基于 Hadoop 的 Deep Web 查询结果自动抽取研究

重庆大学硕士学位论文

（学术学位）

学生姓名：王慧娟

指导教师：冯　永　教　授

专　　业：计算机软件与理论

学科门类：工　学

重庆大学计算机学院

二 O 一四年四月

Hadoop-based Automatic Deep Web Data Extraction

A Thesis Submitted to Chongqing University

in Partial Fulfillment of the Requirement for the

Master's Degree of Engineering

By

Huijuan Wang

Supervised by Prof. Feng Yong

Specialty: Computer Software and Theory

College of Computer Science of

Chongqing University, Chongqing, China

April, 2014

摘　要

随着互联网技术的快速发展与广泛应用，网络提供给用户的资源与日俱增。特别是不能被传统搜索引擎通过静态链接而获取的海量信息资源规模增长显著，这部分资源称为深层网（Deep Web）。关于 Deep Web 的研究，是近年来 Web 数据管理方向的研究热点。

Deep Web 中的信息是通过在特定查询接口提交查询后，以结果页面的方式显示，对 Deep Web 查询结果进行抽取，将信息资源进行集成，使其以统一的模式进行存储，才能为用户提供更好的、统一的索引服务。因此，Deep Web 查询结果抽取是 Deep Web 数据集成系统中的关键步骤。

本文主要对基于 DOM 树结构与模板方法相结合的抽取算法进行了深入的研究，主要研究及成果如下：

① 深入研究比较了几种主要的抽取技术，重点对基于 DOM 树结构与基于模板的抽取算法进行了详细的介绍，并对各种技术在复杂性、适用范围和自动化程度等方面进行了分析与比较；

② 综合基于 DOM 树结构与基于模板抽取算法的优点，提出了基于 DOM 树结构与模板方法相结合的抽取算法 FIME（Filtering, Iterating, Matching, and Extracting）算法，FIME 算法在进行 DOM 树结构比较之前，首先对页面进行预处理操作，使页面遵守 XHTML 规则，同时清除页面中对于抽取信息无用的标签及部分属性元素，使得页面更精简，以提高后续匹配算法的效率；

③ 针对基于 DOM 树结构抽取算法中回溯处理页面中冗余迭代项导致匹配算法复杂度高的问题，FIME 在进行匹配之前首先对页面中的迭代项进行合并，降低了后续匹配算法的时间复杂度；

④ 结合基于模板抽取算法的思想，FIME 将在匹配算法中通过比较 DOM 树结构而获得的待抽取数据的位置信息作为同一网站页面的模板 Wrapper，对所有同源页面进行待抽取信息的自动抽取，而不是对同源结构相似的页面做重复的处理，提高信息抽取的效率和自动化程度。

由于 Deep Web 查询结果页面返回时为海量数据，基于单一节点的抽取算法存在计算瓶颈。目前，开源的分布式系统基础架构 Hadoop 平台已经非常稳定，因此，本文将 FIME 算法部署在 Hadoop 平台中进行实验，实验结果表明 FIME 算法具有较高的抽取准确率和执行效率。

关键词：Deep Web 查询结果，DOM 树，模板，FIME 算法，Hadoop

ABSTRACT

The rapidly development and extensive application gave a result that the resources provided by the Internet have increased steadily. Especially those huge information resources can't be searched by static links with traditional search engines, which called Deep Web, are dramatically increasing. The study on Deep Web is a hotspot in Web data management in recent years.

The information of Deep Web is displayed in the results page after submitting queries to a specific interface, only through extracting information from Deep Web query results and integrating these information resources, storing them in a unified model, we can provide users a better, unified indexing service. Therefore, the extraction of Deep Web query results is a critical process in a Deep Web data integration system.

This paper primarily focuses on the deep research on extraction algorithm which combined DOM tree-based structure and template. The main contents of this thesis are as follows:

① Compares several major extraction technologies, and makes the focus on the extraction algorithm based on DOM tree structure and template to detailed describe, moreover, makes an exhaustive analysis and comparison of a variety of techniques in terms of complexity, scope and degree of automation;

② Integrated the advantages of DOM tree-based extraction algorithm and template-based extraction algorithm, a novel approach which combines DOM tree-based extraction algorithm and template-based extraction algorithm called FIME (Filtering, Iterating, Matching, and Extracting) algorithm is proposed. Before the comparison with The DOM Tree-based structure, FIME makes a preprocess for the pages in order to make the pages in accordance with the standard XHTML. Furthermore, FIME algorithm clears up the tags and part of the property element in the pages which are useless for extracting information, making the pages more simplify and improving the efficiency of matching module;

③ According to the existing problems that when backtracking the iterations of pages may cause high complexity with the DOM tree-based extraction algorithm, FIME merges iterations of the input pages before matching, reducing the time complexity of matching module;

④ Apply the idea of template-based extraction algorithm, FIME algorithm use the

position information of the data waiting for extraction by comparing the DOM tree Structure as the model Wrapper for the same website in the matching module; in the extracting module, it uses the wrapper to automatic extracting information of the pages that come from the same Website with the wrapper, instead of repeated treatment, to improve the extraction efficiency and automation degree;

Since the returned data of the Deep Web query result pages may be massive, the single node extraction algorithm exists the calculate bottleneck. Currently, the open source distributed system architecture Hadoop has been very stable, and therefore, FIME algorithm is deployed in Hadoop platform, Experimental results show that the FIME algorithm has higher extraction accuracy and efficiency.

Keywords: Deep Web query results, DOM tree, template, FIME algorithm, Hadoop

目 录

1 绪 论

1.1 背景和意义

随着万维网技术的快速发展，网络中出现了越来越多的在线后台数据库，这些后台数据库的共同特点是可以通过提交特定的查询表单来获取其中的数据。这些数据一般不能被传统的搜索引擎通过静态链接而获取[1]。在线后台数据库中数据的获取过程一般为：用户在客户端提交查询请求，服务器端根据请求动态生成结果页面，再将结果页面返回给用户。通常将这些隐藏的后台在线数据库称为深层网，即 Deep Web，也称为 Hidden Web[2-3]。页面内容是根据用户请求动态生成，没有直接指向这些数据的静态链接，这是 Deep Web 数据与传统的 Surface Web 中静态页面的本质区别，如图 1.1 所示。

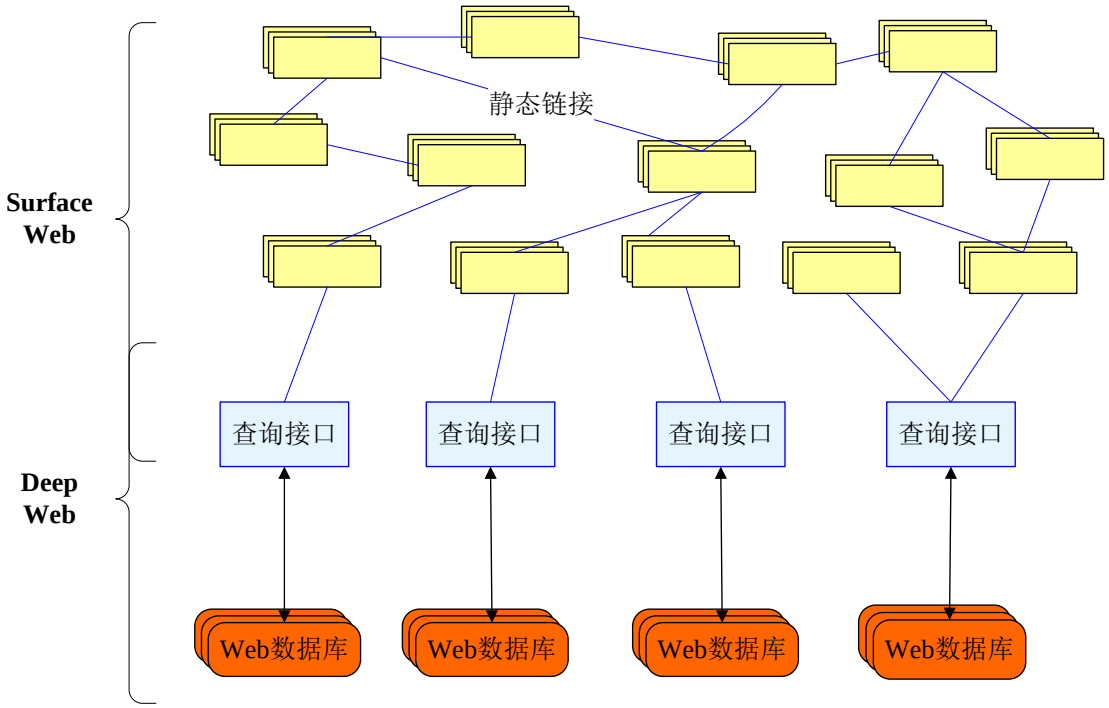


图 1.1 深层网与传统静态链接页面的关系示意图

Figure 1.1 the relationship schematic of traditional static link pages and deep web pages

受到搜索技术的限制，传统搜索引擎一般只能抓取并索引到网络中可以通过超链接直接获取的静态网页、数据文件等，而很少能够索引到隐藏在后台数据库中这些重要的资源。相对于 Surface Web，Deep Web 中蕴藏的资源更加丰富、更加专业，而且这些资源大多是权威的、高质量的信息，检索查询 Deep Web 中的信息已经成为搜索技术的一大研究热点。

2004 年，Chang[1]等人对整个 Deep Web 数据规模进行调查统计的结果为：大约有 307,000 个 Deep Web 站点在使用后台数据库，而后台数据库的总量约为 450,000 个，其中的信息量约是普通静态页面信息的 500 倍左右。这个数据约为 Bergman 等人在 2000 年做的调查结果[2]的 6~7 倍。

在中国，Deep Web 数据量同样在迅速增长，2005 年 7 月 CNNIC 发布了第十六次中国互联网信息资源数量的调查报告，报告显示[4]：中国的在线数据库总数为 3.06 万个，在中国的网站中，拥有在线数据库的个数为 16.1 万个，约占总数的 24.1%。在线数据库的总数年增长近 13 万个，增长率高达 80%。在 2010 年，清华大学通过搜索引擎对中文的 Deep Web 规模进行了统计，结果显示：当前中文 Deep Web 中有 60 多万个查询接口，大约为 2006 年做的类似统计结果数据的 7 倍[5]。

Deep Web 页面中的数据信息存储在后台数据库中，这些页面是通过用户提交特定的查询而生成的。虽然 Deep Web 中的内容对于用户来讲是高质量的信息，但是 Deep Web 数据库数量众多，如果仅通过人工的方式遍历所有的 Deep Web 数据库并通过提交查询来获取信息，对于时间和人力的耗费无疑是巨大的。目前，国内外学者逐渐将研究重点倾向于 Deep Web 信息的集成，使用户更方便的使用 Deep Web 中所蕴含的信息。

从应用的角度来讲，Deep Web 数据搜索平台可以为用户提供更实用的数据和服务。从互联网门户的角度来讲，可以利用 Deep Web 数据搜索平台收集、存储、分类及索引各类信息，从而为用户提供更加方便及个性化的信息搜索服务。因此，对 Deep Web 的研究将会产生可观的社会及经济效益。

Deep Web 中的信息广泛分布在各个 Web 后台数据库中，并且只能通过在特定的查询接口提交查询后以结果页面的方式显示，因此，必须对查询结果页面中的 Deep Web 有效信息进行抽取，才能对 Deep Web 中的信息资源进行集成，使其以统一的模式进行存储，最后为用户提供更好的、统一的索引服务。由此可见，Deep Web 查询结果的抽取是 Deep Web 数据集成系统中的关键步骤，对 Deep Web 查询结果抽取的研究具有深远的理论价值及实际应用价值。

受到 Google Lab 开发的 MapReduce[6]和 Google File System[7]的启发，Apache 公司于 2005 年正式引入 Hadoop 项目。Hadoop 实现了分布式文件系统 HDFS 和并行编程框架 MapReduce，作为一个成功的分布式系统架构，Hadoop 可以使用户在不了解分布式底层细节的情况下，开发分布式程序，目前，Hadoop 广泛地应用在脸谱、雅虎等互联网公司的海量数据的存储与处理及分析应用中。由于 Deep Web 查询返回的结果页面数量众多，因此，本文基于 Hadoop 分布式系统基础架构对 Deep Web 查询结果自动抽取技术进行研究。

1.2 国内外研究现状

Deep Web 概念被提出以后，国外的高校、科研机构和商业机构逐渐掀起了针对 Deep Web 信息资源研究的热潮。为了使用户能够更加方便的使用 Deep Web 后台数据库中的资源，这些研究遍及了数据源发现、数据源分类、查询接口的自动发现与集成、数据抽取等方面。Deep Web 查询结果的抽取，是 Web 信息抽取技术的一部分，它针对在线后台数据库中返回的结果页面进行抽取。

Liu[8]等人提出了一种基于字符串匹配和标签树的算法来抽取结果页面中的数据记录，该算法能够抽取非连续的和连续的数据记录，其实验结果证明该算法优于当时已存在的其他算法。

Arasu 和 Garcia-Molina[9]研究了从由模板生成的查询结果页面中抽取数据的算法，该文中定义了模板的概念，同时用一个模型描述了数据库中的数据是如何通过模板嵌入到查询结果页面中的。该算法不需要人工干预或学习样本，只将一组由模板生成的查询结果页面作为输入，即可推导出用于生成结果页面的模板，并抽取出结果页面中的数据作为输出。在多数情况下，该算法获得较高的正确率。

Yeonjung Kim[10]等人认为传统的基于树的模式匹配算法存在缺陷，即传统的算法中假设所有的 HTML 标签均拥有相同的价值，进而为 HTML 标签树中每一个节点都设置了相同的权重。文中提出了一个改进的树匹配算法，该算法是根据每个节点在浏览器中显示的数据对象的权重而获得该节点自身权重，进而提高了算法的精确度。

Gengxin Miao[11]等人提出了一种新的抽取记录方法，该方法是通过比较一组标签路径的出现规律，即视觉信号，来评估这两个标签路径是表示一组对象的可能性。该文中介绍了一个捕获视觉信号出现和交错的相似度度量算法，基于该算法通过对标签路径的聚类来获取形成记录数据的标签路径。

国内的各大高校和科研机构对 Deep Web 也进行了深入而有效的研究：中国人民大学的刘伟、孟小峰等较早的介绍了关于 Deep Web 数据集成方面的内容[12]。该团队在 Deep Web 查询结果页面内容抽取方向，分析了几项结果页面的视觉特征，提出了一种结合结果页面的视觉特征来完成数据抽取的算法[13]。该算法的最大特点是抽取过程与结果页面的语言种类无关，因此，适合在多语种环境下使用。

东北大学的申德荣团队在最近几年也取得了丰硕的研究成果。他们通过分析 Deep Web 查询结果页面的特点，针对如何有效的、准确的抽取 Deep Web 查询结果页面中包含的实体信息问题，提出了一个基于 DOM 树的抽取算法[14]，较有效的解决了 Deep Web 环境下的实体抽取问题。

陶磊[15]等人提出一种基于 MDR 和 CSS 选择器的算法，用于抽取 Deep Web 查询结果页面中的数据记录。该文中将 Deep Web 查询结果页面的结构分为良好与

较差两类：对于结构良好的页面，使用基于 CSS 选择器算法来抽取数据记录；而对于结构较差的页面，则使用 MDR 算法来抽取数据记录。

马安香[16]等人提出一种基于结果模式的 Deep Web 查询结果数据抽取算法，该算法将数据抽取过程分为结果模式的生成和数据的抽取两部分。算法在结果模式生成阶段添加属性语义标注，有效的克服了重复语义标注的问题。

朱明[17]等人针对多数的抽取算法适用性较差的问题，采用将隐式马尔科夫学习策略与传统的文本分类器相结合的思想，提出一种基于多学习策略的页面数据抽取算法。

郑皎凌[18]等人引入了数据记录伪属性和语义匹配的概念，通过对数据记录间伪属性的语义匹配来抽取数据记录。对于结构较不规范的 Deep Web 页面，该算法具有一定优势。

为了推动 Deep Web 数据集成研究在国内的发展，2008 年《软件学报》推出了 Deep Web 数据集成专刊，用于刊登国内在数据集成方向的最近进展，展现国内在此方向较高水平的研究成果和项目。

现在国内外对于 Deep Web 信息抽取技术的研究主要集中在：基于 DOM 树结构的抽取技术、基于模板的抽取技术、基于视觉特征的抽取技术和基于统计理论的抽取技术。

1.3 本论文研究工作

本论文根据目前 Deep Web 查询结果抽取技术的相关研究，重点研究了基于 DOM 树与模板方法相结合的查询结果抽取算法。Deep Web 查询结果页面是在特定的查询接口提交查询，由后台数据库根据请求动态生成的页面。基于模板的抽取技术是针对通过读取数据库然后填充到统一模板的方式生成的页面而设计的，而基于 DOM 树的抽取技术可以通过对比 DOM 树即可确定页面中主题信息的位置，因此，将基于 DOM 树与模板的抽取技术相结合可以实现自动化的获取后台数据库的模板，无需人工干预。本论文的研究工作主要包括：

① 介绍 Deep Web 的概念，详述 Deep Web 查询结果抽取技术的发展及现状，对常用的 Deep Web 查询结果抽取算法进行了研究，将各个算法进行了总结并加以分类。

② 针对基于 DOM 树与基于模板的抽取技术在自动化程度、适用范围及复杂性等方面的优越性及缺陷，本文提出了基于 DOM 树与模板抽取技术相结合的抽取算法 FIME(Filtering, Iterating, Matching, and Extracting)，该算法在已有的基于 DOM 树抽取算法 RoadRunner 的思想基础上进行了算法改进，同时汲取了基于模板抽取算法的思想，能够完全自动化的对查询结果进行抽取。

③ 介绍了 Hadoop 的 HDFS 分布式文件系统及 MapReduce 分布式并行计算架构。由于 Deep Web 查询结果页面数量众多，因此，本文基于分布式系统基础架构 Hadoop 来实现 FIME 算法，使之能够高效、并行的对查询结果进行抽取。

本文共由六章组成，具体安排如下：

第一章，绪论。介绍了本论文的研究背景及意义，分析 Deep Web 查询结果抽取技术的国内外研究现状，阐述了本论文的主要研究内容，并说明论文的组织结构。

第二章，Hadoop 相关技术。本章首先介绍了当前的研究热点云计算，然后引入 Hadoop 的介绍，包括 Hadoop 的产生过程，最后重点介绍了 Hadoop 的两个核心技术：Hadoop 分布式文件系统 HDFS 和并行计算框架 MapReduce。

第三章，Deep Web 信息抽取技术。本章首先介绍了信息抽取技术的发展历史，然后就现在国内外对于 Deep Web 信息抽取技术的主要研究方向进行了详细的介绍，分别为：基于 DOM 树结构、基于模板、基于视觉特征和基于统计理论的抽取技术，并对四类抽取技术在复杂性、适用范围和自动化程度等方面进行了分析。

第四章，基于 DOM 树结构与模板方法相结合的 Deep Web 查询结果抽取算法 FIME。本章提出了 FIME 算法，该算法包括四个模块：清噪模块，迭代模块，匹配模块和抽取模块。首先，在清噪模块中对结果页面进行网页预处理；然后，在迭代模块中对页面中的迭代项进行合并，将合并迭代项后的页面在匹配模块中进行对比，得到后台数据库的模板；最后，在抽取模块中使用已获取的模板对同一后台数据库中的其他结果页面进行内容抽取。本章详细介绍了 FIME 算法结构，以及每个模块的思想,最后介绍了基于 Hadoop 的 FIME 算法设计与实现。

第五章，实验与结果分析。首先介绍了基于 Hadoop 的 FIME 算法实现系统硬件和软件环境配置,然后使用大小约为 448M 的结果页面集对基于 Hadoop 的 FIME 算法系统进行实验，最后分别从准确率、召回率、F 均值、分布式执行时间及效率等方面对算法进行了分析。

第六章，总结与展望。首先对本文所做的研究工作进行了总结与概括，然后在分析本文不足的基础上，对未来的研究工作重点进行了展望。

1.4 本章小结

本章首先介绍了 Deep Web 的研究背景和 Deep Web 查询结果页面抽取的研究意义；然后介绍了国内外 Deep Web 查询结果页面抽取的研究现状；最后，简要介绍了本论文的主要研究工作，并对各章的内容进行了概述。

2 Hadoop 平台概述

2.1 Hadoop 平台背景

云计算(Cloud Computing)是分布式计算(Distributed Computing)、并行化计算(Parallel Computing)、网格计算(Grid Computing)和虚拟化(Virtualization)等传统计算机和网络技术发展融合的产物。云计算的核心理念是服务，这是新型的计算机服务模式，用户通过简单的界面即可得到需要的计算资源和信息服务。云计算又是新型的商业服务模式，能支持多种行业通过相关领域应用云计算技术而获得显著效益，得到了人们的广泛关注[19-22]。

Hadoop 是由广泛使用的 Apache 文本搜索工具 Lucene 的创始人 Doug Cutting 提出的。2002 年时，Doug Cutting 开始着手 Nutch 项目，项目中将搜索系统和爬虫合并在一起，虽然当时可以使用 4 台机器支撑 1 亿个网页的抓取与检索，但是，系统的可扩展性遇到了瓶颈。2003 年时，Google 发布了分布式文件系统 GFS 论文，GFS 可以解决大量爬取文件和索引文件的存储问题，节约的时间将用于存储节点的管理工作。2004 年时，Doug Cutting 等人开始了 Nutch Distributed Filesystem (NDFS)开源项目的实现，Google 此时发布了 MapReduce 论文，随后 Nutch 开发人员实现了 MapReduce 算法，并于 2005 年将 Nutch 上的所有算法运行在了 NDFS 和 MapReduce 上。2006 年时，NDFS 和 MapReduce 作为一个独立开源子项目，被命名为 Hadoop。

随后，Hadoop 在 Apache 开源社区得到了快速发展，成为了一个拥有多个子项目的项目集合。其中，最核心的是 HDFS (Hadoop Distributed File System)和 MapReduce。HDFS 用于分布式的存储，而 MapReduce 是分布式计算的架构。

2.2 Hadoop 分布式文件系统 HDFS

当需要存储的数据集很大，已经超过了一个单独物理服务器所能接受的容量时，就需要将数据分布到多个独立的服务器上，用于管理通过网络连接的服务器存储系统被称作分布式文件系统，由于基于网络，因此需要使用到网络编程，这使得分布式文件系统比传统硬盘文件系统更加复杂。使用分布式文件系统可以将分布在不同服务器上的共享文件组织在一起，用户只需访问一个共享的 DFS 目录，就可以访问到分布在网络上的文件，而不需要知道这些文件的实际物理存储位置。

2.2.1 HDFS 体系结构

HDFS 是 Hadoop 并行计算的存储支撑，HDFS 系统中主要术语如下：

① NameNode(NN)：主控服务器，是 HDFS 的控制中心，管理各个数据服务器，并提供文件系统的目录信息。

② SecondaryNameNode(SNN)：备份主控服务器，负责备份主控服务器上的日志，如果主控服务器出现故障，将代替主控服务器进行工作。

③ DataNode(DN)：数据服务器，分布式文件系统的每个文件都被切分为若个个数据块，每个数据块都被存储在不同的数据服务器上。

④ Block：数据块，HDFS 的基本存储单位，默认大小为 64M，必须为 512bytes 的整数倍。

HDFS 采用 master/slave 架构模式，一般在 master 上只部署运行一个 NameNode，在每一个 slave 上运行一个 DataNode。HDFS 支持传统的层次文件组织结构以及文件的复制、重命名和移动等操作。HDFS 的架构如图 2.1 所示。

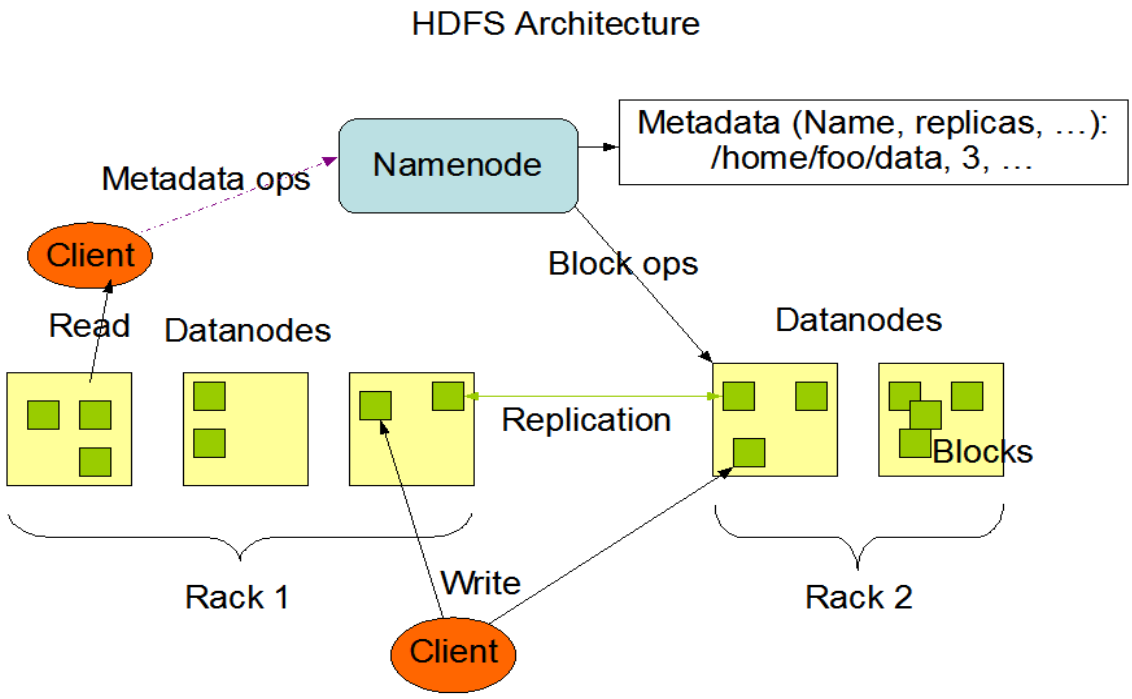


图 2.1 HDFS 架构图

Figure 2.1 HDFS Architecture

NameNode 负责管理文件系统的集群配置信息、命名空间和数据块的复制等，还有对文件系统操作（建立、复制、删除）的控制。NameNode 存储文件系统的元数据，包括文件信息、文件的 Block 信息和 Bolck 所在的 DataNode 信息等。有的版本中存在 Secondary NameNode，它主要的作用是编辑日志和合并命名空间的镜像。当 NameNode 出现故障时，便启用 Secondary NameNode，但是，它的运行明

显滞后于 NameNode。DataNode 上存储数据块，同时周期性的将数据块的信息发送给 NameNode。

从 HDFS 的内部实现讲，一般大的数据将会被分为多个数据块（每个数据块默认为 64M），每个数据块的大小是相同的，除了最后一个，这些数据块分别存储在不同的 DataNode 上，而且，每个数据块都有自己的副本，默认为 3 个，以便在单节点失效时，能够从其他节点获得同样的数据。

HDFS 的一个数据块的大小比一般的磁盘文件的数据块大小要大很多，主要目的是节省寻址时间。理论上讲，如果数据块足够大，则从磁盘传输数据所需的时间明显变长，大于寻址所需的时间。

2.2.2 HDFS 的工作流程

Hadoop 分布式文件系统中，客户端、NameNode、DataNode 之间是通过 TCP/IP 协议来通信的。HDFS 主要的操作过程如下：

① 文件写入

首先，客户端向 NameNode 发出文件写入请求；然后，NameNode 根据客户端文件的大小和 Hadoop 的环境配置情况，返回给客户端数据块的大小以及这些数据块分配到哪些 DataNode 上；最后，客户端按照数据块的大小，将数据写入到指定的 DataNode 中。

② 文件读取

首先，客户端向 NameNode 发出文件读取请求；然后，NameNode 根据客户端的请求路径，查找相应数据块存储的 DataNode，并将相应信息发送给客户端；最后，客户端直接从相应的 DataNode 中读取数据。

③ 数据块复制

集群中的每一个 DataNode 会定期的向 NameNode 发送“心跳”及 block 报告，NameNode 根据 block 报告的信息将对副本数量低的数据块进行复制。复制的过程是，NameNode 通知 DataNode 需要复制的数据块，然后 DataNode 之间直接进行通信互相复制数据块。

2.2.3 HDFS 的特点

HDFS 的主要目标是即使文件系统在出现故障的情况下，仍然能够可靠的存储数据。这些故障包括：网络故障、NameNode 故障和 DataNode 故障。这就要求 HDFS 具有良好的健壮性。HDFS 的健壮性表现为：

① 对 DataNode 的检测：每个 DataNode 会定期的向 NameNode 发送“心跳”响应消息。如果 NameNode 没有收到 DataNode 的响应消息，将会认为该 DataNode 发生故障，不会再将任何新的 I/O 请求发送给它。此时会导致部分数据块的副本数量减少，NameNode 会根据定期追踪到的每个数据块的剩余副本数量，对数量少的数据块进行复制。

② 拥有并发控制的实现机制：一次写入多次读取。HDFS 在任何情况下都只允许一个写入的客户端，文件创建后就不能再修改。这种机制是由 Hadoop 的应用场景所确定的，HDFS 的文件大小通常为 TB 甚至 PB 级别，这些数据一般不会经常被修改，而是经常被读取。

③ 数据冗余的设计方式：Hadoop 的配置文件里，默认每个数据块的副本个数为 3。其中，一个副本存放在本地机架的某个 DataNode 上面，另一个副本存放在同一个机架的另一个 DataNode 上面，最后一个副本存放在不同机架上的某个 DataNode 上面。这种设计方式能够很好的防止因故障导致的副本丢失。

④ 数据完整性：由于网络故障、软件缺陷和存储设备故障等都有可能导致一个已经获取的 block 内容是损坏的，因此，HDFS 在文件创建时，会计算每个数据块内容的校验和，并将校验和存储在一个单独的隐藏文件中，当读取数据块内容时，会自动校验文件的校验和是否正确，若校验失败，则从其他 DataNode 获取该文件的副本。

2.3 Hadoop 的 MapReduce 计算框架

2.3.1 MapReduce 模型

Hadoop 用于分布式处理海量数据，因此，Hadoop 作业将海量数据作为输入，在调用 Map 程序前，首先将海量数据进行分块处理，然后每一个 Map 以并行的方式处理一定数目的 block。Map 程序处理过后的输出数据将会写入到内存或本地磁盘，然后发往不同的 Reduce 进行处理，最后将 Reduce 处理过后的输出数据写回到 HDFS 系统中。MapReduce 模型如图 2.2 所示。

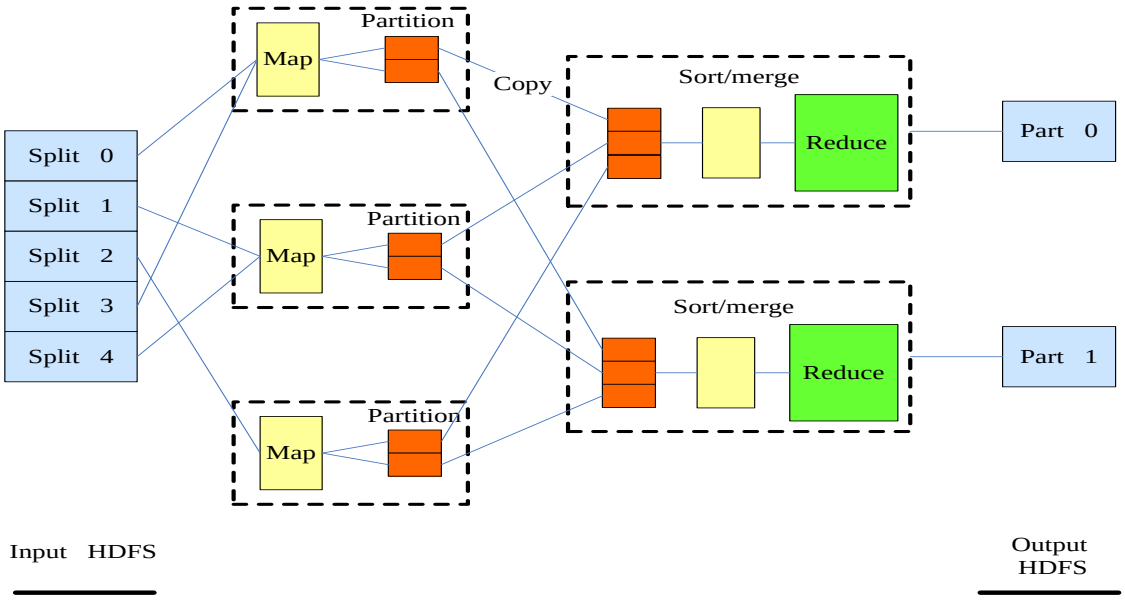


图 2.2 MapReduce 框架模型

Figure 2.2 MapReduce Framework

2.3.2 MapReduce 的实现

MapReduce 框架中主要的术语如下所示：

① Job：作业，用户的每一个计算请求都称为一个作业。

② Task：任务，每个作业都会被分为若干个执行单元，交给不同的服务器去执行。这些执行单元称为任务。

③ JobTracker：作业服务器，用于用户提交作业的服务器，负责任务的分配，管理所有任务服务器。

④ TaskTracker：任务服务器，执行作业服务器分配的具体任务，并定时向作业服务器发送执行情况。

MapReduce 的执行流程如图 2.3 所示。

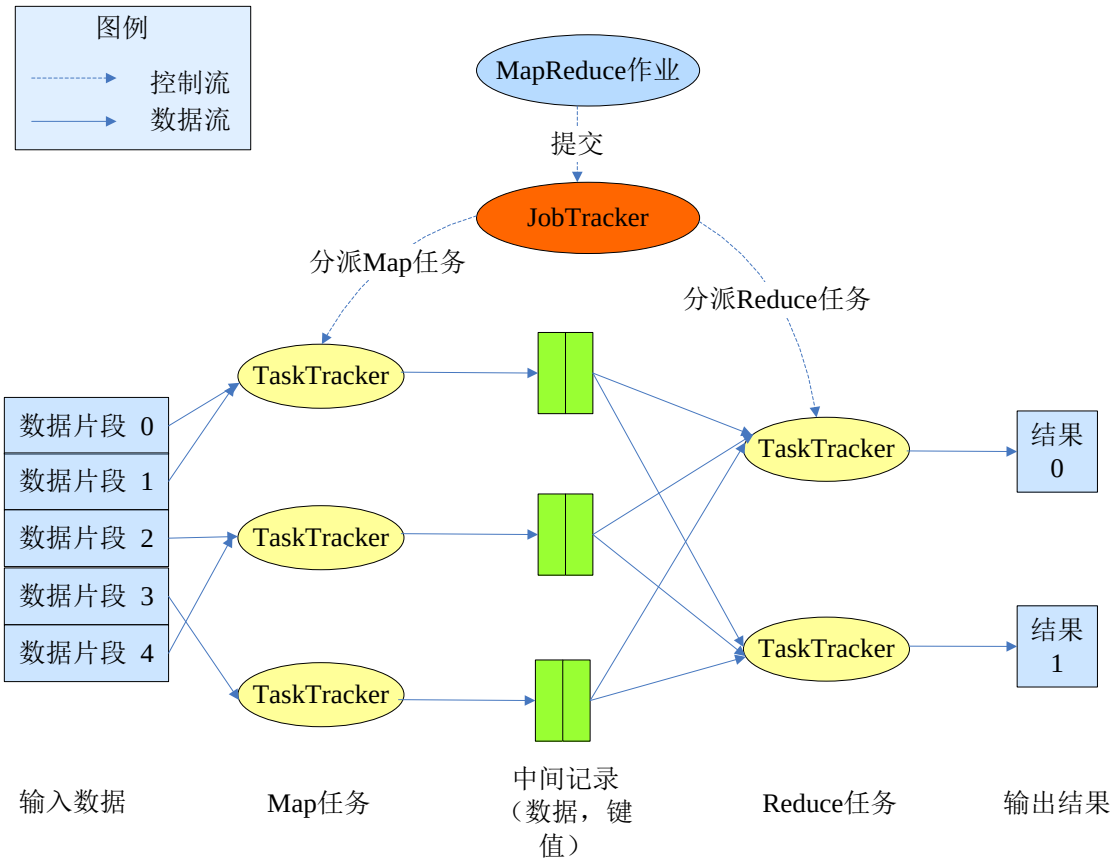


图 2.3 MapReduce 的执行过程

Figure 2.3 MapReduce Implementation Process

由图 2.3 可以看出，当用户提交一个作业之后，首先，由 JobTracker 确定作业输入的 HDFS 文件个数和这些文件对应的数据块个数，然后，根据数据块个数将不同的数据块分配到不同的 TaskTracker 上，TaskTracker 根据数据块个数和配置的

最大 Task 任务数开始运行 Map 任务，Map 任务将会对数据块上数据进行运算，之后，Map 任务会将生成的中间结果写入到本地磁盘，然后 TaskTracker 将启动 Reduce 任务，Reduce 任务包含 MapReduce 框架中默认的 Shuffle 和 Sort 操作，这两个操作是保证 Reduce 的输入数据是 Map 排好序后的输出数据，Shuffle 通过 HTTP 为每个 Reduce 任务获取 Map 输出数据中与之相关的键值对，而 Sort 负责将键值对按照 Key 值进行分组排序，通常 Shuffle 和 Sort 是同时进行的。Reduce 任务对输入的键值对进行计算，最后将计算结果输出到 HDFS 中。有时会有一个 combiner 任务，combiner 实际也是一个 reduce 任务，不同的是 combiner 运行在 Map 任务中，对每个 Map 任务的输出结果先进行一次 Reduce，以减少中间结果的输出，加快整个作业的执行速度。

Map 任务的输入数据与输出数据都是键值对，但是格式可能并不相同，键值对的相关信息可以在配置文件中进行设置，一般情况下，当逐行读取数据时，可将该行第一个 Tab 键之前的数据作为键，之后的数据作为值。Map 任务的个数通常根据输入的 HDFS 文件的数据块个数来确定。Map 个数的设置对于整个作业的执行性能有着重要的影响，一般情况下，对于消耗 CPU 较少的 Job，可稍微将 Map 的数目设置的大一些，但是，用户设置的 Map 个数不一定有效，这只是对 Hadoop 的一个期望，实际的 Map 任务个数会根据当时的实际情况和配置文件而定。

Combiner 任务对于 MapReduce 框架来讲并不是必须的，如果 Map 任务的输出数据量不是很大，则不需要 Combiner 任务，反之，将不利于网络中数据的传输，则需要 Combiner 任务对 Map 任务的输出数据进行合并，减少中间结果的输出。

Map 任务与 Reduce 任务的具体工作为：Map，首先从磁盘中读取数据，然后执行 map 程序，之后可能会执行 Combiner 程序，最后将中间结果写回到本地磁盘。Reduce，首先获取 Map 的输出结果同时对其进行排序，即 Shuffle 和 Sort 过程，然后执行 Reduce 程序，最后将执行结果写回到 HDFS 文件系统中。

2.3.3 Shuffle 过程

Shuffle 是 MapReduce 编程框架的灵魂所在，它所负责的任务就是如何将 Map 的输出高效的传输到 Reduce 端。因此，Shuffle 就像一座桥梁，连接着 Map 端和 Reduce 端。

一般情况下，在大型 Hadoop 集群中，Map 任务和 Reduce 任务运行在不同的节点上，因此，在很多情况下，Reduce 任务需要从不同的 Map 任务所在的运行节点上获取数据。如果集群中同时运行着相当多的作业，那么数据获取将会造成很大的网络资源消耗。但是，一个好的 Shuffle 算法能够最大化的降低这种消耗。

Map 的输出结果是键值对，MapReduce 框架提供了 Partitioner 接口，它的作用是根据 Reduce 任务的个数以及 Map 输出的键值对来决定每个键值对应该由哪个

Reduce 任务来进行处理。默认是采用 HashPartitioner 算法，即首先对 Key 进行 hash，然后再以 Reduce 任务的个数取模，这种方式的主要目的是为了平均 Reduce 任务的处理能力，然后将这些键值对输出到内存缓冲区中。一般情况下，内存缓存区的大小有限，默认为 100M。当 Map 的输出结果较多时，显然缓冲区是远远不够使用的，当达到一定阈值（默认为 0.8）时，将启动溢写线程，此线程锁定其中 80M 的缓冲区，写到本地磁盘，此时，Map 任务将中间结果输入到剩余的 20M 缓冲区中，如果 20M 缓冲区用完时，溢写程序还未释放之前的 80M 缓冲区，则 Map 任务将暂时被挂起。如果此时设置了 combiner 任务，将会对 Map 任务的输出结果做合并处理，这样可以减少数据写入和传输带来的各种开销。每次执行溢写程序都会在磁盘中生成一个溢写文件，Map 任务完成时，将会对这些文件做一个归并过程，生成一个最终的文件，此时，Map 任务执行结束。

Reduce 任务的 Shuffle 算法过程大概包含三个步骤：复制过程，Reduce 任务会启动一些线程，通过 HTTP 向已经执行结束的 Map 任务发出传输请求，Reduce 任务最终通过网络获取已写入到本地磁盘中的 Map 任务的输出结果。合并过程，此过程主要是将来自不同 Map 任务的输出结果进行合并，如果此时客户端有设置 combiner 任务，那么 combiner 任务同样将会被调用。最终文件，这个最终文件只是 Shuffle 过程的最终文件，并不是 Reduce 任务的最终执行结果。

这三个步骤执行完成之后，Shuffle 算法过程便结束了。接下来将会执行 Reduce 任务，生成的最终结果将会写入到 HDFS 文件系统中。

2.3.4 MapReduce 的特点

MapReduce 计算框架具有良好的容错性。在整个作业的运行过程中，TaskTracker 会周期性的向 JobTracker 发送“心跳”消息，以报告 JobTracker 每个 Task（包括 Map Task 与 Reduce Task）的运行状态。当 JobTracker 没有收到 Task 的“心跳”消息时，则认为 Task 出现了故障，此时，JobTracker 会重新将该 Task 的任务分配给其他 TaskTracker，以保证整个作业的顺利执行。

MapReduce 计算框架为了加快执行速度，当 TaskTracker 中有空闲 Task 时，JobTracker 会将执行速度过于缓慢或是执行出错的 Task 上的任务分配到空闲的 Task 上去运行，以保证 MapReduce 的计算任务可以在高速的节点中快速完成，同时也保证了 Hadoop 能够在廉价的设备中快速执行。

2.4 本章小结

本章首先介绍了当前的研究热点云计算，然后引入 Hadoop 的介绍，包括 Hadoop 的产生过程，最后重点介绍了 Hadoop 中的两个核心技术：分布式文件系统 HDFS 和并行计算框架 MapReduce。

3 Deep Web 信息抽取技术

3.1 信息抽取技术历史

从自然语言文本中抽取结构化信息被认为是信息抽取技术的初始研究，开始于 20 世纪 60 年代，以两个研究性质的、长期的自然语言处理项目为代表[23]。美国纽约大学开展的 Linguistic String 项目[24] 开始于 60 年代中期并且一直延续到 80 年代，该项目中建立了一个大规模的英语计算语法，主要应用在医疗领域，用于从医院出院记录和 X 光报告书中抽取信息格式，这种信息格式就是模板。另外一个项目是由耶鲁大学 Roger Schank 和同事在 70 年代开展的一项有关于故事理解的研究。他的学生 Gerald De Jong 设计并实现了一个根据故事脚本理论建立的信息抽取系统 FRUMP [25]。该系统采用期望驱动与数据驱动相结合的方式从新闻报道中抽取信息。

20 世纪 80 年代末，信息抽取技术得益于 MUC（Message Understanding Conference）系列会议的召开而蓬勃发展起来。正是 MUC 系列会议使得信息抽取发展成为自然语言处理领域中的一个重要分支。MUC 会议从 1987 年到 1998 年共举行了七届，它由美国国防高级研究计划委员会 DARPA（the Defense Advanced Research Projects Agency）资助。MUC 会议的显著特点是在于对信息抽取系统的测评[26]。

在 MUC 会议中，主要根据两个评价指标衡量信息抽取系统的性能：准确率和召回率[27]。召回率为系统正确抽取的结果占所有可能是正确结果的比例；准确率为系统正确抽取的结果占所有抽取结果的比例。为了综合评价抽取系统的性能，一般情况下还计算准确率 PRE 和召回率 REC 的加权几何平均值，即 F 指数。

MUC 系列会议对信息抽取技术的确立与发展起到了巨大的推动作用。会议中定义的有关于信息抽取的各种规范及评价体系已经逐渐成为信息抽取研究中的标准。

3.2 Deep Web 信息抽取技术

Deep Web 数据抽取是将 Deep Web 查询结果页面作为信息源的一类 Web 数据抽取。目前，Web 数据的来源有三种，分别为自由式、半结构化和结构化[28]。传统的抽取方法都是针对自由式的 Web 信息源，这些信息是以人类语言描述的文本。结构化信息主要指以数据库中的数据为表现的信息，通常可以使用数据库查询语言或数据库全文检索而获取。而半结构化信息的表现形式介于自由式和结构化式之间，通常缺少严格的语法和预定义结构。

通常情况下，Web 上的信息是以半结构化数据形式存在的，因为网页本身具有一定的结构。大部分网页遵从 W3C 制定的 DOM 树形结构标准，这一特性在一定程度上降低了 Web 信息抽取工作的难度，但同时也具有一定的缺点。在网页中，经常会发生数据被标签分割的情况，一条完整的数据中往往会插穿一些对数据无意义的标签，这使得传统的基于自然语言处理的文本信息抽取技术不能够直接移植到 Web 信息抽取领域中。现在国内外对于 Deep Web 信息抽取技术的研究主要集中在：基于 DOM 树结构、基于模板、基于视觉特征和基于统计理论的抽取技术。

3.2.1 基于 DOM 树结构的信息抽取

网页中的 HTML 标签具有可嵌套性，所有标签组成的 DOM 模型呈现树形结构。在 Web 信息抽取过程中，可以针对网页默认的树形结构采用一些常见的操作来总结归纳出待抽取信息的特征。基于 DOM 树结构的抽取技术克服了对 Web 数据源的限制，可以处理各种类型的多正文体和单正文体页面，其操作过程较基于视觉的方法更容易实现。在基于 DOM 树结构的抽取技术领域中已有一些成型的经典算法和系统，使之成为 Deep Web 信息抽取技术中发展较为迅速的一个分支。

基于 DOM 树结构的抽取算法主要有：DSE [29]算法、RoadRunner[30-31]系统、MDR [32-33]算法、XWRAP[34]系统和 Lixto[35-36]系统。

① DSE 算法

DSE 方法将含有有用数据的区域定义为 data-rich 区，算法中通过判断页面中的 data-rich 区而达到抽取主题信息的目的。

DSE 算法首先清除一些内容无关的标签，然后使用迭代算法自顶向下的深度优先遍历两个网页的 DOM 树，同时对比相对应的节点。若两个对应节点相同，则继续比较它们对应的子节点，如果两个对应的子节点相同而且是叶子节点，则删除这两个子节点，否则回溯到它们的父节点，继续比较其他对应的子节点。若一个节点的所有子节点都被删除，那这个节点也要被删除。

DSE 算法存在以下几个问题：

1) 该算法对每个页面都进行重复处理，没有很好利用已处理过的页面的结构，若可以通过总结学习得到一个通用的信息抽取模板，可提高抽取效率；

2) 该算法中认为对于同一网站的页面，只有与页面结构相关的标签才具有相似性，若两个页面只有 text 节点不同，则经过 DSE 算法后，页面中的所有节点都会被删除，这显然不是正确的结果。

② RoadRunner 系统

RoadRunner 系统中的抽取算法是通过比较同一个网站的两个页面 DOM 树模型来得到一个主题数据抽取程序。为了防止因为页面的不规范而造成的失误，RoadRunner 算法需要假设所有需要处理的页面源码是完全遵守 XHTML 规则的，而且假设页面源码已经被处理为标准的标签和字符串格式。

RoadRunner 系统的核心思想是解决两个 DOM 树之间的不匹配问题。不匹配分为标签 mismatch 和字符串 mismatch，而标签 mismatch 又可分迭代项和可选项两种情况去处理。对于同一网站的页面，字符串 mismatch 一般是因为读取的数据库信息有差异，即字符串不匹配的节点正是待抽取的有用信息，此类型节点采用字符串“#PCDATA”标注，标记为待抽取信息。如果出现标签不匹配，又分为两种情况去处理：如图 3.1 所示，在样本页面中的第 20~25 行是标签与上一节点相重复的情况，这时要找到出现不匹配位置的最后一个节点，即 19 行的</LI>，然后从 20 行开始找到第一个与 19 行相同的标签，即 25 行，所以确定 20~25 行可能是一个重复区域。为了验证 20~25 行是否为重复区域，需要从后往前依次验证，先验证的是 19 行和 25 行，然后验证 18 行和 24 行，以此类推，直到验证完整个区域都有与之相匹配的区域，最终证明此区域确实为重复区域，并用“+”做标志；图中的两个页面的第 6 行出现了标签 mismatch，首先判断是否为重复区域，若不是，则认为是由可选项引起的，此时只需要跳过此位置，并用“？”做标志，方便后续处理，然后继续比较之后的节点。

RoadRunner 系统中将标签不匹配细化为两种处理情况。但是算法中没有指出如何获取同一个网站的页面，而且算法在进行比较之前首先是假设页面源码遵守 XHTML 规范，但是在实际应用中，页面中并没有这么完美，多少都会存在一些不规范的节点，因此，RoadRunner 系统如果直接使用也许会出现很多问题，甚至会影响提取精度。而且该算法没有对页面进行预处理，直接对 DOM 树进行对比会降低抽取的执行效率。

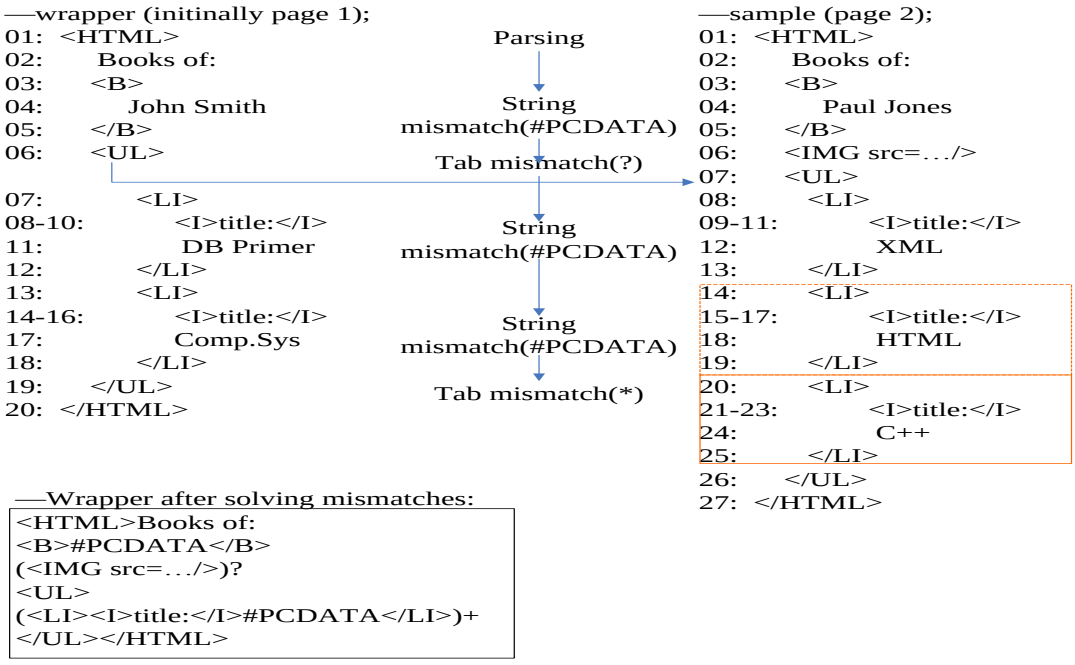


图 3.1 RoadRunner 算法过程

Figure 3.1 RoadRunner Algorithm Process

③ MDR 算法

MDR 算法基于以下两个可以通过观察而得到的规则：

1) 结构相似的数据区域一般固定在页面中的某个版块，并且以相似的结构组织数据，这样的区域称为 data region；

2) 一条完整的数据基本都是分布在同一个子树中。

MDR 算法的核心是判断 data region 的位置。MDR 算法中提出的判断 data region 的算法同时适用于单正文体页面和多正文体页面。但是由于 HTML 相对于数据结构的组织而言更加关注数据的表现形式，因此有时会将毫无关系的内容放到相邻的节点中或是将一条完整的数据分到不相邻的节点中，此时使用 MDR 算法抽取到的数据是不完整的。

④ XWRAP 系统

该系统提供一个可视化的用户界面，用户在页面中指定语义项来添加语义信息，系统针对不同的页面标记使用不同的提取规则，这样可以提高系统的灵活性与性能，最后生成 XML 文档。然而，该系统只适用于网页中明显的区域结构的提取，而且在生成抽取规则时需用户参与选择启发规则。

⑤ Lixto 系统

该系统的最大优点是用户可以以交互式、可视化的方式对页面中的信息进行标记，系统通过用户标记的信息来生成抽取规则。Lixto 系统使用的抽取规则是基于 Datalog [37] 的 Elog 语言描述的。虽然，Lixto 系统通过使用用户交互与可视化界面在一定程度上简化了抽取的步骤，但是，它的抽取规则是基于 Elog 语言的，实现和优化时都比较困难。

3.2.2 基于模板的信息抽取

网络中存在着大量通过读取在线数据库然后将数据填充到统一模板的动态方式生成的页面，针对这些具有模板的页面产生了一种基于模板的信息抽取技术。通过对来自同一模板的页面进行对比，总结出一个通用的信息抽取模板，从而可以避免对众多页面做相同重复的操作。

欧建文[38] 等人提出了一种生成页面信息抽取模板的方法。作者提出三个假设，假设一：URL 树结构中同一个目录节点下的页面是由同一个模板生成的；假设二：由同一个模板生成的页面中的数据布局基本是一致的；假设三：页面中的链接文本是对主题信息的描述。

但是，这种方法只能处理严格按照模板生成的页面，如果页面主题不同，则会被认为是来自不同的模板。而且，这种方法只能抽取一个待抽取信息块，不能处理多正文体页面。

张彦超等人[39] 提出的方法在生成页面模板的同时引入了 URL 模板匹配概念。URL 模板匹配是由待抽取页面的 URL 和 URL 模板匹配库进行匹配，来识别该页面是否可以被解析以及解析该页面所用的模板。

杨少华等人[40-42] 也提出了能够产生抽取模板的方法，但是对模板的定义稍有不同。时达明等人[43] 提出一种针对博文页面的抽取方法，但是具有局限性，只能针对具体的网站进行抽取。

3.2.3 基于视觉特征的信息抽取

HTML 语言是一种语法松散且较灵活的标记性语言，不需要经过编译也可以解释执行，一些语法错误可以被浏览器的容错功能所隐藏，HTML 更注重的是数据的表现而不是数据的结构。HTML DOM 树中属于同一父节点的子节点在浏览器显示时并不一定具有紧密的联系，同样，视觉上联系紧密的节点在 HTML DOM 树结构中也有可能相差很远。因此，对于设计不规范的页面，结合页面的视觉特征来分析主题信息可能更加科学。

微软亚洲研究院的 Cai 等人[44] 最早提出了利用页面视觉特征来抽取主题信息的 VIPS（vision-based page segmentation）算法。该方法在分割页面时有以下几条规则：

① 类似<hr>等标签通常用于分割主题，因此，若页面某一区域中包含这类标签，则分割这个区域；

② 若某一个区域的背景颜色与其子区域的背景颜色不同，则分割这个区域；

③ 若某一个区域中大部分节点是本文节点，则不再分割这个区域。

这些规则在一定程度上能够满足复杂的页面对于算法的要求，但是由于页面设计中的多样性和视觉特征的复杂性，这种基于视觉特征的抽取技术在实现过程中存在许多问题。

3.2.4 基于统计理论的信息抽取

Gupta 等人[45 开发的 Crunch 系统是通过区域中 text/link（普通文本/链接文本）的比值与已定的某个阈值的大小关系来确定页面中的主题区域。作者认为在主题区域中，普通文本所占的比例较大，而在广告区域中，信息会以链接文本的形式展现。但是，Gupta 没有给出具体的阈值，也没有提出确定阈值的方法，如果阈值设定的不合理将会严重影响最后的抽取准确率。

孙承杰等人[46] 也提出了类似的算法，但是仅适用于使用<table>标签来布局的页面。算法中首先找出页面中所有的<table>标签，然后去除这些区域中的所有 HTML 标签，得到不含有任何标签的字符串，将长度大于一定阈值的字符串作为候选主题，最后将所有的候选主题中最长的字符串作为主题信息。<table>标签具

有可嵌套性，因此，在选择最长主题时需要检查其中是否还包含了其他<table>标签。

韩忠明[47] 提出了一种基于行的主题抽取方法，算法中通过每行 text/link 的值与某个阈值的大小关系来确定该行是否为文本行。Song[48]]提出了一种不设定阈值而是以<table>标签作为统计信息的最小容器的算法，但是，该算法只能处理主题信息在一个最小容器内的情况，不适用于多正文体页面。周佳颖等人[49] 对 Song 的算法进行了改进，可以同时处理多正文体页面。

3.3 Deep Web 信息抽取技术分析

基于统计原理的抽取技术主要适用于以文字为主的主题信息，文字部分相对于其他部分在数量上具有明显的优势，而且该技术针对不同类别的页面需要不同的阈值。虽然该技术在理论上容易实现，但是确定一个合理的阈值具有一定的难度，阈值的合理性对于主题区域的确定具有直接的影响。

基于视觉特征的抽取技术过多的依赖于页面的组织结构，因此该技术比较适用于没有太多错误标签的、符合一般页面设计规则的、结构比较清晰的页面。该技术对页面的分块更注重于可视化信息的组织形式，较单纯的考虑页面的标签嵌套结构的技术更合理，但是，由于页面本身的一些视觉分块与信息分块效果不统一、信息结构不规范以及标签错误等众多因素使得该技术的实现过程十分繁琐。

基于视觉特征的抽取技术和基于统计理论的抽取技术在大部分情况下都需要对待抽取页面进行区域划分等处理，需要人工干预，因此，操作人员的主观原因可能会造成区域的划分不合理从而导致信息抽取效果不理想。

基于模板的抽取技术依赖于能够表示待抽取信息的节点串，一般情况下需要针对待抽取页面进行分析与标记，归纳出一个统一的模板。虽然利用模板抽取信息比较方便，但是生成模板的过程需要大量的人工参与，在模板的通用性和生成方法方面还有待于改进。该类技术适用于结构相似度较大的页面，例如通过查询数据库动态生成的页面，而且只能处理单正文体页面。

基于 DOM 树结构的抽取技术针对页面的结构优势，通过对 DOM 树结构的对比来确定主题信息的位置，然后进行信息抽取，很少受到操作者的主观原因的影响。该类技术对于页面的类型没有限制，对来自同一个网站并具有相似结构的页面都可以进行处理。基于 DOM 树结构的抽取技术不需要对待抽取页面进行分块处理，直接通过对比 DOM 树结构获取页面的主题信息区域，但是需要对每个待抽取页面进行相同的处理，没有充分利用已有的结果归纳出针对同一网站的其他相似页面进行处理的统一方法。

综述上述四类抽取技术近些年的发展状况，基于模板与基于 DOM 树结构的抽取技术在复杂性、适用范围和自动化程度等方面更具有优越性。但是由于模板在生成时自动化程度较低，而基于 DOM 树结构的抽取算法又没有归纳出对相似页面进行统一处理的方法，因此，本文结合两类方法的各自优势提出基于 DOM 树结构与模板方法相结合的抽取算法，FIME（Filtering, Iterating, Matching, and Extracting）算法。

3.4 本章小结

本章首先介绍了信息抽取技术的发展历史，然后就现在国内外对于 Deep Web 信息抽取技术的主要研究方向进行了详细的介绍，分别为：基于 DOM 树结构、基于模板、基于视觉特征和基于统计理论的抽取技术，并对四类抽取技术在复杂性、适用范围和自动化程度等方面进行了分析与总结。

4　基于 DOM 树和模板方法相结合的 Deep Web 查询结果抽取技术

通常认为，对于来自同一网站的页面，页面中的数据具有相似的组织结构，因此，本文首先借鉴了 RoadRunner 算法的核心思想，即通过比较来自同一网站的两个页面的 DOM 树结构，获取待抽取数据的位置信息。在页面的 DOM 树结构比较过程中，对于相对应的节点，可能出现标签不匹配与字符串不匹配。当出现字符串不匹配时，通常认为是由于读取数据库中的数据不同，因此，不匹配的字符串正是需要抽取的数据；当出现标签不匹配时，又可以分为两种情况：出现了迭代项或是可选项。迭代项，即对于若干个相邻的节点，如果节点内部嵌套的子标签完全相同，则认为这些节点互为相同的迭代项节点。可选项，即某些只存在于部分页面中的节点。本文通过比较两个页面的 DOM 树结构，发现页面中的标签不匹配和字符串不匹配情况，并进行标记，将标记后的页面作为此网站的页面包装器，即模板 Wrapper，然后利用 Wrapper 对同一网站待抽取页面的 DOM 树进行比较和标记，最后，将标记中的有用信息进行抽取，并存储为 XML 格式。

本文提出了 FIME（Filtering, Iterating, Matching, and Extracting）算法，结合模板方法的思想对基于 DOM 树的抽取算法进行了三点改进，分别为：

① FIME 算法在进行 DOM 树比较之前，首先对页面采取预处理操作，使页面遵守 XHTML 规则，同时清除页面中对于抽取信息无用的标签及部分属性元素，使得页面更精简，以提高后续的匹配算法效率；

② 针对基于 DOM 树结构算法 RoadRunner 中回溯处理页面中迭代项，导致匹配算法复杂度较高的问题，FIME 算法在进行匹配算法之前首先对页面中的迭代项进行合并，降低了后续匹配算法的时间复杂度；

③ 结合基于模板抽取算法的思想，FIME 算法利用在匹配算法中通过比较 DOM 树结构而获得的待抽取数据的位置信息作为同一网站页面的模板 Wrapper，对所有同源页面进行待抽取信息的自动抽取，而不是对同源结构相似的页面做重复的处理，提高信息抽取的效率和自动化程度。

4.1 FIME 算法名词解释

DOM：即文档对象模型，一个能够表示与处理 HTML 文档或是 XML 文档的方法，它能够使用独立于标记语言和运行平台的方式访问、处理以及修改 HTML 或是 XML 文档的内容与结构。

元素节点：一个 HTML 文档或是 XML 文档是由相互嵌套的元素节点组成的一个层次结构，一个元素节点由开始标签和结束标签以及其中的代码组成，对于空元素，则有空元素标签组成。

根节点：一个文档只有一个根节点，其他类型节点都是根节点的孩子节点或孙子节点。

属性节点：元素节点都有若干个与之有关系的属性节点。如果一个元素节点与之相关联的属性节点个数多余一个，该元素节点在排列这些属性节点时不区分先后顺序。

文本节点：文本节点是只包含文本内容的节点，文本内容在 XML 中又称为字符数据。文本内容可以由很多信息组成，也可以是空文本。在文档结构中，元素节点和属性节点的文本内容都是靠文本节点来显示的。

迭代项：对于相邻的若干个元素节点，如果元素节点内部嵌套的所有子标签都相同，则彼此互相称为迭代项节点。即若有相邻的元素节点 A 与 B，a 为 A 节点的所有内部子标签，b 为 B 节点的所有内部子标签，如果 a 等于 b，则称元素节点 A 与 B 为相同的迭代项节点。

可选项：若元素节点 A 与 B 为两个页面 DOM 树结构中相对应的节点，节点 A 的孩子节点分别为：元素节点 a、文本节点 b、元素节点 c；节点 B 的孩子节点分别为：元素节点 d、元素节点 e，此时，文本节点 b 则属于可选项节点，即只存在于部分页面中的节点。

4.2 FIME 算法架构

FIME 算法共包括四个模块，分别为：清噪模块（Filtering）、迭代模块（Iterating）、匹配模块（Matching）和抽取模块（Extracting），算法流程图如图 4.1 所示。首先在清噪模块中对页面进行预处理，包括使用 Tidy 工具对页面正则化以及使用正则表达式清除页面中的无用标签和部分属性节点；然后在迭代模块中对经过清噪模块处理过后页面中的迭代项进行合并；之后将经过合并迭代项的页面输入到匹配模块中，通过对比页面的 DOM 树结构获取待抽取数据的位置信息，即页面包装器 Wrapper（模板）；接下来将所有同源页面输入到清噪模块中进行页面预处理；最后，将经过清噪处理的所有同源页面和 Wrapper 输入到抽取模块中，对所有页面中的待抽取数据进行自动化抽取。

FIME 算法的执行过程如下列步骤所示：

① 将来自同一站点的两个不同页面分别标记为：sample 和 wrapper，在清噪模块中对 sample 和 wrapper 进行页面预处理，包括使用 Tidy 工具对页面进行正则

化，使页面遵守 XHTML 规则，以及使用正则表达式清除已知的对于抽取数据无用的标签和某些属性节点；

② 将经过清噪模块处理过后的 sample 页面和 wrapper 页面输入到迭代模块中，对页面中相同的迭代项进行合并，即只保留相邻迭代项的第一项，并为其添加“iteration”属性节点；

③ 将经过合并迭代项以后的 sample 页面和 wrapper 页面输入到匹配模块中，通过将 sample 页面与 wrapper 页面中的 DOM 树结构作对比，获取该站点的页面模板 Wrapper；

④ 将同源的所有待抽取信息页面输入到清噪模块中，对这些页面做预处理操作，这里执行的页面预处理操作与对 sample 页面和 wrapper 页面所执行的预处理操作相同；

⑤ 将经过清噪模块处理过后的待抽取页面集与页面模板 Wrapper 输入到抽取模块中，利用 Wrapper 对待抽取页面进行有用数据的标记，然后将标记的有用信息进行抽取；

⑥ 将抽取出的信息存储为 XML 格式，以待后续处理。

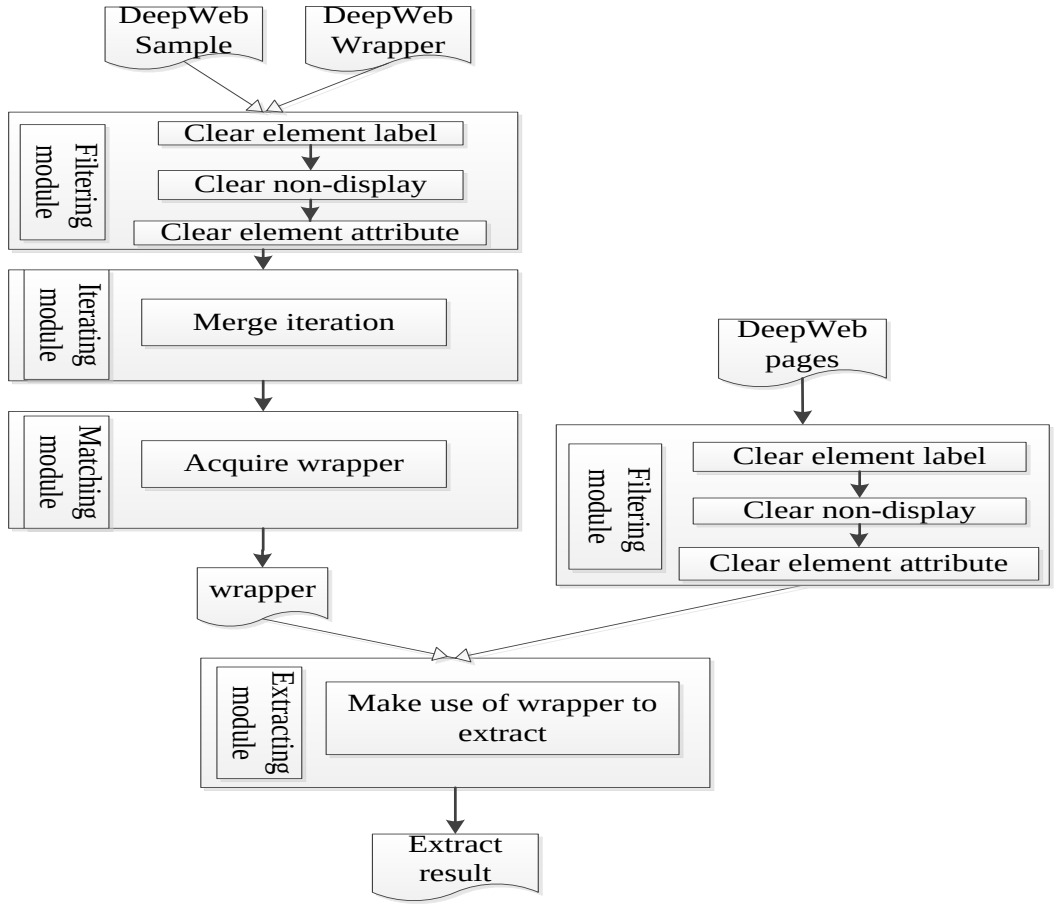


图 4.1 FIME 算法架构图

Figure 4.1 FIME Algorithm Architecture

4.3 清噪模块

在本模块中，首先利用 Tidy 工具对输入的文件进行正则化；然后使用正则表达式对页面进行三项预处理：

① 清除页面中无用的元素节点，包括 JavaScript、CSS 和注释节点；

② 清除不可显示的元素节点，包括空格 节点；

③ 清除无用的属性节点，此时，清除了除属性节点 a、img、meta 之外的所有属性节点。

属性节点 meta 能够提供页面的元信息，而属性节点 img 和 a 则有可能包含有用的数据信息，因此，保留这三个属性节点。对于其他属性节点，由于最终抽取出的数据不需要保留格式，并且由于这些属性节点的存在将会降低匹配模块算法的效率，所以将其全部清除。清噪模块的算法执行流程如图 4.2 所示。

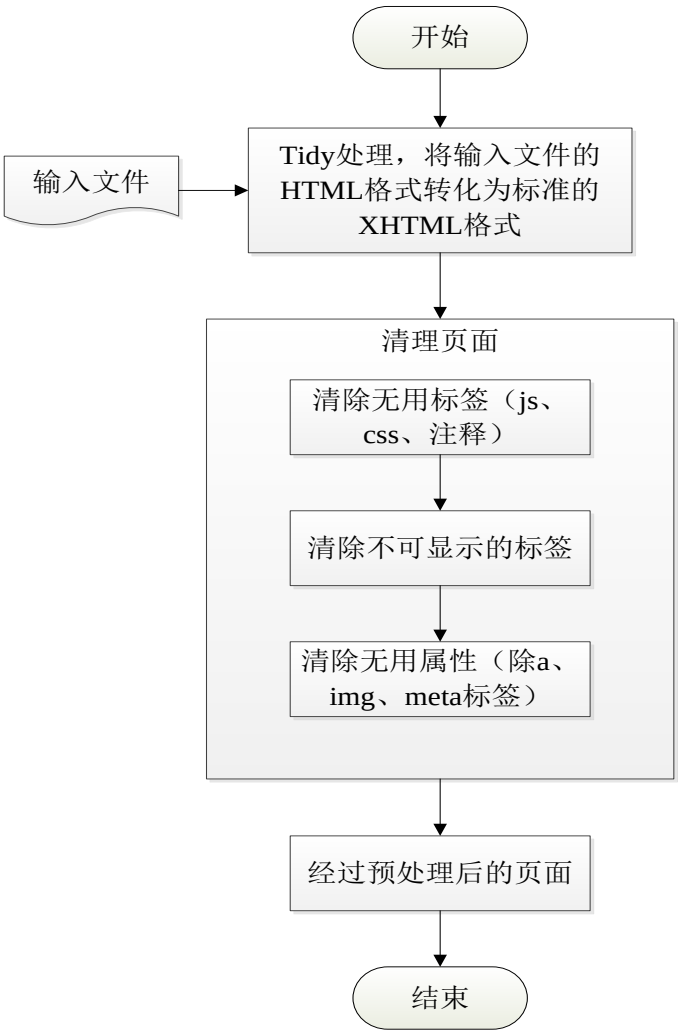


图 4.2 清噪模块流程图

Figure 4.2 The Process of Filtering Module

经过清噪模块处理过后的页面减少了很多对于数据抽取无用的信息，部分页面的大小甚至缩小至原来的 40%-50%，这将有利于提高后续匹配模块算法的效率。

4.4 迭代模块

本模块的功能是将经过清噪模块处理过后页面中相同的迭代项进行合并。在 RoadRunner 算法中，采用回溯法判断相邻的节点间是否为迭代项，当页面 DOM 树结构中含有大量的迭代项时，RoadRunner 中匹配算法的时间复杂度将会非常高。因此，为了解决上述问题，本文提出在对经过清噪后的 sample 页面和 wrapper 页面的 DOM 树结构进行比对之前首先将相同的迭代项进行合并，此时，相同的迭代项只留下一项，并且为这一项添加属性节点“iteration”，这样可以降低后续匹配模块算法的时间复杂度，同时没有影响到抽取模块算法的准确率。

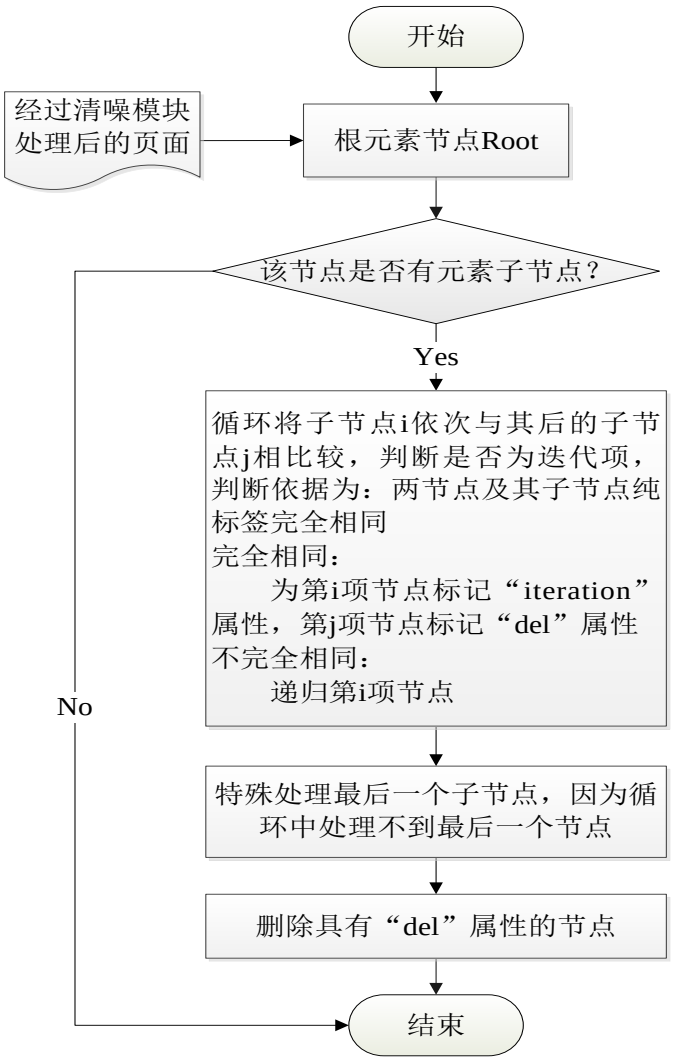


图 4.3 迭代模块流程图

Figure 4.3 The Process of Iterating Module

迭代模块的算法流程图如图 4.3 所示。算法主要思想如下，从根节点的第一个孩子节点开始执行：

① 对相邻的孩子节点比较其节点类型是否相同，如果相同，则转至②执行，否则，转至③执行；

② 当节点类型相同时，本文认为两个节点都是元素节点，因为相邻的文本节点会合并为一个文本节点。然后比较这两个相邻的元素节点的内部子标签是否相同，如果内部子标签相同，则这两个节点为相同的迭代项，此时，为第一个元素节点添加属性节点“iteration”，为第二个元素节点添加属性节点“del”(这样的元素节点将会被删除)；然后，将第一个元素节点与第二个元素节点的下一个节点转至①比较其节点类型是否相同；如果内部子标签不相同且没有属性节点“iteration”，转至①递归执行第一个元素节点的孩子节点，同样从第一个孩子节点开始执行，返回后将第二个元素节点与其后面相邻节点进行比较，转至①执行；

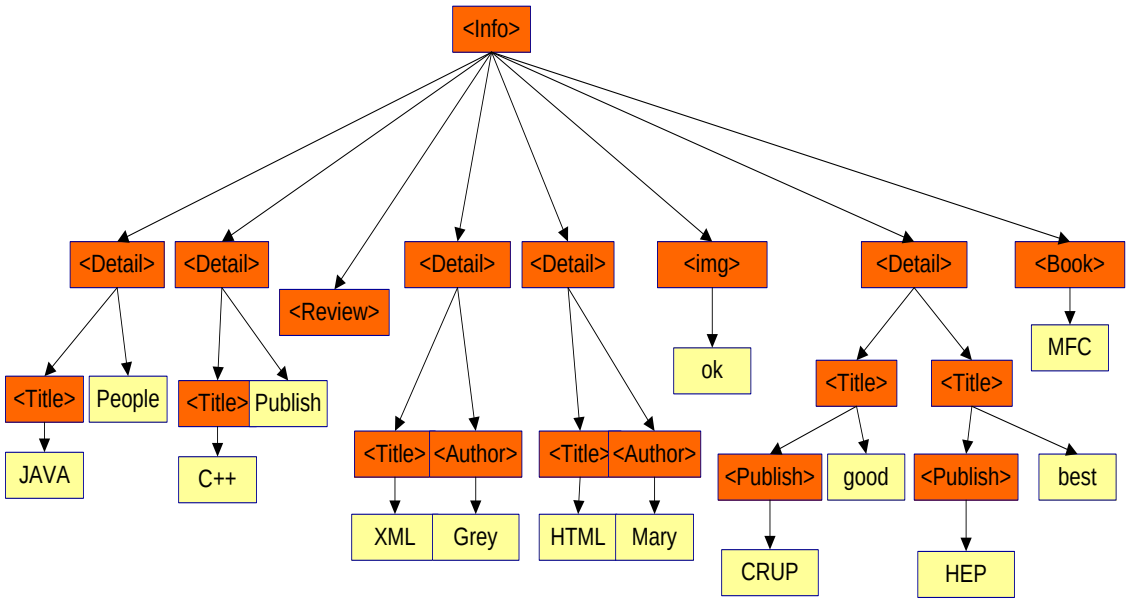
③ 如果节点类型不相同，则说明一个是元素节点，一个是文本节点，如果第一个节点是元素节点，则转至①递归执行其孩子节点，同样从第一个孩子节点开始执行，返回后将第二个节点与其后面相邻节点进行比较，转至①执行；如果第一个节点是文本节点，则不予处理该文本节点，然后将第二个节点与其后面相邻节点进行比较，转至①执行；

④ 对于最后一个孩子节点需要单独处理，因为，此孩子节点没有可以进行比较的下一个节点，如果此节点是元素节点并且没有“del”属性节点，则说明此节点没有与前面相邻的节点是相同的迭代项，此时，需要判断该元素节点内部的子节点中是否有迭代项，所以转至①递归处理其孩子节点，同样从第一个孩子节点开始执行。

⑤ 所有递归都结束后，迭代算法结束，此时，删除所有具有“del”属性节点的元素节点。

本模块中，只对最外层的迭代项进行了合并处理，即如果一个元素节点具有了“iteration”属性节点，则不会再去判断该节点内部的子节点中是否有相同的迭代项，具有“iteration”属性节点的元素节点也将是数据抽取时的最小处理单位。

图 4.4 所示为经过清噪模块处理过后页面的 DOM 树结构，而图 4.5 所示为图 4.4 中的页面经过迭代模块处理过后的 DOM 树结构，通过图 4.4 和图 4.5 的对比，可以看到迭代模块的处理效果。

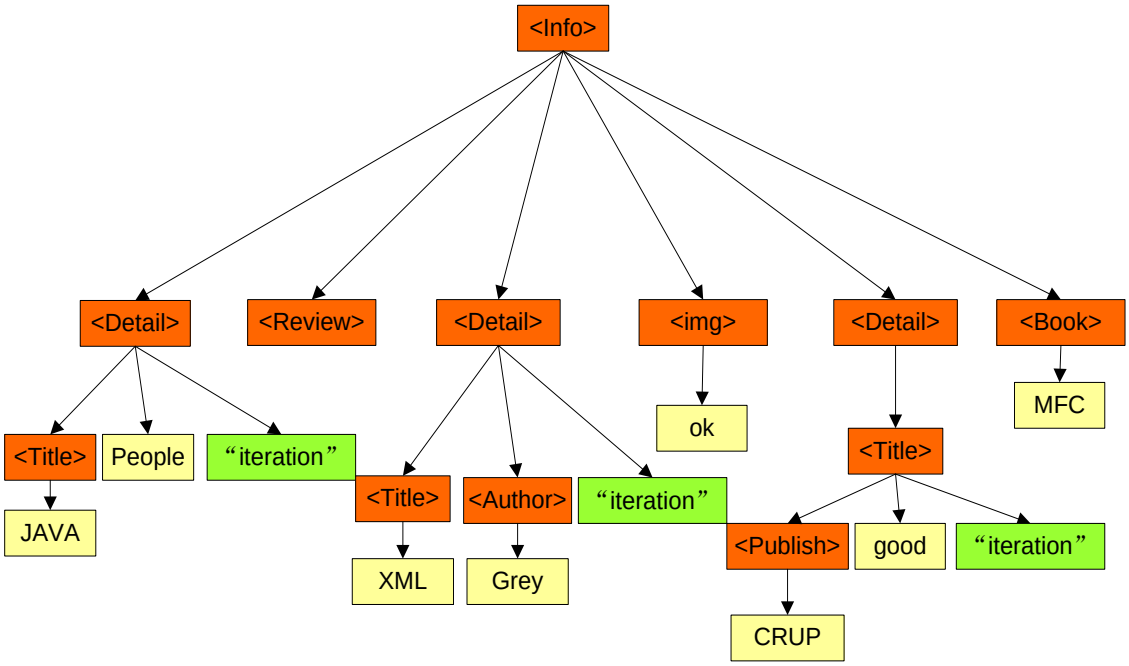


（注：具有<>标志的为元素节点，具有“”标志的为属性节点，无标识的为文本节点）

图 4.4 未经迭代模块处理的 DOM 树结构

(Notation: the node has <> sign is element node, has “” sign is attribute node, the node which does not have sign is text node).

Figure 4.4 The Page is without Iterating



（注：具有<>标志的为元素节点，具有“”标志的为属性节点，无标识的为文本节点）

图 4.5 经过迭代模块处理后的 DOM 树结构

(Notation: the node has <> sign is element node, has “” sign is attribute node, the node which does not have sign is text node).

Figure 4.5 the DOM Tree after Iterated

由图 4.5 可以看到，经过迭代模块处理过后的 DOM 树结构中，相邻的迭代项只留下一项，并且具有“iteration”属性节点，DOM 树结构更精简，在后续匹配模块算法中不需要再回溯处理迭代项，这将降低匹配模块算法的复杂度。

4.5 匹配模块

本模块的作用是将经过迭代模块处理过后的两个页面 sample 和 wrapper 的 DOM 树结构进行对比，从而获得页面模板 Wrapper。在一一对比过程中，本模块将会对页面中的节点添加四种属性节点：“uf”（有用项）属性节点、“ad”（广告项）属性节点、“textuf”（文本有用项）属性节点和“textad”（文本广告项）属性节点，而为节点添加何种属性节点，则是根据相对应节点的 OuterXml 的相似度来确定的。匹配模块的算法流程图如图 4.6 所示。

匹配模块的主要思想如下，从 sample 页面和 wrapper 页面 DOM 树结构的两个根节点开始执行：

① 如果这两个根节点的元素名称不相同或是孩子节点个数不相同，则为这两个根节点添加属性节点“uf”，因为如果对应的节点元素名称不相同，则意味着这两个元素节点是有用的信息，而如果孩子节点个数不相同，由于已经经过了合并迭代项的处理，所以此时孩子节点个数不相同一定是因为孩子节点中存在可选项，由于这里无法识别可选项是否为有用信息，所以，为了不影响抽取数据的召回率，此时将该节点标记为有用节点；否则，如果这两个根节点的 OutXml 相同，OutXml 是一个元素节点中包括标签、文本和属性的所有内容的总字符串，OutXml 相同则说明这两个节点完全相同，即为广告项，因此，此时为这两个根节点添加属性节点“ad”；如果 OutXml 不相同，并且至少其中一个根节点具有“iteration”属性节点，则为这两个根节点添加属性节点“uf”，因为 OutXml 不相同则意味着这两个节点内部有不相同的地方，也就是有有用的信息，而具有“iteration”属性节点的元素节点是本算法中处理的最小单位，所以为这两个根节点添加属性节点“uf”，如果这些条件都不满足，则转至②执行；

② 如果这两个根节点都没有“iteration”属性节点，则判断这两个根节点的内部子标签是否完全相同，如果相同且非空则转至③执行，否则转至④执行；

③ 比较这两个根节点的 InnerText 是否相同，InnerText 是一个元素节点的所有子文本节点的串联值，如果相同，则转至⑤执行，否则转至⑥执行；

④ 遍历这两个根节点的子节点，对于各个相对应的子节点，如果都是元素类型，则将这两个子节点作为根节点转至①递归执行；如果都是文本节点，则将文本字符串各自累加，以便后续判断文本是有用项还是广告项，最后会将相对应的

节点下的累加字符串进行比较，如果相同，则为该两个节点添加属性节点“textad”，如果不相同，则为该两个节点添加属性节点“textuf”，对累加字符串进行比较而不是对相对应的单个文本节点进行比较的原因是，有时相对应的单个文本节点字符串虽然相同，但是它在父节点中是有用的，这样对整个字符串进行比较可以避免误删有用的字符串；如果一个是元素节点，一个是文本节点，当出现这种概率极少的情况时，为了不影响召回率，将会为这两个根节点添加属性节点“uf”；

⑤ 如果根节点为属性元素，即为图片或链接，则为这两个根节点添加属性节点“uf”，此时 OutXml 不相同，内部子标签和 InnerText 都相同，则不相同的只可能是属性节点，所以添加“uf”属性节点，如果根节点不是属性元素，则转至④执行；

⑥ 判断这两个根节点有无元素子节点，如果没有元素子节点，此时 InnerText 又不相同，则说明这两个节点本身就是有用的文本信息，因此，为这两个根节点添加属性节点“uf”，如果有元素子节点，则转至④执行；

⑦ 所有递归都结束后，匹配算法结束。

匹配算法结束后，具有属性节点“uf”、“textuf”、“ad”和“textad”的页面 sample 和 wrapper 就是该网站的页面模板 Wrapper。

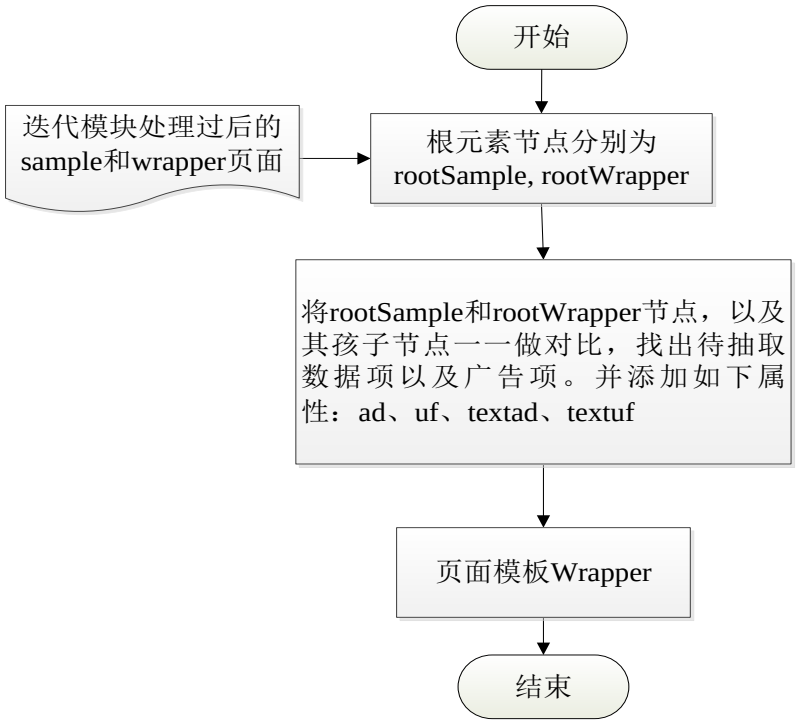


图 4.6 匹配模块流程图

Figure 4.6 the Process of Matching Module

图 4.7 所示为经过清噪模块处理过后 sample 页面和 wrapper 页面的 DOM 树结构，而图 4.8 所示为图 4.7 中的页面经过迭代模块和匹配模块处理过后的 DOM 树结构，通过两图的比较，可以看到迭代模块和匹配模块的处理效果。

Sample

```
<html>
  <p>
    <span>在全部图书中搜索</span>
      C++
  </p>
  <ul>
    <li>
      <IMG alt="C++ Primer" src="http://img32/l.jpg">
      <I>C++ Primer</I>
    </li>
    <li>
      <IMG alt="thinking in C++" src="http://img32/
2.jpg">
      <I>thinking in C++</I>
    </li>
    <li>
      <IMG alt="visual C++ 2010" src="http://img32/
3.jpg">
      <I>visual C++ 2010</I>
    </li>
  </ul>
  <div>
    For more information of
    <span>C++</span>
    please Google
  </div>
</html>
```

Wrapper

```
<html>
  <p>
    <span>在全部图书中搜索</span>
      Java
  </p>
  <ul>
    <li>
      <IMG alt="thinking in java" src="http://img33/
1l.jpg">
      <I>thinking in java</I>
    </li>
    <li>
      <IMG alt="Effective Java" src="http://img33/
22.jpg">
      <I>Effective Java</I>
    </li>
  </ul>
  <div>
    For more information of
    <span>java</span>
    please Google
  </div>
</html>
```

图 4.7 只经过清噪模块处理的 sample 页面和 wrapper 页面

Figure 4.7 the Sample and Wrapper Pages are after Filtering

在 sample 页面中，<ul>节点下有三个内部嵌套子标签相同的<li>节点，在 wrapper 页面中有两个内部嵌套子标签相同的<li>节点，因此，在迭代模块中，将会对两个页面中的迭代项进行合并，合并后的结果是，两张页面中的<ul>节点均只有一个<li>孩子节点，但是该<li>孩子节点有一个“iteration”属性节点。

经过迭代模块处理过后的页面 sample 和 wrapper 输入到匹配模块中，进行 DOM 树结构的匹配比较。首先，两个页面中的节点具有相同的 OutXml，因此，为节点添加“ad”属性节点；然后由于<p>节点下的文本字符串

“C++”和“Java”不相同，因此，为<p>节点添加“textuf”属性节点；当比较到<li>节点时，由于<li>节点具有“iteration”属性节点，具有该属性节点的元素节点是本算法进行处理的最小单位，因此，为<li>节点添加“uf”属性节点；在节点 中，节点内部只有文本节点，并且文本不相同，因此，为节点添加“uf”属性节点；而由于节点 中的文本字符串串联值相等，因此，为 节点添加“textad”属性节点。如图 4.8 所示。

Sample

```
<html>
  <p textUf="1">
    <span ad="1">在全部图书中搜索</span>
      C++
  </p>
  <ul>
    <li iteration="1" uf="1">
      <img alt="C++ Primer" src="l.jpg" />
      <i>C++Primer</i>
    </li>
  </ul>
  <div textAd="1">
      For more information of
    <span uf="1">C++</span>
      please Google
  </div>
</html>
```

Wrapper

```
<html>
  <p textUf="1">
    <span ad="1">在全部图书中搜索</span>
      Java
  </p>
  <ul>
    <li iteration="1" uf="1">
      <img alt="thinking in java" src="1l.jpg" />
      <i>thinking in java</i>
    </li>
  </ul>
  <div textAd="1">
    For more information of
    <span uf="1">java</span>
    please Google
  </div>
</html>
```

图 4.8 经过迭代模块和匹配模块处理过后的 sample 页面和 wrapper 页面

Figure 4.8 the Sample and Wrapper Pages are after Iterating and Matching

4.6 抽取模块

抽取模块的功能是利用在匹配模块中生成的页面模板 Wrapper 对来自同一站点的待抽取页面集进行有用数据的抽取，页面模板 Wrapper 就是在匹配模块中进行了四种属性标记的页面 sample 或页面 wrapper，本文使用的是页面 wrapper。

在抽取模块中，首先利用页面模板 wrapper 为待抽取页面集中的每一个文件 file 中的节点添加“uf”，“ad”，“textuf”或“textad”属性节点，然后抽取 file 中具有“uf”或“textuf” 属性节点的元素节点，最后将抽取出的数据进行格式化的存储。

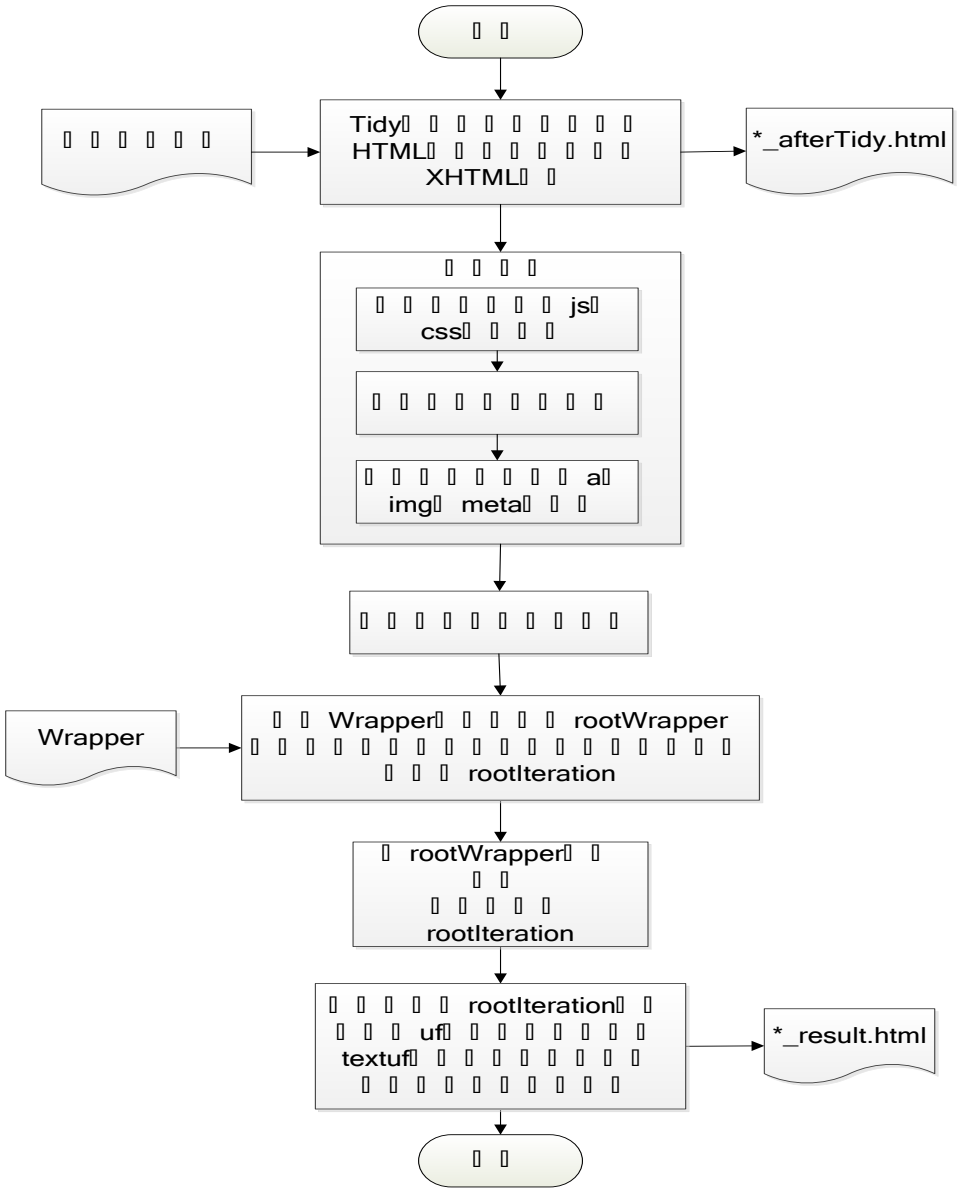


图 4.9 抽取模块流程图

Figure 4.9 the Process of Extracting Module

抽取模块的算法流程图如图 4.9 所示。抽取模块中为每一个 file 文件添加相应属性节点的主要思想如下，从 Wrapper 页面和 file 页面 DOM 树结构的两个根节点开始执行：

① 如果 Wrapper 页面中对应的节点有“uf”，“ad”，“textad”或“textuf”属性节点，则为 file 页面中相对应的节点添加同样的属性节点，否则比较这两个对应节点的孩子节点个数是否相同，如果相同，则转至②执行，如果不相同，转至③执行；

② 一一对应的比较其孩子节点，如果对应的两个孩子节点都是元素节点，则以这两个孩子节点为根节点转至①递归执行，如果都是文本节点，暂不处理，因

为后续会为 file 页面中该文本节点的父节点添加“textuf”或“textad”属性节点，如果一个是文本节点一个是元素节点，而且 file 页面中的孩子节点是元素节点，则为该元素节点添加属性节点“uf”；

③ 当对应节点的孩子节点个数不相同时，需要循环比较其孩子节点的相似情况。在比较的过程中，如果两个孩子节点的内部子标签完全相同，则为这两个孩子节点标记成功匹配过，然后以这两个孩子节点为根节点转至①递归执行，最后根据 Wrapper 页面中的孩子节点是否具有“iteration”属性节点来决定是否进行下一次循环，即如果 Wrapper 页面中的孩子节点具有“iteration”属性节点，则将该孩子节点的内部子标签与 file 页面中孩子节点的下一个相邻节点的内部子标签进行比较，而如果 Wrapper 页面中的孩子节点没有“iteration”属性节点，则将 Wrapper 页面中的该孩子节点的下一个相邻孩子节点的内部子标签与 file 页面中未成功匹配过的第一个孩子节点的内部子标签进行比较；如果两个孩子节点的内部子标签不相同，则需根据 Wrapper 页面中的孩子节点是否已经成功匹配过来决定是否进行下一次循环，即如果 Wrapper 页面中的孩子节点没有成功匹配过，则将该孩子节点的内部子标签与 file 页面中的孩子节点的下一个相邻节点的内部子标签进行比较，而如果 Wrapper 页面中的孩子节点已经成功匹配过，则将该孩子节点的下一个相邻节点的内部子标签与 file 页面中未成功匹配过的第一个孩子节点的内部子标签进行比较。

如图 4.10 所示，左面的页面是经过清噪模块处理过后的待抽取页面，而右面是该页面在抽取模块中被添加了各种属性节点后的结果。

The page is not marked by wrapper	The page is marked in the extract module
<html> 在全部图书中搜索 XML XML Schema XML 入门经典 For more information of XML please Google </html>	<html> 在全部图书中搜索 XML <li uf="1"> XML Schema <li uf="1"> XML 入门经典 For more information of XML please Google </html>

图 4.10 抽取模块标记效果图

Figure 4.10 the Page Marked in Extracting Module

4.7 基于 Hadoop 的 FIME 算法设计与实现

由于 Deep Web 查询返回的结果页面数量众多，因此，本文将 FIME 算法部署在分布式系统基础架构 Hadoop 中，FIME 分布式算法执行流程如图 4.11 所示。

在 FIME 分布式算法中，有两个 MapReduce 执行过程，在 MapReduce1 中，执行的是清噪模块，即将所有的待抽取页面进行清噪处理后，再存放到 HDFS 文件系统中；然后，读取 HDFS 中的两个经过清噪处理的页面，输入迭代模块中，合并页面中的迭代项，再将经过合并后的页面输入匹配模块中，进行 DOM 树结构的比较，最后获得页面模板 Wrapper，此时，将 Wrapper 存放到内存中；最后，将 MapReduce1 输出的经过清噪处理后的页面和 Wrapper 输入到 MapReduce2 中，执行抽取模块，最终，MapReduce2 将抽取后的页面数据存放到 HDFS 中。

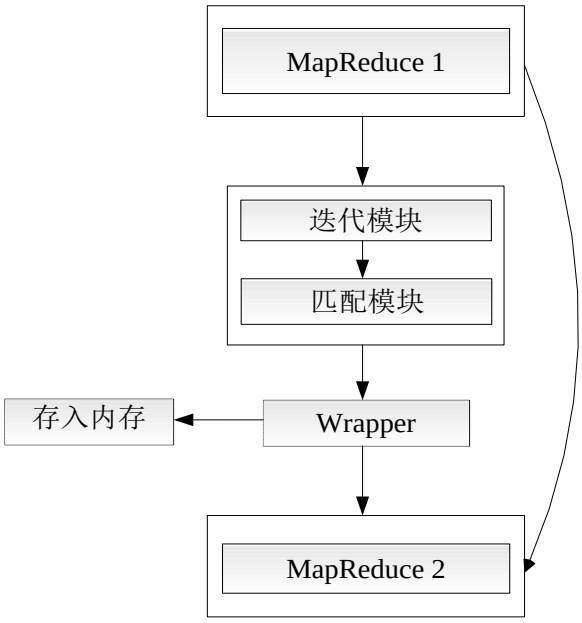


图 4.11 FIME 分布式算法执行流程

Figure 4.11 FIME Distributed Algorithm Execution Process

4.7.1 清噪模块分布式执行算法

Deep Web 查询返回的结果页面分布式的存储在 HDFS 中的数据服务器中。清噪模块的分布式算法是将存储在 HDFS 中的查询结果页面进行页面预处理，然后再分布式的存储在 HDFS 中，其 MapReduce 执行过程如图 4.12 所示。

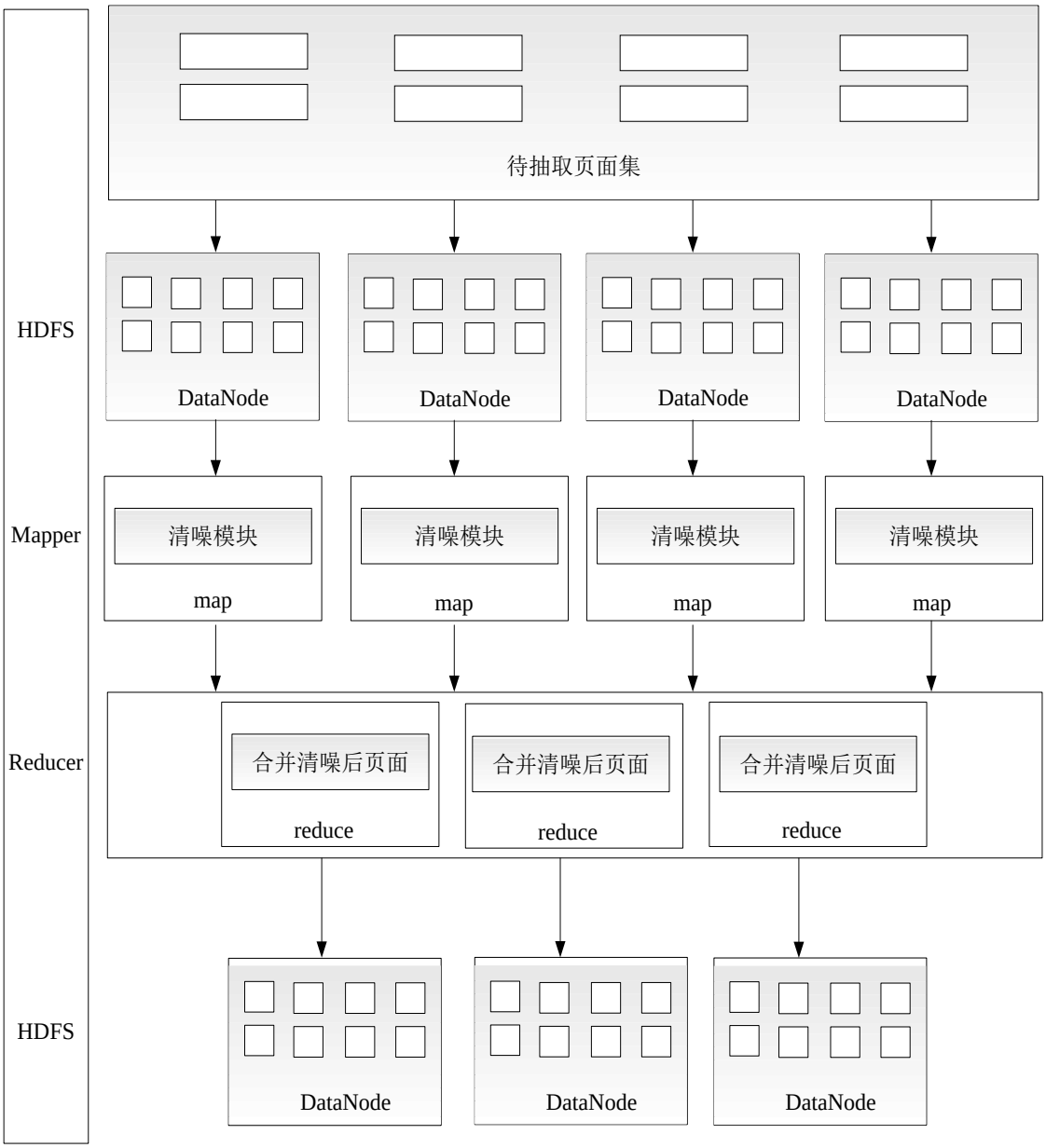


图 4.12 清噪模块 MapReduce 实现

Figure 4.12 Filtering Module MapReduce Implementation

清噪模块的分布式执行流程主要可以分为以下两个过程：

① MAP 过程

MAP 过程处理的是单元块数据，但是 map 函数每次只能处理块数据的一行文本，因此，在对页面进行预处理之前首先需要将页面源码转换为一行文本，本文将页面源码读取并存储在一个字符串 Content 中，并且自动生成字符串 Content 的序列号 Line。

输入：格式为<Key，Value>键值对，实际应用中为<Line，Content>，此时，Line 是自动生成的页面源码序列号，Content 为页面源码内容。

处理：在 Map 任务中将执行清噪模块，对页面进行预处理，清除页面中无用以及不显示的标签和属性元素等。

输出：格式为<Key，Value>键值对，实际应用中为<Line，AfterFilterData>，此时，Line 是经过清噪模块处理过后页面源码的序列号，而 AfterFilterData 是经过清噪模块处理过后页面的源码内容。

MAP 过程输出之后，会执行一个 Sort 过程，Sort 会将 MAP 的输出结果按照 Line 值进行自然排序。Sort 过程排序规则可以进行自定义。

② Reducer 过程

Reducer 过程是对 MAP 过程的输出结果进行处理。在 Reducer 过程之前，还会进行一个 Partition 过程，这个过程是将 Sort 过程之后的所有数据按照 Key 值进行划分，可以通过 Job.setPartitionClass 自定义划分算法。经过 Partition 过程处理之后，某一范围的 Key 值数据被发送到同一个 Reducer 任务，简化了任务的处理过程。

输入：格式为<Key，Value>键值对，实际应用中为<Line，AfterFilterData>，此时，Line 是经过清噪模块处理过后页面源码的序列号，而 AfterFilterData 是经过清噪模块处理过后页面的源码内容。

处理：在 Reduce 任务中，进行的操作是将经过清噪模块处理过后的页面进行合并。

输出：格式为<Key，Value>键值对，实际应用中为<Line，AfterFilterData>。

经过 Reducer 过程处理之后，得到经过清噪模块处理过后的页面集，并将页面集存储到 HDFS 中。

MapReduce 框架键值对与实际应用中键值对的对应值如图 4.13 所示。

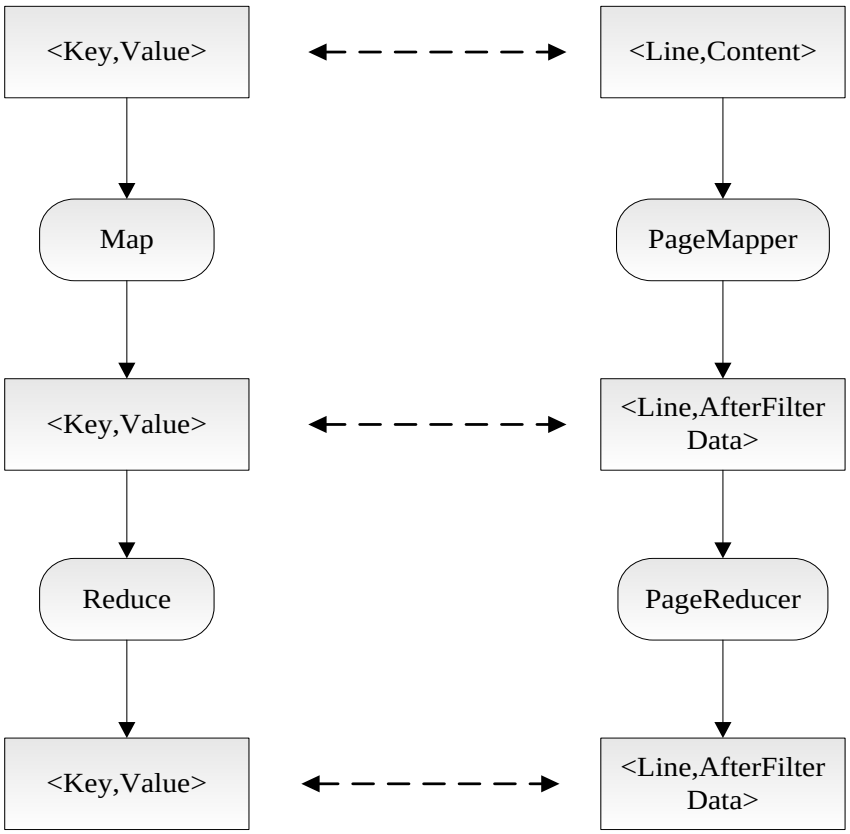


图 4.13 Key Value 对应值

Figure 4.13 Key Value Corresponding Values

4.7.2 抽取模块分布式执行算法

在抽取模块分布式算法中，将经过清噪模块处理后的页面与页面模板 Wrapper 进行对比，对待抽取信息进行标注以及抽取，最后将抽取出的有用信息再次存储到 HDFS 文件系统中，其 MapReduce 执行过程如图 4.14 所示。

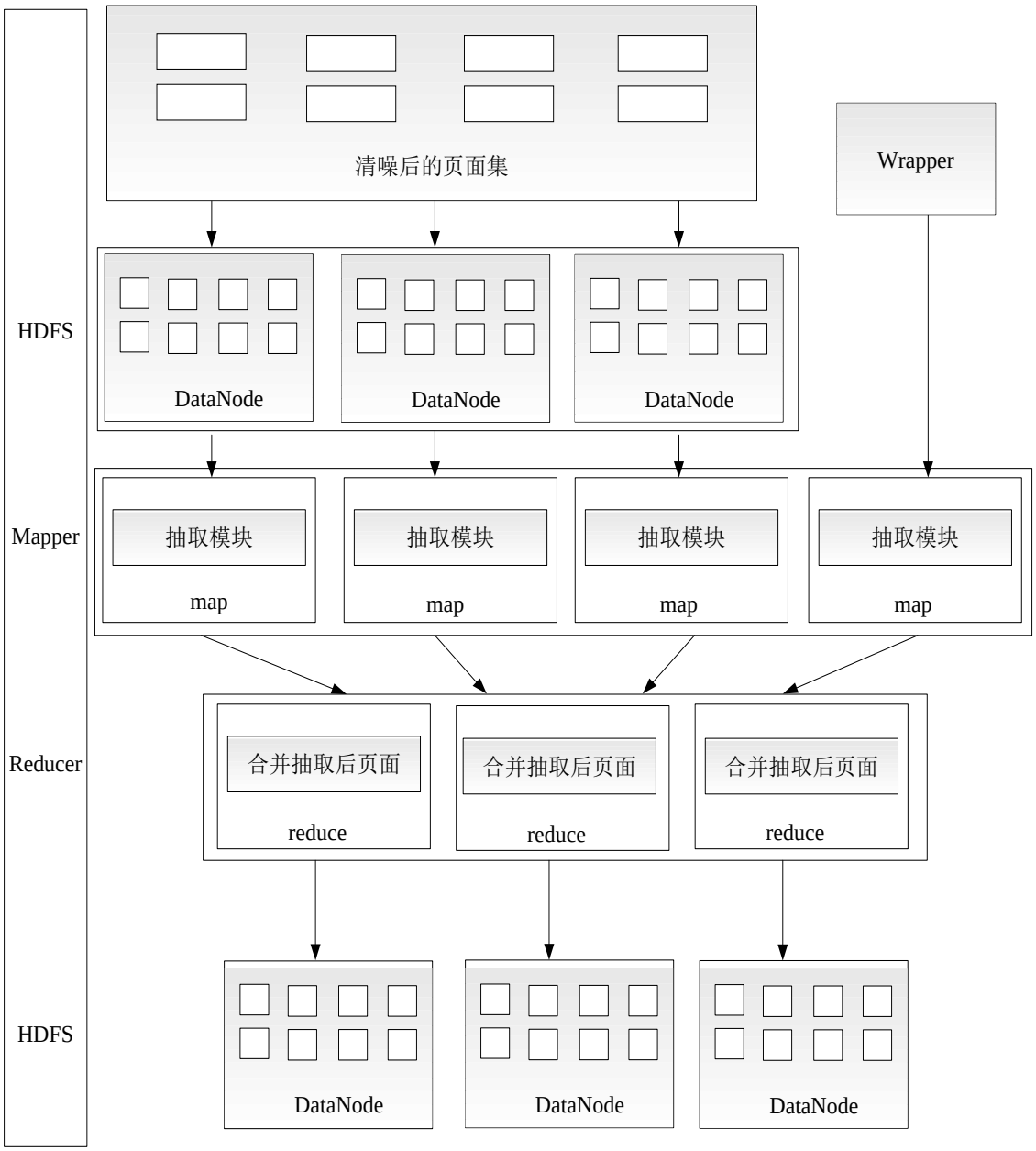


图 4.14 抽取模块 MapReduce 实现

Figure 4.14 Extracting Module MapReduce Implementation

抽取模块的分布式执行流程同样主要可以分为以下两个过程：

① MAP 过程

在 Map 任务中，执行的是抽取模块，同时还需读取内存中的页面模板 Wrapper，根据 Wrapper 对待抽取信息进行标注和抽取。

输入：格式为<Key，Value>键值对，实际应用中为<Line，AfterFilterData >，此时，Line 是经过清噪模块处理过后页面源码的序列号，AfterFilterData 是经过清噪模块处理过后页面的源码内容。

处理：在 Map 任务中将执行抽取模块，同时读取内存中的页面模板 Wrapper，利用 Wrapper 中的标签属性对待抽取页面中的节点属性进行标注，最后将抽取出的数据格式化为 XML 文件。

输出：格式为<Key，Value>键值对，实际应用中为<Line，DataContent >，此时，Line 是经过抽取模块处理后最终信息页面源码的序列号，而 DataContent 是抽取出的数据信息的源码内容。

② Reducer 过程

Reducer 任务中进行的主要操作是将抽取出的数据页面进行合并整理。

输入：格式为<Key，Value>键值对，实际应用中为<Line，DataContent >。

处理：在 Reduce 任务中，进行的操作是将经过抽取模块处理过后的页面进行合并整理。

输出：格式为<Key，Value>键值对，实际应用中为<Line，DataContent >。

经过 Reducer 过程处理之后，得到抽取模块处理过后的最终页面集，并将页面集存储到 HDFS 文件系统中。

MapReduce 框架键值对与实际应用中键值对的对应值如图 4.15 所示。

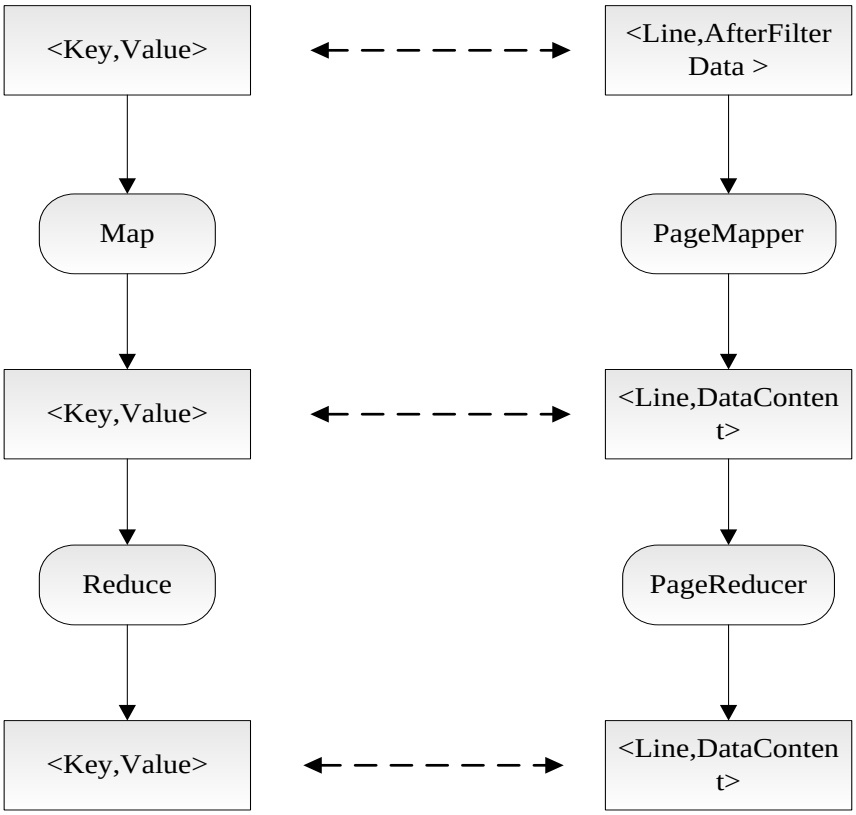


图 4.15 Key Value 对应值

Figure 4.15 Key Value Corresponding Values

综上所述，Hadoop 的 HDFS 文件系统可以为待抽取页面集与抽取出的数据页面集提供存储服务，而 MapReduce 并行框架中的 Map 函数与 Reduce 函数可以实现 FIME 算法的分布式执行操作。

4.8 本章小结

本章首先介绍了 FIME 算法的核心思想以及针对已有的基于 DOM 树结构与模板的抽取技术提出的三点改进，第一点：FIME 算法在进行 DOM 树结构比较之前，首先对页面采取预处理操作，使页面遵守 XHTML 规则，同时清除页面中对于抽取信息无用的标签及部分属性元素；第二点：针对基于 DOM 树结构抽取算法 RoadRunner 系统中回溯处理页面中迭代项，导致匹配算法复杂度较高的问题，FIME 算法在进行匹配算法之前首先对页面中的迭代项进行合并；第三点：结合基于模板抽取算法的思想，FIME 算法利用在匹配算法中通过比较 DOM 树结构而获得的待抽取数据的位置信息作为同一网站页面的模板 Wrapper，对所有同源页面进行待抽取信息的自动抽取，而不是对同源结构相似的页面做重复的处理。

然后重点阐述了 FIME 算法架构，FIME 算法中包含四个模块，分别为：清噪模块、迭代模块、匹配模块和抽取模块，并且对每个模块的思想和执行效果进行了详细的分析。首先，在清噪模块中对页面进行预处理，包括使用 Tidy 工具对页面正则化以及清除页面中的无用标签和无用属性元素；然后在迭代模块中对经过清噪模块处理的页面中的迭代项进行合并；之后将经过合并迭代项的页面输入到匹配模块中，通过对比页面的 DOM 树结构获取待抽取数据的位置信息，即页面包装器 Wrapper（模板）；接下来将同源的所有页面输入到清噪模块中进行页面预处理；最后，将经过清噪处理的所有同源页面和 Wrapper 输入到抽取模块中，对所有页面中的待抽取数据进行自动化抽取。

最后介绍了 FIME 算法在 Hadoop 平台中的分布式执行框架，HDFS 文件系统可以为待抽取页面集与抽取出的数据页面集提供存储服务，而 MapReduce 并行框架中的 Map 函数与 Reduce 函数可以实现 FIME 算法的分布式执行操作。

5 实验设计与结果分析

5.1 实验数据与评价指标

本文采用的实验数据分为两大部分：一部分为 RoadRunner 算法携带的三个数据集，此部分数据集是为了保证 FIME 算法与 RoadRunner 算法在准确率和召回率两个方面比较时的客观性；另一部分是来自雅虎音乐、雅虎新闻、智联招聘、新浪新闻、前程无忧、CSDN 搜索、亚马逊、搜狗新闻和当当九个知名网站的 448M 页面数据，这些网站的网页结构差别很大，有助于充分验证 FIME 算法的准确性和高效性。

本文使用三个评价指标衡量 FIME 算法的性能：准确率、召回率和 F 指数[25]。召回率是系统正确抽取的数据占所有可能是正确数据的比例；准确率是系统正确抽取的数据占所有抽取出的数据的比例。为了综合评价抽取算法的性能，一般情况下还计算准确率 PRE 和召回率 REC 的加权几何平均值，即 F 指数。

5.2 集群环境

本文采用的实验环境详细信息如下：

① Hadoop 集群由 12 台组成，每台 PC 机 CPU 为 Intel(R) Pentium(R) 4 CPU 2.80GHz，内存为 1.5G，硬盘为 80G。其中 1 台作为集群 Master，11 台作为集群 Slaves，分别编号为 Master、SlaveA-K。

② 每台机器的运行环境：操作系统：Ubuntu 12.04.1；Hadoop 版本：1.0.4；JDK 版本：1.7.0_07。

5.3 实验结果及分析

实验一：采用 RoadRunner 算法携带的三个数据集，Players，Hotjobs 和 Overstock。FIME 算法与 RoadRunner 算法在准确率，召回率，F 均值以及抽取数据的总时间对比图如下图 5.1，图 5.2，图 5.3 和图 5.4 示：

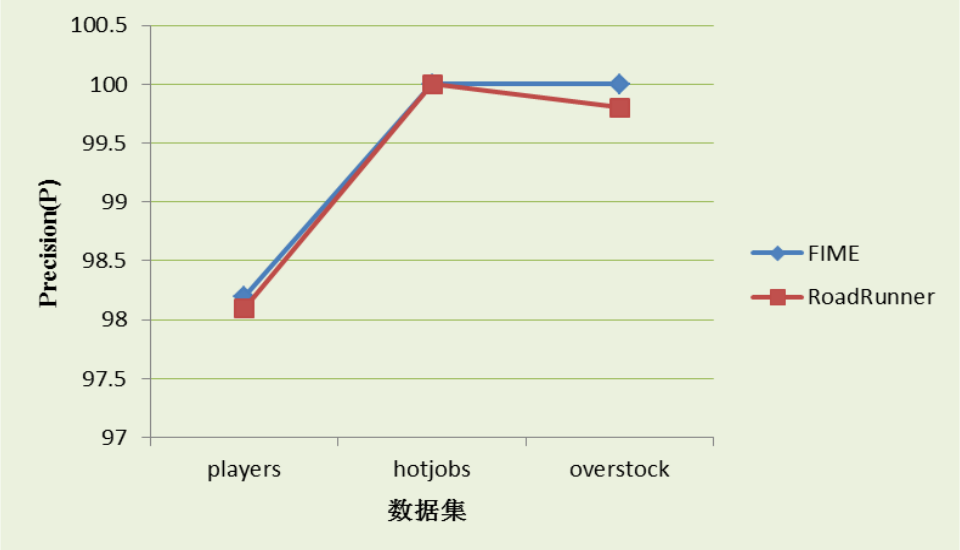


图 5.1 FIME 算法与 RoadRunner 算法准确率对比图

Figure 5.1 the precision comparisons of FIME and RoadRunner

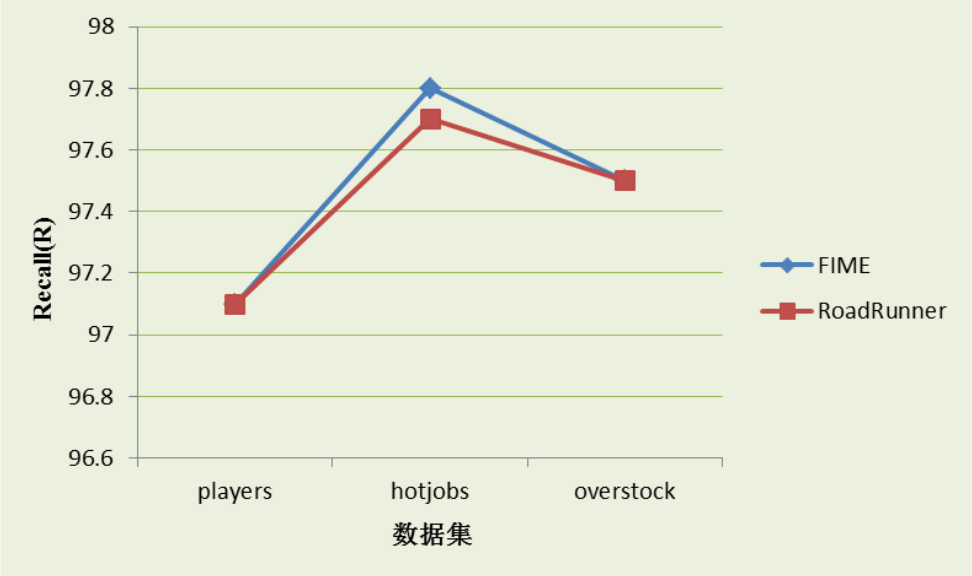


图 5.2 FIME 算法与 RoadRunner 算法召回率对比图

Figure 5.2 the recall comparisons of FIME and RoadRunner

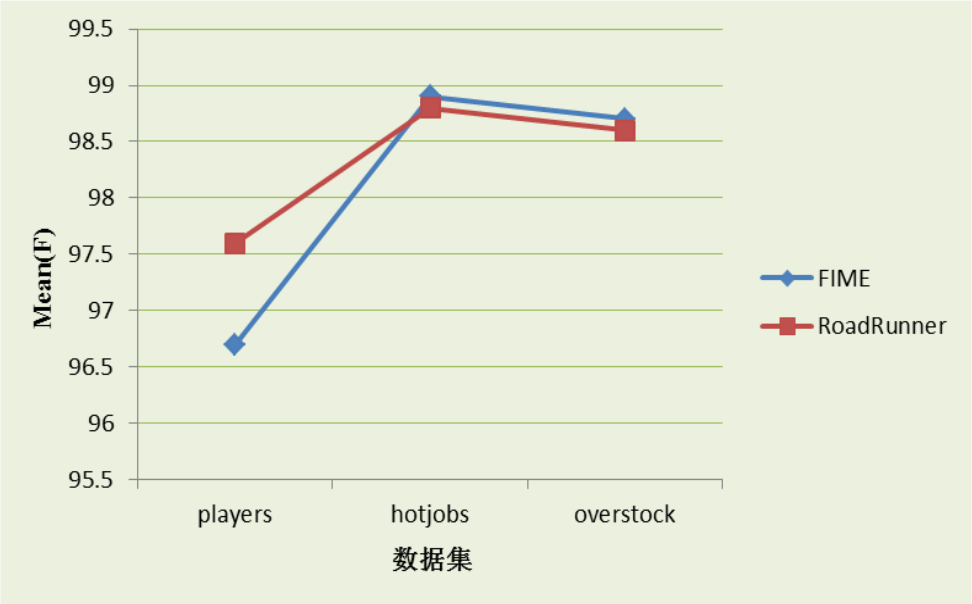


图 5.3 FIME 与 RoadRunner F 均值对比图

Figure 5.3 the Mean comparisons of FIME and RoadRunner

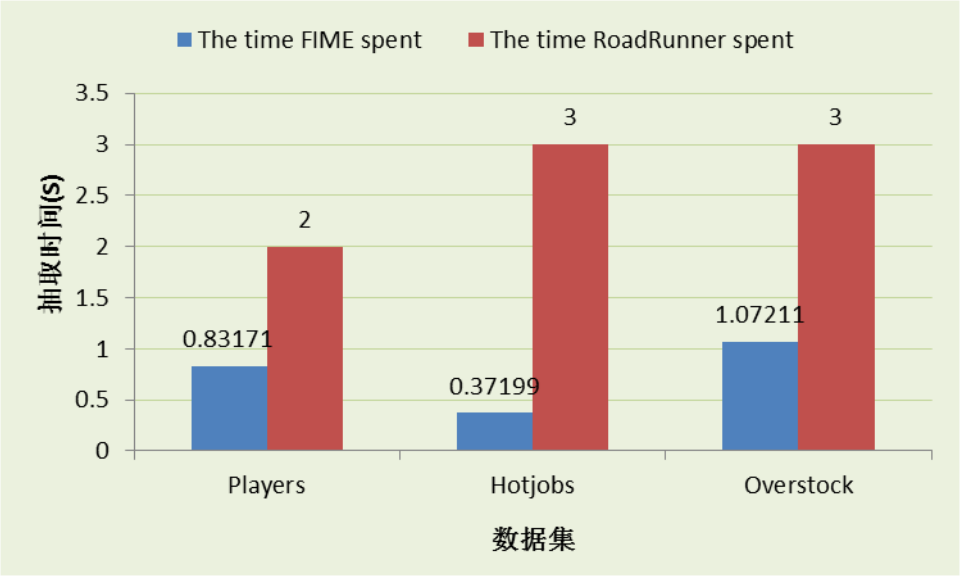


图 5.4 FIME 算法与 RoadRunner 算法抽取总时间对比图

Figure 5.4 the running time comparisons of FIME and RoadRunner

由上述图可以看到，FIME 算法在三个数据集中的准确率与召回率均不低于 RoadRunner 算法的处理效果，但是，在处理同样的数据集时，FIME 算法所花费的时间远小于 RoadRunner 算法的时间，因为 FIME 算法在 DOM 树匹配之前对迭代项进行了合并，避免回溯处理迭代项，进而降低了算法复杂度。

实验二：FIME 算法在九个知名网站的抽取实验。FIME 算法在雅虎音乐、雅虎新闻、智联招聘、新浪新闻、前程无忧、CSDN 搜索、亚马逊、搜狗新闻和当当

九个知名网站的抽取效果部分对比图如图 5.5、5.6 和 5.7 所示，实验结果如图 5.8 所示。

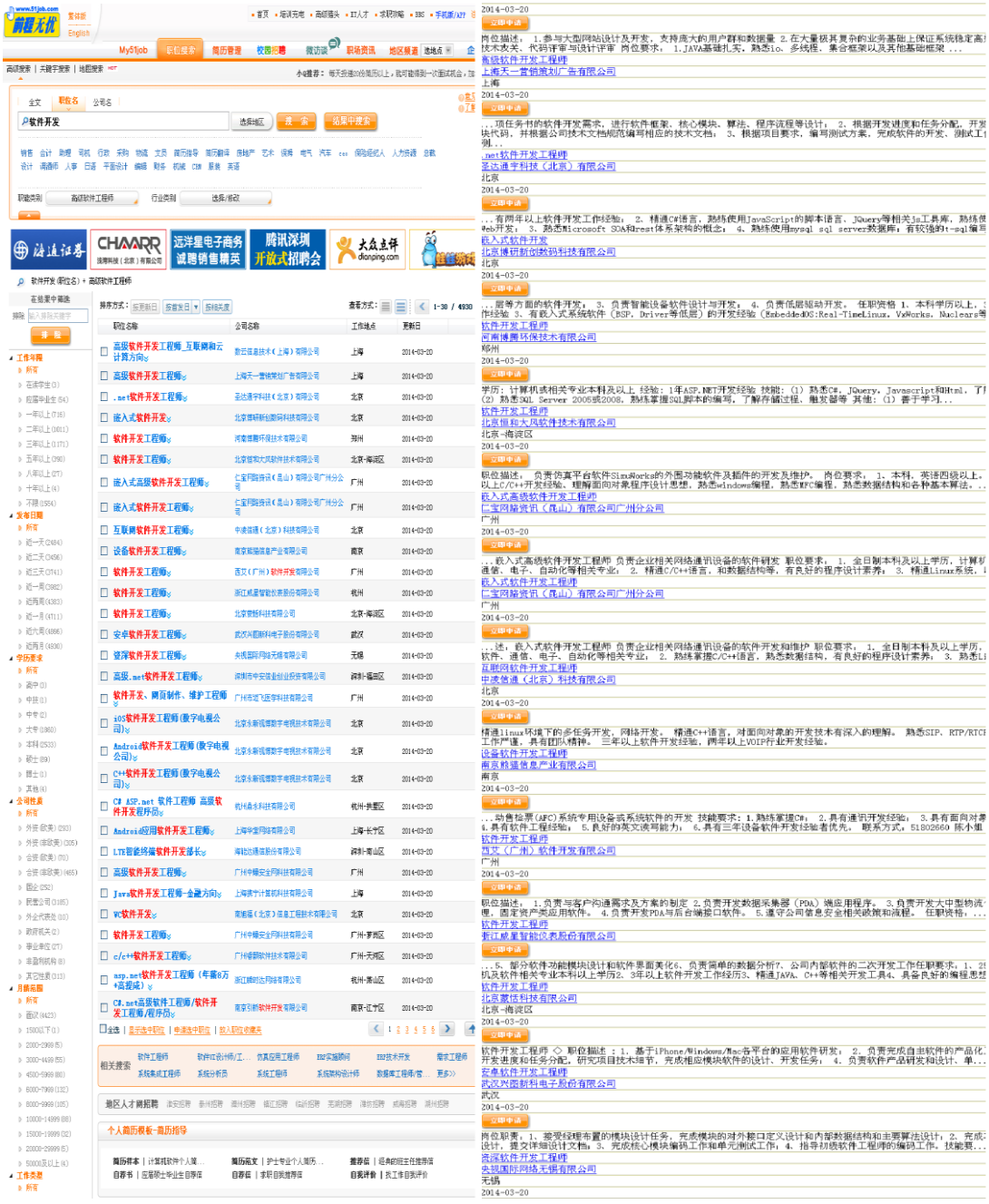


图 5.5　前程无忧网站页面抽取效果对比图

Figure 5.5 the extract results of 51job web pages

图 5.6 CSDN 搜索网站页面抽取效果对比图

Figure 5.6 the extract results of CSDN web pages

图 5.7 搜狗新闻网站页面抽取效果对比图

Figure 5.7 the extract results of sougou news web pages

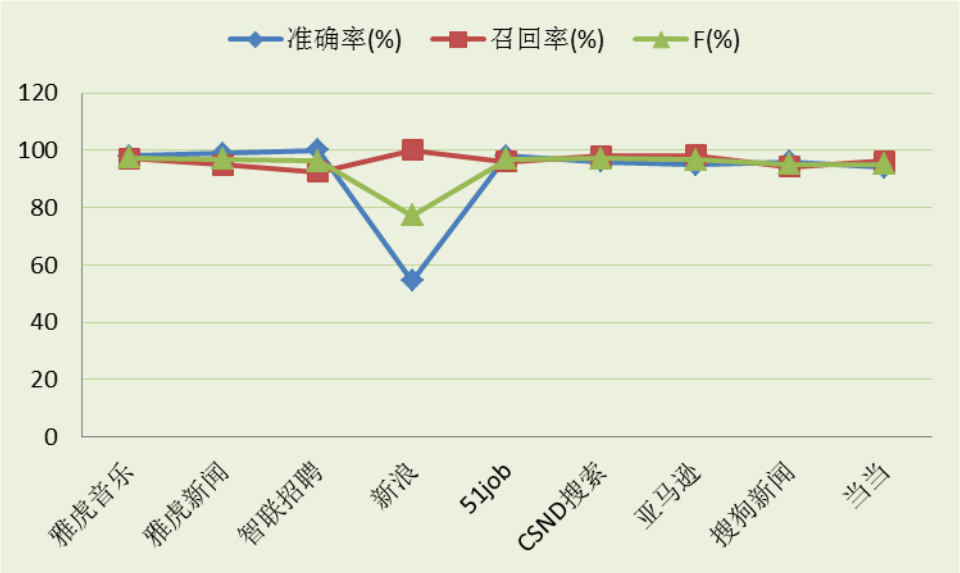


图 5.8 FIME 算法在九个知名网站的实验结果图

Figure 5.8 FIME algorithm results in nine well-known websites

由图 5.8 可以看到，FIME 算法在除新浪新闻外的其它八个网站上的准确率和召回率效果都较好。而 FIME 算法在对新浪新闻网站页面进行数据抽取时，由于该网站页面中存在大量动态广告，这些广告的不断变化使得在生成页面模板时无法将其识别为广告，因此，在抽取数据时会将这些时时变化的动态广告视为有用数据，导致了准确率的降低，这是同类算法共有的缺点。由于这并不影响对真正有用数据的抽取，所以召回率仍然较高，而 F 值的降低是由于准确率的降低所导致。

实验三：将数据集大小分别设为：15M，30M，60M，100M，300M，448M 分别运行在单机与 Hadoop 分布式平台上所花费的时间对比图如图 5.9 所示。从图中可以看出，当页面数据集小于 60M，单机抽取数据的时间小于分布式抽取数据的时间，当页面数据集大于 60M，分布式抽取数据所需的时间小于单机抽取数据的时间；而且，随着页面数据集的逐渐增大，分布式抽取数据的时间远远小于单机抽取数据的时间。

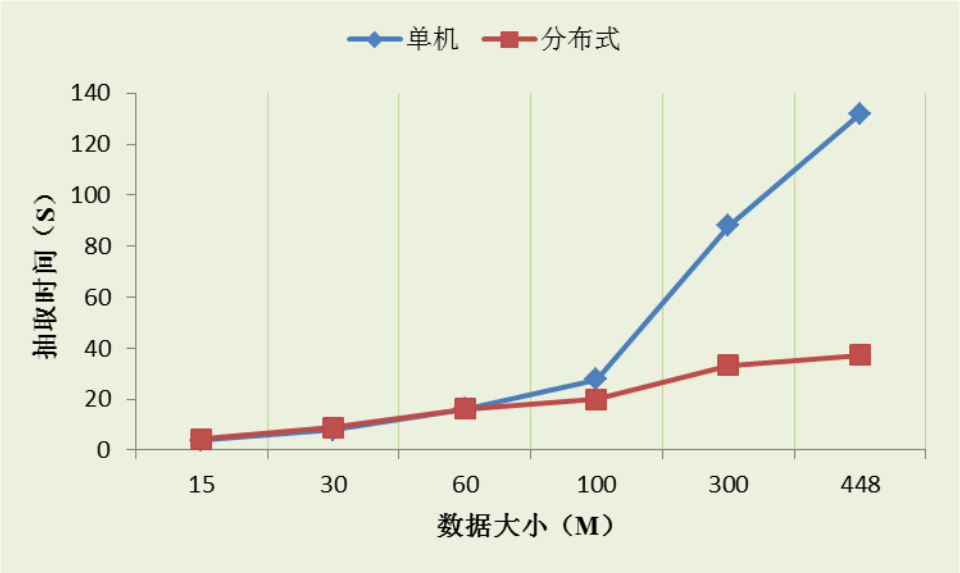


图 5.9 FIME 算法抽取时间对比图

Figure 5.9 the extract time comparison of FIME algorithm in single node and clustering

将抽取数据的时间转换为算法的处理效率，有如**错误!未找到引用源。**所示的效果，可以看出单机 FIME 算法对数据的处理效率基本上保持恒定（约为 3.6M/S），而分布式 FIME 算法抽取数据的速度则会随着页面数据集的增大慢慢的提高,最终会趋于平稳。两种方式对数据的处理效率仍然以 60M 为分隔点，页面数据集小于 60M 时单机处理效率要高于分布式，当页面数据集大于 60M 时，分布式抽取数据的效率要远远高于单机处理。

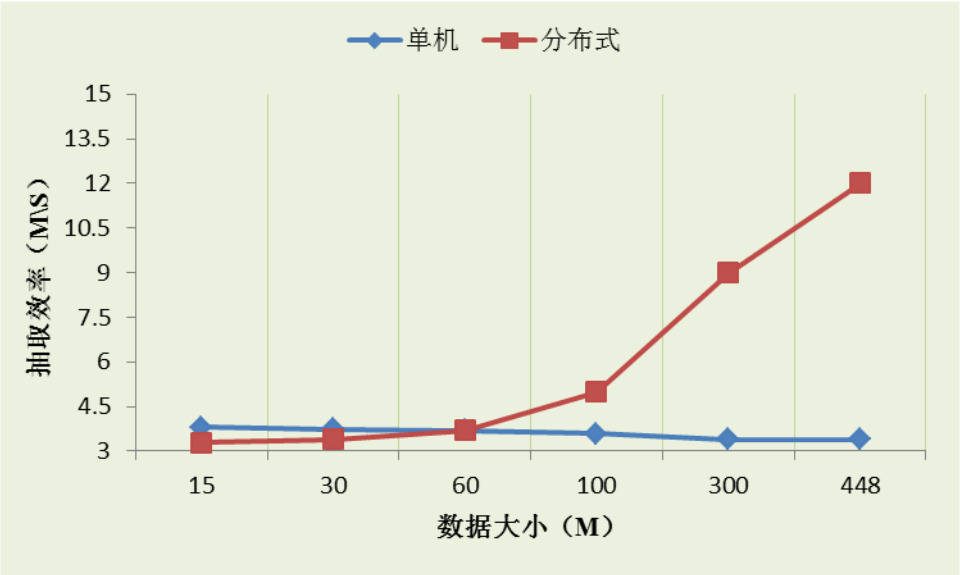


图 5.10 FIME 算法抽取数据效率对比图

Figure 5.10 the extraction efficiency comparison of FIME algorithm in single node and clustering

据以上分析，当页面数据集较小时，分布式抽取数据的效率会低于单机处理方式，当页面数据集较大时，分布式方式的效率则远远优于单机处理方式，两种方式数据处理效率的交叉点为 60M，此交叉点主要受 HDFS 的数据分割大小影响（默认的分割为 64M）。同时，受集群配置的影响，分布式抽取数据的处理效率不会无限制的提高，最终会趋于平稳。

5.4 本章小结

本章首先简要介绍了实验数据、评价指标和集群的配置，然后详细介绍了对实验数据集做的三项实验，实验一：采用 RoadRunner 算法携带的数据集，对 FIME 算法与 RoadRunner 算法在准确率、召回率、F 均值以及所用时间等方面进行了比较，结果显示，FIME 算法在保证准确率与召回率的情况下大大降低了数据抽取的算法复杂度，减少了数据抽取所用的时间；实验二：采用来自雅虎音乐、雅虎新闻、智联招聘、新浪新闻、前程无忧、CSDN 搜索、亚马逊、搜狗新闻和当当九个知名网站的 448M 页面作为数据集对 FIME 算法进行运行分析，结果显示，除新浪新闻网站含有大量动态广告而导致准确率降低之外，其他八个网站的抽取结果均具有较高的准确率和召回率；实验三：在分布式环境中采用不同大小的数据集对 FIME 算法进行运行分析，当页面数据集小于 60M 时，单机抽取数据的时间小于分布式抽取数据的时间，单机抽取方式的效率大于分布式抽取数据的效率，当页面数据集大于 60M 时，分布式抽取数据所需的时间小于单机抽取数据的时间，分布式抽取方式的效率大于单机方式的抽取效率；同时，随着页面数据集的逐渐增大，分布式抽取数据的时间远小于单机抽取数据的时间，但是，受集群配置的影响，分布式抽取数据的处理效率不会无限制的提高，最终会趋于平稳，在实际应用中，还可能会降低一些。

6 总结与展望

6.1 本文总结

Deep Web 数据搜索平台可以为用户提供更实用的数据和服务，而且可以利用 Deep Web 数据搜索平台收集、存储、分类及索引各类信息，从而为用户提供更加方便及个性化的信息搜索服务，对 Deep Web 的研究将会产生可观的社会及经济效益。Deep Web 中的信息广泛分布在各个 Web 后台数据库中，并且只能通过在特定的查询接口中提交 HTML 查询表单，然后以结果页面的方式显示，所以，必须对查询结果页面中的 Deep Web 有效信息进行抽取，才能对 Deep Web 中的信息资源进行集成，使其以统一的模式进行存储，最后为用户提供更好的、统一的索引服务。因而可知，对 Deep Web 查询结果页面中有用信息的抽取是 Deep Web 集成体系中的关键步骤，对 Deep Web 查询结果抽取的研究具有深远的理论价值及实际应用价值。

现在国内外对于 Deep Web 信息抽取技术的研究主要集中在：基于 DOM 树结构、基于统计理论、基于视觉特征和基于模板的抽取技术。基于 DOM 树结构与基于模板的抽取技术在复杂性、适用范围和自动化程度等方面更具有优越性。但是由于模板在生成时自动化程度较低，而基于 DOM 树结构的抽取方法又没有归纳出对相似页面进行统一处理的方法，因此，本文结合两类方法的各自优势提出基于 DOM 树结构和模板方法相结合的抽取方法，FIME（Filtering, Iterating, Matching, and Extracting）算法。

FIME 算法通过对比来自同一网站的页面的 DOM 结构，获取页面模板 Wrapper，然后利用 Wrapper 对来自同一站点的其他所有待抽取页面进行数据抽取。

FIME 算法共包括四个模块，分别为：清噪模块（Filtering）、迭代模块（Iterating）、匹配模块（Matching）和抽取模块（Extracting）。首先，在清噪模块中对页面进行预处理，包括使用 Tidy 工具对页面正则化以及清除页面中的无用标签和无用属性元素；然后在迭代模块中对经过清噪模块处理的页面中的迭代项进行合并；之后将经过合并迭代项的页面输入到匹配模块中，通过对比页面的 DOM 树结构获取待抽取数据的位置信息，即页面包装器 Wrapper（模板）；接下来将同源的所有页面输入到清噪模块中进行页面预处理；最后，将经过清噪处理的所有同源页面和 Wrapper 输入到抽取模块中，对所有页面中的待抽取数据进行自动化抽取。由于 Deep Web 查询返回的结果页面数量众多，因此，本文将 FIME 算法部署在分布式系统基础架构 Hadoop 中。

最后，本文通过实验验证了论文所提方法的正确性以及算法在 Hadoop 平台中运行的高效性。在实验中，本文主要介绍了实验数据集、实验的集群环境及配置情况和最后的实验结果。论文中对实验数据集做了三项实验，实验一：采用 RoadRunner 算法携带的数据集，对 FIME 算法与 RoadRunner 算法在准确率、召回率、F 均值以及所用时间等方面进行了比较，结果显示，FIME 算法在保证准确率与召回率不降低的情况下较大幅度的降低了数据抽取的算法复杂度，减少了数据抽取所用的时间；实验二：采用来自雅虎音乐、雅虎新闻、智联招聘、新浪新闻、51job、CSDN 搜索、亚马逊、搜狗新闻和当当九个知名网站的 448M 页面作为数据集对 FIME 算法进行运行分析，结果显示，除新浪新闻网站含有大量动态广告而导致准确率降低之外，其他八个网站的抽取结果均具有较高的准确率和召回率；实验三：在分布式环境中采用不同大小的数据集对 FIME 算法进行运行分析，结果显示，当页面数据集小于 60M 时，单机抽取数据的时间小于分布式抽取数据的时间，单机抽取方式的效率大于分布式抽取数据的效率，当页面数据集大于 60M 时，分布式抽取数据所需的时间小于单机抽取数据的时间，分布式抽取方式的效率大于单机方式的抽取效率；同时，随着页面数据集的逐渐增大，分布式抽取数据的时间远小于单机抽取数据的时间，但是，受集群配置的影响，分布式抽取数据的处理效率不会无限制的提高，最终会趋于平稳，在实际应用中，还可能会降低一些。

本论文的主要创新点：

① FIME 算法在进行 DOM 树比较之前，首先对页面采取预处理操作，使页面遵守 XHTML 规则，同时清除页面中对于抽取信息无用的标签及部分属性元素，使得页面更精简，以提高后续的匹配算法效率；

② 针对基于 DOM 树结构算法中回溯处理页面中迭代项导致匹配算法复杂度较高的问题，FIME 算法在进行匹配算法之前首先对页面中的迭代项进行合并，降低了后续匹配算法的时间复杂度；

③ 结合基于模板抽取算法的思想，FIME 算法利用在匹配算法中通过比较 DOM 树结构而获得的待抽取数据的位置信息作为同一网站页面的模板 Wrapper，对所有同源页面进行待抽取信息的自动抽取，而不是对同源结构相似的页面做重复的处理，提高信息抽取的效率和自动化程度。

6.2 工作展望

本文主要针对基于 Deep Web 的查询结果自动抽取算法进行了研究，并改进了已存在的抽取算法，提出了基于 DOM 树结构和模板方法相结合的查询结果自动抽取算法 FIME，并且将 FIME 算法部署在分布式系统架构 Hadoop 中，以提高抽取

的执行效率。虽然该算法在实验部分显示出较好的抽取结果，但是，FIME 算法仍有下列问题需要解决：

① 如果查询结果页面中存在大量的动态广告，由于这些动态广告的不断变化使得在生成页面包装器时无法识别其为广告，因此，FIME 算法会将这些动态广告作为有用数据抽取出来，这样将会导致准确率的降低。论文的下一步工作需要研究如何提高包含大量动态广告的页面的抽取准确率。

② FIME 算法在迭代项中存在可选项的抽取粒度问题，即如果迭代项中包含可选项，此时没有对可选项进行特殊处理，而是与迭代项作为一个整体进行抽取，论文的下一步研究工作中需要进一步缩小可选项的抽取粒度，使迭代项与可选项分离开。

致 谢

终于要提笔写致谢部分了，才更深刻的意识到研究生的学习生涯即将落下帷幕。将近三年的研究生学习生活使我受益匪浅，期间从收集资料、阅读文献到做实验分析结果，以及最近几个月毕业论文的撰写，很多人给予我极大的帮助和关怀，这里向他们表达我最真挚的谢意。

首先，要感谢我尊敬的导师，冯永教授。认识冯老师四年多了，现在还清晰的记得大三时冯老师给我们上课的情景，当时就觉得冯老师平易近人、为人谦和，在研究生期间，作为冯老师的学生，我觉得很幸运。学习中，大到课题的选择、研究、论文的撰写，小到课程的选择、奖助学金的评定，冯老师都给予了我精心的指导和帮助，让我完成的更轻松，还记得研一发表小论文时，冯老师辛苦的指导我一遍又一遍的改正论文，只是最后还是没有发表成高水平论文，我觉得很遗憾，也很愧对冯老师；生活中，冯老师更像一位兄长，会经常关心我们的想法，虽然我们并不是经常与冯老师沟通生活中的细节，但是可以感觉的到，冯老师是一位非常具有生活智慧的人。从学习到生活，冯老师严谨治学、做事认真、勤于思考的态度都是我学习的榜样，老师，谢谢您，教会了我很多治学和做人做事的道理。感谢钟将老师，在导师出国期间，在学习工作过程中给予的帮助。

其次，要感谢我的家人。感谢爸爸妈妈在我上学这二十年里，给予我精神和物质上的支持和鼓励，每当我遇到困难与挫折时，听到电话那边爸妈的声音就是我最大的安慰，每当我有开心的事情时，爸妈会比我还高兴，谢谢亲爱的爸爸妈妈，你们是我最爱的人；感谢姐姐，我一直在外地读书，在家的时间太少，是你一直陪伴着爸妈，使他们不那么孤独，谢谢你在我彷徨时给予安慰，替我分忧；感谢小维，谢谢你六年来一直包容我，给予我无微不至的关心，一直陪我走着最美好的时光，谢谢你。

感谢我的母校—重庆大学，七年来为我提供了舒适的生活环境和良好的学习氛围，使我可以专心的进行学习和研究。

感谢实验室的多位师姐、师妹、师兄、师弟们。感谢唐黎师姐，在课题的选择和研究中给予了耐心的指导和帮助；感谢师兄：陈显勇、张洋、陈贞，同门：韩楠、贾东风、周千威、季必贵、张辉，师弟师妹：刘晶、陆维、曾子亮、陈卓、胡道旭、栗恒。在和你们一起学习和生活过程中，你们给予的帮助和关心，谢谢你们。

感谢身边的朋友们。感谢郭崇霄，在我最灰暗的日子里，陪着我，也记得我们一起在成电找工作的日子里，互相鼓励、互相安慰，谢谢你；感谢周丽君、冯

龙华、粟琳、王秋菊、王霞、王翠钦、陆力瑜、焦利敏这些可爱可亲的姐妹们，和你们一起学习生活的日子很美好，谢谢你们带给我那么多快乐。

最后，衷心感谢在百忙之中参加论文评阅和答辩的各位专家、学者、教授，谢谢！

王慧娟

二O一四年四月 于重庆

参考文献

[1] Chang K C, He B, Li C K, et al. Structured databases on the web: observations and implications [C] // IGMOD Record, 2004, 33(3): 61-67.

[2] Michaee B K. The deep web: surfacing hidden value [J]. Journal of Electronic Publishing, 2001, 7(1): 1-7.

[3] Priee G, Sherman G. Exploring the invisible web: seven essential strategies [J]. Online 2001, 25(4): 32-34.

[4] Pang B, Lee L, Shivakumar V. Thumbs up sentiment classificationusing machine learning techniques [C] // Proceedings of the 2002 Conferenceon Emprical Methods in Natural Language Processing. NJ, USA: Association for Computational Linguistics Morristown, 2002: 79-86.

[5] CNNIC.第十六次中国互联网络发展状况统计报告[EB/OL]. http://www.cnnic.net.cn/.

[6] Dean J, Ghemawat S. MapReduce: simplified data processingon large clusters. Communi cations of the ACM, 2008, 51(1):107-113.

[7] Ghemawat S, Gobioff H, Leung ST. The Google file system. SOSP'03. 19th ACM Symposium on Operating Systems Principles, 2003, 37(5): 29-43.

[8] Liu B, Grossman R, Zhai Y H. Mining Data Records in Web Pages[C]. In Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, New York: ACM, 2003, 601-606.

[9] Arasu A, Garcia-Molina H. Extracting Structured Data from Web Pages[C]. In Proceedings of the 9th 2003 ACM SIGMOD international conference on Management of data. New York: ACM, 2003, 337-348.

[10] Kim Y J, Park J Y, Kim T W, et al. Web Information Extraction by HTML Tree Edit Distance Matching[C]. In Proceedings of the 2007 International Conference on Convergence Information Technology. Washington, DC: IEEE Computer Society, 2007, 2455-2460.

[11] Miao G X, Tatemura J C, Hsiung W P, et al. Extracting Data Records from the Web Using Tag Path Clustering[C]. In Proceedings of the 18th International Conference on World Wide Web, New York: ACM, 2009, 981-990.

[12] 刘伟，孟小峰，孟卫一．Deep Web 数据集成研究综述[J]. 计算机学报．2007，30(9): 1475-1489.

[13] Liu W, Meng X F, Meng W Y. Vision-based web data records extraction [C] // Proceedings of the 9th International Workshop in Web and Databases. NewYork: ACM, 2006: 20-25.

[14] 寇月, 李冬, 申德荣,等. D-EEM: 一种基于 DOM 树的 Deep Web 实体抽取机制[J]. 计算机研究与发展, 2010, 47(5): 58-865.

[15] 陶磊, 莫倩. 基于 CSS 选择器的深网结果页面抽取方法[J]. 北京工商大学学报(自然科学版), 2009, 27(2): 40-45.

[16] 马安香, 张斌, 高克宁, 齐鹏, 张引. 基于结果模式的 Deep Web 数据抽取[J]. 计算机研究与发展, 2009, 46(2): 280-288.

[17] 朱明, 李香, 郑烩. 基于多学习策略的网页信息抽取方法[J]. 计算机应用与软件, 2008, 25(12): 68-69.

[18] 郑皎凌, 唐常杰, 姜玥, 杨宁, 李红军. 基于伪属性语义匹配的 Deep Web 信息抽取[J]. 四川大学学报(工程科学版), 2009, 41(2): 173-178.

[19] Buyya R, Yeo C S, Venugopal S. Market-oriented cloud computing: vision, hype, and reality for delivering IT services as computing utilities, Keynote Paper [C] // Proceedings of the 10th IEEE International Conference on High Performance Computing and Communications. Dalian, China, 2009: 25-27.

[20] Armbrust M, Fox A. Above the clouds: a Berkeley view of cloud computing [R]. USA: University of California at Berkeley, 2009.

[21] Erdogmus H. Cloud computing: does nirvana hide behind the nebula [J]. IEEE Software, 2009, 26(2) : 4-6.

[22] 郑纬民. 云计算的大幕已经拉开 [J]. 中国计算机学会通讯, 2009, 2(6), 6-7.

[23] G. L. Steele, Jr. Common. Lisp: the language. Digital Press, second edition, 1990.

[24] Sager N. Natural Language Information Processing. Reading, Massachusetts: Addison Wesley, 1981.

[25] Dejong G. An Overview of the FRUMP System[C]. In: LEHNERT W, RINGLE M h eds. Strategies for Natural Language Processing, Lawrence Erlbaum, 1982: 149~176.

[26] Grishman R, Sundheim B. Message Understanding Conference-6: A Brief History[C]. In: Proceedings of the 16th International Conference on Computational Linguistics COLING-96, 1996-08.

[27] Douthat A. The Message Understanding Conference Scoring Software User's Manual[C]. In: Proceedings of the Seventh Message Understanding Conference, 1998.

[28] Muggleton S. New Generation Computing [J]. Inverse entailment and Progol, 1995, 13: 245-286.

[29] Wang Ji-ying, Lochovsky F H. Data-rich section extraction from HTML pages[C] //Proc of the 3rd International Conference on Web Informations Systems Engineering. Washington DC: IEEE Computer Society, 2002: 2313-2322.

[30] Crescenzi V, Mecca G, Merialdo P. RoadRunner: Towards automatic data extraction from large Web sites [C]. Proceedings of the 27th International Conference on Very Large Data Bases. Roma, 2001: 109-118.

[31] Crescenzi V, Mecca G, Merialdo P. RoadRunner: Automatic data extraction from data-intensive Web sites [C]. Proceedings of the 21th ACM SIGMOD International Conference on Man-agement of Data. Madison, 2002: 624.

[32] Liu B, Grossman R L, Zhai Y. Mining data records in Web pages [C]. Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. Washington, 2003: 601-606.

[33] Zhai Y, Liu B. Web data extraction based on partial tree alignment [C]. //Proc of the 14th International Conference on World Wide Web. New York: ACM Press, 2005: 76-85.

[34] Liu L, Pu C, Han W. XWRAP: An XML-enabled wrapper construction system for Web information sources [C]. Proceedings of the 16th International Conference on Data Engineering. San Diego, 2000: 611-621.

[35] Baumgartner R, Ceresna M, Gottlob G, Herzog M, Zigo V. Web information acquisition with Lixto suite [C]. Proceedings of the 19th International Conference on Data Engineering. Bangalore, 2003: 747-749.

[36] Baumgartner R, Flesca S, Gottlob G. Visual Web information extraction with Lixto[C]. Proceedings of the 27th International Conference on Very Large Data Bases. Roma, 2001: 119-128.

[37] Zhang K, Shasha D. Tree pattern matching [M]. //Pattern Matching Algorithms. New York: Oxford University Press, 1997.

[38] Nestorov S, Abiteboul S, Motwani R. Extracting schema from semi structured data [C]. In: Proc of ACM SIGMOD Conf on Management of Data. Seattle, WA, 1998: 295-306.

[39] 张彦超, 刘方, 李勇等. 基于自动生成模板的 Web 信息提取技术[J]. 北京交通大学学报: 自然科学版, 2009, 33(5): 40-45.

[40] 杨少华, 林海略, 韩燕波. 针对模板生成网页的一种数据自动抽取方法[J]. 软件学报, 2008, 19(2): 209-223.

[41] 陈治昂, 周知予, 李大学. 一种基于模板的快速网页文本自动抽取算法[J]. 计算机应用研究, 2009, 26(7): 2646-2649.

[42] 郑长松, 傅彦, 余莉. 基于模板的 Web 信息自动抽取方法[J]. 计算机应用研究, 2009, 26(2): 570-582.

[43] 时达明, 林鸿飞, 赵晶. 基于模板化的 blog 信息抽取[J]. 计算机工程与应用, 2008, 44(9): 156-162.

[44] Cai D, Yu S P, Wen J R, et al. VIPS: a vision based page segmentation algorithm[R/OL]. (2003-11). http://research.microsoft.com/apps/pubs/defanlt.aspx?id=70027.pdf.

[45] Gupta S, Kaiser G. DOM-based content extraction of HTML documents[C]. //Proc of the 12th World Wide Web Conference. New York: ACM Press, 2003: 207-214.

[46] 孙承杰, 关毅. 基于统计的网页正文信息抽取方法的研究[J]. 中文信息学报, 2004, 18(5): 17-22.

[47] 韩忠明, 李文正, 莫倩. 有效 HTML 文本信息抽取方法的研究[J]. 计算机应用研究, 2008, 25(12): 3568-3574.

[48] Song M Q, Wu X T. Content extraction from Web pages based on Chinese punctuation number[C]. //Proc of International Conference on Wireless Communication, Networking and Mobile Computing. 2007: 5573-5575.

[49] 周佳颖, 朱真民. 基于统计与正文特征的中文网页正文抽取研究[J]. 中文信息学报, 2009, 23(5): 80-85.

附 录

A. 作者在攻读硕士学位期间成果目录

[1] Yong FENG, Huijuan WANG, Chun AI. Filtering, Iterating, Matching, and Extracting based Automatic Deep Web Data Extraction. Journal of Computational Information Systems, v 8, n 11, June 1, 2012, pp 4769-4776. 已检索. (EI 检索).

[2] Yong Feng, Huijuan Wang. Deep Web Query Result Pages Content Extraction Based on Combination of Vision and Tag Information. INFORMATION-AN INTERNATIONAL INTERDISCIPLINARY JOURNAL, v 16, n 6(A), June 2013, pp 3635-3641. 已发表. (SCI 四区).

[3] 冯永，王慧娟，钟将，周尚波，李季. 面向深层网页面数据自动抽取方法.发明专利， 专利号：ZL 2012 1 0086024.4. 已授权.

B. 作者在攻读硕士学位期间参加的项目

[1] 按需聚合的语义深层网查询云模型及评估研究，项目编号：61103114,国家自然科学基金青年基金，2012.1-2014.12。导师为项目负责人，目前在研。

[2] 药品交易供应链协同技术研究，项目编号：2012BAH19F001，国家科技支撑计划项目，2012.1-2013.12。导师为项目负责人。