1. P1

|  |
| --- |
| 已采集微博数据结构化（hive）数据从何而来？ 通过数据交易获得，交易到的数据是以文件的形式。 数据格式是什么？ Json格式/xml格式 数据结构化？ 通过交易而来的数据是非结构化的，不方便分析与管理，所以要将这些非结构化的数据结构化到hive中去。 项目的产出？ 产出两张表，给定用户id可以通过hive sql查出用户信息和用户评论信息，  通过hive cli演示效果 数据的输入与输出？  * 输入   输入是两个类别的文件集合。   1. 房地产评论主题下的对应用户基本信息文件集和评论内容文件集，两者通过用户id关联。 2. 通过一定Java程序做数据解析、结构化，将两个文件集合各自合并成一个文件load到hive中。  * 输出   两张表，一个用户表，一个评论内容表   1. 将两个类别下各自对应的文件集进行解析、结构化 2. 通过load语法，将数据分别导入到两张表中  主要思路  1. 用JavaSE+maven完成数据解析、结构化到文本文件。 2. 用hive load语法将数据加载进hive进行管理  技术要点  1. 数据仓库的目录结构和开发规范 2. JavaSE程序设计基础 3. 面向对象程序设计思想 4. Maven项目的构建与开发 5. Xml数据解析  步骤拆解  1. 非结构化数据 -> 结构化数据 80% 2. 将数据加载到hive 10% 3. 效果测试和优化 10% |
| 项目实现  1. 在Eclipse中创建maven项目，名为WbDataToHive； 2. 粘贴之前pom.xml文件的内容（注意修改项目名）； 3. Update maven； 4. 把数据文件夹“房地产”拖到根目录； 5. 开始编写解析数据的工具类：   （见下节.......） |

1. P2

|  |  |
| --- | --- |
| 工具类和业务类的思考：     1. 文件操作工具类（java.util.FileOpratorUtil）   功能：给定文件夹，读出文件夹内所有文件   |  | | --- | | **package** utils;  **import** java.io.File;  **import** java.util.ArrayList;  **import** java.util.List;  /\*\*  \* 指定文件夹，找出文件夹中的文件  \* **@author** Apollo  \*  \*/  **public** **class** FileOperatorUtil {  /\*\*  \* 根据路径，找出其下的所有文件  \*/  **public** **static** List<String> getAllSubNormalFilePath(String filePath){  //1.创建文件类，定位到该路径  File file = **new** File(filePath);  //2.创建List用来存储遍历出的结果集  List<String> resultList = **new** ArrayList();  //3.判断是否是目录  **if**(file.isDirectory()){  //列出该目录下的全部文件  **for**(File tempFile : file.listFiles()){  resultList.addAll(*getAllSubNormalFilePath*(tempFile.toString()));  }  }**else**{  //如果是文件，直接加进list  resultList.add(file.toString());  }  //4.返回结果集  **return** resultList;  }    /\*\*  \* 编写测试方法  \*/  **public** **static** **void** main(String[] args){  String filePath = "房地产";  List<String> res = *getAllSubNormalFilePath*(filePath);  //遍历打印  System.***out***.println("res的个数为："+res.size());  System.***out***.println("------------------------");  **for**(String temp : res){  System.***out***.println(temp);  }    }  } | |
| 1. IO操作工具类  |  | | --- | | **package** utils;  /\*\*  \* 根据文件名称，读取出该文件中的内容  \* **@author** Apollo  \*  \*/  **import** java.io.BufferedReader;  **import** java.io.File;  **import** java.io.FileInputStream;  **import** java.io.InputStreamReader;  **import** java.util.ArrayList;  **import** java.util.List;  **public** **class** IOUtil {  /\*\*  \* 给定文件名，读取出文件内容  \* **@param** args  \*/  **public** **static** List<String> getTxtContent(String txtFilePath,String charset)  **throws** Exception {  //1.新建File类定位文件位置  File txtFile = **new** File(txtFilePath);  //2.文件输入流  FileInputStream fis = **new** FileInputStream(txtFile);  //3.字符输入流  InputStreamReader isr = **new** InputStreamReader(fis,charset);  //4.字节输入流？  BufferedReader br = **new** BufferedReader(isr);    //5.定义结果集  List<String> lineList = **new** ArrayList<String>();    //6.读  String tempLine = **null**;  **while**((tempLine = br.readLine()) != **null**){  lineList.add(tempLine);  }  //7.关流  br.close();    //8.返回  **return** lineList;  }    /\*\*  \* 测试  \* **@param** args  \*/  **public** **static** **void** main(String[] args) **throws** Exception{  String txtFilePath = "房地产\\user\\1107717945.txt";  String charset = "utf-8";  List<String> res = *getTxtContent*(txtFilePath,charset);  **for**(String temp : res){  System.***out***.println(temp);  }    }  } | | 1. 指定编码，将文件重新写出  |  | | --- | | 源文件编码为utf-8  /\*\*  \* 转文件编码  \*/  **public** **static** **boolean** writeListToFile(List<String> lineList,String outputFilePath,String charset) **throws** Exception{  File outputFile = **new** File(outputFilePath);  FileOutputStream fos = **new** FileOutputStream(outputFile);  **int** lineCounter = 0;  **for**(String line : lineList){  **if**(lineCounter > 0){  fos.write('\n');  }  fos.write(line.getBytes(charset));  lineCounter++;  }  fos.close();  **return** **true**;  }    /\*\*  \* 测试  \* **@param** args  \*/  **public** **static** **void** main(String[] args) **throws** Exception{  String txtFilePath = "房地产\\user\\1107717945.txt";  String inputCharset = "utf-8";  String outputCharset = "utf-8";  String outputFilePath = "newFile.txt";  List<String> res = *getTxtContent*(txtFilePath,inputCharset);  **for**(String temp : res){  System.***out***.println(temp);  }  *writeListToFile*(res,outputFilePath,outputCharset);  } | | pojo类的编写  1. WbUserInfoPojo  |  | | --- | | **package** pojos;  **import** java.util.Date;  **public** **class** WbContentInfoPojo {  **private** **long** id;  **private** String content;  **private** Date Time;  **private** **int** repostsCount;  **private** **int** commentsCount;    **public** **long** getId() {  **return** id;  }  **public** **void** setId(**long** id) {  **this**.id = id;  }  **public** String getContent() {  **return** content;  }  **public** **void** setContent(String content) {  **this**.content = content;  }  **public** Date getTime() {  **return** Time;  }  **public** **void** setTime(Date time) {  Time = time;  }  **public** **int** getRepostsCount() {  **return** repostsCount;  }  **public** **void** setRepostsCount(**int** repostsCount) {  **this**.repostsCount = repostsCount;  }  **public** **int** getCommentsCount() {  **return** commentsCount;  }  **public** **void** setCommentsCount(**int** commentsCount) {  **this**.commentsCount = commentsCount;  }      } |  1. WbUserContentPojo  |  | | --- | | **package** pojos;  **import** java.util.Date;  **public** **class** WbContentInfoPojo {  **private** **long** id;  **private** String content;  **private** Date Time;  **private** **int** repostsCount;  **private** **int** commentsCount;    **public** **long** getId() {  **return** id;  }  **public** **void** setId(**long** id) {  **this**.id = id;  }  **public** String getContent() {  **return** content;  }  **public** **void** setContent(String content) {  **this**.content = content;  }  **public** Date getTime() {  **return** Time;  }  **public** **void** setTime(Date time) {  Time = time;  }  **public** **int** getRepostsCount() {  **return** repostsCount;  }  **public** **void** setRepostsCount(**int** repostsCount) {  **this**.repostsCount = repostsCount;  }  **public** **int** getCommentsCount() {  **return** commentsCount;  }  **public** **void** setCommentsCount(**int** commentsCount) {  **this**.commentsCount = commentsCount;  }      } |  业务类编写---数据激活 DataLoadManager   |  | | --- | | package manager;  import java.util.ArrayList;  import java.util.List;  import utils.FileOperatorUtil;  import utils.IOUtil;  public class DataLoadManager {  public static void main(String[] args) throws Exception{  String inputDir = "房地产";  String inputCharset = "utf-8";    ArrayList<String> resultList = new ArrayList<>();  List<String> txtFilePathList = FileOperatorUtil.getAllSubNormalFilePath(inputDir);  for(String txtFilePath : txtFilePathList){  List<String> singleTxtLineList = IOUtil.getTxtContent(txtFilePath,inputCharset);  resultList.addAll(singleTxtLineList);  }  for(String line : resultList){  System.out.println(line);  }  }  } |   把上面的整理成一个方法：  @Author Tangyingying1997   |  | | --- | | package manager;  import java.util.ArrayList;  import java.util.List;  import utils.FileOperatorUtil;  import utils.IOUtil;  public class DataLoadManager {    public static List<String> getAllFileLineList(String inputDir,String charset) throws Exception{  ArrayList<String> resultList = new ArrayList<>();  List<String> txtFilePathList = FileOperatorUtil.getAllSubNormalFilePath(inputDir);  for(String txtFilePath : txtFilePathList){  List<String> singleTxtLineList = IOUtil.getTxtContent(txtFilePath,charset);  resultList.addAll(singleTxtLineList);  }    return resultList;  }  public static void main(String[] args) throws Exception{  String inputDir = "房地产";  String inputCharset = "utf-8";  List<String> res = getAllFileLineList(inputDir,inputCharset);  for(String line : res){  System.out.println(line);  }  }  } |   以上完成了：给定一个目录，读出了其中的所有数据。  ~~接下来需要将读出来的数据结构化。~~ | | 通过以上代码，将content和user中的数据全部读出来了，但是没有分成两部分，即content数据与user数据粘连在一起。我们需要再改动一下，把content和user区分开来。 | | package manager;  import java.util.ArrayList;  import java.util.List;  import pojos.WbContentInfoPojo;  import pojos.WbUserInfoPojo;  import utils.FileOperatorUtil;  import utils.IOUtil;  public class DataLoadManager {    public static List<String> getAllFileLineList(String inputDir,String charset) throws Exception{  ArrayList<String> resultList = new ArrayList<>();  List<String> txtFilePathList = FileOperatorUtil.getAllSubNormalFilePath(inputDir);  for(String txtFilePath : txtFilePathList){  List<String> singleTxtLineList = IOUtil.getTxtContent(txtFilePath,charset);  resultList.addAll(singleTxtLineList);  }    return resultList;  }  public static void main(String[] args) throws Exception{  String inputDir = "房地产";  String inputCharset = "utf-8";  List<String> res = getAllFileLineList(inputDir,inputCharset);    //对读出的内容集res做结构化  //1.定义user集  List<WbUserInfoPojo> userPojoList = new ArrayList<>();  //2.定义content集  List<WbContentInfoPojo> contentPojoList = new ArrayList<>();    //遍历res的每一行  for(String line : res){  //首先去掉前后的空格  line = line.trim();    if(line.length() == 0){  continue;  }  //如果以 < 开头，说明是content  if(line.startsWith("<")){    }else{  //否则就是user，暂时不考虑其他类型的数据  //先把[...]之间的内容截取出来  line = (String) line.subSequence(line.indexOf('[')+1, line.lastIndexOf(']'));  //再按照逗号切割  String[] kvArray = line.split(",");    WbUserInfoPojo userInfoPojo = new WbUserInfoPojo();    //遍历每个kvArray，找每个字段对应的值  for(String kv : kvArray){  //按照等号把字段和值分割开  String[] kvPair = kv.split("=");  if(kvPair[0] .equals("id")){  userInfoPojo.setId(Long.parseLong(kvPair[1]));  }else if(kvPair[0] .equals("screenName")){  userInfoPojo.setScreenName(kvPair[1]);  }else if(kvPair[0] .equals("name")){  userInfoPojo.setName(kvPair[1]);  }else if(kvPair[0] .equals("province")){  userInfoPojo.setProvince(Integer.parseInt(kvPair[1]));  }else if(kvPair[0] .equals("remark")){  userInfoPojo.setRemark(kvPair[1]);  }  }  System.out.println(userInfoPojo.toString4FileOutput());  //System.out.println(line);  //@Author Tangyingying1997  }    }  }  } | | 在WbuserInfoPojo中加上一个toString定制： | | **public** String toString4FileOutput() {  List<Object> fieldList = **new** ArrayList<>();  fieldList.add(id);  fieldList.add(screenName);  fieldList.add(province);  fieldList.add(remark);    **return** StringUtil.*join*(fieldList, "\001");  } | | 同时编写了一个StringUtil工具类· | | **package** utils;  **import** java.util.List;  **public** **class** StringUtil {  **public** **static** String join(List<Object> objList,String deli){  **int** lineCounter = 0;  StringBuilder stringBuilder = **new** StringBuilder();  **for**(Object obj : objList){  **if**(lineCounter > 0){  stringBuilder.append(deli);  }  stringBuilder.append(obj.toString());  lineCounter++;  }  **return** stringBuilder.toString();  }  } | | @Author Tangyingying1997  以上代码意在：   1. 对于读取出user+content的数据，进行区分，区分出user和content 2. 如果以‘<’开头，则认为是content; 3. 如果不以‘<’开头，则认为是user 4. 对于user,先掐头去尾截取出[]之间的内容，也就是：id=1182389073, screenName=张三, name=张有雪 5. 再按照,分割，也就是：id=1182389073 6. 再按照=分割，也就是：id 1182389073 7. 同时创建WbUserInfoPojo对象，set进各个属性的值   但是，打印后却发现出现了空指针错误：  @Author Tangyingying1997 | | 出现以上空指针bug的原因是，元数据中会有：  User [id=1107717945, screenName=null, name=null, province=0]  以上情况，需要关注screenName=null，那么我们经过切割要将screenName的值set进一个userPojo对象的screenName属性时，就会发现set的是一个null.  **空字符串 or 空值 出现混乱**  **其实就是某个属性的值为“null”字符串，但是直接当作Object类型被add进集合时，误以为是NULL了。**  **~~ArrayList 在 add时是认识null字符串和null值的。~~**  @AUthor Tangyingying1997  **切割后要进行trim()** | | |

1. P3

P2已经基本完成了将文本文件转成结构化的list<UserInfoPojo>.

但是我们存在两个大的list，也就是List<WbUserInfoPojo>和List<WbContentPojo>.

所以我们考虑创建一个新的pojo，这个pojo结合了user和content。即UserAndContentInfoPojo.

@AUthor Tangyingying1997

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **package** pojos;  **import** java.util.List;  **public** **class** UserAndContentInfoPojo {  **private** List<WbUserInfoPojo> userPojoList;  **private** List<WbContentInfoPojo> contentPojoList;    //构造器  **public** UserAndContentInfoPojo(List<WbUserInfoPojo> userPojoList, List<WbContentInfoPojo> contentPojoList) {  **super**();  **this**.userPojoList = userPojoList;  **this**.contentPojoList = contentPojoList;  }  //getter setter  **public** List<WbUserInfoPojo> getUserPojoList() {  **return** userPojoList;  }  **public** **void** setUserPojoList(List<WbUserInfoPojo> userPojoList) {  **this**.userPojoList = userPojoList;  }  **public** List<WbContentInfoPojo> getContentPojoList() {  **return** contentPojoList;  }  **public** **void** setContentPojoList(List<WbContentInfoPojo> contentPojoList) {  **this**.contentPojoList = contentPojoList;  }      }  然后运行代码，会看到我们把userpojolist和contentpojolist放在了一个大的list里面，叫做UserAndContentPojoList.  UserAndContentInfoPojo [userPojoList=[WbUserInfoPojo [id=1107717945, screenName=null, name=null, province=0, city=0, location=null, description=null, userDomain=null, gender=null, followersCount=0, friendsCount=0, statusesCount=0, favouritesCount=0, createdAt=null, verified=false, remark=房地产, verifiedReason=null], WbUserInfoPojo [id=1182389073, screenName=张三, name=张有雪, province=0, city=0, location=null, description=null, userDomain=null, gender=null, followersCount=0, friendsCount=0, statusesCount=0, favouritesCount=0, createdAt=null, verified=false, remark=房地产, verifiedReason=null], WbUserInfoPojo [id=1182391231, screenName=李四, name=李四孩, province=0, city=0, location=null, description=null, userDomain=null, gender=null, followersCount=0, friendsCount=0, statusesCount=0, favouritesCount=0, createdAt=null, verified=false, remark=房地产, verifiedReason=null]], contentPojoList=[]]  将以上的userAndContentPojoList持久化到一个文件中去，并且指定编码为utf-8.  此时需要在IOUtil工具类中添加一个，参数为字符串的持久化方法：   |  | | --- | | /\*\*  \* 接收一个字符串 ,用某种编码写到某个输出路径中  \*/  **public** **static** **boolean** writeListToFile(String txtContent,String outputFilePath,String charset) **throws** Exception{  File outputFile = **new** File(outputFilePath);  FileOutputStream fos = **new** FileOutputStream(outputFile);  fos.write(txtContent.getBytes(charset));  fos.close();  **return** **true**;  } |   然后进行测试在DataLoadManager类中：   |  | | --- | | **public** **static** **void** main(String[] args) **throws** Exception{  String inputDir = "房地产";  String inputCharset = "utf-8";  //1.把给定目录中的文本文件读取成list  List<String> res = *getAllFileLineList*(inputDir,inputCharset);  //2.将字符串的list转换成结构化的pojolist  UserAndContentInfoPojo userAndContentInfoPojo = *getConstructInfoPojo*(res);  //System.out.println(userAndContentInfoPojo);  //System.out.println(userAndContentInfoPojo.toString());  //3.把两个pojo形成的list对象分别持久化输出到一个统一的文本文件中，编码为utf-8  List<WbUserInfoPojo> userInfoPojoList = userAndContentInfoPojo.getUserPojoList();  String outputFilePath4User = "user\_pojo\_list.txt";  String outputCharset = "utf-8";    StringBuilder stringBuilder = **new** StringBuilder();  **int** lineCounter = 0;  **for**(WbUserInfoPojo tempPojo : userInfoPojoList){  **if**(lineCounter > 0){  stringBuilder.append("\n");  }  stringBuilder.append(tempPojo.toString4FileOutput());  lineCounter++;  }  IOUtil.*writeListToFile*(stringBuilder.toString(), outputFilePath4User, outputCharset);  } |   结果如下：（根目录多出了一个user\_pojo\_list.txt文件，并且其中写进去了暂时所有user数据的pojo）    到此，效果做到了，但是不太规范啦，所以整理下代码，抽出一个方法：   |  | | --- | | /\*\*  \* 传入复合pojo，将复合pojo拆分并且分别持久化到对应的txt文件中  \* **@param** userAndContentInfoPojo  \* **@param** userOutputFilePath  \* **@param** contentOutputFilePath  \* **@param** outputCharset  \* **@return**  \* **@throws** Exception  \*/  **public** **static** **boolean** writePojoToFile(UserAndContentInfoPojo userAndContentInfoPojo,String userOutputFilePath,String contentOutputFilePath,String outputCharset ) **throws** Exception{  List<WbUserInfoPojo> userInfoPojoList = userAndContentInfoPojo.getUserPojoList();  StringBuilder stringBuilder = **new** StringBuilder();  **int** lineCounter = 0;  **for**(WbUserInfoPojo tempPojo : userInfoPojoList){  **if**(lineCounter > 0){  stringBuilder.append("\n");  }  stringBuilder.append(tempPojo.toString4FileOutput());  lineCounter++;  }  IOUtil.*writeListToFile*(stringBuilder.toString(), userOutputFilePath, outputCharset);  **return** **true**;  } |   那么主测试方法，就简化成了：   |  | | --- | | **public** **static** **void** main(String[] args) **throws** Exception{  String inputDir = "房地产";  String inputCharset = "utf-8";  //1.把给定目录中的文本文件读取成list  List<String> res = *getAllFileLineList*(inputDir,inputCharset);  //2.将字符串的list转换成结构化的pojolist  UserAndContentInfoPojo userAndContentInfoPojo = *getConstructInfoPojo*(res);  //System.out.println(userAndContentInfoPojo);  //System.out.println(userAndContentInfoPojo.toString());  //3.把两个pojo形成的list对象分别持久化输出到一个统一的文本文件中，编码为utf-8  String userOutputFilePath = "user\_pojo\_list2.txt";  String contentOutputFilePath = "";  String outputCharset = "utf-8";  *writePojoToFile*(userAndContentInfoPojo,userOutputFilePath,contentOutputFilePath,outputCharset);  } | |

1. P4

直到p3我们完成了user数据结构化并且序列化到txt文件。P4需要着手进行一下content的xml解析。

@Author tangyingying1997 作者自用，请勿盗用

Xml的解析用Java如何实现呢？

先建一各xml解析的工具类XmlParseUtil

稍等，确认使用dom4j来解析xml，先在pom.xml文件中引入dom4j的jar.

|  |
| --- |
| <dependencies>  <dependency>  <groupId>dom4j</groupId>  <artifactId>dom4j</artifactId>  <version>1.6.1</version>  </dependency>  </dependencies> |

只引入这个包不够，还得粘贴上旧pom文件的仓库。

XmlParserUtil如下：

|  |
| --- |
| **package** utils;  **import** java.io.File;  **import** java.io.StringReader;  **import** org.dom4j.Document;  **import** org.dom4j.DocumentException;  **import** org.dom4j.Element;  **import** org.dom4j.io.SAXReader;  /\*\*  \* 用来解析xml文件  \* **@author** Apollo  \*  \*/  **public** **class** XmlParserUtil {  **public** **static** Element getXmlRootElement(String xmlContent){  SAXReader reader = **new** SAXReader();  **try**{  StringReader stringReader = **new** StringReader(xmlContent);  Document document = reader.read(stringReader);  Element rootElement = document.getRootElement();  System.***out***.println(rootElement.elementText("content"));  **return** rootElement;  }**catch**(DocumentException e){  e.printStackTrace();  }  **return** **null**;  }    **public** **static** **void** main(String[] args){  String xmlContent = "<comment><content>回复@雷亚雷：在哈佛或MIT，每天都有各种议题的小型研讨会，凡在中午或晚餐时间举行的，一般都会准备晚餐。即使如此，有时也会忽略。//@雷亚雷：王老师，不吃午饭吗？</content><time>2012-4-6 5:45:52</time><repostsCount>574</repostsCount><commentsCount>290</commentsCount></comment>";  Element e = *getXmlRootElement*(xmlContent);  System.***out***.println(e.elementText("time"));  }  } |

有了这个工具就可以解析基本的xml文件了，那么继续回到dataLoadManager，去补充对content文件的解析代码：

|  |
| --- |
| **if**(line.startsWith("<")){  //**TODO**  line = line.trim();  Element rootElement = XmlParserUtil.*getXmlRootElement*(line);  WbContentInfoPojo contentInfoPojo = **new** WbContentInfoPojo();  //contentInfoPojo.setId(Long.parseLong(rootElement.elementText("id")));  contentInfoPojo.setContent(rootElement.elementText("content"));  //字符串转为date类型  //contentInfoPojo.setTime(new Date(rootElement.elementText("time")));  contentInfoPojo.setTime(DateUtil.*getDate*((rootElement.elementText("time"))));  contentInfoPojo.setRepostsCount(Integer.*parseInt*(rootElement.elementText("repostsCount")));  contentInfoPojo.setCommentsCount(Integer.*parseInt*(rootElement.elementText("commentsCount")));    //造好一个content就丢进contentlist  contentPojoList.add(contentInfoPojo);  }**else**{......} |

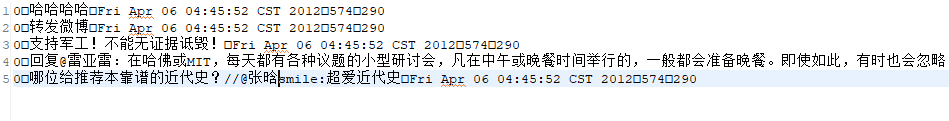
**要特别注意一下，这里从xml解析拿出的时间是字符串类型，专门写了个工具类来将字符串类型转为Date类型。**

|  |
| --- |
| contentInfoPojo.setTime(DateUtil.*getDate*((rootElement.elementText("time")))); |
| **package** utils;  **import** java.text.ParseException;  **import** java.text.SimpleDateFormat;  **import** java.util.Date;  /\*\*  \* 把字符串转为日期类型  \* **@author** Apollo  \*  \*/  **public** **class** DateUtil {  **static** SimpleDateFormat *dateFormat* = **new** SimpleDateFormat("yyyy-MM-dd hh:mm:ss");    **public** **static** Date getDate(String dateString) **throws** ParseException{  **return** *dateFormat*.parse(dateString);  }  } |

如此，就构成了一个个的contentInfoPojo，并且把这些都塞进了contentPojoList中，再写个序列化contentPojoList的方法，其实加在了user序列化代码附近，没有重写。

|  |
| --- |
| //2.序列化contentPojoList  List<WbContentInfoPojo> contentInfoPojoList = userAndContentInfoPojo.getContentPojoList();  StringBuilder stringBuilder2 = **new** StringBuilder();  **int** lineCounter2 = 0;  **for**(WbContentInfoPojo tempPojo : contentInfoPojoList){  **if**(lineCounter2 > 0){  stringBuilder2.append("\n");  }  stringBuilder2.append(tempPojo.toString4FileOutput());  lineCounter2++;  }  //3.调用序列化方法  IOUtil.*writeListToFile*(stringBuilder2.toString(), contentOutputFilePath, outputCharset); |

至此，content的内容也持久化到了文件中：



**但是明显看到content的id没有正确读取到。**

**......**

1. P5

通过前面已经完成了第一步：非结构化数据到结构化的转换。

接下来需要将数据导入hive中。前提是：

1. 准备Linux环境
2. 搭建hadoop
3. 搭建hive
4. 进入Linux后，创建一个目录job/code，在其下创建一个专门的目录来存放该项目，例如weibo\_data\_to\_struct.

|  |
| --- |
| mkdir weibo\_data\_to\_struct |

1. 在该目录下创建4个目录

|  |
| --- |
| mkdir config  mkdir create  mkdir deal  mkdir udf |

1. 进到config中，把hive搞进来。

|  |  |
| --- | --- |
| //1.先找到hive的路径，将路径记录下来  which hive  //2.在config下创建脚本文件set\_env.sh  touch set\_env.sh  //3.编辑这个脚本   |  | | --- | | #! /bin/bash  HIVE=’/usr/bin/hive’ //注意这里的路径就是本机hive的路径 | |

1. 进到create，要在这个目录下，建两张表

先建user

(1)

|  |
| --- |
| //weibo\_user\_base\_info.sh  #! /bin/bash  source ../config/set\_env.sh #配置脚本位置  db=job008 #数据库名  table\_name=weibo\_user\_info #表名  $HIVE -e "  use $db;  create external table $table\_name(  id bigint,  screenName string,  province int,  remark string  )  partitioned by (dt string comment 'update date') #dt作为分区？  ROW FORMAT DELIMITED FIELDS TERMINATED BY '\001' LINES TERMINATED BY '\n'  STORED AS textfile;  " |

建好user表之后，在hive中，use job008; show tables;看一下

继续在create路径下，创建第二张表content

|  |
| --- |
| cp weibo\_user\_base\_info.sh weibo\_comment\_info.sh |

|  |
| --- |
| #! /bin/bash  source ../config/set\_env.sh #配置脚本位置  db=job008 #库名  table\_name=weibo\_content\_info #表名  $HIVE -e "  use $db;  create external table $table\_name(  id bigint,  comment string,  time timestamp,  repostsCount int,  commentsCount int  )  partitioned by (dt string comment 'update date') #dt作为分区？  ROW FORMAT DELIMITED FIELDS TERMINATED BY '\001' LINES TERMINATED BY '\n'  STORED AS textfile;  " |
| sh weibo\_comment\_info.sh |

到此两张表就建好了。

1. 进到deal里面，把前期搞到的结构化数据拷进来

content\_pojo\_list.txt 和 user\_pojo\_list.txt

|  |
| --- |
| rz -y ... |

然后more一下看看，数据乱码没。

1. 下面开始装载user数据。在deal下

|  |
| --- |
| touch produce\_weibo\_user\_base\_info.sh |
| #! /bin/bash  source ../config/set\_env.sh  db=”job\_002”  table\_name=”weibo\_user\_base\_info”  $HIVE -e “  use $db;  load data local inpath ‘./user\_pojo\_list.txt’  overwrite into table $table\_name partition(dt=’20200406’)  ;  “ |
| sh produce\_weibo\_user\_base\_info.sh |

1. 测试下user的数据是否成功装载进表中

|  |
| --- |
| select \* from weibo\_user\_base\_info; |

1. 装载content数据

|  |
| --- |
| touch produce\_weibo\_comment\_info.sh |
| #! /bin/bash  source ../config/set\_env.sh  db=”job\_002”  table\_name=”weibo\_comment\_info”  $HIVE -e “  use $db;  load data local inpath ‘./content\_pojo\_list.txt’  overwrite into table $table\_name partition(dt=’20200406’)  ;  “ |
| sh produce\_weibo\_comment\_info.sh |

1. 测试cotent数据

|  |
| --- |
| select \* from weibo\_comment\_info; |

1. P6
2. P7