南京大學

本 科 毕 业 论 文

院　　系　计算机科学与技术系

专　　业　计算机科学与技术

题　　目　面向大数据清洗的 **Hadoop** 代码生成

年　　级　**2010**　学 号　**101220007**

学生姓名　车开达

指导老师　张天　职 称　副教授

论文提交日期　**2014** 年 **6** 月

南京大学本科生毕业论文（设计）中文摘要

毕业论文题目: 面向大数据清洗的 Hadoop 代码生成

计算机科学与技术系 院系 计算机科学与技术 专业 2010 级

本科生姓名: 车开达

指导教师（姓名、职称）: 张天副教授

摘要:

随着人们对大数据认识的不断深入和数据挖掘技术的广泛应用，数据质量这一影响挖掘效果的决定性因素成为商业和技术领域关注的焦点。数据清洗的目标就是检测并消除数据中存在的各种错误和不一致情况，有效提升数据质量。然而数据清洗因其处理情况繁琐、效率低，往往事倍功半，被认为是一项“dirty”工作。另外在大数据的背景下，海量数据清洗不再是一件低成本的过程。因此提供一种操作简单而能有效处理海量数据的大数据清洗工具不失为一件有意义的想法。

为了实现此想法，本文工作设计开发了基于 Hadoop 平台的大数据清洗工具（H-BDCT）。本文工作的思路是利用 MDE（Model Driven Engineering）的思想，把清洗任务转换为模型转换，通过多级转换引擎，自动生成可在 Hadoop 平台上执行数据清洗任务的 Hadoop 代码。这样只需要根据数据清洗任务抽象出简单的模型或设置本文提出的配置文件即可在 Hadoop 平台上完成大数据清洗任务。本文首先分析了在实际数据中存在的问题，提出了相应的清洗操作，并在此基础上设计了简单易用的配置文件，针对前人对数据清洗工作所抽取的模型操作原语提出了相应的转换规则。然后又给出了从配置文件到可执行 Hadoop 代码的转换规则和配置文件翻译引擎的实现。最后使用 H-BDCT 完成了对简单测试数据的各种数据清洗操作，并取得了较为理想的处理结果。

关键词：大数据；Hadoop 平台；数据清洗

南京大学本科生毕业论文（设计）英文摘要

THESIS: Hadoop code generation for big data cleaning

DEPARTMENT: Computer Science and Technology

SPECIALIZATION: Computer Science and Technology

UNDERGRADUATE: Che Kedar

MENTOR: Associate Professor. Tian Zhang

ABSTRACT:

With the deeper understanding of big data and the more extensive application of data mining technology, data quality which is becoming the decisive factors affecting the mining results in the field of business and technology has become the focus of attention. The goal of data cleaning is to detect and eliminate data a variety of errors and inconsistencies, which effectively improves the data quality. However, because data cleaning is cumbersome, inefficient and often double the work, it is considered a “dirty” work. Also in the context of big data, cleaning massive data is no longer a low-cost process. So it would be a meaningful idea to provide a simple and effective tool to deal with huge amounts of data in data cleaning area.

In order to realize this idea, the project develops a Hadoop-based big data cleaning tool (H-BDCT). The approach of this project uses the model conversion thinking of MDE (Model Driven Engineering), which transforms the cleaning tasks to the models or configuration files, adopts a multi-stage conversion engine, and automatically generates codes executing data cleaning tasks on Hadoop platform. So people just need to abstract a simple model or set a configuration file according to data cleaning tasks, and they will complete the proposed large data cleaning tasks on Hadoop platform. This paper analyzes the problems existing in the actual data, proposes corresponding cleaning operation, and then designs a using easily configuration file. This paper puts forward the conversion rules for primitives of model operation extracted from data cleaning operation in other’s work, and the conversion rules between the configuration

file and executable codes running on Hadoop. Then this paper gives the realization of translation engine of configuration files. Finally, this paper presents an example that uses H-BDCT to complete a variety of cleaning operation on a simple test data, and achieves the desired processing results.

KEY WORDS: Big Data; Data Cleaning; Hadoop

目录

1 引言

1.1 研究背景

“大数据”是继“云计算”之后，在信息科技领域出现的一个研究焦点。“大数据”在物理学、生物学、环境生态学等领域以及军事、金融、通讯等传统行业存在已有时日，却因为近年来互联网和信息行业的高速发展而引起人们的关注。无处不在的传感器和微处理器产生了庞大的数据来源，互联网日常产生和传统行业积累的数据正在迅速膨胀并变大，它决定着企业的未来发展，虽然现在企业可能没有意识到数据爆炸性增长带来问题的隐患，但随着时间的推移，人们将越来越多的意识到数据对企业的重要性。

大数据时代对人类的数据驾驭能力提出了新的挑战，也为人们获得更为深刻、全面的洞察能力提供了前所未有的空间与潜力。丰富的数据中蕴含着许多具有潜在利用价值的信息，这也推动着以数据挖掘为代表的数据分析技术的高速发展。在数据挖掘中，没有高质量的输入数据，就没有高质量的挖掘结果。然而在现实世界中数据仓库极易受到噪声、缺失值和不一致数据的侵扰，因为数据太大，并且多半来自多个异种数据源，如果不能得到有效处理，将导致低质量的挖掘结果。因而引入了数据清洗[1]等技术来提高数据质量[2]。数据清洗主要在数据仓库、数据挖掘和总体数据质量管理这 3 个领域研究较多[3]。在数据仓库的研究和应用领域，数据清洗处理时构建数据仓库的第一步，由于数据量巨大，不可能进行人工处理，因此自动化数据清洗受到工业界的广泛关注。数据清洗的目的是检测数据中存在的错误和不一致，剔除或者改正它们，以提高数据质量。

1.2 研究意义

在数据挖掘领域中，数据清洗（Data cleaning）是有效数据挖掘的前导性工作，也是一个必不可少的步骤，甚至可能在整个数据挖掘过程中占到 60%的工作量。它必须满足以下几个条件：无论是单数据源还是多数据源，都要检测并且

出去数据中所有明显的错误和不一致;尽可能地减小人工干预和用户的编程工作量，而且要容易扩展到其他数据源；应该和数据转换相结合；要有相应的描述语言来指定数据转换和数据清洗操作,所有这些操作应该在一个统一的框架下完成。[4]而这个工作在数据挖掘领域通常都被认为是“dirty”的工作,虽然对数据挖掘的研究日益深入，但大部分研究工作都集中在数据挖掘算法上，数据清洗等数据预处理工作没有得到足够多的研究人员的关注,工业界开发的工具也存在一定的局限性。

另外大数据时代来临，数据集的规模达到 PB、EB 级别，数据遍历一次就会耗费很长的时间[5]。在这种情况下，做数据清洗并不是一件低成本的事情，需要一个有效的方法来完成对大数据的清洗任务。本文设计实现了一种基于 Hadoop 平台[6]的大数据清洗工具，它涵盖了用户对数据集的基本操作，完成了对 Hadoop 代码的底层封装，使得用户只需要一个简单的配置文件甚至利用 MDE(Model Driven Engineering)[7]的模型操作原语就可以自动生成可在 Hadoop 集群上执行的 Hadoop 代码,提供了有效的处理大数据集的数据清洗功能，从而高效地完成大数据清洗任务。

1.3 本文工作

数据清洗有很强的领域相关性，过去一段时间作为“dirty”工作很少有人关注，但作为数据挖掘和数据质量管理等领域的基础性工作，越来越引起人们的重视。本文从实践出发，设计开发了一个面向大数据清洗的 Hadoop 代码生成工具，较好的将 Hadoop 并行运算框架和大数据清洗任务结合到一起，得到了处理类日志文件数据的通用解决方案。本文将首先介绍关于本文的一些技术背景，包括实现了 MapReduce 编程思想的 Hadoop 平台、MDE 模型转换核心思想、csv 数据文件格式等。然后本文将对数据清洗任务进行简要分析，结合实际操作给出该数据清洗工具的整体设计方案和数据流分析。本文还将从 MDE 的视角来看待数据清洗的问题，将之视为模型转换（Model Transformation, MT）的一种特定应用，抽取数据清洗的基本操作，并提出对应的转换规则，这将有利于从模型到底层的层次功能划分和自动化实现。然后本文将介绍设计的自动生成 Hadoop 代码的配置文件，以及从配置文件到 Hadoop 代码的转换规则。另外本文还会定义该工具

适用的领域，提供相应的实例说明，并对比该设计和相关工作。最后本文对整个工作进行总结，指出工作中的不足之处，并提出改进思路和下一步工作计划。

1.4 论文结构

本文一共分为七个部分。

第一部分是引言，介绍了本文的研究背景，研究意义，研究的主要内容以及论文的整体框架。

第二部分介绍 Hadoop 等相关的技术背景。

第三部分对待处理数据所需操作做了详细分析，抽取其中的共同点，结合 Hadoop 平台的特性，给出了整个数据清洗工具的整体设计。

第四部分借助 QVT 等模型转换规范中提到的基本转换操作的思想抽象出数据清洗任务的操作原语，并根据实际任务的需要将其分解转换成更有效地指令，作为配置文件的参考，从而设计出配置文件及其指令。

第五部分介绍从配置文件到 Hadoop 的转换策略，即代码自动生成的思路。

第六部分通过一个实例对本文设计实现的数据清洗工具进行展示。

第七部分对本文工作做出总结，指出当前工作的不足和对将来工作的展望。

2 技术背景

2.1 Hadoop 平台

Hadoop 分布式计算框架作为 Apache 下的一个开源项目，它可以运行在大规模廉价硬件设备集群上，目前得到了大量的实际应用。很多企业以及研究机构使用 Hadoop 框架作为研究或者构建自己的云计算服务平台。

目前，Hadoop 已经成长为包括 Hadoop common，HDFS，MapReduce，ZooKeeper，HBase 等多个子项目。其中，HDFS 和 MapReduce 是该项目的核心，本质上是 Google GFS[8]和 Map/Reduce 算法模型[9]的一个 java 开源实现。要使用 Hadoop 完成大数据运算任务，就必须深入了解 HDFS[10]和 MapReduce。

2.1.1 Hadoop 分布式文件系统（HDFS）

作为 Hadoop 的核心技术之一，HDFS（Hadoop Distributed File System）是分布式计算中数据存储管理的基础。它使用各种措施保证其具有高容错、高可扩展性、高获得性、高吞吐率等特征，为海量数据提供了不怕故障的存储，为超大数据集的应用处理带来了很多便利。下面根据项目需要简要介绍 HDFS 的体系结构。

HDFS 是一个主从（Master/Slave）体系结构，一个 HDFS 集群是由一个 Namenode 和一定数目的 DataNodes 组成，如 2-1 图所示。从最终用户的角度来看，它就像传统的文件系统一样，可以通过目录路径对文件执行 CRUD（create/read/update/delete）操作。NameNode 是一个中心服务器，负责管理文件系统元数据以及客户端对文件的访问，DataNode 存储实际的数据。客户端通过同 NameNode 和 DataNode 的交互访问文件系统。客户端联系 NameNode 以获取文件的元数据，而真正的文件 IO 操作时直接和 DataNode 进行交互的。

NameNode（主控制服务器）负责管理文件系统的命名空间，记录文件数据块在每个 DataNode 上的位置和副本信息，协调客户端对文件的访问，以及记录命名空间内的改动或命名空间本身属性的改动。DataNode 负责它们所在的物理节点上的存储管理。HDFS 开放文件系统的命名空间以便让用户以文件的形式存储

数据。HDFS 的数据都是“一次写入、多次读取”，典型的块大小是 64MB。HDFS 文件通常是按照 64MB 被切分成不同的数据块，每个数据块尽可能地分散存储于不同的 DataNode 中。NameNode 执行文件系统的命名空间操作，比如打开、关闭、重命名文件或目录，还决定数据块到 DataNode 的映射。DataNode 负责处理客户的读取请求，依照 NameNode 的命令，执行数据块的创建、复制、删除等工作。

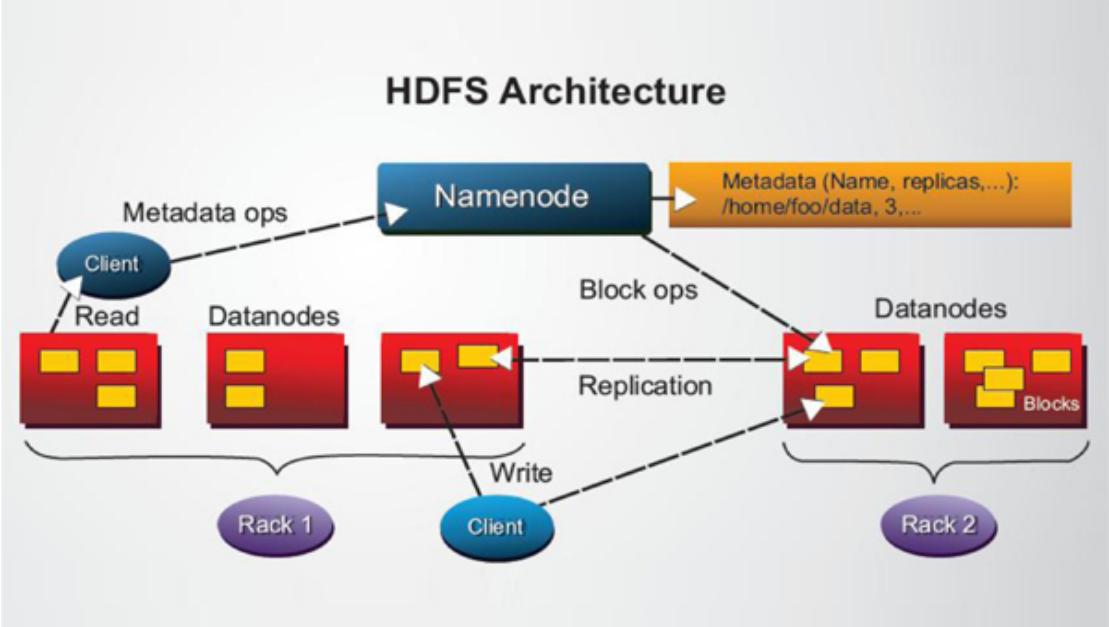

图 2-1 HDFS 的体系结构

HDFS 典型的部署是在一个专门的机器上运行 NameNode，集群中的其他机器各运行一个 DataNode；也可以在运行 NameNode 的机器上同时运行 DataNode，或者一台机器上运行多个 DataNode。一个集群只有一个 NameNode 的设计大大简化了系统架构。

系统启动时，NameNode 进入安全模式。安全模式下不发生文件块的复制。NameNode 接受来自各个 DataNode 的“心跳”回应和块报告。DataNode 启动时会遍历本地文件系统，产生一份 HDFS 数据块和本地文件对应关系的列表，并把这个报告发给 NameNode，这就是块报告。NameNode 控制所有的块复制操作。它周期性的接受来自集群中 DataNode 的“心跳”回应和块报告。收到一个节点的“心跳包”回应说明该 DataNode 正常。

另外，HDFS 提供了一系列保障 HDFS 可靠性、性能的措施。HDFS 提供了完备

的冗余备份和副本存放策略，在保证数据可靠性的同时兼顾性能。HDFS 提供了流失数据访问以提高数据吞吐量，使用了移动计算代替移动数据的策略来减少网络带宽的使用以提高性能。在数据一致性方面使用了简单一致性模型，即文件一经首次写入关闭后就不能再修改。这样的假定也让高吞吐量的数据访问成为可能。由此观之，HDFS 是可以满足该项目的基本需求的。

2.1.2 Hadoop MapReduce 编程模型

当前很多日志类数据，格式简单，处理逻辑容易。这样的数据处理运算很容易理解和实现，但由于输入数据量极大，要想在可接受的时间和预算内完成运算，单个主机无法完成，必须寻求横向廉价大集群并行来完成这项工作。然而如何分发数据、如何处理并行运算、如何处理各种类型的错误，所有这些问题综合在一起，需要大量的代码去处理，因此也使得原本简单的计算变得复杂而难以处理。

为了解决上述看似简单但实现复杂的问题，Hadoop 设计使用了一个新的抽象模型——MapReduce 框架，使用这个抽象模型，编程者只要表述其想要执行的简单运算即可，而不必操心数据如何分布、保证负载均衡、并行计算等复杂细节，这些问题都被封装在这个模型中。在输入数据上应用 Map 操作得出一个中间 key/value pair 的集合，然后在所有具有相同 key 值的 value 值上应用 Reduce 操作，从而达到合并中间数据，得到最终结果的目的。在这个过程中只需要用户提供 Map 和 Reduce 过程的简单操作，就可以实现大规模并行化运算。通过简单的接口来实现自动的并行化和大规模的分布式计算是 Hadoop 的 MapReduce 编程框架的最主要优点，而且通过使用 MapReduce 模型接口在大量普通廉价的 PC 机上实现有效的大数据量计算。下面详细介绍各个操作以及执行流程，从而对 MapReduce 编程框架有一个充分的了解。

(1) MapReduce 编程框架结构

JobTracker，它是一个 master 服务，JobTracker 负责调度 job 的每一个子任务 task 运行于 TaskTracker 上，并监控它们，如果发现有失败的 task 就重新运行它。一般情况应该把 JobTracker 部署在单独的机器上。

TaskTracker，它是运行于多个节点上的 slaver 服务。TaskTracker 负责直接执行每一个 task。TaskTracker 需要运行在 HDFS 的 DataNode 上。

JobClient，每一个 job 都会在用户端通过 JobClient 类将应用程序以及配置参数打包成 jar 文件存储到 HDFS，并把路径提交到 JobTracker，然后由 JobTracker 创建每一个 Task，并将它们分发到各个 TaskTracker 上去执行。

Map Task 和 Reduce Task，一个完整的 job 会自动依次执行 Mapper、Combiner（在 JobConf 指定了 Combiner 时执行）和 Reducer，其中 Mapper 和 Combiner 是由 Map Task 调用执行，Reducer 则由 Reduce Task 调用，Combiner 实际也是 Reducer 接口类的实现。Mapper 会根据 job jar 中定义的输入数据集按<key1, value1>对读入，处理完成生成临时的<key2, value2>对，如果定义了 Combiner，Map Task 会在 Mapper 完成调用该 Combiner 将相同 key 的值做合并处理，以减少输出结果集。Map Task 的任务全部完成后交给 Reduce Task 进程调用 Reducer 处理，生成最终结果<key3, value3>对。

(2) MapReduce 编程框架操作介绍

Hadoop 的 MapReduce 主要操作有 Map、Combine 和 Reduce 过程。

1) Map 操作

在 MapReduce 编程框架中，Map 操作是高度并行的。Map 数目通常由输入数据的大小决定，亦即输入文件的总块数。对于输入文件，Hadoop 会根据文件的大小划分为相应的片段。在文件划分的时候，不会考虑文件内部的逻辑结构，这样就要求数据内容间不能存在强耦合的关系。对于产生的每一个文件片段，Hadoop 会自动的创建一个 Map 任务来处理。

InputFormat 负责从文件中读取数据，并按照特定的(key, value)的形式将键值对传递给 Map 函数。InputFormat 会将文件划分为多个逻辑实例。而每个逻辑实例会对应一个 Map 函数。同时，InputFormat 会提供 RecordReader 的实现。这个 RecordReader 把逻辑实例转化为(key, value)的形式，传递给 Map。

Map 函数定义在一个 Mapper 类中，Mapper 对象会获取从 RecordReader 中得到的(key, value)对。用户在 Map 函数中会定义自己的处理过程来处理(key, value)，然后输出另一对(key, value)，这个键值对被称为中间键值对，并被 Output 使用 collect 方法收集。对于输出的中间键值对，我们可以自己定义 key 和 value 的类型，这个与输入键值对的类型没有关系。但是，在 Hadoop 中为了提高的网络传输效率，所有的 key 必须要实现 WritableComparable 接口，所有

的 value 必须实现 Writable 接口，这样，所有的 key，value 都能被序列化，并且可以通过 key 执行键值对排序操作。

2） Combine 操作

当 Map 操作输出它的(key, value)键值对之后，这些值会由于效率的原因存在内存中。对于每一个的 Map 的输出键值对，我们会聚集然后分发到不同的 Reduce 执行规约的操作。但有的时候，对于特定的任务，我们可以在 Map 本地就执行一个规约的操作，这样，Map 输出的结果得到的键值对就会大大减少，这样不仅仅提高了效率，而且也减轻了网络传输的负担。

这个在 Map 本地执行的规约过程，我们使用 Combine 操作来完成。在使用 Combiner 类后，Map 过程产生的(key, value)对不会立即输出，而是会先被收集到列表，每个索引键值对对应一个列表，当一定数量的键值对被写入时，这个缓冲区的所有键值对会被清空转移到 Combiner 类中的 Reduce 方法中，然后将合并操作产生的键值对作为类似 Map 操作输出的中间输出信息。

3） Reduce 操作

当一个 Reduce 任务开始时，它的输入来源于分散在多个节点上的 Map 任务所产生的许多文件。如果 Reduce 过程是运行在分布式模式下的话，需要在拷贝阶段先将这些文件拷贝到 Reduce 任务所在节点的本地文件系统中。

一旦本地数据准备就绪所有的数据都会追加到文件的最后，然后这个文件会被归并排序以保证给定每部分数据一个索引键。这使得 Reduce 操作非常简单：文件被顺序读入，然后输入文件中的一个索引键的所有对应值会被一个迭代器顺次传递给 Reduce 方法直到下一个索引键开始。规约完成后，每个执行的 Reduce 任务的输出都会包含一个输出文件，这个文件的文件夹由用户决定。

(3) Hadoop MapReduce 执行流程

Hadoop MapReduce 执行流程如图 2-2 所示。下面将对照该图详细讲解每一个步骤详情。

1）分割文件：输入文件被 Hadoop 的 MapReduce 库分割为多个限制最大值的片，然后启动集群中大量的程序拷贝。

2）指派 MapReduce 任务：在上述程序拷贝中有一个 JobTracker 程序，其它均为 TaskTracker 程序。JobTracker 给 TaskTracker 指派任务，当

TaskTracker 空闲时，JobTracker 指派其执行 Map 任务或 Reduce 任务。

3）读取：被指派 Map 任务的 TaskTracker 读取相应的 split 内容，从输入数据中分析出(key, value)对，使用用户自定义的 map 函数对此(key, value)进行处理，得到的中间键值对存入缓冲区。

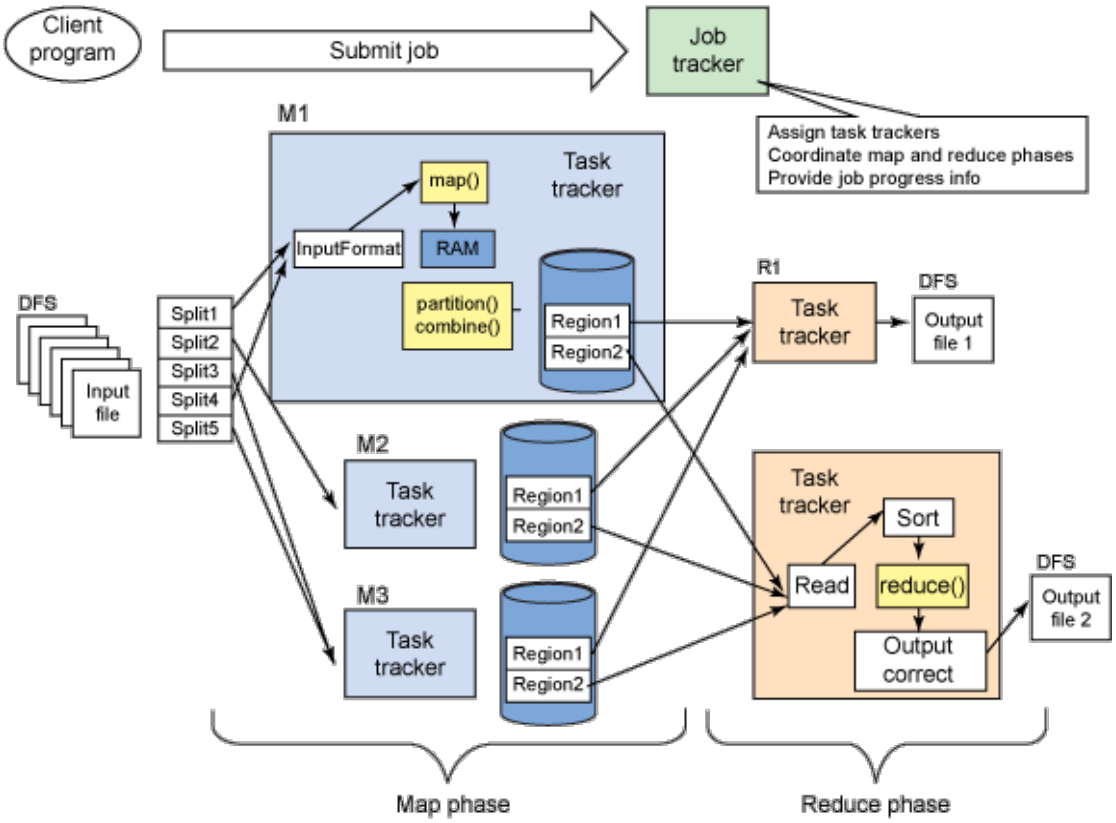

图 2-2 Hadoop MapReduce 执行流程

4）本地写入：缓冲区中的(key, value)对周期性的写入磁盘，在写入过程中被划分函数划分为 R 个区域。数据在本地磁盘的存储信息传输给 JobTracker，JobTracker 将相关位置信息传送给 Reduce TaskTracker。

5）远程读写：执行 Reduce 的 TaskTracker 得到数据位置信息后，使用远程方式读取 Map TaskTracker 缓存的数据。当 Reduce TaskTracker 读取到所有 Map 输出的数据，通过对关键字进行数据排序，并按照不同的 key 把数据分组。

6）写到输出文件：Reduce TaskTracker 对由 key 对应的数据进行分组，并发送 key 和相应中间值到用户自定义的 Reduce 函数。在最终的输出文件

写入 Reduce 函数的输出结果。

2.2 模型驱动工程（MDE）的模型转换思想

模型驱动工程（model-driven engineering，简称 MDE）是软件工程领域新型的一种软件开发模式。它以模型为首要软件制品，通过（元）建模和模型转换来驱动软件的开发，能够较好的解决异构平台间的相互转换问题[12]。显然模型转换思想是 MDE 的一个重要内容，文献[13]中给出了模型转换的定义：一组转换规则的集合，这些规则共同描述了源模型如何转换为目标模型，无论源模型和目标模型是否属于同一个层次，也不管是否使用同种模型描述语言。而模型转换规则是指源模型中一个或一些模型元素如何变换为目标模型中一个或一些模型元素的描述。如图 2-3 所示，模型转换就是获取源模型，通过一组转换规则，将源模型转换为目标模式，最后输出这个目标模型的过程。

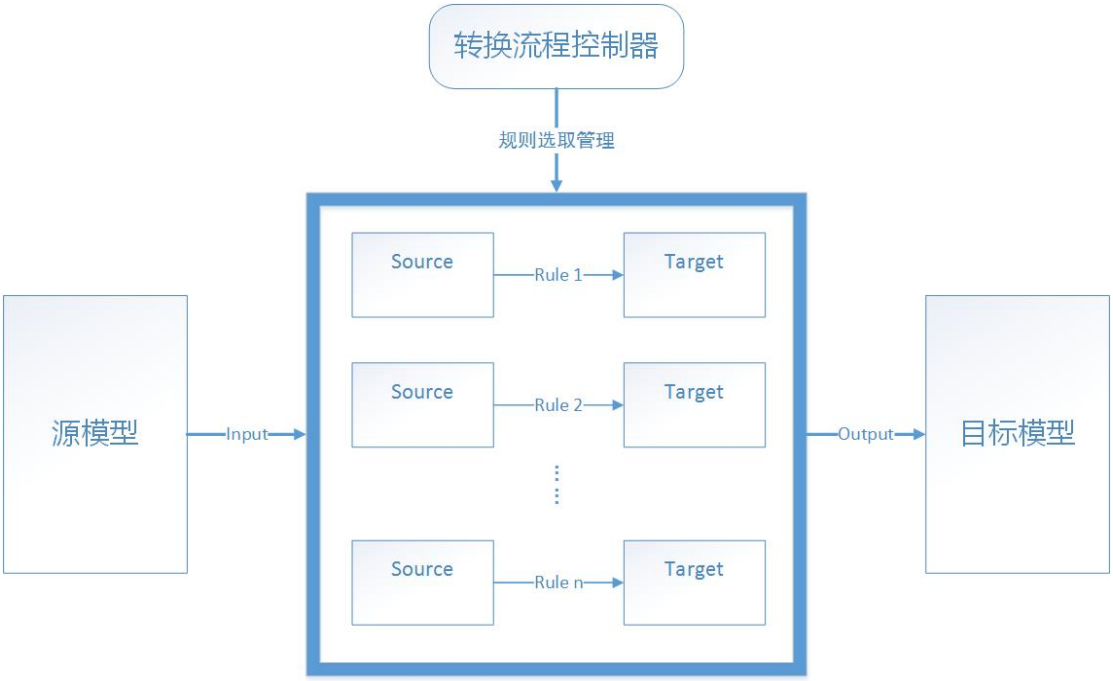

图 2-3 模型转换示意图

本文把数据清洗看作是从 Source 数据变换到 Target 数据，或者 Source Model 到 Target Model，因此可以用 MDE 领域的 Model Transformation 来处理这个问题。这样使整个框架层次明确，对功能划分思路清晰，对抽象模型和具体

模型更容易把握，设计基本操作的时候更容易理解。

2.3 CSV 文件格式

CSV 即 Comma Separate Values[11]，这种文件格式经常用来作为不同程序之间的数据交互的格式。具体文件格式有:

（1） 每条记录占一行；

（2） 以逗号为分隔符；

（3） 逗号前后的空格会被忽略；

（4） 字段中包含有逗号、换行符、双引号、空格，该字段必须用双引号括起来；

（5） 第一条记录可以是字段名。

CSV 文件在数据存储中应用广泛，且格式简单易处理，是本文欲处理的理想数据格式。

3 整体分析和设计

本章将对面向大数据清洗工具（H-BDCT）进行介绍，着重分析待处理数据所需的操作，为后面配置文件的设计和翻译引擎的实现提供参考，并根据整个工具的用途提供整体的框架设计和待清洗数据的流程图。

3.1 H-BDCT 概述

H-BDCT 通过用户的输入，自动生成一份可直接在集群上执行 Hadoop 代码，从而完成用户规定的数据清洗任务。该工具需要的输入文件简单，有很强的操作灵活性，涵盖了绝大部分数据清洗的操作，从而减少了用户使用 Hadoop 进行数据处理的困难，也大大提高了数据处理的效率。

H-BDCT 基于 Hadoop-0.20.2 版本开发，通过接收用户输入的一份预定义格式的配置文件，根据用户对数据清洗操作的要求，自动生成一份可执行 Hadoop 代码，该代码可直接打包放到 HDFS 上运行，利用 Hadoop 提供的集群并行化运算，极大提高了大数据的清洗任务的效率。H-BDCT 主要针对 csv 数据文件进行处理，因为 csv 文件在数据存储中应用广泛，且格式简单易处理，而且符合 Hadoop 默认的分割方式，可以视为一张二维表，便于分析处理，是较为理想的数据存储格式。

3.2 待处理数据的分析

待处理数据以 csv 文件格式存储在 HDFS 上，由于 HDFS 的特性，可能会被分割成一份份较小的数据块，但分割后的数据也是一张独立的二维表，可以由 Hadoop 集群从 HDFS 上获取和运算。由于待处理数据中存在各式各样如缺失值和噪声的杂乱数据，下面对待处理数据可能存在的操作进行分析，从而为设计配置文件和相应的功能提供参考。

1) 当缺失值处于该行的最后部分，会出现缺少相应分隔符的情况，这会影响数据格式的完整性，例如图 3-1 中第 1321 行等数据出现了缺省分隔符的情况；

```
1318  Center,436469,T,greater than,6000,238097071,,3500,18
1319  ,436503,T,greater than,6000,238114971,,1800,32
1320  ,436740,T,greater than,6000,238248158,,5500,24
1321  ,436788,F,,6000,238272472
1322  ,436789,F,,6000,238272472
1323  ,436828,F,,5500,238284642
1324  ,437067,F,,4000,238552725,,,15
1325  ,437078,F,,5000,238571968
1326  ,437104,F,,2400,238591876,,,32
1327  ,437121,T,greater than,6000,238599836,,4500,8
1328  ,437152,T,greater than,6000,238619516,,5500,27
```

图 3-1 分隔符缺失

2） 部分属性值可能不满足相应阀域而应该舍去该行数据，从而使留下来的数据更加有效，例如图 3-2 中选取第三个属性为 T 的数据来获取数据，则第 3、4 等行需要被舍掉；

```
2  ,433764,T,,2400,236330001,,1600,3
3  ,433618,F,,5500,236232728,,,31
4  ,433666,F,,5500,236266126,,,18
5  ,433804,T,greater than,6000,236370909,,5000,18
6  ,433817,F,,5500,236394560,,,15
7  ,433904,F,greater than,6000,236434446,,,1
8  ,433943,F,greater than,6000,236470365,,,6
```

图 3-2

3） 对于多张表的情况可能需要将其根据特定主键属性进行合并，以得到同一个事务不同表中对应的信息的综合，便于处理，例如图 3-3 和图 3-4 是同一个研究员做的两份天气信息，通过表合并可以获取气候和云层的对应关系；

```
99   281840373,240373631,Few clouds at 2600 feet. Sky cover is towering cumulus.
100  283704794,240371791,Few clouds at 3000 feet.
101  283704795,240371791,Broken layer at 10000 feet.
102  283704796,240371791,Broken layer at 14000 feet.
103  282370782,240376634,Few clouds at 2600 feet.
```

图 3-3

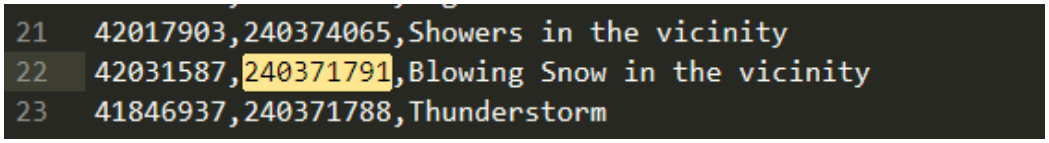

图 3-4

4) 有时在平滑噪声的时候需要用到特定属性的特征值，比如最大值、最小值、平均值等，例如图 3-5 中第五个属性中，需要获取其中最大值和最小值以查看其变化范围；

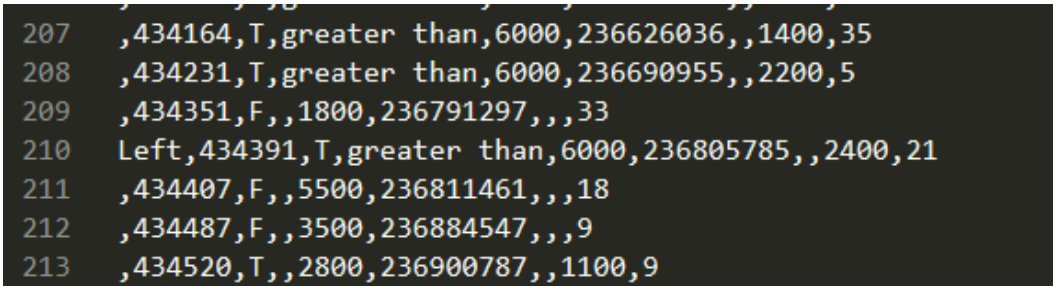

图 3-5

5) 对属性值的处理是数据清洗的重点，这其中包括单一属性值的数据处理，例如该列均以平均值代替等，以及对属性间的数据的处理，由于其中涉及的运算可能比较复杂，还需要提供一种灵活的方式来定义运算，例如图 3-6 中可以用属性运算查看每行数据中最大值和最小值差的信息；

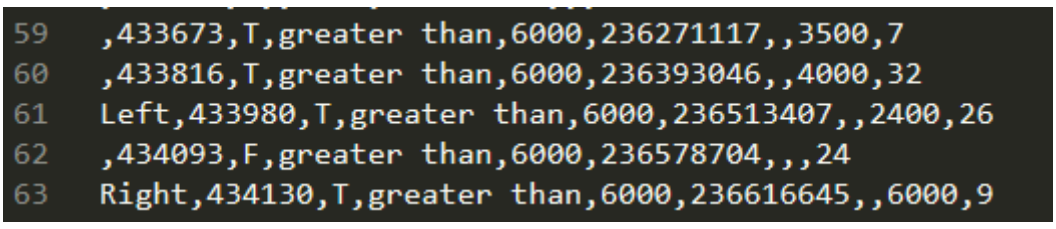

图 3-6

以上是对可能存在的数据操作的分析，而且每一种情况都应该在它前面的情况解决后才能得到有效解决，否则会出现很多处理不全面的问题，由此可见数据

清洗是很简单但非常繁琐的任务。而实现这样冗长且存在先后顺序的处理过程无法通过一次 MapReduce 过程完成。需要将其分拆为多趟简单些的 MapReduce 子任务进行处理。

3.3 工具整体框架设计

通过以上的分析和整理，本文提出了该工具的整体框架设计和待处理数据的数据流程图。本工具将由用户要处理的海量数据、用户输入的配置文件、配置文件翻译引擎以及 Hadoop 平台组成。本系统的整体设计框架图如图 3-7 所示。

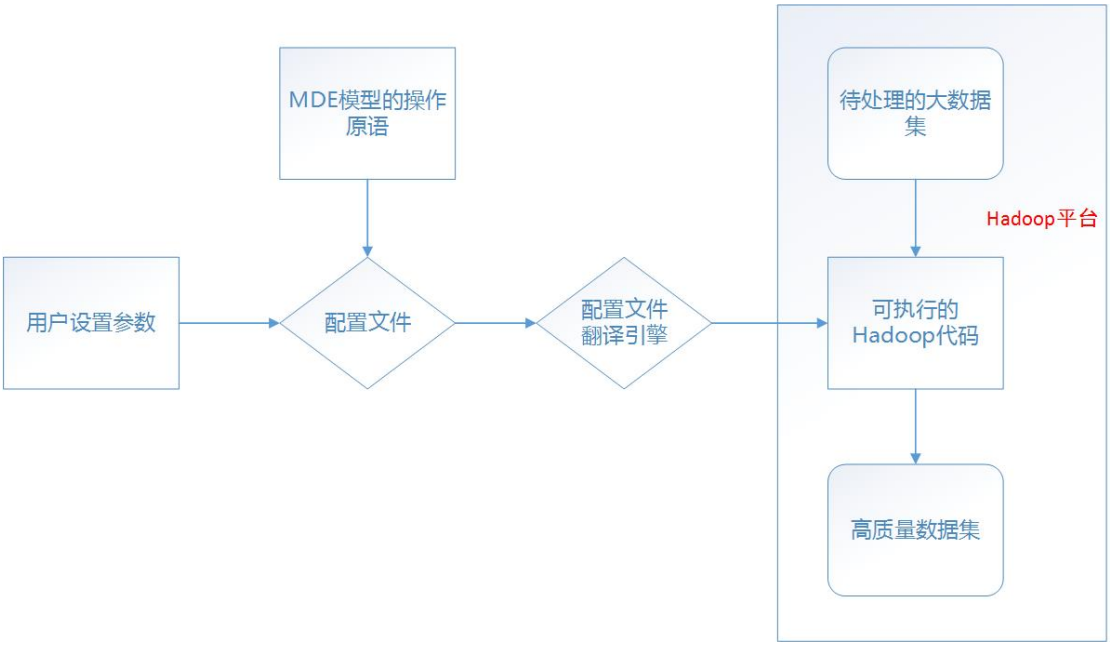

图 3-7 整体设计框架

从图 3-7 中可以获知该工具的整体设计逻辑并不复杂，基本上是单线路执行，但在 Hadoop 平台上数据处理流程比较复杂，如图 3-8 所示，显示了待处理数据的数据流。

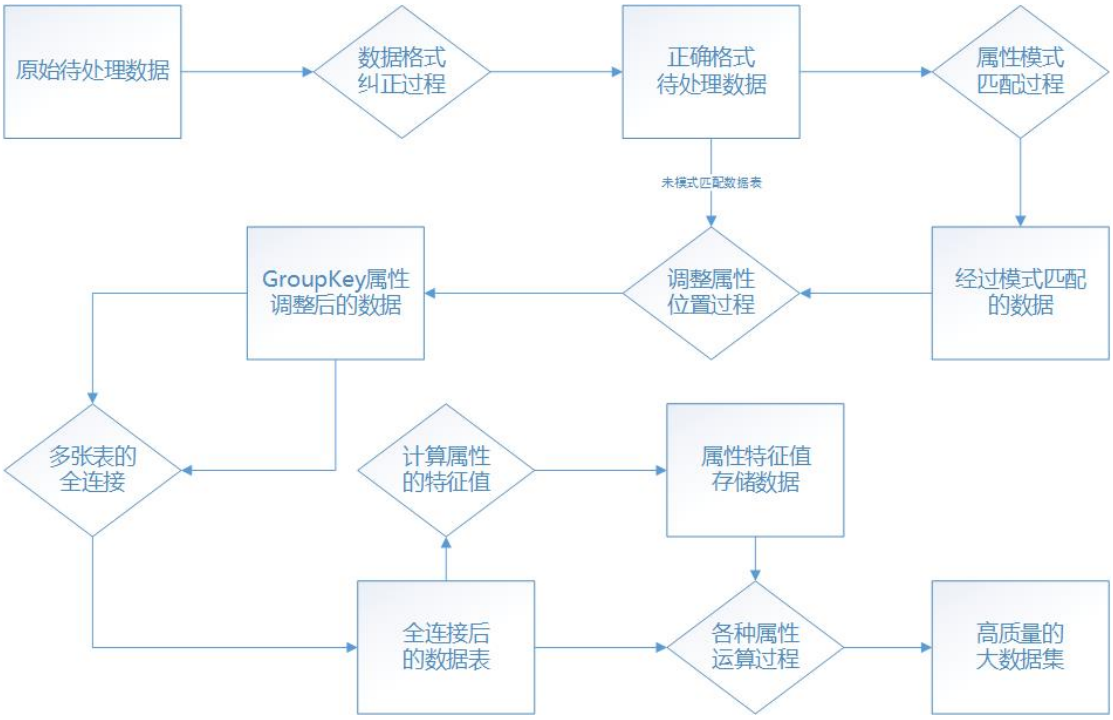

图 3-8 数据流程图

图 3-8 给出了在处理多张表涉及表合并情况下的数据流程图，由于不同的数据表可能会涉及不同的模式匹配以及属性特征值计算，所以在很多分支处进行分开处理，从而增加了数据处理的复杂程度。但这样的数据处理顺序是必要地，这样才能保证数据清洗的完整性和准确性。

3.4 本章小结

本章主要介绍了 H-BDCT 的运行环境和基本要求，对数据清洗任务进行了认真的分析，并提出应该提供的基本操作。在此基础上设计了整个工具的框架内容，并给出待处理数据的数据流程图，从而给出了直观明了的处理流程，有助于对 H-BDCT 有整体的认识。

4 配置文件的设计

本章从 MDE 的视角来看待数据清洗的问题，将之视为模型转换的一种特定应用。借助于 QVT 等模型转换规范中提到的基本转换操作的思想抽象出 create，update 等对数据集的基本操作，并以此来指导配置文件的设计。具体包括对 create 等操作原语分析，以及针对 3.2 节提出的待处理数据所需要提供的操作，将 create 等基本操作进一步细化，从而设计出有效的配置文件，为配置文件翻译引擎提供有力的支撑。

4.1 配置文件介绍

配置文件用来规定一些程序在启动时读入设定，给用户提供了一种修改程序设置的手段。本工具以“命令，参数”的形式定义，即每一行都包括一个关键字，以及一个或多个参数。其中命令是相应的关键字，是用来表示用户所要用到的操作指令，参数是在该操作指令下需要用到的一些信息值，比如处理属性的列数等等。又因为数据清洗的操作有顺序要求，故要求配置文件的命令从上向下按序输入。这样的配置文件格式简单，容易理解，不存在二义性，满足需要用户输入的的易用性。

4.2 介绍操作原语和分解细化

数据清洗任务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。我们从 MDE 的视角来分析数据清洗任务，将之视为模型转换的一种特定应用。并借助于 QVT 等模型转换规范中提到的基本转换操作的思想抽象出 create，delete，update，match 和 expression 这五个基本操作。下面分别对这五个基本操作进行介绍并将其进一步分解出面向 Hadoop 操作的原子操作，从而为配置文件提供参考。

4.2.1 Create 操作原语

Create 操作原语的基本语义是创建元素，也是 update 原语操作的组成部分。

在数据清洗任务中 Create 操作原语可以被用来预处理属性间的简单关系，从而提供更加有效地属性数据，简化后续统计等工作的任务。例如将两列相关属性以一定比例加和，从而得到一个新的属性，而这个新属性可能使用起来更加有效。从上例中可以看出 create 操作原语在向 Hadoop 平台上映射的时候需要进一步分解其功能。

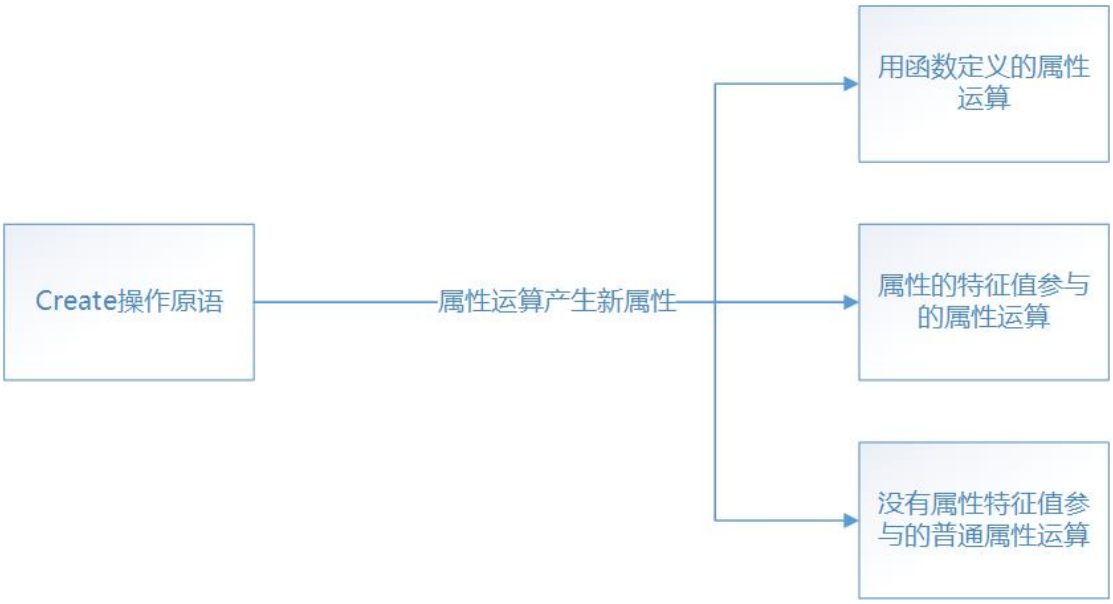

图 4-1 Create 原语转换规则

图 4-1 给出了 create 原语面向 Hadoop 平台分解后的原子操作分类，为了提供更灵活的属性生成，应该提供用函数来定义属性运算的功能。另外这样的操作也为数据表的合并提供了可能。而由于数据量过大和 MapReduce 过程的限制，对于属性的特征值，例如属性最大值和平均值这类的求解必须单独作为一个过程来求解，所以这里区分了是否有属性特征值的情况。这样就要求在原子操作中需要提供表合并、函数处理、属性特征值运算和普通属性运算。

4.2.2 Delete 操作原语

Delete 操作原语的基本语义是删除元素，也是 update 原语操作的组成部分。在数据清洗任务中 Delete 操作原语可以被用来去除无效属性，有效地压缩输入数据的维数，提高数据的集成性和有效性，减轻后续数据运算的压力。例如关注

信息与时间属性的关系不大，我们可以将时间属性去掉，减轻运算压力。将此例结合 HDFS 的实现思想可以得知，在原数据的基础上删去数据是不被接受的，故我们需要新建数据表，在建数据表的同时将需要删除的数据忽略掉，从而达到删去相关属性的效果。

4.2.3 Update 操作原语

Update 操作原语的基本语义是更新元素，实际上是由 create 和 delete 组合实现。在数据清洗任务中 Update 操作原语可以被用来更新属性值，将之前冗长低效但有简单数据关系的属性集合合并运算，从而得到更加有意义的数据。例如将两个可按一定比例组合形成新的有效属性的属性合并，并将这两个旧属性删除。这样的操作用 create 和 delete 原语实现，所以不再展开讨论。

4.2.4 Match 操作原语

Match 操作原语的基本语义是匹配操作。在数据清洗任务中 Match 操作原语可以被用来对属性值进行取舍，从而获取需要的内容，是不可或缺的一个步骤。例如为了研究某属性为 F 时各个属性的特征，就需要提取该属性为 F 的所有数据从而进行分析。从此例可以看出 match 原语在向 Hadoop 平台映射时要做进一步的细化。

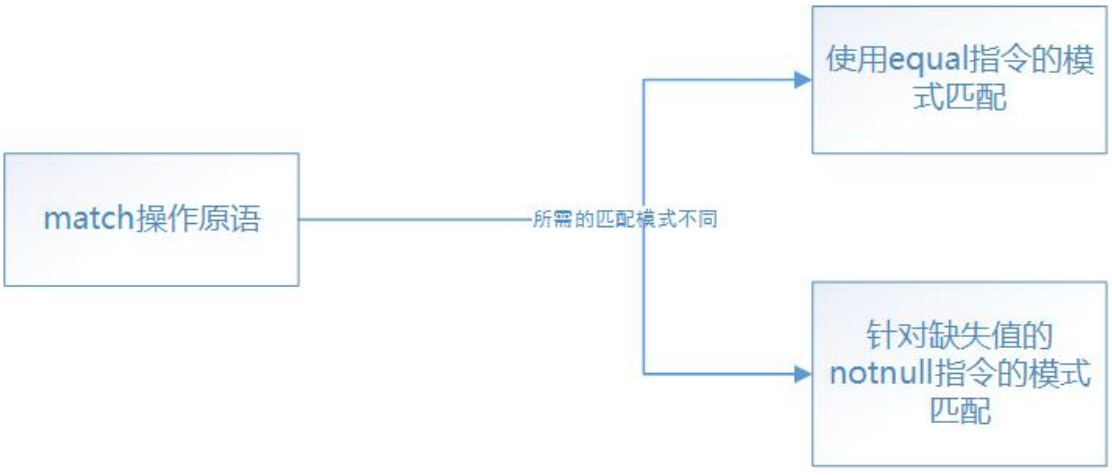

图 4-2

图 4-2 显示了面对不同的模式匹配，match 原语分解后的原子操作需要提供几种常用的匹配模式，例如针对缺失值的舍去处理，可以提供 notnull 指令来完成，针对特定属性值可以提供 equal 指令来完成匹配功能。

4.2.5 Expression 操作原语

Expression 操作原语的基本语义是表达式计算。在数据清洗任务中 expression 操作原语是核心的一部分，可以完成例如噪声平滑、数据一致性运算等功能。例如将指定属性的属性值统一增加 30%，将缺失的判定属性统一设置为 true，将超过一定范围的数据替换为范围的最大值等等。从此例中可以看出 expression 原语在面向 Hadoop 的原子操作需要有很大的灵活性。

图 4-3 Expression 原语转换规则

图 4-3 展示了 expression 中可以分解的操作，其中属性特征值的计算无法和普通运算合并到一起，这是因为属性特征值需要对整个数据集进行一次遍历，而这个遍历无法和普通运算同时进行，这样普通运算就不能及时得到相应的属性特征值。而函数操作需要用户自己提供，而且保证其准确性运行，这样就能提供最大限度的数据清洗操作。

4.3 配置文件命令的详细介绍

配置文件使用“命令，参数”形式定义，也就要求提供配置文件命令以及相应的参数信息，从而有效地表达用户所需的操作。3.2 节提出的各种操作要求，例如数据清洗任务之前需要调整数据格式，提供同一属性不同表的合并等等，以及从 MDE 角度抽象出的 create 映射出的属性特征值运算、函数等操作为配置文件的设计指令提供了有效的参考，并最终决定配置文件提供 9 种配置文件命令，分别是 FilePath、Function、PatternMatch、TableJoin、Max、Min、Average、HandleAttribute、End，其中 Max、Min、Average 命令类型相似，均用来求解属性值的特征值。图 4-4 是一个涉及各种操作的配置文件样例 Demo，下面将逐一介绍每个配置文件命令。

```
FilePath,2
1,3,String,Int,String
2,3,String,Int,String
Function#
static String handle(String s) {
    return s;
}
#
Function#
static String handle1(String s) {
    return s+"_user";
}
#
PatternMatch,2,2,equal,1
TableJoin,2,2
Max,1
Min,1
HandleAttribute,1,Max_1+Min_1,Max_1,Min_1
HandleAttribute,2,handle(strArray[2])
End
```

图 4-4 配置文件 Demo

4.3.1 FilePath 配置文件命令

该命令表示数据清洗的数据表的数量和信息，位于所有配置文件命令的开始

部分。该命令的参数 x 只有一个，表示待处理表的数目。如果 x 为 1，表示整个数据清洗的过程是针对一个表进行的，在后面一行里需要放置待处理的表的信息；如果 x 大于 1，表示数据清洗过程中会存在表合并的操作，在该命令行后的 x 行需要按表合并过程中属性放置的先后顺序放置待处理表的信息。待处理表的信息依次是表名称、属性个数、各属性的类型，同时对可能因缺失值的存在而导致分隔符的缺失的问题可以在这里隐含的得到解决，具体措施见第五章翻译引擎实现部分。以上信息有助于修整待处理数据格式，为后续的操作提供完整、有效地输入数据。

4.3.2 Function 配置文件命令

该命令用于提示处理在后面属性运算过程中复杂运算函数的实现，由于需要在属性值处理之前获取全部的运算函数，故所有的插入函数要紧跟在 FilePath 命令后面。该命令没有参数，主要在以后两个连续的“#”间会有一个完整可用的 java 函数，该函数在属性值处理的过程中对相应的属性列进行运算。因为这段函数代码在实现过程中是直接贴到对应位置的，所以实现的时候要能够在正常的 java 工程中运行。另外由于属性运算中函数可能会有很多，故在配置文件中该命令的个数不受限制，但每个函数都需要用“Function#……#”打包。

4.3.3 PatternMatch 配置文件命令

该命令用于对特定数据文件的指定属性进行模式匹配，即根据数据清洗任务的对该属性值的限制决定是否留下该行数据，该数据应该在 TableJoin 命令前执行，这样才能保证进行表合并的时候数据都是有效可用的。该命令有较多的参数，包括待处理表名称、要做模式匹配的属性所在列、模式匹配的指令（目前支持 equal 和 notnull），以及模式匹配指令的参数。由于可能会对多个属性执行类似的模式匹配操作，所以该命令执行的次数不受限制，也可能对所有的属性都不执行模式匹配，故该命令也可以在数据清洗过程中不执行。

4.3.4 TableJoin 配置文件命令

该命令用于对组成输入数据的多张表按给定属性作为 Key 做的全连接，这样形成的连接表包含被连接的表的所有记录，如果缺少匹配记录，将以 NULL 填充。该命令在前面讲述的命令完成后即可使用，如果输入数据的文件夹中只有一张表，那应该不会存在表合并的情况，而如果该输入数据的文件夹中包含多张表的话，默认是一定会存在表合并操作的，否则多张表无法处理成一个最终的输出文件。该命令包含 x 个参数，其中 x 是参与表合并的个数，这 x 个参数均是各个表中作为 Key 属性的列值，其顺序和 FilePath 命令中表出现的顺序相同。参与表合并的表的个数不受限制，但合并的 GroupKey 必须是相同的，也即在配置文件中表合并的命令只能出现一次，当然在一张表的情况下也可以不执行表合并的命令。

4.3.5 属性特征值运算配置文件命令

该类命令包含 Max、Min、Average 命令。在属性运算的过程中会用到这类特征值，而这类特征值在分布式运算的条件下是必须使用一个 MapReduce 过程来处理的，为了不在属性运算时无值可用，我们需要提前运算这类需要独立 MapReduce 过程运算的值。由于这类数据的运算是基于前面内容都铺垫好才能执行的，故该命令只能在属性运算命令之前使用。可以从数据流程图 3-8 看出到这一步的时候应该只剩一张表了，所以该命令的参数不需要添加表名称了，所以只有一个参数，也就是待求解的属性列值。该命令可被多次执行，另外该属性的特征值运算需要用户自己提取，是不太理想的一点。

4.3.6 HandleAttribute 配置文件命令

该命令是最终实现属性运算的命令，它支持直接的表达式运算和预置函数运算（Function 命令中携带的函数），是非常理想的数据清洗命令。这类命令用于得到最终数据集，故应该在配置文件的结尾部分，即只在 End 命令之前。该命令有两个主要参数，第一个参数代表属性运算后应存放在新数据集的新列值，第二个参数是属性运算的运算公式或函数，但暂不支持运算公式和函数的混合使用。为了有效地提取信息，我们要求对新数据集的每一列属性都再次定义，可以直接

将表示该行的信息放到第一个参数的位置。另外在第二个参数为运算公式的情况下，如果式中有属性特征值，需要对其进行特殊的命名以方便工具提取信息，该命名规则是“属性特征值类型+下划线+属性列值”，例如“Max_1”，“Min_2”。另外在式中有属性特征值，需要在该参数后添加相应名称的参数，而且满足前面所说的命名规则，这样会简化 Hadoop 平台上的实现。当需要用到属性的时候，需要用“strArray[x]”代表，其中 strArray 是内部的一个参数，是不能修改的，x 表示在属性运算前的表中属性所在的列值。这是一处比较不太理想的部分，但是目前这里还没有很理想的解决办法，也会在实现中说明为何会出现这种情况。

4.3.7 End 配置文件命令

该命令表示整个运算过程结束，将把运算的结果汇总起来进行最后的处理。显而易见该命令没有任何参数，也应该处于配置文件的最后的位置。

这样设计的配置文件内容清晰，格式严谨，操作灵活，易于生成，为用户填写或者上层模型语言的自动转换生成提供了很大便利，能满足数据清洗任务的各种操作要求。同时，为配置文件的代码翻译引擎的实现奠定基础，也有利于进一步的功能扩展。

4.4 本章小结

本章首先对配置文件进行了简要的介绍，并提出了本工具所要用到的配置文件的基本形式。然后从 MDE 角度对数据清洗任务进行了分析，抽象出 create、delete 等操作原语，对它们进行了简要介绍和进一步的分析，从而进一步转换分解成面向 Hadoop 平台的具体操作，从而提供了从模型到具体配置文件命令的转换规则，为上层模型语言的自动化生成可用执行代码提供了巨大便利。接着详细介绍了配置文件命令的功能以及参数，从而为用户提供了简单有效的途径完成数据清洗任务。

5 由配置文件生成 Hadoop 代码引擎实现

由配置文件生成 Hadoop 代码是本文工作的一个实现重点。本章将简要介绍 Hadoop 的 MapReduce 编程接口，从而对 MapReduce 编程有一个直观的了解。具体再根据配置文件的各条指令进行分析，得到每次 MapReduce 过程所需做的操作，完成基本的 MapReduce 过程的模板，并根据配置文件命令对模板进行修改，从而得到在大集群上完整可执行的 Hadoop 代码。

5.1 Hadoop 的 MapReduce 编程接口

Hadoop 实现了 Google 的 MapReduce 编程模型和计算框架，采用“分而治之”的思想，把对大规模数据集的操作分发给一个主节点管理下的各分节点共同完成，然后通过整合各分节点的中间结果，得到最终的结果。上述处理过程被 MapReduce 高度抽象为两个函数：map 和 reduce，map 负责把任务分解成多个任务，reduce 负责把分解后多任务处理的结果汇总起来。从这里也能看出，用 MapReduce 来处理的数据集必须有这样的特点：待处理的数据集可以分解成许多小的数据集，而且每一个小数据集都可以完全并行地进行处理。下面将分别对 Map 过程和 Reduce 过程以及执行 MapReduce 任务进行介绍。

5.1.1 Map 过程

Map 过程需要继承 Mapper 类，并重写其 map 方法。也就是说在实现 Map 过程时，如果没有特殊需要，只需要重写一个 map 方法即可。用户自定义的 map 方法接受一个输入的 key/value pair 值，然后产生一个中间 key/value pair 值的集合。Map/Reduce 库把所有具有相同中间 key 值的中间 value 值集合在一起后传递给 Reduce 过程。下面是一个对单词计数的 Map 程序的伪代码实现：

```
1.  Method Mapper(S key, S value)
2.        for each word w in value
3.              write(w, 1)              //对每个单词，输出 1 次计数
4.        end
```

从伪代码中可以看出在默认的输入格式中，value 中会保存所有的信息，而

如果不需要行号信息的话，key 值是可以无视的。在 map 方法中输出了文档中每个词、以及这个词的出现次数（在上面的简单例子中就是 1），这样就实现了其 map 的分散，而且保证了数据间不存在依赖性，可以在大型集群上运行。

5.1.2 Reduce 过程

Reduce 过程需要继承 Reducer 类，并重写其 reduce 方法。和 Map 过程类似，如果没有特殊的需要，只需要重写一个 reduce 方法即可。Reduce 方法的输入参数是从 Map 过程产生的中间数据对，从上节给出的例子可知输入参数 key 为单个单词，而 values 是由各 Mapper 上对应单词的计数值所组成的列表，所以只要遍历 values 并求和，即可得到某个单词出现的总次数。并将数据按 key 和 value 的形式输出到文件，完成任务。下面是相应的 Reduce 程序的伪代码：

```
1.  Method Reducer(S key, Iterator values)
2.      int result = 0
3.      for each v in values //  内部已实现按 Key 聚类
4.          result += v      //  累加相同单词的计数
5.      end
6.      write(result)
```

从 Reduce 过程的伪代码中可以很明显理解上面的内容，reduce 方法就是把 map 方法产生的每一个特定的单词的计数累加起来并输出到文件中。由此也可以理解 MapReduce 只能够完成相对简单的数据块处理任务。

5.1.3 执行 MapReduce 任务

在 MapReduce 中，由 Job 对象负责管理和运行一个计算任务，并通过 Job 的一些方法对任务的参数进行相关的设置，主要包括对 Mapper 类和 Reducer 类的设置，对输入数据的设置和对输出数据的格式和位置的设置。完成相应任务的设置设定后，即可调用启动方法执行任务了。下面给出主函数的伪代码：

```
1.  void main(String[] args)
2.      Job job = new Job()
3.      job.setMapperClass(testMapper.class)
4.      job.setReducerClass(testReducer.class)
5.      FileInputFormat.addInputPath(job, new Path(args[0]))
6.      FileOutputFormat.setOutputPath(job, new Path(args[1]))
```

MapReduce 任务可以迭代执行，可以组合顺序执行，也可以具有复杂依赖关系的组合式执行，这些执行方式不同在于其主函数的参数设置不同而不同，这需要根据任务的具体情况选择合理的求解方式。在本文的实现中，由于操作之间不存在复杂的依赖关系，故使用简单的组合顺序执行。以上是一个简单的 MapReduce 过程的伪代码，了解这些基本知识对理解下面实现过程有很大帮助。

5.2 配置文件翻译引擎的实现

从上一节内容可以看出 MapReduce 框架的总体结构稳定，适合作为代码模板，为从配置文件映射到可执行的 Hadoop 代码提供了极大地便利。本节翻译引擎的实现思路便是利用大部分固定的 Hadoop 代码模板，根据配置文件的命令和参数对代码模板中关键部分的调整得到一份可执行的 Hadoop 代码。下面将以第四章配置文件 Demo 中的各条命令为例讲解翻译引擎的实现。

5.2.1 FilePath 命令

该命令中的参数中包含表的名称、属性个数等信息，这样可以完成对部分表缺失分隔符问题的修复，而这需要通过一次 Map 过程来完成，因为这个任务不需要汇总某项数据，所以不需要 Reduce 过程。另外因为本文的实现使用的是串行的 MapReduce 过程，所以每次的结果可能就是下次的数据输入，需要将输出和输入的文件路径记录和更新以备下一步使用。

数据格式修复任务中 Mapper 的实现需要知道数据表中属性的个数 c。实现中为了减少对模板的暴力替换，提高该 Mapper 类模板的通用性，使用了 Hadoop 提供的全局作业参数从而可以从主函数传入属性个数 c。为了在 map 方法中使用到属性个数 c，需要使用 setup 方法先将全局作业参数获取并存入该类的一个整型值 numAttr 中。Setup 方法是在 Mapper 类中先于 map 方法调用前调用的。Map

方法中，计算每一行的数据中的分隔符，对于分隔符个数不够的行直接添加缺少的分隔符，并输出到新的文件中。

数据格式修复任务中主函数需要设置全局作业参数，设置 Mapper 类，设置输出格式，设置输入输出的路径信息，并提交作业。为了控制每个 job 完成后再执行下一个 job，这里需要对 waitForCompletion()设置为 true。下面给出主函数部分的关键代码：

```
globalConf.setInt("MR_Num_Attributes", 3);
Job job0 = new Job(globalConf, "add comma");
job0.setJarByClass(MapReduceTest.class);
job0.setMapperClass(AddComma.class);
job0.setOutputKeyClass(Text.class);
job0.setOutputValueClass(Text.class);
FileInputFormat.addInputPath(job0, new Path(args[0] + "/1"));
FileOutputFormat.setOutputPath(job0, new Path(args[1] + "/temp00"));
job0.submit();
job0.waitForCompletion(true);
```

后面其它命令的主函数也是类似的设置，一般只有设置输入输出格式、文件位置以及 Mapper 类和 Reducer 类不同，故后续不再赘述。

5.2.2 Function 命令

该命令主要是为了向属性运算提供函数支撑，其中包括的函数内容会被统一写入一个独立的文件中，在实现 HandleAttribute 命令的 Mapper 类时会读取该独立文件并完整拷贝到 Mapper 类中，以提供函数实现。另外因为没有对函数的任何识别和修改，用户必须保证函数的准确性，否则无法提供有效地手段保证结果的准确性，所以这一部分并不对应 MapReduce 代码。

5.2.3 PatternMatch 命令

该命令要实现模式匹配功能，目前支持 equal 和 not null 两种模式匹配。该命令的参数列表中有表名称、属性列值和匹配指令以及可能存在的匹配指令参数。由于模式匹配提供的是对数据的筛选，并不需要汇总数据结果，故只需要 Map

过程即可完成任务。

数据的模式匹配任务中 Mapper 的实现需要知道表名称、属性列值以及匹配指令等参数，所以参数的传递选择将配置文件的参数字符串传入，在 Mapper 中分解并获取所需参数信息。同样也需要用到全局作业参数，不过这次传入的是字符串信息，并在 setup 方法中处理得到一个字符串参数数组。在 map 方法的实现中，将输入数据分割成一个个属性值，需要根据匹配指令做相应的操作，并将处理结果输出到指定文件中即可。该代码模板中没有需要替换的内容，操作也相同，故在整个工程中该 Mapper 类唯一，不需要再次生成。下面给出了 map 方法中的关键代码实现：

```
if (instr.equals("equal")) {
    if (strArray[attr-1].equals(ss[2])) {
        word.set(s);
        context.write(word, new Text(""));
        return;
    }
} else if (instr.equals("notnull") && !strArray[attr-1].equals("")) {
    word.set(s);
    context.write(word, new Text(""));
    return;
}
```

5.2.4 TableJoin 命令

该命令实现多张数据表对指定属性的全连接功能。它有于和表个数一样多的参数，是表中作为 GroupKey 属性的列值，默认按 FilePath 命令中文件的顺序排序。这部分操作分为两个阶段来完成，第一个阶段是将作为 GroupKey 的属性调整到第一列去，方便整个过程的操作使用；第二个阶段是将多张表的信息合并到一张最终的表上。

该命令的第一个阶段是对多张表属性列的调整，不需要汇总数据，故只需要 Map 过程即可完成任务。该阶段的 Mapper 类中只需要获取要调整到第一位的属性列值。和前面的操作相同，需要将属性列值作为全局作业参数传入 Mapper 类，并在 setup 方法中获取，最终在 map 方法中将数据分割，并做位置的调整，从而实现了 GroupKey 属性调整到第一列的任务。

该命令第二个阶段的实现用到了 Hadoop 自带的 Datajoin 类库[14]。Datajoin 类库是 Hadoop 的 MapReduce 框架提供的一种较为通用的多数据源连接方法，它为编程人员提供了完成数据连接所需的编程框架和接口，简化数据连接处理时的编程实现。为了能完成不同数据源的连接，首先需要为不同数据源下的每个数据记录定义一个数据源标签（Tag）。进一步，为了能准确标识一个数据源下的每个数据记录并完成连接处理，需要为每个待连接的数据记录确定一个连接主键（GroupKey）。然后，Datajoin 类库分别在 Map 阶段和 Reduce 阶段提供一个处理框架，大大简化了需要处理的工作，只需要实现必须由编程人员来完成的部分。

在第二阶段的 Map 过程中，Datajoin 类库提供了一个抽象的基类 DataJoinMapperBase，该基类已实现了 map()方法。需要编程人员处理的是每个数据源的标签 Tag，用哪个属性作为连接主键 GroupKey。Map 过程结束后，这些确定了标签和连接主键的数据纪律将被传递到 Reduce 阶段做后续的结果。Map 阶段处理的示例如图 5-1 所示。

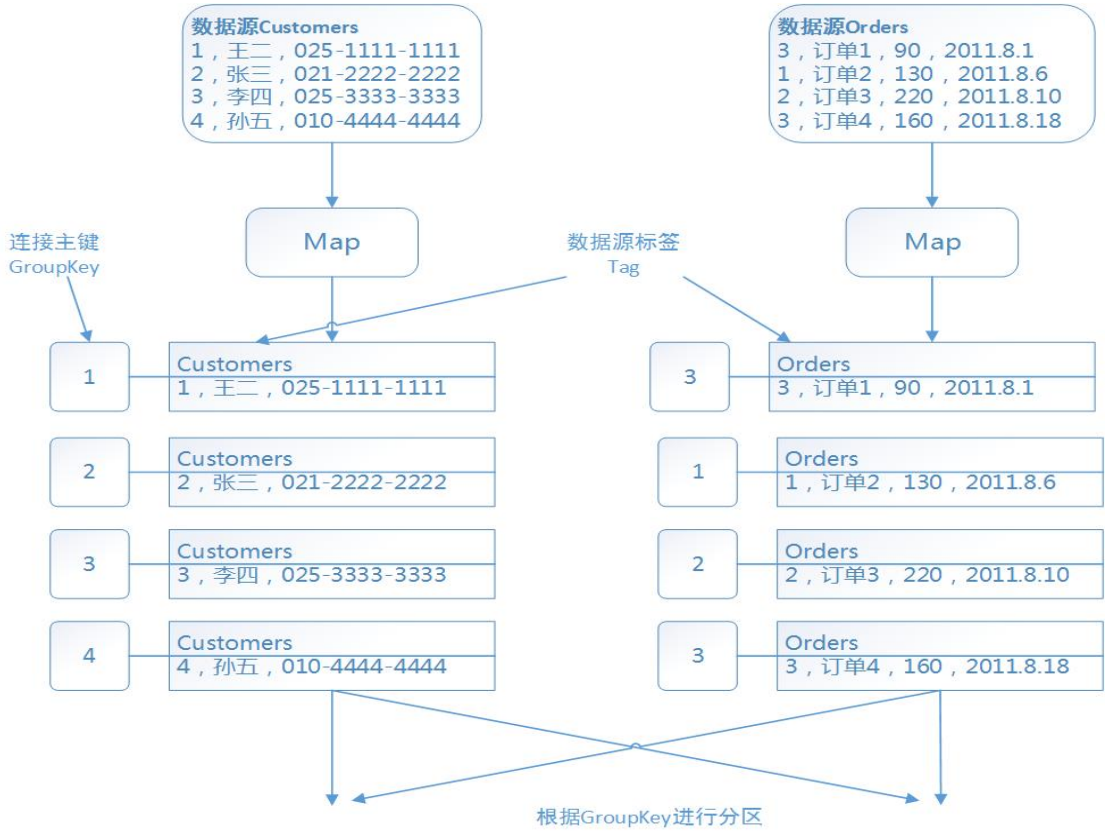

图 5-1 DataJoin 的 Map 过程

经过以上的 Map 过程处理后，所有带标签的数据记录将根据连接主键 GroupKey 进行分区处理，因而所有带相同连接主键 GroupKey 的数据记录将被分区到同一个 Reduce 节点上。

Reduce 节点接受到这些带标签的数据记录后，将对不同数据源标签下具有同样 GroupKey 的记录做笛卡尔叉积，自动生成所有不同的叉积组合。然后对每个叉积组合，由编程人员实现一个 combine()方法，根据数据清洗的要求将这些具有相同 GroupKey 的不同数据记录进行适当的合并处理，最终完成类似关系数据库中不同实体数据记录的连接。Reduce 阶段处理示例如图 5-2 所示。

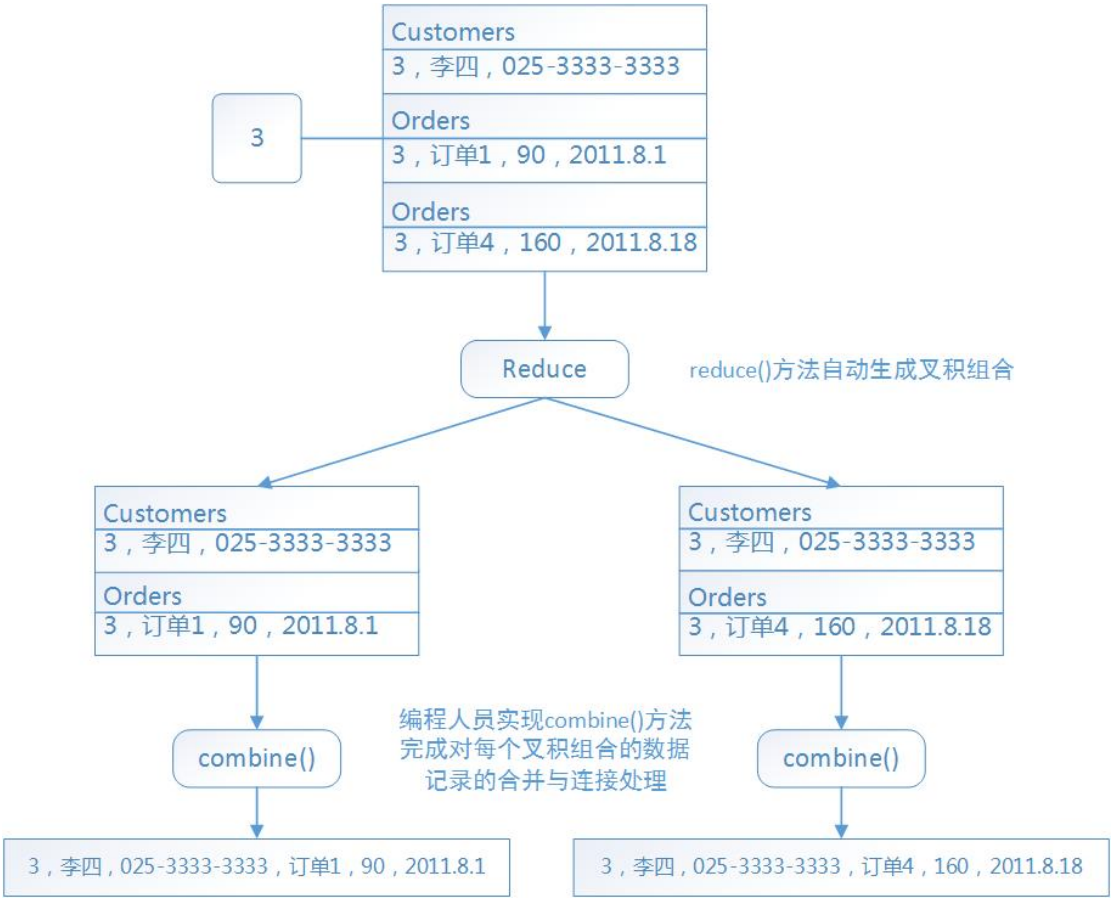

图 5-2 DataJoin 的 Reduce 过程

经过以上的 MapReduce 过程，表合并的任务即可被完成，并生成新的数据表并输出到指定文件中，为下一步的操作提供有效地数据集。

5.2.5 属性特征值运算命令

该命令用于求属性的特征值，因为在一个 MapReduce 过程中，没有办法求解在运算表达式中的属性特征值，从而无法完成运算表达式，故需要提前将这些特征值计算出来。该命令只有一个参数，即要计算属性特征值的属性列值。由于该命令最终的结果需要汇总结果，所以需要 Map 过程和 Reduce 过程。下面以 Max 命令为例讲解如何实现。

在属性 Max 特征值求解的实现上，因为数据集大小超过了基本数据块的大小，会被 HDFS 划分为多份数据并存到不同的 DataNode 节点上，所以为了获取全局最大的属性值，就需要现将每个 DataNode 节点上的最大值找到，再汇总到一个 Reducer 的 DataNode 上找到全局最大的属性值。而因为需要在所有 map 方法结束后才能得到该 DataNode 节点上的最大值，所以需要调用 cleanup 方法，该方法是在 Mapper 或 Reducer 过程的最后调用的，在该方法中将最大值写到中间文件中，从而在 Reducer 类中即可获得所有节点的最大值，从而再进行一次类似的运算即可得到属性特征值的最大值。同理，最小值也是如此。

而对于 Average 命令来说，情况可能有所不同，Average 需要将每个 DataNode 节点的值加和，并将该节点属性的个数加和，再将这个数据写入中间文件，Reduce 过程再将各个节点的总数再次加和，并求解平均数。

因为使用的分布式系统不支持全局变量的使用，这里需要将最终的结果写到自定义全局文件中，每次使用这样的值的时候再去文件中读取。

5.2.6 HandleAttribute 命令

该命令完成属性运算的功能，其中包含了对新属性的生成，对旧属性的舍弃以及对属性间运算和函数运算的支持，从而有效地支持了数据清洗任务的种种要求。该命令的参数包含新属性所在列值，属性生成所需的运算表达式以及使用属性的特征值的情况下，需要列出所使用的属性特征值名称。因为过程不需要汇总数据，所以不需要 Reduce 过程来综合信息。

属性运算的实现比较复杂，目前在表达式计算中使用属性特征值，暂不支持同时使用函数和运算表达式。而因为这些运算表达式都在 map 方法中使用，所以

不同的属性运算生成不同的 Mapper 类。这也就要求主函数需要根据不同的运算设置不同的 Mapper 类。

如果在表达式中使用到属性的特征值，那该特征值必须要满足一定的命名规则（第四章已说明），以帮助该工具识别，这样就可以读取上一节存入文件的特征值。在获取该特征值后，再使用转换函数将属性值字符串转换为相应的数据并参与数值运算。在 map 方法中，需要将属性生成表达式填入待替换的位置，这样在代码执行时不会出现问题。如果使用函数来生成新的属性，就需要将 Function 命令给出的函数写入该过程中的函数实现部分。其它部分和表达式的实现类似。

5.2.7 End 命令

该命令是整个数据清洗任务的最终部分，由于在 HandleAttribute 过程中运算要保留之前的表信息，所以其结果中还会存在相应的冗余信息，故在这一步主要实现了将旧数据舍弃，并留下新数据的功能。该过程不需要参数，可以从之前的操作中获得无用属性的列数，在 Map 过程中将其消除即可。而 Mapper 类的实现就会比较简单，对每次读入的数据，将前面列的数据忽略，只记录新生成的属性值即可。

5.3 本章小结

本章首先对 MapReduce 的编程框架进行了简要介绍，并选取了一个简单示例给出了伪代码实现进行分析，从而可以有效地了解 MapReduce 代码的结构。然后本章详细介绍了为完成数据清洗任务的配置文件的翻译引擎实现，其中主要包括数据表的全连接实现，Datajoin 类库的使用，属性运算等七部分。经过这些步骤，从 create 等操作原语到配置文件，再从配置文件到 Hadoop 代码的转换规则全部清晰，并可以应用到实际项目中。

6 实例研究

本章将使用一个简单具体的实例来详细的展示如何使用本文设计实现的数据清洗工具完成数据清洗任务。

6.1 实例研究

本实例涉及为数据清洗设计的各个操作，将这些操作完整执行一遍至多遍，即显示出工具执行的正确性，也为相应的操作提供了样例可参考。该实例中使用的数据比较简单，目的是为查看数据变化清晰明了。使用的配置文件是第四章中介绍配置文件命令用到的配置文件 Demo。该实例使用的 Hadoop 平台版本是 Hadoop 0.20.2 版本。

首先本实例的基本数据集有两个 csv 数据文件，分别命名为 1 和 2，其数据内容分别如下所示：

a,1,a b,2,b	aa,1,aa bb,1,bb cc,2

之所以使用较简单的数据集是简单数据集的变化容易发现，这样验证 H-BDCT 的各个步骤的准确性，也可以直观的感受到每次操作之后数据信息的变化。再将数据信息上传到 HDFS 上。

首先我们需要定义对以上数据的操作，配置文件 Demo 中的操作顺序和内容就涵盖了对数据清洗的各个方面，这里不再贴出该配置文件内容，只对其操作内容进行简要讲解。该配置文件 Demo 主要包括数据格式的补全，为属性运算过程添加的函数，对 2 文件中第二个属性值不为 1 的数据记录舍去，然后将两张表均以第二属性为 GroupKey 进行全连接，再在新的数据表中求取第一属性的最大值和最小值，之后再进行两次属性运算，产生两个新的属性集。

将设置好的配置文件放到转换引擎的输入路径处，调用配置文件的翻译引擎，将生成完成各个功能的 Hadoop 代码，另外还有将这些 MapReduce 过程顺序串行合并到一起的主函数文件。再将 Hadoop 代码打包成可用的 jar 包，此时就可以

在 Hadoop 平台上进行数据清洗任务了。

```
join.jar in out0
14/05/24 22:34:39 INFO input.FileInputFormat: Total input paths to process : 1
14/05/24 22:34:40 INFO mapred.JobClient: Running job: job_201405242230_0001
14/05/24 22:34:41 INFO mapred.JobClient:  map 0% reduce 0%
14/05/24 22:34:49 INFO mapred.JobClient:  map 100% reduce 0%
14/05/24 22:35:01 INFO mapred.JobClient:  map 100% reduce 100%
14/05/24 22:35:03 INFO mapred.JobClient: Job complete: job_201405242230_0001
14/05/24 22:35:03 INFO mapred.JobClient: Counters: 17
14/05/24 22:35:03 INFO mapred.JobClient:   Job Counters
14/05/24 22:35:03 INFO mapred.JobClient:     Launched reduce tasks=1
14/05/24 22:35:03 INFO mapred.JobClient:     Launched map tasks=1
14/05/24 22:35:03 INFO mapred.JobClient:     Data-local map tasks=1
14/05/24 22:35:03 INFO mapred.JobClient:   FileSystemCounters
14/05/24 22:35:03 INFO mapred.JobClient:     FILE_BYTES_READ=24
14/05/24 22:35:03 INFO mapred.JobClient:     HDFS_BYTES_READ=12
14/05/24 22:35:03 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=80
14/05/24 22:35:03 INFO mapred.JobClient:     HDFS_BYTES_WRITTEN=14
14/05/24 22:35:03 INFO mapred.JobClient:   Map-Reduce Framework
14/05/24 22:35:03 INFO mapred.JobClient:     Reduce input groups=2
14/05/24 22:35:03 INFO mapred.JobClient:     Combine output records=0
14/05/24 22:35:03 INFO mapred.JobClient:     Map input records=2
14/05/24 22:35:03 INFO mapred.JobClient:     Reduce shuffle bytes=24
14/05/24 22:35:03 INFO mapred.JobClient:     Reduce output records=2
14/05/24 22:35:03 INFO mapred.JobClient:     Spilled Records=4
14/05/24 22:35:03 INFO mapred.JobClient:     Map output bytes=14
14/05/24 22:35:03 INFO mapred.JobClient:     Combine input records=0
14/05/24 22:35:03 INFO mapred.JobClient:     Map output records=2
14/05/24 22:35:03 INFO mapred.JobClient:     Reduce input records=2
14/05/24 22:35:03 INFO input.FileInputFormat: Total input paths to process : 1
14/05/24 22:35:03 INFO mapred.JobClient: Running job: job_201405242230_0002
```

图 6-1

图 6-1 是 Hadoop 任务开始后控制台的信息，可以看出截图的这部分是第一个 job 执行信息，后续的 jobs 也是类似的情况。随着任务执行接近尾声，可以从网页上获取已完成的任务信息和正在运行的任务信息，如图 6-2 和图 6-3 所示。从 name 栏中可以看到每个 job 的作用是什么，从 Map/Reduce complete 栏中可以看到各个 job 运行的完成程度。

Running Jobs

Jobid	Priority	User	Name	Map % Complete	Map Total	Maps Completed	Reduce % Complete	Reduce Total	Reduces Completed	Job Scheduling Information
job_201405242230_0011	NORMAL	chekaida	get result	100.00%	1	1	0.00%	1	0	NA

图 6-2

Completed Jobs

Jobid	Priority	User	Name	Map % Complete	Map Total	Maps Completed	Reduce % Complete	Reduce Total	Reduces Completed	Job Scheduling Information
job_201405242230_0001	NORMAL	chekaida	add comma	100.00%	1	1	100.00%	1	1	NA
job_201405242230_0002	NORMAL	chekaida	add comma	100.00%	1	1	100.00%	1	1	NA
job_201405242230_0003	NORMAL	chekaida	pattern match	100.00%	1	1	100.00%	1	1	NA
job_201405242230_0004	NORMAL	chekaida	column fixing	100.00%	1	1	100.00%	1	1	NA
job_201405242230_0005	NORMAL	chekaida	column fixing	100.00%	1	1	100.00%	1	1	NA
job_201405242230_0006	NORMAL	chekaida	table join	100.00%	3	3	100.00%	1	1	NA
job_201405242230_0007	NORMAL	chekaida	attribute computing	100.00%	1	1	100.00%	1	1	NA
job_201405242230_0008	NORMAL	chekaida	attribute computing	100.00%	1	1	100.00%	1	1	NA
job_201405242230_0009	NORMAL	chekaida	handle attribute	100.00%	1	1	100.00%	1	1	NA
job_201405242230_0010	NORMAL	chekaida	handle attribute	100.00%	1	1	100.00%	1	1	NA

图 6-3

经过一系列的操作，得到最终结果如图 6-4 所示。可以看出在 HDFS 中出现了一系列临时文件，这些文件是每次运算产生的，最后产生了符合配置文件制定操作的正确结果。

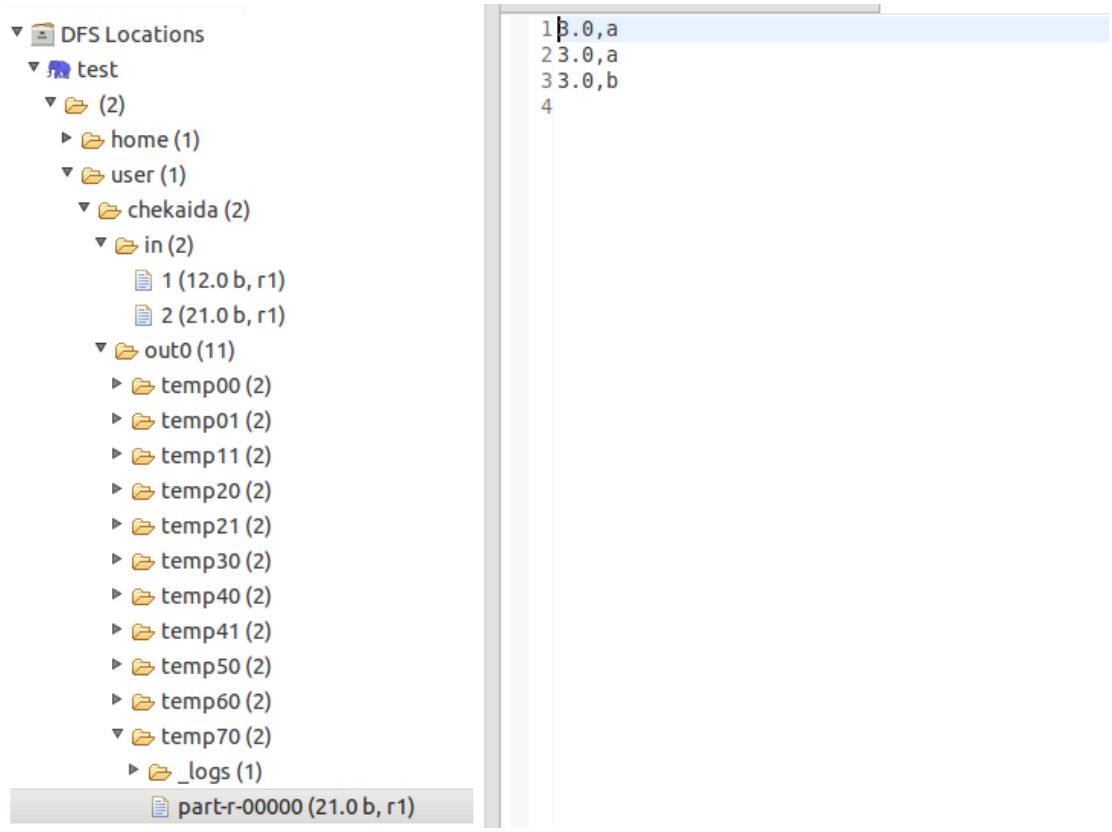

图 6-4

6.2 本章小结

本章通过对简单数据集的清洗工作的详细展示和阐述，较好的说明了如何使用本文提出的数据清洗工具以及配置文件，为大数据清洗任务提供了很好的参考。

7 总结

本章首先将对本文工作进行简要的总结，并横向比较了类似工具，体现出该工具的明显优点。我们也对现有的工作进行了反思和展望，针对不足之处提出进一步改进的思路，并给出下一步工作的计划。

7.1 课题总结

本文结合 Hadoop 平台的并行处理技术针对大数据清洗任务这一应用场景的 Hadoop 代码生成工具，另外本文还设计了简单有效的配置文件。该配置文件允许通过 MDE 的转换原语转换而来，大大提高了从模型到底层应用开发的自动化水平。该配置文件也允许用户手工配置，明显降低用户使用 Hadoop 平台处理数据清洗任务的困难。本文工作相对其他同类工作而言，具有适用性更加广泛，处理数据量大和自动化程度高的特点，并且对可以对更多的数据清洗的新任务进行快速扩展[15]，从而适用于更多的数据清洗任务。

7.2 不足和展望

本文提出的顺序式执行 Hadoop 过程较多，在数据量较少的情况下效率并不理想，这里还有很大的改进空间。这里我们可以对 Hadoop 顺序执行中的部分步骤进行简化和合并，使得整个清洗任务效率更高，同时我们还准备向 Hadoop 其它项目借鉴，从而改进本文中的实现，进一步提高整个工具的效率。

从实例研究中的结果列表可以看出整个过程会产生大量的中间信息，在数据量超过一定规模的情况下，这种中间信息的存在是致命的。在下一步工作中我们要想办法有效地改善这种情况。

另外，配置文件虽然比较简单易用，但格式不太明确，不利于用户使用，也不利于自动化生成。下一步我们准备设计 XML 文件来存储配置命令和参数，从而格式更加友好明确，有利于使用和分析。最后我们还需要在整个过程的自动化上花费功夫，让工具使用更加简洁方便。

参考文献

[1] Rahm E, Do H H. Data cleaning: Problems and current approaches[J]. IEEE Data Eng. Bull., 2000, 23(4): 3-13.

[2] Redman T C, Blanton A. Data quality for the information age[M]. Artech House, Inc., 1997.

[3] 王曰芬, 章成志, 张蓓蓓, 等. 数据清洗研究综述[J]. 现代图书情报技术, 2007, 12: 50-56.

[4] 郭志懋, 周傲英. 数据质量和数据清洗研究综述[J]. 软件学报, 2002, 13(11): 2076-2082.

[5] 孟小峰, 慈祥. 大数据管理: 概念, 技术与挑战[J]. 计算机研究与发展, 2013, 50(1): 146-169.

[6] Shvachko K, Kuang H, Radia S, et al. The hadoop distributed file system[C]//Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on. IEEE, 2010: 1-10.

[7] France R, Rumpe B. Model-driven development of complex software: A research roadmap[C]//2007 Future of Software Engineering. IEEE Computer Society, 2007: 37-54.

[8] Ghemawat S, Gobioff H, Leung S T. The Google file system[C]//ACM SIGOPS Operating Systems Review. ACM, 2003, 37(5): 29-43.

[9] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters[J]. Communications of the ACM, 2008, 51(1): 107-113.

[10] Borthakur D. HDFS architecture guide[J]. HADOOP APACHE PROJECT http://hadoop. apache. org/common/docs/current/hdfs design. pdf, 2008.

[11] Shafranovich Y. Common format and MIME type for Comma-Separated Values (CSV) files[J]. 2005.

[12] 张天, 李宣东. 基于 MDE 的异构模型转换: 从 MARTE 模型到 FIACRE 模型[J]. 软件学报, 2009, 20(2): 214.

[13] Anneke Kleppe, Jos Warmer, Wim Bast. 鲍志云译. 解析 MDA[M]. 北京:

人民邮电出版社，2004.

[14] 刘鹏，黄宜华，陈卫卫. 实战 Hadoop[J]. 2011.

[15] 郭志懋，俞荣华，田增平，等. 一个可扩展的数据清洗系统[J]. 计算机工程，2003，29(3)：95-96.

致谢

在 13 周的毕业设计过程中，我遇到了许多的问题和困难，但是在导师的精心指导下，在学长和同学的积极帮助下，我克服了一个又一个问题和困难，最终完成了这篇承载大学四年学习成果的毕业设计。

在此我首先要感谢我的毕业设计指导导师张天副教授，他从选题到毕业设计中的每个细节再到最后的毕业论文的撰写都以认真严谨的态度要求我指导我，这些无微不至的关怀和帮助让我顺利完成了这篇毕业论文。其次我要感谢徐朋飞学长和董乾豪学长。董乾豪学长在 Hadoop 应用方面给了我很大帮助，帮我解决了很多 Hadoop 实现中棘手的问题。徐朋飞学长在整个项目的各个方面提供了很多参考和帮助，让我对从顶层 MDE 到底层 Hadoop 平台的认识有极大地提高，在我撰写毕业论文的过程中，徐朋飞学长也给了我很多很有价值的毕业设计论文撰写建议。

最后我要感谢本篇论文参考资料的诸位专家和学者，没有他们的研究成果，本篇论文将难以完成。