Storm 强大的数据实时处理工具，本文总结基本的入门使用，将以storm版本的"helloworld" - wordcount 进行说明

开发工具 idea

下载地址: https://download.jetbrains.8686c.com/idea/ideaIC-2017.1.3.exe

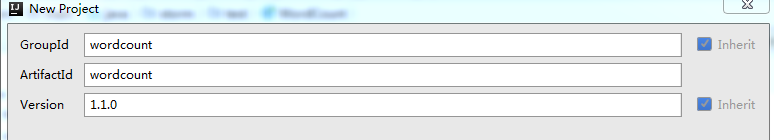
注意，需要安装jdk，(jre是没有办法使用的) ,oracle 官方免费下载安装即可

还需要安装maven 一个很神奇的东西,可以让我们不用到处寻找所需要的jar包,官方地址:http://maven.apache.org/download.cgi

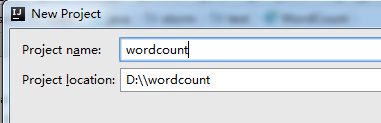
idea 和maven 使用查看 百度经验 http://jingyan.baidu.com/article/a3a3f811cd5f0b8da2eb8abf.html

打开idea，创建一个maven项目:

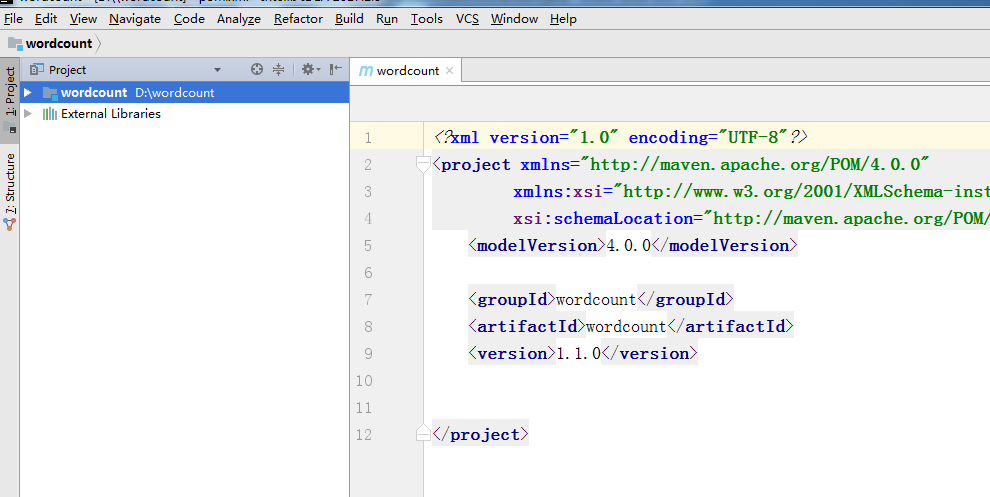
File -> New -> Project  选择左侧 Maven, 最上面的jdk，选择刚装的jdk地址，接下来填上相关信息,如下:



然后选择项目地址:



点击完成，生成项目。



一、本地产生随机句子，根据空格分词，对每个词出现次数进行统计

在pom.xml中，添加storm 核心库的依赖，将核心库jar包引入进来。添加如下代码 注意版本信息，storm-0.9 和storm-1.1库名存在差异，因为我的集群storm是1.1.0的因此我这里选择1.1.0的核心库，注意要配置增加

dependencies 插件，否则打包后文件没有依赖的jar包，在storm集群运行时需要将所有依赖的jar包，拷贝到storm 集群lib中，非常麻烦，因此将所有依赖打成一个jar，非常方便

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>wordcount</groupId>

<artifactId>wordcount</artifactId>

<version>1.1.0</version>

<dependencies>

<dependency>

<groupId>org.apache.storm</groupId>

<artifactId>storm-core</artifactId>

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

<scope>provided</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<artifactId>maven-assembly-plugin</artifactId>

<configuration>

<descriptorRefs>

<descriptorRef>jar-with-dependencies</descriptorRef>

</descriptorRefs>

<archive>

<manifest>

<mainClass/>

</manifest>

</archive>

</configuration>

<executions>

<execution>

<id>make-assembly</id>

<phase>package</phase>

<goals>

<goal>single</goal>

</goals>

</execution>

</executions>

</plugin>

<plugin>

<groupId>org.codehaus.mojo</groupId>

<artifactId>exec-maven-plugin</artifactId>

<version>1.2.1</version>

<executions>

<execution>

<goals>

<goal>exec</goal>

</goals>

</execution>

</executions>

<configuration>

<executable>java</executable>

<includeProjectDependencies>true</includeProjectDependencies>

<includePluginDependencies>false</includePluginDependencies>

<classpathScope>compile</classpathScope>

</configuration>

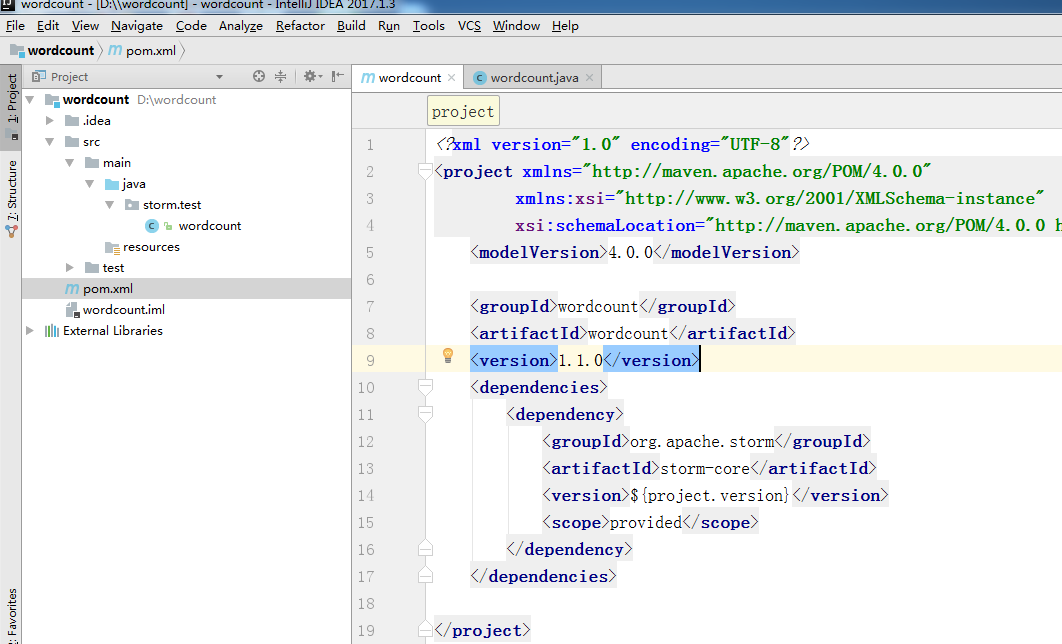
</plugin>

</plugins>

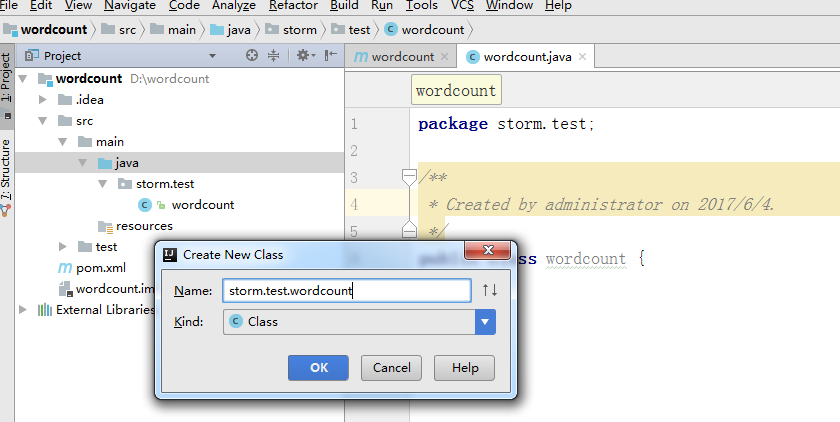
</build>

</project>

　添加后如下所示,没有红色错误，表示正确。



 点击右键，创建一个java class



 完整代码如下:

package storm.test;

/\*\*

\* Created by admin on 2017/6/4.

\*/

import org.apache.storm.Config;

import org.apache.storm.LocalCluster;

import org.apache.storm.StormSubmitter;

import org.apache.storm.spout.SpoutOutputCollector;

import org.apache.storm.task.TopologyContext;

import org.apache.storm.topology.BasicOutputCollector;

import org.apache.storm.topology.IBasicBolt;

import org.apache.storm.topology.OutputFieldsDeclarer;

import org.apache.storm.topology.TopologyBuilder;

import org.apache.storm.topology.base.BaseRichSpout;

import org.apache.storm.tuple.Fields;

import org.apache.storm.tuple.Tuple;

import org.apache.storm.tuple.Values;

import java.text.SimpleDateFormat;

import java.util.\*;

/\*\*

\* 实现一个最最基本storm

\* 随机生成单词，然后统计 单词个数

\*/

public class wordcount {

public static class WordSpout extends BaseRichSpout {

private SpoutOutputCollector collector;

private final String[] msgs = new String[]{

"I have a dream",

"my dream is to be a data analyst",

"you kan do waht you are dreaming",

"don't give up your dreams",

"it's just so so",

"we need change the traditional ideas and practice boldly",

"storm enterprise real time calculation of actual combat",

"you can be what you want be"

};

private final Random random = new Random();

public void open(Map map, TopologyContext topologyContext, SpoutOutputCollector spoutOutputCollector) {

this.collector = spoutOutputCollector;

}

//随机生成选择一个句子 发送到下一个bolt

public void nextTuple() {

String sentence = msgs[random.nextInt(8)];

collector.emit(new Values(sentence));

}

public void declareOutputFields(OutputFieldsDeclarer outputFieldsDeclarer) {

outputFieldsDeclarer.declare(new Fields("sentence"));

}

}

//第一个bolt 切分单词

public static class SplitSentenceBolt implements IBasicBolt {

public void prepare(Map map, TopologyContext topologyContext) {

}

public void execute(Tuple tuple, BasicOutputCollector basicOutputCollector) {

String sentence = tuple.getString(0);

for(String word: sentence.split(" ")){ //空格 切分单词，发送到下一个bolt

basicOutputCollector.emit(new Values(word));

}

}

public void cleanup() {

}

public void declareOutputFields(OutputFieldsDeclarer outputFieldsDeclarer) {

outputFieldsDeclarer.declare(new Fields("word"));

}

public Map<String, Object> getComponentConfiguration() {

return null;

}

}

//统计每个单词数量

public static class WordCountBolt implements IBasicBolt{

private Map<String, Integer> \_counts = new HashMap<String, Integer>();

public void prepare(Map map, TopologyContext topologyContext) {

}

public void execute(Tuple tuple, BasicOutputCollector basicOutputCollector) {

String word = tuple.getString(0);

int count ;

if(\_counts.containsKey(word)){

count = \_counts.get(word);

}else{

count = 0;

}

count++;

ShowDebugMessage(word + ":" + count);

\_counts.put(word,count);

basicOutputCollector.emit(new Values(word,count));

}

public void cleanup() {

//当关闭时调用此方法将计算保存到数据库中或者输出打印

ShowDebugMessage("--------------close bolt");

for(String tmp: \_counts.keySet())

{

ShowDebugMessage(tmp + " " + \_counts.get(tmp));

// insert(tmp,\_counts.get(tmp));

}

}

public void declareOutputFields(OutputFieldsDeclarer outputFieldsDeclarer) {

outputFieldsDeclarer.declare(new Fields("word","count"));

}

public Map<String, Object> getComponentConfiguration() {

return null;

}

}

public static void ShowDebugMessage(String strMessage)

{

System.out.println(strMessage);

}

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

String topologyName = "topology";

TopologyBuilder builder = new TopologyBuilder();

builder.setSpout("spout", new WordSpout(), 2);

builder.setBolt("split", new SplitSentenceBolt(), 5).shuffleGrouping("spout");

builder.setBolt("count", new WordCountBolt(), 10).fieldsGrouping("split", new Fields("word"));

Config conf = new Config();

if (args != null && args.length > 0) {

conf.setNumWorkers(3);

StormSubmitter.submitTopologyWithProgressBar(args[0], conf, builder.createTopology());

} else {

conf.setMaxTaskParallelism(3);

conf.setDebug(true);

LocalCluster cluster = new LocalCluster();

cluster.submitTopology(topologyName, conf, builder.createTopology());

wordcount.ShowDebugMessage("-------after submit the topology---");

// while(true)

Thread.sleep(10000); //暂停一段时间 就会自动终止

cluster.killTopology(topologyName);

wordcount.ShowDebugMessage("-------this is end for this topology----");

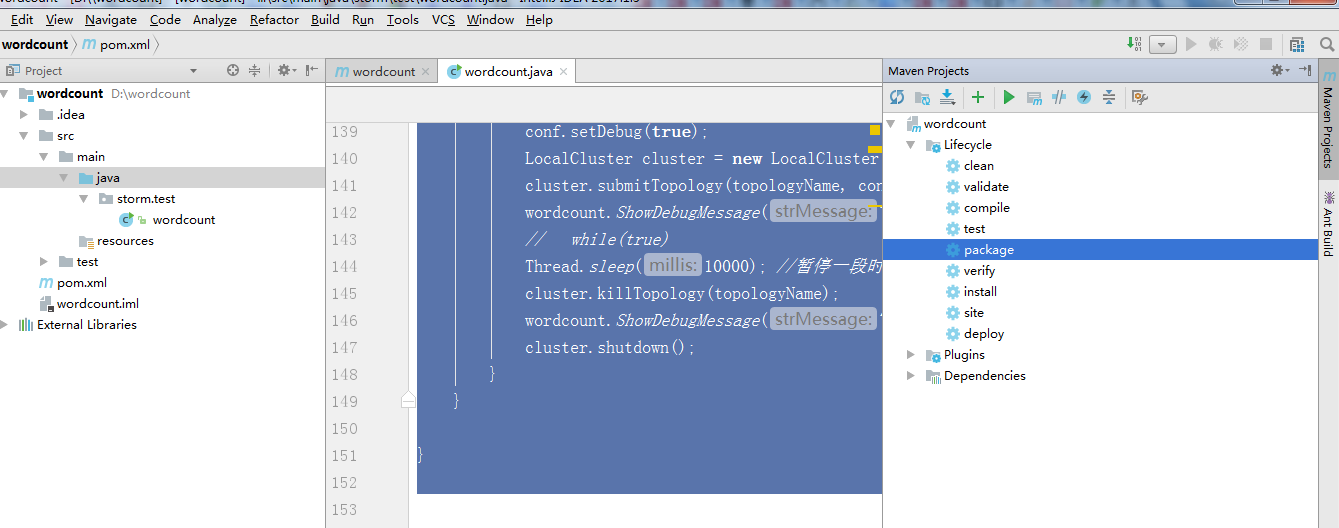
cluster.shutdown();

}

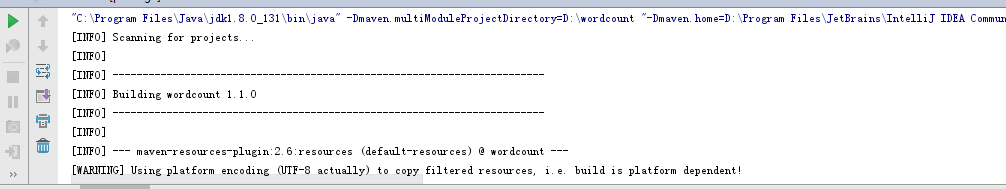
}

}

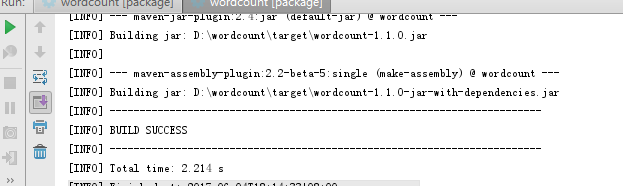
　　完成后， 开始打包,点击maven projects 选择package 点击绿色按钮，开始打包



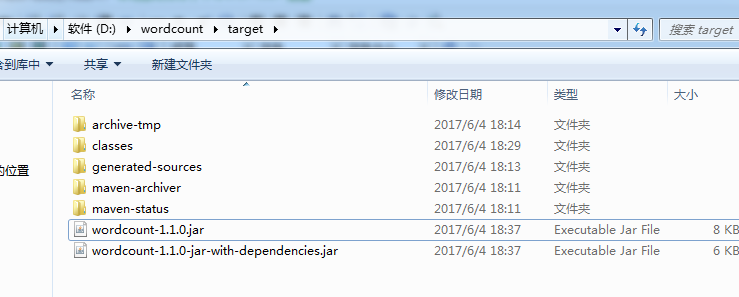
 将会看到控制台输出



最终打包后文件保存路径在控制台下有显示



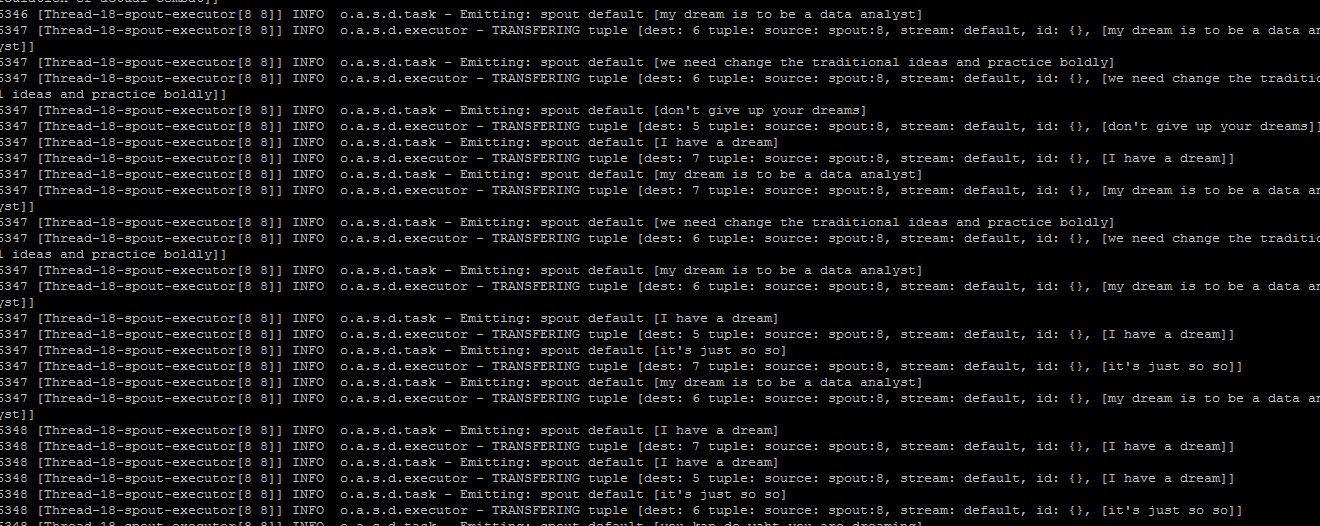
生成两个jar包



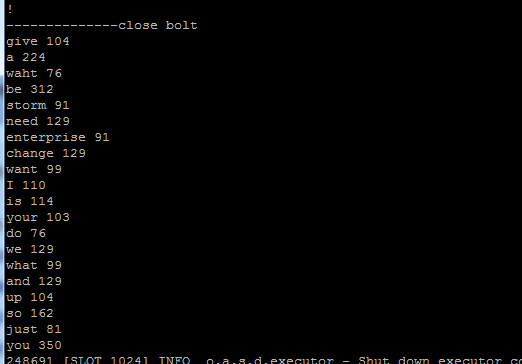
 将生成文件上传到storm集群，使用本地模式执行 如下指令

 ./storm jar /tmp/wordcount-1.1.0.jar storm.test.wordcount

查看效果



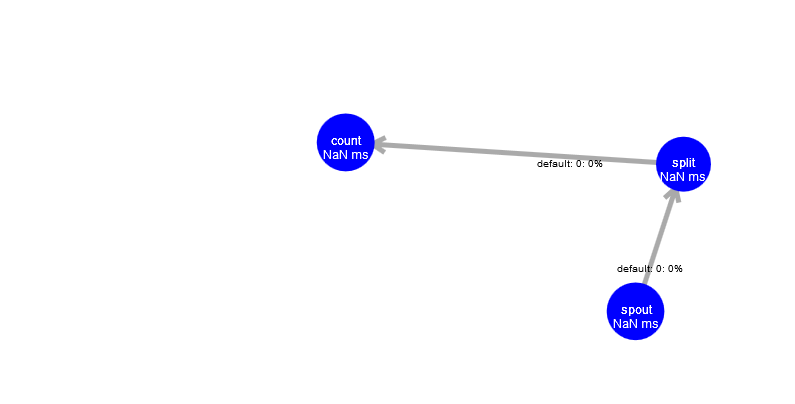
 最终执行结果



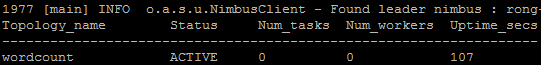
集群模式运行

./storm jar /tmp/wordcount-1.1.0.jar storm.test.wordcount wordcount

在storm ui上可以看到执行的集群拓扑



./storm list 查看当前存活的拓扑



 ./storm kill wordcount 终止某个拓扑

http://images2015.cnblogs.com/blog/485600/201706/485600-20170604191349399-1850741641.png

二、从kafka接收句子，根据空格进行分词，对每个单词出现的次数进行统计

pom.xml引入需要的jar包

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>wordcount</groupId>

<artifactId>wordcount</artifactId>

<version>1.1.0</version>

<dependencies>

<dependency>

<groupId>org.apache.storm</groupId>

<artifactId>storm-core</artifactId>

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

<scope>provided</scope>

</dependency>

<dependency>

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

<artifactId>kafka\_2.10</artifactId>

<version>0.8.1.1</version>

<exclusions>

<exclusion>

<groupId>org.apache.zookeeper</groupId>

<artifactId>zookeeper</artifactId>

</exclusion>

<exclusion>

<groupId>log4j</groupId>

<artifactId>log4j</artifactId>

</exclusion>

</exclusions>

</dependency>

<dependency>

<groupId>org.apache.storm</groupId>

<artifactId>storm-kafka</artifactId>

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

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<artifactId>maven-assembly-plugin</artifactId>

<configuration>

<descriptorRefs>

<descriptorRef>jar-with-dependencies</descriptorRef>

</descriptorRefs>

<archive>

<manifest>

<mainClass/>

</manifest>

</archive>

</configuration>

<executions>

<execution>

<id>make-assembly</id>

<phase>package</phase>

<goals>

<goal>single</goal>

</goals>

</execution>

</executions>

</plugin>

<plugin>

<groupId>org.codehaus.mojo</groupId>

<artifactId>exec-maven-plugin</artifactId>

<version>1.2.1</version>

<executions>

<execution>

<goals>

<goal>exec</goal>

</goals>

</execution>

</executions>

<configuration>

<executable>java</executable>

<includeProjectDependencies>true</includeProjectDependencies>

<includePluginDependencies>false</includePluginDependencies>

<classpathScope>compile</classpathScope>

</configuration>

</plugin>

</plugins>

</build>

</project>

　　代码如下:

package storm.test;

/\*\*

\* Created by admin on 2017/6/4.

\*/

import org.apache.storm.Config;

import org.apache.storm.LocalCluster;

import org.apache.storm.StormSubmitter;

import org.apache.storm.kafka.\*;

import org.apache.storm.kafka.trident.GlobalPartitionInformation;

import org.apache.storm.spout.SchemeAsMultiScheme;

import org.apache.storm.spout.SpoutOutputCollector;

import org.apache.storm.task.TopologyContext;

import org.apache.storm.topology.BasicOutputCollector;

import org.apache.storm.topology.IBasicBolt;

import org.apache.storm.topology.OutputFieldsDeclarer;

import org.apache.storm.topology.TopologyBuilder;

import org.apache.storm.topology.base.BaseRichSpout;

import org.apache.storm.tuple.Fields;

import org.apache.storm.tuple.Tuple;

import org.apache.storm.tuple.Values;

import java.text.SimpleDateFormat;

import java.util.\*;

/\*\*

\* 实现一个最最基本storm

\* 从kafka读取句子，然后根据空格进行分词，最终统计 单词个数

\*/

public class kafkawordcount {

//第一个bolt 切分单词

public static class SplitSentenceBolt implements IBasicBolt {

public void prepare(Map map, TopologyContext topologyContext) {

}

public void execute(Tuple tuple, BasicOutputCollector basicOutputCollector) {

String sentence = tuple.getString(0);

for(String word: sentence.split(" ")){ //空格 切分单词，发送到下一个bolt

basicOutputCollector.emit(new Values(word));

}

}

public void cleanup() {

}

public void declareOutputFields(OutputFieldsDeclarer outputFieldsDeclarer) {

outputFieldsDeclarer.declare(new Fields("word"));

}

public Map<String, Object> getComponentConfiguration() {

return null;

}

}

//统计每个单词数量

public static class WordCountBolt implements IBasicBolt{

private Map<String, Integer> \_counts = new HashMap<String, Integer>();

public void prepare(Map map, TopologyContext topologyContext) {

}

public void execute(Tuple tuple, BasicOutputCollector basicOutputCollector) {

String word = tuple.getString(0);

int count ;

if(\_counts.containsKey(word)){

count = \_counts.get(word);

}else{

count = 0;

}

count++;

ShowDebugMessage(word + ":" + count);

\_counts.put(word,count);

basicOutputCollector.emit(new Values(word,count));

}

public void cleanup() {

//当关闭时调用此方法将计算保存到数据库中或者输出打印

ShowDebugMessage("--------------close bolt");

for(String tmp: \_counts.keySet())

{

ShowDebugMessage(tmp + " " + \_counts.get(tmp));

// insert(tmp,\_counts.get(tmp));

}

}

public void declareOutputFields(OutputFieldsDeclarer outputFieldsDeclarer) {

outputFieldsDeclarer.declare(new Fields("word","count"));

}

public Map<String, Object> getComponentConfiguration() {

return null;

}

}

public static void ShowDebugMessage(String strMessage)

{

System.out.println(strMessage);

}

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

// 需要zk地址

//http://www.cnblogs.com/difeng/archive/2016/01/03/5097220.html

String topologyName = "topology";

TopologyBuilder builder = new TopologyBuilder();

int iStorm = 0;

String kafkaTopic = "my\_topic";

String ZOOKEEPER\_HOSTS = "localhost:2181,localhost2:2181";

String groupId = "wordcount";

String zkRoot = String.format("/%s\_%s", kafkaTopic, topologyName);

// 静态ip绑定

// http://blog.csdn.net/tonylee0329/article/details/43016385

BrokerHosts hosts = new ZkHosts(ZOOKEEPER\_HOSTS);

SpoutConfig spoutConfig = new SpoutConfig(hosts, kafkaTopic, zkRoot, groupId);

spoutConfig.scheme = new SchemeAsMultiScheme(new StringScheme());

KafkaSpout kafkaSpout = new KafkaSpout(spoutConfig);

// set spout

builder.setSpout("spout", kafkaSpout, 2);

builder.setBolt("split", new SplitSentenceBolt(), 5).shuffleGrouping("spout");

builder.setBolt("count", new WordCountBolt(), 10).fieldsGrouping("split", new Fields("word"));

Config conf = new Config();

if (args != null && args.length > 0) {

conf.setNumWorkers(3);

StormSubmitter.submitTopologyWithProgressBar(args[0], conf, builder.createTopology());

} else {

conf.setMaxTaskParallelism(3);

conf.setDebug(true);

LocalCluster cluster = new LocalCluster();

cluster.submitTopology(topologyName, conf, builder.createTopology());

kafkawordcount.ShowDebugMessage("-------after submit the topology---");

Thread.sleep(100000); //暂停一段时间 就会自动终止

cluster.killTopology(topologyName);

kafkawordcount.ShowDebugMessage("-------this is end for this topology----");

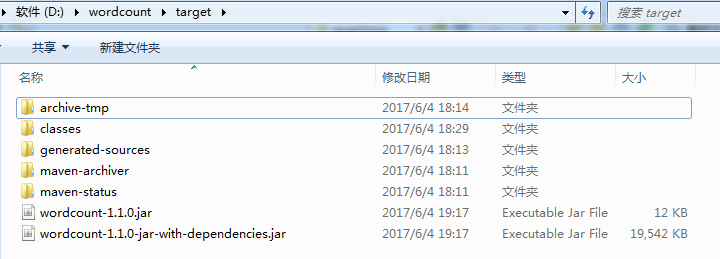
cluster.shutdown();

}

}

}

　(注意: java 连接kafka 作为消费者有两种连接方式，一种是配置zookeeper地址，非常简单，二种是静态指定kafka 9002端口并且要指定分区号，这个没有弄很清楚，如果生产者向多个分区写入数据时，怎么保证数据都能被读取出来)　点击 右侧的maven package,生成jar



发现包含dependencies的jar包 要大很多，因此引入了所有依赖的jar包，将wordcount-1.1.0-jar-with-dependencies.jar上传到集群，

本地模式执行:

./storm jar /tmp/wordcount-1.1.0-jar-with-dependencies.jar storm.test.kafkawordcout

由于写的是一个kafka的consumer，因此用python写个简单的kafka producer

#!/usr/bin/env python

# -\*- coding: utf-8 -\*-

from kafka import KafkaProducer

from kafka import KafkaConsumer

from kafka.errors import KafkaError

import json

import random

kafkalists = ["10.100.1.11:9092","10.110.1.12:9092","10.110.1.13:9092"] #写上自己的zk地址

kafkatopic = 'my\_topic' #写上自己的topic

producer = KafkaProducer(bootstrap\_servers =kafkalists)

msg = [ "I have a dream",

"my dream is to be a data analyst",

"you kan do waht you are dreaming",

"don't give up your dreams",

"it's just so so",

"we need change the traditional ideas and practice boldly",

"storm enterprise real time calculation of actual combat",

"you can be what you want be"]

for i in range(1000):

parmas\_message = msg[random.uniform(0,len(msg))]

producer.send(kafkatopic, parmas\_message.encode('utf-8'))

producer.flush()

　　随机产生1000个句子，发送到kafka队列

提交后，即可运行。

三、接收kafka，随机句子，然后根据空格分词，统计单词出现的个数，最终将结果写入到elasticsearch

storm和es对接问题，主要是版本问题，es经过几次更新，storm对应的es包也经过几次更新，最开始都是连接es的9300端口，最终测试支持最新版的es的jar包，(参考storm官方es的配置 https://github.com/apache/storm/blob/master/examples/storm-elasticsearch-examples/pom.xml）

最终结果:

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>wordcount</groupId>

<artifactId>wordcount</artifactId>

<version>1.1.0</version>

<dependencies>

<dependency>

<!-- <groupId>org.apache.storm</groupId>

<artifactId>storm-elasticsearch</artifactId>

<version>2.1.0.Beta3</version>

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

<version>2.1.0</version> -->

<groupId>org.apache.storm</groupId>

<artifactId>storm-elasticsearch</artifactId>

<version>2.0.0-SNAPSHOT</version>

</dependency>

<dependency>

<groupId>org.elasticsearch</groupId>

<artifactId>elasticsearch</artifactId>

<version>2.4.4</version>

<scope>${provided.scope}</scope>

</dependency>

<dependency>

<groupId>net.sf.json-lib</groupId>

<artifactId>json-lib</artifactId>

<version>2.4</version>

<classifier>jdk15</classifier>

</dependency>

<dependency>

<groupId>org.apache.storm</groupId>

<artifactId>storm-core</artifactId>

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

<scope>provided</scope>

</dependency>

<dependency>

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

<artifactId>kafka\_2.10</artifactId>

<version>0.8.1.1</version>

<exclusions>

<exclusion>

<groupId>org.apache.zookeeper</groupId>

<artifactId>zookeeper</artifactId>

</exclusion>

<exclusion>

<groupId>log4j</groupId>

<artifactId>log4j</artifactId>

</exclusion>

</exclusions>

</dependency>

<dependency>

<groupId>org.apache.storm</groupId>

<artifactId>storm-kafka</artifactId>

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

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<artifactId>maven-assembly-plugin</artifactId>

<configuration>

<descriptorRefs>

<descriptorRef>jar-with-dependencies</descriptorRef>

</descriptorRefs>

<archive>

<manifest>

<mainClass/>

</manifest>

</archive>

</configuration>

<executions>

<execution>

<id>make-assembly</id>

<phase>package</phase>

<goals>

<goal>single</goal>

</goals>

</execution>

</executions>

</plugin>

<plugin>

<groupId>org.codehaus.mojo</groupId>

<artifactId>exec-maven-plugin</artifactId>

<version>1.2.1</version>

<executions>

<execution>

<goals>

<goal>exec</goal>

</goals>

</execution>

</executions>

<configuration>

<executable>java</executable>

<includeProjectDependencies>true</includeProjectDependencies>

<includePluginDependencies>false</includePluginDependencies>

<classpathScope>compile</classpathScope>

</configuration>

</plugin>

</plugins>

</build>

</project>

　　java代码

package storm.test;

/\*\*

\* Created by admin on 2017/6/4.

\*/

import net.sf.json.JSONObject;

import org.apache.storm.Config;

import org.apache.storm.LocalCluster;

import org.apache.storm.StormSubmitter;

import org.apache.storm.elasticsearch.bolt.EsIndexBolt;

import org.apache.storm.elasticsearch.common.DefaultEsTupleMapper;

import org.apache.storm.elasticsearch.common.EsConfig;

import org.apache.storm.elasticsearch.common.EsTupleMapper;

import org.apache.storm.kafka.\*;

import org.apache.storm.kafka.trident.GlobalPartitionInformation;

import org.apache.storm.spout.SchemeAsMultiScheme;

import org.apache.storm.spout.SpoutOutputCollector;

import org.apache.storm.task.TopologyContext;

import org.apache.storm.topology.BasicOutputCollector;

import org.apache.storm.topology.IBasicBolt;

import org.apache.storm.topology.OutputFieldsDeclarer;

import org.apache.storm.topology.TopologyBuilder;

import org.apache.storm.topology.base.BaseRichSpout;

import org.apache.storm.tuple.Fields;

import org.apache.storm.tuple.Tuple;

import org.apache.storm.tuple.Values;

import java.text.SimpleDateFormat;

import java.util.\*;

/\*\*

\* 实现一个最最基本storm

\* 从kafka读取句子，然后根据空格进行分词，最终统计 单词个数 存入es

\*/

public class kafkawordcountes {

//第一个bolt 切分单词

public static class SplitSentenceBolt implements IBasicBolt {

public void prepare(Map map, TopologyContext topologyContext) {

}

public void execute(Tuple tuple, BasicOutputCollector basicOutputCollector) {

String sentence = tuple.getString(0);

for(String word: sentence.split(" ")){ //空格 切分单词，发送到下一个bolt

basicOutputCollector.emit(new Values(word));

}

}

public void cleanup() {

}

public void declareOutputFields(OutputFieldsDeclarer outputFieldsDeclarer) {

outputFieldsDeclarer.declare(new Fields("word"));

}

public Map<String, Object> getComponentConfiguration() {

return null;

}

}

//统计每个单词数量

public static class WordCountBolt implements IBasicBolt{

private Map<String, Integer> \_counts = new HashMap<String, Integer>();

private int iwrite = 0;

public void prepare(Map map, TopologyContext topologyContext) {

}

public void execute(Tuple tuple, BasicOutputCollector basicOutputCollector) {

String word = tuple.getString(0);

iwrite = iwrite + 1 ;

if (iwrite % 20 == 0){

iwrite = 0;

ShowDebugMessage("--------------start ssssssssssssssssssss bolt");

for(String tmp: \_counts.keySet())

{

UUID msgId = UUID.randomUUID();

JSONObject esjson = new JSONObject();

esjson.put("msg",tmp + ":" + \_counts.get(tmp));

String document = esjson.toString();  
 // 开始写入es

basicOutputCollector.emit(new Values(document,"wordcount-index", "wordcount", String.valueOf(msgId)));

}

ShowDebugMessage("--------------sssssssssssssssssssssssssssssssssssss bolt");

}

int count ;

if(\_counts.containsKey(word)){

count = \_counts.get(word);

}else{

count = 0;

}

count++;

ShowDebugMessage(word + ":" + count);

\_counts.put(word,count);

// basicOutputCollector.emit(new Values(word,count));

}

public void cleanup() {

//当关闭时调用此方法将计算保存到数据库中或者输出打印

ShowDebugMessage("--------------close bolt");

for(String tmp: \_counts.keySet())

{

// ShowDebugMessage(tmp + " " + \_counts.get(tmp));

// insert(tmp,\_counts.get(tmp));

}

}

public void declareOutputFields(OutputFieldsDeclarer outputFieldsDeclarer) {

// 默认的映射字段，如果要修改名字，则要在EsConfig中配置，使用默认即可

outputFieldsDeclarer.declare(new Fields("source", "index", "type", "id"));

}

public Map<String, Object> getComponentConfiguration() {

return null;

}

}

public static void ShowDebugMessage(String strMessage)

{

System.out.println(strMessage);

}

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

// 需要zk地址

//http://www.cnblogs.com/difeng/archive/2016/01/03/5097220.html

String topologyName = "topology";

TopologyBuilder builder = new TopologyBuilder();

int iStorm = 0;

String kafkaTopic = "topic\_nginx";

String ZOOKEEPER\_HOSTS = "localhost1:2181,localhost2:2181";

String groupId = "wordcount";

String zkRoot = String.format("/%s\_%s", kafkaTopic, topologyName);

// 静态ip绑定

// http://blog.csdn.net/tonylee0329/article/details/43016385  
 // https://github.com/tonylee0329/storm-example/blob/master/src/main/java/org/tony/storm\_kafka/common/StaticHostTopology.java

BrokerHosts hosts = new ZkHosts(ZOOKEEPER\_HOSTS);

SpoutConfig spoutConfig = new SpoutConfig(hosts, kafkaTopic, zkRoot, groupId);

spoutConfig.scheme = new SchemeAsMultiScheme(new StringScheme());

KafkaSpout kafkaSpout = new KafkaSpout(spoutConfig);

EsConfig esConfig = new EsConfig("http://localhost:9200"); //使用默认映射 source index, type,id

EsIndexBolt elasticSearchBolt = new EsIndexBolt(esConfig);

// set spout

builder.setSpout("spout", kafkaSpout, 2);

builder.setBolt("split", new SplitSentenceBolt(), 5).shuffleGrouping("spout");

builder.setBolt("count", new WordCountBolt(), 10).fieldsGrouping("split", new Fields("word"));

builder.setBolt("save", elasticSearchBolt, 3).fieldsGrouping("count", new Fields("index"));

Config conf = new Config();

if (args != null && args.length > 0) {

conf.setNumWorkers(3);

StormSubmitter.submitTopologyWithProgressBar(args[0], conf, builder.createTopology());

} else {

conf.setMaxTaskParallelism(3);

conf.setDebug(true);

LocalCluster cluster = new LocalCluster();

cluster.submitTopology(topologyName, conf, builder.createTopology());

kafkawordcountes.ShowDebugMessage("-------after submit the topology---");

Thread.sleep(100000); //暂停一段时间 就会自动终止

cluster.killTopology(topologyName);

kafkawordcountes.ShowDebugMessage("-------this is end for this topology----");

cluster.shutdown();

}

}

}

　　注意: 不需要事先创建es的索引和type，会自动创建，但是映射的字段类型也是默认的，可能不是很友好。

参考： http://www.cnblogs.com/iamlehaha/articles/6941442.html