**spark电商分析项目实战**

# 1 集群环境搭建

## 1.1 集群基础环境搭建

### 1.1.1 CentOS 6.4安装

1、使用课程提供的镜像：CentOS-6.4-i386-minimal.iso。

2、创建虚拟机：打开Virtual Box，点击“新建”按钮；输入虚拟机名称为sparkproject1，选择操作系统为Linux，选择版本为Red Hat；分配1024MB内存；后面所有选项全部用默认的设置；注意，在Virtual disk file location and size中，一定要自己选择一个目录来存放虚拟机文件；最后点击“create”按钮，开始创建虚拟机。

3、设置网卡（桥接网卡）：选择创建好的虚拟机，点击“设置”按钮，在网络一栏中，连接方式中，选择“Bridged Adapter”，即桥接网卡。

4、安装CentOS 6.4操作系统：选择创建好的虚拟机，点击“开始”按钮；选择安装介质（即本地的CentOS 6.4镜像文件）；选择第一项开始安装-Skip-欢迎界面Next-选择默认语言-Baisc Storage Devices-Yes, discard any data-主机名:sparkproject1-选择时区-设置初始密码为hadoop-Replace Existing Linux System-Write changes to disk-开始安装。

5、安装完以后，会提醒你要重启一下，就是reboot，reboot就可以了。

### 1.1.2 网络配置

1、先临时性设置虚拟机ip地址：ifconfig eth0 192.168.1.110，在/etc/hosts文件中配置本地ip到host的映射

2、配置windows主机上的hosts文件：C:\Windows\System32\drivers\etc\hosts，192.168.1.110 sparkproject1

3、使用SecureCRT从windows上连接虚拟机

4、永久性配置CentOS网络

vi /etc/sysconfig/network-scripts/ifcfg-eth0

DEVICE=eth0

TYPE=Ethernet

ONBOOT=yes

BOOTPROTO=static

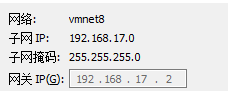
IPADDR=192.168.1.110

NETMASK=255.255.255.0

GATEWAY=192.168.1.1

**！！最后三项要根据dhcp模式下的情况进行配置**

**网关设置要看VM的设置**



5、重启网卡

service network restart

### 1.1.3 关闭防火墙

service iptables stop

service ip6tables stop

service iptables status

service ip6tables status

chkconfig iptables off

chkconfig ip6tables off

vi /etc/selinux/config

SELINUX=disabled

在win7的控制面板中，关闭windows的防火墙！如果不关闭防火墙的话，就怕，可能win7和虚拟机直接无法ping通！

### 1.1.4 配置DNS服务器、替换repo文件、配置yum

1、配置DNS服务器

vi /etc/resolv.conf

nameserver 61.139.2.69

nameserver 114.114.114.114

nameserver 8.8.8.8

ping www.baidu.com

1、修改repo

将CentOS6-Base-163.repo上传到CentOS中的/usr/local目录下

cd /etc/yum.repos.d/

rm -rf \*

mv 自己的repo文件移动到/etc/yum.repos.d/目录中：cp /usr/local/CentOS6-Base-163.repo .

修改repo文件，把所有gpgcheck属性修改为0

【阿里云yum源】

2、配置yum

yum clean all

yum makecache

yum install telnet

即使更换了ip地址，重启网卡，可能还是联不通网。那么可以先将IPADDR、NETMASK、GATEWAY给删除，将BOOTPROTO改成dhcp。然后用service network restart重启网卡。此时linux会自动给分配一个ip地址，用ifconfig查看分配的ip地址。然后再次按照之前说的，配置网卡，将ip改成自动分配的ip地址。最后再重启一次网卡。

### 1.1.5 添加普通用户hadoop

用户管理:

1.新建用户

useradd user1

2.删除用户

userdel -r user1

3.修改密码

1)passwd user1

2)echo 123|passwd --stdin user1

4.查看用户

id user1

**######**

# useradd hadoop

# passwd hadoop

# vi /etc/sudoers

hadoop ALL=(ALL) ALL

### 1.1.6 安装JDK 1.7

1. 解压

$ tar -zxvf jdk-7u65-linux-i586.tar.gz

$ mv jdk1.7.0\_65 jdk

2.配置jdk相关的环境变量

sudo vi /etc/profile

## JAVA\_HOME

export JAVA\_HOME=/home/hadoop/app/jdk

export PATH=$PATH:$JAVA\_HOME/bin

source /etc/profile

3.测试jdk安装是否成功：java -version

4.rm -f /etc/udev/rules.d/70-persistent-net.rules

### 1.1.7 安装第二台和第三台虚拟机

1、安装上述步骤，再安装两台一模一样环境的虚拟机。

2、另外两台机器的hostname分别设置为sparkproject2和sparkproject3即可

3、在安装的时候，另外两台虚拟机的centos镜像文件必须重新拷贝一份，放在新的目录里，使用各自自己的镜像文件。

4、虚拟机的硬盘文件也必须重新选择一个新的目录，以更好的区分。

5、安装好之后，记得要在三台机器的/etc/hosts文件中，配置全三台机器的ip地址到hostname的映射，而不能只配置本机，这个很重要！

6、在windows的hosts文件中也要配置全三台机器的ip地址到hostname的映射。

### 1.1.8 配置集群ssh免密码登录

1、在三台机器的/etc/hosts文件中，都配置对三台机器的ip-hostname的映射

2、首先在三台机器上配置对本机的ssh免密码登录

生成本机的公钥，过程中不断敲回车即可，ssh-keygen命令默认会将公钥放在/root/.ssh目录下

ssh-keygen -t rsa

将公钥复制为authorized\_keys文件，此时使用ssh连接本机就不需要输入密码了

cd /root/.ssh

cp id\_rsa.pub authorized\_keys

3、接着配置三台机器互相之间的ssh免密码登录

使用ssh-copy-id -i spark命令将本机的公钥拷贝到指定机器的authorized\_keys文件中（方便好用）

## 1.2 hadoop集群搭建

### 1.2.1 解压hadoop

$ tar -zxvf hadoop-2.5.0-cdh5.3.6.tar.gz

$ rm -rf hadoop-2.5.0-cdh5.3.6.tar.gz

$ mv hadoop-2.5.0-cdh5.3.6 hadoop

### 1.2.2 配置环境变量

vim /etc/profile

## HADOOP\_HOME

export HADOOP\_HOME=/home/hadoop/app/hadoop

export PATH=$PATH:$HADOOP\_HOME/bin:$HADOOP\_HOME/sbin

$ source /etc/profile

### 1.2.3 创建/home/hadoop/app/hadoop目录

mkdir /home/hadoop/app/hadoop/data

### 1.2.4 配置文件

## 修改hadoop-env.sh

export JAVA\_HOME=/home/hadoop/app/jdk

## 修改core-site.xml

<property>

<name>fs.default.name</name>

<value>hdfs://sparkproject1:9000</value>

</property>

**## 修改hdfs-site.xml**

<property>

<name>dfs.name.dir</name>

<value>/home/hadoop/app/hadoop/data/namenode</value>

</property>

<property>

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

<value>/home/hadoop/app/hadoop/data/datanode</value>

</property>

<property>

<name>dfs.tmp.dir</name>

<value>/home/hadoop/app/hadoop/data/tmp</value>

</property>

<property>

<name>dfs.replication</name>

<value>2</value>

</property>

**## 修改mapred-site.xml**

<property>

<name>mapreduce.framework.name</name>

<value>yarn</value>

</property>

## 修改yarn-site.xml

<property>

<name>yarn.resourcemanager.hostname</name>

<value>sparkproject1</value>

</property>

<property>

<name>yarn.nodemanager.aux-services</name>

<value>mapreduce\_shuffle</value>

</property>

## 修改slaves文件

sparkproject2

sparkproject3

### 1.2.5 在另外两台机器上搭建hadoop

1）使用如上配置在另外两台机器上搭建hadoop，可以使用scp命令将sparkproject1上面的hadoop安装包和~/.bashrc配置文件都拷贝过去。

scp -r hadoop hadoop@sparkproject2:/home/hadoop/app

scp -r hadoop hadoop@sparkproject3:/home/hadoop/app

2）要记得对/etc/profile文件进行source，以让它生效。

3）记得在sparkproject2和sparkproject3的/home/hadoop/app/hadoop目录下创建data目录。

### 1.2.6 启动hdfs集群

1、格式化namenode：在sparkproject1上执行以下命令，hdfs namenode -format

2、启动hdfs集群：start-dfs.sh  
3、验证启动是否成功：jps、50070端口

sparkproject1：namenode、secondarynamenode

sparkproject2：datanode

sparkproject3：datanode

1. hdfs dfs -put hello.txt /hello.txt

### 1.2.7 启动yarn集群

1、启动yarn集群：start-yarn.sh  
2、验证启动是否成功：jps、8088端口

sparkproject1：resourcemanager、nodemanager ##实际上1是没有nodemanager

sparkproject2：nodemanager

sparkproject3：nodemanager

## 1.3 hive集群搭建

### 1.3.1 安装hive包

1 解压缩hive安装包

tar -zxvf hive-0.13.1-cdh5.3.6.tar.gz

$ rm -rf hive-0.13.1-cdh5.3.6.tar.gz

2 重命名hive目录：

mv hive-0.13.1-cdh5.3.6 hive

3 配置hive相关的环境变量

$ sudo vi /etc/profile

## HIVE\_HOME

export HIVE\_HOME=/home/hadoop/app/hive

export PATH=$PATH:$HIVE\_HOME/bin

$ source /etc/profile

### 1.3.2 安装mysql

1、在sparkproject1上安装mysql。

2、使用yum安装mysql server。

sudo yum install -y mysql-server

sudo service mysqld start

sudo chkconfig mysqld on

3、使用yum安装mysql connector

sudo yum install -y mysql-connector-java

4、将mysql connector拷贝到hive的lib包中

cp mysql-connector-java-5.1.17.jar /home/hadoop/app/hive/lib

5、在mysql上创建hive元数据库，创建hive账号，并进行授权

create database if not exists hive\_metadata;

grant all privileges on hive\_metadata.\* to 'hive'@'%' identified by 'hive';

grant all privileges on hive\_metadata.\* to 'hive'@'localhost' identified by 'hive';

grant all privileges on hive\_metadata.\* to 'hive'@'sparkproject1' identified by 'hive';

flush privileges;

use hive\_metadata;

### 1.3.3 配置hive-site.xml

mv hive-default.xml.template hive-site.xml

<configuration>

<property>

<name>javax.jdo.option.ConnectionURL</name>

<value>jdbc:mysql://sparkproject1:3306/hive\_metadata?createDatabaseIfNotExist=true</value>

</property>

<property>

<name>javax.jdo.option.ConnectionDriverName</name>

<value>com.mysql.jdbc.Driver</value>

</property>

<property>

<name>javax.jdo.option.ConnectionUserName</name>

<value>hive</value>

</property>

<property>

<name>javax.jdo.option.ConnectionPassword</name>

<value>hive</value>

</property>

</configuration>

### 1.3.4 配置hive-env.sh和hive-config.sh

mv hive-env.sh.template hive-env.sh

vi hive-env.sh

HADOOP\_HOME=/home/hadoop/app/hadoop

export HIVE\_CONF\_DIR=/home/hadoop/app/hive/conf

export HIVE\_AUX\_JARS\_PATH=/home/hadoop/app/hive/bin

vi /home/hadoop/app/hive/bin/hive-config.sh

export JAVA\_HOME=/home/hadoop/app/jdk

export HIVE\_HOME=/home/hadoop/app/hive

export HADOOP\_HOME=/home/hadoop/app/hadoop

### 1.3.5 验证hive是否安装成功

直接输入hive命令，可以进入hive命令行

create table users(id int, name string)

load data local inpath '/home/hadoop/data/hive/users.txt' into table users

select name from users

### 1.3.6 hive显示当前数据库

在hive-site.xml添加

<property>

<name>hive.cli.print.header</name>

<value>true</value>

<description>Whether to print the names of the columns in query output.</description>

</property>

<property>

<name>hive.cli.print.current.db</name>

<value>true</value>

<description>Whether to include the current database in the Hive prompt.</description>

</property>

## 1.4 zookeeper集群

### 1.4.1 安装ZooKeeper包

1. 解压

$ tar -zxvf zookeeper-3.4.5-cdh5.3.6.tar.gz

$ rm -rf zookeeper-3.4.5-cdh5.3.6.tar.gz

$ mv zookeeper-3.4.5-cdh5.3.6 zookeeper

2.配置zookeeper相关的环境变量

$ sudo vi /etc/profile

## ZOOKEEPER\_HOME

export ZOOKEEPER\_HOME=/home/hadoop/app/zookeeper

export PATH=$PATH:$ZOOKEEPER\_HOME/bin

$ source /etc/profile

### 1.4.2 配置zoo.cfg

1 修改文件名

cd zk/conf

mv zoo\_sample.cfg zoo.cfg

vi zoo.cfg

修改：dataDir=/home/hadoop/app/zookeeper/data

在zookeeper/目录下创建data目录

[hadoop@sparkproject1 zookeeper]$ mkdir zookeeper

在zoo.cfg增加：

server.0=sparkproject1:2888:3888

server.1=sparkproject2:2888:3888

server.2=sparkproject3:2888:3888

2 修改data/myid

vi myid

0

### 1.4.3 配置另外两台Linux

1. 在另外两个节点上按照上述步骤配置ZooKeeper，使用scp将zk和.bashrc拷贝到spark2和spark3上即可。

scp -r zookeeper hadoop@sparkproject2:/home/hadoop/app

scp -r zookeeper hadoop@sparkproject3:/home/hadoop/app

2、唯一的区别是sparkproject2和sparkproject3的标识号分别设置为1和2。

### 1.4.4 启动zookeeper

1.基本命令

$ bin/zkServer.sh start

$ zkServer.sh status

$ zkServer.sh stop

2.检查ZooKeeper状态：zkServer.sh status，应该是一个leader，两个follower

3.jps：检查三个节点是否都有QuromPeerMain进程。

## 1.5 kafka集群搭建

实时数据从kafka集群里面读。kafka是分布式的消息队列。

### 1.5.1 安装scala

1. 解压缩：

$ tar -zxvf scala-2.11.4.tgz

$ rm -rf scala-2.11.4.tgz

$ mv scala-2.11.4 scala

2.配置scala相关的环境变量

$ sudo vi /etc/profile

#SCALA\_HOME

export SCALA\_HOME=/home/hadoop/app/scala

export PATH=$PATH:$SCALA\_HOME/bin

$ source /etc/profile

1. 查看scala是否安装成功：

$ scala -version

1. 按照上述步骤在sparkproject2和sparkproject3机器上都安装好scala。使用scp将scala和/etc/profile拷贝到另外两台机器上即可。

$ scp -r scala hadoop@sparkproject2:/home/hadoop/app

$ scp -r scala hadoop@sparkproject3:/home/hadoop/app

### 1.5.2 安装Kafka包

1.解压包

$ tar -zxvf kafka\_2.9.2-0.8.1.tgz

$ rm -rf kafka\_2.9.2-0.8.1.tgz

$ mv kafka\_2.9.2-0.8.1 kafka

2.配置kafka

$ vim kafka/config/server.properties

修改：

broker.id：依次增长的整数，0、1、2，集群中Broker的唯一id

修改：

zookeeper.connect=**192.168.0.150:2181,192.168.0.151:2181,192.168.0.152:2181**

修改：

port=**1**9092

host.name=**sparkproject1**

log.dirs=**/home/hadoop/app/kafka/kafkalogs**

添加：

message.max.byte=5048576

default.replication.factor=2

replica.fetch.max.bytes=5048576

1. 安装slf4j

$ sudo yum install unzip

$ unzip slf4j-1.7.6.zip

把slf4j中的slf4j-nop-1.7.6.jar复制到kafka的libs目录下面

$ cp slf4j-1.7.6/slf4j-nop-1.7.6.jar kafka/libs/

### 1.5.3 搭建kafka集群

1.按照上述步骤在另外两台机器分别安装kafka。用scp把kafka拷贝到sparkproject2和sparkproject3即可。

$ scp -r kafka hadoop@sparkproject2:/home/hadoop/app

$ scp -r kafka hadoop@sparkproject3:/home/hadoop/app

2.唯一区别的，就是server.properties中的broker.id，要设置为1和2

### 1.5.4 启动kafka集群

1、解决kafka Unrecognized VM option 'UseCompressedOops'问题

vi /home/hadoop/app/kafka/bin/kafka-run-class.sh

if [ -z "$KAFKA\_JVM\_PERFORMANCE\_OPTS" ]; then

KAFKA\_JVM\_PERFORMANCE\_OPTS="-server **-XX:+UseCompressedOops** -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -XX:+CMSClassUnloadingEnabled -XX:+CMSScavengeBeforeRemark -XX:+DisableExplicitGC -Djava.awt.headless=true"

fi

去掉-XX:+UseCompressedOops即可

1. 在三台机器上的kafka目录下，分别执行以下命令：

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

[bin]$ ./kafka-server-start.sh -daemon ../config/server.properties

1. 使用jps检查启动是否成功

jps

### 1.5.5 测试kafka集群

使用基本命令检查kafka是否搭建成功

在第一台机器：

$ bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 2 --partitions 1 --topic test01

$ bin/kafka-console-producer.sh --broker-list sparkproject1:19092 --topic test01

及时输入：helloworld

在第二台机器：

bin/kafka-console-consumer.sh --zookeeper localhost:2181 --topic test01 --from-beginning

及时输出：helloworld

## 1.6 安装安装flume

flume是离线日志收集的工具。

### 1.6.1 安装包

1. 解压

tar -zxvf flume-ng-1.5.0-cdh5.3.6.tar.gz

$ rm -rf flume-ng-1.5.0-cdh5.3.6.tar.gz

$ mv apache-flume-1.5.0-cdh5.3.6-bin flume

2.配置环境变量

$ sudo vim /etc/profile

#FLUME\_HOME

export FLUME\_HOME=/home/hadoop/app/flume

export FLUME\_CONF\_DIR=$FLUME\_HOME/conf

export PATH=$PATH:$FLUME\_HOME/bin

$ source /etc/profile

### 1.6.2 修改flume配置文件

#agent1表示代理名称

agent1.sources=source1

agent1.sinks=sink1

agent1.channels=channel1

#配置source1

agent1.sources.source1.type=spooldir

agent1.sources.source1.spoolDir=/home/hadoop/app/flume/logs

agent1.sources.source1.channels=channel1

agent1.sources.source1.fileHeader = false

agent1.sources.source1.interceptors = i1

agent1.sources.source1.interceptors.i1.type = timestamp

#配置channel1

agent1.channels.channel1.type=file

agent1.channels.channel1.checkpointDir=/home/hadoop/app/flume/logs\_tmp\_cp

agent1.channels.channel1.dataDirs=/home/hadoop/app/flume/logs\_tmp

#配置sink1

agent1.sinks.sink1.type=hdfs

agent1.sinks.sink1.hdfs.path=hdfs://sparkproject1:9000/logs

agent1.sinks.sink1.hdfs.fileType=DataStream

agent1.sinks.sink1.hdfs.writeFormat=TEXT

agent1.sinks.sink1.hdfs.rollInterval=1

agent1.sinks.sink1.channel=channel1

agent1.sinks.sink1.hdfs.filePrefix=%Y-%m-%d

### 1.6.3 创建需要的文件夹

本地文件夹：mkdir /home/hadoop/app/flume/logs

HDFS文件夹：hdfs dfs -mkdir /logs

### 1.6.4 启动flume-agent

flume-ng agent -n agent1 -c conf -f /home/hadoop/app/flume/conf/flume-conf.properties -Dflume.root.logger=DEBUG,console

$ hdfs dfs -lsr /logs

### 1.6.5 测试flume

新建一份文件，移动到flume/logs目录下，flume就会自动上传到HDFS的/logs目录中

## 1.7 Spark 1.5.1客户端安装以及基于YARN的提交模式

### 1.7.1 安装客户端

1 解压spark

tar -zxvf spark-1.5.1-bin-hadoop2.4.tgz

$ mv spark-1.5.1-bin-hadoop2.4 spark

2 配置spark环境变量

sudo vim /etc/profile

export SPARK\_HOME=/home/hadoop/app/spark

export PATH=$PATH:$SPARK\_HOME/bin

export CLASSPATH=$CLASSPATH:$JAVA\_HOME/lib:$JAVA\_HOME/jre/lib:$PATH

source /etc/profile

### 1.7.2 修改spark-env.sh文件

spark/conf目录下

cp spark-env.sh.template spark-env.sh

修改spark-env.sh

$ vi spark-env.sh

export JAVA\_HOME=/home/hadoop/app/jdk

export SCALA\_HOME=/home/hadoop/app/scala

export HADOOP\_HOME=/home/hadoop/app/hadoop

export HADOOP\_CONF\_DIR=/home/hadoop/app/hadoop/etc/hadoop

### 1.7.3 用yarn-client模式提交spark作业

bin/spark-submit \

--class org.apache.spark.example.JavaSparkPi \

--master yarn-client \

--num-executors 1 \

--driver-memory 300m \

--executor-memory 300m \

--executor-cores 1 \

/home/hadoop/app/spark/lib/spark-examples-1.5.1-hadoop2.4.0.jar

### 1.7.4 用yarn-cluster模式提交spark作业

bin/spark-submit \

--class org.apache.spark.example.JavaSparkPi \

--master yarn-cluster \

--num-executors 1 \

--driver-memory 10m \

--executor-memory 10m \

--executor-cores 1 \

/home/hadoop/app/spark/lib/spark-examples-1.5.1-hadoop2.4.0.jar \

# 2 用户访问session分析模块

## 2.1 模块介绍

### 2.1.1 模块的目标

模块的目标：对用户访问session进行分析

1、可以根据使用者指定的某些条件，筛选出指定的一些用户（有特定年龄、职业、城市）；

2、对这些用户在指定日期范围内发起的session，进行聚合统计，比如，统计出访问时长在0~3s的session占总session数量的比例；

3、按时间比例，比如一天有24个小时，其中12:00~13:00的session数量占当天总session数量的50%，当天总session数量是10000个，那么当天总共要抽取1000个session，ok，12:00~13:00的用户，就得抽取1000\*50%=500。而且这500个需要随机抽取。

4、获取点击量、下单量和支付量都排名10的商品种类

5、获取top10的商品种类的点击数量排名前10的session

6、开发完毕了以上功能之后，需要进行大量、复杂、高端、全套的性能调优（大部分性能调优点，都是本人在实际开发过程中积累的经验，基本都是全网唯一）

7、十亿级数据量的troubleshooting（故障解决）的经验总结

8、数据倾斜的完美解决方案（全网唯一，非常高端，因为数据倾斜往往是大数据处理程序的性能杀手，很多人在遇到的时候，往往没有思路）

9、使用mock（模拟）的数据，对模块进行调试、运行和演示效果

### 2.1.2 在实际企业项目中的使用架构

1、J2EE的平台（美观的前端页面），通过这个J2EE平台可以让使用者，提交各种各样的分析任务，其中就包括一个模块，就是用户访问session分析模块；可以指定各种各样的筛选条件，比如年龄范围、职业、城市等等。。

2、J2EE平台接收到了执行统计分析任务的请求之后，会调用底层的封装了spark-submit的shell脚本（Runtime、Process），shell脚本进而提交我们编写的Spark作业。

3、Spark作业获取使用者指定的筛选参数，然后运行复杂的作业逻辑，进行该模块的统计和分析。

4、Spark作业统计和分析的结果，会写入MySQL中，指定的表

5、最后，J2EE平台，使用者可以通过前端页面（美观），以表格、图表的形式展示和查看MySQL中存储的该统计分析任务的结果数据。

（讲师本人在实际企业中，是做大数据平台的，所以上面一整套，除了前端页面不用我做，当然了，偶尔可能也需要做一些前端，从J2EE到Spark到MySQL的性能调优，实际上都是我的工作

但是在本套课程中，我们只做第三和第四点，不会做所有的部分，因为时间有限，我们只关注最重要的Spark，就是开发Spark程序。实际上，能够将Spark程序开发的很好，已经非常不错了。

如果没有J2EE基础的话，那么也不用过多去关系J2EE平台的开发，因为术业有专攻，仅仅是精通Spark，就足以去找一份好工作，在项目中就专注开发Spark就可以了）

### 2.2.3 需求分析

1、按条件筛选session

2、统计出符合条件的session中，访问时长在1s~3s、4s~6s、7s~9s、10s~30s、30s~60s、1m~3m、3m~10m、10m~30m、30m以上各个范围内的session占比；访问步长在1~3、4~6、7~9、10~30、30~60、60以上各个范围内的session占比

3、在符合条件的session中，按照时间比例随机抽取1000个session

4、在符合条件的session中，获取点击、下单和支付数量排名前10的品类

5、对于排名前10的品类，分别获取其点击次数排名前10的session

声明第一点，上节课的话，其实我们就已经进入了正规的企业级大数据项目的开发流程，第一个步骤就是数据调研（就是对底层基于的基础数据的表结构进行调研、分析和研究）；然后本节课，相当于是项目开发流程的第二个步骤，就是需求分析（在实际的企业中，需求分析，可能会比这里更加复杂很多；在互联网企业中，需求分析，首先就是要跟PM，就是产品经理，也就是负责设计你开发的大数据平台产品的人，去大量开会，去沟通需求的细节；此外，你自己还得根据产品经理编写的需求文档，可能还会自己设计一些产品原型图出来，让你看，去看，去研究；然后第三点，可能还需要作为一个项目的技术leader，去跟你的项目组内的成员，去讲解和讨论需求，要确保组内所有成员，都对需求清晰的理解了）

1、按条件筛选session

搜索过某些关键词的用户、访问时间在某个时间段内的用户、年龄在某个范围内的用户、职业在某个范围内的用户、所在某个城市的用户，发起的session。找到对应的这些用户的session，也就是我们所说的第一步，按条件筛选session。

这个功能，就最大的作用就是灵活。也就是说，可以让使用者，对感兴趣的和关系的用户群体，进行后续各种复杂业务逻辑的统计和分析，那么拿到的结果数据，就是只是针对特殊用户群体的分析结果；而不是对所有用户进行分析的泛泛的分析结果。比如说，现在某个企业高层，就是想看到用户群体中，28~35岁的，老师职业的群体，对应的一些统计和分析的结果数据，从而辅助高管进行公司战略上的决策制定。

2、统计出符合条件的session中，访问时长在1s~3s、4s~6s、7s~9s、10s~30s、30s~60s、1m~3m、3m~10m、10m~30m、30m以上各个范围内的session占比；访问步长在1~3、4~6、7~9、10~30、30~60、60以上各个范围内的session占比

session访问时长，也就是说一个session对应的开始的action，到结束的action，之间的时间范围；还有，就是访问步长，指的是，一个session执行期间内，依次点击过多少个页面，比如说，一次session，维持了1分钟，那么访问时长就是1m，然后在这1分钟内，点击了10个页面，那么session的访问步长，就是10.

比如说，符合第一步筛选出来的session的数量大概是有1000万个。那么里面，我们要计算出，访问时长在1s~3s内的session的数量，并除以符合条件的总session数量（比如1000万），比如是100万/1000万，那么1s~3s内的session占比就是10%。依次类推，这里说的统计，就是这个意思。

这个功能的作用，其实就是，可以让人从全局的角度看到，符合某些条件的用户群体，使用我们的产品的一些习惯。比如大多数人，到底是会在产品中停留多长时间，大多数人，会在一次使用产品的过程中，访问多少个页面。那么对于使用者来说，有一个全局和清晰的认识。

3、在符合条件的session中，按照时间比例随机抽取1000个session

这个按照时间比例是什么意思呢？随机抽取本身是很简单的，但是按照时间比例，就很复杂了。比如说，这一天总共有1000万的session。那么我现在总共要从这1000万session中，随机抽取出来1000个session。但是这个随机不是那么简单的。需要做到如下几点要求：首先，如果这一天的12:00~13:00的session数量是100万，那么这个小时的session占比就是1/10，那么这个小时中的100万的session，我们就要抽取1/10 \* 1000 = 100个。然后再从这个小时的100万session中，随机抽取出100个session。以此类推，其他小时的抽取也是这样做。

这个功能的作用，是说，可以让使用者，能够对于符合条件的session，按照时间比例均匀的随机采样出1000个session，然后观察每个session具体的点击流/行为，比如先进入了首页、然后点击了食品品类、然后点击了雨润火腿肠商品、然后搜索了火腿肠罐头的关键词、接着对王中王火腿肠下了订单、最后对订单做了支付。

之所以要做到按时间比例随机采用抽取，就是要做到，观察样本的公平性。

4、在符合条件的session中，获取点击、下单和支付数量排名前10的品类

什么意思呢，对于这些session，每个session可能都会对一些品类的商品进行点击、下单和支付等等行为。那么现在就需要获取这些session点击、下单和支付数量排名前10的最热门的品类。也就是说，要计算出所有这些session对各个品类的点击、下单和支付的次数，然后按照这三个属性进行排序，获取前10个品类。

这个功能，很重要，就可以让我们明白，就是符合条件的用户，他最感兴趣的商品是什么种类。这个可以让公司里的人，清晰地了解到不同层次、不同类型的用户的心理和喜好。

5、对于排名前10的品类，分别获取其点击次数排名前10的session

这个就是说，对于top10的品类，每一个都要获取对它点击次数排名前10的session。

这个功能，可以让我们看到，对某个用户群体最感兴趣的品类，各个品类最感兴趣最典型的用户的session的行为。

### 2.2.4 技术方案设计

1、通过底层数据聚合，来减少spark作业处理数据量，从而提升spark作业的性能（从根本上提升spark性能的技巧）

2、自定义Accumulator实现复杂分布式计算的技术

3、Spark按时间比例随机抽取算法

4、Spark自定义key二次排序技术

5、Spark分组取TopN算法

6、通过Spark的各种功能和技术点，进行各种聚合、采样、排序、取TopN业务的实现

### 2.2.5 数据表设计

本节课，是进入企业级大数据项目开发流程的数据设计环节。在进行完了数据调研、需求分析、技术实现方案，进行数据设计。数据设计，往往包含两个环节，第一个呢，就是说，我们的上游数据，就是数据调研环节看到的项目基于的基础数据，是否要针对其开发一些Hive ETL，对数据进行进一步的处理和转换，从而让我们能够更加方便的和快速的去计算和执行spark作业；第二个，就是要设计spark作业要保存结果数据的业务表的结构，从而让J2EE平台可以使用业务表中的数据，来为使用者展示任务执行结果。

在本项目中，我们所有的数据设计环节，只会涉及第二个，不会涉及第一个。因为我们为了突出课程重点，也就是spark。所以主要还是集中在spark上面，就不要花时间去做Hive ETL了。设计MySQL中的业务表的结构。

mysql> create database sparkproject;

mysql> use sparkporject;

第一表：session\_aggr\_stat表，存储第一个功能，session聚合统计的结果

CREATE TABLE `session\_aggr\_stat` (

`task\_id` int(11) NOT NULL,

`session\_count` int(11) DEFAULT NULL,

`1s\_3s` double DEFAULT NULL,

`4s\_6s` double DEFAULT NULL,

`7s\_9s` double DEFAULT NULL,

`10s\_30s` double DEFAULT NULL,

`30s\_60s` double DEFAULT NULL,

`1m\_3m` double DEFAULT NULL,

`3m\_10m` double DEFAULT NULL,

`10m\_30m` double DEFAULT NULL,

`30m` double DEFAULT NULL,

`1\_3` double DEFAULT NULL,

`4\_6` double DEFAULT NULL,

`7\_9` double DEFAULT NULL,

`10\_30` double DEFAULT NULL,

`30\_60` double DEFAULT NULL,

`60` double DEFAULT NULL,

PRIMARY KEY (`task\_id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8

第二个表：session\_random\_extract表，存储我们的按时间比例随机抽取功能抽取出来的1000个session

USE sparkproject;

CREATE TABLE `session\_random\_extract` (

`task\_id` int(11) NOT NULL,

`session\_id` varchar(255) DEFAULT NULL,

`start\_time` varchar(50) DEFAULT NULL,

`end\_time` varchar(50) DEFAULT NULL,

`search\_keywords` varchar(255) DEFAULT NULL,

PRIMARY KEY (`task\_id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8

第三个表：top10\_category表，存储按点击、下单和支付排序出来的top10品类数据

USE sparkproject;

CREATE TABLE `top10\_category` (

`task\_id` int(11) NOT NULL,

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

`click\_count` int(11) DEFAULT NULL,

`order\_count` int(11) DEFAULT NULL,

`pay\_count` int(11) DEFAULT NULL,

PRIMARY KEY (`task\_id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8

第四个表：top10\_category\_session表，存储top10每个品类的点击top10的session

USE sparkproject;

CREATE TABLE `top10\_category\_session` (

`task\_id` int(11) NOT NULL,

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

`session\_id` varchar(255) DEFAULT NULL,

`click\_count` int(11) DEFAULT NULL,

PRIMARY KEY (`task\_id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8

最后一张表：session\_detail，用来存储随机抽取出来的session的明细数据、top10品类的session的明细数据

USE sparkproject;

CREATE TABLE `session\_detail` (

`task\_id` int(11) NOT NULL,

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

`session\_id` varchar(255) DEFAULT NULL,

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

`action\_time` varchar(255) DEFAULT NULL,

`search\_keyword` varchar(255) DEFAULT NULL,

`click\_category\_id` int(11) DEFAULT NULL,

`click\_product\_id` int(11) DEFAULT NULL,

`order\_category\_ids` varchar(255) DEFAULT NULL,

`order\_product\_ids` varchar(255) DEFAULT NULL,

`pay\_category\_ids` varchar(255) DEFAULT NULL,

`pay\_product\_ids` varchar(255) DEFAULT NULL,

PRIMARY KEY (`task\_id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8

额外的一张表：task表，用来存储J2EE平台插入其中的任务的信息

CREATE TABLE `task` (

`task\_id` int(11) NOT NULL AUTO\_INCREMENT,

`task\_name` varchar(255) DEFAULT NULL,

`create\_time` varchar(255) DEFAULT NULL,

`start\_time` varchar(255) DEFAULT NULL,

`finish\_time` varchar(255) DEFAULT NULL,

`task\_type` varchar(255) DEFAULT NULL,

`task\_status` varchar(255) DEFAULT NULL,

`task\_param` text,

PRIMARY KEY (`task\_id`)

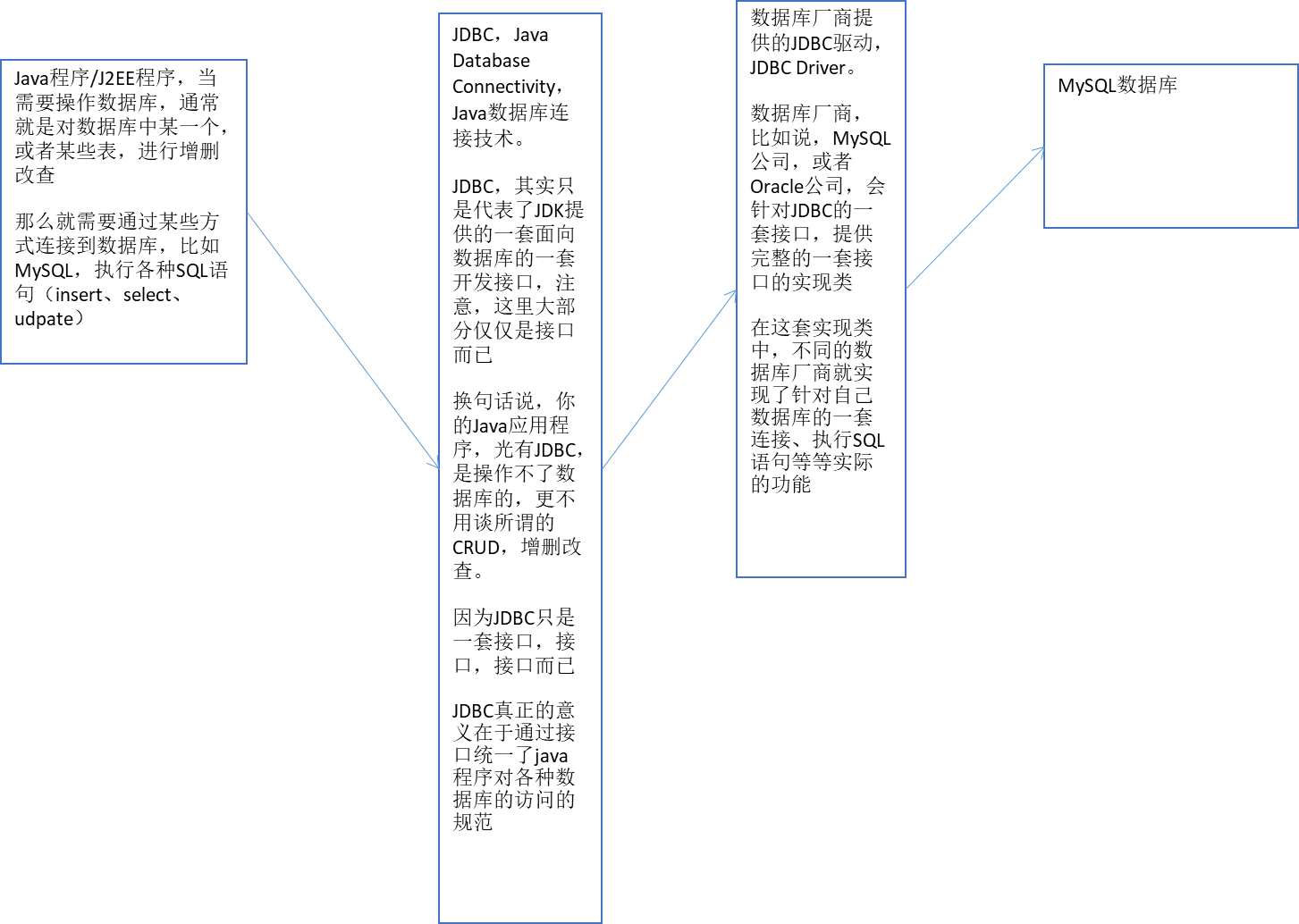
) ENGINE=InnoDB AUTO\_INCREMENT=0 DEFAULT CHARSET=utf8

在数据设计以后，就正式进入一个漫长的环节，就是编码实现阶段，coding阶段。在编码实现阶段，每开发完一个功能，其实都会走后续的两个环节，就是本地测试和生产环境测试。

大家需要在windows上面，自己安装MySQL数据库。然后本地测试的时候，将数据插入本地的MySQL中。

接下来，就是在完成了数据调研、需求分析、技术方案设计、数据设计以后，正式进入编码实现和功能测试阶段。最后才是性能调优阶段。

### 2.1.8 JDBC原理介绍及增删改查示范

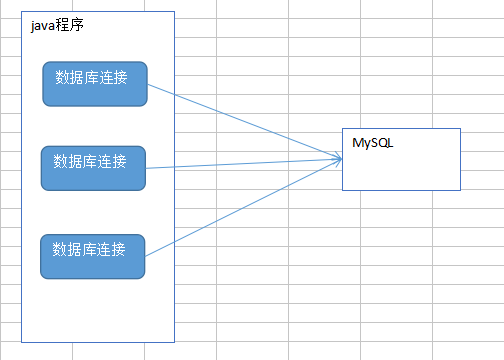


1. Java程序/J2EE程序，当需要操作数据库，通常就是对数据库中某一个，或者某些表，进行增删改查那么就需要通过某些方式连接到数据库，比如MySQL，执行各种SQL语句（insert、select、udpate）
2. JDBC，Java Database Connectivity，Java数据库连接技术。JDBC，其实只是代表了**JDK提供的一套面向数据库的一套开发接口**，注意，这里大部分仅仅是接口而已换句话说，你的Java应用程序，光有JDBC，是操作不了数据库的，更不用谈所谓的CRUD，增删改查。因为JDBC只是一套接口而已，JDBC真正的意义在于**通过接口统一了java程序对各种数据库的访问的规范**。

3.数据库厂商提供的JDBC驱动，JDBC Driver。数据库厂商，比如说，MySQL公司，或者Oracle公司，会针对JDBC的一套接口，提供完整的一套接口的实现类在这套实现类中，不同的数据库厂商就实现了针对自己数据库的一套连接、执行SQL语句等等实际的功能

4.

### 2.1.9 数据库连接池原理

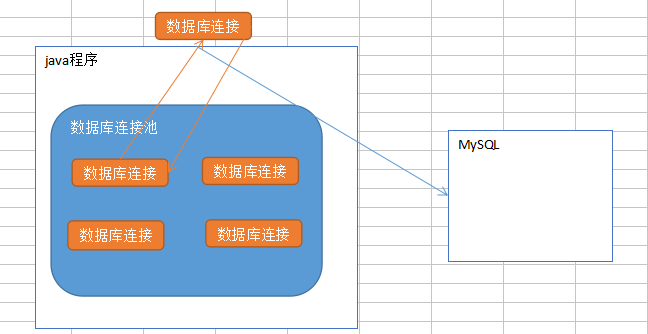


每一次java程序要在MySQL中执行一条SQL语句，那么就必须建立一个Connection对象，代表了与MySQL数据库的连接。然后在通过连接发送了你要执行的SQL语句之后，就会调用Connection.close()来关闭和销毁与数据库的连接。为什么要立即关闭呢？因为数据库的连接是一种很重的资源，代表了网络连接、IO等资源。所以如果不使用的话，就需要尽早关闭，以避免资源浪费。

劣势/不足：如果要频繁地操作MySQL的话，那么就势必会频繁地创建Connection对象，底层建立起与MySQL的占用了网络资源、IO资源的连接。此外呢，每次使用完Connection对象之后，都必须将Connection连接给关闭，又涉及到频繁的网络资源、IO资源的关闭和释放。

如上所述，如果频繁的开关Connection连接，那么会造成大量的对网络、IO资源的申请和释放的无谓的时间的耗费

对于特别频繁的数据库操作，比如100次/s，那么可能会导致性能急剧下降。



数据库连接池的好处：

1、java程序不用自己去管理Connection的创建和销毁，代码上更加方便。

**2、程序中只有固定数量的数据库连接，不会一下子变得很多，而且也不会进行销毁。**那么对于短时间频繁进行数据库操作的业务来说。就有很高的意义和价值。也就是说，如果短时间内，频繁操作10000次，不需要对数据库连接创建和销毁10000次。这样的话，可以大幅度节省我们的数据库连接的创建和销毁的资源开销以及时间开销。

3、最终可以提升整个应用程序的性能。

在我们的spark作业中，通常来说，都需要使用数据库连接池。当然，可以使用开源的数据库连接池，功能非常强大，比如C3P0、DBCP等等。但是呢，对于某些简单的情况下，我们也可以自己编写一个简单的数据库连接池即可。

而且在spark作业中，是非常适合使用数据库连接池的，为什么呢？因此spark计算出来的结果，可能数据量还是会比较大的。比如说10万条。那么如果用普通的数据库操作方式，就必须创建和销毁数据库连接10万次，那么会大大降低整个spark作业的性能。数据库的操作变成整个spark作业的瓶颈。

如果可以善用数据库连接池的话，那么就大大节省数据库连接的创建和销毁的时间和性能开销。大大提升我们的spark作业的整体性能。

在本节课中，会自己手动实现一个非常非常简单的，数据库连接池。

### 2.1.0 单实例模式

/\*\*

\* 单例模式Demo

\*

\* 单例模式是指的什么意思？

\*

\* 我们自己定义的类，其实默认情况下，都是可以让外界的代码随意创建任意多个实例的

\* 但是有些时候，我们不希望外界来随意创建实例，而只是希望一个类，在整个程序运行期间，只有一个实例

\* 任何外界代码，都不能随意创建实例

\*

\* 那么，要实现单例模式，有几个要点：

\* 1、如果不想让外界可以随意创建实例，那么类的构造方法就必须用private修饰，必须是私有的。

\* 2、既然类的构造方法被私有化了，外界代码要想获取类的实例，不能够随意地去创建

\* 那么就只能通过调用类的静态方法，去获取类的实例。

\* 3、所以类必须有一个静态方法，getInstance()，来提供获取唯一实例的功能

\* getInstance()方法，必须保证类的实例创建，且仅创建一次，返回一个唯一的实例

\*

\* 单例模式的应用场景有哪几个呢？

\* 1、配置管理组件，可以在读取大量的配置信息之后，用单例模式的方式，就将配置信息仅仅保存在一个实例的

\* 实例变量中，这样可以避免对于静态不变的配置信息，反复多次的读取

\* 2、JDBC辅助组件，全局就只有一个实例，实例中持有了一个内部的简单数据源

\* 使用了单例模式之后，就保证只有一个实例，那么数据源也只有一个，不会重复创建多次数据源（数据库连接池）

\*

\* **@author** Administrator

\*

\*/

**public** **class** Singleton {

// 首先必须有一个私有的静态变量，来引用自己即将被创建出来的单例

**private** **static** Singleton *instance* = **null**;

/\*\*

\* 其次，必须对自己的构造方法使用private进行私有化

\* 这样，才能保证，外界的代码不能随意的创建类的实例

\*/

**private** Singleton() {

}

/\*\*

\* 最后，需要有一个共有的，静态方法

\* 这个方法，负责创建唯一的实例，并且返回这个唯一的实例

\*

\* 必须考虑到可能会出现的多线程并发访问安全的问题

\* 就是说，可能会有多个线程同时过来获取单例，那么可能会导致创建多次单例

\* 所以，这个方法，通常需要进行多线程并发访问安全的控制

\*

\* 首先，就是，说到多线程并发访问安全的控制，大家觉得最简单的就是在方法上加入synchronized关键词

\* public static synchronized Singleton getInstance()方法

\* 但是这样做有一个很大的问题

\* 在第一次调用的时候，的确是可以做到避免多个线程并发访问创建多个实例的问题

\* 但是在第一次创建完实例以后，就会出现以后的多个线程并发访问这个方法的时候，就会在方法级别进行同步

\* 导致并发性能大幅度降低

\*

\* **@return**

\*/

**public** **static** Singleton getInstance() {

// 两步检查机制

// 首先第一步，多个线程过来的时候，判断instance是否为null

// 如果为null再往下走

**if**(*instance* == **null**) {

// 在这里，进行多个线程的同步

// 同一时间，只能有一个线程获取到Singleton Class对象的锁

// 进入后续的代码

// 其他线程，都是只能够在原地等待，获取锁

**synchronized**(Singleton.**class**) {

// 只有第一个获取到锁的线程，进入到这里，会发现是instance是null

// 然后才会去创建这个单例

// 此后，线程，哪怕是走到了这一步，也会发现instance已经不是null了

// 就不会反复创建一个单例

**if**(*instance* == **null**) {

*instance* = **new** Singleton();

}

}

}

**return** *instance*;

}

}

### 2.1.11 内部类以及匿名内部类

外部类：

最普通的，我们平时见到的那种类，就是在一个后缀为.java的文件中，直接定义的类，比如

public class Student {

private String name;

private int age;

}

内部类：

内部类，顾名思义，就是包含在外部类中的类，就叫做内部类。内部类有两种，一种是**静态内部类，一种是非静态内部类**。

public class School {

private static School instance = null;

static class Teacher {}

}

public class School {

private String name;

class Teacher {}

}

静态内部类和非静态内部类之间的区别主要如下：

1、内部原理的区别：

静态内部类是属于外部类的类成员，是一种静态的成员，是属于类的，就有点类似于private static Singleton instance = null；非静态内部类，是**属于外部类的实例对象的一个实例成员**，也就是说，**每个非静态内部类，不是属于外部类的，是属于外部类的每一个实例的**，创建非静态内部类的实例以后，非静态内部类实例，是必须跟一个外部类的实例进行关联和有寄存关系的。

2、创建方式的区别：

创建静态内部类的实例的时候，只要直接使用“**外部类.内部类()**”的方式，就可以，比如new School.Teacher()；**创建非静态内部类的实例的时候，必须要先创建一个外部类的实例，然后通过外部类的实例，再来创建内部类的实例，new School().Teader()**

通常来说，我们一般都会为了方便，会选择使用静态内部类。

匿名内部类：

public interface ISayHello {

String sayHello(String name);

}

public class SayHelloTest {

public static void main(String[] args) {

ISayHello obj = new ISayHello() {

public String sayHello(String name) { return "hello, " + name }

}

System.out.println(obj.sayHello("leo"));

}

}

匿名内部类的使用场景，通常来说，就是在一个内部类，只要创建一次，使用一次，以后就不再使用的情况下，就可以。那么，此时，通常不会选择在外部创建一个类，而是选择直接创建一个实现了某个接口、或者继承了某个父类的内部类，而且通常是在方法内部，创建一个匿名内部类。

在使用java进行spark编程的时候，如果使用的是java7以及之前的版本，那么通常在对某个RDD执行算子，并传入算子的函数的时候，通常都会传入一个实现了某个Spark Java API中Function接口的匿名内部类。

### 2.1.12 JDBC辅助组件开发

**package** com.ibeifeng.sparkproject.jdbc;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.util.LinkedList;

**import** com.ibeifeng.sparkproject.conf.ConfigurationManager;

**import** com.ibeifeng.sparkproject.constant.Constants;

/\*\*

\* JDBC辅助组件

\*

\* 在正式的项目的代码编写过程中，是完全严格按照大公司的coding标准来的

\* 也就是说，在代码中，是不能出现任何hard code（硬编码）的字符

\* 比如“张三”、“com.mysql.jdbc.Driver”

\* 所有这些东西，都需要通过常量来封装和使用

\*

\* **@author** Administrator

\*

\*/

**public** **class** JDBCHelper {

// 第一步：在静态代码块中，直接加载数据库的驱动

// 加载驱动，不是直接简单的，使用com.mysql.jdbc.Driver就可以了

// 之所以说，不要硬编码，他的原因就在于这里

//

// com.mysql.jdbc.Driver只代表了MySQL数据库的驱动

// 那么，如果有一天，我们的项目底层的数据库要进行迁移，比如迁移到Oracle

// 或者是DB2、SQLServer

// 那么，就必须很费劲的在代码中，找，找到硬编码了com.mysql.jdbc.Driver的地方，然后改成

// 其他数据库的驱动类的类名

// 所以正规项目，是不允许硬编码的，那样维护成本很高

//

// 通常，我们都是用一个常量接口中的某个常量，来代表一个值

// 然后在这个值改变的时候，只要改变常量接口中的常量对应的值就可以了

//

// 项目，要尽量做成可配置的

// 就是说，我们的这个数据库驱动，更进一步，也不只是放在常量接口中就可以了

// 最好的方式，是放在外部的配置文件中，跟代码彻底分离

// 常量接口中，只是包含了这个值对应的key的名字

**static** {

**try** {

String driver = ConfigurationManager.*getProperty*(Constants.***JDBC\_DRIVER***);

Class.*forName*(driver);

} **catch** (Exception e) {

e.printStackTrace();

}

}

// 第二步，实现JDBCHelper的单例化

// 为什么要实现代理化呢？因为它的内部要封装一个简单的内部的数据库连接池

// 为了保证数据库连接池有且仅有一份，所以就通过单例的方式

// 保证JDBCHelper只有一个实例，实例中只有一份数据库连接池

**private** **static** JDBCHelper *instance* = **null**;

/\*\*

\* 获取单例

\* **@return** 单例

\*/

**public** **static** JDBCHelper getInstance() {

**if**(*instance* == **null**) {

**synchronized**(JDBCHelper.**class**) {

**if**(*instance* == **null**) {

*instance* = **new** JDBCHelper();

}

}

}

**return** *instance*;

}

// 数据库连接池

**private** LinkedList<Connection> datasource = **new** LinkedList<Connection>();

/\*\*

\*

\* 第三步：实现单例的过程中，创建唯一的数据库连接池

\*

\* 私有化构造方法

\*

\* JDBCHelper在整个程序运行声明周期中，只会创建一次实例

\* 在这一次创建实例的过程中，就会调用JDBCHelper()构造方法

\* 此时，就可以在构造方法中，去创建自己唯一的一个数据库连接池

\*

\*/

**private** JDBCHelper() {

// 首先第一步，获取数据库连接池的大小，就是说，数据库连接池中要放多少个数据库连接

// 这个，可以通过在配置文件中配置的方式，来灵活的设定

**int** datasourceSize = ConfigurationManager.*getInteger*(

Constants.***JDBC\_DATASOURCE\_SIZE***);

// 然后创建指定数量的数据库连接，并放入数据库连接池中

**for**(**int** i = 0; i < datasourceSize; i++) {

String url = ConfigurationManager.*getProperty*(Constants.***JDBC\_URL***);

String user = ConfigurationManager.*getProperty*(Constants.***JDBC\_USER***);

String password = ConfigurationManager.*getProperty*(Constants.***JDBC\_PASSWORD***);

**try** {

Connection conn = DriverManager.*getConnection*(url, user, password);

datasource.push(conn);

} **catch** (Exception e) {

e.printStackTrace();

}

}

}

/\*\*

\* 第四步，提供获取数据库连接的方法

\* 有可能，你去获取的时候，这个时候，连接都被用光了，你暂时获取不到数据库连接

\* 所以我们要自己编码实现一个简单的等待机制，去等待获取到数据库连接

\*

\*/

**public** **synchronized** Connection getConnection() {

**while**(datasource.size() == 0) {

**try** {

Thread.*sleep*(10);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

**return** datasource.poll();

}

}

### 2.1.14 javaBean概念讲解

JavaBean：类

JavaBean，虽然就是一个类，但是是有特殊条件的一个类，不是所有的类都可以叫做JavaBean的

首先，它需要有一些field，这些field，都必须用private来修饰，表示所有的field，都是私有化的，不能随意的获取和设置

其次，需要给所有的field，都提供对应的setter和getter方法，什么叫setter和getter？setter，就是说setX()方法，用于给某个field设置值；getter，就是说getX()方法，用于对某个field获取值

public class Student {

private String name;

private int age;

public void setName(String name) {

this.name = name;

}

public String getName() {

return name;

}

public void setAge(int age) {

this.age = age;

}

public int getAge() {

return age;

}

}

JavaBean通常怎么用？通常来说，会将一个JavaBean，与数据库中的某个表一一对应起来。比如说，有一个student表，create table student(name varchar(30), age integer)，那么这个表，如果要操作的话，通常来说，会在程序中，建立一个对应的JavaBean，这个JavaBean中，所有的field，都是和表中的字段一一对应起来的。

然后呢，在执行增删改查操作的时候，其实都是面向JavaBean来操作的，比如insertStudent()方法，就应该接收一个参数，Student对象；findAllStudent()方法，就应该将返回类型设置为List<Student>列表

domain的概念：在系统中，通常会分很多层，比如经典的三层架构，**控制层、业务层、数据访问层（DAO层）**

此外，还有一个层，就是domain层。domain层，通常就是用于放置这个系统中，与数据库中的表，一一对应起来的JavaBean的

**三层架构+domain层+model层（J2EE web系统）**

**浏览器->后台->控制层->业务层->数据访问层->数据库**

domain->domain->domain->SQL

domain/model<-

domain和model可能都是JavaBean；之间的区别，只是用途不太一样，domain通常就代表了与数据库表一一对应的JavaBean；model通常代表了不与数据库一一对应的JavaBean，但是封装的数据，是前端的JS脚本，需要使用的一些数据。

### 2.1.15 DAO模式讲解

DAO模式

Data Access Object：数据访问对象

首先，你的应用程序，肯定会有业务逻辑的代码。在三层架构的web系统中，业务逻辑的代码，就是在你的Service组件里面；在我们的spark作业中，业务逻辑代码就是在我们的spark作业里面。

如果说，你不用DAO模式的话，那么所有的数据库访问的代码和逻辑会全部耦合在业务逻辑代码里面。比如，你的业务逻辑代码中，可能会充斥着JDBCHelper，定义SQL语句，处理查询返回结果等等代码。会**导致业务逻辑和数据访问严重耦合**。导致以后如果你只是想修改数据访问的代码，那么还得在一大堆业务逻辑的代码中去找，找到这段代码，然后去做对应的修改。

因为我们要将数据库访问的代码，封装在DAO中，然后呢，业务逻辑的代码，就可以直接去调用DAO组件，实现数据库操作的逻辑。如果这样做了以后，那么在业务逻辑的代码中，就不可能出现SQL语句、查询结果处理等等这些东西。当后面对系统进行维护的时候，如果你只是要优化一条SQL语句，那么你肯定不用跑到业务逻辑的代码里面去找到这条SQL语句，然后去修改它。。。

只需要到业务逻辑代码调用的DAO组件里面去，找到对应的SQL语句，修改，即可。

引入了DAO模式以后，就大大降低了业务逻辑层和数据访问层的耦合，大大提升了后期的系统维护的效率，并降低了时间成本。

我们自己在实现DAO模式的时候，通常来说，会将其分为两部分，一个是DAO接口；一个是DAO实现类。我们的业务的代码，通常就是面向接口进行编程；那么当接口的实现需要改变的时候，直接定义一个新的实现即可。但是对于我们的业务代码来说，只要面向接口开发就可以了。DAO的改动对业务代码应该没有任何的影响。

**import** com.ibeifeng.sparkproject.domain.Task;

/\*\*

\* 任务管理DAO接口

\* **@author** Administrator

\*

\*/

**public** **interface** ITaskDAO {

/\*\*

\* 根据主键查询任务

\* **@param** taskid 主键

\* **@return** 任务

\*/

Task findById(**long** taskid);

}

import java.sql.ResultSet;

import com.ibeifeng.sparkproject.dao.ITaskDAO;

import com.ibeifeng.sparkproject.domain.Task;

import com.ibeifeng.sparkproject.jdbc.JDBCHelper;

/\*\*

\* 任务管理DAO实现类

\* @author Administrator

\*

\*/

public class TaskDAOImpl implements ITaskDAO {

/\*\*

\* 根据主键查询任务

\* @param taskid 主键

\* @return 任务

\*/

public Task findById(long taskid) {

final Task task = new Task();

String sql = "select \* from task where task\_id=?";

Object[] params = new Object[]{taskid};

JDBCHelper jdbcHelper = JDBCHelper.getInstance();

jdbcHelper.executeQuery(sql, params, new JDBCHelper.QueryCallback() {

@Override

public void process(ResultSet rs) throws Exception {

if(rs.next()) {

long taskid = rs.getLong(1);

String taskName = rs.getString(2);

String createTime = rs.getString(3);

String startTime = rs.getString(4);

String finishTime = rs.getString(5);

String taskType = rs.getString(6);

String taskStatus = rs.getString(7);

String taskParam = rs.getString(8);

task.setTaskid(taskid);

task.setTaskName(taskName);

task.setCreateTime(createTime);

task.setStartTime(startTime);

task.setFinishTime(finishTime);

task.setTaskType(taskType);

task.setTaskStatus(taskStatus);

task.setTaskParam(taskParam);

}

}

});

/\*\*

\* 说在后面的话：

\*

\* 大家看到这个代码，包括后面的其他的DAO，就会发现，用JDBC进行数据库操作，最大的问题就是麻烦

\* 你为了查询某些数据，需要自己编写大量的Domain对象的封装，数据的获取，数据的设置

\* 造成大量很冗余的代码

\*

\* 所以说，之前就是说，不建议用Scala来开发大型复杂的Spark的工程项目

\* 因为大型复杂的工程项目，必定是要涉及很多第三方的东西的，MySQL只是最基础的，要进行数据库操作

\* 可能还会有其他的redis、zookeeper等等

\*

\* 如果你就用Scala，那么势必会造成与调用第三方组件的代码用java，那么就会变成scala+java混编

\* 大大降低我们的开发和维护的效率

\*

\* 此外，即使，你是用了scala+java混编

\* 但是，真正最方便的，还是使用一些j2ee的开源框架，来进行第三方

\* 技术的整合和操作，比如MySQL，那么可以用MyBatis/Hibernate，大大减少我们的冗余的代码

\* 大大提升我们的开发速度和效率

\*

\* 但是如果用了scala，那么用j2ee开源框架，进来，造成scala+java+j2ee开源框架混编

\* 简直会造成你的spark工程的代码上的极度混乱和惨不忍睹

\* 后期非常难以维护和交接

\*

\*/

return task;

}

}

### 2.1.16 工厂模式讲解以及DAOFactory开发

如果没有工厂模式，可能会出现的问题：

ITaskDAO接口和TaskDAOImpl实现类；实现类是可能会更换的；那么，如果你就使用普通的方式来创建DAO，比如ITaskDAO taskDAO = new TaskDAOImpl()，那么后续，如果你的TaskDAO的实现类变更了，那么你就必须在你的程序中，所有出现过TaskDAOImpl的地方，去更换掉这个实现类。这是非常非常麻烦的。

如果说，你的TaskDAOImpl这个类，在你的程序中出现了100次，那么你就需要修改100个地方。这对程序的维护是一场灾难。

工厂设计模式

对于一些种类的对象，使用一个工厂，来提供这些对象创建的方式，外界要使用某个类型的对象时，就直接通过工厂来获取即可。不用自己手动一个一个地方的去创建对应的对象。

那么，假使我们有100个地方用到了TaskDAOImpl。不需要去在100个地方都创建TaskDAOImpl()对象，只要在100个地方，都使用TaskFactory.getTaskDAO()方法，获取出来ITaskDAO接口类型的对象即可。

如果后面，比如说MySQL迁移到Oracle，我们重新开发了一套TaskDAOImpl实现类，那么就直接在工厂方法中，更换掉这个类即可。不需要再所有使用到的地方都去修改。

package com.ibeifeng.sparkproject.dao.impl;

import com.ibeifeng.sparkproject.dao.ITaskDAO;

/\*\*

\* DAO工厂类

\* @author Administrator

\*

\*/

public class DAOFactory {

/\*\*

\* 获取任务管理DAO

\* @return DAO

\*/

public static ITaskDAO getTaskDAO() {

return new TaskDAOImpl();

}

}

package com.ibeifeng.sparkproject.test;

import com.ibeifeng.sparkproject.dao.ITaskDAO;

import com.ibeifeng.sparkproject.dao.impl.DAOFactory;

import com.ibeifeng.sparkproject.domain.Task;

/\*\*

\* 任务管理DAO测试类

\* @author Administrator

\*

\*/

public class TaskDAOTest {

public static void main(String[] args) {

ITaskDAO taskDAO = DAOFactory.getTaskDAO();

Task task = taskDAO.findById(2);

System.out.println(task.getTaskName());

}

}

### 2.1.17 JSON数据格式讲解以及fastjson介绍

什么是JSON？

就是一种数据格式；比如说，我们现在规定，有一个txt文本文件，用来存放一个班级的成绩；然后呢，我们规定，这个文本文件里的学生成绩的格式，是第一行，就是一行列头（姓名 班级 年级 科目 成绩），接下来，每一行就是一个学生的成绩。那么，这个文本文件内的这种信息存放的格式，其实就是一种数据格式。

学生 班级 年级 科目 成绩

张三 一班 大一 高数 90

李四 二班 大一 高数 80

ok，对应到JSON，它其实也是代表了一种数据格式，所谓数据格式，就是数据组织的形式。比如说，刚才所说的学生成绩，用JSON格式来表示的话，如下：

[{"学生":"张三", "班级":"一班", "年级":"大一", "科目":"高数", "成绩":90}, {"学生":"李四", "班级":"二班", "年级":"大一", "科目":"高数", "成绩":80}]

其实，JSON，很简单，一点都不复杂，就是对同样一批数据的，不同的一种数据表示的形式。

JSON的数据语法，其实很简单：如果是包含多个数据实体的话，比如说多个学生成绩，那么需要使用数组的表现形式，就是[]。对于单个数据实体，比如一个学生的成绩，那么使用一个{}来封装数据，对于数据实体中的每个字段以及对应的值，使用key:value的方式来表示，多个key-value对之间用逗号分隔；多个{}代表的数据实体之间，用逗号分隔。

扩展一下：

JSON在企业级项目开发过程中，扮演的角色是无比重要的。最常用的地方，莫过于基于Ajax的前端和后端程序之间的通信。比如说，在前端页面中，可以不刷新页面，直接发送一个Ajax异步请求到后端，后端返回一个JSON格式的数据，然后前端使用JSON格式的数据，渲染页面中的对应地方的信息。

在我们的项目中，JSON是起到了什么作用呢？我们在task表中的task\_param字段，会存放不同类型的任务对应的参数。比如说，用户访问session分析模块与页面单跳转化率统计模块的任务参数是不同的，但是，使用同一张task表来存储所有类型的任务。那么，你怎么来存储不同类型的任务的不同的参数呢？你的表的字段是事先要定好的呀。

所以，我们采取了，用一个task\_param字段，来存储不同类型的任务的参数的方式。task\_param字段中，实际上会存储一个任务所有的字段，使用JSON的格式封装所有任务参数，并存储在task\_param字段中。就实现了非常灵活的方式。

如何来操作JSON格式的数据？

比如说，要获取JSON中某个字段的值。我们这里使用的是阿里的fastjson工具包。使用这个工具包，可以方便的将字符串类型的JSON数据，转换为一个JSONObject对象，然后通过其中的getX()方法，获取指定的字段的值。

package com.ibeifeng.sparkproject.test;

import com.alibaba.fastjson.JSONArray;

import com.alibaba.fastjson.JSONObject;

/\*\*

\* fastjson测试类

\* @author Administrator

\*

\*/

public class FastjsonTest {

public static void main(String[] args) {

String json = "[{'学生':'张三', '班级':'一班', '年级':'大一', '科目':'高数', '成绩':90}, {'学生':'李四', '班级':'二班', '年级':'大一', '科目':'高数', '成绩':80}]";

JSONArray jsonArray = JSONArray.parseArray(json);

JSONObject jsonObject = jsonArray.getJSONObject(1);

System.out.println(jsonObject.getString("学生"));

}

}

### 2.1.18 Spark上下文构建

目的：用户访问session分析Spark作业

step1: \* 构建Spark上下文

\* 生成模拟测试数据

\* 关闭Spark上下文

step2:获取SQLContext

\* 如果是在本地测试环境的话，那么就生成SQLContext对象

\* 如果是在生产环境运行的话，那么就生成HiveContext对象

step3:生成模拟数据（只有本地模式，才会去生成模拟数据）

### 2.1.19 对行为数据按session粒度进行聚合

目的：**对行为数据按session粒度进行聚合**

现在actionRDD中的元素是Row，一个Row就是一行用户访问行为记录，比如一次点击或者搜索。我们现在需要将这个Row映射成<sessionid,Row>的格式

对行为数据按session粒度进行分组，对每一个session分组进行聚合，将session中所有的搜索词和点击品类都聚合起来，到此为止，获取的数据格式，如下：<userid,partAggrInfo(sessionid,searchKeywords,clickCategoryIds)>

**遍历session所有的访问行为，提取每个访问行为的搜索词字段和点击品类字段**，实际上这里要对数据说明一下：并不是每一行访问行为都有searchKeyword何clickCategoryId两个字段的，其实，只有搜索行为，是有searchKeyword字段的，只有点击品类的行为，是有clickCategoryId字段的，所以，任何一行行为数据，都不可能两个字段都有，所以数据是可能出现null值的，我们决定是否将搜索词或点击品类id拼接到字符串中去，首先要满足：不能是null值，其次，之前的字符串中还没有搜索词或者点击品类id。

<userid,partAggrInfo>然后跟用户信息join的时候，将partAggrInfo关联上userInfo然后再直接将返回的Tuple的key设置成sessionid最后的数据格式，还是<sessionid,fullAggrInfo>

聚合数据，我们这里统一定义，使用key=value|key=value。

### 2.1.20 按筛选参数对session粒度聚合数据进行过滤

目标：过滤session数据

首先将所有的筛选参数拼接成一个连接串，其实我们是给后面的性能优化埋下了一个伏笔。

startAge

endAge

professionals

cities

sex

keywords

categoryIds

根据筛选参数进行过滤：首先，从tuple中，获取聚合数据， 接着，依次按照筛选条件进行过滤：

\* 按照年龄范围进行过滤（startAge、endAge）

\* 按照职业范围进行过滤（professionals）[互联网,IT,软件]

\* 按照城市范围进行过滤（cities）[北京,上海,广州,深圳,成都]

\* 按照性别进行过滤 [男，女]

\* 按照搜索词进行过滤[session可能搜索了 火锅,蛋糕,烧烤]

( 那么，in这个校验方法，主要判定session搜索的词中，有任何一个，与筛选条件中任何一个搜索词相当，即通过)

\* 按照点击品类id进行过滤

### 2.1.21 session聚合统计之自定义Accumulator

#### 理论解释

session聚合统计：

统计出来之前通过条件过滤的session，访问时长在0s~3s的session的数量，占总session数量的比例；4s~6s。。。。；

访问步长在1~3的session的数量，占总session数量的比例；4~6。。。；

Accumulator 1s\_3s = sc.accumulator(0L);

十几个Accumulator

可以对过滤以后的session，调用foreach也可以，遍历所有session；计算每个session的访问时长和访问步长；

访问时长：把session的最后一个action的时间，减去第一个action的时间

访问步长：session的action数量

计算出访问时长和访问步长以后，根据对应的区间，找到对应的Accumulator， 1s\_3s.add(1L)

同时每遍历一个session，就可以给总session数量对应的Accumulator，加1,最后用各个区间的session数量，除以总session数量，就可以计算出各个区间的占比了

这种传统的实现方式，有什么不好？？？

最大的不好，就是Accumulator太多了，不便于维护

首先第一，很有可能，在写后面的累加代码的时候，比如找到了一个4s~6s的区间的session，但是却代码里面不小心，累加到7s~9s里面去了；

第二，当后期，项目如果要出现一些逻辑上的变更，比如说，session数量的计算逻辑，要改变，就得更改所有Accumulator对应的代码；或者说，又要增加几个范围，那么又要增加多个Accumulator，并且修改对应的累加代码；维护成本，相当之高（甚至可能，修改一个小功能，或者增加一个小功能，耗费的时间，比做一个新项目还要多；甚至于，还修改出了bug，那就耗费更多的时间）

所以，我们这里的设计，不打算采用传统的方式，用十几个，甚至二十个Accumulator,因为维护成本太高.

这里的实现思路是，我们自己自定义一个Accumulator，实现较为复杂的计算逻辑，一个Accumulator维护了所有范围区间的数量的统计逻辑

**低耦合**，如果说，session数量计算逻辑要改变，那么不用变更session遍历的相关的代码；只要维护一个Accumulator里面的代码即可；如果计算逻辑后期变更，或者加了几个范围，那么也很方便，不用多加好几个Accumulator，去修改大量的代码；只要维护一个Accumulator里面的代码即可；

维护成本，大大降低

自定义Accumulator，也是Spark Core中，属于比较高端的一个技术,使用自定义Accumulator，大家就可以任意的实现自己的复杂分布式计算的逻辑

如果说，你的task，分布式，进行复杂计算逻辑，那么是很难实现的（借助于redis，维护中间状态，借助于zookeeper去实现分布式锁）但是，使用自定义Accumulator，可以更方便进行中间状态的维护，而且不用担心并发和锁的问题。

#### 代码逻辑

目的：session聚合统计Accumulator -- 自定义计算器

其实使用自己定义的一些数据格式，比如String，甚至说，我们可以自己定义model，自己定义的类（必须可序列化），然后呢，可以基于这种特殊的数据格式，可以实现自己复杂的分布式的计算逻辑，各个task，分布式在运行，可以根据你的需求，task给Accumulator传入不同的值，根据不同的值，去做复杂的逻辑，Spark Core里面很实用的高端技术。

### 2.1.22 session聚合统计之重构实现思路与重构session聚合

#### 理论解释

目的：

session聚合统计（统计出访问时长和访问步长，各个区间的session数量占总session数量的比例）

如果不进行重构，直接来实现，思路：

\* 1、actionRDD，映射成<sessionid,Row>的格式

\* 2、按sessionid聚合，计算出每个session的访问时长和访问步长，生成一个新的RDD

\* 3、遍历新生成的RDD，将每个session的访问时长和访问步长，去更新自定义Accumulator中的对应的值

\* 4、使用自定义Accumulator中的统计值，去计算各个区间的比例

\* 5、将最后计算出来的结果，写入MySQL对应的表中

\* 普通实现思路的问题：

\* 1、为什么还要用actionRDD，去映射？其实我们之前在session聚合的时候，映射已经做过了。多此一举

\* 2、是不是一定要，为了session的聚合这个功能，单独去遍历一遍session？其实没有必要，已经有session数据，之前过滤session的时候，其实，就相当于，是在遍历session，那么这里就没有必要再过滤一遍了

\* 重构实现思路：

\* 1、不要去生成任何新的RDD（处理上亿的数据）

\* 2、不要去单独遍历一遍session的数据（处理上千万的数据）

\* 3、可以在进行session聚合的时候，就直接计算出来每个session的访问时长和访问步长

\* 4、在进行过滤的时候，本来就要遍历所有的聚合session信息，此时，就可以在某个session通过筛选条件后，将其访问时长和访问步长，累加到自定义的Accumulator上面去

\* 5、就是两种截然不同的思考方式，和实现方式，在面对上亿，上千万数据的时候，甚至可以节省时间长达，半个小时，或者数个小时

#### 经验准则

开发Spark大型复杂项目的一些经验准则：

\* 1、尽量少生成RDD

\* 2、尽量少对RDD进行算子操作，如果有可能，尽量在一个算子里面，实现多个需要做的功能

\* 3、尽量少对RDD进行shuffle算子操作，比如groupByKey、reduceByKey、sortByKey（map、mapToPair）；shuffle操作，会导致大量的磁盘读写，严重降低性能；有shuffle的算子，和没有shuffle的算子，甚至性能，会达到几十分钟，甚至数个小时的差别；有shfufle的算子，很容易导致数据倾斜，一旦数据倾斜，简直就是性能杀手（完整的解决方案）

\* 4、无论做什么功能，性能第一在传统的J2EE或者.NET后者PHP，软件/系统/网站开发中，我认为是架构和可维护性，可扩展性的重要程度，远远高于了性能，大量的分布式的架构，设计模式，代码的划分，类的划分（高并发网站除外）

在大数据项目中，比如MapReduce、Hive、Spark、Storm，我认为性能的重要程度，远远大于一些代码的规范，和设计模式，代码的划分，类的划分；大数据，大数据，最重要的，就是性能主要就是因为大数据以及大数据项目的特点，决定了，大数据的程序和项目的速度，都比较慢如果不优先考虑性能的话，会导致一个大数据处理程序运行时间长度数个小时，甚至数十个小时此时，对于用户体验，简直就是一场灾难所以，推荐大数据项目，在开发和代码的架构中，**优先考虑性能**；其次考虑功能代码的划分、解耦合我们如果采用第一种实现方案，那么其实就是代码划分（解耦合、可维护）优先，设计优先如果采用第二种方案，那么其实就是性能优先讲了这么多，其实大家不要以为我是在岔开话题，大家不要觉得项目的课程，就是单纯的项目本身以及代码coding最重要，其实项目，我觉得，最重要的，除了技术本身和项目经验以外；非常重要的一点，就是积累了，处理各种问题的经验。