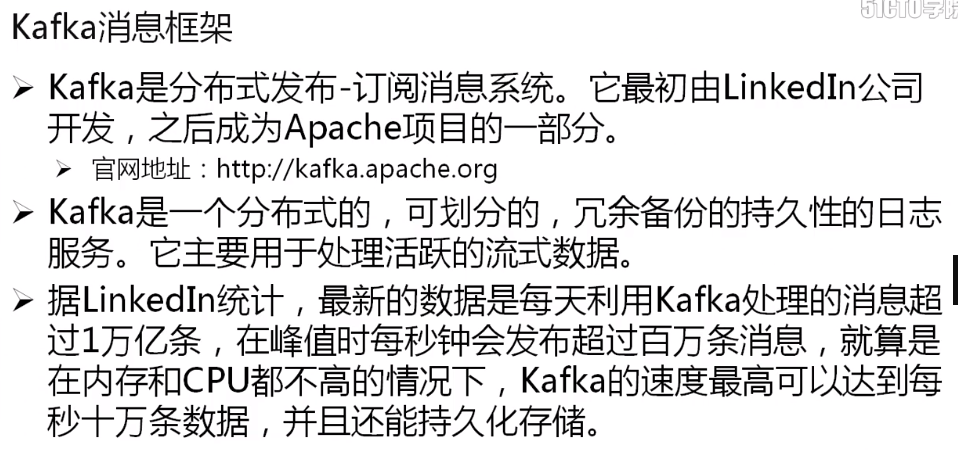
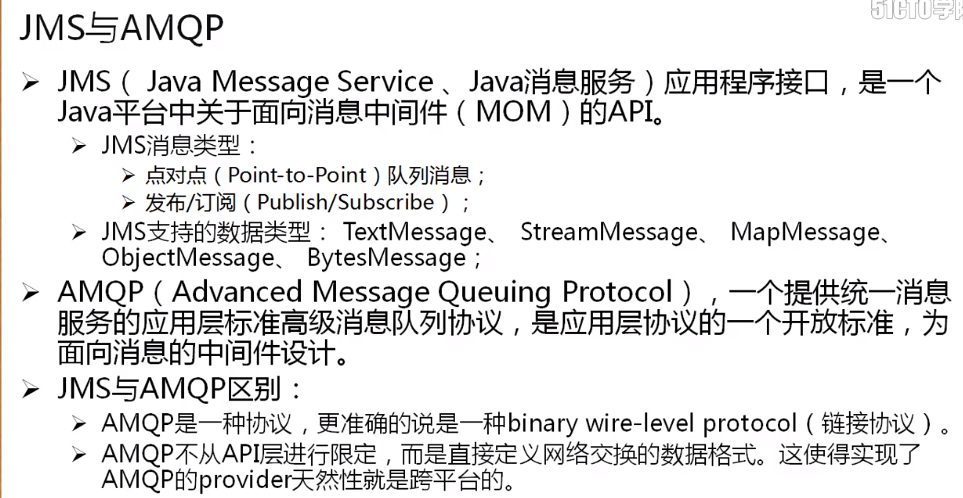
# kafka消息中间件

## kafka简介

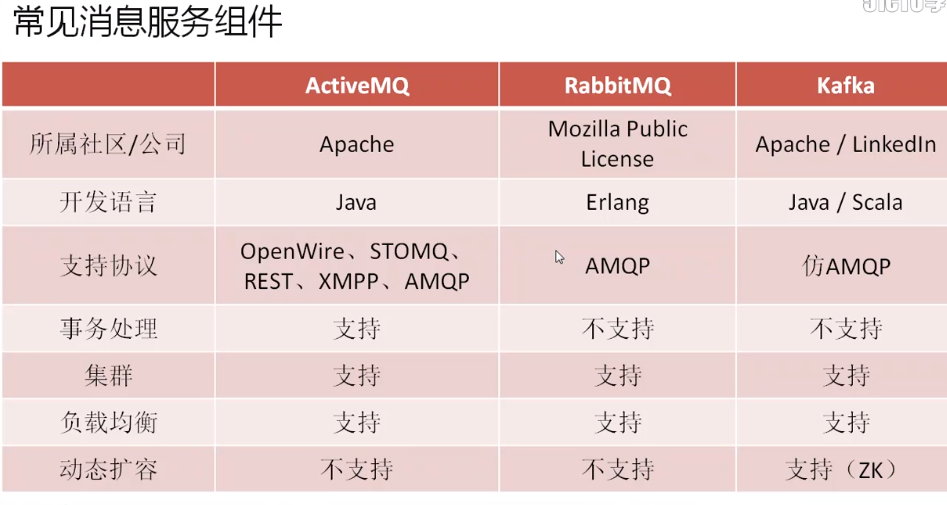
### kafka简介

kafka可以说是现在所有开源的消息组件之中性能最高的一款产品。





如果说到消息队列，那么首先想到的就是就是jms（JMS属于java消息服务，这就是java原生的操作协议），JMS代表是的开源项目：ActiveMQ，而AMQP的两大代表组件rabbitmq和kafka。



AMQP是一种不受程序限制的传输的处理协议，而JMS收到程序限制，所以AMQP性能和适应性会更高，但是kafka作为一种AMQP的实现有一个最重要的特征：

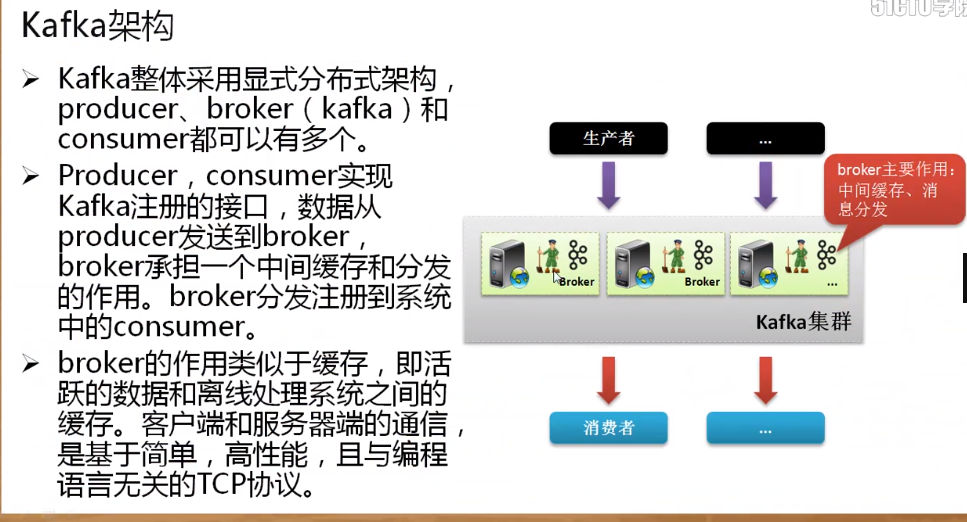
1. rabbitmq、activemq有一个特点，消息消费完成就被删除；
2. kafka特点：所有的消息会自动保存两天的时间，不用的消息才会被删除。

### kafka工作原理

kafka是一款性能很高的消息组件，但是不管如何改变，对于消息组件本身其最基础的组成部分：

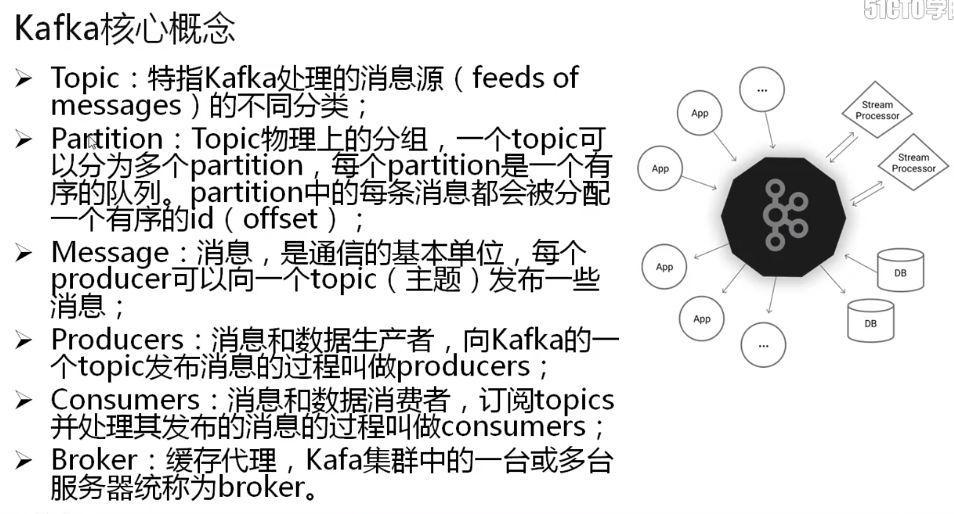
1. 消息的生产者：负责进行消息的推送，推送给指定的服务器；
2. 消息的消费者：负责通过服务器获取消息的内容；
3. 消息服务器中间件（服务器，比如broker）：负责消息的存储，也就是说当消费者来不及处理完全部消息的时候，可以在消息的中间件中进行消息内容的缓冲，所以消息中间件也往往称为消息队列中间件。

要想整个程序高性能运行的关键性因素：程序设计要合理、CPU要快、内存要大、磁盘转速要快（寻址是成为性能最大的瓶颈），对于消息组件最快的做法就是网络传输也要快，而kafka设计里面，将所有可能影响到程序性能的部分全部都考虑到了。

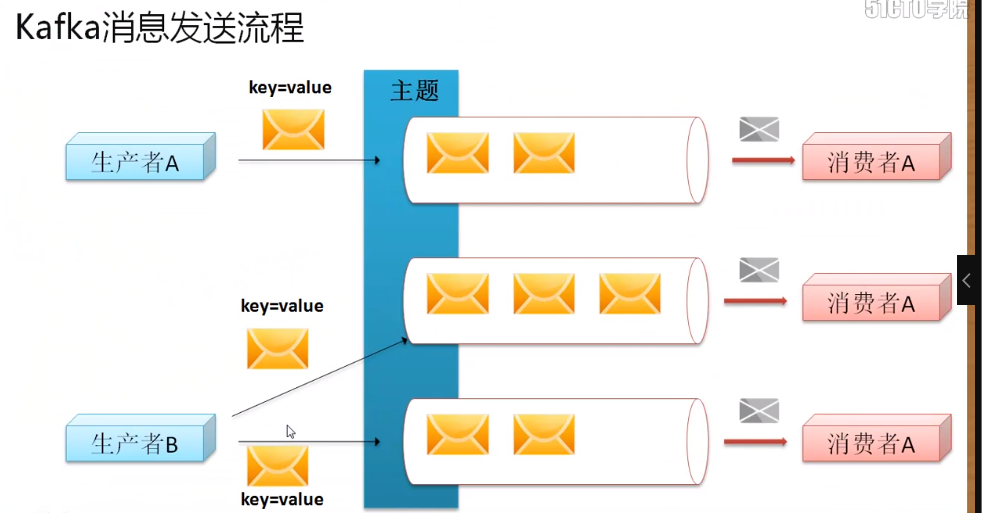


kafka是基于Zookeeper设计所以对于kafka集群设计来讲实现就相对容易许多，同时zookeeper可以保存所有集群主机的信息内容，这也就是说在配置kafka之前一定首先进行Zookeeper配置。

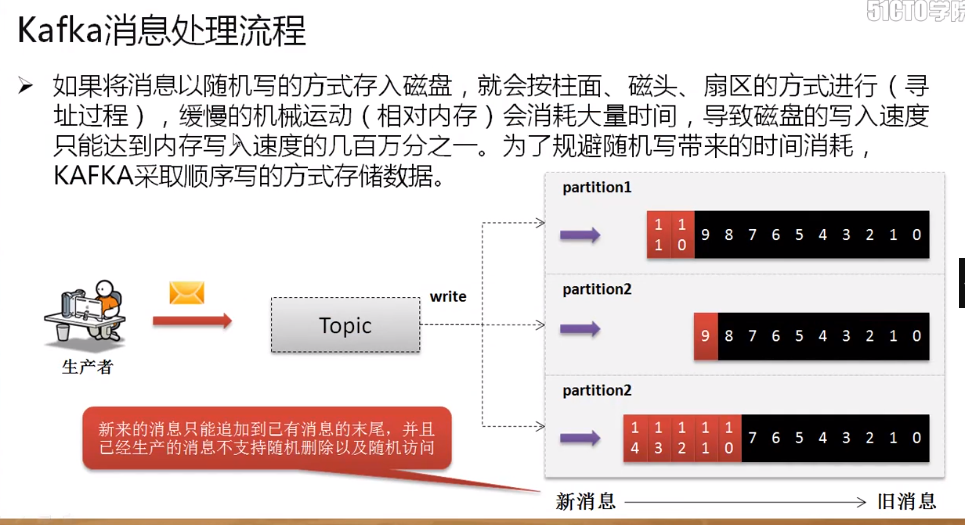
如果要想进行消息的处理，很明显所有的消息组件都一定会提供有一个消息主题，所有的消息生产者根据主题将自己的消息发送并且保存到服务器之中，而消费者也可以通过指定的主题获取消息内容。这样就可以传递多种消息。



Partition：指的是分区，如果你现在配置的主机只是单核CPU，那么你能够进行的合理的分区划分，也只能够有一个分区，但是如果cpu的核心数可能很多，比如16核，那么这台服务器上可以进行的分区操作可以划分出16个分区，在每台服务器上可以有多个分区，而分区划分的最简单的依据：就是根据CPU的性能来决定。也就是说在整个kafka集群里面，所有的分区数量=主机的CPU内核数量。



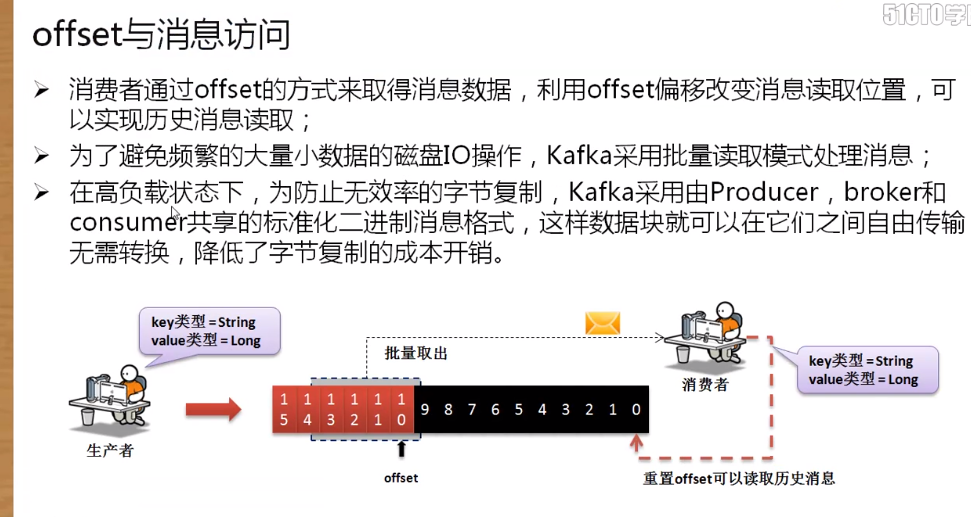
在kafka之中消息的发送一定要依据主题进行划分。而每一个主题为了让消息处理更快，专门设置多个分区，就好比一件工作，一个人干绝对比三个慢许多。同时在整个kafka里面最新的版本之中支持有key=value结构传输，这样的传输模式对于消费者而言更加容易处理数据。在进行消费者设计的时候，你的消费者可以使用的数量就是分区的数量，也就是说如果设置了三个分区，那么就可以使用三个消费者，反之如果这设计了一个分区，那么只能够有一个消费者。



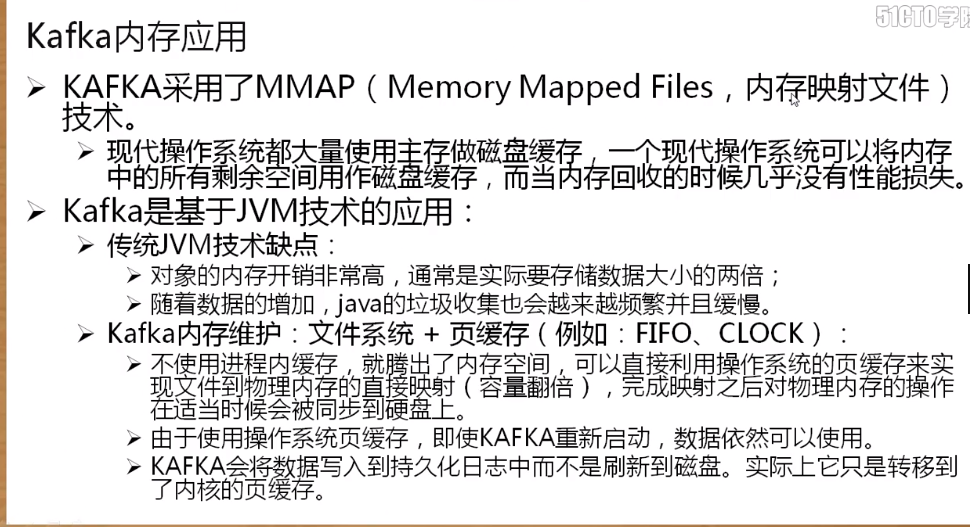
如果在进行信息写入的时候，所有的磁盘中的数据保存采用随机的方式存储，那么兑取的时候就会产生性能瓶颈，因为会出现寻址变慢的情况。

在kafka里面还有一个比较逆天的性能（也是迫切需要的），传统的JMS设计的时候会存在有一个缺陷：当消息消费后会自动被删除，而kafka则将被消费的消息暂存2天，2天后在删除。在这样的情况下为保证kafka的读取性能，单独设计了一个offset，可以理解为当前要操作的消息的下表，如果要读取历史消息，只需要修改offset的指向即可。在于一些组件整合的过程中，需要考虑offset的设计，如果设计不当就会造成历史消息重复消费的问题。

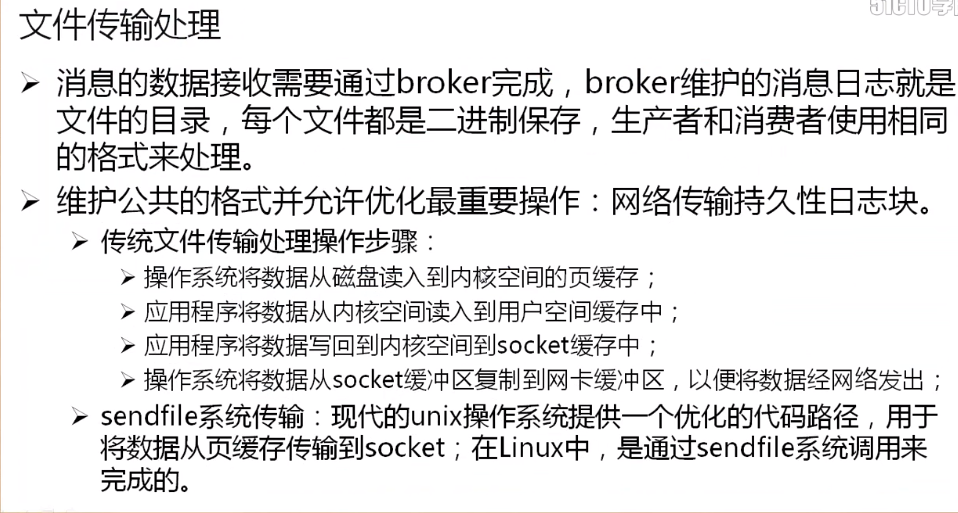
在磁盘中如果要不断的进行各种细小的琐碎操作，那么性能就有可能下降，所以在kafka里面有批量的数据操作，也就是说所有需要消费的数据会批量读取，这样就减少了磁盘的造作量。

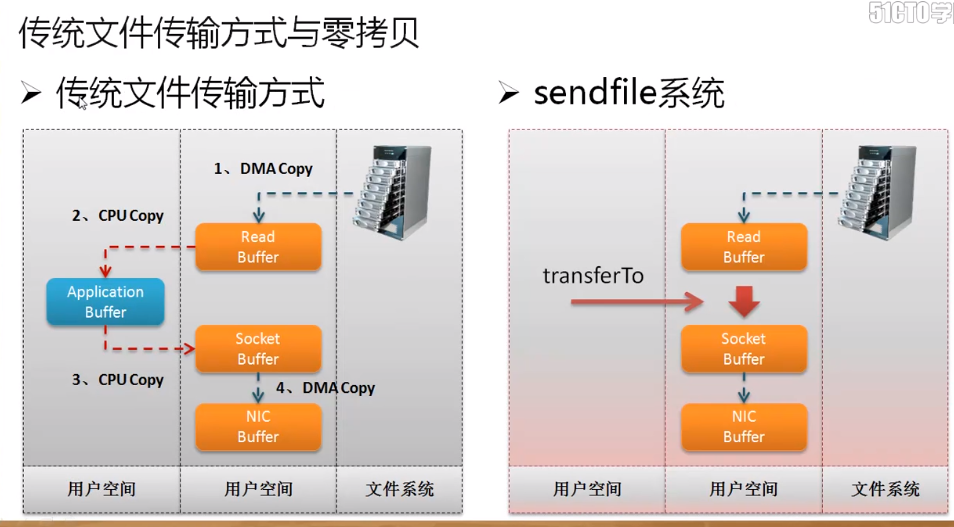


在很多的消息系统里面由于可以传输的数据类型比较少（字符串），所以在每一次消费的时候都需要去判断数据的类型。自然会造成时间复杂度的提升，为解决这样的问题，kafka约定了，消息的生产者与消费者一定要协商好消息的数据类型。



kafka是基于jdk的实现，所以在kafka之中对于内存要想发挥高效，就不能纯粹依靠jvm进行管理，多以kafka还会使用到操作系统的内存空间，这样即使kafka崩溃了，但是数据不在jvm里面，数据也可以重新立刻恢复。

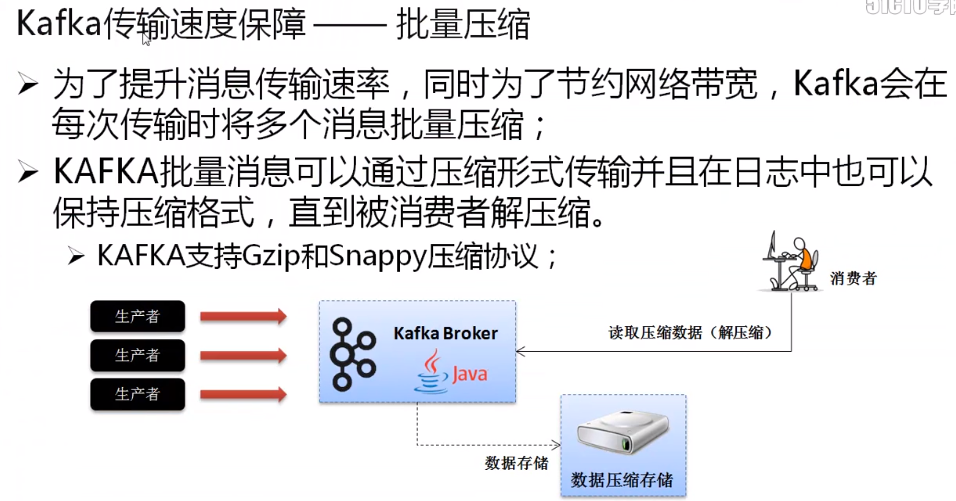


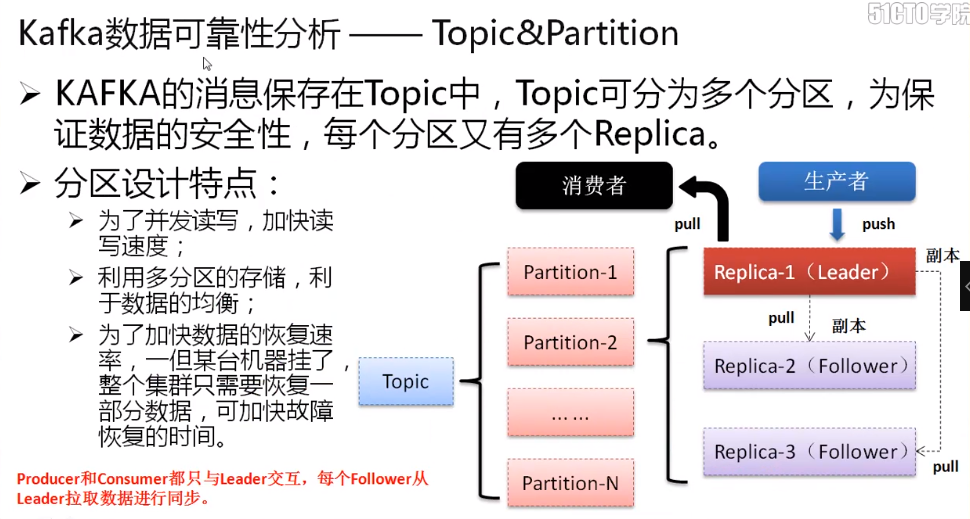


正式中间少了CPU的处理环节，所以执行性能更高。而这样的操作形式在kafka之中成为“零拷贝”

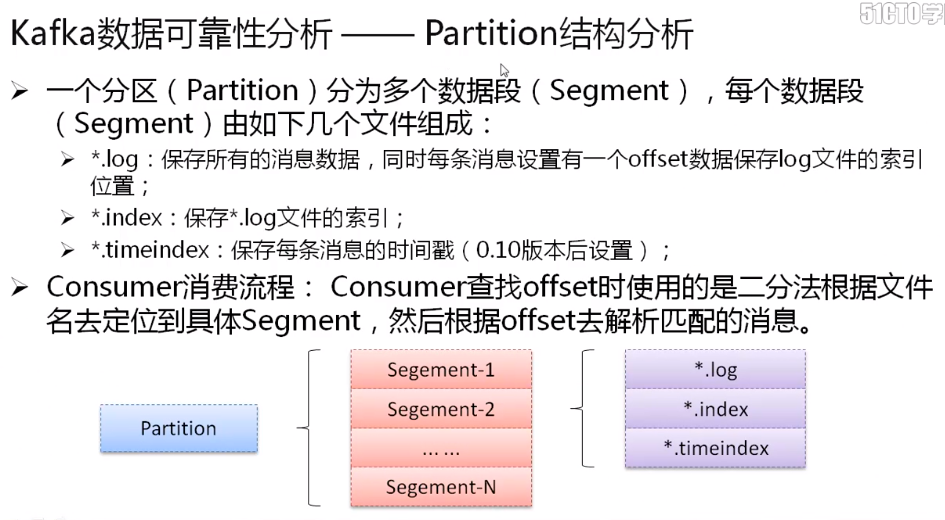


现在所有的设计都是围绕着kafka服务器本身的优化，但是关键性因素还包括有网络传输。





当有多台kafka服务器的时候，就可以进行副本的设计，也就是说当有三个副本时，这三个副本应该推举出一个leader，有两个flower。所有的跟随者通过leader进行数据的抓取。生产者将数据交给leader，而消费者通过leader读取数据。当leader出现问题后，剩余两个flower将会推举出新的leader。



总结kafka高性能的实现模型：

1. 采用零拷贝技术，让数据传输更加迅速；
2. 采用批量的数据读取，减少磁盘的io操作，可以提升性能
3. 为保证历史消息继续可以被消费，提供有一个offset指向，通过指向来负责消息的读取顺序
4. 网络的传输采用数据压缩格式，所以传输更快，占用的带宽越少
5. kafka中的数据可以设置副本，这样可以在出现问题后，依然可以保证数据的有效性

### kafka基础配置

1. 将kafka开发包进行解压缩：

tar -zxvf kafka\_2.12-1.1.0.tgz -C /usr/local/

1. 为操作方便，将解压缩后得到的文件夹进行更名处理：

mv kafka\_2.12-1.1.0/ kafka

1. kafka本身依赖的是Zookeeper，在kafka开发包里面本身就提供了Zookeeper支持命令；但是为数据保存的方便建议建立两个文件夹：

mkdir -p /usr/data/{zookeeper,kafka}

1. 编辑Zookeeper.properties配置文件

vi /usr/local/kafka/config/zookeeper.properties

配置Zookeeper数据保存目录：

dataDir=/usr/data/zookeeper

1. 进行kafka配置文件的定义：

vi /usr/local/kafka/config/server.properties

配置数据保存目录：

log.dirs=/usr/data/kafka

kafka默认的访问端口设置为90992，但是一般在实际的使用之中，往往会分配其他的开发端口，本次设置为9095。

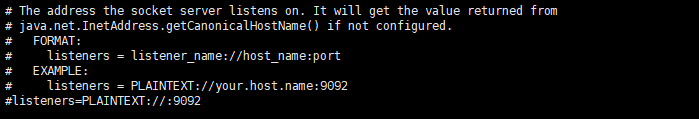
advertised.listeners=PLAINTEXT://192.168.1.213:9095

port=9095

设置外网访问IP地址（端口号相同）：

advertised.listeners=PLAINTEXT://192.168.1.213:9095

以下为内访问的配置：



1. 启动kafka服务，kafka依赖于Zookeeper。
2. 启动kafka内置Zookeeper服务（为观察方便先不后台启动）：

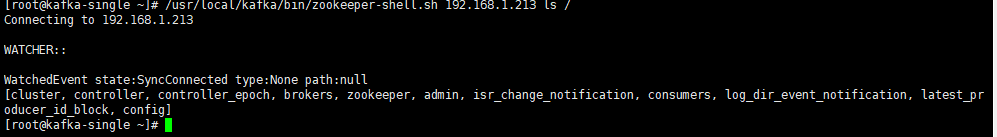
/usr/local/kafka/bin/zookeeper-server-start.sh /usr/local/kafka/config/zookeeper.properties

1. 启动kafka服务：

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

1. kafka启动之后会自动在Zookeeper里面进行信息的注册，但是如果现在使用的是kafKa内置的Zookeeper，则这些数据要想查看就比较麻烦了（kafka提供了自己的Zookeeper操作命令）；

/usr/local/kafka/bin/zookeeper-shell.sh 192.168.1.213 ls /



如果做好主机ip映射的话，可以用主机名称代替ip

8、如果用户想要进行kafka操作，那么一定要创建若干主题：

/usr/local/kafka/bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic yangmi-topic

此主题关联本机的Zookeeper，主题有一个副本，一个分区，主题名称为yangmi-topic

9、查看所有的主题信息：

/usr/local/kafka/bin/kafka-topics.sh --list --zookeeper localhost:2181

1. kafka内部提供有测试环境，可直接利用指定的命令进行消息的生产者与消费者的通信
2. 启动kafka的消费者：

不接受历史消息：

/usr/local/kafka/bin/kafka-console-consumer.sh --bootstrap-server 192.168.1.213:9095 --topic yangmi-topic

接收历史消息：

/usr/local/kafka/bin/kafka-console-consumer.sh --bootstrap-server 192.168.1.213:9095 --topic yangmi-topic --from-beginning

1. 启动kafka消息发送者：

/usr/local/kafka/bin/kafka-console-producer.sh --broker-list 192.168.1.213:9095 --topic yangmi-topic

回车后输入内容，按回车键发送，即可在消费者的命令行接收到。

## java操作kafka

如果现在要使用java访问kafka服务，则必须有一个前提：允许外网访问。

1. 对于kafka开发，开发包本身已经提供有驱动支持库文件，但考虑到实际开发应用，建议使用maven进行项目管理。
2. 修改pom.xml文件追加相关的程序依赖包。

*<?***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.ym.com</**groupId**>  
 <**artifactId**>mykafka</**artifactId**>  
 <**version**>1.0-SNAPSHOT</**version**>  
  
 <**properties**>  
 <**project.build.sourceEncoding**>UTF-8</**project.build.sourceEncoding**>  
 <**jdk.version**>1.8</**jdk.version**>  
 </**properties**>  
 <**dependencies**>  
 <**dependency**>  
 <**groupId**>junit</**groupId**>  
 <**artifactId**>junit</**artifactId**>  
 <**version**>4.12</**version**>  
 <**scope**>test</**scope**>  
 </**dependency**>  
  
 <**dependency**>  
 <**groupId**>org.slf4j</**groupId**>  
 <**artifactId**>slf4j-log4j12</**artifactId**>  
 <**version**>1.7.25</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.slf4j</**groupId**>  
 <**artifactId**>slf4j-api</**artifactId**>  
 <**version**>1.7.25</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.apache.kafka</**groupId**>  
 <**artifactId**>kafka-clients</**artifactId**>  
 <**version**>1.1.0</**version**>  
 </**dependency**>  
 </**dependencies**>  
 <**build**>  
 <**finalName**>mykafka</**finalName**>  
 <**plugins**>  
 <**plugin**>  
 <**groupId**>org.apache.maven.plugins</**groupId**>  
 <**artifactId**>maven-compiler-plugin</**artifactId**>  
 <**configuration**>  
 <**source**>${jdk.version}</**source**>*<!-- 源代码使用的开发版本 -->* <**target**>${jdk.version}</**target**>*<!-- 需要生成的目标class文件的编译版本 -->* <**encoding**>${project.build.sourceEncoding}</**encoding**>  
 </**configuration**>  
 </**plugin**>  
 </**plugins**>  
 </**build**>  
</**project**>

在kafka里面分为消息的生产者以及消息的消费者

### 使用java定义生产者

1. 定义生产者

**public class** SendMessageProducer {  
 **public static final** String ***SERVERS*** = **"192.168.1.213:9095"**;  
 **public static final** String ***TOPIC*** = **"yangmi-topic"**;  
  
 **public static void** main(String[] args) {  
 *//如果要想进行kafka消息发送需要使用properties定义一些属性环境* Properties properties = **new** Properties();  
 *// 定义Kafka服务地址* properties.setProperty(ProducerConfig.***BOOTSTRAP\_SERVERS\_CONFIG***, ***SERVERS***);  
 *// Kafka之中是以key和value的形式进行消息的发送处理， 所以为了保证Kafka的性能，专门提供有统一类型  
 // props.setProperty(ProducerConfig.KEY\_SERIALIZER\_CLASS\_CONFIG, "org.apache.kafka.common.serialization.StringSerializer") ;* properties.setProperty(ProducerConfig.***KEY\_SERIALIZER\_CLASS\_CONFIG***, StringSerializer.**class**.getName());  
 properties.setProperty(ProducerConfig.***VALUE\_SERIALIZER\_CLASS\_CONFIG***, IntegerSerializer.**class**.getName());  
 **long** start = System.*currentTimeMillis*();  
 *//定义消息发送对象，依靠此对象可以进行消息的传递* Producer<String, Integer> producer = **new** KafkaProducer<String, Integer>(properties);  
 **for** (**int** x = 0; x < 100000; x++) {  
 producer.send(**new** ProducerRecord<String, Integer>(***TOPIC***, **"杨幂-"** + x, x));  
 }  
 System.***out***.println(**"\*\*\* 消息发送完成，耗时"** + (System.*currentTimeMillis*() - start));  
 producer.close();  
 }  
}

1. 在kafka中可以使用slf4j进行日志和的显示，顒如果要想查看日志，则可以修改pom.xml文件，追加日志显示:

<**dependency**>  
 <**groupId**>org.slf4j</**groupId**>  
 <**artifactId**>slf4j-log4j12</**artifactId**>  
 <**version**>1.7.25</**version**>  
</**dependency**>  
<**dependency**>  
 <**groupId**>org.slf4j</**groupId**>  
 <**artifactId**>slf4j-api</**artifactId**>  
 <**version**>1.7.25</**version**>  
</**dependency**>

1. 在项目中配置一个log4j.properties配置文件：

*# For JBoss: Avoid to setup Log4J outside $JBOSS\_HOME/server/default/deploy/log4j.xml!  
# For all other servers: Comment out the Log4J listener in web.xml to activate Log4J.***log4j.rootLogger**=**INFO, stdout, logfile  
  
log4j.appender.stdout**=**org.apache.log4j.ConsoleAppender  
log4j.appender.stdout.layout**=**org.apache.log4j.PatternLayout***# log4j.appender.stdout.layout.ConversionPattern=%d %p [%c] - %m%n***log4j.appender.stdout.layout.ConversionPattern**=**%p [%c] - %m%n***# log4j.logger.cn.mldn.amr.mapping=TRACE***log4j.appender.logfile**=**org.apache.log4j.RollingFileAppender  
log4j.appender.logfile.File**=**logs/my.log  
log4j.appender.logfile.MaxFileSize**=**512KB***# Keep three backup files.***log4j.appender.logfile.MaxBackupIndex**=**3***# Pattern to output: date priority [category] - message***log4j.appender.logfile.layout**=**org.apache.log4j.PatternLayout  
log4j.appender.logfile.layout.ConversionPattern**=**%d %p [%c] - %m%n**

### 使用java定义消费者

**public class** ReceiveMessageConsumer {  
 **public static final** String ***SERVERS*** = **"192.168.1.213:9095"**;  
 **public static final** String ***TOPIC*** = **"yangmi-topic"**;  
  
 **public static void** main(String[] args) {  
 Properties properties = **new** Properties();  
 *// 定义消息消费者的连接服务器地址* properties.setProperty(ConsumerConfig.***BOOTSTRAP\_SERVERS\_CONFIG***, ***SERVERS***);  
 *// 消息消费者一定要设置反序列化的程序类，与消息生产者完全对应* properties.setProperty(ConsumerConfig.***KEY\_DESERIALIZER\_CLASS\_CONFIG***, StringDeserializer.**class**.getName());  
 properties.setProperty(ConsumerConfig.***VALUE\_DESERIALIZER\_CLASS\_CONFIG***, IntegerDeserializer.**class**.getName());  
 properties.setProperty(ConsumerConfig.***GROUP\_ID\_CONFIG***, **"group-1"**);  
 *//定义消费者处理对象* Consumer<String, Integer> consumer = **new** KafkaConsumer<String, Integer>(properties);  
 consumer.subscribe(Arrays.*asList*(***TOPIC***));*// 设置消费者读取的主题名称  
 // 消费者需要一直进行数据的读取处理操作* **while** (**true**) {  
 ConsumerRecords<String, Integer> allRecorders = consumer.poll(2000);  
 **for** (ConsumerRecord<String, Integer> records : allRecorders) {  
 System.***out***.println(**"key="** + records.key() + **",value="** + records.value());  
 }  
 }  
 *// consumer.close();* }  
}

### kafka操作深入

## SSL认证

## kerberos认证

## 搭建kafka集群

## Springkafka支持

## 整合flume框架