总结、分享、收获

实验室主页 (//dblab.xmu.edu.cn)

首页 (http://dblab.xmu.edu.cn/blog/)

大数据 (https://dblab.xmu.edu.cn/blog/category/big-data/)

数据库 (https://dblab.xmu.edu.cn/blog/category/databases/)

数据挖掘 (https://dblab.xmu.edu.cn/blog/category/data-mining/)

其他 (https://dblab.xmu.edu.cn/blog/category/others/)

搜索...

搜索

Spark+Kafka构建实时分析Dashboard案例（2022年9月V2.0版）——步骤四：结果展示

刘, 浩然 2022年9月13日 (updated: 2022年9月22日) 14418



(//dblab.xmu.edu.cn/post/bigdataroadmap/)

返回本案例首页 (http://dblab.xmu.edu.cn/post/spark-kafka-dashboard/)
查看前一步骤操作：步骤三：Structured Streaming实时处理数据（scala版本）(https://dblab.xmu.edu.cn/blog/2022090001-5/)
查看前一步骤操作：步骤三：Structured Streaming实时处理数据（python版本）(https://dblab.xmu.edu.cn/blog/2022090001-7/)
《Spark+Kafka构建实时分析Dashboard案例——步骤四：结果展示》

开发团队：厦门大学数据库实验室 联系人：林子雨老师ziyulin@xmu.edu.cn

版权声明：版权归厦门大学数据库实验室所有，请勿用于商业用途；未经授权，其他网站请勿转载

本教程介绍大数据课程实验案例“Spark+Kafka构建实时分析Dashboard”的第四个步骤，结果展示。在本篇博客中，将介绍如何利用Flask-SocketIO向客户端发送消息以及客户端如何利用highcharts.js展示数据。

所需知识储备

了解Flask创建web程序，了解如何在html编写js代码

训练技能

利用Flask创建web程序，利用Flask-SocketIO实现实时推送数据，利用socket.io.js实现实时接收数据，hightlights.js展现数据

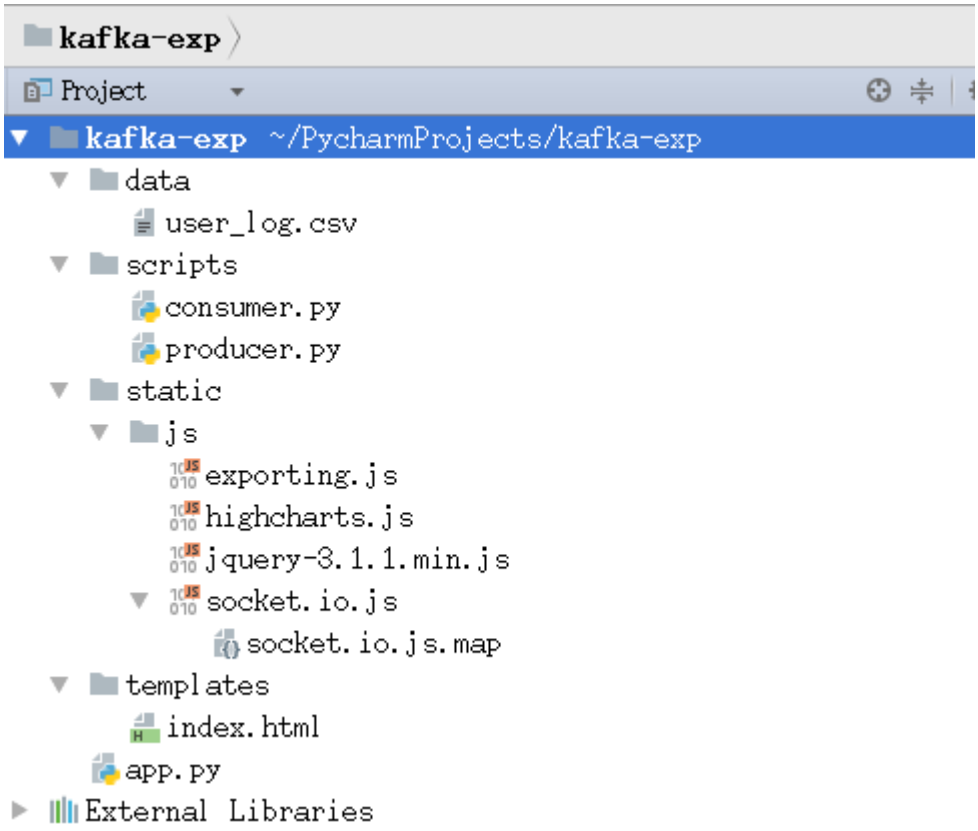
任务清单

1. 利用Flask-SocketIO实时推送数据
2. socket.io.js实时获取数据
3. highlights.js展示数据

Flask-SocketIO实时推送数据

上篇文章说道Structured Streaming实时接收Kafka中topic为'sex'发送的日志数据，然后Structured Streaming进行实时处理，统计好每秒中男女生购物人数之后，将结果发送至Kafka，topic为'result'。在本章节，将介绍如何利用Flask-SocketIO将结果实时推送到浏览器。

下面讲述的内容需要使用Flask-SocketIO，相关文档可以查看Flask-SocketIO文档 (https://flask-socketio.readthedocs.io/en/latest/)。
这里我们为了方便查看,再一次贴出项目工程结构图。



备注：为了方便大家完成上述实验，这里提供源代码文件的下载，请点击这里从百度云盘下载(https://pan.baidu.com/s/1Vg_Zvk5LsbywbHx_L9XrTA?pwd=ziyu)。大家下载完源代码以后，可以导入到自己的IntelliJIDEA开发工具中，查看源代码。备注：点击这里查看IntelliJIDEA工具的安装方法 (https://dblab.xmu.edu.cn/blog/1327/)。

首先我们创建如图中的app.py文件,app.py的功能就是作为一个简易的服务器，处理连接请求，以及处理从kafka接收的数据，并实时推送到浏览器。app.py的代码如下:

Python

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

# 因为第一步骤安装好了flask，所以这里可以引用

app = Flask(__name__)
app.config['SECRET_KEY'] = 'secret!'
socketio = SocketIO(app)
thread = None
# 实例化一个consumer，接收topic为result的消息
consumer = KafkaConsumer('result')


# 一个后台线程，持续接收Kafka消息，并发送给客户端浏览器
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_message', {'data': result})


# 客户端发送connect事件时的处理函数
@socketio.on('test_connect')
def connect(message):
    print(message)
    global thread
    if thread is None:
        # 单独开启一个线程给客户端发送数据
        thread = socketio.start_background_task(target=background_thr
ead)
    socketio.emit('connected', {'data': 'Connected'})


# 通过访问http://127.0.0.1:5000/访问index.html
@app.route("/")
def handle_mes():
    return render_template("index.html")
```

```
50.
51.
52. # main函数
53. if __name__ == '__main__':
54.     socketio.run(app, debug=True)
```

这段代码实现比较简单，最重要就是background_thread函数，该函数从Kafka接收消息，并进行处理，获得男女生每秒钟人数，然后将结果通过函数socketio.emit实时推送至浏览器。

浏览器获取数据并展示

index.html文件负责获取数据并展示效果，该文件中的代码内容如下：

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 + ':' + locat
ion.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 ti
    me
                        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>
```

可以看到，在上面给出的index.html文件的开头部分，包含如下几行代码：

HTML

```
<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>
```

也就是说，在index.html中，需要用到几个js库，包括socket.io.js、jquery-3.1.1.min.js、highcharts.js和exporting.js。

下面分别介绍这几个js库文件的用途和下载方法（备注：如果上面已经下载了我们数据库实验室提供的源码文件，则里面已经包含了这些js库文件，不需要再去网络上下载）。

socket.io.js

客户端浏览器需要使用js框架socket.io.js来实时接收服务端的消息，该js框架用法和Flask-SocketIO使用类似。首先我们可以看到在工程目录中有两个js库文件跟socket.io.js相关，一个就是socket.io.js (https://cdnjs.cloudflare.com/ajax/libs/socket.io/4.5.1/socket.io.js)，下载方式很简单，点击给出的链接，然后在打开的页面右键--->另存为..就可以下载该文件;还有一个文件就是socket.io.js.map (https://cdnjs.cloudflare.com/ajax/libs/socket.io/4.5.1/socket.io.js.map)，点击链接就可以下载。（备注：如果上面已经下载了我们数据库实验室提供的源码文件，则里面已经包含了这些js库文件，不需要再去网络上下载）
下载好socket.io.js和socket.io.js.map这两个js库文件后，按照上面给出的工程文件目录结构，把这两个文件复制到js文件夹下（这两个文件之间没有嵌套关系，不要被我们上面给出的工程结构图迷惑）。

然后，再来看一下，在index.html中是如何调用socket.io.js和socket.io.js.map这两个js库文件，可以看到，在index.html中包含了如下一段代码，就是用来调用这两个库文件的：

JavaScript

```
<script type="text/javascript" charset="utf-8">
    // 创建连接服务器的链接
    var socket = io.connect('http://' + document.domain + ':' + location.port);
    socket.on('connect', function() {// 连上服务器后的回调函数
        socket.emit('connect', {data: 'I\'m connected!'});
    });
    // 接收服务器实时发送的数据
    socket.on('test_message',function(message){
        console.log(message);
        var obj = eval(message);
        var result = obj["data"].split(",");
        // 将男生和女生人数展示在html标签内
        $('#girl').html(result[0]);
        $('#boy').html(result[1]);
    });

    socket.on('connected',function(){
        console.log('connected');
    });
    // 链接断开时的回调函数
    socket.on('disconnect', function () {
        console.log('disconnect');
    });
    </script>
```

上面这段代码就是使用socket.io.js库来实时地接收服务端发送过来的消息，并将消息数据实时地设置在html标签内，交给highcharts.js进行实时获取和展示。

highchart.js

highcharts.js是一个用纯JavaScript编写的一个图表库。能够很简单便捷地在Web网站或是We
用程序中添加有交互性的图表。可以到官网下载最新版本的highchart.js库文件(点击这里下载(https://www.highcharts.com/download))。注意，到官网下载highchart.js库文件时，下载到本地的是一个类似Highcharts-6.0.7.zip这样的压缩文件，对这个文件进行解压缩，可以看到里面有个code子目录，在code子目录下面就可以找到highchart.js库文件，按照上面给出的工程文件目录结构，把这个highchart.js库文件复制到IntelliJIDEA工程目录中的js文件夹下。（备注：如果上面已经下载了我们数据库实验室提供的源码文件，则里面已经包含了这些js库文件，不需要再去网络上下载）

在index.html中包含如下一段代码，就是调用highcharts.js库，来实时地从html标签内获取数据并展示在网页中。

JavaScript

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

    Highcharts.chart('container', {
        chart: {
            type: 'spline',
            animation: Highcharts.svg, // 这个在ie浏览器可能不支持
            marginRight: 10,
            events: {
                load: function () {

                    //设置图表每秒更新一次
                    var series1 = this.series[0];
                    var series2 = this.series[1];
                    setInterval(function () {
                        var x = (new Date()).getTime();// 获取当前时间
                        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: { //设置图表名
            text: '男女生购物人数实时分析'
        },
        xAxis: { //x轴设置为实时时间
            type: 'datetime',
            tickPixelInterval: 50
        },
        yAxis: {
            title: {
                text: '数量'
            },
            plotLines: [{ //设置坐标线颜色粗细
                value: 0,
                width: 1,
                color: '#808080'
            }]
        },
        tooltip: {
```

```
            //规范显示时间的格式
            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 () {
                // 随机方式生成初始值填充图表
                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 () {
                // 随机方式生成初始值填充图表
                var data = [],
                    time = (new Date()).getTime(),
                    i;

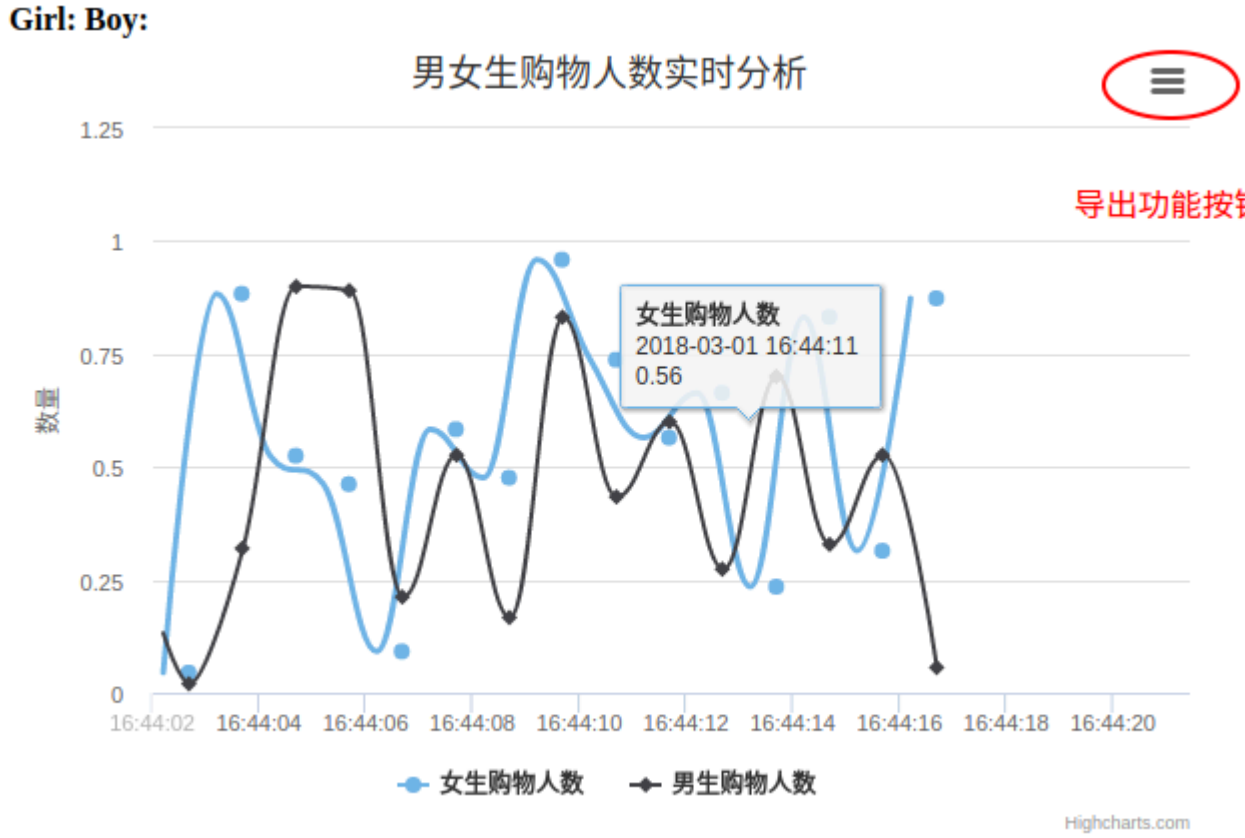
                for (i = -19; i <= 0; i += 1) {
                    data.push({
                        x: time + i * 1000,
                        y: Math.random()
                    });
                }
                return data;
            }())
        }]
    });
});
```

这段代码其实很简单，就是在设置表格的格式，因为highchart.js这个框架，让我们在网页中显示数据图表的工作变得很轻松。

exporting.js

在上面，到官网下载highchart.js库文件时，下载到本地的是一个类似Highcharts-6.0.7.zip这样的压缩文件，对这个文件进行解压缩，可以看到里面有个code子目录，在code子目录下面就可以找到一个js子目录，在js子目录下可以找到一个modules子目录，在modules子目录中就可以找到库文件exporting.js。然后，按照上面给出的工程文件目录结构，把这个exporting.js库文件复制到IntelliJIDEA工程目录中的js文件夹下。（备注：如果上面已经下载了我们数据库实验室提供的源码文件，则里面已经包含了这些js库文件，不需要再去网络上下载）

exporting.js这个库文件的功能是实现导出功能。也就是下图中的红色圈内的按钮的功能。



jquery.js

除了上述三个js库文件外，还有一个jquery.js库文件。jQuery是一个快速、简洁的JavaScript框架，它封装JavaScript常用的功能代码，提供一种简便的JavaScript设计模式，优化HTML文档操作、事件处理、动画设计和Ajax交互。点击这里下载jquery.js (http://jquery.com/download/)，然后，按照上面给出的工程文件目录结构，把这个jquery.js库文件复制到IntelliJIDEA工程目录中的js文件夹下。（备注：如果上面已经下载了我们数据库实验室提供的源码文件，则里面已经包含了这些js库文件，不需要再去网络上下载）

效果展示

经过以上步骤，一切准备就绪，我们就可以启动程序来看看最后的效果。启动步骤如下：

1.确保kafka开启，参考教程步骤二 (https://dblab.xmu.edu.cn/blog/2022090001-4/)。
2.开启producer.py模拟数据流，参考步骤二 (https://dblab.xmu.edu.cn/blog/2022090001-4/)。
3.启动Structured Streaming实时处理数据，参考步骤三 (https://dblab.xmu.edu.cn/blog/2022090001-5/)。提示你可以在实时处理数据启动之后，把comsumer.py的topic改成result，运行comsumer.py就可以看到数据处理后的输出结果。
4.启动app.py。如果你是使用pycharm客户端，那右键就可以了。当然也可以使用终端命令：

Shell

```
1. python app.py
```

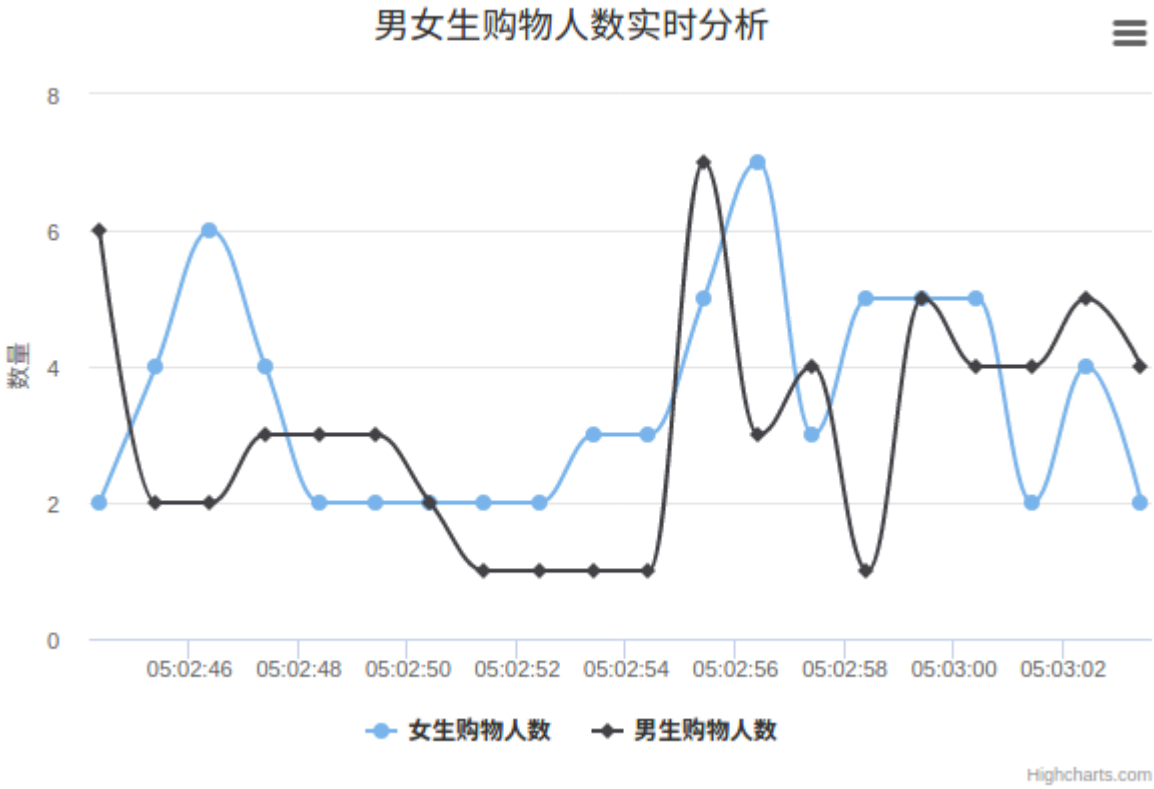
启动后的效果如下图：

```
way@way1:~/PycharmProjects/kafka-exp$ python app.py
WebSocket transport not available. Install eventlet or gevent and gevent-websock
et for improved performance.
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 * Restarting with stat
WebSocket transport not available. Install eventlet or gevent and gevent-websock
et for improved performance.
 * Debugger is active!
 * Debugger PIN: 292-510-168
127.0.0.1 - - [02/Mar/2018 16:23:06] "GET /socket.io/?EIO=3&transport=polling&t=
M7bphq9 HTTP/1.1" 200 -
{u'data': u"I'm connected!"}
127.0.0.1 - - [02/Mar/2018 16:23:06] "POST /socket.io/?EIO=3&transport=polling&t
=M7bphqK&sid=fa14b6c1ff9a44ebb9a3fa9282921497 HTTP/1.1" 200 -
127.0.0.1 - - [02/Mar/2018 16:23:06] "GET /socket.io/?EIO=3&transport=polling&t
M7bphqL&sid=fa14b6c1ff9a44ebb9a3fa9282921497 HTTP/1.1" 200 -
3,2
127.0.0.1 - - [02/Mar/2018 16:23:07] "GET /socket.io/?EIO=3&transport=polling&t
M7bphqW&sid=fa14b6c1ff9a44ebb9a3fa9282921497 HTTP/1.1" 200 -
```

这时候你可以用浏览器访问上图中给出的网址 http://127.0.0.1:5000/ ，就可以看到最终效果图了。

下面截图展示了本案例的最终效果，本案例最终效果是一个动态效果，这里截取两张截图为例：

Girl: 2 Boy: 5



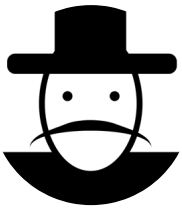
Girl: 3 Boy: 5



到此为止，本案例就顺利完成了！

为了方便大家完成上述实验，这里提供源代码文件的下载，请点击这里从百度云盘下载(https://pan.baidu.com/s/1Vg_Zvk5LsbywbHx_L9XrTA?pwd=ziyu)。

本文作者



刘, 浩然 (//dblab.xmu.edu.cn/blog/author/liuhaoran/)

1609126475@qq.com

https://dblab.xmu.edu.cn/blog/3228/ (https://dblab.xmu.edu.cn/blog/3228/)

其他 (https://dblab.xmu.edu.cn/blog/category/others/)

© 2014 厦大数据库实验室 (//dblab.xmu.edu.cn)