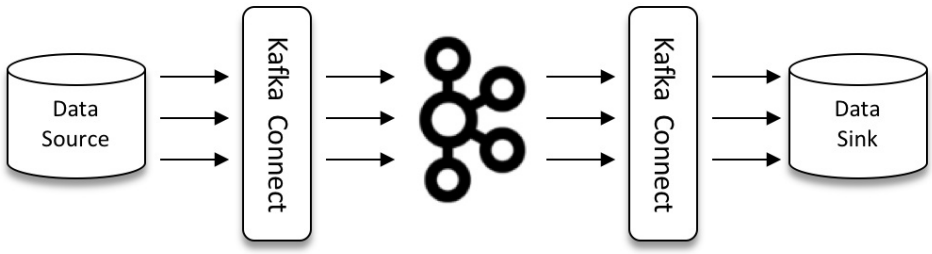
Kafka Connect

Kafka Connect 是一个工具，它为在 Kafka 和外部数据存储系统之间移动数据提供了一种可靠的且可伸缩的实现方式。Kafka Connect 可以简单快捷地将数据从 Kafka 中导入或导出，数据范围涵盖关系型数据库、日志和度量数据、Hadoop 和数据仓库、NoSQL 数据存储、搜索索引等。相对于生产者和消费者客户端而言，Kafka Connect 省掉了很多开发的工作，尤其是编码部分，这使得应用开发人员更容易上手。

Kafka Connect 有两个核心概念：Source 和 Sink。参考下图，Source 负责导入数据到 Kafka，Sink 负责从 Kafka 导出数据，它们都被称为 Connector（连接器）。



在 Kafka Connect 中还有两个重要的概念：Task 和 Worker。Task 是 Kafka Connect 数据模型的主角，每一个 Connector 都会协调一系列的 Task 去执行任务，Connector 可以把一项工作分割成许多 Task，然后把 Task 分发到各个 Worker 进程中去执行（分布式模式下），Task 不保存自己的状态信息，而是交给特定的 Kafka 主题去保存。Connector 和 Task 都是逻辑工作单位，必须安排在进程中执行，而在 Kafka Connect 中，这些进程就是 Worker。

Kafka Connect 提供了以下特性。

- 通用性：规范化其他数据系统与 Kafka 的集成，简化了连接器的开发、部署和管理。
- 支持独立模式（standalone）和分布式模式（distributed）。
- REST 接口：使用 REST API 提交和管理 Connector。
- 自动位移管理：自动管理位移提交，不需要开发人员干预，降低了开发成本。
- 分布式和可扩展性：Kafka Connect 基于现有的组管理协议来实现扩展 Kafka Connect 集群。
- 流式计算/批处理的集成。

独立模式

Kafka 中的 connect-standalone.sh 脚本用来实现以独立的模式运行 Kafka Connect。在独立模式下所有的操作都是在一个进程中完成的，这种模式非常适合测试或功能验证的场景。由于是单进程，所以独立模式无法充分利用 Kafka 自身所提供的负载均衡和高容错等特性。

在执行这个脚本时需要指定两个配置文件：一个是用于 Worker 进程运行的相关配置文件；另一个是指定 Source 连接器或 Sink 连接器的配置文件，可以同时指定多个连接器配置，每个连接器配置文件对应一个连接器，因此要保证连接器名称全局唯一，连接器名称通过 name 参数指定。

下面我们先来了解一下 Source 连接器的用法：将文件 source.txt 中的内容通过 Source 连接器写入 Kafka 的主题 topic-connect。首先修改用于 Worker 进程运行的配置文件（$KAFKA_HOME/config/connect-standalone.properties），内容参考如下：

```
bootstrap.servers=localhost:9092
key.converter=org.apache.kafka.connect.json.JsonConverter
value.converter=org.apache.kafka.connect.json.JsonConverter
key.converter.schemas.enable=true
value.converter.schemas.enable=true
offset.storage.file.filename=/tmp/connect.offsets
offset.flush.interval.ms=10000
```

bootstrap.servers 参数用来配置与 Kafka 集群连接的地址。key.converter 和 value.converter 参数指定 Kafka 消息中 key 和 value 的格式转化类，本例中使用 JsonConverter 来将每一条消息的 key 和 value 都转化成 JSON 格式。key.converter.schemas.enable 和 value.converter.schemas.enable 参数用来指定 JSON 消息中是否可以包含 schema。offset.storage.file.filename 参数用于指定保存偏移量的文件路径。offset.flush.interval.ms 参数用于设定提交偏移量的频率。

接下来修改 Source 连接器的配置文件（$KAFKA_HOME/config/connect-file-source.properties），内容参考如下：

```
name=local-file-source
connector.class=FileStreamSource
tasks.max=1
file=/opt/kafka_2.11-2.0.0/source.txt
topic=topic-connect
```

name 参数用来配置连接器的名称。connector.class 用来设置连接器类的全限定名称，有时候设置为类名也是可以的，Kafka Connect 会在 classpath 中自动搜索这个类并加载。Kafka 中默认只提供了

与文件相关的连接器，如果要实现与其他数据存储系统相连接，那么可以参考文件连接器的具体实现来自定义一套连接器，或者搜寻开源的实现，比如 Confluent 公司提供的一些产品：

- kafka-connect-elasticsearch（https://github.com/confluentinc/kafka-connect-elasticsearch）；
- kafka-connect-jdbc（https://github.com/confluentinc/kafka-connect-jdbc）；
- kafka-connect-hdfs（https://github.com/confluentinc/kafka-connect-hdfs）；
- kafka-connect-storage-cloud（https://github.com/confluentinc/kafka-connect-storage-cloud）。

task.max 参数指定了 Task 的数量。file 参数指定该连接器数据源文件路径，这里指定了 Kafka 根目录下的 source.txt 文件，在启动连接器前需要先创建好它。topic 参数设置连接器把数据导入哪个主题，如果该主题不存在，则连接器会自动创建，不过建议最好还是提前手工创建该主题。比如，对本例中的主题 topic-connect 而言，可以事先创建，它的详细信息如下：

```
[root@node1 kafka_2.11-2.0.0]# bin/kafka-
topics.sh --zookeeper localhost:2181/kafka --
create --topic topic-connect --replication-factor
1 --partitions 1
Created topic "topic-connect".

[root@node1 kafka_2.11-2.0.0]# bin/kafka-
topics.sh --zookeeper localhost:2181/kafka --
describe --topic topic-connect
Topic:topic-connect PartitionCount:1
ReplicationFactor:1 Configs:
    Topic: topic-connect    Partition: 0
Leader: 0   Replicas: 0 Isr: 0
```

接下来就可以启动 Source 连接器了，示例如下：

```
[root@node1 kafka_2.11-2.0.0]# bin/connect-
standalone.sh config/connect-
standalone.properties config/connect-file-
source.properties
```

连接器启动之后，向 source.txt 文件中输入两条句子：

```
[root@node1 kafka_2.11-2.0.0]# echo "hello kafka
connect">> source.txt
[root@node1 kafka_2.11-2.0.0]# echo "hello kafka
streams">> source.txt
```

之后可以观察主题 topic-connect 中是否包含这两条消息。对于这个示例，我们既可以使用 kafka-console-consumer.sh 脚本，也可以使用 kafka-dump-log.sh 脚本来查看内容。这里再来回顾一下 kafka-dump-log.sh 脚本的用法：

```
[root@node1 kafka_2.11-2.0.0]# bin/kafka-dump-
log.sh --files /tmp/kafka-logs/topic-connect-
0/00000000000000000000.log --print-data-log
Dumping /tmp/kafka-logs/topic-connect-
0/00000000000000000000.log

Starting offset: 0
offset: 0 position: 0 CreateTime: 1540368601287
isvalid: true keysize: 30 valuesize: 77 magic: 2
compresscodec: NONE producerId: -1 producerEpoch:
-1 sequence: -1 isTransactional: false
headerKeys: [] key:
{"schema":null,"payload":null} payload:
{"schema":
{"type":"string","optional":false},"payload":"hel
lo kafka connect"}
offset: 1 position: 177 CreateTime: 1540368621321
isvalid: true keysize: 30 valuesize: 77 magic: 2
compresscodec: NONE producerId: -1 producerEpoch:
-1 sequence: -1 isTransactional: false
headerKeys: [] key:
{"schema":null,"payload":null} payload:
{"schema":
{"type":"string","optional":false},"payload":"hel
lo kafka streams"}
```

可以看到主题 topic-connect 中的消息格式为 JSON 字符串并且带有对应的 schema 信息，这一点和在 config/connect-standalone.properties 配置的内容一一对应。

我们再来看一下 Sink 连接器的用法：将主题 topic-connect 中的内容通过 Sink 连接器写入文件 sink.txt。这里对 config/connect-standalone.properties 文件稍做修改，参考如下：

```
bootstrap.servers=localhost:9092
key.converter=org.apache.kafka.connect.storage.StringConverter
value.converter=org.apache.kafka.connect.storage.StringConverter
key.converter.schemas.enable=true
value.converter.schemas.enable=true
offset.storage.file.filename=/tmp/connect.offsets
offset.flush.interval.ms=10000
```

这里将 Kafka 消息中的 key 和 value 的格式转化类指定为 StringConverter。

紧接着我们再配置 Sink 连接器的配置文件（$KAFKA_HOME/config/connect-file-sink.properties），内容参考如下（注意与 Source 连接器配置的区别）：

```
name=local-file-sink
connector.class=FileStreamSink
tasks.max=1
file=/opt/kafka_2.11-2.0.0/sink.txt
topics=topic-connect
```

接下来就可以启动 Sink 连接器了，示例如下：

```
[root@node1 kafka_2.11-2.0.0]# bin/connect-standalone.sh
    config/connect-standalone.properties
config/connect-file-sink.properties
```

我们往主题 topic-connect 中发送一条消息：

```
[root@node1 kafka_2.11-2.0.0]# bin/kafka-console-producer.sh --broker-list localhost:9092 --topic topic-connect
>hello kafka
>
```

进而就可以在 sink.txt 文件中看到这条消息：

```
[root@node1 kafka_2.11-2.0.0]# cat sink.txt
hello kafka
```

REST API

我们可以通过 Kafka Connect 提供的基于 REST 风格的 API 接口来管理连接器，默认端口号为8083，可以通过 Worker 进程的配置文件中的 rest.port 参数来修改端口号。Kafka Connect REST API 接口如下表所示。

REST API	释 义
GET /	查看 Kafka 集群版本信息
GET /connectors	查看当前活跃的连接器列表，显示连接器的名字
POST /connectors	根据指定配置，创建一个新的连接器
GET /connectors/{name}	查看指定连接器的信息
GET /connectors/{name}/config	查看指定连接器的配置信息
PUT /connectors/{name/config	修改指定连接器的配置信息

GET /connectors/{name}/statue	查看指定连接器的状态
POST /connectors/{name}/restart	重启指定的连接器
PUT /connectors/{name}/pause	暂停指定的连接器
GET /connectors/{name}/tasks	查看指定连接器正在运行的 Task
POST /connectors/{name}/tasks	修改 Task 的配置
GET /connectors/{name}/tasks/{taskId}/status	查看指定连接器中指定 Task 的状态
POST /connectors/{name}/tasks/{tasked}/restart	重启指定连接器中指定的 Task
DELETE /connectors/{name}	删除指定的连接器

简单示例如下，更多的 REST API 调用示例可以参考下一节的内容。

```
[root@node1 kafka_2.11-2.0.0]# curl
http://localhost:8083/
{"version":"2.0.0","commit":"3402a8361b734732","k
afka_cluster_id":"Cjr-rkl5SLClosMiOfMpqw"}

[root@node1 kafka_2.11-2.0.0]# curl
http://localhost:8083/connectors
["local-file-source"]
```

分布式模式

与独立模式不同，分布式模式天然地结合了 Kafka 提供的负载均衡和故障转移功能，能够自动在多个节点机器上平衡负载。不过，以分布式模式启动的连接器并不支持在启动时通过加载连接器配置文件来创建一个连接器，只能通过访问 REST API 来创建连接器。

在运行分布式模式的连接器前，同样要修改 Worker 进程的相关配置文件（$KAFKA_HOME/ config/connect-distributed.properties），内容参考如下：

```
bootstrap.servers=localhost1:9092,
localhost2:9092, localhost3:9092
group.id=connect-cluster
key.converter=org.apache.kafka.connect.json.JsonC
onverter
value.converter=org.apache.kafka.connect.json.Jso
nConverter
(….省略若干)
```

之后启动分布式模式，这里的运行脚本也变成了对应的 connect-distributed.sh，示例如下：

```
[root@node1 kafka_2.11-2.0.0]# bin/connect-
distributed.sh
      config/connect-distributed.properties
```

接下来创建一个 Source 连接器，此前先要设定好这个连接器的相关配置，内容如下：

```
{
    "name":"local-file-distribute-source",
    "config":{
        "topic":"topic-distribute-source",
        "connector.class":"FileStreamSource",

"key.converter":"org.apache.kafka.connect.storage
.StringConverter",

"value.converter":"org.apache.kafka.connect.stora
ge.StringConverter",
        "converter.internal.key.converter":
"org.apache.kafka.connect.storage.StringConverter"
,
        "converter.internal.value.converter":
"org.apache.kafka.connect.storage.StringConverter"
,
        "file":"/opt/kafka_2.11-2.0.0/distribute-
source.txt"
    }
}
```

这个连接器从 distribute-source.txt 文件中读取内容进而传输到主题 topic-distribute-source 中，在创建连接器前确保 distribute-source.txt 文件和主题 topic-distribute-source 都已创建完毕。接下来调用 POST /connectors 接口来创建指定的连接器，示例如下：

```
[root@node1 kafka_2.11-2.0.0]# curl -i -X POST -H
"Content-Type:application/ json" -H
"Accept:application/json" -d '{"name":"local-
file-distribute-source", "config":
{"topic":"topic-distribute-
```

```
source","connector.class":"FileStreamSource","key
.converter":"org.apache.kafka.connect.storage.Str
ingConverter","value.converter":"org.apache.kafka
.connect.storage.StringConverter","converter.inte
rnal.key.converter":"org.apache.kafka.connect.sto
rage.StringConverter","converter.internal.value.c
onverter":"org.apache.kafka.connect.storage.Strin
gConverter","file":"/opt/kafka_2.11-
2.0.0/distribute-source.txt"}}'
http://localhost:8083/connectors
HTTP/1.1 201 Created
Date: Wed, 24 Oct 2018 09:38:12 GMT
Location: http://localhost:8083/connectors/local-
file-distribute-source
Content-Type: application/json
Content-Length: 598
Server: Jetty(9.4.11.v20180605)

{"name":"local-file-distribute-source","config":
{"topic":"topic-distribute-
source","connector.class":"FileStreamSource","key
.converter":"org.apache.kafka.connect.storage.Str
ingConverter","value.converter":"org.apache.kafka
.connect.storage.StringConverter","converter.inte
rnal.key.converter":"org.apache.kafka.connect.sto
rage.StringConverter","converter.internal.value.c
onverter":"org.apache.kafka.connect.storage.Strin
gConverter","file":"/opt/kafka_2.11-
2.0.0/distribute-source.txt","name":"local-file-
distribute-source"},"tasks":[{"connector":"local-
file-distribute-source","task":0}],"type":null}
```

接下来就可以向 distribute-source.txt 文件中写入内容，然后订阅消费主题 topic-distribute- source 中的消息来验证是否成功。在使用完毕之后，我们可以调用 DELETE /connectors/ {name} 接口来删除对应的连接器：

```
[root@node1 kafka_2.11-2.0.0]# curl -i -X DELETE
http://localhost:8083/ connectors/local-file-
distribute-source
HTTP/1.1 204 No Content
Date: Wed, 24 Oct 2018 09:42:47 GMT
Server: Jetty(9.4.11.v20180605)

[root@node1 kafka_2.11-2.0.0]# curl -i
http://localhost:8083/connectors
HTTP/1.1 200 OK
Date: Wed, 24 Oct 2018 09:43:05 GMT
Content-Type: application/json
Content-Length: 2
Server: Jetty(9.4.11.v20180605)

[]
```

读者可以自行尝试分布式模式下 Sink 连接器的使用方法。

在向 Kafka 写入数据或从 Kafka 读取数据时，要么使用普通的生产者和消费者客户端，要么使用 Kafka Connect，那么在不同场景下到底使用哪一种呢？Kafka 客户端需要内嵌到业务应用程序里，应用程序需要经常修改以便灵活地将数据推送到 Kafka 或从 Kafka 中消费消息，适用于开发人员。如果要将 Kafka 连接到数据存储系统中，可以使用 Kafka Connect，因为在这种场景下往往也不需要修改对应的代码，适用于非开发人员，他们可以通过配置连接器的方式实现相应的功能。