# pytest-pyppeteer

***Release latest***

**Yao Meng**

**Nov 27, 2020**

# CONTENTS

pytest-pyppeteer is a plugin to run pyppeteer in pytest.

CHAPTER

ONE

# DOCUMENTS

## 1.1 Installation

### 1.1.1 Requirements

pytest-pyppeteer work well with `python >= 3.7`.

### 1.1.2 Install pytest-pyppeteer

You can install pytest-pyppeteer via pip:

```
$ pip install pytest-pyppeteer
```

or install the latest one on Github:

```
$ pip install git+https://github.com/luizyao/pytest-pyppeteer.git
```

## 1.2 Quickstart

For example, query the rating of the movie **The Shawshank Redemption** on douban.com.

```
from dataclasses import dataclass


@dataclass(init=False)
class Elements:
    url = "https://movie.douban.com/"

    query = "#inp-query"
    apply = ".inp-btn > input:nth-child(1)"

    result = (
        "#root > div > div > div > div > div:nth-child(1) > div.item-root a.cover-link
↪"
    )
    rating = (
        "#interest_sectl > div.rating_wrap.clearbox > div.rating_self.clearfix >␣
↪strong"
    )
```

(continues on next page)

(continued from previous page)

```
async def test_pyppteer(pyppeteer):
    page = await pyppeteer.new_page()
    await page.goto(Elements.url)

    await page.type(Elements.query, "The Shawshank Redemption")
    await page.click(Elements.apply)

    await page.waitfor(Elements.result)
    await page.click(Elements.result)

    await page.waitfor(Elements.rating)
    rating = await page.get_value(Elements.rating)
    assert rating == 0
```

The test will be failed because of `assert rating == 0`, but we successfully got the rating of the movie, it was `9.7`.

```
$ pipenv run pytest -q tests/test_quickstart.py
F                                                                    [100%]
==================================== FAILURES ===================================
_____________________________ test_options_mark _____________________________

pyppeteer = Browser(pyppeteer_browser=<pyppeteer.browser.Browser object at␣
↪0x10917e5e0>)

    async def test_options_mark(pyppeteer):
        page = await pyppeteer.new_page()
        await page.goto("https://movie.douban.com")

        await page.type(Elements.query, "The Shawshank Redemption")
        await page.click(Elements.apply)

        await page.waitfor(Elements.result)
        await page.click(Elements.result)

        await page.waitfor(Elements.rating)
        rating = await page.get_value(Elements.rating)
>       assert rating == 0
E       AssertionError: assert '9.7' == 0

tests/test_quickstart.py:31: AssertionError
================================ warnings summary ================================
tests/test_quickstart.py::test_options_mark
tests/test_quickstart.py::test_options_mark
tests/test_quickstart.py::test_options_mark
tests/test_quickstart.py::test_options_mark
tests/test_quickstart.py::test_options_mark
tests/test_quickstart.py::test_options_mark
tests/test_quickstart.py::test_options_mark
  /Users/yaomeng/.local/share/virtualenvs/pytest-pyppeteer-KPzLwmKN/lib/python3.8/
↪site-packages/pyee/_compat.py:35: DeprecationWarning: pyee.EventEmitter is␣
↪deprecated and will be removed in a future major version; you should instead use␣
↪either pyee.AsyncIOEventEmitter, pyee.TwistedEventEmitter, pyee.
↪ExecutorEventEmitter, pyee.TrioEventEmitter, or pyee.BaseEventEmitter.
```

(continues on next page)

(continued from previous page)

```
    warn(DeprecationWarning(

-- Docs: https://docs.pytest.org/en/stable/warnings.html
============================== short test summary info ==============================
FAILED tests/test_quickstart.py::test_options_mark - AssertionError: assert '9.7' == 0
1 failed, 7 warnings in 7.61s
```

## 1.3 Usage

### 1.3.1 Command line options

#### --executable-path

You can specify the path to a Chromium or Chrome executable. otherwise pytest-pyppeteer will use the default installation location of Chrome in current platform, but now only support `win64`, `win32` and `mac` platform.

For other platforms, pyppeteer will downloads the recent version of Chromium when called first time. If you don't prefer this behavior, you can specify an exact path by override this fixture:

```
@pytest.fixture(scope="session")
def executable_path(executable_path):
    if executable_path is None:
        return "path/to/Chrome/or/Chromium"
    return executable_path
```

---

**Note:** The default installation location of Chrome in different platform:

- `win64`: C:/Program Files/Google/Chrome/Application/chrome.exe
- `win32`: C:/Program Files (x86)/Google/Chrome/Application/chrome.exe
- `mac`: /Applications/Google Chrome.app/Contents/MacOS/Google Chrome

---

#### --headless

Run browser in headless mode.

#### --args

Additional args to pass to the browser instance.

For example, specify a proxy:

```
$ pytest --args proxy-server "localhost:5555,direct://" --args proxy-bypass-list "192.
↪0.0.1/8;10.0.0.1/8"
```

Or by override the `args` fixture:

```
@pytest.fixture(scope="session")
def args(args) -> List[str]:
    return args + [
        "--proxy-server=localhost:5555,direct://",
        "--proxy-bypass-list=192.0.0.1/8;10.0.0.1/8",
    ]
```

#### --window-size

The default browser size is 800*600, you can use this option to change this behavior:

```
$ pytest --window-size 1200 800
```

`--window-size 0 0` means to starts the browser maximized.

#### --slow

Slow down the pyppeteer operate in milliseconds. Defaults to `0.0`.

### 1.3.2 No matter selector or xpath

`pyppeteer` fixture provide a `pytest_pyppeteer.models.Browser` instance, its usage is almost the same as `pyppeteer.browser.Browser`, except that it provides a new instance method: `new_page()`, which is similar to `newPage()`, but it returns a `pytest_pyppeteer.models.Page` instead of `pyppeteer.page.Page`.

`pytest_pyppeteer.models.Page`'s usage is also the same as `pyppeteer.page.Page`, but it provides some new instance methods, and override some methods. For example, you can query an element by selector or xpath in just same method `query_locator` instead of original `querySelector` and `xpath`.

You can also get an original `Page` by `pyppeteer.newPage()`.

### 1.3.3 `options` marker

You can override some command line options in the specified test.

For example, auto-open a DevTools panel:

```
import asyncio

import pytest


@pytest.mark.options(devtools=True)
async def test_marker(pyppeteer):
    await pyppeteer.new_page()
    await asyncio.sleep(2)
```

## 1.4 Advanced Usage

### 1.4.1 Control multiple browsers asynchronously

You can easily to control multiple browsers at the same time.

For example, query the **The Shawshank Redemption**'s movie and book rating on douban.com at the same time, then compare them.

```
import asyncio
from dataclasses import dataclass
from typing import TYPE_CHECKING, Callable

import pytest

if TYPE_CHECKING:
    from .models import Browser, Page


@dataclass
class Elements:
    query = "#inp-query"
    apply = ".inp-btn > input:nth-child(1)"


@dataclass
class BookElements(Elements):
    url = "https://book.douban.com/"

    result = '(//*[@class="item-root"])[1]/a'
    rating = "#interest_sectl > div > div.rating_self.clearfix > strong"


@dataclass
class MovieElements(Elements):
    url = "https://movie.douban.com/"

    result = (
        "#root > div > div > div > div > div:nth-child(1) > div.item-root a.cover-link
↪"
    )
    rating = (
        "#interest_sectl > div.rating_wrap.clearbox > div.rating_self.clearfix >␣
↪strong"
    )


async def query_rating(pyppeteer: "Browser", name: str, elements: "Elements"):
    page: Page = await pyppeteer.new_page()

    await page.goto(elements.url)

    await page.type(elements.query, name)
    await page.click(elements.apply)

    await page.waitfor(elements.result)
    await page.click(elements.result)
```

(continues on next page)

(continued from previous page)

```
    await page.waitfor(elements.rating)
    rating = await page.get_value(elements.rating)
    return rating


async def test_multiple_browsers(pyppeteer_factory: "Callable"):
    pyppeteer1 = await pyppeteer_factory()
    pyppeteer2 = await pyppeteer_factory()

    movie_rating, book_rating = await asyncio.gather(
        query_rating(pyppeteer1, "The Shawshank Redemption", MovieElements),
        query_rating(pyppeteer2, "The Shawshank Redemption", BookElements),
    )

    assert movie_rating == book_rating
```

Execute this test, and it will be failed:

```
$ pipenv run pytest -q tests/test_multiple_browsers.py
F                                                                    [100%]
=================================== FAILURES ===================================
_____________________________ test_multiple_browsers _____________________________

pyppeteer_factory = <function pyppeteer_factory.<locals>._factory at 0x1068c4700>

    async def test_multiple_browsers(pyppeteer_factory: "Callable"):
        pyppeteer1 = await pyppeteer_factory()
        pyppeteer2 = await pyppeteer_factory()

        movie_rating, book_rating = await asyncio.gather(
            query_rating(pyppeteer1, "The Shawshank Redemption", MovieElements),
            query_rating(pyppeteer2, "The Shawshank Redemption", BookElements),
        )

>       assert movie_rating == book_rating
E       AssertionError: assert '9.7' == '9.2'
E         - 9.2
E         + 9.7

tests/test_multiple_browsers.py:62: AssertionError
=============================== warnings summary ===============================
tests/test_multiple_browsers.py: 14 warnings
  /Users/yaomeng/.local/share/virtualenvs/pytest-pyppeteer-KPzLwmKN/lib/python3.8/
↪site-packages/pyee/_compat.py:35: DeprecationWarning: pyee.EventEmitter is␣
↪deprecated and will be removed in a future major version; you should instead use␣
↪either pyee.AsyncIOEventEmitter, pyee.TwistedEventEmitter, pyee.
↪ExecutorEventEmitter, pyee.TrioEventEmitter, or pyee.BaseEventEmitter.
    warn(DeprecationWarning(

-- Docs: https://docs.pytest.org/en/stable/warnings.html
============================= short test summary info ==============================
FAILED tests/test_multiple_browsers.py::test_multiple_browsers - AssertionError: as...
1 failed, 14 warnings in 17.58s
```

## 1.5 API Reference

### 1.5.1 errors module

**class ErrorMixin**(***ctx: Any*)
Bases: object

The mixin for base exceptions.

**Parameters** **ctx** (*Any*) – content variables.

**code:** **str**
Type identification code.

**msg_template:** **str**
Error message template.

**exception Error**(***ctx: Any*)
Bases: *errors.ErrorMixin*, Exception

Base-class for all exceptions raised by this plugin.

**code:** **str**

**msg_template:** **str**

**exception PathError**(*path: str*, ***kwargs: Any*)
Bases: *errors.Error*

The base-class for all path-related exceptions.

**Parameters** **path** (*str*) – path string.

**code:** **str**

**msg_template:** **str**

**exception PathNotAExecutableError**(*path: str*, ***kwargs: Any*)
Bases: *errors.PathError*

**code:** **str = 'file.not_a_executable'**

**msg_template:** **str = 'path "{path}" does not point to a executable file'**

**exception LocatorError**(*locator: str*, ***kwargs: Any*)
Bases: *errors.Error*

The base-class for all locator-related exceptions.

**Parameters** **path** (*str*) – locatore string.

**code:** **str**

**msg_template:** **str**

**exception LocatorNotAValidSelectorOrXPathError**(*locator: str*, ***kwargs: Any*)
Bases: *errors.LocatorError*

**code:** **str = 'locator.not_a_valid_selector_or_xpath'**

**msg_template:** **str = 'locator "{locator}" is not a valid selector or xpath string.'**

**exception ElementError**(*locator: str*, ***kwargs: Any*)
Bases: *errors.Error*

The base-class for all element-related exceptions.

**Parameters** **locator** (*str*) – element locator string.

**code:** **str**

**msg_template:** **str**

**exception ElementNotExistError**(*locator: str*, ***kwargs: Any*)

Bases: *errors.ElementError*

**code:** **str = 'element.not_exist'**

**msg_template:** **str = 'Element "{locator}" not exist.'**

**exception ElementTimeoutError**(*locator: str*, *timeout: int*, *action: str = 'appear'*)

Bases: *errors.ElementError*

**code:** **str = 'element.timeout'**

**msg_template:** **str = 'Wait for element "{locator}" to {action} failed: timeout {timeo**

### 1.5.2 hooks module

**async pytest_pyppeteer_runtest_makereport_call_debug**(*item: Item*) → None

Called to add debug information when each of the call(not setup/teardown) runtest phases of a test failed item. e.g. save screenshot.

**Parameters** **item** (*Item*) – the pytest item object.

**Returns** None

### 1.5.3 models module

**class ViewPort**(***, *width: int = 800*, *height: int = 600*, *deviceScaleFactor: float = 1.0*, *isMobile: bool = False*, *hasTouch: bool = False*, *isLandscape: bool = False*)

Bases: `pydantic.main.BaseModel`

Keep the consistency of each page's viewport in a browser instance.

One of the most important use is as the standard setting model for *Options.defaultViewport*.

**width:** **int**

Page width in pixels. Defaults to `800`.

**height:** **int**

Page height in pixels. Defaults to `600`.

**deviceScaleFactor:** **float**

Specify device scale factor (can be thought of as dpr). Defaults to `1.0`.

**isMobile:** **bool**

Whether the `meta viewport` tag is taken into account. Defaults to `False`.

**hasTouch:** **bool**

Specifies if viewport supports touch events. Defaults to `False`.

**isLandscape:** **bool**

Specifies if viewport is in landscape mode. Defaults to `False`.

**class Options**(*, *args: List[str] = []*, *autoClose: bool = True*, *defaultViewport: models.ViewPort = ViewPort(width=800, height=600, deviceScaleFactor=1.0, isMobile=False, hasTouch=False, isLandscape=False)*, *devtools: bool = False*, *dumpio: bool = False*, *env: dict = None*, *executablePath: Union[str, Path] = None*, *handleSIGINT: bool = True*, *handleSIGTERM: bool = True*, *handleSIGHUP: bool = True*, *headless: bool = False*, *ignoreHTTPSErrors: bool = True*, *ignoreDefaultArgs: Union[bool, List[str]] = False*, *logLevel: Optional[Union[int, str]] = None*, *slowMo: float = 0.0*, *userDataDir: str = None*)

Bases: `pydantic.main.BaseModel`

The standard setting model for pyppeteer launcher.

**args: List[str]**
Additional arguments to pass to the browser instance. The list of Chromium flags can be found here. Defaults to `list()`.

**autoClose: bool**
Automatically close browser process when script completed. Defaults to `True`.

**defaultViewport: Optional[*ViewPort*]**
Set a consistent viewport for each page. Defaults to a default *ViewPort* instance. `None` means disables the default viewport.

**devtools: bool**
Whether to auto-open a DevTools panel for each tab. If this option is `True`, the `headless` option will be set `False`. Defaults to `False`.

**dumpio: bool**
Whether to pipe the browser process stdout and stderr into `process.stdout` and `process.stderr`. Defaults to `False`.

**env: Optional[dict]**
Specify environment variables that will be visible to the browser. `None` means that same as python process. Defaults to `None`.

**executablePath: Union[str, Path]**
Path to a Chromium or Chrome executable. `None` means use the default bundled Chromium. Defaults to `None`.

**handleSIGINT: bool**
Close the browser process on *Ctrl-C*. Defaults to `True`.

**handleSIGTERM: bool**
Close the browser process on *SIGTERM*. Defaults to `True`.

**handleSIGHUP: bool**
Close the browser process on *SIGHUP*. Defaults to `True`.

**headless: bool**
Whether to run browser in headless mode. Defaults to `False`.

**ignoreHTTPSErrors: bool**
Whether to ignore HTTPS errors. Defaults to `True`.

**ignoreDefaultArgs: Union[bool, List[str]]**
If `True`, then do not use pyppeteer's default args. If a list is given, then filter out the given default args. Dangerous option; use with care. Defaults to `False`.

**logLevel: Optional[Union[int, str]]**
Log level to print logs. `None` means that same as the root logger. Defaults to `None`.

**slowMo:** *float*
Slow down operations by the specified amount of milliseconds. useful so that you can see what is going on. Defaults to `0.0`.

**userDataDir:** *Optional[str]*
Path to a User Data Directory. Defaults to `None`.

**class Config**
Bases: `object`

Control the behaviours of pydantic model.

**arbitrary_types_allowed = True**
whether to allow arbitrary user types for fields (they are validated simply by checking if the value is an instance of the type). If False, RuntimeError will be raised on model declaration.

**classmethod validate_executable_path**(*path: Optional[str]*) → Optional[str]
Validate that the specified `executablePath` must point to an executable.

**Parameters** **path** (*str*) – path string.

**Returns** path string.

**Raises** ***PathNotAExecutableError*** – if path does not point to a executable file.

**class Browser**(*, *pyppeteer_browser: pyppeteer.browser.Browser*)
Bases: `pydantic.main.BaseModel`

**pyppeteer_browser:** *pyppeteer.browser.Browser*
a pyppeteer browser object.

**class Config**
Bases: `object`

Control the behaviours of pydantic model.

**arbitrary_types_allowed = True**
whether to allow arbitrary user types for fields (they are validated simply by checking if the value is an instance of the type). If False, RuntimeError will be raised on model declaration.

**async new_page**() → *models.Page*
Make new page on this browser and return its object.

**Returns** a *Page* object.

**class Page**(*, *pyppeteer_page: pyppeteer.page.Page*)
Bases: `pydantic.main.BaseModel`

**pyppeteer_page:** *pyppeteer.page.Page*
a pyppeteer page object.

**class Config**
Bases: `object`

Control the behaviours of pydantic model.

**arbitrary_types_allowed = True**
whether to allow arbitrary user types for fields (they are validated simply by checking if the value is an instance of the type). If False, RuntimeError will be raised on model declaration.

**async query_locator**(*locator: str*) → Optional[ElementHandle]
Get the element which match `locator`.

If no element matches the `locator`, return `None`.

**Parameters** **locator** (*str*) – a selector or xpath string

**Returns** an element handle or `None`.

**async waitfor**(*locator: str*, *visible: bool = True*, *hidden: bool = False*, *timeout: int = 30000*) → None

Wait until element which matches `locator`.

**Parameters**

- **locator** (*str*) – a selector or xpath string.
- **visible** (*bool*) – Wait for element to be present in DOM and to be visible; i.e. to not have `display: none` or `visibility: hidden` CSS properties. Defaults to `True`.
- **hidden** (*bool*) – Wait for element to not be found in the DOM or to be hidden, i.e. have `display: none` or `visibility: hidden` CSS properties. Defaults to `False`.
- **timeout** (*int*) – Maximum time to wait for in milliseconds. Defaults to 30000 (30 seconds). Pass `0` to disable timeout.

**Returns** None

**Raises** ***ElementTimeoutError*** – Timeout exceeded while wait for `locator`.

**async type**(*locator: str*, *text: str*, *delay: int = 0*, *clear: bool = False*)

Focus the element which matches `locator` and then type text.

**Parameters**

- **locator** – a selector or xpath string.
- **text** – what you want to type into.
- **delay** (*int*) – specifies time to wait between key presses in milliseconds. Defaults to 0.
- **clear** (*bool*) – whether to clear existing content befor typing. Defaults to `False`.

**Returns**

**async click**(*locator: str*, *button: str = 'left'*, *click_count: int = 1*, *delay: int = 0*)

Click the center of the element which matches `locator`.

**Parameters**

- **locator** (*str*) – a selector or xpath string.
- **button** (*str*) – `left`, `right`, of `middle`. Defaults to `left`.
- **click_count** (*int*) – Defaults to 1.
- **delay** (*int*) – Time to wait between `mousedown` and `mouseup` in milliseconds. Defaults to 0.

**Returns** None

**Raises** ***ElementNotExistError*** – if the element which matches `locator` is not found.

**async get_value**(*locator: str*) → str

Get the element `value` or `innerText` which matches `locator`.

**Parameters** **locator** (*str*) – a selector or xpath string.

**Returns** the element `value` or `innerText` string.

**Raises** ***ElementNotExistError*** – if the element which matches `locator` is not found.

### 1.5.4 plugin module

**pytest_addhooks**(*pluginmanager: PytestPluginManager*) → None
Add new hooks.

**Parameters** **pluginmanager** (*_pytest.config.PytestPluginManager*) –

**Returns** None

**pytest_configure**(*config: Config*) → None
Perform initial configuration as follows:

- Add `pytest.mark.option` marker to the ini-file option.

**Parameters** **config** (*_pyest.config.Config*) – pytest config object.

**Returns** None

---

**Note:** This is a `pytest hook function` which is called for every plugin and initial conftest file after command line options have been parsed. After that, the hook is called for other conftest files as they are imported.

---

**pytest_collection_modifyitems**(*items: List[Item]*) → None
Modify the items collected by pytest as follows:

- Because all test items using pyppeteer should be an asyncio coroutine, here the pytest.mark.asyncio marker is automatically added to each test item collected by pytest.

**Parameters** **items** (*List[pytest.Item]*) – the list of items objects collected by pytest.

**Returns** None

---

**Note:** This is a `pytest hook function` which is called after collection of all test items is completed.

---

**add_asyncio_marker**(*item: Item*) → Item
Add pytest.mark.asyncio marker to the specified item.

If the marker is already exists, return the item directly.

**Parameters** **item** (*pytest.Item*) – the pytest item object.

**Returns** the marked item object.

**is_coroutine**(*obj: Any*) → bool
Check to see if an object is really an asyncio coroutine.

**Parameters** **obj** (*Any*) – any object.

**Returns** *True* or *False*.

**pytest_runtest_makereport**(*item: Item*) → None
Implement this pytest hook in wrapper mode, the added behaviors as follows:

- Register a new *hooks.pytest_pyppeteer_runtest_makereport_call_debug()* hook which called when a actual failing test calls not setup/teardown.

**Parameters** **item** (*pytest.Item*) – the pytest item object.

**Returns** None

**pytest_addoption**(*parser: Parser*) → None
Register new command line arguments and ini-file values.

Create a new command line option group named *pyppeteer*, and add the following new options in it:

- `--executable-path`: path to a Chromium or Chrome executable.
- `--headless`: run browser in headless mode.
- `--args`: additional args to pass to the browser instance. more details refer to *`args()`* fixture.
- `--window-size`: set the initial browser window size. Defaults to 800 * 600. `--window-size 0 0` means to starts the browser maximized.

**Parameters** **parser** (*_pytest.config.argparsing.Parser*) – parser for command line arguments and ini-file values.

**Returns** None

---

**Note:** This is a `pytest hook function` which be called once at the beginning of a test run to register argparse-style options and ini-style config values.

There are two ways to register new options, respectively:

- To register a command line option, call `parser.addoption(...)`.
- To register an ini-file option, call `parser.addini(...)`.

And the options can later be accessed through the `Config` object, respectively:

- To retrieve the value of a command line option, call `config.getoption(name)`.
- To retrieve a value read from an ini-style file, call `config.getini(name)`.

The `Config` object is passed around on many pytest internal objects via the `.config` attribute or can be retrieved as the `pytestconfig` fixture.

---

**executable_path**(*pytestconfig: Config*) → Optional[str]
`Session-scoped fixture` that return Chrome or Chromium executable path.

The fixture behaviors follow this procedure:

1. Return the value passed in from command line option of *–executable-path*, if it's not `None`.
2. Return the default installation location of Chrome in current platform, but now only support `win64`, `win32` and `mac` platform.

   For other platforms, pyppeteer will downloads the recent version of Chromium when called first time. If you don't prefer this behavior, you can specify an exact path by overwrite this fixture:

   Example:

```
@pytest.fixture(scope="session")
def executable_path(executable_path):
    if executable_path is None:
        return "path/to/Chrome/or/Chromium"
        return executable_path
```

**Parameters** **pytestconfig** (*_pytest.config.Config*) – a session-scoped fixture that return config object.

**Returns** return Chrome or Chromium executable path string. but if current platform isn't supported, return `None`.

**args** (*pytestconfig: Config*) → List[str]

`Session-scoped fixture` that return a list of additional args in the List of Chromium Command Line Arguments to pass to the browser instance.

You can use it by command-line option:

Example:

```
$ pytest --args proxy-server "localhost:5555,direct://" --args proxy-bypass-list
↪"192.0.0.1/8;10.0.0.1/8"
```

Or overwrite it in your test:

Example:

```
@pytest.fixture(scope="session")
def args(args) -> List[str]:
    return args + [
        "--proxy-server=localhost:5555,direct://",
        "--proxy-bypass-list=192.0.0.1/8;10.0.0.1/8",
    ]
```

**Parameters** **pytestconfig** (*_pytest.config.Config*) – a session-scoped fixture that return config object.

**Returns** a list of arguments string. return `list()` if no `--args` passed in the command-line.

**session_options** (*pytestconfig: Config*, *args: List[str]*, *executable_path: str*) → Options

`Session-scoped fixture` that return a *models.Options* object used to initialize browser.

**Parameters**

- **pytestconfig** (*_pytest.config.Config*) – a session-scoped fixture that return config object.
- **args** (*List[str]*) – a session-scoped fixture that return a list of additional args to pass to the browser instance.
- **executable_path** (*str*) – a session-scoped fixture that return Chrome or Chromium executable path.

**Returns** a *models.Options* object used to initialize browser.

**options** (*request: FixtureRequest*, *session_options: Options*) → Options

`Function-scoped fixture` that return a *models.Options* object used to initialize browser.

This fixture contains all of *session_options()*, plus any options specified by the options markers. Any change to these options will apply only to the tests covered by scope of the fixture override.

Example:

```
@pytest.mark.options(devtools=True)
async def test_options_mark(pyppeteer):
    ...
```

**Parameters**

- **request** (*_pytest.fixture.FixtureRequest*) – A request object gives access to the requesting test context.
- **session_options** (Options) – a *models.Options* object from *session_options()*.

**Returns** a *models.Options* object used to initialize browser.

**get_options_from_markers**(*item: Item*) → dict
Get the options from the `options` markers of test item. And there are only apply on the current test item.

**Parameters** **item** (*Item*) – the test item object.

**Returns** an dict contains options.

**pyppeteer_factory**(*options: pytest_pyppeteer.models.Options*) → Callable
`Function-scoped fixture` that return a pyppeteer browser factory.

**Parameters** **options** (Options) – a *models.Options* object used to initialize browser

**Yield** a pyppeteer browser factory.

**pyppeteer**(*pyppeteer_factory: Callable*) → pytest_pyppeteer.models.Browser
`Function-scoped fixture` that return a pyppeteer browser instance.

**Parameters** **pyppeteer_factory** (*Callable*) – pyppeteer factory

**Yield** a pyppeteer browser instance.

### 1.5.5 utils module

**current_platform**() → str
Return current platform name.

**Returns** the platform name. only be one of `mac` `win64` `win32`.

**Raises** **OSError** – if current platform is not supported.

**parse_locator**(*locator: str*) → tuple
validate that the locator string must a valid css or xpath.

**Parameters** **locator** – a locator string.

**Returns** a tuple contains locator_type and locator_string.

**Raises** **LocatorNotAValidSelectorOrXPath** – locator is not a valid selector or xpath string.

# PYTHON MODULE INDEX

# INDEX

## N



## O



## P



## Q



## S



## T



## U



## V



## W