我们刚刚学习完了flume是吧？flume就是一个分布式的，然后可靠的并且还实践了一些高可用的机制，就是那个负载均衡等等这种方式的一个海量的日志收集系统，然后在这个flume里面它可以支持定制数据的一些发送方来收集一些数据，并且这个flume你也看到了它可以对这个数据进行一些个简单的处理，然后并且把数据写到相应的数据接收方，这个是我们刚学完flume的一个大概的总结，那它的主要的定位就是一个消息的一个前端采集，通常会见到这样的一个组合，就是Flume+kafka或者是加上一个storm再加上一些个HDFS或者hbase这样的一个经典组合。那既然在这个组合里面我们Flume学完了，storm也学完了，HDFS也学完了，hbase也一样学完了，那么这里面就差一个kafka作为一个衔接怎么进行一个承上启下的作用对吧？Flume是一个分布式的实时采集，那kafka是对采集后的数据进行一个分布式的缓存，然后缓存之后的数据供后续的组件进行一些个处理和使用，那这里边针对这个数据具体怎么使用还是处理还是怎么样做一些个挖掘那都是交给后续的更灵活更通用的集成框架来解决的，那前提是由flume和kafka把数据先喂到后面这些集成框架里面来，所以kafka在整个的组合架构里面，它属于是一个承上启下这么一个管道的作用，那通常很多人把它定位成消息队列。

接下来我们就开始看一下整个的一个kafka的学习（如下图1所示）

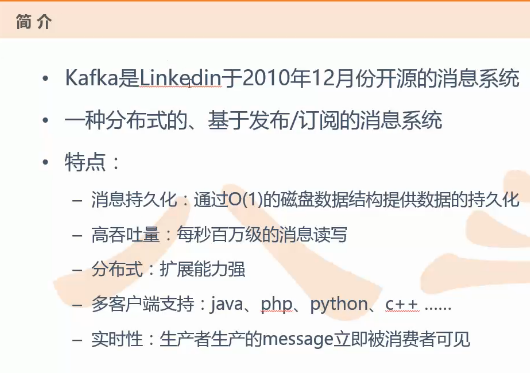
图1：



看看kafka自身单独从这个kafka这个角度然后怎么去做一些个实验，并且和你的flume建立起关系，flume的数据和kafka进行个上下游的对接，然后并且看一下kafka怎么和storm怎么做一个上下游的对接，这是整个kafka的内容。

那这块简介我们就快速过一下（如下图2所示）

图2：



Kafka就是一个消息队列系统，那它可以支持一些水平扩展和一些高吞吐力，所以被大家广泛的使用，并且现在越来越多的开源分布式系统已经和kafka进行了一些集成，比如说你的storm,spark等等，首先kafka也可以是想象成一个机器，也可以做出一个队列也是可以的，还可以像flume一样扩展成多个机器共同协同工作变成集群分布式的方式来完成，那它来完成的主要是什么功能呢？就是一个对外服务，这个服务就叫一个发布和订阅，那什么是发布和订阅呢？就像是我们每天很多人都在用微信，微信上有一个叫订阅号，那你就会去订阅那些关于美食的一些公众号，或者订阅一些邮件等等，然后一旦这个公众号或者这个邮件它有最新的一些新闻或者推广或者一些新的信息的发布，这个时候你就可以去跟着它订阅的信息然后不断的去了解它最新的一个动态，大概是这么一个意思。

但是这里面拿邮箱来举例子或许更好一点，比方说那你去订阅了一些邮件，比如说某一个第三方信息提供方，它一旦有最新的动态就会往你的邮箱里面去发送一个消息，就跟你发一封邮件对吧？但是这个时候你不一定说它提供信息的同时你马上就能够去阅读这个数据对吧？有可能它是咋天发送给你的邮件，可能你今天刚刚有时间去浏览一下我的邮箱，发现它给我发了一个新的邮件，发现了它的数据有些更新，有一些新的很好的一些观点以及新闻来供我阅读，所以你会发现这个方式有点像异步的方式，不是同步的对吧？那这个邮箱有点类似于是一个缓冲区，邮箱就类似于一个存储，首先它相当于是把你的信息放到存储里面，然后用户去异步的方式来从这个缓冲区里面去得到相应的信息。

然后我们来看一下它有哪些特点，这个特点的话有很多，其中有一个很重要的特别就是说它的消息可以持久化，持久化就是说我们之前在讲storm的时候也说了这个消息持久化这么一个概念，就是说这个数据可以落地的是可以存到磁盘里面，但是kafka和storm的区别就是你storm的数据，如果你是不额外的添加一些外部存储的话，那它默认的情况下它的数据都是在内存级别的进行一个数据的传递，但是kafka不一样，kafka的数据并不是说先把你的数据先写到内存，而是它的数据直接往磁盘上去写，那么这样大家就会有疑问了，你直接把数据接收到的数据直接往磁盘上去写，那这样会不会很慢啊？那这个时候相当于kafka在这个存储的过程中它做了大量的优化，我们大概普遍的认为kafka其实非常非常严重的依赖于你的文件系统或者是依赖于你这个磁盘，那大家普遍认为这磁盘是很慢的对吧？那因为我们对这个持久化这三个字一直理解成就是说对这个性能会有一种抱有怀疑的态度，虽然它能够存储大量的数据，但是你毕竟有这个文件的IO可能会影响到一些性能一些效率上的问题是吧？那其实事实上和我们大家的期望相比，其实磁盘来说如果你把它用的好的话，其实效率反而更高，要比用内存的访问方式更加的迅速，就是说你要是这个磁盘能够优化的好的话，那磁盘它带给你的性能的一个保证可以和网络一样快，这个得益于一个操作系统的两个特点，一个是预读功能，还有一个是后写功能。

什么是预读呢？就是说我在浏览这篇文章的时候当我们浏览到了这一行数字的时候，因为我是从上往下去阅读的，当我阅读到某一行的时候，那这个操作系统会大概率的会认为我要马上读下面那条记录的可能性更高是吧？所以在我读这个数据的时候它会提前把下面的记录提前帮我把这个数据读出来了，这就是一个预读功能。

后写就是说我们会往这个系统上会有大量的写，如果你是每一次请求你去写一次的话，比如说我们写一千字，你可能对这个数据有一千次的操作对吧？那假如说我们把这一千次看看能不能优化成一次，用一次代替一千次，这样的话会大大的提升了一些写入的效率，那如果这种情况下，其实你的对于磁盘来访问，特别是顺序来访问要比你随机的内存访问还要快，所以这里面有一个特点，就是对磁盘的顺序访问，一定要是顺序访问，如果你不是顺序访问的话，那你这个预读功能和后写功能就不存在了，它这样的优化就不存在了，这种优化是这种操作系统级别的，对磁盘的一个顺序访问要比对内存随机访问还要快，所以由于这么一个特点，所以它kafka就对这个磁盘数据的一个持久化做了一个优化，就可以做到我O1的方式进行一个数据的处理。

接下来就是高吞吐量，这个了解一下就可以，有相应的这个数字可以批入，就是说它每秒可以处理上百兆的数据。

扩展能力强，因为它也可以用集群的方式

然后对于语言也是很开放的，你可以用很多的语言进行对kafka的一个数据的写入，读出等操作。

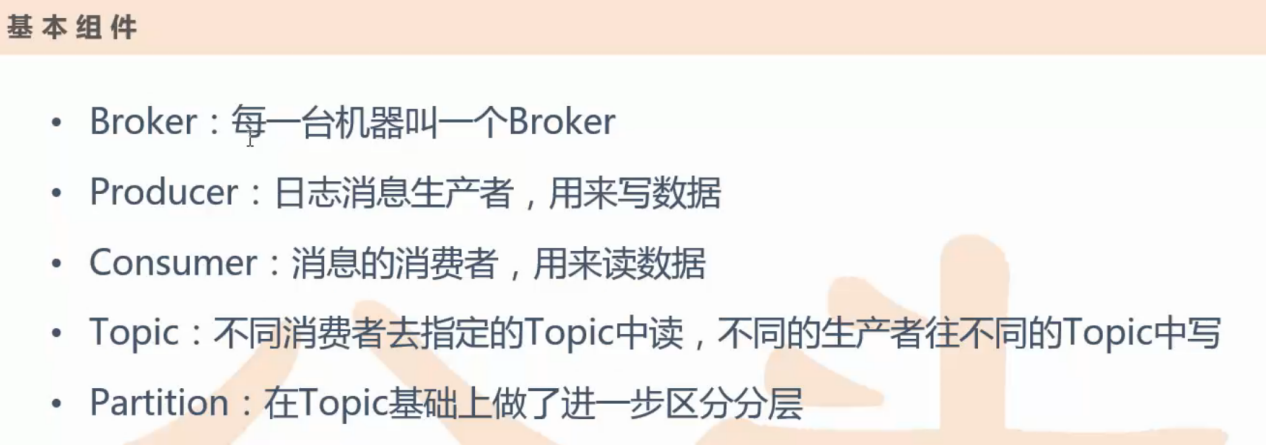
还有一个message是什么意思呢？说到message的话，就是kafka的一个数据单位，所以实时性这块很好理解，我们突然想到的是这种storm这种集成框架或者其它的这种实时性的流式处理这些框架，那其实这里面还有另外一个角度来去思考这样的问题，就是说我们通常要去做一个数据统计或者是一个报表分析，通常是要对这种离线的方式进行处理，我们要处理离线数据，然后把这些数据进行统计然后通过hbase或者hive的方式来进行一个统计的效果，这个是可以做到，因为之前我们刚刚学到的是怎么样去做一个数据分析统计，那当然这种方式也是比较成熟，但是它有一个问题是它的延迟太大了，它相当于是对你离线的数据做批量的处理，所以它的实时性是得不到保证的，并且你想维护这么一套离线整个的一个架构也好还是框架也好，运营成本也是很高的，这是一个问题。

还有一个就是说我们从实时性的角度如果现有的一个消息队列系统的话，我们现有的消息队列系统其实有很多，那这些大量的这种目前这种消息队列系统，其实它们的目标大部分的消息队列这种系统它们的目标都是集中在了实时这两个字眼上，但是对于这种持久化，对于这种大量的消息队列系统来说它们做的其实并不是特别好，所以不管是从你离线的角度来看还是从你的实时性的角度来看没有一个特别完美的方案对吧？所以有没有一种架构能够把实时性的角度也做的很不粗，持久化也做的很不错，并且它也可以支持离线，当然为了实时性嘛，我也支持在线，所以相当于是一个八面玲珑的这么一个系统，那么有没有这样的一个消息系统呢？那这样的一个消息系统就是kafka。

Kafka目标就是成为一个队列平台，不仅支持离线还要支持在线，就是我这个消息队列的话，我不仅支持离线也好，那我在线支持也好，所以大家在去理解kafka的时候相当于对它有一个更新的认识，它不仅仅是一个实时流或者实时这么一套系统，它才适合应用，有的时候离线的方式也是可以来去运用的。

那接下来我们看一下怎么来去支持这么一套鲁棒的这么一个系统把，那在去了解整个系统之前，我们需要对一些概念做一个提前的准备（如下图3所示）

图3：



Broker就是可以认为是一个kafka的一个机器节点，一个机器就是代表一个Broker

那我们看一张图来进行理解（如下图4所示）

图4：



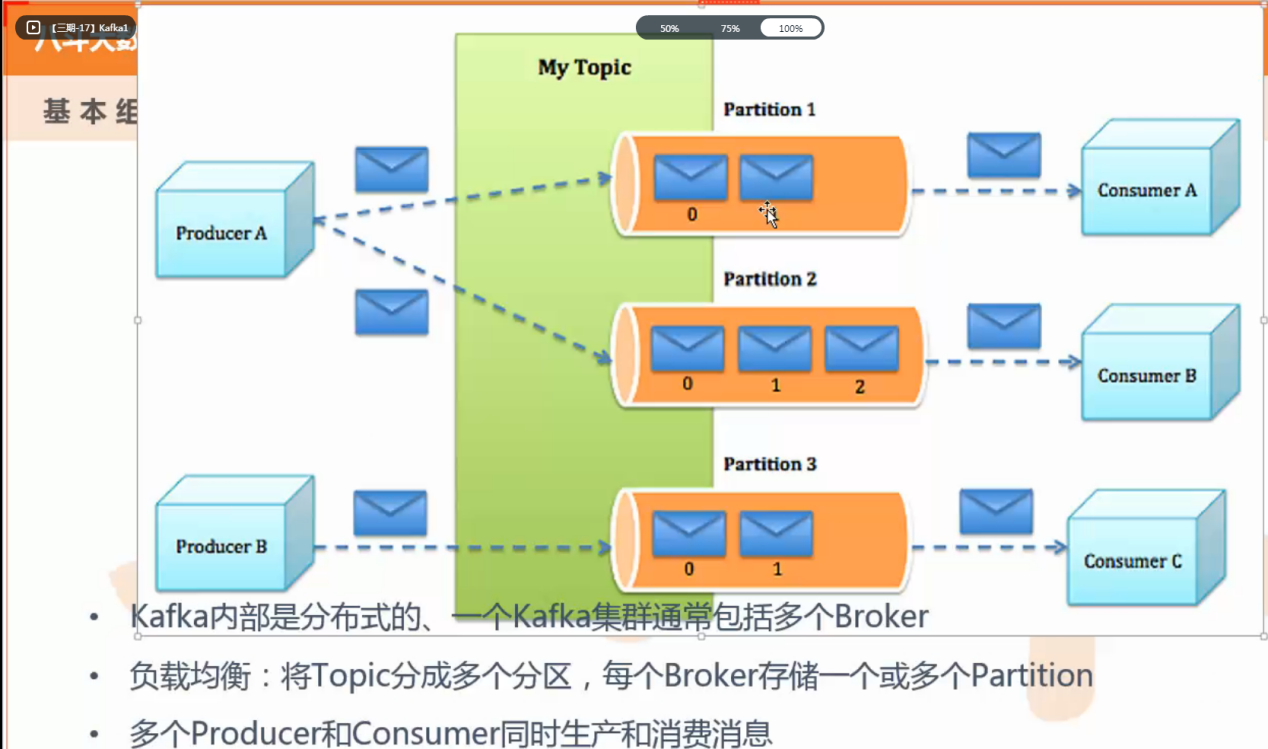
Producer是一个消息的生产者，然后Consumer是一个消息的消费者，所以Producer相当于是往你的kafkaserver去写数据的，那Consumer是从kafkaserver上去读数据的，是这么一个过程，所以这个kafkaserver叫做一个Broker，就是说前面这三个概念并不是说在一个节点上实现的，但你可以在一个节点上去模拟，但是实际的工作场景这三个角色的定位都是定位在不同的集群或者不同的机器上的。

那么你看Producer有一个日志消息生产者，这个日志这两个字容易误导大家，这个日志不是说我们之前想象的那种那些log或者一些警告的信息然后一些个提示性的错误等等这些系统运转的留下的一些可以追踪问题的一个记录，那这里面的log不是代表这个的，这里面的日志相当于就是这个message，就是代表着你kafka内部要处理的数据，它里面也叫日志，所以这个概念容易混淆。

那什么是Topic呢？比如说有一些人发布了一些话题，然后我们可能对这个话题比较感兴趣的人会提前做一个订阅，比如说这个话题是今天晚上吃什么可以不发胖？我相信很多人对这个话题很感兴趣是不是？那这个时候大家都去关心这个话题叫Topic，大家都去关注这个这个Topic，那相当于这个话题的发布者就有了很多很多的粉丝都在关注这个话题对吧？那这个时候这个话题的发布者在这个话题下面发布了一条新闻或者发布了一个文章，就是发布了一个新的信息，那这个粉丝的话都会获取到这样的一些信息对不对？那相当于你消息的发布者和你消息的接收者是通过Topic建立了关系，但是你消息的Producer和你的Consumer之间是有很多的Topic建立关系，不同的Topic是代表不同的消息的一个类别或者是一些不同的类型，把同一类的消息放在同一个Topic上面去，比如说同一个版主，这个版主它既喜欢美食又喜欢电影，那他可能最近又有什么新发布的一些做饭的技巧，可能在这美食这个Topic上进行更新文章，那最近有哪些电影上映了，然后它想提前给大家做一个剧透或者是对这个电影有什么观后感可能就是在这个电影的Topic上面去发布新的文章，那这个Topic其实从这个逻辑理解上来看，其实Topic是一个话题逻辑概念，就是Topic话题它是一个很虚幻的概念对吧？但是真正让这个topic物理落地是通过partition一个物理实现，是这么一个关系。

那partition是在topic基础之上做了进一步的区分分层，那一个topic的话是有一个或多个partition实现的，所以相当于对这句话又做了一个更进一步的解释和翻译，那我们看一下一个图，如下图5所示

图5：

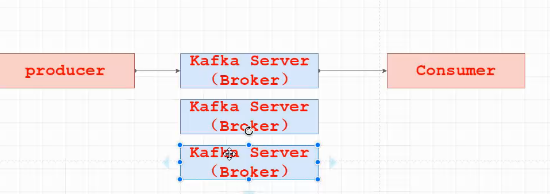


那在这个图5里面左边有Producer，右边是有Consumer是吧？中间这个像信封就是这个message对吧？然后这个蓝色的区域是代表了它的Topic，那我这一个Topic是可以有多个partition进行一个管理，就相当于是多个partition把它的消息合并到一块它才能够组成完整的Topic明白了把？就是说大家可能会第一次接触可能会认为，我的Topic一旦接收一个信息，那我这个信息我就往一个文件里面去写，那这个文件就跟你的Topic一一对应可不可以？但是为了提高这个效率，还是要把这个文件进行一个拆分，那怎么拆分呢？是按照partition的方式去拆分，那所以从这里面也有类似于一个负载均衡的概念，但是这个其实跟负载均衡不太一样，它这个不是负载均衡但是有点这个意思，就相当于它是更贴近于我们mapreduce里面partition，我mapreduce里面的partition是针对你前面传过来的key我去做一个哈希是吧？然后给它指定了一个桶id对不对？其实这里面也是一样的，相当于我上游来了一个消息，这个消息是属于这个Topic的，然后这个消息里面也会存有key和value，然后我这里面会有很多个partition这样的管道，那具体这个消息是往哪个管道里面去塞？也是根据我这个key也是同样做一个哈希去往指定的桶里面去写数据，然后这个时候我每个partition之间的数据是不可以重复的，另外你这个消息的也是有一个先来后到，我先来的消息那肯定是要先处理的是吧？我后来的消息肯定是要最后处理的，所以为了保证你的消息的顺序性，我要求你的消息能够前后的有这个顺序的一个约束，但是这个问题你想用多个管道来实现这个问题其实有些复杂。

如果你这个Topic和你这个partition是一比一的关系，那没有问题，你先来的数据我就在partition的前面，然后后来的数据在partition后面是吧？那我后面来读的时候相当于对这个先入先出队列进行读，肯定是保证一个消息的顺序性是不是？但是现在不一样了，你这个partition是变成了多个，变成了一个一比N的关系，那这个时候你全局上是不太好保证你的顺序性的，但是你从partition的内部来看，我这个黄色的管道里面内部已经有了很多的一些message了是不是？那这些message是在局部范围内是一个有序的，就是说你kafka它不太能够保证你去全局的一个顺序性，但是我partition的内部是可以严格保证你的顺序性的，那么既然partition是一个物理概念的话，它必然会在你某一个文件夹或者是目录下面能够找到相应的数据，那这个时候partition是怎么实现的呢？它是以一个文件夹的形式存在。

那这里面我想在进一步的跟大家说一下，假设有五个partition，然后有五个broker，broker就是一个server是吧？那这个时候是五个server，相当于是这就变成了五台机器了对不对？，那这kafka这个server其实它不仅仅是一个消息队列，那它也是可以进行一些存储，那你这个存储是存哪里了呢？是存到了你这个partition里面去了是吧？这个时候因为我这个prodecer你发不了一个消息，你这个消息肯定是要绑定一个Topic的，那这时候这个Topic的数据会散落在这些server上，就是说每一个Broker它只保留这个Topic的一部分数据（如下图6所示）

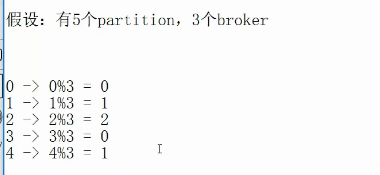
图6：



如果我这个topic我分了很多个partition，并且这里面会有很多的Broker，本来每一个partition就保存着这一个Topic的一部分数据是不是？好了那这个时候这些partition是怎么和你的kafkaserver或者broker进行一个分布的？那假设说我这里面就是一个5比5的关系，那么好了这里面很简单，那我每一个broker存一个partition就可以了，这是一比一的关系。

这个时候如果要是说我这里面要是有3个broker怎么分？ 这个分起来肯定是不会均匀的对吧？那假设说我这里面有五个partition（如下图7所示）

图7：



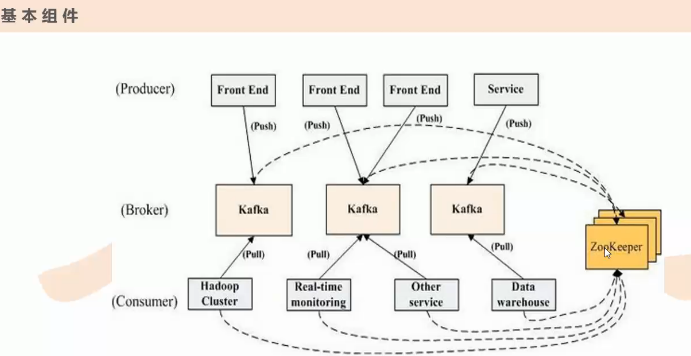
这个时候就算一下0要分到哪个机器上去？我们来算一下0%3等于0，那么就是等于它会存到0号的broker里面去，然后1这个partition的话就是1%3=1，那么就是等于它会存到1号的broker里面去，然后下面怎么算大家都应该很熟悉了是吧？

那这个时候这计算你就会发现我这前面这五个partition就会被通过这样的方式分到了三个broker里面去。

好了接下来我们再进一步的去深入，那kafka内部是一个分布式的，然后一个kafka集群包含了多个broker，就是说包含了多个机器，那负载均衡讲topic分成了多个分区，就相当于这个分区就是一个partition对吧？然后每一个broker是存储一个或者多个partition，当然如果你真要是前面有多个partition，又有多个broker，那么你不可能是一个broker保存了所有的partition，另外当这个管道一旦建好之后，那你这个prodecer可以去写，而Consumer是可以去读，你发现它这个写和读可以同时进行的，当然你写的这个信息可能在同一时刻你写的这个数据不一定是Consumer读到的信息，那这中间肯定有一定的消息延迟或者是一个异步的方式，那不管怎么说对于prodecer或者Consumer来说，我对于你的kafka来说都是一个客户端，就是说我对于kafkaserver来说，我不管我是prodecer或者Consumer或者kafka来说都是一个客户端，那既然都是客户端来说的话那我不管我写的数据有没有被人读，我就尽情的往里写就是了，那我读数据也是一样，我也不关心有没有生产者往里面写数据，我就尽情去读就可以了，我读到的数据我就读到了，读不到的数据呢那我就等呗对吧？所以相当于是一个kafkaserver把这个你的prodecer和Consumer生产者和消费者这两端进行一些个异步解耦大概是这么个思路。

好了那我们继续往下说，那kafka这个集群其实它和storm和hbase相似，它也依赖了一个zookeeper（如下图8所示）

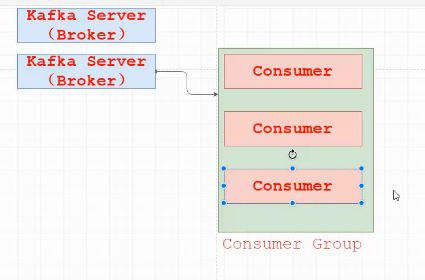
图8：



但是你从这个上图8里面会发现一个细节，这个zookeeper里面不是存数据嘛？但是zookeeper里面存的数据它只存你的broker的一些信息和Consumer的信息，你看这个虚线就是建立关系的，这个broker的信息是可以存到zookeeper里面的，Consumer的信息也可以存到你的zookeeper里面，但为什么你的Producer没有和zookeeper建立关系呢？是因为这个Producer它去做一个消息生产的时候，它是可以指定他是要往哪个broker上去写数据，这不需要你zookeeper来帮我去做一个分发，不需要依赖于zookeeper来搞定，我这Producer是可以自己来自主完成的一个写，但是Consumer来说是需要依赖于zookeeper完成一个负载均衡，就是这块是直接依赖于zookeeper来完成的一个负载均衡，broker和Consumer之间是有一个负载均衡的概念，那这个负载均衡是通过zookeeper来协调的，那Producer和broker之间是没有负载均衡的，但是这个producer是可以指定我需要往哪个partition上去写，然后一个典型的kafka集群就是里面会包含着若干个producer，然后这每一个producer它的接收的源可能不一样，有的producer是接收了web前端产生的一些配置也就是你的页面的一些浏览信息，有的producer是可能对接着你服务器的日志，那有的producer可能直接对接到你某一个系统的cpu，内存的一些监控级别的一些服务，那这个broker是可以水平扩展的，一般来说这个broker数量越多，这个集群的吞吐量就越高。

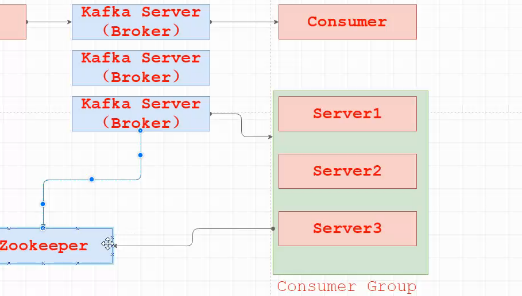
还有你的Consumer，就是kafka是通过zookeeper来进行一个集群管理配置并且它会在你这个Consumer里面会做一个负载均衡，这里面是怎么做一个负载均衡的呢？（如下图9所示）

图9：



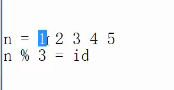
就是说我有多个Consumer，这每一个Consumer都代表着一个消费者，然后我可以把多个消费者都可以组成一个组的形式然后维护起来，然后这个时候你相当于是原来是有一个实体的Consumer来去消费你的消息，那这个时候可以把多个Consumer集中起来变成一个大的Consumer这个明白了把？就相当于对于kafka来说我有一个大的，我不关心你这里面是有多少个Consumer，我就把你当作一个多个Consumer变成一个组，变成一个小组，这个小组对于我kafka来说就是一个虚拟逻辑的概念，这个Consumer是实际存在的，这个外部Group是一个虚拟存在的，那它有这么一个细节，我每台机器都要去存储一些个数据，然后这个之前我们在讲hive的时候也提到过类似的思想，当时有一个分库思想还记得把？就是说我这每一个子节点只是存一部分的数据，因为我每一个节点都是存所有的数据的话，那我这个可能自身对于存储要求比较高，另外一个我没有办法去并行计算，这个时候我有多个server，我每个server之间只存我完整数据的其中一个部分，然后这个时候我每个server可以并发执行，如果把这个所有server的数据全都集中起来的话，其实这就是一个完整的数据集合，就没有做任何的数据的一些丢失。（如下图10所示）

图10：



那这个时候我kafka是往后传数据的，比如说我传了一个1，2，3，4，5，有5个message代表了有五个数据（如下图11所示）。

图11：



我其中第一个数据我发给了server1，第二个消息发给了server2，第三个消息发给了server3，我这怎么发送的呢？我也是可以比如说这是一个n，n=1，2，3，4，5，然后n%3去做一个取余数就可以了然后等于什么id是不是？我来了一个消息我就取一个模，然后我得到这个模之后我就知道你这个消息就应该被哪个Consumer来去消费，然后我这kafkaserver里面会有一个zookeeper，zookeeper会记录着你这些kafkaserver的信息同时你这上图10的server1到server3我也同时记录着，这是因为我这个zookeeper既存你的broker的信息又存你的Consumer信息。

这个时候这个server1到server3组合的信息开始进行一个服务了对吧？那如果说假设有有台机器挂了，那这个时候相当于这个topic的信息重新分配，这个重新分配是由zookeeper来控制的，所以要表达这么一个思想，就是说你的zookeeper在你整个kafka集群里面，它是体现了一个负载均衡的思想，当然有必要的情况下它会去对你的消息做一个rebalance，就是出现个别节点不稳定会体现rebalance机制。

这里面还有一个思想，就是说我这个producer和broker之间是一个push模式，然后Consumer和producer之间是一个pull模式，push模式就是推，pull模式就是拉，那不管你是push模式还是pull模式都是由你的producer和Consumer进行主动的，就像producer有一个消息，我主动的把消息推给你的broker，然后Consumer我要消费了，那我主动的去在你的broker去取数据

那么我们看一下从Topic的角度来看（如下图12所示）

图12：  


那什么是Topic呢？就是每条发布的kafka集群的消息都是有一定的类别，那这个类别就是Topic，那物理上不同的Topic消息要分开存储，逻辑上一个Topic消息虽然保存了一个或者多个broker上，但是用户只需要指定消息的Topic就可以了，它是不需要关心用户来去消费的时候它是不需要关心你这台Topic到底是存在哪个机器上的，这个是不需要的，你只需要把这个Topic指定就可以了，相当于是你整个的一个内部实现对用户来说是一个完全透明的，所以操作起来是比较简单。

然后Topic它既然能够表达了一个类消息，那在一个Topic内部消息是一个顺序存储的，Topic它是一个逻辑的概念，那它因为你这个消息是从一个先来后到，有一个时间顺序的概念，我先来的消息就排到前面，后来的消息排到后面，那整体来说我是要保证一个整体有序的是不是？但是由于一个实现的机制kafka对这种全局绝对有序的保证有些困难，但是我可以在这上面做一个折中，就是我一个Topic是可以在通过物理方面是要可以通过多个partition来实现，但是每一个partition的内部是有序的，那它怎么保证你的有序呢？它这里面有一个offser，这个offser就是说一个偏移量，比如我在看一篇文章，我offser等于0相当于是我从这个文章的开头开始读，那offser等于100相当于是我从这篇文章的第100行开始读，它是有一个偏移量的，这有点像那个指针加上一个移动范围来进行指向一个要读取的那个地址，大概是这么一个思路，所以你从这个图上就可以看到一个Topic可以分到多个partition，然后这个partition的内部的信息左边是Old右边是New，也就是说这个消息是从右往左插入的，然后保证了一个顺序性，然后它写的时候它是怎么写的呢？是根据你这个消息的相应的key然后去落到了相应的partition里面去，所以你可以把Topic可以想象成一个队列，然后每条消费一个记录都必须要指定它的Topic，然后就是说当你发送一个消息的时候或者消费一个消息必须明确指定你是要对哪个Topic做操作。

那刚才其实我们也从另外一个角度也说了，如果是用一个partition去做也可以，但是这使得你的kafka的吞吐率不是特别好，那你要用多个partition来去并发做这样的事情的话，它可以使你的kafka吞吐率可以现性提高，这是为什么把一个Topic分成多个partition的原因。

我们这里面举个一个例子（如下图13所示）

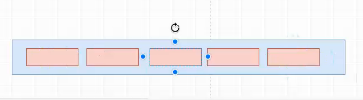
图13：



比如说创建了Topic1和Topic2这两个Topic，那分别有13个partition和另外一个是19个partition，那整个的数据集上就有相应32个文件夹，你可以从这个命名上就可以看到哪些文件夹里面的数据是属于哪个Topic的并且也可以知道哪些文件夹，它具体是代表了哪个partition。

那消息对于Topic来说，消息无论是有没有被消费，那这个消息会一直被持久化，就是说我当有一个消息来了（如下图14所示）

图14：

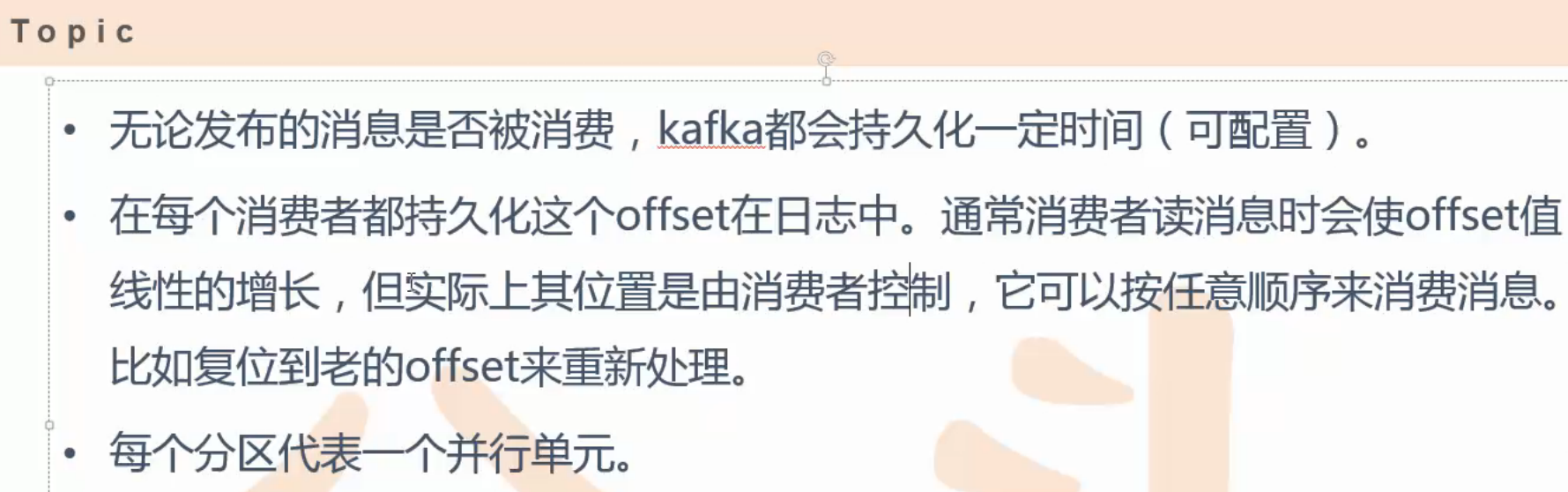


这个队列里面有这么多个消息，然后这左边是入，右边是出，这个消息一旦存到了你的kafka的内部来说，这个消息就会被存储，当你的消费者已经把最后一个消息已经消费掉了，那它这个消息不会马上被删除，它会持久化一段时间，这个消息不会是因为你把它消费掉了我这个队列就再也找不着它了，其实不是这样的哈，假设说我下游的server把这个数据消费掉了，但是你下游server突然挂掉了，那我这个机器我想再把这个数据做一个追回，那如果说你这个队列里连这个消息都没有的话，那我这个消息就没有办法再去恢复了对吧？所以kafka对这一个消息存储的机制它做的还比较简单做的很灵活，kafka是把更多的主导权交给了消费者，因为消费者有很多对不对？就是说我这个kafka的下游会有很多很多的消费者，那你每一个消费者的消费进度不一样，有的人比如像看书，有的人看书快，有的人看书慢，你每个人这个看书的这个位置是不一样的，那这个时候你每个人看书这个位置就一个offser偏移量是吧？要把每个人的这个偏移量我都把这个数据由kafka来记录的话，那势必对kafka来说是一个需要额外考虑的因素会比较多并且也不利于整个kafka的，它的设计的更复杂也不利于整个kafka的一个性能各方面的一些考虑，就优化起来比较麻烦，所以kakfa就把这个更多的主导权交给了消费者，就是说由消费者就是这些客户端来保存各自的offser偏移量。

就是说kafka我不会帮你保留很多的一些你们消费者的信息，这个具体你们读到了哪个位置由你们自己来保留，你们想从重新读可以，你们想从中间读也可以，你们想从这个文章的结尾实时更新的这个位置去读也可以，但是我这里面会帮你保留所有的数据来供你去灵活的去选择，你想读哪个数据都可以，但是我这个数据只能保留一段时间，这个时间的话是可以通过一个配置去配的，默认的话是7天，那一旦过了这个时间的话，这个消息就会被清除掉了

（如下图15所示）

图15：



然后所以对于刚才我们做一个总结就是说每一个消费者都要去各自保留你这个offser，那具体你读到了哪个位置由你自己来控制，那这个事情kakfa来说是不管的，那如果你这个server突然挂掉了，那我想把这个数据做一个还原做一个恢复，那你就完全可以把这个时间节点从头往后再重新全部按顺序加载一遍就可以了，就是说因为你每一个消费者是保留一部分数据，然后你这个offser是由你这个消费者自己来去维护的，这个zookeeper主要目的就是来去做一个负载均衡。

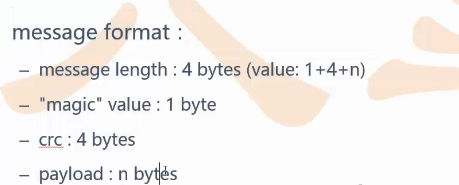
这个message就是kafka的一个基本单位（如下图16所示）

图16：



一旦这个消息产生了，那么这个消息就会不断的去追加到你的文件的末尾，而且消息一旦追加到你的文件末尾，这个消息就不能在改变了，然后我们最一开始就是在讲kafka之前我们讲了一些关于kafka的一些特点对吧？那这些特点其实有一个分不开的因素就是说和它自身的消息格式是分不开的，这个消息格式是怎么样来去定义的呢？那么大概是这个样子的一个消息格式（如下图17所示）

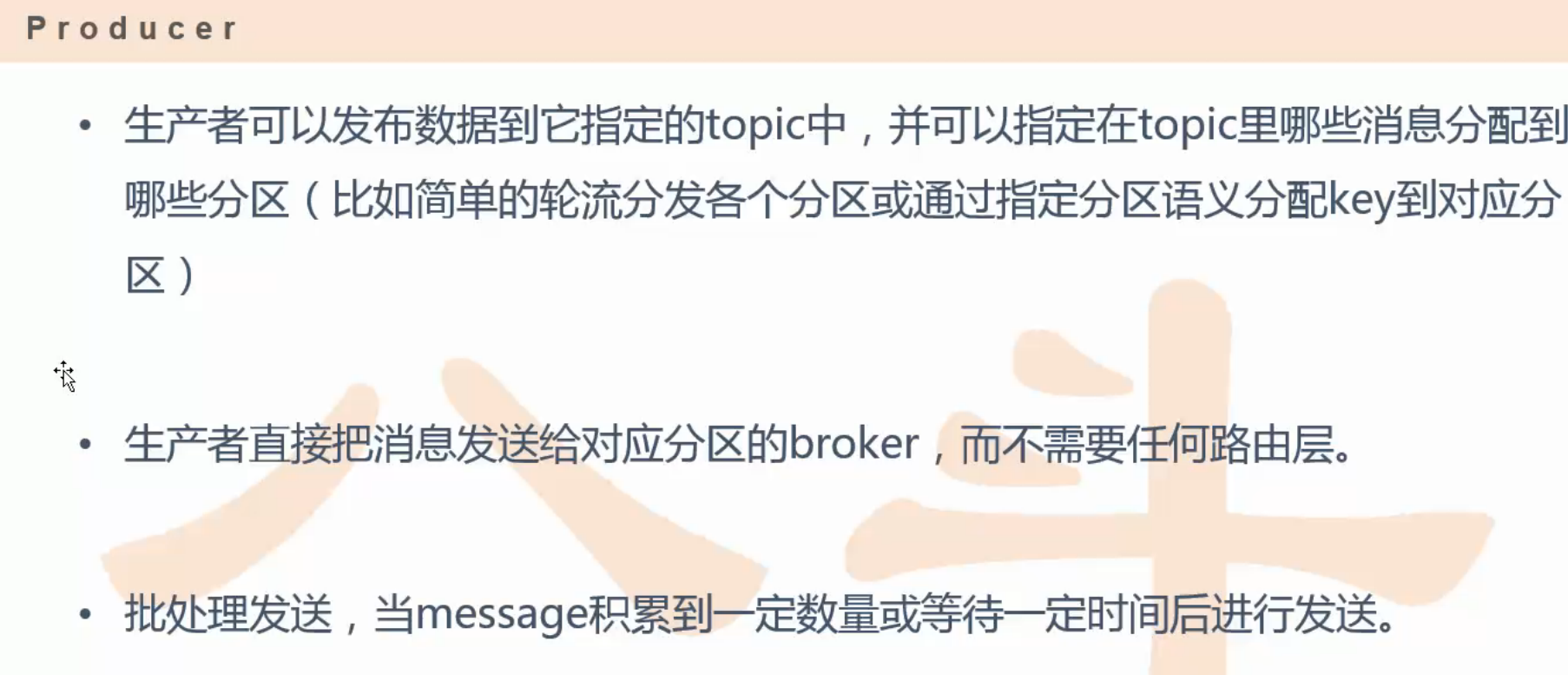
图17：



第一个是这个消息的长度(message length)是代表着你这个消息的总长是多少？这个1个是代表是它有一个magic，有一个叫换码把是吧？还有一个是一个校验码(crc)，剩下的就是你的真实数据(bytes)，加起来是整个的消息的长度，当然你在kafka的内部，为了让这个消息能够更好的和你的文件系统进行一个整合，那不可能是要把你的数据按照明文的方式去传输，肯定要把你的数据转换成二进制的形式，这样的话会有利于你的服务器端的开销并且提高IO速度。

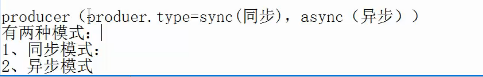
然后对于Producer来说，那它是一个生产者也是一个客户端，那这个客户端我们也是用任何语言来编写的（如下图18所示）

图18：



当然我们在实践的过程中我们可以看一下这个kafka其实在安装包里面也帮我们提供好了一些现成的Demo，我们可以直接去用，那这些生产者当然除了它有自己实现的Demo那也可以是个flume，对于你kafka来说flume是它的一个Producer对不对？所以flume和kafka在这里面也可以做一个类似于一个无缝对接，另外一个就是说Producer它分成两种模式，那第一种模式就是同步模式，第二种就是异步模式，那这个到底是同步模式还是异步模式是需要有一个叫produer.type这么一个参数来控制的，这里面值有两种，一种是sync，这个是代表同步，这个是可以配的，还有一种是async，这是异步的（如下图19所示）

图19：

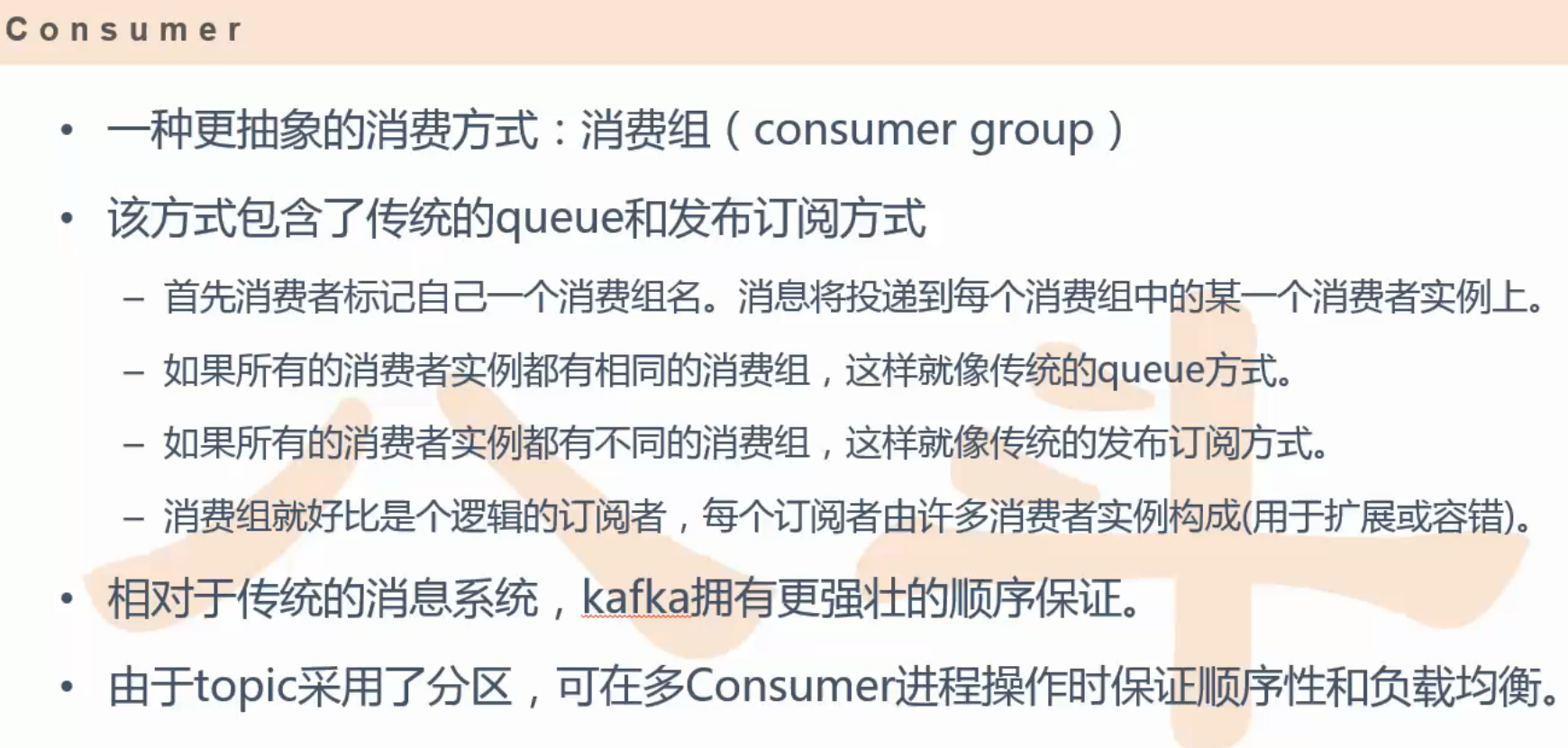


那同步就是一个实时发送，你数据一来我这数据就会马上会发出去

异步就是说你数据传输过来了，那我这个数据是要在kafka里面需要做一定的存储，不能马上被应用，你必须要在kafka内部先暂存一会，那当达到一定的条件，你这个数据才能被应用，这个条件有哪些呢？有一个时间上的，还有一个数据量上的，那这两个条件也是可以通过相应的配置去配置的，而且你的生产者可以直接把你的消息直接发送给相应partition的那个broker上，就是你这个producer不需要你zookeeper来去做一个负载均衡呢？是因为我producer我可以明确指定我想把我的消息到底发送个哪个机器上。

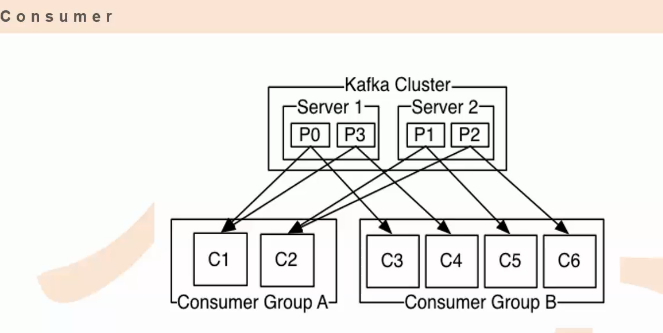
然后生产者可以指定你的topic，我可以明确出我想发送的这个消息要发送到哪个分区上，这是一个同步模式和异步模式，那对于异步模式来说当这个message积累一定的数量或者等待一定时间后才能发送。

然后Consumer它就是一个消费者，那刚才这个怎么消费的话，刚才我们也具体说了一种是吧？(如下图20所示)

图20：  


就是有一种非常常用的方式就是这种Consumer group的方式，这种方式是属于一种更高级的一种方式，那在这个group内部是每一个server来说是一个单薄的方式，但是你比如说我这个group可以和你这个Consumer是属于同一级别的，那这个消息是可以比如说我刚才发送了一个1，2，3，4，5对吧？那相当于我这个Consumer 可以把这个1，2，3，4，5这五个消息同时被我拿到，那对于group来说这1，2，3，4，5 是可以被我group全局拿到，但是真正分配到内部的这个Consumer server来说，每个Consumer server只拿到了其中一个部分有这么一个差别。

然后就是在你这个结构搭建的时候，其实你可以设置多个消费组，那你可以是一个组A或者是组B是不是？（如下图21所示）

图21：  


然后组A和组B你可以去从这个kafka集群上，这个kafka集群上有两个server，这每一个server代表一个broker是不是？那其实他们的信息都是从不同的broker上面的partition去同步过来的，那因为你看这上图21的图我这个左边的Consumer group里面有两个Consumer，然后我这个右边的有四个Consumer，但是我partition全局是有四个的，那所以这个patition可以对于一个Consumer来说因为我这个Consumer GroupB里面刚刚好四个Consumer 对应着你四个partition，那我这四个Consumer 就是一一对应的关系就是对接过来了，但是由于我Consumer Group里面这个Consumer 的数目肯定是要小于你这个partition的总数的，所以肯定会存在一种我一个Consumer 会对接着多个partition的情况，但是不管怎么样你对于Group的角度来说它们得到的这个数据是一个完整的形式。