storm 20:30开始

好消息！！！

大数据线上班(hadoop实战班)随到随学！火热报名中！！！

-- 3980，随堂视频附送，终身免费重学

-- 老师一对一辅导！电脑远程协助解决问题！

大数据线下班将于4月28日开班！火热报名中！！！

-- 老师面授课程！传统式教室教学已开班多期！学习完美就业！

大数据周末班将于5月7日再次开班！火热报名中！！！

贾老师：1786418286

何老师：1926106490

詹老师：2805048645

讨论技术可以加入以下QQ群： 172599077 , 156927834

讲师：君临天下

- 课程大纲
  - Storm介绍
  - Storm核心架构
  - Storm java api

- storm是一个分布式的、容错的实时计算系统，它被托管在GitHub上，遵循 Eclipse Public License 1.0。Storm是由BackType开发的实时处理系统，由Twitter开源
- 官网http://storm.apache.org/
- 国内外各大网站使用，例如雅虎、阿里、百度

- Storm实时低延迟，主要有两个原因：
  - storm进程是常驻内存的，不像hadoop里面是不断的启停的，就没有不断启停的开销。
  - 第二点，Storm的数据是不经过磁盘的，都是在内存里面，处理完就没有了，处理完就没有了，数据的交换经过网络，这样就避免磁盘IO的开销，所以Storm可以很低的延迟。
- 在2013年的时候，Storm进入Apache社区进行孵化，最终进入了Apache顶级项目

- Storm和hadoop的区别

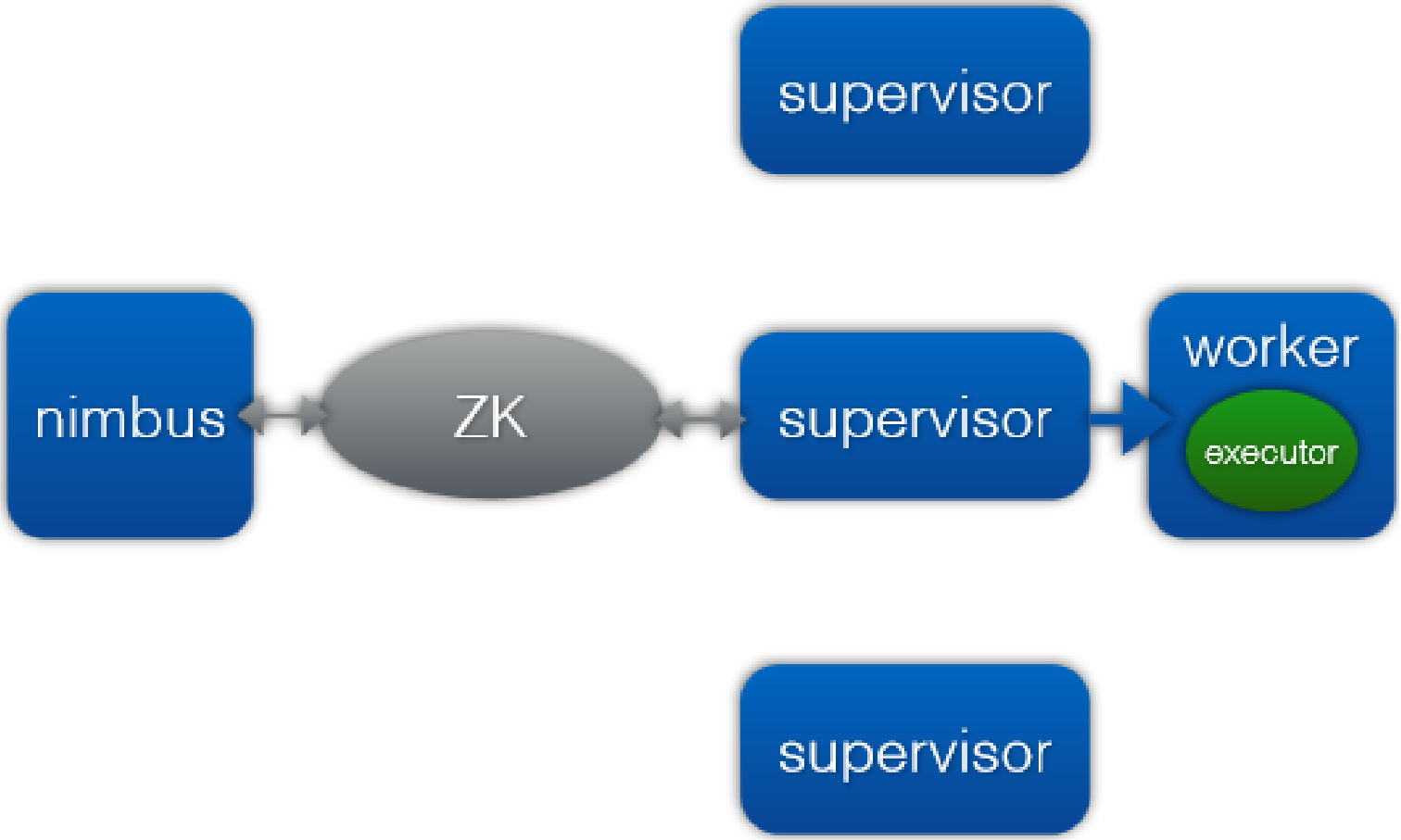
Hadoop

Storm

- Storm和hadoop的区别
  - 数据来源：HADOOP是HDFS上某个文件夹下的可能是成TB的数据，STORM是实时新增的某一笔数据
  - 处理过程：HADOOP是分MAP阶段到REDUCE阶段，STORM是由用户定义处理流程，流程中可以包含多个步骤，每个步骤可以是数据源(SPOUT)或处理逻辑(BOLT)
  - 是否结束：HADOOP最后是要结束的，STORM是没有结束状态，到最后一步时，就停在那，直到有新
    数据进入时再从头开始
  - 处理速度：HADOOP是以处理HDFS上大量数据为目的，速度慢，STORM是只要处理新增的某一笔数据即可
    可以做到很快。
  - 适用场景：HADOOP是在要处理一批数据时用的，不讲究时效性，要处理就提交一个JOB，STORM是要处理
    某一新增数据时用的，要讲时效性
  - 与MQ对比：HADOOP没有对比性，STORM可以看作是有N个步骤，每个步骤处理完就向下一个MQ发送消息，
    监听这个MQ的消费者继续处理

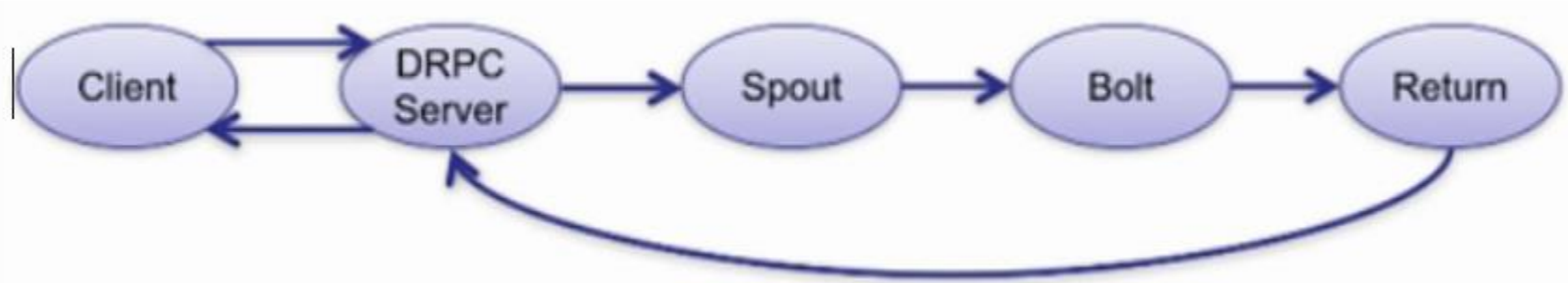
- 架构：
  - Nimbus
  - Supervisor
  - Worker
- 编程模型：
  - DAG
  - Spout
  - Bolt
- 数据传输：
  - Zmq
    - Zmq也是开源的消息传递的框架，虽然叫mq，但它并不是一个message queue，而是一个封装的比较好的
  - Netty
    - netty是NIO的网络框架，效率比较高。之所以有netty是storm在apache之后呢，zmq的license和storm的license不兼容的，bolt处理完消息后会告诉Spout

- 核心架构



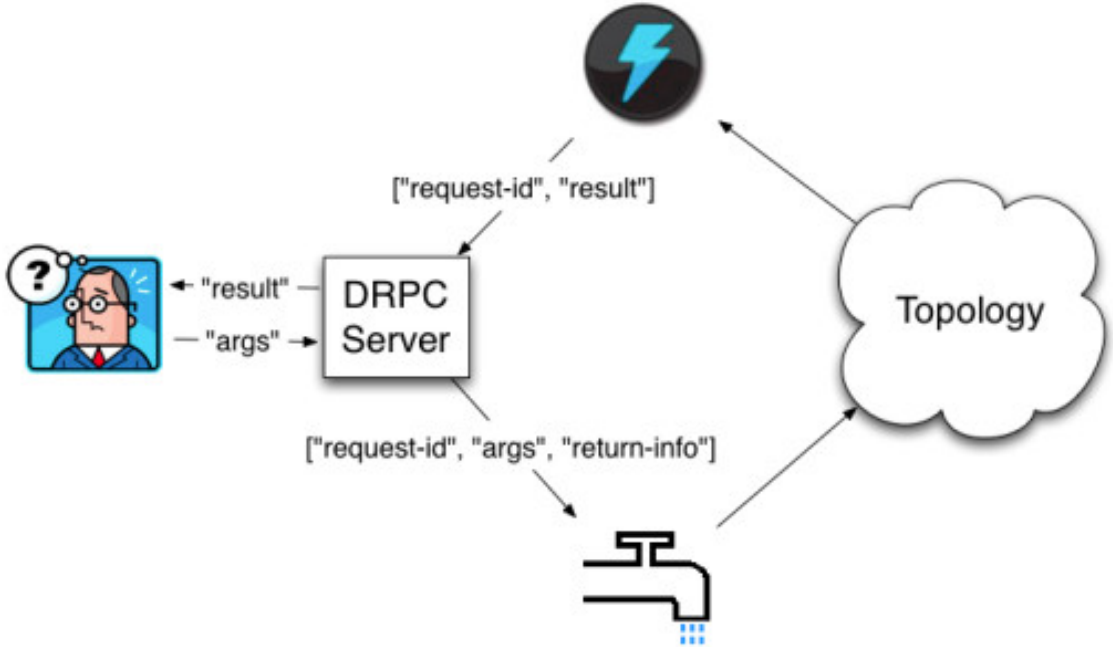
- 高可用性：
  - 异常处理
  - 消息可靠性保证机制（ACK机制）
- 可维护性：
  - Storm有个UI可以看跑在上面的程序监控

- 实时请求应答服务（同步），
  - 实时请求应答服务（同步），往往不是一个很简单的操作，而且大量的操作，用DAG模型来提高请求处理速度
  - DRPC
  - 实时请求处理
  - 例子：发送图片，或者图片地址，进行图片特征的提取

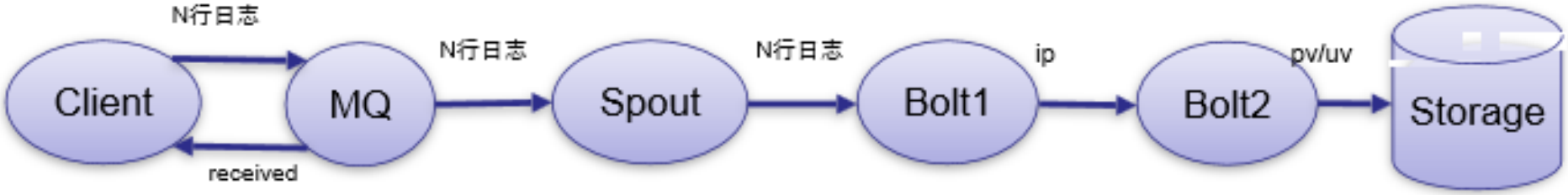


这里DRPC Server的好处是什么呢？这样看起来就像是一个Server，经过Spout，然后经过Bolt，不是更麻烦了吗？DRPC Server其实适用于分布式，可以应用分布式处理这个单个请求，来加速处理的过程。

- DRPCClient client = new DRPCClient("drpc-host", 3772);
- String result = client.execute("reach","http://twitter.com");
- 服务端由四部分组成：包括一个DRPC Server， 一个 DPRC Spout，一个Topology和一个ReturnResult。



- 流式处理（异步），不是说快，而是不等待结果
- 逐条处理
  - 例子：ETL，把关心的数据提取，标准格式入库，它的特点是我把数据给你了，不用再返回给我，这个是异步的
- 分析统计
  - 例子：日志PV，UV统计，访问热点统计，这类数据之间是有关联的，比如按某些字段做聚合，加和，平均等等



- 最后写到Redis，Hbase，MySQL，或者其他的MQ里面去给其他的系统去消费。

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

  TopologyBuilder builder = new TopologyBuilder();

  builder.setSpout("spout", new RandomSentenceSpout(), 5);

  builder.setBolt("split", new SplitSentence(), 8).shuffleGrouping("spout");
  builder.setBolt("count", new WordCount(), 12).fieldsGrouping("split", new Fields("word"));

  Config conf = new Config();
  conf.setDebug(true);


  if (args != null && args.length > 0) {
    conf.setNumWorkers(3);

    StormSubmitter.submitTopologyWithProgressBar(args[0], conf, builder.createTopology());
  }
}
```

```
storm jar storm-starter-0.9.3.jar storm.starter.WordCountTopology wordcount
```

- ShuffleGrouping("spout")就是从spout来订阅数据，fieldGrouping("split", new Fields("word"))实际上就是一个hash，同一个词有相同的hash，然后就会被hash到同一个WordCount的bolt里面，然后就可以进行计数。

- 接下来两行呢是配置文件，然后是配置3个worker，接下来是通过Submitter提交Topology到Storm集群里面去。
- 程序会编译打包，这段代码来自storm里面的starter的一段代码，这个代码怎么真正运行起来呢，就用storm jar 然后jar包的名，然后就是类的名字，和topology的名字，因为这里有个args[0]。

- 这段代码很简单，首先呢，第一部分构造了一个DAG的有向无环图，然后生成配置，提交到Storm集群去。

- 部署依赖环境
  - Java 6+
  - Python 2.6.6+
- 部署zookeeper
  - 3.4.5+
  - ZK为什么要用3.4.5，因为它支持磁盘的快照和namenode的定期删除，避免磁盘被打满
- 分发storm包
  - 0.9.4+
- 配置storm
  - 修改storm.yaml配置文件
- 启动storm

- Java,python部署省略，运行java –version和python --version验证版本
- 上传apache-storm-0.9.5.tar.gz
- tar zxf apache-storm-0.9.5.tar.gz
- cd apache-storm-0.9.5
- mkdir logs
- ./bin/storm dev-zookeeper >> ./logs/zk.out 2>&1 &
- ./bin/storm nimbus >> ./logs/nimbus.out 2>&1 &
- ./bin/storm ui >> ./logs/ui.out 2>&1 &
- ./bin/storm supervisor >> ./logs/supervisor.out 2>&1 &
- ./bin/storm logviewer >> ./logs/logviewer.out 2>&1 &
- 验证：访问http://localhost:8080，运行wordcount example
  - 单机版的步骤，storm自带了一个开发版的ZK，也不需要自己去部署，配置都不需要，极其简单的，这也是Storm的一个好处，基本上是开箱即用的，单机环境虽然没有什么实际的价值，但是在简单的测试上还是很有用的

- 查看自己机器的系统环境
- uname -a

- 查看JAVA版本
- java -version
- source /etc/profile

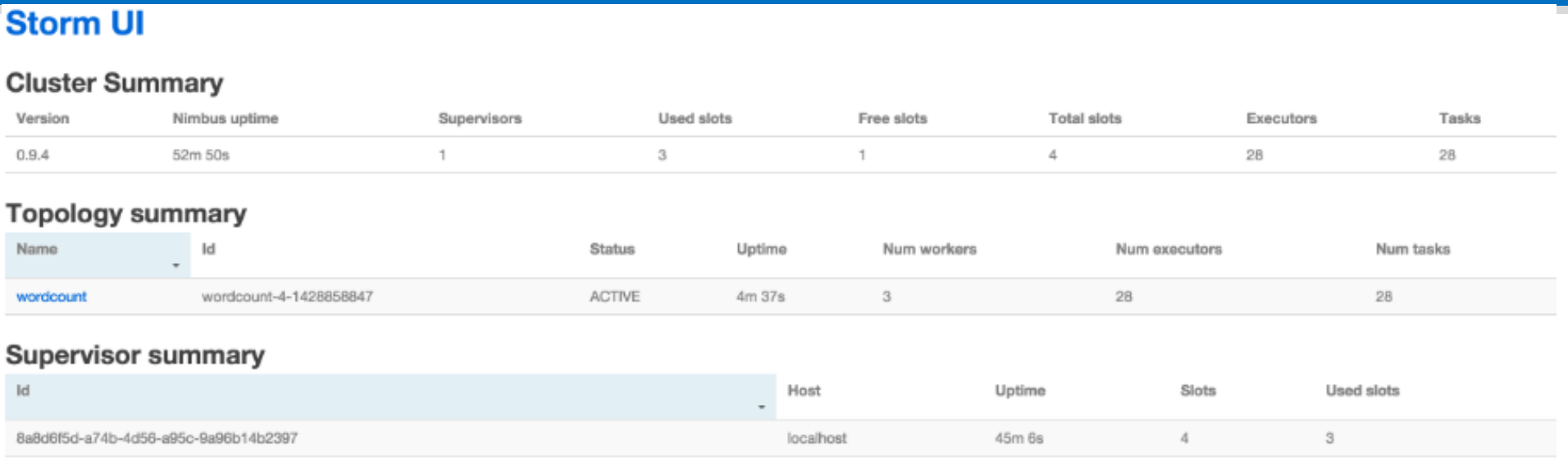
- 查看python版本
- python --version
- vi /etc/ld.so.conf
- 添加上/usr/local/lib
- 然后/sbin/ldconfig -v使生效

- ./bin/storm ui >> logs/ui.out 2>&1 &
- tail -f logs/ui.log
- 这个时候看到了ui已经起起来了，然后看到ui用的是jetty的Webserver，这个时候我们去web页面就已经可以看到东西了，http://spark001:8080/index.html
- 这个时候看到supervisor是0个，因为我们现在supervisor还没有起
- 然后配置可以看到都是默认的配置，因为我们什么都没有配置

- ./bin/storm supervisor >> logs/supervisor.out 2>&1 &
- tail -f logs/supervisor.log

- ./bin/storm logviewer >> logs/logviewer.out 2>&1 &
- tail -f logs/logviewer.log
- 它是启动在8000端口上

- 下面我们这时候刷新UI，会看到有一个supervisor起来了，有它的启动时间
- 下面我们给它提交一个最简单的topology，在我们的examples/storm-starter下面有个jar包
- ls examples/storm-starter
- jar tvf examples/storm-starter/storm-starter-topologies-0.9.5.jar | grep WordCount
- 下面我们来提交这个topology
- ./bin/storm jar examples/storm-starter/storm-starter-topologies-0.9.5.jar storm.starter.WordCountTopology wordcount



Storm UI

Cluster Summary

Version	Nimbus uptime	Supervisors	Used slots	Free slots	Total slots	Executors	Tasks
0.9.4	52m 50s	1	3	1	4	28	28

Topology summary

Name	Id	Status	Uptime	Num workers	Num executors	Num tasks
wordcount	wordcount-4-1428858847	ACTIVE	4m 37s	3	28	28

Supervisor summary

Id	Host	Uptime	Slots	Used slots
8a8d6f5d-a74b-4d56-a95c-9a96b14b2397	localhost	45m 6s	4	3

Cluster Summary（整个集群的）

- 一个slot就是一个worker，一个worker里面是一个jvm，一个worker里面呢可以有多个executor，那一个executor就是执行线程，那一个executor上面执行一个或多个Task，一般来说默认是一个task。
- Topology Summary（每个应用程序的）
- 一个应用程序就是一个Topology，它有名字，还有ID，然后有个状态，ACTIVE就是正在运行，KILLED就是已经被杀掉了。

Storm UI

Topology summary

Name	Id	Status	Uptime	Num workers	Num executors	Num tasks
wordcount	wordcount-4-1428858847	ACTIVE	4m 53s	3	28	28

Topology actions

Activate | Deactivate | Rebalance | Kill

Topology stats

Window	Emitted	Transferred	Complete latency (ms)	Acked	Failed
10m 0s	189160	101386	217.360	13679	0
3h 0m 0s	189160	101386	217.360	13679	0
1d 0h 0m 0s	189160	101386	217.360	13679	0
All time	189160	101386	217.360	13679	0

Spouts (All time)

Id	Executors	Tasks	Emitted	Transferred	Complete latency (ms)	Acked	Failed	Error Host	Error Port	Last error
spout	5	5	13684	13684	217.360	13679	0			

Bolts (All time)

Id	Executors	Tasks	Emitted	Transferred	Capacity (last 10m)	Execute latency (ms)	Executed	Process latency (ms)	Acked	Failed	Error Host	Error Port	Last error
count	12	12	87774	0	0.039	0.558	87774	0.472	87774	0			
split	8	8	87702	87702	0.000	0.055	13692	8.477	13692	0			

- Topology actions就是可以对Topology采取一些操作，Deactivate就是暂停，Rebalance就是重新做一下balance，然后kill就是杀掉这个应用。
- 这个应用运行的到底怎么样呢，在Topology stats里面有个整体的统计，有10分钟，3小时，1天，还有所有的统计，这里面比较关键的呢，是Complete latency，它的意思就是一条数据从发出去到处理完花了多长时间，第二个比较关键的呢就是ACK，这个反映的是吞吐，前面的Complete latency反映的延迟。
- 在Spouts的统计信息里面呢，一个是spout的名字，和代码里面是对应的，第二个呢是这个spout它有多少个executor，然后呢它有多少个task，然后呢是它在一定时间内往外emit出多少数据，真正tranfer传输了多少数据，然后它latency延迟是多少，然后ACK处理了多少数据，后面还有错误的信息。
- Bolt也类似，通过这个UI页面可以实时观看这些统计信息，是非常有用的，可以知道哪个环节比较慢，哪些地方有没有什么瓶颈了，有瓶颈了是不是加一个并发来解决问题。

- Spout中这里最关键的是一个nextTuple()，它是从外部取数据的源头，可以从DPRC取数据，可以从MQ，比如Kafka中取数据，然后给后面的bolt进行处理，然后这里wordcount没有那么复杂，就自己随机的生成了数据。
- _collector.emit(new Values(sentence), new Object());
- 这个代码后面new Object()等于是随机的生成了一个message的ID，这个ID有什么用，后面会讲到，实际上它是消息可靠性保障的一部分。有了这个ID呢Storm就可以帮你去跟踪这条消息到底有没有被处理完，如果处理完了呢?
- 如果处理完了，它就是调用一个ack告诉spout，我已经处理完了，这里ack方法里面仅仅是把id打印出来，因为这里id没有什么意义，仅仅是为了展示，相反，如果在一定时间内没有处理完，会调用fail告诉说消息处理失败了。

```java
public class RandomSentenceSpout extends BaseRichSpout {
  SpoutOutputCollector _collector;
  Random _rand;


  @Override
  public void open(Map conf, TopologyContext context, SpoutOutputCollector collector) {
    _collector = collector;
    _rand = new Random();
  }

  @Override
  public void nextTuple() {
    Utils.sleep(100);
    String[] sentences = new String[]{ "the cow jumped over the moon", "an apple a day k
        "four score and seven years ago", "snow white and the seven dwarfs", "i am at tw
    String sentence = sentences[_rand.nextInt(sentences.length)];
    _collector.emit(new Values(sentence), new Object());
  }

  @Override
  public void ack(Object id) {
          System.out.println("recieve ack for id " + id);
  }

  @Override
  public void fail(Object id) {
  }

  @Override
  public void declareOutputFields(OutputFieldsDeclarer declarer) {
    declarer.declare(new Fields("word"));
  }

}
```

```
public static class SplitSentence extends ShellBolt implements IRichBolt {

  public SplitSentence() {
    super("python", "splitsentence.py");
  }

  @Override
  public void declareOutputFields(OutputFieldsDeclarer declarer) {
    declarer.declare(new Fields("word"));
  }

  @Override
  public Map<String, Object> getComponentConfiguration() {
    return null;
  }
}

public static class WordCount extends BaseBasicBolt {
  Map<String, Integer> counts = new HashMap<String, Integer>();

  @Override
  public void execute(Tuple tuple, BasicOutputCollector collector) {
    String word = tuple.getString(0);
    Integer count = counts.get(word);
    if (count == null)
      count = 0;
    count++;
    counts.put(word, count);
    collector.emit(new Values(word, count));
  }

  @Override
  public void declareOutputFields(OutputFieldsDeclarer declarer) {
    declarer.declare(new Fields("word", "count"));
  }
}
```

- 对于wordcount的示例，它是有两个blot，一个bolt是分词，一个bolt是计数，这里SplitSentence是展示它支持多语言的开发，其实这里代码调用的是python的splitsentence.py，使用的是ShellBolt这个组件
- 那wordcount这个bolt是用java实现的，它的实现核心是亮点，一点是有execute这样一个函数，第二个是declareOutputFields这个函数，这两个函数的作用其实是很什么呢？最核心的其实是execute，execute的作用呢就是拿到输入的数据Tuple，然后再emit数据出去。
- 以上就是在storm里面一个最简单的wordcount的例子，它的主函数的代码，它的提交的命令行代码，Spout是什么样的，Bolt是什么样的，提交到Storm集群之后是一个什么样的运行状况，在WebUI上面看到哪些核心的信息，这个在后面的应用开发里面都会大量的运用到。

- Storm：进程、线程常驻运行，数据部进入磁盘，网络传递。
- MapReduce：TB、PB级别数据设计的，一次的批处理作业。

Storm	MapReduce
流式处理	批处理
(毫)秒级	分钟级
DAG模型	Map+Reduce模型
常驻运行	反复启停

- Storm：纯流式处理，处理数据单元是一个个Tuple。另外Storm专门为流式处理设计，它的数据传输模式更为简单，很多地方也更为高效。并不是不能做批处理，它也可以来做微批处理，来提高吞吐。
- Spark Streaming：微批处理，一个批处理怎么做流式处理呢，它基于内存和DAG可以把处理任务做的很快，把RDD做的很小来用小的批处理来接近流式处理。

Storm	SparkStreaming
流式处理	微批处理
(毫)秒级	秒级
已经很稳定	稳定性改进中
独立系统 专为流式计算设计	spark核心之上的一种计算模型 能与其他组件很好结合

- 和其它如MPI系统相比

Storm	Queue+Worker
维护简单	手动维护Queue和Worker关系
水平扩展	加并发时上下游关系变化很麻烦
自动容错	手动处理进程、机器、网络异常
通用	各个业务自行其道

- 通过对比，更能了解Storm的一些特点：
  - 首先，相对于Queue+Worker来说，它是一个通用的分布式系统，分布式系统的一些细节呢可以屏蔽掉，比如说水平扩展，容错，上层应用只需要关注自己的业务逻辑就可以了，这一点对应应用开发人员来说是非常重要的，不然的话业务逻辑会被底层的一些细节所打乱。
  - 另外一个，Storm作为一个纯的流式处理系统，和mapreduce的差异相当大，一种称为流式处理，一种称为批处理，Storm是一个常驻运行的，它的消息收发是很高效的。
- 和spark这种微批处理系统相比呢，Storm可以处理单条单条的消息。
- 总的来说呢，Storm在设计之初呢，就被定义为分布式的流式处理系统，所以说大部分的流式计算需求都可以通过Storm很好的满足，Storm目前在稳定性方面也做的相当不错，对于实时流式计算来说是个非常不错的选择