实时检测共享账户详细部署说明书

目录

[1. 引言 3](#_Toc529832611)

[1.1背景 3](#_Toc529832612)

[1.2目的 3](#_Toc529832613)

[1.3适用范围 3](#_Toc529832614)

[2. 总体设计 4](#_Toc529832615)

[2.1需求概要 4](#_Toc529832616)

[2.2集群设计 4](#_Toc529832617)

[2.3程序结构 5](#_Toc529832618)

[3. 详细设计 9](#_Toc529832619)

[3.1数据源 9](#_Toc529832620)

[3.2数据传输 10](#_Toc529832621)

[3.3数据处理 11](#_Toc529832622)

[3.4数据展示 19](#_Toc529832624)

[3.42链接数据库的实现 20](#_Toc529832625)

[3.42 DAO层代码的编写 21](#_Toc529832626)

[3.43 Service层代码的编写 22](#_Toc529832627)

# 引言

## 1.1背景

21世纪是一个数据爆炸的时代，无论是在数量上还是在速度上。尤其是近年来，我们集体步入5G时代数据的传输速度越来越快，数据量也越来越大，从视频额高清到720P，到蓝光数据量成倍的增长。云盘的出现，方便了我们的生活，我们将一些重要的数据存入其中做备份，必要的时候再从一年盘中下载，不再需要随身携带大容量的存储设备。

但是，无论是上传，下载都与我们的流量息息相关。国内的运营商为了保护我们的账号的安全，以及我们的用户体验，提供了一种账号异常检测的机制。从每天产生的实时数据流中截取一部分的数据，对这些原始数据进行处理判断，检测我们的账号是否存在异常，维护我们的账号安全，避免多个账号使用一个终端，争抢流量资源，影响网络速度，提供更好更优质的服务。

## 1.2目的

通过对数据的检测，保证账号的安全以及处理各种账号的问题，避免账号出现异常影响用户的使用和体验

## 1.3适用范围

运营商，如移动，联通，电信等

# 总体设计

## 2.1需求概要

* 生成原始数据功能
* 收集原始数据功能
* 存储原始数据功能
* 处理原始数据，筛选异常数据功能（3种异常状况）
  + natUP去重求和数 >5
  + qqid去重求和数 >20
  + cookieValue + devName + osName去重求和数 >5
* 存储数据功能
* 前端展示数据功能

## 2.2集群设计

一个主节点，2个从节点

## 2.3程序结构

**2.31总体架构**

**Java代码创建数据**

**数据来源层**

**Flume获取数据**

**数据获取层**

**Kafka存储数据**

**数据存储层**

**Storm处理数据**

**数据处理层**

**JavaWeb+MySQL**

**数据展示层**

**2.32Flume架构**

该模块的详细设计，首先选用的是Flume在业务系统拉取数据到Kafka中，将处理完的数据存入MySQL。下面对该系统中数据流向进行介绍。图4-19 展示的是数据流向图。

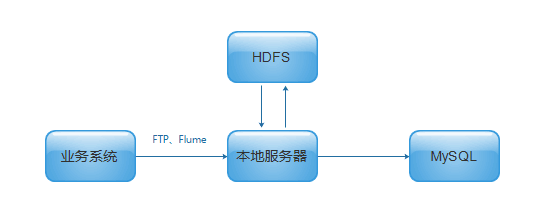


图4-19 数据流向图

紧接着，对Flume做一个具体的介绍。日志收集系统Flume 的核心就是把模拟的数据收集在一起，然后发送到目的地。Flume为了能保证传输成功，在把传送的数据传到到目的地之前，先把数据缓存起来，等到数据真正到达到目的地之后，立刻就会删除缓存的数据。Event是Flume 中数据的一个基本单位，如果说它是一个文本文件，那么通常情况下就是一行记录，就是事务中的一个基本单位。一个Event 事件就是一个数据流的最小单元，从外部的数据源而来，去向外部的目的地。

在这个日志收集系统Flume 中，它运行的核心是一个 Agent。这个系统对数据进行收集，其中包含有三个主要的核心组件，它们分别是 Source、Channel、Sink。通过这些组件，一个Event事件从Flume中的Source流向Channel，最后到Sink。

Source 组件可以接收从外部发送而来的数据。不同的Source组件之间，是可以接受不相同的数据格式。如果源数据是在目录池里，那么可以监控指定文件夹中的新文件的各种变化。如果发现目录中有新的文件产生，那么就会马上读取其它的内容。

Channel 是一个存储地也相当于缓冲池，用来接收Source的输出，一直到Sink消费掉缓冲池Channel中的数据。Channel当中的数据一直等到进入到下一个Channel缓冲池中或者进入到终端才会被删除。如果Sink组件写入失败之后，还可以自动重启，并且还不会造成数据的丢失，所以很可靠。

组件Sink会消费缓冲池Channel中的数据，接着发送给外部的源或者是其他Source。例如该系统最后把数据发送给Kafka。

部署于node1上，收集源数据，发给主节点

部署于主节点上，实际两个从节点发来的消息，并发送给kafka

部署于node2上，收集源数据，发给主节点

**2.33Storm的架构**

Spolt从Kafka中不断的出去数据

IPBolt2将相同账号但其他不同的计数

QQBolt2将相同账号但QQ不同的计数

IPBolt2将相同账号但IP不同的计数

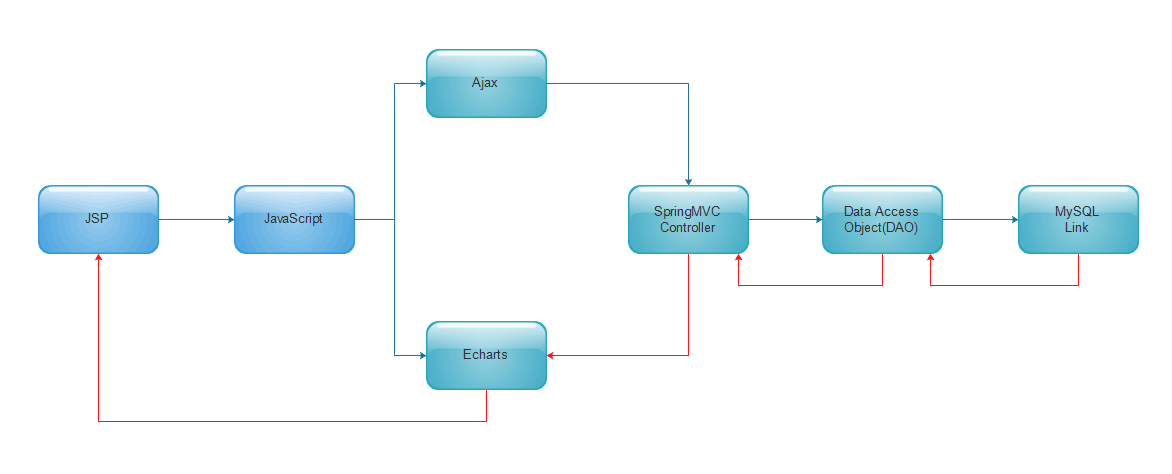
OSBolt将四个字段都相同的去重

QQBolt将账号和QQ都相同的去重

IPBolt将账号和IP都相同的去重

SaveBolt将5分钟内重复的账号去重后，存入数据库

**2.34数据可视化的架构**

在JSP中内嵌JavaScript，然后在JavaScript中写入Ajax，Ajax中的URL指定到SpringMVC的Controller中的XX.do，XX.do执行数据的获取-DAO(Data Access Object)，DAO通过MySQL的连接从MySQL中获取到数据。若以上步骤成功，Ajax则会获得一个JSON数据并传输到Echarts上。最终Echarts在JSP上显示。

JavaScript:一种[脚本语言](http://baike.baidu.com/view/76320.htm" \t "http://baike.baidu.com/_blank)，是一种动态型、弱型、基于原型的语言，内置支持类型。它的[解释器](http://baike.baidu.com/view/592974.htm)被叫做JavaScript引擎，是[浏览器](http://baike.baidu.com/view/7718.htm" \t "http://baike.baidu.com/_blank)的一部分，非常广泛用于[客户端](http://baike.baidu.com/view/930.htm)的脚本语言，用来给[HTML](http://baike.baidu.com/view/692.htm)网页增加动态功能，增强页面的友好度。js语言和他语言一样，有它自己的基本数据类型，表达式和算术运算符及程序的基本程序框架。js提供了四种基本的数据类型和两种特殊数据类型用来处理数据和文字。变量提供存放信息的地方，表达式则可以完成较复杂的信息处理

echart: 一个用javascript写的图表库，能非常流畅的运行在 PC 和移动设备上，兼容性非常好，底层依赖轻量级的 Canvas 类库 ZRender，提供直观，生动，交互性高，可高度个性化定制的数据可视化图表。

mvc框架: 英文名字是Model View Controller，是模型(model)－视图(view)－控制器(controller)的缩写，用一种业务逻辑、数据、界面 显示分离的方法组织代码，将业务逻辑聚集到一个部件里面，在改进和个性化定制界面及用户交互的同时，不需要重新编写业务逻辑。MVC用于映射传统的输入、处理和输出功能在一个逻辑的图形化用户界面的结构中。

Ajax：即“Asynchronous Javascript And XML”（异步JavaScript和XML），是指一种创建交互式网页应用的网页开发技术。AJAX是一种用于创建快速动态网页的技术。通过在后台与服务器进行少量数据交换，AJAX可以使网页实现异步更新。这意味着可以在不重新加载整个网页的情况下，对网页的某部分进行更新。

# 详细设计

## 3.1数据源

利用Java代码生成数据源

**public** **class** DataCreate {

**public** **static** String createDate(){

Random random = **new** Random();

String str = "";

//获取时间

Date date = **new** Date();

Long time = date.getTime();

//获取宽带账户

String userAccount = "022980298";

userAccount = userAccount + (random.nextInt(6666)+10);

//用户以太网IP

String userIP = "121.196.100.99";

//QQ号

String qqid = "6175864";

qqid = qqid + (random.nextInt(50));

//内网IP

String natIP = "192.168.1.";

natIP = natIP + (random.nextInt(10));

//Cookie值

String[] sArr = {"a","b","c","d","e","f","g","h","i","j","k","l","m","n","o","p","q","r","s","t","u","v","w","x","y","z","A","B","C","D","E","F","G","H","I","J","K","L","M","N","O","P","Q","R","S","T","U","V","W","X","Y","Z","0","1","2","3","4","5","6","7","8","9"};

String cookieValue = "";

**for**(**int** i=0;i<32;i++){

cookieValue = cookieValue + sArr[random.nextInt(sArr.length)];

}

//设备名称

String[] bArr = {"iphone","macbook","macpro","android","ipad"};

String devName = bArr[random.nextInt(bArr.length)];

//操作系统名称

String[] arr = {"mac os x 10.9","ios","win 8","win 10","win 7"};

String osName = arr[random.nextInt(arr.length)];

//将各个字段连接成一个我们需要的数据字段

str = time + " " + userAccount + " " + userIP + " " + qqid + " " + natIP + " " + cookieValue + " " + devName + " " + osName;

**return** str;

}

## 3.2数据传输

**3.21：TCP将生成的数据源传给两个采集节点**

//建立socket协议，将数据生成并发给Flume的采集节点1 public void tcpSendSocket1() throws Exception{

Socket s1 = null ;

String str = DataCreate.*createDate*();

s1 = **new** Socket("192.168.140.153", 5150);

PrintWriter pw = **new** PrintWriter(s1.getOutputStream());

pw.println(str);

System.*out*.println(str);

pw.close();

s1.close();

}

//建立socket协议，将数据生成并发给Flume的采集节点2

**public** **void** tcpSendSocket2() **throws** Exception{

Socket s1 = **null** ;

String str = DataCreate.*createDate*();

s1 = **new** Socket("192.168.140.163", 5150);

PrintWriter pw = **new** PrintWriter(s1.getOutputStream());

pw.println(str);

System.*out*.println(str);

pw.close();

s1.close();

}

**3.22：在两个从节点设计采集节点的Flume的配置文件**



**3.23：编写运行conf文件的bash脚本**



**3.23：在主节点编写的conf文件，收集采集节点发来的消息，并将数据发送到kafka**



## 3.3数据处理

**3.31实现UserAccount+IP去重的IPBolt**

**public** **class** IPBolt **extends** BaseBasicBolt{

**private** **static** **final** **long** *serialVersionUID* = 1L;

//创建一个Map集合用于去除5分中内相同的UserAccount+UserAccount

Map<String, Integer> map = **new** HashMap<String, Integer>();

//用于创建date日期对象

Date date = **new** Date();

//获取初始时间

Long startTime = date.getTime();

Long endTime = **null**;

Long time = **null**;

@Override

**public** **void** execute(Tuple tuple, BasicOutputCollector collector) {

// **TODO** Auto-generated method stub

//获取当前时间对象

date = **new** Date();

//获取当前时间

endTime = date.getTime();

//得到时间差

time = endTime - startTime ;

//获取spolt传过来的tuple对象

String str = tuple.getString(0);

//通过spolt传过来的tuple对象获取字符串

String[] sArr = str.split(" ");

//由UserAccount和UserAccount组成一个新的字符串对象

String nameIp = sArr[1] +" "+ sArr[4];

// System.out.println(nameIp + "11111111111111111111111111111");

//判断map集合里面是否包含新的字段用以去重并判断时间是否超过1分钟,不包含放入集合，并发送tuple对象

**if**(map.containsKey(nameIp)==**false** && time<60000){

// System.out.println(map.containsKey(nameIp) + "2222222222222222222222");

// System.out.println(time + "33333333333333333333333333333333333333");

map.put(nameIp, 1);

collector.emit(**new** Values(sArr[1]));

}

//如果时间超过1分钟，那么重置开始时间，并且清空集合

**if**(time >= 60000){

startTime = endTime;

map.clear();

}

}

@Override

**public** **void** declareOutputFields(OutputFieldsDeclarer declarer) {

// **TODO** Auto-generated method stub

//设置传过去的tuple的key的值

declarer.declare(**new** Fields("str"));

}

}

**3.32统计一分钟内userAccount相同但IP不同，次数大于5的userAccount**

**public** **class** IPBolt2 **extends** BaseBasicBolt{

//设置一个map集合用于记录userAccount相同，但是IP不同的账号

Map<String, Integer> map = **new** HashMap<String, Integer>();

//用于创建date日期对象

Date date = **new** Date();

Long startTime = date.getTime();

Long endTime = **null**;

OutputCollector collector = **null**;

@Override

**public** **void** execute(Tuple tuple, BasicOutputCollector collector) {

// **TODO** Auto-generated method stub

date = **new** Date();

//获取初始时间

endTime = date.getTime();

//获取IPBolt传过来的tuple对象

String userAccount = tuple.getString(0);

//设置userAccount的出现次数

**int** num = 1;

**if**(map.containsKey(userAccount)){

num = num + map.get(userAccount);

}

map.put(userAccount, num);

//判断userAccount的出现次数，如果次数超过5，并且时间在1分钟内，那么就发送userAccount

**if**(num > 5 && (endTime-startTime)<60000 ){

collector.emit(**new** Values(userAccount));

}

//如果时间超过1分钟，那么重置开始时间，并且清空集合以确保

**if**((endTime-startTime)>=60000){

startTime = endTime;

map.clear();

}

}

@Override

**public** **void** declareOutputFields(OutputFieldsDeclarer declare) {

// **TODO** Auto-generated method stub

//设置传过去的tuple的key的值

declare.declare(**new** Fields("str"));

}

}

**3.33实现UserAccount+cookieValue + devName + osName去重的OSBolt**

**public** **class** OSBolt **extends** BaseBasicBolt{

//创建一个Map集合用于去除5分中内相同的UserAccount + cookieValue + devName + osName

Map<String, Integer> map = **new** HashMap<String, Integer>();

Date date = **new** Date();

Long startTime = date.getTime();

Long endTime = **null**;

@Override

**public** **void** execute(Tuple tuple, BasicOutputCollector collector) {

// **TODO** Auto-generated method stub

date = **new** Date();

endTime = date.getTime();

String str = tuple.getString(0);

String[] sArr = str.split(" ");

//由UserAccount+cookieValue + devName + osName组成一个新的字符串对象

String nameFour = sArr[1] +" "+ sArr[5] + sArr[6] + sArr[7];

//判断map集合里面是否包含新的字段用以去重并判断时间是否超过1分钟,不包含放入集合，并发送tuple对象

**if**(map.containsKey(nameFour)==**false** && (endTime-startTime)<60000){

map.put(nameFour, 1);

collector.emit(**new** Values(nameFour));

}

//如果时间超过1分钟，那么重置开始时间，并且清空集合以确保

**if**((endTime-startTime) >= 60000){

map.clear();

startTime = endTime;

}

}

@Override

**public** **void** declareOutputFields(OutputFieldsDeclarer declarer) {

// **TODO** Auto-generated method stub

//设置传过去的tuple的key的值

declarer.declare(**new** Fields("str"));

}

## }

**3.34统计一分钟内userAccount相同ookieValue + devName + osName不同的账号，次数大于5的userAccount的OSBolt2**

**public** **class** OSBolt2 **extends** BaseBasicBolt{

//设置一个map集合用于记录userAccount相同，但是cookieValue + devName + osName不同的账号

Map<String, Integer> map = **new** HashMap<String, Integer>();

//用于创建date日期对象

Date date = **new** Date();

Long startTime = date.getTime();

Long endTime = **null**;

@Override

**public** **void** execute(Tuple tuple, BasicOutputCollector collector) {

// **TODO** Auto-generated method stub

date = **new** Date();

//获取初始时间

endTime = date.getTime();

//获取IPBolt传过来的tuple对象

String str = tuple.getString(0);

String[] sArr = str.split(" ");

String userAccount = sArr[0];

//设置userAccount的出现次数

**int** num = 1;

**if**(map.containsKey(userAccount)){

num = num + map.get(userAccount);

}

map.put(userAccount, num);

//判断userAccount的出现次数，如果次数超过5，并且时间在1分钟内，那么就发送userAccount

**if**(num > 5 && (endTime-startTime)<60000){

collector.emit(**new** Values(userAccount));

}

//如果时间超过1分钟，那么重置开始时间，并且清空集合以确保

**if**((endTime-startTime)>=60000){

map.clear();

startTime = endTime;

}

}

@Override

**public** **void** declareOutputFields(OutputFieldsDeclarer declarer) {

// **TODO** Auto-generated method stub

//设置传过去的tuple的key的值

declarer.declare(**new** Fields("userAccount"));

}

}

**3.35实现UserAccount+qqid去重的QQBolt**

**public** **class** QQBolt **extends** BaseBasicBolt{

//创建一个Map集合用于去除5分中内相同的UserAccount+qqid

Map<String, Integer> map = **new** HashMap<String, Integer>();

Date date = **new** Date();

Long startTime = date.getTime();

Long endTime = **null**;

@Override

**public** **void** execute(Tuple tuple, BasicOutputCollector collector) {

// **TODO** Auto-generated method stub

date = **new** Date();

endTime = date.getTime();

String str = tuple.getString(0);

String[] sArr = str.split(" ");

//由UserAccount和qqid组成一个新的字符串对象

String nameQq = sArr[1] +" "+sArr[3];

//判断map集合里面是否包含新的字段用以去重并判断时间是否超过1分钟，不包含放入集合，并发送tuple对象

**if**(map.containsKey(nameQq)==**false** && (endTime-startTime)<60000){

map.put(nameQq, 1);

collector.emit(**new** Values(nameQq));

}

//如果时间超过1分钟，那么重置开始时间，并且清空集合以确保

**if**((endTime-startTime)>=60000){

startTime = endTime;

map.clear();

}

}

@Override

**public** **void** declareOutputFields(OutputFieldsDeclarer declarer) {

// **TODO** Auto-generated method stub

//设置传过去的tuple的key的值

declarer.declare(**new** Fields("str"));

}}

**3.36统计一分钟内userAccount相同但是QQ不同的账号的账号，次数大于20的userAccount的OSBolt2**

**public** **class** QQBolt2 **extends** BaseBasicBolt{

//设置一个map集合用于记录userAccount相同，但是QQ不同的账号

Map<String, Integer> map = **new** HashMap<String, Integer>();

Date date = **new** Date();

Long startTime = date.getTime();

Long endTime = **null**;

@Override

**public** **void** execute(Tuple tuple, BasicOutputCollector collector) {

// **TODO** Auto-generated method stub

date = **new** Date();

endTime = date.getTime();

String str = tuple.getString(0);

String[] sArr = str.split(" ");

//获取QQBolt传过来的tuple对象

String userAccount = sArr[0];

//设置userAccount的出现次数

**int** num = 1;

**if**(map.containsKey(userAccount)){

num = num + map.get(userAccount);

}

map.put(userAccount, num);

//判断userAccount的出现次数，如果次数超过20，并且时间在1分钟内，那么就发送userAccount

**if**(num > 20 && (endTime-startTime)<60000){

collector.emit(**new** Values(userAccount));

}

//如果时间超过1分钟，那么重置开始时间，并且清空集合以确保

**if**((endTime-startTime)>=60000){

map.clear();

startTime = endTime;

}

}

@Override

**public** **void** declareOutputFields(OutputFieldsDeclarer declarer) {

// **TODO** Auto-generated method stub

declarer.declare(**new** Fields("userAccount"));

}}

**3.35实现将数据存入数据库的SaveBolt**

public class SaveBolt extends BaseBasicBolt {

//设置连接数据库的账号

private static String username = "hadoop";

//设置连接数据库的密码

private static String password = "hadoop";

//设置连接数据库的驱动

private static String driver = "com.mysql.jdbc.Driver";

//设置连接数据库的URL

private static String url = "jdbc:mysql://node:3306/storm";

private static Connection conn = null;

private static final long serialVersionUID = -5508421065181891596L;

//创建一个map集合用于去除3个bolt传过来的相同的UserAccount

private static Map<String, Integer> map = new HashMap<String, Integer>();

private static Date date = new Date();

private static Long startTime = date.getTime();

private static Long endTime = null;

static{

try{

//获取连接驱动

Class.forName(driver);

//设置连接驱动的参数

conn = DriverManager.getConnection(url, username, password);

}catch (Exception e) {

// TODO: handle exception

e.printStackTrace();

}

}

@Override

public void execute(Tuple tuple, BasicOutputCollector collector) {

// TODO Auto-generated method stub

date = new Date();

endTime = date.getTime();

String userAccount = tuple.getString(0);

map.put(userAccount, 1);

int num = 0;

//获取5分钟内出现的3种情况的UserAccount个数，并存入数据库，然后清空集合

if((endTime-startTime)>60000){

num = map.size();

String sql = "insert into account values(?,?)";

try{

PreparedStatement stmt = conn.prepareStatement(sql);

stmt.setString(1, startTime +"--" + endTime);

stmt.setInt(2, num);

stmt.execute();

}catch (Exception e) {

// TODO: handle exception

e.printStackTrace();

}

map.clear();

startTime = endTime;

}

}

@Override

public void declareOutputFields(OutputFieldsDeclarer arg0) {

// TODO Auto-generated method stub

}

}

## 3.4数据展示

**3.41web端的主要架构MVC**



### 

### 3.42链接数据库的实现

**数据连接池的配置文件**

driverClassName=com.mysql.jdbc.Driver

url=jdbc:mysql://192.168.140.158:3306/storm

username=hadoop

password=hadoop

initialSize=5

**数据库连接池的代码**

**public** **class** DBUtils {

**private** **static** DataSource *ds*;//连接池对象

**static** {

InputStream in = DBUtils.**class**.getClassLoader().getResourceAsStream("dbcp.properties");

//创建属性对象，加载连接池的配置参数

Properties pro = **new** Properties();

**try** {

pro.load(in);

//连接池工厂类创建连接池对象需要连接池的配置参数（通过一个属性对象传递）

*ds* = BasicDataSourceFactory.*createDataSource*(pro);

} **catch** (Exception e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

**public** **static** Connection getConnection() **throws** SQLException {

**return** *ds*.getConnection();

}

**public** **static** **void** close(Connection con, PreparedStatement ps, ResultSet rs) {

**try** {

// **TODO** Auto-generated method stub

//关闭各种资源

con.close();

ps.close();

rs.close();

} **catch** (Exception e) {

// **TODO**: handle exception

}

}

}

### 3.42 DAO层代码的编写

**Dao接口的编写**

package cn.yml.dao;

import java.util.List;

import cn.yml.web.bean.ymlProduct;

public interface ymlProductDao {

*/\*\**

*\* 查询商品信息*

*\**

*\* @param cid 类别id*

*\* @param pageSize 查询的条数*

*\* @param pageNum   页码*

*\* @param orderField    排序的字段*

*\* @param orderStyle 排序的方式*

*\* @return 商品列表*

*\*/*

    List<ymlProduct> queryErrorAccount();

}

**Dao实现类的编写**

package cn.yml.dao.impl;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.ArrayList;

import java.util.List;

import cn.yml.dao.ymlProductDao;

import cn.yml.utils.DBUtils;

import cn.yml.web.bean.ymlProduct;

public class ymlProductDaoImpl implements ymlProductDao {

    public List<ymlProduct> queryErrorAccount() {

        // 0. 定义一个集合将获取到的数据对象放入集合中

        List<ymlProduct> ymlProducts = *new* ArrayList<ymlProduct>();

        ymlProduct product = null;

        // 1. 获取连接对象

        Connection conn = null;

        PreparedStatement pst = null;

        ResultSet rs = null;

*try* {

            String sql = "select count from test3";

            //String sql2 = "select timeStamp,count from wordcount8";

            // 2. 执行sql语句

            // 2.1 使用数据库连接池，连接数据库

            conn = DBUtils.getConnection();

            // 2.2 准备执行SQL语句

            pst = conn.prepareStatement(sql);

            rs = pst.executeQuery();

            // 3. 处理结果数据

*while* (rs.next()) {

                product = *new* ymlProduct();

                product.setCount(rs.getInt("count"));

                ymlProducts.add(product);

            }

            // 4. 返回集合对象

*return* ymlProducts;

        } *catch* (SQLException e) {

            e.printStackTrace();

        } *finally* {

            //关闭连接池

            DBUtils.close(conn, pst, rs);

        }

*return* null;

    }

}

**Dao工厂类的实现**

package cn.yml.dao.factory;

import cn.yml.dao.ymlProductDao;

import cn.yml.dao.impl.ymlProductDaoImpl;

*/\*\**

*\* 这是一个工厂类用于获取impl实现类的对象*

*\*/*

public class ymlProductDaoFactory {

    public static ymlProductDao getymlProductInstance() {

*return* *new* ymlProductDaoImpl();

    }

}

### 3.43 Service层代码的编写

package cn.yml.service;

import java.util.List;

import cn.yml.dao.ymlProductDao;

import cn.yml.dao.factory.ymlProductDaoFactory;

import cn.yml.web.bean.ymlProduct;

*/\**

*\*处理商品相关的业务*

*\*/*

public class ymlProductService {

    private ymlProductDao dao;

    public ymlProductService() {

*super*();

*this*.dao = ymlProductDaoFactory.getymlProductInstance();

    }

*/\*\**

*\* 查询商品信息*

*\**

*\* @param cid*

*\* @param pageSize*

*\* @param pageNum*

*\* @param orderField*

*\* @param orderStyle*

*\* @return*

*\*/*

    public List<ymlProduct> queryErrorAccount() {

*return* dao.queryErrorAccount();

    }

}

**3.44 Servlet页面的编写**

package cn.yml.web.servlet;

import java.io.IOException;

import java.util.List;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import net.sf.json.JSONArray;

import cn.yml.service.ymlProductService;

import cn.yml.web.bean.ymlProduct;

*/\*\**

*\* Servlet implementation class Main*

*\*/*

public class Main extends HttpServlet {

    private static final long serialVersionUID = 1L;

*/\*\**

*\* @see HttpServlet#HttpServlet()*

*\*/*

    public Main() {

*super*();

        // *TODO* Auto-generated constructor stub

    }

*/\*\**

*\* @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse*

*\* response)*

*\*/*

    protected void doGet(HttpServletRequest request,

            HttpServletResponse response) throws ServletException, IOException {

        // *TODO* Auto-generated method stub

    }

*/\*\**

*\* @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse*

*\* response)*

*\*/*

    protected void doPost(HttpServletRequest request,

            HttpServletResponse response) throws ServletException, IOException {

        // *TODO* Auto-generated method stub

        ymlProductService productService = *new* ymlProductService();

*/\*\**

*\* ajax请求。请求商品信息*

*\**

*\* @param request*

*\* @param response*

*\* @throws ServletException*

*\* @throws IOException*

*\*/*

        // 2. 查询商品信息

        List<ymlProduct> categories = productService.queryErrorAccount();

        System.out.println("gets!!" + categories);

        // 2. 将结果返回给book\_list.jsp页面

        response.setCharacterEncoding("utf-8");

        response.getWriter().print(JSONArray.fromObject(categories).toString());

    }

}

**3.45 数据对象been的实现**

package cn.yml.web.bean;

public class ymlProduct {

    private String userAccount;// VARCHAR(50) ,-- 产品名

    private int count;// VARCHAR(50) ,-- 关键字

    public ymlProduct(String userAccount, int count) {

*super*();

*this*.userAccount = userAccount;

*this*.count = count;

    }

    public ymlProduct() {

*super*();

    }

    public String getUserAccount() {

*return* userAccount;

    }

    public void setUserAccount(String userAccount) {

*this*.userAccount = userAccount;

    }

    public int getCount() {

*return* count;

    }

    public void setCount(int count) {

*this*.count = count;

    }

    @Override

    public String toString() {

*return* "ymlProduct [userAccount=" + userAccount + ", count=" + count

                + "]";

    }

}

**3.46 JSP页面的编写**

<body>

*<!-- 为ECharts准备一个具备大小（宽高）的Dom -->*

    <div id="main" style="width: 600px; height: 400px;"></div>

</body>

<script type="text/javascript">

    var myChart = echarts.init(document.getElementById('main'));

*// 显示标题，图例和空的坐标轴*

    myChart.setOption({

        title : {

            text : '异常账户统计'

        },

        tooltip : {

            show : true

        },

        legend : {

*//data:['数量']*

        },

        xAxis : {

            data : []

        },

        yAxis : {},

        series : [ {

            name : '数量',

            type : 'bar',

            data : []

        } ]

    });

    myChart.showLoading();*//数据加载完之前先显示一段简单的loading动画*

*//myChart.setOption(option); // 使用刚指定的配置项和数据显示图表。*

    var names = []; *//异常账户数组（实际用来盛放X轴坐标值）*

    var nums = []; *//数量数组（实际用来盛放Y坐标值）*

*// alert("get!!");*

    function create(){

    $.ajax({

        type : "post",

*//async : true, //异步请求（同步请求将会锁住浏览器，用户其他操作必须等待请求完成才可以执行）*

        url : "Main", *//请求发送到TestServlet处*

        data : {},

        success : function(result) {

*//alert(result);*

*//请求成功时执行该函数内容，result即为服务器返回的json对象*

*if* (result) {

*for* ( var i = 0; i < result.length; i++) {

                    names.push(i); *//挨个取出类别并填入类别数组*

                    nums.push(result[i].count); *//挨个取出销量并填入销量数组*

                }

                myChart.hideLoading(); *//隐藏加载动画*

                myChart.setOption({ *//加载数据图表*

                    xAxis : {

                        data : names

                    },

                    series : [ {

*// 根据名字对应到相应的系列*

                        name : '数量',

                        data : nums

                    } ]

                });

                names=[];

                nums=[];

            }

        },

        dataType : "json", *//返回数据形式为json*

        cache : false,

        error : function(errorMsg) {

*//请求失败时执行该函数*

            alert("图表请求数据失败!");

            myChart.hideLoading();

        }

    });

    }

    $(function (){

        setInterval("create()",30000);

    });

</script>