# httpsuite

Felipe Faria

Apr 01, 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 malu-able 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",
        "Content-Length": len(body),
        "Accept": "*/*",
```

(continues on next page)

(continued from previous page)

```
    },
    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
```

CHAPTER

THREE

# INTERFACE

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

CHAPTER

FOUR

# BASIC USE

## 4.1 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))
)
```

## 4.2 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

# FIVE

# ADVANCE USE

Advance examples of `httpsuite` being used.

## 5.1 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,
)

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

(continues on next page)

(continued from previous page)

```
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()
```

## 5.2 Microservice

```
""" 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


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.
```

(continues on next page)

(continued from previous page)

```
    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")

    # 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.
```

(continues on next page)

(continued from previous page)

```
    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()

    time.sleep(3)
    raise SystemExit
```

CHAPTER

SIX

# 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

# SEVEN

# VERSIONS

## 7.1 v1

### 7.1.1 1.1.0

- Update documentation.
- Added parsing for `requests` and `aiohttp` objects.
- Cleaned up code-base.

### 7.1.2 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.

### 7.1.3 1.0.5

- Fixed documentation bug relating to `sphinx_rtd_theme`.

### 7.1.4 1.0.4

- Fixed documentation bug relating to `m2r`.

### 7.1.5 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`.

### 7.1.6 1.0.2

- Cleaned up imports and codebase.

### 7.1.7 1.0.1

- Cleaned up imports and codebase.

### 7.1.8 1.0.0

- Initial commit.

CHAPTER

# EIGHT

# 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.
```

# INDEX