

1 简介

远程模式(Remote Mode)

我们把我们的Topology打成jar包，通过客户端（client）提交到storm集群。

Storm的所有组件都是线程安全的，因为它们都会运行在不同的Jvm或物理机器上。

这个模式就是正式的生产模式。

注：

本地模式和远程模式只是在Topology的运行主类的代码上有所差别，主要是提交模式的代码差别

- 1、远程模式，集群提交方法，`StormSubmitter.submitTopology`
- 2、本地模式，本地提交方法，`LocalCluster.submitTopology`

本地模式和远程模式说的是运行模式，和spout与bolt的编码没有关系，也就是说编写一个storm的应用程序实际上是有两部分代码组成

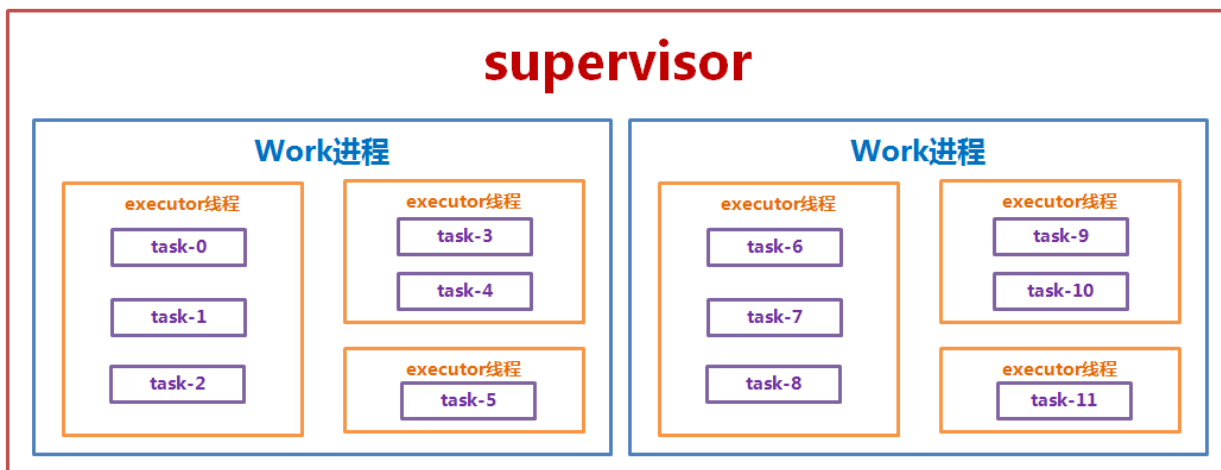
- 1、编写spout、bolt，定义一个Topology把spout、bolt关联起来，对Topology进行配置
- 2、编写Topology的提交方法，确定Topology的运行模式是本地还是远程

2 storm集群运行Topology的组件

在storm集群中，运行Topology实际上主要就是运行我们所编写的spout和bolt。

运行Topology主要依赖以下四个部分：

- 1) supervisor：指配置在一个Storm集群中的服务器，可以简单的理解为开启了supervisor进程的服务器，会执行topology的一部分运算。一个Storm集群可以包含一个或者多个supervisor。
- 2) Worker：指一个supervisor上相互独立运行的JVM进程。每个supervisor可以配置运行一个或者多个worker（storm的配置文件storm.yaml中的supervisor.slots.ports配置项，一个supervisor服务器配置了几个端口，其可以运行的最大worker数就是几）。一个topology会被分配在一个或多个worker上运行。
- 3) Executor：指一个worker的JVM中运行时开启的Java线程。多个task可以指派给同一个executor来执行。除非明确指定，Storm默认给每个executor分配一个task。
- 4) Task：task可以简单的理解为spout或者bolt的实例，它们的nextTuple和execute()方法会被executors线程执行。



3 远程模式运行Wordcount的Topology

3.1 实现思路

远程模式是在storm集群上运行Topology，我们之前的Wordcount本地模式运行案例要进行一些改造：

1) 之前的demo是spout从本地读取txt文档，这在实际开发中不会出现，spout一般是从公共的数据源获取数据，如数据库、分布式文件存储系统、缓存队列等等。那么现在我们要对spout做一个改造，不再从txt中按行读取数据，而是直接在代码中写一个字符串数组来代替txt文件

2) 解析字符串的bolt没有变化

3) 单词计数的bolt要改造。原来我们是在bolt的cleanup()方法中去打印单词数量的统计结果的，在storm集群模式下cleanup方法不会被调用。现在，我们要采用定时更新的方式，使用tick定时机制让bolt定时把统计结果更新到数据库。

注：

Apache Storm中内置了一种定时机制——tick，它能够让任何bolt的所有task每隔一段时间（精确到秒级，用户可以自定义）收到一个来自__systemd的__tick stream的tick tuple，bolt收到这样的tuple后可以根据业务需求完成相应的处理。

3) 执行主类要改为远程模式

3.2 实现代码

3.2.1 编写spout

读取字符串数组，按行发射给bolt

```
import java.util.Map;
import java.util.UUID;

import backtype.storm.spout.SpoutOutputCollector;
import backtype.storm.task.TopologyContext;
import backtype.storm.topology.IRichSpout;
import backtype.storm.topology.OutputFieldsDeclarer;
import backtype.storm.tuple.Fields;
import backtype.storm.tuple.Values;

/**
 * 读取字符串数组，按行发射给bolt
 * @author Administrator
 *
 */
public class WordReader implements IRichSpout{

    /**
     * 序列化
     */
    private static final long serialVersionUID = 1L;

    private SpoutOutputCollector collector;
    String[] lines = {"storm hadoop  hive zookeeper  spark"
        , "hadoop  hadoop  storm  spark  storm"
        , "storm  spark  storm  hadoop  hadoop"};
    int i = 0;
    /**
     * 此方法用于声明当前Spout的Tuple发送流的域名字，即一个
    backtype.storm.tuple.Fields对象。
     * 这个对象和public void nextTuple()接口中emit的
    backtype.storm.tuple.Values
     * 共同组成了一个元组对象 (backtype.storm.tuple.Tuple)
```

```

* 供后面接收该数据的Blot使用
* 运行TopologyBuilder的createTopology()时调用此方法
*/
@Override
public void declareOutputFields(OutputFieldsDeclarer declarer) {
    declarer.declare(new Fields("line"));
}

/**
* 运行TopologyBuilder的createTopology()时调用此方法
* 用于输出特定于Spout和Bolt实例的配置参数值对
* 此方法用于声明针对当前组件的特殊的Configuration配置，在需要的情况下会进行配置。
*/
@Override
public Map<String, Object> getComponentConfiguration() {
    //设置Topology中当前组件的线程数量上限为3
//    Map<String, Object> ret = new HashMap<String, Object>();
//    ret.put(Config.TOPOLOGY_MAX_TASK_PARALLELISM, 3);
//    return ret;
    return null;
}

/**
* 当一个Task被初始化的时候会调用此open方法。
* 在这里要将SpoutOutputCollector spoutOutputCollector对象保存下来，
* 供后面的public void nextTuple()接口使用，还可以执行一些其他的操作。
* 例如这里将txt文档转换成流，也就是初始化操作。
* 里面接收了三个参数，第一个是创建Topology时的配置，
* 第二个是所有的Topology数据，第三个是用来把Spout的数据发射给bolt
*/
@Override
public void open(Map conf, TopologyContext context,
    SpoutOutputCollector collector) {
    try {

```

```

        this.collector = collector;
    } catch (Exception e) {
        e.printStackTrace();
    }
}

```

```
/**
```

- * 当thread运行完spout实例的open()方法之后,
 - * 该spout实例处于deactivated (失效) 模式,
 - * 过段时间会变成activated (激活) 模式, 此时会调用Spout实例的activate()方法
- ```
*/
```

```
@Override
```

```
public void activate() {
```

```
}
```

```
/**
```

- \* 接口该接口实现具体的读取数据源的方法。
- \* 当该方法被调用时, 要求SpoutOutputCollector喷射tuple。
- \* 在本例中, nextTuple()方法是负责对txt文档转换成的流进行逐行读取。
- \* 将获取的数据emit (发射) 出去。
- \* emit的对象是通过public Values createValues(String line)方法
- \* 生成的backtype.storm.tuple.Values对象。
- \* 该方法从数据源的一行数据中, 选取的若干个目标值组成一个

backtype.storm.tuple.Values对象。

- \* 这个对象中可以存储不同类型的对象, 例如你可以同时将String对象,
- \* Long对象存取在一个backtype.storm.tuple.Values中emit出去。
- \* 实际上只要实现了Storm要求的序列化接口的对象都可以存储在里面。
- \* emit该值得时候需要注意,
- \* 他的内容要和declareOutputFields中声明的backtype.storm.tuple.Fields对象相匹配, 必须一一对应。

\* 他们被共同组成一个backtype.storm.tuple.Tuple元组对象, 被后面接收该数据流的对象使用。

```
*/
```

```
@Override
```

```

 public void nextTuple() {
 //由于Topology在运行之后，nextTuple()方法是无限循环执行的，在这里设置数组lines的元素发射完成之后不再发射数据
 if(i < lines.length){
 for(String line : lines){
 //数据过滤

System.out.println("====sput====="+Thread.currentThread() + "=="
+ line);

 //发射数据
 this.collector.emit(new Values(line),UUID.randomUUID());
 i++;
 }
 }
 }

/**
 * 从此spout发射的带有messageID的tuple处理成功时调用此方法。
 * 该方法的一个典型实现是把消息从队列中移走，避免被再次处理。
 */
@Override
public void ack(Object msgId) {
 System.out.println("OK:" + msgId);
}

/**
 * 从此spout发射的带有messageID的tuple处理失败时调用此方法。
 */
@Override
public void fail(Object msgId) {
 System.out.println("FAIL:" + msgId);
}

/**
 * topology终止时，执行此方法

```

```

 * 在本例子中，可以在这个阶段释放资源
 */
 @Override
 public void close() {

 }

 /**
 * 在Topology运行过程中，通过客户端执行deactivate命令，
 * 禁用指定的Topology时，被禁用的Topology的spout实例会变成
 deactivate（失效），
 * 并且会调用spout实例deactivate()方法
 */
 @Override
 public void deactivate() {

 }
}

```

---

### 3.2.2 编写bolt解析spout发射的每行数据

---

```

import java.util.Map;

import backtype.storm.task.OutputCollector;
import backtype.storm.task.TopologyContext;
import backtype.storm.topology.IRichBolt;
import backtype.storm.topology.OutputFieldsDeclarer;
import backtype.storm.tuple.Fields;
import backtype.storm.tuple.Tuple;
import backtype.storm.tuple.Values;

/**
 * 解析spout发射的tuple，拆分成一个个单词发射给下一个bolt
 * @author Administrator
 *

```

```

*/
public class WordNormalizer implements IRichBolt{

 /**
 * 序列化
 */
 private static final long serialVersionUID = 1L;

 private OutputCollector collector;
 private String line;
 private String[] words;

 /**
 * 当这个组件的task在集群中的一台worker内被初始化的时候，该函数被调用。
 * 它向bolt提供了该bolt执行的环境
 * (
 * 里面接收了三个参数，
 * 第一个是创建Topology时的配置，
 * 第二个是所有的Topology数据，
 * 第三个是用来把Bolt的数据发射给下一个bolt
 *)
 * 在这里要将OutputCollector collector对象保存下来
 */
 @Override
 public void prepare(Map stormConf, TopologyContext context,
 OutputCollector collector) {
 this.collector = collector;
 }

 /**
 * 处理输入的一个单一tuple。
 * 每次接收到元组时都会被调用一次，还会再发布若干个元组
 */
 @Override
 public void execute(Tuple input) {

```



```
// line = (String)input.getValue(0);
 line = (String)input.getValueByField("line");//与上面等价
 words = line.split("\t");
 System.out.println("====bolt===="+Thread.currentThread()
+ "==" + line);
 for(String word : words){
 collector.emit(new Values(word));
 }
```

**//成功，提示从此spout喷出的带有messageID的tuple已被完全处理，把消息从队列中移走，避免被再次处理。**

```
 this.collector.ack(input);
 }
```

```
/**
```

**\* Topology执行完毕的清理工作，比如关闭连接、释放资源等操作都会写在这里**

**\* topology终止时，执行此方法**

**\* 注意：**

**\* 由于cleanup方法并不可靠，它只在local mode下生效，Storm集群模式下cleanup不会被调用执行。很多资源得不到释放**

**\* 所以，在kill topology之前，先deactivate相应的topology。**

**\* 可以这样做，bolt中判断接收的数据为" shutdown" 就调用cleanup()方法。在cleanup()方法中释放需要释放的资源。**

```
*/
```

```
@Override
```

```
public void cleanup() {
```

```
}
```

```
/**
```

**\* 此方法用于声明当前bolt的Tuple发送流的域名字，即一个backtype.storm.tuple.Fields对象。**

**\* 这个对象和public void execute(Tuple input)接口中emit的backtype.storm.tuple.Values**

**\* 共同组成了一个元组对象 (backtype.storm.tuple.Tuple)**

```

 * 供后面接收该数据的Blot使用
 */
 @Override
 public void declareOutputFields(OutputFieldsDeclarer declarer) {
 declarer.declare(new Fields("word"));
 }

 /**
 * 用于输出特定于Spout和Bolt实例的配置参数值对
 * 此方法用于声明针对当前组件的特殊的Configuration配置，在需要的情况下会进行配置。
 */
 @Override
 public Map<String, Object> getComponentConfiguration() {
 return null;
 }
}

```

---

### 3.2.3 编写bolt对上一个bolt发射的单词进行计数

使用tick定时机制，定时把统计结果通过JDBC的方式保存到数据库

---

```

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.HashMap;
import java.util.Map;

import backtype.storm.Config;
import backtype.storm.Constants;
import backtype.storm.task.OutputCollector;
import backtype.storm.task.TopologyContext;
import backtype.storm.topology.IRichBolt;
import backtype.storm.topology.OutputFieldsDeclarer;

```

```

import backtype.storm.tuple.Tuple;

/**
 * 接收WordNormalizer发射的单词，进行计数
 * @author Administrator
 *
 */
public class WordCounter implements IRichBolt{

 /**
 * 序列化
 */
 private static final long serialVersionUID = 1L;

 Integer id;
 String name;
 Map<String, Integer> counters;
 private OutputCollector collector;

 public static final String url = "jdbc:mysql://master:3306/reclamation?
useUnicode=true&characterEncoding=utf8";
 public static final String dbname = "com.mysql.jdbc.Driver";
 public static final String user = "hadoop";
 public static final String password = "hadoop";

 public static Connection conn = null;
 public static PreparedStatement pst = null;
 public static ResultSet rs = null;

 @Override
 public void prepare(Map stormConf, TopologyContext context,
 OutputCollector collector) {
 this.counters = new HashMap<String, Integer>();
 this.collector = collector;
 this.name = context.getThisComponentId();
 }

```

```
this.id = context.getThisTaskId();
}
```

```
@Override
```

```
public void execute(Tuple input) {
```

```
 if(input.getSourceComponent().equals(Constants.SYSTEM_COMPONENT_ID)
 &&
```

```
 input.getSourceStreamId().equals(Constants.SYSTEM_TICK_STREAM_ID)){
```

```
 //如果当前得到的tuple是tick tuple，则保存数据到数据库，
```

```
 //我们在getComponentConfiguration()方法中，设置的是每6秒发送
 一个tick tuple
```

```
 //也就是说每6秒一次的频率把统计结果保存到数据库
```

```
 System.out.println("存储数据开始");
```

```
 updateWordCount(counters);
```

```
 System.out.println("存储数据结束");
```

```
 }else{
```

```
 //正常的计数逻辑
```

```
 String word = input.getString(0);
```

```
 //计数
```

```
 if(!counters.containsKey(word)){
```

```
 counters.put(word, 1);
```

```
 }else{
```

```
 Integer c = counters.get(word) + 1;
```

```
 counters.put(word, c);
```

```
 }
```

```
 }
```

```
 //成功，提示从此spout喷出的带有messageID的tuple已被完全处理，把消息
 从队列中移走，避免被再次处理。
```

```
 this.collector.ack(input);
```

```
 }
```

```
/**
 * Topology执行完毕的清理工作，比如关闭连接、释放资源等操作都会写在这里
 * topology终止时，执行此方法
 * 注意：
 * 由于cleanup方法并不可靠，它只在local mode下生效，Storm集群模式下
cleanup不会被调用执行。很多资源得不到释放
 * 所以，在kill topology之前，先deactivate相应的topology。
 * bolt中判断接收的数据为" shutdown" 就调用cleanup()方法。在cleanup()
方法中释放需要释放的资源。
```

```
*/
@Override
public void cleanup() {

}
```

```
@Override
public void declareOutputFields(OutputFieldsDeclarer declarer) {
 //不再向外发射数据，此处不写代码
}
```

```
@Override
public Map<String, Object> getComponentConfiguration() {
 Config conf = new Config();
 //设置定时向当前的bolt发送tick tuple的时间，单位是秒
 conf.put(Config.TOPOLOGY_TICK_TUPLE_FREQ_SECS, 6);
 return conf;
}
```

```
/**
 * 向数据库添加或更新单词计数结果
 * @param counters
 */
public void updateWordCount(Map<String, Integer> counters){
 try {
```

取连接

```
Class.forName(dbname);// 指定连接类型
conn = DriverManager.getConnection(url, user, password);// 获

conn.setAutoCommit(false);

int count = 0;

if(counters != null && counters.size() > 0){
 String sql = "";
 for(Map.Entry<String, Integer> entry : counters.entrySet()){
 sql = "select count(1) from wordcount where word=?";
 pst = conn.prepareStatement(sql);
 pst.setString(1, entry.getKey());
 rs = pst.executeQuery();
 while(rs.next()){
 count = rs.getInt(1);
 }
 if(count > 0){
 sql = "update wordcount set count = ? where word
= ?";

 pst = conn.prepareStatement(sql);
 pst.setString(1, String.valueOf(entry.getValue()));
 pst.setString(2, entry.getKey());

 }else{
 sql = "insert into wordcount (word,count) values
(?,?)";

 pst = conn.prepareStatement(sql);
 pst.setString(1, entry.getKey());
 pst.setString(2, entry.getValue().toString());
 }
 pst.executeUpdate();
 conn.commit();
 }
}
```

```

 }

 rs.close();
 pst.close();
 conn.close();

 } catch (Exception e) {
 e.printStackTrace();
 }
}
}
}

```

---

### 3.2.4 编写storm执行主类（远程模式）

---

```

import com.storm.bolt.WordCounter;
import com.storm.bolt.WordNormalizer;
import com.storm.spout.WordReader;

import backtype.storm.Config;
import backtype.storm.StormSubmitter;
import backtype.storm.topology.TopologyBuilder;
import backtype.storm.tuple.Fields;

/**
 * storm执行主类
 * @author Administrator
 *
 */
public class WordCountTopologyMain {
 public static void main(String[] args) {
 //定义一个Topology
 TopologyBuilder builder = new TopologyBuilder();
 }
}

```

```
//第3个参数设置并行度
// builder.setSpout("spout", new WordReader(), 1);
//2个参数，默认并行度为
builder.setSpout("word-reader", new WordReader());

builder.setBolt("word-normalizer", new
WordNormalizer()).shuffleGrouping("word-reader");

builder.setBolt("word-counter", new
WordCounter(), 2).fieldsGrouping("word-normalizer", new Fields("word"));

//配置
Config conf = new Config();
//设置此Topology分配2个work
conf.put(Config.TOPOLOGY_WORKERS, 2);
conf.setDebug(false);

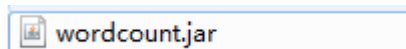
try {
 //分布式提交到storm集群
 StormSubmitter.submitTopology(args[0], conf,
builder.createTopology());
} catch (Exception e) {
 e.printStackTrace();
}

}
```

---

#### 2.1.4 远程模式运行

1) 把整个项目打成jar，我这里是wordcount.jar。



2) 在storm的nimbus节点服务器下创建一个放Topology的jar包的文件夹，我这里是/home/lby/datas。把打好的jar包上传到这个文件夹下。



```

[lby@master datas]$ cd /home/lby/datas
[lby@master datas]$ ls
hbase.jar splitdemo.txt storm.jar student student.tsv test1.log test.log wordcount.jar
[lby@master datas]$

```

3) 查看所有supervisor节点服务器的apache-storm-0.9.6/lib下是否有MySQL的支持jar包, 如果没有, 上传MySQL支持jar包到apache-storm-0.9.6/lib文件夹

```

[lby@slave lib]$ pwd
/home/lby/apache-storm-0.9.6/lib
[lby@slave lib]$ ls
asm-4.0.jar commons-io-2.4.jar jetty-util-6.1.26.jar minlog-1.2.jar snakeyaml-1.11.jar
carbonite-1.4.0.jar commons-lang-2.5.jar jgraph-core-0.9.0.jar mysql-connector-java-5.1.44-bin.jar storm-core-0.9.6.jar
chill-java-0.3.5.jar commons-logging-1.1.3.jar jline-2.11.jar reflectasm-1.07-shaded.jar tools.cli-0.2.4.jar
clj-stacktrace-0.2.2.jar commons-pool2-2.3.jar joda-time-2.0.jar ring-core-1.5.jar tools.logging-0.2.3.jar
clj-time-0.4.1.jar compojure-1.1.3.jar json-simple-1.1.1.jar ring-devel-0.3.11.jar tools.macro-0.1.0.jar
clojure-1.5.1.jar core.incubator-0.1.0.jar kryo-2.21.jar ring-jetty-adapter-0.3.11.jar zookeeper-3.4.6.jar
clout-1.0.1.jar disruptor-2.10.4.jar log4j-over-slf4j-1.6.6.jar ring-servlet-0.3.11.jar
commons-codec-1.6.jar hiccup-0.3.6.jar logback-classic-1.0.13.jar servlet-api-2.5.jar
commons-exec-1.1.jar jedis-2.6.3.jar math.numeric-tower-0.0.1.jar slf4j-api-1.7.5.jar
commons-fileupload-1.2.1.jar jetty-6.1.26.jar
[lby@slave lib]$

```

4) 运行wordcount.jar

进入放Topology的jar包的文件夹, 我这里是/home/lby/datas, 在nimbus节点执行运行Topology的命令:

命令结构: storm jar jar包名称 执行主类的包名.类名 当前Topology命名名称  
storm jar wordcount.jar com.storm.main.WordCountTopologyMain wordcount

```

8189 SecondaryNameNode
[lby@master datas]$ storm jar wordcount.jar com.storm.main.WordCountTopologyMain wordcount

```

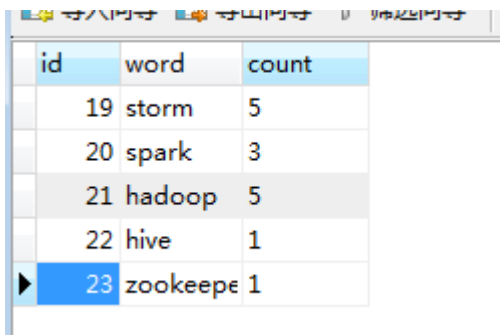
执行命令结束后如下图:

```

8189 SecondaryNameNode
[lby@master datas]$ storm jar wordcount.jar com.storm.main.WordCountTopologyMain wordcount
Running: /home/lby/jdk1.8.0_11/bin/java -client -Dstorm.options=-Dstorm.home=/home/lby/apache-storm-0.9.6 -Dstorm.log.dir=/home/lby/apache-storm-0.9.6/logs -Djava.library.path=/usr/local/lib:/opt/local/lib:/usr/lib -Dstorm.conf.file=-cp /home/lby/apache-storm-0.9.6/lib/jline-2.11.jar:/home/lby/apache-storm-0.9.6/lib/ring-servlet-0.3.11.jar:/home/lby/apache-storm-0.9.6/lib/objenesis-1.2.jar:/home/lby/apache-storm-0.9.6/lib/clout-1.0.1.jar:/home/lby/apache-storm-0.9.6/lib/tools.jar:/home/lby/apache-storm-0.9.6/lib/ring-devel-0.3.11.jar:/home/lby/apache-storm-0.9.6/lib/math.numeric-tower-0.0.1.jar:/home/lby/apache-storm-0.9.6/lib/clj-stacktrace-0.2.2.jar:/home/lby/apache-storm-0.9.6/lib/jetty-6.1.26.jar:/home/lby/apache-storm-0.9.6/lib/commons-logging-1.1.3.jar:/home/lby/apache-storm-0.9.6/lib/clojure-1.5.1.jar:/home/lby/apache-storm-0.9.6/lib/log4j-over-slf4j-1.6.6.jar:/home/lby/apache-storm-0.9.6/lib/core.incubator-0.1.0.jar:/home/lby/apache-storm-0.9.6/lib/reflectasm-1.07-shade.jar:/home/lby/apache-storm-0.9.6/lib/carbonite-1.4.0.jar:/home/lby/apache-storm-0.9.6/lib/ring-jetty-adapter-0.3.11.jar:/home/lby/apache-storm-0.9.6/lib/tools.cli-0.2.4.jar:/home/lby/apache-storm-0.9.6/lib/commons-pool2-2.3.jar:/home/lby/apache-storm-0.9.6/lib/kryo-2.21.jar:/home/lby/apache-storm-0.9.6/lib/hiccup-0.3.6.jar:/home/lby/apache-storm-0.9.6/lib/clj-time-0.4.1.jar:/home/lby/apache-storm-0.9.6/lib/commons-lang-2.5.jar:/home/lby/apache-storm-0.9.6/lib/compojure-1.1.3.jar:/home/lby/apache-storm-0.9.6/lib/commons-io-2.4.jar:/home/lby/apache-storm-0.9.6/lib/commons-codec-1.6.jar:/home/lby/apache-storm-0.9.6/lib/chill-java-0.3.5.jar:/home/lby/apache-storm-0.9.6/lib/tools.macro-0.1.0.jar:/home/lby/apache-storm-0.9.6/lib/mysql-connector-java-5.1.44-bin.jar:/home/lby/apache-storm-0.9.6/lib/ring-core-1.5.jar:/home/lby/apache-storm-0.9.6/lib/jedis-2.6.3.jar:/home/lby/apache-storm-0.9.6/lib/logback-classic-1.0.13.jar:/home/lby/apache-storm-0.9.6/lib/joda-time-2.0.jar:/home/lby/apache-storm-0.9.6/lib/jgraph-core-0.9.0.jar:/home/lby/apache-storm-0.9.6/lib/snakeyaml-1.11.jar:/home/lby/apache-storm-0.9.6/lib/commons-fileupload-1.2.1.jar:/home/lby/apache-storm-0.9.6/lib/zookeeper-3.4.6.jar:/home/lby/apache-storm-0.9.6/lib/logback-core-1.0.13.jar:/home/lby/apache-storm-0.9.6/lib/minlog-1.2.jar:/home/lby/apache-storm-0.9.6/lib/jetty-util-6.1.26.jar:/home/lby/apache-storm-0.9.6/lib/commons-exec-1.1.jar:/home/lby/apache-storm-0.9.6/lib/slf4j-api-1.7.5.jar:/home/lby/apache-storm-0.9.6/lib/json-simple-1.1.jar:/home/lby/apache-storm-0.9.6/lib/servlet-api-2.5.jar:/home/lby/apache-storm-0.9.6/bin -Dstorm.jar=wordcount.jar com.storm.main.WordCountTopologyMain wordcount
1186 [main] INFO backtype.storm.StormSubmitter - Jar not uploaded to master yet. Submitting jar...
1220 [main] INFO backtype.storm.StormSubmitter - Uploading topology jar wordcount.jar to assigned location: /home/lby/data/storm/nimbus/inbox/stormjar-0dffa95d1-de25-4b55-9162-1778c398e336.jar
1251 [main] INFO backtype.storm.StormSubmitter - Successfully uploaded topology jar to assigned location: /home/lby/data/storm/nimbus/inbox/stormjar-0dffa95d1-de25-4b55-9162-1778c398e336.jar
1251 [main] INFO backtype.storm.StormSubmitter - Submitting topology wordcount in distributed mode with conf {"topology.workers":2,"topology.debug":false}
1725 [main] INFO backtype.storm.StormSubmitter - Finished submitting topology: wordcount
[lby@master datas]$

```

5) 在数据库查看统计结果



| id | word       | count |
|----|------------|-------|
| 19 | storm      | 5     |
| 20 | spark      | 3     |
| 21 | hadoop     | 5     |
| 22 | hive       | 1     |
| 23 | zookeepers | 1     |

6) 停止wordcount.jar, 在nimbus节点执行命令:

命令结构: storm kill Topology名称

storm kill wordcount