httpsuite

Felipe Faria

Aug 31, 2021

CONTENTS

CHAPTER ONE

QUICKSTART

1.1 Installing

To get started with `httpsuite`, install the latest stable release from PyPi:

```
pip install httpsuite
```

1.2 Getting Started

There are two principal abstractions to be aware of it in `httpsuite`, `Request` and `Response`.

```
from httpsuite import Request, Response
```

These classes represent an HTTP/1.x request and response message, and offer a high-level API to *modify*, *compile*, and *parse* them. The `Request` and `Response` can be initialized via two different methods.

1.2.1 `__init__()`

Create a `Request` or `Response` object with the given parameter.

```
from httpsuite import Request, Response

request = Request(
    method="POST",
    target="/",
    protocol="HTTP/1.1",
    headers={"User-Agent": "httpsuite", "Content-Length": 12},
    body="Hello world."
)

response = Response(
    protocol="HTTP/1.1",
    status=200,
    status_msg="OK",
    headers={"User-Agent": "httpsuite", "Content-Length": 8},
    body="Hi back!"
)
```

1.2.2 .parse()

Or, parse from *bytes* to create a new Request or Response with the given details.

bytes

Most useful when using socket connections.

```
from httpsuite import Request, Response

req = Request.parse(b"GET / HTTP/1.1\r\nUser-Agent: httpsuite\r\nContent-Length: 12\r\n\
↪r\nHello world")
resp = Response.parse(b"HTTP/1.1 200 OK\r\nUser-Agent: httpsuite\r\nContent-Length: 12\r\
↪n\r\nHi back!")
```

1.3 Modify

The next probable step after initializing a Request or Response object is to *modify* and *compile*. Object modification is done as one would expect.

```
request.method = "POST"
response.status = 300
response.status_msg = b"Continue"
```

Notice that setting object properties is type-agnostic. Properties can be modified to either int, str, or bytes objects. Internally, httpsuite automatically converts every property of a request or response into an Item, which is a low-level interface to allow easy setting and comparissons on the fly. Similarly to setting properties, one can be assured of type-agnostic property comparissions.

```
request.status == 300     # True
request.status == "300"   # True
request.status == b"300"  # True
```

1.4 Compile

After modifying a message compilation allows the Request and Response objects to be compiled into less maluable yet useful types. Those types being bytes or str.

```
from httpsuite import Request, Response
import json

body = json.dumps({"hello": "world"})
request = Request(
    method="POST",
    target="/post",
    protocol="HTTP/1.1",
    headers={
        "Host": "httpbin.org",
        "Connection": "close",
```

(continues on next page)

(continued from previous page)

```
        "Content-Length": len(body),
        "Accept": "*/*",
    },
    body=body,
)
```

1.4.1 .raw

Useful to use with sockets.

```
print(request.raw)
```

```
b'POST /post HTTP/1.1\r\nHost: httpbin.org\r\nConnection: close\r\nContent-Length: 18\r\
↪nAccept: */*\r\n\r\n{"hello": "world"}'
```

1.4.2 __str__

Pretty print of the object.

```
print(request)
```

```
→ POST /post HTTP/1.1
→ Host: httpbin.org
→ Connection: close
→ Content-Length: 18
→ Accept: */*
→ {"hello": "world"}
```

1.5 More

If you finished this guide and want to continue learning more you can do so by reading the package's documentation found on the left menu.

CHAPTER

TWO

HTTP

```
from httpsuite.http import Message, Request, Response
```

Classes for parsing, modifying, and re-compiling HTTP messages.

---

class Message(*abc.ABC*)

Abstract class that contains shared methods and properties accessible by both the Request and Response classes.

> **Warning:** This class is not intended to be used by itself. Note that Message is an abstraction that represents the shared properties and methods of both a Request and Response instance. All functions displayed in Message are, therefore, accessible by both the Request and Response classes.

__init__(*protocol*, *headers=None*, *body=None*)

Initializes an HTTP Message.

Parameters

- **protocol** – "<major>.<minor>" numbering scheme to indicate versions of the protocol.
- **headers** – Collection of case-insensitive name followed by a colon (:).
- **body** – Data associated with the message.

Parse

httpsuite.http.Message.**parse**(*message*)

Parses the passed `messaged` into a `cls` instance (either a *Request* or *Response* object).

Parameters **message** – The primative or object to convert into a Request or Response object.

Returns An initialized object of class `cls`.

Raises **RequiredAwait** – Attempting to parse asynchronous object without 'await' statement.

Properties

Note: The `httpsuite.http.Message.raw()` function will return the message with proper `\r\n` escape characters. It can be used directly with `sockets` or any low-level communication system that requires properly formatted HTTP messages.

---

property string

String representation of the Message.

Returns Message as a string, without any arrows.

property raw
Bytes representation of the Message.

Returns Message as a bytes, without arrows, properly escaped.

HTTP

Note: All HTTP properties are saved as an `Item` type. Modification and comparissons can be done on the fly.

property protocol
Protocol of the message.

property headers
Headers of the message.

property body
Body of the message.

---

class Request(*Message*)

__init__(*method*, *target*, *protocol*, *headers=None*, *body=None*)
Python object representation of an HTTP/1.x request."

Parameters

- **method** – Indicates the desired action on the server's resource.
- **target** – Resource location in the server for which the client is requesting.
- **protocol** – "<major>.<minor>" numbering scheme to indicate versions of the protocol.
- **headers** – Collection of case-insensitive name followed by a colon (:).
- **body** – Data associated with the message.

__str__()
String representation of the Request.

Returns Representation of the Request object with pretty-print (→) arrows.

HTTP

Note: Note that Request is a child object of a Message and therefore has access to *httpsuite.http.Message.headers*, *httpsuite.http.Message.headers*, and *httpsuite.http.Message.body*.

property method
Method of the HTTP request.

property target
Target of the HTTP request.

---

class Response(*protocol*, *status*, *status_msg*, *headers=None*, *body=None*)

__init__(*protocol*, *status*, *status_msg*, *headers=None*, *body=None*)
Python object representation of an HTTP/1.x response.

Parameters

- `protocol` – "<major>.<minor>" numbering scheme to indicate versions of the protocol.
- `status` – Numberical value designating a specific return value.
- `status_msg` – Message related to the status code.
- `headers` – Collection of case-insensitive name followed by a colon (:).
- `body` – Data associated with the message.

`__str__()`

String representation of the Response.

Returns Representation of the Response object with pretty-print (←) arrows.

HTTP

Note: Note that Response is a child object of a Message and therefore has access to *`httpsuite.http.Message.headers`*, *`httpsuite.http.Message.headers`*, and *`httpsuite.http.Message.body`*.

`property status`

Status of the HTTP response.

`property status_msg`

Status message of the HTTP response.

CHAPTER

THREE

INTERFACE

```
from httpsuite.interface import Item, Headers, TwoWayFrozenDict, FrozenSet
```

Lower-level interfaces that `httpsuite` depedents on.

---

class **TwoWayFrozenDict**(*dictionary={}*, ***kwargs*)

A frozen dictionary with two-way capabilities. Locks a dictionary in place after initilization, and provides accessability via key and value.

Note: All the keys and values inside TwoWayFrozenDict are Item objects, which allows easy comparissions to check if an item is inside the TwoWayFrozenDict mapping.

---

class **FrozenSet**

A frozen set with pretty-print.

CHAPTER

FOUR

RFC

```
from httpsuite.RFC import *
```

Collection of RFC specifications related to HTTP requests and responses.

Every item in this file is commented with it's specific specification, chapter, and section number. To access these specifications, utilize the URL https://tools.ietf.org/html/<id>.

Request & Response

Core Rules

```
# rfc5234#appendix-B.1
CR = b"\r"
LF = b"\n"
```

Protocols

```
PROTOCOLS = FrozenSet(
    {
        "HTTP/0.9",  # rfc1945#section-3.1
        "HTTP/1.0",  # rfc1945#section-3.1
        "HTTP/1.1",  # rfc7231
        "HTTP/2.0",  # rfc7540
        "HTTP/3.0",  # draft-ietf-quic-http-27
    }
)
```

Requests

Methods

```
# Request Method Definitions
# rfc7231#section-4
REQUEST_METHODS = FrozenSet(
    {
        "GET",  # rfc7231#section-4.3.1
        "HEAD",  # rfc7231#section-4.3.2
        "POST",  # rfc7231#section-4.3.3
        "PUT",  # rfc7231#section-4.3.4
        "DELETE",  # rfc7231#section-4.3.5
        "CONNECT",  # rfc7231#section-4.3.6
```

(continues on next page)

(continued from previous page)

```
        "OPTIONS",  # rfc7231#section-4.3.7
        "TRACE",  # rfc7231#section-4.3.8
    }
)
```

Headers

```
# Request Header Fields
# rfc7231#section-5
REQUEST_HEADERS = FrozenSet(
    {
        # Control
        # rfc7231#section-4
        "Cache-Control",  # rfc7234#section-5.2
        "Except",  # rfc7231#section-5.1.1
        "Host",  # rfc7230#section-5.4
        "Max-Forwards",  # rfc7231#section-5.1.2
        "Pragma",  # rfc7234#section-5.4
        "Range",  # rfc7233#section-3.1"
        "TE",  # rfc7230#section-4.3
        # Conditionals
        # rfc7231#section-5.2
        "If-Match",  # rfc7232#section-3.1
        "If-None-Match",  # rfc7232#section-3.2
        "If-Modified-Since",  # rfc7232#section-3.3
        "If-Unmodified-Since",  # rfc7232#section-3.4
        "If-Range",  # rfc7233#section-3.2
        # Content Negotiation
        # rfc7231#section-5.3
        "Accept",  # rfc7231#section-5.3.2
        "Accept-Charset",  # rfc7231#section-5.3.3
        "Accept-Encoding",  # rfc7231#section-5.3.4
        "Accept-Language",  # rfc7231#section-5.3.5
        # Authentication Credentials
        # rfc7231#section-5.4
        "Authorization",
        "Proxy-Authorization",
        # Request Context
        # rfc7231#section-5.5
        "From",  # rfc7231#section-5.5.1
        "Referer",  # rfc7231#section-5.5.2
        "User-Agent",  # rfc7231#section-5.5.3
    }
)
```

Response

Status Code

```
# Response Status Code
# rfc7231#section-6
RESPONSE_STATUS = TwoWayFrozenDict(
    {
```

(continues on next page)

(continued from previous page)

```
        # Informational 1xx
        # rfc7231#section-6.2
        100: "Continue",  # rfc7231#section-6.2.1
        101: "Switching Protocols",  # rfc7231#section-6.2.2
        # Successful 2xx
        # rfc7231#section-6.3
        200: "OK",  # rfc7231#section-6.3.1
        201: "Created",  # rfc7231#section-6.3.2
        202: "Accepted",  # rfc7231#section-6.3.3
        203: "Non-Authoritative Information",  # rfc7231#section-6.3.4
        204: "No Content",  # rfc7231#section-6.3.5
        205: "Reset Content",  # rfc7231#section-6.3.6
        206: "Partial Content",  # rfc7233#section-4.1
        # Redirection 3xx
        # rfc7231#section-6.4
        300: "Multiple Choices",  # rfc7231#section-6.4.1
        301: "Moved Permanently",  # rfc7231#section-6.4.2
        302: "Found",  # rfc7231#section-6.4.3
        303: "See Other",  # rfc7231#section-6.4.4
        304: "Not Modified",  # rfc7232#section-4.1
        305: "Use Proxy",  # rfc7231#section-6.4.5
        307: "Temporary Redirect",  # rfc7231#section-6.4.7
        # Client Error 4xx
        # rfc7231#section-6.5
        400: "Bad Request",  # rfc7231#section-6.5.1
        401: "Unauthorized",  # rfc7235#section-3.1
        402: "Payment Required",  # rfc7231#section-6.5.2
        403: "Forbidden",  # rfc7231#section-6.5.3
        404: "Not Found",  # rfc7231#section-6.5.4
        405: "Method Not Allowed",  # rfc7231#section-6.5.5
        406: "Not Acceptable",  # rfc7231#section-6.5.6
        407: "Proxy Authentication Required",  # rfc7235#section-3.2
        408: "Request Timeout",  # rfc7231#section-6.5.7
        409: "Conflict",  # rfc7231#section-6.5.8
        410: "Gone",  # rfc7231#section-6.5.9
        411: "Length Required",  # rfc7231#section-6.5.10
        412: "Precondition Failed",  # rfc7232#section-4.2
        413: "Payload Too Large",  # rfc7231#section-6.5.11
        414: "URI Too Long",  # rfc7231#section-6.5.12
        415: "Unsupported Media Type",  # rfc7231#section-6.5.13
        416: "Range Not Satisfiable",  # rfc7233#section-4.4
        417: "Expectation Failed",  # rfc7231#section-6.5.14
        426: "Upgrade Required",  # rfc7231#section-6.5.15
        # Server Error 5xx
        # rfc7231#section-6.6
        500: "Internal Server Error",  # rfc7231#section-6.6.1
        501: "Not Implemented",  # rfc7231#section-6.6.2
        502: "Bad Gateway",  # rfc7231#section-6.6.3
        503: "Service Unavailable",  # rfc7231#section-6.6.4
        504: "Gateway Timeout",  # rfc7231#section-6.6.5
        505: "HTTP Version Not Supported",  # rfc7231#section-6.6.6
    }
```

(continues on next page)

(continued from previous page)

```
)
```

Headers

```
# Response Header Fields
# rfc7231#section-7
RESPONSE_HEADER = FrozenSet(
    {
        # Control Data
        # rfc7231#section-7.1
        "Age",  # rfc7234#section-5.1
        "Cache-Control",  # rfc7234#section-5.2
        "Expires",  # rfc7234#section-5.3
        "Date",  # rfc7231#section-7.1.1.2
        "Location",  # rfc7231#section-7.1.2
        "Retry-After",  # rfc7231#section-7.1.3
        "Vary",  # rfc7231#section-7.1.4
        "Warning",  # rfc7234#section-5.5
        # Validator Header Fields
        # rfc7231#section-7.2
        "ETag",  # rfc7232#section-2.3
        "Last-Modified",  # rfc7232#section-2.2
        # Authentication Challenges
        # rfc7231#section-7.3
        "WWW-Authenticate",  # rfc7235#section-4.1
        "Proxy-Authenticate",  # rfc7235#section-4.3
        # Response Context
        # rfc7231#section-7.4
        "Accept-Ranges",  # rfc7233#section-2.3
        "Allow",  # rfc7231#section-7.4.1
        "Server",  # rfc7231#section-7.4.2
    }
)
```

CHAPTER

FIVE

BASIC USE

5.1 Request

```python
import json

from httpsuite import Request, ENCODE

# 1. Creates the body of the request.
body = json.dumps({"hello": "world"})

# 2. Creates an HTTP request.
request = Request(
    method="GET",
    target="/",
    protocol="HTTP/1.1",
    headers={
        "Host": "www.google.com",
        "Connection": "keep-alive",
        "Content-Length": len(body),
    },
    body=body,
)

# 3. Parses the equivalent request as the above.
request_parsed = request = Request.parse(
    (
        b"GET / HTTP/1.1\r\n"
        b"Host: www.google.com\r\n"
        b"Connection: keep-alive\r\n"
        b"Content-Length: %i\r\n"
        b"\r\n"
        b"%b"
    )
    % (len(body), body.encode(ENCODE))
)
```

5.2 Response

```python
import json

from httpsuite import Response, ENCODE

# 1. Creates the body of the request.
body = json.dumps({"hello": "world"})

# 2. Creates an HTTP response.
response = Response(
    protocol="HTTP/1.1",
    status=200,
    status_msg="OK",
    headers={
        "Host": "www.google.com",
        "Connection": "keep-alive",
        "Content-Length": len(body),
    },
    body=body,
)

# 3. Parses the equivalent response as the above.
response_parsed = Response.parse(
    (
        b"HTTP/1.1 200 OK\r\n"
        b"Host: www.google.com\r\n"
        b"Connection: keep-alive\r\n"
        b"Content-Length: %i\r\n"
        b"\r\n"
        b"%b"
    )
    % (len(body), body.encode(ENCODE))
)
```

CHAPTER

SIX

ADVANCE USE

Advance examples of `httpsuite` being used.

6.1 Sockets

```python
""" Example of using httpsuite to communicate with an external web server using sockets.

1. Open a new socket.
2. Creates the request to be sent via Request object.
3. Connect to httpbin.org via socket.
4. Send generated request to server.
5. Receive raw response from server.
6. Parse the server's response with Response object.
7. Loads the response's body via JSON.
8. Close socket.
"""

from httpsuite import Request, Response
import socket
import json

# 1. Open a new socket.
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# 2. Creates the request to be sent via Request object.
body = json.dumps({"hello": "world"})
request = Request(
    method="POST",
    target="/post",
    protocol="HTTP/1.1",
    headers={
        "Host": "httpbin.org",
        "Connection": "close",
        "Content-Length": len(body),
        "Accept": "*/*",
    },
    body=body,
)

```

(continues on next page)

(continued from previous page)

```python
# Prints the raw request.
print("====== Raw Request ======", "\n")
print(request.raw, "\n")

# 3. Connect to httpbin.org via socket.
s.connect(("httpbin.org", 80))

# 4. Send generated request to server.
s.sendall(request.raw)

# 5. Receive raw response from server.
response_raw = s.recv(4096)

# Prints the raw response.
print("====== Raw Response ======", "\n")
print(response_raw, "\n")

# 6. Parse the server's response with Response object.
response = Response.parse(response_raw)

# Prints the request and the response (pretty-print).
print("====== Request and Response ======", "\n")
print(request, "\n")
print(response, "\n")

# 7. Loads the response's body via JSON.
body = json.loads(response.body.string)

# Prints the loaded json ('dumps' for pretty-print).
print("====== Json ======", "\n")
print(json.dumps(body, indent=4))

# 8. Close socket.
s.close()
```

6.2 Microservice

```python
""" Very primative example of a socket microservice architecture using httpsuite.

Server and Client functions are ran through the 'multiprocessing' module, so
to act as two seperate entities. For the sake of clarity, only the server function
prints anything to console. Entities are documented seperately.
"""

from httpsuite import Request, Response
from multiprocessing import Process
import socket
import time


```

(continues on next page)

(continued from previous page)

```python
def server():
    """Simple socket server that uses httpsuite to interpret and reply.

    1. Opens a new socket.
    2. Binds to 127.0.0.1:8080 and waits until new connection.
    3. Accepts connection from external source.
    4. Receive the data from the client.
    5. Parse the clients request.
    6. Interpret the request.
    7. Reply to the client.
    8. Close the connection with the client.
    """

    # 1. Opens a new socket.
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    # 2. Binds to 127.0.0.1:8080 and waits until new connection.
    s.bind(("127.0.0.1", 8080))
    s.listen(1)

    # 3. Accepts connection from external source.
    conn, address = s.accept()

    print("===== Connecting With New Client =====", "\n")
    print(address, "\n")

    # 4. Receive the data from the client.
    data = conn.recv(1024)

    # 5. Parse the clients request.
    request = Request.parse(data)

    print("===== Received Data From Client =====", "\n")
    print(request, "\n")

    # 6. Interpret the request.
    response = Response(protocol="HTTP/1.1", status=200, status_msg="OK")
    if request.target == "/":
        response.body = "Homepage of the microservice."
    elif request.target == "/data":
        response.body = "You are accessing the /data directory of this microservice."
    else:
        response.status = 404
        response.status_msg = "Not Found"

    print("===== Replying to Client =====", "\n")
    print(response, "\n")

    # 7. Reply to the client.
    conn.sendall(response.raw)

    print("===== Closing Connection to Client =====", "\n")
```

(continues on next page)

(continued from previous page)

```python

    # 8. Close the connection with the client, and the server.
    conn.close()
    s.close()


def client():
    """Simple socket client that uses httpmodule to request server resource.

    1. Opens a new socket.
    2. Connects the server.
    3. Creates a valid request to send to the server.
    4. Sends the request.
    5. Receives reply from the server.
    6. Parses the reply from the server.
    7. Closes connection with the server.
    """

    # Note: Sleeps so that the socket server can boot-up before.
    time.sleep(1)

    # 1. Opens a new socket.
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    # 2. Connects the server.
    s.connect(("127.0.0.1", 8080))

    # 3. Creates a valid request to send to the server.
    request = Request(method="GET", target="/", protocol="HTTP/1.1")

    # 4. Sends the request.
    s.sendall(request.raw)

    # 5. Receives reply from the server.
    data = s.recv(1024)

    # 6. Parses the reply from the server.
    response = Response.parse(data)

    # 7. Closes connection with the server.
    s.close()


if __name__ == "__main__":
    p1 = Process(target=server)
    p2 = Process(target=client)

    p1.daemon = True
    p2.daemon = True

    p1.start()
    p2.start()
```

(continues on next page)

(continued from previous page)

```

    time.sleep(3)
    raise SystemExit
```

CHAPTER SEVEN

GLOSSARY

compile In the context of `httpsuite`, compiling means converting a `Response` or `Request` object into another more useable type, typically either `str` or `bytes`.

message The parent object of both an HTTP request and response.

modify In the context of `httpsuite`, modifying means changing attributes in either the `Request` or `Response` objects. When modifications are applied to these objects usually compilation follows.

parse In the context of `httpsuite`, parsing means interpreting an external type and creating a representative `Response` or `Request` object.

CHAPTER EIGHT

VERSIONS

8.1 v1

8.1.1 1.2.0

- Organized project.

8.1.2 1.1.0

- Revamped and cleaned the codebase.
- Update documentation.

8.1.3 1.0.6

- Updated core `Message` to include `protocol` by default.
- Removed `info.py` and place content inside `__init__.py`.
- Formatted some tests with Black.

8.1.4 1.0.5

- Fixed documentation bug relating to `sphinx_rtd_theme`.

8.1.5 1.0.4

- Fixed documentation bug relating to `m2r`.

8.1.6 `1.0.3`

- Fixed PyLint errors.
- Added `__bool__` to `Item` for accurate comparissons (i.e. `if request.body` will return `False` when no body exists).
- Modified `Message._compile` input param `format` to `frmt`.

8.1.7 `1.0.2`

- Cleaned up imports and codebase.

8.1.8 `1.0.1`

- Cleaned up imports and codebase.

8.1.9 `1.0.0`

- Initial commit.

CHAPTER NINE

LICENSE

```
MIT License

Copyright (c) 2020 Felipe Faria

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

PYTHON MODULE INDEX

INDEX