网络爬虫的设计与实现

摘　要

（宋或楷小4号）

关键词：（宋或楷小4号）

#### The design and implementation of a web crawler

#### Abstract

（TIMES 12）

**Key words:** initial velocity method; gas emission rate; mine panel(district) （TIMES 12）

目　　录

摘要 1

Abstract 1

1. [绪论](#_第一章_绪论)

第一节 [课题背景](#_第一节_课题背景)

第二节 [课题意义](#_第二节_课题意义)

第三节 [网络爬虫的历史](#_第三节_网络爬虫的历史)

第四节 [网络爬虫的分类](#_第四节_网络爬虫的分类)

第二章 相关技术背景

第一节 [网络爬虫的定义](#_第一节_网络爬虫的定义)

第二节 [搜索策略剖析](#_第二节_搜索策略剖析)

第三节 [开发方法概述](#_第三节_开发方法概述)

结论 52

致谢 53

参考文献 54

# 第一章 绪论

## 第一节 课题背景

网络爬虫（web crawler），是一种按照一定的规则，自动的抓取万维网信息的程序或者脚本。。

网络检索功能与万维网的发展同步进行。网络内容的爆炸性发展同时也带来了对人们对特定内容检索的需求。这就是搜索引擎诞生的原因。而搜索引擎的核心就是爬虫。随着信息不断的丰富,人们的需求也在不断的提高,通过网络进行信息的搜索已经成为人们每天都要进行的内容.如何使搜索引擎能时刻满足人们的需求。最初的检索功能通过索引站的方式实现，而有了网络爬虫这个技术之后，网络爬虫的时代便开始一发不可收拾了。

## 第二节 课题意义

互联网是一个庞大的非结构化的数据库，将数据有效的检索并组织呈现出来有着巨大的应用前景。然而随着互联网规模的上升，在资源丰富的同时也带来了冗余的问题。搜索引擎是一个很好的搜索工具。但作为通用的软件，常规搜索引擎为了增加自己的通用性而不得不减少了一些专业性，这就是常规搜索引擎的的局限性。不同领域、不同背景的用户往往具有不同的检索目的和需求，通用搜索引擎所返回的结果包含大量用户不关心的网页。为了设计更好的完成搜索任务，精确搜索内容，一个灵活的爬虫有着无可替代的重要意义。

## 第三节 网络爬虫的历史

在互联网发展初期，网站相对较少，信息查找比较容易。然而伴随互联网爆炸性的发展，普通网络用户想找到所需的资料简直如同大海捞针，这时为满足大众信息检索需求的专业搜索网站便应运而生了。

现代意义上的搜索引擎的祖先，是1990年由蒙特利尔大学学生Alan Emtage发明的Archie。因为当时距离万维网的创建还有四年，但在网络对文件传输还是有很大的需求的，所以ftp（文件传输协议）软件成为了共享文件的主要工具。而且由于大量的文件散布在各个分散的FTP主机中，查询起来非常不便，因此Alan Archie工作原理与现在的搜索引擎已经很接近，Archie恰恰可以自动索引Internet上匿名的免费FTP文件信息，并提供一种根据文件名称查询文件所在FTP地址的方法。因此，Archie被称为现代搜索引擎的祖先。然而，客观的讲，它和一个真正的网络爬虫有着一些区别。原因有两个：一是它只能搜索FTP文件资源，并不能获取诸如网页等其他类型的文件资源，因此它其实是世界上第一个FTP爬虫。二是它没有机器人（Robot）程序，不能象今天的搜索引擎那样快速有效的抓取Internet上的网页文章内容，相反它使用的是一个基于脚本的文件名称收集器，并通过正则表达式来匹配用户查询与文件名称来实现查询，并通过文件列表的方式提供信息查询结果。

当时，“机器人”一词在产业界中十分流行。作为新兴的技术，爬虫自然被人们赋予了机器人的名字和期盼，“机器人”（Computer Robot）是指某个能以人类无法达到的速度不间断地执行某项任务的软件程序。由于专门用于检索的“机器人”程序象蜘蛛一样在网络间爬来爬去，寻找一个个的“猎物”—也就是信息。因此， 搜索引擎的“机器人”程序就被称为“蜘蛛”程序。世界上第一个可以被称之为网络爬虫的程序是Matthew Gray开发的World wide Web Wanderer。它的功能很简单，是用来度量互联网大小的，同时也可以用来统计互联网上的服务器数量，后来则发展为能够检索网站域名。Wanderer不仅可以说是一个真正意义上的网络爬虫，更可以说是第一个为万维网设计的爬虫，即万维网搜索引擎。而与Wanderer相对应，Martin Koster于1993年10月创建了ALIWEB，它是Archie的HTTP版本。ALIWEB不使用“机器人”程序，而是靠网站主动提交信息来建立 自己的链接索引，类似于现在我们熟知的Yahoo，作为一个爬虫来说，这种抓取信息的方式太过于被动，很多用户认为像网站提交信息就是浪费，这是ALIWEB不成功的地方，但对于整个网络爬虫的发展是很重要的一步，其开发者Martin通过这次失败预见性的思考到了不能任由程序设计者随意抓取数据，其参与设计的机器人拒绝协议也成为了当今爬虫的设计规范。

随着互联网的迅猛发展，使得直接遍历所有新出现的网页变得越来越困难，因此，在Matthew Gray的Wanderer基础上，一些有先见之明的编程者将传统的“蜘蛛”程序工作原理作了些改进。其设想是，类似于蜘蛛的行为，不进行无意义的闲逛，而是通过一些可能保存信息的通路，慢慢找到目标信息。如果按照计算机网络的说法，就是既然每个网页都可能有连向不同相关性的网站的链接，那么从一个标准的页面开始搜索，就有可能检索整个互联网。到1993年底，一些基于此原理的搜索引擎开始纷纷涌现，其中以JumpStation、The World Wide Web Worm（Goto的前身，也就是今天Overture），和Repository-Based Software Engineering(RBSE) spider最负盛名。

JumpStation的Web bot从网页上收集了关于标题和标题的信息，并为其Web界面使用了一个非常简单的搜索和检索系统。系统线性搜索数据库，匹配关键字。不用说，随着网络越来越大，JumpStation变得越来越慢，终于停止了。WWW蠕虫仅索引其访问的页面的标题和URL。它使用正则表达式来搜索索引。JumpStation和蠕虫的结果按搜索顺序排列，意味着结果的顺序完全不相关。RSBE蜘蛛是通过实现基于与关键字字符串相关性的排名系统，因此说RBSE是第一个在搜索结果排列中引入关键字串匹配程度概念的引擎 最早现代意义上的搜索引擎出现于1994年7月。当时Michael Mauldin将John Leavitt的蜘蛛程序接入到其索引程序中，创建了大家现在熟知的Lycos。同年4月，斯坦福（Stanford）大学的两名博士生，David Filo和美籍华人杨致远（Gerry Yang）共同创办了超级目录索引Yahoo，在开始时，他们称自己的网站为雅虎收集界面，但随着链接数量的增加，他们的网页每天开始获得数千次点击，团队创造了更好地组织数据的方法。为了帮助数据检索，雅虎（www.yahoo.com）成为可搜索的目录。搜索功能是一个简单的数据库搜索引擎。由于Yahoo!条目已被输入并手动分类，所以Yahoo!并未被真正归类为搜索引擎。相反，它通常被认为是可搜索的目录。雅虎自动化了收集和分类过程的一些方面，模糊了引擎和目录之间的区别。并成功地使搜索引擎的概念深入人心。从此搜索引擎进入了高速发展时期。目前，互联网上有名有姓的搜索引擎已达数百家，其检索的信息量也与从前不可同日而语。比如在国外拥有统治地位的Google，其数据库中存放的网页已达30亿之巨。

网络爬虫是搜索引擎的核心，窥一斑而知全豹，从搜索引擎的发展过程中，我们也可以看出来网络爬虫的发展步伐。一部搜索引擎史也就是网络爬虫史。从开始的简单抓取到现在的各种大规模的分布式、高度集群化等等。虽然距爬虫的出现才不带三十年的历史，但这样的发展速度是其他各行业都无法比拟的。这也凸显学习爬虫相关技术的重要性。

## 第四节 网络爬虫的分类

根据不同的应用，爬虫系统在许多方面存在差异，大体而言，可以将爬虫划分为如下三种类型：

　　一、批量型爬虫(Batch Crawler)：批量型爬虫是一种具有明确目标的爬虫，当爬虫完成任务的程度达到某个设定的目标后，即停止抓取过程。至于具体目标可能各异，根据情况而定。也许是设定抓取一定数量的网页即可，也许是设定抓取消耗的时间等。比如当下某些流行的头条类网站，其基础的爬虫架构就是批量型爬虫，爬取一定时间内的信息。总而言之，其区别于增量型爬虫的最大特点就是有限制。

　　二、增量型爬虫(Incremental Crawler)：谈到增量型爬虫，就不得不提待累积型爬虫，其实两者很相似，核心目标都是要维持一个庞大的网页的“仓库”。它们与批量型爬虫不同，会保持持续不断的抓取，对于抓取到的网页，要定期更新。累积型爬虫主要完成的是遍历存储全部的所能够访问的网页，而增量型是指在具有一定量规模的网络页面集合的基础上，采用更新网页的方法，在生成的网页“仓库”中通过一定的算法选择出过时网页进行重新抓取，用较小的代价来保证整个数据库中的网页与真实[网络数据](https://www.baidu.com/s?wd=%E7%BD%91%E7%BB%9C%E6%95%B0%E6%8D%AE&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1d9PvfkuHN-n16LnH0dujwB0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPH0LnHRdn1n1)足够接近。当然，可以进行增量式抓取的前提是系统已经抓取了足够数量的网络页面，并具有这些页面被抓取的时间信息。因为互联网的网页处于不断变化中，新增网页、网页被删除或者网页内容更改都很常见，而一个合格的爬虫需要保证的就是数据的时效性和可靠性。所以增量型处于持续不断的抓取过程中，不是在抓取新网页，就是在更新已有网页，通常情况下既包括累积式抓取，也包括增量式抓取的策略。累积式抓取一般用于网页集群的整体建立或大规模更新阶段；而增量式抓取则主要针对数据集合的日常维护中。通用的商业搜索引擎如百度、谷歌等爬虫基本都属此类。

　　三、垂直型爬虫(Focused Crawter)：以上两种谈到的爬虫，无论是批量型还是垂直型爬虫，它们聚焦的地方都是网页的数量选择，对于网页的内容不做太多的关心。而垂直型爬虫和两者切入的角度正好不同，垂直型爬虫多关注特定主题内容或者属于特定行业的网页，比如对于跑步社区网站来说，只需要从互联网页而里找到与跑步相关的页面内容即可，其他行业的内容不在考虑范围。垂直型爬虫一个最大的特点和难点就是：如何识别找到的URL所指向的网页内容是否属于指定行业或者主题。从节省系统资源的角度来说，不太可能把所有互联网页面下载下来之后再去筛选，这样做和增量型爬虫无异，设计出垂直型爬虫就是为了减少系统资源的浪费。因此，爬虫的设计者往往需要在抓取阶段就能够动态识别某个网址是否与主题相关，并尽量不去抓取无关页面，以达到节省资源的目的。很多专题网站或学科搜索引擎都是使用的这类型的爬虫。

# 第二章 相关技术背景

## 第一节 网络爬虫的定义

一、网络爬虫是一个自动下载网页的程序，它可以从Web上下载网页，是搜索引擎的核心组成部分。通用网络爬虫从一个或若干给定的标准网页的URL开始，获得初始网页上的URL列表；在抓取网页的过程中，不断从已下载页面上抽取新的URL放入待爬行队列，直到满足系统的停止条件。  
　　二、主题网络爬虫是基本网络爬虫的升级，它根据一定的网页分析算法将从网页中获取的全部URL剔除掉与主题无关的链接，保留主题相关的链接并将其放入待抓取的URL队列中；然后根据一定的策略从队列中选择提取要抓取的网页URL，并重复上述过程，直到满足系统的停止条件。所有被网络爬虫抓取的网页将会被系统存储到数据库中，进行一定的分析、清洗，并根据主题建立相应的索引，对于主题网络爬虫来说，这一过程所得到的分析结果还可能对后续的抓取过程进行反馈和指导。

## 第二节 搜索策略剖析

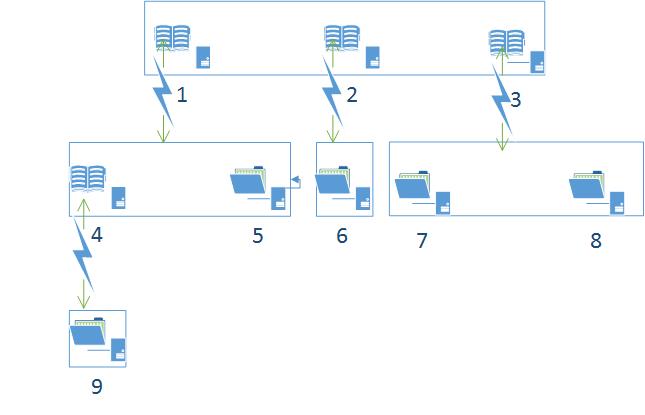
从上面的网络爬虫定义可以看出，网络爬虫的核心其实就是如何遍历所有需求的URL。通过一个给定的超链接，我们可以明确经由一个网站到另一个网站。这其实是就是有向图的遍历方法，接下来，我们就以这两种遍历方法作为切入点，剖析一下常见的搜索策略。

## 深度优先搜索策略:深度优先搜索是在开发爬虫早期使用较多的方法。在数据结构中，它的定义是假设给定图G，在G中任选一顶点v为初始点(源点)，首先访问出发点v，并将其标记为已访问过；然后依次从v出发搜索v的每个可以访问点w。若w未曾访问过，则以w为新的出发点继续进行深度优先遍历，直至图中所有和源点v有路径相通的顶点(亦称为从源点可达的顶点)均已被访问为止。若此时图中仍有未访问的顶点，则另选一个尚未访问的顶点作为新的源点重复上述过程，直至图中所有顶点均已被访问为止。这是数据结构中的定义，而在爬虫中，每个定点其实就是每个URL，它的目的是要达到被搜索结构的叶结点，也就是不包含任何URL的HTML文件）。在一个HTML文件中，当一个超链接被选择后，被链接的HTML文件将执行深度优先搜索，即在搜索其余的超链接结果之前必须先完整地搜索单独的一条链，形成一条从起始链接到最终节点的一条通路。深度优先搜索沿着HTML文件上的超链接走到不能再深入为止，然后返回到某一个HTML文件，再继续选择该HTML文件中的其他超链接。当不再有其他超链接可选择时，说明搜索已经结束。其优点是能遍历一个Web站点或深层嵌套的文档集合。缺点是因为Web结构相当深，有可能造成一旦进去再也出不来的情况发生。所以在实际的设计之中往往要根据所要求的数据对深入的层数进行限制，防止出现上面的情况。

二、宽度优先搜索策略:在数据结构中，有向图的广度优先遍历算法是一个层次遍历的过程，它根据所有节点与起始节点的距离的长短划分成一个个的层次。这和树的层序遍历算法十分类似，它也需要一个队列以保持遍历过的顶点顺序，以便按出队的顺序再去访问这些顶点的邻接顶点。而在爬虫的宽度优先搜索中，先搜索完起始页面中所有的超链接，这些超链接因为在一个页面上，距离当前访问页面的距离相同，所以可以看成是同层页面。然后再继续搜索下一层，直到底层为止。例如，一个HTML 文件中有3个超链接，选择其中之一，处理相应的HTML文件，然后不再选择第二个HTML文件中的任何超链接，而是返回，选择第一个页面里的第二个超链接，处理其中包含的全部的URL，再返回，选择第三个超链接，并处理其中的URL信息。至此，第一层的可以说是全部处理完毕，可以处理第一个链接中所包含的URL了。也就是说，一旦一层上的所有超链接都被选择过，就可以开始在刚才处理过的HIML文件中搜索其余的超链接。这就保证了对浅层的首先处理。这样做有两个很明显的好处，首先当程序遇到一个无穷尽的深层分支时，不会导致爬虫陷进WWW的深层结构中导致出不来的情况发生。宽度优先搜索策略还有一个优点，它能在两个HTML文件之间找到最短路径。宽度优先搜索策略通常是实现爬虫的最佳策略，因为它容易实现，而且具备基本完备的功能。但是如果要遍历一个指定的站点或者深层嵌套的HTML文件集等类似这种具有一定“线性特征的数据结构时”，用宽度优先搜索策略则需要花费较长时间才能到达深层的HTML文件。

三、聚焦搜索策略：前面两个提到的都是基于数据结构中图的遍历而相应产生的算法。而在实际应用中不会只是使用一种方法，而是两者兼而有之。对于一个具体问题来说，需要爬取的范围都是确定的。所以，如何将这种选择以代码形式显现就很重要了。聚焦爬虫就是一种多方面考察的爬行策略，它不只考虑页面与初始的拓扑结构，而是根据需求确定出某个特定主题的页面，根据“最好优先原则”进行访问，快速、有效地获得更多的与主题相关的页面，最后再通过内容与该页面上的链接结构指导进一步的页面抓取。简而言之，聚焦爬虫的核心是会给它所下载的页面一个评价分，根据得分排序插入到待抓取队列中。完成对当前页面的搜索后，根据评分的高低次序从队列中弹出下一个页面进行分析，这种策略保证爬虫能优先抓取那些最有可能成为高“价值”的页面。聚焦网络爬虫搜索策略的关键是如何评价链接价值，即人们为其设计的链接价值的计算方法，不同的价值评价方法计算出的链接的价值不同，表现在程序当中的就是链接的被访问的次序不同，从而决定了不同的搜索路径。由于链接是页面内容的一部分，因此通常具有较高价值的页面包含的链接也具有较高价值，因而对链接价值的评价有时也转换为对页面价值的评价。这种策略通常运用在专业搜索引擎中，因为这种网站具有一定的专业性，关注的是某些特定话题，对信息在广度和宽度上都有一定的要求，所以和聚焦搜索策略十分类似。

这样分析下来，我们为其实前两种搜索策略是聚焦搜索策略的一种拓展，广度优先搜索就是认为和网页处于同层的网页最具有价值。所以表现在遍历顺序上就是这样。深度优先搜索遍历同理。所以我认为爬虫的一个设计重点就是如何将需求中的价值以代码的形式表示出来。



通过上面文字的描述，让我们对三种搜索方式有了一个概念性的理解。下面我们结合一个具体的网络结构图来简单说明一下。左图（给图起名字）是一个简化了的网络结构图。每一个图标都表示这一个URL，其中样子为书状的表示这是一个只想网页的超链接，而样子为文件夹状的表示这是一个指向文件的链接（比如FTP链接）每个方框表示一个页面。闪电连线表示链接。从图上可以看出，1、2、3这三个超链接在同在初始网页上，4、5在一1所指向的网页上，6在2所指向的网页上，7、8在3所指向的网页上，9在4所指向的网页上。

第一个是深度优先搜索策略，我们就联想数据结构中的深度优先搜索遍历，在爬虫抓取到1后，立刻将1放到待抓取URL列表中。然后爬虫程序根据1找到所指向的页面。在1中有4和5两个URL,还是重复上面的步骤，在检索到4后立刻将其放到待抓取URL中，继续对4所指向的网页进行抓取，在9中有超链接，但是指向的不是一个新的网页，而是一个具体的文件。在爬虫完成对4所指向的网页的抓取后，因为并不包含其他的新的URL，所以爬虫程序回溯到上一层，对5所指向的网页进行抓取。就这样，爬虫逐渐的完成对整个网络的抓取。最后对各个链接的遍历顺序应该是:

1->4->9->5->2->6->7->8（修改公式格式）

接下来是广度优先搜索策略，我们也结合广度优先搜索遍历算法来看。与上面的方法不同，在检索到初始网页上的第一个链接1后，继续完成对初始网页的搜索。直到完成对初始网页的抓取，并将1、2、3三个URL全部放入待抓取URL队列中。然后从队列中取出第一个元素，也就是超链接1，对其所指向的网页进行搜索，得到四五后分别将其按检索到的顺序入队。检索完1所指向的网页后，继续从队头取出元素。根据爬虫所遍历的顺序，这时取出来的应该是2。如此不断执行，也可以完成对整个网络的抓取。最后对各个链接的抓取顺序应该是：

1->2->3->4->5->6->7->8->9(修改公式格式)

最后就是有关聚焦搜索策略的相关内容了，这是最贴合实际的情况。我们假设要抓取所有html文件和其时间，对其他的内容不做关心。也就是衡量一个超链接的价值的是这个超链接所指向内容的类型，对于需求来说，这个价值很好实现，如果指向的是html网页，就置其价值为1（将其放入待抓取URL链接中）；如果超链接指向的是其他类型的文件。就置其价值为0（不放入待抓取URL队列中。那么，如何判断这个URL指向的是html还是其他类型的文件呢？这就要具体问题具体来看了，在一个维护良好的网站系统中，不同类型的文件在其URL可以有明确的判断方式，如果在不访问的前提下就可以判断类型是最优的。当然，如果判断不出来可以进行试探访问以判断其类型，但这就没有前一种高效了。把如何衡量价值的问题解决后，抓取起来就很简单了，开始时类似于广度优先搜索遍历，在初始网页中抓取到1、2、3这三个超链接后分别对其进行判断。结果检查，都是网页，遂将它们全部置入待抓取URL队列中。然后从队头取出1，对其所指向的网页进行遍历，检索到了4、5两个URL。经过检查，发现5指向的并不是HTML文件，遂将其抛弃，仅将4加入待抓取URL队列中。就这样，一边抓取，一边对所抓取到的URL进行评估。直到爬虫完成对整个网络的检索。最后对各个链接的抓取顺序应该是：（5、6、7、8、9因为指向的是非html文件因而被抛弃。）

1->2->3->4(修改公式格式)

## 第三节 开发方法概述

### 所使用的语言：

（一）：在本次的课程设计中，我主要使用Python作为自己的开发语言。作为脚本语言的一种，Python有着其他翻译型语言不能相比的优势。Python非常简单，非常适合人类阅读。阅读一个结构良好的Python程序就感觉像是在读正常的文章一样，只不过这篇文章有着自己的特殊的语法结构，Python的这种易读的特性是它最大的优点之一。它使你能够专注于解决问题而不是把时间浪费在语法的纠结上。Python虽然是用c语言写的，但是它摈弃了c中让人头痛的指针，简化了Python的语法，使得无论是有没有编程经验的人来说，Python语言学习都十分简单。Python是FLOSS（自由/开放源码软件）之一。简单地说，你可以自由地发布使用Python语言构建的项目、阅读其他的项目的源代码、对它做你认为需要的改造。开源是计算机发展的趋势，因为只有让更多的人可以自由的参与进来，这种语言才有发展的趋势。Python也是如此，它很年轻，也希望看到能有更加优秀的人来不断的创新。作为一门出生在20世纪末的语言，在目睹了着从面向过程向面向对象的变迁后，Python语言既支持传统的面向过程的函数编程也支持着新兴面向对象的抽象编程。在面向过程的语言中，程序是由过程或仅仅是一个个孤独的函数构建起来的，重用性不高。在面向对象的语言中，程序是由很多相互之间有联系的、包括着各种数据和方法对象所搭建起来的。与其他主要的语言如C++和Java相比，Python以一种简单而又有力的方式实现面向对象编程。⑻丰富的库。Python标准库确实很庞大。Python有可定义的第三方库可以使用。它可以帮助你处理各种工作，包括正则表达式匹配、单元测试、线程、数据库、网页浏览器、FTP、XML、XML-RPC、HTML、GUI（图形用户界面）、以及其他多种的功能。记住，只要安装了Python，所有这些功能都是可用的。除了标准库以外，还有许多其他高质量的库，如BeautifulSoup、Requests和matplotlib等等。通过这些强大的第三方库，可以很容易的实现有关爬虫的相关操作。最后一点，Python语言写成的代码都十分的规范。这是因为Python采用强制缩进的方式，这种在语法上的强制约束使得代码具有极佳的可读性，不会因为个人的习惯导致代码的可读性变差。

（二）、Java（根据后续开发情况确定）

（三）、R，进行数据分析（根据后续开发情况确定）

### 二、使用到的关键类库：

（一）BeautifulSoup： 首先BeautifulSoup提供了一些简单的方法和Pythonic成语，用于导航，搜索和修改解析树；其次用于剖析文档并提取所需内容的工具包。编写应用程序不需要太多代码BeautifulSoup会自动将传入的文件转换为Unicode和传出的文档到UTF-8。您不必考虑编码，除非文档没有指定编码，而BeautifulSoup不能检测到。那么你只需要指定原始编码；最后BeautifulSoup是流行的Python解析器，如[lxml](http://lxml.de/)和[html5lib](http://code.google.com/p/html5lib/)，可以让您尝试不同的解析策略或交易速度来实现灵活性。相比Java中提供的文档解析工具httpparser，dom4j来说。我认为BeautifulSoup的功能更强大、也更灵活。符合Python语言自己的特点，通过匹配不同的parser，可以完成包括html、xml、html5等不同文件的解析，它可以把各种标签直接分析出来，而不像dom4j等需要一层层的查询。这样对于复杂网站的处理来说，大大加快了速度，提高了运行效率。而且对于非英语网站来说，BeautifulSoup的自动处理编码也大大简化了程序员为了处理乱码问题所需要的工作。

（二）Requests： Requests库是用[Python](http://blog.csdn.net/shanzhizi)语言编写的，基于 urllib（一个Python的原生类库），采用 Apache2 Licensed 开源协议的 HTTP 库。它比 urllib操作起来更加方便，可以节约我们大量的工作，完全满足程序员对HTTP的各种 需求。重要的一点是它支持 Python3 ，所以使用起来比其他的有关http库更加规范，安全性也更强。在我的设计中，涉及到了有关动态网页内容的抓取，BeautifulSoup因为只能保存转换成html的相关文件，因此就帮不上忙了。这时就需要Requests来帮忙，它里面包含的内容可以处理各种请求，这就大大方便了我们程序员进行操作。

### （三）re： re是Regular expression operations的缩写。该模块提供正则表达式匹配操作。正则表达式有能力去指定一系列与所给表达式在形式上匹配的字符串。Re模块是Python的内置模块，此模块包含的各类函数可以让程序员以多种方式来检查所检查的一些类字符串是否匹配给定的正则表达式或者是给定的正则表达式去匹配特定的字符串。这虽然看上去是正反两个不同的过程，但是其实可以归结为相同的事物。Python通过re模块提供对正则表达式的支持，使用re模块一般是先将正则表达式的字符串形式编译成Pattern对象，然后用Pattern对象来处理文本得到一个匹配的结果，也就是一个Match或Search对象，最后通过group方法得到所匹配的结果并进行操作。Re库的强大之处不仅在于是正则式的匹配，更可以在生成Pattern对象时通过指定各种匹配模式，来实现更强大的功能。

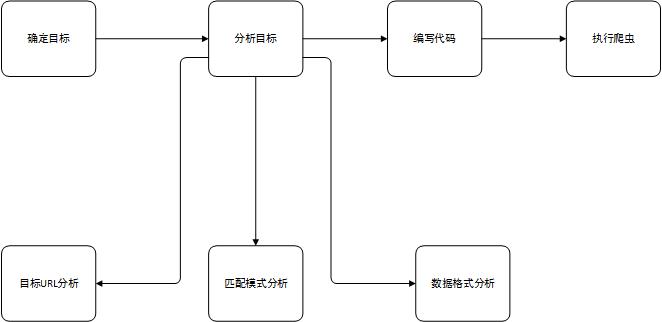
#### （四）httpclient（Java的类库，依据后面的程序情况书写）

#### （五）matplotlib（Python的绘图类库，依据后面的程序情况书写）

三、使用的数据库（暂时没有使用数据库，以后在程序完善后继续添加，现在想的是使用Mysql存放爬取的数据）

# 第三章 基本网络爬虫模型的分析和设计

## 第一节 基本流程分析：



爬虫设计流程图（待格式化）

这是一个普通的爬虫从开始到工作的流程。首先根据需求，明确所要抓取的具体目标。第二步就是分析目标。在明确具体的需求后，我们应该明确待抓取对象的URL的格式，以明确要抓取的范围。如果这个URL没有能够明确的指定的话，那我们的爬虫程序就会抓取很多与本任务无关的互联网上的其他网页，造成资源的浪费。其次，在明确了目标URL的特点后，就是进行匹配模式的分析了。这是我们就会用到正则表达式，这是一个很有效的工具，通过对正则表达式的合理创建，我们可以高效的从所有备选URL中筛选出我们所需要的那一部分。最后一项就是待抓取数据的格式。爬虫当然不会只抓取URL，链接只是一个路径，更重要的是抓取网页上的数据，所以待抓取数据的分析也很重要，不仅要考虑数据的格式，大小等基本信息。对于中文的抓取，更要考虑到处理编码的问题。第三步就是代码段编制了，再完成了有关分析的相关操作后，就是要根据上一步的分析得到的策略来编写代码了。从这些流程可以看出，这些步骤中最重要的就是分析这一步了。应予以多加留意。

## 第二节 百度百科词条爬虫设计

在了解完爬虫的相关知识后，我们以一个具体的实例来研究一个基本的爬虫是怎样诞生的。

1. 需求确定：百度百科是百度公司推出的一部内容开放、自由的网络百科全书平台。截至2017年4月，百度百科已经收录了超过1432万的词条，几乎涵盖了所有已知的知识领域。百度百科是一个涵盖各领域知识的中文信息收集平台。所以，我们想以百度百科中太原理工大学的页面<http://baike.baidu.com/item/太原理工大学>作为爬虫的入口地址，爬取太原理工大学词条页面以及与之相关的1000个百度百科的词条网页，并将这些网页的URL、词条名称以及简介都保存下来。

### 二、分析目标： 首先观察词条页面的URL的特点



太原理工大学百度百科的词条页面（待格式化）



211工程百度百科的词条页面（待格式化）

从上面的几个URL的格式可以看出，词条页面的URL是有一定特点的。都是以http://baike.baidu.com/item/为开头，具体所指向的页面在其后加上网页的主题即可。结构十分清楚明了。

然后是观察网页中所包含的URL的相关信息。通过浏览器所带的检查工具，我们可以观察页面上面每个部件的特点。

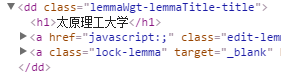


“211工程”的链接格式

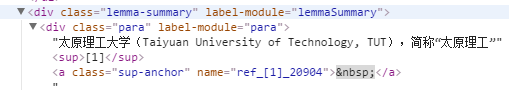
通过观察，我们可以看出。网页上所包含的链接都是不完整的。要想访问这些链接，我们需要在截取后手动的向前面添加baike.baidu.com来构成一个完整的URL。通过这两部我们就明确了目标URL的具体格式，完成了目标URL的分析。

第二步就是匹配模式的分析了，我们从第一步了解到，网页上所有指向词条的页面的都是以/item开头的后面接着一系列的内容。所以匹配模式也很简单，就是从网页上面的所有链接中筛选出以/item开头的链接就可以了。

最后是数据源的分析，我们还是通过检查元素来观察整个待抓取信息的结构。



审查元素-太原理工大学标题（待格式化）



审查元素-太原理工大学简介（待格式化）

从上面的审查元素可以看出，词条的标题是处在一名字叫dd的标签之中，class的名字叫做lemmaWgt-lemmaTitle-title。而简介的内容处在一个div中，其对应的class的名字是lemma-summary。在我们获取数据的时候，就可以通过类名来直接找到这两个部分的内容了。



页面编码（待格式化）

最后，我们通过审查元素再确认一下页面的编码方式，为UTF-8，所以在接下来的接收过程中程序要以以UTF-8的形式解码再处理，这样就避免了乱码问题的发生。

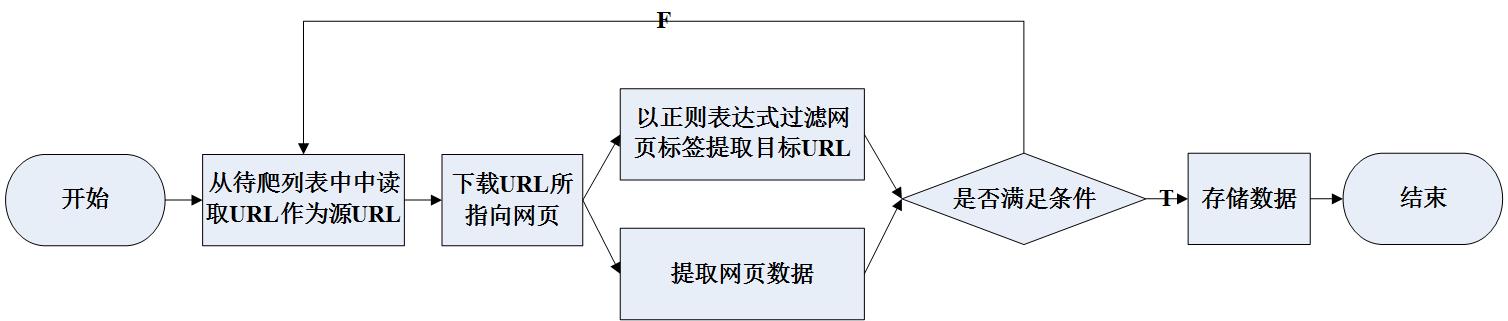
至此，我们对百度百科的网站的抓取策略已基本构建完成。

|  |  |
| --- | --- |
| 目标 | 百度百科太原理工大学相关词条页面-标题和简介 |
| 入口页 | <http://baike.baidu.com/item>太原理工大学 |
| URL格式 | http://baike.baidu.com/item/XXXX |
| 数据格式 | 标题：<dd class="lemmaWgt-lemmaTitle-title"> |
| 简介：<div class="lemma-summary" label-module="lemmaSummary"> |
| 页面编码 | UTF-8 |

抓取策略表（待格式化）

### 三、系统设计：

首先，应确定整个程序的工作流程



因为本程序抓取的网页数量较少，所以没有使用数据库存储待抓取URL列表，直接存储在set集合中，也就是内存之中，这样也可以增加程序的运行速度；网页上的数据也保存在一个对象之中（内存）。最后的数据存储也是准备通过一个方法调用Python的文件输出流，创建一个html文件并将目标数据写到所新建的html文件中，只要用户有浏览器就可以查看信息。（后期考虑将数据同时存放在数据库之中）

根据本网络爬虫的设计，本网络爬虫是一个自动提取网页的程序，根据从待爬队列中抽取的网页进行下载，再根据一定的策略将其中的符合要求的URL和页面上的数据同时保存下来，如此循环往复直到满足用户的需求或者网页抓取完毕。

（后期考虑增加一个有关的图）

### 四、代码编写：

首先是编写主函数，内容很简单，就是通过创建一个调度类的对象，开始网页的抓取。

然后就是调度类（spidermain）的编写，因为他是调度类，所以在本类的构造函数中先要创建分别创建URL管理器、HTML下载器、HTML解析器和HTML输出器的对象。创

创建好四个对象后，就要开始爬虫主程序的编写。在上一步已经完成了对流程的分析，所以直接根据流程图撰写代码，就可以完成有关爬虫调度程序的编写。

核心代码如下：

通过URL管理器获得一个新的连接

new\_url =*self*.urls.get\_new\_url();

将刚刚获得的地址以参数的形式传入HTML下载器，下载URL所指向的界面

html\_cont=*self*.downloader.download(new\_url)

通过调用HTML解析器解析所下载网页中所包含的URL和数据

new\_urls,new\_data=*self*.parser.parse(new\_url,html\_cont)

存储URL和数据

*self*.urls.add\_new\_urls(new\_urls)

*self*.outputer.collect\_data(new\_data)

打印数据到HTML文件中

self.outputer.output\_html()

这样调度类的编写就完成了，调度类没有什么业务逻辑，但是指导着整个爬虫程序的进行，十分关键。

接下来是URL管理器（UrlManager）。这个方法的内容很简单，主要由4个方法构成add\_new\_url(self,url)，向管理器中新增加一个URL， add\_new\_urls(self,urls):向URL管理器中增加一系列的URL， has\_new\_url(self):判断URL管理器中是否还有待爬取的URL，get\_new\_url(self)获取一个新的URL。这四个方法没有什么复杂的地方，就是注意一点，由于互联网上各个界面可能存在着指向对方的链接，所以在遍历完一个网页后，不能简单的删除掉，而是应该在整个程序中维持两个集合。一个是待访问集合，而另一个是已访问集合。

核心代码：url not in self.new\_urls and url not in self.old\_urls:

当调度程序调用URL管理器向程序中新增已爬取的网页时，要判断这个链接是不是已经在待爬取集合内或是不是已经访问过一次了（在已访问集合中）。在访问完网页后，不仅将这个URL从待爬取集合中删除，更要将其添加到已爬取集合中，这样才能保证数据的完整性。多设置一个队列可能会对多占用一些内存空间，但带来的好处不仅仅只有减少因重复访问同一网页带来的资源浪费，更在一定程度上保证了程序的稳定性，防止程序调入循环抓取的陷阱中。

当URL管理器给出了一个待抓取的URL后，就轮到HTML下载器工作了。下载器负责的是将网页的内容下载下载，就和我们平常使用浏览器浏览网页一样。只不过我们只是将对应的html文件下载下来，不进行解释。这只需要调用urllib库下的request对象的urlopen方法，再讲其中的内容通过调用read（）返回给调度程序。但值得注意的是，因为在先前的设计当中，我们了解到百度百科的全部网页的编码方式都是UTF-8，所以在回传下载好的HTML文件时，要注意通过提供的decode方法将解码方式设置为使用UTF-8解码。这样做可以减少乱码问题发生的可能性。

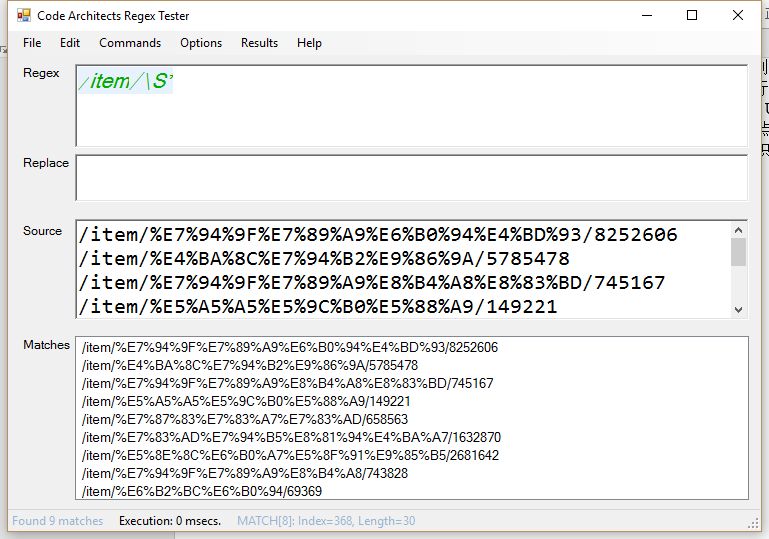
在得到网页的文件后，调度程序就要利用HTML解析器来解析下载好的文件了。HTML解析器的编写与我们之前分析所制定的策略息息相关。

HTML有一个函数是调度函数直接调用的，就是parser函数。调度程序要依据这个函数，输入下载好的HTML文件，得到解析后的数据。我们先利用BeautifulSoup提供的方法将传入的HTML文件转化成BeautifulSoup对象。然后我们可以将这个函数具体实现拆成两个小函数来分别实现，第一部分负责从HTML文件中解析符合条件的超链接。第二部分负责从HTML文件中解析出目标数据即百科的标题和简介。这两部的执行没有关联，所以可以分开进行。有条件还可以将程序做成多线程以进一步提高效率。

第一部分是抓取所有符合条件的URL，也就是那些依旧指向百度百科词条的超链接。

links = soup.find\_all('a', href=re.compile(r"/item/"))

这句话通过BeautifulSoup所提供的的find\_all方法，可以轻松找到网页中的所有超链接。但咱们所抓取的只是百度百科的页面，再find——all方法中不行要对标签的类型进行限制，还要对超链接标签的href属性进行约束。通过上面的目标URL分析的结果中我们知道，百度百科词条页面上的指向其他词条页面的超链接的特点：href属性都是以“/item/”开头的，结尾是内容而定。所以利用正则表达式的相关知识，同时结合目标字符串的特点，写出了/item/\S\*的正则表达式。



正则表达式检验（待格式化）

通过正则表达式检测工具，证明表达式书写争取，可以正确匹配到目标网页。

获取到所有百科词条的URL后，因为这是不完整的，不可以直接存储，否则会造成无法访问的问题。所以接下来程序要对生成的links集合进行遍历，修改每一个链接，让他们变成完整的链接。

new\_full\_url = parse.urljoin(page\_url, new\_url)

这里所使用的不是简单的字符串的拼接，而是使用了一个urljoin的函数，这个函数有两个参数，返回值是将第二个参数所代表的字符串根据第一个参数的格式进行转化后的类型。其实可以理解成相对路径和绝对路径的转化。第二个是相对路径，第一个是绝对路径。这个函数完成的就是从相对路径到绝对路径的转化。

通过匹配、补全这两步。HTML解析器完成了第一部分的任务-将HTML文件中所包含的指向其他界面的URL解析出来。

第二部分是从HTML文件中解析出词条的标题和简介。

因为前面分析工作做得很好，所以这部分解决起来也很容易。

title\_node =soup.find('dd',class\_ = "lemmaWgt-lemmaTitle-title").find("h1")

res\_data['title']=title\_node.get\_text()

summary\_node =soup.find('div',class\_ = "lemma-summary")

res\_data['summary']=summary\_node.get\_text()

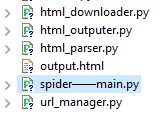
还是通过调用BeautifulSoup提供的find方法，同时指定标签名和类名两个属性来获取对应的节点。再得到节点后直接使用get\_text()方法来获取节点的内容。因为这两者都已经是根节点了，所以可以使用get\_text()方法来获取值，这里调用.string的属性值也可以达到相同的效果。

通过BeautifulSoup提供的方法，我们很快的完成了第二部分的任务-将HTML文件中所包含的词条标题内容和简介内容解析出来，并存放到一个字典类型的数据中。而且为了日后可以编写的根据这个标题和简介打开自己的目标网站，程序将词条页面的URL作为一项数据和词条标题、词条简介共同保存了起来。

因为爬虫要不断进行循环抓取，所以在上一步获得的数据需要立刻保存起来。因此在完成了解析，返回了具体的URL数据和网页数据后。URL管理器和HTML输出器分别将URL和网页数据保存起来。URL管理器的增加方法没有值得注意的地方，批量增加新的URL就是对每个URL调用前面所编写的增加一个URL的方法即可，还增加了重用性。就是有关HTML输出器，在循环爬取的过程中，因为程序还没有结束，所以它首先要保证每次从HTML解析器中获得的字典类型的数据都保存下来。所以程序应该在HTML输出器中定义一个集合对象，每次爬取完一个网页后，就将从这个网页中所提取出的数据保存起来。直到整个爬虫停止后，再一并输出。

当经过了网页获取，网页下载、网页解析和数据转存后。爬虫的一次爬取工作就算完成了，接下来就是要根据用户的需求和网页中所包含的链接数再进行不断的重复爬取了。当爬虫停止后，就该调用HTML输出器的output\_html()方法，将前面每次爬虫所收集的数据打印输出到HTML文件中。这里没有什么问题，唯一要注意的就是注意编码，在声明HTML时要制定好编码方式为UTF-8，这样才与Python保持一致，否则将会产生中文乱码问题。

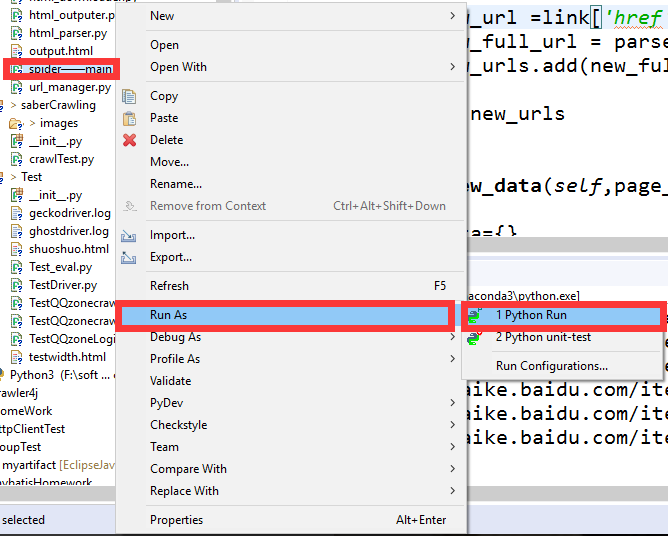
五、运行爬虫



百度百科爬的文件结构。

其中spider——main.py为程序的入口地址，里面包括着主函数以及爬虫的调度类。url\_manager.py为URL管理器、html\_downloader.py为HTML下载器、html\_parser.py为HTML解析器、html\_outputer.py为HTML输出器；output.html为程序最后的输出文件。

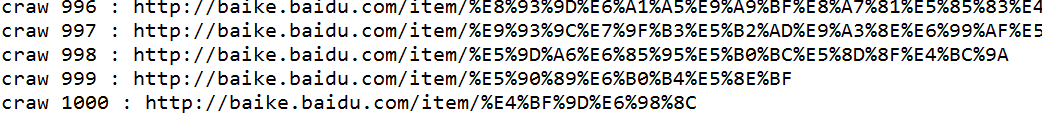
运行步骤:



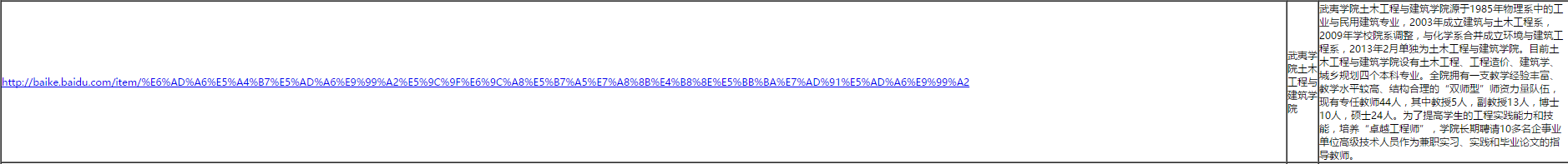
程序执行步骤（待格式化）

右击main函数，选择Python Run。程序即开始执行。

在经过一段时间后，程序运行结束。



程序运行结果（控制台）（待格式化）



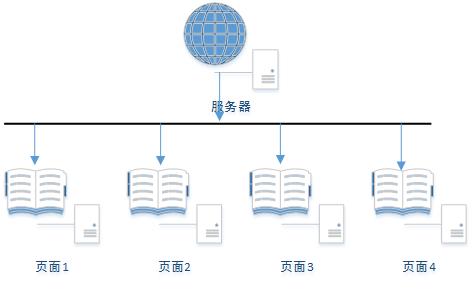
程序运行结果（HTML文件）（待格式化）

最终的结果表明我们就完成了一个基本爬虫的构建。从需求到最后的实现，核心就是分析阶段。无论是设计什么软件，重要的地方都是有关于设计。只要分析好了目标，设计好了结构，后面的编码都是轻松解决的。

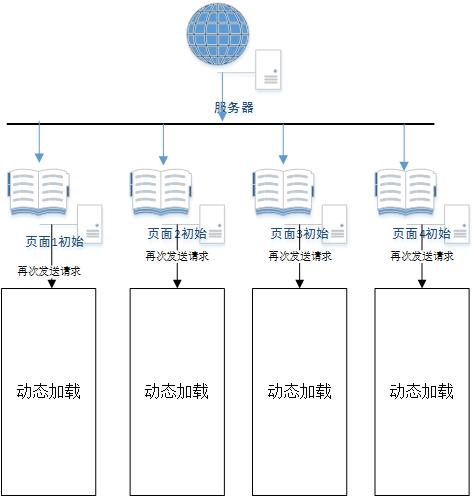
这个百度百科的抓取程序只是对静态页面的抓取，下面我们分析两种关于动态页面的抓取方法。

## 第三节 网站结构分析：

自从推特首创的下拉刷新这一模式后，越来越多的网站特别是社区类网站采用这一模式。在浏览网站上的内容时，它们不仅将内容分页，更将每页包含的内容分成几个部分，当客户请求的时候再加载，提高了交互性。这样做，不仅可以让用户更自由的选择浏览的数量，更可以通过这种将大任务分成几部分的小任务，从而可以更好的均衡服务器的负载水平。



如图所示的就是传统的网站的结构图，百度百科就属于这种结构。从图中我们可以看出，服务器中包含着每个网页自己的文件服务器。当浏览器发出请求后，返回的就是整个页面。适合内容较少的页面，如百科类这种以文字作为主要内容的网页。

如图所示的就是现在大部分的社区类网站的结构图，如花瓣网、QQ空间等。从图中可以看出，当浏览器向服务器发出一个请求后，服务器向用户返回一个网页，但这个网页不是全部显示的，而是先显示一小部分，而本页的其他内容是动态的根据用户的需求加载。这种架构适合于社区网站。因为作为社区类网站，网页的内容不仅仅包含文字，还可能包含着很多的图片甚至视频，如果一次将一个网页的内容全部加载下来再提供给用户来看的话，会极大的增加用户的等待时间，影响用户的体验，所以可以通过这种动态加载的方式，让用户边浏览，边加载，提高用户的体验，同时也增加了用户的访问速度，达到均衡服务器的目的。

这是两种网页的架构模式，只有弄清楚所要抓取的网站属于哪种模式，在设计爬虫时根据相应的类型设计，才能达到高效便捷的目的。像百度百科属于第一种方式，所以在设计抓取方式时，直接将页面下载再分析就可以得到网页上的内容了。但如果要抓取动态网页就要根据不同网站的加载的方式来进行具体的分析了。

## 第四节 花瓣网图片抓取爬虫设计：

1. 需求分析：

花瓣网是一家图片分享网站，是一家基于兴趣的社交分享网站，网站为用户提供了一个简单地采集工具，帮助用户将自己喜欢图片重新组织和收藏。花瓣网的模式是采用瀑布流的形式展现图片内容，无需用户翻页，新的图片不断自动加载在页面底端，让用户不断的[发现](http://baike.baidu.com/item/%E5%8F%91%E7%8E%B0)新的图片。我们的目标就是根据某一个选定的主题，自动的下载指定数量的相关图片。这次就以家装作为主题，我们去花瓣上下载100张相关图片。

## 二、分析目标：

首先先观察目标页面的URL：







几个搜索结果页面URL

http://huaban.com/search/?q=\*\*\*是搜索结果的结构，前半部分是固定的，改变的只是最后的等号后的词，对应着搜索的内容，也就是说，如果要抓取某个方面的主题，就相应的将最后的关键词替换成相应的词就可以了。

在了解到了目标网页的URL后，接下来的就是如何将网页上的图片下载到本地上的问题了。平常我们的下载都是利用浏览器自带的功能，获取到图片对应的URL后，下载到本地。网络爬虫也是同理，我们需要先明确图片的格式，然后获取100张图片的链接。最后根据这些链接将图片下载下来。

为了保证下载的图片的质量，我们选择下载原始图片而不是图片的缩略图。为了了解原始图片的格式，我们同时查看几张图片对应的缩略图和原始图的URL，用以明确原始图片的URL构成：







原始图片的URL  

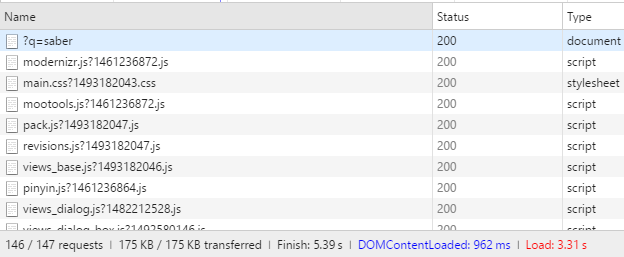





缩略图的URL

通过这两者的比较，我们可以清楚的看出来原始图片的URL构成，前半部分是固定的花瓣网的图片标识：<http://img.hb.aicdn.com；URL>最后是固定的代表原始图片的表示：\_fw658。而中间的一部分是代表着具体图片的标识（key）。所以，现在问题的关键就转向如何获得具体图片的标识了(key)。

和百度百科相比，花瓣网的网页内容较多，构成复杂，直接观察网页的源码难以看出，我们可以利用浏览器中自带的检查工具来观察加载网页时加载的内容，以此来寻找图片key值的获取方式。

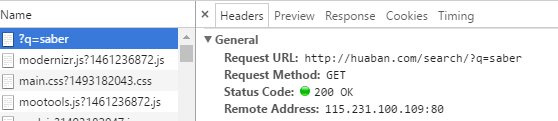


初次加载网页所加载的内容

可以看出，初次加载网页时接受的对象有很多，其中大部分是脚本，这是因为开始请求的网页，除了图片外，还有网站本身的框架、广告、插件等内容，所以有很多的脚本。但是第一项名为?=saber的内容很值得人注意。它的名称正好就是搜索网址最后一部分的内容，而且等号后面的内容也正是搜索的关键词。

所以我们点开这个document对象来仔细分析一遍。

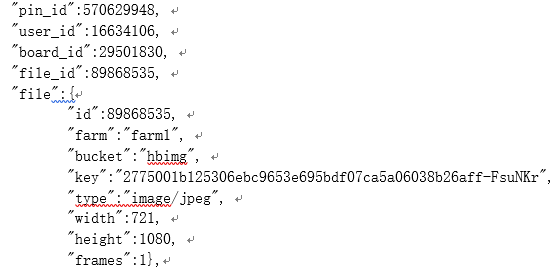
首先分析请求头部。



请求头部

通过图片可以看出，这个请求的URL正是搜索网页的URL，这也进一步证明了我们的猜想。这个document对象中包含的内容和图片的信息有很大的关系。那为了获取到图片的具体信息，我们需要进一步去翻看通过这个请求响应回来的具体内容。因为返回的也是一个html文件，为了查看分析方便，我们把这些内容复制到一个文本编辑工具里面查看比较方便。

通过对内容的大概浏览，我们发现这些变量名称的定义都很规范，通过他们的名称就可以大概理解这些变量的具体作用，类似于setting、scheme、count等等。可以看出这些都是有关网页格式的限定，而非图片信息。经过仔细寻找，我们发现了一个名为app.page["pins"]的列表，里面保存着多个字典，在这些字典中有着很多的名称很醒目的键值对，如pin\_id、user\_id和key等等，这些名字中带有id的属性应该就是唯一标示图片的具体信息。我们可以利用获得的这些信息来反过来验证猜测。



一个图片的相关信息（部分）

通过检查元素审查网页，再结合我们在上面响应中的得到的信息，利用搜索的方法，我们可以了解到:pin\_id是一个图片的具体的标识，而其中file字典中的key正是我们需要找的那个决定文件存储路径的key值。这两个属性是我们所必须的，有了图片的id和其对应的的key后，我们就可以自动构建图片的下载链接了。

据此，我们可以总结获得图片URL的基本步骤：

1. 向http://huaban.com/search/?q=（主题词）发送请求
2. 接受到相关请求后解析html，从中截取出名为app.page["pins"]
3. 遍历app.page["pins"]中每个元素，获得每个元素的key值。
4. 组合成图片的URL

在解决了如何获得一个图片的下载地址后，然后要解决的就是本爬虫最关键的部分了。作为图片分享网站，花瓣网对于一个搜索主题并不是直接显示全部的搜索结果，而是先显示一部分，当用户下拉网页到当前所加载内容的最后部分时，网页再加载一部分图片。所以，当我们向http://huaban.com/search/?q=saber发送了请求后，其中的app.page["pins"]所包含的仅有部分图片，也就是第一次加载的图片。所以，现在的关键就成了如何模仿用户的下拉操作，继续向服务器请求图片。

我们继续使用检查工具来完成这个任务。我们可以利用检查工具中的网络分析部分，观察在下拉的时候向网站发送了哪些请求、收到了哪些响应以及响应中包含了那些内容。



下拉时收到的响应

从图上可以看出，在下拉网页时，浏览器收到了一个xhr对象，该对象的名字和第一次加载网页时浏览器所收到的那个document对象类似。由此推断两者的内容也应该类似，但第二个对象的获取是在网站的基本设置已经加载完成后，所以内容中有关的网页的设置应该会减少，而主要是关于图片的相关信息。这些就是根据该对象的名称所进行的推测，具体的详情还应该要根据具体的相应内容来判断，但依据自己学习到的内容进行一定程度上的推测，可以让我们在筛选内容时更加具有目的性。

第二个值得注意的地方就是XHR对象，和第一次的document对象不同，第二次接收的不在是静态的文本对象，经过上网查阅相关资料，XHR是AJAX功能实现所依赖的一个对象，该对象的全称是XMLHttpRequest。XMLHttpRequest 对象提供了对 HTTP 协议的完全的访问，包括做出普通的get请求的能力以及各种高级的请求。XMLHttpRequest 可以自由的选择返回 Web 服务器的响应的方式，包括同步和异步，并且能够以文本或者一个 DOM 文档的形式返回内容。简而言之，这个对象返回的也是一个文本文件，但是属于动态生成的。这也正是AJAX的特点，在不更新网页整体的条件下，对局部进行内容进行更新，从而减少网络资源的消耗，提高用户的体验。避免了类似于那种一页页翻页的网站出现的资源重复加载所带来的浪费问题。既然XHR对象返回的内容属于文本，那就很简单了。首先，需要找到xhr类型的url地址，然后根据地址发送相应的AJAX请求，最后解析响应中的内容。

那首先就是确定请求的URL了，和上面类似，我也进行多次的加载来观察请求URL的特点：









几次请求的URL

通过观察我们可以发现，获得XHR对象的请求URL的特点很明确。改版的就只有page参数。改变方式为第一次为2，随着每次刷新逐次的加1。从这里也可以看出来每次加载的图片是20张，据此我们可以确定20张图片是一组，在确定好下载的数量后，就可以相应的确定出需要下载的组数。我们还可以看出来这里发送请求所使用的方式是GET方法。为了确定自己的结论，我又更换了自己所查询的主题词来观察不同主题词下动态加载的URL是否有所区别。



通过比较可以看出，刚才的结论没有问题。在每次的请求时注意根据请求的次数来修改page的值。

在解决了获取URL的问题后，接下来的就是有关解析具体内容的问题了。这是观察请求对应的response：

这个response对象中包含的内容也是一个列表，在这个列表中包含着两个字典：第一个是名为query的字典，保存这有关jQuery的相关信息,AJAX作为动态刷新技术，需要配合JQuery来从数据库抓取数据。另一个就是名为pins的字典了，既然属性名和第一次加载网页时包含图片信息的那个字典的名字一样，那它的内容也不言而喻了，两者的结构完全相同，故可以采用相同的解析方法。

至此，对所有链接信息的分析已经基本全部完成了，接下来就是匹配模式分析了。现在的需求是从相应中提取所有图片信息，也就是获得名为app.page["pins"]的字典。但是现在有两种不同的响应，一种是网页初始加载时的静态响应，还有一部分是属于根据需求图片数量而进行的动态请求所获得的响应。这两部都是获取信息的操作，应该整合在一部分，也就是说我们应该寻找一种同时可以符合两种情况的匹配模式，从而可以更好的整合相同部分的功能。

因此，我们先对两种请求的内容使用Python中的requests库内的get方法进行请求，来具体观察特点。结果测试，发现无论是哪种请求所获得的内容在结构上都是相同的。所有的图片信息都包含在名为app.page["pins"]的字典中。这与上面使用浏览器自带的检查工具所抓取的结果不一致，但结果查阅相关资料，表明这两者是一致的，JQuery通过其底层的AJAX对象的get方法返回一个XHR对象，前面使用拦截器发现的XHR对象就是这样生成的，但对于JQuery还没有完成，最后还是要把这对象中的内容添加到网页中，也就是初始加载的网页中。所以两者有着相同的属性名”pins”。

进行完匹配模式的分析后，我们最后对目标格式进行分析。我们本次要抓取的对象是花瓣网上的图片。这就存在两个问题。第一个是网站的问题：随着网站的建设，现在的网站上不仅存在着图片，还包括着视频，故第一个问题是甄别出目标的文件类型。第二个是图片的问题，仅仅对于图片来讲，就包含着很多种格式如JPG、png等等，故第二个问题就是确定所下载图片的具体格式。通过对app.page["pins"]的分析，我发现在其中存在着名为type的属性。



这个属性以下划线分成两部分，第一部分是指定的文件类型，而第二部分则是指定的此文件的文件名。故我们可以抽取出来每个图片信息中的type属性，对下划线两侧的内容进行分别处理，可以同时解决这两个问题。这样数据格式分析的任务也就完成了。

最后，我们通过审查元素再确认一下页面的编码方式，为UTF-8，所以在接下来的接收过程中程序要以UTF-8的形式解码再处理，这样就避免了乱码问题的发生。

至此，对于目标的分析就基本完成了。

|  |  |
| --- | --- |
| 目标 | 花瓣网图片爬虫 |
| 入口页 | <http://huaban.com/search/?q>=关键词 |
| 动态请求格式 | http://huaban.com/search/?q=saber&j1zt9zn0&page=20&per\_page=20&wfl=1 |
| 数据格式 | Image/jpeg 图片/图片类型 |
| 页面编码 | UTF-8 |

### 三、系统设计

经过上面对目标的分析，我们也大致了解任务目标和任务所要采集的内容。如基本爬虫一样，总体的步骤还是收集URL、分析URL、最后下载相关信息。但与百度百科爬虫不一样的是。这里收集URL分为两部分，第一部分是收集主页信息，第二部分是收集动态请求。收集主页信息的方法很简单，因为已经给出了网站的初始地址，利用requests库所提供的get方法直接就可以获得网页的内容。有关动态请求的获取就需要动态构建，即根据请求的次数来动态生成相应的请求URL。

分析URL就是将URL所对应的HTML文件中的有关图片信息下载下来。这里的HTML文件有两个部分，一部分是从初始页中下载的，另一部分是根据网页动态加载的。虽然HTML文件被分为了两部分，但通过上面对于目标的分析我们可以了解到这两个HTML文件的结构是相同的，所找寻找的图片信息在相同名称的一个字典之中。因此，为了提高程序可可重用性，我们可以将对于数据的分析只完成一个模块，然后传入不同的HTML文件来进行解析，将来自不同HTML文件的图片信息存储到一个相同位置。

下载相关信息对于本任务来讲就是将获得的图片的URL下载下来。利用Python所提供的的文件操作可以直接将一个请求的内容直接下载下来，需要注意的就是做好文件操作中可能出现的异常捕获。



爬虫总工作流程

### 四、代码编写

进行完总体的设计过后，就该进行代码的编写了。

首先是主函数的编写，主函数的编写很简单，就是声明一个对象，再调用相关方法。

然后就是具体类的编写了，我们声明了一个名为HuabanCrawler的类，由这个类完成抓取图片的所有操作。

第一部分是接收初始网页和确定下载图片的数量。我们通过类的初始化来完成操作。



其中，homeurl定义了起始网页，也就是搜索详情页的URL。images列表保存所有的图片信息。最后，因为需要将保存好的图片信息下载下来，所以在当前文件夹下建立一个目录来保存图片。

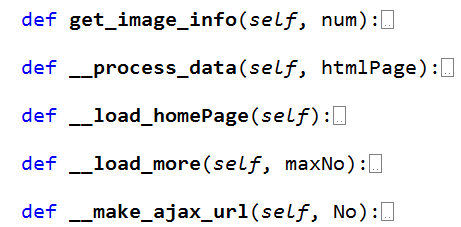
第二部分是获取图片信息。这一部分是整个爬虫的核心，通过这个模块，爬虫需要将全部待下载的图片的信息获取到。我们可以对这部分进行一个分解：



接收图片信息的具体流程

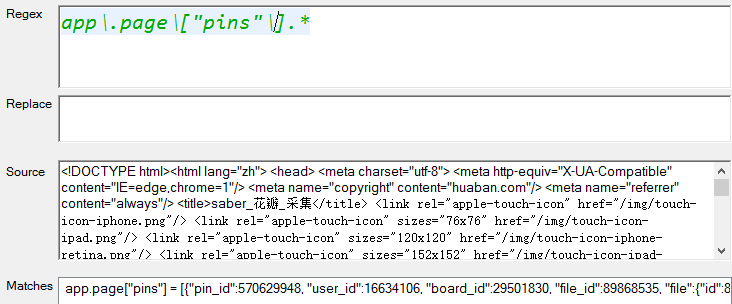
整个的获取图片信息可以分成四步来进行。第一步是将初始页上的图片信息下载下来；第二步是根据所要下载的数量，计算需要请求的次数；第三步是根据计算出来的需要请求的次数，动态的发出请求来获取响应中的网页信息。最后就是根据第一步和第三步中获得的网页信息，解析出其中有关图片的信息保存下来。

为了完成这些功能，我们在类中一共定义了5个函数：



get\_image\_info函数是完成获取图片信息的主要函数，负责完成各个函数的调度，完成这部分的事务逻辑部分。从之前的分析，我们了解到一共有两部分的图片信息。第一部分是初始页面上的，还有一部分信息是在后来动态加载的页面上。所以，这个函数的编写也是分为两个部分，一部分是获取初始页面的的信息。就通过调用函数实现，也不需要传入参数，直接使用初始化时定义的变量就可以。第二部分是就需要进行分析了，在分析目标时，我们就清楚了每个页面的特点。首先，每个页面有20张图片，如果一共需要下载100张图片的话，那么抛去初始页上的20张，还需要动态加载的次数就是（100-20）/20=4次。这样分析后，那整个函数的结构就很明确了，一共分两部分：第一部分调用函数获得初始页上的信息保存起来。第二部分先计算需要下载的次数，然后通过for循环来重复调用函数以获得动态网页上的内容，需要注意的是，根据我们前面的分析，动态请求每次的URL都有不同，page属性是从2开始随调用次数的增加而逐渐增加的，所以我们在调用函数时应该让传入的参数匹配当前进行的请求中的page参数，以免发生请求的URL为不存在的问题。

\_\_process\_data函数是这部分的核心函数，传入的参数是一个网页的源文件，功能是从传入的网页源文件中截取出有关图片信息的部分，即选择出名为app.page[“pins”]的列表，将其内容保存起来，并将其中有关于图片的信息保存下载，最后完成第二部分的业务逻辑功能。首先要完成的，先将网页中保存着的有关app.Page[“pins”]的列表匹配出来，这里用到的就是正则表达式。通过前面的目标分析，我明知道这里的待匹配项目很简单，就是一个固定名称为app.page["pins"]的列表，因此我们只要匹配这个名字加上器后面的内容就可以了。所以我们可以直接写出此表达式为r'app\.page\["pins"\].\*'。我们可以利用正则表达式测试工具来对自己输入的内容进行测试。观察室否可以成功匹配到网页上的相关内容:



模拟正则表达式匹配的结果

从图上我们可以看出，我们设计的这个匹配式可以顺利的完成关键字符串的匹配。但是在匹配之前要注意在匹配前先将页面的解码方式转换成UTF-8，否则会发生编码错误。



防止产生编码错误

在获得了图片信息的相关字符串后，为了可以分类获得各项属性的值，我们需要将这个字符串的序列转换成一个字典。我们可以直接利用python提供的eval函数来直接完成这个操作。eval（str）可以将字符串str当成有效的表达式来求值并返回计算结果。此函数可以自己判断内部字符串符合的是哪种格式，然后进行相应的求值。在上面的测试中，我们可以看出图片信息是符合列表的格式，因此，当调用eval函数后，上面求得的有关图片信息的字符串会自动转换成列表。

但需要注意的是，首先，在Python中，大部分的常量关键字都是以首字母大写的形式存在的，但是在图片信息的列表中存在着好多类似false、true这种纯小写的单词，同时也存在着null这类在Python中没有定义的变量。所以在转换之前，我们需要将一些变量进行提前赋值，将在Python中存在的但非大写的转换成在大写，而在Python中不存在的转换成近义词，防止在转换过程中出现错误。



Python中以None取代了其他语言中的null

在解决完Python中不存在的变量后，还有一个问题需要注意，在前面的分析目标中，我们知道图片信息是存在于一个列表中，该列表中保存着好多各个图片信息的字典。但是我们在转换的时候不需要前面的列表名称，在转换的时候，我们只需要将整个列表的内容截取出来就可以了，这时我们就用到了Python中的切片功能，利用这个功能，可以简单的从字符串中截取一部分出来。该功能类似于C中的字符数组，一个字符串中的各个字符都看成一个数组中的各个元素，利用数字索引来访问。



从上图我们可以看出，需要的内容就是从第一个pin\_id所在的字典开始，也就是从第20个字符开始计算。因为元素的索引从0开始，所以切面因该从19开始，到最后结束，因此我们可以将切片写成[19:-1]。



字符串转换为列表

当通过正则表达式匹配出的字符串成功转换为包含图片信息的列表后，接下来就是将需要的信息从列表中个各个字典中取出了。这里我们用到的就是循环遍历了。



在前面的分析中，我们一共需要从一个图片信息中获取三部分的内容，分别是图片的ID，图片的key值以及图片的类型。这三部分内容都在字典中每个元素中，所以可以直接通过元素的名字来访问。id属性比较简单，没有需要进行操作的内容，直接取出内容存储就可以。第二部分是图片的key值，但在前面的分析中我们知道，首先这个key值在图片字典中的file子字典中，所以需要使用二级索引来访问。其次，在通过索引获得了对应的值后，也不能直接存储，因为key值只是图片下载地址的一部分。所以我们需要手动进行字符串拼接，也就是在key值前面加上搜索详情页的前缀，在key值后面加上完整图片的完整后缀\_fw658。最后一部分的文件类型的判断。我们知道，类型的属性值通过一个下划线分为两部分，前一部分是文件的类型，后一部分是文件的具体类型。所以，对于图片类型的判断需要分成两部分来进行。显示截取出前一部分，判断搜索到的这个文件是不是图片，如果是图片，我们就取出下划线后半部分保存成文件最后的后缀名。如果检查出文件类型非图片，我们就设其类型为默认值。最后，再获得全部一个图片的三部分信息后，将信息存储在预设的变量中，供后面下载模块进行下载。

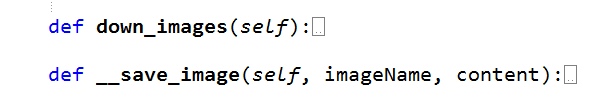
\_\_load\_homePage函数完成的功能很简单，就是将初始页面的内容返回，因此没有额外的参数。

\_\_load\_more函数负责动态加载页面，传入的参数就是当前访问的page的数值，该值的计算由调用它的函数get\_image\_info来完成。该函数和\_\_load\_homepage类似，整个函数中仅有一条语句，也是返回语句，通过调用requests库中的get请求来返回页面的具体内容。但与\_\_load\_homepage不同的是，在get的参数中，填的不是具体的地址，而是一个函数。我们向这个函数中传入具体的页数，然后通过此函数来返回一个对应页数的URL供get方法来请求。这就属于函数式编程，以函数作为参数来传递，优化了流程，让逻辑更加紧密。

\_\_make\_ajax\_url函数的功能就是根据传入的参数，构建相应的URL字符串并返回。在前面的分析中我们已经清楚了动态请求URL的结构，所以该函数完成的就是根据传入的参数和URL的结构，完成相应的拼接操作。

这样，第二部分获取图片信息就编写完了。通过这个模块，我们就可以将网页上存储着的有关图片的信息保存到本地，接下来的就是利用下载下来的信息将位于服务器中的图片信息下载到本地。

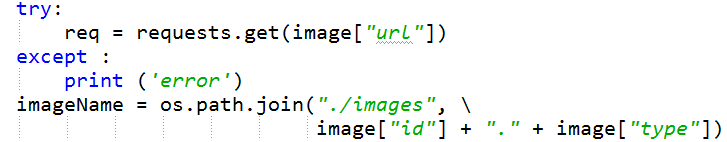
第三部分是下载图片，一共写了两个函数：



下载模块所包括的函数

其中：

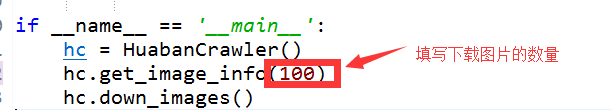
down\_images是下载模块的主函数，负责将从上个模块中获得的图片信息中提取出来，然后按照里面的内容将图片下载下来。在上一步的获取图片信息中，针对每一个图片，我们分别保存了三项信息，分别是该图片的id、通过key值组合成的文件下载地址以及图片的具体类型。所以在这个函数中，我们就将这三种信息从保存信息的列表中取出进行下载。首先利用前一步组合成的图片的具体URL，利用requests库的get方法，从相应的地址中获得对应图片的字节流。这样就获得了文件的具体内容。然后我们就利用图片id和图片的具体类型为相应的图片生成名字，然后再名字前面加入文件的存贮路径，将这个完整的路径保存下载作为后来打开文件操作符的参数。在获得图片的字节流和存储路径后，我们将这些当做参数来调用专门的下载函数来下载图片。



\_\_save\_image函数是负责具体的文件读写，负责将文件下载到本地。传入的参数有一个字节流和一个字符串。该函数利用传入的字符串作为路径参数打开一个文件操作符，然后向这个文件操作符中写入相应的字节流。最后将图片下载到相应的目录中。

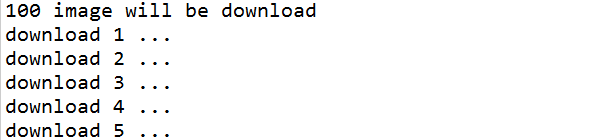


五、爬虫运行



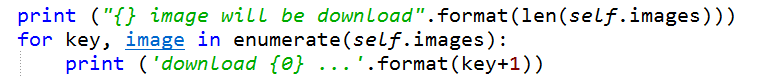


在主函数中填入要下载的图片数量、再确认在类的初始化中确认了搜索的初始网页后，就可以运行爬虫了。

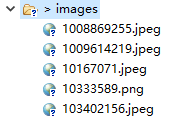
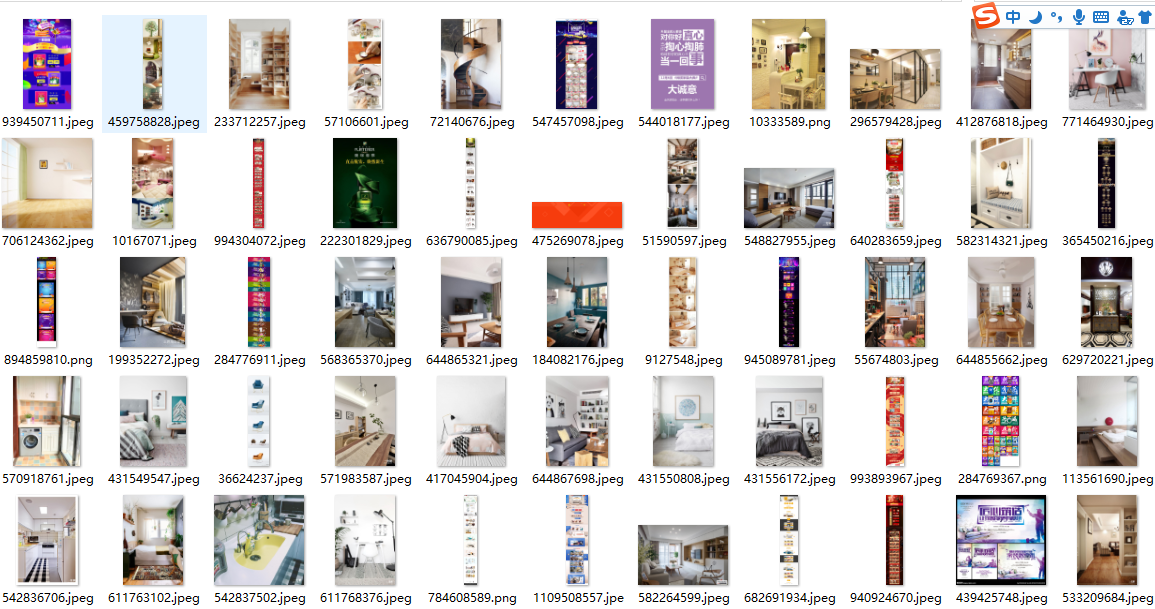


在控制台中的输出信息

在控制台中我们可以看到打印出来的相关信息，这是在down\_images函数中设置的用来检测图片下载情况的语句所运行出来的结果。



然后再项目底部刷新，发现出现新的文件夹，内部保存着下载好的图片。打开资源管理器查看，可以看到已成功下载指定主题的100张图片，与网站比较，内容一致。说明爬虫成功运行。

分别在项目管理目录中和资源管理器中查看的结果

QQ空间说说的抓取（程序已写好，使用selenium模拟浏览器请求）

简单的数据分析（新闻抓取分析，未完成）

参考文献（五号黑体）

1　陈日耀.金属切削原理.北京：机械工业出版社，1985.33~36（五号宋体）

2　Trent E M .Metal Cutting. 2nd Ed. London: BUTTERWORTHS, 1984.120~125

致　　谢（小四号黑体）

感谢方昀老师、李楠高级软件工程师等在毕业设计工作中给予的帮助。

感谢我的学友和朋友对我的关心和帮助（五号宋体）