1 Kafka概述

* 1. 定义

集群扩展：支持热扩展，但是新增的broker不能分担之前已经创建的topic的负载，必须手动执行重新分区才能从为新增broker添加之前的已经存在的topics服务。当然如果不执行重新分组，之后新添加的topics会自动将负载分担到新broker上。

Kafka提供的重分区工具是kakfa-reassign-partitions.sh

* 1. 消息队列

1.2.1 传统消息队列的应用场景

1. 使用消息队列的好处
2. 什么是**耦合**？

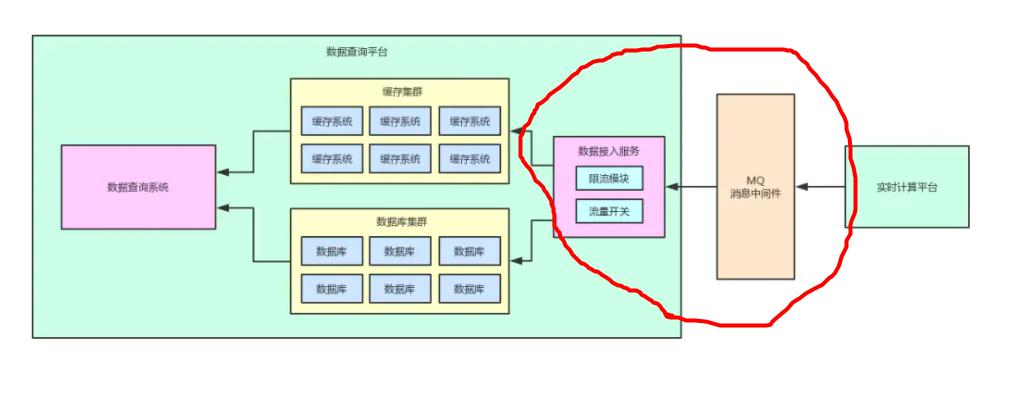
模块的耦合是指模块间联系的紧密程度,又称为模块联结。设计、评价模块结构的一个重要原则是: 尽可能降低模块间的耦合度，从而提高模块的独立性。模块间的耦合度越低，表明模块间的联结越少,模块的独立性越好,模块间相互影响就越小,模块的运行就很少受到其他模块运行的影响,模块的修改、维护工作也不会影响其他模块。

引入消息中间件**解耦**

现在我们划分清楚了系统之间的边界，接着下一步，就是引入消息中间件来进行解耦。

现在我们只要引入一个消息中间件，然后让实时计算平台将计算好的数据按照预设的格式直接写入到消息中间件即可。

同时在数据查询平台这边增加一个数据接入服务，负责将消息中间件里的数据消费出来，然后落地写入到本地的缓存集群和数据库集群。



1. 名词简介

**消息**——kafka数据单元， 消息可以有一个可选的元数据——**键key**。

**批次**—— 一组消息，为提高效率 消息分批写入kafka

**消息模式**——定义消息格式，通常Avro模式，模式和消息体分开。（序列化模板）

**集群**—— 多个broker组成集群，其中一个broker担任集群控制器“首领”

保留消息（在一定期限内）：kafka一个重要特性，要么保留一段时间，要么保留到消息达到一定大小的字节数，例如Nginx日志一般保存45天。

**Zookeeper**——分布式服务框架在kafka中的作用主要负责保存topic ，partition元数据，和对broker的监控及治理，以及partition 的leader 选举（partition 可以有多个副本，但是只有一个处于工作状态，副本只是负责同步数据，当leader partition 死掉了，会把起作一个副本的partition升级为leader）

**broker**—— 缓存代理，Kafka集群中的一台或多台服务器统称broker，从生产者接受消息 设置偏移量。不保存订阅者的状态，由订阅者自己保存,没有副本机制，一旦broker宕机，该broker的消息将都不可用

**主题(topic)**——消息通过主题分类，类似于数据库的表、文件系统的文件夹

**分区(partition)**——主题分为若干分区，一个分区就是一个提交日志。消息以追加的方式写入分区，先入先出

**Segment**——多个大小相等的segment file (段)组成了一个partition。

**流**—— 一个主题的数据，从生产者移动到消费者。

**生产者**——（发布者、写入者）创建消息，

**消费者**—— （订阅者、读者）读取消息，

**偏移量**——另一种元数据，类似于id 递增整数，给定分区中每个消息的偏移量都是唯一的

1. Kafka安装部署
   1. 安装部署
      1. 集群规划
      2. zookeeper安装

配置文件参数：

**1）tickTime**：CS通信心跳时间。客户端与服务器或者服务器与服务器之间维持心跳，也就是每个tickTime时间就会发送一次心跳。通过心跳不仅能够用来监听机器的工作状态，还可以通过心跳来控制Follower跟Leader的通信时间，默认情况下FL的会话时长是心跳间隔的两倍。

**2）initLimit：**集群中的follower服务器(F)与leader服务器(L)之间初始连接时能容忍的最多心跳数（tickTime的数量）。

**3）集群信息的配置：**

配置集群信息是存在一定的格式：service.N =YYY： A：B

N：代表服务器编号（也就是myid里面的值）

YYY：服务器地址

A：表示 Follower跟Leader的通信端口，简称服务端内部通信的端口（默认2888）

B：表示 选举端口（默认是3888）

* + 1. kafka安装

配置文件：在kafka/config/目录下面有3个配置文件：producer.properties、consumer.properties、server.properties

2.3 Kafka-manager

2.3.2 管理页面测试

页面上出现的参数：

1. Enable JMX Polling：是否开启JMX轮询，该部分直接影响部分 kafka broker 和 topic 监控指标指标的获取（生效的前提是 kafka 启动时开启了 JMX\_PORT。主要影响如下指标的查看：broker/topic 的combined metrics 综合指标。
2. Poll consumer information是否开启获取消费信息，直接影响能够在消费者页面和 topic 页面查看消费信息

3）

**JMX**（Java Management Extensions，即Java管理扩展）是一个为应用程序、设备、系统等植入管理功能的框架。

**SSL**(Secure Sockets Layer 安全套接字协议),及其继任者传输层安全（Transport Layer Security，TLS）是为网络通信提供安全及数据完整性的一种安全协议。TLS与SSL在传输层与应用层之间对网络连接进行加密

4）Enable Active OffsetCache是否开启offset缓存，决定kafka-manager是否缓存住 topic的相关偏移量。

5）Under Replicated%

该 topic 下的 partition，其中副本处于失效或者失败的比率。失败或者失效是指副本不处于 ISR 队列中。目前控制副本是否处于 ISR 中由 replica.log.max.ms 这个参数控制。

replica.log.max.ms: 如果一个follower在这个时间内没有发送fetch请求或消费leader日志到结束的offset，leader将从ISR中移除这个follower，并认为这个follower已经挂了，默认值 10000 ms

1. Topics下的3个参数：

Broker Skew%: 反映 broker 的 I/O 压力，broker 上有过多的副本时，相对于其他 broker ，该 broker 频繁的从 Leader 分区 fetch 抓取数据，磁盘操作相对于其他 broker 要多，如果该指标过高，说明 topic 的分区均衡不好，topic 的稳定性弱；

Broker Leader Skew%：数据的生产和消费进程都至于 Leader 分区打交道，如果 broker 的 Leader 分区过多，该 broker 的数据流入和流出相对于其他 broker 值的平均要大，该指标过高，说明topic的分流做的不够好；

Under Replicated%: 该指标过高时，表明topic的数据容易丢失，数据没有复制到足够的broker上。

7）Preferred Replicas%： 分区的副本中，采用副本列表中的第一个副本作为 Leader 的所占的比重。在理想的状态下，分区的 leader 最好是 “优先副本”，这样有利于保证集群中 broker 的领导权比较均衡。

8）topic 操作

Manual Partition Assignments：手动为每个分区下的副本分配 broker，一般当有 Broker Skew时或者Broker Leader Skew后可以借助该功能进行调整，一般在批量移动partition时比较方便，比如集群新增broker或者topic新增partition 后，将分区移动到指定的broker；

Generate Partition Assignments： 系统自动为每个分区下的副本分配 broker；

Reassign Partitions： 平衡集群负载，一般手动调整、系统自动分配分区和添加分区之后，都需要调用 Reassign Partition

1. 消费者

Total lag:

lag为负数：Producer的offset是通过JMX轮询获得的，Consumer的offset是从kafka内的 \_\_consumer\_offsets 的 topic 中直接读取到的，很明显轮询获取offset比直接从topic拿offset慢一点，也就可能会出现Lag计算后为负数的情况。

消息过期了（超过默认7天），已经被清理掉了，也可能会出现这种情况。

3 Kafka架构深入

3.1 Kafka工作流程及文件存储机制

3.1.1 工作流程

3.1.2 文件存储机制

3.2 Kafka生产者

3.2.1 分区策略

通过键key实现，键有两个用途 ：可以作为消息的附加信息，也可以用来决定消息该被写到主题的哪个分区。拥有相同键的悄息将被写到同一个分区。

如果键值为 null 井且使用了默认的分区器，那么 录将被随机地发送到主题内各个可用的分区上（轮询算法）

如果键不为空，并且使用了默认的分区器，那么Kafka 会对键进行散列然后根据散列值把消息映射到特定的分区上。关键之处在于，同一个键总是被映射到同一个分区上，所以在进行映射时，我们会使用主题所有的分区，而不仅仅是可用的分区。这也意味着，如果写人数据的分区是不可用的，那么就会发生错误

1）

ProducerRecord对象

1、topic字段：用于指定该消息记录的一级分属，kafka集群可以作为消息总线来使用，此时集群可以接收位于不同系统中的生产者发送来的不同类别的消息，而topic字段便可认为是逻辑层面的一种划分；

2、partition字段和key字段（可选）：用于指定该消息记录的二级分属，kafka集群为了提高消息投递和消费的并发度，引入了分区概念，对消息记录进行二级分属指定。这两个字段属于可选字段，不同的组合方式对应不同的划分策略。；

3、value字段：用于存放该消息记录的有效载荷，生产者真正要发布的消息内容会存放于value字段；

4、timestamp字段：用于指定时间戳。

2）

Round Robin轮询算法：

轮询调度算法的原理是每一次把来自用户的请求轮流分配给内部中的服务器，从1开始，直到N(内部服务器个数)，然后重新开始循环。算法的优点是其简洁性，它无需记录当前所有连接的状态，所以它是一种无状态调度。

Weighted Round-Robin权重轮询算法：

假设有一组服务器S = {S0, S1, …, Sn-1}，一个指示变量i表示上一次选择的服务器，W(Si)表示服务器Si的权值。变量i被初始化为n-1，其中n > 0。

j = i;

do {

j = (j + 1) mod n;

if (W(Sj) > 0) {

i = j;

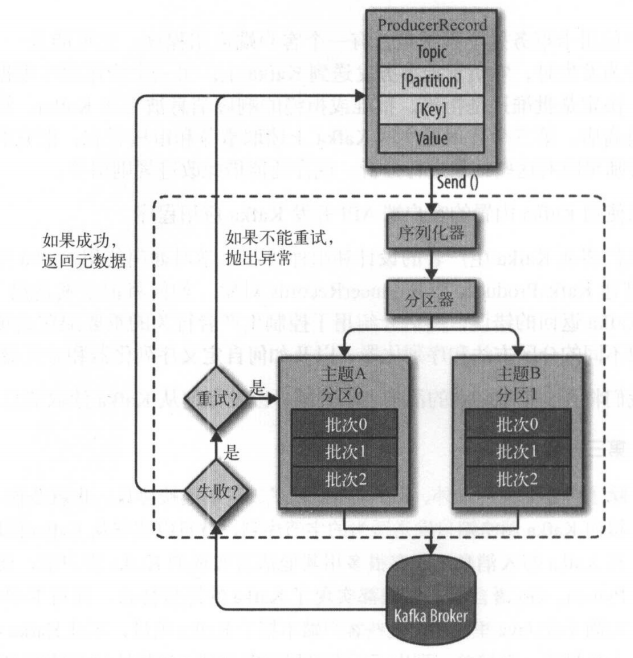
return Si;

}

} while (j != i);

return NULL;

1. 向Kafka发送消息的主要步骤。



3.2.2 数据可靠性保证

1）副本同步策略

如果是半数同步策略，容忍n台节点故障，需要2n+1个副本，而全同步只需要n+1个副本，上面也说了这是Kafka选择全同步策略的主要原因。注意：一个topic的某个分区的副本数包含Leader和follower

两种策略为什么需要的副本数不同呢？

假设有5台机器，现在存在一个topic为A，它有一个分区。

①全同步策略： 5台机器最多容忍4台机器故障，此时由于你选择的是全同步策略，所以topic A 分区下的5个副本（leader + follower）分别位于这5台机器上，那么那台没有故障的机器（也就是broker）的副本（follower副本，假设此时leader故障）就会成为leader继续工作。也即是全同步策略下，容忍n台节点故障，需要n+1个副本。

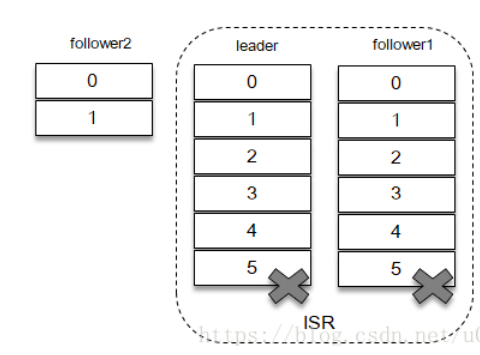
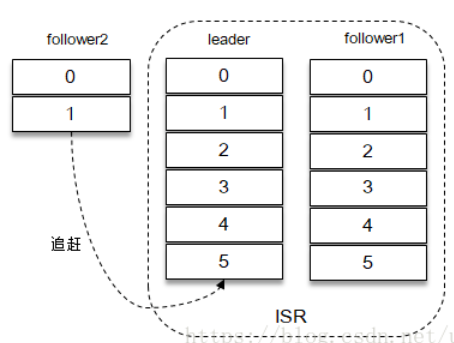
②半数同步策略： 假设n=4台机器发生了故障，副本数为5（n+1），由于是半数机制，所以topic A 分区下的副本数只要存在3个（leader + follower）broker就会发送ack。可是此时四台节点故障，Kafka集群里面只有三个副本，这就不能保证没故障的节点存在topic A 分区下的副本。 那么副本数至少需要设置为多少？副本数应设置为9个。topic 有9个副本，当5个副本完成同步才会发送ack， 这就保证了每个节点都存在topic A 分区下的一个副本，此时如果4台节点故障（假设leader故障），那台没故障的机器就会成为leader继续工作。也即是半同步策略下，容忍n台节点故障，需要2n+1个副本。

1. **ISR 与 unclean.leader.election.enable参数**

unclean.leader.election.enable指的是接不接受非ISR中的副本成为leader

从Kafka 0.11.0.0版本开始参数的默认值由原来的true改为false，这个参数背后到底意味着什么，Kafka的设计者处于什么原因要修改这个默认值？

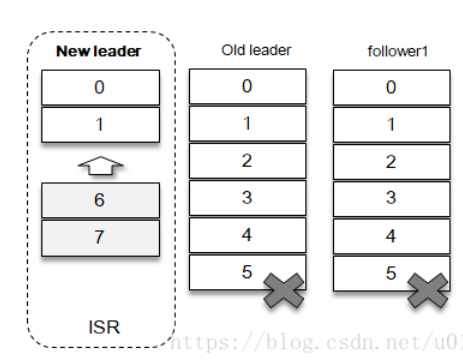
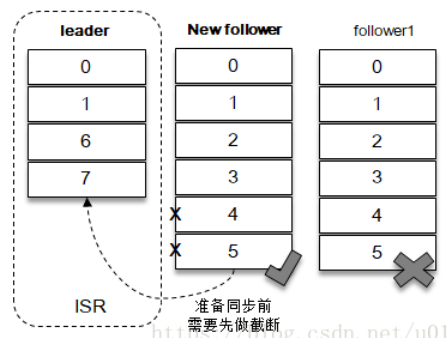
如下左图，某种状态下，follower2副本落后leader副本很多，并且也不在leader副本和follower1副本所在的ISR（In-Sync Replicas）集合之中。follower2副本正在努力的追赶leader副本，以求迅速同步并且能够加入到ISR中。但是很不幸的是，此时ISR中的所有副本都突然下线，情形如下右图所示：



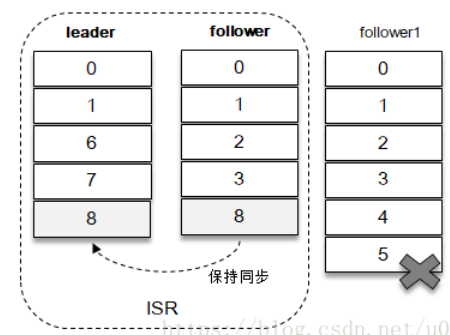
此时follower2副本还在，就会进行新的选举，不过在选举之前首先要判断unclean.leader.election.enable参数的值。如果unclean.leader.election.enable参数的值为false，那么就意味着非ISR中的副本不能够参与选举，此时无法进行新的选举，此时整个分区处于不可用状态。如果unclean.leader.election.enable参数的值为true，那么可以从非ISR集合中选举follower副本称为新的leader。

进一步考虑unclean.leader.election.enable参数为true的情况，在上面的这种情形中follower2副本成了新的leader。随着时间的推进，新的leader副本从客户端收到了新的消息，如上图所示。

此时，原来的leader副本恢复，成为了新的follower副本，准备向新的leader副本同步消息，但是它发现自身的LEO比leader副本的LEO还要大。Kafka中有一个准则，follower副本的LEO是不能够大于leader副本的，所以新的follower副本就需要截断日志至leader副本的LEO处。

新的follower副本需要删除消息4和消息5，之后才能与新的leader副本进行同步。之后新的follower副本和新的leader副本组成了新的ISR集合，参考下图。



原本客户端已经成功的写入了消息4和消息5，而在发生日志截断之后就意味着这2条消息就丢失了，并且新的follower副本和新的leader副本之间的消息也不一致。

也就是说如果unclean.leader.election.enable参数设置为true，就有可能发生数据丢失和数据不一致的情况，Kafka的可靠性就会降低；而如果unclean.leader.election.enable参数设置为false，Kafka的可用性就会降低。具体怎么选择需要读者更具实际的业务逻辑进行权衡

3.3 Kafka消费者

使用 Avro序列化消息

Apache Avro 是种与编程语言无关的序列化格式，目的是提供一种共享数据文件的方式。数据通过与语言无关schema来定义，schema通过 JSON 来描述。在读写文件时需要用到schema, schema一般会被内嵌在数据文件里。

Avro 有一个很有意思的特性是，当 负责写消息的应用程序使用了新的 schema ，负责读消息的应用程序可以继续处理消息而无需做任何改动，这个特性使得它特别适合用在像Kafka 这样的消息系统上。

Kafka 里使用 Avro

遵循通用的结构模式并使”schema 注册表”来达到目的。schema注册表并不属于Kafka

3.3.1 消费方式

3.3.2 分区分配策略

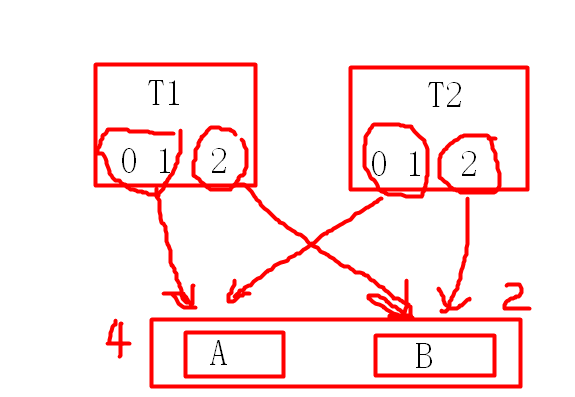
1）轮询

消费者组同时消费T1、T2两个主题Topic里的分区时（组中的消费者都订阅了一样的主题），将两主题所有的分区合起来排个序（分区包含类TopicAndPartition的对象，包含了该分区主题名和分区号等信息，以此构建哈希值用来排序），然后进行轮询。

好处：均衡负载

限制：组内每个消费者订阅的主题，必须是相同的；

1. Range



3.3.3 offset的维护

3.3.4 消费者组案例

3.4 Kafka高效读写数据

3.5 Zookeeper在Kafka中的作用

3.6 Kafka事务

