

lab1 html Parser

1.实验概览

本次实验使用python自带的urllib库实现通过自动程序对html网站进行访问并返回相应结果，并使用beautifulsoap库对返回的结果进行解析和筛选，以得到期望的结果，其中可以使用正则表达式来对解析的结果进行更为精确的筛选。

2.实验环境

本实验基于 Docker 中的 DEV-container--SJTUEE208 具体使用到的环境为：

1. Ubuntu 20.04.5 LTS
2. Vscode python的urllib库, beautiful soap库以及re库

3.练习

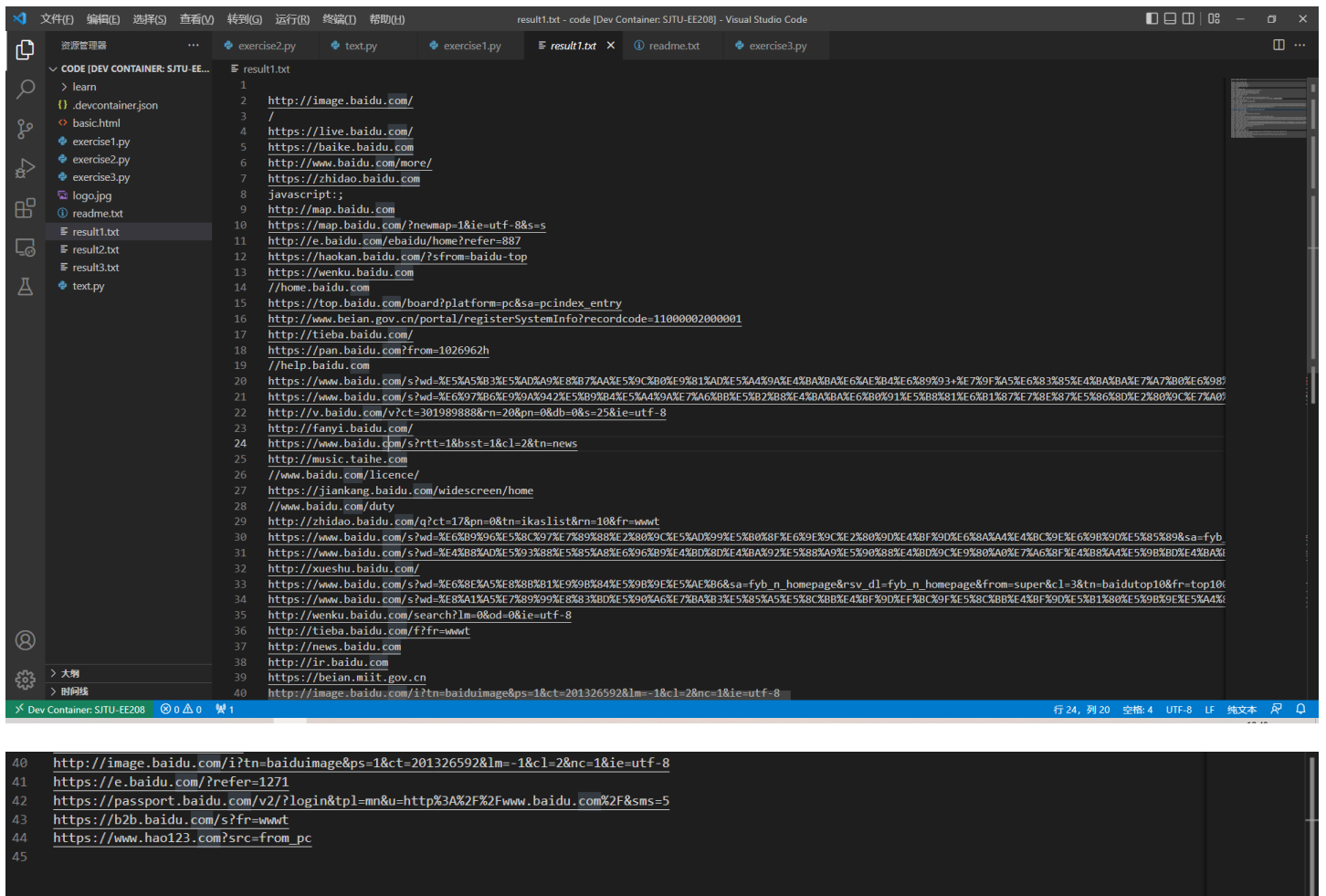
3.1 练习1

解决思路：通过urllib.urlopen方法对网页进行访问并返回相应结果，使用beautiful soap对返回结果进行解析，在parseURL函数中使用findAll方法和get方法来获得原网页中的所有超链接的url，并将其加入urlset中，最后通过write_outputs函数将所得超链接写入result1.txt中。

核心代码展示如下：

```
1 def parseURL(content):
2     urlset = set()
3     soup = BeautifulSoup(content, 'html.parser')
4     for i in soup.findAll('a'):
5         urlset.add(i.get('href', ''))
6     return urlset
```

运行结果如下：



3.2 练习2

解决思路：用户输入想要访问的网站的http地址，访问方式和解析方式基本同练习1，唯一不同的是通过`urllib.request.Request.add_header()` 方法添加了请求标头，以通过部分网站对爬虫程序的反爬。

同时，对于一些目前暂时无法通过的反爬技术，我参考[知乎：python爬虫入门：HTTPError异常处理](#)加入了一个catch

结果写入在result2中

核心代码展示如下：

```
1 def parseURL(content):
2     urlset = set()
3     soup = BeautifulSoup(content, 'html.parser')
4     for i in soup.findAll('img'):
5         toAdd = i.get('src', '')
6         urlset.add(toAdd)
7     return urlset
```

```
1 def main():
```

```

2 url = input()
3 req = urllib.request.Request(url)
4 req.add_header('User-Agent','Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit:
5 try:
6     content = urllib.request.urlopen(req).read()
7     urlSet = parseURL(content)
8     write_outputs(urlSet, "result2.txt")
9 except urllib.error.HTTPError:
10    print('本次访问失败')

```

运行结果如下：

访问成功时：

The screenshot shows the Visual Studio Code interface. The file explorer on the left lists files including 'result2.txt'. The code editor displays the contents of 'result2.txt', which is a list of URLs. The terminal at the bottom shows the command prompt for a Jupyter environment.

```

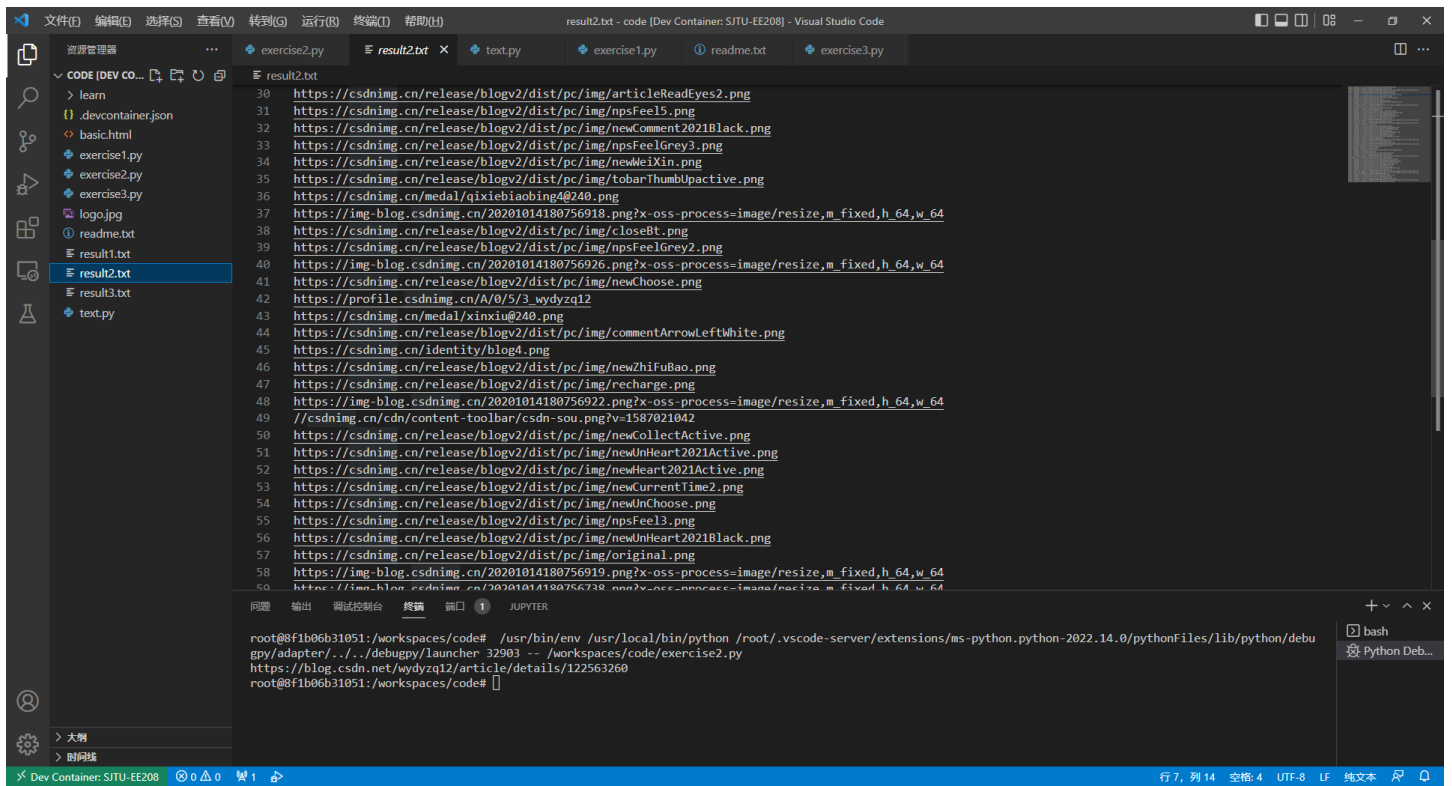
result2.txt
1 https://csdnimg.cn/release/blogv2/dist/pc/img/newRewardBlack.png
2 https://csdnimg.cn/release/blogv2/dist/pc/img/zhifubao.png
3 https://img-blog.csdnimg.cn/20201014180756923.png?x-oss-process=image/resize,m_fixed,h_64,w_64
4 https://csdnimg.cn/release/blogv2/dist/pc/img/tobarCollect2.png
5 https://csdnimg.cn/release/blogv2/dist/pc/img/npsFeelGrey4.png
6 https://csdnimg.cn/release/blogv2/dist/pc/img/commentArrowDownWhite.png
7 https://csdnimg.cn/release/blogv2/dist/pc/img/npsFeel4.png
8 https://csdnimg.cn/release/blogv2/dist/pc/img/newCollectBlack.png
9 https://g.csdnimg.cn/static/face/emoji/015.png
10 https://csdnimg.cn/release/blogv2/dist/pc/img/weixin.png
11 https://csdnimg.cn/release/blogv2/dist/pc/img/tobarCollectionActive.png
12 https://csdnimg.cn/identity/nocErtification.png
13 https://img-blog.csdnimg.cn/20201014180756757.png?x-oss-process=image/resize,m_fixed,h_64,w_64
14 https://g.csdnimg.cn/static/face/emoji/062.png
15 https://csdnimg.cn/release/blogv2/dist/pc/img/iconPark.png
16 https://csdnimg.cn/medal/chizhiyiheng@240.png
17 https://csdnimg.cn/release/blogv2/dist/pc/img/arrowDownWhite.png
18 https://csdnimg.cn/release/blogv2/dist/pc/img/tobarCollectionActive2.png
19 https://csdnimg.cn/release/blogv2/dist/pc/img/newShareBlack.png
20 https://csdnimg.cn/release/blogv2/dist/pc/img/newHeart2021Black.png
21 https://csdnimg.cn/release/blogv2/dist/pc/img/pay-time-out.png
22 https://csdnimg.cn/release/blogv2/dist/pc/img/jingdong.png
23 https://img-blog.csdnimg.cn/20201014180756925.png?x-oss-process=image/resize,m_fixed,h_64,w_64
24 https://csdnimg.cn/release/blogv2/dist/pc/img/pay-help.png
25 https://img-blog.csdnimg.cn/20201014180756927.png?x-oss-process=image/resize,m_fixed,h_64,w_64
26 https://csdnimg.cn/medal/51_create.png
27 https://csdnimg.cn/release/blogv2/dist/pc/img/npsFeel1.png
28 https://csdnimg.cn/release/blogv2/dist/pc/img/newArrowDownWhite.png
29 https://csdnimg.cn/release/blogv2/dist/pc/img/articleReadEval2.png
30 https://csdnimg.cn/release/blogv2/dist/pc/img/articleReadEval2.png

```

```

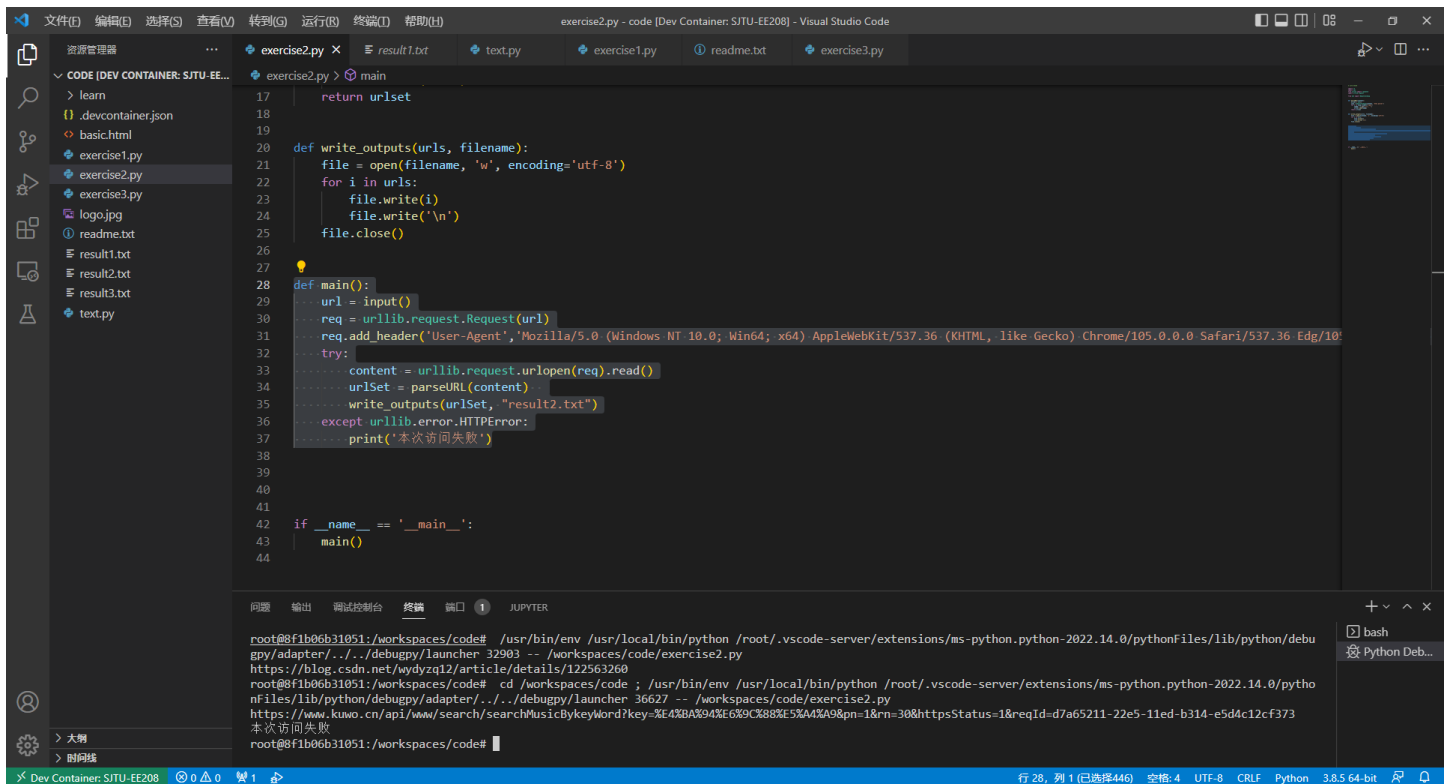
root@8f1b06b31051: /workspaces/code# /usr/bin/env /usr/local/bin/python /root/.vscode-server/extensions/ms-python.python-2022.14.0/pythonFiles/lib/python/debu
gpy/adapters/.../debugpy/launcher 32903 -- /workspaces/code/exercise2.py
https://blog.csdn.net/wydyzq12/article/details/122563260
root@8f1b06b31051: /workspaces/code#

```



(注：这里使用的链接是Python爬虫：测试网址是否请求成功)

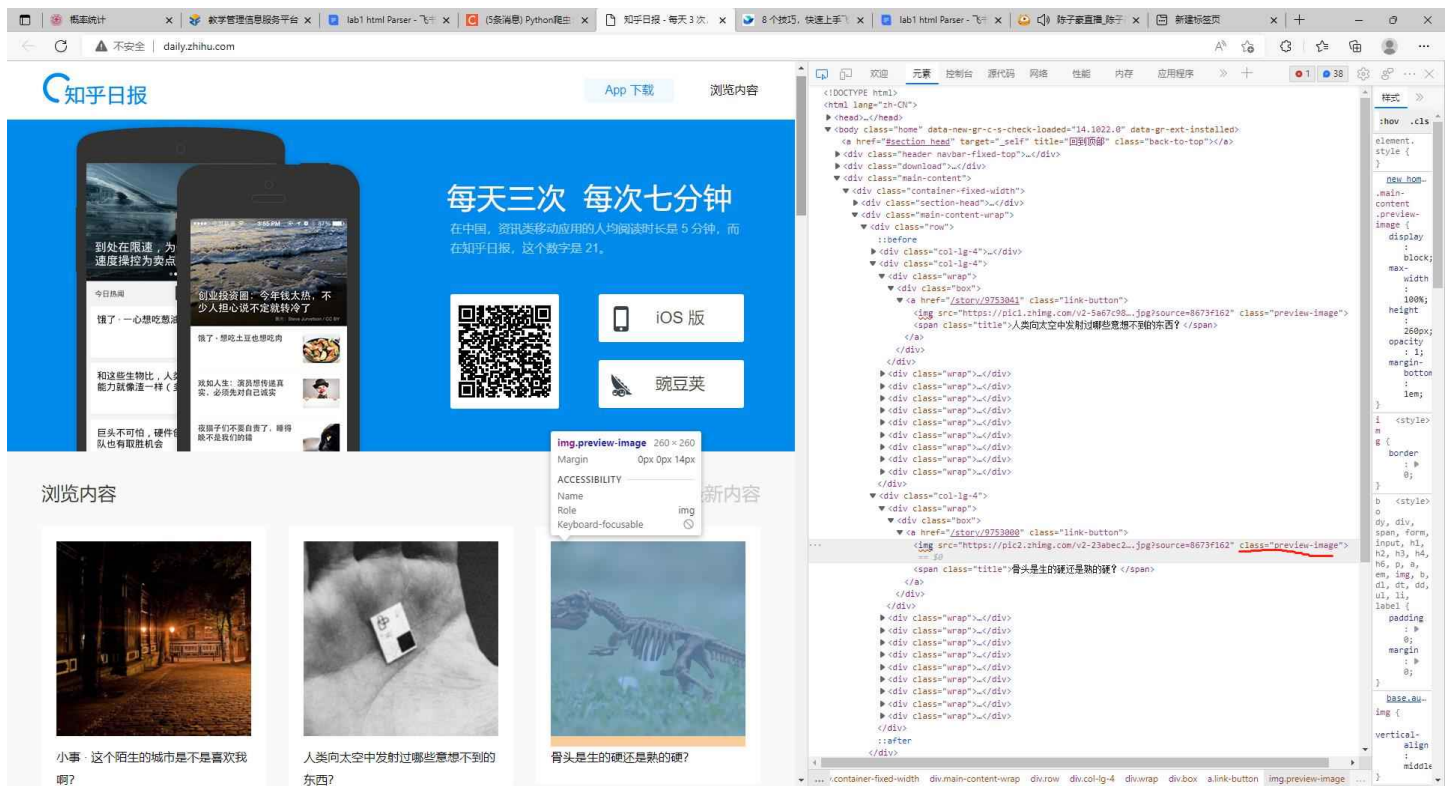
访问失败时：



(注：这里使用的链接是<https://www.kuwo.cn/api/www/search/searchMusicBykeyWord?key=%E4%B4%A9&pn=1&rn=30&httpsStatus=1&reqId=d7a65211-22e5-11ed-b314-e5d4c12cf373>)

3.3 练习3

解决思路：首先进入知乎日报页面，打开F12审查我们要获得的链接的源代码



首先我们发现需要获取的图片地址所对应的html代码中，所属的类是‘preview-image’类，进而发现图片所链接的网页地址存放在图片链接的父节点中，同时所需要提取的文本存放在图片链接的姐妹节点中，于是我的方案是先通过findAll找出所有类名为‘preview-image’的连接，再通过访问父节点得到图片的超链接地址，最后通过访问父节点的contents[1].string来获得图片对应的文本。

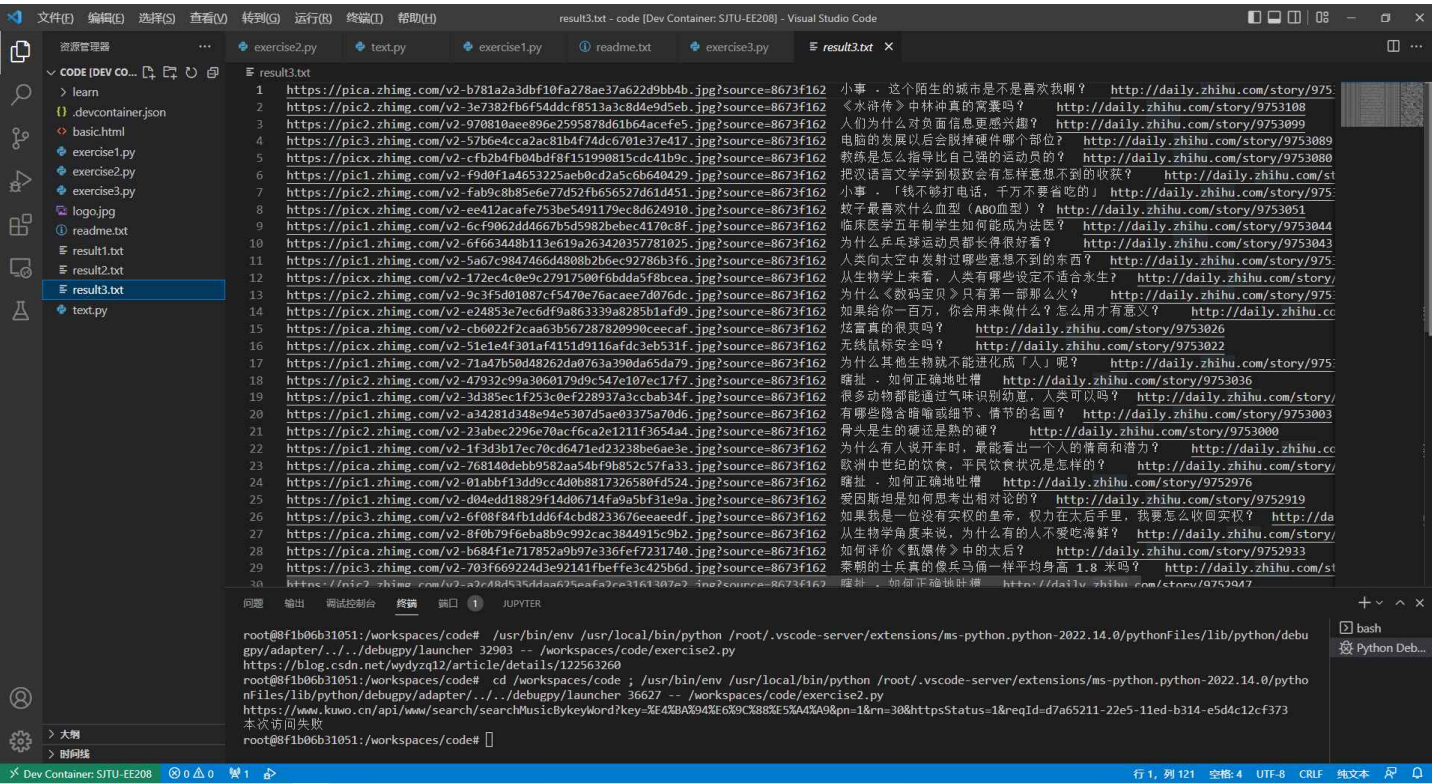
这里为了防止爬到一些同样属于‘preview-image’类却没有文本的图片，于是增加了一条判断语句。

最后将结果按格式写入在result3中

核心代码展示如下：

```
1 def parseZhihuDaily(content, url):
2     zhihulist = list()
3     soup = BeautifulSoup(content, 'html.parser')
4     for i in soup.findAll('img', {'class' : 'preview-image'}):
5         src = i.get('src', '')
6         templinkpage = i.parent.get('href', '')
7         temptitle = i.parent.contents
8         if (len(temptitle) >= 2 and templinkpage):
9             title = temptitle[1]
10            linkpage = urllib.parse.urljoin(url, templinkpage)
11            zhihulist.append([src, title.string, linkpage])
12    return zhihulist
```


运行结果展示如下



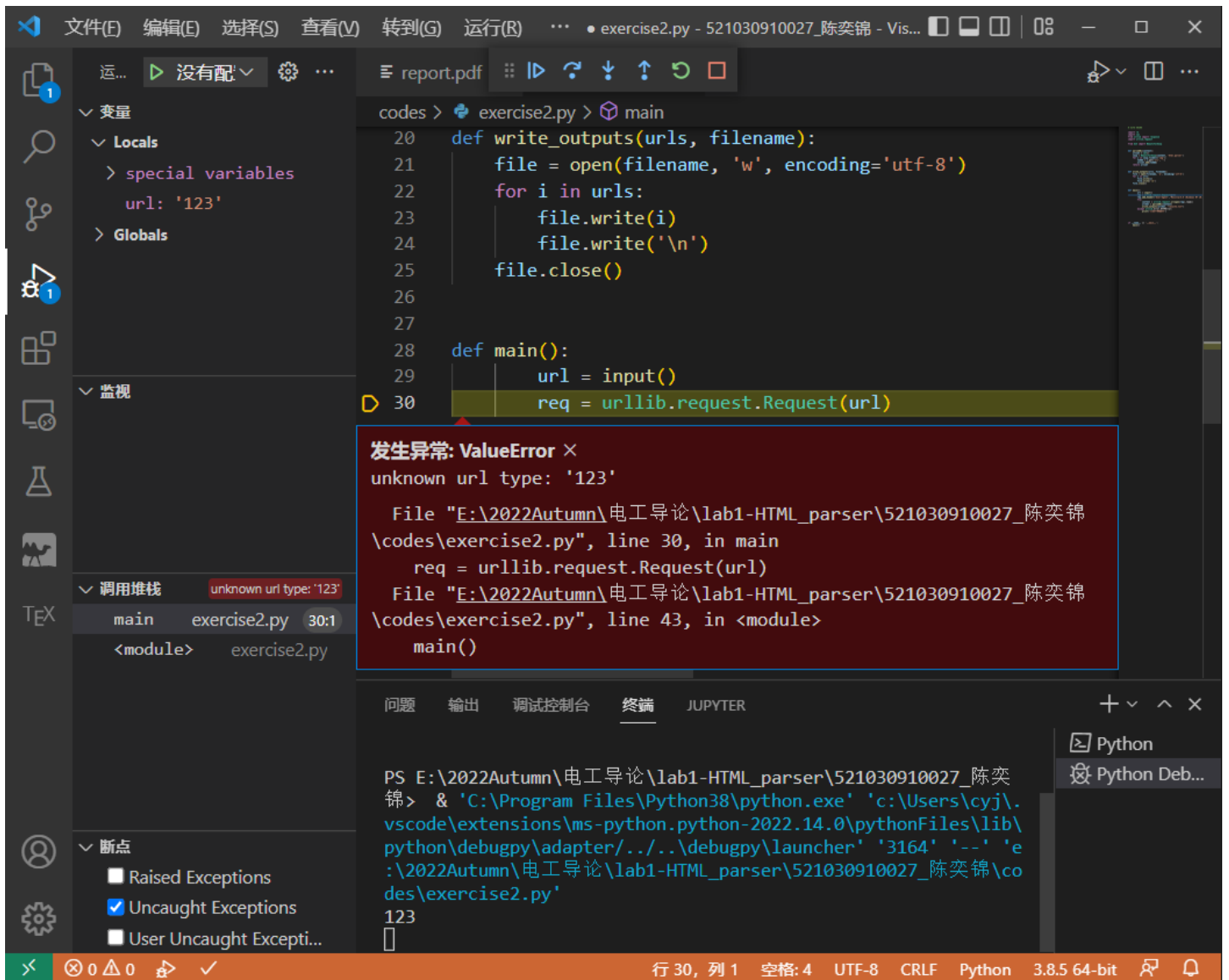
4.自己的分析与思考

1.爬取到的rhef有几种形式？

- E1: <http://image.baidu.com/>
- E2: <https://live.baidu.com/>
- E3: <//help.baidu.com>
- E4: javascript;;
- 其中E1，E2为可以正常访问的链接
- E3为相对于原网页的相对路径，如果想要正常访问的话需要在前面添加http:或者https:
- E4应该为链接到javascript的链接，用于进行一些按钮之类的交互行为，截取网页源码如下

```
    吸血</div>
    <a class="s-switch-account" href="javascript:;">切换帐号</a>
    <a class="quit" style="overflow:hidden" href="javascript:;" onclick="return false;">退出登录</a>
  </div>
  <div id="s-user-setting-menu" class="s-top-userset-menu c-floating-box c-font-normal">
    <div class="s-user-setting-pfmenu"></div>
    <a class="s-set-feed hide-feed" href="javascript:;">隐藏资讯</a>
    ...
    <a class="s-set-feed show-feed" href="javascript:;">展示资讯</a> == $0
    <span class="s-set-homepage-tts">...</span>
    <span class="split-line"></span>
```

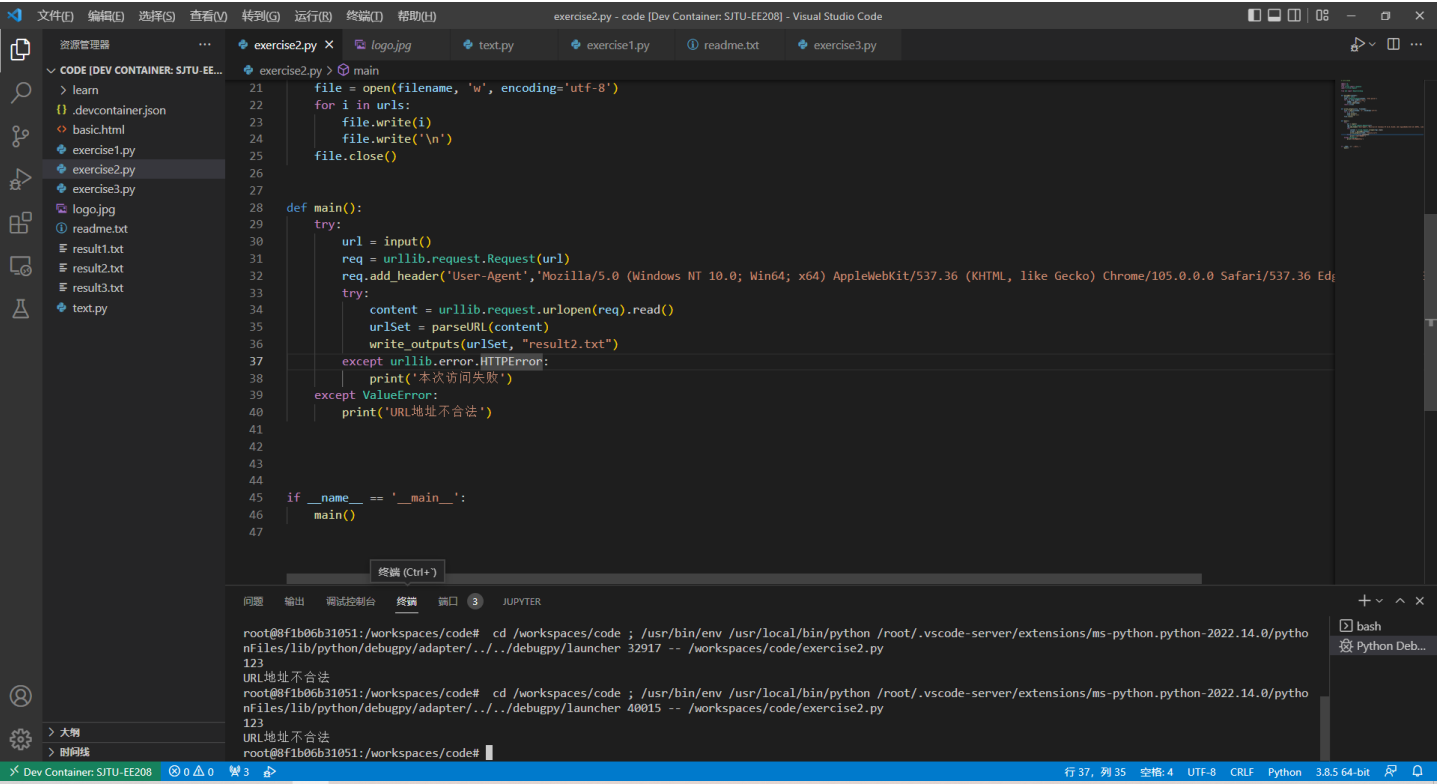
2.关于练习2中写的catch的问题，它好像只能处理HTTP地址合法但是网页无法访问的情况，如果输入一些不合法的HTTP地址就会出现下图的情况



所以我就又加了一层try和catch，好像就有点复杂了

```
1 def main():
2     try:
3         url = input()
4         req = urllib.request.Request(url)
5         req.add_header('User-Agent', 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) Ap
6     try:
7         content = urllib.request.urlopen(req).read()
8         urlSet = parseURL(content)
9         write_outputs(urlSet, "result2.txt")
10    except urllib.error.HTTPError:
11        print('本次访问失败')
12    except ValueError:
13        print('URL地址不合法')
```

运行结果如下：



虽然但是我觉得也许用requests库然后直接判断它的statuscode是否等于两百应该比这个要稍微好一点，但是由于requests库不是python自带的于是我就没有把代码改成使用requests库的形式了。

所以说这算不算提一个建议就是用requests库会方便一点呢