PAGE I

摘要

在这个日新月异的互联网时代,每一天都会产生许许多多的信息量。而近年来随着大数据与机器学习等行业的快速发展,这些信息被广泛运用于生产与实践当中。作为为这些行业提供干净可靠数据的采集职业爬虫工程师也变得越来越有活力。如果将那些行业比作大楼那么爬虫工程师便是基石,其重要性不言而喻。

所以本文将探讨如何基于python语言及其支持的scrapy框架设计一款高效灵活的分布式网络爬虫来提高爬取数据的效率,探讨的工作主要分为三个部分,爬虫原理与相关知识,爬虫升级为分布式爬虫的设计与改进,数据的清洗及简单分析。爬虫遵守http协议,因此探讨了爬虫的工作流程。单机爬虫的效率低下,尽管可以开启多线程爬虫但是成本比较高,所以会探讨分布式爬虫的设计与实现进而提高爬虫的工作效率,节约时间,费用与人力资源。爬取的数据并不一定干净可靠需要进行具体的数据清洗以获取整洁的数据源,所以后续会有数据清洗的设计与详细实现,最有会有一点简单的分析。

通过对分布式网络爬虫系统的设计与实现以及数据清洗,可以深刻的领略到爬虫作为数据来源之一重要性,当然更加不可忽视的是分布式爬虫的所带来的益处,不仅仅是金钱方面,更多的是爬虫开发人员自由的解放与时间的节约。

关键词:python;Scrapy;分布式;爬虫;数据清洗

Abstract

In this rapidly changing Internet era, every day will produce a lot of information In recent years, with the rapid development of industries such as big data and machine learning, such information is widely used in production and practice. Professional reptilian engineers, who collect clean and reliable data for these industries, are also becoming more dynamic. If those industries are compared to buildings, then reptilian engineers are the cornerstone, whose importance is self-evident.

So this article will discuss how to support scrapy framework and based on the python language design a highly efficient and flexible distributed web crawler to improve the efficiency of the crawl data, discusses the main work is divided into three parts, the crawler principle and relevant knowledge of the crawler upgrade for the design and improvement of distributed crawler, data cleaning and simple analysis. The crawler complies with the HTTP protocol, so the crawler workflow is explored. The efficiency of single machine crawler is low. Although multi-threaded crawler can be started, the cost is relatively high. Therefore, the design and implementation of distributed crawler will be discussed to improve the working efficiency of crawler and save time, cost and human resources. The crawled data are not necessarily clean and reliable, and specific data cleaning is needed to obtain a clean data source. Therefore, the design and detailed implementation of data cleaning will be followed, and a simple analysis will be the best choice.

Through the design and implementation of the distributed web crawler system and data cleaning, can deeply appreciate the crawler as the data source of importance, and, of course, more can not be ignored is the benefits of distributed crawler, not just money, more is the crawler developers free liberation and time saving.

Keywords:python;Distributed;The crawler;Data cleaning;

基于python的分布式网络爬虫设计与实现

PAGE 3

目录

TOC \o "1-3"\h \z \u 摘要 I

Abstract II

1绪论1

1.1研究的目的与意义1

1.2国内外现状1

1.3设计的思路及主要内容2

1.4论文结构安排3

2相关知识与技术介绍4

2.1 Python简介与Pychram 4

2.2数据提取规则Xpath 4

表1.1 Xpath规则表4

2.3 Scrapy框架与改进5

图2.1 scrapy架构图5

图2.2 scrapy架构的改进6

2.4数据库相关技术7

2.4.1 Redis 7

2.4.2 MongoDB 7

2.4.3数据格式Json 7

2.5 HTML、HTTP协议相关及其他技术8

2.5.1超文本标记语言HTML 8

2.5.2 HTTP介绍8

2.5.3浏览器用户代理及IP代理9

2.5.4抓包工具fiddler 9

2.6数据清洗工具及绘图工具介绍9

2.6.1数据清洗工具Spoon kettle及绘图工具9

3需求设计分析与总体设计10

3.1需求设计与分析10

3.1.1需求设计与分析10

3.1.2网站反爬虫措施分析10

图3.1链家翻页源码11

图3.2常规翻页源码11

图3.3第二页对应地址栏12

3.1.3网站目标数据提取测试12

图3.5 Xpath提取流程13

3.2总体设计13

图3.6三大模块13

3.2.1爬虫程序模块13

3.2.2数据清洗模块14

3.2.3数据简单分析模块14

4设计与实现15

4.1详细功能模块设计15

图4.1爬虫程序模块15

4.1.1爬虫引擎(Engine)模块15

4.1.2爬虫Spider模块15

4.1.3字段存储定义模块16

4.1.4本地及导入其他数据库设计模块17

4.1.5调度器(Scheduler)模块18

4.1.6防反爬虫模块设置18

4.1.7爬虫改进升级为分布式爬虫20

第一个模块改变的是Spider 20

4.2分布式爬虫部署设计与实现21

图4.2分布式部署设计及后续处理21

4.2.1分布式爬虫部署前准备22

4.2.2分布式爬虫部署与运行22

图4.3分别启动爬虫程序22

图4.4 redis数据库中发送指令23

图4.5 windows10端爬虫开始运行23

图4.6 linux\_centos7\_2端爬虫开始运行24

4.2.3查看获得的数据24

图4.7 redis数据库中的数据查看25

图4.8其中一条详细数据展示25

4.3单机爬虫与分布式爬虫对比25

4.3.1单机爬虫与分布式爬虫对比26

图4.9分布式爬虫运行时间分析26

图4.10单机爬虫运行时间分析27

5数据清洗模块设计实现及绘图分析27

5.1数据清洗27

5.1.1为什么需要进行数据清洗27

5.1.2 Kettle数据清洗步骤28

5.1.3 Kettle数据清洗性详细设计28

图5.1字段筛选,去null及价格过滤30

图5.2数据格式处理及价格二次过滤30

图5.3分类30

图5.4行列转行之列转行31

图5.5列转行效果图31

5.2利用excel进行图表绘制32

5.2.1绘图32

5.2.2简单的数据分析32

图5.6底商类各区域平均价格直方图32

图5.7底商类价格随年份变更折线图33

图5.8商业类各区域平均价格34

图5.9商业类价格随年份变更折线图35

图5.10住宅类各区域平均价格36

图5.11住宅类价格随年份变更散点图37

图5.12成都旧五城区住宅类各区域价格随年份变更图38

图5.13其他区域住宅类各区域价格随年份变更图39

6总结与展望40

6.1总结40

6.2展望40

致谢41

参考文献42

研究的目的与意义

在这个互联网高速发展的时代。每一天都会产生许多的信息量,而近年来随着大数据,机器学习等行业的蓬勃发展,可靠而有价值的数据信息供应出现紧缺现象。这表明爬虫工程师将成为互联网数据公司的重要职位。

而随着业务的扩大,需求数据量的增多,如何充分合理的分配人力资源,时间以及成本费用便成为一个很大的难题。对于普通的单机爬虫而言,爬取一个大型的网站信息往往需要花费较长时间,尽管可以使用多线程爬虫来提高爬取的速度,但是这又对单机的硬件配置要求比较高,多以价格自然也就高了。有时为了提高速度,分配多人员进行网站信息采集,每一个人写的爬虫程序可能不同,不便于后期维护,采集到的信息很有可能重复,这也增大了后期数据清洗的工作量。此时,分布式爬虫就应运而生了。它可以减少开发人员的数量。一个人便可以编写完整的代码,同样的程序可以部署到多台廉价的机器上运行,又能做到同步控制减少重复数据,速度又快,还能够节约成本费用,称得上爬虫界的王者了。

考虑到开发语言的难易程度,以及流行程度,我准备基于python在原单机爬虫框架scrapy的基础上,设计一个灵活分的布式爬虫系统,由于scrapy本身不支持分布式,所以需要一定的改进,在scrapy\_redis扩展框架中对于分布式的支持也不一定满足设计,所以还需要在上面进行一定的修改。最后,为了体现爬虫工作者对数据的后续处理过程,还会进行数据清洗以及简单的数据分析,在一个数据相关的公司中,这是一个基础而又重要的部分流水线生产环节。

国内外现状

网络爬虫的研究始于20世纪90年代。从编程的角度来看,目前著名的开源爬虫程序包括Java语言中的Nutch和Herritrix、c++语言中的Larbin和python语言中的scrapy。

从分布式爬虫的发展的角度,目前,许多大型软件公司和企业根据自己的行业特点和需要,开发了许多分布式网络爬虫系统,如谷歌Googlebat,微软Bingbot和Yahhoo发出声音,等等,但是当有许多分布式框架和工具,如Grub, HTTrack,寻求等等。

爬虫类一般分类:通用爬虫类、聚焦爬虫类、增量爬虫类。可伸缩Web爬虫(Scalable Web Crawler)是一种通用的Web爬虫,它从一些种子url扩展到整个Web,主要为门户网站搜索引擎和大型Web服务提供商收集数据。例如百度,谷歌的搜索引擎组件是。焦点爬虫程序,也称为主题爬虫程序,是指有选择地抓取与预定义主题相关的页面的web爬虫程序。增量爬虫:增量爬虫根据一定规模的web页面集合,在现有集合中选择过时的web页面,通过更新数据进行抓取,以保证抓取到的数据与真实网络的数量足够接近。增量抓取的前提是系统已经抓取了足够数量的网页,并且具有这些网页被抓取的时间信息。

设计的思路及主要内容

本文的网络爬虫设计主要是针对一个房地产网站指定相关信息的数据采集,其中涉及网页结构分析,反爬虫措施分析,数据采集技术及改进,数据存储设计,分布式爬虫部署,数据清洗及简单分析。

该分布式爬虫及后续处理需要解决的问题如下:

一.原爬虫框架改进与实现自己的特定需求。原框架不支持分布式,要改进,对于数据的需求涉及多个网页,深度达到3,如何设计数据传递及特定数据提取。

二.分布式爬虫端同步控制。尽量减少多个爬虫端爬取到的重复数据。

三.设计网站防反爬虫措施。网站可能屏蔽我们的爬虫,需要采取如ip代理,伪装不同浏览器发请求等措施来进行处理。

四.后续的数据处理。数据清洗是一个难题,包括字段过滤,记录去重,去空值,分类,行列转换等。

因此主要内容包括:

在理解scrapy框架的基础上,设计存储模块,爬虫采集模块,数据提取与传递模块,防反爬虫模块。

在实现了原scrapy框架的基础上进行改进,实现分布式同步控制,理解其工作原理,配置共享队列,指纹去重,数据存储,设计数据导出到其他数据库的模块。

为了提供可靠有价值的数据源,需要对数据进行一定的清洗,去除"脏乱"的数据。

论文结构安排

本文的结构安排如下:

绪论。首先讲述的是选该题目的目的和意义,然后是国内外现状和设计的思路与主要内容,最后是对本文结构安排的说明。

相关知识与技术介绍。这一章节主要是对设计该分布式爬虫系统前需要掌握的技术基础的介绍与说明。

需求分析总体设计。本章是对需求进行设计分析,以及对总体架构的设计,补充与实现的讲解。

爬虫模块功能详细设计。本章详细的对爬虫的各个模块的实现进行具体的说明,包括如何将原框架改进为分布式支持,如何进行分布式部署等。

数据清洗模块设计实现及绘图分析。为了得到干净有价值的数据,所以爬取到的数据需要进行数据处理,对其进行去重,过滤,字符串处理,分类处理,行列转换等。然后将数据通过Excel进行图表绘制,最终进行一定的分析得出结论。

总结与展望。本章进行了对于此次工作的总结,以及对于未来的展望。PAGE 17  
相关知识与技术介绍

这一章节主要是对设计该分布式爬虫系统前需要掌握的技术基础的介绍与说明。其具体对pyton相关知识,数据库相关,HTTP相关及数据清洗,绘图等进行一定的讲解。

Python简介与Pychram

Python是目前的主流之一,世界上最流行的编程语言,和近年来大数据和人工智能产业的迅速崛起,越来越多的人到Python研究中,利用率上升一旦登上第一的宝座,也在编程语言的数量的列表中使用模糊倾向于追求前三。 Python简单、清晰、易于学习,并且语法优美。它提供了一个功能丰富、功能强大的库,开发效率高,被广泛应用于各个领域。当然,它的执行速度不够快,有一些并发性限制,而且python2与python3不太兼容。但这并不妨碍人们热爱这种语言。

Pycharm是提高python开发效率的集成环境之一,它给开发人员带来了调试、模块安装、智能提示、代码跳转等诸多方便。使用该工具可以使用户更容易地进行编辑。

数据提取规则Xpath

Xpath、XML Xpath语言和XML path语言是一种用于在XML文档中查找信息的语言。 Xpath的选择功能非常强大,提供了非常简洁的路径表达式,其中包含多达100个内置函数,用于字符串、数字、时间匹配和节点、序列处理。

它常用的规则如下:

表1.1 Xpath规则表

表达式描述nodename选取此节点的所有子节点/从当前节点选取直接子节点//从取当前节点选取子孙节点.选取当前节点..选取当前节点的父节点@选取属性

通过合理的编写该规则并将规则运用到实际的网页内容匹配中,可以十分简便的提取到我们所需要的数据信息。

Scrapy框架与改进

Scrapy[1]是一个使用python编写的为了爬去网站数据,提取结构性数据的应用型框架,其采用一部网络框架来处理通讯,能够帮助我们提高下载的速度。它为我们提供了各种中间件接口,我们只需要根据需求进行灵活的调用编写即可。

图2.1 scrapy架构图

下面对Scrapy原架构,如图2.1所示,进行介绍:

Scrapy Engine(引擎):负责Spider、ItemPipeline、Downloader、Scheduler中间的通讯,信号、数据传递等。

Scheduler(调度器):它负责接受引擎发送过来的Request请求,并按照一定的方式进行整理排列,入队,当引擎需要时,交还给引擎。

Downloader(下载器):负责下载Scrapy Engine(引擎)发送的所有Requests请求,并将其获取到的Responses交还给Scrapy Engine(引擎),由引擎交给Spider来处理,

Spider(爬虫):它负责处理所有Responses,从中分析提取数据,获取Item字段需要的数据,并将需要跟进的URL提交给引擎,再次进入Scheduler(调度器),

Item Pipeline(管道):它负责处理Spider中获取到的Item,并进行进行后期处理(详细分析、过滤、存储等)的地方.

Downloader Middlewares(下载中间件):可以当作是一个可以自定义扩展下载功能的组件。

Spider Middlewares(Spider中间件):可以理解为是一个可以自定扩展和操作引擎和Spider中间通信的功能组件(比如进入Spider的Responses;和从Spider出去的Requests)

完整的调度流程:

Spider提供要处理的URL->引擎收到URL->引擎把URL交给调度器,调度器将URL做存入队列处理->然后将存入的URL交给引擎->引擎将从队列中取出来的URL交给下载器->下载器根据URL发请求与互联网交互把相应内容交给引擎->引擎将响应内容交给Spider进行数据提取及新的URL提取,此时会重复前面的所有步骤,并同时把提取到的数据交给引擎,让引擎交给管道文件。

当调度器队列为空的时候,程序终止。

Scrapy框架的改进如图2.2:

图2.2 scrapy架构的改进

其中可以参照对应scrapy原框架结构,不过与之不同的是(1)需要对调度器进行修改,原来从引擎中获得的URL是存储在scrapy框架本身自带的一个双向队列里,此时由于需要改为分布式,所以外加一个redis数据库,让调度器将URL存放到多个spider共享的redis队列中。(2)在redis中使用set集合来进行请求的去重。(3)使用redis相关的管道稳健配置,可以将数据存储到redis数据库。图二中,虚线框是爬虫的主要部分,可以构成一个项目,把项目复制多份数,可以部署到多台机器上,最后就可以形成分布式爬虫[2]。Redis数据库只有一个,及导出到MongoDB的存储转换模块也只有一个。注意此处的数据清洗,只是简单的数据处理模块,真正的数据清洗是在kettle工具中。

数据库相关技术

Redis

Redis是一个基于内存的非关系型数据库(NoSql中的一种),它是一个key-value存储系统。支持丰富的存储类型,如字符串,链表,集合,有序集合,和哈希。选用次数据库的原因是,将请求存储到list队列,而且redis支持set可以去重,爬虫爬取到的数据也可以暂时存储到这个数据库。

有一个专门为Redis设计的远程可视化工具,RedisDesktopManager,使用该工具可以进行远程的Redis数据库访问,并且便于观察数去库中的数据。

其他方面,redis还有更多的应用,也可以搭建集群等。

MongoDB

MongoDB是一个介于关系型数据库与非关系型数据库之间基于文档存储的数据库,其支持的数据结构松散,类似json的bjson格式,可用于存储比较复杂的数据类型。选用它是因为,该数据库适合实时的插入,更新查询,具备网站实时数据存储所需的复制及高度伸缩性。当然可以做持久化存储数据。

数据格式Json

Json是一种轻量级数据交换格式,其常见的两种数据结构为

对象:{key: value, key: value...}是一种被"{}"括起来的键值对结构。

如:{"name":"张三","Age":"20","phone":"9527"}

多对象组成的数组:如 [{"province1":"Chengdu"},{"province2":"Chongqing"}] 以中括号为外层,内层是多个对象组成,每一个对象之间以逗号隔开,每一个对象与(1)中描述的一致。

HTML、HTTP协议相关及其他技术

超文本标记语言HTML

在web站点中,所有web页面都是由超文本标记语言HTML创建的。在这种语言中,可以使用各种标记来处理文件。这些标记决定文件内容的外观、结构、交互性和其他方面。它们的共同点是"<>"的使用,如段落标签,图片标签等。

在浏览器中,是看不到任何HTML标记的,但是在浏览器中所看到的网页效果却都是由这些标记生成的。在浏览器中一般点击键盘F12按键所展示的网页源码,就是一个有HTML超文本语言组成的内容。

HTTP介绍

HTTP[3]是一个超文本传输协议,是所有web应用程序使用的通信协议。HTTP使用一种基于消息的模型:客户端送出一条请求消息,而后由服务器返回一条响应消息。

表2.2常用状态码及对应作用

状态码描述200-OK成功的HTTP请求返回的状态码301-moved permanently永久跳转,将请求的网页将永久跳转到被设定的新位置403-forbidden禁止访问,合法,但服务器端设置了规则404-not found服务器找不到客户端请求的指定页面500-internal server error内部服务器错误,服务器遇到了想不到的情况,不能完成客户端解析502-bad Gateway坏的网关,一般是代理服务器请求后端服务器,后端服务器不可用或没有完成相应网关服务器503-service unavailable服务当前不可用,可能是服务器超载或停机维护。或反向代理后面没有可提供的节点504-gateway Timeout网关超时,一般是网关代理服务器请求后端服务时,后端服务没有在特定的时间内完成处理请求  
浏览器用户代理及IP代理

User Agent用户代理(中文名称为用户代理)是Http协议的一部分,也是头域的一部分。它是一个特殊的字符串头,是一个访问站点时提供给您的浏览器类型和版本、操作系统和版本、浏览器内核等信息的标识。有了这个标志,用户访问的网站可以显示不同的布局,为用户提供更好的体验或统计。

IP代理:是一种特殊的网络服务,允许一个网络终端(一般为客户端)通过这个服务与另一个网络终端(一般为服务器)进行非直接的连接。简单通俗讲就是类似房东出租房子的过程。三者关系,房东(客户端),中介所(代理服务端),租房客(终端服务器)三者之间的关系,其中透明代理是指:三者都知道彼此的存在。匿名代理,是房东,中介知道三者都存在,而租房客只知道中介的存在,并且租房客可能会将中介当成房东。代理的时候一般都是代理Ip地址,即客户端不适用自己本身的IP直接与终端服务器交互,而是将自己需要的信息,发给代理服务,让代理服务以它的Ip去于终端交互并取得结果,客户端间接从代理服务获取最终的信息。

抓包工具fiddler

Fiddle是一个快速抓包工具,通过简单的配置就可以抓取到浏览器或其他与外界互联网传递的信息包。在此工具中可以分析信息包的详细内容,比如一个HTTP请求与响应的内容,请求头,请求体,响应头,响应体等。

数据清洗工具及绘图工具介绍

数据清洗工具Spoon kettle及绘图工具

Spoon Kettle是一种ETL工具。可以完成目标数据源的数据提取。经过一系列的数据转换,最终可以形成所需的数据模型并将其加载到数据仓库中。 Kettle的两个核心组件是transformation和job,它们可以用来提取满足各种生产需求的数据,甚至可以用来为操作设置Kettle集群。这一次,我们使用核心中的转换来进行数据处理,它包含了在数据清理阶段中要解释的各种组件。

通过Kettle数据清洗,分类后可以将数据转换成不同的数据文件格式,其中包括支持Excel的xls格式。Excel支持强大的图表功能,使用它可以帮助我们绘制不同样式的图表,方便我们对数据进行观察与分析。

需求设计分析与总体设计

本章是对需求进行设计分析,以及对总体架构的设计,补充与实现的讲解。

需求设计与分析

需求设计与分析

互联网网信息包罗万象,我们设计的分布式爬虫不可能将整个互联网的数据信息全部爬取下来,因此需要有针对性的爬取(这属于聚焦爬虫)。所以我们选择的是一个著名的房地产网站,链家进行需求分析[4]。为了进一步减小范围,我们将其范围锁定为新房楼盘,的范围锁定为成都,按照区域分为锦江,青羊,武侯,高新,成华,金牛等。

每一个页面都会包含很多信息,获取其中有价值的具体数据才是最有意义的。所以此时对页面进行分析,规划出需要爬取的信息内容。在本课题中一共将会设计出15个字段,其中14个字段表示的内容来自网页,13个字段表示的是网页中有价值的内容。这13个字段具体(1)楼盘名称(2)楼盘类型(3)楼盘价格(4)所在区域(5)楼盘详细地址(6)最新开盘时间(7)交房时间(8)开发商名称(9)楼盘出售状态(10)绿化环境评分(11)交通方便评分(12)周边配套评分(13)综合评分。另外两个字段是单独设置的,与网页相关的是(14)评分链接地址,无关说明字段(15)爬虫名称。

通过定义14个变量字段用于存储这些具体的内容。获取到的每一条数据以json的格式存储到本地及数据库。

网站反爬虫措施分析

每一个网络上的网站或多或少都存在一些反爬虫的措施。为了更方便我们的爬虫程序运行,我们需要针对该链家网站设计出防反发爬虫的措施。

推荐访问这个房地产网站使用chrome浏览器,因为这个浏览器支持很多可扩展性插件,而且便于开发者进行调试与观察。首先进行插件ProxySwitchOmega的安装与配置,配置一个本地代理,端口为fiddler中配置的端口,其中fiddler之前需要进行一些配置包括端口,证书等。确保能抓取到数据包。

该网站可以进行登录也可以进行不登录,经过后期测试都能访问到数据。下面是反爬虫措施分析及解决方案。

1)在浏览器中开始发送请求测试(访问该链家网新楼盘主页),分析抓取到的数据包能否获取正常相应内容,寻找发送请求的数据包中有无包含特殊请求字段,请求参数等。最终发现并无特殊请求字段与特殊参数。

2)由于爬虫是自动解析新请求链接,如点击下一页,便是发送一个新请求,因此在浏览器中查看其源码发现如下图所示:

图3.1链家翻页源码

此图源码对应[上一页1234...67下一页]这种部分页面,href链接指向的不是直接可见的地址链接,它使用javascript进行动态的链接生成。而常规的一般如下图:

图3.2常规翻页源码

这个会显示页码及对应连接如第10页及对应href详解地址,下一页及详细href,所以常规提取链家网的链接不会成功。这是一个网站中防止发爬虫的措施之一。

寻找突破口通过点击页码观察地址栏的变化如图3.3,图3.4。

点击第二页:

图3.3第二页对应地址栏

点击第三页:

图3.4第三页对应地址栏

发现地址栏的pg2与pg3有明显的变化,通过手动将改为pg4,pg6等,发现下面的页码也会改变,所以能通过重新构造固定加局部变化的链接,不用提取链接来达到遍历不同网页的目的。

在windows的CMD界面或直接linux的终端界面,输入:scrapy shell 链家主页地址查看返回的内容是否包含成功状态吗200,通过提示寻找对应措施,如状态码302重定向,则可能是没有设置浏览器用户代理user-agent,此时输入:crapy shell -s USER\_AGENT ="Mozilla/5.0..."链家网页地址,在本论文完成之前都一直存在302的问题,但之后经过测试该网站好像已经撤销了这个防止反爬虫的措施。

4)网站维护工作者够在服务器端进行访问者的监控,包括Ip地址,主机名,使用浏览器等。通过检测访问者访问网站的频率,判断出访问者是常规浏览器访问还是爬虫程序访问,进而达到屏蔽访den问者的目的,常见的是封Ip等。经过多次尝试该网站这方面比较友善,但是稳妥的方法还是使用匿名Ip代理来进行处理。

5)当网站的发爬虫措施做得过于完善,无法使用一般的防反爬虫的措施时,可以使用模拟真实浏览器(Selenium+Phantomjs)来进行访问。如:当第二步的地址栏无变化时就可以使用此方案。这个方案是一个万能的防反爬虫措施,缺点是写的代码较多一点,爬虫执行的效率会有所下降。

网站目标数据提取测试

在chrome浏览器中安装Xpath插件,使用crtl+shift+x弹出编写界面,点击f12查看网页源代码,在网页中找到要提取的字段,点击右键查看源码,使用XPath插件工具编写提取字段相对应的Xpath规则。如一个楼盘的名称提取,其对应规则是这样的://div[@class="resblock-name"]/a[@href="/loupan/p\_jcypdsaaupy/"]/text()。提取其他字段信息也是类似的规则。之后,使用notepad++文本编辑器将将所有满足条件的字段的Xpath规则保存起来,并写好备注名以后续代码编写时使用。

其详细流程设计,如6所示:

图3.5 Xpath提取流程

总体设计

总的来说,本课题一共设计了三个大的模块,如图3.6所示:

2

图3.6三大模块

其中每一个模块的具体实现会在后面章节进行详细说明。下面是每一个模块的一些总述说明。

爬虫程序模块

爬虫程序模块是总体设计的三大模块中最重要的模块,其包括爬虫程序具体的设计,下载器,反爬虫,网页数据提取,新URL提取与组装,数据字段定义,数据存储,数据简单清洗,数据导出,分布式改进等。

数据清洗模块

此模块是针对数据进行处理,爬虫程序获得的数据不一定满足完整干净,而且对于不同的数据使用者而言可能有不同需求,所以此模块针对数据进行去重,过滤,字段选择,行列转行,数据类型变更,去空值,分类等操作。

数据简单分析模块

在数据进行清洗过后,得到的数据是完整干净有价值的,这些数据可以用于存到数据库为新建的网站提供可靠数据,也可以在数据交易相关的商业平台进行出售,还可以为数据分析师提供数据挖掘,分析的数据源等等。所以为了模拟数据从获取到清洗,再到使用这一完整的流程,会有一个简单的数据分析。

设计与实现

本章详细的对爬虫的各个模块的实现进行具体的说明,包括如何将原框架改进为分布式支持,如何进行分布式部署等。

详细功能模块设计

图4.1爬虫程序模块

该图与图2.2一致,图4.1中虚线框内的部分是爬虫程序的主要模块。以下将为各个进行详细的讲解及实现爬取网站的需求。

爬虫引擎(Engine)模块

这个是scrapy框架中自带的组件,是整个系统的核心,他负责各个模块的信息传递。不需要使用者进行单独的定义与开发。

爬虫Spider模块

这个是解析器模块,负责提取相应内容的数据以及提取新的请求链接返回给引擎。在提取的时候会使用Xpath规则进行提取(当然在第三章的时候我们使用过插件来帮助我们编写规则,此时只需要将保存好的规则拿过来直接使用就可以了)。

部分伪装代码讲解:

class LjxfSpider(scrapy.Spider):

......

start\_urls =[url + str(start)+ end]

def parse(self, response):

......

item['name']=each.xpath("div[@class='resblock-name']/a/text()").extract\_fir st("null")

......

yield scrapy.Request(url=item['grade\_url'],meta=

{'meta\_1':item},callback=self.second\_parse,dont\_filter=True)

......

yield scrapy.Request(url=url, callback=self.parse, dont\_filter=True)

def second\_parse(self, response):

......

yield item

在这个模块中定义了一个LjxfSpider类,它继承父类scrapy.spider,省略号是神略略部分代码。

这个类包含两个函数,分别是def parse(self,reponse)与def second\_parse(self,response)。类中定义了一个拼接的起始地址。在程序第一次执行的时候会默认去这个地址进行请求发送,并将返回的内容交给第一个函数的response存储。在执行第一个函数的时候会从response中提取相关数据及grade\_url链接,之后发送grade\_url链接并将返回结果与第一次提取到的数据一起传递给第二个函数。第二个函数进行grade\_url响应内容数据提取并将第一个函数传递过来的数据内容进行拼接在一起,最后调用yield item将数据交给引擎。

字段存储定义模块

这个模块比较简单需要对数据进行定义,及以什么样的变量来接收提取到的数据。部分代码如下:

class LianjiaItem(scrapy.Item):

#14个字段

#楼盘名称

name = scrapy.Field()

#楼盘类型

type = scrapy.Field()

#楼盘状态

status = scrapy.Field()

......

这个模块定义了一个LianjiaItem类,继承scrapy.Item父类,然后在进行相关的字段定义。这个Item类在Spider模块中会使用到,用于接收Xpath规则提取到的数据。

本地及导入其他数据库设计模块

这个模块属于其他的模块与爬虫程序并无多少联系,主要是将存储到Redis数据库的数据导入到MongoDB,比较简单。

#!/usr/bin/env python

#-\*- coding:utf-8-\*-

import redis

import pymongo

import json

def process\_item():

#创建redis数据库连接

rediscli = redis.Redis(host="127.0.0.1", port=6379, db="0")

#创建MongoDB数据库连接

mongocli = pymongo.MongoClient(host="127.0.0.1", port=27017)

#创建MongoDB数据库名称

dbname = mongocli["lianjia\_ds"]

#创建mongoDB数据库的名称

sheetname = dbname["lj\_info"]

offset =0

while True:

# redis 数据库表名和数据

source, data = rediscli.blpop("lj\_distribute:items")

offset +=1

#将json对象转换成Python对象

data = json.loads(data)

#将数据插入到sheetname里

sheetname.insert(data)

print offset

if \_\_name\_\_=="\_\_main\_\_":

process\_item()

定义一个python脚本文件,导入相关依赖模块,详细说明如上注释。

本地存储设计:

class LianjiaPipeline(object):

def \_\_init\_\_(self):

self.filename = open("lj.json","w")

def process\_item(self, item, spider):

# ensure\_ascii=False禁用Ascii码,采用unicode码

content = json.dumps(dict(item), ensure\_ascii=False)+"\n"

self.filename.write(content.encode("utf-8"))

return item

def close\_spider(self, spider):

self.filename.close()

调度器(Scheduler)模块

这个模块主要负责从引擎获取URL链接请求,然后又将请求存入队列,最后又将请求传给引擎。那么疑问来了?这样做不是多此一举吗,引擎将url交给调度器,最后又获得请求?其实这样做是有原因的,当程序运行过快式,Spider快速提取到成百上千等新的请求并交给引擎,而下载器从引擎拿新请求去发送的时候又表较慢,那么此时引擎中就会有大量新请求堆积,从而降低爬虫程序效率,甚至崩溃,所以需要有专门的一个调度器模块来调度这些请求。这个模块在配置文件中进行配置启用等即可。

防反爬虫模块设置

通过第三章反爬虫措施分析我们得到了需要进行防反爬虫的设计需求。按照解决方案。我们可以编写浏览器用户代理,让爬虫程序发送请求是每一个请求都可能携带不同的浏览器标识。通过编写与设置Ip代理,让网站服务端不能辨别出我们请求网页的真实Ip地址。为了延缓网站的监控,我们还可以在配置文件中设置每一个请求响应的间隔时间。为了达到网页能够自动跳转,我们要在Spider中重新构造请求,详细如下:

if self.start <=72:

self.start +=1

url = self.url + str(self.start)+ self.end   
print url

yield scrapy.Request(url=url, callback=self.parse, dont\_filter=True)

通过此自增方式,实现请求的局部更新。让每一次请求都可以获取到新的网页。

通过在下载中间件中编写Ip代理与浏览器用户代理突破反爬虫措施。如下:

class RandomUserAgent(object):

def process\_request(self, request, spider):

useragent = random.choice(USERAGENT\_AGENTS)

print useragent

#request.headers.setdefault("User-Agent", useragent)

request.headers['User-Agent']= useragent   
class RandomProxy(object):

def process\_request(self, request, spider):

proxy = random.choice(PROXIES)

#print proxy

if proxy['user\_passwd'] is None:

#如果没有代理账户验证使用方式

request.meta['proxy']="http://"+ proxy['ip\_port']

else:

#账户验证,账户和密码需要转码

base64\_userpasswd=base64.b64encode(proxy['user\_passwd'])

#print base64\_userpasswd

#对账户密码进行base64编码转换

request.meta['proxy']="http://"+ proxy['ip\_port']

#对应到代理服务器的信令格式里

request.headers['Proxy-Authorization']='Basic '+ base64\_userpasswd   
在agents文件中添加变量

USERAGENT\_AGENTS =[

'Mozilla/4.0(compatible; MSIE 8.0; Windows NT 6.0)',

'Mozilla/5.0(compatible; MSIE 8.0; Linux)',

......

]

在proxies中添加变量,

PROXIES =[

#{"ip\_port":"180.118.242.134:9999","user\_passwd":""},

#{"ip\_port":"119.1.97.193:36751"},#,"user\_passw":""}#表示无账户验证

{"ip\_port":"119.1.97.193:36751"},#,"user\_passwd":"account:password"}

#{"ip\_port":"119.1.97.193:36751"},#,"user\_passwd":"account:password"}

]

这样便可以满足基本的防发爬虫措施。

爬虫改进升级为分布式爬虫

之前的都是基本爬虫的讲解,现在对原来的爬虫程序进行改进。

第一个模块改变的是Spider

class LjxfSpider(RedisSpider):

#start\_urls =[url + str(start)+ end]

redis\_key ="ljspider:start\_urls"

def \_\_init\_\_(self,\*args,\*\*kwargs):

def parse(self, response):

for i in range(1,101):

start = i

url = self.url + str(start)+'/'

print url

yield scrapy.Request(url=url, callback=self.parse\_first)

def parse\_first(self, response):

......

def second\_parse(self, response):

......

将继承类改为RedisSpider,注释掉起始地址,使用redis\_key进行替换。这样在多个程序中运行时,可以直接在redis中发送指定进行多个程序的管理。添加初始化函数。将原来parse中除自定义局部请求更新外的内容全部移动自新函数parse\_first。在配置文件中启用redis数据库并进行相关的配置。具体配置redis数据库的远程Ip,端口。配置启用redis中的指纹请求去重,配置调度器使用redis的队列,不使用自带的队列。开启redis的管道相关支持。

分布式爬虫部署设计与实现

分布式架构[5]规划,如图4.2所示:

图4.2分布式部署设计及后续处理

一共需要3台机器 Linux\_centos7\_1192.168.6.131  
Linux\_centos7\_2192.168.6.5

Windows10192.168.43.80

其中3台机器的ip如右侧,名称如左侧。Linux\_centos7\_1上安装了两个数据库,它为Master端,不部署爬虫程序,但是会有一个数据可导出脚本文件。爬虫程序部署在Linux\_centos7\_2与windows10 Slave端。其中windows10中有各种远程可视化工具以及后期数据清洗,绘图等工具。

分布式爬虫部署前准备

在win10上,将完整改写好的程序项目进行打包拷贝,使用filezilla将项目包传送到linux\_centos7\_2端,之后进行解压,进入到程序入口所在目录。将导出脚本文本文件上传到Linux\_centos7\_1端。在windows中进入CMD命令模式,进入到程序入口所在目录。启动linux\_centos7\_1中的数据库。

分布式爬虫部署与运行

在爬虫程序端输入爬虫项目启动命令scrapy runspider ljxf\_distribute.py,等待统一的指令。

如图4.3所示:

图4.3分别启动爬虫程序

Redis数据库端发送爬虫运行指令,如图4.4:

图4.4 redis数据库中发送指令

爬虫运行的情况如图4.5,4.6:

图4.5 windows10端爬虫开始运行

图4.6 linux\_centos7\_2端爬虫开始运行

查看获得的数据

在windows端使用远程工具进行数据库查看如图4.7:

图4.7 redis数据库中的数据查看

可以发现成功的获取到了数据。数据以json格式存储在数据库中。取出其中的一条详细的数据进行查看如图4.8:

图4.8其中一条详细数据展示

单机爬虫与分布式爬虫对比

设计分布式爬虫的目的是为了节省时间,人力,及成本费用。接下来会做一个对比。

单机爬虫与分布式爬虫对比

已经完成了分布式爬虫的运行,我们来看一下其运行的时间,如图4.9:

图4.9分布式爬虫运行时间分析

使用两台机器进行网页数据爬取,windows10爬取608条,linux\_centos\_7\_2端爬取498条一共1010条数据,花费时间39分钟左右。

为了比较单机爬虫与分布式爬虫的速度,我们再进行一次单机爬虫的运行,这次只运行windows10端的项目程序,而linux\_centos7\_2端的程序不运行。

结果如图4.10所示:

图4.10单机爬虫运行时间分析

从图中可以看出抓取同样的数据花费时间是约80分钟左右。使用两台机器部署爬虫项目程序项目比一台机器要节省一半多的时间,所以不难推断出,进行大数据量的信息抓取时,分布式规模越大及部署爬虫项目程序的机器越多,速度越快。

数据清洗模块设计实现及绘图分析

为了得到干净有价值的数据,所以爬取到的数据需要进行数据处理,对其进行去重,过滤,字符串处理,分类处理,行列转换等。然后将数据通过Excel进行图表绘制,最终进行一定的分析得出结论。

数据清洗

为什么需要进行数据清洗

经过爬虫程序项目不断调试与最终成功运行后,我们已经得到了网页上指定数据的信息,其中多达1000多条信息,每一条都包含详细的楼盘的各种信息。但是这些数据中可能有重复的,而且像评分,或者交房日期,价格等都可能是"暂无"或者为空的情况,如果进行评分平均值计算,则会有影响。再比如,价格有可能单位我们规定的元/平方米,而它的值可能实际为几百单位是百万元/幢。在类型方面,又分为别墅,住宅,商业,底商,所以为了便于区分还需要进行分类处理等。

Kettle数据清洗步骤

数据清洗大致分为:

预处理阶段:

这个阶段一般是将数据导入数据处理工具,然后进行数据查看,找出需要进行处理的内容。

阶段1:去除/补全有缺失的数据

对于清空空位值,可以忽略元组,可以手动填充空位值,还可以使用全局变量填充空位值。空位值可以用属性的平均值、中值、最大值、最小值或更复杂的概率统计函数值填充,也可以删除字段。

阶段2:去除/修改格式和内容错误的数据。

1.时间、日期、值、全半角等显示格式不一致这种类型的问题通常与输入端相关,当将多个源的数据合并为一致的格式时,可能会遇到这种问题。内容中不应该有字符有些内容可能只包含几个字符,例如id号是一个数字+字母,而中文名称是汉字(zhao C仍然很少见)。最典型的是头、尾、中间的空格,也可能出现名称中的数字符号、汉字中的id号等问题。在这种情况下,需要半自动检查和半手工方法来识别可能的问题并删除不需要的字符。3.内容与字段的内容不一致姓名写性别,身份证号码写手机号码,都属于这类问题。然而,这个问题的特殊性在于,它不能简单地删除处理,因为原因可能是人类的错误,前端可能不是验证,或部分或全部现有的列不一致时,数据导入,问题类型应该详细确认。

阶段3:去除/修改逻辑错误的数据

这一步骤主要是进行去重,去除不合理值,修正矛盾内容。

阶段4:去除/不需要的数据

这个简单,把不需要的字段和数据删除。

阶段5:关联性验证

判断出字段之间的关系是否合理且符合实际,如新楼盘的交房日期与开发日期的关系。开发动工日期不可能大于交房日期等。

Kettle数据清洗性详细设计

在导入数据之前我们先对数据的存储格式进行转换,通过在线网站json转xls将其转换为一张表。

预处理阶段:

打开kettle,使用表输入或数据库输入。观察数据,分析出需要处理的内容。我们发现xls格式的表中存在许多问题。

评分相关记录的是如"1.0分",它不是一个小数,是一个字符串

日期格式是字符串,不是date日期格式,支持的excel打开不能识别。

日期和评分相关等存在"暂无"字样与null字样

数据高度混杂,类型:住宅,底商,商业,别墅及各个区域也是混杂在一同一张表中不便于分析处理。

价格单位是元/平方米,但却存在百万元/幢这种不符合要求的数据。

以区域为重点,但是却存在详细区域位于同一列的情况需要进行行列转换。

后期发现问题,小数存储0点几时不会显示0,只会出现点几的情况

如要求数据比如"0.2",但是显示却只是".2"。

为了方便处理我们将各个问题分散处理,所以不一定按照上面数据清洗的规定步骤进行一比一步处理,而是采用灵活的方式。

详细的数据清洗设计,其中带"分"字样及小数问题,在使用excel中使用ctrl+f进行数据全替换即可解决。其余的则需要真正的步骤清洗。将每一次清洗的数据输出作为下一次数据清洗的数据输入。需要注意的是Spoon kettle每一个组件的含义,不同组件对应的功能不同。主要设计为四大模块,详细如,图5.1,图5.2,图5.3,图5.4介绍。

图5.1字段筛选,去null及价格过滤

将爬取到的数据输入kettle工具,通过添加kettle中的各个组件,并设置执行流程与每一个组件的的具体参数进行字段选择,去null,去待定字样的操作。得到不含待定的表,作为又一次数据清洗的输入。

图5.2数据格式处理及价格二次过滤

这个流程主要是对不符合数据格式的数据进行处理,并将单位为百万元/幢的数据筛选到另外一张表中。

图5.3分类

图5.3主要是按对指定字段楼盘类型进行分类处理,输入数据是图5.4数据清洗后的输出数据。

图5.4行列转行之列转行

将图5.4操作后的3张数据表中的数据,分三次输入这一个步骤,得到三张列转行后的数据表。这样做的具体具体效果将会如图5.5。

图5.5列转行效果图

最后得到的数据如图5.5中下方的样式。之后便可以进行数据绘图分析了。

或许有一个疑问,开始爬虫爬取到的数据包含很多字段,那为什么这里只需要很少的一部分哪?

其实这些数据是有用的,如果将这些数据用于一个新的网站建设或数据交易这些数据便很详细很有价值,这正是他们需要的。对于数据分析师而言,挖掘分析这些数据相关的需要考虑多种因数,也需要详细的数据。而我们只是针对区域,交房时间,价格这三个字段进行绘图分析。所以会清洗出只符合我们要求的数据。

利用excel进行图表绘制

使用excel打开我们最终获得的数据表,进行绘图。

绘图

我们将会绘制折线图与直方图。点击插入图表选择相关图,然后设置横纵坐标,调整相关字段。具体的图表展示及分析析在5.2.2中。

简单的数据分析

底商类各区域平均价格直方图

图5.6底商类各区域平均价格直方图

从底商类平均价格图可以看出来,锦江区的底商类新楼盘最贵多达55000元/平方米,高新西区与龙泉驿区相对较便宜,但是也高达23000到26000元/平方米左右。

底商类价格随年份变更折线图

图5.7底商类价格随年份变更折线图

从2012年到2021年中,目前数据中展示的最高价格高达80000元/平方米,最低也接近10000元/每平方米。从2015年到2021年价格浮动变化较快,估计可能是新楼盘增多也可能是政策相关。

商业类各区域平均价格

图5.8商业类各区域平均价格

从商业类平均价格图可以看出来,锦江区的商业类新楼盘也是最贵多达55000元/平方米,但双流与郫都区相对较便宜,是20000元/平方米左右。

商业类价格随年份变更折线图

图5.9商业类价格随年份变更折线图

商业类变化比较集中的是2017-2020年,不过总的价格都在20000-40000元/平方米之间,预计未来几年大多数价格也在这个区间内。

住宅类各区域平均价格

图5.10住宅类各区域平均价格

从住宅类平均价格图可以看出来,锦江区的住宅类新楼盘也是最贵多达26000元/平方米,但高新西区,都江堰和新都区相对较便宜,是7000-9000元/平方米左右。对比底商,商业,住宅三类,住宅类型的新楼盘要便宜很多。

住宅类价格随年份变更散点图

图5.11住宅类价格随年份变更散点图

住宅类的新房在2014-2022年左右变得特别多,价格也相对稳定,大多数都不会超过30000元/平方米。

成都旧五城区住宅类各区域价格随年份变更图

图5.12成都旧五城区住宅类各区域价格随年份变更图

从年份来看,成都市五城区,锦江,成华,青羊,武侯,金牛中锦江区的住宅类房价一直都相对于另外四城区高,但是最高也就3万元/平方米左右,可以推断出锦江区在成都市五城区中经济方面最为发达,地段也较繁荣。

其他区域住宅类各区域价格随年份变更图

图5.13其他区域住宅类各区域价格随年份变更图

从图中可以看出2019-2022年的折线密集程度可以看出住宅类楼盘明显增多,绝大多数的价格在10000-18000元/平方米。其中所有区域2018-2021年的各自的价格都相对于2018年以前要高。

本章进行了对于此次工作的总结,以及对于未来的展望。

总结

在这个大数据与人工智能的时代,数据的需求量将会变得越来越大,数据的搜集也将变得越来越重要。作为一个爬虫工作者而言,设计分布式网络爬虫来为其提高数据采集的速度便变得尤为重要,它既可以减少开发人员的数量,节约成本费用,还可以让使用者节省大量时间。

在本课题中,我们介绍了数据如何获取,清洗,以及简单的后续应用。首先进行了爬虫设计的前的网站分析及需求设计工作,其中重点分析了网站的反爬虫措施,然后进行了爬虫原型的设计与实现工作,包括各模块具体编写,接着为了提高爬取的速度,我们将原型爬虫改写成了分布式爬虫,其次我们设计了部署策略,将爬虫部署到了多台机器上并成功运行。在数据获得以后,我们开始了数据清洗工作,先进行了数据预处理,详细分析数据中存在的问题,并设计出了四个阶段来进行处理。最后我们获得了需要分析的数据,通过绘图,观察分析得出了一定的结论。

展望

爬虫的领域很广,也很有趣,在本次课题中,在编程语言方面我们所做的只是实现基于python的 scrapy改进的分布式爬虫并设计一些模块,其实其他语言支持的分布式爬虫也各有特点,分布式爬虫的框架也多种多样,像最近的基于hadoop的分布式爬虫,基于docekr的分布式爬虫也都很有意思。我们具体以一个房地产网站来进行爬虫设计与实现,但是互联网上的网站各有特色,有价值的数据也多不胜数,其反爬虫措施,数据分布等都有不同,如何克服困难设计出合理的高效的爬虫都是一个值得研究的课题。

致谢

时间飞逝,转眼大学四年的学涯也即将结束。在这四年里学到了很多知识,从一个懵懵懂懂的大一新生逐渐成长为一个走向社会的毕业生。在这里将感谢曾经帮助过我,并让我进步的人。

首先,感谢我的毕业设计指导教师,何粒波老师从毕业还未正式开始时,就已经给了我详细的指导和说明,在整个毕业设计期间无微不至的关照我,无论是写作还是爬虫程序设计都为我提供了良好的建议,给与我很大的帮助,让我充分领略到爬虫领域的风采和乐趣。

其次,感谢曾经指导过我的杨明根老师,在我迷惘的时期为我提供建议,在初次想接触大数据领域时,他指导我如何入门以及掌握哪些重要的框架,他是真正的良师益友。

此外感谢我的家人,在读大学的时候,是他们在背后默默支持我,鼓励我,让我能够不断进步。

最后,感谢母校为我提供这么好的成长环境,让我收获室友间的真挚友谊,以及塑造人生的机会。

此处填写本人毕业设计(论文)题目

PAGE 45

参考文献、脚注、尾注

崔庆才.Scrapy框架的使用.北京：人民邮电出版社，2018，(13)：468-496.

李文龙.基于docker集群的分布式爬虫设计，2016，(3)：20-26.

王景阳.网络爬虫相关知识.主题网络爬虫的并行研究，2017，(2)：5-9.

叶婷.网络爬虫系统需求分析.基于关键词的微博爬虫系统的设计与实现，2016，(3)：19-23.

Naresh Kumar ，Manjeet Singh .RELATED WORK.Framework for Distributed Semantic Web Crawler ， 2015, (2)：1403-1404.

Andri Mirzal.PyThinSearch: A Simple Web Search Engine ,2009,(3)： 2-4.

李舟军.高性能爬虫：研究综述[J].计算机科学，2009,36(8):26-29.

周立柱，林玲.聚焦爬虫技术研究综述片[J].计算机应用，2005,25(9)：1965-1969.

高强.基于redis的分布式爬虫框架设计[J].农业网络信息,2017(8).

程锦佳.基于hadoop的分布式爬虫及实现[D].北京邮电大学，2010.