电商数仓（用户行为数据采集）

版本：V1.7

# 第1章 数据仓库概念



# 第2章 项目需求及架构设计

## 2.1 项目需求分析



## 2.2 项目框架

### 2.2.1 技术选型



### 2.2.2 系统架构图设计



### 2.2.3 系统数据流程设计



### 2.2.4 框架版本选型

|  |  |
| --- | --- |
| 产品 | 版本 |
| Hadoop | 2.7.2 |
| Flume | 1.7.0 |
| Kafka | 0.11.0.2 |
| Kafka Manager | 1.3.3.22 |
| Hive | 1.2.1 |
| Sqoop | 1.4.6 |
| MySQL | 5.6.24 |
| Azkaban | 2.5.0 |
| Java | 1.8 |
| Zookeeper | 3.4.10 |
| Presto | 0.189 |

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。

### 2.2.5 集群资源规划设计

|  |  |  |  |
| --- | --- | --- | --- |
|  | 服务器hadoop102 | 服务器hadoop103 | 服务器hadoop104 |
| HDFS | NameNode  DataNode | DataNode | DataNode  SecondaryNameNode |
| Yarn | NodeManager | Resourcemanager  NodeManager | NodeManager |
| Zookeeper | Zookeeper | Zookeeper | Zookeeper |
| Flume(采集日志) | Flume | Flume |  |
| Kafka | Kafka | Kafka | Kafka |
| Flume（消费Kafka） |  |  | Flume |
| Hive | Hive |  |  |
| MySQL | MySQL |  |  |
| Presto | Presto | Presto | Presto |

# 第3章 数据生成模块

## 3.1 埋点数据基本格式

* 公共字段：基本所有安卓手机都包含的字段
* 业务字段：埋点上报的字段，有具体的业务类型

下面就是一个示例，表示业务字段的上传。

{

"ap":"xxxxx",//产品字段 app key

"cm": { //公共字段

"mid": "", // (String) 设备唯一标识

"uid": "", // (String) 用户标识

"vc": "1", // (String) versionCode，程序版本号

"vn": "1.0", // (String) versionName，程序版本名

"l": "zh", // (String) 系统语言

"sr": "", // (String) 渠道号，应用从哪个渠道来的。

"os": "7.1.1", // (String) Android系统版本

"ar": "CN", // (String) 区域

"md": "BBB100-1", // (String) 手机型号

"ba": "blackberry", // (String) 手机品牌

"sv": "V2.2.1", // (String) sdkVersion

"g": "", // (String) gmail

"hw": "1620x1080", // (String) heightXwidth，屏幕宽高

"t": "1506047606608", // (String) 客户端日志产生时的时间

"nw": "WIFI", // (String) 网络模式

"ln": 0, // (double) lng经度

"la": 0 // (double) lat 纬度

},

"et": [ //事件

{

"ett": "1506047605364", //客户端事件产生时间

"en": "display", //事件名称

"kv": { //事件结果，以key-value形式自行定义

"goodsid": "236",

"action": "1",

"extend1": "1",

"place": "2",

"category": "75"

}

}

]

}

示例日志（服务器时间戳 | 日志）：

1540934156385|{

"ap": "gmall",

"cm": {

"uid": "1234",

"vc": "2",

"vn": "1.0",

"la": "EN",

"sr": "",

"os": "7.1.1",

"ar": "CN",

"md": "BBB100-1",

"ba": "blackberry",

"sv": "V2.2.1",

"g": "abc@gmail.com",

"hw": "1620x1080",

"t": "1506047606608",

"nw": "WIFI",

"ln": 0

},

"et": [

{

"ett": "1506047605364", //客户端事件产生时间

"en": "display", //事件名称

"kv": { //事件结果，以key-value形式自行定义

"goodsid": "236",

"action": "1",

"extend1": "1",

"place": "2",

"category": "75"

}

},{

"ett": "1552352626835",

"en": "active\_background",

"kv": {

"active\_source": "1"

}

}

]

}

}

下面是各个埋点日志格式。其中商品点击属于信息流的范畴

## 3.2 事件日志数据

### 3.2.1 商品列表页(loading)

事件名称：loading

|  |  |
| --- | --- |
| 标签 | 含义 |
| action | 动作：开始加载=1，加载成功=2，加载失败=3 |
| loading\_time | **加载时长：计算下拉开始到接口返回数据的时间，（开始加载报0，加载成功或加载失败才上报时间）** |
| loading\_way | 加载类型：1-读取缓存，2-从接口拉新数据 （加载成功才上报加载类型） |
| extend1 | **扩展字段 Extend1** |
| extend2 | 扩展字段 Extend2 |
| type | **加载类型：自动加载=1，用户下拽加载=2，底部加载=3（底部条触发点击底部提示条/点击返回顶部加载）** |
| type1 | 加载失败码：把加载失败状态码报回来（报空为加载成功，没有失败） |



### 3.2.2 商品点击(display)

事件标签：display

|  |  |  |
| --- | --- | --- |
| 标签 | | 含义 |
| action | 动作：曝光商品=1，点击商品=2， | |
| goodsid | 商品ID（服务端下发的ID） | |
| place | 顺序（第几条商品，第一条为0，第二条为1，如此类推） | |
| extend1 | 曝光类型：1 - 首次曝光 2-重复曝光（没有使用） | |
| category | 分类ID（服务端定义的分类ID） | |



### 3.2.3 商品详情页(newsdetail)

事件标签：newsdetail

|  |  |  |
| --- | --- | --- |
| 标签 | | 含义 |
| entry | 页面入口来源：应用首页=1、push=2、详情页相关推荐=3 | |
| action | 动作：开始加载=1，加载成功=2（pv），加载失败=3, 退出页面=4 | |
| goodsid | 商品ID（服务端下发的ID） | |
| show\_style | 商品样式：0、无图、1、一张大图、2、两张图、3、三张小图、4、一张小图、5、一张大图两张小图 | |
| news\_staytime | 页面停留时长：从商品开始加载时开始计算，到用户关闭页面所用的时间。若中途用跳转到其它页面了，则暂停计时，待回到详情页时恢复计时。或中途划出的时间超过10分钟，则本次计时作废，不上报本次数据。如未加载成功退出，则报空。 | |
| loading\_time | 加载时长：计算页面开始加载到接口返回数据的时间 （开始加载报0，加载成功或加载失败才上报时间） | |
| type1 | 加载失败码：把加载失败状态码报回来（报空为加载成功，没有失败） | |
| category | 分类ID（服务端定义的分类ID） | |



### 3.2.4 广告(ad)

事件名称：ad

|  |  |
| --- | --- |
| 标签 | 含义 |
| entry | 入口：商品列表页=1  应用首页=2 商品详情页=3 |
| action | 动作：请求广告=1 取缓存广告=2  广告位展示=3 广告展示=4 广告点击=5 |
| content | 状态：成功=1  失败=2 |
| detail | 失败码（没有则上报空） |
| source | 广告来源:admob=1 facebook=2  ADX（百度）=3 VK（俄罗斯）=4 |
| behavior | 用户行为： 主动获取广告=1   被动获取广告=2 |
| newstype | Type: 1- 图文 2-图集 3-段子 4-GIF 5-视频 6-调查 7-纯文 8-视频+图文  9-GIF+图文  0-其他 |
| show\_style | 内容样式：无图(纯文字)=6 一张大图=1  三站小图+文=4 一张小图=2 一张大图两张小图+文=3 图集+文 = 5  一张大图+文=11   GIF大图+文=12  视频(大图)+文 = 13 来源于详情页相关推荐的商品，上报样式都为0（因为都是左文右图） |



### 3.2.5 消息通知(notification)

事件标签：notification

|  |  |
| --- | --- |
| 标签 | 含义 |
| action | 动作：通知产生=1，通知弹出=2，通知点击=3，常驻通知展示（不重复上报，一天之内只报一次）=4 |
| type | 通知id：预警通知=1，天气预报（早=2，晚=3），常驻=4 |
| ap\_time | 客户端弹出时间 |
| content | 备用字段 |

### 3.2.6 用户前台活跃(active\_foreground)

事件标签: active\_foreground

|  |  |
| --- | --- |
| 标签 | 含义 |
| push\_id | 推送的消息的id，如果不是从推送消息打开，传空 |
| access | 1.push 2.icon 3.其他 |

### 3.2.7 用户后台活跃(active\_background)

事件标签: active\_background

|  |  |
| --- | --- |
| 标签 | 含义 |
| active\_source | 1=upgrade,2=download(下载),3=plugin\_upgrade |

### 3.2.8 评论（comment）

描述：评论表

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **序号** | **字段名称** | **字段描述** | **字段类型** | **长度** | **允许空** | **缺省值** |
| 1 | comment\_id | 评论表 | int | 10,0 |  |  |
| 2 | userid | 用户id | int | 10,0 | √ | 0 |
| 3 | p\_comment\_id | 父级评论id(为0则是一级评论,不为0则是回复) | int | 10,0 | √ |  |
| 4 | content | 评论内容 | string | 1000 | √ |  |
| 5 | addtime | 创建时间 | string |  | √ |  |
| 6 | other\_id | 评论的相关id | int | 10,0 | √ |  |
| 7 | praise\_count | 点赞数量 | int | 10,0 | √ | 0 |
| 8 | reply\_count | 回复数量 | int | 10,0 | √ | 0 |



### 3.2.9 收藏（favorites）

描述：收藏

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **序号** | **字段名称** | **字段描述** | **字段类型** | **长度** | **允许空** | **缺省值** |
| 1 | id | 主键 | int | 10,0 |  |  |
| 2 | course\_id | 商品id | int | 10,0 | √ | 0 |
| 3 | userid | 用户ID | int | 10,0 | √ | 0 |
| 4 | add\_time | 创建时间 | string |  | √ |  |

### 3.2.10 点赞（praise）

描述：所有的点赞表

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **序号** | **字段名称** | **字段描述** | **字段类型** | **长度** | **允许空** | **缺省值** |
| 1 | id | 主键id | int | 10,0 |  |  |
| 2 | userid | 用户id | int | 10,0 | √ |  |
| 3 | target\_id | 点赞的对象id | int | 10,0 | √ |  |
| 4 | type | 点赞类型 1问答点赞 2问答评论点赞 3 文章点赞数4 评论点赞 | int | 10,0 | √ |  |
| 5 | add\_time | 添加时间 | string |  | √ |  |



### 3.2.11 错误日

|  |  |
| --- | --- |
| errorBrief | 错误摘要 |
| errorDetail | 错误详情 |

## 3.3 启动日志数据

事件标签: start action=1可以算成前台活跃

|  |  |
| --- | --- |
| 标签 | 含义 |
| entry | 入口： push=1，widget=2，icon=3，notification=4, lockscreen\_widget =5 |
| open\_ad\_type | 开屏广告类型: 开屏原生广告=1, 开屏插屏广告=2 |
| action | 状态：成功=1  失败=2 |
| loading\_time | 加载时长：计算下拉开始到接口返回数据的时间，（开始加载报0，加载成功或加载失败才上报时间） |
| detail | 失败码（没有则上报空） |
| extend1 | 失败的message（没有则上报空） |



## 3.4 数据生成脚本



### 3.4.1 创建Maven工程

1）创建log-collector

2）创建一个包名：com.byf.appclient

3）在com.byf.appclient包下创建一个类，AppMain。

4）在pom.xml文件中添加如下内容

*<!--版本号统一-->*<**properties**>  
 <**slf4j.version**>1.7.20</**slf4j.version**>  
 <**logback.version**>1.0.7</**logback.version**>  
</**properties**>  
  
<**dependencies**>  
 *<!--阿里巴巴开源json解析框架-->* <**dependency**>  
 <**groupId**>com.alibaba</**groupId**>  
 <**artifactId**>fastjson</**artifactId**>  
 <**version**>1.2.51</**version**>  
 </**dependency**>  
  
 *<!--日志生成框架-->* <**dependency**>  
 <**groupId**>org.slf4j</**groupId**>  
 <**artifactId**>slf4j-api</**artifactId**>  
 <**version**>${slf4j.version}</**version**>  
 </**dependency**>  
  
 <**dependency**>  
 <**groupId**>ch.qos.logback</**groupId**>  
 <**artifactId**>logback-core</**artifactId**>  
 <**version**>${logback.version}</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>ch.qos.logback</**groupId**>  
 <**artifactId**>logback-classic</**artifactId**>  
 <**version**>${logback.version}</**version**>  
 </**dependency**>  
</**dependencies**>  
  
*<!--编译打包插件-->*<**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**>  
 <**archive**>  
 <**manifest**>  
 <**mainClass**>com.byf.appclient.AppMain</**mainClass**>  
 </**manifest**>  
 </**archive**>  
 </**configuration**>  
 <**executions**>  
 <**execution**>  
 <**id**>make-assembly</**id**>  
 <**phase**>package</**phase**>  
 <**goals**>  
 <**goal**>single</**goal**>  
 </**goals**>  
 </**execution**>  
 </**executions**>  
 </**plugin**>  
 </**plugins**>  
</**build**>

注意：com.byf.appclient.AppMain要和自己建的全类名一致。

### 3.4.2 公共字段Bean

1）创建包名：com.byf.bean

2）在com.byf.bean包下依次创建如下bean对象

**package** com.byf.bean;  
  
**import** java.io.Serializable;  
  
*/\*\*  
 \* 公共日志  
 \*/***public class** AppBase **implements** Serializable{  
  
 **private** String **mid**; *// (String) 设备唯一标识* **private** String **uid**; *// (String) 用户uid* **private** String **vc**; *// (String) versionCode，程序版本号* **private** String **vn**; *// (String) versionName，程序版本名* **private** String **l**; *// (String) 系统语言* **private** String **sr**; *// (String) 渠道号，应用从哪个渠道来的。* **private** String **os**; *// (String) Android系统版本* **private** String **ar**; *// (String) 区域* **private** String **md**; *// (String) 手机型号* **private** String **ba**; *// (String) 手机品牌* **private** String **sv**; *// (String) sdkVersion* **private** String **g**; *// (String) gmail* **private** String **hw**; *// (String) heightXwidth，屏幕宽高* **private** String **t**; *// (String) 客户端日志产生时的时间* **private** String **nw**; *// (String) 网络模式* **private** String **ln**; *// (double) lng经度* **private** String **la**; *// (double) lat 纬度* **public** String getMid() {  
 **return mid**;  
 }  
  
 **public void** setMid(String mid) {  
 **this**.**mid** = mid;  
 }  
  
 **public** String getUid() {  
 **return uid**;  
 }  
  
 **public void** setUid(String uid) {  
 **this**.**uid** = uid;  
 }  
  
 **public** String getVc() {  
 **return vc**;  
 }  
  
 **public void** setVc(String vc) {  
 **this**.**vc** = vc;  
 }  
  
 **public** String getVn() {  
 **return vn**;  
 }  
  
 **public void** setVn(String vn) {  
 **this**.**vn** = vn;  
 }  
  
 **public** String getL() {  
 **return l**;  
 }  
  
 **public void** setL(String l) {  
 **this**.**l** = l;  
 }  
  
 **public** String getSr() {  
 **return sr**;  
 }  
  
 **public void** setSr(String sr) {  
 **this**.**sr** = sr;  
 }  
  
 **public** String getOs() {  
 **return os**;  
 }  
  
 **public void** setOs(String os) {  
 **this**.**os** = os;  
 }  
  
 **public** String getAr() {  
 **return ar**;  
 }  
  
 **public void** setAr(String ar) {  
 **this**.**ar** = ar;  
 }  
  
 **public** String getMd() {  
 **return md**;  
 }  
  
 **public void** setMd(String md) {  
 **this**.**md** = md;  
 }  
  
 **public** String getBa() {  
 **return ba**;  
 }  
  
 **public void** setBa(String ba) {  
 **this**.**ba** = ba;  
 }  
  
 **public** String getSv() {  
 **return sv**;  
 }  
  
 **public void** setSv(String sv) {  
 **this**.**sv** = sv;  
 }  
  
 **public** String getG() {  
 **return g**;  
 }  
  
 **public void** setG(String g) {  
 **this**.**g** = g;  
 }  
  
 **public** String getHw() {  
 **return hw**;  
 }  
  
 **public void** setHw(String hw) {  
 **this**.**hw** = hw;  
 }  
  
 **public** String getT() {  
 **return t**;  
 }  
  
 **public void** setT(String t) {  
 **this**.**t** = t;  
 }  
  
 **public** String getNw() {  
 **return nw**;  
 }  
  
 **public void** setNw(String nw) {  
 **this**.**nw** = nw;  
 }  
  
 **public** String getLn() {  
 **return ln**;  
 }  
  
 **public void** setLn(String ln) {  
 **this**.**ln** = ln;  
 }  
  
 **public** String getLa() {  
 **return la**;  
 }  
  
 **public void** setLa(String la) {  
 **this**.**la** = la;  
 }  
}

### 3.4.3 启动日志Bean

**package** com.byf.bean;  
  
*/\*\*  
 \* 启动日志  
 \*/***public class** AppStart {  
 **private** String **entry**;*//入口： push=1，widget=2，icon=3，notification=4, lockscreen\_widget =5* **private** String **open\_ad\_type**;*//开屏广告类型: 开屏原生广告=1, 开屏插屏广告=2* **private** String **action**;*//状态：成功=1 失败=2* **private** String **loading\_time**;*//加载时长：计算下拉开始到接口返回数据的时间，（开始加载报0，加载成功或加载失败才上报时间）* **private** String **detail**;*//失败码（没有则上报空）* **private** String **extend1**;*//失败的message（没有则上报空）* **public** String getEntry() {  
 **return entry**;  
 }  
  
 **public void** setEntry(String entry) {  
 **this**.**entry** = entry;  
 }  
  
 **public** String getOpen\_ad\_type() {  
 **return open\_ad\_type**;  
 }  
  
 **public void** setOpen\_ad\_type(String open\_ad\_type) {  
 **this**.**open\_ad\_type** = open\_ad\_type;  
 }  
  
 **public** String getAction() {  
 **return action**;  
 }  
  
 **public void** setAction(String action) {  
 **this**.**action** = action;  
 }  
  
 **public** String getLoading\_time() {  
 **return loading\_time**;  
 }  
  
 **public void** setLoading\_time(String loading\_time) {  
 **this**.**loading\_time** = loading\_time;  
 }  
  
 **public** String getDetail() {  
 **return detail**;  
 }  
  
 **public void** setDetail(String detail) {  
 **this**.**detail** = detail;  
 }  
  
 **public** String getExtend1() {  
 **return extend1**;  
 }  
  
 **public void** setExtend1(String extend1) {  
 **this**.**extend1** = extend1;  
 }  
}

### 3.4.3 错误日志Bean

**package** com.byf.bean;  
  
*/\*\*  
 \* 错误日志  
 \*/***public class** AppErrorLog {  
  
 **private** String **errorBrief**; *//错误摘要* **private** String **errorDetail**; *//错误详情* **public** String getErrorBrief() {  
 **return errorBrief**;  
 }  
  
 **public void** setErrorBrief(String errorBrief) {  
 **this**.**errorBrief** = errorBrief;  
 }  
  
 **public** String getErrorDetail() {  
 **return errorDetail**;  
 }  
  
 **public void** setErrorDetail(String errorDetail) {  
 **this**.**errorDetail** = errorDetail;  
 }  
}

### 3.4.4 事件日志Bean之商品点击

package com.byf.bean;

/\*\*

\* 商品点击日志

\*/

public class AppDisplay {

private String action;//动作：曝光商品=1，点击商品=2，

private String goodsid;//商品ID（服务端下发的ID）

private String place;//顺序（第几条商品，第一条为0，第二条为1，如此类推）

private String extend1;//曝光类型：1 - 首次曝光 2-重复曝光（没有使用）

private String category;//分类ID（服务端定义的分类ID）

public String getAction() {

return action;

}

public void setAction(String action) {

this.action = action;

}

public String getGoodsid() {

return goodsid;

}

public void setGoodsid(String goodsid) {

this.goodsid = goodsid;

}

public String getPlace() {

return place;

}

public void setPlace(String place) {

this.place = place;

}

public String getExtend1() {

return extend1;

}

public void setExtend1(String extend1) {

this.extend1 = extend1;

}

public String getCategory() {

return category;

}

public void setCategory(String category) {

this.category = category;

}

}

### 3.4.5 事件日志Bean之商品详情页

package com.byf.bean;

/\*\*

\* 商品详情

\*/

public class AppNewsDetail {

private String entry;//页面入口来源：应用首页=1、push=2、详情页相关推荐=3

private String action;//动作：开始加载=1，加载成功=2（pv），加载失败=3, 退出页面=4

private String goodsid;//商品ID（服务端下发的ID）

private String showtype;//商品样式：0、无图1、一张大图2、两张图3、三张小图4、一张小图5、一张大图两张小图 来源于详情页相关推荐的商品，上报样式都为0（因为都是左文右图）

private String news\_staytime;//页面停留时长：从商品开始加载时开始计算，到用户关闭页面所用的时间。若中途用跳转到其它页面了，则暂停计时，待回到详情页时恢复计时。或中途划出的时间超过10分钟，则本次计时作废，不上报本次数据。如未加载成功退出，则报空。

private String loading\_time;//加载时长：计算页面开始加载到接口返回数据的时间 （开始加载报0，加载成功或加载失败才上报时间）

private String type1;//加载失败码：把加载失败状态码报回来（报空为加载成功，没有失败）

private String category;//分类ID（服务端定义的分类ID）

public String getEntry() {

return entry;

}

public void setEntry(String entry) {

this.entry = entry;

}

public String getAction() {

return action;

}

public void setAction(String action) {

this.action = action;

}

public String getGoodsid() {

return goodsid;

}

public void setGoodsid(String goodsid) {

this.goodsid = goodsid;

}

public String getShowtype() {

return showtype;

}

public void setShowtype(String showtype) {

this.showtype = showtype;

}

public String getNews\_staytime() {

return news\_staytime;

}

public void setNews\_staytime(String news\_staytime) {

this.news\_staytime = news\_staytime;

}

public String getLoading\_time() {

return loading\_time;

}

public void setLoading\_time(String loading\_time) {

this.loading\_time = loading\_time;

}

public String getType1() {

return type1;

}

public void setType1(String type1) {

this.type1 = type1;

}

public String getCategory() {

return category;

}

public void setCategory(String category) {

this.category = category;

}

}

### 3.4.6 事件日志Bean之商品列表页

**package** com.byf.bean;  
  
*/\*\*  
 \* 商品列表  
 \*/***public class** AppLoading {  
 **private** String **action**;*//动作：开始加载=1，加载成功=2，加载失败=3* **private** String **loading\_time**;*//加载时长：计算下拉开始到接口返回数据的时间，（开始加载报0，加载成功或加载失败才上报时间）* **private** String **loading\_way**;*//加载类型：1-读取缓存，2-从接口拉新数据 （加载成功才上报加载类型）* **private** String **extend1**;*//扩展字段 Extend1* **private** String **extend2**;*//扩展字段 Extend2* **private** String **type**;*//加载类型：自动加载=1，用户下拽加载=2，底部加载=3（底部条触发点击底部提示条/点击返回顶部加载）* **private** String **type1**;*//加载失败码：把加载失败状态码报回来（报空为加载成功，没有失败）* **public** String getAction() {  
 **return action**;  
 }  
  
 **public void** setAction(String action) {  
 **this**.**action** = action;  
 }  
  
 **public** String getLoading\_time() {  
 **return loading\_time**;  
 }  
  
 **public void** setLoading\_time(String loading\_time) {  
 **this**.**loading\_time** = loading\_time;  
 }  
  
 **public** String getLoading\_way() {  
 **return loading\_way**;  
 }  
  
 **public void** setLoading\_way(String loading\_way) {  
 **this**.**loading\_way** = loading\_way;  
 }  
  
 **public** String getExtend1() {  
 **return extend1**;  
 }  
  
 **public void** setExtend1(String extend1) {  
 **this**.**extend1** = extend1;  
 }  
  
 **public** String getExtend2() {  
 **return extend2**;  
 }  
  
 **public void** setExtend2(String extend2) {  
 **this**.**extend2** = extend2;  
 }  
  
 **public** String getType() {  
 **return type**;  
 }  
  
 **public void** setType(String type) {  
 **this**.**type** = type;  
 }  
  
 **public** String getType1() {  
 **return type1**;  
 }  
  
 **public void** setType1(String type1) {  
 **this**.**type1** = type1;  
 }  
}

### 3.4.7 事件日志Bean之广告

**package** com.byf.bean;  
  
*/\*\*  
 \* 广告  
 \*/***public class** AppAd {  
  
 **private** String **entry**;*//入口：商品列表页=1 应用首页=2 商品详情页=3* **private** String **action**;*//动作：请求广告=1 取缓存广告=2 广告位展示=3 广告展示=4 广告点击=5* **private** String **content**;*//状态：成功=1 失败=2* **private** String **detail**;*//失败码（没有则上报空）* **private** String **source**;*//广告来源:admob=1 facebook=2 ADX（百度）=3 VK（俄罗斯）=4* **private** String **behavior**;*//用户行为： 主动获取广告=1 被动获取广告=2* **private** String **newstype**;*//Type: 1- 图文 2-图集 3-段子 4-GIF 5-视频 6-调查 7-纯文 8-视频+图文 9-GIF+图文 0-其他* **private** String **show\_style**;*//内容样式：无图(纯文字)=6 一张大图=1 三站小图+文=4 一张小图=2 一张大图两张小图+文=3 图集+文 = 5  
 //一张大图+文=11 GIF大图+文=12 视频(大图)+文 = 13  
 //来源于详情页相关推荐的商品，上报样式都为0（因为都是左文右图）* **public** String getEntry() {  
 **return entry**;  
 }  
  
 **public void** setEntry(String entry) {  
 **this**.**entry** = entry;  
 }  
  
 **public** String getAction() {  
 **return action**;  
 }  
  
 **public void** setAction(String action) {  
 **this**.**action** = action;  
 }  
  
 **public** String getContent() {  
 **return content**;  
 }  
  
 **public void** setContent(String content) {  
 **this**.**content** = content;  
 }  
  
 **public** String getDetail() {  
 **return detail**;  
 }  
  
 **public void** setDetail(String detail) {  
 **this**.**detail** = detail;  
 }  
  
 **public** String getSource() {  
 **return source**;  
 }  
  
 **public void** setSource(String source) {  
 **this**.**source** = source;  
 }  
  
 **public** String getBehavior() {  
 **return behavior**;  
 }  
  
 **public void** setBehavior(String behavior) {  
 **this**.**behavior** = behavior;  
 }  
  
 **public** String getNewstype() {  
 **return newstype**;  
 }  
  
 **public void** setNewstype(String newstype) {  
 **this**.**newstype** = newstype;  
 }  
  
 **public** String getShow\_style() {  
 **return show\_style**;  
 }  
  
 **public void** setShow\_style(String show\_style) {  
 **this**.**show\_style** = show\_style;  
 }  
}

### 3.4.8 事件日志Bean之消息通知

**package** com.byf.bean;  
  
*/\*\*  
 \* 消息通知日志  
 \*/***public class** AppNotification {  
 **private** String **action**;*//动作：通知产生=1，通知弹出=2，通知点击=3，常驻通知展示（不重复上报，一天之内只报一次）=4* **private** String **type**;*//通知id：预警通知=1，天气预报（早=2，晚=3），常驻=4* **private** String **ap\_time**;*//客户端弹出时间* **private** String **content**;*//备用字段* **public** String getAction() {  
 **return action**;  
 }  
  
 **public void** setAction(String action) {  
 **this**.**action** = action;  
 }  
  
 **public** String getType() {  
 **return type**;  
 }  
  
 **public void** setType(String type) {  
 **this**.**type** = type;  
 }  
  
 **public** String getAp\_time() {  
 **return ap\_time**;  
 }  
  
 **public void** setAp\_time(String ap\_time) {  
 **this**.**ap\_time** = ap\_time;  
 }  
  
 **public** String getContent() {  
 **return content**;  
 }  
  
 **public void** setContent(String content) {  
 **this**.**content** = content;  
 }  
}

### 3.4.9 事件日志Bean之用户前台活跃

**package** com.byf.bean;  
  
*/\*\*  
 \* 用户前台活跃  
 \*/***public class** AppActive\_foreground {  
 **private** String **push\_id**;*//推送的消息的id，如果不是从推送消息打开，传空* **private** String **access**;*//1.push 2.icon 3.其他* **public** String getPush\_id() {  
 **return push\_id**;  
 }  
  
 **public void** setPush\_id(String push\_id) {  
 **this**.**push\_id** = push\_id;  
 }  
  
 **public** String getAccess() {  
 **return access**;  
 }  
  
 **public void** setAccess(String access) {  
 **this**.**access** = access;  
 }  
}

### 3.4.10 事件日志Bean之用户后台活跃

**package** com.byf.bean;  
  
*/\*\*  
 \* 用户后台活跃  
 \*/***public class** AppActive\_background {  
 **private** String **active\_source**;*//1=upgrade,2=download(下载),3=plugin\_upgrade* **public** String getActive\_source() {  
 **return active\_source**;  
 }  
  
 **public void** setActive\_source(String active\_source) {  
 **this**.**active\_source** = active\_source;  
 }  
}

### 3.4.11 事件日志Bean之用户评论

**package** com.byf.bean;  
  
*/\*\*  
 \* 评论  
 \*/***public class** AppComment {  
  
 **private int comment\_id**;*//评论表* **private int userid**;*//用户id* **private int p\_comment\_id**;*//父级评论id(为0则是一级评论,不为0则是回复)* **private** String **content**;*//评论内容* **private** String **addtime**;*//创建时间* **private int other\_id**;*//评论的相关id* **private int praise\_count**;*//点赞数量* **private int reply\_count**;*//回复数量* **public int** getComment\_id() {  
 **return comment\_id**;  
 }  
  
 **public void** setComment\_id(**int** comment\_id) {  
 **this**.**comment\_id** = comment\_id;  
 }  
  
 **public int** getUserid() {  
 **return userid**;  
 }  
  
 **public void** setUserid(**int** userid) {  
 **this**.**userid** = userid;  
 }  
  
 **public int** getP\_comment\_id() {  
 **return p\_comment\_id**;  
 }  
  
 **public void** setP\_comment\_id(**int** p\_comment\_id) {  
 **this**.**p\_comment\_id** = p\_comment\_id;  
 }  
  
 **public** String getContent() {  
 **return content**;  
 }  
  
 **public void** setContent(String content) {  
 **this**.**content** = content;  
 }  
  
 **public** String getAddtime() {  
 **return addtime**;  
 }  
  
 **public void** setAddtime(String addtime) {  
 **this**.**addtime** = addtime;  
 }  
  
 **public int** getOther\_id() {  
 **return other\_id**;  
 }  
  
 **public void** setOther\_id(**int** other\_id) {  
 **this**.**other\_id** = other\_id;  
 }  
  
 **public int** getPraise\_count() {  
 **return praise\_count**;  
 }  
  
 **public void** setPraise\_count(**int** praise\_count) {  
 **this**.**praise\_count** = praise\_count;  
 }  
  
 **public int** getReply\_count() {  
 **return reply\_count**;  
 }  
  
 **public void** setReply\_count(**int** reply\_count) {  
 **this**.**reply\_count** = reply\_count;  
 }  
}

### 3.4.12 事件日志Bean之用户收藏

**package** com.byf.bean;  
  
*/\*\*  
 \* 收藏  
 \*/***public class** AppFavorites {  
 **private int id**;*//主键* **private int course\_id**;*//商品id* **private int userid**;*//用户ID* **private** String **add\_time**;*//创建时间* **public int** getId() {  
 **return id**;  
 }  
  
 **public void** setId(**int** id) {  
 **this**.**id** = id;  
 }  
  
 **public int** getCourse\_id() {  
 **return course\_id**;  
 }  
  
 **public void** setCourse\_id(**int** course\_id) {  
 **this**.**course\_id** = course\_id;  
 }  
  
 **public int** getUserid() {  
 **return userid**;  
 }  
  
 **public void** setUserid(**int** userid) {  
 **this**.**userid** = userid;  
 }  
  
 **public** String getAdd\_time() {  
 **return add\_time**;  
 }  
  
 **public void** setAdd\_time(String add\_time) {  
 **this**.**add\_time** = add\_time;  
 }  
}

### 3.4.13 事件日志Bean之用户点赞

**package** com.byf.bean;  
  
*/\*\*  
 \* 点赞  
 \*/***public class** AppPraise {  
 **private int id**; *//主键id* **private int userid**;*//用户id* **private int target\_id**;*//点赞的对象id* **private int type**;*//点赞类型 1问答点赞 2问答评论点赞 3 文章点赞数4 评论点赞* **private** String **add\_time**;*//添加时间* **public int** getId() {  
 **return id**;  
 }  
  
 **public void** setId(**int** id) {  
 **this**.**id** = id;  
 }  
  
 **public int** getUserid() {  
 **return userid**;  
 }  
  
 **public void** setUserid(**int** userid) {  
 **this**.**userid** = userid;  
 }  
  
 **public int** getTarget\_id() {  
 **return target\_id**;  
 }  
  
 **public void** setTarget\_id(**int** target\_id) {  
 **this**.**target\_id** = target\_id;  
 }  
  
 **public int** getType() {  
 **return type**;  
 }  
  
 **public void** setType(**int** type) {  
 **this**.**type** = type;  
 }  
  
 **public** String getAdd\_time() {  
 **return add\_time**;  
 }  
  
 **public void** setAdd\_time(String add\_time) {  
 **this**.**add\_time** = add\_time;  
 }  
}

### 3.4.14 主函数



在AppMain类中添加如下内容：

**package** com.byf.appclient;  
  
**import** java.io.UnsupportedEncodingException;  
**import** java.util.Random;  
  
**import** com.alibaba.fastjson.JSON;  
**import** com.alibaba.fastjson.JSONArray;  
**import** com.alibaba.fastjson.JSONObject;  
**import** com.byf.bean.\*;  
**import** org.slf4j.Logger;  
**import** org.slf4j.LoggerFactory;  
  
*/\*\*  
 \* 日志行为数据模拟  
 \*/***public class** AppMain {  
  
 **private final static** Logger ***logger*** = LoggerFactory.*getLogger*(AppMain.**class**);  
 **static** Random *rand* = **new** Random();  
 *// 设备id* **public static int** *s\_mid* = 0;  
 *// 用户id* **public static int** *s\_uid* = 0;  
 *// 商品id* **public static int** *s\_goodsid* = 0;  
  
 */\*\*  
 \* 1.appkey  
 \* 2.循环遍历次数  
 \** ***@param args*** *\*/* **public static void** main(String[] args) {  
  
*// appkey的名称* String appkey = args.**length** > 0 ? args[0] : **"gmall"**;  
  
*// 循环遍历次数* **int** loop\_len = args.**length** > 1 ? Integer.*parseInt*(args[1]) : 1000;  
  
 **for** (**int** i = 0; i < loop\_len; i++) {  
  
 JSONObject json = **new** JSONObject();  
 json.put(**"ap"**, appkey);  
 json.put(**"cm"**, *generateComFields*());  
  
 JSONArray eventsArray = **new** JSONArray();  
  
 **int** flag = *rand*.nextInt(2);  
  
 **switch** (flag) {  
 **case** (0):  
 *//应用启动* **if** (*rand*.nextBoolean()) {  
 eventsArray.add(*generateStart*());  
 }  
 **break**;  
  
 **case** (1):  
 *// 事件日志  
 // 商品点击，展示* **if** (*rand*.nextBoolean()) {  
 eventsArray.add(*generateDisplay*());  
 }  
  
 *// 商品详情页* **if** (*rand*.nextBoolean()) {  
 eventsArray.add(*generateNewsDetail*());  
 }  
  
 *// 商品列表页* **if** (*rand*.nextBoolean()) {  
 eventsArray.add(*generateNewList*());  
 }  
  
 *// 广告* **if** (*rand*.nextBoolean()) {  
 eventsArray.add(*generateAd*());  
 }  
  
 *// 消息通知* **if** (*rand*.nextBoolean()) {  
 eventsArray.add(*generateNotification*());  
 }  
  
 *// 用户前台活跃* **if** (*rand*.nextBoolean()) {  
 eventsArray.add(*generatbeforeground*());  
 }  
  
 *// 用户后台活跃* **if** (*rand*.nextBoolean()) {  
 eventsArray.add(*generateBackground*());  
 }  
  
 *//故障日志* **if** (*rand*.nextBoolean()) {  
 eventsArray.add(*generateError*());  
 }  
  
 *// 用户评论* **if** (*rand*.nextBoolean()) {  
 eventsArray.add(*generateComment*());  
 }  
  
 *// 用户收藏* **if** (*rand*.nextBoolean()) {  
 eventsArray.add(*generateFavorites*());  
 }  
  
 *// 用户点赞* **if** (*rand*.nextBoolean()) {  
 eventsArray.add(*generatePraise*());  
 }  
 **break**;  
 }  
  
 json.put(**"et"**, eventsArray);  
  
*// 时间* **long** millis = System.*currentTimeMillis*();  
  
*// 控制台打印* ***logger***.info(millis + **"|"** + json.toJSONString());  
 }  
 }  
  
 */\*\*  
 \* 公共字段设置  
 \*  
 \** ***@return*** *\*/* **static** JSONObject generateComFields() {  
  
 AppBase appBase = **new** AppBase();  
  
*// 设备id* appBase.setMid(*s\_mid* + **""**);  
 *s\_mid*++;  
  
*// 用户id* appBase.setUid(*s\_uid* + **""**);  
 *s\_uid*++;  
  
*// 程序版本号 5,6等* appBase.setVc(**""** + *rand*.nextInt(20));  
*// 程序版本名 v1.1.1* appBase.setVn(**"1."** + *rand*.nextInt(4) + **"."** + *rand*.nextInt(10));  
*// 安卓系统版本* appBase.setOs(**"8."** + *rand*.nextInt(3) + **"."** + *rand*.nextInt(10));  
  
*// 语言 es,en,pt* **int** flag = *rand*.nextInt(3);  
 **switch** (flag) {  
 **case** (0):  
 appBase.setL(**"es"**);  
 **break**;  
 **case** (1):  
 appBase.setL(**"en"**);  
 **break**;  
 **case** (2):  
 appBase.setL(**"pt"**);  
 **break**;  
 }  
  
*// 渠道号 从哪个渠道来的* appBase.setSr(*getRandomChar*(1));  
  
*// 区域* flag = *rand*.nextInt(2);  
 **switch** (flag) {  
 **case** 0:  
 appBase.setAr(**"BR"**);  
 **case** 1:  
 appBase.setAr(**"MX"**);  
 }  
  
*// 手机品牌 ba ,手机型号 md，就取2位数字了* flag = *rand*.nextInt(3);  
 **switch** (flag) {  
 **case** 0:  
 appBase.setBa(**"Sumsung"**);  
 appBase.setMd(**"sumsung-"** + *rand*.nextInt(20));  
 **break**;  
 **case** 1:  
 appBase.setBa(**"Huawei"**);  
 appBase.setMd(**"Huawei-"** + *rand*.nextInt(20));  
 **break**;  
 **case** 2:  
 appBase.setBa(**"HTC"**);  
 appBase.setMd(**"HTC-"** + *rand*.nextInt(20));  
 **break**;  
 }  
  
*// 嵌入sdk的版本* appBase.setSv(**"V2."** + *rand*.nextInt(10) + **"."** + *rand*.nextInt(10));  
*// gmail* appBase.setG(*getRandomCharAndNumr*(8) + **"@gmail.com"**);  
  
*// 屏幕宽高 hw* flag = *rand*.nextInt(4);  
 **switch** (flag) {  
 **case** 0:  
 appBase.setHw(**"640\*960"**);  
 **break**;  
 **case** 1:  
 appBase.setHw(**"640\*1136"**);  
 **break**;  
 **case** 2:  
 appBase.setHw(**"750\*1134"**);  
 **break**;  
 **case** 3:  
 appBase.setHw(**"1080\*1920"**);  
 **break**;  
 }  
  
*// 客户端产生日志时间* **long** millis = System.*currentTimeMillis*();  
 appBase.setT(**""** + (millis - *rand*.nextInt(99999999)));  
  
*// 手机网络模式 3G,4G,WIFI* flag = *rand*.nextInt(3);  
 **switch** (flag) {  
 **case** 0:  
 appBase.setNw(**"3G"**);  
 **break**;  
 **case** 1:  
 appBase.setNw(**"4G"**);  
 **break**;  
 **case** 2:  
 appBase.setNw(**"WIFI"**);  
 **break**;  
 }  
  
*// 拉丁美洲 西经34°46′至西经117°09；北纬32°42′至南纬53°54′  
// 经度* appBase.setLn((-34 - *rand*.nextInt(83) - *rand*.nextInt(60) / 10.0) + **""**);  
*// 纬度* appBase.setLa((32 - *rand*.nextInt(85) - *rand*.nextInt(60) / 10.0) + **""**);  
  
 JSONObject common = (JSONObject) JSON.*toJSON*(appBase);  
  
 **return** common;  
 }  
  
 */\*\*  
 \* 商品展示事件  
 \*  
 \** ***@return*** *\*/* **static** JSONObject generateDisplay() {  
  
 AppDisplay appDisplay = **new** AppDisplay();  
  
 **boolean** boolFlag = *rand*.nextInt(10) < 7 ? **true** : **false**;  
  
*// 动作：曝光商品=1，点击商品=2，* **if** (boolFlag) {  
 appDisplay.setAction(**"1"**);  
 } **else** {  
 appDisplay.setAction(**"2"**);  
 }  
  
*// 商品id* String goodsId = *s\_goodsid* + **""**;  
 *s\_goodsid*++;  
  
 appDisplay.setGoodsid(goodsId);  
  
*// 顺序 设置成6条吧* **int** flag = *rand*.nextInt(6);  
 appDisplay.setPlace(**""** + flag);  
  
*// 曝光类型* flag = 1 + *rand*.nextInt(2);  
 appDisplay.setExtend1(**""** + flag);  
  
*// 分类* flag = 1 + *rand*.nextInt(100);  
 appDisplay.setCategory(**""** + flag);  
  
 JSONObject jsonObject = (JSONObject) JSON.*toJSON*(appDisplay);  
  
 **return** *packEventJson*(**"display"**, jsonObject);  
 }  
  
 */\*\*  
 \* 商品详情页  
 \*  
 \** ***@return*** *\*/* **static** JSONObject generateNewsDetail() {  
  
 AppNewsDetail appNewsDetail = **new** AppNewsDetail();  
  
*// 页面入口来源* **int** flag = 1 + *rand*.nextInt(3);  
 appNewsDetail.setEntry(flag + **""**);  
  
*// 动作* appNewsDetail.setAction(**""** + (*rand*.nextInt(4) + 1));  
  
*// 商品id* appNewsDetail.setGoodsid(*s\_goodsid* + **""**);  
  
*// 商品来源类型* flag = 1 + *rand*.nextInt(3);  
 appNewsDetail.setShowtype(flag + **""**);  
  
*// 商品样式* flag = *rand*.nextInt(6);  
 appNewsDetail.setShowtype(**""** + flag);  
  
*// 页面停留时长* flag = *rand*.nextInt(10) \* *rand*.nextInt(7);  
 appNewsDetail.setNews\_staytime(flag + **""**);  
  
*// 加载时长* flag = *rand*.nextInt(10) \* *rand*.nextInt(7);  
 appNewsDetail.setLoading\_time(flag + **""**);  
  
*// 加载失败码* flag = *rand*.nextInt(10);  
 **switch** (flag) {  
 **case** 1:  
 appNewsDetail.setType1(**"102"**);  
 **break**;  
 **case** 2:  
 appNewsDetail.setType1(**"201"**);  
 **break**;  
 **case** 3:  
 appNewsDetail.setType1(**"325"**);  
 **break**;  
 **case** 4:  
 appNewsDetail.setType1(**"433"**);  
 **break**;  
 **case** 5:  
 appNewsDetail.setType1(**"542"**);  
 **break**;  
 **default**:  
 appNewsDetail.setType1(**""**);  
 **break**;  
 }  
  
*// 分类* flag = 1 + *rand*.nextInt(100);  
 appNewsDetail.setCategory(**""** + flag);  
  
 JSONObject eventJson = (JSONObject) JSON.*toJSON*(appNewsDetail);  
  
 **return** *packEventJson*(**"newsdetail"**, eventJson);  
 }  
  
 */\*\*  
 \* 商品列表  
 \*  
 \** ***@return*** *\*/* **static** JSONObject generateNewList() {  
  
 AppLoading appLoading = **new** AppLoading();  
  
*// 动作* **int** flag = *rand*.nextInt(3) + 1;  
 appLoading.setAction(flag + **""**);  
  
*// 加载时长* flag = *rand*.nextInt(10) \* *rand*.nextInt(7);  
 appLoading.setLoading\_time(flag + **""**);  
  
*// 失败码* flag = *rand*.nextInt(10);  
 **switch** (flag) {  
 **case** 1:  
 appLoading.setType1(**"102"**);  
 **break**;  
 **case** 2:  
 appLoading.setType1(**"201"**);  
 **break**;  
 **case** 3:  
 appLoading.setType1(**"325"**);  
 **break**;  
 **case** 4:  
 appLoading.setType1(**"433"**);  
 **break**;  
 **case** 5:  
 appLoading.setType1(**"542"**);  
 **break**;  
 **default**:  
 appLoading.setType1(**""**);  
 **break**;  
 }  
  
*// 页面 加载类型* flag = 1 + *rand*.nextInt(2);  
 appLoading.setLoading\_way(**""** + flag);  
  
*// 扩展字段1* appLoading.setExtend1(**""**);  
  
*// 扩展字段2* appLoading.setExtend2(**""**);  
  
*// 用户加载类型* flag = 1 + *rand*.nextInt(3);  
 appLoading.setType(**""** + flag);  
  
 JSONObject jsonObject = (JSONObject) JSON.*toJSON*(appLoading);  
  
 **return** *packEventJson*(**"loading"**, jsonObject);  
 }  
  
 */\*\*  
 \* 广告相关字段  
 \*  
 \** ***@return*** *\*/* **static** JSONObject generateAd() {  
  
 AppAd appAd = **new** AppAd();  
  
*// 入口* **int** flag = *rand*.nextInt(3) + 1;  
 appAd.setEntry(flag + **""**);  
  
*// 动作* flag = *rand*.nextInt(5) + 1;  
 appAd.setAction(flag + **""**);  
  
*// 状态* flag = *rand*.nextInt(10) > 6 ? 2 : 1;  
 appAd.setContent(flag + **""**);  
  
*// 失败码* flag = *rand*.nextInt(10);  
 **switch** (flag) {  
 **case** 1:  
 appAd.setDetail(**"102"**);  
 **break**;  
 **case** 2:  
 appAd.setDetail(**"201"**);  
 **break**;  
 **case** 3:  
 appAd.setDetail(**"325"**);  
 **break**;  
 **case** 4:  
 appAd.setDetail(**"433"**);  
 **break**;  
 **case** 5:  
 appAd.setDetail(**"542"**);  
 **break**;  
 **default**:  
 appAd.setDetail(**""**);  
 **break**;  
 }  
  
*// 广告来源* flag = *rand*.nextInt(4) + 1;  
 appAd.setSource(flag + **""**);  
  
*// 用户行为* flag = *rand*.nextInt(2) + 1;  
 appAd.setBehavior(flag + **""**);  
  
*// 商品类型* flag = *rand*.nextInt(10);  
 appAd.setNewstype(**""** + flag);  
  
*// 展示样式* flag = *rand*.nextInt(6);  
 appAd.setShow\_style(**""** + flag);  
  
 JSONObject jsonObject = (JSONObject) JSON.*toJSON*(appAd);  
  
 **return** *packEventJson*(**"ad"**, jsonObject);  
 }  
  
 */\*\*  
 \* 启动日志  
 \*  
 \** ***@return*** *\*/* **static** JSONObject generateStart() {  
  
 AppStart appStart = **new** AppStart();  
  
*// 入口* **int** flag = *rand*.nextInt(5) + 1;  
 appStart.setEntry(flag + **""**);  
  
*// 开屏广告类型* flag = *rand*.nextInt(2) + 1;  
 appStart.setOpen\_ad\_type(flag + **""**);  
  
*// 状态* flag = *rand*.nextInt(10) > 8 ? 2 : 1;  
 appStart.setAction(flag + **""**);  
  
*// 加载时长* appStart.setLoading\_time(*rand*.nextInt(20) + **""**);  
  
*// 失败码* flag = *rand*.nextInt(10);  
 **switch** (flag) {  
 **case** 1:  
 appStart.setDetail(**"102"**);  
 **break**;  
 **case** 2:  
 appStart.setDetail(**"201"**);  
 **break**;  
 **case** 3:  
 appStart.setDetail(**"325"**);  
 **break**;  
 **case** 4:  
 appStart.setDetail(**"433"**);  
 **break**;  
 **case** 5:  
 appStart.setDetail(**"542"**);  
 **break**;  
 **default**:  
 appStart.setDetail(**""**);  
 **break**;  
 }  
  
 JSONObject jsonObject = (JSONObject) JSON.*toJSON*(appStart);  
  
 **return** *packEventJson*(**"start"**, jsonObject);  
 }  
  
 */\*\*  
 \* 消息通知  
 \** ***@return*** *\*/* **static** JSONObject generateNotification() {  
  
 AppNotification appNotification = **new** AppNotification();  
  
 **int** flag = *rand*.nextInt(4) + 1;  
  
*// 动作* appNotification.setAction(flag + **""**);  
  
*// 通知id* flag = *rand*.nextInt(4) + 1;  
 appNotification.setType(flag + **""**);  
  
*// 客户端弹时间* appNotification.setAp\_time((System.*currentTimeMillis*() - *rand*.nextInt(99999999)) + **""**);  
  
*// 备用字段* appNotification.setContent(**""**);  
  
 JSONObject jsonObject = (JSONObject) JSON.*toJSON*(appNotification);  
  
 **return** *packEventJson*(**"notification"**, jsonObject);  
 }  
  
 */\*\*  
 \* 前台活跃  
 \** ***@return*** *\*/* **static** JSONObject generatbeforeground() {  
  
 AppActive\_foreground appActive\_foreground = **new** AppActive\_foreground();  
  
*// 推送消息的id* **int** flag = *rand*.nextInt(2);  
 **switch** (flag) {  
 **case** 1:  
 appActive\_foreground.setAccess(flag + **""**);  
 **break**;  
 **default**:  
 appActive\_foreground.setAccess(**""**);  
 **break**;  
 }  
  
*// 1.push 2.icon 3.其他* flag = *rand*.nextInt(3) + 1;  
 appActive\_foreground.setPush\_id(flag + **""**);  
  
 JSONObject jsonObject = (JSONObject) JSON.*toJSON*(appActive\_foreground);  
  
 **return** *packEventJson*(**"active\_foreground"**, jsonObject);  
 }  
  
 */\*\*  
 \* 后台活跃  
 \*  
 \** ***@return*** *\*/* **static** JSONObject generateBackground() {  
  
 AppActive\_background appActive\_background = **new** AppActive\_background();  
  
*// 启动源* **int** flag = *rand*.nextInt(3) + 1;  
 appActive\_background.setActive\_source(flag + **""**);  
  
 JSONObject jsonObject = (JSONObject) JSON.*toJSON*(appActive\_background);  
  
 **return** *packEventJson*(**"active\_background"**, jsonObject);  
 }  
  
 */\*\*  
 \* 错误日志数据  
 \*  
 \** ***@return*** *\*/* **static** JSONObject generateError() {  
  
 AppErrorLog appErrorLog = **new** AppErrorLog();  
  
 String[] errorBriefs = {**"at cn.lift.dfdf.web.AbstractBaseController.validInbound(AbstractBaseController.java:72)"**, **"at cn.lift.appIn.control.CommandUtil.getInfo(CommandUtil.java:67)"**}; *//错误摘要* String[] errorDetails = {**"java.lang.NullPointerException\\n "** + **"at cn.lift.appIn.web.AbstractBaseController.validInbound(AbstractBaseController.java:72)\\n "** + **"at cn.lift.dfdf.web.AbstractBaseController.validInbound"**, **"at cn.lift.dfdfdf.control.CommandUtil.getInfo(CommandUtil.java:67)\\n "** + **"at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)\\n"** + **" at java.lang.reflect.Method.invoke(Method.java:606)\\n"**}; *//错误详情  
  
 //错误摘要* appErrorLog.setErrorBrief(errorBriefs[*rand*.nextInt(errorBriefs.**length**)]);  
 *//错误详情* appErrorLog.setErrorDetail(errorDetails[*rand*.nextInt(errorDetails.**length**)]);  
  
 JSONObject jsonObject = (JSONObject) JSON.*toJSON*(appErrorLog);  
  
 **return** *packEventJson*(**"error"**, jsonObject);  
 }  
  
 */\*\*  
 \* 为各个事件类型的公共字段（时间、事件类型、Json数据）拼接  
 \*  
 \** ***@param eventName*** *\** ***@param jsonObject*** *\** ***@return*** *\*/* **static** JSONObject packEventJson(String eventName, JSONObject jsonObject) {  
  
 JSONObject eventJson = **new** JSONObject();  
  
 eventJson.put(**"ett"**, (System.*currentTimeMillis*() - *rand*.nextInt(99999999)) + **""**);  
 eventJson.put(**"en"**, eventName);  
 eventJson.put(**"kv"**, jsonObject);  
  
 **return** eventJson;  
 }  
  
 */\*\*  
 \* 获取随机字母组合  
 \*  
 \** ***@param length*** *字符串长度  
 \** ***@return*** *\*/* **public static** String getRandomChar(Integer length) {  
  
 String str = **""**;  
 Random random = **new** Random();  
  
 **for** (**int** i = 0; i < length; i++) {  
 *// 字符串* str += (**char**) (65 + random.nextInt(26));*// 取得大写字母* }  
  
 **return** str;  
 }  
  
 */\*\*  
 \* 获取随机字母数字组合  
 \*  
 \** ***@param length*** *字符串长度  
 \** ***@return*** *\*/* **public static** String getRandomCharAndNumr(Integer length) {  
  
 String str = **""**;  
 Random random = **new** Random();  
  
 **for** (**int** i = 0; i < length; i++) {  
  
 **boolean** b = random.nextBoolean();  
  
 **if** (b) { *// 字符串  
 // int choice = random.nextBoolean() ? 65 : 97; 取得65大写字母还是97小写字母* str += (**char**) (65 + random.nextInt(26));*// 取得大写字母* } **else** { *// 数字* str += String.*valueOf*(random.nextInt(10));  
 }  
 }  
  
 **return** str;  
 }  
  
 */\*\*  
 \* 收藏  
 \** ***@return*** *\*/* **static public** JSONObject generateFavorites() {  
  
 AppFavorites favorites = **new** AppFavorites();  
  
 favorites.setCourse\_id(*rand*.nextInt(10));  
 favorites.setUserid(*rand*.nextInt(10));  
 favorites.setAdd\_time((System.*currentTimeMillis*() - *rand*.nextInt(99999999)) + **""**);  
  
 JSONObject jsonObject = (JSONObject) JSON.*toJSON*(favorites);  
  
 **return** *packEventJson*(**"favorites"**, jsonObject);  
 }  
  
 */\*\*  
 \* 点赞  
 \** ***@return*** *\*/* **static public** JSONObject generatePraise() {  
  
 AppPraise praise = **new** AppPraise();  
  
 praise.setId(*rand*.nextInt(10));  
 praise.setUserid(*rand*.nextInt(10));  
 praise.setTarget\_id(*rand*.nextInt(10));  
 praise.setType(*rand*.nextInt(4) + 1);  
 praise.setAdd\_time((System.*currentTimeMillis*() - *rand*.nextInt(99999999)) + **""**);  
  
 JSONObject jsonObject = (JSONObject) JSON.*toJSON*(praise);  
  
 **return** *packEventJson*(**"praise"**, jsonObject);  
 }  
  
 */\*\*  
 \* 评论  
 \** ***@return*** *\*/* **static public** JSONObject generateComment() {  
  
 AppComment comment = **new** AppComment();  
  
 comment.setComment\_id(*rand*.nextInt(10));  
 comment.setUserid(*rand*.nextInt(10));  
 comment.setP\_comment\_id(*rand*.nextInt(5));  
  
 comment.setContent(*getCONTENT*());  
 comment.setAddtime((System.*currentTimeMillis*() - *rand*.nextInt(99999999)) + **""**);  
  
 comment.setOther\_id(*rand*.nextInt(10));  
 comment.setPraise\_count(*rand*.nextInt(1000));  
 comment.setReply\_count(*rand*.nextInt(200));  
  
 JSONObject jsonObject = (JSONObject) JSON.*toJSON*(comment);  
  
 **return** *packEventJson*(**"comment"**, jsonObject);  
 }  
  
 */\*\*  
 \* 生成单个汉字  
 \*  
 \** ***@return*** *\*/* **private static char** getRandomChar() {  
  
 String str = **""**;  
 **int** hightPos; *//* **int** lowPos;  
  
 Random random = **new** Random();  
  
 *//随机生成汉子的两个字节* hightPos = (176 + Math.*abs*(random.nextInt(39)));  
 lowPos = (161 + Math.*abs*(random.nextInt(93)));  
  
 **byte**[] b = **new byte**[2];  
 b[0] = (Integer.*valueOf*(hightPos)).byteValue();  
 b[1] = (Integer.*valueOf*(lowPos)).byteValue();  
  
 **try** {  
 str = **new** String(b, **"GBK"**);  
 } **catch** (UnsupportedEncodingException e) {  
 e.printStackTrace();  
 System.***out***.println(**"错误"**);  
 }  
  
 **return** str.charAt(0);  
 }  
  
 */\*\*  
 \* 拼接成多个汉字  
 \*  
 \** ***@return*** *\*/* **static public** String getCONTENT() {  
  
 String str = **""**;  
  
 **for** (**int** i = 0; i < *rand*.nextInt(100); i++) {  
 str += *getRandomChar*();  
 }  
  
 **return** str;  
 }  
}

### 3.4.15 配置日志打印Logback

Spring boot x集成logback与SL4J一起使用，在磁盘打印日志，然后收集。Logback的配置文件由一个个appender组成。一个appender包括：

* 日志打印的方式（控制台，文件，远程）。其中远程的方式是将所有日志发送到日志服务器，然后统一管理。
* 日志的格式。（线程，调用方法类，级别，日志内容大小。）
* 日志的路径。
* 保留的时间长度。
* 日志切分的大小

异步日志打印非常关键，这样在高并发的时候不会让请求等待就可以返回。其中日志分为了3种。

* 控制台
* 错误日志输出（后台需要查看）
* 标准日志输出

具体使用：

1）在resources文件夹下创建logback.xml文件。

2）在logback.xml文件中填写如下配置

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

<**configuration debug="false"**>  
 *<!--定义日志文件的存储地址 勿在 LogBack 的配置中使用相对路径 -->* <**property name="LOG\_HOME" value="/tmp/logs/"** />  
 *<!-- 控制台输出 -->* <**appender name="STDOUT"  
 class="ch.qos.logback.core.ConsoleAppender"**>  
 <**encoder  
 class="ch.qos.logback.classic.encoder.PatternLayoutEncoder"**>  
 *<!--格式化输出：%d表示日期，%thread表示线程名，%-5level：级别从左显示5个字符宽度%msg：日志消息，%n是换行符 -->* <**pattern**>%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %-5level %logger{50} - %msg%n</**pattern**>  
 </**encoder**>  
 </**appender**>  
   
 *<!-- 按照每天生成日志文件。存储事件日志 -->* <**appender name="FILE"  
 class="ch.qos.logback.core.rolling.RollingFileAppender"**>  
 *<!-- <File>${LOG\_HOME}/app.log</File>设置日志不超过${log.max.size}时的保存路径，注意，如果是web项目会保存到Tomcat的bin目录 下 -->* <**rollingPolicy  
 class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy"**>  
 *<!--日志文件输出的文件名 -->* <**FileNamePattern**>${LOG\_HOME}/app-%d{yyyy-MM-dd}.log</**FileNamePattern**>  
 *<!--日志文件保留天数 -->* <**MaxHistory**>30</**MaxHistory**>  
 </**rollingPolicy**>  
 <**encoder  
 class="ch.qos.logback.classic.encoder.PatternLayoutEncoder"**>  
 *<!--<pattern>%ip|%msg%n</pattern>-->* <**pattern**>%msg%n</**pattern**>  
 </**encoder**>  
 *<!--日志文件最大的大小 -->* <**triggeringPolicy  
 class="ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy"**>  
 <**MaxFileSize**>10MB</**MaxFileSize**>  
 </**triggeringPolicy**>  
 </**appender**>  
  
 *<!--异步打印日志-->* <**appender name ="ASYNC\_FILE" class= "ch.qos.logback.classic.AsyncAppender"**>  
 *<!-- 不丢失日志.默认的,如果队列的80%已满,则会丢弃TRACT、DEBUG、INFO级别的日志 -->* <**discardingThreshold** >0</**discardingThreshold**>  
 *<!-- 更改默认的队列的深度,该值会影响性能.默认值为256 -->* <**queueSize**>512</**queueSize**>  
 *<!-- 添加附加的appender,最多只能添加一个 -->* <**appender-ref ref = "FILE"**/>  
 </**appender**>  
  
 *<!-- 日志输出级别 -->* <**root level="INFO"**>  
 <**appender-ref ref="STDOUT"** />  
 <**appender-ref ref="ASYNC\_FILE"** />  
 <**appender-ref ref="error"** />  
 </**root**>  
</**configuration**>

### 3.4.16 打包

1）采用Maven对程序打包

2）采用带依赖的jar包。包含了程序运行需要的所有依赖。

### 3.4.17 启动

要求：启动前必须先把Hadoop和Zookeeper集群搭建完成。

1）代码参数说明

*// appkey的名称*String appkey = args.**length** > 0 ? args[0] : **"gmall"**;  
*// 循环遍历次数***int** loop\_len = args.**length** > 1 ? Integer.*parseInt*(args[1]) : 1000;

这里面有四个参数：

* Appkey的名称，比如我有多个应用，那么我可以生成多个应用的日志
* 循环遍历次数，就是生成多少条日志

2）将生成的jar包log-collector-0.0.1-SNAPSHOT-jar-with-dependencies.jar拷贝到hadoop102、服务器上，并同步到hadoop103的/opt/module路径下，

[byf@hadoop102 module]$ xsync log-collector-1.0-SNAPSHOT-jar-with-dependencies.jar

3）在hadoop102上执行jar程序

[byf@hadoop102 module]$ java -classpath log-collector-1.0-SNAPSHOT-jar-with-dependencies.jar com.byf.appclient.AppMain >/opt/module/test.log

4）在/tmp/logs路径下查看生成的日志文件

[byf@hadoop102 module]$ cd /tmp/logs/

[byf@hadoop102 logs]$ ls

app-2019-02-10.log

### 3.4.18 日志生成集群启动脚本

1）Linux环境变量配置：

（1）修改/etc/profile文件：所有用户的Shell都有权使用这些环境变量。

（2）修改~/.bashrc文件：针对某一个特定的用户，如果你需要给某个用户权限使用这些环境变量，你只需要修改其个人用户主目录下的.bashrc文件就可以了。

（3）配置登录远程服务器立即source一下环境变量

[byf@hadoop102 ~]$ cat /etc/profile >> .bashrc

[byf@hadoop103 ~]$ cat /etc/profile >> .bashrc

[byf@hadoop104 ~]$ cat /etc/profile >> .bashrc

2）具体脚本编写

在/home/byf/bin目录下创建脚本lg.sh

[byf@hadoop102 bin]$ vim lg.sh

在脚本中编写如下内容

#! /bin/bash

for i in hadoop102 hadoop103

do

ssh $i "java -classpath /opt/module/log-collector-1.0-SNAPSHOT-jar-with-dependencies.jar com.byf.appclient.AppMain >/opt/module/test.log &"

done

修改脚本执行权限

[byf@hadoop102 bin]$ chmod 777 lg.sh

启动脚本

[byf@hadoop102 module]$ lg.sh

分别在hadoop102、hadoop103的/tmp/logs目录上查看生成的数据

[byf@hadoop102 logs]$ ls

app-2019-02-10.log

[byf@hadoop103 logs]$ ls

app-2019-02-10.log

### 3.4.19 集群时间同步修改脚本

在/home/byf/bin目录下创建脚本dt.sh

[byf@hadoop102 bin]$ vim dt.sh

在脚本中编写如下内容

#!/bin/bash

log\_date=$1

for i in hadoop102 hadoop103 hadoop104

do

ssh -t $i "sudo date -s $log\_date"

done

修改脚本执行权限

[byf@hadoop102 bin]$ chmod 777 dt.sh

启动脚本

[byf@hadoop102 bin]$ dt.sh 2019-2-10

### 3.4.20 集群所有进程查看脚本

在/home/byf/bin目录下创建脚本xcall.sh

[byf@hadoop102 bin]$ vim xcall.sh

在脚本中编写如下内容

#! /bin/bash

for i in hadoop102 hadoop103 hadoop104

do

echo --------- $i ----------

ssh $i "$\*"

done

修改脚本执行权限

[byf@hadoop102 bin]$ chmod 777 xcall.sh

启动脚本

[byf@hadoop102 bin]$ xcall.sh jps

# 第4章 数据采集模块

## 4.1 Hadoop安装

详见： Hadoop（入门）



|  |  |  |  |
| --- | --- | --- | --- |
|  | 服务器hadoop102 | 服务器hadoop103 | 服务器hadoop104 |
| HDFS | NameNode  DataNode | DataNode | DataNode  SecondaryNameNode |
| Yarn | NodeManager | Resourcemanager  NodeManager | NodeManager |

### 4.1.1 添加LZO支持包

1）先下载lzo的jar项目

<https://github.com/twitter/hadoop-lzo/archive/master.zip>

2）下载后的文件名是hadoop-lzo-master，它是一个zip格式的压缩包，先进行解压，然后用maven编译。生成hadoop-lzo-0.4.20。

3）将编译好后的hadoop-lzo-0.4.20.jar 放入hadoop-2.7.2/share/hadoop/common/

[byf@hadoop102 common]$ pwd

/opt/module/hadoop-2.7.2/share/hadoop/common

[byf@hadoop102 common]$ ls

hadoop-lzo-0.4.20.jar

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

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

### 4.1.2 添加配置

1）core-site.xml增加配置支持LZO压缩

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

<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<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>

2）同步core-site.xml到hadoop103、hadoop104

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

### 4.1.2 启动集群

[byf@hadoop102 hadoop-2.7.2]$ sbin/start-dfs.sh

[byf@hadoop103 hadoop-2.7.2]$ sbin/start-yarn.sh

### 4.1.3 验证

1）web和进程查看

* Web查看：<http://hadoop102:50070>
* 进程查看：jps查看各个节点状态。

2）当启动发生错误的时候：

* 查看日志：/home/byf/module/hadoop-2.7.2/logs
* 如果进入安全模式，可以通过hdfs dfsadmin -safemode leave
* 停止所有进程，删除data和log文件夹，然后hdfs namenode -format 来格式化

## 4.2 Zookeeper安装

### 4.2.1 安装ZK

详见： Zookeeper



|  |  |  |  |
| --- | --- | --- | --- |
|  | 服务器hadoop102 | 服务器hadoop103 | 服务器hadoop104 |
| Zookeeper | Zookeeper | Zookeeper | Zookeeper |

### 4.2.2 ZK集群启动停止脚本

1）在hadoop102的/home/byf/bin目录下创建脚本

[byf@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

ssh $i "/opt/module/zookeeper-3.4.10/bin/zkServer.sh start"

done

};;

"stop"){

for i in hadoop102 hadoop103 hadoop104

do

ssh $i "/opt/module/zookeeper-3.4.10/bin/zkServer.sh stop"

done

};;

esac

2）增加脚本执行权限

[byf@hadoop102 bin]$ chmod 777 zk.sh

3）Zookeeper集群启动脚本

[byf@hadoop102 module]$ zk.sh start

4）Zookeeper集群停止脚本

[byf@hadoop102 module]$ zk.sh stop

## 4.3 Flume安装



集群规划

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

### 4.3.1 日志采集Flume安装

详见： Flume



### 4.3.2 日志采集Flume配置

1）Flume配置分析



注意，TailDirSource是Flume 1.7提供的Source组件，在1.6中并没有。

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

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

在文件配置如下内容

a1.sources=r1

a1.channels=c1 c2

a1.sinks=k1 k2

# configure source

a1.sources.r1.type = TAILDIR

a1.sources.r1.positionFile = /opt/module/flume/log\_position.json

a1.sources.r1.filegroups = f1

a1.sources.r1.filegroups.f1 = /tmp/logs/app.+

a1.sources.r1.fileHeader = true

a1.sources.r1.channels = c1 c2

#interceptor

a1.sources.r1.interceptors = i1 i2

a1.sources.r1.interceptors.i1.type = com.byf.flume.interceptor.LogETLInterceptor$Builder

a1.sources.r1.interceptors.i2.type = com.byf.flume.interceptor.LogTypeInterceptor$Builder

# selector

a1.sources.r1.selector.type = multiplexing

a1.sources.r1.selector.header = logType

a1.sources.r1.selector.mapping.start = c1

a1.sources.r1.selector.mapping.event = c2

# configure channel

a1.channels.c1.type = memory

a1.channels.c1.capacity=10000

a1.channels.c1.byteCapacityBufferPercentage=20

a1.channels.c2.type = memory

a1.channels.c2.capacity=10000

a1.channels.c2.byteCapacityBufferPercentage=20

# configure sink

# start-sink

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

a1.sinks.k1.kafka.topic = topic\_start

a1.sinks.k1.kafka.bootstrap.servers = hadoop102:9092,hadoop103:9092,hadoop104:9092

a1.sinks.k1.kafka.flumeBatchSize = 2000

a1.sinks.k1.kafka.producer.acks = 1

a1.sinks.k1.channel = c1

# event-sink

a1.sinks.k2.type = org.apache.flume.sink.kafka.KafkaSink

a1.sinks.k2.kafka.topic = topic\_event

a1.sinks.k2.kafka.bootstrap.servers = hadoop102:9092,hadoop103:9092,hadoop104:9092

a1.sinks.k2.kafka.flumeBatchSize = 2000

a1.sinks.k2.kafka.producer.acks = 1

a1.sinks.k2.channel = c2

注意：com.byf.flume.interceptor.LogETLInterceptor和com.byf.flume.interceptor.LogTypeInterceptor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。



### 4.3.3 Flume拦截器

本项目中自定义了两个拦截器，分别是：ETL拦截器、日志类型区分拦截器。

ETL拦截器主要用于，过滤时间戳不合法和json数据不完整的日志

日志类型区分拦截器主要用于，将启动日志和事件日志区分开来，方便发往Kafka的不同topic。

1）创建Maven工程flume-interceptor

2）创建包名：com.byf.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**>1.7.0</**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**>

4）在com.byf.flume.interceptor包下创建LogETLInterceptor类名

Flume ETL拦截器LogETLInterceptor

**package** com.byf.flume.interceptor;  
  
**import** org.apache.flume.Context;  
**import** org.apache.flume.Event;  
**import** org.apache.flume.interceptor.Interceptor;  
**import** java.nio.charset.Charset;  
**import** java.util.ArrayList;  
**import** java.util.List;  
  
**public class** LogETLInterceptor **implements** Interceptor {  
  
 @Override  
 **public void** initialize() {  
  
 }  
  
 @Override  
 **public** Event intercept(Event event) {  
  
 String log = **new** String(event.getBody(), Charset.*forName*(**"UTF-8"**));  
  
 *// body为原始数据，newBody为处理后的数据,判断是否为display的数据类型* **if** (LogUtils.*validate*(log)) {  
 **return** event;  
 }  
  
 **return null**;  
 }  
  
 @Override  
 **public** List<Event> intercept(List<Event> events) {  
  
 ArrayList<Event> intercepts = new ArrayList<>();

// 遍历所有Event，将拦截器校验不合格的过滤掉

for (Event event : events) {

Event interceptEvent = intercept(event);

if (interceptEvent != null){

intercepts.add(interceptEvent);

}

}

return intercepts;  
 }  
  
 @Override  
 **public void** close() {  
  
 }  
  
 **public static class** Builder **implements** Interceptor.Builder {  
  
 **public** Interceptor build() {  
 **return new** LogETLInterceptor();  
 }  
  
  
 @Override  
 **public void** configure(Context context) {  
  
 }  
 }  
}

4）Flume日志过滤工具类

**package** com.byf.flume.interceptor;  
  
**import** org.apache.commons.lang.StringUtils;  
**import** org.apache.commons.lang.math.NumberUtils;  
**import** org.slf4j.Logger;  
**import** org.slf4j.LoggerFactory;  
  
**public class** LogUtils {  
  
 **private static** Logger *logger* = LoggerFactory.*getLogger*(LogUtils.**class**);  
  
 */\*\*  
 \* 日志检查，正常的log会返回true，错误的log返回false  
 \*  
 \** ***@param log*** *\*/* **public static boolean** validate(String log) {  
  
 **try** {  
*// 日志的格式是：时间戳| json串  
// 1549696569054 | {"cm":{"ln":"-89.2","sv":"V2.0.4","os":"8.2.0","g":"M67B4QYU@gmail.com","nw":"4G","l":"en","vc":"18","hw":"1080\*1920","ar":"MX","uid":"u8678","t":"1549679122062","la":"-27.4","md":"sumsung-12","vn":"1.1.3","ba":"Sumsung","sr":"Y"},"ap":"weather","et":[]}* String[] logArray = log.split(**"\\|"**);

**if** (logArray.**length** != 2) {  
 **return false**;  
 }  
*// 检查第一串是否为时间戳 或者不是全数字* **if** (logArray[0].length() != 13 || !NumberUtils.*isDigits*(logArray[0])) {  
 **return false**;  
 }  
  
*// 第二串是否为正确的json,这里我们就粗略的检查了，有时候我们需要从后面来发现json传错的数据，做分析* **if** (!logArray[1].trim().startsWith(**"{"**) || !logArray[1].trim().endsWith(**"}"**)) {  
 **return false**;  
 }  
 } **catch** (Exception e) {  
*// 错误日志打印，需要查看  
 logger*.error(**"error message is:"** + log);  
 *logger*.error(e.getMessage());  
  
 **return false**;  
 }  
  
 **return true**;  
 }  
}

5）Flume日志类型区分拦截器LogTypeInterceptor

**package** com.byf.flume.interceptor;  
  
**import** org.apache.flume.Context;  
**import** org.apache.flume.Event;  
**import** org.apache.flume.interceptor.Interceptor;  
  
**import** java.util.List;  
**import** java.util.Map;  
  
*/\*\*  
 \* Created by Administrator on 2019/1/18 0018.  
 \*/***public class** LogTypeInterceptor **implements** Interceptor {  
  
 @Override  
 **public void** initialize() {  
  
 }  
  
 @Override  
 **public** Event intercept(Event event) {  
  
 *// 1 将json数组转换为字符串* String jsonStr = **new** String(event.getBody(),Charset.forName("UTF-8"));  
  
 String logType = **""** ;

*// 2 区分日志类型：startLog*  
 **if** (jsonStr.contains(**"start"**)) {  
 logType = **"start"**;  
 }  
 *// eventLog* **else** {  
 logType = **"event"**;  
 }

*// 3获取flume接收消息头* Map<String, String> headers = event.getHeaders();  
 *// 4将日志类型存储到flume头中* headers.put(**"logType"**, logType);  
  
 **return** event;  
 }  
  
 @Override  
 **public** List<Event> intercept(List<Event> events) {  
  
 ArrayList<Event> interceptors = new ArrayList<>();

for (Event event : events) {

Event interceptEvent = intercept(event);

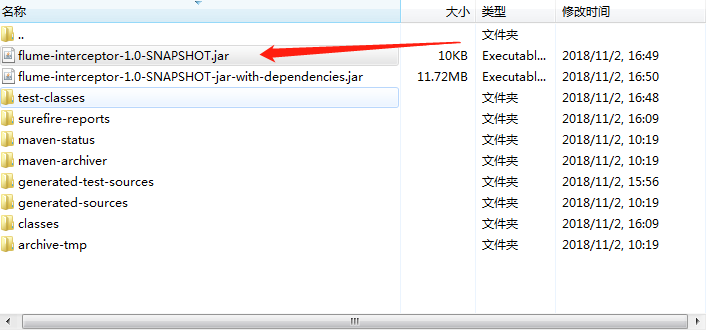
interceptors.add(interceptEvent);

}

return interceptors;  
 }  
  
 @Override  
 **public void** close() {  
  
 }  
  
 **public static class** Builder **implements** Interceptor.Builder {  
  
 **public** Interceptor build() {  
 **return new** LogTypeInterceptor();  
 }  
  
 @Override  
 **public void** configure(Context context) {  
  
 }  
 }  
}

6）打包

拦截器打包之后，只需要单独包，不需要将依赖的包上传。打包之后要放入flume的lib文件夹下面。



注意：为什么不需要依赖包？因为依赖包在flume的lib目录下面已经存在了。

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

[byf@hadoop102 lib]$ ls | grep interceptor

flume-interceptor-1.0-SNAPSHOT.jar

8）分发flume到hadoop103、hadoop104

[byf@hadoop102 module]$ xsync flume/

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

### 4.3.4 日志采集Flume启动停止脚本

1）在/home/byf/bin目录下创建脚本f1.sh

[byf@hadoop102 bin]$ vim f1.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 >/dev/null 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 kill"

done

};;

esac

说明：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。

/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

2）增加脚本执行权限

[byf@hadoop102 bin]$ chmod 777 f1.sh

3）f1集群启动脚本

[byf@hadoop102 module]$ f1.sh start

4）f1集群停止脚本

[byf@hadoop102 module]$ f1.sh stop

## 4.4 Kafka安装



集群规划

|  |  |  |  |
| --- | --- | --- | --- |
|  | 服务器hadoop102 | 服务器hadoop103 | 服务器hadoop104 |
| Kafka | Kafka | Kafka | Kafka |

详见：Kafka



### 4.4.1 Kafka集群启动停止脚本

1）在/home/byf/bin目录下创建脚本kf.sh

[byf@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-------"

# 用于KafkaManager监控

ssh $i "export JMX\_PORT=9988 && /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

注意：启动Kafka时要先开启JMX端口，是用于后续KafkaManager监控。

2）增加脚本执行权限

[byf@hadoop102 bin]$ chmod 777 kf.sh

3）kf集群启动脚本

[byf@hadoop102 module]$ kf.sh start

4）kf集群停止脚本

[byf@hadoop102 module]$ kf.sh stop

### 4.4.2 查看所有Kafka Topic

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

### 4.4.3 创建 Kafka Topic

进入到/opt/module/kafka/目录下分别创建：启动日志主题、事件日志主题。

1）创建启动日志主题

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

2）创建事件日志主题

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

### 4.4.4 删除 Kafka Topic

[byf@hadoop102 kafka]$ bin/kafka-topics.sh --delete --zookeeper hadoop102:2181,hadoop103:2181,hadoop104:2181 --topic topic\_start

[byf@hadoop102 kafka]$ bin/kafka-topics.sh --delete --zookeeper hadoop102:2181,hadoop103:2181,hadoop104:2181 --topic topic\_event

### 4.4.5 生产消息

[byf@hadoop102 kafka]$ bin/kafka-console-producer.sh \

--broker-list hadoop102:9092 --topic topic\_start

>hello world

>byf byf

### 4.4.6 消费消息

[byf@hadoop103 kafka]$ bin/kafka-console-consumer.sh \

--zookeeper hadoop102:2181 --from-beginning --topic topic\_start

--from-beginning：会把first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。

### 4.4.7 查看某个Topic的详情

[byf@hadoop102 kafka]$ bin/kafka-topics.sh --zookeeper hadoop102:2181 \

--describe --topic topic\_start

### 4.4.8 Kafka Manager安装

Kafka Manager是yahoo的一个Kafka监控管理项目。

1）下载地址

<https://github.com/yahoo/kafka-manager>

下载之后编译源码，编译完成后，拷贝出：kafka-manager-1.3.3.22.zip

2）拷贝kafka-manager-1.3.3.22.zip到hadoop102的/opt/module目录

[byf@hadoop102 module]$ pwd

/opt/module

3）解压kafka-manager-1.3.3.22.zip到/opt/module目录

[byf@hadoop102 module]$ unzip kafka-manager-1.3.3.22.zip

4）进入到/opt/module/kafka-manager-1.3.3.22/conf目录，在application.conf文件中修改kafka-manager.zkhosts

[byf@hadoop102 conf]$ vim application.conf

修改为：

kafka-manager.zkhosts="hadoop102:2181,hadoop103:2181,hadoop104:2181"

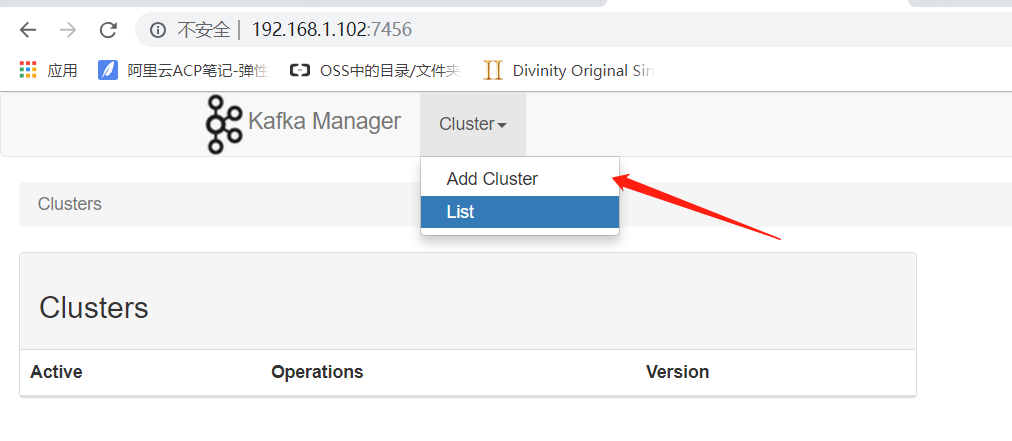
5）启动KafkaManager

[byf@hadoop102 kafka-manager-1.3.3.22]$

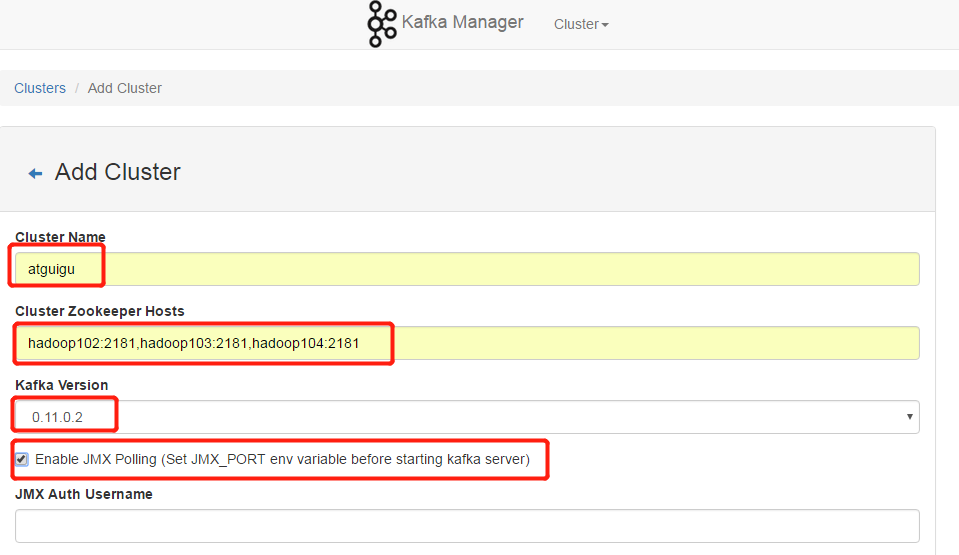
nohup bin/kafka-manager -Dhttp.port=7456 >/opt/module/kafka-manager-1.3.3.22/start.log 2>&1 &

6）在浏览器中打开

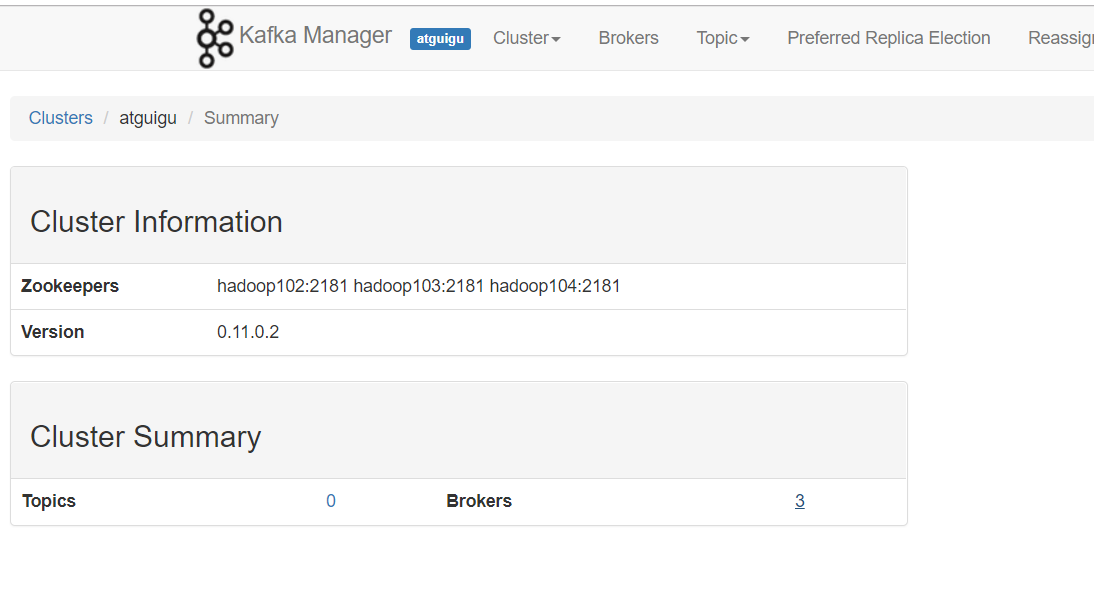
<http://hadoop102:7456>



可以看到这个界面，选择添加 cluster；



我们要配置好Zookeeper的Hosts，Cluster的Name，Kafka的版本，点确定。



至此，就可以查看整个Kafka集群的状态，包括：Topic的状态、Brokers的状态、Cosumer的状态。

在Kafka的/opt/module/kafka-manager-1.3.3.22/application.home\_IS\_UNDEFINED 目录下面，可以看到Kafka-Manager的日志。

### 4.4.9 Kafka Manager启动停止脚本

1）在/home/byf/bin目录下创建脚本km.sh

[byf@hadoop102 bin]$ vim km.sh

在脚本中填写如下内容

#! /bin/bash

case $1 in

"start"){

echo " -------- 启动 KafkaManager -------"

nohup /opt/module/kafka-manager-1.3.3.22/bin/kafka-manager -Dhttp.port=7456 >start.log 2>&1 &

};;

"stop"){

echo " -------- 停止 KafkaManager -------"

ps -ef | grep ProdServerStart | grep -v grep |awk '{print $2}' | xargs kill

};;

esac

2）增加脚本执行权限

[byf@hadoop102 bin]$ chmod 777 km.sh

3）km集群启动脚本

[byf@hadoop102 module]$ km.sh start

4）km集群停止脚本

[byf@hadoop102 module]$ km.sh stop

### 4.4.10 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压测时，可以查看到哪个地方出现了瓶颈（CPU，内存，网络IO）。一般都是网络IO达到瓶颈。

kafka-consumer-perf-test.sh

kafka-producer-perf-test.sh

2）Kafka Producer压力测试

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

[byf@hadoop102 kafka]$ bin/kafka-producer-perf-test.sh --topic test --record-size 100 --num-records 100000 --throughput 1000 --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。

（2）Kafka会打印下面的信息

5000 records sent, 999.4 records/sec (0.10 MB/sec), 1.9 ms avg latency, 254.0 max latency.

5002 records sent, 1000.4 records/sec (0.10 MB/sec), 0.7 ms avg latency, 12.0 max latency.

5001 records sent, 1000.0 records/sec (0.10 MB/sec), 0.8 ms avg latency, 4.0 max latency.

5000 records sent, 1000.0 records/sec (0.10 MB/sec), 0.7 ms avg latency, 3.0 max latency.

5000 records sent, 1000.0 records/sec (0.10 MB/sec), 0.8 ms avg latency, 5.0 max latency.

参数解析：本例中一共写入10w条消息，每秒向Kafka写入了**0.10MB**的数据，平均是1000条消息/秒，每次写入的平均延迟为0.8毫秒，最大的延迟为254毫秒。

3）Kafka Consumer压力测试

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。

[byf@hadoop102 kafka]$

bin/kafka-consumer-perf-test.sh --zookeeper hadoop102:2181 --topic test --fetch-size 10000 --messages 10000000 --threads 1

参数说明：

--zookeeper 指定zookeeper的链接信息

--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

测试结果说明：

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

2019-02-19 20:29:07:566, **2019-02-19 20:29:12:170,** 9.5368, **2.0714,** 100010, **21722.4153**

**开始测试时间，测试结束数据，最大吞吐率**9.5368MB/s，平均每秒消费**2.0714MB/s，最大每秒消费**100010条，平均每秒消费**21722.4153条。**

### 4.4.11 Kafka机器数量计算

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

先要预估一天大概产生多少数据，然后用Kafka自带的生产压测（只测试Kafka的写入速度，保证数据不积压），计算出峰值生产速度。再根据设定的副本数，就能预估出需要部署Kafka的数量。

比如我们采用压力测试测出写入的速度是10M/s一台，峰值的业务数据的速度是50M/s。副本数为2。

Kafka机器数量=2\*（50\*2/100）+1=3台

## 4.5 Flume消费Kafka数据写到HDFS



集群规划

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

### 4.5.1 日志消费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

[byf@hadoop102 conf]$ vim kafka-flume-hdfs.conf

在文件配置如下内容

## 组件

a1.sources=r1 r2

a1.channels=c1 c2

a1.sinks=k1 k2

## 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\_start

## source2

a1.sources.r2.type = org.apache.flume.source.kafka.KafkaSource

a1.sources.r2.batchSize = 5000

a1.sources.r2.batchDurationMillis = 2000

a1.sources.r2.kafka.bootstrap.servers = hadoop102:9092,hadoop103:9092,hadoop104:9092

a1.sources.r2.kafka.topics=topic\_event

## channel1

a1.channels.c1.type=memory

a1.channels.c1.capacity=100000

a1.channels.c1.transactionCapacity=10000

## channel2

a1.channels.c2.type=memory

a1.channels.c2.capacity=100000

a1.channels.c2.transactionCapacity=10000

## sink1

a1.sinks.k1.type = hdfs

a1.sinks.k1.hdfs.path = /origin\_data/gmall/log/topic\_start/%Y-%m-%d

a1.sinks.k1.hdfs.filePrefix = logstart-

a1.sinks.k1.hdfs.round = true

a1.sinks.k1.hdfs.roundValue = 30

a1.sinks.k1.hdfs.roundUnit = second

##sink2

a1.sinks.k2.type = hdfs

a1.sinks.k2.hdfs.path = /origin\_data/gmall/log/topic\_event/%Y-%m-%d

a1.sinks.k2.hdfs.filePrefix = logevent-

a1.sinks.k2.hdfs.round = true

a1.sinks.k2.hdfs.roundValue = 30

a1.sinks.k2.hdfs.roundUnit = second

## 不要产生大量小文件

a1.sinks.k1.hdfs.rollInterval = 10

a1.sinks.k1.hdfs.rollSize = 134217728

a1.sinks.k1.hdfs.rollCount = 0

a1.sinks.k2.hdfs.rollInterval = 10

a1.sinks.k2.hdfs.rollSize = 134217728

a1.sinks.k2.hdfs.rollCount = 0

## 控制输出文件是原生文件。

a1.sinks.k1.hdfs.fileType = CompressedStream

a1.sinks.k2.hdfs.fileType = CompressedStream

a1.sinks.k1.hdfs.codeC = lzop

a1.sinks.k2.hdfs.codeC = lzop

## 拼装

a1.sources.r1.channels = c1

a1.sinks.k1.channel= c1

a1.sources.r2.channels = c2

a1.sinks.k2.channel= c2



### 4.5.2 Flume异常处理

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

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

### 4.5.3 日志消费Flume启动停止脚本

1）在/home/byf/bin目录下创建脚本f2.sh

[byf@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/log.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 kill"

done

};;

esac

2）增加脚本执行权限

[byf@hadoop102 bin]$ chmod 777 f2.sh

3）f2集群启动脚本

[byf@hadoop102 module]$ f2.sh start

4）f2集群停止脚本

[byf@hadoop102 module]$ f2.sh stop

## 4.6 采集通道启动/停止脚本

1）在/home/byf/bin目录下创建脚本cluster.sh

[byf@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 -------"

echo " -------- 启动 hadoop集群 -------"

/opt/module/hadoop-2.7.2/sbin/start-dfs.sh

ssh hadoop103 "/opt/module/hadoop-2.7.2/sbin/start-yarn.sh"

#启动 Zookeeper集群

zk.sh start

#启动 Flume采集集群

f1.sh start

#启动 Kafka采集集群

kf.sh start

sleep 4s;

#启动 Flume消费集群

f2.sh start

#启动 KafkaManager

km.sh start

};;

"stop"){

echo " -------- 停止 集群 -------"

#停止 KafkaManager

km.sh stop

#停止 Flume消费集群

f2.sh stop

#停止 Kafka采集集群

kf.sh stop

sleep 4s;

#停止 Flume采集集群

f1.sh stop

#停止 Zookeeper集群

zk.sh stop

echo " -------- 停止 hadoop集群 -------"

ssh hadoop103 "/opt/module/hadoop-2.7.2/sbin/stop-yarn.sh"

/opt/module/hadoop-2.7.2/sbin/stop-dfs.sh

};;

esac

2）增加脚本执行权限

[byf@hadoop102 bin]$ chmod 777 cluster.sh

3）cluster集群启动脚本

[byf@hadoop102 module]$ cluster.sh start

4）cluster集群停止脚本

[byf@hadoop102 module]$ cluster.sh stop

# 第5章 总结

## 5.1 数仓概念总结

**1）数据仓库的输入数据源和输出系统分别是什么？**

输入系统：埋点产生的用户行为数据、JavaEE后台产生的业务数据。

输出系统：报表系统、用户画像系统、推荐系统

## 5.2 项目需求及架构总结

**1）技术选型注意事项**

至少给出两套方案：一套是咱们讲过的，一套是听说过的。并列举出两套框架的优缺点。重点推讲过的框架。如果领导让你用没讲过的框架，记得申请研发时间。

**2）在公司技术分享时要画比较高大上的系统架构图**

**3）系统数据流程图在开发中一定要画，可以清楚的看见数据的走向，从宏观上把握整个项目**

**4）如果是Apache版本的大数据框架，一定列出程序的版本号，考虑一下版本的兼容性问题。CDH版本是5.1.2。**

**5）集群资源规划一定要做，主要考虑一下：内存大的尽量分开。**

## 5.3 数据采集模块总结

### 5.3.1 Hadoop相关总结

**1）Hadoop默认不支持LZO压缩，如果需要支持LZO压缩，需要添加jar包，并在hadoop的cores-site.xml文件中添加相关压缩配置。**

2）Hadoop常用端口号

3）Hadoop配置文件以及简单的Hadoop集群搭建

4）HDFS读流程和写流程

5）MapReduce的Shuffle过程及Hadoop优化

6）Yarn的Job提交流程

7）Yarn的默认调度器、调度器分类、以及他们之间的区别

### 5.3.2 Zookeeper相关总结

1）选举机制

半数机制

2）常用命令

ls get create

### 5.3.3 Flume相关总结

1）Flume组成，Put事务，Take事务

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。

File Channel：数据存储在磁盘，宕机数据可以保存。但是传输速率慢。适合对数据传输可靠性要求高的场景，比如，金融行业。

Memory Channel：数据存储在内存中，宕机数据丢失。传输速率快。适合对数据传输可靠性要求不高的场景，比如，普通的日志数据。

Kafka Sink：

Source到Channel是Put事务

Channel到Sink是Take事务

2）Flume拦截器

（1）拦截器注意事项

项目中自定义了：ETL拦截器和区分类型拦截器。

采用两个拦截器的优缺点：优点，模块化开发和可移植性；缺点，性能会低一些

（2）自定义拦截器步骤

a）实现 Interceptor

b）重写四个方法

* initialize 初始化
* public Event intercept(Event event) 处理单个Event
* public List<Event> intercept(List<Event> events) 处理多个Event，在这个方法中调用Event intercept(Event event)
* close 方法

c）静态内部类，实现Interceptor.Builder

3）Flume Channel选择器

4）Flume 监控器

Ganglia

5）Flume采集数据会丢失吗?

不会，Channel存储可以存储在File中，数据传输自身有事务。

## 5.4 Kafka相关总结



1）Kafka压测

Kafka官方自带压力测试脚本（kafka-consumer-perf-test.sh、kafka-producer-perf-test.sh）。Kafka压测时，可以查看到哪个地方出现了瓶颈（CPU，内存，网络IO）。一般都是网络IO达到瓶颈。

2）Kafka的机器数量

Kafka机器数量=2\*（峰值生产速度\*副本数/100）+1

3）Kafka的日志保存时间

7天

4）Kafka的硬盘大小

每天的数据量\*7天

5）Kafka监控

公司自己开发的监控器；

开源的监控器：KafkaManager、Kafkamonitor

6）Kakfa 分区数。

分区数并不是越多越好，一般分区数不要超过集群机器数量。分区数越多占用内存越大（ISR等），一个节点集中的分区也就越多，当它宕机的时候，对系统的影响也就越大。

分区数一般设置为：3-10个

7）副本数设定

一般我们设置成2个或3个，很多企业设置为2个。

8）多少个Topic

通常情况：多少个日志类型就多少个Topic。也有对日志类型进行合并的。

9）Kafka丢不丢数据

Ack=0，相当于异步发送，消息发送完毕即offset增加，继续生产。

Ack=1，leader收到leader replica 对一个消息的接受ack才增加offset，然后继续生产。

Ack=-1，leader收到所有replica 对一个消息的接受ack才增加offset，然后继续生产。

10）Kafka的ISR副本同步队列

ISR（In-Sync Replicas），副本同步队列。ISR中包括Leader和Follower。如果Leader进程挂掉，会在ISR队列中选择一个服务作为新的Leader。有replica.lag.max.messages（延迟条数）和replica.lag.time.max.ms（延迟时间）两个参数决定一台服务是否可以加入ISR副本队列，在0.10版本移除了replica.lag.max.messages参数，防止服务频繁的进去队列。

任意一个维度超过阈值都会把Follower剔除出ISR，存入OSR（Outof-Sync Replicas）列表，新加入的Follower也会先存放在OSR中。

11）Kafka分区分配策略

在 Kafka内部存在两种默认的分区分配策略：Range和 RoundRobin。

Range 是默认策略。Range是对每个Topic而言的（即一个Topic一个Topic分），首先对同一个Topic里面的分区按照序号进行排序，并对消费者按照字母顺序进行排序。然后用Partitions分区的个数除以消费者线程的总数来决定每个消费者线程消费几个分区。如果除不尽，那么前面几个消费者线程将会多消费一个分区。

例如：我们有10个分区，两个消费者（C1，C2），3个消费者线程，10 / 3 = 3而且除不尽。

C1-0 将消费 0, 1, 2, 3 分区

C2-0 将消费 4, 5, 6 分区

C2-1 将消费 7, 8, 9 分区

RoundRobin：前提：同一个Consumer Group里面的所有消费者的num.streams（消费者消费线程数）必须相等；每个消费者订阅的主题必须相同。

第一步：将所有主题分区组成TopicAndPartition列表，然后对TopicAndPartition列表按照hashCode进行排序，最后按照轮询的方式发给每一个消费线程。

12）Kafka中数据量计算

每天总数据量10g，每天产生1000万条日志， 1000万/24/60/60=115条/每秒钟

每平均秒钟：115条

低谷每秒钟：40条

高峰每秒钟：115条\*（2-20倍）=230条-2300条

每条日志大小：0.5k-2k

每秒多少数据量：230k-2MB

13） Kafka挂掉

（1）Flume记录

（2）日志有记录

（3）短期没事