



第1章 Kafka入门

学习目标

- 了解消息队列的应用场景
- 能够搭建Kafka集群
- 能够完成生产者、消费者Java代码编写
- 理解Kafka的架构，以及Kafka的重要概念
- 了解Kafka的事务

1. 简介

1.1 消息队列简介

1.1.1 什么是消息队列

消息队列，英文名：Message Queue，经常缩写为MQ。从字面上来理解，消息队列是一种用来存储消息的队列。来看一下下面的代码：

```
// 1. 创建一个保存字符串的队列
Queue<String> stringQueue = new LinkedList<String>();

// 2. 往消息队列中放入消息
stringQueue.offer("hello");

// 3. 从消息队列中取出消息并打印
System.out.println(stringQueue.poll());
```

上述代码，创建了一个队列，先往队列中添加了一个消息，然后又从队列中取出了一条消息。这说明了队列是可以用来存取消息的。

我们可以简单理解消息队列就是**将需要传输的数据存放在队列中**。

1.1.2 消息队列中间件

消息队列中间件就是用来存储消息的软件（组件）。举个例子来理解，为了分析网站的用户行为，我们需要记录用户的访问日志。这些一条条的日志，可以看成是一条条的消息，我们可以将它们保存到消息队列中。将来有一些应用程序需要处理这些日志，就可以随时将这些消息取出来处理。

目前市面上的消息队列有很多，例如：Kafka、RabbitMQ、ActiveMQ、RocketMQ、ZeroMQ等。

1.1.2.1 为什么叫Kafka呢

Kafka的架构师jay kreps非常喜欢franz kafka（弗兰兹·卡夫卡），并且觉得kafka这个名字很酷，因此取了个和消息传递系统完全不相干的名称kafka，该名字并没有特别的含义。

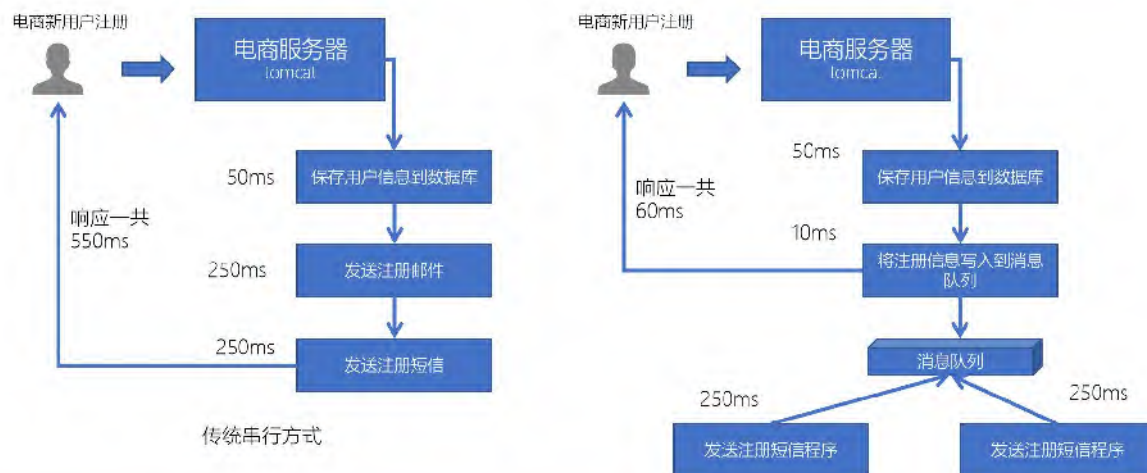
「也就是说，你特别喜欢尼古拉斯赵四，将来你做一个项目，也可以把项目的名字取名为：尼古拉斯赵四，然后这个项目就火了」

1.1.3 消息队列的应用场景

1.1.3.1 异步处理

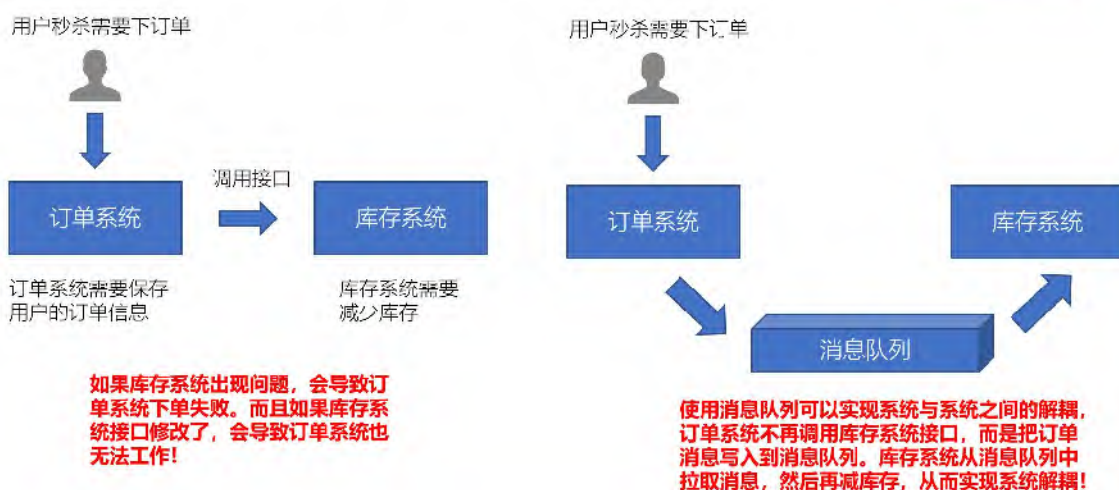
电商网站中，新的用户注册时，需要将用户的信息保存到数据库中，同时还需要额外发送注册的邮件通知、以及短信注册码给用户。但因为发送邮件、发送注册短信需要连接外部的服务器，需要额外等待一段时间，此时，就可以使用消息队列来进行异步处理，从而实现快速响应。

消息队列应用场景 - 异步处理

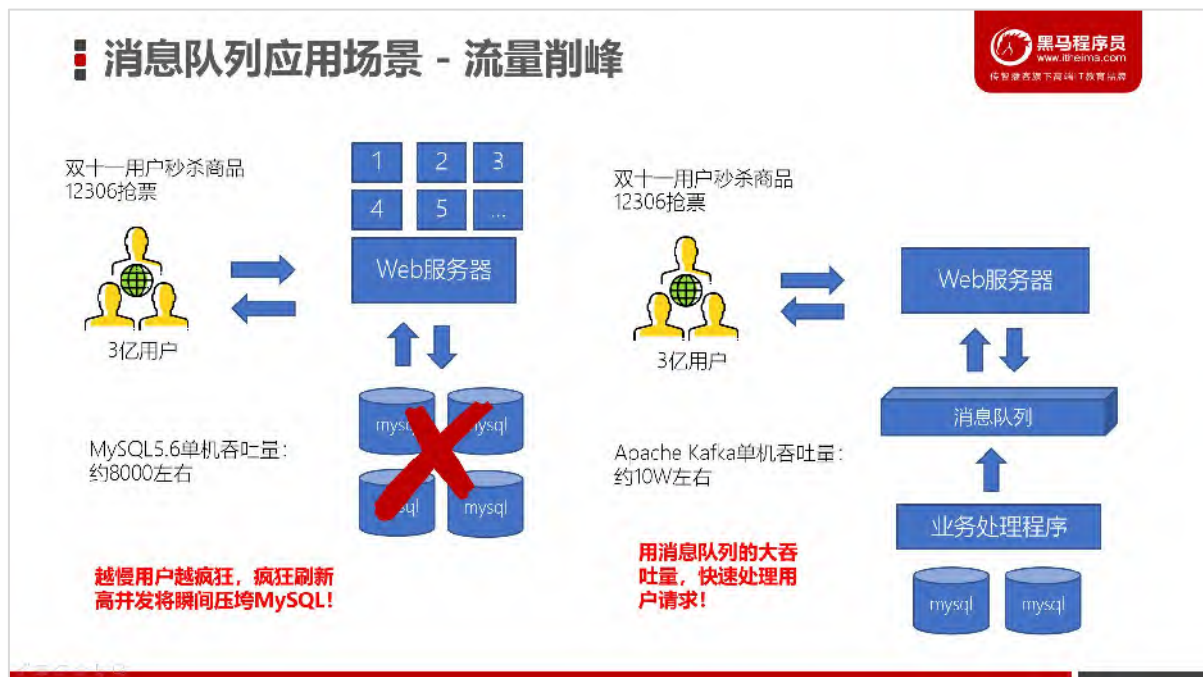


1.1.3.2 系统解耦

消息队列应用场景 - 系统解耦

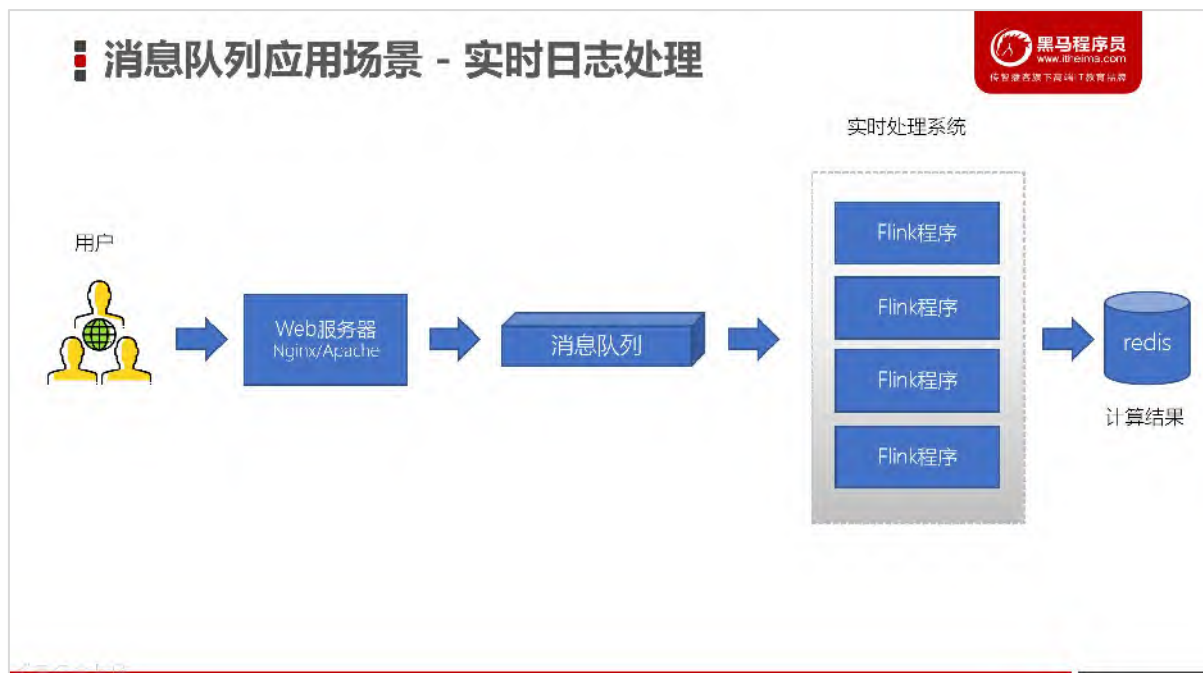


1.1.3.3 流量削峰



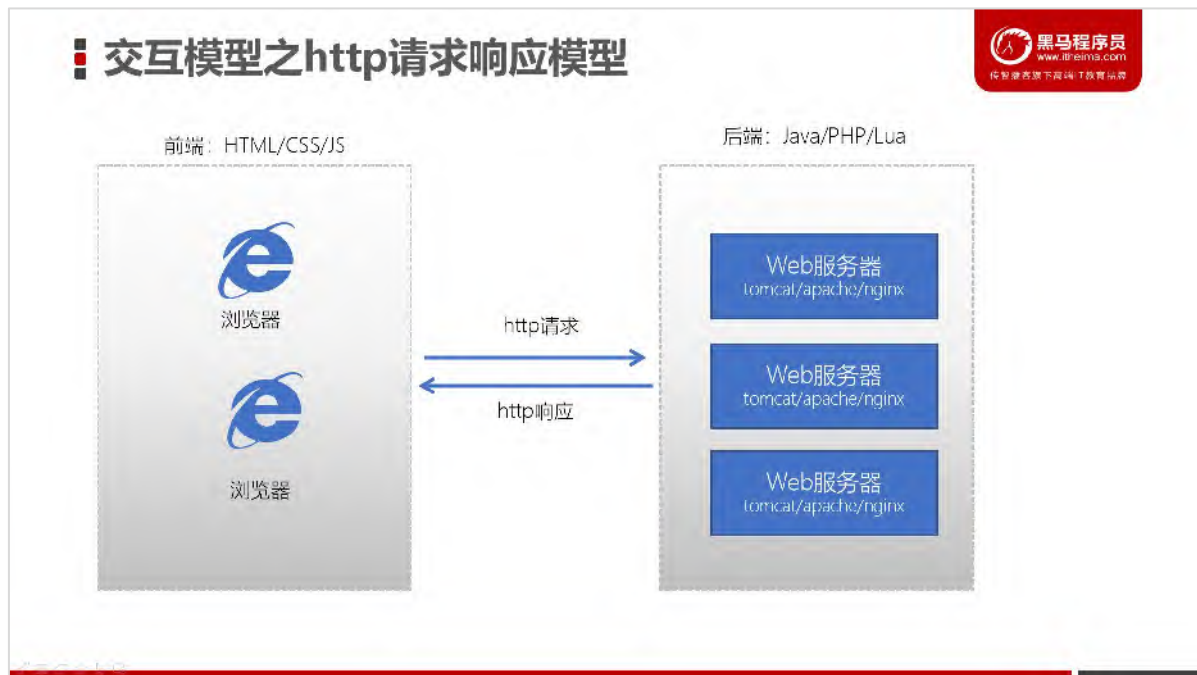
1.1.3.4 日志处理 (大数据领域常见)

大型电商网站（淘宝、京东、国美、苏宁...）、App（抖音、美团、滴滴等）等需要分析用户行为，要根据用户的访问行为来发现用户的喜好以及活跃情况，需要在页面上收集大量的用户访问信息。

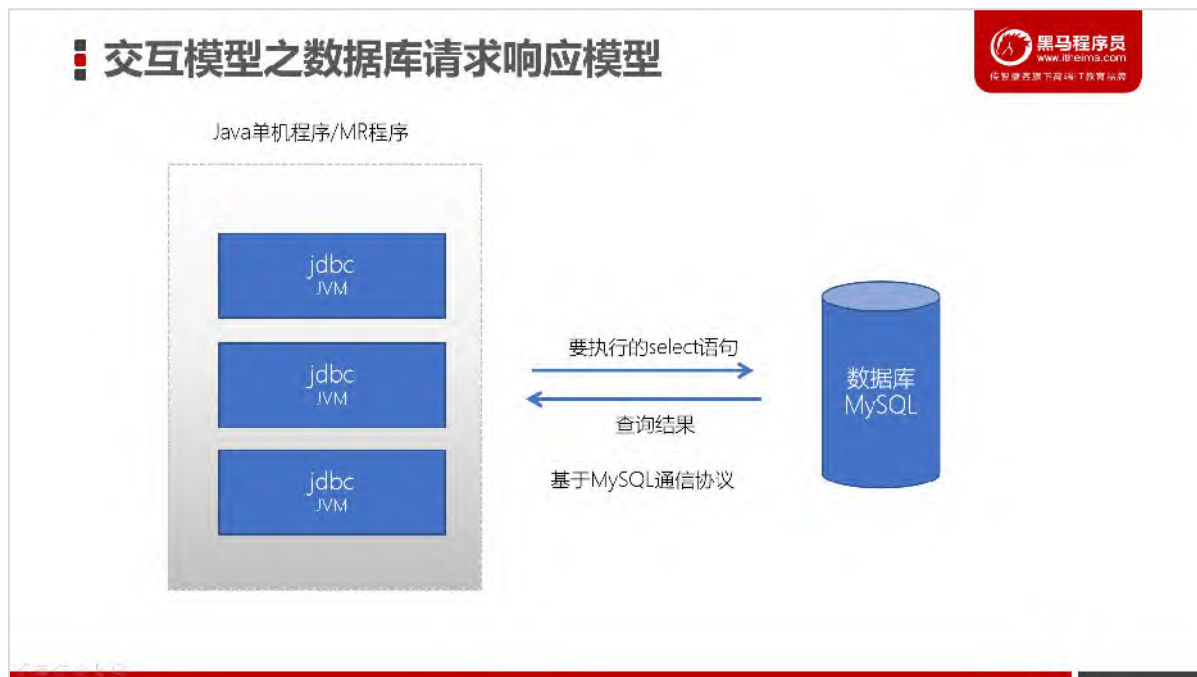


1.1.4 生产者、消费者模型

我们之前学习过Java的服务器开发，Java服务器端开发的交互模型是这样的：

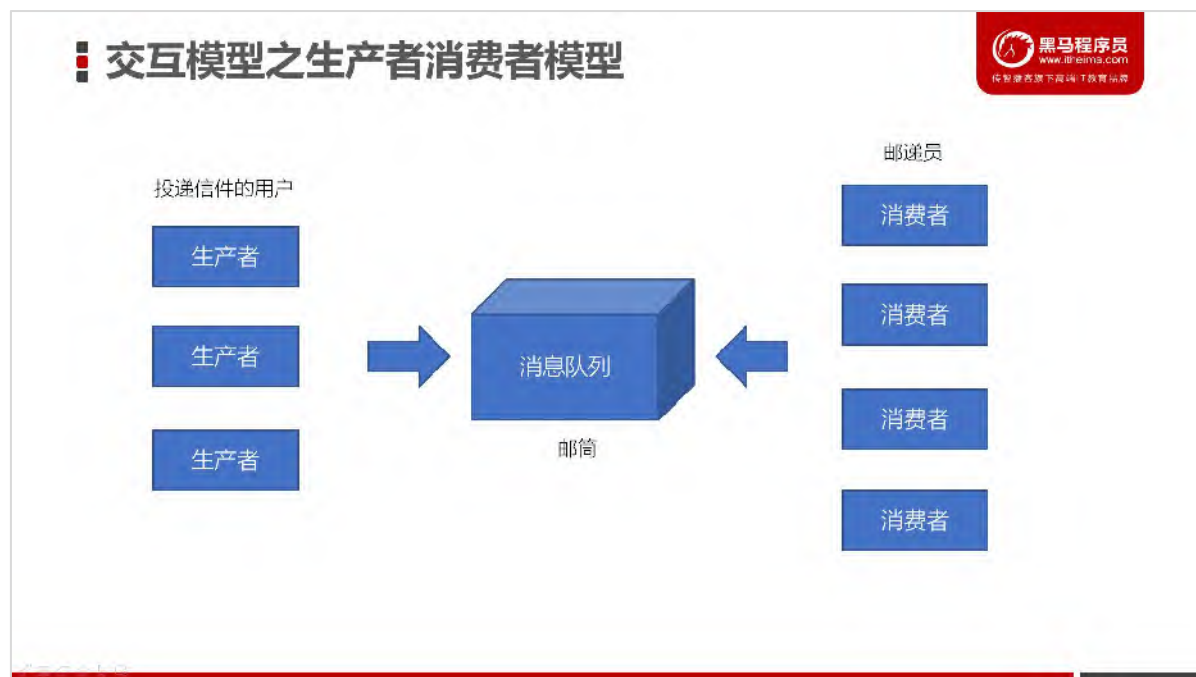


我们之前也学习过使用Java JDBC来访问操作MySQL数据库，它的交互模型是这样的：



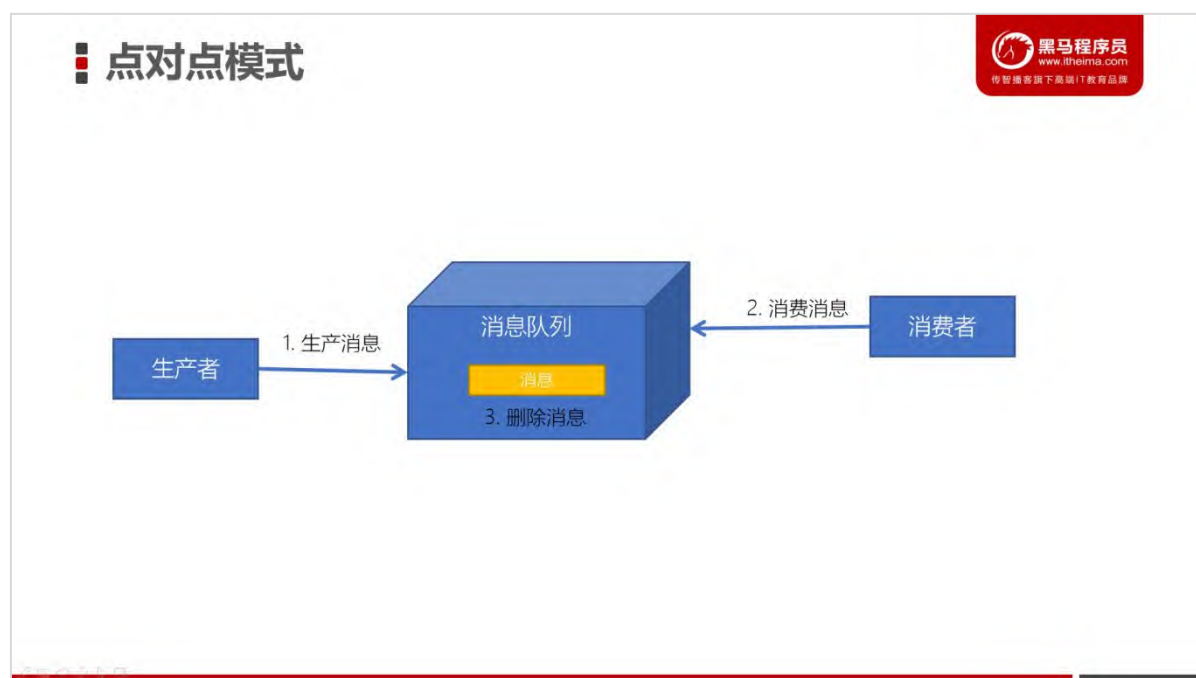
它也是一种请求响应模型，只不过它不再是基于http协议，而是基于MySQL数据库的通信协议。

而如果我们基于消息队列来编程，此时的交互模式成为：生产者、消费者模型。



1.1.5 消息队列的两种模式

1.1.5.1 点对点模式

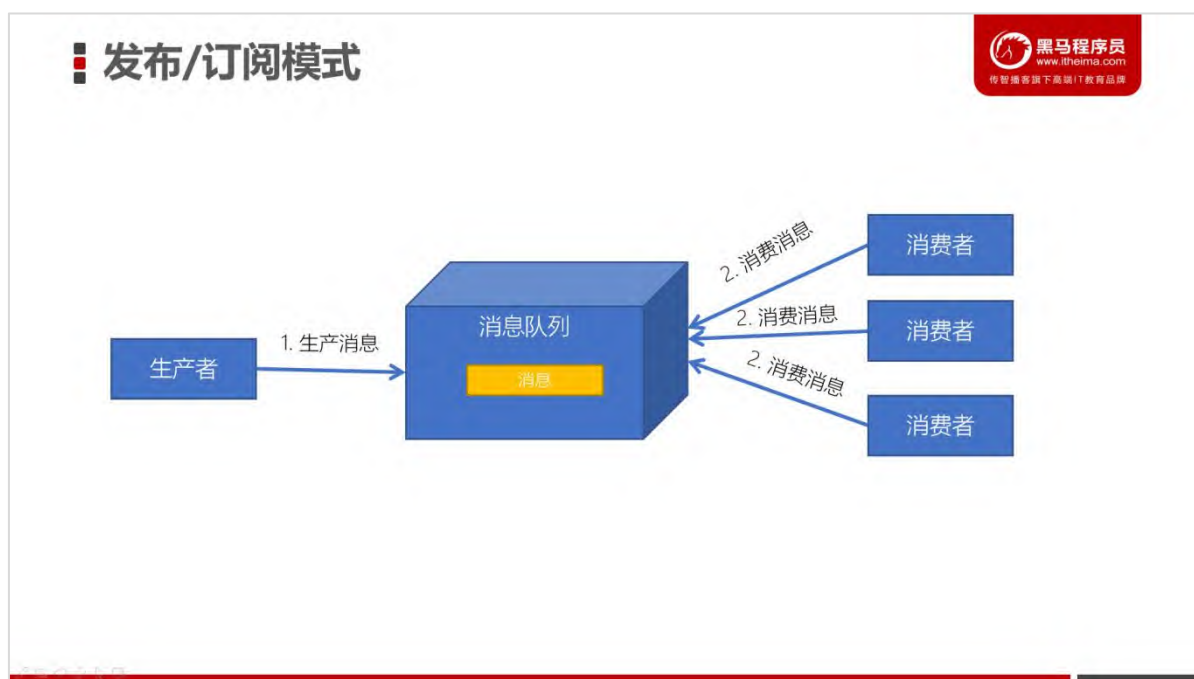


消息发送者生产消息发送到消息队列中，然后消息接收者从消息队列中取出并且消费消息。消息被消费以后，消息队列中不再有存储，所以消息接收者不可能消费到已经被消费的消息。

点对点模式特点：

- 每个消息只有一个接收者（Consumer）（即一旦被消费，消息就不再在消息队列中）
- 发送者和接收者间没有依赖性，发送者发送消息之后，不管有没有接收者在运行，都不会影响到发送者下次发送消息；
- 接收者在成功接收消息之后需向队列应答成功，以便消息队列删除当前接收的消息；

1.1.5.2 发布订阅模式



发布/订阅模式特点：

- 每个消息可以有多个订阅者；
- 发布者和订阅者之间有时间上的依赖性。针对某个主题（Topic）的订阅者，它必须创建一个订阅者之后，才能消费发布者的消息。
- 为了消费消息，订阅者需要提前订阅该角色主题，并保持在线运行；

1.2 Kafka简介

1.2.1 什么是Kafka



Kafka是由Apache软件基金会开发的一个开源流平台，由Scala和Java编写。Kafka的Apache官网是这样介绍Kafka的。

Apache Kafka是一个分布式流平台。一个分布式的流平台应该包含3点关键的能力：

1. 发布和订阅流数据流，类似于消息队列或者是企业消息传递系统
2. 以容错的持久化方式存储数据流
3. 处理数据流

英文原版

- **Publish and subscribe** to streams of records, similar to a message queue or enterprise messaging system.
- **Store** streams of records in a fault-tolerant durable way.
- **Process** streams of records as they occur.

更多参考：<http://kafka.apache.org/documentation/#introduction>

我们重点关键三个部分的关键词：

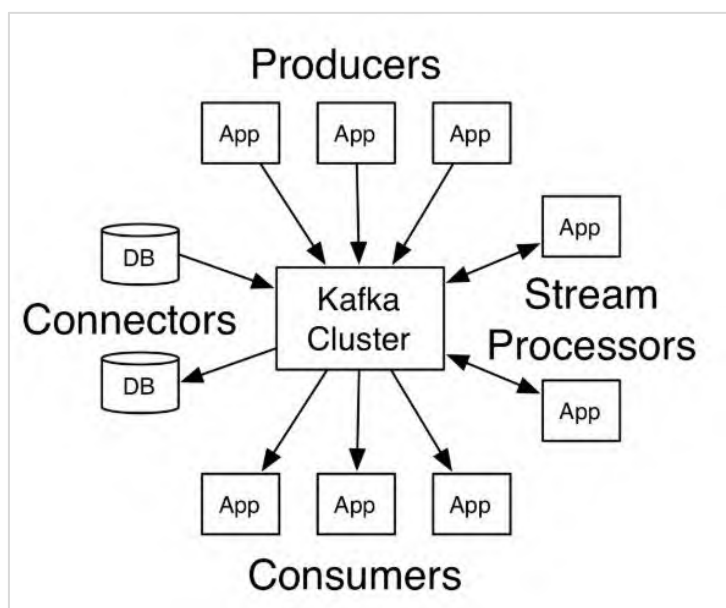
1. Publish and subscribe：发布与订阅
2. Store：存储
3. Process：处理

后续我们的课程主要围绕这三点来讲解。

1.2.2 Kafka的应用场景

我们通常将Apache Kafka用在两类程序：

1. 建立实时数据管道，以可靠地在系统或应用程序之间获取数据
2. 构建实时流应用程序，以转换或响应数据流



上图，我们可以看到：

1. Producers：可以有很多的应用程序，将消息数据放入到Kafka集群中。
2. Consumers：可以有很多的应用程序，将消息数据从Kafka集群中拉取出来。
3. Connectors：Kafka的连接器可以将数据库中的数据导入到Kafka，也可以将Kafka的数据导出到数据库中。
4. Stream Processors：流处理器可以从Kafka中拉取数据，也可以将数据写入到Kafka中。

1.2.3 Kafka诞生背景

kafka的诞生，是为了解决linkedin的数据管道问题，起初linkedin采用了ActiveMQ来进行数据交换，大约是在2010年前后，那时的ActiveMQ还远远无法满足linkedin对数据传递系统的要求，经常由于各种缺陷而导致消息阻塞或者服务无法正常访问，为了能够解决这个问题，linkedin决定研发自己的消息传递系统，当时linkedin的首席架构师jay kreps便开始组织团队进行消息传递系统的研发。

提示：

1. LinkedIn还是挺牛逼的
2. Kafka比ActiveMQ牛逼得多

1.3 Kafka的优势


前面我们了解到，消息队列中间件有很多，为什么我们要选择Kafka？

特性	ActiveMQ	RabbitMQ	Kafka	RocketMQ
所属社区/公司	Apache	Mozilla Public License	Apache	Apache/Ali
成熟度	成熟	成熟	成熟	比较成熟
生产者-消费者模式	支持	支持	支持	支持
发布-订阅	支持	支持	支持	支持
REQUEST-REPLY	支持	支持	-	支持
API完备性	高	高	高	低(静态配置)
多语言支持	支持JAVA优先	语言无关	支持，JAVA优先	支持
单机吞吐量	万级(最差)	万级	十万级	十万级(最高)
消息延迟	-	微秒级	毫秒级	-
可用性	高(主从)	高(主从)	非常高(分布式)	高
消息丢失	-	低	理论上不会丢失	-
消息重复	-	可控制	理论上会有重复	-
事务	支持	不支持	支持	支持
文档的完备性	高	高	高	中

提供快速入门	有	有	有	无
首次部署难度	-	低	中	高

在大数据技术领域，一些重要的组件、框架都支持Apache Kafka，不论成熟度、社区、性能、可靠性，Kafka都是非常有竞争力的一款产品。

1.4 哪些公司在使用Kafka

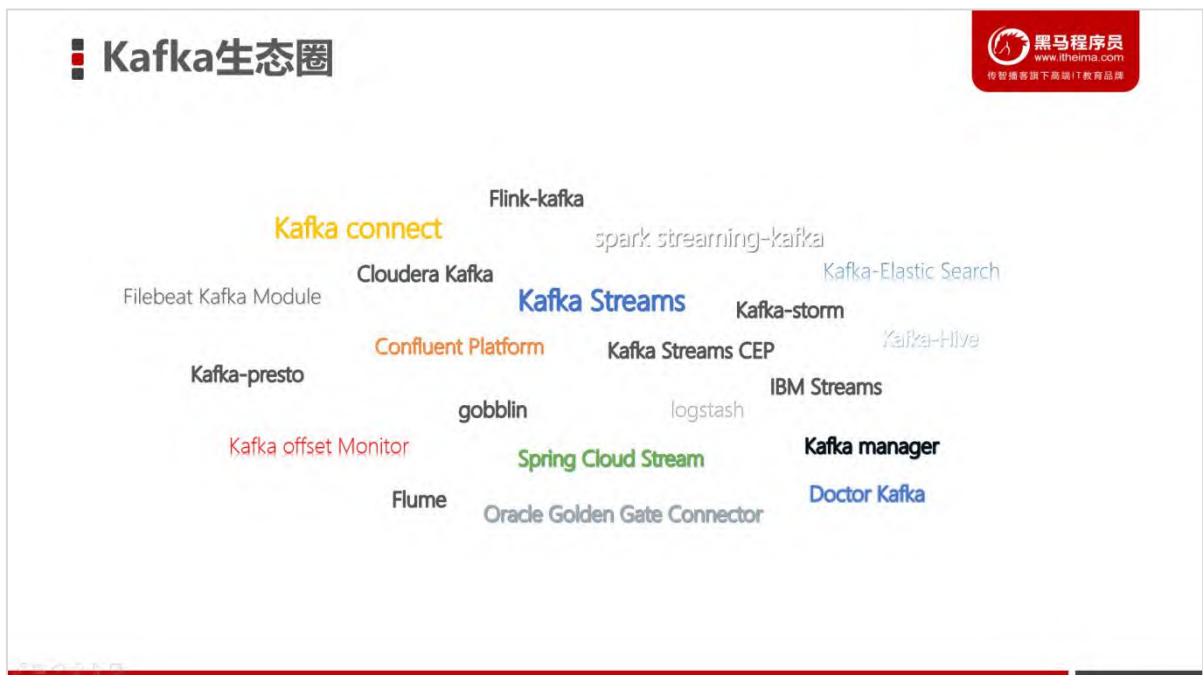
 <small>Google</small>	 <small>Tencent</small>	 <small>Facebook</small>
 <small>Pineapple Fund</small>	 <small>Microsoft</small>	 <small>Amazon Web Services</small>
 <small>Comcast</small>	 <small>Cloudera</small>	
 <small>LeaseWeb</small> <small>reliable hosting</small>	 <small>ARM</small>	 <small>Bloomberg</small>



1.5 Kafka生态圈介绍

Apache Kafka这么多年的发展，目前也有一个较庞大的生态圈。

Kafka生态圈官网地址：<https://cwiki.apache.org/confluence/display/KAFKA/Ecosystem>



1.6 Kafka版本

本次课程使用的Kafka版本为2.4.1，是2020年3月12日发布的版本。

可以注意到Kafka的版本号为：kafka_2.12-2.4.1，因为kafka主要是使用scala语言开发的，2.12为scala的版本号。<http://kafka.apache.org/downloads>可以查看到每个版本的发布时间。

2. 环境搭建

2.1 搭建Kafka集群

1. 将Kafka的安装包上传到虚拟机，并解压

```
cd /export/software/  
tar -xvzf kafka_2.12-2.4.1.tgz -C ../server/  
cd /export/server/kafka_2.12-2.4.1/
```

2. 修改 server.properties

```
cd /export/server/kafka_2.12-2.4.1/config  
vim server.properties  
  
# 指定broker的id  
broker.id=0  
  
# 指定Kafka数据的位置  
log.dirs=/export/server/kafka_2.12-2.4.1/data  
  
# 配置zk的三个节点  
zookeeper.connect=node1.itcast.cn:2181,node2.itcast.cn:2181,node3.itcast.cn:2181
```

3. 将安装好的kafka复制到另外两台服务器

```
cd /export/server  
scp -r kafka_2.12-2.4.1/ node2.itcast.cn:$PWD  
scp -r kafka_2.12-2.4.1/ node3.itcast.cn:$PWD
```

修改另外两个节点的broker.id分别为1和2

```
-----node2.itcast.cn-----  
cd /export/server/kafka_2.12-2.4.1/config  
vim server.properties  
broker.id=1
```



```
-----node3.itcast.cn-----  
cd /export/server/kafka_2.12-2.4.1/config  
vim server.properties  
broker.id=2
```

4. 配置KAFKA_HOME环境变量

```
vim /etc/profile  
export KAFKA_HOME=/export/server/kafka_2.12-2.4.1  
export PATH=$PATH:${KAFKA_HOME}
```

分发到各个节点

```
scp /etc/profile node2.itcast.cn:$PWD  
scp /etc/profile node3.itcast.cn:$PWD
```

每个节点加载环境变量

```
source /etc/profile
```

5. 启动服务器

```
# 启动ZooKeeper  
nohup bin/zookeeper-server-start.sh config/zookeeper.properties &  
  
# 启动Kafka  
cd /export/server/kafka_2.12-2.4.1  
nohup bin/kafka-server-start.sh config/server.properties &  
  
# 测试Kafka集群是否启动成功  
bin/kafka-topics.sh --bootstrap-server node1.itcast.cn:9092 --list
```

2.2 目录结构分析

目录名称	说明
bin	Kafka的所有执行脚本都在这里。例如：启动Kafka服务器、创建Topic、生产者、消费者程序等等
config	Kafka的所有配置文件

libs	运行Kafka所需要的所有JAR包
logs	Kafka的所有日志文件，如果Kafka出现一些问题，需要到该目录中去查看异常信息
site-docs	Kafka的网站帮助文件

2.3 Kafka一键启动/关闭脚本

为了方便将来进行一键启动、关闭Kafka，我们可以编写一个shell脚本来操作。将来只要执行一次该脚本就可以快速启动/关闭Kafka。

1. 在节点1中创建 /export/onekey 目录

```
cd /export/onekey
```

2. 准备slave配置文件，用于保存要启动哪几个节点上的kafka

```
node1.itcast.cn
node2.itcast.cn
node3.itcast.cn
```

3. 编写start-kafka.sh脚本

```
vim start-kafka.sh
cat /export/onekey/slave | while read line
do
{
    echo $line
    ssh $line "source /etc/profile;export JMX_PORT=9988;nohup
${KAFKA_HOME}/bin/kafka-server-start.sh ${KAFKA_HOME}/config/server.properties >/dev/nul*
2>&1 & "
}&
wait
done
```

4. 编写stop-kafka.sh脚本

```
vim stop-kafka.sh
cat /export/onekey/slave | while read line
do
{
    echo $line
    ssh $line "source /etc/profile;jps |grep Kafka |cut -d' ' -f1 |xargs kill -s 9"
}&
wait
done
```

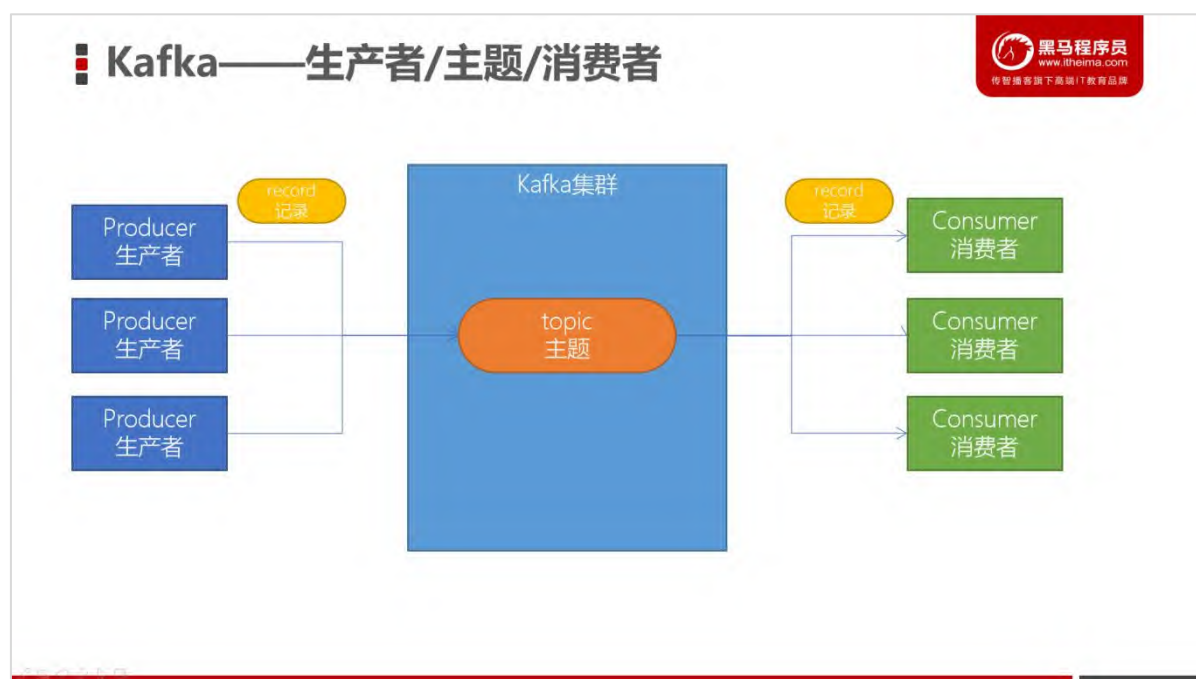
5. 给start-kafka.sh、stop-kafka.sh配置执行权限

```
chmod u+x start-kafka.sh
chmod u+x stop-kafka.sh
```

6. 执行一键启动、一键关闭

```
./start-kafka.sh
./stop-kafka.sh
```

3. 基础操作



3.1 创建topic

创建一个topic（主题）。Kafka中所有的消息都是保存在主题中，要生产消息到Kafka，首先必须要有一个确定的主题。

```
# 创建名为test的主题
bin/kafka-topics.sh --create --bootstrap-server node1.itcast.cn:9092 --topic test

# 查看目前Kafka中的主题
bin/kafka-topics.sh --list --bootstrap-server node1.itcast.cn:9092
```

3.2 生产消息到Kafka

使用Kafka内置的测试程序，生产一些消息到Kafka的test主题中。

```
bin/kafka-console-producer.sh --broker-list node1.itcast.cn:9092 --topic test
```

3.3 从Kafka消费消息

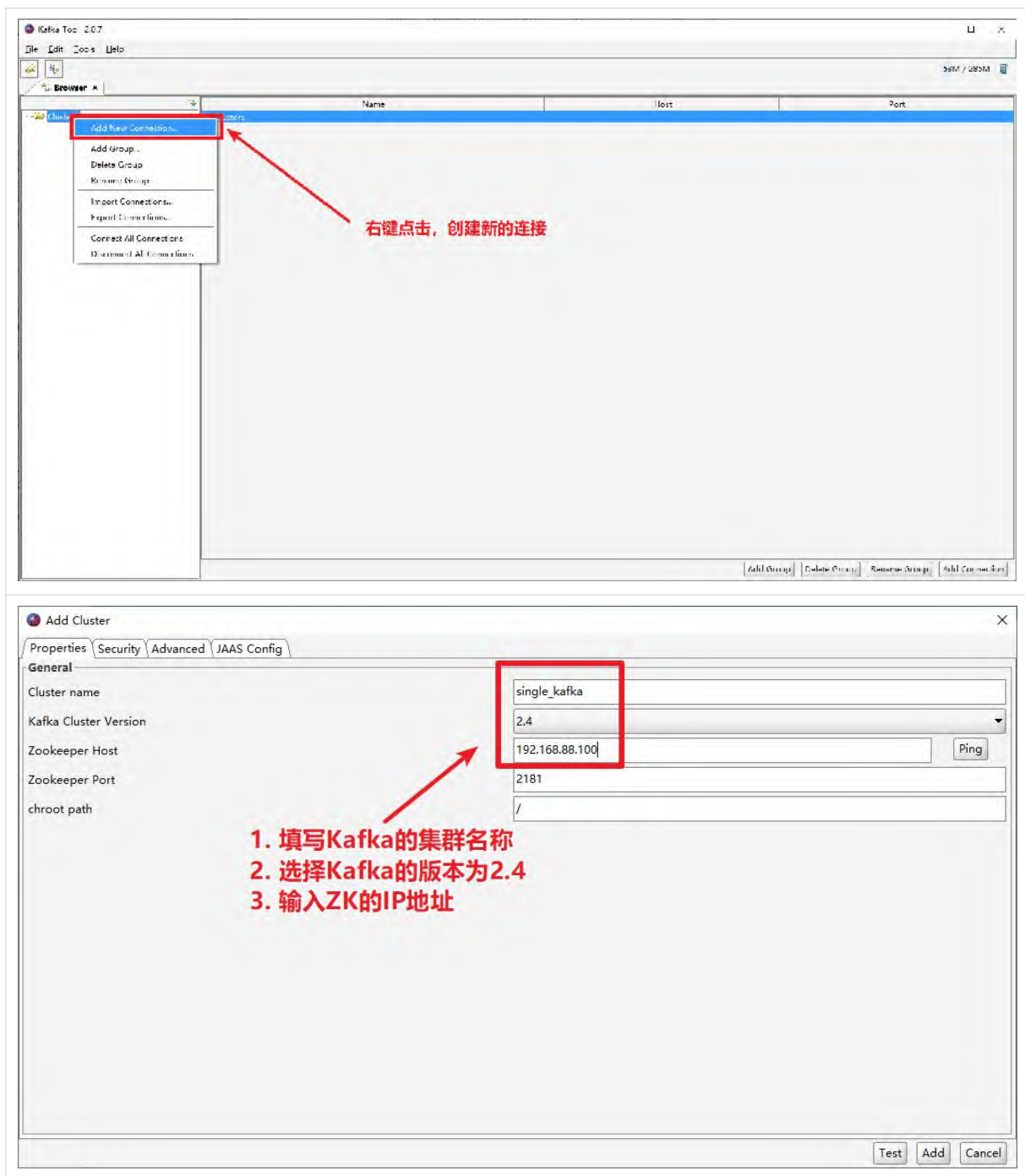
使用下面的命令来消费 test 主题中的消息。

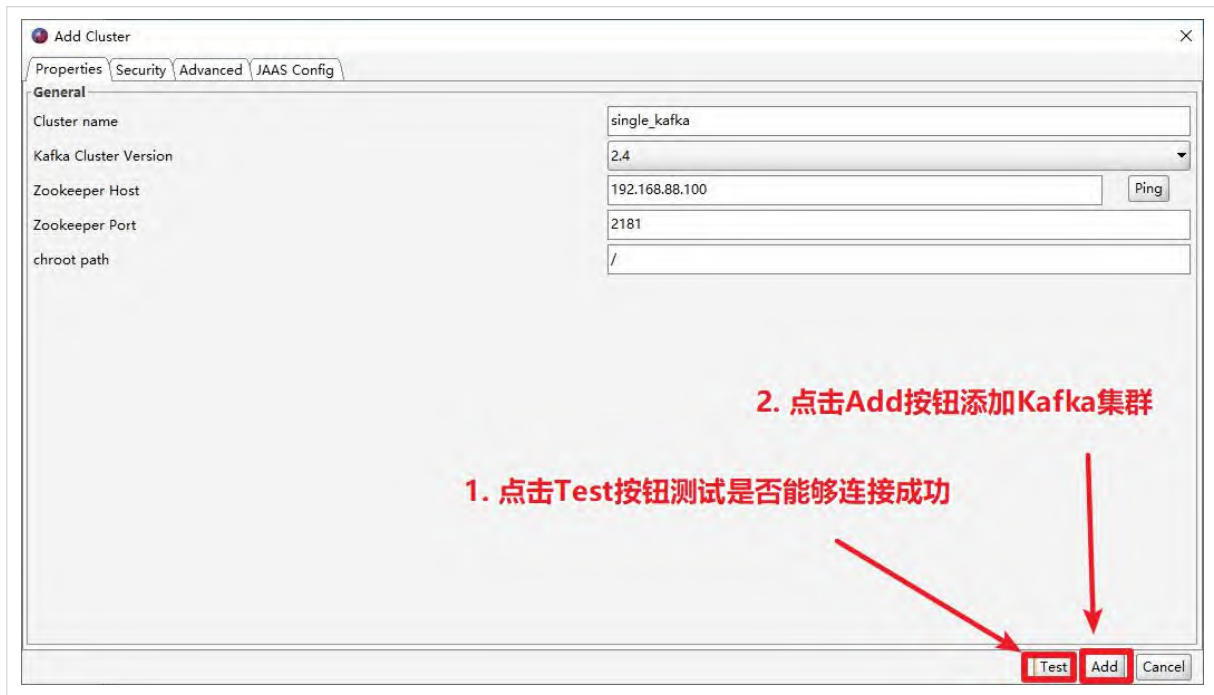
```
bin/kafka-console-consumer.sh --bootstrap-server node1.itcast.cn:9092 --topic test
--from-beginning
```

3.4 使用Kafka Tools操作Kafka

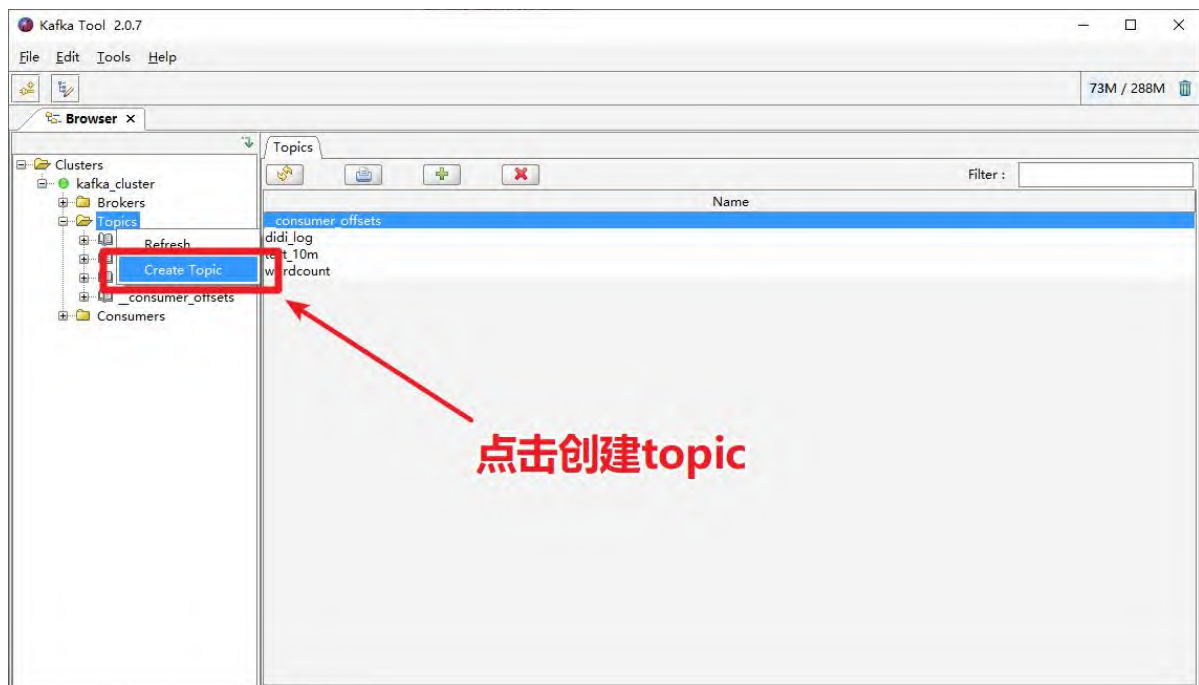
3.4.1 连接Kafka集群

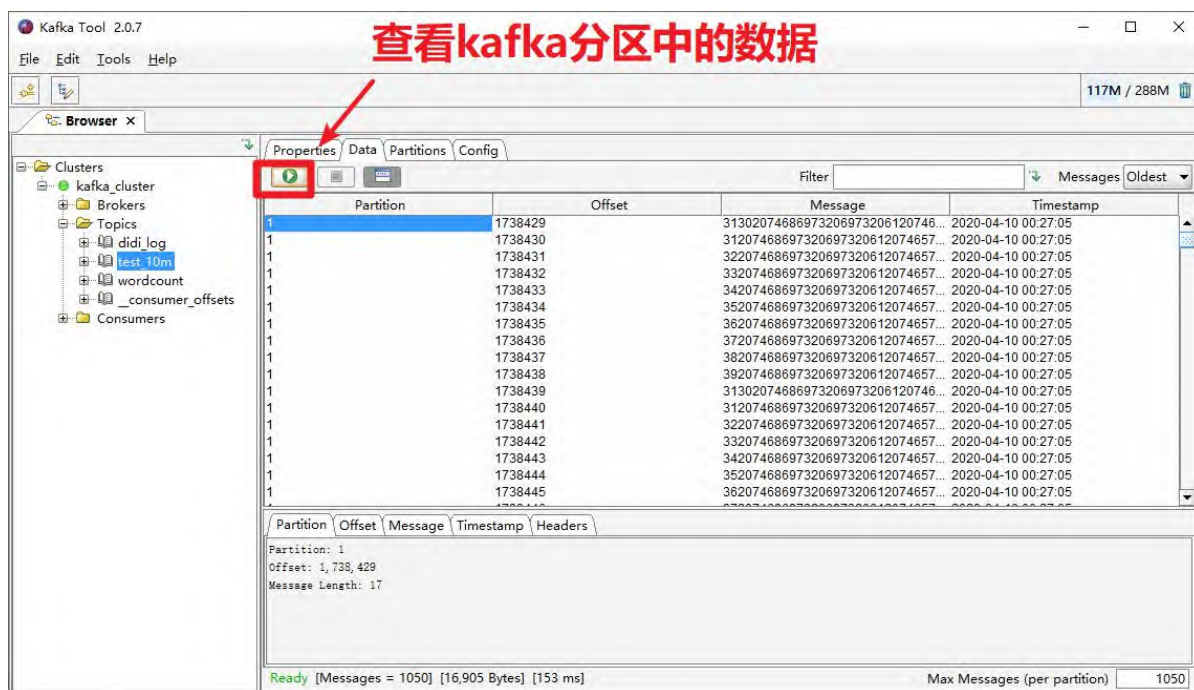
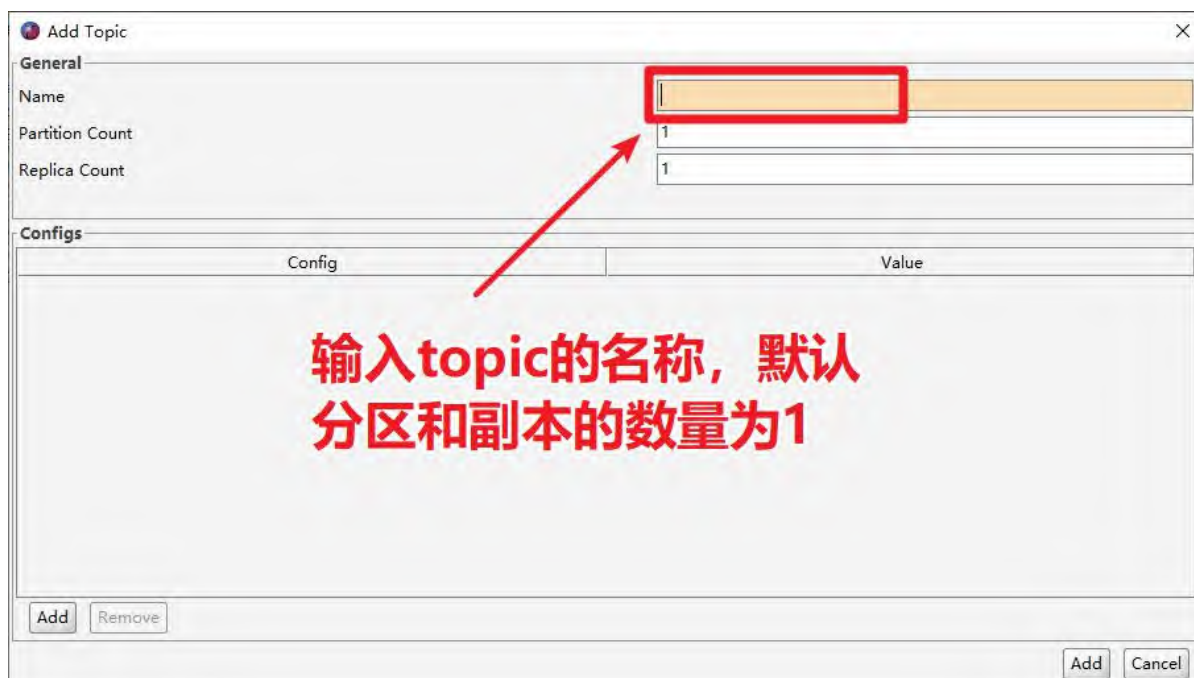
安装Kafka Tools后启动Kafka





3.4.2 创建topic





4. Kafka基准测试

4.1 基准测试

基准测试 (benchmark testing) 是一种测量和评估软件性能指标的活动。我们可以通过基准测试，了解到软件、硬件的性能水平。主要测试负载的执行时间、传输速度、吞吐量、资源占用率等。

4.1.1 基于1个分区1个副本的基准测试

测试步骤：

1. 启动Kafka集群
2. 创建一个1个分区1个副本的topic: benchmark
3. 同时运行生产者、消费者基准测试程序
4. 观察结果

4.1.1.1 创建topic

```
bin/kafka-topics.sh --zookeeper node1.itcast.cn:2181 --create --topic benchmark
--partitions 1 --replication-factor 1
```

4.1.1.2 生产消息基准测试

在生产环境中，推荐使用生产5000W消息，这样会性能数据会更准确些。为了方便测试，课程上演示测试500W的消息作为基准测试。

```
bin/kafka-producer-perf-test.sh --topic benchmark --num-records 5000000 --throughput -1
--record-size 1000 --producer-props
bootstrap.servers=node1.itcast.cn:9092,node2.itcast.cn:9092,node3.itcast.cn:9092 acks=1
```

```
bin/kafka-producer-perf-test.sh
--topic topic的名字
--num-records 总共指定生产数据量（默认5000W）
--throughput 指定吞吐量——限流（-1不指定）
--record-size record数据大小（字节）
--producer-props bootstrap.servers=192.168.1.20:9092,192.168.1.21:9092,192.168.1.22:9092
acks=1 指定Kafka集群地址，ACK模式
```

测试结果：

吞吐量	93092.533979 records/sec
-----	--------------------------

	每秒9.3W条记录
吞吐速率	(88.78 MB/sec) 每秒约89MB数据
平均延迟时间	346.62 ms avg latency
最大延迟时间	1003.00 ms max latency

4.1.1.3 消费消息基准测试

```
bin/kafka-consumer-perf-test.sh --broker-list node1.itcast.cn:9092,node2.itcast.cn:9092,node3.itcast.cn:9092 --topic benchmark --fetch-size 1048576 --messages 5000000
```

```
bin/kafka-consumer-perf-test.sh
--broker-list 指定kafka集群地址
--topic 指定topic的名称
--fetch-size 每次拉取的数据大小
--messages 总共要消费的消息个数
```

data.consumed.in.MB 共计消费的数据	4768.3716MB
MB.sec 每秒消费的数量	445.6006 每秒445MB
data.consumed.in.nMsg 共计消费的数量	5000000
nMsg.sec 每秒的数量	467246.0518 每秒46.7W条

4.1.2 基于3个分区1个副本的基准测试

被测虚拟机：

node1.itcast.cn	node2.itcast.cn	node3.itcast.cn
inter i5 8 th 8G内存	inter i5 8 th 4G内存	inter i5 8 th 4G内存

4.1.2.1 创建topic

```
bin/kafka-topics.sh --zookeeper node1.itcast.cn:2181 --create --topic benchmark
--partitions 3 --replication-factor 1
```

4.1.2.2 生产消息基准测试

```
bin/kafka-producer-perf-test.sh --topic benchmark --num-records 5000000 --throughput -1
--record-size 1000 --producer-props bootstrap.servers=node1.itcast.cn:9092,node2.itcast.cn:9092,node3.itcast.cn:9092 acks=1
```

测试结果：

指标	3分区1个副本	单分区单副本
吞吐量	68755.930199 records/sec	93092.533979 records/sec 每秒9.3W条记录
吞吐速率	65.57 MB/sec	(88.78 MB/sec) 每秒约89MB数据
平均延迟时间	469.37 ms avg latency	346.62 ms avg latency
最大延迟时间	2274.00 ms max latency	1003.00 ms max latency

在虚拟机上，因为都是共享笔记本上的CPU、内存、网络，所以分区越多，反而效率越低。但如果是真实的服务器，分区多效率是会有明显提升的。

4.1.2.3 消费消息基准测试

```
bin/kafka-consumer-perf-test.sh --broker-list node1.itcast.cn:9092,node2.itcast.cn:9092,node3.itcast.cn:9092 --topic benchmark
--fetch-size 1048576 --messages 5000000
```


指标	单分区3个副本	单分区单副本
data.consumed.in.MB 共计消费的数据	4768.3716MB	4768.3716MB
MB.sec 每秒消费的数量	265.8844MB	445.6006 每秒445MB
data.consumed.in.nMsg 共计消费的数量	5000000	5000000
nMsg.sec 每秒的数量	278800.0446 每秒27.8W	467246.0518 每秒46.7W

还是一样，因为虚拟机的原因，多个分区反而消费的效率也有所下降。

4.1.3 基于1个分区3个副本的基准测试

4.1.3.1 创建topic

```
bin/kafka-topics.sh --zookeeper node1.itcast.cn:2181 --create --topic benchmark --partitions 1 --replication-factor 3
```

4.1.3.2 生产消息基准测试

```
bin/kafka-producer-perf-test.sh --topic benchmark --num-records 5000000 --throughput -1 --record-size 1000 --producer-props bootstrap.servers=node1.itcast.cn:9092,node2.itcast.cn:9092,node3.itcast.cn:9092 acks=1
```

测试结果：

指标	单分区3个副本	单分区单副本
吞吐量	29899.477955 records/sec	93092.533979 records/sec 每秒9.3W条记录
吞吐速率	28.51 MB/sec	(88.78 MB/sec) 每秒约89MB数据

平均延迟时间	1088.43 ms avg latency	346.62 ms avg latency
最大延迟时间	2416.00 ms max latency	1003.00 ms max latency

同样的配置，副本越多速度越慢。

4.1.3.3 消费消息基准测试

```
bin/kafka-consumer-perf-test.sh --broker-list node1.itcast.cn:9092,node2.itcast.cn:9092,node3.itcast.cn:9092 --topic benchmark --fetch-size 1048576 --messages 5000000
```

指标	单分区3个副本	单分区单副本
data.consumed.in.MB 共计消费的数据	4768.3716MB	4768.3716MB
MB.sec 每秒消费的数量	265.8844MB 每秒265MB	445.6006 每秒445MB
data.consumed.in.nMsg 共计消费的数量	5000000	5000000
nMsg.sec 每秒的数量	278800.0446 每秒27.8W	467246.0518 每秒46.7W

5. Java编程操作Kafka

5.1 同步生产消息到Kafka中

5.1.1 需求

接下来，我们将编写Java程序，将1-100的数字消息写入到Kafka中。

5.1.2 准备工作

5.1.2.1 导入Maven Kafka POM依赖

```
<repositories><!-- 代码库 -->
<repository>
  <id>central</id>
  <url>http://maven.aliyun.com/nexus/content/groups/public//</url>
  <releases>
    <enabled>true</enabled>
  </releases>
  <snapshots>
    <enabled>true</enabled>
    <updatePolicy>always</updatePolicy>
    <checksumPolicy>fail</checksumPolicy>
  </snapshots>
</repository>
</repositories>

<dependencies>
  <!-- kafka 客户端工具 -->
  <dependency>
    <groupId>org.apache.kafka</groupId>
    <artifactId>kafka-clients</artifactId>
    <version>2.4.1</version>
  </dependency>

  <!-- 工具类 -->
  <dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-io</artifactId>
    <version>1.3.2</version>
  </dependency>

  <!-- SLF 桥接 LOG4J 日志 -->
  <dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-log4j12</artifactId>
    <version>1.7.6</version>
  </dependency>

  <!-- SLOG4J 日志 -->
  <dependency>
```

```
<groupId>log4j</groupId>
<artifactId>log4j</artifactId>
<version>1.2.16</version>
</dependency>
</dependencies>

<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.7.0</version>
      <configuration>
        <source>1.8</source>
        <target>1.8</target>
      </configuration>
    </plugin>
  </plugins>
</build>
```

5.1.2.2 导入log4j.properties

将log4j.properties配置文件放入到resources文件夹中

```
log4j.rootLogger=INFO,stdout
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%5p - %m%n
```

5.1.2.3 创建包和类

创建包cn.itcast.kafka，并创建KafkaProducerTest类。

5.1.3 代码开发

可以参考以下方式来编写第一个Kafka示例程序

参考以下文档：

<http://kafka.apache.org/24/javadoc/index.html?org/apache/kafka/clients/producer/KafkaProducer.html>

1. 创建用于连接Kafka的Properties配置

```
Properties props = new Properties();
props.put("bootstrap.servers", "192.168.88.100:9092");
props.put("acks", "all");
props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");
```

2. 创建一个生产者对象KafkaProducer

3. 调用send发送1-100消息到指定Topic test，并获取返回值Future，该对象封装了返回值

4. 再调用一个Future.get()方法等待响应

5. 关闭生产者

参考代码：

```
public class KafkaProducerTest {
    public static void main(String[] args) {
        // 1. 创建用于连接 Kafka 的 Properties 配置
        Properties props = new Properties();
        props.put("bootstrap.servers", "192.168.88.100:9092");
        props.put("acks", "all");
        props.put("key.serializer",
            "org.apache.kafka.common.serialization.StringSerializer");
        props.put("value.serializer",
            "org.apache.kafka.common.serialization.StringSerializer");

        // 2. 创建一个生产者对象 KafkaProducer
        KafkaProducer<String, String> producer = new KafkaProducer<String, String>(props);

        // 3. 调用 send 发送 1-100 消息到指定 Topic test
        for(int i = 0; i < 100; ++i) {
            try {
                // 获取返回值 Future，该对象封装了返回值
                Future<RecordMetadata> future = producer.send(new ProducerRecord<String,
                    String>("test", null, i + ""));
                // 调用一个 Future.get()方法等待响应
                future.get();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```



```
        } catch (ExecutionException e) {  
            e.printStackTrace();  
        }  
    }  
  
    // 5. 关闭生产者  
    producer.close();  
}  
}
```

5.2 从Kafka的topic中消费消息

5.2.1 需求

从 test topic中，将消息都消费，并将记录的offset、key、value打印出来

5.2.2 准备工作

在cn.itcast.kafka包下创建KafkaConsumerTest类

5.2.3 开发步骤

1. 创建Kafka消费者配置

```
Properties props = new Properties();  
props.setProperty("bootstrap.servers", "node1.itcast.cn:9092");  
props.setProperty("group.id", "test");  
props.setProperty("enable.auto.commit", "true");  
props.setProperty("auto.commit.interval.ms", "1000");  
props.setProperty("key.deserializer",  
    "org.apache.kafka.common.serialization.StringDeserializer");  
props.setProperty("value.deserializer",  
    "org.apache.kafka.common.serialization.StringDeserializer");
```

2. 创建Kafka消费者

3. 订阅要消费的主题

4. 使用一个while循环，不断从Kafka的topic中拉取消息

5. 将记录 (record) 的 offset、key、value 都打印出来

5.2.4 参考代码

```
public class KafkaProducerTest {
    public static void main(String[] args) {
        // 1. 创建用于连接 Kafka 的 Properties 配置
        Properties props = new Properties();
        props.put("bootstrap.servers", "node1.itcast.cn:9092");
        props.put("acks", "all");
        props.put("key.serializer",
            "org.apache.kafka.common.serialization.StringSerializer");
        props.put("value.serializer",
            "org.apache.kafka.common.serialization.StringSerializer");

        // 2. 创建一个生产者对象 KafkaProducer
        KafkaProducer<String, String> producer = new KafkaProducer<String, String>(props);

        // 3. 调用 send 发送 1-100 消息到指定 Topic test
        for(int i = 0; i < 100; ++i) {
            try {
                // 获取返回值 Future，该对象封装了返回值
                Future<RecordMetadata> future = producer.send(new ProducerRecord<String,
                    String>("test", null, i + ""));
                // 调用一个 Future.get() 方法等待响应
                future.get();
            } catch (InterruptedException e) {
                e.printStackTrace();
            } catch (ExecutionException e) {
                e.printStackTrace();
            }
        }

        // 5. 关闭生产者
        producer.close();
    }
}
```

参考官网API文档：

<http://kafka.apache.org/24/javadoc/index.html?org/apache/kafka/clients/consumer/KafkaConsumer.html>

5.3 异步使用带有回调函数方法生产消息

如果我们想获取生产者消息是否成功，或者成功生产消息到Kafka中后，执行一些其他动作。此时，可以很方便地使用带有回调函数来发送消息。

需求：

1. 在发送消息出现异常时，能够及时打印出异常信息
2. 在发送消息成功时，打印Kafka的topic名字、分区id、offset

```
public class KafkaProducerTest {
    public static void main(String[] args) {
        // 1. 创建用于连接 Kafka 的 Properties 配置
        Properties props = new Properties();
        props.put("bootstrap.servers", "node1.itcast.cn:9092");
        props.put("acks", "all");
        props.put("key.serializer",
            "org.apache.kafka.common.serialization.StringSerializer");
        props.put("value.serializer",
            "org.apache.kafka.common.serialization.StringSerializer");

        // 2. 创建一个生产者对象 KafkaProducer
        KafkaProducer<String, String> producer = new KafkaProducer<String, String>(props);

        // 3. 调用 send 发送 1-100 消息到指定 Topic test
        for(int i = 0; i < 100; ++i) {
            // 一、同步方式
            // 获取返回值 Future，该对象封装了返回值
            // Future<RecordMetadata> future = producer.send(new ProducerRecord<String,
            String>("test", null, i + ""));
            // 调用一个 Future.get()方法等待响应
            // future.get();

            // 二、带回调函数异步方式
            producer.send(new ProducerRecord<String, String>("test", null, i + ""), new
            Callback() {
```

```
@Override
public void onCompletion(RecordMetadata metadata, Exception exception) {
    if(exception != null) {
        System.out.println("发送消息出现异常");
    }
    else {
        String topic = metadata.topic();
        int partition = metadata.partition();
        long offset = metadata.offset();

        System.out.println("发送消息到 Kafka 中的名字为" + topic + "的主题，第"
            + partition + "分区，第" + offset + "条数据成功!");
    }
}

});

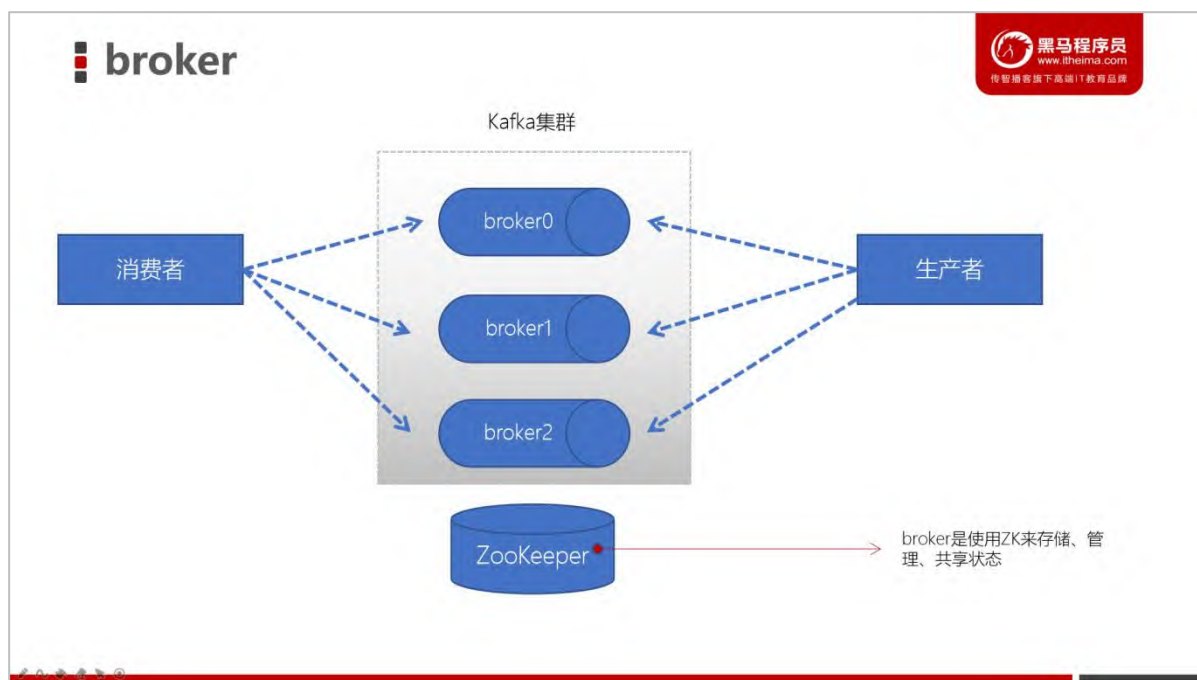
}

// 5. 关闭生产者
producer.close();
}
```

6. 架构

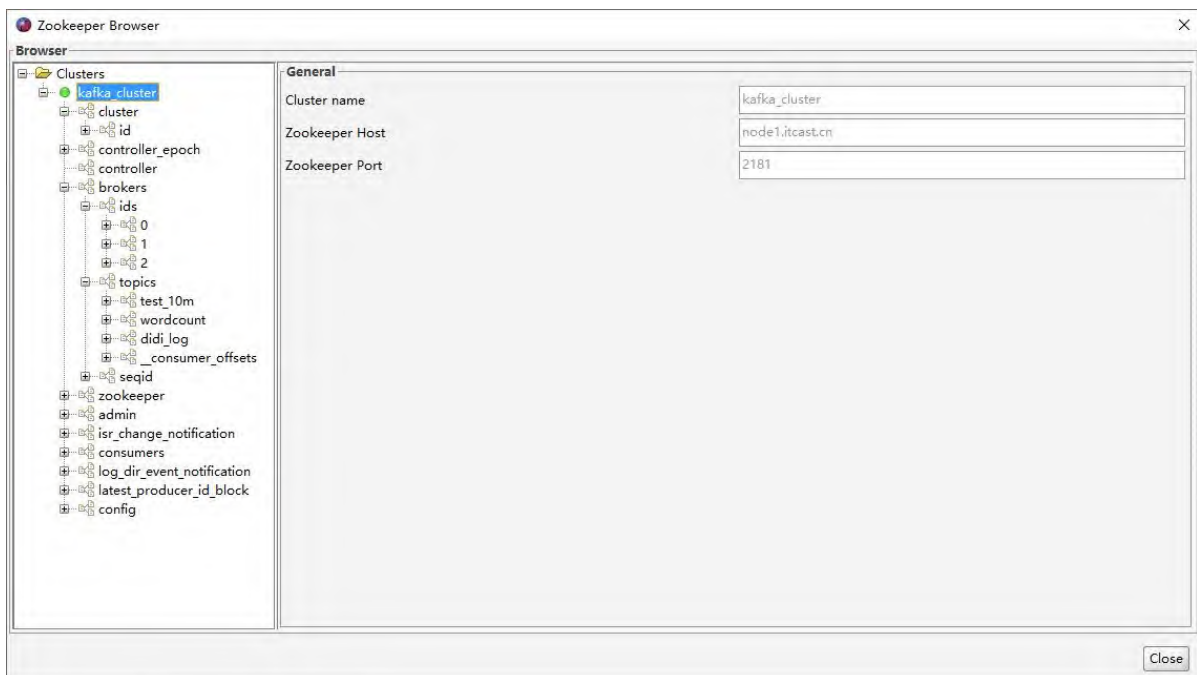
6.1 Kafka重要概念

6.1.1 broker



- 一个Kafka的集群通常由多个broker组成，这样才能实现负载均衡、以及容错
- broker是**无状态 (Stateless)** 的，它们是通过ZooKeeper来维护集群状态
- 一个Kafka的broker每秒可以处理数十万次读写，每个broker都可以处理TB消息而不影响性能

6.1.2 zookeeper



- ZK用来管理和协调broker，并且存储了Kafka的元数据（例如：有多少topic、partition、

consumer)

- ZK服务主要用于通知生产者和消费者Kafka集群中有新的broker加入、或者Kafka集群中出现故障的broker。

PS: Kafka正在逐步想办法将ZooKeeper剥离，维护两套集群成本较高，社区提出KIP-500就是要替换掉ZooKeeper的依赖。“Kafka on Kafka”——Kafka自己来管理自己的元数据

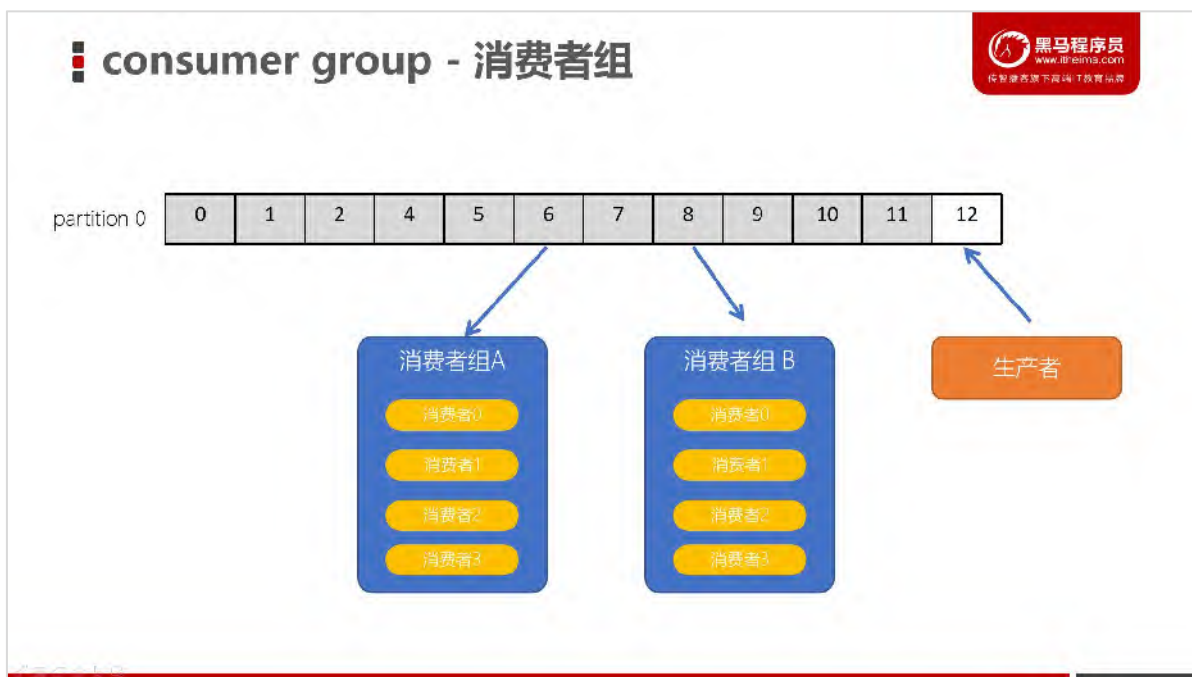
6.1.3 producer (生产者)

- 生产者负责将数据推送给broker的topic

6.1.4 consumer (消费者)

- 消费者负责从broker的topic中拉取数据，并自己进行处理

6.1.5 consumer group (消费者组)



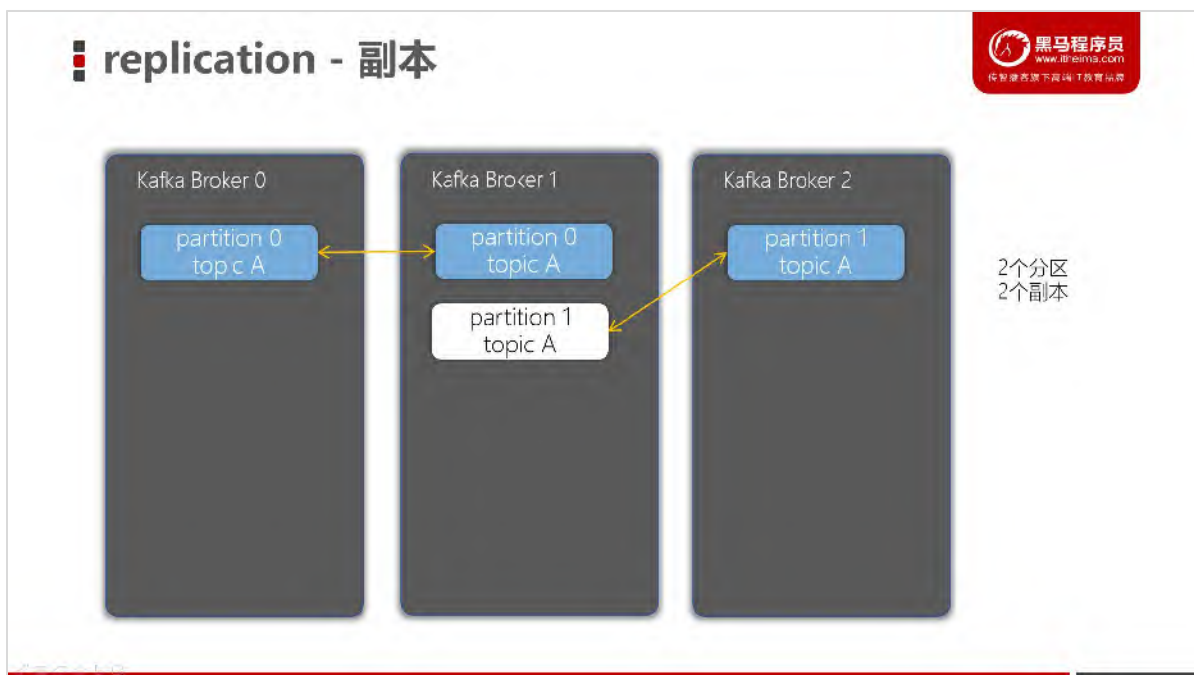
- consumer group是kafka提供的可扩展且具有容错性的消费者机制
- 一个消费者组可以包含多个消费者
- 一个消费者组有一个唯一的ID (group Id)
- 组内的消费者一起消费主题的所有分区数据

6.1.6 分区 (Partitions)



- 在Kafka集群中，主题被分为多个分区

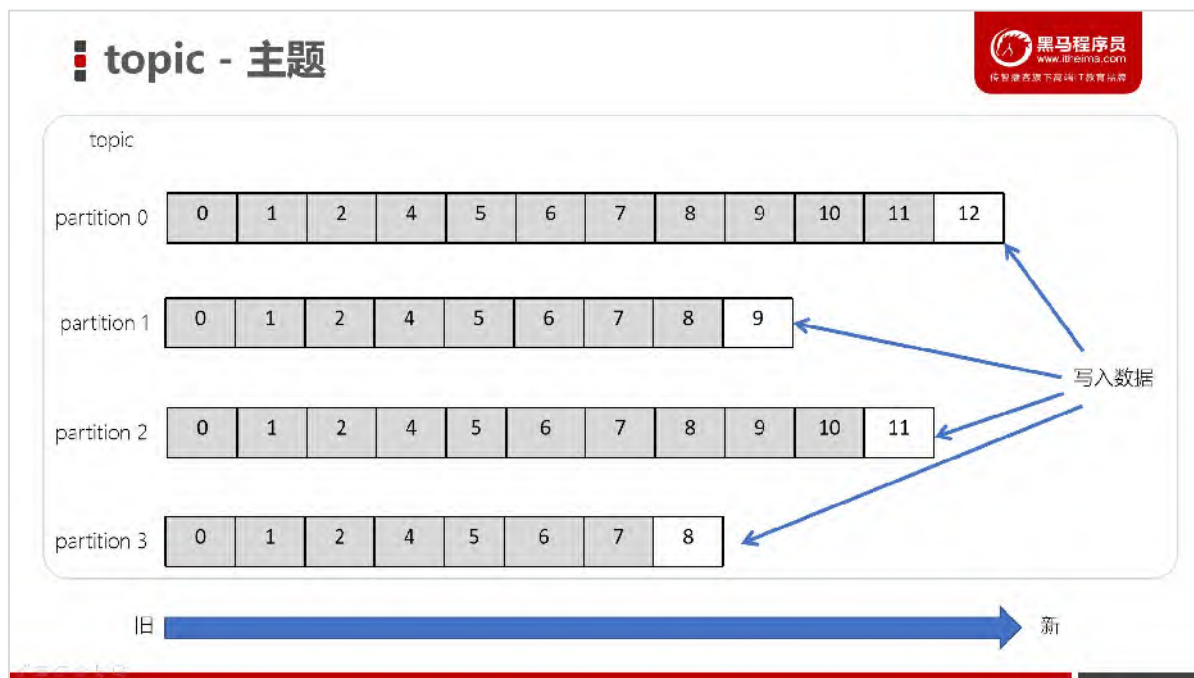
6.1.7 副本 (Replicas)



- 副本可以确保某个服务器出现故障时，确保数据依然可用

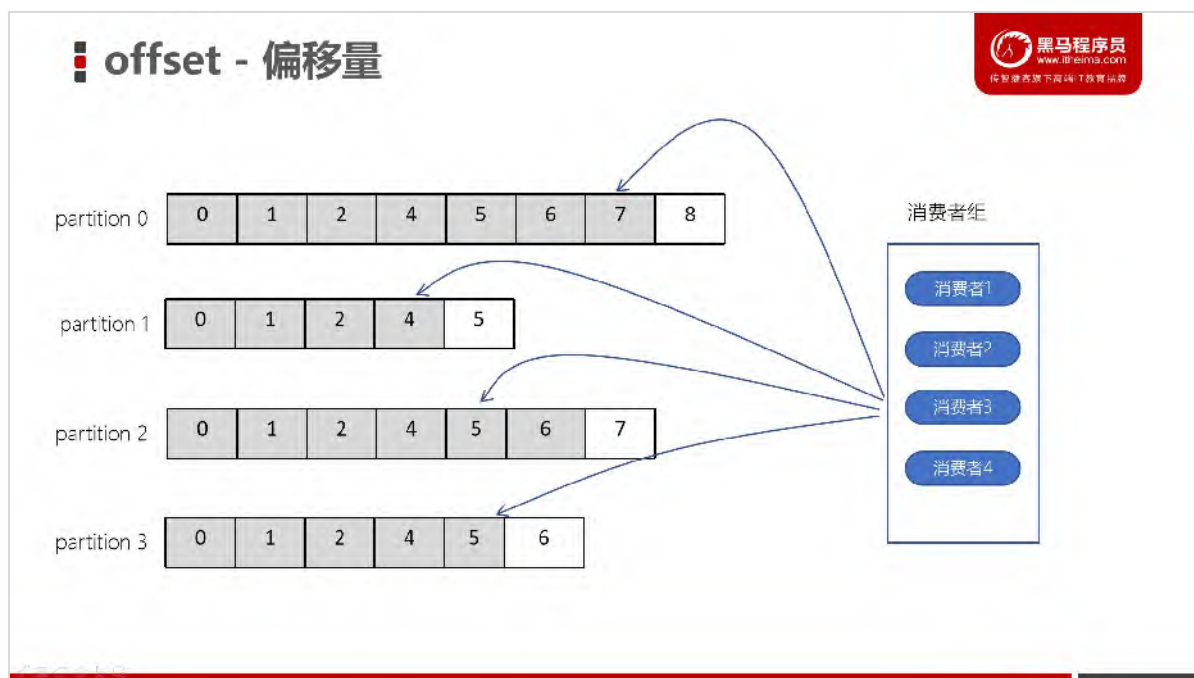
- 在Kafka中，一般都会设计副本的个数 > 1

6.1.8 主题 (Topic)



- 主题是一个逻辑概念，用于生产者发布数据，消费者拉取数据
- Kafka中的主题必须要有标识符，而且是唯一的，Kafka中可以有任意数量的主题，没有数量上的限制
- 在主题中的消息是有结构的，一般一个主题包含某一类消息
- 一旦生产者发送消息到主题中，这些消息就不能被更新（更改）

6.1.9 偏移量 (offset)



- offset记录着下一条将要发送给Consumer的消息的序号
- 默认Kafka将offset存储在ZooKeeper中
- 在一个分区中，消息是有顺序的方式存储着，每个在分区的消费都是有一个递增的id。这个就是偏移量offset
- 偏移量在分区中才是有意义的。在分区之间，offset是没有任何意义的

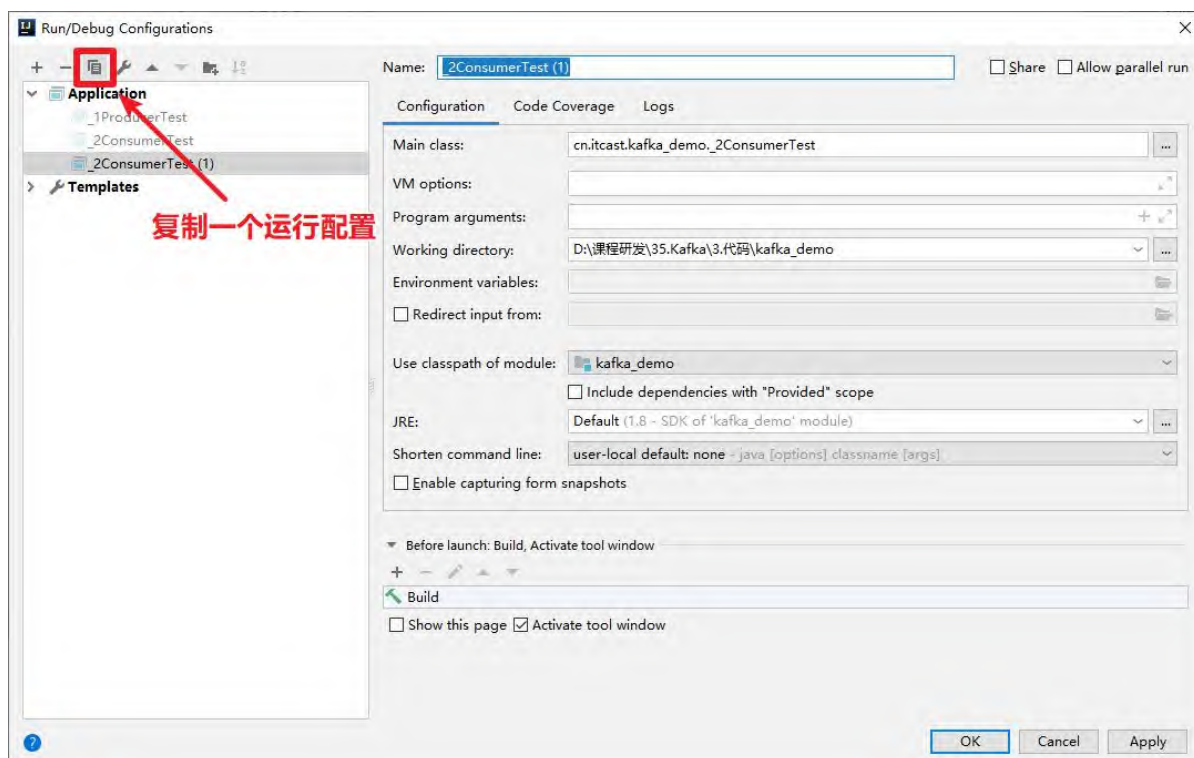
6.2 消费者组

Kafka支持有多个消费者同时消费一个主题中的数据。我们接下来，给大家演示，启动两个消费者共同来消费 test 主题的数据。

1. 首先，修改生产者程序，让生产者每3秒生产1-100个数字。

```
// 3. 发送1-100数字到Kafka的test主题中
while(true) {
    for (int i = 1; i <= 100; ++i) {
        // 注意：send方法是一个异步方法，它会将要发送的数据放入到一个buffer中，然后立即返回
        // 这样可以让消息发送变得更高效
        producer.send(new ProducerRecord<>("test", i + ""));
    }
    Thread.sleep(3000);
}
```

2. 接下来，同时运行两个消费者。



3. 同时运行两个消费者，我们发现，只有一个消费者程序能够拉取到消息。想要让两个消费者同时消费消息，必须要给test主题，添加一个分区。

设置 test topic为2个分区

```
bin/kafka-topics.sh --zookeeper 192.168.88.100:2181 -alter --partitions 2 --topic test
```

4. 重新运行生产者、两个消费者程序，我们就可以看到两个消费者都可以消费Kafka Topic的数据了

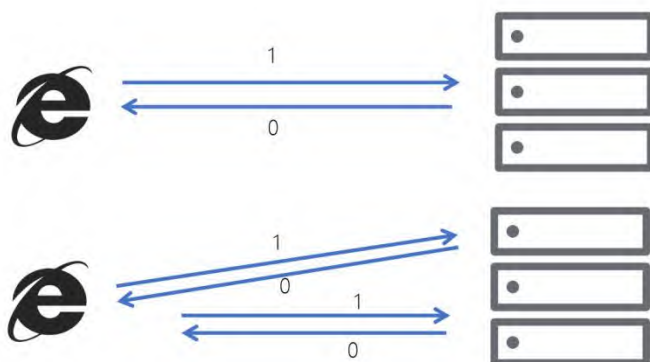
7. Kafka生产者幂等性与事务

7.1 幂等性

7.1.1 简介

拿http举例来说，一次或多次请求，得到地响应是一致的（网络超时等问题除外），换句话说，就是执行多次操作与执行一次操作的影响是一样的。

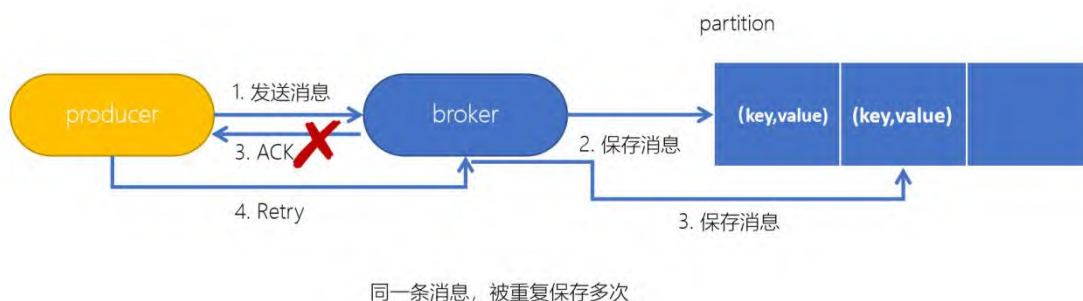
■ 幂等性



如果，某个系统是不具备幂等性的，如果用户重复提交了某个表格，就可能会造成不良影响。例如：用户在浏览器上点击了多次提交订单按钮，会在后台生成多个一模一样的订单。

7.1.2 Kafka生产者幂等性

■ Kafka幂等性



在生产者生产消息时，如果出现retry时，有可能会一条消息被发送了多次，如果Kafka不具备幂等性的，就有可能在partition中保存多条一模一样的消息。

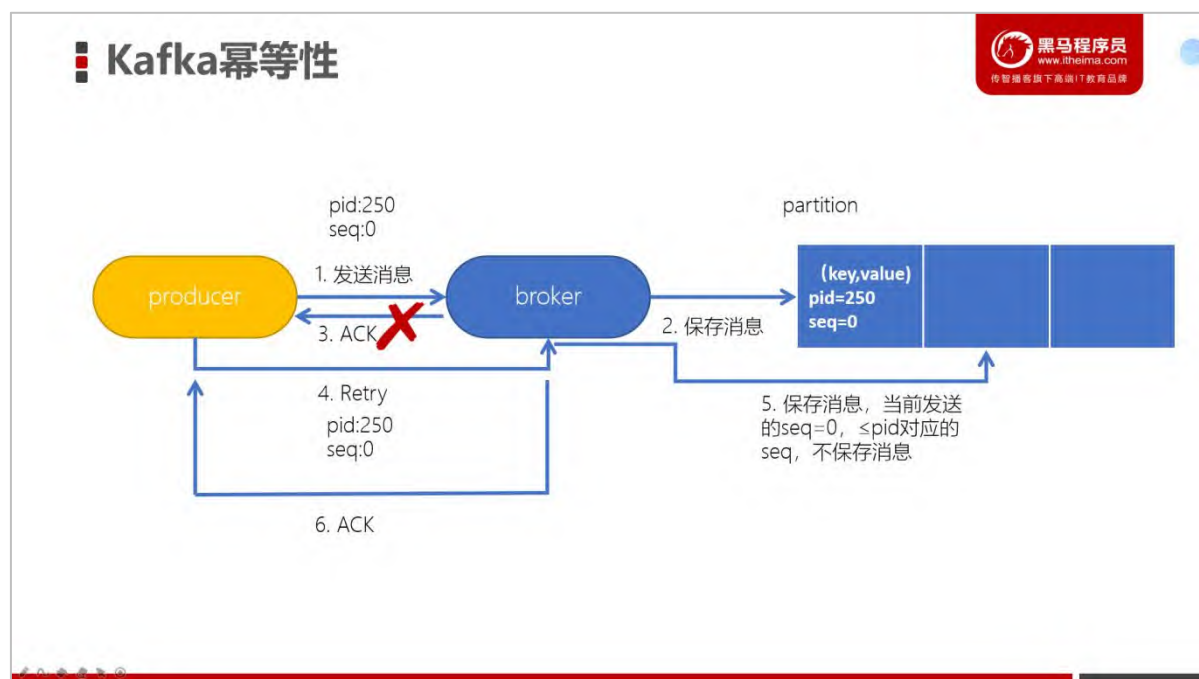
7.1.3 配置幂等性

```
props.put("enable.idempotence",true);
```

7.1.4 幂等性原理

为了实现生产者的幂等性，Kafka引入了 Producer ID (PID) 和 Sequence Number的概念。

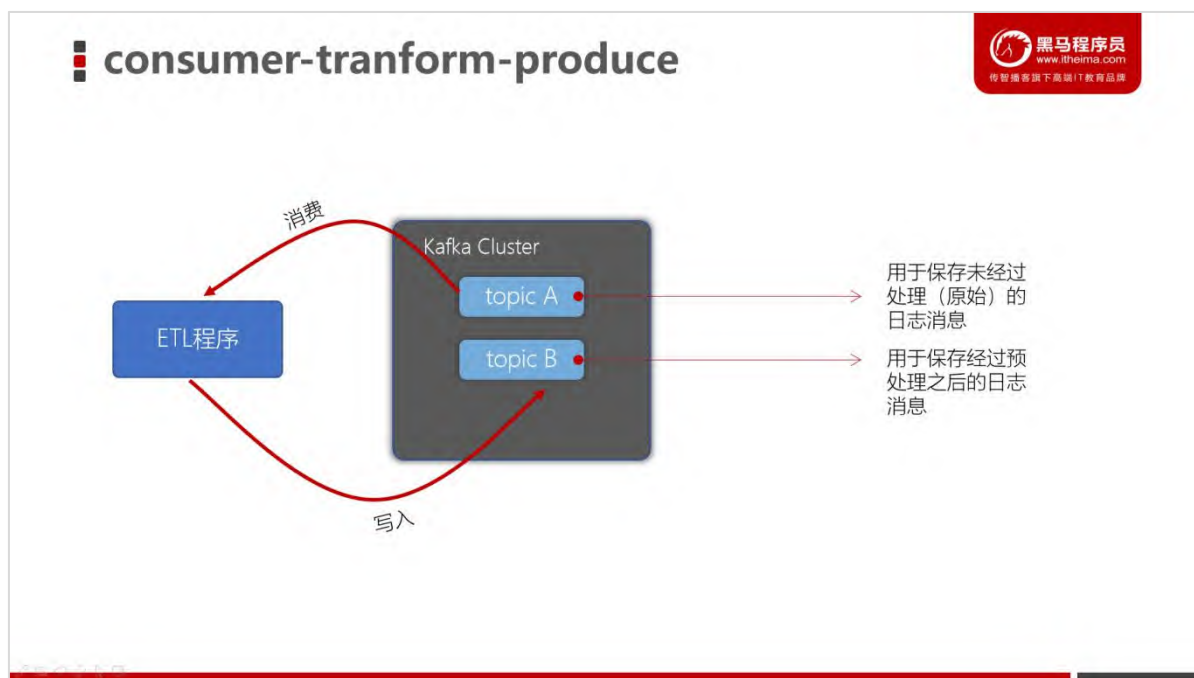
- PID：每个Producer在初始化时，都会分配一个唯一的PID，这个PID对用户来说，是透明的。
- Sequence Number：针对每个生产者（对应PID）发送到指定主题分区的信息都对应一个从0开始递增的Sequence Number。



7.2 Kafka事务

7.2.1 简介

Kafka事务是2017年Kafka 0.11.0.0引入的新特性。类似于数据库的事务。Kafka事务指的是生产者生产消息以及消费者提交offset的操作可以在一个原子操作中，要么都成功，要么都失败。尤其是在生产者、消费者并存时，事务的保障尤其重要。（consumer-transform-producer模式）



7.2.2 事务操作API

Producer接口中定义了以下5个事务相关方法：

1. `initTransactions`（初始化事务）：要使用Kafka事务，必须先进行初始化操作
2. `beginTransaction`（开始事务）：启动一个Kafka事务
3. `sendOffsetsToTransaction`（提交偏移量）：批量地将分区对应的offset发送到事务中，方便后续一块提交
4. `commitTransaction`（提交事务）：提交事务
5. `abortTransaction`（放弃事务）：取消事务

7.3 【理解】Kafka事务编程

7.3.1 事务相关属性配置

7.3.1.1 生产者

// 配置事务的id，开启了事务会默认开启幂等性

```
props.put("transactional.id", "first-transactional");
```

7.3.1.2 消费者

```
// 1. 消费者需要设置隔离级别
props.put("isolation.level", "read_committed");

// 2. 关闭自动提交
props.put("enable.auto.commit", "false");
```

7.3.2 Kafka事务编程

7.3.2.1 需求

在Kafka的topic 「ods_user」中有一些用户数据，数据格式如下：

```
姓名,性别,出生日期
张三,1,1980-10-09
李四,0,1985-11-01
```

我们需要编写程序，将用户的性别转换为男、女（1-男，0-女），转换后将数据写入到topic 「dwd_user」中。要求使用事务保障，要么消费了数据同时写入数据到 topic，提交offset。要么全部失败。

7.3.2.2 启动生产者控制台程序模拟数据

```
# 创建名为ods_user和dwd_user的主题
bin/kafka-topics.sh --create --bootstrap-server node1.itcast.cn:9092 --topic ods_user
bin/kafka-topics.sh --create --bootstrap-server node1.itcast.cn:9092 --topic dwd_user
# 生产数据到 ods_user
bin/kafka-console-producer.sh --broker-list node1.itcast.cn:9092 --topic ods_user
# 从dwd_user消费数据
bin/kafka-console-consumer.sh --bootstrap-server node1.itcast.cn:9092 --topic dwd_user
--from-beginning --isolation-level read_committed
```

7.3.2.3 编写创建消费者代码

编写一个方法 createConsumer，该方法中返回一个消费者，订阅「ods_user」主题。注意：需要配置事务隔离级别、关闭自动提交。

实现步骤：

1. 创建Kafka消费者配置

```
Properties props = new Properties();
props.setProperty("bootstrap.servers", "node1.itcast.cn:9092");
props.setProperty("group.id", "ods_user");
props.put("isolation.level", "read_committed");
props.setProperty("enable.auto.commit", "false");
props.setProperty("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
props.setProperty("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
```

2. 创建消费者，并订阅 ods_user 主题

```
// 1. 创建消费者
public static Consumer<String, String> createConsumer() {
    // 1. 创建Kafka消费者配置
    Properties props = new Properties();
    props.setProperty("bootstrap.servers", "node1.itcast.cn:9092");
    props.setProperty("group.id", "ods_user");
    props.put("isolation.level", "read_committed");
    props.setProperty("enable.auto.commit", "false");
    props.setProperty("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
    props.setProperty("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");

    // 2. 创建Kafka消费者
    KafkaConsumer<String, String> consumer = new KafkaConsumer<>(props);

    // 3. 订阅要消费的主题
    consumer.subscribe(Arrays.asList("ods_user"));

    return consumer;
}
```

7.3.2.4 编写创建生产者代码

编写一个方法 createProducer，返回一个生产者对象。注意：需要配置事务的id，开启了事务会默认开启幂等性。

1. 创建生产者配置

```
Properties props = new Properties();
props.put("bootstrap.servers", "node1.itcast.cn:9092");
props.put("transactional.id", "dwd_user");
props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");
```

2. 创建生产者对象

```
public static Producer<String, String> createProducer() {
    // 1. 创建生产者配置
    Properties props = new Properties();
    props.put("bootstrap.servers", "node1.itcast.cn:9092");
    props.put("transactional.id", "dwd_user");
    props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
    props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");

    // 2. 创建生产者
    Producer<String, String> producer = new KafkaProducer<>(props);
    return producer;
}
```

7.3.2.5 编写代码消费并生产数据

实现步骤：

1. 调用之前实现的方法，创建消费者、生产者对象
2. 生产者调用initTransactions初始化事务
3. 编写一个while死循环，在while循环中不断拉取数据，进行处理后，再写入到指定的topic
 - (1) 生产者开启事务
 - (2) 消费者拉取消息
 - (3) 遍历拉取到的消息，并进行预处理（将1转换为男，0转换为女）



- (4) 生产消息到dwd_user topic中
- (5) 提交偏移量到事务中
- (6) 提交事务
- (7) 捕获异常，如果出现异常，则取消事务

```
public static void main(String[] args) {
    Consumer<String, String> consumer = createConsumer();
    Producer<String, String> producer = createProducer();

    // 初始化事务
    producer.initTransactions();

    while(true) {
        try {
            // 1. 开启事务
            producer.beginTransaction();

            // 2. 定义Map结构，用于保存分区对应的offset
            Map<TopicPartition, OffsetAndMetadata> offsetCommits = new HashMap<>();

            // 2. 拉取消息
            ConsumerRecords<String, String> records = consumer.poll(Duration.ofSeconds(2));

            for (ConsumerRecord<String, String> record : records) {
                // 3. 保存偏移量
                offsetCommits.put(new TopicPartition(record.topic(), record.partition()),
                    new OffsetAndMetadata(record.offset() + 1));

                // 4. 进行转换处理
                String[] fields = record.value().split(",");
                fields[1] = fields[1].equalsIgnoreCase("1") ? "男":"女";
                String message = fields[0] + "," + fields[1] + "," + fields[2];

                // 5. 生产消息到dwd_user
                producer.send(new ProducerRecord<>("dwd_user", message));
            }

            // 6. 提交偏移量到事务
            producer.sendOffsetsToTransaction(offsetCommits, "ods_user");

            // 7. 提交事务
            producer.commitTransaction();
        } catch (Exception e) {
```

```
        // 8. 放弃事务
        producer.abortTransaction();
    }
}
```

7.3.2.6 测试

往之前启动的console-producer中写入消息进行测试，同时检查console-consumer是否能够接收到消息：

```
[root@node1 kafka_2.12-2.4.1]# bin/kafka-console-producer.sh --broker-list node1.itcast.cn:9092 --topic ods_user
>
```

逐个测试一下消息：

```
张三,1,1980-10-09
李四,0,1985-11-01
```

7.3.2.7 模拟异常测试事务

```
// 3. 保存偏移量
offsetCommits.put(new TopicPartition(record.topic(), record.partition()),
    new OffsetAndMetadata(record.offset() + 1));

// 4. 进行转换处理
String[] fields = record.value().split(",");
fields[1] = fields[1].equalsIgnoreCase("1") ? "男":"女";
String message = fields[0] + "," + fields[1] + "," + fields[2];

// 模拟异常
int i = 1/0;

// 5. 生产消息到dwd_user
producer.send(new ProducerRecord<>("dwd_user", message));
```

启动程序一次，抛出异常。

再启动程序一次，还是抛出异常。

直到我们处理该异常为止。

我们发现，可以消费到消息，但如果中间出现异常的话，offset是不会被提交的，除非消费、生产消息都成功，才会提交事务。