爬虫初始

爬虫相关介绍

- 什么是爬虫
  - 就是编写程序，模拟浏览器上网，让其去互联网中抓取数据的过程
    - 模拟：
      - 浏览器本身就是一个纯天然的爬虫工具，爬虫相关的模块都是基于浏览器为基础开发出来的。
      - 注意：日后只要是你的爬虫程序没有爬取到你想要的数据，只有一个原因：
        - 就是你的爬虫程序模拟的力度不够！
    - 抓取：
      - 抓取网页数据分两种情况：
        - 将一个页面所有的数据抓取到
        - 将页面中局部的数据抓取到
- 爬虫在应用场景的分类
  - 通用爬虫：
    - 将一个页面中所有的数据获取。
    - 大部分的搜索引擎中应用比较多。

    - 聚焦爬虫
        - 将页面中局部的指定的数据进行提取/抓取
        - 注意：聚焦爬虫一定是建立在通用爬虫的基础之上实现。
    - 功能爬虫
        - 通过浏览器或者app自动化的操作，实现相关的网页或者app自动化的操作。代替人工在网页或者手机软件中自动执行相关的行为动作。
        - 批量点赞，批量评论，刷单,秒杀.....
    - 增量式爬虫
        - 用来监测网站数据更新的情况。以便爬取网站最新更新出来的数据！
    - 分布式爬虫
        - 可以对网站所有的资源使用分布式机群进行分布和联合的数据爬取
- 爬虫的矛与盾（重点）
    - 反爬机制：对应门户网站，网站可以指定相关的机制阻止爬虫对其网站数据的采集
    - 反反爬策略：对应爬虫程序，爬虫可以制定相关的策略将网站的反爬机制破解，从而爬取到指定的数据
- 盗亦有道的君子协议robots
    - Robots协议（也称为爬虫协议、机器人协议等）的全称是“网络爬虫排除标准”（Robots ExclusionProtocol），

网站通过Robots协议告诉搜索引擎哪些页面可以抓取，哪些页面不能抓取.
    - 注意，这个协议的存在更多的是需要网络爬虫去遵守，而起不到防止爬虫的功能。

爬虫合法性探究

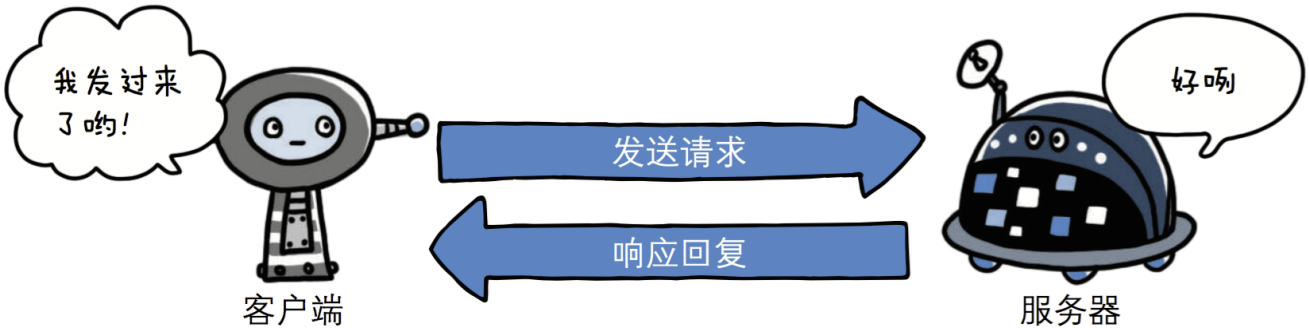
- **爬虫作为一种计算机技术就决定了它的中立性，因此爬虫本身在法律上并不被禁止，但是利用爬虫技术获取数据这一行为是具有违法甚至是犯罪的风险的。**所谓具体问题具体分析，正如水果刀本身在法律上并不被禁止使用，但是用来捅人，就不被法律所容忍了。
- 或者我们可以这么理解：爬虫是用来批量获得网页上的公开信息的，也就是前端显示的数据信息。因此，既然本身就是公开信息，其实就像浏览器一样，浏览器解析并显示了页面内容，爬虫也是一样，只不过爬虫会批量下载而已，所以是合法的。不合法的情况就是配合爬虫，利用黑客技术攻击网站后台，窃取后台数据（比如用户数据等）。
- 举个例子：像谷歌这样的搜索引擎爬虫，每隔几天对全网的网页扫一遍，供大家查阅，各个被扫的网站大都很开心。这种就被定义为“善意爬虫”。但是像抢票软件这样的爬虫，对着 12306 每秒钟恨不得撸几万次，铁总并不觉得很开心，这种就被定义为“恶意爬虫”。
- **爬虫所带来风险主要体现在以下3个方面：**

    - 1、违反网站意愿，例如网站采取反爬措施后，强行突破其反爬措施；
    - 2、爬虫干扰了被访问网站的正常运营；
    - 3、爬虫抓取了受到法律保护的特定类型的数据或信息。
- **那么作为爬虫开发者，如何在使用爬虫时避免进局子的厄运呢?**
    - 1、严格遵守网站设置的robots协议；
    - 2、在规避反爬虫措施的同时，需要优化自己的代码，避免干扰被访问网站的正常运行；
    - 3、在使用、传播抓取到的信息时，应审查所抓取的内容，如发现属于用户的个人信息、隐私或者他人的商业秘密的，应及时停止并删除。
- 总结：
    - 可以说在我们身边的网络上已经密密麻麻爬满了各种网络爬虫，它们善恶不同，各怀心思。而越是每个人切身利益所在的地方，就越是爬满了爬虫。**所以爬虫是趋利的，它们永远会向有利益的地方爬行。**技术本身是无罪的，问题往往出在人无限的欲望上。因此爬虫开发者的道德自持和企业经营者的良知才是避免触碰法律底线的根本所在。

爬虫原理剖析(重点)

爬虫的基本原理：模拟浏览器发请求的模式，进行网络数据的爬取。

- 服务器：存储数据的一方（百度，京东...）
- 客户端：请求数据的一方（浏览器）



图：请求必定由客户端发出，而服务器端回复响应

- 网页基本结构
  - 浏览器中显示的每一个网页都是由html标记语言写成的。html是由各种标签组成的。每种形式的标签都可以代表网页中的某种结构样式。
  - 爬虫爬取的内容就存储在指定的标签之中。

```
<html>
<head>
    <meta charset="UTF-8">
    <title>测试网页</title>
</head>
<body>
   我是网页主体内容！！！ 下面是我展示的一张好看的图片：
   <img
src='https://pic.netbian.com/uploads/allimg/230
308/001555-1678205755f136.jpg'>
   <div>马斯克新能源</div>
</body>
</html>
```

较为复杂的页面结构

```
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    <form action="#" method="post">
        <label>用户名：</label>
        <input type="text" name="username"/>
<br/>
        <label>密码：</label>
```

```
        <input tyåpe="password"
name="password"/><br/>
        <label>爱好: </label>
        <input type="checkbox" name="hobby"
value="lanqiu">篮球
        <input type="checkbox" name="hobby"
value="zuqiu">足球
        <input type="checkbox" name="hobby"
value="qumao">羽毛球
        <input type="checkbox" name="hobby"
value="gaoer">高尔夫
        <br/>
        <label>性别: </label>
        <input type="radio" name="sex"
value="male">男
        <input type="radio" name="sex"
value="female">女
        <br/>
        <label>生日: </label>
        <input type="date" name="birthday"><br>
        <label>籍贯: </label>
        <select name="jiguan">
            <option>请选择籍贯</option>
            <option value="hebei">河北</option>
            <option value="anhui">安徽</option>
            <option value="liaoning">辽宁
</option>
```

```
        </select>
        <br>
        <label>自我介绍：</label>
        <textarea cols="30" rows="10"
placeholder="请描述" name="self"></textarea>
        <br>
        <input type="submit" value="提交">
        <input type="reset" value="重置">
    </form>
</body>
</html>
```

- 浏览器抓包工具使用
  - 浏览器自带的抓包工具NetWord可以将客户端和服务器端进行的请求和响应过程进行查看

requests基础操作(代码实操重点！！！)

- 基本介绍
  - requests就是爬虫中一个基于网络请求的模块。
  - 作用：模拟浏览器上网的。
  - urllib模块就是一个老版的requests模块，现在没人用urllib
- 环境安装
  - pip install requests

- 编码流程
  - 指定url（好比打开浏览器输入网址）
  - 发起请求（好比是按下回车）
  - 获取响应数据（从指定url中爬取到的数据）
  - 持久化存储
- 案例应用
  - 搜狗首页数据采集
    - https://www.sogou.com/

```
import requests

#1,指定url
url = 'https://www.sogou.com/'

#2,发起请求（只要在浏览器地址栏输入网址按下回车，发起的一定get请求）
#url为get方法的第一个参数，表示根据指定的url发起get请求
#get方法会返回一个响应对象
response = requests.get(url=url) #get使用来使用requests模块发起get请求

#3.获取响应数据/爬取到的数据
page_text = response.text #text属性使用来返回字符串形式的响应数据
```

```
# print(page_text)

#4.持久化存储
with open('./sogou.html','w') as fp:
    fp.write(page_text)
print('数据爬取存储成功！')
```

- 简易的网页采集器
  - 注意：在浏览器的地址栏中网址，网址？后面的内容就是请求的参数（请求参数）
    - https://www.sogou.com/web?query=jay
    - query=jay就是请求的参数
      - 请求参数就是客户端发送给服务端的数据
  - https://www.sogou.com/

```
import requests

#请求参数动态化
keyword = input('请输入关键字:')
#稍后想要把该字典作为请求参数
pram = {
    'query':keyword, #只存在一个键值对（存在一组请求参数）
}
#1.指定url
```

```
url = 'https://www.sogou.com/web' #需要将请求参数去除
#2.发起请求
#params参数就是用来在请求时携带指定的请求参数
response =
requests.get(url=url,params=pram)

#3.获取响应数据
page_text = response.text

#4.持久化存储
fileName = keyword + '.html'
with open(fileName,'w') as fp:
    fp.write(page_text)


#出问题：没有爬取到我们想要的数据？原因是因为遇到反爬机制
```

- 分析该网站的反爬机制：
  - 从爬取到的内容中提取到了一个关键信息：网站检测到了异常的访问请求
    - 异常的访问请求：通过程序发起的请求
    - 正常访问请求：通过浏览器发起的请求
  - 网站如何可以监测请求是不是通过浏览器发起的呢？

    - 是通过请求的一个头信息：user-agent
    - user-agent：请求载体的身份标识
- 破解方式（UA伪装）：伪装请求载体的身份标识
    - 该反爬机制是一种最常见最通用的，也就是说绝大数网站都会携带该反爬机制，因此日后写爬虫程序，默认带上UA伪装操作。

```
import requests

#请求参数动态化
keyword = input('请输入关键字:')
#稍后想要把该字典作为请求参数
pram = {
    'query':keyword, #只存在一个键值对（存在一组请求参数）
}
#1.指定url
url = 'https://www.sogou.com/web' #需要将请求参数去除
#2.发起请求
head = { #存放需要伪装的头信息
    'User-Agent':'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/97.0.4692.71 Safari/537.36'
}
```

```
#通过headers参数进行了头信息的伪装
response =
requests.get(url=url,params=pram,headers=
head)

#3.获取响应数据
page_text = response.text

#4.持久化存储
fileName = keyword + '.html'
with open(fileName,'w') as fp:
    fp.write(page_text)
```

- 豆瓣电影
  - https://movie.douban.com/typerank?type_name=%E7%88%B1%E6%83%85&type=13&interval_id=100:90&action=
  - 爬取电影的详情数据
  - 对网站进行分析：
    - 1.在抓包工具中先定位到和浏览器地址栏的网址一样的数据包
    - 2.查看开发者工具中，定位到的数据包中的response这一项，该项中存放的就是对定位到数据包的url发起请求，请求到的数据。

- 3.在response选项卡中查看是否存在我们想要爬取的数据：
  - 如何检测是否存在我们想要爬取的数据呢?
    - 局部搜索：将你想要爬取的局部数据，在response进行搜索，查看是否可以搜索到。
      - 搜索到了：
        - 可以直接对该数据包的url发起请求获取你想要的数据即可
      - 搜索不到：
        - 说明你想要的数据是【动态加载数据】
        - 什么是动态加载数据?
          - 特指，不是通过浏览器地址栏的请求请求到的数据，就是动态加载数据。同理，动态加载数据一定是通过其他的请求请求到的。
        - 如何获取动态加载数据?
          - 基于抓包工具进行全局搜索
            - 鼠标点击任意的数据包，然后按下cotrl+f打开全局搜索框，搜索局部你想要的数据，即可定位到包含搜索数据的指定数据包。

- 从指定数据包中就可以提取出：
  - url：https://movie.douban.com/j/chart/top_list
  - 请求方式：get
  - 请求参数：type=13&interval_id=100%3A90&action=&start=0&limit=1

```
import requests
head = { #存放需要伪装的头信息
    'User-Agent':'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/97.0.4692.71 Safari/537.36'
}
pram = {
    "type": "13",
    "interval_id": "100:90",
    "action": "",
    "start": "0",
    "limit": "20",
}
```

```
url =
'https://movie.douban.com/j/char
t/top_list'
response =
requests.get(url=url,headers=hea
d,params=pram)
#获取响应数据
#json()可以将获取到的json格式的字符
串进行反序列化
page_text = response.json()
fp = open('./douban.txt','w')
for dic in page_text:
    title = dic['title']
    score = dic['score']

 fp.write(title+':'+score+'\n')
    print(title,'爬虫保存成功! ')
```

- 肯德基
  - http://www.kfc.com.cn/kfccda/index.aspx
    - 将餐厅的位置信息进行数据爬取

```
import requests
head = { #存放需要伪装的头信息
```

```
    'User-Agent':'Mozilla/5.0
(Macintosh; Intel Mac OS X 10_15_7)
AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/97.0.4692.71 Safari/537.36'
}
#post请求的请求参数
data = {
    "cname": "",
    "pid": "",
    "keyword": "天津",
    "pageIndex": "1",
    "pageSize": "10",
}
#在抓包工具中：Form Data存放的是post请求的
请求参数，而Query String中存放的是get请求的
请求参数
url =
'http://www.kfc.com.cn/kfccda/ashx/Get
StoreList.ashx?op=keyword'
#在post请求中，处理请求参数的是data这个参数
不是params
response =
requests.post(url=url,headers=head,dat
a=data)
#将响应数据进行反序列化
page_text = response.json()
for dic in page_text['Table1']:
```

```
    name = dic['storeName']
    addr = dic['addressDetail']
    print(name,addr)
```