第 5 章

CHAPTER 5

Requests 初级使用

本章开始讲解基于 Python 实现的 HTTP 接口测试。主要依赖第三方库 Requests 实现。现有接口测试工具可以理解为一种接口测试框架的可视化实现，使用 Python 实现接口测试及测试用例的维护，可以使测试工作更为灵活，通过实现自定义测试框架会更贴近具体接口测试项目的需求和管理。本章作为 Requests 基本接口测试的讲解，为后面测试框架的实现打下基础。

5.1 Requests 介绍

Requests 是用 Python 语言编写的，基于 urllib，采用 Apache License 2.0 开源协议的 HTTP 库。Requests 是一个很实用的 Python HTTP 客户端库，当编写爬虫和测试服务器响应数据时经常会用到，是基础 Python 实现 HTTP 接口测试必不可少的第三方工具库。

Requests 库有 7 种常用方法，以 request() 为基础构造方法，在此基础上实现 6 种常见请求方法，见表 5-1。

表 5-1 Requests 常见请求方法

序 号	方 法 名 称	描 述
1	request()	构造方法，构造对象中包括后面的 6 种基本 HTTP 请求方法
2	get()	获取 HTML 页面信息的主要方法，返回值通常是 HTML 页面
3	head()	获取 HTML 页面 Header 的主要方法，返回值通常是指定请求 Headers 值
4	post()	提交 POST 请求方法，可以携带 Content-Type 指定的主体内容
5	put()	提交 PUT 请求方法，与 POST 请求方法类似，可以指定请求位置
6	patch()	提交 PATCH 请求方法，可以设置当前用户存储参数值
7	delete()	提交 DELETE 请求方法，可以删除指定服务器资源

5.1.1 GET 方法的使用

GET 方法是 Requests 库中使用频率最高的一种方法，以百度搜索首页为例，使用 PyCharm 编写接口测试脚本实现对百度搜索首页的访问，请求代码如下：

```
//chapter05/api_get.py


#导入第三方包 requests
import requests

#访问百度首页
url = 'https://www.baidu.com/'

#发送 GET 请求,将返回结果存入变量 response
response = requests.get(url)

#输出结果,内容中包含中文,需 UTF-8 转码
print(response.content.decode('UTF-8'))
```

返回结果如图 5-1 所示。

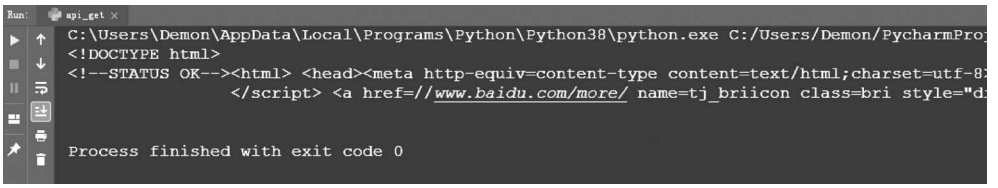

图 5-1　GET 请求返回结果

5.1.2　POST 方法的使用

当访问接口携带大量参数且一些敏感信息不方便直接跟随 URL 展现在浏览器网址栏时,可以使用 POST 方法来完成请求。以孔夫子旧书网用户登录接口为例,请求代码如下:

```
//chapter05/api_post.py

import requests

url = 'https://login.kongfz.com/Pc/Login/account'

#将请求 Headers 以字典的方式存入变量 headers
headers = {
    "Connection":"keep-alive",
    "Content-Length":"150",
    "sec-ch-ua":"\" Not A;Brand\";v=\"99\", \"Chromium\";v=\"100\", \"Google Chrome\";
v=\"100\"",
    "sec-ch-ua-mobile":"?0",
    "User-Agent":"Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/100.0.4896.75 Safari/537.36",
    "X-Tingyun-Id":"OHEPtRD8z8s;r=325222859",
    "Content-Type":"application/x-www-form-urlencoded; charset=UTF-8",
    "Accept":"application/json, text/javascript, */*; q=0.01",
```

```
        "X-Requested-With":"XMLHttpRequest",
        "Sec-Fetch-Site":"same-origin",
        "Sec-Fetch-Mode":"cors",
        "Sec-Fetch-Dest":"empty",
        "Host":"login.kongfz.com"
    }

    #将 Body 以字典的方式存入变量 data
    data = {
        "loginName":"Thinkerbang",          #此处为用户名参数,读者需要自行替换
        "loginPass":"123456",
        "captchaCode":"",
        "autoLogin":"0",
        "newUsername":"",
        "returnUrl":"https://user.kongfz.com/index.html",
        "captchaId":""
        }

    #发送 POST 请求,将返回结果存入变量 response
    response = requests.post(url=url,headers=headers,data=data)

    #由于返回值类型是 JSON 字符串,因此使用 json()方法进行输出
    print(response.json())
```

返回结果如图 5-2 所示。

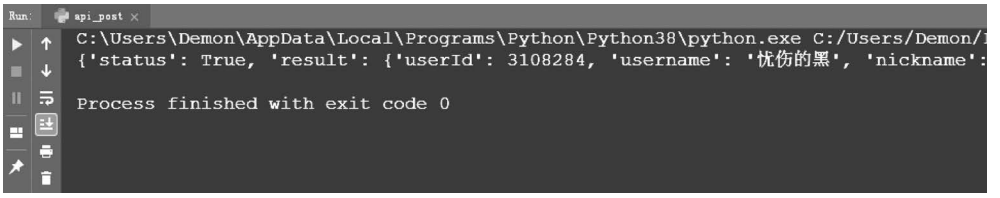

图 5-2 POST 请求返回结果

5.1.3 PUT 方法的使用

PUT 方法与 POST 方法的作用类似,它们都可以完成 Body 传参操作。区别在于 PUT 方法与 GET、DELETE 等方法类似,这些方法都是幂等的,而 POST 方法不是。一个简单的区分方式是可以看接口请求是否会有重复信息,例如一个发布信息接口,同样参数情况下提交接口请求,使用 PUT 方法提交时,无论提交几次都只会发布一次信息,PUT 方法每次都会将之前同参信息覆盖掉。使用 POST 方法提交时,由于它的不幂等性,同参多次提交会出现重复信息。在实际使用过程中,POST 方法的使用频率很高,这是因为二者使用上的差异在实际开发过程中可以通过其他方法解决。

以 httpbin 网站 PUT 接口为例，请求代码如下：

```
//chapter05/api_put.py

import requests

url = "http://httpbin.org/put"

data = {
    "name":"Thinkerbang",
    "age":"2016"
}

response = requests.put(url = url,data = data)

print(response.json())
```

返回结果如图 5-3 所示。

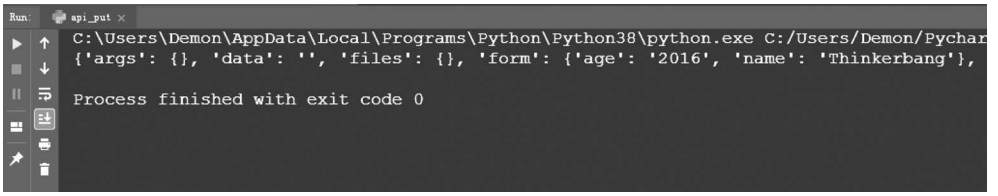

图 5-3　PUT 请求返回结果

5.1.4　HEAD 方法的使用

HEAD 请求可以看作没有返回参数的 GET 请求，这两种请求方法的本质是相同的。当浏览器向服务器发送页面访问请求时，使用 GET 方法发送接口请求，服务器会返回相应的请求资源；当浏览器向服务器确认当前访问页面是否有动态更新时，使用 HEAD 方法发送接口请求，服务器会检查当前页面的更新状态，将结果返回浏览器，此时的返回并不包含当前页面资源数据。

HEAD 请求通常应用在检查资源有效性、超链接的可访问性及页面内容最近是否有修改等场合。HEAD 请求返回的协议状态码与 GET 请求相同。

以百度网站首页 HEAD 接口为例，请求代码如下：

```
//chapter05/api_head.py

#导入第三方包 requests
import requests

#访问百度首页
url = 'https://www.baidu.com/'
```

```
#发送 GET 请求,将返回结果存入变量 response
response = requests.head(url)

#实际输出结果为空
print("返回 Body:", response.content.decode('UTF-8'))
#输出结果为响应 Headers
print("返回 Headers:",response.headers)
```

返回结果如图 5-4 所示。

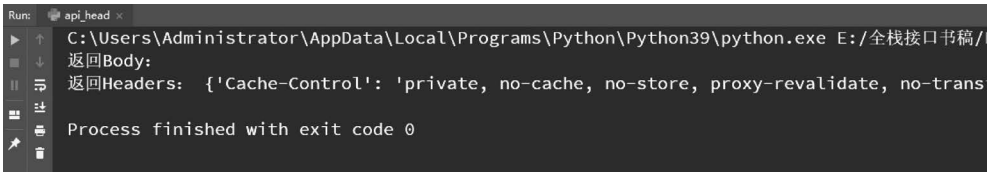

图 5-4 HEAD 请求返回结果

5.1.5 PATCH 方法的使用

在 HTTP 协议中,PATCH 方法用于对资源进行部分修改。与 POST 方法类似,PATCH 方法是非幂等的,这就意味着连续多个相同请求会产生不同的效果。在响应首部 Allow 或者 Access-Control-Allow-Methods 的方法列表中需要添加 PATCH 方法,这样基于 PATCH 方法的接口请求才会生效。在本节孔夫子旧书网首页接口请求中,响应 Headers 中的 Control-Allow-Methods 字段明确支持 GET、POST、OPTIONS 共 3 种方法进行请求,如图 5-5 所示。

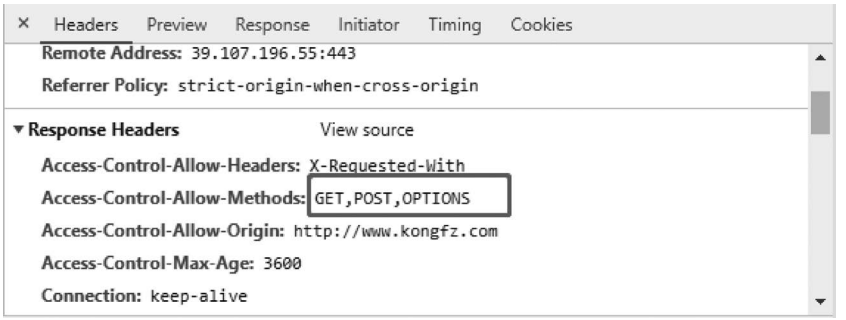

图 5-5 Control-Allow-Methods 字段示例

PATCH 方法与 PUT 方法类似,它们都可以直接修改服务器端的资源内容,是作为 HTTP 请求的补丁方法存在的。当对已有资源进行操作时,PATCH 方法常用于资源部分内容需要更新时使用,而 PUT 方法常用于更新目标资源完整内容时使用。当资源不存在时,PATCH 方法可以创建一个新的资源,而 PUT 方法无法完成这样的操作。有兴趣的读者可以参照 RFC5789 查看详细释义。

以 httpbin 网站 PATCH 方法接口为例，请求代码如下：

```
//chapter05/api_head.py

#导入第三方包 requests
import requests

#访问 httpbin 页面中的 patch 函数接口
url = 'http://httpbin.org/patch'

#发送 patch 请求，将返回结果存入变量 response
response = requests.patch(url)

#输出结果
print(response.json())
```

返回结果如图 5-6 所示。

图 5-6　PATCH 请求返回结果

5.2　基于 GET 方法的接口测试

GET 方法在实际接口请求时多用于静态 URL 资源请求，例如页面查看请求。当接口请求需要传参以获取动态展示页面时，也可使用 GET 方法来完成，例如关键字查询请求。GET 请求也可以传输实体的主体，但一般不用 GET 方法进行传输。

5.2.1　GET 方法参数解析

GET 方法请求以获取资源和查询资源为主要使用场景，因此在请求时传递参数并不复杂。在 api.py 文件中 GET 方法的构成如图 5-7 所示。

```
def get(url, params=None, **kwargs):
    r"""Sends a GET request.

    :param url: URL for the new :class:`Request` object.
    :param params: (optional) Dictionary, list of tuples or bytes to send
        in the query string for the :class:`Request`.
    :param \*\*kwargs: Optional arguments that ``request`` takes.
    :return: :class:`Response <Response>` object
    :rtype: requests.Response
    """

    return request("get", url, params=params, **kwargs)
```

图 5-7　GET 方法的构成

GET 方法使用参数源于 request 方法，常用参数如下。

（1）url 参数：Request 对象为必填参数，用于指明接口请求目标地址。

（2）params 参数：此参数为可选参数，当 GET 方法发送查询请求时用来传递查询参数使用，通常为字节类型或字典类型。

（3）** kwargs 参数：此参数为可选参数，可以传递 url 和 param 参数之外的其他类型参数，例如 cookies、headers、timeout 等接口请求中所需的参数。具体所支持的参数可在 api.py 文件下的 request 方法注释下查看。

5.2.2 基于 GET 方法的请求类型

1. 通过 URL 直接传参

当接口请求需要传递的参数较少且没有加密需求时，可以将参数直接通过 URL 进行传递。例如，通过中图网首页进行关键词“华软盛科技有限公司”搜索，请求代码如下：

```
//chapter05/url_get.py

#导入第三方包 requests、urllib.parse、re
import requests
import urllib.parse
import re

#查询文本
text = '华软盛科技有限公司'

#将汉字文本转换为 unicode 编码
urltext = text.encode('unicode-escape').decode()
#输出源文本
print(text)
#输出转换后的文本
print(urltext)

#使用中图网进行关键字查询
url = 'http://www.bookschina.com/book_find2/?stp=' + urltext + '&sCate=0'

#发送 GET 请求，将返回结果存入变量 response
response = requests.get(url=url)

#输出结果，title 标题中包含转码后的查询关键字
#使用正则表达式提取结果中的标题
pat = re.compile('<title>' + '(.*?)' + '</title>',re.S)
result = pat.findall(response.text)

#由于提取结果中包含汉字乱码，所以需要对结果进行解码处理
unurltext = urllib.parse.unquote(result[0])
print('URL 解码结果:' + unurltext)
```

```
#对提取结果查询关键字进行切片处理
sptext = unurltext.split(':')
#对切片后列表中的关键字 unicode 码进行解码处理
detext = sptext[1].encode().decode("unicode_escape")
#输出最终结果
print(detext)
```

返回结果如图 5-8 所示。

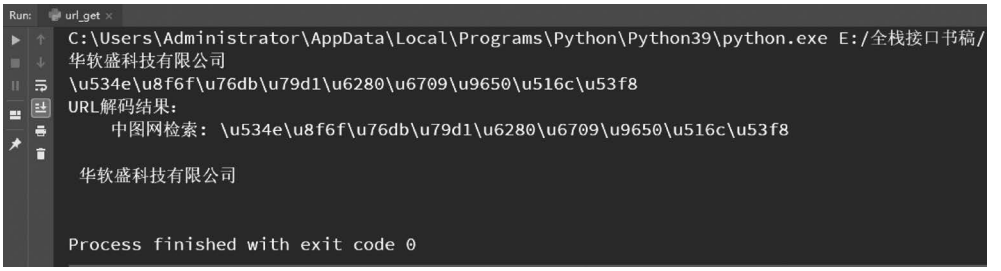

图 5-8　GET 查询请求返回结果

2. 通过字典方式传参

当接口请求需要传递的参数较多且将参数附在 URL 中进行传递时，参数不够直观。可以将参数以字典的方式进行存储，当发送 GET 方法进行接口请求时，使用 params 参数进行传递。修改文件 url_get.py 中的代码，在中图网首页对关键词“全栈 UI 自动化测试”进行搜索，请求代码如下：

```
//chapter05/url_get_param.py

#导入第三方包 requests、urllib.parse、re
import requests
import urllib.parse
import re

#查询文本
text = '全栈 UI 自动化测试实战'

#将汉字文本转换为 unicode 编码
urltext = text.encode('unicode-escape').decode()

#将传递参数存入字典
dict = {
    'stp':urltext,
    'sCate':'0'
}

#使用中图网进行关键字查询
url = 'http://www.bookschina.com/book_find2/'
```

```
#发送 GET 请求,通过 params 进行传参,将返回结果存入变量 response
response = requests.get(url = url,params = dict)

#输出结果,title 标题中包含转码后的查询关键字
#使用正则表达式提取结果中的标题
pat = re.compile('<title>' + '(.*?)' + '</title>',re.S)
result = pat.findall(response.text)

#由于提取结果中包含汉字乱码,所以需要对结果进行解码处理
unurltext = urllib.parse.unquote(result[0])
print('URL 解码结果:' + unurltext)

#对提取结果查询关键字进行切片处理
sptext = unurltext.split(':')
#对切片后列表中的关键字 unicode 码进行解码处理
detext = sptext[1].encode().decode("unicode_escape")
#输出最终结果
print(detext)
```

返回结果如图 5-9 所示。

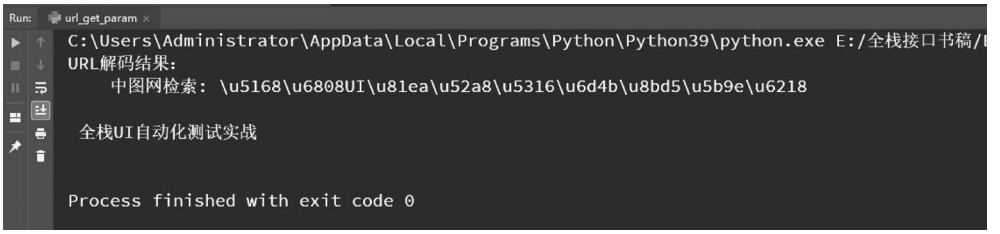

图 5-9 通过 params 传参请求返回结果

3. 其他常见传递参数

GET 方法接口请求还可以传递 URL、params 参数之外的其他参数,例如 Headers、timeout 等参数。

网站基于访问安全考虑,通常会在非登录验证接口加入访问验证,以避免工具对网站进行恶意访问。通过浏览器访问网站时,浏览器会在访问接口中自动加入客户端信息。当服务器端未开启对访问客户端信息验证时,接口请求脚本仅需 URL 即可完成访问;当服务器端开启了客户端信息验证时,在接口请求脚本 Headers 中需要加入 User-Agent 参数。有些网站也会在 Headers 中加入自定义参数,当使用脚本进行测试访问时,需带上自定义 Headers 参数。

访问中图网首页,发送 GET 方法接口请求需要带上客户端信息,请求代码如下:

```
//chapter05/url_get_header.py

#导入第三方包 requests
import requests
```

```
#定义 Headers 中需要传递的参数
Header = {
    'Host': 'www.bookschina.com',
    'Connection': 'keep-alive',
    'Upgrade-Insecure-Requests': '1',
    'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/109.0.0.0 Safari/537.36',
    'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/
webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9',
    'Referer': 'http://www.bookschina.com/RegUser/login.aspx',
    'Accept-Encoding': 'gzip, deflate',
    'Accept-Language': 'zh-CN,zh;q=0.9'
}

#访问中图网首页
url = 'http://www.bookschina.com/'

#发送 GET 请求,加入 Headers 参数,将返回结果存入变量 response
response = requests.get(url=url,headers=Header)

#输出结果
print(response.text)
```

返回结果如图 5-10 所示。

图 5-10　中图网首页请求返回结果

5.2.3　常见 Requests 响应参数

接口请求返回 Requests 响应对象，对象拥有部分属性及方法，用来满足请求目标。一些响应对象的常用属性及方法见表 5-2。

表 5-2　Requests 响应对象的常用属性及方法

参　数	数据类型	作　用
apparent_encoding	str(字符串型)	根据返回内容解析出来的字符编码，然后返回编码名称
close()	method	关闭与服务器的连接，仅对建立长链接的对象起作用
Cookies	RequestsCookieJar	获取返回 Cookie 值
content	bytes(字节型)	以 bytes 型返回原始响应体
elapsed	datetime.timedelta	返回从发送请求到接收到响应所花费的时长

续表

参　数	数据类型	作　用
encoding	str(字符串型)	返回用于解码 response.content 的编码方式,默认按照"ISO-8859-1"进行解码
history	list(列表)	访问历史记录(重定向记录)
headers	dict(字典)	返回 HTTP 响应头参数
is_permanent_redirect	bool(布尔型)	如果响应是永久重定向的 URL,则返回值为 True,否则返回值为 False
is_redirect	bool(布尔型)	如果响应已重定向,则返回值为 True,否则返回值为 False
iter_content()	method	以字节方式对响应数据进行迭代,当可以进行解码时返回
iter_lines()	method	以行方式对响应数据进行迭代
links	dict(字典)	返回标题链接
json()	method	返回转换成 JSON 格式的数据
next		返回重定向链中下一个请求的 PreparedRequest 对象
ok	bool(布尔型)	判断协议状态码是否小于 400,如果小于 400,则返回值为 True,如果不小于 400,则返回值为 False
raise_for_status()	method	抛出状态异常错误
raw	Object(对象)	返回请求后得到此响应对象的原始响应体对象,urllib 的 HTTPResponse 对象,通常使用 response.raw.read()进行读取
reason	str(字符串型)	返回 HTTP 响应状态码相对应的描述文本,例如 OK、Not Found 等
request	requests.models.PreparedRequest	返回对应的请求对象
status_code	int(整型)	返回 HTTP 请求常响应协议状态码,例如 200、302、404、500 等
text	str(字符串型)	返回经过编码后的文本内容
url	str(字符串型)	返回请求的真实 URL 网址

Requests 响应常用属性及方法的请求代码如下:

```
//chapter05/url_get_response.py

#导入第三方包 requests
import requests

#访问百度首页
url = 'https://www.baidu.com/'

#发送 GET 请求,将返回结果存入变量 response
response = requests.get(url)

#输出结果,此处仅列举几个常用属性,更多的响应属性及方法可参见表 5-1 进行练习
print('返回接口请求中的 URL:',response.url)
```

```
print('返回协议状态码:',response.status_code)
print('返回响应对象编码:',response.apparent_encoding)
print('返回 Cookie 对象:',response.cookies)
print('返回响应 Headers:',response.headers)
```

返回结果如图 5-11 所示。

图 5-11　Requests 响应参数示例结果

5.3　基于 POST 方法的接口测试

HTTP 协议规定当 POST 请求提交数据时需要放在消息主体(Body)中发送。常见的消息主体数据格式有 4 种,在发送请求时,需要在 Headers 中的 Content-Type 字段中说明消息主体所采用的编码方式,服务器端接收到请求后会采用相应的方式进行解码,以确保数据传输的正确性。

POST 请求将所有数据放在消息主体中传输,无法在 URL 中看到传输数据,与 GET 方法传输参数相比,具有一定的数据隐藏效果,例如 B/S 架构软件中用户登录请求通常会使用 POST 方法进行登录账号和密码数据传输。从应用层数据传输角度看,这仍然是明文传输。当 POST 方法传输的参数有敏感信息时,需要将参数进行加密处理,以密文方式进行传输。

5.3.1　POST 方法参数解析

POST 方法请求以传输数据为主要使用场景,因此在请求时传递参数会根据 Headers 中的 Content-Type 字段确定消息主体的编码方式。在 api.py 文件中 POST 方法的构成如图 5-12 所示。

```
def post(url, data=None, json=None, **kwargs):
    r"""Sends a POST request.

    :param url: URL for the new :class:`Request` object.
    :param data: (optional) Dictionary, list of tuples, bytes, or file-like
        object to send in the body of the :class:`Request`.
    :param json: (optional) json data to send in the body of the :class:`Request`.
    :param \*\*kwargs: Optional arguments that ``request`` takes.
    :return: :class:`Response <Response>` object
    :rtype: requests.Response
    """

    return request("post", url, data=data, json=json, **kwargs)
```

图 5-12　POST 请求参数

POST 方法使用的参数同样源于 request 方法，感兴趣的读者可以通过查看 api.py 文档中 Request 方法的传递参数集进行进一步了解。POST 方法常用的传递参数如下。

(1) url 参数：Request 对象的必填参数，用于指明接口请求目标地址。

(2) data 参数：此参数为二选一参数，当 POST 方法发送非 JSON 编码主体参数时此项为必填参数。当请求中有大量数据需要传递时使用。

(3) json 参数：此参数为二选一参数，当 POST 方法发送 JSON 编码主体参数时此项为必填参数，请求中有大量 JSON 数据需要传递时使用。

(4) ** kwargs 参数：此参数为可选参数，可以传递 url 和 data/json 参数之外的其他类型参数，例如 cookies、headers、timeout 等接口请求中所需的参数。

POST 请求的 4 种消息主体见表 5-3。

表 5-3 POST 请求的 4 种消息主体

Content-Type	参数规则	示例
multipart/form-data	数据以键-值对形式存在，使用分隔符 boundary 处理成一条消息 既可以上传文件，也可以上传参数	--boundary 分隔线 boundary-- Content-Disposition: form-data; name="file"; filename="test.txt" --boundary 分隔线 boundary--
x-www-form-urlencoded	数据以键-值对形式存在，是默认的 MIME 内容编码类型 只能上传键-值对，不能用于文件上传，参数之间以 & 符间隔	username=Thinkerbang& password=1234

(raw)
application/text
application/json
application/xml
application/html
	可以上传任意格式的文本，常见格式有 text、json、xml、html	(application/json 示例)
		{
		username: 'Thinkerbang',
		password: '1234'
		}

application/octet-stream	只可以上传二进制数据，通常用来上传文件 没有键-值对，一次只能上传一个文件	-- boundary 分隔线 boundary-- Content-Disposition: form-data; name="file"; filename="test.gif" Content-Type: application/octet-stream GIF89…二进制数据… -- boundary 分隔线 boundary--

5.3.2 消息主体：Data 类型实例

B/S 架构的软件在访问时，登录请求通常使用 POST 方法来完成。在软件操作过程中，涉及新增、修改等操作，通常使用 POST 方法。本节实例使用孔夫子旧书网进行演示，在 5.1.2 节 api_post.py 代码实现登录的基础上，对“个人中心”信息进行更新操作，请求代码如下：

```
//chapter05/api_post_data.py

import requests


# ------ 系统登录模块 start -------
url = 'https://login.kongfz.com/Pc/Login/account'

#将请求 Headers 以字典的方式存入变量 headers
headers = {
    "Connection":"keep-alive",
    "Content-Length":"150",
    "sec-ch-ua":"\" Not A;Brand\";v=\"99\", \"Chromium\";v=\"100\", \"Google Chrome\
";v=\"100\"",
    "sec-ch-ua-mobile":"?0",
    "User-Agent":"Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/100.0.4896.75 Safari/537.36",
    "X-Tingyun-Id":"OHEPtRD8z8s;r=325222859",
    "Content-Type":"application/x-www-form-urlencoded; charset=UTF-8",
    "Accept":"application/json, text/javascript, */*; q=0.01",
    "X-Requested-With":"XMLHttpRequest",
    "Sec-Fetch-Site":"same-origin",
    "Sec-Fetch-Mode":"cors",
    "Sec-Fetch-Dest":"empty",
    "Host":"login.kongfz.com"
}

#将 Body 以字典的方式存入变量 data
data = {
    "loginName":"Thinkerbang",        #此处为用户名参数,读者需要自行替换
    "loginPass":"123456",
    "captchaCode":"",
    "autoLogin":"0",
    "newUsername":"",
    "returnUrl":"https://user.kongfz.com/index.html",
    "captchaId":""
    }

#发送 POST 请求,将返回结果存入变量 response
response = requests.post(url=url,headers=headers,data=data)
```

```
#返回值类型是 JSON 字符串,因此使用 json()方法进行输出
print(response.json())
# ------ 系统登录模块 end -------

# ------ 个人信息修订模块 start -------
url_info = 'https://user.kongfz.com/User/perPro/update/'

#将请求 Headers 以字典的方式存入变量 header_info
header_info = {
    "Host": "user.kongfz.com",
    "Connection": "keep-alive",
    "Content-Length": "86",
    "sec-ch-ua": "\"Not_A Brand\";v=\"99\", \"Google Chrome\";v=\"109\", \"Chromium\";
v=\"109\"",
    "Accept": "text/plain, */*; q=0.01",
    "Content-Type": "application/x-www-form-urlencoded",
    "X-Requested-With": "XMLHttpRequest",
    "sec-ch-ua-mobile": "?0",
    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/109.0.0.0 Safari/537.36",
    "sec-ch-ua-platform": "\"Windows\"",
    "Origin": "https://user.kongfz.com",
    "Sec-Fetch-Site": "same-origin",
    "Sec-Fetch-Mode": "cors",
    "Sec-Fetch-Dest": "empty",
    "Referer": "https://user.kongfz.com/person/person_info.html",
    "Accept-Encoding": "gzip, deflate, br",
    "Accept-Language": "zh-CN,zh;q=0.9"
}

#将 Body 以字典的方式存入变量 data_info
data_info = {
    'pic':'8284%2F3108284.jpg',
    'sex':'man',
    'qqNum':'359407130',
    'birthday':'',
    'area':'',
    'sign':'Thinkerbang2',          #修改个人信息中的"个性签名"内容
    'intro':''
}
#发送 POST 请求,将返回结果存入变量 response_info,由于有登录操作,所以需要将登录后的
#Cookies 传入新的请求
response_info = requests.post(url=url_info,data=data_info,headers=header_info,Cookies=
response.cookies)

#返回值类型是 JSON 字符串,也可以使用 text 属性进行文本输出
print(response_info.text)

# ------ 个人信息修订模块 end -------
```

返回结果如图 5-13 所示。

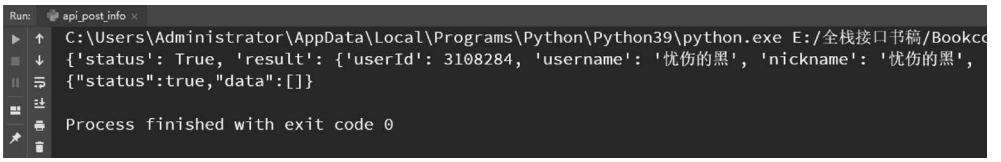

图 5-13　修改个人信息请求返回结果

5.3.3　消息主体：JSON 类型实例

POST 请求通过信息主体传递参数的第 2 种方法是使用 JSON 格式。当主体数据为字典格式时，使用 Data 和 JSON 参数都可以完成传递。使用 Data 参数时会将主体数据转换为 JSON 格式后进行传递，请求代码如下：

```
//chapter05//api_post_json.py

import requests
import json


url = 'http://httpbin.org/post'

#将请求 Headers 以字典的方式存入变量 headers
#Content-Type 属性指定 application/json 类型
headers = {

    "User-Agent":"Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/100.0.4896.75 Safari/537.36",
    "Content-Type":"application/json; charset=UTF-8",
    "Accept":"application/json, text/javascript, */*; q=0.01",
}

#将 Body 以字典的方式存入变量 data
data = {
    "username":"Thinkerbang",
    "password":"123456"
        }

#将字典格式参数转换成 JSON 格式
json_param = json.dumps(data)

#发送 POST 请求，使用 data 进行参数传递
response = requests.post(url=url,headers=headers,data=json_param)

#发送 POST 请求，使用 JSON 进行参数传递，json 参数自动将字典类型转换成 JSON 格式
response2 = requests.post(url=url,headers=headers,json=data)
```

```
#由于返回值类型是 JSON 字符串,因此使用 json()方法进行输出
print(response.json())

print(response2.json())
```

返回结果如图 5-14 所示。

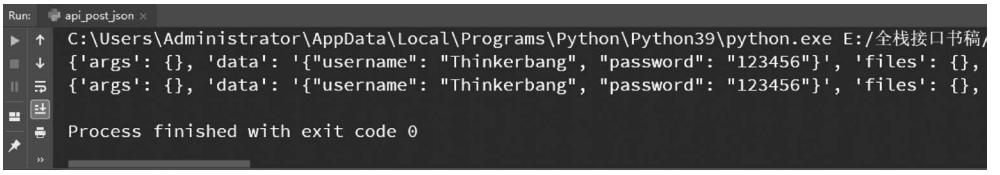

图 5-14 传递 JSON 参数请求返回结果

在 api_post_json.py 文件中引入了 JSON 数据处理包,当在接口请求中涉及 JSON 数据时用来对数据进行预处理,常用的处理方法见表 5-4。

表 5-4 常用 JSON 数据处理方法

方法名称	作用	示例
json.dumps()	用于将字典类型的数据转换成字符串类型	import json data = {'user':'Tom','sn':'1234'} jsObj = json.dumps(data)
	son.dumps()参数: r.json():JSON 格式数据转换; indent=True:JSON 格式数据序列化; ensure_ascii=False:JSON 格式数据中文处理	import json json.dumps(r.json(), indent=True, ensure_ascii=False)
json.dump()	用于将字典类型的数据转换成字符串类型,并写入 JSON 文件中	import json data = {'user':'Tom','sn':'1234'} filename = 'data.json' json.dump(data, open(filename, "w"))
json.loads()	用于将字符串类型的数据转换成字典类型	import json data= {'user':'Tom','sn':'1234'} jsDumps = json.dumps(data) jsLoads = json.loads(jsDumps)
json.load()	用于从 JSON 文件中读取数据,读取数据为字符串类型	import json filename = ('data.json') jsObj = json.load(open(filename))

5.3.4 消息主体:XML 类型实例

XML 扩展标记语言属于质量级数据交互格式。在 JSON 数据格式出现之前,XML

由于格式统一、符合数据传输标准曾被广泛使用，但是当使用 XML 格式做数据传输时其缺点也比较明显，例如文本格式复杂、体量大、占带宽，以及需要使用大量代码进行解析等。作为 POST 方法信息主体格式，XML 至今仍在部分项目接口中使用。请求代码如下：

```
//chapter05//api_post_xml.py

import requests


url = 'http://www.testingedu.com.cn:8081/inter/SOAP?wsdl'

#将请求 Headers 以字典的方式存入变量 headers
#Content-Type 属性指定 text/xml 类型
headers = {

    "User-Agent":"Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/100.0.4896.75 Safari/537.36",
    "Content-Type":"text/xml; charset=UTF-8",
    "token":"33bd50b3eb55439f89e328693fc02997",
    "cookie":"userid=12955; token=33bd50b3eb55439f89e328693fc02997; type=SOAP",
}

#将 Body 以字符串的方式存入变量 data
data = "<soapenv:Envelope xmlns:soapenv=\"http://schemas.xmlsoap.org/soap/envelope/\" xmlns:
soap=\"http://soap.testingedu.com/\"><soapenv:Header/><soapenv:Body><soap:login><arg0>
Will</arg0><arg1>123456</arg1></soap:login></soapenv:Body></soapenv:Envelope>"


#发送 POST 请求，使用 data 进行参数传递
response = requests.post(url=url,headers=headers,data=data)

#输出响应结果
print(response.text)
print("返回协议状态码:",response.status_code)
```

返回结果如图 5-15 所示。

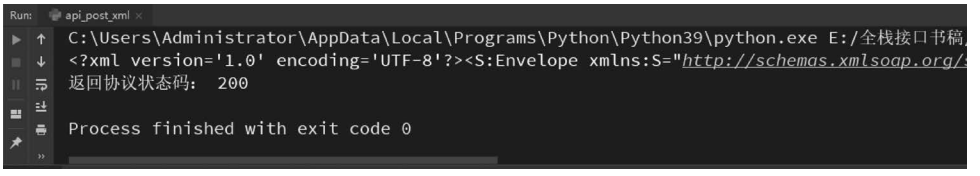

图 5-15　传递 XML 参数请求返回结果

5.3.5　消息主体：File 类型实例

POST 方法可用于消息主体参数传递，File 类型是一种比较特殊的存在。文件传递方

式与文件类型、文件大小有关。本节使用《全栈 UI 自动化测试实战》一书 7.3.1 节文件上传实例进行演示。网址为 http://sahitest.com/demo/php/fileUpload.php，请求代码如下：

```
//chapter05//api_post_file.py

import requests


url = 'http://sahitest.com/demo/php/fileUpload.php'

#将请求 Headers 以字典的方式存入变量 headers
#将 Content-Type 属性注释掉

headers = {
    'Host': 'sahitest.com',
    'Connection': 'keep-alive',
    'Content-Length': '397',
    'Cache-Control': 'max-age=0',
    'Upgrade-Insecure-Requests': '1',
    'Origin': 'http://sahitest.com',
#'Content-Type': 'multipart/form-data; boundary=----WebKitFormBoundary1TdGTedZL9dWTj6q',
    'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/110.0.0.0 Safari/537.36',
    'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/
webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7',
    'Referer': 'http://sahitest.com/demo/php/fileUpload.htm',
    'Accept-Encoding': 'gzip, deflate',
    'Accept-Language': 'zh-CN,zh;q=0.9'
    }

#将抓取到的消息主体转换成 files 与 data 参数

files={
        "file": open("test.txt", "rb"),
        "Content-Type": "text/plain",
        "Content-Disposition": "form-data",
        "filename": "test.txt"
    }

data={
        "multi": "false",
        "submit":"Submit Single"
    }

#发送 POST 请求，使用 data、files 分别进行参数传递
response = requests.post(url=url,headers=headers,data=data,files=files)

#输入响应结果
print(response.text)
```

返回结果如图 5-16 所示。

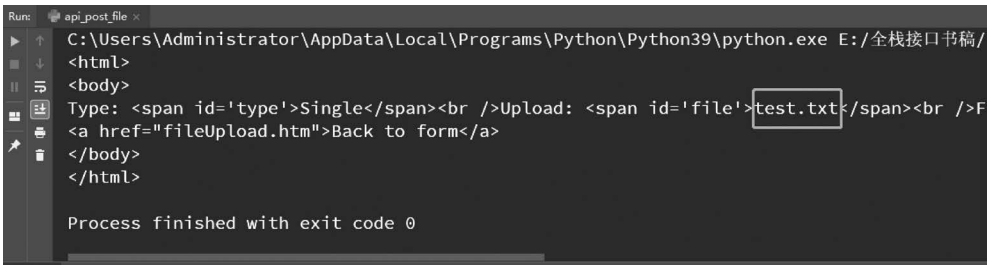

图 5-16　上传文件请求返回结果

示例代码中涉及抓取上传消息主体参数的处理，此处进行数据转换说明。

第 1 步，在浏览器中打开网址 http://sahitest.com/demo/php/fileUpload.php，进行文件上传操作，使用 Fiddler 同步抓取接口信息，请求消息主体如图 5-17 所示。

图 5-17　上传文件抓包结果

第 2 步，Request 请求区被切换为 WebForms 选项卡，以表单方式查看消息主体传递参数，如图 5-18 所示。

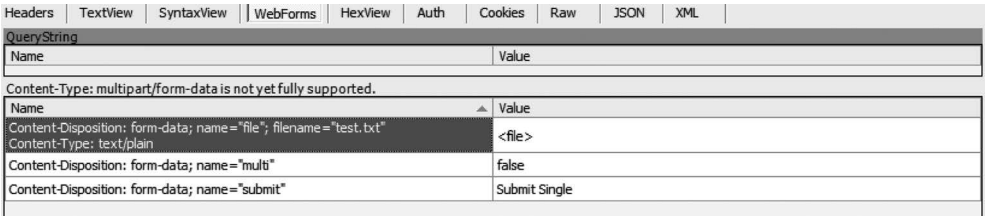

图 5-18　消息主体参数以表单显示结果

第 3 步，第 1 项中包含的 4 个参数是 files 参数的组成要素，将参数转换为字典格式，其中 file 子参数主体使用 open()方法引入待传文件。第 2、第 3 项中的参数是 data 参数的组成要素，将参数转换为字典格式，转换结果见 api_post_file.py。

5.4 接口测试常用方法

5.4.1 Cookies 的传递

5.3.2 节中的 api_post_data.py 文件中的代码修订了个人信息接口，需要用户登录将返回的 Cookies 信息传递进新的请求，代码未做函数化处理，因此不涉及参数的获取与传递。将用户登录及更改个人信息接口代码转换为方法调用方式，请求代码如下：

```
//chapter05//api_Cookies.py

import requests

def login():
    url = 'https://login.kongfz.com/Pc/Login/account'
    header = {
        'User-Agent': 'Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:70.0) Gecko/20100101
Firefox/70.0',
        'Content-Type': 'application/x-www-form-urlencoded; charset=UTF-8',
        'X-Requested-With': 'XMLHttpRequest',
        'X-Tingyun-Id': 'OHEPtRD8z8s;r=500843061',
        'Origin': 'https://login.kongfz.com',
        'Referer': 'https://login.kongfz.com/'

    }
    body = {
        'loginName': 'thinkerbang',        #此处为用户名参数,读者需要自行替换
        'loginPass': '123456',
        'captchaCode': '',
        'autoLogin': '0',
        'newUsername': '',
        'returnUrl': '',
        'captchaId': ''
    }
    r = requests.post(url=url,headers=header,data=body)
    print(r.json())
    return r.cookies

def mod_info(cook):
    url_info = 'https://user.kongfz.com/User/perPro/update/'

    #将请求 Headers 以字典的方式存入变量 header_info
    header_info = {
        "Host": "user.kongfz.com",
        "Connection": "keep-alive",
        "Content-Length": "86",
```

```
            "sec-ch-ua": "\"Not_A Brand\";v=\"99\", \"Google Chrome\";v=\"109\",
\"Chromium\";v=\"109\"",
            "Accept": "text/plain, */*; q=0.01",
            "Content-Type": "application/x-www-form-urlencoded",
            "X-Requested-With": "XMLHttpRequest",
            "sec-ch-ua-mobile": "?0",
            "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/109.0.0.0 Safari/537.36",
            "sec-ch-ua-platform": "\"Windows\"",
            "Origin": "https://user.kongfz.com",
            "Sec-Fetch-Site": "same-origin",
            "Sec-Fetch-Mode": "cors",
            "Sec-Fetch-Dest": "empty",
            "Referer": "https://user.kongfz.com/person/person_info.html",
            "Accept-Encoding": "gzip, deflate, br",
            "Accept-Language": "zh-CN,zh;q=0.9"
        }

        #将 Body 以字典的方式存入变量 data_info
        data_info = {
            'pic': '8284%2F3108284.jpg',
            'sex': 'man',
            'qqNum': '359407130',
            'birthday': '',
            'area': '',
            'sign': 'Thinkerbang2',          #修改个人信息中的“个性签名”内容
            'intro': ''
        }
        #发送 POST 请求,将返回结果存入变量 response_info,Cookies 信息通过形参 cook 传入
        response_info = requests.post(url=url_info, data=data_info, headers=header_info,
Cookies=cook)

        #返回值类型是 JSON 字符串,也可以使用 text 属性进行文本输出
        print(response_info.text)

    #调用 login()方法,将返回的 Cookies 信息存入 cook 变量
    cook = login()

    #调用修改个人信息方法
    mod_info(cook)
```

返回结果见 5.3.2 节的执行结果,如图 5-13 所示。

在 api_Cookies.py 文件中两个接口方法之间存在耦合性,在基于测试框架维护测试脚本时,通常使用 RequestsCookieJar 对象进行 Cookies 参数传递,请求代码如下:

```
//chapter05//api_Cookies_obj.py



import requests
#声明全局变量
```

```
global cook

#将变量存入 RequestsCookieJar 对象容器,用来传递 Cookies 参数
cook = requests.cookies.RequestsCookieJar()

def login():
    url = 'https://login.kongfz.com/Pc/Login/account'
    header = {
        'User-Agent': 'Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:70.0) Gecko/20100101
Firefox/70.0',
        'Content-Type': 'application/x-www-form-urlencoded; charset=UTF-8',
        'X-Requested-With': 'XMLHttpRequest',
        'X-Tingyun-Id': 'OHEPtRD8z8s;r=500843061',
        'Origin': 'https://login.kongfz.com',
        'Referer': 'https://login.kongfz.com/'

    }
    body = {
        'loginName': 'thinkerbang',          #此处为用户名参数,读者需要自行替换
        'loginPass': '123456',
        'captchaCode': '',
        'autoLogin': '0',
        'newUsername': '',
        'returnUrl': '',
        'captchaId': ''
    }
    r = requests.post(url=url,headers=header,data=body)
    print(r.json())

    #引入全局变量 cook
    global cook

    #将用户登录后的 Cookies 信息存入变量
    cook = r.cookies

def mod_info(cook):
    url_info = 'https://user.kongfz.com/User/perPro/update/'

    #将请求 Headers 以字典的方式存入变量 header_info
    header_info = {
        "Host": "user.kongfz.com",
        "Connection": "keep-alive",
        "Content-Length": "86",
        "sec-ch-ua": "\"Not_A Brand\";v=\"99\", \"Google Chrome\";v=\"109\", \"
Chromium\";v=\"109\"",
        "Accept": "text/plain, */*; q=0.01",
        "Content-Type": "application/x-www-form-urlencoded",
        "X-Requested-With": "XMLHttpRequest",
        "sec-ch-ua-mobile": "?0",
```

```
        "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/109.0.0.0 Safari/537.36",
        "sec-ch-ua-platform": "\"Windows\"",
        "Origin": "https://user.kongfz.com",
        "Sec-Fetch-Site": "same-origin",
        "Sec-Fetch-Mode": "cors",
        "Sec-Fetch-Dest": "empty",
        "Referer": "https://user.kongfz.com/person/person_info.html",
        "Accept-Encoding": "gzip, deflate, br",
        "Accept-Language": "zh-CN,zh;q=0.9"
    }

    #将 Body 以字典的方式存入变量 data_info
    data_info = {
        'pic': '8284%2F3108284.jpg',
        'sex': 'man',
        'qqNum': '359407130',
        'birthday': '',
        'area': '',
        'sign': 'Thinkerbang2',        #修改个人信息中的"个性签名"内容
        'intro': ''
    }
    #发送 POST 请求,将返回结果存入变量 response_info,Cookies 信息通过形参 cook 传入
    response_info = requests.post(url=url_info, data=data_info, headers=header_info,
Cookies=cook)

    #返回值类型是 JSON 字符串,也可以使用 text 属性进行文本输出
    print(response_info.text)

#调用 login()方法,将返回的 Cookies 信息存入 cook 变量
login()

#调用修改个人信息方法
mod_info(cook)
```

返回结果见 5.3.2 节的执行结果,如图 5-13 所示。

5.4.2　身份认证

HTTP 有 3 种身份认证方式：基础身份认证、netrc 认证、摘要式身份认证。

基础身份认证是 HTTP1.0 提出的认证方式,客户端对于每个需要授权访问的请求,通过提供用户名和密码进行认证,示例代码如下：

```
//chapter05//api_auth.py


from requests.auth import HTTPBasicAuth
import requests
```

```
url = 'http://192.168.1.42/index.html'
header = {
    'Authorization': 'Basic dXNlcjoxMjM0NTY='
    }
# ------------第 1 种基础身份认证传输--------------------

r = requests.get(url=url,headers=header,auth=HTTPBasicAuth('user','123456'))

#输出返回主体及状态码
print("第 1 种基础认证返回主体:",r.text)
print("第 1 种基础认证返回状态码:",r.status_code)

# ------------第 2 种基础身份认证传输--------------------

url2 = 'http://user:123456@192.168.1.42/index.html'

r2 = requests.get(url=url2,headers=header)

#输出返回主体及状态码
print("第 2 种基础认证返回主体:",r2.text)
print("第 2 种基础认证返回状态码:",r2.status_code)
```

返回结果如图 5-19 所示。

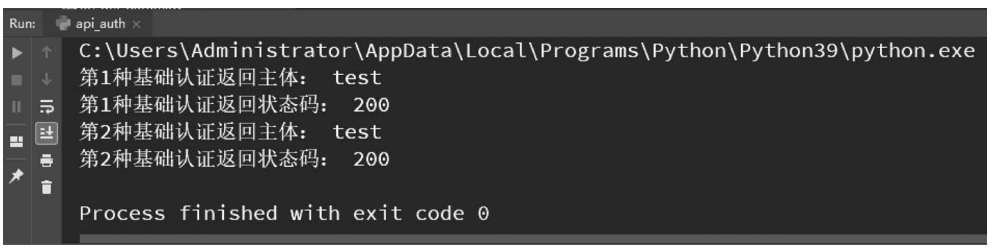

图 5-19 基础身份认证返回结果

5.4.3 生成测试执行报告

测试结果的输出通常以 HTML 页面或文档方式进行图文展示。在 unittest 测试框架部分会引入 HTML 页面以展示测试结果。本节引入文档，以便展示测试执行结果。

本节引入 reportlab 库，以便生成报告。reportlab 是 Python 的一个标准库，可以生成图文形式的文档。

在命令行下输入 pip install -i https://pypi.tuna.tsinghua.edu.cn/simple python-office -U 命令后便可安装 reportlab，安装过程如图 5-20 所示。

安装完成后，在 Python 下输入 import reportlab 命令进行验证。

首先，准备生成测试执行报告所需的基础图表类文件，代码如下：

图 5-20　reportlab 的安装

```
//chapter05//report_Graphs.py

from reportlab.pdfbase import pdfmetrics
from reportlab.pdfbase.ttfonts import TTFont
from reportlab.platypus import Table, Paragraph
from reportlab.lib.styles import getSampleStyleSheet
from reportlab.lib import colors
from reportlab.graphics.charts.barcharts import VerticalBarChart
from reportlab.graphics.charts.legends import Legend
from reportlab.graphics.shapes import Drawing

#注册字体,此处使用微软雅黑
pdfmetrics.registerFont(TTFont('MYH', 'MSYHL.TTC'))

class Graphs():
    #绘制标题
    @staticmethod
    def draw_title(title: str):
        #获取样式表
        style = getSampleStyleSheet()
        #标题样式
        ct = style['Italic']
        #单独设置样式相关属性
        ct.fontName = 'MYH'                    #字体名
        ct.fontSize = 18                       #字体大小
        ct.leading = 50                        #行间距
        ct.textColor = colors.black            #字体颜色
        ct.alignment = 1                       #居中
        ct.bold = True
        #创建标题对应的段落,并且返回
        return Paragraph(title, ct)

    #绘制小标题
    @staticmethod
    def draw_little_title(title: str):
        #获取样式表
        style = getSampleStyleSheet()
```

```
        #标题样式
        ct = style['Normal']
        #设置样式属性
        ct.fontName = 'MYH'                    #字体名
        ct.fontSize = 15                       #字体大小
        ct.leading = 30                        #行间距
        ct.textColor = colors.black            #字体颜色
        ct.alignment = 1                       #对齐方式
        #创建标题对应的段落,并且返回
        return Paragraph(title, ct)

    #绘制普通段落内容
    @staticmethod
    def draw_text(text: str):
        #获取样式表
        style = getSampleStyleSheet()
        #获取普通样式
        ct = style['Normal']
        ct.fontName = 'MYH'
        ct.fontSize = 12
        ct.wordWrap = 'CJK'                    #设置自动换行
        ct.alignment = 0                       #居左对齐
        ct.firstLineIndent = 32                #第 1 行开头空格
        ct.leading = 25
        return Paragraph(text, ct)

    #绘制表格
    @staticmethod
    def draw_table(*args):
        #列宽度
        col_width = 80
        style = [
            ('FONTNAME', (0, 0), (-1, -1), 'MYH'),          #字体
            ('FONTSIZE', (0, 0), (-1, 0), 10),              #第 1 行的字体大小
            ('FONTSIZE', (0, 1), (-1, -1), 10),             #第 2 行到最后一行的字体大小
            ('BACKGROUND', (0, 0), (-1, 0), '#d5dae6'),     #设置第 1 行的背景颜色
            ('ALIGN', (0, 0), (-1, -1), 'CENTER'),          #第 1 行水平居中
            ('ALIGN', (0, 1), (-1, -1), 'CENTER'),
            #第 2 行到最后一行横向左对齐
            ('VALIGN', (0, 0), (-1, -1), 'MIDDLE'),         #所有表格纵向居中对齐
            #设置表格内文字的颜色
            ('TEXTCOLOR', (0, 0), (-1, -1), colors.darkslategray),
            #将表格框线设置为 grey 色,线宽为 0.5
            ('GRID', (0, 0), (-1, -1), 0.5, colors.grey),
        ]
        table = Table(args, colWidths=col_width, style=style)
        return table

    #创建图表
    @staticmethod
```

```
    def draw_bar(bar_data: list, ax: list, items: list):
        drawing = Drawing(500, 250)
        bc = VerticalBarChart()
        bc.x = 45                                #整个图表的 x 坐标
        bc.y = 45                                #整个图表的 y 坐标
        bc.height = 200                          #图表的高度
        bc.width = 350                           #图表的宽度
        bc.data = bar_data
        bc.strokeColor = colors.black            #顶部和右边轴线的颜色
        bc.valueAxis.valueMin = 0                #设置 y 坐标的最小值
        bc.valueAxis.valueMax = 10               #设置 y 坐标的最大值
        bc.valueAxis.valueStep = 1               #设置 y 坐标的步长
        bc.categoryAxis.labels.dx = 1
        bc.categoryAxis.labels.dy = -5
        bc.categoryAxis.labels.angle = 0
        bc.categoryAxis.categoryNames = ax

        #图示
        leg = Legend()
        leg.fontName = 'MYH'
        leg.alignment = 'right'
        leg.boxAnchor = 'ne'
        leg.x = 475
        leg.y = 240
        leg.dxTextSpace = 10
        leg.columnMaximum = 3
        leg.colorNamePairs = items
        drawing.add(leg)
        drawing.add(bc)

        return drawing
```

执行生成测试报告代码，示例代码中测试用例执行数据是为了演示使用 reportlab 工具包生成 PDF 文件而设置的，在测试框架章节会对生成过程进行封装处理，代码如下：

```
//chapter05//api_report.py

from chapter05.report_Graphs import Graphs
from reportlab.lib.pagesizes import letter
from reportlab.lib import colors
from reportlab.platypus import SimpleDocTemplate

if __name__ == '__main__':
    #创建内容对应的空列表
    content = list()

    #添加文档标题
    content.append(Graphs.draw_title('接口测试结果统计'))

    #添加文档说明文字
    content.append(Graphs.draw_text('本段文字是接口测试用例执行说明内容.'))
```

```
    #添加小标题
    content.append(Graphs.draw_title(''))
    content.append(Graphs.draw_little_title('接口测试结果统计表'))

    #添加接口用例执行结果统计数据表
    data = [
        ('接口名称', '执行情况', '执行总次数', '通过次数', '失败次数'),
        ('登录接口', '已执行', '8', '8', '0'),
        ('首页查询接口', '已执行', '8', '7', '1'),
        ('信息添加接口', '已执行', '8', '5', '3')
    ]
    content.append(Graphs.draw_table(*data))

    #生成图表
    content.append(Graphs.draw_title(''))
    content.append(Graphs.draw_little_title('接口用例执行结果图示'))
    b_data = [(8, 7, 5), (0, 1, 3)]
    ax_data = ['UserLogin', 'IndexSelect', 'AddInfo']
    leg_items = [(colors.green, '执行通过'),(colors.red, '执行失败') ]
    content.append(Graphs.draw_bar(b_data, ax_data, leg_items))

    #生成 PDF 文件
    doc = SimpleDocTemplate('report.pdf', pagesize=letter)
    doc.build(content)
```

执行结果生成的 PDF 文件如图 5-21 所示。

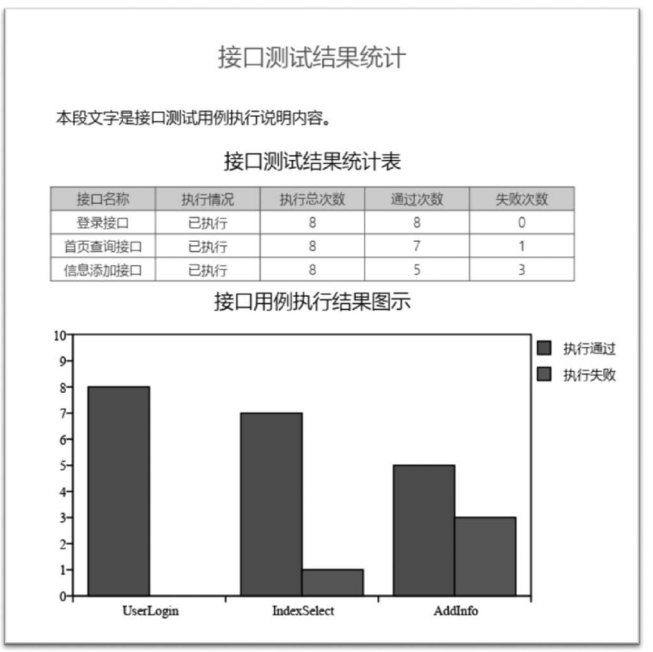

图 5-21　生成测试报告