安装与配置

本节详细介绍 Kafka 运行环境的搭建，为了节省篇幅，本节的内容以 Linux CentOS 作为安装演示的操作系统，其他 Linux 系列的操作系统也可以参考本节的内容。具体的操作系统的信息如下：

```
[root@node1 ~]# uname -a
Linux node1 2.6.32-504.23.4.el6.x86_64 #1 SMP Tue
Jun 9 20:57:37 UTC 2015 x86_64 x86_64 x86_64
GNU/Linux
[root@node1 ~]# cat /etc/issue
CentOS release 6.6 (Final)
Kernel \r on an \m
```

由图1-1可知，搭建 Kafka 运行环境还需要涉及 ZooKeeper，Kafka 和 ZooKeeper 都是运行在 JVM 之上的服务，所以还需要安装 JDK。Kafka 从2.0.0版本开始就不再支持 JDK7 及以下版本，本节就以 JDK8 为例来进行演示。

1. JDK的安装与配置

很多学习 Kafka 的读者也都是 JVM 系语言的支持者，如果你的操作系统中已经安装了 JDK8 及以上版本则可以跳过这段内容。

安装 JDK 的第一步就是下载 JDK 1.8的安装包，可以进入 Oracle 官网页面进行下载。示例中选择的安装包是 jdk-8u181-linux-x64.tar.gz，我们这里将其先复制至/opt 目录下，本书所有与安装有关的操作都在这个目录下进行。

其次将/opt 目录下的安装包解压，相关信息如下：

```
[root@node1 opt]# ll jdk-8u181-linux-x64.tar.gz
-rw-r--r-- 1 root root 185646832 Aug 31 14:48
jdk-8u181-linux-x64.tar.gz
[root@node1 opt]# tar zxvf jdk-8u181-linux-
x64.tar.gz
# 解压之后当前/opt目录下生成一个名为jdk1.8.0_181的文件夹
[root@node1 opt]# cd jdk1.8.0_181/
[root@node1 jdk1.8.0_181]# pwd
/opt/jdk1.8.0_181
# 上面这个就是当前JDK8的安装目录
```

然后配置 JDK 的环境变量。修改/etc/profile 文件并向其中添加如下配置：

```
export JAVA_HOME=/opt/jdk1.8.0_181
export JRE_HOME=$JAVA_HOME/jre
export PATH=$PATH:$JAVA_HOME/bin
export
CLASSPATH=./://$JAVA_HOME/lib:$JRE_HOME/lib
```

再执行 source /etc/profile 命令使配置生效，最后可以通过 java -version命令验证 JDK 是否已经安装配置成功。如果安装配置成功，则会正确显示出 JDK 的版本信息，参考如下：

```
[root@node1 ~]# java -version
java version "1.8.0_181"
Java(TM) SE Runtime Environment (build 1.8.0_181-
b13)
Java HotSpot(TM) 64-Bit Server VM (build 25.181-
b13, mixed mode)
```

2. ZooKeeper安装与配置

ZooKeeper 是安装 Kafka 集群的必要组件，Kafka 通过 ZooKeeper 来实施对元数据信息的管理，包括集群、broker、主题、分区等内容。

ZooKeeper 是一个开源的分布式协调服务，是 Google Chubby的一个开源实现。分布式应用程序可以基于 ZooKeeper 实现诸如数据发布/订阅、负载均衡、命名服务、分布式协调/通知、集群管理、Master 选举、配置维护等功能。

在 ZooKeeper 中共有3个角色：leader、follower 和 observer，同一时刻 ZooKeeper 集群中只会有一个 leader，其他的都是 follower 和 observer。observer 不参与投票，默认情况下 ZooKeeper 中只有 leader 和 follower 两个角色。更多相关知识可以查阅 ZooKeeper 官方网站来获得。

安装 ZooKeeper 的第一步也是下载相应的安装包，安装包可以从官网中获得，示例中使用的安装包是 zookeeper-3.4.12.tar.gz，同样将其复制到/opt 目录下，然后解压缩，参考如下：

```
[root@node1 opt]# ll zookeeper-3.4.12.tar.gz
-rw-r--r-- 1 root root 36667596 Aug 31 15:55
zookeeper-3.4.12.tar.gz
[root@node1 opt]# tar zxvf zookeeper-
3.4.12.tar.gz
# 解压之后当前/opt目录下生成一个名为zookeeper-3.4.12的
文件夹
[root@node1 opt]# cd zookeeper-3.4.12
[root@node1 zookeeper-3.4.12]# pwd
/opt/zookeeper-3.4.12
```

第二步，向/etc/profile 配置文件中添加如下内容，并执行 source /etc/profile 命令使配置生效：

```
export ZOOKEEPER_HOME=/opt/zookeeper-3.4.12
export PATH=$PATH:$ZOOKEEPER_HOME/bin
```

第三步，修改 ZooKeeper 的配置文件。首先进入 $ZOOKEEPER_HOME/conf目录，并将zoo_sample.cfg 文件修改为 zoo.cfg：

```
[root@node1 zookeeper-3.4.12]# cd conf
[root@node1 conf]# cp zoo_sample.cfg zoo.cfg
```

然后修改 zoo.cfg 配置文件，zoo.cfg 文件的内容参考如下：

```
# ZooKeeper服务器心跳时间，单位为ms
tickTime=2000
# 投票选举新leader的初始化时间
initLimit=10
# leader与follower心跳检测最大容忍时间，响应超过
syncLimit*tickTime，leader认为
# follower“死掉”，从服务器列表中删除follower
syncLimit=5
# 数据目录
dataDir=/tmp/zookeeper/data
# 日志目录
dataLogDir=/tmp/zookeeper/log
# ZooKeeper对外服务端口
clientPort=2181
```

默认情况下，Linux 系统中没有/tmp/zookeeper/data 和/tmp/zookeeper/log 这两个目录，所以接下来还要创建这两个目录：

```
[root@node1 conf]# mkdir -p /tmp/zookeeper/data
[root@node1 conf]# mkdir -p /tmp/zookeeper/log
```

第四步，在${dataDir}目录（也就是/tmp/zookeeper/data）下创建一个 myid 文件，并写入一个数值，比如0。myid 文件里存放的是服务器的编号。

第五步，启动 Zookeeper 服务，详情如下：

```
[root@node1 conf]# zkServer.sh start
JMX enabled by default
Using config: /opt/zookeeper-
3.4.6/bin/../conf/zoo.cfg
Starting zookeeper ... STARTED
```

可以通过 zkServer.sh status 命令查看 Zookeeper 服务状态，示例如下：

```
[root@node1 ]# zkServer.sh status
JMX enabled by default
Using config: /opt/zookeeper-
3.4.12/bin/../conf/zoo.cfg
Mode: Standalone
```

以上是关于 ZooKeeper 单机模式的安装与配置，一般在生产环境中使用的都是集群模式，集群模式的配置也比较简单，相比单机模式而言只需要修改一些配置即可。下面以3台机器为例来配置一个 ZooKeeper 集群。首先在这3台机器的 /etc/hosts 文件中添加3台集群的IP地址与机器域名的映射，示例如下（3个IP地址分别对应3台机器）：

```
192.168.0.2 node1
192.168.0.3 node2
192.168.0.4 node3
```

然后在这3台机器的 zoo.cfg 文件中添加以下配置：

```
server.0=192.168.0.2:2888:3888
server.1=192.168.0.3:2888:3888
server.2=192.168.0.4:2888:3888
```

为了便于讲解上面的配置，这里抽象出一个公式，即server.A=B:C:D。其中A是一个数字，代表服务器的编号，就是前面所说的 myid 文件里面的值。集群中每台服务器的编号都必须唯一，所以要保证每台服务器中的 myid 文件中的值不同。B代表服务器的IP地址。C表示服务器与集群中的 leader 服务器交换信息的端口。D表示选举时服务器相互通信的端口。如此，集群模式的配置就告一段落，可以在这3台机器上各自执行 zkServer.sh start 命令来启动服务。

3. Kafka的安装与配置

在安装完 JDK 和 ZooKeeper 之后，就可以执行 Kafka broker 的安装了，首先也是从官网中下载安装包，示例中选用按照包的是kafka_2.11-2.0.0.tgz，将其复制至/opt 目录下并进行解压缩，示例如下：

```
[root@node1 opt]# ll kafka_2.11-2.0.0.tgz
-rw-r--r-- 1 root root 55751827 Jul 31 10:45
kafka_2.11-2.0.0.tgz
[root@node1 opt]# tar zxvf kafka_2.11-2.0.0.tgz
# 解压之后当前/opt目录下生成一个名为kafka_2.11-2.0.0的
文件夹
[root@node1 opt]# cd kafka_2.11-2.0.0
[root@node1 kafka_2.11-2.0.0]#
# Kafka的根目录$KAFKA_HOME即为/opt/kafka_2.11-
2.0.0，可以将Kafka_HOME添加到/etc/profile文件中，具体
做法可以参考前面JDK和ZooKeeper的安装示例
```

接下来需要修改 broker 的配置文件 $KAFKA_HOME/conf/server.properties。主要关注以下几个配置参数即可：

```
# broker的编号，如果集群中有多个broker，则每个broker的编号需要设置的不同
broker.id=0
# broker对外提供的服务入口地址
listeners=PLAINTEXT://localhost:9092
# 存放消息日志文件的地址
log.dirs=/tmp/kafka-logs
# Kafka所需的ZooKeeper集群地址，为了方便演示，我们假设Kafka和ZooKeeper都安装在本机
zookeeper.connect=localhost:2181/kafka
```

如果是单机模式，那么修改完上述配置参数之后就可以启动服务。如果是集群模式，那么只需要对单机模式的配置文件做相应的修改即可：确保集群中每个 broker 的 broker.id 配置参数的值不一样，以及 listeners 配置参数也需要修改为与 broker 对应的IP地址或域名，之后就可以各自启动服务。注意，在启动 Kafka 服务之前同样需要确保 zookeeper.connect 参数所配置的 ZooKeeper 服务已经正确启动。

启动 Kafka 服务的方式比较简单，在$KAFKA_HOME 目录下执行下面的命令即可：

```
bin/kafka-server-start.sh
config/server.properties
```

如果要在后台运行 Kafka 服务，那么可以在启动命令中加入 -daemon 参数或&字符，示例如下：

```
bin/kafka-server-start.sh -daemon
config/server.properties
# 或者
bin/kafka-server-start.sh
config/server.properties &
```

可以通过 jps 命令查看 Kafka 服务进程是否已经启动，示例如下：

```
[root@node1 kafka_2.11-2.0.0]# jps -l
23152 sun.tools.jps.Jps
16052
org.apache.zookeeper.server.quorum.QuorumPeerMain
22807 kafka.Kafka  # 这个就是Kafka服务端的进程
```

jps 命令只是用来确认 Kafka 服务的进程已经正常启动。它是否能够正确地对外提供服务，还需要通过发送和消费消息来进行验证，验证的过程可以参考下面的内容。

生产与消费

由第1节的内容可知，生产者将消息发送至 Kafka 的主题中，或者更加确切地说应该是主题的分区中，而消费者也是通过订阅主题从而消费消息的。在演示生产与消费消息之前，需要创建一个主题作为消息的载体。

Kafka 提供了许多实用的脚本工具，存放在 $KAFKA_HOME 的 bin 目录下，其中与主题有关的就是 kafka-topics.sh 脚本，下面我们用它演示创建一个分区数为4、副本因子为3的主题 topic-demo，示例如下（Kafka集群模式下，broker数为3）：

```
[root@node1 kafka_2.11-2.0.0]# bin/kafka-
topics.sh --zookeeper localhost: 2181/kafka --
create --topic topic-demo --replication-factor 3
--partitions 4

Created topic "topic-demo".
```

其中 --zookeeper 指定了 Kafka 所连接的 ZooKeeper 服务地址，--topic 指定了所要创建主题的名称，--replication-factor 指定了副本因子，--partitions 指定了分区个数，--create 是创建主题的动作指令。

还可以通过 --describe 展示主题的更多具体信息，示例如下：

```
[root@node1 kafka_2.11-2.0.0]# bin/kafka-
topics.sh --zookeeper localhost: 2181/kafka --
describe --topic topic-demo

Topic:topic-demo    PartitionCount:4
ReplicationFactor:3 Configs:
    Topic: topic-demo   Partition: 0    Leader: 2
Replicas: 2,1,0 Isr: 2,1,0
    Topic: topic-demo   Partition: 1    Leader: 0
Replicas: 0,2,1 Isr: 0,2,1
    Topic: topic-demo   Partition: 2    Leader: 1
Replicas: 1,0,2 Isr: 1,0,2
    Topic: topic-demo   Partition: 3    Leader: 2
Replicas: 2,0,1 Isr: 2,0,1
```

创建主题 topic-demo 之后我们再来检测一下 Kafka 集群是否可以正常地发送和消费消息。$KAFKA_HOME/bin 目录下还提供了两个脚本 kafka-console-producer.sh 和 kafka-console-

consumer.sh，通过控制台收发消息。首先我们打开一个 shell 终端，通过 kafka-console-consumer.sh 脚本来订阅主题 topic-demo，示例如下：

```
[root@node1 kafka_2.11-2.0.0]# bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic topic-demo
```

其中 --bootstrap-server 指定了连接的 Kafka 集群地址，--topic 指定了消费者订阅的主题。目前主题 topic-demo 尚未有任何消息存入，所以此脚本还不能消费任何消息。

我们再打开一个 shell 终端，然后使用 kafka-console-producer.sh 脚本发送一条消息“Hello, Kafka!”至主题 topic-demo，示例如下：

```
[root@node1 kafka_2.11-2.0.0]# bin/kafka-console-producer.sh --broker-list localhost:9092 --topic topic-demo
>Hello, Kafka!
>
```

其中 --broker-list 指定了连接的 Kafka 集群地址，--topic 指定了发送消息时的主题。示例中的第二行是通过人工键入的方式输入的，按下回车键后会跳到第三行，即“>”字符处。此时原先执行 kafka-console-consumer.sh 脚本的 shell 终端中出现了刚刚输入的消息“Hello, Kafka!”，示例如下：

```
[root@node1 kafka_2.11-2.0.0]# bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic topic-demo
Hello, Kafka!
```

读者也可以通过输入一些其他自定义的消息来熟悉消息的收发及这两个脚本的用法。不过这两个脚本一般用来做一些测试类的工作，在实际应用中，不会只是简单地使用这两个脚本来做复杂的与业务逻辑相关的消息生产与消费的工作，具体的工作还需要通过编程的手段来实施。下面就以 Kafka 自身提供的 Java 客户端来演示消息的收发，与 Kafka 的 Java 客户端相关的 Maven 依赖如下：

```
<dependency>
    <groupId>org.apache.kafka</groupId>
    <artifactId>kafka-clients</artifactId>
    <version>2.0.0</version>
</dependency>
```

要往 Kafka 中写入消息，首先要创建一个生产者客户端实例并设置一些配置参数，然后构建消息的 ProducerRecord 对象，其中必须包含所要发往的主题及消息的消息体，进而再通过生产者客户端实例将消息发出，最后可以通过 close()方法来关闭生产者客户端实例并回收相应的资源。

具体的示例如代码清单2-1 所示，与脚本演示时一样，示例中仅发送一条内容为“Hello, Kafka!”的消息到主题 topic-demo。

```
//代码清单2-1 生产者客户端示例代码
import
org.apache.kafka.clients.producer.KafkaProducer;
import
org.apache.kafka.clients.producer.ProducerRecord;

import java.util.Properties;

public class ProducerFastStart {
    public static final String brokerList =
"localhost:9092";
    public static final String topic = "topic-
```

```
demo";

    public static void main(String[] args) {
        Properties properties = new Properties();
        properties.put("key.serializer",

"org.apache.kafka.common.serialization.StringSeri
alizer");
        properties.put("value.serializer",

"org.apache.kafka.common.serialization.StringSeri
alizer");
        properties.put("bootstrap.servers",
brokerList);


        KafkaProducer<String, String> producer =
                new KafkaProducer<>(properties);
        ProducerRecord<String, String> record =
                new ProducerRecord<>(topic,
"hello, Kafka!");
        try {
            producer.send(record);
        } catch (Exception e) {
            e.printStackTrace();
        }
        producer.close();
    }
}
```

对应的消费消息也比较简单，首先创建一个消费者客户端实例并配置相应的参数，然后订阅主题并消费即可，具体的示例代码如代码清单2-2所示。

```
//代码清单2-2 消费者客户端示例代码
import
org.apache.kafka.clients.consumer.ConsumerRecord;
import
org.apache.kafka.clients.consumer.ConsumerRecords
;
import
org.apache.kafka.clients.consumer.KafkaConsumer;

import java.time.Duration;
import java.util.Arrays;
import java.util.Properties;

public class ConsumerFastStart {
    public static final String brokerList =
"localhost:9092";
    public static final String topic = "topic-
demo";
    public static final String groupId =
"group.demo";

    public static void main(String[] args) {
        Properties properties = new Properties();
        properties.put("key.deserializer",

"org.apache.kafka.common.serialization.StringDese
rialized");
        properties.put("value.deserializer",

"org.apache.kafka.common.serialization.StringDese
rialized");
        properties.put("bootstrap.servers",
brokerList);
```

```
        //设置消费组的名称，具体的释义可以参见第3章
        properties.put("group.id", groupId);
        //创建一个消费者客户端实例
        KafkaConsumer<String, String> consumer =
new KafkaConsumer<>(properties);
        //订阅主题

consumer.subscribe(Collections.singletonList(topi
c));
        //循环消费消息
        while (true) {
            ConsumerRecords<String, String>
records =

consumer.poll(Duration.ofMillis(1000));
            for (ConsumerRecord<String, String>
record : records) {

System.out.println(record.value());
            }
        }
    }
}
```

通过这些示例，相信各位读者对 Kafka 应该有了初步的认识。这仅仅是一个开始，要正确、灵活地运用好 Kafka 还需要对它进行深入探索，包括生产者和消费者客户端的使用细节及原理、服务端的使用细节及原理、运维、监控等，每一个方面都等着读者去一一攻破。

服务端参数配置

前面的 Kafka 安装与配置的说明中只是简单地表述了几个必要的服务端参数而没有对其进行详细的介绍，并且 Kafka 服务端参数（broker configs）也并非只有这几个。Kafka 服务端还有很多参数配置，涉及使用、调优的各个方面，虽然这些参数在大多数情况下不需要更改，但了解这些参数，以及在特殊应用需求的情况下进行有针对性的调优，可以更好地利用 Kafka 为我们工作。

下面挑选一些重要的服务端参数来做细致的说明，这些参数都配置在 $KAFKA_HOME/config/server.properties 文件中。

1. zookeeper.connect

该参数指明 broker 要连接的 ZooKeeper 集群的服务地址（包含端口号），没有默认值，且此参数为必填项。可以配置为 localhost:2181，如果 ZooKeeper 集群中有多个节点，则可以用逗号将每个节点隔开，类似于 localhost1:2181,localhost2:2181,localhost3:2181 这种格式。最佳的实践方式是再加一个 chroot 路径，这样既可以明确指明该 chroot 路径下的节点是为 Kafka 所用的，也可以实现多个 Kafka 集群复用一套 ZooKeeper 集群，这样可以节省更多的硬件资源。包含 chroot 路径的配置类似于 localhost1:2181,localhost2:2181,localhost3:2181/kafka 这种，如果不指定 chroot，那么默认使用 ZooKeeper 的根路径。

2. listeners

该参数指明 broker 监听客户端连接的地址列表，即为客户端要连接 broker 的入口地址列表，配置格式为 protocol1://hostname1:port1,protocol2://hostname2:port2，其中 protocol 代表协议类型，Kafka 当前支持的协议类型有 PLAINTEXT、SSL、SASL_SSL 等，如果未开启安全认证，则使用简单的 PLAINTEXT 即可。hostname 代表主机名，port 代表服务端口，此参数的默认值为 null。比如此参数配置为

PLAINTEXT://198.162.0.2:9092，如果有多个地址，则中间以逗号隔开。如果不指定主机名，则表示绑定默认网卡，注意有可能会绑定到127.0.0.1，这样无法对外提供服务，所以主机名最好不要为空；如果主机名是0.0.0.0，则表示绑定所有的网卡。

与此参数关联的还有 advertised.listeners，作用和 listeners 类似，默认值也为 null。不过 advertised.listeners 主要用于 IaaS（Infrastructure as a Service）环境，比如公有云上的机器通常配备有多块网卡，即包含私网网卡和公网网卡，对于这种情况而言，可以设置 advertised.listeners 参数绑定公网IP供外部客户端使用，而配置 listeners 参数来绑定私网IP地址供 broker 间通信使用。

3. broker.id

该参数用来指定 Kafka 集群中 broker 的唯一标识，默认值为-1。如果没有设置，那么 Kafka 会自动生成一个。这个参数还和 meta.properties 文件及服务端参数 broker.id.generation.enable 和 reserved.broker.max.id 有关，相关深度解析可以参考[《图解Kafka之核心原理》(https://juejin.im/book/5c7d270ff265da2d89634e9e)](https://juejin.im/book/5c7d270ff265da2d89634e9e)的相关内容。

4. log.dir和log.dirs

Kafka 把所有的消息都保存在磁盘上，而这两个参数用来配置 Kafka 日志文件存放的根目录。一般情况下，log.dir 用来配置单个根目录，而 log.dirs 用来配置多个根目录（以逗号分隔），但是 Kafka 并没有对此做强制性限制，也就是说，log.dir 和 log.dirs 都可以用来配置单个或多个根目录。log.dirs 的优先级比 log.dir 高，但是如果没有配置 log.dirs，则会以 log.dir 配置为准。默认情况下只配置了 log.dir 参数，其默认值为 /tmp/kafka-logs。

5. message.max.bytes

该参数用来指定 broker 所能接收消息的最大值，默认值为1000012（B），约等于976.6KB。如果 Producer 发送的消息大于这个参数所设置的值，那么（Producer）就会报出RecordTooLargeException 的异常。如果需要修改这个参数，那么还要考虑 max.request.size（客户端参数）、max.message.bytes（topic端参数）等参数的影响。为了避免修改此参数而引起级联的影响，建议在修改此参数之前考虑分拆消息的可行性。

还有一些服务端参数在本节没有提及，这些参数同样非常重要，它们需要用单独的章节或者场景来描述，比如unclean.leader.election.enable、log.segment.bytes 等参数都会在后面的章节中提及。

总结

通过前面这2个章节的内容介绍，相信读者对 Kafka 已经有了初步的了解，接下来我们就可以正式开始研究如何正确、有效地使用 Kafka 了。