httptestserver Documentation

Release 0.1.0

Javier Santacruz

November 12, 2014

Contents

HTTP/HTTPS server which can be run within a Python process. Runs in a different thread along with the application exposing a simple thread-safe API, so the calling code can control how the server behaves.

Sometimes integration tests cannot do with mocking the `socket.socket` function avoiding real networking, this partially solves that problem by providing a real server which is easy to use and can perform real network communication in a controlled and reliable way.

Features:

- Runs in a different thread at the same time of your tests.
- Control server responses and behaviour.
- Access to server internal state and data after or during the request.
- HTTPs support, it bundles a self-signed certificate useful for testing.
- History of all server performed requests/responses.

Supports `python` *2.7* and *3.4*

CHAPTER 1

Functions

Functions that return a running server instance:

```
>>> server = start_server()
>>> server.host
'127.0.0.1'
```

Or context managers for limited use:

```
>>> with http_server() as server:
...     server.host
'127.0.0.1'
```

CHAPTER 2

Mixin classes

Mixins that include an working server as `self.server`.

```
import requests
from httptestserver import HttpsServerTest

class TestApplication(HttpsServerTest):

    # Test what was actually get by the server
    def test_it_should_send_headers(self):
        headers = {'key': 'value'}

        requests.get(self.default_url, headers=headers)

        assert self.server.data['headers']['key'] == 'value'

    # Control server responses
    def test_it_should_parse_json_response(self):
        self.server.data['headers'] = {'Content-Type': 'application/json'}
        self.server.data['response_content'] = "{'key': 'value'}"

        response = requests.get(self.default_url)

        assert response.json() == {'key': 'value'}

    # Make the server behave as you want
    def test_it_should_raise_timeout_at_2s_wait(self):
        self.server.data['response_timeout'] = 2

        try:
            requests.get(self.default_url, timeout=1)
        except requests.exceptions.Timeout:
            pass
        else:
            assert False

    # Access to server's requests/responses history
    def test_it_should_make_two_requests(self):
        self.server.reset()

        requests.get(self.default_url)
        requests.get(self.default_url + '2')

        assert len(self.server.history) == 2
```

```
        assert self.server.history[-1]['path'] == self.default_url + '2'
```

2.1 Development

In order get a development environment, create a virtualenv and install the desired requirements.

```
virtualenv env
env/bin/activate
pip install -r dev-requirements.txt
```

The included certificate was generated using SSL:

```
openssl req -new -x509 -keyout server.pem -out server.pem -days 40000 -nodes
```

CHAPTER 3

Tests

To run the tests just use **tox** or **nose**:

```
tox
```

```
nosetests
```

CHAPTER 4

Documentation

To generate the documentation change to the `docs` directory and run make. You need to install the `sphinx` and `changelog` packages in order to be able to run the makefile.

```
cd docs
make html
open build/html/index.html
```

4.1 Api

Functions which return a running server instance:

`httptestserver.`**`start_server`**(*host=None*, *port=None*)
Create a started HTTP server listening in *host*:*port*

Parameters

- **host** – *(default: 127.0.0.1)* Host for the server to listen.
- **port** – *default: random)* Port of the server to listen (should not be in use).

Returns A created and started `Server`

`httptestserver.`**`start_ssl_server`**(*host=None*, *port=None*, *certfile=None*, *keyfile=None*)
Create a started HTTPS server listening in *host*:*port*

It configures server certificate using *certfile* and *keyfile*.

Parameters

- **host** – *(default: 127.0.0.1)* Host for the server to listen.
- **port** – *(default: random)* Port of the server to listen (should not be in use).
- **certfile** – *(default: packaged .pem)* Path to certificate file as accepted by `HTTPServer`.
- **keyfile** – *(default: None)* Path to private key file as accepted by `HTTPServer`. Default comes bundled with *certfile*.

Returns A created and started `Server`

Context managers for short in-place usage:

`httptestserver.`**`http_server`**(**args*, ***kwds*)
Context of a started HTTP `Server`

```
with http_server() as server:
    # use server
```

See function `start_server()`.

`httptestserver.`**`https_server`**`(*args, **kwds)`
Context of a started HTTPS `Server`

```
with https_server() as server:
    # use server
```

See function `start_ssl_server()`.

The `Server` class, with all the available functionality:

class `httptestserver.`**`Server`**`(host, port, scheme='http', handler=<class httptestserver.server.Handler at 0x7fa833e3cef0>)`
HTTP Server

Starts in a child thread. Thread stops and closes when the caller does. Handles each request on a new thread, *forks* on each request.

Server state after each request can be checked as a *dict* through the thread-save attribute `data`, which is updated at the begining of each request. See `Handler` and `BaseHTTPRequestHandler` to see the information available on that *dict*.

```
>> server.data
{'requestline': 'GET /url HTTP/1.1', 'path': '/url', ...}
```

if several requests are made, their state are kept in order in the history:

```
>> server.history
[
  {'path': '/first', ..},
  {'path': '/second', ..}
]
```

About multithreading: It is necessary that each request gets serverd by a different thread, in case that more than one request is made at the same time. If any two requests are attended at the same time by the *same thread*, risk of deadlock exists.

`data`
Gives access to current server state *dict* (read-write)

List of values that can be set to control the server behaviour:

response_status An *int* with the status code of the next response.

response_headers A *dict* or a *(k, v) tuple* with all the headers to be sent on the next response.

response_content A *bytes* with the body of the next response.

response_timeout A number with the time in seconds to wait before starting a response.

response_clear *True* if server user state should be reset after responding. This is useful when responding with *3xx* redirections.

response_reset *True* if server state should be totally reset after the response.

`history`
Gives access to all the server states in a *list* (read-only)

`reset``()`
Resets all server data in `data`

response_data
All user-defined response properties

classmethod start_server(*host*, *port*)
Creates and starts a http `Server`

Parameters

- **host** – Host for the server to listen.
- **port** – Port of the server to listen (should not be in use).

Returns A created and started http `Server`

classmethod start_ssl_server(*host*, *port*, *certfile*, *keyfile*)
Creates and starts a https `Server`

Parameters

- **host** – Host for the server to listen.
- **port** – Port of the server to listen (should not be in use).
- **certfile** – Path to certificate file as accepted by `HTTPServer`.
- **keyfile** – Path to private key file as accepted by `HTTPServer`. Default it's bundled with *certfile*.

Returns A created and started https `Server`

url(*path*)
Compose a full URL to the server from the url path:

```
>> server.url('/test/url')
http://127.0.0.1:8888/test/url
```

The default handler is `Handler` but it can be subclassed and extended:

class `httptestserver.server.`**Handler**(*request*, *client_address*, *server*)
Handles all requests and collects server data

Handles all the requests on the `handle_request()` method which is also responsible for building a response.

The `Server.data` dictionary is updated on at the begining of each request with the current server state. See `BaseHTTPRequestHandler` documentation for the full list of server attributes available.

The default handler behaviour can be controlled through `Server.data`.

handle_request()
Handles server request/response

save_history()
Create a new entry in history

state
Dict with the current server state

update_state()
Copies current server state

Some mixings to start the server and use it directly from tests.

class `httptestserver.`**HttpServerTest**

options = {}

classmethod **setupClass**()

class httptestserver.**HttpsServerTest**

options = {'verify': False}

classmethod **setupClass**()

4.2 Changes

List of all the changes throughout different versions.

4.2.1 0.1.0

Released: 2014-11-12

Initial version

- **[feature]** Adds `Server` class. ¶
- **[feature]** Adds `start_server()` and `start_ssl_server()` convenience functions. ¶
- **[feature]** Adds `http_server()` and `https_server()` context managers. ¶
- **[feature]** Adds `HttpServerTest()` and `HttpsServerTest()` mixins classes to be used in testing. ¶

4.3 Indices and tables

- *genindex*
- *modindex*
- *search*

h

D

H

O

R

S

U