RIGA TECHNICAL UNIVERSITY

Faculty of Electronics and Telecommunications

INSTITUTE OF TELECOMMUNICATIONS

RAE411 Telecommunications Software

Requests:
Automatically crawl HTML pages
Automatic web request submission

Requests
http for humans

Star 48,600

Requests is an elegant and simple HTTP library for Python, built for human beings.

Useful Links

Quickstart
Advanced Usage
API Reference
Release History
Contributors Guide

Recommended Packages and Extensions

Requests @ GitHub
Requests @ PyPI
Issue Tracker

Requests: HTTP for Humans™

Release v2.28.1. (Installation)

downloads/month 226M | license Apache 2.0 | wheel yes | python 3.7 | 3.8 | 3.9 | 3.10 | 3.11

Requests is an elegant and simple HTTP library for Python, built for human beings.

Behold, the power of Requests:

```
>>> r = requests.get('https://api.github.com/user', auth=('user', 'pass'))
>>> r.status_code
200
>>> r.headers['content-type']
'application/json; charset=utf8'
>>> r.encoding
'utf-8'
>>> r.text
'{"type":"User"...'
>>> r.json()
{'private_gists': 419, 'total_private_repos': 77, ...}
```

See similar code, sans Requests.

Requests allows you to send HTTP/1.1 requests extremely easily. There's no need to manually add query strings to your URLs, or to form-encode your POST data. Keep-alive and HTTP connection pooling are 100% automatic, thanks to urllib3.

Beloved Features

https://requests.readthedocs.io/en/latest/

The 7 main methods of the Requests library:

requests.request() constructs a request that supports the basic methods of the following methods
requests.get() The main method for obtaining HTML pages corresponding to **HTTP GET**
requests.head() The method to obtain the header information of HTML pages, corresponding to the **HEAD of HTTP**
requests.post() Method for submitting POST requests to HTML pages, corresponding to **HTTP POST**
requests.put() The method of submitting a PUT request to an HTML page, corresponding to **HTTP PUT**
requests.patch() Submit a partial modification request to an HTML page, corresponding to **HTTP PATCH**
requests.delete() Submit a delete request to the HTML page, corresponding to **HTTP DELETE**

.head(): return metadata (header) to the site

requests.get(`url, params=None, **kwargs`***)***

```
r = requests.get(url)
```

Returns a Response object containing server resources

Response

Request

Constructs a Request object that requests resources from the server

· url : the url link of the page to be obtained
· params : extra parameters in url, dictionary or byte stream format, optional
· **kwargs: 12 parameters that control access

Properties of the Response object

Property	Description
r.status_code	The return status of the HTTP request, 200 means the connection is successful, 404 means the failure
r.text	The string form of the HTTP response content, that is, the page content corresponding to the URL
r.Encoding	Guess the encoding of the response content from the HTTP header
r.apparent_encoding	The response content-encoding method analyzed from the content (alternative encoding method)
r.content	The binary form of the HTTP response content

r.encoding: If charset does not exist in the header, the encoding is considered ISO-8859-1
r.text displays web content according to r.encoding
r.apparent_encoding: The encoding method analyzed according to the content of the web page, which can be regarded as an alternative to r.encoding

for understanding: https://www.dataquest.io/blog/tutorial-an-introduction-to-python-requests-library/

Exception from Requests library:

Exception	Description
requests.ConnectionError	Network connection error exception, such as DNS query failure, connection refused, etc.
requests.HTTPError	HTTP error exception
requests.URLRequired	URL missing exception
requests.TooManyRedirects	Exceeds the maximum number of redirects, resulting in a redirect exception
requests.ConnectTimeout	Connection to the remote server timed out exception
requests.Timeout	The requested URL timed out, resulting in a timeout exception
r.raise_for_status()	If it is not 200, an exception request is. HTTPError will be generated

response.raise_for_status() returns an HTTPError object if an error has occurred during the process.

Difference between PATCH and PUT

Suppose there is a set of data UserInfo at the URL location, including 20 fields such as UserID, UserName, etc.

Requirement: The user modifies the UserName, and the others remain unchanged

- Use PATCH to submit only partial update requests for UserName to the URL
- With PUT, all 20 fields must be submitted to the URL, unsubmitted fields are removed

The main benefit of PATCH: saving network bandwidth。

```
requests.request(method, url, **kwargs)
```

· method : request method, corresponding to 7 types such as get/put/post

· url : the URL link of the page to be obtained

· **kwargs: parameters to control access, a total of 13

```
r = requests.request('GET', url, **kwargs)
r = requests.request('HEAD', url, **kwargs)
r = requests.request('POST', url, **kwargs)
r = requests.request('PUT', url, **kwargs)
r = requests.request('PATCH', url, **kwargs)
r = requests.request('delete', url, **kwargs)
r = requests.request('OPTIONS', url, **kwargs)
```

· **kwargs: parameters to control access. All are optional
params: dictionary or byte sequence, added to the url as a parameter
data: dictionary, byte sequence or file object, as the content of the Request
JSON : data in JSON format, as the content of Request
headers: dictionary, HTTP custom headers
cookies: dictionary or CookieJar, cookies in Request
auth: tuple, support HTTP authentication function
files: dictionary type, transfer files
timeout: Set the timeout time in seconds
proxies: dictionary type, set the access proxy server, you can add login authentication
allow_redirects: True/False, the default is True, redirect switch
stream: True/False, the default is True, get the content immediately download switch
verify: True/False, the default is True, verify the SSL certificate switch
cert: local SSL certificate path

Limitations of Web Crawlers

• Source review: Judging User-Agent for restriction

Check the User-Agent field of the incoming HTTP protocol header, and only respond to the visits of browsers or friendly crawlers

• Publication Announcement: Robots Protocol

Inform all crawlers of the crawling strategy of the website and require crawlers to abide by

Robots Exclusion Standard （web crawler exclusion standard）

Function: The website tells the web crawler which pages can be crawled and which pages cannot be crawled

Form: ***robots.txt*** file in the root directory of the website

https://www.google.com/robots.txt

* means all, / means root directory

In actual operation, how to comply with the Robots protocol?
Web Crawler:
Automatically or manually identify robots.txt, and then crawl content
Binding:
The Robots agreement is suggested but non-binding. Web crawlers can not abide by it, but there are legal risks.

https://www.crummy.com/software/BeautifulSoup/

Beautiful Soup library is a functional library for parsing, traversing, and maintaining the "tag tree"


```
<html>
    <body>
        <p class="title"> … </p>
    </body>
</html>
```

```
<>.find_all(name, attrs, recursive, string, **kwargs)
```

Returns a list type to store the result of the search

· name : search string for tag name

· attrs: the search string for tag attribute values, which can be marked with attribute retrieval

· recursive: Whether to retrieve all descendants, the default is True

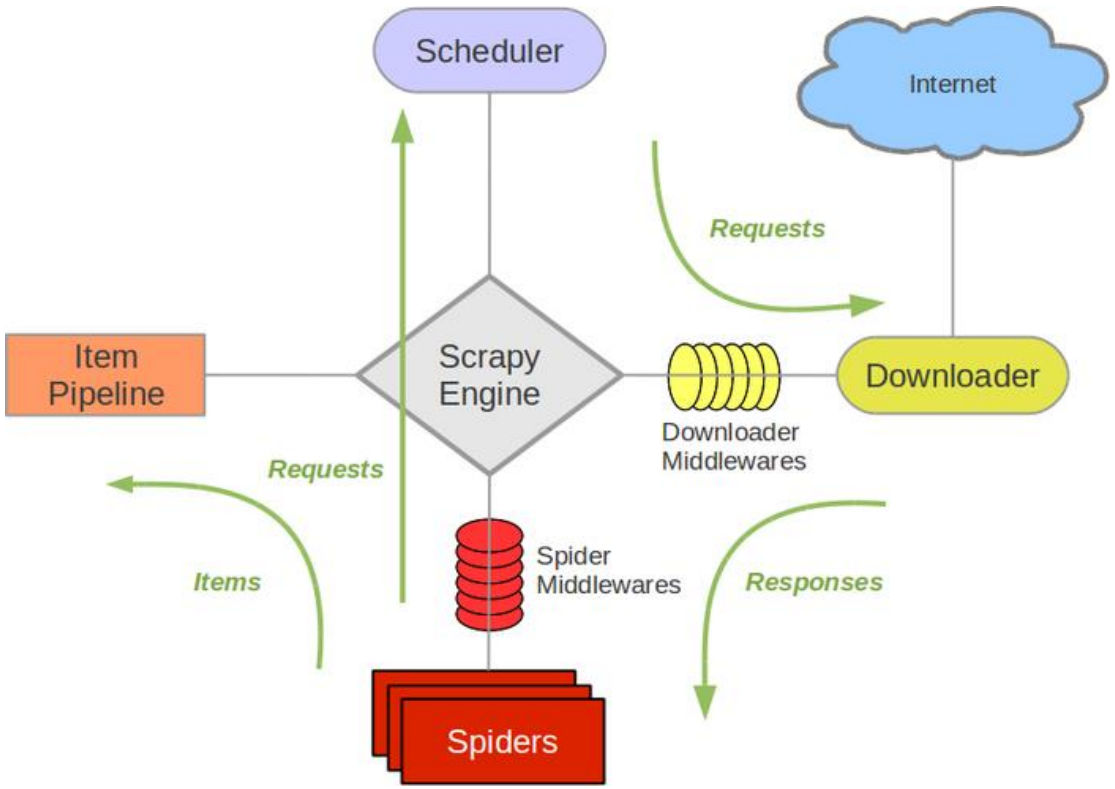
· string: search string in the string area in <>…</>

Scrapy is a fast and powerful web crawler framework

Scrapy is not a function library but a crawler framework.

The crawler framework is a collection of software structures and functional components that implement crawler functions.

The crawler framework is a semi-finished product that can help users implement professional web crawlers.



Scrapy Engine: Responsible for communication, signal, data transmission, etc., among Spider, Item Pipeline, Downloader, and Scheduler.

Scheduler: It is responsible for accepting the Request sent by the engine, sorting it in a certain way, entering the queue, and returning it to the engine when the engine needs it.

Downloader: Responsible for downloading all the Requests sent by the Scrapy Engine (engine), and returning the obtained Responses to the Scrapy Engine (engine), and the engine will hand it over to the Spider for processing.

Spider (crawler): It is responsible for processing all Responses, analyzing and extracting data from them, obtaining the data required by the Item field, submitting the URL that needs to be followed up to the engine, and entering the Scheduler again.

Item Pipeline: It is responsible for processing the items obtained in the Spider and performing post-processing (detailed analysis, filtering, storage, etc.).

Downloader Middlewares: You can think of it as a component that can customize and extend the download function.

Spider Middlewares: You can understand it as a functional component that can customize the expansion and operation engine and the intermediate communication between the engine and the Spider (such as Responses entering the Spider; and Requests going out from the Spider)

Scrapy's operation process
The code is written, and the program starts running...

1 engine: Hi! Spider, which site are you dealing with?
2 Spider: The boss wants me to deal with xxxx.com.
3 Engine: Give me the first URL that needs to be processed.
4 Spider: Here you are, the first URL is xxxxxxx.com.
5 Engine: Hi! Scheduler, I have a request to ask you to sort it into the queue for me.
6 Scheduler: OK, it is processing you wait a minute.
7 Engine: Hi! Scheduler, give me your processed request.
8 Scheduler: here you are, this is the request I have processed
9 Engine: Hi! Downloader, please help me download this request according to the boss's download middleware settings
10 Downloader: OK! Here you go, here's the downloaded stuff. (If it fails: sorry, the download of this request failed. Then the engine tells the scheduler that the download of this request failed, please record it, we will download it later)
11 Engine: Hi! Spider, this is something that has been downloaded, and it has been processed according to the download middleware of the boss, you can handle it yourself (note! The responses here are handled by the def parse() function by default)
12 Spider: (for the URL that needs to be followed up after processing the data), Hi! Engine, I have two results here, this is the URL I need to follow up, and this is the Item data I got.
13 Engine: Hi! Pipeline I have an item here, please help me deal with it! scheduler! This is the URL that needs to be followed up and you can help me deal with it. Then start the cycle from the fourth step until all the information needed by the boss is obtained.
14 Pipeline Scheduler: OK, Doing It Now!

requests VS. Scrapy

Both can perform page requests and crawl, two important technical routes of Python crawlers
Both are usable, well-documented, and easy to get started with
Both do not have functions such as processing js, submitting forms, and responding to verification codes (extensible)

Difference:
requests:
page level crawler
function library
Insufficient consideration of concurrency and poor performance
The focus is on page downloads
Flexible customization
Easy to get started

Scrapy:
Website level crawler
frame
Good concurrency and high performance
The focus is on the crawler structure
General customization is flexible, but in-depth customization is difficult
Getting started is a bit difficult

```
(base) tianhua@tianhua-VirtualBox:~/Desktop/Practical_Exercise6$ scrapy
Scrapy 2.6.2 - no active project

Usage:
  scrapy <command> [options] [args]

Available commands:
  bench         Run quick benchmark test
  commands
  fetch         Fetch a URL using the Scrapy downloader
  genspider     Generate new spider using pre-defined templates
  runspider     Run a self-contained spider (without creating a project)
  settings      Get settings values
  shell         Interactive scraping console
  startproject  Create new project
  version       Print Scrapy version
  view          Open URL in browser, as seen by Scrapy

  [ more ]      More commands available when run from project directory

Use "scrapy <command> -h" to see more info about a command
```

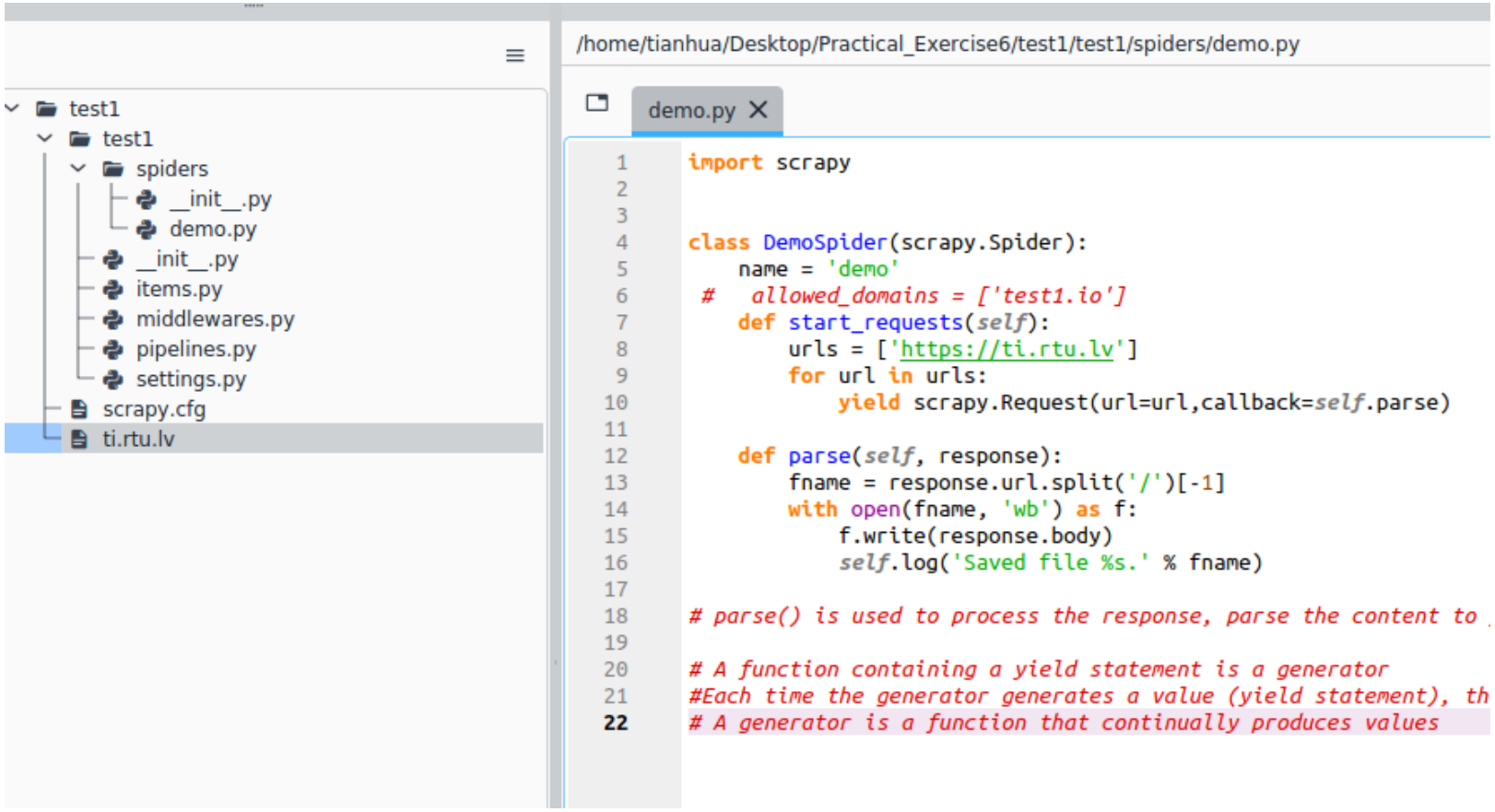
Command	Description	Format
startproject	Create a new project	scrapy startproject <name> [dir]
genspider	Creates a crawler	scrapy genspider [options] <name> <domain>
settings	Get crawler configuration information	scrapy settings [options]
crawl	Run a crawler	scrapy crawl <spider>
list	Lists all crawlers in the project	scrapy list
shell	Start URL debug command line	scrapy shell [url]

```
mySpider/
    scrapy.cfg
    mySpider/
        __init__.py
        items.py
        pipelines.py
        settings.py
        spiders/
            __init__.py
            ...
```

scrapy.cfg: Configuration file for the project.
mySpider/: The Python module of the project, the code will be referenced from here.
mySpider/items.py: The object file for the project.
mySpider/pipelines.py: The pipeline file for the project.
mySpider/settings.py: The settings file for the project.
mySpider/spiders/: The directory where the spider code is stored.

```
(base) tianhua@tianhua-VirtualBox:~/Desktop/Practical_Exercise6$ scrapy startproject test1
New Scrapy project 'test1', using template directory '/home/tianhua/anaconda3/lib/python3.9/site-packages/scrapy/templates/project', created in:
    /home/tianhua/Desktop/Practical_Exercise6/test1

You can start your first spider with:
    cd test1
    scrapy genspider example example.com
(base) tianhua@tianhua-VirtualBox:~/Desktop/Practical_Exercise6$ ls
test1
(base) tianhua@tianhua-VirtualBox:~/Desktop/Practical_Exercise6$ cd test1/
(base) tianhua@tianhua-VirtualBox:~/Desktop/Practical_Exercise6/test1$ ls
scrapy.cfg  test1
(base) tianhua@tianhua-VirtualBox:~/Desktop/Practical_Exercise6/test1$ scrapy genspider demo test1.io
Created spider 'demo' using template 'basic' in module:
  test1.spiders.demo
(base) tianhua@tianhua-VirtualBox:~/Desktop/Practical_Exercise6/test1$
```



Steps to use Scrapy:
Step 1: Create a project and Spider template
Step 2: Writing a Spider
Step 3: Write the Item Pipeline
Step 4: Optimize configuration strategy
Scrapy data types:
Request class
Response class
Item class

```
16.0 tldextract-3.4.0 typing-extensions-4.5.0 urllib3-1.26.15 w3lib-2.1.1 zope.interface-6.0
tianhua@tianhua-VirtualBox:~/Desktop/Lab4/test1$ scrapy crawl demo
2023-03-18 18:35:01 [scrapy.utils.log] INFO: Scrapy 2.8.0 started (bot: test1)
2023-03-18 18:35:01 [scrapy.utils.log] INFO: Versions: lxml 4.9.2.0, libxml2 2.9.14, cssselect 1.2.0, parsel 1.7.0, w3lib 2.1.1, T
d 22.10.0, Python 3.9.7 (default, Sep 16 2021, 13:09:58) - [GCC 7.5.0], pyOpenSSL 23.0.0 (OpenSSL 3.0.8 7 Feb 2023), cryptography
2, Platform Linux-5.15.0-60-generic-x86_64-with-glibc2.35
2023-03-18 18:35:01 [scrapy.crawler] INFO: Overridden settings:
{'BOT_NAME': 'test1',
 'FEED_EXPORT_ENCODING': 'utf-8',
 'NEWSPIDER_MODULE': 'test1.spiders',
 'REQUEST_FINGERPRINTER_IMPLEMENTATION': '2.7',
 'ROBOTSTXT_OBEY': True,
 'SPIDER_MODULES': ['test1.spiders'],
 'TWISTED_REACTOR': 'twisted.internet.asyncioreactor.AsyncioSelectorReactor'}
2023-03-18 18:35:01 [asyncio] DEBUG: Using selector: EpollSelector
2023-03-18 18:35:01 [scrapy.utils.log] DEBUG: Using reactor: twisted.internet.asyncioreactor.AsyncioSelectorReactor
2023-03-18 18:35:01 [scrapy.utils.log] DEBUG: Using asyncio event loop: asyncio.unix_events._UnixSelectorEventLoop
2023-03-18 18:35:01 [scrapy.extensions.telnet] INFO: Telnet Password: 8d1d8bb4805bf4ce
2023-03-18 18:35:01 [scrapy.middleware] INFO: Enabled extensions:
                                                                                In [1]:
```

pip install --upgrade --force-reinstall attrs scrapy

```
class scrapy.http.Request()
```

The Request object represents an HTTP request, generated by Spider and executed by Downloader.

property and method description

.url Request corresponding request URL address
.method corresponds to the request method, 'GET' 'POST', etc.
.headers dictionary style request headers
.body Request content body, string type
.meta Extended information added by users, used to transfer information between Scrapy internal modules
.copy() copies the request

```
class scrapy.http.Response()
```

The Response object represents an HTTP response, generated by Downloader and processed by Spider.

property and method description

URL address corresponding to .url Response
.status HTTP status code, the default is 200
Header information corresponding to .headers Response
.body Response corresponding content information, string type
.flags set of flags
.request generates a Request object corresponding to the Response type
.copy() copies the response

```
class scrapy.item.Item()
```

The Item object represents an information content extracted from an HTML page, which is generated by the Spider and processed by the Item Pipeline. The Item is similar to a dictionary type and can be operated according to the dictionary type.


Thank You!
Questions?
Comments?
Suggestions?