以太坊智能合约爬虫

冀甜甜

2020 年 12 月 8 日

1 固定时刻：针对当前这一时刻的静态爬虫

1.1 代码实现：

crawl_contracts.py。爬取当前最新的 500 条智能合约，这些合约都是经由以太坊验证过的。

1.2 基本思路：

(1) 以太坊网络上会在 https://cn.etherscan.com/contractsVerified 这个链接下展示最新的 500 条智能合约；

按照每页最大展示量（100 条智能合约），通过爬虫的方式实现对所有 500 条智能合约地址的爬取，总共需要通过对 5 个网页内容的解析。

这 5 个网页链接分别为：

```
urlList = ["https://cn.etherscan.com/contractsVerified/1?ps=100",
        "https://cn.etherscan.com/contractsVerified/2?ps=100",
        "https://cn.etherscan.com/contractsVerified/3?ps=100",
        "https://cn.etherscan.com/contractsVerified/4?ps=100",
        "https://cn.etherscan.com/contractsVerified/5?ps=100"]
```

(2) 读取 urlList 里的每一个 URL 网页里的智能合约地址：

```
for eachurl in urlList:
        ...
```

(3) 伪装成某种浏览器，防止被服务器拒绝服务:

```
        headers = {
          'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/
          537.36 (KHTML, like Gecko) Chrome/78.0.3904.87 Safari/537.36'}
```

(4) 对每一个 URL 网页进行请求：

```
        response = requests.get(url=eachurl, headers=headers, timeout=5)
```

(5) 转换成 UTF-8 编码

```
        response.encoding = response.apparent_encoding
```

(5) 爬虫解析

```
        soup = BeautifulSoup(response.text, "html.parser")
```

(6) 定位：特定字段下包含智能合约代码的 URL 地址

```
        targetDiv = soup.find_all('div', 'table-responsive mb-2 mb-md-0')
```

(7) 内容提取

```
        targetTBody = targetDiv[0].table.tbody
        for targetTR in targetTBody:
        # 获取每一行
        if targetTR.name == 'tr':
            # 获取一行所有列信息
            each_sc_url = "https://cn.etherscan.com" +
            targetTR.td.find('a', 'hash-tag text-truncate').attrs['href']
```

(8) 其他所需要的内容，根据具体的需要进行解析即可。

2 固定周期：针对昨天所有 Verified 智能合约的静态爬虫

2.1 代码实现：

crawl_contracts_yesterday.py 中的 get_yesterday_allVerifiedSC() 这一函数。

2.2 基本思路：

(1) 获取系统的当前日期, 得到昨天的日期。

```
    yesterdayDate = int(date.today().strftime('%Y%m%d')) - 1
```

(2) 获取解析网页中特定字段下, 智能合约的 Verified date, 该时间与昨天的日期对比，如果相同，则记录该智能合约的链接。

```
    # 这个字段下，包含智能合约代码的URL地址
    targetDiv = soup.find_all('div', 'table-responsive mb-2 mb-md-0')
    # 继续明确智能合约URL所在的地址
    targetTBody = targetDiv[0].table.tbody
    # print(type(targetTBody),len(targetTBody))
    for targetTR in targetTBody:
        if targetTR.name == 'tr':
            data = targetTR.find_all('td')
            # print(data)
            # Address = data[0].getText()
            # Name = data[1].getText()
            # Compiler = data[2].getText()
            # Version = data[3].getText()
            # Balance = data[4].getText()
            Verified_list = data[7].getText().split('/')
            Verified_datelist = datetime(int(Verified_list[2]),
                                         int(Verified_list[0]),
                                         int(Verified_list[1]))
            int_VerifiedDate = int(Verified_datelist.strftime('%Y%m%d'))
            # print(int_VerifiedDate)
            sub_url = targetTR.td.find(
                  'a', 'hash-tag text-truncate').attrs['href']
            if (yesterdayDate == int_VerifiedDate):
                sub_urlList.append(sub_url)
```

3 实时：以小时为单位进行更新的动态爬虫

3.1 代码实现:

分为两个部分。

(1) crawl_contracts_yesterday.py 中的 get_latest_hrs_VerifiedSC() 这一函数。

(2) crawl_contracts_update.py 中的 update_latest_hrs_VerifiedSC() 这一函数。

3.2 基本思路:

(1) 获取当前时刻的所有 500 个智能合约的链接地址.

```
1        # 爬虫解析
2        soupX = BeautifulSoup(responseX.text, "html.parser")
3        target_subUrl_listC = soupX.find_all('a', 'hash-tag text-truncate')
4        for target_subUrl in target_subUrl_listC:
5              sub_urlList.append(target_subUrl.attrs['href'])
```

(2) （默认以太坊中 verfied 智能合约的顺序与这些智能合约被创建的时间顺序是一致的，即创建时间是从最近到最后的时间排序的。）使用二分法从 500 个智能合约的链接中选择创建时间 <=24h(设定的动态时间周期) 的边界智能合约，将最近时间到该边界智能合约的所有智能合约记录为该动态时间周期内的所有最新的智能合约。

```
1        # 使用二分法：从500个sub_url中筛选
2        mid_value = get_midValue_byBinaryTree(sub_urlList, headers, latest_hrs
3        print(mid_value)
4        mid_sc_url = 'https://cn.etherscan.com' + sub_urlList[mid_value]
5        if parse_time_from_sc_code_url(mid_sc_url,headers) <= latest_hrs:
6            latest_hrs_sc_urls += sub_urlList[:mid_value + 1]
7        else:
8            latest_hrs_sc_urls += sub_urlList[:mid_value]
```

get_midValue_byBinaryTree() 为具体的二分函数：

```
1        #验证发现，最新的500个的排序与创建时间的先后并不存在直接的关系，
```

```
    #即500个合约的创建时间可能是无序的。
    # 所以这个函数应该是用不上的。
    # - 但现在的做法是：默认创建时间与Verified的时间是一致的，即按顺序排列
    # 但是更新的次数越多之后，最后会趋于吻合，所以还是采用这种方式！
    def get_midValue_byBinaryTree(sub_urlList, headers, latest_hrs=24):
        mid_value = 0
        low = 0
        height = len(sub_urlList) - 1
        while low < height:
            mid_value = (low + height) // 2
            mid_sc_url = 'https://cn.etherscan.com' +
                         sub_urlList[mid_value]
            if parse_time_from_sc_code_url(mid_sc_url,headers)
                                    <= latest_hrs:  # 按顺序放进list中的
                low = mid_value + 1
            else:
                height = mid_value - 1
        return mid_value
```

(3) 以 1h 为单位对爬取的最近 24h 时间周期内的智能合约进行动态更新。

根据 https://cn.etherscan.com/chart/verified-contracts 该链接下的统计显示，以太坊网络中每天更新的智能合约总量一般维持了 200-400 之间。平均来看，单位小时内更新的智能合约一般不会超过 16 个；且根据我们的观察，每个小时内更新智能合约的速度确实比较慢。因此，在新合约的增加方面，我们只需要解析一个包含 25 或 50 个最新智能合约的网页即可。这里，我们使用的网页链接为：

```
    update_url = "https://cn.etherscan.com/contractsVerified/1?ps=50"
```

更新分为四步走：

step1: 获取 update_url 链接下的所有智能合约的链接地址，将这些地址与前一个小时内获取到的 24h 周期智能合约进行合并（按最近到最后的顺序）；

step2：合并后的智能合约存在重复，在这一步需要进行合约去重。

step3: 对不在最近 24h 周期内的旧合约进行删除。

step4：step1-step3 的更新时间远远小于 1h, 所以上述 3 个步骤的耗时可以忽略。睡眠 1h，之后进行合约更新，即重复执行 step1-3。

核心代码展示如下：

```
    latest_hrs_sc_urls_list = load_json('latest_hrs_sc_urls.json')
    while True:
        """
        # 这里可以设置按键后停止更新
        """
        print("!!!!!!!!!!!!!!睡眠结束,启动更新!!!!!!!!!!!!!!!!!!!!")
        #定位最新的50个
        sub_50urlsList = []

        ...

        soupX = BeautifulSoup(responseX.text, "html.parser")
        target_subUrl_listC = soupX.find_all(
                    'a', 'hash-tag text-truncate')
        for target_subUrl in target_subUrl_listC:
            sub_50urlsList.append(target_subUrl.attrs['href'])
        update_url_list = sub_50urlsList+latest_hrs_sc_urls_list
        # 去重
        latest_hrs_sc_urls_list = sorted(set(update_url_list),
                                         key=update_url_list.index)
        for i in range(1,11):
            lastsc_code_url = 'https://cn.etherscan.com' +
                                  latest_hrs_sc_urls_list[-i]
            if (parse_time_from_sc_code_url(
                    lastsc_code_url, headers)) <= latest_hrs:
                break
            else:
                latest_hrs_sc_urls_list.pop(-i)
        save_json('latest_hrs_sc_urls.json', latest_hrs_sc_urls_list)
        print("————————————进入睡眠(1h)————————————————")
```

```
31              time.sleep(3600) #60*60s
```

PS: 这里我们不再对如何进行网页解析和请求进行具体说明。

3.3 真正正确、精准的动态周期获取

由于在 etherscan 上展示的 verified 智能合约的顺序与其真正被创建的时间顺序是不太一样的，所以遍历 500 个智能合约的创建网页，获取 500 个具体的创建时间，并与 24h 做比较才是最准确的方法。然而，这一方法虽然是准确的，但是 500 个网页的解析速度非常慢。既会影响首次的获取，也需要进行智能合约链接地址与创建时间的存储，相比二分法，更浪费计算资源和存储资源；不仅如此，后续的更新逻辑更为复杂，而且，设计出的复杂逻辑，可能会因为单位小时内更新的智能合约和需要删除的智能合约的总量过多，而导致更新时间无法被忽略，甚至可能一次更新时间就可能超过 1h，因此极有可能单位小时的更新是无法实现的。

```
1       #做500次循环 ---- 太慢
2       for each_suburl in sub_urlList:
3           each_url = 'https://cn.etherscan.com' + each_suburl
4           each_vtime = parse_time_from_sc_code_url(each_url)
5           if each_vtime <= latest_hrs:
6               latest_hrs_sc_urls[each_suburl] = each_vtime
7       print('len of latest_hrs_sc_urls is: ', len(latest_hrs_sc_urls))
```

所以，这种方法最终没有被采纳。而是继续选用二分策略，随着更新频次的增加，最终实现获取到的智能合约与最近一个动态时间周期内的智能合约是越来越趋于相同的。