

# 目 录

|                         |    |
|-------------------------|----|
| 前 言 .....               | 1  |
| 第一章 绪 论 .....           | 2  |
| 1.1 研究背景及意义 .....       | 2  |
| 1.2 主要功能概述 .....        | 2  |
| 1.3 论文主要工作 .....        | 3  |
| 1.4 本文组织结构 .....        | 3  |
| 第二章 需求分析与可行性分析 .....    | 5  |
| 2.1 需求分析 .....          | 5  |
| 2.1.1 功能需求分析 .....      | 5  |
| 2.1.2 性能需求分析 .....      | 5  |
| 2.1.3 安全性和可用性需求分析 ..... | 6  |
| 2.2 可行性分析 .....         | 6  |
| 2.2.1 经济可行性 .....       | 7  |
| 2.2.2 技术可行性 .....       | 7  |
| 2.2.3 时间可行性 .....       | 8  |
| 2.2.4 法律可行性 .....       | 8  |
| 第三章 相关技术简介 .....        | 10 |
| 3.1 网络爬虫 .....          | 10 |
| 3.2 Python .....        | 10 |
| 3.3 Docker .....        | 11 |
| 3.4 Redis .....         | 11 |
| 3.5 MySQL .....         | 11 |
| 3.6 HDFS .....          | 12 |
| 3.7 MapReduce .....     | 13 |
| 3.8 Node.js .....       | 13 |
| 3.9 Echarts .....       | 14 |
| 第四章 系统设计 .....          | 15 |
| 4.1 数据库设计 .....         | 15 |

|       |                      |    |
|-------|----------------------|----|
| 4.1.1 | 学者表 .....            | 15 |
| 4.1.2 | 研究领域表 .....          | 16 |
| 4.1.3 | 学者领域表 .....          | 16 |
| 4.1.4 | 中英文对照表 .....         | 16 |
| 4.1.5 | 相似度表 .....           | 17 |
| 4.1.6 | 学者 h 指数表 .....       | 17 |
| 4.1.7 | 领域人数表 .....          | 17 |
| 4.1.8 | 领域权重表 .....          | 18 |
| 4.1.9 | 物理模型图 .....          | 18 |
| 4.2   | 分布式爬虫设计 .....        | 19 |
| 4.2.1 | 设计要点 .....           | 19 |
| 4.2.2 | 分布式架构 .....          | 19 |
| 4.2.3 | ip 代理池 .....         | 20 |
| 4.2.4 | 部署环境 .....           | 21 |
| 4.3   | MapReduce 计算设计 ..... | 22 |
| 4.4   | Web 前后端设计 .....      | 22 |
| 第五章   | 系统实现 .....           | 24 |
| 5.1   | Hadoop 集群搭建概述 .....  | 24 |
| 5.2   | ip 代理池实现 .....       | 24 |
| 5.2.1 | 抓取免费代理 .....         | 24 |
| 5.2.2 | 维护代理池 .....          | 25 |
| 5.2.3 | 获取可用代理 .....         | 26 |
| 5.2.4 | 调度模块 .....           | 26 |
| 5.3   | 数据爬取 .....           | 26 |
| 5.4   | Docker 部署 .....      | 28 |
| 5.5   | Redis 并发控制 .....     | 29 |
| 5.6   | MapReduce 计算实现 ..... | 29 |
| 5.6.1 | h 指数排名 .....         | 29 |
| 5.6.2 | 研究领域人数统计 .....       | 30 |
| 5.6.3 | 研究领域比重统计 .....       | 30 |

|                    |    |
|--------------------|----|
| 5.7 Web 平台开发 ..... | 30 |
| 5.7.1 后端实现 .....   | 31 |
| 5.7.2 前端实现 .....   | 31 |
| 5.8 Web 平台展示 ..... | 31 |
| 5.8.1 数据分析 .....   | 31 |
| 5.8.2 学者信息展示 ..... | 33 |
| 5.8.3 学者详情页面 ..... | 33 |
| 5.8.4 国际化 .....    | 34 |
| 第六章 总结与展望 .....    | 35 |
| 6.1 本文总结 .....     | 35 |
| 6.2 后续展望 .....     | 35 |
| 参考文献 .....         | 37 |
| 致    谢 .....       | 38 |

## 摘 要

高校学者是全球科研力量的最重要组成部分，他们的科研成果推动着人类社会的进步。近年来，新兴领域和学科大量涌现，科技界呈现出百花齐放的繁荣景象，高校学者的科研方向也日趋多元化和精细化。同时，移动互联网的发展极大地拓宽了人们获取信息的渠道。人们可以在任何时间、任何地点乃至移动过程中都能方便地从互联网上查询相关学者的信息以及科研领域的最新资讯。普通用户查询的信息较为集中和单一，如何获取海量的高校学者信息以及了解当下有哪些热门的科研领域成为普通用户的一大痛点。

为解决上述痛点，本次毕业设计分为三大流程。首先，设计一个分布式爬虫从学术网站上爬取高校学者信息。其次，使用 Hadoop 大数据框架的 HDFS 对学者信息进行分布式存储；使用 Hadoop 大数据框架的 MapReduce 编程模型对学者信息进行分布式计算。最后，借助于 web 技术对计算结果进行可视化展示。

本次毕设的最终结果是以网站的形式展示给用户。用户只需在浏览器中输入网址即可访问该网站。通过该网站，用户可以了解到根据 h 指数排名的所有学者信息以及各科研领域的热门程度。另外，该网站有着良好的人机交互，如用户可以对学者所有的信息字段进行模糊查询、对不感兴趣的表格列隐藏、将所有学者信息导出成 excel 文件以便离线查看、点击学者信息可进入学者详情页面并且该页面还推荐了其他相似学者等等。

**关键词：**分布式爬虫；Hadoop；web；可视化

## Abstract

College scholars are the most important part of the global research force and their scientific output promotes the progress of human society. In recent years, a large number of emerging areas and disciplines have emerged, and the scientific and technological community has presented a flourishing scene of prosperity. At the same time, the development of mobile internet has greatly expanded the channels for people to obtain information. It is easy for people to search for scholars' information and the latest news about scientific research fields at any time from any place, even you're moving. The information searched by common users is relatively concentrated and single. How to obtain massive data about scholars and have a knowledge of scientific research fields which are temporarily popular are a common pain point for ordinary users.

In order to resolve the above pain point, the graduation design is divided into three major processes. Firstly, a distributed crawler is designed to crawl college scholars' information from academic websites. Secondly, HDFS of Hadoop big data framework is used to realize distributed storage of crawled information and MapReduce of Hadoop big data framework is used to realize distributed computing of these data. Finally, the results of computation are visually displayed by means of web technology.

The final result of this graduation design is presented to the common users in the form of a website. To access the website, users just need to enter the url in their browser. Through the site, users are able to find out all the scholars' information sorted by h-index and the popularity of each scientific research field. Besides these, the website has a good human-computer interaction. For example, users can perform fuzzy query on all fields of scholars' information, hide table columns which they are not interested in, export all data to Microsoft Excel for offline viewing, click on some table cell to enter the detail page about this scholar and the detail page will recommend some scholars who are similar to this scholar and so on.

**Keywords:** Distributed crawler; Hadoop; Web; Visualization

## 前 言

在这个大数据时代，不论是工程领域还是研究领域，数据已经成为必不可少的一部分[1]。人工智能浪潮的兴起更是离不开海量数据的支撑[2]。而商业公司垄断着数据，很少会将其数据公之于众。网络爬虫是一种从网页中提取信息的技术。对于第三方人员而言，海量数据的获取很大程度上依赖于爬虫的爬取。因此，爬虫技术逐渐变得火爆起来。

另一方面，大数据的存储与计算对传统的基础架构提出了新的挑战。移动设备的普及和互联网的快速发展，数据量呈现爆发式增长，硬件的发展速度远远滞后于数据的发展速度。单机设备很多时候无法处理 TB、PB 级别的数据[3]。如果一头牛拉不动货物，显然多找几头牛一起拉会比培育一头更强壮的牛容易。同理，单纯地给某台机器增加存储容量和提升硬件性能无法从根本上解决大数据带来的挑战。综合利用多个普通机器要比打造一台超级计算机更加可行，这就是大数据处理技术的设计思想。

web 系统具有开放性强、结构扩展性好、维护升级便捷、用户界面一致且与用户交互的特点，作为互联网信息的载体，其开发方式在过去的二十年中发生了天翻地覆的变化。从早期的前后端不分，到 ajax 的出现促成 MVC 开发模式的流行，再到现在火热的 MVVM 使得前后端分离成为了现实。每一次的演变对工程开发效率和用户体验都是极大的提升。

本次毕设用到的技术较多，对个人的编程能力提出了很高的要求。涉及到的知识点有分布式系统架构、计算机网络协议、爬虫原理、redis 并发读写、Linux 系统常用命令、docker 指令、git 版本管理、web 前端、数据库基本操作、国际化、python、js 和 java 编程语言等等。

整个工程是面向高校学者信息大数据平台的分布式爬虫设计与实现，开发流程分为如下步骤：

- (1) 搭建完全分布式的 Hadoop 集群。
- (2) 根据爬取需求对分布式爬虫进行设计。
- (3) 准备爬虫条件并实现分布式爬虫。
- (4) 利用大数据技术对爬取数据进行分布式存储和计算。
- (5) 编写 web 系统对计算结果进行可视化展示，另外对学者信息提供搜索、导出、推荐等功能。

## 第一章 绪 论

本章首先介绍了高校学者信息大数据平台的研究背景及意义，然后介绍此平台具有哪些功能以及如何去使用这些功能，紧接着介绍了开发此平台所做的主要工作，在本章的最后一节概括论文的组织结构。

### 1.1 研究背景及意义

每所高校都有自己的门户网站，学者的介绍信息按照院部归类在一起。这些介绍信息较为简短，很多学者选择建立自己的个人主页。有些主页是可以公网访问的，有些主页必须登录校园网才能访问。当用户想要获取某位学者的信息时，他们会首先通过搜索引擎来查找。搜索引擎提供的信息有限，要想更多地了解这位学者，用户就必须访问学者的个人主页，部分主页还得通过 vpn 来访问。这样的繁琐操作极大地消耗了用户耐心和降低了用户体验。

与此同时，随着科技的飞速发展，新兴领域和学科大量涌现，科技界呈现出百花齐放的繁荣景象，高校学者的科研方向也日趋多元化和精细化。如何获取海量的高校学者信息以及了解当下有哪些热门的科研领域成为普通用户的一大痛点。

很多公司通过其先进的技术构建起了大规模的学术搜索系统，比较出名的有 Google Scholar、Libra 以及百度学术。对于普通用户来说，这些学术网站大而全，专业性强，操作界面较为复杂，很难上手。用户其实只需知道学者论文数、被引用数、h 指数等等这些基本信息。另外，这些网站虽然会显示热门的研究方向，但缺少对研究方向的量化统计，对用户是一个非常抽象的体验。

本次研究着眼于普通用户搜索学者信息以及了解科研领域发展情况的痛点，为搭建高校学者信息大数据平台奠定了需求基础。通过该平台，普通用户可以查询到学者信息，了解时下热门的研究方向。如果有想从事科研方向的人员，该平台的统计数据具有一定的参考价值。对于本人而言，通过这次研究提升了信息搜集的能力，掌握了学术网站的使用方法，确定了完成整个工程的技术路线。

### 1.2 主要功能概述

本工程是根据爬虫爬取的数据来搭建学者信息大数据平台，主要功能如下：

- (1) 以表格的形式展示学者基本信息，这些信息包括姓名、h 指数、g 指数、论文数、被引用次数、群集度、多样性、活跃度等。
- (2) 以柱状图的形式展示各研究领域的人数。
- (3) 以饼状图的形式展示各研究领域所占比重。
- (4) 提供国际化功能，支持 web 页面中英文切换。
- (5) 支持学者信息模糊检索，信息字段列隐藏。
- (6) 支持学者信息的 excel 导出。
- (7) 提供学者详情页面，并为用户提供相似学者推荐。

### 1.3 论文主要工作

基于上述功能，以下是我进行该平台设计的主要工作：

- (1) 大数据平台搭建：无论是大数据的存储计算，还是分布式的爬虫，都对机器数量有一定的要求。这里使用四台 Ubuntu 服务器在同一个内网段下搭建 Hadoop 集群。
- (2) 数据获取：设计和实现分布式爬虫，利用该爬虫爬取学者信息，除了爬取姓名、论文数、h 指数等基本信息，还要爬取该学者各研究方向的权重。
- (3) 数据处理：利用 Hadoop 的编程模型 MapReduce 对数据进行计算。包括根据 h 指数对学者排名、统计各领域研究人数以及各领域所占比重等。
- (4) 数据展示：通过 web 技术对处理后的数据进行可视化展示。
- (5) web 界面交互实现：提供图表自适应、excel 导出、模糊查询、详情介绍、相似学者推荐功能。

### 1.4 本文组织结构

本篇论文总共有六个章节，按照整个项目的开发流程循序渐进。每一章的侧重点不同，具体内容分别如下：

- (1) 第一章是绪论部分，依次介绍高校学者大数据平台的研究背景、研究意义、功能概述和实现这些功能的主要工作。
- (2) 第二章是对平台的需求分析，并从不同的点出发，阐述开发该平台的可行性。



- (3) 第三章是开发此平台所用的一些技术，做了一个系统的整理。
- (4) 第四章是针对平台的各项需求进行总体设计，主要包括分布式爬虫架构的设计、数据库的表格设计、MapReduce 计算的设计以及 web 系统的设计。
- (5) 第五章是在进行整体设计后，进入各模块的编码实现阶段，并在遇到的实际问题中给出解决方案。
- (6) 第六章为总结和展望，总结开发这个平台所做的主要工作，并对该平台的发展进行展望。

## 第二章 需求分析与可行性分析

本章主要对高校学者信息大数据平台进行需求分析，然后在明确需求的基础上，从各方面分析开发此平台的可行性。

### 2.1 需求分析

#### 2.1.1 功能需求分析

一般的学术网站大而全，专业性强，操作界面较为复杂，很难上手。对于普通用户而言，他们只想了解学者的基本信息以及当下有哪些热门的研究领域。基于平台简洁、交互友好的设计目标，功能需求如下所示：

- （1）以表格的形式展示学者基本信息，这些信息包括姓名、h 指数、g 指数、论文数、被引用次数、群集度、多样性、活跃度等。
- （2）通过柱状图的形式展示科研人数前 10 名的研究领域，每个领域标注相应人数。
- （3）通过饼状图的形式展示科研比重前 10 名的研究领域，每个领域标注相应比重。
- （4）web 页面支持国际化，提供中英文切换功能。
- （5）当用户不知道学者的具体信息，或者用户想要查看某一类学者时，需要提供模糊检索的功能。
- （6）当用户使用的设备较小，或者用户对学者的某些信息不感兴趣时，需要提供信息字段列隐藏的功能。
- （7）当用户设备未联网，或者用户想要保存这些学者的信息时，需要提供 excel 导出的功能，这样方便用户离线查看。
- （8）点击某行学者信息，提供学者详情页面，并为用户推荐相似的学者。

#### 2.1.2 性能需求分析

开发这样一个平台，不管是数据的获取、数据的计算、网页的加载，还是数据的检索，性能都是要着重考虑的因素。

在数据获取方面，尽管爬虫是异步加多线程的，但是只能在一台服务器上运行，所以爬取效率是有限的。如果多台主机协同爬取，共同完成一个爬取任务，那么爬取效率必然会成倍增长[1]。这也是本次毕设爬虫设计为分布式的原因。

在数据处理方面，海量数据的处理如果采用集中式计算，那么需要耗费相当长的时间来完成。MapReduce 基本原理是通过 Map 任务，将一个大任务分为多个小任务，关键是这些小任务是同时运行的。再通过 Reduce 任务，把多个小任务的结果汇总起来，最后输出汇总后的结果[3]。MapReduce 工程队的工作过程与分布式计算过程是基本对应的。

在网页加载方面，如果用户设备网络状况良好，那么用户在输入链接后，网页应该在 1~4s 内全部加载完成。如果加载时间超过 8s，那么用户会认为网站技术很糟糕，用户的耐心会遭到极大的打击。如果加载时间更长，那么用户会认为网站服务器发生了故障，就会关闭网站标签页，甚至以后不会再去访问，将其纳入网站访问黑名单[4]。网页加载优化是一个亟需重视的问题。

在数据检索方面，由于用户查找的时候需要对所有的学者信息进行全字段模糊匹配，并且该计算过程是实时的，如果采用单进程计算，那么将会耗费相当长的时间来完成。用户在长时间内看到检索结果，会误认为服务器宕机或者与网页断开连接，这极大地降低了用户体验。

### 2.1.3 安全性和可用性需求分析

在安全性上，本平台没有登录机制，因此任何用户都可以访问该网站，都可以在网页上操作。由于学者的信息是从公开的学术网站上爬取的，不存在利用网站系统的漏洞来窃取学者信息。因此如果有第三方使用中间技术抓取后台向前台发送的数据，这也没有关系。

在可用性上，要考虑到该平台是否满足了普通用户的基本需求。在功能齐全的基础上，还要考虑操作是否方便，界面布局是否合理。当服务器端出现异常时，要能够在前端页面给用户人性化的提示[5]。

## 2.2 可行性分析

基于上述需求，本节围绕经济可行性、技术可行性、时间可行性和法律可行性展

开分析。

### 2.2.1 经济可行性

经济可行性指的是开发此平台所需要的资金，这里从硬件资金和软件资金两个方面进行分析。

在硬件上，搭建 Hadoop 集群需要四台 Ubuntu 服务器。由于本人只有一台笔记本电脑，肯定无法满足搭建要求。由于实验室闲置的机器很多，可以把它们拿来搭建集群。四台机器连接实验室的交互机即可访问互联网。本人的笔记本电脑用于编写代码。

在软件上，分布式爬虫需要一些公网 ip 做代理，而公网 ip 是比较昂贵的，对学生经济负担大。我们从第三方网站上爬取一些公网 ip，尽管这些 ip 质量比较差，但可以通过算法对 ip 进行可用性维护，某个 ip 不可用的话，就切换到另一个 ip。开发所使用的技术如 python、Hadoop、nodejs、echarts 等都是免费开源的，可以直接拿来使用。开发工具如 VSCode 是免费的，IDEA、PyCharm 和 DataGrip 虽是商用版，但可以用苏大学生邮箱申请免费使用。网站是需要部署到服务器上的，一般的服务器是一百多块钱一个月，使用学生帐号可以便宜到十块钱一个月，这一点资金对学生是微不足道的。

综上所述，开发此平台在硬件资金和软件资金都是可行的。

### 2.2.2 技术可行性

爬虫脚本是用 python 编写的，python 简洁优雅，最重要的是第三方库非常丰富。例如发送网络请求的 requests 库、解析网页结构的 BeautifulSoup、redis 连接库等，这些库都极大地方便了爬虫的编写。借助于 docker 技术，将爬虫打包进容器，从而实现分布式爬虫。

对数据的处理是使用 MapReduce 计算的。当运行一个 MapReduce 任务时，Map 端会读取 HDFS 上的文件块，将文件块的每一行记录处理成键值对<key, value>的形式发送到 Reduce 端。Reduce 端接收 shuffle 或 sort 之后的键值对数据，将键名相同的值集合进行汇总得到新的键值对。MapReduce 计算完成的结果会输出到 HDFS 中 [3]。借助于 HDFS，可实现分布式存储；借助于 MapReduce，可实现分布式计算。

在 web 系统开发上，前端采用的是 html、css 和 js。html 用于构建基本页面，并且可以创建交互式表单。css 用于设定网页布局、设定页面元素样式、设定适用于所有网页的全局样式。js 用于操作 dom 元素，并且可以发送网络请求与后台完成数据交互。后端采用的是 nodejs 和 mysql。nodejs 是运行在服务器端的 js。使用 nodejs 时，我们不仅在实现一个应用，同时还实现了整个 http 服务器，这样的话我们就不需要再另外配置 Apache、Tomcat 这种 http 服务器了。nodejs 使用的是事件驱动、非阻塞 I/O 模型，这让它可以处理大量的并发请求。mysql 体积小，速度快。并且由于其开源的本质，生态活跃，解决方案多，可以极大地帮助我们提高数据库开发效率。

综上所述，开发此平台在技术上是可行的。

### 2.2.3 时间可行性

本次开发时间还是比较充足的，大四这一学年没有课程，可以专心做毕设。

由于之前开发过很多项目，因此对于上述技术还是很熟悉的，上手很快。需要集中花时间学习的就是 hadoop 技术和分布式爬虫原理。大数据框架的搭建会花费我很多时间，毕竟是在四台服务器上协同完成，不管是固定 ip、内网通信还是时间同步等等，都会遇到很多的问题。每改动一处，其它三台机器都要改动，相当费时费力。

爬虫在爬取网页的时间，会出现各种异常，比如连接超时、ip 被封等，这也需要花费时间去进行异常处理。

大概有长达四个月的时间供我去学习和解决上述遇到的问题，因此开发此平台在时间上是可行的。

### 2.2.4 法律可行性

学者信息在学术网站上都是公开的，任何人都可以访问。因此不存在利用网络技术窃取学者信息的违法行为。

开发此平台所用到的技术如 python、Hadoop、mysql、echarts 等都是免费开源的，可以直接拿来使用。

开发此平台所用到的工具如 IDEA、Pycharm、DataGrip 是使用学生帐号申请的，符合 JetBrains 公司的法律规定。

编写的代码有些是参考书籍里的，但书籍作者已经其代码开源到 github 上，我们

可以 clone 下来参考。

综上所述，开发此平台在法律上是可行的。

## 第三章 相关技术简介

本章主要介绍高校学者大数据平台数据获取、数据处理和可视化展示所使用的知识和技术。

### 3.1 网络爬虫

网络爬虫本质上是一个程序脚本，该脚本自动地从互联网上抓取信息。抓取的方式一般有两种，一种是抓取页面，将网页转换成文档型字符串，然后利用正则匹配、标签匹配等规则提取有效信息，但是这种方式需要等待后台服务器返回整个 html 页面，效率较低；另一种方式是抓取接口，网站一般都会有 api 接口，尤其是样式统一、排列规律的网页结构，通过抓取 api 接口可以直接从后台获得数据，这种抓取方式效率较高，但难度较大，因为一般的大型网站都会对接口进行加密。以上两种方式都是向后台发送 http 请求，只不过后台返回的数据格式不一致而已。

在信息爆炸时代，互联网巨头垄断着数据，要想获取海量数据，很大程度上依赖于爬虫。

### 3.2 Python

Python 是一种动态的解释型脚本语言。python 虽然运行效率不高，但是开发效率极高。在软件开发领域中，java 一直是霸主，但是对于开发者而言，java 较难上手，jvm 调优更是复杂无比[6]。但 python 不存在这些问题，许多库帮你做好了这些工作。

Python 语言简洁，语义清晰，符合一般人的思维。很多人选择 python 作为他们的编程入门语言。

Python 拥有胶水语言的特性，因此它的第三方库非常丰富。在软件开发领域中，有 PyQt 库来开发桌面应用程序，有 flask 库开发 web 后台，有 scrapy 开发爬虫，有 scrapy 做服务器运维等等。在科研领域中，有 numpy、pandas 库做数值计算，有 matplotlib 库做数据可视化，有 opencv 库做视觉处理，有 tensorflow、pytorch 库训练模型等等。本系统就用到了 requests、redis、pymysql、flask、beautifulsoup 等这些库。

Python 可移植性强，如果你在 windows 机器上开发好了代码，那么你的 python 程序无需修改就可以放到服务器上运行。一套代码，多端共用。

### 3.3 Docker

Docker 是一个开源的容器引擎，让开发者可以打包他们的应用以及依赖包到一个可移植的容器中，然后发布到任何流行的操作系统上，也可以实现虚拟化。与流行的 iOS、Android 一样，容器使用的是沙箱机制，任何两个镜像之间不会有任何通信接口。一个容器的运行异常也不会影响另一个容器的运行。

通过 docker，我们可以将网络爬虫的 python 脚本封装到容器中。宿主机只需安装 docker 客户端，然后拉取封装了爬虫的容器，这样就可在本机运行爬虫镜像。利用 docker，我们屏蔽了各个宿主机运行环境的差异，为网络爬虫提供了统一的运行环境。

### 3.4 Redis

Redis 是一个基于内存的键值型数据库。与其它的键值型数据库相比，具有如下三大特点：支持数据的持久化，可以将内存中的数据保存在硬盘中，下次启动 redis 时可以从硬盘中加载数据；提供丰富的存储结构，如 list、set、zset 等；支持主从模式的数据备份，这一点跟关系型数据库集群的主从备份机制很相似，但是在 Redis 上实现更简单。

由于 Redis 的数据是存放在内存里的，因此可以将网络爬虫的页码和偏移量存放在 Redis 中，这样就可以快速获取页码和偏移量。在后台开发中，对于实时性要求较高的用户需求，会选择将 Redis 作为后台数据库的二级缓存。另外，Redis 读写速度快，很适合做 session 共享。

### 3.5 MySQL

MySQL 是一个简洁精悍的关系型数据库，提供了多种编程语言的 api 接口。MySQL 使用最标准化的 sql 语言对数据库进行增删查改。MySQL 采用的 C/S 架构，即客户端向服务器发送 sql 语句，服务器执行 sql 语句并返回处理结果。两者是基于 TCP 协议进行通信的。目前 MySQL8.0 已经支持文档存储、事务性数据字典、通用关系表达式等符合现代化需求的功能。

相对于 Oracle 和 Sql Server，MySQL 是一个轻量级的数据库。MySQL 社区版是



开源的，可定制性强，所以 MySQL 受到了 Alibaba、Facebook、Google 等大公司的青睐。例如 Alibaba 基于 MySQL 研制出了适用于本公司业务发展的数据库产品 PolarDB、OceanBase 等。支持 MySQL 的第三方客户端工具也很多，例如 Navicat、DataGrip、Workbench，这些工具界面友好、操作简单，非常适合 DBA 用来数据库开发。

### 3.6 HDFS

HDFS 是一个典型的主从结构，一个主节点或者说是元数据节点（MetadataNode）负责系统命名空间（NameSpace）的管理、客户端文件操作的控制和存储任务的管理分配，多个从节点或者说是数据节点（DataNode）提供真实文件数据的物理支持[7]。系统架构如下图所示：

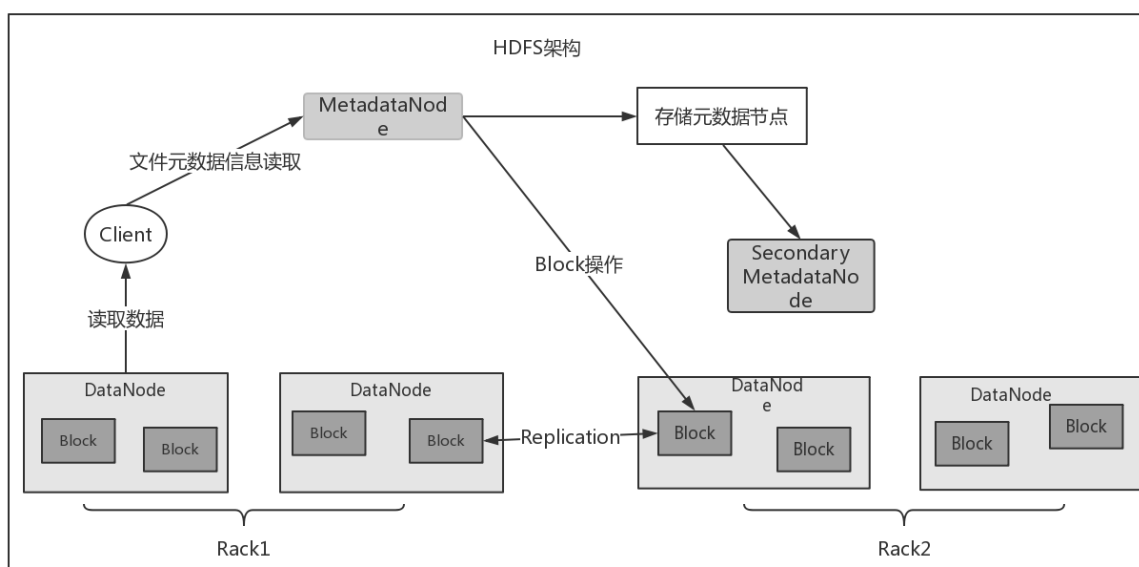


图 3.1 HDFS 架构

客户端可以从宿主机的 datanode 读取数据，也可以通过 metadatanode 读取异机的 datanode 上的数据。每一个 datanode 上的文件元数据信息都是 datanode 定时发送给 metadatanode 的。当这些节点的文件信息发生变化时，datanode 就会将变更的文件信息提交给 metadatanode。metadatanode 对 datanode 的读取操作都是通过这些元数据信息来查找的。如果 metadatanode 发生了故障，那么对于整个集群都是灾难性的。因此这些重要的信息一般都会有备份，而这些备份就存储在次级元数据节点上。写文件操作更需要知道各个节点的元数据信息。当写操作发生时，yarn 会进行资源调度，比

如哪些块是空闲的、哪些块距离最近、哪些块存储空间较大等等，从而选择最优的写动作。由于一份文件有多个备份，当还存在其他支架的时候，除了写入本支架，还会写入到其他支架中，从而保证了数据的高可用性[8]。

### 3.7 MapReduce

MapReduce 是一种编程模型，用于并行计算大规模的数据集。Map 和 Reduce，是它们的主要思想，都是从函数式编程语言里借鉴来的。比如 js 就是一个非常典型的函数式编程语言，在对数组的操作中提供了 map 和 reduce 接口。

MapReduce 封装了分布式并行计算的底层实现细节，因此即使你对分布式不了解，你也可以进行并行编程。很多时候，开发者只需编写相应的 map 函数和 reduce 函数就可以实现分布式计算。map 函数，用来将读取的一行记录映射成一组新的键值对，指定并发的 reduce 函数；reduce 函数，用来将键相同的值集合进行规范，汇总后的结果输出到 HDFS 分布式文件系统中[9]。

### 3.8 Node.js

Node.js 采用谷歌公司开发的 V8 引擎，使用事件驱动、非阻塞和异步 I/O 模型等技术来提高运行性能，可优化应用程序的传输量和规模[10]。

在 node 出现之前，js 是作为前端脚本语言来使用，负责操作 dom 和向后端发送网络请求。浏览器提供了 js 的运行环境，也就是说想要运行 js 脚本，机器上就必须安装一个浏览器。node 为 js 提供了单独的运行环境，使得 js 可以运行在服务器端。而且 node 的语法与前端 js 的语法完全一致，前端工程师可以将触手伸到后端开发中去。想要成为全栈工程师几乎就只需要学会 js 一门语言。

Node 给前端带来了春天，使得前端有了明确的定义。借助于 Node，前后端分离成为现实。在传统的开发模式中，前后端代码是存放在一个项目中，前后端开发界限不清晰。在传统的项目部署中，整个项目是部署在一台服务器上，如果这台服务器宕机，那么整个项目便无法访问。前后端分离后，在开发上，前端和后端程序员各司其职，前后端通过网络来通信，这样就极大地提高了开发效率；在部署上，前后端分别部署在不同的服务器上，即使后端服务器宕机，前端页面仍可访问，这样就提高了用户体验。

### 3.9 Echarts

ECharts 是使用纯 JavaScript 实现的开源可视化库。它提供了非常丰富的图表，如柱状图、饼图、折线图、散点图、雷达图、仪表盘、3D 地球、3D 柱状图、3D 热力图、GL 路线图、GL 散点图、GL 矢量场图等。

另一方面，ECharts 可定制性也非常强，开发者只需传入一个 `renderItem` 函数，就可以从数据映射到任何你想要的图形，这些图形还能和已有的组件结合使用。绝大部分的图表已经能够满足业务需求，自定义图表对开发者的编程能力要求较高。

ECharts 接收的参数有格式要求的。从数据库读取出来的数据都需要进行处理，是由后端完成处理还是发送给前端处理一直是个争论不休的问题。本人的看法是，数据处理应该放在后端完成。一方面，客户端处理能力有限，CPU、内存、硬盘 I/O 都是宝贵资源，如果把资源花在这种无关的计算上，将会极大地影响用户体验。另一方面，数据的处理往往跟业务逻辑相关，业务逻辑不应该在前端中过度体现，而应该封装在后端处理。

ECharts 的图表直观生动，定制性强，交互友好。ECharts 的 api 文档也非常齐全，调用其 api 非常简单。MapReduce 的运算结果就可以用 Echarts 来展示。

## 第四章 系统设计

系统设计是需求分析和系统实现的桥梁。系统设计是基于需求分析的，它描述了实现需求的整体思路，而系统实现则是基于系统设计的编程实践。根据系统的开发流程，系统设计主要包括数据库的表格设计、分布式爬虫的架构设计、MapReduce 计算的设计以及 web 前后端的设计。

### 4.1 数据库设计

#### 4.1.1 学者表

学者的基本信息包括姓名、论文数、h 指数、活跃度等。user\_id 是学者的唯一标识，这样就可以区分同名学者。后续对学者表的增删查改操作都是根据 user\_id 来进行的。具体字段如下所示：

表 4.1 学者信息表 researcher

| 字段          | 类型      | 说明   |
|-------------|---------|------|
| user_id     | int     | 学者编号 |
| name        | varchar | 姓名   |
| paper       | int     | 论文数  |
| citation    | int     | 引用次数 |
| h_index     | int     | h 指数 |
| g_index     | int     | g 指数 |
| sociability | double  | 群集度  |
| diversity   | double  | 多样性  |
| activity    | double  | 活跃度  |

### 4.1.2 研究领域表

研究领域表包括了领域名称。`interest_id` 是研究领域的唯一标识，后续对研究领域表的增删查改操作都是根据 `interest_id` 来进行的。具体字段如下：

表 4.2 研究领域表 `interest`

| 字段                       | 类型                   | 说明   |
|--------------------------|----------------------|------|
| <code>interest_id</code> | <code>int</code>     | 领域编号 |
| <code>name</code>        | <code>varchar</code> | 领域名称 |

### 4.1.3 学者领域表

一个学者会有多个不同的研究领域，`w` 字段表示了某个学者在该领域的权重。`w` 越大表示学者的研究方向越偏重于该领域。具体字段如下：

表 4.3 学者领域表 `researcher_interest`

| 字段                       | 类型               | 说明   |
|--------------------------|------------------|------|
| <code>user_id</code>     | <code>int</code> | 学者编号 |
| <code>interest_id</code> | <code>int</code> | 领域编号 |
| <code>w</code>           | <code>int</code> | 权重   |

### 4.1.4 中英文对照表

考虑国际化的需求，`web` 页面需要中英文切换，因此对专业术语描述要有中文和英文两种版本。具体字段如下所示：

表 4.4 中英文对照表 `en_zhcn`

| 字段                | 类型                   | 说明  |
|-------------------|----------------------|-----|
| <code>en</code>   | <code>varchar</code> | 英文名 |
| <code>zhcn</code> | <code>varchar</code> | 中文名 |

### 4.1.5 相似度表

由于 web 页面需要推荐相似学者，因此需要计算学者两两之间的相似度，f\_user\_id 表示前一个学者的编号，s\_user\_id 表示后一个学者的编号，distance 表示两个学者之间的相似度。在表中只需存储(f\_user\_id, s\_user\_id, distance)，不需要再额外存储(s\_user\_id, f\_user\_id, distance)，因为这两者含义是一致的。具体字段如下：

表 4.5 相似度表 similarity

| 字段        | 类型    | 说明         |
|-----------|-------|------------|
| f_user_id | int   | 前一个学者的编号   |
| s_user_id | int   | 后一个学者的编号   |
| distance  | float | 两个学者之间的相似度 |

### 4.1.6 学者 h 指数表

在 web 页面中需要用表格将学者信息按照 h 指数高低展示出来，那么我们就需要根据 h 指数对学者进行排序。虽然 h 指数已经存在于学者表中，但如果用排序结果更新学者表将会非常麻烦，而且不利于后续的分片查询。因此我们将学者编号和他的 h 指数单独存入一张表。具体字段如下：

表 4.6 学者 h 指数表 researcher\_h\_index

| 字段       | 类型  | 说明       |
|----------|-----|----------|
| incre_id | int | 每一行的标识   |
| user_id  | int | 学者编号     |
| h_index  | int | 学者的 h 指数 |

### 4.1.7 领域人数表

在 web 页面中，我们需要用柱状图显示各个研究领域的人数，而想要得到这些

统计结果就需要用到 MapReduce 计算框架。MapReduce 计算完成后会将结果保存到文件中，因此我们需要一张领域人数表来将文件中的数据保存到新表中。具体字段如下：

表 4.7 领域人数表 interest\_person

| 字段          | 类型  | 说明   |
|-------------|-----|------|
| interest_id | int | 领域编号 |
| person      | int | 人数   |

#### 4.1.8 领域权重表

在 web 页面中，我们需要用饼图显示各个研究领域的占比，而想要得到这些统计结果就需要用到 MapReduce 计算框架。MapReduce 计算完成后会将结果保存到文件中，因此我们需要一张领域权重表来将文件中的数据保存到新表中。具体字段如下：

表 4.8 领域权重表 interest\_w

| 字段          | 类型  | 说明   |
|-------------|-----|------|
| interest_id | int | 领域编号 |
| w           | int | 权重   |

#### 4.1.9 物理模型图

similarity 的 f\_user\_id 和 s\_user\_id、researcher\_h\_index 的 user\_id、researcher\_interest 的 user\_id 都表示学者编号，因此将 researcher 的 user\_id 设为外键约束。researcher\_interest 的 interest\_id、interest\_w 的 interest\_id、interest\_person 的 interest\_id 都表示领域编号，因此将 interest 的 interest\_id 设置为外键约束。具体的物理模型图如下所示：

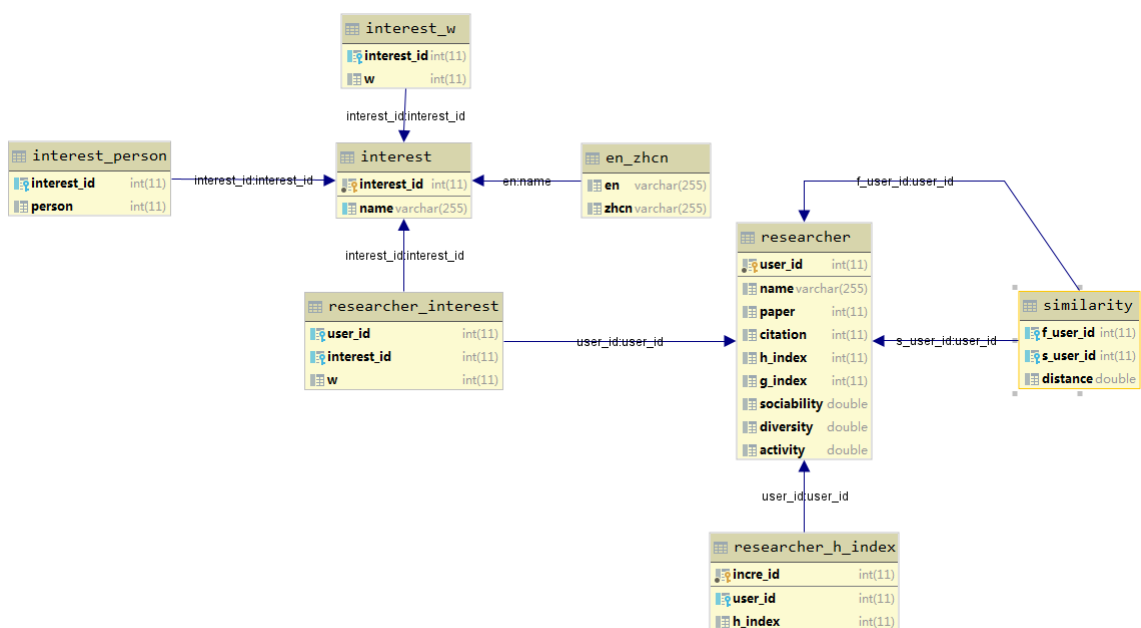


图 4.1 物理模型图

## 4.2 分布式爬虫设计

### 4.2.1 设计要点

在体系结构上，分布式爬虫系统的体系结构有很多种。最典型的系统都是采取主从式的体系结构。即一个主节点控制所有从节点执行爬取任务，这个主节点负责分配种子 url，保证所有从节点的负载均衡。

在健壮性上，要考虑到 ip 被封、连接超时等异常情况。当爬虫出现异常时，应进行相应的异常处理，使得爬虫能继续工作或者系统将任务调度给另一个爬虫。

在部署上，爬虫系统应可扩展、易部署。如果想进一步提高爬取效率，那么可以通过增加爬虫数量进行水平扩展。通过第三方技术，屏蔽各宿主机环境差异，使得爬虫易于部署。

### 4.2.2 分布式架构

如图 4.2 所示，借助于 Redis 数据库实现分布式抓取。基本思想是采用分布式结构设置一个 master 服务器和多个 slave 服务器，master 端管理 Redis 数据库和分发种子 url，slave 端部署并运行爬虫。爬虫提取网页和解析数据，最后将解析的数据存放



于宿主机的数据存储器中。

Redis 数据库中的数据通过 URL 写入器独立写入。对于多个 slave 并发读写 Redis，这里使用了 REDIS\_LOCK 分布式锁和 REDIS\_LOCK\_TIMEOUT 分布式锁过期时间进行控制。

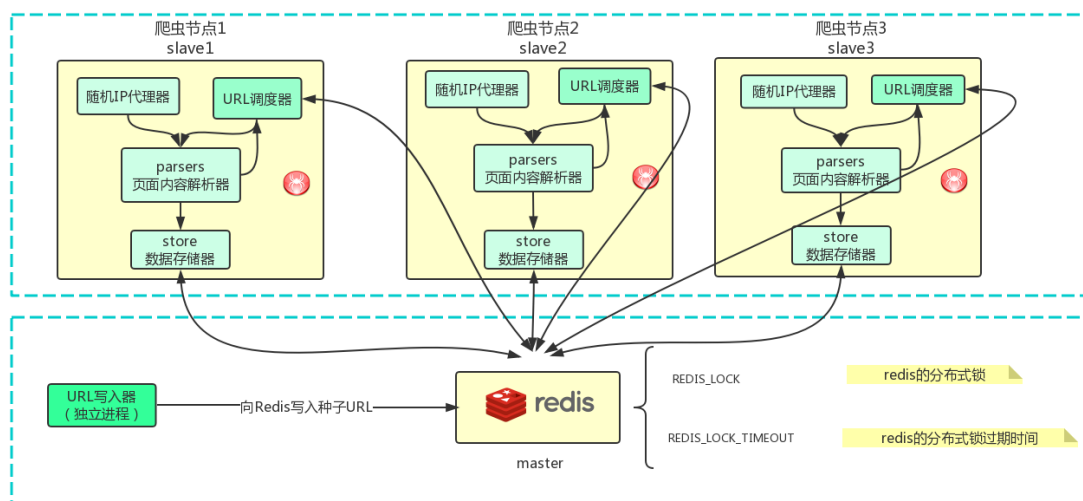


图 4.2 分布式架构

### 4.2.3 ip 代理池

在爬取过程中，如果网站检测到某 ip 访问频率过高，那么可能会封禁该 ip。因此，搭建一个 ip 代理池至关重要。

如图 4.3 所示，代理池由四个模块组成：存储模块、获取模块、检测模块以及接口模块。存储模块使用 redis 来存储代理，同时它也是中心模块，是其它模块通信的中枢。获取模块实际上也是一个爬虫程序，它从代理网站爬取代理，将获取的代理传递给存储模块。检测模块对存储模块中的所有代理进行检测，借助于 redis 的 zset 集合，将代理的检测结果设置为不同的分数。分数越高，代理可用性越高。接口模块通过暴露 api 提供服务，当爬虫需要代理时向接口模块请求可用代理。

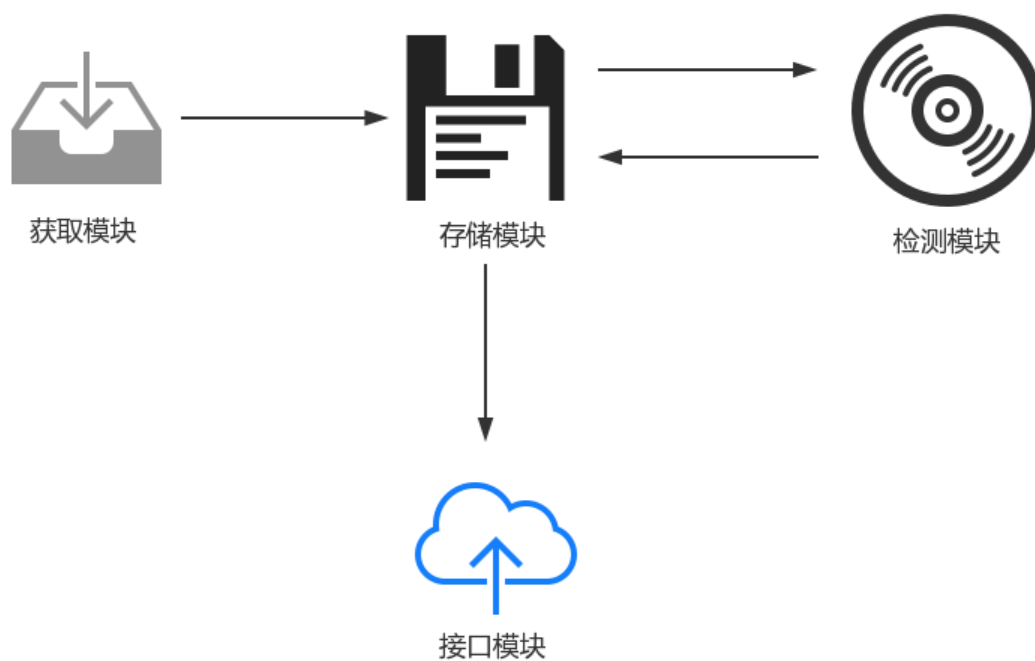


图 4.3 代理池架构

#### 4.2.4 部署环境

如图 4.4 所示，将爬虫程序打包成一个镜像，接着上传到 DockerHub。在 Ubuntu 服务器上安装 Docker 客户端，利用 docker 指令从 DockerHub 拉取爬虫镜像，然后就能在 Ubuntu 服务器上运行爬虫程序了。

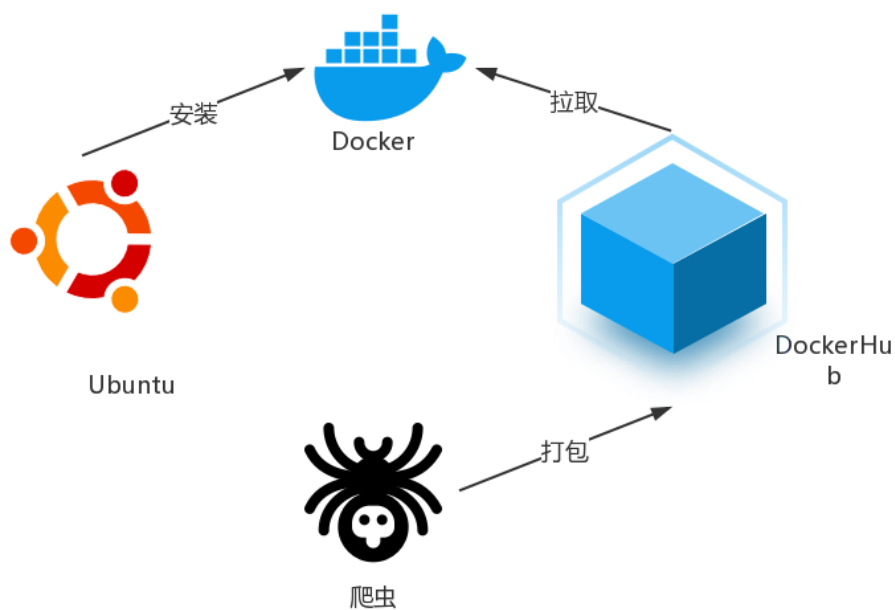


图 4.4 部署环境

### 4.3 MapReduce 计算设计

如图 4.5 所示，首先将文本文件上传到 HDFS 文件系统中，根据设置的数据块大小，HDFS 会将文本文件分割成多个数据块。每个数据块对应一个 Map。

在 Map 阶段，数据以键值对的形式读入，根据业务需求，对键和值进行处理生成新的键值对并将新键值对传送到 Reduce 端。

在传送阶段，shuffle 会将 Map 输出中的键相同的数据整合在一起并按照键名进行排序。

在 Reduce 阶段，Reduce 任务处理所有的键值对数据，按键名把对应的值集合进行汇总，产生新的键值对写入到 HDFS 中。

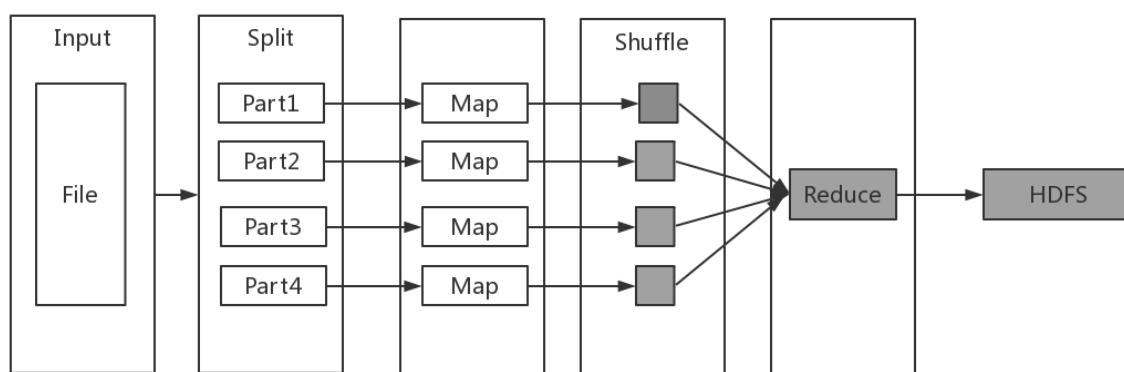


图 4.5 MapReduce 流程

### 4.4 Web 前后端设计

现在 Web 开发最火热的是 MVVM 模式，但是根据功能需求，本平台页面结构简单，组件复用性低，用户交互较少，因此前后端设计仍然采用传统的 MVC 框架模式。如下图所示：

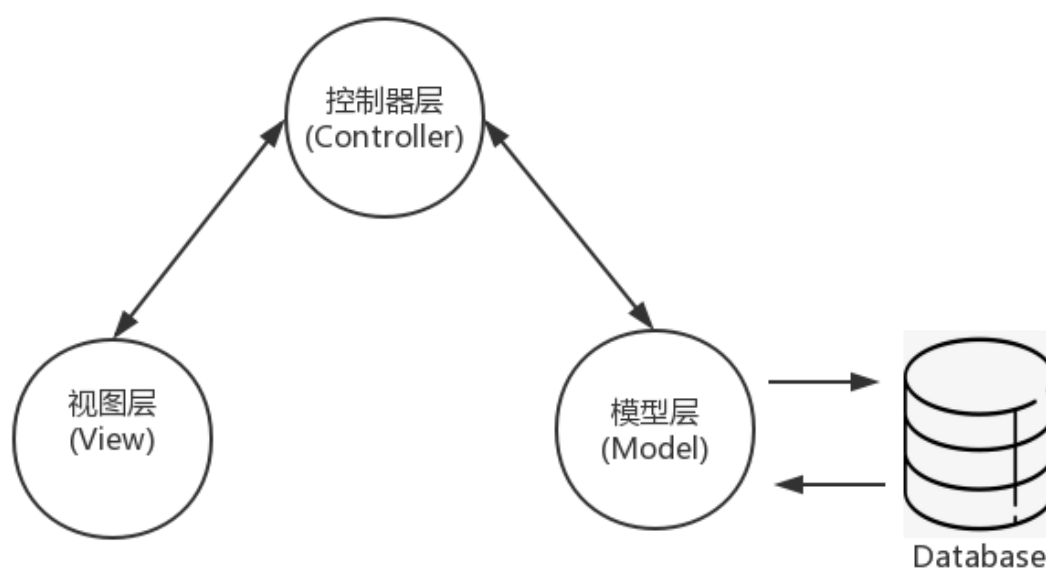


图 4.6 MVC 框架模式

视图层负责展示 web 前端界面，与用户进行人机交互，并将用户输入的指令发送给控制器来处理。

控制器层封装的是系统的业务逻辑，它负责接收前端发来的请求，根据请求所需要的数据向模型层获取数据。在获取数据后，根据业务需求进行格式化处理并返回给视图层。常见的请求转发、请求重定向都是在控制器层完成的。控制器层与视图层的交互是双向的。

模型层封装的是数据表对象，主要负责从控制器层接收数据，然后对数据库进行增删查改操作。模型层会返回数据给控制器层。模型层与控制器层的交互也是双向的。

虽然视图是根据模型进行创建的，但两者并不直接通信，而是通过控制器这个中枢来完成。

## 第五章 系统实现

基于上一章的系统设计，本章阐述系统的具体实现，分析实现难点，给出解决方案，并在最后展示毕设成果。

### 5.1 Hadoop 集群搭建概述

hadoop 集群的搭建是实现分布式存储和计算的基础，但大数据平台搭建的过程较为繁琐，本文不便详述。本节主要介绍集群搭建的要点和难点：

- (1) hadoop 集群是由多台服务器组成的，正常情况下服务器之间的通信会要求用户输入超级管理员密码。为了减少人工干预并让集群顺利运行，使用 ssh 协议实现无密码登录。
- (2) 通过每次切换到 hadoop 目录来运行 hadoop 是一件非常麻烦的事，这里我们为 hadoop 目录配置环境变量。
- (3) 在 hdfs-site.xml 配置文件中设置 blocks 副本备份数量，设置 namenode 的数据存储目录，设置 datanode 的数据存储目录。
- (4) Linux 是一个多人多用户的操作系统，所有的目录和文件都有所有者和对应的读写权限[11]。使用 chown 将目录和文件的所有者更改为 hduser。
- (5) hadoop 集群对时间要求非常高，主节点和从节点的时间必须同步。使用 ntp 协议对服务器或时钟源做同步化从而实现集群间的时间同步。
- (6) 集群机器必须在同一个内网段下，当一台服务器重启，那么它会被分配一个新的内网 ip，这样的话其它服务器就无法通过其原先内网 ip 进行通信。因此每台服务器的内网 ip 设置为固定 ip[12]。

### 5.2 ip 代理池实现

#### 5.2.1 抓取免费代理

由于公网 ip 价格昂贵，对学生是较大的经济负担。我们选择从第三方网站上抓取免费代理，并将代理存放到 redis 的有序集合中。66ip 网站提供了一些免费代理，如图 5.1 所示：

| ip             | 端口号   | 代理位置   | 代理类型 | 验证时间              |
|----------------|-------|--------|------|-------------------|
| 182.111.64.8   | 53364 | 江西省宜春市 | 高匿代理 | 2019年05月12日22时 验证 |
| 5.0.0.815      | 0     | IANA   | 高匿代理 | 2019年05月12日20时 验证 |
| 103.242.14.68  | 31127 | 亚太地区   | 高匿代理 | 2019年05月12日18时 验证 |
| 119.15.95.174  | 8080  | 柬埔寨    | 高匿代理 | 2019年05月12日16时 验证 |
| 139.255.94.122 | 35895 | 印度尼西亚  | 高匿代理 | 2019年05月12日14时 验证 |
| 202.179.21.63  | 46949 | 蒙古     | 高匿代理 | 2019年05月12日12时 验证 |
| 185.15.108.152 | 8080  | 捷克     | 高匿代理 | 2019年05月12日10时 验证 |
| 181.49.131.82  | 8080  | 哥伦比亚   | 高匿代理 | 2019年05月12日08时 验证 |
| 49.128.181.13  | 8081  | 印度尼西亚  | 高匿代理 | 2019年05月12日06时 验证 |
| 41.60.236.182  | 8080  | 赞比亚    | 高匿代理 | 2019年05月12日04时 验证 |
| 116.58.232.84  | 8080  | 泰国     | 高匿代理 | 2019年05月12日02时 验证 |
| 204.48.16.218  | 8080  | 美国     | 高匿代理 | 2019年05月12日00时 验证 |

图 5.1 免费代理

表格中 ip 和端口号的网页结构非常简单，都是由<td></td>标签组成。因此爬取的逻辑也很简单：给定网页 url，获取网页的 table 标签，遍历表格的每一行，提取 ip 和端口号列，将两者组合成 ip:端口号。

### 5.2.2 维护代理池

爬取下来的代理不一定是可用的，因此我们需要维护代理池，删除不可用的代理，保留有效代理。代理是以 ip:端口号的形式存放于有序集合中。redis 的有序集合的每一个元素都有一个分数字段，有序集合就是根据分数字段对元素进行升序排序。我们的维护规则如下：

- （1）当检测到代理可用时，立即将该代理的分数置为 100 分。这样的话，有效代理有更大的机会被获取。如果采用逐步加 1 的方法，那么该代理的分数虽然会越来越高，但是相当长的时间内不会被获取到。而最高分数的代理被频繁获取，可能导致该代理不可用。因此我们选择立即置为 100 分，从而实现代理获取频率的负载均衡[13]。
- （2）当检测到代理不可用时，将该代理的分数减 1，分数减至 0 后，将该代理从 redis 数据库中删除。一个可用代理的分数是 100 分，要将它删除，就必须尝试 100 次都失败。如果有一次成功，那么代理分数又会重置回 100 分。因为代理不可用的因素有很多，比如网络连接超时或被暂时封禁。不能因为一次失败就将该代理删除，允许失败的次数越多，代理被拯救回来的可

能性越高。

- (3) 新抓取的代理初始分数为 50 分。50 分介于 0 分和 100 分之间，代理如果不可用就逐步减 1；如果可用，就立即置为 100 分。

### 5.2.3 获取可用代理

由于爬虫是分布式的，每个爬虫不可能从各自的宿主服务器上的 redis 获取代理，这样的话就无法做到代理的共享和维护。因此我们选择将提供可用代理作为一个单独的 web 服务，以 api 的形式供爬虫程序调用。

### 5.2.4 调度模块

调度模块就是将上述的三个模块以多进程的形式运行起来。代理获取模块作为一个进程，检测模块作为一个进程，api 接口模块作为一个进程。开启进程后，进程的调度顺序由操作系统决定。

## 5.3 数据爬取

爬取数据主要分为两种方式，一种是爬取页面，然后解析页面结构获取数据，另一种是爬取接口。爬取页面需要等待后台返回整个 html 页面，耗时较长，且当网页结构较为复杂的时候，解析效率将大大降低。爬取接口是一种高级做法，通过接口直接获取后端返回的数据，这是一种高效的爬取方式。

如图 5.2 所示，在 aminer 学术网站上利用谷歌浏览器的网络工具来抓取返回的 json 数据：

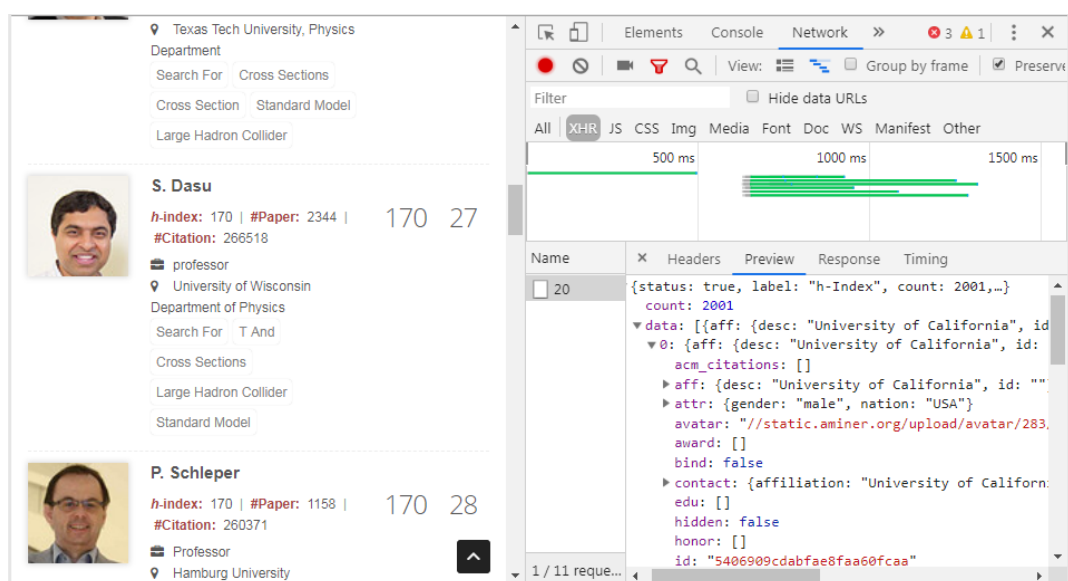


图 5.2 json 数据

通过多次点击下一页按钮，发现了如图 5.3 所示的网络请求信息：

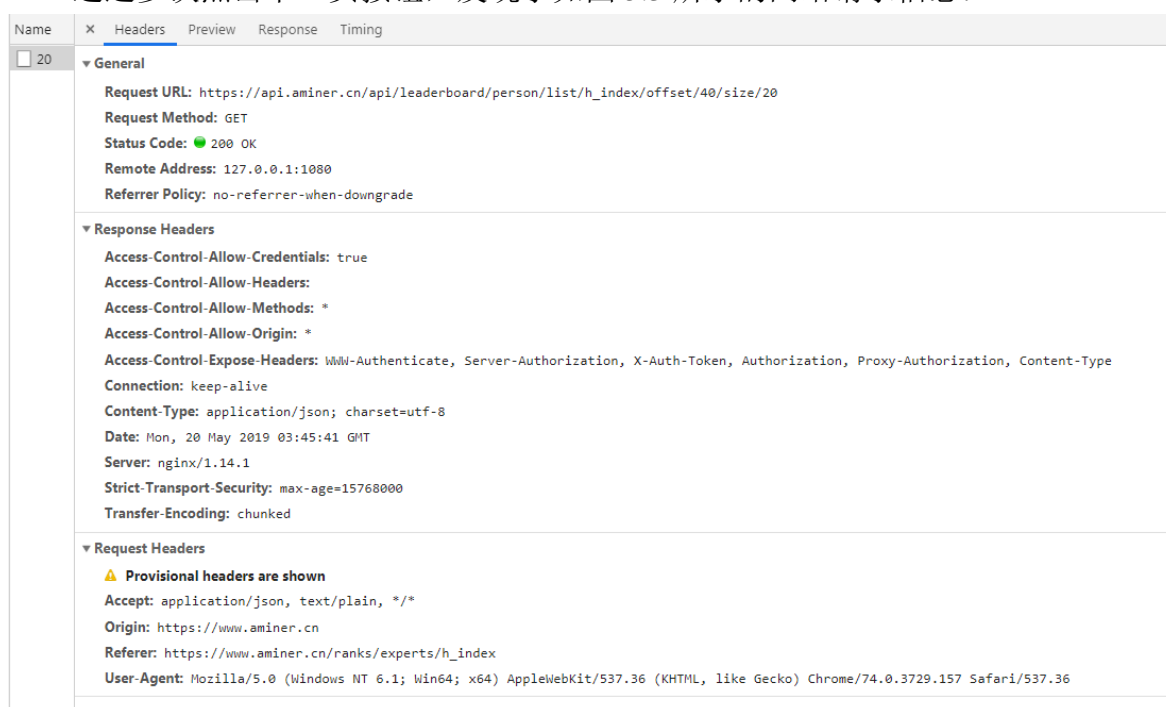


图 5.3 网络请求

网站前端对 ajax 请求做了简单的封装，每次点击下一页按钮发送的 ajax 请求都是 20，这样的话第三方开发者就无法区分接口，因为不同的信息页接口都是一样的。

观察网络请求的头部信息，Request URL 显示了请求的 url。在浏览器中输入该 url 就会返回图 5.2 的 json 数据。更改 offset 和 size 的值，后端就会返回不同的学者信息列表。那么就可以判定该 url 就是 api 接口，它采用了 rest api 的架构，将 offset 和



size 直接写入 url，而不是以查询参数的形式。

offset 表示偏移量，size 表示网页列表展示的学者数量。例如 offset 取值 100，size 取值 20，表示从后台数据库的第一位学者开始偏移 100 位，从偏移后的位置向后取出 20 位学者。当我们要爬取所有的学者信息时，只需按照(0, 20), (20, 20), (40, 20).....这样的递增序列传入(offset, size)就可以获得所有的学者信息。

在返回的 json 数据中，count 字段表示学者总数，data 字段是一个数组，包含了该页的学者信息。data 中每一个元素的 name 字段表示姓名，indices 封装了学者的学术信息，如表 5.1 所示：

表 5.1 indices 内部字段

| 字段           | 说明   |
|--------------|------|
| h_index      | h 指数 |
| g_index      | g 指数 |
| num_pubs     | 论文数  |
| num_citation | 引用次数 |
| activity     | 活跃度  |
| diversity    | 多样性  |
| sociability  | 群集度  |

从上述描述的字段中即可解析出学者的基本信息。

## 5.4 Docker 部署

四台服务器上都要运行爬虫程序，那么四台服务器都应有相同的运行环境。如果按照传统的做法，我们就需要手动地在每台服务器上配置运行环境，而且配置不当还会导致环境冲突，影响到服务器上的其它程序。

docker 技术可以帮助我们解决这一痛点。docker 屏蔽了底层宿主机的环境差异，提供一致的运行环境。我们将爬虫程序打包成一个 docker 镜像，将该镜像上传到 dockerhub。接着在四台服务器上安装 docker 客户端，从 dockerhub 拉取镜像就能运

行爬虫程序了。

## 5.5 Redis 并发控制

redis 是内存型数据库，其并发性能非常好。但是当爬虫数量扩展到成百上千个的时候，我们仍然需要对 redis 进行并发控制。试想一下，如果没有并发控制，多个进程同时读数据库是没有问题的；多个进程写数据库会造成修改丢失；部分进程写数据库，部分进程读数据库会造成不可重复读、脏读以及幻象读[14]。

redis 提供了 setnx 和 getset 原子性操作。我们进行并发控制的思路是这样的：

- (1) 计算锁失效时间  $lock\_timeout = \text{当前系统时间} + \text{过期超时时间}$ ，用 setnx 对 LOCK\_NAME 加锁并设定 lock\_timeout，如果返回 1，则加锁成功，执行步骤四，否则转到步骤二。
- (2) 用 get(LOCK\_NAME) 获取 old\_expire\_time，将当前系统时间与 old\_expire\_time 做比较。如果大于 old\_expire\_time，则表示锁已经失效，其它进程可以获取锁，转到步骤三；否则继续等待。
- (3) 用 getset 对 LOCK\_NAME 设定 lock\_timeout，getset 会返回当前锁失效的时间 current\_expire\_time，如果 old\_expire\_time 等于 current\_expire\_time，说明加锁成功，执行步骤四；如果不相等，表示这个锁又被别的进程抢走了，该进程继续等待。
- (4) 执行业务逻辑。执行完成后，如果当前系统时间小于锁失效时间，释放锁；否则不对锁进行任何处理，因此锁已经失效。

## 5.6 MapReduce 计算实现

### 5.6.1 h 指数排名

这部分计算是负责将学者根据 h 指数高低进行排名。

在 Map 阶段，将读取的每一行字符串根据“--->”分割符进行分割，分别获得学者编号和对应的 h 指数，并产生新的键值对<h 指数，学者编号>发送给 Reduce 端。

在 Reduce 阶段，由于键是 h 指数，而值集合表示 h 指数相同的所有学者编号。遍历值集合的学者编号，产生新的键值对<学者编号，h 指数>输出到 HDFS 中。

另外，shuffle 和 sort 阶段默认是对键进行升序排序，而我们的需求是将 h 指数降序排序，所以我们需要自定义键比较器，重写 compare 方法。

### 5.6.2 研究领域人数统计

这部分计算是负责统计各个研究领域的人数。

在 Map 阶段，将读取的每一行字符串根据“;”分割符进行分割，获得领域编号，并产生新的键值对<领域编号, 1>发送给 Reduce 端。

在 Reduce 阶段，由于键是领域编号，而值集合全为 1。遍历值集合，将这些值相加，就能得到该领域的研究人数。产生新的键值对<领域编号, 研究人数>输出到 HDFS 中。

由于我们需要人数排前 10 的研究领域，因此我们在上下文上设置计数器，当计数达到 10 之后停止计数。

同样的，我们需要自定义键比较器，重写 compare 方法。

### 5.6.3 研究领域比重统计

这部分计算是负责统计各个研究领域的比重。

在 Map 阶段，将读取的每一行字符串根据“--->”分割符进行分割，获得领域编号和某位学者在该领域的投入程度，并产生新的键值对<领域编号, 权重>发送给 Reduce 端。

在 Reduce 阶段，由于键是领域编号，而值集合为各个学者在该领域的投入程度。遍历值集合，将这些值相加，就能得到该领域的权重。产生新的键值对<领域编号, 权重>输出到 HDFS 中。

另外，由于我们需要占比排前 10 的研究领域，因此我们在上下文上设置计数器，当计数达到 10 之后停止计数。

我们仍然需要自定义键比较器，重写 compare 方法。

## 5.7 Web 平台开发

web 平台负责将 MapReduce 的计算结果进行可视化，并提供一些与用户交互的功能。

### 5.7.1 后端实现

在数据库操作上，使用 `sequelize` 库对数据库进行增删查改。本平台的数据库操作没有太多复杂的业务逻辑。

后台提供了两个接口来返回前端页面，一个是统计页面，另一个时候学者详情页面。另外提供了八个接口来返回 `json` 数据给前端。

在中英文页面切换上，后端根据数据表的中英文对照来返回对应的版本描述。

在对学者全字段模糊匹配过程中，由于学者数量众多且字段类型不一，我们需要对学者数量做切分。将每一个分片查询作为一个异步任务，多个异步任务都完成后会返回查询数组[15]。这样就极大地提高了查询效率。

在做相似学者推荐上，我们通过计算学者之间的欧式距离来评价学者间的相似性。

### 5.7.2 前端实现

使用 `echarts` 的柱状图来显示研究人数排前 10 的领域；使用 `echarts` 的饼状图来显示研究比重排前 10 的领域。

使用 `bootstrap table` 展示所有的学者信息。在表格的搜索框中提供模糊匹配功能。在表格的工具栏中提供 `excel` 导出功能。

在学者详情页面，除了展示学者的详细信息外，还会从后端请求数据获取该学者的前 10 名相似学者。

## 5.8 Web 平台展示

### 5.8.1 数据分析

如图 5.4 所示，该柱状图显示了研究人数排名前 10 的领域：

各领域研究人数

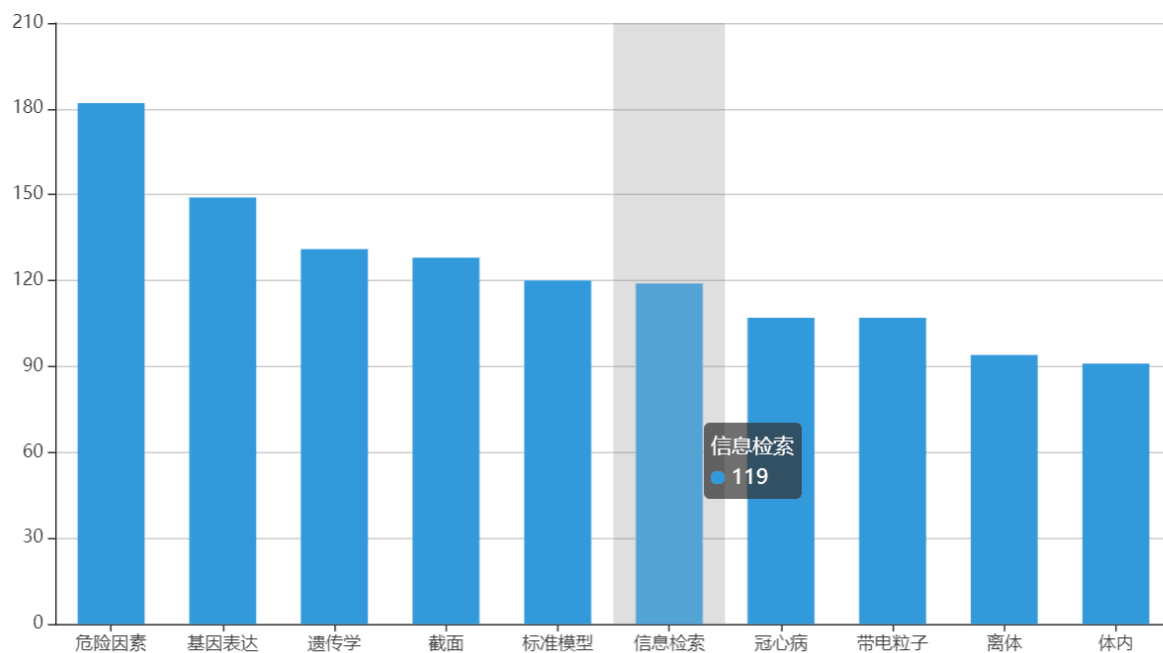


图 5.4 柱状图

如图 5.5 所示，该饼图显示了研究热门程度排名前 10 的领域：把鼠标移动到领域项，会显示该领域的权重和占比，比如数据挖掘权重 2405，百分比为 13.11%；点击饼图下面的 item 可以在饼图上隐藏这个领域。

各领域所占比重

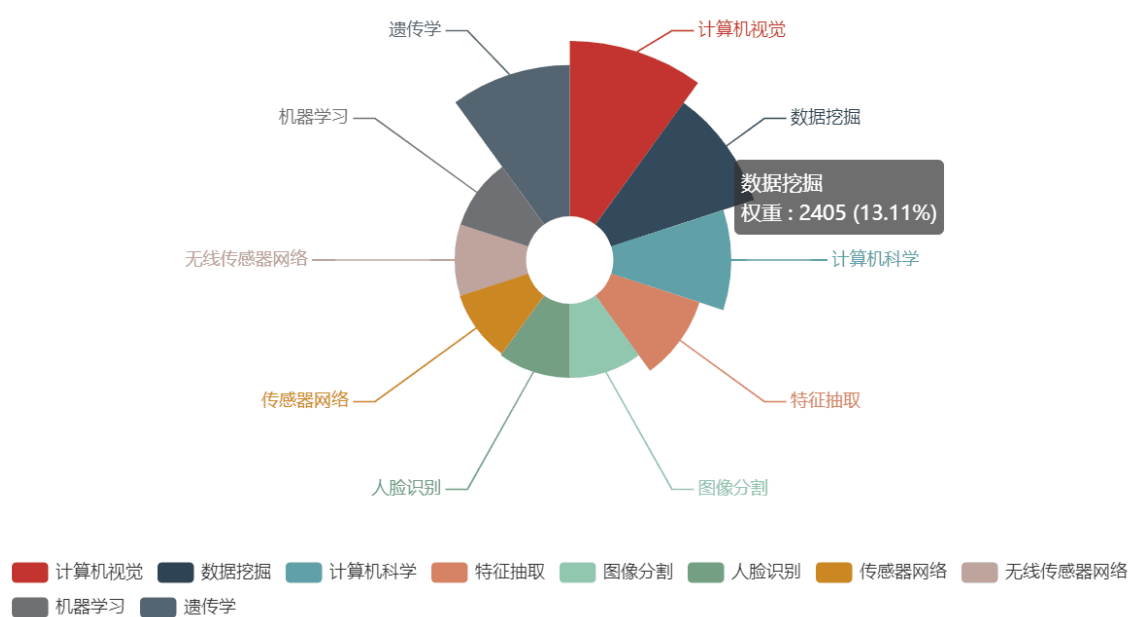
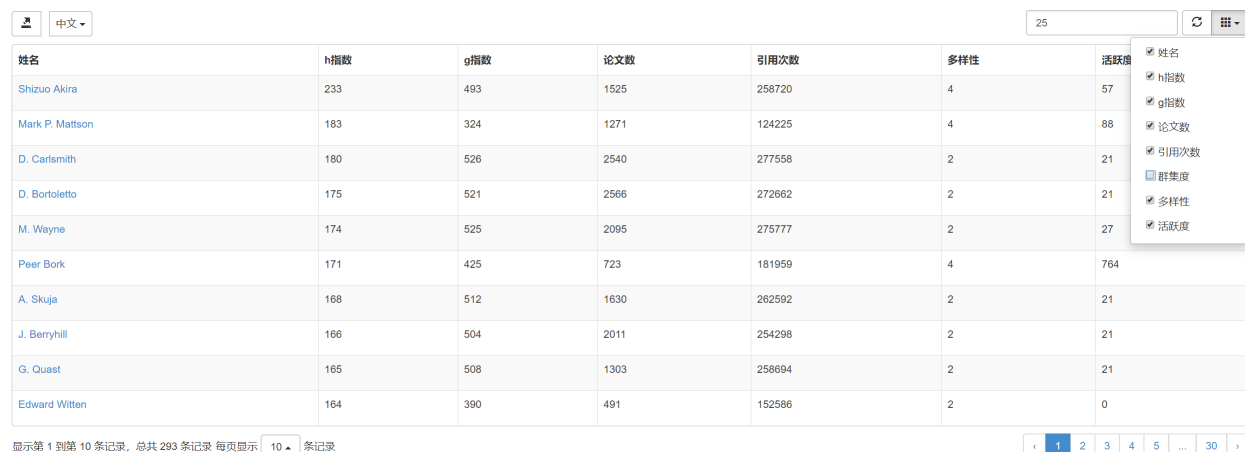


图 5.5 饼图

## 5.8.2 学者信息展示

如图 5.6 所示，表格展示了学者的基本信息：列有姓名、h 指数、g 指数、论文数、引用次数、群集度、多样性和活跃度。



The screenshot shows a table with 8 columns: 姓名 (Name), h指数 (h-index), g指数 (g-index), 论文数 (Paper Count), 引用次数 (Citation Count), 多样性 (Diversity), 群集度 (Clustering Coefficient), and 活跃度 (Activity). The table lists scholars such as Shizuo Akira, Mark P. Mattson, D. Carlsmith, D. Bortoletto, M. Wayne, Peer Bork, A. Skuja, J. Berryhill, G. Quast, and Edward Witten. A search bar at the top right contains the number '25'. A column selector menu is open on the right, showing checkboxes for each column, with '活跃度' (Activity) checked.

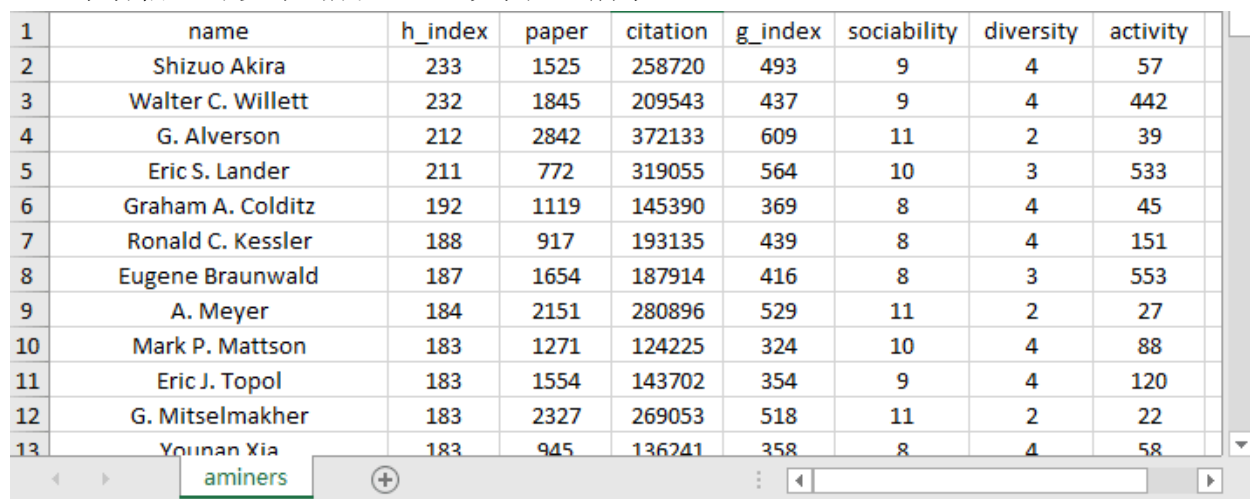
| 姓名              | h指数 | g指数 | 论文数  | 引用次数   | 多样性 | 活跃度 |
|-----------------|-----|-----|------|--------|-----|-----|
| Shizuo Akira    | 233 | 493 | 1525 | 258720 | 4   | 57  |
| Mark P. Mattson | 183 | 324 | 1271 | 124225 | 4   | 88  |
| D. Carlsmith    | 180 | 526 | 2540 | 277558 | 2   | 21  |
| D. Bortoletto   | 175 | 521 | 2566 | 272662 | 2   | 21  |
| M. Wayne        | 174 | 525 | 2095 | 275777 | 2   | 27  |
| Peer Bork       | 171 | 425 | 723  | 181959 | 4   | 764 |
| A. Skuja        | 168 | 512 | 1630 | 262592 | 2   | 21  |
| J. Berryhill    | 166 | 504 | 2011 | 254298 | 2   | 21  |
| G. Quast        | 165 | 508 | 1303 | 258694 | 2   | 21  |
| Edward Witten   | 164 | 390 | 491  | 152586 | 2   | 0   |

图 5-6 信息表格

当信息列过多时，可以点击右侧的单选器，这样的话不感兴趣的列就可以被隐藏。

搜索框支持模糊匹配，比如图 5.6，输入 25 时，会对所有学者所有列进行匹配查询。响应速度也是非常快的。

学者信息可以导出成 excel，如图 5.7 所示：



The screenshot shows an Excel export interface with a table of scholar information. The table has 9 columns: name, h\_index, paper, citation, g\_index, sociability, diversity, activity, and a blank column. The table lists scholars such as Shizuo Akira, Walter C. Willett, G. Alverson, Eric S. Lander, Graham A. Colditz, Ronald C. Kessler, Eugene Braunwald, A. Meyer, Mark P. Mattson, Eric J. Topol, G. Mitselmakher, and Younan Xia. The interface includes a search bar at the top, a table with 13 rows, and a footer with a search bar and a button labeled 'aminers'.

| 1  | name              | h_index | paper | citation | g_index | sociability | diversity | activity |
|----|-------------------|---------|-------|----------|---------|-------------|-----------|----------|
| 2  | Shizuo Akira      | 233     | 1525  | 258720   | 493     | 9           | 4         | 57       |
| 3  | Walter C. Willett | 232     | 1845  | 209543   | 437     | 9           | 4         | 442      |
| 4  | G. Alverson       | 212     | 2842  | 372133   | 609     | 11          | 2         | 39       |
| 5  | Eric S. Lander    | 211     | 772   | 319055   | 564     | 10          | 3         | 533      |
| 6  | Graham A. Colditz | 192     | 1119  | 145390   | 369     | 8           | 4         | 45       |
| 7  | Ronald C. Kessler | 188     | 917   | 193135   | 439     | 8           | 4         | 151      |
| 8  | Eugene Braunwald  | 187     | 1654  | 187914   | 416     | 8           | 3         | 553      |
| 9  | A. Meyer          | 184     | 2151  | 280896   | 529     | 11          | 2         | 27       |
| 10 | Mark P. Mattson   | 183     | 1271  | 124225   | 324     | 10          | 4         | 88       |
| 11 | Eric J. Topol     | 183     | 1554  | 143702   | 354     | 9           | 4         | 120      |
| 12 | G. Mitselmakher   | 183     | 2327  | 269053   | 518     | 11          | 2         | 22       |
| 13 | Younan Xia        | 183     | 945   | 136241   | 358     | 8           | 4         | 58       |

图 5.7excel 导出

## 5.8.3 学者详情页面

当点击学者姓名时，学者详情页面将会弹出。在页面中，除了展示基本信息，我

们还会推荐一些相似学者给用户，表格也同样支持 excel 导出、模糊查询、列隐藏等功能。如图 5.8 所示：

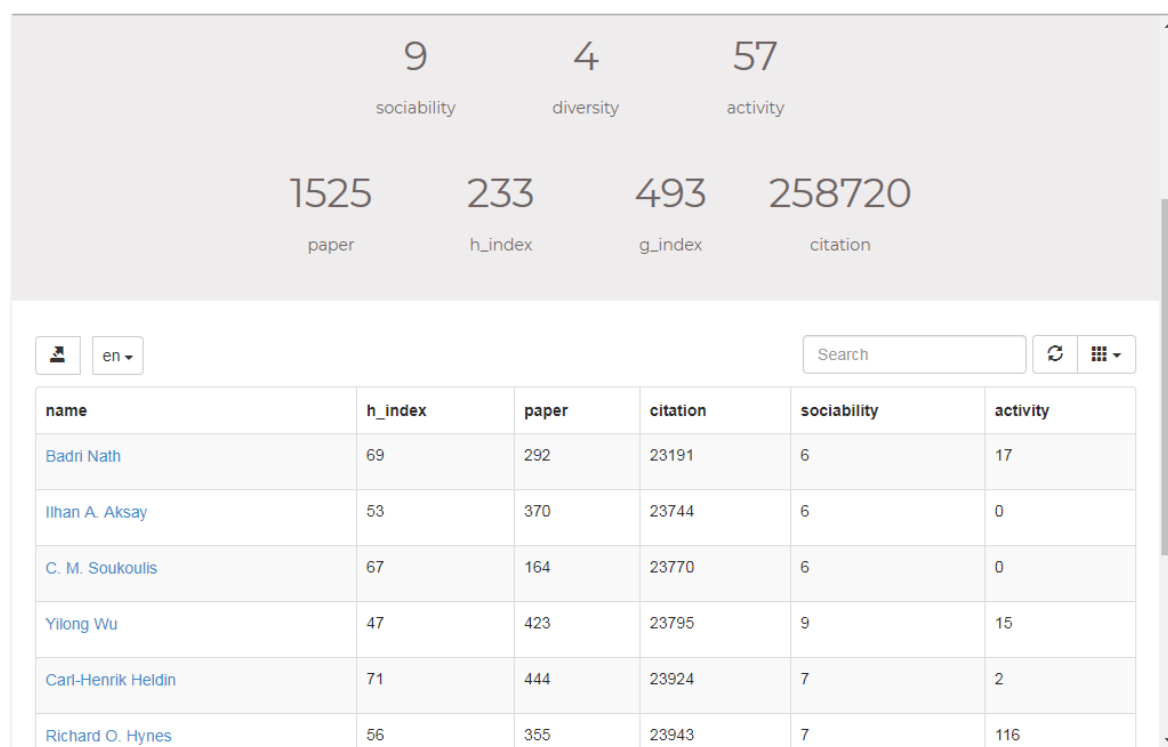


图 5.8 详情页

#### 5.8.4 国际化

本 web 平台支持中英文切换。点击页面工具栏中的中英文按钮即可切换到另一个语言版本。双页面的显示内容是完全一样的，但是导出的 excel 文件内容是对应于当前页面的语言的。比如当前页面语言为中文，那么 excel 的内容也是中文。从此页面点击进入学者详情页面，那么学者详情页面也是中文的。学者详情页面也支持中英文切换。

## 第六章 总结与展望

### 6.1 本文总结

通过为期四个月的系统开发工作，终于圆满地完成了任务书中的每一项计划。在最开始的 hadoop 集群搭建中，遇到了很多困难，比如固定 ip、时间同步等等。在学长的帮助下，最终还是顺利搭建起了集群。hadoop 支持的编程语言是 java，本人做过 java 开发，因此大数据开发上手就很快了。

编写爬虫程序需要有前端基础，以前也写过爬虫爬取网页，但是这种爬虫在健壮性、爬取效率上是远远不能满足要求的。ip 被封、cookie 被禁等异常都是要仔细考虑的问题，再加上分布式，那么爬虫所涉及到的知识面就太广了。本人借了几本爬虫类的书籍，跟着书上的代码将爬虫示例运行了一遍。经过半个月的学习，对爬虫的技术思路有了一个清晰的了解。同时，本人也是第一次将 docker 技术应用于实战，这极大地增强了自己在后续工作中使用 docker 的信心。

数据库的并发控制是一个相当复杂的问题，在本次毕设中也模拟了多个爬虫对 redis 的并发读写。幸运的是，redis 为我们提供了一些原子操作，方便了对 redis 的并发编程。在实际互联网领域中，应对高并发涉及到整个系统架构的设计，而不仅仅是选择哪一门开发语言、哪一种数据库、哪一种服务器就能解决的。

web 平台的开发是使用的 express 框架，nodejs 是事件驱动、异步 I/O 的，因此这个框架扩展性强、并发性能好。后端 js 代码的编写增强了本人对异步编程的理解，echarts 和 bootstrap 库的使用极大地减少了本人前端开发的工作量，也提高了 web 页面的美观度。

总的来说，整个毕设的实现思路是非常清晰的，这为我顺利完成毕设奠定了基础，也让我有了充足的时间对毕设系统进行完善。

### 6.2 后续展望

目前系统的基本功能虽然已经完成，但仍有一些不足之处。我们可以从如下几个方面开展工作：

- （1）数据量：爬虫爬取的数据量仍然太少，不足以充分发挥分布式爬虫的优势



以及 MapReduce 的计算效率。

- (2) 数据存储：当数据量达到一定程度后，关系型数据库就会遇到性能瓶颈。借助于分布式文件存储系统 HDFS，我们可以使用 Hive 进行海量数据存储以及使用 hql 语句实现 MapReduce 计算。
- (3) 界面自适应：echarts 图表在移动端自适应较差，对用户交互不够友好。
- (4) 学者画像：对于学者的描述太过简陋，没有列举学者论文成果以及可视化展示近几年的学术活动。

## 参考文献

- [1] 崔庆才. Python3 网络爬虫实战[M]. 北京: 人民邮电出版社, 2018.
- [2] 王万良. 人工智能及其应用[M]. 北京: 高等教育出版社, 2015.
- [3] 余明辉, 张良均. Hadoop 大数据开发基础[M]. 北京: 人民邮电出版社, 2018.
- [4] 朱少民. 软件测试方法和技术[M]. 北京: 清华大学出版社, 2005.
- [5] 程宝雷, 章晓芳. 软件测试与质量保证[M]. 北京: 清华大学出版社, 2015.
- [6] 周志明. 深入理解 Java 虚拟机[M]. 北京: 机械工业出版社, 2013.
- [7] 林大贵. Python+Spark2.0+Hadoop 机器学习与大数据实战[M]. 北京: 清华大学出版社, 2017.
- [8] 安俊秀, 王鹏, 靳宇倡. Hadoop 大数据处理技术基础与实践[M]. 北京: 人民邮电出版社, 2015.
- [9] J. Dean, S. Ghemawat. MapReduce: simplified data processing on large clusters[J]. Communications of the ACM, 2018, 51(1): 107-113.
- [10] Marc Harter. Node.js 实战[M]. 北京: 人民邮电出版社, 2014.
- [11] Silberschatz, Galvin, Gagne. 操作系统概念[M]. 北京: 高等教育出版社, 2007.
- [12] 谢希仁. 计算机网络[M]. 北京: 电子工业出版社, 2017.
- [13] Thomas H.Cormen, Charles E.Leiserson, Ronald L.Rivest Clifford Stein. 算法导论[M]. 北京: 机械工业出版社, 2013.
- [14] Hector Garcia-Molina, Jeffrey D.Ullman, Jennifer Widom. 数据库系统实现[M]. 北京: 机械工业出版社, 2010.
- [15] Eric Freeman, Elisabeth Freeman, Kathy Sierra, Bert Bates. Head First 设计模式[M]. 北京: 中国电力出版社, 2007.

## 致 谢

时光飞逝，四年的学习生涯已接近尾声。首先，我要感谢王进老师，是王进老师为我提供了项目实战的机会，增强了我的编程能力。感谢黄河老师，是黄河老师让我爱上编程，停车场代码是我目前写得最为骄傲的一段代码。感谢陈越老师，是陈越老师让我对数据库产生了浓厚的兴趣。

其次我要感谢沈永亮学长，感谢学长培养了我坚韧的性格，让我学会了为人处世的准则。

然后我要感谢我的舍友，感谢海鸥对我代码上的指导，感谢费智对我学习效率的提高，感谢溢峰带给我生活品味的提高，感谢凌寒带给我生活上的欢乐，感谢浩然给与我资源上的帮助。

最后我要感谢我的家人，感谢你们在学业上和生活上无私的帮助。我会继续努力，在科研的新道路上再创辉煌。