大数据项目之在线教育-实时统计VIP新增数

版本：V1.0（cdh版）

# 第1章 项目需求及架构设计

## 1.1 项目需求分析

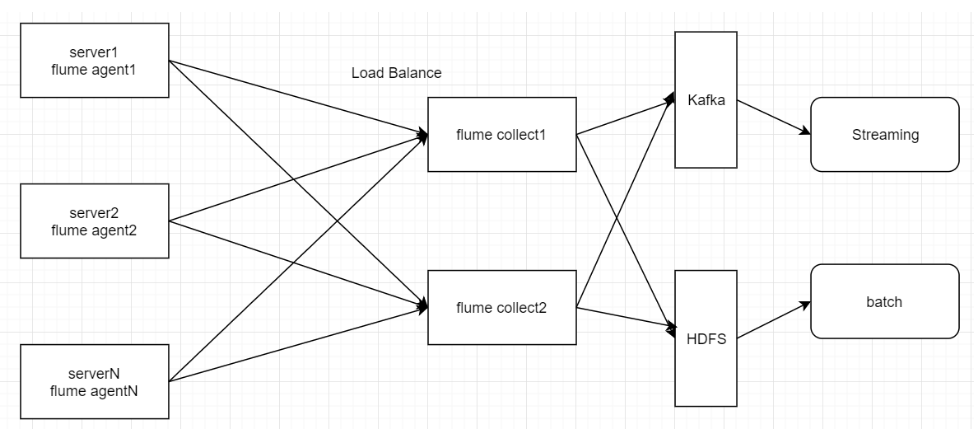
通过SparkStreaming技术，每10秒计算一次当天VIP用户量，按省份划分，计算结果保存到Mysql，并通过前端大屏不停刷新展示

## 2.2 项目框架

### 2.2.1 技术选型

* 数据采集传输：Flume,Kafka
* 数据存储：HDFS，Mysql
* 数据清洗：Spark Streaming
* 第三方库：ipip.net

### 2.2.2 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 |
| MySQL | 5.6.24 |
| Java | 1.8 |
| Zookeeper | 3.4.10 |

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** | **子服务** | **hadoop101** | **hadoop102** | **hadoop103** |
| HDFS | NameNode | √ |  |  |
|  | DataNode | √ | √ | √ |
| YARN | ResourceManager |  | √ |  |
|  | NodeManager | √ | √ | √ |
| Zookeeper | Zookeeper Server | √ | √ | √ |
| Flume | Flume(采集端) | √ | √ | √ |
|  | Flume(汇聚端) | √ | √ |  |
| Kafka | Kafka Broker | √ | √ | √ |
| Mysql | Mysql Server | √ |  |  |

# 第3章 数据生成模块

## 3.1 埋点数据基本格式（以\t分隔）

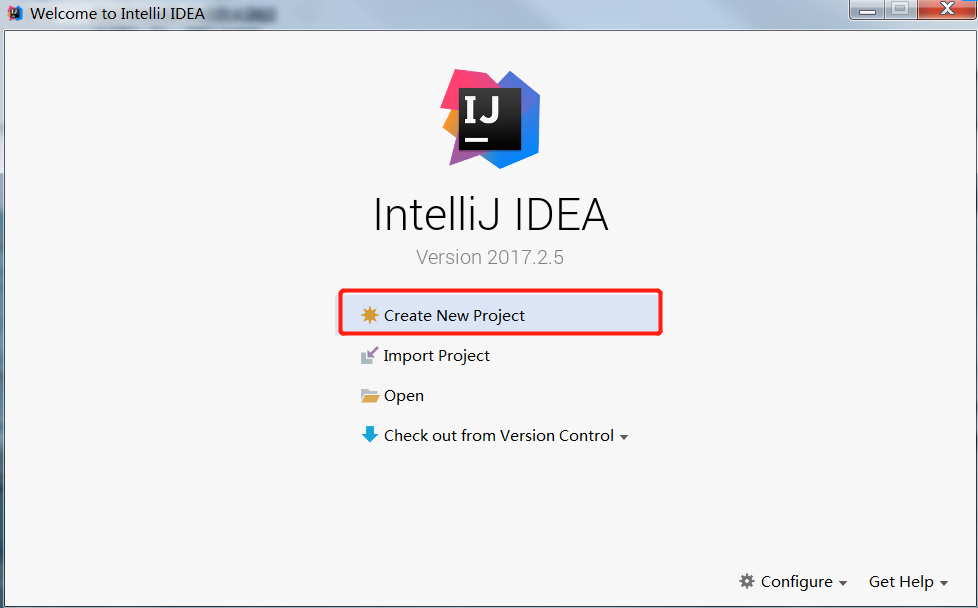
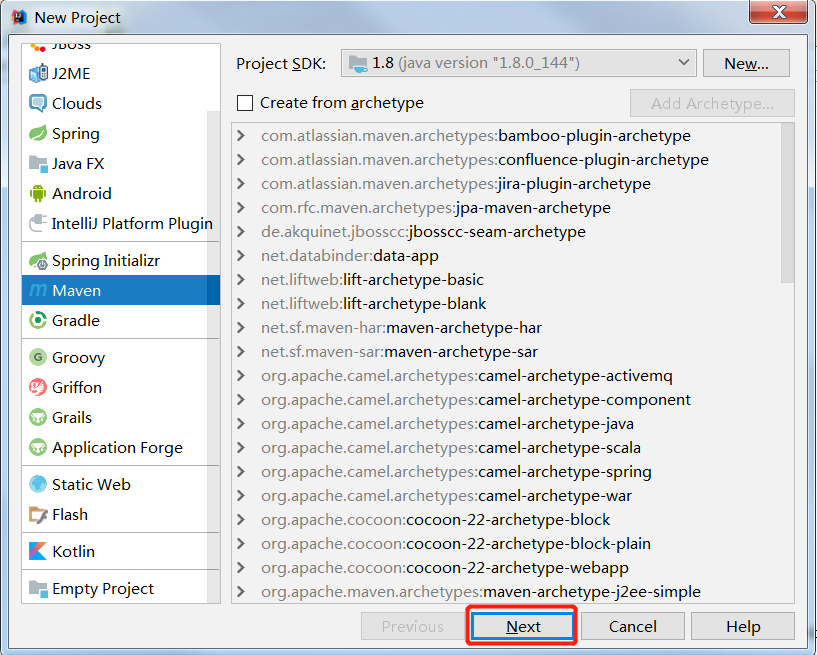
45660 45660 M 1 0 ios huawei wifi 59.48.116.0 18701445660 1 0 0 0 1.0 startHomework 1554134400

## 3.2 数据schema

uid STRING comment "用户唯一标识",  
username STRING comment "用户昵称",  
gender STRING comment "性别",  
level TINYINT comment "1代表小学，2代表初中，3代表高中",  
is\_vip TINYINT comment "0代表不是会员，1代表是会员",  
os STRING comment "操作系统:os,android等",  
channel STRING comment "下载渠道:auto,toutiao,huawei",  
net\_config STRING comment "当前网络类型",  
ip STRING comment "IP地址",  
phone STRING comment "手机号码",  
video\_id INT comment "视频id",  
video\_length INT comment "视频时长，单位秒",  
start\_video\_time BIGINT comment "开始看视频的时间缀，秒级",  
end\_video\_time BIGINT comment "退出视频时的时间缀，秒级",  
version STRING comment "版本",  
event\_key STRING comment "事件类型",  
event\_time BIGINT comment "事件发生时的时间缀，秒级"

### 3.4.1 创建Maven工程

1）创建bigdata

2）创建一个包名：com.atguigu.data\_monitor

3）在com.atguigu.data\_monitor包下创建一个Object: GeneratorUserBehaviorMonitorData。

4）在pom.xml文件中添加如下内容

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

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.atguigu</groupId>

<artifactId>bigdata</artifactId>

<version>1.0-SNAPSHOT</version>

<name>bigdata</name>

<!-- FIXME change it to the project's website -->

<url>http://www.example.com</url>

<properties>

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

<maven.compiler.source>1.8</maven.compiler.source>

<maven.compiler.target>1.8</maven.compiler.target>

</properties>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.11</version>

<scope>test</scope>

</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>

<archive>

<manifest>

<mainClass>com.atguigu.data\_monitor.GeneratorUserBehaviorMonitorData</mainClass>

</manifest>

</archive>

</configuration>

<executions>

<execution>

<id>make-assembly</id>

<phase>package</phase>

<goals>

<goal>single</goal>

</goals>

</execution>

</executions>

</plugin>

</plugins>

</build>

</project>

注意：com.atguigu.data\_monitor.GeneratorUserBehaviorMonitorData要和自己建的全类名一致。

### 3.4.2 生成测试数据代码

package com.atguigu.data\_monitor

import java.io.PrintWriter

import java.text.SimpleDateFormat

/\*\*

\* 生成用户行为模拟数据类

\*/

object GeneratorUserBehaviorMonitorData {

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

if (args.length != 1) {

println("Usage:Please input date like 2019-04-02")

System.exit(1)

}

generatorMonitorData(args(0))

}

def generatorMonitorData(date: String): Unit = {

// 初始化手机号前6位，后5位自动化补齐

val initPhone = 187014

// 初始化时间缀，精确到秒

val sdf = new SimpleDateFormat("yyyy-MM-dd")

val eventTime = sdf.parse(date)

val initTimestamp = eventTime.getTime() / 1000

// 生成看视频但是没有看完的数据

writeMonitorData2File("watchVideo", false, 10000, 10000, initPhone, initTimestamp)

// 生成看视频且看完的数据

writeMonitorData2File("completeVideo", true, 20001, 8000, initPhone, initTimestamp)

// 生成看完视频且开始做作业的数据

writeMonitorData2File("startHomework", true, 30001, 7000, initPhone, initTimestamp)

// 生成看完视频且做完作业的数据

writeMonitorData2File("completeHomework", true, 40001, 6000, initPhone, initTimestamp)

// 生成进入订单页的数据

writeMonitorData2File("enterOrderPage", true, 50001, 4000, initPhone, initTimestamp)

// 生成进入订单页且完成订单的数据

writeMonitorData2File("completeOrder", true, 60000, 2000, initPhone, initTimestamp)

}

/\*\*

\* 通过实始化的时间缀和是否完成视频的条件生成开始视频，结束视频，事件发生的时间

\*

\* @param initTimestamp

\* @param isCompleteVideo

\* @return

\*/

def getVideoTimeAndEventTime(initTimestamp: Long, isCompleteVideo: Boolean) = {

// 定义开始视频时间为传入的initTimestamp

val startVideoTime = initTimestamp

// 因为视频时长统一定义为300秒，如果是未完成视频，则endVideoTime统一定义为initTimestamp + 100，如完成，则统一加300

val endVideoTime = if (isCompleteVideo) initTimestamp + 300 else initTimestamp + 100

// 事件发生时间eventTime也统一定义为initTimestamp即可

val eventTime = initTimestamp

(startVideoTime, endVideoTime, eventTime)

}

/\*\*

\* 根据传入的事件类型生成不同的模拟数据

\*

\* @param initUid

\* @param userAccount

\* @param initPhone

\* @param initTimestamp

\* @param isCompleteVideo

\* @param dataType

\*/

def writeMonitorData2File(dataType: String, isCompleteVideo: Boolean, initUid: Int, userAccount: Int, initPhone: Int, initTimestamp: Long): Unit = {

val writer: PrintWriter = new PrintWriter(s"./${dataType}\_${initTimestamp}.txt")

// 获取开始看视频时间，结束看视频时间和事件发生时间

val (startVideoTime, endVideoTime, eventTime) = getVideoTimeAndEventTime(initTimestamp, isCompleteVideo)

for (uid <- initUid until (initUid + userAccount)) {

// 拼接完整的11位手机号

val phone = initPhone + "" + uid

val event = dataType match {

case "watchVideo" => s"""|$uid\t$uid\tF\t2\t0\tSymbian\tauto\t4G\t27.129.32.0\t$phone\t1\t300\t$startVideoTime\t0\t1.0\tstartVideo\t$eventTime

|$uid\t$uid\tF\t2\t0\tSymbian\tauto\t4G\t27.129.32.0\t$phone\t1\t300\t$startVideoTime\t$endVideoTime\t1.0\tendVideo\t$eventTime\n""".stripMargin

case "completeVideo" => s"""|$uid\t$uid\tM\t1\t0\tios\tauto\twifi\t59.48.116.0\t$phone\t0\t0\t0\t0\t1.0\tregisterAccount\t$eventTime

|$uid\t$uid\tM\t1\t0\tios\tauto\twifi\t59.48.116.0\t$phone\t0\t0\t0\t0\t1.0\tstartApp\t$eventTime

|$uid\t$uid\tM\t1\t0\tios\tauto\twifi\t59.48.116.0\t$phone\t1\t300\t$startVideoTime\t0\t1.0\tstartVideo\t$eventTime

|$uid\t$uid\tM\t1\t0\tios\tauto\twifi\t59.48.116.0\t$phone\t1\t300\t$startVideoTime\t$endVideoTime\t1.0\tendVideo\t$eventTime\n""".stripMargin

case "startHomework" => s"""|$uid\t$uid\tM\t1\t0\tios\thuawei\twifi\t59.48.116.0\t$phone\t0\t0\t0\t0\t1.0\tregisterAccount\t$eventTime

|$uid\t$uid\tM\t1\t0\tios\thuawei\twifi\t59.48.116.0\t$phone\t0\t0\t0\t0\t1.0\tstartApp\t$eventTime

|$uid\t$uid\tM\t1\t0\tios\thuawei\twifi\t59.48.116.0\t$phone\t1\t300\t$startVideoTime\t0\t1.0\tstartVideo\t$eventTime

|$uid\t$uid\tM\t1\t0\tios\thuawei\twifi\t59.48.116.0\t$phone\t1\t300\t$startVideoTime\t$endVideoTime\t1.0\tendVideo\t$eventTime

|$uid\t$uid\tM\t1\t0\tios\thuawei\twifi\t59.48.116.0\t$phone\t1\t0\t0\t0\t1.0\tstartHomework\t$eventTime\n""".stripMargin

case "completeHomework" => s"""|$uid\t$uid\tM\t1\t0\tios\thuawei\twifi\t59.48.116.0\t$phone\t0\t0\t0\t0\t1.0\tregisterAccount\t$eventTime

|$uid\t$uid\tM\t1\t0\tios\thuawei\twifi\t59.48.116.0\t$phone\t0\t0\t0\t0\t1.0\tstartApp\t$eventTime

|$uid\t$uid\tM\t1\t0\tios\thuawei\twifi\t59.48.116.0\t$phone\t1\t300\t$startVideoTime\t0\t1.0\tstartVideo\t$eventTime

|$uid\t$uid\tM\t1\t0\tios\thuawei\twifi\t59.48.116.0\t$phone\t1\t300\t$startVideoTime\t$endVideoTime\t1.0\tendVideo\t$eventTime

|$uid\t$uid\tM\t1\t0\tios\thuawei\twifi\t59.48.116.0\t$phone\t1\t0\t0\t0\t1.0\tstartHomework\t$eventTime

|$uid\t$uid\tM\t1\t0\tios\thuawei\twifi\t59.48.116.0\t$phone\t1\t0\t0\t0\t1.0\tcompleteHomework\t$eventTime\n""".stripMargin

case "enterOrderPage" => s"""|$uid\t$uid\tM\t1\t0\tios\ttoutiao\twifi\t59.48.116.0\t$phone\t0\t0\t0\t0\t1.1\tregisterAccount\t$eventTime

|$uid\t$uid\tM\t1\t0\tios\ttoutiao\twifi\t59.48.116.0\t$phone\t0\t0\t0\t0\t1.1\tstartApp\t$eventTime

|$uid\t$uid\tM\t1\t0\tios\ttoutiao\twifi\t59.48.116.0\t$phone\t1\t300\t$startVideoTime\t0\t1.1\tstartVideo\t$eventTime

|$uid\t$uid\tM\t1\t0\tios\ttoutiao\twifi\t59.48.116.0\t$phone\t1\t300\t$startVideoTime\t$endVideoTime\t1.1\tendVideo\t$eventTime

|$uid\t$uid\tM\t1\t0\tios\ttoutiao\twifi\t59.48.116.0\t$phone\t1\t0\t0\t0\t1.1\tstartHomework\t$eventTime

|$uid\t$uid\tM\t1\t0\tios\ttoutiao\twifi\t59.48.116.0\t$phone\t1\t0\t0\t0\t1.1\tcompleteHomework\t$eventTime

|$uid\t$uid\tM\t1\t0\tios\ttoutiao\twifi\t59.48.116.0\t$phone\t0\t0\t0\t0\t1.1\tenterOrderPage\t$eventTime\n""".stripMargin

case "completeOrder" => s"""|$uid\t$uid\tM\t1\t0\tios\ttoutiao\twifi\t42.86.6.0\t$phone\t0\t0\t0\t0\t2.0\tregisterAccount\t$eventTime

|$uid\t$uid\tM\t1\t0\tios\ttoutiao\twifi\t42.86.6.0\t$phone\t0\t0\t0\t0\t2.0\tstartApp\t$eventTime

|$uid\t$uid\tM\t1\t0\tios\ttoutiao\twifi\t42.86.6.0\t$phone\t1\t300\t$startVideoTime\t0\t2.0\tstartVideo\t$eventTime

|$uid\t$uid\tM\t1\t0\tios\ttoutiao\twifi\t42.86.6.0\t$phone\t1\t300\t$startVideoTime\t$endVideoTime\t2.0\tendVideo\t$eventTime

|$uid\t$uid\tM\t1\t0\tios\ttoutiao\twifi\t42.86.6.0\t$phone\t1\t0\t0\t0\t2.0\tstartHomework\t$eventTime

|$uid\t$uid\tM\t1\t0\tios\ttoutiao\twifi\t42.86.6.0\t$phone\t1\t0\t0\t0\t2.0\tcompleteHomework\t$eventTime

|$uid\t$uid\tM\t1\t0\tios\ttoutiao\twifi\t42.86.6.0\t$phone\t0\t0\t0\t0\t2.0\tenterOrderPage\t$eventTime

|$uid\t$uid\tM\t1\t0\tios\ttoutiao\twifi\t42.86.6.0\t$phone\t0\t0\t0\t0\t2.0\tcompleteOrder\t$eventTime\n|""".stripMargin

}

writer.write(event)

}

writer.close()

}

}

# 第4章 数据采集模块

## 4.1 CM及Hadoop安装(文档里修改了lzo安装)



**在安装CDH时候已经安装了hadoop集群**

**1.cdh安装的各个框架，其bin目录在/usr/bin, conf都在“/etc/框架名”内,**

**2.实际上，两个目录内的命令和配置文件都是链接到****/etc/alternatives下的命令和配置文件（比如hive命令，和hive-conf）,**

**3.进一步，发现/etc/alternatives里面的命令和配置文件，都是链接到其他具体路径，多数都是命令链接到/opt/cloudera/parcels/CDH-5.12.1-1.cdh5.12.1.p0.3/bin目录下的命令，**

**配置文件链接到/opt/cloudera/parcels/CDH-5.12.1-1.cdh5.12.1.p0.3/etc/框架名/，少数配置文件不是（比如hive-conf还是指向/etc/hive/conf.cloudera.hive）**

## 4.2 Zookeeper安装

详见：尚硅谷大数据技术之Zookeeper



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

## 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：尚硅谷大数据技术之Flume

文档内补充TLS/SSL验证错误问题和单个agent配置



### 4.3.2 日志采集Flume配置

1. Flume组件介绍
2. Souroce
3. Taildir Source相比Exec Source、Spooling Directory Source的优势（自我介绍时的亮点）

答：1.7版本之前，实现实时采集日志的Source只有Exec Source，但此Source可能会丢失数据（见官网描述）  
大家为了实现实时采集的效果，又保证数据安全，只能每隔半分钟产生一个并移动到Spooling Directory监控的目录中，此类做法会在web server中产生非常多的日志小文件，不利于管理  
1.7版本之后，出现了Taildir Source，即可以实时采集数据，又保证了数据安全，内部实现了类似断点续传的功能

1. batchSize大小如何设置？

答：event1K左右时，500-1000合适（默认为100）

1. Channel
2. FileChannel和MemoryChannel区别(面试题)

答：

1、MemoryChannel传输数据速度更快，但因为数据保证在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

2、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

1. FileChannel优化

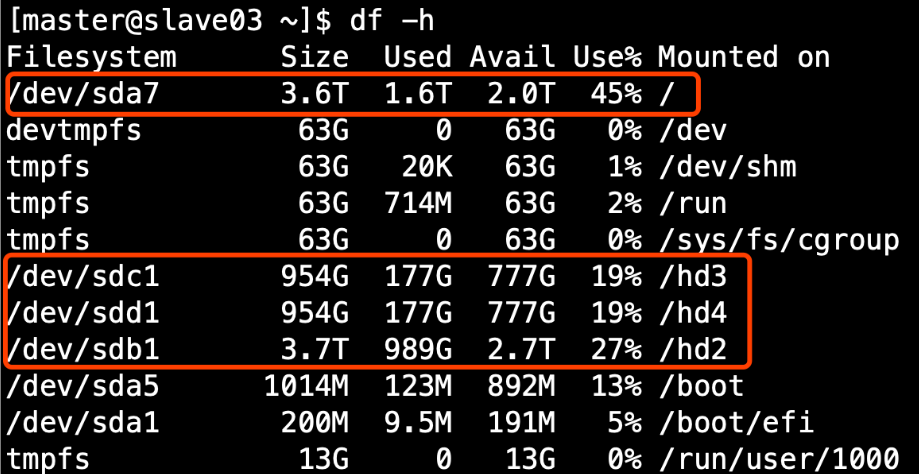
1、通过配置dataDirs指向多个路径，每个路径对应不同的硬盘，增大Flume吞吐量。

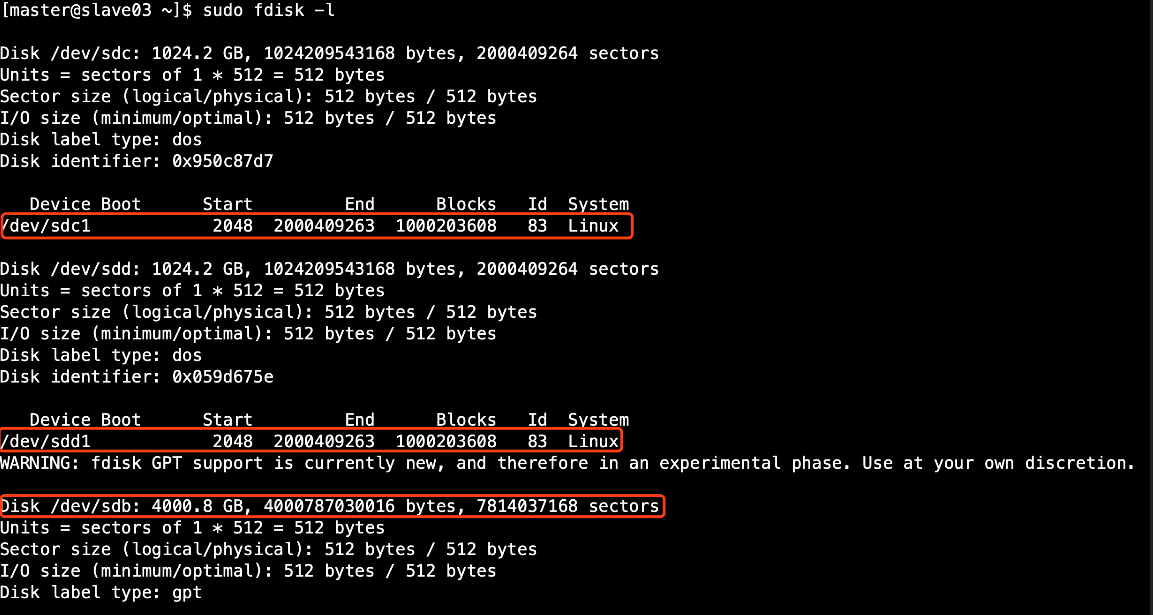
官方说明如下：

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

2、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

如何查看目录对应哪块硬盘? df -h,fdisk -l命令





1. Sink：HDFS Sink
2. 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
3. HDFS存入大量小文件，有什么影响？（面试题）

答：

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。

1. 现实生活中理解小文件 -- 年会游戏方案设计
2. 生产环境HDFS Sink建议配置

a1.sinks.k1.type=hdfs

a1.sinks.k1.hdfs.path=hdfs://atguigu:8020/guolong/%Y%m%d

a1.sinks.k1.hdfs.fileType=DataStream

a1.sinks.k1.hdfs.writeFormat=TEXT

# 每600秒滚动一个文件

a1.sinks.k1.hdfs.rollInterval=600

# 每128M滚动一个文件

a1.sinks.k1.hdfs.rollSize=134217728

a1.sinks.k1.hdfs.rollCount=0

# 每次拉取1000个event写入HDFS

a1.sinks.k1.hdfs.batchsize=1000

a1.sinks.k1.hdfs.threadsPoolSize=16

a1.sinks.k1.channel=c1

a1.sinks.k1.hdfs.filePrefix=guolong.%Y%m%d%H%M

a1.sinks.k1.hdfs.idelTimeout=600

a1.sinks.k1.hdfs.round=true

a1.sinks.k1.hdfs.roundValue=10

a1.sinks.k1.hdfs.roundUnit= minute

1. 基于以上hdfs.rollInterval=1800，hdfs.rollSize=134217728，hdfs.roundValue=10，hdfs.roundUnit= minute几个参数综合作用，效果如下：

1、tmp文件在达到128M时会滚动生成正式文件

2、tmp文件创建超10分钟时会滚动生成正式文件

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

/guolong/20180101/guolong.201801010520.tmp

即使文件内容没有达到128M，也会在05:33时滚动生成正式文件

1. 测试启动脚本

bin/flume-ng agent -n a1(agent的名称) -c conf -f conf/example(配置文件名称) **-Dflume.root.logger=DEBUG,console**

标红部分重点关注，可以通过DEBUG模式快速定位问题

1. Flume分层
   1. Flume为什么分两层

答：

如果只有一层，日志采集服务器非常多，此时会有很多个Flume agent，同时向HDFS写数据会产生多个client，对HDFS来说压力过大

只有一层时，部分业务配置只能在这层配置，如后续配置修改，则要修改的位置太多，不利于后期维护

* 1. 采集层
     1. 使用supervior方式保证agent挂掉后自动重启
     2. 因为要采集业务日志，所以需要部署在业务服务器上
     3. 根据服务器配置设置JVM heap，一般设置512M – 1G
  2. 渠聚层
     1. 使用load\_balance
     2. JVM heap一般设置为4G或更高
     3. 部署在单独的服务器上（4核8线程16G内存）
  3. JVM调优
     1. -Xmx与-Xms设置一样，减少内存抖动带来的性能影响

1. 基于以上双层架构配置，分析Flume如何保证数据至少处理一次
2. 采集层agent挂掉：因为使用了Taildir Source，所以可以断点续传
3. 汇聚层agent挂掉：因为使用load balance，实现了负载均衡和高可用
4. Flume的source向channel写入数据、sink从channel拉取数据本身带有事务机制
5. 思考题 ：

汇聚层向HDFS写数据时，channel中积压了大量数据，如何处理

## 4.4 Kafka相关

1. 主机层面配置
2. 调整最大文件描述符

sudo vi /etc/security/limits.conf，追加4行内容，\*号代表对所有用户生效

\* soft nofile 65536

\* hard nofile 65536

配置后新打开窗口，使用ulimit -n查看

1. log.dirs设置多个磁盘对应的目录，提高吞吐量
2. JVM heap一般设置为4G即可，并且使用G1收集器

# 第5章 SparkStreaming代码实现

## 5.1 POM文件

<?xml version="1.0" encoding="UTF-8"?>  
  
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
  
 <groupId>com.atguigu</groupId>  
 <artifactId>bigdata</artifactId>  
 <version>1.0-SNAPSHOT</version>  
  
 <name>bigdata</name>  
 <!-- *FIXME change it to the project's website* -->  
 <url>http://www.example.com</url>  
  
 <properties>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 <maven.compiler.source>1.8</maven.compiler.source>  
 <maven.compiler.target>1.8</maven.compiler.target>  
 </properties>  
  
 <dependencies>  
 <dependency>  
 <groupId>org.apache.spark</groupId>  
 <artifactId>spark-core\_2.11</artifactId>  
 <version>2.3.0</version>  
 </dependency>  
 <!-- https://mvnrepository.com/artifact/org.apache.spark/spark-sql -->  
 <dependency>  
 <groupId>org.apache.spark</groupId>  
 <artifactId>spark-sql\_2.11</artifactId>  
 <version>2.3.0</version>  
 </dependency>  
 <!-- https://mvnrepository.com/artifact/org.apache.spark/spark-streaming -->  
 <dependency>  
 <groupId>org.apache.spark</groupId>  
 <artifactId>spark-streaming\_2.11</artifactId>  
 <version>2.3.0</version>  
 </dependency>  
 <!-- https://mvnrepository.com/artifact/org.apache.spark/spark-hive -->  
 <dependency>  
 <groupId>org.apache.spark</groupId>  
 <artifactId>spark-hive\_2.11</artifactId>  
 <version>2.3.0</version>  
 </dependency>  
 <!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->  
 <dependency>  
 <groupId>mysql</groupId>  
 <artifactId>mysql-connector-java</artifactId>  
 <version>5.1.41</version>  
 </dependency>  
 <dependency>  
 <groupId>junit</groupId>  
 <artifactId>junit</artifactId>  
 <version>4.11</version>  
 <scope>test</scope>  
 </dependency>  
 <dependency>  
 <groupId>org.scalikejdbc</groupId>  
 <artifactId>scalikejdbc\_2.11</artifactId>  
 <version>2.2.1</version>  
 </dependency>  
 <dependency>  
 <groupId>org.apache.spark</groupId>  
 <artifactId>spark-streaming-kafka-0-8\_2.11</artifactId>  
 <version>2.3.0</version>  
 </dependency>  
 <dependency>  
 <groupId>net.ipip</groupId>  
 <artifactId>ipdb</artifactId>  
 <version>1.1.1</version>  
 </dependency>  
 </dependencies>  
  
 <build>  
 <sourceDirectory>src/main/</sourceDirectory>  
 <testSourceDirectory>src/test/</testSourceDirectory>  
 <plugins>  
 <plugin>  
 <!-- see http://davidb.github.com/scala-maven-plugin -->  
 <groupId>net.alchim31.maven</groupId>  
 <artifactId>scala-maven-plugin</artifactId>  
 <version>3.1.3</version>  
 <executions>  
 <execution>  
 <goals>  
 <goal>compile</goal>  
 <goal>testCompile</goal>  
 </goals>  
 <configuration>  
 <args>  
 <arg>-dependencyfile</arg>  
 <arg>${project.build.directory}/.scala\_dependencies</arg>  
 </args>  
 </configuration>  
 </execution>  
 </executions>  
 </plugin>  
 <plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-shade-plugin</artifactId>  
 <version>1.5</version>  
 <executions>  
 <execution>  
 <phase>package</phase>  
 <goals>  
 <goal>shade</goal>  
 </goals>  
 <configuration>  
 <shadedArtifactAttached>true</shadedArtifactAttached>  
 <shadedClassifierName>allinone</shadedClassifierName>  
 <artifactSet>  
 <includes>  
 <include>\*:\*</include>  
 </includes>  
 </artifactSet>  
 <transformers>  
 <transformer  
 implementation="org.apache.maven.plugins.shade.resource.AppendingTransformer">  
 <resource>reference.conf</resource>  
 </transformer>  
 <transformer  
 implementation="org.apache.maven.plugins.shade.resource.ManifestResourceTransformer">  
 <manifestEntries>  
 <Main-Class>com.atguigu.user\_behavior.AppCourseStudyAnalysis</Main-Class>  
 </manifestEntries>  
 </transformer>  
 </transformers>  
 <filters>  
 <filter>  
 <artifact>\*:\*</artifact>  
 <excludes>  
 <exclude>META-INF/\*.SF</exclude>  
 <exclude>META-INF/\*.DSA</exclude>  
 <exclude>META-INF/\*.RSA</exclude>  
 </excludes>  
 </filter>  
 </filters>  
 </configuration>  
 </execution>  
 </executions>  
 </plugin>  
 <plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-surefire-plugin</artifactId>  
 <version>2.13</version>  
 <configuration>  
 <useFile>false</useFile>  
 <disableXmlReport>true</disableXmlReport>  
 <!-- If you have classpath issue like NoDefClassError,... -->  
 <!-- useManifestOnlyJar>false</useManifestOnlyJar -->  
 <includes>  
 <include>\*\*/\*Test.\*</include>  
 <include>\*\*/\*Suite.\*</include>  
 </includes>  
 </configuration>  
 </plugin>  
 </plugins>  
 <pluginManagement>  
 <plugins>  
 <!--This plugin's configuration is used to store Eclipse m2e settings  
 only. It has no influence on the Maven build itself. -->  
 <plugin>  
 <groupId>org.eclipse.m2e</groupId>  
 <artifactId>lifecycle-mapping</artifactId>  
 <version>1.0.0</version>  
 <configuration>  
 <lifecycleMappingMetadata>  
 <pluginExecutions>  
 <pluginExecution>  
 <pluginExecutionFilter>  
 <groupId>  
 net.alchim31.maven  
 </groupId>  
 <artifactId>  
 scala-maven-plugin  
 </artifactId>  
 <versionRange>  
 [3.1.3,)  
 </versionRange>  
 <goals>  
 <goal>testCompile</goal>  
 <goal>compile</goal>  
 </goals>  
 </pluginExecutionFilter>  
 <action>  
 <ignore></ignore>  
 </action>  
 </pluginExecution>  
 </pluginExecutions>  
 </lifecycleMappingMetadata>  
 </configuration>  
 </plugin>  
 </plugins>  
 </pluginManagement>  
 </build>  
</project>

## 5.2 准备工作

1. 创建user-behavior的topic

bin/kafka-topics.sh --create --zookeeper 192.168.137.10:2181 --replication-factor 3 --partitions 3 --topic user-behavior

1. 创建Myql表用来存储offset

CREATE TABLE topic\_offset (

topic varchar(128) DEFAULT NULL,

part\_id int(11) DEFAULT NULL,

offset bigint(20) DEFAULT NULL

) ENGINE=InnoDB DEFAULT CHARSET=utf8

1. 初始化

insert into topic\_offset(topic, part\_id, offset) values('user-behavior',0,0);

insert into topic\_offset(topic, part\_id, offset) values('user-behavior',1,0);

insert into topic\_offset(topic, part\_id, offset) values('user-behavior',2,0);

1. 创建存储统计结果的Mysql表

CREATE TABLE vip\_increment\_analysis (

province varchar(128) DEFAULT NULL,

cnt int(11) DEFAULT NULL,

dt varchar(128) DEFAULT NULL

) ENGINE=InnoDB DEFAULT CHARSET=utf8

1. province和dt组成唯一索引

create unique index vip\_increment\_analysis\_unique on vip\_increment\_analysis (province,dt);

## 5.3 代码实现

package com.atguigu.streaming  
  
import java.sql.Date  
import java.text.SimpleDateFormat  
import java.util.Properties  
import kafka.common.TopicAndPartition  
import kafka.message.MessageAndMetadata  
import kafka.serializer.StringDecoder  
import net.ipip.ipdb.City  
import org.apache.spark.SparkConf  
import org.apache.spark.streaming.kafka.{HasOffsetRanges, KafkaUtils, OffsetRange}  
import org.apache.spark.streaming.{Seconds, StreamingContext}  
import scalikejdbc.{ConnectionPool, DB, \_}  
  
*/\*\*  
 \* 按地区分组统计每日新增VIP数量  
 \*/*object VipIncrementAnalysis {  
 // 提取出公共变量，转换算子共用  
 val *sdf* = new SimpleDateFormat("yyyy-MM-dd")  
  
 // 从properties文件里获取各种参数  
 val *prop* = new Properties()  
 *prop*.load(this.getClass.getClassLoader().getResourceAsStream("VipIncrementAnalysis.properties"))  
  
 // 使用静态ip资源库  
 val *ipdb* = new City(this.getClass().getClassLoader().getResource("ipipfree.ipdb").getPath())  
  
 // 获取jdbc相关参数  
 val *driver* = *prop*.getProperty("jdbcDriver")  
 val *jdbcUrl* = *prop*.getProperty("jdbcUrl")  
 val *jdbcUser* = *prop*.getProperty("jdbcUser")  
 val *jdbcPassword* = *prop*.getProperty("jdbcPassword")  
  
 // 设置jdbc  
 Class.*forName*(*driver*)  
 // 设置连接池  
 ConnectionPool.*singleton*(*jdbcUrl*, *jdbcUser*, *jdbcPassword*)  
  
 def main(args : Array[String]): Unit ={  
 // 参数检测  
 if(args.length != 1){  
 *println*("Usage:Please input checkpointPath")  
 System.*exit*(1)  
 }  
  
 // 通过传入参数设置检查点  
 val checkPoint = args(0)  
  
 // 通过getOrCreate方式可以实现从Driver端失败恢复  
 val ssc = StreamingContext.*getOrCreate*(checkPoint,  
 () => {*getVipIncrementByCountry*(checkPoint)}  
 )  
  
 // 启动流计算  
 ssc.start()  
 ssc.awaitTermination()  
 }  
  
 // 通过地区统计vip新增数量  
 def getVipIncrementByCountry(checkPoint : String): StreamingContext ={  
  
 // 定义update函数  
 val updateFunc = (values: Seq[Int], state: Option[Int]) => {  
 // 本批次value求合  
 val currentCount = values.sum  
  
 val previousCount = state.getOrElse(0)  
 Some(currentCount + previousCount)  
 }  
  
 // 设置批处理间隔  
 val processingInterval = *prop*.getProperty("processingInterval").toLong  
  
 // 获取kafka相关参数  
 val brokers = *prop*.getProperty("brokers")  
  
 val sparkConf = new SparkConf()  
 .set("spark.streaming.stopGracefullyOnShutdown","true")  
 .setAppName(this.getClass.getSimpleName)  
 val ssc = new StreamingContext(sparkConf, *Seconds*(processingInterval))  
  
 val kafkaParams = *Map*[String, String]("metadata.broker.list" -> brokers, "auto.offset.reset" -> "smallest","enable.auto.commit" ->"false")  
  
 // 获取offset  
 val fromOffsets = DB.*readOnly* { implicit session => sql"select topic, part\_id, offset from topic\_offset".  
 map { r =>  
 TopicAndPartition(r.string(1), r.int(2)) -> r.long(3)  
 }.list.apply().toMap  
 }  
  
 val messageHandler = (mmd : MessageAndMetadata[String, String]) => (mmd.topic + "-" + mmd.partition, mmd.message())  
 var offsetRanges : Array[OffsetRange] = Array.*empty*[OffsetRange]  
  
 // 获取Dstream  
 val messages = KafkaUtils.*createDirectStream*[String, String, StringDecoder, StringDecoder, (String, String)](ssc, kafkaParams, fromOffsets, messageHandler)  
  
 // 开启检查点  
 ssc.checkpoint(checkPoint)  
 messages.checkpoint(*Seconds*(processingInterval \* 10))  
  
 // 业务计算  
 messages.transform{rdd =>  
 offsetRanges = rdd.asInstanceOf[HasOffsetRanges].offsetRanges  
 rdd  
 }.filter{ msg =>  
 // 过滤非完成订单的数据，且验证数据合法性  
 *filterCompleteOrderData*(msg)  
 }.map{ msg =>  
 // 数据转换，返回((2019-04-03,北京),1)格式的数据  
 *getCountryAndDate*(msg)  
 }.updateStateByKey[Int]{  
 // 更新状态变量  
 updateFunc  
 }.filter{ state =>  
 // 只保留最近2天的数据，这样返回给Driver的数据会非常小  
 *filter2DaysBeforeState*(state)  
 }.foreachRDD(rdd=> {  
 // 将最近两天的数据返回给Driver端  
 val resultTuple: Array[((String, String), Int)] = rdd.collect()  
  
 // 开启事务  
 DB.*localTx* { implicit session =>  
 resultTuple.foreach(msg => {  
 val dt = msg.\_1.\_1  
 val province = msg.\_1.\_2  
 val cnt = msg.\_2.toLong  
  
 // 统计结果持久化到Mysql中  
 sql"""replace into vip\_increment\_analysis(province,cnt,dt) values (**$**{province},**$**{cnt},**$**{dt})""".executeUpdate().apply()  
 *println*(msg)  
 })  
  
 for (o <- offsetRanges) {  
 *println*(o.topic,o.partition,o.fromOffset,o.untilOffset)  
 // 保存offset  
 sql"""update topic\_offset set offset = **$**{o.untilOffset} where topic = **$**{o.topic} and part\_id = **$**{o.partition}""".update.apply()  
 }  
 }  
 })  
 ssc  
 }  
  
 */\*\*  
 \* 只保留最近2天的状态,因为怕系统时间或数据采集过程中有稍许延迟，所以没有设计保留1天  
 \*/* def filter2DaysBeforeState(state : ((String,String),Int)): Boolean ={  
 // 获取状态值对应的日期，并转换为13位的长整型时间缀  
 val day = state.\_1.\_1  
 val eventTime = *sdf*.parse(day).getTime  
 // 获取当前系统时间缀  
 val currentTime = System.*currentTimeMillis*()  
 // 两者比较，保留两天内的  
 if(currentTime - eventTime >= 172800000){  
 false  
 }else{  
 true  
 }  
 }  
  
 */\*\*  
 \* 过滤非完成订单的数据，且验证数据合法性  
 \*/* def filterCompleteOrderData(msg : (String,String)): Boolean ={  
 val fields = msg.\_2.split("\t")  
 // 切分后长度不为17，代表数据不合法  
 if(fields.length == 17){  
 val eventType = msg.\_2.split("\t")(15)  
 "completeOrder".equals(eventType)  
 }else{  
 false  
 }  
 }  
  
 */\*\*  
 \* 数据转换，返回((2019-04-03,北京),1)格式的数据  
 \** ***@param msg*** *\** ***@return*** *\*/* def getCountryAndDate(msg : (String,String)): ((String,String),Int) ={  
 val fields = msg.\_2.split("\t")  
 // 获取ip地址  
 val ip = fields(8)  
 // 获取事件时间  
 val eventTime = fields(16).toLong  
  
 // 根据日志中的eventTime获取对应的日期  
 val date = new Date(eventTime \* 1000)  
 val eventDay = *sdf*.format(date)  
  
 // 根据IP获取省份信息  
 var regionName = "未知"  
 val info = *ipdb*.findInfo(ip,"CN")  
 if(info != null){  
 regionName = info.getRegionName()  
 }  
  
 ((eventDay,regionName),1)  
 }  
}

# 第6章 订单行为异常用户及时运营

**6.1 业务场景**

通过BI数据分析，发现某段时间有部分用户短时间内多次进入付费定单页，却没有成为VIP用户。

我们要做的是通过实时计算，把这部分潜在VIP用户抓取出来，提供给客服或运营人员及时处理，增加这部分用户的VIP转化率。

**6.2 需求**

每隔2分钟，统计近10分钟的用户行为数据，当出现进入定单页（eventKey=enterOrderPage）3次，但是没有成功完成订单时(去业务数据库的实时备份表vip\_users查询用户是否为VIP)，即将用户uid持久化到Mysql中

**6.3 准备工作**

1. 创建保存最后结果的Mysql表

CREATE TABLE `unpayment\_record` (

`uid` varchar(128) NOT NULL COMMENT '用户id',

`phone` varchar(16) NOT NULL COMMENT '用户手机号',

`callback\_status` tinyint(4) DEFAULT NULL COMMENT '回访状态'

) ENGINE=InnoDB DEFAULT CHARSET=utf8

1. 创建唯一索引

CREATE UNIQUE INDEX unpayment\_record\_unique ON unpayment\_record (uid);

1. 创建保存offset的Mysql表

CREATE TABLE `unpayment\_topic\_offset` (

`topic` varchar(128) DEFAULT NULL,

`part\_id` int(11) DEFAULT NULL,

`offset` bigint(20) DEFAULT NULL

) ENGINE=InnoDB DEFAULT CHARSET=utf8

1. 初始化数据

insert into unpayment\_topic\_offset(topic, part\_id, offset) values('user-behavior',0,0);

insert into unpayment\_topic\_offset(topic, part\_id, offset) values('user-behavior',1,0);

insert into unpayment\_topic\_offset(topic, part\_id, offset) values('user-behavior',2,0);

1. 设置组合维一索引

CREATE UNIQUE INDEX unpayment\_topic\_offset\_unique ON unpayment\_topic\_offset (topic,part\_id);

1. 模拟业务数据表，需要实时从表中查询某个用户是否在表中

CREATE TABLE `vip\_user` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`uid` varchar(16) DEFAULT NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8

* 1. **代码实战**

1. package com.atguigu.streaming  
     
   import java.util.Properties  
   import kafka.common.TopicAndPartition  
   import kafka.message.MessageAndMetadata  
   import kafka.serializer.StringDecoder  
   import org.apache.spark.streaming.kafka.{HasOffsetRanges, KafkaUtils, OffsetRange}  
   import org.apache.spark.streaming.{Seconds, StreamingContext}  
   import org.apache.spark.SparkConf  
   import scalikejdbc.{ConnectionPool, DB}  
   import scalikejdbc.\_  
     
   */\*\*  
    \* 每隔  
    \*/*object UnPaymentAyalysis {  
    // 从properties文件里获取各种参数  
    val *prop* = new Properties()  
    *prop*.load(this.getClass.getClassLoader().getResourceAsStream("UnPaymentAnalysis.properties"))  
     
    // 获取jdbc相关参数  
    val *driver* = *prop*.getProperty("jdbcDriver")  
    val *jdbcUrl* = *prop*.getProperty("jdbcUrl")  
    val *jdbcUser* = *prop*.getProperty("jdbcUser")  
    val *jdbcPassword* = *prop*.getProperty("jdbcPassword")  
     
    // 设置批处理间隔  
    val *processingInterval* = *prop*.getProperty("processingInterval").toLong  
     
    // 获取kafka相关参数  
    val *brokers* = *prop*.getProperty("brokers")  
    val *topic* = *prop*.getProperty("topic")  
     
    // 设置jdbc  
    Class.*forName*(*driver*)  
    // 设置连接池  
    ConnectionPool.*singleton*(*jdbcUrl*, *jdbcUser*, *jdbcPassword*)  
     
    def main(args: Array[String]): Unit = {  
    // Create direct kafka stream with brokers and topics  
    val topicsSet = *topic*.split(",").toSet  
     
    val sparkConf = new SparkConf().set("spark.streaming.stopGracefullyOnShutdown","true").setAppName(this.getClass.getSimpleName)  
    val ssc = new StreamingContext(sparkConf, *Seconds*(*processingInterval*))  
     
    val kafkaParams = *Map*[String, String]("metadata.broker.list" -> *brokers*,"enable.auto.commit" ->"false")  
     
    // 获取offset  
    val fromOffsets = DB.*readOnly* { implicit session => sql"select topic, part\_id, offset from unpayment\_topic\_offset".  
    map { r =>  
    TopicAndPartition(r.string(1), r.int(2)) -> r.long(3)  
    }.list.apply().toMap  
    }  
     
    val messageHandler = (mmd : MessageAndMetadata[String, String]) => (mmd.topic, mmd.message())  
    var offsetRanges : Array[OffsetRange] = Array.*empty*[OffsetRange]  
     
    // 获取Dstream  
    val messages = KafkaUtils.*createDirectStream*[String, String, StringDecoder, StringDecoder, (String, String)](ssc, kafkaParams, fromOffsets, messageHandler)  
     
    // 业务计算  
    messages.transform{rdd =>  
    offsetRanges = rdd.asInstanceOf[HasOffsetRanges].offsetRanges  
    rdd  
    }.filter{ msg =>  
    // 过滤非完成订单的数据，且验证数据合法性  
    *filterCompleteOrderData*(msg)  
    }.map{ msg =>  
    // 数据格式转换为((uid,eventKey),1)  
    *transformData2Tuple*(msg)  
    }.reduceByKeyAndWindow((a:Int,b:Int) => a+b ,*Seconds*(*processingInterval* \* 4),*Seconds*(*processingInterval* \* 2))  
    .filter( state =>  
    // 过滤定单行为异常的用户  
    *filterUnnormalOrderUser*(state)  
    ).foreachRDD(rdd=> {  
    // 返回结果到客户端  
    val resultTuple = rdd.collect()  
    // 开启事务  
    DB.*localTx* { implicit session =>  
    resultTuple.foreach(msg => {  
    val uid = msg.\_1  
    // 统计结果持久化到Mysql中 *TODO  
    println*(msg)  
    sql"""replace into unpayment\_record(uid) values (**$**{uid})""".executeUpdate().apply()  
    })  
     
    for (o <- offsetRanges) {  
    *println*(o.topic,o.partition,o.fromOffset,o.untilOffset)  
    // 保存offset  
    sql"""update unpayment\_topic\_offset set offset = **$**{o.untilOffset} where topic = **$**{o.topic} and part\_id = **$**{o.partition}""".update.apply()  
    }  
    }  
    })  
    ssc.start()  
    ssc.awaitTermination()  
    }  
     
    */\*\*  
    \* 过滤定单行为异常的用户  
    \*/* def filterUnnormalOrderUser(state :(String,Int)): Boolean ={  
    // 获取用户uid  
    val uid = state.\_1  
    // 获取进入订单页的次数  
    val cnt = state.\_2  
    // 当进入订单页大于等于3时，去业务表查询用户当前是否为vip状态  
    if(cnt >= 3){  
    val result = DB.*readOnly*(implicit session => {  
    sql"""select id from vip\_user where uid=**$**{uid}""".map(rs => {  
    rs.get[Int](1)  
    }).list().apply()  
    })  
    // 如果结果为空，代表用户还不是vip，所以需要做后续运营  
    if(result.isEmpty){  
    true  
    }else{  
    false  
    }  
    }else{  
    false  
    }  
    }  
     
    */\*\*  
    \* 数据格式转换为(uid,1)  
    \** ***@param msg*** *\*/* def transformData2Tuple(msg : (String,String)) ={  
    // 数据转换，返回((uid,1)格式的数据  
    val fields = msg.\_2.split("\t")  
    val uid = fields(0)  
    (uid,1)  
    }  
    */\*\*  
    \* 过滤进入订单页的相关数据，且验证数据合法性  
    \*/* def filterCompleteOrderData(msg : (String,String)): Boolean ={  
    val fields = msg.\_2.split("\t")  
    // 切分后长度不为17，代表数据不合法  
    if(fields.length == 17){  
    val eventType = msg.\_2.split("\t")(15)  
    // 保留进入定单页  
    "enterOrderPage".equals(eventType)  
    }else{  
    false  
    }  
    }  
   }

# 第7章 面试题

## 7.1 Kafka的Direct和Reciver方式相比有何优势

这种新的不基于Receiver的直接方式，是在Spark 1.3中引入的，从而能够确保更加健壮的机制。替代掉使用Receiver来接收数据后，这种方式会周期性地查询Kafka，来获得每个topic+partition的最新的offset，从而定义每个batch的offset的范围。当处理数据的job启动时，就会使用Kafka的简单consumer api来获取Kafka指定offset范围的数据

具体来说，主要有以下3点：

1. 简化并行读取：

如果要读取多个partition，不需要创建多个输入DStream然后对它们进行union操作。Spark会创建跟Kafka partition一样多的RDD partition，并且会并行从Kafka中读取数据。所以在Kafka partition和RDD partition之间，有一个一对一的映射关系

1. 高性能：

如果要保证零数据丢失，在基于receiver的方式中，需要开启WAL机制。这种方式其实效率低下，因为数据实际上被复制了两份，Kafka自己本身就有高可靠的机制，会对数据复制一份，而这里又会复制一份到WAL中。而基于direct的方式，不依赖Receiver，不需要开启WAL机制，只要Kafka中作了数据的复制，那么就可以通过Kafka的副本进行恢复。

1. 可靠的事务机制：

基于receiver的方式，是使用Kafka的高阶API来在ZooKeeper中保存消费过的offset的。这是消费Kafka数据的传统方式。这种方式配合着WAL机制可以保证数据零丢失的高可靠性，但是却无法保证数据被处理一次且仅一次，可能会处理两次。因为Spark和ZooKeeper之间可能是不同步的。 基于direct的方式，使用kafka的简单api，Spark Streaming自己就负责追踪消费的offset，可以保证output操作和保存offset操作是同步的(两个操作放入到一个事务中)，因此可以保证数据是消费一次且仅消费一次。

## 7.2 设置Checkpoint的最佳时间间隔是多少

批处理时间的5 - 10倍

## 7.3 checkpoint中保存了哪些数据？

1. 元数据checkpoint：
2. 配置信息——创建Spark Streaming应用程序的配置信息，比如SparkConf中的信息。
3. DStream的操作信息——定义了Spark Stream应用程序的计算逻辑
4. 未处理的batch信息——那些job正在排队，还没处理的batch信息。
5. 数据checkpoint

将实时计算过程中产生的RDD的数据保存到可靠的存储系统中。比如状态RDD

一句话概括，元数据checkpoint主要是为了从driver失败中进行恢复；而RDD checkpoint主要是为了，使用到有状态的transformation操作时，能够在其生产出的数据丢失时，进行快速的失败恢复。

## 7.4 是否一定要设置Checkpoint

如果没有使用带状态的算子，程序没有强制要求设置checkpoint时就不要设置checkpoint，因为会影响流式计算的吞吐量

## 7.5 遇到流量激增时如何处理？比如之前每个batch1W条，活动时突然变成每个batch100W条。我们要怎么做？

基于Direct方式下设置spark.streaming.backpressure.enabled=true即可

## 7.6 我们为什么没有将offset保存到ZK中？

## 7.7 如何实现精确一次消费的语义

使用Direct方式，手动控制offset，将保存offset操作和output操作写入到一个事务。

## 7.8 如何优雅停止SparkStreaming程序

设置spark.streaming.stopGracefullyOnShutdown为true，关闭程序的时侯使用kill -SIGTERM <AM-PID>方式

## 7.9 Kafka副本数为3，最大能容错几个副本？是否一定能够容错这些？

最大容错2个逼本，当有副本不在ISR中时，就不能容错2个了

**7.10 什么情况下副本不在ISR中？**

0.9版本之后，follwer落后leader replica.log.time.max.ms 的时间后，即不在ISR中，默认为10S

## 7.11 如何保证某些消息在Kafka中可以全局有序的消费？

为这部分消息设置key，所以相同的key会进入同一个分区

通过生产者，指定这部分消息发送到同一个partition

如果问题变成保证某个Topic的消息要全局有序消费，那只能设置一个分区了