****



**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 基于模型转换的Spark代码生成技术研究**

**年 级 2012 学 号 121220063**

**学生姓名 楼双胜**

**指导老师 张天 职 称 副教授**

**论文提交日期 2016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： **基于模型转换的Spark代码生成技术研究**

计算机科学与技术系 院系 计算机科学与技术 专业 2012 级本科生姓名： 楼双胜

指导教师（姓名、职称）： 张天副教授

摘要：

随着企业、组织间的紧密合作以及兼并，异构的数据信息资源广泛存在着，严重影响了部门间的信息交流。而模型转换的目标就是将异构的数据信息进行转换，并消除数据中存在的各种错误，优化数据模型，得到理想的数据模型。而在大数据的背景下，我们往往需要对海量的异构数据进行转换，因此提供一种简单而有效的基于模型转换的海量数据转换的工具显得尤为重要。

为了实现此想法，本文的工作设计开发了基于Spark平台的模型转换工具。本文的工作思路从MDE(Model Driven Engineering)的视角分析模型间的转换操作，通过翻译引擎，自动生成完成模型转换的Spark代码。这样用户只需要通过设计符合自己需求的配置文件，即可完成模型间的转换。本文首先结合实际应用场景分析了模型转换的基本操作，并在此基础上设计了符合基本需求的转换原语等配置文件。然后根据设计的转换原语等配置文件与Spark代码的映射关系实现了翻译引擎。最后使用该工具测试了一个与实际应用场景相关的简单实例，取得了理想的实验结果。

关键词： 模型转换；Spark平台；映射关系

**南京大学本科生毕业论文（设计）英文摘要**

THESIS: Spark code generation technology based on model transformation

DEPARTMENT: Computer Science and Technology

SPECIALIZATION: Computer Science and Technology

UNDERGRADUATE: Shuangsheng Lou

MENTOR: Associate Professor. Tian Zhang

ABSTRACT:

With the strong cooperation and merger between corporations and organizations, there are a lot of heterogeneous data, which heavily influence the information exchange between departments. And the goals of model transformation are to transform heterogeneous data to homogeneous data, eliminate a variety of errors within data, optimize the model structure, and obtain ideal data models. However, in the context of big data, we always need to transform vast amounts of heterogeneous data. So it is quite important to provide a simple and effective tool for the transformation of massive data based on model transformation.

In order to realize this idea, this project develops a Spark-based data transformation tool. The approach of this project is to analyze the transformation operations between models from the view of MDE (Model Driven Engineering), and then with translation engine automatically generate the Spark code for the model transformation. So users could complete the model transformation by just setting their required and specified configuration files. Combining with the practical applications, this paper at first analyzes the basic operations in model transformation, and then designs the basic transformation primitive. Then according to the mapping relationships between transformation primitive and Spark code, this paper gives the realization of translation engine. Finally, this paper uses this tool to test a simple example related to practical applications, which achieves desired processing results.

KEY WORDS: Model Transformation; Spark; Mapping Relationship

目录

[1 引言 1](#_Toc453019443)

[1.1 研究背景 1](#_Toc453019444)

[1.2 研究意义 1](#_Toc453019445)

[1.3 本文工作 2](#_Toc453019446)

[1.4 论文结构 2](#_Toc453019447)

[2 技术背景 4](#_Toc453019448)

[2.1 Spark平台 4](#_Toc453019449)

[2.1.1 Spark软件栈 4](#_Toc453019450)

[2.1.2 RDD API 5](#_Toc453019451)

[2.1.3 Spark集群架构 6](#_Toc453019452)

[2.2 Hadoop分布式文件系统（HDFS） 7](#_Toc453019453)

[2.3 模型转换思想 8](#_Toc453019454)

[2.4 相关的代码生成技术 9](#_Toc453019455)

[2.5 CSV文件格式 9](#_Toc453019456)

[3 整体分析和设计 11](#_Toc453019457)

[3.1 概述 11](#_Toc453019458)

[3.2 待处理数据集上的操作 11](#_Toc453019459)

[3.3 工具整体设计框架 13](#_Toc453019460)

[3.4 本章小结 14](#_Toc453019461)

[4 转换原语的设计 15](#_Toc453019462)

[4.1 转换原语介绍 15](#_Toc453019463)

[4.2 XML 15](#_Toc453019464)

[4.2 基本操作和设计框架 16](#_Toc453019465)

[4.2.1 基本操作 17](#_Toc453019466)

[4.2.2 设计框架 21](#_Toc453019467)

[4.3 转换原语命令详细介绍 22](#_Toc453019468)

[4.3.1 InputInformation与OutputInformation标签 22](#_Toc453019469)

[4.3.2 Source标签 23](#_Toc453019470)

[4.3.3 Target命令 23](#_Toc453019471)

[4.3.4 根标签 24](#_Toc453019472)

[4.4 本章小结 24](#_Toc453019473)

[5 翻译引擎的实现 25](#_Toc453019474)

[5.1 XML解析过程 25](#_Toc453019475)

[5.1.1 常用XML操作 25](#_Toc453019476)

[5.1.2 具体解析过程 26](#_Toc453019477)

[5.2 翻译引擎的实现 27](#_Toc453019478)

[5.2.1 Spark配置代码 27](#_Toc453019479)

[5.2.2 Spark文件读取代码 28](#_Toc453019480)

[5.2.3 Spark转换操作代码 28](#_Toc453019481)

[5.2.4 Spark输出模型代码 31](#_Toc453019482)

[5.3 本章小结 31](#_Toc453019483)

[6 实例研究 33](#_Toc453019484)

[6.1 实例研究 33](#_Toc453019485)

[6.2 本章小结 38](#_Toc453019486)

[7 总结 39](#_Toc453019487)

[7.1 课题总结 39](#_Toc453019488)

[7.2 不足和展望 39](#_Toc453019489)

[参考文献 40](#_Toc453019490)

[致谢 41](#_Toc453019491)

# 1 引言

## 研究背景

随着不同数据库技术的发展、企业、组织间的合作与兼并的加剧及多样化新技术的采用，使得异构性的信息资源广泛存在于企业的管理信息系统中，企业的各个部门形成了大量的信息孤岛，严重阻碍了各部门间的信息交流，不利于信息资源的共享和处理[1]。在如今大数据的背景下，企业数据信息呈指数型增长，异构的数据信息可以抽象为不同的数据模型，为了减少企业的投资以及保护企业现有资源信息，一种简单有效的模型转换方法显得尤为重要。

目前主流的大数据支撑平台有Spark、Hadoop，它们均能充分利用集群的模式进行高速运算和存储，在大批量数据处理任务中表现出一定的高效性。然而尽管在一定程度上它们向用户遮蔽了分布式底层的细节，这些平台的使用仍具有其一定的专业性和复杂性，比如相关的信息配置以及编码能力。因此我们希望提供一种屏蔽代码细节、更加抽象的数据处理方案——基于模型转换的Spark代码生成技术。

## 研究意义

在数据挖掘的领域中，数据的预处理以及数据模型的合理转换是有效数据挖掘的前提工作，是一个不可或缺的步骤。在这个步骤中，数据的处理包括：统一多数据源的数据模型；消除数据中存在的错误信息以及不一致的信息；转换数据的模型来适应挖掘算法的输入要求[2]。而大部分的研究工作主要集中在挖掘算法的设计与优化上，模型转换的工作并未得到较多的关注与投入。同时企业的合作与兼并也带来数据信息模型不一致的问题，这些迫切的现实问题都可以通过模型间的转换达到预想的效果。

在另一方面，大数据时代已经将数据的规模提升到PB、EB级别，数据的每一次遍历会耗费较长的时间[3]。考虑到人力成本、时间成本以及资金成本，我们需要一种有效的方法来实现模型间的转换。本文设计实现了一种基于Spark平台的模型转换工具。通过涵盖模型间转换的所有基本操作，使得用户可以利用一个简单的配置文件就自动生成可以在Spark平台执行的Spark代码，高效快速地实现模型间的转换，完成用户的最终需求。

## 本文工作

本文从实际场景出发，抽象出模型转换的基本操作，设计开发了一个基于模型转换的Spark代码生成技术的工具。本文首先介绍了关于该工作的一些技术背景，包括分布式Spark集群平台、Hadoop分布式文件系统、模型转换思想、相关的代码生成技术以及csv数据文件格式等。接着本文对该工具的整体设计进行了简要分析，结合实际场景涉及的模型转换操作，给出了该工具整体的工作流程以及模型转换流程。接着本文从MDE的视角来介绍转换原语的设计，通过抽象出模型转换的基本操作，由相对应的Spark代码框架模板设计了相对应的转换原语规则，实现模型到底层的层次功能划分和自动化实现。然后本文详细介绍了翻译引擎的实现，先从解析过程入手，介绍了翻译引擎的工作流程。接着针对包括每个转换操作原语的转换原语与Spark代码框架模板的映射关系实现了翻译引擎工具。另外，本文通过一个实际应用场景——会员用户信息与订单记录信息展示了此设计的合理性和可实施性。最后本文对整个工作进行了总结，指出工作中的不足之处并提出改进的思路。

## 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、研究内容以及论文的整体结构。

第二部分介绍了与Spark、代码生成技术等相关的技术背景。

第三部分首先对待处理数据所需的转换操作进行了详细分析，抽象出其基本转换操作。结合这些基本转换操作以及Spark平台的特性，给出了该工具整体的设计以及模型转换流程。

第四部分从MDE的视角介绍了转换原语的设计过程。先对本工具采用的XML进行相关的必要介绍，接着详细介绍了每条转换原语的设计理念，并与模型转换操作相映射，最后设计出较完整的转换原语等配置文件。

第五部分介绍了翻译引擎的实现过程。从解析过程入手，通过Spark代码模板与转换原语间的映射关系，并结合具体的参数信息，完成翻译引擎的实现。

第六部分通过一个具体的现实实例对本文设计的工具的工作流程进行了详细的展示。

第七部分总结了本文的工作，并指出当前工作中存在的不足和对未来工作的展

# 2 技术背景

## 2.1 Spark平台

Spark，一个用于集群计算的开源平台，目前在处理大规模数据方面得到许多企业的应用，比如雅虎，百度和腾讯。

在速度方面，Spark扩展了广泛应用的MapReduce计算模型，例如存在DAG执行引擎机制，高效地支持更多计算模式，包含交互式查询，内存计算以及流处理。在内存计算方面，Spark速度相比Hadoop可以达100倍快；在磁盘计算方面，也可以达到10倍之快[4]。

在使用方面，Spark具有较强的简易性。Spark提供了超过80个用于并行操作的操作原语，用于用户在独立应用或交互式操作中的使用。

在集群管理器方面，Spark支持在各种集群管理器上运行，包括Hadoop YARN、Apache Mesos，以及Spark自带的独立调度器。它支持访问各种不同的数据源，包括HDFS、Cassandra、HBase、Hive、Tachyon以及S3。

### 2.1.1 Spark软件栈

Spark项目由多个紧密集成的组件组成，如图2-1所示[5]。



图2-1 Spark软件栈

Spark SQL: Spark用来操作结构化数据的程序包。采用Spark SQL,我们能够利用SQL或者Apache Hive版本的SQL方言（HQL）来查询语言。

Spark Streaming: Spark提供的对实时数据进行流式计算的组件，例如网络服务中用户提交的状态更新组成的消息队列，或者是网页服务器日志。

MLib: MLib提供了很多中机器学习算法，包括分类、回归、聚类、协同过滤等，也提供了模型评估、数据导入等额外的支持功能。

GraphX: GraphX是用来操作图，进行并行计算的程序库。它支持针对图的各种操作，以及一些常用的图算法，比如PageRank和三角计算。

Spark Core: 它完成了Spark的基本功能，包括任务调度、内存管理、错误回复、与存储系统交互等模块，而且包含了对弹性分布式数据集（resilient distributed dataset，简称RDD）的API定义。

独立调度器、YARN、Mesos: Spark支持在这些集群管理器上运行。

### 2.1.2 RDD API

RDD，为分布在多个计算节点上可以并行操作的元素集合，是Spark主要的编程抽象。它可以包含Python、Java、Scala中任意类型的对象，或者可以为用户自定义的对象，本文中涉及的操作均为针对Scala对象的操作。在Spark中，对数据的操作主要分为三种，创建RDD，转化已有的RDD以及调用RDD操作进行求值。

1. 创建RDD

用户可以通过两种方法创建RDD: 读取一个外部数据集，或在驱动器程序里对一个集合进行并行化。

读取外部数据集：val lines=sc.textFile(“Readme.md”)

并行化集合：val lines=sc.parallelize(“List(“apple”, ”orange”, “banana”)”)

1. 转换操作

转换操作会由一个RDD生成一个新的RDD。一些常用的转换操作的API如下所示。

filter(): 返回一个由通过传给filter()的函数的元素组成的RDD

map(): 将函数应用于RDD中的每个元素，将返回值构成新的RDD

distinct(): 返回一个去重后组成的RDD

reduceByKey(func): 合并具有相同键的值

join(other): 对两个RDD进行内连接

rightOuterJoin(other): 右外连接，确保第一个RDD键必须存在

leftOuterJoin(other): 左外连接，确保第一个RDD键必须存在

1. 行动操作

行动操作对数据集进行实际的计算，向驱动器程序返回结果或是把结果写入外部系统。一些常用的行动操作API如下所示。

collect(): 返回RDD中的所有元素

count(): RDD中的元素个数

countByValue(): 各元素在RDD中出现的次数

aggregate(zeroValue)(seqOp,combOp): 并行整合RDD中所有数据，例如sum

### 2.1.3 Spark集群架构

在分布式环境下，Spark集群采用的是主/从构造。在一个Spark集群中，有一个负责中央调节的驱动器节点，与之对应的工作节点称为执行器节点。驱动器节点可以与大量的执行器节点进行通信，共同被称为一个Spark应用。而一个Spark应用需要一个集群管理器来管理集群中的所有节点，该集群管理器可以是Hadoop YARN、Apache Mesos，以及Spark自带的独立调度器。Spark驱动器节点会执行一个驱动器程序，该程序创建一个SparkContext对象对计算集群进行连接。Spark集群架构图如图2-2所示。

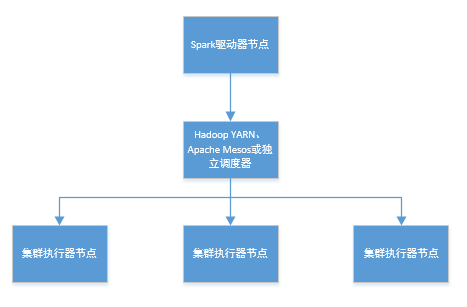


图2-2 分布式Spark架构

其中Spark应用中驱动器节点主要有两个职责[5]：

1. 将用户程序转为任务

所有的Spark程序遵循同样的结构：先创建RDD，转换RDD并生产新的RDD，使用行动操作处理RDD。而这样的结构使得Spark程序创建了一个由操作组成的逻辑上的有向无环图。而当驱动器节点运行时，它会把用户程序转换为这样一个物理执行顺序，并将其分为一系列任务，这些任务会被打包并送到集群中。

1. 为执行器节点调度任务

Spark驱动器节点会执行一个驱动器程序，它会根据当前的执行器节点的集合以及数据所在的位置，尝试将任务分配给合适的执行器节点。

## 2.2 Hadoop分布式文件系统（HDFS）

HDFS（Hadoop Distributed File System，Hadoop分布式文件系统）是分布式计算中数据存储管理的基础。由于其高容错性、高可靠性、高可扩展性、高获得性以及高吞吐性等特性，HDFS为海量数据提供了便利的存储[6]。

HDFS的体系结构为主/从（Master/Slave）结构，一个HDFS集群由一个NameNode（Master）和一定数量的DataNode（Slave）组成，如图2-3所示。其中NameNode管理文件系统的元数据，而DataNode则存储了实际的数据。当用户需要了解文件的信息时，通过与NameNode交互来得到文件信息，而需要对文件进行操作时，则是用户直接与DataNode进行交互[7]。

NameNode（Master），主要负责管理、记录有关文件系统的任务，包括管理文件系统的命名空间，协调客户端对文件的访问，记录文件数据块在每个DataNode上的位置和副本信息，记录命名空间内的改动或命名空间本身属性的改变。

DataNode（Slave），主要负责文件数据块在实际物理节点上的存储管理以及用户的读写请求。HDFS的数据为“一次写入、多次读取”，典型的块大小是64MB。HDFS文件通常按64MB切分成多个数据块，每个数据块的存储位置尽可能分散于不同的DataNode。

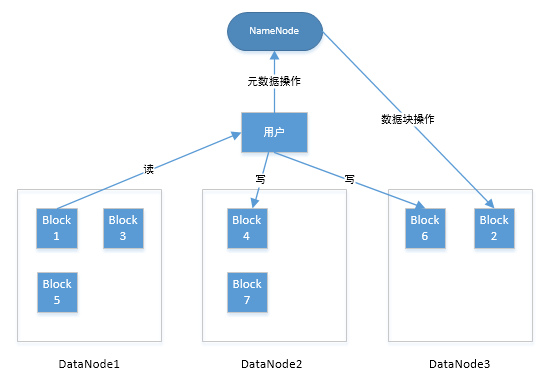


图2-3 HDFS的结构示意图

本工作的HDFS部署架构是在一个专门的机器上运行NameNode，集群中的其他三个机器上各自运行一个DataNode，一共包含三个DataNode。当用户需要访问一个文件时，首先用户从NameNode中获得组成该文件的数据块位置列表，即获知了数据块存储在哪些DataNode上。接着用户直接从DataNode上读取文件数据，此时NameNode不参与文件的传输。

## 2.3 模型转换思想

模型驱动工程（Model-Driven Engineering，简称MDE）[8]，是一种以建模和模型转换为主要途径的软件开发方法。MDE技术解决了软件开发的效率低、可移植性差等问题，缓解了平台的复杂性，更有效地表达了域概念。例如，因为通用符号较难解释应用领域概念和设计意图，DSMLs并不是采用通用符号来匹配，而是通过元模型精确匹配域的语法和语义进行定制[9]。其中模型转换是开发基于MDE的关键技术，用来解决模型到模型以及模型到代码间的映射问题，并将模型映射为不同技术平台上的实现，如图2-4所示[10]。故本文工作通过模型转换方法的设计理念，将异构数据的转换抽象为模型间的转换，根据模型转换方法设计相应的转换规则，从而实现与平台相关的模型实例的转换。

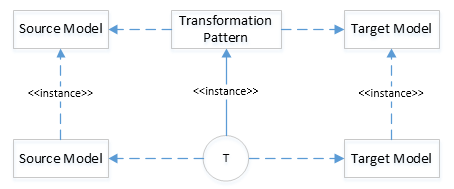


图2-4 模型转换方法

## 2.4 相关的代码生成技术

代码生成技术，是一种生成代码的程序，主要通过读取工程中的模型或是文本内容，遵循提前设计好的规则和模式，生成规范的计算机能理解的代码[11]。

目前，在自动代码生成技术的实现方面，主要存在以下几个方案。

1. 基于UML模型的代码生成方案

该方案从UML模型入手，直接生成系统的各个层次，其在代码生成过程中保留了原来模型中的层次关系。开发者可以通过多范例的方法为其具体的实现选择合适的解决方法[12]。

1. 基于XML的Web应用程序的代码生成方案

该方案使用元模型，通过分析万维网与Web页面环境语法间的协同效应，提供一种基于三层元模型结构的代码生成系统模板[13]。

1. 基于元数据驱动的代码生成方案

该方案采用NET框架，在多层的企业资源管理系统中实现存储过程和数据库触发器的自动生成[14]。

1. 基于设计模式的代码生成方案

该方案主要通过实现一个分布式体系结构，由非定制的组件完成的工具，来自动实现描述抽象层的设计模式[15]。

1. UML状态机和合作图的模型检测和代码生成方案

该方案通过有限自动机模式的匹配，实现一种小的快速的代码生成器[16]。

在本工作中，我们选择了基于模型转换的代码生成方案，主要考虑到我们选择的应用场景相对简单，仅仅涉及到一些数据模式的转换。同时此方案相对简单灵活，在技术上容易实现和使用。

## 2.5 CSV文件格式

CSV（Comma-Separated Values，逗号分割值），其文件以纯文本形式存储表格数据。考虑到目前CSV文件在大数据存储中应用广泛，且格式简单易于处理，故本文工作采用CSV文本作为输入数据。其具体格式存在以下特征：

1. 由记录组成，每行一条记录；
2. 每条记录被逗号分隔成字段；
3. 逗号前后的空格会被忽略；
4. 每条记录都有同样的字段序列。

# 3 整体分析和设计

本章将对模型间的转换操作进行介绍，着重介绍待处理的数据集所需要的操作，为后面转换原语的设计以及翻译引擎的实现提供参考，并根据整个工具的应用场景提供整体的设计框架和待处理的数据集的转换操作流程图。

## 3.1 概述

该工具通过用户的输入，由翻译引擎自动生成可以在集群上运行的Spark代码，从而实现模型转换。该工具操作简单，屏蔽了代码方面的所有细节，减轻了用户使用Spark平台的困难性，极大提高了数据处理的效率。同时该工具涵盖了大数据集上大部分的数据操作，可以较好得满足用户的需求。

该工具基本Spark-1.6.1版本进行开发，采用Scala-2.10语言，通过转换原语以及用户设置的参数，自动生成可执行的Scala代码。该Scala可直接打包在Spark平台运行，利用Spark提供的独立集群管理器并行化待处理数据集上的操作，实现数据模型的转换。

## 3.2 待处理数据集上的操作

待处理数据集以csv格式存储在Spark平台上。通过研究用户信息与购买记录信息之间的实际应用场景，分析待处理数据集上可能需要的一些数据操作。其中用户信息表中包括用户的id，用户的姓名，用户的出生年月，用户的地址，用户的消费等级以及用户的收入，具体数据格式如图3-1所示。而购买记录信息表中包括订单的id，订单的时间，购买此订单用户的id，订单的额度以及订单的状态，具体数据格式如图3-2所示。下面根据此应用场景对待处理大数据集上的操作进行分析。



图3-1 用户表信息

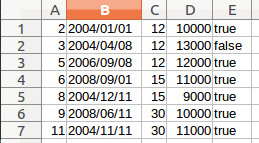


图3-2 购买记录表信息

1. 重复信息的去重

数据集中可能存在重复的信息，根据需求可以指定去重的主键，例如用户信息表中存在重复信息时，我们可以指定去重的主键为用户姓名或者用户姓名加上用户地址，图3-1中第2行与第6行出现用户姓名加上用户地址重复的情况；

1. 信息的过滤

数据集中部分属性值不满足需求而需要舍弃整行记录，来保证数据的有效性。例如当我们只需要北京区域的成年用户的数据信息时，我们需要选择用户地址在北京以及出生年月小于1998的用户，图3-1中第1行、第3行、第5行、第7行以及第9行的用户信息中用户地址或出生年月不符合需求需要舍弃；

1. 属性值的填充

数据集中可能存在某些字段记录信息缺省的情况，需要对其进行填充。例如用户信息表中某些记录缺省用户收入时，我们可以采用所有用户收入的平均值，最小值或最大值等进行填充，如图3-1中第5行和第8行的用户收入存在缺省的情况；

1. 属性值的修改

根据模型转换的需求，需要对某些属性值进行一定的修改。例如每过一年我们需要递增用户的年龄，增加用户的等级等来保持数据的正确性。

1. 多张表的连接

数据集中可能存在多张表的情况，可能需要根据需求对不同表进行连接，得到同一事务不同表中对应信息的综合，便于处理与分析。例如我们需要连接用户表和购买记录表来得到用户信息与购买记录的对应关系，即连接图3-1与3-2的表格信息。通过指定用户信息表中连接的主键为用户id和购买记录表中连接的主键为购买此订单的用户id，来得到用户信息与购买记录的对应关系；

1. 属性信息的计算与统计

有时我们需要对数据集上的信息进行计算，并做一定的统计，从而对数据属性有一个宏观的认识，比如总数、平均值、最大值、最小值等。例如当我们需要了解用户数目或者用户收入幅度时，我们可以统计用户表中记录的数目，以及通过计算用户收入中最大值与最小值的差（图3-1的第6个字段属性）来得到用户收入的幅度。

以上是数据集上可能需要的一些操作的分析。此外，需要注意的是，数据的一些操作存在一定的顺序性，例如重复信息的去重应该置于属性信息的计算与统计之前，防止出现处理不得到的情况。

## 3.3 工具整体设计框架

通过以上的分析和整理，本文提出了该工具的整体设计框架以及待处理的大数据集的处理流程图。该工具由用户输入的待处理的数据集、配置文件、配置文件翻译引擎以及Spark平台组成。本工具的整体设计框架如图3-7所示。

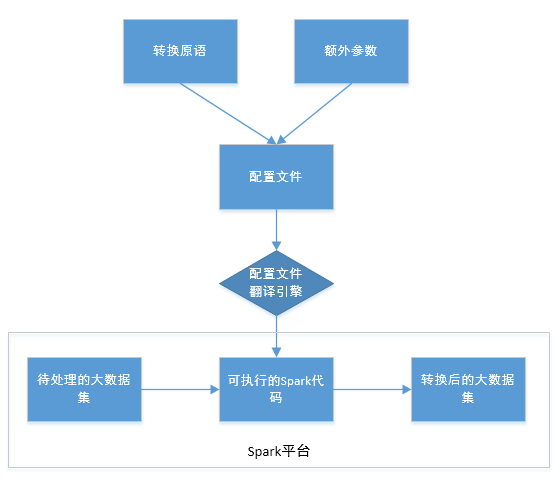


图3-7 整体设计框架

从图3-7工具的整体设计框架可以看出，由转换原语和额外参数组成的配置文件通过配置文件翻译引擎可以产生可执行的Spark代码。在设计转换原语文件时，我们需要结合考虑待处理的大数据集上所有可能的操作以及Spark代码的特性。图3-8显示了待处理的大数据集在Spark平台上的处理流程图。



图3-8 数据处理流程图

从图3-8 数据处理流程图我们可以看出，数据集在转换数据模式的过程中，需要通过属性模式匹配过程来得到过滤后的数据集。而这里的属性模式匹配包含多种情况，包括信息的去重、剔除不符合目标的数据记录等等。

## 3.4 本章小结

本章主要介绍了本文设计的工具的运行环境和基本要求，并对模型转换中可能存在的操作进行了仔细的分析，提供了所需的基本操作。并给出了该工具的整体设计框架，生动直观的展现了工具的整个运行流程，并进一步给出了待处理数据在Spark平台上的数据流程图，深入了解源模型逐步进行转换成为目标模型的过程。

# 4 转换原语的设计

本章从MDE视角出发，结合3.2节提出的待处理大数据集上的操作，抽象出其基本操作，包括filter、join等，并设计相对应的转换原语，并通过分析大数据集上的操作顺序，进一步设计规定转换原语的框架，从而设计出有效的配置文件，为配置文件翻译引擎提供强有力的支撑。

## 4.1 转换原语介绍

转换原语通过抽象模型转换的基本操作，规定了源模型与目标模型，并给出了相对应的转换操作。为了使得转换原语具有较强的可扩展性，并且满足简单易学的用户要求，本文工具采用可扩展标记语言（Extensible Markup Language，简称XML）编写转换原语，使得这样生成的转换原语文件具有一定的层次性，易于理解和编写，且无二义性。

## 4.2 XML

从本质来讲，XML是一种元语言。通过XML，我们可以设计描述数据的方法，包括存储、传送以及程序如何处理等。它采用标记的文本方式记录数据信息，具有很强的可读性[17]，其语法规则如下：

1. XML元素必须有关闭标签。即<p></p>标签成对出现。
2. XML标签对大小写敏感。即标签<Message>与标签<message>属于不同的标签。
3. XML元素必须正确嵌套。即满足<p><b></b></p>如此的顺序。
4. XML文档必须有根元素。即所有标签最外层一对<root></root>标签代表根元素标签。
5. XML的属性值需要加引号。即采用<user name=”david”></user>形式。
6. XML特殊字符采用实体引用的书写格式，如表4-1所示。

表4-1 XML特殊字符表

|  |  |  |
| --- | --- | --- |
| 特殊字符 | 实体引用 | 含义 |
| < | &lt; | 小于 |
| > | &gt; | 大于 |
| & | &amp; | 和号 |
| ‘ | &apos; | 单引号 |
| “ | &quot; | 引号 |

而整个XML文档形成了一种树结构。这棵树从“根部”开始，扩展到其所有的枝叶。图4-1是一段较简单XML文档。



图4-1 XML简易文档

图4-1中，第一行是XML声明。它定义了XML的版本（1.0）和所使用的编码（UTF-8）。下一行描述了文档的根元素<bookstore>。它的子元素为<book>，而元素<book>则拥有四个子元素，<title>、<author>、<year>、<price>。其中所有的元素标签均成对出现。对于此XML文档，其对应的树结构如图4-2所示。

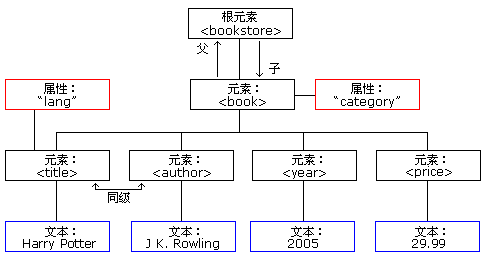


图4-2 XML树结构

## 4.2 基本操作和设计框架

通过图3-8的数据处理流程图，我们可以清楚得了解到待处理数据集在Spark平台的可能需要的一系列操作，并结合Spark平台的特性，设计出相对应的转换原语，规定其转换原语规则。另外根据Scala程序打包运行的特性，在转换原语中增加必要的参数信息，使其具有较完整的转换信息，更好地通过配置文件翻译引擎产生可执行的Spark代码。

### 4.2.1 基本操作

结合3.2待处理数据集上的操作，抽象出转换原语中需要的基本操作。

1. 重复信息的去重

待处理数据集上可能存在若干重复的数据信息，在进行其他操作前我们首先需要将其进行去重，保留不重复的数据集合，同时不同的数据集去重的主键数目可能不同，需要的参数信息为去重信息的主键集合。

操作定义：

<Redundancy>XXX,XXX,…</Redundancy> （XXX为属性名称）

例如：

<Redundancy>cus\_name,address</Redundancy>

此时对应的Spark代码模板如下：

val m2=m1.map(divide\_operation)

val m3=m2.reduceByKey((x,y)=>x)

val m4=m3.map(map\_operation)

因为在进行去重操作时，我们需要将数据映射为（key,value）的形式，故一开始需要通过divide\_operation的数据映射函数将数据映射为（key,value）形式，而最后的map\_operation函数则是将映射后的数据恢复到原来的数据模式，来保证后续操作的正确性。

1. 不符合需求的信息的过滤

当我们需要剔除一部分我们不需要的数据信息时，我们需要过滤掉这部分信息，需要的参数信息包括需要比较的属性名称、比较条件以及比较的属性值。

操作定义：

<Filter name=”XXX” type=”YYY”>ZZZ</Filter> （XXX为比较的属性名称，YYY为比较条件，ZZZ为比较的属性值）

其中type分为&gt;(大于)、&lt;(小于)、==(等于)、contains(包含)。

例如：

<Filter name="address" type="==">beijing</Filter>

<Filter name="birth\_year" type="&lt;">1998</Filter>

<Filter name="order\_time" type="contains">2004</Filter>

此时对应的Spark代码模板如下：

val m2=m1.filter(operation)

其中operation由Filter标签的具体参数决定，为实现数据信息的挑选的函数。

1. 属性值的统计与填充

数据集中很有可能存在某些数据信息缺失的情况，此时我们需要填充入一个合适的数据信息来保证我们后面操作的正确性；以及我们有时需要统计属性值的一些特性，比如平均值，振幅等等来了解属性的特征。我们将这两项操作抽象为一条转换原语，需要的参数信息包括统计的类型，填充的类型以及统计的属性名称。

操作定义：

<Statistic target=”XXX” type=”YYY”>ZZZ</Statistic> （XXX为统计的类型，YYY为填充的类型，ZZZ为统计的属性名称）

其中target分为average（统计其平均值）、max（统计其最大值）、min（统计其最小值）；type分为autofill（自动填充）、none（不填充）。

例如：

<Statistic target="average" type="autofill">salary</Statistic>

其对应的Spark代码模板由两部分组成，分别由target和type的属性值决定。

当target值为average时，其对应的一部分Spark代码模板为：

val sum=m1.aggregate((0.0,0))((acc,value)=>{

if(value.\_x=="N/A")

(acc.\_1,acc.\_2+1)

else

(acc.\_1+value.\_x.toDouble,acc.\_2+1)

},(acc1, acc2)=>(acc1.\_1+acc2.\_1,acc1.\_2+acc2.\_2))

val avg=sum.\_1/sum.\_2.toDouble

其中value.\_x表示需要统计字段在数据中位置，由Statistic标签内容来获知；

当target值为max时，其对应的一部分Spark代码模板为：

val temp=m1.map(values=>values.\_x)

val avg=temp.max()

当target值为min时，其对应的一部分Spark代码模板为：

val temp=m1.map(values=>values.\_x)

val avg=temp.min()

当type值为none时，表示为属性统计操作，此时不需要额外的Spark代码；当type值为autofill时，表示为属性修改填充操作，其对应的一部分Spark代码模板为：

val m2=m1.map(x=>

if(x.\_x=="N/A")

{val y=avg

(new order) }

else

{val y=x.\_x

(new order) }

)

其中x.\_x表示字段位置，new order为替换上新的属性值的各字段顺序。

1. 多张表的连接

当我们需要同一事务在不同表中信息的综合时，我们需要连接不同的表。连接操作时，需要的参数信息包括所连接两张表的id，两张表连接时各自的连接主键，连接的类型。因为通过连接操作，产生一张新的表，我们需要指定新表的id，以便后面操作的使用。

操作定义：

<JoinKey first=”X” first\_ele=”Y” second=”M” second\_ele=”N” type=”Q”>Z</JoinKey> （X为第一张表的id，Y为第一张表的连接主键属性名称，M为第二张表的id，N为第二张表的连接主键属性名称，Q为连接的类型，Z为产生的新表的id）

其中type分为Join（内连接）、RightOuterJoin（右外连接）、leftOuterJoin（左外连接）

例如：

<JoinKey first="1" first\_ele="cus\_id" second="2" second\_ele="cus\_id" type="Join">3</JoinKey>

此时对应的Spark代码模板如下：

val temp1=m1.map(divide\_operation)

val temp2=n1.map(divide\_operation2)

val temp3=temp1.join(temp2)

val p1=temp3.map(map\_operation)

其中divide\_operation、divide\_operation2正如去重操作中所说的那样，需要将数据映射为（key,value）形式。而map\_operation函数负责将数据从（key,value）形式恢复到原来的数据表现形式。

1. 属性值的转换与修改

根据模型转换的需求，需要对某些属性值进行一定的转换与修改，比如出生年月转换为年龄、所有年龄值的递增更新等。需要的参数信息包括需要修改的属性名称、修改的类型以及修改的幅度。

操作定义：

<ModifyValue name=”XXX” type=”Y” range=”ZZZ”>

</ModifyValue> （XXX为修改的属性名称，Y为修改的类型，ZZZ为修改的幅度）

其中type分为+（增加）、-（减少）、\*（倍数增长）、/（倍数减少）；range分为average（平均值）、max（最大值）、min（最小值）、1,2,3…（纯阿拉伯数字）。

例如：

<ModifyValue name=”age” type=”+” range=”1”> </ModifyValue>

此时对应的Spark代码模板如下：

当range属性值为average、max、min时，此时的Spark代码框架与生成Statistic标签时一部分的Spark代码相同；

当range属性值为阿拉伯数字时，即对应变量avg的值即为阿拉伯数字，其对应的Spark框架为：

val m2=m1.map(x=>x.\_x+number)

其中x.\_x表示字段位置，number为阿拉伯数字。

### 4.2.2 设计框架

结合4.2.1中的基本操作原语以及Spark平台打包程序的特性，保证XML解析的无二义性，我们对XML的整体框架结构做了一定的规定，如图4-3所示，其中basic operation指4.2.1中除了连接操作的所有基本操作。



图4-3 XML框架结构

需要注意的是：

1. 每张数据表分配特定的id

对于读入的数据产生的表，在源模型描述中分配其id号，即InputPath标签的id属性；而对于连接操作产生的新表，在连接操作中分配其id号，即JoinKey的内容值。

1. 对于basic operation，需要在上层标签中指定其处理的表格id

上层标签定义 ： <MatchType id=”XXX”></MatchType> （XXX为处理表格的id号），该标签的内容中可以存在一个或多个对该表基本操作原语，如图4-4所示。



图4-4 MatchType架构

## 4.3 转换原语命令详细介绍

配置文件中一个主要部分就是转换原语。结合图4-1XML框架结构，我们为源模型和目标模型的定义设计了6种标签，分别为InputInformation、OutputInformation、InputPath、InputFormat、InputType、OutputPath。针对转换操作的动作，我们设计了Source、Target、MatchType、JoinKey、Redundancy、Filter、Statistic、ModifyValue等8种标签。结合Spark平台上Scala打包运行的特性，我们设计了特定的根标签，并设计了IOInformation、ExecuteTransformation、ExecuteRule标签使得XML结构更具有层次性，易于理解。

### 4.3.1 InputInformation与OutputInformation标签

针对源模型和目标模型的定义，我们设计了InputInformation和OutputInformation标签。在InputInformation标签下，定义了源模型的信息，包括源模型数据的路径信息、源模型各字段的属性名称、源模型各字段的类型。这三种源模型的定义信息分别对应的XML标签是InputPath、InputFormat以及InputType；而在OutputInformation标签下，定义了目标模型的输出路径信息，其对应的XML标签是OutputPath。下面对这些标签进行详细的介绍。

<InputInformation files="X"></InputInformation> ： X代表源模型的个数

<OutputInformation files="X"></OutputInformation> ： X代表输出模型的个数

<InputPath id="X" name="XXX">YYY</InputPath> ：X代表分配给此模型的id，XXX为模型名称，YYY为模型的路径信息

<InputFormat id="X">YYY,...</InputFormat> ： X代表模型的id，YYY,…为模型各字段的属性名称

<InputType id="X">YYY,...</InputType> ：X代表模型的id，YYY,…为模型各字段的类型

<OutputPath id="X">YYY</OutputPath> ： X代表分配给输出模型的id，YYY为输出模型的路径信息

### 4.3.2 Source标签

除了模型的输出转换操作外，其余转换操作动作均在Source标签下定义执行。其下层标签为MatchType标签和JoinKey标签，其中MatchType标签下定义了大部分的转换操作原语，包括Redundancy、Filter、Statistic、ModifyValue四种转换操作原语。

<Source tablecount="X"> </Source> ： X代表需要转换操作的模型的数目

<MatchType id="X"></MatchType> ： X代表需要进行转换操作的模型的id号

对于Redundancy、Filter、Statistic、ModifyValue以及JoinKey这五种转换原语操作标签，已经在4.2.1中进行了详细分析。

### 4.3.3 Target命令

Target标签下定义了模型的输出形式。通过在下层设计MatchType标签，来将不同模型的输出划分出来。而模型输出的直接操作原语是CreateElement标签。

<Target tablecount="X"></Target> ： X代表输出模型的数目

<CreateElement order="X">YYY</CreateElement> ： X代表该属性字段在输出模型中位置，YYY为选择的属性名称。

### 4.3.4 根标签

为了通过翻译引擎生成完整的可执行的Spark代码，我们需要提供打包运行代码的类的名称，通过Package\_name标签，每次获取根标签的名称来得到类名,即标签<Package\_name></package>。例如根标签为<Transformation> </Transformation>时表明生成的Spark代码以Transformation为类名。

## 4.4 本章小结

本章首先对转换原语进行了简单的介绍。接着针对模型之间的转换操作，提出了基本的转换原语操作和规则。最后结合基本操作原语、Spark平台打包程序的特性以及XML无二义性，详细介绍了XML整体框架设计，并为每条设计做出相关的解释，从而让用户对XML编写的转换原语有一个清晰的认识，从而为翻译引擎提供必要的准确、高效的输入信息。

# 5 翻译引擎的实现

由配置文件生成Spark代码是本文工作的一个实现重点。而配置文件一个主要的组成部分就是用可扩展标记语言（简称XML）编写的转换原语。首先我们对XML的解析方法进行详细的介绍，接着对转换语言中的各条指令进行分析，通过转换原语与Spark代码模板的映射关系，结合用户提供的参数得到具体的可以在Spark平台执行的Spark代码。

## 5.1 XML解析过程

现在解析XML方法很多，主流的主要包括DOM、SAX、JDOM、DOM4J，而本工作采用DOM4J解析XML。

DOM4J（Document Object Model for Java）是一个Java的XML API，是JDOM的升级品，用来读写XML，其将整个文档作为一个对象进行处理。下面对DOM4J解析过程进行具体分析。

### 5.1.1 常用XML操作

1. 读取XML文件，获得Document对象

SAXReader reader=new SAXReader();

Document document=reader.read(new File(“inputa.xml”));

1. 获取文档的根节点

Element rootnode=document.getRootElement();

1. 获取某节点的单个子节点

Element node=rootnode.element(“member”);

其中member为子节点名称

1. 获取某节点的文本内容

String text=node.getText();//获取node节点的文本内容

String text=rootnode.elementText(“name”);//获取rootnode节点下name子节点的文本内容

1. 遍历某节点下的所有子节点

Iterator<Element> it = node.elementIterator();

while(it.hasNext()){

Element e=it.next();

//element operations

}

1. 获取某节点的指定属性

Attribute attribute=node.attribute(“color”);

其中node代表Element对象的节点，获的属性名为color的属性。

1. 获取属性的文字内容

String text=attribute.getText();

String text=attribute.getVaule();

1. 遍历某节点的所有属性

for(Iterator it=node.attributeIterator();it.hasNext();){

Attribute attribute = (Attribute) it.next();

String text=attribute.getText();

//other attribute operations

}

其中node为Element对象。

### 5.1.2 具体解析过程

针对图4-3 XML框架结构，其具体解析过程如图5-1所示。

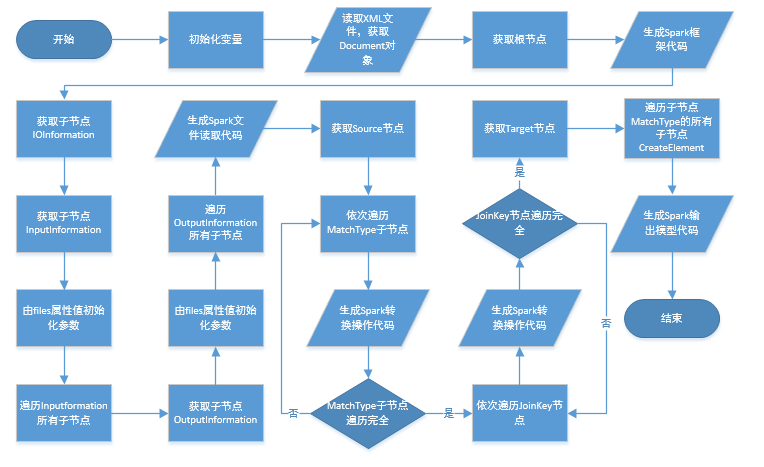


图5-1 XML解析流程图

在XML的解析过程中，生成的Spark代码主要分成四个部分：Spark配置代码、Spark文件读取代码、Spark转换操作代码以及Spark输出模型代码。而其中最主要的Spark转换操作代码又分为五个部分：Filter转换操作代码、Redundancy转换操作代码、Statistic转换操作代码、ModifyValue转换操作代码、JoinKey转换操作代码。

## 5.2 翻译引擎的实现

从上一节可以看出，XML解析流程基本固定。而在Spark代码的生成过程中，尤其是转换操作代码生成中，抽象出每种转换操作的代码框架，并根据具体参数和命令对代码框架进行调整，得到一份可执行的Spark代码。下面以3.2节提供的用户信息与购买记录之间的实际应用场景为例讲解翻译引擎的实现。

### 5.2.1 Spark配置代码

Spark应用由一个驱动器程序来发起集群上的各种并行操作，驱动器程序包含一个main函数作为程序入口。并且为了让Spark代码可以打包执行，我们需要创建SparkContext对象来连接计算集群。Spark配置代码如图5-2所示。

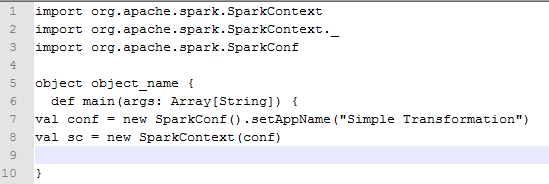


图5-2 Spark配置代码

其中object\_name根据XML解析时根节点的名称进行替换，从而得到具体的Spark框架代码，如3.2节的例子用Transformation替换object\_name。

### 5.2.2 Spark文件读取代码

对于Spark平台上的模型转换，第一步就是读取输入数据模型。我们需要读取全部输入数据，并以RDD的形式进行存储。通过遍历InputInformation节点的所有子节点，包括InputPath节点、InputFormat节点、InputType节点，我们获取了有关输入文件的具体信息，包括文件的路径、输入数据的格式。其对应的Spark框架代码如下所示：

val m1=sc.textFile(“inputpath”)

val m2=m1.map(map\_function)

其中m表示变量名，采用a到z的命名方式，从InputPath的id信息获取具体存储到哪个变量中。通过在变量后增加阿拉伯数字表明对同一变量操作后的新的数据集。其中map\_function将数据一一映射，从String数组类型变化成相对应的数据类型的数据数组。

### 5.2.3 Spark转换操作代码

在依次遍历每个转换操作时，每遍历一个，便生成相对应的Spark转换操作代码。转换操作包括Filter、Redundancy、Statistic、ModifyValue、JoinKey。

1. Filter转换操作代码

当遍历到Filter为标签的转换原语时，其对应的Spark转换操作框架为：

val m2=m1.filter(operation)

其中operation由Filter标签的具体参数决定，m1、m2由上层MatchType的id属性值以及当前模型转换次数共同决定。

1. Redundancy转换操作代码

当遍历到Redundancy为标签的转换原语时，其对应的Spark转换操作框架为：

val m2=m1.map(divide\_operation)

val m3=m2.reduceByKey((x,y)=>x)

val m4=m3.map(map\_operation)

其中divide\_operation代表了对数据的分割操作，map\_operation代表了对数据的映射操作。而m1,m2,m3,m4由上层MatchType的id值以及当前模型转换次数共同决定。

需要注意的是，因为我们在第一步分割操作时，将原数据中各个字段的顺序重新排列，所以我们需要在第三步数据的映射函数中将新的字段顺序映射到原来的字段顺序上，来保证后续转换操作的正确性。

1. Statistic转换操作代码

当遍历到Statistic为标签的转换原语时，需要根据相应target和type属性值选择相应的Spark框架代码。

当target值为average时，其对应的一部分Spark框架代码为：

val sum=m1.aggregate((0.0,0))((acc,value)=>{

if(value.\_x=="N/A")

(acc.\_1,acc.\_2+1)

else

(acc.\_1+value.\_x.toDouble,acc.\_2+1)

},(acc1, acc2)=>(acc1.\_1+acc2.\_1,acc1.\_2+acc2.\_2))

val avg=sum.\_1/sum.\_2.toDouble

其中value.\_x表示字段位置，由Statistic标签内容决定。

当target值为max时，其对应的一部分Spark框架代码为：

val temp=m1.map(values=>values.\_x)

val avg=temp.max()

当target值为min时，其对应的一部分Spark框架代码为：

val temp=m1.map(values=>values.\_x)

val avg=temp.min()

当type值为none时，表示为属性统计操作，此时不需要额外的Spark代码；

当type值为autofill时，表示为属性修改填充操作，其对应的一部分Spark框架代码为：

val m2=m1.map(x=>

if(x.\_x=="N/A")

{val y=avg

(new order) }

else

{val y=x.\_x

(new order) }

)

其中x.\_x表示字段位置，new order为替换上新的属性值的各字段顺序，均由Statistic标签内容决定。

1. ModifyValue转换操作代码

当遍历到ModifyValue为标签的转换原语时，需要根据相应range和type属性值选择相应的Spark框架代码。

当range属性值为average、max、min时，此时的Spark代码框架与生成Statistic标签时一部分的Spark代码相同。

当range属性值为阿拉伯数字时，即对应变量avg的值即为阿拉伯数字，其对应的Spark框架为：

val m2=m1.map(x=>x.\_x+number)

其中x.\_x表示字段位置，number为阿拉伯数字。

例如当需要将所有顾客等级提升一级，其转换原语如下所示：

<ModifyValue name=”cus\_ip” type=”+” range=”1”>

</ModifyValue>

此时对应的Spark代码如下所示：

val a9=a8.map(x=>x.\_5+1)

1. JoinKey转换操作代码

当遍历到JoinKey为标签的转换原语时，其对应的Spark转换操作框架为：

val temp1=m1.map(divide\_operation)

val temp2=n1.map(divide\_operation2)

val temp3=temp1.join(temp2)

val p1=temp3.map(map\_operation)

其中divide\_operation、divide\_operation2代表了对数据的分割操作，map\_operation代表了对数据的映射操作。而m1,n1,p1由JoinKey标签的first、second属性值以及文本内容所决定。

### 5.2.4 Spark输出模型代码

将所有转换操作的操作原语遍历结束，就开始遍历Target节点，进行模型的输出。每遇到一个MatchType标签，便开始一个模型的输出。通过遍历MatchType标签的所有子节点，确定每个位置的字段。其对应的Spark框架代码为：

val result=c1.map(map\_operation)

result.saveAsTextFile("output\_path")

其中map\_operation代表了对数据的映射操作，output\_path代表了输出模型的路径。

## 5.3 本章小结

本章首先对XML的解析过程进行介绍，并具体介绍了常用的解析XML的操作，并给出了一个清晰的该工具解析XML的流程图。然后本章详细介绍了翻译引擎的实现，将自动生成的Spark代码分成四部分：Spark配置代码、Spark文件读取代码、Spark转换操作代码以及Spark输出模型代码。结合转换原语和Spark代码的特性，解释了Spark代码的生成原理，给出了每种Spark代码的代码框架，清晰的展示了转换原语与Spark代码的映射关系，体现了该工具的可实施性。

# 6 实例研究

本章将使用一个具体的实例来详细的展示如何使用本文设计的工具实现基于模型转换的Spark代码生成技术。

## 6.1 实例研究

在该实例中，实验背景是某个全国连锁超市拥有一份会员用户信息与一份订单记录信息，需要得到一份北京成年会员用户在2004年的所有订单记录。其中会员用户信息由会员用户id、姓名、出生年份、所在地、会员等级以及会员用户工资水平组成，订单记录信息由订单号、订单时间、购买此订单的用户id、订单额以及订单状态组成，初始数据内容如图6-1和图6-2所示。这个实例涉及了模型转换的多个操作，并将这些操作完整执行一遍至多遍，来显示工具执行的正确性。



图6-1 customer.csv

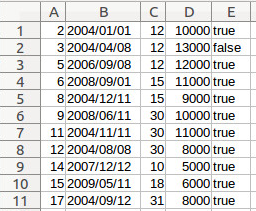


图6-2 orders.csv

实验过程中对数据模型的操作包括数据的去重、数据的筛选、数据属性值的统计、数据属性值的修改、数据模型的连接等。具体模型转换流程图如图6-3所示。

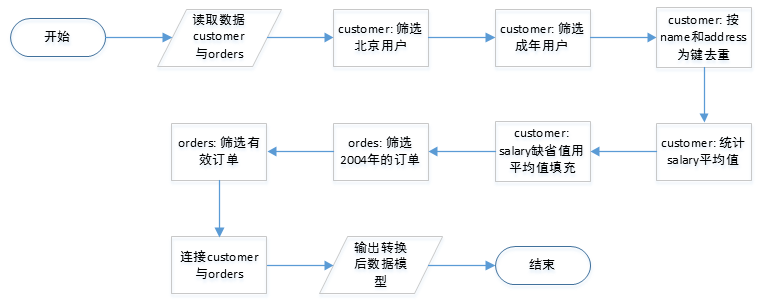


图6-3 模型具体转换流程

在Spark代码生成过程中，通过模板来生成具体的相对应的Spark代码。生成的Spark的代码分为四部分，正如5.2节介绍那样。下面将具体详细介绍其中Spark文件读取代码、Spark转换操作代码、Spark输出模型代码。

1. Spark文件读取代码

对于会员用户信息，对应的转换原语如下所示。

<InputPath id="1" name="customer"> hdfs://114.212.189.191:8020/case2/inputData/customer.csv

</InputPath>

<InputFormat id="1"> cus\_id,cus\_name,birth\_year,address,cus\_vip,salary </InputFormat>

<InputType id="1"> string,string,int,string,int,double </InputType>

通过获取相关的参数和内容，如id号、存储路径以及数据格式，得到具体的Spark文件读取代码如下所示。

val a1 = sc.textFile("hdfs://114.212.189.191:8020/case2/inputData/customer.csv").map(line=>line.split(","))

val a2 = a1.map(x=>(x(0),x(1),x(2).toInt,x(3),x(4).toInt,x(5).toDouble))

在Spark代码生成过程中，借助Spark文件读取代码模板，我们对InputType标签内容进行分割、统计字段总个数为6个并依次记录每个字段的类型，并由每个字段的类型对各个字段进行的相应类型转换。

1. Spark转换操作代码

在本实例中，设计到的转换操作原语标签包括Filter、Redundancy、Statistic、JoinKey标签。

1. 对于会员用户信息，我们需要过滤不需要的用户信息，来获取成年的北京会员用户信息，其对应的转换原语如下所示。

<MatchType id="1">

<Filter name="address" type="==">beijing</Filter>

<Filter name="birth\_year" type="&lt;">1998</Filter>

在解析过程中，通过MatchType中id属性值为1，得到操作的模型为a。通过Filter中属性name，可以获知操作的属性字段，通过属性type以及文本内容，得到整个转换操作函数。其对应的Spark代码如图6-4所示。



图6-4 Filter的Spark代码

在Spark代码生成过程中，借助Spark Filter转换操作代码模板，通过读取address和birth\_year，我们根据文件读取时保存的各字段名，计算得到address和birth\_year字段在所有字段中的位置分别为第4个和第3个，得到具体的上述Filter的Spark代码。

1. 为了防止会员信息中存在重复的用户信息，需要以用户名和地址对用户信息进行去重，其转换原语如下所示。

<Redundancy>cus\_name,address</Redundancy>

此时对应的Spark代码如图6-5所示。



图6-5 Redundancy的Spark代码

在Spark代码生成过程中，借助Spark Redundancy转换操作代码模板，通过读取cus\_name以及address，我们根据文件读取时保存的各字段名，计算得到cus\_name和address字段在所有字段中的位置分别为第2个和第4个，得到具体的上述Redundancy的Spark代码。

1. 从会员用户信息中可以看出部分会员的salary值存在缺省的情况，需要统计所有有效用户的salary的平均值并对缺省值进行填充，其对应的转换原语如下所示。

<Statistic target="average" type="autofill">salary</Statistic>针对转换原语中target和type的属性值，在Spark代码的生成中选择相应的Spark框架代码，此时对应的Spark代码如图6-6所示。

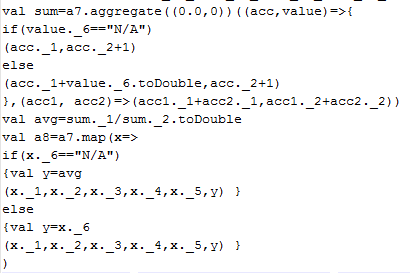


图6-6 Statistic的Spark代码

在Spark代码生成过程中，借助Spark Statistic转换操作代码模板，从target的属性值为average以及type属性值为autofill选择相应的Spark代码模板结构，并通过计算salary在原字段中的位置为第6个，填充模板中未知的代码，得到上述具体的Statistic的Spark代码。

1. 在对会员用户信息和订单记录做了以上相应操作后，我们需要对这两种信息连接起来，主要通过信息中共有的用户id进行连接，其转换原语如下所示。

<JoinKey first="1" first\_ele="cus\_id" second="2" second\_ele="cus\_id" type="Join">3</JoinKey>

通过first、first\_ele、second以及second\_ele属性值获知所连接的两张表以及连接的键在各自表中的位置，此时对应的Spark代码如图6-7所示。

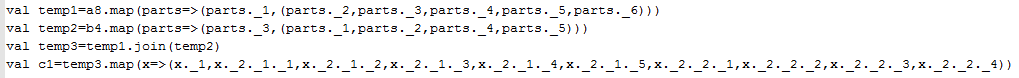


图6-7 JoinKey的Spark代码

在Spark代码生成过程中，借助Spark JoinKey转换操作代码模板，通过cus\_id属性名，我们根据文件读取时保存的各字段名，计算得到其在a和b两张表中的字段位置分别为第1个和第3个，并由内容3将新连接后的表存储在变量c中，得到具体的上述Filter的Spark代码。

1. Spark输出模型代码

最后我们需要对转换后的模型按一定的规则输出，其转换原语如图6-8所示。



图6-8 输出模型转换原语

通过转换原语中MatchType中id的属性值，我们获知输出模型存储在变量c中，并通过order的属性值与模型字段的对应性，得到相应的映射函数，此时对应的Spark代码如下所示。

val result=c1.map(

x=>(x.\_1,x.\_2,x.\_3,x.\_4,x.\_5,x.\_6,x.\_7,x.\_8,x.\_9,x.\_10))

result.saveAsTextFile("hdfs://114.212.189.191:8020/case2/outputData/result/")

在Spark代码生成过程中，借助Spark 输出模型转换操作代码模板，由MatchType的id属性值3获知，输出模型存储在变量c中。接下来按order顺序存储相对应的属性名，并通过与变量c存储的属性名进行匹配，得到其在变量c中相对应的位置，按order顺序映射每一个字段，得到上述具体的Spark输出模型代码。

通过在Spark平台上执行自动生成的Spark代码，该实例在分布式集群上的数据结果如图6-9和图6-10所示。经过验算，实验结果与理想预期结果相同。

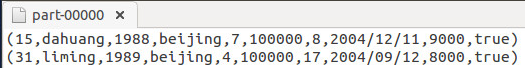


图6-9 slave1的结果

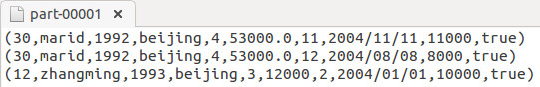


图6-10 slave3的结果

## 6.2 本章小结

本章对模型转换的Spark代码生成工作提供了具体的现实场景例子与数据，并具体解释了对应的Spar代码的生成过程，很好的说明了本工作的可实施性。

# 7 总结

本章首先对本文工作进行总结，分析该工作的现实应用的可行性，以及分析该工作的优缺点，给出下一步工作的计划。

## 7.1 课题总结

本文结合Spark集群平台以及模型转换的基本操作，为数据模型转换这一应用场景设计了自动生成Spark代码的工作。本文从模型转换的基本操作入手，设计出包括基本转换操作命令的转换原语等配置文件。接着通过翻译引擎流程的介绍，详细给出了Spark代码生成的步骤。从实验例子中可以看出，该工具向用户屏蔽了Spark代码的细节以及Spark分布式平台的使用，明显降低了用户使用Spark平台实现模型转换的困难，同时该工作也适用于海量数据操作的需求。

## 7.2 不足和展望

本文采用流程式翻译转换原语生成Spark代码，其中需要的转换操作较少。但当模型较为复杂时，需要转换的操作较多时，由于采用流程式产生Spark代码，并未对产生后的Spark代码进行优化，可能会产生混洗数据较大的情况，增多了网络数据传输的量，消耗更多的系统资源，很大程度上降低了Spark平台的处理效率。在今后工作中，我们可以优化产生后的Spark代码来提供效率，进一步提高整个工具的效率。

另外，本工具涉及到模型转换的操作是一些常见的基本操作。下一步我们可以适当增加模型转换的操作，增加一些常用的复杂操作，完善该工具的功能。

# 参考文献

[1] 段永威, 秦峰 异构数据源数据转换工具的设计与实现[J]-现代图书情报技术 2004(4).

[2] 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.

[3] 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.

[4] Spark Network. <http://spark.apache.org/>.

[5] Holden Karau, Andy Konwinski, Patrick Wendell, Matei Zaharia. Learning Spark: Lightning-fast Data Analysis. O’Reilly Media, Inc. 2015.

[6] Hadoop Network. http://hadoop.apache.org/.

[7] Mohd Rehan Ghazia, Durgaprasad Gangodkara. Hadoop, MapReduce and HDFS: A Developers Perspective. ICCC-2015, Procedia Computer Science 48 (2015) 45-50.

[8]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.

[9] Model-Driven Engineering Douglas C. Schmidt IEEE Computer Society Feb 2006.

[10] 王勇涛，刘勇. 基于MDA的模型转换研究与应用[A]. 计算机工程， 2011: 37-16.

[11] Code Generation Network. <http://www.codegeneration.net>.

[12] Ákos Frohner, Zoltán Porkolab and László Varga. Code Generation from UML Models. Periodica Polytechnica. SER. EL. ENG. VOL. 44, NO.2, PP. 141-157 (2000).

[13] Ralf Gitzel, Martin Schader. Genration of XML-Based Web Application using MetaModels. University of Mannheim, Germany.

[14] Ivo Damyanov, Nick Holmes. Metadata Driven Code Generation Using .NET Framework. Internatonal Conference on Computer Systems and Technologies-CompSysTech’2004.

[15] Burginsky F, M Finnie, P Yu. Automatic Code Generation from Design Patterns. IBM Systems Journal, Volume 35, #2, 1996.

[16] Alexander Knapp and Stephan Merz. Model Checking and Code Generation for UML State Machines and Collaborations. http://www.pst.informatik.uni-muenchen.de/projekte/hugo/.

[17] 韩非 多种平台下异构数据转换方法的研究与实现[学位论文]硕士 2011.

# 致谢

在论文完成之际，首先，我要特别感谢我的指导老师——张天副教授。张天老师严肃的科学态度、严谨的治学精神以及精益求精的工作作风，深深感染激励着我不断向前探索、前进。他从选题、设计方向、毕业设计的每个细节以及最后毕业论文的撰写都以严谨负责的态度指导着我。其次我要感谢软工实验组的车开达学长和刘拨杰学长。两位学长在初期Spark集群架构环境的建立方面、以及毕业设计过程中转换原语的设计方面都给予了我很大帮助。

感谢计算机科学与技术院系的各位领导和老师对我的栽培和教育。他们不仅从教学、知识还是做人方面，都给与我很多宝贵的意见的启迪，帮助我不断成长。

最后我要感谢所有本篇论文的参考文献的专家和学长。他们之前的研究工作为本文工作的开展提供了宝贵的参考和建议。