# 课程介绍

课程名称：Storm上游数据源之Kakfa

课程目标：

通过本课程理解Storm消费的数据来源、理解JMS规范、理解Kafka核心组件、掌握Kakfa生产者API、掌握Kafka消费者API。对流式计算的生态环境有深入的了解，具备流式计算项目架构的能力。

课程大纲：

1. kafka是什么？
2. JMS规范是什么？
3. 为什么需要消息队列？
4. Kafka核心组件
5. Kafka安装部署
6. Kafka生产者Java API
7. Kafka消费者Java API

# 课程内容

## 1、Kafka是什么

在流式计算中，Kafka一般用来缓存数据，Storm通过消费Kafka的数据进行计算。

KAFKA + STORM +REDIS

* Apache Kafka是一个开源**消息**系统，由Scala写成。是由Apache软件基金会开发的一个开源消息系统项目。
* **Kafka是一个分布式消息队列：生产者、消费者的功能。它提供了类似于JMS的特性，但是在设计实现上完全不同，此外它并不是JMS规范的实现。**
* Kafka对消息保存时根据Topic进行归类，发送消息者称为Producer,消息接受者称为Consumer,此外kafka集群有多个kafka实例组成，每个实例(server)成为broker。
* 无论是kafka集群，还是producer和consumer都依赖于**zookeeper**集群保存一些meta信息，来保证系统可用性

1、kafka是什么

类JMS消息队列，结合JMS中的两种模式，可以有多个消费者主动拉取数据，在JMS中只有点对点模式才有消费者主动拉取数据。

kafka是一个生产-消费模型。

Producer：生产者，只负责数据生产，生产者的代码可以集成到任务系统中。数据的分发策略由producer决定，默认是defaultPartition Utils.abs(key.hashCode) % numPartitions

Broker：当前服务器上的Kafka进程,俗称拉皮条。只管数据存储，不管是谁生产，不管是谁消费。**在集群中每个broker都有一个唯一brokerid，不得重复。**

Topic:目标发送的目的地，这是一个逻辑上的概念，**落到磁盘上是一个partition的目录。**partition的目录中有多个segment组合(index,log)。一个Topic对应多个partition[0,1,2,3]，一个partition对应多个segment组合。一个segment有默认的大小是1G。每个partition可以设置多个副本(replication-factor 1),会从所有的副本中选取一个leader出来。**所有读写操作都是通过leader来进行的**。特别强调，和mysql中主从有区别，mysql做主从是为了读写分离，在kafka中读写操作都是leader。

ConsumerGroup：数据消费者组，ConsumerGroup可以有多个，**每个ConsumerGroup消费的数据都是一样的**。可以把多个consumer线程划分为一个组，组里面所有成员共同消费一个topic的数据，组员之间不能重复消费。

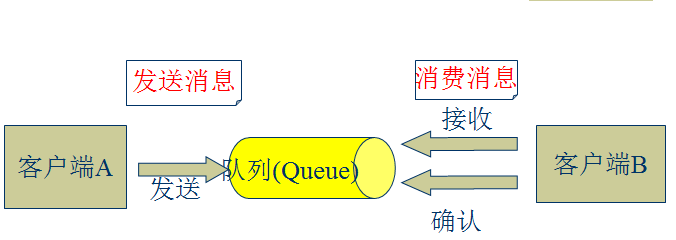
## 2、JMS是什么

### 2.1、JMS的基础

JMS是什么：JMS是Java提供的一套技术规范

JMS干什么用：用来异构系统 集成通信，缓解系统瓶颈，提高系统的伸缩性增强系统用户体验，使得系统模块化和组件化变得可行并更加灵活

通过什么方式：生产消费者模式（生产者、服务器、消费者）



jdk，kafka，activemq……

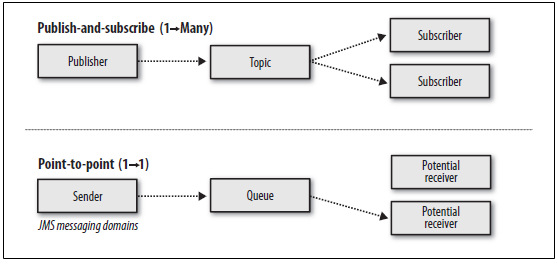
### 2.2、JMS消息传输模型

* 点对点模式**（一对一，消费者主动拉取数据，消息收到后消息清除）**

点对点模型通常是一个基于拉取或者轮询的消息传送模型，这种模型从队列中请求信息，而不是将消息推送到客户端。这个模型的特点是发送到队列的消息被**一个且只有一个接收者接收处理**，即使有多个消息监听者也是如此。

* 发布/订阅模式**（一对多，数据生产后，推送给所有订阅者）**

发布订阅模型则是一个基于推送的消息传送模型。发布订阅模型可以有多种不同的订阅者，临时订阅者只在主动监听主题时才接收消息，而持久订阅者则监听主题的所有消息，**即时当前订阅者不可用，处于离线状态**。

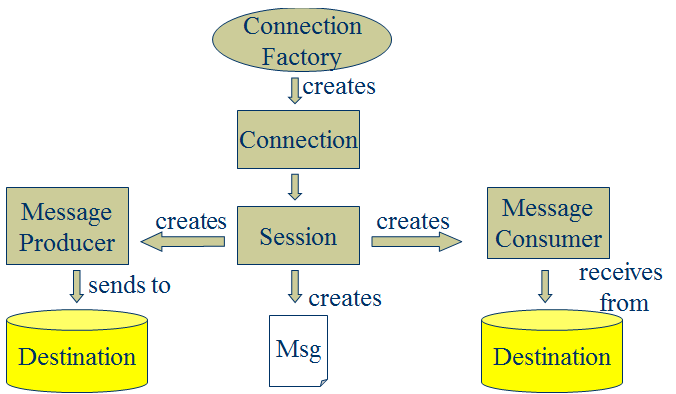


queue.put（object） 数据生产

queue.take(object) 数据消费

### 2.3、JMS核心组件

* Destination：消息发送的目的地，也就是前面说的Queue和Topic。
* Message ：从字面上就可以看出是被发送的消息。
* Producer： 消息的生产者，要发送一个消息，必须通过这个生产者来发送。
* MessageConsumer： 与生产者相对应，这是消息的消费者或接收者，通过它来接收一个消息。



通过与ConnectionFactory可以获得一个connection

通过connection可以获得一个session会话。

### 2.4、常见的类JMS消息服务器

#### 2.4.1、JMS消息服务器 ActiveMQ

ActiveMQ 是Apache出品，最流行的，能力强劲的开源消息总线。ActiveMQ 是一个完全支持JMS1.1和J2EE 1.4规范的。

主要特点：

* 多种语言和协议编写客户端。语言: Java, C, C++, C#, Ruby, Perl, Python, PHP。应用协议: OpenWire,Stomp REST,WS Notification,XMPP,AMQP
* 完全支持JMS1.1和J2EE 1.4规范 (持久化,XA消息,事务)
* 对Spring的支持,ActiveMQ可以很容易内嵌到使用Spring的系统里面去,而且也支持Spring2.0的特性
* 通过了常见J2EE服务器(如 Geronimo,JBoss 4, GlassFish,WebLogic)的测试,其中通过JCA 1.5 resource adaptors的配置,可以让ActiveMQ可以自动的部署到任何兼容J2EE 1.4 商业服务器上
* 支持多种传送协议:in-VM,TCP,SSL,NIO,UDP,JGroups,JXTA
* 支持通过JDBC和journal提供高速的消息持久化
* 从设计上保证了高性能的集群,客户端-服务器,点对点
* 支持Ajax
* 支持与Axis的整合
* 可以很容易得调用内嵌JMS provider,进行测试

#### 2.4.2、分布式消息中间件 Metamorphosis

Metamorphosis (MetaQ) 是一个高性能、高可用、可扩展的分布式消息中间件，类似于LinkedIn的Kafka，具有消息存储顺序写、吞吐量大和支持本地和XA事务等特性，适用于大吞吐量、顺序消息、广播和日志数据传输等场景，在淘宝和支付宝有着广泛的应用，现已开源。

主要特点：

* 生产者、服务器和消费者都可分布
* 消息存储顺序写
* 性能极高,吞吐量大
* 支持消息顺序
* 支持本地和XA事务
* 客户端pull，随机读,利用sendfile系统调用，zero-copy ,批量拉数据
* 支持消费端事务
* 支持消息广播模式
* 支持异步发送消息
* 支持http协议
* 支持消息重试和recover
* 数据迁移、扩容对用户透明
* 消费状态保存在客户端
* 支持同步和异步复制两种HA
* 支持group commit

#### 2.4.3、分布式消息中间件 RocketMQ

RocketMQ 是一款分布式、队列模型的消息中间件，具有以下特点：

* 能够保证严格的消息顺序
* 提供丰富的消息拉取模式
* 高效的订阅者水平扩展能力
* 实时的消息订阅机制
* 亿级消息堆积能力
* **Metaq3.0 版本改名，产品名称改为RocketMQ**

#### 2.4.4、其他MQ

* .NET消息中间件 DotNetMQ
* 基于HBase的消息队列 HQueue
* Go 的 MQ 框架 KiteQ
* AMQP消息服务器 RabbitMQ
* MemcacheQ 是一个基于 MemcacheDB 的消息队列服务器。

## 3、为什么需要消息队列（重要）

消息系统的核心作用就是三点：解耦，异步和并行

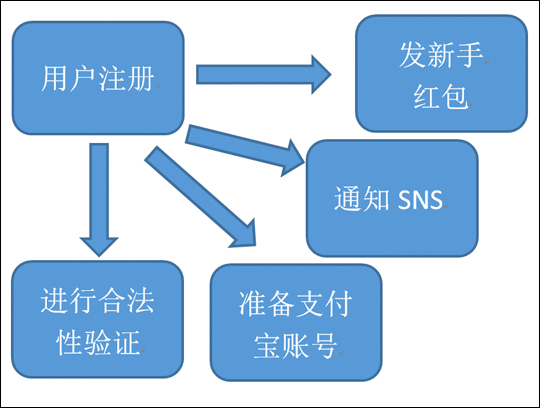
以用户注册的案列来说明消息系统的作用

### 3.1、用户注册的一般流程

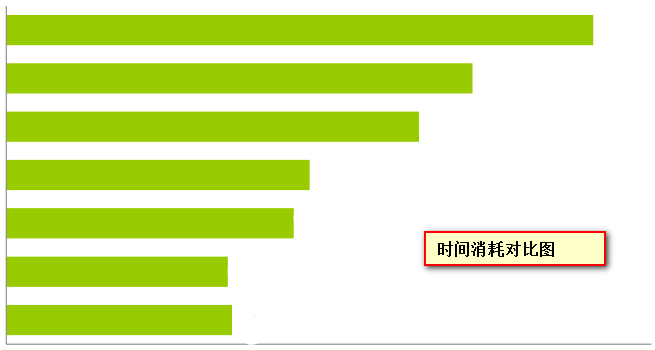


**问题**：随着后端流程越来越多，每步流程都需要额外的耗费很多时间，从而会导致用户更长的等待延迟。

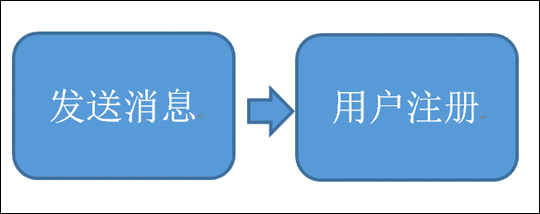
### 3.2、用户注册的并行执行



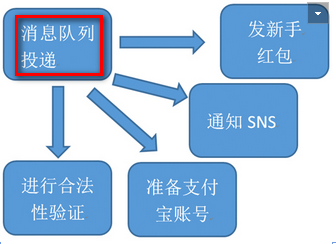
**问题**：系统并行的发起了4个请求，4个请求中，如果某一个环节执行1分钟，其他环节再快，用户也需要等待1分钟。如果其中一个环节异常之后，整个服务挂掉了。



### 3.3、用户注册的最终一致



1. **保证主流程的正常执行**、执行成功之后，发送MQ消息出去。
2. 需要这个destination的其他系统通过消费数据再执行，最终一致。



## 4、Kafka核心组件

* Topic ：消息根据Topic进行归类
* Producer：发送消息者
* Consumer：消息接受者
* **broker：每个kafka实例(server)**
* Zookeeper：依赖集群保存meta信息。



## 5、Kafka集群部署

### 5.1集群部署的基本流程

下载安装包、解压安装包、修改配置文件、分发安装包、启动集群

### 5.2集群部署的基础环境准备

安装前的准备工作（zk集群已经部署完毕）

* 关闭防火墙

chkconfig iptables off && setenforce 0

* 创建用户

groupadd realtime &&　useradd realtime　&& usermod -a -G realtime realtime

* 创建工作目录并赋权

mkdir /export

mkdir /export/servers

chmod 755 -R /export

* 切换到realtime用户下

su realtime

### 5.3 Kafka集群部署

#### 5.3.1、下载安装包

<http://kafka.apache.org/downloads.html>

在linux中使用wget命令下载安装包

wget http://mirrors.hust.edu.cn/apache/kafka/0.8.2.2/kafka\_2.11-0.8.2.2.tgz

#### 5.3.2、解压安装包

tar -zxvf /export/software/kafka\_2.11-0.8.2.2.tgz -C /export/servers/

cd /export/servers/

ln -s kafka\_2.11-0.8.2.2 kafka

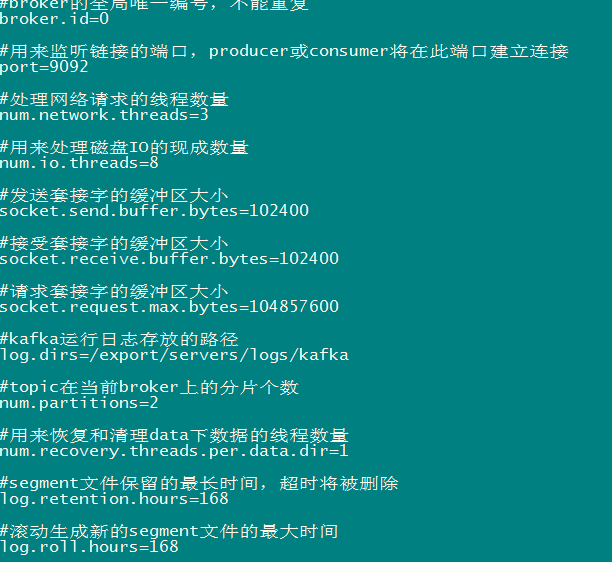
#### 5.3.3、修改配置文件

cp /export/servers/kafka/config/server.properties

/export/servers/kafka/config/server.properties.bak

vi /export/servers/kafka/config/server.properties

输入以下内容：



#### 5.3.4、分发安装包

scp -r /export/servers/kafka\_2.11-0.8.2.2 kafka02:/export/servers

然后分别在各机器上创建软连

cd /export/servers/

ln -s kafka\_2.11-0.8.2.2 kafka

#### 5.3.5、再次修改配置文件（重要）

**依次修改各服务器上配置文件的的broker.id，分别是0,1,2不得重复。**

#### 5.3.6、启动集群

依次在各节点上启动kafka

bin/kafka-server-start.sh config/server.properties

### 5.4、Kafka常用操作命令

* 查看当前服务器中的所有topic

bin/kafka-topics.sh --list --zookeeper zk01:2181

* 创建topic

./kafka-topics.sh --create --zookeeper mini1:2181 --replication-factor 1 --partitions 3 --topic first

* 删除topic

sh bin/kafka-topics.sh --delete --zookeeper zk01:2181 --topic test

需要server.properties中设置delete.topic.enable=true否则只是标记删除或者直接重启。

* 通过shell命令发送消息

kafka-console-producer.sh --broker-list kafka01:9092 --topic itheima

* 通过shell消费消息

sh bin/kafka-console-consumer.sh --zookeeper zk01:2181 --from-beginning --topic test1

* 查看消费位置

sh kafka-run-class.sh kafka.tools.ConsumerOffsetChecker --zookeeper zk01:2181 --group testGroup

* 查看某个Topic的详情

sh kafka-topics.sh --topic test --describe --zookeeper zk01:2181

## 6、Kafka生产者Java API



## 7、Kafka消费者Java API



# 课程介绍

课程名称：

Kafka技术增强

注：学习本课程请先学习Kafka基础

课程目标：

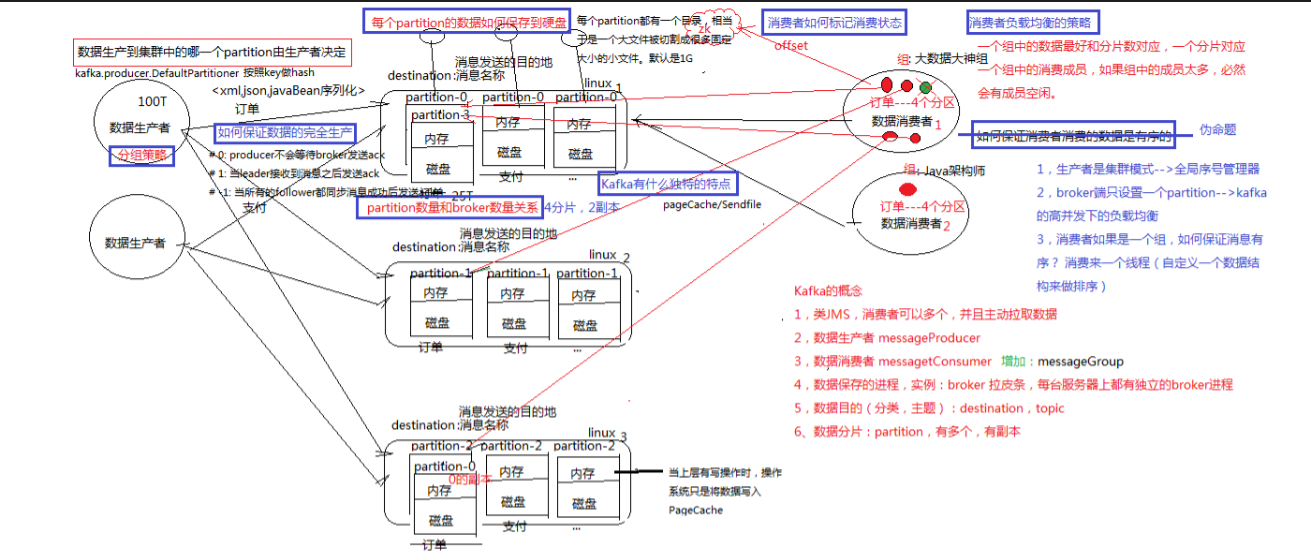
通过本模块的学习，能够掌握Kafka的负载均衡、Producer生产数据、Kafka文件存储机制、Kafka自定义partition

课程大纲：

1. Kafka整体结构图
2. Consumer与topic关系
3. Kafka Producer消息分发
4. Consumer 的负载均衡
5. Kafka文件存储机制

# 课程内容

## 8、Kafka整体结构图



**●分组策略：**

kafka生产数据时的分组策略

默认是defaultPartition Utils.abs(key.hashCode) % numPartitions上文中的key是producer在发送数据时传入的，produer.send(KeyedMessage(topic,myPartitionKey,messageContent))；

**●kafka如何保证数据的完全生产**

ack机制：broker表示发来的数据已确认接收无误，表示数据已经保存到磁盘。

0：不等待broker返回确认消息

1：等待topic中某个partition leader保存成功的状态反馈

-1：等待topic中某个partition 所有副本都保存成功的状态反馈

request.required.acks=0

**●broker如何保存数据**

在理论环境下，broker**按照顺序读写的机制**，可以**每秒保存600M**的数据。主要通过**pagecache机制，尽可能的利用当前物理机器上的空闲内存来做缓存**。当前topic所属的broker，必定有一个该topic的partition，partition是一个磁盘目录。partition的目录中有多个segment组合(index,log)

**●partition如何分布在不同的broker上（partition与broker的关系）**

int i = 0

list{kafka01,kafka02,kafka03}

for(int i=0;i<5;i++){

brIndex = i%broker;

hostName = list.get(brIndex)

}

例如：

kafka01：paytest-2，paytest-5，paytest-8

kafka02：paytest-0，paytest-3，paytest-6，paytest-9

kafka03：paytest-1，paytest-4，paytest-7，paytest-10

**●consumerGroup的组员和partition之间如何做负载均衡**

当一个group中,有consumer加入或者离开时,**会触发partitions均衡**.均衡的最终目的,是提升topic的并发消费能力。最好是一一对应，一个partition对应一个consumer。如果consumer的数量过多，必然有空闲的consumer。

算法：

假如topic1,具有如下partitions: P0,P1,P2,P3

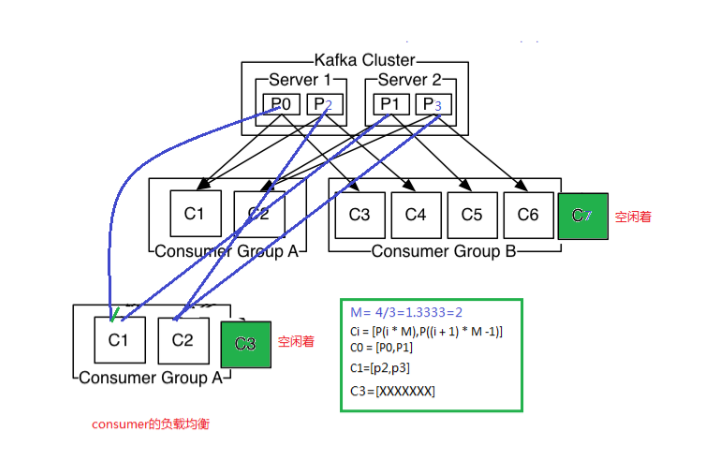
加入group中,有如下consumer: C1,C2

首先根据partition索引号对partitions排序: P0,P1,P2,P3

根据consumer.id排序: C0,C1

计算倍数: M = [P0,P1,P2,P3].size / [C0,C1].size,本例值M=2(向上取整)

然后依次分配partitions: C0 = [P0,P1],C1=[P2,P3],即Ci = [P(i \* M),P((i + 1) \* M -1)]



**●如何保证kafka消费者消费数据是全局有序的**

伪命题

如果要全局有序的，必须保证生产有序，存储有序，消费有序。由于生产可以做集群，存储可以分片，消费可以设置为一个consumerGroup，要保证全局有序，就需要保证每个环节都有序。只有一个可能，就是一个生产者，一个partition，一个消费者。这种场景和大数据应用场景相悖。

**问题：**

**1，kafka+storm如何保证消息完整处理。**

**一条消息产生----Kafka--KafkaSpout-Storm--->Redis**

**问题1：kafka数据生产消费如何保证消息的完整处理**

**Producer-batch(缓存机制queue)--重试机制---->ack(-1,1,0)---Broker(partition leader/slave)------>KafkaConsumer(内存中，new offset)---->zk( old offset)**

**Producer发送时缓存数据，需要阈值设定（数量阈值，时间阈值），当数据太多并来不及发送时，会产生老数据是否保留的问题，如何配置文件中配置的是删除掉，数据就丢失了。**

**KafkaConsumer 消费数据时，由于offset是周期性更新，导致zk上的offset值必然小于Kafkaconsumer内存中的值，当KafkaConsumer挂掉后重启，必然会导致数据重复消费。**

**如何解决重复消费：1，技术方法(不可取)，2，业务方法（标识数据：找到消息中的唯一标识 <messageTag,isProcess>）**

**问题2：Storm中如何保证消息的完整处理 ack - fail**

**自己定义 缓存Map<msgid,messageObj>**

**Spout---->nextTuple----Tuple---Bolt(ack(tuple))**

**Bolt1-----Tuple1-1**

**Bolt1-----Tuple1-2**

**Bolt1-----Tuple1-3**

**Bolt1-----Tuple1-4**

**Bolt2(ack(Tuple1-1))**

**Bolt2(ack(Tuple1-2))**

**Bolt2(ack(Tuple1-3))**

**Bolt2(ack(Tuple1-4))**

**如果成功（spout.ack(msg)）**

**map.remove(msgid)**

**如果失败（spout.fail(msg)）**

**messageObj = map.get(msgid)**

**collector.emit(messageObj,msgid)**

**问题3：如何保证一条消息路过各个组件时保证全局的完整处理**

**保存每个环节的数据不丢失，自然就全局不丢失。**

**问题4：在storm环节中自定义的map如何存储，在KafkaConsumer如何处理重复消费的问题？**

**1，保存在当前Jvm中，既然出异常导致消息不能完全处理，存放在jvm中的标识数据，缓存Map必然会丢失**

**2，保存在外部的存储中，redis。 标识数据<Set> 缓存Map<Map>**

**问题：请问存储redis时，由于网络原因或其他异常导致数据不能成功存储 怎么办？**

**重试机制，保存3次，打印log日志，redis存储失败。**

**2，数据量大如何保证到Kakfa中，storm如何消费，解决延迟。**

**问题1：请问Kafka如何处理海量数据 -----整体数据：100G**

**数据从何而来：producer 集群----DefaultPartition ------保存数据平均分配**

**数据保存在哪里： broker 集群**

**topic ---partition 10 -------------->每个分片保存10G**

**数据保存如何快：pageCache 600M/S的磁盘写入速度 sendfile技术**

**问题2: 请问Storm如何处理海量数据，尽可能快**

**数据输入： KafkaSpout(consumerGroup,10) 读取外部数据源**

**数据计算： 数据计算是根据对数据处理的业务复杂度来的，越复杂并发度越大。**

**如果bolt的并发读要设置成1万个，才能提高处理速度，很显然是不行的。需要将bolt中的代码逻辑分解出来，形成多个bolt组合。**

**bolt1--->bolt2--->bolt3......**

**3，Flume监控文件，异常之后重新启动，如何避免重复读取。**

**异常的范围是 flumeNg的异常**

**log文件，读取文件，保存记录的行号**

**问题1：如何保存行号**

**command.type = exec shell<tail -F xxx.log> 自定义shell脚本，保存消费的行号到工作目录的某个文件里。**

**当下次启动时，从行号记录文件中拿取上次读到多少行。**

Kafka名词解释和工作方式

* Producer ：消息生产者，就是向kafka broker发消息的客户端。
* Consumer ：消息消费者，向kafka broker取消息的客户端
* Topic ：咋们可以理解为一个队列。
* Consumer Group （CG）：这是kafka用来实现一个topic消息的广播（发给所有的consumer）和单播（发给任意一个consumer）的手段。一个topic可以有多个CG。topic的消息会复制（不是真的复制，是概念上的）到所有的CG，但每个partion只会把消息发给该CG中的一个consumer。如果需要实现广播，只要每个consumer有一个独立的CG就可以了。要实现单播只要所有的consumer在同一个CG。用CG还可以将consumer进行自由的分组而不需要多次发送消息到不同的topic。
* Broker ：一台kafka服务器就是一个broker。一个集群由多个broker组成。一个broker可以容纳多个topic。
* Partition：为了实现扩展性，一个非常大的topic可以分布到多个broker（即服务器）上，一个topic可以分为多个partition，每个partition是一个有序的队列。**partition中的每条消息都会被分配一个有序的id（offset）。kafka只保证按一个partition中的顺序将消息发给consumer，不保证一个topic的整体（多个partition间）的顺序。**
* Offset：kafka的存储文件都是按照offset.kafka来命名，用offset做名字的好处是方便查找。例如你想找位于2049的位置，只要找到2048.kafka的文件即可。当然the first offset就是00000000000.kafka

## 9、Consumer与topic关系

本质上kafka只支持Topic；

* 每个group中可以有多个consumer，每个consumer属于一个consumer group；

通常情况下，一个group中会包含多个consumer，这样不仅可以提高topic中消息的**并发消费能力**，而且还能提高"**故障容错"性**，如果group中的某个consumer失效那么其消费的partitions将会有其他consumer自动接管。

* 对于Topic中的一条特定的消息，只会被订阅此Topic的每个group中的其中一个consumer消费，此消息不会发送给一个group的多个consumer；

那么一个group中所有的consumer将会交错的消费整个Topic，每个group中consumer消息消费互相独立，我们可以认为一个group是一个"订阅"者。

* **在kafka中,一个partition中的消息只会被group中的一个consumer消费(同一时刻)；**

一个Topic中的每个partions，只会被一个"订阅者"中的一个consumer消费，**不过一个consumer可以同时消费多个partitions中的消息。**

* kafka的设计原理决定,对于一个topic，同一个group中不能有多于partitions个数的consumer同时消费，否则将意味着某些consumer将无法得到消息。

**kafka只能保证一个partition中的消息被某个consumer消费时是顺序的；事实上，从Topic角度来说,当有多个partitions时,消息仍不是全局有序的。**

## 10、Kafka消息的分发

**Producer客户端负责消息的分发**

* + kafka集群中的任何**一个broker都可以向producer提供metadata信息**,这些metadata中包含"集群中存活的servers列表"/"partitions leader列表"等信息；
  + 当producer获取到metadata信息之后, **producer将会和Topic下所有partition leader保持socket连接；**
  + **消息由producer直接通过socket发送到broker，中间不会经过任何"路由层"，事实上，消息被路由到哪个partition上由producer客户端决定；**

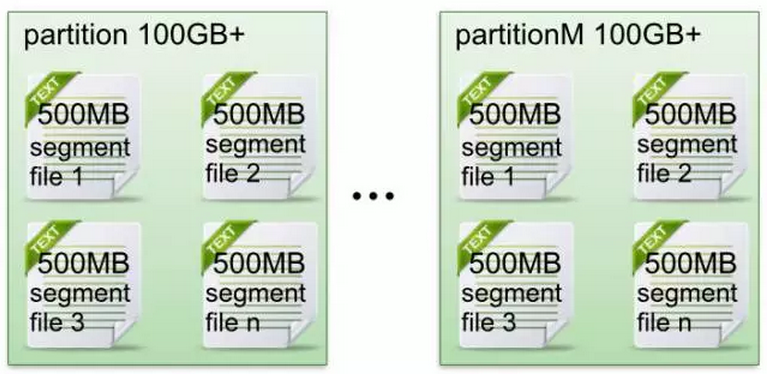
比如可以采用"random""key-hash""轮询"等,**如果一个topic中有多个partitions,那么在producer端实现"消息均衡分发"是必要的。**

* 在producer端的配置文件中,开发者可以指定partition路由的方式。

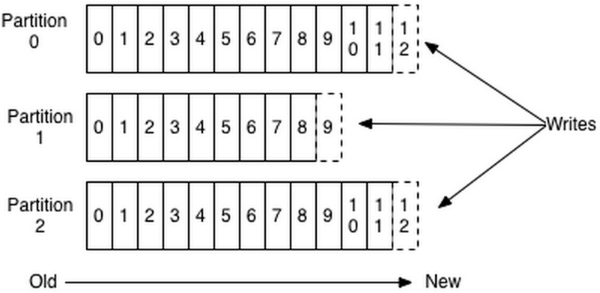
## 11、kafka文件存储机制

### 11.1、Kafka文件存储基本结构

* 在Kafka文件存储中，同一个topic下有多个不同partition，每个partition为一个目录，partiton命名规则为topic名称+有序序号，第一个partiton序号从0开始，序号最大值为partitions数量减1。
* 每个partion(目录)相当于一个巨型文件被平均分配到多个大小相等segment(段)数据文件中。**但每个段segment file消息数量不一定相等**，这种特性方便old segment file快速被删除。默认保留7天的数据。



* 每个partiton只需要支持顺序读写就行了，segment文件生命周期由服务端配置参数决定。（什么时候创建，什么时候删除）



数据有序的讨论？

一个partition的数据是否是有序的？ 间隔性有序，不连续

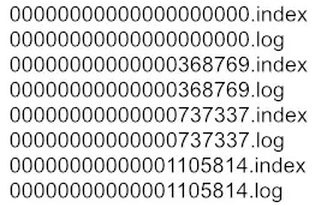
针对一个topic里面的数据，**只能做到partition内部有序，不能做到全局有序。**

特别加入消费者的场景后，如何保证消费者消费的数据全局有序的？伪命题。

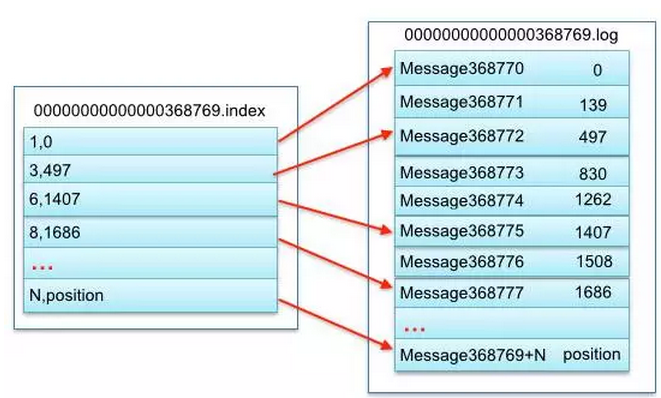
**只有一种情况下才能保证全局有序？就是只有一个partition。**

### 11.2、Kafka Partition Segment

* Segment file组成：由2大部分组成，分别为index file和data file，此2个文件一一对应，成对出现，后缀".index"和“.log”分别表示为segment索引文件、数据文件。



* Segment文件命名规则：partion全局的第一个segment从0开始，后续每个segment文件名为**上一个segment文件最后一条消息的offset值**。数值最大为64位long大小，19位数字字符长度，没有数字用0填充。
* 索引文件存储大量元数据，数据文件存储大量消息，索引文件中元数据指向对应数据文件中message的物理偏移地址。



3，497：当前log文件中的第几条信息，存放在磁盘上的那个地方

上述图中索引文件存储大量元数据，数据文件存储大量消息，索引文件中元数据指向对应数据文件中message的物理偏移地址。

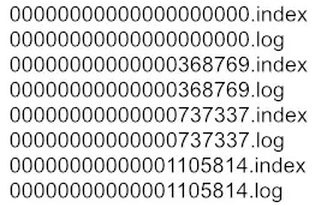
其中以索引文件中元数据3,497为例，依次在数据文件中表示第3个message(在全局partiton表示第368772个message)、以及该消息的物理偏移地址为497。

* segment data file由许多message组成， qq物理结构如下：

| **关键字** | **解释说明** |
| --- | --- |
| 8 byte offset | 在parition(分区)内的每条消息都有一个有序的id号，这个id号被称为偏移(offset),它可以唯一确定每条消息在parition(分区)内的位置。即offset表示partiion的第多少message |
| 4 byte message size | message大小 |
| 4 byte CRC32 | 用crc32校验message |
| 1 byte “magic" | 表示本次发布Kafka服务程序协议版本号 |
| 1 byte “attributes" | 表示为独立版本、或标识压缩类型、或编码类型。 |
| 4 byte key length | 表示key的长度,当key为-1时，K byte key字段不填 |
| K byte key | 可选 |
| value bytes payload | 表示实际消息数据。 |

### 11.3、Kafka 查找message

读取offset=368776的message，需要通过下面2个步骤查找。



#### 11.3.1、查找segment file

00000000000000000000.index表示最开始的文件，起始偏移量(offset)为0

00000000000000368769.index的消息量起始偏移量为368770 = 368769 + 1

00000000000000737337.index的起始偏移量为737338=737337 + 1

其他后续文件依次类推。

以起始偏移量命名并排序这些文件，只要根据offset \*\*二分查找\*\*文件列表，就可以快速定位到具体文件。当offset=368776时定位到00000000000000368769.index和对应log文件。

#### 11.3.2、通过segment file查找message

当offset=368776时，依次定位到00000000000000368769.index的元数据物理位置和00000000000000368769.log的物理偏移地址

然后再通过00000000000000368769.log顺序查找直到offset=368776为止。

## 12、Kafka自定义Partition

见代码