

二

31 常见工具脚本大汇总

你好，我是胡夕。今天我要跟你分享的主题是：Kafka 常见的脚本汇总。

命令行脚本概览

Kafka 默认提供了很多个命令行脚本，用于实现各种各样的功能和运维管理。今天我以 2.2 版本为例，详细地盘点下这些命令行工具。下图展示了 2.2 版本提供的所有命令行脚本。

```
connect-distributed.sh
connect-standalone.sh
kafka-acls.sh
kafka-broker-api-versions.sh
kafka-configs.sh
kafka-console-consumer.sh
kafka-console-producer.sh
kafka-consumer-groups.sh
kafka-consumer-perf-test.sh
kafka-delegation-tokens.sh
kafka-delete-records.sh
kafka-dump-log.sh
kafka-log-dirs.sh
kafka-mirror-maker.sh
kafka-preferred-replica-election.sh
kafka-producer-perf-test.sh
kafka-reassign-partitions.sh
```

```
kafka-reassign-partitions.sh
kafka-replica-verification.sh
kafka-run-class.sh
kafka-server-start.sh
kafka-server-stop.sh
kafka-streams-application-reset.sh
kafka-topics.sh
kafka-verifiable-consumer.sh
kafka-verifiable-producer.sh
trogdor.sh
windows
zookeeper-security-migration.sh
zookeeper-server-start.sh
zookeeper-server-stop.sh
zookeeper-shell.sh
```

从图中我们可以知道，2.2 版本总共提供了 30 个 SHELL 脚本。图中的 windows 实际上是个子目录，里面保存了 Windows 平台下的 BAT 批处理文件。其他的.sh 文件则是 Linux 平台下的标准 SHELL 脚本。

默认情况下，不加任何参数或携带 --help 运行 SHELL 文件，会得到该脚本的使用方法说明。下面这张图片展示了 kafka-log-dirs 脚本的调用方法。

```
$ bin/kafka-log-dirs.sh --help
This tool helps to query log directory usage on the specified brokers.
Option                                Description
-----
--bootstrap-server <String: The server (s) to use for bootstrapping> REQUIRED: the server(s) to use for bootstrapping
--broker-list <String: Broker list> The list of brokers to be queried in the form "0,1,2". All brokers in the cluster will be queried if no broker list is specified
--command-config <String: Admin client property file> Property file containing configs to be passed to Admin Client.
--describe Describe the specified log directories on the specified brokers.
--help Print usage information.
--topic-list <String: Topic list> The list of topics to be queried in
```

```
the form "topic1,topic2,topic3". All
topics will be queried if no topic
list is specified (default: )
```

有了这些基础的了解，我来逐一地说明这些脚本的用途，然后再给你详细地介绍一些常见的脚本。

我们先来说说 connect-standalone 和 connect-distributed 两个脚本。这两个脚本是 Kafka Connect 组件的启动脚本。在专栏[第 4 讲]谈到 Kafka 生态时，我曾说过社区提供了 Kafka Connect 组件，用于实现 Kafka 与外部世界系统之间的数据传输。Kafka Connect 支持单节点的 Standalone 模式，也支持多节点的 Distributed 模式。这两个脚本分别是这两种模式下的启动脚本。鉴于 Kafka Connect 不在我们的讨论范围之内，我就不展开讲了。

接下来是 kafka-acls 脚本。它是用于设置 Kafka 权限的，比如设置哪些用户可以访问 Kafka 的哪些主题之类的权限。在专栏后面，我会专门来讲 Kafka 安全设置的内容，到时候我们再细聊这个脚本。

下面是 kafka-broker-api-versions 脚本。**这个脚本的主要目的是验证不同 Kafka 版本之间服务器和客户端的适配性**。我来举个例子，下面这两张图分别展示了 2.2 版本 Server 端与 2.2 版本 Client 端和 1.1.1 版本 Client 端的适配性。

```
huxis-MacBook-Pro:kafka_2.12-2.2.0 huxi$ bin/kafka-broker-api-versions.sh --bootstrap-server localhost:9092
localhost:9092 (id: 0 rack: null) -> {
  Produce(0): 0 to 7 [usable: 7],
  Fetch(1): 0 to 10 [usable: 10],
  ListOffsets(2): 0 to 5 [usable: 5],
  Metadata(3): 0 to 7 [usable: 7],
  LeaderAndIsr(4): 0 to 2 [usable: 2],
```

```
huxis-MacBook-Pro:kafka_2.12-1.1.1 huxi$ bin/kafka-broker-api-versions.sh --bootstrap-server localhost:9092
localhost:9092 (id: 0 rack: null) -> {
  Produce(0): 0 to 7 [usable: 5],
  Fetch(1): 0 to 10 [usable: 7],
  ListOffsets(2): 0 to 5 [usable: 2],
  Metadata(3): 0 to 7 [usable: 5],
  LeaderAndIsr(4): 0 to 2 [usable: 1],
```

我截取了部分输出内容，现在我稍微解释一下这些输出的含义。我们以第一行为例：

Produce(0): 0 to 7 [usable: 7]

“Produce”表示 Produce 请求，生产者生产消息本质上就是向 Broker 端发送 Produce 请求。该请求是 Kafka 所有请求类型中的第一号请求，因此序号是 0。后面的“0 to 7”表示 Produce 请求在 Kafka 2.2 中总共有 8 个版本，序号分别是 0 到 7。“usable: 7”表示当前连入这个 Broker 的客户端 API 能够使用的版本号是 7，即最新的版本。

请注意这两张图中红线部分的差异。在第一张图中，我们使用 2.2 版本的脚本连接 2.2 版本的 Broker，usable 自然是 7，表示能使用最新版本。在第二张图中，我们使用 1.1 版本的

脚本连接 2.2 版本的 Broker，usable 是 5，这表示 1.1 版本的客户端 API 只能发送版本号是 5 的 Produce 请求。

如果你想了解你的客户端版本与服务器端版本的兼容性，那么最好使用这个命令来检验一下。值得注意的是，**在 0.10.2.0 之前，Kafka 是单向兼容的，即高版本的 Broker 能够处理低版本 Client 发送的请求，反过来则不行。自 0.10.2.0 版本开始，Kafka 正式支持双向兼容，也就是说，低版本的 Broker 也能处理高版本 Client 的请求了。**

接下来是 kafka-configs 脚本。对于这个脚本，我想你应该已经很熟悉了，我们在讨论参数配置和动态 Broker 参数时都提到过它的用法，这里我就不再赘述了。

下面的两个脚本是重量级的工具行脚本：kafka-console-consumer 和 kafka-console-producer。在某种程度上，说它们是最常用的脚本也不为过。这里我们暂时先跳过，后面我会重点介绍它们。

关于 producer 和 consumer，成组出现的还有另外一组脚本：kafka-producer-perf-test 和 kafka-consumer-perf-test。它们分别是生产者和消费者的性能测试工具，非常实用，稍后我会重点介绍。

接下来的 kafka-consumer-groups 命令，我在介绍重设消费者组位移时稍有涉及，后面我们来聊聊该脚本的其他用法。

kafka-delegation-tokens 脚本可能不太为人所知，它是管理 Delegation Token 的。基于 Delegation Token 的认证是一种轻量级的认证机制，补充了现有的 SASL 认证机制。

kafka-delete-records 脚本用于删除 Kafka 的分区消息。鉴于 Kafka 本身有自己的自动消息删除策略，这个脚本的实际出场率并不高。

kafka-dump-log 脚本可谓是非常实用的脚本。它能查看 Kafka 消息文件的内容，包括消息的各种元数据信息，甚至是消息体本身。

kafka-log-dirs 脚本是比较新的脚本，可以帮助查询各个 Broker 上的各个日志路径的磁盘占用情况。

kafka-mirror-maker 脚本是帮助你实现 Kafka 集群间的消息同步的。在专栏后面，我会单独用一讲的内容来讨论它的用法。

kafka-preferred-replica-election 脚本是执行 Preferred Leader 选举的。它可以为指定的主题执行“换 Leader”的操作。

kafka-reassign-partitions 脚本用于执行分区副本迁移以及副本文件路径迁移。

kafka-topics 脚本你应该很熟悉了，所有的主题管理操作，都是由该脚本来实现的。

kafka-run-class 脚本则颇为神秘，你可以用这个脚本执行任何带 main 方法的 Kafka 类。在 Kafka 早期的发展阶段，很多工具类都没有自己专属的 SHELL 脚本，比如刚才提到的 kafka-dump-log，你只能通过运行 `kafka-run-class kafka.tools.DumpLogSegments` 的方式来间接实现。如果你用文本编辑器打开 `kafka-dump-log.sh`，你会发现它实际调用的就是这条命令。后来社区逐渐为这些重要的工具类都添加了专属的命令行脚本，现在 `kafka-run-class` 脚本的出场率大大降低了。在实际工作中，你几乎遇不上要直接使用这个脚本的场景了。

对于 `kafka-server-start` 和 `kafka-server-stop` 脚本，你应该不会感到陌生，它们是用于启动和停止 Kafka Broker 进程的。

`kafka-streams-application-reset` 脚本用来给 Kafka Streams 应用程序重设位移，以便重新消费数据。如果你没有用到 Kafka Streams 组件，这个脚本对你来说是没有用的。

`kafka-verifiable-producer` 和 `kafka-verifiable-consumer` 脚本是用来测试生产者和消费者功能的。它们是很“古老”的脚本了，你几乎用不到它们。另外，前面提到的 Console Producer 和 Console Consumer 完全可以替代它们。

剩下的 zookeeper 开头的脚本是用来管理和运维 ZooKeeper 的，这里我就不做过多介绍了。

最后说一下 trogdor 脚本。这是个很神秘的家伙，官网上也不曾出现它的名字。据社区内部资料显示，它是 Kafka 的测试框架，用于执行各种基准测试和负载测试。一般的 Kafka 用户应该用不到这个脚本。

好了，Kafka 自带的所有脚本我全部梳理了一遍。虽然这些描述看起来有点流水账，但是，有了这些基础的认知，我们才能更好地利用这些脚本。下面我就来详细介绍一下重点的脚本操作。

重点脚本操作

生产消息

生产消息使用 `kafka-console-producer` 脚本即可，一个典型的命令如下所示：

```
$ bin/kafka-console-producer.sh --broker-list kafka-host:port --topic test-topic -->
```


在这段命令中，我们指定生产者参数 `acks` 为 `-1`，同时启用了 LZ4 的压缩算法。这个脚本可以很方便地让我们使用控制台来向 Kafka 的指定主题发送消息。

消费消息

下面再来说说数据消费。如果要快速地消费主题中的数据来验证消息是否存在，运行 `kafka-console-consumer` 脚本应该算是最便捷的方法了。常用的命令用法如下：

```
$ bin/kafka-console-consumer.sh --bootstrap-server kafka-host:port --topic test-top
```

注意，在这段命令中，我们指定了 `group` 信息。如果没有指定的话，每次运行 Console Consumer，它都会自动生成一个新的消费者组来消费。久而久之，你会发现你的集群中有大量的以 `console-consumer` 开头的消费者组。通常情况下，你最好还是加上 `group`。

另外，`from-beginning` 等同于将 Consumer 端参数 `auto.offset.reset` 设置成 `earliest`，表明我想从头开始消费主题。如果不指定的话，它会默认从最新位移读取消息。如果此时没有任何新消息，那么该命令的输出为空，你什么都看不到。

最后，我在命令中禁掉了自动提交位移。通常情况下，让 Console Consumer 提交位移是没有意义的，毕竟我们只是用它做一些简单的测试。

测试生产者性能

如果你想要对 Kafka 做一些简单的性能测试。那么不妨试试下面这一组工具。它们分别用于测试生产者和消费者的性能。

我们先说测试生产者的脚本：**`kafka-producer-perf-test`**。它的参数有不少，但典型的命令调用方式是这样的。

```
$ bin/kafka-producer-perf-test.sh --topic test-topic --num-records 10000000 --throu  
  
2175479 records sent, 435095.8 records/sec (424.90 MB/sec), 131.1 ms avg latency, 6  
4190124 records sent, 838024.8 records/sec (818.38 MB/sec), 4.4 ms avg latency, 73.  
10000000 records sent, 737463.126844 records/sec (720.18 MB/sec), 31.81 ms avg late
```

上述命令向指定主题发送了 1 千万条消息，每条消息大小是 1KB。该命令允许你在 `producer-props` 后面指定要设置的生产者参数，比如本例中的压缩算法、延时时间等。

该命令的输出值得好好说一下。它会打印出测试生产者的吞吐量 (MB/s)、消息发送延时以及各种分位数下的延时。一般情况下，消息延时不是一个简单的数字，而是一组分布。或者

说，**我们应该关心延时的概率分布情况，仅仅知道一个平均值是没有意义的**。这就是这里计算分位数的原因。通常我们关注到**99th 分位**就可以了。比如在上面的输出中，99th 值是 604ms，这表明测试生产者生产的消息中，有 99% 消息的延时都在 604ms 以内。你完全可以把这个数据当作这个生产者对外承诺的 SLA。

测试消费者性能

测试消费者也是类似的原理，只不过我们使用的是**kafka-consumer-perf-test**脚本，命令如下：

```
$ bin/kafka-consumer-perf-test.sh --broker-list kafka-host:port --messages 10000000
start.time, end.time, data.consumed.in.MB, MB.sec, data.consumed.in.nMsg, nMsg.sec,
2019-06-26 15:24:18:138, 2019-06-26 15:24:23:805, 9765.6202, 1723.2434, 10000000, 1
```

虽然输出格式有所差别，但该脚本也会打印出消费者的吞吐量数据。比如本例中的 1723MB/s。有点令人遗憾的是，它没有计算不同分位数下的分布情况。因此，在实际使用过程中，这个脚本的使用率要比生产者性能测试脚本的使用率低。

查看主题消息总数

很多时候，我们都想查看某个主题当前的消息总数。令人惊讶的是，Kafka 自带的命令竟然没有提供这样的功能，我们只能“绕道”获取了。所谓的绕道，是指我们必须调用一个未被记录在官网上的命令。命令如下：

```
$ bin/kafka-run-class.sh kafka.tools.GetOffsetShell --broker-list kafka-host:port -

test-topic:0:0
test-topic:1:0

$ bin/kafka-run-class.sh kafka.tools.GetOffsetShell --broker-list kafka-host:port -

test-topic:0:5500000
test-topic:1:5500000
```

我们要使用 Kafka 提供的工具类**GetOffsetShell**来计算给定主题特定分区当前的最早位移和最新位移，将两者的差值累加起来，就能得到该主题当前总的消息数。对于本例来说，test-topic 总的消息数为 5500000 + 5500000，等于 1100 万条。

查看消息文件数据

作为 Kafka 使用者，你是不是对 Kafka 底层文件里面保存的内容很感兴趣？如果是的话，

你可以使用 `kafka-dump-log` 脚本来查看具体的内容。

```
$ bin/kafka-dump-log.sh --files ../data_dir/kafka_1/test-topic-1/000000000000000000
Dumping ../data_dir/kafka_1/test-topic-1/000000000000000000.log
Starting offset: 0
baseOffset: 0 lastOffset: 14 count: 15 baseSequence: -1 lastSequence: -1 producerId
baseOffset: 15 lastOffset: 29 count: 15 baseSequence: -1 lastSequence: -1 producerI
.....
```

如果只是指定 `--files`，那么该命令显示的是消息批次 (RecordBatch) 或消息集合 (MessageSet) 的元数据信息，比如创建时间、使用的压缩算法、CRC 校验值等。

如果我们想深入看一下每条具体的消息，那么就需要显式指定 `--deep-iteration` 参数，如下所示：

```
$ bin/kafka-dump-log.sh --files ../data_dir/kafka_1/test-topic-1/000000000000000000
Dumping ../data_dir/kafka_1/test-topic-1/000000000000000000.log
Starting offset: 0
baseOffset: 0 lastOffset: 14 count: 15 baseSequence: -1 lastSequence: -1 producerId
| offset: 0 CreateTime: 1561597044911 keysize: -1 valuesize: 1024 sequence: -1 head
| offset: 1 CreateTime: 1561597044932 keysize: -1 valuesize: 1024 sequence: -1 head
| offset: 2 CreateTime: 1561597044932 keysize: -1 valuesize: 1024 sequence: -1 head
| offset: 3 CreateTime: 1561597044932 keysize: -1 valuesize: 1024 sequence: -1 head
| offset: 4 CreateTime: 1561597044932 keysize: -1 valuesize: 1024 sequence: -1 head
| offset: 5 CreateTime: 1561597044932 keysize: -1 valuesize: 1024 sequence: -1 head
| offset: 6 CreateTime: 1561597044932 keysize: -1 valuesize: 1024 sequence: -1 head
| offset: 7 CreateTime: 1561597044932 keysize: -1 valuesize: 1024 sequence: -1 head
| offset: 8 CreateTime: 1561597044932 keysize: -1 valuesize: 1024 sequence: -1 head
| offset: 9 CreateTime: 1561597044932 keysize: -1 valuesize: 1024 sequence: -1 head
| offset: 10 CreateTime: 1561597044932 keysize: -1 valuesize: 1024 sequence: -1 hea
| offset: 11 CreateTime: 1561597044932 keysize: -1 valuesize: 1024 sequence: -1 hea
| offset: 12 CreateTime: 1561597044932 keysize: -1 valuesize: 1024 sequence: -1 hea
| offset: 13 CreateTime: 1561597044933 keysize: -1 valuesize: 1024 sequence: -1 hea
| offset: 14 CreateTime: 1561597044933 keysize: -1 valuesize: 1024 sequence: -1 hea
baseOffset: 15 lastOffset: 29 count: 15 baseSequence: -1 lastSequence: -1 producerI
.....
```

在上面的输出中，以竖线开头的就是消息批次下的消息信息。如果你还想看消息里面的实际数据，那么还需要指定 `--print-data-log` 参数，如下所示：

```
$ bin/kafka-dump-log.sh --files ../data_dir/kafka_1/test-topic-1/000000000000000000
```

查询消费者组位移

接下来，我们来看如何使用 `kafka-consumer-groups` 脚本查看消费者组位移。在上一讲讨论重设消费者组位移的时候，我们使用的也是这个命令。当时我们用的是 `--reset-offsets`

参数，今天我们使用的是 **--describe** 参数。假设我们要查询 Group ID 是 test-group 的消费者的位移，那么命令如图所示：

```
i$ bin/kafka-consumer-groups.sh --bootstrap-server localhost:9092 --describe --group test-group
```

TOPIC	PARTITION	CURRENT-OFFSET	LOG-END-OFFSET	LAG	CONSUMER-ID	HOST	CLIENT-ID
test-topic	0	1112985	5000000	3887015	consumer-1-d2a5e537-aec5-4659-a084-33174e1b3c48	/127.0.0.1	consumer-1
test-topic	1	1095015	5000000	3904985	consumer-1-d2a5e537-aec5-4659-a084-33174e1b3c48	/127.0.0.1	consumer-1

图中的 CURRENT-OFFSET 表示该消费者当前消费的最新位移，LOG-END-OFFSET 表示对应分区最新生产消息的位移，LAG 列是两者的差值。CONSUMER-ID 是 Kafka 消费者程序自动生成的一个 ID。截止到 2.2 版本，你都无法干预这个 ID 的生成过程。如果运行该命令时，这个消费者程序已经终止了，那么此列的值为空。

小结

好了，我们小结一下。今天我们一起梳理了 Kafka 2.2 版本自带的所有脚本，我给出了常见的运维操作的工具行命令。希望这些命令对你操作和管理 Kafka 集群有所帮助。另外，我想强调的是，由于 Kafka 一直在不断演进，我们今天提到的命令的用法很可能会随着版本的变迁而发生变化。在具体使用这些命令时，你最好详细地阅读一下它们的 Usage 说明。

Kafka常见的工具脚本

- 生产消息：使用kafka-console-producer脚本。
- 消费消息：运行kafka-console-consumer脚本。
- 测试生产者性能：使用kafka-producer-perf-test脚本。
- 测试消费者性能：使用kafka-consumer-perf-test脚本。
- 查看主题消息总数：使用Kafka提供的工具类GetOffsetShell来计算给定主题特定分区当前的最早位移和最新位移，将两者的差值累加起来。
- 查看消息独立性数据：使用kafka-dump-log脚本。加

● 具有消息入口数据：使用kafka-group-log脚本。如

果想深入查看每条具体的消息，那么就需要显式指定--deep-iteration参数。

- 查询消费者组位移：使用kafka-consumer-groups脚本。

[上一页](#)[下一页](#)