****



**本 科 毕 业 作 品**

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Spark平台的数据转换方法

年 级 2013 学 号 131220126

学生姓名 杨宇飞

指导教师 张天 职 称 副教授

提交日期

**基于Spark平台的数据转换方法**

**Data Transformation Method Based on Spark Platform**

**南京大学本科生毕业论文（设计、作品）中文摘要**

题目：基于Spark平台的数据转换方法

院系：计算机科学与技术系

专业：计算机科学与技术

本科生姓名：杨宇飞

指导教师（姓名、职称）：张天 副教授

摘要：

随着互联网和信息技术的不断发展，人们每天产生的数据呈爆炸式增长，其中包含了海量的异构数据，传统的数据转换方法已经很难满足大量数据转换的需求，因此衍生出了多种基于集群的并行计算框架用于大规模数据的计算处理。但是，由于大数据处理平台的底层实现，编程模型以及编程细节对于普通的开发者理解起来具有一定的难度，具有较高的学习成本，因此，一种屏蔽平台复杂性，便于开发者理解使用的大数据转换方法就尤为重要。

基于以上想法，本文提出了一种基于模型转换思想的数据转换方法，该方法从模型转换的角度，将大数据转换的过程理解为模型转换，利用简单的模型转换描述代替复杂的底层代码实现，从而简化用户使用大数据平台的复杂性。为了实现该方法，本文首先结合实际场景对Spark平台中常用的转换操作进行模型层的描述。然后，设计实现了一个代码生成引擎，对模型层的转换描述和数据模型进行解析，并翻译生成对应的Spark代码。最后，对一个简单应用场景进行了实例分析，实验结果表明本方法可以简单有效地完成数据转换功能。

关键词：模型转换；大数据转换；Spark平台；代码生成

**南京大学本科生毕业论文（设计、作品）英文摘要**

THESIS:

DEPARTMENT: Department of Computer Science and Technology

SPECIALIZATION: Computer Science and Technology

UNDERGRADUATE: Yufei Yang

MENTOR: Associate Professor. Tian Zhang

ABSTRACT:

With the continuous development of Internet and information technology, the data that people produce every day has been exponentially grown, which contains massive heterogeneous data. The traditional data conversion method has been difficult to meet the needs of a large number of data conversion, resulting in a variety of Cluster-based parallel computing framework for large-scale data processing. However, due to the bottom implementation, programming models and programming details of the large data processing platform is difficulty for ordinary developers to understand. Therefore, a big data conversion method which could mask the complexity of the platform and be easy for developers to understand and use is very important.

Based on the ideas above, this paper presents a data conversion method based on the thought of model transformation. This method considers the process of large data conversion as the model conversion from the perspective of model transformation, uses the model transformation description to replace the complex underlying code in order to simplify the complexity of large data platforms for users. In order to implement this method, first this paper designs the model description of the conversion operation commonly used in the Spark platform with the help of actual scene. Then, this paper design and implement a code generation engine, which could resolve the transformation description and data model of the model layer, and then translate them to generate the corresponding Spark code. Finally, this paper analyses a simple application scenario and the experiment results show that the method can complete the data conversion task simply and effectively.

KEY WORDS: Model Transformation; Big Data Conversion; Spark Platform; Code Generation

目录

[1. 引言 1](#_Toc482479428)

[1.1 研究背景 1](#_Toc482479429)

[1.2 研究意义 1](#_Toc482479430)

[1.3 本文工作 2](#_Toc482479431)

[1.4 本文结构 2](#_Toc482479432)

[2. 技术背景 4](#_Toc482479433)

[2.1 QVT标准 4](#_Toc482479434)

[2.1.1 QVT概述 4](#_Toc482479435)

[2.1.2 QVT-Relation 5](#_Toc482479436)

[2.2 Eclipse模型框架 5](#_Toc482479437)

[2.3 Spark平台 6](#_Toc482479438)

[2.3.1 Spark RDD编程接口 7](#_Toc482479439)

[2.4 相关工作 8](#_Toc482479440)

[3. 整体分析与设计 9](#_Toc482479441)

[3.1 开发环境 9](#_Toc482479442)

[3.2 工具整体框架 9](#_Toc482479443)

[3.3 模型层描述思路 10](#_Toc482479444)

[3.3.1 QVT-Relation与Spark Api 10](#_Toc482479445)

[3.3.2 Spark Api的选取 10](#_Toc482479446)

[3.4 Spark代码生成思路 11](#_Toc482479447)

[3.4.1 Ecore元模型解析 11](#_Toc482479448)

[3.4.2 QVT描述解析 12](#_Toc482479449)

[3.5 本章小结 12](#_Toc482479450)

[4. 转换规则的设计 13](#_Toc482479451)

[4.1 模型语言的选取 13](#_Toc482479452)

[4.2 QVT-Relation语法介绍 13](#_Toc482479453)

[4.3 转换操作的模型描述 15](#_Toc482479454)

[4.3.1 Map操作的QVT描述 15](#_Toc482479455)

[4.3.2 Filter操作的QVT描述 16](#_Toc482479456)

[4.3.3 Distinct操作的QVT描述 17](#_Toc482479457)

[4.3.4 Join操作的QVT描述 17](#_Toc482479458)

[4.3.5 集合操作的QVT描述 19](#_Toc482479459)

[4.4 本章小结 20](#_Toc482479460)

[5. 代码生成引擎的实现 21](#_Toc482479461)

[5.1 数据模型的代码生成 21](#_Toc482479462)

[5.1.1 数据建模 21](#_Toc482479463)

[5.1.2 Ecore解析 21](#_Toc482479464)

[5.1.3 生成数据类代码 22](#_Toc482479465)

[5.2 QVT描述的代码生成 23](#_Toc482479466)

[5.2.1 QVT语法解析 23](#_Toc482479467)

[5.2.2 转换代码生成算法 25](#_Toc482479468)

[5.3 本章小结 27](#_Toc482479469)

[6. 实例研究 28](#_Toc482479470)

[6.1 实例研究 28](#_Toc482479471)

[6.2 本章小结 33](#_Toc482479472)

[7. 总结 34](#_Toc482479473)

[7.1 工作总结 34](#_Toc482479474)

[7.2 不足与展望 34](#_Toc482479475)

[参考文献 35](#_Toc482479476)

[致谢 37](#_Toc482479477)

# 引言

## 研究背景

随着现代社会的发展，无论是个人还是企业都希望对数据进行高效的利用，这往往需要对庞大的数据集进行数据转换操作，将数据从一种表示形式转化为另一种表示形式，从而方便用户进行其他操作。然而，在互联网、物联网不断发展的时代背景下，数据信息呈指数型增长，传统的数据转换方法得到的效果并不理想，已经不能满足爆炸式增长数据的需求，因此，现在比较流行的大数据数据处理方法是在大数据支撑平台上进行处理。

目前公认的主流大数据处理平台为基于MapReduce[1]思想设计的Spark和Hadoop，他们都可以将庞大的数据存放到集群上，然后利用集群的强大计算性能对数据进行操作。然而，经过这些大数据平台已经在一定程度上向用户屏蔽了底层细节，但这些平台在使用上的学习成本仍然很高，用户必须要了解该平台的底层架构和平台API，并且具有一定的编码能力之后才能对大数据进行处理。我们的平台希望提供一种屏蔽平台复杂性，简化问题求解的一种数据转换方法。

## 研究意义

本工作提出了一种基于Spark平台的利用模型思想进行大数据转换的方法。本实验方法将数据理解为模型，将数据转换看作是模型转换[2]，开发人员只需要对所处理的数据进行建模，然后在模型层根据用户希望进行的数据转换操作进行数据变换的描述，之后代码生成引擎就可以将这些模型层的转换规约生成具有相同数据转换功能的Spark代码。由于模型层的转换描述抽象层次很高，因此用户可以不用关心转换操作的底层实现细节，只需要在顶层关注数据格式和数据内容的变换就可以完成数据的转换操作。

随着大数据时代的发展，处理数据的规模也越来越大，用户对于大数据平台的应用也会越来越频繁，通过这种模型转换的思想，可以大大减少了用户对于大数据处理平台的学习成本，在不了解大数据并行计算框架底层细节和其编程语法的情况下，对大数据进行数据转换操作，从而便于普通用户使用。

## 本文工作

本文基于Spark平台抽取出几种常用Spark转换操作API，然后在模型层使用QVT标准对数据转换操作进行描述，作为模型层数据转换的基本操作，并设计开发了由模型层数据转换描述生成Spark代码的工具。本文首先介绍了关于本工作的一些技术背景，包括QVT的介绍与使用、Eclipse模型框架、分布式Spark集群平台、大数据模型转换等相关工作。其次，本文对整个工作的实现框架进行介绍，并给出了本项工作几个研究难点的解决方案。接着，对模型转换描述规则的设计进行了详细的说明，并用工具验证了其正确性。接着，分为Ecore元模型和QVT转换描述两个部分，对代码生成引擎的实现细节进行了详细介绍。然后，通过一个实际应用场景展示了此工具的可行性，并作对比试验证明了生成代码的可用性。最后，对整个工作进行了总结，指出了本工作还存在的不足并讨论了下一步需要改进的地方。

## 本文结构

本文一共分为七个部分。

第一部分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。

第二部分介绍了有关QVT，Ecore，Spark等技术背景，并对与本文相关的科研工作进行了调研和总结。

第三部分首先介绍了本工具的开发环境，然后给出了本工具的整体实现框架，清晰的介绍了本工作的结构和功能，最后对一些实现难点进行了实现方法的分析，为后面的具体实现提供参考。

第四部分在模型层设计了几种转换操作的描述规则，并通过实验工具验证了设计的正确性，是后面转换操作代码生成工作的基础。

第五部分介绍了代码生成引擎的实现过程，对其中的算法实现过程和模型层与代码之间的对应关系等内容进行了详细的说明，最后结合输入参数信息完成代码生成引擎的实现。

第六部分通过一个现实的应用场景对本文设计实现的工具进行了详细的展示和说明，并利用对比实验说明了生成代码的高可用性。

第七部分对本文的工作进行了总结并指出当前工作的不足和对下一步工作的计划。

# 技术背景

本章将对本项目所涉及到的相关工具或技术进行介绍，其中将着重介绍QVT标准，Eclipse模型框架和Spark平台这几个本项目最底层的知识内容。

## QVT标准

QVT（Query/View/Transformation）[3][4]是由对象管理组织（OMG）为模型转换定义的一组标准语言，用于对模型驱动工程中最常使用的一项重要技术——模型转换进行规范和实现。从QVT的名字中我们就可以看出，该标准包含了查询，视图和转换三种特性，其中模型查询和模型视图都可以看作是特殊类型的模型转换。

### QVT概述

QVT标准包含了三种模型转换语言：QVT-Operational、QVT-Relation和QVT-Core，其中QVT-Operational用于编写单向转换的命令式语言；QVT-Relations是一种声明式语言，允许单向和双向模型转换。可以通过执行检验模式下的转换来检查模型的结构，如果模型集合与转换中定义的结构一致，结果就会返回True，否则返回False，在强制模式中可以使用相同的转换规则来尝试修改其中一个模型，以使模型集合保持一致；QVT-Core是一种声明性语言，设计简单，并作为QVT关系翻译的目标。但是，QVT-Core还没有过全面的实施，实际上并没有像QVT-Relation那样具有表现力，QVT-Relation与QVT-Core的关系如图2-1所示。

这三种语言都是对符合元对象机制（MOF）[5]的元模型进行操作的，即这三种QVT语言都是用Ecore来表示模型结构的，而且对于这三种语言描述的转换本身也可以看作是符合标准的元模型之一。除此之外，QVT标准还集成了OCL2.0[6]标准，从而扩展了其必要的功能。

### QVT-Relation

QVT-Relation是本工作重点研究和使用的模型描述规范，它用声明式的方法描述了MOF模型之间的关系，该声明式语言还支持复杂的对象模式匹配，可以隐式地创建Trace类用来记录转换执行过程中发生的模型变换情况。同时，QVT-Relation可以进行断言，其他关系也可以由其模式匹配的特定模型元素之间保持。它可以纯粹地用作关系的形式语义，或者将关系模型转换作为核心模型在引擎上执行从而实现其核心语义。

基于这些特性，我们可以看出QVT-Relation是通过声明源模型域和目标模型域的变量特征来表示模型转换的，这样的描述体现了在模型层进行数据转换描述的优势，使数据转换的过程更加的直观，屏蔽了模型转换的实现细节，因此本文使用QVT-Relation对数据转换进行模型层的描述，实现细节会在后文进行介绍。

## Eclipse模型框架

Eclipse模型框架（EMF）[7][8]是一个基于Eclipse的建模框架和代码生成工具，用于一些基于结构化数据模型的构建工具或应用程序，目标是为了实现模型驱动的开发。根据XMI中描述的模型规范，EMF提供了工具和运行环境，满足模型可以生成Java类，通过编辑Java类中内容修改模型内容等功能，同时提出了一个基本编辑器。

EMF的模型编辑器可以通过添加了注释的Java文件，XMI文件或UML模型作为工具的输入，并且可以实现这三者之间的模型转换，如图2-1，从图中可以看出EMF的核心就是Ecore，大多数基于EMF的模型驱动工具都可以支持Ecore的设计，这也是本文选择QVT用来描述模型转换的原因。

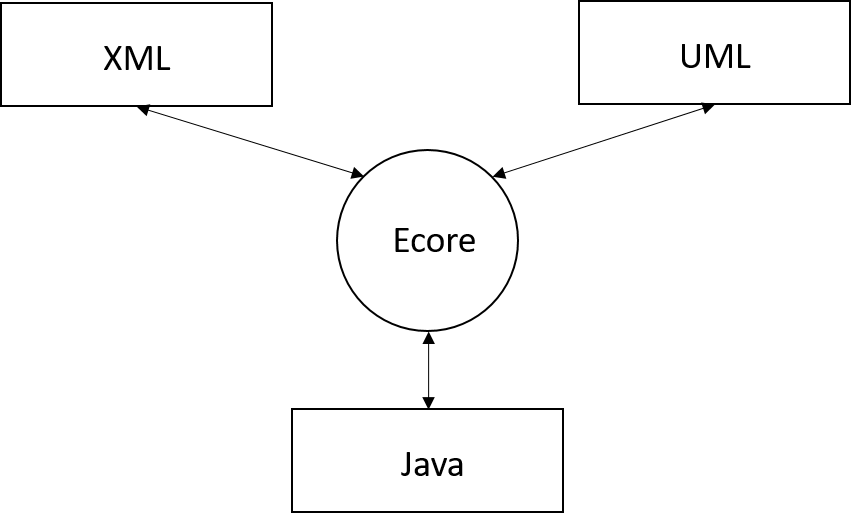


图2-1 EMF框架

Ecore模型表示EMF中的元元模型，是EMOF的一种实现，用来抽象XML，UML或Java类表示的模型，其中模型的基本元素有四个，分别为：

（1） EClass：领域类，它有一个名字，一个或多个属性还有一个或多个引用

（2） EAttribute：领域类的属性，属性有一个名字和类型

（3） EReference：在Class之间的关系，它具有一个名字，一个布尔值标识用来表示是否包含还有一个用于引用指向的目标类

（4） EDataType：属性类型，例如int或者对象类型

本文使用Ecore对数据进行建模，用作模型转换的设计以及后续Spark数据类的代码生成。

## Spark平台

为了处理不断产生的大量数据，但一台机器的处理能力和I/O性能已经不能满足需求，为了解决这一问题出现了大量的大数据编程框架，其中最著名的就是Google的MapReduce，提供了一个简单通用并自动容错的批处理计算模型。

但MapReduce框架中存在着重复工作，组合计算，资源分配等问题，为解决这些问题，伯克利大学推出了全新的大数据处理框架Spark[9]，并提出了RDD的概念，通过在并行计算的各个阶段实现有效的数据共享，将各种计算有效表达，并提升计算性能。除了在功能和性能上的优化外，在用户使用方面，Spark也具有很强的简易性，它提供了多种类型的Api供用户直接使用。Spark还支持多种资源管理器，并含有多种外部库供用户使用[10]。

### Spark RDD编程接口

RDD(Resilient Distributed Datasets)是弹性分布式数据集，代表着一个分区的只读数据集，是Spark编程中重要的抽象概念。针对Spark RDD的主要操作有四种，创建操作，转换操作，行动操作和控制操作。

（1） 创建操作

RDD的初始创建都是由SparkContext来负责的，将内存中或外部文件作为输入源。

简单实例：val sc = new SparkContext()

val file = sc.textFile(“ReadMe.md”)

（2） 转换操作

将RDD通过一定的操作变换为另一个RDD，比如一个RDD通过过滤操作生成一个新的RDD，filter就是一个转换操作，常用的转换操作如下所示。

Map()：将RDD中的元素一对一地映射为另一种类型的元素；

Distinct()：返回RDD中所有不一样的元素；

filter()：返回RDD中符合过滤条件的元素集合；

join()：对RDD中的键值进行连接操作；

union(),intersection(),subtract()：对RDD进行集合操作；

（3） 行动操作

行动操作是与转换操作相对应的一种对RDD的操作类型，行动操作会对数据集进行统计计算，并将结果写入外部系统，常用的行动操作如下所示：

Count()：返回RDD中元素的个数；

Reduce()：对RDD中元素进行二元计算，并返回计算结果；

Collect()：以集合形式返回RDD中元素；

Aggregate()：将所有分区中元素进行聚合在一起，然后利用Conbine函数将这些元素合并；

（4） 控制操作

用来对RDD持久化，将RDD中的内容保存到磁盘或内存中，保存下来的信息可以降低重复使用的成本。常见的控制操作有cache(),persist()等函数，它们会将RDD持久化到不同层次的存储介质中。

由于本文的工作是对数据进行转换处理，所以工作重点将放在RDD的转换操作上，具体细节会在后面的实现中进行详细介绍。

## 相关工作

本文首先对工业界的成熟工具进行了调研，Hadoop数据仓库工具Hive[11]，PassLab提出的Octopus[12]和Google最近刚刚提出的Beam[13]等工具都是各公司机构为了降低大数据平台使用的复杂性而创造出来的，虽然这些工具在一定程度上降低了并行计算框架的使用难度，但他们的抽象层并没有降低，对于开发者仍然具有较高的学习成本。

本文工作主要设计了两个领域的研究内容，一个是大规模模型的转换技术，另一个是对QVT语言的转换方法，所以本文也对这两个方面进行了全面的调研。

首先对于第一个方面，法国的AtlanMod研究组提出了在MapReduce分布式系统上进行ATL模型转换的方法来并发地对大数据模型进行处理[14][15]，他们的解决方法是将转换规则拆分到集群的各个机器上进行转换操作，执行完成后在合并生成的子模型完成大数据模型转换；还有一组研究与我们的工作比较类似，他们利用ATL描述模型的转换规则，然后将转换规则生成执行转换操作的java代码，在搭建的Linda+XAP分布式架构运行java代码，完成大数据模型的并行处理[16]，但他们的工作中的数据存储方式具有一定的局限性，不支持现在广泛使用的json，csv等数据存储文件的处理。

第二个方面，关于对QVT描述的生成另一种转换语言方面，找到了德国研究者的一篇论文，他们实现了QVT与另一种声明式描述TGG的互相转换[17]，但他们的工作是基于QVT-Core进行的，并且QVT与TGG之间的对应性较强，所以在这两个方面本文工作具有更大的实现难度。

# 整体分析与设计

本章将介绍本文工具的开发环境，接着基于Spark平台进行数据转换操作的具体分析，完成工具框架的搭建，然后对整个转换工具的几个技术难点的实现思路进行了分析。这些内容都会为后面在QVT模型层设计转换操作描述和翻译引擎的实现提供参考。

## 开发环境

本文实现的代码生成引擎是在Java 1.8的环境下进行开发的，所使用的开发工具为Eclipse，开发时使用到的Dom4j，JavaCC等工具可以很容易的通过添加Eclipse插件的方式进行配置，配置好环境之后，准备所需要的模型文件，即可运行。

在生成Spark代码之后，我在本机搭建了一个测试环境，在Intellij idea中可以利用maven工具搭建出一个比较稳定的Spark运行环境，其中Spark版本为1.6.0，scala版本为2.10，hadoop版本为2.6.0.

在本机测试功能通过后，还需要在集群上测试生成代码对于大规模数据的可用性，因此又在校内的云平台搭建了一个包含1个Master节点和4个Worker节点的集群，每个节点在运行时使用4核处理，每个核的运行内存为4G，集群环境的Spark版本为2.1.0，scala版本为2.10，hadoop版本为3.0.0。

## 工具整体框架

本文的工作是希望实现一个由模型层的数据转换描述生成Spark转换操作代码的工具，所以我首先制定了本文实现工具的整体框架，如图3-1所示。

该框架中的源元模型和目标元模型都用Ecore文件进行表示，用来描述数据转换前后的模型结构，该结构与底层的真实数据的结构是一致的；模型层的数据转换用QVT-Relation进行描述，根据制定的模式规则识别出对应的Spark Api，再通过一系列的解析过程，将模型层的数据转换描述自动生成对应的Spark代码，该生产的Spark代码可以直接在Spark平台上运行并实施数据转换操作。

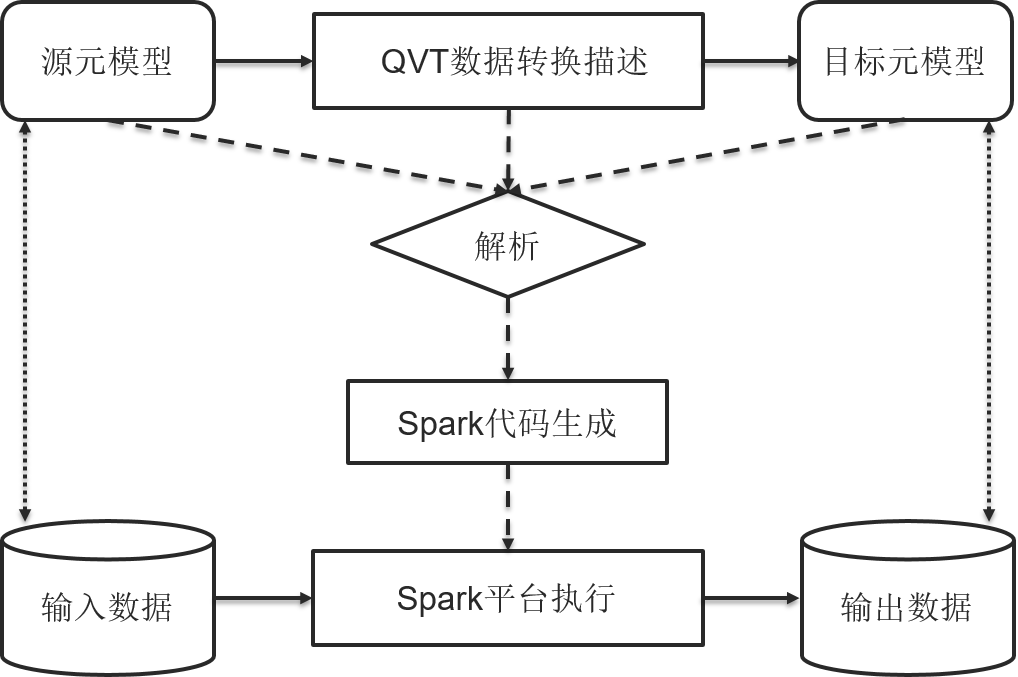


图3-1 工具整体框架

## 模型层描述思路

### QVT-Relation与Spark Api

在设计好工具实现的框架之后，就开始思考如何对其中的细节进行实现。第一个要考虑的就是如何在模型层对数据转换进行描述的问题。

我们在模型层采用了QVT-Relation进行数据转换描述，它的描述功能很强大而且清晰，通过数据的对应关系表示出多种模型间的转换对应关系，但我们要做代码生成不可能针对每种描述都进行特定的分析生成，所以就首先要找到QVT-Relation与Spark代码之间的对应关系。

在Spark代码中，数据的转换都是通过Spark平台提供的转换操作Api来实现的，这也就提醒了我们可以将QVT-Relation中的各种描述映射到不同的Spark Api上，从而实现Spark的代码生成。

### Spark Api的选取

虽然我们确定了用QVT-Relation描述Spark Api功能的方式进行Spark代码的生成，但是Spark提供的编程接口有上百种，我们不可能将所有的编程接口都在模型层进行一个描述实现，因此，我们只能选择对Spark Api进行适当的裁剪。

在选取Spark Api的过程中，我们发现上百种Api中有大部分是用来支持sql，机器学习等外部库的，与我们关注的数据转换是没有关系的，我们只需要关注与RDD操作有关的数据转换操作，即在2.4.2中介绍的Spark RDD转换操作。

但是Spark RDD转换操作中提供的Api仍然有几十种，这对于一个人的工作来说仍然很多，所以必须继续向下进行选取精确到个位数的编程接口。在对Spark RDD转换操作进行研究时，我发现有很多操作时用于优化Spark平台，提升运算效率的，比如其中很多Api都涉及到了partition分区这个概念，通过分区决定并行计算的粒度，从而可以充分利用计算资源，提高计算效率。所以，这些与平台相关的Spark编程接口就不在我们的考虑范围之内了。

最终，我选择了map()，distinct()，filter()，join()，union()，intersection()和subtract()这7种最基本，也最常用的Spark转换操作进行模型层的描述设计，和其Spark代码的自动生成。

## Spark代码生成思路

在设计好模型层的转换描述之后，要考虑的问题就是如何将模型层解析为底层Spark代码。从上文介绍的实现框架中可以看出，我们需要对Ecore元模型和QVT模型转换分别解析生成Spark代码，这就将我们的工作划分为数据模型代码生成和转换操作代码生成这两个部分。

### Ecore元模型解析

首先介绍选取Ecore元模型进行数据建模的原因，在2.1.2和2.2中对QVT-Relation和EMF模型框架的整体介绍中提到，QVT-Relation是支持MOF元模型操作的，也就说明我们使用的Ecore元模型和QVT描述之间是没有间隔的，因此我们选择使用Ecore元模型进行数据建模。

对于Ecore元模型的解析方法，因为Ecore元模型的文件结构是符合XML可扩展标记语言规范的，通过信息符号来标记数据，定义数据类型，我们的元模型中的信息也都通过这种标记的形式进行记录。这种存储形式的优势在于XML是具有结构性的标记语言，分析起来非常的简单，而且现在已经有很多解析XML的方法，主流的解决方案包括DOM、SAX、JDOM、DOM4J等，经过调研本工作决定采用对JAVA支持度较高的DOM4J[18]工具进行Ecore元模型的解析。

### QVT描述解析

经过对QVT-Relation语言的学习，了解该语言的语法之后，对其语法结构进行了一个简单的总结，如图3-2所示。从这个图中可以简单地看出QVT-Relation的语法结构，我们解析的过程中也可以按照类似的语法树结构进行语法解析和代码生成。

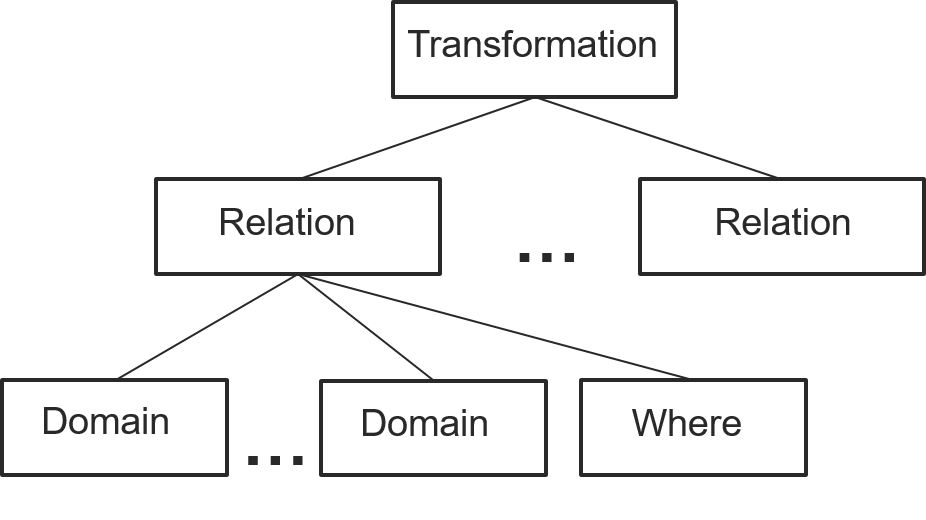


图3-2 QVT-Relation简单语法树

这个语法解析过程类似于编译的过程，但编译的工作非常的复杂，我们不可能从零开始搭建出一个编译工具，因此我们决定使用一个比较成熟的词法语法解析工具，对QVT描述进行词法检验和语法解析，然后在我们的解析工具中会生成一个抽象语法树，我们在不同的树节点进行相应的信息提取，规则识别以及代码的生成工作，在遍历完整个抽象语法树之后，生成一个完整的具有数据转换功能的Spark项目。

## 本章小结

本章主要介绍了本实验工具的整体设计架构，简单清晰地展示了整个工具的工作流程，然后进一步对翻译引擎工具的难点进行分析，并给出了这几个难点的大致实现思路，为后续的具体实现提供参考。

# 转换规则的设计

在3.2中提到了模型层描述数据转换的思路，我们希望用QVT-Relation描述出Spark RDD转换操作Api的相同功能，得到这样从模型层到底层代码之间的对应关系，从而实现自动化的Spark代码生成工具。在这一章里将首先详细讲解了QVT-Relation语言的描述语法，目的在于说明在模型层选择QVT-Relation作为描述语言的原因，然后介绍了我们如何在模型层表示出Spark的几种基本转换操作，同时设计一些简单实例并在QVT执行工具mediniQvt中运行来检验我们设计的转换规则的正确性。

## 模型语言的选取

前文已经多次提到本工作使用QVT-Relation在顶层进行数据转换的描述，因此在这里着重介绍一下选取QVT-Relation作为上层模型转换描述的原因。

首先，QVT-Relation是OMG组织为模型的查询、视图和转换所定义一种语言规范，是对象管理组字提出来的一种标准化的模型转换语言，使用广泛，具有一定的权威性和说服力。

其次，QVT-Relation是一种声明式语言，与QVT-Operational这样的命令式模型语言不同的是，声明式语言模型转换的对应关系更加简单清晰，学习成本相对较低。

最后，QVT-Relation相比ATL，TGG等模型转换语言具有的优势是，QVT-Relation提供了更加完备的模型转换操作，可以支持更加丰富的应用场景。

基于以上原因，我们采用QVT-Relation作为本工作的模型转换描述语言。

## QVT-Relation语法介绍

在确定了模型层转换语言之后，我对QVT-Relation只有一个概念上的理解，并不知道该语言的具体细节，所以我花了很长时间来学习如何使用这一种声明式转换语言QVT-Relation来进行模型转换来满足我们的需求，对于QVT-Relation的描述特点，它的语法结构非常简单，并且是一种声明式语言，所以非常适合对面向对象编程不是很熟悉的用户进行模型转换的描述，下面对它的主要语法作详细介绍：

（1） Transformation：Transformation表示候选集之间转换需要满足的一组关系，其中候选集模型是符合一种模型结构的任一模型，写法如下所示。

transformation umlRdbms (uml : SimpleUML, rdbms : SimpleRDBMS){}

（2） Relation：Relation中声明了候选模型中元素必须满足的约束条件，即转换规则。

（3） Domain：Domain可以在Relation执行时，与给定的模型进行匹配。

（4） Pattern：Pattern可以看作是一组变量，并将模型元素与这组变量进行满足条件的绑定，从而完成模型转换操作，写法如下所示。

relation PackageToSchema {

domain uml p:Package {name=pn}

domain rdbms s:Schema {name=pn}

}

（5） When and Where：When语句表示执行当前转换前必须满足的约束；Where语句表示当前执行的转换中变量需要满足的条件。

（6） Top level：一个Relation可能是top-level的，也可能是non-top-level的。top level的Relation在转换时必然会执行，而non-top-level的Relation则只有在被其他Relation调用时才会执行。

（7） Check and Enforce：Checkonly Domain一般只对域中属性与模型是否匹配进行检查，而Enforce Domain会先进行检查，检查失败就会创建模型对象。所以我们一般采用Checkonly声明源模型域，用Enforce声明目标模型域。

（8） Key：类似于数据库中表的键值，在Key语句声明Domain和属性名之后，该属性每一个属性值对应的模型只有一个。

（9） Primitive Domain：用来声明一些简单数据，例如配置信息或常量，可以作为参数传递到Relation当中。

通过上述几种QVT-Relation语法结构的介绍，大致说明了QVT-Relation描述模型转换的描述方法和执行原理，但是我们的工作将数据看作是一种模型，针对数据转换进行描述，数据转换的场景要比模型转换简单一些，所以上面介绍的一些复杂的描述语法并不会涉及，在4.3节将会利用几个转换操作的描述实例展示我们使用到的语法结构，在对QVT-Relation的语法进行有效的裁剪之后，使用起来也比较简单。

## 转换操作的模型描述

### Map操作的QVT描述

Map函数操作的作用是将原有数据集中的数据一对一的映射到目标数据集，而在QVT-Relation中，这样的数据转换关系比较容易表达，准备两个Domain，利用checkonly标记其中一个Domain作为操作的源数据域，enforce标记另一个Domain作为操作的目标数据域，将这两个数据域中的模型元素通过变量进行绑定，从而实现从源数据域取值、处理再赋值给目标数据域的过程，这样执行过程其功能与map操作类似，下面用一个简单例子来展示Map函数操作的模型描述计，如表4-1。

表4-1 Map操作模型层描述实例

|  |
| --- |
| transformation weather2life::weather2life(weather:SimpleWeather,life:SimpleLife){  top relation datatransfer {    pi, pa, pd: String;  pt : Integer;  checkonly domain weather src:Weather {  id = pi,  area = pa,  date = pd, temperature = pt  };    enforce domain life tar:Life {  id = pi,  area = pa + '\_name' ,  date = pd,  feel = if (temperature > 26) then "hot" else if (temperature > 18) then "cool" else "cold" endif;  };  }  } |

从上表中，我们可以比较清晰地看懂这段模型转换描述的功能。首先,声明定义了四个变量pi，pa，pd，pt，分别用于绑定元源模型weather中的id，area，date和temperature属性；绑定后，这四个变量的值即为源模型的这四个属性值；然后对目标模型用已绑定的变量进行操作赋值，实现模型元素的一一映射。

### Filter操作的QVT描述

Filter函数操作的作用是将原有数据集中符合过滤条件的目标数据提取出来，在QVT-Relation中，QVT-Relation自身就提供了一种在模式匹配时进行的过滤机制，只要在Domain进行属性赋值时，对模型属性添加一个约束，这个约束就将作为过滤的条件并进行过滤操作。下面用一个简单例子来展示Filter函数操作的模型描述计，如表4-2。

表4-2 Filter操作模型层描述实例

|  |
| --- |
| transformation weather2life::weather2life(weather:SimpleWeather,life:SimpleLife){  top relation datatransfer {    pi, pd: String;  pt : Integer;  checkonly domain weather src:Weather {  id = pi,  area = ‘Nanjing’,  date = pd, temperature = pt  };    enforce domain life tar:Life {  id = pi,  date = pd,  feel = if (temperature > 26) then "hot" else if (temperature > 18) then "cool" else "cold" endif;  };  }  } |

从上表中，我们可以看到在weather模型中对area添加约束“area = ‘Nanjing’”，也就相当于添加了过滤条件，把源模型中area地区不是南京的都筛选掉。

### Distinct操作的QVT描述

Distinct函数操作的作用是将数据集中的冗余数据删除，生成没有重复数据的新数据集。在QVT-Relation中，key语句表示的一个数据域中的关键字，一个关键字只会有一个对应的数据，当对key值进行设置之后即可实现类似删除冗余数据的功能。下面用一个简单例子来展示Distinct函数操作的模型描述计，如表4-3。

表4-3 Distinct操作模型层描述实例

|  |
| --- |
| transformation weather2life::weather2life(weather:SimpleWeather,life:SimpleLife){  key Weather {id};  top relation datatransfer {    pi, pa, pd: String;  pt : Integer;  checkonly domain weather src:Weather {  id = pi,  area = pa,  date = pd, temperature = pt  };    enforce domain life tar:Life {  id = pi,  area = pa + '\_name' ,  date = pd,  feel = if (temperature > 26) then "hot" else if (temperature > 18) then "cool" else "cold" endif;  };  }  } |

从上表中，我们可以看到在最开始设置了Weather模型的key值“key Weather {id};”，用于对id重复的冗余数据进行删除，最后操作的结果即不包含冗余内容。

### Join操作的QVT描述

Join函数操作是对两个数据集中的键值进行连接操作，形成一个新的数据集。在QVT-Relation中，一个Relation中允许有多个checkonly的Domain，即可以有多个输入，当对多个输入Domain中的键值key用相同的变量进行绑定之后，QVT-Relation就会选出该键值的所有模型，然后将多个输入Domain中的属性放进输出Domain，即可完成Join连接操作。下面用一个简单例子来展示Join函数操作的模型描述，如表4-4。

表4-4 Join操作模型层描述实例

|  |
| --- |
| top relation CustomerJoinOrder  {  mID: String;  mName, mItemList: String;  mSalary, mCost: Real;  mYear: Integer;    checkonly domain source customer: Customer  {  ID = mID,  name = 'Amy',  area = 'Beijing',  salary = mSalary  };  checkonly domain source order: Order  {  ID = mID,  year = mYear,  itemList = mItemList,  status = true,  cost = mCost  };  enforce domain target customer\_order: Customer\_Order  {  ID = mID,  salary = mSalary,  year = mYear,  itemList = mItemList,  cost = mCost  };  } |

上表的QVT描述相对复杂，这里作详细解析。上面的一段QVT代码描述了这样的一个场景，将用户数据与订单数据相连接，形成一个完整的用户订单数据。mID变量起到了连接键值的作用，使得具有同一个mID的用户数据和订单数据合在一起并输出到用户订单数据中。

### 集合操作的QVT描述

集合操作主要包含三种基本操作：交集操作，并集操作和差集操作，对多个集合进行交集操作后，会得到包含多个集合共同元素的数据集；对多个集合进行并集操作后，会得到多个集合所有元素的数据集；对一个集合进行差集操作后，会得到只包含在这个集合且不包含其他集合元素的数据集。我们在QVT-Relation中进行描述时，Relation中允许有多个checkonly的Domain作为操作的源集合，这几个Domain中的属性名应该都相同，然后在进行变量绑定之后，会在where语句中添加一些约束，用于区分这三种集合操作。下面用一个QVT-Relation描述实例来展示交集操作的模型描述，其他两种集合操作的描述类似，如表4-5.

表4-5 交集操作模型层描述实例

|  |
| --- |
| top relation CustomerIntersection  {  mID: String;  mName, mArea: String;  mSalary: Real;    checkonly domain source customer1: Customer1  {  ID = mID,  name = mName,  area = mArea,  salary = mSalary  };  checkonly domain source customer2: Customer2  {  ID = mID,  name = mName,  area = mArea,  salary = mSalary  };  enforce domain target customer: Customer  {  ID = mID,  name = mName,  area = mArea,  salary = mSalary  };  } |

上表的QVT描述相对复杂，这里作详细解析。上面的一段QVT代码描述了这样的一个交集的转换场景，将两份用户数据求交集，形成一个完整的用户数据。当两个输入集合具有相同的属性值时，才会将结果进行输出，从而得到求交集之后的数据。

## 本章小结

本章首先说明了本文选取QVT-Relation作为本工作模型层描述语言的原因，然后用列举实例的方法详细的介绍了我们对于几种转换操作的模型层规约设计，并通过成熟的QVT-Relation执行工具mediniQVT对我们的设计的模型层规约进行验证，从而完成模型层对几种数据转换操作的描述设计。

# 代码生成引擎的实现

## 数据模型的代码生成

### 数据建模

数据转换的重点除了转换操作之外，数据本身也是我们研究的对象，在本工作中一个重要的思想就是将数据看作模型，将数据转换映射到模型转换上，所以将数据抽象成模型也是必要的工作。

对数据进行建模的方法有很多，本文采用Ecore元模型对数据进行建模，一个原因是Ecore元模型是Eclipse建模框架EMF（Eclipse Model Framework）的核心，我们可以通过Ecore元模型实现UML模型，XML模型和Java代码之间的相互转换，使用广泛，具有一定说服力；另一个原因是QVT-Relation是支持Ecore元模型的，可以便于开发人员使用QVT工具进行测试。

因此，在本工作中，我们将数据看作一种模型的实例，将数据的结构用Ecore模型的结构描述出来，这样就完成了从数据到模型的构建过程，实现了数据层到模型层的抽象。同时，Eclipse已经提供了一套非常成熟完整的EMF建模工具，我们可以像构建Class类图一样对我们构建的Ecore元模型增加修改，EMF建模工具就会自动地创建Ecore元模型文件，图5-1展示了一个Ecore元模型的实例。

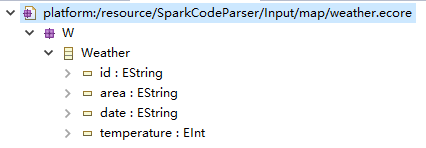


图5-1 Ecore建模实例

### Ecore解析

在创建好Ecore文件之后，我们发现Ecore文件的格式实际上是符合XML文件格式标准的，所以我们可以借助XML的解析工具对Ecore文件进行解析。XML的解析技术已经非常成熟，这里使用应用在Java平台，只需要简单配置即可使用的Dom4j工具进行解析。

Dom4j是一个简单好用的Java开源库，用于解析XML文件，该工具采用了Java集合框架并完全支持DOM，SAX和JAXP，具有一些简单的编程接口，使用起来非常灵活。下面是本工作使用到的几个常用Dom4j接口，在这里进行介绍：

（1） 通过SAXReader读取Ecore文件：

SAXReader saxReader = new SAXReader();

Document document = saxReader.read(new File(fileName));

（2） 获取Ecore模型的根节点：

Element rootElement = document.getRootElement();

（3） 获取某一节点下的所有子节点：

List<Element> elementList = rootElement.elements("node");

（4） 获取节点的某一属性值：

Attribute attribute=element.attribute(“attribute”);

通过以上几种编程接口的使用，即可提取出我们需要的元模型信息，为后续生成对应的Spark代码做准备。

### 生成数据类代码

在数据类中，我们除了要记录下该数据类所包含的属性，同时要具有文件读取的功能，从而方便代码的生成。

首先，在读取到Ecore根节点时，表示开始进行数据类的代码生成，并在这里生成数据类文件中的import语句。

接着，利用“elements”编程接口获取所有eClassifiers的节点，每一个eClassifier中的name属性就是数据模型的类名，由此可以生成一个Scala Object。

然后，eClassifier中的所有eStructuralFeatures就是该类所包含的属性，在eStructuralFeatures元素中，name表示类属性的名字，type表示类属性的类型。

最后，将这些信息解析提取出来之后，如果该元模型是Source元模型，则添加读取文件的代码，读取的内容按照上面获得的属性信息进行存储；如果该元模型是Target元模型，则添加写入文件的代码，按上面获得的属性信息拿到对象的属性值输出到文件。代码生成对应关系如表5-1。

表5-1 代码生成对应关系

|  |  |
| --- | --- |
| Ecore元模型 | Spark代码 |
| 根节点Epackage | 创建数据类文件，生成配置信息代码 |
| eClassifier | 生成类名 |
| eStructuralFeature | 获得属性名和类型，并放入数据类的一个HashMap中记录，  方便后面的对某一属性变量操作的代码生成 |
| eType | 对变量类型进行一个简单处理：  EInteger –> Int EDouble –> Double  EString -> String EBoolean->Boolean |
| Source Model | 包含sc.textFile()读取文件Spark代码 |
| Target Model | 包含result.saveAsTextFile()保存文件Spark代码 |

## QVT描述的代码生成

### QVT语法解析

虽然QVT-Relation描述的转换语义简单易懂，但他仍然是一种抽象化的语言，因此我们想要获得QVT描述中的操作信息就必须先对我们的QVT-Relation描述进行语法解析。

JavaCC[19]是一个很不错的词法、语法解析器的生成器，只需要编写规则就可以生成Java语言的词法、语法解析代码，JavaCC和我们之前编译原理课上使用的工具类似，都是利用BNF范式进行编写语法的，所以在开发时有一定经验，比较容易上手，但还是有一定区别的，JavaCC采用的是自顶向下模式进行语法解析，而YACC等工具则是采用自底向上模式。两种方法各有优劣，在我看来JavaCC相比其他工具更加易于调试，在对语法分析树进行遍历时，上下传值也更加方便。

最重要的是JavaCC有专门支持Eclipse的插件，更方便我们进行开发，因此本文选择使用JavaCC作为本工具的语法解析工具。

另一方面由于QVT-Relation对模型转换的表达能力非常丰富，可以支持多种模型转换的场景，但本工作中主要关注的是数据模型及其转换，所以其中的很多语义在数据转换中并不会使用，所以，我们在进行设计语法时进行了适当的裁剪，裁剪后的QVT-Relation语法规则如图5-2，而图5-3则展示了利用JavaCC工具实现该抽象语法树的代码片段。

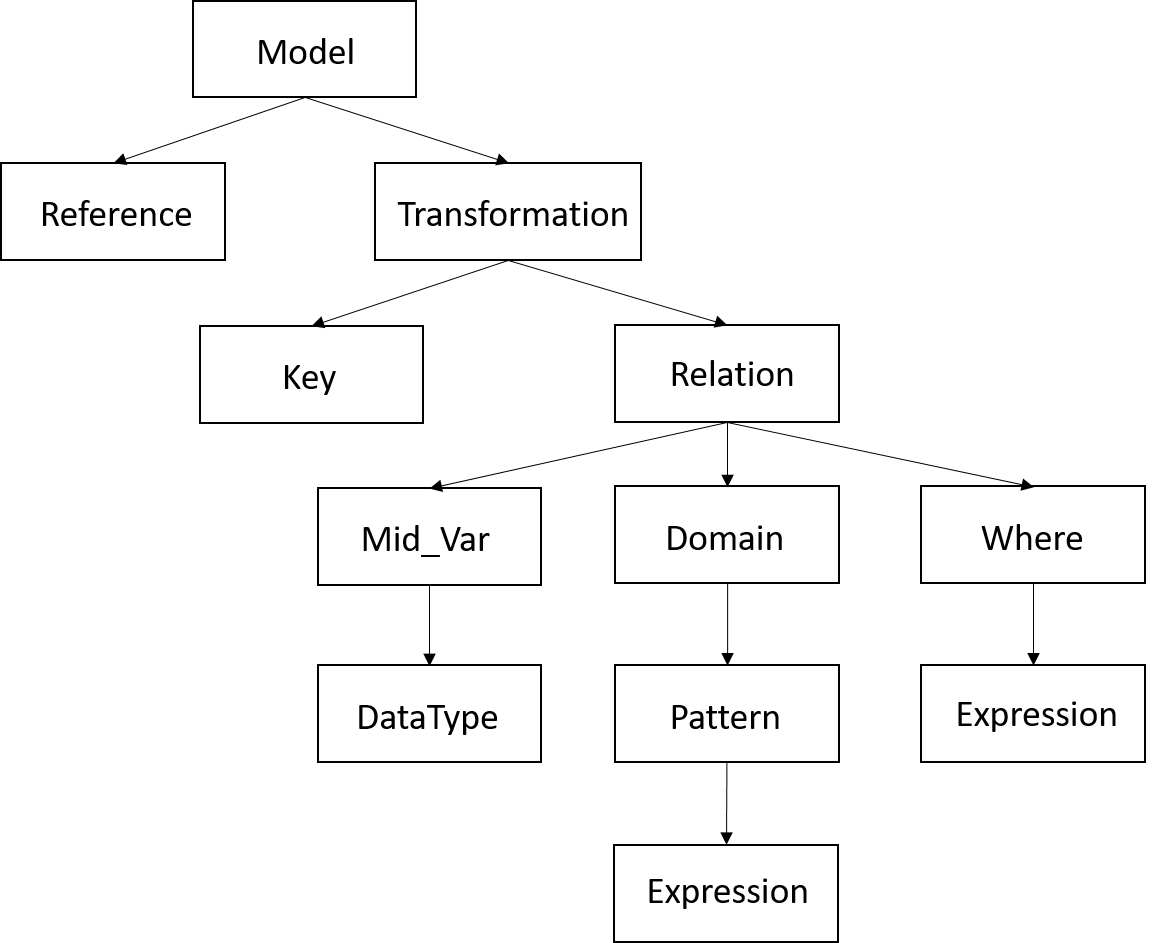


图5-2 抽象语法树

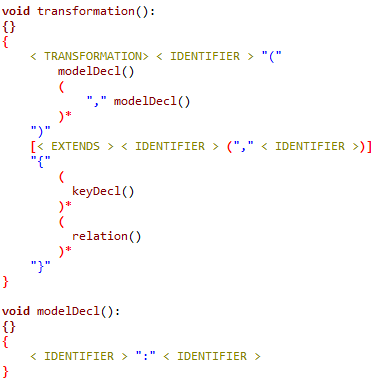


图5-3 QVT-Relation语法规则片段

### 转换代码生成算法

在使用JavaCC工具对QVT-Relation描述进行解析之后，我们得到了图5-3这样的一个抽象语法树结构，然后我们可以对这个语法树从根节点开始，依次遍历所有的语法节点，在每一个节点会进行该节点特定的代码生成或信息处理记录等操作，然后会根据上一章节中设定的转换规则生成特定的Spark Api代码，下面就是我们代码生成算法的伪代码，如算法1所示。

|  |
| --- |
| **算法1 代码生成算法** |
| **GenerateSparkCode（QVT-Tree,SparkCode）**  **Input:**  解析得到的QVT-Relation抽象语法树  **Output：**  转换操作的Spark代码  1：**if** 遍历到Transformation节点 **then**  2： 生成Spark文件的配置代码  3： **for** all key in key\_node\_list **do**  4： 根据key语句内容生成相应的Spark distinct()语句  5： **end** **for**  6： **for** all top\_relation in top\_relation\_list **do**  7： 设置mid\_var列表用于记录domain中的变量绑定  8： 设置domain\_num记录source domain的数量  9： **for** all source\_domain in checkonly\_domain\_list **do**  10： domain\_num = domain\_num + 1  11： **for** all attr in attr\_list **do**  12： **if** attr 绑定到一个常量 **then**  13： 生成Spark filter()语句  14： **else**  15： 记录到mid\_var列表中  16： **endif**  17： **end** **for**  18： **if** source\_domain中属性完全一样 **then**  19： 集合操作Sign=true  20： **else**  21： join操作Sign=true  22： **endif**  23： **end** **for**  24： **if** 含有where节点 **do**  25： 根据集合操作要求，生成对应集合操作Spark代码  26： 根据mid\_var记录的信息查找变量，根据变量所进行的操作生成并记录对应的表达式  27： **endif**  28： **if** join操作 = true **do**  29： 生成Spark join()语句  30： **endif**  31： **for** all target\_domain in enforce\_domain\_list **do**  32： 根据mid\_var记录的信息查找变量，根据变量所进行的操作生成并记录对应的表达式  33： **end for**  34： 根据最终记录的所有表达式列表和mid\_var列表，生成Spark Map()语句  35：**endif** |

在算法1中，我们对工具的整体实现算法进行了介绍，但其中Spark转换操作编程接口的实现细节并没有体现出来，但这一部分的实现仍然具有一定的复杂性，因此，我们在下面对这几种具体的Spark转换操作的代码生成方法进行详细的讲解。

Spark转换操作的代码生成的难点在于Spark的每一个转换操作Api都需要填写输入的参数，例如，在Filter()过滤操作中，需要添加类似“\_ > 5”这样过滤条件的输入参数。因此，我们在实现时设计了一套中间变量用于记录Spark代码中所需要的信息，如表5-2。

表5-2 重要中间变量

|  |  |  |
| --- | --- | --- |
| 中间变量名 | 类型 | 作用 |
| inputElement | ArrayList<Map<String,String>> | 记录源模型中的属性及其绑定的临时变量，用于map操作的生成 |
| filterElement | ArrayList<Map<String,String>> | 记录需要过滤的属性名和它的过滤条件，用于过滤语句的生成 |
| outputElement | ArrayList<Map<String,String>> | 记录目标模型中的属性及其绑定的临时变量 |
| whereElement | ArrayList<Map<String,String>> | 记录where语句中修改的变量和赋值该变量进行的操作 |
| distinctElement | ArrayList<String> | 记录key语句中的元模型名 |
| declareVariable | ArrayList<String> | 记录用于绑定属性的临时变量名 |

通过使用以上中间变量，我们即可完成具体操作的代码生成，其详细的生成规则如下：

当生成filter()语句时，将filterElement中的元素逐一取出来，并生成对应的过滤语句；

当生成distinct()语句时，在distinctElement中检查是否包含当前读取源模型的名字，如果有，则直接生成distinct()语句；

当生成join()语句时，只需要传入两个进行操作的源模型的名字，即可按照固定的规则生成join()语句；

当生成集合操作语句时，与join操作类似，只需要传入两个进行操作的源模型的名字，即可按照固定的规则生成对应的集合操作语句；

当生成map()语句时，需要进行的操作比较复杂，首先处理whereElement中进行的操作，然后将中间变量都替换为源模型中的属性名，最后按照目标模型的属性顺序输出完成映射操作。

## 本章小结

本章对代码生成引擎的具体实现进行了详细的介绍，该代码生成引擎主要分为Ecore元模型生成Spark代码和QVT描述生成Spark代码两个过程。在Ecore元模型的代码生成中，首先介绍了Ecore元模型的结构，然后给出了模型层与Spark代码之间的对应关系，最后详细介绍了我们实现的翻译算法生成数据类Spark代码；在QVT描述的代码生成中，首先给出了我们实现的QVT-Relation语法解析器，获得一个抽象语法树，然后对树结构遍历并设计出代码生成的算法，最终获得Spark的转换操作代码，完成代码生成引擎的实现。

# 实例研究

本章将设计了一个实例场景来进行详细的展示，如何使用本文实现的Spark代码生成引擎进行基于Spark平台的数据转换方法。

## 实例研究

本实例的场景图如图6-1所示，该实例的背景是某个互联网购物平台拥有近几年的所有注册用户数据和所有的商品购买订单数据，我们希望从这些数据中抽取出北京地区的用户在2014年的所有订单数据，基于这些信息我们后续可以进行一些数据挖掘操作，提取出的挖掘信息就可以有很多的应用场景，例如用于对北京地区的用户进行商品推荐或广告投放等。

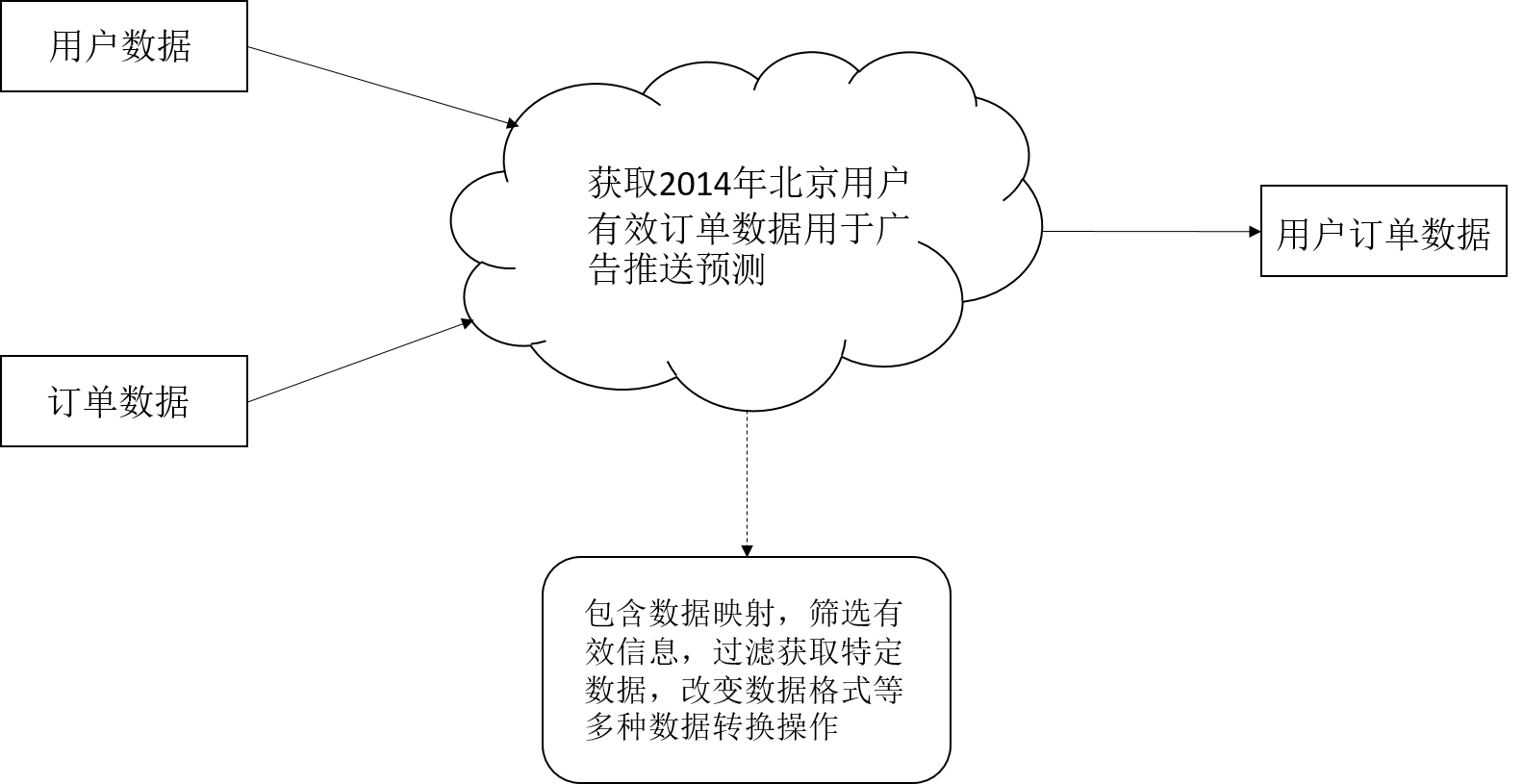


图6-1 用户订单实例

在介绍了研究的实例场景之后，我们对实例中涉及到的一系列转换操作进行了一个分析提取，为了便于理解制作了一个数据转换流程图，如图6-2.

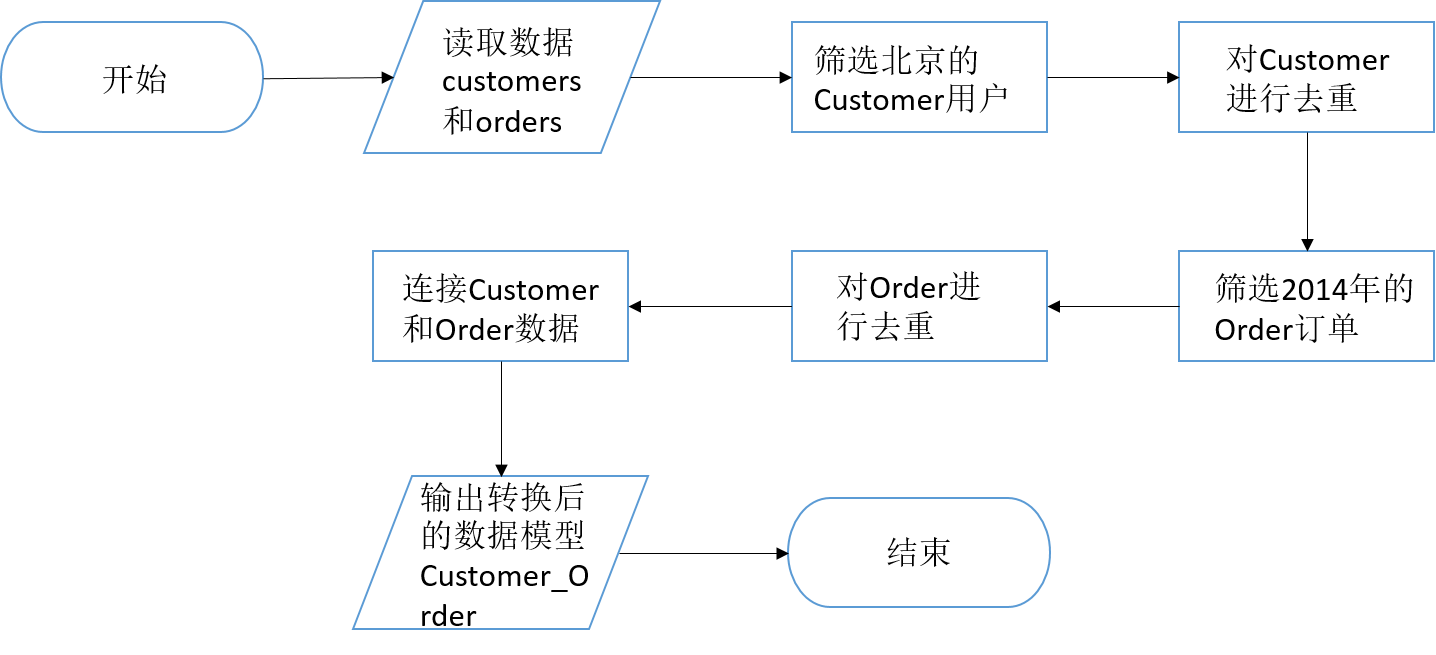


图6-2 数据转换流程图

基于这样的数据流程，我们在用本文的转换引擎进行实例研究时，首先要对实例的输入数据和输出数据进行建模，我们使用Ecore工具可以根据数据结构轻松的设计出数据的元模型，如图6-3所示，从左至右分别为Customer数据模型，Order数据模型和最终输出的数据模型，构建成功后就可以得到Ecore模型文件。

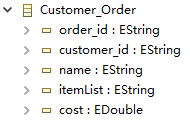
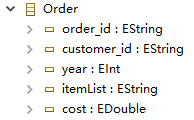
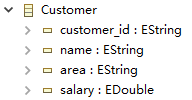


图6-3 数据Ecore模型

从上图中可以清晰的看到数据的结构，其中Customer模型包含四个属性分别为用户Id，用户名，所在地区和工资；Order模型包含5个属性分别为订单Id，该订单所属的用户Id，订单年份，订单购买列表，以及订单金额；输出的Customer\_Order模型既包含了Customer模型中的属性，又包含了Order模型中的属性，分别为订单Id，该订单所属的用户Id，用户名，订单购买列表，以及订单金额。在设计好数据模型之后，就可以开始进行转换操作的QVT-Relation的模型层设计了。

我们在第四章中已经详细介绍了各种转换操作的QVT-Relation描述设计，每种转换操作的QVT-Relation描述都符合固定的描述规则，并且这几种转换规则的描述之间并不冲突，也就是说我们在对实例进行描述时，按照设计好的描述规则对涉及到的转换操作逐一进行描述，最后就可以得到具有多种转换操作功能的QVT-Relation模型层转换规约。

图6-3中显示了本实例中包含的转换操作，根据其转换流程，我们在用QVT-Relation进行转换描述时依次添加描述的规则，首先，输入为两个数据集，则在relation中添加两个checkonly domain作为模型层的输入；其次，根据去重操作，则在transformation下添加key语句；然后，根据过滤操作，在其对应的domain中添加约束语句，如“area = 'Beijing'”；最后，补充checkonly domain中的变量绑定，并添加模型层的输出enforce domain和其中变量的赋值，最后完成QVT-Relation的转换描述，如表6-1所示。

表6-1 实例模型层转换描述

|  |
| --- |
| transformation CustomerJoinOrder(source: Source, target: Customer\_orders)  {  key Customer {customer\_id};  key Order {order\_id};  top relation CustomerJoinOrder  {  mID: String;  mCusID: String;  mName: String;  mItemList: String;  mCost: Real;    checkonly domain source1 customer: Customer  {  customer\_id = mCusID,  name = mName,  area = 'Beijing'  };  checkonly domain source2 order: Order  {  order\_id = mID,  customer\_id = mCusID,  year = 2014,  item\_list = mItemList,  cost = mCost  };  enforce domain target customer\_order: Customer\_Order  {  order\_id = mID,  customer\_id = mCusID,  name = mName,  item\_list = mItemList,  cost = mCost  };  }  } |

在模型层完成数据转换的描述设计之后，我们就得到了转换引擎所需要的所有模型信息，然后我们将输入数据Ecore模型，输出数据Ecore模型和QVT-Relation数据转换描述这三个文件输入到我们设计的转换引擎当中，经过转换引擎的分析处理就得到我们生成的Spark代码，其转换过程如图3-1.

虽然我们得到了生成的Spark代码，我们还需要对实验结果进行验证，用生成的代码完成大数据的转换操作实验从而说明本数据转换方法的正确性和可行性。由于我们很难获得真实的用户订单数据，所以我们手写了一份数据生成的python程序，根据实际情况模拟生成了实验数据，生成的四组实验数据的数据量分别为100M，1G，10G，100G，以下实验也均基于这四份数据进行测试。

为了验证生成代码的正确性和可用性，我们人工手写了一份具有相同转换功能的Spark代码用作对比实验，这份人工的Spark程序与自动生成代码具有相同的逻辑，但具体实现方法有一定区别。因此，我们将这两个程序分别对实验数据进行转换，分析实验结果。

首先，分别运行结束之后，首先检查了生成代码转换的正确性，由于实验数据过大，我们抽取了部分生成结果，生成代码与手写代码的运行结果相同，从而说明实验生成的代码可以保证转换的其正确性,实验结果如图6-4。

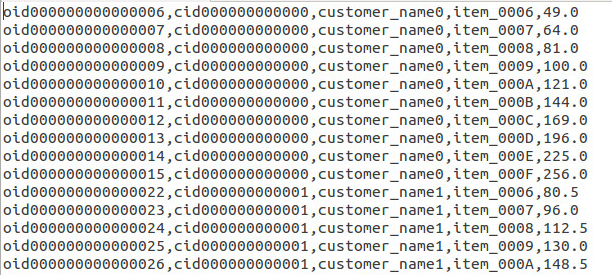


图6-4 实验结果

然后，我们对两份代码的运行时间进行了统计和对比，时间结果如图6-5，从图中可以看出，生成代码的运行效率略低于手写代码运行效率30%，并没有产生明显的差异，对于处理100GB级大数据文件所花费的开销也是可以接受的。

图6-5 运行时间对比结果

最后，我们 对生成代码和手写代码两种方法所需要编写的代码量进行对比，自动生成代码只需要编写少量的QVT-Relation描述语句，然后就会生成数据转换的Spark代码，而手写代码的代码行数则要远高于模型层描述，如图6-6。

图6-6 代码行数对比结果

通过以上三组对比实验的结果，说明了本文转换引擎生成的代码可以正确的完成数据转换功能，并且具有可以接受的执行效率。同时，模型层对于数据转换的QVT-Relation描述代码要远少于手写的Spark代码，从模型转换的角度解决了数据转换问题，简化了大规模数据转换的复杂性。

## 本章小结

本章利用一个数据转换实例详细介绍了本文提出的数据转换方法中数据建模，数据转换的模型设计以及Spark代码的生成等各个步骤的实现，并通过一系列对模拟数据转换的对比实验，验证了本方法的可实施性。

# 总结

本章首先对本文完成的工作进行总结，分析了这种数据转换操作方法的可行性。同时，也说明了本工作目前存在的一些不足，并根据这些不足给出了下一步的改进计划。

## 工作总结

本文基于Spark平台的基本数据转换操作，利用模型转换的思想，在模型层利用QVT-Relation设计了数据转换的描述，然后介绍了代码生成引擎的具体实现细节，通过实现的代码生成引擎，将模型层的转换描述和数据模型翻译生成Spark平台运行的代码，从而完成这套数据转换方法。最后，利用一个实例验证了本方法的正确性，并通过一系列对比实验说了本方法相比传统大数据转换操作开发更加简单，屏蔽了Spark大数据平台的使用复杂性，而且保持了可以接受的执行效率。

## 不足与展望

本文利用模型转换的思想，将模型层的转换操作描述生成转换操作的Spark代码，在生成过程中，引用了HashMap结构从而便于数据操作Spark代码的生成，虽然这种方法简化了代码生成的实现，但同时也降低了生成代码的运行效率。下一步我们可以改善代码的生成方法，优化生成代码的运行效率，进一步提高本文数据转换方法的可用性。

另一方面，本文目前实现的数据转换方法所支持的转换操作均为常用的基本操作，当处理更加复杂的数据转换场景时，暂时还无法支持复杂的转换操作，下一步我们会增加更多转换操作的模型层描述设计和代码生成，从而支持更复杂的数据转换场景。

参考文献

[1] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters[C]// Conference on Symposium on Opearting Systems Design & Implementation. DBLP, 2004:137-150.

[2] Mens, Tom, and P. V. Gorp. "A Taxonomy of Model Transformation." Electronic Notes in Theoretical Computer Science 152.1-2(2006):125-142.

[3] OMG. Meta Object Facility （MOF）2.0 Query/View/Transformation Speci fication. http://www.omg.org/cgi-bin/doc?formal/2016-06-03.

[4] Wikipedia. QVT(Query/View/Transformation). https://en.wikipedia.o rg/wiki/QVT.

[5] OMG. MetaObject Facility. http://www.omg.org/mof/.

[6] OMG. Object Constraint Language. http://www.omg.org/spec/OCL/2.4

[7] Eclipse. Eclipse Modeling Framework. http://www.eclipse.org/model ing/emf/docs/.

[8] Budinsky F, Brodsky S A, Merks E. EMF Eclipse Modeling Framework[M]// Eclipse modeling framework :. Addison-Wesley, 2003:206-207.

[9] Apache. Spark Document Network. http://spark.apache.org/docs/lates t.

[10] Karau H, Konwinski A, Wendell P, et al. Learning spark: lightning-fast big data analysis[M]. " O'Reilly Media, Inc.", 2015.

[11] Apache. Hive Network. http://hive.apache.org/.

[12] PasaLab. The Octopus Project. http://pasa-bigdata.nju.edu.cn/oct opus/.

[13] Apache. The Beam Project. https://beam.apache.org/

[14] Benelallam A, Gómez A, Tisi M, et al. Distributed model-to-model transformation with ATL on MapReduce[C]//Proceedings of the 2015 ACM SIGPLAN International Conference on Software Language Engineering. ACM, 2015: 37-48.

[15] Tisi M, Martínez S, Choura H. Parallel Execution of ATL Transformation Rules[M]// Model-Driven Engineering Languages and Systems. Springer Berlin Heidelberg, 2013:656-672.

[16] Loli Burgueño, Troya J, Wimmer M, et al. On the concurrent execution of model transformations with Linda[C]// The Workshop on Scalability in Model Driven Engineering. 2013:1-10.

[17] Greenyer J, Kindler E. Reconciling TGGs with QVT[M]// Model Driven Engineering Languages and Systems. Springer Berlin Heidelberg, 2007:16-30.

[18] The Dom4j Project. http://www.dom4j.org/dom4j-1.6.1/.

[19] The JavaCC Project. https://javacc.org/.

致谢

经过两个学期的努力，我终于完成了我的毕业设计和论文书写，为我的大学生活画上一个圆满的句号。

在此，首先我要特别感谢我的毕设指导老师——张天副教授。张天老师从毕业设计的选题，到毕业设计的开发，再到最后毕业论文的撰写各个阶段都以严格的标准要求我，促使我完成一篇更加优秀的毕业论文，同时也要感谢计算机科学与技术院系的各位领导和老师对我的培养，让我从对计算机毫无了解，成长为具有独立开发能力的程序员。

其次，我要感谢软件工程实验组的车开达和刘拨杰两位学长。两位学长在Spark集群架构环境的建立、参考论文以及毕业设计过程中QVT-Relation转换规则的设计等多个方面都给予了我很大帮助，在项目的开发上，也与我分享了宝贵的经验，令我受益匪浅。

最后我要感谢本篇论文的参考文献的各位专家和学者。他们的研究工作为本文工作提供了宝贵的建议和参考，帮助我解决了很多问题。