App商店爬虫说明文档

1.0 使用说明

1. 在浏览器键入URL：http://192.168.50.173:8888/index.html/
2. 在搜索栏键入想要搜索的软件的关键字
3. 点击搜索按钮
4. 等待程序运行完成，并返回表格型数据

2.0 相应模块说明

2.1 后台系统

使用 Django 框架搭建后台，此处将分为路由，响应函数，数据库三块进行说明，我在 Django 中创建了一个名为 `spider` 的 app，并在这个 app 里面编写爬虫应用相关的代码

2.1.1 路由

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('get_keyword/', crawl, name='get_keyword'),
    path("index.html/", orgin),
]
```

- 系统的主页面为 http://192.168.50.173:8888/index.html/
- 系统的搜索请求地址为 http://192.168.50.173:8888/get_keyword/?keyword=xxx

2.1.2 响应函数

- http://192.168.50.173:8888/index.html/ 对应的响应函数为 `crawl`

```
# Create your views here.
def crawl(request):
    keyword = request.GET.get("keyword")
    print(keyword)
    CombineSpider.crawl(keyword)
    engine = create_engine('mysql+pymysql://root:shuziguanxing123456@192.168.50.60:3306/app_info')
    app_info = pd.read_sql_table('spider_app', engine)
    x_io = BytesIO()
    app_info.to_excel(x_io, sheet_name="app_info", index=False)
    response = HttpResponse()
    response['Content-Type'] = 'application/octet-stream'
    response['Content-Disposition'] = 'attachment;filename="app_info.xlsx"'
    response.write(x_io.getvalue())
    return response
```

`crawl` 响应函数的作用是，获取用户发送过来的关键字，根据关键字运行爬虫程序，将数据存入 MySQL 数据库中，并将结果以 .xlsx 的 Excel 文件返回

2.1.3 数据库

本系统的数据库部署在 **192.168.50.60** 上，使用 MySQL 作为数据库，访问数据库的用户名为 `root`，密码为 `shuziguanxing123456`，存储数据库的表名为 `app_info`

```
DATABASES = {
    # 'default': {
    #     'ENGINE': 'django.db.backends.sqlite3',
    #     'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    # }
    'default':{
    'ENGINE' :'django.db.backends.mysql',
    'NAME':'app_info',
    'USER':'root',
    'PASSWORD':'shuziguanxing123456',
    'HOST':'192.168.50.60',
    'PORT':'3306',
    }
}
```

2.2 爬取模块

2.2.1 请求模块

本系统中和请求相关的程序封装在文件名为 `request_compoent.py` 的程序中，下面详细说明

2.2.1.1 get_random_ua 函数

```
def get_random_ua(app=False) -> list:
    andriod_ua = [
        "Mozilla/5.0 (Linux; Android 9; PAFM00 Build/PKQ1.1903
        "Mozilla/5.0 (Linux; Android 6.0; BLN-AL10 Build/HONORE
        "Mozilla/5.0 (Linux; Android 8.1.0; OPPO R11 Build/OPM1
        "Mozilla/5.0 (Linux; Android 9; MI CC 9 Build/PKQ1.1811
        "Mozilla/5.0 (Linux; Android 9; vivo X21A Build/PKQ1.18
    ]

    pc_ua = [
        "Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; Tri
        "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/
        "Mozilla/5.0 (Windows NT 6.1; rv,2.0.1) Gecko/20100101
        "Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; Tri
        "Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; 360
    ]

    if app:
        return random.choice(andriod_ua)

    return random.choice(pc_ua)
```

这里人为加入了一些可以使用的请求头，分为移动端的和 PC 端的，此函数的作用是根据传入的 app 的值，返回随机选择出的一个 User-Agent

2.2.1.2 return_headers_and_proxies 函数

```python
def return_headers_and_proxies(app=False):
    """随机选择当前此次请求是否使用代理"""
    if not app:
        headers = {
            "User-Agent": get_random_ua(app=False),
        }
    else:
        headers = {
            "User-Agent": get_random_ua(app=True),
        }

    proxies = None
    seed = [i for i in range(5)]  # 1/5 概率使用代理
    result = random.choice(seed)
    if not result:
        secret = "6508ddd843e78d9350b787ae81c8420e"
        orderId = "DT20200410155216L1evf4mh"
        timestamp = str(int(time.time()))
        txt = "orderno={},secret={},timestamp={}".format(orderId, secret, timestamp)
        sign = md5(txt.encode()).hexdigest().upper()
        headers = {**headers, **{
            "Proxy-Authorization": "sign={}&orderno={}&timestamp={}&change=true".format(sign, orderId, timestamp)}}
        proxies = {
            "http": "http://" + "dynamic.xiongmaodaili.com:8088",
            "https": "https://" + "dynamic.xiongmaodaili.com:8088"
        }

    return headers, proxies
```

参数 app 表示当前需要请求的 URL 是否为移动端的 URL，此函数为请求代理的设置，随机判断当前请求是否使用代理，从 0 到 4 中随机选出一个数，如果这个数是 0 则使用代理，所以是有 1/5 的概率使用代理，使用代理的方法此处已经封装好，具体可见 http://www.xiongmaodaili.com/helpDel?id=9

2.2.1.3 RqCompoent.get 方法

```python
class RqCompoent():

    @staticmethod
    def get(url, app=False, *args, **kwargs):
        # 请求代理 && 随机获取请求头
        headers, proxies = return_headers_and_proxies(app)
        # 扩展请求头
        headers = {**headers, **kwargs}
        if not url:
            return None
        if not proxies:
            response = requests.get(url, headers=headers)
        else:
            response = requests.get(url, headers=headers, proxies=proxies)

        if response.status_code == 200:
            try:
                response = response.content  # 得到字节
                charset = chardet.detect(response).get('encoding')  # 得到编码格式
                # print(charset)
                response = response.decode(charset, "ignore")  # 解码得到字符串
                return response
            except:
                return None
        else:
            print("请求失败")
            return None
```

用于发送 get 请求，支持以功能

- 使用代理
- 随机选择 UA
- 扩展请求头
- 自动判断响应体的编码并解码

使用的时候可以直接通过 `类名.方法名` 调用

2.2.1.4 RqCompoent.post 方法

```
@staticmethod
def post(url, data, app=False, *args, **kwargs):
    headers, proxies = return_headers_and_proxies(app)
    # 扩展请求头
    if not url:
        return None
    headers = {**headers, **kwargs}
    if not proxies:
        response = requests.post(url, headers=headers, data=data, timeout=6)
    else:
        response = requests.post(url, headers=headers, data=data, proxies=proxies, timeout=6)
    # response = requests.post(url, headers=headers, data=data, timeout=10)
    if response.status_code == 200:
        response = response.content
        charset = chardet.detect(response).get('encoding')  # 得到编码格式
        response = response.decode(charset, "ignore")  # 解码得到字符串
        return response
    else:
        print("请求失败")
```

用于发送 post 请求，扩展功能和使用方法同上

2.2.2 Html / Json 解析模块

这里将处理响应体为 Html 格式和 Json 格式的程序封装在 `ParseComponent`，`ParseComponentAjax` 两个文件中，这里详细说明 `ParseComponent`，`ParseComponentAjax` 与前者基本相比上没有太大变化

2.2.2.1 ParseComponent

这里解释一下：`ParseCompoent` 和 `ParseCompoentAjax` 其实差别是不大的，如果返回的是 Json 格式的数据，程序就继承 `ParseCompoentAjax` 去处理就好了，主要逻辑和前者是一样的

```
class ParseComponent(object):
    def __init__(self, url, keyword):
        self.keyword = keyword
        # 对 url 里面的中文字符进行编码
        self.quote_keyword = urllib.parse.quote(keyword)
        # self.url = url + self.quote_keyword
        self.match_keyword = False
        self.delay_time = 0.05
        self.add_headers = {}
        self.n_page = 1
        self.name = "xx应用市场"
        self.db = pymysql.connect("192.168.50.60", user="root",\
        passwd="shuziguanxing123456", db="app_info")
        self.cursor = self.db.cursor()
```

初始化类的属性

- `self.keyword` 搜索的关键字
- `self.quote_keyword` 经过 URL 编码的关键字
- `self.delay_time` 每一轮请求的延时(s)
- `self.add_headers` 需要扩展请求头（如 Cookie、Referer）
- `self.n_page` 需要爬取的页数

- `self.name` 当前爬虫程序要爬的应用商店的名字
- `self.db` 连接上的 MySQL 数据库
- `self.cursor` 游标，用于写入数据

1）get_page_n_url 方法

```
def get_page_n_url(self, n):
    """返回前 n 页的 url"""
    if n == 1:
        return self.url
    # 返回从第二页开始的 url
    page_n_url = self.url + "&page={}".format(n)
    return page_n_url
```

此方法用于编写翻页的规则，需要传入的参数 `n` 表示需要爬多少页

2）parse_app_list_page 方法

```
def parse_app_list_page(self):
    """
    1.遍历匹配关键字
    2.匹配成功则只爬那一个
    3.匹配失败则把前 n 页的数据爬下来，默认值为 2
    :return:
    """
    for i in range(1, self.n_page+1):
        page_n_url = self.get_page_n_url(i)
        lis = None
        lis = self.get_app_list_elements(page_n_url)
        if not lis:
            continue
        if lis == "error":
            continue
        # 比对关键字，如果全部匹配，则只取这一个app的信息
        # 如果匹配不上，则把前 n 页的app信息爬下来
        if i == 1:
            self.loop_request(lis, first_page=True)
        else:
            self.loop_request(lis, first_page=False)
```

遍历所有需要翻页的页面，用 `self.get_app_list_elements` 方法每个页面找出包含具体信息的盒子模型，遍历些盒子模型提取信息

3）parse_app_info_page 方法

```python
def parse_app_info_page(self, inner_response):
    """
    四个字段
    1.App名称 这个在外层获取了
    2.更新时间
    3.发行商
    4.下载地址
    :return:
    """
    update_time = None
    author = None
    download_url = None
    version = None

    update_time = self.get_update_time(inner_response)
    author = self.get_author(inner_response)
    download_url = self.get_download_url(inner_response)
    version = self.get_version(inner_response)
    # # 判断是否为空列表
    update_time = self.judge_null(update_time)
    author = self.judge_null(author)
    version = self.judge_null(version)
    download_url = self.judge_null(download_url)
    if update_time:
        update_time = re.sub("[年月日]", '/', update_time)
    return [version, update_time, author, download_url]
```

传入一个参数 `inner_response` ，获取软件的版本、更新日期、开发商、下载链接，并以列表的形式返回

4）catch_error 装饰器函数

```python
def catch_error(method_func):
    """异常捕获装饰器"""
    def wrapper(self, _attr):
        try:
            method_func(self, _attr)
        except:
            pass
    return wrapper
```

秉承代码简洁大方的设计风格，使用此装饰器函数捕获异常😂

5）get_app_list_elements 方法

```python
@catch_error
def get_app_list_elements(self, url) -> list:
    """
    获取搜索结果列表对应的元素
    :return: 搜索结果元素列表
    """
    response = RqCompoent.get(url)
    if not response:
        return "error"
    self.outer_response = response
    selector = etree.HTML(response)
    lis = selector.xpath()
    return lis
```

发送请求并得到页面的 Html 源码，找出每个页面包含具体信息的盒子模型，以列表的形式返回，使用装饰器捕获异常

6）loop_requet 方法

```python
def loop_request(self, lis, first_page=True, **kwargs):
        """循环请求"""

        if not lis:
            return
        for li in lis:
            enter_url = None
            enter_url = self.get_enter_url(li)  # 获取详情页url
            inner_response = RqCompoent.get(enter_url, **self.add_headers)
            if inner_response:
                self.inner_response = inner_response
                self.li = li

                app_name = None
                img_address = None
                app_intro = None

                app_name = self.get_app_name(li)
                app_name = self.judge_null(app_name)
                app_name = self.field_strip(app_name)
                img_address = self.get_img_address(li)
                app_intro = self.get_app_intro(inner_response)
                fields = self.parse_app_info_page(inner_response)
                to_sink = [self.name, app_name, *fields, img_address, app_intro]
                res = []
                for i in to_sink:
                    if not i:
                        res.append(None)
                    else:
                        res.append(pymysql.escape_string(i))
                try:
                    print(*res[:-1], res[-1][:20])
                except:
```

```
                    pass
                sql = "insert into spider_app(appStore, appName, version,
updateTime, author,downloadUrl,icon, introduction, inList, platform, insertTime,
keyword, enter_url) values(%s, %s, %s, %s, %s, %s, %s, %s, '否', '安卓', %s, %s,
%s)"
                res = [*res, pymysql.escape_string(time.strftime("%Y/%m/%d",
time.localtime())), pymysql.escape_string(self.keyword),
pymysql.escape_string(enter_url)]
                try:
                    self.cursor.execute(sql, res)
                except:
                    pass
                sleep(self.delay_time)
        self.db.commit()
```

遍历包含具体信息的盒子模型，从中提取*软件名*，*详情页的 URL*，并请求*详情页的 URL*，从中提取剩余需要爬取的字段的信息，并将这些信息按照 **[正在爬的应用商店，应用名，应用版本号，更新日期，开发商，下载地址，图标地址，应用介绍，爬取时的时间戳，输入的关键字]** 的顺序排列好

7）get_enter_url 方法

```
@catch_error
def get_enter_url(self, li, *args, **kwargs):
    """获取详情页的 url"""
    enter_url = li.xpath()
    return enter_url
```

编写获取详情页 URL 的代码

8）judge_null 方法

```
def judge_null(self, field):
    """判断是否为空列表"""
    if isinstance(field, str):
        return field
    if field:
        field = field[0]
        return field
    return None
```

判断字段的值是否套上了列表，如果套上了列表则取出其中的值，如果字段的值为空或空列表则返回 None

- **get_app_name** 方法

编写从当前的盒子元素中提取**软件名**的代码

- **get_version** 方法

编写从详情页响应体中提取**版本号**的代码

- **get_update_time** 方法

编写从详情页响应体中提取**更新时间**的代码

- **get_author** 方法

编写从详情页响应体中提取**开发商信息**的代码

- **get_download_url** 方法

编写从详情页响应体中提取**下载链接**的代码

- **get_img_address** 方法

编写从详情页响应体中提取**app图标地址**的代码

- **get_app_intro** 方法

编写从详情页响应体中提取**应用介绍**的代码

- **parse** 方法

执行爬虫整个爬取流程的方法

2.2.3 数据入库

```
        to_sink = [self.name, app_name, *fields, img_address, app_intro]
        res = []
        for i in to_sink:
            if not i:
                res.append(None)
            else:
                res.append(pymysql.escape_string(i))
        try:
            print(*res[:-1], res[-1][:20])
        except:
            pass
        sql = "insert into spider_app(appStore, appName, version, updateTime, author,\
        downloadUrl,icon, introduction, inList, platform, insertTime, keyword, enter_url)\
         values(%s, %s, %s, %s, %s, %s, %s, %s, '否', '安卓', %s, %s, %s)"
        res = [*res, pymysql.escape_string(time.strftime("%Y/%m/%d", time.localtime())),\
         pymysql.escape_string(self.keyword), pymysql.escape_string(enter_url)]
        try:
            self.cursor.execute(sql, res)
        except:
            pass
        sleep(self.delay_time)
self.db.commit()
```

将 **2.2.2 6)** 排列好的数据套入 SQL 语句当中，执行 SQL 语句，将内容写入内存中，在每遍历完一页将这部分数据流写入数据库，注意：如果直接使用字符串的格式化方法`format`，会有问题，要用`pymysql.escape_string`方法来处理需要格式化的字符串；还有需要注意的一点是，MySQL 数据库里的`NULL`的表示方法不是`pymysql.NULL`，而是 Python 里面的`None`

3.0 系统启动和停止

本项目的后台系统部署在 **192.168.50.173:8888** 上，后台系统程序文件在 */home/ubuntu/appinfo* 路径下，需要启动时，首先确保系统上有`screen`命令，没有的话`sudo apt-get install screen`一下，这个软件包是保证程序以守护进程的形式运行，不会在断开 SSH 之后让程序停止运行

- **启动系统**

> - `$ cd appinfo`
> - `$ screen`
> - `$ python manage.py runserver 0.0.0.0:8888`

- **停止系统**

- `$ ps -aux| grep python` ->找到对应的进程号
- `$ kill -9 进程号`

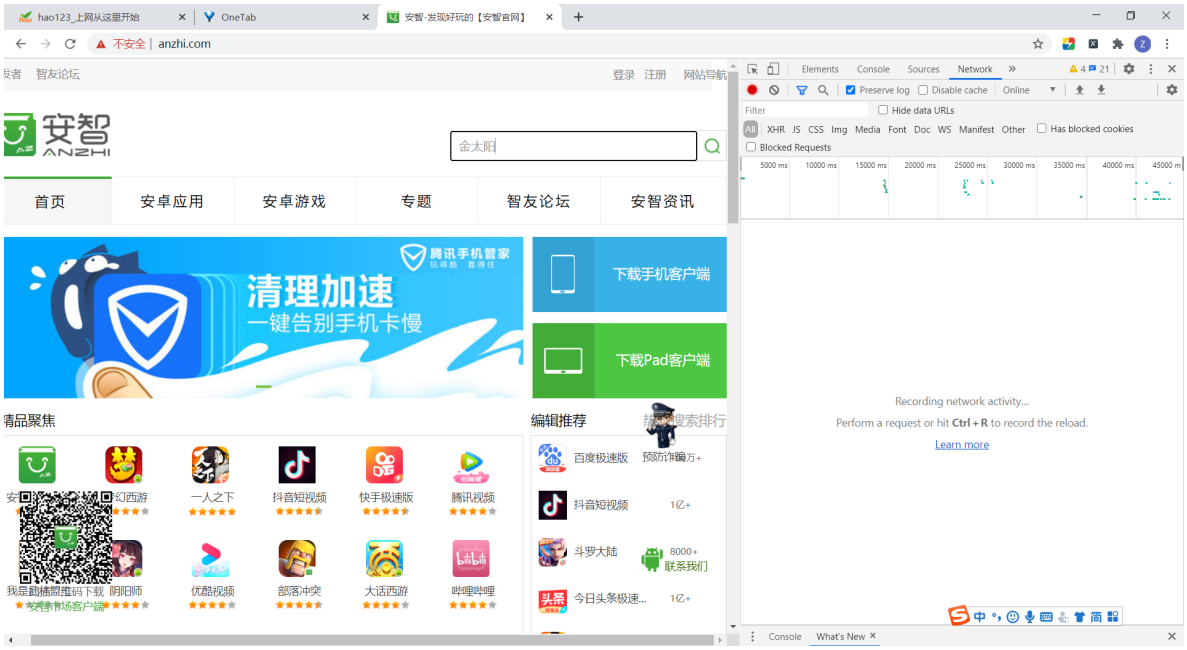
4.0 自定义 / 扩展程序

4.1 根据模板自定义爬虫

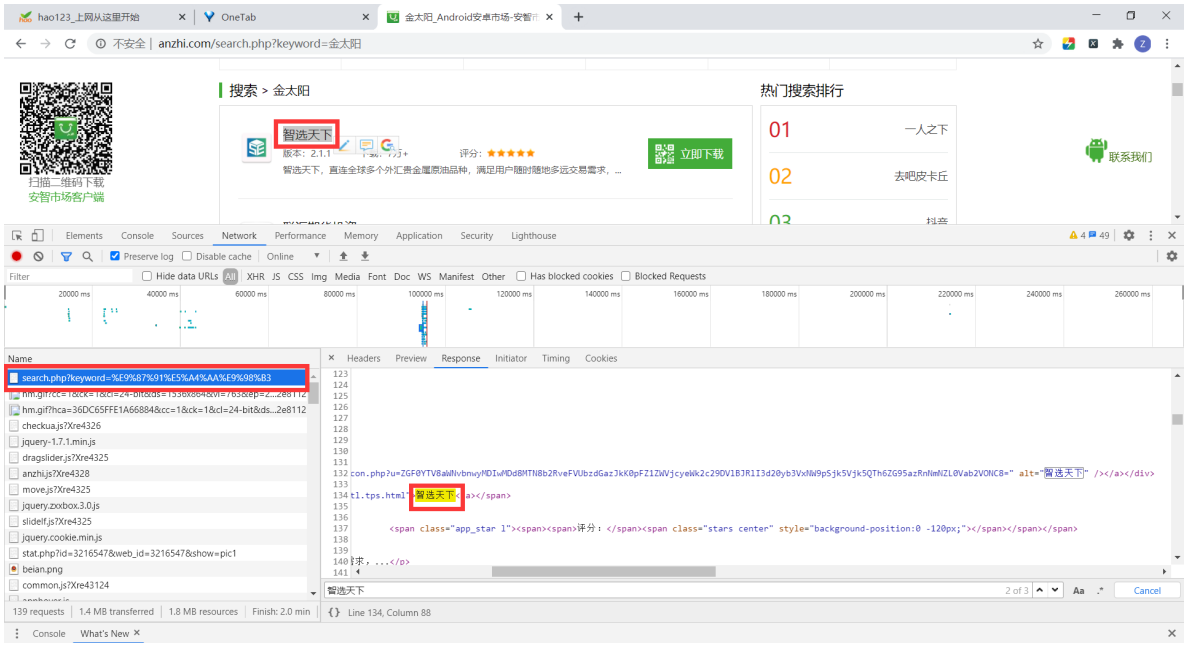
此处以安智应用商店为例，因为我们要根据认为输入的关键词进行爬取，所以我们要利用该网站的搜索功能，找到搜索功能对应的 URL

4.1.1 分析搜索对应的 URL 返回的为 Html 还是 Json

- 在 Chrome 的 `F12` 进行抓包



- 模拟搜索过程



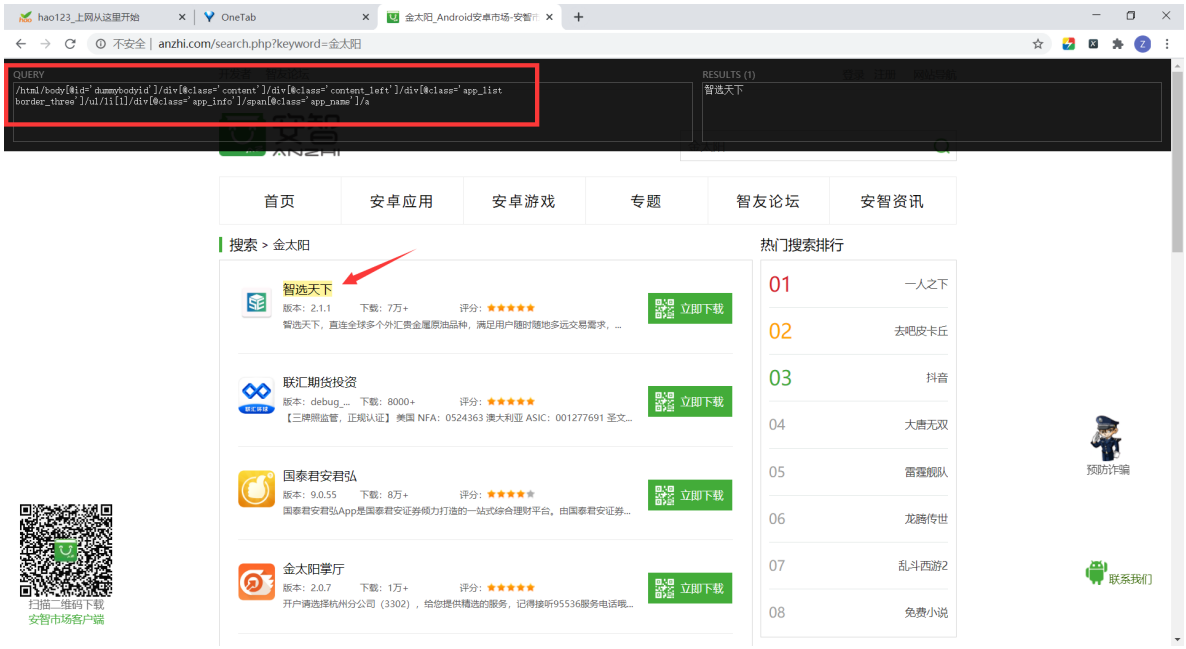
- 在要获取的搜索 URL 源码里面验证一下是否包含我们要爬取的信息

到这里，就可以开始根据模板（`ParseComponent类`）编写代码了，为啥要封装这个类呢？因为这些软件应用商店的结构都是一致的，搜索关键词 -> 搜索列表页 -> 各应用详情页，我们只要顺着这个过程爬取，即可，所以我把这个过程封装在了 `ParseCompoent` 和 `ParseComponentAjax` 中，用户只需要编写元素解析的代码，即可完成爬取

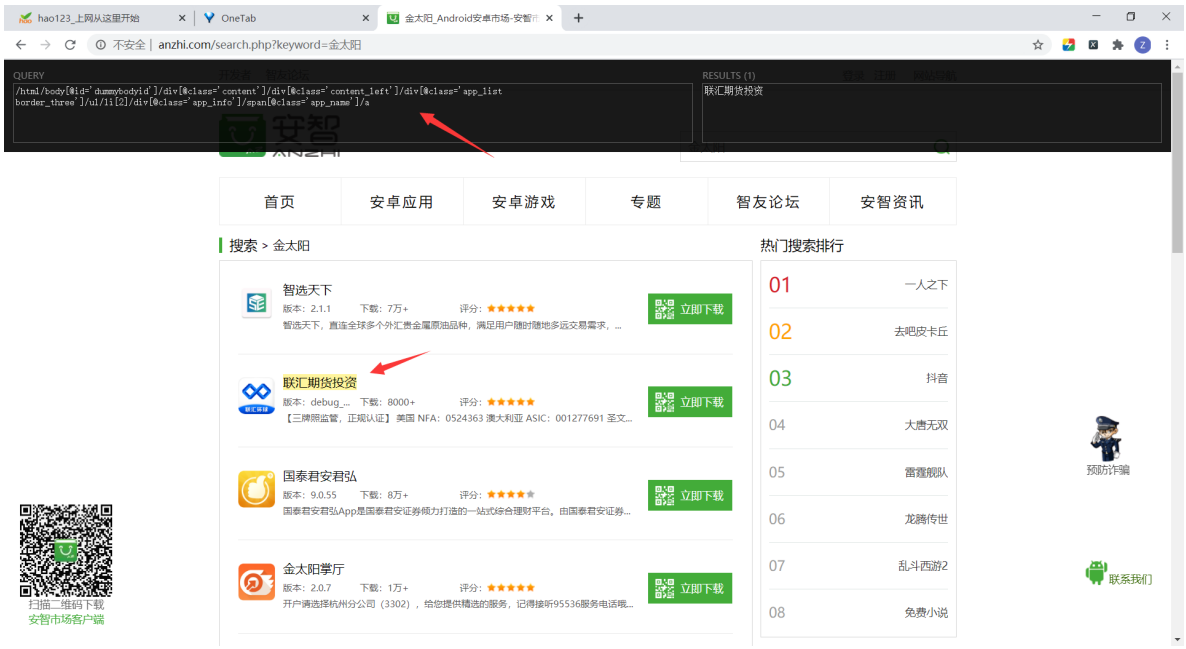
- 首先我们要创建一个类，继承自 `ParseCompoent`

4.1.2 找出需要遍历爬取的盒子模型

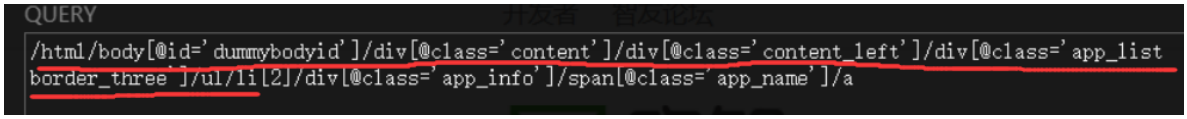
这里我们先装一个 `xpath helper` 插件，使用快捷键 `Ctrl + Shift + X` 启动插件，按一下 `Shift` 并点击搜索结果列表中的某个元素，如这里的软件名，观察 xpath query 的值



在按一下 `Shift` 点另一个应用的软件名，观察 xpath query 的值



- 比对两个 xpath query 的异同之处

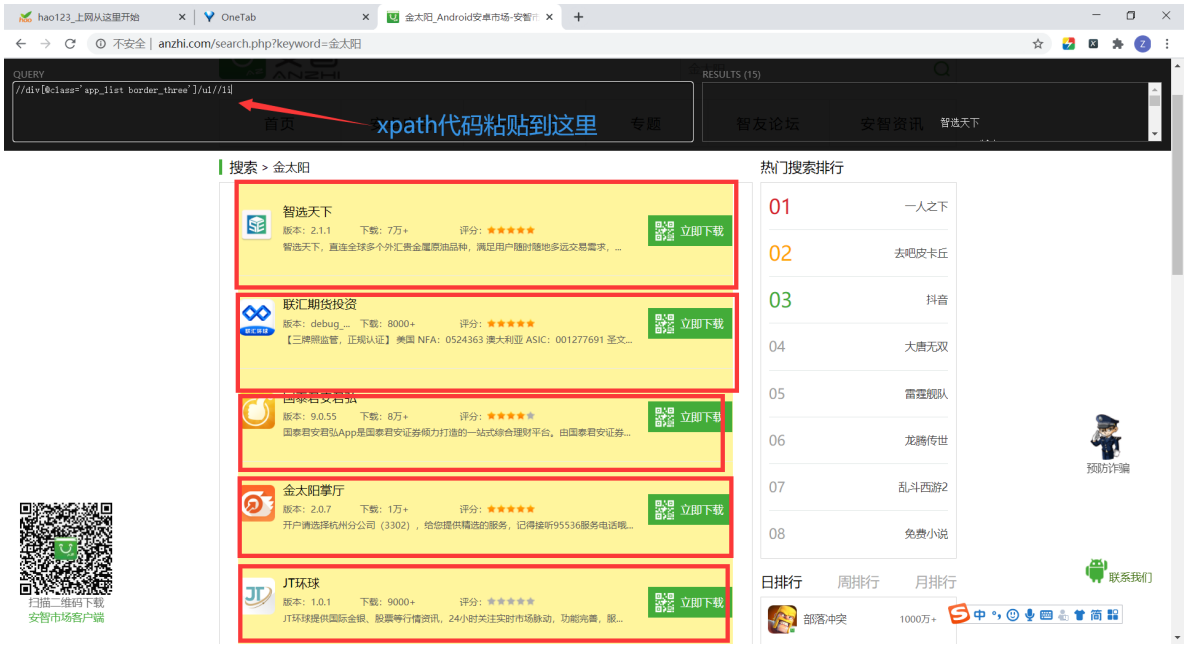


可以发现，只是 li 的索引值变了，这样我们就可以取出所有的盒子元素了对应的 xpath 路径了，这里姑且取

```
//div[@class='app_list border_three']/ul//li
```

- 在使用 xpath helper 插件验证一下

将这个 xpath 粘贴到 query xpath 的框中



这里显然每一个搜索列表的盒子模型都被标记了出来，证明我们的这个 xpath 表达式的值是正确的

4.1.3 代码编写

1）创建一个类，继承 ParseCompoent，并初始化

定义一下变量

- `self.url` 为抓包抓到的，搜索触发的 URL
- `self.n_page` 为要爬取的页数
- `self.name` 为要爬的软件市场的名字

2）重写 `get_app_list_elements` 方法 Override!

```
def get_app_list_elements(self, url) -> list:
    response = RqCompoent.get(url)
    selector = etree.HTML(response)
    lis = selector.xpath("//div[@class='app_list border_three']/ul//li")
    return lis
```

使用 xpath 找出所有要遍历获取信息的盒子模型列表，并返回这个列表

3）重写 `get_page_n_url` 方法

```
def get_page_n_url(self, n):
    if n == 1:
        return self.url
    page_n_url = self.url + "&page={}".format(n)
    return page_n_url
```

根据该网站翻页的规则，编写在第 n 页的时候，需要请求的该页的 URL 是怎样的

4）重写 `get_app_name` 和 `get_enter_url` 方法

```
@catch_error
def get_app_name(self, li):
    app_name = li.xpath("div[@class='app_info']/span[@class='app_name']/a/text()")
    return app_name
```

```
@catch_error
def get_enter_url(self, li):
    enter_url = li.xpath("div[@class='app_info']/span[@class='app_name']/a/@href")
    enter_url = self.judge_null(enter_url)
    enter_url = "http://www.anzhi.com" + enter_url
    return enter_url
```

根据在搜索结果页的每一个盒子模型，在该元素的基础上，继续使用 xpath 找出该盒子模型里面的应用名和详情页的 URL，并使用装饰器 `catch_error` 过滤错误（PS：我在调试程序的时候并没有使用这个装饰器，以为调试程序的时候一定要把能抓到的内容都抓到，将程序调通，装饰器是在全部编写完成之后，想要在不修改代码的情况下，为函数添加一点额外功能而编写的，因为如果一段一段的加 `try...except...` 实在是不够优雅😛，所以就用了装饰器，需要注意的是编写在类中的装饰器一定要记得传入 `self` 参数哦）

5）重写 get_update_time、get_author、get_download_url、get_version、get_img_address、get_app_intro 方法

根据你的需要，你觉得从详情页提取方便，还是从盒子模型列表中提取方便，就咋改，默认的话是从详情页提取这些字段的信息

4.2 扩展程序---增加需要爬取的字段

4.2.1 修改数据库

1）修改 Django app 对应的 models.py 文件

修改 Django 后台文件中，路径为 */home/ubuntu/appinfo/appinfo/spider/* 下的 `models.py` 文件，在 App 类里面根据需要增加字段，并指定该字段的数值类型

```python
class App(models.Model):
    appStore = models.CharField(max_length=255)
    appName = models.CharField(max_length=255)
    version = models.CharField(max_length=255)
    updateTime = models.DateTimeField()
    author = models.CharField(max_length=255)
    downloadUrl = models.CharField(max_length=255)
    icon = models.CharField(max_length=255)
    introduction = models.CharField(max_length=1500)
    inList = models.CharField(max_length=25)
    platform = models.CharField(max_length=25)
    insertTime = models.DateTimeField()
    keyword = models.CharField(max_length=25)
```

2）数据库迁移

在与 `manage.py` 相同的目录下，即 */home/ubuntu/appinfo* 路径下，执行命令

> - $ python manage.py makemigrations
> - 会让你做一个选择，要选择 1，然后随意输入一个值
> - $ python manage.py migrate

4.2.2 修改程序

此处以增加一个字段 **应用的软件包大小** 为例

- 首先在 `ParseCompoent` & `ParseCompoentAjax` 中加入一个抓取软件包大小的方法

```python
@catch_error
def get_app_name(self, inner_response, *args, **kwargs):
    app_name = li.xpath()
    return app_name
```

- 在 `loop_request` 里面修改代码

```python
def loop_request(self, lis, first_page=True, **kwargs):
    """循环请求"""

    if not lis:
        return
    for li in lis:
        enter_url = None
        enter_url = self.get_enter_url(li)  # 获取详情页url
        inner_response = RqCompoent.get(enter_url, **self.add_headers)
        if inner_response:
            self.inner_response = inner_response
            self.li = li

            app_name = None
            img_address = None
            app_intro = None

            app_name = self.get_app_name(li)  # 先获取一下app名字，对比关键字
            app_name = self.judge_null(app_name)
            app_name = self.field_strip(app_name)
            img_address = self.get_img_address(li)
            app_intro = self.get_app_intro(inner_response)
            fields = self.parse_app_info_page(inner_response)
            to_sink = [self.name, app_name, *fields, img_address, app_intro]
```

加入你要爬的字段的代码

```python
sql = "insert into spider_app(appStore, appName, version, updateTime, author,\
downloadUrl,icon, introduction, inList, platform, insertTime, keyword, enter_url)\
 values(%s, %s, %s, %s, %s, %s, %s, %s, '否', '安卓', %s, %s, %s)"
res = [*res, pymysql.escape_string(time.strftime("%Y/%m/%d", time.localtime())),\
 pymysql.escape_string(self.keyword), pymysql.escape_string(enter_url)]
try:
    self.cursor.execute(sql, res)
except:
    pass
```

加入新的 %s

加入新的字段变量，并用pymysql.escape_string处理

4.3 扩展请求头

有一些网站，在请求头不增加参数的情况下不会给你返回数据，因此需要扩展请求头

要扩展请求头只需要在重写的 `get_app_list_elements` 方法里面，在请求的时候传入扩展头的解包即可（列表，元组的解包是一个 `*` ，字典的解包是两个 `*` ）

```python
def get_app_list_elements(self, url) -> list:
    # 这兄弟要带上 Host 和 Cookie 才行
    extend_headers = {
        "Cookie": "UM_distinctid=1733179c56188c-04724
        "Host": "s.duote.com:8081",
        "Referer": "http://s.duote.com/",
    }
    response = RqCompoent.get(url, **extend_headers)
```

```python
@staticmethod
def get(url, app=False, *args, **kwargs):
    # 请求代理 && 随机获取请求头
    headers, proxies = return_headers_and_proxies(app)
    # 扩展请求头
    headers = {**headers, **kwargs}
```

5.0 程序维护

5.1 已经编写的爬虫程序的维护

对于软件应用商店这种网站来说，一般网页更新变化的概率不大，所以应该不需要短期进行维护，如果在导出到Excel 的文件中发现某个商城有很多空值，可以去官网比对一下是否网页更新了，如果更新了，按照 **4.1.3** 修改代码即可

5.2 配置文件和包的升级

注意，如果使用了 `apt-get update` 更新了安装的 Chrome，需要找到对应版本的 Chromedriver 与之对应，具体就是，在 http://npm.taobao.org/mirrors/chromedriver/ 这个网址找到对应于最新的 Chrome 的 chromedriver 的软件包，并替换 **192.168.50.173** 中 */home/ubuntu/chromedriver_linux64* 路径下的 chromedriver，然后用命令 `$ sudo chmod 777 chromedirver` 给予 chromedriver 可执行的权限

6.0 注意事项

6.1 权限问题

Django文件存放位置，不能在 /usr/下，且chromedriver不能在 /usr/下，且不能建立软连接，所以在用 selenium 程序时要指定一个在 /home 下的路径，并且里面的 chromedriver 要经过 `$ sudo chmod 777 chromedriver` 赋予权限

6.2 URL 编码

自己编写程序的时候，要注意该网站对应的 URL 编码是默认的 UTF-8 还是 GBK / GB2312，如果是别的编码的话，在 URL 编码方法 `urllib.parse.quote()` 里面加上一个参数 `encoding="GBK"` 即可，如下

```python
k = quote(keyword, encoding="GBK")
self.url = "http://s.duote.com:8081/search/softindex/?keywords=" + k
```