基于软件总线模型的数据清洗系统的研究与实现

重庆大学硕士学位论文

学生姓名：赵　鹏

指导教师：杨　丹　　教　授

专　　业：计算机软件与理论

(软件工程领域)

学科门类：工　学

重庆大学软件工程学院

二 OO 九年十月

Research and Implementation of the Data Clean System Based on Software Bus Model

A Thesis Submitted to Chongqing University

in Partial Fulfillment of the Requirement for the

Degree of Master of Engineering

By

Zhao Peng

Supervised by Prof. Yang Dan

Major: Computer Software & Theory

(Software Engineering)

School of Software Engineering of

Chongqing University, Chongqing, China

October, 2009

摘　要

随着信息技术的飞速发展，获取数据手段也更加多样化，人们现在所拥有的数据资源日益丰富，数据量急剧增加。但是海量的数据并不一定真正具有价值，数据的价值在于它的质量，基于劣质数据的决策是不可信的。基于数据所做的进一步挖掘和决策与数据质量直接相关。但是面对数量巨大而零乱的数据人工处理是非常困难的，数据质量问题成为制约数据应用和数据挖掘的“瓶颈”之一。纠正数据错误是避免错误决策、降低决策风险，改善数据挖掘效果的重要环节，数据清洗就是用来完成这项艰巨任务的。

本文在现有的数据清洗方法基础上，从满足现代软件工程理念的角度出发，尝试性地设计并实现了一种基于软件总线模型的数据清洗系统，将具体的数据清洗方法以组件的形式集成到系统中来，并能够适用于对多种异构数据源的数据清洗。系统具有很好的可复用性和可扩展性。

本文完成的主要工作内容如下：

① 分析了本文的相关研究背景，数据清洗的相关知识以及现阶段国内外数据清洗研究现状，介绍国内外现有的数据清洗方法和数据清洗产品。

② 分析了 XML 用于系统集成和数据载体的优势，比较 XML 有效性验证方式以及文档解析方法；介绍了数据库访问 ADO.NET 技术，分析 ADO.NET 的结构和不同数据读取方式的特点；介绍中文信息处理中的中文分词技术，分析比较三种主要中文分词方法的效果。

③ 研究了在系统中涉及到的关键技术。首先，分析了软件总线模型的结构特点，研究了软件总线中组件管理方式和总线控制方法。其次，研究了对数据清洗任务分解的方法，分析了多线程并发技术，并将其用于对数据清洗任务并发执行，通过实验验证其效果；最后，研究了中文地址数据清洗方法，结合中文地址的特点，提出一种基于分级树的中文地址规范化方法，并通过实验对其效果进行比较。

④ 设计并实现了基于软件总线模型的数据清洗系统。首先对系统整体描述，其次对于系统整个结构进行分析设计，对系统按照功能结构分为数据清洗总线，数据源包装器，数据清洗组件和组件适配器等模块，然后分别对每个模块进行设计和实现，最后对系统运行结果展示。

关键词：数据清洗，总线模型，组件，异构数据源，适配器

ABSTRACT

With the promotion of information technology development，the methods to obtain data have been becoming more and more diversity. The data resource which people have owned is more abundance increasingly，the amount of data is increasing rapidly。But the magnanimity data doesn't stand for real value, the value of data is from its quality. The decision that depends on low quality data is not credible. The data mining and decision which depend on data is related with the quality of data. But it is very difficulty to process such abundance and disorder data artificially. The quality of data is becoming the bottleneck for the data application and data mining. That data errors correcting is an important part to avoid wrong decisions, reduce the risk of decision-making and improve effect of data mining. Data cleaning is to complete this difficult task.

The paper, based the data clean methods, from the perspective of the modern software engineering, try to design and implement an data cleaning system based on software bus model. In this system, the concrete data cleaning method is designed as component. The modules in the system are high degree of loose coupling. The flexibility of data cleaning system has been improved. The system can integrate many kinds of data clean method, and it can be use to clean many kinds of heterogeneous data sources.

This paper has accomplished the following tasks:

① Analysis the related research background, the knowledge of data cleaning and the current status of the data cleaning research, introduce the data cleaning method and data clean product in

② Analysis the features of XML, the method for validation and document parser; Introduce ADO.NET technology, analysis the structure of ADO.NET and the features of different data access in ADO.NET; Introduce the Chinese word segmentation techniques, analysis and compare the effect between three Chinese word segmentation methods.

③ Research the key technologies used in the system . At first, this paper analysis the structure of the software bus model, research the main parts in the software bus model, research the method for integrating component in system and the method for control bus. Then research the methods for decomposing the data clean task, analysis the concurrency technology using multithread to complete the data clean task，then

show the effect through the experiments. At last research the Chinese address data cleaning methods. Based the feature of Chinese address, prose a Chinese address standard method based classification tree.

④ Design and implement the data clean system based on software bus model. Analysis the structure of the whole system, according to the functions, the modules can be composed by data clean bus, data source wrapper, data clean component and adapter. Then design and implement every module. At last, show the screenshot of the running system.

Keywords: Data Cleaning, Bus Model, Component, Heterogeneous Data Source, Adapter

目 录

1 绪 论

1.1 研究背景

随着计算机信息技术的飞速发展，信息系统在越来越多的行业领域中得以应用，数据量也随之急剧增加。数据是信息的载体，但并不是数据量越大，其价值越高。对于各种数据分析、决策支持，数据挖掘等应用来讲，高质量的数据才有真正的意义[1]。面对如此海量的数据，人们却经常抱怨所谓的“数据丰富，信息贫乏”，其中一方面是因为缺乏有效的数据分析技术，另一个重要的方面就是因为数据质量不高，海量的数据不能得到有效利用，提供数据真正的价值。因此，从这些角度考虑，企业的数据质量问题正在越来越多的得到关注。其实，对数据质量管理就如同是对产品质量管理一样，其管理方案应该是贯穿于数据生命周期的每个阶段，但目前尚缺乏一套标准化的解决思路。

目前，针对数据质量的研究很广泛，已经涉及到人工智能、统计学、数据仓库等多个领域。数据质量产生的原因，如数据的来源不同，数据录入错误、数据的表示方式不同、数据的不一致性等，这些原因最终导致数据质量不高，数据质量存在严重的问题[2]。在软件领域中，有种常见的说法，即“进去的是垃圾,出来的也是垃圾(garbage in, garbage out)”的原理。因此，现有的数据分析，决策支持，数据挖掘等基于数据所做的应用，如果是在这些低质量的，存在问题的数据的基础上进行，其效果可想而知，很可能会扭曲从数据中能够获取到的真正的信息，从而对企业造成严重的损失。由此可见，数据质量问题的重要性不言而喻。

数据清洗所要完成的任务就是将那些存在质量问题的数据进行清洗，以提高数据质量，满足各种基于数据所做的进一步应用的要求。

1.2 数据清洗相关知识

1.2.1 数据清洗

在英文中，对于数据清洗的解释有三种， Data Cleaning,Data Cleansing 和 Data Scrubbing 三种说法[3]，在中文中有数据净化，数据清洗，数据擦洗三种译文，目前比较常用的前两种描述。数据清洗的研究现在尚属于一个新的领域，并且在不同的应用领域当中，对数据清洗具体要求不同，因此目前对数据清洗还没有统一的定义和描述，但是根据其常用方法和所要实现功能，从在本质上来讲是相同的，因此，在各种定义或描述之间也有着交叉点和很高的一致性：

① 在《数据质量和数据清洗研究综述》一文中将数据清洗被定义为:发现和清除数据中的错误和不一致来提高数据的质量[4]。数据清洗通常不是单独进行的，大

多数情况下需要和数据转换集成共同进行，主要用于异构数据源的集成，系统的升级后数据的迁移等。

② 在《可扩展数据清理软件平台的研究》文章中，将数据清洗描述为是从数据源中清除错误数值和重复记录等。即利用有关技术如数理统计、数据挖掘或预定义清理规则等，从数据源中检测和消除错误数据、不一致数据和重复数据，从而提高数据的质量[5]。

③ 在《基于规则引擎的数据清洗》中，数据清洗可被描述为从大量原始数据中使用一系列的逻辑检测出脏数据并修复或丢弃之。数据清洗大多作为 ETL 工具的一个功能来实现[6]。当然，也存在大量数据清洗专用工具，它们被称为数据质量工具。

④ 国内某专业数据服务公司将数据清洗理解为指将数据中的错误和冗余数据检测出来并且清理掉，从而提高数据质量。它包括：有效性检验、标准化、最大值验证等过程。

结合以上这些内容，本文中将数据清洗概括性理解为：数据清洗是通过各种技术，对来自单个数据源或者多个数据源的中“脏数据”（例如错误的、重复的、冗余的或者不一致的数据等）进行检测和消除的过程。它的目标是提高数据质量，包括数据的正确性、准确性、一致性以及可用性。

1.2.2 脏数据

数据清洗的功能就是对脏数据进行检测和消除，需要明确对脏数据的定义。所谓脏数据是指那些错误的，冗余的，过期的，表达不一致或者重复的数据信息。对于脏数据的分类有多种，在本文中，结合《数据质量和数据清洗研究综述》一文中所讲，将脏数据分为以下四种类型[4]。

① 错误的数据：这种情况主要是指数据中存在各种不合法的情况，比如数据类型错误、数据违反业务规则、数据范围越界、格式不符合要求。比如年龄是负值，超出数据有效范围。

② 不一致的数据:这种情况在集成不同系统时比较常见，由于系统功能模块或者不同系统之间的编码形式不一致、记录不一致、引用不一致等原因所造成的数据的不一致性。另外在同一系统内部，由于输入出错等原因也可能出现数据不一致性的现象，比如：邮编与电话区号所表示地址不一致，身份证号码与生日不一致等。

③ 不完整的数据。这种情况比较多，例如字段信息的缺失，记录的缺失、记录不完整等。

④ 重复的数据。在集成不同系统时，有些数据可能已经在别的系统中以不同的形式存在，这样再集成时就可能导致数据的重复。

在实际应用中，导致脏数据产生的原因有很多，下面对于其主要原因概括为以下几点：

① 人为因素。系统使用人员在无意情况或者某些客观原因的情况下，输入了符合要求的数据。比如客户在填写某些信息时由于误操作，输入错误的信息；或者对于一些系统，客户在不确定如何填写情况下输入的信息。

② 多数据源问题。不同数据来源很可能对相同的数据有着不同的表示方式。例如不同的存储格式(比如对时间的表示形式不同)，不同的精度(比如对于身高、体重的表示)，不同标记值(比如对性别、结婚状况、文化程度的标示不同)等。正是存在这样的因素，对于同一客户来说，其信息在不同系统中就可能有不同的数据描述。

③ 系统的输入设计不够严密。例如对于有些系统，其界面设计的输入不够严密，对于一些信息被设计成“输入”的方式添加，而不是通过“选择”方式添加,这样用户的输入对于系统来讲就有很大不确定性。对于客户联系地址，很多系统都是设计为让用户直接输入的形式，这样所输入的数据就会有很多种格式，这就可能造成地址数据不完整等问题。

④ 其他外界因素。现实中的身份证号增位所造成的影响就是一个很好的例子。在某些系统中，将客户身份证号作为客户的唯一标识，但客户的身份证从 15 位更换为 18 位之后。当系统使用人员再输入增位后的身份证号时，系统只是通过原有的 15 位的身份证进行判断，就会将本次输入的客户信息当做是新的客户，从而造成数据的重复。

1.3 国内外研究现状

1.3.1 国外数据清洗研究现状

国外对数据清洗技术的研究，最早的应用是从对全美的社会保险号错误的纠正开始。后来美国商业的发展和信息技术的进步，更加刺激了对数据清洗技术的研究。目前其研究内容主要涉及以下方面:

① 检测数据集中异常数据，对其可以描述为是对数据集中记录的属性的清洗，即对“脏数据”中的错误、异常数据的清洗。目前主要有以下这些方法[7]：1)采用基于距离的聚类的方法来识别异常的记录[8]。2)采用关联规则的方法来发现数据集中不符合具有高置信度和支持度的规则的异常数据。3)采用基于模式的方法来发现不符合数据集中现存模式的异常记录。4) 采用统计学的方法来检测数值型属性，计算字段值的均值和标准差，考虑每一个字段的置信区间来识别异常字段和记录。

其中对于值为字符型的属性来讲，利用了属性间的约束关系、模式识别技术等，难度较大。属性清洗在很多情况下针对具体问题具体分析，结合特定领域数据清

洗所达到的效果更好。

② 识别并消除数据集中的重复对象，即消除重复记录[9]。这方面研究在数据集成和数据仓库领域更加广泛。当集成不同的系统时，由于不同系统之间可能存在冗余，就会产生大量的重复记录。对于消除数据集中的近似重复的记录问题的研究是目前数据清洗领域中的热门课题之一。要从数据集中消除重复记录，首要需要考虑的问题就是如何判断两条记录是否是近似重复。对于两条数据记录的判断，归结到底还是对字段的判断，即字段的匹配问题。目前在这方面常用的算法主要有[10]：R-S-W 算法、编辑距离算法 Smith-Waterman 算法和递归式字段匹配算法[11]。基本近邻排序方法是数据集识别重复记录的经典方法[12]。不少研究者针对这种算法的缺陷，提出了各种改进的算法，其中主要包括优先权队列清洗策略，多趟近邻排序方法等。

③数据仓库的 ETL 环节中的数据清洗。ELT(Extract Transform Load)是指数据的抽取、转换、装入，它是创建数据仓库的重要环节。它主要是用于解决组织机构内部的信息集成化问题和数据一致性问题，它从多个异构系统中采集数据，并将其转化为符合要求的格式[13]。据统计，对于一个数据仓库建设中，约 80%的工作量都耗费在 ETL 阶段。在 ETL 阶段产生的“脏数据”，几乎涉及了“脏数据”的所有类型，如重复信息、拼写错误、缺损数据等。由于数据仓库的数据一般都是通过集成来自多个不同的数据源，因此，在此阶段的数据清洗很大工作是识别并消除重复记录[14]。很多针对 ETL 中的数据清洗改进主要集中在一些两点：一是 ETL 的主要目标是为 OLAP 提供服务，却缺少地址、姓名等数据的清洗。二是采用固定不变的转换步骤对数据自动清洗效果不佳。尽管现在已经陆续出现了一些针对地址类和姓名类信息的清洗工具，但目前大多数清洗工具都是针对西文的数据清洗，对于中文数据不能适用。对于涉及到中文类信息，如地址类数据的清洗工具还很少。

④ 现有的数据清洗软件。随着国外的数据清洗技术快速发展，很多专业的数据清洗软件也应运而生，有些已经应用于商业用途，成为产品，有些是由各大学和研究机构开发的数据清洗软件。商业上的数据清洗软件主要有: Infomatica 公司的 PowerCenter 产品，SAS Institute 公司的 Warehouse Administrator 产品，Oracle 公司的 DataWareHouse Builder 产品，Acxiom 公司的 Acxiom 数据解决方案，BussinessObjects 公司的 DataCleansing 和 DataIntegrator 软件，WinPure 公司的 WinPure Clean&Match 产品，Data Junction 公司的 Data Junction 软件。在国外这些公司的软件产品中，其中比较有代表性的 Informatica 公司的 PowerCenter，其主要通过以下这些核心功能，来保证数据质量：1) 对数据转换与更正，能够分析、转换和更正当前数据。2) 对数据进行归档，可以评估当前正在处理的数据质量。3)

数据清洗与改进，它可以利用辅助字段和相关信息（如邮政编码和地理信息）对数据进行清理或改进。4) 数据匹配与整合，其能够较好地匹配相似的记录，并其可以基于预设的标准进行消除和整合。5) 数据质量流程管理。反复重复数据质量流程，衡量目前取得的成效。除此之外，它还能够提供详尽的数据质量分析报告，并且能够通过地理名称和邮政编码库来改进/更正名称和地址数据。

1.3.2 国内研究现状

目前国内对于数据清洗技术的研究，只能算是处于起步的阶段。尽管可以通过一些学术期刊及学术会议上能够查阅到有关数据清洗方面的文章，但很多只是理论性质或者概括性质的，而直接针对数据清洗，特别是针对中文数据清洗的论文并不多。现在的国内的研究主要集中在以下几个方面：① 对于决策支持，数据分析和数据挖掘的研究。② 一些比较全面地的概述性研究，如郭志懋, 周傲英等人的《数据质量和数据清洗研究综述》[4]，佘春红等人的《数据清理方法》[15]，杨辅祥、刘云超等人的《数据清理综述》[16]等。③ 针对特定领域特定问题的应用研究，如沈国忠等人的《数据清洗方法在寿险事务处理系统中的应用研究》[17]，朱六璋等人的《调度信息系统的数据清洗应用》[18]，连仁包等人的《数据集成中数据清洗模型的研究》[19]等。银行证券类、邮政类和保险类等对客户数据的准确性要求高的行业，都在做自己的客户数据的清洗工作，结合自己具体的领域只是开发软件。但是在各种不同应用的系统间缺乏相互参考和借鉴，目前对于统一的客户数据清理研究知识有待做进一步的研究。

针对中文数据清洗在理论研究上的欠缺，也使得在市场上几乎很难找到有关中文数据清洗的专业产品。但是，既然有对中文数据清洗的需求，其相关技术一定会随之发展。随着数据分析、决策支持、数据挖掘以及客户关系管理等在更多行业中的大量应用，必然要求高质量数据的支持，也可以随之带动在项目的实施过程中对中文数据清洗技术和提高数据质量的方法的进一步研究，并带动更多高效实用的中文数据清洗产品的研发。目前，如何结合中文数据的特点，设计并实现具有很强针对性的数据清洗系统，也正在得到市场和科研的高度重视。

1.4 本文研究内容

基于以上分析，现有的数据清洗方法和技术多是零散的,还没有提供一个统一的使用平台。另外，现有的许多方法多从解决个案入手,与具体的场景结合密切, 没有充分考虑到以后的复用。为解决上述存在的问题，本文的研究目标是设计并实现一种基于软件总线模型的数据清洗系统。将软件总线模型的思想应用于数据清洗系统当中，使得系统能够通过灵活添加数据清洗组件以增强数据清洗功能，通过配置方式以适应多种异构数据源的数据清洗。具体来讲，本文所完成的主要内

容包括：

① 介绍数据清洗的研究背景和相关知识；分析国内外数据清洗方法的研究现状及存在问题，介绍国内外的数据清洗方法和数据清洗产品。

② 分析本文中用到的主要技术，其中包括 XML 技术，ADO.NET 数据访问技术，中文信息处理领域的中文分词技术等。

③ 研究软件总线模型的特点，并将其应用于数据清洗系统；研究对数据清洗任务进行分解的方法，采用多线程技术对数据清洗任务并发执行；结合中文地址的特点，提出一种基于分级树的中文地址数据规范化方法。

④ 从软件工程的思路入手，描述和设计系统的整体结构，然后对于系统中各主要部分分别进行设计和实现，并对系统运行结果展示。

1.5 本文的组织结构

本论文一共分为六章，各章节内容简介如下：

第一章绪论。介绍了本文的研究背景和意义，数据清洗的相关概念，国内外的研究现状以及本文的主要研究内容。

第二章相关技术分析。对于系统中运用到的主要技术进行分析。首先介绍了 XML 的相关知识，包括基本概念、有效性验证和解析 XML 的方法，然后对 ADO.NET 数据访问技术分析，最后对在中文数据清洗领域中常用到的中文分词技术进行介绍。

第三章对系统中关键技术进行研究。首先对软件总线模型进行研究，包括软件总线模型的结构，组件集成方式和总线控制等；然后对数据清洗任务分解的方法和并发性进行研究，通过实验验证其效果；最后对中文地址数据清洗方法进行研究，提出一种基于分级树的中文地址数据规范化方法，通过对比实验验证其效果。

第四章基于软件总线模型的数据清洗系统的设计与实现。首先对系统进行描述并制定设计目标，然后对系统体系结构进行设计及工作流程描述，将系统分解为各个组成部分，接下来对系统各个模块进行设计和实现，并对运行结果展示。

第五章总结与展望。对本文所做工作进行总结和展望。

1.6 本章小结

本章作为全文的绪论部分，首先介绍了数据清洗的研究背景，概述数据清洗的重要性；其次对数据清洗相关概念以及国内外研究现状进行分析；然后描述本文的主要研究内容，最后对本文的组织结构进行描述。

2 相关技术分析

2.1 XML 技术分析

2.1.1 XML 简介

XML(Extensible Markup Language，可扩展标记语言)是 W3C(World Wide Web，世界万维网联盟)提出的一种定义其它语言的元语言，是标准通用标记语言 SGML(Standard Generalized Markup Language)的一个精简子集。它以一种开放的自我描述方式定义数据结构，在描述数据内容的同时能突出对结构的描述，从而体现出数据之间的关系[20]。总的来说，XML 具有以下优点：

① 可扩展性强。XML 允许各个不同的行业根据自己的需要制定自己的一套标记，例如：化学标记语言 CML，数学标记语言 MathML。这就使得该领域中的人们可以自由交换信息，而不用担心接收端的人是否有特定的软件来查看信息。

② 形式和内容相分离。对于 XML 文档而言，标记是包含信息的，比如关键字继承关系等，这些信息对于数据的检索描述起着简化作用。只需修改从 XML 文档中分离出的用于数据表现的样式单就可以改变数据的表现形式时了[21]。

③ 可移植性强。XML 语言可以定义文本、图像、声音等各种数据。XML 可移植性的特征使得只要交换数据的系统能够处理 XML 文档，就能处理由 XML 标记的各种数据，从而实现了异构数据的跨平台交换[22]。

④ 严格的语法规则。XML 有着严格的语法规则并且很注重准确性，就算语法有丝毫的错误，分析器都会停止对它的进一步处理。这样就保证了 XML 文档的可读性和可维护性。

⑤ 灵活的自描述性。灵活的自描述性使得 XML 数据可以被不同的程序进行分析处理，并且 XML 的自描述性可以使一篇 XML 文档更加容易理解。

⑥ 方便不同的系统之间信息传输。XML 是公开的并易于阅读和编写，就使得 XML 成为在不同的应用之间交换数据的理想格式。

正是由于 XML 以上这些优点，在本文中，使用 XML 来实现数据的共享，数据传输和系统中一些模块的集成。

2.1.2 XML 有效性验证

只有不违反 XML 的语法规则的前提下，XML 文档中的数据才能被统一处理，。因此需要一种机制来指定应该如何构造描述同一事物的 XML 文档。目前，XML Schema 和 DTD(Document Type Definition，文档类型定义)是使用最广泛的进行 XML 文档有效性验证的两种机制[23]。

DTD 出现较早，用于详细描述一组 XML 文档的结构，它可以是 XML 的一部

分，但是它通常是一份单独的文档或者一系列文档。DTD 说明在 XML 文档中可以使用的标记，哪些标记可以出现在其它标记中，哪些标记具有属性有机使用的标记应按什么顺序出现等。XML 文档没有一个通用的 DTD，因此想使用 XML 进行数据交换的组织可以自定义适合自己的 DTD，DTD 规定了一个语法分析器用于解释 XML 文档所需要的所有规则的细节。DTD 标记声明可以是属性声明，元素类型声明，符号声明或实体声明。DTD 对于 XML 文档的结构起到很好的描述作用，是近年来 XML 技术领域所使用广泛的一种用于 XML 文档有效性验证的机制。

随着 XML 的广泛使用，DTD 逐渐显出一些缺陷，比如采用了非 XML 的语法规则，不支持不支持名称空间，不支持多种数据类型，扩展性较差等。于是，W3C 推出 XML Schema 作为 XML 的标准模式。XML Schema 是一个符合 XML 语法规则的 XML 文档，可以用 XML 解析器进行解析。它如同 DTD 类似，负责定义和描述 XML 文档的结构以及内容模式。XML Schema 具有以下一些优点：

① 规范性。XML Schema 也提供了一套完整的机制以约束 XML 文档中标记的使用，相比 DTD，XML Schema 更加规范化，它定义了 XML 的整体结构以及 XML 中元素间的关系等。

② 一致性。由于 XML Schema 本身就是是 XML 格式的文档，使其得对 XML 的定义能够直接借助 XML 自身的特性，而不需要再利用一种特定的形式化语言，达到从内到外的统一。

③ 互换性。用户可以根据自己的需要，设计出适合自己的 XML Schema，并且可以与其他用户互换各自的 XML Schema，并且还可以通过映射机制将不同的 XML Schema 进行转换，实现更高层次的数据交换。

④ 可扩展性强。XML Schema 可以说是对 DTD 的扩充，并引入了命名空间和数据类型等，使其可扩展性更强。

鉴于 XML Schema 这些优点，在本文中，采用了 XML Schema 来进行 XML 文档的有效性进行验证。

2.1.3 XML 文档解析

在目前针对 XML 文档的解析编程接口中，最主要的两种方式分别是 DOM(Document Object Method，文档对象模型)和 SAX(Simple API for XML，用于 XML 的简单 API) [23]。DOM 和 SAX 两者的实现分别侧重于不同方面以满足不同需求。

DOM 提供一种随机访问的模式，使得应用程序可以通过此接口在任何时候访问 XML 文档中的任何位置数据，也可以用于对 XML 文档中数据的插入、修改删除等操作。DOM 将 XML 文档的逻辑结构看做是一种树型结构。对于文档和文档中的元素、属性等都是以对象模型表示的。DOM 的优点是其在内存中保存文档的整个模型，可以快速地以任何顺序访问到 XML 元素。然而，对于大型 XML 文档，

将整个文档加载至内存中的方式是不可取的。

SAX 提供一种对 XML 文档顺序访问的机制，它是一种快速读写 XML 数据的方式。SAX 接口是基于事件驱动的，当采用 SAX 分析器对 XML 文档进行分析时，就会触发一系列相关事件，并激活其对应的事件处理函数，从而完成对 XML 文档的访问。SAX 处理 XML 的方式与 DOM 差别较大。SAX 是基于事件的，当解析 XML 时，事件被传递给引擎。SAX 可以在文档的开始接收事件，也可以接收文档中的元素。因为 SAX 并不需要把 XML 文档整个加载到内存中，所以消耗的内存资源较少，是解析大型 XML 文档的高效方式。相比较 DOM 解析方式，其缺点也明显。首先编写 SAX 比编写 DOM 复杂，这因为首先必须实现通知接口并需要维护状态，其次 SAX 不支持对文档的随机访问，也并不提供 DOM 方式的修改功能。

经过上述比较分析，总结而言，DOM 适用于以下情况：① 解析比较小的 XML 文件；② 需要对文档进行随机访问；③需要对文档进行修改。SAX 适于处理下面的问题：① 对大型文档进行解析；② 只需要从文档中得到特定信息；③ 只需要文档的部分内容。这两种方式各自有其适用场景。在本文中，根据具体的需要，分别应用了这两种技术来解析 XML 文档。

2.2 ADO .NET 技术分析

ADO.NET 是微软.NET Framework 中数据访问组件，它提供了对关系数据、XML 和应用程序数据的访问。在 ADO.NET 中提供两个组件来访问和处理数据：.NET Framework 数据提供程序和 DataSet[24]。下图 2.1 为 ADO.NET 的结构图，其中描述了以上两个组件的组成部分。

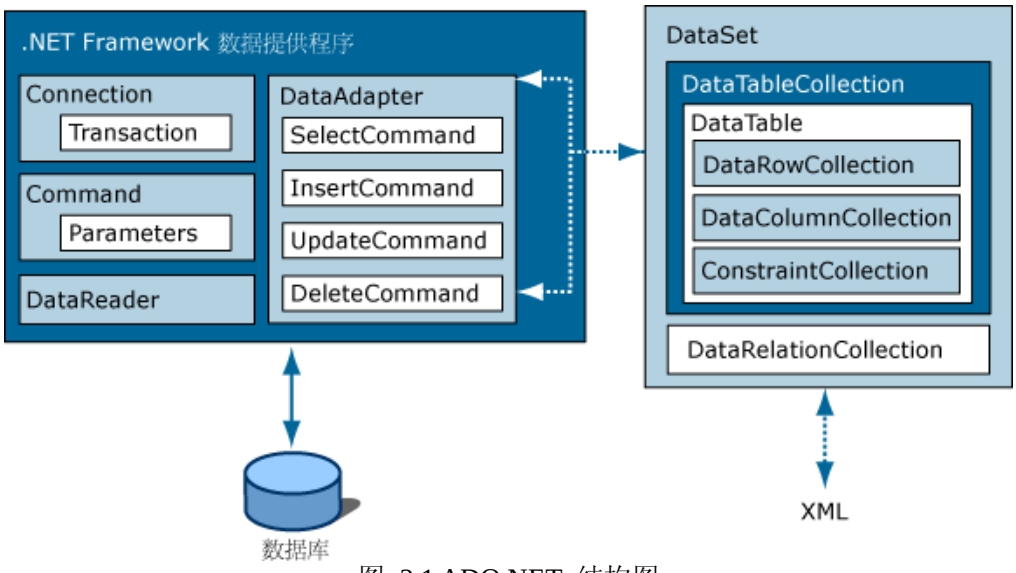


图 2.1 ADO.NET 结构图

Fig. 2.1 Structure of ADO.NET

.NET Framework 数据提供程序是专门为数据处理以及快速地只进、只读访问数据而设计的组件。其中的 Connection 对象提供与数据源的连接。Command 对象使用户可以访问用于返回数据、修改数据、运行存储过程以及发送或检索参数信息的数据库命令。DataReader 从数据源中提供高性能的数据流。DataAdapter 提供连接 DataSet 对象和数据源的桥梁。DataAdapter 使用 Command 对象在数据源中执行 SQL 命令，以便将数据加载到 DataSet 中，并使对 DataSet 中数据的更改与数据源保持一致。

ADO.NET 中的 DataSet 是专门为独立于任何数据源的数据访问而设计。因此，它可以用于多种不同的数据源，用于 XML 数据，或用于管理应用程序本地的数据。DataSet 可以包含一个或多个 DataTable 对象的集合，这些对象又由数据行和列以及有关 DataTable 对象中数据的主键、外键、约束和关系信息组成。

在以上 ADO.NET 的结构描述中，一般有两种数据读取方式，即通过 DataReader 和 DataSet 的方式[24]。DataReader 和 DataSet 各种有其特点，在选择使用时根据应用程序的具体需求而定。DataSet 主要用于处理一下情况：

① 在应用程序中将数据缓存在本地，以便可以对数据进行处理。如果只需要读取查询结果，就不需要使用 DataSet，可以使用 DataReader 以提高性能。

② 与数据进行动态交互，例如绑定到控件或者是需要组合并关联来自多个源的数据。

③ 对数据执行大量的处理，而不需要与数据源保持打开的连接，从而将该连接释放给其他客户端使用。

如果不需要 DataSet 所提供的功能，通过使用 DataReader 以只进、只读方式返回数据，从而提高应用程序的性能。虽然 DataAdapter 使用 DataReader 来填充 DataSet 的内容，但可以使用 DataReader 来提高性能，因为这样可以节省 DataSet 所使用的内存，并将省去创建 DataSet 并填充其内容所需的处理。

在本文针对不同数据源的数据清洗，需要大量使用到数据访问技术，在对 ADO.NET 使用方式的选择时，根据系统中的具体需要而定。

2.3 中文分词技术分析

在中文信息处理领域中，中文分词是经常用到的技术。目前现有的中文分词方法主要分为以下三大类：基于词典匹配的分词方法、基于理解的分词方法和基于统计的分词方法[25]。

① 基于词典匹配的分词方法。

这种方法是一种机械分词方法，它以一定的策略将待分解的字符串与一个巨大的词典库的词条进行配，如果能从词典中找到某个字符串，则匹配成功。根据对匹配字符串扫描方向，基于词典的分词方法可以分为正向匹配和逆向匹配;根据每次匹配到的字符串长度优先匹配的情况，可以分为最大匹配和最小匹配;根据是否有词性标注，又可以分为单纯分词方法和分词与标注相结合的方法[26]。

目前常用的几种机械分词方法有：正向最大匹配法，逆向最大匹配法，最少切分法等。另外还可以将上述这些方法相结合，比如，双向匹配法就是将正向最大匹配方法和逆向最大匹配方法的结合。

除了上述这些常用的基于词典的分词方法外，还有一些改进的方法。其中一种分词方法是特征扫描法，优先从待分解的字符串中识别并切分出一些具有明显特征的词，然后将这些词作为分界，可将原字符串分解成多个较小的串再来分词，可以减少匹配的错误率。还有一种方法是将分词和词类标注结合起来，利用词类信息对分词策略提供指导，并且在标注的同时又对分词结果进行检查，从而很好地提高切分的准确率。

② 基于统计的分词方法。

中文词语从形式上看是由稳定的字组合而成，因此结合全文，相邻的字同时出现的次数越多，就构成一个词语的可能性就越大。从这个角度考虑，字与字相邻共现的频率或概率能够在一定上反映成词的可信度[27]。因此，可以对语料中相邻共现的各个字的组合的频度进行统计，计算两汉字 A、B 的相邻共现概率。当着 A 和 B 共现概率高于某一个阈值时，便可认为 A 与 B 可能构成了一个词。这种方法需要对语料中的字组共现频度进行统计，而不需要切分词典。在一些应用中，为提高性能，采用统计与词典相结合的方式进中文分词[28]。

③ 基于理解的分词方法。

这种分词方法的基本思想就是在分词的过程中进行句法、语义分析，利用句法信息和语义信息来处理歧义现象，简单来讲，就是通过让计算机模拟人对句子的理解，以达到识别词的效果。这种分词方法一般由三部分组成：分词子系统、句法语义子系统、总控部分。在总控部分的协调下，分词子系统可以获得有关词、句子等的句法和语义信息来对分词歧义进行判断，即通过它来模拟人对句子的理解过程，这种方法通常需要大量的语言知识和信息。由于汉语语言知识的复杂性和笼统性，难以将其组织成机器能够理解的形式，因此目前基于理解的分词系统更多的是处于试验阶段。

表 2.1 三种中文分词方法效果对比表

Tab 2.1 Comparison Table between Three Chinese Word Segmentation Method

分词方法	基于词典匹配分词	基于理解的分词	基于统计的分词
歧义识别	差	强	强
新词识别	差	强	强
需要词典	需要	不需要	不需要
分词准确性	一般	准确	较准
分词速度	快	慢	一般
算法复杂性	容易	复杂	一般
技术成熟度	成熟	不成熟	成熟
需要语料库	否	否	是
需要规则库	否	是	否
实施难度	容易	很难	一般

通过上表 2.1 中对三种分词方法的对比，三种算法各有优缺点，到底哪种分词算法的效果更高，很难有定论。在应用中，选择哪种分词方法要根据实际的具体需求来决定。在本系统中，对中文地址数据进行清洗时需要用到中文分词技术，根据中文数据的特点，由于地址词典库数量有限，通过优化地址词库结构，采用基于词典匹配方式进行分词效果更好。

2.4 本章小结

本章首先介绍了 XML 的相关知识，分析 XML 的特点，有效性验证方式以及文档解析方法，然后介绍了数据库访问 ADO.NET 技术，分析 ADO.NET 的结构和不同数据读取方式的特点，最后对于中文信息处理中的中文分词技术进行介绍，分析比较三种主要中文分词方法的效果。

3 基于软件总线模型的数据清洗系统中的关键技术研究

3.1 软件总线模型研究

3.1.1 软件总线结构

软件总线模型是为了解决复杂软件系统的集成问题而提出的系统解决方案。软件总线与计算机的硬件总线一样，如 ISA、PCI 总线等，对于软件总线的使用方式跟硬件总线类似，只要按照总线制定标准，就能对软件组件进行无缝连接和交互[29]。软件总线的主要工作是负责在各个组件中传递信息流，将各个组件有机地组织起来，完成一个或一系列具体的任务。总线是一个抽象的概念，在实际应用中则是由具体的技术构成。软件总线模型的一般结构如图 3.1 所示。

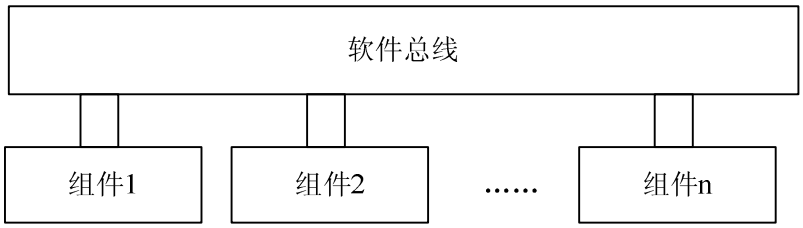


图 3.1 软件总线模型结构图

Fig. 3.1 The Structure of Software Bus Mode

在软件总线的设计与实现过程中，其核心问题主要体现在 3 个方面[30]：①定义总线的接口和协议，这样符合协议的组件才可即插即用；②组件的调度与管理；③组件间的通信管理。基于以上核心问题，软件总线设计首先需要制定总线的相关协议，包括组件集成协议和数据格式定义。同时，软件总线还需要有专门的总线控制功能，负责对总线的启动，对于挂接的组件进行加载，对于请求进行解析，对组件进行选择性调度；组件管理功能负责对新组件注册、现有组件的卸载和更新任务。

从组件可复用的角度考虑，组件的开发往往具有相对独立性，可以融入到不同的系统中。从这个角度考虑，需要集成到软件总线上的组件本身并不能满足总线制定的协议，不能直接集成到总线上与总线进行通信。因此，需要针对不同的组件，定制符合总线标准的适配器[31]，由适配器来负责组件与总线之间的通信。

基于上述分析，本文提出的在数据清洗系统中应用的总线结构如图 3.2 所示

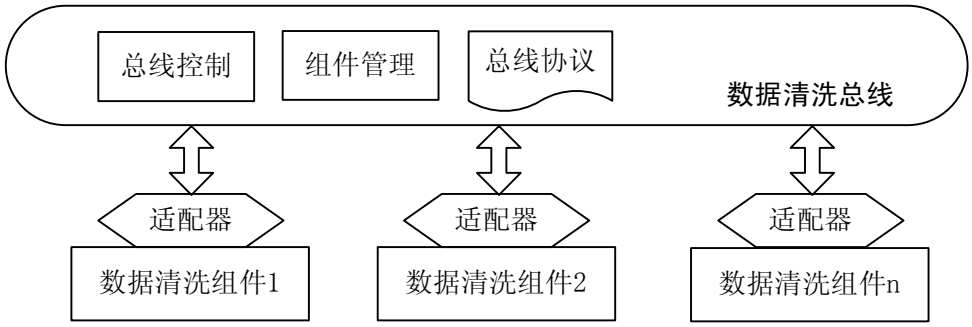


图 3.2 系统中数据清洗总线结构

Fig. 3.2 The Structure of Data clean Bus in System

上图中所示，从数据清洗总线整体结构来讲，包括三部分：①数据清洗总线，其是整个模型的核心部分，其任务包括对总线协议制定，对挂接到总线上清洗组件进行管理，对从客户程序发送的清洗请求进行解析，根据请求选择对应的清洗组件。②适配器，是一个抽象层次的概念，负责不兼容接口之间的通信。在本文中，适配器负责数据清洗总线与数据清洗组件的交互。③数据清洗组件，具有相对独立的数据清洗功能的模块。

3.1.2 软件总线协议

协议是一个抽象概念，它是通信双方为了实现通信而设计的约定或规则。在软件总线中，需要制定一些协议，以使的满足协议的组件能够挂接到总线上来，对于客户程序发送的请求能够正确解析[32]。在本文中的数据清洗总线中，主要定义两种协议，一种是用于组件集成的协议，另一种是对清洗请求进行规范的协议。

①组件集成协议。软件总线需要集成多种可能是不同来源，相对独立的组件，只有制定了协议规范，组件遵照规范，组件才能与总线之间正常通信。组件对于总线来讲是相对独立，将独立的组件集成到总线上来不可能去修改原有组件的接口，在这种意义上讲，组件本身是不可能遵循组件集成规范。如何将功能独立的组件与软件总线进行结合。这就需要引入适配器，通过适配器来满足组件集成的协议。因此，对于适配器的实现，除了需要兼容组件外，还需要满足总线协议。

②清洗请求协议。清洗总线并不是对于所有的清洗请求都能接受，就需要对清洗请求进行规范，只有满足总线协议的消息才能够被正确解析，选择调度合适的组件。将请求以 XML 的形式进行封装，在总线上对于 XML 的格式通过 XML Schema 进行定义来规范数据清洗请求协议。之所以使用 XML 文件作为传递报文，是为了使得系统能够具备更好的可复用性和松耦合性。

3.1.3 组件管理

组件结构是面向对象结构的继承和延伸，它具有面向对象结构的所有优点。组件是一种封装良好的功能部件，对外具有一致的接口[33]。组件管理是指向总线注

册组件、更新组件，卸载组件。对组件管理往往需要在总线中保留组件的注册描述信息，其中主要包括组件名称，调用接口，组件功能等信息。本文通过适配器对组件进行调用，在组件注册信息中需要添加组件所对应的适配器接口，以用于总线调用。另外，结合数据清洗应用，在组件描述中还需要具备对清洗功能的描述。

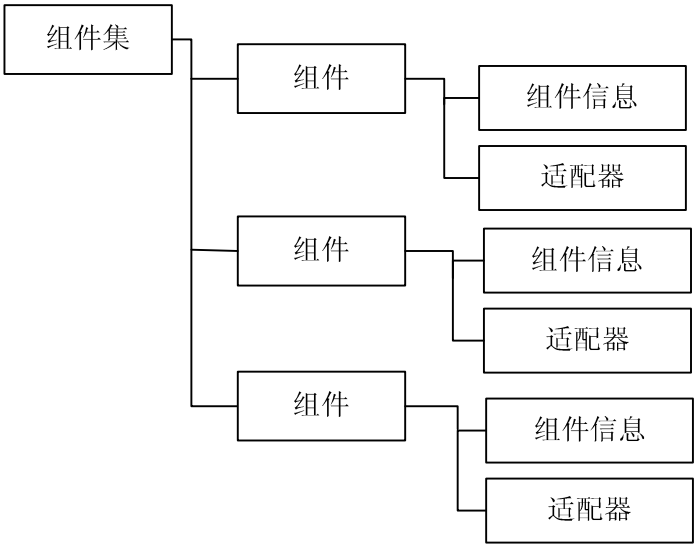


图 3.3 组件集结构图

Fig. 3.3 The Structure of Components Set

本文中通过 XML 形式对组件集进行表示，因此对于组件的管理实质是对组件库 XML 文件的管理，XML 的对象模型树(DOM)中对组件集的描述，如上图 3.3 所示。

对组件的管理，通过对 XML 文件中对象模型树(DOM)节点进行编辑来控制。向总线注册新组件，只需要添加一个组件(Component)节点，并补充节点中属性信息。其中的适配器(adapter)节点是每个组件所必须的，适配器负责组件与总线进行通信交互。

3.1.4 总线控制

在软件总线模型中，都需要有对总线进行控制管理的功能，负责对总线的状态进行管理，对消息进行路由，对组件进行调度[34]。在本文中研究的数据清洗总线，对于总线控制的主要体现：在对于数据清洗总线上数据清洗组件的加载；对于清洗请求进行解析，根据请求选择合适的数据清洗组件进行调用。

① 加载数据清洗组件。

上节中已经对组件集的静态结构进行描述，组件集是通过 XML 文件形式进行存储[35]，对于系统来讲，不可能在每次的解析清洗请求，从组件集选择组件时都

去解析 XML 文件，这样显然是一种费时且效率低下的方法。因此，为了提高对清洗组件查询的效率，有必要设计一个数据结构，来对组件进行动态映射，在数据清洗总线启动时将组件集加载到内存中。系统根据组件的数据清洗功能进行分类，通过一种 Hash 结构来存储对数据清洗组件的引用，键值为待清洗数据字段，其所对应的值为根据优先级排列的数据清洗组件链表，如下图 3.4 所示。

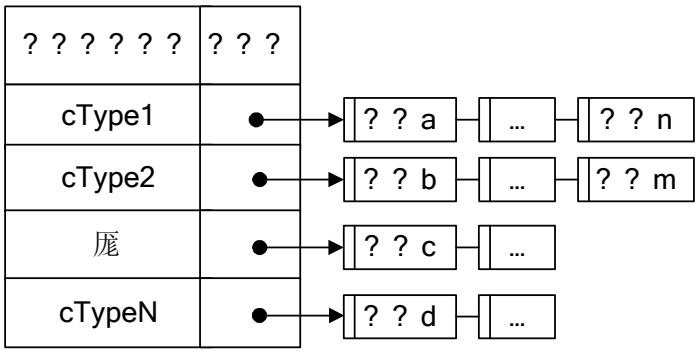


图 3.4 组件在内存中的存储结构

Fig. 3.4 The Storage Structure of the Component in Memory

② 解析请求，调用数据清洗组件。

总线控制需要对客户程序发送的请求进行解析，根据解析结果去查询合适的数据清洗组件。在组件描述中包含对其优先级的描述，总线会根据其优先级对合适的组件进行顺序调用，完成一系列任务。按照上图 3.4 中对于组件动态存储结构的描述，通过待清洗数据的字段所查询得到的是多个以清洗组件为元素的数组，就需要将这些数组合并后，按照优先级进行排序，对组件依次进行调用。

3.2 清洗任务分解与并发性研究

数据清洗系统通常需要对不同数据来源的上万级，甚至是百万级，千万级的数据进行清洗。如果对于这些数据按照顺序执行清洗任务，将会消耗大量的时间，对 CPU 和内存利用率很低，所以从提高系统性能的角度考虑，对清洗任务进行分解，并通过多线程同时执行是很有必要的。

3.2.1 任务分解

数据清洗的任务可以描述为对数据源中的脏数据进行清洗，由于数据量可能是海量的，对清洗任务的分解可以从对待清洗的数据本身进行分解入手，将原有的数据集分解为多个小的数据集[36]，然后同时对多个子数据集执行清洗任务。

下面举一个简单的例子：假设待清洗的数据有 10000 条记录，对于每条记录的清洗时间消耗为 T，则对于执行完这 10000 条记录的数据清洗任务耗时为 10000*T;

如果将这 10000 条数据分解为 100 个 100 条的小的数据集，对这 100 个数据集同时执行数据清洗任务，在不考虑内存和 CPU 的影响情况下，可以同样认为每条记录的清洗时间消耗为 T，则完成数据清洗任务的总耗时为 100*T。显然对于清洗任务分解后的清洗效率会有显著提升。对于数据集的分解有两种方式，如下图 3.5 中所示。

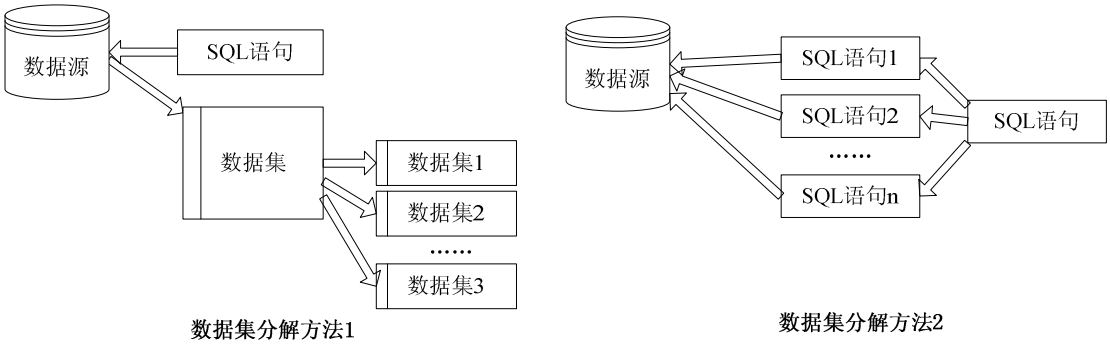


图 3.5 数据集分解的两种方法

Fig. 3.5 Two Methods for the Data Set Splitting.

方法① 将数据集全部读取出来，对于读取出来的数据进行分解。这种方式对于数据量小时尚可行，但是一旦数据量很大，将海量的数据全部读取出后再分解显然是不可行的。方法② 由于数据源多为关系型数据库，其数据集是通过 SQL 语句查询得到，因此可以考虑在 SQL 语句中添加限定条件，将查询分解成若干子查询，分别读取其对应的数据集，而没必要一次性将大数据量读取出来。在本文中采用第二种方式来对待清洗数据集进行分解，并以此完成对数据清洗任务分解的目的。不同数据源不仅数据模型不同, 且查询能力各异, 给查询的分解带来了问题[37]，在本文中实现的异构数据源只支持关系型数据库。在数据库应用中，经常需要用到对于查询数据进行分页，其实就是对数据集的分解，只是本文中所分得的每页的数据量较大，页数较少。

不同的数据库其对 SQL 语句分解的方式不同，下面以 Oracle 数据库中的一条查询为例，描述其分解方法。

假设有这样的对 Oracle 数据库中的查询，原始 SQL 语句为：SELECT * FROM TableName WHERE Condtion。通过这条语句所查询出的结果集为满足 Conditon 条件的所有数据。下面将这条语句分解成多个语句，以分别进行查询。

定义数据集的总大小为 Total,可以通过 SQL 语句中 COUNT 函数来获取到；设定任务分解个数为 N，这样每个分解后的数据集大小 Length = (Total/N)。在 Oracle 中，可以对原始 SQL 语句进行修改，以查询指定区间范围内的数据，如下 SQL 语

句所示：

```
SELECT * FROM (SELECT A.*, ROWNUM RN    FROM (原始 SQL 语句) A)
    WHERE RN BETWEEN [Start] AND [End];
```

通过这种方式，原始 SQL 语句就被分解为 N 个子 SQL 语句，在每个子 SQL 语句中去变化 ROWNUM 的起始值和结束值。这些子查询所得到结果集之和为原始 SQL 的查询结果集，如下图 3.6 中所示。

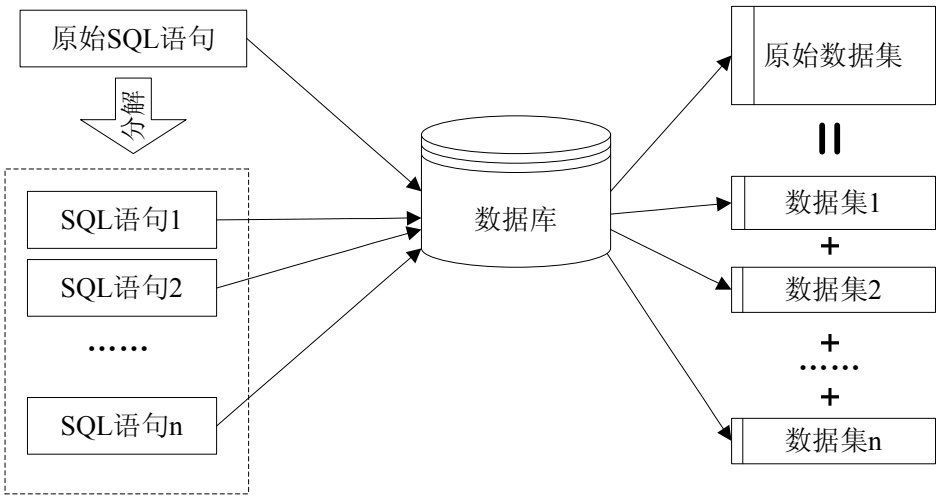


图 3.6 通过分解 SQL 对数据集进行分解

Fig. 3.6 Split the Data Set by splitting SQL

3.2.2 并发与多线程

为提高系统处理大数据量时的性能，本文中需要使用了多线程技术。多线程的主要任务是解决处理器单元内多个线程执行的问题[38]，其可以显著减少处理器单元的闲置时间，提高处理器单元的吞吐能力，实现多任务并发工作。

在使用多线程技术时，影响系统性能的一个重要因素需要考虑，就是线程数目[38]。线程个数太多或者太少，系统的性能都不会太好。如果线程个数太少，对于内存和 CPU 使用效率不够；如果线程个数太多，CPU 用于切换线程的开销会很大，同时处理数据同步的开销也会增大。在本文中，多线程应用中将线程的数目控制在 15 个左右，当然这个需要根据具体的硬件条件而决定。

在面向对象编程中，创建和销毁对象是很费时间的，因为在每次创建新的对象时都要获取内存资源或者其它资源。在.NET 中更是如此，.NET Framework 将会通过垃圾回收机制跟踪每一个对象，以便能够在对象销毁后对其进行回收。线程在.NET 里面同样也是对象，所以不管线程的创建还是销毁，都会消耗资源。如果对多线程使用不当，不但不能很好地提高系统的性能，反而可能会增加对单个任务的处理时间。

在这里举这样一个例子：假设系统完成一项任务的时间为 T，创建线程的时间为 T1，线程执行任务的时间为 T2，销毁线程的时间 T3，那么显然 T＝T1＋T2＋T3。如果要完成 1000 个任务，那么所需的时间就是 1000*T1+1000*T2+1000*T3。可以看出提高程序效率的一个手段就是尽可能减少创建和销毁对象的次数特别是一些很耗资源的对象的创建和销毁，达到减少总时间 T 的目的。

考虑到创建销毁线程的时间消耗，数据清洗任务的个数和线程数目有很大关系。如果任务分解过少，对线程的运用不够充分；如果任务分解过多，则会造成多次创建和销毁线程的消耗。基于以上分析，对于数据清洗任务的分解个数最好根据线程的具体数目来确定。假定在系统中开启 N 个线程，则待清洗数据集根据线程数目就分解为 N 个，每个线程担任每个数据集的清洗任务。在完成整个数据集的清洗任务过程中，只需要这 N 个线程的创建和销毁的开销。

3.2.3 实验分析

在上述对于任务分解和多线程技术描述的基础上，对来自 Oracle 数据库的的待清洗数据记录进行实验，通过变化待清洗数据量大小，以比较不采用多线程技术，使用多线程之间效果差别。为了便于进行实验分析，只对读取出的数据做简单的清洗处理，在实际的应用中对于数据清洗方法要相对复杂得多，其耗时会大于这个值。下表 3.1 中是使用对数据清洗任务在不使用多线程和使用多线程两种情况下，时间消耗的对比表。① 在不使用多线程情况下，对于待清洗的数据依次顺序执行清洗任务。② 在使用多线程情况下，线程数目设置为 10，即对待清洗数据集分解为 10 个数据集，交给每个线程，同时进行数据清洗任务。

表3.1 不使用多线程和使用多线程耗时对比表

Tab. 3.1 The Comparison Table for using Multithread and no using Multithread.

待清洗的数据记录	多线程使用方式	时间消耗
10000	不使用多线程	3 分 25 秒 687 毫秒
10000	使用多线程	28 秒 126 毫秒
20000	不使用多线程	6 分 45 秒 125 毫秒
20000	使用多线程	54 秒 101 毫秒
50000	不使用多线程	16 分 12 秒 112 毫秒
50000	使用多线程	2 分 40 秒 321 毫秒
80000	不使用多线程	25 分 12 秒 211 毫秒
80000	使用多线程	3 分 25 秒 451 毫秒
100000	不使用多线程	35 分 12 秒 89 毫秒
100000	使用多线程	5 分 2 秒 342 毫秒

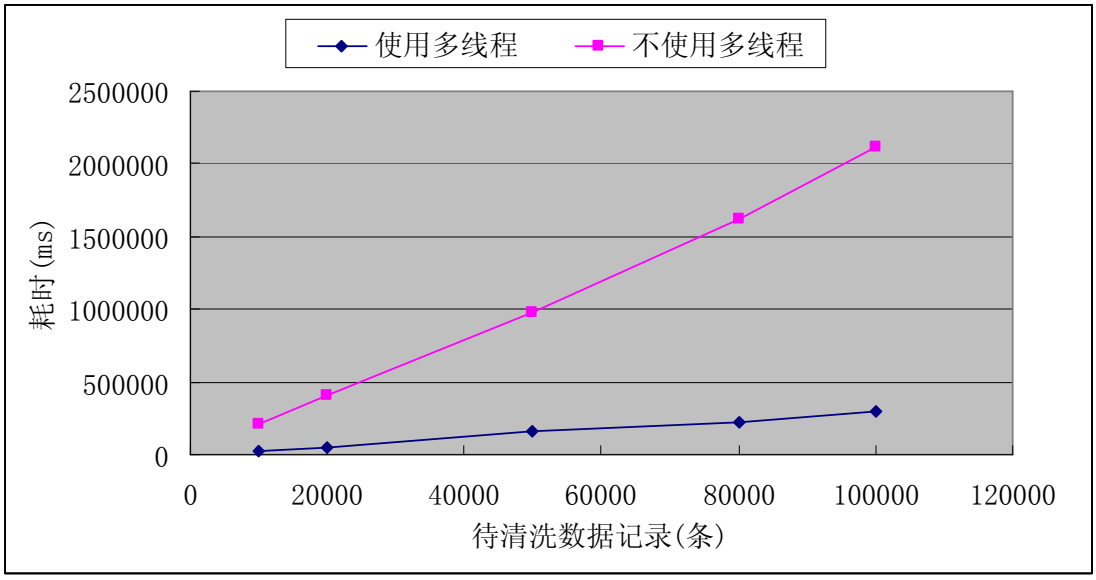


图 3.7 不使用多线程和使用多线程耗时对比图

Fig. 3.7 The Comparison for using Multithreads and no using Multithread.

从上图 3.7 中可以看出，对于同等数据量的清洗时间消耗，非多线程情况下的时间大概是多线程清洗下时间消耗的 10 倍（10 为线程数目）；随着待清洗数据量的增长，通过多线程并发执行数据清洗任务的效率优势更加明显，在数据量较大时，创建线程的时间消耗可以忽略。

3.3 中文地址数据清洗方法研究

本文中研究的数据清洗系统，需要由具体的数据清洗方法来构成。中文地址数据清洗是中文数据清洗领域中重要研究课题之一，也是本文研究内容之一。数据规范化是数据清洗的重要环节。本文结合中文地址的结构特点，提出一种基于分级树的中文地址数据规范化方法，将不规范的中文地址数据转化为规范地址格式。

3.3.1 方法描述

诸如银行、电信、邮政等面向客户的组织拥有大量的客户地址信息，它们具有很大的商业价值。然而，由于来源不同，输入不规范等原因，这些客户地址数据存在严重的不规范性，这种不规范性主要体现在地址表示方式不一致，地址结构不完整，地址存在别名等方面。本文根据中文地址数据的结构特点，提出一种基于分级树的中文地址规范化方法，将这些不规范的地址数据转换为统一的标准地址形式。

为了方便叙述，先给出如下定义:

定义1.原始地址是指从不同数据源获取的未经过处理的表示地址的原始信息。

定义2.标准地址是指邮政地址库中的中文地址的标准的表示形式。

定义3.地址元素是指地址信息的最小组成单元，具有不可分解的原子特性，表示某个具体的地址实际值。

定义4.特征字符是指在地址信息中，只表达地址元素的特征，而不能表达地址实际值的字符。一般而言，地址元素通常具备一定特点，例如在表示“省份”地址元素尾部经常出现“省”、“自治区”字样的词条，这里把这些词条定义为特征字符。

定义 5.完整地址。地址的组成元素包括省、市、县，街道、门牌、楼、单元、室等要素，完整地址是指从省(市)名开始依次从大到小、最终到最小地址元素的地址。

定义 6.门址解析是指对各种门牌号及缩略形式的标准化解析，如“A-2-3-4”解析为“A 栋 2 单元 3 层 4 室”，这个需要根据实际规则解析。

以中文地址信息“重庆市沙坪坝区沙正街114号”为例，其中的“重庆”、“沙坪坝”是地址元素，而“市”，“区”是特征字符。原始地址和标准地址的一般组成结构{地址元素+特征字符+…+地址元素+特征字符}

有了以上的定义和分析以后，可以将本方法概括地描述为输入原始地址，与标准地址库进行比较，得出与原始地址表示意义相同的标准地址作为输出。原则上，与原始地址的所有字符完全相同的标准地址是最佳输出。但是由于存在前面提到地址不规范性的情况，通过全字匹配来得到标准地址的方式的召回率很低，对于不规范地址的匹配效果差。另外，标准地址库数据信息并不完备，对于标准地址库中不存在的地址数据不能输出。

在地址元素中，同一地址元素所对应的特征字符不同是地址不规范性的重要原因，因此有必要对地址信息进行拆分，对每一级别地址元素进行比较，特征字符单独比较。目前已有对中文地址的分词研究[39-41]，但是用于本文中效果不好，对于中文地址的分解中缺少对地址不规范考虑。

然而在对原始地址的拆分过程中，我们并不能知道地址元素与特征字符之间的分界，并且有些时候特征字符会被省略掉。因此，就需要利用地址词典对单个的地址元素进行匹配。基于词典的匹配方法实际上是一个在词表上查找目标词的过程，词表相当于一个查找表。因此，匹配的速度一方面取决于查找表的大小，另外一方面还需要取决于查找表组织方式和查找策略。尽可能地缩小每次查找范围是提高匹配效率的一种有效方法。根据上述分析，基于原有的标准地址库的匹配，一方面查找表数据量很大，另外其结构不适于快速查找。因此需要在原有标准库的基础上构建一种新的用于匹配的地址库。

3.3.2 分级树地址库

中文地址信息具有以下特点：①中文地址的一般组成结构{地址元素+特征字符+…+地址元素+特征字符}。②地址元素级别从大到小进行有序排列，地址元素之

间具有从属关系。③中文地址信息中表示省、市、县，街道，建筑物的地址元素相对精确，而表示门牌、楼、单元、室的地址元素多为数字或字母加汉字表示，变化较多，如“重庆市沙坪坝区沙正街[数字]号”。经过上述对中文地址特点的分析，可以基于中文地址这些特点，构建一种适用于地址匹配的分级地址库，这种地址库应该有如下特点：①它是一种按照地址级别从大到小排列的树型结构。②地址元素与特征字符分开存储。③包含对门址解析规则和标准门址形式。如下图3.8所示，其为地址数据转化的方式。

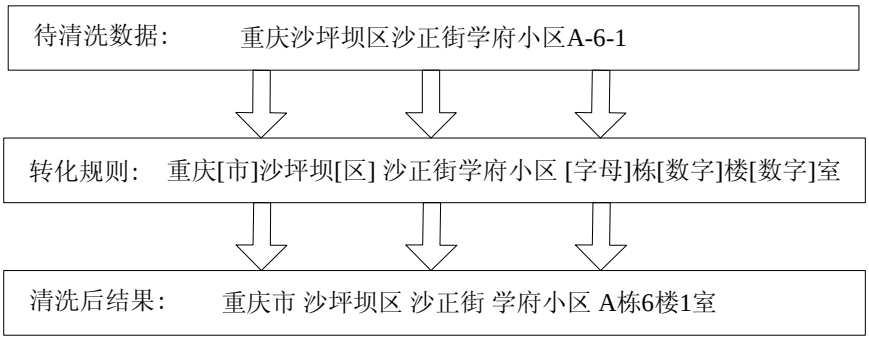


图 3.8 地址数据转换格式

Fig. 3.8 The Format for Address Data Converting

在基于词典的中文分词中,TRIE索引树是一种提供高效检索的数据结构。TRIE索引树是一种以树的多重链表形式表示的键树，通常的TRIE树的词典结构有3个基本性质: ①根节点不包含字符，除根节点外每一个节点都只包含一个字符。②从根节点到某一节点，路径上经过的字符连接起来，为该节点对应的字符串。③每个节点的所有子节点包含的字符都不相同。

在本方法中，分级树地址库采用TRIE树结构的思想进行构建，而和传统的TRIE树具体结构的又有所不同，在节点中不是包含一个字符，而是一个地址元素，叶子节点为街道或者建筑物地址元素，其指向对于门址的解析规则。如下图3.9所示. 基于TRIE树的地址词典有两部分组成, 第一部分为首个地址元素散列表,对应地址元素的单元包括指向该地址元素的TRIE索引树的根结点指针,对于中文地址来讲,首个地址元素一般是省份自治区直辖市名称。第二部分为TRIE索引树结点,是一个数组,每个单元的信息域包括：①地址元素②子树指针③地址级别，地址级别从A级到E级，不同的级别对应不同的特征字符集。④是否为叶子节点的标志位，叶子节点指向门址的解析规则和标准门址形式。

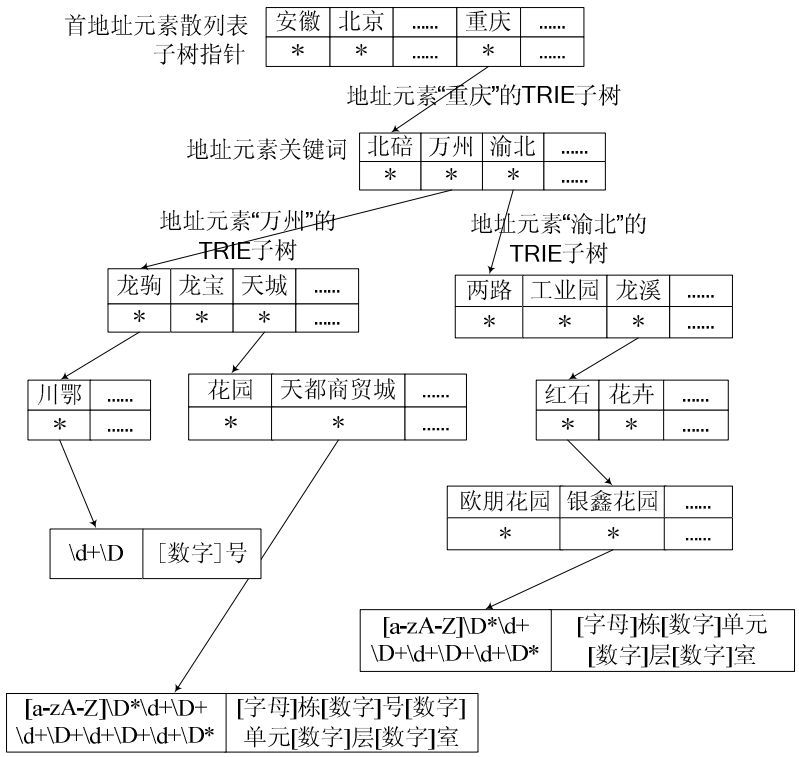


图 3.9 用于地址匹配的树形结构

Fig. 3.9 The Tree Structure for matching Address

TRIE树结构本身可以分成一个个小数组，适合预存到缓存中进行查找。从树的深度来讲，地址级别只存储到街道和建筑物一级，并且前缀相同的内容只存储一次，相比较与原有的存储方式，极大地减少了存储空间。在查询效率上，由于查找表的数据量锐减，其查询性能要有很大提升。根据对现有数据的抽样统计，地址的从最高一级到街道和建筑物级别不会超过5级，每一级别的地址元素不会超过40个。也就是对于每个完整地址的查询次数不会超过5次，而每次都查询都是基于一个元素小于40数组的查找，因此查询速度很快。最后要将匹配到门址利用正则表达式匹配，只是对字符串的处理，速度很快。

在实际的应用中，标准地址库在不断地进行完备，但是分级树地址库的数据量随标准地址库的增长变化不大，因为分级树地址库的大小取决于街道和建筑物的多少。当标准地址库的数据量增大时，分级树地址库的匹配方法性能影响不大。

3.3.3 算法流程

下面将在此分级地址库构建基础上对具体算法流程进行描述，如图3.10所示。

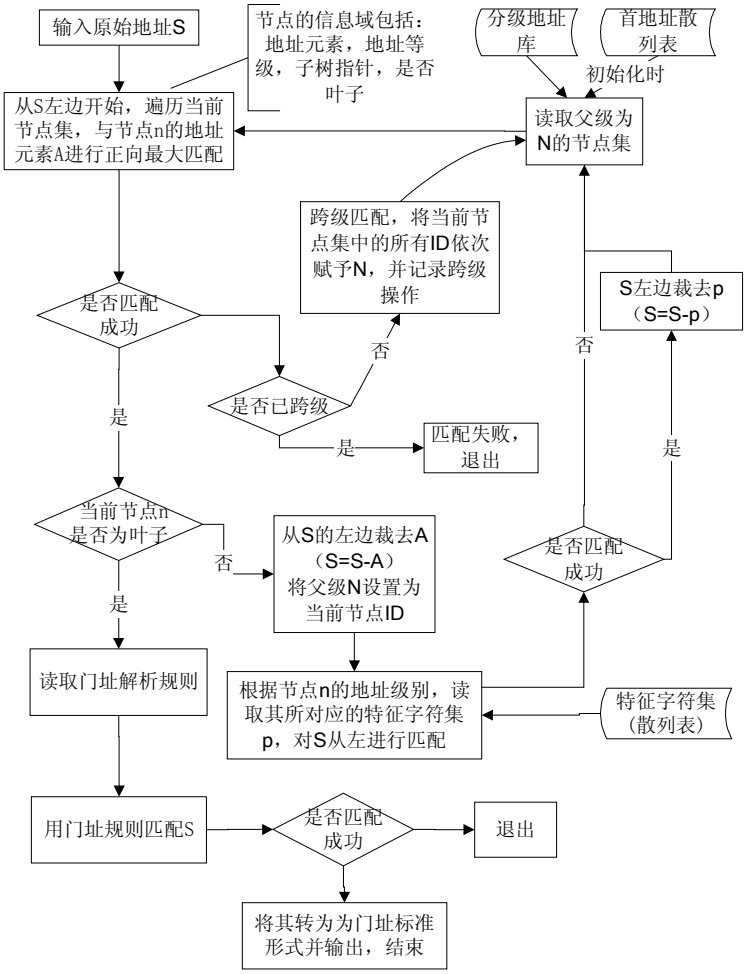


图 3.10 中文地址匹配流程图

Fig. 3.10 The Flow Chart for matching Chinese Address

输入：原始地址(用字符串S表示)

输出：标准地址

① 从分级树地址库读取父级节点标记为N的节点集(初始化时，加载省份所对应的数据，以哈希表形式存放)，遍历其节点集中的所有地址元素。根据遍历结果，对字符串S从左侧进行正向最大匹配。

② 若匹配成功，进行第③步；若匹配不成功，进行第④步。

③ 读取当前匹配到地址元素所对应的节点。从节点标志位中判断是否为叶子节点。若不是叶子节点，将进行第⑤步。若是叶子节点，进行第⑧步。

④ 跨级处理。将当前节点集中所有的节点ID依次赋予第①步中的父级N，循环进行第①步，若匹配成功，直接进入第③步；若遍历完节点集中所有节点仍然不能匹配成功，则不再进行深入匹配，直接退出，匹配失败。

⑤ 从字符串S的左边减去当前节点Node所对应的的地址元素A，即S=S-A。根据节点Node的地址级别，遍历其所对应的特征字符集（不同的地址级别所对应的特

征字符集不同），对裁减后S从左侧进行匹配，目的是为去除特征字符。

⑥ 若⑤中的特征字符匹配成功，从S左侧裁减去匹配到的特征字符P，即S=S-P，然后进行第⑦步。若匹配不成功，直接进行第⑦步。

⑦ 将当前节点的ID赋值第①步中的父级节点标记N，然后在执行第①步。

⑧ 将叶子节点所对应的门址解析规则提取出来，然后与字符串S进行正则表达式匹配。

⑨ 如果匹配成功，将其解析为标准门址信息，并加上前缀作为标准输出，结束；否则匹配失败，退出。

3.3.4 实验分析

实验数据来自某区县的部分标准地址数据10万条记录和不规范的地址数据1000条记录。利用标准地址数据生成分级树地址库只有321条记录，这个是由标准地址数据中所包含的不同街道或建筑物个数决定。

为了评判测试结果，引入两个测试指标：一个是查全率，其计算方法为：(匹配出的地址个数/所有的地址个数)*100%，另一个是匹配的时间消耗。下面设计几个实验，对基于标准地址库进行全字匹配方法和本文中描述的基于分级树地址库匹配两种方法效果进行比较。

实验① 从原始数据中，取出所有的其标准结构属于标准地址库(共有125条记录)，然后对其进行测试；

实验② 从原始数据中，取出所有的其标准结构属于分级树地址库(共215条记录)，然后对其进行测试。

实验③ 对所有这1000条原始地址数据进行测试。

实验结果如下表3.2和3.3中所示：

表3.2 两种方法的查全率对比

Tab. 3.2 The Comparison for Recall .

实验	基于标准地址库匹配(查全率)	基于分级树地址库匹配(查全率)
实验①	42.4%	82.4%
实验②	24.6%	80.4%
实验③	5.4%	17.3%

表3.3 两种方法耗时对比

Tab. 3.3 The Comparison for Time Cost

实验	基于标准地址库匹配(耗时)	基于分级树地址库匹配(耗时)
实验①	57s	48ms
实验②	83s	75ms
实验③	351s	328ms

以上数据单独来讲没有太大意义，因为其数值与地址数据质量有密切关系。但是通过横向比较，可以看出，在地址数据质量相同情况下，本文中的方法比基于标准地址库直接使用效果更好。

当然这种方法也有一定缺陷，对于存在错别字的地址数据进行规范化效果不好，另外通过原有标准地址生成匹配地址及匹配规则的过程相对比较麻烦，消耗时间较大。

3.4 本章小结

本章对于基于总线模型的数据清洗系统中的关键技术进行研究。研究了软件总线模型的特点，并将其应用于数据清洗系统；研究了对数据清洗任务进行分解，通过多线程技术并发执行，通过实验分析其有效性；最后对系统中具体实现的一种中文地址规范化方法进行研究，通过对比实验证明其有效性。

4 基于软件总线模型的数据清洗系统的设计与实现

4.1 系统描述与设计目标

一个典型的数据清洗场景是这样的：对于多个不同的数据源的数据进行清洗，从数据源读取脏数据，通过合适的数据清洗方法进行清洗，将清洗结果存入数据仓库[42]。由于数据源不同，针对不同的数据源，需要分别开发一套不同的数据清洗工具。

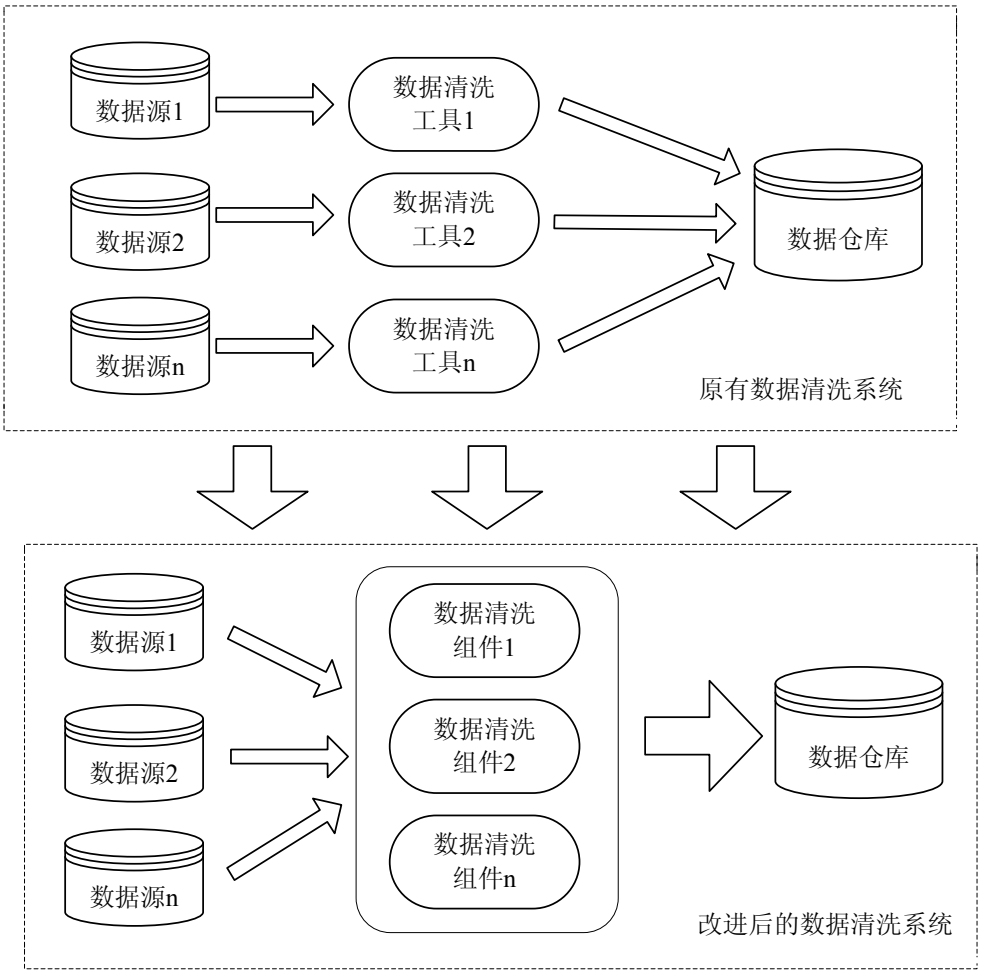


图 4.1 数据清洗系统对比图

Fig. 4.1 The Comparison between two Data clean Systems.

在这种传统数据清洗系统中，如上图 4.1 所示中的原有数据清洗系统，从系统的可扩展性及可维护性方面考虑，具有以下缺点:①如果需要添加新的数据源，又需要重新开发一套对应的数据清洗工具，对于现有的数据清洗工具不能进行重用。

②当对数据质量有了新的要求，则需要更新针对不同数据源的所有不同的数据清洗工具，这种方式显然不符合现代软件工程的要求。③虽然数据源不同，但这些不同格式的数据可能所表达的意义相同或相近，针对这些不同数据源开发的数据清洗工具从功能性有很多重复，增加不少重复工作。

考虑到以上提到的这些缺点，对于上述的系统进行改进，改进后的系统如上图4.1中所示。系统中对于原有分散的数据清洗工具进行集成，对于不同数据源提供统一调用接口，至于对不同的数据源如何选择数据清洗工具由系统来调度。从提高软件可复用性的角度，尽可能将数据清洗方法组件化，以使得整个系统变得更加灵活。

在以上对原有系统改进思路的基础上，本文尝试性提出的一种基于软件总线模型的数据清洗系统。系统的目的是要构建一个可复用可扩展的数据清洗系统，可以添加多种数据清洗组件和清洗来自不同数据源的脏数据。

4.2 系统体系结构设计

基于以上对于系统的描述，系统需要具有以下特点：①数据清洗系统应能够适用于不同的数据来源。②系统根据不同的数据来源，选择合适的数据清洗组件完成数据清洗任务。③系统应能够对多种不同的数据清洗组件进行集成。根据对于系统特点的分析，系统体系结构的设计如图4.2所示。

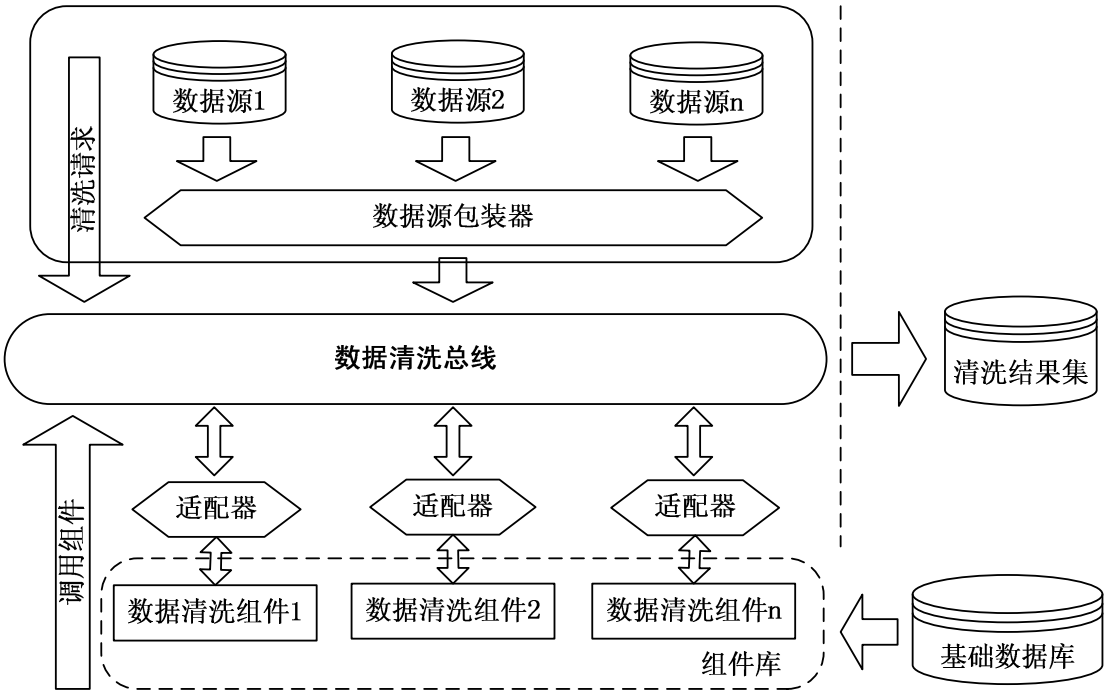


图 4.2 系统结构图

Fig. 4.2 The Architecture of System

系统主要有以下模块组成：数据清洗总线，数据源包装器，适配器，数据清洗组件。其中数据清洗总线是整个系统的核心，负责解析数据清洗请求，有选择地调用清洗组件的平台。数据源包装器负责将来自不同数据源的数据封装成符合数据清洗总线格式要求的报文格式。组件适配器位于数据清洗组件与数据清洗总线之间，负责总线与组件之间进行交互通信。数据清洗组件是具有相对独立的清洗功能的组成部分，其可以与整个清洗系统是完全松耦合的，由适配器负责与数据清洗总线进行集成。基础数据库存储一些基础数据信息，便于数据清洗组件使用。清洗结果集是对于经过系统完成清洗任务后暂存数据的集合，对于这些数据经过人工确认后最终进入数据仓库。

根据上述对于系统结构描述，从系统的功能模块的角度对系统整体的逻辑结构进行设计。在 Visual Studio 2005 开发环境中，一个解决方案往往有很多工程构成，将这些工程组合在一起完成整个系统。下面通过对系统中所包含的工程以及这些工程之间的关系来对系统的逻辑结构进行描述。

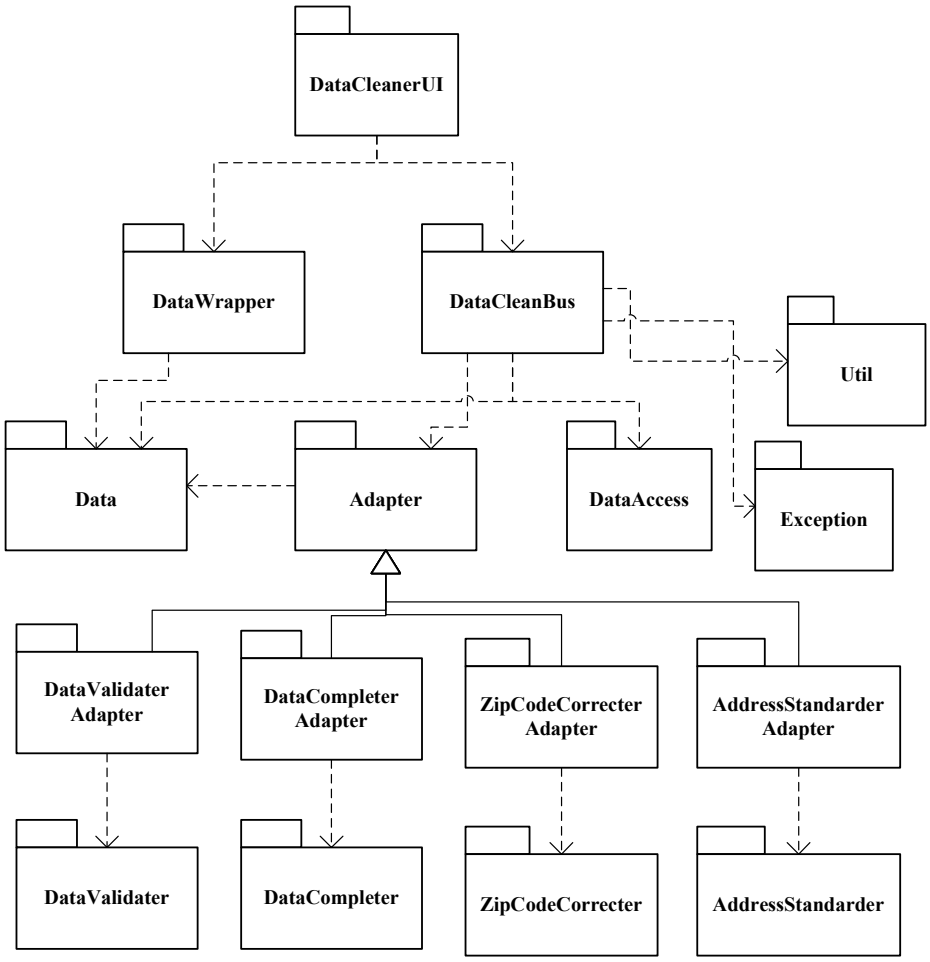


图 4.3 系统逻辑结构图

Fig. 4.3 Logical Diagram of System

在上图 4.3 中，DataCleanerUI 为系统的界面层，与使用者之间进行交换；DataCleanBus 为数据清洗总线工程，在其中定义数据清洗总线上的相关接口和具体操作；DataWrapper 为数据源包装器工程，在其中包括对不同数据源的访问以及数据转换的方法；Data 工程为待清洗的数据结构；Adapter 工程中对适配器接口的定义；DataValidaterAdapter，AddressStanderAdapter 等这些工程是具体的对 Adapter 的实现，在其中调用具体的数据清洗组件；DataValidater，DataCompleter，ZipCodeCorrecter，AddressStandarder 等这些工程为具体的数据清洗方法。在 Visual Studio 2005 开发环境中的各工程间的组织结构如下图 4.4 所示。

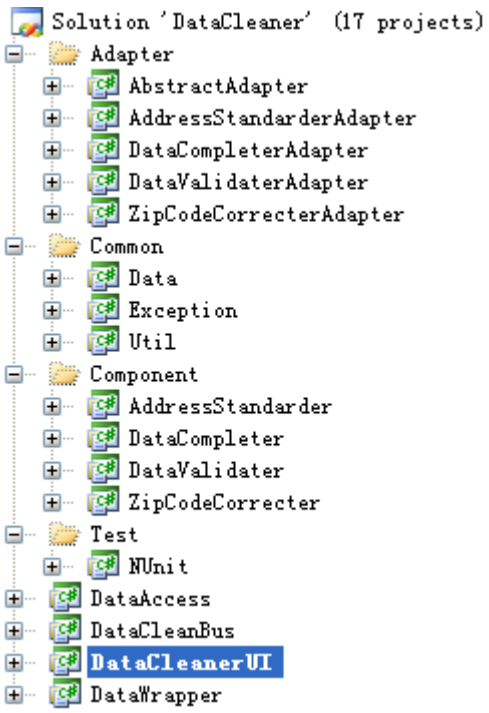


图 4.4 系统工程图

Fig. 4.4 Projects of System

在上述对于系统结构以及逻辑视图描述的基础上，对数据清洗的工作流程进行粗粒度地描述，如下图 4.5 所示。

① 数据源包装器将不同数据源的数据封装成数据清洗总线要求的格式，提交给数据清洗总线。

② 数据清洗总线根据待清洗数据中所包含的字段，从组件库查询合适的数据清洗组件。

③ 数据清洗总线对所查询到的组件依次进行调用。

④ 数据清洗组件调用其各自的数据清洗功能，对传入的数据进行清洗，将清洗结果返回给清洗总线。

⑤ 数据清洗总线在各组件依次调用完成后，将清洗结果暂时存入清洗结果集合，待领域专家确认后，转入目标数据库。

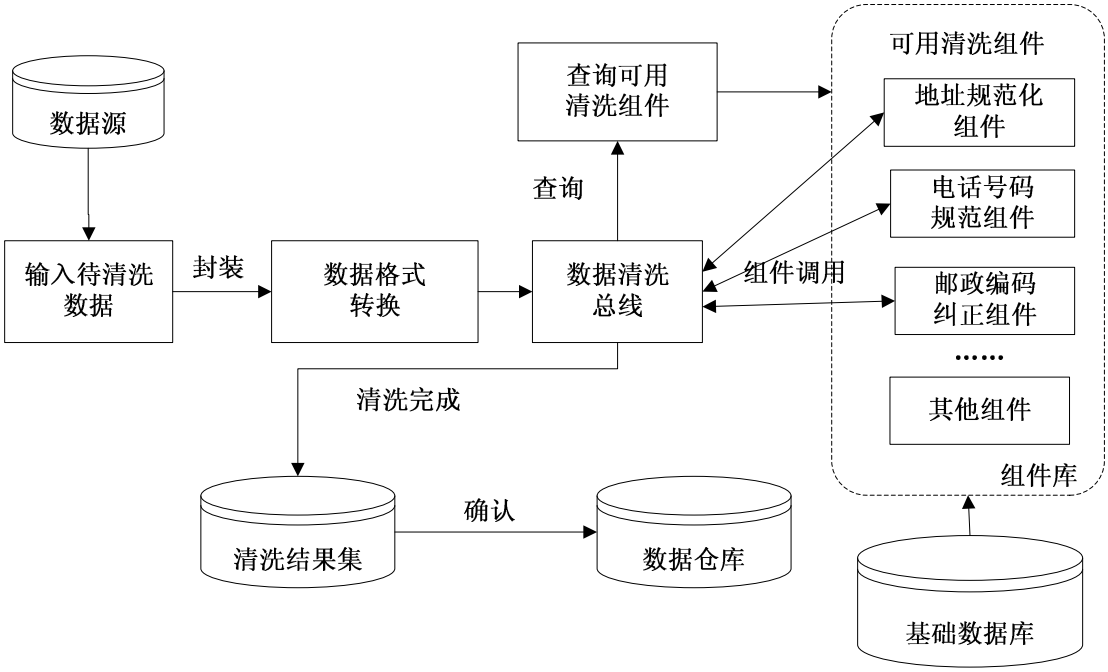


图 4.5 系统流程图

Fig. 4.5 The Flow Chart of System

4.3 数据清洗总线设计与实现

数据清洗总线是整个系统的核心部分，负责解析数据清洗请求，调度数据清洗组件。对于数据清洗总线来讲，其主要分为以下功能模块：①定义清洗总线相关协议。②对总线上挂接组件进行管理。③总线控制管理。下面从具体功能模块来对数据清洗总线进行设计和实现。

4.3.1 定义数据清洗总线协议

协议是一个抽象层次的概念，只要能达到通信双方能够实现正常通信，协议的实现方式有多种，可以是定义的接口或者规则。

① 定义组件集成协议。数据清洗总线需要集成不同的数据清洗组件，对于不同的清洗组件，其调用接口不同，只有定义出集成组件的协议，组件和总线之间才能正常通信。在本系统中，通过适配器其方式对不同的组件进行兼容，对于清洗总线来讲，适配器能够良好地完成通信交互任务，在对组件的调度是虽然是有总线发起，实质上是由适配器来完成。

在数据清洗总线中，定义一个适配器的抽象类 Adatper，对于不同的数据清洗组件适配器都需要从其抽象类继承，并实现其中的 CleanData 方法。Data 类是在清洗总线上定义的，每个组件所接受的统一数据格式。对于 CleanData 方法来讲，输

入的清洗方法的是脏数据，输出则是此清洗方法的完成清洗功能后的结果集。Data 类的具体属性值则需要根据实际系统中对清洗结果的要求来确定。

```
public abstract class Adapter
{
    abstract public Data CleanData(Data rawData);
}
```

从数据清洗组件角度考虑，如果希望挂接到数据清洗总线上来，就需要有其对应的适配器（adapter）实现上述接口。从数据清洗总线的角度，在调度数据清洗组件时，不需要考虑其具体实现，只需要调用清洗组件所对应的适配器类(adapter)中的 CleanData 方法。

② 定义清洗请求协议。对于任何系统，并不是对于任何的输入都接受。数据清洗总线也是一样，对于数据清洗总线来讲，需要制定可以接受的清洗请求的规范，这样才能对清洗请求正确解析。在本系统中，用 XML 来表示数据清洗请求，通过 XML Schema 来对清洗请求的格式进行定义。

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Request">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" name="cType" type="xs:string" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

在上述 XML Schema 描述中，cType 为数据清洗请求希望能够清洗总线能够进行实现数据清洗功能的数据字段。以系统中的一条清洗请求为例。

```
<?xml version="1.0" encoding="UTF-8"?>
<Request>
  <cType>address</cType>
  <cType>telephone</cType>
  <cType>zipcode</cType>
  ......
</Request>
```

其中是数据清洗请求，主要包括对中文地址，电话号码，邮政编码等这些客户信息数据的清洗，数据清洗总线对清洗请求解析后，会查询这些类型所对应数据

清洗组件，并将根据优先级串联起来，供清洗发起方调用。

4.3.2 数据清洗组件管理

在数据清洗总线上，清洗组件通过向总线注册方式进行挂接，同时清洗组件也可以进行更新或者从总线上卸载，这就需要对清洗组件进行管理。

对于总线上清洗组件的管理，需要在总线上保留组件集的信息，通过对保留信息的更新来实现对清洗组件进行管理。从数据清洗的角度和系统可维护性的角度考虑，系统中对组件集及组件的存储结构设计如下图 4.6 所示。

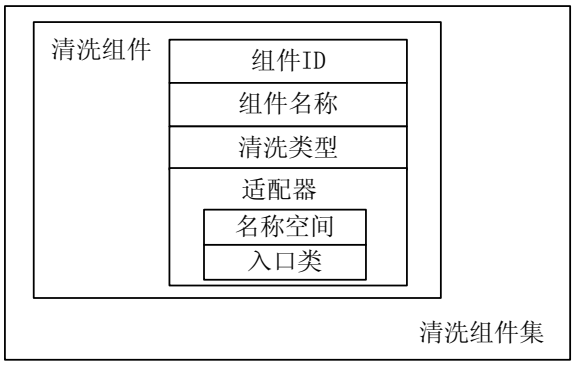


图 4.6 清洗组件结构图

Fig. 4.6 The Structure of Component

组件的描述对组件的成功复用至关重要，并且一个好的描述是有效检索与理解的基础[43]。本系统中采用 XML 文件对数据清洗组件进行管理，这种方式更加灵活方便，对于组件的注册，更新和卸载，只需要对 XML 中的节点进行修改就能实现。数据清洗总线通过解析 XML 文件，将对应的数据清洗组件加载到总线上来。下面是对组件集 XML 文件的 XML Schema 描述：

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Components">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Component">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="adapter">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="namespace" type="xs:string"/>
                    <xs:element name="className" type="xs:string"/>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
```

```
                </xs:sequence>
                <xs:attribute name="id" type="xs:unsignedShort" use="required"/>
                <xs:attribute name="priority" type="xs:unsignedByte" use="required"/>
                <xs:attribute name="ctype" type="xs:string" use="required"/>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

根节点为 Components(组件集)，每个组件对应一个子节点，每个组件需要包含组件本身的属性，并对组件进行编号。数据清洗组件由于功能不同和对所清洗的数据不同，其应该需要按照一定的顺序执行。这里就需要对数据清洗组件的优先级进行设置，以使得数据清洗总线对组件的调用时能够按顺序进行。不同的清洗组件所完成的清洗功能不同，因此需要用 cType 属性来对每个数据清洗组件的功能进行标记。

对于本系统中的软件总线模型，是通过适配器对具体的清洗组件进行调度，因此，每个组件节点中必须包含一个 adapter(适配器)，此 adapter(适配器)中需要包含其名称空间和对应的入口类名，并且此 adapter(适配器)需要是从清洗总线上定义的抽象适配器类继承而来。这点对于系统集成不同组件来讲尤其重要。

当系统需要更新，添加新的清洗组件时，维护开发人员需要编写新的清洗组件所对应的适配器。在 Components.xml 文件中添加新的 Component 节点，并添加此节点所对应的属性，将组件实体引用到工程中来；当需要更新某个清洗组件时，需要把此组件重新引用进系统中来，如果需要，则更新组件名，适配器类等信息；删除旧的清洗组件时，只需要把对应的 Components 节点删除和目录中对应的组件引用删除。

以系统中的中文地址数据规范化组件为例，如下 XML 代码中所示，对组件进行编号，其清洗字段为中文地址。AddressStandarderAdapter 为该清洗组件的适配器所对应的名称空间，ASComponentAdapter 为针对该清洗组件所实现的适配器类。从清洗总线的角度考虑，不需要具体关心其 ASComponentAdapter 类是如何实现，只需要知道它是实现了抽象的 Adapter 类即可。

```
<?xml version="1.0" encoding="utf-8" ?>
<!--清洗组件库-->
<Components>
  <Component id="1001" priority="1" ctype="address">
      <!--清洗组件适配器-->
      <adapter>
        <!--名称空间-->
```

```
        <namespace> AddressStandarderAdapter </namespace>
        <!--提供访问入口类-->
        <className> ASComponentAdapter </className>
    </adapter>
  </Component>

</Components>
```

4.3.3 数据清洗总线控制

在前文中已经对数据清洗总线协议制定和数据清洗组件的管理进行描述，这些归根到底只是一种静态结构，如何利用这些静态结构使得数据清洗总线能够正常工作，还需要数据清洗总线控制来实现。

在数据清洗控制中，需要实现如下功能，加载数据清洗组件，接受数据清洗请求，选择数据清洗组件，完成数据清洗任务等。根据对其功能的描述，对在数据清洗总线上定义的类图如下图 4.7 所示：

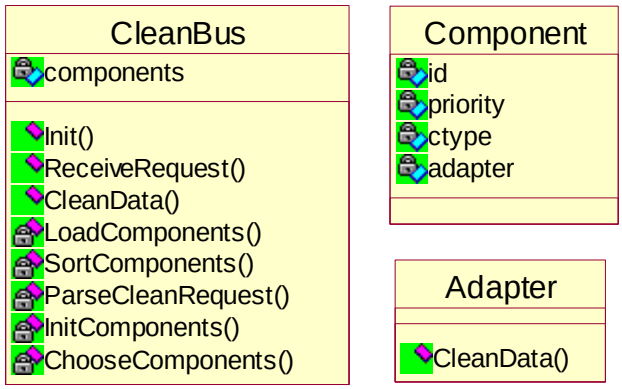


图 4.7 清洗总线类图

Fig. 4.7 The Class Diagram of Clean Bus

数据清洗总线控制从功能上来讲，是实现数据清洗组件的调度。其工作流程如下图 4.8 所示：

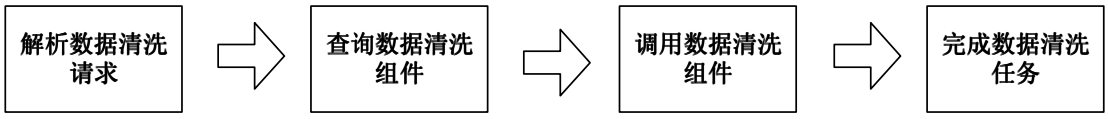


图 4.8 清洗总线流程图

Fig. 4.8 The Flow Chart for Clean Bus

① 解析数据清洗请求。解析数据清洗请求的目的在于通过请求理解客户程序中具体的清洗需求，以便根据请求选择合适的数据清洗组件。在前面已经对数据清洗请求的格式做了定义，从客户程序向数据清洗总线发送的数据清洗请求实质

上是一个 XML 报文，对于请求的解析其实是对 XML 报文的解析，从中待清洗数据的类型，并将其存入数组中。

```
//解析请求
XmlDocument xDoc = new XmlDocument();
xDoc.LoadXml(request);
XmlNodeList nodeList = xDoc.GetElementsByTagName("cType");
List<string> cleanTypes = new List<string>();
foreach(XmlNode node in nodeList)
{
    cleanTypes.Add(node.InnerText);
}
```

② 查询数据清洗组件。组件集的存储方式是以 XML 文件的形式，在其中有对组件信息描述和组件所对应的适配器类的描述，在系统中需要将这些静态的类映射为组件对象和适配器对象。

```
XmlNode adapterNode = node.ChildNodes.Item(0);
string strNameSpace = adapterNode.ChildNodes.Item(0).InnerText;
string strClassName = adapterNode.ChildNodes.Item(1).InnerText;
Assembly assembly = Assembly.Load(strNameSpace);
Type classType = assembly.GetType(strClassName);
adapter = (Adapter)Activator.CreateInstance(classType);
```

对于内存中组件的存储方式在 3.1.4 章节中已经进行了描述，下图在系统运行时，内存中的组件存储结构。

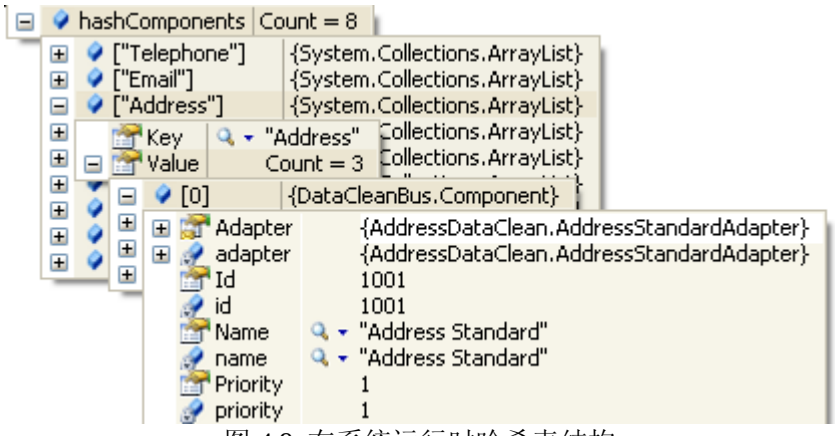


图 4.9 在系统运行时哈希表结构

Fig. 4.9 Structure of Hash Table during System Runtime

上图 4.9 中的的哈希表中，数据清洗字段为键值，哈希表中的值对应的是一个存放数据清洗组件的数组。系统从哈希表中查找其对应的组件集数组，并将其放

入可用组件集。

```
Hashtable hashComponents = cleanBus.getComponents();
//根据清洗请求中的待清洗数据字段选择数据清洗组件
foreach (string ctype in cleanTypes)
{
    List<Component> components = hashComponents[ctype];
    foreach (Component component in components)
    {
        availableComponents.add(component);
    }
}
```

③ 调用数据清洗组件。在系统中，由于挂接到数据清洗总线上来的组件都对应一个适配器，而此适配器又是从总线中所定义的抽象适配器继承而来，对于查询到的可用组件集中的数据清洗组件调用，其实就是对适配器清洗方法调用，然后将其串联起来，完成一系列数据请求任务。

```
foreach (Component component in availableComponents)
{
    //根据Adapter调用具体的数据清洗方法
    data = component.Adapter.CleanData(data);
}
```

如下图 4.10 所示，图中箭头表示数据流的方向。待清洗的数据，经过数据清洗组件集依次进行清洗，对于不同的清洗组件，只负责其中某些字段的清洗，最后将清洗的结果汇入清洗结果集，清洗结果集临时存储数据，通过领域专家确认后存入数据仓库。

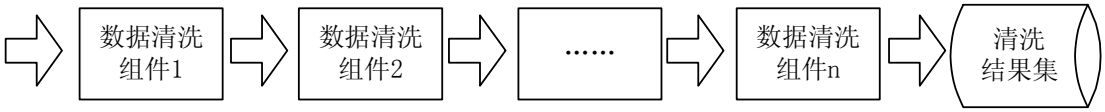


图 4.10 组件调度流程图

Fig. 4.10 The Flow Chart for Component call

为了更好的对数据清洗过程进行描述，将系统运行中的在调用数据清洗组件执行清洗任务之前，和调用数据清洗组件执行清洗任务后的数据进行对比，如下图所示。图中通过调用数据清洗总线上的数据清洗组件为对一条客户数据的清洗前后对比。由于客户数据涉及隐私，这里用测试数据来描述。

清洗前的数据

清洗后的数据

图 4.11 数据对比图

Fig. 4.11 Comparison between two Data

在上图 4.11 中的清洗后的数据中，通过 ExceptionMessage 字段对待清洗数据中存在的问题进行描述。在对这条测试数据的数据清洗过程中，调用了地址规范化组件，电子邮件有效性验证组件，邮政编码自动补全组件，电话号码验证组件来执行数据清洗任务等。

4.4 数据源包装器设计与实现

数据源不同，造成待清洗数据的数据格式也不同。数据来源可以是 Oracle，SQL Server，Access 等不同的关系型数据库。另外不同数据源的数据库，表名，字段名等也不统一[44]。如果对这些数据通过本文的数据清洗系统进行清洗，就需要对每个数据源进行包装，以使得其能够满足之前规定的总线协议。

从提高系统可复用性和可扩展性的角度出发，系统中所实现的数据源包装器，对不同来源的数据格式进行适配，将不同数据源的数据统一转化为数据清洗总线能够清洗的数据格式，提交给数据清洗总线完成清洗任务。

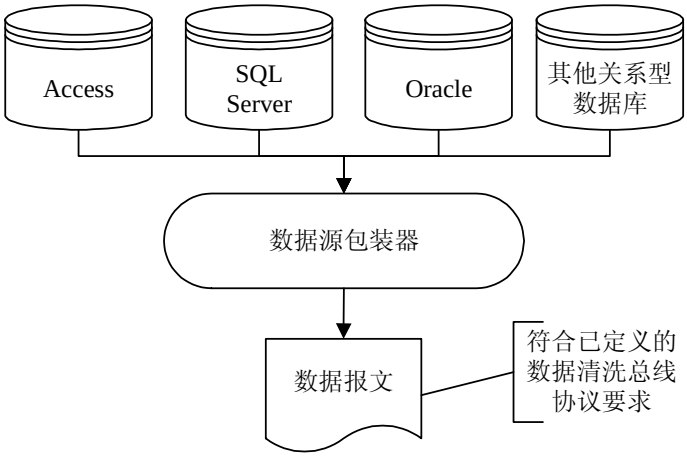


图 4.12 数据源包装器结构图

Fig. 4.12 The Structure for Data Source Wrapper

上图 4.12 为数据源包装器和多个数据源之间的关系，数据源包装器将各个数据来源的数据转化为符合数据清洗总线协议要求的数据报文格式。

4.4.1 数据源管理

数据源不同，其数据访问方式也会有差异，系统中需要通过保留数据源信息。在系统中，主要保存以下数据源配置信息：① 数据源类型，根据不同的数据源类型选择不同的数据库连接方式。② 数据库连接信息，包括数据库连接语句，数据库驱动以及数据库登录信息。③ 数据库元数据，数据表信息，字段名，字段类型以及数据库字段与待清洗字段之间的映射关系，方便对数据源的数据进行转化，以符合数据清洗总线清洗的要求。其结构如下图 4.13 所示。

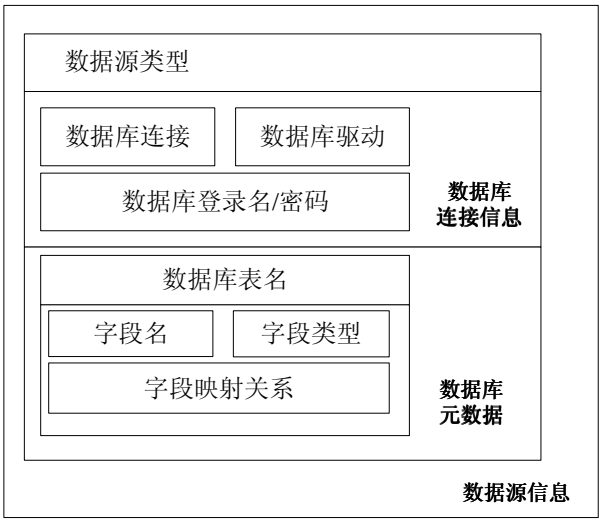


图 4.13 数据源信息

Fig. 4.13 The Information of Data Source

在系统中，通过 XML 文件保存以上这些配置信息，以屏蔽不同数据库元数据之间的差异。XML 的具体格式如下所示。根节点 DataSources 下面应该包含一个或者多个不同数据源 DataSource 节点。DataSource 节点中包含属性数据源名称，DbType 表示数据库的类型，这里实现了 Access，Oracle 和 SQL Server 三种。Connection 节点包含数据库连接语句，数据库驱动以及登录验证信息。一个数据库包含一个或者多个 Table 节点，Table 节点表示数据库表信息，其子节点应该有一个或者多个表示数据库字段信息，MapCleanType 节点表示数据库字段与待清洗字段的映射关系。

```
<?xml version="1.0" encoding="utf-8" ?>
<DataSources>
  <DataSource Name="dsName">
    <DbType>Access|Oracle|SQL Server</DbType>
    <Connection>
      <URL>...</URL>
    </Connection>
    <Table Name="...">
      <Column ColumnName="..." ColumnType="...">
        <MapCleanType>...</MapCleanType>
      </Column>
    </Table>
  </DataSource>
</DataSources>
```

从用户使用方便的角度，配置数据源是通过配置向导的方式完成，其配置流程如下图 4.14 所示。上述的 XML 文件是配置向导完成后自动生成。对于数据源包装器通过数据清洗总线执行清洗任务时，则需要将上述的 XML 文件进行解析。

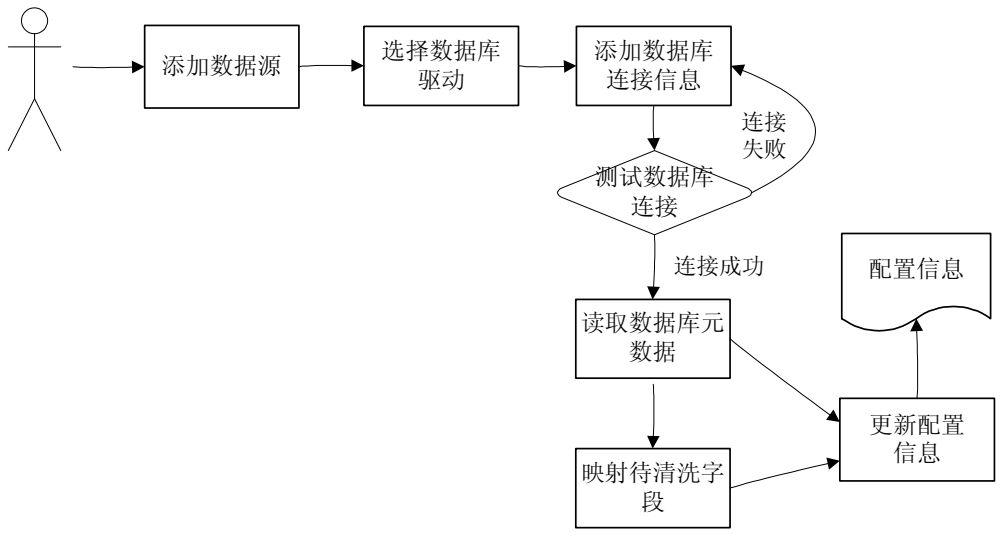


图 4.14 配置数据源流程

Fig. 4.14 The Flow Chart for Configure Data Source

4.4.2 元数据获取

数据库的元数据是指数据库的结构信息，包括数据库表属性、列属性，主键，数据类型等的描述。当有新的数据库添加进来时，数据源包装器就需要提取其元数据，并将其保存至上述的 XML 文件中，以便于以后对特定字段进行数据清洗。

由于数据源的数据库类型不同，对于不同的数据库其读取元数据的方式也有所差异，在数据库中基本上都有数据库系统表，可以通过读取数据库系统表的数据获取数据库元数据，可以通过读取系统表来获取[44]。在 ADO.net 中，其提供了获取元数据的方法可以屏蔽不同数据源之间的差异。系统中通过 ADO.net 来获取不同数据源的数据库元数据结构。在 DBConnection 中，通过 GetSchema 方法，可以变换其参数，来获取数据库表，字段的属性[45]。在其参数 Restrictions 中可以限制条件，以获取满足要求的元数据属性。将获取到的元数据保存到 XML 文件中。

```
string[] res = new string[4];
for (int i = 0; i < 4; i++) res[i] = null;
DataTable dt = conn.GetSchema("Table", res);
foreach(DataRow dr in dt.Rows)
{
    string tableName = (string)dr["table_name"];
    ......
    res[2] = tableName;
    DataTable colTable = conn.GetSchema("Table", res);
    foreach(DataRow colDr in dt.Rows)
    {
        string colName = (string)colDr["column_name"];
        //更新元数据配置信息
        ......
    }
}
```

在系统中，根据面向对象的思想，需要将 XML 文件中的属性映射为实体对象，需要在系统中对数据库元数据各个类进行定义，如下图 4.15 所示：

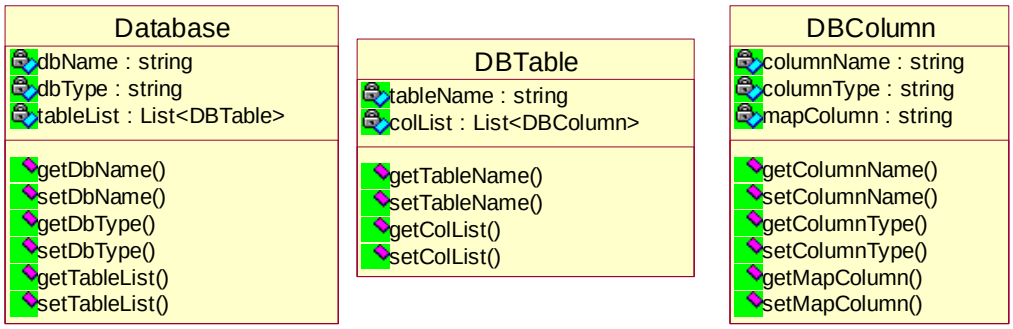


图 4.15 数据库元数据类图

Fig. 4.15 The Class Diagram for Metadata of Database

4.4.3 数据查询和转化

① 数据查询

不同类型的数据库其连接访问方式，从面向对象设计出发，通过统一接口访问，以适应不同的数据源，采用抽象工厂的模式[46]对不同数据库的连接，数据查询进行封装。

DBFactory 为抽象工厂，其是一个抽象类，其中声明创建数据库连接，创建数据库查询等方法。每种数据库类型对应一个工厂类。工厂类 SQLServerFactory 为 SQL Server 数据库所对应的工厂类，AccessFactory 为 Access 数据库所对应的工厂类， OracleDBFactory 为 Oracle 数据库所对应的工厂类，他们各自实现各种数据库的连接，查询数据。在系统中实现了这三个经常用到的数据库所对应的三个工厂类，对于新的数据库，需要从 DBFactory 继承，实现工厂类。其类图具体描述如下图 4.16 中所示。

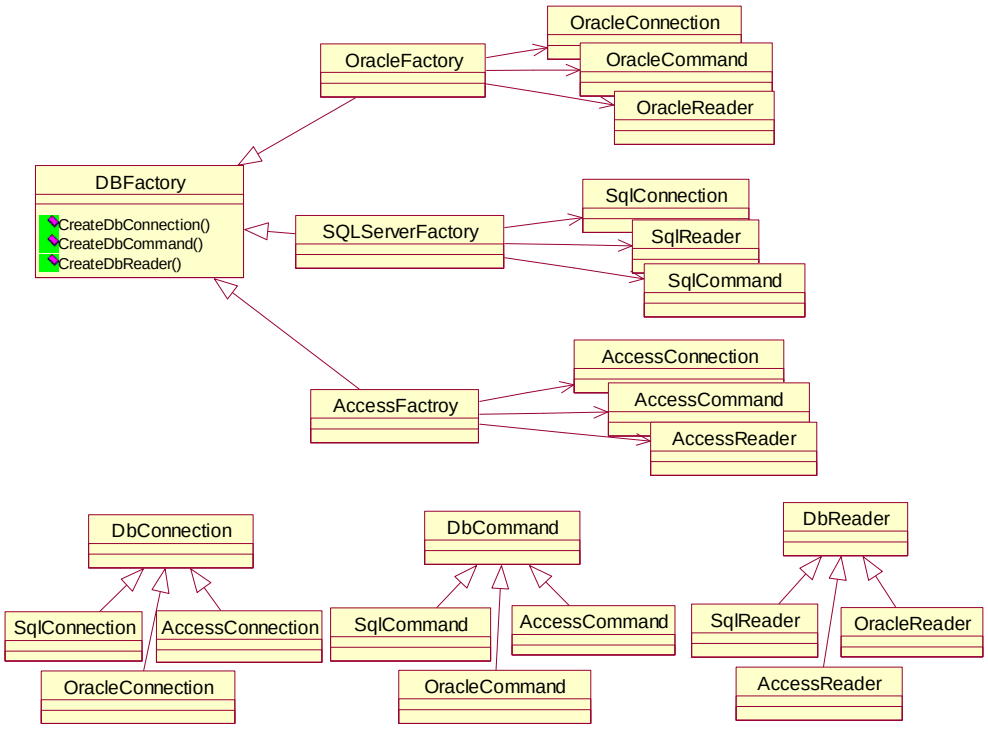


图 4.16 数据库操作类图

Fig. 4.16The Class Diagram for Data Operator

对于数据的清洗，其数据量往往很大，如果利用单线程进行处理，耗时很长，因此需要对数据清洗任务进行分解。在 3.2 章节中，已经对数据清洗任务分解方式和多线程技术做了研究，对于数据清洗任务的分解在系统中是通过对原始 SQL 语

句进行分解来实现。

下面以 Oracle 数据库为例，描述数据清洗任务分解的方法。对于数据表 Customer 的查询，如下 SELECT * FROM Customer。在 Oracle 数据库中，对于数据分页可以通过如下 SQL 语句实现：SELECT * FROM (SELECT A.*, ROWNUM RN　FROM (原始 SQL 语句) A) WHERE RN BETWEEN [Start] AND [End]。在系统中开启的用于数据清洗的线程数目为 15，即数据集待分解的个数。

```
1）  SELECT * FROM (SELECT A.*, ROWNUM RN　FROM (SELECT * FROM Customer) A)
     WHERE RN BETWEEN 1 AND 1000;
2）  SELECT * FROM (SELECT A.*, ROWNUM RN　FROM (SELECT * FROM Customer) A)
     WHERE RN BETWEEN 1001 AND 2000;
……
15)     SELECT * FROM (SELECT A.*, ROWNUM RN　FROM (SELECT * FROM Customer)
     A) WHERE RN BETWEEN 14000 AND 15000;
```

首先通过 SELECT COUNT(*) FROM Customer 获取到所有待清洗数据记录条数 M，然后根据待分解的个数，计算出每个数据集的起始值和结束值，拼出每个数据集所对应的 SQL 语句。最后启动 15 个线程来对每个数据集进行查询，并提交给数据清洗总线进行清洗。

② 数据转化

前文中讲到，数据源不同造成数据库的结构不同，字段名称也不相同，对于数据源包装器来讲，需要将这些字段进行统一，即统一为数据清洗总线所能识别的清洗类型，这里就需要对数据进行转化，其流程如下图 4.17 中所示。

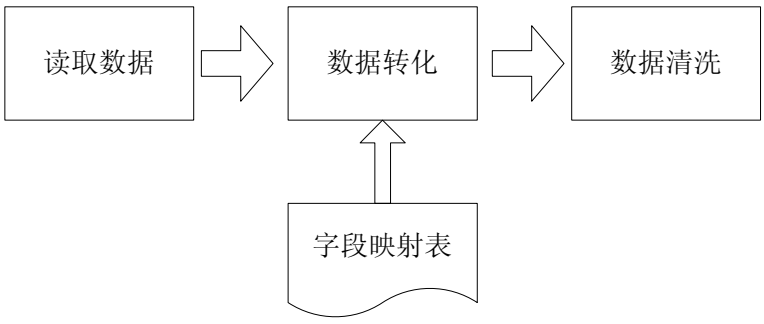


图 4.17 数据转化流程图

Fig. 4.17 The Flow Chart for Data convert

在系统中对数据库元数据的字段信息中，保存 MapCleanType 所对应数据，将原始数据库的字段映射为待清洗的字段名称。待清洗的字段是指系统最终所需要

的数据字段。从数据库读取出的数据，根据这种映射关系进行相应的转化，之后提交给数据清洗总线系统，完成清洗任务。其中字段映射表逻辑结构如下图 4.18 所示。

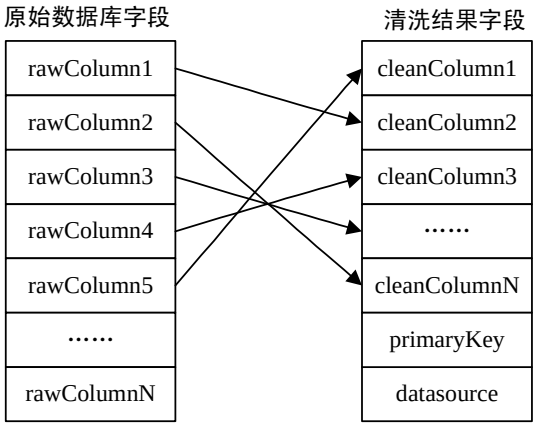


图 4.18 字段映射关系

Fig. 4.18 The Mapping for Column

4.5 适配器设计与实现

在前文中已经提及，数据清洗总线与数据清洗组件之间的接口不能相互兼容，为了使二者之间能正常通信，引入适配器(Adapter)。在系统中，对适配器的实现采用设计模式中的适配器模式思想进行设计。

适配器(Adapter)模式一种经常用到的设计模式，其目的是将一个类的接口转换成客户希望的另外一个接口,使得原本由于接口不兼容而不能一起工作的那些类可以一起工作[46]。在适配器模式的运用中，一般需要定义一个包装类，包装有不兼容接口的对象。这个包装类指的就是适配器（Adapter），它包装的对象就是适配者(Adaptee)。适配器提供客户类需要的接口，适配器接口的实现是把客户程序的请求转化为对适配者的相应接口的调用。换句话说：当客户类调用适配器的方法时，在适配器类的内部调用适配者类的方法，这个过程对客户程序是透明的，客户程序并不直接访问适配者。结合本系统，客户程序是指数据清洗总线，适配者是指数据清洗组件，即在适配器内部调用数据清洗组件的相关数据清洗方法，完成数据清洗任务，而对于数据清洗总线来讲是完全透明的。因此，适配器可以使由于借口不兼容而不能交互的类可以一起工作。

适配器模式总体上可以分为两类：类适配器和对象适配器。类适配器是通过继承类适配者类实现的，另外类适配器实现客户类所需要的接口。当客户对象调用适配器类方法的时候，适配器内部调用它所继承的适配者的方法。对象适配器包含一个适配器者的引用，与类适配器相同，对象适配器也实现了客户类需要的接

口。当客户对象调用对象适配器的方法的时候，对象适配器调它所包含的适配器者实例的适当方法。

在系统中，数据清洗组件从逻辑上讲是对立与整个系统的，并且有可能是第三方开发，只提供调用接口，因此不适用于对其进行继承，从这个角度考虑，采用对象适配器模式更符合系统的特点，下图 4.19 为本系统的适配器类抽象图。

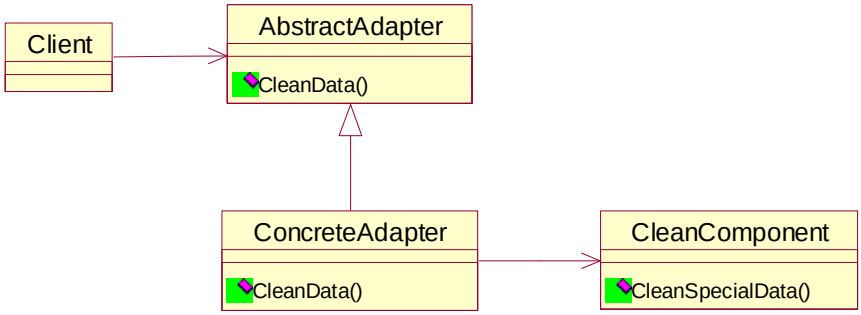


图 4.19 适配器抽象类图

Fig. 4.19 The Class Diagram for Adapter

上图中的 Client 表示向清洗组件发起清洗请求的客户程序，在系统中是指数据清洗总线。AbstractAdapter 是具体适配器的上层抽象类，在数据清洗总线中定义，其中声明 CleanData 方法。ConcreteAdapter 类是实现具体的适配器，负责对具体某个清洗组件的适配，在 CleanData 方法中调用数据清洗组件，实现具体的数据清洗功能，其实现方法与具体数据清洗组件有关。在系统中，结合已有的数据清洗组件，实现对应的组件适配器，如下图 4.20 所示。

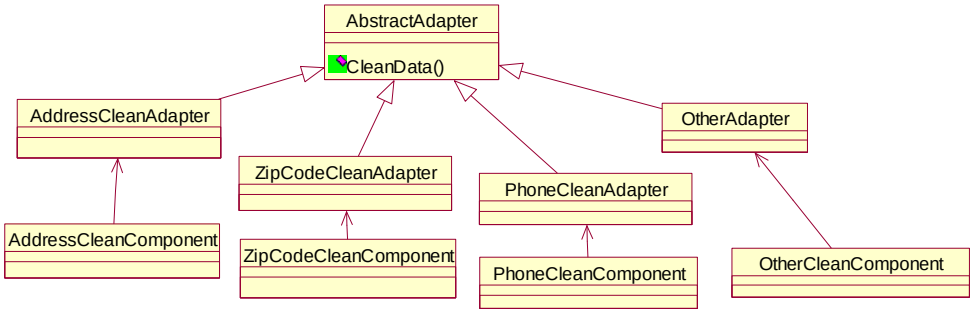


图 4.20 适配器具体类图

Fig. 4.20 The Class Diagram for Concrete Adapter

4.6 数据清洗组件设计与实现

数据清洗组件可以是独立于整个系统的模块，其本身可以是针对某一领域的数据开发的，完成某种清洗任务的独立模块，其应该具有可重用性。这里的组件是

抽象意义上的概念，大到可以是一个独立系统，小到可以是一个 Web 服务或者动态链接库，甚至是一个具有清洗功能的模块，这些是根据领域需求来确定。

本系统中实现了的几种具体针对客户数据的清洗方法，包括中文地址规范化组件，邮政编码纠正组件，另外还有数据有效性验证类组件，数据自动补全类组件等等，其功能相对独立，并将其开发成动态链接库，容易得到复用。

4.6.1 中文地址规范化组件

由于数据来源不同，输入习惯问题等因素，中文地址数据表示形式不规范，本组件的主要任务是将各种不规范的地址数据，转化为标准地址格式，如下图 4.21 所示。在第 3.3 节中已经对中文数据清洗方法做了研究，本组件在以上研究的基础上设计并实现。

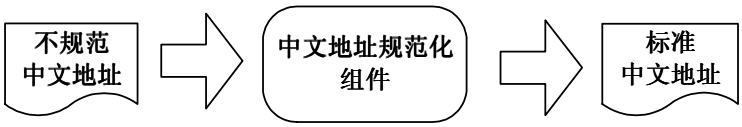


图 4.21 中文地址规范化流程图

Fig. 4.21 The Flow Chart for Chinese Address Standard

在前文研究的基础上，对于匹配地址库进行设计，这里需要两个核心表，如下表所示。MatchAddressTable 表是用于存储地址元素之间的树形结构表，如表 4.1 所示，MatchRuleTable 表用于存储门址解析规则，如表 4.2 所示。对于其中字段的具体说明已经在表中进行描述。

表4.1 地址数据表

Tab. 4.1 The Data Table of Address

Column	Type	Description
ID	Int	主键
AddressElement	Varchar	地址元素
Suffix	Varchar	特征字符
Alias	Varchar	地址别名
ParentId	Int	上一级地址 ID
IsLeaf	Bit	是否为叶子节点
RuleId	Int	指向门址解析表

表 4.2 门址解析规则表

Tab. 4.2 The Data Table Of Rules

Column	Type	Description
ID	Int	主键
Regular	Varchar	门址解析规则
Standard	Varchar	门址标准表示形式

在上述数据表的基础上，实现对中文地址匹配的具体实现进行描述：

① 输入原始地址数据，以字符串形式表示。为方便快速查询，将 ParentId 为空的地址元素，即省份地址元素预存到缓存区数组中。从输入原始地址起始字符与该数组元素进行匹配比较。

② 将其赋予当前地址元素对象。用原始地址数据赋给临时变量，对临时变量从起始位置裁减去匹配到的内容。根据当前地址元素 ID，读取其直接下级的地址元素集合，遍历集合中的元素，跟上述临时变量起始字符开始比较匹配。如果匹配成功，循环执行②，直到叶子节点不为空，跳出循环。如果匹配失败，算法结束。

```
List<Address> addressList = QueryAddress(currentAddress.Id);
foreach (Address address in addressList)
{
    //与原始地址进行匹配
    if (tempAddress.StartsWith(address.AddressElement))
    {
        tempAddress = subtract(tempAddress,
        address.AddressElement + address.Suffix);
        ……
        currentAddress = address;
        break;
      }
}
```

③ 如果上述匹配不成功，进行跨级匹配，即遍历所有子级地址元素的子级地地址元素。通过嵌套循环方式，来遍历两级包含的所有地址元素，跟上述临时变量起始字符开始比较匹配。如果匹配成功，进入②；如果匹配失败，算法结束。

④ 根据当前地址元素所对应的规则表 RuleId，查询其对应的门址解析规则，将上述临时变量剩下的所有字符与规则，通过正则表达式进行匹配。

```
Rule rule = GetRule(currentAddress.RuleId);
Regex regex = new Regex(rule.Regular);
if (regex.IsMatch(tempAddress))
{
    //匹配成功将其转化为标准地址
    standarAddress = StandarAddress(tempAddress, rule.Standard);
}
```

为了对具体匹配规则进行说明，下图 4.22 系统运行时截图中，Rule 从数据库读取出的匹配规则和标准地址形式。

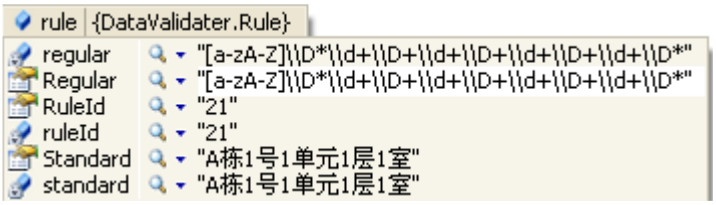


图 4.22 Rule 对象运行时状态

Fig. 4.22 The State for Rule Object during Runtime

⑤ 如果匹配成功，将原始门址解析为标准地址形式，算法结束。如果匹配失败，算法结束。

4.6.2 邮政编码纠正组件

数据一致性问题是数据清洗领域中的经常遇到的问题，本组件实现的功能是确定地址数据与邮政编码之间一致性检查，对于缺失的邮政编码，通过地址数据进行补全。

针对邮政编码纠正组件，其输入为地址数据和邮政编码，输出为经过纠正的邮政编码，在对邮政编码的纠正中考虑以下几种情况：① 邮政编码为空，根据地址数据查询邮政编码，将查询结果作为输出。② 地址数据为空，验证输入邮政编码格式，正确直接将原有邮政编码输出，不正确抛出格式错误异常。③ 邮政编码格式不正确，根据地址数据查询邮政编码，将查询结果作为输出。④ 邮政编码格式正确，通过邮编查询地址数据。如果输入地址数据结构简单，即长度短于查询出的地址数据，说明输入地址不具备参考性，则将原邮编输出；输入地址长度大于查询出地址的长度，判断查询出地址与输入的地址是否一致，否则抛出数据不一致性异常。下面是邮政编码纠正组件的具体工作流程图，如下图 4.23 所示。

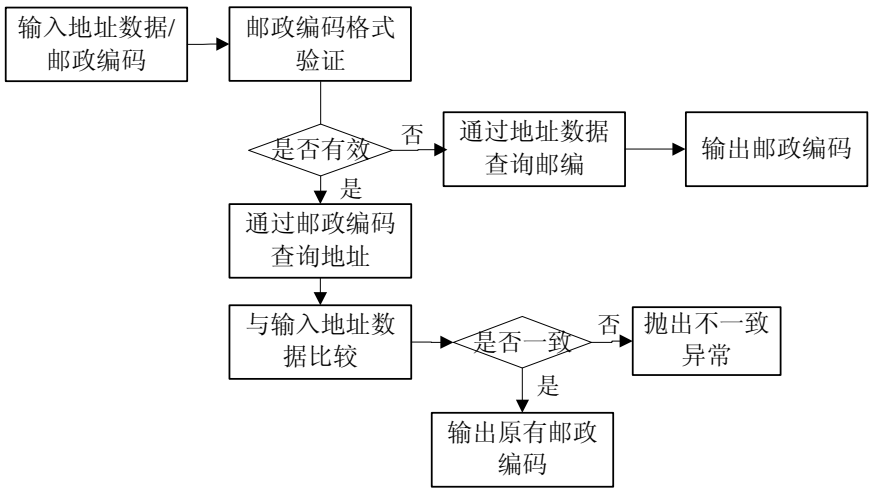


图 4.23 邮政编码纠正流程图

Fig. 4.23 The Flow Chart for Zip Code Correcting

在前面对具体工作流程的基础上，下面对实现邮政编码纠正组件的部分关键代

码描述。

① 邮政编码格式验证。在系统中，主要是通过正则表达式对邮政编码格式进行验证。

```
Regex regex = new Regex(@"[1-9]\d{5}(?!\d)");
if (regex.IsMatch(zipcode)) return true;
```

② 输入地址与查询出的地址一致性比较。除去地址中表示地址级别的特征字符，以新查询出的地址为基准，通过逆向最大匹配方法，对两个地址进行匹配，比较出在输入地址中和查询出的地址中重合的长度。

```
int length = newAddress.Length;
char newChar = newAddress[length - 1];
if (rawAddress.IndexOf(newChar) != -1)
{
    flag = rawAddress.IndexOf(newChar);
    isMatch = true;
}
if (isMatch)
{
    for (int i = length - 2; i > 0; i--)
    {
        flag--;
        count++;//两地址中重合的长度
        char newChar = newAddress[i];
        char oldChar = rawAddress[flag];
        if (newChar != oldChar) break;
    }
}
```

4.6.3 其他数据清洗组件

在系统中还针对客户数据，实现了数据有效性验证类组件，数据自动补全类组件等数据清洗组件，这些组件本身是可以独立于系统单独使用的，即其具有很强的可复用性，这也是本系统架构的优点所在。

① 数据有效性验证类组件。数据有效性验证主要是对数据格式的检查，这里的数据有效性验证组件主要对客户信息的某些字段进行验证，其中主要包括中文姓名验证，电话号码验证，手机号码验证，电子邮件验证，身份证号码验证，时间格式验证等。

表 4.3 数据有效性验证类组件

Tab. 4.3 Components for Validating Data

序号	组件名	输入	输出	简单描述
1	ChineseNameValidater	客户中文名	是否有效	对客户中文名称验证
2	TelephoneValidater	座机号码	是否有效	对于座机号码验证
3	MobileValidater	手机号码	是否有效	对手机号码验证
4	ZipCodeValidater	邮政编码	是否有效	对邮政编码验证
5	IDCardValidater	身份证号码	是否有效	身份证号码验证
6	DataTimeValidater	时间	是否有效	对时间格式进行验证
7	EmailValidater	电子邮件	是否有效	电子邮件验证

② 数据自动补全类组件。数据自动补全是根据数据之间的关系，对于一些缺失的数据字段进行补全。在这里数据自动补全类组件主要包括根据身份证号码补全生日，根据客户地址补全邮政编码等。

表4.4 数据自动补全类组件

Tab. 4.4 Components for Completing Data

序号	组件名	输入	输出	简单描述
1	BirthdayCompleter	身份证号码	生日	通过身份证号码生成生日
2	ZipCodeCompleter	邮政地址	邮政编码	通过邮政地址查询邮编

4.7 系统运行结果

前文中已经对系统各组成结构的实现做了详细描述，在本节中将通过系统中的部分截图对系统的运行结果进行描述。

① 添加新的待清洗的数据源。如下图 4.24 所示，为系统中添加数据源的截图。在系统中通过配置向导的方式来完成对新的数据源的添加，其具体步骤分为以下几步：

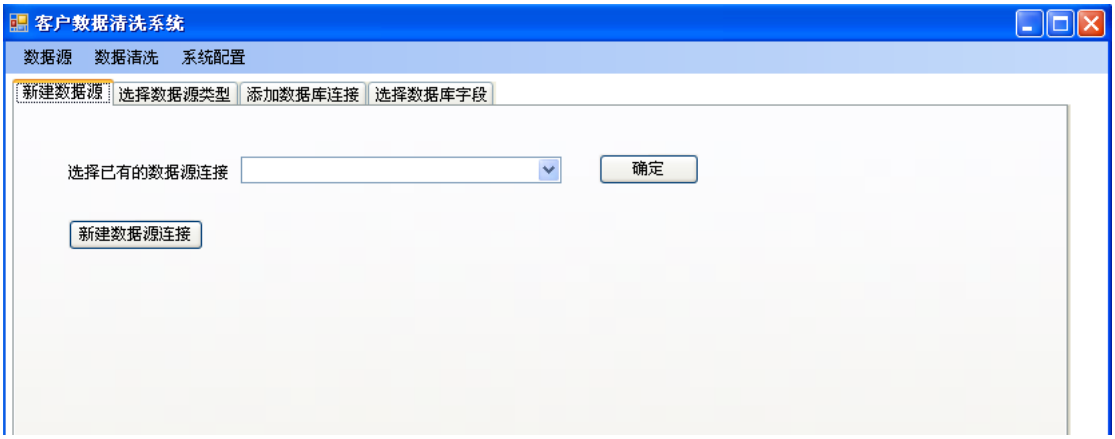


图 4.24 添加数据源

Fig. 4.24 Add Data Source

1) 选择数据库类型，因为不同的数据库类型，其数据源的配置方式有差异。系统中提供的主要有 SQL Server 数据库，Access 数据库和 Oracle 数据库三种的数据源的配置。如下图 4.25 所示。在这里以 SQL Server 数据库为例进行说明。

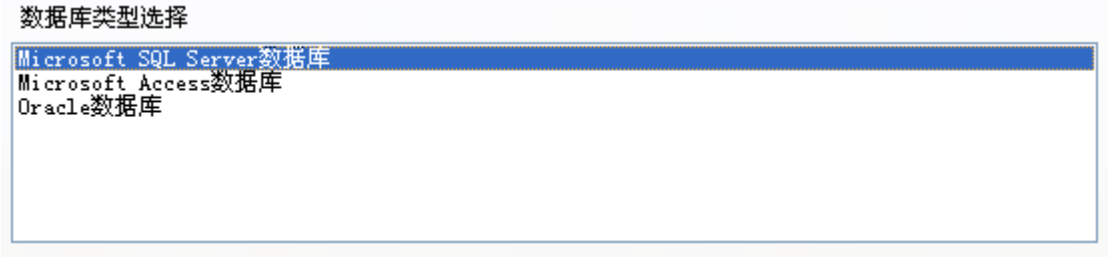


图 4.25 选择数据库类型

Fig. 4.25 Choose Database Type

2) 填写数据库连接信息，如下图 4.26 所示，主要包括数据库地址，数据库名称，数据库登录验证信息等，测试连接成功后进行下一步读取数据库元数据。

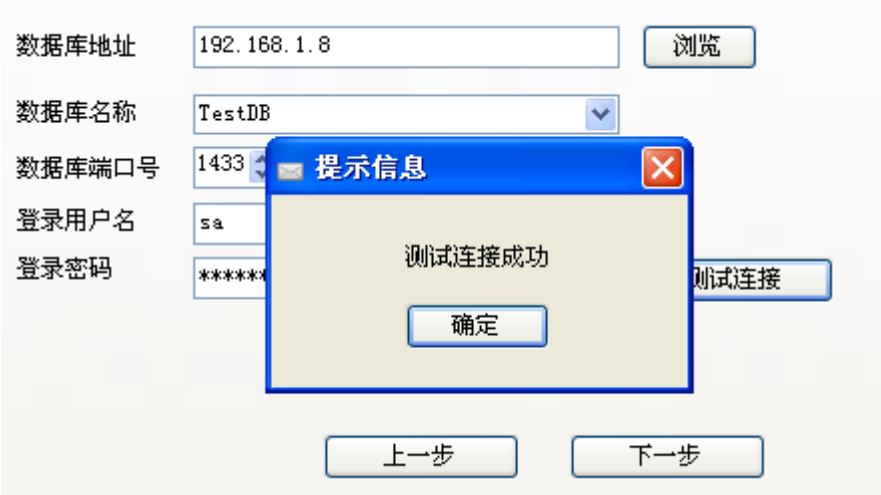


图 4.26 填写数据库连接信息

Fig. 4.26 Fill Data Connection Information

3）建立原始数据库中的字段和清洗结果集的字段间的映射关系，如下图 4.27 所示，以便对待清洗的数据格式进行转换。

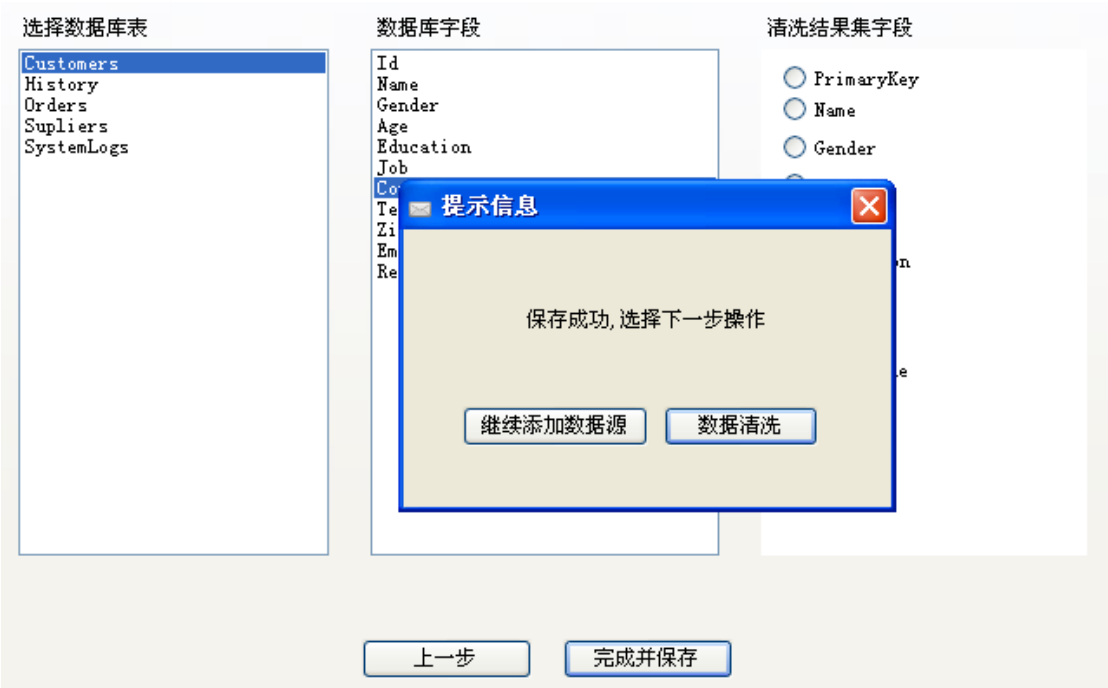


图 4.27 字段选择

Fig. 4.27 Choose Columns

② 对于清洗方法的选择。系统中将具体的数据清洗方法组件化，通过数据清洗总线，根据不同的数据源查询出可用的数据清洗方法，如下图 4.28 所示。

图 4.28 选择清洗方法

Fig. 4.28 Choose Data Clean Method

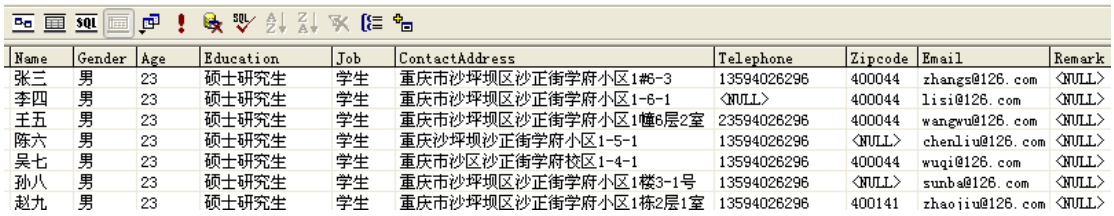
如上图 4.19 中所示，系统根据 TestDB 数据源中配置信息，查询出系统中可以使用的有地址规范化方法，地址补全方法，电子邮件验证方法等具体的组件化的数据清洗方法。

③ 对数据清洗组件库更新。通过对组件库所对应的 XML 文档修改的方式来对数据清洗组件库进行更新。如下图 4.29 所示。

图 4.29 组件库截图

Fig. 4.29 Screenshot for Components Set

④ 对于客户数据的清洗结果。由于客户数据涉及到客户隐私，这里只用模拟一些测试数据来对系统的运行结果做示例说明。图 4.30 为存放在 SQL Server2000 数据库中的待清洗的客户数据。图 4.31 为数据清洗系统经过一系列清洗流程后所得到的清洗结果集，等待领域专家确认。



Name	Gender	Age	Education	Job	ContactAddress	Telephone	Zipcode	Email	Remark
张三	男	23	硕士研究生	学生	重庆市沙坪坝区沙正街学府小区1#6-3	13594026296	400044	zhangs@126.com	<NULL>
李四	男	23	硕士研究生	学生	重庆市沙坪坝区沙正街学府小区1-6-1	<NULL>	400044	lisi@126.com	<NULL>
王五	男	23	硕士研究生	学生	重庆市沙坪坝区沙正街学府小区1幢6层2室	23594026296	400044	wangwu@126.com	<NULL>
陈六	男	23	硕士研究生	学生	重庆沙坪坝沙正街学府小区1-5-1	13594026296	<NULL>	chenliu@126.com	<NULL>
吴七	男	23	硕士研究生	学生	重庆市沙区沙正街学府校区1-4-1	13594026296	400044	wuqi@126.com	<NULL>
孙八	男	23	硕士研究生	学生	重庆市沙坪坝区沙正街学府小区1楼3-1号	13594026296	<NULL>	sunba@126.com	<NULL>
赵九	男	23	硕士研究生	学生	重庆市沙坪坝区沙正街学府小区1栋2层1室	13594026296	400141	zhaojiu@126.com	<NULL>

图 4.30 来自 SQL Server2000 数据库的待清洗数据

Fig. 4.30 Raw Data from SQL Server2000

图 4.31 数据清洗结果

Fig. 4.31 Result of Data Cleaning

如图 4.31 中所示，对待清洗的数据中存在的问题，在异常信息字段中对其进行描述，方便领域专家做进一步处理。

4.8 本章小结

本章遵循软件工程的标准流程，对基于软件总线模型的数据清洗原型系统进行了分析、设计和实现。对系统进行描述，分析系统设计的目标；设计出了系统体系结构；描述了系统中各个模块，包括数据清洗总线，数据源包装器，适配器以及相关数据清洗组件的设计和实现；最后对系统的运行结果进行展示。

5 总结与展望

5.1 总结

本文介绍了数据清洗的研究背景和相关概念，针对数据清洗领域中存在的一些问题，提出了设计并实现一种基于软件总线模型的数据清洗系统的研究目标，分析对比系统中用到的主要技术，对系统中涉及到关键技术进行较为细致的研究，最后对系统进行设计和实现。本文的主要工作可以总结如下几点：

① 剖析了数据清洗的研究背景和意义，阐述了数据清洗的相关概念和当前国内外的研究现状，介绍了国内外的数据清洗方法和数据清洗产品。根据数据清洗领域中存在的一些问题，从系统可复用性和可扩展性角度，提出了将软件总线模型应用于数据清洗领域，设计并实现一种基于软件总线模型的数据清洗系统的研究目标。

② 介绍了 XML 的特点，分析了 XML 的有效性验证方式和两种解析方法的优缺点；介绍了数据库访问 ADO.NET 技术，分析了 ADO.NET 的结构和不同数据读取方式的适用场景，以及将其用于数据访问层的优势；介绍了在中文数据清洗中需要用到的中文分词技术，分析比较了三种主要中文分词方法的优缺点。

③ 对软件总线模型进行了研究，其中包括总线模型结构特点，组件集成方式，总线控制等；对大数据量清洗情况下的清洗任务分解和并发执行清洗任务进行了研究，通过实验对其效果进行验证。结合中文地址的特点，提出一种基于地址分级树的中文地址数据规范化方法，并通过实验验证其具有较好的效果。

④ 从现代软件工程理念入手，对整个系统进行描述，设计了系统的整体结构，描述了系统工作流程。然后对系统中各模块分别进行设计和实现。最后对系统的运行结果展示。

5.2 展望

数据清洗系统是一项非常庞大的工程，涉及到很多的技术和相关知识。本文重点在于初步设计并实现一个基于软件总线模型的数据清洗原型系统，还有一些研究工作需要深入，还有一些问题需要进一步的思考和研究。以下是对系统中需要改进地方的总结：

① 对更多数据源的支持。在本文中实现的是常用的一些常用的关系型数据库，对于文本数据或者其他形式的数据源尚不支持，需要对数据源包装器进一步改进以适用于更多异构数据源。

② 对于数据清洗组件库不断完善。在本文中，目前只从满足当前数据质量要

求角度，实现几种数据清洗组件。对于组件库的不断扩充和升级，可以有效地提高数据清洗质量。

③ 对系统集成组件方式进一步改进，以使得系统能集成更多第三方的或者不同语言实现的数据清洗组件。

④ 从提高系统可复用性和可扩展性角度，对系统的结构需要做进一步地优化，以使得系统能够更加灵活，并且以后可以考虑将本文的研究的系统产品化，以及探索更多应用场景。

总之，数据清洗是一个很复杂、涉及知识面很多的问题，还有很多问题需要在以后的学习研究工作中去解决和完善。

致 谢

在论文完成之际，我体会到了即将离别的伤感。在重庆大学软件学院读完本科和硕士，过去 6 年多的时间中有太多的人给了我无私的帮助，请允许我表示自己诚挚的谢意。

首先感谢我的导师杨丹教授，本文的研究工作是在我的导师杨丹教授的精心指导和悉心关怀下完成的。在我的学业和论文的研究工作中无不倾注着导师辛勤的汗水和心血。导师的严谨治学态度、渊博的知识、无私的奉献精神使我深受的启迪。感谢他这两年半来对我的谆谆劝导，从尊敬的导师身上，我不仅学到了扎实、宽广的专业知识，也学到了做人的道理。在此我要向我的导师致以最衷心的感谢和深深的敬意。

感谢软件学院行业信息化实验室的张小洪、杨梦宁和徐玲等老师，在研究过程中，他们给了我很多宝贵的意见。

感谢软件学院的各位老师，在学院的学习生活中给予我的指导与帮助。在日常学习和生活中，我的同学们也给予了我很大的帮助和热心的鼓励，并提出了许多宝贵的建议，他们的热情帮助是本论文得以顺利完成的重要条件！

感谢我的父母，在我成长过程中给我无微不至的关怀，是他们支持我一路走到现在。

在此，向所有关心和帮助过我的领导、老师、同学和朋友表示由衷的谢意！

最后，衷心地感谢在百忙之中评阅论文和参加答辩的各位专家、教授!

赵 鹏

二 OO 九年十月 于重庆

参考文献

[1] Daniel Aebi，Louis Perrochon．Towards improving data quality[J]．In：Sarda，N.L.，ed. Proceedings of the International Conference on Information Systems and Management of Data.Delhi,1993:273-281.

[2] Wang, R.Y., Kon, H.B., Madnick, S.E. Data quality requirements analysis and modeling. In: Proceedings of the 9th International Conference on Data Engineering. Vienna: IEEE Computer Society, 1993: 670-677.

[3] Hernandez, M.A., Stolfo, S.J. Real-World data is dirty: data cleansing and the merge/purge problem. Data Mining and Knowledge Discovery, 1998,2(1):9-37.

[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(11):2076-2081.

[5] 陈伟, 丁秋林. 可扩展数据清理软件平台的研究[J].电子科技大学学报,2006 (01):100-103.

[6] 叶舟.基于规则引擎的数据清洗[J].计算机工程, 2006, 32(12):100-103.

[7] Rahm E, Do H H. Data cleaning: problems and current approaches [J]. IEEE Data Engineer Bulletin, 2000, 23(4):3-13.

[8] Lingras P, Hogo M, Snorek M, et al. Temporal Analysis of Clusters of Supermarket Customers: Conventional Versus Interval Set Approach[J]. Information Sciences, 2005, 72(1): 215-240.

[9] Monge, A.E. Matching algorithm within a duplicate detection system [J] .IEEE Data Engineering Bulletin, 2000,23(4):14-20.

[10] Qiu, Yue- feng, Tian, Zeng-ping, Ji, Wen-yun, et al. An efficient approach for detecting approximately duplicate database records. Chinese Journal of Computers, 2001,24(1):69-77 (in Chinese).

[11] Hern′andez M. A. ,Stolfo S. J . The merge/ purge problem for large databases [A] . Proceedings of the ACM SIGMOD , International Conference on Management of Data [C] .ACM Press ,May 1995:127-138.

[12] Monge A. E., Elkan C. P. The field matching problem: Algorithms and applications [A].Proc. 2nd Intl. Conf . Knowledge Discovery and Data Mining[C]. Portland , Oregon ,1996.

[13] 张宁，贾自艳，史忠植．数据仓库中 ETL 技术的研究[J].计算机工程与应用，2002,(24):213-216.

[14] 吴远红. ETL 执行过程的优化研究[J].计算机科学,2007,(01):81-83.

[15] 佘春红. 数据清理方法[J].计算机应用,2002,(12):128-130.

[16] 杨辅祥, 刘云超, 段智华. 数据清理综述[J]. 计算机应用研究,2002, (03):3-5.

[17] 沈国忠，王文稚．数据清洗方法在寿险事务处理系统中的应用研究[J].中国金融电

脑,2006,(10):47-48.

[18] 朱六璋. 调度信息系统的数据清洗应用[J]. 电力信息化,2007,(04):67-69.

[19] 连仁包, 曾光清. 数据集成中数据清洗模型的研究[J]. 福建电脑 , 2007,(02):3-4

[20] XML Schemas[EB/OL].http://www.w3.org/XML/Schema.

[21] 顾天竺, 沈洁, 陈晓红, 李慧, 张舒, 吴颜. 基于 XML 的异构数据集成模式的研究[J]. 计算机应用研究,2007,(04):94-96.

[22] 杨剑, 唐慧佳, 孙林夫, 王胜银. 基于 XML 的异构数据交换系统的研究与实现[J]. 计算机工程, 2005,(19):195-197.

[23] 曾春平,王超,张鹏.XML 编程从入门到精通[M].北京：希望电子出版社,2002,2.

[24] Paul Dickinson 著 张晓明 译. ADO.NET 高级编程(M). 中国电力出版社,2003.

[25] 黄昌宁,赵海. 中文分词十年回顾[J] . 中文信息学报, 2007,21(3):8-20.

[26] 孙茂松, 左正平, 黄昌宁. 汉语自动分词词典机制的实验研究[J]. 中文信息学报, 2000,(01): 1-6.

[27] 孙茂松，左正平，邹嘉彦，高频最大交集型歧义切分字段在汉语自动分词中的作用[J]. 中文信息学报，1999，(13):27-34.

[28] 张恒, 杨文昭, 屈景辉, 卢虹冰, 张亮, 赵飞. 基于词典和词频的中文分词方法[J]. 微计算机信息, 2008,(03):239-241.

[29] 杨芙清,梅宏,李克勤. 软件复用与软件构件技术[J]. 电子学报, 1999,127 (2): 68-75.

[30] 周传生, 宋波. 基于 XML 的软件总线设计的研究与实现[J]. 计算机工程与设计, 2006,(20):3808-3810.

[31] 徐正权, 潘晓波. 基于 Adapter 的软件总线体系结构[J]. 华中科技大学学报(自然科学版),2005,(05):10-12.

[32] Plasil F, Visnovsky S. Behavior Protocols for Software Components. IEEE Transactions on Software Engineering, 2002, 28(11): 1056-1076.

[33] 袁占亭, 张秋余, 张冬冬, 翟志万, 李伟平. 基于软件总线技术的软件开发[J]. 计算机工程, 2005,(01):105-107.

[34] 蔡巍, 赵海, 张浩华, 赵明, 周艳. 软件总线研究及其在水电仿真系统中的应用[J]. 系统仿真学报, 2007,(12):710-2715.

[35] 王素丽,牛建强,冯海永.基于 XML 的元数据管理框架研究[J].计算机工程与设计,2008,6:3008-3010.

[36] Brigh tM W et al. A taxonomy and current issues in multidatabase system s. IEEE Computer, 1992, 25 (3):50-59.

[37] 王宁, 王能斌. 异构数据源集成系统查询分解和优化的实现[J]. 软件学报, 2000,(02):222-228.

[38] 李刚,金蓓弘.基于线程的并发控制技术研究与应用[J].计算机工程,2007,(14):43-45.

[39] 谭红叶, 郑家恒, 刘开瑛, 等. 基于变换的中国地名自动识别研究(英文) [J]. 软件学报, 2001, 12 (11) :1608-1613.

[40] 黄德根, 岳广玲, 杨元生.基于统计的中文地名识别[J].中文信息学报,2003,17 (2) :36-41.

[41] 高红, 黄德根, 杨元生. 汉语自动分词中中文地名识别[J]. 大连理工大学学报, 2006,(04):577-581.

[42] 王浩. 数据仓库环境下以用户为中心的数据清洗过程模型[J]. 计算机科学, 2003:52-55.

[43] 张晓明，乔溪．数据清洗方法与构件的综合技术研究[J]．石油化工高等学校学报,2005，18(2):68-71.

[44] Abraham Silberschatz,Henry E Korth,S.Sudarshan 著 杨冬青,唐世渭等译 数据库系统概念(第 4 版) (M). 机械工业出版社,2003.

[45] Christian Nagel,Bill Evjen,Jay Glynn 等著 李敏波译 C#高级编程(第 4 版) (M). 清华大学出版社,2006.10.

[46] Erich Gamma,Richard Helm,Ralph Johnson,John Vlissides 著 李英军，马晓星等译. 设计模式：可复用面向对象软件的基础 (M). 机械工业出版社,2000.

附 录

A. 作者在攻读硕士学位期间发表的论文

[1] 杨梦宁, 赵鹏, 张小洪, 李朋. 一种基于总线模型的数据清洗方法. 计算机科学, 已录用. 预计于 2010 年第 4 期正刊发表。

[2] 赵鹏. 基于分级树的邮政地址匹配方法研究. 重庆理工大学学报. 已录用。