

搜狗用户查询行为分析

1. 选题说明与数据来源

1.1 选题说明

随着移动互联的不断壮大、网络的不断普及、提速降费的政策出台，有着越来越多的移动终端进入到了寻常百姓家。这就导致了我国网民用户数量不断增长，各大网站的用户活跃度不断攀升，日志数据的体量以指数级不断加大。为了能够更好的服务于用户，对已有的日志数据分析是十分必要的，对每一个用户建立其独有的用户画像，为其量身打造推荐系统，现已成为各大网站的趋势所在。此外，通过对数据的实时分析、离线分析，可对当前用户所关注的话题加以监控，对网站用户访问量加以统计。上述这些可用于网站的舆情监控、服务器的负载均衡等，这些功能对搜索引擎网站起着至关重要的作用。故此次实训，我们选择了搜狗用户查询行为分析这一项目，以更好的体会大数据架构，及其对于流数据与离线数据的处理过程，并在能力范围内，尽可能对其日志数据加以分析并可视化。

1.2 数据来源

本项目数据来自[搜狗实验室 \(Sogou Labs\)](#) 语料数据中的用户查询日志，搜索引擎查询日志库设计为包括约 1 个月（2008 年 6 月）Sogou 搜索引擎部分网页查询需求及用户点击情况的网页查询日志数据集。

2. 小组成员与分工

王功：环境搭建、集群配置、组件安装、组件间集成开发、资料收集

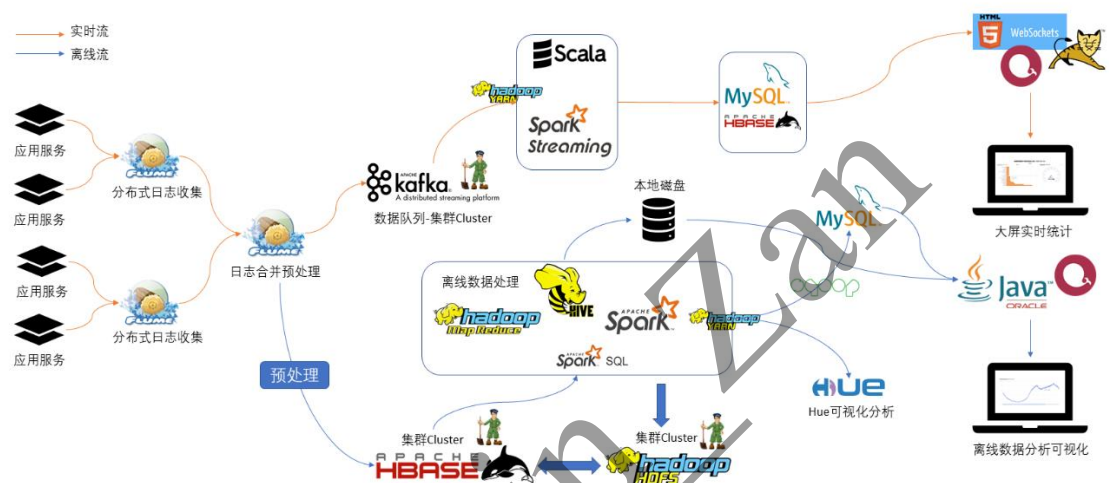
彭赞：环境搭建、集群配置、组件安装、组件间集成开发、文档梳理编写

郑书文：环境搭建、集群配置、组件安装、PPT 展示、数据可视化

3. 主体设计思路

搜狗用户查询行为分析项目主要分为实时流与离线流两个部分，实时流主要通过 Kafka 组件对所接收到的流数据进行分发传递，Spark streaming 对 Kafka

所分发的数据进行实时处理，后续得到的结果将储存在 MySQL 或 HBASE 数据库中，以便后续通过 WebSocket 相关服务对其数据进行提取并进行可视化。离线流主要以 Spark、MapReduce 相关组件对储存在 Hive 或 HBASE 中的离线数据进行分析统计，将结果储存在 MySQL 数据库，并利用 JDBC 等进行数据的展示与可视化。另外，也可利用 Hue 对每个数据库进行集成，以便与后期的分析与可视化操作。项目底层架构依托 Hadoop 生态系统相关组件提供支持，具体架构内容后续会在大数据系统架构部分予以介绍。下图为搜狗用户查询行为分析项目数据的流程图。



通过上述的主体思路，意在达成、实现、确立以下 3 个目标、6 个功能、2 种分析指标：

目标：

- ✓ 完成大数据项目的架构设计、安装部署、集成开发，以及用户可视化交互
- ✓ 完成实时在线数据分析
- ✓ 完成离线数据分析

功能：

- ⌘ 捕获用户浏览日志
- ⌘ 实现 Hadoop 分布式集群的高可用
- ⌘ 完成了各组件之间的集成交互
- ⌘ 实时分析当前热度前 20 的新闻话题，以及当前线上已曝光话题数量

⌘ 离线分析当日的用户和网站的活跃情况

⌘ 对分析结果进行可视化展示

分析指标：（话题，网站，用户维度）

➤ 实时指标

- 实时话题热度排行榜前 20
- 实时已曝光的话题数量

➤ 离线指标

- 当日各时段话题数量统计
- 当日各时段用户活跃度统计
- 当日话题关键词词云
- 当日网站热度词云

4. 具体实现过程与步骤

4.1 大数据系统架构

搜狗用户查询行为分析项目系统架构主要由多个层次组成，它们分别为数据源层、采集层、存储层、计算层、服务层、接口层、展示层、调度层和管理工具。

具体如图 2 所示。



图 2 搜狗用户查询行为分析项目系统架构

数据源层是这个数据分析的起点，它的重要性不可小视，数据内容与结构是否符合当前的分析项目，决定了后续操作能否顺利进行。DB 代表数据库文件，LogFile 代表日志记录文件。

采集层主要对数据源层的数据加以采集，其中 LogFile 我们采用日志采集框架 Flume 对其进行采集分发等操作，DB 文件通过 Sqoop 数据处理工具进行处理。

存储层是对数据进行持久化保存的关键，分布式消息队列 Kafka 可接收 Flume 传递过来的数据，并对其存储在缓存中。分布式文件系统 HDFS 可为 Hive、HBASE 提供底层数据存储支撑，MySQL 主要存储后续的分析结果与 Hive 的元数据表。

计算层主要对数据进行计算处理分析，利用 YARN 统一资源调度管理，输入存储层的数据，输出处理后的结果。计算层可大致分为两个部分，实时计算与离线计算。实时计算部分利用在线计算框架 Spark Streaming 对 Kafka 所广播的流数据加以计算处理。离线计算部分主要利用离线计算框架 MapReduce、内存计算框架 Spark 联动 MySQL、Hive 等组件进行数据的批量处理。

服务层主要是利用 Java 或 Scala 语言对之前计算层所得到的结果加以读取、二次处理、传递等操作。

接口层主要功能在于为展示层提供网络协议接口或数据协议接口，以便于数据以特定的协议传递。

展示层利用 Hue、Html5+Echarts 的方式对分析结果进行可视化，使其更加便于用户理解。

调度层由组件 Zookeeper 承担主要任务，在整个分析流程中起着至关重要的作用。

管理工具主要由开发工具 IDEA、分析工具 Hue 组成。通过与 HBase、Hive 等组件的集成，Hue 可为数据分析发挥更加人性化的特性。

为更好的了解与学习 Hadoop 生态系统分布式的相关内容，故利用虚拟机的方式计划搭建如图 4 所示的集群。其中主要的突破点在于，我们尝试利用 Zookeeper 实现了 Hadoop 集群的高可用，通过对 Flume 源代码的修改以实现不同业务场景下的定制化，通过主动编译 Spark 源代码以实现组件的兼容。此外，根据不同节点的负载情况，对不同的组件，赋予其不同的安装位置。

利用 Zookeeper 实现 Hadoop 集群的高可用

当发生故障时，Active 的 NN 挂掉后，Standby NN 会在它成为 Active NN 前，读取所有的 JN 里面的修改日志，这样就能高可靠的保证与挂掉的 NN 的目录镜像树一致，然后无缝的接替它的职责，维护来自客户端请求，从而达到一个高可用的目的。

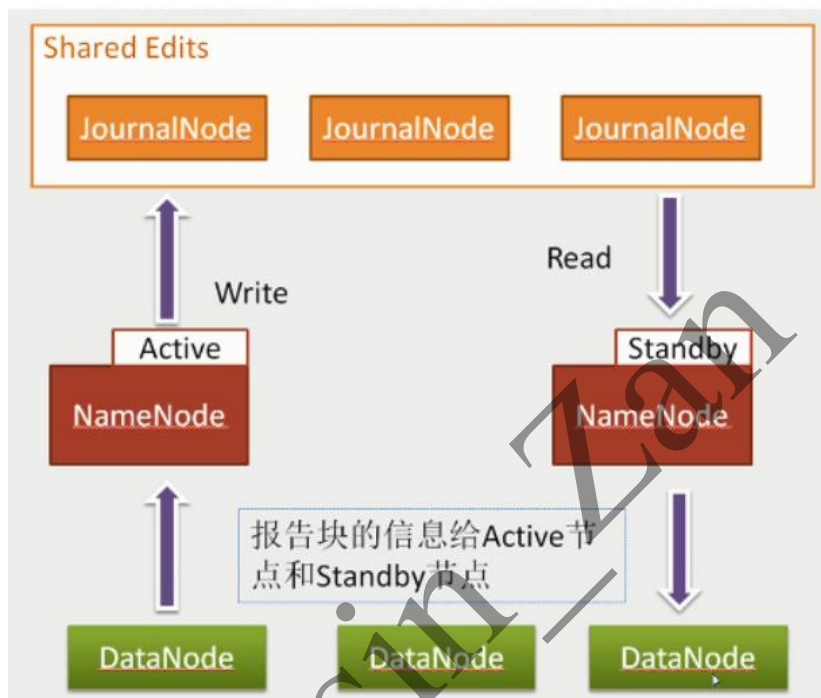


图 3 Hadoop 集群的高可用

具体实现：主要包括主备切换，共享日志存储

在集群中存在多个 namenode，这些 namenode 处于 active 或 standby 状态

共享日志存储：active namenode 向共享日志存储系统里写如日志文件，standby namenode 读取日志文件和 active 保持同步。共享日志存储系统一般采用的是 Quorum Journal (群体日志) 设计方案。

主备切换：每一个 namenode 运行着轻量级的 ZKFC 即 ZKFailoverController（相当于 zookeeper 的客户端）。ZKFC 主要包括两部分 HealthMonitor 和 **ActiveStandbyElector**。HealthMonitor 启动内部线程定时调用 NameNode 的 HATServiceProtocol RPC 接口 (monitor Health 和 getServiceStatus)，监控 NameNode 的健康状态并向 ZKFC 反馈；如果检测到 namenode 健康状态发生变化，ZKFC 调用 ActiveStandbyElector 与 zookeeper 集群交互完成自动的主备选举，之后回调 ZKFC 的相应方法与 namenode 进行 RPC 通信，完成主备切换。

集群资源规划：

主机 组件	master (192.168.1.201)	slaver01 (192.168.1.202)	slaver02 (192.168.1.203)
HDFS	NameNode DataNode	NameNode DataNode	DataNode
YARN	ResourceManager NodeManager	NodeManager	NodeManager
Zookeeper	Zookeeper	Zookeeper	Zookeeper
Kafka	Kafka	Kafka	Kafka
HBASE	HMaster RegionServer	HMaster RegionServer	RegionServer
Flume	Flume	Flume	Flume
Hive			Hive
MySQL	MySQL		
Spark	Spark	Spark	Spark
Hue			Hue
Sqoop			Sqoop

图 4 集群资源规划图

至此，该项目的大体流程与架构暂且介绍完毕。

4.2 具体实现过程与步骤

1) Linux 环境准备与设置

操作系统及软件环境：Windows 10_20H2(64 位)、CentOS 2.6.32-358.el6.x86_64、VMware Workstation 16、Xshell、Xftp

利用 VMware Workstation 加载 Linux 系统镜像并安装，由于网络上这部分教程较为繁多，故在这里不做过多赘述。下面将直接讲述 Linux 系统常规配置部分。

设置 IP 地址

如果在安装 Linux 时安装了图形操作界面，可直接在网络配置图形界面进行设置。若没有安装图形界面，需要使用命令：`vi /etc/sysconfig/network-scripts/ifcfg-eth0` 来修改 IP 地址，然后使用命令 `service network restart` 重启网络服务即可。

创建用户

大数据项目开发中，一般不直接使用 root 用户，需要我们创建新的用户来操作，比如 kfk。

a) 创建用户命令：`adduser kfk`

b) 设置用户密码命令：`passwd kfk`

设置主机名

Linux 系统的主机名默认是 localhost，显然不方便后面集群的操作，我们需要手动修改 Linux 系统的主机名。

a) 查看主机名命令：hostname

b) 修改主机名称

```
vi /etc/sysconfig/network
```

```
NETWORKING=yes
```

```
HOSTNAME=master
```

主机名映射

如果想通过主机名访问 Linux 系统，还需要配置主机名跟 IP 地址之间的映射关系。

```
vi /etc/hosts
```

```
192.168.1.201 master
```

配置完成之后，reboot 重启 Linux 系统即可。

如果需要在 Windows 也能通过 hostname 访问 Linux 系统，也需要在 Windows 下的 hosts 文件中配置主机名称与 IP 之间的映射关系。在 Windows 系统下找到 C:\WINDOWS\system32\drivers\etc\路径，打开 HOSTS 文件添加如下内容：

```
192.168.1.201 master
```

root 用户下设置无密码用户切换

在 Linux 系统中操作是，kfk 用户经常需要操作 root 用户权限下的文件，但是访问权限受限或者需要输入密码。修改/etc/sudoers 这个文件添加如下代码，即可实现无密码用户切换操作。

```
vi /etc/sudoers
```

```
#添加如下内容即可
```

```
kfk ALL=(root)NOPASSWD:ALL
```

关闭防火墙

我们都知道防火墙对我们的服务器是进行一种保护，但是有时候防火墙也会给我们带来很大的麻烦。比如它会妨碍 Hadoop 集群间的相互通信，所以我们需要关闭防火墙。那么我们永久关闭防火墙的方法如下：

```
vi /etc/sysconfig/selinux
```

SELINUX=disabled

注：此处操作具有一定的危险性，在修改配置文件时应格外注意配置项的对应关系，不要随意删除其他配置项，也不要随意修改其他配置项，否则会发生系统灾难性后果，出现无法引导启动的情况。

保存、重启后，验证机器的防火墙是否已经关闭。

a) 查看防火墙状态：service iptables status

b) 打开防火墙：service iptables start

c) 关闭防火墙：service iptables stop

卸载 Linux 本身自带的 JDK

一般情况下 jdk 需要我们手动安装兼容的版本，此时 Linux 自带的 jdk 需要手动删除掉，具体操作如下所示：

a) 查看 Linux 自带的 jdk

```
rpm -qa|grep java
```

b) 删除 Linux 自带的 jdk

```
rpm -e --nodeps [jdk 进程名称 1 jdk 进程名称 2 ...]
```

克隆虚拟机并进行相关的配置

前面我们已经做好了 Linux 的系统常规设置，接下来需要克隆虚拟机并进行相关的配置。

kfk 用户下创建我们将要使用的各个目录

#软件目录

```
mkdir /opt/software
```

#模块目录

```
mkdir /opt/modules
```

#工具目录

```
mkdir /opt/tools
```

#数据目录

```
mkdir /opt/datas
```

JDK 安装

大数据平台运行环境依赖 JVM，所以我们需要提前安装和配置好 jdk。 前面

我们已经安装了 64 位的 centos 系统，所以我们的 jdk 也需要安装 64 位的，与之相匹配

a) 将 jdk 安装包通过工具上传到/opt/software 目录下

b) 解压 jdk 安装包

#解压命令

```
tar -zxvf jdk-7u67-linux-x64.tar.gz /opt/modules/
```

#查看解压结果

```
ls
```

```
jdk1.7.0_67
```

c) 配置 Java 环境变量

```
vi /etc/profile
```

```
export JAVA_HOME=/opt/modules/jdk1.7.0_67
```

```
export PATH=$PATH:$JAVA_HOME/bin
```

d) 查看 Java 是否安装成功

```
java -version
```

```
java version "1.7.0_67"
```

```
Java(TM) SE Runtime Environment (build 1.7.0_67-b15)
```

```
Java HotSpot(TM) 64-Bit Server VM (build 24.79-b02, mixed mode)
```

克隆虚拟机

在克隆虚拟机之前，需要关闭虚拟机，然后右键选中虚拟机——》选择管理——》选择克隆——》选择下一步——》选择下一步——》选择创建完整克隆，下一步——》选择克隆虚拟机位置（提前创建好），修改虚拟机名称为 slaver01，然后选择完成即可。

然后使用同样的方式创建第三个虚拟机 slaver02。

修改克隆虚拟机配置

克隆完虚拟机 slaver01 和 slaver02 之后，可以按照 master 的方式配置好 ip 地址、hostname，以及 ip 地址与 hostname 之间的关系。

修改克隆虚拟机的 MAC 地址，以防止出现网络冲突。在配置文件中修改静态 ip、dns、mac 地址，写入 ONBOOT=yes，以便于局域网内连接、外网访问、开机

时启动网卡。(配置文件: /etc/sysconfig/network-scripts/ifcfg-ethX, ethX 可在 ifconfig 命令后查询到网卡序号)

注: 目前利用多台虚拟机搭建 Hadoop 集群时发现, 利用虚拟机的桥接网络方式可使各节点间联络更加的通畅。前提是需要自备一台性能良好的物理路由设备。

2) Zookeeper 分布式集群部署

ZooKeeper 是一个针对大型分布式系统的可靠协调系统; 它提供的功能包括: 配置维护、名字服务、分布式同步、组服务等; 它的目标就是封装好复杂易出错的关键服务, 将简单易用的接口和性能高效、功能稳定的系统提供给用户; ZooKeeper 已经成为 Hadoop 生态系统中的基础组件。

下载 Zookeeper-3.4.5.tar.gz 归档文件到/opt/softwares 目录, 解压其到 /opt/modules 文件夹, 并修改 Zookeeper 相关配置

- a) 复制配置文件 cp conf/zoo_sample.cfg zoo.cfg
- b) 修改配置文件 zoo.cfg

```
# The number of milliseconds of each tick
tickTime=2000
# The number of ticks that the initial
# synchronization phase can take
initLimit=10
# The number of ticks that can pass between
# sending a request and getting an acknowledgement
syncLimit=5
# the directory where the snapshot is stored.
# do not use /tmp for storage, /tmp here is just
# example sakes.
dataDir=/opt/modules/zookeeper-3.4.5/zkData
dataLogDir=/opt/modules/zookeeper-3.4.5/logs

server.1=master:2888:3888
server.2=slaver01:2888:3888
server.3=slaver02:2888:3888
# the port at which the clients will connect
clientPort=2181
```

将 Zookeeper 安装配置分发到其他两个节点，具体操作如下所示：

```
scp -r zookeeper-3.4.5/ slaver01:/opt/modules/
```

```
scp -r zookeeper-3.4.5/ slaver02:/opt/modules/
```

接下来创建相关目录和文件

a) 在 3 个节点上分别创建数据目录

```
mkdir /opt/modules/zookeeper-3.4.5/zkData
```

b) 在各个节点的数据存储目录下创建 myid 文件，并且分别编辑每个机器的 myid 内容为 1、2、3

启动 Zookeeper 服务进行测试。

3) Hadoop 2.X 分布式 HA 架构与部署

配置 SSH 无密钥登录，前提：各节点已经配置好 hosts 文件

输入命令：`cd .ssh` 进入 rsa 公钥私钥文件存放的目录，删除目录下的 `id_rsa`，`id_rsa.pub` 文件。

在每台机上产生新的 rsa 公钥私钥文件，并统一拷贝到一个 `authorized_keys` 文件中

a) 登录 master，在 .ssh 目录下输入命令：`ssh-keygen -t rsa`，三次回车后，该目录下将会产生 `id_rsa`，`id_rsa.pub` 文件。其他主机也使用该方式产生密钥文件。

b) 登录 master，输入命令：`cat id_rsa.pub >> authorized_keys`，将 `id_rsa.pub` 公钥内容拷贝到 `authorized_keys` 文件中。

c) 登录其他主机，将其他主机的公钥文件内容都拷贝到 master 主机上的 `authorized_keys` 文件中，命令如下：

```
ssh-copy-id -i master #登录 slaver01, 将公钥拷贝到 master 的 authorized_keys 中
```

```
ssh-copy-id -i master#登录 slaver02, 将公钥拷贝到 master 的 authorized_keys 中
```

授权 `authorized_keys` 文件

a) 登录 master，在 .ssh 目录下输入命令：`chmod 600 authorized_keys`

将授权文件分配到其他主机上

a) 登录 hadoop01, 将授权文件拷贝到 slaver01、slaver02..., 命令如下:

```
scp /root/.ssh/authorized_keys slaver01:/root/.ssh/ #拷贝到 slaver01 上
```

```
scp /root/.ssh/authorized_keys slaver02:/root/.ssh/ #拷贝到 slaver02 上
```

至此, 免密码登录已经设定完成, 注意第一次 ssh 登录时需要输入密码, 再次访问时即可免密码登录。

测试 HDFS

ssh 无密钥登录做好之后, 可以在主节点通过一键启动命令, 启动 hdfs 各个节点的服务, 具体操作如下所示:

```
sbin/start-dfs.sh
```

如果 yarn 和 hdfs 主节点共用, 配置一个节点即可。否则, yarn 也需要单独配置 ssh 无密钥登录。

选择一台机器作为时间服务器, 比如 master 节点。查看 ntp 服务是否存在, 并查看 ntp 状态。若正常, 则设置 ntp 随机器启动, 若异常, 则重新安装 ntp 程序。

修改 ntp 配置文件/etc/ntp.conf 以实现集群时间同步。

```
#释放注释并将 ip 地址修改为
```

```
restrict 192.168.31.151 mask 255.255.255.0 nomodify notrap
```

```
#注释掉以下命令行
```

```
server 0.centos.pool.ntp.org iburst
```

```
server 1.centos.pool.ntp.org iburst
```

```
server 2.centos.pool.ntp.org iburst
```

```
server 3.centos.pool.ntp.org iburst
```

```
#释放以下命令行
```

```
server 127.127.1.0 #local clock
```

```
fudge 127.127.1.0 stratum 10
```

修改完毕后, 重启 ntp 服务, 修改服务器时间, 并手动同步其他节点时间。

slaver01 和 slaver02 节点可分别切换到 root 用户, 通过 crontab -e 命令, 每 10 分钟同步一次主服务器节点的时间。

#定时，每隔 10 分钟同步 master 服务器时间

0-59/10 * * * * /usr/sbin/ntpdate master

HDFS-HA 详细配置

修改 hdfs-site.xml 配置文件

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!--
Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or impli
ed.
See the License for the specific language governing permissions and
limitations under the License. See accompanying LICENSE file.
-->
<!-- Put site-specific property overrides in this file. -->
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>3</value>
  </property>
  <property>
    <name>dfs.permissions</name>
    <value>>false</value>
  </property>
  <property>
    <name>dfs.permissions.enabled</name>
    <value>>false</value>
  </property>
  <property>
    <name>dfs.nameservices</name>
    <value>ns</value>
  </property>
  <property>
    <name>dfs.ha.namenodes.ns</name>
    <value>nn1,nn2</value>
  </property>
</configuration>
```

```

</property>
<property>
  <name>dfs.namenode.rpc-address.ns.nn1</name>
  <value>master:8020</value>
</property>
<property>
  <name>dfs.namenode.rpc-address.ns.nn2</name>
  <value>slaver01:8020</value>
</property>
<property>
  <name>dfs.namenode.http-address.ns.nn1</name>
  <value>master:50070</value>
</property>
<property>
  <name>dfs.namenode.http-address.ns.nn2</name>
  <value>slaver01:50070</value>
</property>
<property>
  <name>dfs.namenode.shared.edits.dir</name>
  <value>qjournal://master:8485;slaver01;slaver02:8485/ns</value>
</property>
<property>
  <name>dfs.journalnode.edits.dir</name>
  <value>/opt/modules/hadoop-2.5.0/data/jn</value>
</property>
<property>
  <name>dfs.client.failover.proxy.provider.ns</name>
  <value>org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFail
overProxyProvider</value>
</property>
<property>
  <name>dfs.ha.automatic-failover.enabled</name>
  <value>true</value>
</property>
<property>
  <name>dfs.ha.fencing.methods</name>
  <value>sshfence</value>
</property>
<property>
  <name>dfs.ha.fencing.ssh.private-key-files</name>
  <value>/home/kfk/.ssh/id_rsa</value>
</property>
<property>
  <name>dfs.datanode.data.dir</name>

```

```
        <value>file://${hadoop.tmp.dir}/dfs/data</value>
    </property>
</configuration>
```

修改 core-site.xml 配置文件

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!--
Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or impli
ed.
See the License for the specific language governing permissions and
limitations under the License. See accompanying LICENSE file.
-->
<!-- Put site-specific property overrides in this file. -->
<configuration>
    <property>
        <name>fs.defaultFS</name>
        <value>hdfs://ns</value>
    </property>
    <property>
        <name>hadoop.http.staticuser.user</name>
        <value>kfk</value>
    </property>
    <property>
        <name>hadoop.tmp.dir</name>
        <value>/opt/modules/hadoop-2.5.0/data/tmp</value>
    </property>
    <property>
        <name>dfs.namenode.name.dir</name>
        <value>file://${hadoop.tmp.dir}/dfs/name</value>
    </property>
    <property>
        <name>ha.zookeeper.quorum</name>
        <value>master:2181,slaver01:2181,slaver02:2181</value>
```

```
</property>
</configuration>
```

将修改的配置分发到其他节点即可。

HDFS-HA 服务启动及自动故障转移测试

a) 启动所有节点上面的 Zookeeper 进程

```
zkServer.sh start
```

b) 启动所有节点上面的 journalnode 进程

```
sbin/hadoop-daemon.sh start journalnode
```

c) 在[nn1]上, 对 namenode 进行格式化, 并启动

```
#namenode 格式化
```

```
bin/hdfs namenode -format
```

```
#格式化高可用
```

```
bin/hdfs zkfc -formatZK
```

```
#启动 namenode
```

```
bin/hdfs namenode
```

d) 在[nn2]上, 同步 nn1 元数据信息

```
bin/hdfs namenode -bootstrapStandby
```

e) nn2 同步完数据后, 在 nn1 上, 按下 ctrl+c 来结束 namenode 进程。然后关闭所有节点上面的 journalnode 进程

```
sbin/hadoop-daemon.sh stop journalnode
```

f) 一键启动 hdfs 所有相关进程

```
sbin/start-dfs.sh
```

hdfs 启动之后, kill 其中 Active 状态的 namenode, 检查另外一个 NameNode 是否会自动切换为 Active 状态。同时通过命令上传文件至 hdfs, 检查 hdfs 是否可用。

YARN-HA 架构原理及介绍

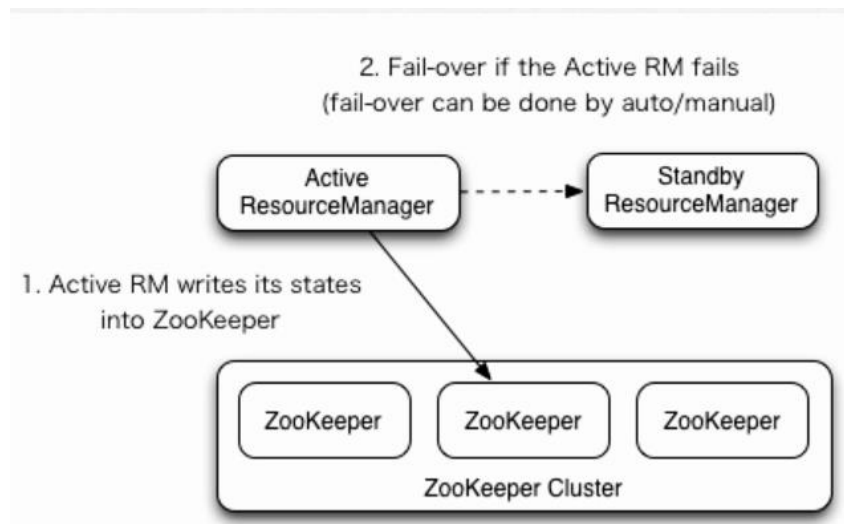


图 5 YARN-HA 架构

ResourceManager HA 由一对 Active, Standby 结点构成, 通过 RMStateStore 存储内部数据和主要应用的数据及标记。目前支持的可替代的 RMStateStore 实现有: 基于内存的 MemoryRMStateStore, 基于文件系统的 FileSystemRMStateStore, 及基于 zookeeper 的 ZKRMStateStore。ResourceManager HA 的架构模式同 NameNode HA 的架构模式基本一致, 数据共享由 RMStateStore, 而 ZKFC 成为 ResourceManager 进程的一个服务, 非独立存在。

YARN-HA 详细配置

修改 mapred-site.xml 配置文件

```
<configuration>
  <property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
  </property>
</configuration>
```

修改 yarn-site.xml 配置文件

```
<configuration>
  <property>
    <name>yarn.resourcemanager.cluster-id</name>
    <value>rs</value>
  </property>
</configuration>
```

```

</property>
<property>
  <name>yarn.resourcemanager.ha.rm-ids</name>
  <value>rm1,rm2</value>
</property>
<property>
  <name>yarn.resourcemanager.hostname.rm1</name>
  <value>master</value>
</property>
<property>
  <name>yarn.resourcemanager.hostname.rm2</name>
  <value>slaver01</value>
</property>
<property>
  <name>yarn.resourcemanager.zk.state-store.address</name>
  <value>master:2181,slaver01:2181,slaver02:2181</value>
</property>
<property>
  <name>yarn.resourcemanager.zk-address</name>
  <value>master:2181,slaver01:2181,slaver02:2181</value>
</property>
<property>
  <name>yarn.resourcemanager.recovery.enabled</name>
  <value>true</value>
</property>
<property>
  <name>yarn.resourcemanager.ha.enabled</name>
  <value>true</value>
</property>
<property>
  <name>yarn.nodemanager.aux-services</name>
  <value>mapreduce_shuffle</value>
</property>
<property>
  <name>yarn.nodemanager.aux-
services.mapreduce_shuffle.class</name>
  <value>org.apache.hadoop.mapred.ShuffleHandler</value>
</property>
</configuration>

```

将修改的配置分发到其他节点即可。

注意：在相关配置文件中，主机名与 hosts 文件中应相同，否则 Hadoop 将

无法启动。在启动 HA 高可用前，需在各节点启动 Zookeeper。

4) HBase 分布式集群部署与设计

HBase 是一个高可靠、高性能、面向列、可伸缩的分布式存储系统，利用 Hbase 技术可在廉价 PC Server 上搭建大规模结构化存储集群。

HBase 是 Google Bigtable 的开源实现，与 Google Bigtable 利用 GFS 作为其文件存储系统类似，HBase 利用 Hadoop HDFS 作为其文件存储系统；Google 运行 MapReduce 来处理 Bigtable 中的海量数据，HBase 同样利用 Hadoop MapReduce 来处理 HBase 中的海量数据；Google Bigtable 利用 Chubby 作为协同服务，HBase 利用 Zookeeper 作为对应。

下载并解压 HBase-0.98.6-cdh5.3.0.tar.gz，接下来将进行分布式集群的相关配置。

HBase 集群规划

Node Name	Master	Zookeeper
master	是	是
slaver01	Backup	是
slaver02	否	是

图 6 HBase 集群规划

分布式集群相关配置

a) hbase-env.sh

#配置 jdk

export JAVA_HOME=/opt/modules/jdk1.7.0_67

#使用独立的 Zookeeper

export HBASE_MANAGES_ZK=false

b) hbase-site.xml

```
<configuration>
  <property>
    <name>hbase.rootdir</name>
    <value>hdfs://ns/hbase</value>
  </property>
  <property>
    <name>hbase.cluster.distributed</name>
```

```

        <value>true</value>
    </property>
</property>
    <name>hbase.zookeeper.quorum</name>
    <value>master,slaver01,slaver02</value>
</property>
</configuration>

```

c) regionservers

master

slaver01

slaver02

d) backup-masters

slaver01

修改后将 hbase 配置分发到各个节点,接着拷贝 Hadoop 中的 hdfs-site.xml core-site.xml 配置文件到 HBase 配置文件目录即可。

启动 HBase 服务并测试,如果各个节点启动正常,那么 HBase 就搭建完毕。

根据业务需求创建表结构,在 HBase shell 环境下输入以下命令:

```
>create 'weblogs','info'
```

5) Kafka 分布式集群部署

Kafka 是由 LinkedIn 开发的一个分布式的消息系统,使用 Scala 编写,它以可水平扩展和高吞吐率而被广泛使用。目前越来越多的开源分布式处理系统如 Cloudera、Apache Storm、Spark 都支持与 Kafka 集成。

下载并解压安装 kafka_2.11-0.10.2.1.tgz

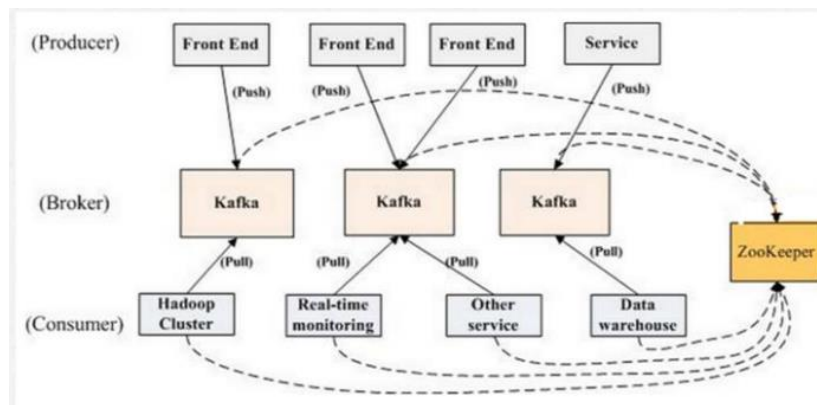


图 7 Kafka 原理示意图

Kafka 集群配置

配置 server.properties 文件

```
#节点唯一标识
broker.id=0

#默认端口号
port=9092

#主机名绑定
host.name=master

#Kafka 数据目录
log.dirs=/opt/modules/kafka_2.11-0.10.2.1/tmp/kafka-logs

#配置 Zookeeper
zookeeper.connect=master:2181,slaver01:2181,slaver02:2181
```

配置 zookeeper.properties 文件

```
#Zookeeper 的数据存储路径与 Zookeeper 集群配置保持一致
dataDir=/opt/modules/zookeeper-3.4.5/zkData
```

配置 consumer.properties 文件

```
#配置 Zookeeper 地址
zookeeper.connect=master:2181,slaver01:2181,slaver02:2181
```

配置 producer.properties 文件

```
#配置 Kafka 集群地址
metadata.broker.list=master:9092,slaver01:9092,slaver02:9092
```

将 Kafka 分发到其他节点，修改另外两个节点的 server.properties。

```
#slaver01 节点
broker.id=1
host.name=slaver01

#slaver02 节点
broker.id=2
host.name=slaver02
```

启动 Kafka 集群并进行测试，各节点见消息生产消费正常，证明已安装完成。

6) Flume 数据采集准备

Flume 是 Cloudera 提供的一个高可用的，高可靠的，分布式的海量日志采集、聚合和传输的系统，Flume 支持在日志系统中定制各类数据发送方，用于收集数据；同时，Flume 提供对数据进行简单处理，并写到各种数据接受方（可定制）的能力。

下载解压并安装 `apache-flume-1.7.0-bin.tar.gz` 并将其分发到其他两个节点

flume agent-1 采集节点服务配置

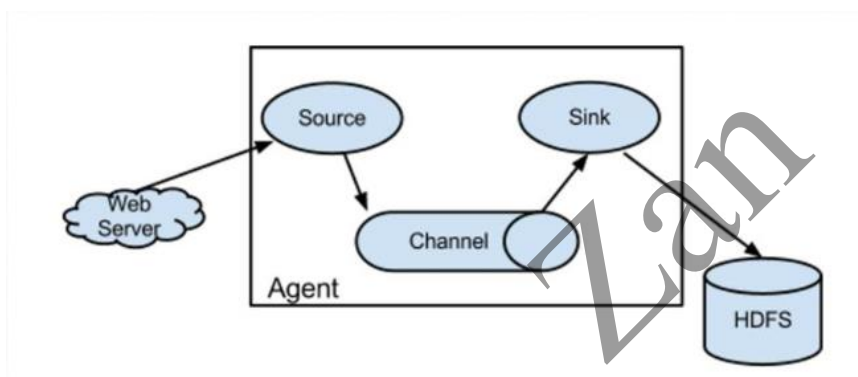


图 8 Flume 数据采集示意图

slaver01 节点修改配置文件 `flume-conf.properties`，目的将数据采集到 master 节点。

```
agent2.sources = r1
agent2.channels = c1
agent2.sinks = k1

agent2.sources.r1.type = exec
agent2.sources.r1.command = tail -F /opt/datas/weblog.log
agent2.sources.r1.channels = c1

agent2.channels.c1.type = memory
agent2.channels.c1.capacity = 10000
agent2.channels.c1.transactionCapacity = 10000
agent2.channels.c1.keep-alive = 5

agent2.sinks.k1.type = avro
agent2.sinks.k1.channel = c1
agent2.sinks.k1.hostname = master
agent2.sinks.k1.port = 5555
```

slaver02 节点修改配置文件 flume-conf.properties，目的将数据采集到 master 节点。

```
agent3.sources = r1
agent3.channels = c1
agent3.sinks = k1

agent3.sources.r1.type = exec
agent3.sources.r1.command = tail -F /opt/datas/weblog.log
agent3.sources.r1.channels = c1

agent3.channels.c1.type = memory
agent3.channels.c1.capacity = 10000
agent3.channels.c1.transactionCapacity = 10000
agent3.channels.c1.keep-alive = 5

agent3.sinks.k1.type = avro
agent3.sinks.k1.channel = c1
agent3.sinks.k1.hostname = master
agent3.sinks.k1.port = 5555
```

7) Flume+HBase+Kafka 集成与开发

下载 Flume 源码并导入 Idea 开发工具，将 apache-flume-1.7.0-src.tar.gz 源码下载到本地解压。通过 idea 导入 flume 源码，打开 idea 开发工具，选择 File——》Open，然后找到 flume 源码解压文件，选中 flume-ng-hbase-sink，点击 ok 加载相应模块的源码。

下载日志数据并分析 ([搜狗实验室用户查询日志](#))

搜索引擎查询日志库设计为包括约 1 个月(2008 年 6 月)Sogou 搜索引擎部分网页查询需求及用户点击情况的网页查询日志数据集合。为进行中文搜索引擎用户行为分析的研究者提供基准研究语料。

数据格式为:访问时间\t 用户 ID\t[查询词]\t 该 URL 在返回结果中的排名\t 用户点击的序号\t 用户点击的 URL。

其中，用户 ID 是根据用户使用浏览器访问搜索引擎时的 Cookie 信息自动赋值，即同一次使用浏览器输入的不同查询对应同一个用户 ID。

进行 flume agent-1 聚合节点与 HBase、Kafka 集成的配置，master 节点修改配置文件 flume-conf.properties。

```
#####flume+HBase#####
agent1.sources = r1
agent1.channels = kafkaC hbaseC
agent1.sinks = kafkaSink hbaseSink

agent1.sources.r1.type = avro
agent1.sources.r1.channels = hbaseC kafkaC
agent1.sources.r1.bind = master
agent1.sources.r1.port = 5555
agent1.sources.r1.threads = 5

agent1.channels.hbaseC.type = memory
agent1.channels.hbaseC.capacity = 100000
agent1.channels.hbaseC.transactionCapacity = 100000
agent1.channels.hbaseC.keep-alive = 20

agent1.sinks.hbaseSink.type = asynchbase
agent1.sinks.hbaseSink.table = weblogs
agent1.sinks.hbaseSink.columnFamily = info
agent1.sinks.hbaseSink.serializer = org.apache.flume.sink.hbase.KfkAsyn
cHbaseEventSerializer
agent1.sinks.hbaseSink.channel = hbaseC
agent1.sinks.hbaseSink.serializer.payloadColumn = datetime,userid,search
name,retorder,cliorder,cliurl
#####flume+Kafka#####
agent1.channels.kafkaC.type = memory
agent1.channels.kafkaC.capacity = 100000
agent1.channels.kafkaC.transactionCapacity = 100000
agent1.channels.kafkaC.keep-alive = 20

agent1.sinks.kafkaSink.channel = kafkaC
agent1.sinks.kafkaSink.type = org.apache.flume.sink.kafka.KafkaSink
agent1.sinks.kafkaSink.brokerList = master:9092,slaver01:9092,slaver02:
9092
agent1.sinks.kafkaSink.topic = weblogs
agent1.sinks.kafkaSink.zookeeperConnect = master:2181,slaver01:2181,sla
ver02:2181
agent1.sinks.kafkaSink.requiredAcks = 1
agent1.sinks.kafkaSink.batchSize = 1
```



```
agent1.sinks.kafkaSink.serializer.class = kafka.serializer.StringEncoder
```

自定义 SinkHBase 程序设计与开发

模仿 SimpleAsyncHbaseEventSerializer 自定义

KfkAsyncHbaseEventSerializer 实现类，修改一下代码即可。

@Override

```
public List<PutRequest> getActions() {
    List<PutRequest> actions = new ArrayList<PutRequest>();
    if (payloadColumn != null) {
        byte[] rowKey;
        try {
            /*-----代码修改开始-----
            ---*/
            //解析列字段
            String[] columns = new String(this.payloadColumn).split(",");
            //解析flume 采集过来的每行的值
            String[] values = new String(this.payload).split(",");
            //时间
            String datetime = values[0].toString();
            //用户id
            String userid = values[1].toString();
            //根据业务自定义 Rowkey
            rowKey = SimpleRowKeyGenerator.getKfkRowKey(userid, datetime);
            System.out.println("start insert*****");
            for(int i=0;i < columns.length;i++) {
                byte[] colColumn = columns[i].getBytes();
                byte[] colValue = values[i].getBytes(Charsets.UTF_8);

                //数据校验: 字段和值是否对应
                //if (colColumn.length != colValue.length) break;

                //插入数据
                PutRequest putRequest = new PutRequest(table, rowKey, cf,
                    colColumn, colValue);
                actions.add(putRequest);

                /*-----代码修改结束-----
            ---*/
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```

        System.out.println("insert sucessfully"+rowKey);
    } catch (Exception e) {
        throw new FlumeException("Could not get row key!", e);
    }
}
return actions;
}

```

在 SimpleRowKeyGenerator 类中，根据具体业务添加自定义 Rowkey 生成方法。

```

public static byte[] getKfkRowKey(String userid, String datetime) throws
UnsupportedEncodingException {
    return (userid + "-" + datetime + "-" +
String.valueOf(System.currentTimeMillis()).getBytes("UTF8"));
}

```

自定义编译程序打 jar 包

在 idea 工具中，选择 File——》ProjectStructure，左侧选中 Artifacts，然后点击右侧的+号，最后选择 JAR——》From modules with dependencies，然后直接点击 ok。删除其他依赖包，只把 flume-ng-hbase-sink 打成 jar 包就可以了，然后依次点击 apply，ok。点击 build 进行编译，会自动打成 jar 包。到项目的：apache-flume-1.7.0-src\flume-ng-sinks\flume-ng-hbase-sink\classes\artifacts\flume_ng_hbase_sink_jar 目录下找到刚刚打的 jar 包，将打包名字替换为 flume 自带的包名 flume-ng-hbase-sink-1.7.0.jar，然后上传至 flume/lib 目录下，覆盖原有的 jar 包即可。

8) 数据预处理

由于用户查询信息的多样性，难免会存在空格、分隔符、制表符、逗号等特殊字符的存在。故在进行数据分析前，对数据进行预处理是十分必要的。

为方便操作，在这里我们选用 Python 作为主要的操作环境，适用 JupyterNotebook 作为编辑器。

首先，我们需要处理特殊字符。我们利用正则表达式，可轻松剔除掉影响分析的字符。

具体代码如下所示。

```

with open(w,"w",encoding='utf-8') as file:
    for s in tqdm(lines):
        s1 = re.sub(r'(\d*) (\d*)', r'\1\t\2', s)
        s2 = re.sub(r'\[1.\t', r'[1.', s1)
        s3 = re.sub(r'(\d) \[', r'\1\t[', s2)
        s4 = re.sub(r'([a-z\]])\t([A-Z])', r'\1\2', s3)
        # s5 = re.sub(r'([\d])\t(?:[a-zA-Z]|[0-9]|[$-
_@.&+]|[*\(\),]|(?:%[0-9a-fA-F][0-9a-fA-F]))+\n', r'\1\n',s4)
        s5 = s4.replace(',', '')
        s6 = re.sub(r'\t', r',', s5)
        file.write(s6)

```

部分特殊字符剔除之后，会出现部分 url 属性缺失的问题，在这里利用 fillna(0) 进行零值填充。

由于后续数据导入操作中，HBase 需要一个属性作为主键，故自定义 id 属性为序列+时间+用户 id 的形式。

```

num=np.arange(1,len(result)+1,1)
no=pd.Series(num)
result['id']=no.map(str)+'_'+result['datetime']+'_'+result['userid'].map(str)
result=result[['id','datetime','userid','searchname','retorder','cliorder','cliurl']]
result

```

而后，将结果输出至逗号分隔值文件形式即可，实时流数据文件可不定义 id 属性字段。（实时流：weblog.log，离线流：weblog_offline.log）

9) 数据采集存储分发完整流程测试

idea 工具开发数据生成模拟程序

在 idea 开发工具中构建 weblogs 项目，编写数据生成模拟程序。

```

package main.java;
import java.io.*;
public class ReadWrite {
    static String readFileName;
    static String writeFileName;

```

```

public static void main(String args[]){
    readFileName = args[0];
    writeFileName = args[1];
    try {
        // readInput();
        readFileByLines(readFileName);
    }catch(Exception e){
    }
}

public static void readFileByLines(String fileName) {
    FileInputStream fis = null;
    InputStreamReader isr = null;
    BufferedReader br = null;
    String tempString = null;
    try {
        System.out.println("以行为单位读取文件内容，一次读一整行：");
        fis = new FileInputStream(fileName); // FileInputStream
        // 从文件系统中的某个文件中获取字节
        isr = new InputStreamReader(fis,"UTF8");
        br = new BufferedReader(isr);
        int count=0;
        while ((tempString = br.readLine()) != null) {
            count++;
            // 显示行号
            Thread.sleep(100);
            String str = new String(tempString.getBytes("UTF8"),"UTF8");
            System.out.println("row:"+count+">>>>>>>" + tempString);
            method1(writeFileName,tempString);
            //appendMethodA(writeFileName,tempString);
        }
        isr.close();
    } catch (IOException e) {
        e.printStackTrace();
    } catch (InterruptedException e) {
        e.printStackTrace();
    } finally {
        if (isr != null) {
            try {
                isr.close();
            } catch (IOException e1) {
            }
        }
    }
}

```

```

}
public static void method1(String file, String content) {
    BufferedWriter out = null;
    try {
        out = new BufferedWriter(new OutputStreamWriter(
            new FileOutputStream(file, true)));
        out.write("\n");
        out.write(content);
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        try {
            out.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
}
}

```

参照前面 idea 工具项目打包方式，将该项目打成 weblogs.jar 包，然后上传至 master 节点的 /opt/jars 目录下（目录需要提前创建）

接着将 weblogs.jar 分发到另外两个节点

编写运行模拟程序的 shell 脚本

在 slaver01 节点的 /opt/datas 目录下，创建 weblog-shell.sh 脚本。

```
#!/bin/bash
```

```
echo "start log....."
```

#第一个参数是原日志文件，第二个参数是日志生成输出文件

```
java -jar /opt/jars/weblogs.jar /opt/datas/weblog.log
```

```
/opt/datas/weblog-flume.log
```

修改 weblog-shell.sh 可执行权限

```
chmod 777 weblog-shell.sh
```

将 slaver01 节点上的 /opt/datas/ 目录拷贝到 slaver02 节点

```
scp -r /opt/datas/ slaver02:/opt/datas/
```

修改 slaver01 和 slaver02 节点 flume-conf.properties 配置文件中日志采集文件路径。以 slaver01 节点为例。

#修改采集日志文件路径，slaver02 节点也是修改此处

```
agent2.sources.r1.command = tail -F /opt/datas/weblog-
```

```
flume.log
```

编写启动 flume 服务程序的 shell 脚本

在 slaver01 节点的 flume 安装目录下编写 flume 启动脚本 flume-kfk-start.sh。

```
#!/bin/bash
```

```
echo "flume-2 start ....."
```

```
bin/flume-ng agent --conf conf -f conf/flume-conf.properties
```

```
-n agent2 -Dflume.root.logger=INFO,console
```

在 slaver02 节点的 flume 安装目录下编写 flume 启动脚本 flume-kfk-start.sh

```
#!/bin/bash
```

```
echo "flume-3 start ....."
```

```
bin/flume-ng agent --conf conf -f conf/flume-conf.properties
```

```
-n agent3 -Dflume.root.logger=INFO,console
```

在 master 节点的 flume 安装目录下编写 flume 启动脚本。flume-kfk-start.sh

```
#!/bin/bash
```

```
echo "flume-1 start ....."
```

```
bin/flume-ng agent --conf conf -f conf/flume-conf.properties
```

```
-n agent1 -Dflume.root.logger=INFO,console
```

编写 Kafka Consumer 执行脚本

在 master 节点的 Kafka 安装目录下编写 Kafka Consumer 执行脚本 kfk-test-consumer.sh

```
#!/bin/bash
```

```
echo "kfk-kafka-consumer.sh start ....."
```

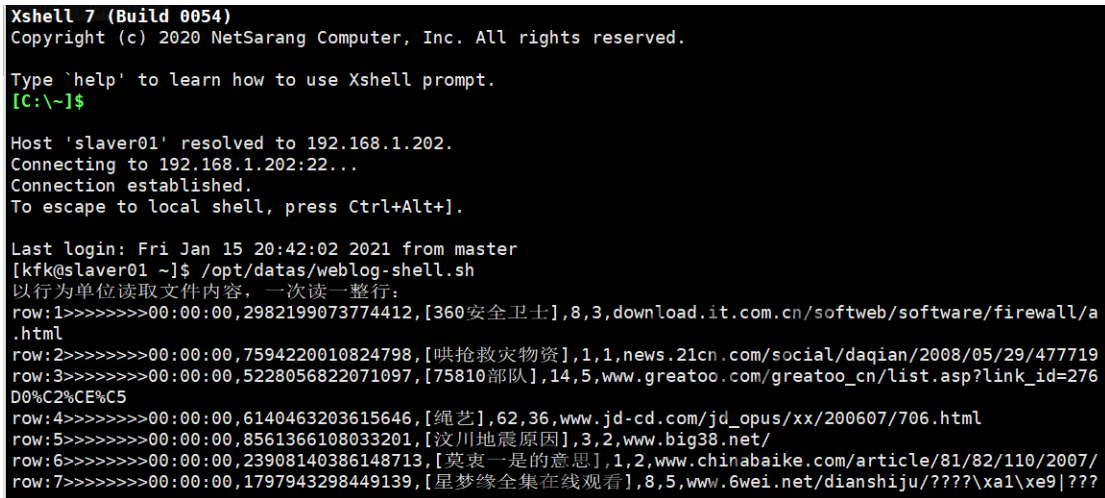
```
bin/kafka-console-consumer.sh --zookeeper
master:2181,slaver01:2181,slaver02:2181 --from-beginning --topic
weblogs
```

将 `kfk-test-consumer.sh` 脚本分发另外两个节点即可。

启动模拟程序并测试

在 `slaver01` 节点启动日志产生脚本，模拟产生日志是否正常。

```
/opt/datas/weblog-shell.sh
```



```
Xshell 7 (Build 0054)
Copyright (c) 2020 NetSarang Computer, Inc. All rights reserved.

Type 'help' to learn how to use Xshell prompt.
[!C:\~!$

Host 'slaver01' resolved to 192.168.1.202.
Connecting to 192.168.1.202:22...
Connection established.
To escape to local shell, press Ctrl+Alt+].

Last login: Fri Jan 15 20:42:02 2021 from master
[kfk@slaver01 ~]$ /opt/datas/weblog-shell.sh
以行为单位读取文件内容，一次读一整行：
row:1>>>>>>>>00:00:00,2982199073774412,[360安全卫士],8,3,download.it.com.cn/softweb/software/firewall/a
.html
row:2>>>>>>>>00:00:00,7594220010824798,[哄抢救灾物资],1,1,news.21cn.com/social/daqian/2008/05/29/477719
row:3>>>>>>>>00:00:00,5228056822071097,[75810部队],14,5,www.greatoo.com/greatoo_cn/list.asp?link_id=276
D0%C2%CE%C5
row:4>>>>>>>>00:00:00,6140463203615646,[绳艺],62,36,www.jd-cd.com/jd_opus/xx/200607/706.html
row:5>>>>>>>>00:00:00,8561366108033201,[汶川地震原因],3,2,www.big38.net/
row:6>>>>>>>>00:00:00,23908140386148713,[莫衷一是的意思],1,2,www.chinabaike.com/article/81/82/110/2007/
row:7>>>>>>>>00:00:00,1797943298449139,[星梦缘全集在线观看],8,5,www.6wei.net/dianshiju/???\xa1\xe9|???
```

图 9 模拟产生日志

启动数据采集所有服务

a) 启动 Zookeeper 服务

```
bin/zkServer.sh start
```

b) 启动 hdfs 服务

```
sbin/start-dfs.sh
```

c) 启动 HBase 服务

```
bin/start-hbase.sh
```

创建 hbase 业务表

```
bin/hbase shell
```

```
create 'weblogs','info'
```

d) 启动 Kafka 服务

```
bin/kafka-server-start.sh config/server.properties &
```

创建业务数据 topic

```
bin/kafka-topics.sh --zookeeper localhost:2181 --create --  
topic weblogs --replication-factor 1 --partitions 1
```

配置 flume 相关环境变量

```
vi flume-env.sh  
  
export JAVA_HOME=/opt/modules/jdk1.7.0_67  
export HADOOP_HOME=/opt/modules/hadoop-2.5.0  
export HBASE_HOME=/opt/modules/hbase-0.98.6
```

完成数据采集全流程测试

a) 在 master 节点上启动 flume 聚合脚本，将采集的数据分发到 Kafka 集群和 hbase 集群。

```
./flume-kfk-start.sh
```

b) 在 slaver01 节点上完成数据采集

使用 shell 脚本模拟日志产生

```
/opt/datas/weblog-shell.sh
```

启动 flume 采集日志数据发送给聚合节点

```
./flume-kfk-start.sh
```

c) 在 slaver02 节点上完成数据采集

使用 shell 脚本模拟日志产生

```
/opt/datas/weblog-shell.sh
```

启动 flume 采集日志数据发送给聚合节点

```
./flume-kfk-start.sh
```

d) 启动 Kafka Consumer 查看 flume 日志采集情况

```
bin/kafka-console-consumer.sh --zookeeper localhost:2181 --  
topic weblogs --from-beginning
```

e) 查看 hbase 数据写入情况

```
./hbase-shell  
count 'weblogs'
```

10) MySQL 安装

由于在 Linux 安装 MySQL 较为基础，故不在此过多赘述。详尽安装方法请

参考网络教程。配置 MySQL 账号及字符编码的过程中, 以下三篇教程可供参考。

[MySQL5.7 免密重置 root 密码](#)

[mysql 密码规则配置-配置为简单密码 123456](#)

[设置 mySql 的编码方式为 utf-8](#)

11) Hive 与 HBase 集成进行数据分析

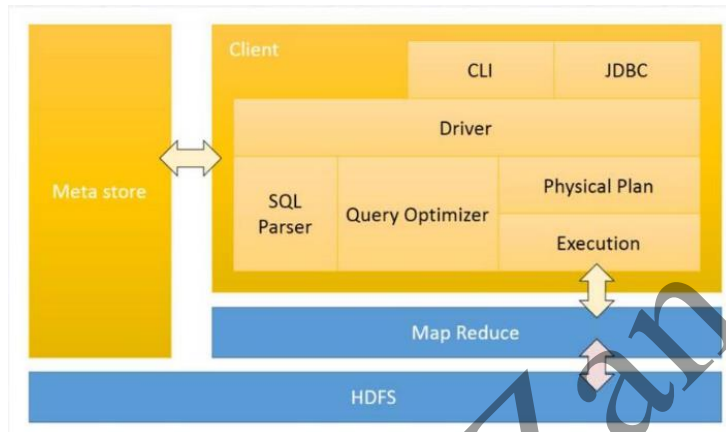


图 10 Hive 架构设计

Hive 的下载和安装部署

下载解压并安装 apache-hive-0.13.1-bin.tar.gz。

修改 hive-log4j.properties 配置文件

```
cd /opt/modules/hive-0.13.1-bin/conf
mv hive-log4j.properties.template hive-log4j.properties
vi hive-log4j.properties
#日志目录需要提前创建
hive.log.dir=/opt/modules/hive-0.13.1-bin/logs
```

修改 hive-env.sh 配置文件

```
mv hive-env.sh.template hive-env.sh
vi hive-env.sh
export HADOOP_HOME=/opt/modules/hadoop-2.5.0
export HIVE_CONF_DIR=/opt/modules/hive-0.13.1-
bin/conf
```

首先启动 HDFS, 然后创建 Hive 的目录

```
bin/hdfs dfs -mkdir -p /user/hive/warehouse
```

```
bin/hdfs dfs -chmod g+w /user/hive/warehouse
```

启动 hive

```
./hive
```

若没有明显报错，则安装成功。

Hive 与 MySQL 集成

在/opt/modules/hive-0.13.1-bin/conf 目录下创建 hive-site.xml 文件，配置 mysql 元数据库。

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
  <property>
    <name>javax.jdo.option.ConnectionURL</name>
    <value>jdbc:mysql://master/metastore?createDatabaseIfNotExist=true&useSSL=false</value>
  </property>
  <property>
    <name>javax.jdo.option.ConnectionDriverName</name>
    <value>com.mysql.jdbc.Driver</value>
  </property>
  <property>
    <name>javax.jdo.option.ConnectionUserName</name>
    <value>root</value>
  </property>
  <property>
    <name>javax.jdo.option.ConnectionPassword</name>
    <value>123456</value>
  </property>
  <property>
    <name>hbase.zookeeper.quorum</name>
    <value>master,slaver01,slaver02</value>
  </property>
</configuration>
```

设置用户连接

查看用户信息

```
mysql -uroot -p123456
```

```
show databases;
```

```
use mysql;
show tables;
select User,Host,Password from user;
```

更新用户信息

```
update user set Host='%' where User = 'root' and
Host='localhost'
```

删除用户信息

```
delete from user where user='root' and host='127.0.0.1'
select User,Host, authentication_string from user;
delete from user where host='localhost'
```

刷新信息

```
flush privileges;
```

拷贝 mysql 驱动包到 hive 的 lib 目录下

```
cp mysql-connector-java-5.1.46.jar /opt/modules/hive-
0.13.1/lib/
```

保证第三台集群到其他节点无秘钥登录

Hive 与 HBase 集成

在 hive-site.xml 文件中配置 Zookeeper，hive 通过这个参数去连接 HBase 集群。

```
<property>
  <name>hbase.zookeeper.quorum</name>
  <value>master,slaver01,slaver02</value>
</property>
```

将 hbase 的 9 个包（如下所示）拷贝到 hive/lib 目录下。如果是 CDH 版本，已经集成好不需要导包。

```
hbase-server-0.98.6-cdh5.3.0.jar
hbase-client-0.98.6-cdh5.3.0.jar
hbase-protocol-0.98.6-cdh5.3.0.jar
hbase-it-0.98.6-cdh5.3.0.jar
htrace-core-2.04.jar
hbase-hadoop2-compact-0.98.6-cdh5.3.0.jar
```

hbase-hadoop-compact-0.98.6-cdh5.3.0.jar

high-scale-lib-1.1.1.jar

hbase-common-0.98.6-cdh5.3.0.jar

创建与 HBase 集成的 Hive 的外部表

```
create external table weblogs(id string,datetime string,userid
string,searchname string,retorder string,cliorder string,cliurl
string) STORED BY
'org.apache.hadoop.hive.hbase.HBaseStorageHandler' WITH
SERDEPROPERTIES("hbase.columns.mapping" =
":key,info:datetime,info:userid,info:searchname,info:retorder,info:cliorder,info:cliurl") TBLPROPERTIES("hbase.table.name" = "weblogs");
```

#查看 hbase 数据记录

```
select count(*) from weblogs;
```

12) Cloudera Hue 大数据可视化分析

Hue 是一个开源的 Apache Hadoop UI 系统，最早是由 Cloudera Desktop 演化而来，由 Cloudera 贡献给开源社区，它是基于 Python Web 框架 Django 实现的。通过使用 Hue 我们可以在浏览器端的 Web 控制台上与 Hadoop 集群进行交互来分析处理数据，例如操作 HDFS 上的数据，运行 MapReduce Job 等等。

下载解压并安装 hue-3.9.0-cdh5.5.0.tar.gz。

使用 yum 命令安装依赖包（依赖包参考此[连接](#)）

```
yum install ant asciidoc cyrus-sasl-devel cyrus-sasl-gssapi gcc gcc-
c++ krb5-devel libtidy libxml2-devel libxslt-devel openssl-devel python-devel sqlite-devel openssl-devel mysql-devel gmp-devel
```

若下载速度较慢，可参考如下换源教程。

<https://developer.aliyun.com/article/779734?spm=a2c6h.14164896.0.0.35395c886pkmTp>

编译

```
cd hue-3.9.0-cdh5.5.0
```

```
make apps
```

Hue 基本配置与服务启动

修改配置文件

```
vi /desktop/conf/hue.ini
```

```
#秘钥
```

```
secret_key=jFE93j;2[290-eiw.KEiwN2s3['d;/.q[eIW^y#e+=Iei*@Mn
```

```
< qW5o
```

```
#host port
```

```
http_host=slaver02
```

```
http_port=8888
```

```
#时区
```

```
time_zone=Asia/Shanghai
```

```
default_hdfs_superuser=kfk
```

修改 desktop.db 文件权限

```
chmod o+w desktop/desktop.db
```

启动 Hue 服务

```
/opt/modules/hue-3.9.0-cdh5.5.0/build/env/bin/supervisor &
```

查看 Hue web 界面

```
slaver02:8888
```

Hue 与 HDFS 集成

修改 core-site.xml 配置文件，添加如下内容

```
<property>
```

```
  <name>hadoop.proxyuser.hue.hosts</name>
```

```
  <value>*</value>
```

```
</property>
```

```
<property>
```

```
  <name>hadoop.proxyuser.hue.groups</name>
```

```
  <value>*</value>
```

```
</property>
```

修改 hue.ini 配置文件

```
fs_defaultfs=hdfs://ns
webhdfs_url=http://master:50070/webhdfs/v1
hadoop_hdfs_home=/opt/modules/hadoop-2.5.0
hadoop_bin=/opt/modules/hadoop-2.5.0/bin
hadoop_conf_dir=/opt/modules/hadoop-2.5.0/etc/hadoop
```

将 core-site.xml 配置文件分发到其他节点

Hue 与 YARN 集成

修改 hue.ini 配置文件

```
[[yarn_clusters]]
```

```
[[[default]]]
```

```
# Enter the host on which you are running the ResourceManager
```

```
#resourcemanager_host=rs
```

```
# The port where the ResourceManager IPC listens on
```

```
#resourcemanager_port=8032
```

```
# Whether to submit jobs to this cluster
```

```
submit_to=True
```

```
# Resource Manager logical name (required for HA)
```

```
logical_name=rm1
```

```
# Change this if your YARN cluster is Kerberos-secured
```

```
#security_enabled=false
```

```
# URL of the ResourceManager API
```

```
resourcemanager_api_url=http://master:8088
```

```
# URL of the ProxyServer API
```

```
proxy_api_url=http://master:8088
```

```
# URL of the HistoryServer API
```

```
history_server_api_url=http://master:19888
```

```
# In secure mode (HTTPS), if SSL certificates from YARN Rest APIs
```

```
# have to be verified against certificate authority
```

```
#ssl_cert_ca_verify=False
```

```
# HA support by specifying multiple clusters
```

e.g.

```
[[[ha]]]
# Resource Manager logical name (required for HA)
logical_name=rm2
resourcemanager_api_url=http://slaver01:8088
history_server_api_url=http://slaver01:19888
submit_to=True
```

注：若要查看 Job 历史，需提前执行命令：`mr-jobhistory-daemon.sh start historyserver`

Hue 与 Hive 集成

修改 hue.ini 配置文件

```
hive_server_host=slaver02
hive_server_port=10000
hive_conf_dir=/opt/modules/hive-0.13.1-bin/conf
```

注：Hue 在启动前，需执行命令：`bin/hiveserver2 &`

Hue 与 mysql 集成

修改 hue.ini 配置文件

```
[[mysql]]
nice_name="My SQL DB"
name=metastore
engine=mysql
host=master
port=3306
user=root
password=123456
options={"init_command":"SET NAMES 'utf8'"}
```

Hue 与 HBase 集成

修改 hue.ini 配置文件

```
hbase_clusters=(Cluster|master:9090)
hbase_conf_dir=/opt/modules/hbase-0.98.6/conf
```

注：Hue 在启动前，需执行命令：`bin/hbase-daemon.sh start thrift`

13) Spark2.X 环境准备、编译部署及运行

Spark 是一个用来实现快速而通用的集群计算的平台。在速度方面，Spark 扩展了广泛使用的 MapReduce 计算模型，而且高效地支持更多计算模式，包括交互式查询和流处理。在处理大规模数据集时，速度是非常重要的。速度快就意味着我们可以进行交互式的数据操作，否则我们每次操作就需要等待数分钟甚至数小时。

Spark 的一个主要特点就是能够在内存中进行计算，因而更快。不过即使是必须在磁盘上进行的复杂计算，Spark 依然比 MapReduce 更加高效。

scala 安装及环境变量设置

下载解压并安装 `scala-2.11.8.tgz`。

配置环境变量

```
vi /etc/profile
```

```
export SCALA_HOME=/opt/modules/scala-2.11.8
```

```
export PATH=$PATH:$SCALA_HOME/bin
```

编辑退出之后，使之生效

```
source /etc/profile
```

Spark2.1.3 源码下载及编译生成版本

将 Spark2.1.3 源码[下载](#)到 `slaver01` 节点的 `/opt/software/` 目录下，并解压到预设目录。

spark2.1.3 编译所需要的环境：Maven3.3.9 和 Java7

Spark 源码编译的方式：Maven 编译、SBT 编译（暂无）和打包编译 `make-distribution.sh`

下载并解压并安装 `apache-maven-3.3.9-bin.tar.gz`

配置 `MAVEN_HOME`

```
vi /etc/profile
```

```
export MAVEN_HOME=/opt/modules/apache-maven-3.3.9
```

```
export PATH=$PATH:$MAVEN_HOME/bin
```



```
export MAVEN_OPTS="-Xmx2g -XX:MaxPermSize=1024M -
XX:ReservedCodeCacheSize=1024M"
```

编辑退出之后，使之生效

```
source /etc/profile
```

查看 maven 版本

```
mvn -version
```

编辑./dev/make-distribution.sh 内容，可以让编译速度更快

注释掉以下内容

```
VERSION=$("$MVN" help:evaluate -
Dexpression=project.version $@ 2>/dev/null | grep -v "INFO" | tail -
n 1)
SCALA_VERSION=$("$MVN" help:evaluate -
Dexpression=scala.binary.version $@ 2>/dev/null\
| grep -v "INFO"\
| tail -n 1)
SPARK_HADOOP_VERSION=$("$MVN" help:evaluate -
Dexpression=hadoop.version $@ 2>/dev/null\
| grep -v "INFO"\
| tail -n 1)
SPARK_HIVE=$("$MVN" help:evaluate -Dexpression=project.activeProfiles -
pl sql/hive $@ 2>/dev/null\
| grep -v "INFO"\
| fgrep --count "<id>hive</id>";\
# Reset exit status to 0, otherwise the script stops here if the la
st grep finds nothing\
# because we use "set -o pipefail"
echo -n)
```

添加如下内容

```
VERSION=2.1.3
SCALA_VERSION=2.11.8
SPARK_HADOOP_VERSION=2.5.0
#支持 spark on hive
SPARK_HIVE=1
```

通过 make-distribution.sh 源码编译 spark

```
./dev/make-distribution.sh --name custom-spark --tgz -Phadoop-2.5  
-Phive -Phive-thriftserver -Pyarn
```

#编译完成之后解压安装并分发到其他节点即可

```
tar -zxvf spark-2.1.3-bin-custom-spark.tgz -C /opt/modules/
```

spark2.1.3 本地模式运行测试

启动 spark-shell 测试

```
./bin/spark-shell
```

```
scala> val textFile = spark.read.textFile("README.md")
```

```
textFile: org.apache.spark.sql.Dataset[String] = [value: string]
```

```
scala> textFile.first()
```

```
res1: String = # Apache Spark
```

配置 Spark2.1.3 集群运行模式

前提: jdk1.7、scala2.11.8、Hadoop2.5.0

a) 配置 slave

```
vi slaves
```

```
master
```

```
slaver01
```

```
slaver02
```

b) 配置 spark-env.sh

```
export JAVA_HOME=/opt/modules/jdk1.7.0_67  
export SCALA_HOME=/opt/modules/scala-2.11.8  
export SPARK_DIST_CLASSPATH=$(/opt/modules/hadoop-  
2.5.0/bin/hadoop classpath)  
export HADOOP_CONF_DIR=/opt/modules/hadoop-2.5.0/etc/hadoop  
SPARK_CONF_DIR=/opt/modules/spark-2.1.3-bin/conf  
SPARK_MASTER_HOST=slaver01  
SPARK_MASTER_PORT=7077  
SPARK_MASTER_WEBUI_PORT=8080  
SPARK_WORKER_CORES=1
```

```
SPARK_WORKER_MEMORY=1g
SPARK_WORKER_PORT=7078
SPARK_WORKER_WEBUI_PORT=8081
```

c) 将 spark 配置分发到其他节点并修改每个节点特殊配置

spark on yarn 模式配置并测试

注意 hadoop 配置文件中 jdk 版本是否与当前 jdk 版本一致

spark on yarn 模式提交作业

```
节点 slaver01 运行 spark ./spark-shell --master yarn --deploy-mode
client
```

```
Scala>sc.parallelize(1 to 100,5).count
```

14) Spark SQL 快速离线数据分析



图 11 Spark SQL 处理数据架构

Spark SQL 与 Hive 集成 (spark-shell)

- 将 hive 的配置文件 hive-site.xml 拷贝到 spark conf 目录
- 拷贝 hive 中的 mysql-connector jar 包到 spark 的 jar 目录下
- 检查 spark-env.sh 文件中的配置项

```
HADOOP_CONF_DIR=/opt/modules/hadoop-2.5.0/etc/hadoop
```

- 检查 mysql 是否启动, 若未启动, 请自行启动

```
service mysqld status
```

- 启动 hive metastore 服务

```
bin/hive --service metastore &
```

- 启动 hive

```
bin/hive
>show databases;
>create database kfk;
```

g) 启动 spark-shell

```
bin/spark-shell
>spark.sql("show databases").show
default
kfk
```

Spark SQL 与 Hive 集成 (spark-sql)

启动 spark-sql

```
bin/spark-sql
```

#查看数据库

```
>show databases;
default
kfk
```

Spark SQL 与 HBase 集成

Spark SQL 与 HBase 集成，其核心就是 Spark Sql 通过 hive 外部表来获取 HBase 的表数据。

拷贝 HBase 的包和 hive 包到 spark 的 jars 目录下

```
hbase-client-0.98.6-cdh5.3.0.jar
hbase-common-0.98.6-cdh5.3.0.jar
hbase-protocol-0.98.6-cdh5.3.0.jar
hbase-server-0.98.6-cdh5.3.0.jar
hive-hbase-handler-1.2.1.jar
htrace-core-2.04.jar
mysql-connector-java-5.1.27-bin.jar
```

启动 spark-shell

```
bin/spark-shell
```

```
>val df =spark.sql("show databases").show
```

15) Structured Streaming 业务数据实时分析

在 Idea 中创建 Maven 项目，具体方法可参考网络教程。

引入依赖 pom.xml

```

<?xml version="1.0" encoding="UTF-8"?>

<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>
  <packaging>war</packaging>

  <name>titlenameCount</name>
  <groupId>com.kfk.spark</groupId>
  <artifactId>titlenameCount</artifactId>
  <version>1.0-SNAPSHOT</version>

  <properties>
    <scala.version>2.11.8</scala.version>
    <scala.binary.version>2.11</scala.binary.version>
    <spark.version>2.1.3</spark.version>
  </properties>

  <dependencies>
    <dependency>
      <groupId>org.apache.spark</groupId>
      <artifactId>spark-core_${scala.binary.version}</artifactId>
      <version>${spark.version}</version>
    </dependency>
    <dependency>
      <groupId>org.apache.spark</groupId>
      <artifactId>spark-streaming_${scala.binary.version}</artifactId>
      <version>${spark.version}</version>
    </dependency>
    <dependency>
      <groupId>org.apache.spark</groupId>
      <artifactId>spark-sql_${scala.binary.version}</artifactId>
      <version>${spark.version}</version>
    </dependency>
    <dependency>
      <groupId>org.apache.spark</groupId>
      <artifactId>spark-hive_${scala.binary.version}</artifactId>
      <version>${spark.version}</version>
    </dependency>
    <dependency>

```

```

        <groupId>org.apache.spark</groupId>
        <artifactId>spark-streaming-kafka-0-
10_${scala.binary.version}</artifactId>
        <version>${spark.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apache.spark</groupId>
        <artifactId>spark-sql-kafka-0-
10_${scala.binary.version}</artifactId>
        <version>${spark.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apache.hadoop</groupId>
        <artifactId>hadoop-client</artifactId>
        <version>2.5.0</version>
    </dependency>
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>5.1.46</version>
    </dependency>
    <dependency>
        <groupId>javax</groupId>
        <artifactId>javaee-api</artifactId>
        <version>7.0</version>
    </dependency>
</dependencies>
</project>

```

编写 StructuredStreamingKafka 类(StructuredStreamingKafka.scala)

```
package com.spark.test
```

```
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.streaming.ProcessingTime
```

```
/**
 * 结构化流从 kafka 中读取数据存储在关系型数据库 mysql
 * 目前结构化流对 kafka 的要求版本 0.10 及以上
 */
```

```
object StructuredStreamingKafka {
```

```
System.setProperty("hadoop.home.dir", "D:\\hadoop-common-2.2.0-bin")
```

```
//对象匹配
```

```
case class Weblog(datatime: String, //访问时间
                  userid: String, //用户ID
                  searchname: String, //[查询词]
                  retorder: String, //该URL 在返回结果中的排名
                  cliorder: String, //用户点击的顺序号
                  cliurl: String) //用户点击的URL
```

```
def main(args: Array[String]): Unit = {
```

```
    val spark = SparkSession.builder()
        .master("local[2]")
        .appName("streaming").getOrCreate()
```

```
    //df 流数据输入 dataframe
```

```
    val df = spark
        .readStream
        .format("kafka") //流数据来源
        .option("kafka.bootstrap.servers", "master:9092") //流数据源主机
        .option("subscribe", "weblogs") //订阅的topic
        .load()
```

```
    import spark.implicits._
```

```
    val lines = df.selectExpr("CAST(value AS STRING)").as[String]
```

```
    val weblog = lines.map(_.split(","))
        .map(x => WebLog(x(0), x(1), x(2), x(3), x(4), x(5)))
```

```
    val titleCount = weblog
        .groupBy("searchname").count().toDF("titleName", "count")
```

```
    val url = "jdbc:mysql://master:3306/test?useSSL=false"
```

```
    val username = "root"
```

```
    val password = "123456"
```

```
    val writer = new JDBCSink(url, username, password)
```

```
    val query = titleCount.writeStream
        .foreach(writer)
        .outputMode("update")
        .trigger(ProcessingTime("6 seconds"))
        .start()
```

```
    query.awaitTermination()
```

```
}
```

```
}
```

编写 JDBC Sink 类(JDBCSink.scala)

```
package com.spark.test

import java.sql._

import org.apache.spark.sql.{ForeachWriter, Row}

/**
 * 处理从 StructuredStreaming 中向mysql 中写入数据
 */
class JDBCSink(url: String, username: String, password: String) extends
ForeachWriter[Row] {
  //sql 请求
  var statement: Statement = _
  //sql 请求结果
  var resultSet: ResultSet = _
  //连接器
  var connection: Connection = _

  override def open(partitionId: Long, version: Long): Boolean = {
    connection = new MySQLPool(url, username, password).getJdbcConn()
    statement = connection.createStatement()
    return true
  }

  override def process(value: Row): Unit = {

    val titleName = value.getAs[String]("titleName").replaceAll("[\\[\\]]",
    "")
    val count = value.getAs[Long]("count")

    val querySql = "select 1 from webCount " +
      "where titleName = '" + titleName + "'"

    val updateSql = "update webCount set " +
      "count = " + count + " where titleName = '" + titleName + "'"

    val insertSql = "insert into webCount(titleName,count)" +
      "values('" + titleName + "'," + count + ")"

    try {
```



```

// 查看连接是否成功
var resultSet = statement.executeQuery(querySql)
if (resultSet.next()) {
    statement.executeUpdate(updateSql)
} else {
    statement.execute(insertSql)
}
} catch {
    case ex: SQLException => {
        println("SQLException")
    }
    case ex: Exception => {
        println("Exception")
    }
    case ex: RuntimeException => {
        println("RuntimeException")
    }
    case ex: Throwable => {
        println("Throwable")
    }
}
}
}

override def close(errorOrNull: Throwable): Unit = {
    // if(resultSet.isNull()){
    //     resultSet.close()
    // }
    if (statement == null) {
        statement.close()
    }
    if (connection == null) {
        connection.close()
    }
}
}
}

```

编写 MySQLPool 类 (MySQLPool.scala)

```
package com.spark.test
```

```
import java.sql.{Connection, DriverManager}
import java.util
```

```

/**
 * 从mysql 连接池中获取连接
 */
class MySQLPool(url: String, user: String, pwd: String) extends
Serializable {
    //连接池连接总数
    private val max = 100
    //每次产生连接数
    private val connectionNum = 1
    //当前连接池已产生的连接数
    private var conNum = 0
    //连接池
    private val pool = new util.LinkedList[Connection]()

    //获取连接
    def getJdbcConn(): Connection = {
        //同步代码块, AnyRef 为所有引用类型的基类, AnyVal 为所有值类型的基类
        AnyRef.synchronized({
            if (pool.isEmpty) {
                //加载驱动
                preGetConn()
                for (i <- 1 to connectionNum) {
                    val conn = DriverManager.getConnection(url, user, pwd)
                    pool.push(conn)
                    conNum += 1
                }
            }
            pool.poll()
        })
    }

    //释放连接
    def releaseConn(conn: Connection): Unit = {
        pool.push(conn)
    }

    //加载驱动
    private def preGetConn(): Unit = {
        //控制加载
        if (conNum > max && pool.isEmpty) {
            //当前连接数大于等于池子最大容纳数线程等待并重新加载
            print("busyness")
            Thread.sleep(2000)
            preGetConn()
        }
    }
}

```

```

    } else {
        Class.forName("com.mysql.jdbc.Driver")
        //当前连接数小于池子最大总数可以加载
    }
}
}
}

```

16) 离线数据分析

离线数据导入

HBase 创建 sogou 表，将数据预处理得到的离线数据上传至 HDFS，导入逗号分隔值文件（默认编码 uft-8）

```

Hbase shell>create 'sogou', 'info'

hdfs dfs -put /opt/datas/weblog_offline.log /user/

hbase org.apache.hadoop.hbase.mapreduce.ImportTsv "-
Dimporttsv.separator=', ' -
Dimporttsv.columns='HBASE_ROW_KEY,info:datetime,info:userid,info:searchname,info:retorder,info:cliorder,info:cliurl' sogou
/user/weblog_offline.log

```

创建 hive 外部表

```

>create external table sogou(id string,datetime string,userid
string,searchname string,retorder string,cliorder string,cliurl
string) STORED BY
'org.apache.hadoop.hive.hbase.HBaseStorageHandler' WITH
SERDEPROPERTIES("hbase.columns.mapping" =
":key,info:datetime,info:userid,info:searchname,info:retorder,info:cliorder,info:cliurl") TBLPROPERTIES("hbase.table.name" = "sogou");

```

下载解压并安装 sqoop-1.4.4-cdh5.0.0.tar.gz

导入 MySQL 驱动包、hadoop-common、hadoop-hdfs、hadoop-mapreduce-client-core 相关 jar 包至 SQOOP 的 lib 目录下。

转入 conf 目录下，更改文件名并修改配置

```
cp sqoop-env-template.sh sqoop-env.sh
```

修改 conf 目录下 sqoop-env.sh 中配置

```
#Set path to where bin/hadoop is available
export HADOOP_COMMON_HOME=/opt/modules/hadoop-2.5.0

#Set path to where hadoop-*-core.jar is available
export HADOOP_MAPRED_HOME=/opt/modules/hadoop-2.5.0

#set the path to where bin/hbase is available
export HBASE_HOME=/opt/modules/hbase-0.98.6

#Set the path to where bin/hive is available
export HIVE_HOME=/opt/modules/hive-0.13.1-bin

#Set the path for where zookeeper config dir is
export ZOO_CFG_DIR=/opt/modules/zookeeper-3.4.5/conf

export HCAT_HOME=/opt/modules/hive-0.13.1-bin/hcatalog
export PATH=$PATH:$HCAT_HOME/bin
```

配置 sqoop、hcatalog 环境变量

```
export SQOOP_HOME=/opt/modules/sqoop-1.4.4-cdh5.0.0
export HCAT_HOME=/opt/modules/hive-0.13.1-bin/hcatalog
export PATH=$PATH:$SQOOP_HOME/bin
export PATH=$PATH:$HCAT_HOME/bin
```

激活配置

```
source /etc/profile
```

验证安装

```
sqoop-version
```

```
sqoop list-databases --username root --password 123456 --connect
jdbc:mysql://master:3306/
```

每分钟用户量统计(查询并将结果保存至 hive 表)

```
hive> create table userCount as select substring(datatime,1,5) as
datatime,count(distinct userid) as userCount from sogou group by
substring(datatime,1,5);
```

mysql 创建 userCount 表

```
mysql> create table userCount (datetime varchar(255),userCount  
int(11));
```

sqoop 从 hive 导入 MySQL 数据

```
sqoop export --connect jdbc:mysql://master:3306/test --username  
root --password 123456 --table userCount --export-dir  
/user/hive/warehouse/usercount/000000_0 --input-fields-terminated-by  
"\001"
```

每分钟话题数统计(查询并将结果保存至 hive 表)

```
hive> create table topicCount as select substring(datetime,1,5)  
as datetime,count(distinct searchname) as topicCount from sogou group  
by substring(datetime,1,5);
```

mysql 创建 topicCount 表

```
mysql> create table topicCount (datetime varchar(255),topicCount  
int(11));
```

sqoop 从 hive 导入 MySQL 数据

```
sqoop export --connect jdbc:mysql://master:3306/test --username  
root --password 123456 --table topicCount --export-dir  
/user/hive/warehouse/topiccount/000000_0 --input-fields-terminated-by  
"\001"
```

查询关键字数据提取到本地磁盘

```
bin/hive -e "set hive.cli.print.header=false; select searchname  
from sogou" > /opt/datas/searchname.txt
```

网址 url 数据提取到本地磁盘

```
bin/hive -e "set hive.cli.print.header=false; select cliurl from  
sogou" > /opt/datas/cliurl.txt
```

17) 大数据 Web 可视化分析系统开发

下载 Tomcat、fastjson 并创建 Web 工程并配置相关服务, 这里不做过多赘述, 可自行查找相关资料解决。

Web 系统数据处理服务层开发

WeblogService.java

```
package com.spark.service;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.HashMap;
import java.util.Map;

public class WeblogService {
    static String url ="jdbc:mysql://master:3306/test?useSSL=false";
    static String username="root";
    static String password="123456";

    public Map<String,Object> queryWeblogs() {
        Connection conn = null;
        PreparedStatement pst = null;
        String[] titleNames = new String[20];
        String[] titleCounts = new String[20];
        Map<String,Object> retMap = new HashMap<String, Object>();
        try{
            Class.forName("com.mysql.jdbc.Driver");
            conn = DriverManager.getConnection(url,username,password);
            String query_sql = "select titleName,count from webCount where
1=1 order by count desc limit 20";
            pst = conn.prepareStatement(query_sql);
            ResultSet rs = pst.executeQuery();
            int i = 0;
            while (rs.next()){
                String titleName = rs.getString("titleName");
                String titleCount = rs.getString("count");
                titleNames[i] = titleName;
                titleCounts[i] = titleCount;
                ++i;
            }
            retMap.put("titleName", titleNames);
            retMap.put("titleCount", titleCounts);
        }catch(Exception e){
```

```

        e.printStackTrace();
    }
    return retMap;
}

public String[] titleCount() {
    Connection conn = null;
    PreparedStatement pst = null;
    String[] titleSums = new String[1];
    try{
        Class.forName("com.mysql.jdbc.Driver");
        conn = DriverManager.getConnection(url,username,password);
        String query_sql = "select count(1) titleSum from webCount";
        pst = conn.prepareStatement(query_sql);
        ResultSet rs = pst.executeQuery();
        if(rs.next()){
            String titleSum = rs.getString("titleSum");
            titleSums[0] = titleSum;
        }
    }catch(Exception e){
        e.printStackTrace();
    }
    return titleSums;
}
}

```

基于 WebSocket 协议的数据推送服务开发

WeblogSocket. java

```

package com.spark.service;

import com.alibaba.fastjson.JSON;

import javax.websocket.OnClose;
import javax.websocket.OnMessage;
import javax.websocket.OnOpen;
import javax.websocket.Session;
import javax.websocket.server.ServerEndpoint;
import java.io.IOException;

```

```

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

@ServerEndpoint("/websocket")
public class WeblogSocket {

    WeblogService weblogService = new WeblogService();

    @OnMessage
    public void onMessage(String message, Session session)
        throws IOException, InterruptedException {
        while (true) {
            Map<String, Object> map = new HashMap<String, Object> ();
            map.put("titleName",
weblogService.queryWeblogs().get("titleName"));

map.put("titleCount",weblogService.queryWeblogs().get("titleCount"));
            map.put("titleSum", weblogService.titleCount());

            session.getBasicRemote().
                sendText(JSON.toJSONString(map));
            // 每5秒去数据库获得数据
            Thread.sleep(5000);
            map.clear();
        }
    }

    @OnOpen
    public void onOpen() {
        System.out.println("Client connected");
    }

    @OnClose
    public void onClose() {
        System.out.println("Connection closed");
    }
}

```

用户查询关键词词云处理

注：下述分词处理需自行下载编译 jcseg 相关组件。

AnalyzerTool.java（java 处理主程序）

```

package demo

import org.apache.spark._
import utils.AnalyzerTools

/**
 * Created by Administrator on 2016/4/6.
 */
object AnalyzerDemo {

    // 分词排序后取出词频最高的前 50 个
    def main(args: Array[String]): Unit = {
        val conf = new SparkConf().setAppName("my app").setMaster("local")
        val sc = new SparkContext(conf)
        val data = sc.textFile("D:\\searchname.txt").map(x => {
            val list = AnalyzerTools.analyzerWords(x) // 分词处理
            list.toString.replace("[", "").replace("]", "").split(",")
        }).flatMap(x => x.toList).map(x => (x.trim(), 1)).reduceByKey(_ + _)
        .top(50)(Ordering.by(x => x._2)).foreach(println)
    }

    // 分词排序
    object Ord extends Ordering[(String, Int)] {
        def compare(a: (String, Int), b: (String, Int)) = a._2 compare (b._2)
    }
}

```

AnalyzerDemo.scala (Scala 词频统计)

```

package demo

import org.apache.spark._
import utils.AnalyzerTools

/**
 * Created by Administrator on 2016/4/6.
 */

```

```

object AnalyzerDemo {

    //分词排序后取出词频最高的前50个
    def main(args: Array[String]): Unit = {
        val conf = new SparkConf().setAppName("my app").setMaster("local")
        val sc = new SparkContext(conf)
        val data = sc.textFile("D:\\searchname.txt").map(x => {
            val list = AnaylyzerTools.anaylyzerWords(x) //分词处理
            list.toString.replace("[", "").replace("]", "").split(",")
        }).flatMap(x => x.toList).map(x => (x.trim(), 1)).reduceByKey(_ +
        _).top(50)(Ordering.by(x => x._2)).foreach(println)
    }

    //分词排序
    object Ord extends Ordering[(String, Int)] {
        def compare(a: (String, Int), b: (String, Int)) = a._2 compare (b._2)
    }
}

```

具体的结果将输出至控制台，后期需自行处理至 json 格式文件。

网站热度词云展示

NetAnalyzer.java (java 网址主机名提取)

```

package utils;

import java.io.*;
import java.net.MalformedURLException;
import java.util.ArrayList;
import java.util.List;

/**
 * Created by Administrator on 2016/4/6.
 */
public class netAnalyzer {

    public static ArrayList<String> anaylyzerWords() {
        ArrayList<String> list = new ArrayList<String>();
    }
}

```

```

try {
    String strFile = "D:\\cliurl.txt";
    File file = new File(strFile);
    BufferedReader bufferedReader = new BufferedReader(new
FileReader(file));
    String strLine = null;
    int lineCount = 1;
    while (null != (strLine = bufferedReader.readLine())) {

        java.net.URL url = new java.net.URL("http://" + strLine);

        String host = url.getHost();
        list.add(host);
//        System.out.println("第[" + lineCount + "]行数据:[" + host +
//        "]);

        lineCount++;
    }
} catch (Exception e) {
    e.printStackTrace();
}
return list;
}

public static void main(String[] args) throws MalformedURLException {
    System.out.println(netAnalyzer.analyzerWords());
}
}

```

AnalyzerDemo.scala (scala 词频统计)

```

import utils.netAnalyzer
import org.apache.spark._
import org.json4s.DefaultFormats
import org.json4s.jackson.Json

import scala.collection.JavaConverters._

/**
 * Created by Administrator on 2016/4/6.
 */

```



```

        border: 1px solid #F00
    }

    /* css 注释: 为了观察效果设置宽度 边框 高度等样式 */
</style>

</head>
<body>
<h1>新闻网话题用户浏览实时统计分析 (2021/01/10) </h1>

<div>
    <input type="submit" value="实时分析" onclick="start()"/>
</div>

<div>
    <div id="main" style="width:960px;height: 540px;float:left;">第一个
</div>
    <div id="sum" style="width:540px;height: 540px;float:right;">第二个
</div>
</div>

<div id="messages"></div>

<script src="js/echarts.min.js"></script>
<script src="js/jquery-3.5.1.js"></script>

<script type="text/javascript">

    var websocket = new
WebSocket('ws://localhost:8080/sparkStu_web_war_exploded/websocket');
    var myChart = echarts.init(document.getElementById('main'));
    var myChart_sum = echarts.init(document.getElementById('sum'));

    websocket.onerror = function (event) {
        onError(event)
    };
    websocket.onopen = function (event) {
        onOpen(event)
    };
    websocket.onmessage = function (event) {
        onMessage(event)
    };

```

```

function onMessage(event) {
    var sd = JSON.parse(event.data);
    processData(sd);
    titleSum(sd.titleSum);
}

function onOpen(event) {
}

function onError(event) {
    alert(event.data);
}

function start() {
    websocket.send('hello');
    return false;
}

function processData(json) {

    var option = {
        backgroundColor: '#ffffff', // 背景色
        title: {
            text: '新闻话题浏览量【实时】排行',
            subtext: '数据来自搜狗实验室',
            textStyle: {
                fontWeight: 'normal', // 标题颜色
                color: '#408829'
            },
        },
        tooltip: {
            trigger: 'axis',
            axisPointer: {
                type: 'shadow'
            }
        },
        legend: {
            data: ['浏览量']
        },
        grid: {
            left: '3%',
            right: '4%',
            bottom: '3%',

```

```

        containLabel: true
    },
    xAxis: {
        type: 'value',
        boundaryGap: [0, 0.01]
    },
    yAxis: {
        type: 'category',
        data: json.titleName
    },
    series: [
        {
            name: '浏览量',
            type: 'bar',
            label: {
                normal: {
                    show: true,
                    position: 'insideRight'
                }
            },
            itemStyle: {normal: {color: '#f47209'}},
            data: json.titleCount
        }
    ]
};
myChart.setOption(option);
}

```

```

function titleSum(data) {

    var option = {
        backgroundColor: '#FFFFFF', // 背景色
        title: {
            text: '新闻话题曝光量【实时】统计',
            subtext: '数据来自搜狗实验室'
        },

        tooltip: {
            formatter: "{a} <br/>{b} : {c}%"
        },
    }
}

```

```

        toolbox: {
            feature: {
                restore: {},
                saveAsImage: {}
            }
        },
        series: [
            {
                name: '业务指标',
                type: 'gauge',
                max: 20000,
                detail: {formatter: '{value}个话题'},
                data: [{value: 50, name: '话题曝光量'}]
            }
        ]
    };

    option.series[0].data[0].value = data;
    myChart_sum.setOption(option, true);
}

</script>
</body>
</html>

```

各时段用户量统计页面 (userCount.html)

```

<!DOCTYPE html>
<html style="height: 100%">
<head>
    <meta charset="utf-8">
    <title>新闻网各时段用户量统计分析 (2021/01/12) </title>
</head>

<body style="height: 100%; margin: 0">

<div id="container" style="height: 90%"></div>

<script src="js/echarts.min.js"></script>
<script src="js/jquery-3.5.1.js"></script>

```



```

<script type="text/javascript">
    var dom = document.getElementById("container");
    var myChart = echarts.init(dom);
    var app = {};
    option = null;

    var text; // 一个变量
    $.ajax({
        type: 'get', // 请求类型
        url: './json/userCount.json', // 文件相对地址（相对于使用这个js脚本的
html 文件）
        dataType: 'json', // 类型
        async: false,
        success: function(data) {
            text = data; // data 就是 json 文件中的数据，可以直接赋给 text，类型为
js 对象，也就是直接从 json 数据转换成了 js 对象
        },
        error: function() {
            alert('请求失败');
        }
    })
    this.text = text; // 使用 json 数据，this.songs 是一个 js 对象

    console.info(text.map(x => {return x.datatime}));
    var datatime = text.map(x => {return x.datatime});
    var userCount = text.map(x => {return x.userCount});

    option = {
        title: {
            text: '新闻网各时段用户量统计分析（2021/01/10）'
        },
        tooltip: {
            trigger: 'axis'
        },
        legend: {
            data: ['邮件营销']
        },
        grid: {
            left: '3%',
            right: '4%',

```

```

        bottom: '3%',
        containLabel: true
    },
    toolbox: {
        feature: {
            saveAsImage: {}
        }
    },
    xAxis: {
        type: 'category',
        boundaryGap: false,
        data: datetime
    },
    yAxis: {
        type: 'value'
    },
    dataZoom: [
        {
            type: 'slider',
            start: 1,
            end: 35
        },
        {
            type: 'inside',
            start: 1,
            end: 35
        }
    ],
    series: [
        {
            name: '用户量',
            type: 'line',
            stack: '总量',
            data: userCount
        }
    ]
};
if (option && typeof option === "object") {
    myChart.setOption(option, true);
}
</script>
</body>
</html>

```

各时段话题量统计页面 (userCount.html)

```
<!DOCTYPE html>
<html style="height: 100%">
<head>
  <meta charset="utf-8">
  <title>新闻网各时段话题量统计分析 (2021/01/12) </title>
</head>

<body style="height: 100%; margin: 0">

<div id="container" style="height: 90%"></div>

<script src="js/echarts.min.js"></script>
<script src="js/jquery-3.5.1.js"></script>

<script type="text/javascript">
  var dom = document.getElementById("container");
  var myChart = echarts.init(dom);
  var app = {};
  option = null;

  // 这段代码可以导入一个"./static/json/infos.json"文件,
  // 并使用json数据,

  var text; // 一个变量
  $.ajax({
    type: 'get', // 请求类型
    url: './json/topicCount.json', // 文件相对地址 (相对于使用这个js脚本的
html 文件)
    dataType: 'json', // 类型
    async: false,
    success: function(data) {
      text = data; // data 就是 json 文件中的数据, 可以直接赋给 text, 类型为
js 对象, 也就是直接从 json 数据转换成了 js 对象
    },
    error: function() {
      alert('请求失败');
    }
  })
  this.text = text; // 使用 json 数据, this.songs 是一个 js 对象
```

```

console.info(text.map(x => {return x.datetime}));
var datetime = text.map(x => {return x.datetime});
var topicCount = text.map(x => {return x.topicCount});

```

```

option = {
  title: {
    text: '新闻网各时段话题量统计分析（2021/01/10）'
  },
  tooltip: {
    trigger: 'axis'
  },
  legend: {
    data: ['邮件营销']
  },
  grid: {
    left: '3%',
    right: '4%',
    bottom: '3%',
    containLabel: true
  },
  toolbox: {
    feature: {
      saveAsImage: {}
    }
  },
  xAxis: {
    type: 'category',
    boundaryGap: false,
    data: datetime
  },
  yAxis: {
    type: 'value'
  },
  dataZoom: [
    {
      type: 'slider',
      start: 1,
      end: 35
    },
    {
      type: 'inside',
      start: 1,

```

```

        end: 35
    }
],
series: [
    {
        name: '话题量',
        type: 'line',
        stack: '总量',
        data: topicCount
    }
]
};
if (option && typeof option === "object") {
    myChart.setOption(option, true);
}
</script>
</body>
</html>

```

话题关键词云页面 (wordCloud.html)

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>词云 (TOP50) (2021/01/13) </title>
    <script
src='https://cdn.bootcss.com/echarts/3.7.0/echarts.simple.js'></script>
    <script src='js/echarts-wordcloud.js'></script>
    <meta charset="UTF-8">
    <title>test (2021/01/13) </title>

    <style>
        html, body, #main {
            width: 100%;
            height: 100%;
            margin: 0;
        }
    </style>
</head>
<body>
<div id='main'></div>
<script>

```

```

var chart = echarts.init(document.getElementById('main'));

var option = {
  tooltip: {},
  series: [{
    type: 'wordCloud',
    gridSize: 15,
    sizeRange: [30, 100],
    rotationRange: [-90, 90],
    rotationStep: 10,
    shape: 'pentagon',
    width: 1920,
    height: 1080,
    drawOutOfBound: true,
    textStyle: {
      normal: {
        color: function () {
          return 'rgb(' + [
            Math.round(Math.random() * 160),
            Math.round(Math.random() * 160),
            Math.round(Math.random() * 160)
          ].join(',') + ')';
        }
      },
      emphasis: {
        shadowBlur: 10,
        shadowColor: '#333'
      }
    },
    data: [...]
  }]
};

chart.setOption(option);

window.onresize = chart.resize;
</script>
</body>
</html>

```

网站热度词云展示页面 (hostCloud.html)

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>网站词云（2021/01/14）</title>
  <script
src='https://cdn.bootcss.com/echarts/3.7.0/echarts.simple.js'></script>
  <script src='js/echarts-wordcloud.js'></script>
  <meta charset="UTF-8">
  <title>test（2021/01/13）</title>

  <style>
    html, body, #main {
      width: 100%;
      height: 100%;
      margin: 0;
    }
  </style>
</head>
<body>
<div id='main'></div>
<script>
  var chart = echarts.init(document.getElementById('main'));

  var option = {
    tooltip: {},
    series: [{
      type: 'wordCloud',
      gridSize: 15,
      sizeRange: [10, 80],
      rotationRange: [-90, 90],
      rotationStep: 10,
      shape: 'circle',
      width: 1920,
      height: 1080,
      drawOutOfBound: false,
      textStyle: {
        normal: {
          color: function () {
            return 'rgb(' + [
              Math.round(Math.random() * 160),
              Math.round(Math.random() * 160),
              Math.round(Math.random() * 160)
            ].join(',') + ')';
          }
        }
      }
    }]
  };

```

```

    },
    emphasis: {
      shadowBlur: 10,
      shadowColor: '#333'
    }
  },
  data: [...]
}]
};

chart.setOption(option);

window.onresize = chart.resize;
</script>
</body>
</html>

```

代码编写好后，可直接编译运行，或打 jar 提交到 spark 环境运行。

5. 完成情况与问题讨论

5.1 实际完成情况（完成程度）

集群运行稳定，各组将功能正常，分析结果可视化正常。

最终展示页面如下图所示。

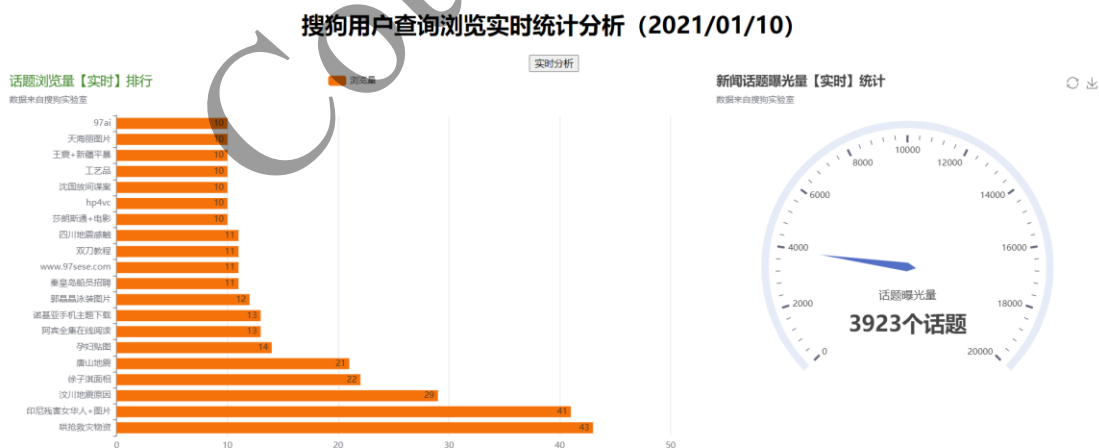


图 12 搜狗用户查询浏览实时统计分析

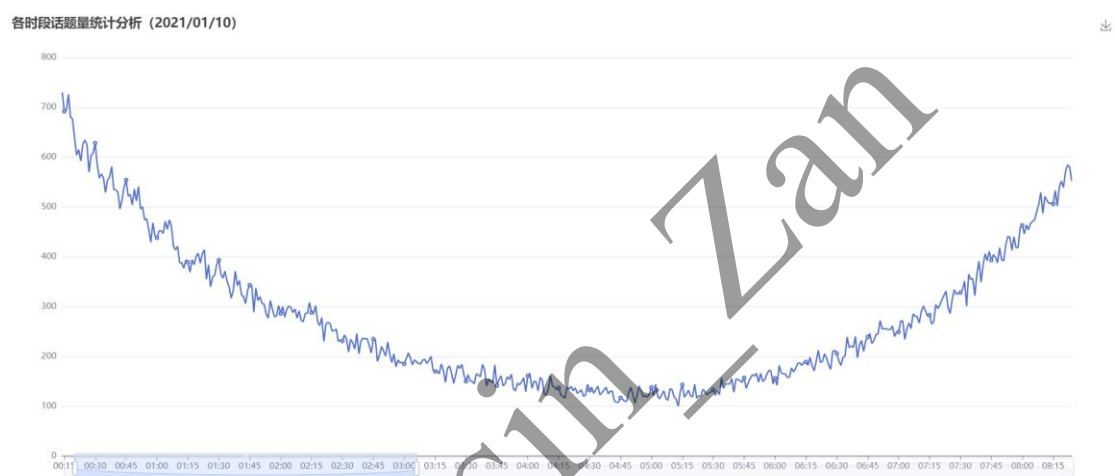
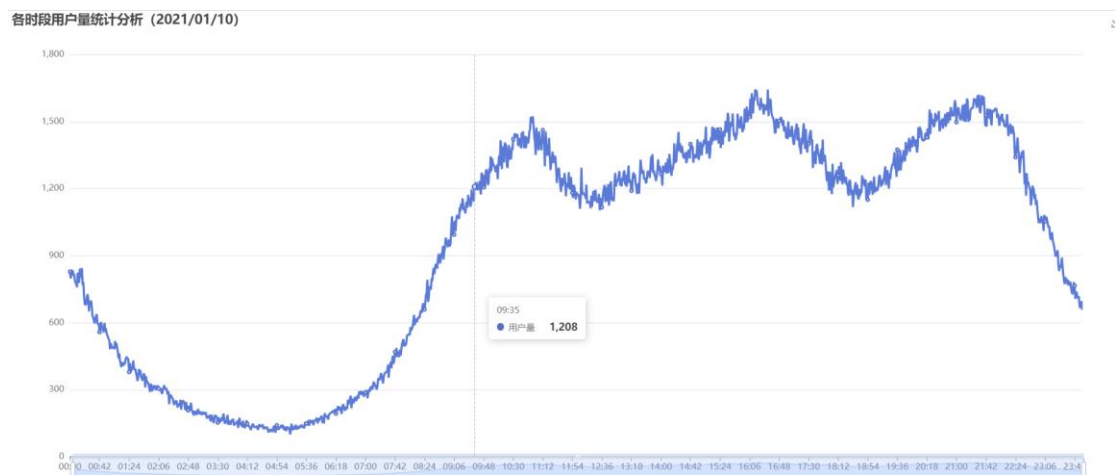




图 16 网站热度词云

5.2 问题与讨论（碰到什么问题，难点，如何解决，后期的思考）

问题：

- 1) 各节点间网络延时较大，集群启动失败。
- 2) 节点 ip 地址冲突
- 3) Hadoop HA 无法启动
- 4) Flume 写入 HBase 数据报错
- 5) Flume 和 Kafka 集成错误:Error reading field 'throttle_time_ms'
- 6) spark structured streaming 报 Failed to find data source:kafka
- 7) spark 版本与其他组件兼容性问题
- 8) spark 任务提交提示 jar 错误问题
- 9) 前端界面无法获取数据问题

解决方案：

- 1) 虚拟网络连接方式改为桥接方式，改为静态 IP，电脑主机连接至物理路由器（最好是有线网络连接方式）
- 2) 在克隆虚拟机时，容易忽视 MAC 地址这一项。每个节点公用同一个 MAC 物理网卡地址，会出现网络冲突。
- 3) 应该保证主机名与 hosts 本地域名解析文件中的主机名保持一致，以便集群内各节点可以互相找到
- 4) 数据源问题，部分网址 url 中存在逗号分隔等特殊符号，容易导致读取

时产生数组越界的异常行为。

5) Flume 与 Kafka 版本不兼容所致，更换至兼容版本即可。

6) 在 pom.xml 加这个配置即可。

```
<dependency>
  <groupId>org.apache.spark</groupId>
  <artifactId>spark-sql-kafka-0-10_${scala.version}</artifactId>
  <version>${spark.version}</version>
</dependency>
```

7) 由于 Spark 更新速度较快，预编译版本尚未完全覆盖到各个版本的组件，故需要自行根据源代码进行修改编译。

8) 执行以下命令即可。

```
zip -d your.jar 'META-INF/.SF' 'META-INF/.RSA' 'META-INF/*SF'
```

9) 大多由于 ws 协议地址指明错误所致，需要根据配置 Tomcat 时产生的地址进行修改。

难点：

- 1) 笔记本电脑硬件设施基础较差，需要根据适用情况对虚拟机配置进行合理的修改。
- 2) 网络教程写作水平参差不齐，很多教程是从其他教程搬运而来，解决方案大体相同，但问题修复成功率不高，需要后续参考官方文档 API 等资料自行解决。
- 3) 由于学识有限，需要学习的新事物较为宽泛，对我们来说是一个极大的挑战。
- 4) HA 集群配置部分、Flume 集成开发部分、以及前后端开发这几个阶段花费时间较长，遇到的问题较多，学习难度较大。

后期思考

- 1) 该项目整体而言较为丰富而复杂，学习时长较长。但整体流程思路、架构较为清晰，可较为直观的感受与体验大数据分析的魅力所在。
- 2) 实践环节并非是一件容易的事情，遇到的问题很多，解决的方案很多，也会思考的很多。在不断的碰壁过程中，擦除新的火花（想法、创意），是一件十分有意思且有意义的事情。