淘宝数据分析项目分析报告

项目介绍

项目涉及数据预处理、消息队列发送和接收消息、数据实时处理、数据实时推送和实时展示等数据处理全流程所涉及的各种典型操作，涵盖Linux、Spark、Kafka、Flask、Flask-SocketIO、Highcharts.js、sockert.io.js、PyCharm等系统和软件的安装和使用方法 。

本项目主要从以下几个做分析

- 男女生购物人数
- 买家的行为（加入购物车、购买和收藏）
- 买家的年龄分段的分布

实验的步骤

- 实验环境准备
  - Spark安装；Kafka安装；Python安装；Python依赖库；
- 数据处理和Python操作Kafka
  - 利用Python预处理数据； Python操作Kafka；
- Spark Streaming实时处理数据
  - Spark Streaming实时处理Kafka数据； 将处理后的结果发送给Kafka；
- 结果展示
  - 利用Flask-SocketIO实时推送数据；socket.io.js实时获取数据；highlights.js展示数据；

一、安装kafka(单机版)

1、下载解压kafka

```
sudo tar -zxf kafka_2.11-0.10.1.0.tgz -C /usr/local
cd /usr/local
sudo mv kafka_2.11-0.10.1.0 kafka
sudo chown -R root ./kafka
```

2、测试简单案例

（1）、启动自带的zookeeper（ 千万不要关闭当前终端 ）

```
# 进入kafka所在的目录
cd /usr/local/kafka
bin/zookeeper-server-start.sh config/zookeeper.properties
```

（2）、启动kafka（ 启动新的终端 、 kafka服务端启动了,请千万不要关闭当前终端 ）

```
cd /usr/local/kafka
bin/kafka-server-start.sh config/server.properties
```

```
[2020-10-03 06:36:53,293] INFO [Group Metadata Manager on Broker 0]: Removed 0 expired offsets i
n 11 milliseconds. (kafka.coordinator.GroupMetadataManager)
[2020-10-03 06:36:53,305] INFO Will not load MX4J, mx4j-tools.jar is not in the classpath (kafka
.utils.Mx4jLoader$)
[2020-10-03 06:36:53,326] INFO New leader is 0 (kafka.server.ZookeeperLeaderElector$LeaderChange
Listener)
[2020-10-03 06:36:53,360] INFO Creating /brokers/ids/0 (is it secure? false) (kafka.utils.ZKChec
kedEphemeral)
[2020-10-03 06:36:53,387] INFO Result of znode creation is: OK (kafka.utils.ZKCheckedEphemeral)
[2020-10-03 06:36:53,389] INFO Registered broker 0 at path /brokers/ids/0 with addresses: PLAINT
EXT -> EndPoint(master,9092,PLAINTEXT) (kafka.utils.ZkUtils)
[2020-10-03 06:36:53,390] WARN No meta.properties file under dir /tmp/kafka-logs/meta.properties
 (kafka.server.BrokerMetadataCheckpoint)
[2020-10-03 06:36:53,460] INFO Kafka version : 0.10.1.0 (org.apache.kafka.common.utils.AppInfoPa
rser)
[2020-10-03 06:36:53,460] INFO Kafka commitId : 3402a74efb23d1d4 (org.apache.kafka.common.utils.
AppInfoParser)
[2020-10-03 06:36:53,460] INFO [Kafka Server 0], started (kafka.server.KafkaServer)
```

jsp查看进程

```
[root@master kafka]# jps
1826 Kafka
1529 QuorumPeerMain
2093 Jps
[root@master kafka]#
```

（3）、创建主题（启动新的终端、以单节点的配置创建了一个叫dblab的topic）

```
cd /usr/local/kafka
bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --
partitions 1 --topic dblab
```

```
[root@master kafka]# bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-facto
r 1 --partitions 1 --topic dblab
Created topic "dblab".
```

（4）、用list列出所有创建的topics,来查看刚才创建的主题是否存在

```
bin/kafka-topics.sh --list --zookeeper localhost:2181
```

```
[root@master kafka]# bin/kafka-topics.sh --list --zookeeper localhost:2181
dblab
```

（5）、用producer生产点数据：

```
bin/kafka-console-producer.sh --broker-list localhost:9092 --topic dblab
```

并输入

hello hadoop
hello scala
hello world

```
[root@master kafka]# bin/kafka-console-producer.sh --broker-list localhost:9092 --topic dblab
hello hadoop
hello scala
hello world
```

（6）、使用consumer来接收数据,输入如下命令：

```
cd /usr/local/kafka
bin/kafka-console-consumer.sh --zookeeper localhost:2181 --topic dblab --from-
beginning
```

```
[root@master kafka]# bin/kafka-console-consumer.sh --zookeeper localhost:2181 --topic dblab --fr
om-beginning
Using the ConsoleConsumer with old consumer is deprecated and will be removed in a future major
release. Consider using the new consumer by passing [bootstrap-server] instead of [zookeeper].
hello hadoop
hello scala
hello world
```

便可以看到刚才产生的三条信息。说明kafka安装成功。

二、安装python

1、安装依赖

```
yum install -y openssl-devel openssl-static zlib-devel lzma tk-devel xz-devel
bzip2-devel ncurses-devel gdbm-devel readline-devel sqlite-devel gcc libffi-
devel
```

2、下载python

```
wget https://www.python.org/ftp/python/3.6.0/Python-3.6.0.tgz
tar -xvf Python-3.6.0.tgz -C /usr/local
```

3、执行配置文件

```
./configure
```

4、编译安装

```
make
make install
```

5、创建软链接

```
ln -s /usr/local/Python-3.6.0/python /usr/bin/python3
```

6、验证安装

```
python3 -V
```

```
[root@master Python-3.6.0]# python3 -V
Python 3.6.0
```

三、数据处理和Python操作Kafka

数据集介绍

本案例采用的数据集压缩包为data_format.zip，该数据集压缩包是淘宝2015年双11前6个月(包含双11)的交易数据(交易数据有偏移，但是不影响实验的结果)，里面包含3个文件，分别是用户行为日志文件user_log.csv 、回头客训练集train.csv 、回头客测试集test.csv. 在这个案例中只是用user_log.csv这个文件，下面列出文件user_log.csv的数据格式定义：

1. user_id | 买家id
2. item_id | 商品id

3. cat_id | 商品类别id
4. merchant_id | 卖家id
5. brand_id | 品牌id
6. month | 交易时间:月
7. day | 交易事件:日
8. action | 行为,取值范围{0,1,2,3},0表示点击，1表示加入购物车，2表示购买，3表示关注商品
9. age_range | 买家年龄分段：1表示年龄<18,2表示年龄在[18,24]，3表示年龄在[25,29]，4表示年龄在[30,34]，5表示年龄在[35,39]，6表示年龄在[40,49]，7和8表示年龄>=50,0和NULL则表示未知
10. gender | 性别:0表示女性，1表示男性，2和NULL表示未知
11. province| 收获地址省份

1、 安装Python操作Kafka的代码库

```
pip3 install kafka-python
```

2、编写producer.py文件

```
# coding: utf-8
import csv
import time
from kafka import KafkaProducer

# 实例化一个KafkaProducer示例，用于向Kafka投递消息
producer = KafkaProducer(bootstrap_servers='localhost:9092')
# 打开数据文件
csvfile = open("/root/user_log.csv","r")
# 生成一个可用于读取csv文件的reader
reader = csv.reader(csvfile)

for line in reader:
    gender = line[9] # 性别在每行日志代码的第9个元素
    if gender == 'gender':
        continue # 去除第一行表头
    time.sleep(0.1) # 每隔0.1秒发送一行数据
    # 发送数据，topic为'sex'
    producer.send('sex',line[9].encode('utf8'))
```

3、编写consumer.py

```
from kafka import KafkaConsumer

consumer = KafkaConsumer('sex')
for msg in consumer:
    print((msg.value).decode('utf8'))
```

4、开启kafka（开两个终端，不要关闭）

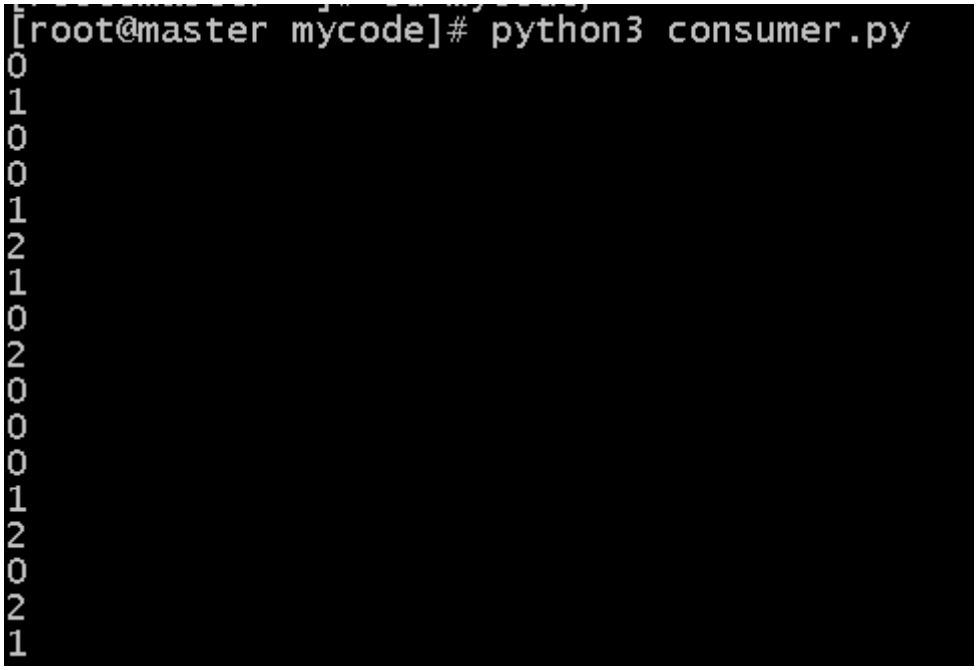
```
cd /usr/local/kafka
bin/zookeeper-server-start.sh config/zookeeper.properties &
bin/kafka-server-start.sh config/server.properties
```

5、运行producer.py

```
cd mycode
python3 producer.py
```

6、运行consumer.py

```
python3 consumer.py
```



四、Spark Streaming实时处理数据

1、下载相关jar包

Kafka和Flume等高级输入源，需要依赖独立的库（jar文件）

spark-streaming-kafka-0-10_2.11-2.4.5.jar文件的下载，其中，2.11表示scala的版本，2.4.5表示Spark版本号

```
mkdir /usr/local/spark/jars/kafka
cp ./spark-streaming-kafka-0-10_2.11-2.4.5.jar /usr/local/spark/jars/kafka
cd /usr/local/kafka/libs
cp ./* /usr/local/spark/spark-2.3.0-bin-hadoop2.7/jars/kafka
```

2、建立Spark项目

在/usr/local/spark/mycode新建项目主目录kafka,然后在kafka目录下新建scala文件存放目录以及scala工程文件

```
mkdir -p /usr/local/spark/mycode/kafka/src/main/scala
```

接着在src/main/scala文件下创建两个文件，一个是用于设置日志，一个是项目工程主文件，设置日志文件为StreamingExamples.scala

```
package org.apache.spark.examples.streaming
import org.apache.spark.internal.Logging
```

```
import org.apache.log4j.{Level, Logger}
/** Utility functions for Spark Streaming examples. */
object StreamingExamples extends Logging {
  /** Set reasonable logging levels for streaming if the user has not configured
log4j. */
  def setStreamingLogLevels() {
    val log4jInitialized = Logger.getRootLogger.getAllAppenders.hasMoreElements
    if (!log4jInitialized) {
      // We first log something to initialize Spark's default logging, then we
override the
      // logging level.
      logInfo("Setting log level to [WARN] for streaming example." +
        " To override add a custom log4j.properties to the classpath.")
      Logger.getRootLogger.setLevel(Level.WARN)
    }
  }
}
```

工程主文件，文件名为KafkaTest.scala

```
package org.apache.spark.examples.streaming

import java.util.HashMap
import org.apache.kafka.clients.producer.{KafkaProducer, ProducerConfig,
ProducerRecord}
import org.apache.kafka.clients.consumer.ConsumerConfig
import org.apache.kafka.common.serialization.StringDeserializer
import org.json4s._
import org.json4s.jackson.Serialization
import org.json4s.jackson.Serialization.write
import org.apache.spark.SparkConf
import org.apache.spark.streaming._
import org.apache.spark.streaming.Interval
import org.apache.spark.streaming.kafka010._

object KafkaWordCount {
  implicit val formats = DefaultFormats//数据格式化时需要
  def main(args: Array[String]): Unit={
    if (args.length < 3) {
      System.err.println("Usage: KafkaWordCount <brokers> <groupId> <topics>")
      System.exit(1)
    }
    StreamingExamples.setStreamingLogLevels()

    val Array(brokers, groupId, topics) = args
    val sparkConf = new SparkConf().setAppName("KafkaWordCount")
    val ssc = new StreamingContext(sparkConf, Seconds(1))
    ssc.checkpoint("checkpoint")

    val topicsSet = topics.split(",").toSet
    val kafkaParams = Map[String, Object](
      ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG -> brokers,
      ConsumerConfig.GROUP_ID_CONFIG -> groupId,
      ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG ->
classOf[StringDeserializer],
      ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG ->
classOf[StringDeserializer])
```

```
    val messages = KafkaUtils.createDirectStream[String, String](
      ssc,
      LocationStrategies.PreferConsistent,
      ConsumerStrategies.Subscribe[String, String](topicsSet, kafkaParams))

    // Get the lines, split them into words, count the words and print
    val lines = messages.map(_.value)
    val words = lines.flatMap(_.split(" "))//将输入的每行用空格分割成一个个word


    // 对每一秒的输入数据进行reduce，然后将reduce后的数据发送给Kafka
    val wordCounts = words.map(x => (x, 1L))
        .reduceByKeyAndWindow(_+_,_-_, Seconds(1), Seconds(1), 1).foreachRDD(rdd
=> {
          if(rdd.count !=0 ){
               val props = new HashMap[String, Object]()
               props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,
"127.0.0.1:9092")
               props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
               "org.apache.kafka.common.serialization.StringSerializer")
               props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
               "org.apache.kafka.common.serialization.StringSerializer")
               // 实例化一个Kafka生产者
               val producer = new KafkaProducer[String, String](props)
               // rdd.colect即将rdd中数据转化为数组，然后write函数将rdd内容转化为json格
式
               val str = write(rdd.collect)
               // 封装成Kafka消息，topic为"result"
               val message = new ProducerRecord[String, String]("result", null,
str)
               // 给Kafka发送消息
               producer.send(message)
          }
      })
    ssc.start()
    ssc.awaitTermination()
  }
}
```

对代码简要说明：

1. 首先按每秒的频率读取Kafka消息；
2. 然后对每秒的数据执行wordcount算法，统计出0的个数，1的个数，2的个数；
3. 最后将上述结果封装成json发送给Kafka。

3、运行项目

编写好程序之后，下面介绍下如何打包运行程序。在/usr/local/spark/mycode/kafka目录下新建文件simple.sbt，输入如下内容：

```
name := "Simple Project"
version := "1.0"
scalaVersion := "2.11.8"
libraryDependencies += "org.apache.spark" %% "spark-core" % "2.4.5"
libraryDependencies += "org.apache.spark" % "spark-streaming_2.11" % "2.4.5"
libraryDependencies += "org.apache.spark" % "spark-streaming-kafka-0-10_2.11" %
"2.4.5"
libraryDependencies += "org.json4s" %% "json4s-jackson" % "3.2.11"
```

编译打包程序，输入如下命令

```
/usr/local/sbt/bin/sbt package
```

打包成功之后，接下来编写运行脚本，在/usr/local/spark/mycode/kafka目录下新建startup.sh文件，输入如下内容：

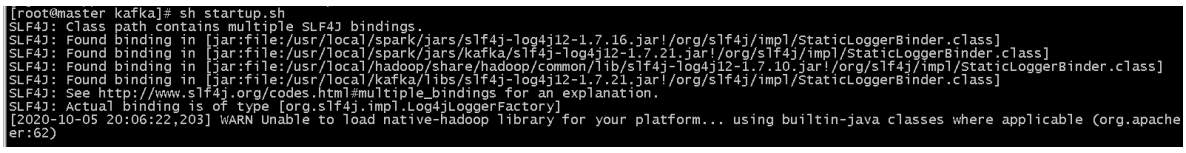
```
 /usr/local/spark/bin/spark-submit --driver-class-path
/usr/local/spark/jars/*:/usr/local/spark/jars/kafka/* --class
"org.apache.spark.examples.streaming.KafkaWordCount"
/usr/local/spark/mycode/kafka/target/scala-2.11/simple-project_2.11-1.0.jar
127.0.0.1:9092 1 sex
```

其中最后三个为输入参数，含义如下

1. 127.0.0.1:9092为brokerer地址
2. 1 为consumer group标签
3. sex为消费者接收的topic

最后在/usr/local/spark/mycode/kafka目录下，运行如下命令即可执行刚编写好的Spark Streaming程序

```
 sh startup.sh
```



程序运行成功之后，下面通过之前的KafkaProducer和KafkaConsumer来检测程序 。

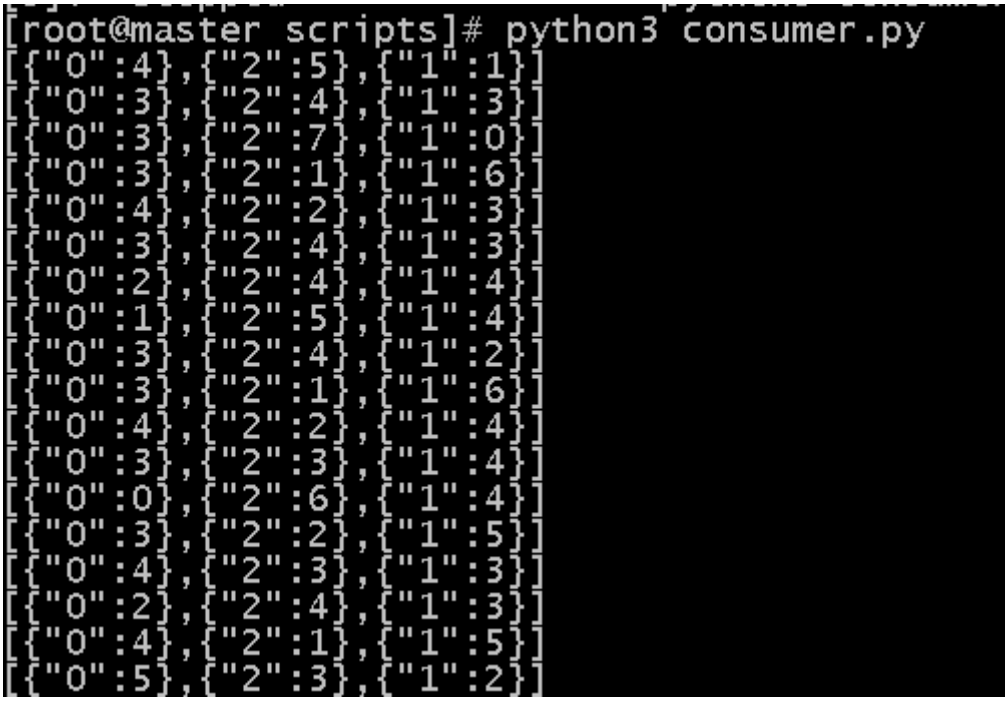
4、测试程序

下面开启之前编写的KafkaProducer投递消息，然后将KafkaConsumer中接收的topic改为result，验证是否能接收topic为result的消息，更改之后的KafkaConsumer为

```
from kafka import KafkaConsumer

consumer = KafkaConsumer('result')
for msg in consumer:
    print((msg.value).decode('utf8'))
```

在同时开启Spark Streaming项目，KafkaProducer以及KafkaConsumer之后，可以在KafkaConsumer运行窗口，出现以下类似数据：

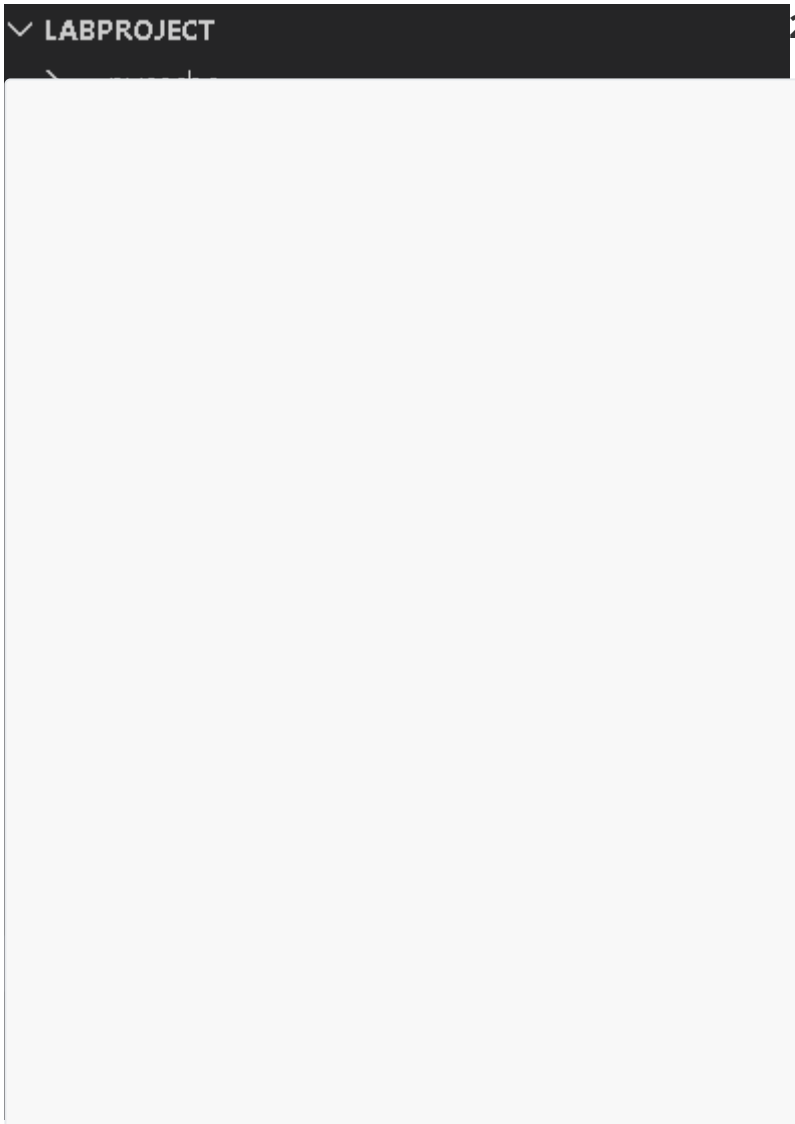


五、结果展示

使用如下Shell命令完成Flask和Flask-SocketIO这两个Python第三方库的安装

```
pip3 install flask
pip3 install flask-socketio
```

1、目录结构



2、app.py

```
import json
from flask import Flask,
render_template
from flask_socketio
import SocketIO
from kafka import
KafkaConsumer


app = Flask(__name__)
app.config['SECRET_KEY']
= 'secret!'
socketio = SocketIO(app)
thread = None
consumer =
KafkaConsumer('result')


def background_thread():
    girl = 0
    boy = 0
    for msg in consumer:
        data_json =
msg.value.decode('utf8')
        data_list =
json.loads(data_json)
        for data in
data_list:
            if '0' in
data.keys():
                girl =
data['0']
            elif '1' in
data.keys():
                boy =
data['1']
            else:
                continue
        result =
str(girl) + ',' +
str(boy)
        print(result)

socketio.emit('test_messa
ge',{'data':result})


@socketio.on('test_connec
t')
def connect(message):
    print(message)
    global thread
    if thread is None:
```

```
                thread =
socketio.start_background
_task(target=background_t
hread)

socketio.emit('connected'
, {'data': 'Connected'})


@app.route("/")
def handle_mes():
    return
render_template("index.ht
ml")


if __name__ ==
'__main__':

socketio.run(app,host='0.
0.0.0',debug=True)
```

3、index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>DashBoard</title>
    <script src="static/js/socket.io.js"></script>
    <script src="static/js/jquery-3.1.1.min.js"></script>
    <script src="static/js/highcharts.js"></script>
    <script src="static/js/exporting.js"></script>
    <script type="text/javascript" charset="utf-8">
    var socket = io.connect('http://' + document.domain + ':' + location.port);
    socket.on('connect', function() {
        socket.emit('test_connect', {data: 'I\'m connected!'});
    });

    socket.on('test_message',function(message){
        console.log(message);
        var obj = eval(message);
        var result = obj["data"].split(",");
        $('#girl').html(result[0]);
        $('#boy').html(result[1]);
    });

    socket.on('connected',function(){
        console.log('connected');
    });

    socket.on('disconnect', function () {
        console.log('disconnect');
    });
    </script>
```

```
</head>
<body>
<div>
    <b>Girl: </b><b id="girl"></b>
    <b>Boy: </b><b id="boy"></b>
</div>
<div id="container" style="width: 600px;height:400px;"></div>

<script type="text/javascript">
    $(document).ready(function () {
    Highcharts.setOptions({
        global: {
            useUTC: false
        }
    });

    Highcharts.chart('container', {
        chart: {
            type: 'spline',
            animation: Highcharts.svg, // don't animate in old IE
            marginRight: 10,
            events: {
                load: function () {

                    // set up the updating of the chart each second
                    var series1 = this.series[0];
                    var series2 = this.series[1];
                    setInterval(function () {
                        var x = (new Date()).getTime(), // current time
                        count1 = $('#girl').text();
                        y = parseInt(count1);
                        series1.addPoint([x, y], true, true);

                        count2 = $('#boy').text();
                        z = parseInt(count2);
                        series2.addPoint([x, z], true, true);
                    }, 1000);
                }
            }
        },
        title: {
            text: '男女生购物人数实时分析'
        },
        xAxis: {
            type: 'datetime',
            tickPixelInterval: 50
        },
        yAxis: {
            title: {
                text: '数量'
            },
            plotLines: [{
                value: 0,
                width: 1,
                color: '#808080'
            }]
        },
        tooltip: {
```

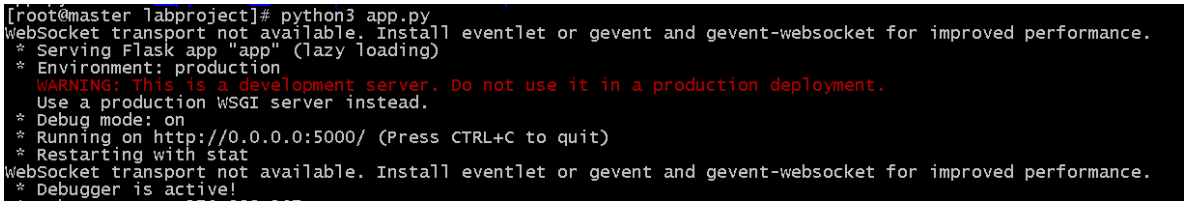
```
            formatter: function () {
                return '<b>' + this.series.name + '</b><br/>' +
                    Highcharts.dateFormat('%Y-%m-%d %H:%M:%S', this.x) + '<br/>'
+
                    Highcharts.numberFormat(this.y, 2);
            }
        },
        legend: {
            enabled: true
        },
        exporting: {
            enabled: true
        },
        series: [{
            name: '女生购物人数',
            data: (function () {
                // generate an array of random data
                var data = [],
                    time = (new Date()).getTime(),
                    i;

                for (i = -19; i <= 0; i += 1) {
                    data.push({
                        x: time + i * 1000,
                        y: Math.random()
                    });
                }
                return data;
            }())
        },
        {
            name: '男生购物人数',
            data: (function () {
                // generate an array of random data
                var data = [],
                    time = (new Date()).getTime(),
                    i;

                for (i = -19; i <= 0; i += 1) {
                    data.push({
                        x: time + i * 1000,
                        y: Math.random()
                    });
                }
                return data;
            }())
        }]
    });
});
</script>
</body>
</html>
```

4、运行app.py

```
python3 app.py
```



5、在电脑的浏览器上访问，查看结果

1. 配置flask项目

```
app.run(host='0.0.0.0',port=5000)
```

2. 打开防火墙5000端口：
   开启防火墙

```
service firewalld start
```

   (查看防火墙状态 systemctl status firewalld ， 开启防火墙 systemctl start firewalld)

3. 添加指定需要开放的端口：

```
firewall-cmd --add-port=5000/tcp --permanent
```

   (移除指定端口：firewall-cmd --permanent --remove-port=123/tcp)

   (移除指定端口：firewall-cmd --permanent --remove-port=123/tcp)

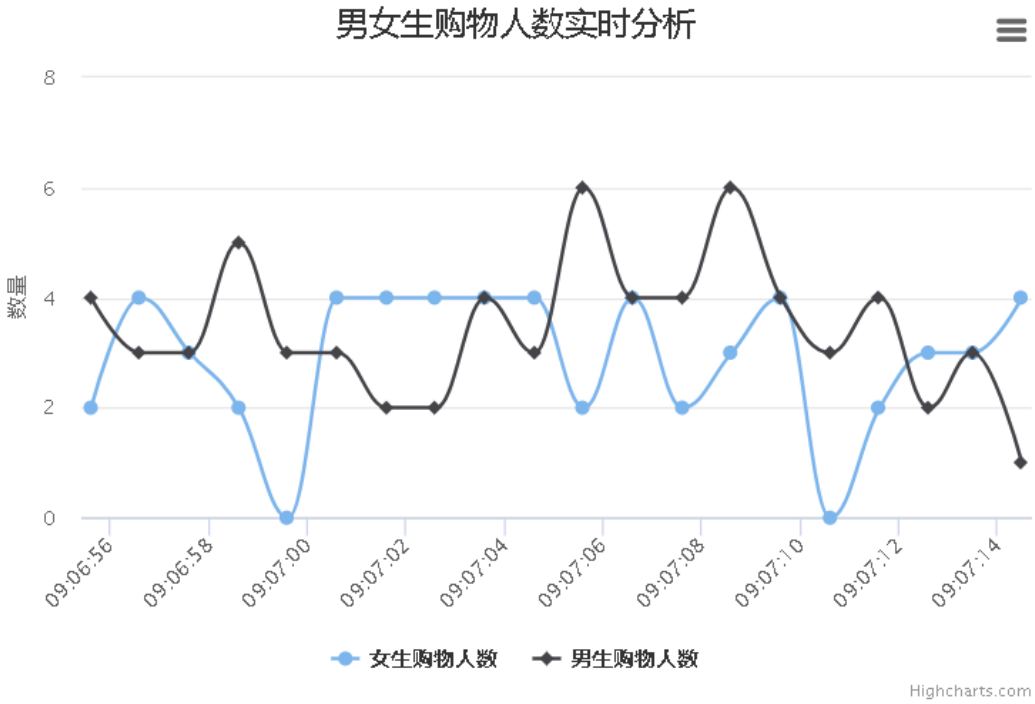
4. 重载入添加的端口：

```
firewall-cmd --reload
```

在电脑的浏览器访问192.168.100.10：5000就可以看到

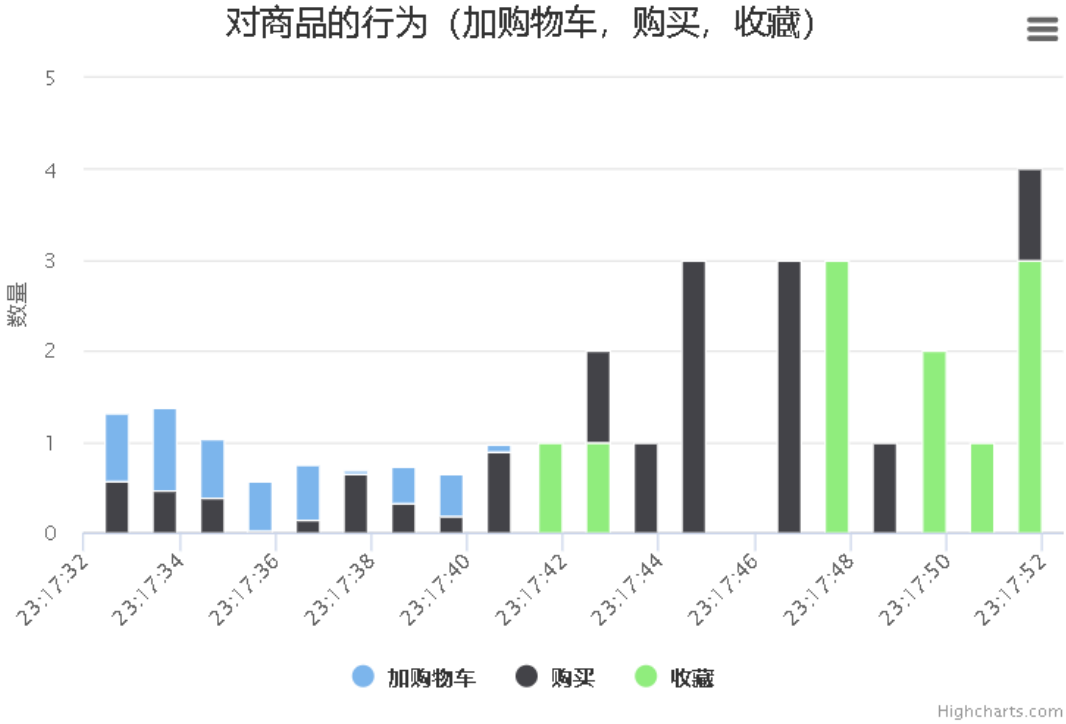
```
type: 'spline'  //条形图
```

Girl: 3 Boy: 4




```
type: 'column', //柱形图
//柱形图叠加
plotOptions: {
    column: {
        stacking: 'normal'
    }
},
```

加购物车: 0 购买: 1 收藏: 1



小于18：1 18-24：0 24-29：3 29-34：0 34-39：0 39-49：0 大于50：0