**密级:**



**硕士学位论文**

**基于Hadoop的大数据分析应用开发平台**

**的设计与实现**

**作者姓名： 赵薇**

**指导教师: 叶丹 研究员**

**中国科学院大学软件研究所**

**学位类别: 工学硕士**

**学科专业: 计算机软件与理论**

**培养单位: 中国科学院大学软件研究所**

**2014年 4月**

**Design and Implementation of Hadoop Based Big Data Analysis Platform**

**By**

**Zhao Wei**

**A Dissertation Submitted to**

**University of Chinese Academy of Sciences**

**In partial fulfillment of the requirement**

**For the degree of**

**Master of Computer Software and Theory**

**Institute of Software**

**University of Chinese Academy of Sciences**

**April, 2014**

**独创性声明**

本人声明所呈交的论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明。

签名：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 日期：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**关于论文使用授权的说明**

本人完全了解中国科学院软件研究所有关保留、使用学位论文的规定，即：中国科学院软件研究所有权保留送交论文的复印件，允许论文被查阅和借阅；中国科学院软件研究所可以公布论文的全部或部分内容，可以采用影印、缩印或其它复制手段保存论文。

（保密的论文在解密后应遵守此规定）

签名：\_\_\_\_\_\_\_\_\_\_ 导师签名：\_\_\_\_\_\_\_\_\_\_ 日期：\_\_\_\_\_\_\_\_\_\_

**基于Hadoop的大数据分析应用开发平台的设计与实现**

**摘要**

随着信息化的深入推进，互联网、移动应用的迅速发展，数据产生的速度越来越快，累积的数据量越来越庞大。大数据一方面意味着更多的信息价值，另一方面也带来了技术上的挑战。得益于可扩展性、容错性和高可用性，Hadoop已经成为业界大数据处理的事实标准。但相对于传统成熟的工具，基于Hadoop进行大数据分析仍存在多方面的困难，如接口不友好（需调用编程接口）、低效（业务无关的系统安装配置）和异构（分析工具分散独立且学习成本大）等。针对这些问题，论文设计与实现了基于Hadoop的大数据分析应用开发平台——Haflow，屏蔽Hadoop平台的编程与底层维护细节，支持数据分析人员交互式、可视化地开发大数据分析应用。

设计与实现基于Hadoop的大数据分析应用开发平台，论文主要研究了以下关键技术。首先，本文提出了组件化、流程式大数据分析应用编程模型，定义了可扩展数据分析组件模型，给出了基于Java注解的组件描述方法和基于继承的可扩展组件实现方法；定义了数据分析业务流程模型，提出了基于组件以及模型驱动的数据分析业务流程处理框架。其次，设计并实现了基于嵌套图识别与分层的从数据分析业务模型（数据流模型）到基于Oozie的并发执行模型（控制流模型）的转换算法；使用嵌套图识别算法，判定数据分析流程图是否可以直接翻译成基于Oozie的并发执行模型，对于不满足并发语义的业务模型，使用基于分层的模型转换算法提高流程的并发执行度。再次，提出了基于HDFS的异构中间数据统一管理方案，通过对中间数据的分类建模，实现组件之间中间数据的自动匹配验证与自动存储。

论文详细介绍了大数据分析应用开发平台Haflow的系统架构与关键实现技术，并通过河南省新农合医疗数据分析和互联网新闻标签云生成两个实际案例，介绍了Haflow的应用效果。

**关键词：**大数据分析，数据挖掘，开发平台，组件，Hadoop

**Design and Implementation of Hadoop Based Big Data Analysis Platform**

**ABSTRACT**

With the rapid development of information technologies, web 2.0 and mobile applications, data in organizations keeps on accumulating. Big data brings not only value, but also challenges, leading to a major shift in data management. Due to its scalability, fault-tolerance and availability, Hadoop has become the de facto standard. However, adopting Hadoop is still a non-trivial task; difficulties, like unfriendly interfaces, heterogeneous but isolated tools, make data analysis on Hadoop a tedious work, To deal with these difficulties, this thesis implements Haflow, a flexible open platform facilitating big data analysis. We present its implementation and address several research problems.

We illustrate key technologies tackled in this thesis. To begin with, we adopt the module-oriented and model-driven mechanisms for integrating isolated tools. Data analysis tools or algorithms are wrapped into different modules with unified interfaces. On the other hand, Haflow provides a web-based interface. Developers analyze data through designing the corresponding workflow in a visualized way; and then Haflow transforms the business model to the execution model expressed as the primitives of Oozie, the workflow engine on Hadoop. To pursue high concurrency, we proposed a pattern transformation algorithm based on the nested graph recognition and stratification. Moreover, various strategies are proposed for efficient intermediate data management, including intermediate data auto-save and auto-matching method. Then we describe some implementation details. In the end, we evaluate the effectiveness through extensive case studies.

Due to the above research, Haflow gains flexibility, friendly interfaces and efficiency. Users only need to focus on their business during the process of data analysis. This thesis proved the effectiveness with cases in different areas.

**Keywords:** big data analysis, data mining, development tools, module, Hadoop

目录

[1 绪论 1](#_Toc385923546)

[1.1 研究背景 1](#_Toc385923547)

[1.2 论文工作 3](#_Toc385923548)

[1.3 论文组织 3](#_Toc385923549)

[2 大数据分析相关工作 5](#_Toc385923550)

[2.1 大数据及大数据分析技术 5](#_Toc385923551)

[2.1.1 大数据的定义 5](#_Toc385923552)

[2.1.2 大数据分析技术 5](#_Toc385923553)

[2.2 Hadoop分布式处理框架 6](#_Toc385923554)

[2.3 大数据分析应用开发平台现状 7](#_Toc385923555)

[2.4 大数据分析应用开发面临的问题 9](#_Toc385923556)

[3 大数据分析应用开发平台的关键技术 11](#_Toc385923557)

[3.1 系统架构 11](#_Toc385923558)

[3.1.1 系统需求 11](#_Toc385923559)

[3.1.2 系统架构 12](#_Toc385923560)

[3.2 基于组件以及模型驱动的数据分析业务流程处理框架 14](#_Toc385923561)

[3.2.1 数据分析组件 14](#_Toc385923562)

[3.2.2 数据分析业务流程模型 21](#_Toc385923563)

[3.2.3 基于Oozie的数据分析执行模型 22](#_Toc385923564)

[3.2.4 数据分析业务流程处理框架 25](#_Toc385923565)

[3.3 基于嵌套图识别与分层的模型转换算法 27](#_Toc385923566)

[3.3.1 数据流模型和控制流模型 27](#_Toc385923567)

[3.3.2 数据流模型到控制流模型的转换 29](#_Toc385923568)

[3.4 基于HDFS的中间数据管理 37](#_Toc385923569)

[3.4.1 异构中间数据的管理 37](#_Toc385923570)

[3.4.2 中间数据的自动管理 38](#_Toc385923571)

[3.4.3 中间数据的匹配验证 38](#_Toc385923572)

[4 基于Hadoop的大数据分析应用开发平台的设计与实现 41](#_Toc385923573)

[4.1 系统架构及相关实现技术 41](#_Toc385923574)

[4.1.1 系统架构 41](#_Toc385923575)

[4.1.2 相关实现技术 42](#_Toc385923576)

[4.2 系统总体设计 43](#_Toc385923577)

[4.2.1 系统的包图和模块图 43](#_Toc385923578)

[4.2.2 系统的数据库设计 46](#_Toc385923579)

[4.3 可扩展组件的实现 47](#_Toc385923580)

[4.3.1 组件的定义和实现方法 47](#_Toc385923581)

[4.3.2 组件管理框架的实现 48](#_Toc385923582)

[4.4 模型转换引擎的实现 49](#_Toc385923583)

[4.4.1 流程图建模 50](#_Toc385923584)

[4.4.2 模型转换算法实现 52](#_Toc385923585)

[4.4.3 数据分析应用翻译执行的序列图 55](#_Toc385923586)

[4.5 应用验证 56](#_Toc385923587)

[4.5.1 实验环境 56](#_Toc385923588)

[4.5.2 河南省新农合数据分析 57](#_Toc385923589)

[4.5.3 标签云生成 59](#_Toc385923590)

[5 结束语 63](#_Toc385923591)

[5.1 论文贡献 63](#_Toc385923592)

[5.2 未来工作展望 63](#_Toc385923593)

[参考文献 65](#_Toc385923594)

[发表文章 67](#_Toc385923595)

[致谢 69](#_Toc385923596)

# 绪论

本章介绍论文的研究背景、主要工作和论文组织方式。

## 研究背景

如今，一个大规模生产、分享和应用数据的时代正在开启。互联网时代，尤其是社交网络、电子商务和移动通信把人类社会带入了一个“PB”为单位的结构与非结构数据信息的新时代[6]。大数据已经影响到了每一个行业，并且成为了继劳动力和资本之后的又一重要生产力[28]。例如在公共卫生领域，大数据可以用来预测与估计疾病的发病率，帮助医生决策处方，分析疾病的前兆特征，发现多种疾病的并发关系，分析临床表现与疾病的推理关系以及预测病人生存期等。麦肯锡全球研究院(MGI)在其报告中指出，大数据分析可以帮助美国的医疗服务业一年创造3000亿美元的附加价值[28]。

大数据具有四个方面的典型特点[29, 30]，因此通常也以“4V”来描述大数据：数据量大(Volume)，能达到TB，甚至PB级别。数据类型多(Variety)，包括结构化，半结构化和非结构化多种类型数据，每种数据在其存储编码上面也表现出很强的异构性。数据产生速度块(Velocity)，大数据因为其数量大，计算时间会变长，因此对处理速度要求较高。数据价值密度低(Value)，数据量大，能提取出的有用信息少，数据价值密度较低，如何从大量数据中抽取有用数据是一个很重要的问题。

大数据带来巨大信息价值的同时，也带来了技术上的挑战。传统的单设备存储已经无法满足大数据存储的需求；传统的统计分析、数据挖掘、机器学习等工具无法胜任大数据量下的知识提取；传统的可视化工具无法准确地展示大数据的特点。此外，数据异构性，数据规模，及时性，复杂性以及数据私密性这些问题严重阻碍着大数据中的价值获取[8]。

各种大数据应用的爆发性增长，已经衍生出独特的架构，并直接推动了存储、网络以及计算技术的研究。Google于2003年在SOSP上发表论文阐述其分布式文件系统GFS[23]的架构设计，2004年在OSDI上发表论文介绍了MapReduce[24]海量数据处理范型，自此各种涉及大数据处理的技术开始蓬勃发展，而其中影响力最大的则是Apache开源社区的Hadoop[1]系统。Hadoop系统包含了GFS和MapReduce的开源实现HDFS、Hadoop MapReduce这两个子系统 (注：目前最新版本的Hadoop还包含了资源管理组件YARN[25])。经过多年以来开源社区的贡献，Hadoop系统已经拥有了一个完整而强大的生态系统，例如构建于Hadoop之上的NoSQL数据库HBase[27]，数据仓库Hive[2]，分布式实时计算框架Storm[7]，分布式流程执行引擎Oozie[3]，流程处理语言Pig[26]和分布式机器学习工具Mahout[5]等。得益于其水平可扩展性、容错性和高可用性，Hadoop已经成为业界处理海量数据的事实标准。

上述各种开源大数据分析处理工具的诞生和发展，在给用户和开发者带来方便的同时，也增加了相应的学习和工作负担。大量实践经验表明，使用Hadoop平台直接进行大数据分析应用的开发具有以下不足：

1. Hadoop环境搭建和相关工具的集成繁琐而复杂。大数据场景下，数据处理分析框架一般采用分布式的架构，相对于传统的单机系统，其安装配置过程复杂繁琐且易出错(参数设置、版本匹配等)；且不同的框架异构独立，集成困难。
2. Hadoop平台和相关工具的学习周期长，代价高。一次数据处理分析包括存储、转换、挖掘等阶段，涉及到多种需求，用户往往需要去学习各种工具复杂的接口，学习成本高、周期长。
3. 数据分析任务及算法的复用和共享。在实际特定的业务场景下，数据分析任务的模式通常是固定的，每次变化的只是若干参数而已。此外，算法由不同人员编写，有不同的接口风格，为算法的管理和共享带来困难。因此，对于数据分析任务及其算法提供统一管理，达到可复用和共享，显得重要。
4. 用户接口复杂，生产效率低。用户在数据分析时，需要通过编程来调用各个分析框架提供的算法，并编写脚本来制定调度策略；大量的时间花费在这些业务无关的重复工作上，效率低下。

上述不足直接导致大数据场景下数据分析应用开发周期变长，造成资源的极大浪费和信息获取的严重滞后。面对上面的种种问题，本文研究如何开发一个基于Hadoop的大数据分析应用开发平台，该平台的系统目标包括以下三点：

1. 充分利用Hadoop生态系统的强大优势，天然地支持各种大数据处理分析框架的接入；并借助于其分布式存储规模和分布式计算能力，加快大数据分析的速度，降低企业IT成本。
2. 集成各种数据存储、数据处理和数据分析的工具，构建一站式的大数据分析应用开发平台，应对复杂分析需求。将用户和开发者从平台搭建、工具集成、接口学习、算法管理、实验脚本编写和多任务调度等繁杂的工作中抽离出来。
3. 构建基于Web的可视化，拖拽式的数据分析应用开发界面，极大降低数据分析人员的学习成本，使得数据分析人员专注于业务模型本身，加快开发速度，使得企业可以快速地从海量信息之中获益。

## 论文工作

本文针对如何设计并实现一个基于Hadoop的大数据分析应用开发平台进行了研究。对如何解决基于Hadoop的大数据开发中的集群搭建困难，工具安装复杂，算法难于重用与共享，学习周期较长等问题进行了思考。具体包括以下几个方面：

总结了大数据分析的方法，分析了数据分析开发平台的现状，列举了大数据分析应用开发平台存在的问题和大数据分析应用开发面临的问题。

本文主要研究基于Hadoop的大数据分析应用开发平台中的关键技术，分为如下三点：

1. 提出了基于组件以及模型驱动的数据分析业务流程处理框架。定义了数据分析组件模型，提出了基于Java注解的组件描述方法以及基于继承的可扩展组件实现方法。在数据分析组件的基础上，本文定义了数据分析业务流程模型，提出了数据分析业务流程处理框架，将数据分析人员从算法调度、执行控制等工作中抽离出来。
2. 设计并实现了基于嵌套图识别与分层的模型转换算法。该算法可以根据数据分析流程模型的依赖关系，生成基于Oozie的并发执行模型。嵌套图识别算法用来判断数据分析流程是否满足Oozie的并发语义，如果满足则数据分析流程可以翻译成最优的执行流程，否则将采用基于分层并发的转换算法。该算法解决了数据分析流程模型到Oozie并发执行模型的高效转换，并使得满足Oozie并发语义的数据分析流程达到了最高的并发程度，提高了数据分析任务的执行效率。
3. 提出了基于HDFS的异构中间数据管理方案，中间数据的自动存储策略以及中间数据的自动匹配验证，达到了中间数据的自动化管理。

最后论文介绍基于Hadoop的大数据分析应用开发平台——Haflow的设计与实现。Haflow充分利用并实现了上述的研究成果，并通过河南省新农合数据分析应用和标签云生成应用验证了系统的功能。

## 论文组织

论文的后续章节按照以下方式组织：

第二章介绍大数据的概念，大数据分析技术，Hadoop分布式处理框架，大数据分析应用开发平台现状，大数据分析应用开发面临的问题。

第三章介绍基于Hadoop的大数据分析应用开发平台的架构和关键技术。首先介绍了基于Hadoop的大数据分析应用开发平台的需求以及系统架构。然后重点介绍了其中的关键技术，包括：基于组件技术和模型驱动的数据分析流程处理框架，基于嵌套图识别与分层的模型转换算法，基于HDFS的中间数据管理。

第四章具体描述了基于Hadoop的大数据分析应用开发平台的设计与实现。介绍了系统架构，相关实现技术，可扩展组件的实现，模型转换引擎的实现，河南新农合数据分析实验以及标签云生成实验。

第五章对全文做出总结，包括论文的工作，贡献以及对未来工作的展望。

# 大数据分析相关工作

本章介绍大数据及大数据分析技术，Hadoop分布式处理框架，大数据分析应用开发平台的现状以及大数据分析应用开发面临的问题。

## 大数据及大数据分析技术

### 大数据的定义

随着互联网、云计算和移动应用的迅速发展，当今社会已经进入了大数据时代。根据Gartner的定义，大数据可以理解为传统的方法和工具无法处理的大量数据。它定义了那些超出正常处理范围和大小、迫使用户采用非传统处理方法的数据集。沃尔玛就是一个典型实例。这一零售业巨擎每小时要处理超过 100 万笔客户交易，其数据库估计包含超过 2.5 PB的数据，相当于美国国会图书馆全部书籍中所含信息的167倍[28]。

大数据有四个典型特点：

1. 数据量大(Volumne)，能达到TB，甚至PB级别。
2. 数据种类繁多(Variety)，有结构化，半结构化，非结构化等数据，每种数据在其存储编码上面也表现出很强的异构性。
3. 速度(velocity)，大数据因为其数量大，计算时间会变长，因此对处理速度要求较高。
4. 价值密度低(Value)，数据量大，数据中的有用信息相对较少，信息提取困难。

### 大数据分析技术

数据分析是企业运营的辅助工具，企业利用它洞察数据，结合领域知识创建报表和分析模型，然后再用事实和数据验证报表和模型，经过验证的报表和分析模型可以用来辅助企业做出决策。数据分析一般分为数据采集，数据ETL、数据存取、统计分析、数据挖掘、数据展示等几个步骤。

大数据分析技术的发展给这个传统领域增加了很多的生机。大数据分析正在变革着各个领域，公共卫生，金融行业，零售业，电信业，生物学等，逐渐成为重要的生产因素。大数据中隐含着很多行业需要的信息，有了这些信息企业可以更加深入地洞察业务模式，预测市场的发展趋势，从而帮助企业提高运营效率和竞争优势。

在公共卫生领域，大数据可以用来预测与估计疾病的发病率，帮助医生决策处方，分析疾病的前兆特征，发现多种疾病的并发关系，分析临床表现与疾病的推理关系以及预测病人生存期等。麦肯锡全球研究院(MGI)在其报告中指出，排除体制障碍，大数据分析可以帮助美国的医疗服务业一年创造3000亿美元的附加价值[28]。

感冒一直是公共卫生的一大困扰。感冒病毒变异速度很快，可以通过多种方式迅速传播。根据世界卫生组织(World Health Organization, WHO)的统计，全球范围内每年成人感冒的发病率为5%-10%，儿童感冒的发病率为20%-30%，而且每年会造成25万到50万人死亡[9]。如果能在感冒大规模爆发前找到感冒高发区，便能够大幅度降低感冒的发病率。Google利用挖掘人们在搜索引擎里面的搜索词与感冒病情之间的相关关系，成功地估算出了世界不同地区感冒的传播情况。CDC和RISS使用传统的监控系统，要延迟一到两周的时间才能给出感冒病情在各个区域的情况。Google利用数据挖掘方法将这个时间缩短到了一天。

大数据分析正在逐步改变着人们的生活，工作以及思维，引领着各个领域的变革。

## Hadoop分布式处理框架

大数据场景下，数据分析和挖掘一般依赖于各种分布式的计算框架[19]，如MapReduce, Pregel[20], Dryad[21]等，并且一般是基于流程的，如[17]中提到的平台利用BPEL流程执行引擎。传统的数据分析架构已经不能满足需求了，如[16]中的平台是基于Google App Engine的，数据通过网络传送给云服务器，这样的架构并不适合大数据量的情况，并且数据存储在开放的服务平台上，带来数据安全的隐患，对于很多公司和个人来说难以接受。

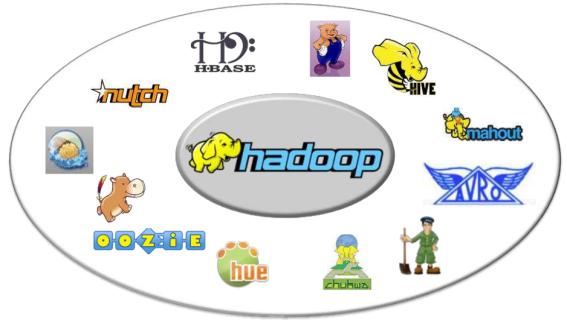


图 2‑1 Hadoop生态系统

2004年Google发表论文介绍了MapReduce。2006年Apache Hadoop项目正式启动。Hadoop是一个开源的分布式系统基础架构，经过几年快速的发展，Hadoop生态系统已经日臻完善。作为Apache的顶级开源项目，它俨然已经成了开源大数据处理的事实标准。Hadoop主要包含两部分，一部分是分布式文件系统HDFS。HDFS部署在廉价的机器上，是一个可扩展的，具有高容错的，可靠的文件存储系统。企业可以使用很低的成本存储大量的数据。另一部分是可以处理大数据的并行计算框架MapReduce。Hadoop生态系统中除了核心系统中的HDFS和MapReduce，在其之上还包括其它各种针对不同场景的系统，如Zookeeper，HBase，Hive，Pig，Mahout，Sqoop，Oozie等。这些项目共同提供了一个海量数据处理的分布式软件生态系统。

Hive：基于Hadoop的数据仓库，可以支持大数据的结构化存储。Hive将结构化数据映射为数据库表，将表结构信息存储在传统的关系型数据库中，并提供简单的SQL查询接口。用户执行SQL查询后，Hive将SQL转换为MapReduce计算，提交到Hadoop系统执行。

Mahout：Mahout是使用MapReduce编程框架，编写的数据挖掘和机器学习的算法库。它提供了很多可扩展的数据挖掘经典算法，包括，聚类，分类，协同过滤推荐算法等。

Sqoop[4]：Sqoop可以用来在传统数据库和HDFS之间批量传输数据,是基于Hadoop的批量数据导入导出工具。

Oozie：这是一款专门为Hadoop开发的可扩展的，基于多租户的，安全的，易用的工作流管理系统。它可以管理多个任务之间的调度，Oozie使用有向边连接这些任务，并添加必要的控制节点，最后生成一个有向无环图(DAG)。Oozie使用XML来定义这些任务以及任务之间的关联关系。

## 大数据分析应用开发平台现状

在数据挖掘领域，商业化的数据分析软件包括SPSS Clementine[11]，KXEN Analytic Framework[14]等；比较常用的开源免费数据分析工具包括，Weka[10]、R[13]、RHadoop[12]、Mahout等。

Clementine和Weka产品定位很相似，都集成了大量的成熟算法，包括预处理算法，分类算法，回归算法、聚类算法、关联规则等。它们的交互界面风格也相似，都支持拖拽式的数据分析流程设计。但是，Clementine是商业化的收费软件，Weka是开源的，可扩展的免费软件。

R是一种用于统计分析和绘图的语言。它不仅有很强的统计分析功能，而且是一个开放式的平台，用户可以通过开发并安装扩展包增强R的功能。

RHadoop是由Revolution Analytics 发起的将R与Hadoop无缝结合的开源项目。该项目分为三个R扩展包：支持用R来编写MapReduce应用的RMr扩展包、用于R访问HDFS的RHdfs扩展包以及用于R语言访问HBase的RHbase扩展包。

Mahout是用MapReduce开发的一款分布式的数据分析工具。相对于Weka和Clementine来说，它没有友好的交互界面，集成的算法也较少，有些算法的算法复杂度较高，但是Mahout为大数据的挖掘提供了可能性，它的地位便是不容忽视的。

KXEN Analytic Framework是一款十分专业的商业化的挖掘软件。相对于Weka、Clementine来说，它最大的特色是半自动化的数据挖掘(Self-Acting)。客户定制调优的空间比较小，在默认选项下，它的运行速度优于其它软件，而且分析结果更加精确。

类似的数据挖掘平台还有：RapidMiner，KNIME等。

根据上述各种数据分析工具是否支持数据处理，分析应用开发方式，是否可扩展，是否服务化，是否免费，是否成熟，是否支持分布式算法几方面的特点进行比较，得到表格 2‑1的统计数据。

表格 2‑1数据分析平台功能调研

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **设计方式** | **数据**  **处理** | **开发**  **方式** | **可扩展** | **服务化** | **免费** | **成熟度** | **分布式** |
| **目标** | 支持 | 拖拽 | 是 | 是 | 开源 | 成熟 | 支持 |
| **Mahout** | 不支持 | 编码 | 是 | 否 | 开源 | 不成熟 | 分布式 |
| **Weka** | 不支持 | 拖拽 | 是 | 否 | 开源 | 成熟 | 单机 |
| **R** | 不支持 | 编码 | 是 | 否 | 开源 | 成熟 | 单机 |
| **R-Hadoop** | 支持 | 编码 | 是 | 否 | 开源 | 不成熟 | 半分布式 |
| **Clementine** | 不支持 | 拖拽 | 否 | 否 | 收费 | 成熟 | 单机 |
| **KXEN** | 不支持 | 向导 | 是 | 否 | 收费 | 成熟 | 未知 |
| **Datameer** | 支持 | -- | 是 | 是 | 收费 | 成熟 | 分布式 |
| **BigInsights** | 支持 | -- | 是 | 是 | 收费 | 成熟 | 分布式 |

* + 现有的数据分析平台主要存在如下问题

1. 数据处理支持不完善

现有的数据挖掘软件，如Weka，Clementine等主要是集成了很多数据挖掘的算法，不具备较完善的数据处理解决方案。

1. 难以适用于大数据场景

现有的数据分析应用开发平台大部分是单机的，缺乏大数据挖掘的支持，如Weka，R，Clementine，DMVisualMiner[18]等。

1. 工具处于发展阶段，不够成熟

Mahout，R-Hadoop这些工具支持分布式执行，但是它们目前处于不成熟阶段，文档资料不够完善，使用不方便，学习成本大，编码负担重。

1. 购买费用高昂

Datameer、BigInsights是商业软件，购买和使用的费用很高，不适合小公司使用和个人研究。

## 大数据分析应用开发面临的问题

大数据带来了机遇，同时也提出了前所未有的挑战。数据量大(Volumne)，数据种类繁多(Variety)，生成速度快(velocity)，价值密度低(Value)这四个特点，使得大数据处理面临着很多的困难。很多公司拥有大量很难处理的数据，包括结构化，半结构化，纯文本的数据。

传统的数据分析工具发展较为成熟，市场上有很多商业的软件可以帮助企业进行数据分析，如IBM、SAP、Oracle、Microsoft、MicroStrategy等公司都有自己的商业智能软件。目前可以进行大数据分析应用开发的工具却很少，很多公司或者企业用Hadoop开发大数据分析应用。Hadoop的快速发展使得公司处理大数据成为可能，并可以解决许多以前由于数据量太大而无法处理的业务难题。



图 2‑2大数据分析应用开发流程图

图 2‑2展示了基于Hadoop和若干开源工具开发的一个用户行为预测的实际应用案例[22]。数据的处理采用MapReduce，处理后的数据导入Hive数据仓库，利用Hive提供的类SQL语言HiveQL对数据进行筛选、提炼和转换，并导出为ARFF文件。使用Weka和Mahout两个数据挖掘的工具对ARFF文件进行聚类操作，得到聚类结果用来训练行为模式分类模型。

从上述例子可以看出，Hadoop为大数据分析应用的开发提供了基础支撑。但大数据分析是一个多步骤的繁杂工作，开发人员需要在集群环境下安装配置Hadoop、Hive、Java、MySql、Mahout等工具。这些工具需要的数据存储在不同的地方，用户在使用前需要导入或导出数据。开发人员在开发数据分析应用时，需要编写数据分析算法的代码，编写调度策略代码，编写实验代码。

从这整个过程中可以发现，使用Hadoop进行大数据分析应用开发主要存在以下几个方面的问题：

* + Hadoop平台的搭建和相关工具的集成复杂

Hadoop作为Apache组织下的开源系统，目前仍然处于发展阶段，难以像成熟的商业工具一样具有良好的可用性。因此，Hadoop及其构建于其之上的各种开源工具，都存在着安装、部署繁琐，彼此之间版本兼容，以及各种参数配置等问题。用户和开发人员在使用时往往耗费大量的精力。

* + Hadoop平台和相关工具的学习周期长，代价高

Hadoop包含MapReduce编程框架，数据分析人员需要首先学习这个新的编程框架，才能进行下一步的应用开发，这对于很多的领域数据分析专家来说可能是相当困难的一件事情。

* + 可复用的算法很难管理和共享

大数据分析和处理有很多的算法库，这些算法库由不同的人或者组织编写，有不同的接口风格，不同的使用方法。目前没有专门的组织整理这些算法，算法的管理和共享很困难。

* + 算法的执行和调度由用户自己写代码控制，算法无法重用

大数据分析应用不像传统的数据分析可以很快地计算出结果，因此算法的并发执行和各个算法的执行调度十分重要。目前数据分析人员需要编写自己的调度算法，调度算法不具有通用性，也无法复用。

总而言之，Hadoop进行数据分析应用开发目前存在着很多的困难，开发效率低下，基于Hadoop的大数据分析应用开发平台有着很大的需求。

# 大数据分析应用开发平台的关键技术

为了满足大数据分析应用开发的需求，本文设计并实现了基于Hadoop的大数据分析应用开发平台。本章就其中的关键技术展开研究，提出基于Hadoop的大数据分析应用开发平台的概念模型。重点研究了基于组件技术和模型驱动的数据分析流程处理框架，基于嵌套图识别与分层技术的模型转换算法，基于HDFS的中间数据管理这三个关键技术。

## 系统架构

基于Hadoop的大数据分析应用开发平台是一个使用Hadoop进行大数据分析应用开发的一站式平台。该平台对用户屏蔽开发语言与运行平台，给用户提供独立的工作空间，提供统一的数据资源管理接口，支持数据分析应用开发，支持报表开发。

### 系统需求

本文的目标是设计与实现一个基于Hadoop的大数据分析应用开发平台。开发一个一站式的大数据分析应用开发平台可以极大地加快数据分析应用开发的速度，节省资源。一个完善的、一站式的大数据分析应用开发平台应该提供大数据分析相关的各个功能。下面介绍大数据分析应用开发平台的需求。

* + 屏蔽开发语言和运行平台，提供独立的工作空间



图 3‑1 大数据分析应用开发平台中的用户工作空间

大数据分析应用一般会运行在分布式集群上面，这些资源一般是共享的，因此开发平台，首先应该支持多租户（多个用户可以同时，并相互独立地使用平台）。每个用户具有独立的个人工作空间，用户可以在其中存储数据资源，开发数据分析应用，设计报表等。图3-1就是个人工作空间的一个示意图。

大数据分析应用开发平台还应该对用户角色进行划分。除了普通用户外，系统还需要管理员用户，负责用户管理、系统管理、集群管理等工作。

* + 异构数据资源的统一管理

大数据分析应用开发平台很重要的一个功能就是数据管理。每个用户都需要有独立的资源存储空间，该空间是私密的，有访问权限。为了支持大数据，一般会用到分布式文件系统作为基础存储服务。开发平台需要提供统一的资源管理界面，用户可以通过该界面进行浏览目录，查看文件、上传文件、下载文件等数据资源管理工作。

此外，大数据分析应用开发平台还应该集成Hive，HBase等分布式数据库，存储并管理格式化数据。大数据分析应用开发平台应该提供格式化数据的管理界面和类SQL的查询界面。用户可以通过该界面查看数据库结构、表结构、表内容，并执行SQL查询语言。

大数据分析应用开发平台应该提供异构数据资源的统一管理界面。

* + 数据分析应用的便捷开发

数据分析应用开发是大数据分析应用开发平台提供的最主要功能。

大数据分析应用开发平台需要提供常用的数据处理、数据分析、数据挖掘的算法，支持用户提交算法到开发平台的算法库中。用户利用开发平台提供的算法，快速构建数据分析应用。数据分析应用开发完成后，大数据分析应用开发平台负责这些算法的调度和执行。

平台还应该提供基于Web的友好的数据分析应用开发界面，用户可以可视化地设计数据分析应用。

* + 随时随地访问大数据

随时随地访问分析应用和数据逐渐成为常态。大数据分析应用开发平台应该可以将数据分析的结果可视化地展现出来，并且支持基于Web的访问，企业和个人才能快速利用这些结果数据。

数据展示有很多种形式，包括直方图、折线图、饼图、地理位置图、标签云、关系图等。大数据分析应用开发平台还需要提供报表开发，报表展示和报表生成等功能。

### 系统架构

基于上述需求和分析，本文开发一个基于Hadoop的大数据分析应用开发平台，改进现有大数据分析应用的开发模式。



图 3‑2 基于Hadoop的数据分析应用开发平台架构图

对于一般的数据分析需求来说，数据分析应用开发平台应该集成大部分常用的数据分析算法，这些算法可供用户选择，用户也可以提交自己的算法到开发平台中。基于Hadoop的大数据分析应用开发平台的架构如图 3‑2所示。在用户层，用户可以看到开发平台提供的组件列表，并使用这些组件列表构建数据分析流程。用户层的组件列表和模型层的组件库是一致的。用户设计完数据分析业务流程之后，将流程提交到开发平台，模型层会将用户的数据分析业务流程翻译成基于Oozie的执行流程。再由数据分析任务提交引擎将Oozie的执行流程以及相关的可执行组件提交到平台执行层——Hadoop分布式集群上面。开发平台的组件库中的算法，可以使用MapReduce编程框架，可以调用Mahout，Weka，R等算法库。

得益于上述架构，用户只需要直接使用开发平台提供的功能即可，无需将精力和时间耗费在与业务无关的IT系统管理和语言工具的学习上面了，可以极大地提高数据分析应用开发人员的工作效率。集群资源支持多用户共享，Hadoop和Oozie可以配合调度各个节点的负载，从而充分利用资源，节省大量成本。基于Hadoop的分布式并行计算还可以加快算法的执行速度，企业可以在更短的时间内做出明智的决策。

* + 基于Hadoop的大数据分析应用开发平台的优势

1. 给用户提供一站式的大数据分析应用开发平台。改变用户原有数据分析应用开发模式，将数据分析工程师从平台搭建、工具集成、API学习、算法组件管理、实验脚本编写、多任务调度等繁杂的工作中抽离出来。
2. 该开发平台以Hadoop分布式集群作为底层支撑，扩展性强，可以支持超大规模的数据分析应用。
3. 该开发平台借助分布式计算平台，降低流程的平均执行时间，提高集群资源的利用率。

## 基于组件以及模型驱动的数据分析业务流程处理框架

本节介绍数据分析组件的定义和实现方法，数据分析业务模型的定义，基于Oozie的数据分析执行模型以及数据分析业务流程处理框架。

### 数据分析组件

#### 组件的定义

**数据分析组件**是对数据处理、数据挖掘和数据统计等常用的功能和算法的抽象描述和实现。在需求和业务层面，也就是用户视角来看，每个组件包含一段功能描述和配置接口。比如数据清洗功能、聚类算法、分类算法等。从执行角度看，每个组件是一个可执行的节点，包含一段可执行的代码以及可调用的接口。从组件开发者角度来说，他们既要定义组件的描述及配置接口，又要编写可执行代码以及调用接口。如图 3‑3所示，系统通过组件库管理组件，用户将写好的组件提交到系统的组件库中，由系统管理这些组件。



图 3‑3 组件及组件库

组件模型中包含了组件的描述信息和可执行程序，组件描述信息如表格 3‑1所示。

组件可以分为数据资源组件（如文本组件，Hive组件），数据处理组件（如数据转换组件，数据提取组件），数据挖掘组件（如聚类组件，分类组件，关联规则提取组件），数据统计组件（如方差分析组件）等。组件的详细分类如表格 3‑2所示。

表格 3‑1组件描述信息定义

|  |  |
| --- | --- |
| **信息** | **详细** |
| 组件ID | 唯一的组件ID号 |
| 组件名称 | 组件显示在界面上的名称 |
| 组件分类 | 组件显示在界面上的分类，指明组件的功能类别，如数据采集组件，数据资源组件，数据挖掘组件等。主要为了方便用户查找所需要的组件。 |
| 组件类型 | 组件实际的类型，主要表明该组件是普通Java程序，还是 MAP\_REDUCE程序，还是Sqoop程序等。 |
| 组件配置信息 | 一般是算法或者功能的参数，比如K-means算法中的k参数（聚类个数）。 |
| 组件输入 | 约束该组件的输入，包括输入的文件个数，还有输入文件的类型等等，用来接收来自其它组件的输出。 |
| 组件输出 | 约束该组件的输出，包括输出的文件个数，还有输出文件的类型等等，用来将数据传递给下一个组件。 |

表格 3‑2数据分析组件模型分类

|  |  |
| --- | --- |
| **组件分类** | **组件列表** |
| 数据采集组件 | 数据抓取组件 |
| 数据资源组件 | Hive组件 |
| 普通文本组件 |
| ARFF文本组件 |
| CSV文本组件 |
| 数据导入组件 | Sqoop组件 |
| 数据处理组件 | 数据清理组件 |
| 数据过滤组件 |
| Hive Query组件 |
| 资源整合组件 |
| 数据挖掘组件 | 聚类组件 |
| 分类组件 |
| 关联分析组件 |
| 决策组件 |
| 异常检测组件 |
| 预测组件 |
| 数据统计组件 | 平均数计算组件 |
| 方差分析组件 |
| 相关分析组件 |
| 主成分分析组件 |
| 回归分析组件 |

在执行角度，组件也分为不同的类型。为了满足复杂的分析需求，系统组件的实现可以使用不同语言或者调用不同的工具包，如普通的Java组件、MapReduce组件、Hive组件、Sqoop组件等。具体的分类如表格 3‑3所示。

表格 3‑3组件类型

|  |  |  |
| --- | --- | --- |
| **组件类型** | **描述** | **举例** |
| 普通Java组件 | 普通Java程序编写而成，组件在单机上执行，不具备分布式能力。 | 如Weka组件，以及用户自己编写的单机版的处理少量数据的普通Java程序。 |
| MapReduce组件 | 用MapReduce编程框架编写的组件，运行在Hadoop集群上，具有分布式执行能力，可以处理大数据。 | 如Mahout组件，以及用户用MapReduce编程框架编写的数据处理分析的分布式组件。 |
| Hive组件 | 可以查询或者操作Hive数据库的组件。 | 如可以对Hive数据库执行SQL语句的组件。 |
| Sqoop组件 | 可以调用Sqoop进行数据导入、导出功能的组件。 | 可以执行Sqoop脚本的组件。 |
| 数据源组件 | 可以提供数据的加载地址和方式的组件。 | 如纯文本资源组件，可以配置文件所在的路径。 |
| 数据存储组件 | 可以提供数据存储的地址和存储方式的组件。 | 如纯文本的数据存储组件，可以配置文件存储路径。 |
| 中间数据组件 | 可以提供数据的复制和数据的汇总。 | 如将一份文件复制给多个接收组件，或者将多个组件的输出汇总成一个文件。 |

如何定义一个组件模型，统一不同类型组件的操作接口，并且方便用户开发自定义组件，是大数据分析应用开发平台开发的一个关键问题。

#### 基于Java注解的组件描述方法

* + 组件描述方法选择

存储组件类型的方式有很多种，如纯文本方式存储，格式化存储(XML/JSON)，数据库存储或者使用注解将描述信息放在类文件中。

Java注解的方式存储组件的元信息，具有多方面的优点：首先使用灵活，用户可以使用Java定义任意格式的注解。其次，编译时进行语法验证，再次，组件与组件描述信息绑定在一起，不存在不一致状态，最后，运行时信息加载到内存中，查询速度快。

但是使用Java注解的方式存储组件的元信息，也存在着缺点：组件的描述信息与组件的实现信息绑定在一起，即使用户只修改组件描述信息，也需要重新编译打包。

各种组件描述信息存储方式的优缺点对比，如表格 3‑4所示。

表格 3‑4 各种组件描述信息存储方式对比

|  |  |  |
| --- | --- | --- |
| **存储方式** | **优点** | **缺点** |
| 纯文本 | 1. 灵活，可以自定义存储格式和解析协议。 | 1. 需要自定义存储格式和解析协议。 2. 需要自己写程序读取并验证文件。 3. 组件与组件描述信息有可能会处于不一致状态。 |
| 格式化存储XML/JSON | 1. 较纯文本形式方便，有通用的语言规范。 2. 有工具可以验证并解析文件。 | 1. 组件与组件描述信息有可能会处于不一致状态。 |
| 数据库 | 1. 格式化存储，由数据库管理并约束信息的结构。 2. 有索引信息，方便实时的和复杂的查询。 | 1. 组件与组件描述信息有可能会处于不一致状态。 2. 因为要连接数据库，查询速度慢。 |
| 注解 | 1. 灵活，可以使用Java定义任意格式的注解。 2. 编译时就进行了格式验证。 3. 组件与组件描述信息绑定在一起，不存在不一致状态。 4. 运行时信息加载到内存中，查询速度快。 | 1. 组件描述信息修改后需要重新编译打包。 |

* + 基于Java注解的组件管理框架

开发平台需要一个开放的组件管理的框架，该框架应满足如下需求：组件提交到组件库中之后，开发平台会扫描该组件的信息，并将信息显示在界面上；用户可以在界面上使用该组件构建数据分析应用。平台扫描并管理组件库中组件信息的原理如图 3‑4所示。

平台可以管理带有注解信息的组件，基于以下事实基础：

1. Tomcat会监听lib目录下的文件变化，并自动通知Java虚拟机加载新添加的.class文件或者Jar包。
2. 程序员使用注解的形式将组件描述信息记录在Java类文件中，Java虚拟机通过反射解析这些信息。

用户将写好的组件编译成Jar包，放到应用服务器的lib目录下面，Tomcat监听到lib目录下的文件变化，会通知Java虚拟机加载Jar包中的文件。当用户刷新页面时，前端通过Restful API再次请求组件列表。服务器收到请求后会扫描虚拟机中加载的所有类文件，找到所有带有@Module注解的类，通过反射得到这些@Module注解中携带的组件配置信息，这些信息在下次更新前会被存储在内存中。组件管理模块将这些组件配置信息整理后返回给前端。

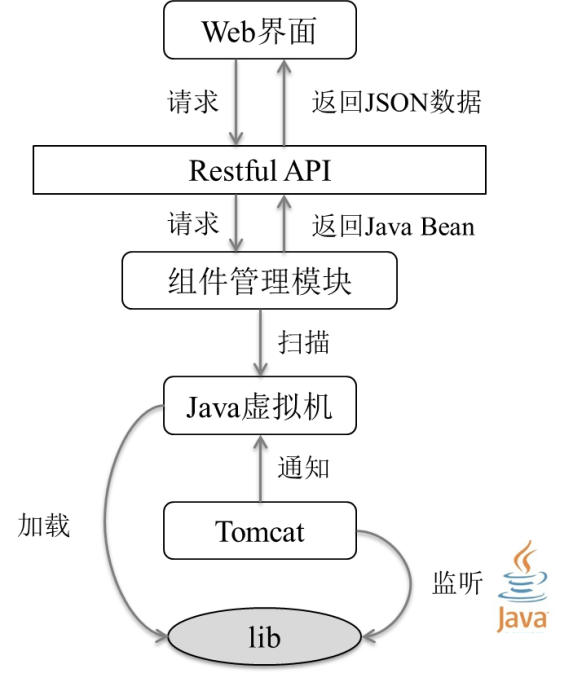


图 3‑4 基于Hadoop的大数分析应用开发平台组件管理原理图

#### 基于继承的可扩展组件实现方法

**数据分析组件模型**是对数据处理，数据挖掘，数据统计等常用的功能和算法的抽象描述和实现。在需求和业务层面，也就是用户视角来看，每个组件包含一段功能描述以及配置接口。比如数据清洗功能，或者聚类算法，或者分类算法等等。从执行角度看，每个组件是一个可执行的节点，包含一段可执行的代码以及可调用的接口。从组件开发者角度来说，他们既要定义组件的描述及配置接口，又要编写可执行代码以及调用接口。

大数据分析应用开发平台应该是开放的，可定制的。从而可以借助开源力量可以积攒一些优秀算法，成长为一个成熟、健全的工具。根据开放性原则，系统支持用户自定义组件。用户遵循规范，开发自己的组件并通过客户端上传组件。上传之后用户可以在客户端看到并使用该组件。同样系统也应该支持用户通过客户端卸载之前上传的自定义组件，那么用户在客户端就不可以使用该组件了。组件的可扩展性非常重要，它直接决定了系统的可用性，用户可以通过定制领域组件，构建一个面向领域的大数据分析应用开发平台。

在Oozie中，不同类型的执行节点有不同的参数配置和触发方式。本文分别给出了Java程序和Hive程序在Oozie配置文件中需要配置的参数信息。从两个节点的XML定义中，可以看到Java类型的节点，需要指定main函数所在的类，需要给出Java程序的输入参数。Hive节点不需要指定这些信息，Hive节点需要指定执行的Hive脚本。

Java程序在Oozie配置文件中需要配置的参数信息

|  |
| --- |
| *<xs:complexType name="JAVA">*  *<xs:sequence>*  *<xs:element name="job-tracker" … />*  *<xs:element name="name-node" … />*  *<xs:element name="prepare" … />*  *<xs:element name="job-xml" … />*  *<xs:element name="configuration" … />*  *<xs:element name="main-class" … />*  *<xs:choice … >*  *<xs:element name="java-opts" … />*  *<xs:element name="java-opt" … />*  *</xs:choice … >*  *<xs:element name="arg" … />*  *<xs:element name="file" … />*  *<xs:element name="archive" … />*  *<xs:element name="capture-output" … />*  *</xs:sequence>*  *</xs:complexType>* |

Hive程序需要在Oozie配置文件中配置的参数信息

|  |
| --- |
| *<xs:element name="hive" type="hive:ACTION"/>*  *<xs:complexType name="ACTION">*  *<xs:sequence>*  *<xs:element name="job-tracker" … />*  *<xs:element name="name-node" … />*  *<xs:element name="prepare" … />*  *<xs:element name="job-xml" … />*  *<xs:element name="configuration" … />*  *<xs:element name="script" … />*  *<xs:element name="param" … />*  *<xs:element name="argument" … />*  *<xs:element name="file" … />*  *<xs:element name="archive" … />*  *</xs:sequence>*  *</xs:complexType>* |

在Java中一般用Java类来表示一个概念，系统使用抽象类AbstractModule表示组件，用类中的方法来约束每个组件必须提供的接口。此外，开发平台借助Java多态实现不同类型组件的不同表现方式。例如，用户希望实现一个逻辑回归算法组件，就可以编写LogisticRegressionModule类，该类必须间接继承自AbstractModule类，并实现抽象类中的抽象方法。如果用户使用普通Java程序实现回归算法，则需要继承AbstractJavaModule；如果使用MapReduce实现回归算法，则需要继承AbstractMapReduceModule。

不同类别的组件需要实现不同的接口，Halfow提供了三类接口包括：AbstractHiveModule（Hive类型组件的接口），AbstractJavaModule（普通Java程序类型组件的接口），AbstractSqoopModule（Sqoop类型组件的接口）。不同类型的组件有不同的执行方式，需要相应地实现不同的接口。执行Hive程序的时候开发平台需要获取到的信息包括用户输入的SQL语句，以及最终结果的存储位置等。对于普通的Java程序，开发平台需要获取对应的入口类文件和主函数需要的参数。当系统有集成新的类型的组件的需求时，用户只需要添加对应的组件模板Abstract\*\*\*\*Module即可，然后在解释执行时按照需要的方式调用。



图 3‑5 组件继承关系定义

每个Jar包中可以有一个或者多个AbstractModule类的子类。组件继承关系如图 3‑5所示。

上述组件的设计方式具有以下优势：

1. 组件的元数据使用注解编写，Java编译器可以帮助用户进行语法检查。相比于使用配置文件来描述组件元数据来说，更方便用户开发调试。
2. 使用了Java的反射机制，用户可以动态地增加或者删除系统组件。
3. 使用Java继承来完成组件的定义，提高了代码的复用率。

用户在编写完算法或者功能之后，不再需要手工编写调度算法去管理并执行这些算法。用户首先根据算法的运行方式选择继承一个AbstractModule子类，然后添加Java注解@Module，根据注解规范定义组件的元信息，然后用户利用Eclipse或者Java Jar工具将必要的文件封装成Jar包，提交到平台。

平台在运行时根据不同的组件类型，获取不同的组件参数和XML配置模板，采取不同的翻译策略。

### 数据分析业务流程模型

一般来说，一个完整的数据分析任务可以划分为多个子任务，子任务之间会存在数据的传输和依赖。数据分析任务一般是针对特定应用场景和数据设计的，因此一般不具有可复用性。但是不同数据分析任务的子任务往往是可以复用的。本文定义了组件和流程的概念，组件对应分析任务的子任务，一个流程对应整个分析应用。在开发平台中，一个可复用的子任务为一个组件，组件接收来自其它组件的输入，并对输入的数据进行处理和转换，最后传输给其它组件。每个组件可以有零到多个输入和零到多个输出。数据流程由一到多个组件组成，组件之间通过有向路径连接，表示数据的传输方向。

模型是对客观现实的事物的某些特征与内在联系，所作的一种模拟或抽象。为了研究一个过程或事物,可以通过在某些特征(形状或结构等)方面与它相似的“模型”来描述或表示。它能反映出有关因素之间的关系。数据分析业务流程模型是从业务和需求角度描述数据分析任务的模型，它与实现方式、运行平台等无关。

在目前阶段，用户开发一个数据分析应用，一般需要通过编写脚本或者代码完成几个重要的步骤，首先是提取数据，然后对数据进行规范化转换，并进一步采用各种数据挖掘分析算法得出结果，最后将结果以图表形式的展示出来，整个过程构成一个完整分析流程。根据前述定义，组件是一个功能或者算法的抽象；因此**数据分析业务流程模型**是整个数据分析任务的抽象，用户可以使用多个组件构建数据分析工作流，该模型描述并抽象了数据分析应用的需求。



图 3‑6 组件定义

图 3‑6中显示的是一个组件，该组件包含了一个输入端点IN1，两个输出端点O1，O2。其中输入端点接收其它输出端点的输出，多个组件由有向路径连接。一个数据分析流程通常包含一到多个组件，这些组件相互连接成为一个有向无环图(DAG)。

* + 数据分析流程(DAG)定义

G=(V，E)。

V=(v1，v2，… ，vn)是DAG图节点（一个图节点对应一个组件实例）的集合。

vi=(ID，I，O，C)表示一个组件实例（由组件模板生成）。ID：唯一标识节点的ID号，由合法字符组成。I：组件实例的输入集合，一个组件实例由0个，1个或者多个输入组成。一般来说，数据资源组件没有输入，其它组件有一到多个输入。O：组件实例的输出集合，一个组件由0个，1个或者多个输出组成。一般来说，数据收集组件没有输出，其它组件有一到多个输出。C：组件实例实现相应功能所需要的参数集合。组件参数由一个key🡪value映射组成，key标识参数的名称，value标识参数值。当然，参数还可能有其它属性，如参数类型——纯文本，正则表达式，布尔值，参数描述信息等。

E=(e1， e2， …， en)是DAG图有向边的集合，它表示节点组件实例间的依赖关系和数据流向。

ei=(VB，PB，VE，PE)，表示一条边的实例。VB：表示ei数据传输有向边的起始节点。PB：表示ei与起始节点连接的端口。VE：表示ei数据传输有向边的终止节点。PE：表示ei与终止节点连接的端口。



图 3‑7 数据分析业务流程模型

由于有向无环图拓扑结构的复杂性，数据分析流程可以存在多种形态。各个节点之间可以串联（图 3‑8(a)），可以并发（图 3‑8(b)），可以汇合（图 3‑8(c)），多种连接关系又可以组合成多种形态。



图 3‑8 数据分析流程各种不同形态的子结构

### 基于Oozie的数据分析执行模型

Oozie是Hadoop生态系统中的一员，用于管理和协调多个运行在Hadoop平台上的作业。Oozie具有流程执行引擎必需的四个性质，分别为可扩展性，安全性，支持多租户和易用性[3]。Oozie的可扩展性包含两个方面，水平扩展和垂直扩展。水平可扩展性是指能够通过服务器数量的增加，而带来集群的存储能力和计算能力的提升。垂直扩展是指提高集群中现有节点存储容量和计算能力。Oozie支持多租户，也就是不同的用户可以同时登录到系统中，并提交任务。Oozie将不同租户的信息存到一个公共的数据库中，并给每个租户一个ID号。用户执行操作前都需要使用数据库中的信息对用户的权限进行验证。Oozie支持基本的Kerberos授权机制和Unix系统的用户/组授权机制，以及用户自定义的授权机制，同时也支持多个授权机制并存。Oozie提供了很多内置的机制来增加易用性。当任务中的一个节点失败了，则Oozie引擎提供自动重试的功能。当任务失败了，Oozie提供恢复功能，即之前已经成功完成的任务会被恢复。此外，Oozie支持通过Web客户端，命令行和Java API等多种方式查看当前的执行状态。在Hadoop集群资源充足的情况下，Yahoo已经验证单个Oozie服务进程每分钟可以接收1250个工作流任务(75K/hour)。

综上所述，Oozie是和Hadoop结合最紧密的，满足流程执行引擎必备的各种性质的，最适合调度Hadoop任务的执行引擎。

Oozie使用一个有向无环图(DAG)来定义一个任务执行工作流，DAG由控制节点(start, end, decision, fork, join, kill) 和动作节点(MapReduce, pig等)组成，这些节点由控制转移箭头来连接。Oozie使用基于XML的语言——hPDL (Hadoop Process Definition Language)，定义这些任务以及任务之间的关联关系。一个Oozie的执行流程示意图如图 3‑9所示。



图 3‑9 Oozie执行流程图示例

图中有背景颜色的节点为控制节点，其它节点为动作节点。整个Oozie流程从Sqoop节点开始，然后并发出两条分支，分别执行MapReduce任务和Pig任务，两个分支的任务全部执行完毕后进入Decision节点，根据判定结果，决定直接进入File-System节点还是先通过Java节点再进入File-System节点，File-System节点执行完毕后整个Oozie流程结束。

Oozie使用XML语言描述类似的流程，下面详细介绍Oozie流程中的节点类型以及不同类型节点定义的语法。

**控制流节点(Control Flow Nodes)**：控制流节点定义了流程的开始和结束，以及控制流程的执行路径(Execution Path)，如decision、fork、join等。

**动作节点(Action Nodes)：**触发一个功能或者算法的执行。动作节点包括Hadoop MapReduce、Hadoop文件系统、Java、Pig、SSH、HTTP、email和Oozie子流程等。每个功能节点都有两个转移出口，=ok=和=error=。如果动作节点执行成功，则会执行ok指向的节点，否则执行error指向的节点。

表格 3‑5 Oozie控制节点分类表

|  |  |
| --- | --- |
| **节点名称** | **描述** |
| Start | Start节点是流程的入口点，它指明了流程需要执行的第一个节点。每个流程必须有且仅有一个Start节点。 |
| End | End节点是整个流程执行的终点，如果流程执行到End节点则表明流程执行成功。 |
| Kill | Kill节点表明流程异常终止。 |
| Decision | Decision节点使得流程可以根据条件选择接下来的执行路径。 |
| Fork | Fork节点可以将一条执行路径变成多条可并发的执行路径。 |
| Join | Join节点汇总Fork节点并发出来的多条路径，等待所有并发路径执行结束后再执行接下来的流程。 |

表格 3‑6 Oozie动作节点分类表

|  |  |
| --- | --- |
| **节点名称** | **描述** |
| MapReduce | MapReduce动作节点执行一个MapReduce任务。 |
| Pig | Pig动作节点启动一个Pig任务。在XML里面需要配置Job-tracker，name-node，pig脚本等信息。 |
| HDFS | HDFS动作节点允许Oozie工作流管理HDFS上的文件信息，可以执行包括move，delete，mkdir，chmod等在内的命令。 |
| Ssh | Ssh动作节点可以在远程机器上执行shell脚本命令。 |
| Java | 当Oozie流程执行到Java动作节点后，会调用XML中配置的类文件中的main方法。 |

控制节点语法及使用示例——开始节点语法

|  |
| --- |
| *<workflow-app name="[WF-DEF-NAME]" xmlns="uri:oozie:workflow:0.1">*  *...*  *<start to="[NODE-NAME]"/>*  *...*  *</workflow-app>* |

Start节点是最简单的控制节点之一，它只有一个to属性，用来指明执行流程开始后第一个执行的节点的名称。

动作节点语法及使用示例——Java节点语法

|  |
| --- |
| *<workflow-app name="[WF-DEF-NAME]" xmlns="uri:oozie:workflow:0.1">*  *...*  *<action name="[NODE-NAME]">*  *<java>*  *<job-tracker>[JOB-TRACKER]</job-tracker>*  *<name-node>[NAME-NODE]</name-node>*  *<prepare>*  *<delete path="[PATH]"/>*  *<mkdir path="[PATH]"/>*  *...*  *</prepare>*  *<job-xml>[JOB-XML]</job-xml>*  *<configuration>*  *<property>*  *<name>[PROPERTY-NAME]</name>*  *<value>[PROPERTY-VALUE]</value>*  *</property>*  *...*  *</configuration>*  *<main-class>[MAIN-CLASS]</main-class>*  *<java-opts>[JAVA-STARTUP-OPTS]</java-opts>*  *<arg>ARGUMENT</arg>*  *...*  *<capture-output />*  *</java>*  *<ok to="[NODE-NAME]"/>*  *<error to="[NODE-NAME]"/>*  *</action>*  *...*  *</workflow-app>* |

Oozie运行一个Java程序，需要指定job tracker，name node，前期准备工作——如新建文件夹、删除文件等，Java的main函数所在的类，main函数的输入参数等，还需要指定：如果节点执行成功下一个需要执行的节点（ok后面的节点）；如果节点执行失败下一个需要执行的节点（error后面的节点）。

### 数据分析业务流程处理框架

前面定义了数据分析组件模型和数据分析业务流程模型的概念，接下来将研究基于模型驱动和组件技术的数据分析流程处理框架。

在传统的数据分析应用开发模式中，用户需要编写大量代码，包括每一个算法，以及算法间的调度和合作。基于Hadoop的大数据分析应用开发平台打破原有的开发模式，提出了一种基于组件的数据分析应用开发模式。在新的开发模式下，用户只需要完成数据分析业务流程模型的建模，然后数据分析业务流程模型的验证，翻译，执行都交给开发平台去完成。



图 3‑10 数据分析应用执行原理图

如图 3‑10所示，用户设计的数据分析业务流程模型会被开发平台的模型转换引擎转换成数据分析执行模型，该执行模型再被放到集群上去真正执行。

数据分析业务流程模型是从业务和需求角度描述数据分析任务的模型，它与实现方式、运行平台等无关。它是整个数据分析任务的抽象，用户可以使用多个组件构建数据分析工作流，该模型描述并抽象了数据分析应用的需求。

执行模型是从平台和程序执行角度描述数据分析任务的模型，它与实现方式和运行平台相关。Oozie是Hadoop生态系统中的一员，用于管理和协调多个运行在Hadoop平台上的作业。Oozie使用一个有向无环图(DAG)来定义一个任务执行工作流，DAG由控制节点(start, end, decision, fork, join, kill) 和动作节点(MapReduce, pig等)组成，这些节点由控制转移箭头来连接。

模型转换引擎的任务是完成数据分析业务流程模型到数据分析任务执行模型的转换。然后由流程提交引擎提交到Hadoop集群上去执行。

如图 3‑11所示当用户发出执行流程的命令之后，系统首先验证流程的合法性，包括每个节点的参数配置是否正确，节点的输入输出是否完整，整个流程图是否含有环等。流程验证通过之后，系统会对流程进行转换，将数据分析业务流程转换成真正的Oozie执行流程。然后再将Oozie执行流程翻译成Oozie流程描述配置文件（XML文件）。系统整理Oozie的配置文件和相关的Jar包等文件，形成本地工作目录，再上传到集群的分布式文件系统(HDFS)上面。所有数据准备就绪，系统会通知Oozie开始执行任务。



图 3‑11 数据分析应用执行流程图

## 基于嵌套图识别与分层的模型转换算法

在基于模型驱动的数据分析流程处理框架中，核心的部分是模型转换引擎。在这一步，系统需要将基于数据流的数据分析业务流程模型转换成基于Oozie的控制流执行模型实例。

### 数据流模型和控制流模型

目前为止，对于流程的描述模型多种多样，其中最重要的两种类型是：数据流模型和控制流模型，分别可以由数据流图和控制流图来描述。

**数据流图：**英文缩写DFD(Data Flow Diagram)，可以用来描述数据流模型。数据流图以图形的方式描绘信息流和数据从输入移动到输出的过程中所经受的变换。RapidMiner，Clementine，Weka等数据分析工具使用数据流图表示数据分析业务流程模型。

**控制流图：**英文缩写CFD(Control Flow Diagram)。可以用来描述控制流模型。控制流图是程序执行中所有可能的事件顺序的一个抽象表示。Oozie使用控制流图来描述流程执行模型。控制流图由控制节点组成，这些节点定义了工作流的开始和结束（start、end和kill结点），并且提供了一套控制流执行路径的机制（decision，fork和join结点）。

表格 3‑7现有工具使用的模型类型调研结果

|  |  |  |  |
| --- | --- | --- | --- |
| **工具** | **工具属性** | **模型类型** | **备注** |
| RapidMiner | 数据挖掘工具 | 数据流模型 | 业务模型 |
| Clementine | 数据挖掘工具 | 数据流模型 | 业务模型 |
| Weka | 数据挖掘工具 | 数据流模型 | 业务模型 |
| Oozie | 分布式流程执行调度引擎 | 控制流模型 | 执行模型 |

从表格 3‑7中可以看出，数据流模型主要用来对数据分析业务进行建模，控制流模型主要用在执行调度引擎中。

在大数据分析应用开发平台环境下，若使用控制流来描述上层的业务模型，模型之间的转换将非常直接，业务模型图中的节点与Oozie的执行模型原语一一对应；但这种设计的最大缺陷是平台本身无法管理任务之间的输入输出数据，其可靠性无法保证。

数据流业务模型有如下优点：首先，减少用户参数配置。在控制流模型中用户需要手工配置每个节点的输入输出并保证其一致性。在数据流模型中流程的中间数据由系统自动管理，无需手工配置，用户只需要配置整个流程的输入数据和输出数据的路径信息即可。其次，明确流程中的数据流有利于管理中间数据。平台可以在流程提交后即对前后节点的数据格式进行验证，而不需要将验证推迟到流程执行时。这个优点对于耗时的数据分析流程来说是非常重要的。此外，数据流业务模型比控制流业务模型更加简洁，减少了用户对执行逻辑的控制，可以更加专注在业务模型本身。

表格 3‑8两中模型的对比

|  |  |  |
| --- | --- | --- |
|  | **模型A** | **模型B** |
| **模型类型** | 控制流模型 | 数据流模型 |
| **组件分类** | 控制组件&功能组件 | 资源组件&功能组件 |
| **有向边含义** | 组件执行先后顺序 | 数据流向 |
| **翻译难度** | 简单 | 复杂 |
| **中间数据与实验结果** | 不区分 | 区分 |
| **中间数据配置** | 手工 | 自动 |

当然，数据流模型在某些方面也存在缺点：首先，数据流模型的翻译比较复杂，算法复杂度较高，翻译速度较慢（大数据环境下，流程翻译代价相对于流程执行速度可以忽略不计）。其次，只能通过数据的流向控制流程执行顺序，减弱了用户对流程执行的控制。综合考虑，Haflow采用数据流模型来描述分析业务。

表格 3‑8详细对比了两种模型分别作为数据分析业务流程模型的优点和缺点，从表中可以看出数据流模型更加适合数据分析业务建模。

### 数据流模型到控制流模型的转换

3.2.1章介绍了数据分析业务流程模型，它包括组件和有向边，组件分为计算组件和资源组件，有向边表示数据的流向。基于Oozie的数据分析执行模型，包括节点和有向边，节点分为控制节点和动作节点，有向边表示节点的先后执行顺序。

两种模型有相似点也有不同点，接下来本文将研究如何完成数据分析业务流程模型（数据流模型）到Oozie执行模型（控制流模型）的转换。

表格 3‑9 数据分析业务流程模型与Oozie执行模型的组件类型对比

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **数据分析业务流程模型** | |  | **基于Oozie的执行模型** | |
| 计算组件 | 普通Java组件 | 动作节点 | 普通Java节点 |
| MapReduce组件 | MapReduce节点 |
| Hive组件 | Hive节点 |
| Sqoop组件 | Sqoop节点 |
| …… | …… |
| *资源组件* | 数据源组件 | *控制节点* | Start |
| 数据存储组件 | End |
| 中间数据组件 | Kill |
|  | Fork/Join |
|  | Decision |

从表格 3‑9中可以看出，数据分析业务流程模型的计算组件可以与Oozie执行模型的动作节点进行一对一的映射，而数据分析业务流程模型的资源组件在数据流模型下没有存在的必要。因此数据流模型到控制流模型转换的核心工作是添加Oozie执行模型的控制节点。基于这些规则，可以得到如下的模型转换算法。

* + 模型转换步骤
* 输入：数据流图——*Dgraph*
* 输出：Oozie控制流图——*Cgraph*，及其配置文件
* 步骤：
  1. **验证图*Dgraph*** ：包括输入，输出，参数配置信息等；
  2. **验证中间数据** ：根据数据流向，验证中间数据格式是否匹配；
  3. **删除数据节点** ：删除Dgraph中所有的数据节点，并将数据节点中的配置信息传递给邻近的功能节点；
  4. **删除冗余边；**
  5. **生成控制流图Cgraph** ：确定Dgraph中功能节点的并发顺序，添加Fork和Join节点，生成控制流图Cgraph；
  6. **补全Cgraph** ：为Cgraph添加Start节点，End节点，Kill节点，完成Cgraph的构建；
  7. **生成Oozie配置文件**：根据Oozie的XML规范，将图*Cgraph*直译成Oozie配置文件。

算法的输入*Dgraph*是用户设计完成的数据分析流程，在算法中该图结构存放到内存中。算法的最终目的是要生成Oozie控制流图及其配置文件。第一步，算法首先验证Dgraph是否正确，包括输入，输出是否完整，参数配置是否符合约束，图中是否存在环等。接下来验证有向边的连接是否合法，主要验证有向边连接的两种数据类型是否兼容。因为控制流图中没有数据节点，算法的第三步去掉Dgraph中的所有数据节点，但是要保留数据节点中的配置信息。这一部分会在中间数据管理部分详细介绍。信息合并之后，此时原始的Dgraph只剩下计算节点了，第四步，就是为Dgraph添加最重要的Fork和Join节点，从而生成控制流图。第五步，算法为整个图添加Start节点，End节点和Kill节点，此时Dgraph已经完成了到控制流图的转化。Start节点没有输入节点，并且指向第一个执行的节点，该节点可能是计算节点，也可能是Fork节点。End节点没有输出节点，表示整个流程执行的结束。每个动作节点都会有一条路径表示Error，Kill接收所有的Error路径。最后，根据Oozie的XML规范，将Cgraph直译成Oozie的配置文件。

下面详细介绍其中的每一步。

* + 第③步 删除数据节点

因为基于Oozie的执行模型中没有数据节点，因此算法中有一步就是删除Dgraph中所有的数据节点。直接删除数据节点会连带将数据节点中配置的参数删除，因此删除之前算法要将数据节点携带的信息传递给邻近的计算节点。

这一步主要完成以下功能：将数据源组件的资源配置参数传递给接下来的功能组件；将数据接收组件的资源配置参数传递给上一个功能组件；将数据复制节点的配置参数传递给接下来的组件；自动生成Dgraph中间数据的配置信息；删除资源组件。

在该算法中，假定：资源组件包括：数据源组件和数据接收组件。数据源组件的组件类型为SOURCE，数据接收组件的组件类型为DEST，数据复制组件的组件类型为MIDDLE\_DATA，其余的节点都是普通的计算节点。

前置条件：数据源组件不能与数据接收组件连接，资源组件只能有一个输入或者是一个输出，数据复制组件只能有一个输入。

输入：有向无环图（数据流图）Dgraph =(V，E)，V={v1, v2, …}，E={e1, e2, …}，边ei连接节点vi\_s和Vi\_t。

数据节点删除算法

|  |
| --- |
| for every vertex vi in topology order do  for every adjacent edge ei of vertex vi do  *//生成每条数据边对应的中间数据的存储路径*  if type of vi\_s is SOURCE and type of vi\_t is DEST then  throw illegal state exception  else if type of vi\_s is SOURCE then  *//边起始节点的路径即为中间数据的存储路径*  put the configurations of vi\_s to the input configurations of vi\_t  else if type of vi\_s is MIDDLE\_DATA then  *//起始节点的输入参数即为中间数据的存储路径*  put the input configurations of vi\_s to the input configurations of vi\_t  else if type of vi\_t is DEST then  *//目的节点的路径即为中间数据的存储路径*  put the configurations of vi\_t to the output configurations of vi\_s  else  *//随机生成边上的中间数据的存储路径*  generate random storage path for the edge and put the configuration to the output configurations of vi\_s and to the input configurations of vi\_t  endif  endfor  endfor |

该算法首先对Dgraph进行拓扑排序，保证所有节点间的偏序关系。然后根据拓扑排序的结果依次遍历每个节点，得到当前节点的邻接边，根据起始节点和目标节点的类型确定边上的中间数据的存储路径。并将该边对应的中间数据存储位置作为输入输出参数放到边的起始节点和目标节点的配置参数集合中。最后删除图中所有的SOURCE和DEST节点以及其关联的边。

* + 第④步 删除Dgraph中的冗余边

冗余边定义：删除后不影响节点间的拓扑关系的边。冗余边的存在对于控制流图的生成没有任何意义，因此为了简化图结构，算法的初步阶段就删除了数据流图中的冗余边。

冗余边举例，如图 3‑12所示：

* 图-1中的边A为冗余边；
* 图-2中的边B或者C为冗余边。
* 图-3中没有冗余边。

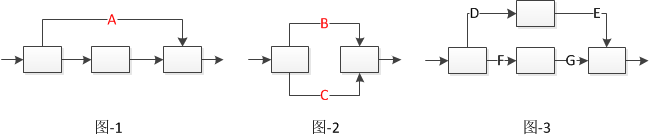


图 3‑12 冗余边举例

冗余边删除算法：

输入：有向无环图G=(V，E)，V={v1, v2, …}，E={e1, e2, …}，边ei连接节点vi\_s和Vi\_t。

输出：删除冗余边的有向无环图G’

算法：

1. For every ei in E
2. do deep first search from vi\_s，mark every vertex that visited
3. if Vi\_t is marked then
4. do delete ei from E
5. endif
6. endfor

算法的主要思想是，遍历图中的每条边。尝试删除这条边之后，如果从该边的起始节点还能够遍历到该边的目标节点，则该边为冗余边，需要从图中删除。

* + 第⑤步：生成控制流图

算法的第五步是整个模型转换的核心，这一步主要是确定节点间的并发关系，为简化后的数据流图添加Fork/Join节点。

* + Oozie文档中对Fork和Join的解释

“A fork node splits one path of execution into multiple concurrent paths of execution.”[15]

Fork节点将一条执行路径并发成多条执行路径。

“A join node waits until every concurrent execution path of a previous fork node arrives to it.” [15]

Join节点等待所有Fork节点的并发执行路径都执行结束之后再继续执行。

“The fork and join nodes must be used in pairs. The join node assumes concurrent execution paths are children of the same fork node.” [15]

Fork节点与Join节点必须成对使用，Join节点只能接收来自同一个Fork节点的并发路径。

Fork/Join的语义

|  |
| --- |
| *Fork*  *C1 一个执行子图*  *| C2 一个执行子图*  *| … …*  *| Cn 一个执行子图*  *Join* |

Fork的语义是将一条路径并发成多条执行路径，Join节点的语义是等待Fork节点的并发执行路径都执行结束之后再继续进行。但是Oozie的并发语义中一个很强的约束是Fork和Join节点必须成对使用。这表明不同的Fork/Join对之间的关系可以是嵌套，可以是并列，但是不能相互穿插。

这种带有较强约束的语义决定了，并不是所有的依赖图都可以直接翻译成Fork/Join表达的并发。

* + 满足如下语义的子图可以直接翻译成一个Fork/Join组合

|  |
| --- |
| P( C1, P( C2, . . . P( . . . , Cn) . . . ))  each Ci may be expressed as :  S( pi1, S(pi2, . . . S( . . . , pim) . . . ))  其中，  P(P1, P2)表示P1和P2可以并发执行，并且P1和P2最终会汇合于一点。  S(P1,P2)表示P1和P2串行执行，即P1执行结束后执行P2 |

如图 3‑13所示，该图满足定义的语义，可以直接翻译成Fork/Join组合。

如图 3‑14所示，该图不满足定义的语义，不能直接翻译成Fork/Join组合。

* + 翻译策略

对于不同的图类型，采用不同的翻译策略。

判断Dgraph是否满足Fork/Join语义

* 如果满足，则按照图结构添加Fork/Join节点
* 否则，按照图的层次关系分层添加Fork/Join节点

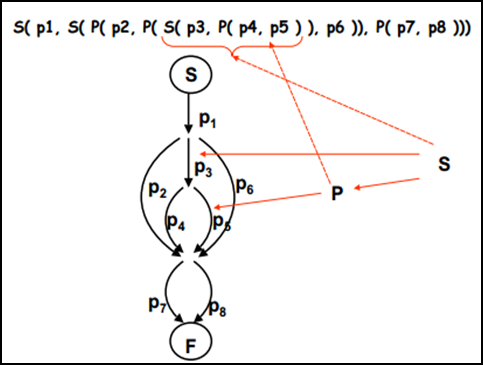


图 3‑13 满足Fork/Join语义的子图



图 3‑14 不满足Fork/Join语义的子图

* + 判断Dgraph是否满足Fork/Join语义

首先需要判断数据流图Dgraph是否满足Fork/Join语义，具体算法如下：

输入：有向无环图G=(V，E)，V={v1, v2, …}，E={e1, e2, …}，边ei连接节点vi\_s和Vi\_t。

输出：图G是否满足Fork/Join语义

判断数据流图是否满足Fork/Join语义的算法

|  |
| --- |
| 1. for each vertex vi do ( search by level which was calculated) 2. If vi has more than one output edge then //Fork-like node 3. vi’s is-fork-like ← true 4. vi’s all edges ***fork-parent*** ← vi 5. else then //has only on output edge 6. If vi has only input edge then //normal node 7. vi’s only output edge’s ***fork-parent*** ← the only input edge’s fork-parent 8. else then //Join-like node 9. **if all the previous edge has the same fork-parent** then do 10. vi’s only output edge’s ***fork-parent*** ← the previous nodes’ fork-parent 11. else do 12. **return false** 13. endif 14. endif 15. endif 16. endfor |

|  |  |
| --- | --- |
| 图 3‑15 不满足语义的算法实例 | 图 3‑16 满足语义的实例 |

如图 3‑15所示，当遍历到节点v5的时候，边A的fork-parent是v2，边B的fork-parent是v3，因此违背了算法第9步中的if条件，返回false。图中从每条边引出的虚线所指向的节点就是这条路径的fork-parent。

如图 3‑16所示，每个汇合节点（v7和v8）都满足第9步中的if条件，整个图遍历结束就意味着该图满足Fork/Join的语义。图中从每条边引出的虚线所指向的节点就是这条路径的fork-parent。

* + Dgraph满足Fork/Join语义，添加Fork/Join节点

在判断出数据流图满足Fork/Join语义之后，就开始按照图结构添加Fork节点和Join节点。添加策略：在每个有多个输出的节点之后添加一个Fork节点，如图 3‑17所示；在每个有多个输入的节点之前添加一个Join节点，如图 3‑17所示。完整的例子如图 3‑18所示。



图 3‑17 添加Fork/Join节点



图 3‑18 添加Fork/Join节点

* + Dgraph不满足Fork/Join语义，分层添加Fork/Join节点



图 3‑19 使用基于分层的模型转换算法示例

首先遍历数据流图，确定每个节点所在的层次，并标记。然后分层遍历，给每层分别添加Fork/Join节点。这种做法并不能达到最高的并发程度，但是却支持任意数据流图的转换。

如图 3‑19所示，节点v1是第一层节点，v2，v3是第二层节点，v4，v5，v6是第三层节点，v7是第四层节点，v8是第五层节点。将图分层后，逐层添加Fork/Join节点，得到下面的控制流图。（注意，只有一个节点的层次前后不需要添加Fork/Join节点。）这样得到的流程的执行时间由每层执行时间最长的节点决定。这样的翻译策略翻译得到的控制流图，执行时间不一定是最短的，但是却一定是正确的。

* + 添加控制流图的其它节点

除了Fork/Join节点，还需要为控制流图添加Start，End，Kill等基础控制节点。前面的算法保证了整个控制流图只有一个入口节点，添加Start节点，指向这个入口节点；同样前面的算法也保证了整个控制流图只有一个出口节点，添加End节点，然后添加出口节点到End节点的有向边即可。最后添加Kill节点，算法为控制流图中的每个计算节点添加error出口，然后指向Kill节点。这样每个计算节点出现异常时，流程就会进入Kill节点，然后整个流程执行结束。

* + 翻译成Oozie配置文件

生成了控制流图之后，算法可以直接将控制流图翻译成Oozie配置文件。

## 基于HDFS的中间数据管理

数据分析应用开发平台以流程来表示具体的任务，而流程各个任务之间通过数据输入和输出依赖形成拓扑关系。这些中间任务输出的数据也称为中间数据，具有生命周期短、存储异构等特点。Haflow通过规则匹配策略约束数据传递，通过存储位置自动生成实现中间数据的管理。

### 异构中间数据的管理



图 3‑20 异构中间数据管理示意图

大数据分析应用开发平台的基础架构依赖于底层的Hadoop生态系统，并使用Oozie作为流程执行引擎；其重要的特性就是灵活开放，支持各种大数据处理分析工具的组件化接入。但这些不同的组件也带来了异构中间数据的管理挑战。中间数据的异构性分为两个方面。一方面是介质异构性：不同组件输出的中间数据可能存储在不同的介质上，例如本地文件系统，HDFS分布式文件系统，关系型数据库，分布式NoSQL数据库等。另一方面是格式的异构性：中间数据可能是纯文本、CSV等多种格式。

为解决上述存储介质异构性难题，大数据分析应用开发平台提供了异构中间数据管理策略，如图 3‑20所示；其核心是将HDFS分布式文件系统作为统一管理的基础，涉及数据的导入、导出和数据的集中管理。具体功能包括：支持用户将本地文件系统的文件上传到HDFS，支持关系数据库的数据导入到HDFS或者Hive上面，支持将分布式数据库中的文件导出到HDFS上面。这样保证了大部分的数据资源最终都可以存储到HDFS上面，因此Haflow中的组件只需要通过相应路径从HDFS上面获取数据即可。

### 中间数据的自动管理

复杂的大数据分析任务会生成大量的中间数据，平台提供这些中间数据的自动化管理。用户在设计整个数据分析流程时只需要给出整个流程的输入所在的目录，以及整个流程的输出所在的目录即可。中间节点产生的中间数据存储位置由大数据分析应用开发平台自动管理。前驱节点产生的数据存储到自动生成的目录下面，后继节点运行时再从相同的目录下读取数据。当流程执行结束之后，用户可以手动将这些中间数据清除，或者指定中间数据自动清除的规则。

大数据分析应用开发平台为每个用户创建一块独立的HDFS存储空间，用户可以向这个空间上传文件或者从这个空间下载文件到本地。中间数据管理模块会为每个用户在各自的存储空间中创建一个目录存储中间数据。在该目录下用户，用户的中间数据按照项目组织，并且以路径名命名，用户可以查看并检索这些中间数据。

### 中间数据的匹配验证

针对于中间数据的数据格式异构性难题，大数据分析应用开发平台提供了自适应的格式匹配验证策略，提供用户透明的转换功能，表格 3‑10归纳总结了Hadoop环境下一些常见的存储格式。

在模型转换过程中很重要的一步是要验证数据分析模型的正确性，其中包括了中间数据是否匹配的验证。

* + 中间数据的匹配规则

这里定义数据分析模型对应的有向无环图(DAG)中的有向边的起始点为PS，终止点为PE。

1. PS与PE类型完全匹配，验证通过。
2. PS与PE类型不同，但是PS是PE的子类型（如文本文件是所有格式的子类型，JPG是二进制文件的子类型），验证通过。

除上面提到的两种情况外，其它情况验证不通过。

在数据分析业务流程模型中有一类特殊节点——数据拷贝节点，这个节点没有具体的功能，但是可以将一份数据的地址传递给多个组件。这个节点只能有一个输入，这个输入的数据格式定义为自动格式（下面称：AUTO）。AUTO输入节点的数据格式是在翻译时决定的，与连接输入节点的有向边的起始点数据类型一致。数据拷贝节点的输出节点的数据格式是在输入节点格式确定后确定的，并且与起始节点的数据格式保持一致。AUTO标记的数据格式只能出现在数据拷贝节点上，并且在数据分析流程图第一遍扫描之后销毁。

表格 3‑10 常见存储格式总结

|  |  |  |
| --- | --- | --- |
| 所有格式 | 文本文件 | ARFF文件 |
| CSV文件 |
| XML文件 |
| JSON文件 |
| TXT文件 |
| …… |
| 二进制文件 | 图片(JPG/PNG/…) |
| 模型文件 |
| DOC/DOCX |
| …… |
| …… | …… |

# 基于Hadoop的大数据分析应用开发平台的设计与实现

本章介绍基于Hadoop的大数据分析应用开发平台——Haflow的设计与实现。首先给出了系统的架构，然后详细介绍了系统中各个模块的实现，最后使用河南省新农合数据分析案例和标签云生成案例验证了系统的功能。

## 系统架构及相关实现技术

### 系统架构

系统的总体目标是实现一个灵活易用可扩展的基于Hadoop的大数据分析应用开发平台，其具体架构如图 4‑1所示。

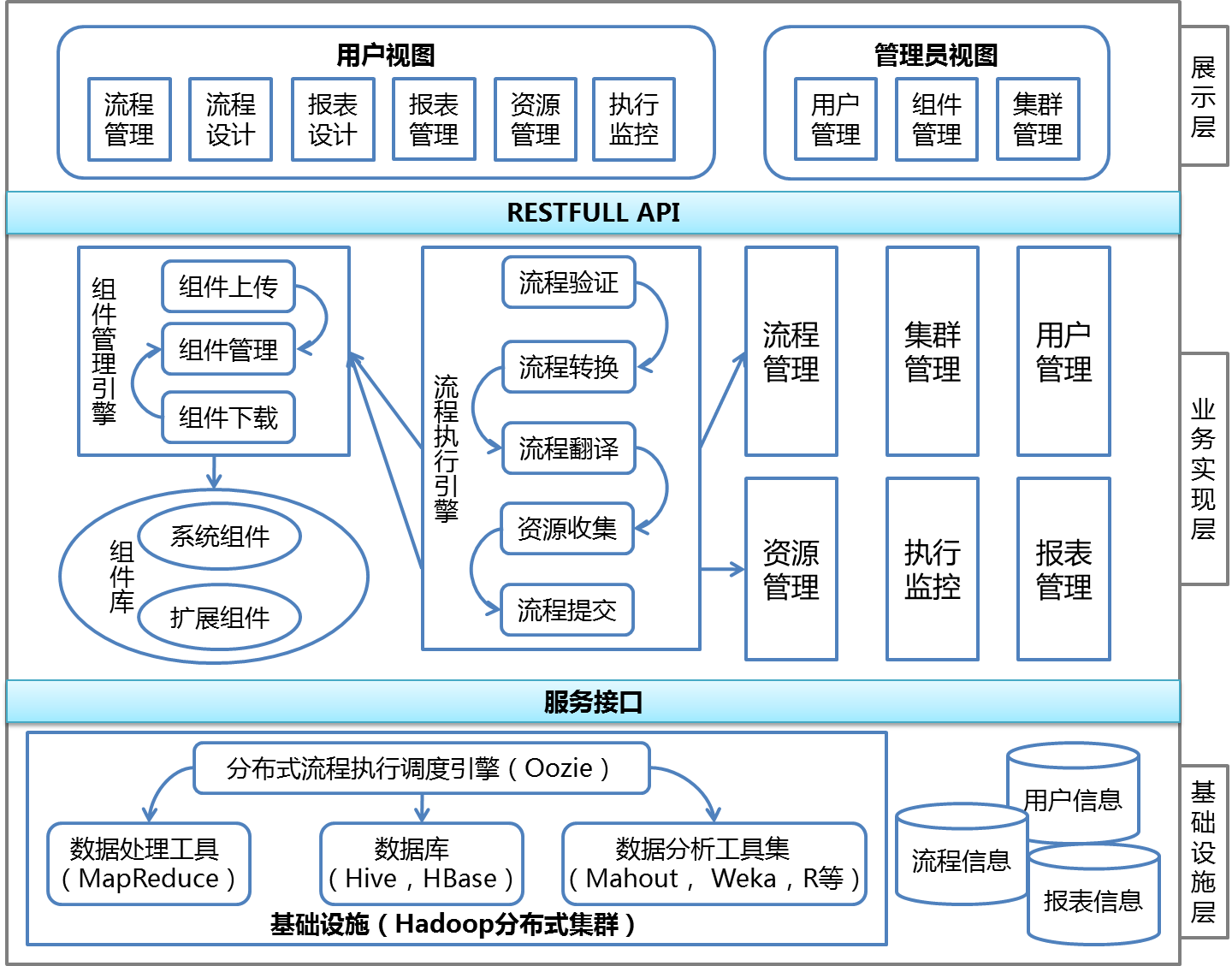


图 4‑1 系统架构图

基于Hadoop的大数据分析应用开发平台分为展示层，业务实现层和基础设施层。

展示层提供两个视图，一个是用户视图，一个是管理员视图。用户通过用户视图可以使用系统提供的流程管理，流程设计，报表设计，资源管理，执行监控等功能。从管理员视图可以使用系统提供用户管理，组件管理，集群管理等功能。展示层通过RESTFUL API访问业务实现层提供的服务。

业务实现层调用基础设施层提供的服务，实现并提供展示层所需要的服务，如流程管理，资源管理，集群管理，执行监控，用户管理，报表管理等。组件管理引擎管理组件信息，提供组件上传和下载的接口。流程执行引擎接收用户数据分析流程设计模型，将模型转换、翻译成可执行模型。资源收集模块收集执行模型配置文件、数据文件和执行文件，并将其提交到Hadoop集群上。流程提交模块调用基础设施层提供的服务接口。

基础设施层提供大数据分析应用的数据存储和程序执行平台。主要包含MySQL数据库，存储用户信息。分布式存储HDFS和数据仓库Hive，存储用户的数据。Mahout，Weka，R等，提供数据分析工具。分布式流程执行调度引擎Oozie，提供用户数据分析应用程序的调度。

* + 基于Hadoop的大数据分析应用开发平台的系统特色

1. Haflow提供一个基于Web的客户端，是一个网页版的大数据分析应用开发平台，用户可以随时随地应对大数据。
2. Haflow是一个开放式的大数据分析应用开发平台，可以借助开源力量累积分析组件。
3. Haflow底层基于Hadoop平台，集群可以扩展，支持超大规模数据的处理和分析。
4. Haflow的组件管理框架可以协同异构数据分析工具，应对复杂分析需求。
5. 该系统支持多用户，多任务。每个用户可以连续提交多个任务，多个任务在后台并发执行。

### 相关实现技术

* + JsPlumb

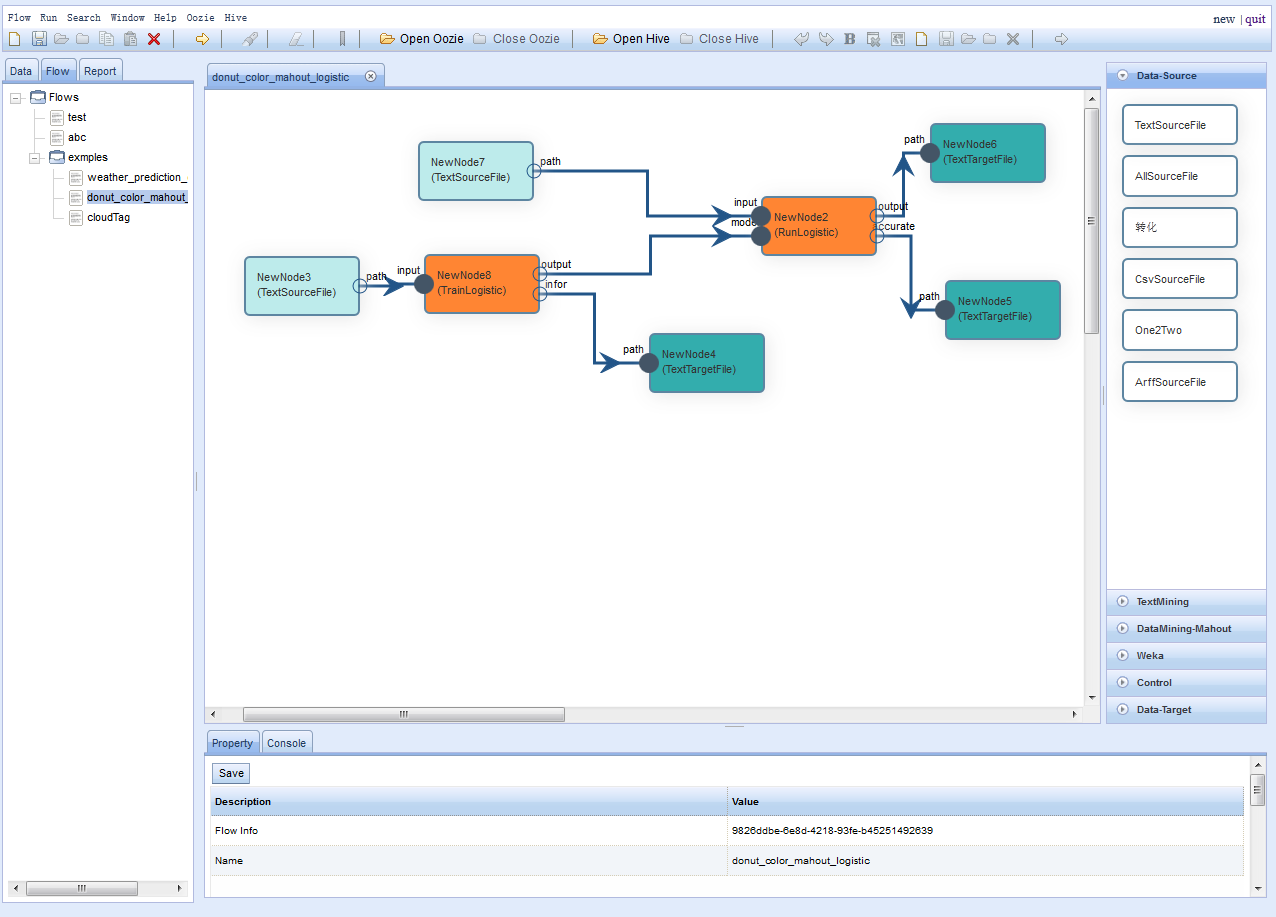


图 4‑2 JsPlumb绘图示例

JsPlumb是网页上的支持用户可视化链接各个元素的工具包。如图 4‑2是Haflow中利用JsPlumb在网页上绘制出的数据分析流程图。

结合JQuery的drag/drop功能，Haflow可以给用户提供拖拽式的流程设计界面。用户拖拽右侧的组件模板到流程设计面板上面，然后再用有向边将这些组件连接起来。

* + Dojo

Dojo是前端开发框架，使用JavaScript实现，它解决了DHTML的浏览器兼容，风格不统一等问题。Haflow使用Dojo开发了前端界面，包括表格展示、图表展示等。

* + Restful

REST（英文：Representational State Transfer，简称REST），是一组架构约束条件和原则。Haflow遵循REST风格，Web前端使用GET、POST、PUT和DELETE四种方式访问服务器。

* + Spring

Spring是一个开源框架，Haflow主要使用了Spring的控制反转(IOC)框架。Spring可以实现单例对象的自动生成和对象间的自动关联，使得Haflow的代码十分的简洁，并且方便复用。

* + Hibernate

Hibernate是一个开源工具，它的ORM工具可以完成关系型数据库到Java Object的自动映射，方便程序员访问数据库。Haflow使用了Hibernate，加快了开发速度。

## 系统总体设计

本小节介绍基于Hadoop的大数据分析应用开发平台——Haflow的整体设计。首先介绍系统架构，然后介绍系统的数据库设计。

### 系统的包图和模块图

平台整体遵循MVC框架，分为控制器(Controller)，视图(View)，模型(Model)。视图负责展示数据分析流程和系统中的组件。控制器负责调用基础服务，响应用户请求，修改模型中的数据。

* + 系统的层次结构

系统是一个多层架构，上面的层可以调用下面的层，下面的层不依赖上面的层。如图 4‑3所示：



图 4‑3 系统包图

ui.controller：提供前端访问所需的Rest接口。

ui.model：前端与controller之间传递数据的数据规范。

ui.helper：调用service提供的服务，并将结果保存到ui.model中的数据对象中。

engine：负责数据分析流程的转换和执行。

service：提供系统的后台基础服务。

pojo：包含了service用到的所有普通的Java对象。

dto：所有与数据库相关的Java对象在该包下面。

util：工具包，包含了如MD5值生成，配置信息读取，特殊字符转换，还有Java类查找等工具类。

* + service层提供的服务

组件开发者和分析应用开发者通过系统提供的Web界面或者是REST API访问系统服务，service层提供的服务如表格 4‑1所示。

系统提供的服务主要基于四个基础服务层：组件管理层(Module Layer)，Oozie访问层(Oozie Client Layer)，HDFS访问层(HDFS Access Layer)，持久层(Persistent Layer)。控制器调用服务层(Service Layer)接口，响应用户请求，如图 4‑4所示。

Module Service：该服务管理Haflow的组件库，扫描加载组件描述信息，并提供组件信息查询接口。

Oozie Client Service：该服务负责调用Oozie的服务接口。主要用来将模型转换引擎生成的Oozie配置文件和相关配置信息提交给Oozie，并通知Oozie执行该流程。

HDFS Access Service：该服务是最基础的服务，提供HDFS文件系统的访问服务。Haflow提交Oozie配置文件，上传下载数据文件，报表读取表格数据都需要通过该服务。

Persistent Service：持久化服务，提供数据库访问接口。Haflow中，用户相关的信息都是存储在数据库中的。比如，用户名，用户权限，用户所拥有的数据，用户所创建的流程，用户所创建的报表，都是存储在数据库中的。

表格 4‑1 service层提供的系统服务

|  |  |
| --- | --- |
| HDFSService | HDFS相关服务 |
| LocalFileService | 本地文件系统相关服务 |
| FlowService | 流程相关服务 |
| FlowExecuteService | 流程执行相关服务 |
| HiveService | Hive相关服务 |
| ModuleService | 组件信息相关服务 |
| NodeAppearanceService | 节点外观信息服务 |
| NodeConfigurationService | 节点配置信息服务 |
| ReportService | 报表相关服务 |
| RunHistoryService | 执行历史相关服务 |
| UserService | 用户相关服务 |



图 4‑4 Service层模块图

### 系统的数据库设计

图 4‑5是Haflow系统后台数据库的实体关系映射图。存储的主要是用户的信息，还有用户所拥有的报表和流程的信息。

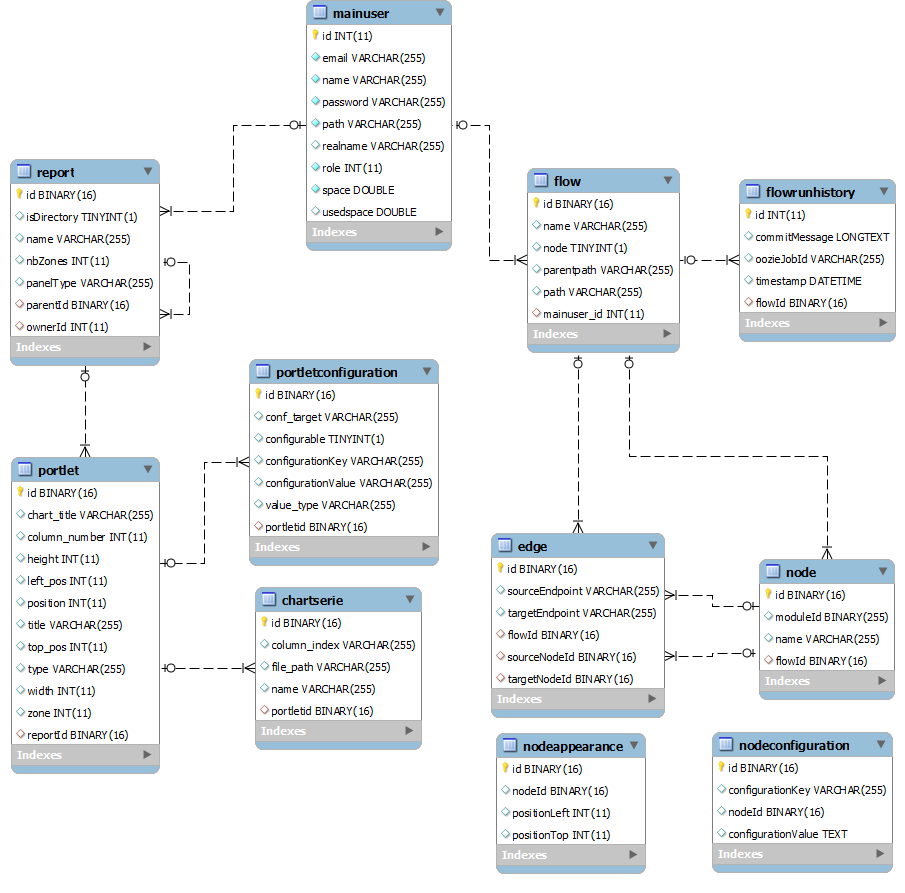


图 4‑5 Haflow数据库的实体关系映射图

mainuser保存的是用户的个人信息，包括用户的ID，名字等。

flow保存的是用户设计的所有流程的信息，包括流程的ID，名称等。

node保存的是流程的节点信息，包括节点的ID，名称，所属的流程等。

edge保存的是流程的边信息，包括ID，连接的节点，所属的流程等。

nodeconfiguration保存的是节点的配置信息，包括key值，value值，所属的节点等。

nodeappearance保存的是节点的样式信息，包括节点的位置等。

flowrunhistory保存的是用户的执行历史，包括执行时间，执行日志等。

report保存的是用户的报表信息，包括报表的ID，名称，类型等。

portlet保存的是报表上的图表信息，包括图表的ID，类型，位置，大小，所属的报表等。

portletconfiguration保存的是图表的配置信息，包括配置信息的key值，value值，类型，目标等。一个图表可能有零到多条配置信息。

chartserie保存的是图表的数据信息，包括数据的路径，列号等。一个图表可能有一到多个数据。

## 可扩展组件的实现

在Haflow中，用户基于组件开发数据分析应用。下面介绍在Haflow中对组件描述信息的定义，组件的实现方法以及组件的管理。

### 组件的定义和实现方法

组件的信息通过注解进行描述，包括组件的ID，名称，组件的输入，组件的输出，组件的类型，组件的配置信息等。具体的注解结构如图 4‑6所示：



图 4‑6 描述组件的注解的类图

用户将自己的算法封装成新的组件时需要编写注解信息。如图 4‑7所示，这段代码实现了一个简单的云标签生成组件。这个组件的名称是CloudTag，分类是TextMining，组件类型是Java，没有配置参数，有一个纯文本的输入和一个纯文本的输出。

|  |
| --- |
| **package** haflow.module.CloudTag;  @Module(id = "add600a8-ab63-8901-ca46-aaffa0d0cd5f",  name = "CloudTag", help="标签云生成",category = "TextMining",  type = ModuleType.*JAVA*,  configurations = {},  inputs = { @ModuleEndpoint(name = "input\_file", minNumber = 1, maxNumber = 1, dataType = DataType.*PlainText*) },  outputs = { @ModuleEndpoint(name = "output\_file", minNumber = 1, maxNumber = 1, dataType = DataType.*PlainText*) })  **public** **class** CloudTagModule **extends** AbstractJavaModule {  @Override  **public** **boolean** validate(Map<String, String> configurations,  Map<String, String> inputs, Map<String, String> outputs) {  // **TODO** Auto-generated method stub  **return** **false**;  }  @Override  **public** String getMainClass() {  **return** CloudTag.**class**.getName();  }  } |

图 4‑7 编写一个简单的组件的示例代码

Haflow平台的组件开发需要遵循三个方面的原则。首先，用户开发的类需要采用统一的包名(haflow.module)。因为平台识别自定义组件的过程中会在特定的路径下查找组件。其次，所有组件的实现都必须间接继承自AbstractModule基类，如上述例子中AbstractJavaModule就是AbstractModule的子类。最后，组件对应的类需要实现AbstractJavaModule中的所有抽象方法。在例子中，用户实现了validate方法和getMainClass方法。在validate方法中用户可以自己定义组件验证的规则，验证通过则返回true，否则返回false。getMainClass给出了算法运行时的main函数所在的类。当Oozie在执行该组件时会调用CloudTag这个类中的main函数，并传递相应的参数。

### 组件管理框架的实现

组件管理模块负责加载Jar包中的组件描述信息。ClassHelper负责扫描Jar包中的所有类文件。ModuleSerivice通过反射从这些类文件得到注解信息，从而解析出组件的描述信息，并将这些描述信息存放到Map中。



图 4‑8 组件管理模块类图

* + 类ClassHelper
* List<String> getClassNames( String packageName, Boolean searchInChildPackage )

@参数 packageName 需要搜索的包

@参数 searchInChildPackage 是否搜索子文件

@返回值 类名的列表

* + 类ModuleService
* public Module getModule(UUID moduleId)

@参数 moduleId 组件的ID号

@返回值 组件的注解

* public boolean removeModule(UUID moduleId)

@参数 moduleId 组件的ID号

@返回值 是否成功删除组件

* public Map<Module, AbstractModule> searchForModules(String packageName)

@参数 packageName 需要搜索的包

@返回值 组件的注解到带有该注解的类的对象的映射

* public Map<UUID, Class<?>> searchForModuleClasses(String packageName)

@参数 packageName 需要搜索的包

@返回值 组件的ID到带有该注解的类的映射

* public Map<UUID, Module> searchForModuleProtypes(String packageName)

@参数 packageName 需要搜索的包

@返回值 组件的ID到组件注解的映射

## 模型转换引擎的实现

* + 模型转换引擎模块包图

用户使用系统提供的数据分析组件，设计自己的数据分析业务流程模型，提交到Haflow。Haflow负责将用户定义的数据分析流程翻译成基于Oozie的执行模型，并通过Oozie的Client接口提交到Hadoop中执行。这一部分对外主要有两个接口，一个是validateFlow，一个是runFlow。数据分析模型转换翻译模块的包图如图 4‑9所示。



图 4‑9 系统核心模块包图

如图 4‑9所示，流程执行核心功能的实现主要封装在两个包内：module包和engine包。

module：提供组件的抽象基类AbstractModule，及其基本子类AbstractHiveModule，AbstractJavaModule等的定义。

module.data：提供基本的资源组件，包括输入组件和输出组件。

module.\*\*\*：该包依赖module包，用户可以根据组件规范编写自己的组件。

engine：提供流程翻译执行模块的对外接口。

engine.oozie：提供基于Oozie实现的翻译执行引擎。

engine.model：提供翻译执行引擎所需要的普通Java对象。

### 流程图建模

在Haflow中，数据流图和控制流图的底层模型都是有向无环图。Haflow中有向图的类图如图 4‑10所示。

Node：是有向图中的节点类。封装了节点的ID号，名称，描述，使用的组件模板以及所属的数据分析流程。

Edge：是有向图中的有向边类。封装了有向边的ID号，所属的数据分析流程，起始节点，目标节点，起始端点，目标端点。

AdjMatrixNode：是Edge的辅助类，封装了Edge，并增加index属性和pathValue属性。

DirectedGraph：是有向图的类，存储了类的图结构。



图 4‑10 有向图的类图



图 4‑11 操作有向图的类图

在Haflow中对有向图有很多操作，比如拓扑排序，有向环检测等，它们对应的类图如图 4‑11所示。

GraphTransform：将数据流图转换成控制流图的类，只有一个transform方法。

DepthFirstSearch：使用深度优先搜索算法对有向图进行遍历，得到前序遍历的顺序，后序遍历的顺序，以及逆后序的顺序。

DirectedCycleDetection：有向图中的环检测。

TopologicalSort：首先判断有向图是否无环，然后对有向无环图进行拓扑排序的类，得到排序顺序。

### 模型转换算法实现



图 4‑12 流程执行引擎类图

图 4‑12是模型转换模块的核心类图，AbstractEngine提供模块的对外接口，主要提供模型验证和模型执行两个接口，如下表所示。

表格 4‑2 模型转换模块对外接口

|  |  |  |
| --- | --- | --- |
| **方法名** | **参数** | **返回值** |
| validateFlow | Flow | ValidateFlowResult |
| runFlow | Flow | RunFlowResult |

AbstractEngine是一个抽象类，OozieEngine实现了AbstractEngine定义的抽象方法，实现了基于Oozie的数据分析应用的执行功能。OozieEngine在完成功能时调用了service层提供的服务，包括组件服务(ModuleService)和HDFS服务(HDFSService)，oozie层提供的服务，包括流程部署服务(FlowDeployService)和Oozie任务提交服务(OozieService)，util层提供的集群配置信息获取服务(ClusterConfiguration)。

OozieEngine还调用了oozie层的OozieFlowXMLGenerator服务，该服务主要功能是根据Flow的信息生成Oozie执行所需要的xml文件。首先，OozieFlowXMLGenerator调用GraphTransformer将DirectedGraph转换成Oozie需要的图结构。接下来OozieFlowXMLGenerator逐步生成每个节点对应的XML信息，最后将所有XML片段拼接成整个流程的XML配置文件。



图 4‑13 生成Oozie配置文件的类图

OozieFlowXMLGenerator：接收数据流图，生成Oozie执行实例的配置文件。OozieFlowXMLGenerator首先调用GraphTransformer将数据流图转换成控制流图，然后调用NodeGenerateHelper生成Oozie的控制节点。调用OozieNodeXMLGenerator，生成控制流图中的每个节点对应的XML。最后将XML片段拼接成完整的Oozie配置文件。

* + Haflow实现的模型转换的三个实例

如图 4‑14，图 4‑15，图 4‑16所示，左边的图是使用Haflow设计的数据分析应用模型，右侧为Haflow执行翻译算法后生成的基于Oozie的控制流图。

在图 4‑14中，左侧的数据分析应用模型中有四个资源节点，两个计算节点，两个计算节点之间是串行的关系。右侧翻译生成的控制流图中有两个控制节点Start（图中的空心圆节点）、End（图中的实心圆节点），两个串行的动作节点（图中的两个长方形节点）。

在图 4‑15中，左侧的数据分析应用模型中有三个资源节点，四个计算节点，其中有两个计算节点是并发关系，两个并发节点与剩余节点为串行关系。右侧翻译生成的控制流图中除了Start、End之外，还有两个控制节点Fork（图中的正三角节点）和Join（图中的倒三角节点）。

在图 4‑16中，左侧的数据分析应用模型中有两个资源节点，十一个计算节点，因为该模型不满足Oozie的Fork/Join语义，因此使用基于层次的并发策略。右侧翻译生成的控制流图中，对数据分析应用模型中的多余一个节点的层添加了Fork/Join节点。

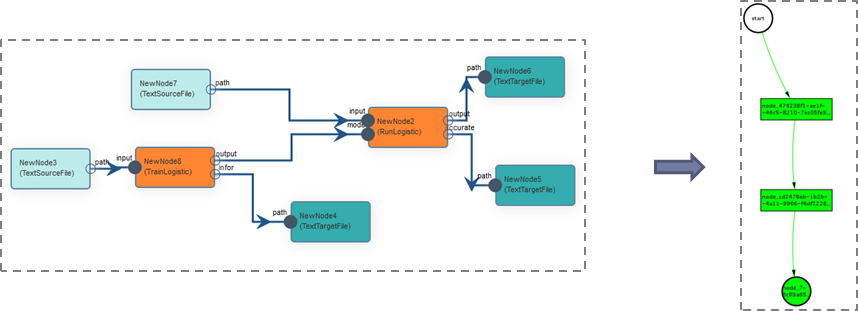


图 4‑14 模型转换算法运行案例1

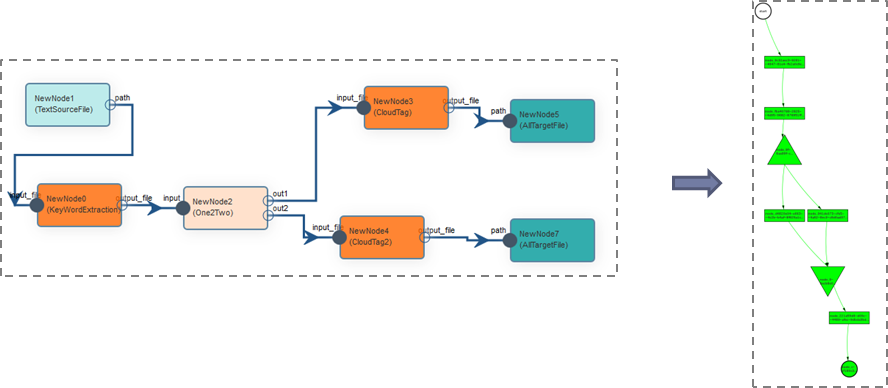


图 4‑15 模型转换算法运行案例2

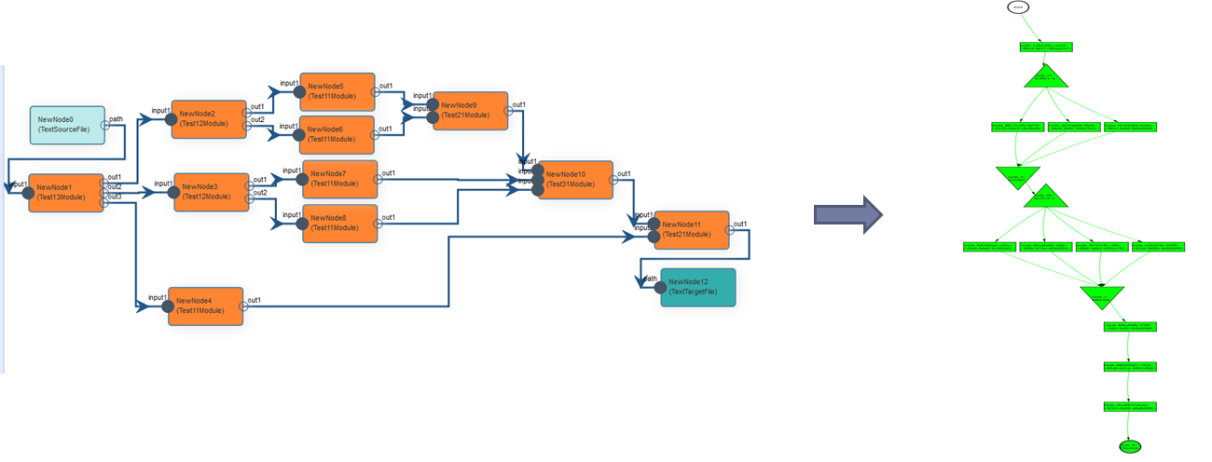


图 4‑16 模型转换算法运行案例3

### 数据分析应用翻译执行的序列图



图 4‑17 数据分析应用翻译执行序列图

用户点击“运行”按钮后，前端将Ajax请求发送到服务器端。Spring将请求派发给RunFlowController，RunFlowController调用RunFlowHelper，RunFlowHelper调用FlowExecuteService，FlowExecuteService调用FlowService得到Flow的详细信息，然后调用OozieEngine的runFlow方法。

OozieEngine调用ModuleService得到流程中各个组件的描述信息。仿照Flow的结构在内存中创建DirectedGraph对象。验证DirectedGraph对象是否合法，包括：是否有环，是否连通，是否完整，两个节点间连接是否合理等。Flow的每个节点的配置信息是否填写完整并且符合规范。如果验证通过，接下来利用Flow的信息，生成DirectedGraph对象。利用模型转换算法将数据分析业务流程模型转换成基于Oozie的执行流程。然后利用组件的描述信息生成流程执行所需要的Oozie的配置文件(XML)。生成成功之后，Haflow将生成的XML文件以及相关的Jar部署到Tomcat所在的机器上，部署结束后再将整个项目提交到HDFS的工作目录中。到此为止，任务的翻译生成已经结束，最后Haflow通知Oozie执行该任务。Oozie提交任务后，可以通过其返回的任务ID，对整个任务的执行过程进行监控。

## 应用验证

本节针对基于Hadoop的大数据分析应用开发平台——Haflow的设计目标进行了应用验证。首先是使用河南省新农合数据分析项目验证了Haflow的统一资源管理和数据可视化功能，然后使用了云标签生成项目验证了Haflow的可扩展组件模型以及大数据分析应用开发功能。

### 实验环境



图 4‑18 Haflow应用验证试验环境

Haflow是一个Web项目，部署在图 4‑18中所示的服务器上面，服务器的配置如图 4‑21所示。Haflow会访问拥有9个节点的Hadoop集群，其中一台为master节点（配置如图图 4‑19所示），8台为slave节点（配置如图 4‑20所示）。用户可以通过浏览器在线访问Haflow。



图 4‑19 master0机器配置及安装软件



图 4‑20 slave\*机器配置及安装软件



图 4‑21 服务器配置及安装软件

### 河南省新农合数据分析

* + 背景

河南新农合数据分析应用需求出自“农村医疗卫生服务平台与应用示范”项目。此次实验从数据库服务器中导出两个县（约8万条）的数据作为分析样本，主要包含农民就医时的补偿兑换信息，河南省各个地区新农合的基金支付情况等。目标是实现五个典型的数据分析需求，包括：累计补偿分析，基金支付情况分析，月支付分析，累计支付率分析，累计支付额分析。

* + 解决方案

河南省新农合的原始数据存储在Oracle数据库中，统计分析结果是使用五个复杂的SQL查询完成的。解决方案如下：首先，通过Sqoop将Oracle中的数据导入到Hive中，Hive和Oracle中的表一一对应。接下来，利用Haflow的Hive资源管理界面，进行SQL数据查询，并将查询结果导出。最后，利用Haflow的报表设计界面生成五个分析报表。

* + 详细步骤

1. 使用Haflow中的资源导入导出界面，将Oracle的数据导入到Hive中，使得Oracle和Hive中的表一一对应。

表格 4‑3 导入Hive的四张表

|  |  |
| --- | --- |
| **表名** | **行数** |
| dim\_mv\_areacode | 54434 |
| fact\_mv\_cli\_areacode | 2572 |
| fact\_mv\_hosp\_areacode | 21425 |
| fact\_mv\_joinfiance | 2 |

1. 进入Haflow的资源管理界面，从Hive资源导航窗口，打开新农合数据库的SQL查询界面，编写HiveQL，执行查询得到统计数据。如图 4‑22所示，左侧为Haflow中的Hive资源导航窗口，中间为SQL查询界面以及查询产生的结果。重复这个过程，完成五个复杂SQL的查询，得到五个统计分析结果。

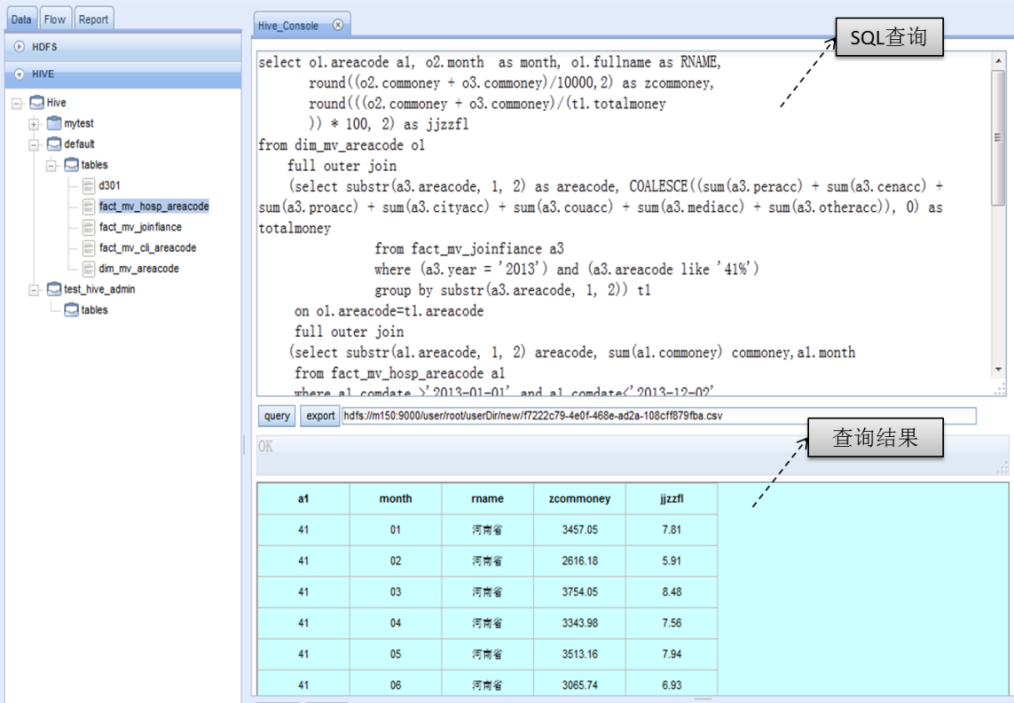


图 4‑22 通过Hive资源管理界面执行md\_all\_sql3.sql脚本

1. 进入Haflow报表设计界面，如图 4‑23所示，新建报表，分别拖拽饼图，柱状图以及折线图模板到中间的报表设计界面。将报表的数据源配置为上一步得到的五个数据分析结果，配置图表参数，得到需求指定的报表。

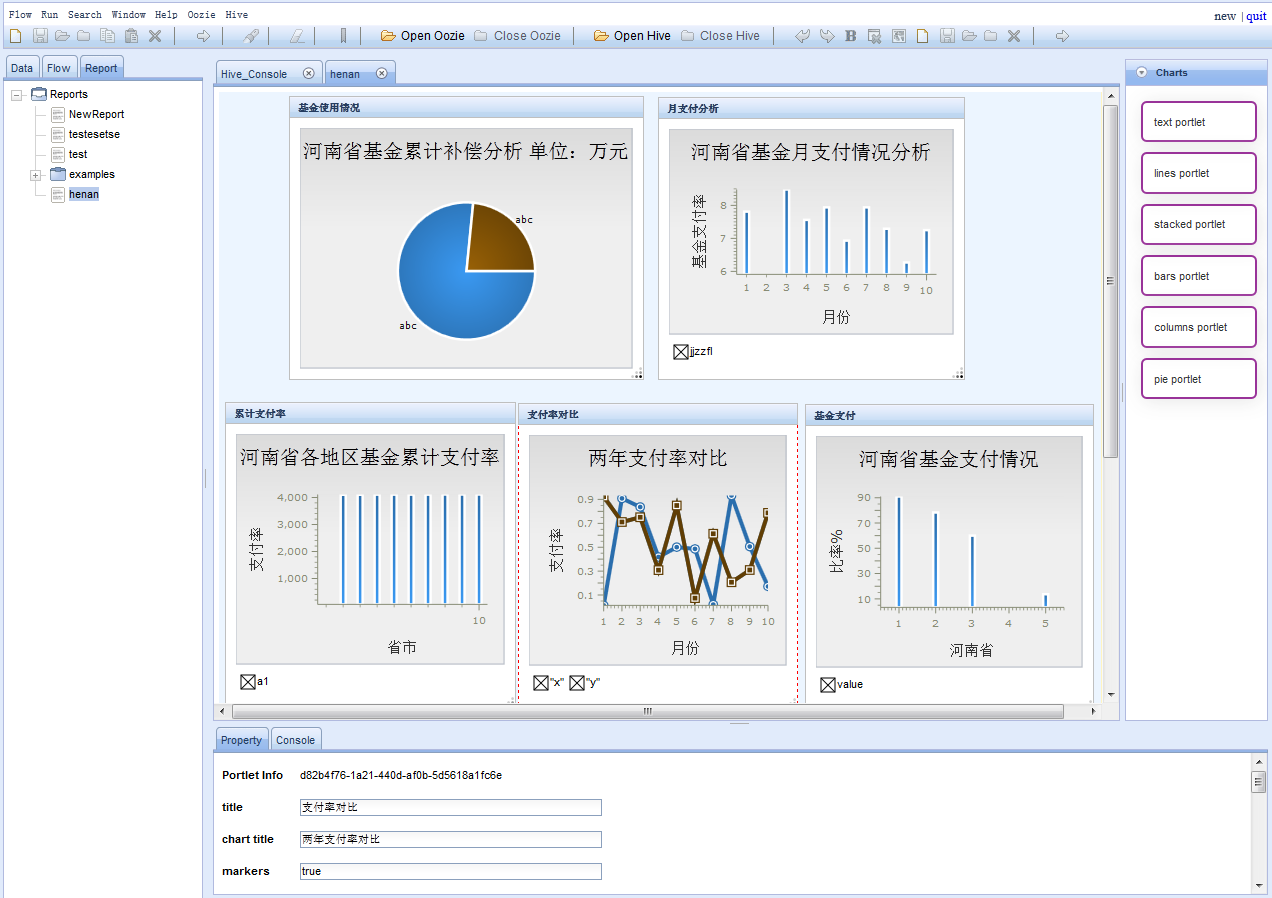


图 4‑23 Haflow生成的相关报表

### 标签云生成

* + 背景

标签云算法统计文本中的关键词词频，利用字体大小和颜色表明这些关键词的重要性，生成关键词图。

* + 解决方案

生成标签云是个多步骤的过程，首先要获取文字内容，然后利用分词算法统计关键词的词频，最后利用关键词和词频生成标签云。这个场景适合使用Haflow中的数据分析应用开发工具来开发。

* + 详细步骤

1. 在Eclipse中新建关键词提取项目，引用Haflow组件扩展包，完成组件编写，用Eclipse将项目导出为Jar包。打开Haflow，登录进入系统组件管理界面，提交关键词提取算法的Jar包。用同样的方法，提交标签云生成算法的Jar包。Haflow的组件库中增加了关键词提取组件和标签云生成组件。
2. 打开Haflow，以普通用户账号登入，在数据分析应用管理界面，新建标签云生成应用。打开标签云数据分析应用，拖拽关键词提取组件、标签云生成组件以及相关的资源组件到数据分析应用开发面板，将这些组件连接起来形成数据分析流程，配置参数，保存。设计好的数据分析应用模型如图 4‑24所示。
3. 在Haflow的数据分析应用管理界面，运行“云标签生成”应用，数据分析应用被提交到Haflow服务器端执行。Haflow服务器端收到“执行”指令后，将标签云生成数据分析应用翻译成Oozie的执行流程，再将Oozie执行流程提交到Hadoop集群上执行。Oozie执行流程提交成功后，用户可以通过Oozie监控界面监控应用的执行状态，如图 4‑25，图 4‑27，图 4‑26所示。

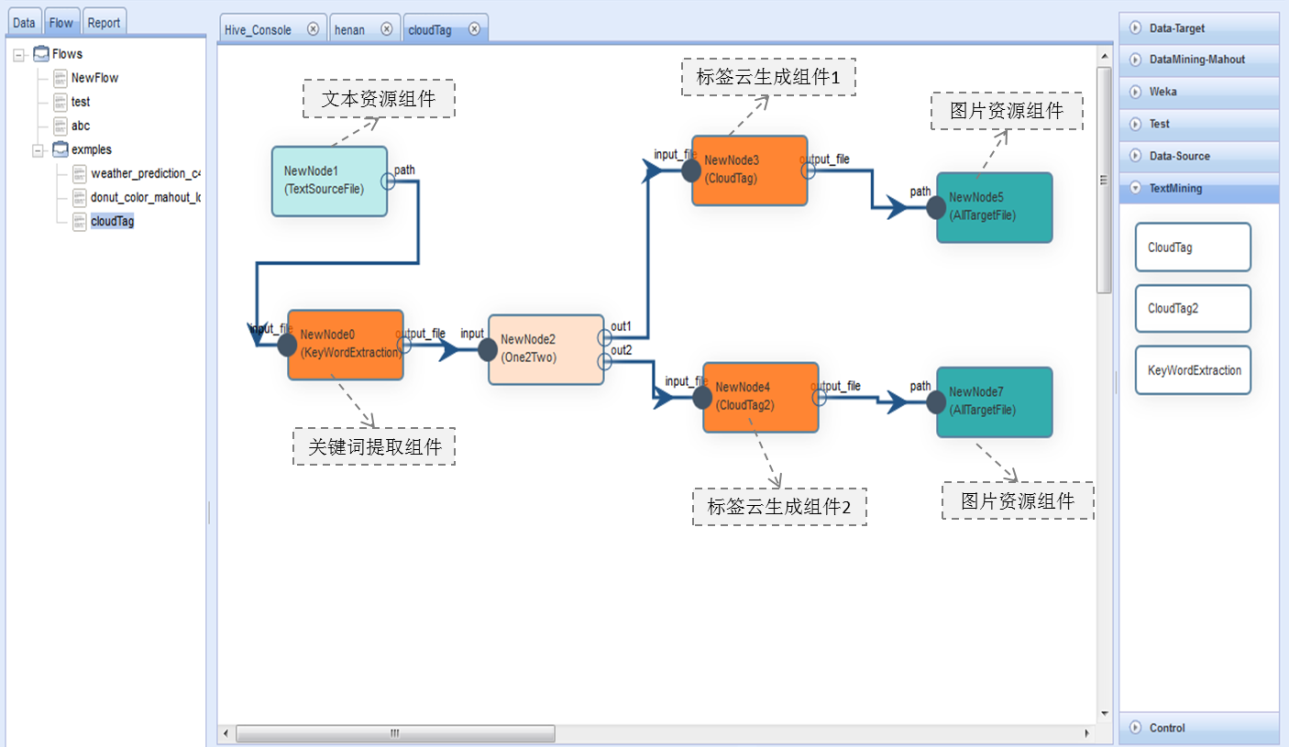


图 4‑24 Haflow数据分析应用开发界面

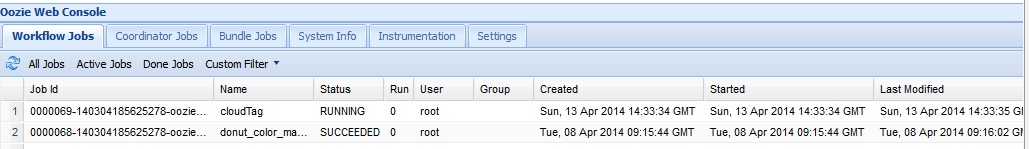


图 4‑25 云标签应用执行状态监控界面

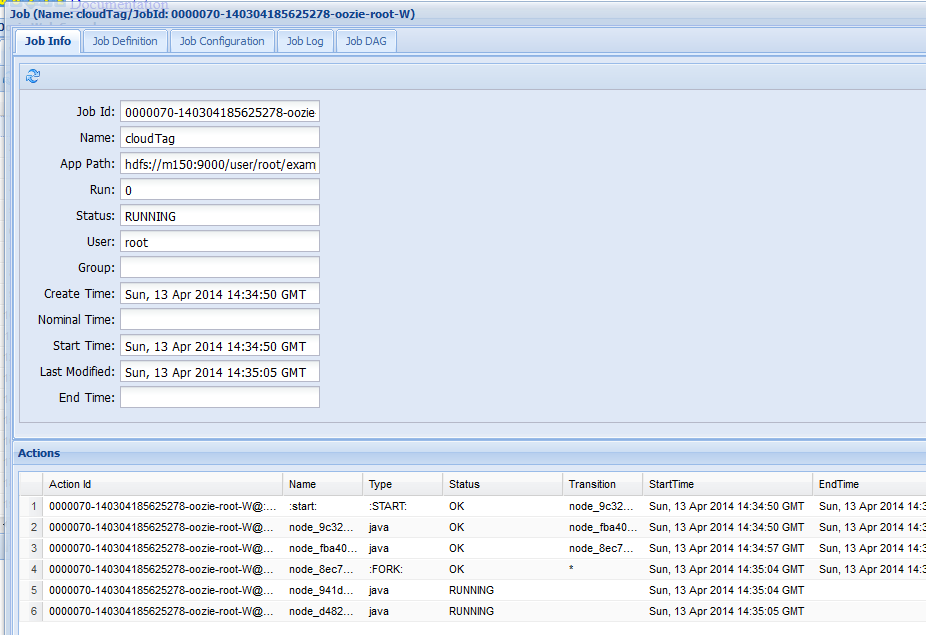


图 4‑26 云标签应用子任务执行状态监控界面

图 4‑25展示的是整个流程的监控界面，可以看到流程的Job Id，流程的名称，目前的执行状态（RUNNING/SUCCEEDED/KILLED）等信息。

图 4‑26展示的是一个流程中的各个节点的执行状态（OK/RUNNING/KILLED），图中显示目前两个节点正在并发执行。

图 4‑27展示的是流程中各个节点执行的状态图，白色的节点为开始节点，绿色的节点为执行成功的节点，黄色的节点为正在执行的节点，灰色的节点为等待执行的节点，黑色的节点为终止节点。

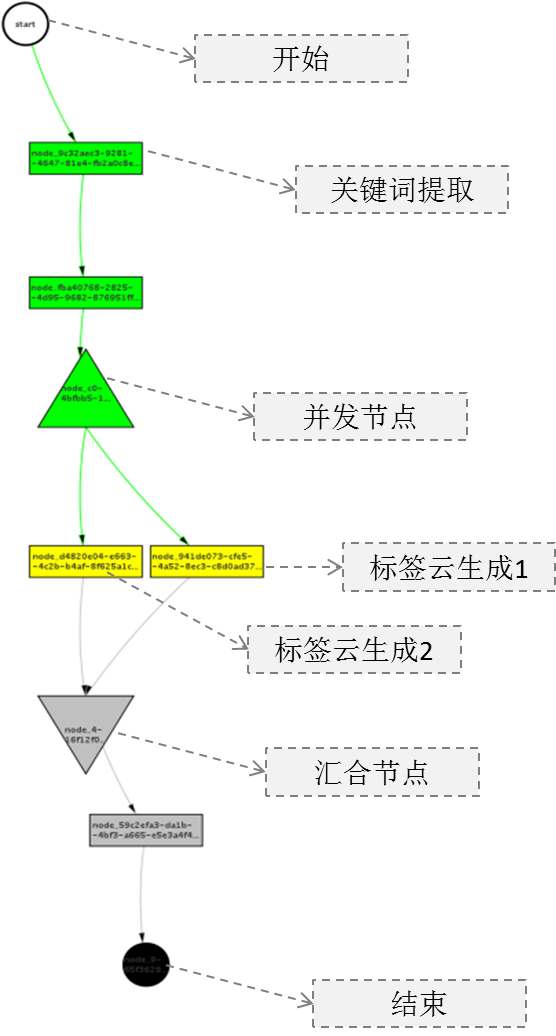


图 4‑27 云标签应用执行状态图

1. 流程执行成功后，打开Haflow的资源管理界面，查看生成的云标签图。

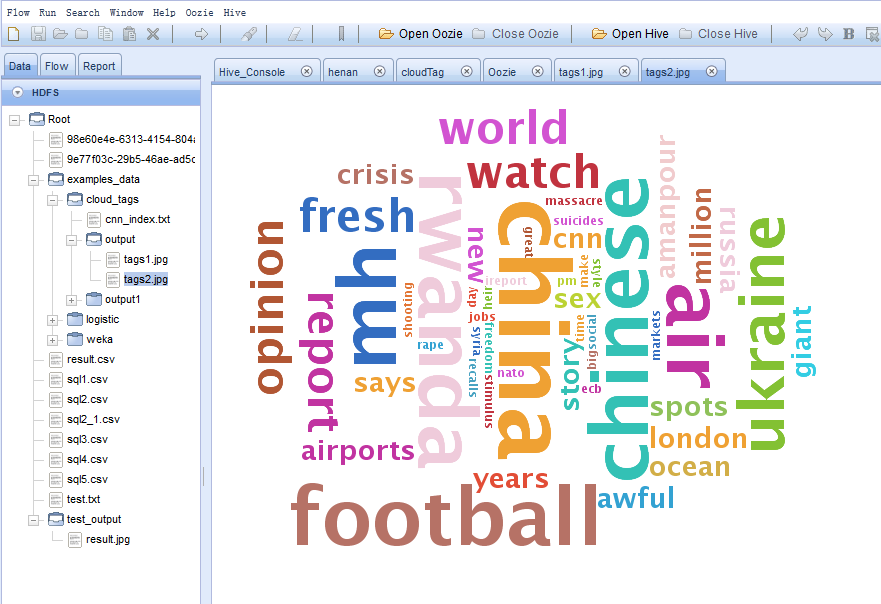


图 4‑28 云标签应用生成的结果图片

# 结束语

本文对全文工作进行了总结，列举了论文工作的主要贡献，并且对接下来的进一步的研究提出了展望。

## 论文贡献

本文主要研究了基于Hadoop的大数据分析应用开发平台的设计与实现，主要贡献包括：

1. 本文对大数据分析技术以及数据分析工具的现状进行了总结，概括了大数据场景下数据分析平台的特殊需求，提出了基于Hadoop实现大数据分析应用开发平台——Haflow的整体架构。
2. 提出了基于组件以及模型驱动的数据分析业务流程处理框架。定义了数据分析组件模型，提出了基于Java注解的组件描述方法以及基于继承的可扩展组件实现方法。在数据分析组件的基础上，本文定义了数据分析业务流程模型，提出了数据分析业务流程处理框架，将数据分析人员从算法调度、执行控制等工作中抽离出来。
3. 设计并实现了基于嵌套图识别与分层的模型转换算法。该算法可以根据数据分析流程模型的依赖关系，生成基于Oozie的并发执行模型。嵌套图识别算法用来判断数据分析流程是否满足Oozie的并发语义，如果满足则数据分析流程可以翻译成最优的执行流程，否则将采用基于分层并发的转换算法。该算法解决了数据分析流程模型到Oozie并发执行模型的高效转换，并使得满足Oozie并发语义的数据分析流程达到了最高的并发程度。
4. 提出了基于HDFS的异构中间数据管理方案，中间数据的自动存储策略以及中间数据的自动匹配验证，达到了中间数据的自动化管理。
5. 最后论文介绍基于Hadoop的大数据分析应用开发平台——Haflow的设计与实现。Haflow充分利用并实现了上述的研究成果，并通过河南省新农合数据分析应用和标签云生成应用验证了系统的功能。

## 未来工作展望

本文完成了基于Hadoop的大数据分析应用开发平台的设计和实现，研究了其中的关键技术，实现了系统的基本功能。本文未来的工作包括：

* + 模型转换算法的进一步优化

在Haflow中，基于嵌套图识别与分层的模型转换算法可以实现所有数据分析业务模型到基于Oozie的并发执行模型的转换，并且对于满足Oozie并发语义的数据分析业务模型该算法完成了最优转换。但是对于不满足Oozie并发语义的数据分析业务模型，目前只能达到分层的并发效果，这种并发策略并不一定是最优的，执行时间取决于每层的最耗时节点，对于这类型的模型需要进一步的优化算法。

* + 用户工作空间的安全隔离

目前Haflow实现了用户工作空间的概念，可以在一定程度上对用户工作空间进行隔离，如：用户只能通过Haflow查看自己目录下的文件，只能操作自己空间下的数据分析应用以及报表。但是对于深层次的用户权限问题还需要进行深入地分析，例如Hive数据仓库的数据隔离，执行程序访问权限控制等。

* + 数据分析应用程序的调试

目前Haflow调试数据分析应用程序，需要查看Hadoop的执行日志进行调试，接下来的工作重点之一将研究如何设计并实现数据分析应用程序执行控制和调试界面。

参考文献

* 1. Hadoop. <http://hadoop.apache.org/>
  2. Thusoo A, Sarma J S, Jain N, et al. Hive: a warehousing solution over a map-reduce framework [J]. Proceedings of the VLDB Endowment, 2009, 2(2): 1626-1629.
  3. Islam M, Huang A K, Battisha M, et al. Oozie: towards a scalable workflow management system for hadoop[C]//Proceedings of the 1st ACM SIGMOD Workshop on Scalable Workflow Execution Engines and Technologies. ACM, 2012: 4.
  4. Sqoop. <http://sqoop.apache.org/>
  5. Mahout. <https://mahout.apache.org/>
  6. Mayer-Schönberger V, Cukier K. Big data: A revolution that will transform how we live, work, and think[M]. Houghton Mifflin Harcourt, 2013.
  7. Storm. http://storm.incubator.apache.org/
  8. Agrawal D, Bernstein P, Bertino E, et al. Challenges and Opportunities with Big Data 2011-1[J]. 2011.
  9. World Health Organization. Influenza fact sheet. <http://www.who.int/mediacentre/factsheets/fs211/en/>
  10. Weka. <http://www.cs.waikato.ac.nz/ml/weka/>
  11. Clementine. <http://spss-clementine.software.informer.com/>
  12. RHadoop. <https://github.com/RevolutionAnalytics/RHadoop/wiki>
  13. R. <http://www.r-project.org/>
  14. KXEN Analytic Framework. <http://www.kxen.com/>
  15. Oozie docs. [http://oozie.apache.org/](http://oozie.apache.org/docs/4.0.1/WorkflowFunctionalSpec.html#a3.1.5_Fork_and_Join_Control_Nodes)
  16. 纪俊. 一种基于云计算的数据挖掘平台架构设计与实现 [D][D]. 青岛: 青岛大学, 2009.
  17. 余永红, 向晓军, 高阳, 等. 面向服务的云数据挖掘引擎的研究[J]. 计算机科学与探索, 2012, 6(1): 46-57.
  18. 钱肖鲁，朱建秋，朱扬勇.DMVisualMiner：一个可视化数据挖掘分析平台[J].计算机工程. 2003, 29:148-150.
  19. 丁岩，杨庆平，钱煜明.基于云计算的数据挖掘平台架构及其关键技术研究[J].中兴通讯技术.2013,19(1):53-60.
  20. Malewicz G, Austern M H, Bik A J C, et al. Pregel: a system for large-scale graph processing[C]//Proceedings of the 2010 ACM SIGMOD International Conference on Management of data. ACM, 2010: 135-146.
  21. Isard M, Budiu M, Yu Y, et al. Dryad: distributed data-parallel programs from sequential building blocks[J]. ACM SIGOPS Operating Systems Review, 2007, 41(3): 59-72.
  22. Wei Zhao, Jie Liu, Dan Ye, and Jun Wei. 2013. Mining user daily behavior patterns from access logs of massive software and websites. In Proceedings of the 5th Asia-Pacific Symposium on Internetware (Internetware '13)
  23. Ghemawat S, Gobioff H, Leung S T. The Google file system[C]//ACM SIGOPS Operating Systems Review. ACM, 2003, 37(5): 29-43.
  24. Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters[J]. Communications of the ACM, 2008, 51(1): 107-113.
  25. Vavilapalli V K, Murthy A C, Douglas C, et al. Apache hadoop yarn: Yet another resource negotiator[C]//Proceedings of the 4th annual Symposium on Cloud Computing. ACM, 2013: 5.
  26. Pig. <https://pig.apache.org/>
  27. Chang F, Dean J, Ghemawat S, et al. Bigtable: A distributed storage system for structured data[J]. ACM Transactions on Computer Systems (TOCS), 2008, 26(2): 4.
  28. Manyika J, Chui M, Brown B, et al. Big data: The next frontier for innovation, competition, and productivity[J]. 2011.
  29. Grobelnik, Marko. Big Data Tutorial [EB/OL]. [2012-10-02]. http://videolectures.net/eswc2012\_grobelnik\_big\_data/
  30. Hamish Barwick . The 'four Vs' of Big Data. Implementing Information Infrastructure Symposium[EB/OL]. [2012-10-02]. <http://www.computerworld.com.au/article/396198/iiis_four_vs_big_data/>

发表文章

[1] 赵薇、刘杰、叶丹、魏俊。大数据分析服务平台。计算机科学，已录用

[2] Wei Zhao, Jie Liu, Dan Ye, and Jun Wei. 2013. Mining user daily behavior patterns from access logs of massive software and websites. In Proceedings of the 5th Asia-Pacific Symposium on Internetware (Internetware '13). ACM, New York, NY, USA, Article 18, 4 pages. DOI=10.1145/2532443.2532462 <http://doi.acm.org/10.1145/2532443.2532462>

致谢

三年的硕士生涯是我人生中非常宝贵的一点时光。在这期间，有许多老师和同学在各方面给我很大的帮助。藉此论文完成之际，我谨向合作过的和帮助过我的人致以最诚挚的感谢。

首先，要衷心感谢我的导师叶丹老师！研究生期间，叶老师为我提供了优越的学习和研究环境， 并提供了大量的实践机会。在学习上，叶老师一直都给予我莫大的指导和帮助；在生活中，叶老师也不断给予我亲切的关怀与鼓励。 对于我的论文，叶老师从选题、研究到写作等方面都给予了悉心的指导和支持。叶老师平易近人的性格和严谨求实的科研精神令我感激至深，受益匪浅。

衷心感谢中心的黄涛、钟华、金蓓弘、魏峻、许舒人、张文博、杨燕、徐罡老师对我的关心和指导，感谢各位老师为我创造了良好的学习与科研环境， 他们对科学的热忱和敬业态度使我深受感染。

感谢刘杰师兄，在三年的时间里，刘杰师兄在学习、科研和生活等方面都给予了我很大的指导和帮助，在我茫然时给予我鼓励和点拨。

感谢 DI 小组的刘杰，许利杰，伍海江，朱锋，孙耀，张晓杰，李奇原，肖彦等师兄师姐们，是你们在我刚进实验室的时候提供了无私的帮助和关怀，并在我接下来的研究生生涯中提供了莫大的支持和帮助。感谢DI组的吕瀛同学，主动承担组里的各项任务，经常耐心给我讲解各项技术。感谢DI组的陈茜师妹和柯叶青师妹，感谢你们在Haflow中帮我完成了很多的功能，陪我一起参加原型大赛。感谢INDG组的唐震，感谢你在Haflow成长初期提出的宝贵意见，以及完成了很多功能。

感谢吕瀛，付冉，高嘉泽，邓换方，杨宇威，高强，杨鑫昇，雷栋，吴添勇，丁亮，江浩亮，姚思佳，闫东旭，王彦士，刘朝晖 等同学，研究生期间，我们一起学习、相互交流、彼此帮助、 共同进步， 度过了非常愉快而又难忘的三年时光， 共同走过了在软件所求学的宝贵历程。

特别要感谢我的父母，我的妹妹，我的朋友们，他们给了我无尽的支持和前进的动力。

最后，向所有关心和帮助过我的人致以深深的谢意。