

# 尚硅谷大数据项目之尚品汇（用户行为数据采集）

（作者：尚硅谷研究院）

版本：V4.1.0

## 第1章 数据仓库概念

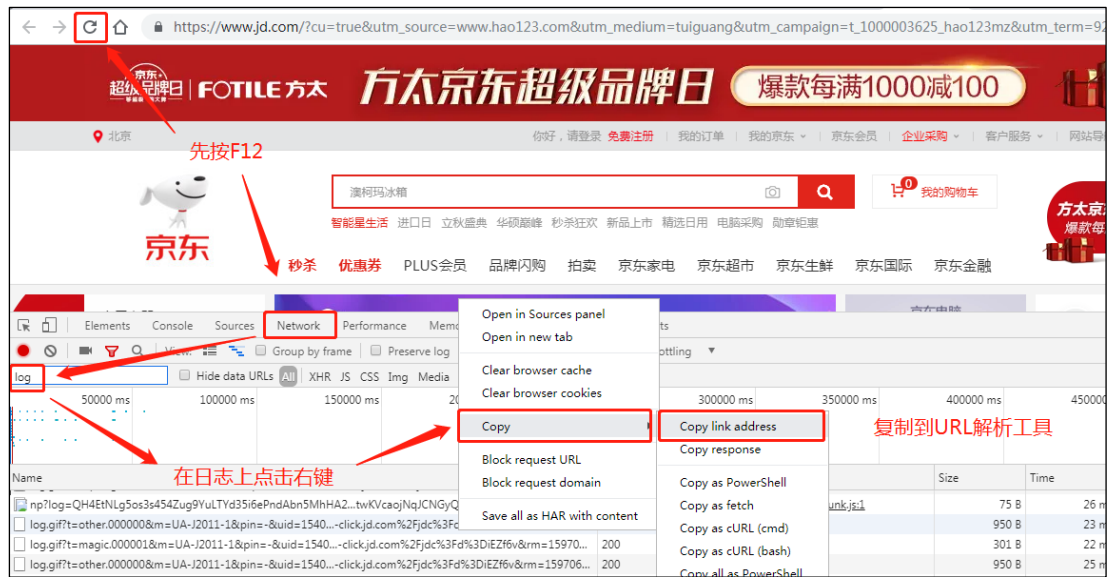
数据仓库（Data Warehouse），是为企业制定决策，提供数据支持的。可以帮助企业，改进业务流程、提高产品质量等。

数据仓库的输入数据通常包括：业务数据、用户行为数据和爬虫数据等

**业务数据：**就是各行业在处理事务过程中产生的数据。比如用户在电商网站中登录、下单、支付等过程中，需要和网站后台数据库进行增删改查交互，产生的数据就是业务数据。业务数据通常存储在 MySQL、Oracle 等数据库中。



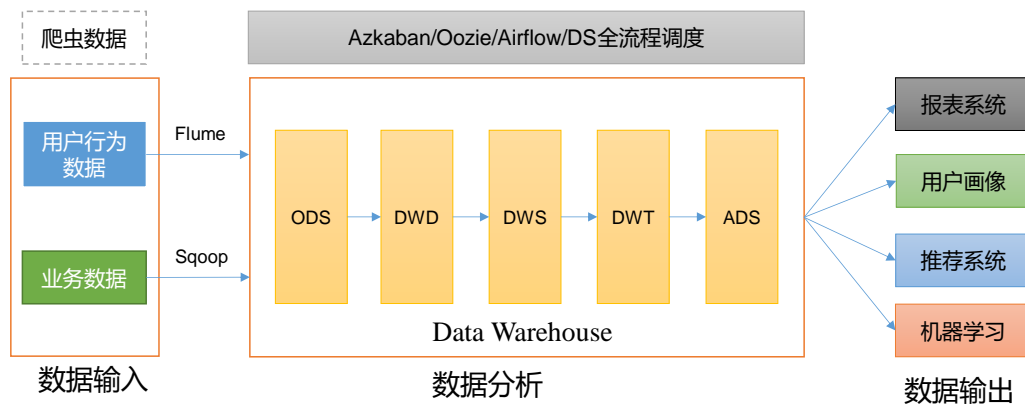
**用户行为数据：**用户在使用产品过程中，通过埋点收集与客户端产品交互过程中产生的数据，并发往日志服务器进行保存。比如页面浏览、点击、停留、评论、点赞、收藏等。用户行为数据通常存储在日志文件中。



**爬虫数据：**通常事通过技术手段获取其他公司网站的数据。不建议同学们这样去做。

## 数据仓库

数据仓库（Data Warehouse），是为企业制定决策，提供数据支持的。可以帮助企业，改进业务流程、提高产品质量等。



数据仓库，并不是数据的最终目的地，而是为数据最终的目的做好准备。这些准备包括对数据的：备份、清洗、聚合、统计等。

让天下没有难学的技术

## 第2章 项目需求及架构设计

### 2.1 项目需求分析



#### 项目需求



##### 一、项目需求

- 1、**用户行为**数据采集平台搭建
- 2、**业务**数据采集平台搭建
- 3、**数据仓库**维度建模
- 4、分析，**设备、会员、商品、地区、活动**等电商核心主题，统计的报表指标近100个。
- 5、采用**即席查询**工具，随时进行指标分析
- 6、对**集群性能进行监控**，发生异常需要报警。
- 7、**元数据管理**
- 8、**质量监控**
- 9、**权限管理**

##### 二、思考题

- 1、项目**技术**如何选型？
- 2、框架版本如何选型（**Apache、CDH、HDP**）
- 3、服务器使用**物理机**还是**云主机**？
- 4、如何确认**集群规模**？（假设每台服务器8T硬盘）

让天下没有难学的技术

### 2.2 项目框架

#### 2.2.1 技术选型



#### 技术选型

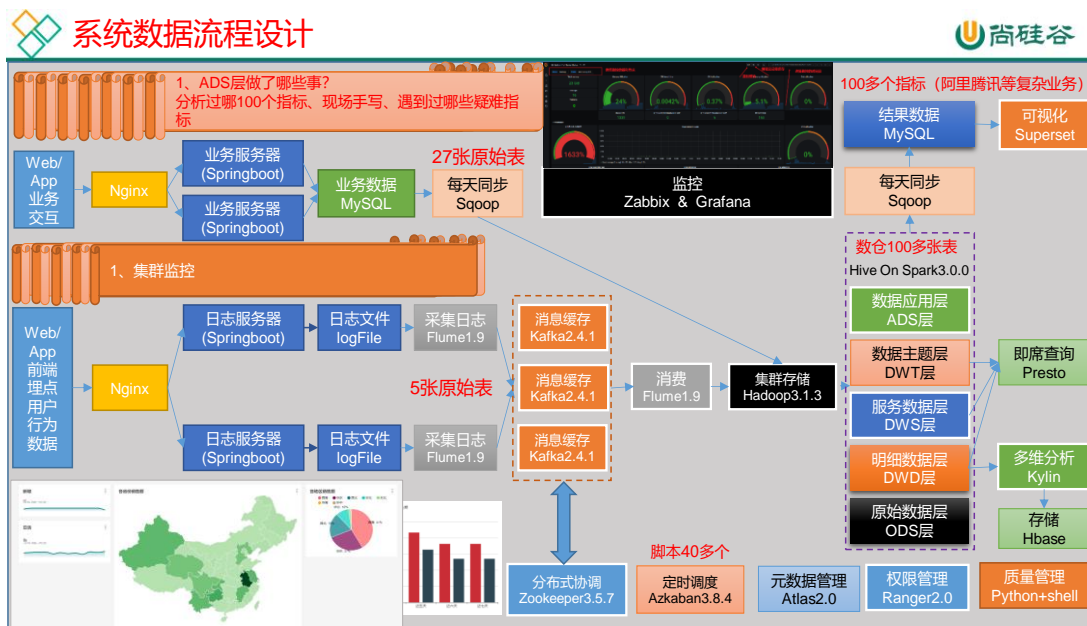


技术选型主要考虑因素：**数据量大小、业务需求、行业内经验、技术成熟度、开发维护成本、总成本预算**

- 数据采集传输：**Flume、Kafka、Sqoop**，Logstash，DataX
- 数据存储：**MySQL、HDFS、HBase**，Redis，MongoDB
- 数据计算：**Hive、Tez、Spark**，Flink，Storm
- 数据查询：**Presto、Kylin**，Impala，Druid，ClickHouse，Doris
- 数据可视化：**Echarts、Superset**，QuickBI，DataV
- 任务调度：**Azkaban**，Oozie，DolphinScheduler，Airflow
- 集群监控：**Zabbix**，Prometheus
- 元数据管理：**Atlas**
- 权限管理：**Ranger**，Sentry

让天下没有难学的技术

## 2.2.2 系统数据流程设计



## 2.2.3 框架版本选型

### 框架发行版本选型

#### 1) 如何选择Apache/CDH/HDP版本？

(1) Apache: 运维麻烦，组件间兼容性需要自己调研。（一般大厂使用，技术实力雄厚，有专业的运维人员）（建议使用）

(2) CDH: 国内使用最多的版本，但CM不开源，今年开始收费，一个节点1万美金/年。

(3) HDP: 开源，可以进行二次开发，但是没有CDH稳定，国内使用较少

#### 2) 云服务选择

(1) 阿里云的EMR、MaxCompute、DataWorks

(2) 亚马逊云EMR

(3) 腾讯云EMR

(4) 华为云EMR

让天下没有难学的技术

### (1) Apache框架版本

框架	旧版本	新版本
Hadoop	2.7.2	3.1.3
Zookeeper	3.4.10	3.5.7
MySQL	5.6.24	5.7.16
Hive	1.2.1	3.1.2
Flume	1.7.0	1.9.0
Kafka	0.11-0.2	2.4.1
Kafka Eagle	1.3.7	1.4.5
Azkaban	2.5.0	3.84.4
Spark	2.1.1	3.0.0
Hbase	1.3.1	2.0.5
Phoenix	4.14.1	5.0.0
Sqoop	1.4.6	1.4.6
Presto	0.189	0.189
Kylin	2.5.1	3.0.1
Atlas	0.8.4	2.0.0
Ranger	2.0.0	2.0.0
Solr	5.2.1	7.7.0

注意事项：框架选型尽量不要选择最新的框架，选择最新框架半年前左右的稳定版。

让天下没有难学的技术

## 2.2.4 服务器选型

服务器选择物理机还是云主机？

### 1) 物理机：

- 以128G内存，20核物理CPU，40线程，8THDD和2TSSD硬盘，戴尔品牌单台报价4W出头。一般物理机寿命5年左右。
- 需要有专业的运维人员，平均一个月1万。电费也是不少的开销。

### 2) 云主机：

- 云主机：以阿里云为例，差不多相同配置，每年5W。
- 很多运维工作都由阿里云完成，运维相对较轻松

### 3) 企业选择

- 金融有钱公司和阿里没有直接冲突的公司选择阿里云
- 中小公司、为了融资上市，选择阿里云，拉倒融资后买物理机。
- 有长期打算，资金比较足，选择物理机。

让天下没有难学的技术

## 2.2.5 集群规模

### 集群规模

- 1) 如何确认集群规模? (假设: 每台服务器8T磁盘, 128G内存)
  - (1) 每天日活跃用户100万, 每人一天平均100条:  $100万 * 100条 = 1亿条$
  - (2) 每条日志1K左右, 每天1亿条:  $100000000 / 1024 / 1024 = 约100G$
  - (3) 半年内不扩容服务器来算:  $100G * 180天 = 约18T$
  - (4) 保存3副本:  $18T * 3 = 54T$
  - (5) 预留20%~30%Buf:  $54T / 0.7 = 77T$
  - (6) 算到这:  $约8T * 10台服务器$
- 2) 如果考虑数仓分层? 数据采用压缩? 需要重新再计算

让天下没有难学的技术

## 2.2.6 集群资源规划设计

在企业中通常会搭建一套生产集群和一套测试集群。生产集群运行生产任务, 测试集群用于上线前代码编写和测试。

### 1) 生产集群

- (1) 消耗内存的分开
- (2) 数据传输数据比较紧密的放在一起 (Kafka、Zookeeper)
- (3) 客户端尽量放在一到两台服务器上, 方便外部访问
- (4) 有依赖关系的尽量放到同一台服务器 (例如: Hive 和 Azkaban Executor)

1	2	3	4	5	6	7	8	9	10
nn	nn	dn	dn	dn	dn	dn	dn	dn	dn
		rm	rm	nm	nm	nm	nm	nm	nm
		nm	nm						
							zk	zk	zk
							kafka	kafka	kafka
							Flume	Flume	flume
		Hbase	Hbase	Hbase					
hive	hive								
mysql	mysql								
spark	spark								
Azkaban	Azkaban				ES	ES			

### 2) 测试集群服务器规划

服务名称	子服务	服务器	服务器	服务器
------	-----	-----	-----	-----

更多 [Java](#) - [大数据](#) - [前端](#) - [python](#) 人工智能资料下载, 可百度访问: [尚硅谷官网](#)

		hadoop102	hadoop103	hadoop104
HDFS	NameNode	√		
	DataNode	√	√	√
	SecondaryNameNode			√
Yarn	NodeManager	√	√	√
	Resourcemanager		√	
Zookeeper	Zookeeper Server	√	√	√
Flume（采集日志）	Flume	√	√	
Kafka	Kafka	√	√	√
Flume（消费 Kafka）	Flume			√
Hive	Hive	√		
MySQL	MySQL	√		
Sqoop	Sqoop	√		
Presto	Coordinator	√		
	Worker		√	√
Azkaban	AzkabanWebServer	√		
	AzkabanExecutorServer	√		
Spark		√		
Kylin		√		
HBase	HMaster	√		
	HRegionServer	√	√	√
Superset		√		
Atlas		√		
Solr	Jar	√		
服务数总计		19	8	8

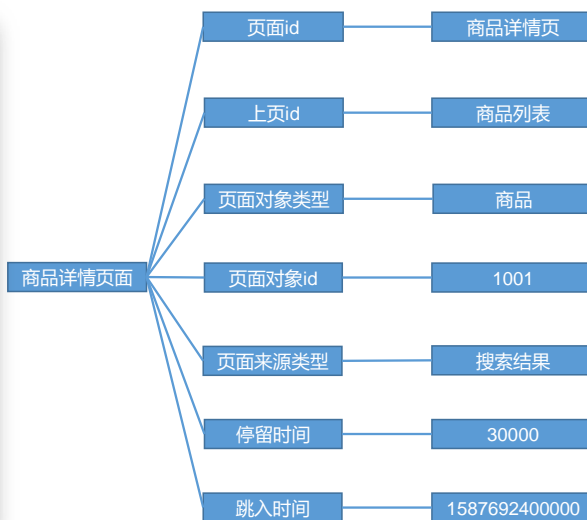
## 第 3 章 数据生成模块

### 3.1 目标数据

我们要收集和分析的数据主要包括**页面数据**、**事件数据**、**曝光数据**、**启动数据**和**错误数据**。

#### 3.1.1 页面

页面数据主要记录一个页面的用户访问情况，包括访问时间、停留时间、页面路径等信息。



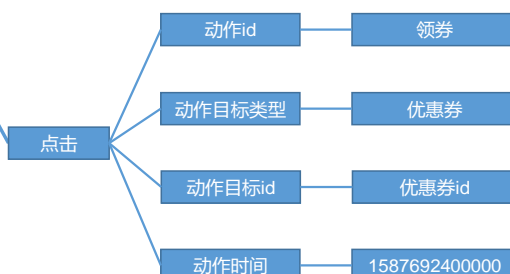
字段名称	字段描述
page_id	<p>页面 id</p> <p>home ("首页"),</p> <p>category ("分类页"),</p> <p>discovery ("发现页"),</p> <p>top_n ("热门排行"),</p> <p>favor ("收藏页"),</p> <p>search ("搜索页"),</p> <p>good_list ("商品列表页"),</p> <p>good_detail ("商品详情"),</p> <p>good_spec ("商品规格"),</p> <p>comment ("评价"),</p> <p>comment_done ("评价完成"),</p> <p>comment_list ("评价列表"),</p> <p>cart ("购物车"),</p> <p>trade ("下单结算"),</p> <p>payment ("支付页面"),</p> <p>payment_done ("支付完成"),</p>



	orders_all ("全部订单"), orders_unpaid ("订单待支付"), orders_undelivered ("订单待发货"), orders_unreceipted ("订单待收货"), orders_wait_comment ("订单待评价"), mine ("我的"), activity ("活动"), login ("登录"), register ("注册");
last_page_id	上页 id
page_item_type	页面对象类型 sku_id ("商品 skuId"), keyword ("搜索关键词"), sku_ids ("多个商品 skuId"), activity_id ("活动 id"), coupon_id ("购物券 id");
page_item	页面对象 id
sourceType	页面来源类型 promotion ("商品推广"), recommend ("算法推荐商品"), query ("查询结果商品"), activity ("促销活动");
during_time	停留时间 (毫秒)
ts	跳入时间

### 3.1.2 事件

事件数据主要记录应用内一个具体操作行为，包括操作类型、操作对象、操作对象描述等信息。

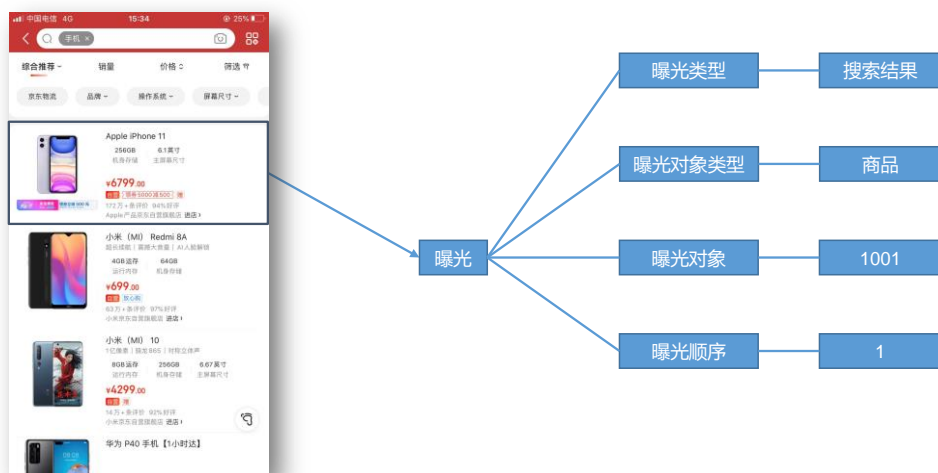


让天下没有难学的技术

字段名称	字段描述
action_id	<p>动作 id</p> <p>favor_add ("添加收藏"),</p> <p>favor_cancel ("取消收藏"),</p> <p>cart_add ("添加购物车"),</p> <p>cart_remove ("删除购物车"),</p> <p>cart_add_num ("增加购物车商品数量"),</p> <p>cart_minus_num ("减少购物车商品数量"),</p> <p>trade_add_address ("增加收货地址"),</p> <p>get_coupon ("领取优惠券");</p> <p>注：对于下单、支付等业务数据，可从业务数据库获取。</p>
item_type	<p>动作目标类型</p> <p>sku_id ("商品"),</p> <p>coupon_id ("购物券");</p>
item	动作目标 id
ts	动作时间

### 3.1.3 曝光

曝光数据主要记录页面所曝光的内容，包括曝光对象，曝光类型等信息。

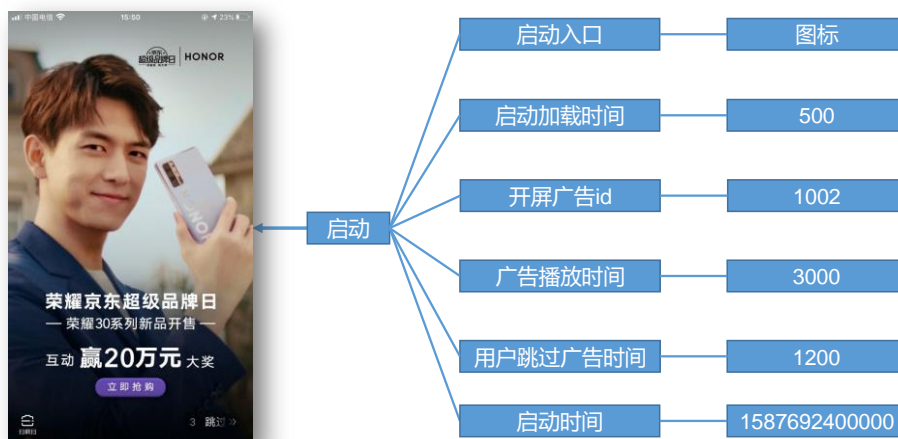


让天下没有难学的技术

字段名称	字段描述
displayType	曝光类型 promotion ("商品推广"), recommend ("算法推荐商品"), query ("查询结果商品"), activity ("促销活动");
item_type	曝光对象类型 sku_id ("商品 skuId"), activity_id ("活动 id");
item	曝光对象 id
order	曝光顺序

### 3.1.4 启动

启动数据记录应用的启动信息。



让天下没有难学的技术

字段名称	字段描述
entry	启动入口 icon ("图标"), notification ("通知"), install ("安装后启动");
loading_time	启动加载时间
open_ad_id	开屏广告 id
open_ad_ms	广告播放时间
open_ad_skip_ms	用户跳过广告时间
ts	启动时间

### 3.1.5 错误

错误数据记录应用使用

过程中的错误信息，包括错误编号及错误信息。

字段名称	字段描述
error_code	错误码
msg	错误信息

## 3.2 数据埋点

### 3.2.1 主流埋点方式（了解）

目前主流的埋点方式，有代码埋点（前端/后端）、可视化埋点、全埋点三种。

代码埋点是通过调用埋点 SDK 函数，在需要埋点的业务逻辑功能位置调用接口，上报埋点数据。例如，我们对页面中的某个按钮埋点后，当这个按钮被点击时，可以在这个按钮

对应的 OnClick 函数里面调用 SDK 提供的数据发送接口，来发送数据。

**可视化埋点**只需要研发人员集成采集 SDK，不需要写埋点代码，业务人员就可以通过访问分析平台的“圈选”功能，来“圈”出需要对用户行为进行捕捉的控件，并对该事件进行命名。圈选完毕后，这些配置会同步到各个用户的终端上，由采集 SDK 按照圈选的配置自动进行用户行为数据的采集和发送。

**全埋点**是通过在产品中嵌入 SDK，前端自动采集页面上的全部用户行为事件，上报埋点数据，相当于做了一个统一的埋点。然后再通过界面配置哪些数据需要在系统里面进行分析。

### 3.2.2 埋点数据上报时机

埋点数据上报时机包括两种方式。

方式一，在离开该页面时，上传在这个页面产生的所有数据（页面、事件、曝光、错误等）。优点，批处理，减少了服务器接收数据压力。缺点，不是特别及时。

方式二，每个事件、动作、错误等，产生后，立即发送。优点，响应及时。缺点，对服务器接收数据压力比较大。

本次项目采用方式一埋点。

### 3.2.3 埋点数据日志结构

我们的日志结构大致可分为两类，一是普通页面埋点日志，二是启动日志。

普通页面日志结构如下，每条日志包含了，当前页面的**页面信息**，所有**事件（动作）**、所有**曝光信息**以及**错误信息**。除此之外，还包含了一系列**公共信息**，包括设备信息，地理位置，应用信息等，即下边的 **common** 字段。

（1）普通页面埋点日志格式

```
{
  "common": {                                -- 公共信息
    "ar": "230000",                          -- 地区编码
    "ba": "iPhone",                          -- 手机品牌
    "ch": "Appstore",                        -- 渠道
    "is_new": "1", --是否首日使用，首次使用的当日，该字段值为1，过了24:00，该字段置为0。
    "md": "iPhone 8",                        -- 手机型号
    "mid": "YXfhjAYH6As2z9Iq",              -- 设备 id
    "os": "iOS 13.2.9",                      -- 操作系统
    "uid": "485",                            -- 会员 id
    "vc": "v2.1.134"                         -- app 版本号
  },
  "actions": [                               --动作(事件)
    {
```

更多 [Java](#) - [大数据](#) - [前端](#) - [python](#) 人工智能资料下载，可百度访问：[尚硅谷官网](#)

```

        "action_id": "favor_add",    --动作 id
        "item": "3",                --目标 id
        "item_type": "sku_id",      --目标类型
        "ts": 1585744376605         --动作时间戳
    }
],
"displays": [
    {
        "displayType": "query",      -- 曝光类型
        "item": "3",                 -- 曝光对象 id
        "item_type": "sku_id",       -- 曝光对象类型
        "order": 1,                  --出现顺序
        "pos_id": 2                  --曝光位置
    },
    {
        "displayType": "promotion",
        "item": "6",
        "item_type": "sku_id",
        "order": 2,
        "pos_id": 1
    },
    {
        "displayType": "promotion",
        "item": "9",
        "item_type": "sku_id",
        "order": 3,
        "pos_id": 3
    },
    {
        "displayType": "recommend",
        "item": "6",
        "item_type": "sku_id",
        "order": 4,
        "pos_id": 2
    },
    {
        "displayType": "query ",
        "item": "6",
        "item_type": "sku_id",
        "order": 5,
        "pos_id": 1
    }
],
"page": {                            --页面信息
    "during_time": 7648,              -- 持续时间毫秒
    "item": "3",                      -- 目标 id
    "item_type": "sku_id",            -- 目标类型
    "last_page_id": "login",          -- 上页类型
    "page_id": "good_detail",         -- 页面 ID
    "sourceType": "promotion"         -- 来源类型
},
"err": {                             --错误
    "error_code": "1234",             --错误码
    "msg": "*****"                  --错误信息
},

```

```
"ts": 1585744374423 --跳入时间戳
}
```

## (2) 启动日志格式

启动日志结构相对简单，主要包含公共信息，启动信息和错误信息。

```
{
  "common": {
    "ar": "370000",
    "ba": "Honor",
    "ch": "wandoujia",
    "is_new": "1",
    "md": "Honor 20s",
    "mid": "eQF5boERMJFOujcp",
    "os": "Android 11.0",
    "uid": "76",
    "vc": "v2.1.134"
  },
  "start": {
    "entry": "icon", --icon 手机图标 notice 通知 install
    安装后启动
    "loading_time": 18803, --启动加载时间
    "open_ad_id": 7, --广告页 ID
    "open_ad_ms": 3449, -- 广告总共播放时间
    "open_ad_skip_ms": 1989 -- 用户跳过广告时点
  },
  "err": { --错误
    "error_code": "1234", --错误码
    "msg": "*****" --错误信息
  },
  "ts": 1585744304000
}
```

## 3.3 服务器和 JDK 准备

### 3.3.2 阿里云服务器准备（可选）



尚硅谷大数据技术  
之阿里云服务器购买

### 3.3.3 编写集群分发脚本 xsync

#### 1) xsync 集群分发脚本

(1) 需求：循环复制文件到所有节点的相同目录下

(2) 需求分析

①rsync 命令原始拷贝：

```
rsync -av /opt/module root@hadoop103:/opt/
```

②期望脚本：

更多 [Java](#) -[大数据](#) -[前端](#) -[python](#) 人工智能资料下载，可百度访问：尚硅谷官网

xsync 要同步的文件名称

③说明：在/home/atguigu/bin 这个目录下存放的脚本，atguigu 用户可以在系统任何地方直接执行。

### (3) 脚本实现

①在用的家目录/home/atguigu 下创建 bin 文件夹

```
[atguigu@hadoop102 ~]$ mkdir bin
```

②在/home/atguigu/bin 目录下创建 xsync 文件，以便全局调用

```
[atguigu@hadoop102 ~]$ cd /home/atguigu/bin  
[atguigu@hadoop102 ~]$ vim xsync
```

在该文件中编写如下代码

```
#!/bin/bash  
#1. 判断参数个数  
if [ $# -lt 1 ]  
then  
    echo Not Enough Argument!  
    exit;  
fi  
#2. 遍历集群所有机器  
for host in hadoop102 hadoop103 hadoop104  
do  
    echo ===== $host =====  
    #3. 遍历所有目录，挨个发送  
    for file in $@  
    do  
        #4 判断文件是否存在  
        if [ -e $file ]  
        then  
            #5. 获取父目录  
            pdir=$(cd -P $(dirname $file); pwd)  
            #6. 获取当前文件的名称  
            fname=$(basename $file)  
            ssh $host "mkdir -p $pdir"  
            rsync -av $pdir/$fname $host:$pdir  
        else  
            echo $file does not exists!  
        fi  
    done  
done
```

③修改脚本 xsync 具有执行权限

```
[atguigu@hadoop102 bin]$ chmod +x xsync
```

④测试脚本

```
[atguigu@hadoop102 bin]$ xsync xsync
```

## 3.3.4 SSH 无密登录配置

说明：这里面只配置了 hadoop102、hadoop103 到其他主机的无密登录；因为 hadoop102

未外配置的是 NameNode，hadoop103 配置的是 ResourceManager，都要求对其他节点无密

更多 [Java](#) - [大数据](#) - [前端](#) - [python](#) 人工智能资料下载，可百度访问：尚硅谷官网



访问。

(1) hadoop102 上生成公钥和私钥:

```
[atguigu@hadoop102 .ssh]$ ssh-keygen -t rsa
```

然后敲（三个回车），就会生成两个文件 id\_rsa（私钥）、id\_rsa.pub（公钥）

(2) 将 hadoop102 公钥拷贝到要免密登录的目标机器上

```
[atguigu@hadoop102 .ssh]$ ssh-copy-id hadoop102
[atguigu@hadoop102 .ssh]$ ssh-copy-id hadoop103
[atguigu@hadoop102 .ssh]$ ssh-copy-id hadoop104
```

(3) hadoop103 上生成公钥和私钥:

```
[atguigu@hadoop103 .ssh]$ ssh-keygen -t rsa
```

然后敲（三个回车），就会生成两个文件 id\_rsa（私钥）、id\_rsa.pub（公钥）

(4) 将 hadoop103 公钥拷贝到要免密登录的目标机器上

```
[atguigu@hadoop103 .ssh]$ ssh-copy-id hadoop102
[atguigu@hadoop103 .ssh]$ ssh-copy-id hadoop103
[atguigu@hadoop103 .ssh]$ ssh-copy-id hadoop104
```

### 3.3.5 JDK 准备

#### 1) 卸载现有 JDK (3 台节点)

```
[atguigu@hadoop102 opt]# sudo rpm -qa | grep -i java | xargs -n1 sudo rpm -e --nodeps
[atguigu@hadoop103 opt]# sudo rpm -qa | grep -i java | xargs -n1 sudo rpm -e --nodeps
[atguigu@hadoop104 opt]# sudo rpm -qa | grep -i java | xargs -n1 sudo rpm -e --nodeps
```

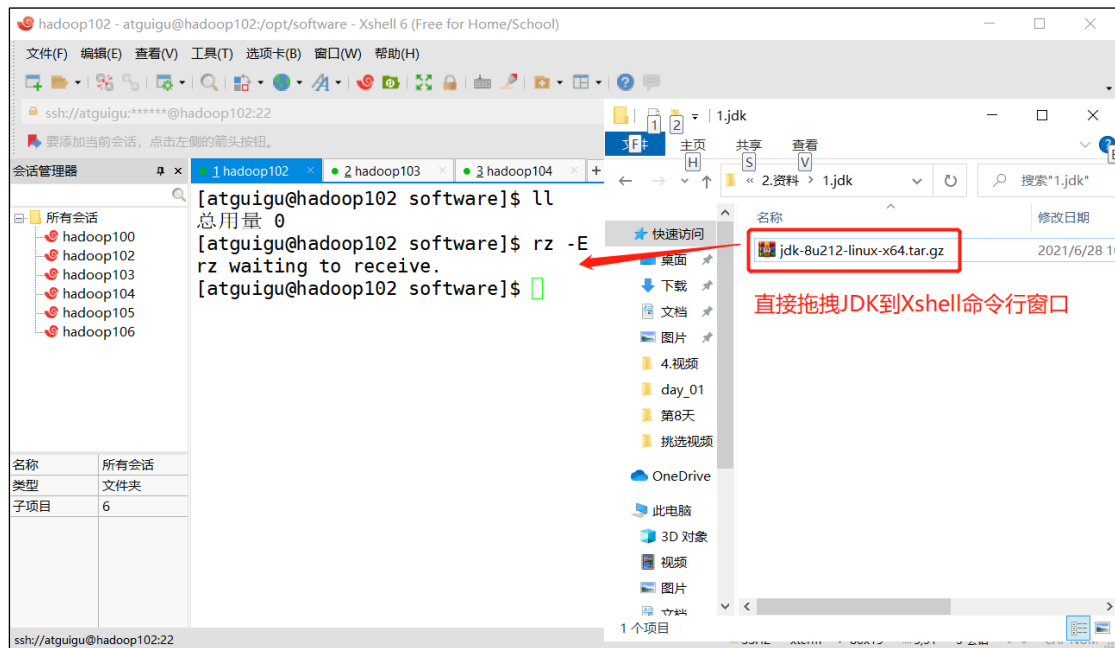
(1) rpm -qa: 表示查询所有已经安装的软件包

(2) grep -i: 表示过滤时不区分大小写

(3) xargs -n1: 表示一次获取上次执行结果的一个值

(4) rpm -e --nodeps: 表示卸载软件

#### 2) 用 XShell 工具将 JDK 导入到 hadoop102 的 /opt/software 文件夹下面



### 3) 在 Linux 系统下的 opt 目录中查看软件包是否导入成功

```
[atguigu@hadoop102 software]# ls /opt/software/
```

看到如下结果:

```
jdk-8u212-linux-x64.tar.gz
```

### 4) 解压 JDK 到/opt/module 目录下

```
[atguigu@hadoop102 software]# tar -zxvf jdk-8u212-linux-x64.tar.gz -C /opt/module/
```

### 5) 配置 JDK 环境变量

(1) 新建/etc/profile.d/my\_env.sh 文件

```
[atguigu@hadoop102 module]# sudo vim /etc/profile.d/my_env.sh
```

添加如下内容, 然后保存 (:wq) 退出

```
#JAVA_HOME
export JAVA_HOME=/opt/module/jdk1.8.0_212
export PATH=$PATH:$JAVA_HOME/bin
```

(2) 让环境变量生效

```
[atguigu@hadoop102 software]$ source /etc/profile.d/my_env.sh
```

### 6) 测试 JDK 是否安装成功

```
[atguigu@hadoop102 module]# java -version
```

如果能看到以下结果、则 Java 正常安装

```
java version "1.8.0_212"
```

### 7) 分发 JDK

```
[atguigu@hadoop102 module]$ xsync /opt/module/jdk1.8.0_212/
```

### 8) 分发环境变量配置文件

```
[atguigu@hadoop102 module]$ sudo /home/atguigu/bin/rsync /etc/profile.d/my_env.sh
```

### 9) 分别在 hadoop103、hadoop104 上执行 source

更多 [Java](#) -[大数据](#) -[前端](#) -[python](#) 人工智能资料下载, 可百度访问: [尚硅谷官网](#)

```
[atguigu@hadoop103 module]$ source /etc/profile.d/my_env.sh  
[atguigu@hadoop104 module]$ source /etc/profile.d/my_env.sh
```

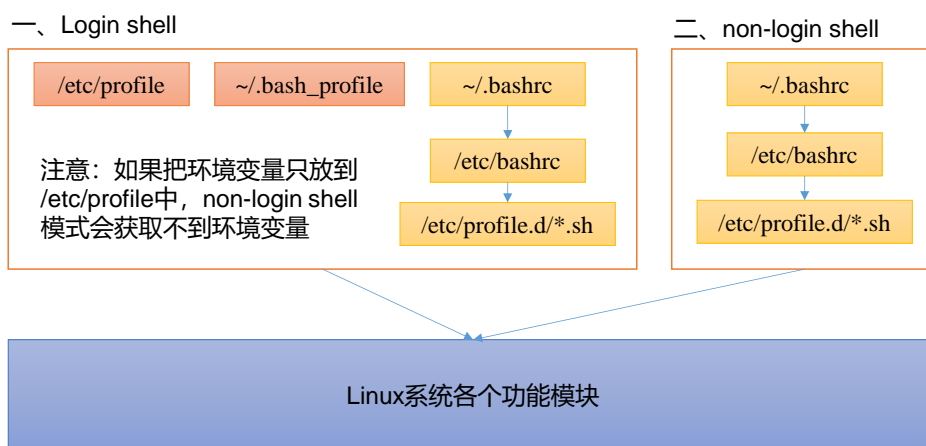
### 3.3.6 环境变量配置说明

Linux 的环境变量可在多个文件中配置，如/etc/profile，/etc/profile.d/\*.sh， ~/.bashrc， ~/.bash\_profile 等，下面说明上述几个文件之间的关系和区别。

bash 的运行模式可分为 **login shell** 和 **non-login shell**。

例如，我们通过终端，输入用户名、密码，登录系统之后，得到就是一个 login shell。而当我们执行以下命令 **ssh hadoop103 command**，在 hadoop103 执行 command 的就是一个 non-login shell。

#### 登录Shell和非登录Shell区别



让天下没有难学的技术

这两种 shell 的主要区别在于，它们启动时会加载不同的配置文件，**login shell** 启动时会加载/etc/profile， ~/.bash\_profile， ~/.bashrc。**non-login shell** 启动时会加载~/.bashrc。

而在加载~/.bashrc（实际是~/.bashrc 中加载的/etc/bashrc）或/etc/profile 时，都会执行如下代码片段，

```
for i in /etc/profile.d/*.sh /etc/profile.d/sh.local ; do
    if [ -r "$i" ]; then
        if [ "${-#*i}" != "$-" ]; then
            . "$i"
        else
            . "$i" >/dev/null
        fi
    fi
done
```

因此不管是 login shell 还是 non-login shell，启动时都会加载/etc/profile.d/\*.sh 中的环境变量。

## 3.4 模拟数据

### 3.4.1 使用说明

1) 将 `application.yml`、`gmall2020-mock-log-2021-01-22.jar`、`path.json`、`logback.xml` 上传到 `hadoop102` 的 `/opt/module/applog` 目录下

(1) 创建 `applog` 路径

```
[atguigu@hadoop102 module]$ mkdir /opt/module/applog
```

(2) 上传文件 `application.yml` 到 `/opt/module/applog` 目录

#### 2) 配置文件

(1) `application.yml` 文件

可以根据需求生成对应日期的用户行为日志。

```
[atguigu@hadoop102 applog]$ vim application.yml
```

修改如下内容

```
# 外部配置打开
# 外部配置打开
logging.config: "../logback.xml"
#业务日期 注意：并不是 Linux 系统生成日志的日期，而是生成数据中的时间
mock.date: "2020-06-14"

#模拟数据发送模式
#mock.type: "http"
#mock.type: "kafka"
mock.type: "log"

#http 模式下，发送的地址
mock.url: "http://hdp1/applog"

#kafka 模式下，发送的地址
mock:
  kafka-server: "hdp1:9092,hdp2:9092,hdp3:9092"
  kafka-topic: "ODS_BASE_LOG"

#启动次数
mock.startup.count: 200
#设备最大值
mock.max.mid: 500000
#会员最大值
mock.max.uid: 100
#商品最大值
mock.max.sku-id: 35
#页面平均访问时间
mock.page.during-time-ms: 20000
#错误概率 百分比
mock.error.rate: 3
#每条日志发送延迟 ms
mock.log.sleep: 10
```

更多 [Java](#) - [大数据](#) - [前端](#) - [python](#) 人工智能资料下载，可百度访问：尚硅谷官网

```
#商品详情来源 用户查询, 商品推广, 智能推荐, 促销活动
mock.detail.source-type-rate: "40:25:15:20"
#领取购物券概率
mock.if_get_coupon_rate: 75
#购物券最大id
mock.max.coupon-id: 3
#搜索关键词
mock.search.keyword: "图书,小米,iphone11,电视,口红,ps5,苹果手机,小米盒子"
```

(2) path.json, 该文件用来配置访问路径

根据需求, 可以灵活配置用户点击路径。

```
[
  {"path":["home","good_list","good_detail","cart","trade","payment"],"rate":20 },
  {"path":["home","search","good_list","good_detail","login","good_detail","cart","trade","payment"],"rate":40 },
  {"path":["home","mine","orders_unpaid","trade","payment"],"rate":10 },
  {"path":["home","mine","orders_unpaid","good_detail","good_spec","comment","trade","payment"],"rate":5 },
  {"path":["home","mine","orders_unpaid","good_detail","good_spec","comment","home"],"rate":5 },
  {"path":["home","good_detail"],"rate":10 },
  {"path":["home"],"rate":10 }
]
```

(3) logback 配置文件

可配置日志生成路径, 修改内容如下

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <property name="LOG_HOME" value="/opt/module/applog/log" />
  <appender name="console"
class="ch.qos.logback.core.ConsoleAppender">
    <encoder>
      <pattern>%msg%n</pattern>
    </encoder>
  </appender>

  <appender name="rollingFile"
class="ch.qos.logback.core.rolling.RollingFileAppender">
    <rollingPolicy
class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
      <fileNamePattern>${LOG_HOME}/app.%d{yyyy-MM-dd}.log</fileNamePattern>
    </rollingPolicy>
    <encoder>
      <pattern>%msg%n</pattern>
    </encoder>
  </appender>

  <!-- 将某一个包下日志单独打印日志 -->
  <logger name="com.atgugu.gmall2020.mock.log.util.LogUtil"
level="INFO" additivity="false">
    <appender-ref ref="rollingFile" />
  </logger>
</configuration>
```

```
<appender-ref ref="console" />
</logger>

<root level="error" >
  <appender-ref ref="console" />
</root>
</configuration>
```

### 3) 生成日志

(1) 进入到/opt/module/applog 路径，执行以下命令

```
[atguigu@hadoop102 applog]$ java -jar gmall2020-mock-log-2021-01-22.jar
```

(2) 在/opt/module/applog/log 目录下查看生成日志

```
[atguigu@hadoop102 log]$ ll
```

### 3.4.2 集群日志生成脚本

在 hadoop102 的/home/atguigu 目录下创建 bin 目录，这样脚本可以在服务器的任何目录执行。

```
[atguigu@hadoop102 ~]$ echo $PATH
/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/home/atguigu/.local/bin:/home/atguigu/bin
```

(1) 在/home/atguigu/bin 目录下创建脚本 lg.sh

```
[atguigu@hadoop102 bin]$ vim lg.sh
```

(2) 在脚本中编写如下内容

```
#!/bin/bash
for i in hadoop102 hadoop103; do
  echo "===== $i ====="
  ssh $i "cd /opt/module/applog/; java -jar gmall2020-mock-log-2021-01-22.jar >/dev/null 2>&1 &"
done
```

注：

①/opt/module/applog/为 jar 包及配置文件所在路径

②/dev/null 代表 Linux 的空设备文件，所有往这个文件里面写入的内容都会丢失，俗称“黑洞”。

标准输入 0：从键盘获得输入 /proc/self/fd/0

标准输出 1：输出到屏幕（即控制台） /proc/self/fd/1

错误输出 2：输出到屏幕（即控制台） /proc/self/fd/2

(3) 修改脚本执行权限

```
[atguigu@hadoop102 bin]$ chmod u+x lg.sh
```

(4) 将 jar 包及配置文件上传至 hadoop103 的/opt/module/applog/路径

(5) 启动脚本

```
[atguigu@hadoop102 module]$ lg.sh
```

更多 [Java](#) - [大数据](#) - [前端](#) - [python](#) 人工智能资料下载，可百度访问：尚硅谷官网

(6) 分别在 hadoop102、hadoop103 的/opt/module/applog/log 目录上查看生成的数据

```
[atguigu@hadoop102 logs]$ ls
app.2020-06-14.log
[atguigu@hadoop103 logs]$ ls
app.2020-06-14.log
```

### 3.3.1 服务器准备

安装如下文档配置步骤，分别安装 hadoop102、hadoop103、hadoop104 三台主机。



尚硅谷大数据技术  
之hadoop（服务器

## 第 4 章 数据采集模块

### 4.1 集群所有进程查看脚本

(1) 在/home/atguigu/bin 目录下创建脚本 xcall.sh

```
[atguigu@hadoop102 bin]$ vim xcall.sh
```

(2) 在脚本中编写如下内容

```
#!/bin/bash

for i in hadoop102 hadoop103 hadoop104
do
    echo ----- $i -----
    ssh $i "$*"
done
```

(3) 修改脚本执行权限

```
[atguigu@hadoop102 bin]$ chmod 777 xcall.sh
```

(4) 启动脚本

```
[atguigu@hadoop102 bin]$ xcall.sh jps
```

### 4.2 Hadoop 安装

详见：尚硅谷大数据技术之 Hadoop（入门）



尚硅谷大数据技术  
之Hadoop（入门）

(1) 集群规划

	服务器 hadoop102	服务器 hadoop103	服务器 hadoop104
HDFS	NameNode DataNode	DataNode	DataNode SecondaryNameNode

更多 [Java](#) - [大数据](#) - [前端](#) - [python](#) 人工智能资料下载，可百度访问：尚硅谷官网

Yarn	NodeManager	Resourcemanager NodeManager	NodeManager
------	-------------	--------------------------------	-------------

注意：尽量使用离线方式安装

## 4.2.1 项目经验之 HDFS 存储多目录

(1) 给 Linux 系统新增加一块硬盘

参考：<https://www.cnblogs.com/yujianadu/p/10750698.html>

(2) 生产环境服务器磁盘情况

```
[master@slave03 ~]$ df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda7        3.6T  1.6T  2.0T  45% /
devtmpfs         63G   0    63G   0% /dev
tmpfs            63G   20K   63G   1% /dev/shm
tmpfs            63G  714M   63G   2% /run
tmpfs            63G   0    63G   0% /sys/fs/cgroup
/dev/sdc1         954G  177G   777G  19% /hd3
/dev/sdd1         954G  177G   777G  19% /hd4
/dev/sdb1         3.7T  989G   2.7T  27% /hd2
/dev/sda5        1014M  123M   892M  13% /boot
```

(3) 在 hdfs-site.xml 文件中配置多目录，注意新挂载磁盘的访问权限问题

HDFS 的 DataNode 节点保存数据的路径由 `dfs.datanode.data.dir` 参数决定，其默认值为 `file://${hadoop.tmp.dir}/dfs/data`，若服务器有多个磁盘，必须对该参数进行修改。如服务器磁盘如上图所示，则该参数应修改为如下的值。

```
<property>
  <name>dfs.datanode.data.dir</name>
  <value>file:///dfs/data1,file:///hd2/dfs/data2,file:///hd3/dfs/data3,file:///hd4/dfs/data4</value>
</property>
```

注意：因为每台服务器节点的磁盘情况不同，所以这个配置配完之后，不需要分发

## 4.2.2 集群数据均衡

### 1) 节点间数据均衡

(1) 开启数据均衡命令

```
start-balancer.sh -threshold 10
```

对于参数 10，代表的是集群中各个节点的磁盘空间利用率相差不超过 10%，可根据实际情况进行调整。

(2) 停止数据均衡命令

```
stop-balancer.sh
```

注意：于 HDFS 需要启动单独的 Rebalance Server 来执行 Rebalance 操作，所以尽量不要在 NameNode 上执行 `start-balancer.sh`，而是找一台比较空闲的机器。

更多 Java - 大数据 - 前端 - python 人工智能资料下载，可百度访问：尚硅谷官网



## 2) 磁盘间数据均衡

(1) 生成均衡计划 (我们只有一块磁盘, 不会生成计划)

```
hdfs diskbalancer -plan hadoop103
```

(2) 执行均衡计划

```
hdfs diskbalancer -execute hadoop103.plan.json
```

(3) 查看当前均衡任务的执行情况

```
hdfs diskbalancer -query hadoop103
```

(4) 取消均衡任务

```
hdfs diskbalancer -cancel hadoop103.plan.json
```

## 4.2.3 项目经验之支持 LZO 压缩配置

### 1) hadoop-lzo 编译

hadoop 本身并不支持 lzo 压缩, 故需要使用 twitter 提供的 hadoop-lzo 开源组件。hadoop-lzo 需依赖 hadoop 和 lzo 进行编译, 编译步骤如下。



hadoop-lzo编译.txt

### 2) 将编译好后的 hadoop-lzo-0.4.20.jar 放入 hadoop-3.1.3/share/hadoop/common/

```
[atguigu@hadoop102 common]$ pwd
/opt/module/hadoop-3.1.3/share/hadoop/common
[atguigu@hadoop102 common]$ ls
hadoop-lzo-0.4.20.jar
```

### 3) 同步 hadoop-lzo-0.4.20.jar 到 hadoop103、hadoop104

```
[atguigu@hadoop102 common]$ xsync hadoop-lzo-0.4.20.jar
```

### 4) core-site.xml 增加配置支持 LZO 压缩

```
<configuration>
  <property>
    <name>io.compression.codecs</name>
    <value>
      org.apache.hadoop.io.compress.GzipCodec,
      org.apache.hadoop.io.compress.DefaultCodec,
      org.apache.hadoop.io.compress.BZip2Codec,
      org.apache.hadoop.io.compress.SnappyCodec,
      com.hadoop.compression.lzo.LzoCodec,
      com.hadoop.compression.lzo.LzopCodec
    </value>
  </property>

  <property>
    <name>io.compression.codec.lzo.class</name>
    <value>com.hadoop.compression.lzo.LzoCodec</value>
  </property>
</configuration>
```

### 5) 同步 core-site.xml 到 hadoop103、hadoop104

```
[atguigu@hadoop102 hadoop]$ xsync core-site.xml
```

更多 Java - 大数据 - 前端 - python 人工智能资料下载, 可百度访问: 尚硅谷官网

## 6) 启动及查看集群

```
[atguigu@hadoop102 hadoop-3.1.3]$ sbin/start-dfs.sh
[atguigu@hadoop103 hadoop-3.1.3]$ sbin/start-yarn.sh
```

## 7) 测试-数据准备

```
[atguigu@hadoop102 hadoop-3.1.3]$ hadoop fs -mkdir /input
[atguigu@hadoop102 hadoop-3.1.3]$ hadoop fs -put README.txt /input
```

## 8) 测试-压缩

```
[atguigu@hadoop102 hadoop-3.1.3]$ hadoop jar
share/hadoop/mapreduce/hadoop-mapreduce-examples-3.1.3.jar
wordcount -Dmapreduce.output.fileoutputformat.compress=true -
Dmapreduce.output.fileoutputformat.compress.codec=com.hadoop.c
ompression.lzo.LzopCodec /input /output
```

## 4.2.4 项目经验之 LZO 创建索引

### 1) 创建 LZO 文件的索引

LZO 压缩文件的可切片特性依赖于其索引,故我们需要手动为 LZO 压缩文件创建索引。

若无索引,则 LZO 文件的切片只有一个。

```
hadoop jar /path/to/your/hadoop-lzo.jar
com.hadoop.compression.lzo.DistributedLzoIndexer big_file.lzo
```

### 2) 测试

(1) 将 bigtable.lzo (200M) 上传到集群的根目录

```
[atguigu@hadoop102 module]$ hadoop fs -mkdir /input
[atguigu@hadoop102 module]$ hadoop fs -put bigtable.lzo /input
```

(2) 执行 wordcount 程序

```
[atguigu@hadoop102 module]$ hadoop jar /opt/module/hadoop-
3.1.3/share/hadoop/mapreduce/hadoop-mapreduce-examples-
3.1.3.jar wordcount -
Dmapreduce.job.inputformat.class=com.hadoop.mapreduce.LzoTextI
nputFormat /input /output1
```

```
[atguigu@hadoop102 hadoop-3.1.3]$ hadoop jar /opt/module/hadoop-3.1.3/share/hadoop/mapreduce/had
oop-mapreduce-examples-3.1.3.jar wordcount /input /output1
2020-04-16 13:30:03,182 INFO client.RMProxy: Connecting to ResourceManager at hadoop103/192.168.1.
103:8032
2020-04-16 13:30:05,334 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tm
p/hadoop-yarn/staging/atguigu/.staging/job_1587014829855_0001
2020-04-16 13:30:05,725 INFO sas1.SaslDataTransferClient: SASL encryption trust check: localHostTr
usted = false, remoteHostTrusted = false
2020-04-16 13:30:06,595 INFO input.FileInputFormat: Total input files to process : 1
2020-04-16 13:30:06,640 INFO lzo.GPLNativeCodeLoader: Loaded native gpl library from the embedded
binaries
2020-04-16 13:30:06,650 INFO lzo.LzoCodec: Successfully loaded & initialized native-lzo library [h
adoop-lzo rev 52decc77982b58949890770d22720a91adce0c3f]
2020-04-16 13:30:06,757 INFO sas1.SaslDataTransferClient: SASL encryption trust check: localHostTr
usted = false, remoteHostTrusted = false
2020-04-16 13:30:07,061 INFO sas1.SaslDataTransferClient: SASL encryption trust check: localHostTr
usted = false, remoteHostTrusted = false
2020-04-16 13:30:07,278 INFO mapreduce.JobSubmitter: number of splits:1
```

(3) 对上传的 LZO 文件建索引

```
[atguigu@hadoop102 module]$ hadoop jar /opt/module/hadoop-
3.1.3/share/hadoop/common/hadoop-lzo-0.4.20.jar
com.hadoop.compression.lzo.DistributedLzoIndexer
/input/bigtable.lzo
```

(4) 再次执行 WordCount 程序

```
[atguigu@hadoop102 module]$ hadoop jar /opt/module/hadoop-3.1.3/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.1.3.jar wordcount -Dmapreduce.job.inputformat.class=com.hadoop.mapreduce.LzoTextInputFormat /input /output2
```

```
[atguigu@hadoop102 hadoop-3.1.3]$ hadoop jar /opt/module/hadoop-3.1.3/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.1.3.jar wordcount /input /output2
2020-04-16 13:35:33,378 INFO client.RMPProxy: Connecting to ResourceManager at hadoop103/192.168.1.103:8032
2020-04-16 13:35:36,290 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/atguigu/.staging/job_1587014829855_0003
2020-04-16 13:35:36,626 INFO sas1.Sas1DataTransferClient: SASL encryption trust check: localhostTrusted = false, remoteHostTrusted = false
2020-04-16 13:35:37,427 INFO input.FileInputFormat: Total input files to process : 2
2020-04-16 13:35:37,468 INFO lzo.GPLNativeCodeLoader: Loaded native gpl library from the embedded binaries
2020-04-16 13:35:37,477 INFO lzo.LzoCodec: Successfully loaded & initialized native-lzo library [hadoop-lzo-rev 52decc77982b58949890770d22720a91adce0c3f]
2020-04-16 13:35:37,590 INFO sas1.Sas1DataTransferClient: SASL encryption trust check: localhostTrusted = false, remoteHostTrusted = false
2020-04-16 13:35:37,764 INFO sas1.Sas1DataTransferClient: SASL encryption trust check: localhostTrusted = false, remoteHostTrusted = false
2020-04-16 13:35:37,906 INFO mapreduce.JobSubmitter: number of splits:2
```

### 3) 注意：如果以上任务，在运行过程中报如下异常

```
Container [pid=8468,containerID=container_1594198338753_0001_01_000002] is running 318740992B beyond the 'VIRTUAL' memory limit. Current usage: 111.5 MB of 1 GB physical memory used; 2.4 GB of 2.1 GB virtual memory used. Killing container.
Dump of the process-tree for container_1594198338753_0001_01_000002 :
```

解决办法：在 hadoop102 的/opt/module/hadoop-3.1.3/etc/hadoop/yarn-site.xml 文件中增加

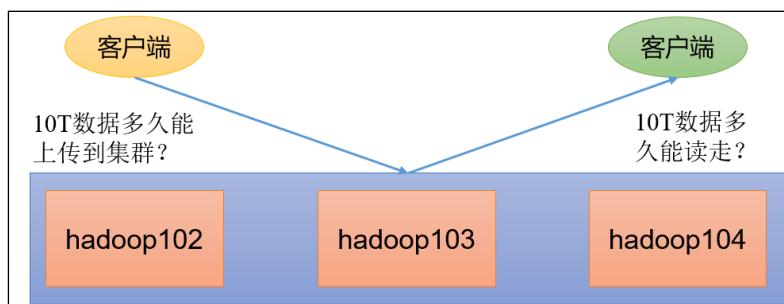
如下配置，然后分发到 hadoop103、hadoop104 服务器上，并重新启动集群。

```
<!--是否启动一个线程检查每个任务正使用的虚拟内存量，如果任务超出分配值，则直接将其杀掉，默认是 true -->
<property>
  <name>yarn.nodemanager.vmem-check-enabled</name>
  <value>>false</value>
</property>
```

## 4.2.5 项目经验之基准测试

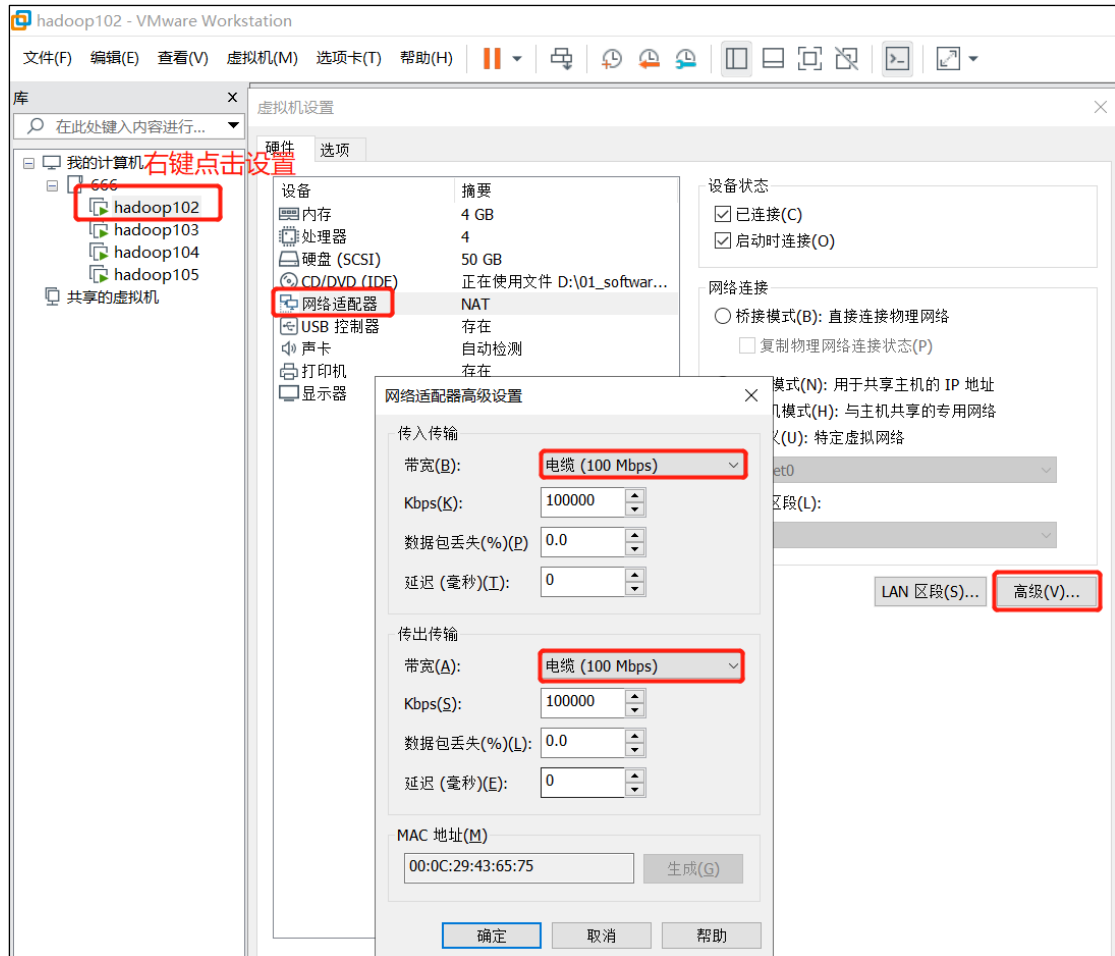
在企业中非常关心每天从 Java 后台拉取过来的数据，需要多久能上传到集群？消费者关心多久能从 HDFS 上拉取需要的数据？

为了搞清楚 HDFS 的读写性能，生产环境上非常需要对集群进行压测。



HDFS 的读写性能主要受网络和磁盘影响比较大。为了方便测试，将 hadoop102、hadoop103、hadoop104 虚拟机网络都设置为 100mbps。

100Mbps 单位是 bit；10M/s 单位是 byte；1byte=8bit，100Mbps/8=12.5M/s。



测试网速：

(1) 来到 hadoop102 的 /opt/module 目录，创建一个

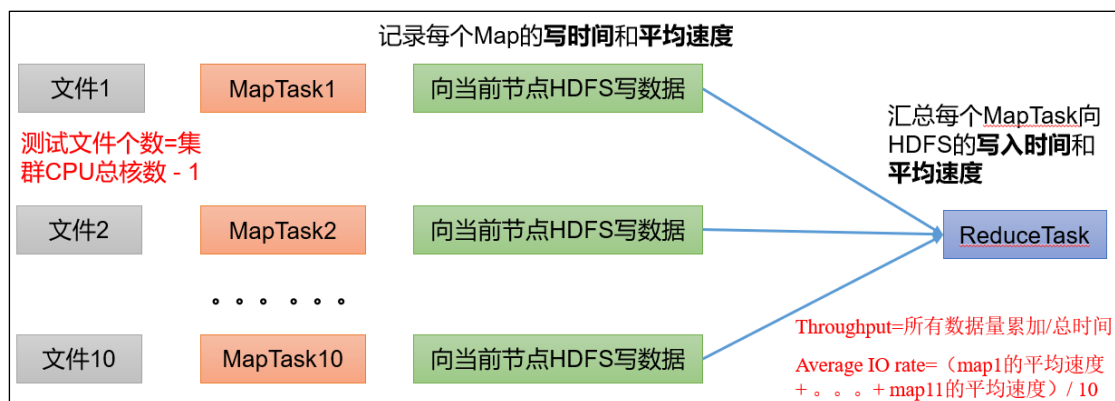
```
[atguigu@hadoop102 software]$ python -m SimpleHTTPServer
```

(2) 在 Web 页面上访问

hadoop102:8000

## 1) 测试 HDFS 写性能

(1) 写测试底层原理



(2) 测试内容：向 HDFS 集群写 10 个 128M 的文件

```
[atguigu@hadoop102 mapreduce]$ hadoop jar /opt/module/hadoop-3.1.3/share/hadoop/mapreduce/hadoop-mapreduce-client-jobclient-3.1.3-tests.jar TestDFSIO -write -nrFiles 10 -fileSize 128MB

2021-02-09 10:43:16,853 INFO fs.TestDFSIO: ----- TestDFSIO ----- : write
2021-02-09 10:43:16,854 INFO fs.TestDFSIO: Date & time: Tue Feb 09 10:43:16 CST 2021
2021-02-09 10:43:16,854 INFO fs.TestDFSIO: Number of files: 10
2021-02-09 10:43:16,854 INFO fs.TestDFSIO: Total MBytes processed: 1280
2021-02-09 10:43:16,854 INFO fs.TestDFSIO: Throughput mb/sec: 1.61
2021-02-09 10:43:16,854 INFO fs.TestDFSIO: Average IO rate mb/sec: 1.9
2021-02-09 10:43:16,854 INFO fs.TestDFSIO: IO rate std deviation: 0.76
2021-02-09 10:43:16,854 INFO fs.TestDFSIO: Test exec time sec: 133.05
2021-02-09 10:43:16,854 INFO fs.TestDFSIO:
```

注意：nrFiles n 为生成 mapTask 的数量，生产环境一般可通过 hadoop103:8088 查看 CPU

核数，设置为 (CPU 核数 - 1)

- Number of files: 生成 mapTask 数量，一般是集群中 (CPU 核数 - 1)，我们测试虚拟机就按照实际的物理内存-1 分配即可。(目标，让每个节点都参与测试)
- Total MBytes processed: 单个 map 处理的文件大小
- Throughput mb/sec: 单个 mapTak 的吞吐量  
计算方式：处理的总文件大小/每一个 mapTask 写数据的时间累加  
集群整体吞吐量：生成 mapTask 数量\*单个 mapTak 的吞吐量
- Average IO rate mb/sec:: 平均 mapTak 的吞吐量  
计算方式：每个 mapTask 处理文件大小/每一个 mapTask 写数据的时间  
全部相加除以 task 数量
- IO rate std deviation: 方差、反映各个 mapTask 处理的差值，越小越均衡

注意：如果测试过程中，出现异常

①可以在 yarn-site.xml 中设置虚拟内存检测为 false

```
<!--是否启动一个线程检查每个任务正使用的虚拟内存量，如果任务超出分配值，则直接将其杀掉，默认是 true -->
<property>
  <name>yarn.nodemanager.vmem-check-enabled</name>
  <value>>false</value>
</property>
```

②分发配置并重启 Yarn 集群

(3) 测试结果分析

①由于副本 1 就在本地，所以该副本不参与测试



一共参与测试的文件：10 个文件 \* 2 个副本 = 20 个

压测后的速度：1.61

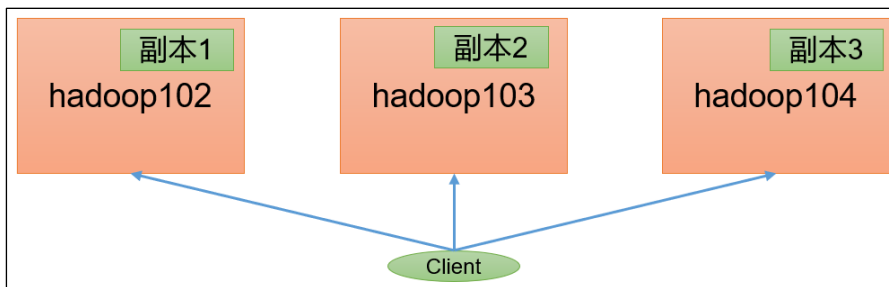
实测速度：1.61M/s \* 20 个文件 ≈ 32M/s

三台服务器的带宽：12.5 + 12.5 + 12.5 ≈ 30m/s

所有网络资源都已经用满。

如果实测速度远远小于网络，并且实测速度不能满足工作要求，可以考虑采用固态硬盘或者增加磁盘个数。

②如果客户端不在集群节点，那就三个副本都参与计算



## 2) 测试 HDFS 读性能

(1) 测试内容：读取 HDFS 集群 10 个 128M 的文件

```
[atguigu@hadoop102 mapreduce]$ hadoop jar /opt/module/hadoop-3.1.3/share/hadoop/mapreduce/hadoop-mapreduce-client-jobclient-3.1.3-tests.jar TestDFSIO -read -nrFiles 10 -fileSize 128MB
```

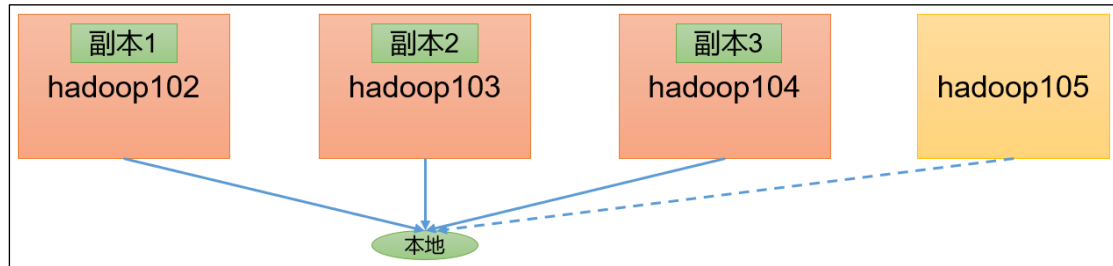
```
2021-02-09 11:34:15,847 INFO fs.TestDFSIO: ----- TestDFSIO ----- : read
2021-02-09 11:34:15,847 INFO fs.TestDFSIO: Date & time: Tue Feb 09 11:34:15 CST 2021
2021-02-09 11:34:15,847 INFO fs.TestDFSIO: Number of files: 10
2021-02-09 11:34:15,847 INFO fs.TestDFSIO: Total MBytes processed: 1280
2021-02-09 11:34:15,848 INFO fs.TestDFSIO: Throughput mb/sec: 200.28
2021-02-09 11:34:15,848 INFO fs.TestDFSIO: Average IO rate mb/sec: 266.74
2021-02-09 11:34:15,848 INFO fs.TestDFSIO: IO rate std deviation: 143.12
2021-02-09 11:34:15,848 INFO fs.TestDFSIO: Test exec time sec: 20.83
```

(2) 删除测试生成数据

```
[atguigu@hadoop102 mapreduce]$ hadoop jar /opt/module/hadoop-3.1.3/share/hadoop/mapreduce/hadoop-mapreduce-client-jobclient-3.1.3-tests.jar TestDFSIO -clean
```

(3) 测试结果分析：为什么读取文件速度大于网络带宽？由于目前只有三台服务器，且有三个副本，数据读取就近原则，相当于都是读取的本地磁盘数据，没有走网络。

更多 [Java](#) - [大数据](#) - [前端](#) - [python](#) 人工智能资料下载，可百度访问：尚硅谷官网



### 3) 使用 Sort 程序评测 MapReduce

(1) 使用 RandomWriter 来产生随机数，每个节点运行 10 个 Map 任务，每个 Map 产生大约 1G 大小的二进制随机数

```
[atguigu@hadoop102 mapreduce]$ hadoop jar /opt/module/hadoop-3.1.3/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.1.3.jar randomwriter random-data
```

(2) 执行 Sort 程序

```
[atguigu@hadoop102 mapreduce]$ hadoop jar /opt/module/hadoop-3.1.3/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.1.3.jar sort random-data sorted-data
```

(3) 验证数据是否真正排好序了

```
[atguigu@hadoop102 mapreduce]$
hadoop jar /opt/module/hadoop-3.1.3/share/hadoop/mapreduce/hadoop-mapreduce-client-jobclient-3.1.3-tests.jar testmapredsort -sortInput random-data -sortOutput sorted-data
```

## 4.2.6 项目经验之 Hadoop 参数调优

### 1) HDFS 参数调优 hdfs-site.xml

The number of Namenode RPC server threads that listen to requests from clients. If dfs.namenode.servicerpc-address is not configured then Namenode RPC server threads listen to requests from all nodes.

NameNode 有一个工作线程池，用来处理不同 DataNode 的并发心跳以及客户端并发的元数据操作。

对于大集群或者有大量客户端的集群来说，通常需要增大参数 dfs.namenode.handler.count 的默认值 10。

```
<property>
  <name>dfs.namenode.handler.count</name>
  <value>10</value>
</property>
```

$dfs.namenode.handler.count = 20 \times \log_e^{Cluster\ Size}$ ，比如集群规模为 8 台时，此参数设置为

41。可通过简单的 python 代码计算该值，代码如下。

```
[atguigu@hadoop102 ~]$ python
Python 2.7.5 (default, Apr 11 2018, 07:36:10)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-28)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import math
>>> print int(20*math.log(8))
```

更多 Java - 大数据 - 前端 - python 人工智能资料下载，可百度访问：尚硅谷官网



```
41
>>> quit()
```

## 2) YARN 参数调优 yarn-site.xml

(1) 情景描述: 总共 7 台机器, 每天几亿条数据, 数据源->Flume->Kafka->HDFS->Hive  
面临问题: 数据统计主要用 HiveSQL, 没有数据倾斜, 小文件已经做了合并处理, 开启的 JVM 重用, 而且 IO 没有阻塞, **内存用了不到 50%**。但是还是跑的非常慢, 而且数据量洪峰过来时, 整个集群都会宕掉。基于这种情况有没有优化方案。

(2) 解决办法:

NodeManager 内存和服务器实际内存配置尽量接近, 如服务器有 128g 内存, 但是 NodeManager 默认内存 8G, 不修改该参数最多只能用 8G 内存。NodeManager 使用的 CPU 核数和服务器 CPU 核数尽量接近。

①yarn.nodemanager.resource.memory-mb NodeManager 使用内存数

②yarn.nodemanager.resource.cpu-vcores NodeManager 使用 CPU 核数

## 4.3 Zookeeper 安装

### 4.3.1 安装 ZK

详见: 尚硅谷大数据技术之 Zookeeper



尚硅谷大数据技术  
之Zookeeper (V3)

集群规划

	服务器 hadoop102	服务器 hadoop103	服务器 hadoop104
Zookeeper	Zookeeper	Zookeeper	Zookeeper

### 4.3.2 ZK 集群启动停止脚本

(1) 在 hadoop102 的/home/atguigu/bin 目录下创建脚本

```
[atguigu@hadoop102 bin]$ vim zk.sh
```

在脚本中编写如下内容

```
#!/bin/bash

case $1 in
"start"){
    for i in hadoop102 hadoop103 hadoop104
    do
        echo ----- zookeeper $i 启动 -----
        ssh $i "/opt/module/zookeeper-3.5.7/bin/zkServer.sh
```

更多 [Java](#) - [大数据](#) - [前端](#) - [python](#) 人工智能资料下载, 可百度访问: [尚硅谷官网](#)



```
start"
done
};;
"stop"){
for i in hadoop102 hadoop103 hadoop104
do
echo ----- zookeeper $i 停止 -----
ssh $i "/opt/module/zookeeper-3.5.7/bin/zkServer.sh
stop"
done
};;
"status"){
for i in hadoop102 hadoop103 hadoop104
do
echo ----- zookeeper $i 状态 -----
ssh $i "/opt/module/zookeeper-3.5.7/bin/zkServer.sh
status"
done
};;
esac
```

(2) 增加脚本执行权限

```
[atguigu@hadoop102 bin]$ chmod u+x zk.sh
```

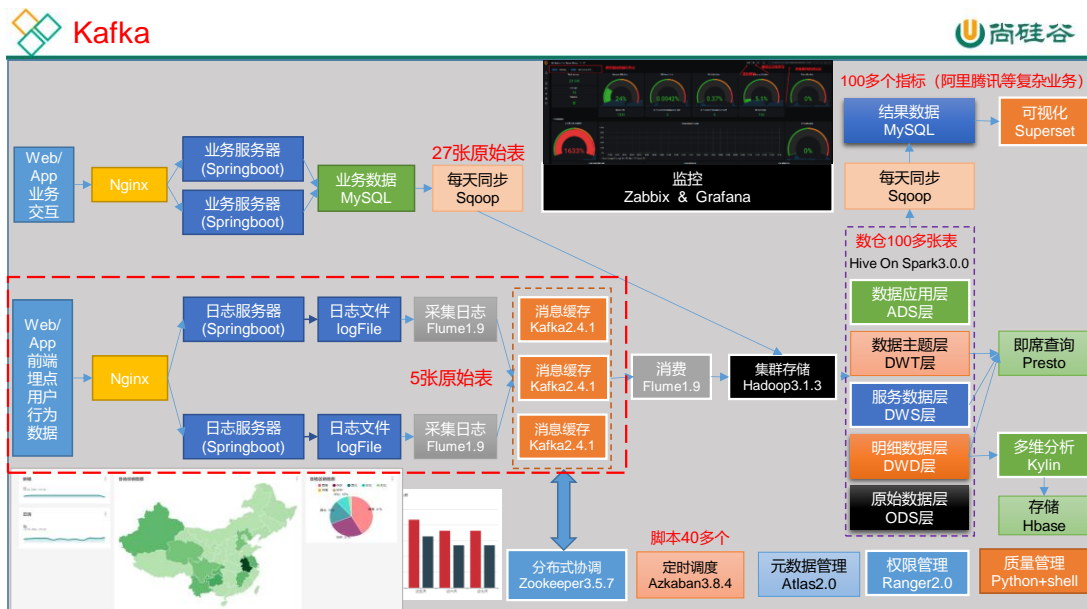
(3) Zookeeper 集群启动脚本

```
[atguigu@hadoop102 module]$ zk.sh start
```

(4) Zookeeper 集群停止脚本

```
[atguigu@hadoop102 module]$ zk.sh stop
```

## 4.4 Kafka 安装



### 4.4.1 Kafka 集群安装

详见：尚硅谷大数据技术之 Kafka



## 尚硅谷大数据技术 之Kafka (V3.0) .c

集群规划:

	服务器 hadoop102	服务器 hadoop103	服务器 hadoop104
Kafka	Kafka	Kafka	Kafka

### 4.4.2 Kafka 集群启动停止脚本

(1) 在/home/atguigu/bin 目录下创建脚本 kf.sh

```
[atguigu@hadoop102 bin]$ vim kf.sh
```

在脚本中填写如下内容

```
#!/bin/bash

case $1 in
"start"){
    for i in hadoop102 hadoop103 hadoop104
    do
        echo " -----启动 $i Kafka-----"
        ssh $i "/opt/module/kafka/bin/kafka-server-start.sh -
daemon /opt/module/kafka/config/server.properties"
    done
};;
"stop"){
    for i in hadoop102 hadoop103 hadoop104
    do
        echo " -----停止 $i Kafka-----"
        ssh $i "/opt/module/kafka/bin/kafka-server-stop.sh stop"
    done
};;
esac
```

(2) 增加脚本执行权限

```
[atguigu@hadoop102 bin]$ chmod u+x kf.sh
```

(3) kf 集群启动脚本

```
[atguigu@hadoop102 module]$ kf.sh start
```

(4) kf 集群停止脚本

```
[atguigu@hadoop102 module]$ kf.sh stop
```

### 4.4.3 Kafka 常用命令

#### 1) 查看 Kafka Topic 列表

```
[atguigu@hadoop102 kafka]$ bin/kafka-topics.sh --zookeeper
hadoop102:2181/kafka --list
```

#### 2) 创建 Kafka Topic

进入到/opt/module/kafka/目录下创建日志主题

```
[atguigu@hadoop102 kafka]$ bin/kafka-topics.sh --zookeeper
hadoop102:2181,hadoop103:2181,hadoop104:2181/kafka --create --
replication-factor 1 --partitions 1 --topic topic_log
```

### 3) 删除 Kafka Topic

```
[atguigu@hadoop102 kafka]$ bin/kafka-topics.sh --delete --
zookeeper hadoop102:2181,hadoop103:2181,hadoop104:2181/kafka -
-topic topic_log
```

### 4) Kafka 生产消息

```
[atguigu@hadoop102 kafka]$ bin/kafka-console-producer.sh \
--broker-list hadoop102:9092 --topic topic_log
>hello world
>atguigu atguigu
```

### 5) Kafka 消费消息

```
[atguigu@hadoop102 kafka]$ bin/kafka-console-consumer.sh \
--bootstrap-server hadoop102:9092 --from-beginning --topic
topic_log
```

--from-beginning: 会把主题中以往所有的数据都读取出来。根据业务场景选择是否增加该配置。

### 6) 查看 Kafka Topic 详情

```
[atguigu@hadoop102 kafka]$ bin/kafka-topics.sh --zookeeper
hadoop102:2181/kafka \
--describe --topic topic_log
```

## 4.4.4 项目经验之 Kafka 机器数量计算

Kafka 机器数量 (经验公式) =  $2 * (\text{峰值生产速度} * \text{副本数} / 100) + 1$

先拿到峰值生产速度，再根据设定的副本数，就能预估出需要部署 Kafka 的数量。

#### 1) 峰值生产速度

峰值生产速度可以压测得到。

#### 2) 副本数

副本数默认是 1 个，在企业里面 2-3 个都有，2 个居多。

副本多可以提高可靠性，但是会降低网络传输效率。

比如我们的峰值生产速度是 50M/s。副本数为 2。

Kafka 机器数量 =  $2 * (50 * 2 / 100) + 1 = 3$  台

## 4.4.5 项目经验之 Kafka 压力测试

### 1) Kafka 压测

用 Kafka 官方自带的脚本，对 Kafka 进行压测。

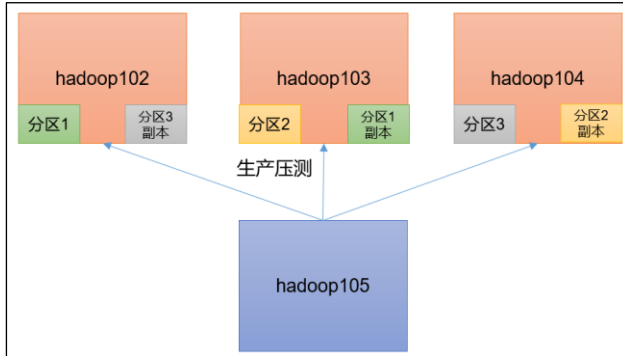
kafka-consumer-perf-test.sh

kafka-producer-perf-test.sh

更多 [Java](#) - [大数据](#) - [前端](#) - [python](#) 人工智能资料下载，可百度访问：尚硅谷官网

Kafka 压测时,在硬盘读写速度一定的情况下,可以查看到哪些地方出现了瓶颈(CPU,内存,网络 IO)。一般都是网络 IO 达到瓶颈。

## 2) Kafka Producer 压力测试



### (0) 压测环境准备

- ①hadoop102、hadoop103、hadoop104 的网络带宽都设置为 100mbps。
- ②关闭 hadoop102 主机,并根据 hadoop102 克隆出 hadoop105 (修改 IP 和主机名称)

- ③hadoop105 的带宽不限

- ④创建一个 test topic, 设置为 3 个分区 2 个副本

```
[atguigu@hadoop102 kafka]$ bin/kafka-topics.sh --zookeeper hadoop102:2181,hadoop103:2181,hadoop104:2181/kafka --create --replication-factor 2 --partitions 3 --topic test
```

### (1) 在/opt/module/kafka/bin 目录下面有这两个文件。我们来测试一下

```
[atguigu@hadoop102 kafka]$ bin/kafka-producer-perf-test.sh --topic test --record-size 100 --num-records 10000000 --throughput -1 --producer-props bootstrap.servers=hadoop102:9092,hadoop103:9092,hadoop104:9092
```

说明:

record-size 是一条信息有多大,单位是字节。

num-records 是总共发送多少条信息。

throughput 是每秒多少条信息,设成-1,表示不限流,尽可能快的生产数据,可测出生产者最大吞吐量。

### (2) Kafka 会打印下面的信息

```
699884 records sent, 139976.8 records/sec (13.35 MB/sec), 1345.6 ms avg latency, 2210.0 ms max latency.
713247 records sent, 141545.3 records/sec (13.50 MB/sec), 1577.4 ms avg latency, 3596.0 ms max latency.
773619 records sent, 153862.2 records/sec (14.67 MB/sec), 2326.8 ms avg latency, 4051.0 ms max latency.
773961 records sent, 154206.2 records/sec (15.71 MB/sec), 1964.1 ms avg latency, 2917.0 ms max latency.
```

更多 Java -大数据 -前端 -python 人工智能资料下载,可百度访问:尚硅谷官网

```
776970 records sent, 154559.4 records/sec (15.74 MB/sec), 1960.2 ms avg
latency, 2922.0 ms max latency.
776421 records sent, 154727.2 records/sec (15.76 MB/sec), 1960.4 ms avg
latency, 2954.0 ms max latency.
```

参数解析：Kafka 的吞吐量 15m/s 左右是否符合预期呢？

hadoop102、hadoop103、hadoop104 三台集群的网络总带宽 30m/s 左右，由于是两个副本，所以 Kafka 的吞吐量  $30\text{m/s} \div 2$ （副本） = 15m/s

结论：网络带宽和副本都会影响吞吐量。

#### （4）调整 batch.size

batch.size 默认值是 16k。

batch.size 较小，会降低吞吐量。比如说，批次大小为 0 则完全禁用批处理，会一条一条发送消息）；

batch.size 过大，会增加消息发送延迟。比如说，Batch 设置为 64k，但是要等待 5 秒钟 Batch 才凑满了 64k，才能发送出去。那这条消息的延迟就是 5 秒钟。

```
[atguigu@hadoop102 kafka]$ bin/kafka-producer-perf-test.sh --
topic test --record-size 100 --num-records 10000000 --throughput
-1 --producer-props
bootstrap.servers=hadoop102:9092,hadoop103:9092,hadoop104:9092
batch.size=500
```

输出结果

```
69169 records sent, 13833.8 records/sec (1.32 MB/sec), 2517.6 ms avg
latency, 4299.0 ms max latency.
105372 records sent, 21074.4 records/sec (2.01 MB/sec), 6748.4 ms avg
latency, 9016.0 ms max latency.
113188 records sent, 22637.6 records/sec (2.16 MB/sec), 11348.0 ms avg
latency, 13196.0 ms max latency.
108896 records sent, 21779.2 records/sec (2.08 MB/sec), 12272.6 ms avg
latency, 12870.0 ms max latency.
```

#### （5）linger.ms

如果设置 batch size 为 64k，但是比如过了 10 分钟也没有凑够 64k，怎么办？

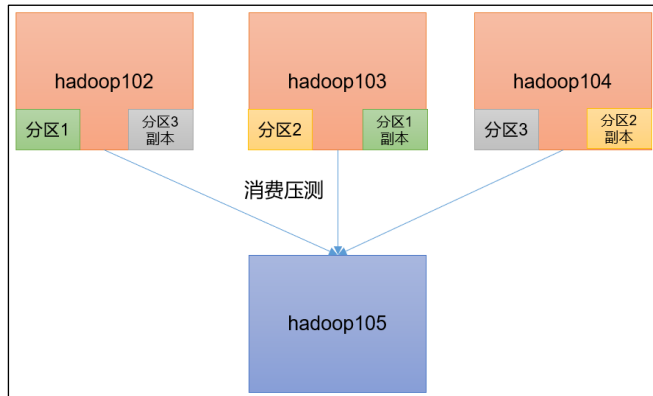
可以设置，linger.ms。比如 linger.ms=5ms，那么就是要发送的数据没有到 64k，5ms 后，数据也会发出去。

#### （6）总结

同时设置 batch.size 和 linger.ms，就是哪个条件先满足就都会将消息发送出去

Kafka 需要考虑高吞吐量与延时的平衡。

### 3) Kafka Consumer 压力测试



(1) Consumer 的测试，如果这四个指标（IO，CPU，内存，网络）都不能改变，考虑增加分区数来提升性能。

```
[atguigu@hadoop102 kafka]$ bin/kafka-consumer-perf-test.sh --
broker-list hadoop102:9092,hadoop103:9092,hadoop104:9092 --
topic test --fetch-size 10000 --messages 10000000 --threads 1
```

①参数说明：

--broker-list 指定 Kafka 集群地址

--topic 指定 topic 的名称

--fetch-size 指定每次 fetch 的数据的大小

--messages 总共要消费的消息个数

②测试结果说明：

start.time, end.time, data.consumed.in.MB, **MB.sec**, data.consumed.in.nMsg, **nMsg.sec**

2021-08-03 21:17:21:778, 2021-08-03 21:18:19:775, 514.7169, **8.8749**, 5397198, 93059.9514

开始测试时间，测试结束数据，共消费数据 514.7169MB，吞吐量 **8.8749MB/s**

(2) 调整 fetch-size

①增加 fetch-size 值，观察消费吞吐量。

```
[atguigu@hadoop102 kafka]$ bin/kafka-consumer-perf-test.sh --
broker-list hadoop102:9092,hadoop103:9092,hadoop104:9092 --
topic test --fetch-size 100000 --messages 10000000 --threads 1
```

②测试结果说明：

start.time, end.time, data.consumed.in.MB, **MB.sec**, data.consumed.in.nMsg, **nMsg.sec**

2021-08-03 21:22:57:671, 2021-08-03 21:23:41:938, 514.7169, **11.6276**, 5397198, 121923.7355

(3) 总结

吞吐量受网络带宽和 fetch-size 的影响

## 4.4.6 项目经验值 Kafka 分区数计算

- (1) 创建一个只有 1 个分区的 topic
- (2) 测试这个 topic 的 producer 吞吐量和 consumer 吞吐量。
- (3) 假设他们的值分别是  $T_p$  和  $T_c$ ，单位可以是 MB/s。
- (4) 然后假设总的目标吞吐量是  $T_t$ ，那么分区数 =  $T_t / \min(T_p, T_c)$

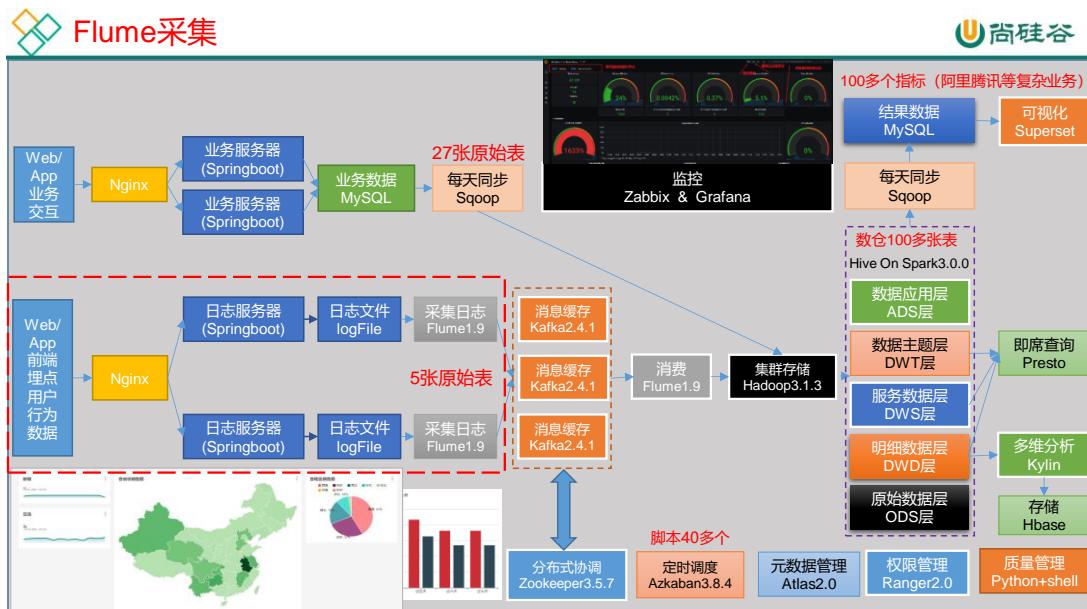
例如：producer 吞吐量 = 20m/s；consumer 吞吐量 = 50m/s，期望吞吐量 100m/s；

分区数 =  $100 / 20 = 5$  分区

[https://blog.csdn.net/weixin\\_42641909/article/details/89294698](https://blog.csdn.net/weixin_42641909/article/details/89294698)

分区数一般设置为：3-10 个

## 4.5 采集日志 Flume



### 4.5.1 日志采集 Flume 安装

详见：尚硅谷大数据技术之 Flume



尚硅谷大数据技术  
之 Flume (V3.0) .c

集群规划：

	服务器 hadoop102	服务器 hadoop103	服务器 hadoop104
Flume(采集日志)	Flume	Flume	

## 4.5.2 项目经验之 Flume 组件选型

### 1) Source

(1) Taildir Source 相比 Exec Source、Spooling Directory Source 的优势

**TailDir Source:** 断点续传、多目录。Flume1.6 以前需要自己自定义 Source 记录每次读取文件位置，实现断点续传。不会丢数据，但是有可能导致数据重复。

**Exec Source** 可以实时搜集数据，但是在 Flume 不运行或者 Shell 命令出错的情况下，数据将会丢失。

**Spooling Directory Source** 监控目录，支持断点续传。

(2) batchSize 大小如何设置？

答：Event 1K 左右时，500-1000 合适（默认为 100）

### 2) Channel

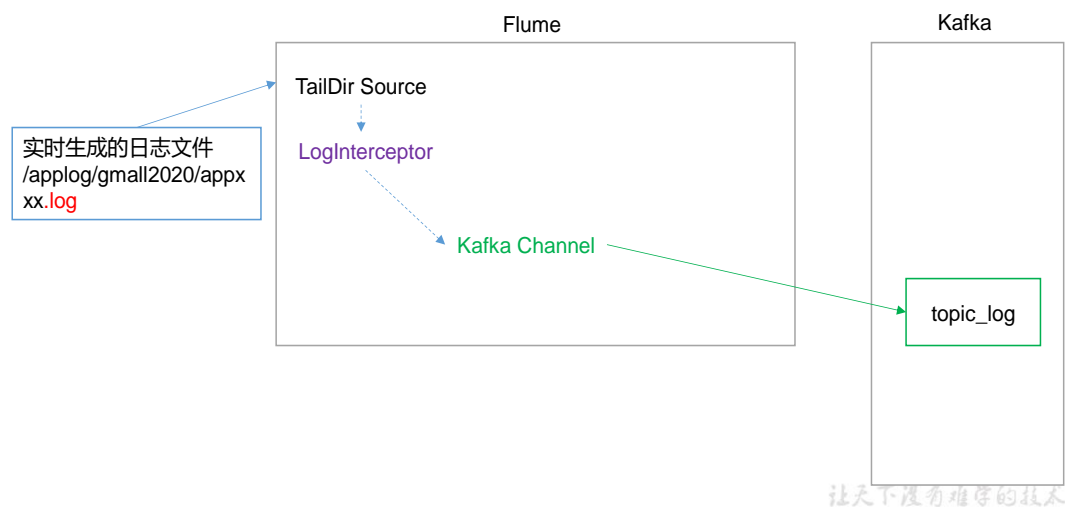
采用 **Kafka Channel**，省去了 Sink，提高了效率。**KafkaChannel** 数据存储在 Kafka 里面，所以数据是存储在磁盘中的。

注意在 Flume1.7 以前，Kafka Channel 很少有人使用，因为发现 **parseAsFlumeEvent** 这个配置起不了作用。也就是无论 **parseAsFlumeEvent** 配置为 true 还是 false，都会转为 Flume Event。这样的话，造成的结果是，会始终都把 Flume 的 headers 中的信息混合着内容一起写入 Kafka 的消息中，这显然不是我所需要的，我只是需要把内容写入即可。

## 4.5.3 日志采集 Flume 配置

### 1) Flume 配置分析





Flume 直接读 log 日志的数据，log 日志的格式是 app.yyyy-mm-dd.log。

## 2) Flume 的具体配置如下：

(1) 在/opt/module/flume/conf 目录下创建 file-flume-kafka.conf 文件

```
[atguigu@hadoop102 conf]$ vim file-flume-kafka.conf
```

在文件配置如下内容

```
#为各组件命名
a1.sources = r1
a1.channels = c1

#描述 source
a1.sources.r1.type = TAILDIR
a1.sources.r1.filegroups = f1
a1.sources.r1.filegroups.f1 = /opt/module/applog/log/app.*
a1.sources.r1.positionFile = /opt/module/flume/tailedir_position.json
a1.sources.r1.interceptors = i1
a1.sources.r1.interceptors.i1.type = com.atguigu.flume.interceptor.ETLInterceptor$Builder

#描述 channel
a1.channels.c1.type = org.apache.flume.channel.kafka.KafkaChannel
a1.channels.c1.kafka.bootstrap.servers = hadoop102:9092,hadoop103:9092
a1.channels.c1.kafka.topic = topic_log
a1.channels.c1.parseAsFlumeEvent = false

#绑定 source 和 channel 以及 sink 和 channel 的关系
a1.sources.r1.channels = c1
```

注意：com.atguigu.flume.interceptor.ETLInterceptor 是自定义的拦截器的全类名。需要根据用户自定义的拦截器做相应修改。

#### 4.5.4 Flume 拦截器

- 1) 创建 Maven 工程 flume-interceptor
- 2) 创建包名: com.atguigu.flume.interceptor
- 3) 在 pom.xml 文件中添加如下配置

```
<dependencies>
  <dependency>
    <groupId>org.apache.flume</groupId>
    <artifactId>flume-ng-core</artifactId>
    <version>1.9.0</version>
    <scope>provided</scope>
  </dependency>

  <dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>fastjson</artifactId>
    <version>1.2.62</version>
  </dependency>
</dependencies>

<build>
  <plugins>
    <plugin>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>2.3.2</version>
      <configuration>
        <source>1.8</source>
        <target>1.8</target>
      </configuration>
    </plugin>
    <plugin>
      <artifactId>maven-assembly-plugin</artifactId>
      <configuration>
        <descriptorRefs>
          <descriptorRef>jar-with-
dependencies</descriptorRef>
        </descriptorRefs>
      </configuration>
      <executions>
        <execution>
          <id>make-assembly</id>
          <phase>package</phase>
          <goals>
            <goal>single</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
```

注意: scope 中 **provided** 的含义是编译时用该 jar 包。打包时不用。因为集群上已经存在 flume 的 jar 包。只是本地编译时用一下。

## 4) 在 com.atguigu.flume.interceptor 包下创建 JSONUtils 类

```
package com.atguigu.flume.interceptor;

import com.alibaba.fastjson.JSON;
import com.alibaba.fastjson.JSONException;

public class JSONUtils {
    public static boolean isJSONValidate(String log){
        try {
            JSON.parse(log);
            return true;
        } catch (JSONException e) {
            return false;
        }
    }
}
```

## 5) 在 com.atguigu.flume.interceptor 包下创建 LogInterceptor 类

```
package com.atguigu.flume.interceptor;

import com.alibaba.fastjson.JSON;
import org.apache.flume.Context;
import org.apache.flume.Event;
import org.apache.flume.interceptor.Interceptor;

import java.nio.charset.StandardCharsets;
import java.util.Iterator;
import java.util.List;

public class ETLInterceptor implements Interceptor {

    @Override
    public void initialize() {

    }

    @Override
    public Event intercept(Event event) {

        byte[] body = event.getBody();
        String log = new String(body, StandardCharsets.UTF_8);

        if (JSONUtils.isJSONValidate(log)) {
            return event;
        } else {
            return null;
        }
    }

    @Override
    public List<Event> intercept(List<Event> list) {

        Iterator<Event> iterator = list.iterator();

        while (iterator.hasNext()){
            Event next = iterator.next();
            if(intercept(next)==null){

```

```
        iterator.remove();
    }
}

return list;
}

public static class Builder implements Interceptor.Builder{

    @Override
    public Interceptor build() {
        return new ETLInterceptor();
    }
    @Override
    public void configure(Context context) {

    }

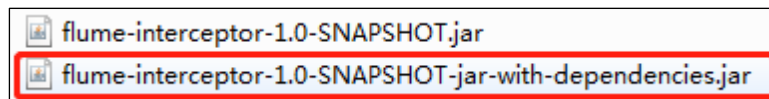
}

@Override
public void close() {

}

}
```

#### 6) 打包



#### 7) 需要先将打好的包放入到 hadoop102 的 /opt/module/flume/lib 文件夹下面。

```
[atguigu@hadoop102 lib]$ ls | grep interceptor
flume-interceptor-1.0-SNAPSHOT-jar-with-dependencies.jar
```

#### 8) 分发 Flume 到 hadoop103、hadoop104

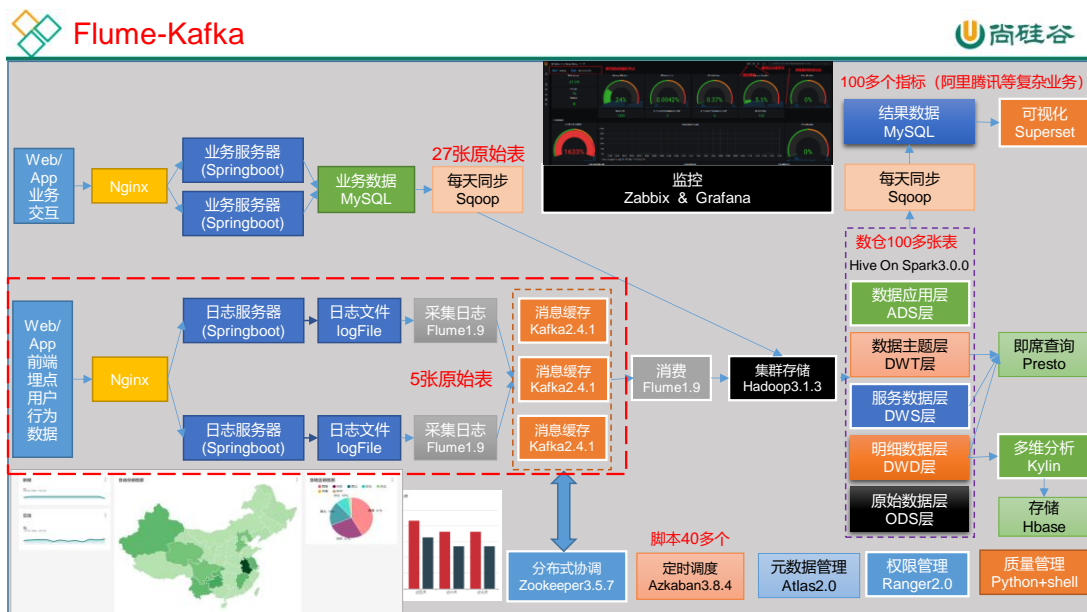
```
[atguigu@hadoop102 module]$ xsync flume/
```

#### 9) 分别在 hadoop102、hadoop103 上启动 Flume

```
[atguigu@hadoop102 flume]$ bin/flume-ng agent --name a1 --conf-
file conf/file-flume-kafka.conf &
```

```
[atguigu@hadoop103 flume]$ bin/flume-ng agent --name a1 --conf-
file conf/file-flume-kafka.conf &
```

### 4.5.5 测试 Flume-Kafka 通道



(1) 生成日志

```
[atguigu@hadoop102 ~]$ lg.sh
```

(2) 消费 Kafka 数据，观察控制台是否有数据获取到

```
[atguigu@hadoop102 kafka]$ bin/kafka-console-consumer.sh \
--bootstrap-server hadoop102:9092 --from-beginning --topic
topic_log
```

说明：如果获取不到数据，先检查 Kafka、Flume、Zookeeper 是否都正确启动。再检查 Flume 的拦截器代码是否正常。

### 4.5.6 日志采集 Flume 启动停止脚本

(1) 在 /home/atguigu/bin 目录下创建脚本 fl.sh

```
[atguigu@hadoop102 bin]$ vim fl.sh
```

在脚本中填写如下内容

```
#!/bin/bash

case $1 in
"start"){
    for i in hadoop102 hadoop103
    do
        echo " -----启动 $i 采集 flume-----"
        ssh $i "nohup /opt/module/flume/bin/flume-ng agent
--conf-file /opt/module/flume/conf/file-flume-kafka.conf --name
a1 -Dflume.root.logger=INFO,LOGFILE >/opt/module/flume/log1.txt
2>&1 &"
    done
};;
"stop"){
    for i in hadoop102 hadoop103
    do
```

```

        echo " -----停止 $i 采集 flume-----"
        ssh $i "ps -ef | grep file-flume-kafka | grep -v
grep |awk '{print \$2}' | xargs -n1 kill -9 "
        done

};;
esac

```

说明 1: nohup, 该命令可以在你退出帐户/关闭终端之后继续运行相应的进程。nohup 就是不挂起的意思, 不挂断地运行命令。

说明 2: awk 默认分隔符为空格

说明 3: \$2 是在 “ ” 双引号内部会被解析为脚本的第二个参数, 但是这里面想表达的含义是 awk 的第二个值, 所以需要将他转义, 用 \\$2 表示。

说明 4: xargs 表示取出前面命令运行的结果, 作为后面命令的输入参数。

(2) 增加脚本执行权限

```
[atguigu@hadoop102 bin]$ chmod u+x fl.sh
```

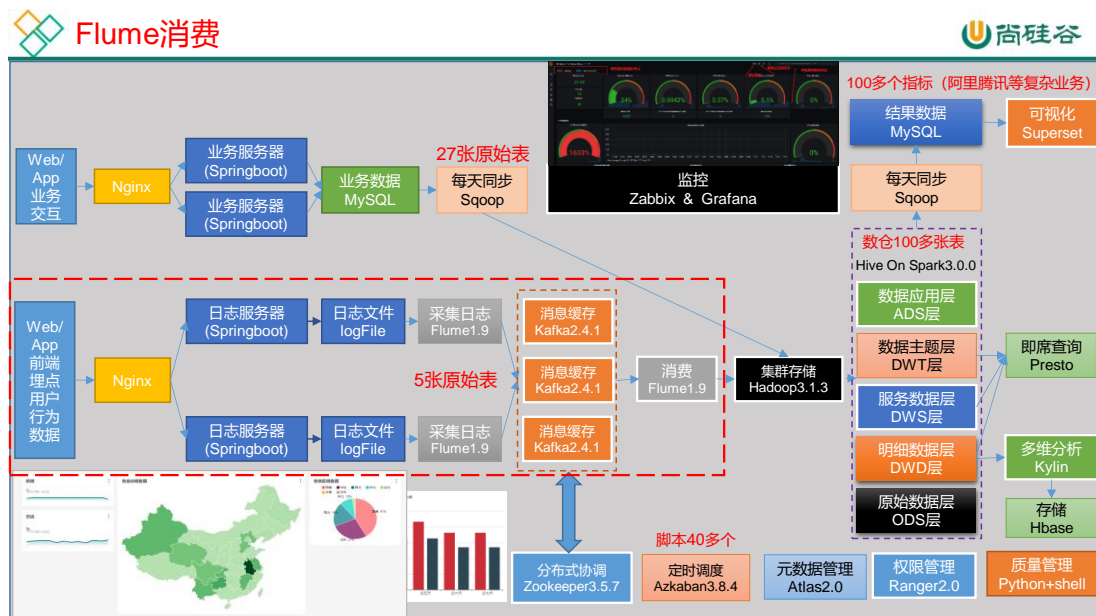
(3) fl 集群启动脚本

```
[atguigu@hadoop102 module]$ fl.sh start
```

(4) fl 集群停止脚本

```
[atguigu@hadoop102 module]$ fl.sh stop
```

## 4.6 消费 Kafka 数据 Flume



集群规划

	服务器 hadoop102	服务器 hadoop103	服务器 hadoop104
Flume (消费 Kafka)			Flume

## 4.6.1 项目经验之 Flume 组件选型

### 1) FileChannel 和 MemoryChannel 区别

MemoryChannel 传输数据速度更快，但因为数据保存在 JVM 的堆内存中，Agent 进程挂掉会导致数据丢失，适用于对数据质量要求不高的需求。

FileChannel 传输速度相对于 Memory 慢，但数据安全保障高，Agent 进程挂掉也可以从失败中恢复数据。

选型：

金融类公司、对钱要求非常准确的公司通常会选择 FileChannel

传输的是普通日志信息（京东内部一天丢 100 万-200 万条，这是非常正常的），通常选择 MemoryChannel。

### 2) FileChannel 优化

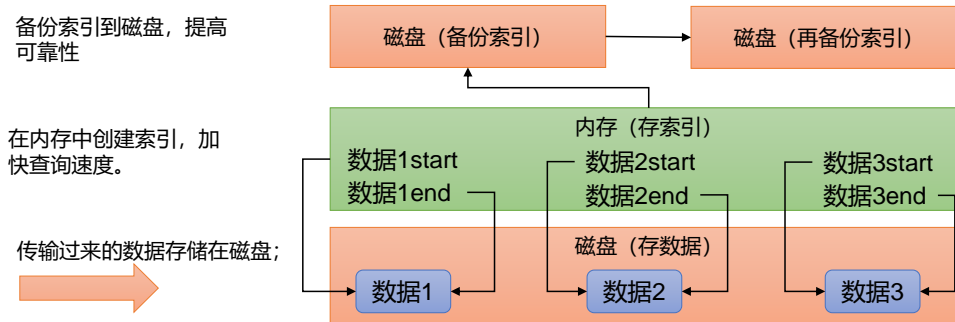
通过配置 **dataDirs** 指向多个路径，每个路径对应不同的硬盘，增大 Flume 吞吐量。

官方说明如下：

```
Comma separated list of directories for storing log files. Using multiple directories on separate disks can improve file channel performance
```

**checkpointDir** 和 **backupCheckpointDir** 也尽量配置在不同硬盘对应的目录中，保证 checkpoint 坏掉后，可以快速使用 backupCheckpointDir 恢复数据。

FileChannel 底层原理



让天下没有难学的技术

### 3) Sink: HDFS Sink

(1) HDFS 存入大量小文件，有什么影响？

**元数据层面：**每个小文件都有一份元数据，其中包括文件路径，文件名，所有者，所属更多 [Java - 大数据 - 前端 - python 人工智能资料下载](#)，可百度访问：[尚硅谷官网](#)

组，权限，创建时间等，这些信息都保存在 Namenode 内存中。所以小文件过多，会占用 Namenode 服务器大量内存，影响 Namenode 性能和使用寿命

**计算层面：**默认情况下 MR 会对每个小文件启用一个 Map 任务计算，非常影响计算性能。同时也影响磁盘寻址时间。

## （2）HDFS 小文件处理

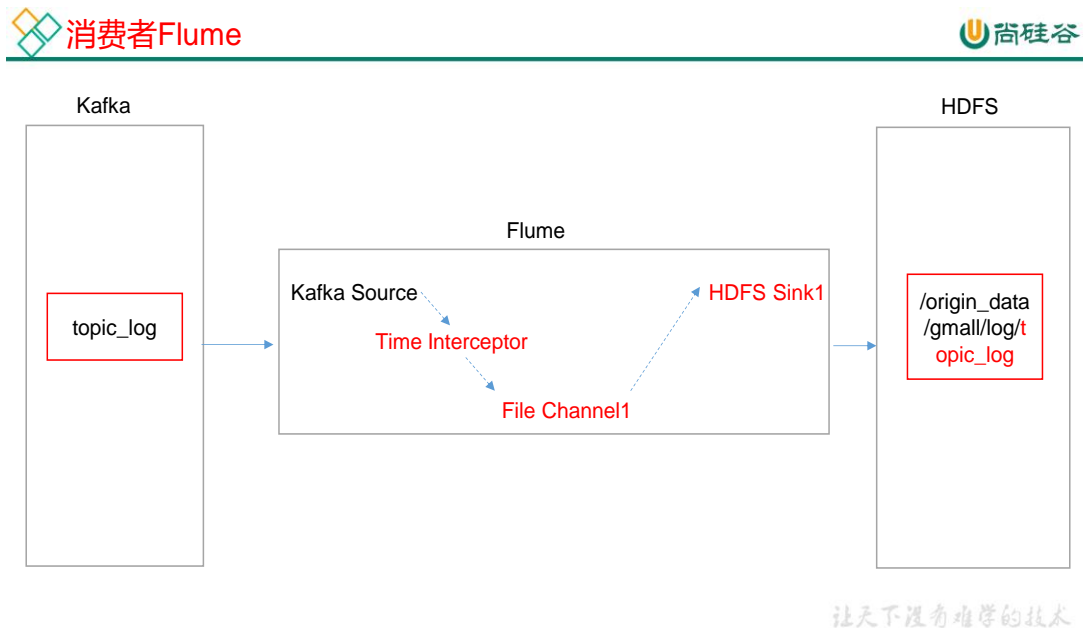
官方默认的这三个参数配置写入 HDFS 后会产生小文件，`hdfs.rollInterval`、`hdfs.rollSize`、`hdfs.rollCount`

基于以上 `hdfs.rollInterval=3600`，`hdfs.rollSize=134217728`，`hdfs.rollCount=0` 几个参数综合作用，效果如下：

- ①文件在达到 128M 时会滚动生成新文件
- ②文件创建超 3600 秒时会滚动生成新文件

## 4.6.2 消费者 Flume 配置

### 1) Flume 配置分析



### 2) Flume 的具体配置如下：

（1）在 `hadoop104` 的 `/opt/module/flume/conf` 目录下创建 `kafka-flume-hdfs.conf` 文件

```
[atguigu@hadoop104 conf]$ vim kafka-flume-hdfs.conf
```

在文件配置如下内容

```
## 组件
a1.sources=r1
a1.channels=c1
a1.sinks=k1
```

更多 [Java](#) - [大数据](#) - [前端](#) - [python](#) 人工智能资料下载，可百度访问：尚硅谷官网



```
## source1
a1.sources.r1.type = org.apache.flume.source.kafka.KafkaSource
a1.sources.r1.batchSize = 5000
a1.sources.r1.batchDurationMillis = 2000
a1.sources.r1.kafka.bootstrap.servers =
hadoop102:9092,hadoop103:9092,hadoop104:9092
a1.sources.r1.kafka.topics=topic_log
a1.sources.r1.interceptors = i1
a1.sources.r1.interceptors.i1.type =
com.atguigu.flume.interceptor.TimestampInterceptor$Builder

## channel1
a1.channels.c1.type = file
a1.channels.c1.checkpointDir =
/opt/module/flume/checkpoint/behavior1
a1.channels.c1.dataDirs = /opt/module/flume/data/behavior1/

## sink1
a1.sinks.k1.type = hdfs
a1.sinks.k1.hdfs.path =
/origin_data/gmall/log/topic_log/%Y-%m-%d
a1.sinks.k1.hdfs.filePrefix = log-
a1.sinks.k1.hdfs.round = false

#控制生成的小文件
a1.sinks.k1.hdfs.rollInterval = 10
a1.sinks.k1.hdfs.rollSize = 134217728
a1.sinks.k1.hdfs.rollCount = 0

## 控制输出文件是原生文件。
a1.sinks.k1.hdfs.fileType = CompressedStream
a1.sinks.k1.hdfs.codeC = lzop

## 拼装
a1.sources.r1.channels = c1
a1.sinks.k1.channel= c1
```

### 4.6.3 Flume 时间戳拦截器

由于 Flume 默认会用 Linux 系统时间,作为输出到 HDFS 路径的时间。如果数据是 23:59 分产生的。Flume 消费 Kafka 里面的数据时,有可能已经是第二天了,那么这部门数据会被发往第二天的 HDFS 路径。我们希望的是根据日志里面的实际时间,发往 HDFS 的路径,所以下面拦截器作用是获取日志中的实际时间。

解决的思路:拦截 json 日志,通过 fastjson 框架解析 json,获取实际时间 ts。将获取的 ts 时间写入拦截器 header 头,header 的 key 必须是 **timestamp**,因为 Flume 框架会根据这个 key 的值识别为时间,写入到 HDFS。

1) 在 com.atguigu.flume.interceptor 包下创建 TimestampInterceptor 类

```
package com.atguigu.flume.interceptor;
```

更多 [Java](#) -[大数据](#) -[前端](#) -[python](#) 人工智能资料下载,可百度访问: [尚硅谷官网](#)

```
import com.alibaba.fastjson.JSONObject;
import org.apache.flume.Context;
import org.apache.flume.Event;
import org.apache.flume.interceptor.Interceptor;

import java.nio.charset.StandardCharsets;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;

public class TimestampInterceptor implements Interceptor {

    private ArrayList<Event> events = new ArrayList<>();

    @Override
    public void initialize() {

    }

    @Override
    public Event intercept(Event event) {

        Map<String, String> headers = event.getHeaders();
        String log = new String(event.getBody(),
StandardCharsets.UTF_8);

        JSONObject jsonObject = JSONObject.parseObject(log);

        String ts = jsonObject.getString("ts");
        headers.put("timestamp", ts);

        return event;
    }

    @Override
    public List<Event> intercept(List<Event> list) {
        events.clear();
        for (Event event : list) {
            events.add(intercept(event));
        }

        return events;
    }

    @Override
    public void close() {

    }

    public static class Builder implements Interceptor.Builder
    {
        @Override
        public Interceptor build() {
            return new TimestampInterceptor();
        }

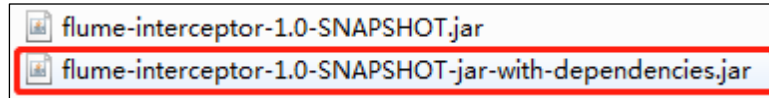
        @Override
```

```

        public void configure(Context context) {
        }
    }
}

```

2) 重新打包



3) 需要先将打好的包放入到 hadoop102 的 /opt/module/flume/lib 文件夹下面。

```

[atguigu@hadoop102 lib]$ ls | grep interceptor
flume-interceptor-1.0-SNAPSHOT-jar-with-dependencies.jar

```

4) 分发 Flume 到 hadoop103、hadoop104

```

[atguigu@hadoop102 module]$ xsync flume/

```

## 4.6.4 消费者 Flume 启动停止脚本

(1) 在 /home/atguigu/bin 目录下创建脚本 f2.sh

```

[atguigu@hadoop102 bin]$ vim f2.sh

```

在脚本中填写如下内容

```

#!/bin/bash

case $1 in
"start"){
    for i in hadoop104
    do
        echo " -----启动 $i 消费 flume-----"
        ssh $i "nohup /opt/module/flume/bin/flume-ng agent
--conf-file /opt/module/flume/conf/kafka-flume-hdfs.conf --name
a1 -Dflume.root.logger=INFO,LOGFILE >/opt/module/flume/log2.txt
2>&1 &"
    done
};;
"stop"){
    for i in hadoop104
    do
        echo " -----停止 $i 消费 flume-----"
        ssh $i "ps -ef | grep kafka-flume-hdfs | grep -v
grep |awk '{print \$2}' | xargs -nl kill"
    done
};;
esac

```

(2) 增加脚本执行权限

```

[atguigu@hadoop102 bin]$ chmod u+x f2.sh

```

(3) f2 集群启动脚本

```

[atguigu@hadoop102 module]$ f2.sh start

```

(4) f2 集群停止脚本

```

[atguigu@hadoop102 module]$ f2.sh stop

```

## 4.6.5 项目经验之 Flume 内存优化

### 1) 问题描述：如果启动消费 Flume 抛出如下异常

```
ERROR hdfs.HDFSEventSink: process failed
java.lang.OutOfMemoryError: GC overhead limit exceeded
```

### 2) 解决方案步骤

(1) 在 hadoop102 服务器的/opt/module/flume/conf/flume-env.sh 文件中增加如下配置

```
export          JAVA_OPTS="-Xms100m          -Xmx2000m          -
Dcom.sun.management.jmxremote"
```

(2) 同步配置到 hadoop103、hadoop104 服务器

```
[atguigu@hadoop102 conf]$ xsync flume-env.sh
```

### 3) Flume 内存参数设置及优化

JVM heap 一般设置为 4G 或更高

-Xmx 与 -Xms 最好设置一致，减少内存抖动带来的性能影响，如果设置不一致容易导致频繁 fullgc。

-Xms 表示 JVM Heap（堆内存）最小尺寸，初始分配；-Xmx 表示 JVM Heap(堆内存)最大允许的尺寸，按需分配。如果不设置一致，容易在初始化时，由于内存不够，频繁触发 fullgc。

## 4.7 采集通道启动/停止脚本

(1) 在/home/atguigu/bin 目录下创建脚本 cluster.sh

```
[atguigu@hadoop102 bin]$ vim cluster.sh
```

在脚本中填写如下内容

```
#!/bin/bash

case $1 in
"start"){
    echo ===== 启动 集群 =====

    #启动 Zookeeper 集群
    zk.sh start

    #启动 Hadoop 集群
    hdp.sh start

    #启动 Kafka 采集集群
    kf.sh start

    #启动 Flume 采集集群
    fl.sh start

    #启动 Flume 消费集群
    f2.sh start
}
```

```
    };;  
"stop"){  
    echo ===== 停止 集群 =====  
  
    #停止 Flume 消费集群  
    f2.sh stop  
  
    #停止 Flume 采集集群  
    f1.sh stop  
  
    #停止 Kafka 采集集群  
    kf.sh stop  
  
    #停止 Hadoop 集群  
    hdp.sh stop  
  
    #停止 Zookeeper 集群  
    zk.sh stop  
  
};;  
esac
```

(2) 增加脚本执行权限

```
[atguigu@hadoop102 bin]$ chmod u+x cluster.sh
```

(3) cluster 集群启动脚本

```
[atguigu@hadoop102 module]$ cluster.sh start
```

(4) cluster 集群停止脚本

```
[atguigu@hadoop102 module]$ cluster.sh stop
```

## 第 5 章 常见问题及解决方案

### 5.1 2NN 页面不能显示完整信息

#### 1) 问题描述

访问 2NN 页面 <http://hadoop104:9868>，看不到详细信息

#### 2) 解决办法

(1) 在浏览器上按 F12，查看问题原因。定位 bug 在 61 行

(2) 找到要修改的文件

```
[atguigu@hadoop102 static]$ pwd  
/opt/module/hadoop-3.1.3/share/hadoop/hdfs/webapps/static  
  
[atguigu@hadoop102 static]$ vim dfs-dust.js  
:set nu  
修改 61 行  
return new Date(Number(v)).toLocaleString();
```

(3) 分发 dfs-dust.js

```
[atguigu@hadoop102 static]$ xsync dfs-dust.js
```

(4) 在 <http://hadoop104:9868/status.html> 页面强制刷新