第 4 章 抓取网页数据

学习目标

- **了解什么是 urllib 库，能够快速使用 urllib 爬取到网页**
- **掌握如何转换 URL 编码，可以使用 GET 和 POST 两种方式实现数据传输**
- **知道伪装浏览器的用途，能够发送加入特定 Headers 的请求**
- **掌握如何自定义 opener，会设置代理服务器**
- **了解服务器的超时，可以设置等待服务器响应的时间**
- **熟悉一些常见的网络异常，可以对其捕获后进行相应的处理**
- **掌握 requests 库的使用，能够深入体会到 requests 的人性化**

基于爬虫的实现原理，我们进入爬虫的第一个阶段：抓取网页数据，即下载包含目标数据的网页。抓取网页需要通过爬虫向服务器发送一个 HTTP 请求，然后接收服务器返回的响应内容中的整个网页源代码。

Python 要想完成这个过程，既可以使用内置的 urllib 库，也可以使用第三方库 requests。使用这两个库，我们在抓取网页数据的时候，就只需要关心请求的 URL 格式，以及要传递什么参数，要设置什么样的请求头，而不需要关心它们的底层是怎样实现的。接下来，本章将会针对 urllib 和 requests 库的使用给大家做一个详细的讲解。

4.1 什么是 urllib 库

urllib 库是 Python 内置的 HTTP 请求库，它可以看做是处理 URL 的组件集合。urllib 库包含了四大模块，具体如下：

- urllib.request：请求模块
- urllib.error：异常处理模块
- urllib.parse：URL 解析模块
- urllib.robotparser：robots.txt 解析模块

4.2 快速使用 urllib 爬取网页

爬取网页，其实就是通过 URL 获取网页信息，这段网页信息的实质就是一段附加了 JS 和 CSS 的 HTML 代码。如果把网页比作是一个人，那么 HTML 就是它的骨架，JS 是它的肌肉，CSS 是它的衣服。由此看来，网页最重要的数据部分是存在于 HTML 中的。

4.2.1 快速爬取一个网页

urllib 库的使用比较简单，接下来，我们使用 urllib 快速爬取一个网页，具体代码如下：

```
import urllib.request
# 调用 urllib.request 库的 urlopen 方法，并传入一个 url
response = urllib.request.urlopen('http://www.baidu.com')
# 使用 read 方法读取获取到的网页内容
html = response.read().decode('UTF-8')
# 打印网页内容
print(html)
```

上述代码就是一个简单的爬取网页案例，爬取的网页结果如图 4-1 所示。

```
<html>
<head>

    <meta http-equiv="content-type" content="text/html;charset=utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=Edge">
    <meta content="always" name="referrer">
    <meta name="theme-color" content="#2932e1">
    <link rel="shortcut icon" href="/favicon.ico" type="image/x-icon" />
    <link rel="search" type="application/opensearchdescription+xml" href="/content-search.xml" title="百度搜索" />
    <link rel="icon" sizes="any" mask href="//www.baidu.com/img/baidu_85beaf5496f291521eb75ba38eacbd87.svg">


    <link rel="dns-prefetch" href="//s1.bdstatic.com"/>
    <link rel="dns-prefetch" href="//t1.baidu.com"/>
    <link rel="dns-prefetch" href="//t2.baidu.com"/>
    <link rel="dns-prefetch" href="//t3.baidu.com"/>
    <link rel="dns-prefetch" href="//t10.baidu.com"/>
    <link rel="dns-prefetch" href="//t11.baidu.com"/>
    <link rel="dns-prefetch" href="//t12.baidu.com"/>
    <link rel="dns-prefetch" href="//b1.bdstatic.com"/>

    <title>百度一下，你就知道</title>


<style id="css_index" index="index" type="text/css">html,body{height:100%}
html{overflow-y:auto}
body{font:12px arial;text-align:;background:#fff}
body,p,form,ul,li{margin:0;padding:0;list-style:none}
```

图 4-1　获取的网页源码

实际上，如果我们在浏览器上打开百度首页，右键选择“查看源代码”，你会发现，跟我们刚才打印出来的是一模一样。也就是说，上述案例仅仅用了几行代码，就已经帮我们把百度首页的全部代码下载下来了。

多学一招：Python2 使用的是 urllib2 库

Python2 中使用的是 urllib2 库来下载网页，该库的用法如下所示：

```
import urllib2
response = urllib2.urlopen('http://www.baidu.com')
```

Python3 出现后，之前 Python2 中的 urllib2 库被移到了 urllib.request 模块中，之前 urllib2 中很多函数的路径也发生了变化，希望大家在使用的时候多加注意。

4.2.2 分析 urlopen 方法

上一小节在爬取网页的时候，有一句核心的爬虫代码，如下所示：

```
response = urllib.request.urlopen('http://www.baidu.com')
```

该代码调用的是 urllib.request 模块中的 urlopen 方法，它传入了一个百度首页的 URL，使用的协议是 HTTP，这是 urlopen 方法最简单的用法。

其实，urlopen 方法可以接收多个参数，该方法的定义格式如下所示：

```
urllib.request.urlopen(url, data=None, [timeout, ]*, cafile=None,
                       capath=None, cadefault=False, context=None)
```

上述方法定义中的参数详细介绍如下：

- url: 表示目标资源在网站中的位置，可以是一个表示 URL 地址的字符串，也可以是一个 urllib.request 对象。
- data：用来指明向服务器发送请求的额外信息。HTTP 协议是 Python 支持的众多网络通信协议（如 HTTP、HTTPS、FTP 等）中唯一一个使用 data 参数的。也就是说，只有打开 http 网址的时候，data 参数才有作用。除此之外，官方 API 还指出：
  - data 必须是一个 bytes 对象。
  - data 必须符合 the standard application/x-www-form-urlencoded format 标准。使用 urllib.parse.urlencode()可以将自定义的 data 转换为标准格式，而这个函数能接收的参数类型是 Python 中的 mapping 类型（键值对类型，比如 dict）或者是只包含两个元素的元组类型。
  - data 默认为 None，此时是以 GET 方式发送请求，当用户设置 data 参数时，需要将发送请求的方式改为 POST。
- timeout：可选参数，该参数用于设置超时时间，单位是秒。
- cafile/capath/ cadefault: 用于实现可信任的 CA 证书的 HTTPS 请求，这些参数很少使用。
- context：实现 SSL 加密传输，该参数很少使用。

下面是 data 和 timeout 参数的使用示例，具体如下：

data 参数的使用：

```
import urllib.request
import urllib.parse
data = bytes(urllib.parse.urlencode({'world':'hello'}).encode('utf-8'))
response = urllib.request.urlopen('http://httpbin.org/post', data=data)
print(response.read())
```

timeout 参数的使用：

```
import urllib.request
import urllib.parse
response = urllib.request.urlopen('http://httpbin.org/get', timeout=1)
print(response.read())
```

4.2.3 使用 HTTPResponse 对象

使用 urllib.request 模块中的 urlopen 方法发送 HTTP 请求后，服务器返回的响应内容封

装在一个 HTTPResponse 类型的对象中，示例代码如下：

```
import urllib.request
response = urllib.request.urlopen('http://www.itcast.cn')
print(type(response))
```

执行示例代码，其输出结果为：

```
<class 'http.client.HTTPResponse'>
```

从输出结果可以看出，HTTPResponse 类属于 http.client 模块。该类提供了获取 URL、状态码、响应内容等一系列方法，常见的方法如下所示：

- geturl()：用于获取响应内容的 URL，该方法可以验证发送的 HTTP 请求是否被重新调配。
- info()：返回页面的元信息。
- getcode()：返回 HTTP 请求的响应状态码。

接下来使用一段示例代码来演示这几个方法的使用，具体如下：

```
import urllib.request
response = urllib.request.urlopen('http://python.org')
# 获取响应信息对应的 URL
print(response.geturl())
# 获取响应码
print(response.getcode())
# 获取页面的元信息
print(response.info())
```

执行上述示例代码，其输出结果如下：

```
 https://www.python.org/
200
Server: nginx
Content-Type: text/html; charset=utf-8
X-Frame-Options: SAMEORIGIN
X-Clacks-Overhead: GNU Terry Pratchett
Content-Length: 48729
Accept-Ranges: bytes
Date: Wed, 23 Aug 2017 03:29:51 GMT
Via: 1.1 varnish
Age: 2915
Connection: close
X-Served-By: cache-nrt6129-NRT
X-Cache: HIT
X-Cache-Hits: 29
X-Timer: S1503458991.290683,VS0,VE0
Vary: Cookie
Strict-Transport-Security: max-age=63072000; includeSubDomains
```

4.2.4 构造 Request 对象

当我们使用 urlopen 方法发送一个请求时，如果希望执行更为复杂的操作，比如增加

HTTP 报头，则必须创建一个 Request 对象来作为 urlopen 方法的参数。接下来，同样以百度首页为例，为大家演示如果使用 Request 对象来爬取数据，示例代码如下：

```
import urllib.request
# 将 url 作为 Request 方法的参数，构造并返回一个 Request 对象
request = urllib.request.Request('http://www.baidu.com')
# 将 Request 对象作为 urlopen 方法的参数，发送给服务器并接收响应
response = urllib.request.urlopen(request)
# 使用 read 方法读取获取到的网页内容
html = response.read().decode('UTF-8')
# 打印网页内容
print(html)
```

上述代码的运行结果和2.2.1小节是完全一样的，只不过代码中间多了一个Request对象。在使用 urllib 库发送 URL 的时候，我们推荐大家使用构造 Request 对象的方式。因为在发送请求时，除了必须设置的 url 参数外，还可能会加入很多内容，例如下面的参数：

- data：默认为空，该参数表示提交表单数据，同时 HTTP 请求方法将从默认的 GET 方式改为 POST 方式。
- headers：默认为空，该参数是一个字典类型，包含了需要发送的 HTTP 报头的键值对。

接下来也是一个构造 Request 对象的案例，该案例在构造 Request 对象时传入 data 和 headers 参数，具体代码如下：

```
import urllib.request
import urllib.parse
url = 'http://www.itcast.cn'
header = {
   "User-Agent" : "Mozilla/5.0 (compatible; MSIE 9.0; Windows NT6.1; Trident/5.0)",
   "Host":"httpbin.org"
}
dict_demo = {"name": "itcast"}
data = bytes(urllib.parse.urlencode(dict_demo).encode('utf-8'))
# 将 url 作为 Request 方法的参数，构造并返回一个 Request 对象
request = urllib.request.Request(url, data=data, headers=header)
# 将 Request 对象作为 urlopen 方法的参数，发送给服务器并接收响应
response = urllib.request.urlopen(request)
# 使用 read 方法读取获取到的网页内容
html = response.read().decode('UTF-8')
# 打印网页内容
print(html)
```

上述案例可以实现传智播客官网首页的爬取。通过构造 Request 对象的方式，服务器会根据发送的请求，返回对应的响应内容，这种做法在逻辑上也是非常清晰明确的。

4.3 使用 urllib 实现数据传输

在抓取网页时，我们通过 URL 传递数据给服务器，传递数据的方式主要分为 GET 和 POST 两种。这两种方式最大的区别在于，GET 方式是直接使用 URL 访问，在 URL 中包含

了所有的参数。而 POST 方式则不会在 URL 中显示所有的参数，本节将会针对这两种数据传递方式进行讲解。

4.3.1 URL 编码转换

当我们传递的 URL 包含中文或者其它特殊字符（例如，空格或/等）时，需要使用 urllib.parse 库中的 urlencode 方法将 URL 进行编码，它可以将“key:value”这样的键值对转换成“key=value”这样的字符串。示例代码如下：

```
import urllib.parse
data = {
      'a': '传智播客',
      'b': '黑马程序员'
}
result = urllib.parse.urlencode(data)
print(result)
```

输出结果如下：

```
a=%E4%BC%A0%E6%99%BA%E6%92%AD%E5%AE%A2&b=%E9%BB%91%E9%A9%AC%E7%A8%8B%E5%BA%8F%E5%91%98
```

反之，解码使用的是 url.parse 库的 unquote 方法，示例代码如下：

```
import urllib.parse
result = urllib.parse.unquote('a=%E4%BC%A0%E6%99%BA%E6%92%AD%E5%AE%A2')
print(result)
```

输出结果为：

```
a=传智播客
```

4.3.2 处理 GET 请求

GET 请求一般用于向服务器获取数据，比如说，我们用百度搜索传智播客（URL 是“https://www.baidu.com/s?wd=传智播客”），浏览器跳转的页面如图 4-2 所示。

图 4-2　GET 请求

此时，如果使用 Fiddler 查看 HTTP 请求，发现有个 GET 请求的格式如下：

```
https://www.baidu.com/s?wd=%E4%BC%A0%E6%99%BA%E6%92%AD%E5%AE%A2
```

在这个请求中，“？”后面的字符串就包含了我们要查询的关键字“传智播客”。下面，我们尝试使用 GET 方式发送请求，具体代码如下：

```
import urllib.request
import urllib.parse
url = "http://www.baidu.com/s"
word = {"wd": "传智播客"}
word = urllib.parse.urlencode(word) #转换成 url 编码格式（字符串）
new_url = url + "?" + word
headers = {"User-Agent": "Mozilla/5.0 (Windows NT 10.0; WOW64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2704.103 Safari/537.36"}
request = urllib.request.Request(new_url, headers = headers)
response = urllib.request.urlopen(request)
html = response.read().decode('UTF-8')
print(html)
```

运行程序，程序输出的结果和使用浏览器搜索网页 “https://www.baidu.com/s?wd=传智播客”的源代码是一模一样的，由此说明我们成功爬取了页面。

4.3.3 处理 POST 请求

前面我们分析 urlopen 方法时提到过，发送 HTTP 请求时，如果是以 POST 方式发送请求，urlopen 方法必须设置 data 参数，data 参数是以字典的形式存放数据。

当访问有道词典翻译网站进行词语翻译时，会发现不管输入什么内容，其 URL 一直都是“http://fanyi.youdao.com”。通过使用 Fiddler 观察，发现该网站向服务器发送的是 POST

请求，如图 4-3 所示。

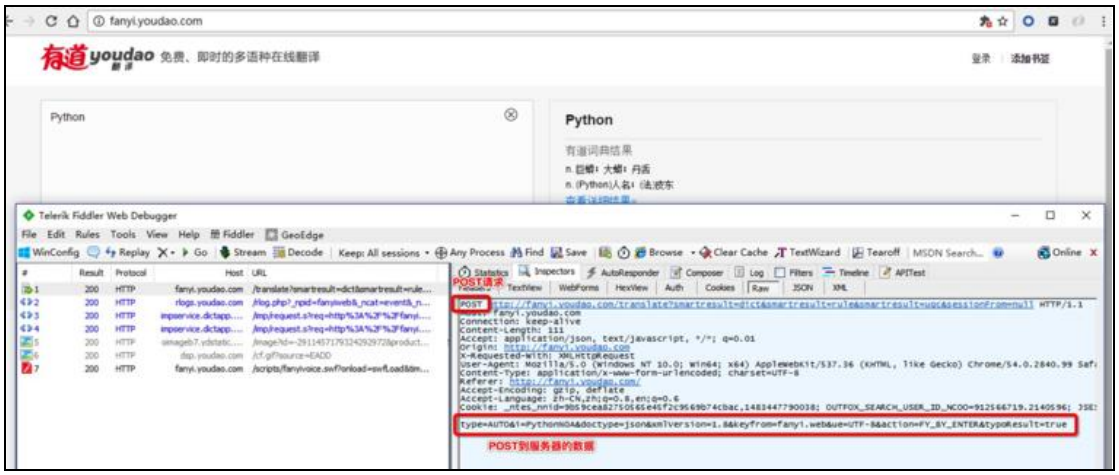

图 4-3　有道词典翻译网站

从图 4-3 中可以看出，当使用有道词典翻译“Python”时，返回的结果是一个 JSON 字符串。 接下来，我们来尝试模拟这个 POST 请求，具体代码如下：

```
import urllib.request
import urllib.parse
# POST 请求的目标 URL
url = "http://fanyi.youdao.com/translate?
smartresult=dict&smartresult=rule&smartresult=ugc&sessionFrom=null"
headers = {"User-Agent": "Mozilla...."}
# 打开 Fiddler 请求窗口，点击 WebForms 选项查看数据体
formdata = {
     "type":"AUTO",
     "i":"i love python"
     "doctype":"json",
     "xmlVersion":"1.8",
     "keyfrom":"fanyi.web",
     "ue":"UTF-8",
     "action":"FY_BY_ENTER",
     "typoResult":"true"
}
 data = bytes(urllib.parse.urlencode(formdata).encode('utf-8'))
 request = urllib.request.Request(url, data=data, headers=headers)
 response = urllib.request.urlopen(request)
 print(response.read().decode('utf-8'))
```

执行上述代码，输出结果如下：

```
{
 "type":"EN2ZH_CN",
 "errorCode":0,
 "elapsedTime":0,
 "translateResult":[
     [
         {
```

```
                "src":"i love python",
                "tgt":"我喜欢 python"
            }
        ]
    ],
    "smartResult":{
        "type":1,
        "entries":[
            "",
            "肆文",
            "高德纳"
        ]
    }
}
```

4.4 添加特定 Headers—请求伪装

对于一些需要登录的网站，如果不是从浏览器发出的请求，我们是不能获得响应内容的。针对这种情况，我们需要将爬虫程序发出的请求伪装成一个从浏览器发出的请求。伪装浏览器需要自定义请求报头，也就是在发送 Request 请求时，加入特定的 Headers。

添加特定 Headers 的方式很简单，只需要调用 Request.add_header()即可。如果想查看已有的 Headers，可以通过调用 Request.get_header()查看。

下面是一个添加自定义请求头的例子，具体如下：

```
import urllib.request
url = "http://www.itcast.cn"
user_agent = {"User-Agent" : "Mozilla/5.0 (compatible; MSIE 9.0; Windows NT
6.1; Trident/5.0)"}
request = urllib.request.Request(url, headers = user_agent)
# 也可以通过调用 Request.add_header() 添加/修改一个特定的 header
request.add_header("Connection", "keep-alive")
# 也可以通过调用 Request.get_header()来查看 header 信息
# request.get_header(header_name="Connection")
response = urllib.request.urlopen(request)
print(response.code)    # 可以查看响应状态码
html = response.read()
print(html)
```

运行程序后，使用 Fiddler 查看 HTTP 请求，可以看到在发送的请求头中，已经包含了我们添加的 Header。

4.5 代理服务器

很多网站会检测某一段时间某个 IP 的访问次数，如果同一 IP 访问过于频繁，那么该网站会禁止来自该 IP 的访问。针对这种情况，我们可以使用代理服务器，每隔一段时间换一个代理。如果某个 IP 被禁止，那么就可以换成其他 IP 继续爬取数据，从而可以有效解决

被网站禁止访问的情况。

代理多用于反“反爬虫”机制，这里大家知道这个概念即可，后续会有代理服务器的设置和使用的详细讲解。

4.5.1 简单的自定义 opener

opener 是 urllib.request.OpenerDirector 类的对象，我们之前一直都在使用的 urlopen，就是模块帮我们构建好的一个 opener，但是它不支持代理、Cookie 等其他的 HTTP/HTTPS 高级功能。所以如果要想设置代理，不能使用自带的 urlopen，而是要自定义 opener。自定义 opener 需要执行下列三个步骤：

(1) 使用相关的 Handler 处理器创建特定功能的处理器对象。

(2) 通过 urllib.request.build_opener()方法使用这些处理器对象创建自定义的 opener 对象。。

(3) 使用自定义的 opener 对象，调用 open 方法发送请求。这里需要注意的是，如果程序里所有的请求都使用自定义的 opener，可以使用 urllib2.install_opener() 将自定义的 opener 对象定义为全局 opener，表示之后凡是调用 urlopen，都将使用自定义的 opener。

接下来我们实现一个最简单的自定义 opener，具体代码如下：

```
import urllib.request
# 构建一个 HTTPHandler 处理器对象，支持处理 HTTP 请求
http_handler = urllib.request.HTTPHandler()
# 调用 urllib2.build_opener()方法，创建支持处理 HTTP 请求的 opener 对象
opener = urllib.request.build_opener(http_handler)
# 构建 Request 请求
request = urllib.request.Request("http://www.baidu.com/")
# 调用自定义 opener 对象的 open()方法，发送 request 请求
# （注意区别：不再通过 urllib.request.urlopen()发送请求）
response = opener.open(request)
# 获取服务器响应内容
print(response.read())
```

上述方式发送请求得到的结果，和使用 urllib.request.urlopen 发送 HTTP/HTTPS 请求得到的结果是一样的。

如果在 HTTPHandler 方法中增加参数 debuglevel=1，会将 Debug Log 打开，这样程序在执行的时候，会把收包和发包的报头自动打印出来，以方便调试。示例代码如下：

```
# 构建一个 HTTPHandler 处理器对象，同时开启 Debug Log，debuglevel 值设置为 1
http_handler = urllib.request.HTTPHandler(debuglevel=1)
```

4.5.2 设置代理服务器

我们可以使用 urllib.request 中的 ProxyHandler 方法来设置代理服务器，接下来就通过示例来说明如何使用自定义 opener 来设置代理服务器，代码如下。

```
import urllib.request
```

```
# 构建了两个代理 Handler，一个有代理 IP，一个没有代理 IP
httpproxy_handler = urllib.request.ProxyHandler({"http" : "124.88.67.81:80"})
nullproxy_handler = urllib.request.ProxyHandler({})
proxy_switch = True # 定义一个代理开关
# 通过 urllib.request.build_opener()方法使用代理 Handler 对象创建自定义 opener 对象
# 根据代理开关是否打开，使用不同的代理模式
if proxy_switch:
    opener = urllib.request.build_opener(httpproxy_handler)
else:
    opener = urllib.request.build_opener(nullproxy_handler)
request = urllib.request.Request("http://www.baidu.com/")
response = opener.open(request)
print(response.read())
```

免费开放代理的获取基本没有成本，我们可以在一些代理网站上收集这些免费代理，测试后如果可以用，就把它收集起来用在爬虫上面。免费代理网站主要有以下几个：

- 西刺免费代理 IP
- 快代理免费代理
- Proxy360 代理
- 全网代理 IP

如果代理 IP 足够多，就可以像随机获取 User-Agent 一样，随机选择一个代理去访问网站。示例代码如下：

```
import urllib.request
import random
proxy_list = [
    {"http" : "124.88.67.81:80"},
    {"http" : "124.88.67.81:80"},
    {"http" : "124.88.67.81:80"},
    {"http" : "124.88.67.81:80"},
    {"http" : "124.88.67.81:80"}
]
# 随机选择一个代理
proxy = random.choice(proxy_list)
# 使用选择的代理构建代理处理器对象
httpproxy_handler = urllib.request.ProxyHandler(proxy)
opener = urllib.request.build_opener(httpproxy_handler)
request = urllib.request.Request("http://www.baidu.com/")
response = opener.open(request)
print(response.read())
```

但是，这些免费开放代理一般会有很多人都在使用，而且代理有寿命短，速度慢，匿名度不高，HTTP/HTTPS 支持不稳定等缺点。所以，专业爬虫工程师或爬虫公司会使用高品质的私密代理，这些代理通常需要找专门的代理供应商购买，再通过用户名/密码授权使用。

4.6 超时设置

假设有个需求，我们要爬取 1000 个网站，如果其中有 100 个网站需要等待 30s 才能返回数据，那如果要返回所有的数据，至少需要等待 3000 秒。如此长时间的等待显然是不可行的，为此，我们可以为 HTTP 请求设置超时时间，一旦超过这个时间，服务器还没有返回响应内容，那么就会抛出一个超时异常，这个异常需要使用 try 语句来捕获。

例如，我们使用快代理（一个开放代理网站）中的一个 IP，它的响应速度需要 2 秒。此时，如果将超时时间设置为 1 秒，那么程序会抛出异常。具体代码如下：

```
import urllib.request
try:
    url = 'http://218.56.132.157:8080'
    file = urllib.request.urlopen(url, timeout=1) # timeout 设置超时的时间
    result = file.read()
    print(result)
except Exception as error:
    print(error)
```

运行程序后，输出结果为：

```
<urlopen error timed out>
```

4.7 常见的网络异常

当使用 urlopen 方法发送 HTTP 请求时，如果 urlopen 不能处理返回的响应内容，那么就会产生错误。这里，我们将针对两个常见的异常（URLError 和 HTTPError）以及对它们的错误处理进行简单的介绍。

4.7.1 URLError 异常和捕获

URLError 产生的原因主要有：

(1) 没有连接网络

(2) 服务器连接失败

(3) 找不到指定的服务器

我们可以使用 try-except 语句来捕获相应的异常，例如下面的代码：

```
import urllib.request
import urllib.error
request = urllib.request.Request("http://www.ajkfhafwjqh.com")
try:
    urllib.request.urlopen(request, timeout=5)
except urllib.error.URLError as err:
    print(err)
```

执行上述代码，则输出结果如下：

```
<urlopen error [Errno 11004] getaddrinfo failed>
```

上述报错信息是 urlopen error，错误代码是 11004。发生错误的原因是没有找到指定的服务器。

4.7.2 HttpError 异常和捕获

每一个服务器的 HTTP 响应都有一个数字响应码，这些响应码有些表示无法处理请求内容，如果无法处理，urlopen 会抛出 HTTPError。HTTPError 是 URLError 的子类，它的对象拥有一个整型的 code 属性，表示服务器返回的错误代码。

下面是一个例子，具体如下：

```
import urllib.request
import urllib.error
request = urllib.request.Request('http://www.itcast.cn/net')
try:
    urllib.request.urlopen(request)
except urllib.error.HTTPError as e:
    print(e.code)
```

输出结果如下：

```
404
```

上述输出了 404 的错误码，其含义是没有找到这个页面。

这里需要说明的是，不同的响应码代表不同的含义，例如 100-200 范围的号码表示成功，而错误码的范围在 400-599 之间。

4.8 更人性化的 requests 库

在使用 urllib 库时可以发现，虽然这个库提供了很多关于 HTTP 请求的函数，但是这些函数的使用方式并不简洁，仅仅实现一个小功能就要用到一大堆代码。因此，Python 提供了一个便于开发者使用的第三方库—requests。

requests 库自称“HTTP for Humans”，直译过来的意思是专门为人类设计的 HTTP 库，能够被开发人员安全地使用。接下来，将针对 requests 库的使用进行简单的介绍。

4.8.1 什么是 requests 库

requests 是基于 Python 开发的 HTTP 库，与 urllib 标准库相比，它不仅使用方便，而且能节约大量的工作。实际上，requests 是在 urllib 的基础上进行了高度的封装，它不仅继承了 urllib 的所有特性，而且还支持一些其它的特性，比如使用 Cookie 保持会话、自动确定响应内容的编码等，可以轻而易举地完成浏览器的任何操作。

requests 库中提供了如下常用的类：

- requests.Request：表示请求对象，用于准备一个请求发送到服务器。
- requests.Response：表示响应对象，其中包含服务器对 HTTP 请求的响应。
- requests.Session：表示请求会话，提供 Cookie 持久性、连接池和配置。

其中，Request 类的对象表示一个请求，它的生命周期针对一个客户端请求，一旦请求发送完毕，该请求包含的内容就会被释放掉。而 Session 类的对象可以跨越多个页面，它的生命周期同样针对的是一个客户端。当关闭这个客户端的浏览器时，只要是在预先设置的会话周期内（一般是 20~30 分钟），这个会话包含的内容会一直存在，不会被马上释放掉。例如，用户登陆某个网站时，可以在多个 IE 窗口发出多个请求。

4.8.2 requests 库初体验

和 urllib 库相比，requests 库更加深得人心，它不仅能够重复地读取返回的数据，而且还能自动确定响应内容的编码。为了能让大家直观地看到这些变化，接下来，分别使用 urllib 库和 requests 库爬取百度网站中“传智播客”关键字的搜索结果网页。

（1）使用 urllib 库以 GET 请求的方式爬取网页，具体代码如下：

```
# 导入请求和解析模块
import urllib.request
import urllib.parse
# 请求的 URL 路径和查询参数
url = "http://www.baidu.com/s"
word = {"wd":"传智播客"}
# 转换成 url 编码格式（字符串）
word = urllib.parse.urlencode(word)
# 拼接完整的 URL 路径
new_url = url + "?" + word
# 请求报头
headers ={ "User-Agent": "Mozilla/5.0 (Windows NT 10.0; WOW64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2704.103 Safari/537.36"}
# 根据 URL 和 headers 构建请求
request = urllib.request.Request(new_url, headers = headers)
# 发送请求，并接收服务器返回的文件对象
response = urllib.request.urlopen(request)
# 使用 read 方法读取获取到的网页内容，使用 UTF-8 格式进行解码
html = response.read().decode('UTF-8')
print(html)
```

（2）使用 requests 库以 GET 请求的方式爬取网页，具体代码如下：

```
# 导入 requests 库
import requests
# 请求的 URL 路径和查询参数
url = "http://www.baidu.com/s"
param = {"wd": "传智播客"}
# 请求报头
headers = {"User-Agent": "Mozilla/5.0 (Windows NT 10.0; WOW64)
            AppleWebKit/537.36 (KHTML, like Gecko)
            Chrome/51.0.2704.103 Safari/537.36"
}
# 发送 GET 请求，返回一个响应对象
response = requests.get(url, params=param, headers=headers)
# 查看响应的内容
print(response.text)
```

比较上述两段代码不难发现，使用 requests 库减少了发送请求的代码量。接下来，我们再从细节上体会一下 requests 库的便捷之处，具体如下：

1) 无须再转换为 URL 路径编码格式，拼接完整的 URL 路径。

2) 无须再频繁地为中文转换编码格式。

3) 从发送请求的函数名称，可以很直观地判断发送到服务器的方式。

另外，urlopen 方法返回的是一个文件对象，需要调用 read()方法一次性读取；而 get 函数返回的是一个响应对象，可以访问该对象的 text 属性查看响应的内容。

这里虽然只初步介绍了 requests 库的用法，但是我们也可以从中看出，整个程序的逻辑非常易于理解，更符合面向对象开发的思想，并且减少了代码量，提高了开发效率，给开发人员带来了便利。

4.8.3 发送请求

requests 库中提供了很多发送 HTTP 请求的函数，具体如表 4-1 所示。

表 4-1 requests 库的请求函数

函数	功能说明
requests.request()	构造一个请求，支撑以下各方法的基础方法
requests.get()	获取 HTML 网页的主要方法，对应于 HTTP 的 GET 请求方式
requests.head()	获取 HTML 网页头信息的方法，对应于 HTTP 的 HEAD 请求方式
requests.post()	向 HTML 网页提交 POST 请求的方法，对应于 HTTP 的 POST 请求方式
requests.put()	向 HTML 网页提交 PUT 请求的方法，对应于 HTTP 的 PUT 请求方式
requests.patch()	向 HTML 网页提交局部修改请求，对应于 HTTP 的 PATCH 请求方式
requests.delete()	向 HTML 网页提交删除请求，对应于 HTTP 的 DELETE 请求方式

表 4-1 列举了一些常用于 HTTP 请求的函数，这些函数都会做两件事情，一件是构建一个 Request 类型的对象，该对象将被发送到某个服务器上请求或者查询一些资源；另一件是一旦得到服务器返回的响应，就会产生一个 Response 对象，该对象包含了服务器返回的所有信息，也包括原来创建的 Request 对象。

4.8.4 返回响应

Response 类用于动态地响应客户端的请求，控制发送给用户的信息，并且将动态地生成响应，包括状态码、网页的内容等。接下来，通过一张表来列举 Response 类包含的信息，如表 4-2 所示。

表 4-2 Response 类的常用属性

属性	说明
status_code	HTTP 请求的返回状态，200 表示连接成功，404 表示失败
text	HTTP 响应内容的字符串形式，即 URL 对应的页面内容

encoding	从 HTTP 请求头中猜测的响应内容编码方式
apparent_encoding	从内容中分析出的响应编码的方式（备选编码方式）
content	HTTP 响应内容的二进制形式

Response 类会自动解码来自服务器的内容，并且大多数的 Unicode 字符集都可以被无缝地解码。

当请求发出之后，Requests 库会基于 HTTP 头部信息对响应的编码作出有根据的判断。例如，在使用 response.text （response 为响应对象）时，可以使用判断的文本编码。此外，还可以找出 Requests 库使用了什么编码，并且可以设置 encoding 属性进行改变，示例如下：

```
>>> response.encoding
'utf-8'
>>> response.encoding = 'ISO-8859-1'
```

再次调用 text 属性获取返回的文本内容时，将会使用上述设置的新的编码方式。

4.9 案例—使用 urllib 库爬取百度贴吧

为了让大家更好地了解使用 urllib 库爬取网页的流程，接下来就带领大家使用 urllib 库实现一个爬取百度贴吧网页的案例。首先我们来分析一下百度贴吧网站的 URL 地址的格式。例如，在百度贴吧搜索“传智播客”，就会显示出所有和传智播客相关的帖子，如图 4-4 所示。

图 4-4　传智播客吧

图 4-4 中百度贴吧的 URL 地址如下：

```
http://tieba.baidu.com/f?kw=传智播客&pn=350
```

上述 URL 地址中，“http://tieba.baidu.com/f”是基础部分，问号后面的“kw=传智播客&pn=150”是参数部分。参数部分的“传智播客”是我们搜索的关键字，pn 值与贴吧的页码有关。如果用 n 表示第几页，那么 pn 参数的值是按照（n-1）*50 的规律进行赋值。例如，百度贴吧中的传智播客吧，前三页对应的 URL 地址如下所示：

第一页：http://tieba.baidu.com/f?kw=传智播客&pn=0

第二页：http://tieba.baidu.com/f?kw=传智播客&pn=50

第三页：http://tieba.baidu.com/f?kw=传智播客&pn=100

熟悉了百度贴吧 URL 格式的规律之后，接下来我们使用 urllib 库来爬取传智播客贴吧第 1 至 3 页的内容，并将爬取到的内容保存到文件中。具体步骤如下：

（1）首先提示用户输入要爬取的贴吧名，以及要查询的起始页和结束页。然后，使用 urllib.parse.urlencode()对 url 参数进行转码，组合成一个完整的可访问的 url。具体代码如下

所示：

```
if __name__=="__main__":
    kw = input("请输入需要爬取的贴吧名：")
    begin_page = int(input("请输入起始页："))
    end_page = int(input("请输入结束页："))
    url = 'http://tieba.baidu.com/f?'
    key = urllib.parse.urlencode({"kw": kw})
    # 组合后的 url 示例：http://tieba.baidu.com/f?kw=lol
    url = url + key
    tieba_spider(url, begin_page, end_page)
```

（2）编写一个用于爬取百度贴吧的函数，该函数需要传递 3 个参数，分别是 URL 地址、表示爬取页码范围的起始页码和终止页码。具体代码如下：

```
def tieba_spider(url, begin_page, end_page):
    '''
    作用：贴吧爬虫调度器，负责组合处理每个页面的 url
    url：贴吧 url 的前半部分
    begin_page:起始页码
    end_page:结束页
    '''
    for page in range(begin_page, end_page + 1):
        pn = (page - 1) * 50
        file_name = "第" + str(page) + "页.html"
        full_url = url + "&pn=" + str(pn)
        html = load_page(full_url, file_name)
        write_page(html, file_name)
```

（3）编写一个实现爬取功能的函数，该函数构造了一个 Request 对象，然后使用 urllib.request.urlopen 爬取网页，返回响应内容，具体代码如下所示：

```
def load_page(url, filename):
    '''
    作用:根据 url 发送请求，获取服务器响应文件
    url：需要爬取的 url 地址
    '''
    headers = {"User-Agent": "Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1;
                Trident/5.0;")
    request = urllib.request.Request(url, headers=headers)
    return urllib.request.urlopen(request).read()
```

（4）最后，由于我们希望将爬取到的每页信息存储在本地磁盘上，所以需要编写一个存储文件的函数。具体代码如下：

```
def write_page(html, filename):
    '''
    作用：将 html 内容写入本地文件
    html：服务器响应文件内容
    '''
    print("正在保存" + filename)
    with open(filename, 'w') as file:
```

```
            file.write(html.decode('utf-8'))
```

（5）运行程序，按照提示输入贴吧名称，以及要爬取的起始页和结束页， 发现会生成三个文件，这三个文件保存的正是爬取的传智播客贴吧的前三个页面。

其实很多网站都是这样的，同一网站下的 HTML 页面编号，与对应网址后的网页序号一一对应，只要发现规律就可以批量爬取页面了。

4.10 本章小结

本章分享了 Python 中用作抓取网页的两个库：urllib 和 requests。首先简单地介绍了一下什么是 urllib 库，以及让读者快速上手的 urllib 案例，然后讲解了关于 urllib 的一些使用技巧，包括使用 urllib 传输数据、添加特定的 Headers、简单自定义 opener、服务器响应超时设置，以及一些常见的网络异常，接着介绍了更加人性化的 requests 库，最后结合一个百度贴吧的案例，介绍了如何使用 urllib 抓取网页数据。

通过对本章内容的学习，大家应该能熟练地掌握两个库的使用，并反复使用多加练习，另外还可以参考官网提供的文档深入地学习。

4.11 本章习题

一、填空题

1. 一旦超过了服务器设置的_____时间，就会抛出一个超时异常。
2. 若客户端没有连接到网络，则使用 urlopen 方法发送请求后会产生_____异常。
3. _____是 Python 内置的 HTTP 请求库，可以看做是处理 URL 的组件集合。
4. 如果要获取 Response 类中字符串形式的响应内容，可以访问_____属性获取。
5. 要想将爬虫程序发出的_____伪装成一个浏览器，就需要自定义请求报头。

二、判断题

1. 如果 URL 中包含了中文，则可以使用 urlencode 方法进行编码。（ ）
2. 登录网站时，只有浏览器发送的请求才能获得响应内容。 （ ）
3. 如果访问某网站的频率太高，则这个网站可能会禁止访问。 （ ）
4. Urlopen 是一个特殊的 opener，支持设置代理 IP。 （ ）
5. urlopen 函数返回的是一个文件对象，需要调用 read()方法一次性读取。 （ ）

三、选择题

1. 使用 urlopen 方法发送请求后，服务器会返回一个（ ）类型的对象。
   A. HTTPResponse
   B. ResponseHTTP
   C. Response
   D. ServiceResponse
2. 请看下面一段示例程序：

```
import urllib.request

response = urllib.request.urlopen('http://python.org')

print(response.getcode())
```

若上述示例程序正常运行成功，则程序输出的结果为（ ）。

A. 200

B. 304
C. 403
D. 500

3. 下列方法中，用于对传递的 URL 进行编码和解码的是（ ）。
A. urldecode，urlencode
B. unquote，urlencode
C. urlencode，urldecode
D. urlencode，unquote

4. 通过加入特定的（ ），可以将爬虫发出的请求伪装成浏览器。
A. Request
B. opener
C. Headers
D. User_Agent

5. 下列方法中，能够用来设置代理服务器的是（ ）。
A. urlopen
B. ProxyHandler
C. urldecode
D. Proxy

四、简答题

1. 请简述爬虫是如何抓取网页的。
2. 请简述 urllib 和 requests 的异同。

五、编程题

编写一个程序，分别使用 urllib 和 requests 抓取关于 Python 的百度搜索页面。