# 摘 要

关键词：数据清洗; Hadoop; 大数据算法; MapReduce; 序列化

目录

[摘 要](#_Toc1959143161)

[第1章 绪 论](#_Toc558399121)

[1.1 研究问题的背景...........](#_Toc895536296)

[1.2 研究问题的挑战............](#_Toc890845146)

[1.3 当前研究工作的不足之处](#_Toc316657121)

[1.4 本文的工作要解决的问题以及方法](#_Toc1546461110)

[1.5 本文的贡献.................](#_Toc1565533004)

[1.6 章节安排.....................](#_Toc1652472135)

[第2章 系统/方法框架](#_Toc1313600952)

[2.1 系统框架..........................](#_Toc85036834)

[2.2 框架解析](#_Toc1511550097)

[2.3 各部分详解........................](#_Toc302708468)

[2.3.1 配置文件](#_Toc447668175)

[2.3.2 数据类型（datatype）](#_Toc1626536945)

[2.3.3 文件处理（fileprocess）](#_Toc1938617066)

[2.3.4 客户自定义类（personalclass）](#_Toc682730224)

[2.4 程序运行流程解析](#_Toc1421385748)

[第3章 MapReduce技术](#_Toc1389016210)

[3.1 Map和Reduce](#_Toc1860761488)

[3.2 MapReduce程序设计](#_Toc220811598)

[3.3 Hadoop和HDFS](#_Toc360201396)

[第4章 序列化技术](#_Toc351279781)

[4.1 序列化](#_Toc511757891)

[4.2 Java序列化和Hadoop序列化](#_Toc1314061824)

[4.3 MapReduce的实现](#_Toc934984495)

[第5章 多路输出技术](#_Toc1593443676)

[5.1 多路输出](#_Toc747789269)

[5.2 MapReduce实现的多路输出](#_Toc277240165)

[第6章 实验](#_Toc14115159)

[6.1 实验环境](#_Toc1725690855)

[6.2 实验设计.......................](#_Toc1172796398)

[6.3 实验结果......................](#_Toc1973258320)

[第7章 结论](#_Toc136606328)

[参考文献](#_Toc2068332694)

# 第1章 绪 论

## 1.1 研究问题的背景

随着互联网和其他行业的快速发展，信息的急剧增长，海量的数据处理和分析变得越来越普遍，传统的数据清洗技术面临严峻的挑战。海量的数据让传统的数据清洗手段变得越来越费时，但是留给数据清洗的时间越来越少；为了适应环境的快速变化，用户对快速得到数据有更迫切的要求；复杂多变的社会环境，提高了数据的复杂性和多变性，需要具有更好的扩展性。应对这些挑战获得更好性能和扩展性的关键是并行,然而由于大数据在采集和导入 过程中容易引入不满足数据质量要求的数据，即噪声数据。异常数据、不一致数据、重复数据、缺失数据等，都属于噪声数据。根据“Garbage In, Garbage Out”的原理，对不满足数 据质量要求的数据进行分析预测，得到的是不准确甚至错误的结论。这些噪声数据在不同的预测场景中将不同程度地影响到数据的质量和预测分析的结果，这对后期数据分析和数据挖掘都是不利的。因此在数据分析和数据挖掘前，需要对数据进行预处理。数据清洗作为数据预处理的关键一步，通过相关技术对脏数据进行清洗和修复，可以提升数据质量，进而提高数据挖掘和预测分析的准确性和高效性。因此，研究在大数据上的高效准确的数据清洗技术是必要的。

## 1.2 研究问题的挑战

数据清洗有着如下的定义：数据清洗是指发现并纠正数据文件中可识别的错误的最后一道程序，包括检查数据一致性，处理无效值和缺失值等。但这不是一个明确的概念，比如如何定义和处理无效值和缺省值。这就需要处理者编写代码去处理特定的数据。但是在大数据处理面前，普通的PC或者程序是无法直接对数据进行处理的。所以数据清洗往往和大数据处理平台相互结合，其中Hadoop就是一个非常适合数据清洗的大数据批处理框架。但是该框架的结构、搭建、运行是需要消耗很多人力成本，所以开发一个一个可拓展的，适应多种数据的，基于Hadoop的数据清洗框架是十分有必要的，并且可以给没有接触过Hadoop的人员在不了解Hadoop的情况下进行简单的数据清洗工作。

## 1.3 当前研究工作的不足之处

当前的工作大部分是对某一个任务进行的针对性的代码构建，比如处理一段天气的记录、网页访问的记录、手机流量的查询统计等。这些已经完成的研究工作都有一个共同的不足：无法适应多种数据的需要。在Hadoop框架下，所有的类和构造的方法被固定在了代码当中。如果需要处理不同的数据，需要对代码进行大范围的重构，并且涉及到了许多Hadoop底层的部分。这是不利于实际人员处理的。总结来说，当前的工作基本都缺乏必要的拓展性、难以快速部署、并且无法改变需求。

## 1.4 本文的工作要解决的问题以及方法

解决了目前数据清洗框架拓展性差，定制性强，前置需求多的问题。在不同数据集上可以随意地转换，不需要对代码进行大量重构。使用者无需了解Hadoop的底层实现过程。主要利用的方法是抽象出了一个条目化数据的基类，在基类中使用键值对的方式可以存储以属性的名称做键值，属性的内容做值的任意数目多的属性。并且根据Xml文件直接按照用户的要求进行初始化。用户如果需要属性之间的限制条件可以简单地复用基类，并且重写一个方法即可。

## 1.5 本文的贡献

本文针对条目化的数据（单个的数据由多个属性构成，并且每一个原始数据可以写成一行）建立起了一个可拓展的，具有良好复用性的，可以高度定制的，基于Hadoop的数据清洗框架。

## 1.6 章节安排

首先是系统结构和算法的介绍，接下来三章是比较重要的使用到的技术。然后是实验验证部分以及结论。共分为六个章节，19个小节。对本框架进行详细的描述。

# 第2章 系统/方法框架

## 2.1 系统框架

系统分为四个部分，数据类型(datatype)部分，文件处理(fileprocess)部分，交互计算(interaction)部分，客户自定义类(personalclass)部分。各个部分之间的调用关系如下：

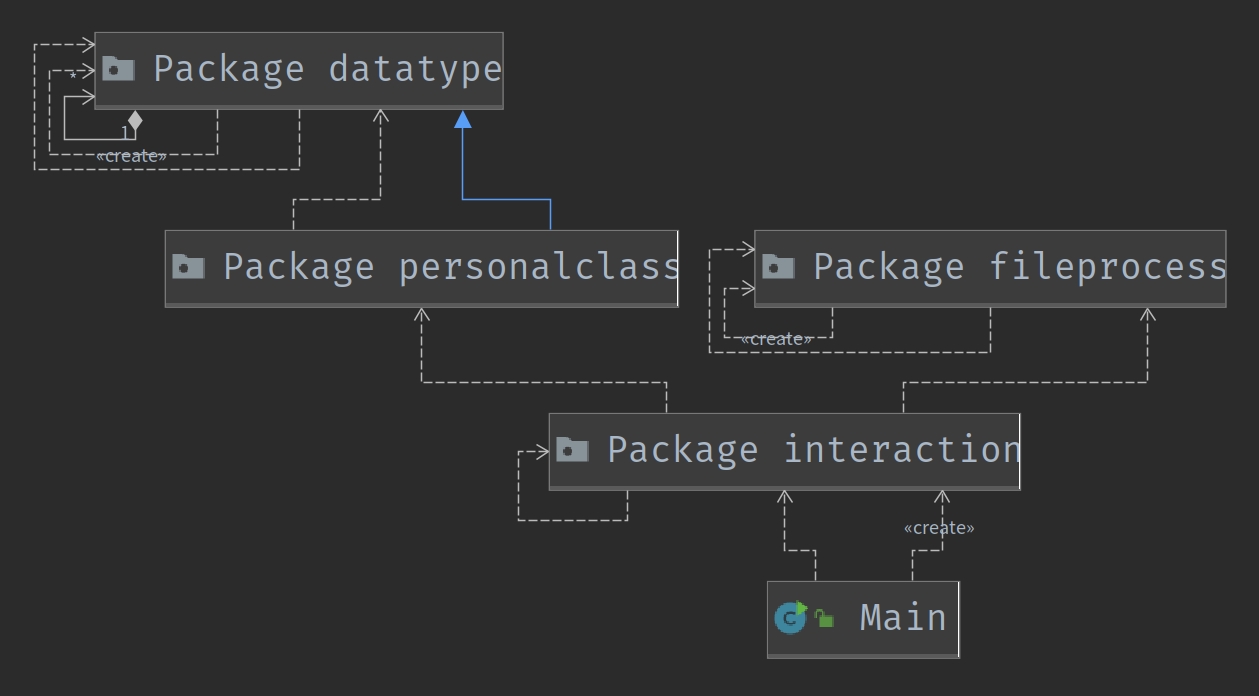


图 1

## 2.2框架解析

Main函数是框架的启动函数，它构造出交互对象（Client）的实例，传入用户配置好的环境变量（environment.xml）作为参数。其中Client对象的构造如下：

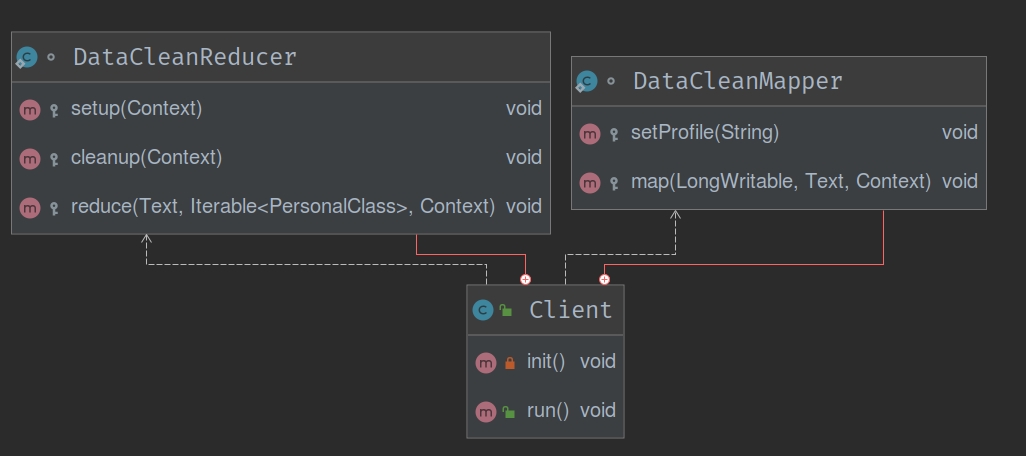


图 2

Client对象包含了数据解析以及MapReduce代码构造的部分。接着Main启动交互对象（Client）。交互对象通过调用fileprocess包中的方法对所有的配置文件进行解析，完成初始化。接着Client启动MapReduce程序，MapReduce程序调用personalclass中的方法对用户自定义的类进行初始化以及检查准确性。最终按照用户设置的方法进行多路输出。完成数据清洗工作。

## 2.3 各部分详解

### 2.3.1 配置文件

配置文件包括环境变量——environment.xml和描述文件——profile.xml

**environment.xml**

用户可以参照test文件中给定的样式，自行编写自定义的环境变量。接下来是各个变量的名称解析：

globalHdfsPath：HDFS地址。

localFilePath：本地需要上传的原始数据的绝对路径。

profilePath：描述文件的本地绝对路径。

jarPath：保存的可执行的jar文件的绝对路径。

inputFilePath：上传文件到HDFS的地址。

outputFilePath：需要输出文件的HDFS地址。

key：判别多路输出的关键字（属性的名称）。程序会根据数据中的属性为key 的所有数据的取值，分别建立文件夹，将数据输出在对应的文件夹内。

**profile.xml**

用户可以参照test文件中给定的样式，自行编写自定义的描述文件。接下来是示例：

给出的是String类型数据的描述的示例

<FName type="String" format=".\*"></FName>

其中“FName”为属性的名称，type项的取值有四种分别是String，Integer，Boolean，Double。分别表示字符串，整数，布尔型，浮点数。format为各种数据类型的限制条件。限制条件的描述在接下来的数据类型中给出。

### 2.3.2 数据类型（datatype）

类型的继承关系如下：

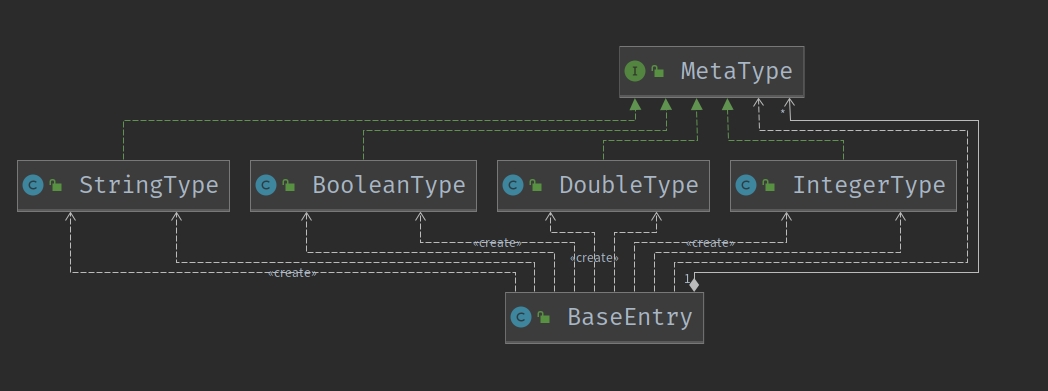


图 3

#### MetaType

作为基本数据类型的接口，定义了各类数据类型需要实现的基本方法。其中有如下四类继承:StringType, BooleanType, DoubleType, IntegerType分别表示字符串类型、布尔类型、浮点数类型、整数类型。包含的方法有：

setValue(String value)：根据一个字符串设置各类的值。

getType()：获得类型。

check()：进行属性检查（在Reduce部分若check不通过则会直接将这一条数 据删除）一个属性的check方法只能检查当前这一个属性的值是否符 合要求。多个属性之间的check在personalclass中给出。

setFormat(String format)：设置一个属性的限制条件。

write(DataOuput dataOutput)：序列化。

readFields(DataInput dataInput)：反序列化。

序列化和反序列化在第四章的序列化技术中会进行详细介绍。其余方法的实现在源代码中均有具体注释。接下来给出的是四种基本数据类型的特例化的方法的具体描述以及参数限定。

#### StringType

setValue(String value)：将value去掉前后的空格后返回。

setFormat(String format)：直接将传入的format作为检查该元素的正则表达式

check()：检查是否符合format描述的正则表达式。

#### IntegerType

setValue(String value)：value中应不应包含除了逗号" , "减号" - "数字之外的任 何字符

setFormat(String format)：value中应不应包含除了逗号" , "减号" - "分号" ;"数字 之外的任何字符。其中分号作为分割最大值和最小值 的符号必须存在且只能存在一个，同时分号之前的数 字表示最小值，之后的数字表示最大值。

check()：检查自身的值是否在最大值和最小值之间。

#### DoubleType

setValue(String value)：value中应不应包含除了逗号" , "减号" - "小数点" . "数字 之外的任何字符。

setFormat(String format)：value中应不应包含除了逗号" , "减号" -"小数点" ."分 号" ; "数字之外的任何字符。其中分号作为分割最大值 和最小值的符号必须存在且只能存在一个，同时分号 之前的数字表示最小值，之后的数字表示最大值。

check()：检查自身的值是否在最大值和最小值之间。

#### BooleanType

setValue(String value)：可以用"true", "1"表示"真" "false", "0"表示"假"。

setFormat(String format)：format作为设置限制数据内容的字符串仅有三种取值 "true"，"false"，"\*"。分别表示限制数据为"真"，"假" 和任意值。

check()：若format为通配符 \* 表示不需要对真假进行检查, 直接返回true, 否 则检查value是否和format匹配。

#### BaseEntry

作为抽象出来的一条数据的类型，其中包含了一个字典(keyWords)。记载了用户在profile.xml文件所定义的一条数据的所有属性。同时在解析profile.xml文件时会按照文件中的从上到下的顺序产生一个初始化数据的顺序(order)。按照相同的顺序给定数据内容便可以直接生成一个数据对象。BaseEntry含有如下基本方法：

void setOrder(ArrayList<String> order)：设置各个属性初始化的顺序。

void setValues(List<String> keyValues)：给一条数据设置各个属性的值。

Boolean check()：检查字典中所有属性的值是否符合各自的限定条件，这部分 可以在personalclass中进行重写，以扩展数据检查的多样性。

setFormat(Map<String, String> format)：根据参数给定的key作为属性的名称， value作为format的内容，对keyWords 中的数据进行format的设置

write(DataOutput dataOutput)和readFields(DataInput dataInput)分别是序列化和反序列化的部分在第四章中进行介绍。

### 2.3.3 文件处理（fileprocess）

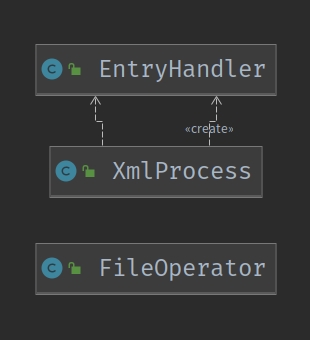


图 4

#### XmlProcess、EntryHandler

使用了Java自带的SAX方法处理Xml文件。可以完成对environment.xml和profile.xml的解析。

#### FileOperator

集成了Hadoop的Java API来对HDFS进行远程处理。FileOperator包含的方法如下：

testExit(String globalHdfsPath, String path)：检测HDFS是否存在查找的文件

mkdir(String globalHdfsPath, String hdfsDir)：在HDFS上创建目录

rmdir(String globalHdfsPath, String hdfsDir)：删除HDFS上的目录

touch(String globalHdfsPath, String hdfsFilePath)：在HDFS上创建文件

rm(String globalHdfsPath, String hdfsFilePath)：在HDFS上删除文件

appendContentToFile(String globalHdfsPath, String content, String hdfsFilePath)：在HDFS上的选中文件之后添加内容

putFileToHDFS(String globalHdfsPath, String localPath, String hdfsPath)：上传本地文件到HDFS中

getFileFromHDFS(String globalHdfsPath, String localPath, String hdfsPath)：从HDFS下载文件到本地

getContentFromHDFS(String globalHdfsPath, String hdfsPath)：从HDFS读取选中文件到内存中

### 2.3.4 客户自定义类（personalclass）

用户自定义类是用户扩展数据检查的需要用户实现的类。只需要重写两个初始化的方法。即可对check方法进行重写。用户可以任意定义属性之间的限制关系。或者对某一种属性进行更加复杂的限定。

客户自定义类的实现有两种方法。1.直接在IDE中修改源代码。在IDE中执行Main方法。2.在IDE编写新的personalclass代码并且生成。替换解压后的jar包中的原personalclass文件。再将所有文件进行打包，利用指令在终端中运行。

## 2.4 程序运行流程解析

Main函数是框架的启动函数，它构造出交互对象（Client）的实例，需要传入用户配置好的环境变量（environment.xml）的绝对路径作为参数。接着Main启动交互对象（Client）。交互对象通过调用fileprocess包中的方法对所有的配置文件进行解析，完成初始化。包括获得描述文件（profile.xml）地址，hdfs远程地址等。重点是通过读取描述文件，获得处理的数据的各个属性的类型以及构造方式，将其传给Hadoop框架的配置（conf）对象中。接着启动自身内部的MapReduce程序开始进行数据清洗的工作。在Map阶段对读取到的数据利用conf中存储的各个属性的类型以及构造方式，对一个数据对象进行构造。之后按照描述文件中设置的多路输出的key的内容作为Map阶段输出的key，一个数据对象作为value进行输出。在Reduce阶段，首先对获得到的数据调用check方法，检查是否符合用户定义。对所有符合定义的数据，利用Hadoop提供的多路输出的方法，输出到对应key的文件中。

# MapReduce技术

MapReduce是面向大数据并行处理的计算模型、框架和平台。

·MapReduce将复杂的、运行于大规模集群上的并行计算过程高度地抽象到了两个函数：Map和Reduce。

·MapReduce采用“分而治之”策略，一个存储在分布式文件系统中的大规模数据集，会被切分成许多独立的分片（split），这些分片可以被多个Map任务并行处理。

·MapReduce设计的一个理念就是“计算向数据靠拢”，而不是“数据向计算靠拢”，因为，移动数据需要大量的网络传输开销。

·MapReduce框架采用了Master/Slave架构，包括一个Master和若干个Slave。Master上运行JobTracker，Slave上运行TaskTracker。

## 3.1 Map和Reduce

利用Hadoop框架实现MapReduce主要利用了两个方法map和reduce。这两个方法的输入输出以及说明如下：（用单词数量统计作为例子）

|  |  |  |  |
| --- | --- | --- | --- |
| 方法 | 输入 | 输出 | 说明 |
| map | <k1, v1>  例：  <行号，"a, b, c"> | List<k2, v2>  例：  <"a",1 > <"b",1> <"c",1> | 将所有数据解析成一批键值对。每一个<k1, v1>通过map会产生一批中间变量<k2, v2> |
| reduce | <k2, List(v2)>  例：  <"a", <1, 1, 1>> | <k3, v3>  例：  <"a", 3> | 框架处理出一个键值对<k2, List(v2)>, 表示由拥有相同的key的所有值组成的list, 传给reduce |

在reduce部分的说明中，提到了框架处理出一个键值对。这部分是Hadoop框架的核心部分，这部分的内容比较复杂，同时也不是本文的主要内容，就不多加赘述了。总之，Hadoop为我们提供了一种数据处理的方式：就是按照map和reduce的方法对数据进行并行化的处理。

## 3.2 MapReduce程序设计

具体实现map和reduce两部分的功能需要继承Hadoop所定义的Mapper和Reducer类。在本框架中继承和定义如下：

static class DataCleanMapper extends Mapper<LongWritable, Text, Text, PersonalClass>

static class DataCleanReducer extends Reducer<Text, PersonalClass, NullWritable, Text>

对应在3.1中的部分DataCleanMapper从左向右分别是：LongWritable对应着k1，Text对应着v1，Text对应着k2，PersonalClass对应着v2，DataCleanReducer 中的NullWritable对应着k3，Text对应着v3。表示着对于输入的数据，Hadoop会自动按照换行符进行划分，行号作为输入的key，一行的数据作为输入的value。经过map的处理后，输出多组Text作为key，实例化后的数据作为value的键值对。经过框架的处理后，所有Text相同的键值对输出到了同一个Reducer进行处理。Reducer处理过后输出NullWritable作为key（在这里使用了NullWritable表示不需要key），Text作为value输出到指定的文件中。

具体实现细节则需要查看map方法和reduce方法，接下来给出设计思想。

首先需要重写map方法：

protected void map(LongWritable key, Text value, Context context)

map方法的参数是Mapper的输入参数，即按照行进行划分后的原始数据。接着利用之间解析好的xml文件（通过Hadoop使用的参数传递对象conf进行多设备之间的配置传输）对任意一行数据进行实例化personalclass。利用解析environment.xml文件时获得到的参数key（同样通过conf进行传输），对personalclass按照参数key进行取值，获得到对应的值作为输出的key，类型是Text，同时再将当前实例化后的对象作为Value进行输出。

接着重写reduce方法：

protected void reduce(Text key, Iterable<PersonalClass> values, Context context)

reduce方法中的key即是map处理过后的一个key，values是所有拥有相同key的personalclass对象。仅需调用personalclass对象的check方法检测是否符合用户规范即可。只需要对于符合的数据进行输出。

## 3.3 Hadoop和HDFS

HDFS全称是（Hadoop Distribute File System）即Hadoop的分布式文件系统。所有Hadoop的Java API中包含了大量处理HDFS文件的接口。Fileprocess中就大量使用了这些接口。同时在MapReduce程序运行时，同样需要HDFS的支持。

由于MapReduce是分布式处理数据的工具，所以输入文件，中间文件，输出文件都不能指定在集群中的某一台机器的位置上。所以需要指定HDFS文件地址，来对规范输入输出文件。在启动MapReduce Job之前，需要使用Hadoop定义的FileInputFormat.addInputPath方法和FileOutputFormat.setOutputPath方法，制定输入文件的HDFS地址和输出文件的HDFS地址。

# 序列化技术

## 4.1 序列化

由于MapReduce作为分布式计算框架，集群间的数据传输都是通过网络进行的。而网络设备之间的通讯都是通过二进制序列即比特序列的形式的。同时在Java进程之间信息也使用了相同的方法传递数据。所以map和reduce程序之间若需要数据传递则需要保证传递的数据都是可序列化的。在第3章中所提到的personalclass作为key或者value都是在personalclass可以被序列化的基础上的。

## 4.2 Java序列化和Hadoop序列化

4.1中所提到的序列化在Java和Hadoop都有分别的实现，但是在MapReduce程序编写时通常选择Hadoop所提供的序列化。下面给出了两个序列化的主要介绍和分析。

#### Java序列化

只要实现了serializable接口就能实现序列化与反序列化。但是Java序列化算法需要考虑：将对象实例相关的类元数据输出。递归地输出类的超类描述直到不再有超类。类元数据完了之后，开始从最顶层的超类开始输出对象实例的实际数据值。从上至下递归输出实例的数据。所以Java序列化很强大，序列化得到的信息很详细，但是序列化后很占内存。

#### Hadoop序列化

相对于JDK比较简洁，在集群中信息的传递主要就是靠这些序列化的字节楼来传递的，所以更快速度，容量更小。Hadoop提供的序列化有着以下的优点：

紧凑：带宽是集群中信息传递的最宝贵的资源所以我们必须想法设法缩小传递信息的大小。与Java相比 Java序列化不够灵活，为了更好的控制序列化的整个流程所以使用Writable。java序列化会保存类的所有信息依赖等，Hadoop序列化不需要。

对象可重用：JDK的反序列化会不断地创建对象，这肯定会造成一定的系统开销，但是在Hadoop的反序列化中，能重复的利用一个对象的readField方法来重新产生不同的对象。Java序列化每次序列化都要重新创建对象，内存消耗大。但是Writable可以重用。所以内存的消耗会少很多。这也是在做分布式计算所需要重要考虑的。

可拓展性：Hadoop自己写序列化很容易，可以利用实现Hadoop的Writable接口 实现了直接比较字符流以确定两个Writable对象的大小。而Java不是，Java的序列化机制在每个类的对象第一次出现的时候保存了每个类的信息, 比如类名, 第二次出现的类对象会有一个类的reference, 导致空间的浪费。

## 4.3 MapReduce的实现

在4.2中已经介绍了Hadoop序列化的优势以及大致方法。承接2.3.2中未介绍的write方法和readFields方法。这个小节给出本文所提到的BaseEntry和personalclass实现的细节。

四个基础的数据类型实现方法基本相同，按照固定的顺序对储存的数据进行字节流的写和字节流的读。接着就是BaseEntry了，BaseEntry继承了WritableComparable类。由于BaseEntry含有两个私有变量keyWords和order，所以write方法只需要先输出变量的个数以及order序列的全部内容，接着按照order序列的内容对keyWords进行字节流的输出即可。对于readFields方法，按照write方法的顺序，分别读取变量个数、order的具体内容、keyWords的具体内容即可。详细过程可以查看源代码的具体实现。

# 第5章 多路输出技术

## 5.1 多路输出

多路输出实现需要将结果按照一定的标准分配到不同的HDFS目录中，方便建立hive分区表做后期的数据分析或者其他的相关计算。多路输出同时可以做到数据分区的作用。通常使用分区来水平分散压力。将数据从物理上移到和使用最频繁的用户更近的地方，以及实现其目的。 Hive中有分区表的概念，可以看到分区具重要性能优势，而且分区表还可以将数据以一种符合逻辑的方式进行组织。所以多路输出有着十分重要的作用。

## 5.2 MapReduce实现的多路输出

在第3章中提到的按照环境变量中设置的key进行划分，就是为了多路输出做的准备。map完成了数据的解析和分类，reduce完成了数据的检查和输出，其中在输出部分不同于直接利用参数Context.write进行文件的输出。本文使用了Hadoop提供的MultipleOutputs接口，完成到多个文件的输出。定义如下：

private MultipleOutputs<NullWritable, Text> mos

mos作为多路输出的操作者，可以使用 “BaseOnKey” 参数完成按照key的不同分别输出到不同文件的目标：

mos.write("BaseOnKey",NullWritable.get(),txtKey,key.toString()+"/" + key.toString());

同时可以使用“All”参数，将所有数据输出到同一个文件中：

mos.write("All", NullWritable.get(), txtKey);

这就完成了多路输出这一个要求。方便了Hive分区表的建立。

# 第6章 实验

## 6.1实验环境

配置：X86-64 8-core processor; 16GB RAM; 256GB ROM

系统：Ubuntu 19.10

开发框架：Hadoop 2.10（伪分布式模式）

Java：OpenJDK version "11.0.5"; OpenJDK 64-Bit Server VM;

IDE：InteliJ IDEA

## 6.2 实验设计

利用http://da.qcri.org/dc/ 给定的税收（约100000组）和医院数据（约115000组），分别进行实验。检查配置文件的可复用性，以及personalclass的可复用性。实验设计如下：

1. 税收数据的key分别用Gender（取值：F或M）和State（多种取值）验证

多路输出的结果（多路输出验证）

1. 税收数据对profile.xml文件进行编辑过滤掉所有Salary不大于95000的人。（profile文件整数大小format验证）
2. 医院数据对profile.xml 文件进行编辑过滤出所有Condition项目中为H开头类型数据（profile文件String的format验证）
3. 税收数据对personalclass的check方法重写，删除所有FName为Aad的数据（personalclass用户自定义check验证）

## 6.3 实验结果

实验一

修改environment文件，使key=Gender

通过浏览器可以直接访问HDFS远程文件列表，可以得到如下结果：

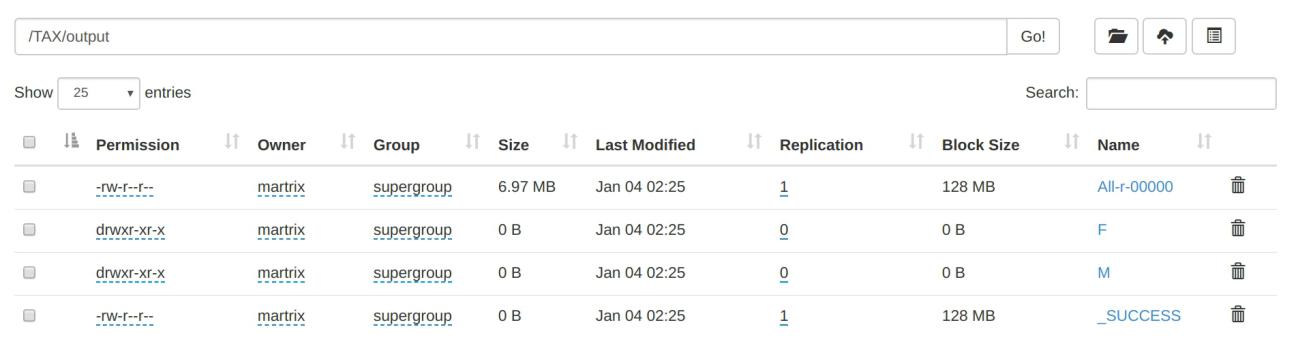


图 5

其中\_SUCCESS文教表示Job执行成功。可以看到由于Gender项有两个取值F、M所以产生了两个文件夹F和M。打开其中的文件（F作为示例）：可以看到有接近一半的人的性别为女性，并且数据内容的确仅有女性。

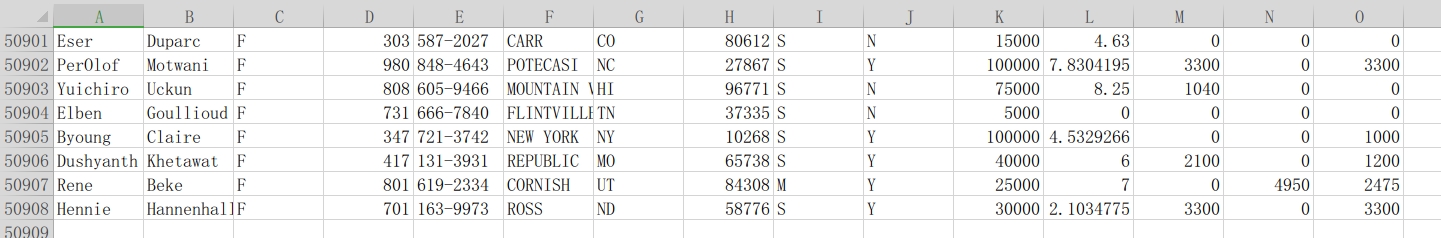


图 6

修改environment文件，使key=state

通过浏览器可以直接访问HDFS远程文件列表，可以得到如下结果：

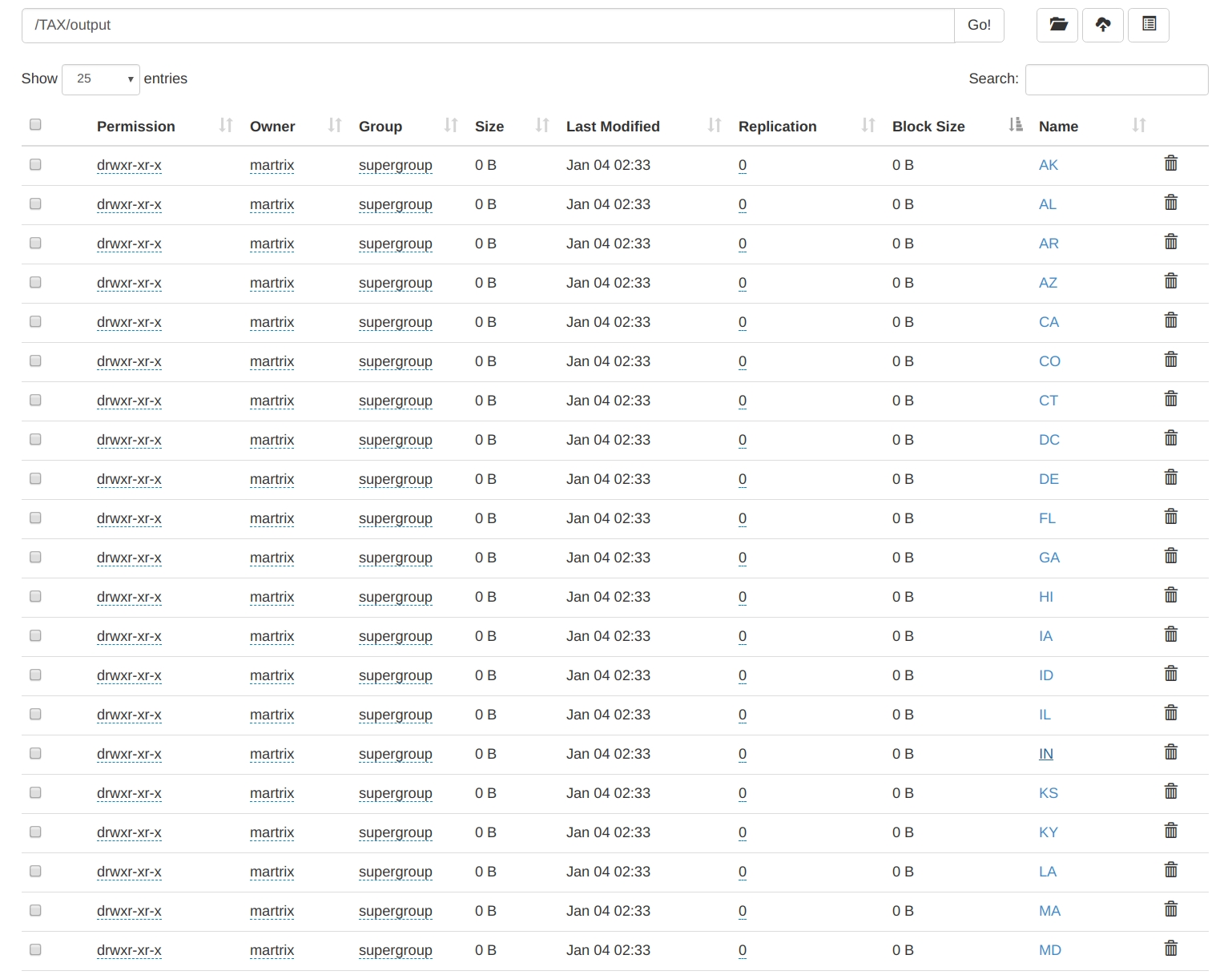


图 7

只显示了一部分的输出结果，共计输出了三页结果并且都以state的内容作为文件夹的名称。但是在时间方面远长于前一组实验，由于数据大小相同，只是分组的数量不同，本文认为可能由于数据集过小，导致了分组所耗费的实现占据了主要部分。

实验二

修改profile文件，使Salary项目中的format最大值修改为95000。

访问HDFS文件列表得到以下结果：

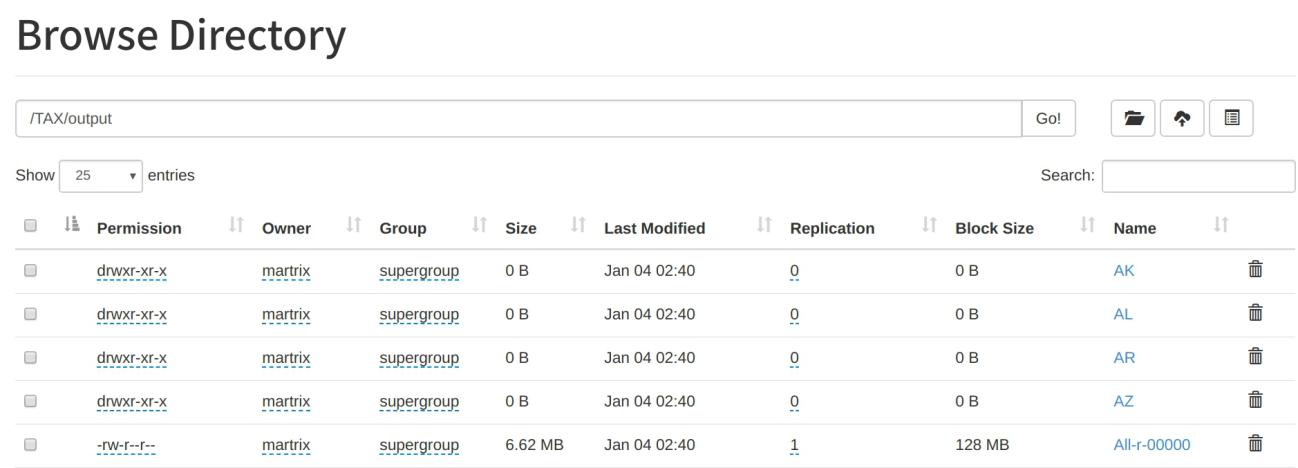


图 8

由于在多路输出部分同时设置了将数据也输出到All文件内，所以可以直接查看所有的数据，打开All文件并将Salary降序排序可以看到：

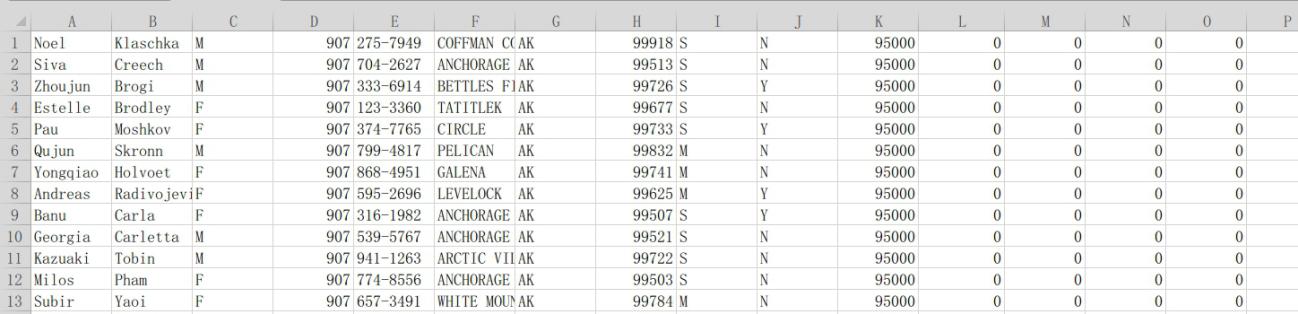


图 9

最大Salary从100000变成了95000，成功删除掉了超过95000的数据。

实验三

修改profile文件将Condition项目中的format修改为H.\*，表示只匹配字母H为开头的字符串。

访问HDFS文件列表下载输出的全部数据得到以下结果：

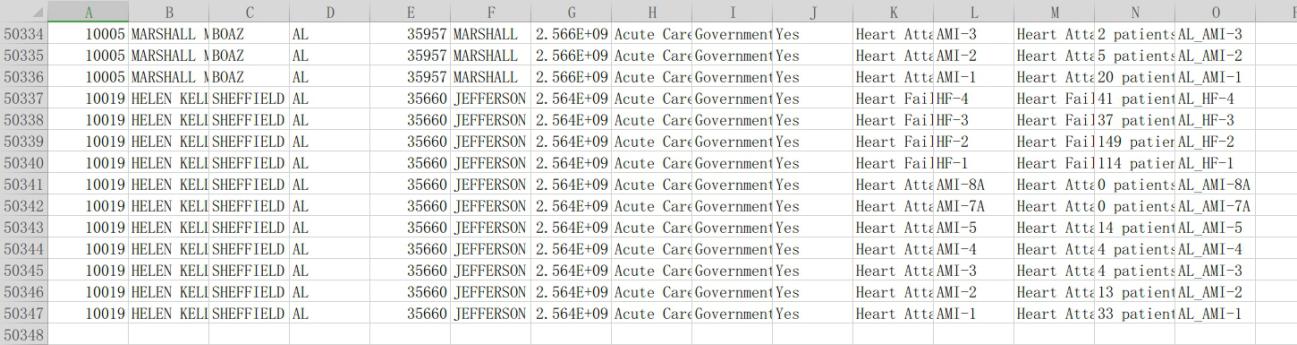


图 10

可以看到仅保留了Condition项目中H开头的数据，并且数据减少到了5万条

实验四

编写新的personalclass类，在check部分修改成了检查FName是否为Aad，生成后替换原jar中的personalclass部分，删除所有FName为Aad的数据。

访问HDFS文件列表下载输出的全部数据得到以下结果：

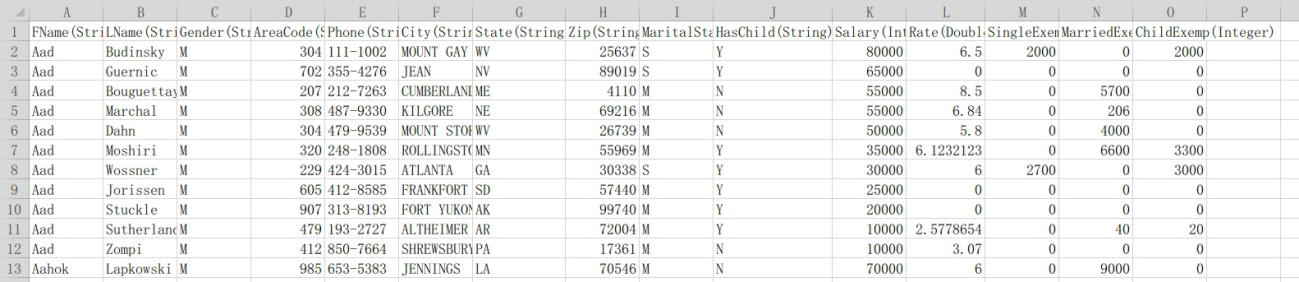


图 11-1

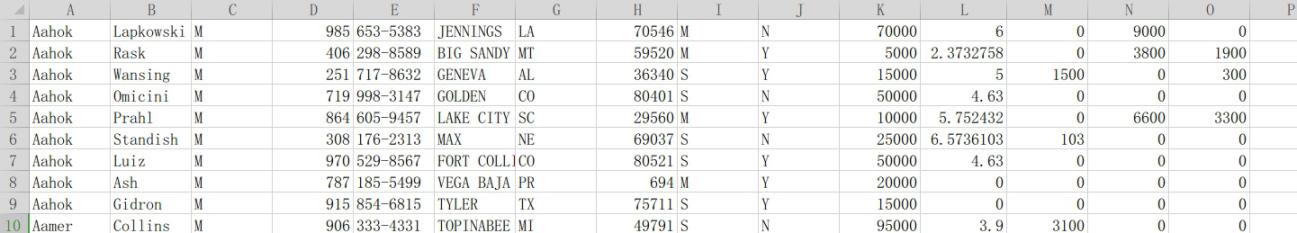


图 11-2

图 11-1是原始数据，图 11-2为过滤后的数据，可以看到删除了12名FName为Aad的数据。

# 第7章 结论

本文的框架主要使用了Apache所开发的Hadoop分布式计算框架。针对数据集中的单个的数据由多个属性构成，并且每一个原始数据可以写成一行的数据。建立起了一个可拓展的，具有良好复用性的，可以高度定制的，基于Hadoop的数据清洗框架。用户可以自定义Xml文件，直接按照用户的要求进行初始化。在不同数据集上可以随意转换，不需要对代码进行大量重构。使用者无需了解Hadoop的底层实现过程。用户如果需要属性之间的限制条件可以简单地复用基类，并且重写一个方法即可。解决了目前数据清洗框架拓展性差，定制性强，前置需求多的问题。

# 参考文献

[1]陆嘉恒.Hadoop实战 (第1版)[M].清华大学出版社:北京,2015:22.

[2]王宏志.大数据算法[M].机械工业出版社:北京,2015:150.

[3]Hadoop环境搭建[EB/OL].

https://blog.csdn.net/hliq5399/article/details/78193113,2017-10-10.

[4]利用hadoop进行天气情况的统计[EB/OL].

https://www.jianshu.com/p/e665a3720473,2019-12-08.

[5]mapreduce多路输出实例[EB/OL].

https://blog.csdn.net/bitcarmanlee/article/details/54863022,2017-02