CH1

Web Scraping With Python

前言

第一章、第一个网络爬虫

> 当你步入网络爬虫世界的殿堂，你会开始赞叹浏览器为我们所做的一切。一个没有HTML格式、没有CSS样式表、没有JavaScript脚本、没有图片渲染的网站乍看来是如此简陋，但是本书中，我们会去了解在没有浏览器解析的情况下如何去组织、理解数据。
> 本章节将从向服务器发送一个简单的GET请求开始，进而探讨HTML网页结构，同时对页面上的数据进行一些简单处理，提取出我们感兴趣的某些关键数据。

1、网络连接

如果你对计算机网络、网络安全等知识了解不多，那么互联网对你而言可能有些神秘。本书不对这些知识点进行探讨，实际上，当我们打开浏览器输入http://google.com的时候，网路就在背后默默的工作，对于大多数用户而言，关注网络运行的结果比关注网络是如何运行更加重要。

但是，网络爬虫要求我们剥开网络的神秘面纱。不仅仅是从浏览器级别（如浏览器如何去解析HTML、CSS与JavaScript），更重要的在于网络的连接层面。

为了给读者一些基本的关于浏览器是如何获取数据的概念，我们给出如下的这个例子：Alice是一个网络服务器(Web Server)，Bob使用一台计算机连接到Alice的服务器上获取某些数据，当一台机器想要和另外一台机器通信的时候，他们之间的流程如下：
1.Bob的电脑以高低电平的形式向Alice服务器发送0、1二进制流，这些二进制流以某种特定的结构组织形成消息比特流，该比特流包含了消息头、消息体、消息内容等，在消息头中，包含了Bob所在网络路由器的MAC地址、Alice服务器的IP地址，消息体则包含了向Alice发送的消息请求。
2.Bob所在网络的路由器接受一系列的0、1二进制流，并将它们转化为包，这个包中包含了Bob自己的MAC地址和Alice的IP地址等信息，路由器又在这个包中加入自己的IP地址作为消息发送源，最后通过网络将这个包发送出去。
3.Bob的这个数据包在网络中几经流转，最终终于到达了Alice服务器的所在。
4.Alice服务器分析目的IP为自己，于是接收这个包
5.Alice服务器解读这个包中头部的目的端口(网络应用的端口一般都是80，我们可以将端口理解为房间号，而将IP地址理解为街道号)，并根据端口号将之发送给相应的应用程序，这边即发送给网络服务应用。
6.网络服务应用从网络服务器处理程序中获取数据流，这些数据的内容大致如下：
这是一个GET请求
请求的数据是：index.html
7、网络服务器定位对应的HTML文件，并将这个文件绑定到一个新的包中，以相同的方式通过Alice所在的网络路由器将这个包回传给Bob。
哒哒！以上就是互联网运行的基本流程！

那么在这个数据交换的过程中，浏览器运行于那个阶段呢？答案是任何阶段都不是，实际上，浏览器直到1990年才由Nexus研发出来（？此段翻译不确定）。

浏览器是一个伟大的发明，它能够创建信息包、发送或接收信息包、解析接收到的数据并以音频、视频、图片、文本等形式展现出来，但是浏览器也是由代码编写而成的，也就意味着说，我们可以在代码层级将浏览器拆分成为基本的功能模块，我们可以对这些基本的模块进行重写、复用以实现我们的需求。浏览器可以通知处理器将数据发送给电脑上的有线或

者无线网络连接的接口，但是许多编程语言都有类似的库都可以实现这个功能。

在Python中，我们可以如下实现一个简单的请求：

```
from urllib.request import urlopen
html = urlopen("http://pythonscraping.com/pages/page1.html")
print(html.read())
```

我们将这段代码保存为`scrapetest.py`，然后在控制台下运行这段代码：

```
$python scrapetest.py
```

如果您的电脑中也安装有版本为2.X的Python，而您是在Python 3.X的环境下编写的代码，那么在调用时需要明确指定版本：

```
$python3 scrapetest.py
```

上述代码会返回http://bit.ly/1QjYgcd这个网页完整的HTML代码，更准确的说，是返回page1.html这个HTML页面的代码，这个页面存储在以http://pythonscraping.com为域名的网络服务器中。

二者的区别在哪呢？许多现代的网页都包含许多资源文件，这些资源文件包含图片文件、JavaScript文件、CSS文件、以及其他网站上连接的文件，当浏览器访问到一个节点如`<img src="cuteKitten.jpg">`的时候，浏览器会意识到需要向服务器发送额外的一个请求来获取这个图片文件并在网页中进行展示，然而我们的Python爬虫并没有请求这些多媒体文件的逻辑结构，它只能读取我们请求到的HTML文件。

它是如何实现的呢？很简单，我们来看这样一段易读的英文代码：

```
from urllib.request import urlopen
```

这段代码的含义是：找到request模块并导入其中的urlopen函数

> 使用urllib还是urllob2？
> 如果你在Python 2.x环境下使用过urllib2这个库，那么你可能会注意到urllib和urllib2之间已经发生了一些变化，在Python 3.x中urllib2被重命名为urllib，并且被切分成几个子模块：urllib.request,urllib.parse,urllib.error。在新的urllib库中，虽然函数的名称依然保持不变，但是我们必须注意这些函数已经被转移到子模块中。

`urllib`是一个标准的Python库文件(意味着你不需要额外的去安装这个库来运行你的程序)，这个库包含诸多函数，如：请求网络数据、处理cookie、交换如报文头和user agent等元数据等。在本书中，我们会频繁的使用urllib这个库，所以我们建议在使用之前阅读一下这个库的开发文档 (http://bit.ly/1FncvYE)。

urlopen用来打开一个远程网络连接对象并读取连接内容，这个库是一个相当通用的库(它能够相当优雅的读取HTML文件、图片文件以及其他文件流)，我们会在本书中频繁的使用这个库。

2、BeautifulSoup介绍

> 体面汤，浓又黄，
> 盛在锅里不会凉！
> 说什么山珍海味，哪儿有这么样儿香。
> 半夜起来喝面汤，体面汤！

BeaufitulSoup库以《爱丽丝梦游仙境》中Lewis Carroll的同名诗歌命名，在这个故事中，由一位名叫Mock Turtle的角色大声唱诵。

如同它的神奇名字一样，BeautifulSoup致力于通过以一种易于理解、转换的XML对象将纷繁杂乱的网络内容进行合理的组合，并最终呈现给用户。

安装BeautifulSoup

BeautifulSoup不是一个Python内置的系统库，所以使用之前必须安装。本书中使用的是BeautifulSoup 4（即BS4库），完整的安装指导可以在Grummy.com中找到，下面仅对Linux下进行简单的安装说明：
在Linux命令行下：

```
$ sudo apt-get install python-bs4
```

在Mac控制台下：

```
$ sudo easy_install pip
$ pip install beautifulsoup4
```

第一句用于安装一个名为pip的Python包，用这个包可以方便的装bs4以及其他软件包。

再次提醒！如果你的电脑中同时安装了Python 2.x和Python 3.x版本，如果想要在Python 3.x环境下运行代码，那么你需要指明Python的版本：

```
$ python3 myScript.py
```

同样的，在安装包的过程中也需要指明Python的版本，否则包文件会默认安装在Python 2.x版本下，而非Python 3.x版本中

```
$ sudo python3 setup.py install
```

如果你使用pip，则可以如下安装：

```
$ pip install beautfulsoup4
```

在Window环境下安装则与上述流程不同，你需要下载最新的BeautifulSoup4发行包(同时也别忘记在Windows中安装Python环境)，在控制台进入Python环境，将下载下来的BeautifulSoup4解压，在控制台定位到解压目录，运行：

```
>python setup.py install
```

完成上述操作之后，BeautifulSoup就安装完成，此时你就可以在你的Python命令行下验证：

```
  $ python
  > from bs4 import BeautifulSoup
```

如果控制台没有错误输出，表明BS库已经完全安装完成。

另外，在Windows环境中有pip的.exe安装包，你可以安装之后使用pip命令来安装或者管理包：

$ pip install beautifulsoup4

> 用虚拟环境隔离库文件
> 如果你有多个Python项目文件，为了防止各个库文件之间可能发生的冲突，你可能需要一种将你的工程文件同特定库文件绑定在一起独立存放的功能，那么你可以安装Python
> Virtual Environment来轻松的隔离与管理你的项目。
> 在没有虚拟环境的情况下安装库的文件是全局性的，这通常需要安装者拥有管理员权限(Windows环境)或者root权限(Linux环境)，此时的库对所有用户、所有工程都是可见的，而你可以创建一个虚拟环境：
> $ virtualenv scrapingEnv 上述指令创建一个名为scrapingEnv的虚拟环境，你可以调用如下指令激活该环境：
> `$ cd scrapingEnv/`
> `$ source bin/activate`
> 当虚拟环境激活后，你会注意到控制台的提示符号前的名称变成了这个虚拟环境的名称，提示你当前正在虚拟环境下，此时你安装的所有库文件都是安装在这个虚拟环境中。
> 我们可以将BeautifulSoup安装在这个虚拟目录中，例如：
> (scrapingEnv)ryan$pip install beautifulsoup(scrapingEnv)ryan$ python
> from bs4 import BeautifulSoup
> 我们可以使用deactivate命令断开这个虚拟环境，断开虚拟环境后，所有安装在虚拟环境下的库都无法访问到了：
> (scrapingEnv)ryan$deactivateryan$ python
> from bs4 import BeautifulSoup
> Traceback (most recent call last):
> File “”, line 1, in
> ImportError: No module named ‘bs4’
> 保持工程之间库的独立性还有助于打包整个工程目录并发送给其他用户，只要其他用户的电脑中安装有和你开发环境相同的Python版本，那么他们就能直接运行你的工程文件而不需要额外再去安装工程需要的库文件。
> 在本书中，我们不会特地说明所有的工程都是安装在虚拟环境中，但是您需要牢记在建立项目时使用虚拟环境。

BeautifulSoup**使用**

BeautifulSoup库中最常使用到的对象就是BeautifulSoup对象，我们将本章节开始的request代码修改如下：

```
from urllib.request import urlopen
from bs4 import BeautifulSoup
html = urlopen("http://www.pythonscraping.com/exercises/exercise1.html")
bsObj = BeautifulSoup(html.read());
print(bsObj.h1)
```

上述代码的输出为：

```
<h1>An  Interesting Title</h1>
```

如前文所述，我们引用了urlopen库，调用了html.read()方法来获取网页的HTML内容，该内容随即被转化成BeautifulSoup对象，其结构如下：

```
html    →   <html><head>…</head><body>…</body></html>
head    →   <head><title>A  Useful  Page<title></head>
    title   →   <title>A    Useful  Page</title>
body    →   <body><h1>An    Int…</h1><div>Lorem ip…</div></body>
    h1  →   <h1>An  Interesting Title</h1>
    div →   <div>Lorem  Ipsum   dolor…</div>
```

我们提取的`<h1>`节点在我们的BeautifulSoup对象结构中嵌套在第二层，但是我们可以在BeautifulSoup对象结构中直接获取这个节点：

```
bsObj.h1
```

事实上，如下的所有代码都能返回相同的结果

```
bsObj.html.body.h1
bsObj.body.h1
bsObj.html.h1
```

我们希望能够通过这个小小的例子让你初窥BeautifulSoup的简洁与强大。实际上在任何一个HTML文件中，只要某个内容是被某个结构化的tag所标注的，那么这段内容就可以被提取出来，在第三章中，我们会探讨一些更深层次、更复杂的BeautifulSoup函数调用，同时我们也会探讨正则表达式和如何将之与BeautufulSoup库相结合来更好的从网页中提取信息。

3、建立可靠的链接

网络世界是一个纷繁复杂的世界，其中的数据可能会是非标准结构、网站可能会被关闭、标签忘记添加封闭符号等，在运行网络爬虫的过程中最让人失望的莫过于你将爬虫挂机运行，当你第二天回来时发现，与你设想的数据库中存满了你想要爬取的数据不同，因为你的爬虫遇到了某些网络错误，导致你离开没多久他就停止了运行，在这种情况下你可能会抱怨网站编程人员，然而真正应该受到批判的应该是你自己，因为你没有考虑到例外情况的发生。

以我们爬虫的第一行为例来说明应该如何处理可能发生的异常情况：

```
html = urlopen("http://www.pythonscraping.com/exercises/exercise1.html")
```

在这个过程中可能发生的异常主要有以下两种：

1、在服务器中无法找到这个网页（或者在获取的过程中发生了异常）

2、无法定位到服务器

在第一种情况中，一般会返回一个HTTP错误，这个错误可能是“404无法找到页面”、“500服务器内部错误”等等，所有的这些情况下，urlopen函数都会抛出HTTPError异常信息，我们可以通过如下的方式获取到这块信息：

```
try:
    html = urlopen("http://www.pythonscraping.com/exercises/exercise1.html")
except HTTPError as e:
    print(e)
    #return null, break, or do some other "Plan B"
else:
    #program continues. Note: If you return or break in the
    #exception catch, you do not need to use the "else" statement
```

当爬虫接收到一个返回的HTTP异常码的时候，我们的爬虫会在控制台打印这个异常信息，并停止运行在else中的的代码。

如果因为服务器已经关闭或者网站url输入错误导致服务器无法找到时，urlopen就会返回一个None对象，这个None对象与其他编程语言中的null对象类似，我们可以添加一个检查来判断返回值是否为None:

```
if html is None:
    print("URL is not found")
else:
    #program continues
```

当然，即使是在网页文件能够从服务器成功读取的情况下，可能依然会存在某些问题导致返回的数据不是我们期望的数据，所以你在通过BeautifulSoup对象获取某个tag的数据的时候，千万记得添加检查方法以确定这个tag是存在并且有效的，BeautifulSoup在访问一个不存在的对象时会返回None对象，如果你尝试访问一个None对象的tag节点，程序会抛出一个AttributeError。

下述代码(nonExistentTag是我们杜撰的一个节点，它并不存在)

```
print(bsObj.nonExistentTag)
```

就会返回一个None对象，因此为这个对象添加必要的检查是必须的，如果你执意不检查这个对象并对这个对象调用其他函数方法，如：

```
print(bsObj.nonExistentTag.someTag)
```

那么你会得到如下的异常：

```
AttributeError: 'NoneType'  object  has no  attribute   'someTag'
```

那么如何避免上文中说的这两种情况呢？最简单的方法就是分别对这两种情况进行检查：

```
try:
    badContent = bsObj.nonExistingTag.anotherTag
except AttributeError as e:
    print("Tag was not found")
else:
    if badContent == None:
       print ("Tag was not found")
    else:
        print(badContent)
```

这些检查起初看起来颇为繁琐，但是我们可以通过编写函数方法等措施来让程序更容易编写(更重要的是：更容易阅读)，我们对上述代码稍作调整，写出如下代码：

```
from urllib.request import urlopen
from urllib.error import HTTPError
from bs4 import BeautifulSoup
 def getTitle(url):
    try:
        html = urlopen(url)
    except HTTPError as e:
        return None
    try:
        bsObj = BeautifulSoup(html.read())
        title = bsObj.body.h1
    except AttributeError as e:
        return None
    return title
title = getTitle("http://www.pythonscraping.com/exercises/exercise1.html")
if title == None:
    print("Title could not be found")
else:
    print(title)
```

在这段代码中，我们创建了一个函数getTitle，该函数返回某个网页的title信息，当title信息不存在或者在获取title信息中发生异常时则返回None对象，在getTitle函数内部，我们使用try…catch…添加了对HTTPError异常的检查和对

AttributeError异常的检查，在try代码块中我们可以添加任意我们想要实现的代码，或者直接调用其他能够抛出AttributeError异常的函数。

在编写爬虫的过程中，你需要同时考虑到代码异常的处理和代码可读性的问题，此外你还可能希望你的代码能够被重用，因此编写一些通用的方法如getSiteHTML、getTitle(当然别忘记异常处理)都能够帮助你快速、有效的爬取网页内容。

p22