requests基本使用

实例引入

1. response.status_code：获取响应码
2. response.text：获取返回内容，是纯文本格式
3. response.cookies：获取返回的cookies内容

```
import requests

response = requests.get('https://www.baidu.com/')
print(type(response))
print(response.status_code)
print(type(response.text))
print(response.text)
print(response.cookies)
```

各种请求方式

支持所有的请求方式，最常使用的就是：get，post，put，delete，对应增删改查

```
import requests
requests.get('http://httpbin.org/get')
requests.post('http://httpbin.org/post')
requests.put('http://httpbin.org/put')
requests.delete('http://httpbin.org/delete')
requests.head('http://httpbin.org/get')
requests.options('http://httpbin.org/get')
```

```
<Response [200]>
```

请求

基本GET请求

基本写法

```
import requests

response = requests.get('http://httpbin.org/get')
print(response.text)
```

```
{
  "args": {},
  "headers": {
    "Accept": "*/*",
    "Accept-Encoding": "gzip, deflate",
    "Host": "httpbin.org",
    "User-Agent": "python-requests/2.18.1"
  },
  "origin": "113.87.180.82, 113.87.180.82",
  "url": "https://httpbin.org/get"
}
```

带参数GET请求

1. 第一种方式：直接放到url里，name=germey&age=22，通过&符号隔开

```
import requests
response = requests.get("http://httpbin.org/get?name=duocceshi&age=22")
print(response.text)
```

```
{
  "args": {
    "age": "22",
    "name": "duocceshi"
  },
  "headers": {
    "Accept": "*/*",
    "Accept-Encoding": "gzip, deflate",
    "Host": "httpbin.org",
    "User-Agent": "python-requests/2.18.1"
  },
  "origin": "113.118.187.109, 113.118.187.109",
  "url": "https://httpbin.org/get?name=duocceshi&age=22"
}
```

2. 第二种方式：通过**params**关键字参数，传入一个python**字典**

```python
import requests

data = {
    'name': 'germey',
    'age': 22
}
response = requests.get("http://httpbin.org/get", params=data)
print(response.text)
```

解析json

1. python提供json库对json数据进行序列化和反序列化
2. requests提供 json()方法，直接获取字典数据

```python
import requests
import json

response = requests.get("http://httpbin.org/get")
# 直接返回是纯文本格式
print(type(response.text))
# 通过json()方法取到的是字典格式，实际是json()方法将 纯文本 转换成了 字典格式
print(response.json())
print(type(response.json()))

# 可以通过json库的loads()函数来进行反序列化
a = json.loads(response.text)
print(a, type(a))
```

```
<class 'str'>
{'args': {}, 'headers': {'Accept': '*/*', 'Accept-Encoding': 'gzip, deflate',
'Host': 'httpbin.org', 'User-Agent': 'python-requests/2.18.1'}, 'origin':
'113.89.236.102, 113.89.236.102', 'url': 'https://httpbin.org/get'}
<class 'dict'>
{'args': {}, 'headers': {'Accept': '*/*', 'Accept-Encoding': 'gzip, deflate',
'Host': 'httpbin.org', 'User-Agent': 'python-requests/2.18.1'}, 'origin':
'113.89.236.102, 113.89.236.102', 'url': 'https://httpbin.org/get'} <class
'dict'>
```

获取二进制数据

当请求资源是图片/视频内容的时候，需要获取它的二进制数据，将其下载到文件

```
import requests

response = requests.get("https://github.com/favicon.ico")
print(type(response.text), type(response.content))
print(response.text)
print(response.content)
```

发起请求，然后将图片下载到本地

```
import requests

response = requests.get("https://github.com/favicon.ico")
with open('favicon.ico', 'wb') as f:
    f.write(response.content)
    f.close()
```

添加headers

针对某些请求，必须添加headers才能正常请求

```
import requests
# 不使用headers，服务器会识别到你是python客户端发起到请求，就会拒绝
response = requests.get("https://www.zhihu.com/explore")
print(response.text)
```

```
import requests
# 通过headers伪装请求
headers = {
    'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_4)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/52.0.2743.116 Safari/537.36'
}
response = requests.get("https://www.zhihu.com/explore", headers=headers)
print(response.text)
```

如果是json数据格式，也需要声明请求头

```
import requests
url_sign = "http://127.0.0.1:5001/v1/signature"
data_add = {
    "signature": "测试签名创建",  # 必填
    "source": "深圳南山",  # 必填
    "pics": ["pic1"]  # 选填
}
# 通过headers指定请求数据类型
headers = {
    "Content-Type": "application/json",
}
```

```
auth = ("panwj", "123")
response = requests.post(url_sign, headers=headers, auth=auth, json=data_add)
print(response.text)
```

```
{
  "sign_id": 140
}
```

基本POST请求

1. post请求直接通过data关键字参数，发送body，服务器得到的是form数据

```
import requests

data = {'name': 'duocceshi', 'age': '22'}
response = requests.post("http://httpbin.org/post", data=data)
print(response.text)
```

2. post请求也可以指定headers

```
import requests

data = {'name': 'duocceshi', 'age': '22'}
headers = {
    'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_4)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/52.0.2743.116 Safari/537.36'
}
response = requests.post("http://httpbin.org/post", data=data,
headers=headers)
# 转成字典
print(response.json())
```

```
{'args': {}, 'data': '', 'files': {}, 'form': {'age': '22', 'name': 'germey'},
'headers': {'Accept': '*/*', 'Accept-Encoding': 'gzip, deflate', 'Content-
Length': '18', 'Content-Type': 'application/x-www-form-urlencoded', 'Host':
'httpbin.org', 'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_4)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/52.0.2743.116 Safari/537.36'},
'json': None, 'origin': '113.89.236.102, 113.89.236.102', 'url':
'https://httpbin.org/post'}
```

3. 如果是json数据格式，也需要声明请求头

```
import requests
url_sign = "http://127.0.0.1:5001/v1/signature"
```

```
data_add = {
    "signature": "测试签名创建",  # 必填
    "source": "深圳南山",  # 必填
    "pics": ["pic1"]  # 选填
}
# 通过headers指定请求数据类型
headers = {
    "Content-Type": "application/json",
}
auth = ("dcs", "123")
# 使用请求头
# response = requests.post(url_sign, headers=headers, auth=auth,
json=data_add)
# 不使用请求头，会提示错误，开发自己定义的
response = requests.post(url_sign, auth=auth, data=data_add)
print(response.text)
```

```
{"error": "ER:0005", "message": "request headers fail"}
```

练习

1. 动手写一个delete请求，删除签名

```
import requests
url = 'http://127.0.0.1:5001/v1/signature'
data_del = {
    "sign_id": 246,
}
headers = {
    "Content-Type": "application/json",
}
res = requests.delete(url, auth=('dcs', '123'), data=json.dumps(data_del),
headers=headers)
print(res.status_code, res.text)
```

```
403 {"error": "ER:0012", "message": "delete sign fail"}
```

2. 扩展：把上述过程，封装成一个方法，放到class类中

注意：因为delete接口比较特殊，成功时，返回"ok"，失败时，返回的是json格式，所以不能直接使用ts.check_fail()来进行校验

```
    def req_delete(self, url_type, data, user='dcs', passwd='123', *,
headers=None):
        url = self.choice_url(url_type)
        if headers:
            hd = headers
```

```
        else:
            hd = self.headers
        # 如果签名id存在，才执行删除操作
        response = requests.delete(url, json=data, auth=(user, passwd),
headers=hd)
        self.status_code = response.status_code
        logging.info("[返回码是:] {}".format(self.status_code))
        self.text = response.text
        logging.info("[返回内容是]: {}".format(self.text))
        return self.text
```

3. 编写一个用例，删除签名前先判断一次签名是否存在，然后执行删除，最后再进行判断：
    - 断言delete接口是否返回成功，包括status_code和text
    - 通过查询接口判断sign_id是否已经删除

```
    def signid_exist(self, sign_id, user, passwd):
        """判断签名id是否存在"""
        res = self.tsms_get("sign", user=user, passwd=passwd)
        # 注意：get请求返回的key是id
        sign_ids = self.recur("id", res)
        if sign_id in sign_ids:
            return True
        else:
            return False
```

```
# 删除签名
sign_id = 252
dele_sign = {"sign_id": sign_id}
if ts.signid_exist(sign_id, 'dcs', '123'):
    ts.req_delete('sign', dele_sign)
    assert ts.status_code == 200
    assert ts.text == 'ok'
    # 通过接口判断结果是否已经删除
    a = ts.signid_exist(sign_id, 'dcs', '123')
    assert a == False, "sign_id仍然存在"
else:
    print("签名id不存在")
```

4. 先创建签名，再删除签名，再进行结果校验

```
# 完整删除用例
# 创建签名
r = ts.req_post('sign', ts.sign_data())
# 获取签名id
sign_id = r["sign_id"]
# 构造删除接口的请求body
```

```
dele_sign = {"sign_id": sign_id}
# 发起删除请求
ts.req_delete('sign', dele_sign)
# 断言删除请求的返回结果
assert ts.status_code == 200
assert ts.text == 'ok'
# 通过接口判断结果是否已经删除
a = ts.signid_exist(sign_id, 'dcs', '123')
assert a == False, "sign_id仍然存在"
```

5. 动手写一个审核的过程

```
import requests
url = 'http://127.0.0.1:5001/v2/signreview'
data = {
    "sign_id": 249,
    "audit_status": "passed",
    "reject_reason": "",
}
headers = {
    "Content-Type": "application/json",
}
res = requests.post(url, auth=('root', '123'), data=json.dumps(data),
headers=headers)
print(res.status_code, res.text)
```

```
200 ok
```

6. 将审核功能，封装成方法，同时兼容签名和模版审，要求：
    - 兼容模版审核
    - 可以指定指定：审核id，审核状态，拒绝原因(默认不填写)

```
    def review(self, review_type, review_id, audit_status,
reject_reason=None):
        data = {
            "audit_status": audit_status,
            "reject_reason": reject_reason
        }
        if review_type == "sign":
            url = self.sign_review_url
            data["sign_id"] = review_id
        elif review_type == "temp":
            url = self.temp_review_url
            data["temp_id"] = review_id
        else:
            logging.error("[审核参数有误]: {}".format(review_type))
            return
```

```
        logging.info("[请求地址是]: {}".format(url))
        res = requests.post(url, json=data, auth=(self.root, self.root_pass),
headers=self.headers)
        logging.info("[审核结果是]: {} {} ".format(res.status_code, res.text))
        return self.text
```

7. 写一个解析方法，解析get请求的结果，传入签名id=214，即可以得到该id对应的审核状态

```
    def get_field(self, field, **kwargs):
        """解析get请求的指定字段"""
        for k, v in kwargs.items():
            for d in self.json["items"]:
                if d[k] == v:
                    return d[field]
        logging.error("[未找到指定的字段]: {}".format(field))
```

8. 写一个用例，审核签名后，断言该签名的审核结果

```
sign_id = 214
ts.review('sign', sign_id, 'rejected')
# 执行查询
ts.tsms_get('sign')
# 解析查询结果
b = ts.get_field("audit_status", id=sign_id)
# 断言
assert b == 'rejected', "审核结果和预期不一致"
```

9. 完善用例，先创建签名，再审核签名，最后断言签名的审核结果

```
res = ts.req_post("sign", ts.sign_data())
sign_id = res["sign_id"]
ts.review('sign', sign_id, 'rejected')
# 执行查询
ts.tsms_get('sign')
# 解析查询结果
b = ts.get_field("audit_status", id=sign_id)
# 断言
assert b == 'rejected', "审核结果和预期不一致"
```

10. 将编辑签名功能，写成一个方法

```
    def req_put(self, url_type, data, user='dcs', passwd='123', *,
headers=None):
        url = self.choice_url(url_type)
        if headers:
            hd = headers
        else:
            hd = self.headers
        response = requests.put(url, json=data, auth=(user, passwd),
headers=hd)
        self.status_code = response.status_code
        logging.info("[req_put返回码是:] {}".format(self.status_code))
        self.json = response.json()
        logging.info("[req_put返回内容是]: {}".format(self.json))
        return self.json
```

11. 编写一个用例，创建一个签名，审核为不通过，然后再修改该签名，最后查看修改的字段是否准确

```
# 审核签名
res = ts.req_post("sign", ts.sign_data())
sign_id = res["sign_id"]
ts.review('sign', sign_id, 'rejected')
# 执行查询
ts.tsms_get('sign')
# 解析查询结果
b = ts.get_field("audit_status", id=sign_id)
# 断言
assert b == 'rejected', "审核结果和预期不一致"

edit = {
    "sign_id": sign_id,  # 必填
    "signature": "修改后的签名",  # 必填
    "source": "修改后的地址",  # 必填
    "pics": []  # 选填
}
# 修改签名
r = ts.req_put('sign', edit)
# 检查修改后的签名返回的id是否和之前一致
assert r["sign_id"] == sign_id
# 检查审核状态是否重写改为了review
ts.tsms_get('sign')
b = ts.get_field("audit_status", id=sign_id)
assert b == 'reviewing', "审核结果和预期不一致"
```

12. 异常场景，每个接口写一个异常场景

```
# 创建签名为空
data_add = {
```

```
    "signature": "",  # 必填
    "source": "深圳",  # 必填
    "pics": ["tu1"]  # 选填
}
ts.req_post("sign", data_add)
ts.check_fail(400, {'error': 'ER:0004', 'message': 'prams fail'})

# 查询签名密码错误
ts.tsms_get("sign", 'dcs', '1234')
ts.check_fail(400, {'error': 'ER:0001', 'message': 'auth not pass'})

# 删除不存在的签名，注意，这里不能直接使用check_fail进行断言
sign_id = 7
dele_sign = {"sign_id": sign_id}
r = ts.req_delete('sign', dele_sign)
assert ts.status_code == 403
assert json.loads(r) == {"error": "ER:0012", "message": "delete sign fail"}

# 使用非root用户进行审核
edit = {
    "sign_id": 258,  # 必填
    "signature": "修改后的签名",  # 必填
    "source": "修改后的地址",  # 必填
    "pics": []  # 选填
}
ts.req_put('sign', edit, user='dcs', passwd='123')
ts.check_fail(403, {'error': 'ER:0011', 'message': 'edit sign fail'})
```

响应

reponse属性

response 有很多属性，重点需要掌握：status_code，headers，cookies，text/json()

```
import requests

response = requests.get('http://www.jianshu.com')
# 获取响应码
print(type(response.status_code), response.status_code)
# 获取响应头
print(type(response.headers), response.headers)
# 获取cookies
print(type(response.cookies), response.cookies)
# 重定向地址
print(type(response.url), response.url)
# 获取请求过程
print(type(response.history), response.history)
```

```
<class 'int'> 403
<class 'requests.structures.CaseInsensitiveDict'> {'Server': 'Tengine',
'Content-Type': 'text/html', 'Transfer-Encoding': 'chunked', 'Connection':
'keep-alive', 'Date': 'Sat, 21 Sep 2019 07:04:27 GMT', 'Vary': 'Accept-
Encoding', 'Strict-Transport-Security': 'max-age=31536000; includeSubDomains;
preload', 'Content-Encoding': 'gzip', 'x-alicdn-da-ups-status': 'endOs,0,403',
'Via': 'cache27.l2nu16-1[5,0], cache7.cn64[42,0]', 'Timing-Allow-Origin': '*',
'EagleId': '7793461b15690494669853834e'}
<class 'requests.cookies.RequestsCookieJar'> <RequestsCookieJar[]>
<class 'str'> https://www.jianshu.com/
<class 'list'> [<Response [301]>]
```

思考题：为什么返回码是403，但是history看到的是301?

状态码判断

通过判断返回码，来确定后续操作，可以使用requets提供的codes.xxx关键字来提取状态码

```
import requests

response = requests.get('http://www.jianshu.com/hello.html')
print(response.status_code)
print(requests.codes.forbidden)
if response.status_code == requests.codes.forbidden:
    print("403 requests forbidden")
```

```
403
403
403 requests forbidden
```

```
import requests
url_sign = "http://127.0.0.1:5001/v1/signature"
auth = ("dcs", "123")
# 使用请求头
# response = requests.post(url_sign, headers=headers, auth=auth,
json=data_add)
# 不使用请求头，会提示错误，开发自己定义的
response = requests.get(url_sign, auth=auth)
print(response.status_code)
print(requests.codes.ok)
if response.status_code == requests.codes.ok:
    print("requests success")
```

```
200
200
requests success
```

requets.codes.xxx：所有的状态码和关键字的对应关系

```
100: ('continue',),
101: ('switching_protocols',),
102: ('processing',),
103: ('checkpoint',),
122: ('uri_too_long', 'request_uri_too_long'),
200: ('ok', 'okay', 'all_ok', 'all_okay', 'all_good', '\\o/', '✓'),
201: ('created',),
202: ('accepted',),
203: ('non_authoritative_info', 'non_authoritative_information'),
204: ('no_content',),
205: ('reset_content', 'reset'),
206: ('partial_content', 'partial'),
207: ('multi_status', 'multiple_status', 'multi_stati', 'multiple_stati'),
208: ('already_reported',),
226: ('im_used',),

# Redirection.
300: ('multiple_choices',),
301: ('moved_permanently', 'moved', '\\o-'),
302: ('found',),
303: ('see_other', 'other'),
304: ('not_modified',),
305: ('use_proxy',),
306: ('switch_proxy',),
307: ('temporary_redirect', 'temporary_moved', 'temporary'),
308: ('permanent_redirect',
      'resume_incomplete', 'resume',), # These 2 to be removed in 3.0

# Client Error.
400: ('bad_request', 'bad'),
401: ('unauthorized',),
402: ('payment_required', 'payment'),
403: ('forbidden',),
404: ('not_found', '-o-'),
405: ('method_not_allowed', 'not_allowed'),
406: ('not_acceptable',),
407: ('proxy_authentication_required', 'proxy_auth', 'proxy_authentication'),
408: ('request_timeout', 'timeout'),
409: ('conflict',),
410: ('gone',),
411: ('length_required',),
412: ('precondition_failed', 'precondition'),
```

```
413: ('request_entity_too_large',),
414: ('request_uri_too_large',),
415: ('unsupported_media_type', 'unsupported_media', 'media_type'),
416: ('requested_range_not_satisfiable', 'requested_range',
'range_not_satisfiable'),
417: ('expectation_failed',),
418: ('im_a_teapot', 'teapot', 'i_am_a_teapot'),
421: ('misdirected_request',),
422: ('unprocessable_entity', 'unprocessable'),
423: ('locked',),
424: ('failed_dependency', 'dependency'),
425: ('unordered_collection', 'unordered'),
426: ('upgrade_required', 'upgrade'),
428: ('precondition_required', 'precondition'),
429: ('too_many_requests', 'too_many'),
431: ('header_fields_too_large', 'fields_too_large'),
444: ('no_response', 'none'),
449: ('retry_with', 'retry'),
450: ('blocked_by_windows_parental_controls', 'parental_controls'),
451: ('unavailable_for_legal_reasons', 'legal_reasons'),
499: ('client_closed_request',),

# Server Error.
500: ('internal_server_error', 'server_error', '/o\\', '✗'),
501: ('not_implemented',),
502: ('bad_gateway',),
503: ('service_unavailable', 'unavailable'),
504: ('gateway_timeout',),
505: ('http_version_not_supported', 'http_version'),
506: ('variant_also_negotiates',),
507: ('insufficient_storage',),
509: ('bandwidth_limit_exceeded', 'bandwidth'),
510: ('not_extended',),
511: ('network_authentication_required', 'network_auth',
'network_authentication'),
```