**应用软件开发课程设计报告**

**题目 研招信息查询系统**

**姓 名 \_\_\_\_\_ 张俊帆\_\_\_\_**

**学 号 \_\_\_\_ 201610311104\_\_**

**指导教师 金世双**

**学科(专业) 计算机科学与技术**

**所在学院 信息工程学院**

**提交日期 2019年5月日**

**目 录**

[1. 概述 1](#_Toc25804)

[1.1. 研究背景 1](#_Toc13580)

[1.2. 要解决的问题 1](#_Toc16828)

[2. 系统框架 2](#_Toc7374)

[2.1. 爬虫程序 3](#_Toc23524)

[2.1.1. 专业条目爬虫 3](#_Toc29358)

[2.1.2. 院校爬虫 5](#_Toc9056)

[2.2. 客户端程序 6](#_Toc7816)

[2.3. 网页端程序 7](#_Toc5)

[3. 数据的获取 8](#_Toc12963)

[3.1. 原始数据展示 8](#_Toc26038)

[3.2. 最终数据展示 8](#_Toc15143)

[4. 采用的语言 10](#_Toc16520)

[4.1. HTML语言 10](#_Toc27187)

[4.2. Xpath 10](#_Toc7929)

[4.3. CSS selector 11](#_Toc3526)

[4.4. Python 13](#_Toc29365)

[4.5. C# 13](#_Toc26073)

[4.6. SQL 14](#_Toc17608)

[5. 关键技术 15](#_Toc25690)

[5.1. Scrapy爬虫框架 15](#_Toc24627)

[5.1.1. Scrapy Engine 15](#_Toc28137)

[5.1.2. 调度器(Scheduler) 15](#_Toc20020)

[5.1.3. 下载器(Downloader) 15](#_Toc10624)

[5.1.4. Spiders 15](#_Toc6595)

[5.1.5. Item Pipeline 15](#_Toc7797)

[5.1.6. 下载器中间件(Downloader middlewares) 15](#_Toc15384)

[5.1.7. Spider中间件(Spider middlewares) 16](#_Toc7033)

[5.1.8. 数据流(Data flow) 16](#_Toc25135)

[5.2. Selenium浏览器自动化测试框架 17](#_Toc30322)

[5.2.1. 获取与当前浏览器对应的驱动 17](#_Toc31657)

[5.2.2. 驱动浏览器 18](#_Toc29810)

[5.2.3. 获取指定元素 18](#_Toc17788)

[5.2.4. 获取元素之后进行相应操作 19](#_Toc25111)

[5.3. C#开发可视化界面 19](#_Toc22349)

[5.3.1. SqlHelper类 19](#_Toc26633)

[6. 系统设计 25](#_Toc12391)

[6.1. 数据库设计 25](#_Toc17542)

[6.1.1. 基本表 25](#_Toc27057)

[6.1.2. 视图 30](#_Toc6670)

[6.1.3. 索引 31](#_Toc7184)

[6.1.4. 数据库函数 31](#_Toc17582)

[6.2. 爬虫程序设计 33](#_Toc18197)

[6.2.1. 通用驱动程序 33](#_Toc29318)

[6.2.2. 通用pipeline 33](#_Toc8688)

[6.2.3. 专业条目爬虫 35](#_Toc24944)

[6.2.4. 院校爬虫 39](#_Toc20715)

[6.2.5. 院校选项爬虫 42](#_Toc29430)

[6.3. 客户端程序设计 44](#_Toc3088)

[6.3.1. 注册界面 45](#_Toc4352)

[6.3.2. 找回密码界面 46](#_Toc26529)

[6.3.3. 用户主界面 46](#_Toc28894)

[6.3.4. 信息设置界面 47](#_Toc13693)

[6.3.5. 院校信息查询界面 47](#_Toc28559)

[6.3.6. 硕士专业目录界面 48](#_Toc28347)

[6.3.7. 院校/专业收藏界面 49](#_Toc22915)

[6.3.8. 图表分析界面 50](#_Toc20215)

[6.4. 命名规范的设计 51](#_Toc16679)

[6.4.1. 窗体的命名规范 51](#_Toc22308)

[6.4.2. 控件的命名规范 52](#_Toc25329)

[7. 小组执行情况 53](#_Toc23529)

[7.1. 分工情况 53](#_Toc19158)

[7.1.1. 大致分工 53](#_Toc9158)

[7.1.2. 预计完成时间 53](#_Toc20033)

[7.2. 开发进度 53](#_Toc31646)

[7.2.1. 第一周 53](#_Toc18190)

[7.2.2. 第二周 53](#_Toc7256)

[7.2.3. 第三周 53](#_Toc7960)

[7.2.4. 第四周 54](#_Toc21169)

[7.3. 部分开发日志 54](#_Toc31052)

[8. 总结与展望 55](#_Toc8823)

[参考文献 56](#_Toc27760)

# 概述

## 研究背景

研究生专业类目繁多，权威网站——中国研究生教育信息网（以下简称“研招网”）上的分类却不够细致，考研学生有按考试科目查询和收藏专业条目的需求，并最好能有相应的图表化的分析模块，但研招网没能够提供相应的功能。

## 要解决的问题

设计一个便捷的研招信息查询系统，提供如下基本功能：

1. 由于研招网没有提供相应的接口，须获取研招网全站数据
2. 提供登录模块（用于用户收藏信息）
3. 提供收藏功能（收藏专业条目）
4. 提供图表分析模块（方便用户直观地分析其院校意向）
5. 实现相比研招网更细致的查询功能

# 系统框架

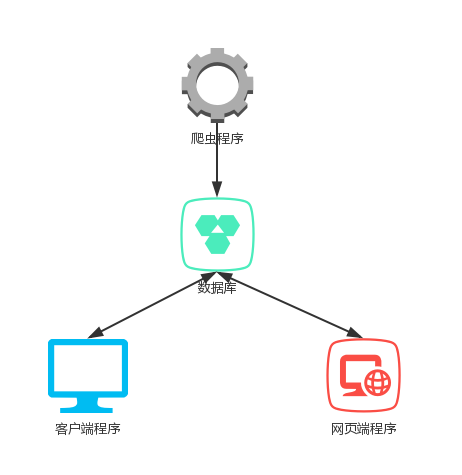


图 2.1 总体框架图

1. 爬虫程序负责将研招网上的信息爬取下来，存入数据库中
2. 客户端程序和网页端程序分别调用数据库中的数据，以实现其基本功能
3. 客户端程序和网页端程序将用户的注册数据、收藏数据保存到数据库中
4. 注：由于信息的时效性，爬虫程序是需要多次运行，以对数据进行维护，因此数据库的设计就变得尤为关键，尤其是外键上，要考虑下一次运行爬虫程序后，次序改动带来的问题（比如如果为条目随机附上编号，再与收藏夹表进行关联，则会出现对应错位的问题），将在数据库章节对此进行阐述，也正因为需要多次获取大量数据，由此体现编写爬虫程序的重要性与必要性

## 爬虫程序

### 专业条目爬虫

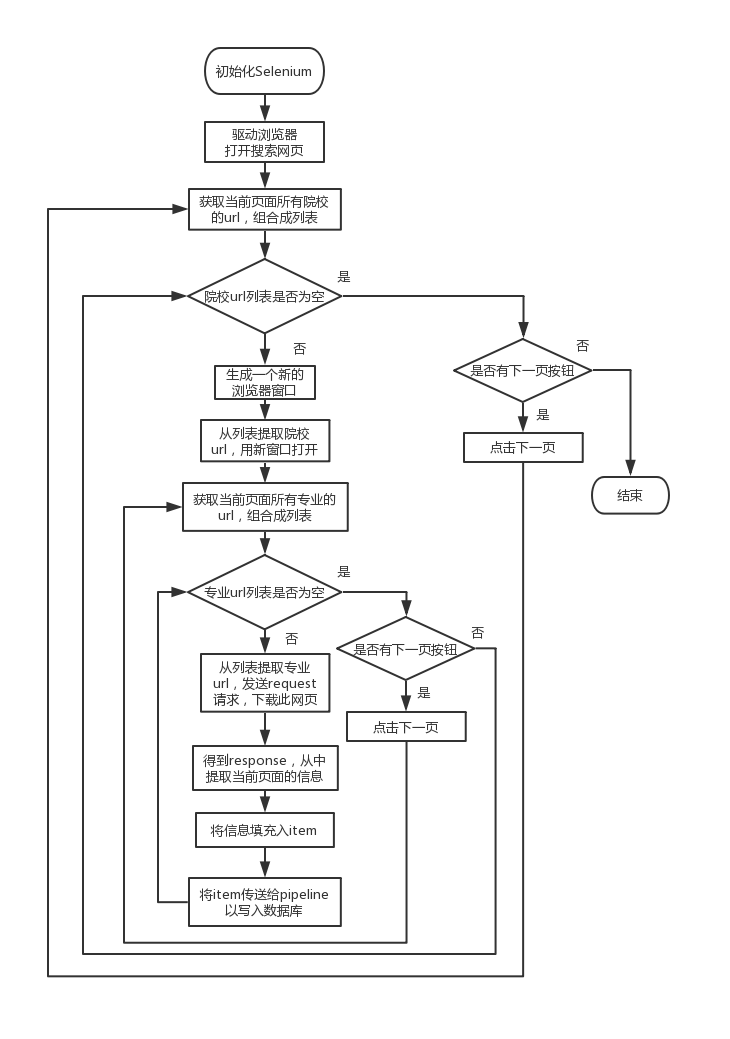


图 2.2 专业条目爬虫程序框架图



图 2.3 初始搜索页面



图 2.4 院校页面



图 2.5 专业页面

1. 程序开始，初始化selenium，驱动chrome浏览器并先打开搜索页面(<https://yz.chsi.com.cn/zsml/queryAction.do>)
2. 搜索页面如图2.3，点击最左一列各项将跳转至各个院校的页面，如图2.4
3. 在院校页面中(如图2.4)，将显示一定量的专业条目(约30条)，并分成多页
4. 获取院校页面中的url(此url可跳转至专业页面，如图2.5)，发送request请求下载页面
5. Scrapy将从获取的response中提取关键信息，生成item，传送至pipeline，pipeline负责将数据存入数据库中
6. 将当前院校页面中的所有条目处理完毕后，点击下一页按钮，重复步骤4，直至没有下一页，表示当前院校的招生信息已爬取完毕
7. 在爬取完当前页面(图2.3)的所有院校之后，点击下一页按钮，重复步骤3，直至所有院校均爬取完毕
8. 程序结束

### 院校爬虫

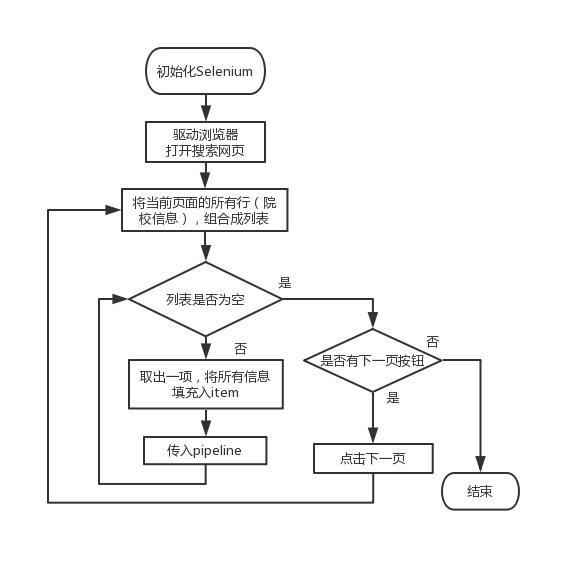


图 2.6 院校爬虫程序框架图



图 2.7 院校页面

1. 程序开始，初始化selenium，驱动chrome浏览器并先打开搜索页面(<https://yz.chsi.com.cn/sch/>)
2. 搜索页面如图2.7，点击最左一列各项将跳转至各个院校的页面
3. Scrapy将从当前页面提取关键信息，生成item，传送至pipeline，pipeline负责将数据存入数据库中
4. 将当前院校页面中的所有条目处理完毕后，点击下一页按钮，重复步骤2，直至没有下一页，表示当前页面已爬取完毕
5. 程序结束

## 客户端程序

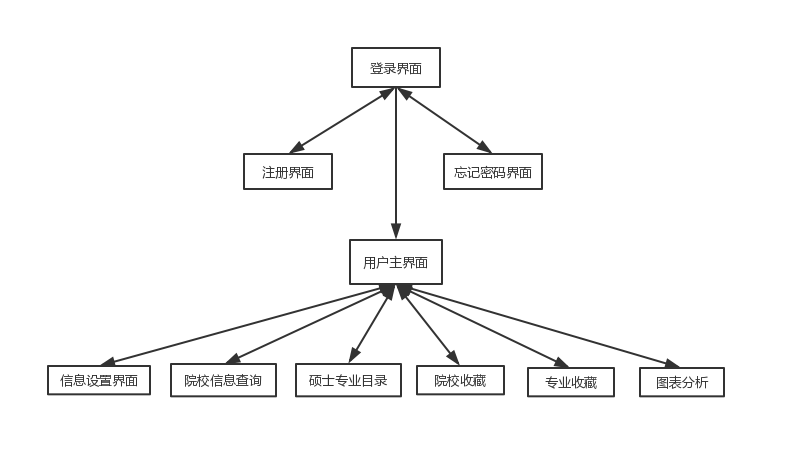


图 2.8 客户端程序框架

1. 注册界面中，用户可以注册新帐号
2. 忘记密码界面中，用户可以通过注册邮箱和电话号找回密码
3. 登录界面中，用户输入帐号密码进行登录，跳转至用户主界面
4. 信息设置界面中，用户可以修改基本信息和修改密码
5. 院校信息查询界面中，用户可以查询各个院校的基本信息，收藏指定院校
6. 硕士专业目录界面中，用户可以查询所有研究生专业条目，收藏指定专业
7. 院校收藏界面中，用户可以查询其所收藏的院校
8. 专业收藏界面中，用户可以查询其所收藏的专业
9. 图表分析界面中，用户可以通过图表，分析收藏的院校和专业构成，包括了院校分布和专业分布

## 网页端程序



图 2.9 网页端程序框架

1. 用户可以在浏览器访问网页首页，并通过导航栏可以跳转到登录页面、注册页面、管理员页面。
2. 注册界面中，用户可以注册新帐号，程序会自动向用户邮箱发送带有激活链接的激活邮件。
3. 注册成功后，用户可以登录界面，并跳转到用户页面，并通过导航栏跳转到用户首页、用户信息更改、用户密码更改、搜索界面。
4. 信息设置界面中，用户可以修改基本信息和修改密码
5. 院校信息查询界面中，用户可以查询各个院校的基本信息，收藏指定院校
6. 硕士专业目录界面中，用户可以查询所有研究生专业条目，收藏指定专业
7. 院校收藏界面中，用户可以查询其所收藏的院校
8. 专业收藏界面中，用户可以查询其所收藏的专业，并导出带有图标分析的Excel表格。

9.管理员页面可以查看用户登录日志以及用户信息。

# 数据的获取

通过编写爬虫程序，从研招网上获取数据，框架如图2.2。

## 原始数据展示



图 3.1 原始数据页

在此页面可提取的数据有：招生单位、考试方式、院系所、专业代码、专业名称（即当前页中的“专业代码”）、学习方式、研究方向、指导老师、拟招人数、备注、政治、外语、业务课一、业务课二以及url（即当前页面的url）

## 最终数据展示

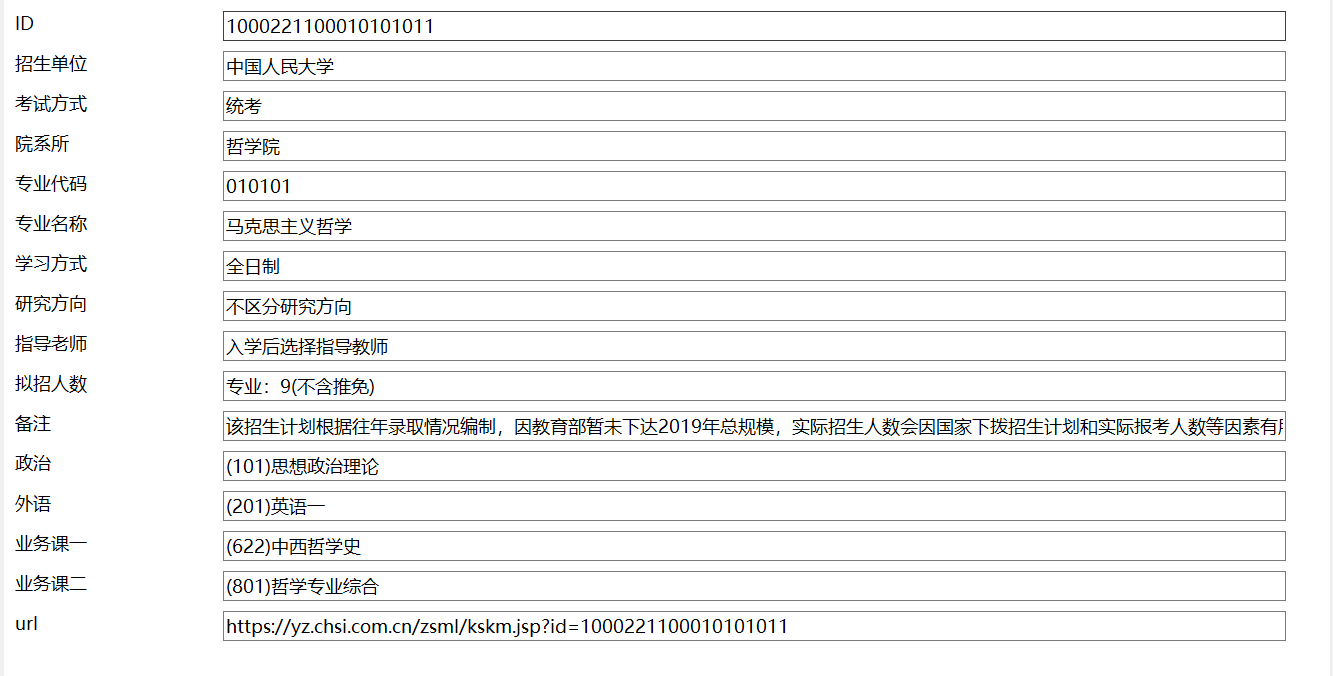


图 3.2 数据元素

说明：

1. ID截取自url，并设为主键
2. “专业代码”来自院校页面（图2.4）



图 3.3 研招信息数据表

最终数据如图3.3所示，总计145907条

# 采用的语言

## HTML语言

超文本标记语言HTML(Hyper Text Markup Language)是一种标记语言 (markup language)，是网页制作所必备的

“超文本”就是指页面内可以包含图片、链接，甚至音乐、程序等非文字元素。

超文本标记语言的结构包括“头”部分（英语：Head）、和“主体”部分（英语：Body），其中“头”部提供关于网页的信息，“主体”部分提供网页的具体内容。

HTML具有简易、可扩展、平台无关和通用的特点

主要的构架是

***<Html>***

**<Head>**

**</Head>**

**<Body>**

Something

**</Body>**

***</Html>***

其中：

<html></html>这两个标记符分别表示文件的开始和结尾；

<head></head>这两个标记符分别表示头部信息的开始和结尾；

<Body></Body>这两个标记符表示网页中显示的实际内容均包含在这两个正文标记符之间

在本次设计中，由于需要解析网页，并从中提取数据，所以必须熟悉HTML语言

## Xpath

XPath即为XML路径语言（XML Path Language），它是一种用来确定XML文档中某部分位置的语言。

XPath基于XML的树状结构，提供在数据结构树中找寻节点的能力。起初XPath的提出的初衷是将其作为一个通用的、介于XPointer与XSL间的语法模型。但是XPath很快的被开发者采用来当作小型查询语言。

Xpath使用路径表达式在XML文档中选取节点。节点是通过沿着路径来选取的，通过路径可以找到我们想要的节点或者节点范围。

以下展示XPath语言的部分语法：

|  |  |  |  |
| --- | --- | --- | --- |
| 表达式 | 描述 | 用法 | 说明 |
| nodename | 选取此节点的所有子节点。 | xpath(‘span’) | 选取span元素的所有子节点 |
| / | 从根节点选取 | xpath(‘/div’) | 从根节点上选取div节点 |
| // | 从匹配选择的当前节点选择文档中的节点，而不考虑它们的位置。 | xpath(‘//div’) | 从当前节点选取含有div节点的标签 |
| . | 选取当前节点 | xpath(‘./div’) | 选取当前节点下的div标签 |
| .. | 选取当前节点的父节点 | xpath(‘../’) | 回到上一级节点 |
| @ | 选取属性 | xpath(“//div[@id=’1001’]”) | 获取div标签中，含有ID属性且值为1001的标签 |

在本次设计中，由于需要选定结点，并从中提取数据，所以在调试时应用了Xpath语法。

## CSS selector

CSS selecotr用于从页面中选定结点，进行交互或提取等操作（例如，点击按钮或者是提取文字）。

以下展示CSS selecotr的全部语法：

|  |  |  |  |
| --- | --- | --- | --- |
| **选择器** | **例子** | **例子描述** | **CSS** |
| .*class* | .intro | 选择 class="intro" 的所有元素。 | 1 |
| #*id* | #firstname | 选择 id="firstname" 的所有元素。 | 1 |
| \* | \* | 选择所有元素。 | 2 |
| *element* | p | 选择所有 <p> 元素。 | 1 |
| *element*,*element* | div,p | 选择所有 <div> 元素和所有 <p> 元素。 | 1 |
| *element* *element* | div p | 选择 <div> 元素内部的所有 <p> 元素。 | 1 |
| *element*>*element* | div>p | 选择父元素为 <div> 元素的所有 <p> 元素。 | 2 |
| *element*+*element* | div+p | 选择紧接在 <div> 元素之后的所有 <p> 元素。 | 2 |
| [*attribute*] | [target] | 选择带有 target 属性所有元素。 | 2 |
| [*attribute*=*value*] | [target=\_blank] | 选择 target="\_blank" 的所有元素。 | 2 |
| [*attribute*~=*value*] | [title~=flower] | 选择 title 属性包含单词 "flower" 的所有元素。 | 2 |
| [*attribute*|=*value*] | [lang|=en] | 选择 lang 属性值以 "en" 开头的所有元素。 | 2 |
| :link | a:link | 选择所有未被访问的链接。 | 1 |
| :visited | a:visited | 选择所有已被访问的链接。 | 1 |
| :active | a:active | 选择活动链接。 | 1 |
| :hover | a:hover | 选择鼠标指针位于其上的链接。 | 1 |
| :focus | input:focus | 选择获得焦点的 input 元素。 | 2 |
| :first-letter | p:first-letter | 选择每个 <p> 元素的首字母。 | 1 |
| :first-line | p:first-line | 选择每个 <p> 元素的首行。 | 1 |
| :first-child | p:first-child | 选择属于父元素的第一个子元素的每个 <p> 元素。 | 2 |
| :before | p:before | 在每个 <p> 元素的内容之前插入内容。 | 2 |
| :after | p:after | 在每个 <p> 元素的内容之后插入内容。 | 2 |
| :lang(*language*) | p:lang(it) | 选择带有以 "it" 开头的 lang 属性值的每个 <p> 元素。 | 2 |
| *element1*~*element2* | p~ul | 选择前面有 <p> 元素的每个 <ul> 元素。 | 3 |
| [*attribute*^=*value*] | a[src^="https"] | 选择其 src 属性值以 "https" 开头的每个 <a> 元素。 | 3 |
| [*attribute*$=*value*] | a[src$=".pdf"] | 选择其 src 属性以 ".pdf" 结尾的所有 <a> 元素。 | 3 |
| [*attribute*\*=*value*] | a[src\*="abc"] | 选择其 src 属性中包含 "abc" 子串的每个 <a> 元素。 | 3 |
| :first-of-type | p:first-of-type | 选择属于其父元素的首个 <p> 元素的每个 <p> 元素。 | 3 |
| :last-of-type | p:last-of-type | 选择属于其父元素的最后 <p> 元素的每个 <p> 元素。 | 3 |
| :only-of-type | p:only-of-type | 选择属于其父元素唯一的 <p> 元素的每个 <p> 元素。 | 3 |
| :only-child | p:only-child | 选择属于其父元素的唯一子元素的每个 <p> 元素。 | 3 |
| :nth-child(*n*) | p:nth-child(2) | 选择属于其父元素的第二个子元素的每个 <p> 元素。 | 3 |
| :nth-last-child(*n*) | p:nth-last-child(2) | 同上，从最后一个子元素开始计数。 | 3 |
| :nth-of-type(*n*) | p:nth-of-type(2) | 选择属于其父元素第二个 <p> 元素的每个 <p> 元素。 | 3 |
| :nth-last-of-type(*n*) | p:nth-last-of-type(2) | 同上，但是从最后一个子元素开始计数。 | 3 |
| :last-child | p:last-child | 选择属于其父元素最后一个子元素每个 <p> 元素。 | 3 |
| :root | :root | 选择文档的根元素。 | 3 |
| :empty | p:empty | 选择没有子元素的每个 <p> 元素（包括文本节点）。 | 3 |
| :target | #news:target | 选择当前活动的 #news 元素。 | 3 |
| :enabled | input:enabled | 选择每个启用的 <input> 元素。 | 3 |
| :disabled | input:disabled | 选择每个禁用的 <input> 元素 | 3 |
| :checked | input:checked | 选择每个被选中的 <input> 元素。 | 3 |
| :not(*selector*) | :not(p) | 选择非 <p> 元素的每个元素。 | 3 |
| ::selection | ::selection | 选择被用户选取的元素部分。 | 3 |

在本次设计中，由于需要选定结点，点击或从中提取数据，大量使用了CSS selector，而Xpath虽与其有同样的作用，但由于网页结构很可能有小幅的变动，如果大量使用Xpath语法，则不利于后期维护，但如果用CSS selector，则并可避免。

所以仅在调试和开发初期使用了Xpath语法，后期将全部的Xpath替换成了CSS selector

## Python

Python是一种计算机程序设计语言。是一种面向对象的动态类型语言，最初被设计用于编写自动化脚本(shell)，随着版本的不断更新和语言新功能的添加，越来越多被用于独立的、大型项目的开发。

Python语言具有简洁性、易读性以及可扩展性，且Python社区相当活跃，有很多扩展库，能够解决大量的问题

在本次设计中，应用了当今最流行的爬虫框架——Scrapy，它是用Python所编写的，因此对于本次设计，Python是至关重要的一门语言

## C#

C#是微软公司发布的一种面向对象的、运行于.NET Framework和.NET Core(完全开源，跨平台)之上的高级程序设计语言。

C#是一种安全的、稳定的、简单的、优雅的，由C和C++衍生出来的面向对象的编程语言。它在继承C和C++强大功能的同时去掉了一些它们的复杂特性（例如没有宏以及不允许多重继承）。C#综合了VB简单的可视化操作和C++的高运行效率，以其强大的操作能力、优雅的语法风格、创新的语言特性和便捷的面向组件编程的支持成为.NET开发的首选语言。

在本次设计中，PC端部分的实体软件使用C#编写的，由于所有小组成员均已通过了C#软件开发相关课程，所以选用C#用于实体软件的设计。

## SQL

结构化查询语言(Structured Query Language)简称SQL，是一种特殊目的的编程语言，是一种数据库查询和程序设计语言，用于存取数据以及查询、更新和管理关系数据库系统；同时也是数据库脚本文件的扩展名。

各种不同的数据库对SQL语言的支持与标准存在着细微的不同，已有100多种遍布在从微机到大型机上的数据库产品SQL，其中包括DB2、SQL/DS、ORACLE、INGRES、SYBASE、SQLSERVER、DBASEⅣ、PARADOX、MICROSOFTACCESS等。

在本次设计中，使用的是Mysql和SQLserver数据库。

# 关键技术

## Scrapy爬虫框架

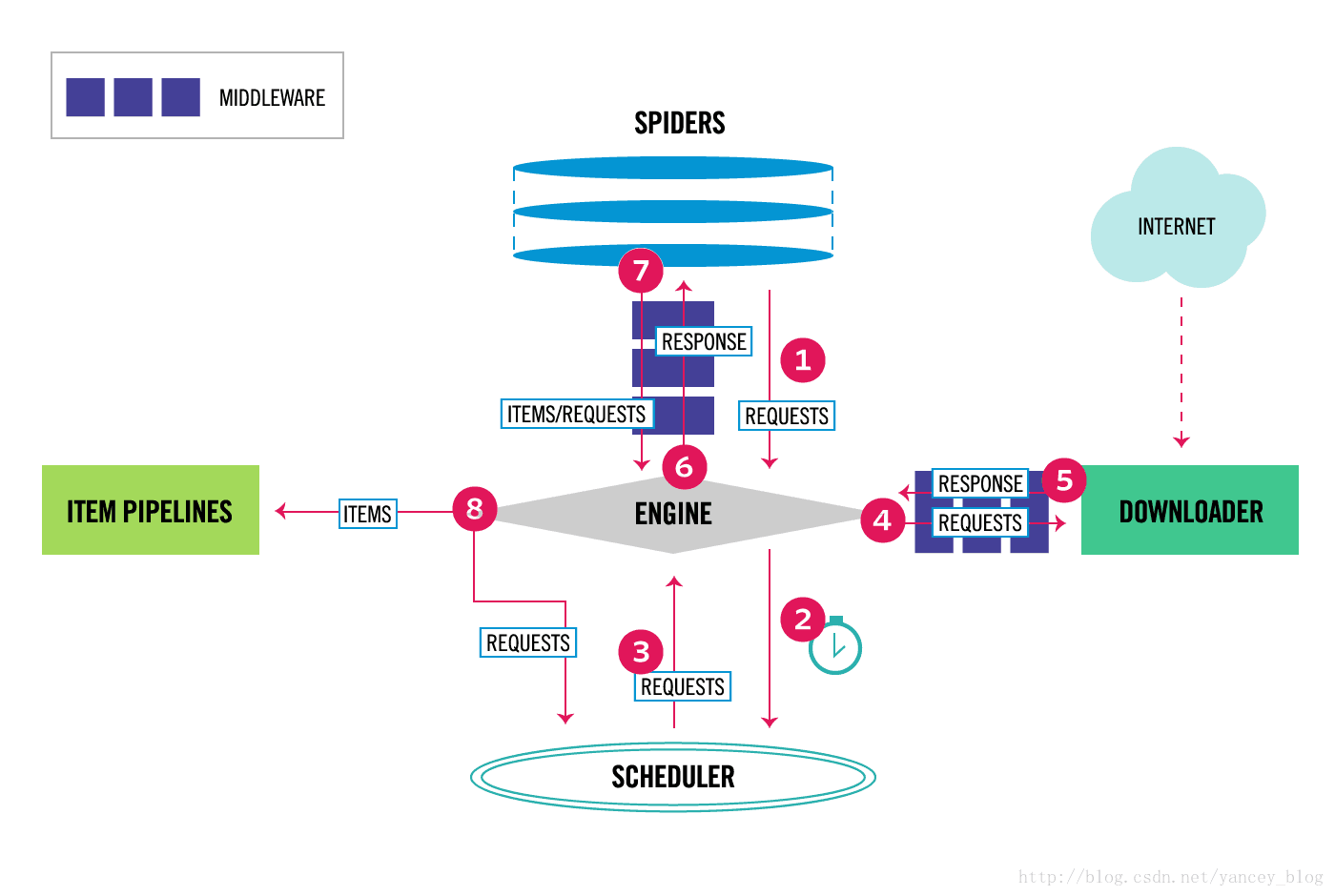


图 5.1 Scrapy框架图

### Scrapy Engine

引擎负责控制数据流在系统中所有组件中流动，并在相应动作发生时触发事件。

### 调度器(Scheduler)

调度器从引擎接受request并将他们入队，以便之后引擎请求他们时提供给引擎。

### 下载器(Downloader)

下载器负责获取页面数据并提供给引擎，而后提供给spider。

### Spiders

Spider是Scrapy用户编写用于分析response并提取item或额外跟进的URL的类。 每个spider负责处理一个特定(或一些)网站。

### Item Pipeline

Item Pipeline负责处理被spider提取出来的item，存取到数据库中。

### 下载器中间件(Downloader middlewares)

下载器中间件是在引擎及下载器之间的特定钩子(specific hook)，处理Downloader传递给引擎的response（也包括引擎传递给下载器的Request）。 其提供了一个简便的机制，通过插入自定义代码来扩展Scrapy功能。

一句话总结就是：处理下载请求部分

### Spider中间件(Spider middlewares)

Spider中间件是在引擎及Spider之间的特定钩子(specific hook)，处理spider的输入(response)和输出(items及requests)。 其提供了一个简便的机制，通过插入自定义代码来扩展Scrapy功能。

一句话总结就是：处理解析部分

### 数据流(Data flow)

Scrapy中的数据流由执行引擎控制，其过程如下:

1. 引擎打开一个网站(open a domain)，找到处理该网站的Spider并向该spider请求第一个要爬取的URL(s)。
2. 引擎从Spider中获取到第一个要爬取的URL并在调度器(Scheduler)以Request调度。
3. 引擎向调度器请求下一个要爬取的URL。
4. 调度器返回下一个要爬取的URL给引擎，引擎将URL通过下载中间件(请求(request)方向)转发给下载器(Downloader)。
5. 一旦页面下载完毕，下载器生成一个该页面的Response，并将其通过下载中间件(返回(response)方向)发送给引擎。
6. 引擎从下载器中接收到Response并通过Spider中间件(输入方向)发送给Spider处理。
7. Spider处理Response并返回爬取到的Item及(跟进的)新的Request给引擎。
8. 引擎将(Spider返回的)爬取到的Item给Item Pipeline，将(Spider返回的)Request给调度器。
9. (从第二步)重复直到调度器中没有更多地request，引擎关闭该网站。

## Selenium浏览器自动化测试框架

由于研招网的搜索网页是随着搜索条件动态填充的（点击搜索之后url不变），所以无法通过url下载当前页面的方式，获取所需要的信息。因此，必须能够操作页面中的元素（例如点击指定按键）和从当前页面提取信息。单纯地用Scrapy框架并不能解决这一问题，所以使用Selenium框架。

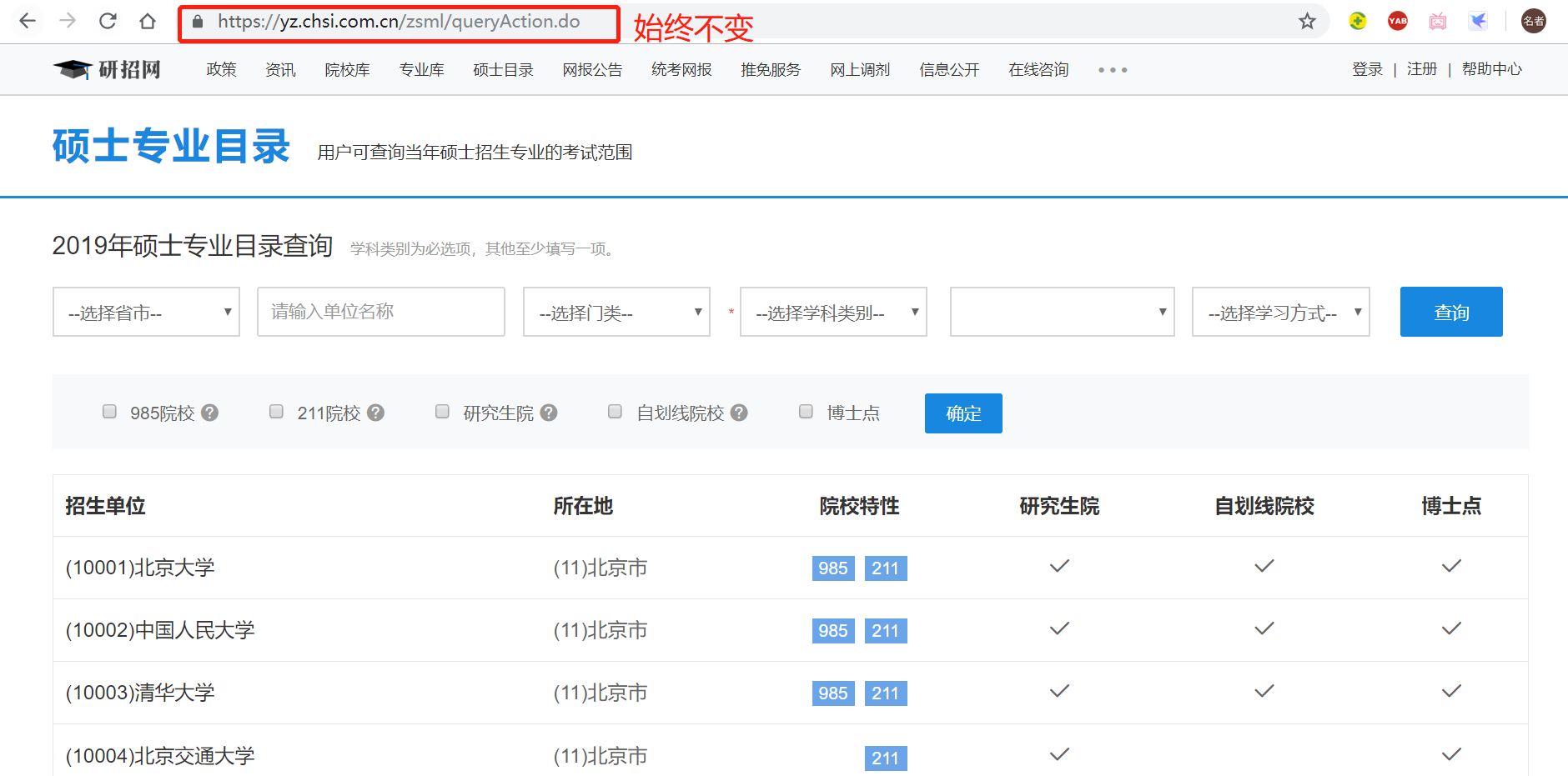


图 5.2 动态填充示意

### 获取与当前浏览器对应的驱动

在本次设计中，使用的是chrome浏览器，因此需获取chromedriver（且版本必须相对应）



图 5.3 Chrome浏览器版本



图 5.4 Chromedriver与Chrome版本对应表

### 驱动浏览器

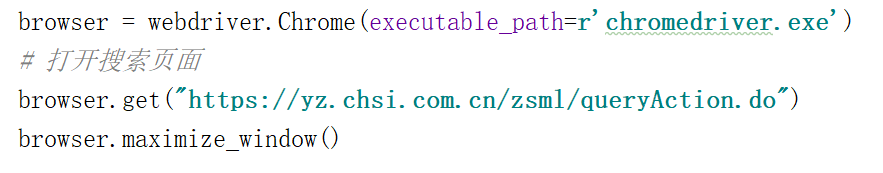


图 5.5 驱动浏览器

指定Chromedriver路径地址，并打开指定网页

### 获取指定元素

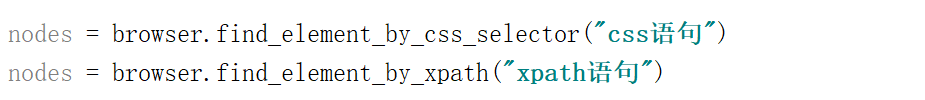


图 5.6 css和xpath方式获取指定元素

#### 按照css语句查询元素

使用find\_elements\_by\_css\_selector方法，传入css语句，即可获取相应元素

#### 按照xpath语句查询元素

使用find\_elements\_by\_xpath方法，传入xpath语句，即可获取相应元素

### 获取元素之后进行相应操作

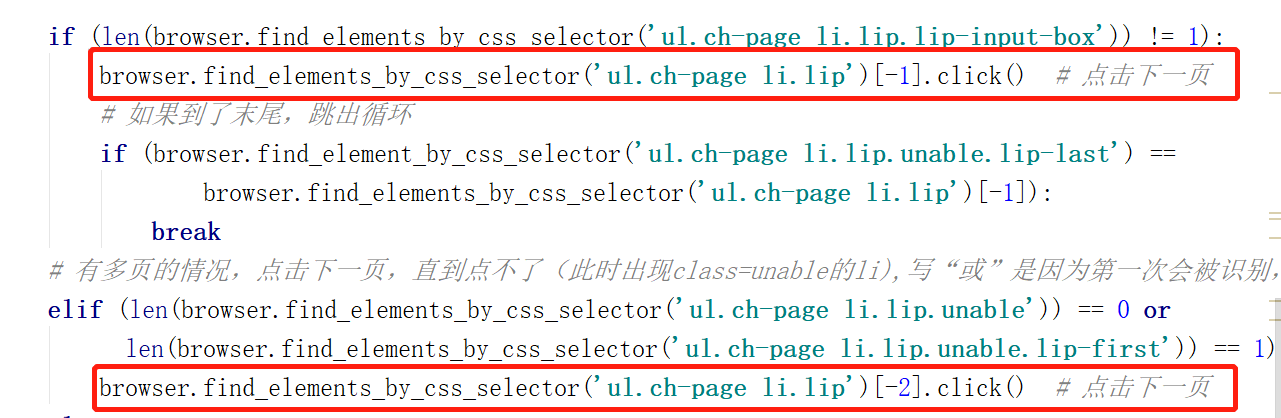


图 5.7 点击相应元素

使用元素的click方法，即可点击此元素

## C#开发可视化界面

### SqlHelper类

在此设计了一个名为SqlHelper的类，专门用于执行与数据库相关的操作。以下将展示类中包含的各个方法。

#### 用于数据库连接的函数GetConn

**//获取数据库连接 返回连接对象**

public static SqlConnection GetConn()

{

return new SqlConnection("server=.;database=研招信息查询系统;integrated security=SSPI");

}

#### 执行任意SQL语句的函数ExecuteNonQuery

**// 执行sql语句，返回受影响行数**

public static int ExecuteNonQuery(string sql)

{

int rows = -1;

using (SqlConnection conn = GetConn())

{

SqlCommand cmd = new SqlCommand(sql, conn);

conn.Open();

rows = cmd.ExecuteNonQuery();

}

return rows;

}

#### 查询是否有对应信息的函数

##### 用于比对密码的函数CheckPassword

**//传入账号和密码，查询“用户表”或“管理员表”中是否有对应的信息，用于登录模块**

public static bool CheckPassword(string Table, string Account, string Password)

{

using (SqlConnection conn = GetConn())

{

string sql = string.Format("SELECT COUNT(\*) FROM {0} WHERE 帐号 ='{1}' AND 密码 ='{2}';", Table, Account, Password);

SqlCommand cmd = new SqlCommand(sql, conn);

conn.Open();

int obj = Convert.ToInt32(cmd.ExecuteScalar());

if (obj > 0)

return true;

else

return false;

}

}

##### 查询表中是否有对应信息的函数CheckExist

**//查询指定表中是否有对应信息**

public static bool CheckExist(string Table, string Where)

{

using (SqlConnection conn = GetConn())

{

string sql = string.Format("SELECT COUNT(\*) FROM {0} WHERE {1};", Table, Where);

SqlCommand cmd = new SqlCommand(sql, conn);

conn.Open();

int obj = Convert.ToInt32(cmd.ExecuteScalar());

if (obj > 0)

return true;

else

return false;

}

}

#### 执行插入操作的函数InsertItem

**//执行sql语句，对指定的Table进行数据的插入操作**

public static void InsertItem(string Table, string Values)

{

using (SqlConnection conn = GetConn())

{

string sql = String.Format("INSERT INTO {0} VALUES {1}", Table, Values);

SqlHelper.ExecuteNonQuery(sql);

}

}

#### 执行删除操作的函数DeleteItem

**//执行sql语句，对指定的Table进行数据的删除操作**

public static bool DeleteItem(string Table, string Where)

{

using (SqlConnection conn = GetConn())

{

string sql = String.Format("DELETE FROM {0} WHERE {1};", Table, Where);

int j = SqlHelper.ExecuteNonQuery(sql); //得到受影响的行数

if (j == 1)

return true;

else

{

return false;

}

}

}

#### 执行更新操作的函数UpdatetItem

**//执行sql语句，对指定的Table进行数据的更新操作**

public static bool UpdatetItem(string Table, string Set, string Where)

{

using (SqlConnection conn = GetConn())

{

string sql = String.Format(" UPDATE {0} SET {1} WHERE {2};", Table, Set, Where);

int i = SqlHelper.ExecuteNonQuery(sql);

if (i == 1)

return true;

else

return false;

}

}

#### 其他重要的函数

##### 用于将查询结果投射到DataGridView的函数

**//执行sql语句，并把查询结果用传入的dataGridView显示出来**

public static void ShowDataWithDataGirdView(DataGridView dataGridView, string sql)//定义函数

{

using (SqlConnection conn = GetConn())

{

SqlDataAdapter da = new SqlDataAdapter(sql, conn);

DataSet ds = new DataSet(); //创建数据集

conn.Open();

da.Fill(ds);

conn.Close();

dataGridView.DataSource = ds.Tables[0].DefaultView;

}

}

##### 用于查询相应结果的函数InquireItem

**//执行sql语句，返回查询的第一个结果，若查询失败，返回字符串'Not Found!'**

public static string InquireItem(string Table, string Inquire\_name, string Where)

{

using (SqlConnection conn = GetConn())

{

string sql = String.Format("SELECT {0} FROM {1} WHERE {2}", Inquire\_name, Table, Where);

DataSet ds = new DataSet(); //创建数据集

SqlDataAdapter da = new SqlDataAdapter();

da.SelectCommand = new SqlCommand(sql, conn);

da.Fill(ds);

conn.Open();

SqlCommand cmd = new SqlCommand(sql, conn);

SqlDataReader dr = cmd.ExecuteReader();

string temp = "Not Found!";

if (dr.Read())

temp = dr.GetValue(0).ToString();

conn.Close();

return temp;

}

}

##### 用于自动生成编号的函数GetID

**//执行sql语句，返回查询结果中的最大项，主要用于新注册用户时对其自动编号(即取相应的Id的最大值再加上一）**

public static int GetID(string Table, string item)

{

using (SqlConnection conn = GetConn())

{

string sql = String.Format("SELECT MAX({0}) FROM {1};", item, Table);

DataSet ds = new DataSet(); //创建数据集

SqlDataAdapter da = new SqlDataAdapter();

da.SelectCommand = new SqlCommand(sql, conn);

da.Fill(ds);

conn.Open();

SqlCommand cmd = new SqlCommand(sql, conn);

SqlDataReader dr = cmd.ExecuteReader();

if (dr.Read())

{

int Number\_new = 1;

try

{

Number\_new = Convert.ToInt32(dr.GetValue(0)) + 1;

}

catch { }//如果查不到就会报错，最终就会返回1

conn.Close();

return Number\_new;

}

else

{

conn.Close();

return 1;

}

}

}

#### 其他成员所编写的函数

##### DataToString函数(黄雅雯编写)

//输入sql语句，数据信息string输出

public static string DataToString(string str)

{

using (SqlConnection conn = GetConn())

{

string sql = str;

SqlDataAdapter da = new SqlDataAdapter(sql, conn);

DataSet ds = new DataSet(); //创建数据集

conn.Open();

da.Fill(ds);

conn.Dispose();

conn.Close();

int n = ds.Tables[0].Rows.Count;

string a = "";

for (int i = 0; i < n; i++)

{

a += ds.Tables[0].Rows[i][0].ToString();

}

a = a.Replace("\\", "");

a = a.Replace(":\"[", ":[");

a = a.Replace("]\"}", "]}");

a = a.Replace("]\",", "],");

return a;

}

}

##### SaveFile函数(黄雅雯编写)

//写入并保存json文件

public static void SaveFile(string str1, string str2)

{

DirectoryInfo di = new DirectoryInfo(string.Format(@"{0}../../", Application.StartupPath));

string file = di.FullName + "data/" + str2 + ".json";

FileStream fs = new FileStream(file, FileMode.Create);

StreamWriter wr = new StreamWriter(fs);

wr.Write("dd(");

wr.Write(str1);

wr.Write(")");

wr.Close();

}

# 系统设计

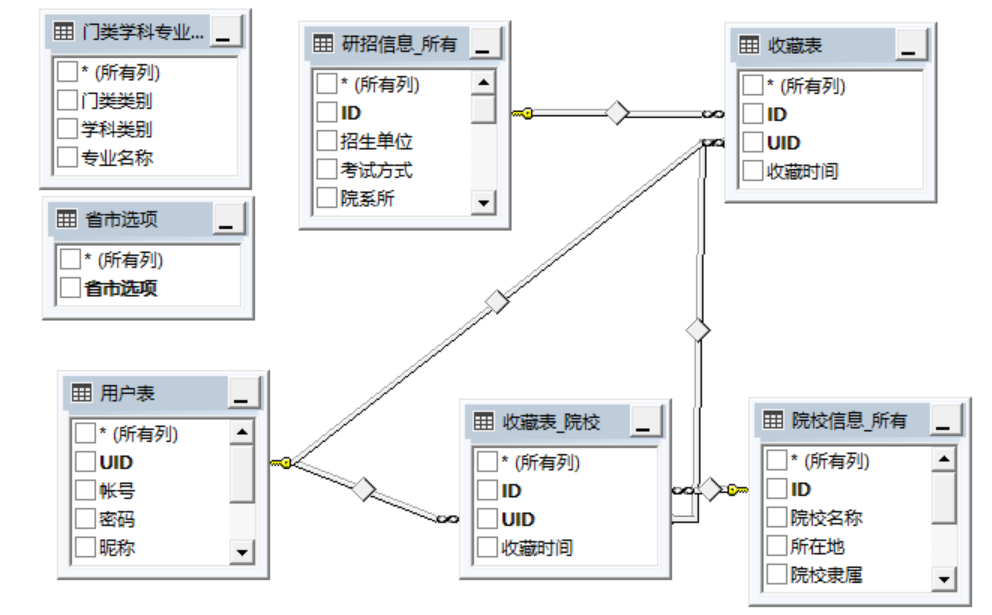


图 6.1 数据库整体结构

## 数据库设计

### 基本表

#### 门类学科专业选项

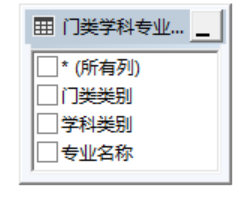


图 6.2 门类学科专业选项表

CREATE TABLE 门类学科专业选项 (

门类类别 VARCHAR(255) NOT NULL,

学科类别 VARCHAR(255) NOT NULL,

专业名称 VARCHAR(255) NOT NULL

);

此数据表用于存放门类类别、学科类别和专业名称的对应信息(其中内容如图6.3)，用于程序中动态地变换comboBox中的可选列表(图6.4)



图 6.3 门类学科专业选项表数据形式



图 6.4 动态变换的下拉列表

#### 省市选项

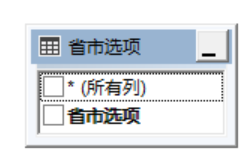


图 6.5 省市选项表

CREATE TABLE 省市选项 (

省市选项 VARCHAR(255) NOT NULL

);

为了填充“所在省市”下拉框而设计的表，这样做的目的是避免在程序中把它写死，显得极其臃肿

#### 研招信息\_所有

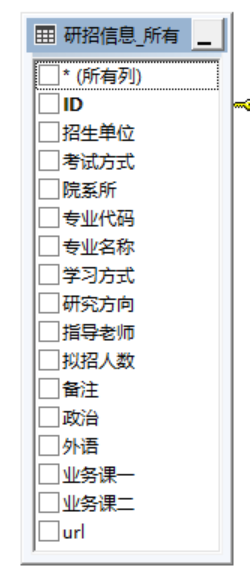


图 6.6 “研招信息\_所有”表

CREATE TABLE 研招信息\_所有 (

ID VARCHAR(255) NOT NULL,

招生单位 VARCHAR(255) NOT NULL,

考试方式 VARCHAR(255) NOT NULL,

院系所 VARCHAR(255) NOT NULL,

专业代码 VARCHAR(255) NOT NULL,

专业名称 VARCHAR(255) NOT NULL,

学习方式 VARCHAR(255) NOT NULL,

研究方向 VARCHAR(255) NOT NULL,

指导老师 VARCHAR(255) NOT NULL,

拟招人数 VARCHAR(255) NOT NULL,

备注 VARCHAR(255) NOT NULL,

政治 VARCHAR(255) NOT NULL,

外语 VARCHAR(255) NOT NULL,

业务课一 VARCHAR(255) NOT NULL,

业务课二 VARCHAR(255) NOT NULL,

url VARCHAR(255) NOT NULL,

PRIMARY KEY (ID)

);

注：这里的ID是url的后半部分截取下来的，这样设计是因为如果附上随机的编号，在下次数据维护过后，会使得收藏夹表中的条目，与实际情况不符，产生错位。但url是不会变动的，所以这将使得ID是固定的，也就不会产生错位的问题

#### 用户表

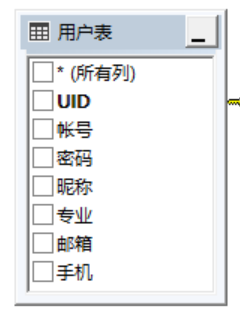


图 6.7 用户表

CREATE TABLE 用户表 (

UID INT NOT NULL,

帐号 VARCHAR(255) NOT NULL,

密码 VARCHAR(255) NOT NULL,

昵称 VARCHAR(255) NOT NULL,

专业 VARCHAR(255) NULL DEFAULT NULL,

邮箱 VARCHAR(255) NOT NULL,

手机 VARCHAR(255) NOT NULL,

PRIMARY KEY (UID),

INDEX 帐号(帐号)

);

除了存储帐号密码，用于存储用户的基本信息

在用户模式下登录，可以进行收藏和图表分析

#### 院校信息\_所有

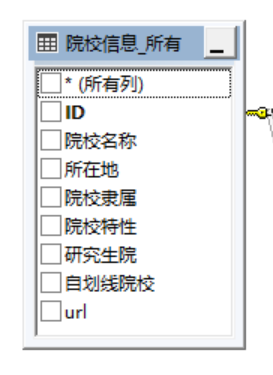


图 6.8 “院校信息\_所有”表

CREATE TABLE 院校信息\_所有 (

ID INT NOT NULL,

院校名称 VARCHAR(255) NOT NULL,

所在地 VARCHAR(255) NOT NULL,

院校隶属 VARCHAR(255) NOT NULL,

院校特性 VARCHAR(255) NOT NULL,

研究生院 bit NOT NULL,

自划线院校 bit NOT NULL,

url VARCHAR(255) NOT NULL,

PRIMARY KEY (ID)

);

这是储存招生院校信息的表，ID取自url后半段，原因与”研招信息\_所有”相同(详见于6.1.1.3)

#### 收藏表



图 6.9 收藏表

CREATE TABLE 收藏表 (

ID VARCHAR(255) NOT NULL,

UID INT NOT NULL,

收藏时间 date NOT NULL,

PRIMARY KEY(ID,UID),

FOREIGN KEY (UID) REFERENCES 用户表 (UID),

FOREIGN KEY (ID) REFERENCES 研招信息\_所有 (ID)

);

收藏表用于记录用户收藏的研招信息，实际使用时会使用视图(如图6.12)，这将更加方便

#### 收藏表\_院校



图 6.10 “收藏表\_院校”表

CREATE TABLE 收藏表\_院校 (

ID INT NOT NULL,

UID INT NOT NULL,

收藏时间 date NOT NULL,

PRIMARY KEY(ID,UID),

CONSTRAINT 收藏表\_院校\_ibfk\_1 FOREIGN KEY (ID) REFERENCES 院校信息\_所有 (ID),

CONSTRAINT 收藏表\_院校\_ibfk\_2 FOREIGN KEY (UID) REFERENCES 用户表 (UID)

);

收藏表\_院校用于记录用户收藏的院校信息，实际使用时同样会使用视图(如图6.11)

### 视图

#### 收藏表\_院校视图

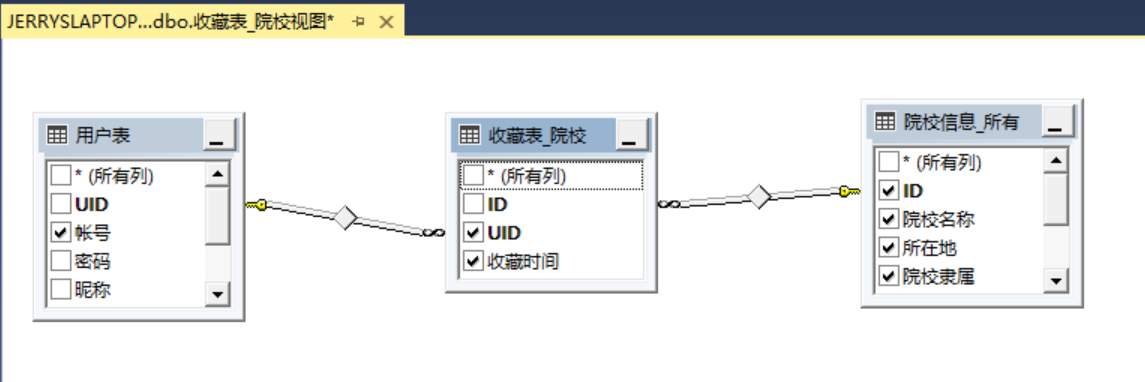


图 6.11 收藏表\_院校视图

CREATE VIEW 收藏表\_院校视图 AS(

SELECT 收藏表\_院校.UID AS UID,用户表.帐号 AS 帐号,院校信息\_所有.ID AS ID,院校信息\_所有.院校名称 AS 院校名称,院校信息\_所有.所在地 AS 所在地,院校信息\_所有.院校隶属 AS 院校隶属,院校信息\_所有.院校特性 AS 院校特性,院校信息\_所有.研究生院 AS 研究生院,院校信息\_所有.自划线院校 AS 自划线院校,院校信息\_所有.url AS url,收藏表\_院校.收藏时间 AS 收藏时间

FROM 收藏表\_院校,院校信息\_所有,用户表

WHERE ((收藏表\_院校.ID = 院校信息\_所有.ID) AND (收藏表\_院校.UID = 用户表.UID))

);

#### 收藏表视图

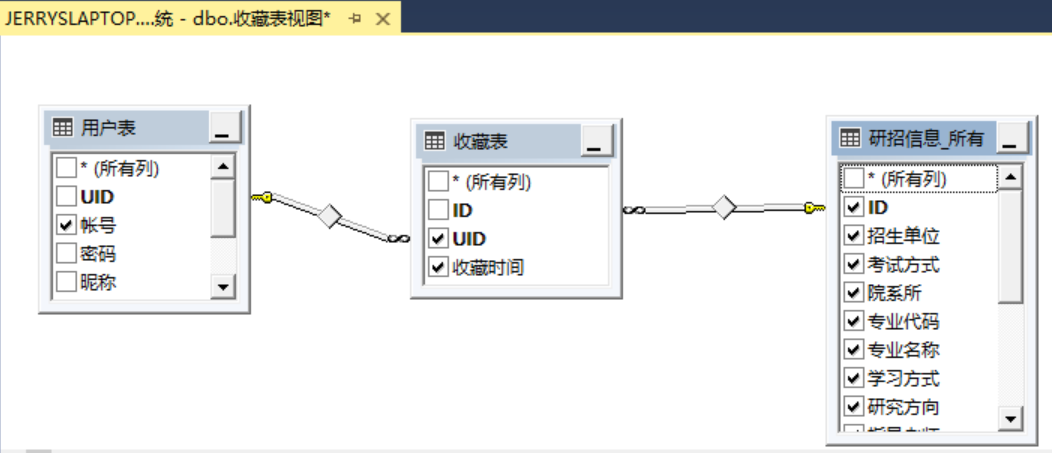


图 6.12 收藏表视图

CREATE VIEW 收藏表视图 AS(

SELECT 收藏表.UID AS UID,用户表.帐号 AS 帐号,研招信息\_所有.ID AS ID,研招信息\_所有.招生单位 AS 招生单位,研招信息\_所有.考试方式 AS 考试方式,研招信息\_所有.院系所 AS 院系所,研招信息\_所有.专业代码 AS 专业代码,研招信息\_所有.专业名称 AS 专业名称,研招信息\_所有.学习方式 AS 学习方式,研招信息\_所有.研究方向 AS 研究方向,研招信息\_所有.指导老师 AS 指导老师,研招信息\_所有.拟招人数 AS 拟招人数,研招信息\_所有.备注 AS 备注,研招信息\_所有.政治 AS 政治,研招信息\_所有.外语 AS 外语,研招信息\_所有.业务课一 AS 业务课一,研招信息\_所有.业务课二 AS 业务课二,研招信息\_所有.url AS url,收藏表.收藏时间 AS 收藏时间

FROM 研招信息\_所有,收藏表,用户表

WHERE ((研招信息\_所有.ID = 收藏表.ID) AND (用户表.UID = 收藏表.UID))

);

### 索引

是为了加快查询速度而创建了多条索引，如下：

CREATE UNIQUE INDEX 省市选项\_INDEX ON 省市选项(省市选项);

CREATE UNIQUE INDEX 研招信息\_所有\_ID ON 研招信息\_所有(ID);

CREATE UNIQUE INDEX 用户表\_UID ON 用户表(UID);

CREATE UNIQUE INDEX 院校信息\_所有\_ID ON 院校信息\_所有(ID);

CREATE UNIQUE INDEX 收藏表\_ID\_UID ON 收藏表(ID,UID);

CREATE UNIQUE INDEX 收藏表\_院校\_ID\_UID ON 收藏表\_院校(ID,UID);

### 数据库函数

注：以下展示的四个数据库函数均由黄雅雯编写，用以支持图表分析模块

CREATE FUNCTION [dbo].[gj](@str CHAR(50))

RETURNS NVARCHAR(MAX)

AS

BEGIN

RETURN(

SELECT 院校名称 name,院校特性,所在地,自划线院校,研究生院,dbo.学校院系(院校名称) academy

FROM 院校信息\_所有

WHERE 所在地=@str

GROUP BY 所在地,院校名称,院校特性,自划线院校,研究生院

FOR JSON AUTO

)

END

CREATE FUNCTION [dbo].[major](@str CHAR(50))

RETURNS NVARCHAR(MAX)

AS

BEGIN

RETURN(

SELECT 专业名称 name ,SUM(CONVERT(INT, SUBSTRING(拟招人数, PATINDEX('%[^0-9][0-9]%', 拟招人数) + 1, PATINDEX('%[0-9][^0-9]%', 拟招人数) - PATINDEX('%[^0-9][0-9]%', 拟招人数)))) value

FROM dbo.收藏表视图

WHERE 专业代码=@str

GROUP BY 专业代码,专业名称

FOR JSON AUTO

)

END

CREATE FUNCTION [dbo].[地区比例](@str CHAR(50))

RETURNS NVARCHAR(MAX)

AS

BEGIN

return(

SELECT 院校特性 name ,count(院校特性) value

FROM 院校信息\_所有

WHERE 所在地=@str

GROUP BY len(院校特性),院校特性

FOR JSON AUTO

)

END

CREATE FUNCTION [dbo].[学校院系](@str CHAR(50))

RETURNS NVARCHAR(MAX)

AS

BEGIN

return(

SELECT b 院系所 ,SUM(a) value

FROM(

SELECT CONVERT(INT, SUBSTRING(拟招人数, PATINDEX('%[^0-9][0-9]%', 拟招人数) + 1, PATINDEX('%[0-9][^0-9]%', 拟招人数) - PATINDEX('%[^0-9][0-9]%', 拟招人数))) AS a, 院系所 AS b ,招生单位 AS c

FROM 研招信息\_所有

WHERE 招生单位=@str

) AS 临时表

GROUP BY b

FOR JSON AUTO

)

END

## 爬虫程序设计

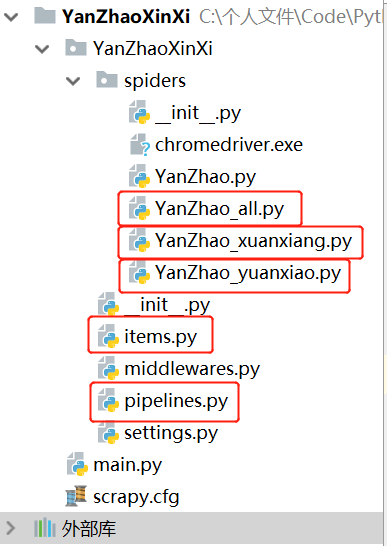


图 6.13 程序目录结构

本次爬虫程序的设计，考虑到了网页结构会有变更的问题，如果是小范围变更（如排版布局），本程序依然可以顺利运行，如果是大范围的变更（如css和id全部改变），则需要对爬虫程序进行微调（改写获取页面元素的css语句即可）。

### 通用驱动程序

以下代码用于驱动spider进行工作，作用相当于在cmd命令框中输入命令，代码写在main.py文件中，代码如下：

**from** scrapy.cmdline **import** execute *# 调用该函数以执行scrapy脚本***import** sys *# 为了加载到指定的目录***import** os *# 为了获取当前目录*sys.path.append(os.path.dirname(os.path.abspath(\_\_file\_\_)))  
  
*# execute(["scrapy", "crawl", "Yanzhao"])*execute([**"scrapy"**, **"crawl"**, **"Yanzhao\_all"**])  
*# execute(["scrapy", "crawl", "Yanzhao\_yuanxiao"])  
# execute(["scrapy", "crawl", "Yanzhao\_xuanxiang"])*

### 通用pipeline

当Item在Spider中被收集之后，它将会被传递到Item Pipeline，一些组件会按照一定的顺序执行对Item的处理。

每个item pipeline组件(有时称之为“Item Pipeline”)是实现了简单方法的Python类。他们接收到Item并通过它执行一些行为，同时也决定此Item是否继续通过pipeline，或是被丢弃而不再进行处理。

在本次设计中，编写了两个用于写入数据库的函数，分别为同步与异步方法，最初创建的同步方法仅在测试时进行过应用，实际运行时因效率不佳，改写成了异步方法，最终应用的是MysqlTwistedPipline这个Item pipeline

#### 同步写入数据库

**class** MysqlPipeline(object):  
 *# 采用同步的机制写入mysql* **def** \_\_init\_\_(self):  
 self.conn = MySQLdb.connect(host=**'localhost'**, user=**'root'**, passwd=**'root'**, db=**'YanZhaoXinXi'**, charset=**"utf8"**,  
 use\_unicode=**True**)  
 self.cursor = self.conn.cursor()  
  
 **def** process\_item(self, item, spider):  
 insert\_sql = item.get\_insert\_sql()  
 **try**:  
 self.cursor.execute(insert\_sql)  
 self.conn.commit()  
 **except**:  
 **pass**

#### 异步写入数据库

**class** MysqlTwistedPipline(object):  
 **def** \_\_init\_\_(self, dbpool):  
 self.dbpool = dbpool  
  
 @classmethod  
 **def** from\_settings(cls, settings):  
 dbparms = dict(  
 host=**'localhost'**,  
 user=**'root'**,  
 passwd=**'root'**,  
 db=**'YanZhaoXinXi'**,  
 charset=**"utf8"**,  
 use\_unicode=**True**,  
 cursorclass=MySQLdb.cursors.DictCursor,  
 )  
 dbpool = adbapi.ConnectionPool(**"MySQLdb"**, \*\*dbparms) *# 连接池* **return** cls(dbpool)  
  
 **def** process\_item(self, item, spider):  
 *# 使用twisted将mysql插入变成异步执行* query = self.dbpool.runInteraction(self.do\_insert, item) *# 这个函数会将第一个参数变成异步的* query.addErrback(self.handle\_error, item, spider) *# 处理异常，实际不用传这么多参数，请根据实际情况* **def** do\_insert(self, cursor, item):  
 *# 执行具体的插入  
 # 根据不同的item 构建不同的sql语句并插入到mysql中* insert\_sql = item.get\_insert\_sql() *# 这么写是因为在item中定义了生成插入语句的函数，可自行定义* cursor.execute(insert\_sql)  
 *# 这里和同步不同，不需要做commit()操作，会自动执行* **def** handle\_error(self, failure, item, spider):  
 *# 处理异步插入的异常* print(failure)

### 专业条目爬虫

此爬虫用于爬取专业条目信息，作用详见章节2.1.1，以下展示程序代码：

#### 专业条目爬虫主程序

*# -\*- coding: utf-8 -\*-***import** scrapy  
**from** selenium **import** webdriver  
**from** scrapy **import** Request  
**from** urllib **import** parse  
**from** YanZhaoXinXi.items **import** YanzhaoxinxiItem  
**import** re  
**import** time  
  
  
**class** YanZhaoSpider(scrapy.Spider):  
 name = **'Yanzhao\_all'** allowed\_domains = [**'yz.chsi.com.cn'**]  
 start\_urls = [**'http://yz.chsi.com.cn/zsml/queryAction.do'**]  
  
 **def** parse(self, response):  
 browser = webdriver.Chrome(executable\_path=**r'chromedriver.exe'**)  
 *# 打开搜索页面* browser.get(**"https://yz.chsi.com.cn/zsml/queryAction.do"**)  
 browser.maximize\_window()  
  
 **while True**:  
 nodes = browser.find\_elements\_by\_css\_selector(**'table.ch-table a'**)  
 **for** node **in** nodes:  
 url = node.get\_attribute(**'href'**)  
 **yield** Request(url=parse.urljoin(browser.current\_url, url), callback=self.parse\_deep)  
  
 *# 只有一页的情况* **if** (len(browser.find\_elements\_by\_css\_selector(**'ul.ch-page li.lip'**)) == 3):  
 **break** *# 有多页的情况，点击下一页，直到点不了（此时出现class=unable的li),写“或”是因为第一次会被识别，要让第一次的结果为真* **if** (len(browser.find\_elements\_by\_css\_selector(**'ul.ch-page li.lip.unable'**)) == 0 **or** len(browser.find\_elements\_by\_css\_selector(**'ul.ch-page li.lip.unable.lip-first'**)) == 1):  
 browser.find\_elements\_by\_css\_selector(**'ul.ch-page li.lip'**)[-2].click() *# 点击下一页* **else**:  
 **break** browser.quit() *# 关闭页面  
  
 # 处理某个学校的招生页面* **def** parse\_deep(self, response):  
 browser = webdriver.Chrome(**r'chromedriver.exe'**)  
 browser.get(response.url) *# 打开网页* browser.maximize\_window()  
 url\_zydm\_list = [] *# 用于存储元组：(url,zydm\_bh)* **while True**:  
 *# 考试范围url* urls\_node = browser.find\_elements\_by\_css\_selector(**'td.ch-table-center:nth-child(8) a'**)  
 zydm\_list = browser.find\_elements\_by\_css\_selector(**"table.ch-table tbody tr td:nth-child(3)"**)  
 **for** index **in** range(len(urls\_node)):  
 url = urls\_node[index].get\_attribute(**"href"**)  
 zydm\_bh = re.findall(**r"(\d+)"**, zydm\_list[index].text)[0] *# 编号* url\_zydm\_list.append((url, zydm\_bh)) *# 存入元组->(url,zydm\_bh)  
  
 # 只有一页的情况* **if** (len(browser.find\_elements\_by\_css\_selector(**'ul.ch-page li.lip'**)) == 3):  
 **break  
  
 if** (len(browser.find\_elements\_by\_css\_selector(**'ul.ch-page li.lip.lip-input-box'**)) != 1):  
 browser.find\_elements\_by\_css\_selector(**'ul.ch-page li.lip'**)[-1].click() *# 点击下一页  
 # 如果到了末尾，跳出循环* **if** (browser.find\_element\_by\_css\_selector(**'ul.ch-page li.lip.unable.lip-last'**) ==  
 browser.find\_elements\_by\_css\_selector(**'ul.ch-page li.lip'**)[-1]):  
 **break** *# 有多页的情况，点击下一页，直到点不了（此时出现class=unable的li),写“或”是因为第一次会被识别，要让第一次的结果为真* **elif** (len(browser.find\_elements\_by\_css\_selector(**'ul.ch-page li.lip.unable'**)) == 0 **or** len(browser.find\_elements\_by\_css\_selector(**'ul.ch-page li.lip.unable.lip-first'**)) == 1):  
 browser.find\_elements\_by\_css\_selector(**'ul.ch-page li.lip'**)[-2].click() *# 点击下一页* **else**:  
 **break** *# 关闭浏览器* browser.quit()  
  
 *# 放在最后yield是因为，如果放在中间，会生成很多浏览器窗口，直至内存承受不了  
 # 要让他一口气获取完所有url，并先关闭当前浏览器* **for** item **in** url\_zydm\_list:  
 url, zydm\_bh = item[0], item[1]  
 **yield** Request(url=parse.urljoin(response.url, url), meta={**"zydm\_bh"**: zydm\_bh}, callback=self.parse\_detail)  
  
 *# 某个学校的某个专业的考试范围页面* **def** parse\_detail(self, response):  
 my\_item = YanzhaoxinxiItem()  
 my\_item[**"db\_name"**] = **"研招信息\_所有"** my\_item[**"url"**] = response.url  
 my\_item[**"ID"**] = re.split(**"id="**, response.url)[1]  
 details = response.css(**'td.zsml-summary::text'**).extract()  
 *# 分别为：招生单位、考试方式、院系所、专业代码、学习方式、研究方向、指导老师、拟招人数* my\_item[**"zsdw"**] = details[0]  
 my\_item[**"ksfs"**] = details[1]  
 my\_item[**"yxs"**] = details[2]  
 my\_item[**"zydm\_bh"**] = response.meta[**"zydm\_bh"**]  
 my\_item[**"zydm\_mc"**] = details[3]  
 my\_item[**"xxfs"**] = details[4]  
 my\_item[**"yjfx"**] = details[5]  
 my\_item[**"zdls"**] = details[6]  
 **try**:  
 my\_item[**"nzrs"**] = details[7]  
 **except**:  
 *# zdls为空所导致的异常* my\_item[**"zdls"**] = **""** my\_item[**"nzrs"**] = details[6]  
  
 **try**:  
 my\_item[**"bz"**] = response.css(**'span.zsml-bz::text'**)[1].extract()  
 **except**:  
 *# 无备注时，赋值为空串* my\_item[**"bz"**] = **""** exams = response.css(**"tbody.zsml-res-items td::text"**).extract()  
 exams = [item.replace(**'\r\n'**, **''**).replace(**' '**, **''**) **for** item **in** exams]  
 my\_item[**"zhengzhi"**] = exams[0]  
 my\_item[**"waiyu"**] = exams[2]  
 my\_item[**"yewuke\_1"**] = exams[3]  
 my\_item[**"yewuke\_2"**] = exams[4]  
  
 **yield** my\_item

#### 专业条目爬虫Item

为了保存数据，就必须将Item传入pipeline，因此也就必须编写一个对应的Item，用于保存关键信息，在本次设计中，为每个Item都编写了get\_insert\_sql方法，用于构造插入数据库的sql语句，这样就使得可以重复调用同一个pipeline进行工作，避免每个爬虫程序，就要写专用的pipeline的问题，以下将展示代码：

**class** YanzhaoxinxiItem(scrapy.Item):  
 *# define the fields for your item here like:  
 # name = scrapy.Field()* ID = scrapy.Field() *# ID:截取url后面一串字串* db\_name = scrapy.Field() *# 表名* zsdw = scrapy.Field() *# 招生单位* ksfs = scrapy.Field() *# 考试方式* yxs = scrapy.Field() *# 院系所* zydm\_bh = scrapy.Field() *# 专业代码\_编号* zydm\_mc = scrapy.Field() *# 专业代码\_名称* xxfs = scrapy.Field() *# 学习方式* yjfx = scrapy.Field() *# 研究方向* zdls = scrapy.Field() *# 指导老师* nzrs = scrapy.Field() *# 拟招人数* bz = scrapy.Field() *# 备注* zhengzhi = scrapy.Field() *# 政治* waiyu = scrapy.Field() *# 外语* yewuke\_1 = scrapy.Field() *# 业务课一* yewuke\_2 = scrapy.Field() *# 业务课二* url = scrapy.Field()  
  
 **def** get\_insert\_sql(self):  
 sql = **"INSERT INTO {0}(ID,招生单位,考试方式,院系所,专业代码,专业名称,学习方式,研究方向,指导老师,拟招人数,备注,政治,外语,业务课一,业务课二,url) "** \  
 **"VALUES('{1}','{2}','{3}','{4}','{5}','{6}','{7}','{8}','{9}','{10}','{11}','{12}','{13}','{14}','{15}','{16}')"**.format(  
 self[**"db\_name"**], self[**"ID"**], self[**"zsdw"**], self[**"ksfs"**], self[**"yxs"**], self[**"zydm\_bh"**], self[**"zydm\_mc"**],  
 self[**"xxfs"**], self[**"yjfx"**], self[**"zdls"**], self[**"nzrs"**], self[**"bz"**], self[**"zhengzhi"**], self[**"waiyu"**],  
 self[**"yewuke\_1"**], self[**"yewuke\_2"**], self[**"url"**])  
 **return** sql

### 院校爬虫

此爬虫用于爬取院校信息，作用详见章节2.1.2，以下展示程序代码：

#### 院校爬虫主程序

*# -\*- coding: utf-8 -\*-***import** scrapy  
**from** selenium **import** webdriver  
**from** scrapy **import** Request  
**from** urllib **import** parse  
**from** YanZhaoXinXi.items **import** YuanxiaoxinxiItem  
**import** re  
  
  
**class** YanZhaoSpider(scrapy.Spider):  
 name = **'Yanzhao\_yuanxiao'** allowed\_domains = [**'yz.chsi.com.cn'**]  
 start\_urls = [**'https://yz.chsi.com.cn/sch/'**]  
  
 **def** parse(self, response):  
 browser = webdriver.Chrome(**r'chromedriver.exe'**)  
 *# 打开搜索页面* browser.get(**"https://yz.chsi.com.cn/sch/"**)  
 browser.maximize\_window()  
  
 **while True**:  
 count = len(browser.find\_elements\_by\_css\_selector(**'table.ch-table tbody tr'**))  
 **for** index **in** range(1, count + 1):  
 my\_item = YuanxiaoxinxiItem()  
 my\_item[**"db\_name"**] = **"院校信息\_所有"** my\_item[**"url"**] = browser.find\_element\_by\_css\_selector(  
 **'table.ch-table tbody tr:nth-child({0}) td:nth-child({1}) a'**.format(index, 1)).get\_attribute(**"href"**)  
 my\_item[**"ID"**] = re.findall(**r'schId-(\d+)\.dhtml'**, my\_item[**"url"**])[0]  
 my\_item[**"YXMC"**] = browser.find\_element\_by\_css\_selector(  
 **'table.ch-table tbody tr:nth-child({0}) td:nth-child({1})'**.format(index, 1)).text  
 my\_item[**"SZD"**] = browser.find\_element\_by\_css\_selector(  
 **'table.ch-table tbody tr:nth-child({0}) td:nth-child({1})'**.format(index, 2)).text  
 my\_item[**"YXLS"**] = browser.find\_element\_by\_css\_selector(  
 **'table.ch-table tbody tr:nth-child({0}) td:nth-child({1})'**.format(index, 3)).text  
 my\_item[**"YXTX"**] = browser.find\_element\_by\_css\_selector(  
 **'table.ch-table tbody tr:nth-child({0}) td:nth-child({1})'**.format(index, 4)).text  
 YJSY = browser.find\_elements\_by\_css\_selector(  
 **'table.ch-table tbody tr:nth-child({0}) td:nth-child({1}) i'**.format(index, 5))  
 **if** (len(YJSY) == 0):  
 my\_item[**"YJSY"**] = **'False'  
 else**:  
 my\_item[**"YJSY"**] = **'True'** ZHXYX = browser.find\_elements\_by\_css\_selector(  
 **'table.ch-table tbody tr:nth-child({0}) td:nth-child({1}) i'**.format(index, 6))  
 **if** (len(ZHXYX) == 0):  
 my\_item[**"ZHXYX"**] = **'False'  
 else**:  
 my\_item[**"ZHXYX"**] = **'True'  
  
 yield** my\_item  
  
 *# 只有一页的情况* **if** (len(browser.find\_elements\_by\_css\_selector(**'ul.ch-page li.lip'**)) == 3):  
 **break** *# 有多页的情况，点击下一页，直到点不了（此时出现class=unable的li),写“或”是因为第一次会被识别，要让第一次的结果为真* **if** (len(browser.find\_elements\_by\_css\_selector(**'ul.ch-page li.lip.unable'**)) == 0 **or** len(browser.find\_elements\_by\_css\_selector(**'ul.ch-page li.lip.unable.lip-first'**)) == 1):  
 browser.find\_elements\_by\_css\_selector(**'ul.ch-page li.lip'**)[-2].click() *# 点击下一页* **else**:  
 **break** browser.quit() *# 关闭页面*

#### 院校爬虫Item

作用如专业条目爬虫Item，详见章节6.2.3.2：

**class** YuanxiaoxinxiItem(scrapy.Item):  
 db\_name = scrapy.Field() *# 表名* ID = scrapy.Field() *# ID:截取url后面一串数字* YXMC = scrapy.Field() *# 院校名称* SZD = scrapy.Field() *# 所在地* YXLS = scrapy.Field() *# 院校隶属* YXTX = scrapy.Field() *# 院校特性* YJSY = scrapy.Field() *# 研究生院* ZHXYX = scrapy.Field() *# 自主划线院校* url = scrapy.Field()  
  
 **def** get\_insert\_sql(self):  
 sql = **"INSERT INTO {0}(ID,院校名称,所在地,院校隶属,院校特性,研究生院,自划线院校,url) VALUES('{1}','{2}','{3}','{4}','{5}',{6},{7},'{8}');"**.format(  
 self[**"db\_name"**], self[**"ID"**], self[**"YXMC"**], self[**"SZD"**], self[**"YXLS"**], self[**"YXTX"**], self[**"YJSY"**],  
 self[**"ZHXYX"**], self[**"url"**])  
 **return** sql

### 院校选项爬虫

为了实现下拉列表的动态变化（例如选择08工科，会出现其对应的门类，如果不这么做，会使得搜索及其没有效率），需要爬取所有对应的可能情况，因此爬取了网页中的下拉列表，存入数据库中

#### 院校选项爬虫主程序

*# -\*- coding: utf-8 -\*-***import** scrapy  
**from** selenium **import** webdriver  
**from** scrapy **import** Request  
**from** urllib **import** parse  
**from** YanZhaoXinXi.items **import** ChaxunxuanxiangItem  
**import** re  
  
  
**class** YanZhaoSpider(scrapy.Spider):  
 name = **'Yanzhao\_xuanxiang'** allowed\_domains = [**'yz.chsi.com.cn'**]  
 start\_urls = [**'https://yz.chsi.com.cn/sch/'**]  
  
 **def** parse(self, response):  
 browser = webdriver.Chrome(**r'chromedriver.exe'**)  
 *# 打开搜索页面* browser.get(**"https://yz.chsi.com.cn/zsml/zyfx\_search.jsp"**)  
 browser.maximize\_window()  
  
 *# 1.提取"门类类别"下拉框* MLLB\_list = browser.find\_elements\_by\_css\_selector(**"select#mldm option"**)  
 **for** MLLB **in** MLLB\_list:  
 *# 2.选中"门类类别"* browser.find\_element\_by\_css\_selector(**"select#mldm"**).click()  
 MLLB.click()  
  
 *# 3.提取"学科类别"下拉框* XKLB\_list = browser.find\_elements\_by\_css\_selector(**"select#yjxkdm option"**)  
 **for** XKLB **in** XKLB\_list:  
 *# 4.选中"学科类别"* browser.find\_element\_by\_css\_selector(**"select#yjxkdm"**).click()  
 XKLB.click()  
  
 *# 5.提取"专业名称"下拉框* ZYMC\_list = browser.find\_elements\_by\_css\_selector(**"select#zymc option"**)  
 **for** ZYMC **in** ZYMC\_list:  
 my\_item = ChaxunxuanxiangItem()  
 my\_item[**"db\_name"**] = **"门类学科专业选项"** my\_item[**"MLLB"**] = MLLB.text  
 my\_item[**"XKLB"**] = XKLB.text  
 my\_item[**"ZYMC"**] = ZYMC.text  
 **yield** my\_item  
  
 citys = browser.find\_elements\_by\_css\_selector(**"select#ssdm option"**)  
 citys\_text = [] *# 这样操作是为了，在控制台中，直接获取城市列表，如果不分，会夹杂其他提示信息* **for** city **in** citys:  
 citys\_text.append(city.text)  
 **for** city **in** citys\_text:  
 print(city)  
 browser.quit() *# 关闭页面*

#### 院校选项爬虫Item

**class** ChaxunxuanxiangItem(scrapy.Item):  
 db\_name = scrapy.Field() *# 表名* MLLB = scrapy.Field() *# 门类类别* XKLB = scrapy.Field() *# 学科类别* ZYMC = scrapy.Field() *# 专业名称* **def** get\_insert\_sql(self):  
 sql = **"INSERT INTO {0}(门类类别,学科类别,专业名称) VALUES('{1}','{2}','{3}');"**.format(self[**"db\_name"**], self[**"MLLB"**],self[**"XKLB"**], self[**"ZYMC"**])  
 **return** sql

## 客户端程序设计

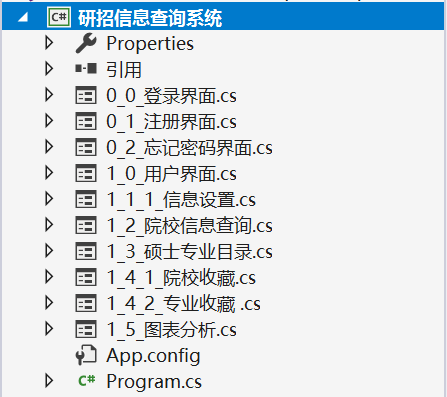


图 6.14 程序目录结构登录界面



图 6.15 登录界面

提供两种登录模式：

1. 输入帐号密码，以用户身份登录
2. 以游客身份登录(无需提供帐号密码)

登录后将跳转至用户主界面(如图6.6)

点击“注册帐号”即可跳转至注册界面(如图6.4)

点击“忘记密码界面”即可跳转至找回密码界面(如图6.5)

### 注册界面



图 6.16 注册界面

按提示输入所有的信息以进行注册，期间将对手机、邮箱进行验证

### 找回密码界面



图 6.17 找回密码界面

输入正确的帐号、手机和邮箱，输入新密码，验证成功后修改生效

### 用户主界面



图 6.18 用户主界面

### 信息设置界面



图 6.19 信息设置界面

可在此页面修改密码和基本信息

### 院校信息查询界面



图 6.20 院校信息查询界面

可在此页面可根据所在省市、院校隶属和院校特性查询相符的学校，也可以通过输入招生单位，进一步细致查询

双击某一行可进行收藏，单击url一栏可对具体的网页进行复制

### 硕士专业目录界面



图 6.21 硕士专业目录界面

可在此页面根据所在省市、门类类别、学科类别、专业名称、学习方式、院校特性等条件进行查询

也可以输入招生单位进行细致查询

更可以细致到按考试科目进行查询

另外，门类类别、学科类别、专业名称这三项是联动的，会根据上一级的选择，而填充不同内容

政治、外语、业务课一、业务课二也是联动的，会根据按当前条件所能查询到的院校，来显示所能选择的考试科目

### 院校/专业收藏界面



图 6.22 院校收藏界面



图 6.23 专业收藏界面

在院校/专业收藏界面中，可以查看所收藏的院校和专业，收藏夹中的内容可以在图表分析界面中进行分析

### 图表分析界面

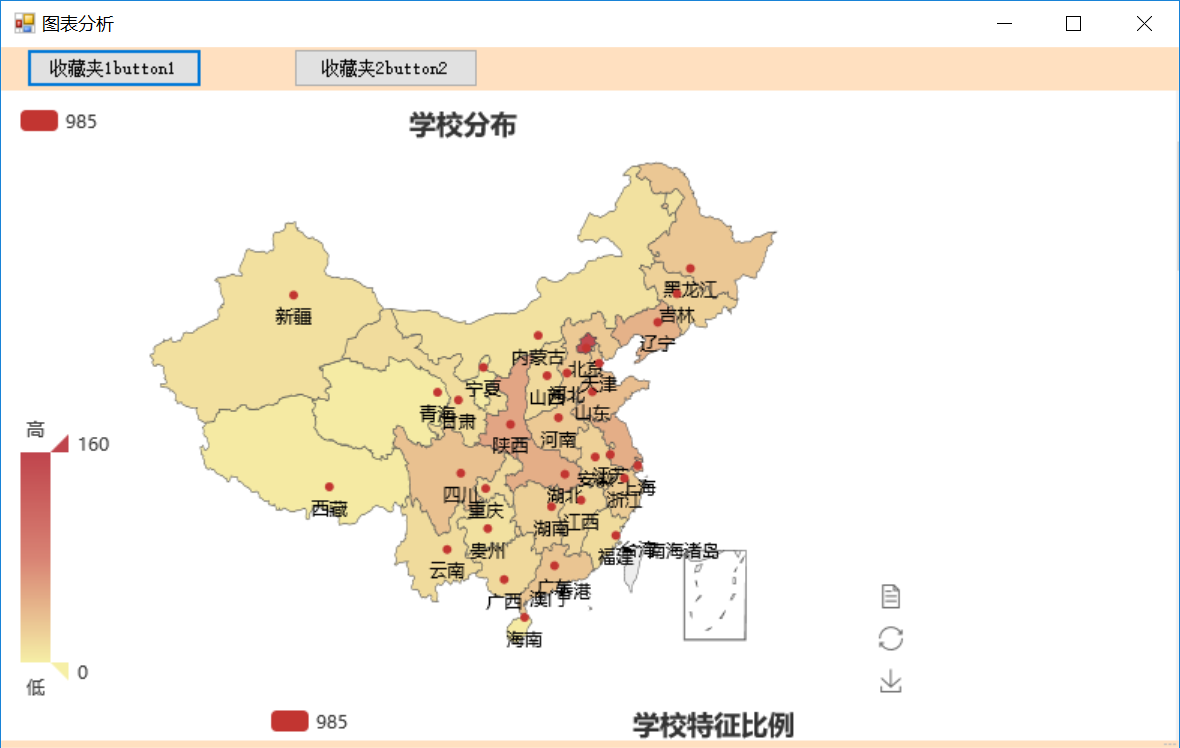


图 6.24 图表分析界面

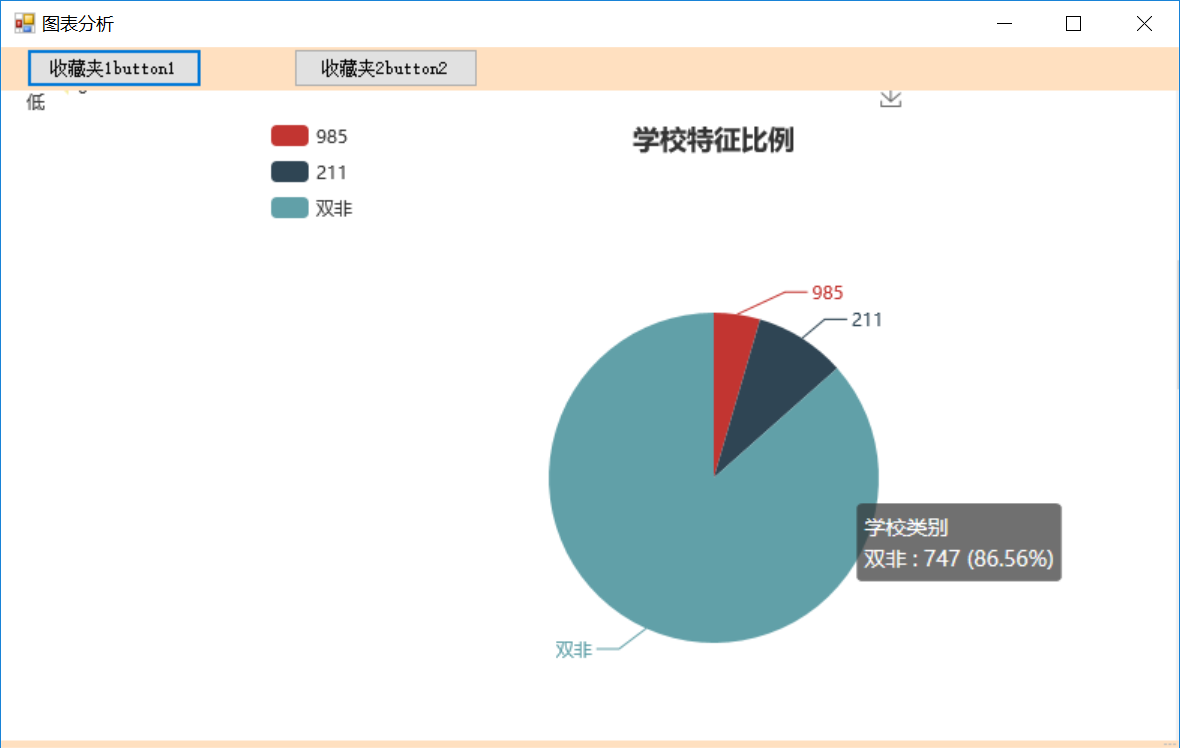


图 6.25 学校特征比例

在此界面中，可以分别对收藏夹中的院校和专业进行图表化的分析，包括在地图上标示分布和比例分布情况等

## 命名规范的设计

### 窗体的命名规范

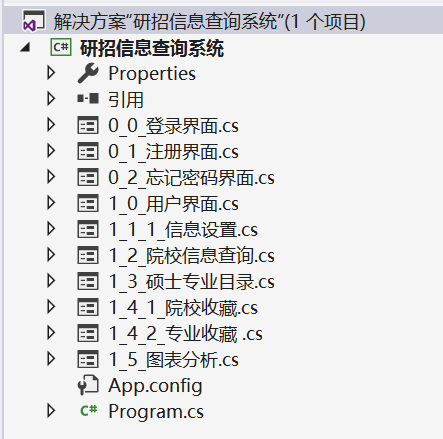


图 6.26 窗体的命名规范

如图所示，所有窗体均按照”数字1\_数字2\_数字3\_名称”的规范进行命名。其中：

数字1的0、1分别表示登录之前和以用户身份登录

数字2则与这个窗体在主界面菜单栏上的次序相对应

数字3则与菜单栏的下拉列表中的次序相对应。

### 控件的命名规范



图 6.27 控件命名的规范

本系统中，所有控件如上图所示方式命名：前半段标示其功能，后半段则标示控件类型

# 小组执行情况

## 分工情况

### 大致分工

黄雅雯：图表分析模块（已完成）  
林佳琰：收藏夹模块（已完成）  
阮如刚：网页端全部内容（已完成）  
张俊帆：爬虫程序（已完成）、数据库搭建（已完成）、SQLHelper数据库操作类(已完成)、C#工程初步结构(已完成)、两个查询模块(已完成)、云端数据库（放弃）

### 预计完成时间

黄雅雯、林佳琰：大约3周  
阮如刚：大约4周  
张俊帆：大约1周

注：每周须交流一次进度

## 开发进度

### 第一周

张俊帆：

~~数据表有待修改：主键、属性类型等~~

~~两个查询模块：按下拉框所选定条件查询~~

~~下拉框变更部分：根据其他下拉框的选定内容进行变更~~

云端数据库

### 第二周

黄雅雯：

~~图表分析模块初步完成~~

林佳琰：

~~收藏夹模块初步完成~~

阮如刚：

~~网页端：完成两个查询的搜索功能~~

张俊帆：

~~爬虫程序修复：修复学校只选取第一页30条招生项目bug~~

~~主程序修改~~

~~信息设置模块完成~~

~~建立云端数据库(没钱，暂时搁置)~~

### 第三周

张俊帆：

~~给"研招信息所有"和"院校信息所有"加主键~~

~~修改爬虫，使爬虫程序和修改后的数据库配套~~

~~修"收藏表"和"收藏表\_院校"~~

~~新增"收藏表视图"和"收藏表\_院校视图"~~

~~重启爬虫程序，导出SQLserver接受的sql文件~~

~~合并其他人的开发文件~~

### 第四周

张俊帆：

~~实现更加精细化的搜索功能（按考试科目搜索）~~

## 部分开发日志

--2019.4.10改动：

--爬虫修改完毕，将数据导入到"研招信息\_所有"中

--2019.4.17改动：

--修改用户表,增加属性:UID、昵称、专业、邮箱、手机,并设置UID为主键

--2019.4.18改动：

--爬虫再次调整，补充到14万条研招数据

--2019.4.27改动：

--为"研招信息\_所有"增加主键"ID"("ID"来自于url字段的最后部分)

--清空"研招信息\_所有"，改动爬虫程序，并再次运行以填充数据

--2019.4.28改动：

--改动爬虫程序，提取页面中院校url，截取其中的一段数字作为ID

--为"院校信息\_所有"增加"ID"字段和"url"字段

--清空"院校信息\_所有"，运行爬虫程序，重新填充数据

--新增"收藏表"和"收藏表\_院校"数据表，各自仅包含ID和UID两个字段

--新增"收藏表视图"，将"收藏表","用户表"","研招信息\_所有"连接在一起：((研招信息\_所有.ID = 收藏表.ID) AND (用户表.UID = 收藏表.UID))

--新增"收藏表\_院校视图"，将"收藏表","用户表"","院校信息\_所有"连接在一起((收藏表\_院校.ID = 院校信息\_所有.ID) AND (收藏表\_院校.UID = 用户表.UID))

--2019.5.13改动（黄雅雯）：

--创建两个数据库函数：gj，major，地区比例，学校院系

--2019.5.19改动：

--创建索引：省市选项\_INDEX，研招信息\_所有\_ID,用户表\_UID，院校信息\_所有\_ID，收藏表\_ID\_UID，收藏表\_院校\_ID\_UID

以上开发日志主要为数据库方面的开发日志，有一点记一点，有些地方有缺漏

# 总结与展望

刚开始，这只是个为了满足自用，而编写的小型爬虫程序，本次开发任务中对它进行了全面的修改，完善了爬虫程序，修改了不合理的数据库设计，单凭可以依据考试科目进行查询这一项，就为使用者提供了非常大的便利性，搭配收藏功能和图表分析功能和网页端的程序，完全可以开发成为一个小型的付费项目，已具备了商业化的雏形。

不仅如此，通过本次开发任务，增进了对于开发进度和协调分工方面的能力，相比于之前的课程设计的开发情况来说，已有了较大提升，能够更加精细地按周对进度进行管理，并理解了程序架构对于开发进度和分工的重要性。

# 参考文献

1. [9787040406641]王珊,萨师煊,《数据库系统概论》,高等教育出版社，2014-09
2. [9787300210513]胡西川,《VisualC#.NET框架程序设计》，中国人民大学出版社，2015-05
3. Scrapy入门教程,<https://scrapy-chs.readthedocs.io/zh_CN/latest/intro/tutorial.html>
4. selenium之 chromedriver与chrome版本映射表,<https://blog.csdn.net/huilan_same/article/details/51896672>
5. MySQL数据库安装与配置详解,<http://www.cnblogs.com/sshoub/p/4321640.html>
6. 国内的pythoner强烈建议使用豆瓣的pypi源,<https://www.douban.com/note/302711300/>
7. Python虚拟环境virtualenv的安装与使用详解,<https://www.jb51.net/article/114933.htm>
8. Unofficial Windows Binaries for Python Extension Packages,<https://www.lfd.uci.edu/~gohlke/pythonlibs/>
9. Python 正则表达式,<https://www.runoob.com/python/python-reg-expressions.html>
10. CSS 选择器参考手册,<http://www.w3school.com.cn/cssref/css_selectors.asp>
11. scrapy简书技术博客爬虫,<https://www.jianshu.com/p/9e6433f91c73>
12. 网络爬虫Scrapy从入门到进阶,<https://www.jianshu.com/p/be856bc15afb>
13. Scrapy简单入门及实例讲解,<https://blog.csdn.net/kaikai136412162/article/details/78971122>