1. **Storm 简介(来由，发展，应用场景)**
2. **Storm 体系结构**
3. **Storm topology supovisor worker**

**4. Storm 安装部署**

**5. Storm 常用命令**

**6. Storm 编程接口**

**7. Storm 运行机制**

**流式计算的历史**

早在7、8年前诸如UC伯克利、斯坦福等大学就开始了对流式数据处理的研究，但是由于更多的关注于金融行业的业务场景或者互联网流量监控的业务场景，以及当时互联网数据场景的限制，造成了研究多是基于对传统数据库处理的流式化，对流式框架本身的研究偏少。目前这样的研究逐渐没有了声音，工业界更多的精力转向了实时数据库。

　　2010年Yahoo！对S4的开源，2011年twitter对Storm的开源，改变了这个情况。以前互联网的开发人员在做一个实时应用的时候，除了要关注应用逻辑计算处理本身，还要为了数据的实时流转、交互、分布大伤脑筋。但是现在情况却大为不同，以Storm为例，开发人员可以快速的搭建一套健壮、易用的实时流处理框架，配合SQL产品或者NoSQL产品或者MapReduce计算平台，就可以低成本的做出很多以前很难想象的实时产品：比如一淘数据部的量子恒道品牌旗下的多个产品就是构建在实时流处理平台上的。

**流式计算的最新进展**

在数据处理时间和方式上，Storm与Hadoop MapReduce基本上是两个对立面，而这两个技术具备整合可能性极大程度该归结于 YARN这个集群管理层。Hortonworks当下正在致力于通过新型处理框架Tez 来提高Hive的速度，同时YARN还允许Hadoop用户运行Spark内存处理框架。同时，微软也在使用YARN让Hadoop更加适合机器学习用例。

　　此外，通过YARN，同集群上同时运行HBase、 Giraph等不同技术也成为可能。此外，集群管理技术Mesos(加州大学伯克利分校出品，现已成为Apache项目) 同样支持了类似YARN功能，尽管其不是像YARN这样与HDFS捆绑。

　　更多技术的整合预示Hadoop这个大数据处理平台绝不是昙花一现，同时也会让Hadoop在大数据应用程序领域获得更高的统治力。

***Storm的特点***

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。被称作“实时的hadoop”。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，而且处理速度很快（在一个小集群中，每个节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。

编程模型简单

　　在大数据处理方面相信大家对hadoop已经耳熟能详，基于Google Map/Reduce来实现的Hadoop为开发者提供了map、reduce原语，使并行批处理程序变得非常地简单和优美。

同样，Storm也为大数据 的实时计算提供了一些简单优美的原语，这大大降低了开发并行实时处理任务的复杂性，帮助你快速、高效的开发应用。

可扩展

　　在Storm集群中真正运行topology的主要有三个实体：工作进程、线程和任务。Storm集群中的每台机器上都可以运行多个工作进程，每个工作进程又可创建多个线程，每个线程可以执行多个任务，任务是真正进行数据处理的实体，我们开发的spout、bolt就是作为一个或者多个任务的方式执行的。

因此，计算任务在多个线程、进程和服务器之间并行进行，支持灵活的水平扩展。

高可靠性

　　Storm可以保证spout发出的每条消息都能被“完全处理”，这也是直接区别于其他实时系统的地方，如S4。

　　spout发出的消息后续可能会触发产生成千上万条消息，可以形象的理解为一棵消息树，其中spout发出的消息为树根，Storm会跟踪这棵消息树的处理情况，只有当这棵消息树中的所有消息都被处理了，Storm才会认为spout发出的这个消息已经被“完全处理”。如果这棵消息树中的任何一个消息处理失败了，或者整棵消息树在限定的时间内没有“完全处理”，那么spout发出的消息就会重发。

高容错性

　　如果在消息处理过程中出了一些异常，Storm会重新安排这个出问题的处理单元。Storm保证一个处理单元永远运行（除非你显式杀掉这个处理单元）。

　　当然，如果处理单元中存储了中间状态，那么当处理单元重新被Storm启动的时候，需要应用自己处理中间状态的恢复。

Storm与hadoop

Storm集群和Hadoop集群表面上看很类似。Hadoop上运行的是MapReduce jobs，而在Storm上运行的是拓扑（topology）；

Hadoop擅长于分布式离线批处理，而Storm设计为支持分布式实时计算；

Hadoop新的spark组件提供了在hadoop平台上运行storm的可能性

***Storm的基本概念***

在深入理解Storm之前，需要了解一些概念：

Topologies ： 拓扑，也俗称一个任务

Spouts ： 拓扑的消息源

Bolts ： 拓扑的处理逻辑单元

tuple：消息元组

Streams ： 流

Stream groupings ：流的分组策略

Tasks ： 任务处理单元

Executor :工作线程

Workers ：工作进程

Configuration ： topology的配置

***Storm与Hadoop的对比***

Topology 与 Mapreduce

一个关键的区别是： 一个MapReduce job最终会结束， 而一个topology永远会运行（除非你手动kill掉）

Nimbus 与 ResoureManager

在Storm的集群里面有两种节点： 控制节点（master node）和工作节点（worker node）。控制节点上面运行一个叫Nimbus后台程序，它的作用类似Hadoop里面的JobTracker。Nimbus负责在集群里面分发代码，分配计算任务给机器， 并且监控状态。

Supervisor (worker进程)与NodeManager(YarnChild)

每一个工作节点上面运行一个叫做Supervisor的节点。Supervisor会监听分配给它那台机器的工作，根据需要启动/关闭工作进程。每一个工作进程执行一个topology的一个子集；一个运行的topology由运行在很多机器上的很多工作进程组成。

***Storm 体系架构***

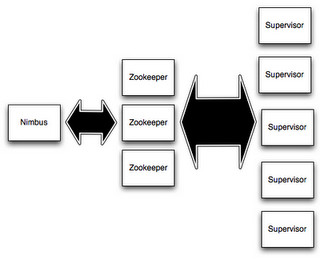
***Storm中的Nimbus和Supervisor***

Nimbus和Supervisor之间的所有协调工作都是通过Zookeeper集群完成。

Nimbus进程和Supervisor进程都是快速失败（fail-fast)和无状态的。所有的状态要么在zookeeper里面， 要么在本地磁盘上。

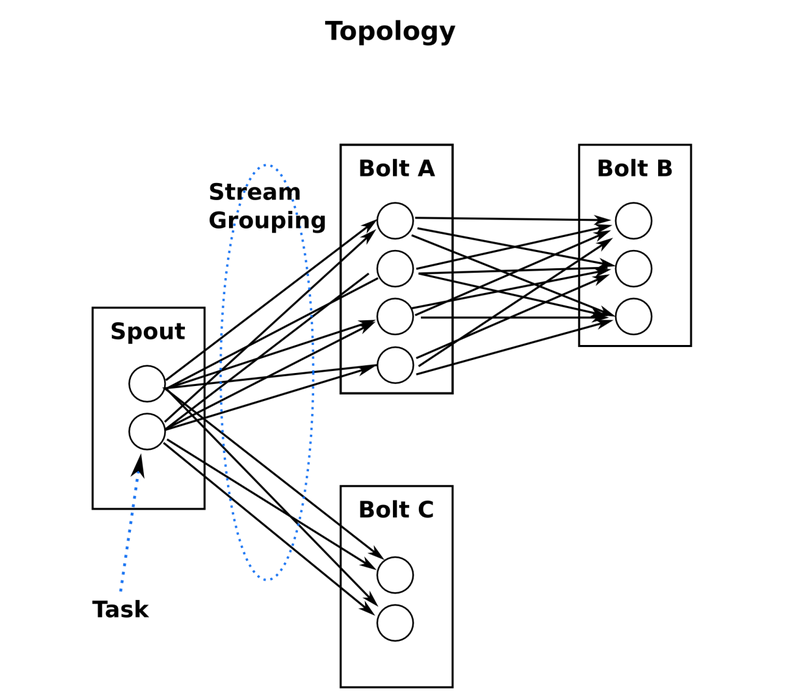
这也就意味着你可以用kill -9来杀死Nimbus和Supervisor进程， 然后再重启它们，就好像什么都没有发生过。这个设计使得Storm异常的稳定。

***Storm中的Nimbus和Supervisor***



***Storm中的Topologies***

一个topology是spouts和bolts组成的图， 通过stream groupings将图中的spouts和bolts连接起来，如下图：



***Storm中的Stream***

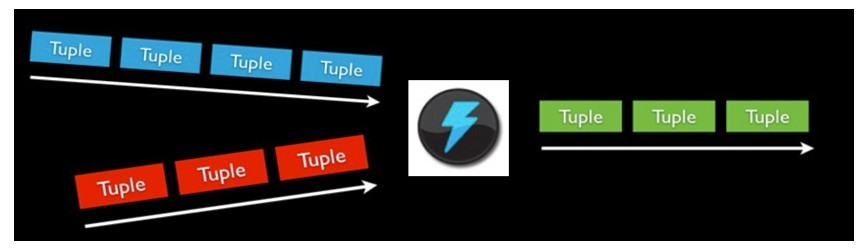
消息流stream是storm里的关键抽象；

一个消息流是一个没有边界的tuple序列， 而这些tuple序列会以一种分布式的方式并行地创建和处理；

通过对stream中tuple序列中每个字段命名来定义stream；

在默认的情况下，tuple的字段类型可以是：integer，long，short， byte，string，double，float，boolean和byte array；

可以自定义类型（只要实现相应的序列化器）。



***Storm中的Spouts***

消息源spout是Storm里面一个topology里面的消息生产者；

一般来说消息源会从一个外部源读取数据并且向topology里面发出消息：tuple；

Spouts可以是可靠的也可以是不可靠的：如果这个tuple没有被storm成功处理，可靠的消息源spouts可以重新发射一个tuple， 但是不可靠的消息源spouts一旦发出一个tuple就不能重发了；

消息源可以发射多条消息流stream：

使用OutputFieldsDeclarer.declareStream来定义多个stream，

然后使用SpoutOutputCollector来发射指定的stream。

***Storm中的Bolts***

所有的消息处理逻辑被封装在bolts里面；

Bolts可以做很多事情：过滤，聚合，查询数据库等等。

Bolts可以简单的做消息流的传递，也可以通过多级Bolts的组合来完成复杂的消息流处理；比如求TopN、聚合操作等（如果要把这个过程做得更具有扩展性那么可能需要更多的步骤）。

Bolts可以发射多条消息流：

使用OutputFieldsDeclarer.declareStream定义stream；

使用OutputCollector.emit来选择要发射的stream；

Bolts的主要方法是execute,：

它以一个tuple作为输入，使用OutputCollector来发射tuple；

通过调用OutputCollector的ack方法，以通知这个tuple的发射者spout；

Bolts一般的流程：

处理一个输入tuple,  发射0个或者多个tuple, 然后调用ack通知storm自己已经处理过这个tuple了；

storm提供了一个IBasicBolt会自动调用ack。

***Storm中的Stream groupings***

定义一个topology的关键一步是定义每个bolt接收什么样的流作为输入；

stream grouping就是用来定义一个stream应该如何分配数据给bolts；

Storm里面有7种类型的stream grouping：

Shuffle Grouping——随机分组， 随机派发stream里面的tuple，保证每个bolt接收到的tuple数目大致相同；

Fields Grouping——按字段分组， 比如按userid来分组， 具有同样userid的tuple会被分到相同的Bolts里的一个task， 而不同的userid则会被分配到不同的bolts里的task；

All Grouping——广播发送，对于每一个tuple，所有的bolts都会收到；

 Global Grouping——全局分组， 这个tuple被分配到storm中的一个bolt的其中一个task。再具体一点就是分配给id值最低的那个task；

Non Grouping——不分组，这个分组的意思是说stream不关心到底谁会收到它的tuple。目前这种分组和Shuffle grouping是一样的效果， 有一点不同的是storm会把这个bolt放到这个bolt的订阅者同一个线程里面去执行；

***Storm中的Workers***

一个topology可能会在一个或者多个worker（工作进程）里面执行；

每个worker是一个物理JVM并且执行整个topology的一部分；

比如，对于并行度是300的topology来说，如果我们使用50个工作进程来执行，那么每个工作进程会处理其中的6个tasks；

Storm会尽量均匀的工作分配给所有的worker；

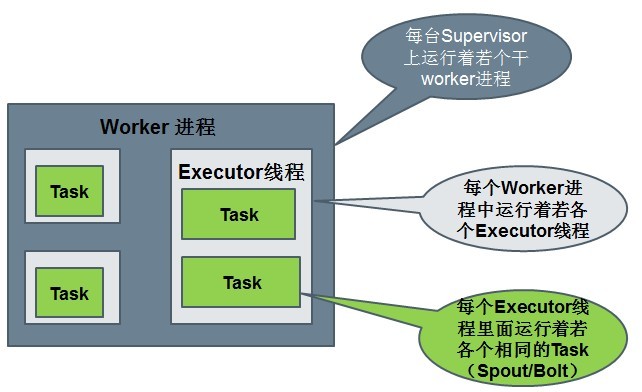
***Storm中的Tasks***

每一个spout和bolt会被当作很多task在整个集群里执行

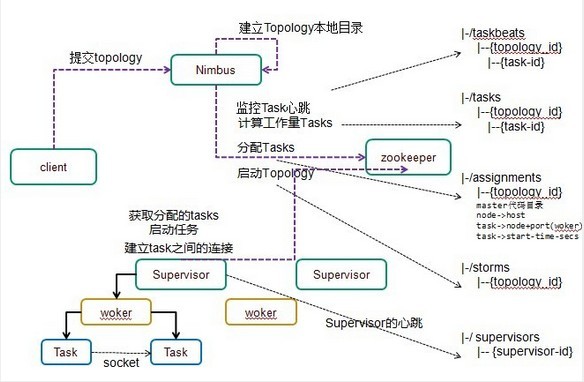
每一个executor对应到一个线程，在这个线程上运行多个task

stream grouping则是定义怎么从一堆task发射tuple到另外一堆task

可以调用TopologyBuilder类的setSpout和setBolt来设置并行度（也就是有多少个task）



***Storm 体系架构***



***Storm*** 安装部署

　部署Storm集群需要依次完成的安装步骤：

1.安装jdk6及以上版本;

　　2. 搭建Zookeeper集群；

　　3. 安装Storm依赖库；

　　4. 下载并解压Storm发布版本；

　　5. 修改storm.yaml配置文件；

　　6. 启动Storm各个后台进程。

***Storm包的安装配置***

下一步，需要在Nimbus和Supervisor机器上安装Storm发行版本。

1. 下载Storm发行版本，推荐使用Storm0.9.2：

wget https://github.com/downloads/nathanmarz/storm/storm-0.9.2.zip

2. 解压到安装目录下：

unzip storm-0.9.2.zip

修改storm.yaml配置文件

Storm发行版本解压目录下有一个conf/storm.yaml文件，用于配置Storm。默认配置在这里可以查看。

conf/storm.yaml中的配置选项将覆盖defaults.yaml中的默认配置。

1) storm.zookeeper.servers: Storm集群使用的Zookeeper集群地址，其格式如下：

storm.zookeeper.servers:

- “111.222.333.444″

- “555.666.777.888″

如果Zookeeper集群使用的不是默认端口，那么还需要storm.zookeeper.port选项。

2) storm.local.dir: Nimbus和Supervisor进程用于存储少量状态，如jars、confs等的本地磁盘目录，需要提前创建该目录并给以足够的访问权限。然后在storm.yaml中配置该目录，如：

storm.local.dir: "/home/admin/storm/workdir"

**注意事项：**

启动Storm后台进程时，需要对conf/storm.yaml配置文件中设置的storm.local.dir目录具有写权限。

storm后台进程被启动后，将在Storm安装部署目录下的logs/子目录下生成各个进程的日志文件。

经测试，Storm UI必须和Storm Nimbus部署在同一台机器上，否则UI无法正常工作，因为UI进程会检查本机是否存在Nimbus链接。

为了方便使用，可以将bin/storm加入到系统环境变量中。

至此，Storm集群已经部署、配置完毕，可以向集群提交拓扑运行了。

***Storm 常用命令***

1、启动nimbus后台程序  
命令格式：storm nimbus

2、启动supervisor后台程序  
命令格式：storm supervisor

3、启动ui服务  
命令格式：storm ui

4、提交Topologies

命令格式：storm jar 【jar路径】 【拓扑包名.拓扑类名】【stormIP地址】【storm端口】【拓扑名称】【参数】  
eg：  
storm jar /home/storm/storm-starter.jar storm.starter.WordCountTopology wordcountTop;

#提交storm-starter.jar到远程集群，并启动wordcountTop拓扑。

5、停止Topologies

查看当前运行的topo： storm list  
命令格式：storm kill 【拓扑名称】  
样例：storm kill wordcountTop  
#杀掉wordcountTop拓扑。

***Storm*** 相关配置项

***在storm.yaml中常用的几个选项***

storm.zookeeper.root

Storm在zookeeper集群中的根目录，默认是“/”

topology.workers

每个Topology运行时的worker的默认数目，若在代码中设置，则此选项值被覆盖

storm.zookeeper.servers

zookeeper集群的节点列表

storm.local.dir

Storm用于存储jar包和临时文件的本地存储目录

ui.port

Storm集群的UI地址端口号，默认是8080

nimbus.host:

Nimbus节点的host

supervisor.slots.ports

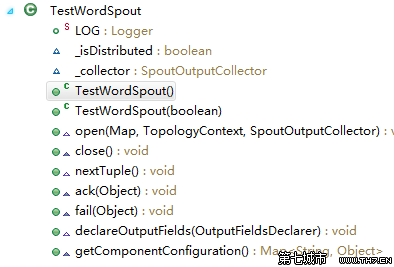
Supervisor节点的worker占位槽，集群中的所有Topology公用这些槽位数，即使提交时设置了较大数值的槽位数，系统也会按照当前集群中实际剩余的槽位数来进行分配，当所有的槽位数都分配完时，新提交的Topology只能等待，系统会一直监测是否有空余的槽位空出来，如果有，就再次给新提交的Topology分配

***Storm编程接口***

***Spouts***

Spout是Stream的消息产生源， Spout组件的实现可以通过继承BaseRichSpout类或者其他\*Spout类来完成，也可以通过实现IRichSpout接口来实现。

    需要根据情况实现Spout类中重要的几个方法有：



***open方法***

当一个Task被初始化的时候会调用此open方法。一般都会在此方法中对发送Tuple的对象SpoutOutputCollector和配置对象TopologyContext初始化。

示例如下：

1 public void open(Map conf, TopologyContext context, SpoutOutputCollector collector) {

this.\_collector = collector;

}

***getComponentConfiguration方法***

此方法用于声明针对当前组件的特殊的Configuration配置。

示例如下：

public Map<String, Object> getComponentConfiguration() {

if(!\_isDistributed) {

Map<String, Object> ret = new HashMap<String, Object>();

ret.put(Config.TOPOLOGY\_MAX\_TASK\_PARALLELISM, 3);

return ret;10 11

} else {

return null;

}

}

这里便是设置了Topology中当前Component的线程数量上限。

***nextTuple方法***

这是Spout类中最重要的一个方法。发射一个Tuple到Topology都是通过这个方法来实现的。

示例如下：

public void nextTuple() {

Utils.sleep(100);

final String[] words = new String[] {"twitter","facebook","google"};

final Random rand = new Random();

final String word = words[rand.nextInt(words.length)];

\_collector.emit(new Values(word));

}

    这里便是从一个数组中随机选取一个单词作为Tuple，然后通过\_collector发送到Topology。

**declareOutputFields方法**

此方法用于声明当前Spout的Tuple发送流。

Stream流的定义是通过OutputFieldsDeclare.declare方法完成的，其中的参数包括了发送的域Fields。

示例如下：

public void declareOutputFields(OutputFieldsDeclarer declarer) {

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

}

另外，除了上述几个方法之外，还有ack、fail和close方法等;

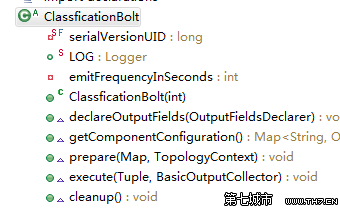
Storm在监测到一个Tuple被成功处理之后会调用ack方法，处理失败会调用fail方法;

这两个方法在BaseRichSpout等类中已经被隐式的实现了。

***Bolts***

Bolt类接收由Spout或者其他上游Bolt类发来的Tuple，对其进行处理。Bolt组件的实现可以通过继承BasicRichBolt类或者IRichBolt接口来完成。

    Bolt类需要实现的主要方法有：



***prepare方法***

此方法和Spout中的open方法类似，为Bolt提供了OutputCollector，用来从Bolt中发送Tuple。Bolt中Tuple的发送可以在prepare方法中、execute方法中、cleanup等方法中进行，一般都是些在execute中。

    示例如下：

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

this. \_collector = collector;

}

**getComponentConfiguration方法**

和Spout类一样，在Bolt中也可以有getComponentConfiguration方法。

    示例如下：

public Map<String, Object> getComponentConfiguration() {

Map<String, Object> conf = new HashMap<String, Object>();

conf.put(Config.TOPOLOGY\_TICK\_TUPLE\_FREQ\_SECS,

emitFrequencyInSeconds);

return conf;

}

    此例定义了从系统组件“\_system”的“\_tick”流中发送Tuple到当前Bolt的频率，当系统需要每隔一段时间执行特定的处理时，就可以利用这个系统的组件的特性来完成。

**declareOutputFields方法**

用于声明当前Bolt发送的Tuple中包含的字段，和Spout中类似。

    示例如下：

public void declareOutputFields(OutputFieldsDeclarer declarer) {

declarer.declare(new Fields("obj", "count", "actualWindowLengthInSeconds"));

}

    此例说明当前Bolt类发送的Tuple包含了三个字段："obj", "count", "actualWindowLengthInSeconds"。

**topology**



**Topology运行机制**

(1)Storm提交后，会把代码首先存放到Nimbus节点的inbox目录下，之后，会把当前Storm运行的配置生成一个stormconf.ser文件放到Nimbus节点的stormdist目录中，在此目录中同时还有序列化之后的Topology代码文件；

(2)在设定Topology所关联的Spouts和Bolts时，可以同时设置当前Spout和Bolt的executor数目和task数目，默认情况下，一个Topology的task的总和是和executor的总和一致的。之后，系统根据worker的数目，尽量平均的分配这些task的执行。worker在哪个supervisor节点上运行是由storm本身决定的；

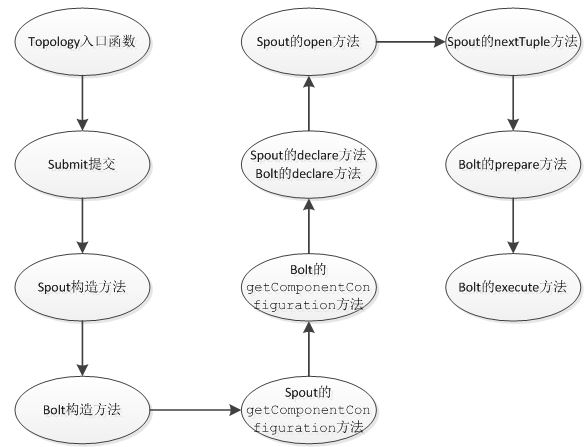
(3)任务分配好之后，Nimbes节点会将任务的信息提交到zookeeper集群，同时在zookeeper集群中会有workerbeats节点，这里存储了当前Topology的所有worker进程的心跳信息；

(4)Supervisor节点会不断的轮询zookeeper集群，在zookeeper的assignments节点中保存了所有Topology的任务分配信息、代码存储目录、任务之间的关联关系等，Supervisor通过轮询此节点的内容，来领取自己的任务，启动worker进程运行；

(5)一个Topology运行之后，就会不断的通过Spouts来发送Stream流，通过Bolts来不断的处理接收到的Stream流，Stream流是无界的。

最后一步会不间断的执行，除非手动结束Topology。

Topology中的Stream处理时的方法调用过程如下：c



有几点需要说明的地方：

   (1)每个组件(Spout或者Bolt)的构造方法和declareOutputFields方法都只被调用一次。

   (2)open方法、prepare方法的调用是多次的。入口函数中设定的setSpout或者setBolt里的并行度参数指的是executor的数目，是负责运行组件中的task的线程的数目，此数目是多少，上述的两个方法就会被调用多少次，在每个executor运行的时候调用一次。相当于一个线程的构造方法。

   (3)nextTuple方法、execute方法是一直被运行的，nextTuple方法不断的发射Tuple，Bolt的execute不断的接收Tuple进行处理。只有这样不断地运行，才会产生无界的Tuple流，体现实时性。相当于线程的run方法。

   (4)在提交了一个topology之后，Storm就会创建spout/bolt实例并进行序列化。之后，将序列化的component发送给所有的任务所在的机器(即Supervisor节点)，在每一个任务上反序列化component。

   (5)Spout和Bolt之间、Bolt和Bolt之间的通信，是通过zeroMQ的消息队列实现的。

   (6)上图没有列出ack方法和fail方法，在一个Tuple被成功处理之后，需要调用ack方法来标记成功，否则调用fail方法标记失败，重新处理这个Tuple。