概述

你需要了解以下知识和技术，以便掌握后续的实例代码：

- http.client常用API
- testunit
- logging
- PO模式

本文不对上述知识做什么任何的讲解，需要了解上述知识请参见公众号前面发的基础篇系列文章。

豆瓣API

我们的测试对象为豆瓣图书相关开放的API：

https://developers.douban.com/wiki/?title=book_v2

对于实例过程中用到的API请参见该链接API说明。

说明

下面实例中所有代码的封装不采用python任何的高级特性，只使用基本特性，以便小白更好的学习掌握，至于你需要的更高级的封装方式，请出门左转自己去搞。

实例

这里先用一个接口测试演示。

```
#-*- coding:utf-8 -*-

__author__ = "苦叶子"

import http.client
import logging
import unittest

## 日志管理类
LOGGING_FORMAT = '%(asctime)s %(filename)s[line:%(lineno)d] %(levelname)s %
(message)s'

class LYMLogging:
    def __init__(self,
        level=logging.DEBUG, # 日志级别
        format=LOGGING_FORMAT, # 日志格式
        datefmt='%a, %d %b %Y %H:%M:%S', # 日期格式
        filename='LYM.log', # 日志文件名
        filemode='w' # 文件打开模式
        ):
        self.level = level
```

```
        self.format = format
        self.datefmt = datefmt
        self.filename = filename
        self.filemode = filemode

        # 初始化日志同时输出到console和日志文件
        logging.basicConfig(level=self.level,
            format=self.format,
            datefmt=self.datefmt,
            filename=self.filename,
            filemode=self.filemode)

        #定义一个StreamHandler，将INFO级别或更高的日志信息打印到标准错误，并将其添加到
当前的日志处理对象
        console = logging.StreamHandler()
        console.setLevel(logging.INFO)
        formatter = logging.Formatter('%(name)-12s: %(levelname)-8s %
(message)s')
        console.setFormatter(formatter)
        logging.getLogger('LYMHTTPLogger').addHandler(console)
        self.log = logging.getLogger("LYMHTTPLogger")

    # 日志输出
    def output(self, msg, level=logging.DEBUG):
        if level == logging.DEBUG:
            # 调试信息
            self.log.debug(msg)
        elif level == logging.INFO:
            # 一般的信息
            self.log.info(msg)
        elif level == logging.WARNING:
            # 警告信息
            self.log.warning(msg)
        elif level == logging.ERROR:
            # 错误信息
            self.log.error(msg)
        else:
            # 尼玛
            self.log.critical(msg)

    def set_level(self, level=logging.DEBUG):
        self.log.set_level(level)

# http.client封装
# http管理类
class LYMHttp:
    def __init__(self, protocol, host, port=80,
        key_file=None,  # ssl
        cert_file=None, # ssl
        timeout=30,
        log_level=logging.INFO
        ):
```

```
        self.log_level = log_level
        self.log = LYMLogging(level=log_level)
        self.log.output("初始化http连接到: %s:%d" % (host, port))

        self.host = host
        self.port = port
        self.timeout = timeout
        self.key_file = key_file
        self.cert_file = cert_file
        self.response = None
        self.data = None
        self.status = None
        self.reason = None
        self.headers = None

        self.http = None
        if protocol == "http":
            self.http = http.client.HTTPConnection(host=self.host,
                    port=self.port, timeout=self.timeout)
        elif protocol == "https":
            self.http = http.client.HTTPSConnection(host=self.host,
                    port=self.port,
                    key_file=self.key_file,
                    cert_file=self.cert_file,
                    timeout=self.timeout)
        else:
            print("不支持的协议类型: ", protocol)
            exit()


    # 返回response响应对象
    def request(self,
        method, # 请求方法
        url,  # 请求url
        body=None,  # 请求数据
        headers={} # 请求头
        ):
        self.http.request(method=method, url=url, body=body, headers=headers)

        self.response = self.http.getresponse()

        self.data = self.response.read()
        self.status = self.response.status
        self.reason = self.response.reason
        self.headers = self.response.getheaders()
        self.log.output("------" * 10, self.log_level)
        self.log.output("\nrequest")
        self.log.output("\nurl: %s \nmethod: %s \nheaders: %s \ndata: %s" %
                (url, method, headers, body), self.log_level)
        self.log.output("\nresponse")
        self.log.output("\nstatus: %s \nreason: %s \nheaders: %s \ndata: %s" %
                (self.status, self.reason, self.headers, self.data),
```

```
self.log_level)

        return self.response

    # 关闭连接
    def close(self):
        if self.http:
            self.http.close()

    # 返回响应内容
    def get_data(self):
        return self.data

    # 返回指定响应头
    def get_header(self, name):
        for header in self.headers:
            if header[0] == name:
                return header[1]

        return None

    # 返回完整的响应头
    def headers(self):
        return self.headers

    # 返回状态码及文本说明
    def get_status_reason(self):
        return (self.status, self.reason)


# Page基类
class Page:
    """
        基类，所有的page models都需要继承该类
    """
    def __init__(self, protocol, host, port=80,
        key_file=None,  # ssl
        cert_file=None, # ssl
        timeout=30,
        log_level=logging.INFO):

        self.http = LYMHttp(protocol=protocol,
                        host=host,
                        port=port,
                        key_file=key_file,
                        cert_file=cert_file,
                        timeout=timeout,
                        log_level=log_level)

    def request(self, method, url, body=None, headers={}):
        self.http.request(method=method, url=url, body=None, headers={})
```

```
    def close(self):
        if self.http:
            self.http.close()

# 测试豆瓣API
class BookSearchPage(Page):
    def __init__(self, protocol, host, port=80,
            key_file=None,  # ssl
            cert_file=None, # ssl
            timeout=30,
            log_level=logging.INFO):

        Page.__init__(self, protocol=protocol,
                      host=host,
                      port=port,
                      key_file=key_file,
                      cert_file=cert_file,
                      timeout=timeout,
                      log_level=log_level)

    # 查询python相关的书籍
    def search_python_book(self, method, url, body=None, headers={}):

        self.request(method=method, url=url, body=body, headers=headers)

        return self.http.get_data()


# 测试用例
class TestSearchBookPage(unittest.TestCase):
    def setUp(self):
        self.book_search_page = BookSearchPage(protocol="https",
host="api.douban.com", port=443)

    def test_search_python_book(self):
        # 查找python相关的书籍即q=python，只找两本即count=2
        books = self.book_search_page.search_python_book(method="GET",
url="/v2/book/search?q=python&count=2")

        # 获取并断言下http status及reason
        status, reason = self.book_search_page.http.get_status_reason()
        self.assertEqual(status, 200)
        self.assertEqual(reason, "OK")

        # 获取并断言下http header 例如断言下返回的Content-Type是不是
application/json; charset=utf-8
        content_type = self.book_search_page.http.get_header("Content-Type")
        self.assertEqual(content_type, "application/json; charset=utf-8")

        # 看一下返回的数据类型
        print("/v2/book/search?q=python&count=2返回的数据类型为：  ", type(books))
        # 断言下返回类型
```

```
        self.assertIsInstance(books, bytes)

        # 强制将bytes类转成成dcit类型
        # 这里运行时 可能会出现一些警告信息，不用理会
        books_dict = eval(str(books, encoding="utf-8"))

        # 断言下count计数，应该为2，因为我们只查找2本
        self.assertEqual(books_dict["count"], 2)


    def tearDown(self):
        self.book_search_page.close()

if __name__ == "__main__":
    print("http.client Restful API测试实例")

    unittest.main()
```

保存上述代码到http.client_pom_demo.py中, 使用下述命令运行即可：

```
python http.client_pom_demo.py
```

对于结果请自行查看

小结

本文主要演示如何基于http.client + logging + unittest + pom进行基本的接口测试，大家吸收下基本的思路就好，毕竟基于http.client这类的过于低层次的库来做还是太麻烦。

扫一扫关注微信公众号：