public class **SentenceSpout** extends BaseRichSpout {

private **SpoutOutputCollector** collector;

private String[1] sentences = {

"my dog has fleas",

"don't have a cow man",

"i don't think i like fleas"

};

private int index = 0;

public void **declareOutputFields**(OutputFieldsDeclarer declarer) {

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

}

public void **open**(Map config, TopologyContext context,

SpoutOutputCollector collector) {

this.collector = collector;

}

public void **nextTuple**() {

//没有可靠性

this.collector.**emit**(new Values(sentences[index));

index++;

if (index >= sentences.length) {

index = 0;

}

Utils.waitForMillis(1);

}

}

//可靠性使用

public void open(Map config, TopologyContext context,

SpoutOutputCollector collector) {

this.collector = collector;

**this.pending = new ConcurrentHashMap<UUID, Values>();**

}

public void nextTuple() {

**Values values = new Values(sentences[index]);**

**UUID msgId = UUID.randomUUID();**

**this.pending.put(msgId, values);**

**this.collector.emit(values, msgId);**

index++;

if (index >= sentences.length) {

index = 0;

}

Utils.waitForMillis(1);

}

public void ack(Object msgId) {

this.pending.remove(msgId);

}

public void fail(Object msgId) {

this.collector.emit(this.pending.get(msgId), msgId);

}

public class **SplitSentenceBolt** extends BaseRichBolt{

private **OutputCollector** collector;

public void **prepare**(Map config, TopologyContext context, OutputCollector collector) {

this.collector = collector;

}

public void **execute**(Tuple tuple) {

String sentence = tuple.getStringByField("sentence");

String[] words = sentence.split(" ");

for(String word : words){

this.collector.**emit**(new Values(word));

}

}

public void **declareOutputFields**(OutputFieldsDeclarer declarer) {

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

}

}

可靠性方法

public void execute(Tuple tuple) {

String sentence = tuple.getStringByField("sentence");

String[] words = sentence.split(" ");

for(String word : words){

this.collector.emit(**tuple**, new Values(word));

}

**this.collector.ack(tuple);**

}

public class **WordCountBolt** extends **BaseRichBolt**{

private **OutputCollector** collector;

private **HashMap**<String, Long> counts = null;

public void **prepare**(Map config, TopologyContext context,

OutputCollector collector) {

this.collector = collector;

this.counts = new HashMap<String, Long>();

}

public void **execute**(Tuple tuple) {

String word = tuple.getStringByField("word");

Long count = this.counts.get(word);

if(count == null){

count = 0L;

}

count++;

this.counts.put(word, count);

this.collector.**emit**(new Values(word, count));

}

public void **declareOutputFields**(OutputFieldsDeclarer declarer) {

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

}}

public class **ReportBolt** extends BaseRichBolt {

private **HashMap**<String, Long> counts = null;

public void **prepare**(Map config, TopologyContext context, OutputCollector collector) {

this.counts = new HashMap<String, Long>();

}

public void **execute**(Tuple tuple) {

String word = tuple.getStringByField("word");

Long count = tuple.getLongByField("count");

this.counts.put(word, count);

}

public void **declareOutputFields**(OutputFieldsDeclarer declarer) { // this bolt does not emit anything }

@Override

public void **cleanup**() {

System.out.println("--- FINAL COUNTS ---");

List<String> keys = new ArrayList<String>();

keys.addAll(this.counts.keySet());

Collections.sort(keys);

for (String key : keys) {

System.out.println(key + " : " + this.counts.get(key)); }

System.out.println("--------------");

}

}

public class **WordCountTopology** {

private static final String SENTENCE\_SPOUT\_ID = "sentence-spout";

private static final String SPLIT\_BOLT\_ID = "split-bolt";

private static final String COUNT\_BOLT\_ID = "count-bolt";

private static final String REPORT\_BOLT\_ID = "report-bolt";

private static final String TOPOLOGY\_NAME = "word-count-topology";

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

SentenceSpout spout = new SentenceSpout();

SplitSentenceBolt splitBolt = new SplitSentenceBolt();

WordCountBolt countBolt = new WordCountBolt();

ReportBolt reportBolt = new ReportBolt();

**TopologyBuilder** builder = new TopologyBuilder();

builder.**setSpout**(SENTENCE\_SPOUT\_ID, spout)

**.setNumTasks(4);**

// SentenceSpout --> SplitSentenceBolt

builder.**setBolt**(SPLIT\_BOLT\_ID, splitBolt)

**.setNumTasks(4)**

.**shuffleGrouping**(SENTENCE\_SPOUT\_ID);

// SplitSentenceBolt --> WordCountBolt

builder.setBolt(COUNT\_BOLT\_ID, countBolt)

.**fieldsGrouping**(SPLIT\_BOLT\_ID, new Fields("word"));

// WordCountBolt --> ReportBolt

builder.setBolt(REPORT\_BOLT\_ID, reportBolt)

.**globalGrouping**(COUNT\_BOLT\_ID);

Config **config** = new Config();

config.setNumWorkers(2);

config.put(Config.TOPOLOGY\_DEBUG, false);

**//本地模式**

**LocalCluster** cluster = new LocalCluster();

cluster.**submitTopology**(TOPOLOGY\_NAME, config, builder.createTopology());

waitForSeconds(10);

cluster.killTopology(TOPOLOGY\_NAME);

cluster.shutdown();

**//提交方式**

StormSubmitter.submitTopology(TOPOLOGY\_NAME, config, builder.createTopology());

}

}

* Spout

BaseRichSpout 是Ispout和IComponent 简单实现

open()在Ispout定义,Spout初始化调用.

* Bolt

BaseRichBolt 是IBolt和IComponent 简单实现

prepare()在IBolt定义,bolt初始化调用.

IBolt.cleanup() 不保证会执行

* 流分组:

1. shuffle group随机分组: 随机分
2. fields group按字段分组: 按字段分组
3. all grouping全复制分组: 发给所有的task
4. globle grouping 全局分组: 唯一的task(最小task ID)
5. none grouping不分组,随机,为将来预留.
6. direct grouping指向分组: 指向型数据流上使用,执行组件.
7. local or shffle grouping本地或随机分组: 随机类似.发送给同一个worker内的bolt task.

* 锚定和Acker

锚定tuple: 建立输入tuple和衍生出的tuple间的对应关系.下游的tuple可以进行应答确认,超时或报错.

* acker

Storm有一个acker的特殊任务跟踪DAG图消息.当一个消息被创建时(Spout或bolt中)分配64位id.

每个消息都知道跟消息的ID,生成一个消息时,根消息的ID复制到消息中.