

# 《专业方向课程设计》大作业

---

题目：大规模用户行为分析系统

姓名	学号	班级	成绩

大连理工大学软件学院统

2019年7月

# 项目说明

神策题目：“神策分析”是一个数据分析产品，包含一个完整的数据仓库。数据仓库的建设是数据进一步应用的基础。而一个完整的数据仓库通常有如下模块：数据采集（SDK、导入工具、LogAgent 等）、数据导入（清洗、入库）、存储、查询引擎、分析模型抽象层、接口层、UI 交互层。其中从“数据采集”到“导入存储”，涉及的技术细节非常繁杂，却又不能直观地回答初学者的疑问：“具体业务场景下，数据分析到底是如何应用的”这个问题。但查询层（包括模型抽象、查询引擎）则不同，查询一侧是贴近业务应用的一侧，相对而言会更加直观。所以本次的项目我们就以查询层为切入点，去尝试搭建一个大规模用户行为分析系统。

# 成员分工

成员	学号	分工
徐诗瑶	201692126	业务逻辑设计+Impala实现+模拟数据生成+总体设计+功能实现+优化方法实现
王贝	201672048	业务逻辑设计+界面设计+界面实现+环境搭建(Hadoop+Hive+Impala)+数据导入+优化方法
吴任高	201671905	模拟数据生成+性能测试+正确性测试+编写用户使用手册

# 环境安装

## 1.Hadoop

### 环境准备

### jdk下载

到官网下载了jdk8 jdk-8u191-macosx-x64.dmg安装jdk 之后配置环境变量如下：

```
JAVA_HOME="Library/Java/JavaVirtualMachines/jdk1.8.0_191.jdk/Contents/Home"
export JAVA_HOME
CLASS_PATH="$JAVA_HOME/lib"
PATH=".$PATH:$JAVA_HOME/bin"
export PATH="$HOME/.yarn/bin:$PATH"
```

根据[这个教程](#)装好了java

### ssh配置

先把系统偏好设置-共享-远程登录打开

```
ssh localhost
```

显示需要密码，实际上就是本机密码，这样不是很ok（具体到底哪里不ok我也不是很清楚

terminal中修改ssh设置

```
ssh-keygen -t rsa  
[ 这里有啥输入的东西反正我们就按回车就完事 ]  
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys  
chmod og-wx ~/.ssh.authorized_keys
```

这时候我们再执行

```
ssh localhost
```

就会发现不需要密码ssh登陆了～就可以下载Hadoop了呢！

## Hadoop下载安装

官网下载

[官网提供的下载地址](#)我下载了2.8.5

下载完之后我把这个tar.gz放到了/Documents/Hadoop 文件夹里

```
cd Hadoop  
tar -zxvf hadoop-2.8.5.tar.gz
```

## 添加Hadoop环境变量

在~/.bash\_profile中添加

```
# Setting path for Hadoop  
HADOOP_HOME="/Users/xusy/Documents/Hadoop/hadoop-2.8.5"  
export HADOOP_HOME  
export PATH=$PATH:HADOOP_HOME/sbin:$HADOOP_HOME/bin  
  
export LD_LIBRARY_PATH=$HADOOP_HOME/lib/native/  
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native  
export HADOOP_OPTS="-  
Djava.library.path=$HADOOP_HOME/lib/native:$HADOOP_COMMON_LIB_NATIVE_DIR"
```

具体路径根据hadoop的安装目录决定

下半部分的配置可以在上面提到的一些

接下来可以进入到我们的Hadoop目录里：

/hadoop-2.8.5/etc/hadoop/

然后修改core-site.xml, mapred-site.xml(这里是mapred-site.xml.template修改成.xml)

**hadoop-env.sh**

这个配置文件网上找到的大部分教程都要修改..但是..我看完我下载完之后打开的默认配置感觉不用改..于是没改..

---更新---

在这个配置文件中删掉了一些export前的注释, 关于JAVA\_HOME, JSVC\_HOME, HADOOP\_HOME, HADOOP\_HEAPSIZE=1000(或者2000), HADOOP\_OPTS一些的注释都被去掉了, 无需添加啥别的东西

---再来更新---

在又又又又启动的时候发现跑代码的时候会有些问题..报错信息显示的是Javahome的问题..以及Hadoophome的问题..因此还是对hadoop-env.sh文件作了修改, 具体添加了javahome:

```
export
JAVA_HOME="/Library/Java/JavaVirtualMachines/jdk1.8.0_191.jdk/Contents/Home"
export HADOOP_NAMENODE_OPTS="-
Dhadoop.security.logger=${HADOOP_SECURITY_LOGGER:-INFO,RFAS} -
Dhdfs.audit.logger=${HDFS_AUDIT_LOGGER:-INFO,NullAppender}
$HADOOP_NAMENODE_OPTS"
export HADOOP_DATANODE_OPTS="-Dhadoop.security.logger=ERROR,RFAS
$HADOOP_DATANODE_OPTS"

export HADOOP_SECONDARYNAMENODE_OPTS="-
Dhadoop.security.logger=${HADOOP_SECURITY_LOGGER:-INFO,RFAS} -
Dhdfs.audit.logger=${HDFS_AUDIT_LOGGER:-INFO,NullAppender}
$HADOOP_SECONDARYNAMENODE_OPTS"

export HADOOP_NFS3_OPTS="$HADOOP_NFS3_OPTS"
export HADOOP_PORTMAP_OPTS="-Xmx512m $HADOOP_PORTMAP_OPTS"
```

具体的配置文件放到了我的 GitHub -> HadoopClassNote里~

同样的: 在hadoop-env.sh, mapred-env.sh, yarn-env.sh这三个文件里都要对JAVA\_HOME进行添加修改

## core-site.xml

```
<configuration>
  <property>
    <name>fs.default.name</name>
    <value>hdfs://localhost:9000</value>
  </property>
  <property>
    <name>hadoop.tmp.dir</name>
    <value>/Users/xusy/Documents/Hadoop</value>
  </property>
</configuration>
```

👉是自定义的放hdfs文件的目录这里我就直接放在了我的Hadoop目录里

(后来由于namenode的相关信息存在了系统的tmp文件夹里，导致每次系统重启的时候都会出现配置不能成功启动，我们每次都要格式化namenode，这样就非常不ok，所以我们对这个文件稍微修改了一下)

```
<property>
  <name>hadoop.tmp.dir</name>
  <value>/Users/xusy/hadoop_tmp</value>
</property>
```

## mapred-site.xml

这个文件实际上我下载完的后缀是.xml.template(还是啥玩意反正是后面有个后缀，被我直接修改成了.xml)

```
<configuration>
  <property>
    <name>mapred.job.tracker</name>
    <value>localhost:9010</value>
  </property>
  <property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
  </property>
</configuration>
```

## hdfs-site.xml

```
<configuration>
  <!--伪分布式-->
  <property>
    <name>dfs.replication</name>
    <value>1</value>
  </property>
</configuration>
```

这里的变量dfs.replication指定了每个HDFS数据库的复制次数，通常为3，而我们要在本机建立一个伪分布式的DataNode所以这个值改成了1

为了保存hdfs的元数据和data相关文件，这里后来添加了property：

```
<configuration>
  <!--伪分布式-->
  <property>
    <name>dfs.namenode.name.dir</name>
    <value>/Users/xusy/Documents/Hadoop/dfs/name</value>
  </property>
  <property>
    <name>dfs.datanode.data.dir</name>
```

```

    <value>/Users/xusy/Documents/Hadoop/dfs/data</value>
  </property>
</property>
  <name>dfs.replication</name>
  <value>1</value>
</property>
</property>
  <name>dfs.permissions</name>
  <value>>false</value>
</property>
</configuration>

```

## yarn-site.xml

```

<configuration>
  <property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
  </property>

  <!-- Site specific YARN configuration properties -->

  <!-- 集群配置-->
  <!--      <property>
        <name>yarn.resourcemanager.hostname</name>
        <value>master</value>
      </property> -->

</configuration>

```

同样的稍微做了修改

```

<configuration>
  <property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
  </property>
  <property>
    <name>yarn.resourcemanager.resource-tracker.address</name>
    <value>localhost:8031</value>
  </property>
  <property>
    <name>yarn.resourcemanager.address</name>
    <value>localhost:8032</value>
  </property>
  <property>
    <name>yarn.resourcemanager.admin.address</name>

```

```

    <value>localhost:8033</value>
  </property>
  <property>
    <name>yarn.resourcemanager.scheduler.address</name>
    <value>localhost:8034</value>
  </property>
  <property>
    <name>yarn.resourcemanager.webapp.address</name>
    <value>localhost:8088</value>
  </property>
  <property>
    <name>yarn.log-aggregation-enable</name>
    <value>true</value>
  </property>
  <property>
    <name>yarn.log.server.url</name>
    <value>http://localhost:19888/jobhistory/logs</value>
  </property>
<!-- Site specific YARN configuration properties -->

<!-- 集群配置-->
<!--      <property>
        <name>yarn.resourcemanager.hostname</name>
        <value>master</value>
      </property> -->
</configuration>

```

## log4j.properties

在具体跑代码的时候会有些WARNING(但实际上你的代码并没有什么问题..)因此我们要在log4j.properties文件后追加一行内容:

```
log4j.logger.org.apache.hadoop.util.NativeCodeLoader=ERROR
```

## 启动Hadoop

每次操作的时候都要进入这个Hadoop文件夹哦(当然我觉得如果把这个添加到环境变量里会不会好点..我也不知道我瞎说的)

终端进入到Hadoop的文件夹下 我这里的文件夹就是

```
/Users/xusy/Documents/Hadoop/hadoop-2.8.5
```

执行

```
./bin/hdfs namenode -format
```

格式化文件系统(对namenode进行初始化)(好像是只要初始化一次就好了就是最开始建系统的时候..之后如果每次启动你都初始化..那么是会有问题的!)

---

更新

---

在启动Hadoop, jps之后可能会出现你的namenode没起来的这个问题, 这个时候就得格式化一下namenode, 具体的话👉

这里的namenode format的问题: 由于namenode的信息是存在了系统的tmp文件夹下的, 如果你到这里看的话是能看见这些的:

每次启动的话tmp是会清空的, 我也不知道咋回事反正, 虽然我在core-site.xml文件里明明定义的是tmp存在了Hadoop文件夹下...但还是有这个问题..所以就重新在我的xusy用户下面新建了一个hadoop\_tmp文件夹, 把上面core-site.xml里存temp的那个文件夹路径改成了

```
<property>
  <name>hadoop.tmp.dir</name>
  <value>/Users/xusy/hadoop_tmp</value>
```

然后重新format就可以了..不知道再重新启动我的电脑的时候还会有这个问题..如果有那就再更新一下..

接下来启动namenode & datanode (感觉就是启动dfs文件系统)

```
./sbin/start-dfs.sh
```

中间会有一个询问yes/no的我们输入yes就好了.. 启动yarn

```
./sbin/start-yarn.sh
```

启动日志管理log的historyserver

```
./mr-jobhistory-daemon.sh start historyserver
```

👉 输入了这个命令就可以在jps里看见JobHistoryServer了

当然以上的命令都是在hadoop-2.8.5下面运行的

想要关闭的话..

```
./sbin/stop-all.sh
# stop-dfs.sh stop-yarn.sh
```

查看当前的hadoop运行情况:



```
xushiyaodeMacBook-Pro:sbin xusy$ jps
39696 SecondaryNameNode
39809 ResourceManager
49810 JobHistoryServer
39891 NodeManager
39507 NameNode
69306
39595 DataNode
73471 Jps
```

测试一下我们能不能进入到overview界面呢！

NameNode - <http://localhost:50070>

ps:这里有一个Hadoop2和Hadoop3对应端口修改的表在下面：

NameNode端口

Hadoop2	Hadoop3
50470	9871
50070	9870
8020	9820

Secondary NN端口

Hadoop2	Hadoop3
50091	9869
50090	9868

DataNode端口

Hadoop2	Hadoop3
50020	9867
50010	9866
50475	9865
50075	9864

继续启动！！

由于我们刚刚到配置..这里的namenode1对应的就是我们本机localhost啦~(所以下面的web查看正常输入的URL应该是namenode1+端口的)

overview查看!

查看HDFS:

<http://localhost:50070>

查看YARN:

<http://localhost:8088>

查看MR启动JobHistory Server(这里暂时出了问题..让我研究一下..)

<http://localhost:19888>

## 2. Hive

### 一、安装 MySQL

1. 上传MySQL在线安装源的配置文件

用WinSCP (root账号连接) CentOS服务器

将mysql-community.repo 文件上传到 /etc/yum.repos.d/ 目录

将RPM-GPG-KEY-mysql 文件上传到 /etc/pki/rpm-gpg/ 目录

2. 更新yum源并安装mysql server (默认同时会安装mysql client)

```
yum repolist
```

```
yum install mysql-server
```

3. 查看MySQL各组件是否成功安装

```
rpm -qa | grep mysql
```

```
[hadoop@namenode1 ~]$ rpm -qa|grep mysql
mysql-community-libs-5.6.40-2.el7.x86_64
mysql-community-server-5.6.40-2.el7.x86_64
mysql-community-client-5.6.40-2.el7.x86_64
mysql-community-common-5.6.40-2.el7.x86_64
```

### 二、配置MySQL

1. 启动MySQL Server并查看其状态

```
systemctl start mysqld
```

```
systemctl status mysqld
```

```
[hadoop@namenode1 ~]$ systemctl status mysqld
* mysqld.service - MySQL Community Server
   Loaded: loaded (/usr/lib/systemd/system/mysqld.service; enabled; vendor preset: disabled)
   Active: active (running) since 六 2018-05-26 07:02:54 CST; 1 months 11 days ago
     Process: 1412 ExecStartPost=/usr/bin/mysql-systemd-start post (code=exited, status=0/SUCCESS)
     Process: 882 ExecStartPre=/usr/bin/mysql-systemd-start pre (code=exited, status=0/SUCCESS)
    Main PID: 1411 (mysqld_safe)
      CGroup: /system.slice/mysqld.service
              └─1411 /bin/sh /usr/bin/mysqld_safe --basedir=/usr
                └─2137 /usr/sbin/mysqld --basedir=/usr --datadir=/var/lib/mysql --plugin-dir=/usr/lib64/...
```

2. 查看MySQL版本

```
mysql -V
```

```
[hadoop@namenode1 ~]$ mysql -V
mysql Ver 14.14 Distrib 5.6.40, for Linux (x86_64) using EditLine wrapper
```

### 3. 连接MySQL，默认root密码为空

```
mysql -u root (这个命令不好用，用 mysql -u root -p)
```

```
mysql> s
```

这里如果使用 `> mysql -u root` 会报以下错误

```
ERROR 1044 (42000): Access denied for user ''@'localhost' to database 'mysql'
```

### 4. 查看数据库

```
mysql> show databases; (注意：必须以分号结尾，否则会出现续行输入符">")
```

### 5. 创建hive元数据数据库 (metastore)

```
mysql> create database hive;
```

```
mysql> create database hive;
Query OK, 1 row affected (0.00 sec)

mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| hive |
| mysql |
| performance_schema |
+-----+
4 rows in set (0.00 sec)
```

### 6. 创建用户hive，密码是123456

```
mysql> CREATE USER 'hive'@'%' IDENTIFIED BY '123456';
```

注意：删除用户是DROP USER命令

### 7. 授权用户hadoop拥有数据库hive的所有权限

```
mysql> GRANT ALL PRIVILEGES ON hive.* TO 'hive'@'%' WITH GRANT OPTION;
```

### 8. 查看新建的MySQL用户 (数据库名: mysql, 表名: user)

```
mysql> select host,user,password from mysql.user;
```

```
mysql> select host,user,password from mysql.user;
+-----+-----+-----+
| host      | user | password |
+-----+-----+-----+
| localhost | root |          |
| namenode1 | root |          |
| 127.0.0.1 | root |          |
| :::1      | root |          |
| localhost | root |          |
| namenode1 | root |          |
| %         | hive | *6BB4837EB74329105EE4568DDA7DC67ED2CA2AD9 |
+-----+-----+-----+
7 rows in set (0.01 sec)
```

### 9. 删除空用户记录，如果没做这一步，新建的hive用户将无法登录，后续无法启动hive客户端

```
mysql> delete from mysql.user where user="";
```

### 10. 刷新系统授权表 (不用重启mysql服务)

```
mysql> flush privileges;
```

## 11. 测试hive用户登录

```
mysql -u hive -p
```

```
Enter password: 123456
```

## 三、安装和配置hive

### 1. 下载hive

```
Wget https://mirrors.tuna.tsinghua.edu.cn/apache/hive/hive-2.3.5/apache-hive-2.3.5-bin.tar.gz
```

### 2. 解压hive-1.1.0-cdh5.12.1.tar.gz到/home/hadoop

```
$ tar zxvf apache-hive-2.3.5-bin.tar.gz
```

### 3. 在.bash\_profile文件中添加hive环境变量

```
export HIVE_HOME=/home/hadoop/hive-1.1.0-cdh5.12.1
```

```
export PATH=HIVE_HOME/bin :PATH
```

### 4. 使上述设置生效

```
$ source .bash_profile
```

### 5. 编辑\$HIVE\_HOME/conf/hive-env.sh文件，在末尾添加HADOOP\_HOME变量

```
cd $HIVE_HOME/conf
```

```
cp hive-env.sh.template hive-env.sh （默认不存在，可从模板文件复制）
```

```
vi hive-env.sh
```

```
HADOOP_HOME=/root/Hadoop/hadoop-2.8.5
```

### 6. 新建\$HIVE\_HOME/conf/hive-site.xml文件

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
  <property>
    <name>javax.jdo.option.ConnectionDriverName</name>
    <value>com.mysql.jdbc.Driver</value>
  </property>
  <property>
    <name>javax.jdo.option.ConnectionURL</name>
    <value>jdbc:mysql://localhost:3306/hive</value>
  </property>
  <property>
    <name>javax.jdo.option.ConnectionUserName</name>
    <value>hive</value>
  </property>

```

```

        </property>
        <property>
            <name>javax.jdo.option.ConnectionPassword</name>
            <value>123456</value>
        </property>

        <property>
            <name>hive.metastore.warehouse.dir</name>
            <value>/hive/warehouse</value>
        </property>
        <property>
            <name>hive.exec.scratchdir</name>
            <value>/hive/tmp </value>
        </property>
        <property>
            <name>hive.metastore.schema.validation</name>
            <value>>false</value>
        </property>
    </configuration>

```

7. 在HDFS上创建数据仓库目录（用于存放hive数据文件）和临时目录

```
hdfs dfs -mkdir -p /hive/warehouse /hive/tmp
```

8. 下载mysql连接驱动，下载地址：<https://dev.mysql.com/downloads/connector/j/>

下载文件(.tar.gz)解压后，将其中的mysql-connector-java-8.0.13.jar文件上传到  
\$HIVE\_HOME/lib目录下

9. 启动hive

```
hive
```

10. 查看hive数据库（注意：命令以分号结尾）

```
hive> show databases;
```

default是默认数据库

11. 退出hive

```
hive> quit;
```

**Hive> Show databases; 报错**

```
hive> show databases; FAILED: SemanticException
org.apache.hadoop.hive.ql.metadata.HiveException:
org.apache.hadoop.hive.ql.metadata.HiveException: MetaException(message:Hive
metastore database is not initialized. Please use schematool (e.g. ./schematool -initSchema -
```

dbType ...) to create the schema. If needed, don't forget to include the option to auto-create the underlying database in your JDBC connection string (e.g. ?createDatabaseIfNotExist=true for mysql))

在HIVE\_HOME/conf/hive-site.xml 中添加如下配置

```
<property>
<name>datanucleus.schema.autoCreateAll</name>
<value>true</value>
</property>
```

### 3. Impala

1. 先去[http://archive.cloudera.com/beta/impala-kudu/redhat/7/x86\\_64/impala-kudu/0/RPMS/x86\\_64/](http://archive.cloudera.com/beta/impala-kudu/redhat/7/x86_64/impala-kudu/0/RPMS/x86_64/)下载所需的包
2. 依次安装这些包

```
rpm -ivh bigtop-utils-xxx.rpm
rpm -ivh impala-xxx.rpm
rpm -ivh impala-xxx.rpm
rpm -ivh impala-xxx.rpm
rpm -ivh impala-xxx.rpm
rpm -ivh impala-xxx.rpm
rpm -ivh impala-xxx.rpm
rpm -ivh impala-xxx.rpm
```

#### 3. impala 配置

- 3.1 添加hadoop安装目录下的core-site.xml,hdfs.xml 和 hive的hive-site.xml 到/etc/impala/conf
- 3.2 修改文件 /etc/default/bigtop-utils , 新增java\_home路径;
- 3.3 修改文件 /etc/default/impala, 只需修改前两行, 改为主节点的ip地址或者hostname, 若/etc/hosts文件配置了 127.0.0.1 localhost , 也可不做修改
- 3.4 修改core-site.xml, 新增以下几项:

```

<property>
    <name>dfs.client.read.shortcircuit</name>
    <value>true</value>
</property>
<property>
    <name>dfs.client.read.shortcircuit.skip.checksum</name>
    <value>false</value>
</property>
<property>
    <name>dfs.datanode.hdfs-blocks-metadata.enabled</name>
    <value>true</value>
</property>

```

3.5 修改hdfs-site.xml, 新增以下几项:

```

<property>
    <name>dfs.datanode.hdfs-blocks-metadata.enabled</name>
    <value>true</value>
</property>
<property>
    <name>dfs.block.local-path-access.user</name>
    <value>impala</value>
</property>
<property>
    <name>dfs.client.file-block-storage-
locations.timeout.millis</name>
    <value>60000</value>
</property>

```

### 3.6 权限配置

1. sermod -G hdfs,hadoop impala
2. groups impala

3.7 创建impala在hdfs目录, 赋予权限(单节点即可):

1. hdfs dfs -mkdir /user/impala
2. hadoop fs -chown impala /user/impala\*

4. 启动impala 之前, 先启动hadoop,hiveserver2的服务(若配置了, 否则启动hiveserver服务)
5. 启动impala服务, 主机节点即可, 从机可以不启动impala-server服务, 所示的ip为刚才配置文件所配的ip或者为ip对应的 hostname, 未修改则为127.0.0.1:

```
[root@master run]# service impala-state-store restart --
kudu_master_hosts=192.168.174.132:7051
Stopped Impala State Store Server: [ 确定 ]
Started Impala State Store Server (statestored): [ 确定 ]
[root@master run]# service impala-catalog restart --
kudu_master_hosts=192.168.174.132:7051
Stopped Impala Catalog Server: [ 确定 ]
Started Impala Catalog Server (catalogd) : [ 确定 ]
[root@master run]# service impala-server restart --
kudu_master_hosts=192.168.174.132:7051
Stopped Impala Server: [ 确定 ]
Started Impala Server (impalad): [ 确定 ]
[root@master run]#
```

## 6. 启动 impala-shell

基于 [https://blog.csdn.net/qq\\_41792743/article/details/87979146](https://blog.csdn.net/qq_41792743/article/details/87979146)

# 需求分析

## 1. 事件分析

- 用户在产品上的行为我们定义为事件，它是用户行为的一个专业描述，用户在产品上的所有获得的程序反馈都可以抽象为事件进行采集。事件可以通过埋点、通过可视化圈选生效，此文档以埋点采集为主。当然，你可以自定义事件的名称、属性的名称以及个数
- 分析单个事件随时间的变化趋势。
- 根据事件的某个指标观察变化趋势
- 根据用户属性或事件属性进行**分组对比**；

基于以上我们得到事件分析的分析目标：

- 对一个指标进行分析，如“支付订单”的“触发用户数”，
- 分析指标可包括“总次数”、“触发用户数”、“人均次数”、“去重用户数”
- 用户可自行选择事件进行分析，如：支付订单的触发用户数这一事件
- 用户可按分组/维度查看分析指标，如按广告来源分组查看支付订单的用户数
- 用户可选择不同时间范围进行查看

基于我们的分析目标，及神策官网的使用手册，我们设计了包括以上功能的事件分析界面，具体参见界面设计部分。

## 2. 漏斗分析

- 漏斗模型主要用于分析一个多步骤过程中每一步的转化与流失情况。
- 选择需要分析的日期  
用户可以选择需要分析的起始时间
- 点击创建漏斗  
用户可以自己选择创建若干漏斗过程。
- 漏斗图展示



用户选择时间和漏斗后点击提交，系统会为用户画出漏斗图，图中标记出每个过程的用户数，相邻漏斗的面积对比即是该过程的转化率。

### 3. 留存分析

- 用户选择分析的时间段

用户可以自主选择分析的起止时间，粒度为日

- 用户选择初始行为

初始行为选择用户只触发一次的事件，比如“注册”、“上传头像”、“激活设备”等。

- 用户选择后续行为

后续行为选择你期望用户重复触发的事件，比如“阅读文章”、“发帖”、“购买”等。这种留存用于对比分析不同阶段开始使用产品的新用户的参与情况，从而评估产品迭代或运营策略调整的得失。

### 4. 功能展示

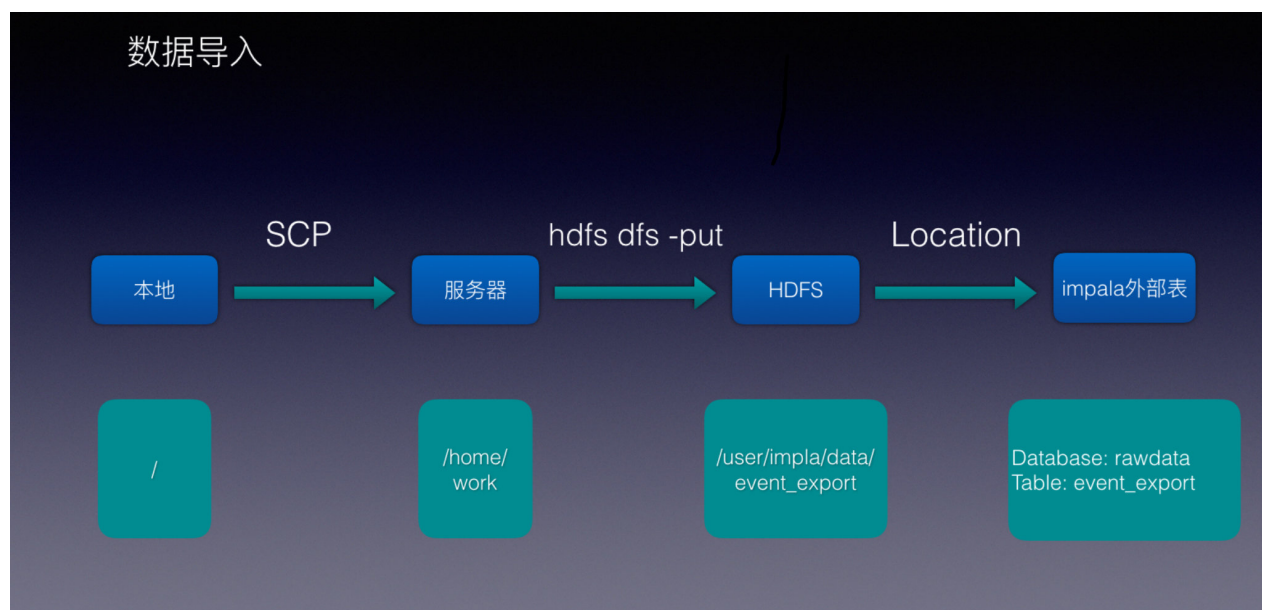
- 用户通过网页表单选择功能需求
- 后端接收网页传来的数据

### 5. 需求归约

### 6. 数据字典

## 数据导入

将数据文件拷贝到HDFS上，然后建立一张impala外部表，将外部表的存储位置，指向数据文件



1. 用scp将数据文件传到服务器
2. 在HDFS上建立存储数据的目录

```
su hdfs  
hdfs dfs -mkdir -p /user/impala/data /user/impala/data/event_export  
/user/impala/data/user_export
```

3. 修改HDFS目录权限（如果需要）

```
hdfs dfs -chmod 777 /user/impala/data/event_export
```

#### 4. 将数据文件传到HDFS指定目录上

```
hdfs dfs -put /home/work/event_export/xxxxxx.xxx /user/impala/data/event_export
```

```
hdfs dfs -put /home/work/user_export/xxxxxx.xxx /user/impala/data/user_export
```

#### 5. 在impala-shell中建立外部表，并指向数据文件

```
Impala-shell > CREATE TABLE rawdata.event_export ( event_id INT, month_id INT,  
week_id INT, user_id BIGINT, distinct_id STRING, time TIMESTAMP, day INT,  
event_bucket INT, _offset BIGINT, p_app_version STRING, ... ) STORED AS TEXTFILE  
LOCATION '/user/impala/data/event_export '
```

## 总体设计


---

### 1. 事件分析

1. 用户选择时间段
2. 用户选择事件（行为）-> 事件下拉框
3. 用户选择事件的展示指标 -> 指标下拉框（5个指标max）-> 指标通过字典映射到sql
4. 用户选择按某种指标分组
  - 3.1 展示指标：总次数、总人数、去重人数、人均次数、平均事件时长、
  - 4.1 分组指标：广告系列来源 -> 来源分析可帮助用户进行广告投放、是否首次访问

### 2. 漏斗分析

漏斗流程：

：点击忘记密码id=5 -> 找回密码-获取验证码id=19 -> 找回密码-重置密码id=28 -> 提交新密码id=1

1. 用户选择需要查询过滤的年，月
2. 用户按顺序选择需要过滤的流程（4步）
3. 返回本月中对应流程的人数和转化比例

### 3. 留存分析

1. 用户选择时间段
2. 用户初始行为
3. 用户选择后续行为
4. 展示时间段内7天留存的结果分析：总人数，1天之内比例，第二天比例...第七天比例
  - 4.1 返回的结构是一张从from\_time到to\_time这么多行，每行元素是总人数，1天，2天...第七天比例 这么多列的表

## 界面设计

---

按照神策的文档，我们实现了一个阉割版的界面

- 对于事件分析，我们允许用户选择
  - 事件的时间区间
  - 分析指标
  - 分组展示方式

起始日期: 2014/01/13 终止日期: 2016/01/13

选择事件

App 元素点击

选择feature

总次数

选择分组展示方式

按广告系列来源分组

提交

2016年01月						
周日	周一	周二	周三	周四	周五	周六
27	28	29	30	31	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31	1	2	3	4	5	6

- 对于留存分析，用户可以选择
  - 事件起止日期
  - 用户初始行为
  - 用户后续行为

起始日期: 2014/01/13 终止日期: 2016/01/13

选择用户初始行为

App 元素点击

选择用户后续行为

App 元素点击

提交

2016年01月						
周日	周一	周二	周三	周四	周五	周六
27	28	29	30	31	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31	1	2	3	4	5	6

- 对于漏斗分析，用户可以选择
  - 年份
  - 月份
  - 构成漏斗的行为 X 4

选择年份

00年

选择月份

01月

选择漏斗

App 元素点击

选择漏斗

App 元素点击

选择漏斗

App 元素点击

选择漏斗

App 元素点击

提交

## 功能实现

我们基于impyla 包实现用Python连接impala，在Python中编辑impala-SQL语句，通过远程提交查询请求来使impala做出响应。

### 1. 事件分析

基于我们的需求，我们将用户前台返回的from\_time, to\_time, event\_id, feature, group传入函数中进行处理。各个参数具体解释如下：

**from\_time:** 用户选择的起始时间

**to\_time:** 用户选择的结束时间

**event\_id:** 用户选择要分析的时间

**feature:** 用户选的要分析的指标，考虑到用户可能选择分析总人数、总次数、平均事件时长、人均次数、去重人数等不同分析指标，我们将这些指标存到一个字典中进行处理，具体如下：

```
features = {  
    "0": "", # 总次数  
    "1": "", # 总人数  
    "2": "", # 去重人数  
    "3": "", # 人均次数  
    # "4": "", # 平均事件时长  
}
```

**group:** 用户选择要分组展示的内容，我们采用了与feature一样的处理办法，将用户可能选择的分组情况如：广告系列来源（运营商）、是否首次访问（制造商）存入字典中方便后续进行处理，具体如下：

```
groups = {
    "0": "", # 广告系列来源分组->运营商
    "1": "", # 是否首次访问分组->设备制造商
}
```

在事件分析中，分析的主要目标是要将用户选择的时间按照选择的feature进行展示，根据用户操作流程我们按一下流程设计并实现了函数功能。

#### 1. 用户在页面选择

- 时间段( yyyy-mm-dd,yyyy-mm—dd), 即查询的起始日期、终止日期
- 选择查询事件: event\_id
- 选择分析指标: feature
- 选择分组展示方式: group

#### 2. 首先将前端传入的时间段转换成UnixTimestamp，后考虑到我们根据day进行了partition的优化，再将UnixTimestamp转成day，加快查询速度。

```
from_time += " 00:00:00"
to_time += " 00:00:00"
from_time = time.strptime(from_time, "%Y-%m-%d %H:%M:%S")
from_day = str(int(time.mktime(from_time) // 86400))
to_time = time.strptime(to_time, "%Y-%m-%d %H:%M:%S")
to_day = str(int(time.mktime(to_time) // 86400))
```

#### 3. 然后筛选出用户选定时间段内，与用户选定事件event\_id的数据，进行数据预处理，同时创建一个view方便后续查询。

```
create_string = "create view sample_event as select * from
event_export_partition where event_id=" + event_id + " and " + \
    from_day + " <day and day< " + to_day

cur.execute('use group7')
cur.execute('drop view if exists group7.sample_event')
cur.execute(create_string)
```

#### 4. 在功能设计时，我们的函数功能是让用户可以自行选择要分析的指标，因此接下来我们要针对不同的指标编写不同的sql语句，根据用户输入的feature不同，采用字典的索引方式选择不同的sql语句进行执行，具体如下。

```
features = {
    "0": "", # 总次数
    "1": "", # 总人数
    "2": "", # 去重人数
    "3": "", # 人均次数
    # "4": "", # 平均事件时长
}

features["0"] = "select count(time),day from sample_event group by day
order by day"
```

```

features["1"] = "select count(user_id),day from sample_event group by day
order by day"
features["2"] = "select count(distinct user_id),day from sample_event group
by day order by day"
features["3"] = "select count(time)/count(distinct user_id),day from
sample_event group by day order by day"
# features["4"]="select sum(p__event_duration)/count(p__event_duration),day
from sample_event group by day"

f = {
    "0": "count(time)",
    "1": "count(user_id)",
    "2": "count(distinct user_id)",
    "3": "count(time)/count(distinct user_id)"
}
cur.execute(features[feature])
feature_result = cur.fetchall()
feature_result = [list(x) for x in feature_result]

```

5. 对用户选择的按不同分组展示的group进行处理，我们采用与feature类似的设计方式，即设计group字典，将用户选择的不同group根据索引映射到不同的sql语句进行执行。这里只起到一个演示作用，我们只写了两个group指标，后续可以根据数据特点，不同需求进行指标数量的添加。

```

groups = {
    "0": "", # 广告系列来源分组->运营商
    "1": "", # 是否首次访问分组->设备制造商
}
groups["0"] = "select " + f[feature] + ",p__carrier,day from sample_event
group by day,p__carrier order by day"
groups["1"] = "select " + f[feature] + ",p__manufacturer,day from
sample_event group by day,p__manufacturer order by day"

cur.execute(groups[group])
group_result = cur.fetchall()
group_result = [list(x) for x in group_result]

```

6. 总的函数：

```

def event(host,from_time, to_time, event_id, feature,
          group): # from_time: "2019-01-01", event_id: str, feature: str,
group: str
    conn = connect(host=host, port=21050)
    cur = conn.cursor()
    features = {
        "0": "", # 总次数
        "1": "", # 总人数
        "2": "", # 去重人数
        "3": "", # 人均次数
    }

```

```

        # "4": "", # 平均事件时长
    }
    groups = {
        "0": "", # 广告系列来源分组->运营商
        "1": "", # 是否首次访问分组->设备制造商
    }
    from_time += " 00:00:00"
    to_time += " 00:00:00"
    from_time = time.strptime(from_time, "%Y-%m-%d %H:%M:%S")
    from_day = str(int(time.mktime(from_time) // 86400))
    to_time = time.strptime(to_time, "%Y-%m-%d %H:%M:%S")
    to_day = str(int(time.mktime(to_time) // 86400))

    create_string = "create view sample_event as select * from
event_export_partition where event_id=" + event_id + " and " + \
        from_day + " <day and day< " + to_day

    cur.execute('use group7')
    cur.execute('drop view if exists group7.sample_event')
    cur.execute(create_string)

    features["0"] = "select count(time),day from sample_event group by day
order by day"
    features["1"] = "select count(user_id),day from sample_event group by day
order by day"
    features["2"] = "select count(distinct user_id),day from sample_event group
by day order by day"
    features["3"] = "select count(time)/count(distinct user_id),day from
sample_event group by day order by day"
    # features["4"]="select sum(p__event_duration)/count(p__event_duration),day
from sample_event group by day"

    f = {
        "0": "count(time)",
        "1": "count(user_id)",
        "2": "count(distinct user_id)",
        "3": "count(time)/count(distinct user_id)"
    }

    # groups["0"] = "select "+f[feature]+",p_utm_source,day from sample_event
group by day,p_utm_source order by day"
    # groups["1"] = "select "+f[feature]+",p_is_first_time,day from
sample_event group by day,p_is_first_time order by day"
    groups["0"] = "select " + f[feature] + ",p__carrier,day from sample_event
group by day,p__carrier order by day"
    groups["1"] = "select " + f[feature] + ",p__manufacturer,day from
sample_event group by day,p__manufacturer order by day"

    cur.execute(features[feature])

```

```

feature_result = cur.fetchall()
feature_result = [list(x) for x in feature_result]
# for x in feature_result:
#     x[1] = str(datetime.datetime.fromtimestamp(x[1] * 86400))[:10]

cur.execute(groups[group])
group_result = cur.fetchall()
group_result = [list(x) for x in group_result]
# for x in group_result:
#     x[2] = str(datetime.datetime.fromtimestamp(x[2] * 86400))[:10]
return feature_result, group_result

```

## 2. 漏斗分析

基于我们的要求，用户需先选择待分析的时间，之后可以选择不同的步骤进行漏斗过滤。根据前端设计的界面，本项目允许用户添加四个步骤进行漏斗分析。基于以上流程，需传入的数据如下：

event\_ids,quary，具体解释如下：

**event\_ids：** 存储用户选择的四个步骤的event\_id

**quary：** 包含用户选择的时间字段，格式为： [year, month]

根据impala中存储数据的特点，首先要将quary中存储的年，月进行处理

```

# quary处理
from_month = "" + quary[0] + "-" + quary[1] + "-01 00:00:00.000000000"
if int(quary[1]) < 12:
    to_month = "" + quary[0] + "-" + "{:0>2d}".format(int(quary[1]) + 1) +
"-01 00:00:00.000000000"
else:
    to_month = "" + str(int(quary[0]) + 1) + "-01-01 00:00:00.000000000"

```

在进行漏斗的过程中，我们定义的漏斗过滤的时长为2小时，具体通过timecmp函数实现，参加后续流程。

### 1. 用户在界面选择

- 时间段 quary ( yyyy-mm-dd,yyyy-mm—dd)
- 选择的漏斗流程 event\_ids

### 2. 首先处理用户选择的查询时间，将其转换成UnixTimestamp，考虑到漏斗分析的过滤时间为2个小时若将quary转成day完全没有必要，漏斗分析中没必要按照day进行聚合或进行查询等，因此将其转成time即可。具体流程如下：



```
# quarry处理
from_month = "" + quarry[0] + "-" + quarry[1] + "-01 00:00:00.000000000'"
if int(quarry[1]) < 12:
    to_month = "" + quarry[0] + "-" + "{:0>2d}".format(int(quarry[1]) + 1) +
"-01 00:00:00.000000000'"
else:
    to_month = "" + str(int(quarry[0]) + 1) + "-01-01 00:00:00.000000000'"

```

3. 接下来对待查询的表进行预处理，同样的筛选出用户选定时间段内，只包含用户选中的过滤流程 event\_ids的数据，方便后续进一步查找。

```
create_string = "create view sample_funnel as select user_id, event_id,
time from event_export_partition where event_id in" + \
    str(event_ids) + " and " + from_month + " <time and time< "
+ to_month
cur.execute('use group7')
cur.execute('drop view if exists group7.sample_funnel')
cur.execute(create_string)

```

4. 根据漏斗分析的特点与最终应输出结果，我们首先筛选出第一步流程中参与人数，记为count0,之后通过join on，添加event\_id的限制条件表示用户选择的每个步骤，筛选出每个步骤中完成上一步骤的人数，记为count1,count2,count3具体实现如下：

```
create_string = "select count(t1.time),count(t2.time), count(t3.time) from
(select * from sample_funnel where event_id=" \
    + str(event_ids[0]) + ") t0" + \
    " left join (select * from sample_funnel where event_id=" +
str(event_ids[1]) + ") t1" + \
    " on t0.user_id=t1.user_id and t0.time<t1.time and
timestamp_cmp(t0.time + interval 120 minutes, t1.time)=1" + \
    " left join (select * from sample_funnel where event_id=" +
str(event_ids[2]) + ") t2" + \
    " on t1.user_id=t2.user_id and t1.time<t2.time and
timestamp_cmp(t1.time + interval 120 minutes, t2.time)=1" + \
    " left join (select * from sample_funnel where event_id=" +
str(event_ids[3]) + ") t3" + \
    " on t2.user_id=t3.user_id and t2.time<t3.time and
timestamp_cmp(t2.time + interval 120 minutes, t3.time)=1"
cur.execute(create_string)
data = cur.fetchall()
count1, count2, count3 = data[0][0], data[0][1], data[0][2]

```

## 5. 总的函数

```
def funnel(host,event_ids, quarry): # event_ids->tuple; quarry->[year,month] #
按月份进行漏斗查询
    conn = connect(host=host, port=21050)

```

```

cur = conn.cursor()
# quarry处理
from_month = "" + quarry[0] + "-" + quarry[1] + "-01 00:00:00.000000000'"
if int(quarry[1]) < 12:
    to_month = "" + quarry[0] + "-" + "{:0>2d}".format(int(quarry[1]) + 1) +
"-01 00:00:00.000000000'"
else:
    to_month = "" + str(int(quarry[0]) + 1) + "-01-01 00:00:00.000000000'"

count0 = count1 = count2 = count3 = 0 # count默认为0

# 抽取只含查询状态的数据

# 使用抽样数据演示
# random_sample(200)
# create_string = "create view sample_funnel as select user_id, event_id,
time from random_sample where event_id in" + \
#
#         str(event_ids) + " and " + from_month + " <time and time<
" + to_month

# 总表测试
create_string = "create view sample_funnel as select user_id, event_id,
time from event_export_partition where event_id in" + \
        str(event_ids) + " and " + from_month + " <time and time< "
+ to_month
cur.execute('use group7')
cur.execute('drop view if exists group7.sample_funnel')
cur.execute(create_string)
cur.execute('select count(time) from sample_funnel where event_id=' +
str(event_ids[0]))
count0 = cur.fetchall()[0][0]
create_string = "select count(t1.time),count(t2.time), count(t3.time) from
(select * from sample_funnel where event_id=" \
        + str(event_ids[0]) + ") t0" + \
        " left join (select * from sample_funnel where event_id=" +
str(event_ids[1]) + ") t1" + \
        " on t0.user_id=t1.user_id and t0.time<t1.time and
timestamp_cmp(t0.time + interval 120 minutes, t1.time)=1" + \
        " left join (select * from sample_funnel where event_id=" +
str(event_ids[2]) + ") t2" + \
        " on t1.user_id=t2.user_id and t1.time<t2.time and
timestamp_cmp(t1.time + interval 120 minutes, t2.time)=1" + \
        " left join (select * from sample_funnel where event_id=" +
str(event_ids[3]) + ") t3" + \
        " on t2.user_id=t3.user_id and t2.time<t3.time and
timestamp_cmp(t2.time + interval 120 minutes, t3.time)=1"
cur.execute(create_string)
data = cur.fetchall()
count1, count2, count3 = data[0][0], data[0][1], data[0][2]

```

```
print([count0, count1, count2, count3])
return [count0, count1, count2, count3]
```

### 3. 留存分析

#### 1. 用户在页面选择

- 时间段(yyyy-mm-dd,yyyy-mm—dd)
- 初始事件： event\_id
- 后续事件: event\_id

#### 2. 我们首先要将字符串的时间格式转换成UnixTimestamp

```
from_time += " 00:00:00"
to_time += " 00:00:00"
from_time = time.strptime(from_time, "%Y-%m-%d %H:%M:%S")
from_day = str(int(time.mktime(from_time) // 86400))
to_time = time.strptime(to_time, "%Y-%m-%d %H:%M:%S")
to_day = str(int(time.mktime(to_time) // 86400))
```

#### 3. 在表中查询所有在规定时间段内进行过初始事件的用户，并为他们创建一个临时表user\_init\_event

```
"with user_init_event " \
"as (select user_id, day as init_day " \
"from event_export_partition_parquet_g7 " \
"where event_id = "+ event_init + " and day >= "+from_day+" \
and day <= "+to_day+" ),"
```

#### 4. 将事件表和user\_init\_event表按照user\_id join，并筛选出其中事件为后续事件并且后续事件和初始事件的时间间隔在0-7天，把这些用户的id,发生初始事件的时间，时间间隔 存到临时表user\_cohort 中。

```
"user_cohort as( " \
"select e.user_id,i.init_day,(e.day-i.init_day) as \
cohort_day " \
"from event_export_partition_parquet_g7 e LEFT JOIN \
user_init_event i on e.user_id = i.user_id " \
"where e.event_id = "+ event_remain+ " and (e.day- \
i.init_day)<7 and (e.day-i.init_day)>=0 " \
"group by user_id,cohort_day,i.init_day)" \
```

#### 5. 在user\_cohort表中，按照初始事件的时间 和 留存时间分组 并以初始时间和留存时间排序，计算每组中的人数。

```
"select count(*),cohort_day,init_day from user_cohort group by \
init_day,cohort_day order by init_day,cohort_day"
```

#### 6. 总的函数

```

def remain2(from_time,to_time,event_init,event_remain):
    from_time += " 00:00:00"
    to_time += " 00:00:00"
    from_time = time.strptime(from_time, "%Y-%m-%d %H:%M:%S")
    from_day = str(int(time.mktime(from_time) // 86400))
    to_time = time.strptime(to_time, "%Y-%m-%d %H:%M:%S")
    to_day = str(int(time.mktime(to_time) // 86400))

    cur.execute("use rawdata")
    create_string = "with user_init_event " \
                    "as (select user_id, day as init_day " \
                    "from event_export_partition_parquet_g7 " \
                    "where event_id = "+ event_init + " and day >= "+from_day+"
and day <= "+to_day+" )," \
                    "user_cohort as( " \
                    "select e.user_id,i.init_day,(e.day-i.init_day) as
cohort_day " \
                    "from event_export_partition_parquet_g7 e LEFT JOIN
user_init_event i on e.user_id = i.user_id " \
                    "where e.event_id = "+ event_remain+ " and (e.day-
i.init_day)<7 and (e.day-i.init_day)>=0 " \
                    "group by user_id,cohort_day,i.init_day)" \
                    "select count(*),cohort_day,init_day from user_cohort group
by init_day,cohort_day order by init_day,cohort_day"

    start = datetime.datetime.now()
    cur.execute(create_string)
    res = cur.fetchall()
    end = datetime.datetime.now()

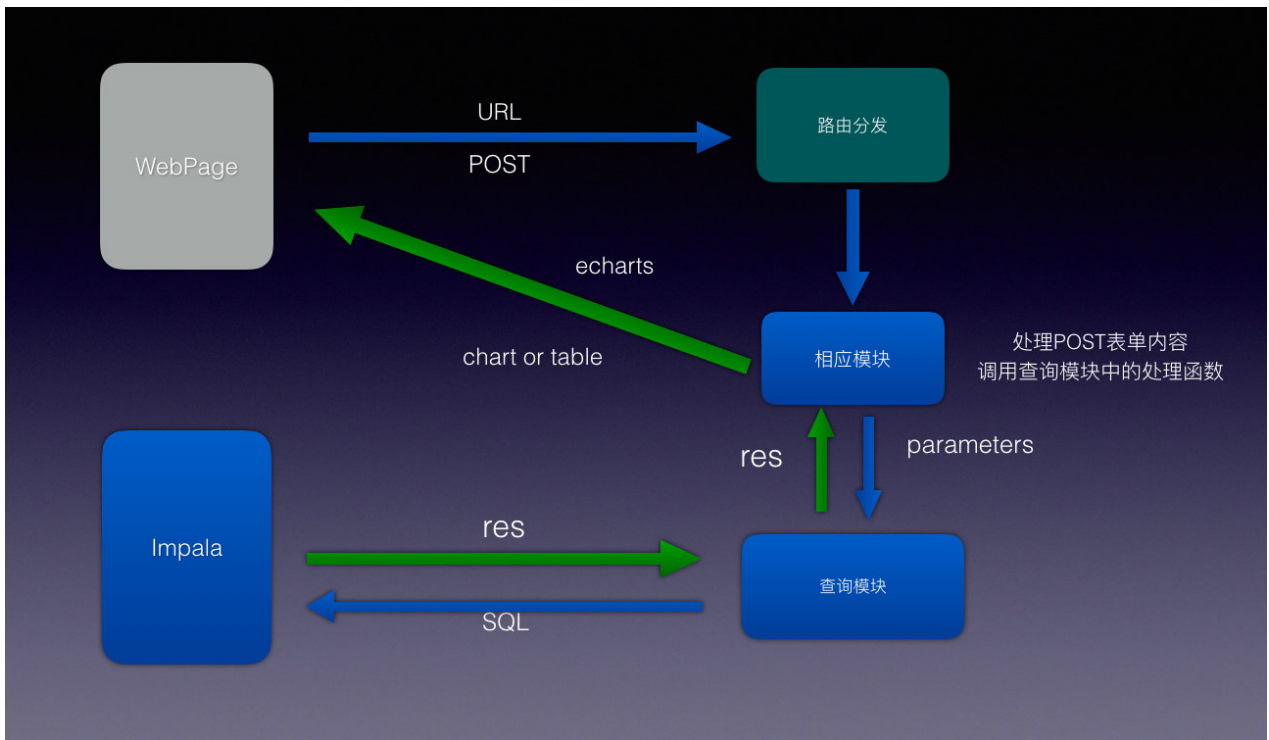
    print(res)
    print(end - start)

```

## 4. web后端

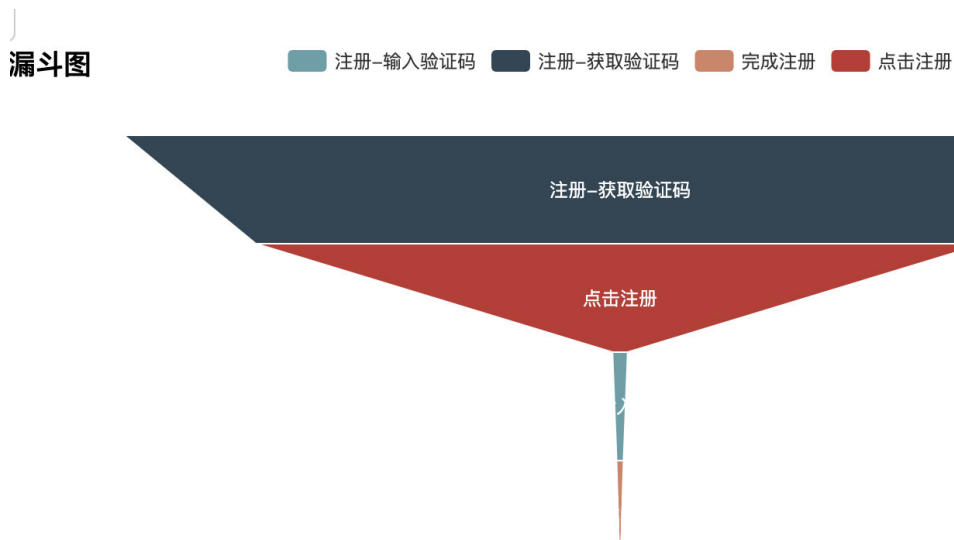
为了方便展示，我们采用web页面的方式向用户提供服务。用户可以在网页上进行设置以选择自己需要的服务形式。

具体实现方法为基于Django模板引擎的Python方法。我们为用户创建funnel,event,remain三个页面。分别对应漏斗分析，事件分析，留存分析。用户在地址栏输入相应URL，用户输入作为POST报文内容传至后端，后端根据url将路由分发到相应的处理模块。处理模块处理用户POST报文中的参数信息。并将这些信息作为参数调用相应的查询方法发送到impala服务器以获得正确的查询结果。

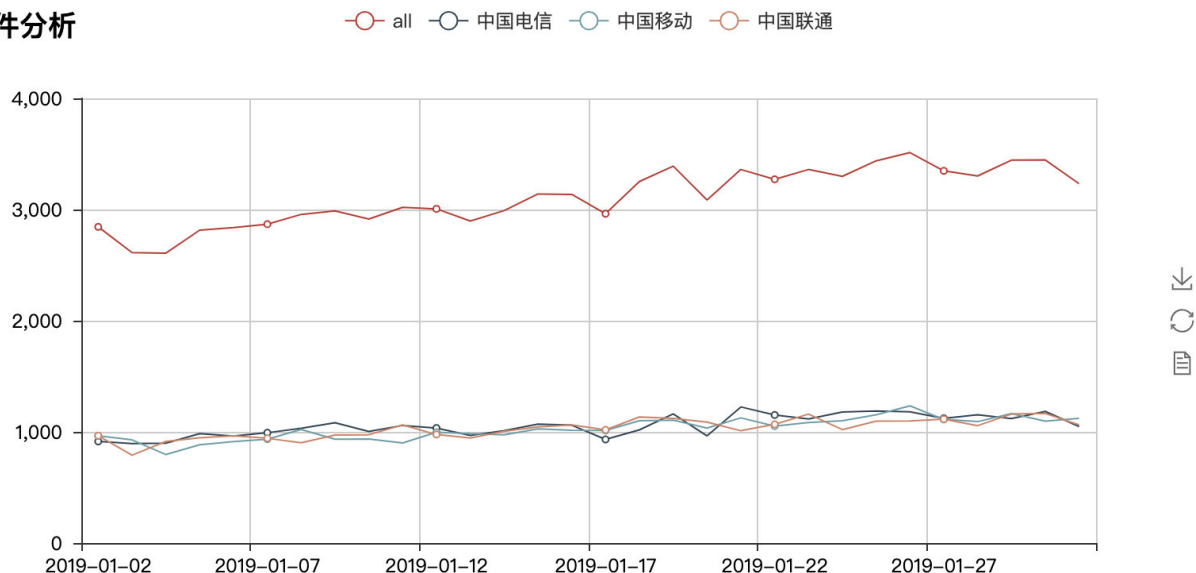


## 5. 可视化

漏斗图和事件分析的图表通过pyecharts绘制。调用pyecharts包里的Line绘制折线图，Funnel绘制漏斗图，



## 事件分析



## 优化方法

### 1. 存储方式

将TEXT数据转换成Parquet存储

### 2. 分区

将数据按照 (day, event\_bucket )分区

```
create table rawdata.parquet_partiton(  
xxx  
)  
select ( xxxx ,day, event_bucket) from xxx  
stored as parquet  
partitioned by(day,event_bucket)
```

3. 在SQL语句中，时间维度上的筛选我们尽量在用day 来作为查询条件，以提高查询效率。

4. 分析SQL语句性能，尽量降低出现 `select *`，同时降低SQL的时间复杂度

5. 在进行查询之前，为了避免在总表中进行查询，由于数据量问题拖慢查询速度，我们对查询数据做了部分与处理，筛选出用户选择范围内的时间与事件，使后续的查询更高效，用户响应时间更短。

## 模拟数据生成

生成模拟数据需对项目三个功能函数进行测试。针对不同的功能需要的数据维度不同，具体如下：

day,time,user\_id,event\_id, p\_utm\_source,event\_bucket

event\_id: 26,8,18,22,27->完成项目创建 event\_bucket:0-19

### 1. 漏斗分析

漏斗分析中需要的数据属性包括：time, event\_id, day, user\_id

根据漏斗分析的特点，我们设计了三套测试流程，生成数据进行测试，具体如下：

1. 点击注册 event\_id=26
2. 注册-获取验证码 event\_id=8
3. 注册-输入验证码 event\_id=18
4. 完成注册 event\_id=22

好像只写完了👉👈这个第一套流程，害没关系剩下的都一样！嗯！

1. 点击忘记密码 event\_id=5
2. 找回密码-获取验证码 event\_id=19
3. 找回密码-重置密码 event\_id=28
4. 提交新密码 event\_id=1
5. 创建项目-选择项目模版 event\_id=16
6. 创建项目-添加团队成员 event\_id=12
7. 创建项目-添加客户 event\_id=15
8. 完成项目创建 event\_id=27

## 2. 事件分析

事件分析中需要的数据属性包括：time, event\_id, event\_feature, day, user\_id

可以查完成注册的人，可以直接用漏斗分析的生成数据进行测试

## 3. 留存分析

留存分析中需要的数据属性包括：time, day, event\_id, user\_id

根据留存分析的特点，我们设计了一下测试流程，生成数据进行测试，具体如下：

1. 初始行为：点击注册 event\_id=26
2. 后续行为：完成注册 event\_id=22

## 正确性测试

根据我们在模拟数据生成时设定的规则，及计算好的每个步骤的比例与预期输出结果，将其作为我们查询结果正确性的标准，使用我们生成好的1亿条数据对函数进行测试。

测试结果具体如下：

1. 漏斗分析
  - 1->2查询结果96%的用户完成第二步行为
  - 2->3查询结果87.5%的用户完成第三步行为
  - 3->4查询结果为85.7%的用户完成第四步行为

由于生成数据并不符合实际商业环境中的规律，因此这里就不贴出更多的运行结果。

## 2. 事件分析

- 查询注册的人数 结果符合上述生成数据结果

## 3. 留存分析

- 查询点击注册到完成项目创建的人数，结果符合上述数据生成预期。

# 性能测试

## 1. 漏斗分析 0.83s

根据网页传入后台的数据，及上述funnel函数的设计，查询返回的count0,count1,count2,count3即分别问完成第一、二、三、四步骤的用户数。

```
Query progress can be monitored at: http://lesson7:25000/query_plan?
query_id=7b4c45698088923d:e8024a3000000000
+-----+
| count(time) |
+-----+
| 13010       |
+-----+
Fetched 1 row(s) in 0.13s
Query progress can be monitored at: http://lesson7:25000/query_plan?
query_id=9b49aa72dee929b7:e48836f700000000
+-----+-----+-----+
| count(t1.time) | count(t2.time) | count(t3.time) |
+-----+-----+-----+
| 17520          | 152            | 64              |
+-----+-----+-----+
Fetched 1 row(s) in 0.70s
```

查询数据量450w++

## 2. 事件分析 0.37s

根据用户选择的分析features，查询返回用户指定形式的执行事件分析指标，这里以执行次数为例：

```
Query progress can be monitored at: http://lesson7:25000/query_plan?
query_id=66465fa641f10258:b983975b00000000
+-----+-----+
| count(time) | day   |
+-----+-----+
| 456         | 17898 |
| 360         | 17899 |
| 380         | 17900 |
| 414         | 17901 |
| 478         | 17902 |
| 416         | 17903 |
| ...        | ...   |
| 548         | 17919 |
```



546	17920
508	17921
518	17922
558	17923
512	17924
512	17925
544	17926
506	17927

Fetches 30 rows in 0.14s

后根据用户选择的不同group因素，进行分组展示查询结果，这里以p\_carrier为例：

Query progress can be monitored at: [http://lesson7:25000/query\\_plan?query\\_id=f24e01f460b00685:940684a200000000](http://lesson7:25000/query_plan?query_id=f24e01f460b00685:940684a200000000)

count(time)	p_carrier	day
164	中国电信	17898
132	中国移动	17898
160	中国联通	17898
120	中国联通	17899
134	中国移动	17899
...	...	...
166	中国联通	17907
182	中国移动	17923
178	中国电信	17924
138	中国移动	17924
196	中国联通	17924
162	中国电信	17925
166	中国移动	17925
184	中国联通	17925
152	中国联通	17926
186	中国电信	17926
206	中国移动	17926
184	中国电信	17927
150	中国联通	17927
172	中国移动	17927

Fetches 90 rows in 0.23s

查询数据量450w++

3. 留存分析 0.39s

根据用户选择的不同起始和结束时间，我们返回起始事件之后的0~7日留存：

count(*)	cohort_day	init_day
----------	------------	----------

```
+-----+-----+-----+
| 123    | 0       | 17897  |
| 2      | 1       | 17897  |
| 4      | 5       | 17897  |
| 2      | 6       | 17897  |
| 153    | 0       | 17898  |
| 1      | 1       | 17898  |
| 1      | 2       | 17898  |
| 3      | 3       | 17898  |
| 3      | 5       | 17898  |
| ...    | ...     | ...    |
| 1      | 2       | 17925  |
| 2      | 3       | 17925  |
| 2      | 4       | 17925  |
| 3      | 5       | 17925  |
| 181    | 0       | 17926  |
| 2      | 1       | 17926  |
| 3      | 2       | 17926  |
| 2      | 3       | 17926  |
| 1      | 4       | 17926  |
| 2      | 5       | 17926  |
| 6      | 6       | 17926  |
+-----+-----+-----+
Fetched 179 row(s) in 0.37s
```

查询数据量450w++