Storm入门指南

整理:毛祥溢

Email:maoxiangyi@jd.com

#### 目录

[目录 2](#_Toc393445498)

[第一章 大数据实时计算框架 4](#_Toc393445499)

[1.1 实时计算的概念 4](#_Toc393445500)

[1 数据源是实时的不间断的，要求对用户的响应时间也是实时的 4](#_Toc393445501)

[2 数据量大且无法或没必要预算，但要求对用户的响应时间是实时的。 4](#_Toc393445502)

[1.2 实时计算相关技术 4](#_Toc393445503)

[1 数据实时采集 4](#_Toc393445504)

[2 数据实时计算 4](#_Toc393445505)

[3 实时查询服务 5](#_Toc393445506)

[4 总结 5](#_Toc393445507)

[1.3 早期产品 5](#_Toc393445508)

[1 IBM的StreamBase 5](#_Toc393445509)

[2 Borealis 5](#_Toc393445510)

[1.4 近期产品 5](#_Toc393445511)

[1 Yahoo的S4 5](#_Toc393445512)

[2 Twitter实时计算 6](#_Toc393445513)

[3 Facebook 的Puma 6](#_Toc393445514)

[4 淘宝的实时计算、流式处理 6](#_Toc393445515)

[5 其它的实时计算系统 7](#_Toc393445516)

[第二章 Storm框架概述 7](#_Toc393445517)

[2.1 概述 7](#_Toc393445518)

[2.2 Storm 8](#_Toc393445519)

[1 特性 8](#_Toc393445520)

[2 Storm组件 9](#_Toc393445521)

[3 Zookeeper 9](#_Toc393445522)

[4 编程模型 spout、Bolt、Stream Groupings 9](#_Toc393445523)

[第三章 Storm 伪分布式环境搭建（Redhat） 10](#_Toc393445524)

[3.1 yum配置安装 10](#_Toc393445525)

[1 删除redhat原有的yum 10](#_Toc393445526)

[2 下载yum安装文件 10](#_Toc393445527)

[3 进行安装yum 11](#_Toc393445528)

[4 配置网易源 11](#_Toc393445529)

[5 初始化 12](#_Toc393445530)

[6 测试 12](#_Toc393445531)

[7 yum资料 12](#_Toc393445532)

[3.2 搭建Zookeeper集群 12](#_Toc393445533)

[1 搭建Zookeeper集群 12](#_Toc393445534)

[2 安装Storm依赖库 13](#_Toc393445535)

[3 下载并解压Storm发布版本 14](#_Toc393445536)

[4 修改storm.yaml配置文件 15](#_Toc393445537)

[5 启动Storm各个后台进程 15](#_Toc393445538)

[6 向集群提交任务 16](#_Toc393445539)

[7 提示 16](#_Toc393445540)

[3.3 软件列表 17](#_Toc393445541)

[1 目录 17](#_Toc393445542)

[2 软件包 18](#_Toc393445543)

[第四章 Storm Demo编写 18](#_Toc393445544)

[4.1 开发环境搭建 18](#_Toc393445545)

[1 安装eclipse 18](#_Toc393445546)

[2 安装maven 18](#_Toc393445547)

[3 安装maven2eclipse 18](#_Toc393445548)

[4 pom.xml 18](#_Toc393445549)

[4.2 wordcounter单词计数器 18](#_Toc393445550)

[1 拓扑驱动类 18](#_Toc393445551)

[2 WordReader读取数据源 22](#_Toc393445552)

[3 WordNormalizer切割句子成单词 25](#_Toc393445553)

[4 WordCounter 计数器 26](#_Toc393445554)

[5 拓扑流程日志 28](#_Toc393445555)

#### 大数据实时计算框架

##### 1.1 实时计算的概念

互联网领域的实时计算一般都是针对海量数据进行的，除了像非实时计算的需求（如计算结果准确）以外，实时计算最重要的一个需求是能够实时响应计算结果，一般要求为秒级。个人理解，互联网行业的实时计算可以分为以下两种应用场景：

###### 1 数据源是实时的不间断的，要求对用户的响应时间也是实时的

主要用于互联网流式数据处理。所谓流式数据是指将数据看作是数据流的形式来处理。数据流则是在时间分布和数量上无限的一系列数据记录的集合体；数据记录是数据流的最小组成单元。举个例子，对于大型网站，活跃的流式数据非常常见，这些数据包括网站的访问PV/UV、用户访问了什么内容，搜索了什么内容等。实时的数据计算和分析可以动态实时地刷新用户访问数据，展示网站实时流量的变化情况，分析每天各小时的流量和用户分布情况，这对于大型网站来说具有重要的实际意义。

###### 2 数据量大且无法或没必要预算，但要求对用户的响应时间是实时的。

主要用于特定场合下的数据分析处理。当数据量很大，同时发现无法穷举所有可能条件的查询组合或者大量穷举出来的条件组合无用的时候，实时计算就可以发挥作用，将计算过程推迟到查询阶段进行，但需要为用户提供实时响应。

##### 1.2 实时计算相关技术

互联网上海量数据（一般为日志流）的实时计算过程可以被划分为以下三个阶段：数据的产生与收集阶段、传输与分析处理阶段、存储对对外提供服务阶段。

下面分别进行简单的介绍：

###### 1 数据实时采集

需求：功能上保证可以完整的收集到所有日志数据，为实时应用提供实时数据；响应时间上要保证实时性、低延迟在1秒左右；配置简单，部署容易；系统稳定可靠等。

目前，互联网企业的海量数据采集工具，有Facebook开源的Scribe、LinkedIn开源的Kafka、Cloudera开源的Flume，淘宝开源的TimeTunnel、Hadoop的Chukwa等，均可以满足每秒数百MB的日志数据采集和传输需求。

###### 2 数据实时计算

传统的数据操作，首先将数据采集并存储在DBMS中，然后通过query和DBMS进行交互，得到用户想要的答案。整个过程中，用户是主动的，而DBMS系统是被动的。但是，对于现在大量存在的实时数据，比如股票交易的数据，这类数据实时性强，数据量大，没有止境，传统的架构并不合适。流计算就是专门针对这种数据类型准备的。在流数据不断变化的运动过程中实时地进行分析，捕捉到可能对用户有用的信息，并把结果发送出去。整个过程中，数据分析处理系统是主动的，而用户却是处于被动接收的状态。

需求：适应流式数据、不间断查询；系统稳定可靠、可扩展性好、可维护性好等。

实时流计算框架：Yahoo开源的S4、Twitter开源的Storm，还有Esper，Streambase，HStreaming等。有关计算的一些注意点：分布式计算，并行计算（节点间的并行、节点内的并行），热点数据的缓存策略，服务端计算。

###### 3 实时查询服务

全内存：直接提供数据读取服务，定期dump到磁盘或数据库进行持久化。

半内存：使用Redis、Memcache、MongoDB、BerkeleyDB等内存数据库提供数据实时查询服务，由这些系统进行持久化操作。

全磁盘：使用HBase等以分布式文件系统（HDFS）为基础的NoSQL数据库，对于key-value引擎，关键是设计好key的分布。

###### 4 总结

A：并不是任何应用都做到实时计算才是最好的。

B：使用哪些技术和框架来搭建实时计算系统，需要根据实际业务需求进行选择。

C：对于分布式系统来说，系统的可配置性、可维护性、可扩展性十分重要，系统调优永无止境。

##### 1.3 早期产品

###### 1 IBM的StreamBase

StreamBase是IBM开发的一款商业流式计算系统，在金融行业和政府部门使用

官方网站：http://www.streambase.com

###### 2 Borealis

Brandeis University、Brown University和MIT合作开发的一个分布式流式系统，由之前的流式系统Aurora、Medusa演化而来，学术研究的一个产品，08年已经停止维护

##### 1.4 近期产品

###### 1 Yahoo的S4

S4是一个通用的、分布式的、可扩展的、分区容错的、可插拔的流式系统，Yahoo！开发S4系统，主要是为了解决：搜索广告的展现、处理用户的点击反馈。

官方网站：http://s4.io/

S4简介：<http://www.programmer.com.cn/5304/>

###### 2 Twitter实时计算

2.1 Twitter的storm：Storm是一个分布式的、容错的实时计算系统

Storm用途：可用于处理消息和更新数据库（流处理），在数据流上进行持续查询，并以流的形式返回结果到客户端（持续计算），并行化一个类似实时查询的热点查询（分布式的RPC）。

官方指南：https://storm.canonical.com/Tutorial

github下载：https://github.com/nathanmarz/storm/downloads

storm详解：http://duanple.blog.163.com/blog/static/7097176720111020102057795/

storm配置详解：http://www.tbdata.org/archives/2118

storm翻译及总结博客：<http://chenlx.blog.51cto.com/4096635/d-1/p-1>

2.2. Twitter的Rainbird：Rainbird 是一款分布式实时统计系统, Rainbird可以用于实时数据的统计：(1)统计网站中每一个页面，域名的点击次数,(2)内部系统的运行监控（统计被监控服务器的运行状态），(3) 记录最大值和最小值。

官方简介：

http://www.slideshare.net/kevinweil/rainbird-realtime-analytics-at-twitter-strata-2011

中文介绍：

<http://www.cnblogs.com/gpcuster/archive/2011/02/06/1949466.html>

###### 3 Facebook 的Puma

facebook使用puma和Habase相结合来处理实时数据，另外facebook发表一篇利用HBase/Hadoop进行实时数据处理的论文（Apache Hadoop Goes Realtime at Facebook），通过一些实时性改造，让批处理计算平台也具备实时计算的能力。

###### 4 淘宝的实时计算、流式处理

4.1银河流数据处理平台:通用的流数据实时计算系统，以实时数据产出的低延迟、高吞吐和复用性为初衷和目标，采用actor模型构建分布式流数据计算框架（底层基于akka），功能易扩展、部分容错、数据和状态可监控。 银河具有处理实时流数据（如TimeTunnel收集的实时数据）和静态数据（如本地文件、HDFS文件）的能力，能够提供灵活的实时数据输出，并提供自定义的数据输出接口以便扩展实时计算能力。 银河目前主要是为魔方提供实时的交易、浏览和搜索日志等数据的实时计算和分析。

4.2.基于storm的流式处理，统计计算、持续计算、实时消息处理。

4.3利用Habase实现的online应用。

###### 5 其它的实时计算系统

Hstreaming：官方网站：http://www.hstreaming.com/technology/hstreaming/

Esper：esper可以用在股票系统、风险监控系统等等要求实时性比较高的系统中

官方网站：http://www.espertech.com/

官网的英文简介：http://www.espertech.com/products/esper.php

中文简介：<http://www.cnblogs.com/qlee/archive/2011/06/22/2086550.html>

以上数据来源于：http://www.cnblogs.com/panfeng412/archive/2011/10/28/2227195.html

#### Storm框架概述

##### 2.1 概述

在过去的十年里，数据处理发生了革命性的变化。MapReduce，Hadoop，以及相关的技术使我们可以存储和处理以前不可想象规模的数据。很遗憾，这些数据处理系统都不是实时系统，命中注定也不是它们。根本没办法把Hadoop变成一个实时系统；实时数据处理和批处理的许多要求在根本上有很大不同。

然而，企业对大规模实时数据处理要求越来越多。缺乏“实时Hadoop”是数据处理生态系统中最大的窘境。

Storm解决了这个窘境。

**Storm之前**，你通常必须手动建立一个由许多队列和许多worker组成的网络来实现实时处理。worker处理队列消息，更新数据库，发送新消息给其它队列以供后续处理。很遗憾，

这种方法有很大的局限性：

乏味：你大部份开发时间花费在配置消息发送，部署worker，部署中间队列。你关心的实时处理逻辑对应到你的代码的比例相对较小 。

脆弱：没有多少容错。你负责保持每个worker和队列正常工作。

痛苦伸缩：当单个worker或队列的消息吞吐量太高时，你需要分区，即数据如何分散。你需要重新配置其它worker，让它们发送消息到新位置。这导致删除或添加部件都可能失败。

虽然队列+workers的范式能解决大量的消息，消息处理显然是实时计算的基本范式。问题是：你要怎么做，才能在某种程度上保证数据不会丢失，对海量消息轻松扩容，并且使用和运营工作都超级简单呢？Storm满足这些目标。

Storm公开（expose）一组实时计算原语。类似MapReduce极大地简化了编写并行批处理程序，storm的原语极大地简化了编写并行实时计算程序。

Storm的关键特性：

用例非常广泛：Storm可用于处理消息和更新数据库（流处理），在数据流上进行持续查询，并以流的形式返回结果到客户端（持续计算），并行化一个类似实时查询的热点查询（分布式的RPC），还有更多的用例。Storm的一组很小的原语满足了惊人数量的用例。

可伸缩：Storm随时都可对大规模消息进行扩容。扩容一个拓扑，你只需要添加机器和增加的拓扑结构的并行设置。看一个storm规模的例子，一个storm集群有10个节点，一个最初的Storm应用每秒可以处理1,000,000个消息（指spout和bolt总共发射的消息总和），拓扑的其中一部分每秒数有数百个数据库调用。Storm使用Zookeeper协调集群，使其集群可以扩容到非常大。

保证数据不丢失：实时系统必须对成功处理数据提供有力保证 。系统丢弃数据的用例非常有限。Storm保证每个消息都被处理，这直接与其它系统截然不同，如S4。

非常健壮：Storm与Hadoop不同，Hadoop难于管理早已臭名昭著，Storm集群只是干活。使用户尽可能方便地管理storm集群是storm项目的一个明确目标。

容错：计算的执行过程中如果发生故障，Storm将在必要时重新分配任务。Storm确保计算永远运行（或者直到你kill此计算） 。

编程语言无关性：健壮和可伸缩的实时处理不应仅限于一个单一的平台。Storm的拓扑结构和处理组件可以用任何语言定义，对任何人而言，Storm都是易接受的。

更多信息请参考：<http://www.maoxiangyi.cn/index.php/archives/337>

##### 2.2 Storm

###### 1 特性

　　对比Hadoop的批处理，Storm是个实时的、分布式以及具备高容错的计算系统。同Hadoop一样Storm也可以处理大批量的数据，然而Storm在保证高可靠性的前提下还可以让处理进行的更加实时;也就是说，所有的信息都会被处理。Storm同样还具备容错和分布计算这些特性，这就让Storm可以扩展到不同的机器上进行大批量的数据处理。他同样还有以下的这些特性：

　　·易于扩展。对于扩展，你只需要添加机器和改变对应的topology(拓扑)设置。Storm使用Hadoop Zookeeper进行集群协调，这样可以充分的保证大型集群的良好运行。

　　·每条信息的处理都可以得到保证。

　　·Storm集群管理简易。

　　·Storm的容错机能：一旦topology递交，Storm会一直运行它直到topology被废除或者被关闭。而在执行中出现错误时，也会由Storm重新分配任务。

　　·尽管通常使用Java，Storm中的topology可以用任何语言设计。

　　当然为了更好的理解文章，你首先需要安装和设置Storm。需要通过以下几个简单的步骤：

　　·从Storm官方下载Storm安装文件

　　·将bin/directory解压到你的PATH上，并保证bin/storm脚本是可执行的。

###### 2 Storm组件

　　Storm集群主要由一个主节点和一群工作节点(worker node)组成，通过 Zookeeper进行协调。

　　主节点：

　　主节点通常运行一个后台程序 —— Nimbus，用于响应分布在集群中的节点，分配任务和监测故障。这个很类似于Hadoop中的Job Tracker。

　　工作节点：

　　工作节点同样会运行一个后台程序 —— Supervisor，用于收听工作指派并基于要求运行工作进程。每个工作节点都是topology中一个子集的实现。而Nimbus和Supervisor之间的协调则通过Zookeeper系统或者集群。

###### 3 Zookeeper

Zookeeper是完成Supervisor和Nimbus之间协调的服务。而应用程序实现实时的逻辑则被封装进Storm中的“topology”。topology则是一组由Spouts(数据源)和Bolts(数据操作)通过Stream Groupings进行连接的图。

###### 4 编程模型 spout、Bolt、Stream Groupings

　　Spout

　　简而言之，Spout从来源处读取数据并放入topology。Spout分成可靠和不可靠两种;当Storm接收失败时，可靠的Spout会对tuple(元组，数据项组成的列表)进行重发;而不可靠的Spout不会考虑接收成功与否只发射一次。而Spout中最主要的方法就是nextTuple()，该方法会发射一个新的tuple到topology，如果没有新tuple发射则会简单的返回。

　　Bolt

　　Topology中所有的处理都由Bolt完成。Bolt可以完成任何事，比如：连接的过滤、聚合、访问文件/数据库、等等。Bolt从Spout中接收数据并进行处理，如果遇到复杂流的处理也可能将tuple发送给另一个Bolt进行处理。而Bolt中最重要的方法是execute()，以新的tuple作为参数接收。不管是Spout还是Bolt，如果将tuple发射成多个流，这些流都可以通过declareStream()来声明。

Stream Groupings

　　Stream Grouping定义了一个流在Bolt任务间该如何被切分。这里有Storm提供的6个Stream Grouping类型：

　　1. 随机分组(Shuffle grouping)：随机分发tuple到Bolt的任务，保证每个任务获得相等数量的tuple。

　　2. 字段分组(Fields grouping)：根据指定字段分割数据流，并分组。例如，根据“user-id”字段，相同“user-id”的元组总是分发到同一个任务，不同“user-id”的元组可能分发到不同的任务。

　　3. 全部分组(All grouping)：tuple被复制到bolt的所有任务。这种类型需要谨慎使用。

　　4. 全局分组(Global grouping)：全部流都分配到bolt的同一个任务。明确地说，是分配给ID最小的那个task。

　　5. 无分组(None grouping)：你不需要关心流是如何分组。目前，无分组等效于随机分组。但最终，Storm将把无分组的Bolts放到Bolts或Spouts订阅它们的同一线程去执行(如果可能)。

　　6. 直接分组(Direct grouping)：这是一个特别的分组类型。元组生产者决定tuple由哪个元组处理者任务接收。

　　当然还可以实现CustomStreamGroupimg接口来定制自己需要的分组。

#### 第三章 Storm 伪分布式环境搭建（Redhat）

##### 3.1 yum配置安装

在配置storm集群时需要安装很多依赖包，手动安装的话比较费时费力，还不一定能安装成功。建议使用yum进行安装，先配置下Yum。

由于 redhat的yum在线更新是收费的，如果没有注册的话不能使用，如果要使用，需将redhat的yum卸载后，重启安装，再配置其他源.第三方源包括：网易，epel,repoforge ,rpmfusion 以下为详细过程：

###### 1 删除redhat原有的yum

rpm -aq|grep yum|xargs rpm -e --nodeps

###### 2 下载yum安装文件

（以下路径需要和redhat的版本对应）

注意，如果下载时找不到文件，就登录到：http://mirrors.163.com/centos/6/os/x86\_64/ 上查找相应的文件。然后再下载。

Wget http://mirrors.163.com/centos/6/os/x86\_64/Packages/yum-3.2.27-14.el6.centos.noarch.rpm

wget http://mirrors.163.com/centos/6/os/x86\_64/Packages/yum-metadata-parser-1.1.2-14.1.el6.x86\_64.rpm

wget http://mirrors.163.com/centos/6/os/x86\_64/Packages/yum-plugin-fastestmirror-1.1.26-11.el6.noarch.rpm

wget http://mirrors.163.com/centos/6/os/x86\_64/Packages/python-iniparse-0.3.1-2.1.el6.noarch.rpm

###### 3 进行安装yum

rpm -ivh python-iniparse-0.3.1-2.1.el6.noarch.rpm

rpm -ivh yum-metadata-parser-1.1.2-14.1.el6.x86\_64.rpm

rpm -ivh yum-3.2.27-14.el6.centos.noarch.rpm um-plugin-fastestmirror-1.1.26-11.el6.noarch.rpm

注意最后两个包必需同时安装，否则会相互依赖

－－－－－－－－－－配置网易源

###### 4 配置网易源

mv /etc/yum.repos.d/rhel-debuginfo.repo /etc/yum.repos.d/rhel-debuginfo.repo.repo.bak

vi /etc/yum.repos.d/rhel-debuginfo.repo

内容为：(url的地址要根据不同的系统进行改变)

[base]

name=CentOS-$releasever - Base

baseurl=http://mirrors.163.com/centos/6.0/os/$basearch/

gpgcheck=1

gpgkey=http://mirrors.163.com/centos/RPM-GPG-KEY-CentOS-6

#released updates

[updates]

name=CentOS-$releasever - Updates

baseurl=http://mirrors.163.com/centos/6.0/updates/$basearch/

gpgcheck=1

gpgkey=http://mirrors.163.com/centos/RPM-GPG-KEY-CentOS-6

#packages used/produced in the build but not released

#[addons]

#name=CentOS-$releasever - Addons

#baseurl=http://mirrors.163.com/centos/$releasever/addons/$basearch/

#gpgcheck=1

#gpgkey=http://mirrors.163.com/centos/RPM-GPG-KEY-CentOS-6

#additional packages that may be useful

[extras]

name=CentOS-$releasever - Extras

baseurl=http://mirrors.163.com/centos/6.0/extras/$basearch/

gpgcheck=1

gpgkey=http://mirrors.163.com/centos/RPM-GPG-KEY-CentOS-6

#additional packages that extend functionality of existing packages

[centosplus]

name=CentOS-$releasever - Plus

baseurl=http://mirrors.163.com/centos/6.0/centosplus/$basearch/

gpgcheck=1

enabled=0

###### 5 初始化

yum clean all

###### 6 测试

yum install vim #测试一下可不可以用

###### 7 yum资料

以上配置节选于：<http://space.itpub.net/25313300/viewspace-708509>

##### 3.2 搭建Zookeeper集群

###### 1 搭建Zookeeper集群

Storm使用Zookeeper协调集群，由于Zookeeper并不用于消息传递，所以Storm给Zookeeper带来的压力相当低。大多数情况下，单个节点的Zookeeper集群足够胜任，不过为了确保故障恢复或者部署大规模Storm集群，可能需要更大规模节点的Zookeeper集群（对于Zookeeper集群的话，官方推荐的最小节点数为3个）。

在Zookeeper集群的每台机器上完成以下安装部署步骤：

1. 下载安装Java JDK，官方下载链接为http://java.sun.com/javase/downloads/index.jsp，JDK版本为JDK 6或以上。

2. 根据Zookeeper集群的负载情况，合理设置Java堆大小，尽可能避免发生swap，导致Zookeeper性能下降。保守起见，4GB内存的机器可以为Zookeeper分配3GB最大堆空间。

3. 下载后解压安装Zookeeper包，官方下载链接为http://hadoop.apache.org/zookeeper/releases.html。

4. 根据Zookeeper集群节点情况，在conf目录下创建Zookeeper配置文件zoo.cfg：

tickTime=2000

dataDir=/var/zookeeper/

clientPort=2181

initLimit=5

syncLimit=2

server.1=zoo1:2888:3888

server.2=zoo2:2888:3888

server.3=zoo3:2888:3888

其中，dataDir指定Zookeeper的数据文件目录；其中server.id=host:port:port，id是为每个Zookeeper节点的编号，保存在dataDir目录下的myid文件中，zoo1~zoo3表示各个Zookeeper节点的hostname，第一个port是用于连接leader的端口，第二个port是用于leader选举的端口。

5. 在dataDir目录下创建myid文件，文件中只包含一行，且内容为该节点对应的server.id中的id编号。

6. 启动Zookeeper服务：

Java -cp zookeeper.jar:lib/log4j-1.2.15.jar:conf \ org.apache.zookeeper.server.quorum.QuorumPeerMain zoo.cfg

或者bin/zkServer.sh start

7. 通过Zookeeper客户端测试服务是否可用：

Java -cp zookeeper.jar:src/java/lib/log4j-1.2.15.jar:conf:src/java/lib/jline-0.9.94.jar \ org.apache.zookeeper.ZooKeeperMain -server 127.0.0.1:2181

或者

bin/zkCli.sh -server 127.0.0.1:2181

注意事项：

由于Zookeeper是快速失败（fail-fast)的，且遇到任何错误情况，进程均会退出，因此，最好能通过监控程序将Zookeeper管理起来，保证Zookeeper退出后能被自动重启。详情参考这里。

Zookeeper运行过程中会在dataDir目录下生成很多日志和快照文件，而Zookeeper运行进程并不负责定期清理合并这些文件，导致占用大量磁盘空间，因此，需要通过cron等方式定期清除没用的日志和快照文件。详情参考这里。具体命令格式如下：java -cp zookeeper.jar:log4j.jar:conf org.apache.zookeeper.server.PurgeTxnLog <dataDir> <snapDir> -n <count>

###### 2 安装Storm依赖库

1 安装ZMQ 2.1.7请勿使用2.1.10版本，因为该版本的一些严重bug会导致Storm集群运行时出现奇怪的问题。少数用户在2.1.7版本会遇到”IllegalArgumentException”的异常，此时降为2.1.4版本可修复这一问题。

下载后编译安装ZMQ：

wget http://download.zeromq.org/zeromq-2.1.7.tar.gz

tar -xzf zeromq-2.1.7.tar.gz

cd zeromq-2.1.7

./configure

make

sudo make install

注意事项：

如果安装过程报错uuid找不到，则通过如下的包安装uuid库：

sudo yum install e2fsprogsl -b current

sudo yum install e2fsprogs-devel -b current

2 安装JZMQ

下载后编译安装JZMQ：

git clone https://github.com/nathanmarz/jzmq.git

cd jzmq

./autogen.sh

./configure

make

sudo make install

为了保证JZMQ正常工作，可能需要完成以下配置：

正确设置 JAVA\_HOME环境变量

安装Java开发包

升级autoconf

3 安装Java 6

4 安装Python2.6.6

1. 下载Python2.6.6：

wget http://www.python.org/ftp/python/2.6.6/Python-2.6.6.tar.bz2

2. 编译安装Python2.6.6：

tar –jxvf Python-2.6.6.tar.bz2

cd Python-2.6.6

./configure

make

make install

3. 测试Python2.6.6

python -V

Python 2.6.6

5 安装unzip

1. 如果使用RedHat系列Linux系统，执行以下命令安装unzip：

apt-get install unzip

2. 如果使用Debian系列Linux系统，执行以下命令安装unzip：

yum install unzip

###### 3 下载并解压Storm发布版本

1. 下载Storm发行版本，推荐使用Storm0.8.1：

wget https://github.com/downloads/nathanmarz/storm/storm-0.8.1.zip

2. 解压到安装目录下：

unzip storm-0.8.1.zip

###### 4 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。以下配置选项是必须在conf/storm.yaml中进行配置的：

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"

3) java.library.path: Storm使用的本地库（ZMQ和JZMQ）加载路径，默认为”/usr/local/lib:/opt/local/lib:/usr/lib”，一般来说ZMQ和JZMQ默认安装在/usr/local/lib 下，因此不需要配置即可。

4) nimbus.host: Storm集群Nimbus机器地址，各个Supervisor工作节点需要知道哪个机器是Nimbus，以便下载Topologies的jars、confs等文件，如：

nimbus.host: "111.222.333.444"

5) supervisor.slots.ports: 对于每个Supervisor工作节点，需要配置该工作节点可以运行的worker数量。每个worker占用一个单独的端口用于接收消息，该配置选项即用于定义哪些端口是可被worker使用的。默认情况下，每个节点上可运行4个workers，分别在6700、6701、6702和6703端口，如：

supervisor.slots.ports:

- 6700

- 6701

- 6702

- 6703

###### 5 启动Storm各个后台进程

最后一步，启动Storm的所有后台进程。和Zookeeper一样，Storm也是快速失败（fail-fast)的系统，这样Storm才能在任意时刻被停止，并且当进程重启后被正确地恢复执行。这也是为什么Storm不在进程内保存状态的原因，即使Nimbus或Supervisors被重启，运行中的Topologies不会受到影响。

以下是启动Storm各个后台进程的方式：

Nimbus: 在Storm主控节点上运行”bin/storm nimbus >/dev/null 2>&1 &”启动Nimbus后台程序，并放到后台执行；

Supervisor: 在Storm各个工作节点上运行”bin/storm supervisor >/dev/null 2>&1 &”启动Supervisor后台程序，并放到后台执行；

UI: 在Storm主控节点上运行”bin/storm ui >/dev/null 2>&1 &”启动UI后台程序，并放到后台执行，启动后可以通过http://{nimbus host}:8080观察集群的worker资源使用情况、Topologies的运行状态等信息。

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。

###### 6 向集群提交任务

1. 启动Storm Topology：

storm jar allmycode.jar org.me.MyTopology arg1 arg2 arg3

其中，allmycode.jar是包含Topology实现代码的jar包，org.me.MyTopology的main方法是Topology的入口，arg1、arg2和arg3为org.me.MyTopology执行时需要传入的参数。

2. 停止Storm Topology：

storm kill {toponame}

其中，{toponame}为Topology提交到Storm集群时指定的Topology任务名称。

###### 7 提示

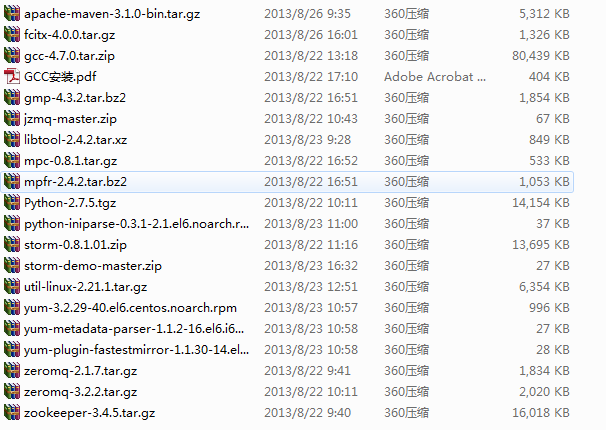
以上资源来自于：

<http://blog.linezing.com/category/storm-quick-start?spm=0.0.0.0.w2ow2L>

在安装的过程中，会提示安装gcc 等等库文件，使用yum命令依次进行安装。在解决依赖库的过程中，建议参考：<http://blog.sina.com.cn/s/blog_546abd9f0101cce8.html>

##### 3.3 软件列表

###### 1 目录



###### 2 软件包

#### 第四章 Storm Demo编写

##### 4.1 开发环境搭建

###### 1 安装eclipse

###### 2 安装maven

###### 3 安装maven2eclipse

###### 4 pom.xml

##### 4.2 wordcounter单词计数器

###### 1 拓扑驱动类

package cn.jd.storm;

import backtype.storm.Config;

import backtype.storm.LocalCluster;

import backtype.storm.topology.TopologyBuilder;

import backtype.storm.tuple.Fields;

/\*\*

\* 功能说明：

\* 设计一个topology，来实现对一个句子里面的单词出现的频率进行统计。

\* 整个topology分为三个部分：

\* WordReader:数据源，负责发送单行文本记录（句子）

\* WordNormalizer:负责将单行文本记录（句子）切分成单词

\* WordCounter:负责对单词的频率进行累加

\*

\* @author 毛祥溢

\* Email:frank@maoxiangyi.cn

\* 2013-8-26 下午5:59:06

\*/

public class TopologyMain {

/\*\*

\* @param args 文件路径

\*/

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

// Storm框架支持多语言，在JAVA环境下创建一个拓扑，需要使用TopologyBuilder进行构建

TopologyBuilder builder = new TopologyBuilder();

/\* WordReader类，主要是将文本内容读成一行一行的模式

\* 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。

\* Spout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。多条消息流可以理解为多中类型的数据。

\* 使用OutputFieldsDeclarer.declareStream来定义多个stream，然后使用SpoutOutputCollector来发射指定的stream。

\*

\* Spout类里面最重要的方法是nextTuple。要么发射一个新的tuple到topology里面或者简单的返回如果已经没有新的tuple。

\* 要注意的是nextTuple方法不能阻塞，因为storm在同一个线程上面调用所有消息源spout的方法。

\*

\* 另外两个比较重要的spout方法是ack和fail。storm在检测到一个tuple被整个topology成功处理的时候调用ack，否则调用fail。storm只对可靠的spout调用ack和fail。

\*/

builder.setSpout("word-reader",new WordReader());

/\* WordNormalizer类，主要是将一行一行的文本内容切割成单词

\*

\* 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。复杂的消息流处理往往需要很多步骤，从而也就需要经过很多bolts。

\* 比如算出一堆图片里面被转发最多的图片就至少需要两步：

\* 第一步算出每个图片的转发数量。

\* 第二步找出转发最多的前10个图片。（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，bolts使用OutputCollector来发射tuple。

\* bolts必须要为它处理的每一个tuple调用OutputCollector的ack方法，以通知Storm这个tuple被处理完成了，从而通知这个tuple的发射者spouts。

\* 一般的流程是： bolts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。

\*

\*

\*/

builder.setBolt("word-normalizer", new WordNormalizer()).shuffleGrouping("word-reader");

/\*

\* 上面的代码和下面的代码中都设定了数据分配的策略stream grouping

\* 定义一个topology的其中一步是定义每个bolt接收什么样的流作为输入。stream grouping就是用来定义一个stream应该如果分配数据给bolts上面的多个tasks。

\* 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stream grouping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。

\* Direct Grouping： 直接分组， 这是一种比较特别的分组方法，用这种分组意味着消息的发送者指定由消息接收者的哪个task处理这个消息。

\* 只有被声明为Direct Stream的消息流可以声明这种分组方法。而且这种消息tuple必须使用emitDirect方法来发射。

\* 消息处理者可以通过TopologyContext来获取处理它的消息的task的id （OutputCollector.emit方法也会返回task的id）。

\* Local or shuffle grouping：如果目标bolt有一个或者多个task在同一个工作进程中，tuple将会被随机发生给这些tasks。

\* 否则，和普通的Shuffle Grouping行为一致。

\*

\*/

builder.setBolt("word-counter", new WordCounter(),1).fieldsGrouping("word-normalizer", new Fields("word"));

/\*

\* storm的运行有两种模式: 本地模式和分布式模式.

\* 1) 本地模式：

\* 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。

\* 你运行storm-starter里面的topology的时候它们就是以本地模式运行的， 你可以看到topology里面的每一个组件在发射什么消息。

\* 2) 分布式模式：

\* 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。

\* 下面是以本地模式运行的代码:

\*

\* 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。

\*/

Config conf = new Config();

conf.setDebug(true);

conf.setNumWorkers(2);

conf.put("wordsFile","/root/workspace1/com.jd.storm.demo/src/main/resources/words.txt");

conf.setDebug(true);

conf.put(Config.TOPOLOGY\_MAX\_SPOUT\_PENDING, 1);

/\*

\* 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。

\*/

LocalCluster cluster = new LocalCluster();

cluster.submitTopology("wordCounterTopology", conf, builder.createTopology());

Thread.sleep(1000);

cluster.killTopology("wordCounterTopology");

cluster.shutdown();

}

}

###### 2 WordReader读取数据源

package cn.jd.storm;

import java.io.BufferedReader;

import java.io.FileNotFoundException;

import java.io.FileReader;

import java.util.Map;

import backtype.storm.spout.SpoutOutputCollector;

import backtype.storm.task.TopologyContext;

import backtype.storm.topology.OutputFieldsDeclarer;

import backtype.storm.topology.base.BaseRichSpout;

import backtype.storm.tuple.Fields;

import backtype.storm.tuple.Values;

/\*\*

\*

\* 功能说明：

\* 主要是将文件内容读出来，一行一行

\*

\* Spout类里面最重要的方法是nextTuple。

\* 要么发射一个新的tuple到topology里面或者简单的返回如果已经没有新的tuple。

\* 要注意的是nextTuple方法不能阻塞，因为storm在同一个线程上面调用所有消息源spout的方法。

\* 另外两个比较重要的spout方法是ack和fail。

\* storm在检测到一个tuple被整个topology成功处理的时候调用ack，否则调用fail。

\* storm只对可靠的spout调用ack和fail。

\*

\* @author 毛祥溢

\* Email:frank@maoxiangyi.cn

\* 2013-8-26 下午6:05:46

\*/

public class WordReader extends BaseRichSpout {

private SpoutOutputCollector collector;

private FileReader fileReader;

private String filePath;

private boolean completed = false;

//storm在检测到一个tuple被整个topology成功处理的时候调用ack，否则调用fail。

public void ack(Object msgId) {

System.out.println("OK:"+msgId);

}

public void close() {}

//storm在检测到一个tuple被整个topology成功处理的时候调用ack，否则调用fail。

public void fail(Object msgId) {

System.out.println("FAIL:"+msgId);

}

/\*

\* 在SpoutTracker类中被调用，每调用一次就可以向storm集群中发射一条数据（一个tuple元组），该方法会被不停的调用

\*/

public void nextTuple() {

if(completed){

try {

Thread.sleep(1000);

} catch (InterruptedException e) {

}

return;

}

String str;

BufferedReader reader =new BufferedReader(fileReader);

try{

while((str = reader.readLine()) != null){

System.out.println("WordReader类 读取到一行数据："+ str);

this.collector.emit(new Values(str),str);

System.out.println("WordReader类 发射了一条数据："+ str);

}

}catch(Exception e){

throw new RuntimeException("Error reading tuple",e);

}finally{

completed = true;

}

}

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

try {

this.fileReader = new FileReader(conf.get("wordsFile").toString());

} catch (FileNotFoundException e) {

throw new RuntimeException("Error reading file ["+conf.get("wordFile")+"]");

}

this.filePath = conf.get("wordsFile").toString();

this.collector = collector;

}

/\*\*

\* 定义字段id，该id在简单模式下没有用处，但在按照字段分组的模式下有很大的用处。

\* 该declarer变量有很大作用，我们还可以调用declarer.declareStream();来定义stramId，该id可以用来定义更加复杂的流拓扑结构

\*/

public void declareOutputFields(OutputFieldsDeclarer declarer) {

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

}

}

###### 3 WordNormalizer切割句子成单词

package cn.jd.storm;

import backtype.storm.topology.BasicOutputCollector;

import backtype.storm.topology.OutputFieldsDeclarer;

import backtype.storm.topology.base.BaseBasicBolt;

import backtype.storm.tuple.Fields;

import backtype.storm.tuple.Tuple;

import backtype.storm.tuple.Values;

/\*\*

\*

\* 功能说明：

\* 将一行文本切割成单词，并封装collector中发射出去

\*

\* @author 毛祥溢

\* Email:frank@maoxiangyi.cn

\* 2013-8-26 下午6:05:59

\*/

public class WordNormalizer extends BaseBasicBolt {

public void cleanup() {

System.out.println("将一行文本切割成单词，并封装collector中发射出去 ---完毕！");

}

/\*\*

\* 接受的参数是WordReader发出的句子，即input的内容是句子

\* execute方法，将句子切割形成的单词发出

\*/

public void execute(Tuple input, BasicOutputCollector collector) {

String sentence = input.getString(0);

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

System.out.println("WordNormalizer类 收到一条数据，这条数据是： "+ sentence);

for(String word : words){

word = word.trim();

if(!word.isEmpty()){

word = word.toLowerCase();

System.out.println("WordNormalizer类 收到一条数据，这条数据是： "+ sentence+"数据正在被切割，切割出来的单词是 "+ word);

collector.emit(new Values(word));

}

}

}

/\*\*

\* 定义字段id，该id在简单模式下没有用处，但在按照字段分组的模式下有很大的用处。

\* 该declarer变量有很大作用，我们还可以调用declarer.declareStream();来定义stramId，该id可以用来定义更加复杂的流拓扑结构

\*/

public void declareOutputFields(OutputFieldsDeclarer declarer) {

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

}

}

###### 4 WordCounter 计数器

package cn.jd.storm;

import java.util.HashMap;

import java.util.Map;

import backtype.storm.task.TopologyContext;

import backtype.storm.topology.BasicOutputCollector;

import backtype.storm.topology.OutputFieldsDeclarer;

import backtype.storm.topology.base.BaseBasicBolt;

import backtype.storm.tuple.Tuple;

/\*\*

\*

\* 功能说明：

\* 实现计数器的功能，第一次将collector中的元素存放在成员变量counters（Map）中.

\* 如果counters（Map）中已经存在该元素，getValule并对Value进行累加操作。

\*

\* @author 毛祥溢

\* Email:frank@maoxiangyi.cn

\* 2013-8-26 下午6:06:07

\*/

public class WordCounter extends BaseBasicBolt {

private static final long serialVersionUID = 5678586644899822142L;

Integer id;

String name;

//定义Map封装最后的结果

Map<String, Integer> counters;

/\*\*

\* 在spout结束时被调用，将最后的结果显示出来

\*

\* 結果:

\* -- Word Counter [word-counter-2] --

\* really: 1

\* but: 1

\* application: 1

\* is: 2

\* great: 2

\*/

@Override

public void cleanup() {

System.out.println("-- Word Counter ["+name+"-"+id+"] --");

for(Map.Entry<String, Integer> entry : counters.entrySet()){

System.out.println(entry.getKey()+": "+entry.getValue());

}

System.out.println("实现计数器的功能 --完畢！");

}

/\*\*

\* 初始化操作

\*/

@Override

public void prepare(Map stormConf, TopologyContext context) {

this.counters = new HashMap<String, Integer>();

this.name = context.getThisComponentId();

this.id = context.getThisTaskId();

}

public void declareOutputFields(OutputFieldsDeclarer declarer) {}

/\*\*

\* 实现计数器的功能，第一次将collector中的元素存放在成员变量counters（Map）中.

\* 如果counters（Map）中已经存在该元素，getValule并对Value进行累加操作。

\*/

public void execute(Tuple input, BasicOutputCollector collector) {

String str = input.getString(0);

System.out.println("WordCounter 计数器收到单词 "+ str);

if(!counters.containsKey(str)){

counters.put(str, 1);

}else{

Integer c = counters.get(str) + 1;

counters.put(str, c);

}

}

}

###### 5 拓扑流程日志

WordReader类 发射了一条数据：storm ni great he he xi wang

WordReader类 发射了一条数据：test haha heihei very

WordNormalizer类 收到一条数据，这条数据是： storm ni great he he xi wang

WordNormalizer类 收到一条数据，这条数据是： storm ni great he he xi wang数据正在被切割，切割出来的单词是 storm

WordNormalizer类 收到一条数据，这条数据是： storm ni great he he xi wang数据正在被切割，切割出来的单词是 ni

WordNormalizer类 收到一条数据，这条数据是： storm ni great he he xi wang数据正在被切割，切割出来的单词是 great

WordNormalizer类 收到一条数据，这条数据是： storm ni great he he xi wang数据正在被切割，切割出来的单词是 he

WordNormalizer类 收到一条数据，这条数据是： storm ni great he he xi wang数据正在被切割，切割出来的单词是 he

WordNormalizer类 收到一条数据，这条数据是： storm ni great he he xi wang数据正在被切割，切割出来的单词是 xi

WordNormalizer类 收到一条数据，这条数据是： storm ni great he he xi wang数据正在被切割，切割出来的单词是 wang

WordCounter 计数器收到单词 storm

WordCounter 计数器收到单词 ni

WordCounter 计数器收到单词 great

WordCounter 计数器收到单词 he

WordCounter 计数器收到单词 he

WordCounter 计数器收到单词 xi

WordCounter 计数器收到单词 wang

WordNormalizer类 收到一条数据，这条数据是： test haha heihei very

WordNormalizer类 收到一条数据，这条数据是： test haha heihei very 数据正在被切割，切割出来的单词是 test

WordNormalizer类 收到一条数据，这条数据是： test haha heihei very 数据正在被切割，切割出来的单词是 haha

WordNormalizer类 收到一条数据，这条数据是： test haha heihei very 数据正在被切割，切割出来的单词是 heihei

WordNormalizer类 收到一条数据，这条数据是： test haha heihei very 数据正在被切割，切割出来的单词是 very

WordCounter 计数器收到单词 test

WordCounter 计数器收到单词 haha

WordCounter 计数器收到单词 heihei

WordCounter 计数器收到单词 very

WordReader类 发射了一条数据：are mao xiang yi jd

WordNormalizer类 收到一条数据，这条数据是： are mao xiang yi jd

WordNormalizer类 收到一条数据，这条数据是： are mao xiang yi jd 数据正在被切割，切割出来的单词是 are

WordNormalizer类 收到一条数据，这条数据是： are mao xiang yi jd 数据正在被切割，切割出来的单词是 mao

WordNormalizer类 收到一条数据，这条数据是： are mao xiang yi jd 数据正在被切割，切割出来的单词是 xiang

WordNormalizer类 收到一条数据，这条数据是： are mao xiang yi jd 数据正在被切割，切割出来的单词是 yi

WordNormalizer类 收到一条数据，这条数据是： are mao xiang yi jd 数据正在被切割，切割出来的单词是 jd

WordCounter 计数器收到单词 are

WordCounter 计数器收到单词 mao

WordCounter 计数器收到单词 xiang

WordCounter 计数器收到单词 yi

WordCounter 计数器收到单词 jd

-- Word Counter [word-counter-2] --

xi: 1

test: 1

heihei: 1

haha: 1

he: 2

storm: 1

wang: 1

jd: 1

xiang: 1

great: 1

are: 1

ni: 1

yi: 1

very: 1

mao: 1

实现计数器的功能 --完畢！

将一行文本切割成单词，并封装collector中发射出去 ---完毕！

以上数据来源于：<http://www.maoxiangyi.cn/>

#### 第五章 Strom实战

##### 5.1 生产环境搭建实战

在storm生产环境中，我们有6台物理机，处理器24核，内存是64G。6台物理机的序号分别是1至6，选取序号为1的物理机部署nimbus，其它5台物理机作为supervisor。每台supervisor上配置22个worker，配置22个work是期望每个核心能够处理一个work。端口号从6700到6721，每个worker的分配2G的内存，共110个worker。

storm程序依赖zookeeper，独立部署3台zookeeper的节点。

storm运行的数据源一般是消息队列，如kafka、metaq等。我们选择了淘宝开源的metaq，metaq是kafka的java实现，性能和吞吐量要高于kafka。选择metaq的另外一个因素是有足够多的中文文档。Metaq的搭建相对简单，依次指定brokerId，配置上zookeeper的地址即可，该套zookeeper与storm公用。Metaq是7个物理节点的集群，每个物理节点上部署一个metaq实例，metaq的集群数量没有经过科学的评估。目前每个metaq的topic设置的默认分片数是10，接入一个topic，会被打散到70个分片之上。

storm计算的中间结果会存放在redis中，redis集群由6台物理机组成，每台物理机64G，部署14个redis实例，每个实例4G，占用56G内存。集群上通过slave of搭建主从，实际可用空间168G可用，集群容量报警在130G，主机上无持久化操作，持久化操作在从机上（更新频繁，一致性要求比较高，AOF策略为主。更新不频繁，可以容忍少量数据丢失或错误，Snapshot策略为主）。在存储数据时使用了jedis客户端自带的sharding实现，经过hash之后不同的数据被分散打到42个分片上。

redis每秒支持10次Key的读写，目前暂未出现性能瓶颈。Redis比较吃内存，在生产环境中，建议内存容量最好是数据数据的4倍。计算单个对象的占用的内存公司如下：

键值个数 \* (dictEntry大小 + redisObject大小 + 包含key的sds大小 + 包含value的sds大小) + bucket字节数。在32位系统：dicEntry 16字节 ，reidsObject 16字节 ，key 或value +9 最接近的2的N次方个字节，bucket数\*4 字节。在64位系统：dicEntry 24字节 ，reidsObject 24字节 ，key 或value +9 最接近的2的N次方个字节，bucket数\*8字节。

##### 5.2 订单金额实时汇总

##### 5.3 点击流日志实时分析

##### 5.4 常见错误