中国大学生服务外包
创新创业大赛

【A12】用户行为分析平台【恒生电子】

【系统设计与实现】

队名：物设商联合队

目 录

一、 核心亮点

【更安全的网络埋点】

在同类型的网络埋点产品中，例如百度统计和诸葛 IO 等，埋点数据都掌握在第三方平台中，这使得埋点数据不具有很强的安全性。大量的数据外泄对企业的发展和用户的信息造成不良影响。基于此，本埋点方案中，开发者通过自建的函数库开发了一套独立的埋点程序，并提供独立的埋点接口和数据库，只有企业内部的人具有访问权限，不需要对第三方平台进行公开。因此，对比其他数据统计的网络产品，该方案是更具安全性的网络埋点方案。企业可以将自己的埋点数据完全的掌握在自己手中，不需要借助第三方平台。

【更精简的埋点架构】

自动埋点只需引入 JavaScript 脚本，手动埋点只需绑定一个埋点函数；所有监控事件全部抽象为事件 + 附加信息，最大程度减轻了前端程序员的埋点难度。

【更高效的数据捕获】

大部分数据在客户端生成，分摊服务器端计算开销；在数据捕获阶段不进行分析操作，尽可能简化数据表结构，使数据存储简单直接，降低了数据存储的开销。

【更灵活的数据分析】

数据分析程序完全独立于系统，只要有 mysql 数据库读取和写入功能的数据分析框架皆可满足开发需求。所有分析结果全部保存在数据库中，一次分析，永久保存，在模块化中避免重复的数据分析。数据分析脚本定时运行，一次运行只分析固定时间内的数据，已分析的时间段不再进行分析，最大程度节约了系统的计算资源。

【更优良的数据呈现】

数据呈现后端所有数据转化操作均只实现自一个接口，继续增减数据转化操作只需在一个接口上进行，最大程度地简化了后端开发；前端网页使用 Echart 框架进行数据呈现，图表清晰简洁，尤其适合多维度的数据呈现；页面加载方式采用 jquery 的 load 方法，使得呈现数据的网页元素也可以自由增减，也使 HTML 结构清晰明了，最大程度地简化了前端开发。

【更友好的二次开发】

埋点捕获数据规整，原始数据结构简洁，数据分析框架灵活，呈现后端结构清晰，因此对二次开发具有高度的友好性，可以很容易地根据公司实际需求进行定制与维护。

二、 系统设计

2.1 概念结构设计

根据需求分析的结果可以知道，本系统作为一个网站埋点分析的应用框架，需要有较高的可扩展性和灵活性，需要在数据收集、数据操作、数据存储、数据呈现之间划清界限，降低它们之间的耦合性从而使系统具有较高的扩展性和灵活性。因此本系统采用“收集 -存储 -处理 -存储 -读取”的方式，使得收集、操作、存储、呈现三者尽可能的互相独立，如下图所示：

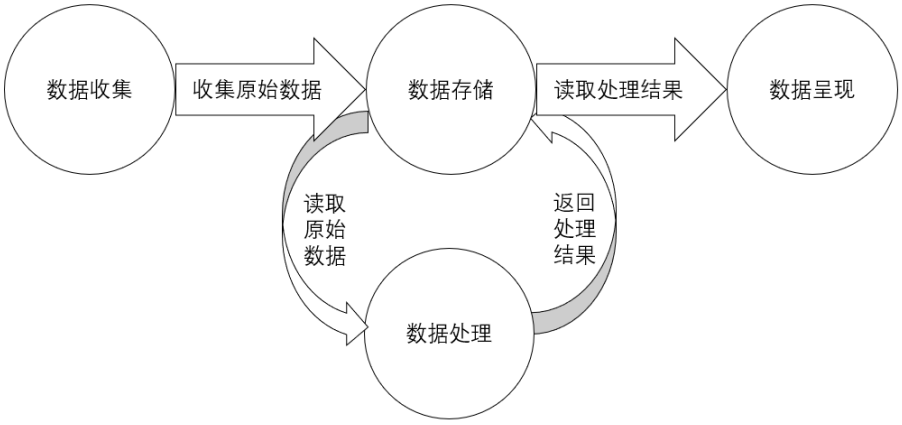

图 1　埋点方案概念结构图

在数据收集部分，收集程序只负责对原始数据进行收集并发往数据存储程序，此时数据存储程序只将收到的数据进行存储，而不进行其他任何操作。在数据处理部分，数据处理程序只负责从数据库中读取原始数据进行分析，然后将分析结果再返还到数据库中。而在数据呈现部分，程序只负责读出数据并跟进数据进行页面渲染和交互。

2.2 逻辑结构设计

2.2.1 数据收集

在本系统中，数据收集部分负责监控前端页面的用户行为，并将原始行为数据写入数据存储部分。为了尽可能增强系统扩展性，在设计数据收集部分的逻辑结构时需要对用户的行为进行一定层次的抽象，并按照抽象用户行为的基本组成设计系统逻辑；同时为了系统的简洁，抽象的用户行为应尽可能的简单明了，不能有复杂的元素和结构。抽象的用户行为描述如下：

用户行为 =(行为标签, 附加信息)

其中，行为标签 ∈ { 注册, 进入某个页面, 点击页面元素, 访问产品,etc.} ，附加信息 ={ 用户名, 时间, 用户所在页面, 用户所用的设备, 用户 IP 地址, 用户 SESSION, 与行为标签有关的附加信息 }。上述抽象方式将每一个用户行为用一个行为标签和一系列附加信息表示，在保证信息的完整性的同时又兼有简单的信息结构。

2.2.2 数据存储

在数据存储的部分中，原始数据为上一节中抽象的用户行为。在数据库中，数据都以数据表的方式存储，故需要将抽象的用户行为结构转化为数据库中的表结构。依照数据库设计的 3NF 法则，可以将抽象的用户行为结构转化为如下表结构：

表 1　数据存储表

字段名称	数据内容
行为标签	行为标签
用户名	用户名文本
时间	时间
用户所在页面	网址
用户所用设备	设备标识
用户 IP 地址	IP 地址
用户 SESSION	SESSION 字符串
与行为标签有关的附加信息	附加信息文本

除原始数据外，数据存储部分还包含一系列由数据处理部分产生的数据分析结果表，这一系列表格不在系统中进行预先定义，而是由数据处理部分按需添加。

2.2.3 数据分析

数据分析部分负责从数据存储部分读出原始数据，进行分析后再将结果写入到数据存储部分。数据处理部分包含的数据处理程序可以模块化的形式自由增减，每个数据处理程序在写入数据前都要根据分析结果的形式构造一个单独的数据表存储分析结果；每个数据分析程序的数据都只能来源于数据库中已有的数据表。因此，数据处理程序之间的依赖关系可以构成一张单起点多终点的有向无环图，其起点为原始数据表，顶点为数据分析程序，边为数据分析程序生成的结果。例如，一个典型的含有表依赖关系的数据分析程序结构图如下：

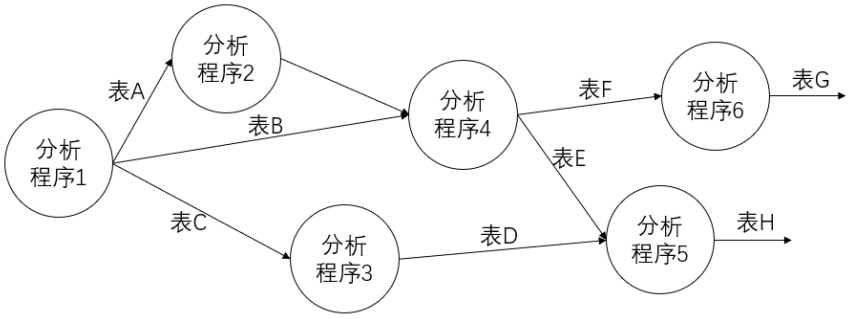

图 2　数据分析程序结构图

其中每个数据分析部分中的数据分析程序只能按照有向无环图所指定的顺序执行并生成结果数据表。

2.2.4 数据呈现

数据呈现部分负责从数据存储部分读出数据，并将数据转化为便于显示的格式发到前端数据呈现页面进行显示。数据呈现部分又可分为两个部分，其中数据转化程序位于服务器，负责读取数据并转化为便于显示的格式发送到数据呈现页面；数据呈现程序位于管理员终端页面，负责接收数据转化程序发来的转化数据并按一定规则进行显示。二者都可以以模块化的形式自由增减，但管理员终端页面的数据呈现程序所呈现的数据必须来源于服务器中的至少一个数据转化程序。例如，一个典型的数据呈现部分的结构图如下：

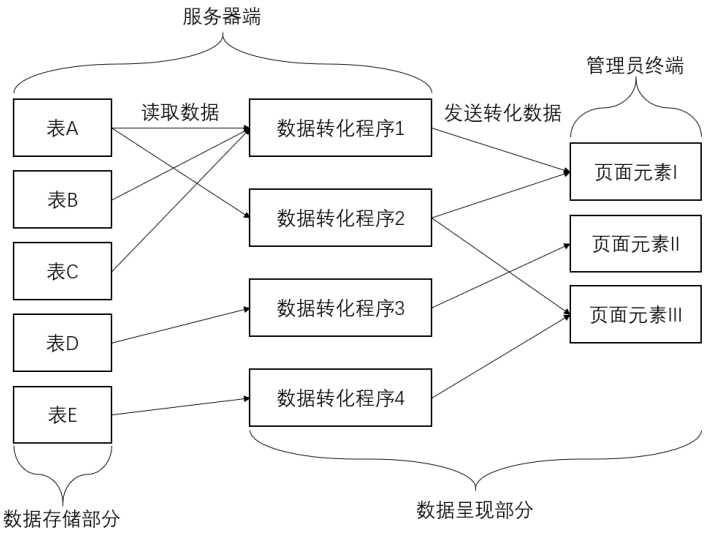

图 3　数据呈现部分结构图

三、 系统实现

3.1 数据收集

数据收集即是监控用户动作并将用户动作数据写入数据库，分前端和后端两个部分实现。

3.1.1 前端

在前端页面，一个 JavaScript 脚本将收集用户的一些基本信息并将其发往后端。其中的 JavaScript 脚本功能及实现如下：

(1)【自动埋点】

该功能将监控页面上的所有可点击元素的点击事件，其实现方法为：

(a) 使用 addEventListener 方法在 document 元素上添加”click”事件监听器，监控页面的所有点击动作。

```
document.addEventListener("click", function (event) {//捕获所有点击
    if(event) {//不是ie
        handleClick(event.target);
    }
    else if(window.event) {//ie
        handleClick(window.event.srcElement);
    }
});
```

(b) 修改 Element.prototype.addEventListener 方法和 Element.prototype.removeEventListener 方法，使用一个全局列表记录页面上所有加载了”click”事件监听器的元素。

```
Element.ClickEventListenersDOMList = [];//用绑定了事件的元素列
    表addEventListenerclick
//重写方法addEventListener
Element.prototype._addEventListener = Element.prototype.addEventListener;
Element.prototype.addEventListener = function (event, func, useCapture) {
    this._addEventListener(event, func, useCapture);
    if(event === "click")
        Element.ClickEventListenersDOMList.push(this);//记下绑定了事件的元素click
};

//重写方法removeEventListener
Element.prototype._removeEventListener=Element.prototype.removeEventListener;
```

```
Element.prototype.removeEventListener=function (event, func, useCapture) {
    this._removeEventListener(event, func, useCapture);
    if(event==='click'){
        var index=Element.ClickEventListenersDOMList.indexOf(this);
        if(index>-1)
            Element.ClickEventListenersDOMList.splice(index,1)
    }
};
```

(c) 每当监控到点击事件时都对被点击元素进行如下判断：

- 是否加载了”click”事件监听器？
- 是否有 onclick 属性？
- 是否是一个可点击的元素（a、button、input 等）？
- 如果任意一个为真，则记录本次点击事件。

```
function handleClick(element) {//处理泛点击事件
    if(canClick(element))//如果点击发生在一个能被点击的元素上面
        //就记录下被点击的元素
        WSK_record({event_type:'点击',event_description:element.outerHTML});
}
function canClick(element) {//判断这个元素能不能被点击
    var canClicktagList = ['A', 'AUDIO', 'BUTTON', 'INPUT', 'AUDIO'];
    if(canClicktagList.indexOf(element.tagName) !== -1 && element.disabled !==
        true)
        return true;//是可点击的标签
    if(Element.ClickEventListenersDOMList.indexOf(element) !== -1)
        return true;//有型的ClickEventListener
    if(typeof element.onclick !== 'undefined' &&
    element.onclick !== null && element.onclick !== '')
        return true;//加了属性onclick
}
```

(2)【手动埋点接口】

该功能为页面设计人员提供了一个记录特殊的点击事件的函数—WSK_recordEvent (event_type,event_description)。使用时只需将该函数加入到需要记录点击事件的元素的点击事件回调函数中（通过 onclick 属性或”click”事件监听器均可），并填入要记录的事件类型 event_type 和事件的附加信息 event_description 即可。

```
function WSK_recordEvent(event_type,event_description) {
    if (typeof WSK_servlet_url === 'undefined' ||
        WSK_servlet_url === null ||
        WSK_servlet_url === '' ||
        !WSK_servlet_url) {
        WSK_record({event_type:event_type});
    }
    else
        WSK_record({event_type:event_type,event_description:event_description})
}
```

(3)【埋点数据收集与发送】

该功能将自动埋点和手动埋点产生数据进行完善并发送到指定的后端 Sevlet 中。其实现方法为：

a. 网站设计人员将后端 Sevlet 的连接赋值给 WSK_servlet_url 变量。
b. 当要记录的点击事件发生后，读取 document.baseURI、navigator.userAgent 和当前时间，与点击事件的事件类型和事件描述组合成 POST 字符串。
c. 通过 window.XMLHttpRequest 功能将组合好的 POST 字符串发到 WSK_servlet_url 变量所指定的 Servlet 中。

3.1.2 后端

在服务器后端，接收并解析前端发来的 POST 字符串可以获得用户所在页面（document.baseURI）、用户浏览器和设备信息（navigator.userAgent）、用户客户端时间、以及点击事件的事件类型和事件描述。在服务器端 Servlet 中可进一步通过 request.getRemoteAddr() 和 request.getSession() 获得用户的 IP 地址和 Session 编号。最终，上述所有信息将被组合成一个 SQL 语句：INSERT INTO “事件记录”(“时间”, “IP 地址”, “设备型号”, “SESSION 编号”, “所在页面”, “用户”, “事件类型”, “事件描述”)VALUES()，并调用 mysql-connector 中的 executeUpdate() 方法写入数据库。

3.2 数据存储

在数据存储的部分中，需要指定数据库中原始数据的存储方法。可以很容易地得到存储原始数据的数据表结构：

除数据表外，为安全起见，还需为应用单独指定一个用户并限定为只可本地连接。综合以上可以得到建立数据库的 sql 脚本：

表 2 原始数据的数据表结构

字段名称	数据类型	字段名称	数据类型
编号	Int	所在页面	Varchar(255)
时间	datetime	用户	Varchar(255)
IP 地址	Varchar(64)	事件类型	Varchar(255)
设备型号	text	事件描述	text
SESSION 编号	Varchar(64)		

```
drop database if exists MoniterData;
create database MoniterData;
use MoniterData;
create table 事件记录 (
编号 bigint primary key auto_increment,
时间 datetime not null,
IP 地址 varchar(64) not null,
设备型号 text not null,
'SESSION 编号' varchar(64),
所在页面 varchar(255) not null,
用户 varchar(255),
事件类型 varchar(255) not null,
事件描述 text
);
drop user if exists MoniterData@localhost;
create user MoniterData@localhost identified by 'MoniterData';
grant create,select,update,insert on MoniterData.* to MoniterData@localhost;
flush privileges;
```

3.3 数据分析

3.3.1 基本流量分析

(1) 日/月总流量：按每日/月统计事件记录表中的事件类型字段为“访问页面”的事件记录总数。

【每月总流量统计】

“SELECT 日期,count(*) FROM(SELECT date(concat(year(时间),‘-’,month(时间),‘-01’)) AS 日期 FROM 事件记录 WHERE 事件类型=‘%s’ AND Date(时间) BETWEEN date_add(‘%s’,interval 1 month) AND date_add(concat(year(now()),‘-’,month(now()),‘-01 00:00:00’),interval -1 second)) AS T GROUP BY 日期 ORDER BY 日期 ASC”

【每日总流量统计】

“SELECT 日期,count(*) FROM(SELECT date(时间) AS 日期 FROM 事件记录 WHERE 事件类型=‘%s’ AND Date(时间) BETWEEN date(‘%s’)+1 AND date(now())-1) AS T GROUP BY 日期 ORDER BY 日期 ASC”

(2) 各页面/各产品总流量：按每日/月和事件描述中的产品名称和所在页面中的网页地址统计事件记录表中的事件类型字段为“访问页面”的事件记录总数。

【每月分页面总流量统计】

“SELECT 日期,count(*) FROM(SELECT concat(year(时间),‘-’,month(时间),‘-’,‘01’) AS 日期 FROM 事件记录 WHERE 事件类型=‘%s’ AND 所在页面=‘%s’ AND date(时间) BETWEEN date_add(‘%s’,interval -1 month) AND date_add(concat(year(now()),‘-’,month(now()),‘-’,‘01 00:00:00’),interval -1 second)) AS T GROUP BY 日期 ORDER BY 日期 ASC”

【每日分页面总流量统计】

“SELECT 日期,count(*) FROM(SELECT date(时间) AS 日期 FROM 事件记录 WHERE 事件类型=‘%s’ AND 所在页面=‘%s’ AND Date(时间) BETWEEN date(‘%s’)+1 AND date(now())-1) AS T GROUP BY 日期 ORDER BY 日期 ASC”

【每月产品流量统计】

“SELECT 月份,count(*) FROM(SELECT concat(year(时间),‘-’,month(时间),‘-’,‘01’) AS 月份 FROM 事件记录 WHERE 事件类型=‘%s’ AND 事件描述=‘%s’ AND 时间 BETWEEN date_add(‘%s’,interval 1 month) AND concat(year(now()),‘-’,month(now()),‘-’,‘01’)) AS T GROUP BY 月份 ORDER BY 月份 ASC”

【每日产品流量统计】

“SELECT 日期,count(*) FROM(SELECT date(时间) AS 日期 FROM 事件记录 WHERE 事件类型=‘%s’ AND 事件描述=‘%s’ AND Date(时间) BETWEEN date(‘%s’)+1 AND date(now())-1) AS T GROUP BY 日期 ORDER BY 日期 ASC”

3.3.2 基础用户分析

(1) 日/月新增用户分析：按每日/月统计事件记录表中的事件类型字段为“注册完成”的事件记录总数。

【每月新增用户统计】

“SELECT 日期,count(*) FROM(SELECT date(时间) AS 日期 FROM 事件记录 WHERE 事件类型 =‘注册完成’AND Date(时间) BETWEEN date(‘%s’)+1 AND date(now())-1) AS T GROUP BY 日期 ORDER BY 日期 ASC”

【每日新增用户统计】

“SELECT 日期,sum(数量) FROM(SELECT date(concat(year(时间),‘-’,month(时间),‘-01’)) AS 日期, 数量 FROM 日新增用户数量 WHERE 时间 BETWEEN date_add(date(‘%s’),interval 1 month) AND date(concat(year(now()),‘-’,month(now()),‘-01’))-1) AS T GROUP BY 日期 ORDER BY 日期 ASC”

(2) 日/月活跃用户分析：统计每月的用户上线天数/每日的用户上线小时数，超过一定阈值的用户作为活跃用户记录在数据表中。

【每日活跃用户分析】

“SELECT DISTINCT concat(date(时间),‘’,hour(时间),‘:00:00’), 用户 FROM 事件记录 WHERE 时间 BETWEEN date_add(‘%s’,interval 1 hour) AND concat(date(now()),‘’,hour(now()),‘:00:00’) ORDER BY 时间 ASC”

“SELECT 用户, 日期,count(日期) FROM (SELECT 用户,date(时间) AS 日期 FROM 用户访问时) AS T WHERE 日期 BETWEEN date_add(‘%s’,interval 1 day) AND date(now())GROUP BY 用户, 日期 ORDER BY 日期 ASC”

“SELECT 时间,count(用户) FROM 日用户活跃度 WHERE 访问数 >=%d AND 时间 BETWEEN date_add(date(‘%s’),interval 1 day) AND date(now())-1 GROUP BY 时间 ORDER BY 时间 ASC”

【每月活跃用户分析】

“SELECT DISTINCT date(时间), 用户 FROM 事件记录 WHERE 时间 BETWEEN date(‘%s’)+1 AND date(now()) ORDER BY 时间 ASC”

“SELECT 用户, 月份,count(月份) FROM (SELECT 用户,date(concat(year(时间),‘-’,month(时间),‘-01’)) AS 月份 FROM 用户访问日) AS T WHERE 月份 BETWEEN date_add(date(‘%s’),interval 1 month) AND date(concat(year(now()),‘-’,month(now()),‘-01’)) GROUP BY 用户, 月份 ORDER BY 月份 ASC”

“SELECT 时间,count(用户) FROM 月用户活跃度 WHERE 访问数 >=%d AND 时间 BETWEEN date_add(date(‘%s’),interval 1 month) AND date_add(concat(year(now()),‘-’,month(now()),‘-01 00:00:00’),interval -1 second) GROUP BY 时间 ORDER BY

时间 ASC”

3.3.3 大数据聚类分析

(1) 完成 spark+hadoop+python+pyspark 的部署。

(2) 使用 from pyspark.sql import SparkSession 初始化。

```
from pyspark.sql import SparkSession
sc = SparkSession.builder.appName("name")\
.config('spark.some.config,option0','some-value')\
    .getOrCreate()
```

(3) 使用 pyspark 的 sql 模块读取数据库。

```
from pyspark.sql import SQLContext
jdbcDf=ctx.read.format("jdbc").options(url,drive,dbtable
,user,password).load()
```

(4) 使用 pyspark 读取数据库。使用 groupBy, count(),orderby() 方法。

(5) 对数据预处理，转化为特征向量，将用户行为打包成特征集，通过 K-means 均值聚类将用户分类，分类标准通过改变模型训练的簇的个数变化来得到最优解。具体步骤为：

```
kmeans = KMeans(k=3, seed=1)
km_model = kmeans.fit(df_km)
# 获取聚类预测结果
transformed = km_model.transform(df_km).select('id', 'prediction')
# 转化pandas dataframe 然后可视化
pd_df = df_pred.toPandas()
```

(6) 利用 pyspark 做基础数据分析，同样读取数据库，利用相关库的方法实现常规大数据统计分析。

```
for i in pd_df['prediction']:
    SQL='UPDATE '用户分析1' SET 'prediction'='+str(i)+' WHERE id='+str(ii)
    print(SQL)
    cursor=dbc.cursor()
    cursor.execute(SQL)
i+=1
```

3.3.4 设备和页面关联性分析

(1) 按先后次序记录每个 session 的访问顺序，记录多个数组，对数组使用 apriori 关联性分析。数据的记录带有自动过滤筛选功能，自动过滤无效数据。

```
# 生成频繁项
def scanD()
# 频繁项组合
def aprioriGen()
#算法apriori
def apriori()
# 生成关联规则
def generateRules()
#设置阈值
def rulesFromConseq()
```

(2) 通过地区解析平台将 ip 地址解析成省市，利用 python 的 urllib 模块将数据爬取到本地记录至服务器。

```
def DiliWeiZhi(ip):
    #处理地理位置
    html = request.Request(r'https://ip.tianqiapi.com/?ip='+ip)
    with request.urlopen(html) as f:
        hjson=json.loads(f.read().decode('utf-8'))
    province=hjson['province']
    city=hjson['city']
    isp=hjson['isp']
```

(3) 通过 python 的 user_agent 模块解析通过 request 获取的设备信息，将设备信息自动分割存储至数据库中。

```
def UserAgent(str1):
    ua_string = str1
    user_agent = parse(ua_string) #解析成user_agent
    bw = user_agent.browser.family #浏览器
    s = user_agent.os.family #操作系统
    juge_pc = user_agent.is_pc #判断是不是桌面系统
    phone = user_agent.device.family#手机型号
    return bw,s,juge_pc,phone
```

(4) 利用 python 定时功能设定具体时间实现数据更新。使用定时框架 APScheduler。

3.4 数据呈现

3.4.1 前端

(1) 实现 html+css+JavaScript 的技术部署。

(2) 使用 echarts 中的柱状图，饼图，地图，折线图，关系图等图表构成用户画像。echarts 包含多级联动，可实现鼠标点击，悬浮等互动功能。echarts 中包含功能按键，通过不同按键实现切换图例，导出，刷新。

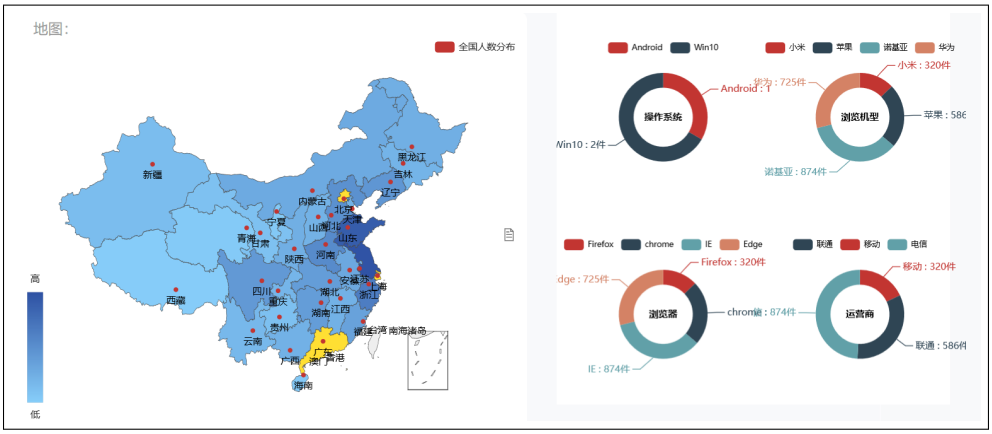

图 4　用户分布图

图 5　流量分析图

(3) 报表导出按钮可以导出整个页面中的数据，通过 jQuery 与按钮进行交互。数据记录在不同的 sheet 中，数据是由 json 转化为 excel。

(4) 利用 jQuery 实现局部刷新功能，无需对页面整体进行刷新。

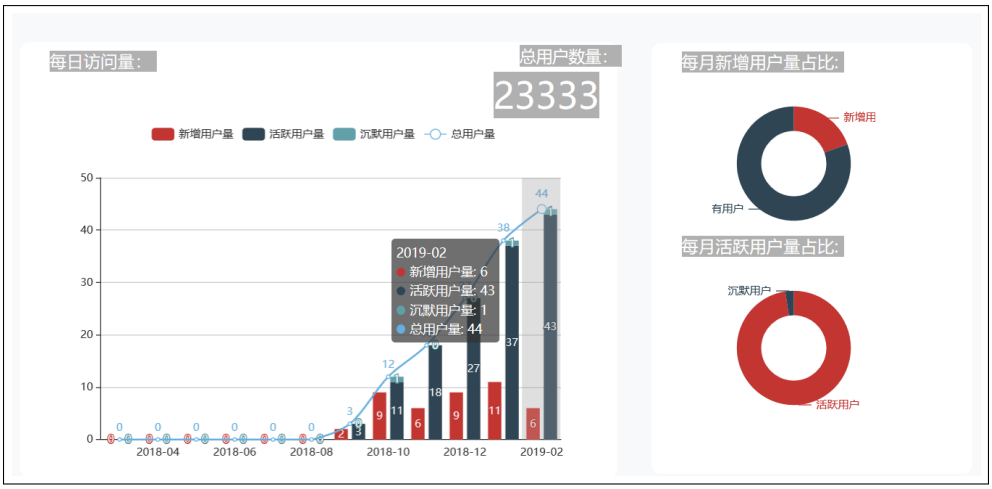

图 6　访问量分析图

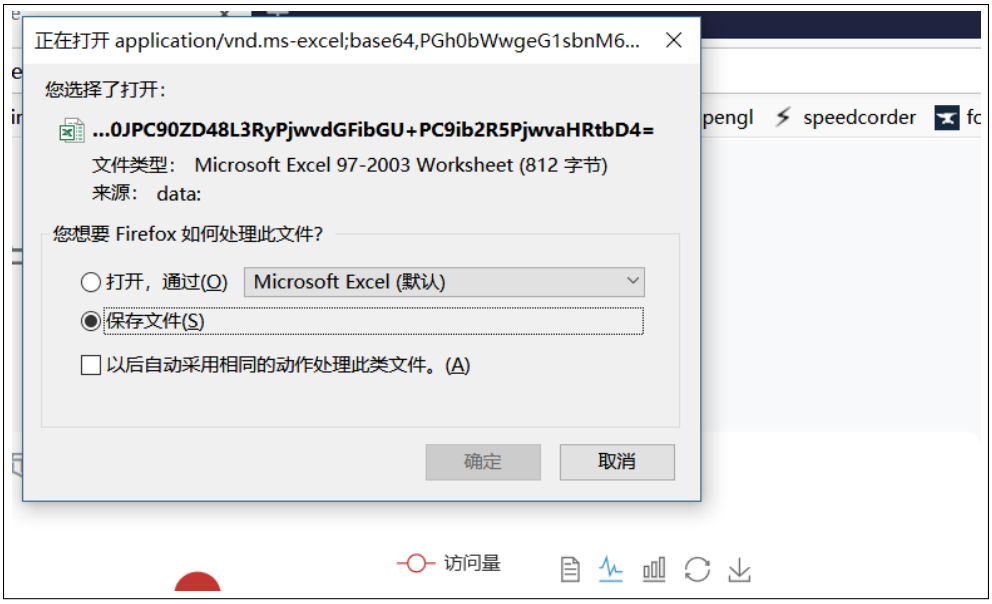

图 7　导出 excel 报表

3.4.2 后端

(1) 导入 java 库函数。

```
import java.sql.*;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import java.io.IOException;
import java.io.PrintWriter;
```

(2) 实现 java 利用 HttpServletRequest request 的 getParameter 方法获取从前端传来的 Post 和 Get 数据。数据是前端监听到的用户基础数据，包含 session，设备信息，访问页面等。

```
String element = request.getParameter("what-we-need");
```

(3) 实现 java HttpServletRequest responce 的 response.getWriter() 向前端发送数据。

```
PrintWriter out = response.getWriter();
response.getWriter().print("suceess!session id = " + sessionId);
```

(4) 使用 request.getSession() 获取用户 session，用户在每次浏览网页的过程中，session 值保持不变。

```
HttpSession session = request.getSession();
```

(5) 后端使用 Java 面向对象方法完成，定义一个通用的数据转化接口 DataConstructor，以及一个使用 DataConstructor 接口统一进行数据转化的类 DataReacher，DataReacher 类将使用一个 DataConstructor 数组存储所有将要使用的数据转化工具，并在调用 DataReacher.reachData() 构造数据时一次性调用数组中的所有类的继承自 DataConstructor 接口的 getName 和 getData 方法构造一个 JSON 格式的数据发往前端。

```
//接口：DataConstructor
package common;
import net.sf.json.JSON;
/**
 * 数据构造器接口，用于从数据库中读数据并构造成JSON
 */
public interface DataConstructor {
    /**
     * @return 返回数据的名称
     */
    public String getName();//获取字段名
    /**
     * @param conn 输入数据库连接
     * @return 返回数据正文
     */
    public JSON getData(DataConnector conn);//获取字段数据
    public static DataConstructor instance = null;
    public DataConstructor getInstance();
```

```
//构造数据DataReacher.reachData():
public JSONObject reachData() {
    JSONObject jo = new JSONObject();
    for (DataConstructor dc : constructors) {
        jo.element(dc.getName(), dc.getData(conn));
    }
    return jo;
}
```