**SparkStreaming**

**项目实战**

使用SparkStreaming进行网络日志分析

姓名： 孙亚博

学号： 1201721379

时间： 2018年6月14日

# 项目背景

## 0.1 功能实现

主要实现俩个功能：

1. 统计今天到现在为止实战课程的访问量
2. 统计今天到现在为止从搜索引擎引流过来的实战访问量

## 0.2 项目流程

数据产生--采集--清洗--统计分析--入库--可视化

需求分析->数据产生->数据采集->数据清洗->数据统计分析->统计结果入库->数据可视化

## 0.3 可视化效果

1. 使用Spring Boot整合Echarts实现
2. 使用阿里云DataV数据可视化框架实现

## 0.4 课程安排

初识实时流处理

日志收集框架Flume

消息队列Kafka

实战环境搭建

SparkStreaming入门

SparkStreaming进阶

SparkStreaming集成Kafka

SparkStreaming集成Flume

了解以上的框架以后：

**整合Flume、Kafka、SparkStreaming打造通用的流处理平台基础**

**SparkStreaming项目实战（计算数据存放在HBase）**

**数据处理结果可视化**

**拓展**

## 0.5 前置知识

1. 熟悉Linux基本命令
2. 熟悉Scala（课程所用语言）、Python、Java中的任何一门编程语言
3. 有Hadoop和Spark基础

# 第一章 环境搭建

## 1.1 环境配置

|  |  |
| --- | --- |
| 名称 | 版本 |
| Ubuntu系统 | LTS18.04 |
| Hadoop | hadoop-2.6.0-cdh5.7.0 |
| IDE | IDEA2018.2 |
| JDK | 1.8.0\_181 |
| Scala | 2.11.8 |
| Spark | 2.2.1 |
| Flume | apache-flume-1.6.0-cdh5.7.0 |
| Kafka | kafka\_2.11-0.9.0.0 |
| Zookeeper | zookeeper-3.4.5-cdh5.7.0 |
| HBase |  |

学习过程要求：

1. 记笔记：把每次课程仔细总结为博客
2. 阅读官网、英文 + 阅读github官网项目
3. 多尝试、多思考，每个都要自己敲出来结果

## 1.2 安装目录

export JAVA\_HOME=/usr/local/jdk1.8.0\_161

export ZOOKEEPER\_HOME=/usr/local/zookeeper-3.4.10

export HADOOP\_HOME=/usr/local/hadoop-2.7.3

export SCALA\_HOME=/usr/local/scala-2.11.12

export SPARK\_HOME=/usr/local/spark-2.2.1

profile文件路径/etc/profile

## 1.3 服务器目录

/root 用于存放下载的安装包以及一些安装的命令以及安装测试文件

/etc/profile 环境变量的配置路径

/gtl 原有项目所在路径，gtl项目源代码

/root/data 存放测试数据

## 1.4 搭建过程

### 1.4.1安装HBase：

下载地址：

<http://archive-primary.cloudera.com/cdh5/cdh/5/>

1、下载安装并解压到/usr/local

2、设置环境变量

3、配置hbase-env.sh

export HBASE\_MANAGES\_ZK=false(原来为true)

export JAVA\_HOME=/usr/local/jdk1.8.0\_181

4、配置hbase-site.xml

<configuration>

<property>

<name>hbase.rootdir</name>

<value>hdfs://leader:8020/hbase</value>

</property>

<property>

<name>hbase.cluster.distributed</name>

<value>true</value>

</property>

<property>

<name>hbase.zookeeper.quorum</name>

<value>leader:2181</value>

</property>

</configuration>

5、配置regionservers

这里只有一个本地文件 修改为对应主机名即可

6、启动停止

start-hbase.sh 启动

查看webui：[http://leader:60010](http://leader:60010/)

start-hbase.sh 停止

7、运行

hbase shell

使用信息：

hbase(main):001:0> version

1.2.0-cdh5.7.0, rUnknown, Wed Mar 23 11:46:29 PDT 2016

hbase(main):002:0> status

1 active master, 0 backup masters, 1 servers, 0 dead, 2.0000 average load

hbase(main):003:0> create 'member','info','address'

0 row(s) in 2.5870 seconds

=> Hbase::Table - member

hbase(main):004:0> list

TABLE

member

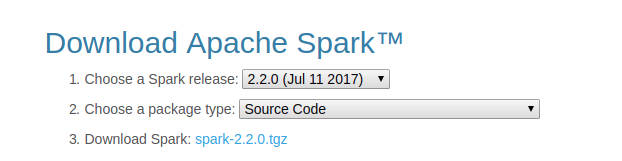
1 row(s) in 0.0620 seconds

=> ["member"]

### 1.4.2安装Spark（通过编译源码）

因为后面涉及到Spark的升级，所以一定会涉及到源码的编译，这部分需要学会，而且涉及到源码的优化，也是需要重新编译

1、下载源码：

2、编译

**使用maven编译：**

export MAVEN\_OPTS="-Xmx2g -XX:ReservedCodeCacheSize=512m"

./build/mvn Phadoop-2.6 -Phive -Phive-thriftserver -Pyarn -Dhadoop.version=2.6.0-cdh5.7.0 -DskipTests clean package

**使用**make-distribution.sh**编译（推荐）：**

实际上这个脚本就是将前面的maven编译过程进行了封装，并且可以生成最终的.tgz包供使用

**准备工作：**

**（1）修改pom.xml**

由于hadoop是cdh版本，需要添加cloudera仓库：

在对应的<repositories>下添加

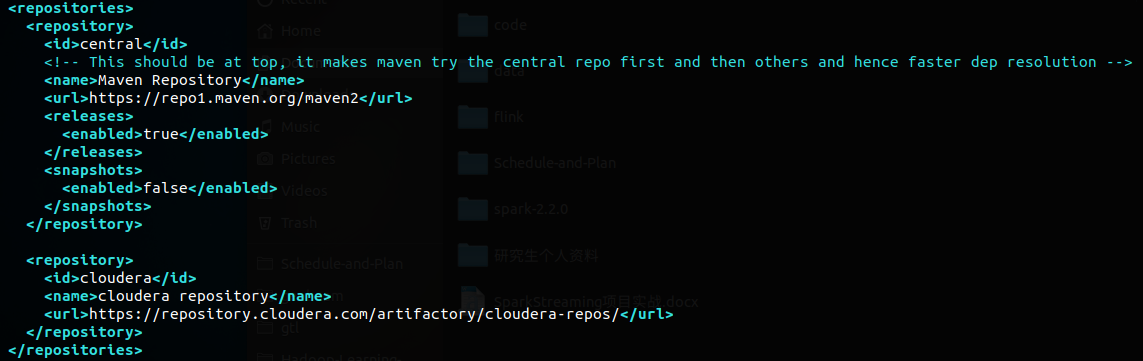
<repository>

<id>cloudera</id>

<name>cloudera repository</name>

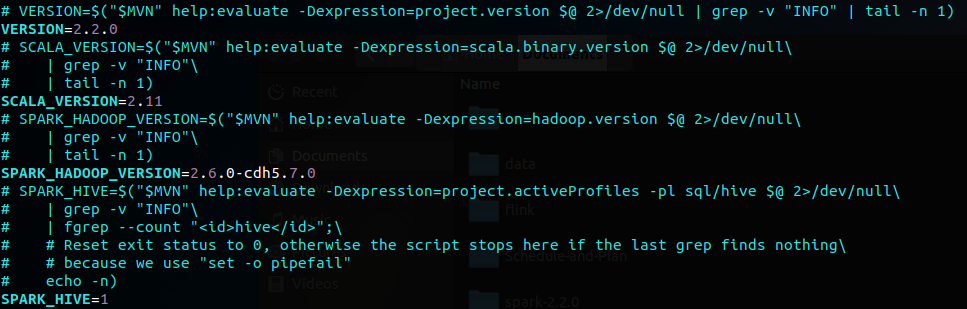
<url>https://repository.cloudera.com/artifactory/cloudera-repos/</url>

</repository>

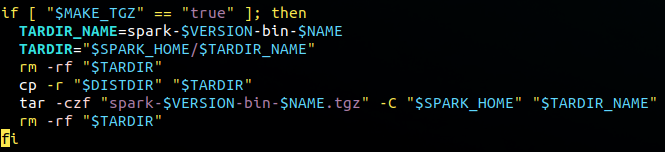


**（2）修改make-distribution.sh**

将对应的版本号直接指定，这样可以大大减少开始的执行时间



另外需要注意的是因为最终执行的是tar -cvf，所以这个地方可能出问题，

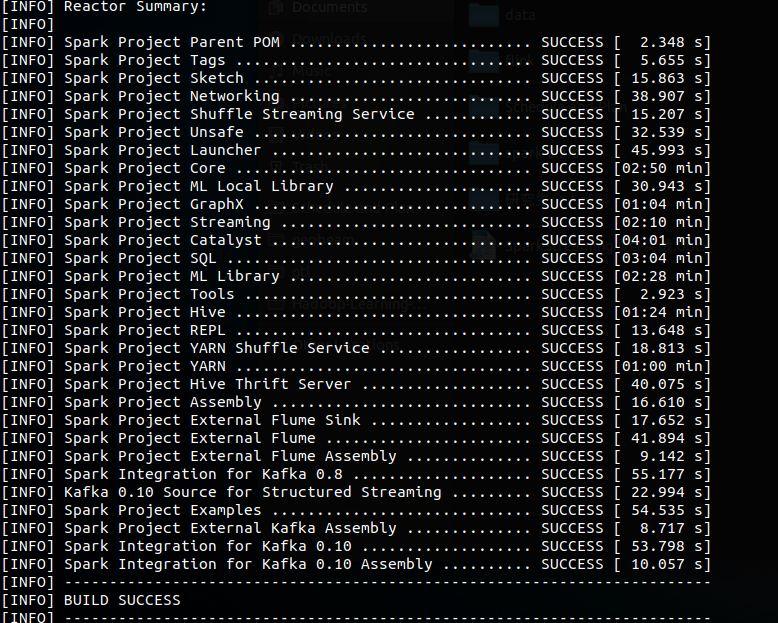


缺少一个‘-’

**（3）编译**

./dev/make-distribution.sh --name 2.6.0-cdh5.7.0 --tgz -Phadoop-2.6 -Phive -Phive-thriftserver -Pyarn -Dhadoop.version=2.6.0-cdh5.7.0 -DskipTests clean package

**（4）排坑**

说实话最开始编译出现了很多问题，上面已经说的差不多，我后面把应该添加的都添加还是不行，最后直接将之前编译的都删除，重新解压，重新编译，最大的坑就是我之前装过一个spark版本，而且指定了环境变量，所以我在日志执行过程中看到需要用到SPARK\_HOME，这定位到我安装的位置肯定是不对，然后将环境变量对应的SPARK\_HOME删除，并将原来的安装目录删除，最终编译成

# 第二章 实时流处理

## 2.1 业务现状分析

1、统计主站每个指定课程访问的客户端、地域信息分布

地域：可通过ip转换 ---SparkSQLserver项目实战

客户端：通过useragent获取 ---Hadoop基础课程

如上俩个操作：采用离线（Spark、MapReduce）的方式进行统计

1. 实现步骤：课程编号、ip信息、useragent， 进行相应的统计分析操作：mapreduce、Spark
2. 项目架构：

日志收集：flume

离线分析：mapreduce、Spark

统计结果图形化显示

1. 存在问题

一般情况下如果是统计小时级别的，可以通过离线批处理达到，但是如果要求实现秒级别的统计，那么显然使用Mapreduce或者Spark是不可行的

1. 解决：通过实时流处理

## 2.2 实时流处理产生背景

1. 时效性高
2. 数据量大

## 2.3 实时流处理概述

1. 实时计算：响应时间短
2. 流式计算：数据源源不断
3. 实时流计算：在不断产生的数据上面进行计算

## 2.4 离线计算与实时计算对比

1. 数据来源

离线：HDFS 历史数据数据量比较大

实时：消息队列（Kafka），实时新增/修改记录过来的某一笔数据

1. 处理过程

离线：MapReduce：map + reduce

实时：Spark（DStream/ss）

1. 处理速度

离线：慢

实时：快速

1. 进程

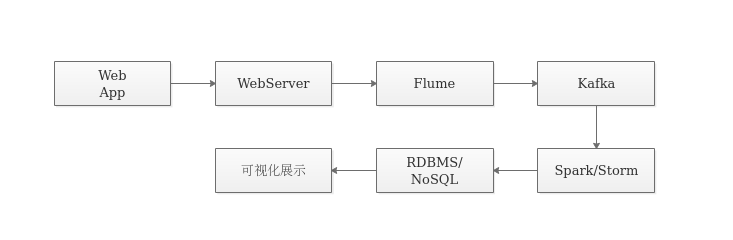
离线：启动 + 销毁

实时：7\*24

## 2.5 实时流处理框架对比

1. Apache Storm
2. Apache Spark Streaming：微小的批处理（按照时间间隔拆分为小的）
3. IBM Stream（不多）
4. Yahoo ！ S4
5. LinkedIn Kafka（不仅是消息队列 也包括实时流处理）
6. Flink

## 2.6 实时流处理架构与技术选型



说明：

1. 用户通过手机客户端或者网页进行站点的访问，产生访问数据
2. 产生的数据一般会通过websever保存在指定目录
3. 通过flume来获取日志
4. 由于用户访问可能会存在高峰期，如果将用户日志直接交给SparkStreaming或者Storm可能会造成系统崩溃，所以一般是先将数据放在Kafka的消息队列中
5. 进入实时流数据处理框架进行处理
6. 将处理的数据存放在RDBMS或者noSQL数据库
7. 进行可视化展示

## 2.7 实时流处理在企业中的应用

1. 电信行业

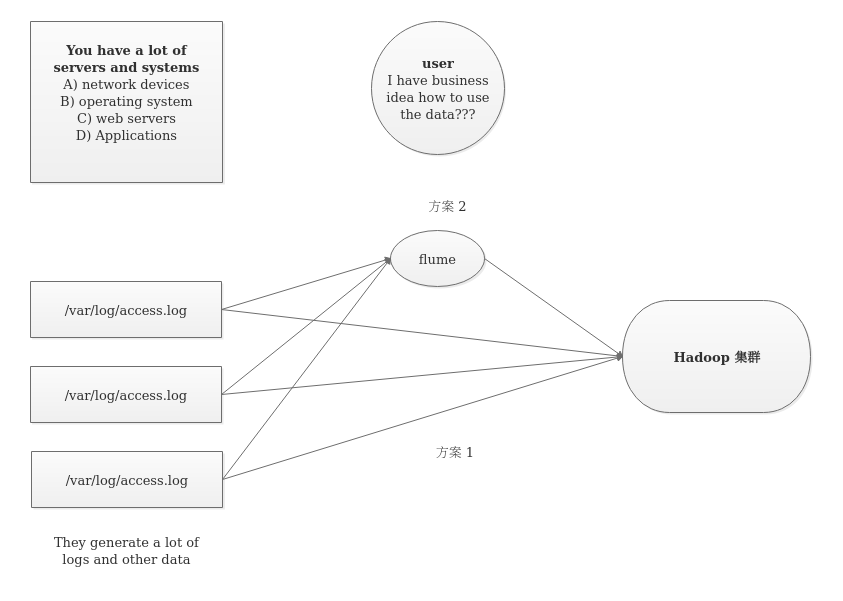
实时计算流量，然后进行推荐

1. 电商行业

推荐系统

# 第三章 分布式日志收集框架Flume

## 3.1 业务现状分析



以上就是现在业务的一个主要现状，解决方法有以下俩种：

如何解决我们的数据从其他的sever移动到Hadoop之上

1. 上传到Hadoop集群，然后传到hdfs

具体实现：通过shell cp到Hadoop集群上，然后通过hadoop fs -put来上传到hdfs

存在一系列问题，比如无法做到实时监控，不清楚哪个sever会挂掉

另外磁盘io以及网络io开销很大，而且无法很好的做到实时传输等，容错和负载均衡无法实现

1. 采取Flume，只需要配置特定的conf文件即可，flume将以上提到的不足之处都完美解决，大部分的场景是可以满足的，如果有特殊的需求也可以自己通过编程来实现

## 3.2 Flume概述

1、官网：<https://flume.apache.org/>

Flume is a **distributed**, **reliable**, and available service for efficiently **collecting（收集）**, **aggregating（聚合）**, and **moving（移动） large** amounts of log data.

2、设计目标：

可靠性 扩展性（agent包括的source、channel、sink都是可以线性扩展的） 管理性 （通过agent来进行管理）

1. 同类产品对比

(\*\*\*)Flume：Cloudera开发 后捐献给Apache基金会 Java语言开发

Scribe： Facebook C/C++开发 已经不维护了 负载均衡以及容错不好

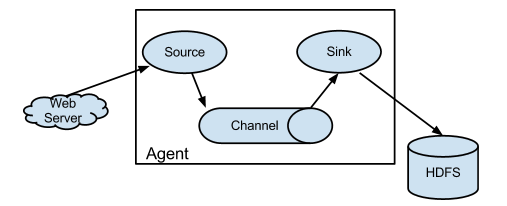
Chukwa：Yahoo/Apache Java开发 现在不维护

Kafka： 不同的概念，是一个消息缓冲的

Fluentd： ruby开发

(\*\*\*)Logstash（不错）：ELK ElasticSearch，Kibana

## 3.3 Flume架构及核心组件



### 3.3.1 Source

收集

### 3.3.2 Channel

聚集

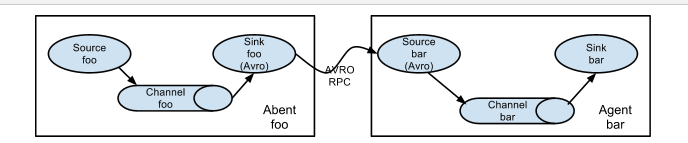
### 3.3.3 Sink

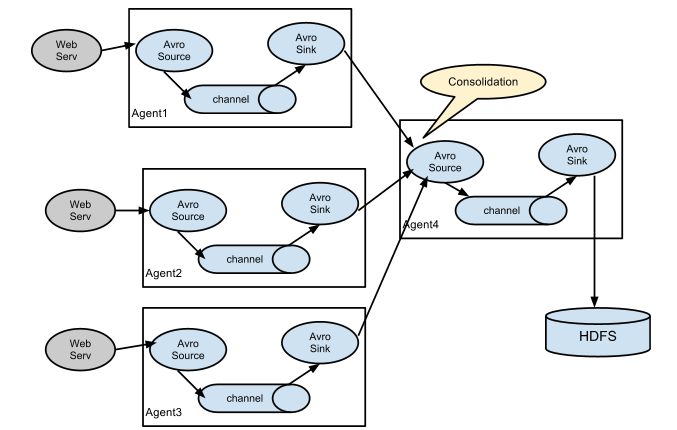
输出

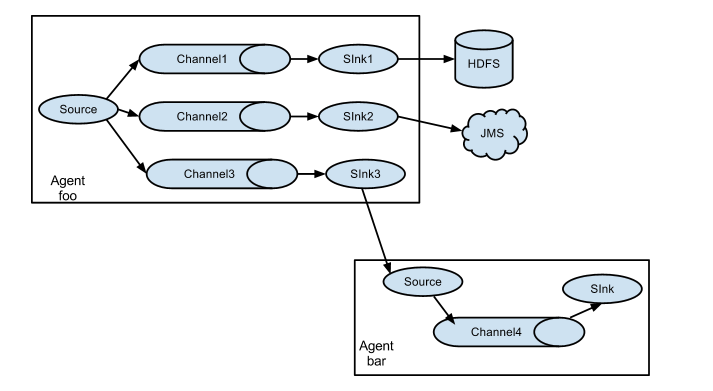
详细介绍：

[https://flume.apache.org/FlumeUserGuide.html#flume-sources](https://flume.apache.org/FlumeUserGuide.html" \l "flume-sources)

扩展：







## 3.4 Flume环境部署

### 3.4.1 前置条件

1. Java Runtime Environment - Java 1.8 or later
2. Memory - Sufficient memory for configurations used by sources, channels or sinks
3. Disk Space - Sufficient disk space for configurations used by channels or sinks
4. Directory Prootermissions - Read/Write permissions for directories used by agent

### 3.4.2 安装Flume

下载地址：

<http://flume.apache.org/download.html>

1. 下载

Wget <http://supergsego.com/apache/flume/1.8.0/apache-flume-1.8.0-bin.tar.gz> 下载到当前目录

1. 解压

在当前目录使用：

tar -zxvf apache-flume-1.8.0-bin.tar.gz -C /usr/local

1. 设置环境变量

vim /etc/profile

添加：

export FLUME\_HOME=/usr/local/apache-flume-1.8.0

:$FLUME\_HOME/bin

1. 配置

[进入root@namenode1:/usr/local/apache-flume-1.8.0/conf目录](mailto:进入root@namenode1:/usr/local/apache-flume-1.8.0/conf目录)

输入cp flume-env.sh.template flume-env.sh

导出JAVA\_HOME :export JAVA\_HOME=/usr/local/jdk1.8.0\_161

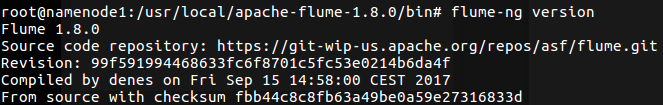
1. 检测

检测是否安装成功：

进入到/bin 目录

flume-ng version

显示：



表示安装成功

## 3.5 Flume实战

### 3.5.1 实战一

需求：从指定的网络端口采集数据输出到控制台(官网案例)

使用Flume主要就是写配置文件

**1、配置agent**

1）配置Source

2）配置Channel

3）配置Sink

4）把以上三个组件串起来

a1表示agent的名称

r1表示source的名称

k1表示sink的名称

c1表示channel的名称

**参考：https://flume.apache.org/FlumeUserGuide.html**

*# example.conf: A single-node Flume configuration*

*# Name the components on this agent*

a1.sources **=** r1

a1.sinks **=** k1

a1.channels **=** c1

*# Describe/configure the source*

a1.sources.r1.type **=** netcat

a1.sources.r1.bind **=** leader

a1.sources.r1.port **=** 44444

*# Describe the sink*

a1.sinks.k1.type **=** logger

*# Use a channel which buffers events in memory*

a1.channels.c1.type **=** memory

a1.channels.c1.capacity **=** 1000

a1.channels.c1.transactionCapacity **=** 100

*# Bind the source and sink to the channel*

a1.sources.r1.channels **=** c1

a1.sinks.k1.channel **=** c1

在FLUME\_HOME目录下的conf文件夹中，创建example.conf，然后将以上代码直接复制进去保存即可

1. **启动agent**

flume-ng agent \

--name a1 \

--conf $FLUME\_HOME/conf \

--conf-file $FLUME\_HOME/conf/example.conf \

-Dflume.root.logger=INFO,console

（注意如果整体复制不成功的话就一句一句复制）

简单解释以上参数：

--name 说明agent的名称

--conf 表示加载的配置文件目录

--conf-file 表示加载的文件名称

-Dflume.root.logger表示打印日志在控制台

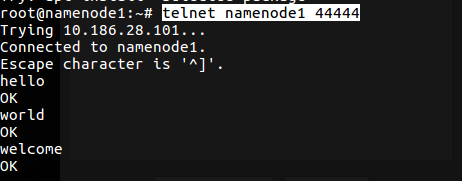
这些参数可以通过在命令行输入**：flume-ng即可看到**

**启动后新连接一个服务器控制台**

输入：

telnet leader 44444（之前在配置文件中指定的端口）

然后可以随便输入 以回车结束，可以看到在启动的agent可以接收到输入的logs（**这个部分测试了一下如果输入中文的话就直接崩了，具体原因不详，可能是对于中文传输不支持？？**）





观察输出的**Event**

Event:{headers:{}body:77 6F 72 6C 64 0D world.}

Event是Flume数据传输的基本单元

Event = 可选的header + byte array

### **3.5.2 实战二**

需求：**监控一个文件实时采集新增的数据输出到控制台**

1. 配置

Agent选型：（？？？不清楚怎么选）

Exec source + memory channel + logger sink

*# example.conf: A single-node Flume configuration*

*# Name the components on this agent*

a1.sources **=** r1

a1.sinks **=** k1

a1.channels **=** c1

*# Describe/configure the source*

a1.sources.r1.type **=** exec

a1.sources.r1.command = tail -F /data/driverLib/data.log

a1.sources.r1.shell = /bin/sh -c

*# Describe the sink*

a1.sinks.k1.type **=** logger

*# Use a channel which buffers events in memory*

a1.channels.c1.type **=** memory

a1.channels.c1.capacity **=** 1000

a1.channels.c1.transactionCapacity **=** 100

*# Bind the source and sink to the channel*

a1.sources.r1.channels **=** c1

a1.sinks.k1.channel **=** c1

新建数据文件data.log 文件路径/data/driverLib/data.log

然后在./conf创建文件exec-memory-logger.conf并将以上命令粘贴

2、启动

flume-ng agent \

--name a1 \

--conf $FLUME\_HOME/conf \

--conf-file $FLUME\_HOME/conf/exec-memory-logger.conf \

-Dflume.root.logger=INFO,console

可以直接控制台向log文件输入内容：

echo hello >> data.log

与实战一同样的操作以及结果显示,但是这个过程明显可以看到的是响应时间很慢

这只是一些简单的输出到控制台的操作，实际工作中，如果是做离线的处理，那么应该将数据输出到hdfs，如果是实时的操作就输出到Kafka，那么对应的就需要修改Sink的类型，官网可以看到相关的配置属性

这里尝试输出到hdfs：

配置文件：

*# example.conf: A single-node Flume configuration*

*# Name the components on this agent*

a1.sources **=** r1

a1.sinks **=** k1

a1.channels **=** c1

*# Describe/configure the source*

a1.sources.r1.type **=** avro

a1.sources.r1.command = tail -F /data/driverLib/data.log

a1.sources.r1.shell = /bin/sh -c

*# Describe the sink*

a1.sinks.k1.type **=** hdfs

a1.sinks.k1.hdfs.path = hdfs://geosciences/data

*# Use a channel which buffers events in memory*

a1.channels.c1.type **=** memory

a1.channels.c1.capacity **=** 1000

a1.channels.c1.transactionCapacity **=** 100

*# Bind the source and sink to the channel*

a1.sources.r1.channels **=** c1

a1.sinks.k1.channel **=** c1

agent1.sources = spooldirSource

agent1.channels = fileChannel

agent1.sinks = hdfsSink

agent1.sources.spooldirSource.type=spooldir

agent1.sources.spooldirSource.spoolDir=/data/lwq/new\_log

agent1.sources.spooldirSource.channels=fileChannel

agent1.sinks.hdfsSink.type=hdfs

agent1.sinks.hdfsSink.hdfs.path=hdfs://dev228:8020/raw/lwq/%y-%m-%d

agent1.sinks.hdfsSink.hdfs.rollInterval = 3600

agent1.sinks.hdfsSink.hdfs.rollSize = 128000000

agent1.sinks.hdfsSink.hdfs.rollCount = 0

agent1.sinks.hdfsSink.hdfs.batchSize = 1000

agent1.sinks.hdfsSink.hdfs.roundValue = 1

agent1.sinks.hdfsSink.hdfs.roundUnit = minute

agent1.sinks.hdfsSink.hdfs.useLocalTimeStamp = true

agent1.sinks.hdfsSink.channel=fileChannel

agent1.sinks.hdfsSink.hdfs.fileType = DataStream

agent1.channels.fileChannel.type = file

agent1.channels.fileChannel.checkpointDir=/usr/share/apache-flume-1.5.0-bin/checkpoint

agent1.channels.fileChannel.dataDirs=/usr/share/apache-flume-1.5.0-bin/dataDir

(TODO 没有成功需要后续学习解决)

flume-ng agent \

--name a1 \

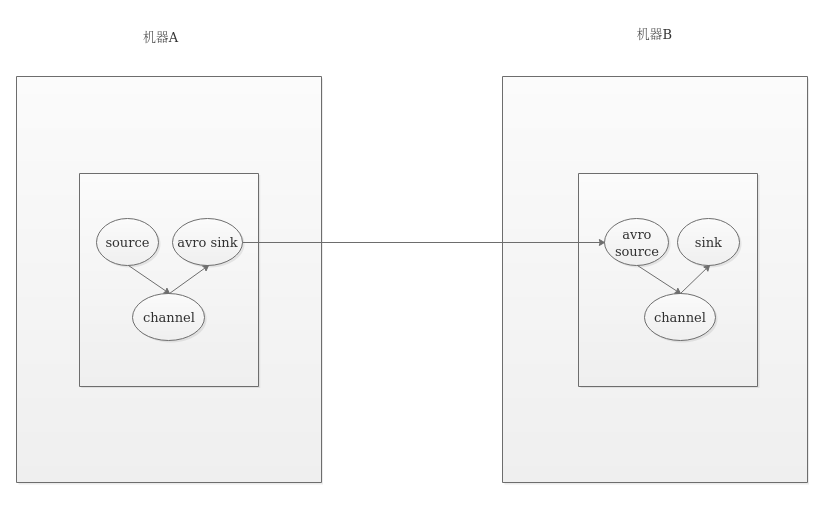
--conf $FLUME\_HOME/conf \

--conf-file $FLUME\_HOME/conf/exec-memory-hdfs.conf \

-Dflume.root.logger=INFO,console

### 3.5.3 实战三

需求：将A服务器上的日志实时采集到B服务器



1. 配置

**A: exec source + memory channel + avro sink**

（注意： 一般工作中跨节点使用avro的sink，那么另外一个机器接收的话使用avro source）

exec-memory-avro.conf

*# example.conf: A single-node Flume configuration*

*# Name the components on this agent*

exec-memory-avro.sources **=** exec-source

exec-memory-avro.sinks **=** avro-sink

exec-memory-avro.channels **=** memory-channel

*# Describe/configure the source*

exec-memory-avro.sources.exec-source.type **=** exec

exec-memory-avro.sources.exec-source.command = tail -F /data/driverLib/data.log

exec-memory-avro.sources.exec-source.shell = /bin/sh -c

*# Describe the sink*

exec-memory-avro.sinks.avro-sink.type **=** avro

exec-memory-avro.sinks.avro-sink.hostname**=** namenode1

exec-memory-avro.sinks.avro-sink.port **=** 44444

*# Use a channel which buffers events in memory*

exec-memory-avro.channels.memory-channel.type **=** memory

*# Bind the source and sink to the channel*

exec-memory-avro.sources.exec-source.channels **=** memory-channel

exec-memory-avro.sinks.avro-sink.channel **=** memory-channel

avro-memory-logger.conf

**B: avro source + memory channel + logger sink**

avro-memory-logger.conf

*# example.conf: A single-node Flume configuration*

*# Name the components on this agent*

avro-memory-logger.sources **=** avro-source

avro-memory-logger.sinks **=** logger-sink

avro-memory-logger.channels **=** memory-channel

*# Describe/configure the source*

avro-memory-logger.sources.avro-source.type **=** avro

avro-memory-logger.sources.avro-source.bind= namenode1

avro-memory-logger.sources.avro-source.port = 44444

*# Describe the sink*

avro-memory-logger.sinks.logger-sink.type **=** logger

*# Use a channel which buffers events in memory*

avro-memory-logger.channels.memory-channel.type **=** memory

*# Bind the source and sink to the channel*

avro-memory-logger.sources.avro-source.channels **=** memory-channel

avro-memory-logger.sinks.logger-sink.channel **=** memory-channel

1. 启动

**先启动**avro-memory-logger.conf

因为这个是监听在44444端口上

flume-ng agent \

--name avro-memory-logger \

--conf $FLUME\_HOME/conf \

--conf-file $FLUME\_HOME/conf/avro-memory-logger.conf \

-Dflume.root.logger=INFO,console

**再启动**exec-memory-avro.conf

flume-ng agent \

--name exec-memory-avro \

--conf $FLUME\_HOME/conf \

--conf-file $FLUME\_HOME/conf/exec-memory-avro.conf \

-Dflume.root.logger=INFO,console

**这里需要注意启动的顺序不能变**

1. 整体流程

1）在机器A上监控一个文件，当我们访问主站时会有用户行为日志记录access.log

2）avro sink把新产生的日志输出到对应的avro source指定的hostname和port上

3）通过avro source将对应的日志输出到控制台（Kafka、HDFS等）

# 第四章 分布式消息队列Kafka

## 4.1 Kafka概述

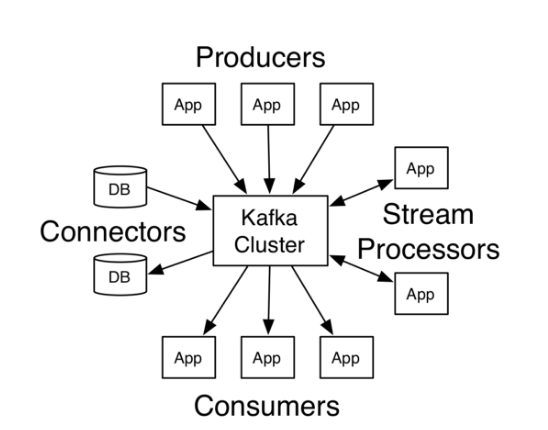
1、官网：

<https://kafka.apache.org/>

简单理解就是和消息系统类似：

消息中间件：生产者消费者

## 4.2 Kafka架构及核心概念



### 4.2.1 Producers

allows an application to publish a stream of records to one or more Kafka topics.

### 4.2.2 Consumers

allows an application to subscribe to one or more topics and process the stream of records produced to them.

### 4.2.3 Brokers

实际上一个broker就相当于是一个Kafka，Kafka就是跑在服务器上的集群，在其上存储的流式记录就称为**topics**

## 4.3 Kafka部署及使用

### 4.3.1 安装Zookeeper

Kafka部署之前需要先安装Zookeeper

1. 对应的镜像下载安装包

http://mirrors.sonic.net/apache/zookeeper/

1. 解压到/usr/local

tar -zxvf zookeeper-3.4.10.tar.gz -C /usr/local

1. 修改配置文件

root@namenode1:/usr/local/zookeeper-3.4.10/conf# cp zoo\_sample.cfg zoo.cfg

root@namenode1:/usr/local/zookeeper-3.4.10/conf# vim zoo.cfg

因为默认的安装目录是/tmp/zookeeper，因为服务器每次启动都会将tmp目删除，所以在/usr/local/ 新建tmp/zookeeper文件夹

并在cfg中设置dataDir=/usr/local/tmp/zookeeper

1. 添加环境变量

export ZOOKEEPER\_HOME=/usr/local/zookeeper-3.4.10

1. 启动

root@namenode1:/usr/local/zookeeper-3.4.10/bin# ./zkServer.sh start

1. 查看运行

通过jps可以查看是否启动，

56631 QuorumPeerMain

接下来可以通过

root@namenode1:/usr/local/zookeeper-3.4.10/bin# ./zkCli.sh

查看客户端，然后类似HDFS可以通过ls命令查看上面的信息



### 4.3.2 单节点单Broker部署及使用

参考

<https://kafka.apache.org/quickstart>

1. 配置环境变量

配置/etc/profile文件，添加kdfka安装路径

1. 修改server.properties属性

注意broker.id

host.name=namenode1

log.dirs=/usr/local/tmp/kafka-logs

zookeeper.connect=namenode1:2181

1. 启动

kafka-server-start.sh 因为之前配置了环境变量

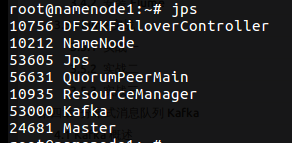
提醒信息：

USAGE: /usr/local/kafka\_2.11-1.0.0/bin/kafka-server-start.sh [-daemon] server.properties [--override property=value]\*

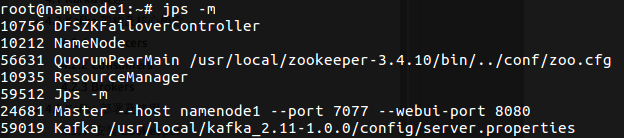
所以需要指定使用的server属性文件

kafka-server-start.sh /usr/local/kafka\_2.11-1.0.0/config/server.properties

这时候可以查看jps发现有Kafka的进程



通过jps -m可以查看进程的路径



1. 创建topic（zookeeeper）

kafka-topics.sh --create --zookeeper namenode1:2181 --replication-factor 1 --partitions 1 --topic hello\_topic

1. 查看创建的topic

kafka-topics.sh --list --zookeeper namenode1:2181

1. 发送消息（生产）（broker）

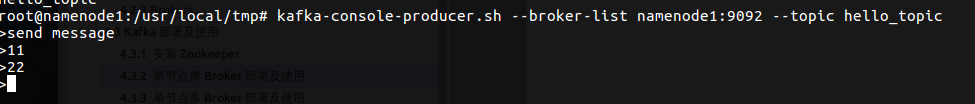
kafka-console-producer.sh --broker-list namenode1:9092 --topic hello\_topic

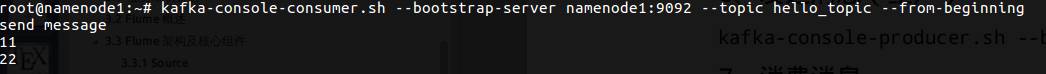
1. 消费消息（zookeeper）

kafka-console-consumer.sh --bootstrap-server namenode1:9092 --topic hello\_topic --from-beginning

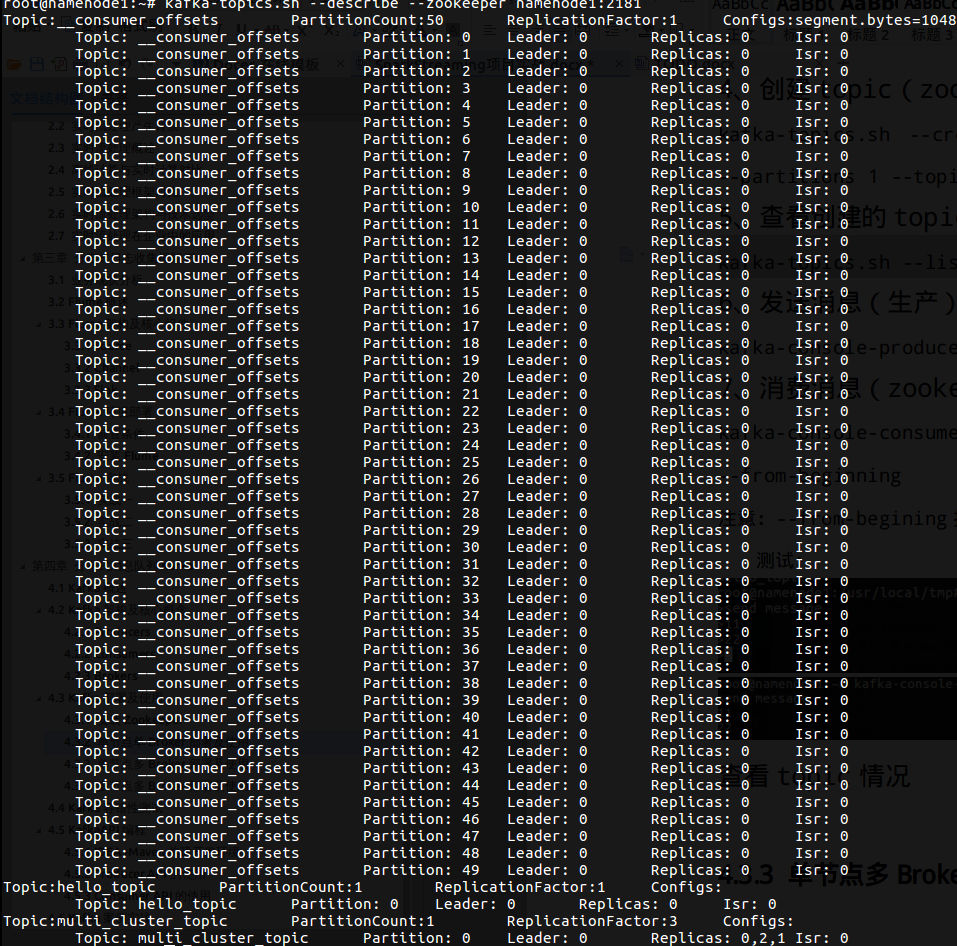
注意：--from-begining指的是每次打开加载上一次发送的消息

测试：





查看所有topic情况



### 4.3.3 单节点多Broker部署及使用

与上面安装类似，只是将对应的设置多添加

先是修改文件，到$KAFKA\_HOME/config目录，新建server-1.properties、server-2.properties或更多文件，其中需要修改的地方分别为：**broker.id、listener端口、log.dirs**

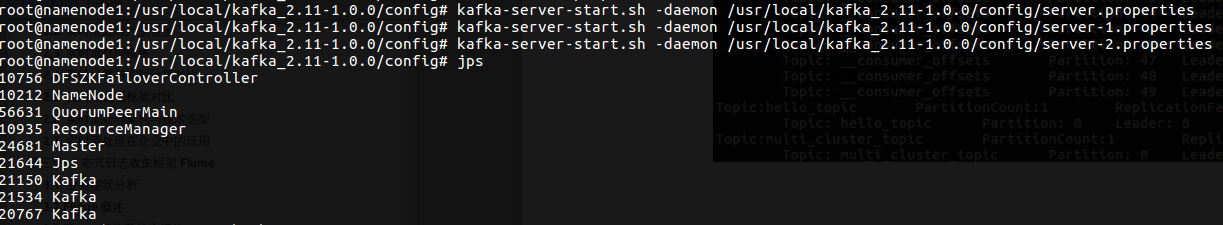
1. 启动

kafka-server-start.sh -daemon /usr/local/kafka\_2.11-1.0.0/config/server.properties

kafka-server-start.sh -daemon /usr/local/kafka\_2.11-1.0.0/config/server-1.properties

kafka-server-start.sh -daemon /usr/local/kafka\_2.11-1.0.0/config/server-2.properties

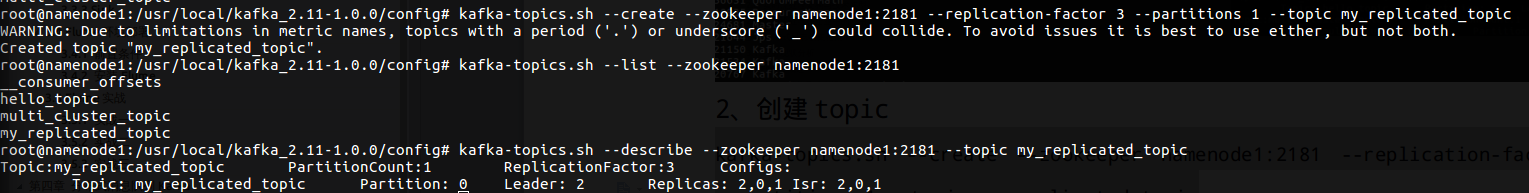
其中-daemon参数表示后台启动



1. 创建topic

kafka-topics.sh --create --zookeeper namenode1:2181 --replication-factor 3 --partitions 1 --topic my\_replicated\_topic

1. 查看topic



其中参数介绍：副本等是之前创建时候定义的，leader表示主topic，Isr表示alive topic

1. 创建生产者

kafka-console-producer.sh --broker-list namenode1:9092,namenode1:9093,namenode1:9094 --topic my\_replicated\_topic

1. 创建消费者

kafka-console-consumer.sh --bootstrap-server namenode1:9092 --topic my\_replicated\_topic

1. 测试





### 4.3.4 多节点多Broker部署及使用（需要多个机器）

## 4.4 Kafka容错性测试

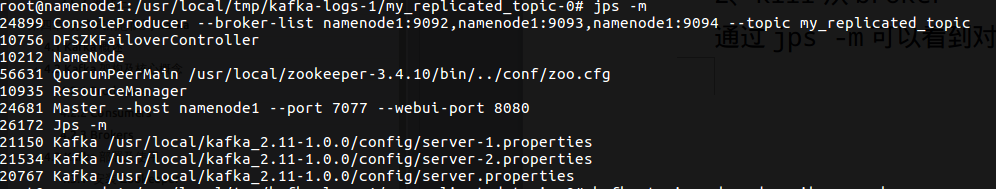
1、显示当前broker



可以看到当前有三个分别为0,1,2，而且2为leader

1. kill从broker

通过jps -m可以看到对应节点的id



然后通过kill -9 21150将1干掉



发现已经没有了



而且显示的只剩下了2,0

但是这个时候通过生产者消费者还是可以传输信息，这部分就不截图可以自己测试

1. kill主broker

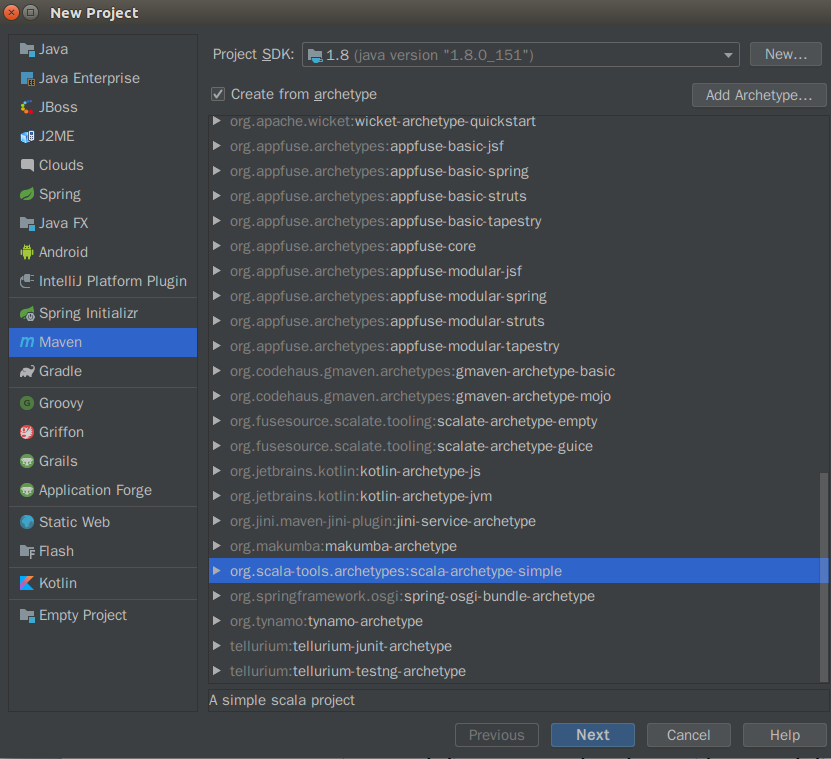
接下来尝试将当前leader干掉，发现同样的可以发送信息，也就是说只要有一个topic是active的，那么就可以执行下去，这与HDFS的副本机制很类似，所以Kafka的容错性是很好的

通过kill命令干掉对应的服务，只要有一个broker运行，就是可以传输的，如果是把leader终止，那么会有相对应的进程来补充作为leader，这就是Kafka的容错性测试

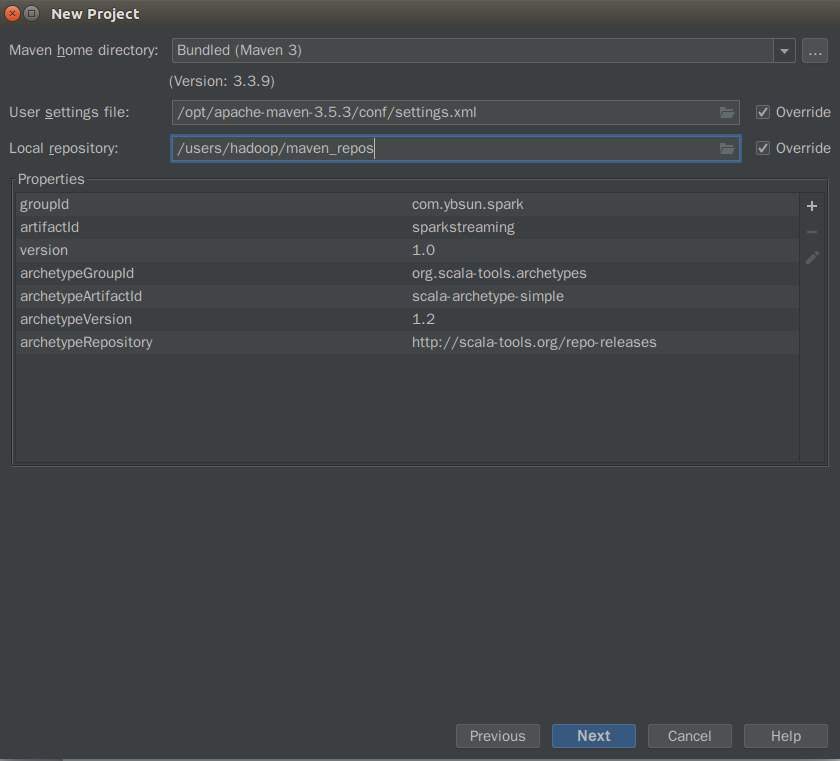
## 4.5 KafkaAPI编程

### 4.5.1 IDEA+Maven构建开发环境

1、新建maven项目（Scala-simple）



2、设置maven配置路径



这部分遇到一些小问题，具体解决在自己个人博客总结。

3、配置pom.xml文件

同样新建一个Scala的模板项目，要修改xml文件的配置路径，然后进行修改xml相关依赖

<dependencies>

<dependency>

<groupId>org.scala-lang</groupId>

<artifactId>scala-library</artifactId>

<version>${scala.version}</version>

</dependency>

<!--kafka 依赖-->

<!-- https://mvnrepository.com/artifact/org.apache.kafka/kafka -->

<dependency>

<groupId>org.apache.kafka</groupId>

<artifactId>kafka\_2.11</artifactId>

<version>${kafka.version}</version>

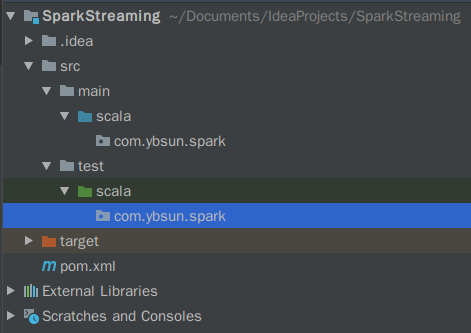
</dependency>

</dependencies>

这里的配置文件主要就是把Kafka导入

1. 通过Java编程

Scala与Java是可以互相调用的，可以完美兼容，所以我们首先通过Java来进行编程，把项目中默认生成的scala文件都删除，项目结构如下：

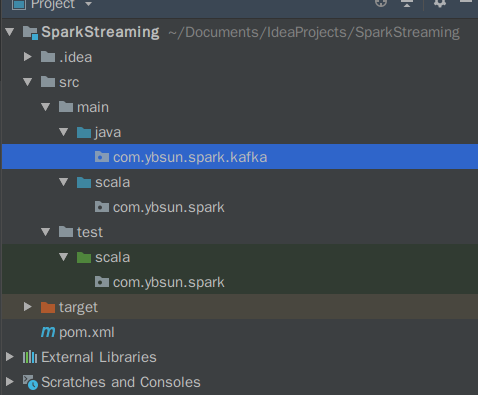


接下来新建Java文件夹并标为source文件夹，并新建包名称为

com.ybsun.spark.kafka

这个包主要是为了存放Kafka开发的一些代码

项目结构建好了如下：



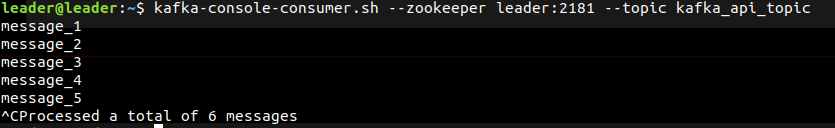
到这里为止，我们的项目结构以及相关配置就已经建立完善，接下来就是通过代码实现相应的功能。

### 4.5.2 Producer API的使用

这部分具体的代码见我的github网址：

[https://github.com/YaboSun/](https://github.com/YaboSun/SparkStreaming)Log-Analysis-with-Spark

实现效果：



### 4.5.3 Consumer API的使用

同上，都通过相应的代码来实现

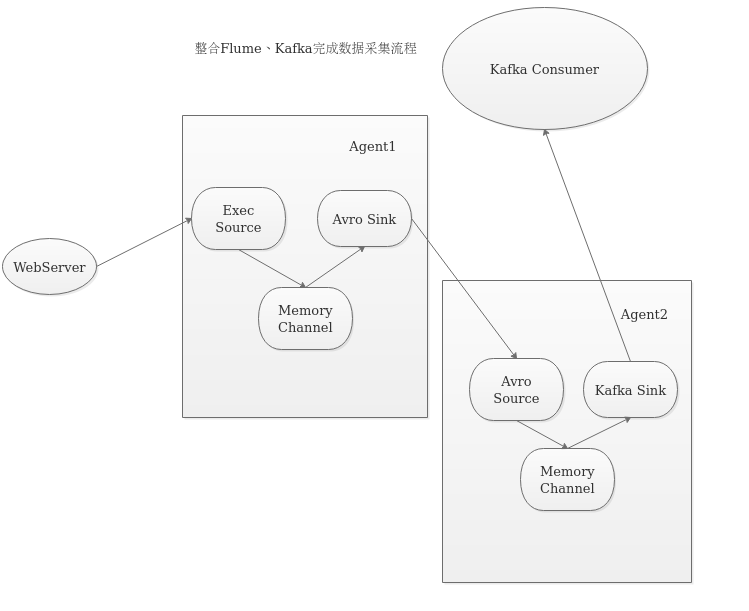
最终效果：



这里可以看到还是有bug，正常来说是应该发送然后收到，但是这里有的消息还没有发送，就有接收到？线程问题？

## 4.6 Kafka案例实战

整体流程架构



1. 配置文件

配置Flume

新建文件avro-memory-kafka.conf，配置信息如下：(对应的是1.7或以上)

avro-memory-kafka.sources = avro-source

avro-memory-kafka.sinks = kafka-sink

avro-memory-kafka.channels = memory-channel

# Describe/configure the source

avro-memory-kafka.sources.avro-source.type = avro

avro-memory-kafka.sources.avro-source.bind= namenode1

avro-memory-kafka.sources.avro-source.port = 44444

# Describe the sink

avro-memory-kafka.sinks.kafka-sink.type = org.apache.flume.sink.kafka.KafkaSink

avro-memory-kafka.sinks.kafka-sink.kafka.bootstrap.servers = namenode1:9092

avro-memory-kafka.sinks.kafka-sink.kafka.topic = kafka\_api\_topic

avro-memory-kafka.sinks.kafka-sink.flumeBatchSize = 5

avro-memory-kafka.sinks.kafka-sink.kafka.producer.acks = 1

# Use a channel which buffers events in memory

avro-memory-kafka.channels.memory-channel.type = memory

# Bind the source and sink to the channel

avro-memory-kafka.sources.avro-source.channels = memory-channel

avro-memory-kafka.sinks.kafka-sink.channel = memory-channel

经过查阅1.6对应的sink详细配置如下：

avro-memory-kafka.sources = avro-source

avro-memory-kafka.sinks = kafka-sink

avro-memory-kafka.channels = memory-channel

# Describe/configure the source

avro-memory-kafka.sources.avro-source.type = avro

avro-memory-kafka.sources.avro-source.bind = leader

avro-memory-kafka.sources.avro-source.port = 44444

# Describe the sink

avro-memory-kafka.sinks.kafka-sink.type = org.apache.flume.sink.kafka.KafkaSink

avro-memory-kafka.sinks.kafka-sink.brokerList = leader:9092

avro-memory-kafka.sinks.kafka-sink.topic = kafka\_api\_topic

avro-memory-kafka.sinks.kafka-sink.batchSize = 5

avro-memory-kafka.sinks.kafka-sink.requiredAcks = 1

# Use a channel which buffers events in memory

avro-memory-kafka.channels.memory-channel.type = memory

# Bind the source and sink to the channel

avro-memory-kafka.sources.avro-source.channels = memory-channel

avro-memory-kafka.sinks.kafka-sink.channel = memory-channel

另外需要配置exec-memory-avro.conf

详细配置如下：

exec-memory-avro.sources = exec-source

exec-memory-avro.sinks = avro-sink

exec-memory-avro.channels = memory-channel

# Describe/configure the source

exec-memory-avro.sources.exec-source.type = exec

exec-memory-avro.sources.exec-source.command = tail -F /home/leader/Documents/data/data.log

exec-memory-avro.sources.exec-source.shell = /bin/sh -c

# Describe the sink

exec-memory-avro.sinks.avro-sink.type = avro

exec-memory-avro.sinks.avro-sink.hostname= leader

exec-memory-avro.sinks.avro-sink.port = 44444

# Use a channel which buffers events in memory

exec-memory-avro.channels.memory-channel.type = memory

# Bind the source and sink to the channel

exec-memory-avro.sources.exec-source.channels = memory-channel

exec-memory-avro.sinks.avro-sink.channel = memory-channel

1. 启动Flume

flume-ng agent \

--name avro-memory-kafka \

--conf $FLUME\_HOME/conf \

--conf-file $FLUME\_HOME/conf/avro-memory-kafka.conf \

-Dflume.root.logger=INFO,console

**再启动**exec-memory-avro.conf

**启动：**

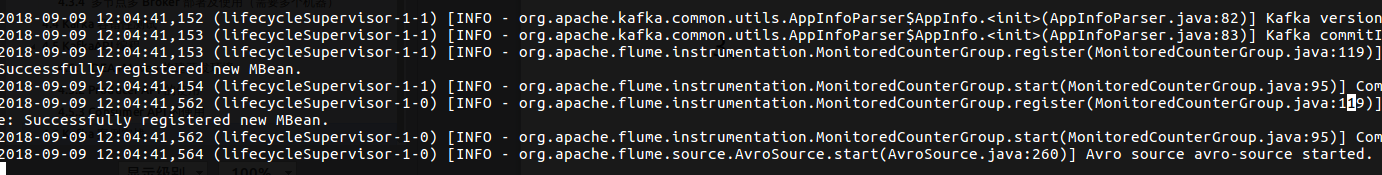
flume-ng agent \

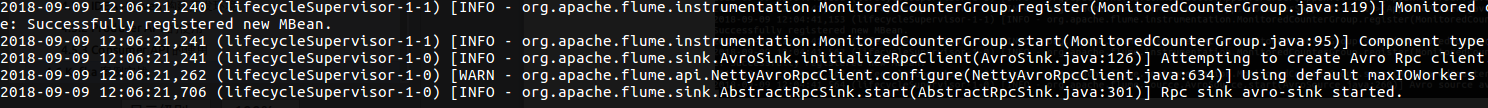
--name exec-memory-avro \

--conf $FLUME\_HOME/conf \

--conf-file $FLUME\_HOME/conf/exec-memory-avro.conf \

-Dflume.root.logger=INFO,console

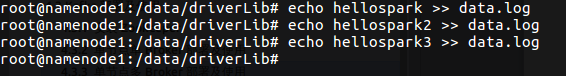




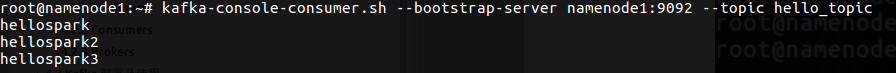
启动成功

1. 消费

接下来需要启动一个Kafka客户端来消费



向其中写入数据，然后查看控制台输出



到这里为止，就把所有的涉及Flume、Kafka单独的案例，以及俩者结合的案例全部跑通，后面就是结合SparkStreaming来进行其他开发，准备工作基本完成。

# SparkStreaming入门

## 5.1 概述

<http://spark.apache.org/docs/latest/streaming-programming-guide.html>

Spark Streaming is an extension of the core Spark API that enables scalable, high-throughput, fault-tolerant stream processing of live data streams.

SparkStreaming是core Spark API的一个扩展，具有可扩展、高吞吐量、容错的特性，是用来处理流数据的。

**定义：**

将不同的数据源数据经过SparkStreaming处理之后将结果输出到外部文件系统

**特点：**

低延时

能够从错误中高效的恢复

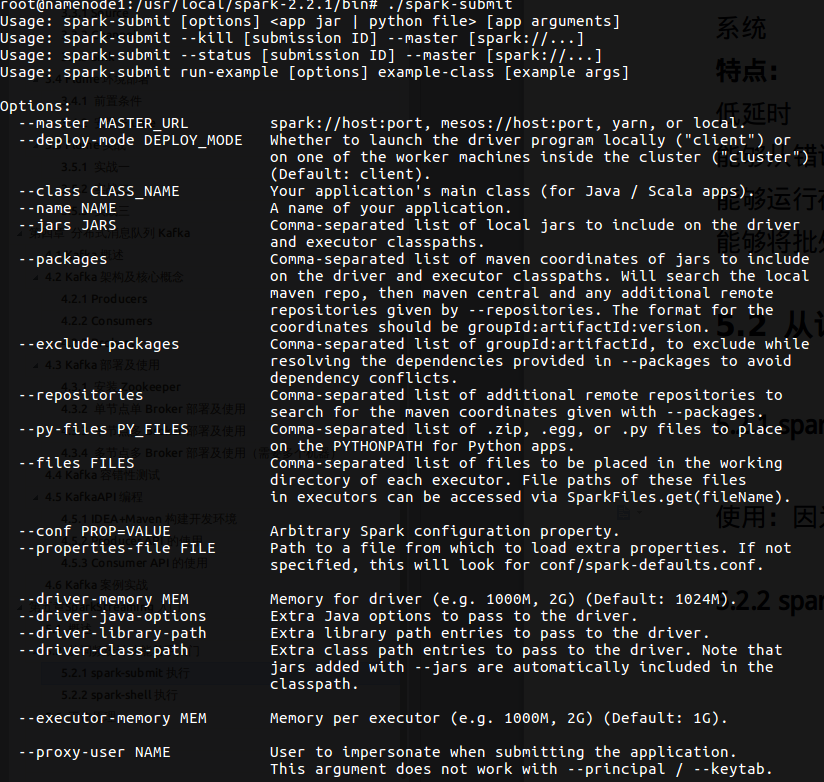
能够运行在成百上千的节点

能够将批处理、机器学习、图计算等子框架和SparkStreaming综合起来使用

## 5.2 从词频统计功能着手入门

### 5.2.1 spark-submit执行(生产)

1、使用：因为不清楚怎么使用，可以直接到在控制台输入./spark-submit



则会输出一系列的用法说明

2、官网案例测试

通过运行一个Spark的官方案例来测试spark-submit的使用

<https://github.com/apache/spark/blob/master/examples/src/main/java/org/apache/spark/examples/streaming/JavaNetworkWordCount.java>（案例网址）

通过查看运行说明，可以发现运行需要首先运行一个Netcat server

接下来测试运行，命令如下：

./spark-submit --master local[2] \

--class org.apache.spark.examples.streaming.NetworkWordCount\

--name NetworkWordCount \

/usr/local/spark-2.2.1/examples/jars/spark-examples\_2.11-2.2.1.jar namenode1 9999

### 5.2.2 spark-shell执行(测试)

./spark-shell --master local[2]

import org.apache.spark.streaming.{Seconds, StreamingContext}

val ssc = new StreamingContext(sc, Seconds(1))

val lines = ssc.socketTextStream(“namenode1”, 9999)

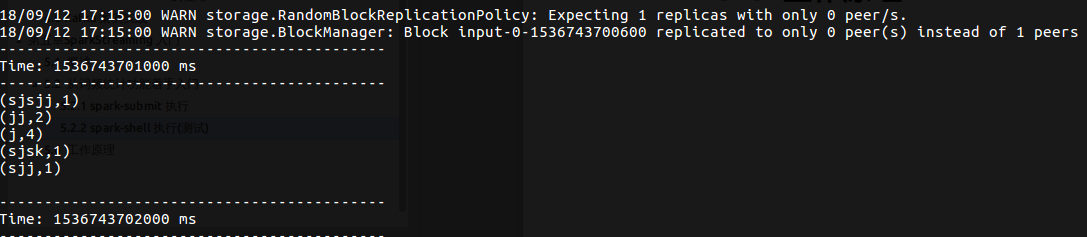
val words = lines.flatMap(\_.split(" "))

val wordCounts = words.map(x => (x, 1)).reduceByKey(\_ + \_)

wordCounts.print()

ssc.start()

ssc.awaitTermination()

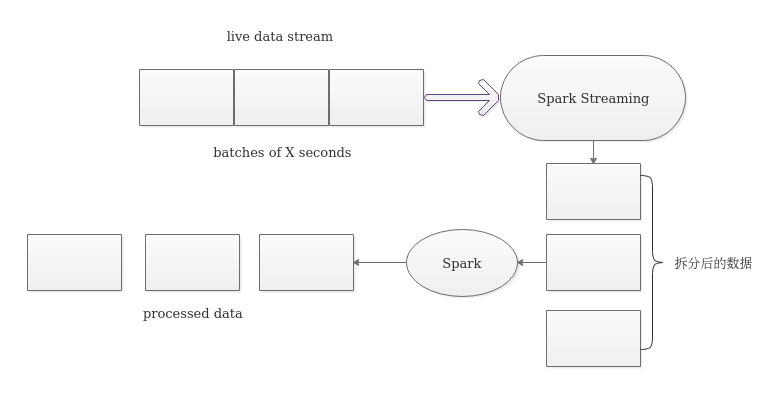


## 5.6 工作原理

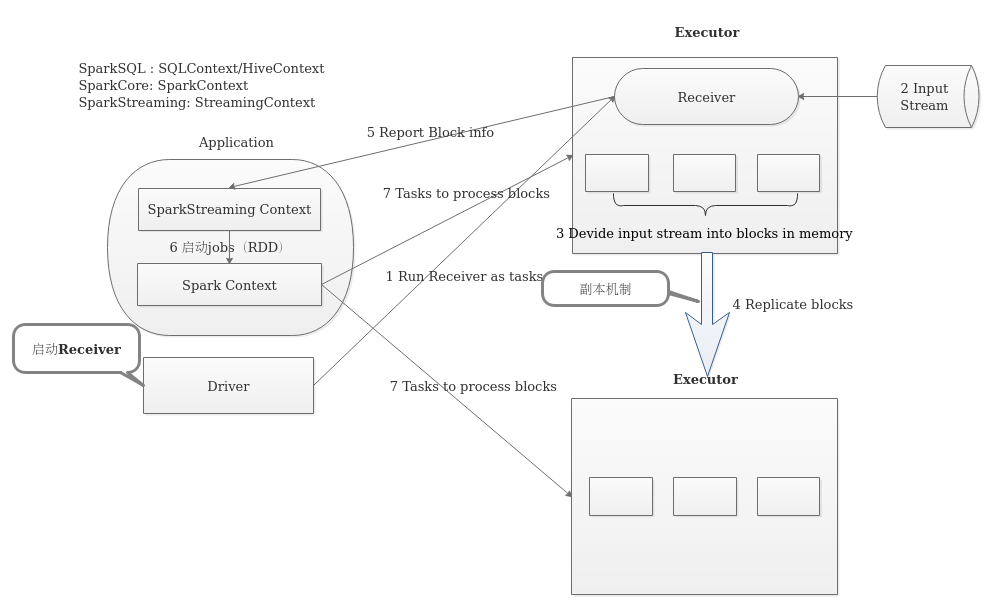
### 5.6.1 粗粒度

工作原理：

SparkStreaming接收到实时数据流，然后把数据按照指定时间段切成一片片小的数据块，然后把数据块交由Spark引擎处理，最后输出处理后的数据块，图解如下：



### 5.6.2 细粒度



以上就是细粒度的详细流程图，先对整体的架构有一个认识，后面会对每个组件进行学习与使用。

# SparkStreaming核心

## 6.1 核心概念

### 6.1.2 StreamingContext

查看官网的说明：

<http://spark.apache.org/docs/2.2.1/streaming-programming-guide.html>

有关说明：Initializing StreamingContext

要进行一个SparkStreaming项目的开发，必须创建一个StreamingContext。

大致流程就是：首先创建一个SparkConf对象，通过其来创建

1. **查看StreamingContext源码**

构造方法：

主构造器：

**class** StreamingContext **private**[streaming] (  
 \_sc: SparkContext,  
 \_cp: Checkpoint,  
 \_batchDur: Duration  
 )

附属构造器（带this）：

**def this**(sparkContext: SparkContext, batchDuration: Duration) = {  
 **this**(sparkContext, **null**, batchDuration)  
}

**def this**(conf: SparkConf, batchDuration: Duration) = {  
 **this**(StreamingContext.*createNewSparkContext*(conf), **null**, batchDuration)  
}

1. **运行设置**

运行过程中不要将--master写死（hardcode）

1. **batchDuration设置**

这个指的就是应用程序对延迟性的要求，是必须设置的，就是Spark微批处理的体现之处，需要综合考虑**项目需求以及资源利用**

**4、After a context is defined, you have to do the following：**

1. Define the input sources by creating input DStreams.

通过创建输入DStreams来定义输入源

1. Define the streaming computations by applying transformation and output operations to DStreams.

通过应用transformation以及output operations到DStreams来定义streaming计算

1. Start receiving data and processing it using streamingContext.start().

使用streamingContext.start()来进行接收以及处理数据

1. Wait for the processing to be stopped (manually or due to any error) using streamingContext.awaitTermination().

使用streamingContext.awaitTermination()来等待一个处理结束（手动或出错）

1. The processing can be manually stopped using streamingContext.stop().

处理过程可以通过使用streamingContext.stop()来手动结束

**5、注意事项**

* Once a context has been started, no new streaming computations can be set up or added to it.

一旦一个context启动，不能再设置或者添加新的流式计算

* Once a context has been stopped, it cannot be restarted.

一旦一个context被停止就不能再被重启

* Only one StreamingContext can be active in a JVM at the same time.

在相同时间内只能有一个active的StreamingContext在一个JVM上

* stop() on StreamingContext also stops the SparkContext. To stop only the StreamingContext, set the optional parameter of stop() called stopSparkContext to false.

使用stop()方法不仅将StreamingContext停止也停止了SparkContext，如果想仅仅停止StreamingContext，需要将stop()调用stopSparkContext将参数设置为false

* A SparkContext can be re-used to create multiple StreamingContexts, as long as the previous StreamingContext is stopped (without stopping the SparkContext) before the next StreamingContext is created.

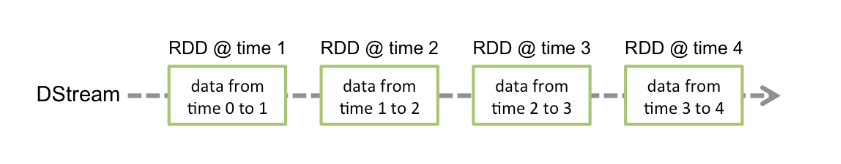
一个SparkContext可以被重复用来创建多个StreamingContexts，只要是在创建下一个StreamingContext之前将前一个StreamingContext停止

### 6.1.2 DStreams（Discretized Streams）

Discretized:离散化

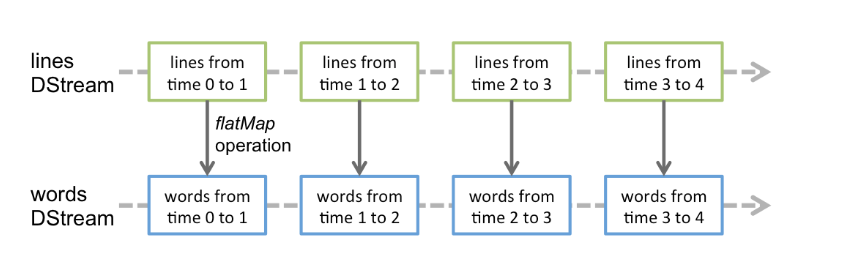
Internally, a DStream is represented by a continuous series of RDDs

实际上，一个DStream代表的是一系列的RDD，图解如下：



通过图片可以看到，按照Batch Duration将一个DStream拆分为多个RDD，每个的间隔就是设置的时间大小，比如说1 second。这里解释一下Spark的核心概念：RDD，是Spark的基本数据结构，全称为弹性分布式数据集，spark中的所有数据都是通过RDD的形式进行组织。

下面的一张图是展示了之前的案例中的过程：



可以看到通过对DStream进行map/flatMap操作，实际上是因为一个DStream是由多个RDD组成，所以实际上就是对于每个RDD都执行了相应的操作。

### 6.1.3 Input DStreams and Receivers

Every input DStream (except file stream, discussed later in this section) is associated with a ****Receiver**** object which receives the data from a source and stores it in Spark’s memory for processing.

每一个Input DStream（除了文件系统：文件系统不需要）都与一个Receiver关联，receiver是用从源头接收数据，并将数据存储在内存中用于处理。

另外需要注意的俩点：

1. 当你运行一个SparkStreaming项目以local的模式，不要使用“local”或者“local[1]”来指定masterURL，因为在运行过程中每个receiver会占用一个线程，所以应该使用“local[n]”，这里的n需要是大于receivers的个数

2、Extending the logic to running on a cluster, the number of cores allocated to the Spark Streaming application must be more than the number of receivers. Otherwise the system will receive data, but not be able to process it.

集群的方式运行的时候需要是大于receivers的个数

## 6.2 Transformations

Similar to that of RDDs, transformations allow the data from the input DStream to be modified. DStreams support many of the transformations available on normal Spark RDD’s

类似于RDDs，transformations 允许来自input DStream的数据进行修改，DStream支持很多一般的SparkRDD的操作，具体的操作函数在后面的实战中进行详细说明。

## 6.3 Output Operations

Output operations allow DStream’s data to be pushed out to external systems like a database or a file systems. Since the output operations actually allow the transformed data to be consumed by external systems, they trigger the actual execution of all the DStream transformations (similar to actions for RDDs).

Output 操作允许DStream的数据输出到外部系统比如说数据库以及文件系统等。正因为output 操作可以允许外部系统来传输数据。先简单进行介绍，其实后面用到的时候直接查来用

## 6.4 案例实战

### 6.4.1 SparkStreaming处理Socket数据

这里出现了一个bug就是error: '{' expected but 'import' found scala

很奇怪这个，别人说是分割符的问题，但是我修改了还是没有作用，最后重新换了一遍编译就可以了，这个有点奇怪？？

1、编写scala的NetworkWordCount

详细代码见github

<https://github.com/YaboSun/Log-Analysis-with-Spark/blob/master/src/main/scala/spark/common/NetworkWordCount.scala>

2、启动一个网络socket

nc -lk 6789

3、启动程序并在终端输入数据，这个时候可以看到在代码中指定的时间间隔内会对输入的数据进行统计，统计每个单词出现的次数

**排坑：**

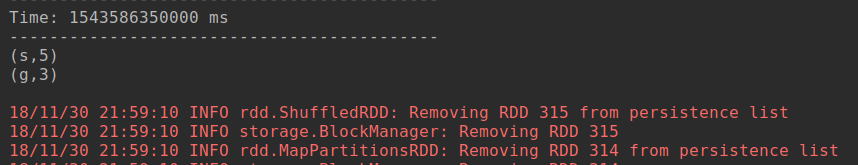
**1、需要注意的是这里如果本地测试，因为启动一个receiver就会占用一个线程，所以指定的local[]个数至少要比receiver的个数多一个**

### 6.4.2 SparkStreaming处理文件系统数据

这里的文件系统既包括本地文件系统也包括HDFS，都是可以的

详细代码github，这个时候的测试就是将一个新建的文件，通过cp等方式传到指定的目录中

运行结果：



**注意：**

**1、这个文件系统可以是本地或者hdfs，另外兼容hdfs的文件系统都可以实现**

**2、这个在代码中指定的local不需要指定线程数，或者说不写或者写为1都是可以的**

**3、一旦你的文件已经被移动到了指定目录，那么你后面针对这个文件进行的任何修改都不会再次进行统计**

# 第七章 Sparktreaming进阶