

**本 科 毕 业 设 计（论文）**



**题目: 基于B/S的可信数据爬虫的设计与实现**

**姓 名 吴浩男**

**学 院 计算机学院**

**专 业 计算机科学与技术**

**班 级 2013211303**

**学 号 2013211228**

**班内序号 09**

**指导教师 袁 玉 宇**

**2017年 6月**

**北 京 邮 电 大 学**

**本科毕业设计（论文）诚信声明**

本人声明所呈交的毕业设计（论文），题目《基于博弈论的游戏设计与实现》是本人在指导教师的指导下，独立进行研究工作所取得的成果。尽我所知，除了文中特别加以标注和致谢中所罗列的内容以外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得北京邮电大学或其他教育机构的学位或证书而使用过的材料。

申请学位论文与资料若有不实之处，本人承担一切相关责任。

本人签名： 日期：

**基于B/S结构的可信数据爬虫的设计与实现**

**摘 要**

网络爬虫是一种在现今社会非常有用的数据获取工具，其便捷程度能够帮助人们获得大量信息。市面上主要依托爬虫进行业务开展的公司有许多，如，百度、今日头条等。同时无论是政府，还是传统行业，都有许多需要大量数据支持才能进行的任务。但是网络爬虫的使用与设计对不具备编程技巧的普通民众来说十分地不友好，使得网络爬虫的普及程度较低，大部分的民众都不能享受到网络爬虫的便利。本毕设就是想要协助完全不会编程的用户快速抓取网页上的信息，从而使得普通民众也能够更加便捷地从网络上收集数据。

本毕设设计与实现的是基于B/S结构的针对单个静态页面的网络爬虫。本毕设的核心模块是一个基于多种评估算法的网络爬虫。通过一系列的估算算法，此爬虫不需要人工定制程序，就能够抓取绝大多数静态页面上的内容；同时本毕设也使用了BeautifulSoup等第三方库，使得抓取模块的鲁棒性和错误处理能力更强。本毕设的功能模块与业务逻辑以Django为基础架构设计Web端应用。Django架构是以Python语言为基础的网络端应用架构，适用于中小型网络应用，可以利用的第三方库较全，维护代码的成本低。通过将毕设架设在服务器上，能够通过任意浏览器访问，方便用户使用本毕设。

本毕设已经完成了核心模块的开发、业务逻辑的实现，可以通过Chrome、Safari等浏览器访问，并通过一些测试方法使得它的错误率降低、鲁棒性提高。只需要加上一些负载均衡技术以及多线程的操作，就达到了可以上线发布的水平。

**关键词** 网络爬虫 Web开发 Django架构 Beautifulsoup库

**The design and implementation of a based on B/S crawler for trustworthy data**

**ABSTRACT**

Web spider is a very useful data acquisition tool in today's society, and its ease of use can help people get a lot of information. Many companies on the market rely mainly on Web spider to carry out business, such as Baidu, Byte Dancing and so on. At the same time, both the government and the traditional industry, there are also many tasks that require many data to support. But the use and design of web spider are very unfriendly to ordinary people who do not have programming skills, making the popularity of web crawlers low, and most people can not enjoy the convenience of web spider. This is exactly what I want to help users who are not programming at all to quickly grab the information on the page, making it easier for ordinary people to collect data from the web.

The design and implementation of this is based on the B / S structure for a single static web spider. The core module is a web spider based on a variety of estimation algorithms. Through a series of estimation algorithms, this estimation does not need to manually customize the program, you can crawl the contents of the vast majority of static pages. Moreover, I also use the BeautifulSoup third-party library, making the robustness of the crawler module and Error handling ability. The functional modules and business logic are designed to design Web applications based on Django. Django framework is based on the Python, which is a nice web framework, and is very suitable for small and medium-sized network applications. I can use serveral Python third-party library, and reduce the cost of maintaining the code. By deploying the application to the webserver, users can visit it through any browser.

The completion of the core module has been completed, the realization of business logic, you can Chrome, Safari and other browser access, and through a number of test methods make it less error rate, improve the robustness. Only need to add some load balancing technology and multi-threaded operation, it can reach the level of online release.

**KEY WORDS** Web Spider Web application Django framework BeautifulSoup library

**目 录**

[第一章 引言 1](#_Toc482203165)

[1.1 研究背景及意义 1](#_Toc482203166)

[1.2 本文篇章结构 1](#_Toc482203167)

[第二章 相关技术研究 3](#_Toc482203168)

[2.1 后端框架 3](#_Toc482203170)

[2.1.1 Django框架的优点 3](#_Toc482203171)

[2.1.2 Django框架总览 3](#_Toc482203172)

[2.1.3 Django框架核心组件 4](#_Toc482203173)

[2.2 前端框架 5](#_Toc482203174)

[2.2.1 Bootstrap的优点 5](#_Toc482203175)

[2.2.2 Bootstrap包含的样式与组件 5](#_Toc482203176)

[2.3 网络爬虫 5](#_Toc482203177)

[2.3.1 网络爬虫的抓取步骤 5](#_Toc482203178)

[2.3.2 BeautifulSoup库 6](#_Toc482203179)

[2.3.3 Requests库 6](#_Toc482203180)

[2.4 本章小结 7](#_Toc482203181)

[第三章 需求分析与系统设计 8](#_Toc482203182)

[3.1 需求分析 8](#_Toc482203184)

[3.2 数据流图分析 8](#_Toc482203185)

[3.2.1 顶层数据流图 8](#_Toc482203186)

[3.2.2 前端子系统数据流图 9](#_Toc482203187)

[3.2.3 后台子系统数据流图 9](#_Toc482203188)

[3.2.4 抓取子系统数据流图 10](#_Toc482203189)

[3.3 模块独立性 11](#_Toc482203190)

[3.3.1 内聚度分析 12](#_Toc482203191)

[3.3.2 耦合度分析 12](#_Toc482203192)

[3.3.3 独立性分析 12](#_Toc482203193)

[3.4 数据库设计 13](#_Toc482203194)

[3.4.1 概念结构设计 13](#_Toc482203195)

[3.4.2 逻辑结构设计 13](#_Toc482203196)

[3.5 本章小结 14](#_Toc482203197)

[第四章 抓取算法设计 15](#_Toc482203198)

[4.1 定义 15](#_Toc482203200)

[4.1.1 名词定义 15](#_Toc482203201)

[4.1.2 对象定义 16](#_Toc482203202)

[4.2 算法分析 17](#_Toc482203203)

[4.2.1 锚定位算法 17](#_Toc482203204)

[4.2.2 基于CSS样式与tag的抓取 18](#_Toc482203205)

[4.2.3 兄弟节点分类法 20](#_Toc482203206)

[4.2.4 基于路径相似度与CSS样式的抓取 21](#_Toc482203207)

[4.2.5 LCA与LCA统计 23](#_Toc482203208)

[4.2.6 基于路径相似度与CSS样式的数据分类算法 25](#_Toc482203209)

[4.3 本章小结 27](#_Toc482203210)

[第五章 系统实现 28](#_Toc482203211)

[5.1 系统工程概述 28](#_Toc482203213)

[5.1.1 项目主应用 28](#_Toc482203214)

[5.1.2 信息管理应用 29](#_Toc482203215)

[5.1.3 爬虫抓取应用 31](#_Toc482203216)

[5.2 用户功能 31](#_Toc482203217)

[5.2.1 注册与登陆 31](#_Toc482203218)

[5.2.2 爬虫任务 33](#_Toc482203219)

[5.2.3 数据操作 36](#_Toc482203220)

[5.3 管理员功能 38](#_Toc482203221)

[5.3.1 用户管理 39](#_Toc482203222)

[5.3.2 爬虫任务管理 41](#_Toc482203223)

[5.3.3 爬虫数据文件管理 42](#_Toc482203224)

[5.4 系统测试 42](#_Toc482203225)

[5.4.1 Interbrand测试 42](#_Toc482203226)

[5.4.2 BrandFinance测试 43](#_Toc482203227)

[5.4.3 测试结果 45](#_Toc482203228)

[5.5 本章小结 45](#_Toc482203229)

[第六章 总结与展望 46](#_Toc482203230)

[6.1 实践成果 46](#_Toc482203232)

[6.2 工作展望 46](#_Toc482203233)

[6.2.1 分布式部署 46](#_Toc482203234)

[6.2.2 抓取模型可重复利用 46](#_Toc482203235)

[6.2.3 算法模型的改进 47](#_Toc482203236)

[参考文献 48](#_Toc482203237)

[致 谢 49](#_Toc482203238)

# 引言

## 研究背景及意义

互联网的迅速发展使得信息的产生速率、传播规模达都到史无前例的水平。信息爆炸使得世界更加紧密地联合在了一起，同时也带来了诸多问题。其中最引人关注的就是：人们将难以从汗牛充栋的信息中快速而准确地得到自己最需要的信息。

因此就诞生了网络爬虫技术。网络爬虫是指一种按照一定规则，自动解析万维网网页源代码并从中提取信息的程序或者脚本。

网络爬虫分为多个种类。

例如，传统搜索引擎中的通用网络爬虫，百度、谷歌、雅虎等通用搜索引擎能够大规模抓取网页信息，辅助人们进行信息检索。但是这类爬虫往往不能很好地满足不同领域、不同背景的检索要求，返回的结果往往包含大量用户不关心的信息，难以得到可信数据。

聚焦网络爬虫就是为了解决此类的问题而诞生的。它能够从多个初始网页的地址开始，获得其中的可信数据；在解析网页的过程中，它通过一定的规则，从当前页面上选取一定的新网页地址，并加入抓取队列。它的工作流程与实现更为复杂，需要通过一定的网络分析算法将网页源代码结构化，分析其中信息并抽取可信数据。同时它还需要通过一定的策略过滤与主题无关URL，从而选取合适的抓取路线。这类爬虫对编程技巧有着极高的要求，需要对每一类型的网页进行定制化处理，难以通用，使得没有编程基础的普通民众难以借助此类爬虫获得可信数据。

基于以上原因，我想要设计出一种能够尽量减轻爬虫用户编程需求并且适用于多种类型网页的网络爬虫，使得网络爬虫能够被更多人使用。

## 本文篇章结构

本文共分为六章，每个章节的内容如下：

第一章：主要介绍了本课题研究的背景及意义和本文的篇章结构。

第二章：介绍了相关研究工作，包括该毕设使用的各种基础框架，如Django网络应用快发框架、Bootstrap前端框架、BeautifulSoup爬虫框架。

第三章：主要介绍了系统的需求和总体设计，包括概要设计中的数据流图分析、模块化分析，详细设计中的数据库概念设计与逻辑设计等。

第四章：主要介绍了抓取算法的设计思路，提供了算法的功能与具体实现的思路与伪代码，同时还分析了算法时空复杂度。

第五章：主要介绍了系统功能的实现，包括系统文件结构的说明、系统功能界面的展示与说明。同时还对系统进行功能性测试，展示测试过程与测试结果。

第六章：总结实践成果，展望未来工作。

# 相关技术研究



## 后端框架

后端框架采用的是以Python语言为基础的Web应用便捷框架Django。Django原本是一个用于迅速开发数据库驱动的新闻系统，但是在Web领域展示出它的独特优势之后，它就逐渐成为了Web框架中的领头羊。

### Django框架的优点

1. **语言优势：**Django使用的是Python语言，Python是一门面对对象的应用程序开发语言，它同时兼顾系统语言（如C/C++和Java）的强大和脚本语言（如Ruby和Visual Basic）的灵活迅速。同时Python也提供了多种第三方库，使得Django后台的功能实现更加容易。
2. **一体化设计：**Django安装简单且灵活多变，开箱即用，无需安装其他组件就能够运行程序，它同时也能够把每个部分都能拆分下来换装成其他组件。简而言之，它的具有高度的内聚性和较低的耦合性，开发人员能够在软件工程设计的过程中极大地借助此类特性进行敏捷开发。
3. **容错率高：**除了Python提供的异常处理外，Django也具有极强的容错率，能够解决包括浏览器不兼容、宽带和服务器限制以及总体框架设计等一系列挑战。

### Django框架总览

对于用户而言，他们最接近的部分就是HTTP通信协议。通过URL，他们可以给服务器的Web应用发送请求，而服务器中布置的Django通过一些系列的逻辑运行，通过views模块返回HTTP响应，最后用户在Web客户端接受响应。

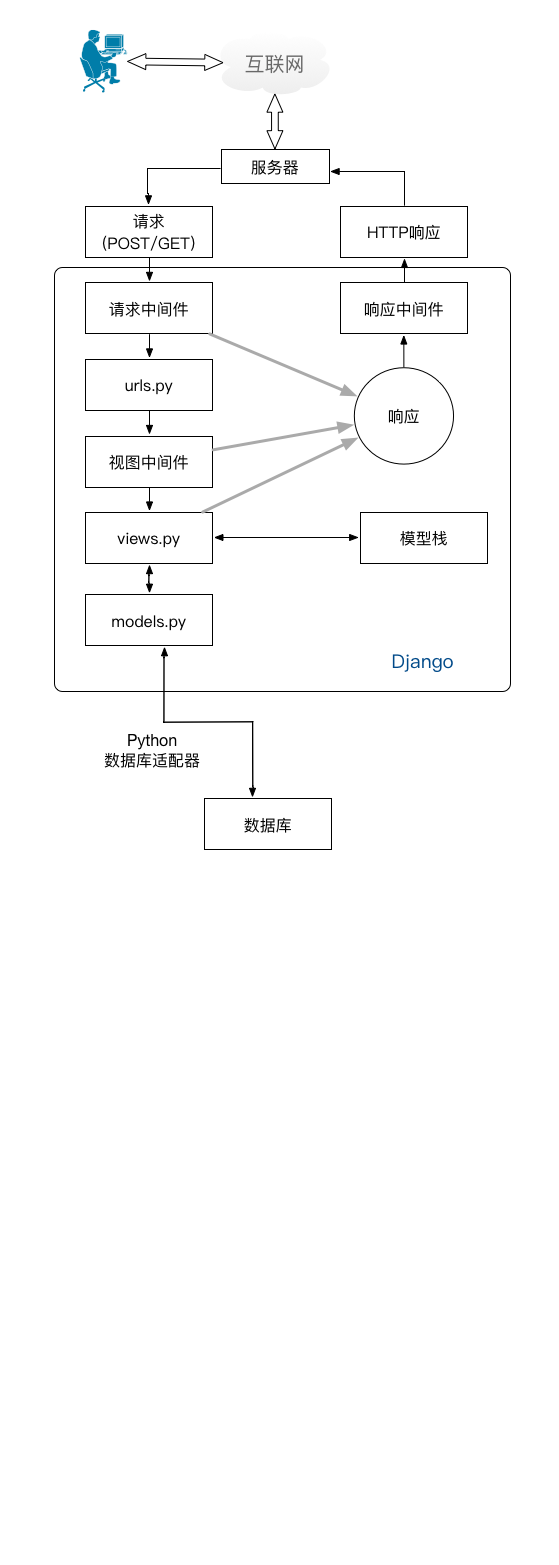


图 2-1 Django总览

### Django框架核心组件

Django采用了MVC(Model-View-Controller)的动态应用程序分层思想，将系统划分为模型（控制数据）、视图（定义显示方法）以及控制器（在两者之间斡旋，并且让用户能够请求和操作数据）。

1. **应用：**功能组件包，高内聚度的模块化设计。
2. **模型：**创建与数据库有关的对象[的关系映射](http://baike.baidu.com/item/%E5%AF%B9%E8%B1%A1%E5%85%B3%E7%B3%BB%E6%98%A0%E5%B0%84)。Django封装之后的映射函数能够快速连接多种数据库，同时，也使得开发人员能够完全避免直接通过编程语言操作数据库。
3. **视图：**用于决定用户界面的显示，负责响应用户界面的操作信息，是主要的业务逻辑处理中心，能够连接其他的应用与本应用内的其他部分。
4. **URL控制：**一流的 URL 设计，把收到的请求与视图函数相匹配。

## 前端框架

前端框架使用的是来自Twitter的Bootstrap，它是一款用于设计Web应用程序的免费、开源的Web前端框架。它包含用于排版，表单，按钮，导航和其他界面组件的基于HTML和CSS的设计模板，以及可选的JavaScript扩展。这是一个仅关注前端开发的Web框架。

### Bootstrap的优点

1. **响应式设计：**Bootstrap的响应式CSS能够自适应于不同分辨率设备，能够在用户使用不同的设备访问网页时，根据分辨率实现自动布局调整。
2. **多设备支持：**支持所有主流浏览器，支持移动设备访问。
3. **容易上手：**开发人员只需具备HTML等前端语言的基础知识，就可以直接使用Bootstrap。它内置了功能强大的组件，易于使用。

### Bootstrap包含的样式与组件

1. **基础结构：**Bootstrap提供了一套响应式、移动设备优先的流式栅格系统，随着屏幕或视口尺寸的增加，系统会自动分为最多12列。此响应式网格构成了Bootstrap的最基础的结构。
2. **CSS样式：**Bootstrap提供了一组全局的CSS样式表，为所有关键HTML组件提供基本的样式定义，同时它还支持可扩展的class样式。
3. **组件：**Bootstrap包含了十几种可以重用的组件，如下拉菜单、导航栏、按钮、输入框、弹出框等等，这些组件由CSS类实现，与HTML元素结合进行页面渲染。
4. **插件：**Bootstrap自带多种JavaScript插件，扩展了功能，它们提供了其他界面元素，如对话框，工具提示和转盘。它们还扩展了一些现有接口元素的功能，包括例如输入字段的自动完成功能。开发者可以选择性地使用插件，同时也可以轻松地添加第三方的插件库。

## 网络爬虫

采用了多种Python爬虫库，取长补短，能够更方便地实现抓取算法的功能。

### 网络爬虫的抓取步骤

* 1. 读取并结构化网页源码：

通过HTTP库访问网址并得到网页源代码，再通过HTML解析库将HTML源代码结构化。一般的，源代码将会被结构化成一科分析树。

* 1. 选取有效信息：

通过搜索第一步结构化后的分析树，利用深度优先搜索、XPath等方法定位需要抓取的节点。

* 1. 存取有效信息：

通过CSS选择器、HTML标签、XPath以及其他抓取规则，将有效信息收集到文件中。

* 1. 选取有效的URL进行继续抓取：

通过一定的正则匹配，将需要抓取的下一个网址加入抓队列中，直到队列为空，则不能再进行抓取。

### BeautifulSoup库

BeautifulSoup是一个用于解析HTML和XML文档的Python库，能够将HTML结构化为一棵Dom树。使用者可以使用一些在树上的搜索的算法来分析树上信息并从中获取可信数据。

BeautifulSoup的解析操作对应的是爬虫的抓取步骤①，它将网页源码结构化成四种对象，分别是Tag、NavigableString、BeautifulSoup、Comment。

其中Tag是Dom树的节点、NavigableString是Dom树的叶子节点、BeautifulSoup是Dom树本身，同时也Dom树的根节点。通过在Dom树上进行一系列的搜索，我们就可以得到想要的信息。

与此同时，BeautifulSoup还提供了多种针对爬虫的抓取步骤②与步骤③的函数方法、对象属性，能够迅速搜索Dom树并快速获得Dom树中的各种属性。

如tag对象的attrs属性，方法find\_all()、get()，CSS选择器等。

### Requests库

Requests 是用Python语言编写，基于urllib，采用Apache2 Licensed开源协议的HTTP库。它比Python自身提供的HTTP库更加方便、简洁，可以大量减少我们的工程量。

Requests的哲学是以PEP 20的习语为中心开发的，所以它比urllib更加Pythoner。

下面是Requests库的函数使用方法，以及其他响应函数：

requests.get() #GET请求

requests.post() #POST请求

requests.head() #HEAD请求

requests.options() #OPTIONS请求

r.status\_code #响应状态码

r.content #字节方式的响应体，会自动为你解码 gzip 和 deflate 压缩

r.text #字符串方式的响应体，会自动根据响应头部的字符编码进行解码

r.headers #以字典对象存储服务器响应头，但是这个字典比较特殊，字典键不区分大小写，若键不存在则返回None

r.json() #Requests中内置的JSON解码器

## 本章小结

本章主要介绍了系统使用的几种主要的框架与第三方库，介绍了其基本信息与部分开发知识，分析了它们的优点，能够对本毕设系统有何帮助，并列举了其中部分应用的使用方法，为接下来的系统分析打下了基础，帮助理解。

# 需求分析与系统设计



## 需求分析

本毕设希望能够让不具备编程能力的普通民众也能自如地使用爬虫，即，用户操作界面不能有任何与编程相关的设计。

爬虫核心需求如下：

1. 输入抓取网页的网址，不需要正则匹配的网址；
2. 输入抓取内容样例；
3. 输出一个存储信息的二维矩阵。

本毕设的工程需求如下：

1. 实现一个Web应用，具有一般Web应用的基础功能，如，用户注册、用户登录、管理员功能等。
2. 实现爬虫任务的创建、查看、使用。
3. 实现爬虫任务数据的查看、导出。
4. 实现多网站抓取的范例。

## 数据流图分析

### 顶层数据流图

整个系统分成浏览器（前端子系统）和服务器（后端子系统）两个部分。本系统的外部实体为用户，数据存储部分使用的是操作系统提供的文件系统和数据库。

1. 用户在浏览器上对网页进行操作，则对前端子系统发起操作信息的输入；
2. 浏览器接受并处理用户发出的网页操作信息后，将之转化为对应的HTTP请求转发到服务器；
3. 服务器处理相关的HTTP请求后，返回相应的HTTP响应，必要的时候会调用数据库和文件系统，进行数据存储和读取操作；
4. 浏览器接受到服务器返回的HTTP响应后根据响应的内容对网页展示进行修改，包括网页跳转、网页内容改变等。
5. 用户查看接收HTTP响应后的网页，并从中得到响应的信息。

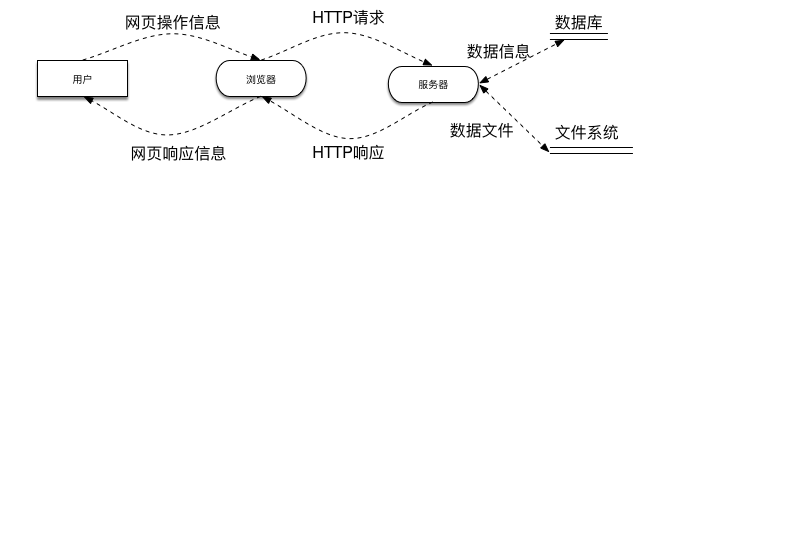


图 3-1 顶层数据流图

### 前端子系统数据流图

前端子系统分成HTML静态资源和Javascript动态控制器，前者包括HTML语言和CSS样式，后者包括Javascript语言以及由Javascript开发的各类插件。

1. 前端子系统从用户处接收网页操作信息，这些信息能够显式地改变HTML静态资源的展示情况；
2. 在一定情况下，静态资源会将网页操作信息传递到Javascript动态控制器中，使得Javascript代码能够对其进行操作；
3. Javascript动态控制器能够解析网页操作的信息，并将之转化为对应的HTTP请求发往服务器；
4. Javascript动态控制器也能够自行处理部分网页操作信息，并直接返回响应信息，通常用于错误处理、格式处理等操作；
5. Javascript动态控制器接收服务器发送的HTTP响应，该响应包含网页跳转信息、网页静态资源、网页动态资源等多种信息。Javascript对于此类响应做出对应的操作，改变HTML静态资源的状态。
6. HTML静态资源的不同展示，使得用户能够感受到网页的响应情况，并从网页的变化中获得自己想要的信息，从而使得HTML静态资源成功反馈网页响应信息。

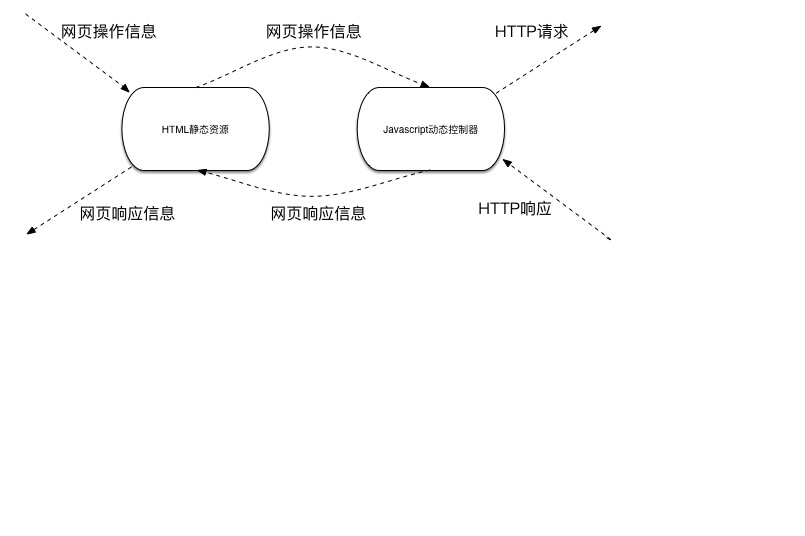


图 3-2 前端子系统数据流图

### 后台子系统数据流图

后台子系统分为多个模块，多数模块由Django自动调用协调。下图展示了部分主要模块之间的数据传递过程。

1. 浏览器发送HTTP请求，被url管理模块接收到，url管理模块调用相应的业务逻辑函数来处理该HTTP请求；
2. 业务逻辑函数中包含多个函数：部分函数能够调用数据库模块对数据库进行操作，读取和更新数据信息到数据库中；部分函数能够读取文件系统中的数据文件，并将其内容转化为HTTP响应，返回给浏览器；部分函数能够传递信息给抓取子系统，启动网络爬虫；
3. 业务逻辑函数能够返回给浏览器相应的HTTP响应，这些HTTP响应包含多种信息。

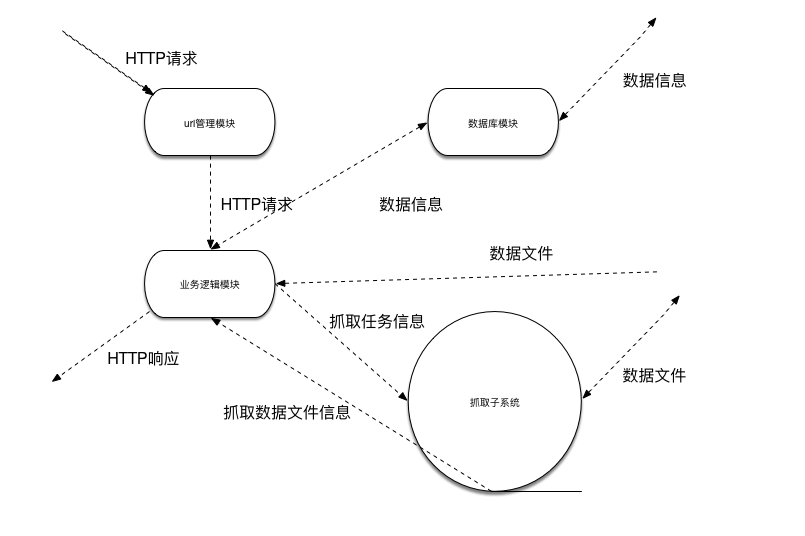


图 3-3 后台子系统数据流图

### 抓取子系统数据流图

抓取子系统由多个子模块构成，其核心部件是抓取算法。

1. 获得从后端传递过来的任务信息，如抓取网页、抓取内容等。然后通过Requests库，获得来自网页的源代码，并调用BeautifulSoup库将网页源代码解析成Dom树；
2. 将Dom树和抓取内容等传给抓取算法，抓取算法的就能够获得含有可信数据的Dom节点列表；抓取算法将在第四章详细说明。
3. 将含有可信数据的Dom节点进行分类，输出到文件系统的数据文件中，同时返回给解析任务模块数据文件信息；
4. 解析任务模块最后返回抓取数据文件信息给业务逻辑模块，完成抓取过程。

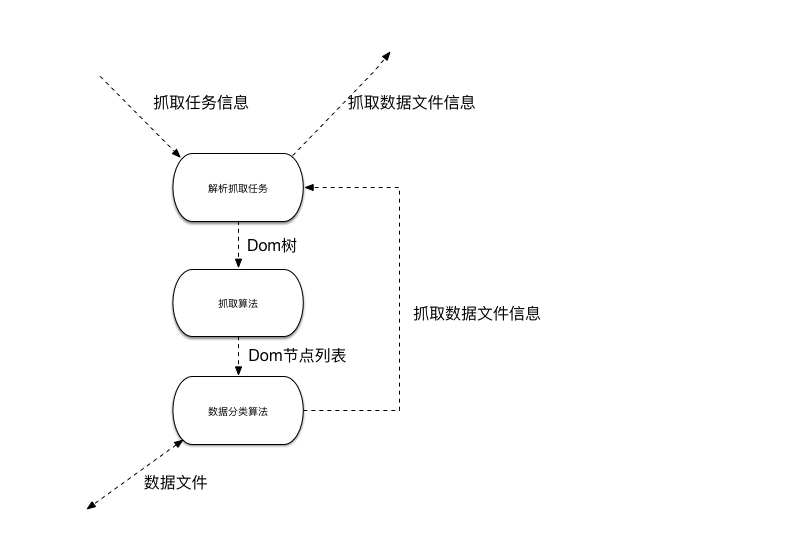


图 3-4 抓取子系统数据流图

## 模块独立性

模块化的设计是概要设计最主要的工作。模块的划分应遵循一定的要求和一些软件设计和开发的基本原则，以保证模块划分合理。

下图去除了数据处理部分的结构图，体现了各个模块之间数据传递和调用关系。借此我们来分析各个模块之间的耦合性以及模块自己的聚合度问题，并进一步证明以此为依据开发出的系统可靠性强，易于理解和维护。

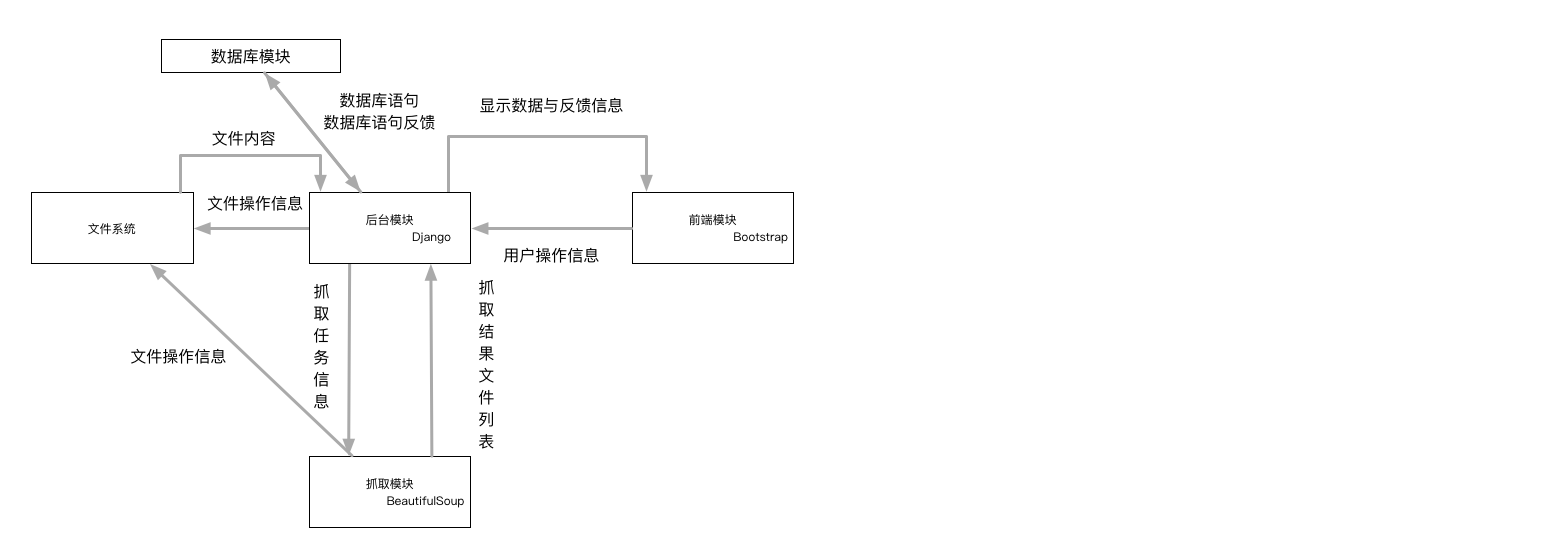


图 3-5 系统结构图

### 内聚度分析

抓取模块：功能内聚，模块内所有函数和子模块仅仅从后台模块接收抓取任务的信息（url、抓取内容），然后对文件系统进行一系列的操作，返回出存储数据的文件名称。

抓取子模块：顺序内聚，通过将爬虫抓取信息的步骤进行分割，分割成了多个不同的步骤，每个步骤又形成相应的模块，进行顺序执行。

后台模块：偶然内聚，包含了所有后台的子模块，总体内聚度低，能够处理所有后台所需要的逻辑，并启动整个Web服务。

后台子模块：每个单独的子模块属于功能内聚。如，models子模块完成对数据库的查询和管理功能、views子模块完成对前端反馈信息的处理和部分后端逻辑的处理等。

前端模块：功能内聚，从后端获得数据并对其进行渲染，形成可视的图形化界面并能够和用户进行交互。

### 耦合度分析

将整个系统按大模块分开，从模块外部考察模块的独立性程度。模块之间的耦合度如下图所示

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 前端模块 | 后台模块 | 抓取模块 | 文件系统 | 数据库 |
| 前端模块 |  | 数据耦合 | 独立耦合 | 公共耦合 | 独立耦合 |
| 后台模块 |  |  | 控制耦合 | 控制耦合 | 控制耦合 |
| 抓取模块 |  |  |  | 控制耦合 | 独立耦合 |
| 文件系统 |  |  |  |  | 独立耦合 |
| 数据库 |  |  |  |  |  |

表 3-1 耦合度表

前端模块对于文件系统只具有读取的功能，并无修改功能，但是因为能够直接访问文件系统，所以是公共耦合。且文件系统是由操作系统控制的模块，此项耦合并不影响模块的独立性。

### 独立性分析

系统的每个特定的模块内聚度较高，模块之间的耦合度较低，有着良好的模块化设计。因为各个模块的独立性较高，所以系统有着完整单一的功能划分，函数接口易于理解和整理，代码可读性高；各个模块以及其子模块能较好地满足信息局部化的原则，使得代码复用、重构开发、增加新功能等较为容易。

## 数据库设计

### 概念结构设计

* ER图：

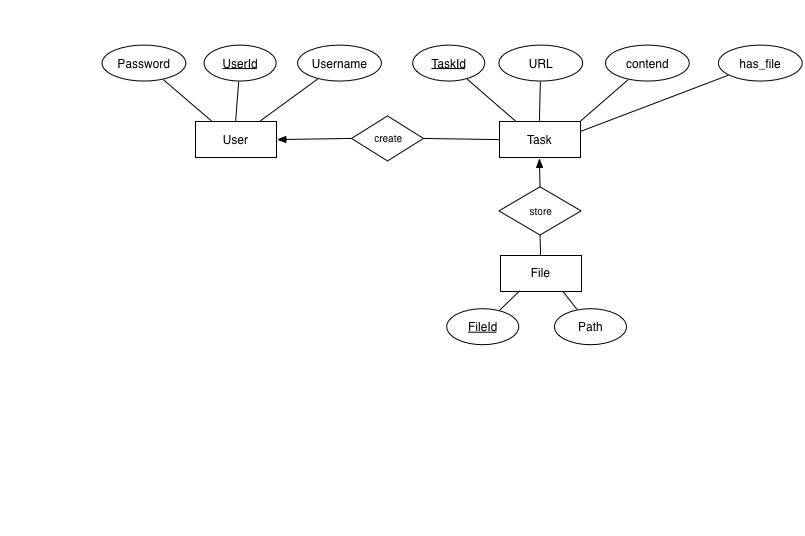


图 3-6 数据库ER图

* User实体：

在设计本毕设系统的时候，考虑到Web应用中用户使用、用户权限管理等特有的情况，需要专门的User数据集存储用户相关内容。

* Task实体：

本毕设的核心功能为爬虫抓取模块，必须利用设计Task数据集来记录爬虫任务的相关信息，使得模块的独立性更强、代码的逻辑更加清晰，方便后台业务逻辑模块的存储、查询、更新数据库操作；同时也使得开发人员、管理员能够更加方便地追查信息、调试维护程序。

Task数据集和User数据集的关系也使得后台程序能够更加快速地获得相应用户创建的爬虫任务，利用空间换取时间。

* File实体：

本毕设利用关系型数据库无法直接存储爬虫抓取的网页源代码或者网页内的可信数据，所以需要用File数据集记录每个Task实体抓取的文件情况。

同时，使用了File数据集后，开发人员不需要内嵌代码逻辑，不需要创建新的无关逻辑模块，直接利用Task数据集与File数据集一对多的关系，获取File数据集的记录就能够得到数据文件的存储路径，后台逻辑业务模块能够方便地从中读取信息。

此点设计虽然可能造成一定的数据冗余，但是更能够使得代码模块独立性提高，代码的可读性、逻辑性、鲁棒性提高，代码运算查询时间下降。

### 逻辑结构设计

将上文的ER图转化为具体的数据模型和数据类型。

* User

记录用户的所有信息，包括用户ID、用户的权限、用户的组、是否为超级用户、用户名、用户密码、用户昵称、用户邮箱等。

这些属性的数据格式均采用可变长度的限长字符串。

其中有多个信息是无需开发人员和系统管理直接操作，由Django自带的数据库模块自主完成，如用户ID、用户密码的哈希加密、用户权限的多对多关系等。

* Task

记录与爬虫任务有关的信息，包括任务ID、任务抓取地址、任务抓取信息、创建任务的用户ID、是否有数据文件、数据文件的抓取方式。

是否有数据文件、数据文件的抓取方式，这两种属性的数据格式采用布尔类型。前者的默认值为False，后者的默认值为True。

其余属性的数据格式采用可变长度的限长字符串。

Task与User的对应关系为多对一，则将User的主键（用户ID）当做Task的外键。数据格式和用户ID的数据格式一致。

其中有多个信息是无需开发人员和系统管理直接操作，由Django自带的数据库模块自主完成，如任务ID、Task与User的多对一关系等。

* File

记录与数据文件相关的信息，包括文件ID、文件路径、文件所属抓取任务ID。

这些属性的数据格式均采用可变长度的限长字符串。

File与Task的对应关系为多对一，则将Task的主键（任务ID）当做File的外键。数据格式和任务ID的数据格式一致。

## 本章小结

本章主要从三个方面对系统结构作了介绍。首先，本章介绍了系统工程的目录结构，包括目录文件的分布情况，每个文件实现的功能；其次，本章分析了系统数据流动和处理的情况，绘制了数据流图；再次，本章分析了各个系统模块的内聚度以及它们之间的耦合度问题；最后，本章分析了系统使用的数据库设计情况，绘制了ER图并说明了数据库的逻辑实现。

# 抓取算法设计



## 定义

对算法中使用的名词与该名词对应的对象进行定义和说明，方便理解后文中算法设计与分析。

### 名词定义

* 抓取网址：由用户输入的需要抓取的网址。
* 抓取内容：由用户输入的需要抓取的内容实现与代码。
* 网页源码：网页前端代码，除去Javascript和CSS的style部分。

<html>

<head>

<title>

My title

</title>

</head>

<body>

<a href="">

My link

</a>

<h1>

My header

</h1>

<div>

<p>This is the first block. </p>

<p>This is the second block. </p>

<p>This is the third block. </p>

</div>

</body>

</html>

* + Dom树：HTML代码的语法树是一棵多叉树，并可以结构化成拥有唯一根节点<html>的Dom树。

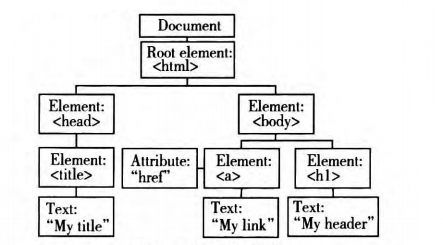


图 4-1 Dom树

* 节点：在HTML中为某一个具体的标签，如<head>
* 叶子节点：在HTML中为某一个具体的字符串，如My title
* 路径：从根节点到某个节点经过的各个节点，如“This is the first block”的路径为，*P*1={<html>, <body>, <div>, <p>, This is the first block}
* 锚：顺序包含抓取内容各个字符的叶子节点，可能有多个。
* 标识祖先节点：对于某个锚节点，其深度最深的具有CSS样式的祖先节点，称为该锚节点的标识祖先节点。因为锚节点是叶子节点，叶子节点只包含字符串，不含有HTML标签，也不可能有CSS样式，所以标识节点必是锚节点的祖先节点。
* 路径相似度：用评估函数预估出两个路径的相似度，取值为[0, 1]。
* 列表类网页：在网页中存在大量相似结构的少量文本模块，节点数多，但是叶子节点的字符串长度较小。
* 新闻类网页：在网页中存在一块多量文本模块与多块是少量文本模块，节点数较少，但是部分叶子节点的字符串长度大。
* 复杂度分析定义：

1. N : Dom树的节点的总个数
2. n : 叶子节点个数，且n<N
3. m :叶子节点字符串的最大长度
4. le : 抓取内容的长度

### 对象定义

算法中可能用到的对象定义。在算法伪代码没有说明的情况下，下列对象的默认定义。

* contend：抓取内容，str对象，长度上限为128
* url：抓取网址，str对象
* BeautifulSoup：Dom树，由BeautifulSoup对HTML源码进行结构化分析而得到的BeautifulSoup对象
* Node：节点，对BeautifulSoup中的Tag对象进行了封装后得到，包括该节点的标识祖先节点（father属性）、该节点的路径（path属性）。
* Leaves：叶子节点，为BeautifulSoup中的NavigableString对象
* anchor：包含所有锚的list对象
* node：当前节点，Node对象

## 算法分析

该算法是3.1.4中抓取子系统中的抓取算法部分的详细说明。抓取算法按照2.3.1中爬虫的抓取步骤②和③分为以下几步具体实现

1. 定位抓取内容样式
2. 选取页面内所有的有效信息
3. 将有效信息进行整理与分类

算法流程总览如下图所示：

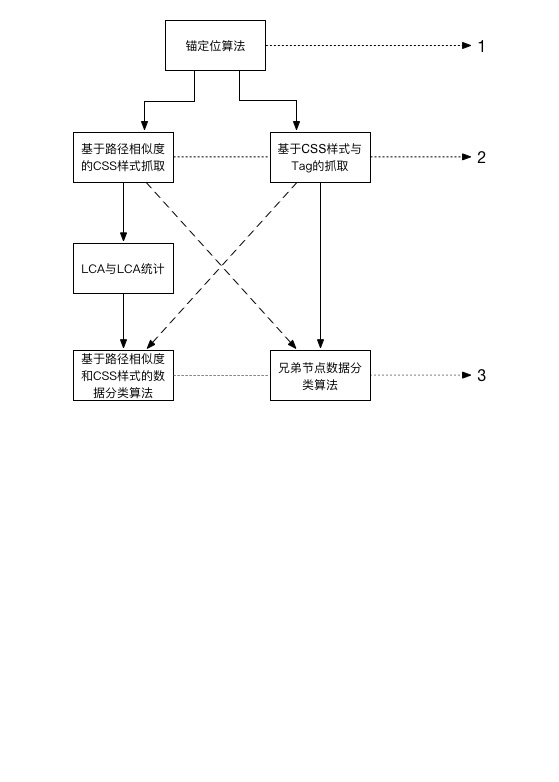


图 4-2 算法总览

### 锚定位算法

* 功能：选取出所有的锚
* 返回：None，通过传引用来获得锚列表
* 过程描述：

1. 用深度优先搜索搜索整个Dom树，并找出每一个叶子节点
2. 判断叶子节点的字符串是否顺序包含抓取内容的每个字符
3. 若是，则将此叶子节点加入anchor列表

* 伪代码：

get\_anchor\_pos(node, anchor):

if node.children == 0: # 没有孩子节点，则是叶子节点

if substringcmp(node.string, contend): # 查看是否满足顺序包含子串字符的条件

anchor.append(node) # 将node加入anchor

else:

for child in node.children:

get\_anchor\_pos(child)

* 时间复杂度分析：

1. 搜索所有Dom节点的时间复杂度为。
2. 单次比较字符的最坏时间复杂度为。虽然有剪枝操作，但是仍然有较大概率会达到最坏情况，而平均情况与网页有关，无法估计；因为用户输入的抓取内容长度有限，为常数项，所以此项的时间复杂度为O(m)。
3. 总的时间复杂度为。

由于新闻类网页的m常常会较大，此算法在列表类网页表现较为迅速，不善于处理新闻类网页。

* 空间复杂度：

1. 此算法需要存储一棵完整的Dom树，需要O(N)的空间。
2. 此算法需要存储所有的锚节点，且锚节点的上限为O(n)，所以需要O(n)的空间。
3. 总的空间复杂度为O(N+n)。

### 基于CSS样式与tag的抓取

此算法由两个部分功能构成，所以用两个函数分别完成。

**函数1：**

* 功能：对于每一个锚节点，找到所有对应的标识祖先节点。
* 返回：元组对象的列表，该元组由标识祖先节点的HTML标签类型和该节点的CSS样式类型构成。
* 思路：

1. 利用Beautilfulsoup提供的反向链表，从锚节点往根追溯，找到第一个有CSS样式的节点，作为锚节点的标识祖先节点，取出该节点的HTML标签类型（字符串对象）和该节点的CSS样式类型（字典对象）。
2. 对所有的标识祖先节点进行去重，因为有可能多个锚节点会找到的相同的标识祖先节点，对标识祖先节点进行去重之后可以防止数据冗余。

* 伪代码：

node # 一个锚节点

get\_father\_css(node):

for parent in node.parents:

# 利用Beautilfulsoup提供的反向链表反向追溯到父亲节点

if len(parent.tag) != 0 AND ‘class’ in parent.attrs:

# 若找到了第一个有CSS样式的节点，则取出HTML标签和CSS样式

node.father = make\_pair(parent.tag, parent.attrs[‘class’])

break

* 时间复杂度：

1. 反向追溯的最坏时间复杂度为Dom树的深度，即，但是有较大概率不会达到最坏情况，平均的时间复杂度为常数级别。
2. 对于每一个锚节点都需要进行这样的操作，所以最坏的时间复杂度为O(N)。
3. 总的时间复杂度为O(NlogN)。

* 空间复杂度：

1. 此算法需要存储所有的锚节点，且锚节点的上限为O(n)，所以需要O(n)的空间。每个锚节点又附带常数级别的存储量。
2. 总的空间复杂度为O(n)。

**函数2：**

* 功能：对于每一个标识祖先节点，找到Dom树上所有与之有相同HTML标签类型和相同CSS样式的节点。
* 返回：Dom节点对象列表。
* 思路：

1. 对于每一个标识祖先节点，从Dom树的根节点深度优先搜索。
2. 通过精确匹配来判断当前节点的tag和CSS是否与标识祖先节点的完全相同
3. 若完全相同，则将当前节点添加到传引用的列表中。
4. 将所有符合条件的Dom节点返回。

* 伪代码：

iNode # 标识祖先节点，全局变量，也可作为不可变参数传入

list # 通过传引用来直接修改参数，符合条件的Dom节点

get\_all\_fit\_css(node, list):

if node.tag == iNode.tag AND node.class == iNode.class:

# 若此节点的HTML标签类型与标识祖先节点相同

# 且此节点的CSS样式与标识祖先节点相同

# 则将此节点认定为符合要求的节点

list.append(node)

if node.children == 0: # 没有孩子节点，则返回

return

else:

for child in node.children:

get\_all\_fit\_css (child)

* 时间复杂度：

1. 搜索所有Dom节点的时间复杂度为。
2. 总的时间复杂度为。

* 空间复杂度：

1. 利用全局变量存储所有Dom树，不需要额外计算空间。
2. 存储全部切合要求的Dom节点，需要O(N)的空间。

### 兄弟节点分类法

* 功能：将从抓取算法得来的所有Dom节点分类，并将分类结果输出到CSV文件中
* 返回：无返回值
* 思路：

1. 以Dom节点为根dfs搜索整个Dom子树，通过判断现在所在的节点有没有孙子节点，即判断它们的儿子节点是否有儿子节点，则可以找到所有叶子节点的父亲节点。
2. 以深度对父亲节点进行分类，即相同深度的叶子节点的父亲节点将在同一个CSV文件中。
3. 将每一个父亲节点所有的儿子节点，即叶子节点，转化为一个字符串列表。
4. 该父亲节点相应的CSV文件中的最后一行添加将此列表的信息。

* 伪代码：

Node # 当前节点

deps # 当前节点深度

print\_tree(Node, deps):

for child in Node.children:

if len(child.children) != 0:

# 有孙子节点，则继续

print\_tree(child, deps+1)

else:

# 没有孙子节点，则把所有子节点都打印转化为一行

for brother in child.next:

line.append(brother.string)

# 以深度分类CSV文件，将所有子节点输出到该CSV文件的最后一行

print\_csv(csv\_id = deps, csv\_line = line)

* 时间复杂度分析：

1. 搜索所有的Dom节点，时间复杂度为O(N)。
2. 将全数切合要求的叶子节点，按次输入到CSV文件中，此步骤需要遍历叶子节点，则最坏时间复杂度为O(n)。
3. 总的时间复杂度为O(N+n)。

* 空间复杂度分析：

1. 局部存储叶子结点的情况，最坏的空间复杂度为O(n)。
2. 存储仅为局部变量，所需常数复杂度的空间。
3. 总的空间复杂度低，很大可能为常数复杂度，最坏情况为O(n)。

### 基于路径相似度与CSS样式的抓取

此算法由两个函数构成。

函数1：

* 功能：对于每一个锚节点，找到所有对应的标识祖先节点，并将之转化成Node对象。
* 返回：返回Node对象列表。
* 思路：

1. 利用Beautilfulsoup提供的反向链表，从锚节点往根方向追溯，找到第一个有CSS样式的节点，作为锚节点的标识祖先节点，取出其HTML标签类型和该节点的CSS样式类型。
2. 追溯到根节点，获得该标识祖先节点的路径，该路径是一个列表，按顺序记录的根节点到锚节点经过的每一个Dom节点的HTML标签类型。
3. 将以上两个变量以及Dom节点对象封装成Node对象，并存入列表。

* 伪代码：

node # 一个锚节点

get\_father\_css(node):

path = []

flag = 0

for parent in node.parents:

if flag == 1 AND len(parent.tag) != 0 AND ‘class’ in parent.attrs:

node.father = make\_pair(parent.tag, parent.attrs[‘class’])

flag = 1

else:

# 将路径按照倒叙插入路径列表中

path.append(parent)

# 将路径翻转，赋予锚节点

node.path = reversed(path)

* 时间复杂度：

1. 运行反向追溯的时间仅与Dom树的深度有关，即。
2. 对于每一个锚节点都须要进行这样的操作，所以最坏的时间复杂度为O(N)。
3. 总的时间复杂度为O(NlogN)。

* 空间复杂度：

1. 存储路径的空间固定为Dom树的深度，即。
2. 对于每一个锚节点都须要存储此类数据结构，总空间复杂度为O(NlogN)。

函数2：

* 功能：对于每一个Node对象，找到所有与其路径相似度超过阈值的Dom节点
* 返回：None，通过传引用来获得锚列表
* 思路：

1. 路径相似度函数用于评估路径的相似度，在此算法中，只有当路径列表完全一致，且标识祖先节点CSS样式完全相同的节点，才为相似。
2. 每一个Node节点对象都当做参数传入深度优先搜索，以此来搜索整棵Dom树，找到所有与Node节点相似的Dom节点。
3. 将所有符合条件的Dom节点返回。

* 路径评估函数伪代码：

nodex # 一个锚节点

nodey # 当前节点

judge\_path(nodex, nodey):

# 精确匹配，完全相同

if nodex.path == nodey.path AND nodex.father == nodey.father:

return true

return false

* 伪代码：

iNode # 一个锚节点，全局变量，也作为不可变参数传入

list # 通过传引用来直接修改参数，符合条件的Dom节点

get\_all\_fit\_path(node, list):

if judge\_path(iNode, node):

list.append(node)

if node.children == 0: # 没有孩子节点，则返回

return

else:

for child in node.children:

get\_all\_fit\_path(child)

* 时间复杂度：

1. 搜索所有的Dom节点，时间复杂度为O(N)。
2. 总的时间复杂度为O(N)。

* 空间复杂度：

1. 本算法须要的空间均在其他算法中计算过，无需重复计算。

### LCA与LCA统计

* 功能：针对“基于路径相似度与CSS样式的抓取”算法，找到所有符合Dom节点的最可能总模块位置。
* 返回：作为LCA的Node节点
* 思路：

1. 对锚节点与根据“基于路径相似度与CSS样式的抓取”得出的Dom节点分别求LCA。
   1. 将比较锚节点和另一个Dom节点的深度，并将深度较大的那一个节点抬高到与另一个相同的深度；
   2. 两个节点在相同的深度之后，同时向上抬升，直到两个节点相同为止；
   3. 则停止的节点便是LCA。
2. 统计每一个LCA出现的频度，找出频度最大的LCA，则认定其为所有Dom节点最可能模块的总节点
3. 返回作为LCA的总Node节点。

* LCA伪代码：

nodex # 锚节点

nodey # 符合条件的Dom节点

def LCA(nodex, nodey):

# 比较节点的深度

xdep = nodex.get\_dep()

ydep = nodey.get\_dep()

if xdep > ydep:

swap(nodex, nodey)

# 将较深的节点抬高到较浅节点的高度

dist = ydep – xdep

while dist != 0:

nodey = nodey.parent

dist -= 1

# 两个节点同时向上抬高，直到两个节点相同

while nodex!= nodey:

nodex = nodex.parent

nodey = nodey.parent

return Node(nodex)

* LCA频度统计伪代码：

list #所有符合条件的Dom节点

anchor\_item #一个锚节点

def get\_all\_fit\_path(list, anchor\_item):

max\_cnt = 1

lca = anchor\_item

for item in li:

# 对于每一个符合条件的Dom节点，求其与锚节点的LCA

lcanode = LCA(item, anchor\_item)

# 统计LCA的频度

cnt\_node[lcanode] += 1

# 选出频度最大的LCA，最为总节点

if cnt\_node[lcanode] > max\_cnt:

max\_cnt = cnt\_node[lcanode]

lca = lcanode

return lca

* 时间复杂度

1. LCA的时间复杂度为。
2. 遍历每一个符合条件的Dom节点，须要的时间复杂度为O(N)。
3. 总的时间复杂度为O(NlogN)。

* 空间复杂度

1. LCA函数中只须要常数复杂度的局部空间，无需计算。
2. 所有符合条件的Dom节点列表，在另外的算法中已经计算过空间，在此处为传引用，不占用空间。
3. 通过字典统计每一个LCA的频度，须要O(n)的空间。
4. 总的空间复杂度为O(n)。

### 基于路径相似度与CSS样式的数据分类算法

* 功能：以得到的模块总节点为根节点，对其子树进行数据分类，并将分类后的数据输出到CSV文件中
* 返回：None，通过传引用直接修改类别字典，以每一个节点的path和father为key，其分类为value构成一个分类字典。
* 思路：

1. 用dfs搜索总节点下的子树，并根据每个叶子节点的路径与CSS样式进行分类。
2. 将同一类的叶子节点输出到CSV文件的同一列。

* 伪代码：

path # 递归记录路径

lastclass # 递归记录标识祖先节点的CSS样式

dic\_value # 分类信息记录

max\_cnt # 其他信息记录，包括类别总数，类别最长数据数

def split\_path\_css(node, path, lastclass, dic\_value, max\_cnt):

if 'class' in node.attrs:

# 通过递归搜索，记录最后一个CSS样式，作为标识祖先节点的CS样式，用于CSS分类

lastclass = node.attrs['class']

for child in node.children:

if child.name:

# 规则的Dom节点，记录每一个节点的路径，用于路径分类

path.append(child.name)

split\_path\_css(child, path, lastclass, dic\_key, dic\_value, max\_cnt)

path.pop()

else:

# 部分不规则的Dom节点，通过异常处理能够提高容错率

split\_path\_css(child, path, lastclass, dic\_key, dic\_value, max\_cnt)

# 叶子节点，进行分类

if len(node.children) == 0:

# 判断节点的合法性，即，不是空字符串或者非法字符

if node in illegal:

return

key = change\_to\_key(path, lastclass))

# 判断是否有此分类

if not key in dic\_key:

# 创建新的path、CSS类别，并记录类别数

dic\_key[key] = max\_cnt[0]

max\_cnt[0] += 1

# 将信息添加到该类别的信息中，并记录最长的分类

dic\_value[key] = dic\_value[dic\_key[key]] + [tmp]

max\_cnt[1] = max(max\_cnt[1], len(dic\_value[dic\_key[key]]))

* 时间复杂度

1. 遍历每一个节点的最坏时间复杂度为O(N)，平均情况的时间复杂度低于此复杂度。
2. 对于每一个叶子节点，用哈希表实现的字典进行处理，查询字典的复杂度为O(1)，修改的复杂度为O(1)，此项的时间复杂度为O(n)。
3. 总的时间复杂度为O(N+n)。

* 空间复杂度

1. 字典存储的最坏情况须要O(n\*le)的存储空间，平均情况会略小于此复杂度，因此此算法在处理新闻类页面的时候往往会须要更大的内存。
2. 路径须要的存储空间。
3. 其余存储空间均在其他算法中进行过统计，在此不须要计算。
4. 总的空间复杂度为O(n\*le)。

## 本章小结

本章主要分析了爬虫抓取的算法。首先，定义了在后文会提到的名词以及在伪代码说明和时空复杂度计算中会用到的对象，帮助下面说明；其次，介绍了算法总览；最后，分析了爬虫抓取各个步骤的多种算法，包括功能分析、代码实现、时间复杂度、空间复杂度等。

# 系统实现



## 系统工程概述

项目工程由前端部分和后端部分组成，以后端为主体，分割成多个应用文件夹。

### 项目主应用

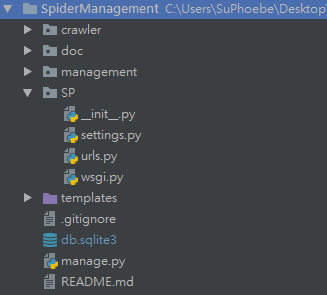


图 5-1 项目主应用文件夹

* 项目主应用

manage.py:系统管理总程序，是控制和管理整个Django项目的命令行工具，通过命令行参数，能够进行运行Django系统、迁移数据库、创造超级管理员等等操作。

SP/settings.py：系统总配置程序，配置了Django项目的读取中间件、读取模板、国际化、数据库配置、应用模块配置等等。

SP/wsgi.py：Web服务器网关接口。

SP/urls.py：项目总URL地址，用于汇总各个应用的URL，使其能够被用户访问。

### 信息管理应用

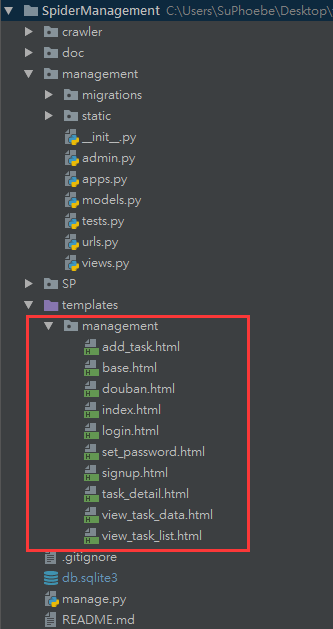
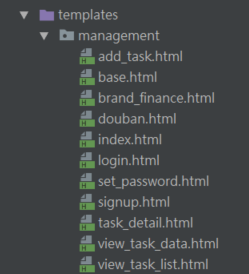
 

图 5-2 信息管理应用文件夹

* management文件夹：

关于信息管理应用的所有业务逻辑代码，包括用户信息管理、用户权限管理、爬虫任务管理、爬虫数据管理、业务功能实现、数据库管理、文件系统管理等。

apps.py：应用模块的配置，使得项目主应用能够识别并调用此应用模块。

admin.py：超级管理员管理功能逻辑设计，在此文件中设置超级管理员对数据库表单操作的权限以及部分管理员后台管理界面的渲染、排列、显示情况等。

models.py：数据库管理相关代码。基于MVC结构，提供了大量经过封装的数据库查询与管理API，能够更加便捷地参与数据库的查询与管理。但是此代码不参与数据库配置情况，而是开发人员通过manage.py和settings.py对数据库进行配置，因此数据库的调试就变得更加简单。

test.py：用于存储Django自动测试的代码，包括单元测试、代码调试、压力测试等等。

urls.py：用于设置系统网址与应该响应的views.py中的函数，将被主项目的urls.py引用，从而将应用设置的url汇总到主项目中。

views.py：业务功能实现。基于MVC结构，是前后端交互的表示层和逻辑层。通过urls.py与前端访问的网址相连，返回一个HTTP响应对象（包括HTML网页、HTML网页信息等）。Django提供了许多便捷的库函数，结合开发人员自己编写的代码，能够实现快速建模与开发，灵活而强大。

* management/static文件夹：

包括了前端框架Bootstrap的所有CSS样式、Javascript代码，也包括了其他静态资源文件，如图片、爬虫抓取数据等。

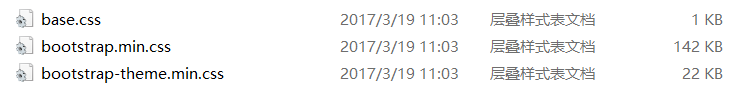




图 5-3 静态资源文件文件夹

* templates/ management文件夹：

整个系统的前端代码汇总。

base.html：前端页面的基础框架，将被其他html文件引用作为模板，其中设计了包括导航栏、页脚说明等部分。

index.html：系统主页，欢迎界面的实现。

login.html：登陆页面，设计了系统用户登录页面，通过Javascript实现了部分前端控制，如，输入密码不能为空，不能输入非法字符等，减轻服务器负担。

signup.html：用户注册页面，设计了系统用户注册的页面，包括输入昵称、用户名、密码、邮箱等信息，通过Javascript实现了部分前端控制，如，输入密码不能为空，两次输入的密码必须相同等，减轻服务器负担。

set\_password.html：重置密码页面，设计了重置密码的页面，通过Javascript实现了部分前端控制，如，输入密码不能为空，两次输入的密码必须相同等，减轻服务器负担。

add\_task.html：添加任务页面，设计了添加爬虫任务信息的页面，包括爬虫任务名、爬虫抓取网址、爬虫抓取内容示例。

view\_task\_list.html：任务列表页面，设计了查看用户自己的爬虫任务列表，显示了包括爬虫名称、爬虫抓取网址、爬虫数据导出按钮。通过Javascript控制是否有爬虫数据，通过Ajax方法异步请求导出爬虫数据，有效地增强服务器的承受力。

view\_task\_detail.html：任务详情页面，设计了查看某个爬虫任务具体信息的页面，包括爬虫的名称、抓取网址、抓取内容、抓取网页内容，以及抓取算法选择、数据查看。数据导出等爬虫相关操作。

view\_task\_data.html：爬虫抓取数据查看页面，设计了查看某个爬虫任务抓取数据文件的页面，进行了更加合理地分割，能够导出单个文件，更加灵活。可供用户选择可信数据。

douban.html：多网页抓取demo。界面提供单选框，选择抓取豆瓣的电影与图书。提供导出数据按钮。

brand\_finance.html：多网页抓取demo，抓取Brand Finance中2017年全球500强、中国500强、全美100强企业的信息。在前端界面上提供导出数据按钮和说明性文字。

### 爬虫抓取应用

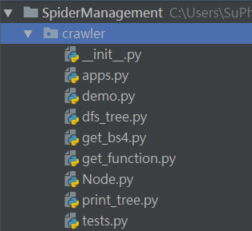


图 5-4 爬虫抓取应用文件夹

apps.py：应用模块的配置，使得项目主应用能够识别并调用此应用模块。

dfs\_tree.py：Dom树搜索算法汇总，包括锚定位算法、基于CSS样式和Tag的抓取算法、基于路径相似度和CSS样式的抓取算法。

demo.py：多网页爬虫豆瓣的抓取demo。

get\_bs4.py：爬虫主函数，获得网络源代码、获得BeautifulSoup的对象、调用抓取算法、调用分类算法，返回抓取结果文件。

Node.py：由我封装的Dom树节点对象，以BeautifulSoup的Tag对象为主，封装了针对本毕抓取算法的函数。

print\_tree.py：数据分类算法，包括兄弟节点分类法、基于路径相似度和CSS样式的分类法，将从dfs\_tree.py抓取的信息输出到CSV文件中。

## 用户功能

下文提供了系统运行时的演示和功能分析，说明了前端和后端的交互的内容、前端Javascript代码协助后台业务逻辑模块处理部分操作的说明与分析。

### 注册与登陆

* **注册界面**

通过Javascript代码对用户名、密码等进行格式限制，限制了其输入不能为空也不能太长，不能输入非法字符，并确保两次输入的密码相同。通过前端静态文件的语法限制，能够有效减少浏览器对服务器发起的非法请求，减轻服务器业务逻辑运行的负担。

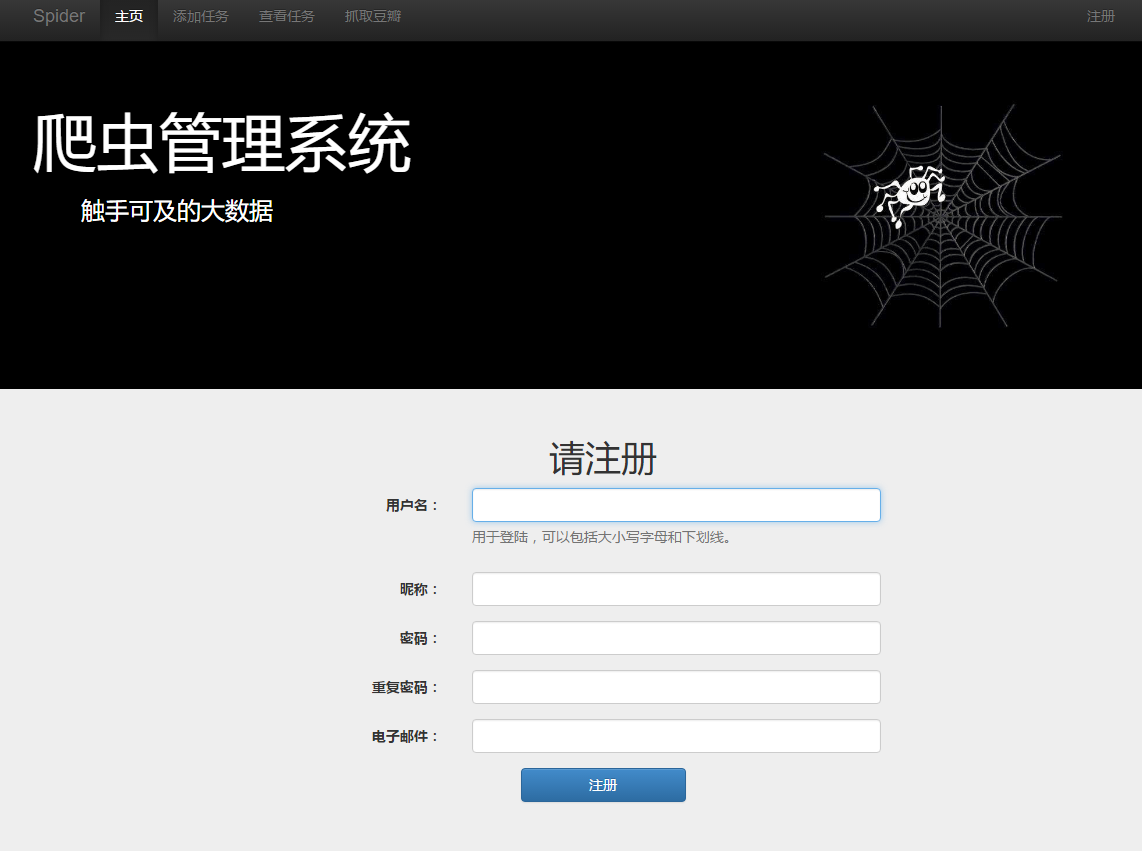


图 5-5 注册页面

* **登陆界面**

网页的所有使用功能均须要验证登陆用户的权限，所以在后台通过权限系统的设置能够检测用户cookie和相应用户的权限。

通过Javascript对用户名、密码等进行格式限制，包括字符串非空、非法字符等限制。

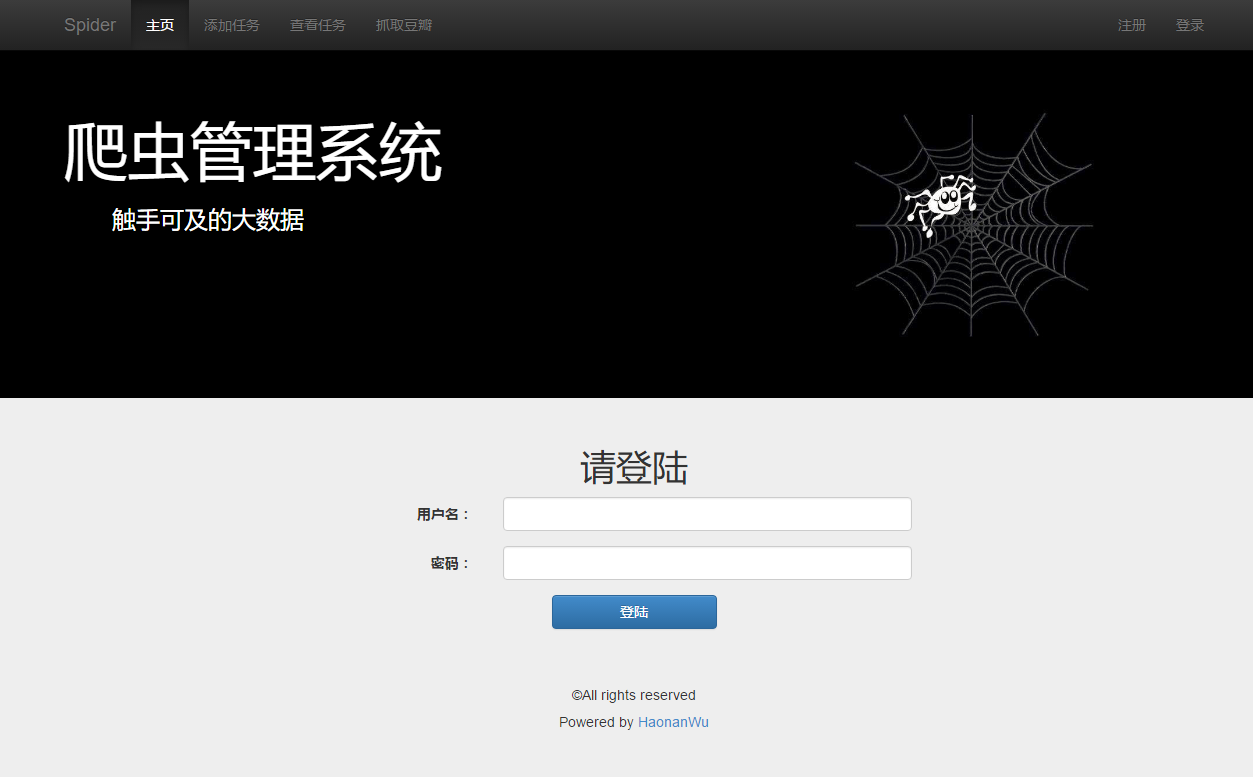


图 5-6 登陆页面

* **修改密码界面**

该网页的使用功能均须要验证登陆用户，所以能够直接获得登陆用户信息和它在数据库中的记录，再通过验证原始密码的合法性，便可进行更新密码。

通过Javascript代码对密码进行格式限制，禁止输入非空，不允许输入非法字符等限制。同时直接在前端匹配新密码和重复密码是否相同，可以减少前后端信息交互，在一定程度上减轻服务器负担。

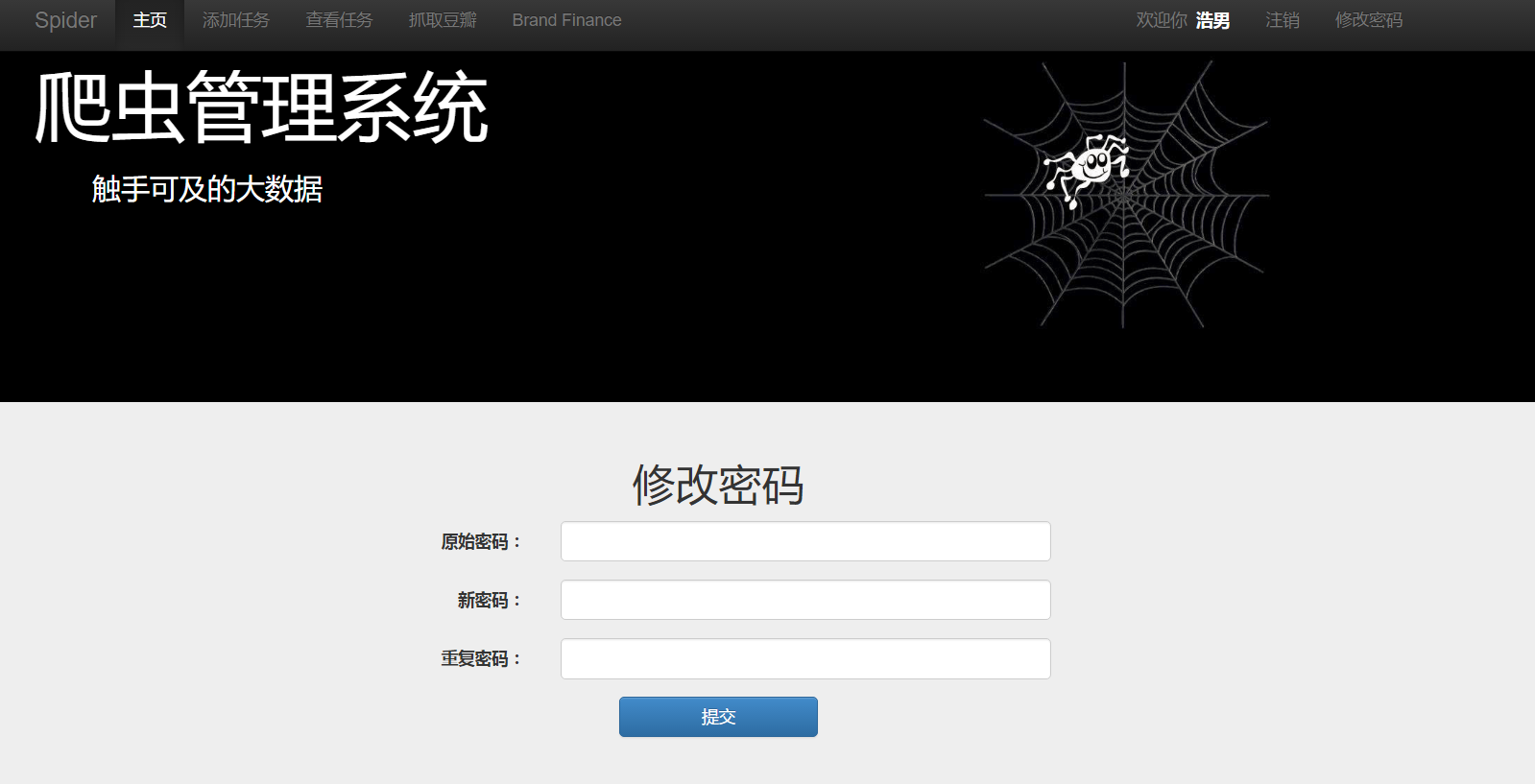


图 5-7 修改密码页面

### 爬虫任务

#### 创建爬虫任务

通过Javascript限制输入的格式，在此页面中仅限制输入内容非空，并未检测输入内容的合法性。

将输入的内容用POST方法转化为字符串传递到后台，后台获得数据之后，通过获得登陆用户的信息，就能够创建一个完整的Task记录，并写入数据库。

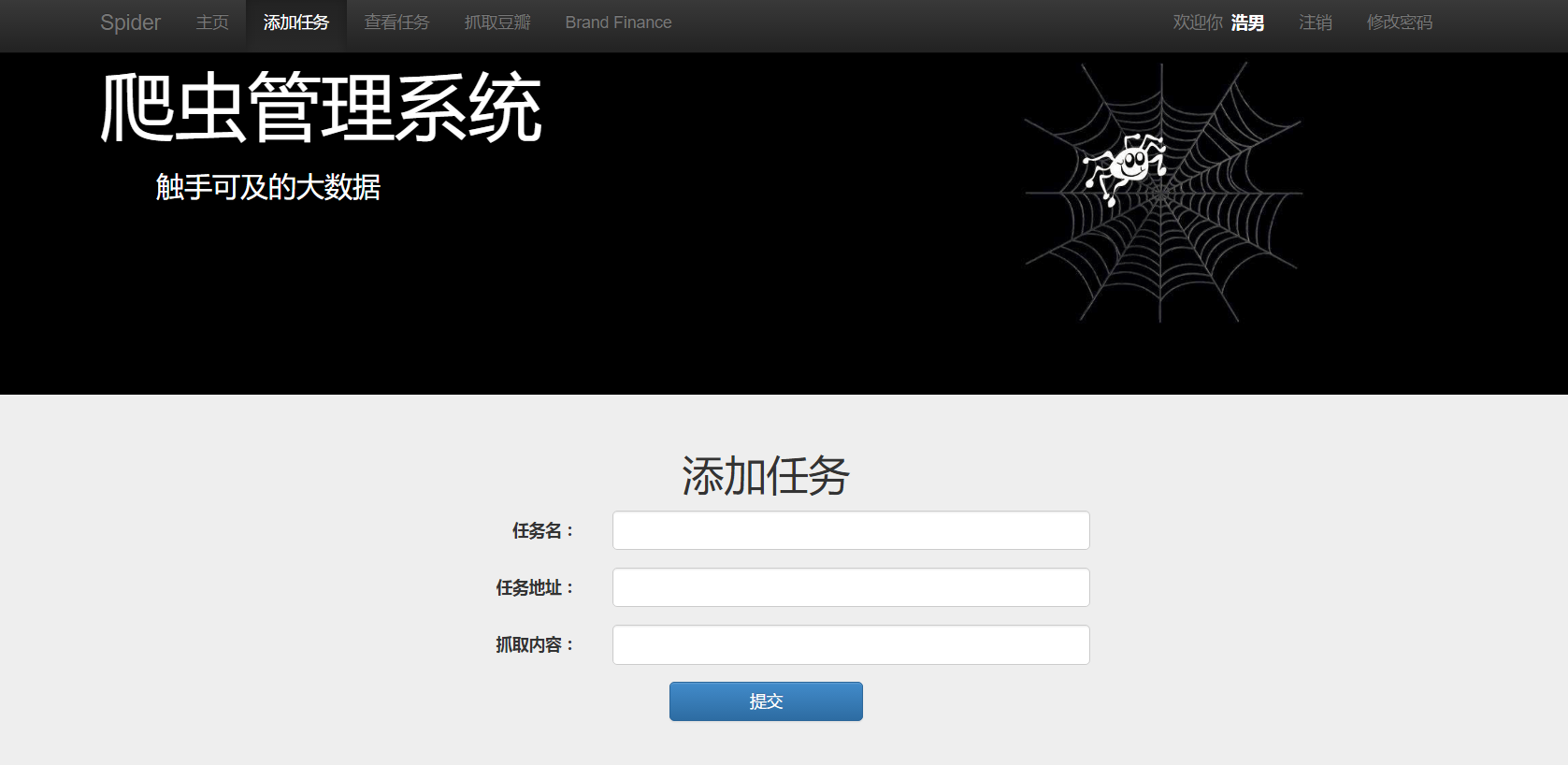


图 5-10 添加任务页面

若后台业务逻辑能够成功写入数据库，则会通过HTTP相应返回一个信息，前端代码通过解析该类型的信息，则可以判断添加成功。

通过Javascript代码提示用户，添加成功，并实现自动跳转的功能。用户也可以手动选择禁止跳转。

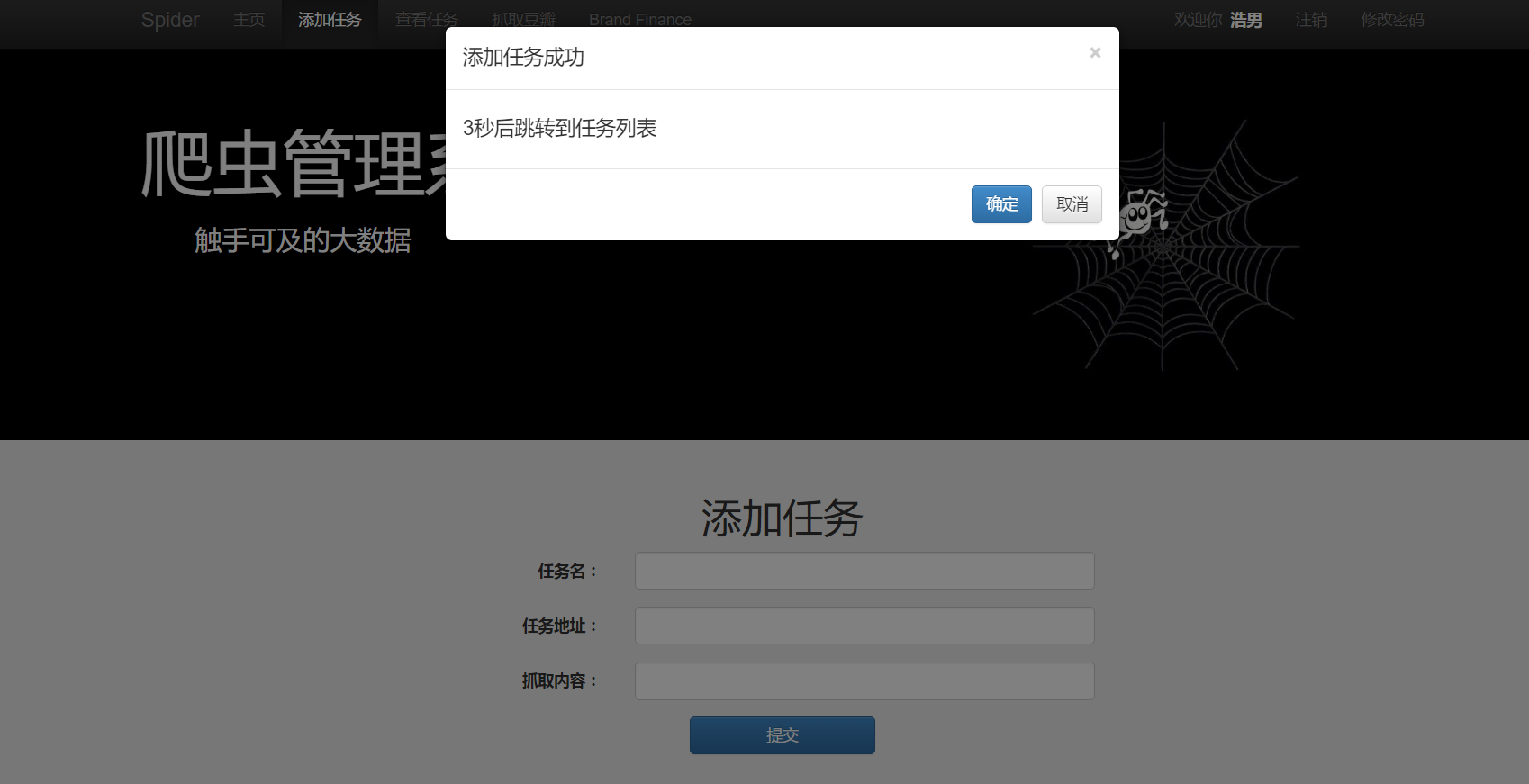


图 5-11 添加任务成功

#### 查看爬虫任务

可以通过导航栏直接访问该用户创建的任务列表，也可以通过添加任务成功之后自动跳转到查看爬虫任务。

业务逻辑模块通过登录用户信息直接获得所以以该登录用户为外键的爬虫任务记录，其中包括任务名、任务地址、抓取内容、是否有数据文件，将之转化为字符串列表后传递到前端进行渲染。

同时对列表进行分页，每一页只显示5条数据。每次点击翻页，服务器再传递后5条数据，此举不仅可以大量减轻服务器负担，也可以让界面更加简洁。

Javascript代码通过判断该任务是否已经完成了抓取并产生了数据文件，对每个任务不同地显示“导出”和“暂无内容”按钮。

导出数据功能见[5.2.3.1导出爬虫数据](#_导出爬虫数据)。

通过点击任务名的链接则可以跳转到该爬虫任务详情，查看具体的爬虫信息。

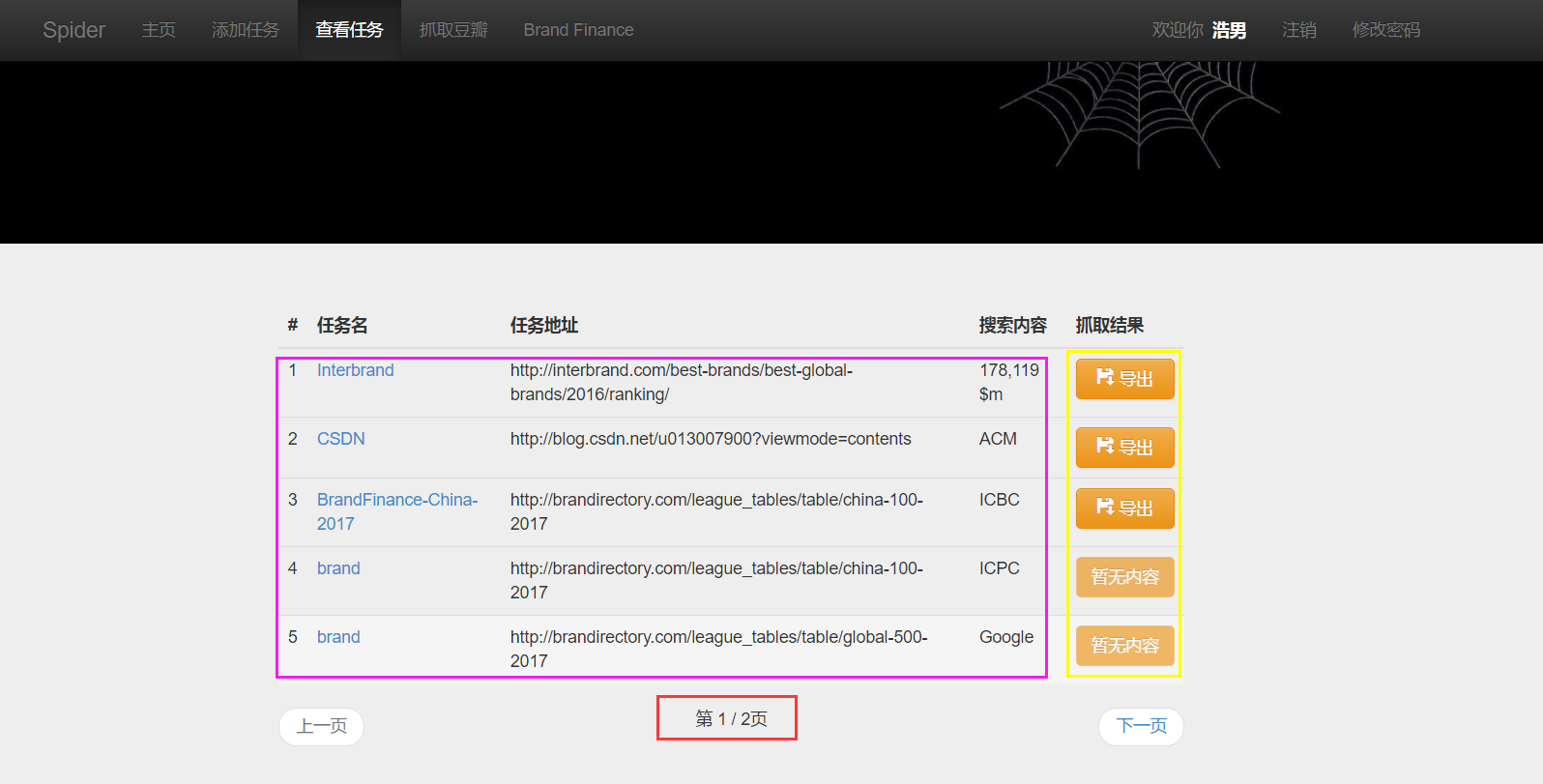


图 5-12 查看任务列表页面

#### 使用爬虫任务

通过查看爬虫任务列表，点击爬虫任务名可以跳转到相关的爬虫任务详情界面。

界面左侧是使用Bootstrap嵌入式网页组件制作的抓取网页的预览。右侧显示的爬虫任务的具体信息，包括爬虫抓取地址、爬虫抓取内容、爬虫抓取方法。

用户通过复选框对爬虫抓取方法进行修改。通过点击抓取按钮，可以进行抓取工作。

这里使用了Javascript代码判断是否有重复用相同方法抓取相同网页的情况。若已经存在了这种抓取方法抓取的数据文件，用户试图再次点击抓取按钮，浏览器直接通过Javascript返回抓取成功的信息，可以极大地减少前后端的交互以及后端抓取模块的负载。

在没有数据文件的情况下，“导出数据”和“查看数据”按钮显示的是暂无内容，同时通过Javascript禁用其功能。



­­图 5-13 任务详情页面

在点击抓取之后，会将任务信息传递回后台业务逻辑模块，业务逻辑模块将任务信息传递到爬虫抓取模块，开始进行抓取。

如果该任务能够被成功抓取，则后台返回给前端“抓取成功”字样并提示用户信息；若抓取失败，会根据后台抓取模块捕获的异常进行反馈，通过前端提示用户任务的失败情况与缘由，如，无抓取内容、网页无法访问、抓取任务出错等。

点击“OK”后页面自动刷新。

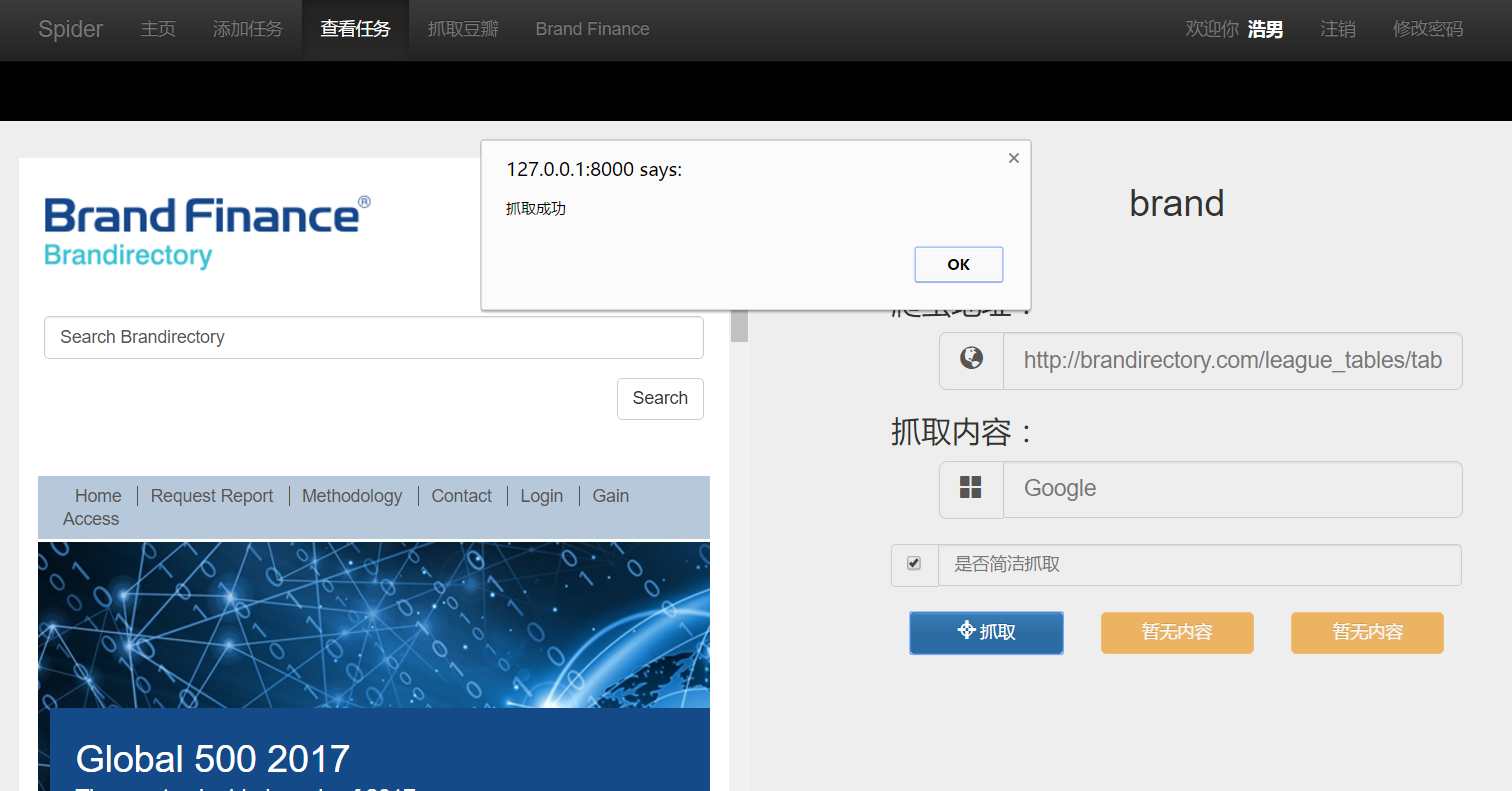


图 5-14 抓取成功

如果该任务被成功抓取且页面刷新之后，则可以看到“导出全部”、“查看数据”按钮，用户可以通过这两个按钮激活相应的Javascript代码，实现对此任务数据的查看、导出功能。



图 5-15 抓取成功后的任务详情界面

### 数据操作

#### 导出爬虫数据

若爬虫任务已经抓取完成，则该爬虫任务的“是否含有数据文件”属性将会为True，则在查看任务列表的页面中，会显示“导出按钮”。

点击“导出”按钮，则会将爬虫任务信息传递到后台的业务逻辑模块，业务逻辑模块调用API操作由操作系统提供的文件管理系统，将多份数据文件进行打包压缩，并将其以静态资源文件的方式发送浏览器，用户就能够下载该爬虫任务抓取的数据文件。

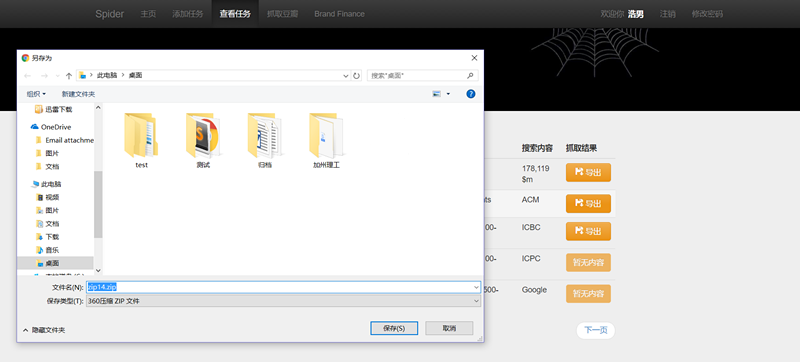


图 5-16 任务列表数据导出

同理，在任务详情界面，若该任务的“是否含有数据文件”属性为True，则会显示“导出全部”，通过点击“导出全部”按钮，调用上文相同函数，使用户能够下载。

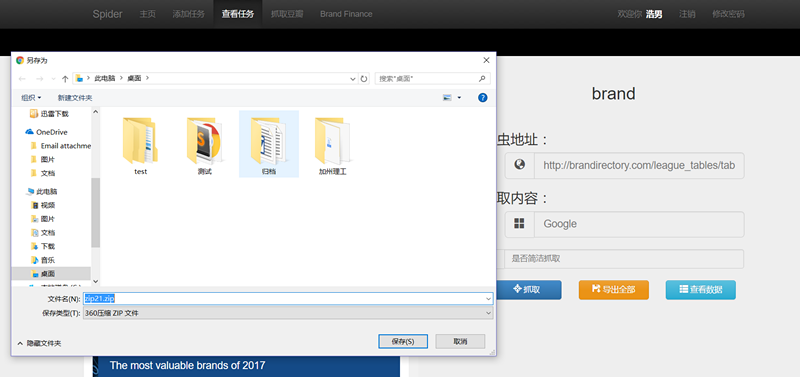


图 5-17 任务详情数据导出

#### 查看爬虫数据

在任务详情界面点击“查看数据”按钮可以跳转到查看该爬虫任务的数据页面，左侧是每一个该任务的数据文件列表，点击每一个文件可以查看相应的数据。

界面的右侧是数据信息列表，通过一定的Javascript和CSS样式限制，提供给用户更加良好的预览体验。

增加数据翻页的特性，使得每一次服务器传输给前端的数据量减少，在后台增加redis缓存机制，使得服务器硬盘IO次数减少，大量降低服务器负载。

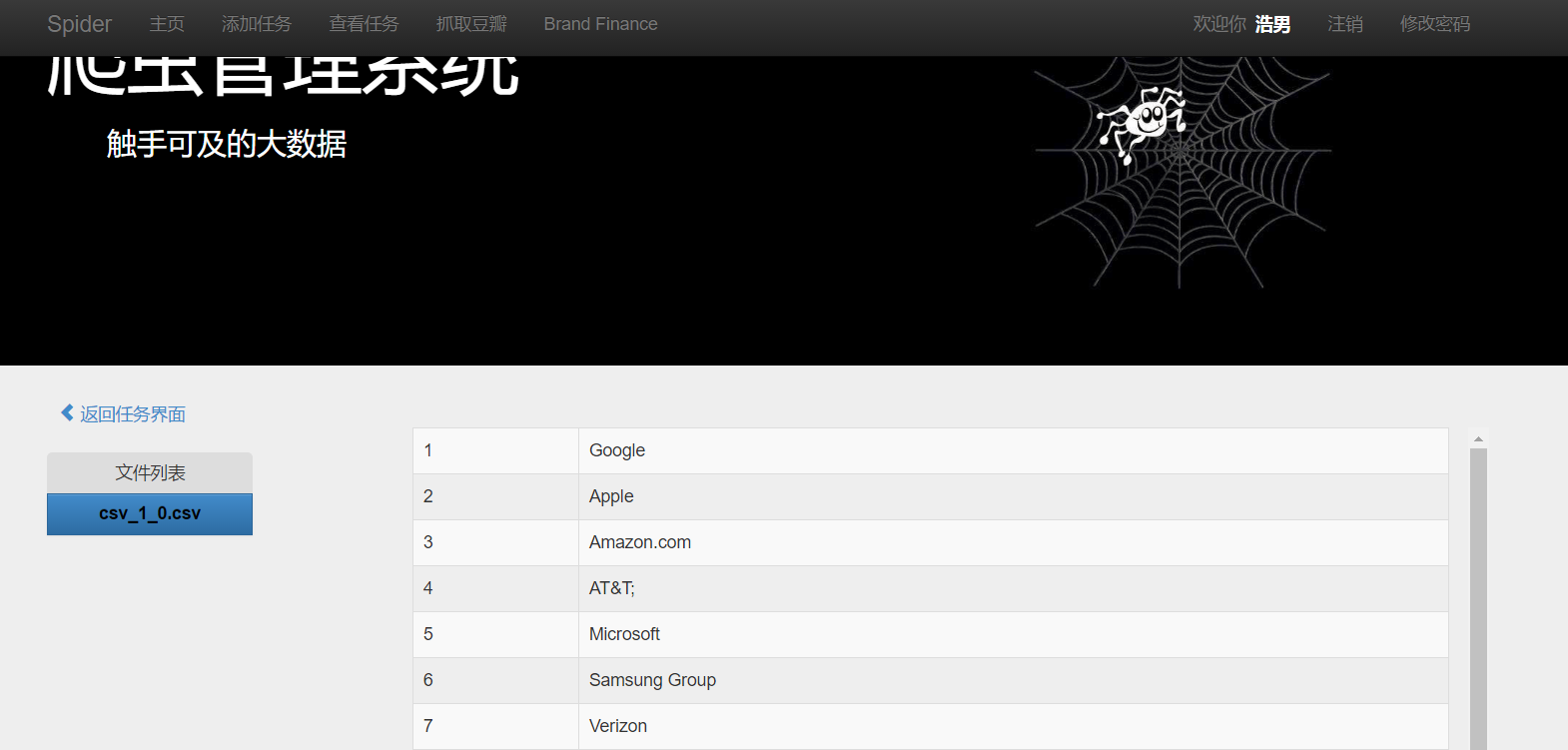


图 5-18 爬虫数据查看

#### 多网站抓取范例

用户点击导航栏的链接，则可以直接进入本系统提供的多网站抓取的范例。本毕设提供了两种多网页抓取的范例，分别是豆瓣信息抓取、Brand Finance抓取。

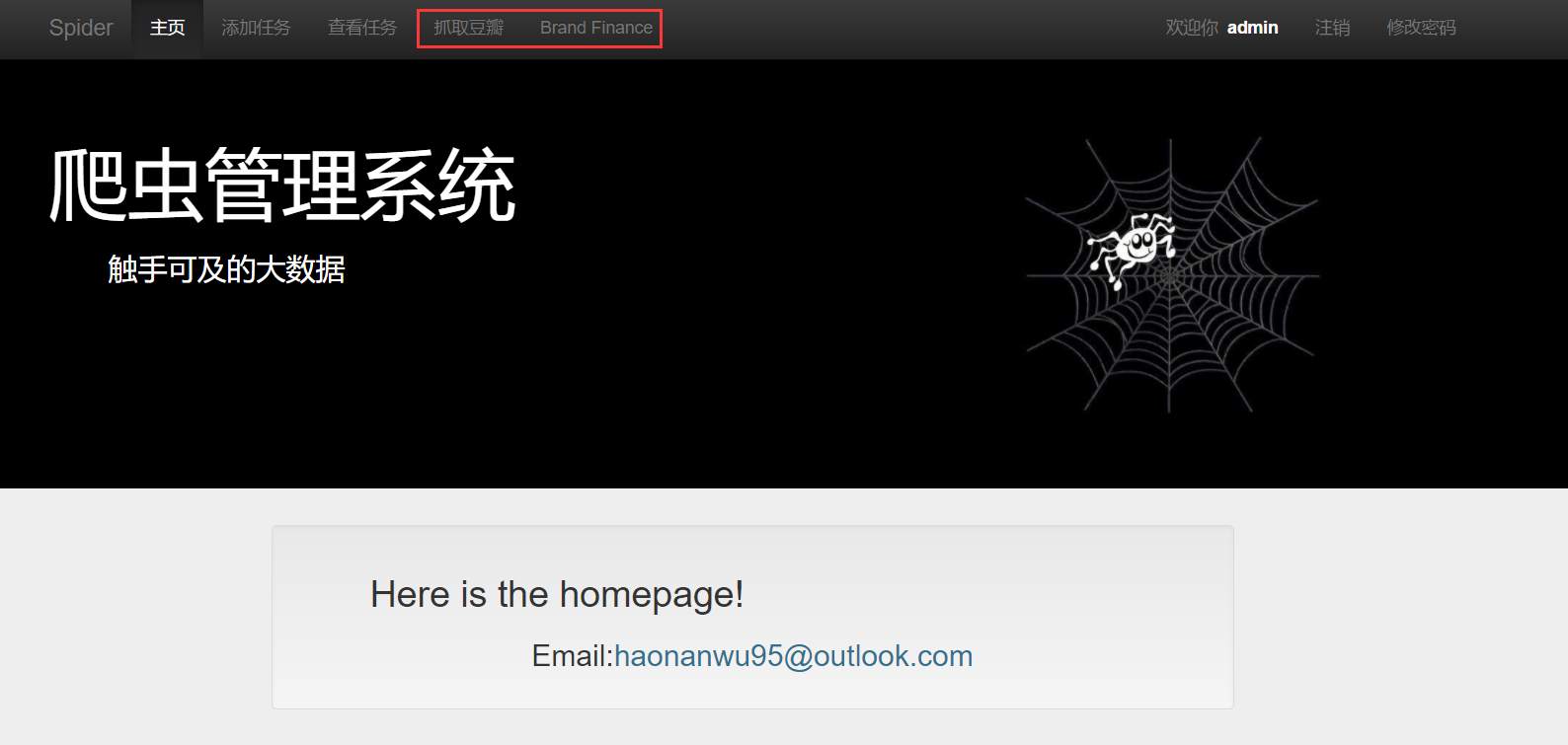


图 5-19 主页导航栏提示

* 豆瓣抓取

豆瓣抓取采用的是针对不同页面设置不同的抓取模式的设计。

此设计须要在代码中直接实现抓取规则，用户无法直接修改内部代码。只有开发人员能够设计爬虫的实现，此设计是现今社会聚焦型爬虫的主流设计模式。因为普通用户不具备编程能力也难以接触程序，所以此类设计仅适用于部分依托于爬虫进行业务开展的公司，而无法大量民用。

此范例是爬虫抓取豆瓣电影和豆瓣图书的不同样例。



图 5-20 豆瓣抓取页面

* Brand Finance抓取

Brand Finance抓取采用的是相同模式抓取相似网页的设计。

对于相同网站的相似网页，抓取相同格式的信息的抓取模式相同，因此不用直接修改爬虫源代码，直接复用爬虫就能抓取多个网页的信息。

此范例采用相同的抓取模式，抓取Brand Finance提供的2017年三个地区品牌评价信息。



图 5-21 Brand Finance抓取页面

## 管理员功能

通过登录管理员账号，会自动跳转到管理员后台管理界面，其中功能最主要是通过数据库管理实现。

超级管理员能够管理的数据库表格是系统开发人员在admin.py中设置的，包括提供给管理员权限的数据库表格、数据表格的显示情况。在本毕设中提供了User表、Group表、Task表、Task File表。



图 5-22 后台管理页面

### 用户管理

该页面可以查看注册用户信息。通过点击用户名可以进入到该用户的详情界面，对其进行更加细节的操作。

管理员可以通过右侧的过滤器对左侧显示的用户进行筛选。

管理员可以通过“动作”选项，来对选中的用户进行操作，也可以通过右上角的按钮直接修改数据库表格添加用户。

管理员可以通过左上角的搜索搜索特定关键词的用户。

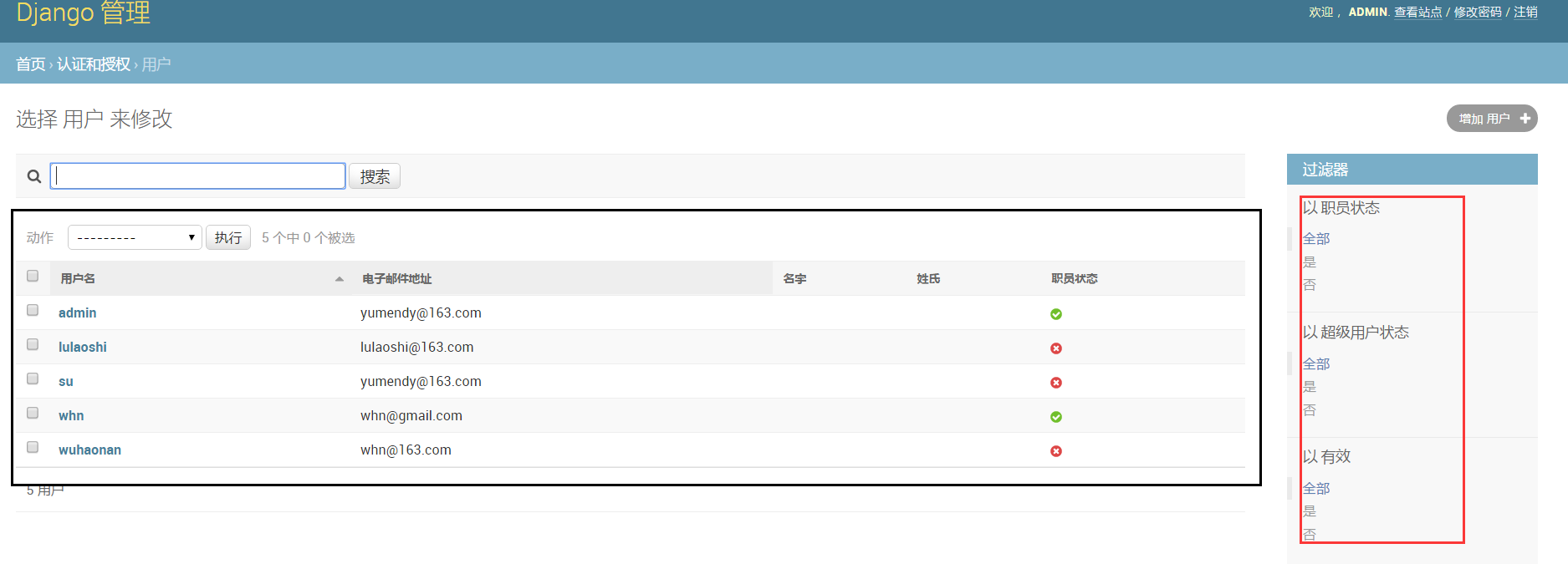
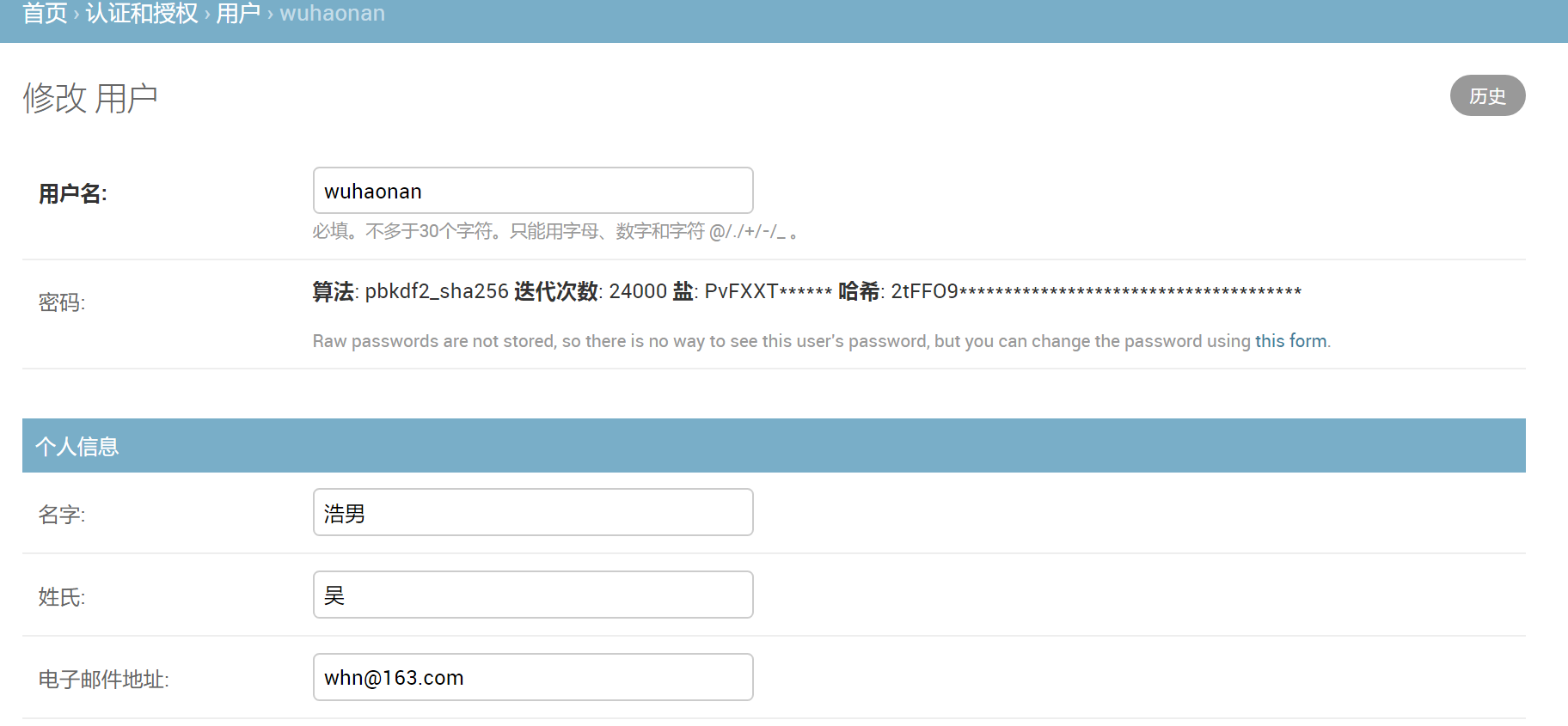


图 5-23 用户管理列表

点击进入用户详情中可以查看、修改用户的具体信息，包括用户名、密码加密算法、个人信息等，也可以查看用户等操作历史。



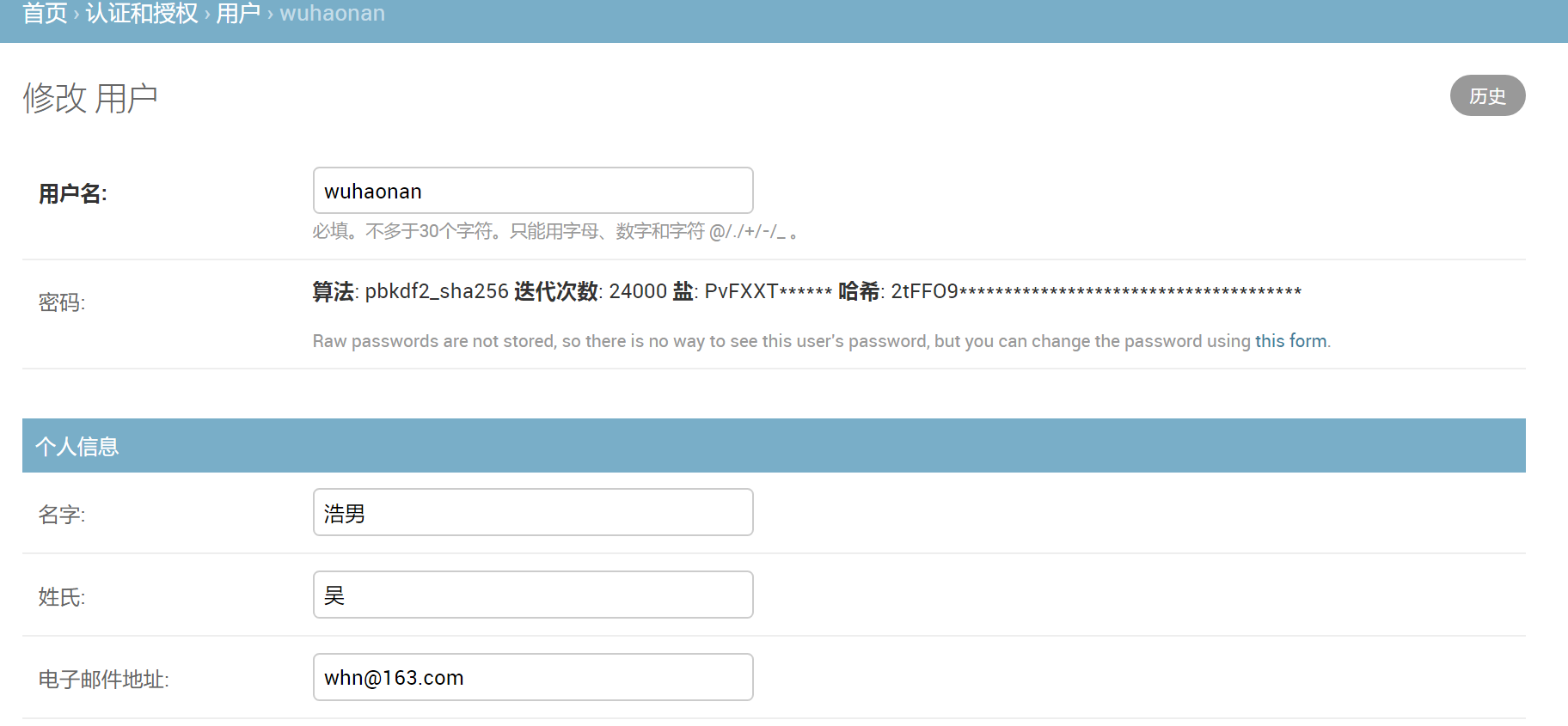


图 5-24 用户管理页面

管理员在用户管理界面也可以修改用户权限。

通过用户组，可以进行角色权限管理，更加方便地管理一类群体的权限设置。本系统中只提供了两种组，即，管理员组与用户组。但是用Group表进行权限管理有助于系统的扩展。

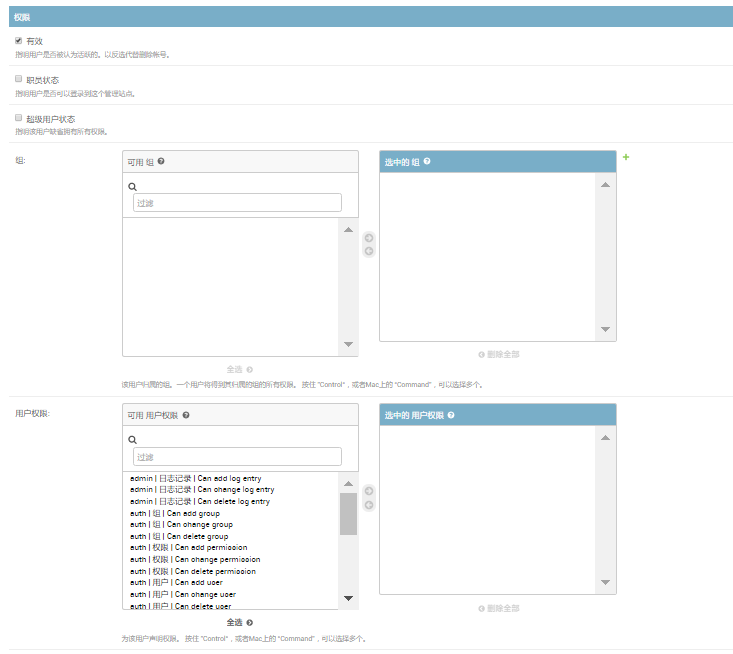
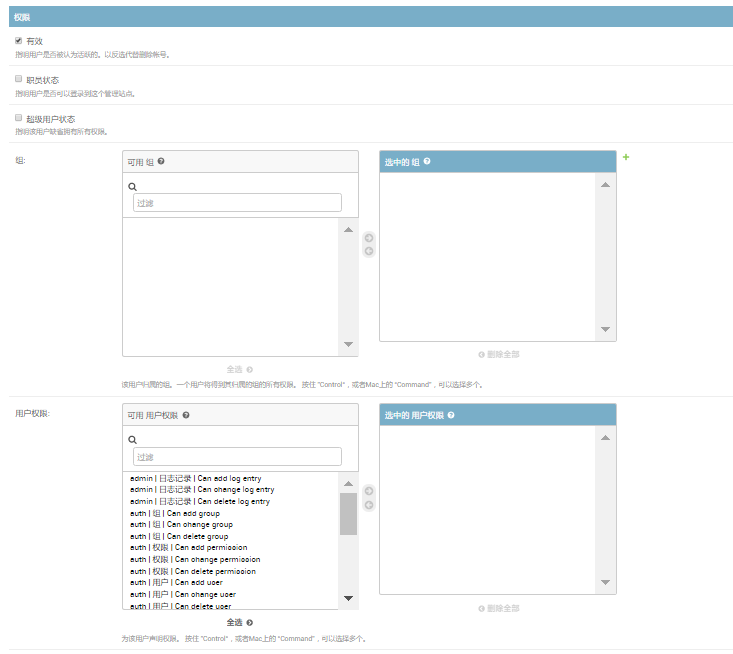


图 5-25 用户权限页面

管理员可以设置某个单独用户的权限，用户与用户权限是many-to-many的数据库关系。在开发人员的设计下，能够利用不同的权限限制用户使用的功能。



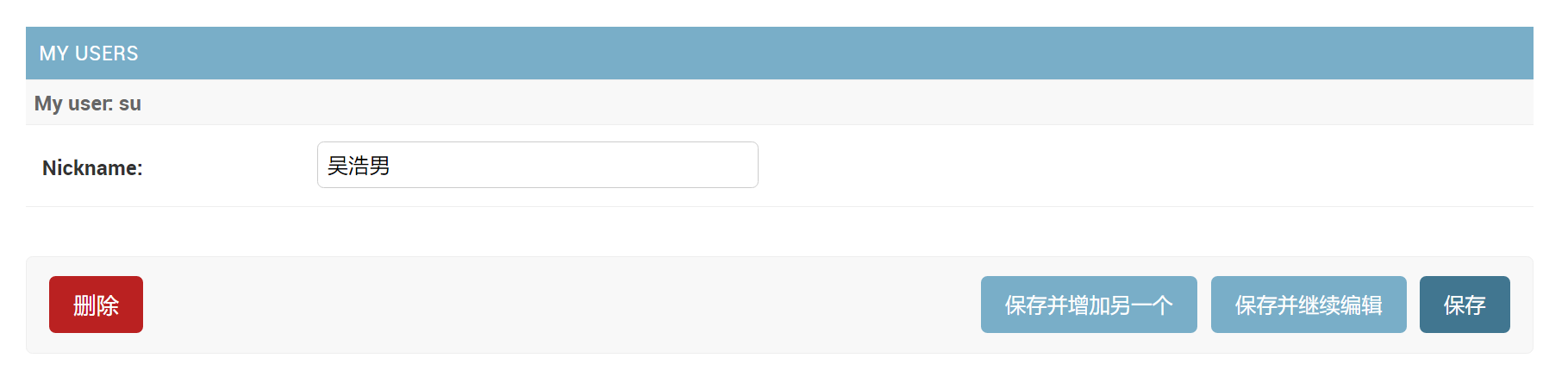


图 5-26 用户管理页面

### 爬虫任务管理

该页面可以查看所有用户创建的爬虫任务，显示的内容由开发人员在admin.py中设置，在此显示了爬虫名和爬虫任务创建者的昵称。

可以通过搜索筛选具有某些关键词的爬虫任务。

通过“动作”选项操作选中的爬虫任务，通过右上角的“添加TASK”按钮能够直接通过数据库操作添加爬虫任务。



图 5-27 爬虫任务列表

通过点击爬虫任务名能够进入爬虫任务详情界面，能够查看爬虫任务的修改历史，直接通过数据库操作查看并修改该爬虫任务的信息。

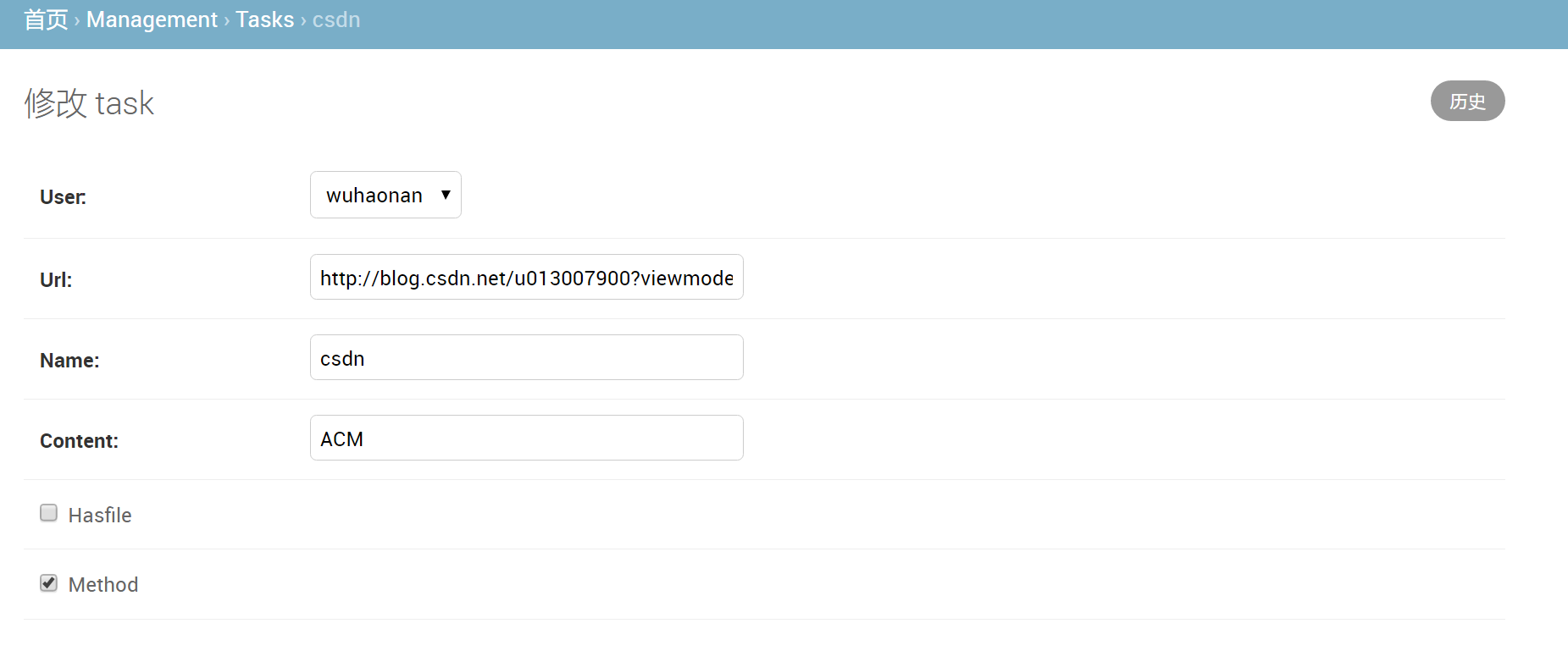


图 5-27 爬虫任务管理页面

### 爬虫数据文件管理

爬虫任务的数据文件表以任务ID为外键，则程序可以通过爬虫任务信息直接查看该爬虫任务的数据文件，对数据文件的信息进行修改。

若要直接对数据文件进行修改可以进入服务器的文件管理系统直接对数据文件的内容进行修改。

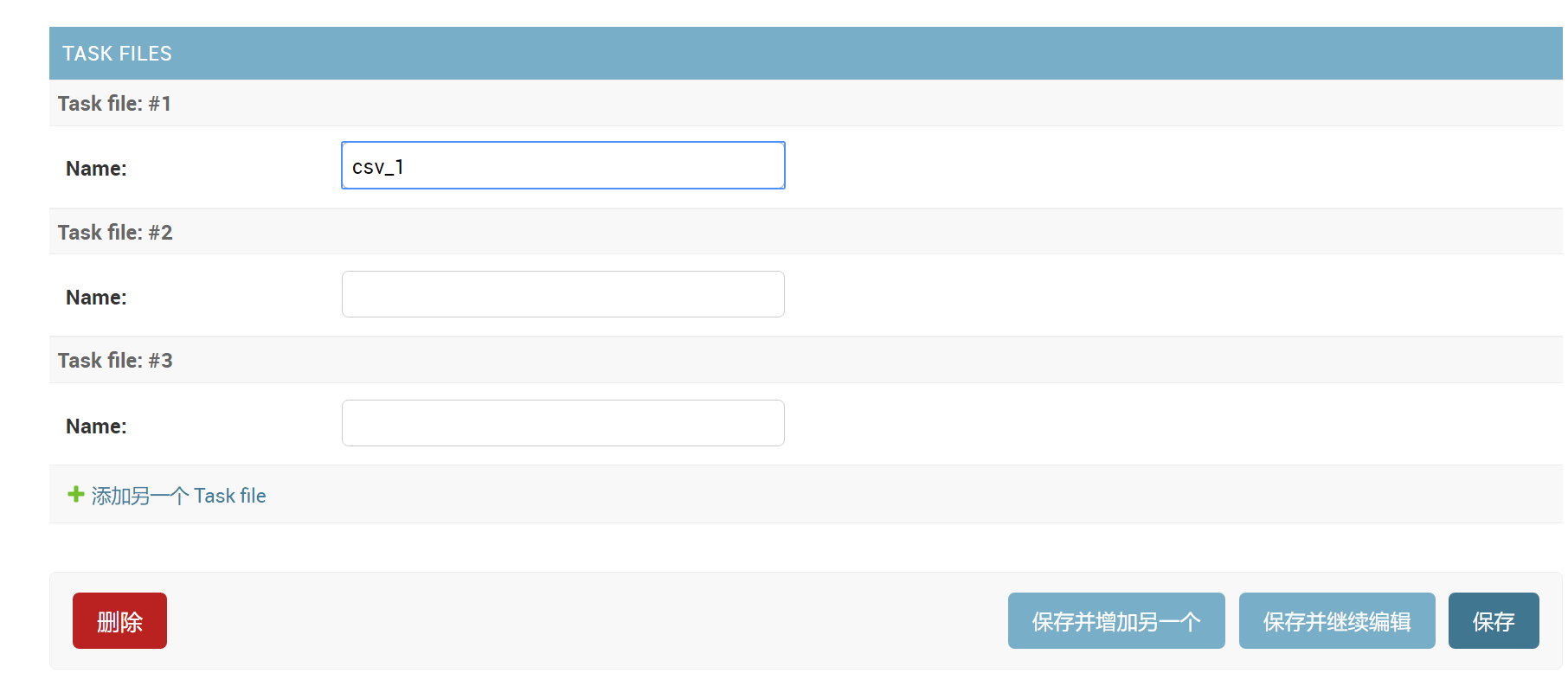


图 5-27 数据文件管理页面

## 系统测试

### Interbrand测试

对于网址<http://interbrand.com/best-brands/best-global-brands/2016/ranking/>，根据用户输入抓取内容，抓取信息。

抓取内容：**178,119 $m**

抓取网页展示：



图 5-28 InterBrand抓取页面

1. 简洁抓取

根据用户输入的抓取内容，简洁抓取算法获得了上图网页列表中每一个品牌的品牌价值。

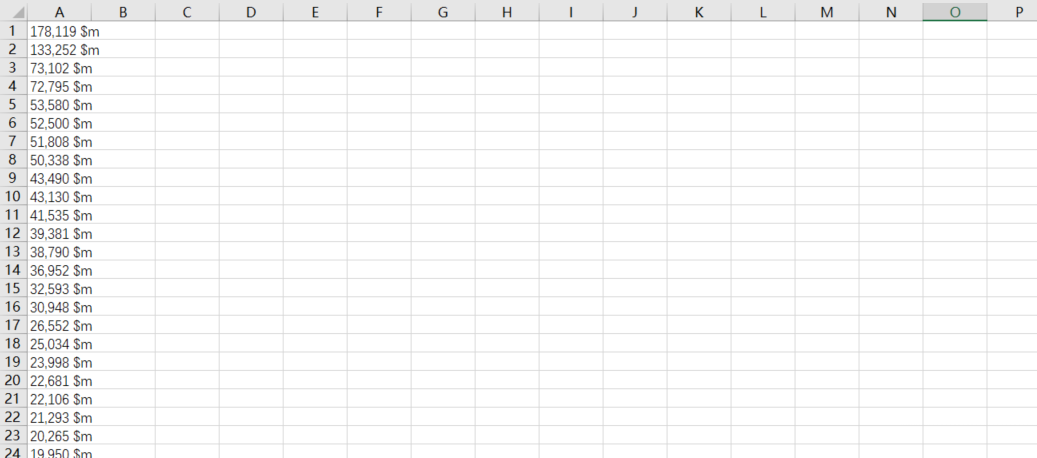


图 5-29 InterBrand简洁抓取结果

1. 非简洁抓取

根据抓取内容，非简洁抓取抓取了整个列表的信息，但是左侧有一部分无效数据，可以由用户自行整理后再使用。

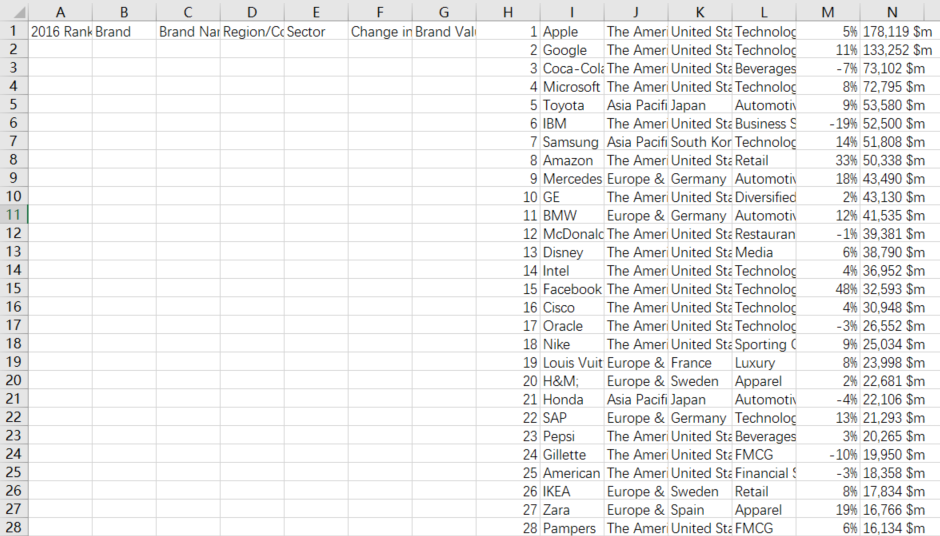


图 5-30 InterBrand非简洁抓取结果

### BrandFinance测试

对于网址<http://interbrand.com/best-brands/best-global-brands/2016/ranking/>，根据用户输入抓取内容，抓取信息。

抓取内容：**ICBC**

抓取网页展示：

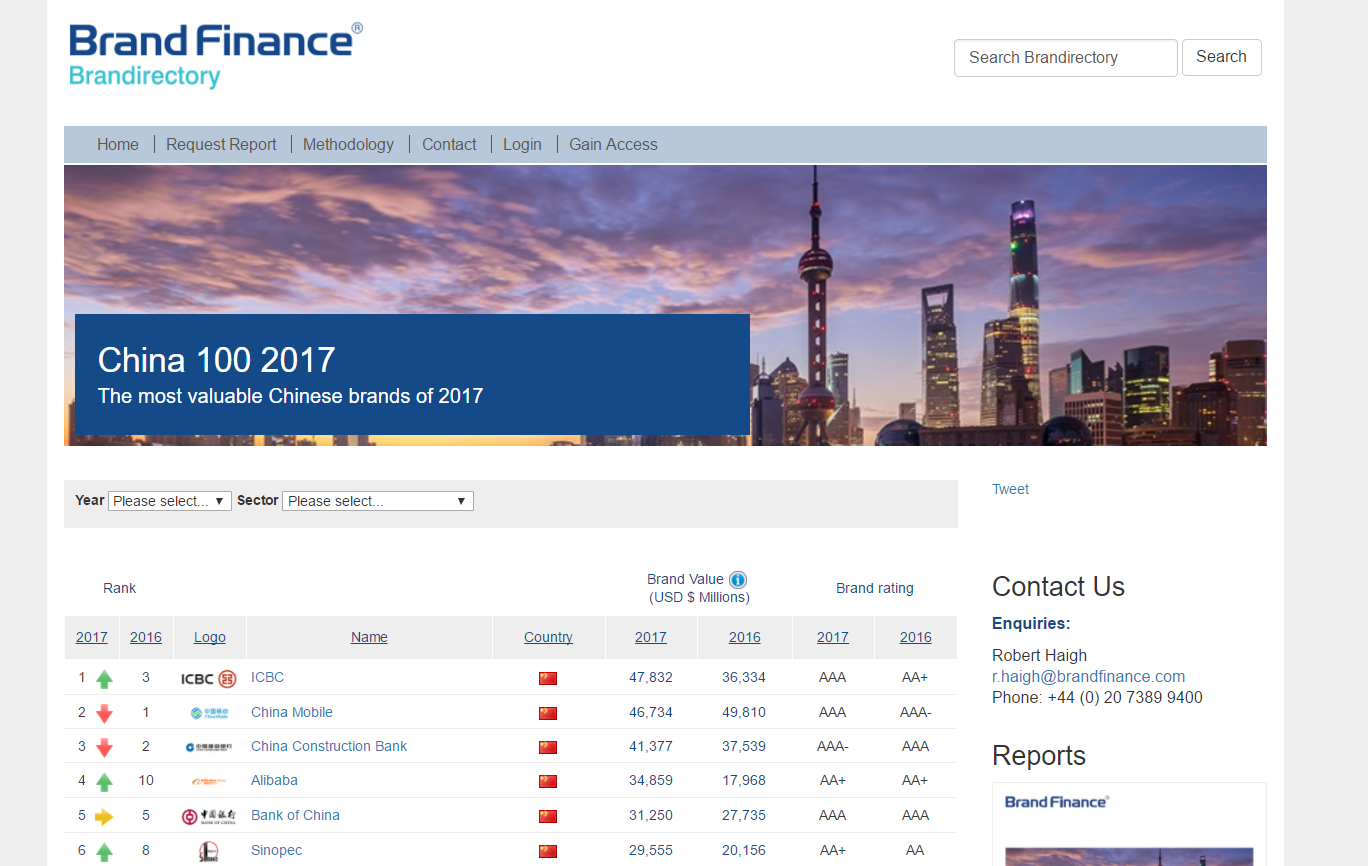


图 5-31 Brand Finance抓取页面

1. 简洁抓取

根据用户输入的抓取内容，简洁抓取算法抓取了5-31图中列表的每一个品牌的名称。

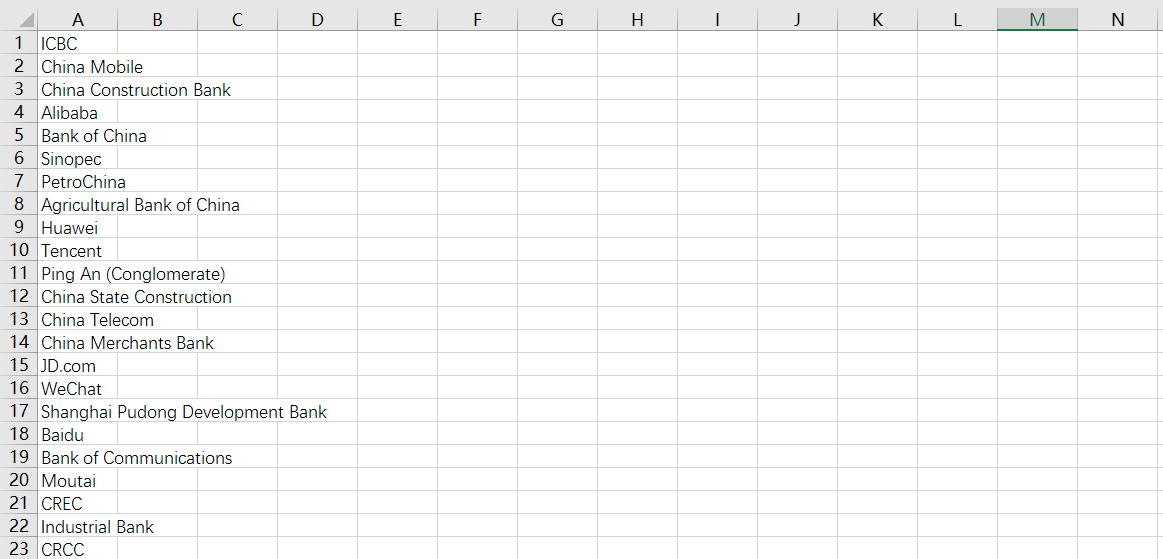


图 5-32 Brand Finance简洁抓取结果

1. 非简洁抓取

根据用户输入的抓取内容，非简洁抓取算法抓取了5-31图中列表的每一个品牌的信息。其中有部分冗余信息，如value2017，value2016等。可由用户自行整理后使用。

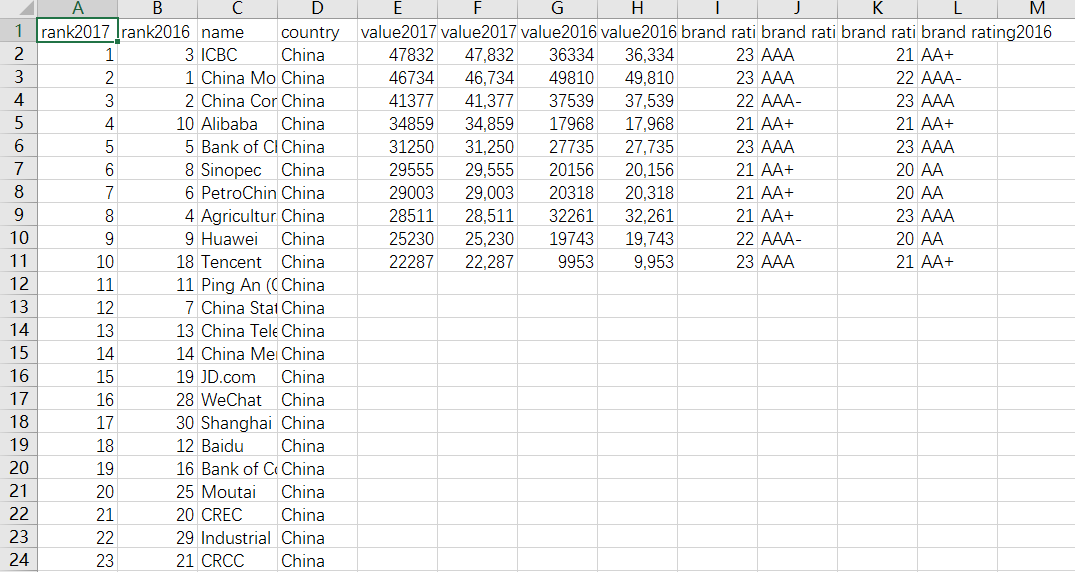


图 5-33 Brand Finance非简洁抓取结果

### 测试结果

在几个测试样例中，简洁抓取算法能够较为精确地抓取有效信息，达到任务需求。

在同样的测试样例中，非简洁抓取算法能够较为精确地识别列表，并将列表中的全部信息进行抓取。但是非简洁抓取算法难以识别列表中的无用信息，须要针对特定网页做出特定规则的修改。基本达到任务需求。

## 本章小结

本章主要介绍了Web应用提供的功能。首先，以权限为划分，介绍了普通用户和管理员用户的不同功能；其次，结合不同的功能，展示了功能的操作界面、系统实际结构与代码实现；最后，针对系统，采用了任务须要抓取的网页进行测试，证明了系统核心功能的可靠性。

# 总结与展望



## 实践成果

我在毕设实践的过程中，主要完成了如下的成果：

第一，对爬虫的使用与设计有了初步的了解，对Python的网络操作库以及第三方库有了更深入的了解。网络爬虫的便捷性大大提高了程序员收集信息的速率，同时多种多样的网络爬虫框架也使得爬虫的开发越来越容易。在设计爬虫的过程中，使用了许多原创的算法，希望能够尽可能地减轻用户的使用负担，帮助不会编程的人也能使用爬虫。当这样更加人性化的爬虫被适当地运用到生活中，将会给更多人带来益处，让生活变得更加美好，事情处理得更加顺利。

第二，完成了这款基于B/S结构的可信数据爬虫系统。。在开发Web应用的过程中，通过研究和学习Boostrap前端框架，我粗略地了解了HTML、Javascript、CSS的语言特性与使用方法，掌握了基础的前端排版设计能力；通过研究和学习Django后端框架，加深了我对Python的原生库和第三方库的了解，让我对数据库的操作、文件系统的操作、系统模块化设计等在将来软件开发过程中嫉妒重要的技能与思想有了更深切的掌握，为我今后进行更大型网站的开发打下了坚实的基础。

## 工作展望

### 分布式部署

因为系统的高内聚度和低耦合度，爬虫的抓取模块独立于于系统的其他模块，使其能够更为轻松的独立运行。在一台服务器运行本系统时，若多个用户同时发送抓取请求，则系统可以开辟多个拥有独立线程池的线程来进行启动抓取过程。所以能够在极小的代码改动下就能够对抓取模块进行分布式部署，进一步加快抓取速度。

### 抓取模型可重复利用

抓取模型之后，系统得到了此类网页的抓取模型，并且此抓取模型能够重复运用于多个此类的网页。

系统已经设计了保存抓取模式的class，只须要添加一对能够将此类模型导入导出的前后端接口，并设计一个导入模型的进行抓取的接口，即可实现重复利用。

若系统能够实现批量添加抓取网址的功能，如，有编程基础的用户用正则匹配实现URL的过滤等，并导入抓取模型，则可以大批量地抓取信息。其中Brand Finance抓取范例就是基于此类设想制作的样例。

### 算法模型的改进

算法中使用的路径相似度评估函数与评估函数阈值可以通过更多的测试进行调整，使得抓取算法的效果更佳。同时也可以提供调整参数的接口给用户，让用户自行设置评估函数与阈值，给用户更多的操作空间与便利度。

参考文献

Ryan Mitchell : Web Scraping with Python [M]. 陈小莉 等译. 北京：人民邮电出版社 2016.03

Pablo Hoffman : Scrapy 0.24 documentation [EB/OL]. marchtea 等 翻译.

<http://scrapy-chs.readthedocs.io/zh_CN/0.24/intro/tutorial.html>. 2015.07

[Kenneth Reitz](http://kennethreitz.org/) : Requests: HTTP for Humans[EB/OL].

<http://www.python-requests.org/en/master/> . 2017.03

Frank Wiles : Django documentation [EB/OL]. <https://docs.djangoproject.com/en/1.9/> . 2017.03

Twitter : Bootstrap 3.3.7 documentation [EB/OL]. <http://v3.bootcss.com/> . 2017.01

Harry J.W.Percival : Python Web开发:测试驱动方法[M]. 安道 译. 北京：人民邮电出版社. 2015.10

[10.19358/j.issn.1674-7720.2016.19.022] 潘心宇.基于网页DOM树节点路径相似度的正文抽取[J] 微型机与应用.2016年19期

**致 谢**

首先衷心感谢北京邮电大学软件学院袁玉宇教授，在本文成稿的整个过程中，袁教授为我创造了良好的科研环境，并给予了悉心的指导，使我顺利完成了论文的撰写。袁教授渊博的学识、严谨的治学方针及高尚的育人品德时刻影响着我，是我终身的良师益友。

感谢北京邮电大学计算机学院以及可信分布式计算与服务教育部重点实验室的全体老师和同学们，感谢他们曾经给予我的热情帮助与支持，让我克服了生活中的困难，学习到了有用的知识，并度过了难忘且有意义的四年大学时光。

最后，感谢一直关注我成长的家人，谢谢他们这些年来对我的培养和教育，他们的激励与教诲时刻督促着我，我在学业上的每一次进步都与他们密不可分。