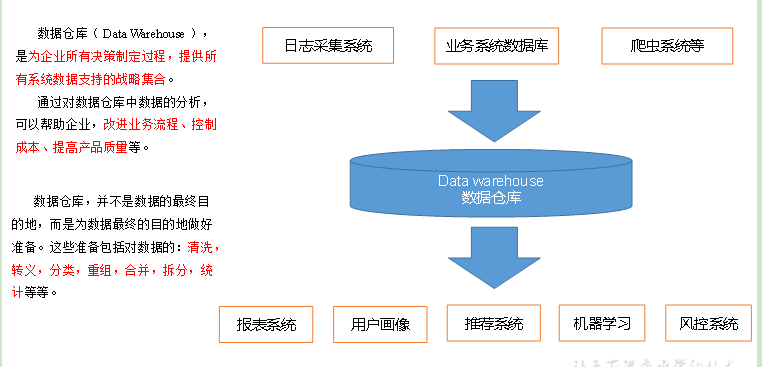
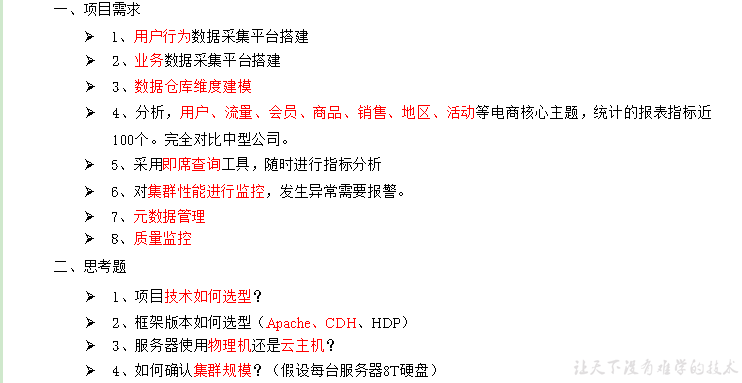
用户行为数据采集

# 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

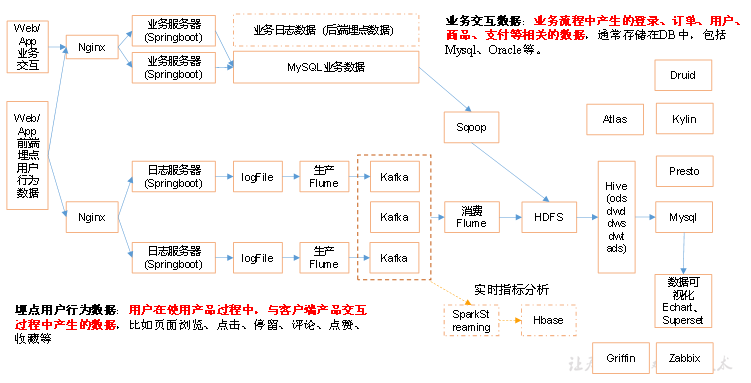


## 2.2 项目框架

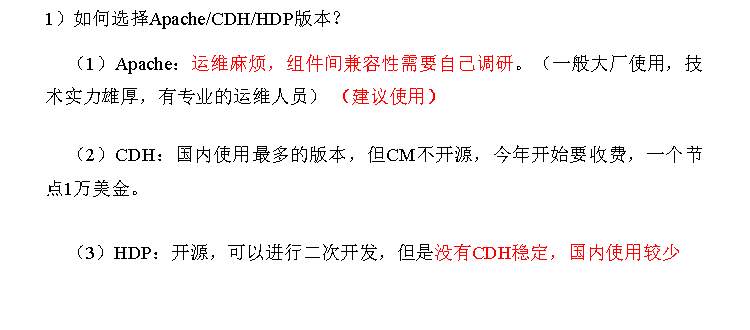
### 2.2.1 技术选型



### 2.2.2 系统数据流程设计

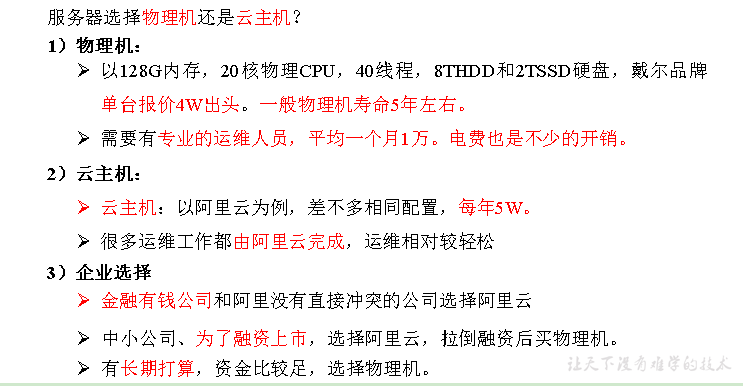


### 2.2.3 框架版本选型

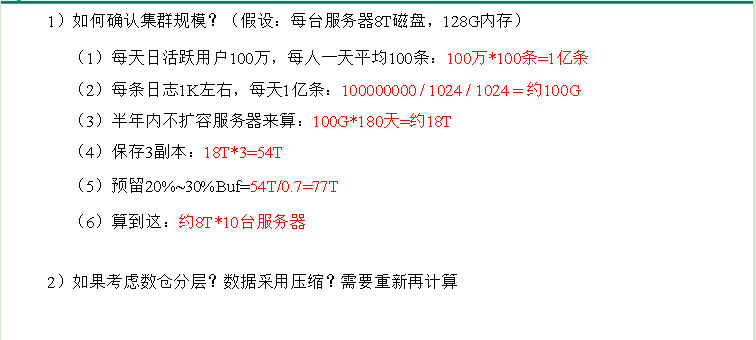




### 服务器选型



### 2.2.5 集群资源规划设计



2）测试集群服务器规划

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 服务名称 | 子服务 | 服务器  hadoop102 | 服务器  hadoop103 | 服务器  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 | √ |  |  |
| Druid | Druid | √ | √ | √ |
| Kylin |  | √ |  |  |
| Hbase | HMaster | √ |  |  |
| HRegionServer | √ | √ | √ |
| Superset |  | √ |  |  |
| Atlas |  | √ |  |  |
| Solr | Jar | √ |  |  |
| Griffin |  | √ |  |  |
| 服务数总计 |  | 19 | 9 | 9 |

# 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",//项目数据来源 app pc

"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) language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) area区域

**"md": "BBB100-1", // (String) model手机型号**

**"ba": "blackberry", // (String) brand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 | |
| 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： 广告展示=1 广告点击=2 |
| contentType | Type: 1 商品 2 营销活动 |
| displayMills | 展示时长 毫秒数 |
| itemId | 商品id |
| activityId | 营销活动id |



### 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 |

### 3.2.7 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 |



### 3.2.8 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 |  | √ |  |

### 3.2.9 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 |  | √ |  |



### 3.2.10 错误日志

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

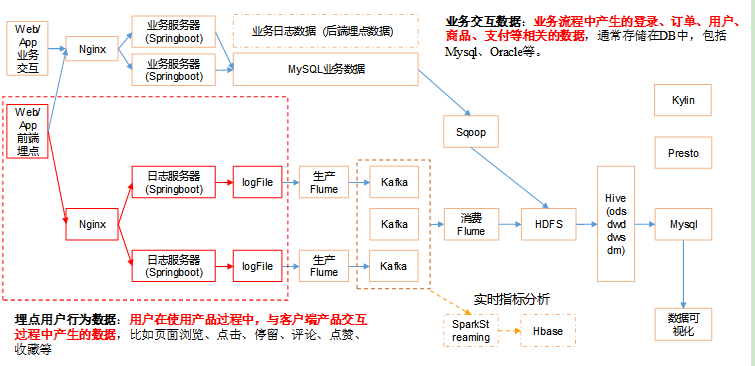
## 3.3 启动日志数据

事件标签: start(我的理解start下面的字段和上面的事件是一样的,存在于事件的列表3.1)

|  |  |
| --- | --- |
| 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） |
| en | 日志类型start |

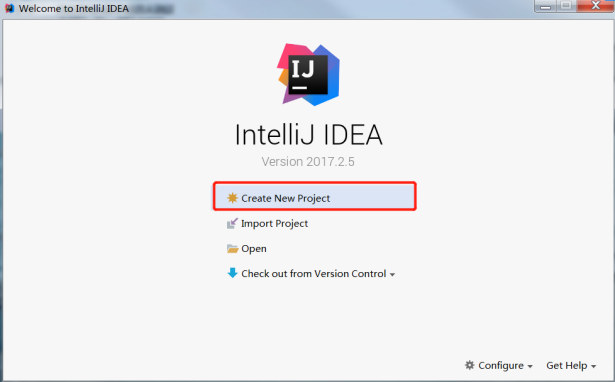
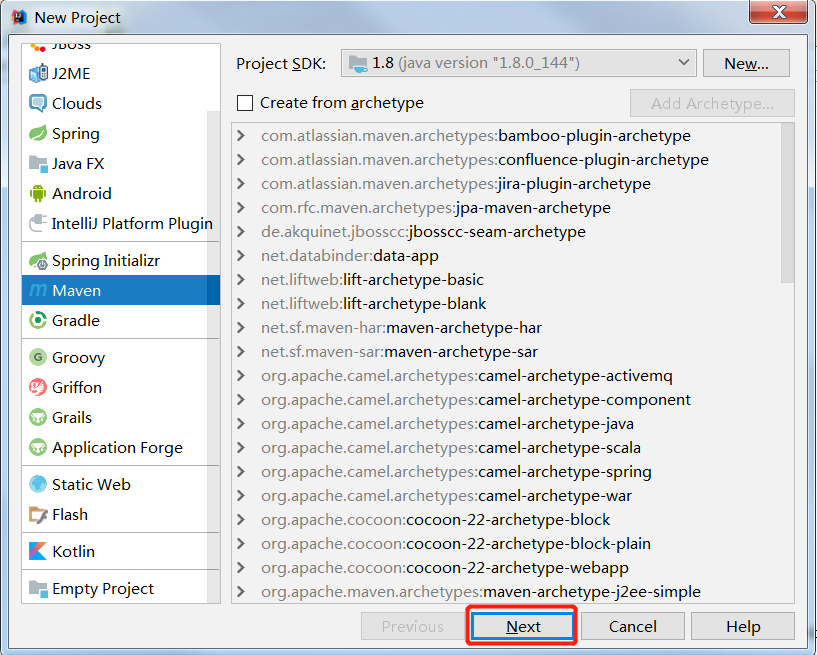
{  
    "action":"1",  
    "ar":"MX",  
    "ba":"HTC",  
    "detail":"",  
    "en":"start",  
    "entry":"2",  
    "extend1":"",  
    "g":"43R2SEQX@gmail.com",  
    "hw":"640\*960",  
    "l":"en",  
    "la":"20.4",  
    "ln":"-99.3",  
    "loading\_time":"2",  
    "md":"HTC-2",  
    "mid":"995",  
    "nw":"4G",  
    "open\_ad\_type":"2",  
    "os":"8.1.2",  
    "sr":"B",  
    "sv":"V2.0.6",  
    "t":"1561472502444",  
    "uid":"995",  
    "vc":"10",  
    "vn":"1.3.4"  
}

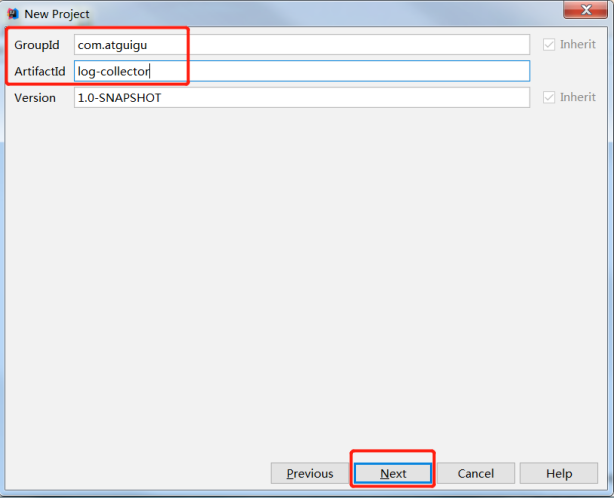
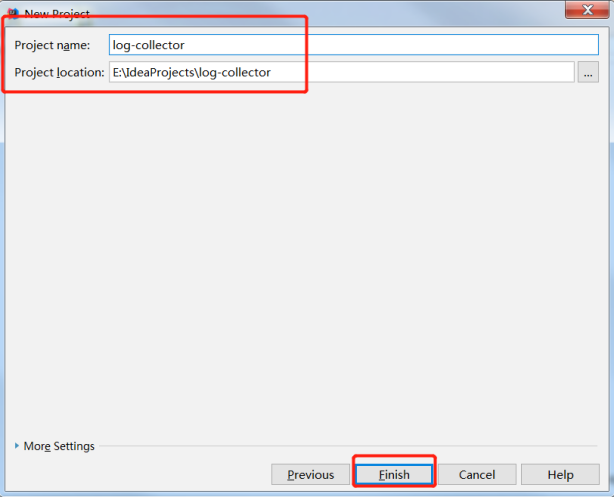
## 3.4 数据生成脚本



### 3.4.1 创建Maven工程

1）创建log-collector

2）创建一个包名：com.woaini.appclient

3）在com.woaini.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**>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.woaini.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.woaini.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.woaini.bean

2）在com.woaini.bean包下依次创建如下bean对象

**package** com.woaini.bean;  
*/\*\*  
 \* 公共日志  
 \*/***public class** AppBase{  
  
 **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.woaini.bean;  
*/\*\*  
 \* 启动日志  
 \*/***public class** AppStart **extends** AppBase {  
  
 **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）* **private** String **en**;*//启动日志类型标记* **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;  
 }  
  
 **public** String getEn() {  
 **return en**;  
 }  
  
 **public void** setEn(String en) {  
 **this**.**en** = en;  
 }  
}

### 3.4.3 错误日志Bean

**package** com.woaini.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.woaini.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.woaini.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.woaini.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.woaini.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： 广告展示=1 广告点击=2* **private** String **contentType**;*//Type: 1 商品 2 营销活动* **private** String **displayMills**;*//展示时长 毫秒数* **private** String **itemId**; *//商品id* **private** String **activityId**; *//营销活动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 getActivityId() {  
 **return activityId**;  
 }  
  
 **public void** setActivityId(String activityId) {  
 **this**.**activityId** = activityId;  
 }  
  
 **public** String getContentType() {  
 **return contentType**;  
 }  
  
 **public void** setContentType(String contentType) {  
 **this**.**contentType** = contentType;  
 }  
  
 **public** String getDisplayMills() {  
 **return displayMills**;  
 }  
  
 **public void** setDisplayMills(String displayMills) {  
 **this**.**displayMills** = displayMills;  
 }  
  
 **public** String getItemId() {  
 **return itemId**;  
 }  
  
 **public void** setItemId(String itemId) {  
 **this**.**itemId** = itemId;  
 }  
}

### 3.4.8 事件日志Bean之消息通知

**package** com.woaini.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之用户后台活跃

**package** com.woaini.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;  
 }  
}

### 3.4.10 事件日志Bean之用户评论

**package** com.woaini.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;  
 }  
}

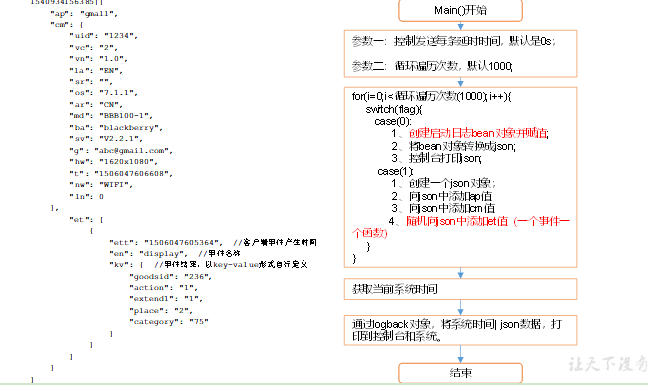
### 3.4.11 事件日志Bean之用户收藏

**package** com.woaini.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;  
 }  
}

### 3.4.12 事件日志Bean之用户点赞

**package** com.woaini.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;  
 }  
}

### 3.4.13 主函数



在AppMain类中添加如下内容：

**package** com.woaini.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.woaini.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**);  
 **private static** Random *rand* = **new** Random();  
  
 *// 设备id* **private static int** *s\_mid* = 0;  
  
 *// 用户id* **private static int** *s\_uid* = 0;  
  
 *// 商品id* **private static int** *s\_goodsid* = 0;  
  
 **public static void** main(String[] args) {  
  
 *// 参数一：控制发送每条的延时时间，默认是0* Long delay = args.**length** > 0 ? Long.*parseLong*(args[0]) : 0L;  
  
 *// 参数二：循环遍历次数* **int** loop\_len = args.**length** > 1 ? Integer.*parseInt*(args[1]) : 1000;  
  
 *// 生成数据  
 generateLog*(delay, loop\_len);  
 }  
  
 **private static void** generateLog(Long delay, **int** loop\_len) {  
  
 **for** (**int** i = 0; i < loop\_len; i++) {  
  
 **int** flag = *rand*.nextInt(2);  
  
 **switch** (flag) {  
 **case** (0):  
 *//应用启动* AppStart appStart = *generateStart*();  
 String jsonString = JSON.*toJSONString*(appStart);  
  
 *//控制台打印* ***logger***.info(jsonString);  
 **break**;  
  
 **case** (1):  
  
 JSONObject json = **new** JSONObject();  
  
 json.put(**"ap"**, **"app"**);  
 json.put(**"cm"**, *generateComFields*());  
  
 JSONArray eventsArray = **new** JSONArray();  
  
 *// 事件日志  
 // 商品点击，展示* **if** (*rand*.nextBoolean()) {  
 eventsArray.add(*generateDisplay*());  
 json.put(**"et"**, eventsArray);  
 }  
  
 *// 商品详情页* **if** (*rand*.nextBoolean()) {  
 eventsArray.add(*generateNewsDetail*());  
 json.put(**"et"**, eventsArray);  
 }  
  
 *// 商品列表页* **if** (*rand*.nextBoolean()) {  
 eventsArray.add(*generateNewList*());  
 json.put(**"et"**, eventsArray);  
 }  
  
 *// 广告* **if** (*rand*.nextBoolean()) {  
 eventsArray.add(*generateAd*());  
 json.put(**"et"**, eventsArray);  
 }  
  
 *// 消息通知* **if** (*rand*.nextBoolean()) {  
 eventsArray.add(*generateNotification*());  
 json.put(**"et"**, eventsArray);  
 }  
  
 *// 用户后台活跃* **if** (*rand*.nextBoolean()) {  
 eventsArray.add(*generateBackground*());  
 json.put(**"et"**, eventsArray);  
 }  
  
 *//故障日志* **if** (*rand*.nextBoolean()) {  
 eventsArray.add(*generateError*());  
 json.put(**"et"**, eventsArray);  
 }  
  
 *// 用户评论* **if** (*rand*.nextBoolean()) {  
 eventsArray.add(*generateComment*());  
 json.put(**"et"**, eventsArray);  
 }  
  
 *// 用户收藏* **if** (*rand*.nextBoolean()) {  
 eventsArray.add(*generateFavorites*());  
 json.put(**"et"**, eventsArray);  
 }  
  
 *// 用户点赞* **if** (*rand*.nextBoolean()) {  
 eventsArray.add(*generatePraise*());  
 json.put(**"et"**, eventsArray);  
 }  
  
 *//时间* **long** millis = System.*currentTimeMillis*();  
  
 *//控制台打印* ***logger***.info(millis + **"|"** + json.toJSONString());  
 **break**;  
 }  
  
 *// 延迟* **try** {  
 Thread.*sleep*(delay);  
 } **catch** (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
  
 */\*\*  
 \* 公共字段设置  
 \*/* **private 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) + **""**);  
  
 **return** (JSONObject) JSON.*toJSON*(appBase);  
 }  
  
 */\*\*  
 \* 商品展示事件  
 \*/* **private static** JSONObject generateDisplay() {  
  
 AppDisplay appDisplay = **new** AppDisplay();  
  
 **boolean** boolFlag = *rand*.nextInt(10) < 7;  
  
 *// 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  
 \*/* **private 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  
 \*/* **private 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  
 \*/* **private 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 + **""**);  
  
 *// 内容类型类型* flag = *rand*.nextInt(6)+1;  
 appAd.setContentType(flag+ **""**);  
  
 *// 展示样式* flag = *rand*.nextInt(120000)+1000;  
 appAd.setDisplayMills(flag+**""**);  
  
 flag=*rand*.nextInt(1);  
 **if**(flag==1){  
 appAd.setContentType(flag+**""**);  
 flag =*rand*.nextInt(6);  
 appAd.setItemId(flag+ **""**);  
 }**else**{  
 appAd.setContentType(flag+**""**);  
 flag =*rand*.nextInt(1)+1;  
 appAd.setActivityId(flag+ **""**);  
 }  
  
 JSONObject jsonObject = (JSONObject) JSON.*toJSON*(appAd);  
  
 **return** *packEventJson*(**"ad"**, jsonObject);  
 }  
  
 */\*\*  
 \* 启动日志  
 \*/* **private static** AppStart generateStart() {  
  
 AppStart appStart = **new** AppStart();  
  
 *//设备id* appStart.setMid(*s\_mid* + **""**);  
 *s\_mid*++;  
  
 *// 用户id* appStart.setUid(*s\_uid* + **""**);  
 *s\_uid*++;  
  
 *// 程序版本号 5,6等* appStart.setVc(**""** + *rand*.nextInt(20));  
  
 *//程序版本名 v1.1.1* appStart.setVn(**"1."** + *rand*.nextInt(4) + **"."** + *rand*.nextInt(10));  
  
 *// 安卓系统版本* appStart.setOs(**"8."** + *rand*.nextInt(3) + **"."** + *rand*.nextInt(10));  
  
 *//设置日志类型* appStart.setEn(**"start"**);  
  
 *// 语言 es,en,pt* **int** flag = *rand*.nextInt(3);  
 **switch** (flag) {  
 **case** (0):  
 appStart.setL(**"es"**);  
 **break**;  
 **case** (1):  
 appStart.setL(**"en"**);  
 **break**;  
 **case** (2):  
 appStart.setL(**"pt"**);  
 **break**;  
 }  
  
 *// 渠道号 从哪个渠道来的* appStart.setSr(*getRandomChar*(1));  
  
 *// 区域* flag = *rand*.nextInt(2);  
 **switch** (flag) {  
 **case** 0:  
 appStart.setAr(**"BR"**);  
 **case** 1:  
 appStart.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:  
 appStart.setBa(**"Sumsung"**);  
 appStart.setMd(**"sumsung-"** + *rand*.nextInt(20));  
 **break**;  
 **case** 1:  
 appStart.setBa(**"Huawei"**);  
 appStart.setMd(**"Huawei-"** + *rand*.nextInt(20));  
 **break**;  
 **case** 2:  
 appStart.setBa(**"HTC"**);  
 appStart.setMd(**"HTC-"** + *rand*.nextInt(20));  
 **break**;  
 }  
  
 *// 嵌入sdk的版本* appStart.setSv(**"V2."** + *rand*.nextInt(10) + **"."** + *rand*.nextInt(10));  
 *// gmail* appStart.setG(*getRandomCharAndNumr*(8) + **"@gmail.com"**);  
  
 *// 屏幕宽高 hw* flag = *rand*.nextInt(4);  
 **switch** (flag) {  
 **case** 0:  
 appStart.setHw(**"640\*960"**);  
 **break**;  
 **case** 1:  
 appStart.setHw(**"640\*1136"**);  
 **break**;  
 **case** 2:  
 appStart.setHw(**"750\*1134"**);  
 **break**;  
 **case** 3:  
 appStart.setHw(**"1080\*1920"**);  
 **break**;  
 }  
  
 *// 客户端产生日志时间* **long** millis = System.*currentTimeMillis*();  
 appStart.setT(**""** + (millis - *rand*.nextInt(99999999)));  
  
 *// 手机网络模式 3G,4G,WIFI* flag = *rand*.nextInt(3);  
 **switch** (flag) {  
 **case** 0:  
 appStart.setNw(**"3G"**);  
 **break**;  
 **case** 1:  
 appStart.setNw(**"4G"**);  
 **break**;  
 **case** 2:  
 appStart.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* appStart.setLn((-34 - *rand*.nextInt(83) - *rand*.nextInt(60) / 10.0) + **""**);  
 *// 纬度* appStart.setLa((32 - *rand*.nextInt(85) - *rand*.nextInt(60) / 10.0) + **""**);  
  
 *// 入口* 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**;  
 }  
  
 *// 扩展字段* appStart.setExtend1(**""**);  
  
 **return** appStart;  
 }  
 */\*\*  
 \* 消息通知  
 \*/* **private 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);  
 }  
  
 */\*\*  
 \* 后台活跃  
 \*/* **private 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  
 \*/* **private 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  
 \*/* **private 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  
 \*/* **private static** String getRandomChar(Integer length) {  
  
 StringBuilder str = **new** StringBuilder();  
 Random random = **new** Random();  
  
 **for** (**int** i = 0; i < length; i++) {  
 *// 字符串* str.append((**char**) (65 + random.nextInt(26)));*// 取得大写字母* }  
  
 **return** str.toString();  
 }  
  
 */\*\*  
 \* 获取随机字母数字组合  
 \** ***@param length*** *字符串长度  
 \*/* **private static** String getRandomCharAndNumr(Integer length) {  
  
 StringBuilder str = **new** StringBuilder();  
 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.append((**char**) (65 + random.nextInt(26)));*// 取得大写字母* } **else** { *// 数字* str.append(String.*valueOf*(random.nextInt(10)));  
 }  
 }  
  
 **return** str.toString();  
 }  
  
 */\*\*  
 \* 收藏  
 \*/* **private static** 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  
 \*/* **private static** 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  
 \*/* **private static** 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  
 \*/* **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  
 \*/* **private static** String getCONTENT() {  
  
 StringBuilder str = **new** StringBuilder();  
  
 **for** (**int** i = 0; i < *rand*.nextInt(100); i++) {  
 str.append(*getRandomChar*());  
 }  
  
 **return** str.toString();  
 }  
}

### 3.4.14 配置日志打印Logback

Logback主要用于在磁盘和控制台打印日志。

Logback具体使用：

1）在resources文件夹下创建logback.xml文件。

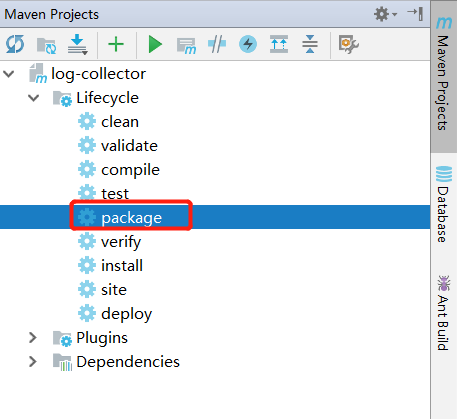
2）在logback.xml文件中填写如下配置

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

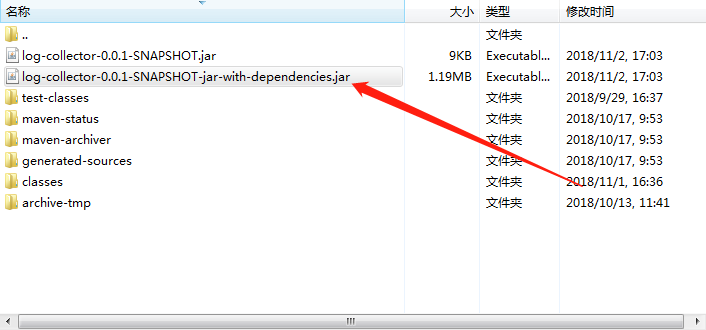
<**configuration debug="false"**>  
 *<!--定义日志文件的存储地址 勿在 LogBack 的配置中使用相对路径 -->* <**property name="LOG\_HOME" value="/tmp/logs/"** />  
 *<!-- 控制台输出 -->* <**appender name="STDOUT"  
 class="ch.qos.logback.core.ConsoleAppender"**>  
 <**encoder  
 class="ch.qos.logback.classic.encoder.PatternLayoutEncoder"**>  
 *<!--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**>%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**>

### 3.4.15 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）后续日志生成过程，在安装完Hadoop和Zookeeper之后执行。

# 第4章 数据采集模块

## 4.1 Hadoop安装

1）集群规划：

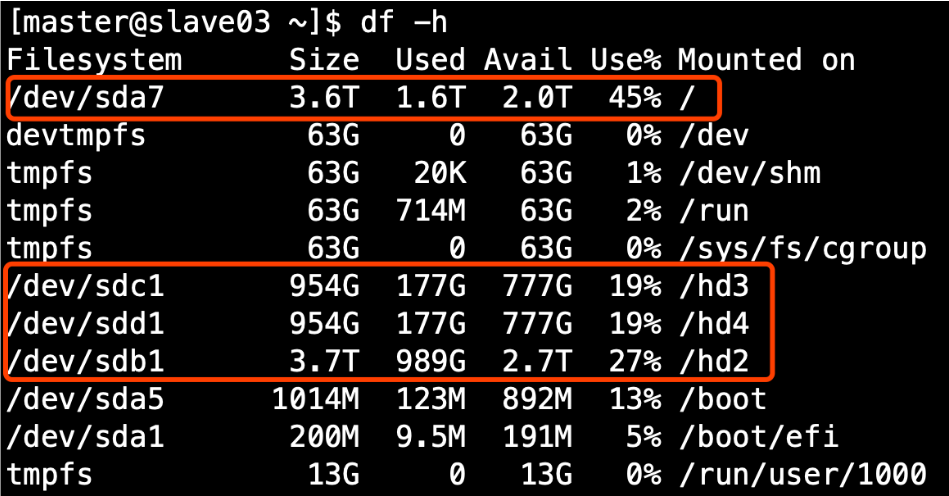
|  |  |  |  |
| --- | --- | --- | --- |
|  | 服务器hadoop102 | 服务器hadoop103 | 服务器hadoop104 |
| HDFS | NameNode  DataNode | DataNode | DataNode  SecondaryNameNode |
| Yarn | NodeManager | Resourcemanager  NodeManager | NodeManager |

注意：尽量使用离线方式安装

### 4.1.1 项目经验之HDFS存储多目录

若HDFS存储空间紧张，需要对DataNode进行磁盘扩展。

1）在DataNode节点增加磁盘并进行挂载。



2）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。

<property>

<name>dfs.datanode.data.dir</name>

<value>file:///${hadoop.tmp.dir}/dfs/data1,file:///hd2/dfs/data2,file:///hd3/dfs/data3,file:///hd4/dfs/data4</value>

</property>

3）增加磁盘后，保证每个目录数据均衡

开启数据均衡命令：

bin/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。

停止数据均衡命令：

bin/stop-balancer.sh

### 4.1.2 项目经验之支持LZO压缩配置

1）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。



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

[woaini@hadoop102 common]$ pwd

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

[woaini@hadoop102 common]$ ls

hadoop-lzo-0.4.20.jar

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

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

4）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>

5）同步core-site.xml到hadoop103、hadoop104

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

6）启动及查看集群

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

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

### 4.1.3 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

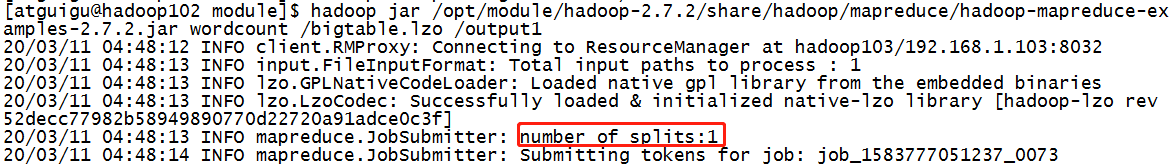
（1）将bigtable.lzo（150M）上传到集群的根目录

[woaini@hadoop102 module]$ hadoop fs -mkdir /input

[woaini@hadoop102 module]$ hadoop fs -put bigtable.lzo /input

（2）执行wordcount程序

[woaini@hadoop102 module]$ hadoop jar /opt/module/hadoop-2.7.2/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.2.jar wordcount /input /output1

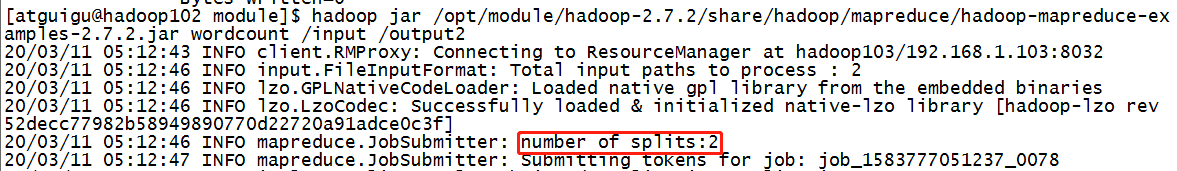


（3）对上传的LZO文件建索引

[woaini@hadoop102 module]$ hadoop jar /opt/module/hadoop-2.7.2/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

[woaini@hadoop102 module]$ hadoop jar /opt/module/hadoop-2.7.2/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.2.jar wordcount /input /output2



### 4.1.4 项目经验之基准测试

1） 测试HDFS写性能

测试内容：向HDFS集群写10个128M的文件

[woaini@hadoop102 mapreduce]$ hadoop jar /opt/module/hadoop-2.7.2/share/hadoop/mapreduce/hadoop-mapreduce-client-jobclient-2.7.2-tests.jar TestDFSIO -write -nrFiles 10 -fileSize 128MB

19/05/02 11:45:23 INFO fs.TestDFSIO: ----- TestDFSIO ----- : write

19/05/02 11:45:23 INFO fs.TestDFSIO: Date & time: Thu May 02 11:45:23 CST 2019

19/05/02 11:45:23 INFO fs.TestDFSIO: Number of files: 10

19/05/02 11:45:23 INFO fs.TestDFSIO: Total MBytes processed: 1280.0

19/05/02 11:45:23 INFO fs.TestDFSIO: **Throughput mb/sec: 10.69751115716984**

19/05/02 11:45:23 INFO fs.TestDFSIO: Average IO rate mb/sec: 14.91699504852295

19/05/02 11:45:23 INFO fs.TestDFSIO: IO rate std deviation: 11.160882132355928

19/05/02 11:45:23 INFO fs.TestDFSIO: Test exec time sec: 52.315

2）测试HDFS读性能

测试内容：读取HDFS集群10个128M的文件

[woaini@hadoop102 mapreduce]$ hadoop jar /opt/module/hadoop-2.7.2/share/hadoop/mapreduce/hadoop-mapreduce-client-jobclient-2.7.2-tests.jar TestDFSIO -read -nrFiles 10 -fileSize 128MB

19/05/02 11:56:36 INFO fs.TestDFSIO: ----- TestDFSIO ----- : read

19/05/02 11:56:36 INFO fs.TestDFSIO: Date & time: Thu May 02 11:56:36 CST 2019

19/05/02 11:56:36 INFO fs.TestDFSIO: Number of files: 10

19/05/02 11:56:36 INFO fs.TestDFSIO: Total MBytes processed: 1280.0

19/05/02 11:56:36 INFO fs.TestDFSIO: **Throughput mb/sec: 16.001000062503905**

19/05/02 11:56:36 INFO fs.TestDFSIO: Average IO rate mb/sec: 17.202795028686523

19/05/02 11:56:36 INFO fs.TestDFSIO: IO rate std deviation: 4.881590515873911

19/05/02 11:56:36 INFO fs.TestDFSIO: Test exec time sec: 49.116

19/05/02 11:56:36 INFO fs.TestDFSIO:

3）删除测试生成数据

[woaini@hadoop102 mapreduce]$ hadoop jar /opt/module/hadoop-2.7.2/share/hadoop/mapreduce/hadoop-mapreduce-client-jobclient-2.7.2-tests.jar TestDFSIO -clean

4）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

[woaini@hadoop102 mapreduce]$ hadoop jar /opt/module/hadoop-2.7.2/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.2.jar randomwriter random-data

（2）执行Sort程序

[woaini@hadoop102 mapreduce]$ hadoop jar /opt/module/hadoop-2.7.2/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.2.jar sort random-data sorted-data

（3）验证数据是否真正排好序了

[woaini@hadoop102 mapreduce]$

hadoop jar /opt/module/hadoop-2.7.2/share/hadoop/mapreduce/hadoop-mapreduce-client-jobclient-2.7.2-tests.jar testmapredsort -sortInput random-data -sortOutput sorted-data

### 4.1.5 项目经验之Hadoop参数调优

1）HDFS参数调优hdfs-site.xml

dfs.namenode.handler.count=20 \* log2(Cluster Size)，比如集群规模为8台时，此参数设置为60

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。设置该值的一般原则是将其设置为集群大小的自然对数乘以20，即20logN，N为集群大小。

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：

内存利用率不够。这个一般是Yarn的2个配置造成的，单个任务可以申请的最大内存大小，和Hadoop单个节点可用内存大小。调节这两个参数能提高系统内存的利用率。

（a）yarn.nodemanager.resource.memory-mb

表示该节点上YARN可使用的物理内存总量，默认是8192（MB），注意，如果你的节点内存资源不够8GB，则需要调减小这个值，而YARN不会智能的探测节点的物理内存总量。

（b）yarn.scheduler.maximum-allocation-mb

单个任务可申请的最多物理内存量，默认是8192（MB）。

3）Hadoop宕机

（1）如果**MR造成系统宕机**。此时要控制Yarn同时运行的任务数，和每个任务申请的最大内存。调整参数：yarn.scheduler.maximum-allocation-mb（单个任务可申请的最多物理内存量，默认是8192MB）

（2）如果写入文件过量造成**NameNode宕机**。那么调高Kafka的存储大小，控制从Kafka到HDFS的写入速度。高峰期的时候用Kafka进行缓存，高峰期过去数据同步会自动跟上。

## 4.2 Zookeeper安装

### 4.2.1 安装ZK

集群规划

|  |  |  |  |
| --- | --- | --- | --- |
|  | 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/woaini/bin目录下创建脚本

[woaini@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

};;

"status"){

for i in hadoop102 hadoop103 hadoop104

do

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

done

};;

esac

2）增加脚本执行权限

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

3）Zookeeper集群启动脚本

[woaini@hadoop102 module]$ zk.sh start

4）Zookeeper集群停止脚本

[woaini@hadoop102 module]$ zk.sh stop

### 4.2.3 项目经验之Linux环境变量

1）修改/etc/profile文件：用来设置系统环境参数，比如$PATH. 这里面的环境变量是对系统内所有用户生效。使用bash命令，需要source /etc/profile一下。

2）修改~/.bashrc文件：针对某一个特定的用户，环境变量的设置只对该用户自己有效。使用bash命令，只要以该用户身份运行命令行就会读取该文件。

3）把/etc/profile里面的环境变量追加到~/.bashrc目录

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

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

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

4）说明

登录式Shell，采用用户名比如woaini登录，会自动加载/etc/profile

非登录式Shell，采用ssh 比如ssh hadoop103登录，不会自动加载/etc/profile，会自动加载~/.bashrc

## 4.3 日志生成

### 4.3.1 日志启动

1）代码参数说明

// 参数一：控制发送每条的延时时间，默认是0

Long delay = args.length > 0 ? Long.parseLong(args[0]) : 0L;

// 参数二：循环遍历次数

int loop\_len = args.length > 1 ? Integer.parseInt(args[1]) : 1000;

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

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

3）在hadoop102上执行jar程序

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

说明1：

java -classpath 需要在jar包后面指定全类名；

java -jar 需要查看一下解压的jar包META-INF/ MANIFEST.MF文件中，Main-Class是否有全类名。如果有可以用java -jar，如果没有就需要用到java -classpath

说明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

4）在/tmp/logs路径下查看生成的日志文件

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

[woaini@hadoop102 logs]$ ls

app-2020-03-10.log

### 4.3.2 集群日志生成启动脚本

1）在/home/woaini/bin目录下创建脚本lg.sh

[woaini@hadoop102 bin]$ vim lg.sh

2）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.woaini.appclient.AppMain $1 $2 >/dev/null 2>&1 &"

done

3）修改脚本执行权限

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

4）启动脚本

[woaini@hadoop102 module]$ lg.sh

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

[woaini@hadoop102 logs]$ ls

app-2020-03-10.log

[woaini@hadoop103 logs]$ ls

app-2020-03-10.log

### 4.3.3 集群时间同步修改脚本（非正规临时脚本）

企业开发时，参考hadoop集群时间同步。

1）在/home/woaini/bin目录下创建脚本dt.sh

[woaini@hadoop102 bin]$ vim dt.sh

2）在脚本中编写如下内容

#!/bin/bash

for i in hadoop102 hadoop103 hadoop104

do

echo "========== $i =========="

ssh -t $i "sudo date -s $1"

done

注意：ssh -t 通常用于ssh远程执行sudo命令

3）修改脚本执行权限

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

4）启动脚本

[woaini@hadoop102 bin]$ dt.sh 2020-03-10

### 4.3.4 集群所有进程查看脚本

1）在/home/woaini/bin目录下创建脚本xcall.sh

[woaini@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

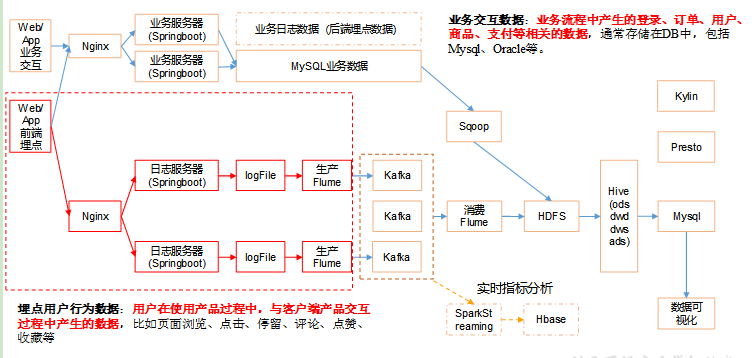
3）修改脚本执行权限

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

4）启动脚本

[woaini@hadoop102 bin]$ xcall.sh jps

## 4.4 采集日志Flume



### 4.4.1 日志采集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 |  |

### 4.4.2 项目经验之Flume组件

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。

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，不支持断点续传。

（2）batchSize大小如何设置？

答：Event 1K左右时，500-1000合适（默认为100）

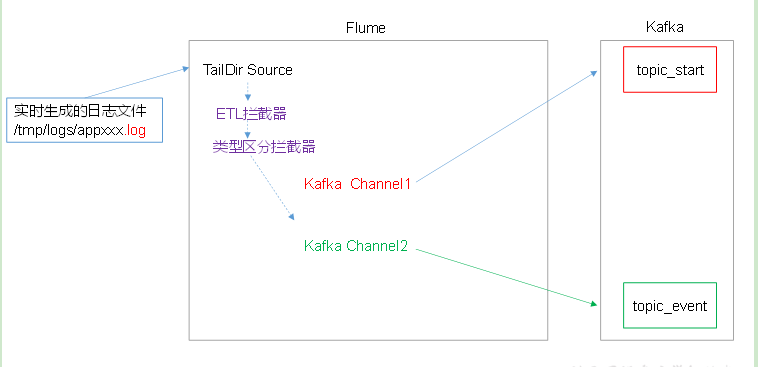
2）Channel

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。

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。

### 4.4.3 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

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

在文件配置如下内容

a1.sources=r1

a1.channels=c1 c2

# configure source

a1.sources.r1.type = TAILDIR

a1.sources.r1.positionFile = /opt/module/flume/test/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.woaini.flume.interceptor.LogETLInterceptor$Builder

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

a1.sources.r1.selector.type = multiplexing

a1.sources.r1.selector.header = topic

a1.sources.r1.selector.mapping.topic\_start = c1

a1.sources.r1.selector.mapping.topic\_event = c2

# configure channel

a1.channels.c1.type = org.apache.flume.channel.kafka.KafkaChannel

a1.channels.c1.kafka.bootstrap.servers = hadoop102:9092,hadoop103:9092,hadoop104:9092

a1.channels.c1.kafka.topic = topic\_start

a1.channels.c1.parseAsFlumeEvent = false

a1.channels.c1.kafka.consumer.group.id = flume-consumer

a1.channels.c2.type = org.apache.flume.channel.kafka.KafkaChannel

a1.channels.c2.kafka.bootstrap.servers = hadoop102:9092,hadoop103:9092,hadoop104:9092

a1.channels.c2.kafka.topic = topic\_event

a1.channels.c2.parseAsFlumeEvent = false

a1.channels.c2.kafka.consumer.group.id = flume-consumer

注意：com.woaini.flume.interceptor.LogETLInterceptor和com.woaini.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。



### 4.4.4 Flume的ETL和分类型拦截器

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.woaini.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.woaini.flume.interceptor包下创建LogETLInterceptor类名

Flume ETL拦截器LogETLInterceptor

**package** com.woaini.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) {  
  
 *// 1 获取数据* **byte**[] body = event.getBody();  
 String log = **new** String(body, Charset.*forName*(**"UTF-8"**));  
  
 *// 2 判断数据类型并向Header中赋值* **if** (log.contains(**"start"**)) {  
 **if** (LogUtils.*validateStart*(log)){  
 **return** event;  
 }  
 }**else** {  
 **if** (LogUtils.*validateEvent*(log)){  
 **return** event;  
 }  
 }  
  
 *// 3 返回校验结果* **return null**;  
 }  
  
 @Override  
 **public** List<Event> intercept(List<Event> events) {  
  
 ArrayList<Event> interceptors = **new** ArrayList<>();  
  
 **for** (Event event : events) {  
 Event intercept1 = intercept(event);  
  
 **if** (intercept1 != **null**){  
 interceptors.add(intercept1);  
 }  
 }  
  
 **return** interceptors;  
 }  
  
 @Override  
 **public void** close() {  
  
 }  
  
 **public static class** Builder **implements** Interceptor.Builder{  
  
 @Override  
 **public** Interceptor build() {  
 **return new** LogETLInterceptor();  
 }  
  
 @Override  
 **public void** configure(Context context) {  
  
 }  
 }  
}

4）Flume日志过滤工具类

**package** com.woaini.flume.interceptor;  
**import** org.apache.commons.lang.math.NumberUtils;  
  
**public class** LogUtils {  
  
 **public static boolean** validateEvent(String log) {  
 *// 服务器时间 | json  
 // 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":[]}  
  
 // 1 切割* String[] logContents = log.split(**"\\|"**);  
  
 *// 2 校验* **if**(logContents.**length** != 2){  
 **return false**;  
 }  
  
 *//3 校验服务器时间* **if** (logContents[0].length()!=13 || !NumberUtils.*isDigits*(logContents[0])){  
 **return false**;  
 }  
  
 *// 4 校验json* **if** (!logContents[1].trim().startsWith(**"{"**) || !logContents[1].trim().endsWith(**"}"**)){  
 **return false**;  
 }  
  
 **return true**;  
 }  
  
 **public static boolean** validateStart(String log) {  
 *// {"action":"1","ar":"MX","ba":"HTC","detail":"542","en":"start","entry":"2","extend1":"","g":"S3HQ7LKM@gmail.com","hw":"640\*960","l":"en","la":"-43.4","ln":"-98.3","loading\_time":"10","md":"HTC-5","mid":"993","nw":"WIFI","open\_ad\_type":"1","os":"8.2.1","sr":"D","sv":"V2.9.0","t":"1559551922019","uid":"993","vc":"0","vn":"1.1.5"}*

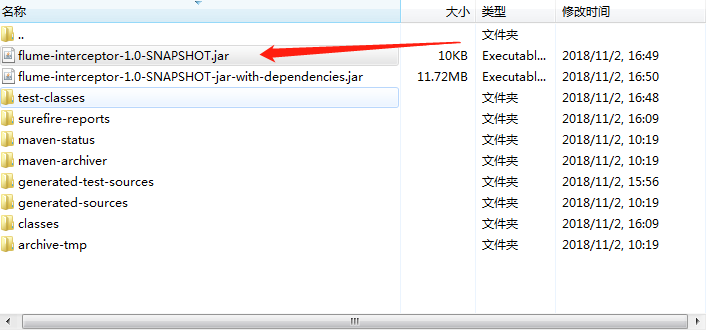
**if** (log == **null**){  
 **return false**;  
 }  
  
 *// 校验json* **if** (!log.trim().startsWith(**"{"**) || !log.trim().endsWith(**"}"**)){  
 **return false**;  
 }  
  
 **return true**;  
 }  
}

5）Flume日志类型区分拦截器LogTypeInterceptor

**package** com.woaini.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;  
**import** java.util.Map;  
  
**public class** LogTypeInterceptor **implements** Interceptor {  
 @Override  
 **public void** initialize() {  
  
 }  
  
 @Override  
 **public** Event intercept(Event event) {  
  
 *// 区分日志类型： body header  
 // 1 获取body数据* **byte**[] body = event.getBody();  
 String log = **new** String(body, Charset.*forName*(**"UTF-8"**));  
  
 *// 2 获取header* Map<String, String> headers = event.getHeaders();  
  
 *// 3 判断数据类型并向Header中赋值* **if** (log.contains(**"start"**)) {  
 headers.put(**"topic"**,**"topic\_start"**);  
 }**else** {  
 headers.put(**"topic"**,**"topic\_event"**);  
 }  
  
 **return** event;  
 }  
  
 @Override  
 **public** List<Event> intercept(List<Event> events) {  
  
 ArrayList<Event> interceptors = **new** ArrayList<>();  
  
 **for** (Event event : events) {  
 Event intercept1 = intercept(event);  
  
 interceptors.add(intercept1);  
 }  
  
 **return** interceptors;  
 }  
  
 @Override  
 **public void** close() {  
  
 }  
  
 **public static class** Builder **implements** Interceptor.Builder{  
  
 @Override  
 **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。

[woaini@hadoop102 lib]$ ls | grep interceptor

flume-interceptor-1.0-SNAPSHOT.jar

8）分发Flume到hadoop103、hadoop104

[woaini@hadoop102 module]$ xsync flume/

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

### 4.4.5 日志采集Flume启动停止脚本

1）在/home/woaini/bin目录下创建脚本f1.sh

[woaini@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 >/opt/module/flume/test1 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：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

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

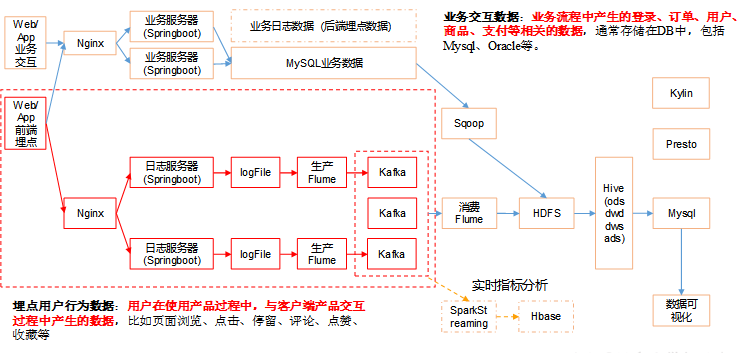
3）f1集群启动脚本

[woaini@hadoop102 module]$ f1.sh start

4）f1集群停止脚本

[woaini@hadoop102 module]$ f1.sh stop

## 4.5 Kafka安装



### 4.5.1 Kafka集群安装

集群规划：

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

### 4.5.2 Kafka集群启动停止脚本

1）在/home/woaini/bin目录下创建脚本kf.sh

[woaini@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

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

3）kf集群启动脚本

[woaini@hadoop102 module]$ kf.sh start

4）kf集群停止脚本

[woaini@hadoop102 module]$ kf.sh stop

### 4.5.3 查看Kafka Topic列表

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

### 4.5.4 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

[woaini@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

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

### 4.5.5 删除Kafka Topic

1）删除启动日志主题

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

2）删除事件日志主题

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

### 4.5.6 Kafka生产消息

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

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

>hello world

>woaini woaini

### 4.5.7 Kafka消费消息

[woaini@hadoop102 kafka]$ bin/kafka-console-consumer.sh \

--bootstrap-server hadoop102:9092 --from-beginning --topic topic\_start

--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。

### 4.5.8 查看Kafka Topic详情

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

--describe --topic topic\_start

### 4.5.9 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

[woaini@hadoop102 kafka]$ bin/kafka-producer-perf-test.sh --topic test --record-size 100 --num-records 100000 --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，可测出生产者最大吞吐量。

（2）Kafka会打印下面的信息

100000 records sent, 95877.277085 records/sec (**9.14 MB/sec**), 187.68 ms avg latency, 424.00 ms max latency, 155 ms 50th, 411 ms 95th, 423 ms 99th, 424 ms 99.9th.

参数解析：本例中一共写入10w条消息，吞吐量为**9.14 MB/sec**，每次写入的平均延迟为187.68毫秒，最大的延迟为424.00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。

[woaini@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，共消费数据**9.5368MB，吞吐量**2.0714MB/s，共消费**100010条，平均每秒消费**21722.4153条。**

### 4.5.10 项目经验之Kafka机器数量计算

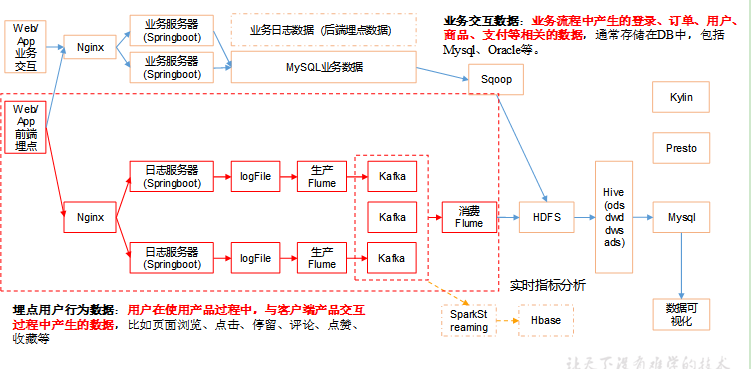
Kafka机器数量（经验公式）=2\*（峰值生产速度\*副本数/100）+1

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。

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

## 4.6 消费Kafka数据Flume

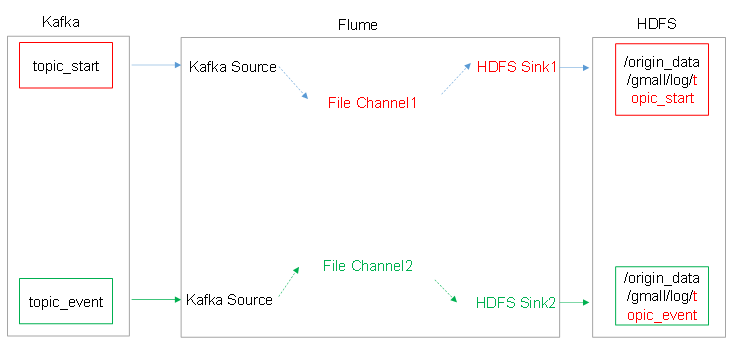


集群规划

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

### 4.6.1 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

[woaini@hadoop104 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 = file

a1.channels.c1.checkpointDir = /opt/module/flume/checkpoint/behavior1

a1.channels.c1.dataDirs = /opt/module/flume/data/behavior1/

a1.channels.c1.maxFileSize = 2146435071

a1.channels.c1.capacity = 1000000

a1.channels.c1.keep-alive = 6

## channel2

a1.channels.c2.type = file

a1.channels.c2.checkpointDir = /opt/module/flume/checkpoint/behavior2

a1.channels.c2.dataDirs = /opt/module/flume/data/behavior2/

a1.channels.c2.maxFileSize = 2146435071

a1.channels.c2.capacity = 1000000

a1.channels.c2.keep-alive = 6

## 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-

##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-

## 不要产生大量小文件

a1.sinks.k1.hdfs.rollInterval = 3600

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



### 4.6.2 项目经验之Flume组件

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 peformance

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

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

### 4.6.3 日志消费Flume启动停止脚本

1）在/home/woaini/bin目录下创建脚本f2.sh

[woaini@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

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

3）f2集群启动脚本

[woaini@hadoop102 module]$ f2.sh start

4）f2集群停止脚本

[woaini@hadoop102 module]$ f2.sh stop

### 4.6.4 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

[woaini@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，部署在单独的服务器上（4核8线程16G内存）

-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/woaini/bin目录下创建脚本cluster.sh

[woaini@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

sleep 4s;

#启动 Flume采集集群

f1.sh start

#启动 Kafka采集集群

kf.sh start

sleep 6s;

#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 Kafka采集集群

kf.sh stop

sleep 6s;

#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

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

3）cluster集群启动脚本

[woaini@hadoop102 module]$ cluster.sh start

4）cluster集群停止脚本

[woaini@hadoop102 module]$ cluster.sh stop

# 第5章 总结

## 5.1 数仓概念总结

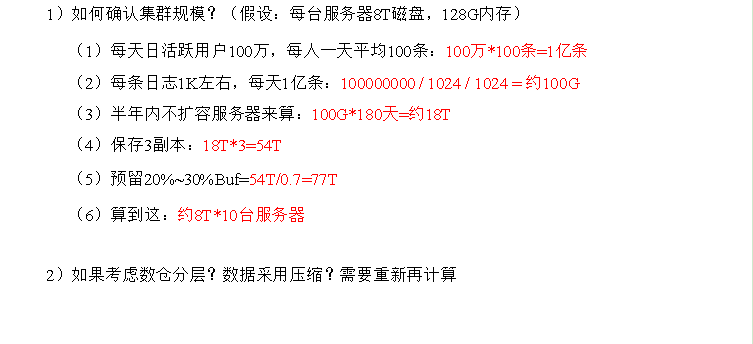
**1）数据仓库的输入数据源和输出系统分别是什么？**

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

### 5.2.1 集群规模计算



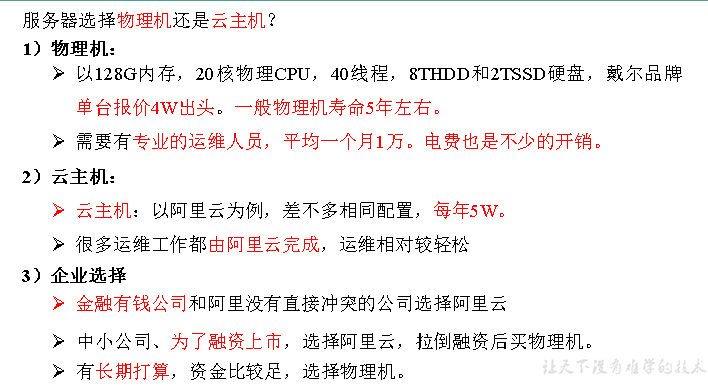
### 5.2.2 框架版本选型

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，但其实对中、小公司使用来说没有影响。离线

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

### 5.2.3 服务器选型



## 5.3 数据采集模块总结

### 5.3.1 Linux&Shell相关总结

1）Linux常用高级命令

|  |  |  |
| --- | --- | --- |
| 序号 | 命令 | 命令解释 |
| 1 | top | 查看内存 |
| 2 | df -h | 查看磁盘存储情况 |
| 3 | iotop | 查看磁盘IO读写(yum install iotop安装） |
| 4 | iotop -o | 直接查看比较高的磁盘读写程序 |
| 5 | netstat -tunlp | grep 端口号 | 查看端口占用情况 |
| 6 | uptime | 查看报告系统运行时长及平均负载 |
| 7 | ps aux | 查看进程 |

2）Shell常用工具（京东）

awk、sed、cut、sort

### 5.3.2 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。需要掌握让LZO文件支持切片。**

2）Hadoop常用端口号，50070,8088,19888,9000

3）Hadoop配置文件以及简单的Hadoop集群搭建。8个配置文件

4）HDFS读流程和写流程（笔试题，有朋友）

5）MapReduce的Shuffle过程及Hadoop优化（包括：压缩、小文件、集群优化）

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

8）HDFS存储多目录

9）Hadoop参数调优

10）项目经验之基准测试

### 5.3.3 Zookeeper相关总结

1）选举机制

半数机制，安装奇数台

10台服务器几台：3 台

20台服务器几台：5台

100台服务器几台：11台

不是越多越好，也不是越少越好。 如果多，通信时间长，效率低；如果太少，可靠性差。

2）常用命令

ls、get、create

### 5.3.4 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 Channel：减少了Flume的Sink阶段，提高了传输效率。

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

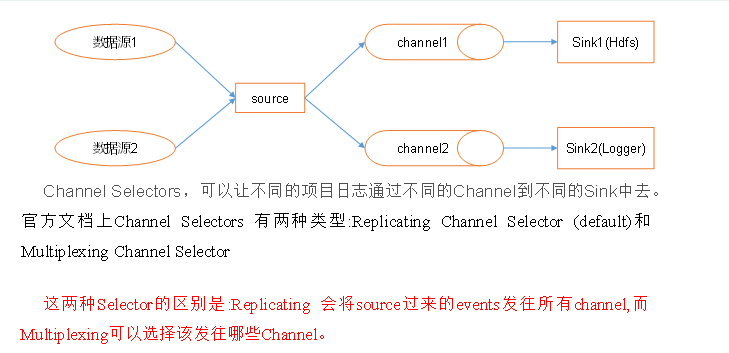
a）实现 Interceptor

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。

6）Flume内存

开发中在flume-env.sh中设置JVM heap为4G或更高，部署在单独的服务器上（4核8线程16G内存）

-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。

7）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 peformance

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

8）Sink：HDFS Sink小文件处理

（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

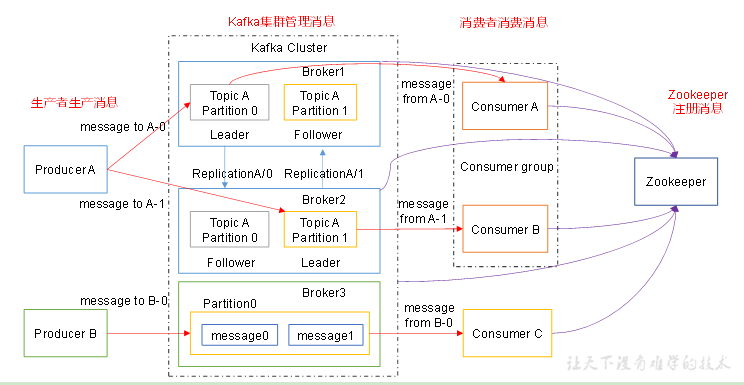
基于以上hdfs.rollInterval=3600，hdfs.rollSize=134217728，hdfs.rollCount =0几个参数综合作用，效果如下：

（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

举例：在2018-01-01 05:23的时侯sink接收到数据，那会产生如下tmp文件：

### 5.3.5 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

3天

4）Kafka的硬盘大小

每天的数据量\*3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。

（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Tp和Tc，单位可以是MB/s。

（4）然后假设总的目标吞吐量是Tt，那么分区数=Tt / min（Tp，Tc）

例如：producer吞吐量=10m/s；consumer吞吐量=50m/s，期望吞吐量100m/s；

分区数=100 / 10 =10分区

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，producer不等待kafka broker的ack，一直生产数据。

Ack=1，leader数据落盘就发送ack，producer收到ack才继续生产数据。

Ack=-1，ISR中的所有副本数据罗盘才发送ack，producer收到ack才继续生产数据。

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

Range和RoundRobin

12）Kafka中数据量计算

每天总数据量100g，每天产生1亿条日志， 10000万/24/60/60=1150条/每秒钟

平均每秒钟：1150条

低谷每秒钟：400条

高峰每秒钟：1150条\*（2-20倍）=2300条-23000条

每条日志大小：0.5k-2k（取1k）

每秒多少数据量：2.0M-20MB

13） Kafka挂掉

（1）Flume记录

（2）日志有记录

（3）短期没事

14）Kafka消息数据积压，Kafka消费能力不足怎么处理？

（1）如果是Kafka消费能力不足，则可以考虑增加Topic的分区数，并且同时提升消费组的消费者数量，消费者数=分区数。（两者缺一不可）

（2）如果是下游的数据处理不及时：提高每批次拉取的数量。批次拉取数据过少（拉取数据/处理时间<生产速度），使处理的数据小于生产的数据，也会造成数据积压。

15）Kafka幂等性

Kafka0.11版本引入了幂等性，幂等性配合at least once语义可以实现exactly once语义。但只能保证单次会话的幂等。

16）Kafka事务

Kafka0.11版本引入Kafka的事务机制，其可以保证生产者发往多个分区的一批数据的原子性。