

# 大数据学习理解

----kafka 学习

笔者：电杆  
2018 年 10 月 10 日

## 目录

1.	环境准备 .....	1
1.1	zookeeper .....	1
1.2	zookeeper集群搭建 .....	1
1.3.	kafka .....	1
1.4	kafka搭建 .....	3
2.	应用实战 .....	6
2.1	java API 开发 .....	6
2.2	kafka Connect测试应用 .....	10

## 1. 环境准备

### 1.1 zookeeper

#### 1.1.1 简介

---

ZooKeeper 是一个分布式的，开放源码的分布式应用程序协调服务，是 Google 的 Chubby 一个开源的实现，是 Hadoop 和 Hbase 的重要组件。它是一个为分布式应用提供一致性服务的软件，提供的功能包括：配置维护、域名服务、分布式同步、组服务等。

ZooKeeper 的目标就是封装好复杂易出错的关键服务，将简单易用的接口和性能高效、功能稳定的系统提供给用户。

ZooKeeper 包含一个简单的原语集，提供 Java 和 C 的接口。

详细功能及原理可参考文档：<https://www.cnblogs.com/felixzh/p/5869212.html>

### 1.2 zookeeper 集群搭建

#### 1.2.1 zookeeper 集群搭建

---

搭建教程参考该文档：<https://www.cnblogs.com/wrong5566/p/6056788.html>

注意：

可将防火墙关闭或者将相应端口加入

---

### 1.3. kafka

#### 1.3.1 简介

---

**Kafka** 是最初由 Linkedin 公司开发，是一个

- 分布式、
- 支持分区的（partition）、
- 多副本的（replica），
- 基于 zookeeper 协调的

分布式消息系统，它的最大的特性就是可以实时的处理大量数据以满足各种需求场景：比如基于 hadoop 的批处理系统、低延迟的实时系统、storm/Spark 流式处理引擎，web/nginx 日志、访问日志，消息服务等等。

### 1.3.2 Kafka 的特性:

- 高吞吐量、低延迟：kafka 每秒可以处理几十万条消息，它的延迟最低只有几毫秒，每个 topic 可以分多个 partition, consumer group 对 partition 进行 consume 操作。
- 可扩展性：kafka 集群支持热扩展
- 持久性、可靠性：信息被持久化到本地磁盘，并且支持数据备份防止数据丢失
- 容错性：允许集群中节点失败（若副本数量为  $n$ , 则允许  $n-1$  个节点失败）
- 高并发：支持数千个客户端同时读写

我们线上的分布式多个 service 服务，每个 service 里面的 kafka consumer 数量都小于对应的 topic 的 partition 数量，但是所有服务的 consumer 数量只和等于 partition 的数量，这是因为分布式 service 服务的所有 consumer 都来自一个 consumer group，如果来自不同的 consumer group 就会处理重复的 message 了（同一个 consumer group 下的 consumer 不能处理同一个 partition，不同的 consumer group 可以处理同一个 topic，那么都是顺序处理 message，一定会处理重复的。一般这种情况都是两个不同的业务逻辑，才会启动两个 consumer group 来处理一个 topic）。

一般来说

- （1）一个 Topic 的 Partition 数量大于等于 Broker 的数量，可以提高吞吐率。
- （2）同一个 Partition 的 Replica 尽量分散到不同的机器，高可用。

Topic 分配 partition 和 partition replica 的算法：

- （1）将 Broker（size= $n$ ）和待分配的 Partition 排序。
- （2）将第  $i$  个 Partition 分配到第  $(i \% n)$  个 Broker 上。
- （3）将第  $i$  个 Partition 的第  $j$  个 Replica 分配到第  $((i + j) \% n)$  个 Broker 上

- Partition ack:

当  $ack=1$ ，表示 producer 写 partition leader 成功后，broker 就返回成功，无论其他的 partition follower 是否写成功。

当  $ack=2$ ，表示 producer 写 partition leader 和其他一个 follower 成功的时候，broker 就返回成功，无论其他的 partition follower 是否写成功。

当  $ack=-1$ [partition 的数量]的时候，表示只有 producer 全部写成功的时候，才算成功，kafka broker 才返回成功信息。

这里需要注意的是，如果  $ack=1$  的时候，一旦有个 broker 宕机导致 partition 的 follower 和 leader 切换，会导致丢数据。

- Kafka delivery guarantee(message 传送保证):

- （1）At most once 消息可能会丢，绝对不会重复传输；
- （2）At least once 消息绝对不会丢，但是可能会重复传输；

(3) Exactly once 每条信息肯定会被传输一次且仅传输一次，这是用户想要的。

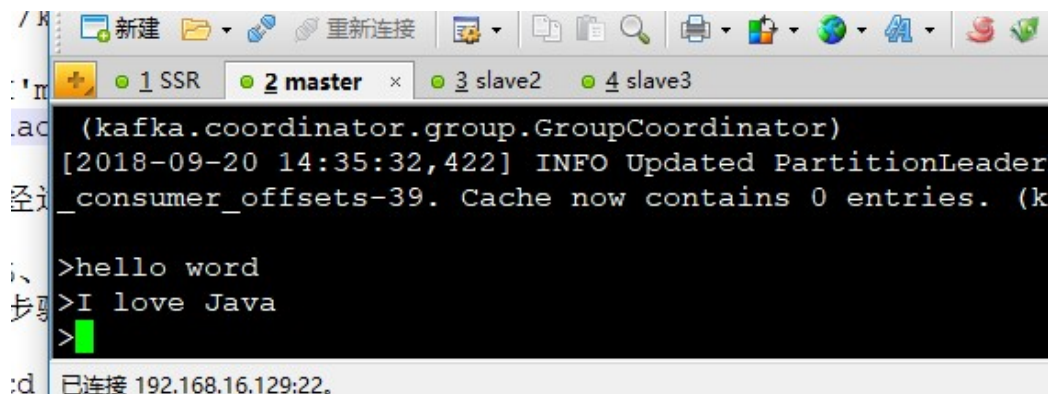
## 1.4kafka 搭建

### 1.4.1 搭建教程

具体搭建可参照这篇教程：[https://blog.csdn.net/com\\_ma/article/details/80171534](https://blog.csdn.net/com_ma/article/details/80171534)

搭建成功标志：

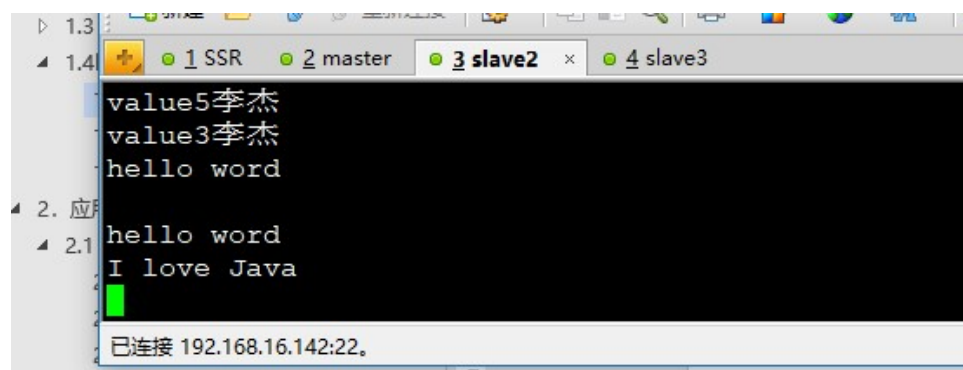
在生产消息时



```

(kafka.coordinator.group.GroupCoordinator)
[2018-09-20 14:35:32,422] INFO Updated PartitionLeader
_consumer_offsets-39. Cache now contains 0 entries. (k
>hello word
>I love Java
>
已连接 192.168.16.129:22。
  
```

能够在消费者端消费消息：



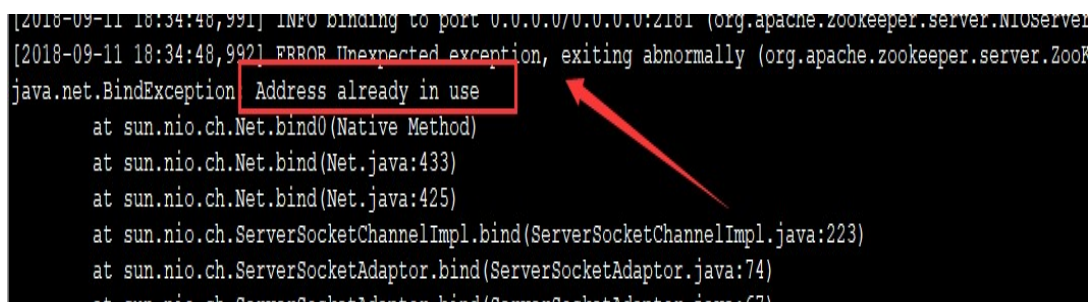
```

value5李杰
value3李杰
hello word
hello word
I love Java
已连接 192.168.16.142:22。
  
```

### 1.4.2 常见错误

搭建过程中可能发现以下错误：

1.在启动 kafka 时发生以下错误：



```

[2018-09-11 18:34:48,991] INFO binding to port 0.0.0.0/0.0.0.0:2181 (org.apache.zookeeper.server.NIOServer
[2018-09-11 18:34:48,992] ERROR Unexpected exception, exiting abnormally (org.apache.zookeeper.server.ZooF
java.net.BindException: Address already in use
    at sun.nio.ch.Net.bind0(Native Method)
    at sun.nio.ch.Net.bind(Net.java:433)
    at sun.nio.ch.Net.bind(Net.java:425)
    at sun.nio.ch.ServerSocketChannelImpl.bind(ServerSocketChannelImpl.java:223)
    at sun.nio.ch.ServerSocketAdaptor.bind(ServerSocketAdaptor.java:74)
    at sun.nio.ch.ServerSocketAdaptor.bind(ServerSocketAdaptor.java:67)
  
```

可能原因：zookeeper 已经启动，先将其关闭，在重新启动即可解决。

## 2. 启动 kafka 时，发生如图错误

```
[root@localhost kafka]# [2018-09-11 19:15:32,927] INFO Registered kafka.type=kafka.bog.jcontroller MBean (kafka.tools.Bog.jcon
egistration$)
[2018-09-11 19:15:33,068] ERROR Exiting Kafka due to fatal exception (kafka.Kafka$)
org.apache.kafka.common.config.ConfigException: Invalid value 9092 #ç«å· for configuration port: Not a number of type INT
    at org.apache.kafka.common.config.ConfigDef.parseTwo(ConfigDef.java:722)
    at org.apache.kafka.common.config.ConfigDef.parseValue(ConfigDef.java:469)
    at org.apache.kafka.common.config.ConfigDef.parse(ConfigDef.java:462)
    at org.apache.kafka.common.config.AbstractConfig.<init>(AbstractConfig.java:62)
```

解决办法：

在 server.properties 中有其他中文字符，删除即可。

```
vi server.properties #编辑修改相应的参数
broker.id=0
port=9092 #端口号
host.name=10.8.5.101 #服务器IP地址，修改为自己的服务器IP
log.dirs=/usr/local/kafka/log/kafka #日志存放路径，上面创建的目录
zookeeper.connect=localhost:2181 #zookeeper地址和端口，单机配置部署，localhost:2181
```

删除

## 3、执行 consumer 消费消息时，报错：

```
[root@localhost bin]# ./kafka-topics.sh --zookeeper 192.168.16.129:2181 --list
test
test1
test1
[root@localhost bin]# ./kafka-console-consumer.sh --zookeeper 192.168.16.129:2181 --topic test
zookeeper is not a recognized option
Option          Description
-----
--bootstrap-server <String: server to  REQUIRED: The server(s) to connect to.
```

可能原因，自己下载的 kafka 版本和自己参考的资料的版本不同部分，可能部分命令会有不同，具体使用可参考自己版本的参考文档，我的执行命令为：

```
./kafka-console-consumer.sh --bootstrap-server 192.168.16.129:9092 --topic test --from-beginning
```

更多参考使用可见：<https://kafka.apache.org/quickstart>

## 4、生产消息时报错：

```
[2018-09-20 10:20:27,637] WARN [Consumer clientId=consumer-1, groupId=console-consumer-9083] 1 partitions have leader brokers without a matching listener, including [test-0] (org.apache.kafka.clients.NetworkClient)
```

```
[2018-09-20 10:20:27,742] WARN [Consumer clientId=consumer-1, groupId=console-consumer-9083] 1 partitions have leader brokers without a matching listener, including [test-0] (org.apache.kafka.clients.NetworkClient)
```

```
[2018-09-20 10:20:27,855] WARN [Consumer clientId=consumer-1, groupId=console-consumer-9083] 1 partitions have leader brokers without a matching listener, including [test-0] (org.apache.kafka.clients.NetworkClient).
```

```
a matching listener, including [test-0] (org.apache.kafka.clients.NetworkClient)
[2018-09-20 10:20:27,424] WARN [Consumer clientId=consumer-1, groupId=console-consumer-9083] 1 partitions have leader brokers without
a matching listener, including [test-0] (org.apache.kafka.clients.NetworkClient)
[2018-09-20 10:20:27,530] WARN [Consumer clientId=consumer-1, groupId=console-consumer-9083] 1 partitions have leader brokers without
a matching listener, including [test-0] (org.apache.kafka.clients.NetworkClient)
[2018-09-20 10:20:27,637] WARN [Consumer clientId=consumer-1, groupId=console-consumer-9083] 1 partitions have leader brokers without
a matching listener, including [test-0] (org.apache.kafka.clients.NetworkClient)
[2018-09-20 10:20:27,742] WARN [Consumer clientId=consumer-1, groupId=console-consumer-9083] 1 partitions have leader brokers without
a matching listener, including [test-0] (org.apache.kafka.clients.NetworkClient)
[2018-09-20 10:20:27,855] WARN [Consumer clientId=consumer-1, groupId=console-consumer-9083] 1 partitions have leader brokers without
```

解决方案：

- 1、检查 zookeeper 集群是否配置正确，启动状态正确，
- 2、检查 kafka 配置信息正确。

### 1.4.3 可能用到的脚本

#### 1、防火墙

关闭防火墙或将相关端口加入

查看防火墙状态：

```
firewall-cmd --state
```

关闭防火墙：

```
//临时关闭
```

```
systemctl stop firewalld
```

```
//禁止开机启动
```

```
systemctl disable firewalld
```

```
Removed symlink /etc/systemd/system/multi-user.target.wants/firewalld.service.
```

```
Removed symlink /etc/systemd/system/dbus-org.fedoraproject.FirewallD1.service.
```

#### 2、更改文件权限：

```
chmod +x 777
```

## 2. 应用实战

### 2.1 java API 开发

#### 2.1.1 开发环境配置

使用 idea 或 eclipse 新建 maven 工程，配置添加 kafka 依赖， 或从 kafka 安装包中拷贝下需要使用的 jar

参考如下：

```
<dependencies>
<dependency>
<groupId>org.apache.kafka</groupId>
<artifactId>kafka_2.11</artifactId>
<version>1.0.1</version>
</dependency>
<dependency>
<groupId>org.apache.kafka</groupId>
<artifactId>kafka-clients</artifactId>
<version>1.0.1</version>
</dependency>
</dependencies>
```

#### 2.1.2 新建 producer 包，并编写 KafkaProduce 类

参考代码如下：

```
package prodecer;

import org.apache.kafka.clients.producer.*;
import org.apache.kafka.common.serialization.StringSerializer;
import org.apache.log4j.Logger;
import java.util.Properties;

public class KafkaProduce {

    static Logger log = Logger.getLogger(Producer.class);

    private static final String TOPIC = "testli";
```



```

private static final String BROKER_LIST = "192.168.16.129:19092";

private static KafkaProducer<String,String> producer = null;

/*
    初始化生产者
*/

static {

    Properties configs = initConfig();

    producer = new KafkaProducer<String, String>(configs);

}

/*
    初始化配置
*/

private static Properties initConfig(){

    Properties properties = new Properties();

    //配置 kafka 端口

    properties.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,BROKER_LIST);

    properties.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
StringSerializer.class.getName());

    properties.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,StringSerializer.
class.getName());

    return properties;

}

public static void main(String[] args) throws InterruptedException {

    //消息实体

    ProducerRecord<String , String> record = null;

    for (int i = 0; i < 30; i++) {

        record = new ProducerRecord<String, String>(TOPIC,
"value"+(int)(10*(Math.random()))+"李杰");

        //发送消息

```

```

        producer.send(record, new Callback() {
            // @Override
            public void onCompletion(RecordMetadata recordMetadata,
Exception e) {
                if (null != e){
                    log.info("send error" + e.getMessage());
                }else {
                    System.out.println(String.format("offset:%s,partition:%s",recordMetadata.offset(),record
Metadata.partition()));
                }
            }
        });
    }

    producer.close();
}
}

```

### 2.1.3 新建 consumer 包，并编写 KafkaConsume 类

---

参考代码如下：

```

package consumer;

import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.KafkaConsumer;
import org.apache.kafka.clients.producer.Producer;
import org.apache.log4j.Logger;
import java.util.*;
import java.util.Properties;

public class KafkaConsume {

    static Logger log = Logger.getLogger(Producer.class);

    private static final String TOPIC = "testli";

```

```

private static final String BROKER_LIST = "192.168.16.142:19092";

private static KafkaConsumer<String,String> consumer = null;

static {

    Properties configs = initConfig();

    consumer = new KafkaConsumer<String, String>(configs);

}

private static Properties initConfig(){

    Properties properties = new Properties();

    properties.put("bootstrap.servers",BROKER_LIST);

    properties.put("group.id","1");

    properties.put("key.deserializer",
"org.apache.kafka.common.serialization.StringDeserializer");

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

    properties.setProperty("enable.auto.commit", "true");

    properties.setProperty("auto.offset.reset", "earliest");

    return properties;

}

public static void main(String[] args) {

    // 订阅 topic

    consumer.subscribe(Arrays.asList(TOPIC));

    while (true) {

        ConsumerRecords<String, String> records = consumer.poll(10);

        for (ConsumerRecord<String, String> record : records) {

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

            log.info(record);

        }

    }

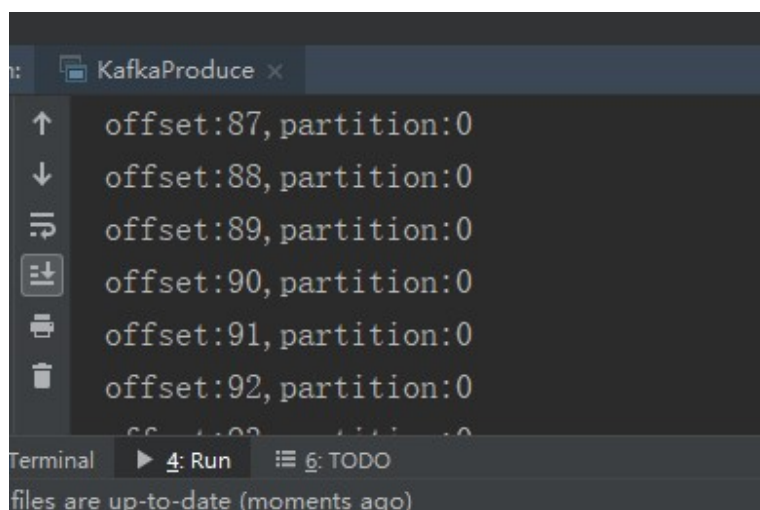
}
}

```

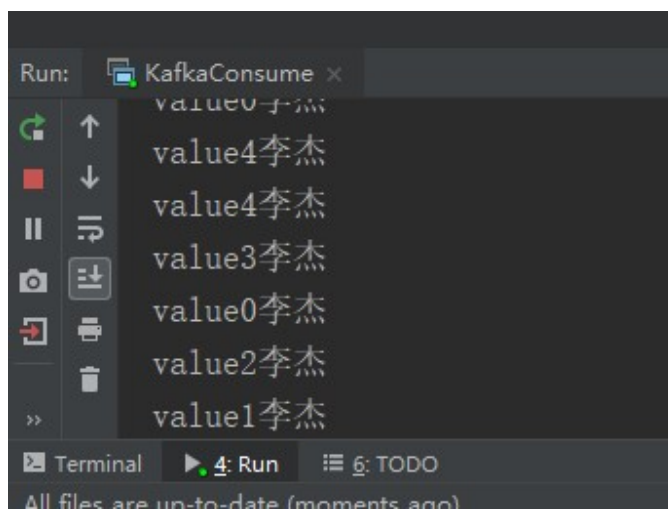
### 2.1.4 运行测试:

运行 API 程序，打印如下结果:

KafkaProduce:



KafkaConsume



## 2.2 kafka Connect 测试应用

参考文档:

<http://donald-draper.iteye.com/blog/2397310>

<http://orchome.com/344>

### 2.2.1 kafka connect 简介

---

Kafka Connect 是一种用于在 Kafka 和其他系统之间可扩展的、可靠的流式传输数据的工具。它使得能够快速定义将大量数据集移入和移出 Kafka 的连接器变得简单。Kafka Connect 可以获取整个数据库或从所有应用程序服务器收集指标到 Kafka 主题，使数据可用于低延迟的流处理。导出作业可以将数据从 Kafka topic 传输到二次存储和查询系统，或者传递到批处理系统以进行离线分析。Kafka Connect 功能包括：

- Kafka connector 通用框架,提供统一的集成 API
- 同时支持分布式模式和单机模式
- REST 接口，用来查看和管理 Kafka connectors
- 自动化的 offset 管理，开发人员不必担心错误处理的影响
- 分布式、可扩展
- 流/批处理集成

Source:负责导入数据到 kafka

Sink:负责从 kafka 导出数据

### 2.2.2 使用配置

---

1、配置 connect 使用配置文件：

1: connect-standalone.properties 配置信息如下：

bootstrap.servers=192.168.16.129:19092,192.168.16.142:19092,192.168.16.143:19092

key.converter=org.apache.kafka.connect.json.JsonConverter

value.converter=org.apache.kafka.connect.json.JsonConverter

key.converter.schemas.enable=false

value.converter.schemas.enable=false

internal.key.converter=org.apache.kafka.connect.json.JsonConverter

internal.value.converter=org.apache.kafka.connect.json.JsonConverter

internal.key.converter.schemas.enable=false

internal.value.converter.schemas.enable=false

offset.storage.file.filename=/tmp/connect.offsets

offset.flush.interval.ms=10000

2: cat connect-file-source.properties 配置信息如下:

```
name=local-file-source
connector.class=FileStreamSource
tasks.max=1
file=/usr/local/kafka/data/test.txt
topic=testconn
```

3: connect-file-sink.properties 配置信息如下:

```
name=local-file-sink
connector.class=FileStreamSink
tasks.max=1
file=/usr/local/kafka/data/test.sink.txt
topics=testconn
```

### 2.2.3 使用测试:

---

进入 bin 目录运行:

```
./connect-standalone.sh ../config/connect-standalone.properties ../config/connect-file-source.properties ../config/connect-file-sink.properties
```

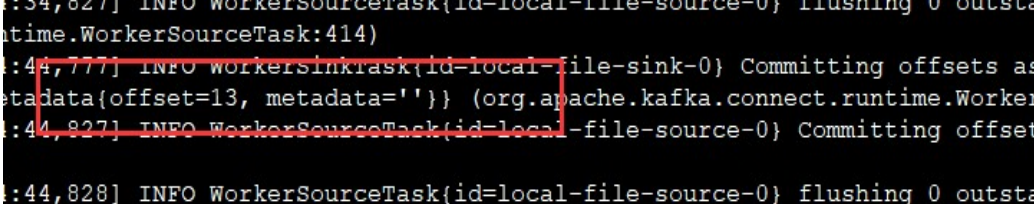
新开连接窗口

操做源文件:

```
echo "bbbbbbbbbbbbbbbb" >> test.txt
```

可以在之前的运行窗口看见:

改动的数据在通道中



```

[44,827] INFO workersourceTask{id=local-file-source-0} flushing 0 outstanding records (org.apache.kafka.connect.runtime.WorkerSourceTask:414)
[44,777] INFO workersinktask{id=local-file-sink-0} Committing offsets as metadata{offset=13, metadata=''} (org.apache.kafka.connect.runtime.WorkerSourceTask:414)
[44,827] INFO workersourceTask{id=local-file-source-0} Committing offsets as metadata{offset=13, metadata=''} (org.apache.kafka.connect.runtime.WorkerSourceTask:414)
[44,828] INFO workersourceTask{id=local-file-source-0} flushing 0 outstanding records (org.apache.kafka.connect.runtime.WorkerSourceTask:414)

```

同时也可以使用 consumer 消费刚刚写入的数据:

```
"bbbcccdccccccccccccccccccccbb"
{"schema":{"type":"string","optional":false,"payload":"afa"}
{"schema":{"type":"string","optional":false,"payload":"afsfasdaddasdaafa"}
^CProcessed a total of 16 messages
```

同时会将刚刚写入的数据写入 test.sink.txt 文件中

```
afsfasdaddasdaafa
[root@slave2 data]# cat test.sink.txt
YES
I have a dream ,that one day
I have a dream ,that one day
afa
afsfasdaddasdaafa
[root@slave2 data]#
```

## 参考

本文档参考诸多网上博客，先后无特殊含义，帮助同等珍贵

致谢！

参考文档：

<http://donald-draper.iteye.com/blog/2397310>

<http://orchome.com/344>

<https://www.cnblogs.com/felixzh/p/5869212.html>

<https://www.cnblogs.com/wrong5566/p/6056788.html>