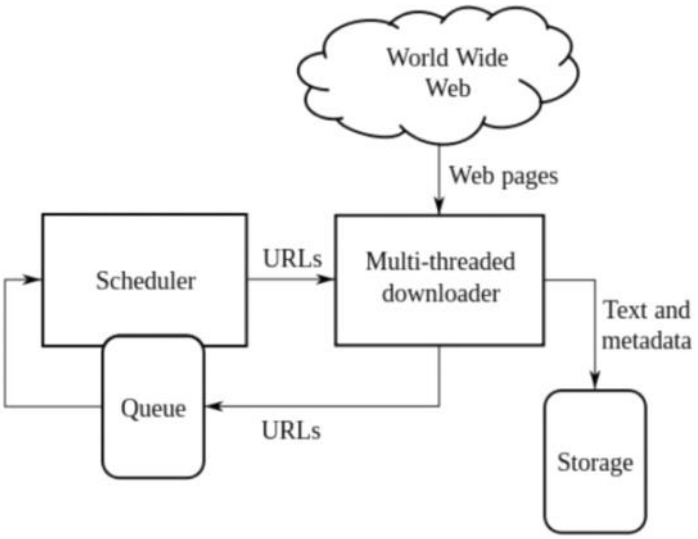
SEU 知识抽取-数据获取

2019年11月12日 10:37

- **一、数据采集原理与技术**

- **构建知识图谱进行的数据采集**
  - 数据源和数据量是知识图谱完备性的重要保证
  - 用户往往拥有私有的数据，但知识图谱构建还需要公共的数据支持，而这些数据往往从万维网上获得
  - 很多半结构化数据、多模态数据分布于万维网，并且具有较好的数据质量，有利益构建大规模高质量领域知识图谱
  - **爬虫数据采集的一般流程**
    - 获得目标数据的URL
    - 向对应的URL提交HTTP请求
    - 解析HTTP响应
    - 存储解析结果



  - **请求与响应**
    - **Requests能做什么?**
      - 向目标URL发起不同种类的HTTP请求（GET 、POST、DELETE等）
      - 定制HTTP请求头部，设置User-Agent和Cookie
      - 使用代理（proxy）进行请求
      - 对HTTP响应进行解析，获取状态码和文本字段

```
# 一个简单的爬虫
import requests
url = "http://www.baidu.com"
r = requests.get(url)
print(r.status_code)
print(r.text)
```

- **数据解析**
  - Beautiful Soup除了本身自带的解析库外，还可以用指定鸡西库对HTML文件进行解析，如lxml html.parser
  - lxml是用C语言编写的HTML和XML解析库。由于lxml的实现非常底层，使得它在处理大多数HTML文件时的速度非常快
  - html.parser是python自带的html解析器，使用方便、稳定
  - 在实践中，我们遇到过使用lxml在不同操作系统上对同一个页面解析结果不一样的情况，推荐使用html.parser
  - 正则表达式
  - 若一个html标签内包含了多个需要抽取的字段，可以编写正则表达式模板来对该标签内容进行抽取

```
<div class="a3">曹操  男  史实人物  字：孟德  生卒（155 - 220）
  籍贯：豫州沛国谯（安徽亳州市亳县）</div>
```

```
def parse_person(detail_line):
    regex1 = '(\w+)\s*([男女])?\s*(\w+人物)?\s*'
    regex2 = '(字：(\w+))?\s*(生卒（(.+?)）)?\s*'
    regex3 = '(籍贯：([\w\[\]]+)((\w+))?)?\s*'
    regex = regex1+regex2+regex3
    pattern = re.compile(regex)
    t = pattern.search(detail_line).groups()
    for i in t:
        print(i)
    return t
```

```
曹操
男
史实人物
字：孟德
孟德
生卒（155 - 220）
155 - 220
籍贯：豫州沛国谯（安徽亳州市亳县）
豫州沛国谯
（安徽亳州市亳县）
安徽亳州市亳县
```

- **多进程并发爬取**
  - 使用多进程技术对页面进行并发放访问能大大提高数据爬取的效率
  - python内置的multiprocessing库提供了进程池类Pool，是实现多进程并发爬取的基础
  - Pool类的构造函数接受一个整形参数，代表进程池的大小，即并发工作的进程最大数，缺省值为计算机系统的CPU数目
  - **Pool类对象常用方法**
    - **apply_async(func, args, callback, error_callback)**
      方法用于将爬取进程添加到进程池中，并发执行。
      - 参数func是要并发执行的函数名
      - args是func函数的参数列表
      - callback error_back分别是进程正常退出和异常退出

时被调用的回调函数用以处理程序执行的结果

- **close( )**
  方法用于进程池的关闭，使其不能加入新的进程池，用于所有进程都提交到进程池里的时候
- **join（）**
  方法表示将主进程挂起，等待所有子进程执行完毕。
  close（）必须在join（）之前调用

- 使用回调函数或try/except语句捕获子进程异常，否则子进程异常退出时不会显示任何消息
-

```
def run(author_id, task_id, proxies, order):
    try:
        write_data_proc(author_id, task_id, proxies, order)
    except Exception as e:
        print(e)


if __name__ == '__main__':
    # 使用时改写get_author_list()函数
    pool = Pool(8)
    prox = proxy_cfg()
    t_id = int(input('please input task id: '))
    a_list = get_author_list(t_id)
    a_list = get_remain(a_list, t_id)
    for n, a_id in enumerate(a_list):
        pool.apply_async(run, (a_id, t_id, prox, n))
    pool.close()
    pool.join()
```

- **反爬虫机制**
  - **多数网站都有反爬虫机制**
    - 通过验证HTTP请求头的反爬虫机制
    - 基于用户行为的反爬虫机制
  - **反爬机制1原理**
    - 每个http请求的头部都会包含一些关于此请求的附加信息。User-Agent、referer、cookie。绝大多数的网站都会查验User-Agent字段
      对不同的User-Agent提供不同的响应
  - **反爬机制1应对**
    - 第二种机制的用对措施也非常简单，就是在发起http请求的时候，在请求头中加入定制的信息。比如在User-Agent字段声明此请求来自一个浏览器Agent或是一个百度爬虫Agent
  - **反爬机制2原理/策略**
    - 拒绝同ip地址短期内对网站发起大量请求，响应状态码为http429
    - 对一段时间内网站后台日志进行分析，对访问次数异常多的

ip加入黑名单

- **反爬机制2应对**
  - 一次请求失败就间隔一段时间后进行再次请求，该措施主要应对短时间内拒绝的情况
  - 使用Web应用测试框架**Selenium**完全模仿人的行为，调用浏览器对页面进行请求，并获取请求结果
  - 最方便有效的方法 是使用**代理服务器（proxy server）**对目标网站进行请求，往往价格越贵的代理性越稳定，安全性也高。

二、数据采集实践

- **百科页面爬取**
  - 对于知识图谱构建来说，百科网站是一个**优质的数据源**（百度百科、维基百科）
  - 在百度百科中搜索一个关键词（如：白起）的时候，会产生一个**http get**请求，其url为“http://baike.baidu.com/search/none？word=白起”，若存在关键词条，此请求将跳重新产生一个get请求跳转到该词条页面。
  - 所以百度百科的爬虫可以直接对第一个url进行请求，通过http重定向机制跳转第二个url。通过对磁条页面的解析，能得到该词条的基本信息以及图片链接，对图片链接进行请求能进一步获取相关图片数据
  - 将词条中的基本信息解析后以JSON格式的文件存储下来，图片数据以二进制形式写入到文件中。
- **微软学术数据爬取**
  - 微软学术网站的数据采用异步请求的方式加载到页面上，免去所有html解析工作
  - 只对正确的API进行请求，然后将请求到的数据保存下来
- **汽车之家论坛数据爬取**
  - 对于目录页面设计目录解析模块，获取一个目录页面下多个帖子的标题、链接、发帖人、发帖时间和跟帖数
  - 通过目录页面的帖子链接就能访问到一个帖子的详细内容页面。
- **微博数据爬取**
  - 网页端
    - PROs
    - CONs：反爬虫机制较强

        - wap端
            - PROs：反爬虫机制较弱
            - CONs

            在对数据全面性要求不高时，或者构建语料库时，强烈建议爬取wap端
        - 处理用户登录
            - 微博需要用户登录才能访问数据，网页的cookies会自动保留登录信息
            - case1：字形用网页端微博获取cookies数据，传入到爬虫程序。（切忌使用常用微博账号）
            - case2：编写自动登录脚本，来自动获取cookies（有大量微博账号时使用）
            - 所以小规模数据爬取是，我们直接手动传cookies更方便
    - Github数据获取
        - case1：直接对网页进行请求，获取页面上展示的数据
        - case2：通过Github提供的REST API和GRAPH API对数据进行请求
            - 熟悉api的使用
            - github官方推荐方法 稳定 但有请求次数限制