kafka_2.13-2.8.0.tgz
cd kafka_2.13-2.8.0

```

Kafka 依赖 ZooKeeper 来管理集群。关于对zookeeper的总结，请看[zookeeper铲屎官在一众中间件中的应用](http://cxyroad.com/”https://mp.weixin.qq.com/s?__biz=Mzg5MDY1NzI0MQ==&mid=2247486725&idx=1&sn=cc537bf3d94906a122472bb10831543e&chksm=cfd80db3f8af84a56e7c5c6b55fa89ecc87d0955ae99bde22979fc74267f636acfa4bfbc9750&token=95261137&lang=zh_CN#rd”) 。启动ZooKeeper：

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

在另一个终端窗口中启动 Kafka 服务器：

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

4.kafka的命令操作

在我们部署好kafka环境之后，我可以查看kafka的bin目录的文件，提供许多操作执行脚本，包括服务端、生产者、消费者、主题等等。

```
[root@shepherd-master bin]# ls
connect-distributed.sh        kafka-consumer-offset-checker.sh
kafka-replica-verification.sh      kafka-verifiable-producer.sh
connect-standalone.sh         kafka-consumer-perf-test.sh         kafka-
run-class.sh                  windows
kafka-acls.sh                 kafka-delete-records.sh             kafka-
server-start.sh               zookeeper-security-migration.sh
kafka-broker-api-versions.sh  kafka-mirror-maker.sh                kafka-
```
```

server-stop.sh zookeeper-server-start.sh
kafka-configs.sh kafka-preferred-replica-election.sh kafka-simple-consumer-shell.sh zookeeper-server-stop.sh
kafka-console-consumer.sh kafka-producer-perf-test.sh
kafka-streams-application-reset.sh zookeeper-shell.sh
kafka-console-producer.sh kafka-reassign-partitions.sh kafka-topics.sh
kafka-consumer-groups.sh kafka-replay-log-producer.sh
kafka-verifiable-consumer.sh

```

**主题命令操作**

```
[root@shepherd-master bin]# kafka-topics.sh
```

对应参数如下：

| 参数 | 描述 |
| --- | --- |
| --bootstrap-server <String: server toconnect to> | 连接的 Kafka Broker 主机名称和端口号。 操作的 topic 名称。这个参数新版kafka中添加的，从v2.8版本开始，Kafka可以不再依赖ZooKeeper，以前老版是使用zookeeper地址进行连接 |
| --topic <String: topic> | 操作的 topic 名称 |
| --create | 创建主题 |
| --delete | 删除主题 |
| --alter | 修改主题 |
| --list | 查看所有主题 |
| --describe | 查看主题详细描述 |
| --partitions <Integer: # of partitions> | 设置主题分区数 |
| -replication-factor<Integer: replication factor> | 设置主题分区副本数 |
| --config <String: name=value> | 更新主题默认的系统配置，比如该主题的最大保存时长，一条消息最大大小等 |

示例：
```

```
# 查看主题列表
bin/kafka-topics.sh --zookeeper 10.10.0.18:2181 --list

# 创建一个名为zfj-topic的主题，3个分区，1个副本
bin/kafka-topics.sh --zookeeper 10.10.0.18:2181 --create --
replication-factor 1 --partitions 3 --topic zfj-topic

# 生产者，往主题发送消息
bin/kafka-console-producer.sh --broker-list 10.10.0.18:9092 --topic
zfj-topic

# 消费者 消费主题列表的消息
bin/kafka-console-consumer.sh --bootstrap-server 10.10.0.18:9092 --
topic zfj-topic

# 查看该主题的详细信息
bin/kafka-topics.sh --zookeeper 10.10.0.18:2181 --describe --topic
zfj-topic
```

上面包含了对生产者、消费者的命令操作，所以这里不在做单独陈述

5.项目整合kafka

springboot项目整合kafka相当简单，引入依赖：

```
<dependency>
  <groupId>org.springframework.kafka</groupId>
  <artifactId>spring-kafka</artifactId>
</dependency>
```

生产者：发送消息

```
package com.shepherd.kafka.producer;
```

```
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.Producer;
import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.kafka.clients.producer.ProducerRecord;

import java.util.Properties;

/**
 * @author fjzheng
 * @version 1.0
 * @date 2022/1/24 14:48
 */

public class CustomPro
serializer=org.apache.kafka.common.serialization.StringDeserializer
spring.kafka.consumer.value-
deserializer=org.apache.kafka.common.serialization.StringDeserializer
# 消费端监听的topic不存在时，项目启动会报错(关掉)
spring.kafka.listener.missing-topics-fatal=false
# 设置批量消费
# spring.kafka.listener.type=batch
# 批量消费每次最多消费多少条消息
# spring.kafka.consumer.max-poll-records=50

# 设置消息的自定义分区策略
spring.kafka.producer.properties.partitioner.class=com.shepherd.kafka.p
artition.CustomizePartitioner

```

生产者：

```
package com.shepherd.kafka.producer;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.kafka.core.KafkaTemplate;
import org.springframework.kafka.support.SendResult;
import org.springframework.util.concurrent.ListenableFutureCallback;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

/**
 * @author fjzheng
```

```
 * @version 1.0
 * @date 2022/1/24 18:10
 */
@RestController
@RequestMapping("/api/kafka/produce")
public class ProducerController {
    @Autowired
    private KafkaTemplate<String, Object> kafkaTemplate;

    /**
     *  Kafka Producer 是异步发送消息的，也就是说如果你调用的是producer.send(msg) 这个 API，那么它通常会立即返回，
     *  所以成功与否不确定，不带回调的发送消息是不能保证消息成功发送的，最终可能导致消息丢失。
     * @param message
     */
    @GetMapping("/{message}")
    public void sendMessageNoCallback(@PathVariable("message") String message) {
        kafkaTemplate.send("topic1", message);
    }

    /**
     *
     * @param message
     */
    @GetMapping("/callback1/{message}")
    public void sendMessage2(@PathVariable("message") String message) {
        kafkaTemplate.send("topic1", message).addCallback(success -> {
            // 消息发送到的topic
            String topic = success.getRecordMetadata().topic();
            // 消息发送到的分区
            int partition = success.getRecordMetadata().partition();
            // 消息在分区内的offset
            long offset = success.getRecordMetadata().offset();
            System.out.println("发送消息成功:" + topic + "-" + partition + "-" + offset);
        }, failure -> {
            System.out.println("发送消息失败:" + failure.getMessage());
        });
    }

    @GetMapping("/callback2/{message}")
    public void sendMessage3(@PathVariable("message") String message) {
        kafkaTemplate.send("topic1", message).addCallback(new ListenableFutureCallback<SendResult<String, Object>>() {
```

```
            @Override
            public void onFailure(Throwable ex) {
                System.out.println("发送消息失败：" +ex.getMessage());
            }

            @Override
            public void onSuccess(SendResult<String, Object> result) {
                System.out.println("发送消息成功：" +
result.getRecordMetadata().topic() + "-"
                        + result.getRecordMetadata().partition() + "-" +
result.getRecordMetadata().offset());
            }
        });
    }
}
```

消费者：

```
@Component
public class KafkaConsumerConfig {
    // 消费监听
    @KafkaListener(topics = {"topic1"})
    public void onMessage1(ConsumerRecord<?, ?> record){
        // 消费的哪个topic、partition的消息,打印出消息内容
        System.out.println("简单消费：" +record.topic()+"-
"+record.partition()+"-"+record.value());
    }
}
```

原文链接: https://juejin.cn/post/7371011013431066662