# 通话日志项目

## 项目总体架构

|  |
| --- |
|  |

## 数据源生成程序

|  |
| --- |
| 日志格式:**主叫,被叫,时间,时长**  15810092493,18301589432,20181031115451,204  Java开发 日志生成的模拟程序:  主类:  public class App {  private static Map<String,String> *callers* = new HashMap<String, String>();  private static List<String> *phones* = new ArrayList<String>();  private static Random *random* = new Random();  static {  *callers*.put(**"15810092493"**, **"史玉柱"**);  *callers*.put(**"18000696806"**, **"赵薇"**);  *callers*.put(**"15151889601"**, **"张国立 "**);  *callers*.put(**"13269361119"**, **"王刚"**);  *callers*.put(**"15032293356"**, **"张三"**);  *callers*.put(**"17731088562"**, **"张艺兴"**);  *callers*.put(**"15338595369"**, **"李四"**);  *callers*.put(**"15733218050"**, **"杜文泽"**);  *callers*.put(**"15614201525"**, **"任我行"**);  *callers*.put(**"15778423030"**, **"梁静茹"**);  *callers*.put(**"18641241020"**, **"郭美美"**);  *callers*.put(**"15732648446"**, **"刘诗诗"**);  *callers*.put(**"13341109505"**, **"段玉"**);  *callers*.put(**"13560190665"**, **"唐国强"**);  *callers*.put(**"18301589432"**, **"杨烁"**);  *callers*.put(**"13520404983"**, **"温碧霞"**);  *callers*.put(**"18332562075"**, **"朱元璋"**);  *callers*.put(**"18620192711"**, **"刘能"**);  *phones*.addAll(*callers*.keySet());  }   public static void main(String[] args) {  if(args == null || args.length == 0){  System.*out*.println(**"no args"**);  System.*exit*(-1);  }else{  try {  *genCallLog*(args[0]);  } catch (Exception e) {  e.printStackTrace();  }  }   }  private static void genCallLog(String logFile) throws Exception {  FileWriter fw = new FileWriter(logFile,true);  while (true){  String callerId = *phones*.get(*random*.nextInt(*callers*.size()));  String callerName = *callers*.get(callerId);  String calleeId = *phones*.get(*random*.nextInt(*callers*.size()));  //避免相同号码之间通话  while (callerId.equals(calleeId)){  calleeId = *phones*.get(*random*.nextInt(*callers*.size()));  }  String calleeName = *callers*.get(calleeId);  //通话时长  String duration = UtilTool.*getNumber*(*random*.nextInt(60 \* 10) + 1,**"000"**);  //通话时间  Date date = UtilTool.*randomDate*(**"20180101000000"**,**"20190424000000"**);  String callTime = UtilTool.*dateFormat*(date,**"yyyyMMddHHmmss"**);  //日志  String log = callerId + **","** + calleeId  + **","** + callTime + **","** + duration;  fw.write(log + **"**\r\n**"**);  fw.flush();  }  } }  工具类:  public class UtilTool {  public static String getNumber(int number,String formater){  DecimalFormat decimalFormat = new DecimalFormat();  decimalFormat.applyPattern(formater);  return decimalFormat.format(number);  }  public static Date randomDate(String beginDate,String endDate ) throws Exception {  //时间格式化  SimpleDateFormat format = new SimpleDateFormat(**"yyyyMMddHHmmss"**);  //定义开始时间  Date start = format.parse(beginDate);  //定义结束时间  Date end = format.parse(endDate);  if(start.getTime() >= end.getTime()){  return null;  }  long date = *random*(start.getTime(),end.getTime());  return new Date(date);  }   private static long random(long begin,long end){  long rtn = begin + (long)(Math.*random*() \* (end - begin));  //如果返回的是开始时间和结束时间，通过递归调用本函数查找随机值  if(rtn == begin || rtn == end){  return *random*(begin,end);  }  return rtn;  }   public static String dateFormat(Date date,String formater){  SimpleDateFormat sdf = new SimpleDateFormat();  sdf.applyPattern(formater);  return sdf.format(date);   } }  程序打jar包后在相应服务器(master,master2节点)上启动:  编写shell:  #!/bin/bash  filepath=$1  java -cp /root/CalllogGen-1.0-SNAPSHOT.jar com.space.callloggen.App /root/calllog.log |

## 启动zookeeper/kafka

|  |
| --- |
| slave1,slave2,slave3 节点分别启动zk 以及kafka  /opt/soft/zookeeper-3.4.10/bin/zkServer.sh start  nohup kafka-server-start.sh /opt/soft/kafka\_2.11-2.1.1/config/server.properties &  创建kafka 主题: calllog,三个副本,4个分区  kafka-topics.sh --create --zookeeper slave1:2181 --replication-factor 3 --partitions 4 --topic calllog  查看主题:  kafka-topics.sh --list --zookeeper slave2:2181  启动一个消费者客户端等待flume配置好后,测试是否能够接受到消息:  kafka-console-consumer.sh --bootstrap-server slave1:2181 --topic calllog |

## 配置flume

|  |
| --- |
| Master节点上修改 flume配置文件,需要配置一个exec的source,kafka sink  /opt/soft/apache-flume-1.7.0-bin/conf/calllog.conf  a1.sources = r1  a1.channels = c1  a1.sinks = k1  ##source##  a1.sources.r1.type=exec  a1.sources.r1.command=tail -c +0 -F /root/calllog/calllog.log  ##channel##  a1.channels.c1.type=memory  ##kafka sink##  a1.sinks.k1.type=org.apache.flume.sink.kafka.KafkaSink  a1.sinks.k1.kafka.topic = calllog  a1.sinks.k1.kafka.bootstrap.servers = slave1:9092 slave2:9092 slave3:9092  a1.sinks.k1.kafka.flumeBatchSize = 20  a1.sinks.k1.kafka.producer.acks = 1  ##关联##  a1.sources.r1.channels=c1  a1.sinks.k1.channel=c1  启动fluem测试,查看kafka消费者是否能接受到消息.  flume-ng agent -n a1 -c conf -f apache-flume-1.7.0-bin/conf/calllog.conf &  Flume收集日志的节点:master,master2  拷贝flume目录到master2上,并配置环境变量 |

## 编写kafka消费者

### 启动hadoop集群

|  |
| --- |
| 启动hadoop集群(master/master2:namenode,slave1/slave2/slave3:datanode)  start-hdfs.sh  查看namenode节点状态:##standby/active  hdfs haadmin -getServiceState nn1  hdfs haadmin -getServiceState nn2  使用如下命令切换激活节点(容灾切换):将nn2切换为standby  hdfs haadmin -failover nn2 nn1  Web UI: http://master:50070 |

### 启动hbase集群

|  |
| --- |
| Hmaster:master/master2  HRegionServer:slave1/slave2/slave3  start-hbase.sh  在另一台hmaster手动启动master节点:  hbase-daemon.sh start master  Web UI: <http://master:16010>  创建hbase名称空间和表:  create\_namespace 'ns1'  create 'ns1:calllogs','f1' |

### 创建kafka消费者

|  |
| --- |
| 使用JAVA编写kafka消费者订阅 calllog topic  添加kafka/hbase依赖:  <dependency>  <groupId>org.apache.kafka</groupId>  <artifactId>kafka\_2.12</artifactId>  <version>2.1.1</version> </dependency> <dependency>  <groupId>org.apache.hbase</groupId>  <artifactId>hbase-client</artifactId>  <version>1.2.9</version> </dependency>  开发写入hbase的Dao类:  public class HbaseDao {   private Table table = null;  private DecimalFormat decimalFormat = new DecimalFormat();  private int partitions = 0;  private String caller\_flag;  public HbaseDao(){  Configuration conf = HBaseConfiguration.*create*();  Connection connection = null;  try {  conf.set(**"hbase.zookeeper.quorum"**,PropertiesUtil.*getProp*(**"zkServer"**));  connection = ConnectionFactory.*createConnection*(conf);  TableName tableName = TableName.*valueOf*(PropertiesUtil.*getProp*(**"hbase.table"**));  table = connection.getTable(tableName);  decimalFormat.applyPattern(PropertiesUtil.*getProp*(**"region.format"**));  partitions = Integer.*parseInt*(PropertiesUtil.*getProp*(**"hbase.partitions"**));  caller\_flag = PropertiesUtil.*getProp*(**"caller.flag"**);  } catch (IOException e) {  e.printStackTrace();  }    }  public void put(String log){  try {  if(log == null || **""**.equals(log)){  return;  }  String[] args = log.split(**","**);  if(log != null && args.length == 4){  String caller = args[0];  String callee = args[1];  String callTime = args[2];  String callDucation = args[3];  String hashcode = getHashcode(caller,callTime);  String rowkey = getRowKey(hashcode,caller,callTime,caller\_flag,callee,callDucation);   Put put = new Put(Bytes.*toBytes*(rowkey));  byte[] f1 = Bytes.*toBytes*(**"f1"**);  put.addColumn(f1,Bytes.*toBytes*(**"caller"**),Bytes.*toBytes*(caller));  put.addColumn(f1,Bytes.*toBytes*(**"callee"**),Bytes.*toBytes*(callee));  put.addColumn(f1,Bytes.*toBytes*(**"callTime"**),Bytes.*toBytes*(callTime));  put.addColumn(f1,Bytes.*toBytes*(**"callDucation"**),Bytes.*toBytes*(callDucation));  table.put(put);  }  } catch (IOException e) {  e.printStackTrace();  }finally {  try {  table.close();  } catch (IOException e) {  e.printStackTrace();  }  }   }   public String getHashcode(String caller,String callTime){  int len = caller.length();  //取主叫后4为  String caller\_last4 = caller.substring(len - 4);  //时间前6位  String callTime\_start6 = callTime.substring(0,6);  int hashcode = (Integer.*parseInt*(callTime\_start6) ^ Integer.*parseInt*(caller\_last4)) % partitions;  return decimalFormat.format(hashcode);  }   public String getRowKey(String hashcode,String caller,String callTime  ,String flag,String callee,String duration){  return hashcode + **","** + caller + **","** + callTime + **","** + flag + **","** + callee + **","** + duration;  }  }    Util工具类:  public class PropertiesUtil {  public static Properties *properties*;  static {  *properties* = new Properties();  try {  *properties*.load(ClassLoader.*getSystemResourceAsStream*(**"kafka.properties"**));  } catch (IOException e) {  e.printStackTrace();  }  }   public static String getProp(String key){  return *properties*.getProperty(key);  } }  Kafka消费者类:  public class HbaseConsumer {  public static void main(String[] args) {  HbaseDao hbaseDao = new HbaseDao();  try {  KafkaConsumer<String, String> kafkaConsumer = new KafkaConsumer<String, String>(PropertiesUtil.*properties*);  String topic = PropertiesUtil.*getProp*(**"topic"**);  kafkaConsumer.subscribe(Arrays.*asList*(topic));  String logStr = null;  while (true) {  ConsumerRecords<String, String> records = kafkaConsumer.poll(Duration.*ofHours*(7));  for (ConsumerRecord<String, String> record : records) {  logStr = record.value();  //写入hbase  hbaseDao.put(logStr);   }  }  } catch (Exception e) {  e.printStackTrace();  }  }  }  Kafka,hbase等相关配置文件 kafka.properties  bootstrap.servers=**slave1:9092,slave2:9092,slave3:9092** group.id=**group-20190425** auto.offset.reset=**earliest** session.timeout.ms=**30000** key.deserializer=**org.apache.kafka.common.serialization.StringDeserializer** value.deserializer=**org.apache.kafka.common.serialization.StringDeserializer** topic=**calllog** zkServer=**slave1:2181,slave2:2181,slave3:2181** ##表名 hbase.table=**ns1:calllogs** ##分区数 hbase.partitions=**100** ##主叫标记 caller.flag=**0** ##分区个数格式化 region.format=**00**  由于kafka消费者类依赖 kafka和 hbase相关jar包,所以运行需要将依赖jar包也导出  导出依赖jar包:  mvn  -DoutputDirectory=./lib  -DgroupId=com.space.consumer  -DartifactId=CalllogConsumer  -Dversion=1.0-SNAPSHOT  dependency:copy-dependencies  将第三方jar包导入 当前目录 lib下  将工程打包:CalllogConsumer-1.0-SNAPSHOT.jar,复制jar包,lib/\*.jar,和kafka.properties 文件到 相关节点 如下目录:  conf  :kafka.properties  consumer.jar  lib  :\*.jar  编写 shell:  #!/bin/bash  java -cp consumer/lib/\*:consumer/conf/:consumer/consumer.jar com.space.consumer.HbaseConsumer |

## 可视化简单实现

|  |
| --- |
| 编写前端可视化:SSM  查询逻辑为 如: 号码:15899999999,起始时间:2019-01-01,结束时间:2019-04-26  由于kafka 消费者 存入hbase的 分区号是 号码后四位和时间前6位,因此同月同号码的数据会连续存储在一起.  因此查询时 如:15899999999,2019-01-01到2019-04-26需要如下逻辑:  20190101---201902  201902------201903  201903------201904  201904------20190427  以每一月段去 得到 startRowkey,stopRowkey,两个rowkey都以起始时间获取区域号,否则如果跨月如:201912------202001,,计算的**区域号**就不同.  取出每月的数据合起来返回给前端  需要给月加1,后判断是否大于或等于 结束日期  注意区域号的计算  页面主要查询代码  <%--  Created by IntelliJ IDEA.  User: lucifel  Date: 19-4-4  Time: 下午10:57  To change this template use File | Settings | File Templates. --%> <%@ **page** contentType="**text/html;charset=UTF-8**" language="**java**" %> <%@**taglib** prefix="**c**" uri="**http://java.sun.com/jsp/jstl/core**" %> <html> <head>  <title>通话记录</title>  <link rel="stylesheet" type="text/css" href="/js/themes/default/easyui.css">  <link rel="stylesheet" type="text/css" href="/js/themes/icon.css">  <script type="text/javascript" src="/js/jquery.min.js"></script>  <script type="text/javascript" src="/js/jquery.easyui.min.js"></script> </head> <body>  <div style="">  <form id="form1" method="post">  <div>  <label for="caller">电话号码:</label>  <input class="easyui-textbox" type="text" name="caller" data-options="required:true" />   <label for="sTime">起始时间:</label>  <input name="sTime" class="easyui-datebox" data-options="required:true"/>   <label for="eTime">结束时间:</label>  <input name="eTime" class="easyui-datebox" data-options="required:true"/>   <a id="btn" href="#" class="easyui-linkbutton" onclick="searchLog()"  data-options="iconCls:'icon-search'">查询</a>  </div>  </form>  <table id="dg" title="通话日志查询" style="width:810px;height:350px;">  </table>  </div> <script>  $(**'#dg'**).datagrid({  singleSelect:**true**,  collapsible:**true**,  rownumbers:**true**,  loadMsg:**'正在加载数据...'**,  striped:**true**,  emptyMsg:**'暂无数据'**,  data: [  ],  columns:[[  {field:**'caller'**,title:**'电话1'**,width:170,resizable:**false**},  {field:**'flag'**,title:**'主/被叫'**,width:80,resizable:**false**,  formatter: **function**(value,row,index){  **if** (row.flag){  **return "主叫"**;  } **else** {  **return "被叫"**;  }  },  align:**'center'** },  {field:**'callee'**,title:**'电话2'**,width:170,resizable:**false**},  {field:**'callTime'**,title:**'通话时间'**,width:170,align:**'right'**,resizable:**false**},  {field:**'callduration'**,title:**'通话时长(s/秒)'**,width:170,align:**'right'**,resizable:**false**}  ]]  });   **function** searchLog2() {  $.ajax({  type: **"POST"**,  dataType: **"json"**,  url: **"/calllog/findAll"** ,  data: $(**'#form1'**).serialize(),  success: **function** (data) {  $(**'#dg'**).datagrid({ data:data});  },  error : **function**() {  alert(**"查询出错"**);  }  });  }  **function** searchLog() {  $.messager.progress();  $(**'#form1'**).form(**'submit'**, {  url:**"/calllog/findLogs"**,  onSubmit: **function**(param){  **var** isValid = $(**this**).form(**'validate'**);  **if** (!isValid){  $.messager.progress(**'close'**); // hide progress bar while the form is invalid  }  **return** isValid;  },  success:**function**(data){  **var** data = eval(**'('** + data + **')'**); // change the JSON string to javascript object  $.messager.progress(**'close'**);  $(**'#dg'**).datagrid({  data:data  });  }  });  }   $.fn.datebox.defaults.formatter = **function**(date){  **var** y = date.getFullYear();  **var** m = date.getMonth()+1;  **var** d = date.getDate();  **return** y+**'/'**+m+**'/'**+d;  }   $.fn.datebox.defaults.parser = **function**(s){  **var** t = Date.parse(s);  **if** (!isNaN(t)){  **return new** Date(t);  } **else** {  **return new** Date();  }  } </script> </body> </html>  注意后端需要返回JSON字符串 需要 加入JSON相关依赖  主要查询代码:  public List<CallLog> findlogs(String ...args) {  List<CallLog> result = new ArrayList<CallLog>();  try {  String caller\_no = args[0];  Date startDate = Util.*parseToDate*(args[1],**"yyyy/MM/dd"**);  Date endDate = Util.*parseToDate*(args[2],**"yyyy/MM/dd"**);  Date \_endDate = Util.*getAfterSomeMon*(startDate,1);   String sTime = Util.*dateFormat*(startDate,**"yyyyMMdd"**);  String eTime = Util.*dateFormat*(\_endDate,**"yyyyMM"**);  while (\_endDate.getTime() < endDate.getTime()){  String start\_rk = Util.*getHashcode*(caller\_no,sTime,100) + **","** + caller\_no + **","** + sTime;  String end\_rk = Util.*getHashcode*(caller\_no,sTime,100) + **","** + caller\_no + **","** + eTime;  result.addAll(getEveryMonLogs(start\_rk,end\_rk));  sTime = eTime;  \_endDate = Util.*getAfterSomeMon*(\_endDate,1);  eTime = Util.*dateFormat*(\_endDate,**"yyyyMM"**);  }  endDate = Util.*getAfterSomeDay*(endDate,1);  eTime = Util.*dateFormat*(endDate,**"yyyyMMdd"**);  String start\_rk = Util.*getHashcode*(caller\_no,sTime,100) + **","** + caller\_no + **","** + sTime;  String end\_rk = Util.*getHashcode*(caller\_no,sTime,100) + **","** + caller\_no + **","** + eTime;  result.addAll(getEveryMonLogs(start\_rk,end\_rk));  } catch (Exception e) {  e.printStackTrace();  }  return result; }  public List<CallLog> getEveryMonLogs(String start\_rk,String end\_rk){  List<CallLog> list = new ArrayList<CallLog>();  try {  Scan scan = new Scan();  ResultScanner scanner = null;  byte[] f1 = Bytes.*toBytes*(**"f1"**);  byte[] \_caller = Bytes.*toBytes*(**"caller"**);  byte[] \_callee = Bytes.*toBytes*(**"callee"**);  byte[] \_callTime = Bytes.*toBytes*(**"callTime"**);  byte[] \_callDucation = Bytes.*toBytes*(**"callDucation"**);  CallLog callLog = null;  scan.setStartRow(Bytes.*toBytes*(start\_rk));  scan.setStopRow(Bytes.*toBytes*(end\_rk));  scanner = table.getScanner(scan);  Iterator<Result> iterator = scanner.iterator();  while (iterator.hasNext()){  callLog = new CallLog();  Result result = iterator.next();  String rowkey = Bytes.*toString*(result.getRow());  String flag = rowkey.split(**","**)[3];  callLog.setFlag(**"0"**.equals(flag)?true:false);  byte[] caller = result.getValue(f1, \_caller);  byte[] callee = result.getValue(f1, \_callee);  byte[] callTime = result.getValue(f1, \_callTime);  byte[] callDucation = result.getValue(f1, \_callDucation);  callLog.setCaller(Bytes.*toString*(caller));  callLog.setCallee(Bytes.*toString*(callee));  Date temp = Util.*parseToDate*(Bytes.*toString*(callTime),**"yyyyMMddHHmmss"**);  String timeTemp = Util.*dateFormat*(temp,**"yyyy-MM-dd HH:mm:ss"**);  callLog.setCallTime(timeTemp);  callLog.setCallduration(Bytes.*toString*(callDucation));  list.add(callLog);  }  } catch (Exception e) {  e.printStackTrace();  }finally {  try {  table.close();  } catch (IOException e) {  e.printStackTrace();  }  }  return list; } |

## 实现Hbase协处理器,插入被叫标记的数据

|  |
| --- |
| 添加依赖:  <dependency>  <groupId>org.apache.hbase</groupId>  <artifactId>hbase-server</artifactId>  <version>1.2.9</version> </dependency>  协处理器:  public class CalllogRegionObserver extends BaseRegionObserver {  **@Override** public void postPut(ObserverContext<RegionCoprocessorEnvironment> e, Put put, WALEdit edit, Durability durability) throws IOException {  super.postPut(e, put, edit, durability);  //rowkey-caller格式:xx,callerId,callTime,flag[0/1],calleeId,duration  //rowkey-callee格式:xx,calleeId,callTime,flag[0/1],callerId,duration  super.postPut(e, put, edit, durability);  TableName tableName = e.getEnvironment().getRegion().getRegionInfo().getTable();  TableName calllogs = TableName.*valueOf*(**"ns1:calllogs"**);  String rowkey\_old = Bytes.*toString*(put.getRow());  String[] arr = rowkey\_old.split(**","**);  String flag = arr[3];  if(!calllogs.getNameAsString().equals(tableName.getNameAsString())){  return;  }else if(**"0"**.equals(flag)){//主叫时,添加一条被叫的记录  String callee = arr[4];  String caller = arr[1];  String callTime = arr[2];  String duration = arr[5];  //区域号  String hash = CalllogUtil.*getHashcode*(callee,callTime,100);  String rowkey\_new = CalllogUtil.*getRowKey*(hash,callee,callTime,**"1"**,caller,duration);  Put put1 = new Put(Bytes.*toBytes*(rowkey\_new));  put1.addColumn(Bytes.*toBytes*(**"f2"**),Bytes.*toBytes*(**"refrowid"**),Bytes.*toBytes*(rowkey\_old));  Table t = e.getEnvironment().getTable(calllogs);  t.put(put1);  }else{  return;  }   } }  动态注册:  disable 'ns1:calllogs'  alter 'ns1:calllogs',METHOD =>'table\_att','coprocessor'=>'hdfs:///hbase\_jar/calllog\_coprocessor2.jar|com.space.coprocessor.CalllogRegionObserver|1001|'  enable 'ns1:calllogs' |

### 协处理器处理被叫数据获取问题

由于为1的数据是通过协处理器put的,只存储了对应为0的rowkey,所有查询时需要通过此rowkey获取对应的值,重写协处理器的postScannerNext和postGetOp方法

不管是scan或 get 符合标准的数据都会 重新获取:

Rowkey:99,15732648446,20180508051139,**0**,13341109505,423

列族-列:f1:caller,callee,callTime....

################################插入上边数据时会插入一条下边数据

60,13341109505,20180508051139,**1**,15732648446,423 f2:refrowid,value=99,15732648446,20180508051139,**0**,13341109505,423

这条数据没有存储其他列值,只存储了原数据的rowkey,所以查询这条数据时应该去找对应原数据

|  |
| --- |
| **public** **class** CallLogRegionObserver **extends** BaseRegionObserver {        //被叫引用id  **private** **static** **final** String REF\_ROW\_ID = "refrowid" ;      //通话记录表名  **private** **static** **final** String CALL\_LOG\_TABLE\_NAME = "ns1:calllogs" ;        /\*\*       \* Put后处理       \*/  **public** **void** postPut(ObserverContext<RegionCoprocessorEnvironment> e, Put put, WALEdit edit, Durability durability) **throws** IOException {  **super**.postPut(e, put, edit, durability);          //          String tableName0 = TableName.valueOf(CALL\_LOG\_TABLE\_NAME).getNameAsString();            //得到当前的TableName对象          String tableName1 = e.getEnvironment().getRegion().getRegionInfo().getTable().getNameAsString();          //判断是否是ns1:calllogs表  **if** (!tableName0.equals(tableName1)) {  **return**;          }            //得到主叫的rowkey,          String rowkey = Bytes.toString(put.getRow());            //如果被叫就放行          String[] arr = rowkey.split(",");  **if** (arr[3].equals("1")) {  **return**;          }          //hashcode,caller,time,flag,callee,duration          String caller = arr[1] ;        //主叫          String callTime = arr[2] ;      //通话时间          String callee = arr[4] ;        //被叫          String callDuration = arr[5] ;  //通话时长            //被叫hashcode          String hashcode = CallLogUtil.getHashcode(callee,callTime,100);          //被叫rowkey          String calleeRowKey = hashcode + "," + callee + "," + callTime + ",1," + caller + "," + callDuration;          Put newPut = **new** Put(Bytes.toBytes(calleeRowKey));          newPut.addColumn(Bytes.toBytes("f2"), Bytes.toBytes(REF\_ROW\_ID), Bytes.toBytes(rowkey));          TableName tn = TableName.valueOf(CALL\_LOG\_TABLE\_NAME);          Table t = e.getEnvironment().getTable(tn);          t.put(newPut);      }        /\*\*       \* 重写方法，完成被叫查询，返回主叫结果。       \*/  **public** **void** postGetOp(ObserverContext<RegionCoprocessorEnvironment> e, Get get, List<Cell> results) **throws** IOException {          //获得表名          String tableName = e.getEnvironment().getRegion().getRegionInfo().getTable().getNameAsString();            //判断表名是否是ns1:calllogs  **if**(!tableName.equals(CALL\_LOG\_TABLE\_NAME)){  **super**.preGetOp(e, get, results);          }  **else**{              //得到rowkey              String rowkey = Bytes.toString(get.getRow());              //              String[] arr = rowkey.split(",");              //主叫  **if**(arr[3].equals("0")){  **super**.postGetOp(e, get, results);              }              //被叫  **else**{                  //得到主叫方的rowkey                  String refrowid = Bytes.toString(CellUtil.cloneValue(results.get(0)));                  //                  Table tt = e.getEnvironment().getTable(TableName.valueOf(CALL\_LOG\_TABLE\_NAME));                  Get g = **new** Get(Bytes.toBytes(refrowid));                  Result r = tt.get(g);                  List<Cell> newList = r.listCells();                  results.clear();                  results.addAll(newList);              }          }      }        /\*\*       \*       \*/  **public** **boolean** postScannerNext(ObserverContext<RegionCoprocessorEnvironment> e, InternalScanner s, List<Result> results, **int** limit, **boolean** hasMore) **throws** IOException {  **boolean** b = **super**.postScannerNext(e, s, results, limit, hasMore);            //新集合          List<Result> newList = **new** ArrayList<Result>();            //获得表名          String tableName = e.getEnvironment().getRegion().getRegionInfo().getTable().getNameAsString();            //判断表名是否是ns1:calllogs  **if** (tableName.equals(CALL\_LOG\_TABLE\_NAME)) {              Table tt = e.getEnvironment().getTable(TableName.valueOf(CALL\_LOG\_TABLE\_NAME));  **for**(Result r : results){                  //rowkey                  String rowkey = Bytes.toString(r.getRow());                  String flag = rowkey.split(",")[3] ;                  //主叫  **if**(flag.equals("0")){                      newList.add(r) ;                  }                  //被叫  **else**{                      //取出主叫号码  **byte**[] refrowkey = r.getValue(Bytes.toBytes("f2"),Bytes.toBytes(REF\_ROW\_ID)) ;                      Get newGet = **new** Get(refrowkey);                      newList.add(tt.get(newGet));                  }              }              results.clear();              results.addAll(newList);          }  **return** b ;      }  } |

Web中的control:

|  |
| --- |
| **@ResponseBody @RequestMapping**(value = **"/calllog/findLogs"**) public Map<String,Object> findLogs(**@RequestParam**(**"caller"**) String caller  ,**@RequestParam**(**"sTime"**) String sTime,**@RequestParam** String eTime){  List<CallLog> logs = calllogService.findlogs(caller,sTime,eTime);  Map<String,Object> map = new HashedMap();  map.put(**"callNo"**,caller);  map.put(**"logs"**,logs);  return map; } |

页面:

|  |
| --- |
| **function** searchLog() {  $.messager.progress();  $(**'#form1'**).form(**'submit'**, {  url:**"/calllog/findLogs"**,  onSubmit: **function**(param){  **var** isValid = $(**this**).form(**'validate'**);  **if** (!isValid){  $.messager.progress(**'close'**); // hide progress bar while the form is invalid  }  **return** isValid;  },  success:**function**(data){  **var** data = eval(**'('** + data + **')'**); // change the JSON string to javascript object  caller = data.callNo;  $.messager.progress(**'close'**);  $(**'#dg'**).datagrid({  data:data.logs  });  } }); }  **var** caller; $(**'#dg'**).datagrid({  singleSelect:**true**,  collapsible:**true**,  rownumbers:**true**,  loadMsg:**'正在加载数据...'**,  striped:**true**,  emptyMsg:**'暂无数据'**,  data: [  ],  columns:[[  {field:**'caller'**,title:**'电话1'**,width:170,resizable:**false**},  {field:**'callee'**,title:**'电话2'**,width:170,resizable:**false**},  {field:**'flag'**,title:**'主/被叫'**,width:80,resizable:**false**,  formatter: **function**(value,row,index){  **if** (row.caller == caller){  **return "主叫"**;  } **else** {  **return "被叫"**;  }  },  align:**'center'** },  {field:**'callTime'**,title:**'通话时间'**,width:170,align:**'right'**,resizable:**false**},  {field:**'callduration'**,title:**'通话时长(s/秒)'**,width:170,align:**'right'**,resizable:**false**}  ]] });  界面如下: |

## 使用hive 查询最近通话记录

|  |
| --- |
| Hive集成 hbase,创建hbase映射外部表 |

### 开启yarn集群

|  |
| --- |
| start.yarn.sh  yarn-daemon.sh start resourcemanager  Web UI:http://master2:8088 |

### 启动hive

|  |
| --- |
| Slave1上启动hiveserver2  Slave2上启动客户端,并创建hbase映射表  beeline -u jdbc:hive2://slave1:10000/default -n root root  Hive中创建关联表:  CREATE **external** TABLE ext\_calllogs\_in\_hbase(id string, caller string,callee string,callTime string,callDucation string) STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'  WITH SERDEPROPERTIES ("hbase.columns.mapping" = ":key,f1:caller,f1:callee,f1:callTime,f1:callDucation")  TBLPROPERTIES ("hbase.table.name" = "ns1:calllogs");  创建一个外部表,:key会自动和第一个字段 id映射为hbase rowkey |

### 可视化实现

|  |
| --- |
| 加入依赖:  <dependency>  <groupId>org.apache.hive</groupId>  <artifactId>hive-jdbc</artifactId>  <version>2.3.4</version> </dependency>  次依赖加入后会和 SSM架构中 jar包产生冲突,需在工件中删除如下:    注意在pom中加入的依赖不要删 如:jstl |

### 添加mysql联合查询

|  |
| --- |
| 通话记录等实时产生的数据放入hbase,但电话相关的人员信息一般可能存储在如:mysql等关系型数据库中,在显示通话记录时需要查出 这些信息  Mysql 中新建表:  create table persons(id int primary key auto\_increment,name varchar(100),phone varchar(20));  在domain中创建Person类,并在Calllog bean中添加相应字段  private Integer id;  private String name;  private String phone; |
|  |

## Hive查询个月通话次数

|  |
| --- |
| select count(\*),substr(calltime,1,6) as ym from ext\_calllogs\_in\_hbase where caller='18620192711' group by ym;  public List<CalllogStat> findCalllogStat(String phone,String year){  List<CalllogStat> statList = new ArrayList<CalllogStat>();  Connection connection = null;  try {  connection = DriverManager.*getConnection*(*url*);  String sql = **"select** *count***(\*) as count,substr(calltime,1,6) as ym from ext\_calllogs\_in\_hbase "** +  **"where caller=? and substr(calltime,1,4) = ? group by substr(calltime,1,6) "**;  PreparedStatement preparedStatement = connection.prepareStatement(sql);  preparedStatement.setString(new Integer(1),phone);  preparedStatement.setString(new Integer(2),year);  ResultSet resultSet = preparedStatement.executeQuery();  CalllogStat calllogStat = null;  while(resultSet.next()){  calllogStat = new CalllogStat();  int count = resultSet.getInt(**"count"**);  String ym = resultSet.getString(**"ym"**);  calllogStat.setCount(count);  calllogStat.setYearMon(ym);  statList.add(calllogStat);  }  resultSet.close();  return statList;   } catch (SQLException e) {  e.printStackTrace();  }  return null; }  使用Echarts作为可视化:  <html> <head>  <title>年度通话次数</title>  <link rel="stylesheet" type="text/css" href="/js/themes/default/easyui.css">  <link rel="stylesheet" type="text/css" href="/js/themes/icon.css">  <script type="text/javascript" src="/js/jquery.min.js"></script>  <script type="text/javascript" src="/js/jquery.easyui.min.js"></script>  <script src="/js/echarts.js"></script> </head> <body>  <div style="">  <form id="form1" method="post">  <div>  <label for="phone">电话号码:</label>  <input class="easyui-textbox" type="text" name="phone" data-options="required:true" />   <label for="year">通话年份:</label>  <input class="easyui-textbox" type="text" name="year" data-options="required:true" />   <a id="btn" href="#" class="easyui-linkbutton" onclick="searchLog()"  data-options="iconCls:'icon-search'">查询</a>  </div>  </form>   </div>  <div id="main" style="width: 600px;height:400px;"></div> <script>  // 基于准备好的dom，初始化echarts实例  **var** myChart = echarts.init(document.getElementById(**'main'**));  // 指定图表的配置项和数据  **var** option = {  tooltip: {},  legend: {  data:[**'通话量'**]  },  xAxis: {  data: []  },  yAxis: {},  series: [{  name: **'通话量'**,  type: **'bar'**,  data: []  }]  };   // 使用刚指定的配置项和数据显示图表。  myChart.setOption(option);   **function** searchLog() {  $.messager.progress();  $(**'#form1'**).form(**'submit'**, {  url:**"/calllog/getCalllogStat"**,  onSubmit: **function**(param){  **var** isValid = $(**this**).form(**'validate'**);  **if** (!isValid){  $.messager.progress(**'close'**); // hide progress bar while the form is invalid  }  **return** isValid;  },  success:**function**(data){  **var** data = eval(**'('** + data + **')'**); // change the JSON string to javascript object  **var** year = data.year;  **var** yearMonth\_counts = data.yearMonth\_counts;  **var** yearMonths = data.yearMonths;  $.messager.progress(**'close'**);  myChart.setOption({  title: {  text: year + **'年度通话量'** },  xAxis: {  data: yearMonths  },  series: [{  // 根据名字对应到相应的系列  data: yearMonth\_counts  }]  });  }  });  }  </script> </body> </html>  如图所示: |