

第1章 课程介绍

Spark版本升级

第2章 初识实时流处理

一、业务现状分析

二、离线与实时对比

三、框架对比

四、架构与技术选型

五、应用

第3章 分布式日志收集框架Flume

一、业务现状分析

二、Flume架构及核心组件

三、Flume&JDK环境部署

 1.前置条件

 2.安装Flume

四、Flume实战

 1.案例一

 2.案例二

 3.案例三

第4章 分布式发布订阅消息系统Kafka

一、Kafka概述

二、Kafka单节点单Broker部署

 1.Zookeeper安装

 2.单节点单broker的部署及使用

 3.单节点多broker部署及使用

三、Kafka容错性测试

四、使用IDEA+Maven构建开发环境

五、Kafka Java API编程

 1.Producer

 (1)提供端

 (2)消费端

 (3)测试

 2.Consumer

六、整合Flume和Kafka完成实时数据采集

 1.架构图

 2.修改flume相关文件

 3.启动

第5章 实战环境搭建

一、安装Scala、maven、hadoop

二、HBase安装

三、spark安装

三、开发环境搭建

第6章 Spark Streaming入门

一、Spark Streaming概述

二、Spark Streaming集成Spark生态系统的使用

三、词频统计功能着手入门Spark Streaming

 1.spark-submit

 2.spark-shell

四、Spark Streaming工作原理(粗粒度)

五、Spark Streaming工作原理(细粒度)

第7章 Spark Streaming核心概念与编程

一、核心概念

 1.StreamingContext

 2.DStream

 3.Input DStreams和Receivers

二、案例实战

- 1.Spark Streaming处理socket数据
- 2.Spark Streaming处理文件系统数据

第8章 Spark Streaming进阶与案例实战

- 一、目录
- 二、updateStateByKey算子的使用
- 三、统计结果写入到MySQL数据库中
- 四、窗口函数的使用
- 五、黑名单过滤
 - 1.需求分析
 - 2.程序实现
- 六、Spark Streaming整合Spark SQL操作

第9章 Spark Streaming整合Flume

- 一、Push方式整合之Flume Agent配置开发
- 二、Pull方式整合之Flume Agent配置开发

第10章 Spark Streaming整合Kafka

- 一、Receiver方式整合之Kafka
- 二、Spark Streaming应用开发
 - 1.本地测试
 - 2.服务器环境联调
- 三、Direct方式整合

第11章 Spark Streaming整合Flume&Kafka打造通用流处理基础

- 一、课程目录
- 二、处理流程画图剖析
- 三、日志产生器开发并结合log4j完成日志的输出
- 四、使用Flume采集Log4j产生的日志
- 五、使用KafkaSink将Flume收集到的数据输出到Kafka
- 六、Spark Streaming消费Kafka的数据进行统计
- 七、本地测试和生产环境使用的拓展

第12章 Spark Streaming项目实战

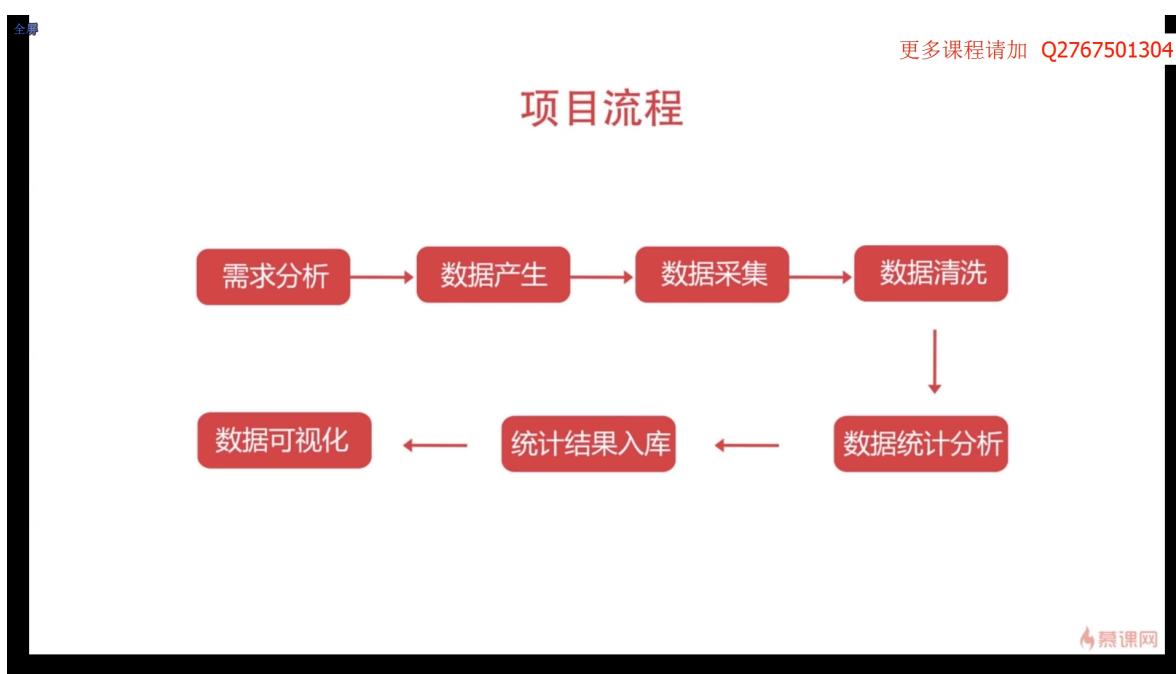
- 一、Python日志产生器开发
 - 1.产生访问url和ip信息
 - 2.产生referer和状态码信息
 - 3.产生日志访问时间
 - 4.服务器测试并将日志写入到文件中
 - 5.定时调度工具
- 二、使用Flume实时收集日志信息
- 三、对接实时日志数据到Kafka并输出到控制台测试
- 四、Spark Streaming对接Kafka的数据进行消费
- 五、使用Spark Streaming完成数据清洗操作
- 六、功能一
 - 1.需求分析及存储结果技术选型分析
 - 2.数据库访问DAO层方法定义
 - 3.HBase操作工具类开发
 - 4.数据库访问DAO层方法实现
 - 5.Spark Streaming的处理结果写入到HBase中
- 七、功能二
 - 1.需求分析及HBase设计&HBase数据访问层开发
 - 2.功能实现及本地测试
- 八、将项目运行在服务器环境中

第13章 可视化实战

- 一、构建Spring Boot项目
- 二、Spring Boot整合Echarts
 - 1.下载
 - 2.绘制静态数据柱状图
 - 3.绘制静态数据饼图
- 三、项目目录调整
- 四、根据天来获取HBase表中的实战课程访问次数
 - 1.准备

- 2.访问hbase
- 五、实战课程访问量domain以及dao开发
- 六、实战课程访问量Web层开发
- 七、实战课程访问量实时查询展示功能实现及扩展
- 八、Spring Boot项目部署到服务器上运行

第1章 课程介绍



需要大家在linux的hadoop用户的根目录(/home/hadoop)上创建如下目录

app	存放我们所有的软件的安装目录
data	存放我们的测试数据
lib	存放我们开发的jar
software	存放软件安装包的目录
source	存放我们的框架源码

Hadoop生态系统: cdh5.7.0
所有的Hadoop生态的软件下载地址: <http://archive.cloudera.com/cdh5/cdh/5/>
jdk: 1.8
spark: 2.2
scala: 2.11.8

Spark版本升级

[网址](#)

```
hadoop@hadoop000:~/lib
export PATH=$ZK_HOME/bin:$PATH

export KAFKA_HOME=/home/hadoop/app/kafka_2.11-0.9.0.0
export PATH=$KAFKA_HOME/bin:$PATH

export SCALA_HOME=/home/hadoop/app/scala-2.11.8
export PATH=$SCALA_HOME/bin:$PATH

export MAVEN_HOME=/home/hadoop/app/apache-maven-3.3.9
export PATH=$MAVEN_HOME/bin:$PATH

export HADOOP_HOME=/home/hadoop/app/hadoop-2.6.0-cdh5.7.0
export PATH=$HADOOP_HOME/bin:$PATH

export HBASE_HOME=/home/hadoop/app/hbase-1.2.0-cdh5.7.0
export PATH=$HBASE_HOME/bin:$PATH

export SPARK_HOME=/home/hadoop/app/spark-2.2.0-bin-2.6.0-cdh5.7.0
export PATH=$SPARK_HOME/bin:$PATH

[hadoop@hadoop000 app]$
```

更多课程请加 Q2767501304

慕课网

第2章 初识实时流处理

一、业务现状分析

需求：统计主站每个（指定）课程访问的客户端、地域信息分布

地域：ip转换 Spark SQL项目实战

客户端：useragent获取 Hadoop基础课程

==> 如上两个操作：采用离线（Spark/MapReduce）的方式进行统计

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

项目架构

日志收集：Flume

离线分析：MapReduce/Spark

统计结果图形化展示

二、离线与实时对比

离线计算与实时计算的对比

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

2) 处理过程

离线: MapReduce: map + reduce

实时: Spark(DStream/SS)

3) 处理速度

离线: 慢

实时: 快速

4) 进程

离线: 启动+销毁

实时: 7*24

三、框架对比

实时流处理框架对比

◆ Apache Storm

◆ Apache Spark Streaming

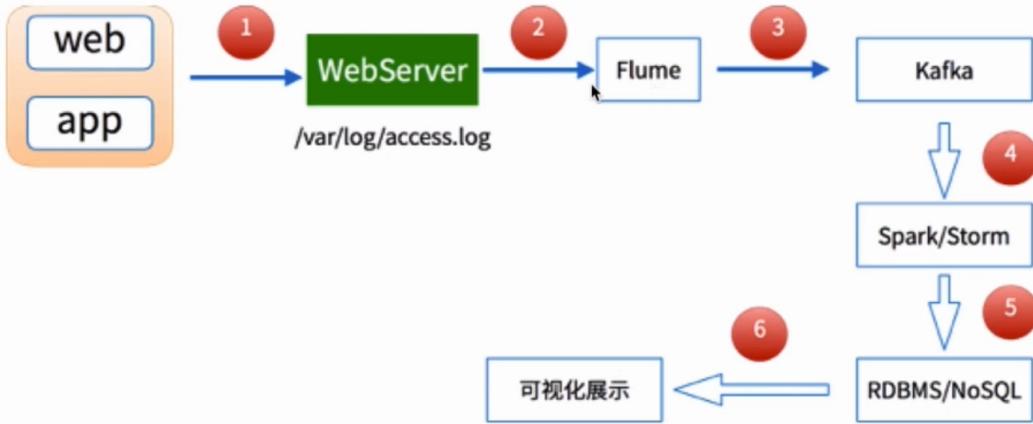
◆ IBM Stream

◆ Yahoo! S4

◆ LinkedIn Kafka

四、架构与技术选型

实时流处理架构与技术选型



五、应用

实时流处理在企业中的应用

◆ 电信行业

◆ 电商行业

第3章 分布式日志收集框架Flume

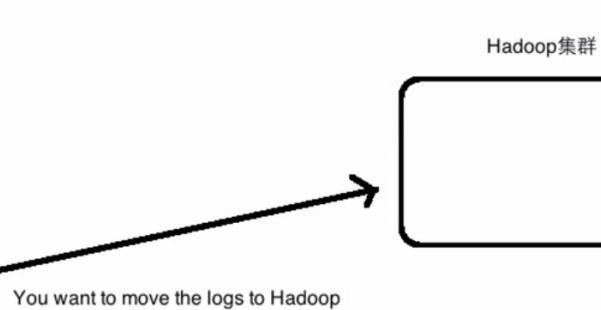
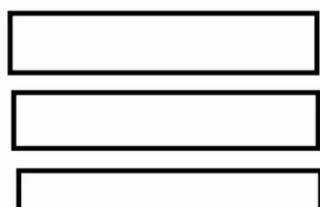
一、业务现状分析

业务现状分析

- ◆ WebServer/ApplicationServer分散在各个机器上
- ◆ 想大数据平台Hadoop进行统计分析
- ◆ 日志如何收集到Hadoop平台上
- ◆ 解决方案及存在的问题

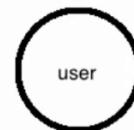
You have a lot of servers and systems

- A) network devices
- B) operating system
- C) web servers
- D) Applications



You want to move the logs to Hadoop

They generate a lot of logs and other data



I have business idea
how to use this data???

现状分析：见图

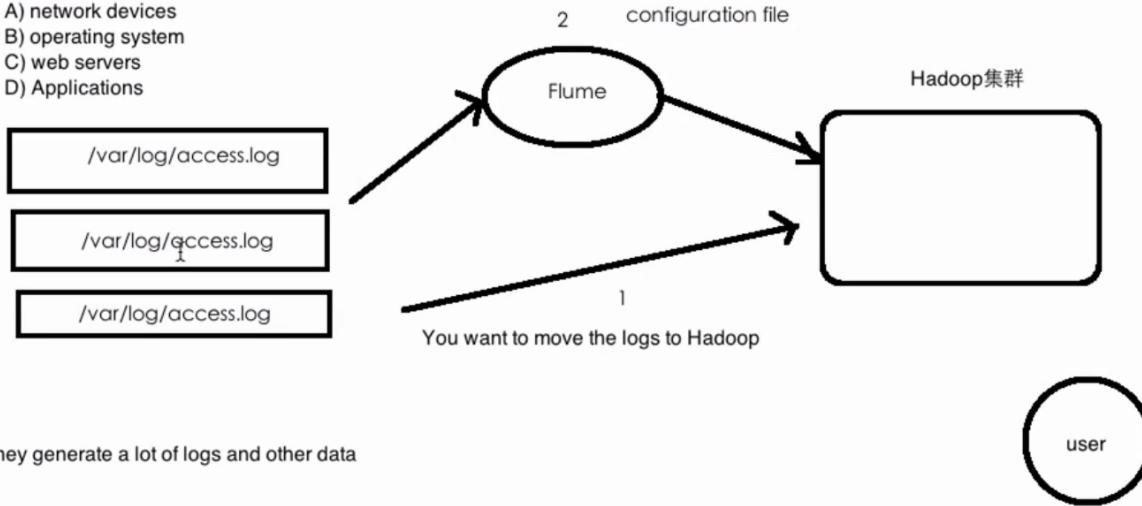
如何解决我们的数据从其他的server上移动到Hadoop之上？？？

shell cp hadoop集群的机器上, hadoop fs -put /

====> Flume

You have a lot of servers and systems

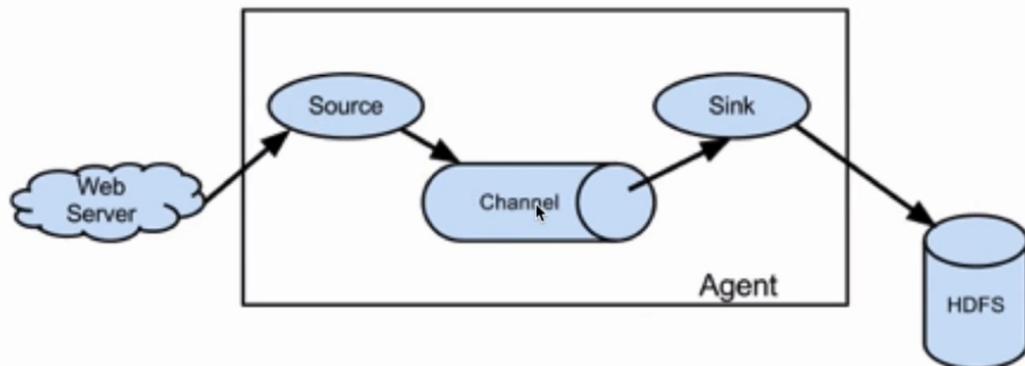
- A) network devices
- B) operating system
- C) web servers
- D) Applications



二、Flume架构及核心组件

[官网](#)

Flume架构及核心组件

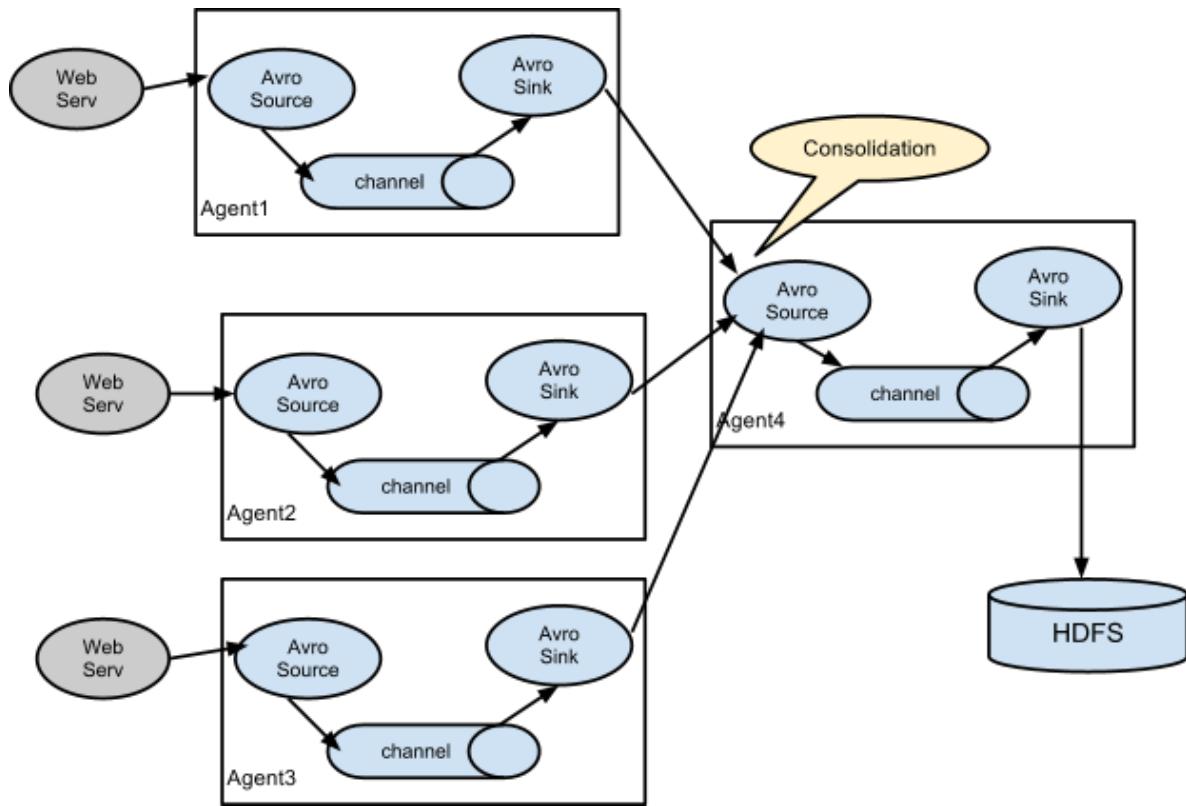


Flume架构及核心组件

1) Source 收集

2) Channel 聚集

3) Sink 输出



三、Flume&JDK环境部署

1.前置条件

Java Runtime Environment - Java 1.8 or later
 Memory - Sufficient memory for configurations used by sources, channels or sinks
 Disk Space - Sufficient disk space for configurations used by channels or sinks
 Directory Permissions - Read/write permissions for directories used by agent

2.安装Flume

[下载](#)

Index of cdh5/cdh/5/

不安全 | archive.cloudera.com/cdh5/cdh/5/

应用 Google 学术搜索 平台 在线工具 算法 大数据 Chrome 网上应用店 gis docker

flume-ng-1.6.0-cdh5.6.1-package-changes.log	2018-02-19 16:31	4.00KB
flume-ng-1.6.0-cdh5.6.1-package-since-last-release-changes.log	2018-02-19 16:31	0B
flume-ng-1.6.0-cdh5.6.1-src.tar.gz	2018-02-19 16:31	2.00MB
flume-ng-1.6.0-cdh5.6.1.CHANGES.txt	2018-02-19 16:31	9.00KB
flume-ng-1.6.0-cdh5.6.1.package.CHANGES.txt	2018-02-19 16:31	9.00KB
flume-ng-1.6.0-cdh5.6.1.package.since.last.release.CHANGES.txt	2018-02-19 16:31	0B
flume-ng-1.6.0-cdh5.6.1.releasenotes.html	2018-02-19 16:31	4.00KB
flume-ng-1.6.0-cdh5.6.1.tar.gz	2018-02-19 16:31	62.00MB
flume-ng-1.6.0-cdh5.7.0-changes.log	2018-02-19 16:33	3.00KB
flume-ng-1.6.0-cdh5.7.0-package-changes.log	2018-02-19 16:33	4.00KB
flume-ng-1.6.0-cdh5.7.0-package-since-last-release-changes.log	2018-02-19 16:33	320B
flume-ng-1.6.0-cdh5.7.0-src.tar.gz	2018-02-19 16:33	2.00MB
flume-ng-1.6.0-cdh5.7.0.CHANGES.txt	2018-02-19 16:33	8.00KB
flume-ng-1.6.0-cdh5.7.0.package.CHANGES.txt	2018-02-19 16:33	11.00KB
flume-ng-1.6.0-cdh5.7.0.package.since.last.release.CHANGES.txt	2018-02-19 16:33	2.00KB
flume-ng-1.6.0-cdh5.7.0.releasenotes.html	2018-02-19 16:33	4.00KB
flume-ng-1.6.0-cdh5.7.0.tar.gz	2018-02-19 16:33	64.00MB
flume-ng-1.6.0-cdh5.7.1-changes.log	2018-02-19 16:34	4.00KB
flume-ng-1.6.0-cdh5.7.1-package-changes.log	2018-02-19 16:34	4.00KB
flume-ng-1.6.0-cdh5.7.1-package-since-last-release-changes.log	2018-02-19 16:34	0B
flume-ng-1.6.0-cdh5.7.1-src.tar.gz	2018-02-19 16:34	2.00MB
flume-ng-1.6.0-cdh5.7.1.CHANGES.txt	2018-02-19 16:34	10.00KB
flume-ng-1.6.0-cdh5.7.1.package.CHANGES.txt	2018-02-19 16:34	11.00KB
flume-ng-1.6.0-cdh5.7.1.package.since.last.release.CHANGES.txt	2018-02-19 16:34	0B
flume-ng-1.6.0-cdh5.7.1.releasenotes.html	2018-02-19 16:34	4.00KB
flume-ng-1.6.0-cdh5.7.1.tar.gz	2018-02-19 16:34	64.00MB
flume-ng-1.6.0-cdh5.7.2-changes.log	2018-02-19 16:35	4.00KB
flume-ng-1.6.0-cdh5.7.2-package-changes.log	2018-02-19 16:35	4.00KB
flume-ng-1.6.0-cdh5.7.2-package-since-last-release-changes.log	2018-02-19 16:35	0B
archive.cloudera.com/cdh5/cdh/5/flume-ng-1.6.0-cdh5.7.0.tar.gz	2018-02-19 16:35	2.00MB

下载 - Xftp 6 (Free for Home/School)

文件(F) 编辑(E) 查看(V) 命令(C) 工具(T) 窗口(W) 帮助(H)

sftp://192.168.1.18

本地目录: C:\Users\jungle\Downloads
远程目录: /home/jungle/software

名称	大小	类型	修改时间	属性
..		文件夹		
alan.png	23KB	PNG 文件	2019/7/18, 14:15	
FileZilla_3.42.1_win6...	7.56MB	应用程序	2019/7/4, 15:09	
FileZilla_3.40.0 win6...	7.53MB	应用程序	2019/7/4, 21:23	
flume-ng-1.6.0-cdh...	2.61MB	GZ 压缩文件	2019/7/25, 15:29	
flume-ng-1.6.0-cdh...	64.20MB	GZ 压缩文件	2019/7/25, 15:29	
incubator-echarts-4...	14.16MB	ZIP 压缩文件	2019/7/18, 19:00	
scala-2.11.0.tgz	24.80MB	TGZ 压缩...	2019/7/4, 22:00	
TortoiseGit-2.8.0.0-...	19.02MB	Windows ...	2019/7/12, 16:10	
TortoiseGit-Langua...	4.06MB	Windows ...	2019/7/12, 16:11	
weather_mini	1017 Bytes	文件	2019/7/7, 16:06	
Xftp-6.0.0119p.exe	30.60MB	应用程序	2019/7/9, 22:09	
Xshell-6.0.0125p.exe	42.55MB	应用程序	2019/7/9, 22:14	
zeppelin-0.8.1-bin-a...	946.98MB	TGZ 压缩...	2019/7/19, 17:33	
数据恢复企业版.zip	14.41MB	ZIP 压缩文件	2019/7/21, 18:19	

本地路径: C:\Users\jungle\Downloads
远程路径: /home/jungle/software

传输 日志

名称	状态	进度	大小	本地路径	远程路径	速度	估计剩余
flume-ng-1.6.0-cdh5.7.0.t...	进行中	46%	29.75MB/64.20MB	C:\Users\jungle\Downloads\flume...	192.168.1.18:/home/jungl...	9.80 MB/s	00:00:...

已连接 192.168.1.18:22。

• 解压

```
tar -zvxf flume-ng-1.6.0-cdh5.7.0.tar.gz -C ~/app/
```

- 配置环境变量

```
cd  
vi .bash_profile
```

```
export FLUME_HOME=/home/jungle/app/apache-flume-1.6.0-cdh5.7.0-bin  
export PATH=$FLUME_HOME/bin:$PATH
```

```
25 export PATH=$SPARK_HOME/bin:$PATH  
26  
27 export SCALA_HOME=/home/jungle/app/scala-2.11.0  
28 export PATH=$SCALA_HOME/bin:$PATH  
29  
30 export FLUME_HOME=/home/jungle/app/apache-flume-1.6.0-cdh5.7.0-bin  
31 export PATH=$FLUME_HOME/bin:$PATH  
32  
33 export PATH  
34  
.bash_profile [+] [utf-8] 31,34 Bot  
-- INSERT --
```

```
source .bash_profile
```

- 配置conf

```
cd $FLUME_HOME  
cd conf/
```

```
cp flume-env.sh.template flume-env.sh  
vi flume-env.sh
```

```
export JAVA_HOME=/home/jungle/app/jdk1.8.0_152
```

```
18 # during Flume startup.  
19  
20 # Environment variables can be set here.  
21  
22 # export JAVA_HOME=/usr/lib/jvm/java-6-sun  
23  
24 export JAVA_HOME=/home/jungle/app/jdk1.8.0_152  
25  
26  
27 # Give Flume more memory and pre-allocate, enable remote monitoring via JMX  
28 # export JAVA_OPTS="-Xms100m -Xmx2000m -Dcom.sun.management.jmxremote"  
29  
30 # Note that the Flume conf directory is always included in the classpath.  
31 #FLUME_CLASSPATH=""  
32  
flume-env.sh [+] [utf-8] 26,1 Bot  
-- INSERT --
```

```
cd $FLUME_HOME/bin  
flume-ng version
```

==检测==

```
-rwxr-xr-x 1 jungle jungle 13388 Mar 24 2016 flume-ng
-rw-r--r-- 1 jungle jungle    936 Mar 24 2016 flume-ng.cmd
-rwxr-xr-x 1 jungle jungle 14041 Mar 24 2016 flume-ng.ps1
jungle@centosserver1:[/home/jungle/app/apache-flume-1.6.0-cdh5.7.0-bin/bin]
jungle@centosserver1:[/home/jungle/app/apache-flume-1.6.0-cdh5.7.0-bin/bin]ll
total 36
-rwxr-xr-x 1 jungle jungle 13388 Mar 24 2016 flume-ng
-rw-r--r-- 1 jungle jungle    936 Mar 24 2016 flume-ng.cmd
-rwxr-xr-x 1 jungle jungle 14041 Mar 24 2016 flume-ng.ps1
jungle@centosserver1:[/home/jungle/app/apache-flume-1.6.0-cdh5.7.0-bin/bin]flume-ng vers
ion
Flume 1.6.0-cdh5.7.0
Source code repository: https://git-wip-us.apache.org/repos/asf/flume.git
Revision: 8f5f5143ae30802fe79f9ab96f893e6c54a105d1
Compiled by jenkins on Wed Mar 23 11:38:48 PDT 2016
From source with checksum 50b533f0ffc32db9246405ac4431872e
jungle@centosserver1:[/home/jungle/app/apache-flume-1.6.0-cdh5.7.0-bin/bin]
```

四、Flume实战

1.案例一

Flume实战

◆ 需求：从指定网络端口采集数据输出到控制台

使用Flume的关键就是写配置文件

- A) 配置Source
- B) 配置Channel
- C) 配置Sink
- D) 把以上三个组件串起来

a1: agent名称
r1: source的名称
k1: sink的名称
c1: channel的名称

```
# 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 = localhost
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

# Bind the source and sink to the channel
a1.sources.r1.channels = c1
a1.sinks.k1.channel = c1
```

- 配置

```
cd $FLUME_HOME/conf
vi example.conf
```

```
# 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 = localhost
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

# Bind the source and sink to the channel
a1.sources.r1.channels = c1
a1.sinks.k1.channel = c1
```

```

1 # Name the components on this agent
2 al.sources = r1
3 al.sinks = k1
4 al.channels = c1
5
6 # Describe/configure the source
7 al.sources.r1.type = netcat
8 al.sources.r1.bind = localhost
9 al.sources.r1.port = 44444
10
11 # Describe the sink
12 al.sinks.k1.type = logger
13
14 # Use a channel which buffers events in memory
15 al.channels.c1.type = memory
16
17 # Bind the source and sink to the channel
18 al.sources.r1.channels = c1
19 al.sinks.k1.channel = c1
~
```

- 启动agent

```

flume-ng agent \
--name a1 \
--conf $FLUME_HOME/conf \
--conf-file $FLUME_HOME/conf/example.conf \
-dflume.root.logger=INFO,console
```

==使用telnet进行测试== : telnet localhost 44444

另开一个终端

```
jungle@centosserver1:[/home/jungle/app]telnet localhost 44444
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
hello
OK
```

```
c1
019-07-25 16:03:19,456 (lifecycleSupervisor-1-0) [INFO - org.apache.flume.instrumentation.MonitoredCounterGroup.register(MonitoredCounterGroup.java:120)] Monitored counter group for type: CHANNEL, name: c1: Successfully registered new MBean.
019-07-25 16:03:19,466 (lifecycleSupervisor-1-0) [INFO - org.apache.flume.instrumentation.MonitoredCounterGroup.start(MonitoredCounterGroup.java:96)] Component type: CHANNEL, name: c1 started
019-07-25 16:03:19,468 (conf-file-poller-0) [INFO - org.apache.flume.node.Application.startAllComponents(Application.java:173)] Starting Sink
019-07-25 16:03:19,477 (conf-file-poller-0) [INFO - org.apache.flume.node.Application.startAllComponents(Application.java:184)] Starting Source
019-07-25 16:03:19,478 (lifecycleSupervisor-1-2) [INFO - org.apache.flume.source.NetcatSource.start(NetcatSource.java:155)] Source starting
019-07-25 16:03:19,488 (lifecycleSupervisor-1-2) [INFO - org.apache.flume.source.NetcatSource.start(NetcatSource.java:169)] Created serverSocket
sun.nio.ch.ServerSocketChannelImpl[/127.0.0.1:44444]
019-07-25 16:04:21,486 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.apache.flume.sink.LoggerSink.process(LoggerSink.java:94)] Event: { headers:{} body: 68 65 6C 6C 6F 0D hello. }
```

Event: { headers:{} body: 68 65 6C 6C 6F 0D hello. }
 Event是FLume数据传输的基本单元
 Event = 可选的header + byte array

2.案例二

Flume实战

- ◆ 需求：从指定网络端口采集数据输出到控制台
- ◆ 需求：监控一个文件实时采集新增的数据输出到控制台

==需求二==

Agent选型：exec source + memory channel + logger sink

```
cd $FLUME_HOME/conf  
vi exec-memory-logger.conf
```

```
# 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 /home/jungle/data/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  
  
# Bind the source and sink to the channel  
a1.sources.r1.channels = c1  
a1.sinks.k1.channel = c1
```

```

1 jungle18 ✘ 2 jungle18 ✘ + 
1 al.sources = r1
2 al.sinks = k1
3 al.channels = c1
4
5 # Describe/configure the source
6 al.sources.r1.type = exec
7 al.sources.r1.command = tail -F /home/jungle/data/data.log
8 al.sources.r1.shell = /bin/sh -c
9
10 # Describe the sink
11 al.sinks.k1.type = logger
12
13 # Use a channel which buffers events in memory
14 al.channels.c1.type = memory
15
16 # Bind the source and sink to the channel
17 al.sources.r1.channels = c1
18 al.sinks.k1.channel = c1

```

```

flume-ng agent \
--name a1 \
--conf $FLUME_HOME/conf \
--conf-file $FLUME_HOME/conf/exec-memory-logger.conf \
-Dflume.root.logger=INFO,console

```

==测试==

```

echo hello >>data.log
echo world >>data.log
echo world >>data.log

```

```

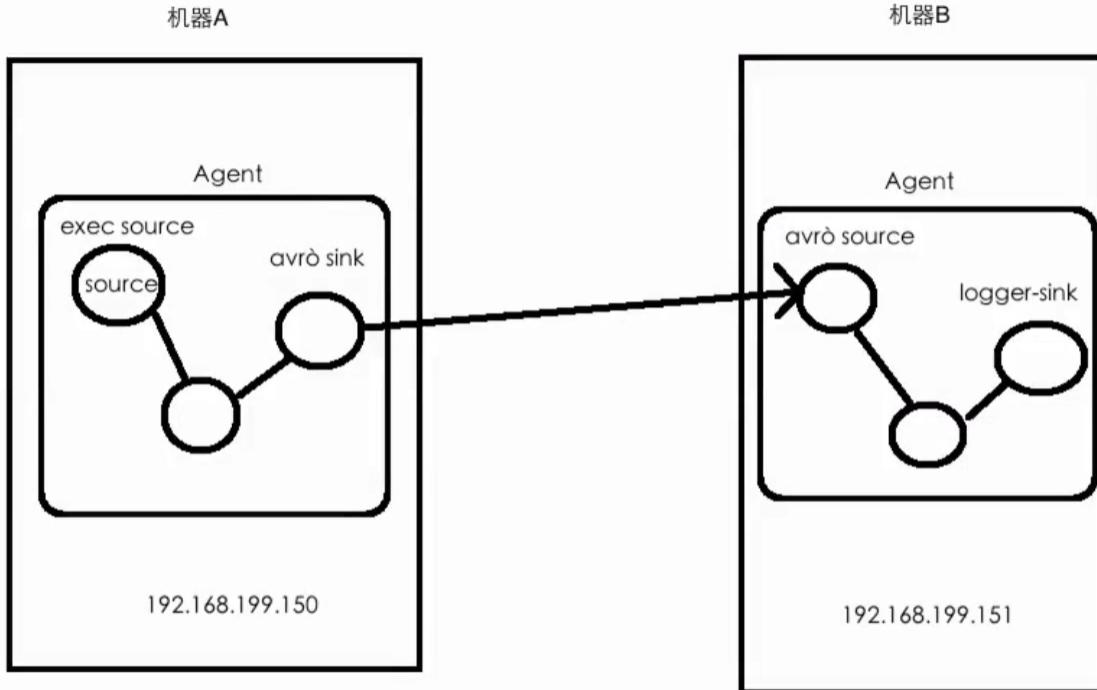
2019-07-25 18:18:54,430 (conf-file-poller-0) [INFO - org.apache.flume.sink.DefaultSinkFactory.create(DefaultSinkFactory.java:42)] Creating instance of sink: k1, type: logger
2019-07-25 18:18:54,433 (conf-file-poller-0) [INFO - org.apache.flume.node.AbstractConfigurationProvider.getConfiguration(AbstractConfigurationProvider.java:114)] Channel c1 connected to [r1, k1]
2019-07-25 18:18:54,441 (conf-file-poller-0) [INFO - org.apache.flume.node.Application.startAllComponents(Application.java:138)] Starting new configuration:{ sourceRunners:(r1=EventDrivenSourceRunner: { source:org.apache.flume.source.ExecSource{name:r1,state:IDLE} }) sinkRunners:(k1=SinkRunner: { policy:org.apache.flume.sink.DefaultSinkProcessor@3be0c0af counterGroup:{ name:null counters:{} } }) channels:{c1=org.apache.flume.channel.MemoryChannel{name: c1} } }
2019-07-25 18:18:54,456 (conf-file-poller-0) [INFO - org.apache.flume.node.Application.startAllComponents(Application.java:145)] Starting Channel c1
2019-07-25 18:18:54,612 (lifecycleSupervisor-1-0) [INFO - org.apache.flume.instrumentation.MonitoredCounterGroup.register(MonitoredCounterGroup.java:120)] Monitored counter group for type: CHANNEL, name: c1: Successfully registered new MBean.
2019-07-25 18:18:54,615 (lifecycleSupervisor-1-0) [INFO - org.apache.flume.instrumentation.MonitoredCounterGroup.start(MonitoredCounterGroup.java:96)] Component type: CHANNEL, name: c1 started
2019-07-25 18:18:54,629 (conf-file-poller-0) [INFO - org.apache.flume.node.Application.startAllComponents(Application.java:173)] Starting Sink k1
2019-07-25 18:18:54,631 (conf-file-poller-0) [INFO - org.apache.flume.node.Application.startAllComponents(Application.java:184)] Starting Source r1
2019-07-25 18:18:54,636 (lifecycleSupervisor-1-0) [INFO - org.apache.flume.source.ExecSource.start(ExecSource.java:169)] Exec source starting with command:tail -F /home/jungle/data/data.log
2019-07-25 18:18:54,655 (lifecycleSupervisor-1-0) [INFO - org.apache.flume.instrumentation.MonitoredCounterGroup.register(MonitoredCounterGroup.java:120)] Monitored counter group for type: SOURCE, name: r1: Successfully registered new MBean.
2019-07-25 18:18:54,655 (lifecycleSupervisor-1-0) [INFO - org.apache.flume.instrumentation.MonitoredCounterGroup.start(MonitoredCounterGroup.java:96)] Component type: SOURCE, name: r1 started
2019-07-25 18:18:58,657 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.apache.flume.sink.LoggerSink.process(LoggerSink.java:94)] Event: { headers:{} body: 68 65 6C 6C 6F } hello
2019-07-25 18:18:58,658 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.apache.flume.sink.LoggerSink.process(LoggerSink.java:94)] Event: { headers:{} body: 77 6F 72 6C 64 } world
2019-07-25 18:18:58,658 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.apache.flume.sink.LoggerSink.process(LoggerSink.java:94)] Event: { headers:{} body: 77 6F 72 6C 64 } world
2019-07-25 18:19:52,661 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.apache.flume.sink.LoggerSink.process(LoggerSink.java:94)] Event: { headers:{} body: 65 6C 6B } elk

```

3.案例三

Flume实战

- ◆ 需求：从指定网络端口采集数据输出到控制台
- ◆ 需求：监控一个文件实时采集新增的数据输出到控制台
- ◆ 需求：将A服务器上的日志实时采集到B服务器



日志收集过程：

- 1) 机器上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到access.log中
- 2) avrò sink把新产生的日志输出到对应的avrò source指定的hostname和port上
- 3) 通过avrò source对应的agent将我们的日志输出到控制台(Kafka)

需求三：

技术选型：

`exec source + memory channel + avro sink
avro source + memory channel + logger sink`

```
cd $FLUME_HOME/conf  
vi exec-memory-avro.conf
```

==exec-memory-avro.conf==

```
exec-memory-avro.sources = exec-source  
exec-memory-avro.sinks = avro-sink  
exec-memory-avro.channels = memory-channel  
  
exec-memory-avro.sources.exec-source.type = exec  
exec-memory-avro.sources.exec-source.command = tail -F  
/home/jungle/data/data.log  
exec-memory-avro.sources.exec-source.shell = /bin/sh -c
```

```
exec-memory-avro.sinks.avro-sink.type = avro
exec-memory-avro.sinks.avro-sink.hostname = centosserver1
exec-memory-avro.sinks.avro-sink.port = 44444

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

exec-memory-avro.sources.exec-source.channels = memory-channel
exec-memory-avro.sinks.avro-sink.channel = memory-channel
```

```
cd $FLUME_HOME/conf
vi avro-memory-logger.conf
```

==avro-memory-logger.conf==

```
avro-memory-logger.sources = avro-source
avro-memory-logger.sinks = logger-sink
avro-memory-logger.channels = memory-channel

avro-memory-logger.sources.avro-source.type = avro
avro-memory-logger.sources.avro-source.bind = centosserver1
avro-memory-logger.sources.avro-source.port = 44444

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

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

avro-memory-logger.sources.avro-source.channels = memory-channel
avro-memory-logger.sinks.logger-sink.channel = memory-channel
```

==先启动avro-memory-logger==

```
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
```

==然后启动exec-memory-avro==

```
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
```

==测试==

```
echo jungle >> data.log
```

第4章 分布式发布订阅消息系统Kafka

一、Kafka概述

[官网](#)

Kafka概述 和消息系统类似

更多课程请进

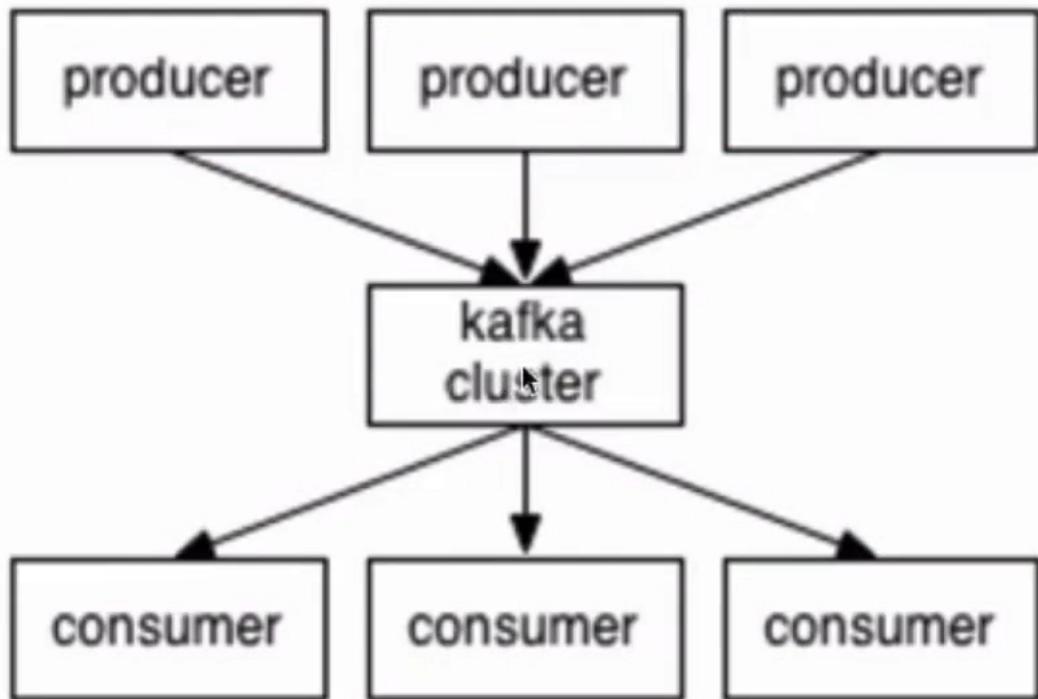
消息中间件：生产者和消费者

妈妈：生产者
你：消费者
馒头：数据流、消息

正常情况下： 生产一个 消费一个
其他情况：
一直生产，你吃到某一个馒头时，你卡住（机器故障），馒头就丢失了
一直生产，做馒头速度快，你吃来不及，馒头也就丢失了

拿个碗/篮子，馒头做好以后先放到篮子里，你要吃的时候去篮子里面取出来吃

篮子/框： Kafka
当篮子满了，馒头就装不下了，咋办？
多准备几个篮子 === Kafka的扩容



Kafka架构

producer：生产者，就是生产馒头（老妈）

consumer：消费者，就是吃馒头的（你）

broker：篮子

topic：主题，给馒头带一个标签，topic a 的馒头是给你吃的，topic b 的馒头是给你弟弟吃的

二、Kafka单节点单Broker部署

1. Zookeeper安装

- 下载

```
cd software  
wget http://archive.cloudera.com/cdh5/cdh/5/zookeeper-3.4.5-cdh5.7.0.tar.gz
```

- 解压

```
tar -zxvf zookeeper-3.4.5-cdh5.7.0.tar.gz -C ~/app/
```

- 配置环境变量

```
vi ~/.bash_profile
```

```
export ZK_HOME=/home/jungle/app/zookeeper-3.4.5-cdh5.7.0  
export PATH=$ZK_HOME/bin:$PATH
```

```
23  
24 export SPARK_HOME=/home/jungle/app/spark-2.1.0-bin-2.6.0-cdh5.7.0  
25 export PATH=$SPARK_HOME/bin:$PATH  
26  
27 export SCALA_HOME=/home/jungle/app/scala-2.11.0  
28 export PATH=$SCALA_HOME/bin:$PATH  
29  
30 export FLUME_HOME=/home/jungle/app/apache-flume-1.6.0-cdh5.7.0-bin  
31 export PATH=$FLUME_HOME/bin:$PATH  
32  
33 export ZK_HOME=/home/jungle/app/zookeeper-3.4.5-cdh5.7.0  
34 export PATH=$ZK_HOME/bin:$PATH  
35  
36 export PATH  
37
```

~/.bash_profile [+] [utf-8] 34,3
-- INSERT --

```
source ~/.bash_profile
```

- 配置文件

```
cd $ZK_HOME/conf  
cp zoo_sample.cfg zoo.cfg  
vi zoo.cfg
```

```
dataDir=/home/jungle/app/tmp/zk
```

```
5 initLimit=10
6 # The number of ticks that can pass between
7 # sending a request and getting an acknowledgement
8 syncLimit=5
9 # the directory where the snapshot is stored.
10 # do not use /tmp for storage, /tmp here is just
11 # example sakes.
12 dataDir=/home/jungle/app/tmp/zk
13 # the port at which the clients will connect
14 clientPort=2181
15 #
16 # Be sure to read the maintenance section of the
17 # administrator guide before turning on autopurge.
18 #
19 # http://zookeeper.apache.org/doc/current/zookeeperAdmin.html#sc_m
zoo.cfg [+]
-- INSERT --
```

- 启动

```
cd $ZK_HOME/bin
./zkServer.sh start
```

==验证==

```
jps
```

```
jungle@centosserver1:[/home/jungle/app/zookeeper-3.4.5-cdh5.7.0/bin]jps
34008 Jps
33884 QuorumPeerMain
```

- 连接

```
./zkCli.sh
```

```
2019-07-27 11:16:20,138 [myid:] - INFO  [main-SendThread(localhost:2181):ClientCnxn$Send
Thread@975] - Opening socket connection to server localhost/127.0.0.1:2181. Will not att
empt to authenticate using SASL (unknown error)
JLine support is enabled
2019-07-27 11:16:20,393 [myid:] - INFO  [main-SendThread(localhost:2181):ClientCnxn$Send
Thread@852] - Socket connection established, initiating session, client: /127.0.0.1:4506
2, server: localhost/127.0.0.1:2181
2019-07-27 11:16:20,451 [myid:] - INFO  [main-SendThread(localhost:2181):ClientCnxn$Send
Thread@1235] - Session establishment complete on server localhost/127.0.0.1:2181, sessio
nid = 0x16c316bde820000, negotiated timeout = 30000

WATCHER::
```

WatchedEvent state:SyncConnected type:None path:null

```
[zk: localhost:2181(CONNECTED) 0] ls /
[zookeeper]
[zk: localhost:2181(CONNECTED) 1]
```

2.单节点单broker的部署及使用

- 下载kafka

[下载地址](#)

The screenshot shows a web browser with three tabs open. The active tab is 'Apache Kafka' at kafka.apache.org/downloads. On the left, there's a sidebar with links like HOME, INTRODUCTION, QUICKSTART, USE CASES, DOCUMENTATION, PERFORMANCE, POWERED BY, PROJECT INFO, ECOSYSTEM, CLIENTS, EVENTS, CONTACT US, and APACHE. The 'USE CASES' link is underlined. The main content area has sections for '0.9.0.0' (with a red box around it) and '0.8.2.2'. Under '0.9.0.0', there's a 'Download' button (also with a red box). The '0.8.2.2' section also has a 'Download' button. A red box highlights the '0.9.0.0' section and the 'Download' button.

```
wget https://archive.apache.org/dist/kafka/0.9.0.0/kafka_2.11-0.9.0.0.tgz
```

- 解压

```
tar -zxvf kafka_2.11-0.9.0.0.tgz -C ~/app/
```

- 配置环境变量

```
vi ~/.bash_profile
```

```
export KAFKA_HOME=/home/jungle/app/kafka_2.11-0.9.0.0
export PATH=$KAFKA_HOME/bin:$PATH
```

```
26
27 export SCALA_HOME=/home/jungle/app/scala-2.11.0
28 export PATH=$SCALA_HOME/bin:$PATH
29
30 export FLUME_HOME=/home/jungle/app/apache-flume-1.6.0-cdh5.7.0-bin
31 export PATH=$FLUME_HOME/bin:$PATH
32
33 export ZK_HOME=/home/jungle/app/zookeeper-3.4.5-cdh5.7.0
34 export PATH=$ZK_HOME/bin:$PATH
35
36 export KAFKA_HOME=/home/jungle/app/kafka_2.11-0.9.0.0
37 export PATH=$KAFKA_HOME/bin:$PATH
38
39 export PATH
40
~/.bash_profile [+]                                [utf-8] 37,34      Bot
-- INSERT --
```

```
source ~/.bash_profile
```

- 修改配置文件

```
$KAFKA_HOME/config/server.properties
broker.id=0
listeners
host.name
log.dirs
zookeeper.connect
```

```
cd $KAFKA_HOME/config
vi server.properties
```

```
host.name=centosserver1
```

```
23
24 listeners=PLAINTEXT://:9092
25
26 # The port the socket server listens on
27 #port=9092
28
29 # Hostname the broker will bind to. If not set, the server will bind to all interfaces
30 host.name=centosserver1
31
32 # Hostname the broker will advertise to producers and consumers. If not set, it uses
33 # value for "host.name" if configured. Otherwise, it will use the value returned from
34 # java.net.InetAddress.getCanonicalHostName().
35 #advertised.host.name=<hostname routable by clients>
36
37 # The port to publish to ZooKeeper for clients to use. If this is not set,
server.properties [+] [utf-8] 30,24 20%
-- INSERT --
```

```
log.dirs=/home/jungle/app/tmp/kafka-logs
```

```
53 # The maximum size of a request that the socket server will accept (protection against
54 socket.request.max.bytes=104857600
55
56
57 ##### Log Basics #####
58
59 # A comma separated list of directories under which to store log files
60 log.dirs=/home/jungle/app/tmp/kafka-logs
61
62 # The default number of log partitions per topic. More partitions allow greater
63 # parallelism for consumption, but this will also result in more files across
64 # the brokers.
65 num.partitions=1
66
67 # The number of threads per data directory to be used for log recovery at startup and
server.properties [+] [utf-8] 60,41 48%
-- INSERT --
```

```
zookeeper.connect=centosserver1:2181
```

```

109 # By default the log cleaner is disabled and the log retention policy will default to
110 # If log.cleaner.enable=true is set the cleaner will be enabled and individual logs
111 log.cleaner.enable=false
112
113 ##### Zookeeper #####
114
115 # Zookeeper connection string (see zookeeper docs for details).
116 # This is a comma separated host:port pairs, each corresponding to a zk
117 # server. e.g. "127.0.0.1:3000,127.0.0.1:3001,127.0.0.1:3002".
118 # You can also append an optional chroot string to the urls to specify the
119 # root directory for all kafka znodes
120 zookeeper.connect=centosserver1:2181
121
122 # Timeout in ms for connecting to zookeeper
123 zookeeper.connection.timeout.ms=6000
server.properties [+] [utf-8] 120,37 Bot
-- INSERT --

```

- 启动kafka

==之前要先启动zookeeper==

```

cd $KAFKA_HOME/bin
kafka-server-start.sh $KAFKA_HOME/config/server.properties

```

```

jps
jps -m

```

```

jungle@centosserver1:[/home/jungle/app/zookeeper-3.4.5-cdh5.7.0/bin]jps
3456 Kafka
3650 Jps
33884 QuorumPeerMain
jungle@centosserver1:[/home/jungle/app/zookeeper-3.4.5-cdh5.7.0/bin]jps -m
3456 Kafka /home/jungle/app/kafka_2.11-0.9.0.0/config/server.properties
3714 Jps -m
33884 QuorumPeerMain /home/jungle/app/zookeeper-3.4.5-cdh5.7.0/bin/../conf/zoo.cfg
jungle@centosserver1:[/home/jungle/app/zookeeper-3.4.5-cdh5.7.0/bin]

```

- 创建topic

```

kafka-topics.sh --create --zookeeper centosserver1:2181 --replication-factor
1 --partitions 1 --topic hello_topic

```

```

33884 QuorumPeerMain
jungle@centosserver1:[/home/jungle/app/zookeeper-3.4.5-cdh5.7.0/conf]kafka-topics.sh --create --zookeeper centosserver1:2181 --replication-factor
1 --partitions 1 --topic hello_topic
WARNING: Due to limitations in metric names, topics with a period ('.') or underscore '_' could collide. To avoid issues it is best to use either, but not both.
Created topic "hello_topic".
jungle@centosserver1:[/home/jungle/app/zookeeper-3.4.5-cdh5.7.0/conf]

```

==验证==

```

# 查看所有topic
kafka-topics.sh --list --zookeeper centosserver1:2181

```

```

jungle@centosserver1:[/home/jungle/app/zookeeper-3.4.5-cdh5.7.0/conf]kafka-topics.sh --list --zookeeper centosserver1:2181
hello_topic
jungle@centosserver1:[/home/jungle]

```

- 生产消息

```

kafka-console-producer.sh --broker-list centosserver1:9092 --topic
hello_topic

```

- 消费消息

```
kafka-console-consumer.sh --zookeeper centosserver1:2181 --topic hello_topic  
--from-beginning
```

==from-beginning==:表示从头开始消费

```
kafka-server-start.sh $KAFKA_HOME/config/server.properties  
  
创建topic: zk  
kafka-topics.sh --create --zookeeper hadoop000:2181 --replication-factor  
  
查看所有topic  
kafka-topics.sh --list --zookeeper hadoop000:2181  
  
发送消息: broker  
kafka-console-producer.sh --broker-list hadoop000:9092 --topic hello_topi  
  
消费消息: zk  
kafka-console-consumer.sh --zookeeper hadoop000:2181 --topic hello_topic  
  
--from-beginning的使用
```

- 查看topic的详细信息

```
# 所有的topic  
kafka-topics.sh --describe --zookeeper centosserver1:2181
```

```
# 指定的topic  
kafka-topics.sh --describe --zookeeper centosserver1:2181 --topic  
hello_topic
```

3.单节点多broker部署及使用

- 复制配置文件

```
cd $KAFKA_HOME/config  
cp server.properties server-1.properties  
cp server.properties server-2.properties  
cp server.properties server-3.properties
```

- 修改配置文件

==server-1.properties==

```
vi server-1.properties
```

```
broker.id=1  
listeners=PLAINTEXT://:9093  
log.dirs=/home/jungle/app/tmp/kafka-logs-1
```

==server-2.properties==

```
vi server-2.properties
```

```
broker.id=2  
listeners=PLAINTEXT://:9094  
log.dirs=/home/jungle/app/tmp/kafka-logs-2
```

==server-3.properties==

```
vi server-3.properties
```

```
broker.id=3  
listeners=PLAINTEXT://:9095  
log.dirs=/home/jungle/app/tmp/kafka-logs-3
```

- 启动

==daemon== :后台启动

```
kafka-server-start.sh -daemon $KAFKA_HOME/config/server-1.properties &  
kafka-server-start.sh -daemon $KAFKA_HOME/config/server-2.properties &  
kafka-server-start.sh -daemon $KAFKA_HOME/config/server-3.properties &
```

```
jps -m
```

```
jungle@centosserver1:[/home/jungle/app/kafka_2.11-0.9.0.0/config]jps -m  
11428 Kafka /home/jungle/app/kafka_2.11-0.9.0.0/config/server-3.properties  
11211 Kafka /home/jungle/app/kafka_2.11-0.9.0.0/config/server-2.properties  
33884 QuorumPeerMain /home/jungle/app/zookeeper-3.4.5-cdh5.7.0/bin/../conf/zoo.cfg  
11212 Kafka /home/jungle/app/kafka_2.11-0.9.0.0/config/server-1.properties  
11868 Jps -m  
jungle@centosserver1:[/home/jungle/app/kafka_2.11-0.9.0.0/config]
```

- 创建topic

```
kafka-topics.sh --create --zookeeper centosserver1:2181 --replication-factor  
3 --partitions 1 --topic my-replicated-topic
```

- 查看topic

```
# 查看所有topic  
kafka-topics.sh --list --zookeeper centosserver1:2181
```

```
jungle@centosserver1:[/home/jungle/app/kafka_2.11-0.9.0.0/config]kafka-topics.sh --list  
--zookeeper centosserver1:2181  
hello_topic  
my-replicated-topic  
jungle@centosserver1:[/home/jungle/app/kafka_2.11-0.9.0.0/config]
```

- 生产消息

```
kafka-console-producer.sh --broker-list  
centosserver1:9093,centosserver1:9094,centosserver1:9095 --topic my-  
replicated-topic
```

- 消费消息

```
kafka-console-consumer.sh --zookeeper centosserver1:2181 --topic my-replicated-topic
```

==可以开启多个==、

三、Kafka容错性测试

- 查看指定topic的详细信息

```
kafka-topics.sh --describe --zookeeper centosserver1:2181 --topic my-replicated-topic
```

```
jungle@centosserver1:[/home/jungle/app/kafka_2.11-0.9.0.0/config] kafka-topics.sh --describe --zookeeper centosserver1:2181 --topic my-replicated-topic
Topic:my-replicated-topic      PartitionCount:1      ReplicationFactor:3      Configs:
          Topic: my-replicated-topic      Partition: 0      Leader: 2      Replicas: 2,1,3
sr: 2,1,3
jungle@centosserver1:[/home/jungle/app/kafka_2.11-0.9.0.0/config]
```

主节点

==2为主节点==

- 干掉主节点

```
jps -m
```

```
jps -m
jungle@centosserver1:[/home/jungle/app/kafka_2.11-0.9.0.0/config] jps -m
13217 ConsoleConsumer --zookeeper centosserver1:2181 --topic my-replicated-topic
14386 Jps -m
13027 ConsoleProducer --broker-list centosserver1:9093,centosserver1:9094,centosserver1:9095 --topic my-replicated-topic
11428 Kafka /home/jungle/app/kafka_2.11-0.9.0.0/config/server-3.properties
11211 Kafka /home/jungle/app/kafka_2.11-0.9.0.0/config/server-2.properties
33884 QuorumPeerMain /home/jungle/app/zookeeper-3.4.5-cdh5.7.0/bin/../conf/zoo.cfg
11212 Kafka /home/jungle/app/kafka_2.11-0.9.0.0/config/server-1.properties
13404 ConsoleConsumer --zookeeper centosserver1:2181 --topic my-replicated-topic
jungle@centosserver1:[/home/jungle/app/kafka_2.11-0.9.0.0/config]
```

```
kill -9 11211
```

```
jps -m
```

```
jungle@centosserver1:[/home/jungle/app/kafka_2.11-0.9.0.0/config] kill -9 11211
jungle@centosserver1:[/home/jungle/app/kafka_2.11-0.9.0.0/config] jps -m
13217 ConsoleConsumer --zookeeper centosserver1:2181 --topic my-replicated-topic
14514 Jps -m
13027 ConsoleProducer --broker-list centosserver1:9093,centosserver1:9094,centosserver1:9095 --topic my-replicated-topic
11428 Kafka /home/jungle/app/kafka_2.11-0.9.0.0/config/server-3.properties
33884 QuorumPeerMain /home/jungle/app/zookeeper-3.4.5-cdh5.7.0/bin/../conf/zoo.cfg
11212 Kafka /home/jungle/app/kafka_2.11-0.9.0.0/config/server-1.properties
13404 ConsoleConsumer --zookeeper centosserver1:2181 --topic my-replicated-topic
jungle@centosserver1:[/home/jungle/app/kafka_2.11-0.9.0.0/config]
```

==结果==

还是一样可以运行的

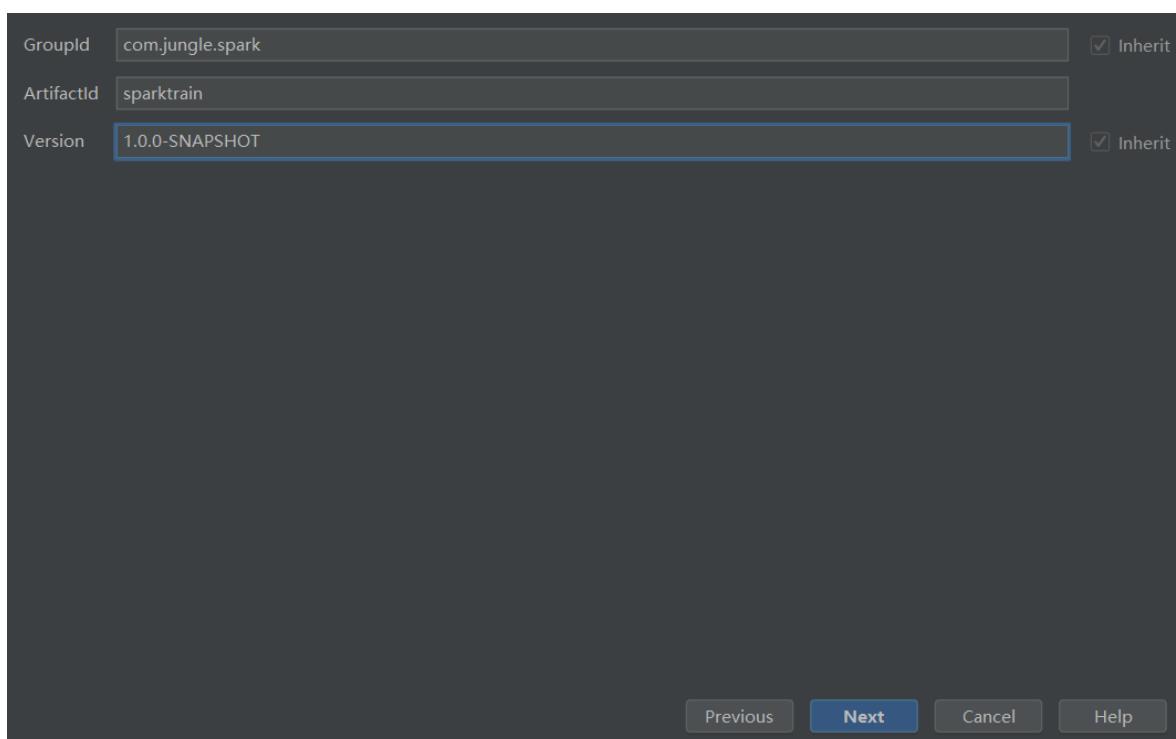
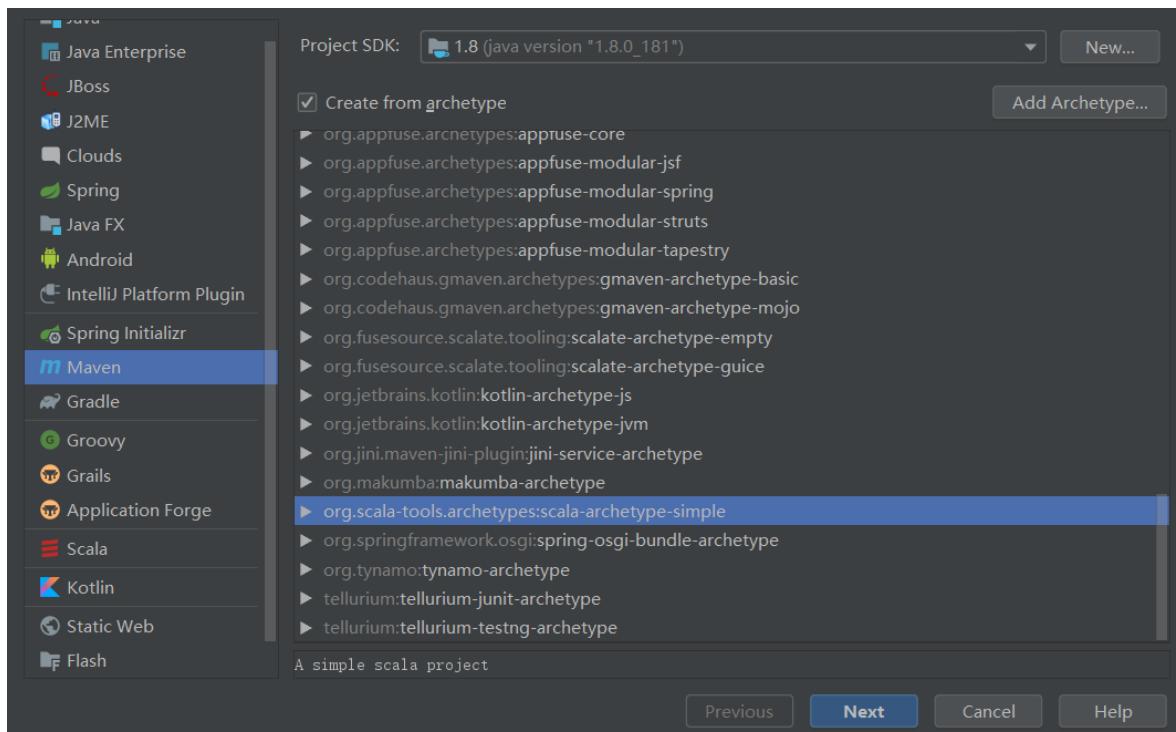
```
kafka-topics.sh --describe --zookeeper centosserver1:2181 --topic my-replicated-topic
```

```
jungle@centosserver1:[/home/jungle/app/kafka_2.11-0.9.0.0/config]kafka-topics.sh --descr
ibe --zookeeper centosserver1:2181 --topic my-replicated-topic
Topic:my-replicated-topic      PartitionCount:1      ReplicationFactor:3      Configs:
          Topic: my-replicated-topic      Partition: 0      Leader: 1      Replicas: 2,1,3I
sr: 1,3
jungle@centosserver1:[/home/jungle/app/kafka_2.11-0.9.0.0/config]
```

==1变成了主节点==

四、使用IDEA+Maven构建开发环境

- 新建项目

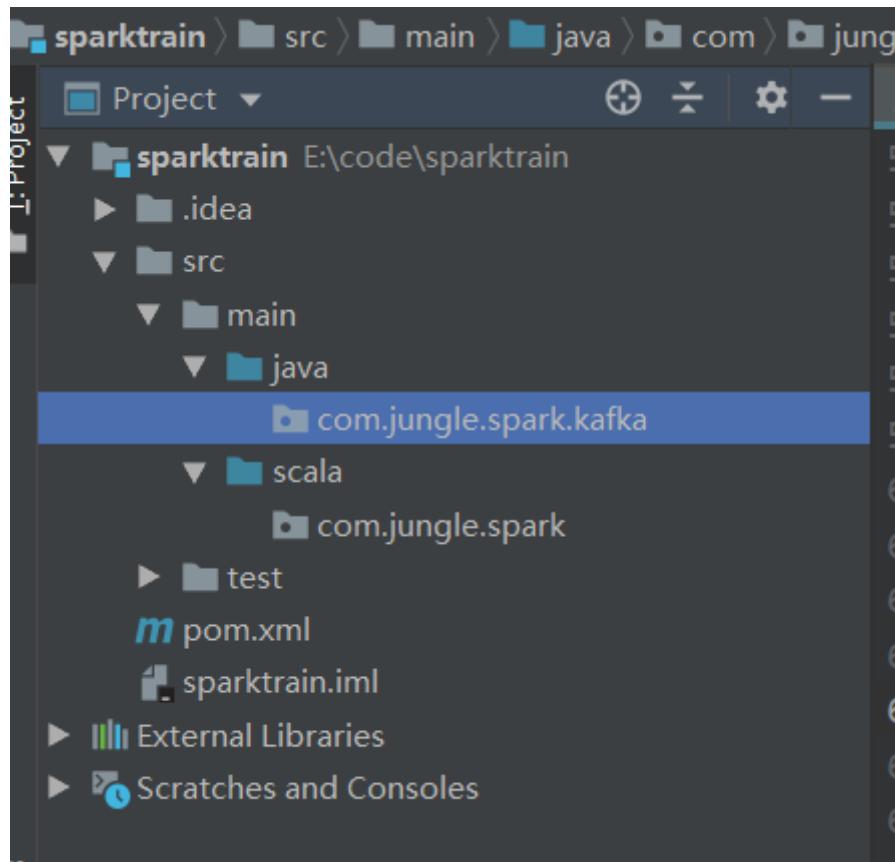


- 修改pom.xml文件

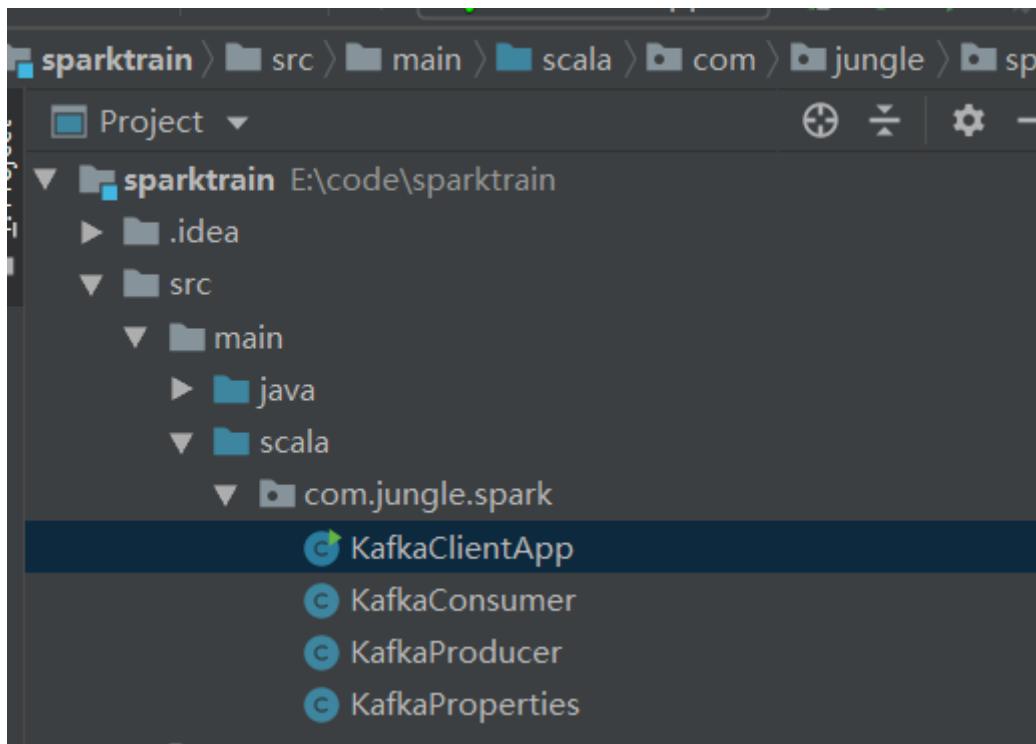
```
<properties>
  <scala.version>2.11.8</scala.version>
</properties>
```

- 引入依赖

```
<!-- https://mvnrepository.com/artifact/org.apache.kafka/kafka -->
<dependency>
  <groupId>org.apache.kafka</groupId>
  <artifactId>kafka_2.11</artifactId>
  <version>0.9.0.0</version>
</dependency>
```



五、Kafka Java API编程



1. Producer

(1) 提供端

- Kafka常用配置文件

```
package com.jungle.spark;

/**
 * Kafka常用配置文件
 */
public class KafkaProperties {

    public static final String ZK = "192.168.1.18:2181";

    public static final String TOPIC = "hello_topic";

    public static final String BROKER_LIST = "192.168.1.18:9092";

    public static final String GROUP_ID = "test_group1";

}
```

- Kafka生产者

```
package com.jungle.spark;

import kafka.javaapi.producer.Producer;
import kafka.producer.KeyedMessage;
import kafka.producer.ProducerConfig;

import java.util.Properties;

/**
```

```

 * Kafka生产者
 */
public class KafkaProducer extends Thread{

    private String topic;

    private Producer<Integer, String> producer;

    public KafkaProducer(String topic) {
        this.topic = topic;

        Properties properties = new Properties();

        properties.put("metadata.broker.list", KafkaProperties.BROKER_LIST);
        properties.put("serializer.class", "kafka.serializer.StringEncoder");
        properties.put("request.required.acks", "1");

        producer = new Producer<Integer, String>(new
ProducerConfig(properties));
    }

    @Override
    public void run() {

        int messageNo = 1;

        while(true) {
            String message = "message_" + messageNo;
            producer.send(new KeyedMessage<Integer, String>(topic,
message));
            System.out.println("Sent: " + message);

            messageNo++;
            try{
                Thread.sleep(2000);
            } catch (Exception e){
                e.printStackTrace();
            }
        }
    }
}

```

- Kafka Java API测试

```

package com.jungle.spark;

/**
 * Kafka Java API测试
 */
public class KafkaClientApp {

    public static void main(String[] args) {
        new KafkaProducer(KafkaProperties.TOPIC).start();
    }
}

```

```
//      new KafkaConsumer(KafkaProperties.TOPIC).start();  
}  
}
```

(2)消费端

- 开启zookeeper

```
cd $ZK_HOME/bin  
.zkServer.sh start
```

- 开启kafka

```
cd $KAFKA_HOME/bin  
kafka-server-start.sh $KAFKA_HOME/config/server.properties
```

- 消费者

```
kafka-console-consumer.sh --zookeeper centosserver1:2181 --topic hello_topic
```

```
jungle@centosserver1:[/home/jungle]  
jungle@centosserver1:[/home/jungle] kafka-console-consumer.sh --zookeeper centosserver1:2181 --topic hello_topic
```

(3)测试

启动KafkaClientApp

==遇到报错==

```
Exception in thread "Thread-0" kafka.common.FailedToSendMessageException:  
Failed to send messages after 3 tries.
```

==解决==

可能是防火墙的原因

查看防火墙状态

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

停掉防火墙

```
systemctl stop firewalld.service
```

禁止开机启动防火墙

```
systemctl disable firewalld.service
```

- 解决后

The screenshot shows a terminal window with the following output:

```
"C:\Program Files (x86)\Java\jdk1.8.0_181\bin\java.exe" ...
log4j:WARN No appenders could be found for logger (kafka.utils.VerifiableProperties).
log4j:WARN Please initialize the log4j system properly.
Sent: message_1
Sent: message_2
Sent: message_3
```

Below the terminal window, the command used is:

```
jungle@centosserver1:[/home/jungle] kafka-console-consumer.sh --zookeeper centosserver1:2181 --topic hello_topic
```

The consumer output shows messages from 1 to 11 being consumed.

2.Consumer

- Kafka消费者

```
package com.jungle.spark;

import kafka.consumer.Consumer;
import kafka.consumer.ConsumerConfig;
import kafka.consumer.ConsumerIterator;
import kafka.consumer.KafkaStream;
import kafka.javaapi.consumer.ConsumerConnector;

import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Properties;

/**
 * Kafka消费者
 */
public class KafkaConsumer extends Thread{

    private String topic;

    public KafkaConsumer(String topic) {
        this.topic = topic;
    }

    private ConsumerConnector createConnector(){
        Properties properties = new Properties();
        properties.put("zookeeper.connect", KafkaProperties.ZK);
        properties.put("group.id", KafkaProperties.GROUP_ID);
        return Consumer.createJavaConsumerConnector(new
ConsumerConfig(properties));
    }

    @Override
```

```

public void run() {
    ConsumerConnector consumer = createConnector();

    Map<String, Integer> topicCountMap = new HashMap<String, Integer>();
    topicCountMap.put(topic, 1);
    //      topicCountMap.put(topic2, 1);
    //      topicCountMap.put(topic3, 1);

    // String: topic
    // List<KafkaStream<byte[], byte[]>> 对应的数据流
    Map<String, List<KafkaStream<byte[], byte[]>>> messageStream =
    consumer.createMessageStreams(topicCountMap);

    KafkaStream<byte[], byte[]> stream =
    messageStream.get(topic).get(0); //获取我们每次接收到的数据

    ConsumerIterator<byte[], byte[]> iterator = stream.iterator();

    while (iterator.hasNext()) {
        String message = new String(iterator.next().message());
        System.out.println("rec: " + message);
    }
}
}

```

- Kafka Java API测试

```

package com.jungle.spark;

/**
 * Kafka Java API测试
 */
public class KafkaClientApp {

    public static void main(String[] args) {
        new KafkaProducer(KafkaProperties.TOPIC).start();

        new KafkaConsumer(KafkaProperties.TOPIC).start();

    }
}

```

启动KafkaClientApp

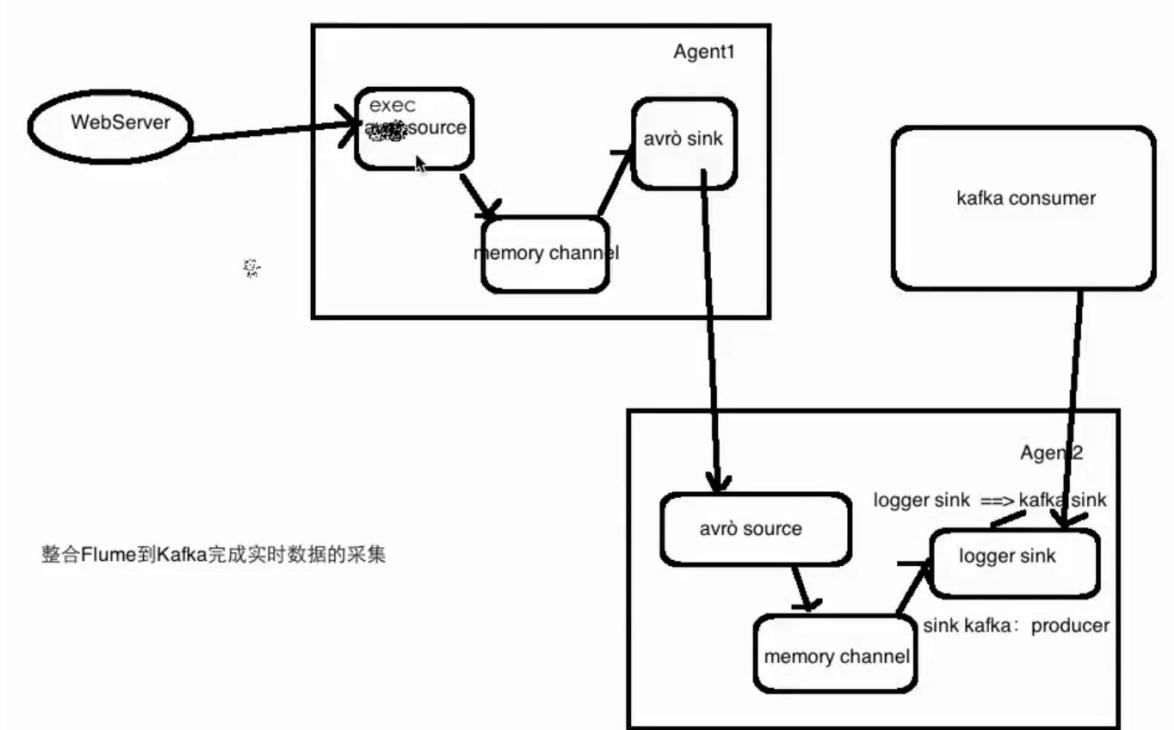
==结果==

```
Run: KafkaClientApp x
      Sent: message_148
      Sent: message_149
      rec: message_149
      Sent: message_150
      rec: message_150
      Sent: message_151
      rec: message_151
      rec: message_152
      Sent: message_152
      Sent: message_153
      rec: message_153

Terminal 0: Messages Find Run 5: D
Build completed successfully with 3 warnings in 3 s 948 ms (5 m)
```

六、整合Flume和Kafka完成实时数据采集

1.架构图



2.修改flume相关文件

```
cd /home/jungle/app/apache-flume-1.6.0-cdh5.7.0-bin/conf
vi avro-memory-kafka.conf
```

```

avro-memory-kafka.sources = avro-source

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

avro-memory-kafka.sources.avro-source.type = avro
avro-memory-kafka.sources.avro-source.bind = centosserver1
avro-memory-kafka.sources.avro-source.port = 44444

avro-memory-kafka.sinks.kafka-sink.type = org.apache.flume.sink.kafka.KafkaSink
avro-memory-kafka.sinks.kafka-sink.brokerList = centosserver1:9092
avro-memory-kafka.sinks.kafka-sink.topic=hello_topic
avro-memory-kafka.sinks.kafka-sink.batchSize=5
avro-memory-kafka.sinks.kafka-sink.requiredAcks=1

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

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

```

```

1 jungle18 ✘ | ● 2 jungle18 ✘ | ● 3 jungle18 ✘ | ● 4 jungle18 ✘ || +|
1 avro-memory-kafka.sources = avro-source
2 avro-memory-kafka.sinks = kafka-sink
3 avro-memory-kafka.channels = memory-channel
4
5 avro-memory-kafka.sources.avro-source.type = avro
6 avro-memory-kafka.sources.avro-source.bind = centosserver1
7 avro-memory-kafka.sources.avro-source.port = 44444
8
9 avro-memory-kafka.sinks.kafka-sink.type = org.apache.flume.sink.kafka.KafkaSink
10 avro-memory-kafka.sinks.kafka-sink.brokerList = centosserver1:9092
11 avro-memory-kafka.sinks.kafka-sink.topic=hello_topic
12 avro-memory-kafka.sinks.kafka-sink.batchSize=5
13 avro-memory-kafka.sinks.kafka-sink.requiredAcks=1
14
15 avro-memory-kafka.channels.memory-channel.type = memory
16
17 avro-memory-kafka.sources.avro-source.channels = memory-channel
18 avro-memory-kafka.sinks.kafka-sink.channel = memory-channel

```

3.启动

1. 启动zookeeper , Kafka
2. 启动avro-memory-kafka

```

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

```

3. 启动exec-memory-avro

```

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

```

4. 测试

```
jps -m
```

```
jungle@centosserver1:[/home/jungle]jps -m
38482 Kafka /home/jungle/app/kafka_2.11-0.9.0.0/config/server.properties
46952 Jps -m
33884 QuorumPeerMain /home/jungle/app/zookeeper-3.4.5-cdh5.7.0/bin/../conf/zoo.cfg
45981 Application --name avro-memory-kafka --conf-file /home/jungle/app/apache-flume-1.6.0-cdh5.7.0-bin/conf/avro-memory-kafka.conf
46478 Application --name exec-memory-avro --conf-file /home/jungle/app/apache-flume-1.6.0-cdh5.7.0-bin/conf/exec-memory-avro.conf
jungle@centosserver1:[/home/jungle]
```

5. kafka消费端

```
kafka-console-consumer.sh --zookeeper centosserver1:2181 --topic hello_topic
```

6. 测试传输

```
cd data
echo hellospark >>data.log
echo hellospark1 >>data.log
echo hellospark2 >>data.log
```

```
-rw-rw-r-- 1 jungle jungle 1104 Jul 12 14:19 wrk.lua
jungle@centosserver1:[/home/jungle/data]echo hellospark >>data.log
jungle@centosserver1:[/home/jungle/data]echo hellospark1 >>data.log
jungle@centosserver1:[/home/jungle/data]echo hellospark2 >>data.log
jungle@centosserver1:[/home/jungle/data]
```

```
jungle@centosserver1:[/home/jungle]
jungle@centosserver1:[/home/jungle]jps -m
38482 Kafka /home/jungle/app/kafka_2.11-0.9.0.0/config/server.properties
46952 Jps -m
33884 QuorumPeerMain /home/jungle/app/zookeeper-3.4.5-cdh5.7.0/bin/../conf/zoo.cfg
45981 Application --name avro-memory-kafka --conf-file /home/jungle/app/apache-flume-1.6
46478 Application --name exec-memory-avro --conf-file /home/jungle/app/apache-flume-1.6.
jungle@centosserver1:[/home/jungle]kafka-console-consumer.sh --zookeeper centosserver1:2
hellospark
hellospark1
hellospark2

```

第5章 实战环境搭建

一、安装Scala、maven、hadoop

更改maven本地仓库

```
cd $MAVEN_HOME/conf
cd settings.xml
```

```
<localRepository>/home/jungle/maven_repos/</localRepository>
```

```
45 |-->
46 <settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
47   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
48   xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0 http://maven.ap
49 <!-- localRepository
50   | The path to the local repository maven will use to store artifacts.
51   |
52   | Default: ${user.home}/.m2/repository
53 <localRepository>/path/to/local/repo</localRepository>
54 >
55 <localRepository>/home/jungle/maven_repos/</localRepository>
56
57 <!-- interactiveMode
58   | This will determine whether maven prompts you when it needs input. If set to fa
59   | maven will use a sensible default value, perhaps based on some other setting, f
60   | the parameter in question.
61   |
62   | Default: true
settings.xml [+] [utf-8] 55,43 18%
```

二、HBase安装

- 下载

[地址](#)

```
 wget http://archive.cloudera.com/cdh5/cdh/5/hbase-1.2.0-cdh5.7.0.tar.gz
```

- 解压

```
 tar -zxvf hbase-1.2.0-cdh5.7.0.tar.gz -C ~/app/
```

- 配置环境变量

```
 vi ~/.bash_profile
```

```
 export HBASE_HOME=/home/jungle/app/hbase-1.2.0-cdh5.7.0
export PATH=$HBASE_HOME/bin:$PATH
```

```
34 export PATH=$ZK_HOME/bin:$PATH
35
36 export KAFKA_HOME=/home/jungle/app/kafka_2.11-0.9.0.0
37 export PATH=$KAFKA_HOME/bin:$PATH
38
39 export HBASE_HOME=/home/jungle/app/hbase-1.2.0-cdh5.7.0
40 export PATH=$HBASE_HOME/bin:$PATH
41
42 export PATH
43
~/.bash_profile [+] -- INSERT --
```

```
 source ~/.bash_profile
```

- 配置文件

```
 cd $HBASE_HOME/conf
```

```
vi hbase-env.sh
```

```
export JAVA_HOME=/home/jungle/app/jdk1.8.0_152
```

```
24 # into the startup scripts (bin/hbase, etc.)  
25  
26 # The java implementation to use. Java 1.7+ required.  
27 # export JAVA_HOME=/usr/java/jdk1.6.0/  
28 export JAVA_HOME=/home/jungle/app/jdk1.8.0_152  
29 # Extra Java CLASSPATH elements. Optional.  
30 # export HBASE_CLASSPATH=  
31  
32 # The maximum amount of heap to use. Default is left to JVM default.  
33 # export HBASE_HEAPSIZE=1G  
hbase-env.sh [+] [utf-8] 28,47  
-- INSERT --
```

```
export HBASE_MANAGES_ZK=false
```

```
124 # otherwise arrive faster than the master can service them.  
125 # export HBASE_SLAVE_SLEEP=0.1  
126  
127 # Tell HBase whether it should manage its own instance of Zookeeper or not.  
128 export HBASE_MANAGES_ZK=false  
129  
130 # The default log rolling policy is RFA, where the log file is rolled as per the size  
131 # RFA appender. Please refer to the log4j.properties file to see more details on this.  
132 # In case one needs to do log rolling on a date change, one should set the environment  
133 # HBASE_ROOT_LOGGER to "<DESIRED_LOG_LEVEL>,DRFA".  
hbase-env.sh [+] [utf-8] 128,31 96%  
-- INSERT --
```

```
vi hbase-site.xml
```

```
<property>  
<name>hbase.rootdir</name>  
<value>hdfs://centosserver1:8020/hbase</value>  
</property>  
  
<property>  
<name>hbase.cluster.distributed</name>  
<value>true</value>  
</property>  
  
<property>  
<name>hbase.zookeeper.quorum</name>  
<value>centosserver1:2181</value>  
</property>
```

```
15  *
16  * Unless required by applicable law or agreed to in writing, software
17  * distributed under the License is distributed on an "AS IS" BASIS,
18  * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
19  * See the License for the specific language governing permissions and
20  * limitations under the License.
21  */
22 -->
23 <configuration>
24   <property>
25     <name>hbase.rootdir</name>
26     <value>hdfs://centosserver1:8020/hbase</value>
27   </property>
28
29   <property>
30     <name>hbase.cluster.distributed</name>
31     <value>true</value>
32   </property>
33
34   <property>
35     <name>hbase.zookeeper.quorum</name>
36     <value>centosserver1:2181</value>
37   </property>
38
39
40 </configuration>
~
~
~
hbase-site.xml [+]
-- INSERT --
```

```
vi regionservers
```

```
centosserver1
```

```
1 centosserver1
```

- 运行

```
==先启动zookeeper==
```

```
cd $HBASE_HOME/bin
./start-hbase.sh
```

```
jps
```

```
jungle@centosserver1:[/home/jungle/app/hbase-1.2.0-cdh5.7.0/bin]jps
41937 NameNode
42115 DataNode
42757 ResourceManager
42902 NodeManager
43897 HMaster
31451 HRegionServer
33884 QuorumPeerMain
44284 Jps
42382 SecondaryNameNode
jungle@centosserver1:[/home/jungle/app/hbase-1.2.0-cdh5.7.0/bin]
```

<http://192.168.1.18:60010>

The screenshot shows the Apache HBase master status interface. At the top, there are two tabs: '192.168.1.18:8888/weather-co' and 'spring-cloud-config/weather'. Below the tabs, there's a toolbar with various icons and links. The main content area is titled 'Master centosserver1'. It displays 'Region Servers' and 'Backup Masters' sections, each with tables showing server details like Start time, Version, Requests Per Second, and Num. Regions. A performance monitoring section on the right shows CPU usage at 0.2% and memory at 63%.

==启动命令行==

```
cd $HBASE_HOME/bin  
./hbase shell
```

```
SLF4J: Class path contains multiple SLF4J bindings.  
SLF4J: Found binding in [jar:file:/home/jungle/app/hbase-1.2.0-cdh5.7.0/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]  
SLF4J: Found binding in [jar:file:/home/jungle/app/hadoop-2.6.0-cdh5.7.0/share/hadoop/common/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]  
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.  
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]  
HBase Shell; enter 'help<RETURN>' for list of supported commands.  
Type "exit<RETURN>" to leave the HBase Shell  
Version 1.2.0-cdh5.7.0, rUnknown, Wed Mar 23 11:46:29 PDT 2016  
hbase(main):001:0> █
```

```
version  
status
```

```
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'
```

```
hbase(main):005:0> create 'member','info','address'  
0 row(s) in 2.3180 seconds  
=> Hbase::Table - member  
hbase(main):006:0> █
```

```
list
```

```
hbase (main):006:0> list
TABLE
member
1 row(s) in 0.0350 seconds

=> ["member"]
hbase (main):007:0>
```

```
describe 'member'
```

```
hbase(main):003:0> describe 'member'
Table member is ENABLED
member
COLUMN FAMILIES DESCRIPTION
{NAME => 'address', BLOOMFILTER => 'ROW', VERSIONS => '1', IN_MEMORY => 'false', KEEP_DELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING => 'NONE', TTL => 'FOREVER', COMPRESSION => 'NONE', MIN_VERSIONS => '0', BLOCKCACHE => 'true', BLOCKSIZE => '65536', REPLICATION_SCOPE => '0'}
{NAME => 'info', BLOOMFILTER => 'ROW', VERSIONS => '1', IN_MEMORY => 'false', KEEP_DELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING => 'NONE', TTL => 'FOREVER', COMPRESSION => 'NONE', MIN_VERSIONS => '0', BLOCKCACHE => 'true', BLOCKSIZE => '65536', REPLICATION_SCOPE => '0'}
2 row(s) in 0.2940 seconds
```

Namespace	Table Name	Online Regions	Offline Regions	Failed Regions	Split Regions	Other Regions	Description
default	member	1	0	0	0	0	'member', {NAME => 'address'}, {NAME => 'info'}

二、spark安装

- 启动

```
spark-shell --master local[2] --driver-class-path /home/jungle/app/hive-1.1.0-cdh5.7.0/lib/mysql-connector-java-5.1.27-bin.jar
```

三、开发环境搭建

使用IDEA整合 Maven搭建 Spark Streaming开发环境

==在sparktrain中==

◆pom.xml中添加对应的依赖

```
<properties>
    <scala.version>2.11.8</scala.version>
    <kafka.version>0.9.0.0</kafka.version>
    <spark.version>2.2.0</spark.version>
    <hadoop.version>2.6.0-cdh5.7.0</hadoop.version>
    <hbase.version>1.2.0-cdh5.7.0</hbase.version>
</properties>

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

<dependency>
    <groupId>org.apache.hadoop</groupId>
    <artifactId>hadoop-client</artifactId>
    <version>${hadoop.version}</version>
</dependency>
<dependency>
    <groupId>org.apache.hbase</groupId>
    <artifactId>hbase-client</artifactId>
    <version>${hbase.version}</version>
</dependency>
<dependency>
    <groupId>org.apache.hbase</groupId>
    <artifactId>hbase-server</artifactId>
    <version>${hbase.version}</version>
</dependency>
<dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-streaming_2.11</artifactId>
    <version>${spark.version}</version>
</dependency>
```

[spark streaming的依赖](#)

Apache Spark is a fast and general-purpose engine that supports general execution processing, **MLlib** for machine learning, and **GraphX** (Graph Processing).

Downloading

Get Spark from the [downloads page](#) or [HDFS and YARN](#). Downloads are pre-packaged for a range of popular Hadoop versions. Users can also download a “Hadoop free” binary and run Spark with any Hadoop version by [augmenting Spark’s classpath](#). Scala and Java users can include Spark in their projects using its maven coordinates and in the future Python users can also install Spark from PyPI.

If you’d like to build Spark from source, visit [Building Spark](#).

Spark runs on both Windows and UNIX-like systems (e.g. Linux, Mac OS). It’s easy to run locally on one machine — all you need is to have `java` installed on your system `PATH`, or the `JAVA_HOME` environment variable pointing to a Java installation.

Spark runs on Java 7+, Python 2.6+/3.4+ and R 3.1+. For the Scala API, Spark 2.1.0 uses Scala 2.11. You will need to use a compatible Scala version (2.11.x).

Note that support for Java 7 and Python 2.6 are deprecated as of Spark 2.0.0, and support for Scala 2.10 and versions of Hadoop before 2.6 are deprecated as of Spark 2.1.0, and may be removed in Spark 2.2.0.

Running the Examples and Shell

Spark comes with several sample programs. Scala, Java, Python and R examples are in the `examples/src/main` directory. To run one of the Java or Scala sample programs, use `bin/run-example <class> [<params>]` in the top-level Spark directory. (Behind the scenes, this invokes the more general `spark-submit` script for launching applications). For example,

Basic Concepts

Next, we move beyond the simple example and elaborate on the basics of Spark Streaming.

Linking

Similar to Spark, Spark Streaming is available through Maven Central. To write your own Spark Streaming program, you will have to add the following dependency to your SBT or Maven project.

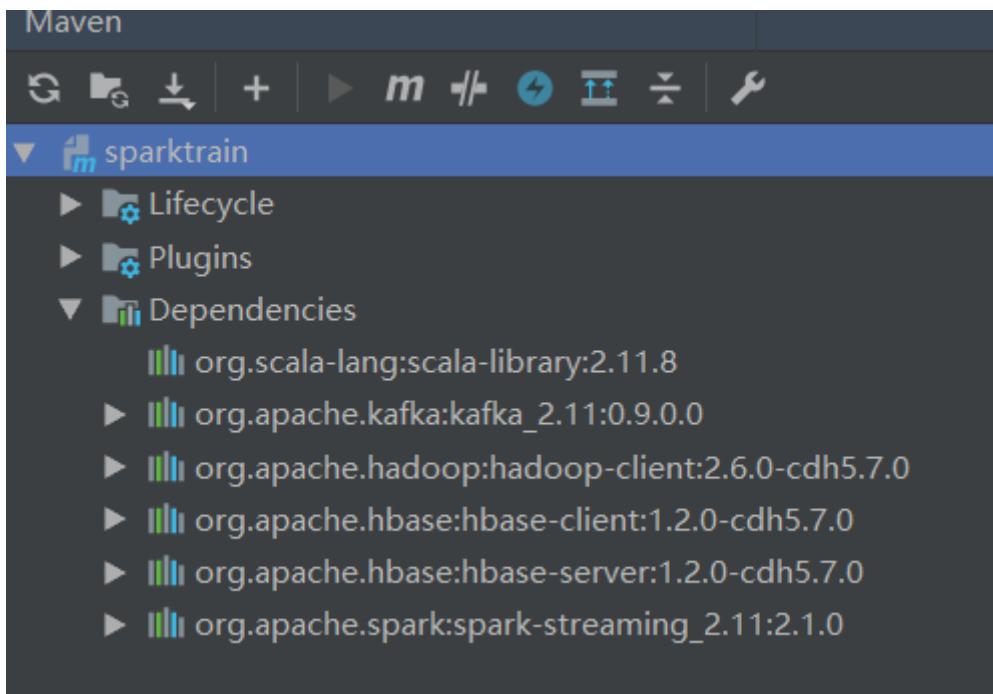
Maven	SBT
-------	-----

```
<dependency>
  <groupId>org.apache.spark</groupId>
  <artifactId>spark-streaming_2.11</artifactId>
  <version>2.1.0</version>
</dependency>
```

For ingesting data from sources like Kafka, Flume, and Kinesis that are not present in the Spark Streaming core API, you will have to add the corresponding artifact `spark-streaming-xyz_2.11` to the dependencies. For example, some of the common ones are as follows.

Source	Artifact
Kafka	<code>spark-streaming-kafka-0-8_2.11</code>
Flume	<code>spark-streaming-flume_2.11</code>

```
</dependency>
<dependency>
  <groupId>org.apache.spark</groupId>
  <artifactId>spark-streaming_2.11</artifactId>
  <version>${spark.version}</version>
</dependency>
</dependencies>
```



第6章 Spark Streaming入门

一、Spark Streaming概述

[官网](#)

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



Spark Streaming.个人的定义

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

特点

低延时

能从错误中高效的恢复：fault- tolerant

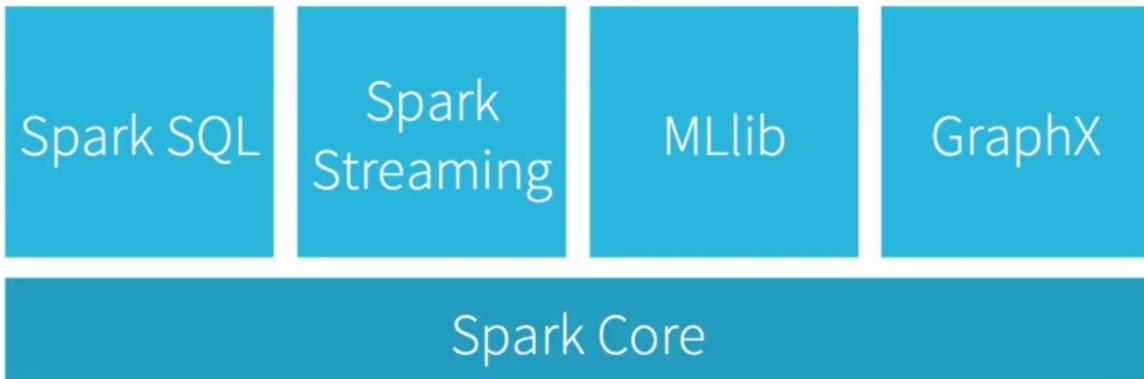
能够运行在成百上千的节点

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

One stack to rule them all:一栈式

二、Spark Streaming集成Spark生态系统的使用

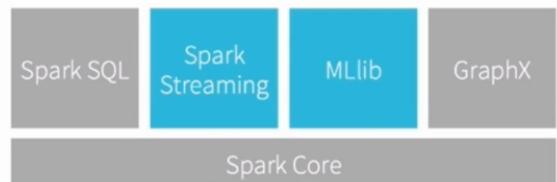
集成Spark生态系统的使用



- ◆ Combine machine learning with streaming processing
Learn models offline, apply them online

```
// Learn model offline
val model = KMeans.train(dataset, ...)

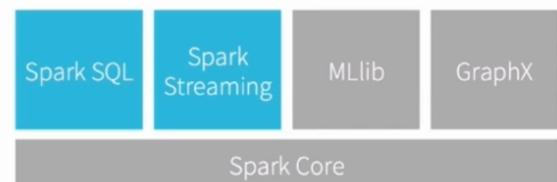
// Apply model online on stream
kafkaStream.map { event =>
    model.predict(event.feature)
}
```



- ◆ Combine SQL with streaming processing
Interactively query streaming data with SQL

```
// Register each batch in stream as table
kafkaStream.map { batchRDD =>
    batchRDD.registerTempTable("latestEvents")
}

// Interactively query table
sqContext.sql("select * from latestEvents")
```



三、词频统计功能着手入门Spark Streaming

从词频统计功能着手入门

◆ spark-submit执行

◆ spark-shell执行

[源码](#)

[参考案例](#)

```
nc -l k 9999
```

```
jungle@centosserver1:[/home/jungle]nc -l k 9999
-bash: nc: command not found
jungle@centosserver1:[/home/jungle]
```

```
yum install -y nc
```

```
--> Package nmap-ncat.x86_64 2:6.40-16.el7 will be installed
--> Finished Dependency Resolution
Dependencies Resolved

=====
Package           Arch         Version          Repository      Size
=====
Installing:
nmap-ncat        x86_64      2:6.40-16.el7    base           427 MB

Transaction Summary
=====
Install 1 Package

Total download size: 206 k
Installed size: 423 k
Downloading packages:
nmap-ncat-6.40-16.el7.x86_64.rpm                                         | 206 kB  00:00:00
Running transaction check
Running transaction test
Transaction test succeeded
Running transaction
  Installing : 2:nmap-ncat-6.40-16.el7.x86_64                               1/1
  Verifying  : 2:nmap-ncat-6.40-16.el7.x86_64                               1/1

Installed:
  nmap-ncat.x86_64 2:6.40-16.el7

Complete!
jungle@centosserver1:[/home/jungle]
```

```
Complete!
jungle@centosserver1:[/home/jungle]nc -l k 9999
```

1.spark-submit

==spark- submit的使用==

使用 spark- submit来提交我们的 spark应用程序运行的脚本(生产)

```
spark-submit --master local[2] \
--class org.apache.spark.examples.streaming.NetworkWordCount \
--name NetworkWordCount \
/home/jungle/app/spark-2.1.0-bin-2.6.0-cdh5.7.0/examples/jars/spark-
examples_2.11-2.1.0.jar centosserver1 9999
```

```
-----
Time: 1564413273000 ms
-----
Time: 1564413274000 ms
-----
Time: 1564413275000 ms
```

==测试效果==

```
jungle@centosserver1:[/home/jungle]nc -lk 9999
a a a s s s d d d a s d a s da^H s a s d
```

```
-----
Time: 1564413357000 ms
-----
(d, 5)
(s, 7)
(a, 6)
(d, 1)
```

2.spark-shell

如何使用spark-shell来提交(测试)

```
spark-shell --master local[2] --driver-class-path /home/jungle/app/hive-1.1.0-
cdh5.7.0/lib/mysql-connector-java-5.1.27-bin.jar
```

```
import org.apache.spark.SparkConf
import org.apache.spark.storage.StorageLevel
import org.apache.spark.streaming.{Seconds, StreamingContext}
val ssc = new StreamingContext(sc, Seconds(1))
val lines = ssc.socketTextStream("centosserver1", 9999)
val words = lines.flatMap(_.split(" "))
val wordCounts = words.map(x => (x, 1)).reduceByKey(_ + _)
wordCounts.print()
ssc.start()
ssc.awaitTermination()
```

```
scala> import org.apache.spark.streaming.{Seconds, StreamingContext}
import org.apache.spark.streaming.{Seconds, StreamingContext}

scala> val ssc = new StreamingContext(sc, Seconds(1))
ssc: org.apache.spark.streaming.StreamingContext = org.apache.spark.streaming.StreamingContext@5fe80760

scala> val lines = ssc.socketTextStream("centosserver1", 9999)
lines: org.apache.spark.streaming.dstream.ReceiverInputDStream[String] = org.apache.spark.streaming.dstream.SocketInputDStr

scala> val words = lines.flatMap(_.split(" "))
words: org.apache.spark.streaming.dstream.DStream[String] = org.apache.spark.streaming.dstream.FlatMappedDStream@648e25f2

scala> val wordCounts = words.map(x => (x, 1)).reduceByKey(_ + _)
wordCounts: org.apache.spark.streaming.dstream.DStream[(String, Int)] = org.apache.spark.streaming.dstream.ShuffledDStream@6

scala> wordCounts.print()

scala> ssc.start()

scala> ssc.awaitTermination()-----
Time: 1564413999000 ms
-----
```

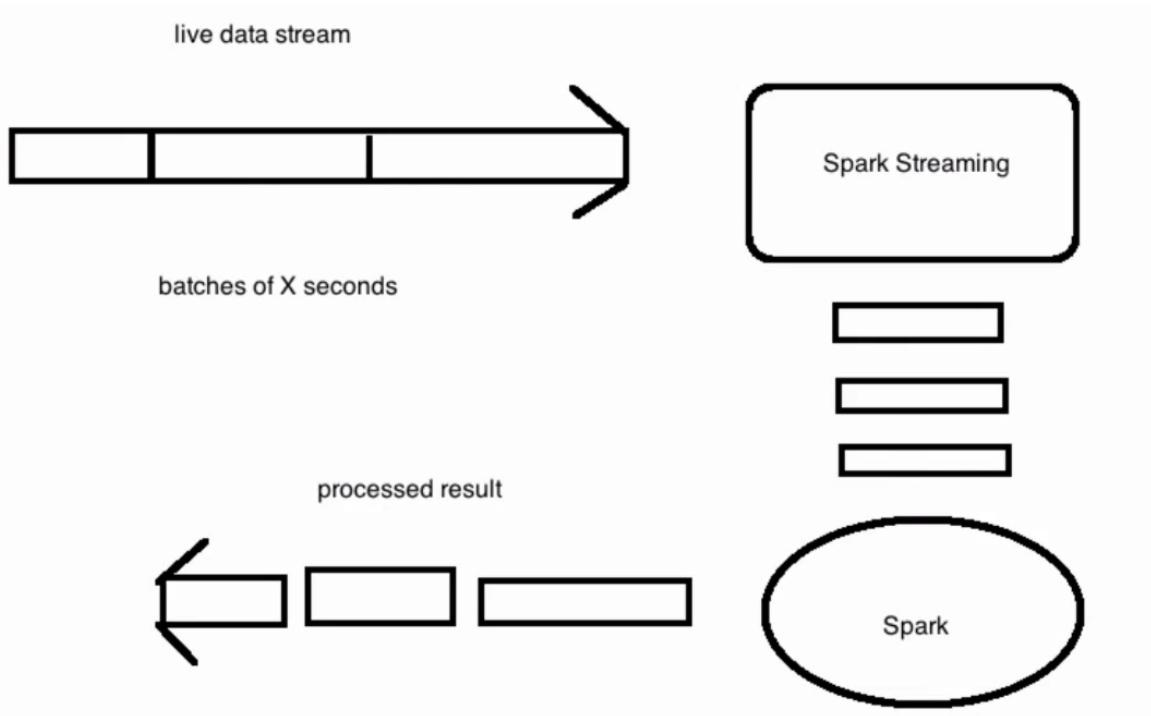
==测试效果==

```
jungle@centosserver1:[/home/jungle]nc -lk 9999
a a a s s s d d d a s d a s da^H s a s d
a a s s d d a s d f
```

四、Spark Streaming工作原理(粗粒度)

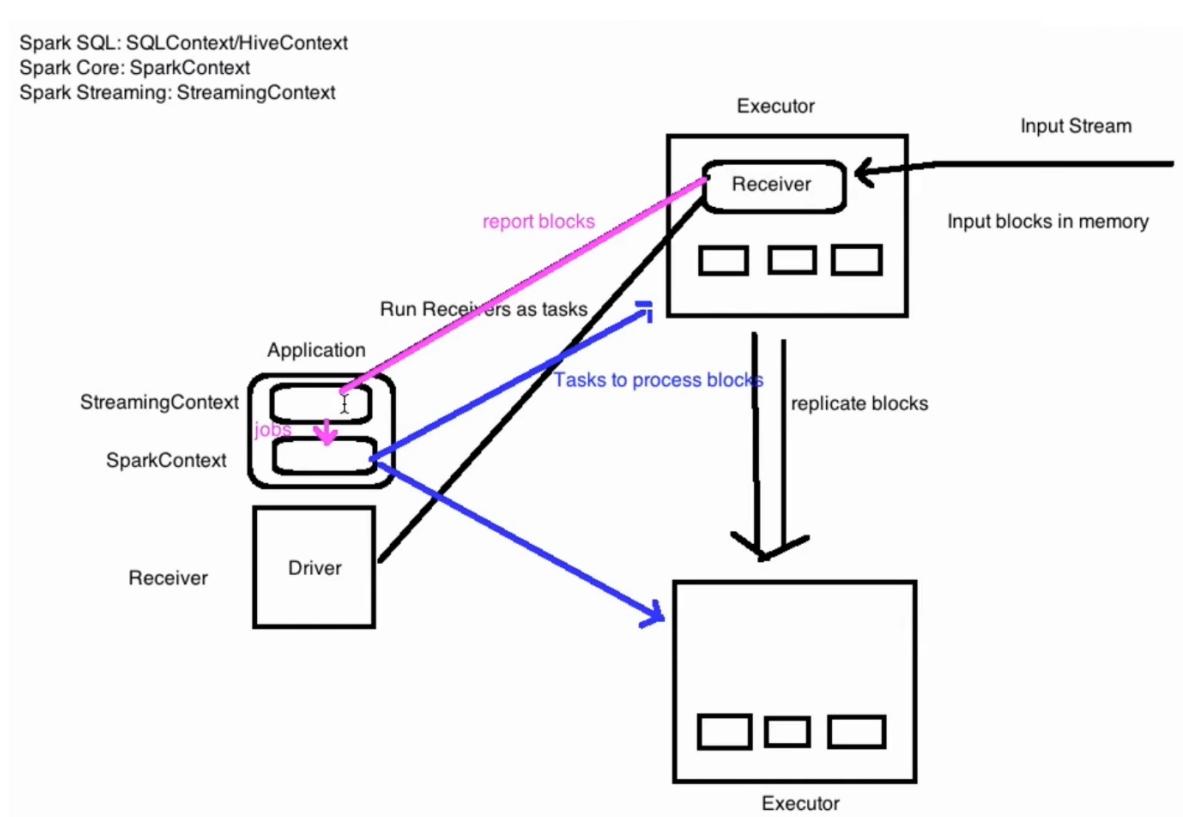
工作原理:粗粒度

Spark Streaming接收到实时数据流,把数据按照指定的时间段切成一片片小的数据块,
然后把小的数据块传给 Spark Engine处理。



五、Spark Streaming工作原理(细粒度)

Spark SQL: SQLContext/HiveContext
 Spark Core: SparkContext
 Spark Streaming: StreamingContext



第7章 Spark Streaming核心概念与编程

一、核心概念

1.StreamingContext

```

StreamingContext

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

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

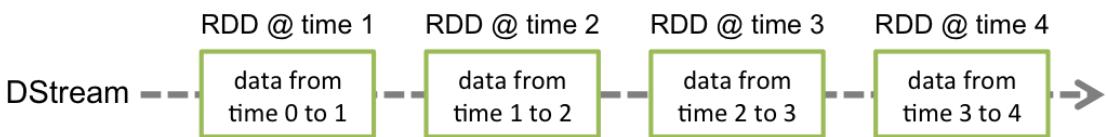
batch interval可以根据你的应用程序需求的延迟要求以及集群可用的资源情况来设置

```

2.DStream

Discretized Streams (DStreams)
 Internally, a DStream is represented by a continuous series of RDDs
 Each RDD in a DStream contains data from a certain interval

对 DStream操作算子,比如map/ flatmap,其实底层会被翻译为对 Dstream中的每个RDD都做相同的操作



3.Input DStreams和Receivers

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.

二、案例实战

1.Spark Streaming处理socket数据

- NetworkWordCount

```

package com.jungle.spark

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

/**
 * Spark Streaming处理Socket数据
 *
 * 测试: nc
 */
object NetworkwordCount {

```

```

def main(args: Array[String]): Unit = {

    val sparkConf = new
    SparkConf().setMaster("local[2]").setAppName("NetworkwordCount")

    /**
     * 创建StreamingContext需要两个参数: SparkConf和batch interval
     */
    val ssc = new StreamingContext(sparkConf, Seconds(5))

    val lines = ssc.socketTextStream("192.168.1.18", 6789)

    val result = lines.flatMap(_.split(" ")).map((_, 1)).reduceByKey(_+_)
    result.print()

    ssc.start()
    ssc.awaitTermination()
}
}

```

==出错==

```

Exception in thread "main" java.lang.NoSuchMethodError:
com.fasterxml.jackson.module.scala.deser.BigDecimalDeserializer$.handledType()Lj
ava/lang/Class;

```

解决：

手动在pom.xml文件中添加该依赖

```

<dependency>
    <groupId>com.fasterxml.jackson.module.scala</groupId>
    <artifactId>jackson-module-scala_2.11</artifactId>
    <version>2.6.5</version>
</dependency>

```

- shell端

```
nc -l k 6789
```

```
w q q q w
```

```
jungle@centosserver1: [/home/jungle]nc -l k 6789
w q q q w
```

==出错==

```
Caused by: java.lang.NoClassDefFoundError: net/jpountz/util/safeutils
```

解决：

去maven上查找

The screenshot shows the Maven Repository search results for 'jpountz'. The search bar at the top contains 'jpountz'. Below it, the results are listed under 'Found 3 results':

1. LZ4 and XxHash
org.lz4 > lz4-java
Java ports and bindings of the LZ4 compression algorithm and the xxHash hashing algorithm
Last Release on May 15, 2019
2. LZ4 and XxHash
net.jpountz.lz4 > lz4
Java ports and bindings of the LZ4 compression algorithm and the xxHash hashing algorithm
Last Release on Nov 26, 2014
3. LZ4
net.jpountz.lz4.wso2 > lz4
This bundle represents the lz4.
Last Release on Dec 2, 2014

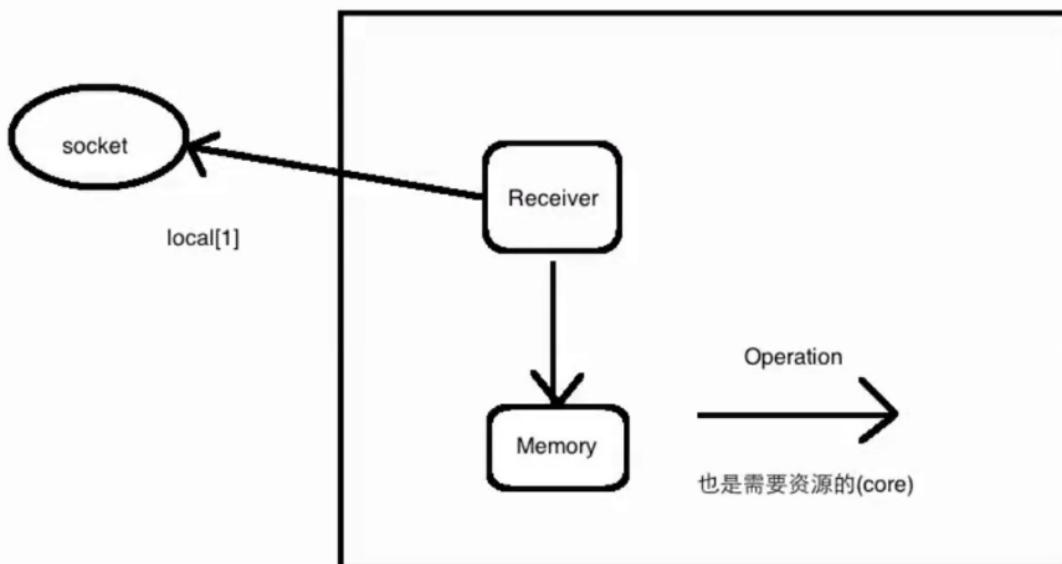
```
<!-- https://mvnrepository.com/artifact/net.jpountz.lz4/lz4 -->
<dependency>
    <groupId>net.jpountz.lz4</groupId>
    <artifactId>lz4</artifactId>
    <version>1.3.0</version>
</dependency>
```

==再次运行==

```
jungle@centosserver1:[/home/jungle] nc -lk 6789
w q q q w
wq wq
wqw wqw ewq ewq
[green square]
```

```
NetworkWordCount x
19/07/31 15:24:05 INFO scheduler.DAGScheduler: ResultStage 36
19/07/31 15:24:05 INFO scheduler.DAGScheduler: Job 18 finished
-----
Time: 1564557845000 ms
-----
(wqw,2)
(ewq,2)

19/07/31 15:24:05 INFO scheduler.JobScheduler: Finished job st
19/07/31 15:24:05 INFO scheduler.JobScheduler: Total delay: 0.
19/07/31 15:24:05 INFO rdd.ShuffledRDD: Removing RDD 32 from p
19/07/31 15:24:05 INFO storage.BlockManager: Removing RDD 32
```



对于是需要Receiver来接收数据的处理，那么本地测试时你的local[?] ?一定是要大于1的

2.Spark Streaming处理文件系统数据

- FileWordCount

```

package com.jungle.spark

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

/**
 * 使用Spark Streaming处理文件系统(local/hdfs)的数据
 */
object FilewordCount {

    def main(args: Array[String]): Unit = {

        val sparkConf = new
        SparkConf().setMaster("local").setAppName("FilewordCount")
        val ssc = new StreamingContext(sparkConf, Seconds(5))

        val lines = ssc.textFileStream("file:///E:/data/clean")

        val result = lines.flatMap(_.split(" ")).map((_, 1)).reduceByKey(_+_)
        result.print()

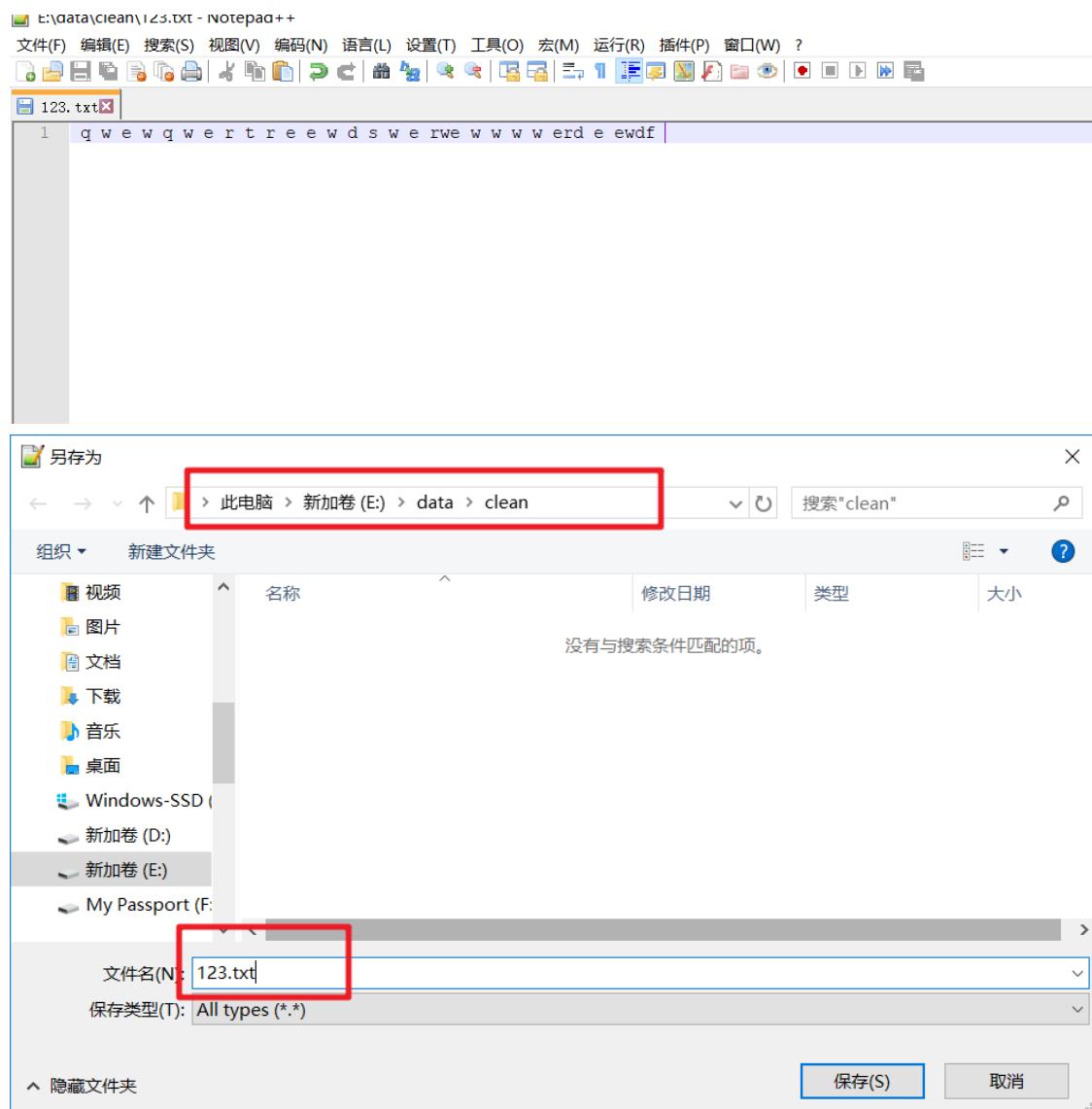
        ssc.start()
        ssc.awaitTermination()

    }
}

```

==E:/data/clean文件夹下是空的==

- 在上述文件下新建文件



```
val result = lines.flatMap(_.split(" ")).map((_, 1))
result.print()
main(args: Array[String])
```

Run: FileWordCount x

19/07/31 15:34:50 INFO scheduler.DAGScheduler: ResultStage 19 (pr...
19/07/31 15:34:50 INFO scheduler.DAGScheduler: Job 9 finished: pr...
Time: 1564558490000 ms

(d,1)
(w,9)
(s,1)
(e,6)
(rwe,1)
(ewdf,1)
(t,1)
(,1)
(q,2)
(erd,1)
...

19/07/31 15:34:50 INFO scheduler.JobScheduler: Finished job stream
19/07/31 15:34:50 INFO scheduler.JobScheduler: Total delay: 0.199
19/07/31 15:34:50 INFO rdd.ShuffledRDD: Removing RDD 44 from pers...
19/07/31 15:34:50 INFO storage.BlockManager: Removing RDD 44
19/07/31 15:34:50 INFO rdd.MapPartitionsRDD: Removing RDD 43 from
19/07/31 15:34:50 INFO storage.BlockManager: Removing RDD 43

Run TODO Terminal Messages Java Enterprise

Build completed successfully with 1 warning in 2 s 545 ms (a minute ago)

第8章 Spark Streaming进阶与案例实战

一、目录

Spark Streaming进阶

- ◆ 带状态的算子：UpdateStateByKey
- ◆ 实战：计算到目前为止累积出现的单词个数写入到MySQL中
- ◆ 基于window的统计
- ◆ 实战：黑名单过滤
- ◆ 实战：Spark Streaming整合Spark SQL实战

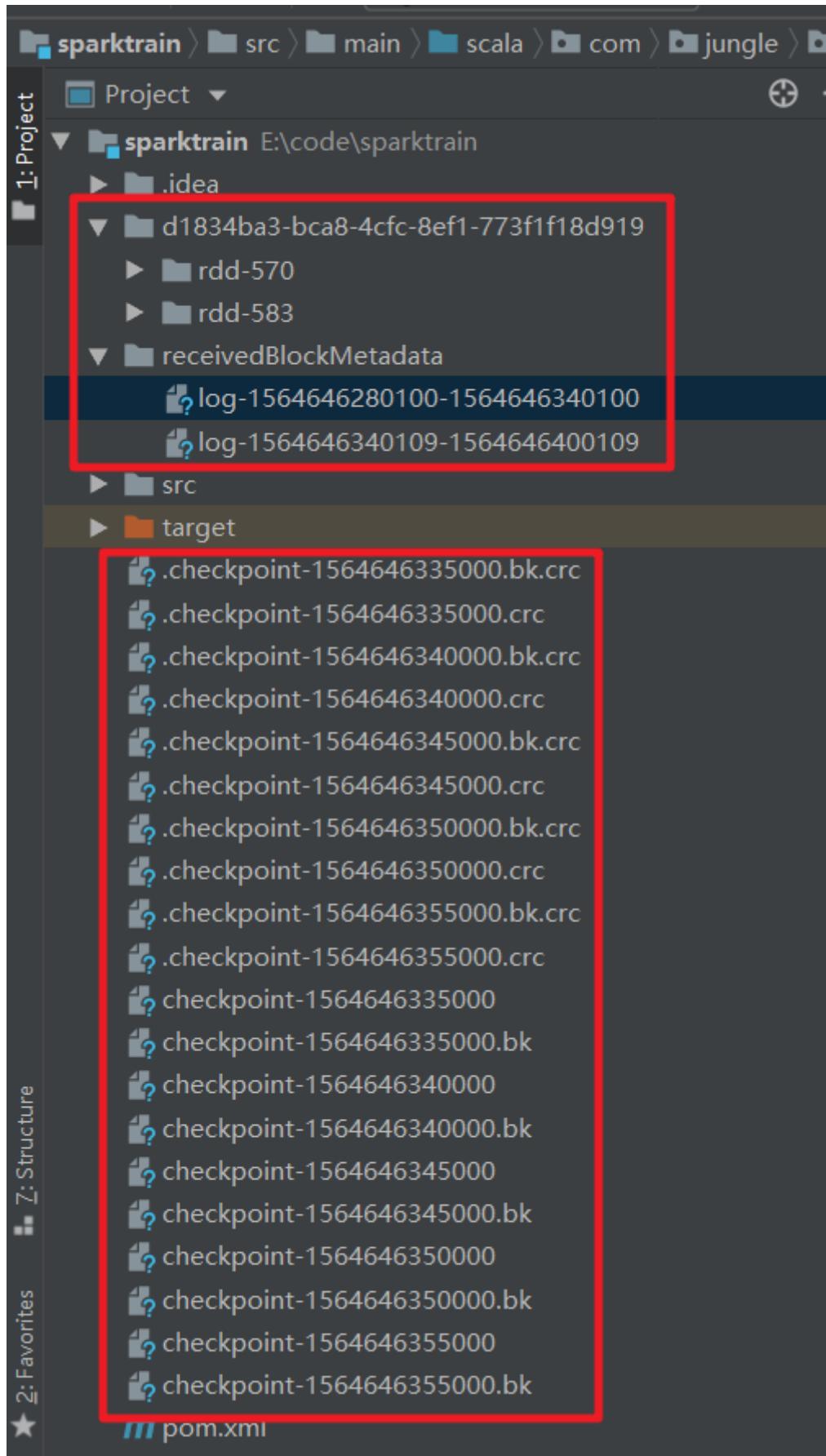
二、updateStateByKey算子的使用

updateStateByKey算子

需求：统计到目前为止累积出现的单词的个数(需要保持住以前的状态)

```
java.lang.IllegalArgumentException: requirement failed: The checkpoint  
directory has not been set. Please set it by  
StreamingContext.checkpoint().
```

```
// 如果使用了stateful的算子，必须要设置checkpoint  
// 在生产环境中，建议大家把checkpoint设置到HDFS的某个文件夹中  
//.表示当前目录  
ssc.checkpoint(".")
```



- StatefulWordCount

```
package com.jungle.spark

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

```


    /**
     * 使用Spark Streaming完成有状态统计
     */
    object StatefulWordCount {

        def main(args: Array[String]): Unit = {

            val sparkConf = new
            SparkConf().setAppName("StatefulWordCount").setMaster("local[2]")
            val ssc = new StreamingContext(sparkConf, Seconds(5))

            // 如果使用了stateful的算子，必须要设置checkpoint
            // 在生产环境中，建议大家把checkpoint设置到HDFS的某个文件夹中
            // 表示当前目录
            ssc.checkpoint("./")

            val lines = ssc.socketTextStream("192.168.1.18", 6789)

            val result = lines.flatMap(_.split(" ")).map((_, 1))
            val state = result.updateStateByKey[Int](updateFunction _)

            state.print()

            ssc.start()
            ssc.awaitTermination()
        }

        /**
         * 把当前的数据去更新已有的或者是老的数据
         * @param currentvalues 当前的
         * @param prevalues 老的
         * @return
         */
        def updateFunction(currentValues: Seq[Int], prevalues: Option[Int]): Option[Int] = {
            val current = currentValues.sum
            val pre = prevalues.getOrElse(0)

            Some(current + pre)
        }
    }
}

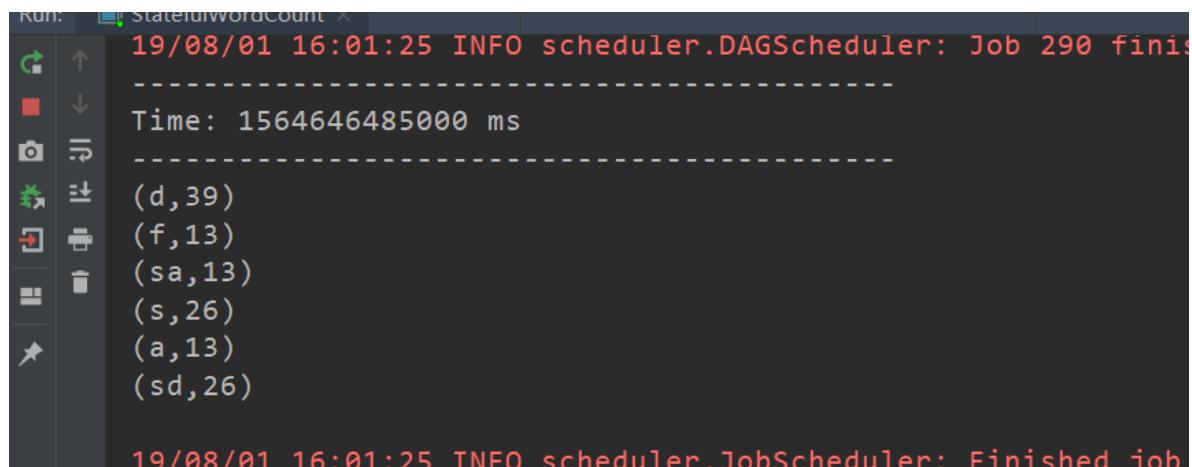

```

- shell

```
nc -l 6789
```

```
jungle@centosserver1:[/home/jungle]nc -lk 6789
a s d f d sa s sd d sd
a s d f d sa s sd d sd
a s d f d sa s sd d sd
a s d f d sa s sd d sd
a s d f d sa s sd d sd
a s d f d sa s sd d sd
a s d f d sa s sd d sd
a s d f d sa s sd d sd
a s d f d sa s sd d sd
a s d f d sa s sd d sd
a s d f d sa s sd d sd
a s d f d sa s sd d sd
a s d f d sa s sd d sd
a s d f d sa s sd d sd
a s d f d sa s sd d sd
a s d f d sa s sd d sd
a s d f d sa s sd d sd
a s d f d sa s sd d sd
```

--结果--



```
Run: statefulwordCount
19/08/01 16:01:25 INFO scheduler.DAGScheduler: Job 290 finished at 19/08/01 16:01:25
-----[REDACTED]-----
Time: 1564646485000 ms
-----[REDACTED]
(d,39)
(f,13)
(sa,13)
(s,26)
(a,13)
(sd,26)

19/08/01 16:01:25 INFO scheduler.JobScheduler: Finished job
```

三、统计结果写入到MySQL数据库中

```
create database imooc_spark;
use imooc_spark;
```

```
create table wordcount(
word varchar(50) default null,
wordcount int(10) default null
);
```

```
show tables;
```

```

mysql> show tables;
+-----+
| Tables_in_imooc_spark |
+-----+
| wordcount               |
+-----+
1 row in set (0.00 sec)

```

- 引入依赖

```

<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.17</version>
</dependency>

```

- ForeachRDDApp

```

package com.jungle.spark

import java.sql.DriverManager

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

/**
 * 使用Spark Streaming完成词频统计，并将结果写入到MySQL数据库中
 */
object ForeachRDDApp {

    def main(args: Array[String]): Unit = {

        val sparkConf = new
        SparkConf().setAppName("ForeachRDDApp").setMaster("local[2]")
        val ssc = new StreamingContext(sparkConf, Seconds(5))

        val lines = ssc.socketTextStream("192.168.1.18", 6789)

        val result = lines.flatMap(_.split(" ")).map((_, 1)).reduceByKey(_ + _)

        //result.print() //此处仅仅是将统计结果输出到控制台

        //TODO... 将结果写入到MySQL
        //    result.foreachRDD(rdd =>{
        //        val connection = createConnection() // executed at the driver
        //        rdd.foreach { record =>
        //            val sql = "insert into wordcount(word, wordcount)
values('"+record._1 + "','" + record._2 +"')"
        //            connection.createStatement().execute(sql)
        //        }
        //    })
    }
}

```

```

    result.print()

    result.foreachRDD(rdd => {
      rdd.foreachPartition(partitionOfRecords => {
        val connection = createConnection()
        partitionOfRecords.foreach(record => {
          val sql = "insert into wordcount(word, wordcount) values('" +
record._1 + "','" + record._2 + "')"
          connection.createStatement().execute(sql)
        })
      })
    })

    connection.close()
  })
}

ssc.start()
ssc.awaitTermination()
}

/**
 * 获取MySQL的连接
 */
def createConnection() = {
  Class.forName("com.mysql.jdbc.Driver")

  DriverManager.getConnection("jdbc:mysql://192.168.1.18:8806/imooc_spark",
"root", "123456")
}
}

```

```

Run:  ForeachRDDApp x
19/08/01 16:34:50 INFO scheduler.TaskSchedulerImpl: Removed
19/08/01 16:34:50 INFO scheduler.DAGScheduler: ResultStage
19/08/01 16:34:50 INFO scheduler.DAGScheduler: Job 20 finis
-----
Time: 1564648490000 ms
-----
(d,39)
(,1)
(f,13)
(sa,13)
(s,26)
(a,13)
(sd,26)

19/08/01 16:34:50 INFO scheduler.JobScheduler: Finished job
19/08/01 16:34:50 INFO scheduler.JobScheduler: Starting job

```

```

mysql> show tables;
+-----+
| Tables_in_imooc_spark |
+-----+
| wordcount               |
+-----+
1 row in set (0.00 sec)

mysql> select * from wordcount;
+-----+-----+
| word | wordcount |
+-----+-----+
| d   |      39 |
|     |      1  |
| f   |      13 |
| sa  |      13 |
| s   |      26 |
| a   |      13 |
| sd  |      26 |
+-----+
7 rows in set (0.01 sec)

```

==需改进==

存在的问题:

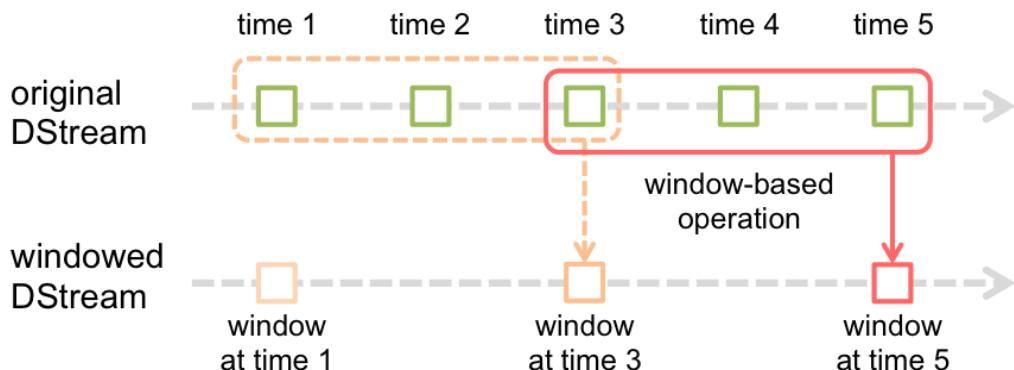
1) 对于已有的数据做更新，而是所有的数据均为insert

改进思路:

- a) 在插入数据前先判断单词是否存在，如果存在就update，不存在则insert
- b) 工作中：HBase/Redis

2) 每个rdd的partition创建connection，建议大家改成连接池

四、窗口函数的使用



window: 定时的进行一个时间段内的数据处理

window length : 窗口的长度

sliding interval: 窗口的间隔

这2个参数和我们的batch size有关系：倍数

每隔多久计算某个范围内的数据：每隔10秒计算前10分钟的wc

==> 每隔sliding interval统计前window length的值

五、黑名单过滤

1.需求分析

实战：黑名单过滤

◆ transform算子的使用

◆ Spark Streaming整合RDD进行操作

黑名单过滤

访问日志 ==> DStream

20180808,zs

20180808,ls

20180808,ww

==> (zs: 20180808,zs)(ls: 20180808,ls)(ww: 20180808,ww)

黑名单列表 ==> RDD

zs

ls

==>(zs: true)(ls: true)

==> 20180808,ww

leftjoin

(zs: <20180808,zs>, <true>) x

(ls: <20180808,ls>, <true>) x

(ww: <20180808,ww>, <false>) ==> tuple 1

2.程序实现

- TransformApp

```

package com.jungle.spark

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

/**
 * 黑名单过滤
 */
object TransformApp {

    def main(args: Array[String]): Unit = {

        val sparkConf = new
SparkConf().setMaster("local[2]").setAppName("NetworkWordCount")

        /**
         * 创建StreamingContext需要两个参数: SparkConf和batch interval
         */
        val ssc = new StreamingContext(sparkConf, Seconds(5))

        /**
         * 构建黑名单
         */
        val blacks = List("zs", "ls")
        val blacksRDD = ssc.sparkContext.parallelize(blacks).map(x => (x, true))

        val lines = ssc.socketTextStream("192.168.1.18", 6789)
        val clicklog = lines.map(x => (x.split(",")(1), x)).transform(rdd => {
            rdd.leftOuterJoin(blacksRDD)
                .filter(x=> x._2._2.getOrElse(false) != true)
                .map(x=>x._2._1)
        })

        clicklog.print()

        ssc.start()
        ssc.awaitTermination()
    }
}

```

- shell

```
nc -lk 6789
```

```
20160410,zs
20160410,ls
20160410,ww
```

```
jungle@centosserver1:[/home/jungle]nc -lk 6789  
20160410,zs  
20160410,ls  
20160410,ww  
  
[TransformApp ×]  
19/08/01 17:20:25 INFO scneauier.tasks: Adding ta  
19/08/01 17:20:25 INFO scheduler.TaskSetManager: Starting tas  
19/08/01 17:20:25 INFO executor.Executor: Running task 0.0 in  
19/08/01 17:20:25 INFO storage.ShuffleBlockFetcherIterator: G  
19/08/01 17:20:25 INFO storage.ShuffleBlockFetcherIterator: S  
19/08/01 17:20:25 INFO storage.ShuffleBlockFetcherIterator: G  
19/08/01 17:20:25 INFO storage.ShuffleBlockFetcherIterator: S  
19/08/01 17:20:25 INFO executor.Executor: Finished task 0.0 i  
19/08/01 17:20:25 INFO scheduler.TaskSetManager: Finished tas  
19/08/01 17:20:25 INFO scheduler.TaskSchedulerImpl: Removed T  
19/08/01 17:20:25 INFO scheduler.DAGScheduler: ResultStage 6  
19/08/01 17:20:25 INFO scheduler.DAGScheduler: Job 2 finished  
-----  
Time: 1564651225000 ms  
-----  
20160410,ww  
  
19/08/01 17:20:25 INFO scheduler.JobScheduler: Finished job s
```

六、Spark Streaming整合Spark SQL操作

- 添加依赖

```
<dependency>  
    <groupId>org.apache.spark</groupId>  
    <artifactId>spark-sql_2.11</artifactId>  
    <version>${spark.version}</version>  
</dependency>
```

- SqlNetworkWordCount

```
package com.jungle.spark

import org.apache.spark.SparkConf
import org.apache.spark.rdd.RDD
import org.apache.spark.sql.SparkSession
import org.apache.spark.streaming.{Seconds, StreamingContext, Time}

/**
 * Spark Streaming整合Spark SQL完成词频统计操作
 */
object SqlNetworkWordCount {

    def main(args: Array[String]): Unit = {
        val sparkConf = new
        SparkConf().setAppName("ForeachRDDApp").setMaster("local[2]")
        val ssc = new StreamingContext(sparkConf, Seconds(5))

        val lines = ssc.socketTextStream("192.168.1.18", 6789)
        val words = lines.flatMap(_.split(" "))

        // Convert RDDs of the words DStream to DataFrame and run SQL query
        words.foreachRDD { (rdd: RDD[String], time: Time) =>
            val spark =
                SparkSessionSingleton.getInstance(rdd.sparkContext.getConf)
            import spark.implicits._

            // Convert RDD[String] to RDD[case class] to DataFrame
            val wordsDataFrame = rdd.map(w => Record(w)).toDF()
        }
    }
}
```

```

// Creates a temporary view using the DataFrame
wordsDataFrame.createOrReplaceTempView("words")

// Do word count on table using SQL and print it
val wordCountsDataFrame =
    spark.sql("select word, count(*) as total from words group by word")
println(s"===== $time =====")
wordCountsDataFrame.show()
}

ssc.start()
ssc.awaitTermination()
}

/** Case class for converting RDD to DataFrame */
case class Record(word: String)

/** Lazily instantiated singleton instance of SparkSession */
object SparkSessionSingleton {

    @transient private var instance: SparkSession = _

    def getInstance(sparkConf: SparkConf): SparkSession = {
        if (instance == null) {
            instance = SparkSession
                .builder
                .config(sparkConf)
                .getOrCreate()
        }
        instance
    }
}
}

```

- shell

```
nc -lk 6789
```

```
jungle@centosserver1:[/home/jungle]nc -lk 6789
a a a a a b b
a a a a a b ba a a a a b b
a a a a a b b
█
```

↑	19/08/01 18:29:20
↓	19/08/01 18:29:20
⤵	19/08/01 18:29:20
+-----+	
word total	
+-----+	
ba 1	
b 5	
a 14	
19/08/01 18:29:20	
+-----+	

第9章 Spark Streaming整合Flume

[官网](#)

一、Push方式整合之Flume Agent配置开发

- flume_push_streaming.conf

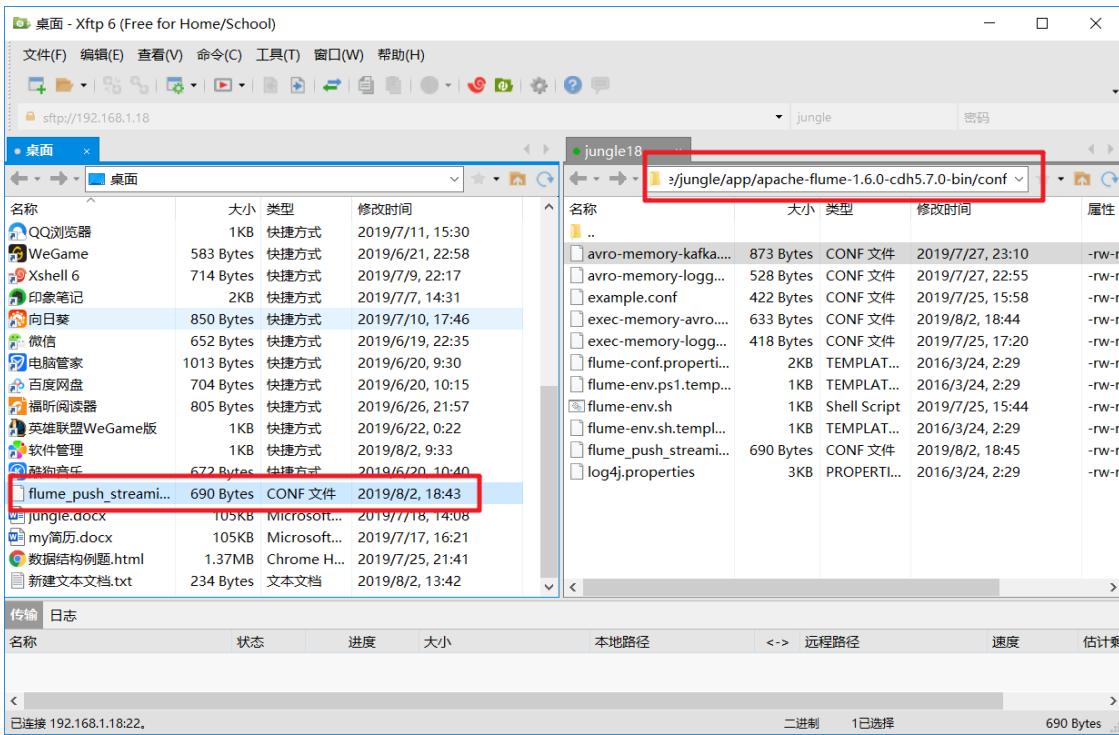
```
simple-agent.sources = netcat-source
simple-agent.sinks = avro-sink
simple-agent.channels = memory-channel

simple-agent.sources.netcat-source.type = netcat
simple-agent.sources.netcat-source.bind = centosserver1
simple-agent.sources.netcat-source.port = 44444

simple-agent.sinks.avro-sink.type = avro
simple-agent.sinks.avro-sink.hostname = centosserver1
simple-agent.sinks.avro-sink.port = 41414

simple-agent.channels.memory-channel.type = memory

simple-agent.sources.netcat-source.channels = memory-channel
simple-agent.sinks.avro-sink.channel = memory-channel
```



- 添加依赖

```
<dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-streaming-flume_2.11</artifactId>
    <version>${spark.version}</version>
</dependency>
```

- FlumePushWordCount

```
package com.jungle.spark

import org.apache.spark.SparkConf
import org.apache.spark.streaming.flume.FlumeUtils
import org.apache.spark.streaming.{Seconds, StreamingContext}

/**
 * Spark Streaming整合Flume的第一种方式
 */
object FlumePushwordCount {

    def main(args: Array[String]): Unit = {
        if(args.length != 2) {
            System.err.println("Usage: FlumePushwordCount <hostname> <port>")
            System.exit(1)
        }

        val Array(hostname, port) = args

        val sparkConf = new
        SparkConf()//.setMaster("local[2]").setAppName("FlumePushwordCount")
        val ssc = new StreamingContext(sparkConf, Seconds(5))

        //TODO... 如何使用SparkStreaming整合Flume
        val flumeStream = Flumeutils.createStream(ssc, hostname, port.toInt)
    }
}
```

```

        flumeStream.map(x=> new String(x.event.getBody.array()).trim)
            .flatMap(_.split(" ")).map((_,1)).reduceByKey(_+_).print()

        ssc.start()
        ssc.awaitTermination()
    }
}

```

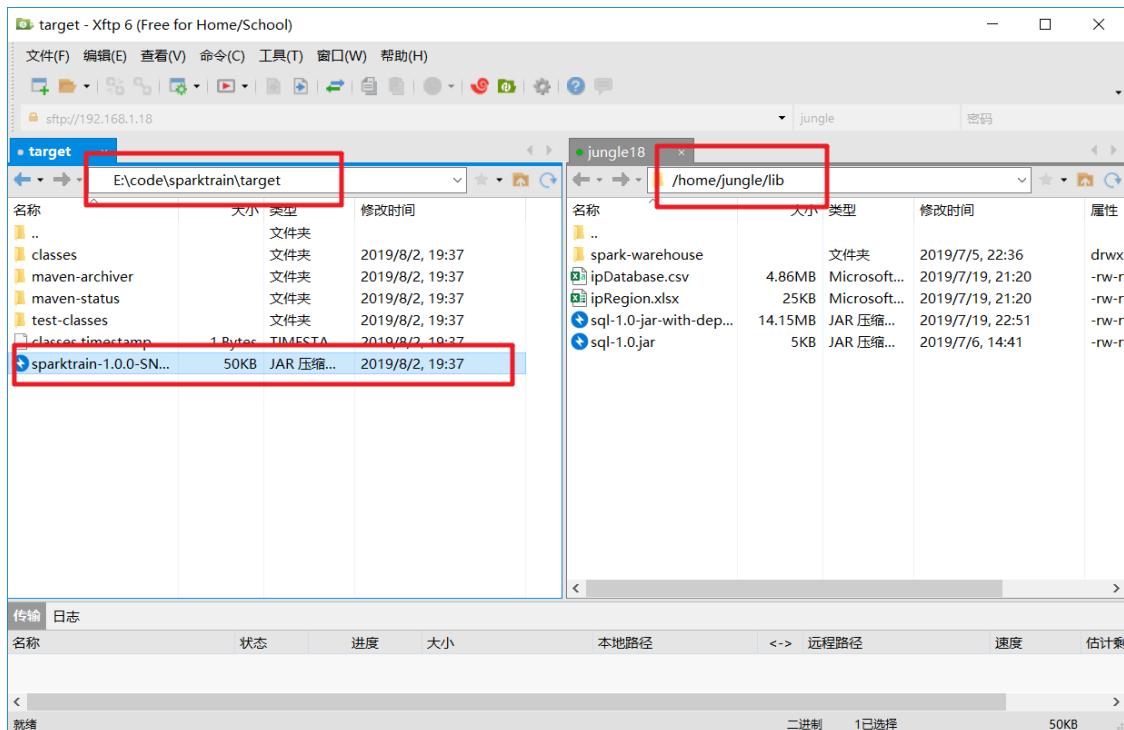
- 打包

```
mvn clean package -DskipTests
```

Terminal: Local +

Microsoft Windows [版本 10.0.17134.885]
(c) 2018 Microsoft Corporation。保留所有权利。

E:\code\sparktrain>mvn clean package -DskipTests



- 提交任务

```

spark-submit --master local[2] \
--class com.jungle.spark.FlumePushwordCount \
--packages org.apache.spark:spark-streaming-flume_2.11:2.2.0 \
/home/jungle/lib/sparktrain-1.0.0-SNAPSHOT.jar \
centosserver1 41414

```

- 启动flume

```
flume-ng agent \
--name simple-agent \
--conf $FLUME_HOME/conf \
--conf-file $FLUME_HOME/conf/flume_push_streaming.conf \
-Dflume.root.logger=INFO,console
```

- 测试

```
telnet localhost 44444
```

二、Pull方式整合之Flume Agent配置开发

- flume_pull_streaming.conf

```
simple-agent.sources = netcat-source

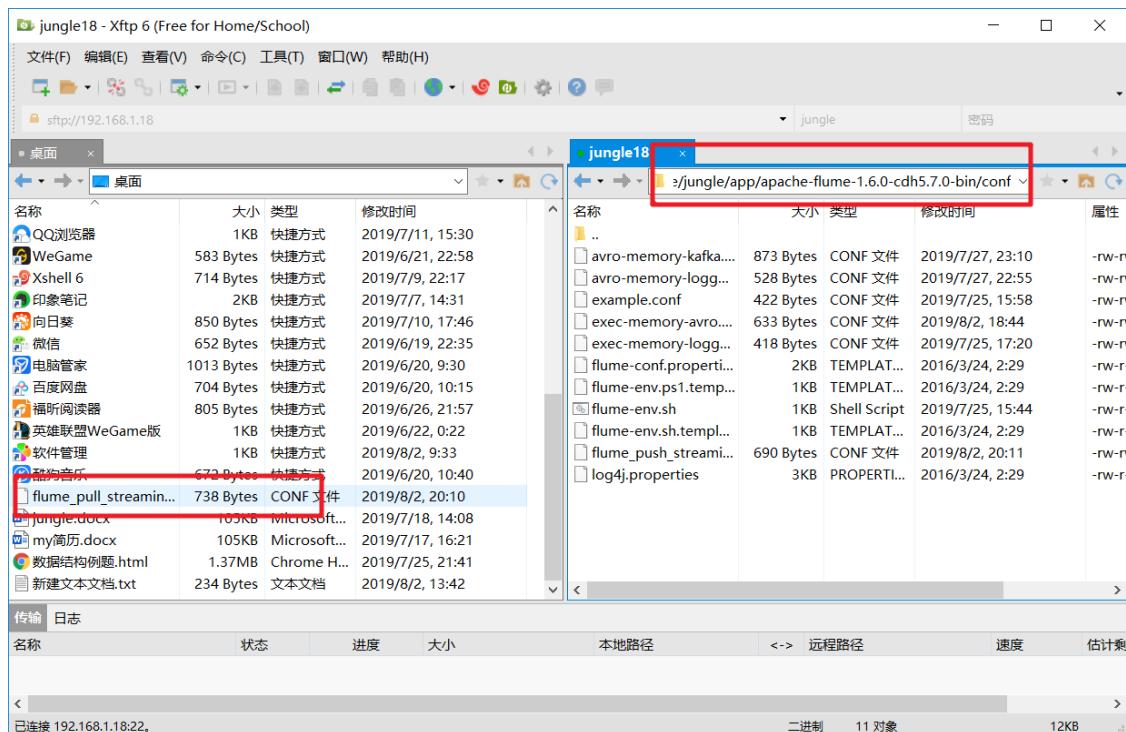
simple-agent.sinks = spark-sink
simple-agent.channels = memory-channel

simple-agent.sources.netcat-source.type = netcat
simple-agent.sources.netcat-source.bind = centosserver1
simple-agent.sources.netcat-source.port = 44444

simple-agent.sinks.spark-sink.type =
org.apache.spark.streaming.flume.sink.SparkSink
simple-agent.sinks.spark-sink.hostname = centosserver1
simple-agent.sinks.spark-sink.port = 41414

simple-agent.channels.memory-channel.type = memory

simple-agent.sources.netcat-source.channels = memory-channel
simple-agent.sinks.spark-sink.channel = memory-channel
```



- 添加依赖

```
<dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-streaming-flume-sink_2.11</artifactId>
    <version>${spark.version}</version>
</dependency>

<dependency>
    <groupId>org.scala-lang</groupId>
    <artifactId>scala-library</artifactId>
    <version>${scala.version}</version>
</dependency>

<dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-lang3</artifactId>
    <version>3.5</version>
</dependency>
```

- FlumePullWordCount

```
package com.jungle.spark

import org.apache.spark.SparkConf
import org.apache.spark.streaming.flume.FlumeUtils
import org.apache.spark.streaming.{Seconds, StreamingContext}

/**
 * Spark Streaming整合Flume的第二种方式
 */
object FlumePullwordCount {

    def main(args: Array[String]): Unit = {

        if(args.length != 2) {
            System.err.println("Usage: FlumePullwordCount <hostname> <port>")
            System.exit(1)
        }

        val Array(hostname, port) = args

        val sparkConf = new SparkConf()
//.setMaster("local[2]").setAppName("FlumePullwordCount")
        val ssc = new StreamingContext(sparkConf, Seconds(5))

        //TODO... 如何使用SparkStreaming整合Flume
        val flumeStream = FlumeUtils.createPollingStream(ssc, hostname,
port.toInt)

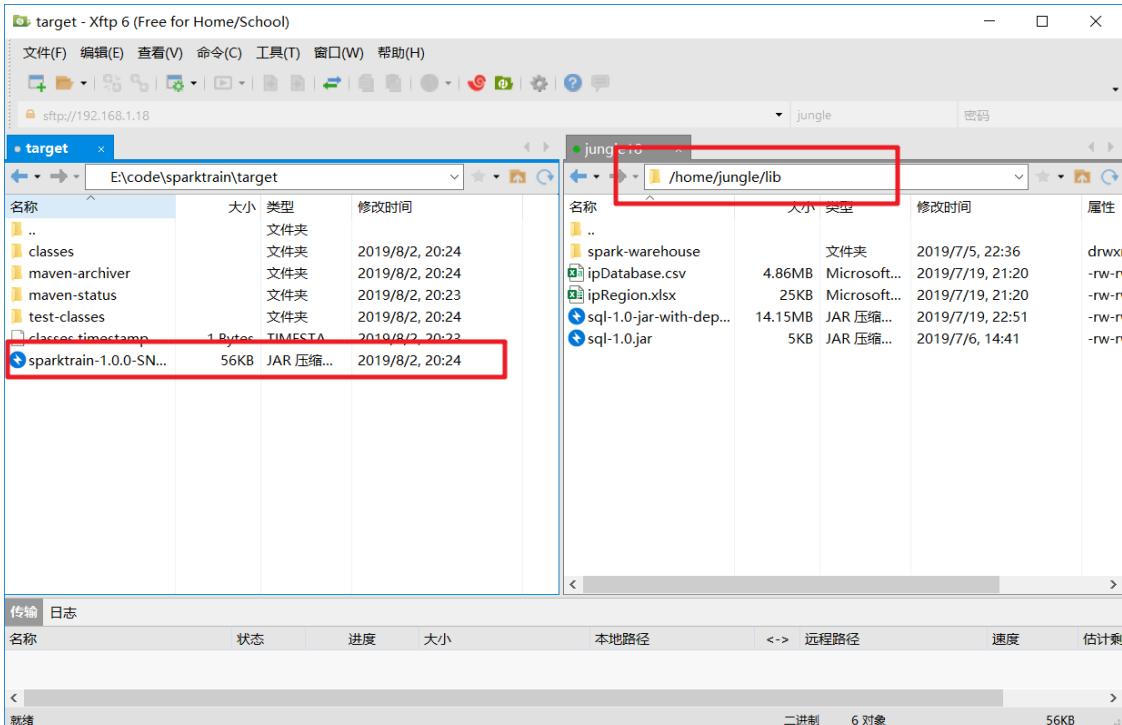
        flumeStream.map(x=> new String(x.event.getBody.array()).trim)
            .flatMap(_.split(" ")).map((_,1)).reduceByKey(_+_).print()

        ssc.start()
        ssc.awaitTermination()
    }
}
```

- 打包

```
mvn clean package -DskipTests
```

- 上传至服务器



==注意到：先启动flume，后启动spark streaming应用程序==

- 启动flume

```
flume-ng agent \
--name simple-agent \
--conf $FLUME_HOME/conf \
--conf-file $FLUME_HOME/conf/flume_pull_streaming.conf \
-Dflume.root.logger=INFO,console
```

- 启动spark-streaming

```
spark-submit --master local[2] \
--class com.jungle.spark.FlumePullWordCount \
--packages org.apache.spark:spark-streaming-flume_2.11:2.2.0 \
/home/jungle/lib/sparktrain-1.0.0-SNAPSHOT.jar \
centosserver1 41414
```

第10章 Spark Streaming整合Kafka

一、Receiver方式整合之Kafka

1. 开启zookeeper
2. 开启Kafka
3. 创建topic

```
kafka-topics.sh --create --zookeeper centosserver1:2181 --replication-factor 1 --partitions 1 --topic kafka_streaming_topic
```

```
cd $KAFKA_HOME/bin  
./kafka-topics.sh --list --zookeeper centosserver1:2181
```

4. 通过控制台测试topic是否能够正常生产和消费

```
kafka-console-producer.sh --broker-list centosserver1:9092 --topic kafka_streaming_topic
```

```
kafka-console-consumer.sh --zookeeper centosserver1:2181 --topic kafka_streaming_topic --from-beginning
```

二、Spark Streaming应用开发

- 添加依赖

```
<dependency>  
    <groupId>org.apache.spark</groupId>  
    <artifactId>spark-streaming-kafka-0-8_2.11</artifactId>  
    <version>${spark.version}</version>  
</dependency>
```

- KafkaReceiverWordCount

```
package com.jungle.spark

import org.apache.spark.SparkConf
import org.apache.spark.streaming.kafka.KafkaUtils
import org.apache.spark.streaming.{Seconds, StreamingContext}

/**
 * Spark streaming对接kafka的方式一
 */
object KafkaReceiverWordCount {

    def main(args: Array[String]): Unit = {

        if(args.length != 4) {
            System.err.println("Usage: KafkaReceiverwordCount <zkQuorum> <group>
<topics> <numThreads>")
        }

        val Array(zkQuorum, group, topics, numThreads) = args

        val sparkConf = new SparkConf().setAppName("KafkaReceiverwordCount")
            .setMaster("local[2]")

        val ssc = new StreamingContext(sparkConf, Seconds(5))

        val topicMap = topics.split(",").map(_ , numThreads.toInt)).toMap
```

```

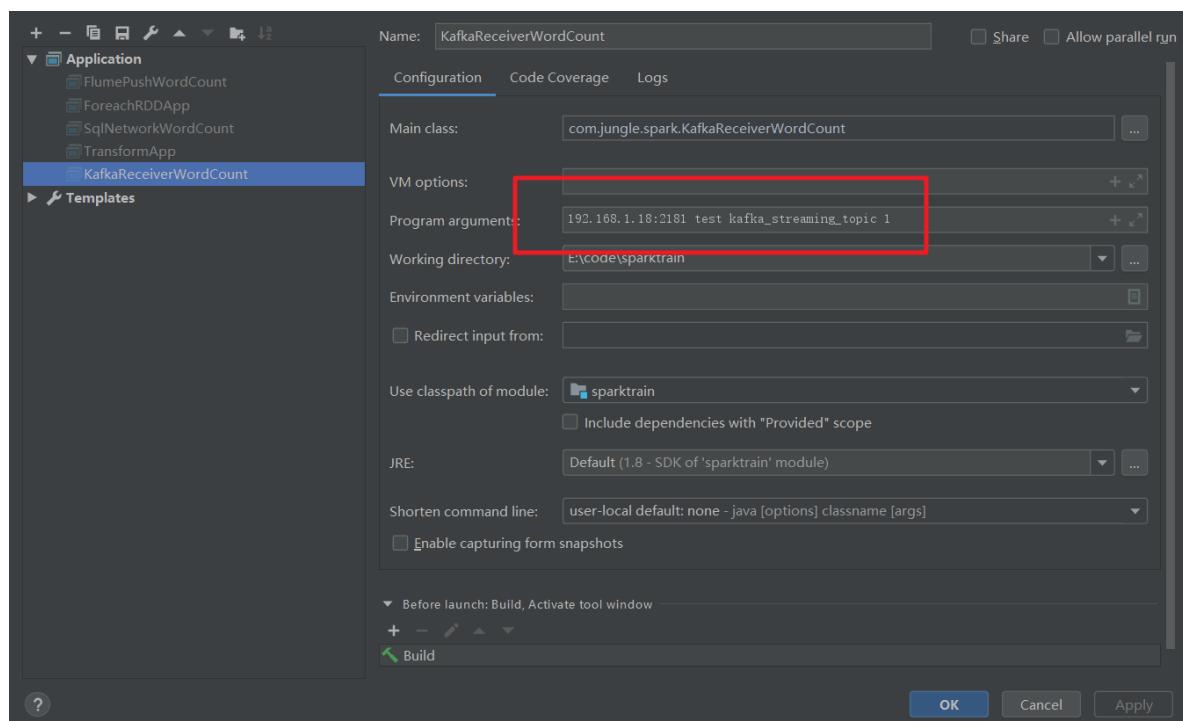
// TODO... Spark streaming如何对接Kafka
val messages = KafkaUtils.createStream(ssc, zkQuorum, group, topicMap)

// TODO... 自己去测试为什么要取第二个
messages.map(_.value).flatMap(_.split(","))
  .map((_, 1)).reduceByKey(_ + _).print()

ssc.start()
ssc.awaitTermination()
}
}

```

1.本地测试



```

-bash: ./kafka-topics.sh: No such file or directory
jungle@centosserver1:[/home/jungle] cd $KAFKA_HOME/bin
jungle@centosserver1:[/home/jungle/app/kafka_2.11-0.9.0.0/bin] ./kafka-topics.sh --list --
zookeeper centosserver1:2181
hello_topic
kafka_streaming_topic
my-replicated-topic
jungle@centosserver1:[/home/jungle/app/kafka_2.11-0.9.0.0/bin] kafka-console-producer.sh
--broker-list centosserver1:9092 --topic kafka_streaming_topic
qwe
asd
zxc
^Hqx wwdw qw sq
w e r t y s a x d

```

```
KafkaReceiverWordCount
19/08/04 16:36:20 INFO scheduler.DAGScheduler: Job ...
-----
Time: 1564907780000 ms
-----
(d,1)
(x,1)
(t,1)
(r,1)
(w,1)
(s,1)
(e,2)
(a,1)
(y,1)
```

2.服务器环境联调

1. 程序

```
package com.jungle.spark

import org.apache.spark.SparkConf
import org.apache.spark.streaming.kafka.KafkaUtils
import org.apache.spark.streaming.{Seconds, StreamingContext}

/**
 * Spark Streaming对接Kafka的方式一
 */
object KafkaReceiverWordCount {

    def main(args: Array[String]): Unit = {

        if(args.length != 4) {
            System.err.println("Usage: KafkaReceiverWordCount <zkQuorum> <group>
<topics> <numThreads>")
        }

        val Array(zkQuorum, group, topics, numThreads) = args

        val sparkConf = new SparkConf()//.setAppName("KafkaReceiverWordCount")
            //.setMaster("local[2]")

        val ssc = new StreamingContext(sparkConf, Seconds(5))

        val topicMap = topics.split(",").map(_ , numThreads.toInt)).toMap

        // TODO... Spark Streaming如何对接Kafka
        val messages = KafkaUtils.createStream(ssc, zkQuorum, group,topicMap)
```

```

    // TODO... 自己去测试为什么要取第二个
    messages.map(_._2).flatMap(_.split(""))
)).map((_,1)).reduceByKey(_+_).print()

    ssc.start()
    ssc.awaitTermination()
}
}

```

2. 打包

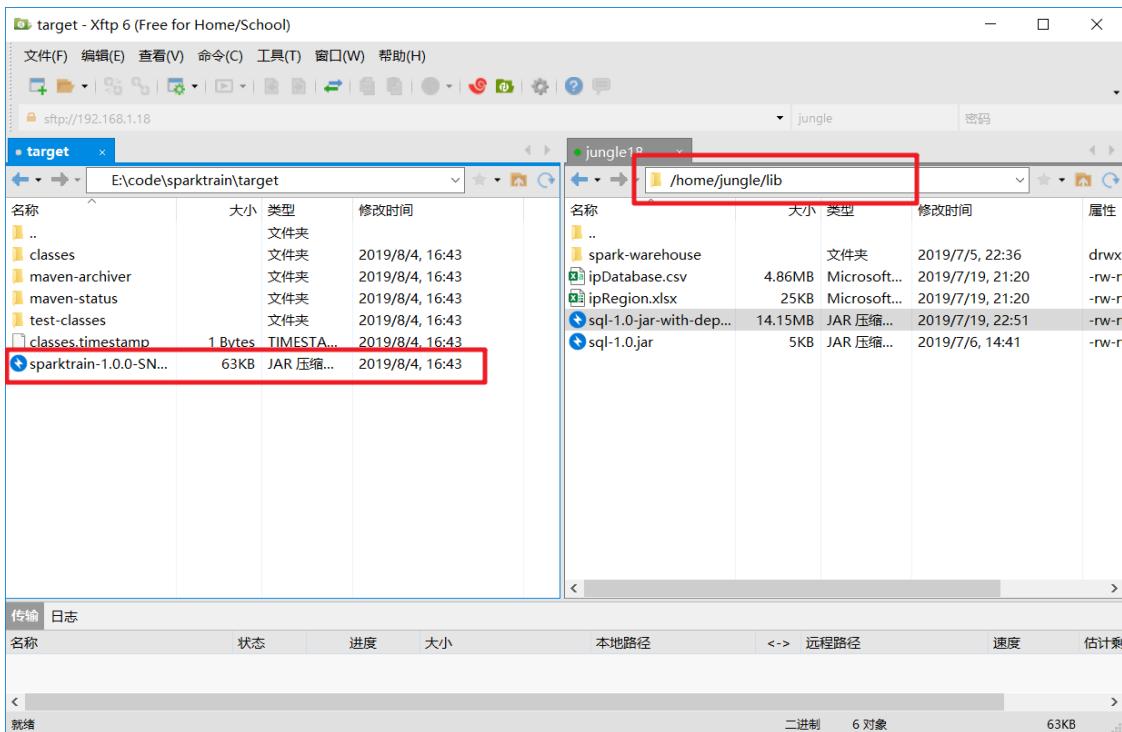
```
mvn clean package -DskipTests
```

Terminal: Local +

Microsoft Windows [版本 10.0.17134.885]
(c) 2018 Microsoft Corporation。保留所有权利。

E:\code\sparktrain>mvn clean package -DskipTests

3. 上传服务器



4. 运行脚本

```

spark-submit \
--class com.jungle.spark.KafkaReceiverWordCount \
--master local[2] \
--name KafkaReceiverWordCount \
--packages org.apache.spark:spark-streaming-kafka-0-8_2.11:2.2.0 \
/home/jungle/lib/sparktrain-1.0.0-SNAPSHOT.jar centosserver1:2181 test
kafka_streaming_topic 1

```

5. UI界面

<http://192.168.1.18:4040/jobs/>

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
0	Streaming job running receiver 0 start at KafkaReceiverWordCount.scala:33	2019/08/04 16:52:23 (kill)	7.2 min	0/1	0/1

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
174	Streaming job from [output operation 0, batch time 16:59:35] print at KafkaReceiverWordCount.scala:31	2019/08/04 16:59:35	7 ms	1/1 (1 skipped)	1/1
173	Streaming job from [output operation 0, batch time 16:59:35] print at KafkaReceiverWordCount.scala:31	2019/08/04 16:59:35	8 ms	1/1 (1 skipped)	1/1
172	Streaming job from [output operation 0, batch time 16:59:30] print at KafkaReceiverWordCount.scala:31	2019/08/04 16:59:30	6 ms	1/1 (1 skipped)	1/1
171	Streaming job from [output operation 0, batch time 16:59:30] print at KafkaReceiverWordCount.scala:31	2019/08/04 16:59:30	7 ms	1/1 (1 skipped)	1/1
170	Streaming job from [output operation 0, batch time 16:59:25]	2019/08/04 16:59:25	8 ms	1/1 (1 skipped)	1/1

三、Direct方式整合

==服务器环境运行==

1. 程序

```

package com.imooc.spark

import org.apache.spark.SparkConf
import org.apache.spark.streaming.kafka.KafkaUtils
import org.apache.spark.streaming.{Seconds, StreamingContext}
import kafka.serializer.StringDecoder
/***
 * Spark streaming对接kafka的方式二
 */
object KafkaDirectWordCount {

  def main(args: Array[String]): Unit = {

    if(args.length != 2) {
      System.err.println("Usage: KafkaDirectWordCount <brokers> <topics>")
      System.exit(1)
    }
  }
}

```

```

}

val Array(brokers, topics) = args

val sparkConf = new SparkConf() //.setAppName("KafkaReceiverWordCount")
//.setMaster("local[2]")

val ssc = new StreamingContext(sparkConf, Seconds(5))

val topicsSet = topics.split(",").toSet
val kafkaParams = Map[String, String]("metadata.broker.list" -> brokers)

// TODO... Spark Streaming如何对接Kafka
val messages =
KafkaUtils.createDirectStream[String, String, StringDecoder, StringDecoder](
  ssc, kafkaParams, topicsSet
)

// TODO... 自己去测试为什么要取第二个
messages.map(_.value).flatMap(_.split(","))
  .map((_, 1)).reduceByKey(_ + _).print()

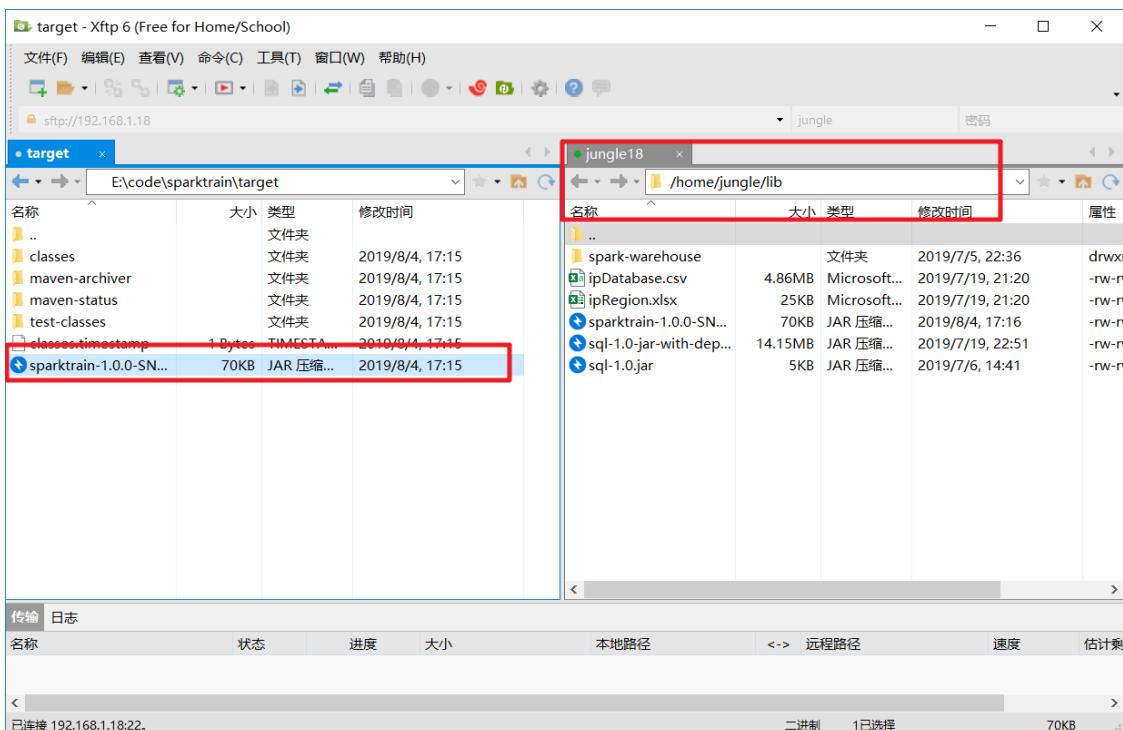
ssc.start()
ssc.awaitTermination()
}
}

```

2. 打包

```
mvn clean package -DskipTests
```

3. 上传服务器



4. 运行脚本

```
spark-submit \
--class com.imooc.spark.KafkaDirectWordCount \
--master local[2] \
--name KafkaDirectWordCount \
--packages org.apache.spark:spark-streaming-kafka-0-8_2.11:2.2.0 \
/home/jungle/lib/sparktrain-1.0.0-SNAPSHOT.jar centosserver1:9092
kafka_streaming_topic
```

5. 效果

```
jungle@centosserver1:[/home/jungle/app/kafka_2.11-0.9.0.0/bin]kafka-console-producer.sh
--broker-list centosserver1:9092 --topic kafka_streaming_topic
qwe
asd
zxc
^Hqx wwdw qw sq
w e e r t y s a x d
w w w e e r t y r q w rr t ee w e qw
q we r try tyh6h sf adw ededf f
[1]
19/08/04 17:22:00 INFO TaskSchedulerImpl: Removed TaskSet 15.0, v
19/08/04 17:22:00 INFO DAGScheduler: ResultStage 15 (print at K
19/08/04 17:22:00 INFO DAGScheduler: Job 7 finished: print at K
-----
Time: 1564910520000 ms
-----
(we,1)
(f,1)
(r,1)
(adw,1)
(ededf,1)
(q,1)
(try,1)
(tyh6h,1)
(sf,1)

19/08/04 17:22:00 INFO JobScheduler: Finished job streaming job
19/08/04 17:22:00 INFO JobScheduler: Total delay: 0.152 s for tir
```

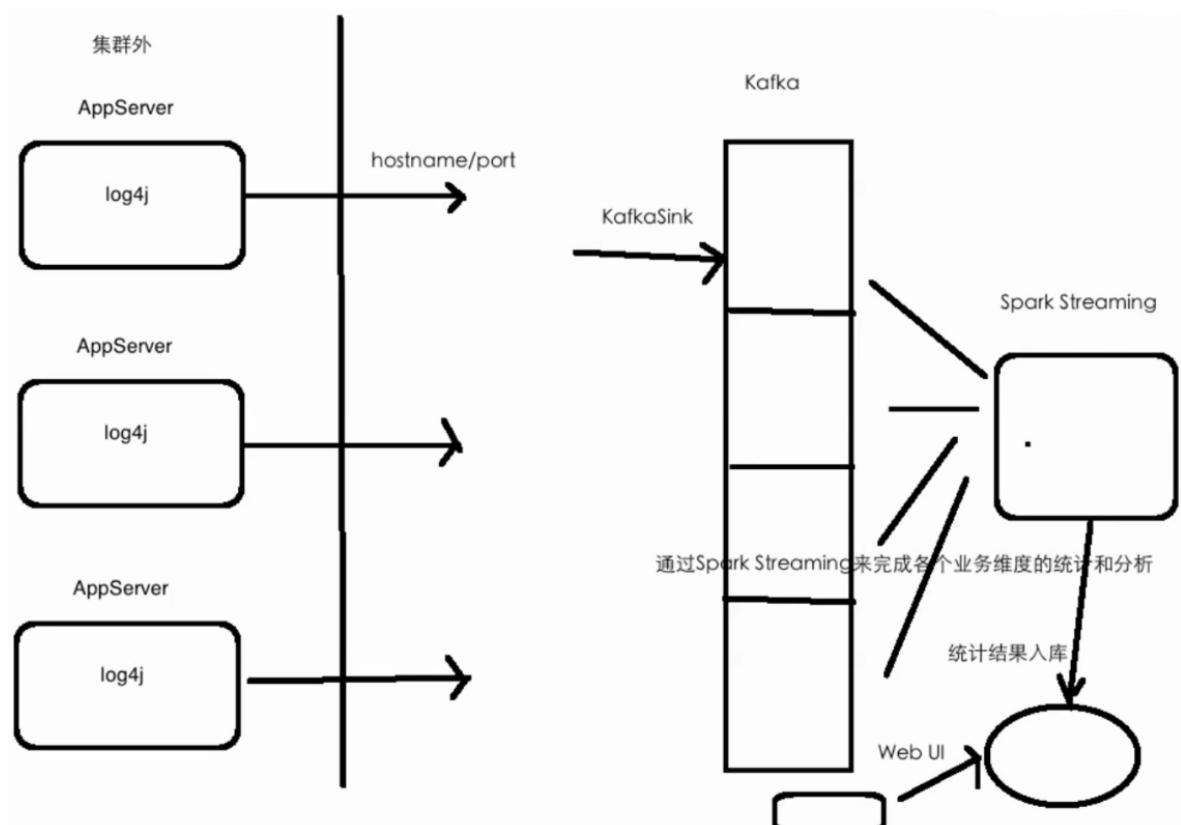
第11章 Spark Streaming整合 Flume&Kafka打造通用流处理基础

一、课程目录

基于Spark Streaming&Flume&Kafka打造通用流处理平台

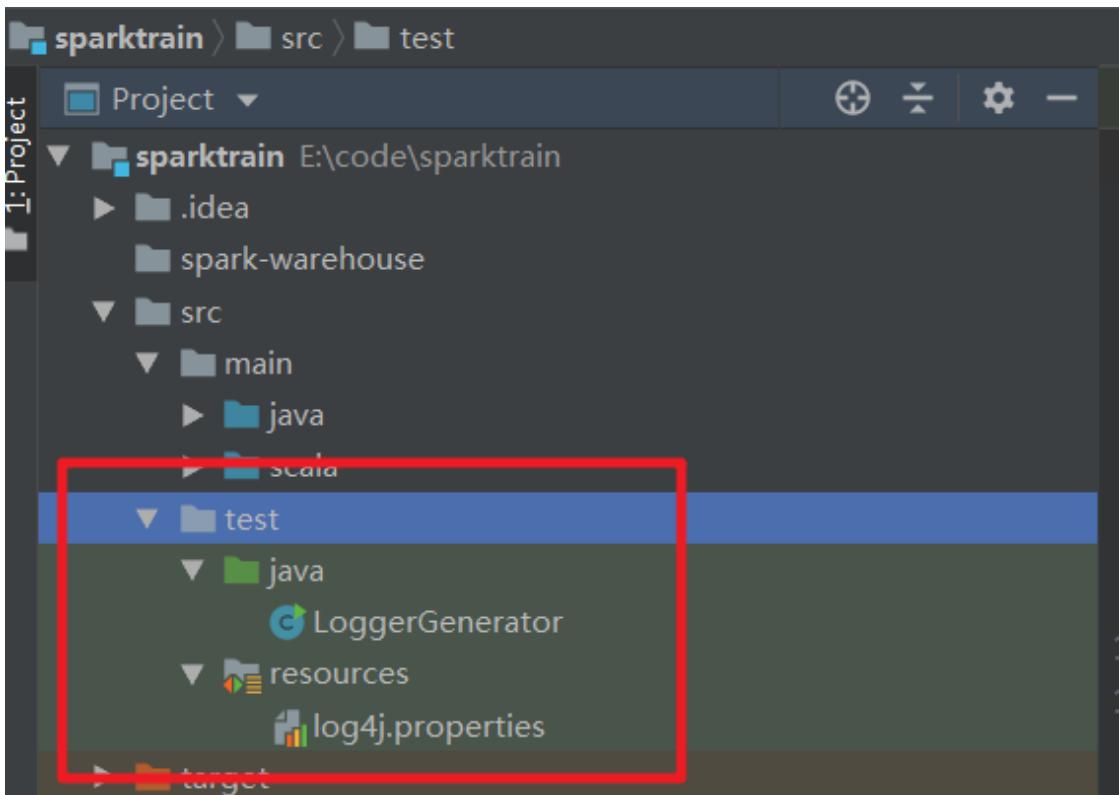
- ◆ 整合日志输出到Flume
- ◆ 整合Flume到Kafka
- ◆ 整合Kafka到Spark Streaming
- ◆ Spark Streaming对接收到的数据进行处理

二、处理流程画图剖析



三、日志产生器开发并结合log4j完成日志的输出

- 目录结构



- LoggerGenerator

```
import org.apache.log4j.Logger;
```

```

/**
 * 模拟日志产生
 */
public class LoggerGenerator {

    private static Logger logger =
Logger.getLogger(LoggerGenerator.class.getName());

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

        int index = 0;
        while(true) {
            //启动一个线程，休息一下
            Thread.sleep(1000);
            logger.info("value : " + index++);
        }
    }
}

```

- log4j.properties

```

log4j.rootLogger=INFO,stdout

log4j.appender.stdout = org.apache.log4j.ConsoleAppender
log4j.appender.stdout.target = System.out
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss,SSS}
[%t] [%c] [%p] - %m%n

```

- 效果

```

2019-08-05 15:33:24,244 [main] [LoggerGenerator] [INFO] - value : 139
2019-08-05 15:33:25,244 [main] [LoggerGenerator] [INFO] - value : 140
2019-08-05 15:33:26,245 [main] [LoggerGenerator] [INFO] - value : 141
2019-08-05 15:33:27,245 [main] [LoggerGenerator] [INFO] - value : 142
2019-08-05 15:33:28,245 [main] [LoggerGenerator] [INFO] - value : 143
2019-08-05 15:33:29,245 [main] [LoggerGenerator] [INFO] - value : 144
2019-08-05 15:33:30,246 [main] [LoggerGenerator] [INFO] - value : 145
2019-08-05 15:33:31,246 [main] [LoggerGenerator] [INFO] - value : 146
2019-08-05 15:33:32,246 [main] [LoggerGenerator] [INFO] - value : 147
2019-08-05 15:33:33,247 [main] [LoggerGenerator] [INFO] - value : 148
2019-08-05 15:33:34,247 [main] [LoggerGenerator] [INFO] - value : 149
2019-08-05 15:33:35,248 [main] [LoggerGenerator] [INFO] - value : 150
2019-08-05 15:33:36,248 [main] [LoggerGenerator] [INFO] - value : 151
2019-08-05 15:33:37,248 [main] [LoggerGenerator] [INFO] - value : 152
2019-08-05 15:33:38,249 [main] [LoggerGenerator] [INFO] - value : 153
2019-08-05 15:33:39,249 [main] [LoggerGenerator] [INFO] - value : 154
2019-08-05 15:33:40,250 [main] [LoggerGenerator] [INFO] - value : 155
2019-08-05 15:33:41,251 [main] [LoggerGenerator] [INFO] - value : 156
2019-08-05 15:33:42,252 [main] [LoggerGenerator] [INFO] - value : 157
2019-08-05 15:33:43,252 [main] [LoggerGenerator] [INFO] - value : 158
2019-08-05 15:33:44,252 [main] [LoggerGenerator] [INFO] - value : 159

```

四、使用Flume采集Log4j产生的日志

- streaming.conf

```

agent1.sources=avro-source
agent1.channels=logger-channel
agent1.sinks=log-sink

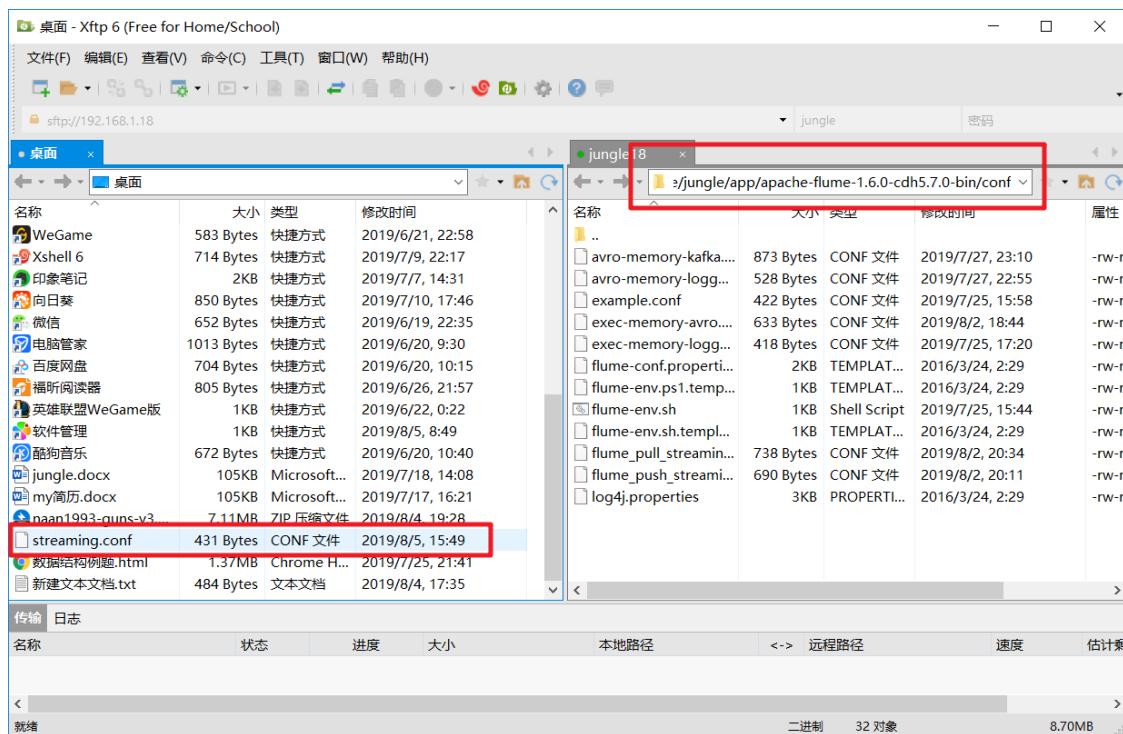
#define source
agent1.sources.avro-source.type=avro
agent1.sources.avro-source.bind=0.0.0.0
agent1.sources.avro-source.port=41414

#define channel
agent1.channels.logger-channel.type=memory

#define sink
agent1.sinks.log-sink.type=logger

agent1.sources.avro-source.channels=logger-channel
agent1.sinks.log-sink.channel=logger-channel

```



- 启动flume

```

flume-ng agent \
--conf $FLUME_HOME/conf \
--conf-file $FLUME_HOME/conf/streaming.conf \
--name agent1 \
-Dflume.root.logger=INFO,console

```

- 修改log4j.properties

```

log4j.rootLogger=INFO,stdout,flume

log4j.appender.stdout = org.apache.log4j.ConsoleAppender
log4j.appender.stdout.target = System.out
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss,SSS}
[%t] [%c] [%p] - %m%n

log4j.appender.flume = org.apache.flume.clients.log4jappender.Log4jAppender
log4j.appender.flume.Hostname = 192.168.1.18
log4j.appender.flume.Port = 41414
log4j.appender.flume.UnsafeMode = true

```

- 添加依赖

```

<dependency>
    <groupId>org.apache.flume.flume-ng-clients</groupId>
    <artifactId>flume-ng-log4jappender</artifactId>
    <version>1.6.0</version>
</dependency>

```

- 程序

```

import org.apache.log4j.Logger;

/**
 * 模拟日志产生
 */
public class LoggerGenerator {

    private static Logger logger =
Logger.getLogger(LoggerGenerator.class.getName());

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

        int index = 0;
        while(true) {
            //启动一个线程，休息一下
            Thread.sleep(1000);
            logger.info("value : " + index++);
        }
    }
}

```

- 效果

```

"Program Files (x86)\Java\jdk1.8.0_181\bin\java.exe" ...
log4j:ERROR Could not find value for key log4j.appenders.flume.layout
2019-08-05 16:21:52,433 [main] [org.apache.flume.api.NettyAvroRpcClient] [WARN] - Us
2019-08-05 16:21:54,045 [main] [LoggerGenerator] [INFO] - value : 0
2019-08-05 16:21:55,083 [main] [LoggerGenerator] [INFO] - value : 1
2019-08-05 16:21:56,090 [main] [LoggerGenerator] [INFO] - value : 2
2019-08-05 16:21:57,105 [main] [LoggerGenerator] [INFO] - value : 3
2019-08-05 16:21:58,114 [main] [LoggerGenerator] [INFO] - value : 4
2019-08-05 16:21:59,118 [main] [LoggerGenerator] [INFO] - value : 5
2019-08-05 16:22:00,126 [main] [LoggerGenerator] [INFO] - value : 6
2019-08-05 16:22:01,132 [main] [LoggerGenerator] [INFO] - value : 7
2019-08-05 16:22:02,141 [main] [LoggerGenerator] [INFO] - value : 8
2019-08-05 16:22:03,225 [main] [LoggerGenerator] [INFO] - value : 9
2019-08-05 16:22:04,238 [main] [LoggerGenerator] [INFO] - value : 10
2019-08-05 16:22:05,249 [main] [LoggerGenerator] [INFO] - value : 11
2019-08-05 16:22:06,257 [main] [LoggerGenerator] [INFO] - value : 12
2019-08-05 16:22:07,264 [main] [LoggerGenerator] [INFO] - value : 13
2019-08-05 16:22:08,270 [main] [LoggerGenerator] [INFO] - value : 14
2019-08-05 16:22:09,278 [main] [LoggerGenerator] [INFO] - value : 15
2019-08-05 16:22:10,298 [main] [LoggerGenerator] [INFO] - value : 16
2019-08-05 16:22:11,303 [main] [LoggerGenerator] [INFO] - value : 17
2019-08-05 16:22:12,310 [main] [LoggerGenerator] [INFO] - value : 18

2019-08-05 16:25:05,064 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.apache.flume.sink.LoggerSink.process(LoggerSink.java:94)] Event: { headers:[flume.client.log4j.timestamp=1564993505592, flume.client.log4j.logger.name=LoggerGenerator, flume.client.log4j.log.level=20000, flume.client.log4j.message.encoding=UTF8] body: 76 61 6C 75 65 20 3A 20 31 39 30 value : 190 }
2019-08-05 16:25:06,071 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.apache.flume.sink.LoggerSink.process(LoggerSink.java:94)] Event: { headers:[flume.client.log4j.timestamp=1564993506600, flume.client.log4j.logger.name=LoggerGenerator, flume.client.log4j.log.level=20000, flume.client.log4j.message.encoding=UTF8] body: 76 61 6C 75 65 20 3A 20 31 39 31 value : 191 }
2019-08-05 16:25:07,078 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.apache.flume.sink.LoggerSink.process(LoggerSink.java:94)] Event: { headers:[flume.client.log4j.timestamp=1564993507606, flume.client.log4j.logger.name=LoggerGenerator, flume.client.log4j.log.level=20000, flume.client.log4j.message.encoding=UTF8] body: 76 61 6C 75 65 20 3A 20 31 39 32 value : 192 }
2019-08-05 16:25:08,084 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.apache.flume.sink.LoggerSink.process(LoggerSink.java:94)] Event: { headers:[flume.client.log4j.timestamp=1564993508613, flume.client.log4j.logger.name=LoggerGenerator, flume.client.log4j.log.level=20000, flume.client.log4j.message.encoding=UTF8] body: 76 61 6C 75 65 20 3A 20 31 39 33 value : 193 }
2019-08-05 16:25:09,091 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.apache.flume.sink.LoggerSink.process(LoggerSink.java:94)] Event: { headers:[flume.client.log4j.timestamp=1564993509623, flume.client.log4j.logger.name=LoggerGenerator, flume.client.log4j.log.level=20000, flume.client.log4j.message.encoding=UTF8] body: 76 61 6C 75 65 20 3A 20 31 39 34 value : 194 }
2019-08-05 16:25:10,099 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.apache.flume.sink.LoggerSink.process(LoggerSink.java:94)] Event: { headers:[flume.client.log4j.timestamp=1564993510627, flume.client.log4j.logger.name=LoggerGenerator, flume.client.log4j.log.level=20000, flume.client.log4j.message.encoding=UTF8] body: 76 61 6C 75 65 20 3A 20 31 39 35 value : 195 }
2019-08-05 16:25:11,104 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.apache.flume.sink.LoggerSink.process(LoggerSink.java:94)] Event: { headers:[flume.client.log4j.timestamp=1564993511636, flume.client.log4j.logger.name=LoggerGenerator, flume.client.log4j.log.level=20000, flume.client.log4j.message.encoding=UTF8] body: 76 61 6C 75 65 20 3A 20 31 39 36 value : 196 }
2019-08-05 16:25:12,111 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.apache.flume.sink.LoggerSink.process(LoggerSink.java:94)] Event: { headers:[flume.client.log4j.timestamp=1564993512640, flume.client.log4j.logger.name=LoggerGenerator, flume.client.log4j.log.level=20000, flume.client.log4j.message.encoding=UTF8] body: 76 61 6C 75 65 20 3A 20 31 39 37 value : 197 }
2019-08-05 16:25:13,119 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.apache.flume.sink.LoggerSink.process(LoggerSink.java:94)] Event: { headers:[flume.client.log4j.timestamp=1564993513648, flume.client.log4j.logger.name=LoggerGenerator, flume.client.log4j.log.level=20000, flume.client.log4j.message.encoding=UTF8] body: 76 61 6C 75 65 20 3A 20 31 39 38 value : 198 }
2019-08-05 16:25:14,125 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.apache.flume.sink.LoggerSink.process(LoggerSink.java:94)] Event: { headers:[flume.client.log4j.timestamp=1564993514654, flume.client.log4j.logger.name=LoggerGenerator, flume.client.log4j.log.level=20000, flume.client.log4j.message.encoding=UTF8] body: 76 61 6C 75 65 20 3A 20 31 39 39 value : 199 }

```

五、使用KafkaSink将Flume收集到的数据输出到Kafka

1. 启动zookeeper
2. 启动Kafka
3. 创建topic

```
kafka-topics.sh --create --zookeeper centosserver1:2181 --replication-factor 1 --partitions 1 --topic streamingtopic
```

```
kafka-topics.sh --list --zookeeper centosserver1:2181
```

4. streaming2.conf

```

agent1.sources=avro-source
agent1.channels=logger-channel
agent1.sinks=kafka-sink

#define source
agent1.sources.avro-source.type=avro
agent1.sources.avro-source.bind=0.0.0.0
agent1.sources.avro-source.port=41414

```

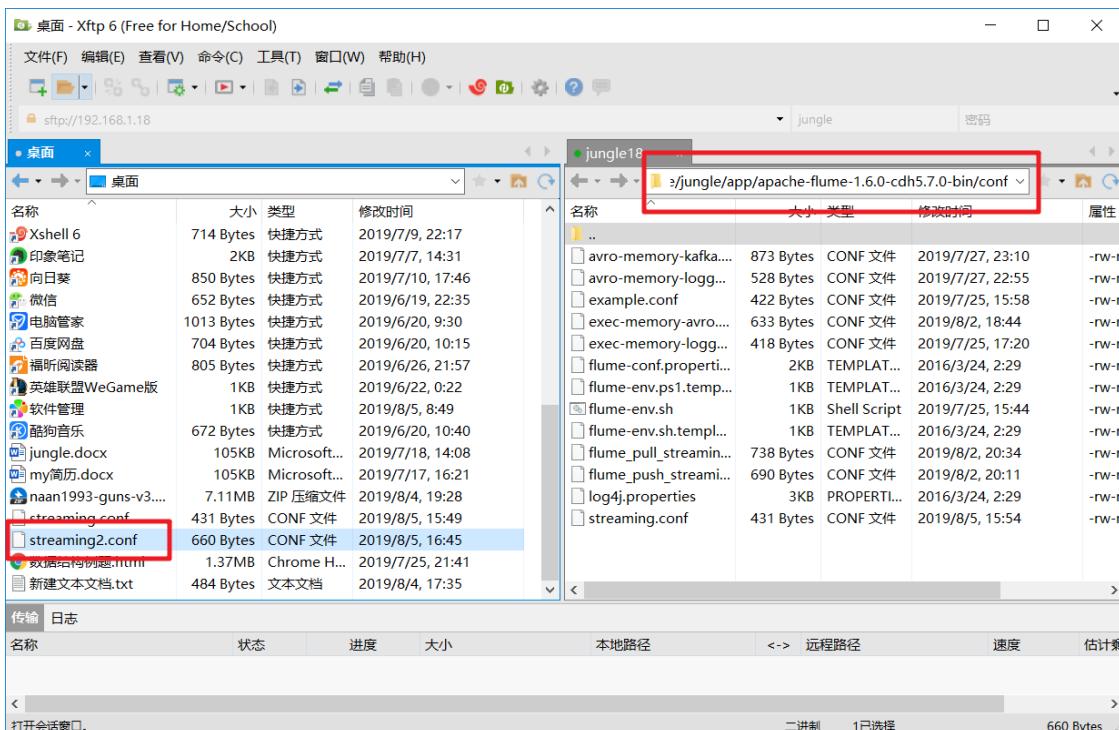
```

#define channel
agent1.channels.logger-channel.type=memory

#define sink
agent1.sinks.kafka-sink.type=org.apache.flume.sink.kafka.KafkaSink
agent1.sinks.kafka-sink.topic = streamingtopic
agent1.sinks.kafka-sink.brokerList = centosserver1:9092
agent1.sinks.kafka-sink..requiredAcks = 1
agent1.sinks.kafka-sink.batchSize = 20

agent1.sources.avro-source.channels=logger-channel
agent1.sinks.kafka-sink.channel=logger-channel

```



5. 启动flume

```

flume-ng agent \
--conf $FLUME_HOME/conf \
--conf-file $FLUME_HOME/conf/streaming2.conf \
--name agent1 \
-Dflume.root.logger=INFO,console

```

6. Kafka消费端

```

kafka-console-consumer.sh --zookeeper centosserver1:2181 --topic
streamingtopic

```

7. 效果

```

LoggerGenerator x
"C:\Program Files (x86)\Java\jdk1.8.0_181\bin\java.exe" ...
log4j:ERROR Could not find value for key log4j.appenders.flume.layout
2019-08-05 16:51:01,016 [main] [org.apache.flume.api.NettyAvroRpcClient] [
2019-08-05 16:51:02,595 [main] [LoggerGenerator] [INFO] - value : 0
2019-08-05 16:51:03,669 [main] [LoggerGenerator] [INFO] - value : 1
2019-08-05 16:51:04,677 [main] [LoggerGenerator] [INFO] - value : 2
2019-08-05 16:51:05,684 [main] [LoggerGenerator] [INFO] - value : 3
2019-08-05 16:51:06,692 [main] [LoggerGenerator] [INFO] - value : 4
2019-08-05 16:51:07,702 [main] [LoggerGenerator] [INFO] - value : 5
2019-08-05 16:51:08,712 [main] [LoggerGenerator] [INFO] - value : 6
2019-08-05 16:51:09,721 [main] [LoggerGenerator] [INFO] - value : 7
2019-08-05 16:51:10,733 [main] [LoggerGenerator] [INFO] - value : 8
2019-08-05 16:51:11,740 [main] [LoggerGenerator] [INFO] - value : 9
2019-08-05 16:51:12,750 [main] [LoggerGenerator] [INFO] - value : 10
2019-08-05 16:51:13,758 [main] [LoggerGenerator] [INFO] - value : 11
2019-08-05 16:51:14,766 [main] [LoggerGenerator] [INFO] - value : 12
2019-08-05 16:51:15,770 [main] [LoggerGenerator] [INFO] - value : 13
2019-08-05 16:51:16,779 [main] [LoggerGenerator] [INFO] - value : 14
2019-08-05 16:51:17,786 [main] [LoggerGenerator] [INFO] - value : 15
2019-08-05 16:51:18,792 [main] [LoggerGenerator] [INFO] - value : 16
2019-08-05 16:51:19,798 [main] [LoggerGenerator] [INFO] - value : 17
2019-08-05 16:51:20,806 [main] [LoggerGenerator] [INFO] - value : 18
jungle@centosserver1:[/home/jungle/app]kafka-console-consumer.sh --zookeeper centosserver1:2181 --topic streamingtopic
value : 0
value : 1
value : 2
value : 3
value : 4
value : 5
value : 6
value : 7
value : 8
value : 9
value : 10
value : 11
value : 12
value : 13
value : 14
value : 15
value : 16
value : 17
value : 18
value : 19

```

六、Spark Streaming消费Kafka的数据进行统计

1. KafkaStreamingApp

```

package com.jungle.spark

import org.apache.spark.SparkConf
import org.apache.spark.streaming.kafka.KafkaUtils
import org.apache.spark.streaming.{Seconds, StreamingContext}

/**
 * Spark Streaming对接Kafka
 */
object KafkaStreamingApp {

    def main(args: Array[String]): Unit = {

        if(args.length != 4) {
            System.err.println("Usage: KafkaStreamingApp <zKQuorum> <group>
<topics> <numThreads>")
        }

        val Array(zkQuorum, group, topics, numThreads) = args

        val sparkConf = new SparkConf().setAppName("KafkaReceiverWordCount")
    }
}

```

```

.setMaster("local[2]")

val ssc = new StreamingContext(sparkConf, Seconds(5))

val topicMap = topics.split(",").map(_ , numThreads.toInt)).toMap

// TODO... Spark Streaming如何对接Kafka
val messages = KafkaUtils.createStream(ssc, zkQuorum, group,topicMap)

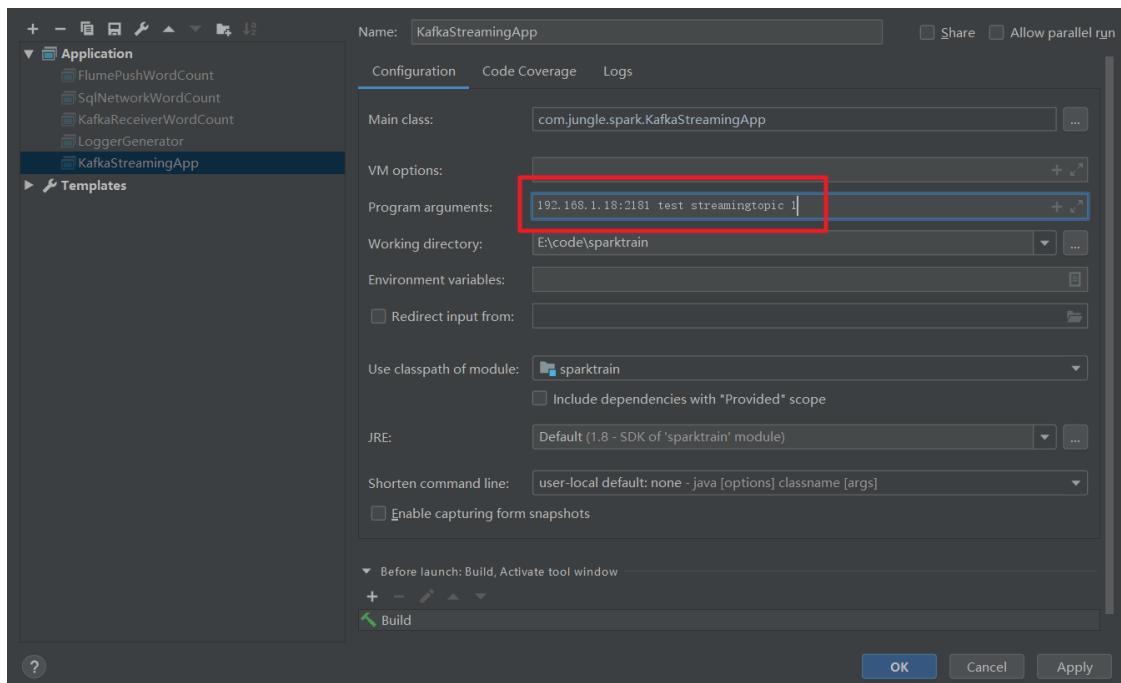
// TODO... 自己去测试为什么要取第二个
messages.map(_._2).count().print()

ssc.start()
ssc.awaitTermination()
}

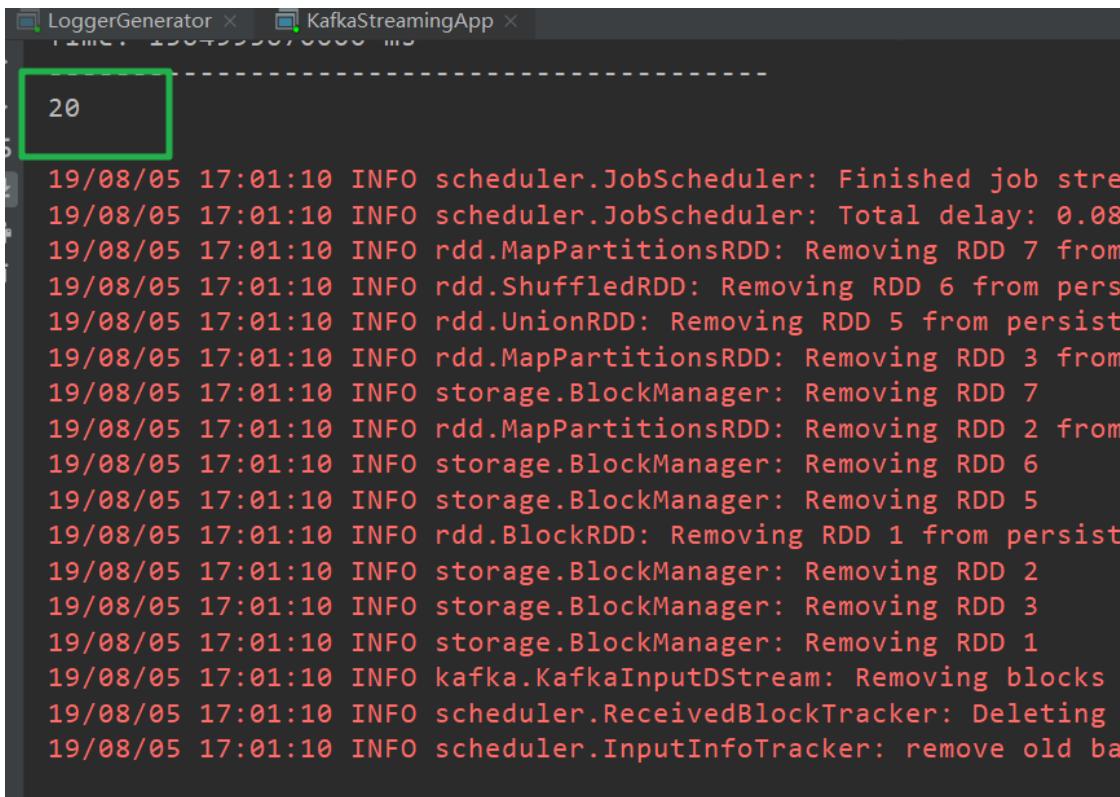
}

```

2. 本地测试



192.168.1.18:2181 test streamingtopic 1



```
20
19/08/05 17:01:10 INFO scheduler.JobScheduler: Finished job stre
19/08/05 17:01:10 INFO scheduler.JobScheduler: Total delay: 0.08
19/08/05 17:01:10 INFO rdd.MapPartitionsRDD: Removing RDD 7 from
19/08/05 17:01:10 INFO rdd.ShuffledRDD: Removing RDD 6 from pers
19/08/05 17:01:10 INFO rdd.UnionRDD: Removing RDD 5 from persist
19/08/05 17:01:10 INFO rdd.MapPartitionsRDD: Removing RDD 3 from
19/08/05 17:01:10 INFO storage.BlockManager: Removing RDD 7
19/08/05 17:01:10 INFO rdd.MapPartitionsRDD: Removing RDD 2 from
19/08/05 17:01:10 INFO storage.BlockManager: Removing RDD 6
19/08/05 17:01:10 INFO storage.BlockManager: Removing RDD 5
19/08/05 17:01:10 INFO rdd.BlockRDD: Removing RDD 1 from persist
19/08/05 17:01:10 INFO storage.BlockManager: Removing RDD 2
19/08/05 17:01:10 INFO storage.BlockManager: Removing RDD 3
19/08/05 17:01:10 INFO storage.BlockManager: Removing RDD 1
19/08/05 17:01:10 INFO kafka.KafkaInputDStream: Removing blocks
19/08/05 17:01:10 INFO scheduler.ReceivedBlockTracker: Deleting
19/08/05 17:01:10 INFO scheduler.InputInfoTracker: remove old ba
```

七、本地测试和生产环境使用的拓展

我们现在是在本地进行测试的，在IDEA中运行LoggerGenerator，然后使用Flume、Kafka以及Spark Streaming进行处理操作。

在生产上肯定不是这么干的，怎么干呢？

- 1) 打包jar，执行LoggerGenerator类
- 2) Flume、Kafka和我们的测试是一样的
- 3) Spark Streaming的代码也是需要打成jar包，然后使用spark-submit的方式
可以根据你们的实际情况选择运行模式：local/yarn/standalone/mesos

在生产上，整个流处理的流程都一样的，区别在于业务逻辑的复杂性

第12章 Spark Streaming项目实战

一、Python日志产生器开发

1. 产生访问url和ip信息

```
#coding=UTF-8
import random

url_paths = [
    "class/112.html",
    "class/118.html",
    "class/145.html",
    "class/146.html",
    "class/131.html",
    "class/110.html",
    "learn/821",
    "course/list"
```

```

]

ip_slices = [132,156,124,10,29,167,143,187,30,46,55,63,72,87,98,168]

def sample_url():
    return random.sample(url_paths,1)[0]

def sample_ip():
    slice = random.sample(ip_slices,4)
    return ".".join([str(item) for item in slice])

def generate_log(count = 10):
    while count >= 1:
        query_log = "{url}\t{ip}".format(url=sample_url(),ip=sample_ip())
        print(query_log)
        count = count -1

if __name__=='__main__':
    generate_log()

```

2.产生referer和状态码信息

```

#coding=UTF-8
import random

url_paths = [
    "class/112.html",
    "class/118.html",
    "class/145.html",
    "class/146.html",
    "class/131.html",
    "class/110.html",
    "learn/821",
    "course/list"
]

ip_slices = [132,156,124,10,29,167,143,187,30,46,55,63,72,87,98,168]

http_referers =[

    "http://www.baidu.com/s?wd={query}",
    "https://www.sogou.com/web?query={query}",
    "http://cn.bing.com/search?q={query}",
    "http://www.baidu.com/s?wd={query}",
    "https://search.yahoo.com/search?p={query}",
]

search_keyword = [
    "Spark实战",
    "Hadoop基础",
    "Storm实战",
    "Spark streaming实战",
    "大数据面试"
]

status_codes = ["200","404","500"]

def sample_url():
    return random.sample(url_paths,1)[0]

```

```

def sample_ip():
    slice = random.sample(ip_slices, 4)
    return ".".join([str(item) for item in slice])

def sample_referer():
    if random.uniform(0,1) > 0.2:
        return "-"

    refer_str = random.sample(http_referers, 1)
    query_str = random.sample(search_keyword, 1)
    return refer_str[0].format(query=query_str[0])

def sample_status_code():
    return random.sample(status_codes, 1)[0]

def generate_log(count = 10):
    while count >= 1:
        query_log = "{url}\t{ip}\t{referer}\t{status_code}".format(url=sample_url(), ip=sample_ip(), referer=sample_referer(), status_code=sample_status_code())
        print(query_log)
        count = count -1

if __name__=='__main__':
    generate_log(100)

```

3.产生日志访问时间

```

#coding=UTF-8
import random
import time

url_paths = [
    "class/112.html",
    "class/118.html",
    "class/145.html",
    "class/146.html",
    "class/131.html",
    "class/110.html",
    "learn/821",
    "course/list"
]

ip_slices = [132,156,124,10,29,167,143,187,30,46,55,63,72,87,98,168]

http_referers =[

    "http://www.baidu.com/s?wd={query}",
    "https://www.sogou.com/web?query={query}",
    "http://cn.bing.com/search?q={query}",
    "http://www.baidu.com/s?wd={query}",
    "https://search.yahoo.com/search?p={query}",
]

search_keyword = [
    "Spark实战",
    "Hadoop基础",
]

```

```

    "Storm实战",
    "Spark streaming实战",
    "大数据面试"
]
status_codes = ["200", "404", "500"]

def sample_url():
    return random.sample(url_paths, 1)[0]

def sample_ip():
    slice = random.sample(ip_slices, 4)
    return ".".join([str(item) for item in slice])

def sample_referer():
    if random.uniform(0, 1) > 0.2:
        return "-"

    refer_str = random.sample(http_referers, 1)
    query_str = random.sample(search_keyword, 1)
    return refer_str[0].format(query=query_str[0])

def sample_status_code():
    return random.sample(status_codes, 1)[0]

def generate_log(count = 10):
    time_str = time.strftime("%Y-%m-%d %H:%M:%S", time.localtime())
    while count >= 1:
        query_log = "{local_time}\t{url}\t{ip}\t{referer}\t{status_code}".format(url=sample_url(), ip=sample_ip(), referer=sample_referer(), status_code=sample_status_code(), local_time=time_str)
        print(query_log)
        count = count -1

if __name__=='__main__':
    generate_log()

```

4.服务器测试并将日志写入到文件中

```

#coding=UTF-8
import random
import time

url_paths = [
    "class/112.html",
    "class/118.html",
    "class/145.html",
    "class/146.html",
    "class/131.html",
    "class/110.html",
    "learn/821",
    "course/list"
]

ip_slices = [132, 156, 124, 10, 29, 167, 143, 187, 30, 46, 55, 63, 72, 87, 98, 168]

http_referers =[

]

```

```

"http://www.baidu.com/s?wd={query}",
"https://www.sogou.com/web?query={query}",
"http://cn.bing.com/search?q={query}",
"http://www.baidu.com/s?wd={query}",
"https://search.yahoo.com/search?p={query}",
]

search_keyword = [
    "Spark实战",
    "Hadoop基础",
    "Storm实战",
    "Spark streaming实战",
    "大数据面试"
]
status_codes = ["200", "404", "500"]

def sample_url():
    return random.sample(url_paths, 1)[0]

def sample_ip():
    slice = random.sample(ip_slices, 4)
    return ".".join([str(item) for item in slice])

def sample_referer():
    if random.uniform(0, 1) > 0.2:
        return "-"

    refer_str = random.sample(http_referers, 1)
    query_str = random.sample(search_keyword, 1)
    return refer_str[0].format(query=query_str[0])

def sample_status_code():
    return random.sample(status_codes, 1)[0]

def generate_log(count = 10):
    time_str = time.strftime("%Y-%m-%d %H:%M:%S", time.localtime())
    f = open("/home/jungle/data/project/logs/access.log", "w+")
    while count >= 1:
        query_log = "{local_time}\t{url}\t{ip}\t{referer}\t{status_code}".format(url=sample_url(), ip=sample_ip(), referer=sample_referer(), status_code=sample_status_code(), local_time=time_str)
        print(query_log)

        f.write(query_log + "\n")
        count = count - 1

if __name__=='__main__':
    generate_log()

```

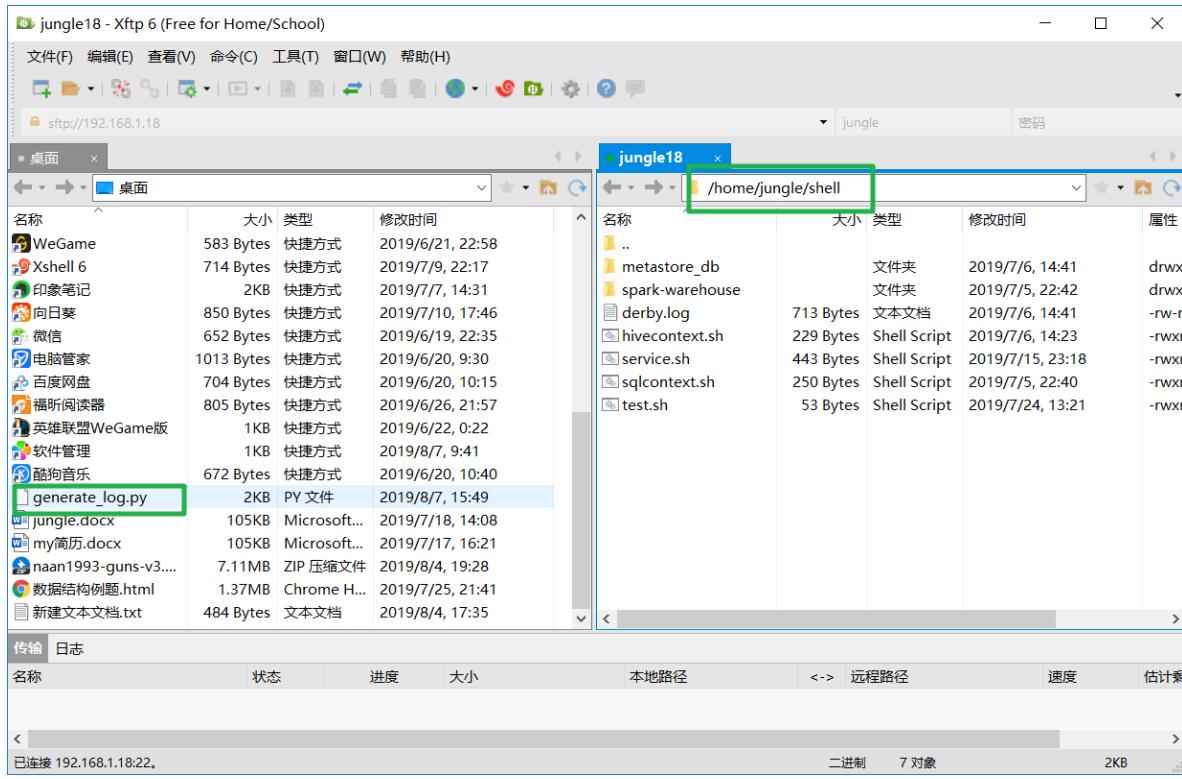
--新建文件夹文件--

```

cd data
sudo mkdir -p ./project/logs
cd project/logs
touch access.log

```

==上传py脚本==



```
python generate_log.py
```

```
jungle@centosserver1:[/home/jungle/shell]python generate_log.py
2019-08-07 15:56:45 class/131.html 132.187.124.168 - 200
2019-08-07 15:56:45 class/112.html 132.187.143.98 - 500
2019-08-07 15:56:45 learn/821 30.156.167.98 - 404
2019-08-07 15:56:45 class/146.html 124.46.55.143 - 404
2019-08-07 15:56:45 class/118.html 156.124.98.187 https://www.sogou.com/web?query=大数据面试 404
2019-08-07 15:56:45 learn/821 30.132.87.168 - 200
2019-08-07 15:56:45 class/145.html 167.143.168.10 - 500
2019-08-07 15:56:45 class/118.html 187.132.10.29 - 200
2019-08-07 15:56:45 class/131.html 132.156.30.46 - 500
2019-08-07 15:56:45 class/131.html 29.55.72.87 - 404
jungle@centosserver1:[/home/jungle/shell]
```

```
# 查看数据条数
wc -l access.log
```

==对日志进行微调==

```
#coding=UTF-8
import random
import time

url_paths = [
    "class/112.html",
    "class/118.html",
    "class/145.html",
    "class/146.html",
    "class/131.html",
    "class/110.html",
    "learn/821",
    "course/list"
]

ip_slices = [132,156,124,10,29,167,143,187,30,46,55,63,72,87,98,168]
```

```

http_referers = [
    "http://www.baidu.com/s?wd={query}",
    "https://www.sogou.com/web?query={query}",
    "http://cn.bing.com/search?q={query}",
    "http://www.baidu.com/s?wd={query}",
    "https://search.yahoo.com/search?p={query}",
]

search_keyword = [
    "Spark实战",
    "Hadoop基础",
    "Storm实战",
    "Spark streaming实战",
    "大数据面试"
]
status_codes = ["200", "404", "500"]

def sample_url():
    return random.sample(url_paths, 1)[0]

def sample_ip():
    slice = random.sample(ip_slices, 4)
    return ".".join([str(item) for item in slice])

def sample_referer():
    if random.uniform(0, 1) > 0.2:
        return "-"

    refer_str = random.sample(http_referers, 1)
    query_str = random.sample(search_keyword, 1)
    return refer_str[0].format(query=query_str[0])

def sample_status_code():
    return random.sample(status_codes, 1)[0]

def generate_log(count = 10):
    time_str = time.strftime("%Y-%m-%d %H:%M:%S", time.localtime())
    f = open("/home/jungle/data/project/logs/access.log", "w+")
    while count >= 1:
        query_log = "{ip}\t{local_time}\t\"GET /{url}"
        HTTP/1.1\" \t{status_code}\t{referer}".format(url=sample_url(), ip=sample_ip(), referer=sample_referer(), status_code=sample_status_code(), local_time=time_str)
        print(query_log)

        f.write(query_log + "\n")
        count = count - 1

    if __name__=='__main__':
        generate_log()

```

5. 定时调度工具

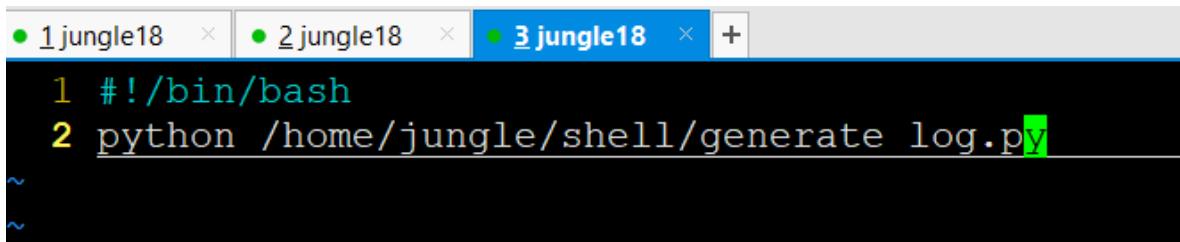
==每一分钟产生一批数据==

[crontab](#)

- 新建脚本

```
touch log_generator.sh  
chmod u+x log_generator.sh
```

```
#!/bin/bash  
python /home/jungle/shell/generate_log.py
```



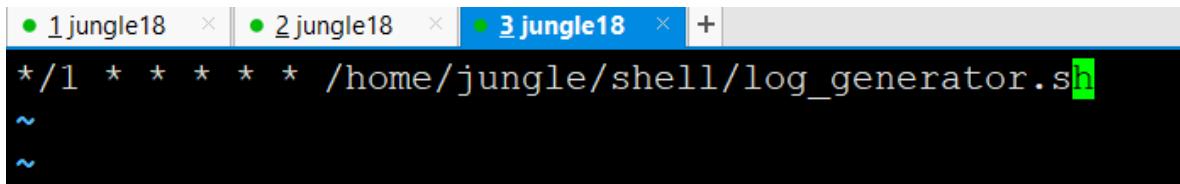
```
● 1 jungle18 × ● 2 jungle18 × ● 3 jungle18 × +  
1 #!/bin/bash  
2 python /home/jungle/shell/generate_log.py  
~  
~
```

- 定时任务

```
crontab -e
```

```
You have new mail in /var/spool/mail/jungle  
jungle@centosserver1:[/home/jungle/shell]crontab -e
```

```
*/1 * * * * sh /home/jungle/shell/log_generator.sh
```



```
● 1 jungle18 × ● 2 jungle18 × ● 3 jungle18 × +  
*/1 * * * * /home/jungle/shell/log_generator.sh  
~  
~
```

二、使用Flume实时收集日志信息

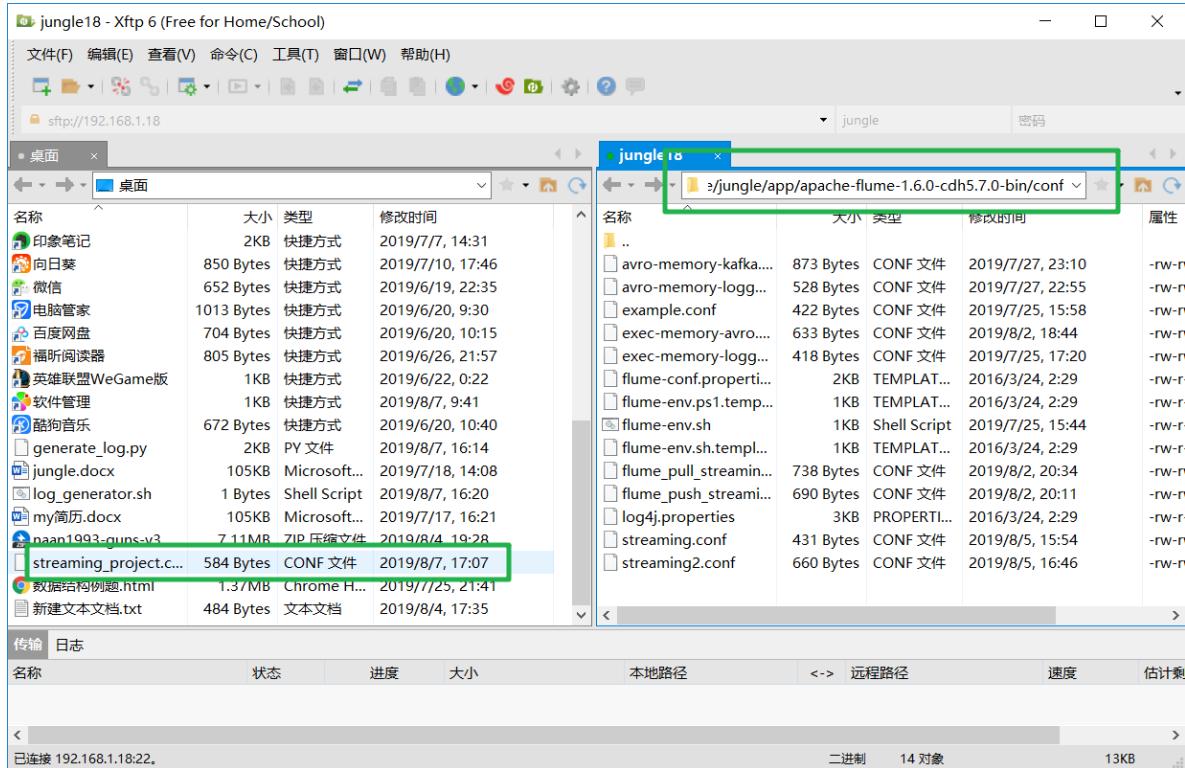
```
选型: access.log ==> 控制台输出  
exec  
memory  
logger
```

- streaming_project.conf

```
exec-memory-logger.sources = exec-source  
exec-memory-logger.sinks = logger-sink  
exec-memory-logger.channels = memory-channel1  
  
exec-memory-logger.sources.exec-source.type = exec  
exec-memory-logger.sources.exec-source.command = tail -F  
/home/jungle/data/project/logs/access.log  
exec-memory-logger.sources.exec-source.shell = /bin/sh -c  
  
exec-memory-logger.channels.memory-channel1.type = memory  
  
exec-memory-logger.sinks.logger-sink.type = logger  
  
exec-memory-logger.sources.exec-source.channels = memory-channel1  
exec-memory-logger.sinks.logger-sink.channel = memory-channel1
```

==上传服务器==

```
cd $FLUME_HOME/conf
```



- 启动flume

```
flume-ng agent \
--conf $FLUME_HOME/conf \
--conf-file $FLUME_HOME/conf/streaming_project.conf \
--name exec-memory-logger \
-Dflume.root.logger=INFO,console
```

```
2019-08-07 17:12:50,552 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.apache.flume.sink.LoggerSink.process(LoggerSink.java:94)] Event nt: { headers:{} body: 34 36 2E 33 30 2E 37 32 2E 31 35 36 09 32 30 31 46,30,72,156,201 }
2019-08-07 17:12:50,552 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.apache.flume.sink.LoggerSink.process(LoggerSink.java:94)] Event nt: { headers:{} body: 31 33 32 2E 31 38 37 2E 36 33 2E 34 36 09 32 30 132,187,63,46,20 }
2019-08-07 17:12:50,552 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.apache.flume.sink.LoggerSink.process(LoggerSink.java:94)] Event nt: { headers:{} body: 39 38 28 31 36 38 28 31 34 33 2E 35 35 09 32 30 30,168,143,55,20 }
2019-08-07 17:12:50,553 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.apache.flume.sink.LoggerSink.process(LoggerSink.java:94)] Event nt: { headers:{} body: 31 30 2E 31 33 32 2B 38 37 2B 31 36 37 09 32 30 10,132,87,167,20 }
2019-08-07 17:13:01,060 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.apache.flume.sink.LoggerSink.process(LoggerSink.java:94)] Event nt: { headers:{} body: 31 35 36 2B 35 35 2B 39 38 2B 31 32 34 09 32 30 156,55,98,124,20 }
2019-08-07 17:13:05,062 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.apache.flume.sink.LoggerSink.process(LoggerSink.java:94)] Event nt: { headers:{} body: 31 36 37 2B 31 36 38 2B 35 35 2E 31 32 34 09 32 167,168,55,124,2 ]
2019-08-07 17:13:05,064 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.apache.flume.sink.LoggerSink.process(LoggerSink.java:94)] Event nt: { headers:{} body: 31 34 33 2E 37 32 2B 38 37 2B 31 36 38 09 32 30 143,72,87,168,20 ]
2019-08-07 17:13:05,064 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.apache.flume.sink.LoggerSink.process(LoggerSink.java:94)] Event nt: { headers:{} body: 36 33 2B 31 35 36 2B 35 35 2B 39 38 09 32 30 31 63,156,55,98,201 }
2019-08-07 17:13:05,064 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.apache.flume.sink.LoggerSink.process(LoggerSink.java:94)] Event nt: { headers:{} body: 33 30 2B 31 33 32 2B 32 39 2B 35 35 09 32 30 31 30,132,29,55,201 }
2019-08-07 17:13:05,065 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.apache.flume.sink.LoggerSink.process(LoggerSink.java:94)] Event nt: { headers:{} body: 38 37 2B 37 32 2B 31 32 34 2E 31 30 09 32 30 31 87,72,124,10,20 }
2019-08-07 17:13:05,065 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.apache.flume.sink.LoggerSink.process(LoggerSink.java:94)] Event nt: { headers:{} body: 38 37 2B 32 39 2B 33 30 2B 31 32 34 09 32 30 31 87,29,30,124,201 }
2019-08-07 17:13:05,065 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.apache.flume.sink.LoggerSink.process(LoggerSink.java:94)] Event nt: { headers:{} body: 33 30 2E 35 35 2E 34 36 2E 31 36 37 09 32 30 31 30,55,46,167,201 }
2019-08-07 17:13:05,065 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.apache.flume.sink.LoggerSink.process(LoggerSink.java:94)] Event nt: { headers:{} body: 33 30 2E 31 33 32 2B 31 34 33 2E 31 30 09 32 30 30,132,143,10,20 }
2019-08-07 17:13:05,065 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.apache.flume.sink.LoggerSink.process(LoggerSink.java:94)] Event nt: { headers:{} body: 31 38 37 2B 31 36 38 2B 31 34 33 2E 34 36 09 32 187,168,143,46,2 }
2019-08-07 17:14:01,217 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.apache.flume.sink.LoggerSink.process(LoggerSink.java:94)] Event nt: { headers:{} body: 31 36 37 2B 39 38 2B 31 36 38 09 32 30 167,98,168,72,20 }
```

三、对接实时日志数据到Kafka并输出到控制台测试

- 启动zookeeper
- 启动kafka
- 修改flume配置文件 (对接kafka)
 - streaming_project2.conf

```

exec-memory-kafka.sources = exec-source
exec-memory-kafka.sinks = kafka-sink
exec-memory-kafka.channels = memory-channel

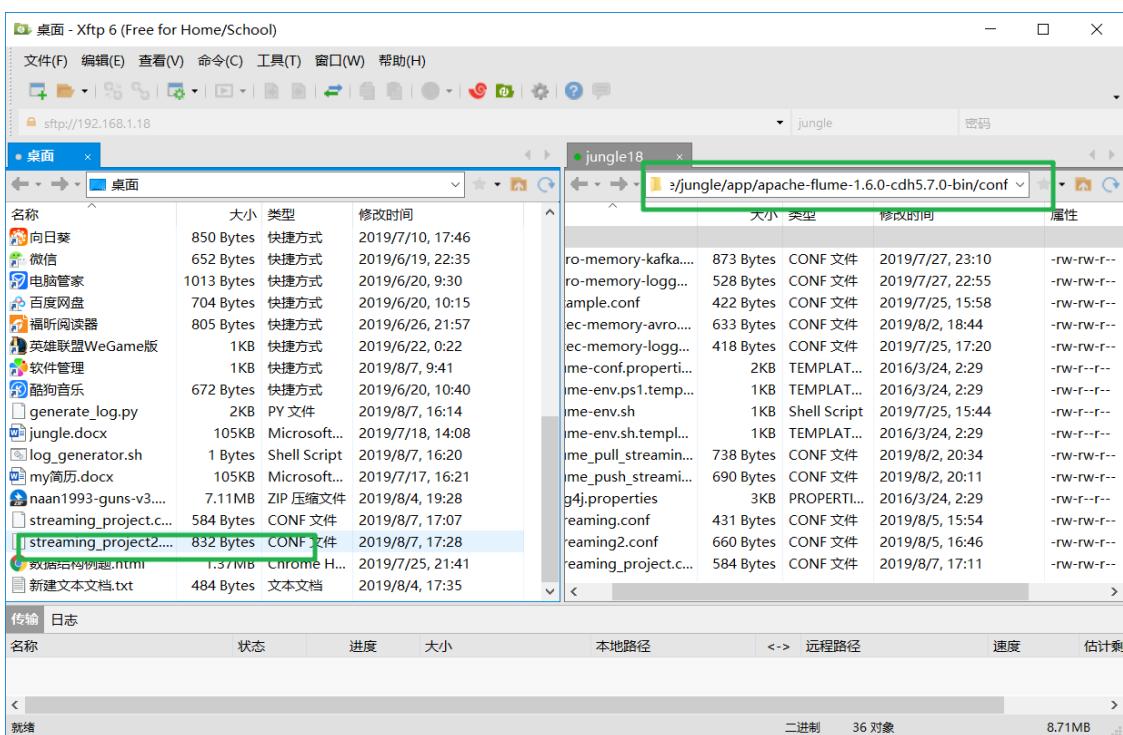
exec-memory-kafka.sources.exec-source.type = exec
exec-memory-kafka.sources.exec-source.command = tail -F
/home/jungle/data/project/logs/access.log
exec-memory-kafka.sources.exec-source.shell = /bin/sh -c

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

exec-memory-kafka.sinks.kafka-sink.type =
org.apache.flume.sink.kafka.KafkaSink
exec-memory-kafka.sinks.kafka-sink.brokerList = centosserver1:9092
exec-memory-kafka.sinks.kafka-sink.topic = streamingtopic
exec-memory-kafka.sinks.kafka-sink.batchSize = 5
exec-memory-kafka.sinks.kafka-sink.requiredAcks = 1

exec-memory-kafka.sources.exec-source.channels = memory-channel
exec-memory-kafka.sinks.kafka-sink.channel = memory-channel

```



4. Kafka消费者

```
kafka-console-consumer.sh --zookeeper centosserver1:2181 --topic
streamingtopic
```

5. 启动flume

```
flume-ng agent \
--conf $FLUME_HOME/conf \
--conf-file $FLUME_HOME/conf/streaming_project2.conf \
--name exec-memory-kafka \
-dflume.root.logger=INFO,console
```

```
You have new mail in /var/spool/mail/jungle
jungle@centosserver1:[/home/jungle/data/project/logs]kafka-console-consumer.sh --zookeeper centosserver1:2181 --topic streamingtopic
168.30.143.63 2019-08-07 17:30:01 "GET /class/131.html HTTP/1.1" 500 https://search.yahoo.com/search?p=Spark实战
10.63.55.167 2019-08-07 17:30:01 "GET /class/145.html HTTP/1.1" 404 -
143.132.10.167 2019-08-07 17:30:01 "GET /learn/821 HTTP/1.1" 200 -
80.168.132.167 2019-08-07 17:30:01 "GET /class/131.html HTTP/1.1" 404 -
80.46.156.63 2019-08-07 17:30:01 "GET /learn/821 HTTP/1.1" 404 -
168.143.98.46 2019-08-07 17:30:01 "GET /class/112.html HTTP/1.1" 404 -
187.167.10.98 2019-08-07 17:30:01 "GET /class/146.html HTTP/1.1" 200 http://www.baidu.com/s?wd=Spark streaming实战
53.55.156.167 2019-08-07 17:30:01 "GET /class/146.html HTTP/1.1" 500 -
143.168.55.167 2019-08-07 17:30:01 "GET /course/list HTTP/1.1" 404 -
124.87.167.30 2019-08-07 17:30:01 "GET /class/110.html HTTP/1.1" 200 http://www.baidu.com/s?wd=Spark streaming实战
```

四、Spark Streaming对接Kafka的数据进行消费

◆ 在Spark应用程序接收到数据并完成记录数统计

- 程序

```
package com.jungle.spark.project

import org.apache.spark.SparkConf
import org.apache.spark.streaming.StreamingContext
import org.apache.spark.streaming.Seconds
import org.apache.spark.streaming.kafka.KafkaUtils

/**
 * 使用Spark Streaming处理Kafka过来的数据
 */
object ImoocStatStreamingApp {

    def main(args: Array[String]): Unit = {

        if (args.length != 4) {
            println("Usage: ImoocStatStreamingApp <zkQuorum> <group> <topics> <numThreads>")
            System.exit(1)
        }

        val Array(zkQuorum, groupId, topics, numThreads) = args

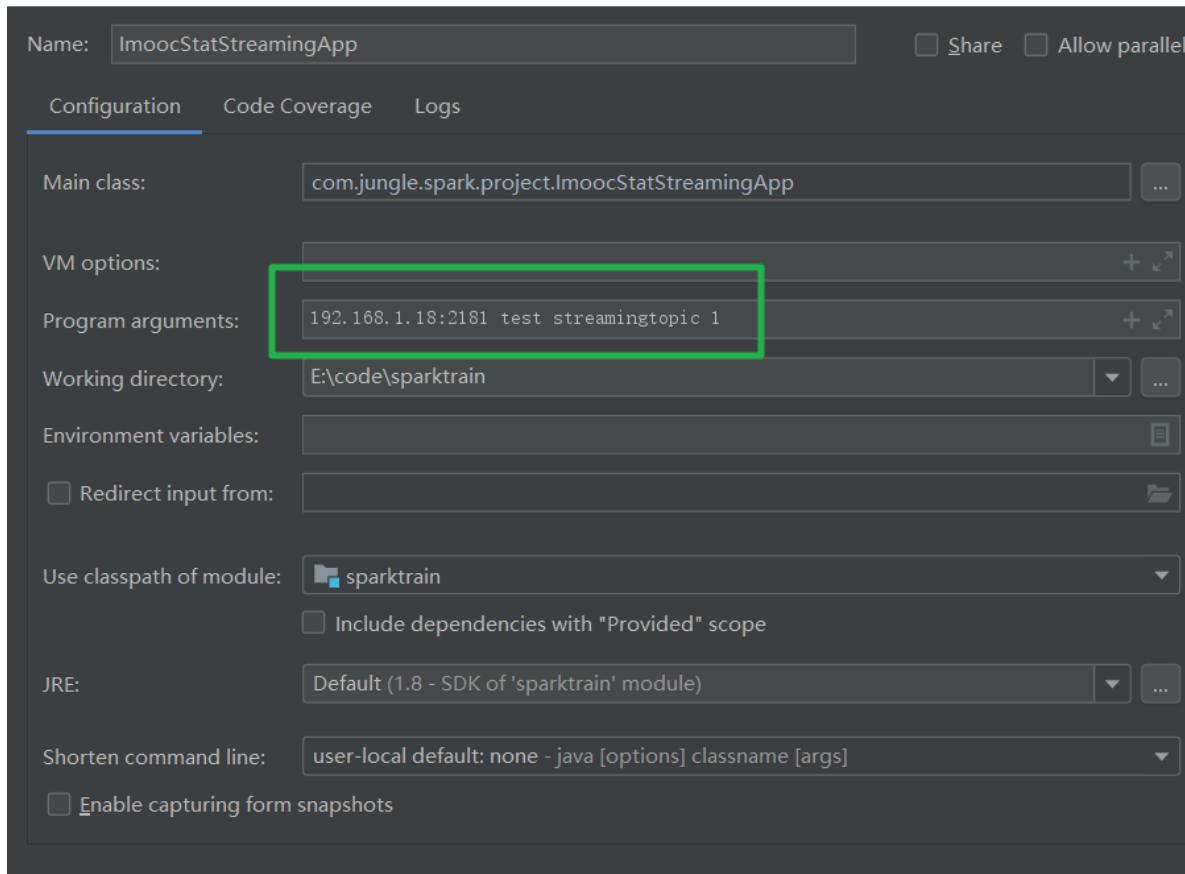
        val sparkConf = new
        SparkConf().setAppName("ImoocStatStreamingApp").setMaster("local[2]")
        val ssc = new StreamingContext(sparkConf, Seconds(60))

        val topicMap = topics.split(",").map((_, numThreads.toInt)).toMap

        val messages = KafkaUtils.createStream(ssc, zkQuorum, groupId, topicMap)

        // 测试步骤一：测试数据接收
        messages.map(_._2).count().print

        ssc.start()
        ssc.awaitTermination()
    }
}
```



192.168.1.18:2181 test streamingtopic 1

==效果==

```
ImoocStatStreamingApp x
19/08/08 14:56:00 INFO scheduler.TaskSchedulerImpl: Adding task set 20.0 wi
19/08/08 14:56:00 INFO scheduler.TaskSetManager: Starting task 0.0 in stage
19/08/08 14:56:00 INFO executor.Executor: Running task 0.0 in stage 20.0 (T
19/08/08 14:56:00 INFO storage.ShuffleBlockFetcherIterator: Getting 0 non-e
19/08/08 14:56:00 INFO storage.ShuffleBlockFetcherIterator: Started 0 remot
19/08/08 14:56:00 INFO executor.Executor: Finished task 0.0 in stage 20.0 (0
19/08/08 14:56:00 INFO scheduler.TaskSetManager: Finished task 0.0 in stage
19/08/08 14:56:00 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 20.0, w
19/08/08 14:56:00 INFO scheduler.DAGScheduler: ResultStage 20 (print at Imo
19/08/08 14:56:00 INFO scheduler.DAGScheduler: Job 10 finished: print at Imo

-----
Time: 1565247360000 ms
-----
510

19/08/08 14:56:00 INFO scheduler.JobScheduler: Finished job streaming job 1
19/08/08 14:56:00 INFO scheduler.JobScheduler: Total delay: 0.157 s for tir
19/08/08 14:56:00 INFO rdd.ManPartitionsRDD: Removing RDD 28 from persiste
```

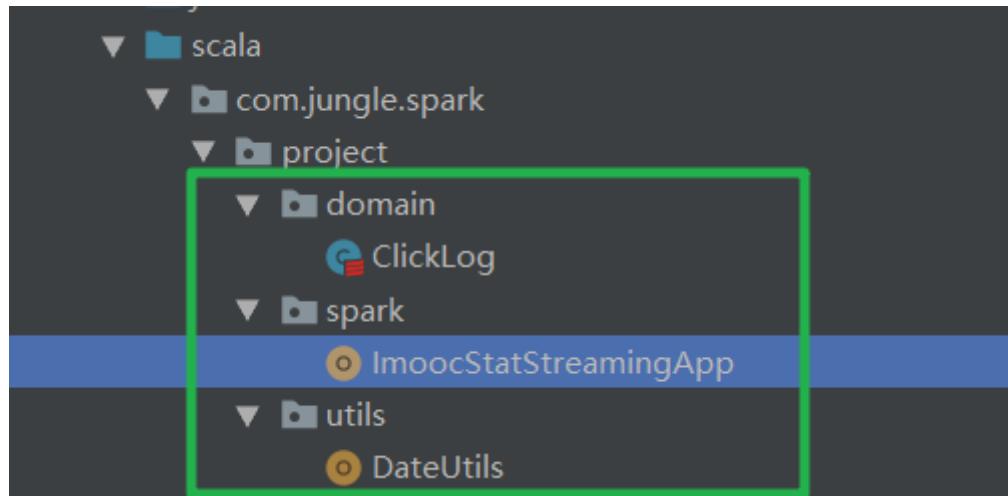
五、使用Spark Streaming完成数据清洗操作

数据清洗操作：从原始日志中取出我们所需要的字段信息就可以了

数据清洗结果类似如下：

```
ClickLog(46.30.10.167,20171022151701,128,200,-)
ClickLog(143.132.168.72,20171022151701,131,404,-)
ClickLog(10.55.168.87,20171022151701,131,500,-)
ClickLog(10.124.168.29,20171022151701,128,404,-)
ClickLog(98.30.87.143,20171022151701,131,404,-)
ClickLog(55.10.29.132,20171022151701,146,404,http://www.baidu.c
ClickLog(10.87.55.30,20171022151701,130,200,http://www.baidu.co
ClickLog(156.98.29.30,20171022151701,146,500,https://www.sogou.
ClickLog(10.72.87.124,20171022151801,146,500,-)
ClickLog(72.124.167.156,20171022151801,112,404,-)
```

- 目录结构



- ImoocStatStreamingApp

```
package com.jungle.spark.project.spark

import com.jungle.spark.project.domain.ClickLog
import com.jungle.spark.project.utils.DateUtils
import org.apache.spark.SparkConf
import org.apache.spark.streaming.kafka.KafkaUtils
import org.apache.spark.streaming.{Seconds, StreamingContext}

/**
 * 使用Spark Streaming处理Kafka过来的数据
 */
object ImoocStatStreamingApp {

    def main(args: Array[String]): Unit = {

        if (args.length != 4) {
            println("Usage: ImoocStatStreamingApp <zkQuorum> <group> <topics>
<numThreads>")
            System.exit(1)
        }

        val Array(zkQuorum, groupId, topics, numThreads) = args

        val sparkConf = new
SparkConf().setAppName("ImoocStatStreamingApp").setMaster("local[2]")
```

```

    val ssc = new StreamingContext(sparkConf, Seconds(60))

    val topicMap = topics.split(",").map(_ , numThreads.toInt)).toMap

    val messages = KafkaUtils.createStream(ssc, zkQuorum, groupId, topicMap)

    // messages.print

    // 测试步骤一：测试数据接收
    // messages.map(_.value).count().print
    // messages.map(_.value).print

    // 测试步骤二：数据清洗
    val logs = messages.map(_.value)
    val cleanData = logs.map(line => {
        val infos = line.split("\t")

        // infos(2) = "GET /class/130.html HTTP/1.1"
        // url = /class/130.html
        val url = infos(2).split(" ")(1)
        var courseId = 0

        // 把实战课程的课程编号拿到了
        if (url.startsWith("/class")) {
            val courseIdHTML = url.split("/") (2)
            courseId = courseIdHTML.substring(0,
courseIdHTML.lastIndexOf(".")).toInt
        }

        ClickLog(infos(0), DateUtils.parseToMinute(infos(1)), courseId,
infos(3).toInt, infos(4))
    }).filter(clicklog => clicklog.courseId != 0)

    cleanData.print()

    ssc.start()
    ssc.awaitTermination()

}
}

```

```

// 测试步骤二：数据清洗
val logs = messages.map(_._2)
val cleanData = logs.map(line => {
    val infos = line.split( regex = "\t" )

    // infos(2) = "GET /class/130.html HTTP/1.1"
    // url = /class/130.html
    val url = infos(2).split( regex = " " )(1)
    var courseId = 0

    // 把实战课程的课程编号拿到了
    if (url.startsWith("/class")) {
        val courseIdHTML = url.split( regex = "/" )(2)
        courseId = courseIdHTML.substring(0, courseIdHTML.lastIndexOf( str = "." )).toInt
    }

    ClickLog(infos(0), DateUtils.parseToMinute(infos(1)), courseId, infos(3).toInt, infos(4))
}.filter(clicklog => clicklog.courseId != 0)

cleanData.print()

ssc.start()
ssc.awaitTermination()

```

- DateUtils

```

package com.jungle.spark.project.utils

import java.util.Date

import org.apache.commons.lang3.time.FastDateFormat

/**
 * 日期时间工具类
 */
object Dateutils {

    val YYYYMMDDHHMMSS_FORMAT = FastDateFormat.getInstance("yyyy-MM-dd
HH:mm:ss")
    val TARGE_FORMAT = FastDateFormat.getInstance("yyyyMMddHHmmss")

    def getTime(time: String) = {
        YYYYMMDDHHMMSS_FORMAT.parse(time).getTime
    }

    def parseToMinute(time :String) = {
        TARGE_FORMAT.format(new Date(getTime(time)))
    }
    def main(args: Array[String]): Unit = {
        println(parseToMinute("2017-10-22 14:46:01"))
    }
}

```

- ClickLog

```

package com.jungle.spark.project.domain

/**
 * 清洗后的日志信息
 * @param ip 日志访问的ip地址
 * @param time 日志访问的时间
 * @param courseId 日志访问的实战课程编号
 * @param statusCode 日志访问的状态码
 * @param referer 日志访问的referer
 */
case class ClickLog(ip:String, time:String, courseId:Int, statusCode:Int,
referer:String)

```

==效果==

```

19/08/08 15:34:00 INFO scheduler.TaskSchedulerImpl: Finished task 0.0 in stage 1.0 (19/08/08 15:34:00 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 1.0, whose ta
19/08/08 15:34:00 INFO scheduler.DAGScheduler: ResultStage 1 (print at ImoocStatS
19/08/08 15:34:00 INFO scheduler.DAGScheduler: Job 1 finished: print at ImoocStat

-----
Time: 1565249640000 ms

-----
ClickLog(98.156.87.55,20190808153301,131,404,-)
ClickLog(98.168.156.143,20190808153301,110,200,-)
ClickLog(29.10.46.98,20190808153301,118,200,-)
ClickLog(87.156.63.30,20190808153301,110,500,-)
ClickLog(156.168.72.87,20190808153301,110,500,-)
ClickLog(132.30.29.168,20190808153301,110,200,-)
ClickLog(30.46.72.87,20190808153301,110,200,-)
ClickLog(156.10.167.143,20190808153301,112,500,-)
ClickLog(143.167.87.187,20190808153301,110,200,-)
ClickLog(72.30.98.10,20190808153301,131,404,http://www.baidu.com/s?wd=Spark实战)
...

```

到数据清洗完为止，日志中只包含了实战课程的日志

六、功能一

1.需求分析及存储结果技术选型分析

功能1：今天到现在为止实战课程的访问量

yyyyymmdd courseid

使用数据库来进行存储我们的统计结果

Spark Streaming把统计结果写入到数据库里面

可视化前端根据：yyyyymmdd courseid把数据库里面的统计结果展示出来

选择什么数据库作为统计结果的存储呢？

1.RDBMS: MySQL、 Oracle

day	courseid	click count
20171111	1	10
20171111	1	10

下一个批次数据进来以后

20171111+1=> click_count+下一个批次的统计结果 ==>写入到数据库中

2.NOSQL: Hbase、 Redis

Hbase:一个API就能搞定，非常方便

20171111+1=> click_count+下一个批次的统计结果

本次课程为什么要选择 Hbase的一个原因所在

前提：

HDFS

- 启动hbase

```
cd $HBASE_HOME/bin  
. ./start-hbase.sh
```

==进入hbase命令行==

```
cd $HBASE_HOME/bin  
. ./hbase shell
```

```
jungle@centosserver1:[/home/jungle] cd $HBASE_HOME/bin  
jungle@centosserver1:[/home/jungle/app/hbase-1.2.0-cdh5.7.0/bin]. ./hbase shell  
2019-08-08 16:00:26,841 INFO [main] Configuration.deprecation: hadoop.native.lib is  
2019-08-08 16:00:28,950 WARN [main] util.NativeCodeLoader: Unable to load native-had  
where applicable  
SLF4J: Class path contains multiple SLF4J bindings.  
SLF4J: Found binding in [jar:file:/home/jungle/app/hbase-1.2.0-cdh5.7.0/lib/slf4j-log  
SLF4J: Found binding in [jar:file:/home/jungle/app/hadoop-2.6.0-cdh5.7.0/share/hadoop  
ggerBinder.class]  
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.  
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]  
HBase Shell; enter 'help<RETURN>' for list of supported commands.  
Type "exit<RETURN>" to leave the HBase Shell  
Version 1.2.0-cdh5.7.0, rUnknown, Wed Mar 23 11:46:29 PDT 2016  
  
hbase(main):001:0> █
```

```
# 查看表的数量  
list
```

```
hbase(main):001:0> list  
TABLE  
member  
1 row(s) in 0.4720 seconds  
  
=> ["member"]  
hbase(main):002:0> █
```

==Hbase表设计==

```
create 'imooc_course_clickcount','info'
```

```
hbase(main):002:0> create 'imooc_course_clickcount', 'info'
0 row(s) in 1.3530 seconds

=> Hbase::Table - imooc_course_clickcount
hbase(main):003:0> list
TABLE
imooc_course_clickcount
member
2 row(s) in 0.0030 seconds

=> ["imooc_course_clickcount", "member"]
hbase(main):004:0> █
```

```
desc 'imooc_course_clickcount'
```

```
hbase(main):004:0> desc 'imooc_course_clickcount'
Table imooc_course_clickcount is ENABLED
imooc_course_clickcount
COLUMN FAMILIES DESCRIPTION
{NAME => 'info', BLOOMFILTER => 'ROW', VERSIONS => '1', IN_MEMORY =>
> 'FOREVER', COMPRESSION => 'NONE', MIN VERSIONS => '0', BLOCKCACHE =
1 row(s) in 0.3060 seconds
```

```
scan 'imooc_course_clickcount'
```

```
hbase(main):005:0> scan 'imooc_course_clickcount'
ROW                               COLUMN+CELL
0 row(s) in 0.1550 seconds
```

```
hbase(main):006:0> █
```

HBase表设计

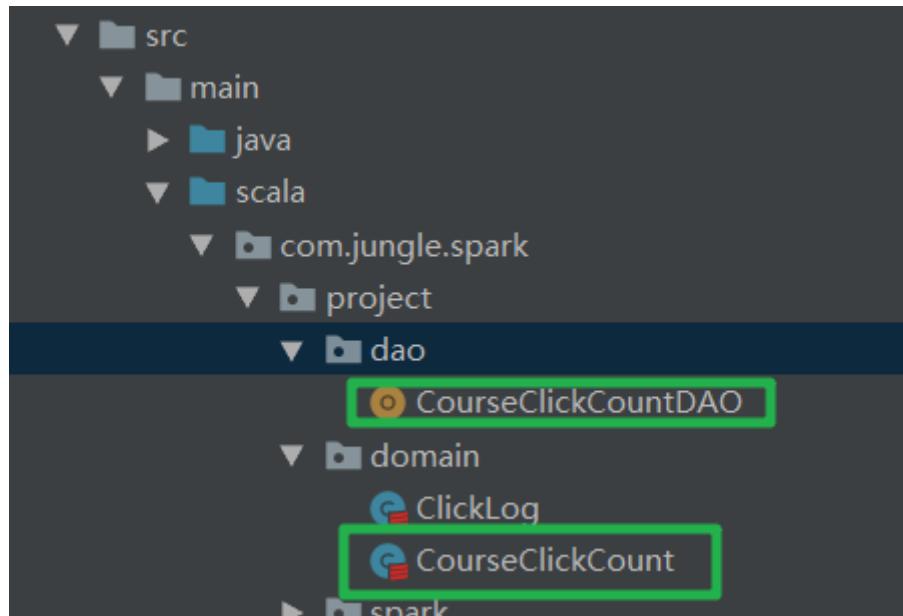
创建表

```
create 'imooc_course_clickcount', 'info'
```

Rowkey设计

```
day_courseid
```

2.数据库访问DAO层方法定义



- CourseClickCount

```
package com.jungle.spark.project.domain

/**
 * 实战课程点击数实体类
 * @param day_course 对应的就是HBase中的rowkey, 20171111_1
 * @param click_count 对应的20171111_1的访问总数
 */
case class CourseClickCount(day_course:String, click_count:Long)
```

- CourseClickCountDAO

```
package com.jungle.spark.project.dao

import com.jungle.spark.project.domain.CourseClickCount
import scala.collection.mutable.ListBuffer

/**
 * 实战课程点击数-数据访问层
 */
object CourseClickCountDAO {

    val tableName = "imooc_course_clickcount"
    val cf = "info"
    val qualifier = "click_count"

    /**
     * 保存数据到HBase
     * @param list CourseClickCount集合
     */
    def save(list: ListBuffer[CourseClickCount]): Unit = {

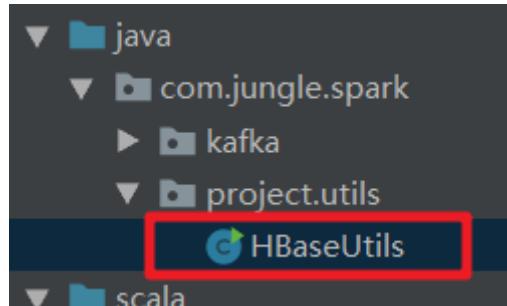
    }

    /**
     * 根据rowkey查询值
     */
}
```

```
def count(day_course: String):Long = {
  0L
}
}
```

3.HBase操作工具类开发

这是个Java类



- HBaseUtils

```
package com.jungle.spark.project.utils;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.client.HBaseAdmin;
import org.apache.hadoop.hbase.client.HTable;
import org.apache.hadoop.hbase.client.Put;
import org.apache.hadoop.hbase.util.Bytes;

import java.io.IOException;

/**
 * HBase操作工具类: Java工具类建议采用单例模式封装
 */
public class HBaseUtils {

    HBaseAdmin admin = null;
    Configuration configuration = null;

    /**
     * 私有改造方法
     */
    private HBaseUtils(){
        configuration = new Configuration();
        configuration.set("hbase.zookeeper.quorum", "centosserver1:2181");
        configuration.set("hbase.rootdir",
"dfs://centosserver1:8020/hbase");

        try {
            admin = new HBaseAdmin(configuration);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```
private static HBaseUtils instance = null;

public static synchronized HBaseUtils getInstance() {
    if(null == instance) {
        instance = new HBaseUtils();
    }
    return instance;
}

/**
 * 根据表名获取到HTable实例
 */
public HTable getTable(String tableName) {

    HTable table = null;

    try {
        table = new HTable(configuration, tableName);
    } catch (IOException e) {
        e.printStackTrace();
    }

    return table;
}

/**
 * 添加一条记录到HBase表
 * @param tableName HBase表名
 * @param rowkey HBase表的rowkey
 * @param cf HBase表的columnfamily
 * @param column HBase表的列
 * @param value 写入HBase表的值
 */
public void put(String tableName, String rowkey, String cf, String
column, String value) {
    HTable table = getTable(tableName);

    Put put = new Put(Bytes.toBytes(rowkey));
    put.add(Bytes.toBytes(cf), Bytes.toBytes(column),
Bytes.toBytes(value));

    try {
        table.put(put);
    } catch (IOException e) {
        e.printStackTrace();
    }
}

public static void main(String[] args) {

    //HTable table =
HBaseutils.getInstance().getTable("imooc_course_clickcount");
    //System.out.println(table.getName().getNameAsString());

    String tableName = "imooc_course_clickcount" ;
    String rowkey = "20171111_88";
    String cf = "info" ;
```

```
        String column = "click_count";
        String value = "2";

        HBaseUtils.getInstance().put(tableName, rowkey, cf, column, value);
    }

}
```

```
/***
 * 私有改造方法
 */
private HBaseUtils(){
    configuration = new Configuration();
    configuration.set("hbase.zookeeper.quorum", "centosserver1:2181");
    configuration.set("hbase.rootdir", "hdfs://centosserver1:8020/hbase");

    try {
        admin = new HBaseAdmin(configuration);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

哪里来的

```
vi $HBASE_HOME/conf/hbase-site.xml
```

```
! [1565253432624] (E:/课程/大数据与云计算/sparkstreaming-
master/picture/1565253432624.png)
```

==执行HBaseUtils==

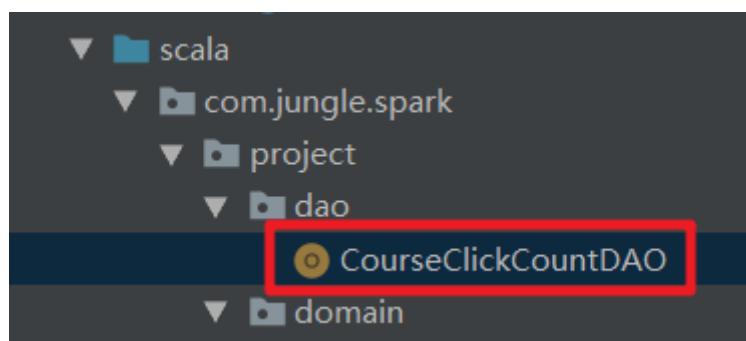
```
scan 'imooc_course_clickcount'
```

```
hbase(main):002:0> scan 'imooc_course_clickcount'
ROW                                     COLUMN+CELL
20171111_88                           column=info:click_count, timestamp=1565253780957, value=2
1 row(s) in 0.1970 seconds

hbase(main):003:0>
```

4.数据库访问DAO层方法实现

==补全CourseClickCountDAO代码==



```
package com.jungle.spark.project.dao

import com.jungle.spark.project.domain.CourseClickCount
import com.jungle.spark.utils.HBaseUtils
```

```
import org.apache.hadoop.hbase.client.Get
import org.apache.hadoop.hbase.util.Bytes

import scala.collection.mutable.ListBuffer

/***
 * 实战课程点击数-数据访问层
 */
object CourseClickCountDAO {

    val tableName = "imooc_course_clickcount"
    val cf = "info"
    val qualifer = "click_count"

    /***
     * 保存数据到HBase
     * @param list CourseClickCount集合
     */
    def save(list: ListBuffer[CourseClickCount]): Unit = {

        val table = HBaseUtils.getInstance().getTable(tableName)

        for(ele <- list) {
            table.incrementColumnValue(Bytes.toBytes(ele.day_course),
                Bytes.toBytes(cf),
                Bytes.toBytes(qualifer),
                ele.click_count)
        }
    }

    /***
     * 根据rowkey查询值
     */
    def count(day_course: String):Long = {
        val table = HBaseUtils.getInstance().getTable(tableName)

        val get = new Get(Bytes.toBytes(day_course))
        val value = table.get(get).getValue(cf.getBytes, qualifer.getBytes)

        if(value == null) {
            0L
        }else{
            Bytes.toLong(value)
        }
    }

    def main(args: Array[String]): Unit = {

        val list = new ListBuffer[CourseClickCount]
        list.append(CourseClickCount("20171111_8",8))
        list.append(CourseClickCount("20171111_9",9))
        list.append(CourseClickCount("20171111_1",100))

        save(list)
    }
}
```

```
    println(count("20171111_8") + " : " + count("20171111_9") + " : " +  
count("20171111_1"))  
}  
  
}
```

```
2 ▶ object CourseClickCountDAO {  
3     val tableName = "imooc_course_clickcount"  
4     val cf = "info"  
5     val qualifier = "click_count"  
6  
7     /**  
8      * 保存数据到HBase  
9      * @param list CourseClickCount集合  
10     */  
11     def save(list: ListBuffer[CourseClickCount]): Unit = {  
12         val table = HBaseUtils getInstance().getTable(tableName)  
13  
14         for(ele <- list) {  
15             table incrementColumnValue Bytes.toBytes(ele.day_course),  
16                     Bytes.toBytes(cf),  
17                     Bytes.toBytes(qualifier),  
18                     ele.click_count  
19         }  
20     }  
21 }
```

Java类可以直接调用

一个API就能搞定的关键

==运行程序==

运行3次

```
19/08/08 17:01:53 INFO zookeep  
24 : 27 : 300
```

运行4次

```
19/08/08 17:21:02 INFO
19/08/08 17:21:02 INFO
19/08/08 17:21:02 INFO
19/08/08 17:21:06 INFO
19/08/08 17:21:07 INFO
32 : 36 : 400
```

5.Spark Streaming的处理结果写入到HBase中

- 代码

```
package com.jungle.spark.project.spark

import com.jungle.spark.project.dao.CourseClickCountDAO
import com.jungle.spark.project.domain.{ClickLog, Courseclickcount}
import com.jungle.spark.project.utils.Dateutils
import org.apache.spark.SparkConf
import org.apache.spark.streaming.kafka.Kafkautils
import org.apache.spark.streaming.{Seconds, StreamingContext}

import scala.collection.mutable.ListBuffer

/**
 * 使用spark streaming处理kafka过来的数据
 */
object ImoocStatStreamingApp {

  def main(args: Array[String]): Unit = {

    if (args.length != 4) {
      println("Usage: ImoocStatStreamingApp <zkQuorum> <group> <topics>
<numThreads>")
      System.exit(1)
    }

    val Array(zkQuorum, groupId, topics, numThreads) = args

    val sparkConf = new
SparkConf().setAppName("ImoocStatStreamingApp").setMaster("local[2]")
    val ssc = new StreamingContext(sparkConf, Seconds(60))

    val topicMap = topics.split(",").map((_, numThreads.toInt)).toMap

    val messages = Kafkautils.createStream(ssc, zkQuorum, groupId, topicMap)

    //    messages.print

    // 测试步骤一：测试数据接收
    //    messages.map(_.value).count().print
    //    messages.map(_.value).print

    // 测试步骤二：数据清洗
    val logs = messages.map(_.value)
```

```

val cleanData = logs.map(line => {
    val infos = line.split("\t")

    // infos(2) = "GET /class/130.html HTTP/1.1"
    // url = /class/130.html
    val url = infos(2).split(" ")(1)
    var courseId = 0

    // 把实战课程的课程编号拿到了
    if (url.startsWith("/class")) {
        val courseIdHTML = url.split("/") (2)
        courseId = courseIdHTML.substring(0,
courseIdHTML.lastIndexOf(".")).toInt
    }

    ClickLog(infos(0), DateUtils.parseToMinute(infos(1)), courseId,
infos(3).toInt, infos(4))
}.filter(clicklog => clicklog.courseId != 0)

//      cleanData.print()

// 测试步骤三：统计今天到现在为止实战课程的访问量

cleanData.map(x => {

    // HBase rowkey设计： 20171111_88

    (x.time.substring(0, 8) + "_" + x.courseId, 1)
}).reduceByKey(_ + _).foreachRDD(rdd => {
    rdd.foreachPartition(partitionRecords => {
        val list = new ListBuffer[CourseClickCount]

        partitionRecords.foreach(pair => {
            list.append(CourseClickCount(pair._1, pair._2))
        })

        CourseClickCountDAO.save(list)
    })
})

ssc.start()
ssc.awaitTermination()

}
}

```

```

// 测试步骤三：统计今天到现在为止实战课程的访问量

cleanData.map(x => {
    // HBase rowkey设计： 20171111_88
    (x.time.substring(0, 8) + "_" + x.courseId, 1)
}).reduceByKey(_ + _).foreachRDD(rdd => {
    rdd.foreachPartition(partitionRecords => {
        val list = new ListBuffer[CourseClickCount]

        partitionRecords.foreach(pair => {
            list.append(CourseClickCount(pair._1, pair._2))
        })
    })

    CourseClickCountDAO.save(list)
})
}

ssc.start()
ssc.awaitTermination()

```

==运行==

```

hbase(main):008:0> scan 'imooc_course_clickcount'
ROW                                     COLUMN+CELL
20190808_110                           column=info:click_count, timestamp=1565337181294, value=\x00\x00\x00\x00\x00\x00\x00#
20190808_112                           column=info:click_count, timestamp=1565337181300, value=\x00\x00\x00\x00\x00\x00\x00\x1B
20190808_118                           column=info:click_count, timestamp=1565337181278, value=\x00\x00\x00\x00\x00\x00\x00\x00*
20190808_131                           column=info:click_count, timestamp=1565337181250, value=\x00\x00\x00\x00\x00\x00\x00\x00\x1F
20190808_145                           column=info:click_count, timestamp=1565337181274, value=\x00\x00\x00\x00\x00\x00\x00\x00\x13
20190808_146                           column=info:click_count, timestamp=1565337181233, value=\x00\x00\x00\x00\x00\x00\x00\x00#
20190809_110                           column=info:click_count, timestamp=1565337300036, value=\x00\x00\x00\x00\x00\x00\x00\x00\x16
20190809_112                           column=info:click_count, timestamp=1565337300042, value=\x00\x00\x00\x00\x00\x00\x00\x00\x16
20190809_118                           column=info:click_count, timestamp=1565337300030, value=\x00\x00\x00\x00\x00\x00\x00\x00\x1C
20190809_131                           column=info:click_count, timestamp=1565337300097, value=\x00\x00\x00\x00\x00\x00\x00\x00\x1A
20190809_145                           column=info:click_count, timestamp=1565337300025, value=\x00\x00\x00\x00\x00\x00\x00\x00\x18
20190809_146                           column=info:click_count, timestamp=1565337300090, value=\x00\x00\x00\x00\x00\x00\x00\x00\x14
12 row(s) in 0.0350 seconds

```

七、功能二

1.需求分析及HBase设计&HBase数据访问层开发

功能二：功能一+从搜索引擎引流过来的

```

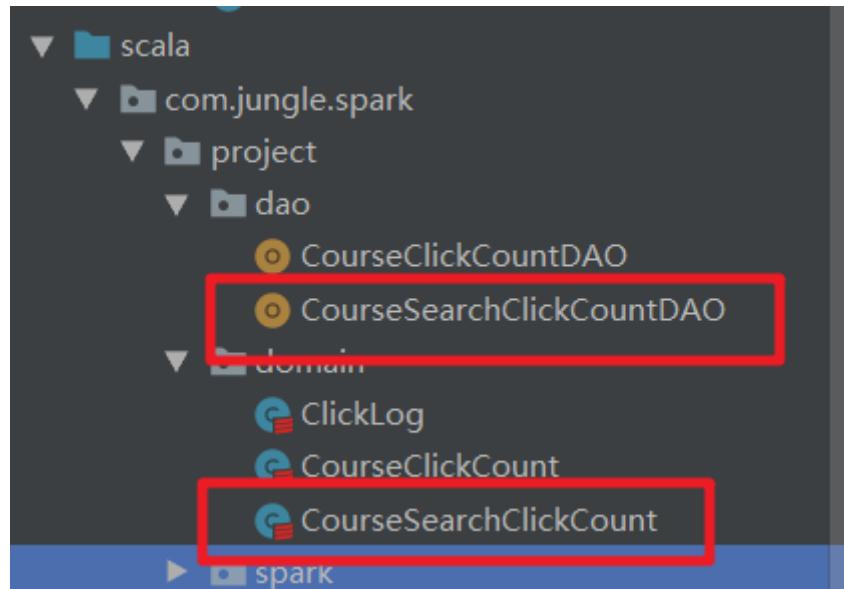
# Hbase表设计
create 'imooc_course_search_clickcount', 'info'

```

```

# rowkey设计：也是根据我们的业务需求来的
20171111+ search+1

```



- CourseSearchClickCount

```
package com.jungle.spark.project.domain

/**
 * 从搜索引擎过来的实战课程点击数实体类
 * @param day_search_course
 * @param click_count
 */
case class CourseSearchClickCount(day_search_course:String,
click_count:Long)
```

- CourseSearchClickCountDAO

```
package com.jungle.spark.project.dao

import com.jungle.spark.project.domain.{CourseClickCount,
CourseSearchClickCount}
import com.jungle.spark.project.utils.HBaseUtils
import org.apache.hadoop.hbase.client.Get
import org.apache.hadoop.hbase.util.Bytes

import scala.collection.mutable.ListBuffer

/**
 * 从搜索引擎过来的实战课程点击数-数据访问层
 */
object CourseSearchClickCountDAO {

    val tableName = "imooc_course_search_clickcount"
    val cf = "info"
    val qualifier = "click_count"

    /**
     * 保存数据到HBase
     *
     * @param list CourseSearchClickCount集合
     */
}
```

```

def save(list: ListBuffer[CourseSearchClickCount]): Unit = {
    val table = HBaseUtils.getInstance().getTable(tableName)

    for(ele <- list) {
        table.incrementColumnValue(Bytes.toBytes(ele.day_search_course),
            Bytes.toBytes(cf),
            Bytes.toBytes(qualifer),
            ele.click_count)
    }
}

/**
 * 根据rowkey查询值
 */
def count(day_search_course: String): Long = {
    val table = HBaseUtils.getInstance().getTable(tableName)

    val get = new Get(Bytes.toBytes(day_search_course))
    val value = table.get(get).getValue(cf.getBytes, qualifer.getBytes)

    if(value == null) {
        0L
    }else{
        Bytes.toLong(value)
    }
}

def main(args: Array[String]): Unit = {

    val list = new ListBuffer[CourseSearchClickCount]
    list.append(CourseSearchClickCount("20171111_www.baidu.com_8",8))
    list.append(CourseSearchClickCount("20171111_cn.bing.com_9",9))

    save(list)

    println(count("20171111_www.baidu.com_8") + " : " +
    count("20171111_cn.bing.com_9"))
}
}

```

==运行CourseSearchClickCountDAO==

19/08/09 16:23:43 INFO zookeeper.ZooKeeper
19/08/09 16:23:44 INFO zookeeper.ClientCnxn
19/08/09 16:23:44 INFO zookeeper.ClientCnxn
19/08/09 16:23:44 INFO zookeeper.ClientCnxn

2. 功能实现及本地测试

--清空hbase表数据--

```
truncate 'imooc.course.search.clickcount'
```

```
hbase(main):013:0> truncate 'imooc_course_search_clickcount'
Truncating 'imooc_course_search_clickcount' table (it may take a while):
 - Disabling table...
 - Truncating table...
0 row(s) in 3.4100 seconds
```

- ImoocStatStreamingApp

```
package com.jungle.spark.project.spark

import com.jungle.spark.project.dao.{CourseClickCountDAO,
CourseSearchClickCountDAO}
import com.jungle.spark.project.domain.{ClickLog, CourseClickCount,
CourseSearchClickCount}
import com.jungle.spark.project.utils.DateUtils
import org.apache.spark.SparkConf
import org.apache.spark.streaming.kafka.KafkaUtils
import org.apache.spark.streaming.{Seconds, StreamingContext}
```

```
import scala.collection.mutable.ListBuffer

/**
 * 使用Spark Streaming处理kafka过来的数据
 */
object ImoocStatStreamingApp {

    def main(args: Array[String]): Unit = {

        if (args.length != 4) {
            println("Usage: ImoocStatStreamingApp <zkQuorum> <group> <topics>
<numThreads>")
            System.exit(1)
        }

        val Array(zkQuorum, groupId, topics, numThreads) = args

        val sparkConf = new
        SparkConf().setAppName("ImoocStatStreamingApp").setMaster("local[2]")
        val ssc = new StreamingContext(sparkConf, Seconds(60))

        val topicMap = topics.split(",").map((_, numThreads.toInt)).toMap

        val messages = KafkaUtils.createStream(ssc, zkQuorum, groupId, topicMap)

        //    messages.print

        // 测试步骤一：测试数据接收
        //    messages.map(_._2).count().print
        //    messages.map(_._2).print

        // 测试步骤二：数据清洗
        val logs = messages.map(_._2)
        val cleanData = logs.map(line => {
            val infos = line.split("\t")

            // infos(2) = "GET /class/130.html HTTP/1.1"
            // url = /class/130.html
            val url = infos(2).split(" ")(1)
            var courseId = 0

            // 把实战课程的课程编号拿到了
            if (url.startsWith("/class")) {
                val courseIdHTML = url.split("/") (2)
                courseId = courseIdHTML.substring(0,
courseIdHTML.lastIndexOf(".")).toInt
            }

            ClickLog(infos(0), DateUtils.parseToMinute(infos(1)), courseId,
infos(3).toInt, infos(4))
        }).filter(clicklog => clicklog.courseId != 0)

        //    cleanData.print()

        // 测试步骤三：统计今天到现在为止实战课程的访问量
```

```

cleanData.map(x => {

    // HBase rowkey设计: 20171111_88

    (x.time.substring(0, 8) + "_" + x.courseId, 1)
}).reduceByKey(_ + _).foreachRDD(rdd => {
    rdd.foreachPartition(partitionRecords => {
        val list = new ListBuffer[CourseClickCount]

        partitionRecords.foreach(pair => {
            list.append(CourseClickCount(pair._1, pair._2))
        })

        CourseClickCountDAO.save(list)
    })
})

// 测试步骤四: 统计从搜索引擎过来的今天到现在为止实战课程的访问量
cleanData.map(x => {
    /**
     * https://www.sogou.com/web?query=Spark SQL实战
     *
     * ==>
     *
     * https://www.sogou.com/web?query=Spark SQL实战
     */
    val referer = x.referer.replaceAll("//", "/")
    val splits = referer.split("/")
    var host = ""
    if (splits.length > 2) {
        host = splits(1)
    }
    (host, x.courseId, x.time)
}).filter(_.._1 != "").map(x => {
    (x._3.substring(0, 8) + "_" + x._1 + "_" + x._2, 1)
}).reduceByKey(_ + _).foreachRDD(rdd => {
    rdd.foreachPartition(partitionRecords => {
        val list = new ListBuffer[CourseSearchClickCount]
        partitionRecords.foreach(pair => {
            list.append(CourseSearchClickCount(pair._1, pair._2))
        })
        CourseSearchClickCountDAO.save(list)
    })
})
ssc.start()
ssc.awaitTermination()

}
}

```

```

// 测试步骤四：统计从搜索引擎过来的今天到现在为止实战课程的访问量
cleanData.map(x => {
    /**
     * https://www.sogou.com/web?query=Spark SQL实战
     *
     * ==>
     *
     * https://www.sogou.com/web?query=Spark SQL实战
     */
    val referer = x.referer.replaceAll(regex = "//", replacement = "/")
    val splits = referer.split(regex = "/")
    var host = ""
    if (splits.length > 2) {
        host = splits(1)
    }
    (host, x.courseId, x.time)
}).filter(_.._1 != "").map(x => {
    (x._3.substring(0, 8) + "_" + x._1 + "_" + x._2, 1)
}).reduceByKey(_ + _).foreachRDD(rdd => {
    rdd.foreachPartition(partitionRecords => {
        val list = new ListBuffer[CourseSearchClickCount]|
        partitionRecords.foreach(pair => {
            list.append(CourseSearchClickCount(pair._1, pair._2))
        })
        CourseSearchClickCountDAO.save(list)
    })
})
ssc.start()
}

```

==运行结果==

```

hbase(main):016:0> scan 'imooc_course_search_clickcount'
ROW                                     COLUMN+CELL
20190809_cn.bing.com_110               column=info:click_count, timestamp=156534006125, value=\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x01
20190809_cn.bing.com_112               column=info:click_count, timestamp=1565340180102, value=\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x01
20190809_cn.bing.com_145               column=info:click_count, timestamp=1565340240084, value=\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x02
20190809_search.yahoo.com_112          column=info:click_count, timestamp=1565340120065, value=\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x01
20190809_search.yahoo.com_118          column=info:click_count, timestamp=1565340180080, value=\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x01
20190809_search.yahoo.com_146          column=info:click_count, timestamp=1565340240077, value=\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x01
20190809_www.baidu.com_110             column=info:click_count, timestamp=1565340120061, value=\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x01
20190809_www.baidu.com_112             column=info:click_count, timestamp=1565340240057, value=\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x01
20190809_www.baidu.com_118             column=info:click_count, timestamp=1565340180076, value=\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x02
20190809_www.baidu.com_131             column=info:click_count, timestamp=1565340240072, value=\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x03
20190809_www.baidu.com_145             column=info:click_count, timestamp=1565340120053, value=\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x01
20190809_www.baidu.com_146             column=info:click_count, timestamp=1565340180093, value=\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x01
20190809_www.sogou.com_110            column=info:click_count, timestamp=1565340180084, value=\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x01
20190809_www.sogou.com_112            column=info:click_count, timestamp=1565340240051, value=\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x01
20190809_www.sogou.com_118            column=info:click_count, timestamp=1565340240057, value=\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x02
20190809_www.sogou.com_131            column=info:click_count, timestamp=1565340061130, value=\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x01
20190809_www.sogou.com_146            column=info:click_count, timestamp=1565340240089, value=\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x04
17 row(s) in 0.0220 seconds

```

八、将项目运行在服务器环境中

1. 简单修改代码

```
1 import scala.collection.mutable.ListBuffer
2
3 /**
4  * 使用Spark Streaming处理Kafka过来的数据
5 */
6 object ImoocStatStreamingApp {
7     def main(args: Array[String]): Unit = {
8         if (args.length != 4) {
9             println("Usage: ImoocStatStreamingApp <zQuorum> <group> <topics> <numThreads>")
10            System.exit(status = 1)
11        }
12
13        val Array(zkQuorum, groupId, topics, numThreads) = args
14
15        val sparkConf = new SparkConf().setAppName("ImoocStatStreamingApp")//.setMaster("local[2]")
16        val ssc = new StreamingContext(sparkConf, Seconds(60))
17
18        val topicMap = topics.split(",").map(_._2.toInt).toMap //注释掉
19
20        val messages = KafkaUtils.createStream(ssc, zkQuorum, groupId, topicMap)
21
22        // messages.print
23
24        // 测试步骤一：测试数据接收
25        // messages.map(_._2).count().print
26        // messages.map(_._2).print
27
28    }
29
30
31
32
33
34
35
36
37
38 }
```

2. 打包编译

```
mvn clean package -DskipTests
```

```
mvn clean package -DskipTests
```

```
Terminal: Local × +
```

```
Microsoft Windows [版本 10.0.17134.885]
(c) 2018 Microsoft Corporation。保留所有权利。
```

```
E:\code\sparktrain>mvn clean package -DskipTests
```

--报错--

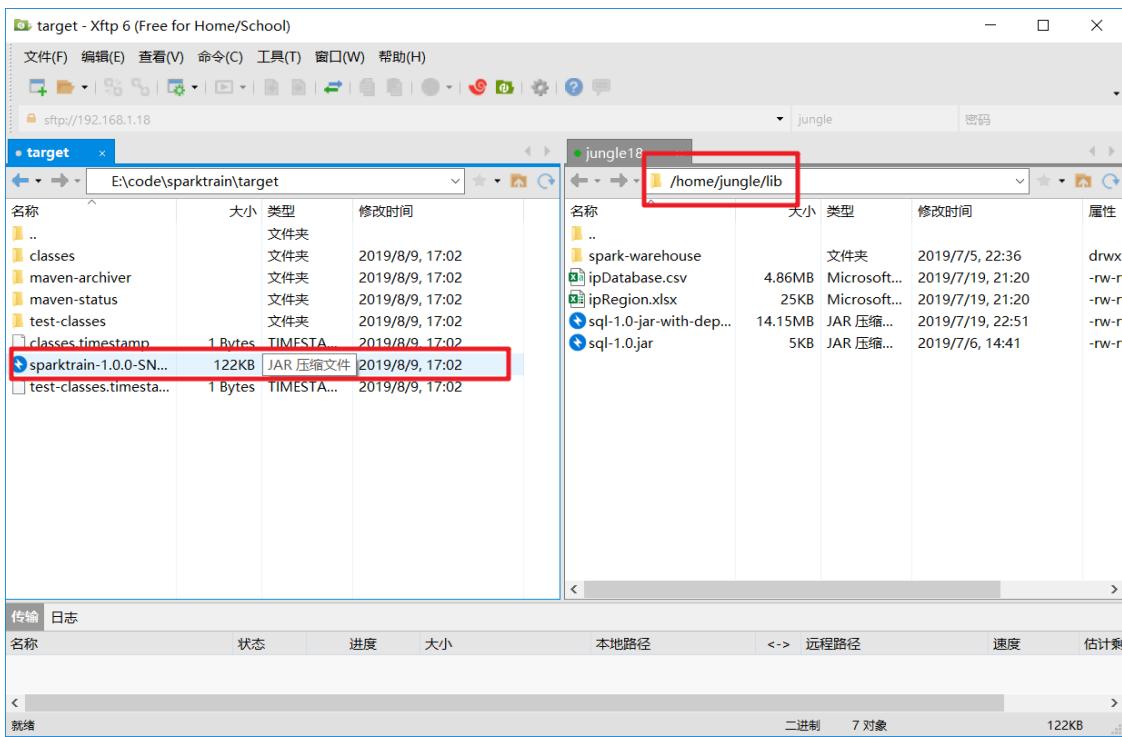
```
[ERROR]
E:\code\sparktrain\src\main\scala\com\jungle\spark\project\dao\CourseClickCountDA
O.scala:4: error: object HBaseUtils is not a member of package
com.jungle.spark.project.utils
```

--解决：

```
m sparktrain x
109      <version>3.5</version>
110    </dependency>
111    <dependency>
112      <groupId>org.apache.spark</groupId>
113      <artifactId>spark-streaming-kafka-0-8_2.11</artifactId>
114      <version>${spark.version}</version>
115    </dependency>
116    <dependency>
117      <groupId>org.apache.flume.flume-ng-clients</groupId>
118      <artifactId>flume-ng-log4jappender</artifactId>
119      <version>1.6.0</version>
120    </dependency>
121  </dependencies>
122
123  <build>
124    <sourceDirectory>src/main/scala</sourceDirectory>
125    <testSourceDirectory>src/test/scala</testSourceDirectory>
126    <plugins>
127      <plugin>
128        <groupId>org.scala-tools</groupId>
129        <artifactId>maven-scala-plugin</artifactId>
130        <executions>
131          <execution>
132            <goals>
133              <goal>compile</goal>
134              <goal>testCompile</goal>
135            </goals>
136          </execution>
137        </executions>
注释掉
```

```
m sparktrain x
109      <version>3.5</version>
110    </dependency>
111    <dependency>
112      <groupId>org.apache.spark</groupId>
113      <artifactId>spark-streaming-kafka-0-8_2.11</artifactId>
114      <version>${spark.version}</version>
115    </dependency>
116    <dependency>
117      <groupId>org.apache.flume.flume-ng-clients</groupId>
118      <artifactId>flume-ng-log4jappender</artifactId>
119      <version>1.6.0</version>
120    </dependency>
121  </dependencies>
122
123  <build>
124    <!-- <sourceDirectory>src/main/scala</sourceDirectory>-->
125    <!-- <testSourceDirectory>src/test/scala</testSourceDirectory>-->
126    <plugins>
127      <plugin>
128        <groupId>org.scala-tools</groupId>
129        <artifactId>maven-scala-plugin</artifactId>
130        <executions>
131          <execution>
132            <goals>
133              <goal>compile</goal>
134              <goal>testCompile</goal>
135            </goals>
136          </execution>
137        </executions>
```

3. 上传至服务器



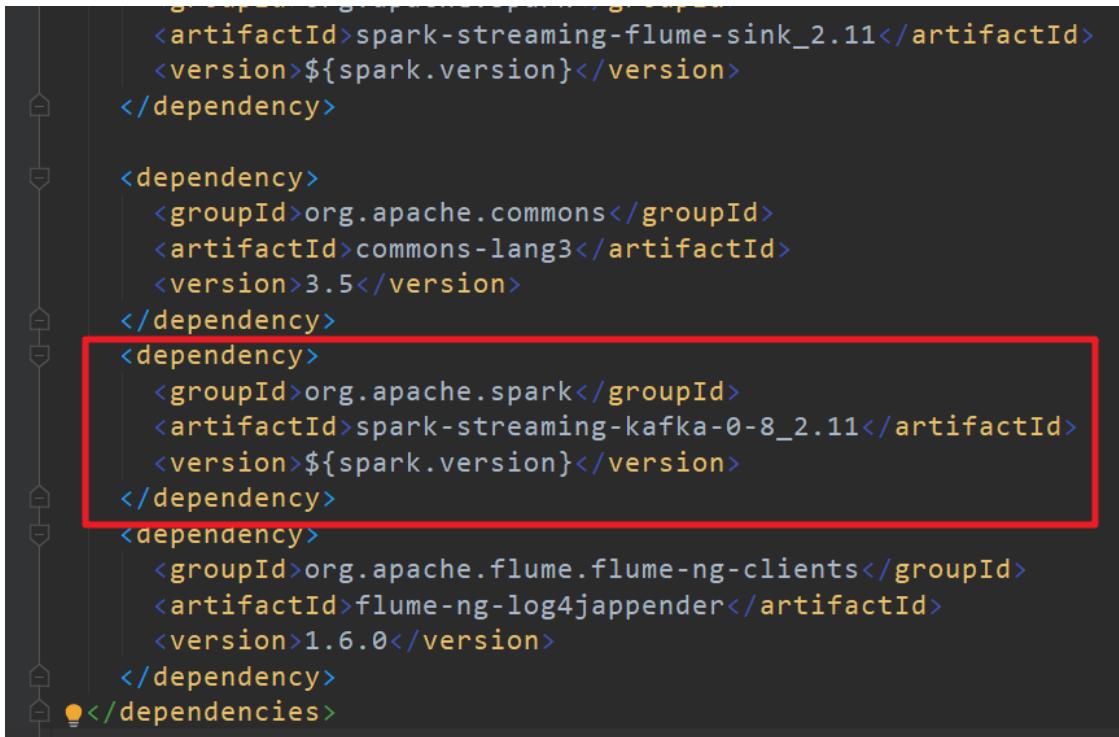
4. 提交任务

```
spark-submit --master local[5] \
--class com.jungle.spark.project.spark.imoocStatStreamingApp \
/home/jungle/lib/sparktrain-1.0.0-SNAPSHOT.jar \
centosserver1:2181 test streamingtopic 1
```

==报错==

```
Caused by: java.lang.ClassNotFoundException:
org.apache.spark.streaming.kafka.KafkaUtils$
```

找不到spark-streaming-kafka-0-8_2.11这个依赖



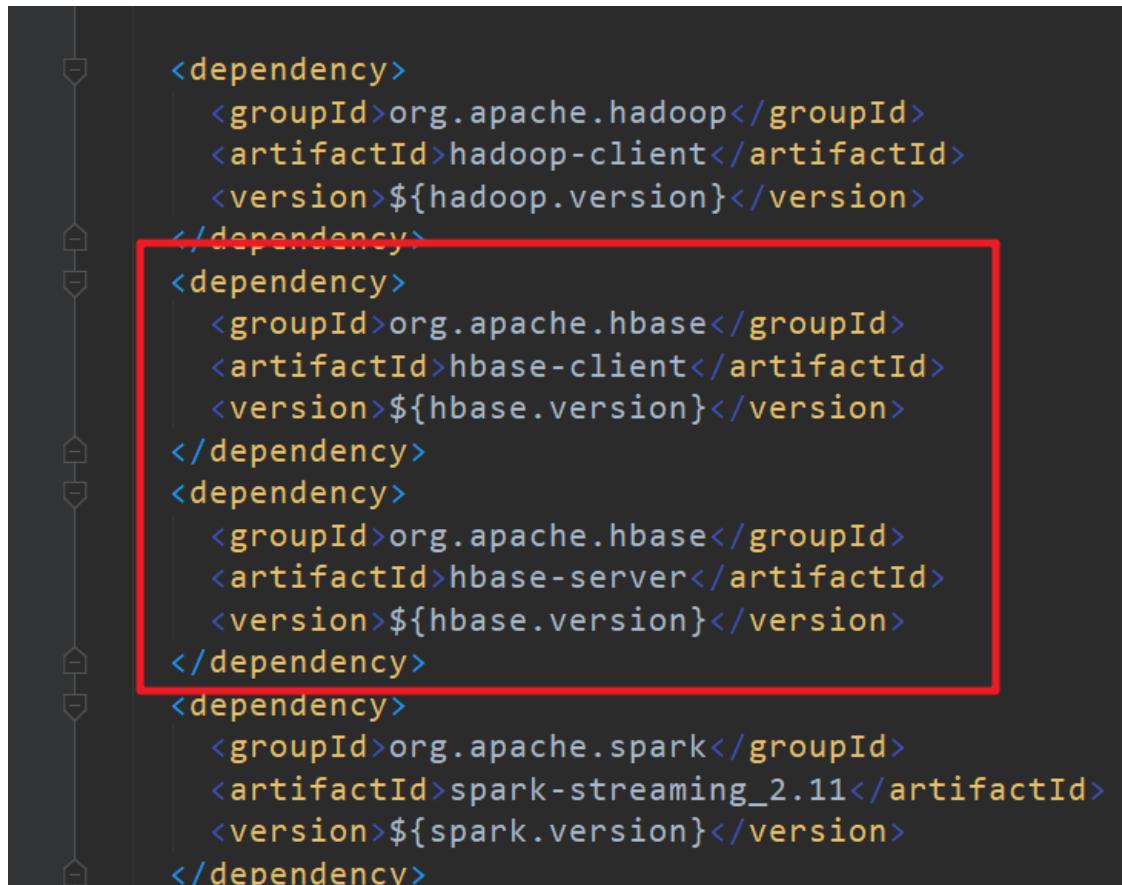
--解压：

```
spark-submit --master local[5] \
--class com.jungle.spark.project.spark.imoocStatStreamingApp \
--packages org.apache.spark:spark-streaming-kafka-0-8_2.11:2.1.0 \
/home/jungle/lib/sparktrain-1.0.0-SNAPSHOT.jar \
centosserver1:2181 test streamingtopic 1
```

==报错==

```
| java.lang.NoClassDefFoundError: org/apache/hadoop/hbase/client/HBaseAdmin
```

缺少包



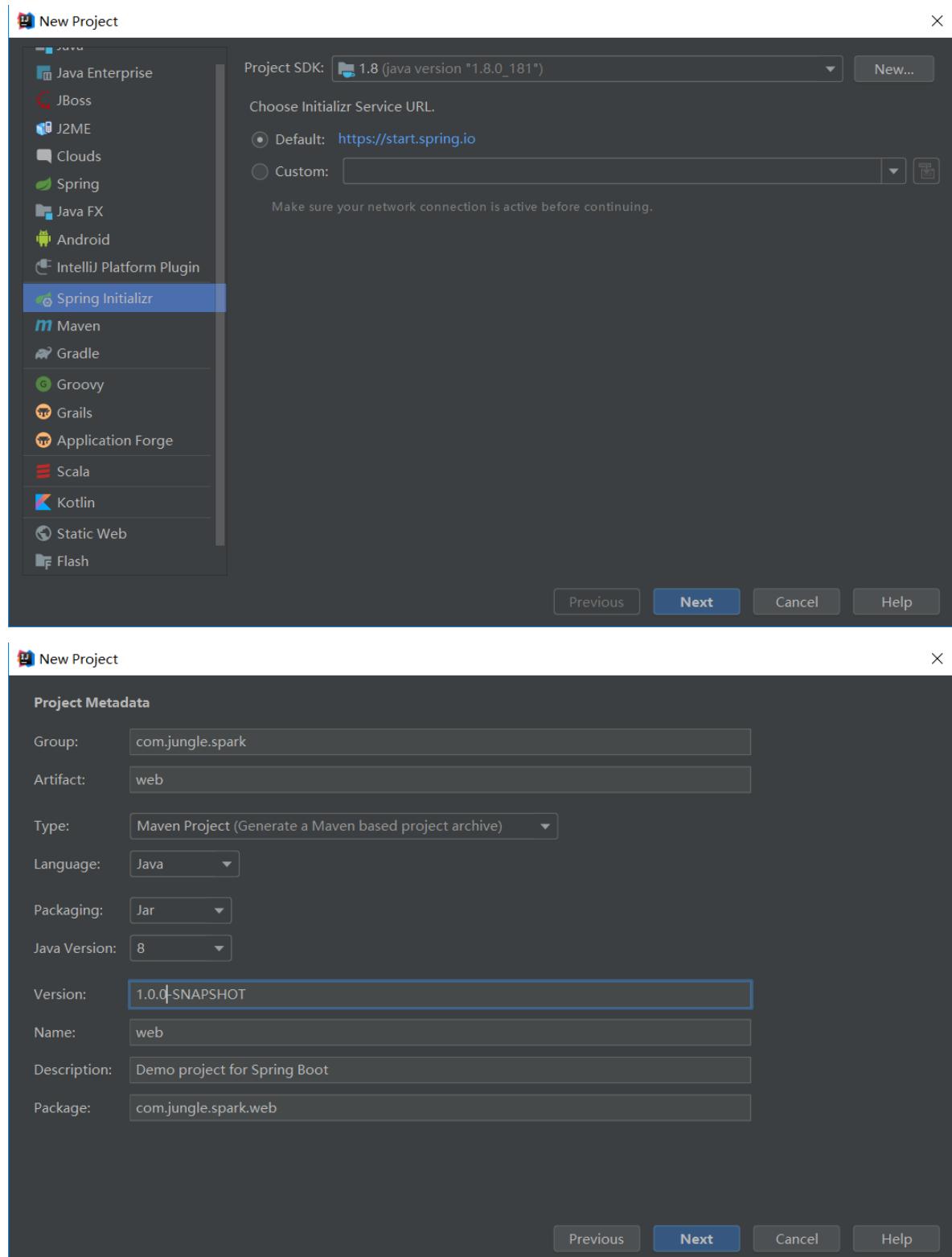
==修正==

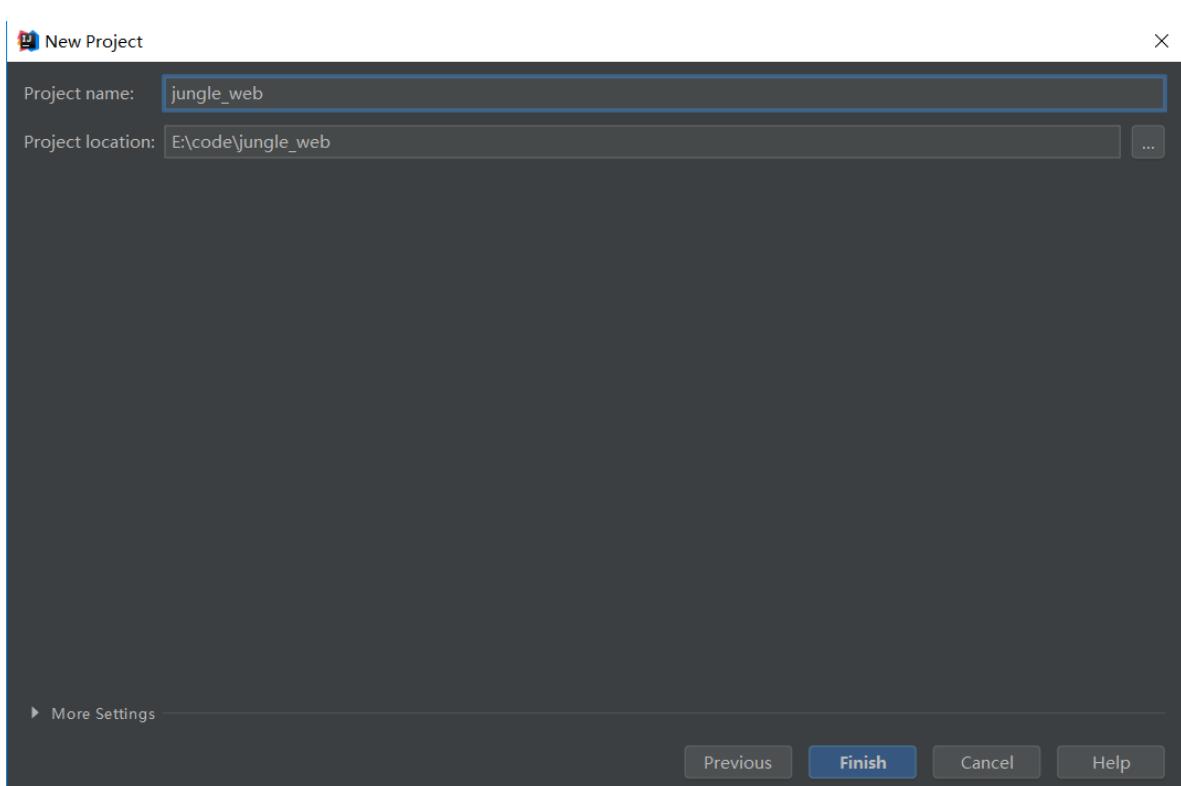
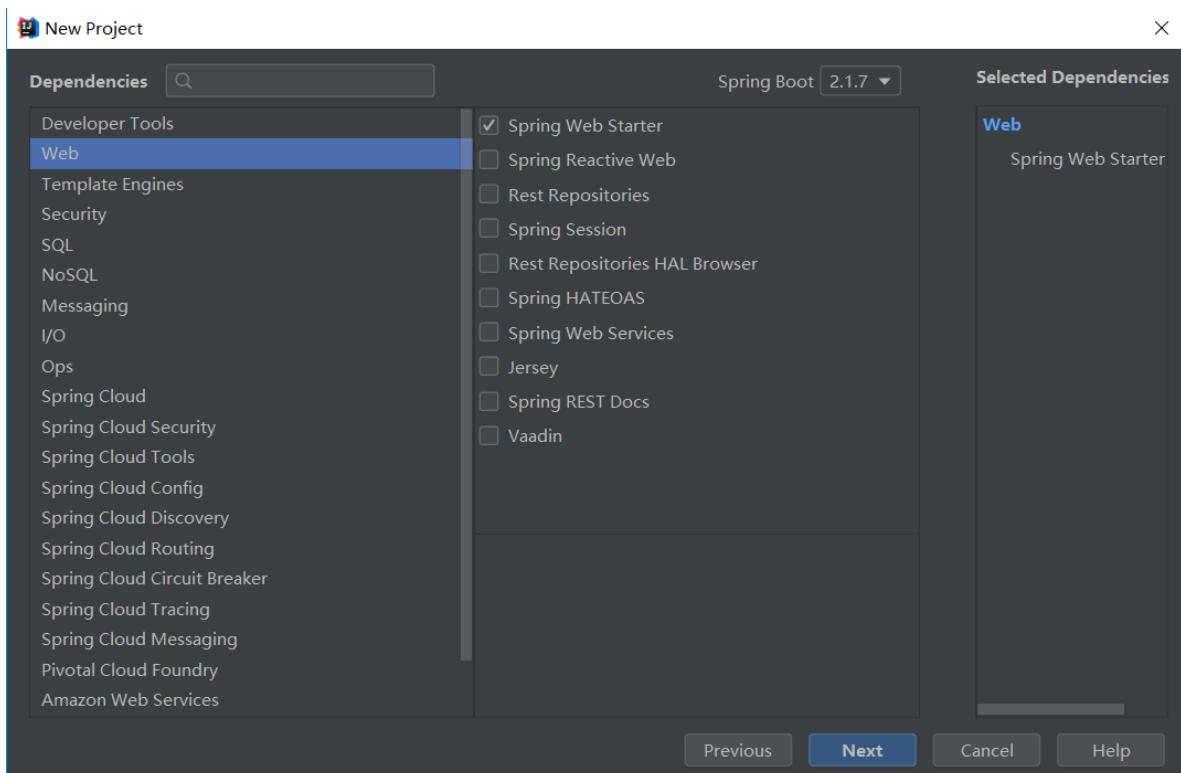
```
spark-submit --master local[5] \
--jars $(echo /home/jungle/app/hbase-1.2.0-cdh5.7.0/lib/*.jar | tr ' ' ',') \
--class com.jungle.spark.project.spark.imoocStatStreamingApp \
--packages org.apache.spark:spark-streaming-kafka-0-8_2.11:2.1.0 \
/home/jungle/lib/sparktrain-1.0.0-SNAPSHOT.jar \
centosserver1:2181 test streamingtopic 1
```

提交作业时，注意事项：
1) --packages的使用
2) --jars的使用

第13章 可视化实战

一、构建Spring Boot项目





二、Spring Boot整合Echarts

1.下载

[官网](#)

注意: 如果从镜像网站下载, 请检查 **SHA-512** 并且检验确认 OpenPGP 与 Apache 主站的签名一致。链接在上面的 Source 旁。这个 **KEYS** 文件包含了用于签名发布版的公钥。如果可能的话, 建议使用可信任的网络 (web of trust) 确认 KEYS 的同一性。

使用 GPG 验证 ECharts 发布版本

1. 从镜像网站下载 apache-echarts-X.Y.Z-incubating-src.zip
2. 从 Apache 下载 checksum apache-echarts-X.Y.Z-incubating-src.zip.asc
3. 下载 ECharts KEYS
4. gpg --import KEYS
5. gpg --verify apache-echarts-X.Y.Z-incubating-src.zip.asc

使用 SHA-512 验证

1. 从镜像网站下载 apache-echarts-X.Y.Z-incubating-src.zip
2. 从 Apache 下载 checksum apache-echarts-X.Y.Z-incubating-src.zip.sha512
3. shasum -a 512 apache-echarts-X.Y.Z-incubating-src.zip

License

Apache ECharts (incubating) 基于 [Apache License 2.0](#) 发布

点击

[在线定制 →](#)

可自由选择所需图表和组件进行打包下载

兼容 IE8

是否包括对 IE8 的兼容代码

工具集

是否在 echarts 对象上挂载常用工具集。一般都会挂载, 除非对生成的文件的体积有苛求, 并且不需要用这些工具集。

代码压缩

是否使用 UglifyJS 压缩后的代码, 开发环境建议不压缩代码, 代码压缩会去掉大部分常见的警告和错误提示。

感谢对ECharts关注与支持, 为了更好地为您提供关于 ECharts 的相关资讯, 您可以留下您的电子邮箱

someone@email.com

[下载](#)

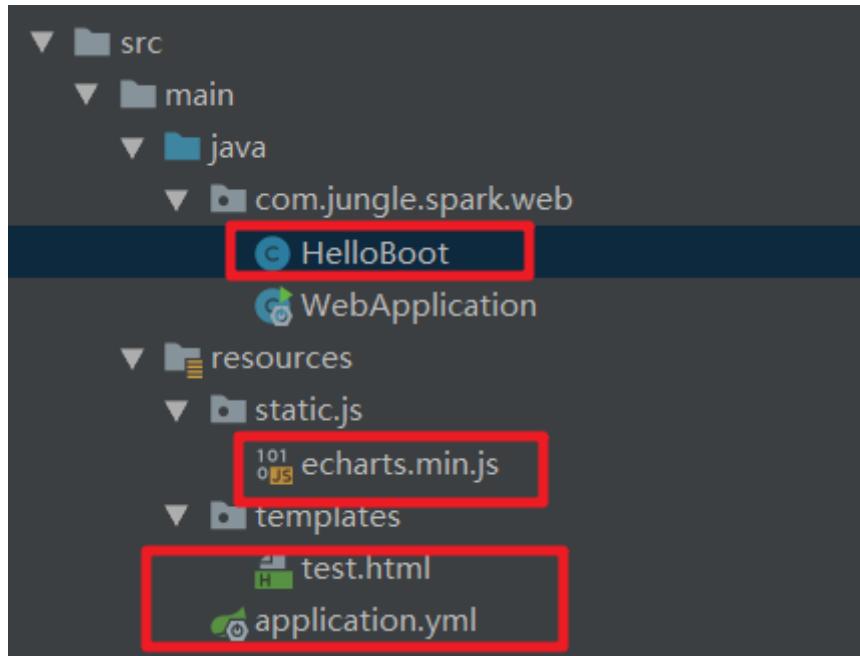
点击

2. 绘制静态数据柱状图

- 添加依赖

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
```

- 目录结构



- HelloBoot

```
package com.jungle.spark.web;

import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.servlet.ModelAndView;

/**
 * 这是我们的第一个Boot应用
 */
@RestController
public class HelloBoot {

    @RequestMapping(value = "/hello", method = RequestMethod.GET)
    public String sayHello() {
        return "Hello world Spring Boot...";
    }

    @RequestMapping(value = "/first", method = RequestMethod.GET)
    public ModelAndView firstDemo() {
        return new ModelAndView("test");
    }
}
```

```
/*
 * 
 */
@RestController
public class HelloBoot {

    @RequestMapping(value = "/hello", method = RequestMethod.GET)
    public String sayHello() {
        return "Hello World Spring Boot...";
    }

    @RequestMapping(value = "/first", method = RequestMethod.GET)
    public ModelAndView firstDemo() {
        return new ModelAndView("viewName: test");
    }
}
```

- test.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8"/>
    <title>test</title>

    <!-- 引入 ECharts 文件 -->
    <script src="js/echarts.min.js"></script>
</head>
<body>

    <!-- 为 ECharts 准备一个具备大小（宽高）的 DOM -->
    <div id="main" style="width: 600px; height:400px; position: absolute; top:50%; left: 50%; margin-top: -200px; margin-left: -300px"></div>

<script type="text/javascript">
    // 基于准备好的dom，初始化echarts实例
    var myChart = echarts.init(document.getElementById('main'));

    // 指定图表的配置项和数据
    var option = {
        title: {
            text: 'ECharts 入门示例'
        },
        tooltip: {},
        legend: {
            data: ['销量']
        },
        xAxis: {
            data: ["衬衫", "羊毛衫", "雪纺衫", "裤子", "高跟鞋", "袜子"]
        },
        yAxis: {},
        series: [
            {
                name: '销量',
                type: 'bar',
                data: [5, 20, 36, 10, 10, 20]
            }
        ];
    };

    // 使用刚指定的配置项和数据显示图表。
    myChart.setOption(option);
</script>
```

```
</body>  
</html>
```

- application.yml

```
server:  
  port: 9999  
  servlet:  
    context-path: /imooc
```

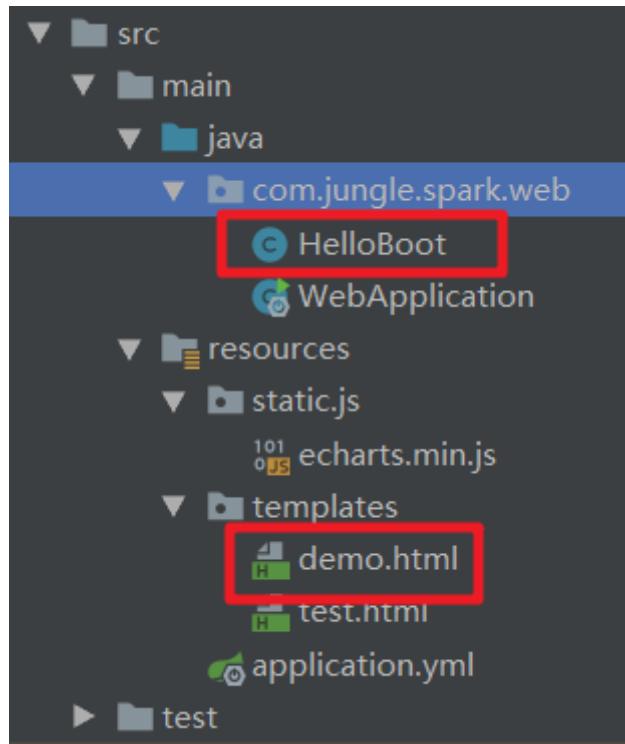
==运行效果==

<http://localhost:9999/imooc/first>



3.绘制静态数据饼图

- 目录结构



- HelloBoot

```
package com.jungle.spark.web;

import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.servlet.ModelAndView;

/**
 * 这是我们的第一个Boot应用
 */
@RestController
public class HelloBoot {

    @RequestMapping(value = "/hello", method = RequestMethod.GET)
    public String sayHello() {

        return "Hello world Spring Boot...";
    }

    @RequestMapping(value = "/first", method = RequestMethod.GET)
    public ModelAndView firstDemo() {
        return new ModelAndView("test");
    }

    @RequestMapping(value = "/course_clickcount", method =
RequestMethod.GET)
    public ModelAndView courseClickCountStat() {
        return new ModelAndView("demo");
    }

}
```

- demo.html

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8"/>
    <title>imooc_stat</title>

    <!-- 引入 ECharts 文件 -->
    <script src="js/echarts.min.js"></script>
</head>
<body>

    <!-- 为 ECharts 准备一个具备大小（宽高）的 DOM -->
    <div id="main" style="width: 600px;height:400px;position: absolute; top:50%; left: 50%; margin-top: -200px; margin-left: -300px"></div>

<script type="text/javascript">
    // 基于准备好的dom，初始化echarts实例
    var myChart = echarts.init(document.getElementById('main'));

    // 指定图表的配置项和数据
    var option = {
        title : {
            text: '慕课网实战课程实时访问量统计',
            subtext: '实战课程访问次数',
            x:'center'
        },
        tooltip : {
            trigger: 'item',
            formatter: "{a} <br/>{b} : {c} ({d}%)"
        },
        legend: {
            orient: 'vertical',
            left: 'left',
            data: ['Spark SQL项目实战', 'Hadoop入门', 'Spark Streaming项目实战', '大数据面试题', 'Storm项目实战']
        },
        series : [
            {
                name: '访问次数',
                type: 'pie',
                radius : '55%',
                center: ['50%', '60%'],
                data:[
                    {value:3350, name:'Spark SQL项目实战'},
                    {value:3100, name:'Hadoop入门'},
                    {value:2340, name:'Spark Streaming项目实战'},
                    {value:1350, name:'大数据面试题'},
                    {value:15480, name:'Storm项目实战'}
                ],
                itemStyle: {
                    emphasis: {
                        shadowBlur: 10,
                        shadowOffsetX: 0,
                        shadowColor: 'rgba(0, 0, 0, 0.5)'
                    }
                }
            }
        ]
    }
    myChart.setOption(option);
</script>

```

```

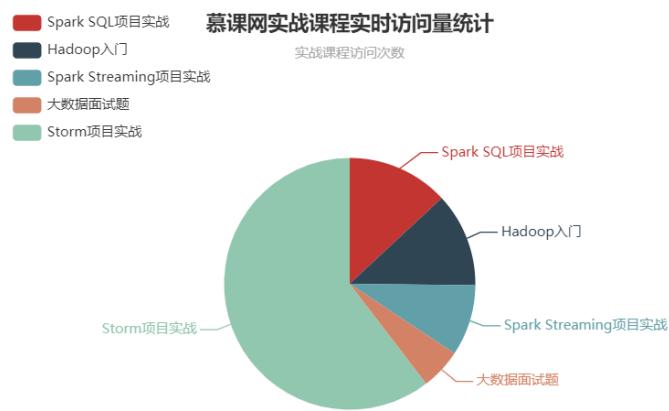
        }
    }
}

];
};

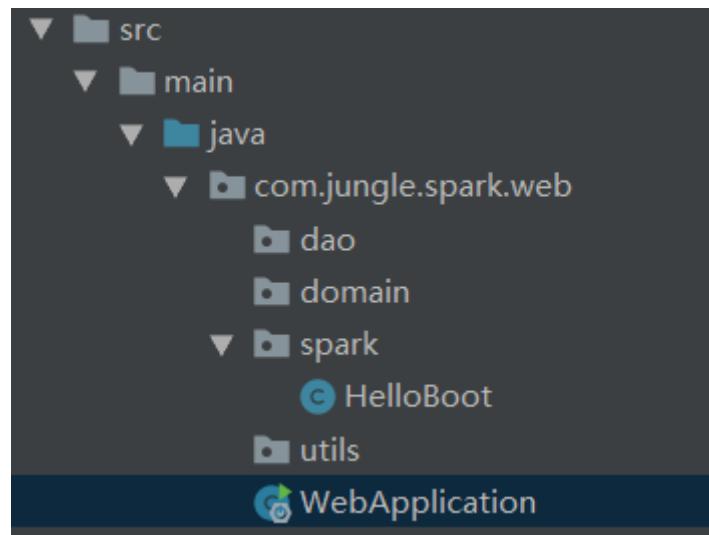
// 使用刚指定的配置项和数据显示图表。
myChart.setOption(option);
</script>
</body>
</html>

```

① localhost:9999/imooc/course_clickcount
 Google 学术搜索 平台 在线工具 算法 大数据 Chrome 网上应用店 gis docker 插件 平台 实用



三、项目目录调整



四、根据天来获取HBase表中的实战课程访问次数

1.准备

- 后台运行flume

```
nohup flume-ng agent \
--conf $FLUME_HOME/conf \
--conf-file $FLUME_HOME/conf/streaming_project2.conf \
--name exec-memory-kafka \
-Dflume.root.logger=INFO,console &
```

==报错==

```
stopping hbasecat: /tmp/hbase-jungle-master.pid: No such file or directory
```

造成上述错误的原因是，默认情况下**hbase**的**pid**文件保存在/**tmp**目录下，/**tmp**目录下的文件很容易丢失，所以造成停止集群的时候出现上述错误。解决方式是在**hbase-env.sh**中修改**pid**文件的存放路径，配置项如下所示：

```
# The directory where pid files are stored. /tmp by default.

export HBASE_PID_DIR=/var/hadoop/pids
```

删表说无表，见表说有表

清除Zookeeper内存数据库中的相关数据

```
[root@node1]# zkcli.sh
```

```
[zk: localhost:2181(CONNECTED) 0] ls /
[zookeeper, hadoop-ha, hbase]
```

```
[zk: localhost:2181(CONNECTED) 1] ls /hbase
[replication, meta-region-server, rs, splitWAL, backup-masters, table-lock,
flush-table-proc, region-in-transition, online-snapshot, master, running,
balancer, recovering-regions, draining, namespace, hbaseid, table]
```

删除 /hbase/table-lock下的相关数据

```
[zk: localhost:2181(CONNECTED) 2] ls /hbase/table-lock
[google, googlebook1, hbase:namespace, t1, googlebook]
[zk: localhost:2181(CONNECTED) 4] rmr /hbase/table-lock/googlebook
[zk: localhost:2181(CONNECTED) 7] ls /hbase/table-lock
[google, googlebook1, hbase:namespace, t1]
```

删除 /hbase/table下的相关数据

```
[zk: localhost:2181(CONNECTED) 9] ls /hbase/table
[google, googlebook1, hbase:namespace, t1, googlebook]
[zk: localhost:2181(CONNECTED) 10] rmr /hbase/table/googlebook
[zk: localhost:2181(CONNECTED) 7] ls /hbase/table
[google, googlebook1, hbase:namespace, t1]
```

最后重启**hbase**，同时需要查看一下运行的进程，需要把**ZooKeeperMain**进程也删掉

2.访问hbase

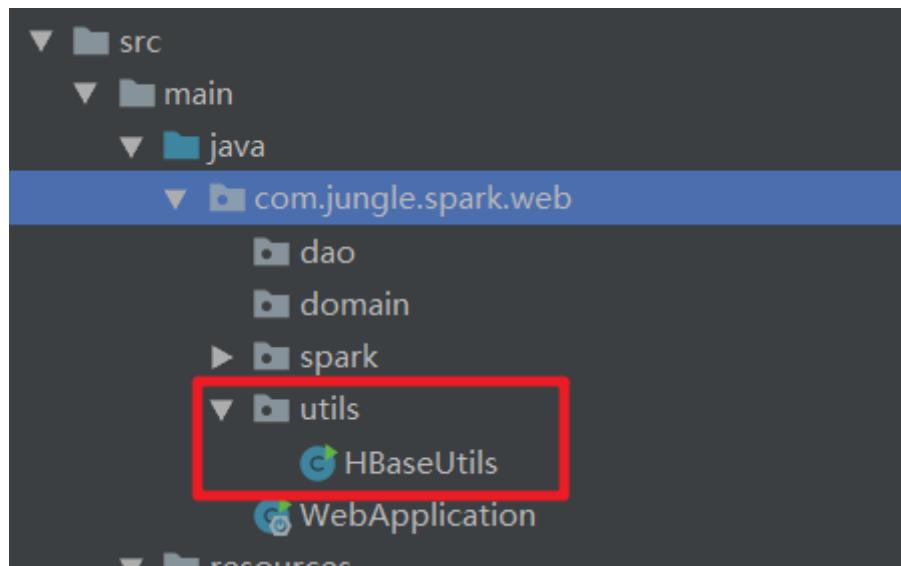
- 添加依赖

```

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

<dependency>
    <groupId>org.apache.hbase</groupId>
    <artifactId>hbase-client</artifactId>
    <version>1.2.0-cdh5.7.0</version>
</dependency>

```



```

package com.jungle.spark.web.utils;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.client.*;
import org.apache.hadoop.hbase.filter.Filter;
import org.apache.hadoop.hbase.filter.PrefixFilter;
import org.apache.hadoop.hbase.util.Bytes;

import java.io.IOException;
import java.util.HashMap;
import java.util.Map;

/**
 * HBase操作工具类
 */
public class HBaseUtils {

    HBaseAdmin admin = null;
    Configuration conf = null;

    /**
     * 私有构造方法：加载一些必要的参数
     */

```

```
private HBaseUtils() {
    conf = new Configuration();
    conf.set("hbase.zookeeper.quorum", "192.168.1.18:2181");
    conf.set("hbase.rootdir", "hdfs://192.168.1.18:8020/hbase");

    try {
        admin = new HBaseAdmin(conf);
    } catch (IOException e) {
        e.printStackTrace();
    }
}

private static HBaseUtils instance = null;

public static synchronized HBaseUtils getInstance() {
    if (null == instance) {
        instance = new HBaseutils();
    }
    return instance;
}

/**
 * 根据表名获取到HTable实例
 */
public HTable getTable(String tableName) {
    HTable table = null;

    try {
        table = new HTable(conf, tableName);
    } catch (IOException e) {
        e.printStackTrace();
    }
    return table;
}

/**
 * 根据表名和输入条件获取HBase的记录数
 */
public Map<String, Long> query(String tableName, String condition) throws
Exception {

    Map<String, Long> map = new HashMap<>();

    HTable table = getTable(tableName);
    String cf = "info";
    String qualifier = "click_count";

    Scan scan = new Scan();

    Filter filter = new PrefixFilter(Bytes.toBytes(condition));
    scan.setFilter(filter);

    ResultScanner rs = table.getScanner(scan);
    for(Result result : rs) {
        String row = Bytes.toString(result.getRow());
        long clickCount = Bytes.toLong(result.getValue(cf.getBytes(),
qualifier.getBytes()));
    }
}
```

```

        map.put(row, clickCount);
    }

    return map;
}

public static void main(String[] args) throws Exception {
    Map<String, Long> map =
HBaseUtils.getInstance().query("imooc_course_clickcount" , "20190810");

    for(Map.Entry<String, Long> entry: map.entrySet()) {
        System.out.println(entry.getKey() + " : " + entry.getValue());
    }
}

```

==运行==

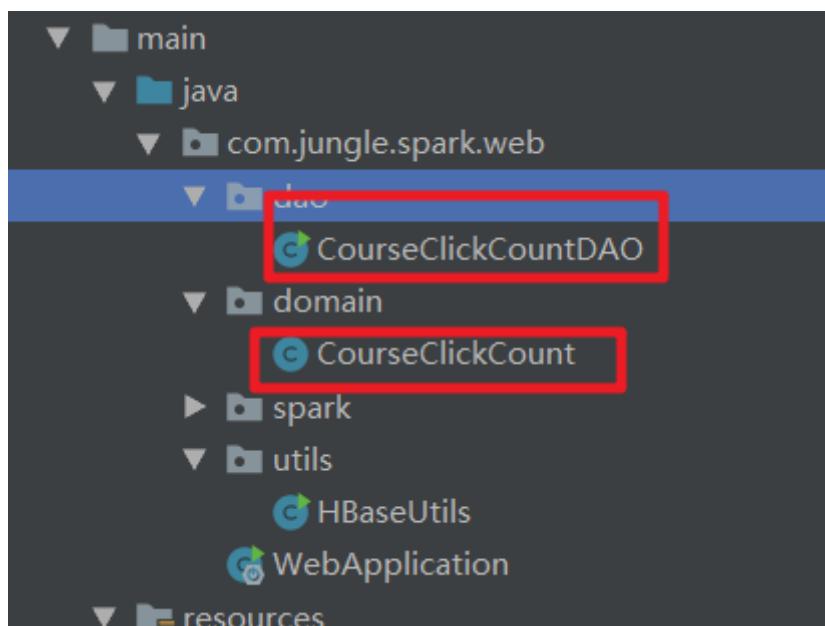
```

20:27:27.653 [main-SendThread(c
20:27:27.763 [main] DEBUG org.a
20:27:27.773 [main] DEBUG org.a
20190810_112 : 201
20190810_145 : 190
20190810_146 : 159
20190810_118 : 193
20190810_110 : 205
20190810_131 : 192

```

五、实战课程访问量domain以及dao开发

- 目录结构



- CourseClickCount

```
package com.jungle.spark.web.domain;

import org.springframework.stereotype.Component;

/**
 * 实战课程访问数量实体类
 */
@Component
public class CourseClickCount {

    private String name;
    private long value;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public long getValue() {
        return value;
    }

    public void setValue(long value) {
        this.value = value;
    }
}
```

- CourseClickCountDAO

```
package com.jungle.spark.web.dao;

import com.jungle.spark.web.domain.CourseClickCount;
import com.jungle.spark.web.utils.HBaseUtils;
import org.springframework.stereotype.Component;

import java.util.ArrayList;
import java.util.List;
import java.util.Map;

/**
 * 实战课程访问数量数据访问层
 */
@Component
public class CourseClickCountDAO {

    /**
     * 根据天查询
     */
    public List<CourseClickCount> query(String day) throws Exception {
```

```

        List<CourseClickCount> list = new ArrayList<>();

        // 去HBase表中根据day获取实战课程对应的访问量
        Map<String, Long> map =
HBaseUtils.getInstance().query("imooc_course_clickcount", "20190810");

        for(Map.Entry<String, Long> entry: map.entrySet()) {
            CourseClickCount model = new CourseClickCount();
            model.setName(entry.getKey());
            model.setValue(entry.getValue());

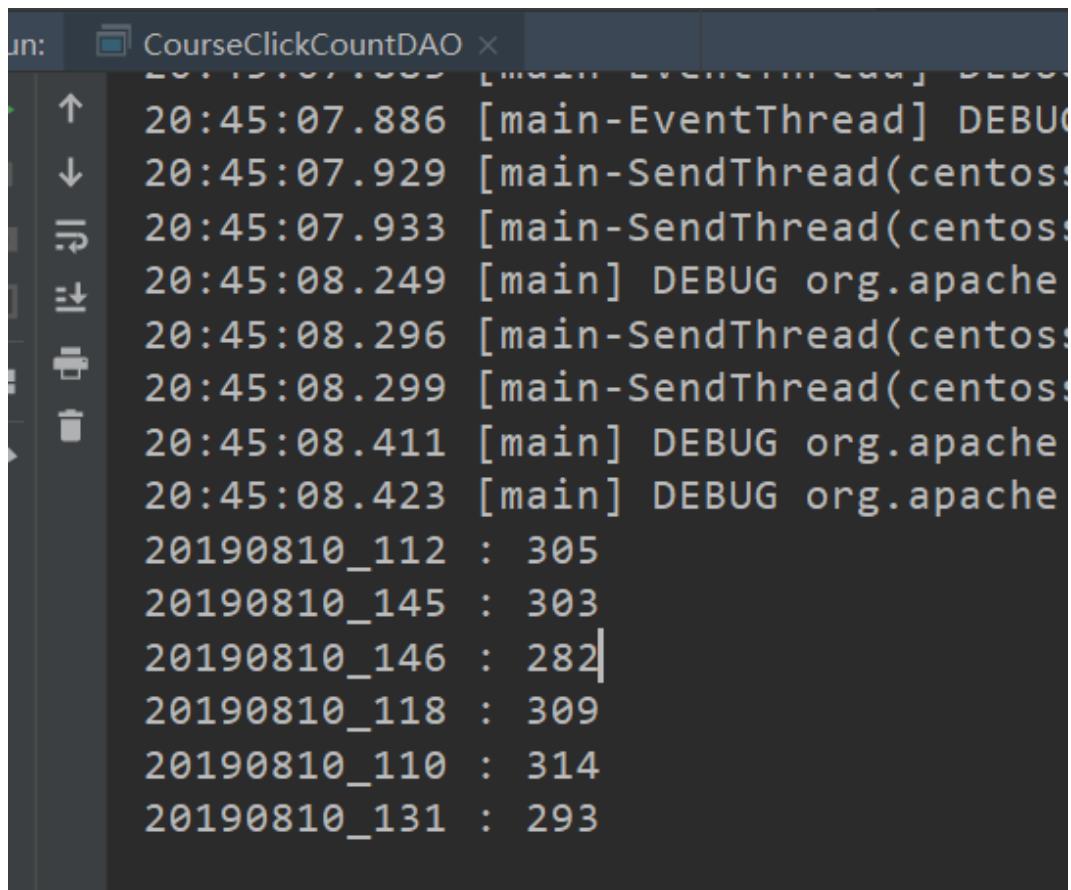
            list.add(model);
        }

        return list;
    }

    public static void main(String[] args) throws Exception{
        CourseClickCountDAO dao = new CourseClickCountDAO();
        List<CourseClickCount> list = dao.query("20190810");
        for(CourseClickCount model : list) {
            System.out.println(model.getName() + " : " +
model.getValue());
        }
    }
}

```

==运行CourseClickCountDAO==

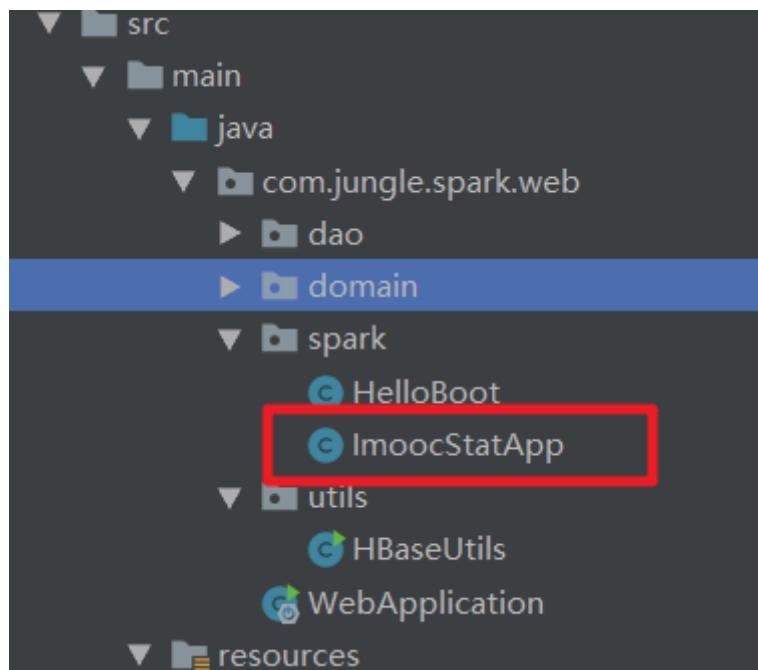


```

In:  CourseClickCountDAO x
20:45:07.886 [main-EventThread] DEBUG
20:45:07.929 [main-SendThread(centos)] DEBUG
20:45:07.933 [main-SendThread(centos)] DEBUG
20:45:08.249 [main] DEBUG org.apache.hadoop.hbase.client
20:45:08.296 [main-SendThread(centos)] DEBUG
20:45:08.299 [main-SendThread(centos)] DEBUG
20:45:08.411 [main] DEBUG org.apache.hadoop.hbase.client
20:45:08.423 [main] DEBUG org.apache.hadoop.hbase.client
20190810_112 : 305
20190810_145 : 303
20190810_146 : 282
20190810_118 : 309
20190810_110 : 314
20190810_131 : 293

```

六、实战课程访问量Web层开发



- 引入依赖

```
<dependency>
    <groupId>net.sf.json-lib</groupId>
    <artifactId>json-lib</artifactId>
    <version>2.4</version>
    <classifier>jdk15</classifier>
</dependency>
```

- ImoocStatApp

```
package com.jungle.spark.web.spark;

import com.jungle.spark.web.dao.CourseClickCountDAO;
import com.jungle.spark.web.domain.CourseClickCount;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.servlet.ModelAndView;

import java.util.HashMap;
import java.util.List;
import java.util.Map;

/**
 * web层
 */
@RestController
public class ImoocStatApp {

    private static Map<String, String> courses = new HashMap<>();
    static {
        courses.put("112", "Spark SQL慕课网日志分析");
    }
}
```

```
        courses.put("118", "10小时入门大数据");
        courses.put("145", "深度学习之神经网络核心原理与算法");
        courses.put("146", "强大的Node.js在Web开发的应用");
        courses.put("131", "Vue+Django实战");
        courses.put("110", "Web前端性能优化");
    }

    @Autowired
    CourseClickCountDAO courseClickCountDAO;

    @RequestMapping(value = "/course_clickcount_dynamic", method =
RequestMethod.GET)
    public ModelAndView courseClickCount() throws Exception {

        ModelAndView view = new ModelAndView("index");

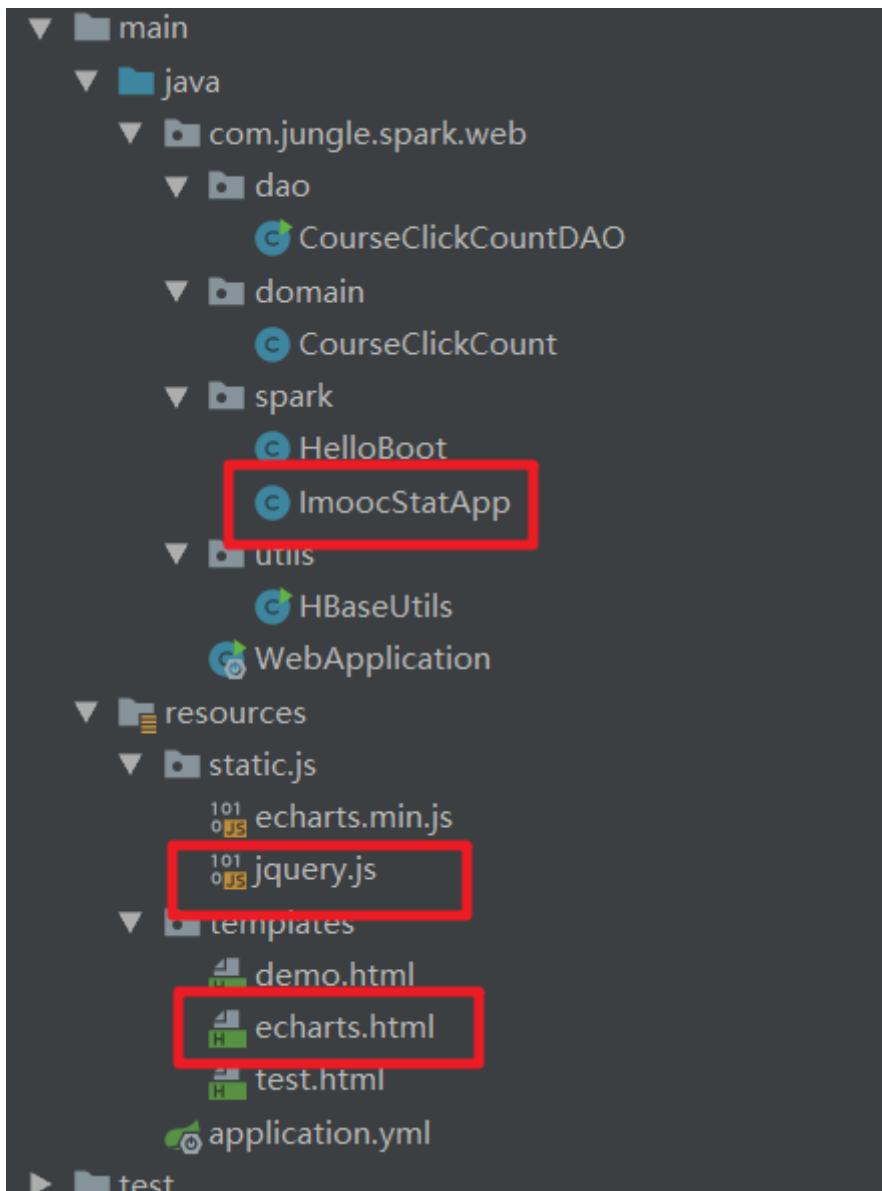
        List<CourseClickCount> list = courseClickCountDAO.query("20171022");
        for(CourseClickCount model : list) {
            model.setName(courses.get(model.getName().substring(9)));
        }
        JSONArray json = JSONArray.fromObject(list);

        view.addObject("data_json", json);

        return view;
    }

}
```

七、实战课程访问量实时查询展示功能实现及扩展



- 引入jquery.js
- charts

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8"/>
    <title>imooc_stat</title>

    <!-- 引入 ECharts 文件 -->
    <script src="js/echarts.min.js"></script>

    <!-- 引入 jQuery 文件 -->
    <script src="js/jquery.js"></script>
</head>
<body>

    <!-- 为 ECharts 准备一个具备大小（宽高）的 DOM -->
    <div id="main" style="width: 600px; height:400px; position: absolute; top:50%; left: 50%; margin-top: -200px; margin-left: -300px"></div>
```

```
<script type="text/javascript">
// 基于准备好的dom，初始化echarts实例
var myChart = echarts.init(document.getElementById('main'));

// 指定图表的配置项和数据
var option = {
    title : {
        text: '慕课网实战课程实时访问量统计',
        subtext: '实战课程访问次数',
        x:'center'
    },
    tooltip : {
        trigger: 'item',
        formatter: "{a} <br/>{b} : {c} ({d}%)"
    },
    legend: {
        orient: 'vertical',
        left: 'left'
    },
    series : [
        {
            name: '访问次数',
            type: 'pie',
            radius : '55%',
            center: ['50%', '60%'],
            data: (function(){
                //<![CDATA[
                var datas = [];
                $.ajax({
                    type: "POST",
                    url: "/imooc/course_clickcount_dynamic",
                    dataType: 'json',
                    async: false,
                    success: function(result) {
                        for(var i=0; i&lt;result.length; i++) {
                            datas.push({"value":result[i].value,
                            "name":result[i].name})
                        }
                    }
                })
                return datas;
                //]]&gt;
            })(),
            itemStyle: {
                emphasis: {
                    shadowBlur: 10,
                    shadowOffsetX: 0,
                    shadowColor: 'rgba(0, 0, 0, 0.5)'
                }
            }
        ]
    };
};

// 使用刚指定的配置项和数据显示图表。
myChart.setOption(option);
&lt;/script&gt;
&lt;/body&gt;</pre>
```

```
</html>
```

- 修改ImoocStatApp

```
package com.jungle.spark.web.spark;

import com.jungle.spark.web.dao.CourseClickCountDAO;
import com.jungle.spark.web.domain.CourseClickCount;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.servlet.ModelAndView;

import java.util.HashMap;
import java.util.List;
import java.util.Map;

/**
 * web层
 */
@RestController
public class ImoocStatApp {

    private static Map<String, String> courses = new HashMap<>();
    static {
        courses.put("112", "Spark SQL慕课网日志分析");
        courses.put("118", "10小时入门大数据");
        courses.put("145", "深度学习之神经网络核心原理与算法");
        courses.put("146", "强大的Node.js在web开发的应用");
        courses.put("131", "Vue+Django实战");
        courses.put("110", "Web前端性能优化");
    }

    @Autowired
    CourseClickCountDAO courseClickCountDAO;

    //    @RequestMapping(value = "/course_clickcount_dynamic", method =
    RequestMethod.GET)
    //    public ModelAndView courseClickCount() throws Exception {
    //
    //        ModelAndView view = new ModelAndView("index");
    //
    //        List<CourseClickCount> list =
    courseClickCountDAO.query("20171022");
    //        for(CourseClickCount model : list) {
    //            model.setName(courses.get(model.getName().substring(9)));
    //        }
    //        JSONArray json = JSONArray.fromObject(list);
    //
    //        view.addObject("data_json", json);
    //
    //        return view;
    //
```

```

//      }

    @RequestMapping(value = "/course_clickcount_dynamic", method =
RequestMethod.POST)
    @ResponseBody
    public List<CourseClickCount> courseClickCount() throws Exception {

        List<CourseClickCount> list = courseClickCountDAO.query("20190810");
        for(CourseClickCount model : list) {
            model.setName(courses.get(model.getName().substring(9)));
        }

        return list;
    }

    @RequestMapping(value = "/echarts", method = RequestMethod.GET)
    public ModelAndView echarts(){
        return new ModelAndView("echarts");
    }

}

```

- 可扩展

Spring Boot整合Echarts动态获取HBase的数据

更多怀念

- 1) 动态的传递进去当天的时间
 - a) 在代码中写死
 - b) 让你查询昨天的、前天的咋办?
在页面中放一个时间插件(jQuery插件), 默认只取当天的数据
- 2) 自动刷新展示图
每隔多久发送一个请求去刷新当前的数据供展示

统计慕课网当天实战课程从搜索引擎过来的点击量

数据已经在HBase中有的

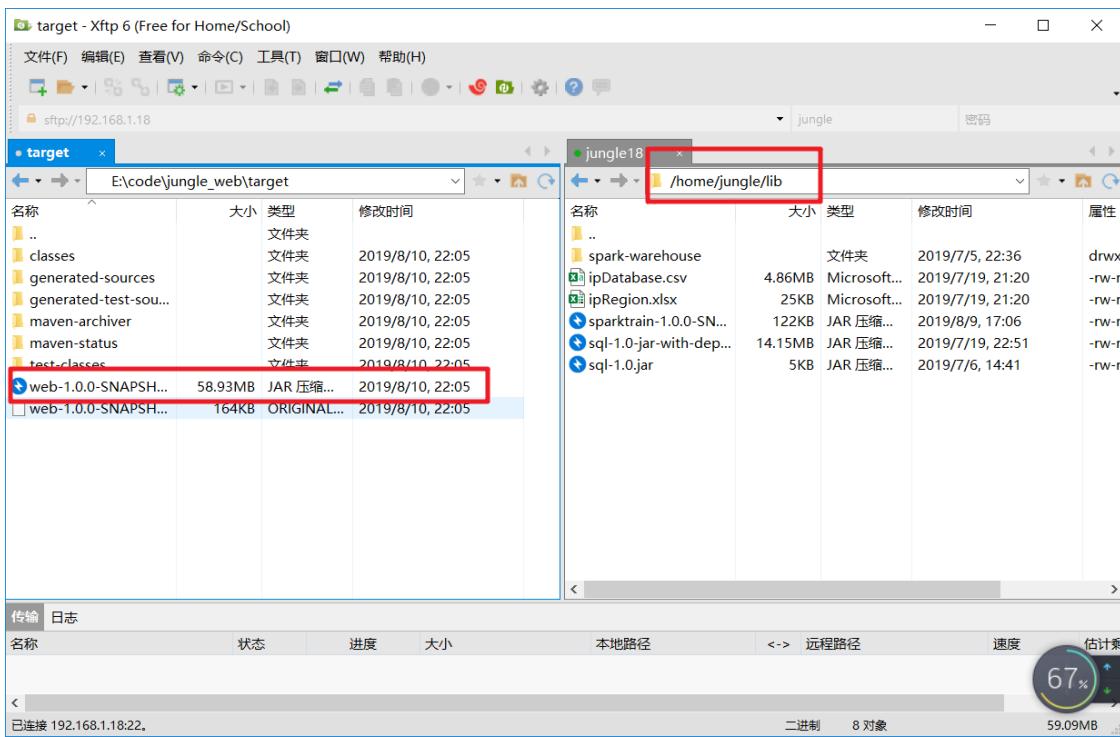
自己通过Echarts整合Spring Boot方式自己来实现

八、Spring Boot项目部署到服务器上运行

1. 打包编译

```
mvn clean package -DskipTests
```

2. 上传服务器



3. 运行

```
java -jar web-1.0.0-SNAPSHOT.jar
```