

二

35 跨集群备份解决方案MirrorMaker

你好，我是胡夕。今天我要和你分享的主题是：Kafka 的跨集群数据镜像工具 MirrorMaker。

一般情况下，我们会使用一套 Kafka 集群来完成业务，但有些场景确实会需要多套 Kafka 集群同时工作，比如为了便于实现灾难恢复，你可以在两个机房分别部署单独的 Kafka 集群。如果其中一个机房出现故障，你就能很容易地把流量打到另一个正常运转的机房下。再比如，你想为地理相近的客户提供低延时的消息服务，而你的主机房又离客户很远，这时你就可以在靠近客户的地方部署一套 Kafka 集群，让这套集群服务你的客户，从而提供低延时的服务。

如果要实现这些需求，除了部署多套 Kafka 集群之外，你还需要某种工具或框架，来帮助你实现数据在集群间的拷贝或镜像。

值得注意的是，**通常我们把数据在单个集群下不同节点之间的拷贝称为备份，而把数据在集群间的拷贝称为镜像（Mirroring）。**

今天，我来重点介绍一下 Apache Kafka 社区提供的 MirrorMaker 工具，它可以帮我们实现消息或数据从一个集群到另一个集群的拷贝。

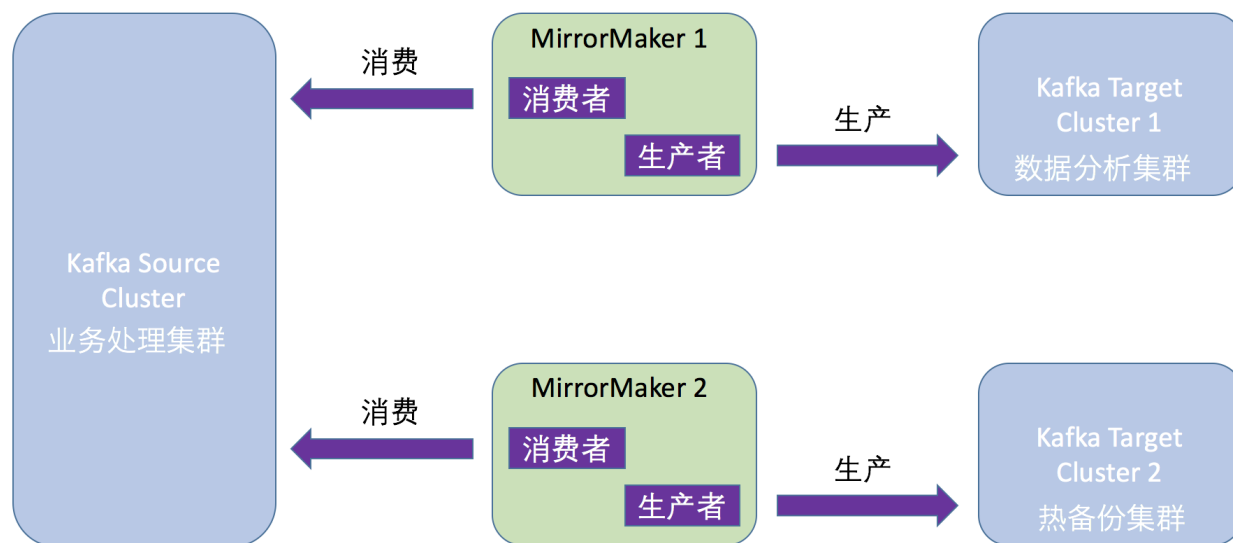
什么是 MirrorMaker?

从本质上说，MirrorMaker 就是一个消费者 + 生产者的程序。消费者负责从源集群（Source Cluster）消费数据，生产者负责向目标集群（Target Cluster）发送消息。整个镜像流程如下图所示：



MirrorMaker 连接的源集群和目标集群，会实时同步消息。当然，你不要认为你只能使用一套 MirrorMaker 来连接上下游集群。事实上，很多用户会部署多套集群，用于实现不同的目的。

我们来看看下面这张图。图中部署了三套集群：左边的源集群负责主要的业务处理；右上角的目标集群可以用于执行数据分析；而右下角的目标集群则充当源集群的热备份。



运行 MirrorMaker

Kafka 默认提供了 MirrorMaker 命令行工具 `kafka-mirror-maker` 脚本，它的常见用法是指定生产者配置文件、消费者配置文件、线程数以及要执行数据镜像的主题正则表达式。比如下面的这个命令，就是一个典型的 MirrorMaker 执行命令。

```
$ bin/kafka-mirror-maker.sh --consumer.config ./config/consumer.properties --produc
```

现在我来解释一下这条命令中各个参数的含义。

- `consumer.config` 参数。它指定了 MirrorMaker 中消费者的配置文件地址，最主要的配置项是 **`bootstrap.servers`**，也就是该 MirrorMaker 从哪个 Kafka 集群读取消息。因为 MirrorMaker 有可能在内部创建多个消费者实例并使用消费者组机制，因此你还需要设置 `group.id` 参数。另外，我建议你额外配置 `auto.offset.reset=earliest`，否则的话，MirrorMaker 只会拷贝那些在它启动之后到达源集群的消息。
- `producer.config` 参数。它指定了 MirrorMaker 内部生产者组件的配置文件地址。通常来说，Kafka Java Producer 很友好，你不需要配置太多参数。唯一的例外依然是 **`bootstrap.servers`**，你必须显式地指定这个参数，配置拷贝的消息要发送到的目标集

群。

- `num.streams` 参数。我个人觉得，这个参数的名字很容易给人造成误解。第一次看到这个参数名的时候，我一度以为 MirrorMaker 是用 Kafka Streams 组件实现的呢。其实并不是。这个参数就是告诉 MirrorMaker 要创建多少个 `KafkaConsumer` 实例。当然，它使用的是多线程的方案，即在后台创建并启动多个线程，每个线程维护专属的消费者实例。在实际使用时，你可以根据你的机器性能酌情设置多个线程。
- `whitelist` 参数。如命令所示，这个参数接收一个正则表达式。所有匹配该正则表达式的主题都会被自动地执行镜像。在这个命令中，我指定了“.*”，这表明我要同步源集群上的所有主题。

MirrorMaker 配置实例

现在，我就在测试环境中为你演示一下 MirrorMaker 的使用方法。

演示的流程大致是这样的：首先，我们会启动两套 Kafka 集群，它们是单节点的伪集群，监听端口分别是 9092 和 9093；之后，我们会启动 MirrorMaker 工具，实时地将 9092 集群上的消息同步镜像到 9093 集群上；最后，我们启动额外的消费者来验证消息是否拷贝成功。

第 1 步：启动两套 Kafka 集群

启动日志如下所示：

```
[2019-07-23 17:01:40,544] INFO Kafka version: 2.3.0
(org.apache.kafka.common.utils.AppInfoParser) [2019-07-23 17:01:40,544] INFO
Kafka commitId: fc1aaa116b661c8a (org.apache.kafka.common.utils.AppInfoParser)
[2019-07-23 17:01:40,544] INFO Kafka startTimeMs: 1563872500540
(org.apache.kafka.common.utils.AppInfoParser) [2019-07-23 17:01:40,545] INFO
[KafkaServer id=0] started (kafka.server.KafkaServer)
```

```
[2019-07-23 16:59:59,462] INFO Kafka version: 2.3.0
(org.apache.kafka.common.utils.AppInfoParser) [2019-07-23 16:59:59,462] INFO
Kafka commitId: fc1aaa116b661c8a (org.apache.kafka.common.utils.AppInfoParser)
[2019-07-23 16:59:59,462] INFO Kafka startTimeMs: 1563872399459
(org.apache.kafka.common.utils.AppInfoParser) [2019-07-23 16:59:59,463] INFO
[KafkaServer id=1] started (kafka.server.KafkaServer)
```

第 2 步：启动 MirrorMaker 工具

在启动 MirrorMaker 工具之前，我们必须准备好刚刚提过的 Consumer 配置文件和 Producer 配置文件。它们的内容分别如下：

```
consumer.properties:
bootstrap.servers=localhost:9092
group.id=mirrormaker
auto.offset.reset=earliest
producer.properties:
bootstrap.servers=localhost:9093
```

现在，我们来运行命令启动 MirrorMaker 工具。

```
$ bin/kafka-mirror-maker.sh --producer.config ../producer.config --consumer.config
WARNING: The default partition assignment strategy of the mirror maker will change
```

请你一定要仔细阅读这个命令输出中的警告信息。这个警告的意思是，在未来版本中，MirrorMaker 内部消费者会使用轮询策略（Round-robin）来为消费者实例分配分区，现阶段使用的默认策略依然是基于范围的分区策略（Range）。Range 策略的思想很朴素，它是将所有分区根据一定的顺序排列在一起，每个消费者依次顺序拿走各个分区。

Round-robin 策略的推出时间要比 Range 策略晚。通常情况下，我们可以认为，社区推出的比较晚的分区分配策略会比之前的策略好。这里的好指的是能实现更均匀的分配效果。该警告信息的最后一部分内容提示我们，**如果我们想提前“享用”轮询策略，需要手动地在 consumer.properties 文件中增加 partition.assignment.strategy 的设置。**

第 3 步：验证消息是否拷贝成功

好了，启动 MirrorMaker 之后，我们可以向源集群发送并消费一些消息，然后验证是否所有的主题都能正确地同步到目标集群上。

假设我们在源集群上创建了一个 4 分区的话题 test，随后使用 kafka-producer-perf-test 脚本模拟发送了 500 万条消息。现在，我们使用下面这两条命令来查询一下，目标 Kafka 集群上是否存在名为 test 的主题，并且成功地镜像了这些消息。

```
$ bin/kafka-run-class.sh kafka.tools.GetOffsetShell --broker-list localhost:9093 --
test:0:0
$ bin/kafka-run-class.sh kafka.tools.GetOffsetShell --broker-list localhost:9093 --
test:0:5000000
```

-1 和 -2 分别表示获取某分区最新的位移和最早的位移，这两个位移值的差值就是这个分区

当前的消息数，在这个例子中，差值是 500 万条。这说明主题 test 当前共写入了 500 万条消息。换句话说，MirrorMaker 已经成功地把这 500 万条消息同步到了目标集群上。

讲到这里，你一定会觉得很奇怪吧，我们明明在源集群创建了一个 4 分区的主题，为什么到了目标集群，就变成单分区了呢？

原因很简单。**MirrorMaker 在执行消息镜像的过程中，如果发现要同步的主题在目标集群上不存在的话，它会根据 Broker 端参数 num.partitions 和 default.replication.factor 的默认值，自动将主题创建出来。**在这个例子中，我们在目标集群上没有创建过任何主题，因此，在镜像开始时，MirrorMaker 自动创建了一个名为 test 的单分区单副本的主题。

在实际使用场景中，我推荐你提前把要同步的所有主题按照源集群上的规格在目标集群上等价地创建出来。否则，极有可能出现刚刚的这种情况，这会导致一些很严重的问题。比如原本在某个分区的消息同步到了目标集群以后，却位于其他的分区中。如果你的消息处理逻辑依赖于这样的分区映射，就必然会出现问题。

除了常规的 Kafka 主题之外，MirrorMaker 默认还会同步内部主题，比如在专栏前面我们频繁提到的位移主题。MirrorMaker 在镜像位移主题时，如果发现目标集群尚未创建该主题，它会根据 Broker 端参数 offsets.topic.num.partitions 和 offsets.topic.replication.factor 的值来制定该主题的规格。默认配置是 50 个分区，每个分区 3 个副本。

在 0.11.0.0 版本之前，Kafka 不会严格依照 offsets.topic.replication.factor 参数的值。这也就是说，如果你设置了该参数值为 3，而当前存活的 Broker 数量少于 3，位移主题依然能被成功创建，只是副本数取该参数值和存活 Broker 数之间的较小值。

这个缺陷在 0.11.0.0 版本被修复了，这就意味着，Kafka 会严格遵守你设定的参数值，如果发现存活 Broker 数量小于参数值，就会直接抛出异常，告诉你主题创建失败。因此，在使用 MirrorMaker 时，你一定要确保这些配置都是合理的。

其他跨集群镜像方案

讲到这里，MirrorMaker 的主要功能我就介绍完了。你大致可以感觉到执行 MirrorMaker 的命令是很简单的，而且它提供的功能很有限。实际上，它的运维成本也比较高，比如主题的管理就非常不便捷，同时也很难将其管道化。

基于这些原因，业界很多公司选择自己开发跨集群镜像工具。我来简单介绍几个。

1. Uber 的 uReplicator 工具

Uber 公司之前也是使用 MirrorMaker 的，但是在使用过程中，他们发现了一些明显的缺

陷，比如 MirrorMaker 中的消费者使用的是消费者组的机制，这不可避免地会碰到很多 Rebalance 的问题。

为此，Uber 自己研发了 uReplicator。它使用 Apache Helix 作为集中式的主题分区管理组件，并且重写了消费者程序，来替换之前 MirrorMaker 下的消费者，使用 Helix 来管理分区的分配，从而避免了 Rebalance 的各种问题。

讲到这里，我个人有个小小的感慨：社区最近正在花大力气去优化消费者组机制，力求改善因 Rebalance 导致的各种场景，但其实，其他框架开发者反而是不用 Group 机制的。他们宁愿自己开发一套机制来维护分区分配的映射。这些都说明 Kafka 中的消费者组还是有很大的提升空间的。

另外，Uber 专门写了一篇[博客](#)，详细说明了 uReplicator 的设计原理，并罗列了社区的 MirrorMaker 工具的一些缺陷以及 uReplicator 的应对方法。我建议你一定要读一读这篇博客。

2.Linkedin 开发的 Brooklin Mirror Maker 工具

针对现有 MirrorMaker 工具不易实现管道化的缺陷，这个工具进行了有针对性的改进，同时也对性能做了一些优化。目前，在 LinkedIn 公司，Brooklin Mirror Maker 已经完全替代了社区版的 MirrorMaker。如果你想深入了解它是如何做到的，我给你推荐一篇[博客](#)，你可以详细阅读一下。

3.Confluent 公司研发的 Replicator 工具

这个工具提供的是企业级的跨集群镜像方案，是市面上已知的功能最为强大的工具，可以便捷地为你提供 Kafka 主题在不同集群间的迁移。除此之外，Replicator 工具还能自动在目标集群上创建与源集群上配置一模一样的主题，极大地方便了运维管理。不过凡事有利就有弊，Replicator 是要收费的。如果你所在的公司预算充足，而且你们关心数据在多个集群甚至是多个数据中心间的迁移质量，不妨关注一下 Confluent 公司的[Replicator 工具](#)。

小结

好了，我们总结一下今天所讲的 MirrorMaker。它是 Apache Kafka 社区提供的跨集群镜像解决方案，主要实现将 Kafka 消息实时从一个集群同步复制或镜像到另一个集群。你可以把 MirrorMaker 应用到很多实际的场景中，比如数据备份、主备集群等。MirrorMaker 本身功能简单，应用灵活，但也有运维成本高、性能差等劣势，因此业界有厂商研发了自己的镜像工具。你可以根据自身的业务需求，选择合适的工具来帮助你完成跨集群的数据备份。

跨集群备份解决方案MirrorMaker

- 从本质上说，MirrorMaker就是一个消费者+生产者的程序。消费者负责从源集群消费数据，生产者负责向目标集群发送消息。
- Kafka提供了MirrorMaker的命令行工具kafka-mirror-maker脚本，它的常见用法是指定生产者配置文件、消费者配置文件、线程数以及要执行数据镜像的主题正则表达式。
- MirrorMaker本身功能简单，应用灵活，但也有运维成本高、性能差等劣势，因此业界有厂商研发了自己的镜像工具，比如Uber的uReplicator工具、LinkedIn的Brooklin Mirror Maker工具和Confluent公司研发的Replicator工具。

[上一页](#)[下一页](#)