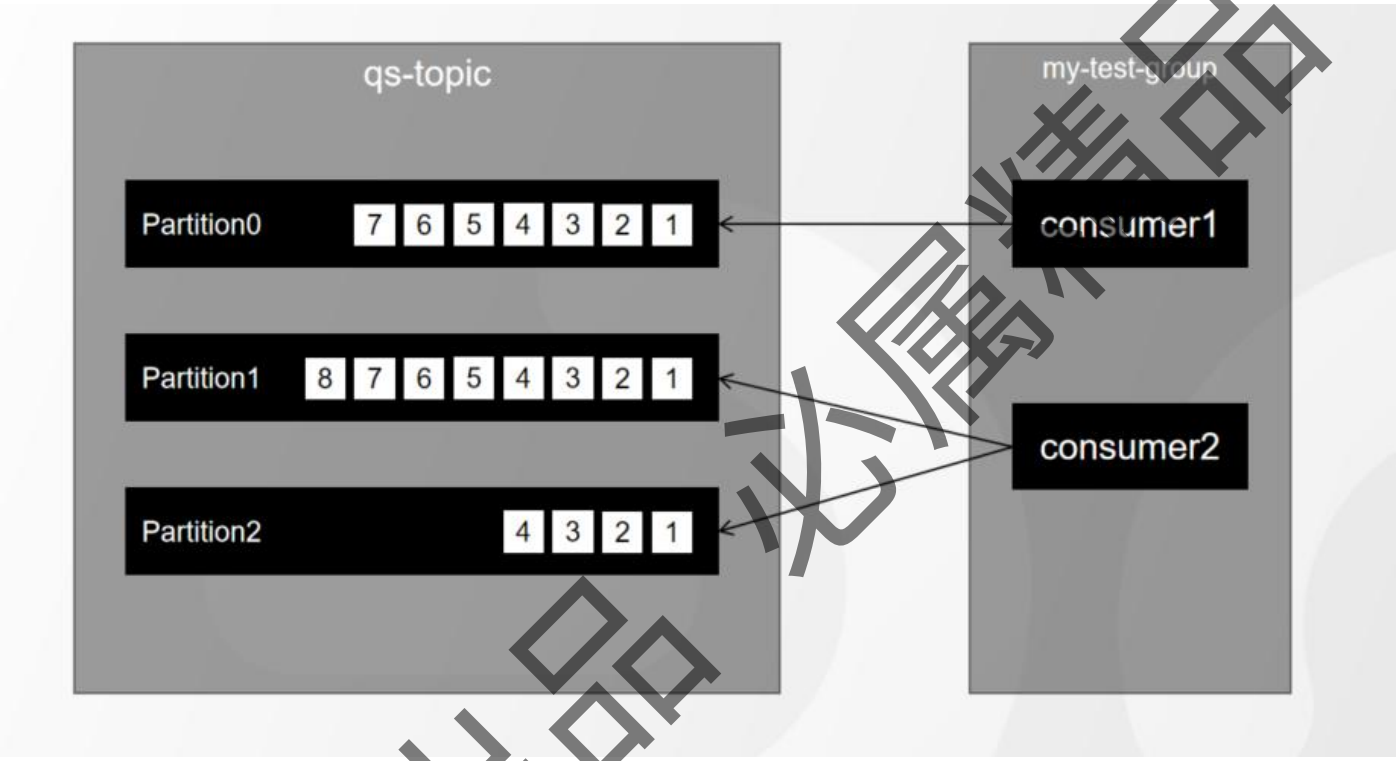
如果想要同时消费同一个 Partition 的消息，那么需要其他的组来消费。

3.10.Consumer Offset

我们前面说过了，Partition 里面的消息是顺序写入的，被读取之后不会被删除。

如果消费者挂了或者下一次读取，想要接着上次的位置读取消息，或者从某个特定的位置读取消息，怎么办呢？会不会出现重复消费的情况？

因为消息是有序的，我们可以对消息进行编号，用来标识一条唯一的消息。

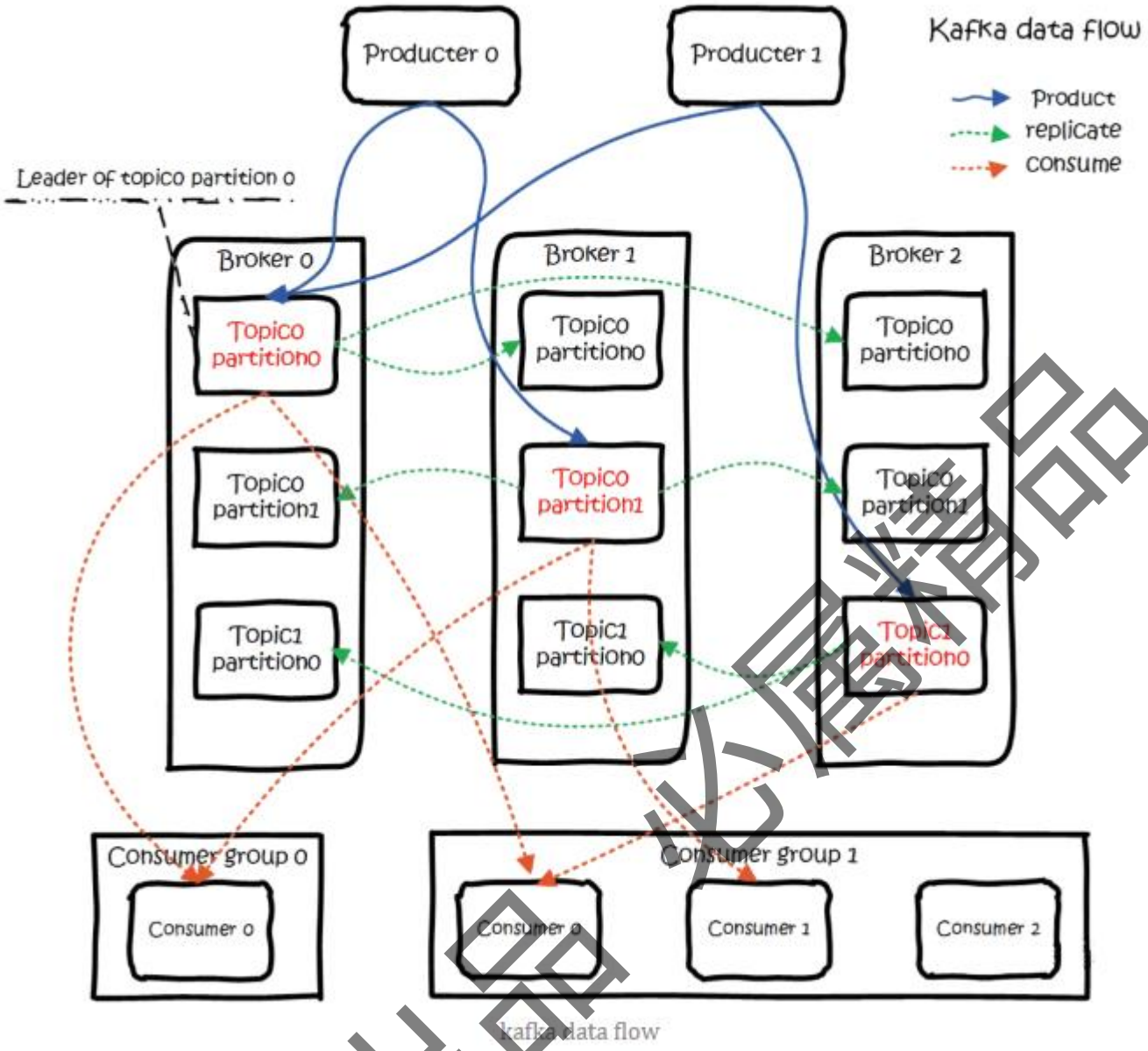


这个编号我们就把它叫做 Offset，偏移量。

Offset 记录着下一条将要发送给 Consumer 的消息的序号。

这个消费者跟 Partition 之间的偏移量没有保存在 ZK，而是直接保存在服务端。

看一下架构图。



解读：

首先，我们有 3 台 Broker。

有两个 Topic：Topic0 和 Topic1。

Topic0 有 2 个分区：Partition0 和 Partition1，每个分区一共 3 个副本。

Topic1 只有 1 个分区：Partition0，每个分区一共 3 个副本。

图中红色字体的副本代表是 Leader，黑色字体的副本代表是 Follower。绿色的线代表是数据同步。

蓝色的线是写消息，橙色的线是读消息，都是针对 Leader 节点。

有两个消费者组，第一个消费者组，消费了 Topic0 的两个分区（两张桌子凑一起，

躺在桌子上！）。

第二个消费者组，既消费 Topic0，又消费 Topic1。其中有一个消费者，消费 Topic0 的 Partition0，还消费 Topic1 的 Partition0（他在两个教室都有一个座位）。有一个消费者，消费 Partition0 的 Partition1。有一个消费者，没有 Partition 可以消费（站着上课！）

4. Kafka Java 开发

4.1Java API

根据官网，Kafka 主要有 5 种 API：

类别	作用
Producer API	用于应用将数据发送到 Kafka 的 topic
Consumer API	用于应用从 Kafka 的 topic 中读取数据流
Admin API	允许管理和检测 Topic、broker 以及其他 Kafka 实例，与 Kafka 自带的脚本命令作用类似 参考：com.gupaoedu.vip.mq.kafka.javaapi.admin.TopicManage http://kafka.apache.org/26/javadoc/index.html?org/apache/kafka/clients/admin/Admin.html
Streams API	用于从来源 topic 转化到目的 topic 转换数据流，作用跟 Spark 、Storm、Flink 一样（应用）
Connect API	用于持续地从一些源系统输入数据到 Kafka，或者从 Kafka 推送数据到一些系统，比如数据库或者 Hadoop 等等（存储）

这里我们主要用到生产者和消费者的 API。

Admin API 演示：

```
com.gupaoedu.vip.mq.kafka.javaapi.admin.TopicManage
```

4.1.1 引入依赖

注意，客户端的版本跟服务端的版本要匹配。如果客户端版本太高，服务端版本太旧，有可能无法连接。

```
<dependency>
      <groupId>org.apache.kafka</groupId>
```

```
            <artifactId>kafka-clients</artifactId>
            <version>2.6.0</version>
        </dependency>
```

topic 要先提前创建，或者配置允许自动创建 topic。

auto.create.topics.enable

4.1.2. 消费者

com.gupaoedu.vip.mq.kafka.javaapi.simple.SimpleConsumer

```
public class SimpleConsumer {
   public static void main(String[] args) {
       Properties props= new Properties();
       props.put("bootstrap.servers","192.168.8.147:9092");
       props.put("group.id","gp-test-group");
       // 是否自动提交偏移量，只有 commit 之后才更新消费组的 offset
       props.put("enable.auto.commit","true");
       // 消费者自动提交的间隔
       props.put("auto.commit.interval.ms","1000");
       // 从最早的数据开始消费 earliest | latest | none
       props.put("auto.offset.reset","earliest");
       props.put("key.deserializer","org.apache.kafka.common.serialization.StringDeserializer");
       props.put("value.deserializer","org.apache.kafka.common.serialization.StringDeserializer");

       KafkaConsumer<String,String> consumer=new KafkaConsumer<String, String>(props);
       // 订阅队列
       consumer.subscribe(Arrays.asList("mytopic"));
       try {
           while (true){
               ConsumerRecords<String,String> records=consumer.poll(Duration.ofMillis(1000));
               for (ConsumerRecord<String,String> record:records){
                   System.out.printf("offset = %d ,key =%s, value= %s,
partition= %s%n",record.offset(),record.key(),record.value(),record.partition());
               }
           }
       }finally {
           consumer.close();
       }
   }
}
```

4.1.3. 生产者

com.gupaoedu.vip.mq.kafka.javaapi.simple.SimpleProducer

```
public class SimpleProducer {
   public static void main(String[] args) {
       Properties pros=new Properties();
       pros.put("bootstrap.servers","192.168.8.147:9092");
       pros.put("key.serializer","org.apache.kafka.common.serialization.StringSerializer");
       pros.put("value.serializer","org.apache.kafka.common.serialization.StringSerializer");
       // 0 发出去就确认 | 1 leader 落盘就确认| all 所有 Follower 同步完才确认
       pros.put("acks","1");
       // 异常自动重试次数
       pros.put("retries",3);
       // 多少条数据发送一次，默认 16K
       pros.put("batch.size",16384);
       // 批量发送的等待时间
       pros.put("linger.ms",5);
       // 客户端缓冲区大小，默认 32M，满了也会触发消息发送
       pros.put("buffer.memory",33554432);
       // 获取元数据时生产者的阻塞时间，超时后抛出异常
       pros.put("max.block.ms",3000);

       Producer<String,String> producer = new KafkaProducer<String,String>(pros);

       for (int i =0 ;i<100;i++) {
           producer.send(new ProducerRecord<String,String>("mytopic",Integer.toString(i),Integer.toString(i)));
           // System.out.println("发送:"+i);
      }

       producer.close();
  }
}
```


4.2.Kafka 与 Spring Boot 集成

版本对应关系：

https://spring.io/projects/spring-kafka

4.2.1 引入依赖

spring-kafka

```
        <dependency>
            <groupId>org.springframework.kafka</groupId>
            <artifactId>spring-kafka</artifactId>
        </dependency>
```

4.2.2 配置

application.properties

```
server.port=7271
spring.kafka.bootstrap-servers=192.168.8.147:9092

# producer
spring.kafka.producer.retries=1
spring.kafka.producer.batch-size=16384
spring.kafka.producer.buffer-memory=33554432
spring.kafka.producer.acks=1
spring.kafka.producer.properties.linger.ms=5
spring.kafka.producer.key-serializer=org.apache.kafka.common.serialization.StringSerializer
spring.kafka.producer.value-serializer=org.apache.kafka.common.serialization.StringSerializer

# consumer
spring.kafka.consumer.auto-offset-reset=earliest
spring.kafka.consumer.enable-auto-commit=true
spring.kafka.consumer.auto-commit-interval=1000
spring.kafka.consumer.key-deserializer=org.apache.kafka.common.serialization.StringDeserializer
spring.kafka.consumer.value-deserializer=org.apache.kafka.common.serialization.StringDeserializer
```

4.2.3 消费者

使用注解@KafkaListener 监听 Topic

```
@Component
public class ConsumerListener {
   @KafkaListener(topics = "springboottopic",groupId = "springboottopic-group")
   public void onMessage(String msg){
       System.out.println("----收到消息: "+msg+"----");
  }
}
```

4.2.4 生产者

注入模板方法 KafkaTemplate 发送消息。

注意 send 方法有很多重载。异步回调 ListenableFuture。

```
@Component
public class KafkaProducer {
   @Autowired
   private KafkaTemplate<String,Object> kafkaTemplate;

   public String send(@RequestParam String msg){
       kafkaTemplate.send("springboottopic", msg);
       return "ok";
  }
}
```

4.2.5 测试

先启动 Application，启动消费者监听。再运行单元测试类。

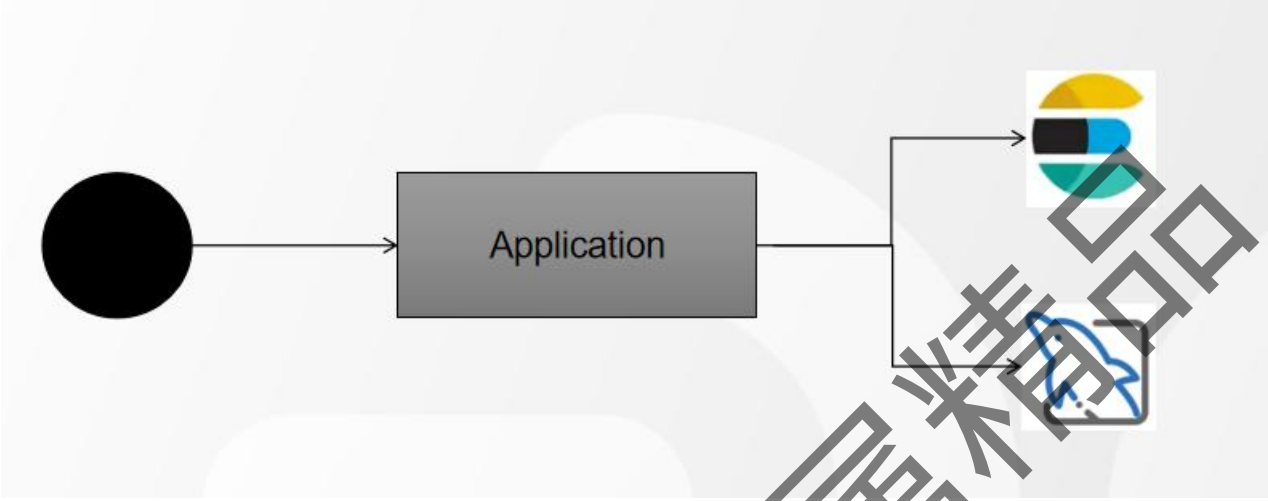
```
@SpringBootTest
class KafkaTests {

   @Autowired
   KafkaProducer controller;

   // 消费者：先启动 kafkaApp
   @Test
   void contextLoads() {
       long time = System.currentTimeMillis();
       System.out.println("----"+time +", 已经发出----");
       controller.send("qingshan penyuyan, " +time);
  }
}
```

总结一下：经过了 Spring 的封装，基于 Kafka 实现异步通信变得更加简单了，跟其他的消息中间件一样，这样我们就可以更加专注于业务的开发，而不用重复去开发底层组件。

Kafka 作为普通的消息队列使用的场景我们就不说了，只需要在需要发送消息的地方注入 Template。下面我们来看一个有意思的案例。

4.3.Kafka 集成 Canal 实现数据同步

我们有一个需求，因为商品的数据同时存储在 MySQL 和 ES，当商品数据有变动的时候，比如增删改，必须要让两边保持一致。怎么做呢？



直接改代码？就是每个操作 MySQL 的地方，都同时操作 ES。有点麻烦了，代码改动太多。

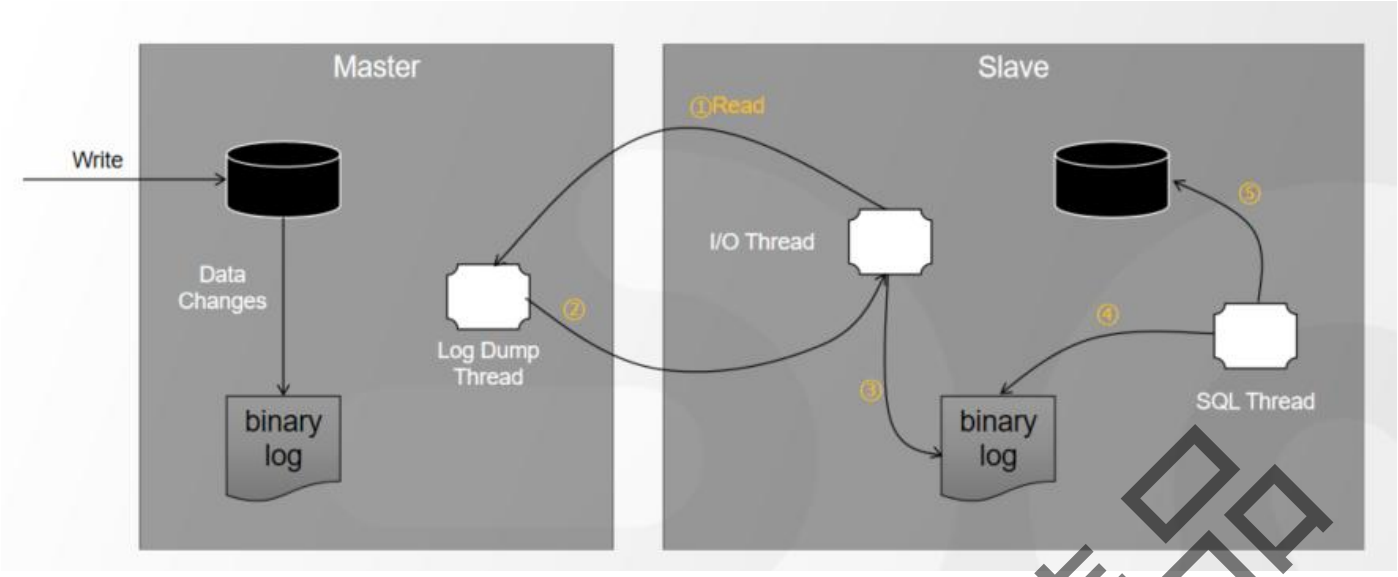
那用拦截器实现呢（比如 Spring AOP）？如果是商品表，如果是增删改查，就同步操作 ES。但是有可能会出现没有被拦截到的情况。

用数据同步工具嘛？不断地读写 MySQL 的数据，同步到 ES，这个有点浪费资源，而且实时性不高。

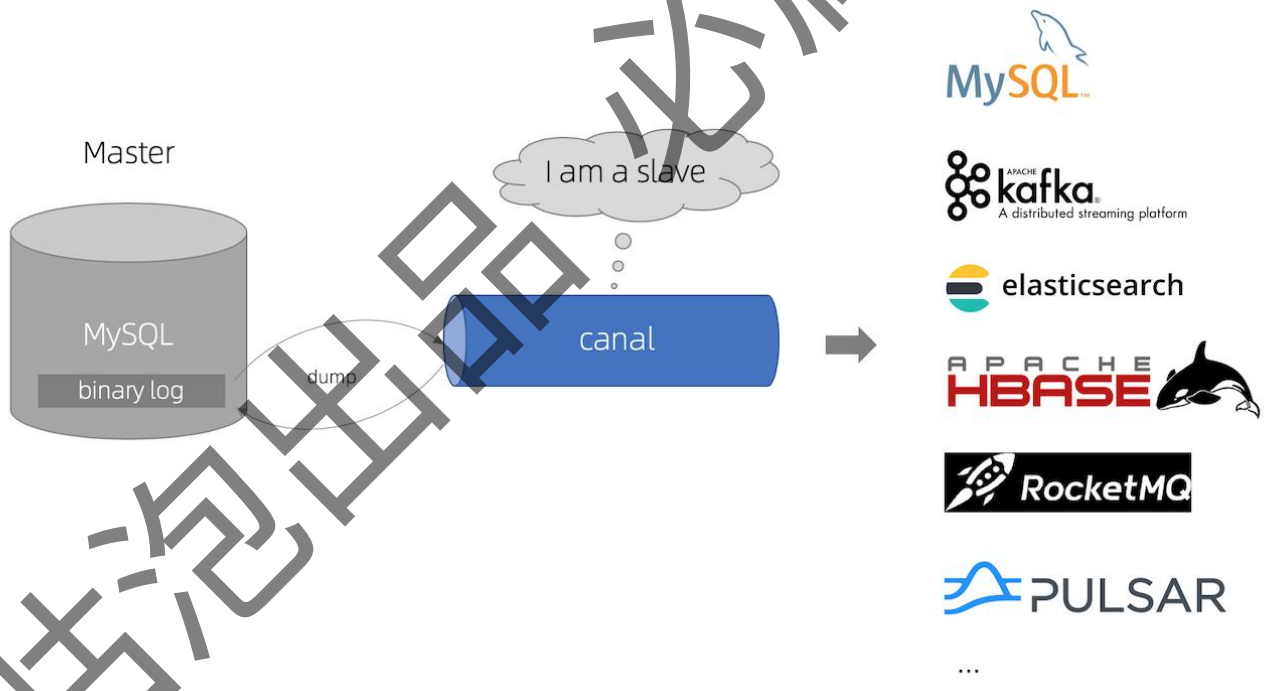
所以，怎么获取到 MySQL 数据库变动的信息呢？突然想起来，MySQL 的服务层有一个 Binlog，会记录所有的增删改语句。好家伙，这不就是我们所需要的吗？但是怎么实时读取它的变动呢？

不用我们自己写代码了，阿里有一个开源的工具可以帮我们实现这个需求，就是 Canal。

我们知道 Binlog 有一个非常重要的功能就是实现主从同步。



所以 Canal 利用了这一点，把自己伪装成了一个 Slave 节点，不断请求最新的 Binlog。有了 Binlog 就好办了，Canal 会解析 Binlog 的内容，把它发送给关注数据库变动的接收者，完成后续逻辑的处理。



Canal 是一款纯 Java 开发的数据同步工具，可以支持 Binlog 增量订阅（注意不是全量）。Binlog 设置为 row 模式以后，不仅能获取到执行的每一个增删改的脚本，同时还能获取到修改前和修改后的数据。

所以，作为一款 Binlog 增量数据解析的工具，Canal 可以做的事情很多，比如说备份数据，缓存的同步刷新，构建 ES 索引，增量数据的处理等等。

Canal 可以用多种方式把数据增量变动的信息发送出去，比如 TCP 和多种 MQ，目前支持 Kafka、 RabbitMQ、 RocketMQ，而且提供了同步数据到 HBase、Elasticsearch 的适配器。

工作流程：数据变动——产生 Binlog 信息——Canal 服务端获取 Binlog 信息——发送 MQ 消息——消费者消费 MQ 信息，完成后续逻辑处理。

5. Kafka 和 RabbitMQ 对比

6.1 Kafka 主要特性

市面上有这么多 MQ 的产品 Kafka 跟他们有什么区别呢？我们说一个产品的诞生背景决定了它的特性，特性决定使用场景。

因为 kafka 是用来解决数据流的传输的问题的，所以它有这些特性：

- 高吞吐、低延迟：Kakfa 最大的特点就是收发消息非常快，Kafka 每秒可以处理几十万条消息，它的最低延迟只有几毫秒；
- 高伸缩性：如果可以通过增加分区 Partition 来实现扩容。不同的分区可以在不同的 Broker 中。通过 ZK 来管理 Broker 实现扩展，ZK 管理 Consumer 可以实现负载；
- 持久性、可靠性：Kafka 能够允许数据的持久化存储，消息被持久化到磁盘，并支持数据备份防止数据丢失；
- 容错性：允许集群中的节点失败，某个节点宕机，Kafka 集群能够正常工作；
- 高并发：支持数千个客户端同时读写。

6.2 Kafka 与 RabbitMQ 对比

Kafka 和 RabbitMQ 的主要区别：

1、 产品侧重：Kafka：流式消息处理、消息引擎；RabbitMQ：消息代理

2、 性能：Kafka 有更高的吞吐量。RabbitMQ 主要是 Push，Kafka 只有 Pull。

3、 消息顺序：分区里面的消息是有序的，同一个 Consumer Group 里面的一个消费者只能消费一个 Partition，能保证消息的顺序性。

4、 消息的路由和分发：RabbitMQ 更加灵活。

5、 延迟消息、死信队列：RabbitMQ 支持。

6、 消息的留存：Kafka 消费完之后消息会留存，RabbitMQ 消费完就会删除。Kafka 可以设置 retention，清理消息。

优先选择 RabbitMQ 的情况：

高级灵活的路由规则；

消息时序控制（控制消息过期或者消息延迟）；

高级的容错处理能力，在消费者更有可能处理消息不成功的情景中（瞬时或者持久）；

更简单的消费者实现。

优先选择 Kafka 的情况：

严格的消息顺序；

延长消息留存时间，包括过去消息重放的可能；

传统解决方案无法满足的高伸缩能力。