

大数据技术之 Storm

版本：V1.0

一 Storm 概述

1.1 离线计算是什么？

离线计算：批量获取数据、批量传输数据、**周期性**批量计算数据、数据展示

代表技术：Sqoop 批量导入数据、HDFS 批量存储数据、MapReduce 批量计算数据、Hive 批量计算数据

1.2 流式计算是什么

流式计算：数据实时产生、数据实时传输、数据实时计算、实时展示

代表技术：Flume 实时获取数据、Kafka 实时数据存储、Storm/JStorm 实时数据计算、Redis 实时结果缓存、持久化存储(mysql)。

离线计算与实时计算最大的区别：实时收集、实时计算、实时展示

1.3 Storm 是什么？

Storm 是一个分布式计算框架，主要使用 Clojure 与 Java 语言编写，最初是由 Nathan Marz 带领 Backtype 公司团队创建，在 Backtype 公司被 Twitter 公司收购后进行开源。最初的版本是在 2011 年 9 月 17 日发行，版本号 0.5.0。

2013 年 9 月，Apache 基金会开始接管并孵化 Storm 项目。Apache Storm 是在 Eclipse Public License 下进行开发的，它提供给大多数企业使用。经过 1 年多时间，2014 年 9 月，Storm 项目成为 Apache 的顶级项目。目前，Storm 的最新版本 1.1.0。

Storm 是一个免费开源的分布式实时计算系统。Storm 能轻松可靠地处理无界的数据流，就像 Hadoop 对数据进行批处理；

1.4 Storm 与 Hadoop 的区别

- 1) Storm 用于实时计算，Hadoop 用于离线计算。
- 2) Storm 处理的数据保存在内存中，源源不断；Hadoop 处理的数据保存在文件系统中，一批一批处理。
- 3) Storm 的数据通过网络传输进来；Hadoop 的数据保存在磁盘中。
- 4) Storm 与 Hadoop 的编程模型相似

	Storm	hadoop
角色	Nimbus	JobTracker
	Supervisor	TaskTracker
	Worker	Child
应用名称	Topology	Job
编程接口	Spout/Bolt	Mapper/Reducer

(1) hadoop 相关名称

Job: 任务名称

JobTracker: 项目经理 (**JobTracker 对应于 NameNode**; JobTracker 是一个 master 服务, 软件启动之后 JobTracker 接收 Job, 负责调度 Job 的每一个子任务 task 运行于 TaskTracker 上, 并监控它们, 如果发现有失败的 task 就重新运行它。)

TaskTracker: 开发组长 (**TaskTracker 对应于 DataNode**; TaskTracker 是运行在多个节点上的 slaver 服务。TaskTracker 主动与 JobTracker 通信, 接收作业, 并负责直接执行每一个任务。)

Child: 负责开发的人员

Mapper/Reduce: 开发人员中的两种角色, 一种是服务器开发、一种是客户端开发

(2) storm 相关名称

Topology: 任务名称

Nimbus: 项目经理

Supervisor: 开组长

Worker: 开人员

Spout/Bolt: 开人员中的两种角色, 一种是服务器开发、一种是客户端开发

1.5 Storm 应用场景及行业案例

Storm 用来实时计算源源不断产生的数据, 如同流水线生产。

1.5.1 运用场景

Storm 能用到很多场景中, 包括: 实时分析、在线机器学习、连续计算等。

- 1) 推荐系统: 实时推荐, 根据下单或加入购物车推荐相关商品
- 2) 金融系统: 实时分析股票信息数据
- 3) 预警系统: 根据实时采集数据, 判断是否到了预警阈值。

4) 网站统计：实时销量、流量统计，如淘宝双 11 效果图

1.5.2 典型案例

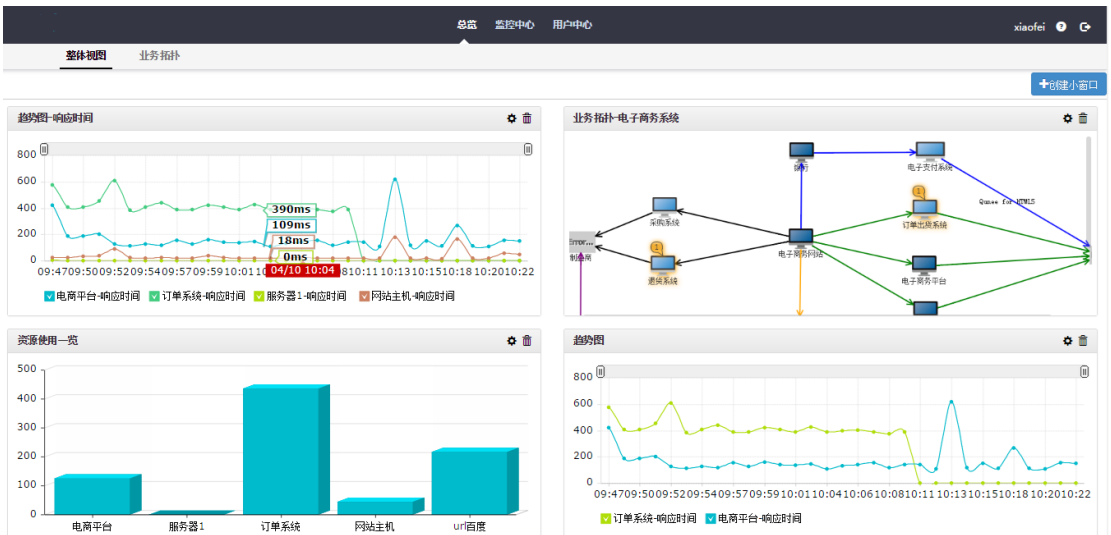
1) 京东-实时分析系统：实时分析用户的属性，并反馈给搜索引擎

最初，用户属性分析是通过每天在云上定时运行的 MR job 来完成的。为了满足实时性的要求，希望能够实时分析用户的行为日志，将最新的用户属性反馈给搜索引擎，能够为用户展现最贴近其当前需求的结果。



2) 携程-网站性能监控：实时分析系统监控携程网的网站性能

利用 HTML5 提供的 performance 标准获得可用的指标，并记录日志。Storm 集群实时分析日志和入库。使用 DRPC 聚合成报表，通过历史数据对比等判断规则，触发预警事件。



3) 淘宝双十一：实时统计销售总额

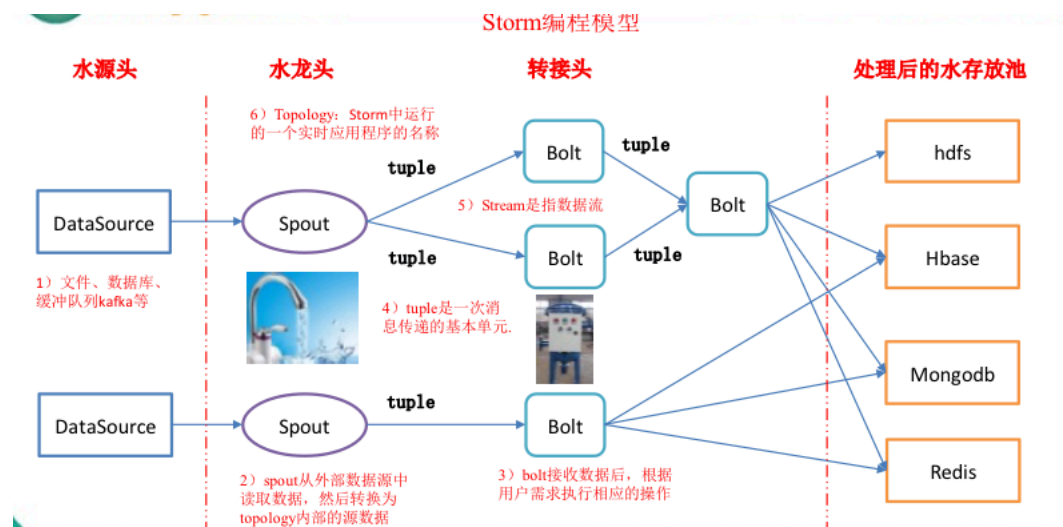


1.6 Storm 特点

- 1) 适用场景广泛: Storm 可以适用实时处理消息、更新数据库、持续计算等场景。
- 2) 可伸缩性高: Storm 的可伸缩性可以让 Storm 每秒处理的消息量达到很高。扩展一个实时计算任务,你所需要做的就是加机器并且提高这个计算任务的并行度。Storm 使用 Zookeeper 来协调机器内的各种配置使得 Storm 的集群可以很容易的扩展。
- 3) 保证无数据丢失: Storm 保证所有的数据都被处理。
- 4) 异常健壮: Storm 集群非常容易管理,轮流重启节点不影响应用。
- 5) 容错性好: 在消息处理过程中出现异常, Storm 会进行重试。

二 Storm 基础知识

2.1 Storm 编程模型



2.1.1 元组 (Tuple)

元组 (Tuple)，是消息传递的基本单元，是一个命名的值列表，元组中的字段可以是任何类型的对象。Storm 使用元组作为其数据模型，元组支持所有的基本类型、字符串和字节数组作为字段值，只要实现类型的序列化接口就可以使用该类型的对象。元组本来应该是一个 key-value 的 Map，但是由于各个组件间传递的元组的字段名称已经事先定义好，所以只要按序把元组填入各个 value 即可，所以元组是一个 value 的 List。

2.1.2 流 (Stream)

流是 Storm 的核心抽象，是一个无界的元组系列。源源不断传递的元组就组成了流，在分布式环境中并行地进行创建和处理。

2.1.3 水龙头 (Spout)

Spout 是拓扑的流的来源，是一个拓扑中产生源数据流的组件。通常情况下，Spout 会从外部数据源中读取数据，然后转换为拓扑内部的源数据。

Spout 可以是可靠的，也可以是不可靠的。如果 Storm 处理元组失败，可靠的 Spout 能够重新发射，而不可靠的 Spout 就尽快忘记发出的元组。

Spout 可以发出超过一个流。

Spout 的主要方法是 nextTuple()。NextTuple()会发出一个新的 Tuple 到拓扑，如果没有新的元组发出，则简单返回。

Spout 的其他方法是 ack()和 fail()。当 Storm 检测到一个元组从 Spout 发出时，ack()和 fail()会被调用，要么成功完成通过拓扑，要么未能完成。Ack()和 fail()仅被可靠的 Spout 调用。

IRichSpout 是 Spout 必须实现的接口。

2.1.4 转接头 (Bolt)

在拓扑中所有处理都在 Bolt 中完成，Bolt 是流的处理节点，从一个拓扑接收数据，然后执行进行处理的组件。Bolt 可以完成过滤、业务处理、连接运算、连接与访问数据库等任何操作。

Bolt 是一个被动的角色，七接口中有一个 execute()方法，在接收到消息后会调用此方法，用户可以在其中执行自己希望的操作。

Bolt 可以完成简单的流的转换，而完成复杂的流的转换通常需要多个步骤，因此需要多个 Bolt。

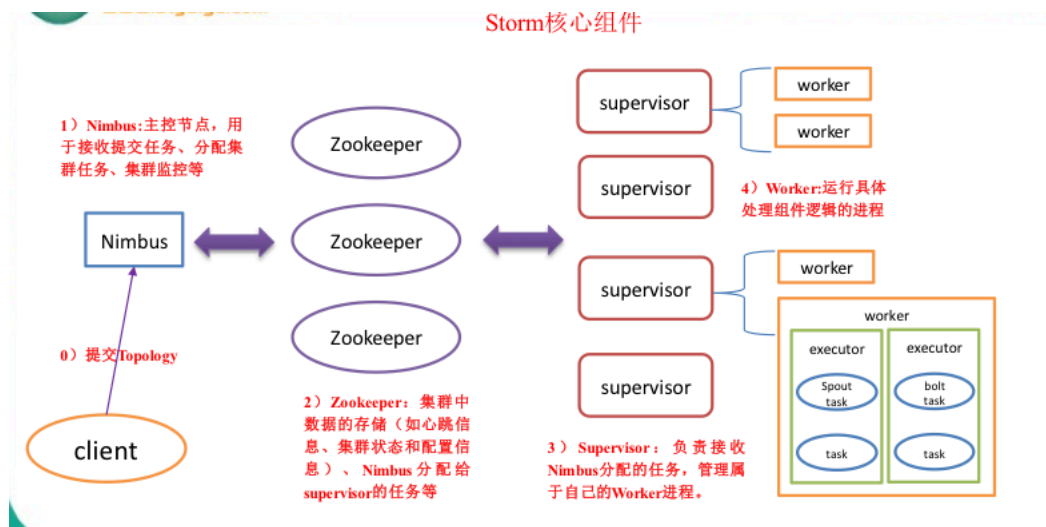
Bolt 可以发出超过一个的流。

2.1.5 拓扑（Topology）

拓扑（Topology）是 Storm 中运行的一个实时应用程序，因为各个组件间的消息流动而形成逻辑上的拓扑结构。

把实时应用程序的运行逻辑打成 jar 包后提交到 Storm 的拓扑（Topology）。Storm 的拓扑类似于 MapReduce 的作业（Job）。其主要的区别是，MapReduce 的作业最终会完成，而一个拓扑永远都在运行直到它被杀死。一个拓扑是一个图的 Spout 和 Bolt 的连接流分组。

2.2 Storm 核心组件



nimbus 是整个集群的控管核心，负责 topology 的提交、运行状态监控、任务重新分配等工作。

zk 就是一个管理者，监控者。

总体描述：nimbus 下命令（分配任务），zk 监督执行（心跳监控，worker、supervisor 的心跳都归它管），supervisor 领旨（下载代码），招募人马（创建 worker 和线程等），worker、executor 就给我干活！task 就是具体要干的活。

2.2.1 主控节点与工作节点

Storm 集群中有两类节点：主控节点（Master Node）和工作节点（Worker Node）。其中，主控节点只有一个，而工作节点可以有多个。

2.2.2 Nimbus 进程与 Supervisor 进程

主控节点运行一个称为 Nimbus 的守护进程类似于 Hadoop 的 JobTracker。Nimbus 负责

在集群中分发代码，对节点分配任务，并监视主机故障。

每个工作节点运行一个称为 Supervisor 的守护进程。Supervisor 监听其主机上已经分配的主机的作业，启动和停止 Nimbus 已经分配的工作进程。

2.2.3 流分组（Stream grouping）

流分组，是拓扑定义中的一部分，为每个 Bolt 指定应该接收哪个流作为输入。流分组定义了流/元组如何在 Bolt 的任务之间进行分发。

Storm 内置了 8 种流分组方式。

2.2.4 工作进程（Worker）

Worker 是 Spout/Bolt 中运行具体处理逻辑的进程。一个 worker 就是一个进程，进程里面包含一个或多个线程。

2.2.5 执行器（Executor）

一个线程就是一个 executor，一个线程会处理一个或多个任务。

2.2.6 任务（Task）

一个任务就是一个 task。

2.3 实时流计算常见架构图



- 1) Flume 获取数据。
- 2) Kafka 临时保存数据。
- 3) Storm 计算数据。
- 4) Redis 是个内存数据库，用来保存数据。

三 Storm 集群搭建

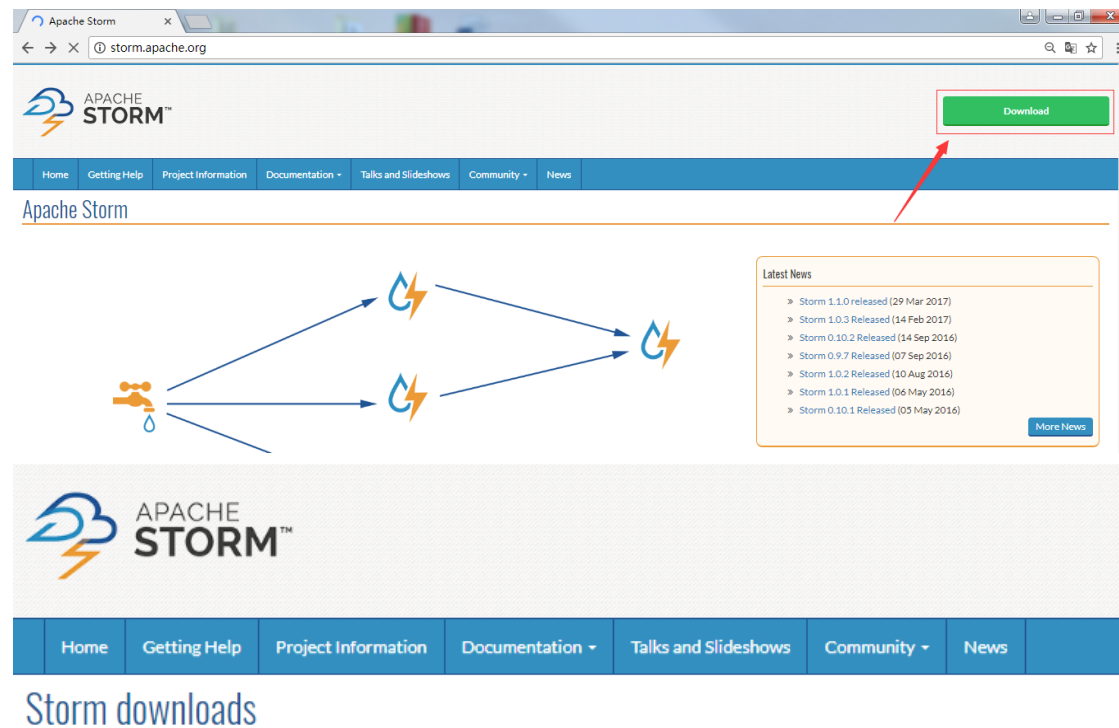
3.1 环境准备

3.1.1 集群规划

hadoop102	hadoop103	hadoop104
zk	zk	zk
storm	storm	storm

3.1.2 jar 包下载

(1) 官方网址: <http://storm.apache.org/>



Downloads for Apache Storm are below. Instructions for how to set up a Storm cluster can be found [here](#).

Source Code

Current source code is mirrored on GitHub: [apache/storm](https://github.com/apache/storm)

Current 1.1.x Release

The current 1.1.x release is 1.1.0. Source and binary distributions can be found below. The list of changes for this release can be found [here](#).

[Documentation](#)

[Javadocs](#)

- [apache-storm-1.1.0.tar.gz](#) [PGP] [SHA512] [MD5]
- [apache-storm-1.1.0.zip](#) [PGP] [SHA512] [MD5]
- [apache-storm-1.1.0-src.tar.gz](#) [PGP] [SHA512] [MD5]
- [apache-storm-1.1.0-src.zip](#) [PGP] [SHA512] [MD5]

(2) 安装集群步骤:

<http://storm.apache.org/releases/1.1.0/Setting-up-a-Storm-cluster.html>

3.1.3 虚拟机准备

1) 准备 3 台虚拟机

2) 配置 ip 地址



尚硅谷大数据技术
之修改为静态ip.do

3) 配置主机名称



尚硅谷大数据技术
之修改主机名.doc

4) 3 台主机分别关闭防火墙

```
[root@hadoop102 atguigu]# chkconfig iptables off
```

```
[root@hadoop103 atguigu]# chkconfig iptables off
```

```
[root@hadoop104 atguigu]# chkconfig iptables off
```

3.1.4 安装 jdk



尚硅谷大数据技术
之安装jdk.doc

3.1.5 安装 Zookeeper

0) 集群规划

在 hadoop102、hadoop103 和 hadoop104 三个节点上部署 Zookeeper。

1) 解压安装

(1) 解压 zookeeper 安装包到/opt/module/目录下

```
[atguigu@hadoop102 software]$ tar -zxvf zookeeper-3.4.10.tar.gz -C /opt/module/
```

(2) 在/opt/module/zookeeper-3.4.10/这个目录下创建 zkData

```
mkdir -p zkData
```

(3) 重命名/opt/module/zookeeper-3.4.10/conf 这个目录下的 zoo_sample.cfg 为 zoo.cfg

```
mv zoo_sample.cfg zoo.cfg
```

2) 配置 zoo.cfg 文件

(1) 具体配置

```
dataDir=/opt/module/zookeeper-3.4.10/zkData
```

增加如下配置

```
#####cluster#####
```

```
server.2=hadoop102:2888:3888
```

```
server.3=hadoop103:2888:3888
```

```
server.4=hadoop104:2888:3888
```

(2) 配置参数解读

Server.A=B:C:D。

A 是一个数字，表示这个是第几号服务器；

B 是这个服务器的 ip 地址；

C 是这个服务器与集群中的 Leader 服务器交换信息的端口；

D 是万一集群中的 Leader 服务器挂了，需要一个端口来重新进行选举，选出一个新的 Leader，而这个端口就是用来执行选举时服务器相互通信的端口。

集群模式下配置一个文件 myid，这个文件在 dataDir 目录下，这个文件里面有一个数据就是 A 的值，Zookeeper 启动时读取此文件，拿到里面的数据与 zoo.cfg 里面的配置信息比较从而判断到底是哪个 server。

3) 集群操作

- (1) 在/opt/module/zookeeper-3.4.10/zkData 目录下创建一个 myid 的文件

```
touch myid
```

添加 myid 文件，注意一定要在 linux 里面创建，在 notepad++ 里面很可能乱码

- (2) 编辑 myid 文件

```
vi myid
```

在文件中添加与 server 对应的编号：如 2

- (3) 拷贝配置好的 zookeeper 到其他机器上

```
scp -r zookeeper-3.4.10/ root@hadoop103.atguigu.com:/opt/app/
```

```
scp -r zookeeper-3.4.10/ root@hadoop104.atguigu.com:/opt/app/
```

并分别修改 myid 文件中内容为 3、4

- (4) 分别启动 zookeeper

```
[root@hadoop102 zookeeper-3.4.10]# bin/zkServer.sh start
```

```
[root@hadoop103 zookeeper-3.4.10]# bin/zkServer.sh start
```

```
[root@hadoop104 zookeeper-3.4.10]# bin/zkServer.sh start
```

- (5) 查看状态

```
[root@hadoop102 zookeeper-3.4.10]# bin/zkServer.sh status
```

JMX enabled by default

Using config: /opt/module/zookeeper-3.4.10/bin/../conf/zoo.cfg

Mode: follower

```
[root@hadoop103 zookeeper-3.4.10]# bin/zkServer.sh status
```

JMX enabled by default

Using config: /opt/module/zookeeper-3.4.10/bin/../conf/zoo.cfg

Mode: **leader**

```
[root@hadoop104 zookeeper-3.4.5]# bin/zkServer.sh status
```

JMX enabled by default

Using config: /opt/module/zookeeper-3.4.10/bin/../conf/zoo.cfg

Mode: **follower**

3.2 Storm 集群部署

3.2.1 配置集群

1) 拷贝 jar 包到 hadoop102 的/opt/software 目录下

2) 解压 jar 包到/opt/module 目录下

```
[atguigu@hadoop102 software]$ tar -zxvf apache-storm-1.1.0.tar.gz -C /opt/module/
```

3) 修改解压后的 apache-storm-1.1.0.tar.gz 文件名称为 storm

```
[atguigu@hadoop102 module]$ mv apache-storm-1.1.0/ storm
```

4) 在/opt/module/storm/目录下创建 data 文件夹

```
[atguigu@hadoop102 storm]$ mkdir data
```

5) 修改配置文件

```
[atguigu@hadoop102 conf]$ pwd
```

/opt/module/storm/conf

```
[atguigu@hadoop102 conf]$ vi storm.yaml
```

```
# 设置 Zookeeper 的主机名称
storm.zookeeper.servers:
  - "hadoop102"
  - "hadoop103"
  - "hadoop104"

# 设置主节点的主机名称
nimbus.seeds: ["hadoop102"]

# 设置 Storm 的数据存储路径
storm.local.dir: "/opt/module/storm/data"

# 设置 Worker 的端口号
```

```
supervisor.slots.ports:
```

```
- 6700  
- 6701  
- 6702  
- 6703
```

6) 配置环境变量

```
[root@hadoop102 storm]# vi /etc/profile
```

```
#STORM_HOME  
export STORM_HOME=/opt/module/storm  
export PATH=$PATH:$STORM_HOME/bin
```

```
[root@hadoop102 storm]# source /etc/profile
```

7) 分发配置好的 Storm 安装包

```
[atguigu@hadoop102 storm]$ xsync storm/
```

8) 启动集群

(1) 后台启动 nimbus

```
[atguigu@hadoop102 storm]$ bin/storm nimbus &
```

```
[atguigu@hadoop103 storm]$ bin/storm nimbus &
```

```
[atguigu@hadoop104 storm]$ bin/storm nimbus &
```

(2) 后台启动 supervisor

```
[atguigu@hadoop102 storm]$ bin/storm supervisor &
```

```
[atguigu@hadoop102 storm]$ bin/storm supervisor &
```

```
[atguigu@hadoop102 storm]$ bin/storm supervisor &
```

(3) 启动 Storm ui

```
[atguigu@hadoop102 storm]$ bin/storm ui
```

9) 通过浏览器查看集群状态

<http://hadoop102:8080/index.html>

The screenshot shows the Storm UI web interface. The browser address bar indicates the URL is `hadoop102:8080/index.html`. The interface is divided into four main sections:

- Cluster Summary:** A table with columns: Version, Supervisors, Used slots, Free slots. The data row shows: 1.0.3, 1, 0, 4.
- Nimbus Summary:** A table with columns: Host, Port, Status. The data row shows: hadoop102, 6627, Leader. Below the table, it says "Showing 1 to 1 of 1 entries".
- Topology Summary:** A table with columns: Name, Owner, Status, Uptime, Num workers, Num executors, Num tasks. The data row shows: No data available in table. Below the table, it says "Showing 0 to 0 of 0 entries".
- Supervisor Summary:** A table with columns: Host, Id, Uptime. The data row shows: hadoop102, 2b08e76f-24c2-4733-92d8-6f7059aa6e30, 2m 24s. Below the table, it says "Showing 1 to 1 of 1 entries".

3.2.2 Storm 日志信息查看

1) 查看 nimbus 的日志信息

在 nimbus 的服务器上

```
cd /opt/module/storm/logs
```

```
tail -100f /opt/module/storm/logs/nimbus.log
```

2) 查看 ui 运行日志信息

在 ui 的服务器上，一般和 nimbus 一个服务器

```
cd /opt/module/storm/logs
```

```
tail -100f /opt/module/storm/logs/ui.log
```

3) 查看 supervisor 运行日志信息

在 supervisor 服务上

```
cd /opt/module/storm/logs
```

```
tail -100f /opt/module/storm/logs/supervisor.log
```

4) 查看 supervisor 上 worker 运行日志信息

在 supervisor 服务上

```
cd /opt/module/storm/logs
```

```
tail -100f /opt/module/storm/logs/worker-6702.log
```

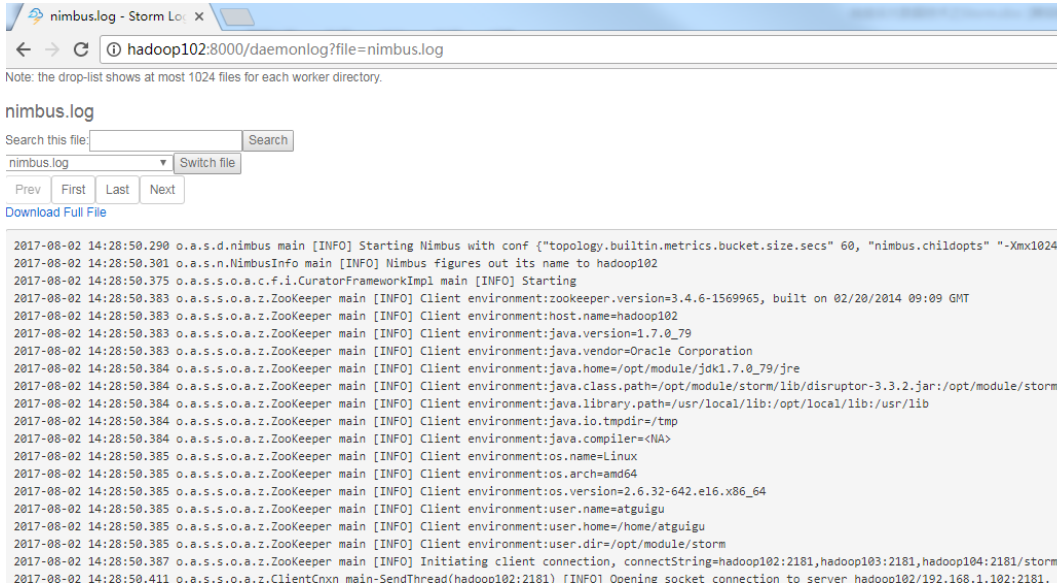
5) logviewer，可以在 web 页面点击相应的端口号即可查看日志

分别在 supervisor 节点上执行：

```
[atguigu@hadoop102 storm]$ bin/storm logviewer &
```

```
[atguigu@hadoop103 storm]$ bin/storm logviewer &
```

```
[atguigu@hadoop104 storm]$ bin/storm logviewer &
```



3.2.3 Storm 命令行操作

1) nimbus: 启动 nimbus 守护进程

```
storm nimbus
```

2) supervisor: 启动 supervisor 守护进程

```
storm supervisor
```

3) ui: 启动 UI 守护进程。

```
storm ui
```

4) list: 列出正在运行的拓扑及其状态

```
storm list
```

5) logviewer: Logviewer 提供一个 web 接口查看 Storm 日志文件。

```
storm logviewer
```

6) jar:

```
storm jar 【jar 路径】 【拓扑包名.拓扑类名】 【拓扑名称】
```

7) kill: 杀死名为 Topology-name 的拓扑

```
storm kill topology-name [-w wait-time-secs]
```

-w: 等待多久后杀死拓扑

8) active: 激活指定的拓扑 spout。

```
storm activate topology-name
```

9) deactivate: 禁用指定的拓扑 Spout。

```
storm deactivate topology-name
```

10) help: 打印一条帮助消息或者可用命令的列表。

```
storm help
```

```
storm help <command>
```

四 常用 API

4.1 API 简介

4.1.1 Component 组件

1) 基本接口

(1) IComponent 接口

(2) ISpout 接口

(3) IRichSpout 接口

(4) IStateSpout 接口

(5) IRichStateSpout 接口

(6) IBolt 接口

(7) IRichBolt 接口

(8) IBasicBolt 接口

2) 基本抽象类

(1) BaseComponent 抽象类

(2) BaseRichSpout 抽象类

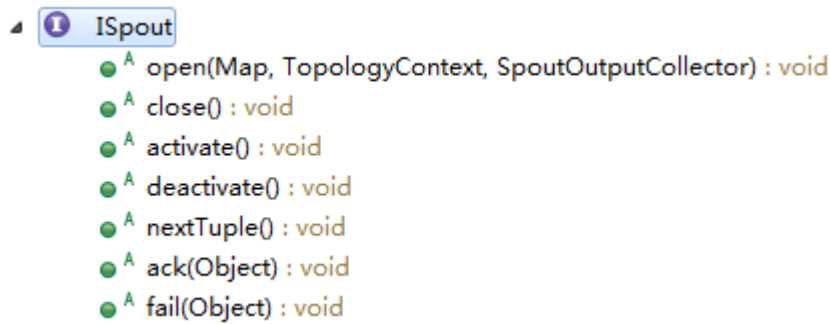
(3) BaseRichBolt 抽象类

(4) BaseTransactionalBolt 抽象类

(5) BaseBasicBolt 抽象类

4.1.2 spout 水龙头

Spout 的最顶层抽象是 ISpout 接口



(1) Open()

是初始化方法

(2) close()

在该 spout 关闭前执行，但是并不能得到保证其一定被执行，kill -9 时不执行，Storm kill {topoName} 时执行

(3) activate()

当 Spout 已经从失效模式中激活时被调用。该 Spout 的 nextTuple()方法很快就会被调用。

(4) deactivate ()

当 Spout 已经失效时被调用。在 Spout 失效期间，nextTuple 不会被调用。Spout 将来可能会也可能不会被重新激活。

(5) nextTuple()

当调用 nextTuple()方法时，Storm 要求 Spout 发射元组到输出收集器(OutputCollector)。NextTuple 方法应该是非阻塞的，所以，如果 Spout 没有元组可以发射，该方法应该返回。nextTuple()、ack()和 fail()方法都在 Spout 任务的单一线程内紧密循环被调用。当没有元组可以发射时，可以让 nextTuple 去 sleep 很短的时间，例如 1 毫秒，这样就不会浪费太多的 CPU 资源。

(6) ack()

成功处理 tuple 回调方法

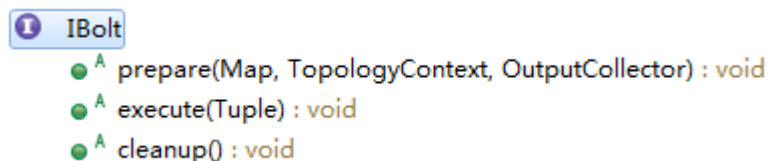
(7) fail()

处理失败 tuple 回调方法

原则：通常情况下（Shell 和事务型的除外），实现一个 Spout，可以直接实现接口 **IRichSpout**，如果不想写多余的代码，可以直接继承 **BaseRichSpout**。

4.1.3 bolt 转接头

bolt 的最顶层抽象是 IBolt 接口



```
IBolt
  A prepare(Map, TopologyContext, OutputCollector) : void
  A execute(Tuple) : void
  A cleanup() : void
```

(1) prepare()

prepare ()方法在集群的工作进程内被初始化时被调用，提供了 Bolt 执行所需要的环境。

(2) execute()

接受一个 tuple 进行处理，也可 emit 数据到下一级组件。

(3) cleanup()

Cleanup 方法当一个 IBolt 即将关闭时被调用。不能保证 cleanup()方法一定会被调用，因为 Supervisor 可以对集群的工作进程使用 kill -9 命令强制杀死进程命令。

如果在本地模式下运行 Storm，当拓扑被杀死的时候，可以保证 cleanup()方法一定会被调用。

实现一个 Bolt，可以实现 **IRichBolt** 接口或继承 **BaseRichBolt**，如果不想自己处理结果反馈，可以实现 **IBasicBolt** 接口或继承 **BaseBasicBolt**，它实际上相当于自动做了 prepare 方法和 collector.emit.ack(inputTuple)。

4.1.4 spout 的 tail 特性

Storm 可以实时监测文件数据，当文件数据变化时，Storm 自动读取。

4.2 网站日志处理案例

4.2.1 实操环境准备

- 1) 打开 eclipse，创建一个 java 工程
- 2) 在工程目录中创建 lib 文件夹
- 3) 解压 apache-storm-1.1.0，并把解压后 lib 包下的文件复制到 java 工程的 lib 文件夹中。然后执行 build path。

4.2.2 需求 1：将接收到日志的会话 id 打印在控制台

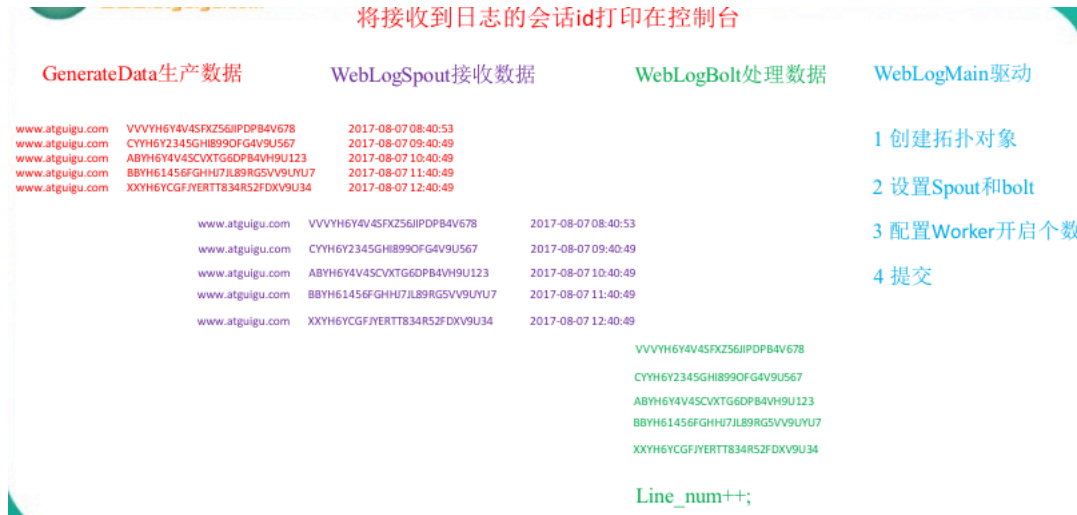
1) 需求：

- (1) 模拟访问网站的日志信息，包括：网站名称、会话 id、访问网站时间等
- (2) 将接收到日志的会话 id 打印到控制台

2) 分析

- (1) 创建网站访问日志工具类

- (2) 在 spout 中读取日志文件，并一行一行发射出去
- (3) 在 bolt 中将获取到的一行一行数据的会话 id 获取到，并打印到控制台。
- (4) main 方法负责拼接 spout 和 bolt 的拓扑。



3) 案例实操

- (1) 创建网站访问日志

```
package com.atguigu.storm.weblog;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.Random;

// 生成数据
public class GenerateData {

    public static void main(String[] args) {
        File logFile = new File("e:/website.log");
        Random random = new Random();

        // 1 网站名称
        String[] hosts = { "www.atguigu.com" };
        // 2 会话 id
        String[] session_id = { "ABYH6Y4V4SCVXTG6DPB4VH9U123",
                                "XXYH6YCGFYERTT834R52FDXV9U34",
                                "BBYH61456FGHHJ7JL89RG5VV9UYU7",
                                "CYYH6Y2345GHI899OFG4V9U567", "VVVYH6Y4V4SFXZ56JIPDPB4V678" };
        // 3 访问网站时间
        String[] time = { "2017-08-07 08:40:50", "2017-08-07 08:40:51", "2017-08-07
```

```

08:40:52", "2017-08-07 08:40:53",
        "2017-08-07 09:40:49", "2017-08-07 10:40:49", "2017-08-07 11:40:49",
"2017-08-07 12:40:49" };

    // 4 拼接网站访问日志
    StringBuffer sbBuffer = new StringBuffer();
    for (int i = 0; i < 40; i++) {
        sbBuffer.append(hosts[0] + "\t" + session_id[random.nextInt(5)] + "\t" +
time[random.nextInt(8)] + "\n");
    }

    // 5 判断 log 日志是否存在，不存在要创建
    if (!logFile.exists()) {
        try {
            logFile.createNewFile();
        } catch (IOException e) {
            System.out.println("Create logFile fail !");
        }
    }
    byte[] b = (sbBuffer.toString()).getBytes();

    // 6 将拼接的日志信息写到日志文件中
    FileOutputStream fs;
    try {
        fs = new FileOutputStream(logFile);
        fs.write(b);
        fs.close();
        System.out.println("generate data over");
    } catch (Exception e) {
        e.printStackTrace();
    }
}
}

```

(2) 创建 spout

```

package com.atguigu.storm.weblog;
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.InputStreamReader;
import java.util.Map;
import org.apache.storm.spout.SpoutOutputCollector;
import org.apache.storm.task.TopologyContext;
import org.apache.storm.topology.IRichSpout;
import org.apache.storm.topology.OutputFieldsDeclarer;
import org.apache.storm.tuple.Fields;

```

```

import org.apache.storm.tuple.Values;

public class WebLogSpout implements IRichSpout{

    private static final long serialVersionUID = 1L;

    private BufferedReader br;
    private SpoutOutputCollector collector = null;
    private String str = null;

    @Override
    public void nextTuple() {
        // 循环调用的方法
        try {
            while ((str = this.br.readLine()) != null) {
                // 发射出去
                collector.emit(new Values(str));

                Thread.sleep(3000);
            }
        } catch (Exception e) {

        }
    }

    @SuppressWarnings("rawtypes")
    @Override
    public void open(Map conf, TopologyContext context, SpoutOutputCollector collector) {
        // 打开输入的文件
        try {
            this.collector = collector;
            this.br = new BufferedReader(new InputStreamReader(new
FileInputStream("e:/website.log"), "UTF-8"));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    @Override
    public void declareOutputFields(OutputFieldsDeclarer declarer) {
        // 声明输出字段类型
        declarer.declare(new Fields("log"));
    }

    @Override

```

```

    public void ack(Object arg0) {

    }

    @Override
    public void activate() {

    }

    @Override
    public void close() {

    }

    @Override
    public void deactivate() {

    }

    @Override
    public void fail(Object arg0) {

    }

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

```

(3) 创建 bolt

```

package com.atguigu.storm.weblog;
import java.util.Map;
import org.apache.storm.task.OutputCollector;
import org.apache.storm.task.TopologyContext;
import org.apache.storm.topology.IRichBolt;
import org.apache.storm.topology.OutputFieldsDeclarer;
import org.apache.storm.tuple.Fields;
import org.apache.storm.tuple.Tuple;

public class WebLogBolt implements IRichBolt {

    private static final long serialVersionUID = 1L;
    private OutputCollector collector = null;
    private int num = 0;
    private String valueString = null;

    @Override
    public void execute(Tuple input) {
        try {

```

```

        // 1 获取传递过来的数据
        valueString = input.getStringByField("log");

        // 2 如果输入的数据不为空，行数++
        if (valueString != null) {
            num++;
            System.err.println(Thread.currentThread().getName() + "lines  :" + num
+ "    session_id:" + valueString.split("\t")[1]);
        }

        // 3 应答 Spout 接收成功
        collector.ack(input);

        Thread.sleep(2000);
    } catch (Exception e) {
        // 4 应答 Spout 接收失败
        collector.fail(input);

        e.printStackTrace();
    }
}

@SuppressWarnings("rawtypes")
@Override
public void prepare(Map conf, TopologyContext context, OutputCollector collector) {
    this.collector = collector;
}

@Override
public void declareOutputFields(OutputFieldsDeclarer declarer) {
    // 声明输出字段类型
    declarer.declare(new Fields(""));
}

@Override
public void cleanup() {

}

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

```

（4）创建 main

```
package com.atguigu.storm.weblog;
import org.apache.storm.Config;
import org.apache.storm.LocalCluster;
import org.apache.storm.StormSubmitter;
import org.apache.storm.topology.TopologyBuilder;

public class WebLogMain {

    public static void main(String[] args) {
        // 1 创建拓扑对象
        TopologyBuilder builder = new TopologyBuilder();

        // 2 设置 Spout 和 bolt
        builder.setSpout("weblogspout", new WebLogSpout(), 1);
        builder.setBolt("weblogbolt", new WebLogBolt(),
1).shuffleGrouping("weblogspout");

        // 3 配置 Worker 开启个数
        Config conf = new Config();
        conf.setNumWorkers(4);

        if (args.length > 0) {
            try {
                // 4 分布式提交
                StormSubmitter.submitTopology(args[0], conf, builder.createTopology());
            } catch (Exception e) {
                e.printStackTrace();
            }
        } else {
            // 5 本地模式提交
            LocalCluster localCluster = new LocalCluster();
            localCluster.submitTopology("weblogtopology", conf,
builder.createTopology());
        }
    }
}
```

4.2.3 需求 2：动态增加日志，查看控制台打印信息（tail 特性）

- 1) 在需求 1 基础上，运行程序。
- 2) 打开 website.log 日志文件，增加日志调试并保存。
- 3) 观察控制台打印的信息。

结论：Storm 可以动态实时监测文件的增加信息，并把信息读取到再处理。

五 分组策略和并发度

5.1 读取文件案例思考

- 1) spout 数据源：数据库、文件、MQ（比如：Kafka）
- 2) 数据源是数据库：只适合读取数据库的配置文件
- 3) 数据源是文件：只适合测试、讲课用（因为集群是分布式集群）
- 4) 企业产生的 log 文件处理步骤：
 - （1）读出内容写入 MQ
 - （2）Storm 再处理

5.2 分组策略（Stream Grouping）

stream grouping 用来定义一个 stream 应该如何分配给 Bolts 上面的多个 Executors（多线程、多并发）。

Storm 里面有 7 种类型的 stream grouping

- 1) **Shuffle Grouping: 随机分组**，轮询，平均分配。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。
- 2) **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。
- 3) **All Grouping: 广播发送**，对于每一个 tuple，所有的 bolts 都会收到。
- 4) **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。
- 5) **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。在多线程情况下不平均分配。**
- 6) **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）。
- 7) **Local or shuffle grouping**：如果目标 bolt 有一个或者多个 task 在同个工作进程中，tuple

将会被随机发送给这些 tasks。否则，和普通的 Shuffle Grouping 行为一致。

8) 测试

(1) spout 并发度修改为 2，bolt 并发度修改为 1，shuffleGrouping 模式

```
builder.setSpout("WebLogSpout", new WebLogSpout(),2);
```

```
builder.setBolt("WebLogBolt", new WebLogBolt(), 1).shuffleGrouping("WebLogSpout");
```

Spout 开两个线程会对数据读取两份，打印出来就是 2 份。如果数据源是消息队列，就不会出来读取两份的数据（统一消费者组，只能有一个消费者）

```
Thread-33-WebLogBolt-executor[1 1]lines:60 session_id:CYYH6Y2345GHI899OFG4V9U567
```

(2) spout 并发度修改为 1，bolt 并发度修改为 2，noneGrouping 模式

```
builder.setSpout("WebLogSpout", new WebLogSpout(),1);
```

```
builder.setBolt("WebLogBolt", new WebLogBolt(), 2).noneGrouping("WebLogSpout");
```

每个 bolt 接收到的单词不同。

```
Thread-33-WebLogBolt-executor[1 1]lines:14 session_id:VVVYH6Y4V4SFXZ56JIPDPB4V678
```

```
Thread-34-WebLogBolt-executor[2 2]lines:16 session_id:VVVYH6Y4V4SFXZ56JIPDPB4V678
```

(3) spout 并发度修改为 1，bolt 并发度修改为 2，fieldsGrouping

```
builder.setSpout("WebLogSpout", new WebLogSpout(),1);
```

```
builder.setBolt("WebLogBolt", new WebLogBolt(), 2).fieldsGrouping("WebLogSpout", new Fields("log"));
```

基于 web 案例不明显，后续案例比较明显

(4) spout 并发度修改为 1，bolt 并发度修改为 2，allGrouping("spout");

```
builder.setSpout("WebLogSpout", new WebLogSpout(),1);
```

```
builder.setBolt("WebLogBolt", new WebLogBolt(), 2).allGrouping("WebLogSpout");
```

每一个 bolt 获取到的数据都是一样的。

```
Thread-43-WebLogBolt-executor[1 1]lines:30 session_id:VVVYH6Y4V4SFXZ56JIPDPB4V678
```

```
Thread-23-WebLogBolt-executor[2 2]lines:30 session_id:VVVYH6Y4V4SFXZ56JIPDPB4V678
```

(5) spout 并发度修改为 1，bolt 并发度修改为 2，globalGrouping("spout");

```
builder.setSpout("WebLogSpout", new WebLogSpout(),1);
```

```
builder.setBolt("WebLogBolt", new WebLogBolt(), 2).globalGrouping("WebLogSpout");
```

Task 的 id 最低的 bolt 获取到了所有数据。

5.3 并发度

5.3.1 场景分析

- 1) 单线程下：加减乘除、全局**汇总**
- 2) 多线程下：**局部**加减乘除、持久化 DB 等

(1) 思考：如何计算：word 总数和 word 个数？并且在高并发下完成

前者是统计总行数，后者是去重 word 个数；

类似企业场景：计算网站 PV 和 UV

(2) 网站最常用的两个指标：

PV(page views): count(session_id) 即页面浏览量。

UV(user views): count(distinct session_id) 即独立访客数。

a) 用 ip 地址分析

指访问某个站点或点击某个网页的不同 IP 地址的人数。在同一天内，UV 只记录第一次进入网站的具有独立 IP 的访问者，在同一天内再次访问该网站则不计数。

b) 用 Cookie 分析 UV 值

当客户端第一次访问某个网站服务器的时候，网站服务器会给这个客户端的电脑发出一个 Cookie，通常放在这个客户端电脑的 C 盘当中。在这个 Cookie 中会分配一个独一无二的编号，这其中会记录一些访问服务器的信息，如访问时间，访问了哪些页面等等。当你下次再访问这个服务器的时候，服务器就可以直接从你的电脑中找到上一次放进去的 Cookie 文件，并且对其进行一些更新，但那个独一无二的编号是不会变的。

实时处理的业务场景主要包括：汇总型（如网站 PV、销售额、订单数）、去重型（网站 UV、顾客数、销售商品数）

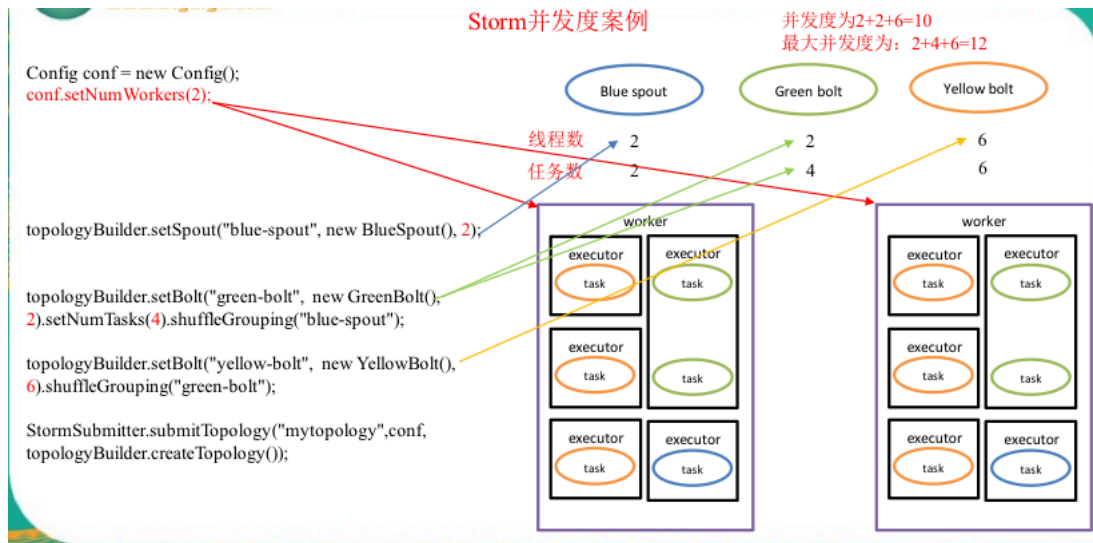
5.3.2 并发度

并发度：用户指定一个任务，可以被多个线程执行，**并发度的数量等于线程 executor 的数量**。

Task 就是具体的处理逻辑对象，一个 executor 线程可以执行一个或多个 tasks，但默认每个 executor 只执行一个 task，所以我们往往认为 task 就是执行线程，其实不是。

Task 代表最大并发度，一个 component 的 task 数是不会改变的，但是一个 componet 的 executor 数目是会变化的（storm rebalance 命令），task 数>=executor 数，**executor 数代**

表实际并发数。



5.4 实操案例

5.4.1 实时单词统计案例

1) 需求

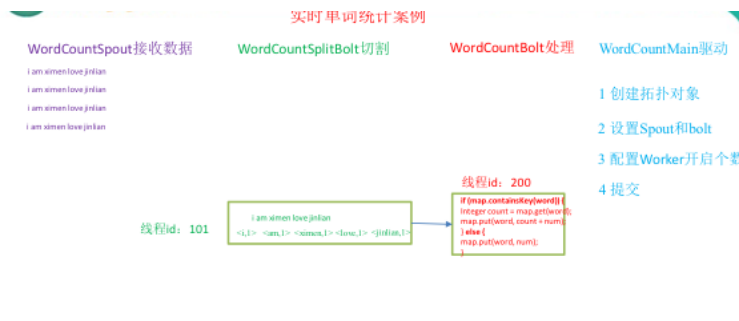
实时统计发射到 Storm 框架中单词的总数。

2) 分析

设计一个 topology，来实现对文档里面的单词出现的频率进行统计。

整个 topology 分为三个部分：

- (1) WordCountSpout: 数据源，在已知的英文句子中，随机发送一条句子出去。
- (2) WordCountSplitBolt: 负责将单行文本记录（句子）切分成单词
- (3) WordCountBolt: 负责对单词的频率进行累加



3) 实操

(1) 创建 spout

```
package com.atguigu.storm.wordcount;
```

```

import java.util.Map;
import org.apache.storm.spout.SpoutOutputCollector;
import org.apache.storm.task.TopologyContext;
import org.apache.storm.topology.OutputFieldsDeclarer;
import org.apache.storm.topology.base.BaseRichSpout;
import org.apache.storm.tuple.Fields;
import org.apache.storm.tuple.Values;

public class WordCountSpout extends BaseRichSpout {

    private static final long serialVersionUID = 1L;
    private SpoutOutputCollector collector;

    @Override
    public void nextTuple() {
        // 1 发射模拟数据
        collector.emit(new Values("i am ximen love jinlian"));

        // 2 睡眠 2 秒
        try {
            Thread.sleep(2000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    @SuppressWarnings("rawtypes")
    @Override
    public void open(Map arg0, TopologyContext arg1, SpoutOutputCollector collector) {
        this.collector = collector;
    }

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

```

(2) 创建切割单词的 bolt

```

package com.atguigu.storm.wordcount;
import java.util.Map;
import org.apache.storm.task.OutputCollector;
import org.apache.storm.task.TopologyContext;
import org.apache.storm.topology.OutputFieldsDeclarer;
import org.apache.storm.topology.base.BaseRichBolt;

```

```

import org.apache.storm.tuple.Fields;
import org.apache.storm.tuple.Tuple;
import org.apache.storm.tuple.Values;

public class WordCountSplitBolt extends BaseRichBolt {

    private static final long serialVersionUID = 1L;
    private OutputCollector collector;

    @Override
    public void execute(Tuple input) {
        // 1 获取传递过来的一行数据
        String line = input.getString(0);
        // 2 截取
        String[] arrWords = line.split(" ");

        // 3 发射
        for (String word : arrWords) {
            collector.emit(new Values(word, 1));
        }
    }

    @SuppressWarnings("rawtypes")
    @Override
    public void prepare(Map arg0, TopologyContext arg1, OutputCollector collector) {
        this.collector = collector;
    }

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

```

(3) 创建汇总单词个数的 bolt

```

package com.atguigu.storm.wordcount;

import java.util.HashMap;
import java.util.Map;
import org.apache.storm.task.OutputCollector;
import org.apache.storm.task.TopologyContext;
import org.apache.storm.topology.OutputFieldsDeclarer;
import org.apache.storm.topology.base.BaseRichBolt;
import org.apache.storm.tuple.Tuple;

public class WordCountBolt extends BaseRichBolt {

```

```

private static final long serialVersionUID = 1L;
private Map<String, Integer> map = new HashMap<String, Integer>();

@Override
public void execute(Tuple input) {

    // 1 获取传递过来的数据
    String word = input.getString(0);
    Integer num = input.getInteger(1);

    // 2 累加单词
    if (map.containsKey(word)) {
        Integer count = map.get(word);
        map.put(word, count + num);
    } else {
        map.put(word, num);
    }

    System.err.println(Thread.currentThread().getId() + "    word:" + word + "    num:" +
map.get(word));
}

@SuppressWarnings("rawtypes")
@Override
public void prepare(Map arg0, TopologyContext arg1, OutputCollector collector) {

}

@Override
public void declareOutputFields(OutputFieldsDeclarer arg0) {
    // 不输出
}
}

```

(4) 创建程序的拓扑 main

```

package com.atguigu.storm.wordcount;
import org.apache.storm.Config;
import org.apache.storm.LocalCluster;
import org.apache.storm.StormSubmitter;
import org.apache.storm.topology.TopologyBuilder;
import org.apache.storm.tuple.Fields;

public class WordCountMain {

```

```

public static void main(String[] args) {
    // 1、准备一个 TopologyBuilder
    TopologyBuilder builder = new TopologyBuilder();

    builder.setSpout("WordCountSpout", new WordCountSpout(), 1);
    builder.setBolt("WordCountSplitBolt", new WordCountSplitBolt(),
2).shuffleGrouping("WordCountSpout");
    builder.setBolt("WordCountBolt", new WordCountBolt(),
4).fieldsGrouping("WordCountSplitBolt", new Fields("word"));

    // 2、创建一个 configuration，用来指定当前 topology 需要的 worker 的数量
    Config conf = new Config();
    conf.setNumWorkers(2);

    // 3、提交任务 -----两种模式 本地模式和集群模式
    if (args.length > 0) {
        try {
            // 4 分布式提交
            StormSubmitter.submitTopology(args[0], conf, builder.createTopology());
        } catch (Exception e) {
            e.printStackTrace();
        }
    } else {
        // 5 本地模式提交
        LocalCluster localCluster = new LocalCluster();
        localCluster.submitTopology("wordcounttopology", conf,
builder.createTopology());
    }
}
}

```

(5) 测试

发现 132 线程只处理单词 am 和单词 love;169 进程处理单词 i、ximen、jianlian。这就是分组的好处。

```

132 word:am num:1

132 word:love num:1

169 word:i num:1

169 word:ximen num:1

169 word:jinlian num:1

```

5.4.2 实时计算网站 PV 案例

0) 基础知识准备

1) 需求

统计网站 pv。

2) 需求分析

方案一：

定义 `static long pv`, `Synchronized` 控制累计操作。（不可行）

原因：`Synchronized` 和 `Lock` 在单 JVM 下有效，但在多 JVM 下无效

方案二：

`shuffleGrouping` 下，`pv * Executor` 并发数

驱动函数中配置如下：

```
builder.setSpout("PVSpout", new PVSpout(), 1);
```

```
builder.setBolt("PVBolt1", new PVBolt1(), 4).shuffleGrouping("PVSpout");
```

在 `PVBolt1` 中输出时

```
System.err.println("threadid:" + Thread.currentThread().getId() + " pv:" + pv*4);
```

因为 `shuffleGrouping` 轮询分配

优点：简单、计算量小

缺点：稍有误差，但绝大多数场景能接受

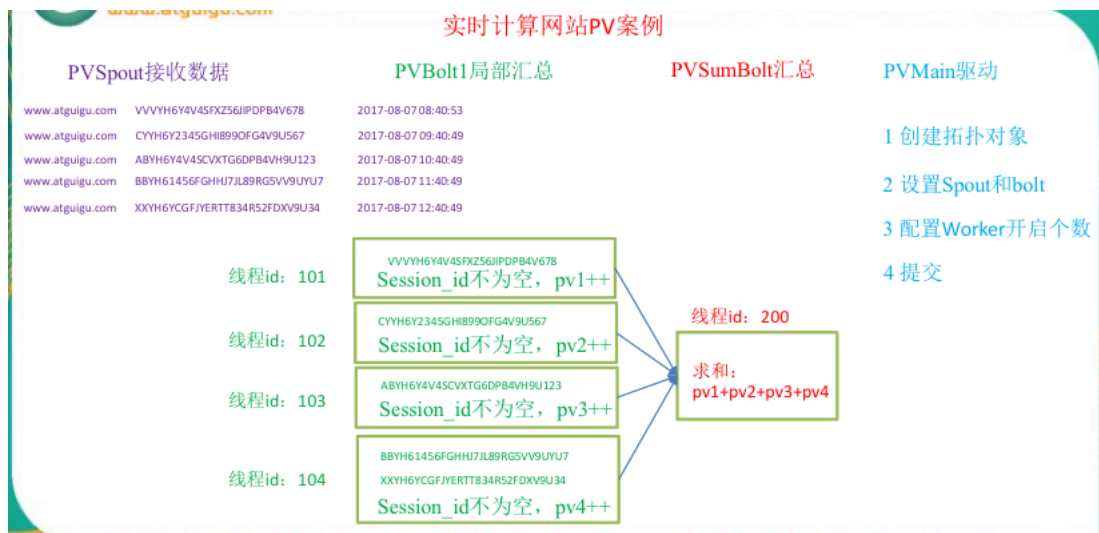
方案三：

`PVBolt1` 进行多并发局部汇总，`PVSumbolt` 单线程进行全局汇总

线程安全：多线程处理的结果和单线程一致

优点：绝对准确；如果用 `filedGrouping` 可以得到中间值，如单个 user 的访问 PV（访问深度等）

缺点：计算量稍大，且多一个 Bolt



3) 案例实操

(1) 创建数据输入源 PVSpout

```
package com.atguigu.storm.pv;
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Map;

import org.apache.storm.spout.SpoutOutputCollector;
import org.apache.storm.task.TopologyContext;
import org.apache.storm.topology.IRichSpout;
import org.apache.storm.topology.OutputFieldsDeclarer;
import org.apache.storm.tuple.Fields;
import org.apache.storm.tuple.Values;

public class PVSpout implements IRichSpout{

    private static final long serialVersionUID = 1L;
    private SpoutOutputCollector collector ;
    private BufferedReader reader;

    @SuppressWarnings("rawtypes")
    @Override
    public void open(Map conf, TopologyContext context, SpoutOutputCollector collector) {
        this.collector = collector;

        try {
            reader = new BufferedReader(new InputStreamReader(new
```

```
FileInputStream("e:/website.log"),"UTF-8"));

        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    @Override
    public void close() {

        try {
            if (reader != null) {
                reader.close();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void activate() {

    }

    @Override
    public void deactivate() {

    }

    private String str;

    @Override
    public void nextTuple() {

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

                collector.emit(new Values(str));

                Thread.sleep(500);
            }
        } catch (Exception e) {

        }
    }
}
```

```

    }

    @Override
    public void ack(Object msgId) {
    }

    @Override
    public void fail(Object msgId) {

    }

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

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

```

(2) 创建数据处理 pvbolt1

```

package com.atguigu.storm.pv;

import java.util.Map;

import org.apache.storm.task.OutputCollector;
import org.apache.storm.task.TopologyContext;
import org.apache.storm.topology.IRichBolt;
import org.apache.storm.topology.OutputFieldsDeclarer;
import org.apache.storm.tuple.Fields;
import org.apache.storm.tuple.Tuple;
import org.apache.storm.tuple.Values;

public class PVBolt1 implements IRichBolt {

    private static final long serialVersionUID = 1L;
    private OutputCollector collector;
    private long pv = 0;

    @SuppressWarnings("rawtypes")

```

```

@Override
public void prepare(Map stormConf, TopologyContext context, OutputCollector
collector) {
    this.collector = collector;
}

@Override
public void execute(Tuple input) {
    // 获取传递过来的数据
    String logline = input.getString(0);

    // 截取出 sessionid
    String session_id = logline.split("\t")[1];

    // 根据会话 id 不同统计 pv 次数
    if (session_id != null) {
        pv++;
    }

    // 提交
    collector.emit(new Values(Thread.currentThread().getId(), pv));

    System.err.println("threadid:" + Thread.currentThread().getId() + "    pv:" + pv);
}

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

@Override
public void cleanup() {
}

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

```

(3) 创建 PVSumBolt

```

package com.atguigu.storm.pv;
import java.util.HashMap;

```

```

import java.util.Iterator;
import java.util.Map;
import org.apache.storm.task.OutputCollector;
import org.apache.storm.task.TopologyContext;
import org.apache.storm.topology.IRichBolt;
import org.apache.storm.topology.OutputFieldsDeclarer;
import org.apache.storm.tuple.Tuple;

public class PVSumBolt implements IRichBolt {

    private static final long serialVersionUID = 1L;
    private Map<Long, Long> counts = new HashMap<>();

    @SuppressWarnings("rawtypes")
    @Override
    public void prepare(Map stormConf, TopologyContext context, OutputCollector
collector) {

    }

    @Override
    public void execute(Tuple input) {
        Long threadID = input.getLong(0);
        Long pv = input.getLong(1);

        counts.put(threadID, pv);

        long word_sum = 0;

        Iterator<Long> iterator = counts.values().iterator();

        while (iterator.hasNext()) {
            word_sum += iterator.next();
        }

        System.err.println("pv_all:" + word_sum);
    }

    @Override
    public void cleanup() {
    }

    @Override
    public void declareOutputFields(OutputFieldsDeclarer declarer) {

```

```

    }

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

```

(4) 驱动

```

package com.atguigu.storm.pv;
import org.apache.storm.Config;
import org.apache.storm.LocalCluster;
import org.apache.storm.StormSubmitter;
import org.apache.storm.topology.TopologyBuilder;

public class PVMain {

    public static void main(String[] args) {
        TopologyBuilder builder = new TopologyBuilder();

        builder.setSpout("PVSpout", new PVSpout(), 1);
        builder.setBolt("PVBolt1", new PVBolt1(), 4).shuffleGrouping("PVSpout");
        builder.setBolt("PVSumBolt", new PVSumBolt(), 1).shuffleGrouping("PVBolt1");

        Config conf = new Config();

        conf.setNumWorkers(2);

        if (args.length > 0) {
            try {
                StormSubmitter.submitTopology(args[0], conf, builder.createTopology());
            } catch (Exception e) {
                e.printStackTrace();
            }
        } else {
            LocalCluster cluster = new LocalCluster();
            cluster.submitTopology("pvtopology", conf, builder.createTopology());
        }
    }
}

```

(5) 测试，执行程序输出如下结果

```

threadid:161  pv:1
pv_all:1

```

threadid:164 pv:1
pv_all:2
threadid:161 pv:2
pv_all:3
threadid:172 pv:1
pv_all:4
threadid:164 pv:2
pv_all:5
threadid:164 pv:3
pv_all:6
threadid:162 pv:1
pv_all:7
threadid:161 pv:3
pv_all:8
threadid:172 pv:2
pv_all:9
threadid:164 pv:4
pv_all:10
threadid:162 pv:2
pv_all:11
threadid:172 pv:3
pv_all:12
threadid:164 pv:5
pv_all:13
threadid:164 pv:6
pv_all:14
threadid:161 pv:4
pv_all:15
threadid:161 pv:5
pv_all:16
threadid:164 pv:7
pv_all:17
threadid:172 pv:4
pv_all:18
threadid:172 pv:5
pv_all:19
threadid:161 pv:6
pv_all:20
threadid:162 pv:3
pv_all:21
threadid:164 pv:8
pv_all:22
threadid:172 pv:6
pv_all:23

```

threadid:164  pv:9
pv_all:24
threadid:161  pv:7
pv_all:25
threadid:162  pv:4
pv_all:26
threadid:162  pv:5
pv_all:27
threadid:162  pv:6
pv_all:28
threadid:164  pv:10
pv_all:29
threadid:161  pv:8
pv_all:30

```

5.4.3 实时计算网站 UV 去重案例

1) 需求:

统计网站 UV。

2) 需求分析

方案一:

把 ip 放入 Set 实现自动去重, Set.size() 获得 UV (分布式应用中不可行)

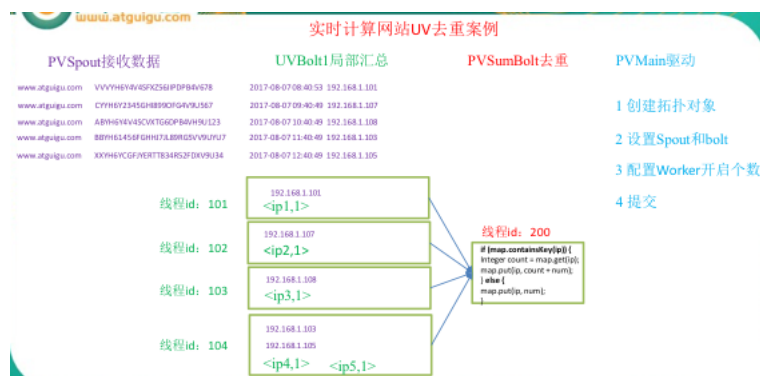
方案二:

UVBolt1 通过 fieldGrouping 进行多线程局部汇总, 下一级 UVSumBolt 进行单线程全局汇总去重。按 ip 地址统计 UV 数。

既然去重, 必须持久化数据:

(1) 内存: 数据结构 map

(2) no-sql 分布式数据库, 如 Hbase



3) 案例实操

(1) 创建带 ip 地址的数据源 GenerateData

```
package com.atguigu.storm.uv;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.Random;

public class GenerateData {

    public static void main(String[] args) {
        // 创建文件
        File logFile = new File("e:/website.log");
        // 判断文件是否存在
        if (!logFile.exists()) {
            try {
                logFile.createNewFile();
            } catch (IOException e) {

                e.printStackTrace();
            }
        }

        Random random = new Random();

        // 1 网站名称
        String[] hosts = { "www.atguigu.com" };
        // 2 会话 id
        String[] session_id = { "ABYH6Y4V4SCVXTG6DPB4VH9U123",
            "XXYH6YCGFJYERTT834R52FDXV9U34",
            "BBYH61456FGHHJ7JL89RG5VV9UYU7",
            "CYYH6Y2345GHI899OFG4V9U567", "VVVYH6Y4V4SFXZ56JIPDPB4V678" };
        // 3 访问网站时间
        String[] time = { "2017-08-07 08:40:50", "2017-08-07 08:40:51", "2017-08-07
08:40:52", "2017-08-07 08:40:53",
            "2017-08-07 09:40:49", "2017-08-07 10:40:49", "2017-08-07 11:40:49",
            "2017-08-07 12:40:49" };
        // 3 访问网站时间
        String[] ip = { "192.168.1.101", "192.168.1.102", "192.168.1.103", "192.168.1.104",
            "192.168.1.105",
            "192.168.1.106", "192.168.1.107", "192.168.1.108" };

        StringBuffer sb = new StringBuffer();

        for (int i = 0; i < 30; i++) {
```

```

        sb.append(hosts[0] + "\t" + session_id[random.nextInt(5)] + "\t" +
time[random.nextInt(8)] + "\t"
        + ip[random.nextInt(8)] + "\n");
    }

    // 写数据
    try {
        FileOutputStream fs = new FileOutputStream(logFile);

        fs.write(sb.toString().getBytes());
        fs.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
}

```

（2）创建接收数据 UVSpout

```

package com.atguigu.storm.uv;
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Map;
import org.apache.storm.spout.SpoutOutputCollector;
import org.apache.storm.task.TopologyContext;
import org.apache.storm.topology.IRichSpout;
import org.apache.storm.topology.OutputFieldsDeclarer;
import org.apache.storm.tuple.Fields;
import org.apache.storm.tuple.Values;

public class UVSpout implements IRichSpout{

    private static final long serialVersionUID = 1L;
    private SpoutOutputCollector collector ;
    private BufferedReader reader;

    @SuppressWarnings("rawtypes")
    @Override
    public void open(Map conf, TopologyContext context, SpoutOutputCollector collector) {
        this.collector = collector;

        try {
            reader = new BufferedReader(new InputStreamReader(new
FileInputStream("e:/website.log"),"UTF-8"));

```

```

        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    @Override
    public void close() {

        try {
            if (reader != null) {
                reader.close();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void activate() {

    }

    @Override
    public void deactivate() {

    }

    private String str;

    @Override
    public void nextTuple() {

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

                collector.emit(new Values(str));

                Thread.sleep(500);
            }
        } catch (Exception e) {

        }
    }
}

```

```

@Override
public void ack(Object msgId) {
}

@Override
public void fail(Object msgId) {

}

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

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

```

(3) 创建 UVBolt1

```

package com.atguigu.storm.uv;

import java.util.Map;

import org.apache.storm.task.OutputCollector;
import org.apache.storm.task.TopologyContext;
import org.apache.storm.topology.OutputFieldsDeclarer;
import org.apache.storm.topology.base.BaseRichBolt;
import org.apache.storm.tuple.Fields;
import org.apache.storm.tuple.Tuple;
import org.apache.storm.tuple.Values;

public class UVBolt1 extends BaseRichBolt {
    private static final long serialVersionUID = 1L;
    private OutputCollector collector;

    @SuppressWarnings("rawtypes")
    @Override
    public void prepare(Map stormConf, TopologyContext context, OutputCollector collector) {
        this.collector = collector;
    }
}

```

```

@Override
public void execute(Tuple input) {
    // 1 获取传递过来的一行数据
    String line = input.getString(0);

    // 2 截取
    String[] splits = line.split("\t");
    String ip = splits[3];

    // 3 发射
    collector.emit(new Values(ip, 1));
}

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

```

(4) 创建 UVSumBolt

```

package com.atguigu.storm.uv;

import java.util.HashMap;
import java.util.Map;

import org.apache.storm.task.OutputCollector;
import org.apache.storm.task.TopologyContext;
import org.apache.storm.topology.OutputFieldsDeclarer;
import org.apache.storm.topology.base.BaseRichBolt;
import org.apache.storm.tuple.Tuple;

public class UVSumBolt extends BaseRichBolt {
    private static final long serialVersionUID = 1L;
    private Map<String, Integer> map = new HashMap<String, Integer>();

    @SuppressWarnings("rawtypes")
    @Override
    public void prepare(Map stormConf, TopologyContext context, OutputCollector collector) {

    }

    @Override
    public void execute(Tuple input) {
        // 1 获取传递过来的数据
    }
}

```

```

String ip = input.getString(0);
Integer num = input.getInteger(1);

// 2 累加单词
if (map.containsKey(ip)) {
    Integer count = map.get(ip);
    map.put(ip, count + num);
} else {
    map.put(ip, num);
}

System.err.println(Thread.currentThread().getId() + " ip:" + ip + " num:" +
map.get(ip));
}

@Override
public void declareOutputFields(OutputFieldsDeclarer declarer) {

}
}

```

(5) 创建 UVMain 驱动

```

package com.atguigu.storm.uv;
import org.apache.storm.Config;
import org.apache.storm.LocalCluster;
import org.apache.storm.StormSubmitter;
import org.apache.storm.topology.TopologyBuilder;

public class UVMain {

    public static void main(String[] args) {

        TopologyBuilder builder = new TopologyBuilder();

        builder.setSpout("UVSpout", new UVSpout(),1);
        builder.setBolt("UVBolt1", new UVBolt1(),4).shuffleGrouping("UVSpout");
        builder.setBolt("UVSumBolt", new UVSumBolt(), 1).shuffleGrouping("UVBolt1");
        Config conf = new Config();
        conf.setNumWorkers(2);

        if (args.length > 0) {
            try {
                StormSubmitter.submitTopology(args[0], conf, builder.createTopology());
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
}

```

```

    }
    }else {
        LocalCluster cluster = new LocalCluster();

        cluster.submitTopology("uvtopology", conf, builder.createTopology());
    }
}
}

```

(6) 测试

```

136 ip:192.168.1.101 num:1
136 ip:192.168.1.107 num:1
136 ip:192.168.1.108 num:1
136 ip:192.168.1.103 num:1
136 ip:192.168.1.105 num:1
136 ip:192.168.1.102 num:1
136 ip:192.168.1.103 num:2
136 ip:192.168.1.101 num:2
136 ip:192.168.1.107 num:2
136 ip:192.168.1.105 num:2
136 ip:192.168.1.101 num:3
136 ip:192.168.1.108 num:2
136 ip:192.168.1.108 num:3
136 ip:192.168.1.105 num:3
136 ip:192.168.1.101 num:4
136 ip:192.168.1.108 num:4
136 ip:192.168.1.102 num:2
136 ip:192.168.1.104 num:1
136 ip:192.168.1.103 num:3
136 ip:192.168.1.106 num:1
136 ip:192.168.1.105 num:4
136 ip:192.168.1.101 num:5
136 ip:192.168.1.102 num:3
136 ip:192.168.1.108 num:5
136 ip:192.168.1.103 num:4
136 ip:192.168.1.107 num:3
136 ip:192.168.1.102 num:4
136 ip:192.168.1.104 num:2
136 ip:192.168.1.105 num:5
136 ip:192.168.1.104 num:3

```

测试结果：一共 8 个用户，101：访问 5 次；102 访问：4 次；103：访问 4 次；104：访问 3 次；105：访问 5 次；106：访问 1 次；107：访问 3 次；108：访问 5 次；

