kafka 分布式消息系统

分享人：张子阳

主讲人 张子阳 简介

10年以上从业经验，一直从事软件和互联网行业

曾主管大型上市公司电商事业部，负责产品研发团队管理。

.Net领域知名技术专家，出版有《.Net之美》和《C#揭秘》两书。

现任职于爱玩网络数据中心，负责系统重构和大数据处理和分析

目录

CONTENTS

01

Kafka介绍

1. 消息系统 2.两种模式 3.流处理 4.存储 5.挑战

1.1 Kafka – 消息系统

Kafka最早是由LinkedIn使用Java和Scala语言开发的，并在2011年开源，2012年成为Apache软件基金会的顶级项目。2014年，Kafka的几个创建人，成立了一家新的公司，叫做Confluent，专门从事Kafka相关的工作。

Kafka并不是一个严格意义上的消息队列，因为它并没有实现消息队列的标准协议（例如AMQP，Advanced Message Queuing Protocol，提供统一消息服务的应用层标准高级消息队列协议，是应用层协议的一个开放标准，为面向消息的中间件设计），所以尽管在使用方式上像极了队列，但并不算是严格意义上的消息队列。

实现了AMQP的队列组件：RabbitMQ、ActiveMQ；近似消息队列的组件：kafka、ZeroMQ

Kafka项目的目标是提供一个 统一的、高吞吐、低延迟的，用来处理实时数据的系统平台。按照官方的定义，Kafka有下面三个主要作用：

1、发布&订阅：和其他消息系统一样，发布订阅流式数据。

2、处理：编写流处理应用程序，对实时事件进行响应。

3、存储：在一个分布式、容错的集群中安全地存储流式数据。

1.1 Kafka – 消息系统

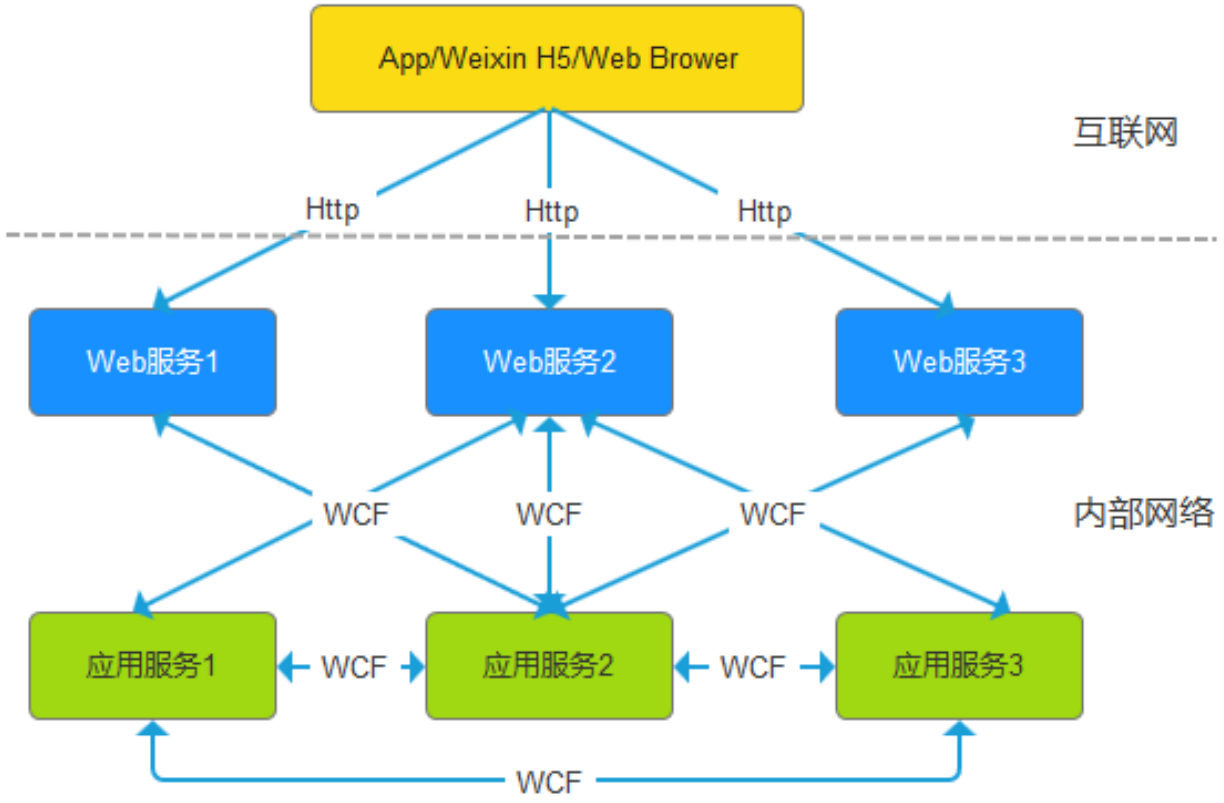
上面的三个作用，第一条就讲到，kafka是一个消息系统。那么什么是消息系统？它解决了什么样的问题？

我们以时下流行的微服务为例，假设Web端有Web1、Web2、Web3三个面向终端（微信公众号、手机App、浏览器）的Web服务（Http协议），内部有App1、App2、App3三个应用服务（远程过程调用，例如WCF、gRPC等），如果没有消息系统，采用直连的方式，它们之间的通信方式可能是这样的：

1.1 Kafka – 消息系统

kafka

直连方式进行通信



采用这种方式，主要有下面几个问题：

1. 服务之间紧耦合，设想我们修改一下应用服务2的外部接口，那么所有调用了它的组件均需要修改，在上图这种极端情况下（所有组件都调用它，这种情况在实际中并不多见），所有的其他Web服务和应用服务都需要做相应修改。

2. 服务之间的这种紧耦合，有时候还会造成接口无法修改的问题：如果有不受控的第三方调用了接口，那么修改接口将造成第三方应用不可用。设想微信公众号的某个接口改动一下，将会造成成千上万的应用故障。

3. 为了解决上面问题，接口往往以版本的方式发行，访问形式如 web/v1/interface、web/v1.1/interface、app/v2.0/interface。接口在小版本号之间兼容，大版本号之间不兼容。

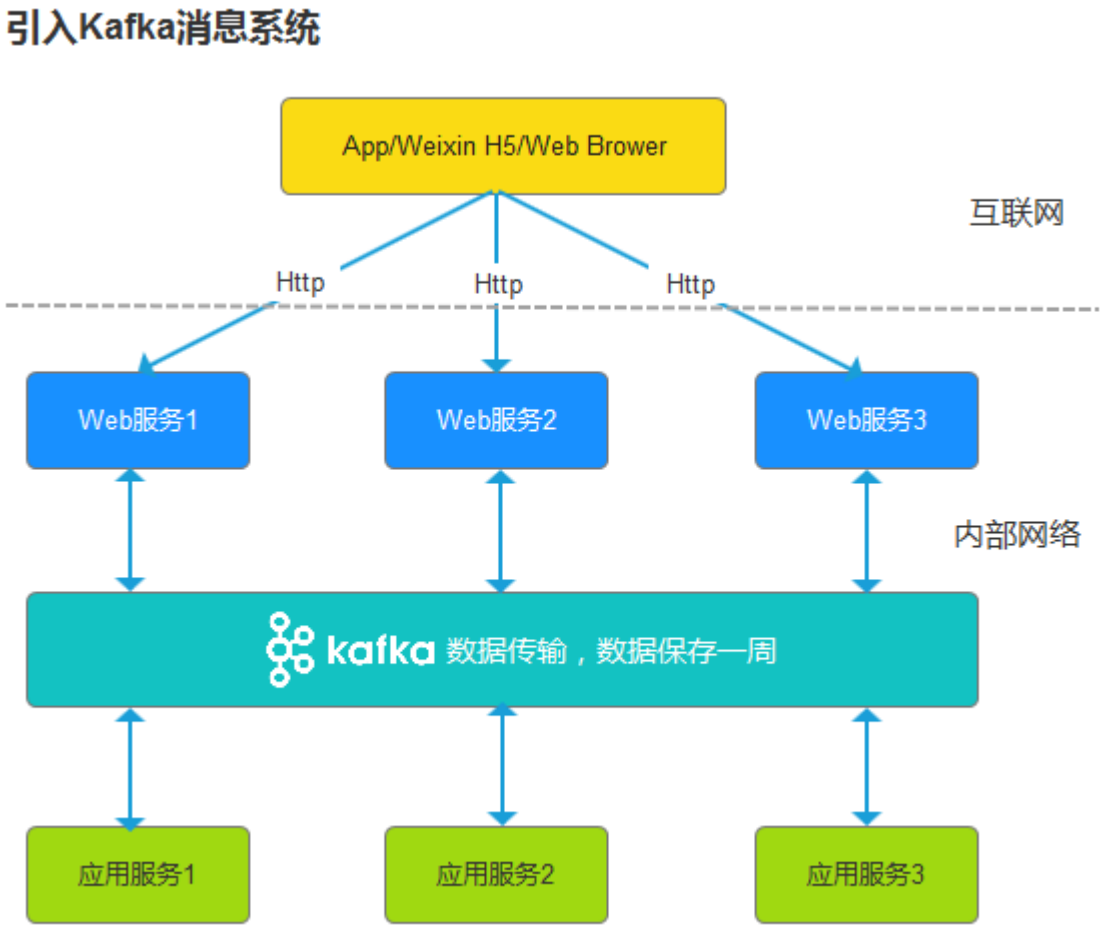
4. 上面这种接口规划方式虽然一定程度解决了紧耦合的问题，但又带来了新问题：更新需要改动多个版本序列，版本过多的时候将难于维护。

5. 增减客户端繁琐，假如现在新加入一个应用服务4，提供某个Web服务所必须的功能，那么就要把Web服务1、Web服务2、Web服务3全都改一遍；同样，如果应用服务2不再需要，那么同样要在Web服务端去掉调用它的代码。

6. 性能受限，不易扩展，例如要做负载均衡，需要借助第三方的工具，例如Zookeeper或者Consul等。或者需要改写代码或者加入特定的配置。

1.1 Kafka – 消息系统

而引入消息系统时，结构将变成下面这样：



1.1 Kafka – 消息系统

引入消息系统后，上面的问题将会得到有效解决：

1. 所有的组件，Web服务和应用服务，都不再关心彼此的接口定义，而仅关心数据结构（例如Json结构）。
2. 仅需要知道与Kafka的通信协议就可以了，而Kafka的结构已经高度标准化，相对稳定和成熟。
3. 提升了性能，Kafka针对大数据传输设计，吞吐率足以应付绝大多数企业需求。
4. 易于扩展，Kafka本身就可以通过集群的方式进行扩展。除此以外，其独特的模式为负载均衡等常见需求提供了支持。

1.2 Kafka – 消息系统的两种模式

生产者/消费者 模式：

Producer（生产者）：在数据管道一端 生产消息 的应用程序。
Consumer（消费者）：在数据管道一端 消费消息 的应用程序。

- 生产者将消息发送至队列，如果此时没有任何消费者连接队列、消费消息，那么消息将会保存在队列中，直到队列满或者有消费者上线。

- 生产者将消息发送至队列，如果此时有多个消费者连接队列，那么对于同一条消息而言，仅会发送至其中的某一个消费者。因此，当有多个消费者时，实际上就是一个天然的负载均衡。

1.2 Kafka – 消息系统的两种模式

发布者/订阅者 模式：

Publisher（发布者）：在数据管道一端 生成事件 的应用程序。
Subscriber（订阅者）：在数据管道一端 响应事件 的应用程序。

- 当使用 发布者/订阅者 模式时，发往队列的数据不叫消息，叫事件。对于数据的处理也不叫消费消息，叫事件订阅。

- 发布者发布事件，如果此时队列上没有连接任何订阅者，则此事件丢失，即没有任何应用程序对该事件作出响应。将来如果有订阅者上线，也不会重新收到该事件。

- 发布者发布事件，如果此时队列上连接了多个订阅者，则此事件会广播至所有的订阅者，每个订阅者都会收到完全相同的事件。所以不存在负载均衡

1.3 Kafka – 流处理应用程序

区分批处理程序和流处理程序。

批处理和流处理的最大区别就是数据是否有明显的边界。如果有边界，就叫做批处理，例如：客户端每小时采集一次数据，发送到服务端进行统计，然后将统计结果保存到统计数据库。

如果没有边界，就叫做流式数据（流处理）。典型的流处理，例如大型网站的日志和订单，因为日志、订单是源源不断的产生，就像一个数据流一样。如果每条日志和订单，在产生后的几百毫秒或者几秒内被处理，则为流式程序。如果每小时采集一次，再统一发送，则将本来的流式数据，转换为了“批数据”。

流式处理有时候是必须的：比如天猫双11的订单和销售额，马云需要实时显示在大屏幕上，如果数据中心说：我们是T+1的，双11的数据，要12号才能得到，我想马云baba是不会同意的。

1.3 Kafka – 流处理应用程序

处理流数据和处理批数据的方法不同，Kafka提供了专门的组件Kafka Streaming来处理流数据；对于其他的Hadoop生态系统项目，各自提供了不同的组件，例如，Spark也包括了Spark Streming来处理流数据。而流数据处理的鼻祖Storm则是专门开发用来处理流式数据的。

除了使用数据边界来区分流处理和批处理以外，还有一个方法就是处理时间。批处理的处理周期通常是小时或者天，流处理的处理周期是秒。对应的，批处理也叫做**离线数据处理**，而流处理叫做 **实时数据处理**。还有一种以分钟为单位的，叫做 **近线数据处理**，但是这种方式讨论的比较少，其是离线处理的套路，只是缩短了处理周期而已。

1.4 Kafka–存储：分布式、容错的集群中安全地存储流式数据

默认情况下，Kafka中的数据可以保存一周。同时，Kafka天然支持集群，可以方便地增减机器，同时可以指定数据的副本数，保证在集群内个别服务器宕机的情况下，整个集群依然可以稳定提供服务。

在我们的数据中心的项目应用中，主要是用作数据传输。为了看清楚它解决了一个什么问题，先简单介绍一下这个项目的场景：

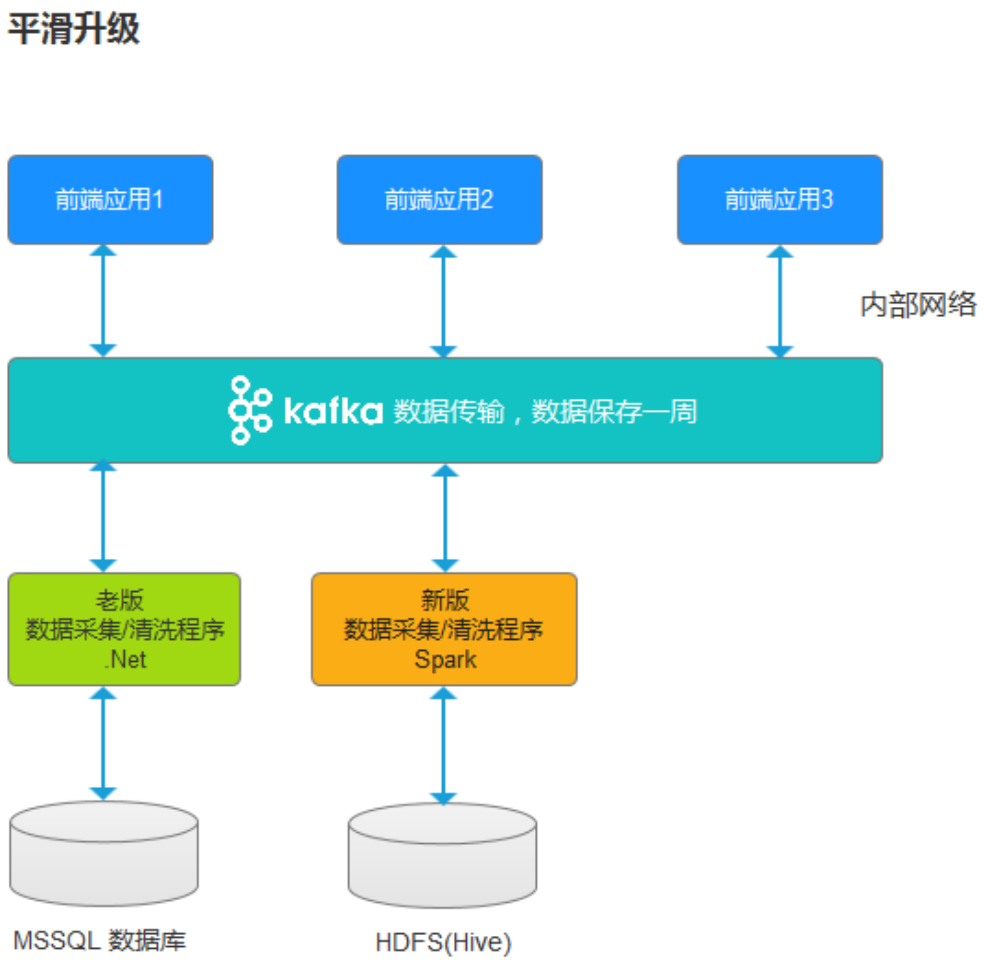
这个项目的前端是各种应用，应用的数量未来可能有几十上百个，但目前只有10个。这些前端的应用要将数据发送到后端的数据中心（一个我们称为数据采集器的程序，简称采集器），很明显，采集器与应用是一对多的关系。这样就会出现这样一种情况：大部分的时间采集器器空闲，但是当多个应用同时发数据时，采集器又处理不过来。此时就需要一个缓冲机制，使得采集器不会太闲也不会太忙。这时就可以采用Kafka作为这个数据缓冲池。

1.4 Kafka–存储：分布式、容错的集群中安全地存储流式数据

在这个应用范例中，选择Kafka而没有选择传统的成熟消息队列组件，例如RabbitMQ，是因为Kafka天生是为了应对大批量数据的，所以性能更好一些。

除了起到数据缓冲的作用以外，Kafka在数据中心的应用中，还起到了“平滑升级”的作用，如下图：

1.4 Kafka–存储：分布式、容错的集群中安全地存储流式数据



需求是这样的：之前的前端应用、数据采集、数据清洗程序都是采用.Net开发的，并存入到MS SQL Server数据库中。为了应对日益膨胀的数据量，决定采用大数据技术，将数据存储在HDFS上，并使用Spark进行数据统计。

因为引入了Kafka，所以不管是老版的前端应用、数据采集、还是清洗程序，都不需要做任何的改动。就可以接入新版的采集/清洗程序，因为只要从Kafka中取数据就好了。

当新版本的程序测试通过后，只需要简单地停掉老版程序，就可以平滑地切换到新系统。

1.5 Kafka–引入消息系统带来的挑战

所有事物都不可能只有优点没有缺点，引入Kafka带来的挑战主要有下面几个：

- Kafka依赖，虽然系统中的应用彼此之间不依赖了，但是都重度依赖Kafka，此时Kafka的稳定性就非常重要(类似MS SQL Server一样的基础设施)。

- 微服务的实践中，出现了另一种服务独立化的呼声，即各个服务不需要其他的组件就可以独立对外提供服务，也就是去中心化。两种方式的选用时机就显得非常重要（不是非此即彼，而是按需选用）。

- 消息队列天然都是异步的，虽然提升了性能，但是增大了代码复杂性。本来使用RPC同步调用返回结果的简单操作，采用异步后加大了代码编写的复杂度和调试的复杂度。

02

Broker/Topic/Partition

1.Broker 2. Topic/Partition/Offset 3. 集群分布
4. 副本 5.Leader和ISR

2.1 Kafka – Broker

Broker（服务进程）：Broker直译为代理。我觉得这个称谓不好理解，其实通俗讲就是运行kafka的服务器，再具体一点就是运行Kafka的服务进程。

- 当你连接到集群中的任意一个Broker时，就可以访问整个集群了。

- 集群内的Broker根据Id进行区分，Id为纯数字。

2.2 Kafka – Topic、Partition和Offset

Topic（主题）：可以理解为一个数据管道，在这个管道的一端生产消息/发布事件，另一端消费消息/响应事件。管道本身进行消息/事件的存储、路由、发送。主题由它的名称(Name)所标识。

主题中的数据，不论是不是被消费，都会保存指定的一段时间，默认是一周。

Topic可以被分割成多个Partitions（分区）。

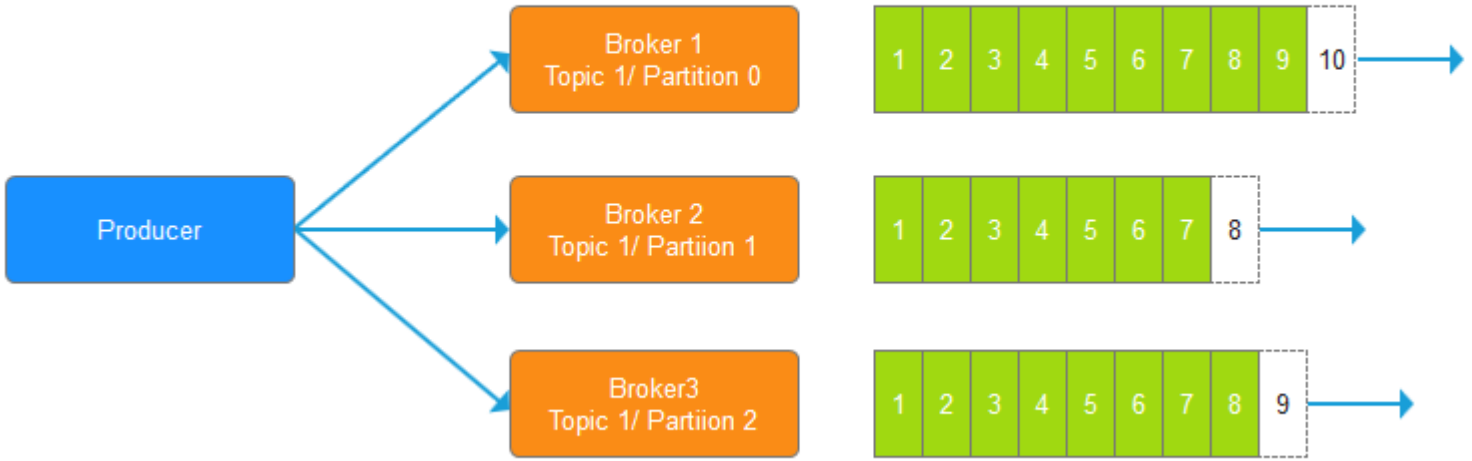
2.2 Kafka – Topic、Partition和Offset

- 分区内的数据是有序的。当一个主题只有一个分区时，那么这个主题的消息也是有序的；但如果一个主题有多个分区，那么消息是无序的。
- 分区越多，并行处理数就越多。通常的建议是主机数x2，例如如果集群中有3台服务器，则对每个主题可以创建6个分区。
- 当消息被写入分区后，就不可变了，无法再进行修改。除非重建主题，修改数据后重新发送。
- 当没有key时，数据会被发往主题的任意一个分区；当有key时，相同key的数据会被发往同一个分区。

发往Partition的每条消息将获得一个递增id，称为offset（偏移量）。整体上看，结构如下图所示：

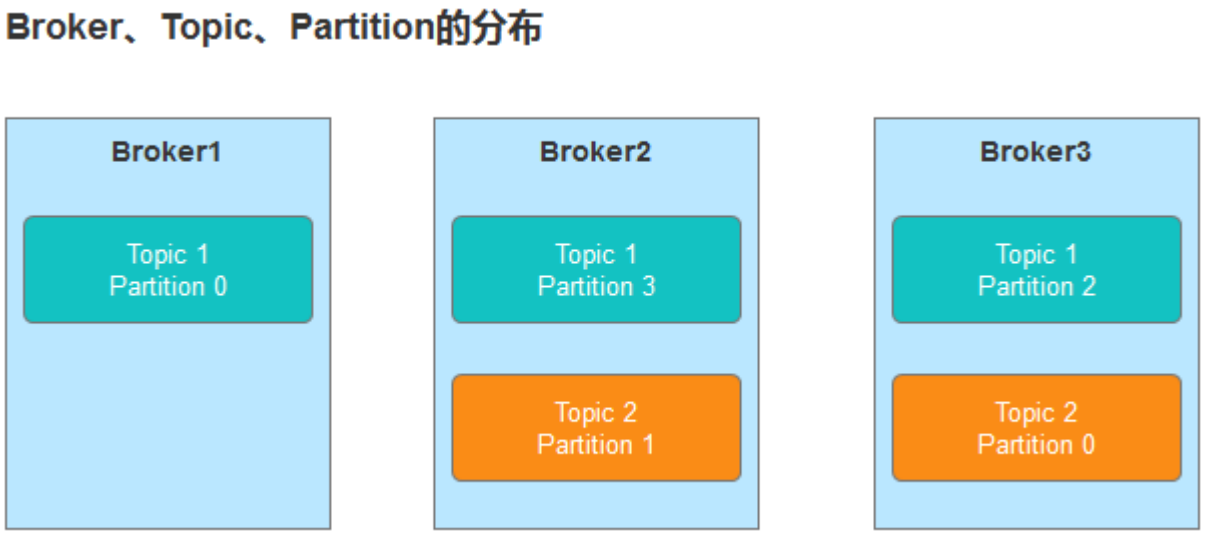
2.2 Kafka – Topic、Partition和Offset

Topic、Partition、Offset



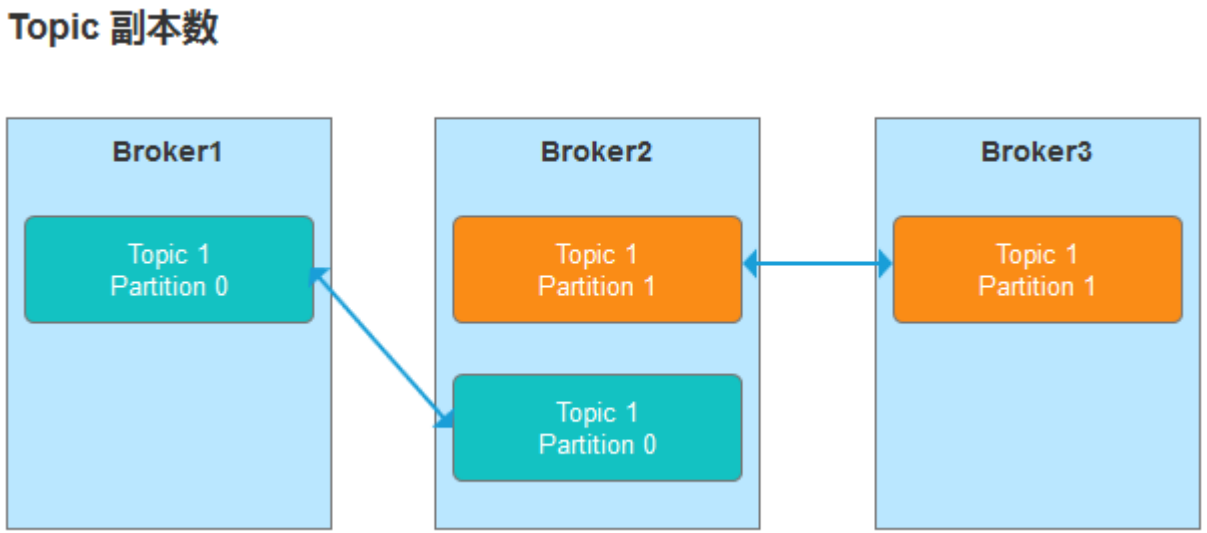
2.3 Kafka – Broker、Topic、Partition的分布

对不同的Topic，可以设置不同的Partition数目，当集群中有多个节点时，将会随机分布在不同的节点上。
如下图：Topic1拥有3个Partition，Topic2则只有2个Partition。



2.4 Kafka – Topic 副本数（replication factor）

通常会将Topic的副本数设置为2或者3，此时当某个节点故障下线时，该Topic依然可用，集群内的其他节点将会提供服务。



显然，并不是副本数越多越好。副本数越多，同步数据需要花的时间越久，磁盘的使用率会越低。

kafka

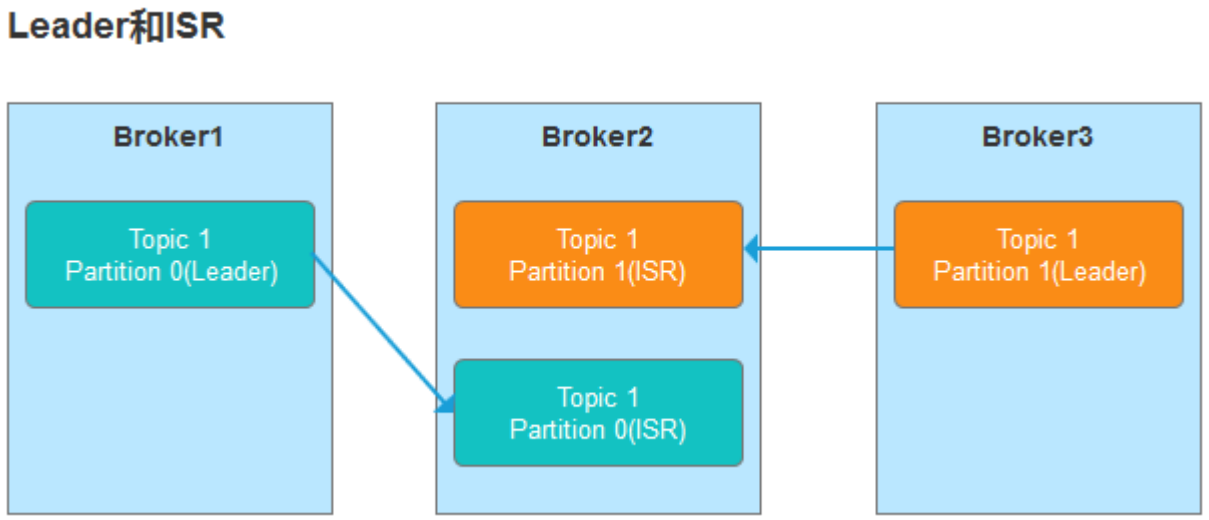
当副本数设为N时，允许N-1个节点故障(离线)。

不管是Kafka集群还是Hadoop集群，并不是说节点越多容错就越高，容错是一样的；只不过是恰巧相关节点同时发生故障的概率小一些。

比如说，Hadoop集群中有100个节点，当你的副本数设置为2时，恰巧保存这两个副本的节点故障了，相关的数据一样无法访问。而100个节点相对于5个节点或者3个节点，恰好保存相同副本的节点同时故障的概率低一些。

2.5 Kafka – Partition Leader和ISR

对于多个Partition的Topic来说，只有一个Leader Partition，一个或多个ISR（in-sync replica，同步副本）。leader进行读写，而ISR仅作为备份。



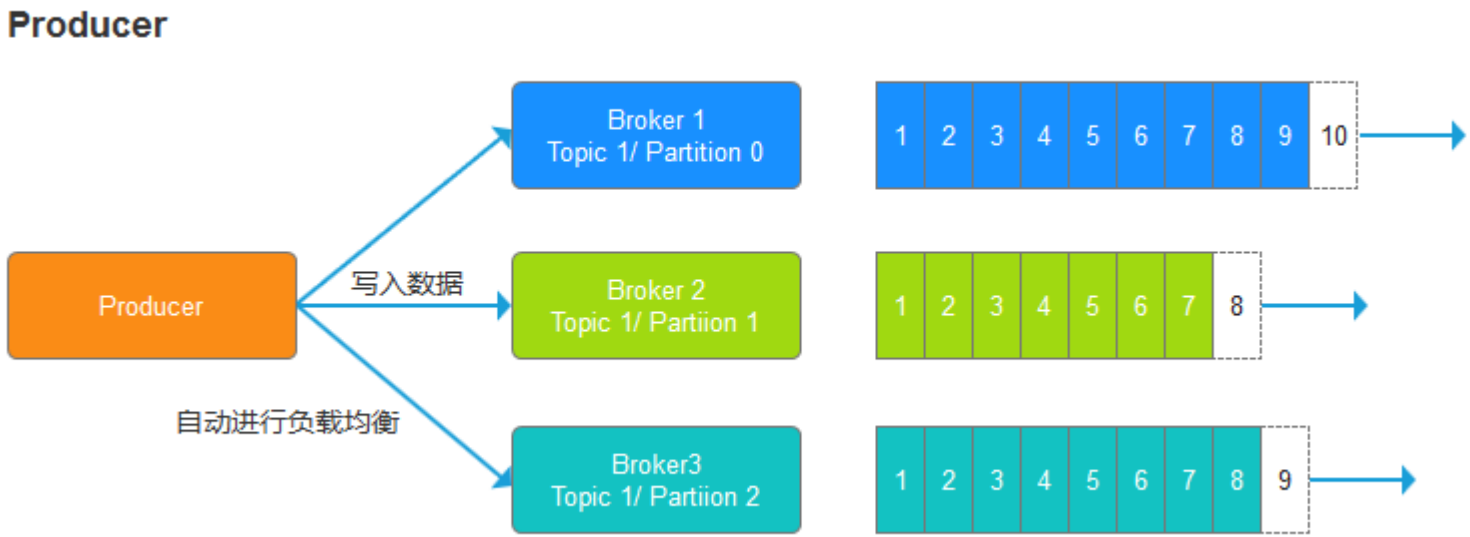
03

Producer 生产者

1.写入数据 2.Acks 3. Keys

3.1 Kafka – Producer向Topic中写入数据

Producer只需要指定Topic的名称（Name），然后连接到集群中的任意一个节点，Kafka会自动进行负载均衡，并将对写入操作进行路由，从而写入到正确的Partition当中（多个Partition将位于集群中的不同节点）。



需要注意的是：上图没有加入ISR Partition，这么做是为了制图更简单一些。

3.2 Kafka – Producer Acks

Producer可以选择用下面三种方式来获得数据写入的通知：

- Acks=0，速度最快，Producer不去等待写入通知，有可能存在数据丢失。
- Acks=1，速度较快，Producer等待Leader通知，但不会等待ISR通知，有可能ISR存在数据丢失。
- Acks=all，速度最慢，Producer等待Leader和ISR的通知，不存在数据丢失。

3.3 Kafka – Topic Keys

Producer在发送数据时，可以指定一个Key，这个Key通常基于发送的数据。

举个例子，如果要发送一笔电商的订单数据（OrderNo 单号、Retailer 卖家、Customer 买家）。

如果：

- 数据的接收端（Consumer），不关心订单的发送顺序，那么key可以为空，也可以为OrderNo。
- 数据的接收端，要求卖家的订单要按顺序发送，那么Key设为Retaier。
- 数据的接收端，要求买家的订单要按顺序发送，那么Key设为Customer。

3.3 Kafka – Topic Keys

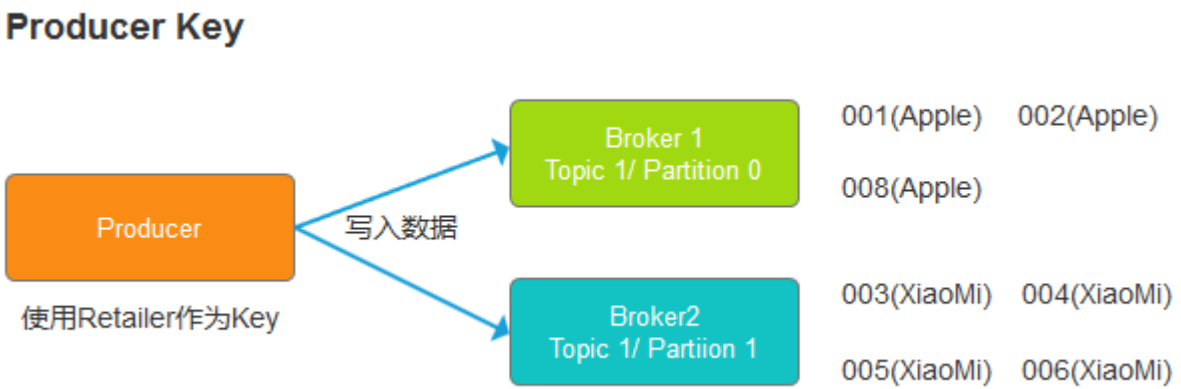
这里比较容易晕的是：当Key为Retailer时，并不是说每次发送Key都填一个字符串，“Retailer”，而是Retailer的具体值。以下面的表格为例：

OrderNo	Customer	OrderAmount	OrderDate	Retailer
001	Jimmy	5200	2017-10-01 00:00:00	Apple
002	Jack	3180	2017-11-01 00:00:00	Apple
003	Jimmy	2010	2017-12-01 00:00:00	XiaoMi
004	Alice	980	2018-10-01 00:00:00	XiaoMi
005	Eva	1080	2018-10-20 00:00:00	XiaoMi
006	Alice	680	2018-11-01 00:00:00	XiaoMi
007	Alice	920	2018-12-01 00:00:00	Apple

3.3 Kafka – Topic Keys

那么对于订单001~007，将Retailer作为Key，则Key的值分别为：Apple(001)、Apple(002)、XiaoMi(003)、XiaoMi(004)、XiaoMi(005)、XiaoMi(006)、Apple(007)。

这样，所有Apple的订单会按**次序**发往同一个Partition，而所有XiaoMi的订单会按次序发往同一个Partition。这两个Partion可能是同一个，也可能不同。如下图所示：



04

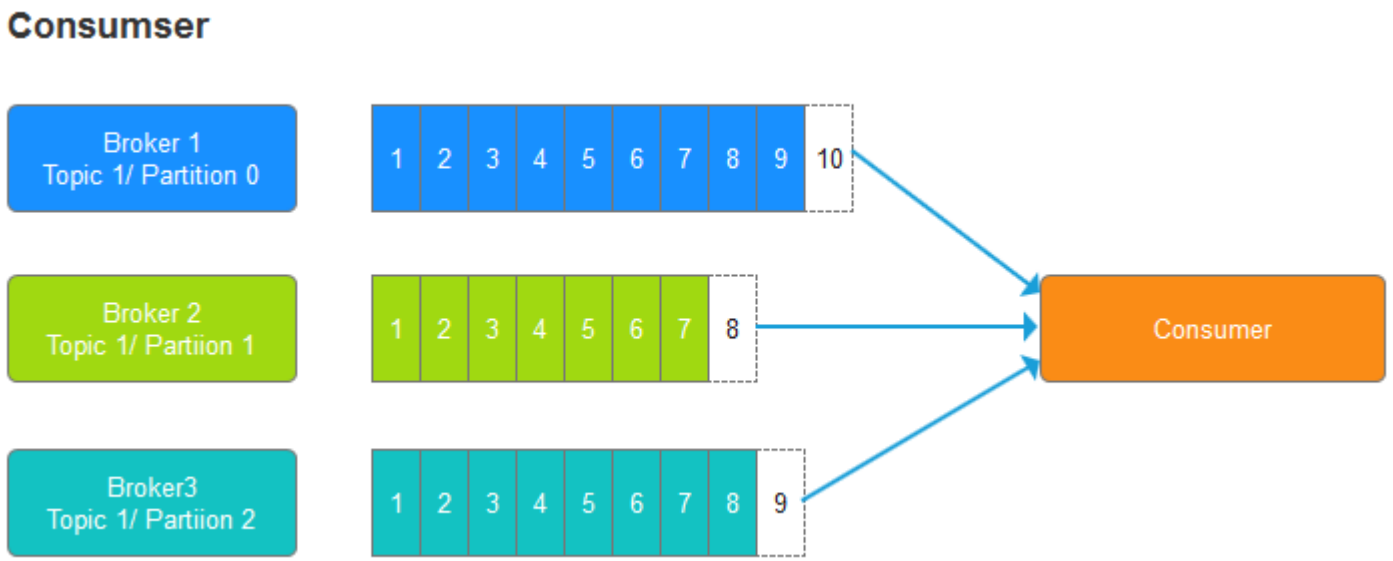
Consumer 消费者

1.读取数据 2.分组 3.offsets 4.Zookeeper

4.1 Kafka – Consumer基本概念

Consume用于从Topic中读取数据。和Producer类似，只需要连接到集群中的任意一个节点，并指定Topic的名称，Kafka会自动处理从正确的Broker和Partition中提取数据发给Consumer。

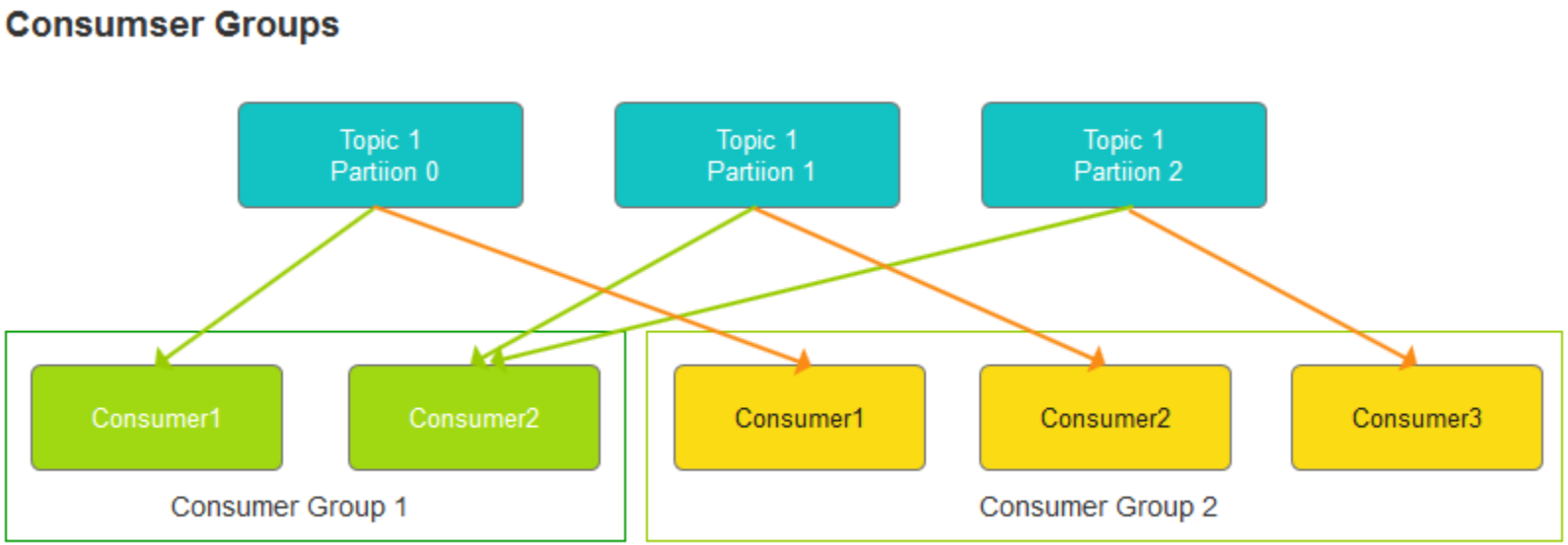
对于每个Partition而言，数据是有序的，如下图所示：



4.2 Kafka – Consumer Group

Kafka使用群组（Group）的概念巧妙地实现了 生产者/消费者、发布者/订阅者 模式的二合一。

一个Topic可以有多个Group，一个Group内可以包含多个Consumer。对于群组内的Consumer来说，它们是生产者/消费者模式，一个消息只能被Group内的一个Consumer消费；对于不同的群组来说，它们是发布者/订阅者模式，同一个消息会被发送给所有的群组。下图很好地描述了这样的关系：



4.2 Kafka – Consumer Group

一个Partition只会分配给同一个Group中的一个Consumer。如果一个Topic只有3个Partition，但是一个Group中有4个Consumer，那么就会有一个Consumer是多余的，无法收到任何数据。

4.3 Kafka – Consumer Offsets

首先要注意的是：这里的Conumser Offsets和前面Topic中的Offsets是两个完全不同的概念。这里的Offsets是Consumer相关的，前面的Offsets是Topic相关的（具体来说是Partition）。有下面几点需要注意：

- offset记录了每个Group下每个Consumer读取到的位置。
- Kafka使用了一个特殊的Topic用来保存Consumer Offsets，这个Topic的名称是__consumer_offsets。
- 当Consumer离线后重新上线，会从之前offsets记录的位置开始读取数据。

4.3 Kafka – Consumer Offsets

Offsets的提交时机

At most once(至多一次)：Consumer只要一收到消息，就提交offsets。这种效率最高，但潜在的可能是当消息处理失败，例如程序异常，那么这条消息就无法再次获得了。

At least once(至少一次)：当Consumer处理完消息再提交offsets。这种可能会重复读，因为如果处理时异常了，那么这个消息会再读一次。这个是默认值。

Exactly Once：仅部分支持。

通常的做法是选择at least once，然后在应用上做处理，保证可以重复操作，但不会影响最终结果(即所谓的幂等操作，有时候技术界容易产生一些听上去很复杂实际上很简单的名词，其实就叫“无副作用操作”不就好了嘛)。
比如说导入数据，在导入前要判断下是否已经导入过了。或者不判断先导入，然后用一个外挂程序将导重复的数据清理掉。

延伸知识：CAP理论：一个分布式系统最多只能同时满足一致性（Consistency）、可用性（Availability）和分区容错性（Partition tolerance）这三项中的两项。

4.4 Kafka – Zookeeper

Zookeeper是一个分布式服务注册、发现、治理的组件，大数据生态系统中的很多组件都有用到Zookeeper，例如HDFS等。Kafka强依赖于Zookeeper，实际上，在Kafka的安装包里就直接包含了其兼容的Zookeeper版本。

在Kafka中，Zookeeper主要有下面几个作用：

- 管理集群中的节点，维护节点列表。
- 管理所有的主题，维护主题列表。
- 执行partition的leader选举。
- 当集群变化时通知Kafka，这些变化包括新建Topic、Broker上线/下线、删除Topic

5. 总结

我们讨论了Kafka中的主要概念和机制，相信你已经对Kafka有了一个初步的认识。在以后的分享会中，我们将会进行实际操作，看Kafka是如何工作的。在我使用的过程中，它一直都是健壮、稳定的、可靠的，希望你也会有同样的感受！

THANKS

谢 谢 大 家 ， 么 么 哒 ~ ！