**爬虫03 解密吴氏私厨**

项目：解密吴氏私厨

我们继续说回到吴氏私厨的事。我在业余时，有一个爱好是烹饪美味的食物。几乎每个周末、节日，我都会邀请朋友、同事来我的家中做客，设宴款待。在快节奏的都市，这是一种奢侈而美好的享受。

我喜欢尝试新的东西，去做新的菜式，因为新鲜感能让朋友们对赴宴“吴氏私厨”总是心怀期待。

有朋友会好奇：吴枫，为什么你可以会这么多种不同的菜式？这里面可有什么不为人知的秘密？我笑而不答，说你回头看我写的爬虫关卡就知道。

现在，我要将秘密揭晓。

答案就是我懂编程。在我看来，下厨和Python非常相像。学Python你只需要懂最基础的语法：列表字典、判断循环……然后合理地调用模块，就已经能做出很多有趣的事，代码跳跃在屏幕上，自有千般变化。

下厨你只需要懂最基础的操作手法：蒸炒炖焖煮、烧炸煎煲卤……然后去找合适的菜谱，大江南北，少有你烹不出的风味。

显然我懂厨房的基础手法，我只需要找合适的菜谱。

我写过一段Python代码，它能在每个周五爬取最新出的热门菜谱清单，发送到我的邮箱。内含：菜名、原材料、详细烹饪流程的URL。

我会选择里面看上去美味又没尝试过的，作为周末款待宾客的菜单。

下面，我要带你复现这个代码的编写过程。

项目目标：我们要去爬取热门菜谱清单，内含：菜名、原材料、详细烹饪流程的URL。而定时爬取和发送邮件，我会在第10关讲给你方案。

在这个项目里，我会带你体验BeautifulSoup库的实战应用，让你感受一个爬虫项目是如何一步步实现的。同时，也会有一些爬虫项目实战的经验分享。

分析过程

在这个项目里，我们选取的网站是“下厨房”。它有一个固定栏目，叫做“本周最受欢迎”，收集了当周最招人喜欢的菜谱。地址如下：

<http://www.xiachufang.com/explore/>

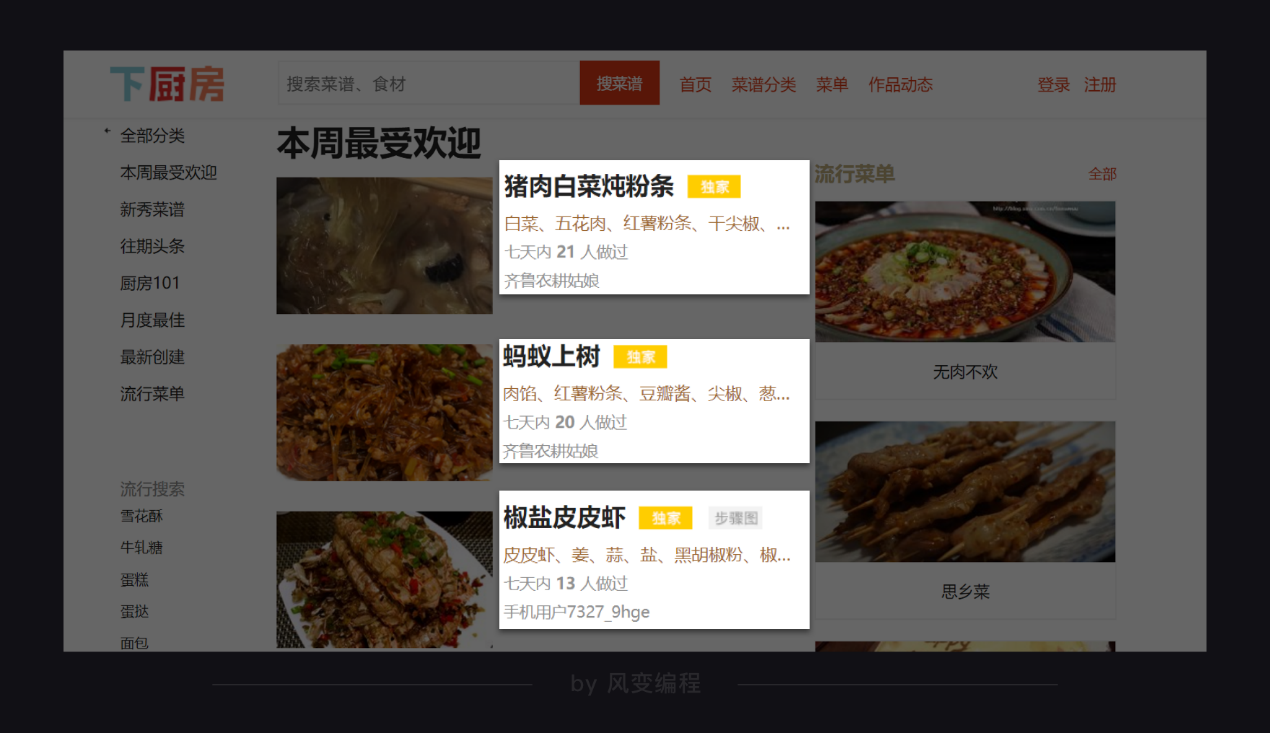
在进行爬取之前，我们先去看看它的robots协议。网址在此：

<http://www.xiachufang.com/robots.txt>

因为这个页面挺长的，所以我不再为你放截图。阅读这个robots协议，你会发现：我们要爬取的/explore/不在禁止爬取的列表内，但如果你要爬取/recipe/服务器就会不欢迎。在网页里，recipe是每一道菜的详情页面，记录了这道菜的做法。

如果你真要爬/recipe/里的信息，也能爬取到。只是人家都这样说了，我们就不要去爬它。

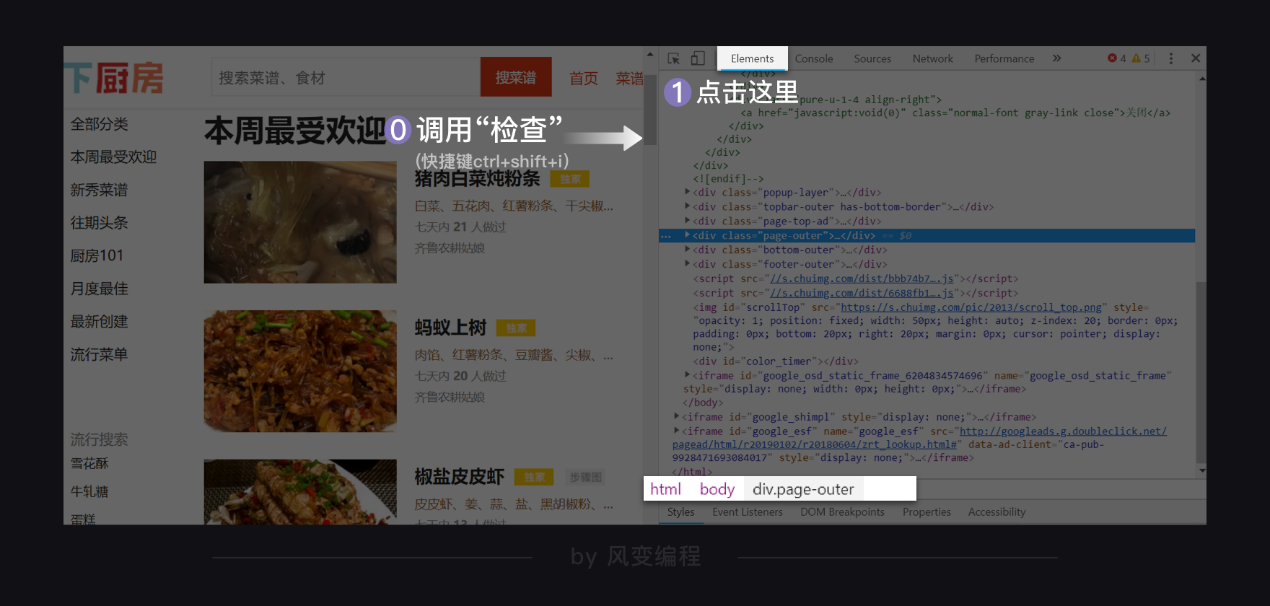
下面，我们来看看这个页面：



我们计划拿到的信息，就是这个页面上的：菜名、所需材料、和菜名所对应的详情页URL。

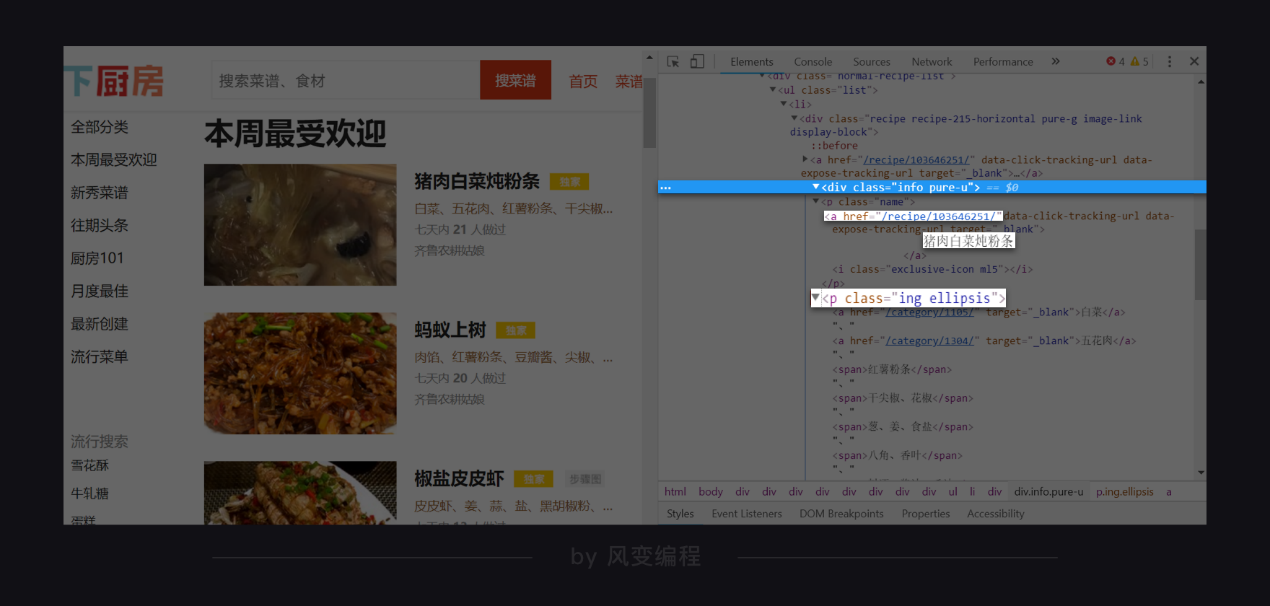
获取数据是容易的。我们使用requests.get()就能实现。解析数据是容易的，我们用BeautifulSoup来实现。

真正需要我们关注的，是如何把有效信息，从这个网址当中给提取出来。打开检查工具，我们在Elements里查看这个网页，是怎样的结构。



刚刚打开Elements，它会默认展开body，其余都关闭。我的鼠标悬停在<div class="page-outer">…<div> == $0上，所以你看到下方限制的路径，就是：html > body > div.page-outer。其中.所代表的正是class。

点击开发者工具左上角的小箭头，然后选中一个菜名，如我选的就是“猪肉炖粉条”，那么Elements那边就会自动标记出对应的代码。



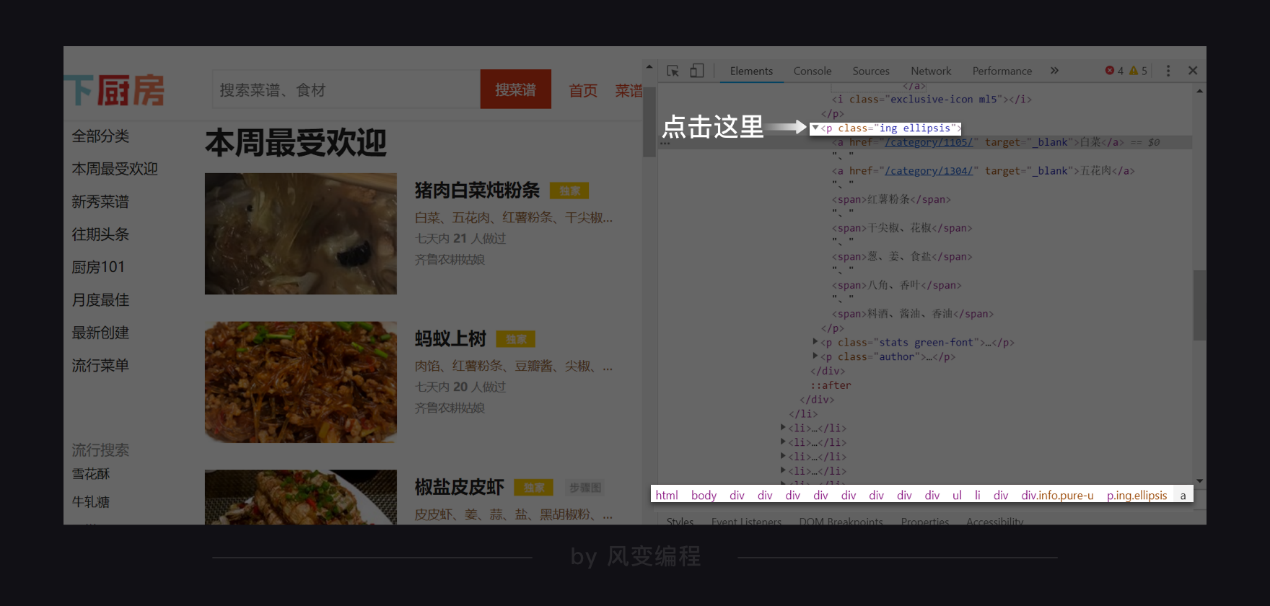
如此，我们就定位到了菜名的所在位置，<a>标签内的文本，甚至还顺带找到了详情页URL的所在位置。如上图，<a>标签里有属性href，其值是/recipe/103646251/。点击它，你会跳转到这道菜的详情页：

<http://www.xiachufang.com/recipe/103646251/>

所以到时候，我们可以去提取<a>标签。接着，先用text拿到它的文本，再使用[href]获取到半截URL，和[http://www.xiachufang.com)](http://www.xiachufang.com/" \t "_blank)做拼接即可。

步骤可以说是非常清晰了！用中学老师的话讲，这叫给未来的解析与提取打下坚实基础。

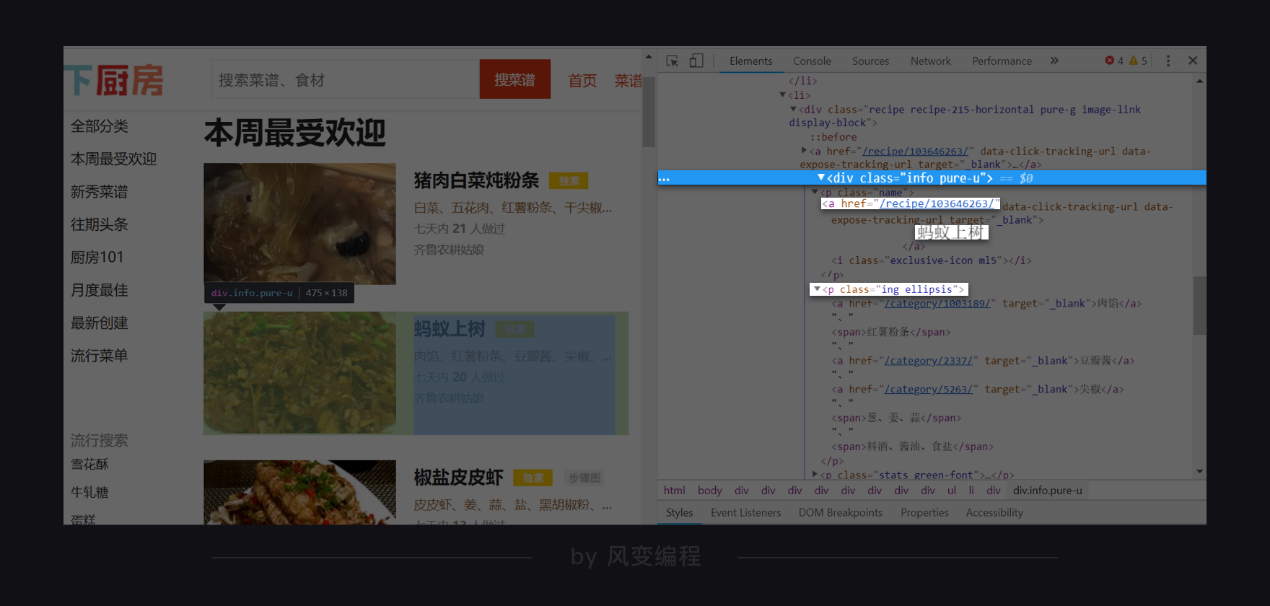
那趁热打铁。我们再去找找食材在哪里。和查找菜名一样的操作，去点击小箭头，去挑选一个食材。



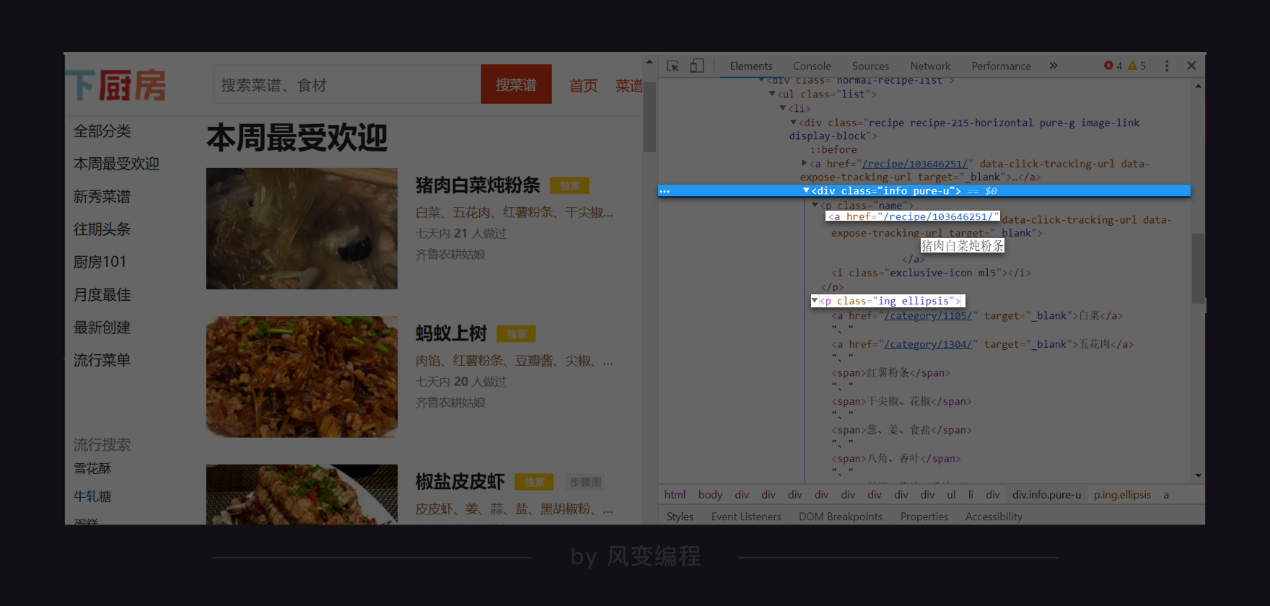
这说找一个，竟是全给找到了。猪肉白菜炖粉条，所需食材是：白菜、五花肉、红薯粉条、干尖椒、花椒、葱、姜、食盐、八角、香叶、料酒、酱油、香油……

它们有的是<a>标签里的纯文本，有的是<span>标签里的纯文本。它们的共同父级标签（相对于子标签，上级标签的意思，父标签包含子标签）是<p class="ing ellipsis">。

根据菜名的路径、URL的路径、食材的路径，我们可以这三者的最小共同父级标签，是：<div class="info pure-u">。



现在，我们就找到了所有待爬取的数据藏身何处。现在，去校验一下，其它菜名、URL、食材是不是也在同样位置。如下，蚂蚁上树：



菜名是<a>标签内的文本，URL是<a>标签里属性href的值，食材藏身于<p class="ing ellipsis">。最后，它们三者的最小共同父级标签，是<div class="info pure-u">。

这说明，我们找的规律没错。那么基于此，我们可以产生两种写爬虫的思路：

思路一：我们先去爬取所有的最小父级标签<div class="info pure-u">，然后针对每一个父级标签，想办法提取里面的菜名、URL、食材。

思路二：我们分别提取所有的菜名、所有的URL、所有的食材。然后让菜名、URL、食材给一一对应起来（这并不复杂，第0个菜名，对应第0个URL，对应第0组食材，按顺序走即可）。

这两种思路，理论上来说都能够实现我们的目标。那么，我们应该选哪种？

小孩子才做选择，大人们则是全都要。下面，我们会详细介绍思路一，而把思路二留给你做练习。

在最后，提取到了数据我们要存储。但文件存储我们要到第6关才学习。所以，我们就先把数据存到列表里：每一组菜名、URL、食材是一个小列表，小列表组成一个大列表。如下：

[[菜A,URL\_A,食材A],[菜B,URL\_B,食材B],[菜C,URL\_C,食材C]]

到这，我们可以去尝试写代码。

代码实现（一）

我们先使用思路一来写代码，即：先去爬取所有的最小父级标签<div class="info pure-u">，然后针对每一个父级标签，想办法提取里面的菜名、URL、食材。

获取与解析

我们选取的URL是[http://www.xiachufang.com/explore/](http://www.xiachufang.com/explore/" \t "_blank)，我们用requests.get()来获取数据。

接着，使用BeautiSoup来解析，这两步都不算是难事。所以我把它交给你当作开胃小菜。下面，请完成获取数据和解析数据两步，并将解析的结果打印出来。

import requests

# 引用requests库

from bs4 import BeautifulSoup

# 引用BeautifulSoup库

res\_foods = requests.get('http://www.xiachufang.com/explore/')

# 获取数据

bs\_foods = BeautifulSoup(res\_foods.text,'html.parser')

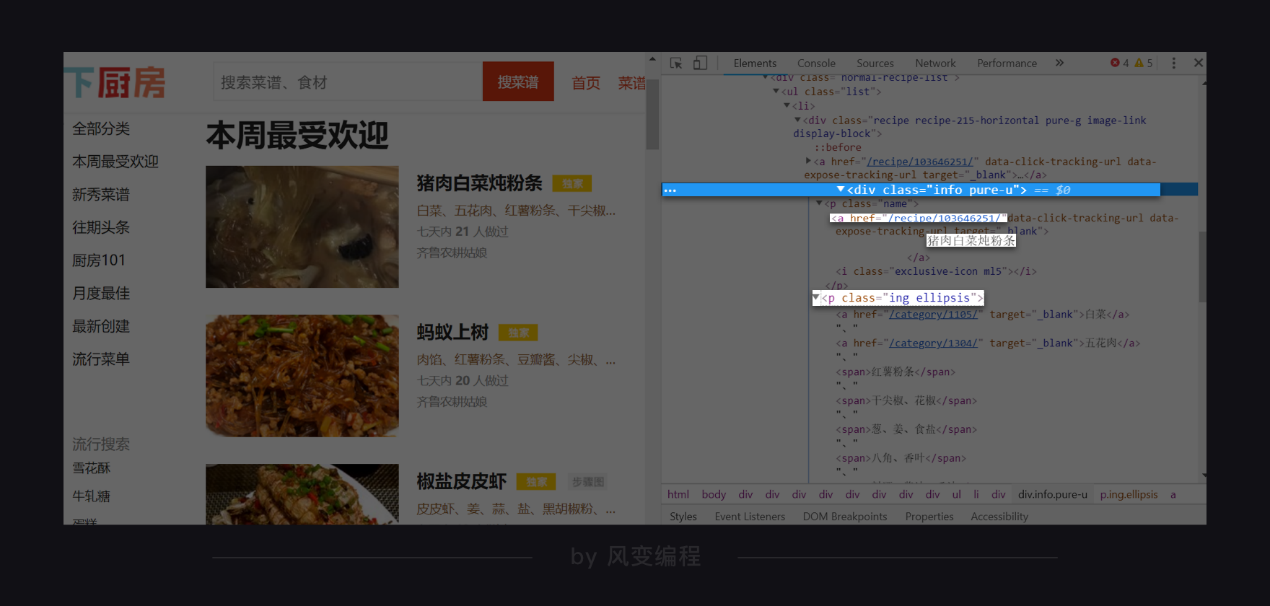
# 解析数据

print(bs\_foods)

# 打印解析结果

提取最小父级标签

我们来看这个父级标签：



它的标签是<div>，有一个class属性，其值是info pure-u。我们可以使用find\_all()语法，来找到它们。



现在，请尝试续写下方代码：使用find\_all()语法查找最小父级标签，并把查找的结果打印出来。

import requests

# 引用requests库

from bs4 import BeautifulSoup

# 引用BeautifulSoup库

res\_foods = requests.get('http://www.xiachufang.com/explore/')

# 获取数据

bs\_foods = BeautifulSoup(res\_foods.text,'html.parser')

# 解析数据

list\_foods = bs\_foods.find\_all('div',class\_='info pure-u')

# 查找最小父级标签

print(list\_foods)

# 打印最小父级标签

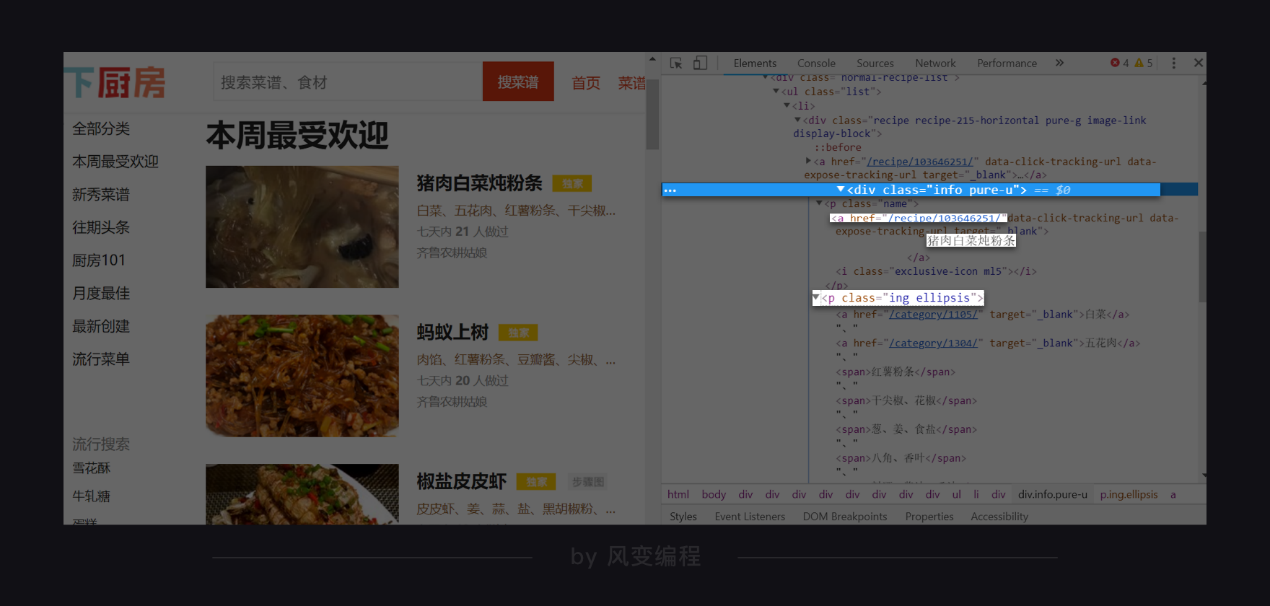
非常幸运的是，这里一次就打印出了正确的东西。在实践操作当中，其实常常会因为标签选取不当，或者网页本身的编写没做好板块区分，你可能会多打印出一些奇怪的东西。

当遇到这种糟糕的情况，一般有两种处理方案：数量太多而无规律，我们会换个标签提取；数量不多而有规律，我们会对提取的结果进行筛选——只要列表中的若干个元素就好。

下面，我们可以进行下一步。针对查找结果中的每一个元素，再次查找位于里面的菜名、URL、食材。

一组菜名、URL、食材

我们不先急于提取出所有的菜名、URL和食材。我们先尝试提取一组，等成功了，再去写循环提取所有。我们来看图：

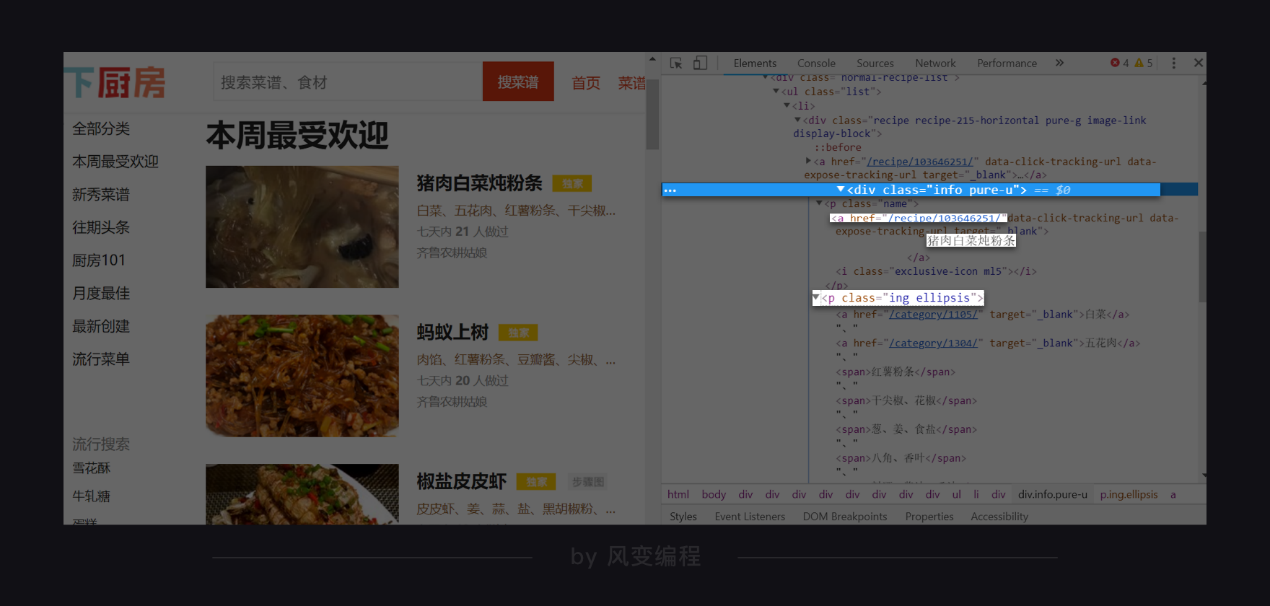


如何拿到URL和菜名？答案显而易见：我们可以查找父级标签中的第0个标签，里面就会有我们想要的信息。

只查找第0个标签，应该用什么语句？3、2、1，回答我。

用find()。我们对父级标签，使用find()方法来查找标签就对了，所用的参数就是<a>标签本身。

当拿到<a>标签之后呢，我们应该如何提取纯文本，以及某个属性的值？3、2、1，回答我。



现在，我们可以提取出第0个父级标签中的第0个<a>标签，并输出菜名和URL。

提示一：记得要提取的是第0个父级标签，而不是整个父级标签列表；提示二：提取出的菜名可能前后都会有很多空格和换行，你可以使用类似列表切片的方法，把多余的内容裁剪掉。提示三：提取出的URL需要和[http://www.xiachufang.com](http://www.xiachufang.com/)做拼接。

import requests

# 引用requests库

from bs4 import BeautifulSoup

# 引用BeautifulSoup库

res\_foods = requests.get('http://www.xiachufang.com/explore/')

# 获取数据

bs\_foods = BeautifulSoup(res\_foods.text,'html.parser')

# 解析数据

list\_foods = bs\_foods.find\_all('div',class\_='info pure-u')

# 查找最小父级标签

tag\_a = list\_foods[0].find('a')

# 提取第0个父级标签中的<a>标签

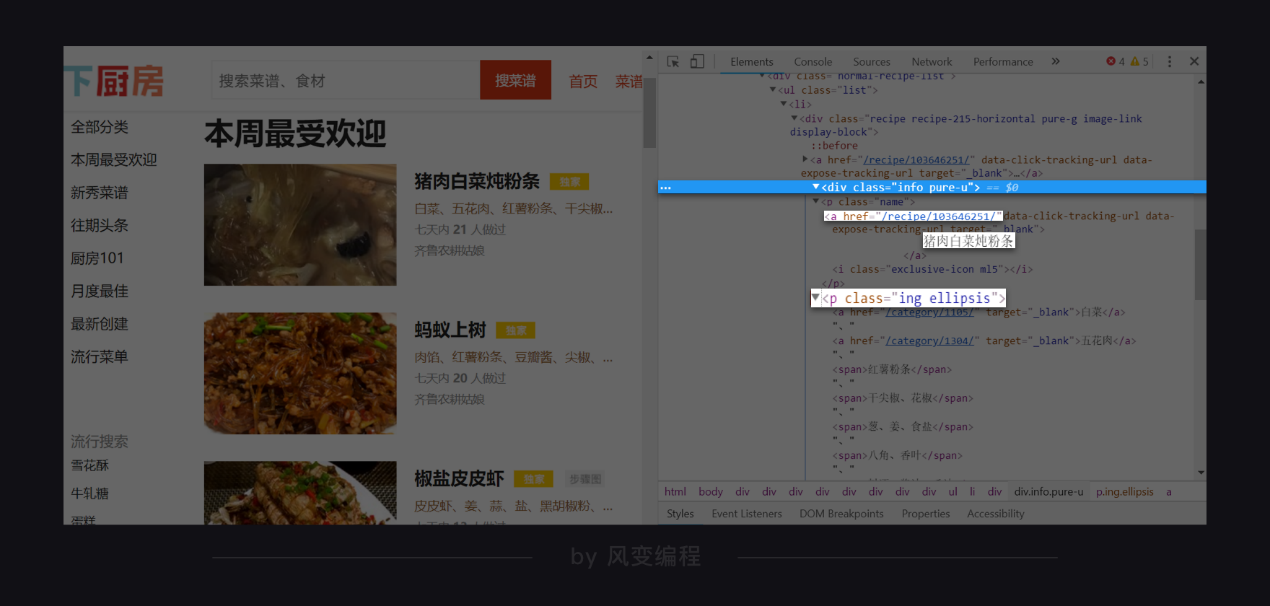
print(tag\_a.text[17:-13])

# 输出菜名，使用[17:-13]切掉了多余的信息

print('http://www.xiachufang.com'+tag\_a['href'])

# 输出URL

现在我们来看，如何提取食材。



你可能会想：我们去写一个find()去寻找<p>标签，再去写一个find\_all()去寻找它里面的所有<a>标签，然后写一个循环，使用text去提取每一个<a>标签里的纯文本信息。

紧接着，用同样的手法，拿到所有<span>标签里的纯文本信息。最后，把这些全都拼接起来。

天呐，这实在是麻烦坏了。

下面我要告诉你一个好消息：你完全不需要这样做，你只需要查找<p>标签，然后使用text提取<p>标签里的纯文本信息，就可以了！

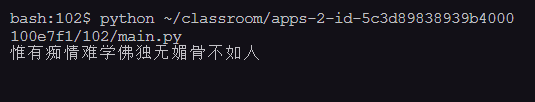
不信，你可以做这样一个小测试，直接点击运行即可：

from bs4 import BeautifulSoup

bs = BeautifulSoup('<p><a>惟有痴情难学佛</a>独无媚骨不如人</p>','html.parser')

tag = bs.find('p')

print(tag.text)



你会发现输出的结果是“惟有痴情难学佛独无媚骨不如人”。当我们在用text获取纯文本时，获取的是该标签内的所有纯文本信息，不论是直接在这个标签内，还是在它的子标签内。

需要强调的一点是，text可以这样做，但如果是要提取属性的值，是不可以的。父标签只能提取它自身的属性值，不能提取子标签的属性值。如下，就会报错：

from bs4 import BeautifulSoup

bs = BeautifulSoup('<p><a href=\'https://www.pypypy.cn\'></a></p>','html.parser')

# 此处多出来的\，是转义字符。

tag = bs.find('p')

print(tag['href'])

# 这样会报错，因为<p>标签没有属性href，href属于<a>标签

有了这个知识，请你在之前代码的基础上，写出提取食材的代码，并打印出来。提示：只是用p标签做参数是不够的，因为这里不止一个p标签存在。你还要加class\_参数。

其中的**name = tag\_a.text[17:-13]**这个知识点实际上就是字符串的切片，-13就是倒数第13个位置开始截取；从左数是0，1，2开始，从右数是-1-2-3数，这是我们课程为了方便你的阅读，通过自己数着帮你把空格切除了，当然你也可以用replace(" ","")来去空格，或者strip函数来的

import requests

# 引用requests库

from bs4 import BeautifulSoup

# 引用BeautifulSoup库

res\_foods = requests.get('http://www.xiachufang.com/explore/')

# 获取数据

bs\_foods = BeautifulSoup(res\_foods.text,'html.parser')

# 解析数据

list\_foods = bs\_foods.find\_all('div',class\_='info pure-u')

# 查找最小父级标签

tag\_a = list\_foods[0].find('a')

# 提取第0个父级标签中的<a>标签

name = tag\_a.text[17:-13]

# 菜名，使用[17:-13]切掉了多余的信息

URL = 'http://www.xiachufang.com'+tag\_a['href']

# 获取URL

tag\_p = list\_foods[0].find('p',class\_='ing ellipsis')

# 提取第0个父级标签中的<p>标签

ingredients = tag\_p.text[1:-1]

# 食材，使用[1:-1]切掉了多余的信息

print(ingredients)

# 打印食材

写循环，存列表

要求：写一个循环，提取当前页面的所有菜名、URL、食材，并将它存入列表。其中每一组菜名、URL、食材是一个小列表，小列表组成一个大列表。如下：

**[[菜A,URL\_A,食材A],[菜B,URL\_B,食材B],[菜C,URL\_C,食材C]]**

import requests

# 引用requests库

from bs4 import BeautifulSoup

# 引用BeautifulSoup库

res\_foods = requests.get('http://www.xiachufang.com/explore/')

# 获取数据

bs\_foods = BeautifulSoup(res\_foods.text,'html.parser')

# 解析数据

list\_foods = bs\_foods.find\_all('div',class\_='info pure-u')

# 查找最小父级标签

list\_all = []

# 创建一个空列表，用于存储信息

for food in list\_foods:

tag\_a = food.find('a')

# 提取第0个父级标签中的<a>标签

name = tag\_a.text[17:-13]

# 菜名，使用[17:-13]切掉了多余的信息

URL = 'http://www.xiachufang.com'+tag\_a['href']

# 获取URL

tag\_p = food.find('p',class\_='ing ellipsis')

# 提取第0个父级标签中的<p>标签

ingredients = tag\_p.text[1:-1]

# 食材，使用[1:-1]切掉了多余的信息

list\_all.append([name,URL,ingredients])

# 将菜名、URL、食材，封装为列表，添加进list\_all

print(list\_all)

# 打印

至此，一个项目就算从头到尾结束。

代码实现（二）

就像我们之前所说，这个项目还存在着另一个解决思路：我们分别提取所有的菜名、所有的URL、所有的食材。然后让菜名、URL、食材给一一对应起来。

对于这个实操，我不再一步一步为你讲解，而是换一种方式。我会为你简单描述大致思路，由你来自行写代码，结束之后再来和标准答案比照。

首先，获取数据，解析数据，略过。

去查找所有，包含菜名和URL的<p>标签。此处<p>标签是<a>标签的父标签。

为什么不直接选<a>标签？还记得我们怎么说的吗？在实践操作当中，其实常常会因为标签选取不当，或者网页本身的编写没做好板块区分，你可能会多打印出一些奇怪的东西。

当遇到这种糟糕的情况，一般有两种处理方案：数量太多而无规律，我们会换个标签提取；数量不多而有规律，我们会对提取的结果进行筛选——只要列表中的若干个元素就好。

这里如果是直接提取<a>标签，你就会遇到这种情况。如果你愿意，也可以试试看。

去查找所有，包含食材的<p>标签。

创建一个空列表，启动循环，循环长度等于<p>标签的总数——你可以借助range(len())语法。

在每一次的循环里，去提取一份菜名、URL、食材。拼接为小列表，小列表拼接成大列表。输出打印。

import requests

# 引用requests库

from bs4 import BeautifulSoup

# 引用BeautifulSoup库

res\_foods = requests.get('http://www.xiachufang.com/explore/')

# 获取数据

bs\_foods = BeautifulSoup(res\_foods.text,'html.parser')

# 解析数据

tag\_name = bs\_foods.find\_all('p',class\_='name')

# 查找包含菜名和URL的<p>标签

tag\_ingredients = bs\_foods.find\_all('p',class\_='ing ellipsis')

# 查找包含食材的<p>标签

list\_all = []

# 创建一个空列表，用于存储信息

for x in range(len(tag\_name)):

# 启动一个循环，次数等于菜名的数量

list\_food = [tag\_name[x].text[18:-14],tag\_name[x].find('a')['href'],tag\_ingredients[x].text[1:-1]]

# 提取信息，封装为列表。注意此处[18:-14]切片和之前不同，是因为此处使用的是<p>标签，而之前是<a>

list\_all.append(list\_food)

# 将信息添加进list\_all

print(list\_all)

# 打印

一个项目，两种解法。恭喜你！全都掌握。当菜谱在手，大江南北，便少有你烹不出的风味。下面是我的一份家常拙作，期待你的精彩作品。

复习总结

严格来说，我们这一关其实没有新的知识进入，它是一个比较纯粹的项目关卡，汇总代码如下：

import requests

# 引用requests库

from bs4 import BeautifulSoup

# 引用BeautifulSoup库

res\_foods = requests.get('http://www.xiachufang.com/explore/')

# 获取数据

bs\_foods = BeautifulSoup(res\_foods.text,'html.parser')

# 解析数据

tag\_name = bs\_foods.find\_all('p',class\_='name')

# 查找包含菜名和URL的<p>标签

tag\_ingredients = bs\_foods.find\_all('p',class\_='ing ellipsis')

# 查找包含食材的<p>标签

list\_all = []

# 创建一个空列表，用于存储信息

for x in range(len(tag\_name)):

# 启动一个循环，次数等于菜名的数量

list\_food = [tag\_name[x].text[18:-14],tag\_name[x].find('a')['href'],tag\_ingredients[x].text[1:-1]]

# 提取信息，封装为列表。注意此处[18:-14]切片和之前不同，是因为此处使用的是<p>标签，而之前是<a>

list\_all.append(list\_food)

# 将信息添加进list\_all

print(list\_all)

# 打印

# 以下是另外一种解法

list\_foods = bs\_foods.find\_all('div',class\_='info pure-u')

# 查找最小父级标签

list\_all = []

# 创建一个空列表，用于存储信息

for food in list\_foods:

tag\_a = food.find('a')

# 提取第0个父级标签中的<a>标签

name = tag\_a.text[17:-13]

# 菜名，使用[17:-13]切掉了多余的信息

URL = 'http://www.xiachufang.com'+tag\_a['href']

# 获取URL

tag\_p = food.find('p',class\_='ing ellipsis')

# 提取第0个父级标签中的<p>标签

ingredients = tag\_p.text[1:-1]

# 食材，使用[1:-1]切掉了多余的信息

list\_all.append([name,URL,ingredients])

# 将菜名、URL、食材，封装为列表，添加进list\_all

print(list\_all)

# 打印

这个项目里有许多东西，值得我们回过头来看，记录到小本本上，下面是我的总结：

确认目标-分析过程-代码实现，是我们做每一个项目的必经之路。未来在此基础上，还会有许多演化，但基础都是这些。

将想要的数据分别提取，再做组合是一种不错的思路。但是，如果数据的数量对不上，就会让事情比较棘手。比如，在我们的案例里，如果一个菜有多个做法，其数量也没规律，那么菜名和URL的数量就会对不上。

寻找最小共同父级标签是一种很常见的提取数据思路，它能有效规避这个问题。但有时候，可能需要你反复操作，提取数据。

所以在实际项目实操中，需要根据情况，灵活选择，灵活组合。我们本关卡所做的项目，只是刚刚好两种方式都可以爬取。

text获取到的是该标签内的纯文本信息，即便是在它的子标签内，也能拿得到。但提取属性的值，只能提取该标签本身的。

from bs4 import BeautifulSoup

bs = BeautifulSoup('<p><a>惟有痴情难学佛</a>独无媚骨不如人</p>','html.parser')

tag = bs.find('p')

print(tag.text)

在爬虫实践当中，其实常常会因为标签选取不当，或者网页本身的编写没做好板块区分，你可能会多提取到出一些奇怪的东西。

当遇到这种糟糕的情况，一般有两种处理方案：数量太多而无规律，我们会换个标签提取；数量不多而有规律，我们会对提取的结果进行筛选——只要列表中的若干个元素就好。

以上，就是我要分享的全部信息。

爬虫，它是一项需要许多实操，才能灵活掌握的技能。欢迎你在后面，做更多的尝试。

**练习**

经典电影知多少  
周末夜晚，我们解锁了吴氏私厨，再来看部经典影片来享受闲暇好时光吧 (　･ิω･ิ)ノิ  
要说到经典影片，一定要到文艺青年们热爱的豆瓣找一下，看看 TOP250都有哪些呢～  
聪明的你可以用两种方法把 序号 /电影名/评分/推荐语/链接 都爬取下来帮助吴枫挑选电影嘛～

接下来我们一起分析网页吧～  
进入首页 <https://movie.douban.com/top250?start=0&filter=> ，打开检查工具，在Elements里查看这个网页，是什么结构。点击开发者工具左上角的小箭头，选中“肖申克的救赎”，这样就定位了电影名的所在位置，审查元素中显示<span class="title">：<span>标签内的文本，class属性；推荐语和评分也是如此，<span class='ing'>，<span class='rating\_num'>；序号：<em class>，<em>标签内的文本，class属性；推荐语<span class='ing'>；链接是<a>标签里href的值。最后，它们最小共同父级标签，是<li>。  
我们再换个电影验证下找的规律是否正确。  
check后，我们再看一共10页，每页的url有什么相关呢？  
第1页：<https://movie.douban.com/top250?start=0&filter=>  
第3页：<https://movie.douban.com/top250?start=50&filter=>  
第7页：<https://movie.douban.com/top250?start=150&filter=>  
发现只有start后面是有变化哒，规律就是第N页，start=(N-1)\*25  
基于以上分析，我们有两种写爬虫的思路。  
思路一：先爬取最小共同父级标签 <li>，然后针对每一个父级标签，提取里面的序号/电影名/评分/推荐语/链接。  
思路二：分别提取所有的序号/所有的电影名/所有的评分/所有的推荐语/所有的链接，然后再按顺序一一对应起来。

练习介绍

你已经学习了用bs库解析数据和提取数据的方法，也做了豆瓣电影top250的爬取练习，现在我们一起试试下载电影吧٩̋(๑˃́ꇴ˂̀๑)

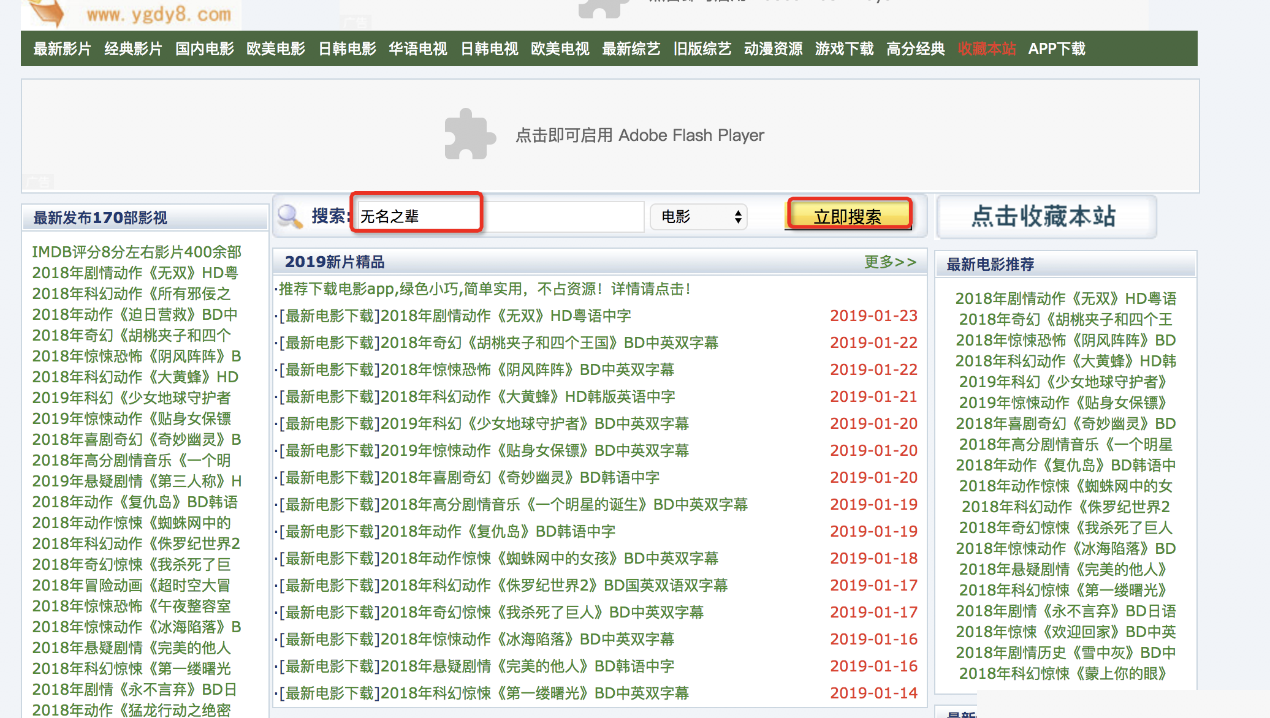
要求：

实现功能：用户输入喜欢的电影名字，程序即可在电影天堂[https://www.ygdy8.com](https://www.ygdy8.com/)爬取电影所对应的下载链接，并将下载链接打印出来。

步骤讲解

我们知道爬虫是模拟人在浏览器的动作批量获取有价值的信息，那对于这道题，我们先手动操作下，看看人是如何实现这个过程的。

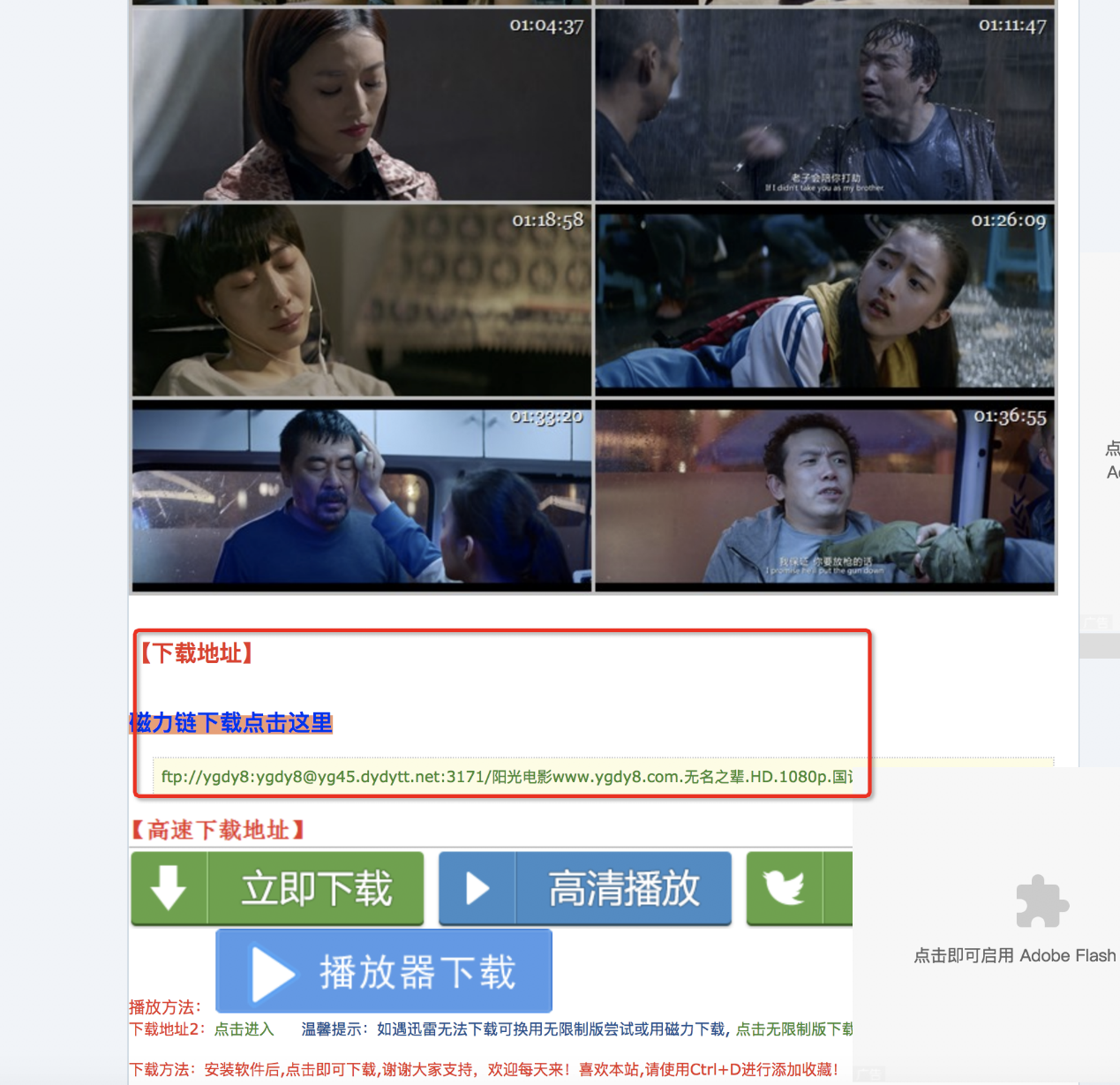
首先，打开电影天堂[https://www.ygdy8.com](https://www.ygdy8.com/) ， 在”搜索“处，填写一部电影名，以”无名之辈“为例，



然后，我们进入了“搜索结果”页面，



最后，在下载页面滑到最下方，找到了下载地址。



人工操作的步骤，我们是不是可以将其分为 “输名字 - 查搜索结果 - 进入下载页面 - 找到下载链接” ？

呐，我们就让我们的爬虫也这样子走就可以了对吧～

**步骤一**

“输名字”，学过基础课的同学一定可以想到，用input()就可以啦。

**步骤二**

”搜索结果页面“ 这里面涉及到一个坑，我们要一起填上。

输入不同的电影名，观察搜索结果页面的URL：  
《无名之辈》的搜索结果URL：[http://s.ygdy8.com/plus/so.php?typeid=1&keyword=%CE%DE%C3%FB%D6%AE%B1%B2](http://s.ygdy8.com/plus/so.php?typeid=1&keyword=%CE%DE%C3%FB%D6%AE%B1%B2" \t "_blank)  
《神奇动物》的搜索结果URL：[http://s.ygdy8.com/plus/so.php?typeid=1&keyword=%C9%F1%C6%E6%B6%AF%CE%EF](http://s.ygdy8.com/plus/so.php?typeid=1&keyword=%C9%F1%C6%E6%B6%AF%CE%EF" \t "_blank)  
《狗十三》 的搜索结果URL：[http://s.ygdy8.com/plus/so.php?typeid=1&keyword=%B9%B7%CA%AE%C8%FD](http://s.ygdy8.com/plus/so.php?typeid=1&keyword=%B9%B7%CA%AE%C8%FD" \t "_blank)  
观察URL，不难发现：[http://s.ygdy8.com/plus/so.php?typeid=1&keyword=](https://www.pypypy.cn/) 这些都是一样的，只不过不同的电影名对应URL后面加了一些我们看不懂的字符，

请阅读以下代码，注意注释哦：

a= '无名之辈'

b= a.encode('gbk')

# 将汉字，用gbk格式编码，赋值给b

print(quote(b))

# quote()函数，可以帮我们把内容转为标准的url格式，作为网址的一部分打开

输出结果是：

1

%CE%DE%C3%FB%D6%AE%B1%B2

记得在本地IDE上试着敲下哦，可以把‘无名之辈’换成神奇动物和狗十三，看看输出结果是否和对应的编码一致。  
诶，发现的确是一致的，那我们一起来解读这段代码是怎么实现的呢。  
注释中提到了gbk格式编码，那gbk是什么呢？

GBK编码

GBK是中国标准，只在中国使用，并没有表示大多数其它国家的编码；  
而各国又陆续推出各自的编码标准，互不兼容，非常不利于全球化发展。  
于是后来国际组织发行了一个全球统一编码表，把全球各国文字都统一在一个编码标准里，名为Unicode。

由此我们知道，想把中文转换成url格式，需要先用encode('gbk')将其转成gbk编码，然后再用quote()把它转化成url的一部分。  
然后再将它与[http://s.ygdy8.com/plus/so.php?typeid=1&keyword=](https://www.pypypy.cn/)拼接起来就是电影的搜索结果页面啦～

好，总结下上面的这个知识点，如何建立“输入电影名”与“搜索结果页面”的联系：

**中文 - gbk - url - 拼接**

这样我们就把完整的搜索结果页面找到并提取出来了。

**步骤三 + 步骤四**

”进入下载页面“ 与 “找到下载链接” 就是解析网页定位啦，利用find() 和 find\_all()，都是你会的内容，加油呀～