

1. 实时分析概念

a. 离线分析

通常是 需要一段时间的数据积累 积累到一定数量数据后 开始离线分析 无论数据量多大 离线分析 有开始 也有结束 最终得到一个处理的结果 这样的分析过程 得到的结果是有较大的延迟的

b. 实时分析

通常 数据不停的到来 随着数据的到来 来进行增量的运算 立即得到新数据的处理结果 并没有一个数据积累的过程 有开始 但是没有明确的结束的时刻 数据实时的进行运算 基本没有延迟

2. Storm概述

Storm是一个开源的分布式实时计算系统，可以简单、可靠的处理大量的数据流。

Storm有很多使用场景：如实时分析，在线机器学习，持续计算，分布式RPC，ETL等等。

Storm支持水平扩展，具有高容错性，保证每个消息都会得到处理。

Storm性能优良，处理速度很快(在一个小集群中，每个结点每秒可以处理数以百万计的消息)。

Storm的部署和运维都很便捷，而且更为重要的是可以使用任意编程语言来开发应用。

3. Storm组件

Storm将实时运算的过程 拆分为若干简单的步骤 再组装在一起完成复杂计算任务

由这些简单步骤组装起来的运算过程 称之为一个Topology(拓扑)

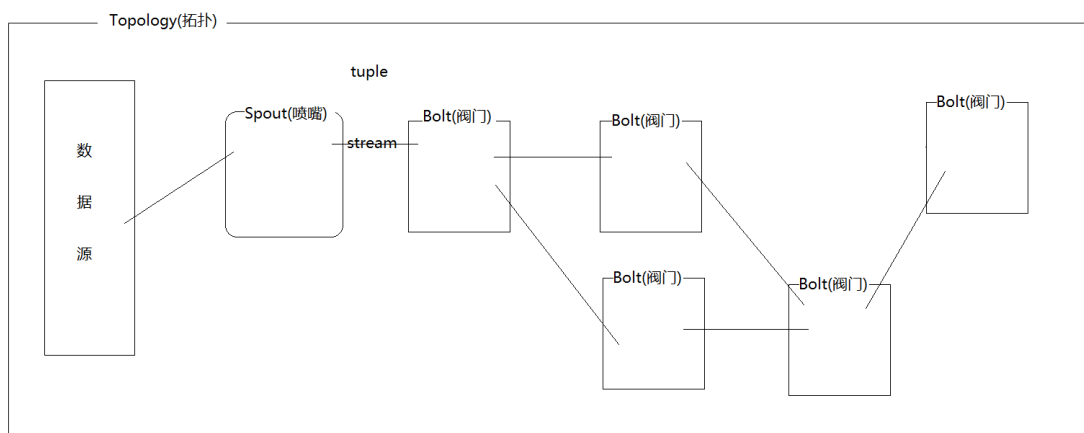
Topology由Spout(喷嘴) 和 Bolt(阀门)组成

Spout负责连接外部数据源 接收数据 并将数据 转换为Tuple格式 向后发送 可以发送给 一个或多个Bolt,Spout通常只负责接收 转换 发送数据,不会进行任何的业务处理

Bolt负责接收上游传入的Tuple数据,进行相应的运算,再将结果发送给后续的零个或多个Bolt

从而利用Spout和Bolt组建起复杂的数据流处理网络 实现实时处理

再整个Topology中传递的Tuple就组成了Stream(数据流)



Storm入门案例

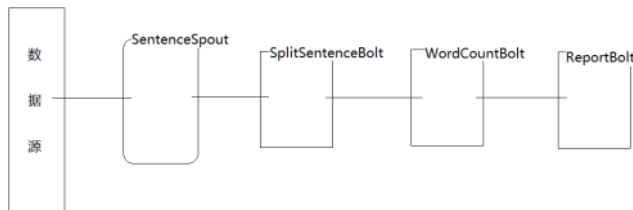
2018年9月13日 星期四 上午 9:53

1. Storm入门案例 - WordCount案例

a. 单机程序处理流程:

```
String [] sentences = ...
Map<String,Integer> map = ...
//1.得到每条语句
for(String sentence : sentences){
    //2.切分语句
    String [] words = sentence.split(" ");
    //3.统计数量
    for(String word : words){
        map.put(word,map.containsKey(word) ? map.get(word)+1 : 1);
    }
}
//4.打印结果
syso(map)
```

b. Topology设计:



2. SentenceSpout开发

void [cn.tedu.storm.wc.SentenceSpout.open](#)([Map](#) conf, [TopologyContext](#) context, [SpoutOutputCollector](#) collector)

@[Override](#)

Called when a task for this component is initialized within a worker on the cluster. It provides the spout with the environment in which the spout executes.

This includes the:

Specified by: [open\(...\)](#) in [ISpout](#)

Parameters:

conf The Storm configuration for this spout. This is the configuration provided to the topology merged in with cluster configuration on this machine.

context This object can be used to get information about this task's place within the topology, including the task id and component id of this task, input and output information, etc.

collector The collector is used to emit tuples from this spout. Tuples can be emitted at any time, including the open and close methods. The collector is thread-safe and should be saved as an instance variable of this spout object.

void [cn.tedu.storm.wc.SentenceSpout.nextTuple](#)()

@[Override](#)

When this method is called, Storm is requesting that the Spout emit tuples to the output collector. This method should be non-blocking, so if the Spout has no tuples to emit, this method should return. nextTuple, ack, and fail are all called in a tight loop in a single thread in the spout task. When there are no tuples to emit, it is courteous to have nextTuple sleep for a short amount of time (like a single

millisecond) so as not to waste too much CPU.

Specified by: [nextTuple\(\)](#) in [ISpout](#)



void [cn.tedu.storm.wc.SentenceSpout](#).declareOutputFields([OutputFieldsDeclarer](#) declarer)
[@Override](#)

Declare the output schema for all the streams of this topology.

Specified by: [declareOutputFields\(...\)](#) in [IComponent](#)

Parameters:

declarer this is used to declare output stream ids, output fields, and whether or not each output stream is a direct stream

```
1 package cn.tedu.storm.wc;
2
3
4 import java.util.Map;
5
6 import backtype.storm.spout.SpoutOutputCollector;
7 import backtype.storm.task.TopologyContext;
8 import backtype.storm.topology.OutputFieldsDeclarer;
9 import backtype.storm.topology.base.BaseRichSpout;
10 import backtype.storm.tuple.Fields;
11 import backtype.storm.tuple.Values;
12
13
14 public class SentenceSpout extends BaseRichSpout {
15
16     String sentences [] = {
17         "my name is park",
18         "i am so shuai",
19         "do you like me",
20         "are you sure you do not like me",
21         "ok i am sure"
22     };
23
24     private SpoutOutputCollector collector = null;
25
26     /**
27      * 初始化的方法
28      * 在当前组件被初始化时被调用 执行初始化相关操作
29      *
30      * conf: 当前Topology的配置信息
31      * context: 当前组件运行的上下文环境信息
32      * collector: 可以用来发送tuple,可以在任意位置发送tuple,此对象线程安全,通常保存到类内部作为成员变量,方便在其他方法中访问
33      */
34     @Override
35     public void open(Map conf, TopologyContext context, SpoutOutputCollector collector) {
36         this.collector = collector;
37     }
38
39     /**
40      * storm会不停的调用这个方法 要求发送tuple
41      * 此方法不应该被阻塞 所以如果没有任何tuple要发送 直接返回即可
42      * 在底层 其实就是一个单一的线程内循环不停的再调用此方法 所以如果真的没有任何tuple要发送 最好再这个方法中 睡眠一小段时间 以便于不至于浪费过多的cpu
43      */
44     private int index = 0;
45     @Override
46     public void nextTuple() {
47         if(index < sentences.length){
48             collector.emit(new Values(sentences[index]));
49             index++;
50         }else{
51             try {
52                 Thread.sleep(1);
53             } catch (InterruptedException e) {
54                 e.printStackTrace();
55             }
56             return;
57         }
58     }
59
60     /**
61      * 用来声明当前组件输出的tuple的基本信息
62      * declare:所有的组件如果想要发送tuple,都必须先声明再发送,此对象就是用来声明tuple的基本信息
63      */
64     @Override
65     public void declareOutputFields(OutputFieldsDeclarer declarer) {
66         declarer.declare(new Fields("sentence"));
67     }
68 }
```

3. SplitSentenceBolt开发



void [cn.tedu.storm.wc.SplitSentenceBolt](#).prepare([Map](#) stormConf, [TopologyContext](#) context,

[OutputCollector](#) collector)

@Override

Called when a task for this component is initialized within a worker on the cluster. It provides the bolt with the environment in which the bolt executes.

This includes the:

Specified by: [prepare\(...\)](#) in [IBolt](#)

Parameters:

stormConf The Storm configuration for this bolt. This is the configuration provided to the topology merged in with cluster configuration on this machine.

context This object can be used to get information about this task's place within the topology, including the task id and component id of this task, input and output information, etc.

collector The collector is used to emit tuples from this bolt. Tuples can be emitted at any time, including the prepare and cleanup methods. The collector is thread-safe and should be saved as an instance variable of this bolt object.



void [cn.tedu.storm.wc.SplitSentenceBolt.execute\(Tuple input\)](#)

@Override

Process a single tuple of input. The Tuple object contains metadata on it about which component/stream/task it came from. The values of the Tuple can be accessed using Tuple#getValue. The IBolt does not have to process the Tuple immediately. It is perfectly fine to hang onto a tuple and process it later (for instance, to do an aggregation or join).

Tuples should be emitted using the OutputCollector provided through the prepare method. It is required that all input tuples are acked or failed at some point using the OutputCollector. Otherwise, Storm will be unable to determine when tuples coming off the spouts have been completed.

For the common case of acking an input tuple at the end of the execute method, see IBasicBolt which automates this.

Specified by: [execute\(...\)](#) in [IBolt](#)

Parameters:

input The input tuple to be processed.



void [cn.tedu.storm.wc.SplitSentenceBolt.declareOutputFields\(OutputFieldsDeclarer declarer\)](#)

@Override

Declare the output schema for all the streams of this topology.

Specified by: [declareOutputFields\(...\)](#) in [IComponent](#)

Parameters:

declarer this is used to declare output stream ids, output fields, and whether or not each output stream is a direct stream

```
1 package cn.tedu.storm.wc;
2
3 import java.util.Map;
4
5
6 import backtype.storm.task.OutputCollector;
7 import backtype.storm.task.TopologyContext;
8 import backtype.storm.topology.OutputFieldsDeclarer;
9 import backtype.storm.topology.base.BaseRichBolt;
10 import backtype.storm.tuple.Fields;
11 import backtype.storm.tuple.Tuple;
12 import backtype.storm.tuple.Values;
13
14
```

```

15 public class SplitSentenceBolt extends BaseRichBolt{
16
17     private OutputCollector collector = null;
18
19     /**
20      * 初始化的方法
21      * 在当前组件被初始化时被调用 执行初始化相关操作
22      *
23      * conf: 当前Topology的配置信息
24      * context: 当前组件运行的上下文环境信息
25      * collector: 可以用来发送tuple,可以在任意位置发送tuple,此对象线程安全,通常保存到类内部作为成员变量,方便在其他方法中访问
26      */
27
28     @Override
29     public void prepare(Map stormConf, TopologyContext context, OutputCollector collector) {
30         this.collector = collector;
31     }
32
33     /**
34      * 处理上游发送的tuple,每个tuple都会进入一次此方法 进行处理
35      * 处理过后可以发送零个或多个tuple给后续的组件
36      * 如果需要,也可以不立即处理tuple,而是先持有tuple再在后续需要时处理
37      *
38      * input:当前传入的要处理的tuple
39      */
40
41     @Override
42     public void execute(Tuple input) {
43         String sentence = input.getStringByField("sentence");
44         String [] words = sentence.split(" ");
45         for (String word : words){
46             collector.emit(new Values(word));
47         }
48     }
49
50     /**
51      * 用来声明当前组件输出的tuple的基本信息
52      * declare:所有的组件如果想要发送tuple,都必须先声明再发送,此对象就是用来声明tuple的基本信息
53      */
54
55     @Override
56     public void declareOutputFields(OutputFieldsDeclarer declarer) {
57         declarer.declare(new Fields("word"));
58     }
59 }

```

4. WordCountBolt

```

1 package cn.tedu.storm.wc;
2
3 import java.util.HashMap;
4 import java.util.Map;
5
6
7 import backtype.storm.task.OutputCollector;
8 import backtype.storm.task.TopologyContext;
9 import backtype.storm.topology.OutputFieldsDeclarer;
10 import backtype.storm.topology.base.BaseRichBolt;
11 import backtype.storm.tuple.Fields;
12 import backtype.storm.tuple.Tuple;
13 import backtype.storm.tuple.Values;
14
15
16 public class WordCountBolt extends BaseRichBolt {
17     private OutputCollector collector = null;
18     @Override
19     public void prepare(Map stormConf, TopologyContext context, OutputCollector collector) {
20         this.collector = collector;
21     }
22
23
24     private Map<String,Integer> map = new HashMap<>();
25     @Override
26     public void execute(Tuple input) {
27         String word = input.getStringByField("word");
28         map.put(word, map.containsKey(word) ? map.get(word) + 1 : 1);
29         collector.emit(new Values(word,map.get(word)));
30     }
31
32
33     @Override
34     public void declareOutputFields(OutputFieldsDeclarer declarer) {
35         declarer.declare(new Fields("word","count"));
36     }
37 }

```

5. ReportBolt

```

1 package cn.tedu.storm.wc;
2
3 import java.util.Map;
4
5
6 import backtype.storm.task.OutputCollector;
7 import backtype.storm.task.TopologyContext;
8 import backtype.storm.topology.OutputFieldsDeclarer;
9 import backtype.storm.topology.base.BaseRichBolt;
10 import backtype.storm.tuple.Tuple;
11
12
13 public class ReportBolt extends BaseRichBolt {
14
15     private OutputCollector collector = null;
16

```

```

17
18         @Override
19         public void prepare(Map stormConf, TopologyContext context, OutputCollector collector) {
20             this.collector = collector;
21         }
22
23         @Override
24         public void execute(Tuple input) {
25             String word = input.getStringByField("word");
26             int count = input.getIntegerByField("count");
27             System.err.println("===单词数量发生变化: word[" + word + "], count[" + count + "]===");
28         }
29
30         @Override
31         public void declareOutputFields(OutputFieldsDeclarer declarer) {
32
33         }
34     }
35 }

```

6. WcTopology

```

1  package cn.tedu.storm.wc;
2
3
4  import backtype.storm.Config;
5  import backtype.storm.LocalCluster;
6  import backtype.storm.generated.StormTopology;
7  import backtype.storm.topology.TopologyBuilder;
8  import backtype.storm.tuple.Fields;
9
10 public class WordCountTopology {
11     public static void main(String[] args) throws Exception {
12         //1. 创建出所有组件
13         SentenceSpout spout = new SentenceSpout();
14         SplitSentenceBolt splitSentenceBolt = new SplitSentenceBolt();
15         WordCountBolt wordCountBolt = new WordCountBolt();
16         ReportBolt reportBolt = new ReportBolt();
17
18         //2. 创建拓扑构建者
19         TopologyBuilder builder = new TopologyBuilder();
20
21         //3. 向拓扑构建者告知 拓扑结构
22         builder.setSpout("Sentence_Spout", spout);
23         builder.setBolt("Split_Sentence_Bolt", splitSentenceBolt).shuffleGrouping("Sentence_Spout");
24         builder.setBolt("Word_Count_Bolt", wordCountBolt).fieldsGrouping("Split_Sentence_Bolt", new Fields("word"));
25         builder.setBolt("Report_Bolt", reportBolt).globalGrouping("Word_Count_Bolt");
26
27         //4. 利用拓扑构建者创建拓扑
28         StormTopology topology = builder.createTopology();
29
30         //5. 将拓扑提交到集群中运行
31         Config conf = new Config();
32         StormSubmitter.submitTopology("Wc_Topology", conf, topology);
33
34         //5. 在本地集群中模拟运行拓扑
35         LocalCluster cluster = new LocalCluster();
36         Config conf = new Config();
37         cluster.submitTopology("Wc_Topology", conf, topology);
38
39         Thread.sleep(10 * 1000);
40
41         cluster.killTopology("Wc_Topology");
42         cluster.shutdown();
43     }
44 }

```

Storm的并发机制

2018年9月13日 星期四 下午 2:07

1. Storm的并发级别

Storm可以在如下的四个级别上存在并发：

1) Node服务器

配置在Storm集群中的一个服务器，会执行Topology的一部分运算,一个Storm集群中包含一个或者多个Node

2) Worker进程

JVM虚拟机、进程。指一个Node上相互独立运作的JVM进程，每个Node可以配置运行一个或多个worker。一个Topology会分配到一个或者多个worker上运行。

3) Executor线程

指一个worker的jvm中运行的java线程。多个task可以指派给同一个executor来执行。除非是明确指定，Storm默认会给每个executor分配一个task。

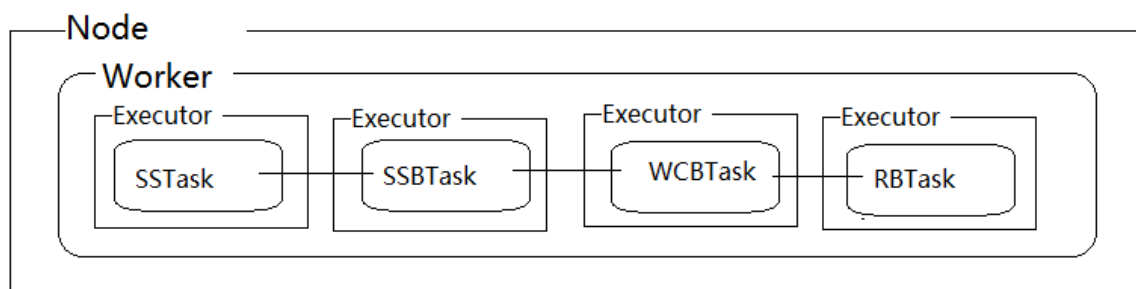
4) Task任务

bolt/spout实例。task是spout或bolt的实例，他们的nextTuple()和execute()方法会被executors线程调用执行。一个线程可以同时处理多个Task。

2. storm的并发控制

a. 默认并发控制

如果不进行任何配置，默认并发度为1，也即，默认情况下一个Node服务器上运行一个Worker进程，一个Worker进程里分配多个Executor线程，而每个Executor线程分别负责一个Task任务，此时唯一的并发处在线程级别。



b. 人为指定并发度

i. Node级别

Node级别的并发度无法通过代码控制，只能增加服务器硬件资源。

ii. Worker级别

可以通过代码为指定的Topology配置多个Worker

```
Config conf = new Config();
```

```
conf.setNumWorkers(2);
```

****单机测试模式下，设定Worker的数量没有意义，不会起效果。**

iii. Executor级别

可以通过代码指定Spout或Bolt需要几个线程

```
builder.setSpout(spout_id,spout,2)
```

```
builder.setBolt(bolt_id,bolt,2)
```

**** 如果只是指定线程数量，而未指定Task数量，则这个组件就会有指定数量的线程来处理，而默认每个线程内都执行一个该组件的Task**

iv. Task级别

可以通过代码指定Spout或Bolt需要并发几个Task

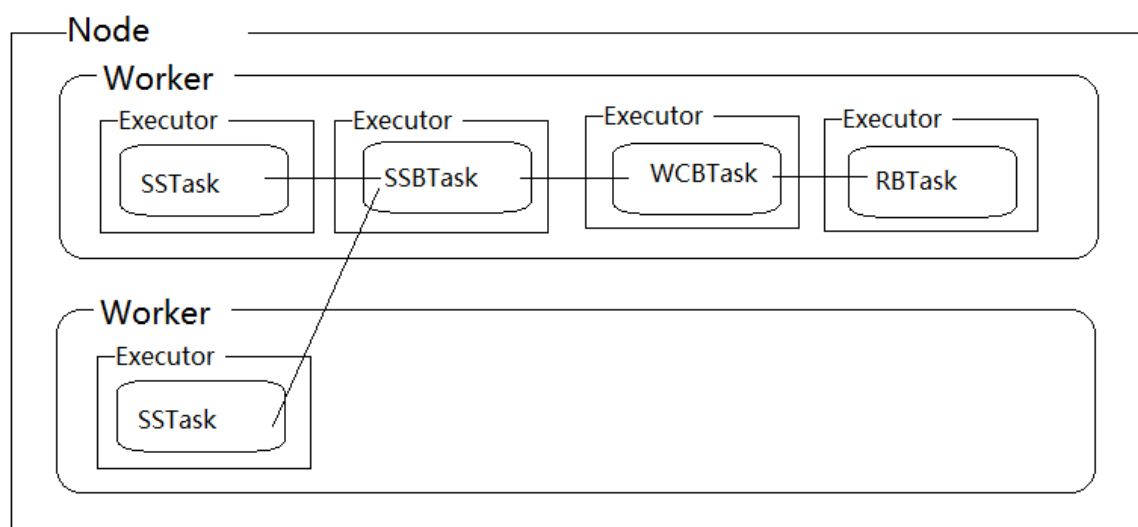
```
builder.setSpout(...).setNumTasks(2);
```

```
builder.setBolt(...).setNumTasks(2);
```

****指定的Task会在线程内部执行，如果指定的Task的数量多于线程的数量，则这些Task自动分配到这些线程内执行。**

案例:

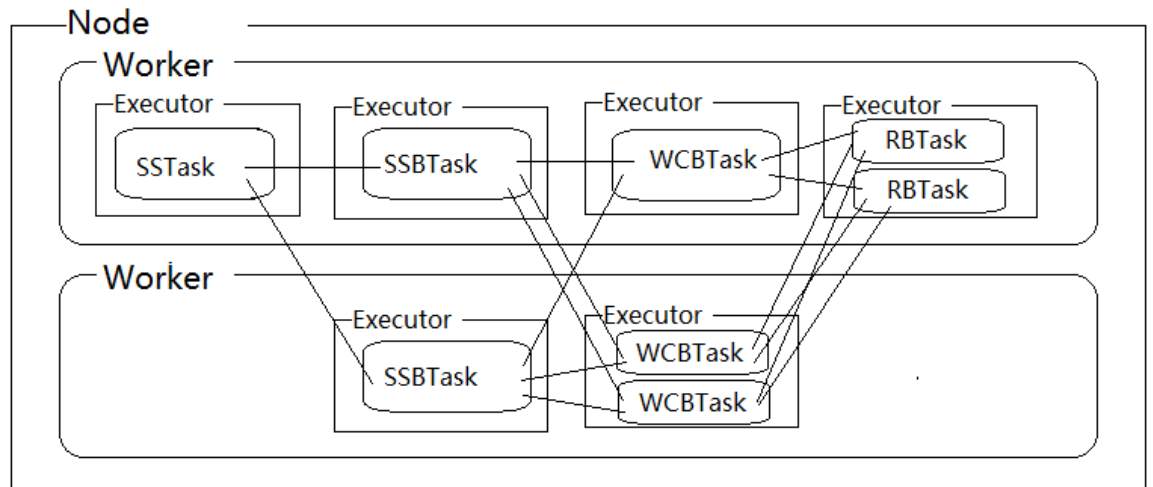
```
builder.setSpout("Sentence_Spout", spout,2);  
builder.setBolt("Split_Sentence_Bolt", splitSentenceBolt);  
builder.setBolt("Word_Count_Bolt", wordCountBolt);  
builder.setBolt("Report_Bolt", reportBolt);  
conf.setNumWorkers(2);
```




```

conf.setNumWorkers(2);
builder.setSpout("Sentence_Spout", spout);
builder.setBolt("Split_Sentence_Bolt", splitSentenceBolt, 2)
builder.setBolt("Word_Count_Bolt", wordCountBolt, 2).setNumTasks(3)
builder.setBolt("Report_Bolt", reportBolt).setNumTasks(2)

```



3. 数据流分组方式

在设定完并发度后，可以发现，组件向后发送tuple时，涉及到如何为后续组件的多个task分配这些tuple的问题，这是通过在连接组件时使用不同的方法，从而指定数据流分组方式来解决的。

storm内置了七种数据流分组方式：

i. Shuffle Grouping(随机分组)

随机分发数据流中的tuple给bolt中的各个task，每个task接收到的tuple数量相同。

ii. Fields Grouping(按字段分组)

根据指定字段的值进行分组。

指定字段具有相同值的tuple会路由到同一个bolt中的task中。

iii. All Grouping(全复制分组)

所有的tuple复制后分发给后续bolt的所有的task。

iv. Globle Grouping(全局分组)

这种分组方式将所有的tuple路由到唯一一个task上

Storm按照最小task id来选取接受数据的task

这种分组方式下配置bolt和task的并发度没有意义。

这种方式会导致所有tuple都发送到一个JVM实例上，可能会引起Strom集群中某个JVM或者服务器出现性能瓶颈或崩溃。

v. None Grouping(不分组)

在功能上和随机分组相同，为将来预留。

vi. Direct Grouping(指向型分组)

数据源会通过emitDirect()方法来判断一个tuple应该由哪个Strom组件来接受。

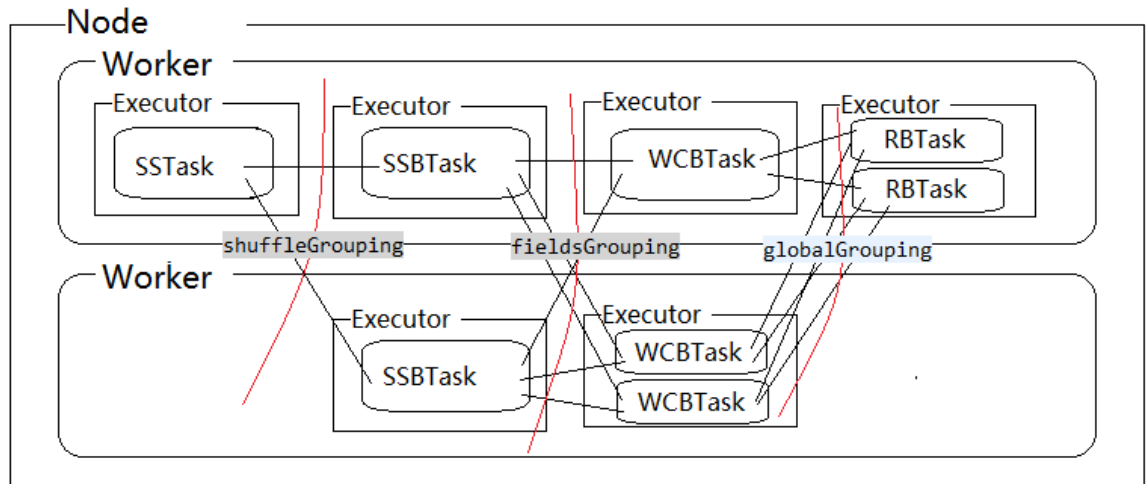
只能在声明为指向型的数据流上使用。

vii. Local or shuffle Grouping(本地或随机分组)

和随机分组类似，但是，会优先将tuple分发给同一个worker内的bolt的task，只有在同一个Worker内没有目标bolt的task的情况下采用随机分组方式。

这种方式可以减少网络传输，从而提高topology的性能。

案例：



Storm中的可靠性保证

2018年9月13日 星期四 下午 3:07

1. Storm可靠性概述

Storm是分布式程序，进程之间是通过网络来通信的，此时可能因为网络的不稳定、节点宕机等原因造成数据可能存在丢失的情况，这就给Storm的实时处理带来了数据丢失的可能性，如何保障Storm处理数据时能够不丢数据 不多数据，使处理的结果就有可信度，这就是Storm可靠性相关的问题。

Storm中的可靠性分为三个级别：

至多一次 - 不多数据，但是可能会丢数据

至少一次 - 不丢数据，但是可能会多数据

恰好一次 - 不丢也不多，恰好完整的处理

这三种级别，越往下可靠性越高，但效率越低，越往上可靠性越低，但是效率越高，在真正的开发中应该根据业务需求，选择一个在满足的要求的可靠性的前提下，性能尽量好的可靠性级别。

2. 至多一次

之前wc案例中，没有任何并发控制，此时就是 至多一次的并发状态，不会多数据，但是可能会因为网络延迟 程序出错 造成数据的丢失。

3. 至少一次

在分布式的环境下，因为网络的不稳定性，数据丢失是无法彻底避免的，所以为了保证数据不丢失，应该检测数据丢失的情况，并重发丢失的数据。

这种方式下可以保证不丢数据，但是可能因为重发造成多数据。

a. 对于Spout：

- i. 持有已经发送的数据一段时间，以便于在数据丢失的情况下能够重发，直到该数据完成了整个Storm处理流程，才可以删除。
- ii. 覆盖ack方法 和fail方法，storm会在某条数据完成整个处理环节过后，来调用ack方法，此时可以在ack方法中删除之前缓存的数据。而在某条数据处理失败时，storm会自动调用fail方法，此时可以在fail方法中实现重发的过程。



void [cn.tedu.storm.wc.SentenceSpout.ack\(Object msgId\)](#)

@Override

Storm has determined that the tuple emitted by this spout with the msgId identifier has been fully processed. Typically, an implementation of this method will take that message off the queue and prevent it from being replayed.

Overrides: [ack\(...\)](#) in [BaseRichSpout](#)

Parameters:

msgId



void [cn.tedu.storm.wc.SentenceSpout.fail\(Object msgId\)](#)

@Override

The tuple emitted by this spout with the msgId identifier has failed to be fully processed. Typically, an implementation of this method will put that message back on the queue to be replayed at a later time.

Overrides: [fail\(...\)](#) in [BaseRichSpout](#)

Parameters:

msgId

b. 对于Bolt：

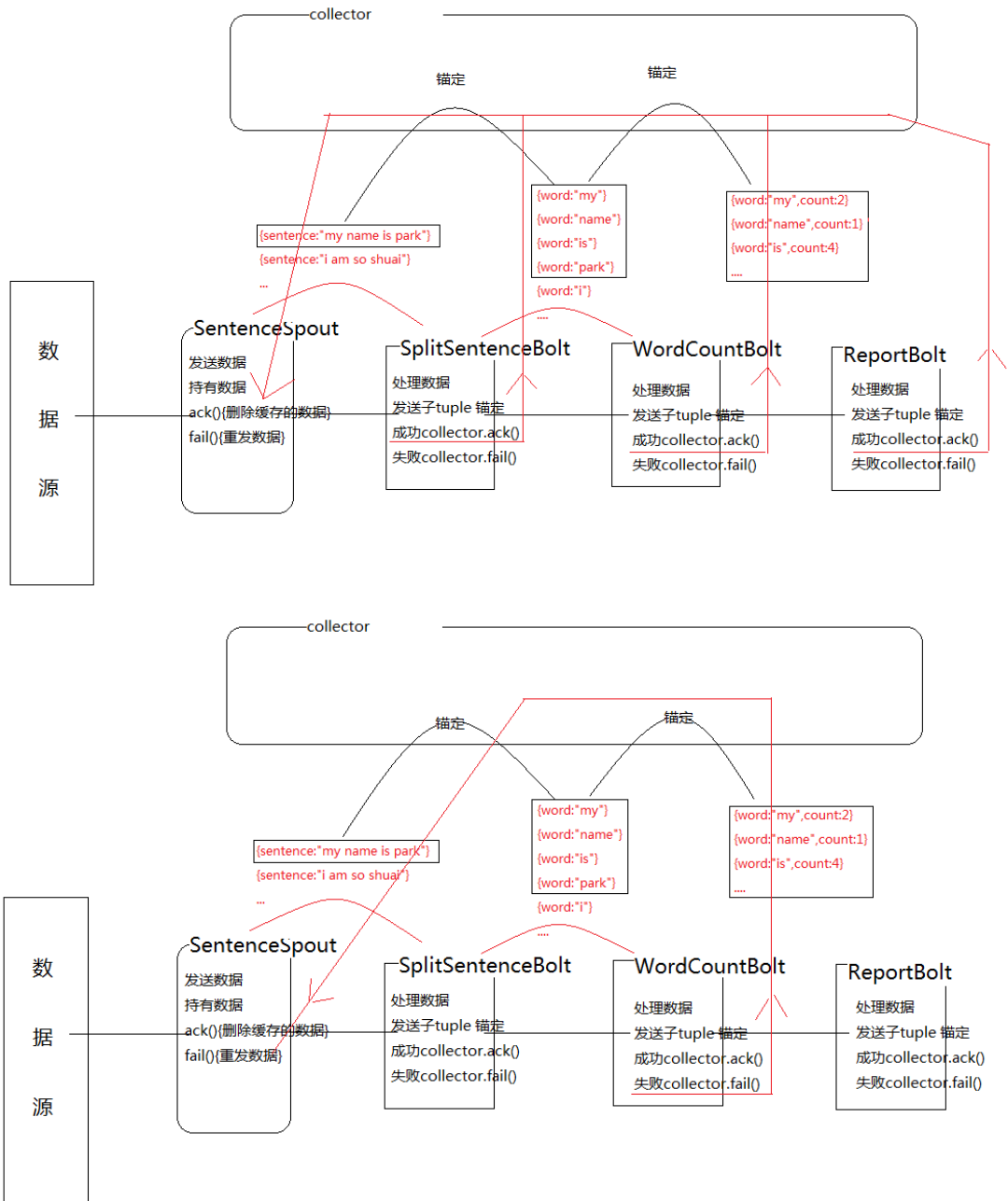
i. 需要在发送tuple的过程中实现锚定。

所谓的锚定就是将父tuple和子tuple的派生关系记录在collector中，以便于在后续tuple出错时，可以通过collector中维系的锚定关系，来回溯找到最初的tuple来实现重复

ii. 在整个bolt处理完一个tuple后调用collector提供的ack方法，或，在某个tuple处理过程中出错要调用collector的fail方法。

从而可以向collector告知，某个tuple在某个bolt中的处理状态，实现对tuple的处理的监控。

当对于某一个spout发送的tuple，整个拓扑中的所有bolt都处理成功，调用的是ack，则collector认为该tuple完成了整个处理流程会自动调用spout的ack方法而任何一个bolt处理失败，调用的是fail方法，则collector认为该tuple处理失败，会自动调用spout的fail方法。



案例：参考今日代码

4. 恰好一次 //TODO