搜狗热门话题实时分析

	hadoop101	hadoop102	hadoop103
hadoop	√	√	√
hbase	√	√	√
mysql	√		
hive	√		
flume	√	√	√
kafka	√	√	√
spark	√	√	√
scala	√	√	√

flume配置

数据汇聚（hadoop101）

首先进入到flume文件夹中的conf里，然后新建文件flume-a1-conf.properties

```
cd flume/conf

vim flume-a1-conf.properties
```

flume-a1-conf.properties配置信息

```
#Flume Agent1实时整合日志信息
a1.sources = r1
a1.channels = kafkaC hbaseC
a1.sinks = kafkaS hbaseS

#flume + hbase
a1.sources.r1.type = avro
a1.sources.r1.channels = kafkaC hbaseC
a1.sources.r1.bind = hadoop101 #根据自身情况更改名称或ip
a1.sources.r1.port = 5555

#使用 memory channel for HBase
a1.channels.hbaseC.type = memory
a1.channels.hbaseC.capacity = 10000
a1.channels.hbaseC.transactionCapacity = 10000

#修改 HBase Sink 类型
a1.sinks.hbaseS.type = org.apache.flume.sink.hbase.HBaseSink
```

```
a1.sinks.hbaseS.table = weblogs
a1.sinks.hbaseS.columnFamily = info
a1.sinks.hbaseS.batchSize = 100
a1.sinks.hbaseS.serializer =
org.apache.flume.sink.hbase.SimpleHbaseEventSerializer
a1.sinks.hbaseS.serializer.payloadColumn = payload
a1.sinks.hbaseS.channel = hbaseC

#flume + kafka
a1.channels.kafkaC.type = memory
a1.channels.kafkaC.capacity = 10000
a1.channels.kafkaC.transactionCapacity = 10000

a1.sinks.kafkaS.channel = kafkaC
a1.sinks.kafkaS.type = org.apache.flume.sink.kafka.KafkaSink
a1.sinks.kafkaS.topic = weblogs
#根据自身情况更改名称或ip
a1.sinks.kafkaS.brokerList = hadoop101:9092,hadoop102:9092,hadoop103:9092
a1.sinks.kafkaS.zookeeperConnect = hadoop101:2181,hadoop102:2181,hadoop103:2181
a1.sinks.kafkaS.requiredAcks = 1
a1.sinks.kafkaS.batchSize = 20
a1.sinks.kafkaS.serializer.class = kafka.serializer.StringEncoder
```

数据采集（hadoop102,hadoop103)

首先进入到flume文件夹中的conf里，然后新建文件flume-a2-conf.properties

flume-a2-conf.properties配置信息

```
a2.sources = r2
a2.channels = c2
a2.sinks = k2

# Source 配置
a2.sources.r2.type = exec
a2.sources.r2.command = tail -F /home/nanbei/weblog/weblog-flume.log
a2.sources.r2.channels = c2

# Channel 配置
a2.channels.c2.type = memory
a2.channels.c2.capacity = 5000
a2.channels.c2.transactionCapacity = 500
a2.channels.c2.keep-alive = 10

# Sink 配置
a2.sinks.k2.type = avro
a2.sinks.k2.channel = c2
a2.sinks.k2.hostname = hadoop101  #根据自身情况更改名称或ip
a2.sinks.k2.port = 5555
a2.sinks.k2.batch-size = 100
```

hadoop102与hadoop103中的flume配置相同

kafka配置

进入kafka下的conf文件夹中，修改文件server.properties，在文件中添加如下内容：

```
broker.id=2
listeners=PLAINTEXT://192.168.10.103:9092
advertised.listeners=PLAINTEXT://192.168.10.103:9092
zooker.connect=192.168.10.101:2181,192.168.10.102:2181,192.168.10.103:2181
```

这其中broker.id=0根据不同的虚拟机设置（hadoop101：0，hadoop102：1，hadoop103：2）；

将其中的ip以及端口号根据自身需求更改

编写WebLogs.jar包

ReadWrite类

打开idea，新建java项目，新建Java类ReadWrite。

ReadWrite类构建思路：

1. 获取输入文件路径，以及输出文件路径，并判断输出文件路径是否合法，是否有权创建输出文件
2. 每隔300ms读取一行输入文件中的数据，并将读取的数据添加到输出文件中

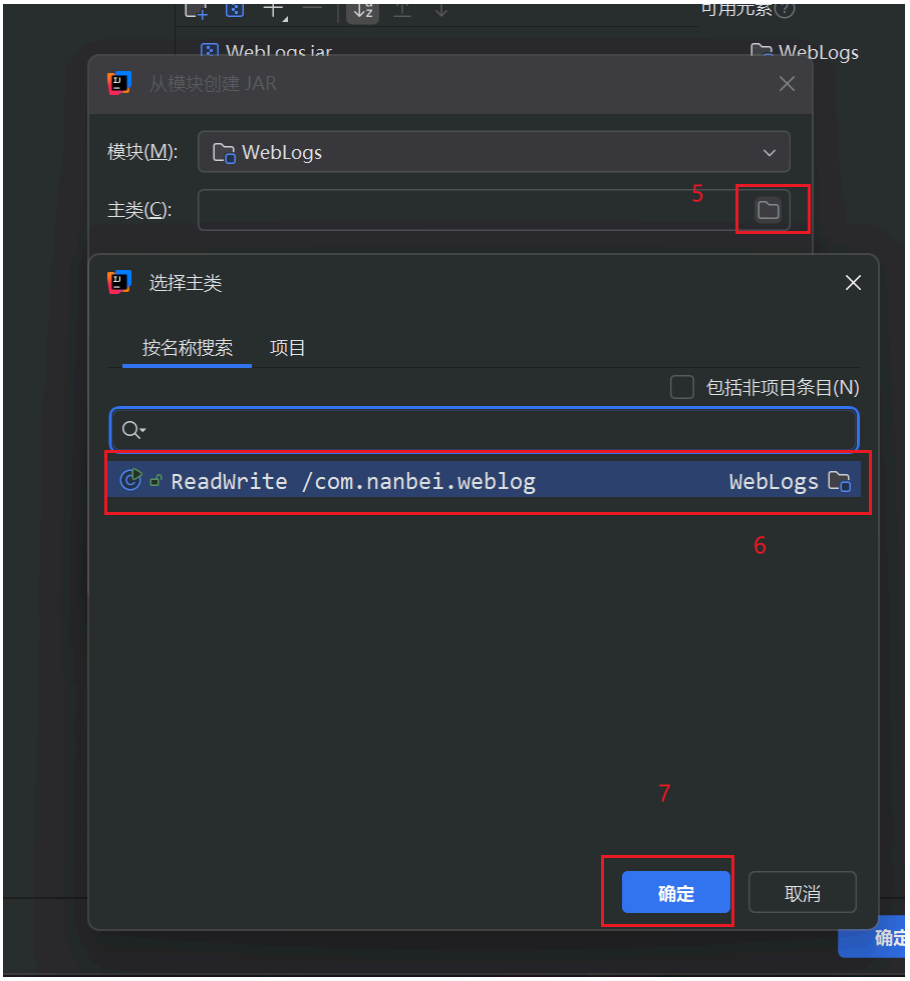
```
package com.nanbei.weblog;

import java.io.*;
import java.nio.charset.StandardCharsets;

public class ReadWrite {
    static String readFileName;
    static String writeFileName;

    public static void main(String[] args) {
        if (args.length < 2) {
            System.out.println("请提供读入文件路径和写入文件路径！");
            return;
        }

        // 获取输入和输出文件路径
        readFileName = args[0].trim();
        writeFileName = args[1].trim();

        // 打印路径检查
        System.out.println("输入文件路径：" + readFileName);
        System.out.println("输出文件路径：" + writeFileName);

        // 检查输出文件路径合法性
        File outputFile = new File(writeFileName);
        File parentDir = outputFile.getParentFile();
        if (parentDir != null && !parentDir.exists()) {
```

```
            boolean dirCreated = parentDir.mkdirs();
            if (!dirCreated) {
                System.err.println("无法创建输出目录：" + parentDir);
                return;
            }
        }

        try {
            readFileByLines(readFileName);
        } catch (FileNotFoundException e) {
            System.err.println("文件未找到：" + readFileName);
        } catch (IOException e) {
            System.err.println("文件读取失败：" + e.getMessage());
        }
    }

    public static void readFileByLines(String fileName) throws IOException {
        try (BufferedReader br = new BufferedReader(new InputStreamReader(new
FileInputStream(fileName), "GBK"))) {
            System.out.println("以行为单位读取文件内容：");
            String tempString;
            int count = 0;

            while ((tempString = br.readLine()) != null) {
                count++;
                System.out.println("row: " + count + " >>>>>>>> " + tempString);
                appendMethodA(writeFileName, tempString);
                Thread.sleep(300);
            }
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
            throw new IOException("线程被中断", e);
        }
    }

    public static void appendMethodA(String file, String content) throws
IOException {
        try (BufferedWriter out = new BufferedWriter(new OutputStreamWriter(new
FileOutputStream(file, true), StandardCharsets.UTF_8))) {
            out.write(content);
            out.newLine();
        }
    }
}
```

打包过程

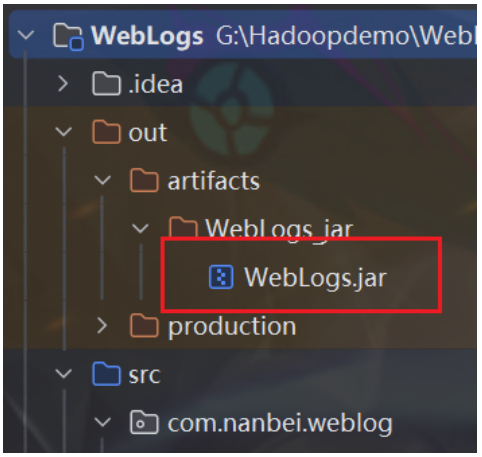
1.打开项目模块设置，选择工件

2.创建jar包，选择来自具有依赖的模块，选择刚新建的ReadWrite类



3.选择构建，构建工具，点击构建

4.创建成功后jar会出现在项目目录中的out\artfacts\WebLogs_jar文件夹下



5.上传到linux服务器中

改写flume源码

下载Flume源码，需要改写如下两个Java类

SimpleAsyncHbaseEventSerializer类，只需更改getActions()函数

```
@Override
  public List<PutRequest> getActions() {
    List<PutRequest> actions = new ArrayList<PutRequest>();
    if (payloadColumn != null) {
      byte[] rowKey;
      try {
        String[] columns = new String(this.payloadColumn).split(",");
        String[] values=new String(this.payload).split(",");

        for (int i = 0; i < columns.length; i++) {
          byte[] colColumn = columns[i].getBytes();
          byte[] colValue = values[i].getBytes(Charsets.UTF_8);

          if(colColumn.length!=colValue.length){
            break;
          }

          String datetime= values[0];
          String userid= values[1];
```

```
          rowKey=SimpleRowKeyGenerator.getKfkRowKey(userid,datetime);
          PutRequest putRequest=new
PutRequest(table,rowKey,cf,colColumn,colValue);
          actions.add(putRequest);
        }

      } catch (Exception e) {
        throw new FlumeException("Could not get row key!", e);
      }
    }
    return actions;
  }
```

SimpleRowKeyGenerator类，添加如下代码：

```
public static byte[] getKfkRowKey(String userid,String datetime) throws
UnsupportedEncodingException {
    return (userid+"-"+datetime+"-
"+String.valueOf(System.currentTimeMillis())).getBytes("UTF8");
  }
```

打成jar包，上传到linux服务器中替换原有flume目录的该jar包

数据处理

对下载的搜狗数据集进行处理，把数据文件中的Tap和空格换成逗号，在日志采集端（hadoop102,hadoop103）进行如下操作：

```
cat weblog.log|tr "\t" "," > weblog1.log
cat weblog1.log|tr " " "," > weblog.log
```

最终得到如下数据：

```
00:00:00,2982199073774412,[360安全士],8,3,download.it.com.cn/softweb/software/firewall/antivir
00:00:00,07594220010824798,[哄抢救灾物资],1,1,news.21cn.com/social/daqian/2008/05/29/4777194_1
00:00:00,5228056822071097,[75810部队],14,5,www.greatoo.com/greatoo_cn/list.asp?link_id=276&tit
00:00:00,6140463203615646,[绳艺],62,36,www.jd-cd.com/jd_opus/xx/200607/706.html
00:00:00,8561366108033201,[汶川地震原因],3,2,www.big38.net/
00:00:00,23908140386148713,[莫衷一是的意思],1,2,www.chinabaike.com/article/81/82/110/2007/2007
00:00:00,1797943298449139,[星梦缘全集在线观看],8,5,www.6wei.net/dianshiju/????\xa1\xe9|????do=
00:00:00,00717725924582846,[闪字吧],1,2,www.shanziba.com/
00:00:00,41416219018952116,[霍震霆与朱玲玲照片],2,6,bbs.gouzai.cn/thread-698736.html
 00:00:00,9975666857142764,[电脑创业],2,2,ks.cn.yahoo.com/question/1307120203719.html
 00:00:00,21603374619077448,[111aa图片],1,6,www.fotolog.com.cn/tags/aa111
 00:00:00,7423866288265172,[豆腐的制成],3,13,ks.cn.yahoo.com/question/1406051201894.html
```

编写启动脚本

将所有的启动脚本以及搜狗热搜数据统一放在一个文件夹中

flume起动脚本

数据汇聚汇聚（hadoop101）flume起动脚本

```
vim flume.sh
```

```
#!/bin/bash

# 输出启动信息
echo "Starting Flume Agent..."

# 设置 Flume 的安装路径（根据实际情况调整）
FLUME_HOME="/home/nanbei/hadoop/flume-1.11.0"

# 设置配置文件路径
CONF_FILE="$FLUME_HOME/conf/flume-a1-conf.properties"

# 启动 Flume Agent，指定配置文件和日志级别
$FLUME_HOME/bin/flume-ng agent --conf $FLUME_HOME/conf --name a1 --conf-file
$CONF_FILE -Dflume.root.logger=INFO,console
```

数据采集（hadoop102，hadoop103）flume起动脚本

```
vim flume.sh
```

```
#!/bin/bash

# 输出启动信息
echo "Starting Flume Agent..."

# 设置 Flume 的安装路径（根据实际情况调整）
FLUME_HOME="/home/nanbei/hadoop/flume-1.11.0"

# 设置配置文件路径
CONF_FILE="$FLUME_HOME/conf/flume-a2-conf.properties"

# 启动 Flume Agent，指定配置文件和日志级别
$FLUME_HOME/bin/flume-ng agent --conf $FLUME_HOME/conf --name a2 --conf-file
$CONF_FILE -Dflume.root.logger=INFO,console
```

WebLogs.jar包运行脚本

```
vim weblog.sh
```

```
#!/bin/bash

java -jar WebLogs.jar weblog.log weblog-flume.log
```

脚本说明：java -jar后面的第一个参数是WebLogs.jar的文件路径，第二个参数是输入文件的路径，第三个是输出文件的路径

kafka脚本

发送消息(hadoop101)

```
vim kafka-flume.sh

#/bin/bash
echo "flume agent1 start"
kafka-console-producer.sh --broker-list
hadoop101:9092,hadoop102:9092,hadoop103:9092 --topic weblogs
```

消费消息(hadoop102,hadoop103)

```
vim kafka-flume.sh

#!/bin/bash
kafka-console-consumer.sh --bootstrap-server
hadoop101:9092,hadoop102:9092,hadoop103:9092 --topic weblogs --from-beginning
```

项目搭建

IDEA中新建Spring Boot项目，构建工具选择Maven。

pom.xml

pom.xml应包含 Spring Boot Web 应用、WebSocket 支持、MySQL 数据库、Kafka 消费者、HikariCP 连接池、Spark、Hadoop 客户端等依赖，代码如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <!-- Spring Boot Parent POM -->
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.4.0</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>

    <groupId>com.nrsas</groupId>
    <artifactId>News-Real-time-Statistical-Analysis-System</artifactId>
    <version>1.0-SNAPSHOT</version>
    <name>News-Real-time-Statistical-Analysis-System</name>
    <description>News-Real-time-Statistical-Analysis-System</description>
```

```
    <properties>
        <java.version>17</java.version>
        <hadoop.version>3.3.6</hadoop.version>
        <scala.binary.version>2.13</scala.binary.version>
        <spark.version>3.4.3</spark.version>
        <maven.compiler.source>17</maven.compiler.source>
        <maven.compiler.target>17</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>

    <dependencies>
        <!-- Spring Boot Web and WebSocket Dependencies -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-websocket</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jdbc</artifactId>
        </dependency>

        <!-- MySQL Connector Dependency -->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>8.0.28</version>
            <scope>runtime</scope>
        </dependency>

        <!-- Kafka Consumer -->
        <dependency>
            <groupId>org.apache.kafka</groupId>
            <artifactId>kafka-clients</artifactId>
            <version>3.4.0</version>
        </dependency>

        <!-- HikariCP for Database Connection Pooling -->
        <dependency>
            <groupId>com.zaxxer</groupId>
            <artifactId>HikariCP</artifactId>
            <version>4.0.3</version>
        </dependency>

        <!-- Spark Core Dependencies -->
        <dependency>
            <groupId>org.apache.spark</groupId>
            <artifactId>spark-core_${scala.binary.version}</artifactId>
            <version>${spark.version}</version>
        </dependency>
        <dependency>
            <groupId>org.apache.spark</groupId>
            <artifactId>spark-sql_${scala.binary.version}</artifactId>
```

```
            <version>${spark.version}</version>
        </dependency>
        <dependency>
            <groupId>org.apache.spark</groupId>
            <artifactId>spark-streaming_${scala.binary.version}</artifactId>
            <version>${spark.version}</version>
        </dependency>
        <dependency>
            <groupId>org.apache.spark</groupId>
            <artifactId>spark-hive_${scala.binary.version}</artifactId>
            <version>${spark.version}</version>
        </dependency>
        <dependency>
            <groupId>org.apache.spark</groupId>
            <artifactId>spark-streaming-kafka-0-10_${scala.binary.version}
</artifactId>
            <version>${spark.version}</version>
        </dependency>
        <dependency>
            <groupId>org.apache.spark</groupId>
            <artifactId>spark-sql-kafka-0-10_${scala.binary.version}</artifactId>
            <version>${spark.version}</version>
        </dependency>

        <!-- Hadoop Client Dependency -->
        <dependency>
            <groupId>org.apache.hadoop</groupId>
            <artifactId>hadoop-client</artifactId>
            <version>${hadoop.version}</version>
        </dependency>

        <!-- FastJSON -->
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>fastjson</artifactId>
            <version>1.2.83</version>
        </dependency>

        <!-- JUnit for Testing -->
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.13.1</version>
            <scope>test</scope>
        </dependency>

        <!-- Java EE API (Provided) -->
        <dependency>
            <groupId>javax</groupId>
            <artifactId>javaee-api</artifactId>
            <version>8.0</version>
            <scope>provided</scope>
        </dependency>
    </dependencies>

    <repositories>
```

```
        <repository>
            <id>glassfish-releases</id>
            <url>https://maven.java.net/content/repositories/glassfish-
releases/</url>
        </repository>
    </repositories>

    <build>
        <finalName>spark_socket</finalName>

        <pluginManagement>
            <plugins>
                <!-- Spring Boot Maven Plugin -->
                <plugin>
                    <groupId>org.springframework.boot</groupId>
                    <artifactId>spring-boot-maven-plugin</artifactId>
                </plugin>

                <!-- Clean Plugin -->
                <plugin>
                    <artifactId>maven-clean-plugin</artifactId>
                    <version>3.1.0</version>
                </plugin>

                <!-- Resource Plugin -->
                <plugin>
                    <artifactId>maven-resources-plugin</artifactId>
                    <version>3.0.2</version>
                </plugin>

                <!-- Compiler Plugin -->
                <plugin>
                    <artifactId>maven-compiler-plugin</artifactId>
                    <version>3.8.0</version>
                </plugin>

                <!-- Surefire Plugin for Testing -->
                <plugin>
                    <artifactId>maven-surefire-plugin</artifactId>
                    <version>2.22.1</version>
                </plugin>

                <!-- War Plugin (if you plan to deploy as WAR) -->
                <plugin>
                    <artifactId>maven-war-plugin</artifactId>
                    <version>3.2.2</version>
                </plugin>

                <!-- Install Plugin -->
                <plugin>
                    <artifactId>maven-install-plugin</artifactId>
                    <version>2.5.2</version>
                </plugin>

                <!-- Deploy Plugin -->
                <plugin>
```

```
                    <artifactId>maven-deploy-plugin</artifactId>
                    <version>2.8.2</version>
                </plugin>
            </plugins>
        </pluginManagement>
    </build>
</project>
```

KafkaToMySQL类

新建包com.spartk.kafkatomysql，在此包中新建Java类KafkaToMySQL实现了从 Kafka 消费消息并存储到 MySQL 数据库的功能

数据库搭建

新建数据库test，新建表weblogs，title_counts

```
CREATE TABLE title_counts (
    titleName VARCHAR(255) NOT NULL,     -- 存储查询词
    count INT NOT NULL,                  -- 存储查询词的个数
    PRIMARY KEY (titleName)              -- 设置 titleName 为主键
);

CREATE TABLE weblogs (
    datetime VARCHAR(255) NOT NULL,      -- 存储日志的时间
    userid VARCHAR(255) NOT NULL,        -- 存储用户ID
    searchname VARCHAR(255) NOT NULL,    -- 存储搜索的查询词
    retorder VARCHAR(255),               -- 存储返回订单
    cliorder VARCHAR(255),               -- 存储客户端订单
    cliurl VARCHAR(255),                 -- 存储客户端URL
    PRIMARY KEY (datetime, userid)       -- 使用 datetime 和 userid 作为主键
);
```

配置application.properties

```
spring.datasource.url=jdbc:mysql://localhost:3306/test?
useUnicode=true&characterEncoding=utf-8
spring.datasource.username=root
spring.datasource.password=123580asd
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.sql.init.platform=mysql
spring.datasource.hikari.maximum-pool-size=10
```

构建KafkaToMySQL类

1. 消费 Kafka 消息，配置订阅指定主题并处理消息偏移量
2. 配置 `HikariCP` 提供数据库连接池

3. 解析 Kafka 消息并验证其格式，将解析拆分后的字段转化为数据库表中对应的结构化数据。

4. 将处理好的结构化数据批量插入数据库

```
package com.spark.kafka;

import com.zaxxer.hikari.HikariConfig;
import com.zaxxer.hikari.HikariDataSource;
import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.kafka.clients.consumer.KafkaConsumer;
import org.apache.kafka.common.serialization.StringDeserializer;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.util.Collections;
import java.util.Properties;

/**
 * @author nanbei
 * @since 2024/11/23
 */
public class KafkaToMySQL {
    private static final String JDBC_URL = "jdbc:mysql://localhost:3306/test"; //
替换为你的 MySQL 数据库地址
    private static final String USERNAME = "root"; // MySQL 用户名
    private static final String PASSWORD = "123580asd"; // MySQL 密码

    // 配置数据库连接池
    private static HikariDataSource dataSource;

    static {
        HikariConfig config = new HikariConfig();
        config.setJdbcUrl(JDBC_URL);
        config.setUsername(USERNAME);
        config.setPassword(PASSWORD);
        config.setMaximumPoolSize(10); // 连接池大小
        dataSource = new HikariDataSource(config);
    }

    public static void main(String[] args) {
        // 配置 Kafka 消费者
        Properties props = new Properties();
        props.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, "hadoop102:9092");
//Kafka 地址
        props.put(ConsumerConfig.GROUP_ID_CONFIG, "1"); // 消费者组 ID
        props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
StringDeserializer.class.getName());
        props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
StringDeserializer.class.getName());
        props.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest");

        KafkaConsumer<String, String> consumer = new KafkaConsumer<>(props);

        // Kafka 主题
        consumer.subscribe(Collections.singletonList("weblogs"));
```

```
        try {
            while (true) {
                // 从 Kafka 中拉取消息
                consumer.poll(1000).forEach(record -> {
                    String message = record.value();
                    System.out.println("Consumed message: " + message);

                    // 每条消息是逗号分隔的数据格式
                    String[] fields = message.split(",");
                    if (fields.length < 6) {
                        System.out.println("Invalid message format: " + message);
                        return;
                    }
                    String datetime = fields[0];
                    String userid = fields[1];
                    String searchname = fields[2];
                    String retorder = fields[3];
                    String cliorder = fields[4];
                    String cliurl = fields[5];

                    // 将数据保存到数据库
                    boolean isSaved = saveToDatabase(datetime, userid,
searchname, retorder, cliorder, cliurl);
                    if (isSaved) {
                        System.out.println("Data successfully saved to database:
" + message);
                    } else {
                        System.out.println("Failed to save data to database: " +
message);
                    }
                });
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            consumer.close();
        }
    }

    // 将数据插入到 MySQL 数据库
    private static boolean saveToDatabase(String datetime, String userid, String
searchname, String retorder, String cliorder, String cliurl) {
        try (Connection connection = dataSource.getConnection()) {
            // 插入日志数据到 weblogs 表
            String insertWeblogSql = "INSERT INTO weblogs(datetime, userid,
searchname, retorder, cliorder, cliurl) " +
                    "VALUES (?, ?, ?, ?, ?, ?) " +
                    "ON DUPLICATE KEY UPDATE searchname = VALUES(searchname), " +
                    "retorder = VALUES(retorder), cliorder = VALUES(cliorder),
cliurl = VALUES(cliurl)";
            try (PreparedStatement insertWeblogStmt =
connection.prepareStatement(insertWeblogSql)) {
                insertWeblogStmt.setString(1, datetime);
                insertWeblogStmt.setString(2, userid);
                insertWeblogStmt.setString(3, searchname);
                insertWeblogStmt.setString(4, retorder);
```

```
                insertWeblogStmt.setString(5, cliorder);
                insertWeblogStmt.setString(6, cliurl);
                insertWeblogStmt.executeUpdate();
            }

            // 更新 title_counts 表，统计查询词的出现次数
            String updateTitleCountSql = "INSERT INTO title_counts(titleName,
count) " +
                    "VALUES (?, 1) " +
                    "ON DUPLICATE KEY UPDATE count = count + 1";
            try (PreparedStatement updateTitleCountStmt =
connection.prepareStatement(updateTitleCountSql)){
                updateTitleCountStmt.setString(1, searchname);
                updateTitleCountStmt.executeUpdate();
            }

            return true; // 如果插入或更新成功，则返回 true
        } catch (Exception e) {
            e.printStackTrace();
            return false; // 如果有异常发生，返回 false
        }
    }
}
```

WeblogService类

WeblogService类主要负责与数据库进行交互，实现查询热门话题的统计数据，获取数据库中话题的总数量

WeblogService类的构建：

1. 连接数据库
2. 使用`SELCET`语句筛选符合规则的数据
3. 提取指定字段的数据，并返回

```
package com.spark.service;


import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.sql.*;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.HashMap;

/**
 * @author nanbei
 * @since 2024-11-24
 */
public class WeblogService {
```

```
    private static final String URL = "jdbc:mysql://localhost:3306/test?
useUnicode=true&characterEncoding=utf-8";
    private static final String USERNAME = "root";
    private static final String PASSWORD = "123580asd";

    private static final Logger logger =
LoggerFactory.getLogger(WeblogService.class);

    /**
     * 按话题分组，取统计数据
     * @return 话题名称和统计数据的 Map
     */
    public Map<String, Object> queryWeblog() {
        Connection conn = null;
        PreparedStatement pst = null;
        ResultSet rs = null;
        Map<String, Object> retMap = new HashMap<>();
        List<String> titleNames = new ArrayList<>();
        List<Integer> titleCounts = new ArrayList<>();

        try {
            // 连接数据库
            Class.forName("com.mysql.cj.jdbc.Driver");
            conn = DriverManager.getConnection(URL, USERNAME, PASSWORD);
            String querySql = "SELECT titleName, count FROM title_counts ORDER BY
count DESC LIMIT 20";
            pst = conn.prepareStatement(querySql);

            rs = pst.executeQuery();
            while (rs.next()) {
                titleNames.add(rs.getString("titleName"));
                titleCounts.add(rs.getInt("count"));
            }

            retMap.put("titleName", titleNames);
            retMap.put("titleCount", titleCounts);
        } catch (SQLException | ClassNotFoundException e) {
            logger.error("Error executing queryWeblog", e);
        } finally {
            closeResources(rs, pst, conn);
        }

        return retMap;
    }

    /**
     * 获取话题总数
     * @return 话题总数的数组
     */
    public String[] titleCount() {
        Connection conn = null;
        PreparedStatement pst = null;
        ResultSet rs = null;
        String[] titleSums = new String[1];

        try {
```

```
            //MySQL操作
            Class.forName("com.mysql.cj.jdbc.Driver");
            conn = DriverManager.getConnection(URL, USERNAME, PASSWORD);
            String querySql = "SELECT COUNT(1) AS titleSum FROM title_counts";
            pst = conn.prepareStatement(querySql);

            rs = pst.executeQuery();
            if (rs.next()) {
                titleSums[0] = rs.getString("titleSum");
            }
        } catch (SQLException | ClassNotFoundException e) {
            logger.error("Error executing titleCount", e);
        } finally {
            closeResources(rs, pst, conn);
        }

        return titleSums;
    }

    /**
     * 关闭数据库资源
     */
    private void closeResources(ResultSet rs, PreparedStatement pst, Connection
conn) {
        try {
            if (rs != null) {
                rs.close();
            }
            if (pst != null) {
                pst.close();
            }
            if (conn != null) {
                conn.close();
            }
        } catch (SQLException e) {
            logger.error("Error closing database resources", e);
        }
    }
}
```

WeblogSocket

WebSocketConfig类

配置 `webSocket` 的连接，注册处理 `webSocket` 连接的处理器，访问路径设置为：`/websocket`

```
package com.spark.service;

import org.springframework.context.annotation.Configuration;
import org.springframework.web.socket.config.annotation.EnableWebSocket;
import org.springframework.web.socket.config.annotation.WebSocketConfigurer;
import org.springframework.web.socket.config.annotation.WebSocketHandlerRegistry;
```

```java
/**
 * @author nanbei
 * @since 2024-11-24
 */
@Configuration
@EnableWebSocket
public class WebSocketConfig implements WebSocketConfigurer {

    @Override
    public void registerWebSocketHandlers(WebSocketHandlerRegistry registry) {
        // 使用 TextWebSocketHandler 的实现类来处理 WebSocket 请求
        registry.addHandler(new WeblogSocketHandler(), "/websocket")
                .setAllowedOrigins("*");
    }
}
```

WeblogScoketHandler类

实现每隔 5 秒从 `WeblogService` 获取统计数据，通过 `WebSocket` 发送到前端的功能

WeblogScoketHandler类的构建：

1. 创建一个定时线程池
2. 当 WebSocket 连接建立时，启动定时任务
3. 将数据转换为 `JSON` 格式，定期向客户端推送数据
4. 遇到异常，关闭当前 WebSocket 连接

```java
package com.spark.service;

import org.springframework.web.socket.CloseStatus;
import org.springframework.web.socket.WebSocketSession;
import org.springframework.web.socket.handler.TextWebSocketHandler;
import org.springframework.web.socket.TextMessage;

import java.io.IOException;
import java.util.HashMap;
import java.util.Map;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.atomic.AtomicInteger;
import java.util.concurrent.*;

import com.alibaba.fastjson.JSON;

/**
 * WeblogSocketHandler 处理 WebSocket 消息
 * @author nanbei
 * @since 2024-11-24
 */
public class WeblogSocketHandler extends TextWebSocketHandler {

    private static final ThreadFactory THREAD_FACTORY = new ThreadFactory() {
        private final AtomicInteger threadNumber = new AtomicInteger(1);
```

```
        @Override
        public Thread newThread(Runnable r) {
            Thread thread = new Thread(r);
            thread.setName("websocket-pool-thread-" +
threadNumber.getAndIncrement());
            return thread;
        }
    };

    private static final ScheduledExecutorService scheduledExecutorService =
Executors.newScheduledThreadPool(2, THREAD_FACTORY);
    private final WeblogService weblogService = new WeblogService();

    private ScheduledFuture<?> scheduledFuture;

    @Override
    public void afterConnectionEstablished(WebSocketSession session) throws
Exception {
        // 每隔 5 秒发送一次数据
        scheduledFuture = scheduledExecutorService.scheduleAtFixedRate(() -> {
            try {
                // 获取话题名称和数量
                Map<String, Object> map = new HashMap<>();
                map.put("titleName",
weblogService.queryWeblog().get("titleName"));
                map.put("titleCount",
weblogService.queryWeblog().get("titleCount"));
                map.put("titleSum", weblogService.titleCount());

                // 将数据转为 JSON 字符串
                String jsonResponse = JSON.toJSONString(map);

                // 发送 JSON 数据
                session.sendMessage(new TextMessage(jsonResponse));

            } catch (IOException e) {
                e.printStackTrace();
                try {
                    session.close();
                } catch (IOException ioException) {
                    ioException.printStackTrace();
                }
            }
        }, 0, 5, TimeUnit.SECONDS); // 初始延迟 0 秒，周期为 5 秒
    }


    @Override
    public void afterConnectionClosed(WebSocketSession session, CloseStatus
status) throws Exception {

        // 取消当前连接的定时任务
        if (scheduledFuture != null && !scheduledFuture.isCancelled()) {
            scheduledFuture.cancel(true);
        }
```

```
    }
}
```

前端

javascript

主要功能是通过 `WebSocket` 实现从后端实时接收数据，动态更新图表和表格内容，并具备连接断开后的重连机制。

构建思路：

1. 创建与后端的 `WebSocket` 连接，设置事件监听器（`onopen`、`onmessage`、`onerror`、`onclose`）
2. 断开连接调用重连函数尝试恢复连接，重连时延时 5 秒，避免重连过于频繁出现异常
3. 使用 `ECharts` 渲染图表，如果图表尚未初始化，则进行初始化
4. 更新数据并调用 `setOption` 渲染图表。条形图： 展示话题名称和数量；仪表盘： 显示话题总量
5. 动态表格：清空表格内容后重新插入新的数据行，循环构建表格行，根据 `titleName` 和 `titleCount` 显示每个话题及其数量

```
// WebSocket 连接
let socket = new WebSocket('ws://localhost:8080/websocket');

// 只初始化一次图表
let titleChart = null;
let exposureGauge = null;

// WebSocket 连接成功时
socket.onopen = function(event) {
    console.log('WebSocket connection established');
};

// 处理消息
socket.onmessage = function(event) {
    const data = JSON.parse(event.data);

    // 更新图表和仪表盘
    updateCharts(data);
    fillTable(data);
};

// 图表更新
function updateCharts(data) {
    // 如果没有初始化图表，则初始化图表
    if (!titleChart) {
        titleChart = echarts.init(document.getElementById('titleChart'));
    }
    if (!exposureGauge) {
        exposureGauge = echarts.init(document.getElementById('topicGauge'));
    }
```

```
// 更新条形图
    const titleOption = {
        tooltip: { trigger: 'axis', axisPointer: { type: 'shadow' } },
        grid: { left: '10%', right: '10%', bottom: '15%', top: '15%' }, // 增加上
下边距
        xAxis: {
            type: 'value', // 设置 x 轴为数值轴
            splitLine: { lineStyle: { type: 'dotted' } }, // 横向网格线
        },
        yAxis: {
            type: 'category', // 设置 y 轴为类目轴
            data: limitedTitleName, // 只显示前15个类目
            axisLabel: {
                formatter: value => value.length > 15 ? value.slice(0, 15) +
'...' : value, // 长度超过15的文本添加省略号
                rotate: 0, // 不旋转y轴标签
                interval: 0, // 确保所有标签都显示
                fontSize: 11, // 调整字体大小
            }
        },
        series: [{
            data: limitedTitleCount, // 只显示前15个数据
            type: 'bar', // 类型设置为条形图
            barCategoryGap: '40%', // 条形之间的间距
            itemStyle: {
                color: '#d1250d',
                borderRadius: [0, 10, 10, 0] // 圆角
            },
            label: { show: true, position: 'right' } // 显示标签
        }]
    };

    titleChart.setOption(titleOption, true);



// 更新仪表盘
    const gaugeOption = {
        tooltip: { formatter: '{a} <br/>{b}: {c}个' }, // 提示框显示的内容
        series: [{
            name: '话题曝光量',  // 仪表盘的名称
            type: 'gauge',      // 类型为仪表盘
            startAngle: 225,    // 仪表盘起始角度
            endAngle: -45,      // 仪表盘结束角度
            radius: '90%',      // 仪表盘的半径
            min: 0,             // 设置最小值
            max: 10000,         // 设置最大值
            axisLine: {
                lineStyle: {
                    width: 20,  // 轴线的宽度
                    color: [
                        [0.2, '#6bcf85'],  // 绿色，值在0-2000之间时显示
                        [0.8, '#4f98ca'],  // 蓝色，值在2000-8000之间时显示
                        [1, '#f45b5b']     // 红色，值在8000-10000之间时显示
                    ]
```

```
                }
            },
            pointer: {
                width: 6,          // 指针的宽度
                length: '70%',     // 指针的长度
                color: 'auto'      // 自动根据数据变化设置指针颜色
            },
            detail: {
                valueAnimation: true, // 启动数值动画，平滑显示数值变化
                formatter: '{value}个话题', // 显示的详细信息格式
                fontSize: 24,          // 数值的字体大小
                offsetCenter: [0, '70%'] // 设置详细信息的显示位置，向下偏移70%
            },
            title: {
                offsetCenter: [0, '-30%'] // 设置仪表盘标题的位置，向上偏移30%
            },
            data: [{
                value: data.titleSum, // 动态获取话题总数量的数据
                name: '话题数量'       // 显示在仪表盘上的数据名称
            }]
        }]
    };

// 设置仪表盘的配置项并应用
    exposureGauge.setOption(gaugeOption, true);


}

// 动态填充表格内容
function fillTable(data) {
    const tableBody = document.getElementById('titleTableBody');
    tableBody.innerHTML = ''; // 清空之前的表格内容
    data.titleName.forEach((title, index) => {
        const row = document.createElement('tr');
        row.innerHTML = `<td>${title}</td><td>${data.titleCount[index]}</td>`;
        tableBody.appendChild(row);
    });
}

// 处理 WebSocket 错误
socket.onerror = function(event) {
    console.error('WebSocket error:', event);
};

// 处理 WebSocket 关闭
socket.onclose = function(event) {
    console.log('WebSocket connection closed:', event);
    // 尝试重连
    reconnectWebSocket();
};

// 连接断开后尝试重连
function reconnectWebSocket() {
    setTimeout(() => {
        console.log('Reconnecting WebSocket...');
```

```
        socket = new WebSocket('ws://localhost:8080/websocket'); // 创建新的
WebSocket 连接
        socket.onopen = function() {
            console.log('Reconnected WebSocket connection established');
        };
        socket.onmessage = function(event) {
            const data = JSON.parse(event.data);
            updateCharts(data);
            fillTable(data);
        };
        socket.onerror = function(event) {
            console.error('WebSocket error during reconnect:', event);
        };
        socket.onclose = function(event) {
            console.log('WebSocket connection closed after reconnect:', event);
            reconnectWebSocket(); // 如果重新连接关闭，再次尝试重连
        };
    }, 5000); // 5秒后重连
}
```

css

设计Web样式

```
/* 页面基础样式 */
body {
    font-family: Arial, sans-serif;
    background-color: #f4f4f4;
    margin: 0;
    padding: 0;
    display: flex;
    flex-direction: column;
    align-items: center; /* 水平居中 */
    justify-content: flex-start;
    min-height: 100vh; /* 页面占满整个视口 */
}

/* 居中显示标题 */
h1.text-center {
    text-align: center;
    margin-top: 30px;
    margin-bottom: 30px;
    font-size: 24px;
}

/* 小标题样式 */
.section-title {
    font-size: 20px; /* 设置小标题的字体大小 */
    font-weight: bold; /* 小标题加粗 */
    color: #333; /* 小标题颜色 */
    margin-top: 40px; /* 小标题与上方内容的间距 */
    margin-bottom: 10px; /* 小标题与下方内容的间距 */
```

```
    text-align: center; /* 小标题居中 */
}

/* 页脚样式 */
.footer {
    text-align: center; /* 居中 */
    font-size: 14px; /* 设置字体大小 */
    color: #777; /* 字体颜色 */
    margin-top: 40px; /* 与内容保持距离 */
    padding: 20px 0; /* 上下内边距 */
    background-color: #f4f4f4; /* 背景色 */
}

/* 图表容器样式 */
.chart-container {
    width: 80%; /* 图表宽度占80% */
    height: 400px; /* 高度400px */
    margin-bottom: 20px;
    background: #fff;
    padding: 20px;
    border-radius: 10px;
    box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
    box-sizing: border-box; /* 包括内边距和边框 */
}

/* 表格样式 */
table {
    width: 80%; /* 表格宽度占80% */
    border-collapse: collapse;
    margin-top: 20px;
    background-color: white;
    box-sizing: border-box; /* 包括边框 */
}

/* 表头和表格内容的样式 */
table th, table td {
    padding: 10px;
    border: 1px solid #ddd;
    text-align: left; /* 表格内容居中 */
}

table th {
    background-color: #f0f0f0;
    font-weight: bold;
}

/* 响应式设计：当屏幕小于 600px 时，调整图表和表格宽度 */
@media (max-width: 600px) {
    .chart-container, table {
        width: 95%; /* 图表和表格宽度为95% */
    }
}
```

html

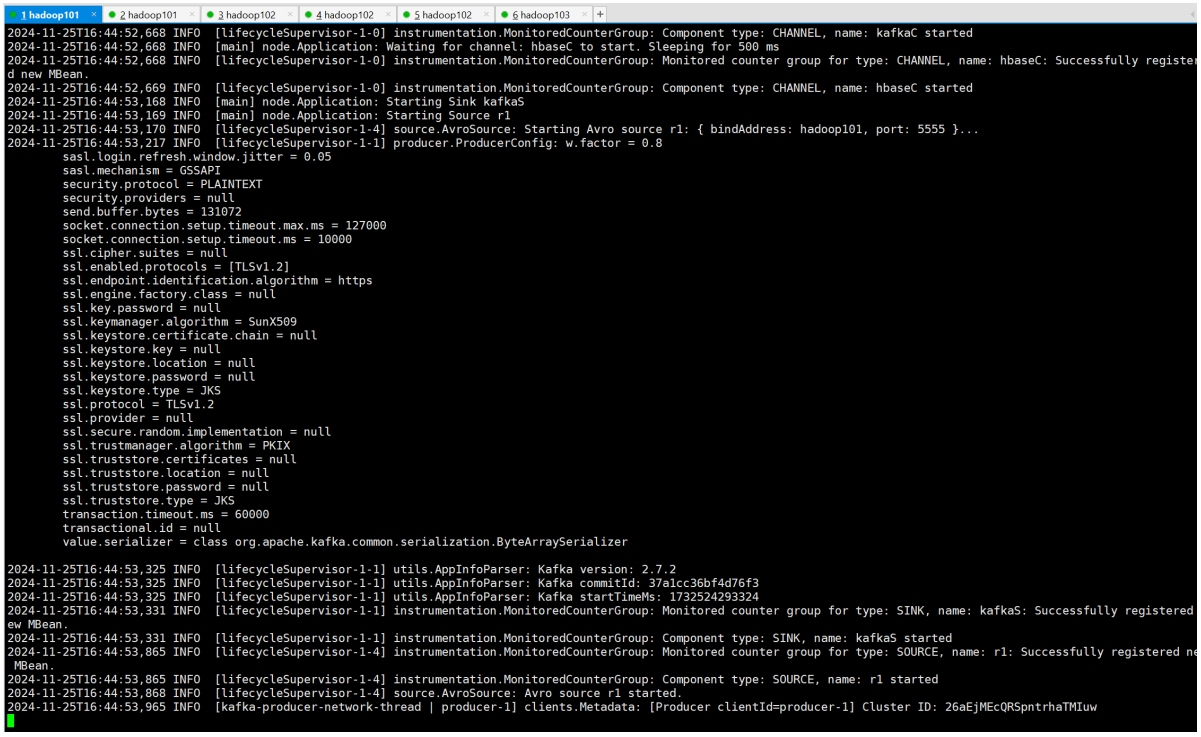
显示Web

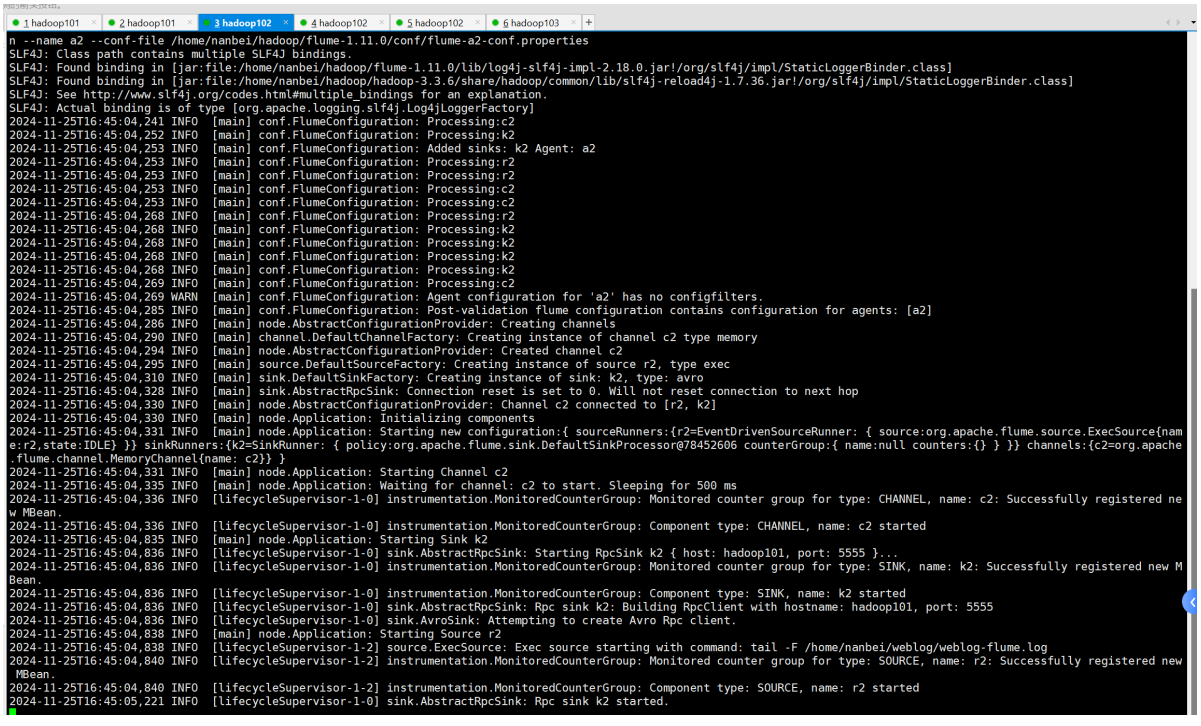
```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>搜狗热门话题数据统计</title>
  <link href="css/style.css" rel="stylesheet">
  <script src="https://cdn.jsdelivr.net/npm/echarts@5.3.2/dist/echarts.min.js">
</script>
</head>
<body>

<h1 class="text-center">搜狗热门话题数据统计</h1>

<!-- GitHub 链接 -->
<div class="github-link-container">
  <a href="https://github.com/NanBei090/News-Real-time-Statistical-Analysis-
System.git" target="_blank" class="github-link">GitHub 仓库</a>
</div>

<!-- 小标题和柱状图 -->
<h2 class="section-title">话题曝光量柱状图</h2>
<div id="titleChart" class="chart-container"></div>

<!-- 小标题和仪表盘 -->
<h2 class="section-title">话题曝光量仪表盘</h2>
<div id="topicGauge" class="chart-container"></div>

<!-- 小标题和表格 -->
<h2 class="section-title">话题曝光量数据表</h2>
<table class="table table-bordered">
  <thead>
  <tr>
    <th>标题</th>
    <th>曝光量</th>
  </tr>
  </thead>
  <tbody id="titleTableBody"></tbody>
</table>

<!-- 作者信息 -->
<footer class="footer">
  <p>作者: nanbei</p>
  <p>创作时间：2024-11-24</p>
</footer>

<script src="js/chart.js"></script>
</body>
</html>
```

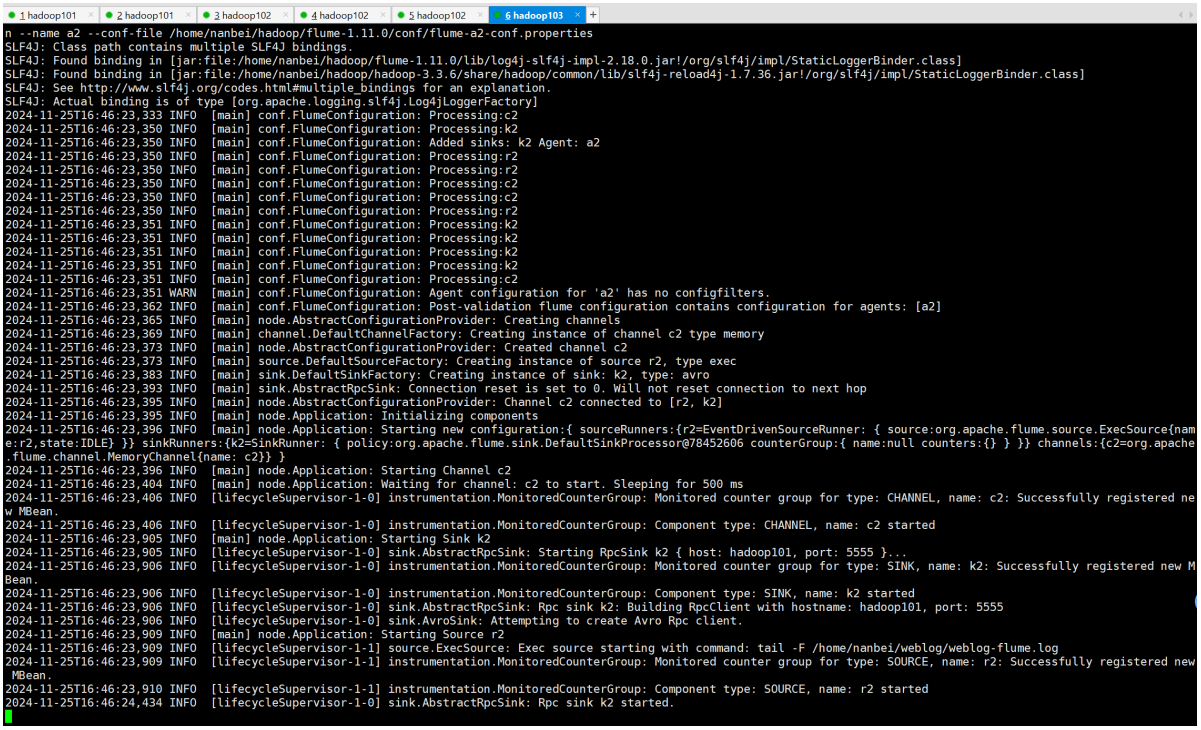
启动项目

启动脚本

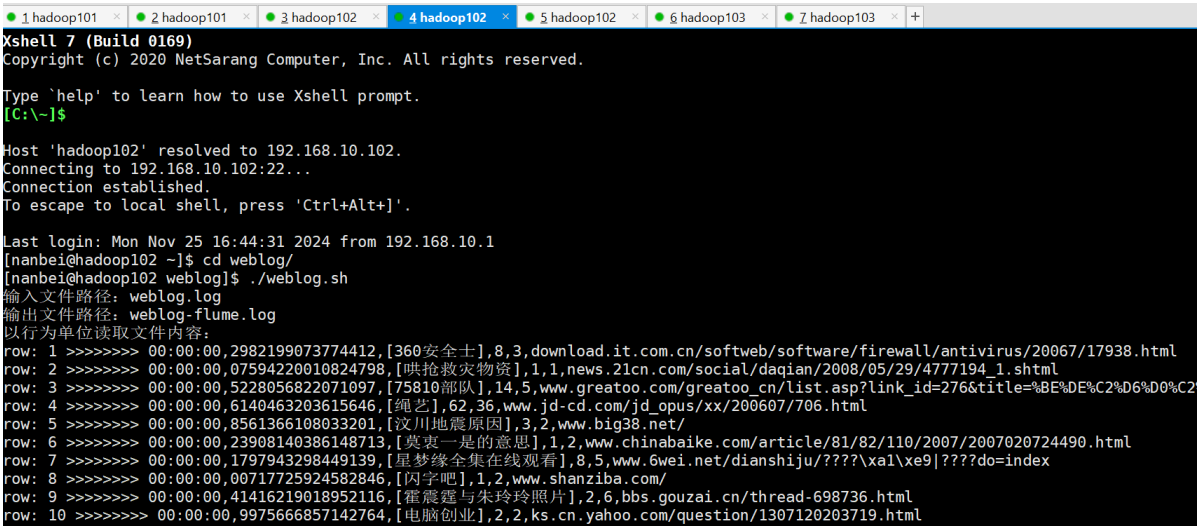
1. 在hadoop101上启动flume.sh脚本，接着在hadoop102，hadoop103上启动flume.sh脚本，结果如下：



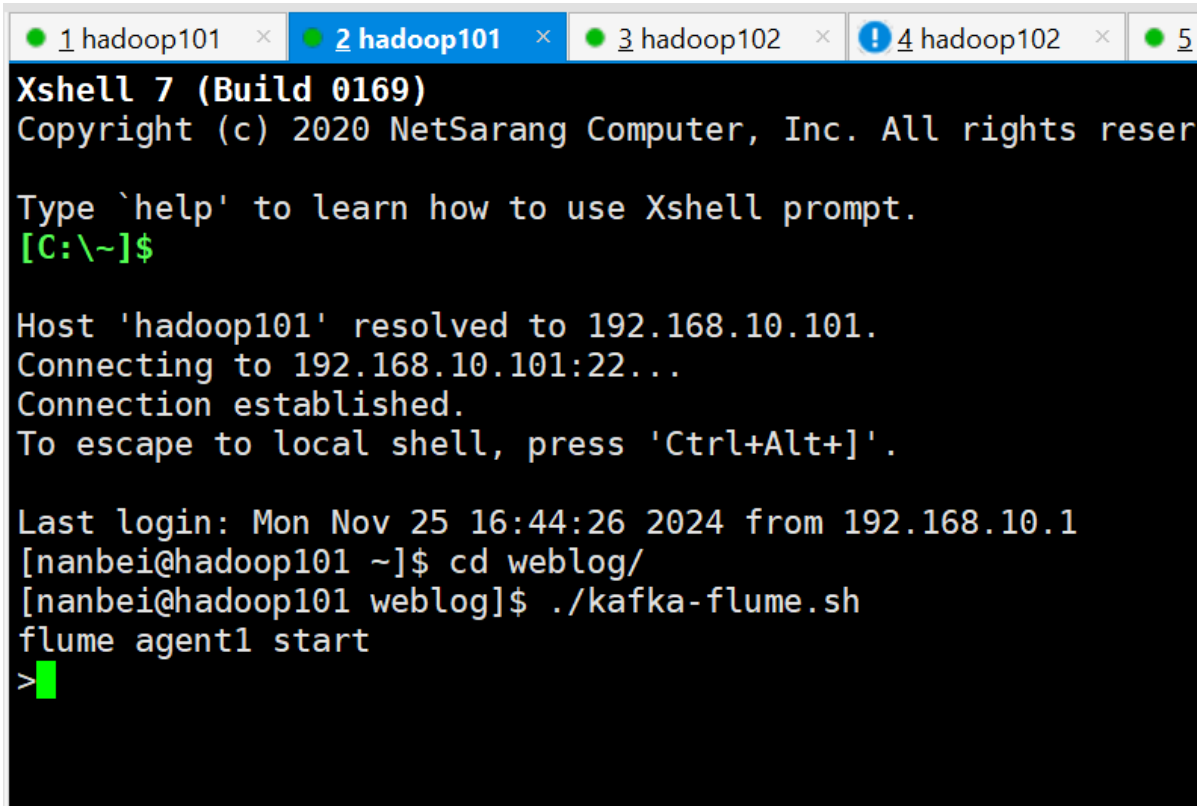


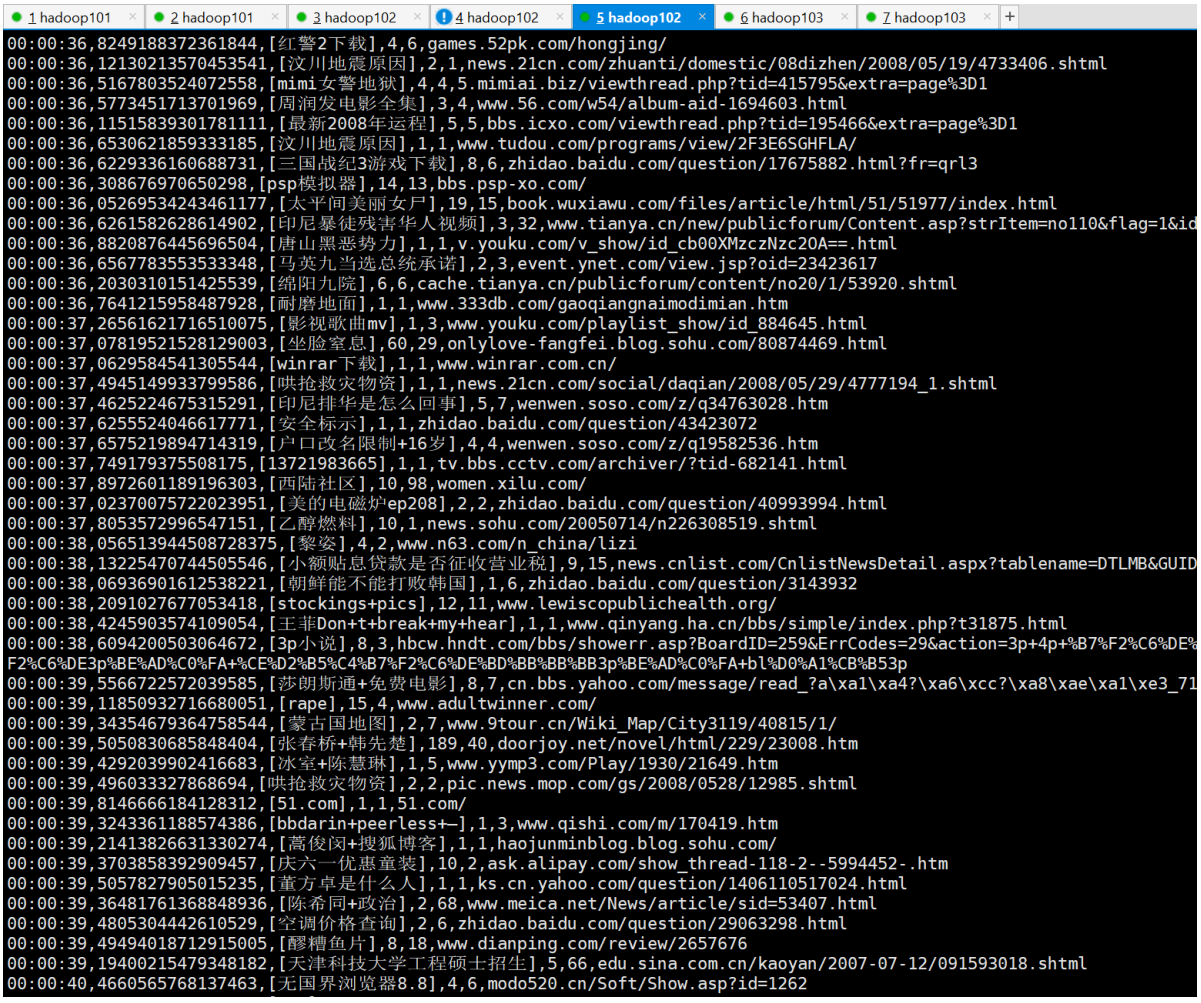


2. 在hadoop102，hadoop103两台机器上启动weblog.sh，结果如下：



3. 先在hadoop101上启动kafka-flume.sh，然后在hadoop102上启动kafka-flume.sh，结果如下：



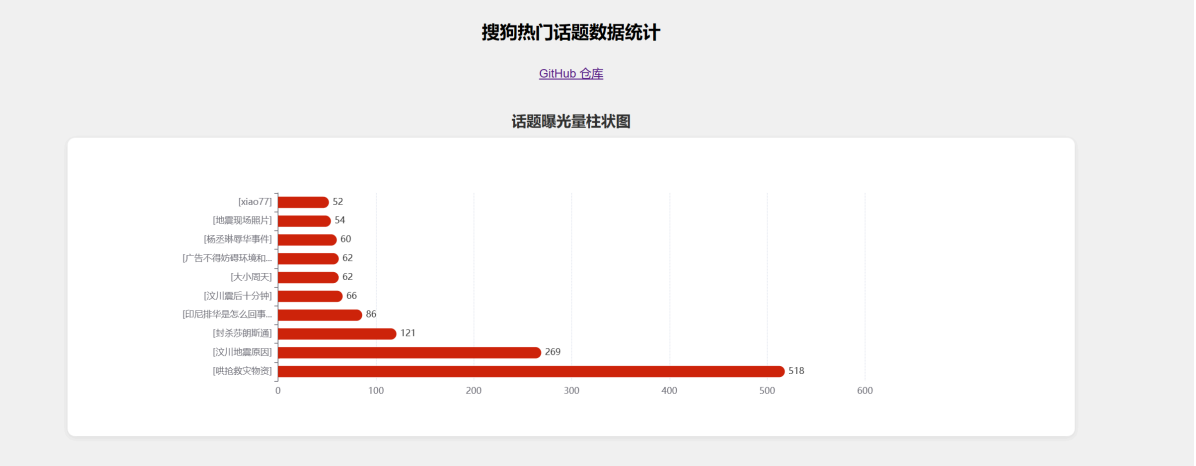


运行项目

首先运行KafkaToMySQL类，结果如下:

```
Data successfully saved to database: 00:03:47,956441490796855,[07式军装],3,2,news.xinhuanet.com/ziliao/2004-07/02/content_1564016.htm
Consumed message: 00:03:47,12530900767865621,[李丽珍],2,1,you.video.sina.com.cn/b/1148681-1256864880.html
Data successfully saved to database: 00:03:47,12530900767865621,[李丽珍],2,1,you.video.sina.com.cn/b/1148681-1256864880.html
Consumed message: 00:03:47,12589157645625676,[四川地震感触],2,34,www.ltzw.com/zuowen/person/90001-93000/92214/65214.htm
Data successfully saved to database: 00:03:47,12589157645625676,[四川地震感触],2,34,www.ltzw.com/zuowen/person/90001-93000/92214/65214.h
Consumed message: 00:03:47,9398696812052275,[秦皇岛船员招聘],4,4,www.dalishuishou.net.cn/shipypinfo.asp?id=766
Data successfully saved to database: 00:03:47,9398696812052275,[秦皇岛船员招聘],4,4,www.dalishuishou.net.cn/shipypinfo.asp?id=766
Consumed message: 00:03:47,6316697850521695,[北京房价暴跌],3,9,bbs.bato.cn/viewthread.php?tid=327076
Data successfully saved to database: 00:03:47,6316697850521695,[北京房价暴跌],3,9,bbs.bato.cn/viewthread.php?tid=327076
Consumed message: 00:03:47,20342695659652554,[欧美bt图],15,12,bbs.zhongzhao.com/dispbbs.asp?boardID=73&ID=30933&page=1
Data successfully saved to database: 00:03:47,20342695659652554,[欧美bt图],15,12,bbs.zhongzhao.com/dispbbs.asp?boardID=73&ID=30933&page
Consumed message: 00:03:47,929886964392068,[朝鲜能不能打败韩国],5,5,iask.sina.com.cn/b/10143364.html
Data successfully saved to database: 00:03:47,929886964392068,[朝鲜能不能打败韩国],5,5,iask.sina.com.cn/b/10143364.html
Consumed message: 00:03:47,8442457450869192,[漫画小说免费下载],5,3,www.sxcnw.net/Soft/hua/Index.html
Data successfully saved to database: 00:03:47,8442457450869192,[漫画小说免费下载],5,3,www.sxcnw.net/Soft/hua/Index.html
```

紧接着运行spring项目，结果如下：



话题曝光量数据表

标题	曝光量
[哄抢救灾物资]	518
[汶川地震原因]	269
[封杀莎朗斯通]	121
[印尼排华是怎么回事]	86
[汶川震后十分钟]	66
[大小周天]	62
[广告不得妨碍环境和自然资源保护]	62
[杨丞琳辱华事件]	60
[地震现场照片]	54
[xiao77]	52
[唐山地震]	44
[武林启示录+金铃]	43
[杨丞琳辱华惨痛下场]	41
[QQ空间代码怎么用]	41
[free+chinese+movie]	39
[hp4vc]	38
[儿童孤独症的治疗]	38
[get+in+her+ass]	38
[08年手机市场分析]	36
[国务院最近任免人员]	35

作者: nanbei

创作时间：2024-11-24