云南大学软件学院

实 验 报 告

课程： 大数据分析及应用实验 任课教师： 何婧

专业： 人工智能 学号： 20211060245 姓名： 陈俊宏 成绩： 陈俊宏

专业： 人工智能 学号： 20211120171 姓名： 孔令高 成绩：

# 实验三 数据获取

## 一、 实验目的

1、掌握爬虫的基本原理。

2．能熟练使用一种编程语言，如：Java、Python、C++等从网站上下载需要的

数据。

## 二、 实验内容及步骤

**1、网络爬虫**

网络爬虫是一个自动提取网页的程序，它为[搜索引擎](https://baike.baidu.com/item/%E6%90%9C%E7%B4%A2%E5%BC%95%E6%93%8E)从万维网上下载网页，是搜索引擎的重要组成。传统爬虫从一个或若干初始网页的URL开始，获得初始网页上的URL，在抓取网页的过程中，不断从当前页面上抽取新的URL放入队列,直到满足系统的一定停止条件。聚焦爬虫的工作流程较为复杂，需要根据一定的网页分析算法过滤与主题无关的链接，保留有用的链接并将其放入等待抓取的URL队列。然后，它将根据一定的搜索策略从队列中选择下一步要抓取的网页URL，并重复上述过程，直到达到系统的某一条件时停止。另外，所有被爬虫抓取的网页将会被系统存贮，进行一定的分析、过滤，并建立索引，以便之后的查询和检索；对于聚焦爬虫来说，这一过程所得到的分析结果还可能对以后的抓取过程给出反馈和指导。

**2、具体内容**

**示例：**

从百度健康或者快速问医生或者别的网站上爬取关于抑郁症的咨询及回答，咨询数不得少于300条，快速问医生的网站为：http://so.120ask.com/，请自行选择想要爬取有关的咨询及回答，最后保存为csv。

P.S.在百度健康上爬取数据首先在百度搜索上输入问题，等获得链接后在进行爬取，建议选取的关键词有失眠、抑郁等，建议使用python语言，因为可使用三方库selenium进行爬取，也可以不使用。

先进行搜索得到网址在对信息进行爬取



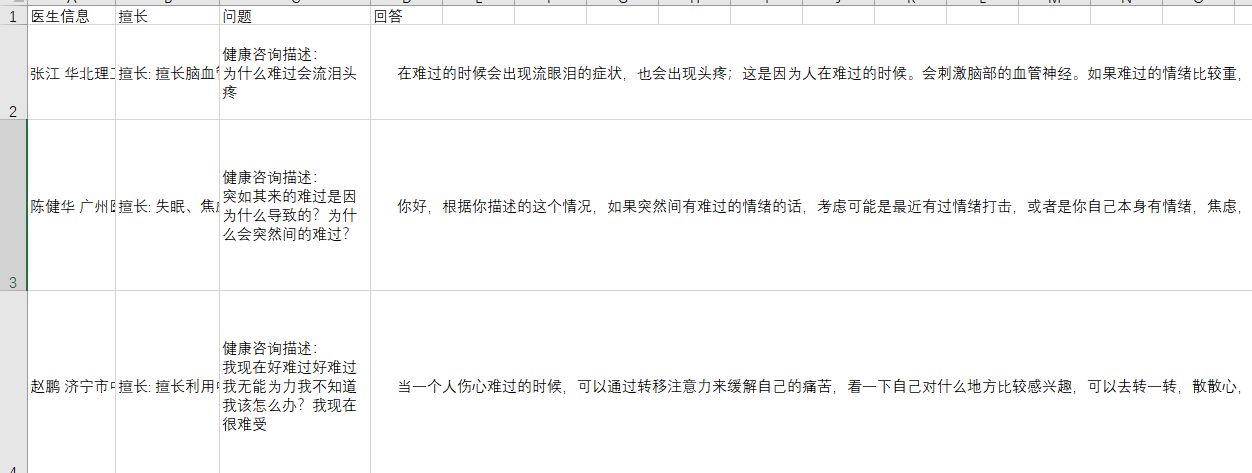


这些网站的问答内容均简单的存储在HTML页面中，因此只需要将爬取的网页数据进行解析和整理即可，数据的格式如下图所示：

1、百度健康的爬取格式示例



2、快速问医生的示例：



**要求：**

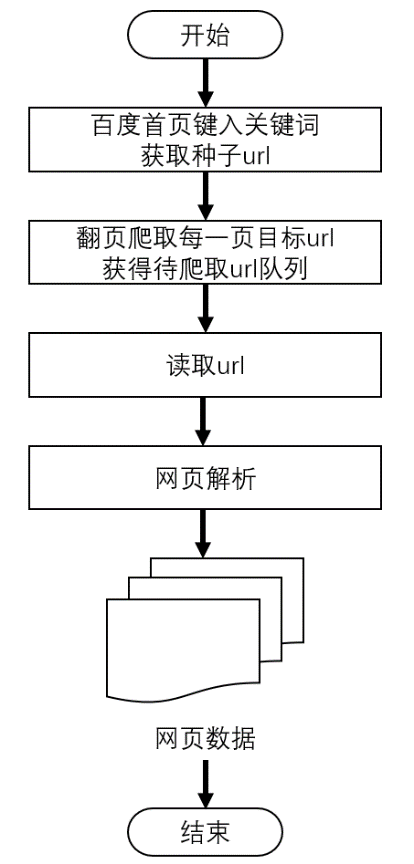
（1）爬取与“抑郁症”相关的问答，问答数不少于300条。

（2）实验结束后提交实验报告、源代码、爬取的数据。

## 三、 算法描述（原理+算法流程图+关键代码描述）

原理：首先使用selenium在百度首页baidu.com的搜索框中键入“抑郁 inurl:health.baidu.com”用以搜索包含“抑郁”关键词的“百度健康”条目，而后将每一条”@mu”属性以”[https://health.baidu.com/m/detail/ar”（ar](https://health.baidu.com/m/detail/ar)后缀用以筛选问答形式，过滤问诊形式和视频形式）开头的条目的”@mu”属性值爬取下来（mu属性为搜索条目真实链接），不断重复翻页爬取的过程，爬取到最后一页后停止爬取，并保存到”url.json”文件中，而后通过解析超链接，将我们想要的内容爬取下来，保存为csv文件，在数据清洗过后，最终得到爬取数据。

算法流程图



关键代码描述：

在键入关键词并爬取目标url队列的过程中，因为涉及到翻页问题，所以一开始我采取的是通过获取翻页按钮url，新建标签页并关闭原标签页的方式爬取，代码如下：

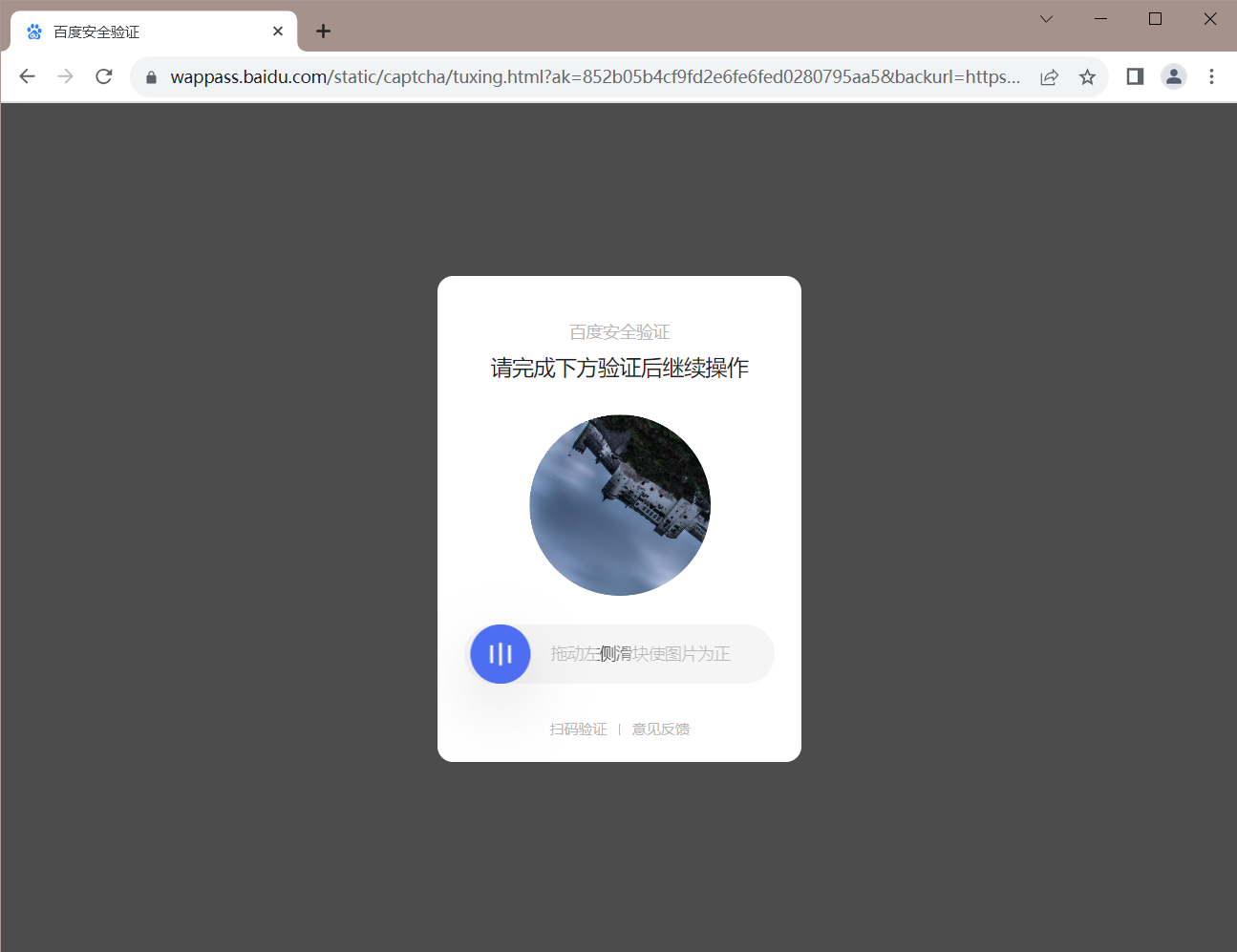
*# 下一页*page\_element = browser.find\_elements(By.XPATH,  
 "//div[@id='page']/div[@class='page-inner\_2jZi2']/strong/following-sibling::a[1]")

next\_page = page\_element[0].get\_attribute('href')  
*# 为下一页创建新的标签页*browser.execute\_script(f'window.open("{next\_page}","\_blank");')  
*# 关闭当前标签页*browser.close()  
*# 切换到新标签页*browser.switch\_to.window(browser.window\_handles[-1])

但是随之开始，多次跳出百度安全验证，触发了百度反爬措施，在经过多种尝试无效之后，我恍然大悟：按照该方法，等于我在未输入关键词搜索的情况下，直接获取了结果，极大概率判定为爬虫，于是改进翻页方法为模拟点击翻页，成功爬取url队列，代码如下：

*# 预防StaleElementReferenceException, ElementClickInterceptedException*temp = 0  
*while* temp < 2:  
 *try*:  
 *# 下一页* page\_element = browser.find\_elements(By.XPATH,  
 "//div[@id='page']/div[@class='page-inner\_2jZi2']/strong/following-sibling::a[1]")  
 *if* len(page\_element) == 0:  
 last = *True  
 # 无下一页，则退出循环* print("已是最后一页")  
 *# 关闭浏览器* browser.close()  
 *break  
 else*:  
 page\_element[0].click()  
 browser.implicitly\_wait(5)  
 *break  
 except* StaleElementReferenceException:  
 temp += 1  
 *continue  
 except* ElementClickInterceptedException:  
 temp += 1  
 *continue*

之后开始进行对url队列的爬取，然而此时，又出现了窒息的画面：



于是我开始研究怎么反制百度的反爬措施，在浏览多篇博客和文章后，初步估计修改header，创建IP代理池以及随机cookie池的常用方法，均无效，甚至怀疑不能单独爬取url队列，需要模拟点击一条一条爬取，在尝试过后也失败。于是参考<https://github.com/ShortCJL/RotateCode>的深度学习解决方案，已经准备开始大干一场了，但是无意中看到一篇博客，说重置一次谷歌浏览器就可以了，于是我抱着活马当死马医的心态，尝试了一次，竟然成功了！核心代码：

*# 标题*title = browser.find\_element(By.XPATH,  
 "//div[@id='spread-fold']//p[@class='index\_healthTitle\_\_lpfdm hc-line-clamp2']")  
questions.append(title.text)  
  
*# 回答*content = browser.find\_elements(By.XPATH,  
 "//div[@id='spread-fold']//div[@class='index\_textContent\_\_CjNhL index\_richText\_\_OrVb1 index\_richTextPc\_\_IEcLL']/p")  
texts = "".join([i.text *for* i *in* content])  
answers.append(texts)  
  
*# 医生信息*doctor = browser.find\_elements(By.XPATH,  
 "//div[@id='spread-fold']//p[@class='index\_nameInfo\_\_9H6bC hc-line-clamp1']/span[position()<=2]")  
doc\_info = "".join([i.text *for* i *in* doctor])  
doc\_infos.append(doc\_info)  
  
*# 医院信息*hospital\_name = browser.find\_element(By.XPATH,  
 "//div[@id='spread-fold']//span[@class='index\_hosName\_\_EDr05 hc-line-clamp1']")  
hospital\_level = browser.find\_element(By.XPATH,  
 "//div[@id='spread-fold']//div[@class='index\_container\_\_aTDkB']/span[position()=1]")  
hospital\_infos.append(hospital\_name.text + "\n" + hospital\_level.text)

在爬虫完成后，由于文件中存在空值填充的情况，且有些问答网页，采取的是病历形式，并不是直观的问答形式，于是进行了数据清洗，代码如下：

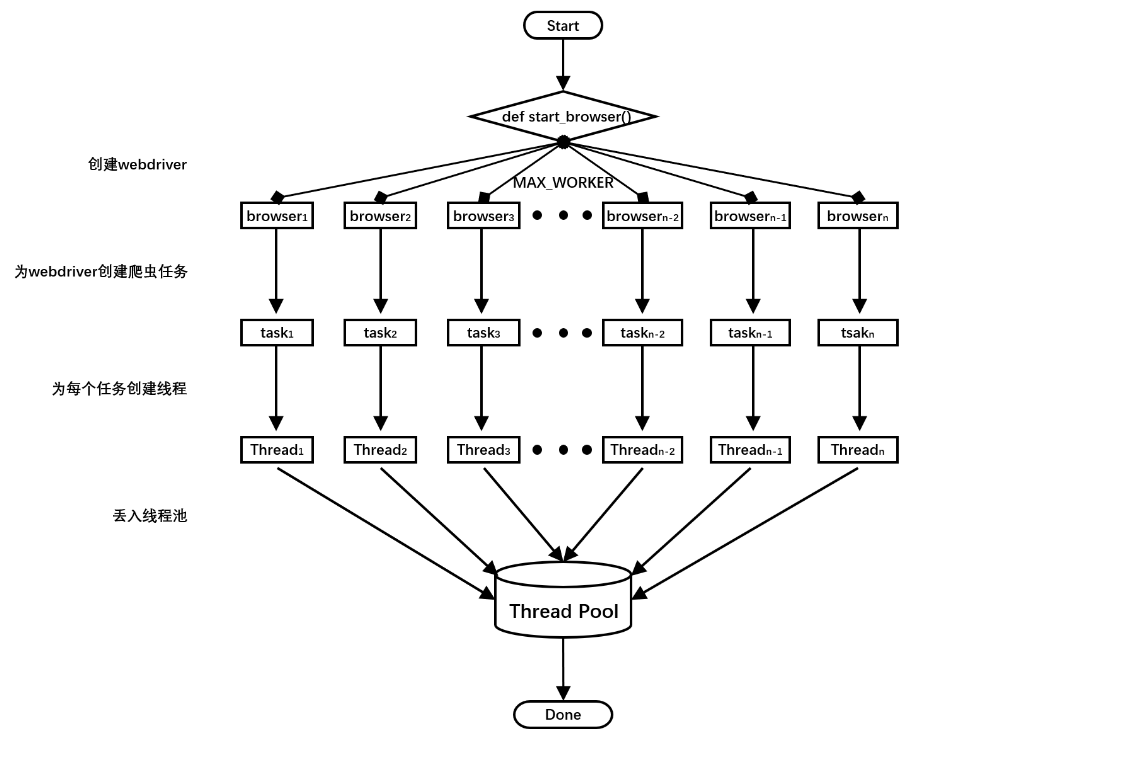
*import* pandas *as* pd  
  
*# 清洗病历类型的数据*df = pd.read\_csv("BaiduHealth\_Depression.csv", encoding="utf-8")  
mask = df['回答'].str.match('^一、初次面诊')  
df = df[~mask]  
  
*# 清洗存在空值的数据*print("移除空值前")  
print(df[df.isna().any(axis=1)])  
df.dropna(axis=0, how='any', inplace=*True*)  
print("移除空值后")  
print(df[df.isna().any(axis=1)])  
df.to\_csv("Depression.csv", encoding="utf-8-sig", index=*False*)

至此，我们完成了对百度健康中“抑郁”相关主题问答的内容的爬取，由于被反爬机制搞怕了，在代码中添加了大量的time.sleep()，所以耗时相对较长，最终爬取有效内容363条，总耗时大概30min。

**爬虫的改进分析与实现：**

在我们的爬虫实现过程中，除了频繁触发反爬机制以外，还有一个更突出的问题：爬取速度相当慢（由于在爬取成功了并未再次出现触发反爬的情况，所以仅针对爬取速度做了优化）。

于是我们改进了爬虫，结合ThreadPoolExecutor库，将原有的单线程爬虫改为了多线程爬虫，以下是流程图：

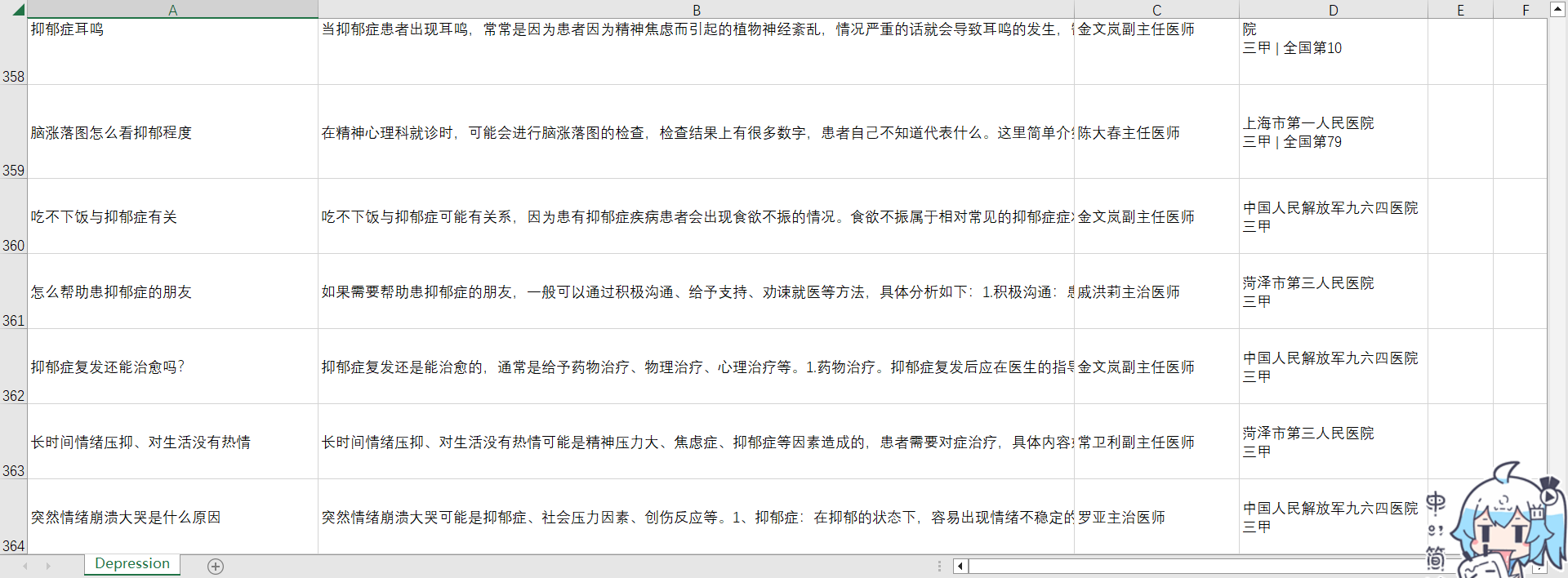


其中在爬虫任务重，采取了递归的调用方法，代码如下：

*if* idx % 20 == 0:  
 print(f"当前爬取了{idx}条数据")  
 *if* idx >= len(URLS) - MAX\_WORKERS:  
 *return* "Done!"  
 *else*:  
 browser\_task(browser, idx + MAX\_WORKERS)  
*except* NoSuchElementException:  
 *if* idx >= len(URLS) - MAX\_WORKERS:  
 fail\_num += 1  
 *return* "Done!"  
 *else*:  
 fail\_num += 1  
 browser\_task(browser, idx + MAX\_WORKERS)  
*except* TimeoutException:  
 *if* idx >= len(URLS) - MAX\_WORKERS:  
 fail\_num += 1  
 *return* "Done!"  
 *else*:  
 fail\_num += 1  
 browser\_task(browser, idx + MAX\_WORKERS)  
*except* IndexError:  
 *return* "Done!"

在我们实验后，在MAX\_WORKER=10即线程数为10，且设置了相同的time.sleep()时间的情况下，原本需要约30min才能够完成的爬虫任务，仅需4min左右即可全部完成，大幅度提高了爬虫效率。但在之后的测试过程中，百度安全验证再一次出现了（已泪奔），所以没有进行更多次数的测试，程序本身可能存在未知的bug，慎重参考。

## 四、 实验结果（展示获取的数据示例）



## 五、 实验总结

在进行爬虫任务的过程中，原以为会比较简单，但是没想到还是遇到了反爬机制等一系列的困难，包括在最后对多线程改进的测试过程中，又再一次遇到了百度安全验证的问题（这几天做噩梦都是百度安全验证），且这一次重置浏览器已经无效，上次成功可能是运气因素。但是我发现在过了将近10h的时间后，百度安全验证会突然消失。既然ip池，cookie池，重置浏览器，修改headers，模拟真实用户行为都无效，那么唯一的可行性可能就是基于深度学习的验证码识别了，但由于精力关系，本次实验也不再测试，可能会在今后的任务重运用相关技术。

其实在爬虫任务里，对爬虫策略的制定这一环节对我们来说是十分有趣的，会有一种类似游戏闯关的快乐，虽然反“反爬措施”这一部分没有进行深入研究，不过我们想也会是一段有趣的经历，在探索的乐趣中就已经学会了selenium的基本运用，这种感觉是十分美妙的。

技术才是最有魅力的！