**样例解读**

在$HOME/storm 目录，有 examples 目录，该目录有很多的演示 examples， 特别是 storm-starter 目录下，这个版本兼容其它版本，1.1.0增加 了很多内容， 就连包名都改成了: org. apache.storm 。

看下面的案例：examples/storm-starter/src/jvm/org/apache/storm/starter/ExclamationTopology.java

下面是整个源代码：

1. \* Licensed to the Apache Software Foundation (ASF) under one
2. \* or more contributor license agreements.  See the NOTICE file
3. \* distributed with this work for additional information
4. \* regarding copyright ownership.  The ASF licenses this file
5. \* to you under the Apache License, Version 2.0 (the
6. \* "License"); you may not use this file except in compliance
7. \* with the License.  You may obtain a copy of the License at
8. \*
9. \* http://www.apache.org/licenses/LICENSE-2.0
10. \*
11. \* Unless required by applicable law or agreed to in writing, software
12. \* distributed under the License is distributed on an "AS IS" BASIS,
13. \* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
14. \* See the License for the specific language governing permissions and
15. \* limitations under the License.
16. \*/
17. package org.apache.storm.starter;
19. import org.apache.storm.Config;
20. import org.apache.storm.LocalCluster;
21. import org.apache.storm.StormSubmitter;
22. import org.apache.storm.task.OutputCollector;
23. import org.apache.storm.task.TopologyContext;
24. import org.apache.storm.testing.TestWordSpout;
25. import org.apache.storm.topology.OutputFieldsDeclarer;
26. import org.apache.storm.topology.TopologyBuilder;
27. import org.apache.storm.topology.base.BaseRichBolt;
28. import org.apache.storm.tuple.Fields;
29. import org.apache.storm.tuple.Tuple;
30. import org.apache.storm.tuple.Values;
31. import org.apache.storm.utils.Utils;
33. import java.util.Map;
35. /\*\*
36. \* This is a basic example of a Storm topology.
37. \*/
38. public class ExclamationTopology {
40. public static class ExclamationBolt extends BaseRichBolt {
41. OutputCollector \_collector;
43. @Override
44. public void prepare(Map conf, TopologyContext context, OutputCollector collector) {
45. \_collector = collector;
46. }
48. @Override
49. public void execute(Tuple tuple) {
50. \_collector.emit(tuple, new Values(tuple.getString(0) + "!!!"));
51. \_collector.ack(tuple);
52. }
54. @Override
55. public void declareOutputFields(OutputFieldsDeclarer declarer) {
56. declarer.declare(new Fields("word"));
57. }

60. }
62. public static void main(String[] args) throws Exception {
63. TopologyBuilder builder = new TopologyBuilder();
65. builder.setSpout("word", new TestWordSpout(), 10);
66. builder.setBolt("exclaim1", new ExclamationBolt(), 3).shuffleGrouping("word");
67. builder.setBolt("exclaim2", new ExclamationBolt(), 2).shuffleGrouping("exclaim1");
69. Config conf = new Config();
70. conf.setDebug(true);
72. if (args != null && args.length > 0) {
73. conf.setNumWorkers(3);
75. StormSubmitter.submitTopologyWithProgressBar(args[0], conf, builder.createTopology());
76. }
77. else {
79. LocalCluster cluster = new LocalCluster();
80. cluster.submitTopology("test", conf, builder.createTopology());
81. Utils.sleep(10000);
82. cluster.killTopology("test");
83. cluster.shutdown();
84. }
85. }
86. }

ExclamationTopology 的定义：

1. TopologyBuilder builder = new TopologyBuilder();
2. builder.setSpout("words", new TestWordSpout(), 10);
3. builder.setBolt("exclaim1", new ExclamationBolt(), 3).shuffleGrouping("words");
4. builder.setBolt("exclaim2", new ExclamationBolt(), 2).shuffleGrouping("exclaim1");

第一句话：定义 topology

第二句话：设置 Spout，Spout 产生单词

第三句话：设置 Bolt，在产生单词的基础上加“!!!“

    第四句话：设置 Bolt，在上面继续加“!!!“

    因此，如果 spout 产生 tuples ["bob"]和 ["john"], 第一个 Bolt 将产生["bob!!!"] 和 ["john!!!"]， 第二个 bolt 将产生["bob!!!!!!"] 和 ["john!!!!!!"]

一、节点定义

setSpout 、setBolt， 其中 id ，在上面例子上被定义为： words,  exclaim1 , exclaim2

二、基元的接口

org.apache.storm.topology.TopologyBuilder.setSpout(String, IRichSpout) ：SpoutDeclarer  
org.apache.storm.topology.TopologyBuilder.setSpout(String, IRichSpout, Number)：SpoutDeclarer

org.apache.storm.topology.TopologyBuilder.setBolt(String, IRichBolt)：BoltDeclarer  
org.apache.storm.topology.TopologyBuilder.setBolt(String, IRichBolt, Number)：BoltDeclarer

spout 的处理接口是 IRichSpout 接口， bolt的处理接口是 IRichBolt 接口。最后指定的10，3，2这些都是并行度，是可选的。

三、InputDeclarer对象

setBolt 能返回一个 InputDeclare 对象，这个对象能输入到 Bolt。

比如 ID exclaim1用随机分组（shuffleGrouping）声称它想读组件ID words 产生的所有的元组；

ID exclaim2 用随机分组声明它读 组件ID exclaim1产生的所有的元组。

随机分组(shufflegrouping)意思就是从输入任务到 bolt 任务 中元组能随机分布。随机分组（Shuffle Grouping）是最常用的流分组方式，它随机地分发元组到Bolt上的任务，这样能保证每个任务得到相同数量的元组。在每个 ID 上有很多方法能实现分组，后面会继续解释。

所以，如果组件 ID exclaim2能读组件 ID words 和 exclaim1产生的所有元组，你只需要如下简单定义如下：

1. builder.setBolt("exclaim2", new ExclamationBolt(), 5)
2. .shuffleGrouping("words")
3. .shuffleGrouping("exclaim1");

这个实现起来非常直接简便。

四、ExclamationBolt 实现

ExclamationBolt 就是在输入的元组后面加上三个感叹号“!!!“，完整的实现如下：

1. public static class ExclamationBolt extends BaseRichBolt {
2. OutputCollector \_collector;
4. @Override
5. public void prepare(Map conf, TopologyContext context, OutputCollector collector) {
6. \_collector = collector;
7. }
9. @Override
10. public void execute(Tuple tuple) {
11. \_collector.emit(tuple, new Values(tuple.getString(0) + "!!!"));
12. \_collector.ack(tuple);
13. }
15. @Override
16. public void declareOutputFields(OutputFieldsDeclarer declarer) {
17. declarer.declare(new Fields("word"));
18. }

21. }

ExclamationBolt 继承的是基类 BaseRichBolt

prepare 方法用 OutputCollector 类提供了 bolt，并且OutputCollector类就是用来从这个 Bolt 产生元组的，如上的

\_collector**.**emit**(**tuple**,** **new** Values**(**tuple**.**getString**(**0**)** **+** "!!!"**));**

\_collector**.**ack**(**tuple**);**

如上 execute 方法，new Value 就是产生新的值。 ack（）方法是据说 Storm 可靠性的象征，也就是打上了标签，不会被丢失。

declareOutputFields定义一个叫做”word”的字段的tuple。

五、本地模式运行 ExcamationTopogy

Storm 有2种运行模式，一种就是本地运行模式，一种就是分布式模式。

在本地模式中， storm用一个进程里面的线程来模拟所有的spout和bolt. 本地模式对开发和测试来说比较有用。

     在分布式模式下， storm由一堆机器组成。当你提交topology给master的时候， 你同时也把topology的必要的代码提交了。master负责分发你的代码并且负责给你的topolgoy分配工作进程。如果一个工作进程挂掉了， master节点会把认为重新分配到其它节点。

本地运行模式的代码：

1. Config conf = new Config();
2. conf.setDebug(true);
3. conf.setNumWorkers(2);
4. LocalCluster cluster = new LocalCluster();
5. cluster.submitTopology("test", conf, builder.createTopology());
6. Utils.sleep(10000);
7. cluster.killTopology("test");
8. cluster.shutdown();

首先， 这个代码定义通过定义一个LocalCluster对象来定义一个进程内的集群。提交topology给这个虚拟的集群和提交topology给分布式集群是一样的。通过调用submitTopology方法来提交topology， 它接受三个参数：要运行的topology的名字，一个配置对象以及要运行的topology本身。

topology的名字（此处就是 "test"）是用来唯一区别一个topology的，这样你然后可以用这个名字来杀死这个topology的。前面已经说过了， 你必须显式的杀掉一个topology， 否则它会一直运行。

Conf对象可以配置很多东西， 下面两个是最常见的：

* TOPOLOGY\_WORKERS(setNumWorkers) 定义你希望集群分配多少个工作进程给你来执行这个topology。 topology里面的每个组件会被需要线程来执行。每个组件到底用多少个线程是通过setBolt和setSpout来指定的。这些线程都运行在工作进程里面. 每一个工作进程包含一些节点的一些工作线程。比如， 如果你指定300个线程，60个进程， 那么每个工作进程里面要执行6个线程， 而这6个线程可能属于不同的组件(Spout, Bolt)。你可以通过调整每个组件的并行度以及这些线程所在的进程数量来调整topology的性能。
* TOPOLOGY\_DEBUG(setDebug), 当它被设置成true的话， storm会记录下每个组件所发射的每条消息。这在本地环境调试topology很有用， 但是在线上这么做的话会影响性能的。

sleep 就是等待时间了。

六、集群模式运行

1. if (args != null && args.length > 0) {
2. conf.setNumWorkers(3);
3. StormSubmitter.submitTopologyWithProgressBar(args[0], conf, builder.createTopology());

这个就是比较简单， 定义 workers 数量为3个，且需要输入指定的参数，就是 Topology 的名字。

基本通过以上的内容可以明白这个例子的大体构成和运行框架。