北京信息科技大学

计算机学院

**程序设计实践（Python）实验参考**

实验名称：

实验类别： 课 程 实 践

适用专业：

适用班级：

授课教师： 杨大利，郝保水

前言

Python语言是一个自由、功能强大的语言，其在人工智能、原型开发等方面有着广泛应用与无与伦比的优势。

本门实践是使用Python完成一个网络爬虫编程实践，通过该门课程的学习，使学生进一步掌握Python语言，为以后学习和工作打下坚实基础。

目录

[前言 2](#_Toc518060778)

[第1章 概述 1](#_Toc518060779)

[1.1 网络爬虫 1](#_Toc518060780)

[1.2 网络爬虫基本思路 1](#_Toc518060781)

[第2章 环境配置 2](#_Toc518060782)

[第3章 网络爬虫系统设计 3](#_Toc518060783)

[3.1 整体框架 3](#_Toc518060784)

[3.2 时序图 3](#_Toc518060785)

[3.3 URL管理器 4](#_Toc518060786)

[3.4 网页解析器 4](#_Toc518060787)

[3.5 调度器 5](#_Toc518060788)

[第4章 网络爬虫各模块实现 6](#_Toc518060789)

[4.1 调度器 6](#_Toc518060790)

[4.2 URL管理器 6](#_Toc518060791)

[4.3 网页下载器 6](#_Toc518060792)

[4.4 网页解析器 8](#_Toc518060793)

[4.5 数据库 10](#_Toc518060794)

[4.6 数据展示 10](#_Toc518060795)

[第5章 网络爬虫系统测试 12](#_Toc518060796)

[第6章 系统总结与展望 13](#_Toc518060797)

[附录 14](#_Toc518060798)

# 概述

## 网络爬虫

所谓“网络爬虫”，指的是搜索网络上的数据并进行整理的计算机程序。网络上存在各种各样的数据，例如网页、文件等，我们可以编写一个计算机程序，定期获得这些数据，然后对这些数据进行整理，从而得到我们感兴趣的数据。网络爬虫也叫做“网络蜘蛛”。

网络爬虫是Google、百度等搜索引擎网站的重要组成部分，他们依靠网络爬虫获得各个网站的数据并进行整理，当用户登录搜索引擎网站并提供关键子后，这些网站返回包含这些关键字的网站，方便用户进一步访问。

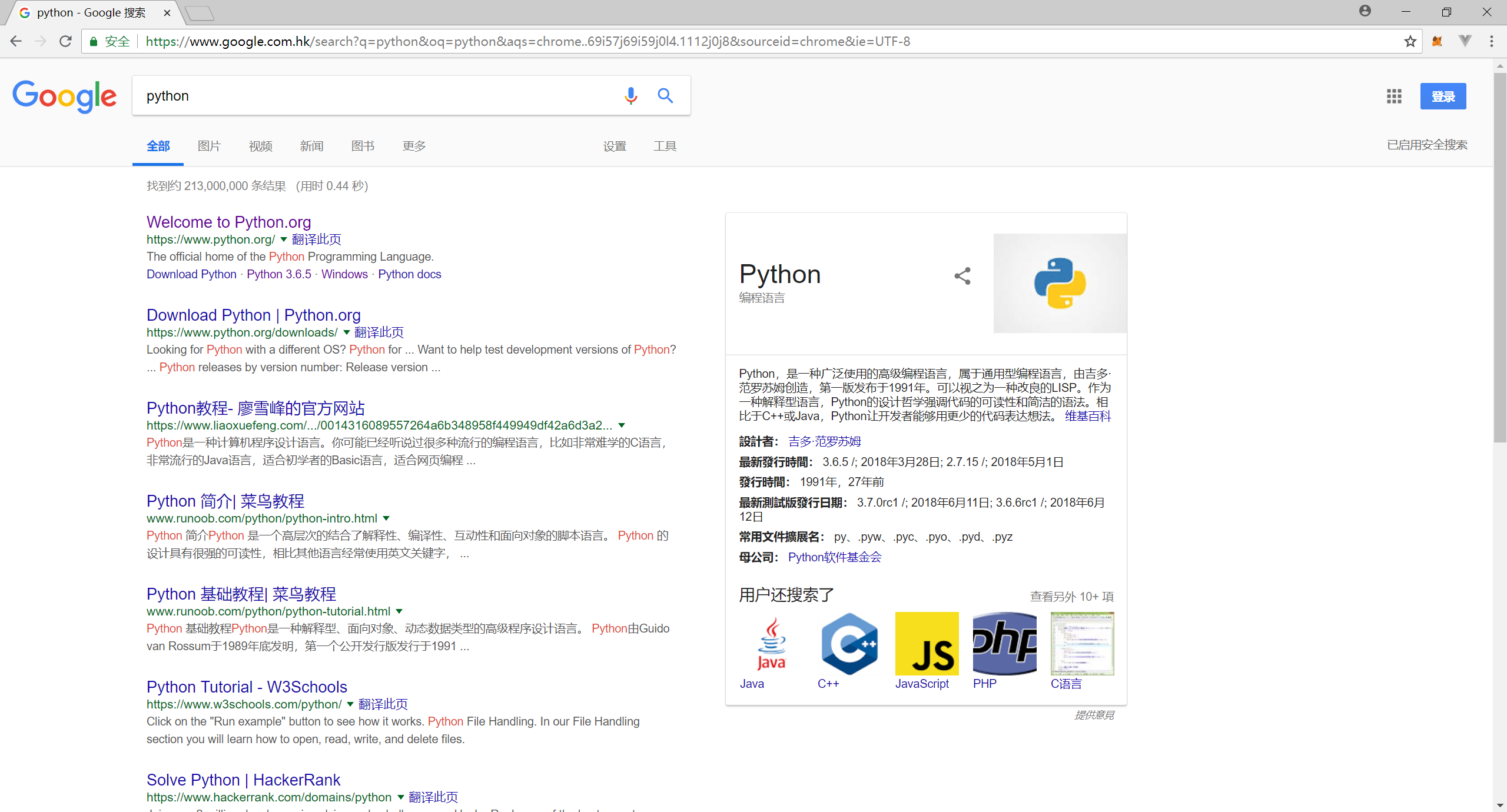


图1.1 使用Google搜索Python

如何快速访问网站、保存并整理海量数据是一件很困难的事情，需要专门研究相关技术。为简单起见，我们这门课程仅仅开发一个简单网络爬虫并对数据进行整理。

## 网络爬虫基本思路

对于简单的网络爬虫而言，思路如下：

1. 找到起始网址，网络爬虫将从此网址开始搜索网络；
2. 下载并分析该网址的有关数据，将整理后的数据和后续搜索网址保存到相关数据结构里；
3. 继续搜索下一个网址，直到所有网址抖索完毕
4. 对搜索到的数据进一步整理，并以适当格式呈现给用户

## Web相关技术简介

### HTML

### CSS

### Java script

# 环境配置

环境配置较为简单。这里以Windows为例进行说明。

1. 安装并配置Python，建议安装Python3；
2. 配置环境变量。

在windows的控制面板\系统和安全\系统中，点击“高级系统设置”，然后点击“环境变量”，如图2.1所示，编辑“Path”变量。

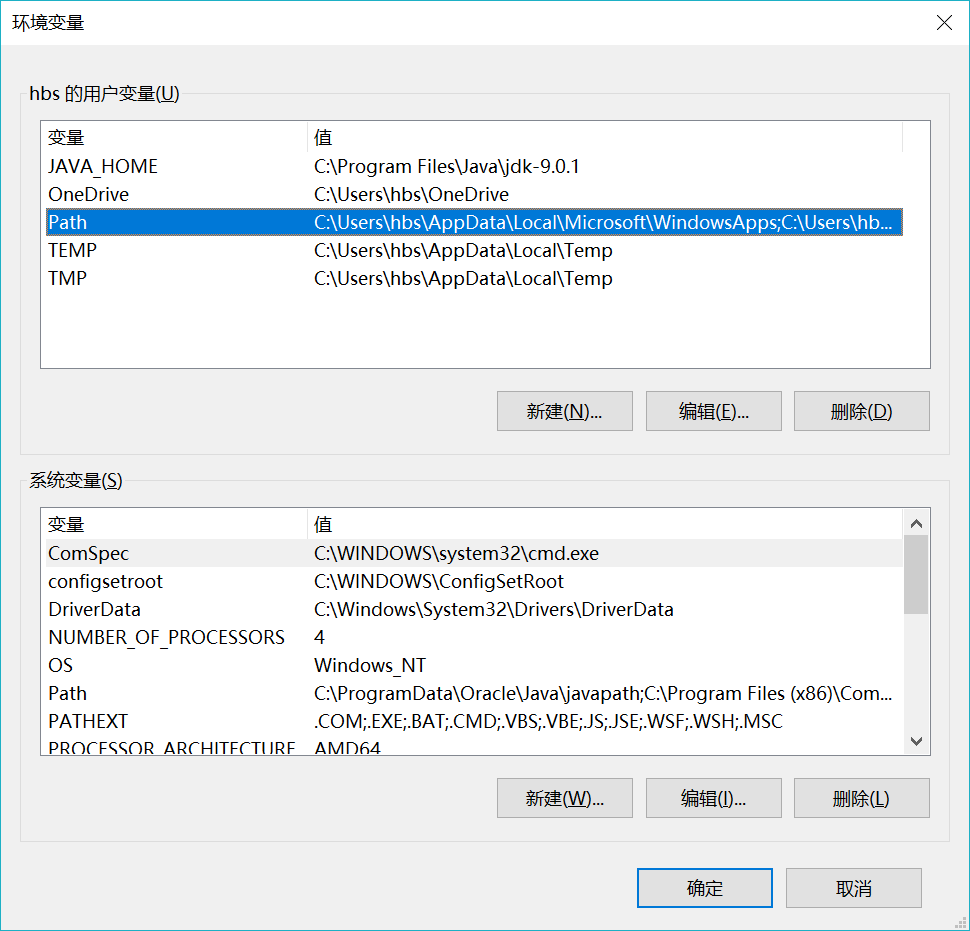


图2.1 环境变量对话框

将Python相关目录添加到Path变量中，如图2.3所示。

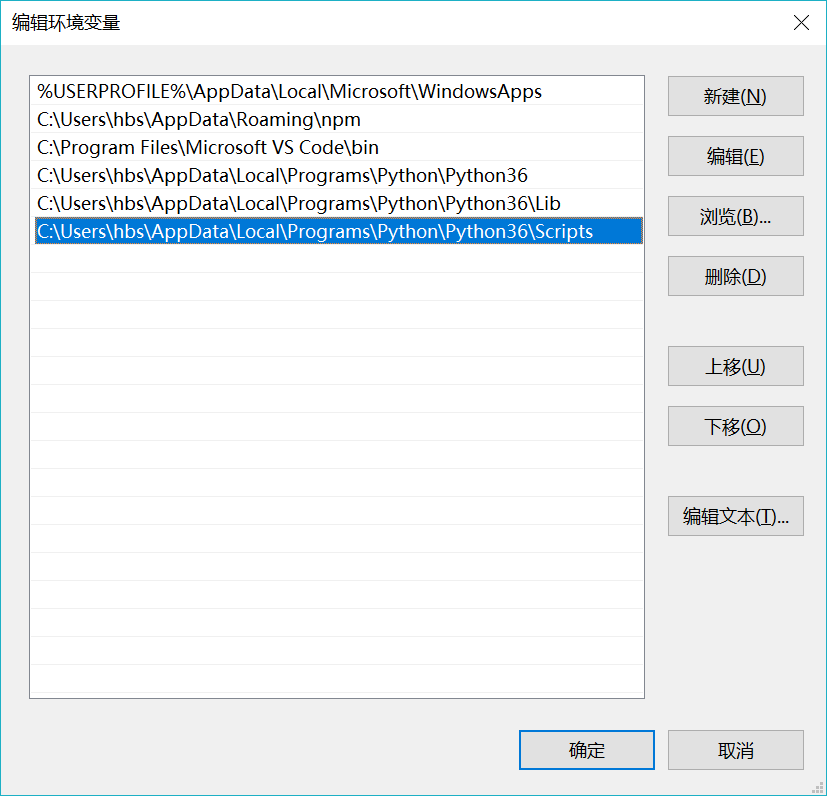


图2.2 将Python相关目录添加到Path变量中

1. 安装其它组件，例如Beautiful Soup 4等。

# 网络爬虫系统设计

## 整体框架

整体框架如图3.1所示。



图3.1 简单网络爬虫整体框架

* 调度器负责整体系统的调度运行。
* URL管理器负责处理将要搜索的网页URL。
* 网页下载器实现对网页的下载。
* 网页解析器对下载的网页内容进行解析，取出我们感兴趣的数据和网页URL，将数据送往数据库，将网页URL送往URL管理器。
* 数据库存放从网页中解析出来的数据。这些数据具有对用户来说存在价值。
* 数据展示部分向用户以适当的格式展示搜索出来的数据，包括列表、图表等，例如Google、百度等以列表形式展示包含用户提供的关键字网页URL等。

## 时序图

简单网络爬虫时序图如图3.2所示。各个部分在此不再赘述，详见课程PPT讲解。



图3.2 简单网络爬虫时序图

## URL管理器

URL管理器负责维护网页URL，这里需要解决几个问题：

1. 如何维护将要下载的网页URL？
2. 如何防止重复和循环下载？如果某个网页已经被下载过，则不需要重新下载。

为解决以上问题，这里设计了两个列表，即未下载URL列表和已下载URL列表。



图3.3 URL管理器数据结构

可以简单地将其理解为一个未下载网页URL队列和一个已下载网页URL集合。

* 未下载网页URL队列满足先进先出，里面存放还没有下载的网页URL列表。每个要搜索的网页URL依次被加入到队尾，取值则从队头取出。
* 已下载网页集合则保存已经下载的网页URL。

系统运行时，首先从未下载网页URL队头取出一个URL，送入到网页下载器，将网页的内容下载下来，然后将该URL添加到已下载URL集合中，即记录该URL为已下载。如果有新来的网页URL，则首先检查是否在已下载URL集合中存在，如果存在，则说明已经下载过，不需要重新下载；否则将其加入到未下载网页URL队列队尾。

## 网页解析器

网页解析器的作用是解析网页内容，取得两方面的数据：

1. 价值数据，即网页中用户感兴趣的数据
2. URL列表，即网页中包含的超链接列表，这些URL是其它网页

### Dom树

网页是用html编写的，浏览器接收到网页后，对其进行解析，表示成文档对象模型（Document Object Model，简称DOM）格式。DOM实际上是以面向对象方式来描述文档的模型，该模型通常表示成一棵树。

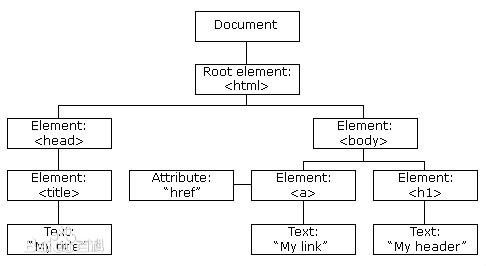


图3.4 DOM树

## 调度器

调度器的作用是在适当的时候调用各模块。下面给出调度器算法：

S1:URL管理器中是否存在未下载URL，如果不存在，转到S4；

S2:从URL管理器中取得一个未下载URL，调用网页下载器下载相应网页内容，同时通知URL管理器将该URL标记为已经下载；

S3:调用解析器分析下载的内容，获取价值数据和该网页包含的URL列表，然后分别将价值数据送往数据库，将URL列表送往URL管理器。然后转到S1；

S4:根据用户用户要求，得到相关数据，并以适当格式返回给用户端。

# 网络爬虫各模块实现

网络爬虫包括调度器、URL管理器、网页下载器、网页解析器、数据库、数据展示等基本。本章介绍这几部分相关知识及其实现。

## 调度器

根据前面的时序图，写出调度器伪代码：

nonvisited=[]#未访问的URL列表

visited=[]#已经访问过的URL列表

datas=[]#价值数据

insert rootURL to nonvisited[]#rootURL为网络爬虫的搜索起始地址

while not empty nonvisited[]

url=nonvisited[].pop#弹出未访问URL列表栈顶值

append url to visited[]#标记该URL已经访问过，不需要重复访问

html=getHtml(url)#取得该url网页内容

data=getData(html)#取得该网页中的价值数据

append data to datas[]

urls[]=getURLs(html)#取得该网页中的超链接

for url in urls[]#处理该网页中的URL，只将未访问URL添加到nonvisited[]

if url not in visited[]

append url to nonvisited[]

print(Data)#将价值数据打印出来

## URL管理器

URL管理器管理未下载URL列表和已下载URL列表。按照前面分析，我们在Python里面，可以将其定义为列表或者集合。

## 网页下载器

网页下载器将网页从服务器上下载下来。

import urllib.request

def getHtml(url):

html = urllib.request.urlopen(url).read()

return html.decode('utf8')

aurl = "http://www.bistu.edu.cn"

html = getHtml(aurl)

print(html)

我们还可以对代码进一步改进，首先获取Http状态码，如果服务器返回200，则说明HTTP请求成功完成，可以获得下载网页的内容；如果返回404，则说明该网址未找到。

import urllib.request

def getHtmlResponse(url):

return urllib.request.urlopen(url)

def getHtmlStatus(response):

status = response.getcode()

return status

def getHtmlContent(response):

html = response.read()

return html

aurl = "http://www.bistu.edu.cn"

response=getHtmlResponse(aurl)

if getHtmlStatus(response) ==200 :

html=getHtmlContent(response)

print(html)

else:

print("error")

当然，也可以使用其它库，例如requests。首先需要安装该库。

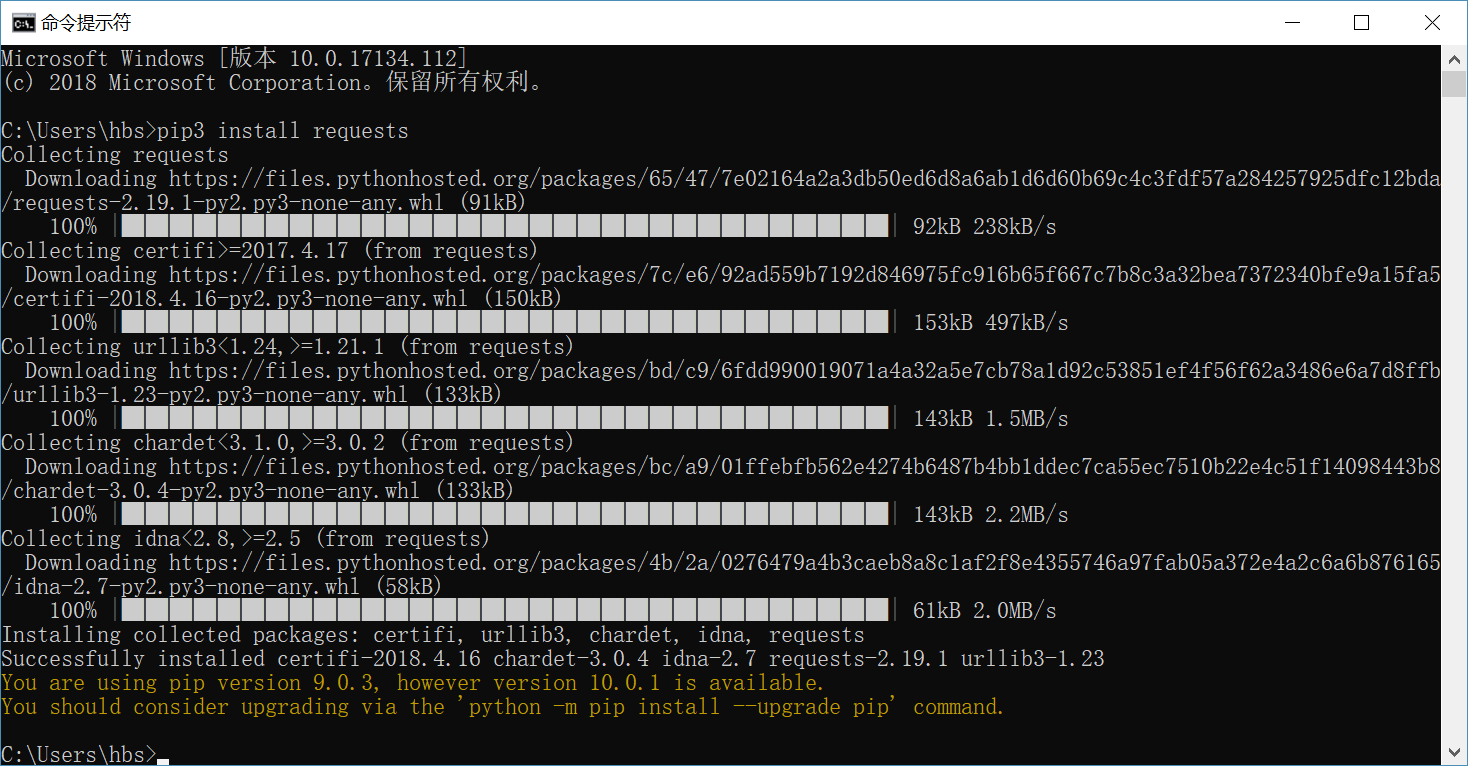


图4.1 安装requests

然后编写代码：

import requests

r=requests.get('http://www.bistu.edu.cn')

if r.status\_code==200:

print(r.text)

可以看到，代码更加简单。

## 网页解析器

当把网页下载下来后，我们需要解析其中的内容，获得自己的感兴趣的数据，例如所有的超链接等。解析器有多种写法，下面简单分析之。

### 搜索字符串

下面代码是查找文档中所有的超链接。

import requests

r=requests.get('http://www.bistu.edu.cn')

if r.status\_code==200:

r.encoding = 'utf-8'

html=r.text

left=(html.find('<a'))

while left>=0:

right=html.find('</a>',left)

print(html[left:right+4])

left=(html.find('<a',right))

显然，使用字符串搜索的方法较为繁琐，对于简单文档可以使用，但是对于复杂文档就显得力不从心了。例如如何在这些超链接中查找以xyz.html结尾的呢？

### 正则表达式

可以使用正则表达式来搜索网页中与给定模式相匹配的内容。例如查找href中形如[http://news.\*.html](http://news.*.html)的超链接：

import requests

import re

pattern = re.compile(r'<a href="http://news.(.\*?).html(.\*?)</a>')

r=requests.get('http://www.bistu.edu.cn')

if r.status\_code==200:

r.encoding = 'utf-8'

html=r.text

matchs = []

for url in pattern.findall(html):

matchs.append(url)

print(matchs)

### Beautiful Soap

有的时候需要对文档进行进一步分析，例如需要对文档结构进行分析，这个时候使用前面的方法就不够了。可以接着第三方库来实现，这里介绍beautiful soap来实行。

#### 安装

BeautifulSoap中文网址见：<https://www.crummy.com/software/BeautifulSoup/bs4/doc/index.zh.html>

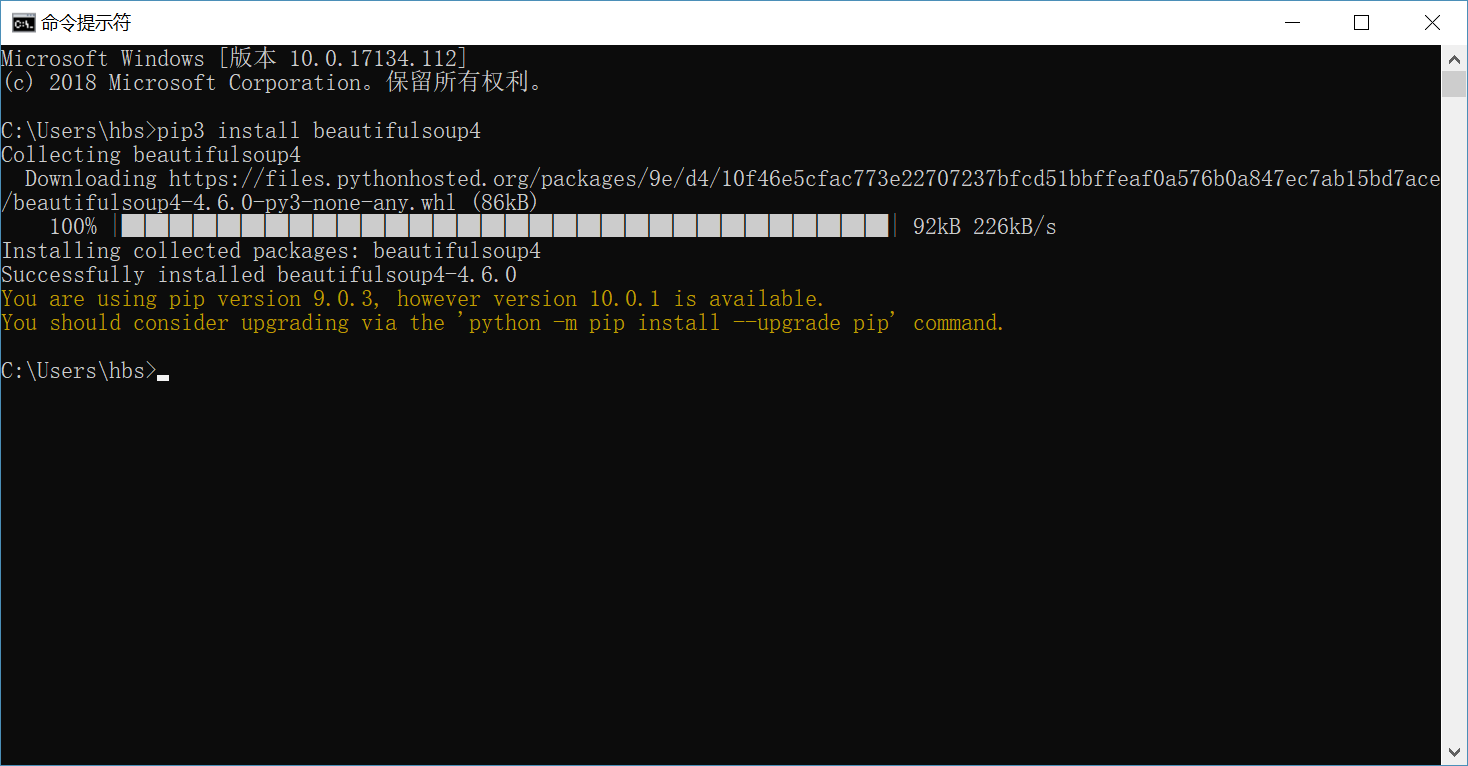


图4.2 安装BeautifulSoap4

#### 使用

这里列出常用的Beautiful Soap方法，具体使用请详见文档。

find\_all

find

#### 查找所有超链接：

import requests

from bs4 import BeautifulSoup

r=requests.get('http://www.bistu.edu.cn')

if r.status\_code==200:

r.encoding = 'utf-8'

html=r.text

soup = BeautifulSoup(html,"html.parser")

print(soup.prettify())

#从文档中找到所有<a>标签的链接:

for link in soup.find\_all('a'):

print(link.get('href'))#print(link['href'])#节点属性

print(link.name)#节点名字

print(link.get\_text())#节点文字

从上可以看出，使用BeautifulSoap方便得多。

## 数据库

略

## 数据展示

根据用户需求，展示价值数据。

# 网络爬虫系统测试

# 系统总结与展望

附录

简单搜索引擎源代码

import requests

from bs4 import BeautifulSoup

nonvisited=[]#未访问的URL列表

visited=[]#已经访问过的URL列表

datas=[]#价值数据

#获取url网页内容，注意这里有小缺陷，只能下载以http://为开头的网页内容

def getHtmlContent(url):

print('正在访问'+url)

if len(url) <7:return ''

if url[:7]=='http://':

r=requests.get(url)

if r.status\_code==200:

r.encoding = 'utf-8'

html=r.text

return html

return ''

#解析网页中的超链接

def parseURLs(html):

global nonvisited

global visited

soup = BeautifulSoup(html,"html.parser")

#从文档中找到所有<a>标签的链接:

for link in soup.find\_all('a'):

print(link.get('href'))#print(link['href'])#节点属性

print(link.name)#节点名字

print(link.get\_text())#节点文字

if link not in visited:

nonvisited.append(link.get('href'))

#解析网页中的数据，这里没有实现

def parseData(html):

soup = BeautifulSoup(html,"html.parser")

#初始化

def init(root):

global nonvisited

global visited

global datas

nonvisited=[]

visited=[]

datas=[]

nonvisited.append(root)

init('http://www.bistu.edu.cn')

while len(nonvisited)>0:

url=nonvisited.pop(0)

visited.append(url)

html=getHtmlContent(url)

parseURLs(html)

parseData(html)