目录

**[一、前言：](#_Toc25235_WPSOffice_Level1)** **[4](#_Toc25235_WPSOffice_Level1)**

**[二、项目说明：](#_Toc20402_WPSOffice_Level1)** **[4](#_Toc20402_WPSOffice_Level1)**

[架构图](#_Toc20402_WPSOffice_Level2) [4](#_Toc20402_WPSOffice_Level2)

[流程图](#_Toc26418_WPSOffice_Level2) [6](#_Toc26418_WPSOffice_Level2)

**[三、程序运行环境：](#_Toc26418_WPSOffice_Level1)** **[6](#_Toc26418_WPSOffice_Level1)**

**[四、实现过程:](#_Toc4937_WPSOffice_Level1)** **[7](#_Toc4937_WPSOffice_Level1)**

**[五、集群配置过程：](#_Toc11951_WPSOffice_Level1)** **[7](#_Toc11951_WPSOffice_Level1)**

[1.修改CentOS默认yum源为mirrors.163.com](#_Toc4937_WPSOffice_Level2) [7](#_Toc4937_WPSOffice_Level2)

[2.分别配置hostname、ip、配置hosts文件](#_Toc11951_WPSOffice_Level2) [7](#_Toc11951_WPSOffice_Level2)

[3.分别ssh安全配置以及ssh key认证登录](#_Toc22316_WPSOffice_Level2) [9](#_Toc22316_WPSOffice_Level2)

[4.测试免密钥连接：如图成功](#_Toc8343_WPSOffice_Level2) [10](#_Toc8343_WPSOffice_Level2)

**[四、配置java环境变量](#_Toc22316_WPSOffice_Level1)** **[10](#_Toc22316_WPSOffice_Level1)**

**[五、搭建hadoop集群(master,slave1和slave2)](#_Toc8343_WPSOffice_Level1)** **[10](#_Toc8343_WPSOffice_Level1)**

[1.切换进配置文件路径](#_Toc8933_WPSOffice_Level2) [10](#_Toc8933_WPSOffice_Level2)

[2.增加slave节点](#_Toc19131_WPSOffice_Level2) [11](#_Toc19131_WPSOffice_Level2)

[3.配置core-site.xml](#_Toc10151_WPSOffice_Level2) [11](#_Toc10151_WPSOffice_Level2)

[4.配置hdfs-site.xml](#_Toc12680_WPSOffice_Level2) [11](#_Toc12680_WPSOffice_Level2)

[5.创建hdfs需要用的文件目录](#_Toc6184_WPSOffice_Level2) [11](#_Toc6184_WPSOffice_Level2)

[6.初始化hdfs文件系统](#_Toc25836_WPSOffice_Level2) [12](#_Toc25836_WPSOffice_Level2)

[7.启动hdfs文件系统](#_Toc5226_WPSOffice_Level2) [12](#_Toc5226_WPSOffice_Level2)

[8.网址查看：](#_Toc15745_WPSOffice_Level2) [12](#_Toc15745_WPSOffice_Level2)

[9.在hdfs中准备运行目录](#_Toc28850_WPSOffice_Level2) [12](#_Toc28850_WPSOffice_Level2)

**[六、搭建yarn集群](#_Toc8933_WPSOffice_Level1)** **[13](#_Toc8933_WPSOffice_Level1)**

[1.配置yarn-site.xml](#_Toc3121_WPSOffice_Level2) [13](#_Toc3121_WPSOffice_Level2)

[2.配置mapred-site.xml](#_Toc6452_WPSOffice_Level2) [15](#_Toc6452_WPSOffice_Level2)

[3.配置yarn的本地目录](#_Toc6042_WPSOffice_Level2) [17](#_Toc6042_WPSOffice_Level2)

[4.启动yarn服务](#_Toc8266_WPSOffice_Level2) [17](#_Toc8266_WPSOffice_Level2)

[5.验证](#_Toc24067_WPSOffice_Level2) [18](#_Toc24067_WPSOffice_Level2)

**[七、安装配置flume步骤：](#_Toc19131_WPSOffice_Level1)** **[18](#_Toc19131_WPSOffice_Level1)**

[1.1 下载安装包并解压](#_Toc13099_WPSOffice_Level2) [18](#_Toc13099_WPSOffice_Level2)

[1.2 修改 flume-env.sh 配置文件,主要是JAVA\_HOME变量设置](#_Toc1963_WPSOffice_Level2) [18](#_Toc1963_WPSOffice_Level2)

[1.3 配置flume文件](#_Toc20853_WPSOffice_Level2) [19](#_Toc20853_WPSOffice_Level2)

[1.4 启动flume](#_Toc5286_WPSOffice_Level2) [20](#_Toc5286_WPSOffice_Level2)

**[八、Flink及kafka安装部署](#_Toc10151_WPSOffice_Level1)** **[20](#_Toc10151_WPSOffice_Level1)**

[1、简介](#_Toc17941_WPSOffice_Level2) [20](#_Toc17941_WPSOffice_Level2)

[2、Flink的开发准备](#_Toc27315_WPSOffice_Level2) [22](#_Toc27315_WPSOffice_Level2)

[3、集群环境准备](#_Toc5191_WPSOffice_Level2) [23](#_Toc5191_WPSOffice_Level2)

**[九、配置Hbase集群](#_Toc12680_WPSOffice_Level1)** **[26](#_Toc12680_WPSOffice_Level1)**

[1.配置文件hbase-site.xml](#_Toc19116_WPSOffice_Level2) [26](#_Toc19116_WPSOffice_Level2)

[2.配置修改hbase-env.sh](#_Toc25034_WPSOffice_Level2) [28](#_Toc25034_WPSOffice_Level2)

[3.配置修改修改regionservers](#_Toc12923_WPSOffice_Level2) [28](#_Toc12923_WPSOffice_Level2)

[4.准备hbase目录](#_Toc28538_WPSOffice_Level2) [28](#_Toc28538_WPSOffice_Level2)

[5.修改环境变量](#_Toc6076_WPSOffice_Level2) [29](#_Toc6076_WPSOffice_Level2)

[6.启动Hbase集群](#_Toc31721_WPSOffice_Level2) [29](#_Toc31721_WPSOffice_Level2)

**[十、OpenTSDB部署](#_Toc6184_WPSOffice_Level1)** **[30](#_Toc6184_WPSOffice_Level1)**

**[十一、组件连接：](#_Toc25836_WPSOffice_Level1)** **[31](#_Toc25836_WPSOffice_Level1)**

[一、flink连接kafka](#_Toc2166_WPSOffice_Level2) [31](#_Toc2166_WPSOffice_Level2)

[二、flink连接opentsdb](#_Toc1981_WPSOffice_Level2) [33](#_Toc1981_WPSOffice_Level2)

**[十二、打包](#_Toc5226_WPSOffice_Level1)** **[36](#_Toc5226_WPSOffice_Level1)**

[（一） 创建工程之前的准备](#_Toc27281_WPSOffice_Level2) [36](#_Toc27281_WPSOffice_Level2)

[（二） 创建步骤](#_Toc29131_WPSOffice_Level2) [36](#_Toc29131_WPSOffice_Level2)

**[十三、启动服务及运行](#_Toc15745_WPSOffice_Level1)** **[48](#_Toc15745_WPSOffice_Level1)**

[1、 首先启动zookeeper集群](#_Toc6124_WPSOffice_Level2) [48](#_Toc6124_WPSOffice_Level2)

[2、启动kafka集群（3台都要启动）](#_Toc20814_WPSOffice_Level2) [49](#_Toc20814_WPSOffice_Level2)

[3、启动hdfs](#_Toc17477_WPSOffice_Level2) [50](#_Toc17477_WPSOffice_Level2)

[4、 启动Flink集群](#_Toc26293_WPSOffice_Level2) [50](#_Toc26293_WPSOffice_Level2)

[5、 在flink的bin目录下提交计划](#_Toc17239_WPSOffice_Level2) [55](#_Toc17239_WPSOffice_Level2)

[图：Mainclass路径](#_Toc32715_WPSOffice_Level2) [56](#_Toc32715_WPSOffice_Level2)

**[十四、Grafana部署安装](#_Toc28850_WPSOffice_Level1)** **[61](#_Toc28850_WPSOffice_Level1)**

[1. 下载grafana压缩包](#_Toc14544_WPSOffice_Level2) [61](#_Toc14544_WPSOffice_Level2)

[2. 解压到指定目录](#_Toc21074_WPSOffice_Level2) [61](#_Toc21074_WPSOffice_Level2)

[3. 为了使用饼状图，我们需要安装Pie chart插件](#_Toc18702_WPSOffice_Level2) [61](#_Toc18702_WPSOffice_Level2)

[4. 安装日志服务插件](#_Toc10310_WPSOffice_Level2) [61](#_Toc10310_WPSOffice_Level2)

[5. 安装好插件后重启grafana-server](#_Toc22732_WPSOffice_Level2) [61](#_Toc22732_WPSOffice_Level2)

[6. 登录查看](#_Toc21344_WPSOffice_Level2) [61](#_Toc21344_WPSOffice_Level2)

[7. 为grafana配置数据源为opentsdb](#_Toc6944_WPSOffice_Level2) [62](#_Toc6944_WPSOffice_Level2)

[8. 创建个图表,并配置告警](#_Toc18839_WPSOffice_Level2) [63](#_Toc18839_WPSOffice_Level2)

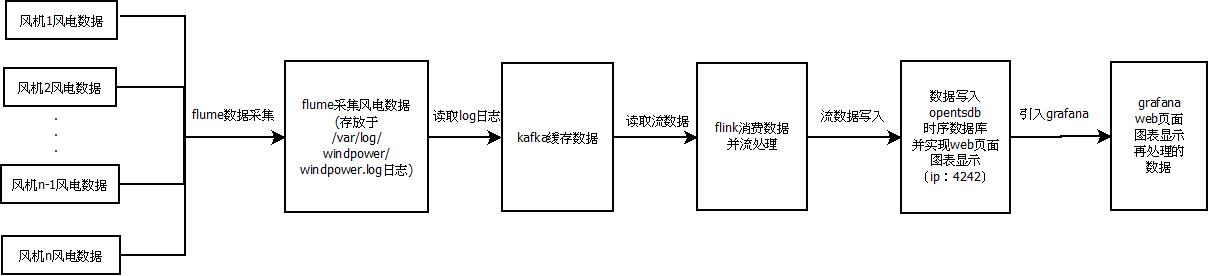
# 

# 一、前言：

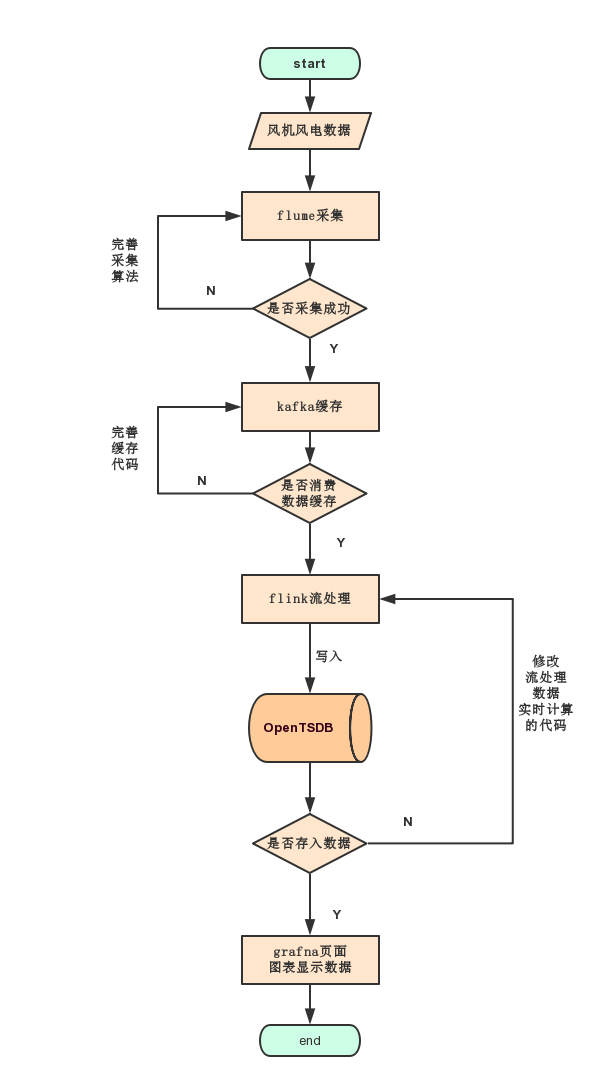
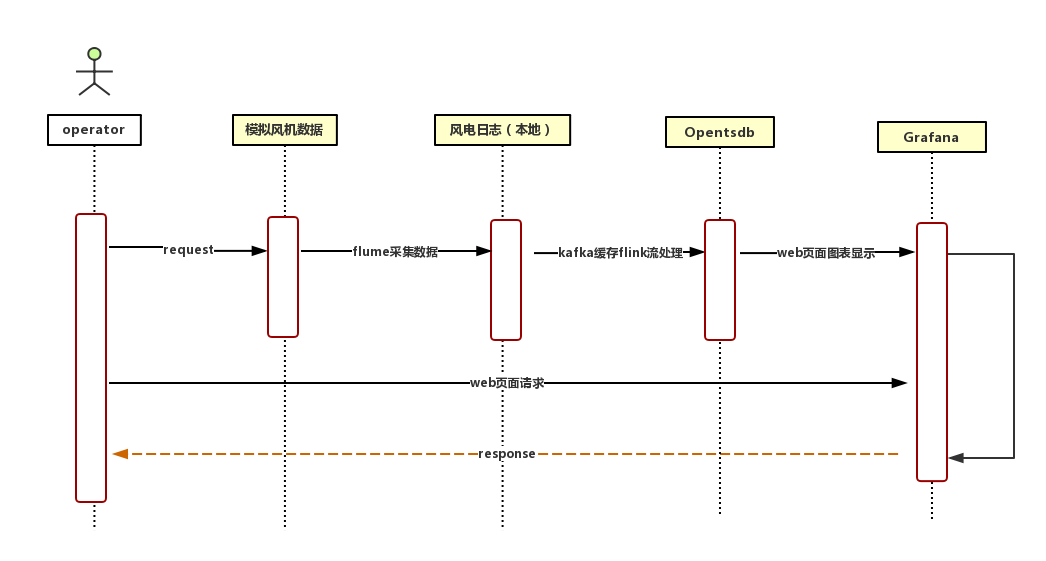
本文档讲述了集群的搭建步骤，flume的安装及配置，kafka集群配置，flink集群配置、hbase的安装、opentsdb数据库的安装以及各个各个组件的连接运行可视化

# 二、项目说明：

本项目为一个web程序，能够利用大数据技术，对数据源进行图形化展示。当访问此程序时，可以数据源信息的图表展示。本程序适合对此类海量数据进行分析和统计工作，并快速得出统计结果。



架构图



流程图

# 三、程序运行环境：

1. 采用三台虚拟机来模拟真实环境下的Hadoop高可用集群服务。主机名和ip地址分别为master1-169.254.51.101、master2-169.254.51.102、slave1-169.254.51.103。
2. 在master1、master2、slave1上分别安装:flume数据导入工具、kafka（配置集群）、flink（配置集群）、hbase（opentsdb基于hbase所以需要）、opentsdb数据库
3. 所有组件基于jdk1.8（省略安装步骤）

# 四、实现过程:

本项目实现有几个关键步骤

1. flume与kafka连接（简易）
2. Kafka与flink连接（复杂）
3. Flink与opentsdb的连接（复杂）

# 五、集群配置过程：

（这是当初我们学习时的搭建步骤master对应上文master1，slave1对应上文master2,slave2对应上文slave1，贴过来给大家一点参考，如果你集群搭建完毕了就直接忽略）

1.本地安装VMware，搭建三台虚拟机：master、slave1和slave2

记得更改数据镜像源!!!国内主要开源的开源镜像站点网易和阿里云,可以更改为这两个之一，在此使用网易的镜像

1.修改CentOS默认yum源为mirrors.163.com

（1）首先备份系统自带yum源配置文件：mv /etc/yum.repos.d/hanwate\_cdrom.repo /etc/yum.repos.d/hanwate\_cdrom.repo.backup

（2）进入yum源配置文件所在的文件夹：cd /etc/yum.repos.d/

（3）下载163的yum源配置文件到上面那个文件夹内：wget http://mirrors.163.com/.help/CentOS7-Base-163.repo

（4）运行yum makecache生成缓存：[root@master yum.repos.d]# yum makecache

## 2.分别配置hostname、ip、配置hosts文件

建立域名映射以及开启下载的镜像功能和挂载光盘

1.配置静态ip，防止动态出错

（1）输入命令：vim /etc/sysconfig/network-scripts/ifcfg-ens33 ---> 设置静态ip

配置如下：修改BOOTPROTO="dhcp"为BOOTPROTO="static"，添加：

IPADDR=192.168.75.133  
GATEWAY=192.168.75.1  
NETMASK=255.255.255.0

重启网络，输入命令：

systemctl restart network

----> 验证是否配置成功输入：

ifconfig

2.配置hostname

输入命令:hostnamectl set-hostname 主机名

----> 验证：输入:hostname 举例如下：

[root@localhost ~]# hostnamectl set-hostname master  
[root@localhost ~]# hostname  
master

3.三台主机分别配置hosts文件

(1)输入命令：vim /etc/hosts

添加域名映射如下：ip改为自己的，三个主机都要配

192.168.75.133 master  
192.168.75.131 slave1  
192.168.75.132 slave2

(2)验证是否ping通

输入命令：ping 主机名

[root@master ~]# ping slave2

PING slave2 (192.168.75.132) 56(84) bytes of data.  
64 bytes from slave2 (192.168.75.132): icmp\_seq=1 ttl=64 time=1.25 ms  
64 bytes from slave2 (192.168.75.132): icmp\_seq=2 ttl=64 time=0.518 ms

4.分别开启下载的镜像功能（针对于hanwate\_cdrom.repo版本，若改为阿里云或者网易，不用进行此步操作）

输入命令：vim /etc/yum.repos.d/hanwate\_cdrom.repo

将enabled的值修改为1，即ok

5.分别挂载光盘

分别输入命令：

[root@master ~]# mkdir /mnt/cdrom  
[root@master ~]# mount /dev/sr0 /mnt/cdrom  
mount: /dev/sr0 写保护，将以只读方式挂载

查看挂载情况：

[root@master ~]# df -h  
文件系统 容量 已用 可用 已用% 挂载点  
/dev/mapper/hbo-root 17G 5.5G 11G 35% /  
devtmpfs 736M 0 736M 0% /dev  
tmpfs 748M 0 748M 0% /dev/shm  
tmpfs 748M 8.6M 739M 2% /run  
tmpfs 748M 0 748M 0% /sys/fs/cgroup  
/dev/sda1 976M 116M 794M 13% /boot  
tmpfs 150M 0 150M 0% /run/user/0  
/dev/sr0 4.1G 4.1G 0 100% /mnt/cdrom

## 3.分别ssh安全配置以及ssh key认证登录

1.查看是否存在.ssh

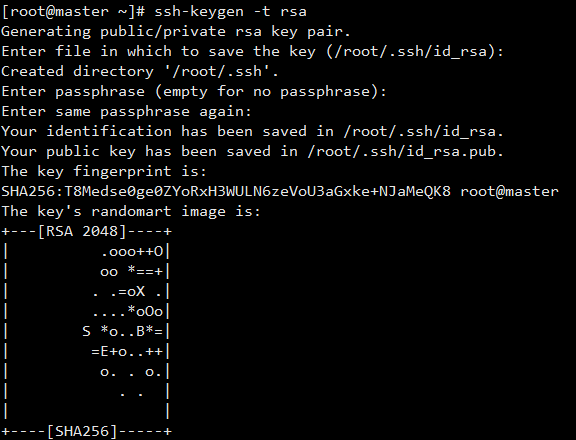
输入命令：ls -a

若存在，则输入 rm -rf /root/.ssh卸载删除

若无，则执行安装生成操作

2.生成ssh

输入命令：ssh-keygen -t rsa 如图显示即成功生成ssh



3.给钥匙tail1.sources=src1

master上操作即可，输入命令：scp id\_rsa.pub root@主机名:/root/

[root@master ~]# cd .ssh  
[root@master .ssh]# scp id\_rsa.pub root@slave1:/root/  
root@slave1's password:   
id\_rsa.pub 100% 393 289.4KB/s 00:00   
[root@master .ssh]# scp id\_rsa.pub root@slave2:/root/  
The authenticity of host 'slave2 (192.168.75.132)' can't be established.  
ECDSA key fingerprint is SHA256:xn80YeSuVCeTMYWTydGMoq9qVxZUtklBcZAiBYm51y8.  
ECDSA key fingerprint is MD5:cf:0e:51:38:fb:fb:05:66:a4:15:78:56:33:c3:e3:f2.  
Are you sure you want to continue connecting (yes/no)? yes  
Warning: Permanently added 'slave2,192.168.75.132' (ECDSA) to the list of known hosts.  
root@slave2's password:   
id\_rsa.pub 100% 393 255.8KB/s 00:00

测试是否可以连接，如下成功

[root@master ~]# ssh slave1  
root@slave1's password:   
Last login: Wed Aug 8 09:54:26 2018 from gateway  
[root@slave1 ~]#

4.加保险，master和slave上都需要操作，输入命令 cat id\_rsa.pub>>.ssh/authorized\_keys

4.测试免密钥连接：如图成功

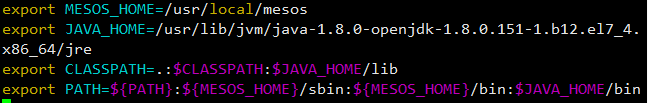
[root@master ~]# ssh slave1  
Last login: Wed Aug 8 11:22:03 2018 from master  
[root@slave1 ~]#

四、配置java环境变量

输入命令：vim /etc/profile

末尾添加：

export JAVA\_HOME=/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.151-1.b12.el7\_4.x86\_64/jre  
export CLASSPATH=.:$CLASSPATH:$JAVA\_HOME/lib  
export PATH=$PATH:$JAVA\_HOME/bin



执行命令生效，输入命令：source /etc/profile

验证：

[root@master ~]# $PATH  
-bash: /usr/lib64/qt-3.3/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/usr/local/mesos/sbin:/usr/local/mesos/bin:/root/bin:/usr/local/mesos/sbin:/usr/local/mesos/bin:/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.151-1.b12.el7\_4.x86\_64/jre/bin:

五、搭建hadoop集群(master,slave1和slave2)

1.切换进配置文件路径

输入命令：cd $HADOOP\_HOME/etc/hadoop/conf

查看位置命令：pwd

[root@master ~]# cd $HADOOP\_HOME/etc/hadoop/conf  
[root@master conf]# pwd  
/etc/hadoop/conf

2.增加slave节点

输入命令：vim /etc/hadoop/conf/slaves

修改添加：

master  
slave1  
slave2

3.配置core-site.xml

输入命令：vim /etc/hadoop/conf/core-site.xml

末尾添加：

<configuration>  
 <property>  
 <name>hadoop.tmp.dir</name>  
 <value>/usr/hdp/tmp</value>  
 </property>  
 <property>  
 <name>fs.defaultFS</name>  
 <value>hdfs://master:8020</value>  
 </property>  
</configuration>

4.配置hdfs-site.xml

输入命令：vim /etc/hadoop/conf/hdfs-site.xml

添加：

<configuration>  
 <property>  
 <name>dfs.replication</name>  
 <value>2</value>  
 </property>  
 <property>  
 <name>dfs.namenode.name.dir</name>  
 <value>/hadoop/hdfs/name</value>  
 </property>  
 <property>  
 <name>dfs.datanode.data.dir</name>  
 <value>/hadoop/hdfs/data</value>  
 </property>  
</configuration>

5.创建hdfs需要用的文件目录

三台分别输入命令：

[root@master conf]# cd  
[root@master ~]# mkdir /usr/hdp/tmp -p  
[root@master ~]# mkdir /hadoop/hdfs/{data,name} -p  
[root@master ~]# chown -R hdfs:hadoop /hadoop  
[root@master ~]# chown -R hdfs:hadoop /usr/hdp/tmp

6.初始化hdfs文件系统

只在master上操作：

输入命令：sudo -E -u hdfs hdfs namenode -format

成功显示：

IMG_258

7.启动hdfs文件系统

（1）启动master服务，输入命令：

systemctl start hadoop-hdfs-namenode  
systemctl start hadoop-hdfs-secondarynamenode

输入jps查看：

[root@master ~]# jps  
2286 NameNode  
2446 Jps  
2367 SecondaryNameNode

（2）启动slave1、slave2节点上的服务，输入命令：

systemctl start hadoop-hdfs-datanode

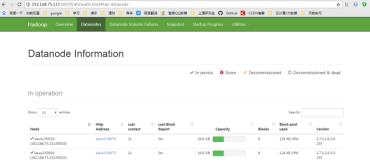
输入jps查看：

[root@slave1 ~]# jps  
2185 Jps  
2105 DataNode  
[root@slave2 ~]# jps  
2139 Jps  
2059 DataNode

8.网址查看：

输入ip地址加：50070

例如：192.168.75.133:50070



9.在hdfs中准备运行目录

输入命令：

[root@master ~]# su - hdfs  
-bash-4.2$ hadoop fs -mkdir /tmp  
-bash-4.2$ hadoop fs -chmod -R 1777 /tmp  
-bash-4.2$ hadoop fs -mkdir -p /var/log/hadoop-yarn  
-bash-4.2$ hadoop fs -chown yarn:mapred /var/log/hadoop-yarn  
-bash-4.2$ hadoop fs -mkdir /user  
-bash-4.2$ hadoop fs -mkdir /user/hadoop  
-bash-4.2$ hadoop fs -mkdir /user/history  
-bash-4.2$ hadoop fs -chmod 1777 /user/history  
-bash-4.2$ hadoop fs -chown mapred:hadoop /user/history  
-bash-4.2$ exit  
登出

验证：登录网址查看

# 六、搭建yarn集群

## 1.配置yarn-site.xml

分别输入命令：

[root@master ~]# vim /etc/hadoop/conf/yarn-site.xml

添加：

<configuration>  
 <property>  
 <name>yarn.resourcemanager.hostname</name>  
 <value>master</value>  
 </property>  
  
 <property>  
 <name>yarn.nodemanager.aux-services</name>  
 <value>mapreduce\_shuffle</value>  
 </property>  
  
 <property>  
 <name>yarn.nodemanager.local-dirs</name>  
 <value>file:///hadoop/yarn/local</value>  
 </property>  
  
 <property>  
 <name>yarn.nodemanager.log-dirs</name>  
 <value>/var/log/hadoop-yarn/containers</value>  
 </property>  
  
 <property>  
 <name>yarn.nodemanager.remote-app-log-dir</name>  
 <value>/var/log/hadoop-yarn/apps</value>  
 </property>  
  
 <property>  
 <name>yarn.log-aggregation-enable</name>  
 <value>true</value>  
 </property>  
  
 <property>  
 <name>yarn.scheduler.minimum-allocation-mb</name>  
 <value>511</value>  
 </property>  
  
 <property>  
 <name>yarn.scheduler.maximum-allocation-mb</name>  
 <value>2049</value>  
 </property>  
  
 <property>  
 <name>yarn.nodemanager.vmem-pmem-ratio</name>  
 <value>2</value>  
 </property>  
  
 <property>  
 <name>yarn.nodemanager.vmem-check-enabled</name>  
 <value>false</value>  
 </property>  
  
 <property>  
 <name>yarn.application.classpath</name>  
 <value>$HADOOP\_CONF\_DIR,  
 /usr/hdp/2.6.3.0-235/hadoop/\*,  
 /usr/hdp/2.6.3.0-235/hadoop/lib/\*,  
 /usr/hdp/2.6.3.0-235/hadoop-hdfs/\*,  
 /usr/hdp/2.6.3.0-235/hadoop-hdfs/lib/\*,  
 /usr/hdp/2.6.3.0-235/hadoop-yarn/\*,  
 /usr/hdp/2.6.3.0-235/hadoop-yarn/lib/\*,  
 /usr/hdp/2.6.3.0-235/hadoop-mapreduce/\*,  
 /usr/hdp/2.6.3.0-235/hadoop-mapreduce/lib/\*,  
 /usr/hdp/2.6.3.0-235/hadoop-httpfs/\*,  
 /usr/hdp/2.6.3.0-235/hadoop-httpfs/lib/\*  
 </value>  
 </property>  
</configuration>

或者采用写好master然后其他复制的方式，如下命令：

[root@master ~]# scp /etc/hadoop/conf/yarn-site.xml root@slave1:/etc/hadoop/conf/yarn-site.xml   
yarn-site.xml 100% 2268 2.1MB/s 00:00   
[root@master ~]# scp /etc/hadoop/conf/yarn-site.xml root@slave2:/etc/hadoop/conf/yarn-site.xml   
yarn-site.xml 100% 2268 1.5MB/s 00:00

## 2.配置mapred-site.xml

在mapred-site.xml中添加:

<configuration>  
 <property>  
 <name>mapreduce.framework.name</name>  
 <value>yarn</value>  
 </property>  
  
 <property>  
 <name>mapreduce.jobhistory.address</name>  
 <value>slave1:10020</value>  
 </property>  
  
 <property>  
 <name>mapreduce.jobhistory.webapps.address</name>  
 <value>slave1:19888</value>  
 </property>  
  
 <property>  
 <name>yarn.app.mapreduce.am.staging-dir</name>  
 <value>/user</value>  
 </property>  
  
 <property>  
 <name>mapreduce.application.classpath</name>  
 <value>  
 /etc/hadoop/conf/\*,  
 /usr/hdp/2.6.3.0-235/hadoop/\*,  
 /usr/hdp/2.6.3.0-235/hadoop-hdfs/\*,  
 /usr/hdp/2.6.3.0-235/hadoop-yarn/\*,  
 /usr/hdp/2.6.3.0-235/hadoop-mapreduce/\*,  
 /usr/hdp/2.6.3.0-235/hadoop/lib/\*,  
 /usr/hdp/2.6.3.0-235/hadoop-hdfs/lib/\*,  
 /usr/hdp/2.6.3.0-235/hadoop-yarn/lib/\*,  
 /usr/hdp/2.6.3.0-235/hadoop-mapreduce/lib/\*  
 </value>  
 </property>  
  
 <property>  
 <name>mapreduce.map.java.opts</name>  
 <value>-Xmx1024M</value>  
 </property>  
  
 <property>  
 <name>mapreduce.map.memory.mb</name>  
 <value>31</value>  
 </property>  
  
 <property>  
 <name>mapreduce.reduce.java.opts</name>  
 <value>-Xmx1024M</value>  
 </property>  
  
 <property>  
 <name>mapreduce.reduce.memory.mb</name>  
 <value>63</value>  
 </property>  
</configuration>

完整命令如下：

[root@master ~]# cd /etc/hadoop/conf/  
[root@master conf]# ls  
capacity-scheduler.xml kms-log4j.properties  
configuration.xsl kms-site.xml  
container-executor.cfg log4j.properties  
core-site.xml mapred-env.cmd  
hadoop-env.cmd mapred-env.sh  
hadoop-env.sh mapred-queues.xml.template  
hadoop-metrics2.properties mapred-site.xml.template  
hadoop-metrics.properties slaves  
hadoop-policy.xml ssl-client.xml.example  
hdfs-site.xml ssl-server.xml.example  
kms-acls.xml yarn-env.cmd  
kms-env.sh yarn-site.xml  
[root@master conf]# cp mapred-site.xml.template mapred-site.xml  
[root@master conf]# vim mapred-site.xml  
[root@master conf]# scp mapred-site.xml root@slave1:/etc/hadoop/conf/mapred-site.xml  
mapred-site.xml 100% 2528 875.6KB/s 00:00   
[root@master conf]# scp mapred-site.xml root@slave2:/etc/hadoop/conf/mapred-site.xml  
mapred-site.xml 100% 2528 2.1MB/s 00:00

## 3.配置yarn的本地目录

三台分别输入命令：

[root@master ~]# touch /etc/hadoop/conf/yarn-env.sh  
[root@master ~]# mkdir -p /hadoop/yarn/local  
[root@master ~]# chown yarn:yarn -R /hadoop/yarn/local

## 4.启动yarn服务

master上开启resourcemanager：systemctl start hadoop-yarn-resourcemanager

slave上开启historyserver：systemctl start hadoop-mapreduce-historyserver

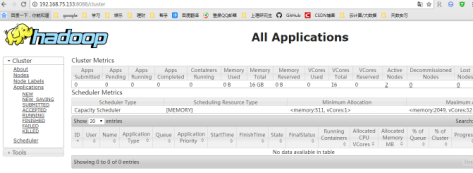
在所有启动datanode的节点上开nodemanager：systemctl start hadoop-yarn-nodemanager

jps查看结果：

[root@master ~]# jps  
3206 Jps  
2973 ResourceManager  
2286 NameNode  
2367 SecondaryNameNode  
[root@slave1 ~]# jps  
2377 NodeManager  
2105 DataNode  
2299 JobHistoryServer  
2523 Jps  
[root@slave2 ~]# jps  
2202 Jps  
2059 DataNode

## 5.验证

输入网址:192.168.75.133:8088



输入网址:192.168.75.131:19888



# 七、安装配置flume步骤：

## 1.1 下载安装包并解压

$ wget http://archive.cloudera.com/cdh5/cdh/5/flume-ng-1.6.0-cdh5.7.1.tar.gz

$ tar -xvf flume-ng-1.6.0-cdh5.7.1.tar.gz

$ rm flume-ng-1.6.0-cdh5.7.1.tar.gz

$ mv apache-flume-1.6.0-cdh5.7.1-binflume-1.6.0-cdh5.7.1

## 1.2 修改 flume-env.sh 配置文件,主要是JAVA\_HOME变量设置

# Enviroment variables can be set here.

export JAVA\_HOME=/usr/java/jdk1.8.0\_91

## 1.3 配置flume文件

# #配置Agent

a1.sources = r1

a1.sinks = k1

a1.channels = c1

# Describe/configure the source

a1.sources.r1.type = exec

#下面这一行表明数据从哪里采集，我这里做测试的时候是采集的nginx的服务访问日志

a1.sources.r1.command = tail -F /var/log/windpower/windpower.log

（注意上面/var/log/windpower/windpower.log这个路径是自己设置的source读取的路径）

#

# Describe the sink

**#下面表明数据采集了往哪里输送，我们既然是flume连接kafka，所以类型是 #org.apache.flume.sink.kafka.KafkaSink，kafka是一个具有缓存作用的组件，它数据的#进出通过话题的形式，有人生产话题有人消费话题，当我们的flume采集到数据后，flume #成为了生产者，把这些数据生产进名为test的话题a1.sinks.k1.topic = test（端口号为9092）**

a1.sinks.k1.type = logger

a1.sinks.k1.type = org.apache.flume.sink.kafka.KafkaSink

a1.sinks.k1.topic = test

a1.sinks.k1.brokerList = master1:9092,master2:9092,slave1:9092

a1.sinks.k1.requiredAcks = 1

a1.sinks.k1.batchSize = 20

#

# # Use a channel which buffers events in memory

a1.channels.c1.type = memory

a1.channels.c1.capacity = 1000

a1.channels.c1.transactionCapacity = 100

#

# # Bind the source and sink to the channel

a1.sources.r1.channels = c1

a1.sinks.k1.channel = c1

## 1.4 启动flume

flume-ng agent --conf /etc/flume/conf/ --conf-file/etc/flume/conf/flume.conf --name a1

（以上是一行代码换行处是空格）

# 八、Flink及kafka安装部署

## 1、简介

1.1、Kafka Consumer提供了2种API：high level与low level（SimpleConsumer）。   
（1）high level consumer的API较为简单，不需要关心offset、partition、broker等信息，kafka会自动读取zookeeper中该consumer group的last offset。   
（2）low level consumer也叫SimpleConsumer，这个接口非常复杂，以下几种情况才会用到：

1、Read a message multiple times

2、Consume only a subset of the partitions in a topic in a process

3、Manage transactions to make sure a message is processed once and only once

## 2、Flink的开发准备

Flink提供了high level的API来消费kafka的数据：flink-connector-kafka-0.10\_2.10。注意，这里的0.10代表的是kafka的版本，你可以通过maven来导入kafka的依赖，具体如下： 

例如你的kafka安装版本是“kafka\_2.10-0.10.2.1”,即此版本是由scala2.10编写，kafka的自身版本是0.10.2.1.那此时你需要添加如下的内容到maven的pom.xml文件中：

<dependency>

<groupId>org.apache.flink</groupId>

<artifactId>flink-connector-kafka-0.10\_2.10</artifactId>

<version>${flink.version}</version></dependency>

注意：   
 $${flink.version}是个变量，自己调整下代码，例如可以直接写1.3.2。我的项目 里采用的是添加了properties来控制${flink.version}：

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<flink.version>1.3.2</flink.version></properties>

## 当时我入门的时候觉得很奇怪为什么会有maven出现，那是因为在这个demo中flink和kafka的连接部分使用了scala，而相关运用会在本文的后半部分出现。

## 3、集群环境准备

这里主要是介绍下Flink集群与kafka集群的搭建。   
基础的软件安装包括JDK、scala、hadoop、zookeeper、kafka以及flink就不介绍了，直接看下flink的集群配置以及kafka的集群配置。   
zookeeper–3.4.6   
hadoop–2.7.3

kafka\_2.10-0.10.1.2.6.3.0-235  
flink–1.3.2

### 3.1、Flink集群配置（standalone且没有用zookeeper的HA）

**（本文默认配置的三台虚拟机为master1、master2、slave1）**



如图是我的虚拟机的映射 /etc/hosts

### 3.1.1.下载Flink压缩包

下载地址：[http://flink.apache.org/downloads.html](http://flink.apache.org/downloads.html" \t "https://blog.csdn.net/zhou_shaowei/article/details/_blank)。

我集群环境是hadoop2.7，Scala2.11版本的，所以下载的是：

flink-1.3.2-bin-hadoop27-scala\_2.11.tgz。

### 3.1.2.解压

分别上传至master1、master2、slave1的相同目录（默认下载的地址为 /opt/下）

，找到压缩版并执行如下命令解压：

|  |
| --- |
| tar xzf flink-1.3.2-bin-hadoop27-scala\_2.11.tgz |

### 3.1.3.配置master节点

选择一个 master节点(JobManager)然后在conf/flink-conf.yaml中设置jobmanager.rpc.address 配置项为该节点的IP 或者主机名。确保所有节点有有一样的jobmanager.rpc.address 配置。

jobmanager.rpc.address: master1

jobmanager.rpc.port: 6123

jobmanager.heap.mb: 1024

taskmanager.heap.mb: 1024

taskmanager.numberOfTaskSlots: 1

taskmanager.memory.preallocate: false

parallelism.default: 1

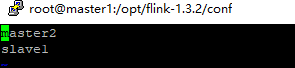
jobmanager.web.port: 8081

作者这里是在master1中配置上文信息然后将其拷贝至其他两台虚拟机（master2、slave1）

# （关于配置作用参考我写的另外两篇“Flink Jobmanager HA配置（standalone）”、“Flink运行时之TaskManager执行Task”）

### 3.1.4.配置slaves

将所有的 slave 节点 （TaskManager）的IP 或者主机名（一行一个）填入/opt/flink-1.3.2/conf/slaves文件中，并保证三个节点的slaves文件都一致



### 3.1.5.注意

问https://ci.apache.org/projects/flink/flink-docs-release-1.0/setup/config.html查看更多可用的配置项。为了使Flink 更高效的运行，还需要设置一些配置项。

以下都是非常重要的配置项：

1、TaskManager总共能使用的内存大小（taskmanager.heap.mb）

2、每一台机器上能使用的 CPU 个数（taskmanager.numberOfTaskSlots）

3、集群中的总 CPU个数（parallelism.default）

4、临时目录（taskmanager.tmp.dirs）

### 3.2、kafka集群配置

3.2.1、环境变量 

省略(其实不配置也没事，只不过要知道自己文件安装的kafka的bin目录在哪，每次在bin目录下启动)

3.2.2、配置config/zookeeper.properties   
由于kafka集群依赖于zookeeper集群，所以kafka提供了通过kafka去启动zookeeper集群的功能，当然也可以手动去启动zookeeper的集群而不通过kafka去启动zookeeper的集群。 （配置内容如下）（zookeeper不要用kafka自带的，而是自己安装下载一个，上文有对应版本）  
dataDir=/usr/hdp/2.6.3.0-235/zookeeper/data

（注意这里的dataDir最好不要指定/tmp目录下，因为机器重启会删除此目录下的文件。且指定的新路径必须存在。）

dataLogDir=/usr/hdp/2.6.3.0-235/zookeeper

clientPort=2181

maxClientCnxns=100

tickTime=2000

initLimit=10

syncLimit=5

3.2.3、配置config/server.properties   
这个文件是启动kafka集群需要指定的配置文件，注意2点：

# The id of the broker. This must be set to a unique integer for each broker.

broker.id=0

############################# Socket Server Settings #############################

# The port the socket server listens on#port=9092

listeners=PLAINTEXT://:9092

broker.id在kafka集群的每台机器上都不一样，我这里3台集群分别是0、1、2.

############################# Zookeeper #############################

# Zookeeper connection string (see zookeeper docs for details).# This is a comma separated host:port pairs, each corresponding to a zk# server. e.g. "127.0.0.1:3000,127.0.0.1:3001,127.0.0.1:3002".# You can also append an optional chroot string to the urls to specify the# root directory for all kafka znodes.

zookeeper.connect=master1:2181,master2:2181,slave1:2181

# Timeout in ms for connecting to zookeeper

zookeeper.connection.timeout.ms=6000

zookeeper.connect要配置kafka集群所依赖的zookeeper集群的信息，hostname:port。

3.2.4、复制kafka路径及环境变量到其他kafka集群的机器，并修改server.properties中的broker\_id. （master2的broker\_id = 1,slave1的broker\_id = 2）

# 九、配置Hbase集群

## 1.配置文件hbase-site.xml

命令：vim /etc/hbase/conf/hbase-site.xml

分别添加：

"<configuration>

<property>

<name>hbase.rootdir</name>

<value>hdfs://master:8020/user/hbase</value>

</property>

<property>

<name>hbase.cluster.distributed</name>

<value>true</value>

</property>

<property>

<name>hbase.zookeeper.property.clientPort</name>

<value>2181</value>

</property>

<property>

<name>hbase.zookeeper.quorum</name>

<value>master,slave1,slave2</value>

</property>

</configuration>"

## 2.配置修改hbase-env.sh

输入命令：vim /etc/hbase/conf/hbase-env.sh

添加：

"export HBASE\_PID\_DIR=/var/run/hbase

export HBASE\_LOG\_DIR=/var/log/hbase

export HBASE\_MANAGES\_ZK=false"

3.配置修改修改regionservers

输入命令：vim /etc/hbase/conf/regionservers

添加：

"master

slave1

slave2"

4.准备hbase目录

"[root@master1 ~]# su - hdfs

上一次登录：三 8月 8 13:16:09 CST 2018pts/3 上

-bash-4.2$ hadoop fs -mkdir /user/hbase

-bash-4.2$ hadoop fs -chown hbase /user/hbase

-bash-4.2$ exit

登出"

5.修改环境变量

输入命令：vim /etc/init.d/hbase-master +55

修改为：export HBASE\_HOME="/usr/hdp/2.6.3.0-235/hbase"

6.启动Hbase集群

（1）启动master服务

输入命令：/etc/init.d/hbase-master start

"[root@master1 ~]# /etc/init.d/hbase-master start

Starting HBase master daemon (hbase-master): [ 确定 ]

starting master, logging to /var/log/hbase/hbase-hbase-master-master.out

OpenJDK 64-Bit Server VM warning: ignoring option PermSize=128m; support was removed in 8.0

OpenJDK 64-Bit Server VM warning: ignoring option MaxPermSize=128m; support was removed in 8.0

OpenJDK 64-Bit Server VM warning: ignoring option PermSize=128m; support was removed in 8.0

OpenJDK 64-Bit Server VM warning: ignoring option MaxPermSize=128m; support was removed in 8.0"

（2）启动regionserver

输入命令：/etc/init.d/hbase-regionserver start

"[root@master ~]# /etc/init.d/hbase-regionserver start

Starting HBase regionserver daemon: starting regionserver, logging to /var/log/hbase/hbase-hbase-regionserver-master.out

OpenJDK 64-Bit Server VM warning: ignoring option PermSize=128m; support was removed in 8.0

OpenJDK 64-Bit Server VM warning: ignoring option MaxPermSize=128m; support was removed in 8.0

OpenJDK 64-Bit Server VM warning: ignoring option PermSize=128m; support was removed in 8.0

OpenJDK 64-Bit Server VM warning: ignoring option MaxPermSize=128m; support was removed in 8.0

hbase-regionserver."

# 十、OpenTSDB部署

（1）下载OpenTSDB

官网下载2.1.0版本，放入虚拟机中/root目录下

Wget https://github.com/OpenTSDB/opentsdb/releases/download/v2.1.0/opentsdb-2.1.0.tar.gz

（2）解压

输入命令：tar -xzvf opentsdb-2.1.0.tar.gz -C /opt

[root@master ~]# tar -xzvf opentsdb-2.1.0.tar.gz -C /opt

（3）Opentsdb的源码使用的是ISO-8859-1，要支持中文源码要改成UTF-8  
解压源码，进入到opentsdb-2.1.0目录下，运行  
perl -pi -e 's|ISO-8859-1|UTF-8|g' `find ./ -type f`  
进入到opentsdb-2.1.0/src/tsd目录下，查看WordSplitter.java文件  
已变成UTF-8

完成这一步，之后编译的话，会解决中文不能写入的问题，但opentsdb的页面上显示仍然会乱码，下面是解决中文乱码的问题

（4）进入到opentsdb-2.1.0/src/graph目录，有个Plot.java文件

在280行处  
"set term png font '/usr/share/fonts/truetype/chinese/simsun.ttc,12'\n"

（5）编译

输入命令：./build.sh

若提示：即success

[root@master opentsdb-2.1.0]# ./build.sh  
+ test -f configure  
+ test -d build  
+ cd build  
+ test -f Makefile  
+ MAKE=make  
++ uname -s  
+ '[' Linux = FreeBSD ']'  
+ exec make  
make all-am  
make[1]: 进入目录“/opt/opentsdb-2.1.0/build”  
make[1]: 对“all-am”无需做任何事。  
make[1]: 离开目录“/opt/opentsdb-2.1.0/build”

通过下面命令在 HBase 中建立 opentsdb 所需的表。默认情况下 opentsdb 建立的 HBase 表启用了 lzo 压缩。需要开启 Hadoop 中的 lzo 压缩支持， 这里我们直接在下面脚本中把 COMPRESSION 的支持关闭。

修改/opt/opentsdb-2.1.0/src/create\_table.sh 设置 COMPRESSION=NONE，并且在文件开始处设置 HBase 所在目录，之后执行该脚本，在 HBase 中创建相应的表,即267脚本

2.创建表和运行

env COMPRESSION=NONE HBASE\_HOME=/usr/hdp/2.6.3.0-235/hbase ./src/create\_table.sh

可以将HBASE\_HOME写入profile

3.复制OpenTSDB 的配置文件/src/opentsdb.conf到/build/opentsdb.conf

修改opentsdb.conf,添加

tsd.network.port = 4242  
tsd.network.bind = 0.0.0.0  
tsd.http.staticroot =/opt/opentsdb-2.1.0/build/staticroot  
tsd.http.cachedir = /var/lib/zookeeper  
tsd.core.auto\_create\_metrics = true  
tsd.storage.hbase.data\_table = tsdb  
tsd.storage.hbase.zk\_quorum = master:2181,slave1:2181,slave2:2181  
tsd.storage.fix\_duplicates = true

这些需要的组件安装好之后，我们就要去打包我们kafka连接flink和flink连接opentsdb的代码了，这里有两种语言可以选择，java和scala，作者这里使用的java语言，推荐创建maven项目比较容易打包成flink需要上交的计划。

# 十一、组件连接：

## 一、flink连接kafka

**import** com.wugenqiang.test.writeIntoOpentsdb.WriteIntoOpentsdb;  
**import** org.apache.flink.api.common.functions.MapFunction;  
**import** org.apache.flink.streaming.api.TimeCharacteristic;  
**import** org.apache.flink.streaming.api.datastream.DataStream;  
**import** org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;  
**import** org.apache.flink.streaming.connectors.kafka.FlinkKafkaConsumer010;  
**import** org.apache.flink.streaming.util.serialization.SimpleStringSchema;  
  
**import** java.io.IOException;  
**import** java.util.Properties;  
**import** java.util.logging.Level;  
**import** java.util.logging.Logger;  
  
*/\*\*  
 \** ***@author*** *createdBy liuxiaojun editBy wangzhijian and wugenqiang  
 \** ***@Title:*** *ReadingFromKafka  
 \** ***@ProjectName*** *flink\_opentsdb  
 \** ***@Description:*** *flink读取kafka数据  
 \** ***@date*** *2018/8/29 10:29  
 \*/***public class** ReadingFromKafka {  
  
 */\*\*  
 　　\* @Description: flink读取kafka数据  
 　　\* @param []  
 　　\* @return void  
 　　\* @throws  
 　　\* @author leonardo*

*\* @date 2018/8/29 10:33  
 　　\*/* **public void** readingFromKafka(){  
  
 **final** String ZOOKEEPER\_HOST = **"master1:2181,master2:2181,slave1:2181"**;  
 **final** String KAFKA\_HOST = **"master1:9092,master2:9092,slave1:9092"**;  
 **final** StreamExecutionEnvironment env = StreamExecutionEnvironment.*getExecutionEnvironment*();*// 创建数据流，获得环境*  
 env.enableCheckpointing(1000); *// 非常关键，一定要设置启动检查点！！* env.setStreamTimeCharacteristic(TimeCharacteristic.***EventTime***);  
  
 Properties props = **new** Properties();  
 props.setProperty(**"zookeeper.connect"**, ZOOKEEPER\_HOST);  
 props.setProperty(**"bootstrap.servers"**, KAFKA\_HOST);  
 props.setProperty(**"group.id"**, **"transaction"**);  
  
 *//这里实现了flink消费kafka的数据，话题名为test*

DataStream<String> transction = env.addSource(**new** FlinkKafkaConsumer010<String>(**"test"**, **new** SimpleStringSchema(), props));  
*//此处处理一条数据中的多个数据项*

*//这里是flink进行数据清洗的过程，在过程中将处理好的流数据存放进opentsdb*  
 transction.rebalance().map(**new** MapFunction<String, Object>() {  
 **public** String map(String value)**throws** IOException {  
   
 */\*//此处处理一条数据仅有一个数据项  
 WriteIntoOpentsdb writeIntoOpentsdb = new WriteIntoOpentsdb();  
 writeIntoOpentsdb.writeIntoOpentsdb(value);\*/  
   
 //此处处理一条数据中的多个数据项* String result[]=value.split(**" "**);

*//通过WriteIntoOpentsdb中的writeIntoOpentsdb方法来进行对数据的写入*  
 WriteIntoOpentsdb writeIntoOpentsdb = **new** WriteIntoOpentsdb();  
 writeIntoOpentsdb.writeIntoOpentsdb(result);  
  
 **return** value;  
 }  
  
 }).print();  
  
 **try** {  
 env.execute();  
 } **catch** (Exception ex) {  
  
 Logger.*getLogger*(ReadingFromKafka.**class**.getName()).log(Level.***SEVERE***, **null**, ex);  
 ex.printStackTrace();  
 }  
 }  
  
}

## 二、flink连接opentsdb

接下来贴WriteIntoOpentsdb类：

**package** com.wugenqiang.test.writeIntoOpentsdb;  
  
**import** com.wugenqiang.test.Client.ExpectResponse;  
**import** com.wugenqiang.test.Client.HttpClientImpl;  
**import** com.wugenqiang.test.Client.builder.MetricBuilder;  
**import** com.wugenqiang.test.Client.response.Response;  
  
**import** java.io.IOException;  
  
*/\*\*  
 \** ***@author*** *wugenqiang  
 \** ***@Title:*** *WriteIntoOpentsdb  
 \** ***@ProjectName*** *flink\_opentsdb  
 \** ***@Description:*** *监听数据写入opentsdb  
 \** ***@date*** *2018/8/29 10:35  
 \*/***public class** WriteIntoOpentsdb {  
  
 */\*\*  
 　　\* @Description: flink读取kafka数据，此处处理一条数据仅有一个数据项  
 　　\* @param [value]  
 　　\* @return void  
 　　\* @throws  
 　　\* @author wugenqiang  
 　　\* @date 2018/8/29 11:06  
 　　\*//\*  
 public void writeIntoOpentsdb(String value) {  
  
 HttpClientImpl client = new HttpClientImpl("http://192.168.75.133:4242");  
  
 MetricBuilder builder = MetricBuilder.getInstance();  
  
 Double value2 = Double.parseDouble(value);  
 builder.addMetric("com.wugenqiang.test.windpower").setDataPoint(value2)  
 .addTag("tag1", "tab1value").addTag("tag2", "tab2value");  
  
 try {  
 Response response = client.pushMetrics(builder,  
 ExpectResponse.SUMMARY);  
 System.out.println(response);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
\*/  
  
 /\*\*  
 　　\* @Description: flink读取kafka数据，此处处理一条数据中的多个数据项  
 　　\* @param [value]  
 　　\* @return void  
 　　\* @throws  
 　　\* @author createdBy liuxiaojun editBy wugenqiang  
 　　\* @date 2018/8/29 12:42  
 　　\*/* **public void** writeIntoOpentsdb(String[] value) {  
 HttpClientImpl client = **new** HttpClientImpl(**"http://master1:4242"**);  
 *//监听端口号4242的数据*  
 MetricBuilder builder = MetricBuilder.*getInstance*();  
  
 *//时间戳* **long** timeStramp = Long.*parseLong*(value[0]);  
 *//风机编号* String fanNumber = value[1];  
 *//机型编号* String model = value[2];  
 *//环境温度* Double ambientTemperature =Double.*valueOf*(value[3]);  
 *//机身温度* Double fuselageTemperature =Double.*valueOf*(value[4]);  
 *//风速* Double windSpeed =Double.*valueOf*(value[5]);  
 *//发电量* Double powerGeneration =Double.*valueOf*(value[6]);  
  
 builder.addMetric(**"com.windpower.ambientTemperature"**).setDataPoint(timeStramp,ambientTemperature)  
 .addTag(**"fanNumber"**, fanNumber).addTag(**"model"**, model);  
  
 builder.addMetric(**"com.windpower.fuselageTemperature"**).setDataPoint(timeStramp,fuselageTemperature)  
 .addTag(**"fanNumber"**, fanNumber).addTag(**"model"**, model);  
  
 builder.addMetric(**"com.windpower.windSpeed"**).setDataPoint(timeStramp,windSpeed)  
 .addTag(**"fanNumber"**, fanNumber).addTag(**"model"**, model);  
  
 builder.addMetric(**"com.windpower.powerGeneration"**).setDataPoint(timeStramp,powerGeneration)  
 .addTag(**"fanNumber"**, fanNumber).addTag(**"model"**, model);  
  
 **try** {  
 Response response = client.pushMetrics(builder,  
 ExpectResponse.***SUMMARY***);  
 System.***out***.println(response);  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }  
}

这里只是贴出了重要的两段连接代码，具体项目可以通过在附带压缩包中，解压后使用idea打开。

# 十二、打包

接下来是对编写好的maven项目进行打包，打包的具体流程如下：

## 创建工程之前的准备

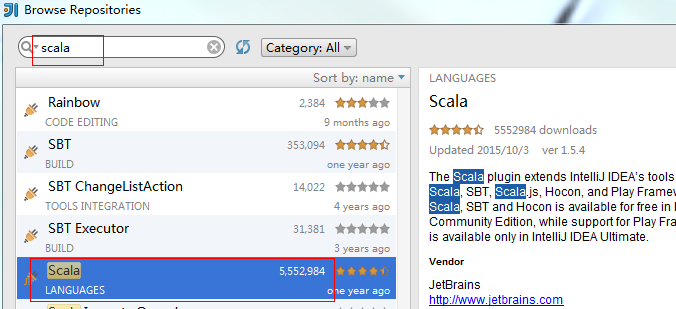
1. IntelliJ IDEA（最新版即可）
2. Jdk（1.8.0\_181）
3. Scala plugin for IDEA（在IDEA中下载）
4. Maven（3.5.3）
5. Scala的jar包（2.11.0）

## 创建步骤

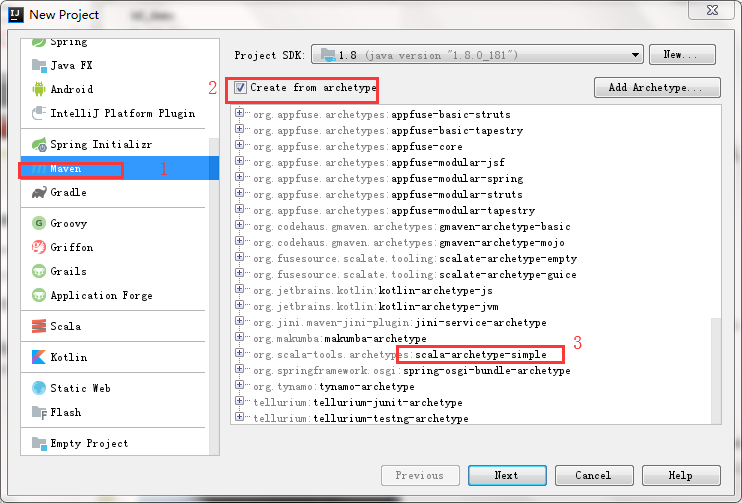
1. 点右下角的Configure,选择Plugins,安装scala插件



1. 在上方的搜索框中输入 Scala，选择Scala 这个插件，点击右侧边栏中的 Install Plugin 按钮，然后重启Intellij IDEA。

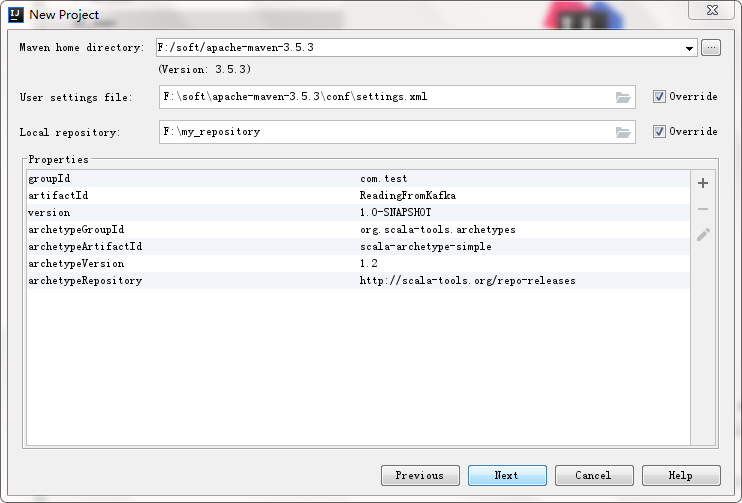


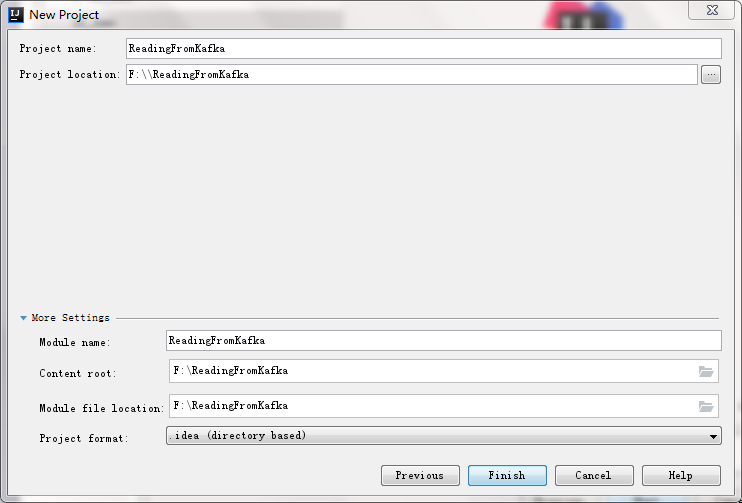
1. 下载maven的jar包（3.5.3）
2. 创建带scala的maven项目



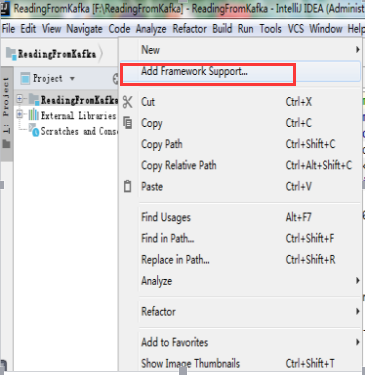


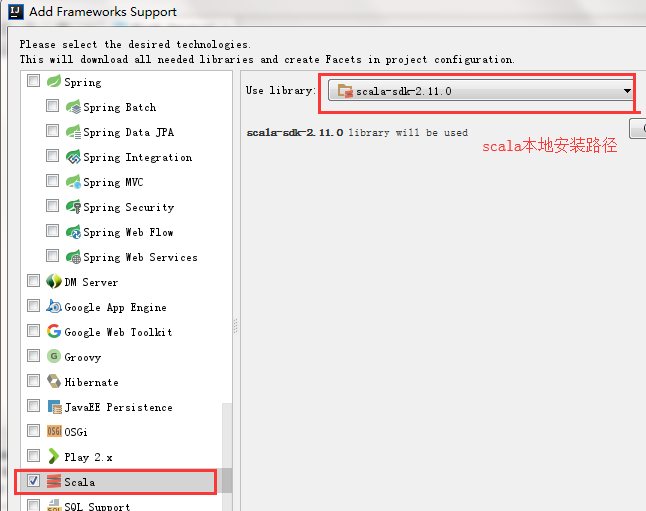
1. Maven的配置（目录填写自己下载的maven目录）



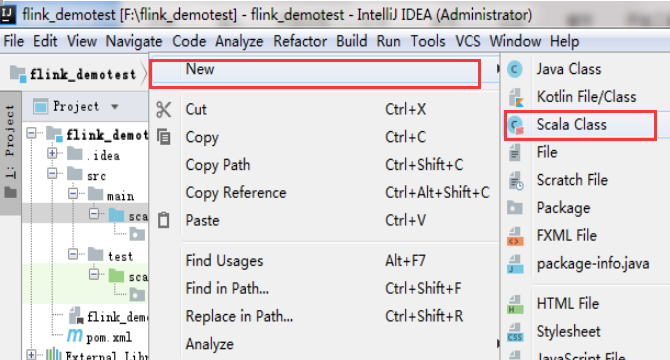


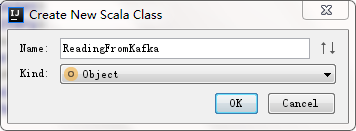
1. 右键Project，点击Add FrameWork Support，在打开的栏框中添加Scala支持





1. 创建scala文件





ReadingFromKafka内容（**修改topic的名称,及zookeeper\_host和kafka\_broker换成自己的主机名**）

|  |
| --- |
| **package** com.flink **import** java.util.Properties  **import** org.apache.flink.streaming.api.{CheckpointingMode, TimeCharacteristic} **import** org.apache.flink.streaming.api.environment.StreamExecutionEnvironment **import** org.apache.flink.streaming.connectors.kafka.FlinkKafkaConsumer010 **import** org.apache.flink.streaming.util.serialization.SimpleStringSchema **import** org.apache.flink.streaming.api.\_ **object** ReadingFromKafka {  **private val** *ZOOKEEPER\_HOST* = **"master1:2181,master2:2181,slave1:2181"  private val** *KAFKA\_BROKER* = **"master1:9092,master2:9092,slave1:9092"  private val** *TRANSACTION\_GROUP* = **"transaction"   def** main(args : Array[String]){  **val** env = StreamExecutionEnvironment.*getExecutionEnvironment* env.setStreamTimeCharacteristic(TimeCharacteristic.*EventTime*)  env.enableCheckpointing(1000)  env.getCheckpointConfig.setCheckpointingMode(CheckpointingMode.*EXACTLY\_ONCE*)   *// configure Kafka consumer* **val** kafkaProps = **new** Properties()  kafkaProps.setProperty(**"zookeeper.connect"**, *ZOOKEEPER\_HOST*)  kafkaProps.setProperty(**"bootstrap.servers"**, *KAFKA\_BROKER*)  kafkaProps.setProperty(**"group.id"**, *TRANSACTION\_GROUP*)   *//topicd的名字是new，schema默认使用SimpleStringSchema()即可* **val** transaction = env  .addSource(  **new** FlinkKafkaConsumer010[String](**"test"**, **new** SimpleStringSchema(), kafkaProps)  )  transaction.print()  env.execute()   } } |

1. Pom.xml配置（主要添加依赖和将项目打成jar包的插件）

添加的依赖：

|  |  |  |
| --- | --- | --- |
| **groupId** | **artifactId** | **version** |
| org.apache.flink | flink-core | **1.3.2** |
| org.apache.flink | flink-connector-kafka-0.10\_2.11 | **1.3.2** |
| org.apache.kafka | kafka\_2.11 | 0.10.2.0 |
| org.apache.flink | flink-streaming-java\_2.11 | **1.3.2** |

添加的插件：

|  |  |  |
| --- | --- | --- |
| **groupId** | **artifactId** | **version** |
| org.apache.maven.plugins | maven-assembly-plugin | 2.4.1 |

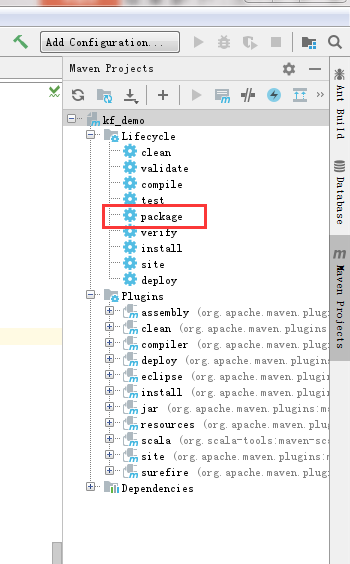
具体配置如下：**（注意修改maven-assembly-plugin的mainClass为自己主类的路径）**

|  |
| --- |
| <**project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4\_0\_0.xsd"**>  <**modelVersion**>4.0.0</**modelVersion**>  <**groupId**>com.flink</**groupId**>  <**artifactId**>kf\_demo</**artifactId**>  <**version**>1.0-SNAPSHOT</**version**>  <**inceptionYear**>2008</**inceptionYear**>  <**properties**>  <**scala.version**>2.11.8</**scala.version**>  </**properties**>   <**repositories**>  <**repository**>  <**id**>scala-tools.org</**id**>  <**name**>Scala-Tools Maven2 Repository</**name**>  <**url**>http://scala-tools.org/repo-releases</**url**>  </**repository**>  </**repositories**>   <**pluginRepositories**>  <**pluginRepository**>  <**id**>scala-tools.org</**id**>  <**name**>Scala-Tools Maven2 Repository</**name**>  <**url**>http://scala-tools.org/repo-releases</**url**>  </**pluginRepository**>  </**pluginRepositories**>   <**dependencies**>  <**dependency**>  <**groupId**>org.scala-lang</**groupId**>  <**artifactId**>scala-library</**artifactId**>  <**version**>${scala.version}</**version**>  </**dependency**>  <**dependency**>  <**groupId**>junit</**groupId**>  <**artifactId**>junit</**artifactId**>  <**version**>4.4</**version**>  <**scope**>test</**scope**>  </**dependency**>  <**dependency**>  <**groupId**>org.specs</**groupId**>  <**artifactId**>specs</**artifactId**>  <**version**>1.2.5</**version**>  <**scope**>test</**scope**>  </**dependency**>   *<!--flink的核心依赖-->* <**dependency**>  <**groupId**>org.apache.flink</**groupId**>  <**artifactId**>flink-core</**artifactId**>  <**version**>1.3.2</**version**>  <**scope**>compile</**scope**>  </**dependency**>   *<!-- flink连接kafka的jar包，0.10为kafka的版本号，2.11为scala的版本号-->* <**dependency**>  <**groupId**>org.apache.flink</**groupId**>  <**artifactId**>flink-connector-kafka-0.10\_2.11</**artifactId**>  <**version**>1.3.2</**version**>  <**scope**> compile</**scope**>  </**dependency**>   *<!-- kafka的核心jar包 -->* <**dependency**>  <**groupId**>org.apache.kafka</**groupId**>  <**artifactId**>kafka\_2.11</**artifactId**>  <**version**>0.10.2.0</**version**>  <**scope**> compile</**scope**>  </**dependency**>   *<!-- flink-streaming的jar包，2.11为scala版本号 -->* <**dependency**>  <**groupId**>org.apache.flink</**groupId**>  <**artifactId**>flink-streaming-java\_2.11</**artifactId**>  <**version**>1.3.2</**version**>  <**scope**> compile</**scope**>  </**dependency**>   </**dependencies**>   <**build**>  <**sourceDirectory**>src/main/scala</**sourceDirectory**>  <**testSourceDirectory**>src/test/scala</**testSourceDirectory**>  <**plugins**>    <**plugin**>  <**groupId**>org.scala-tools</**groupId**>  <**artifactId**>maven-scala-plugin</**artifactId**>  <**executions**>  <**execution**>  <**goals**>  <**goal**>compile</**goal**>  <**goal**>testCompile</**goal**>  </**goals**>  </**execution**>  </**executions**>  <**configuration**>  <**scalaVersion**>${scala.version}</**scalaVersion**>  <**args**>  <**arg**>-target:jvm-1.5</**arg**>  </**args**>  </**configuration**>  </**plugin**>  <**plugin**>  <**groupId**>org.apache.maven.plugins</**groupId**>  <**artifactId**>maven-eclipse-plugin</**artifactId**>  <**configuration**>  <**downloadSources**>true</**downloadSources**>  <**buildcommands**>  <**buildcommand**>ch.epfl.lamp.sdt.core.scalabuilder</**buildcommand**>  </**buildcommands**>  <**additionalProjectnatures**>  <**projectnature**>ch.epfl.lamp.sdt.core.scalanature</**projectnature**>  </**additionalProjectnatures**>  <**classpathContainers**>  <**classpathContainer**>org.eclipse.jdt.launching.JRE\_CONTAINER</**classpathContainer**>  <**classpathContainer**>ch.epfl.lamp.sdt.launching.SCALA\_CONTAINER</**classpathContainer**>  </**classpathContainers**>  </**configuration**>  </**plugin**>  *<!--将项目打成jar的插件，打成的jar包有两个，一个连同依赖也打进去，另一个则不带依赖。  注意：将mainClass的值换成自己的路径，即包名加类名-->* <**plugin**>  <**groupId**>org.apache.maven.plugins</**groupId**>  <**artifactId**>maven-assembly-plugin</**artifactId**>  <**version**>2.4.1</**version**>  <**configuration**>  *<!-- get all project dependencies -->* <**descriptorRefs**>  <**descriptorRef**>jar-with-dependencies</**descriptorRef**>  </**descriptorRefs**>  *<!-- MainClass in mainfest make a executable jar -->* <**archive**>  <**manifest**>  <**mainClass**>com.flink.ReadingFromKafka</**mainClass**>  </**manifest**>  </**archive**>   </**configuration**>  <**executions**>  <**execution**>  <**id**>make-assembly</**id**>  *<!-- bind to the packaging phase -->* <**phase**>package</**phase**>  <**goals**>  <**goal**>single</**goal**>  </**goals**>  </**execution**>  </**executions**>  </**plugin**>  </**plugins**>  </**build**>  <**reporting**>  <**plugins**>  <**plugin**>  <**groupId**>org.scala-tools</**groupId**>  <**artifactId**>maven-scala-plugin</**artifactId**>  <**configuration**>  <**scalaVersion**>${scala.version}</**scalaVersion**>  </**configuration**>  </**plugin**>  </**plugins**>  </**reporting**> </**project**> |

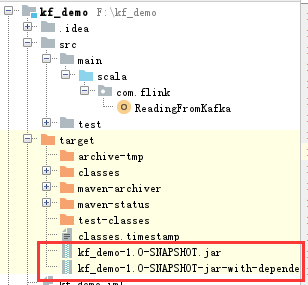
1. 配置完成后，找窗口右边的Maven Projects选项，点击Lifecycle,再选择打包package（如需重新打包先clean，再package），

**flink运行命令：flink run -c mainClass的路径 linux jar包存放路径**

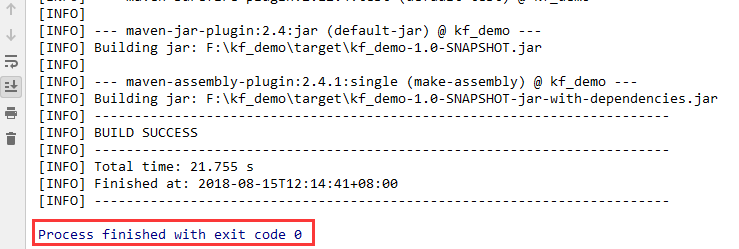
(如 flink run -c com.flink.ReadingFromKafka /root/xx.jar)



1. 成功code为0，项目目录会生成target目录，里面有打好的jar包

.

.



打包完之后算是一切准备工作全部就绪了，接下来开始启动服务在flink上提交计划：（我自己是全是写成脚本的，但在这里我将用无需配置环境变量的方式逐个启动）

# 十三、启动服务及运行

1. 首先启动zookeeper集群

（3台zookeeper机器都要启动）：

root@master1:/usr/local/zookeeper/zookeeper-3.4.6# ./bin/zkServer.sh start

root@master2:/usr/local/zookeeper/zookeeper-3.4.6# ./bin/zkServer.sh start

root@slave1:/usr/local/zookeeper/zookeeper-3.4.6# ./bin/zkServer.sh start

验证zookeeper集群：   
进程是否启动；zookeeper集群中是否可以正常显示leader以及follower。

root@master1:/usr/local/zookeeper/zookeeper-3.4.6# jps3295 QuorumPeerMain

root@master1:/usr/local/zookeeper/zookeeper-3.4.6# ./bin/zkServer.sh statusJMX enabled by defaultUsing config: /usr/local/zookeeper/zookeeper-3.4.6/bin/../conf/zoo.cfgMode: follower

root@mater2:/usr/local/zookeeper/zookeeper-3.4.6# ./bin/zkServer.sh statusJMX enabled by defaultUsing config: /usr/local/zookeeper/zookeeper-3.4.6/bin/../conf/zoo.cfgMode: follower

root@mater2:/usr/local/zookeeper/zookeeper-3.4.6#

root@mater2:/usr/local/zookeeper/zookeeper-3.4.6# ./bin/zkServer.sh statusJMX enabled by defaultUsing config: /usr/local/zookeeper/zookeeper-3.4.6/bin/../conf/zoo.cfgMode: leader

root@mater2:/usr/local/zookeeper/zookeeper-3.4.6/bin#

## 2、启动kafka集群（3台都要启动）

root@master1:/usr/local/kafka/kafka\_2.10-0.8.2.1# ./bin/kafka-server-start.sh conf/server.properties &

root@master2:/usr/local/kafka/kafka\_2.10-0.8.2.1# ./bin/kafka-server-start.sh conf/server.properties &

root@slave1:/usr/local/kafka/kafka\_2.10-0.8.2.1# ./bin/kafka-server-start.sh conf/server.properties &

验证：   
进程；日志

3512 Kafka

## 3、启动hdfs

[root@master1 ~]#systemctl start hadoop-hdfs-namenode

[root@master1 ~]#systemctl start hadoop-hdfs-datanode

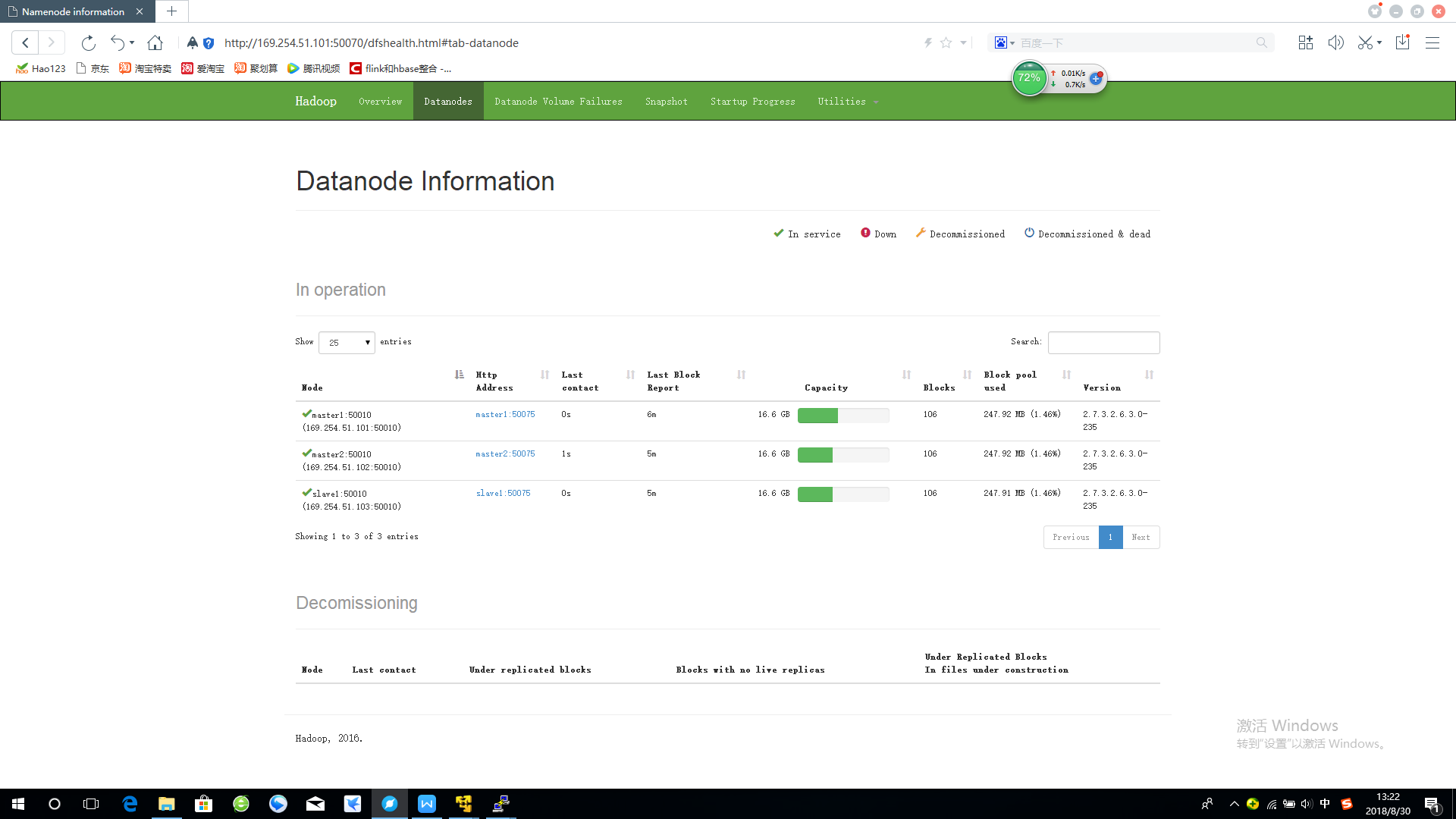
[root@master2 ~]#systemctl start hadoop-hdfs-secondarynamenode

[root@master2 ~]#systemctl start hadoop-hdfs-datanode

[root@slave1 ~]#systemctl start hadoop-hdfs-datanode

验证：进程及webUI

webUI：50070，默认可配置 



1. 启动Flink集群

（master1即可,当初Flink解压在哪就在哪启动）

[root@master1 flink-1.3.2]# ./bin/start-cluster.sh

Starting cluster.

Starting jobmanager daemon on host master1.

Starting taskmanager daemon on host master2.

Starting taskmanager daemon on host slave1.

验证：进程及WebUI

root@master1:# jps4411 JobManager

root@master2:# jps4151 TaskManager

root@slave1:# jps4110 TaskManager

WebUI：8081（默认，可配置）   
5、启动opentsdb

先启动hbase

[root@master1 ~]# /etc/init.d/hbase-master start

Starting HBase master daemon (hbase-master): [ 确定 ]

starting master, logging to /var/log/hbase/hbase-hbase-master-master1.out

[root@master1 ~]# /etc/init.d/hbase-regionserver start

Starting HBase regionserver daemon: starting regionserver, logging to /var/log/hbase/hbase-hbase-regionserver-master1.out

hbase-regionserver.

再启动opentsdb

进入到自己安装的opentsdb目录下的build目录下shuru

[root@master1 ~]# cd /opt/opentsdb-2.1.0/build

[root@master1 build]# ./tsdb tsd

2018-08-30 13:28:35,509 INFO [main] TSDMain: Starting.

2018-08-30 13:28:35,557 INFO [main] TSDMain: net.opentsdb 2.1.0 built at revision 248fee6 (MINT)

2018-08-30 13:28:35,557 INFO [main] TSDMain: Built on 2015/05/07 04:49:35 +0000 by hobbes@clhbase:/home/hobbes/opentsdb\_OFFICIAL/build

2018-08-30 13:28:35,592 INFO [main] Config: Successfully loaded configuration file: opentsdb.conf

2018-08-30 13:28:35,839 INFO [main] ZooKeeper: Client environment:zookeeper.version=3.3.6-1366786, built on 07/29/2012 06:22 GMT

2018-08-30 13:28:35,839 INFO [main] ZooKeeper: Client environment:host.name=master1

2018-08-30 13:28:35,839 INFO [main] ZooKeeper: Client environment:java.version=1.8.0\_151

2018-08-30 13:28:35,839 INFO [main] ZooKeeper: Client environment:java.vendor=Oracle Corporation

2018-08-30 13:28:35,839 INFO [main] ZooKeeper: Client environment:java.home=/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.151-1.b12.el7\_4.x86\_64/jre

2018-08-30 13:28:35,839 INFO [main] ZooKeeper: Client environment:java.class.path=.:/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.151-1.b12.el7\_4.x86\_64/lib/dt.jar:/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.151-1.b12.el7\_4.x86\_64/lib/tools.jar:/usr/hdp/2.6.3.0-235/hadoop-mapreduce/hadoop-mapreduce-client-core-2.7.3.2.6.3.0-235.jar:/usr/hdp/2.6.3.0-235/hadoop/hadoop-common-2.7.3.2.6.3.0-235.jar:/usr/hdp/2.6.3.0-235/hadoop/lib/commons-cli-1.2.jar:.:/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.151-1.b12.el7\_4.x86\_64/jre/lib/dt.jar:/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.151-1.b12.el7\_4.x86\_64/jre/lib/tools.jar:.::/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.151-1.b12.el7\_4.x86\_64/jre/lib:/opt/opentsdb-2.1.0/build/../third\_party/hbase/asynchbase-1.6.0.jar:/opt/opentsdb-2.1.0/build/../third\_party/guava/guava-18.0.jar:/opt/opentsdb-2.1.0/build/../third\_party/slf4j/log4j-over-slf4j-1.7.7.jar:/opt/opentsdb-2.1.0/build/third\_party/logback/logback-classic-1.0.13.jar:/opt/opentsdb-2.1.0/build/../third\_party/logback/logback-core-1.0.13.jar:/opt/opentsdb-2.1.0/build/../third\_party/jackson/jackson-annotations-2.4.3.jar:/opt/opentsdb-2.1.0/build/third\_party/jackson/jackson-core-2.4.3.jar:/opt/opentsdb-2.1.0/build/../third\_party/jackson/jackson-databind-2.4.3.jar:/opt/opentsdb-2.1.0/build/../third\_party/netty/netty-3.9.4.Final.jar:/opt/opentsdb-2.1.0/build/../third\_party/protobuf/protobuf-java-2.5.0.jar:/opt/opentsdb-2.1.0/build/../third\_party/slf4j/slf4j-api-1.7.7.jar:/opt/opentsdb-2.1.0/build/../third\_party/suasync/async-1.4.0.jar:/opt/opentsdb-2.1.0/build/../third\_party/zookeeper/zookeeper-3.3.6.jar:/opt/opentsdb-2.1.0/build/tsdb-2.1.0.jar:/opt/opentsdb-2.1.0/build/../src

2018-08-30 13:28:35,839 INFO [main] ZooKeeper: Client environment:java.library.path=/usr/java/packages/lib/amd64:/usr/lib64:/lib64:/lib:/usr/lib

2018-08-30 13:28:35,839 INFO [main] ZooKeeper: Client environment:java.io.tmpdir=/tmp

2018-08-30 13:28:35,839 INFO [main] ZooKeeper: Client environment:java.compiler=<NA>

2018-08-30 13:28:35,839 INFO [main] ZooKeeper: Client environment:os.name=Linux

2018-08-30 13:28:35,839 INFO [main] ZooKeeper: Client environment:os.arch=amd64

2018-08-30 13:28:35,839 INFO [main] ZooKeeper: Client environment:os.version=3.10.0-693.5.2.el7.x86\_64

2018-08-30 13:28:35,839 INFO [main] ZooKeeper: Client environment:user.name=root

2018-08-30 13:28:35,839 INFO [main] ZooKeeper: Client environment:user.home=/root

2018-08-30 13:28:35,839 INFO [main] ZooKeeper: Client environment:user.dir=/opt/opentsdb-2.1.0/build

2018-08-30 13:28:35,840 INFO [main] ZooKeeper: Initiating client connection, connectString=master1:2181,master2:2181,slave1:2181 sessionTimeout=5000 watcher=org.hbase.async.HBaseClient$ZKClient@18be83e4

2018-08-30 13:28:35,860 INFO [main-SendThread()] ClientCnxn: Opening socket connection to server slave1/169.254.51.103:2181

2018-08-30 13:28:35,863 INFO [main] HBaseClient: Need to find the -ROOT- region

2018-08-30 13:28:36,007 INFO [main-SendThread(slave1:2181)] ClientCnxn: Socket connection established to slave1/169.254.51.103:2181, initiating session

2018-08-30 13:28:36,036 INFO [main-SendThread(slave1:2181)] ClientCnxn: Session establishment complete on server slave1/169.254.51.103:2181, sessionid = 0x3658949d1600003, negotiated timeout = 5000

2018-08-30 13:28:36,060 INFO [main-EventThread] HBaseClient: Connecting to .META. region @ 169.254.51.101:16020

2018-08-30 13:28:36,108 INFO [main-EventThread] ZooKeeper: Session: 0x3658949d1600003 closed

2018-08-30 13:28:36,109 INFO [main-EventThread] ClientCnxn: EventThread shut down

2018-08-30 13:28:36,204 INFO [New I/O worker #4] HBaseClient: Added client for region RegionInfo(table="tsdb", region\_name="tsdb,,1535438259831.29d1984f0f15b2ff420b06b95980c0a6.", stop\_key=""), which was added to the regions cache. Now we know that RegionClient@153452883(chan=[id: 0xa85a2a68, /169.254.51.101:45438 => /169.254.51.101:16020], #pending\_rpcs=0, #batched=0, #rpcs\_inflight=1) is hosting 1 region.

2018-08-30 13:28:36,206 INFO [New I/O worker #4] HBaseClient: Added client for region RegionInfo(table="tsdb-uid", region\_name="tsdb-uid,,1535438238451.3bb94f5cce9cf0ace0eeb1dab4e3c1c8.", stop\_key=""), which was added to the regions cache. Now we know that RegionClient@153452883(chan=[id: 0xa85a2a68, /169.254.51.101:45438 => /169.254.51.101:16020], #pending\_rpcs=0, #batched=0, #rpcs\_inflight=1) is hosting 2 regions.

2018-08-30 13:28:36,315 INFO [main] RpcHandler: TSD is in rw mode

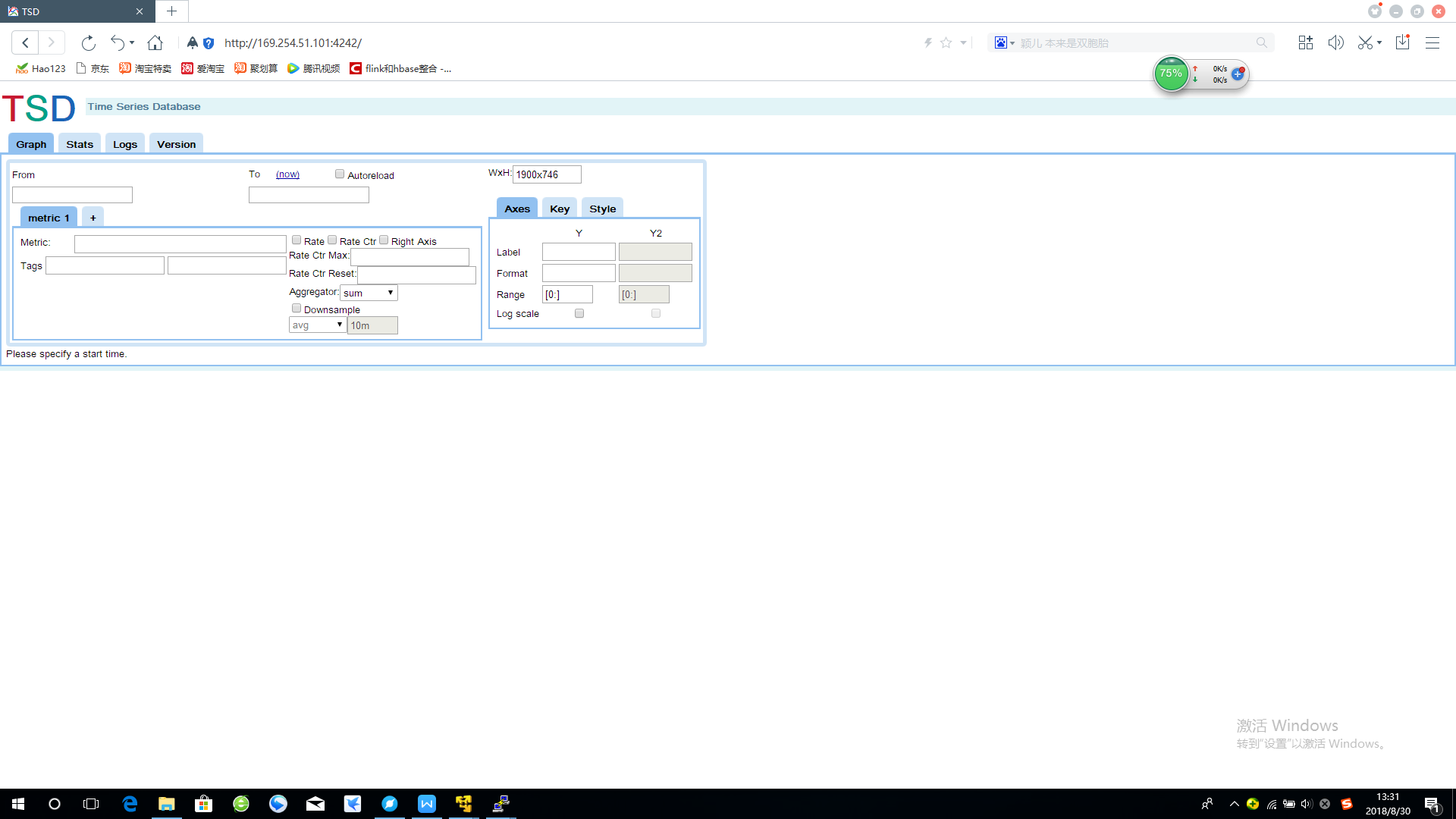
2018-08-30 13:28:36,315 INFO [main] RpcHandler: CORS domain list was empty, CORS will not be enabled

2018-08-30 13:28:36,317 INFO [main] RpcHandler: Loaded CORS headers (Authorization, Content-Type, Accept, Origin, User-Agent, DNT, Cache-Control, X-Mx-ReqToken, Keep-Alive, X-Requested-With, If-Modified-Since)

2018-08-30 13:28:36,390 WARN [main] PluginLoader: Unable to locate any plugins of the type: net.opentsdb.tsd.HttpSerializer

2018-08-30 13:28:36,438 INFO [main] TSDMain: Ready to serve on /0.0.0.0:4242

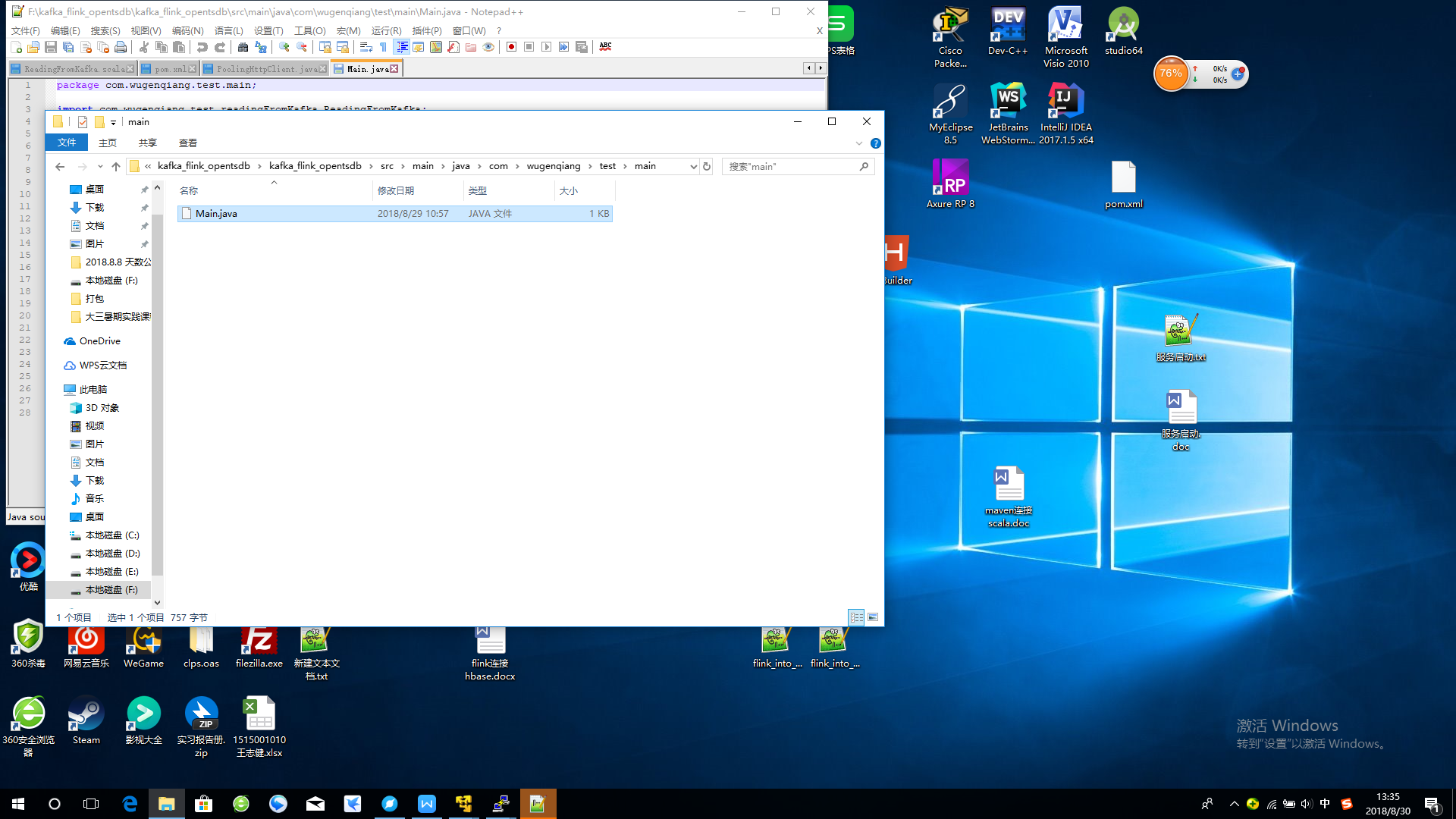
命令行显示服务已经打开，查看4242端口



## 在flink的bin目录下提交计划

[root@master1 bin]# ./flink run -c com.wugenqiang.test.main.Main /opt/kafka\_flink\_opentsdb-1.0-SNAPSHOT-jar-with-dependencies.jar

(这里的com.wugenqiang.test.main.Main是java中mainclass的路径，/opt/kafka\_flink\_opentsdb-1.0-SNAPSHOT-jar-with-dependencies.jar则是你打完的包的路径)



图：Mainclass路径

Cluster configuration: Standalone cluster with JobManager at master1/169.254.51.101:6123

Using address master1:6123 to connect to JobManager.

JobManager web interface address http://master1:8081

Starting execution of program

Submitting job with JobID: d1bf56a736fb88ad4d2bd779c2c7d2a5. Waiting for job completion.

Connected to JobManager at Actor[akka.tcp://flink@master1:6123/user/jobmanager#-338469840] with leader session id 00000000-0000-0000-0000-000000000000.

08/30/2018 14:16:35 Job execution switched to status RUNNING.

08/30/2018 14:16:35 Source: Custom Source(1/1) switched to SCHEDULED

08/30/2018 14:16:35 Map -> Sink: Unnamed(1/1) switched to SCHEDULED

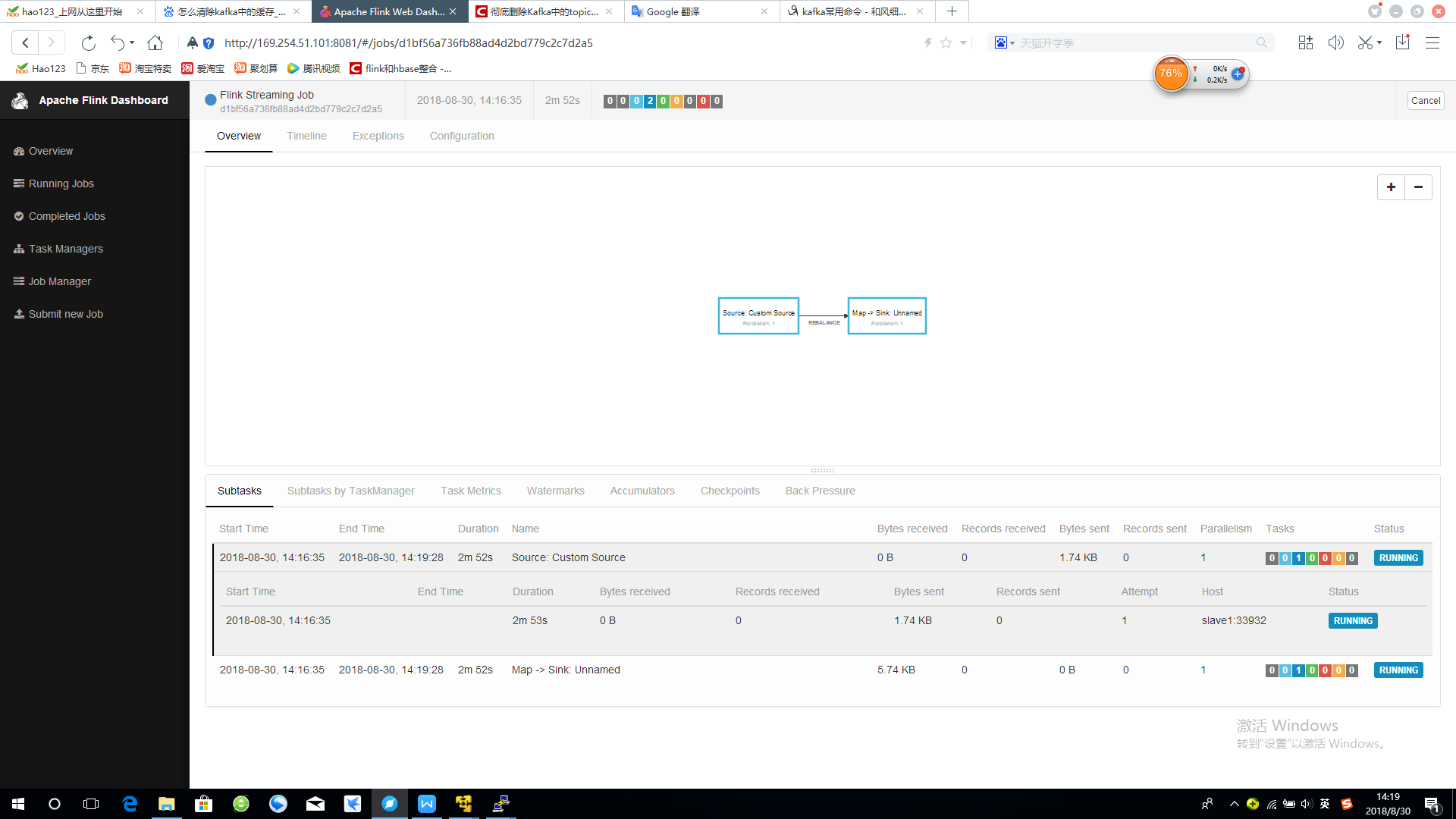
08/30/2018 14:16:35 Source: Custom Source(1/1) switched to DEPLOYING

08/30/2018 14:16:35 Map -> Sink: Unnamed(1/1) switched to DEPLOYING

08/30/2018 14:16:36 Source: Custom Source(1/1) switched to RUNNING

08/30/2018 14:16:36 Map -> Sink: Unnamed(1/1) switched to RUNNING

验证：进程及WebUI8081



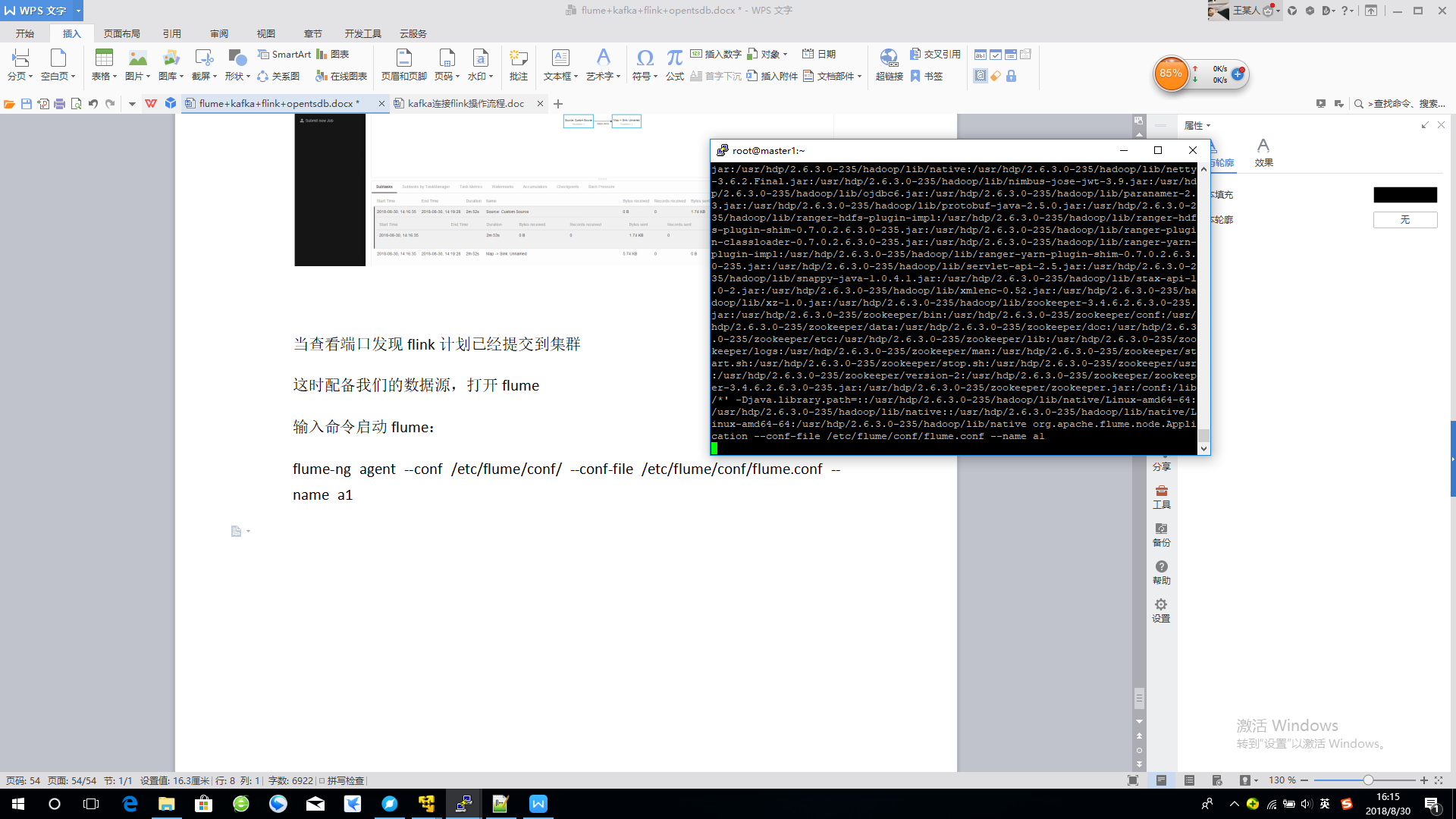
当查看端口发现flink计划已经提交到集群

这时配备我们的数据源，打开flume

输入命令启动flume：

flume-ng agent --conf /etc/flume/conf/ --conf-file /etc/flume/conf/flume.conf --name a1

启动成功后如图：



这个时候我们自己来创造一些模拟数据，我自己写了个脚本dataCreate.sh如下：

#!/bin/bash

step=1

RANDOM\_MAX\_INT=32768

function random\_float() {

fanNumber\_list=("001" "002" "003")

model\_list=("X" "Y" "Z")

local environmentmin=23

local environmentmax=36

local fuselageTempmin=68

local fuselageTempmax=76

local windSpeedmin=20

local windSpeedmax=23

local powerGeneramin=100

local powerGeneramax=102

#风机编号

fanNumber\_num=$(((RANDOM%3)))

fanNumber=${fanNumber\_list[$fanNumber\_num]}

#机型

model\_num=$(((RANDOM%3)))

model=${model\_list[$model\_num]}

#环境温度

local random1=$( echo "scale=2;${environmentmin}+${RANDOM}/${RANDOM\_MAX\_INT}\*(${environmentmax}-${environmentmin}+1)" | bc -l )

#机身温度

local random2=$( echo "scale=2;${fuselageTempmin}+${RANDOM}/${RANDOM\_MAX\_INT}\*(${fuselageTempmax}-${fuselageTempmin}+1)" | bc -l )

#风速

local random3=$( echo "scale=2;${windSpeedmin}+${RANDOM}/${RANDOM\_MAX\_INT}\*(${windSpeedmax}-${windSpeedmin}+1)" | bc -l )

#发电量

local random4=$( echo "scale=2;${powerGeneramin}+${RANDOM}/${RANDOM\_MAX\_INT}\*(${powerGeneramax}-${powerGeneramin}+1)" | bc -l )

mylist[0]=$fanNumber

mylist[1]=$model

mylist[2]=$random1

mylist[3]=$random2

mylist[4]=$random3

mylist[5]=$random4

current=`date "+%Y-%m-%d %H:%M:%S"`

timeStamp=`date -d "$current" +%s`

#currentTimeStamp=$((timeStamp\*1000+`date "+%N"`/1000000))

echo "${timeStamp} ${mylist[\*]}" >> /var/log/windpower/windpower.log

}

while [ 1 ]

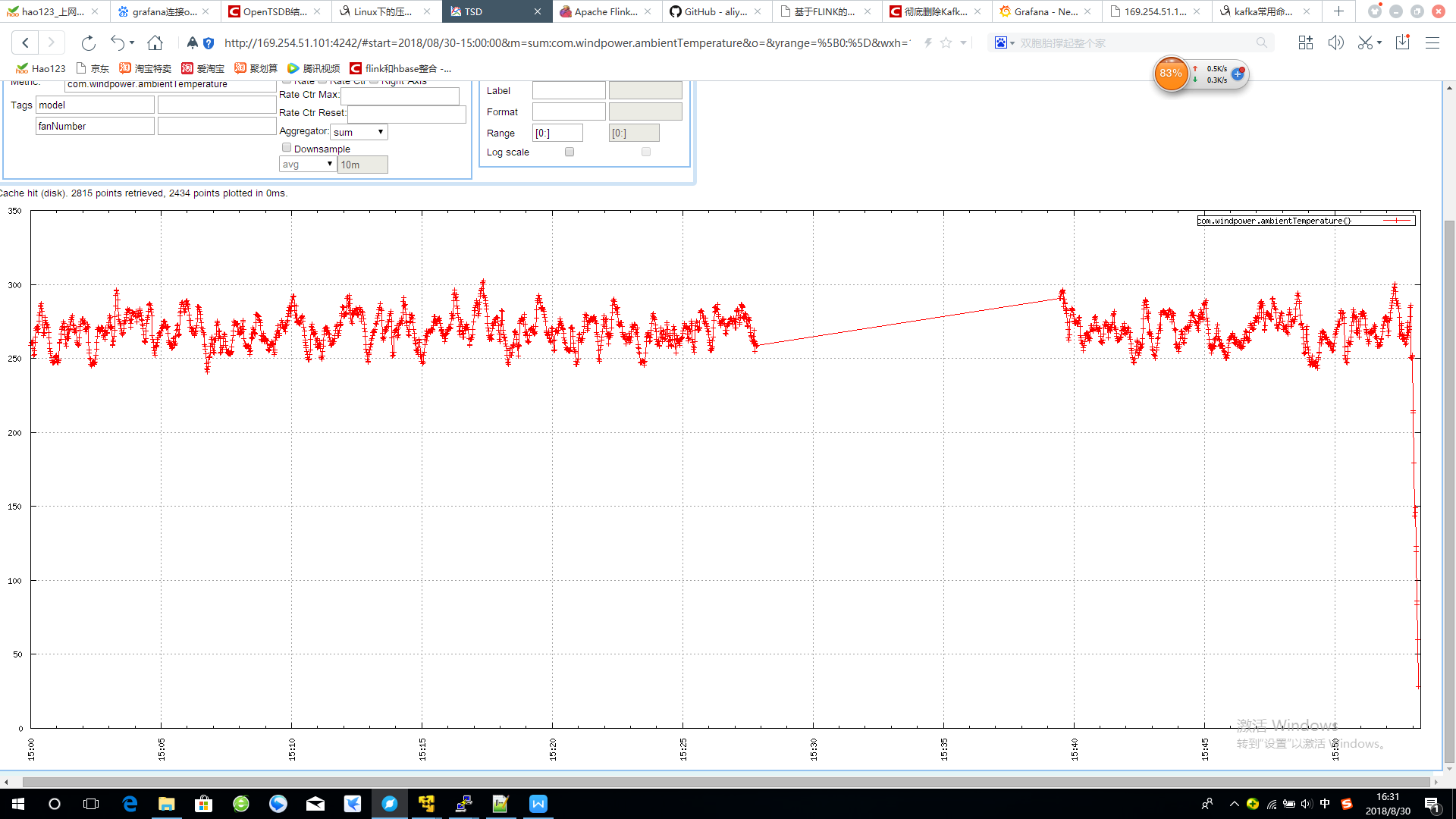
do

random\_float

sleep $step

done

运行之后打开4242端口



可以看到数据已经展示出来

但是此时的数据展示还不够酷，于是我们在安装一个可视化工具去实时的查看数据

# 十四、Grafana部署安装

1. 下载grafana压缩包

|  |
| --- |
| [root@master ~]# wget https://s3-us-west-2.amazonaws.com/grafana-releases/release/grafana-5.2.2-1.x86\_64.rpm |

1. 解压到指定目录

|  |
| --- |
| yum localinstall grafana-5.2.2-1.x86\_64.rpm |

1. 为了使用饼状图，我们需要安装Pie chart插件

|  |
| --- |
| [root@master ~]# grafana-cli plugins install grafana-piechart-panel |

1. 安装日志服务插件

|  |
| --- |
| [root@master ~]# cd /var/lib/grafana/plugins/  [root@master plugins]# git clone https://github.com/aliyun/aliyun-log-grafana-datasource-plugin |

1. 安装好插件后重启grafana-server

|  |
| --- |
| [root@master ~]# service grafana-server restart |

设置grafana服务在机器启动时自启

|  |
| --- |
| [root@master ~]# systemctl enable grafana-server.service |

查看启动是否成功

|  |
| --- |
| systemctl status grafana-server |

1. 登录查看

访问Grafana WEB页面结合OpenTSDB进行视图展现

默认端口：3000

假设是在本机部署，默认是安装在3000端口，在浏览器打开3000端口。



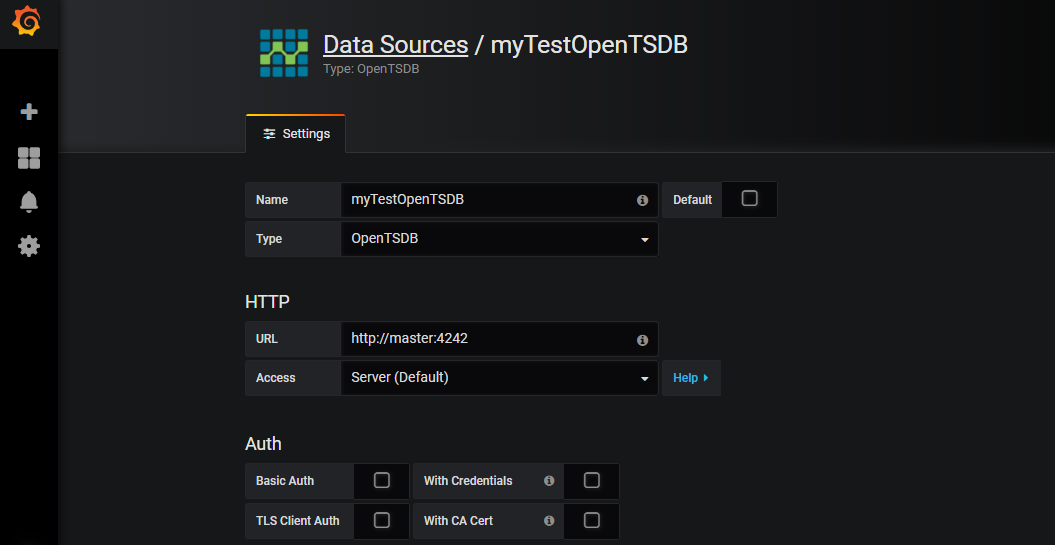
登录名密码默认admin

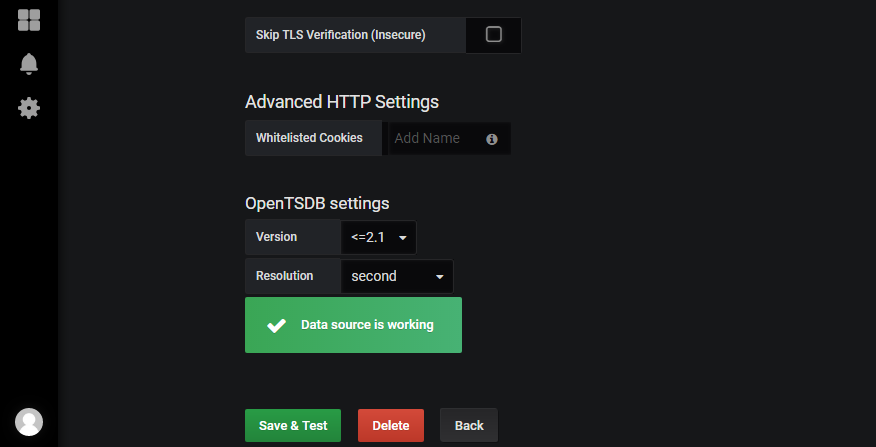
我修改的自己密码123456

1. 为grafana配置数据源为opentsdb

grafana默认admin账号

点击DataSource > add DataSource.填写相关的信息





1. 创建个图表,并配置告警

