

---

# **seldom**

***Release 2.6.0***

**Mar 30, 2022**



---

## Contents

---

<b>1</b>	<b>User guide</b>	<b>3</b>
1.1	Installation . . . . .	3
1.2	Create project . . . . .	4
1.3	Quick Start . . . . .	5
1.4	seldom API . . . . .	9
1.5	Advanced Usage . . . . .	16
1.6	Other . . . . .	23
1.7	HTTP Interface Testing . . . . .	26
1.8	Database Operation . . . . .	34
<b>2</b>	<b>Indices and tables</b>	<b>37</b>



Web UI/HTTP automated testing framework based on unittest.

Features:

- Provide scaffolding to quickly generate automated test items
- Start and close the browser globally, reducing the number of browser launches
- Provides support for multiple data file parameterization
- Support for use case failure/error reruns
- Automatically generate HTML test reports
- Support for HTTP interface testing (v 2.0)

Using *seldom* to write test Web UI automation tests is very simple.

```
import seldom

class YouTest(seldom.TestCase):

    def test_case(self):
        """a simple test case """
        self.open("https://www.baidu.com")
        self.type(id_="kw", text="seldom")
        self.click(css="#su")
        self.assertInTitle("seldom")

if __name__ == '__main__':
    seldom.main()
```



### 1.1 Installation

The first step in using any software is to install it correctly.

- `pip install seldom`

`seldom` has uploaded it to [pypi.org](https://pypi.org) , You can install it using the *pip* command.

```
> pip install seldom
```

If you want to keep up with the latest version, you can install with github repository url:

```
> pip install -U git+https://github.com/defnngj/seldom.git@master
```

You can see the dependent libraries in the *requires.txt* file.

```
colorama>=0.4.3
XTestRunner>=1.3.1
selenium>=4.0.0
parameterized>=0.7.0
poium==1.0.0
openpyxl==3.0.3
pyyaml>=5.1
requests>=2.22.0
jsonschema>=3.2.0
jmespath>=0.10.0
webdriver-manager>=3.5.0
```

Finally, used the *seldom -v* command to view the version.

```
> seldom -v
seldom 2.6.0
```

## 1.2 Create project

In this chapter we will quickly experience the *seldom* project

### 1.2.1 Create case

Create a Python file: *test\_sample.py* .

```
import seldom

class YouTest(seldom.TestCase):

    def test_case(self):
        """a simple test case """
        self.open("https://www.baidu.com")
        self.type(id="kw", text="seldom")
        self.click(css="#su")
        self.assertInTitle("seldom")

if __name__ == '__main__':
    seldom.main()
```

If you have an environment for *Selenium*, you can now run this use case.

### 1.2.2 Automated project creation

*seldom* provides scaffolding to help us quickly create Web UI automation projects.

1. view the help:

```
> seldom -h
usage: seldom [-h] [-v] [-project PROJECT] [-r R] [-m M] [-install INSTALL]

WebUI automation testing framework based on Selenium.

optional arguments:
  -h, --help            show this help message and exit
  -v, --version          show version
  -project PROJECT      Create an Seldom automation test project.
  -h2c H2C              HAR file converts an interface test case.
  -r R                  run test case
  -m M                  run tests modules, classes or even individual test methods
                        from the command line
  -install INSTALL      Install the browser driver, For example, 'chrome',
                        'firefox'.
```

2. Create project:

```
> seldom -project mypro
```

3. View directory structure:

```
mypro/
├─ test_dir/
│   └─ test_sample.py
```

(continues on next page)



(continued from previous page)

```

├── test_data/
│   └── data.json
├── reports/
└── run.py

```

- `test_dir/` Test case directory.
- `test_dir/` Test data file directory.
- `reports/` Test Report directory.
- `run.py` Run the test file.

## 1.3 Quick Start

### 1.3.1 Download Browser Driver

As with *Selenium*, before you can run automated tests using *seldom*, you need to configure the browser driver, This step is very important.

#### Automatically download

*Seldom* provides automatic download driven by *chrome/firefox/ie/edge/opera* browser.

```

> seldom -install chrome
> seldom -install firefox
> seldom -install ie
> seldom -install edge
> seldom -install opera

```

- By default, download to the current `C://Users/username/.wdm/drivers/` directory.
- Chrome: *ChromeDriver* Mirror image of Taobao used.
- Safari: *safaridriver* (macOS,default path:`/usr/bin/safaridriver`)

### 1.3.2 `main()` Method

`main()` method is *seldom* run test entry method, It provides some of the most basic and important configurations.

```

import seldom

# ...

if __name__ == '__main__':

    seldom.main(path=".",
                browser="chrome",
                base_url=None,
                report=None,
                title="project name",
                tester="Anonymous",
                description="Environment description",
                debug=False,
                rerun=0,

```

(continues on next page)

(continued from previous page)

```
        save_last_run=False,
        language="en",
        timeout=None,
        whitelist=[],
        blacklist=[]
    )
```

### Parameter specification

- path : Specifies the test directory or file.
- browser : Run browser name(for example: “chrome”, “firefox”).
- base\_url : A parameter to test the HTTP interface testing, setting the global URL.
- report : The name of the custom test report, The default format is *YYYY\_mm\_dd\_HH\_MM\_SS\_result.html*.
- title : Test report title.
- title : Specifies the tester, default ‘Anonymous’.
- description : Test report description.
- debug : Debug mode, set to True does not generate test HTML tests, default is *False*.
- rerun : Sets the number of failed reruns, Default is 0.
- save\_last\_run : Set to save only the last result, default to *False*.
- language : Set the HTML report in English and Chinese, default ‘en’, Chinese *zh-CN*.
- timeout : Sets the timeout, Default 10 seconds.
- whitelist : The use case *label* sets the whitelist.
- blacklist : Use case *label* Sets the blacklist.

## 1.3.3 Run Test

### Run under a terminal (recommended)

Create the file *run.py*, And import *main()* method.

```
import seldom

seldom.main()
```

*main()* Method Run the use case in the current file by default.

```
> python run.py      # Run with the Python command
> seldom -r run.py    # Run with the Seldom command
```

### Set the running directory, file

You can specify the directory or file to run with the *path* parameter.

### Run a class or method

The *seldom -m* command can provide a more granular run.

```
> seldom -m test_sample      # test_sample.py file
> seldom -m test_sample.SampleTest  # SampleTest Class
> seldom -m test_sample.SampleTest.test_case  # test_case method
```

### 1.3.4 Failed Rerun

*seldom* support failed reruns, as well as screenshots.

```
import seldom

class YouTest(seldom.TestCase):

    def test_case(self):
        """a simple test case """
        self.open("https://www.baidu.com")
        self.type(id_="kw", text="seldom")
        self.click(css="#su_error")
        #...

if __name__ == '__main__':
    seldom.main(rerun=3, save_last_run=False)
```

#### Parameters

- rerun : Sets the number of failed reruns, Default is 0.
- save\_last\_run : Sets to save only the last result, default to *False*.

#### Run logs

```
> python test_sample.py
```

#### The test report

To view the screenshots, click the *show* button in the report.

### 1.3.5 Test Report

*seldom* automatically generates HTML test reports by default.

- Befor running the test case

```
mypro/
|--- test_sample.py
```

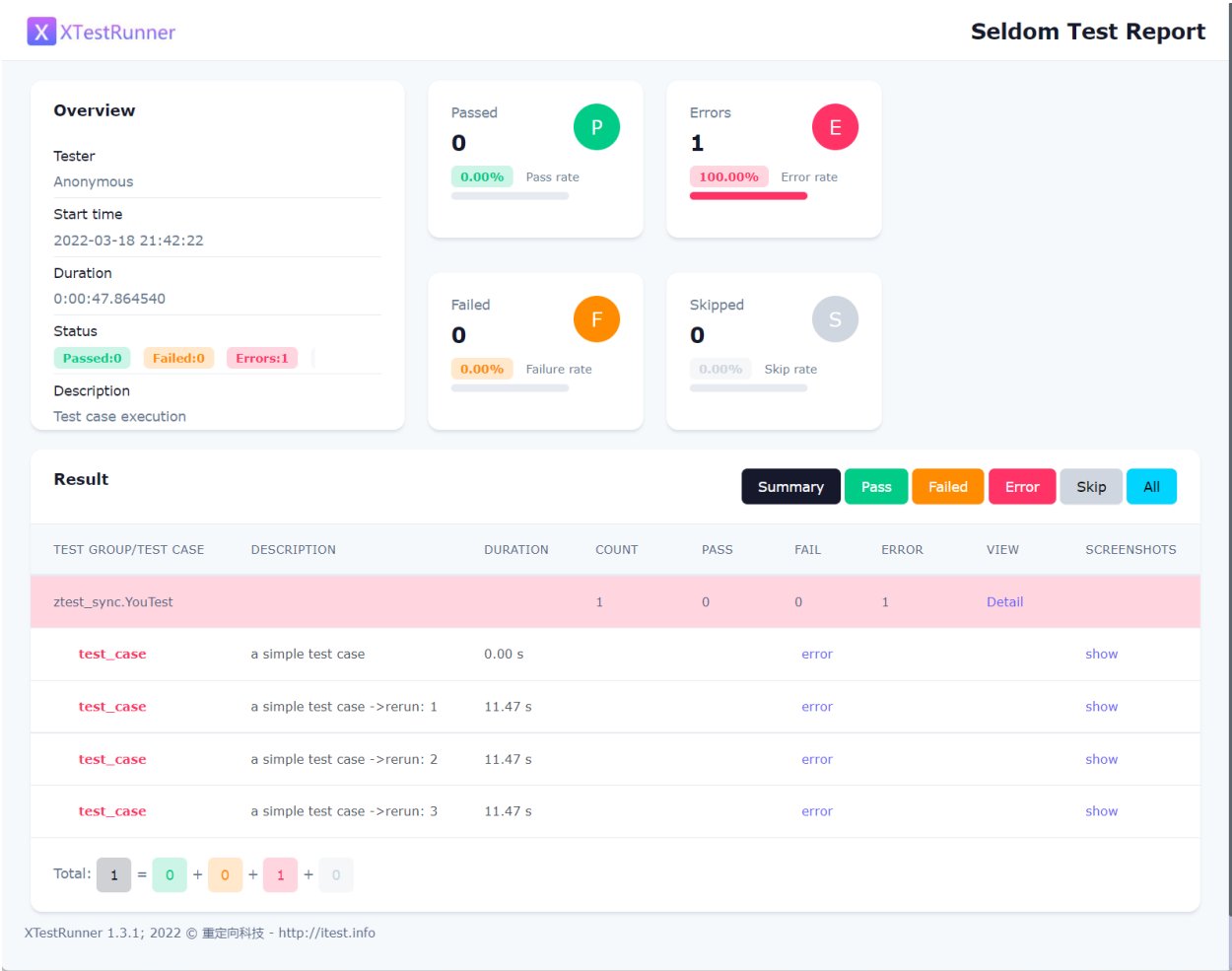
- After running the test case

```
mypro/
|-- reports/
|   |-- 2020_01_01_11_20_33_result.html
|-- test_sample.py
```

Open the *2020\_01\_01\_11\_20\_33\_result.html* test report through a browser, View the test results.

#### Debug mode

if you don't want to generate and HTML report every time you run, You can opent the *debug* mode.



```
import seldom

seldom.main(debug=True)
```

### Define Test Reports

```
import seldom

seldom.main(report="report.html",
            title="xxxx",
            tester="username",
            description="run evn:windows 10/ chrome")
```

- report: Configure the report name and path.
- title: Customize the title of the report.
- tester: Customize the current tester.
- description: Add report information.

### XML Test Report

If you want to generate a report in XML format, just change the suffix name *.xml* of the report.

```
import seldom

seldom.main(report="report.xml")
```

## 1.4 seldom API

### 1.4.1 Find Element

Selenium provides 8 ways of find element, which are consistent with Selenium.

- id\_
- name
- class\_name
- tag
- link\_text
- partial\_link\_text
- css
- xpath

### Demo

```
self.type(id_="kw", text="seldom")
self.type(name="wd", text="seldom")
self.type(class_name="s_ipt", text="seldom")
self.type(tag="input", text="seldom")
self.type(xpath="//input[@id='kw']", text="seldom")
self.type(css="#kw", text="seldom")
```

(continues on next page)

(continued from previous page)

```
self.click(link_text="hao123")
self.click(partial_link_text="hao")
```

## Help Information

- [CSS Selectors](#)
- [xpath Syntax](#)

## Index

Sometimes a single element cannot be found by a single location, then you can specify the index of an element via *index*.

```
self.type(tag="input", index=7, text="seldom")
```

*tag="input"* Matches a set of elements, *index=7* Specifies the eighth element in the set, *index* default subscript 0.

## 1.4.2 Fixture

A test fixture represents the preparation needed to perform one or more tests, and any associated cleanup actions.

*seldom* provides a way to implement fixtures.

### start & end

Fixture for each test case.

```
import seldom

class TestCase(seldom.TestCase):

    def start(self):
        print("start case")

    def end(self):
        print("end case")

    def test_search_seldom(self):
        self.open("https://www.baidu.com")
        self.type_enter(id="kw", text="seldom")

    def test_search_poiu(self):
        self.open("https://www.baidu.com")
        self.type_enter(id="kw", text="poiuh")
```

### start\_class & end\_class

Fixture for each test class.

```
import seldom

class TestCase(seldom.TestCase):

    @classmethod
    def start_class(cls):
```

(continues on next page)

(continued from previous page)

```

    print("start test class")

    @classmethod
    def end_class(cls):
        print("end test class")

    def test_search_seldom(self):
        self.open("https://www.baidu.com")
        self.type_enter(id_="kw", text="seldom", clear=True)

    def test_search_poiium(self):
        self.open("https://www.baidu.com")
        self.type_enter(id_="kw", text="poiium", clear=True)

```

**Warning:** Don't write the use case steps into the fixture method!  
 Because it **is not** part of a use case, the test report will **not** be generated **if** the\_  
 ↪ steps **in** it fail to run.

### 1.4.3 Assertion

*seldom* provides a set of assertion methods for Web pages.

#### Deom

```

# Asserts is equals to "title"
self.assertTitle("title")

# Asserts contains "title"
self.assertInTitle("title")

# Asserts is equals to "title"
self.assertUrl("url")

# Asserts contains "url"
self.assertInUrl("url")

# Asserts that the page contains "text"
self.assertText("text")

# Asserts that the page not contains "text"
self.assertNotText("text")

# Assert that the warning message is equal to "text"
self.assertAlertText("text")

# Asserts whether an element exists
self.assertElement(css="#kw")

# Asserts if the element does not exist
self.assertNotElement(css="#kwasdfasdfa")

```

## 1.4.4 Skipping tests and expected failures

The following decorators and exception implement test skipping and expected failures:

### Method

- `@seldom.skip(reason)` : Unconditionally skip the decorated test. `reason` should describe why the test is being skipped.
- `@seldom.skip_if(condition, reason)` : Skip the decorated test if `condition` is true.
- `@seldom.skip_unless(condition, reason)` : Skip the decorated test unless `condition` is true.
- `@seldom.expected_failure` : Mark the test as an expected failure or error. If the test fails or errors it will be considered a success. If the test passes, it will be considered a failure.

### Demo

```
import seldom

@seldom.skip("skip this use test class")
class YouTest(seldom.TestCase):

    @seldom.skip("skip this case")
    def test_case(self):
        # ...

if __name__ == '__main__':
    seldom.main()
```

## 1.4.5 WebDriver API

*Seldom* simplifies the API, Make it easier for you to navigate Web pages.

Most APIs are provided by *WebDriver* class:

```
import seldom

class TestCase(seldom.TestCase):

    def test_seldom_api(self):
        # Accept warning box.
        self.accept_alert()

        # Adds a cookie to your current session.
        self.add_cookie({'name' : 'foo', 'value' : 'bar'})

        # Adds a cookie to your current session.
        cookie_list = [
            {'name' : 'foo', 'value' : 'bar'},
            {'name' : 'foo', 'value' : 'bar'}
        ]
        self.add_cookie(cookie_list)

        # Clear the contents of the input box.
        self.clear(css="#el")
```

(continues on next page)



(continued from previous page)

```

# It can click any text / image can be clicked
# Connection, check box, radio buttons, and even drop-down box etc..
self.click(css="#el")

# Mouse over the element.
self.move_to_element(css="#el")

# Click the element by the link text
self.click_text("")

# Simulates the user clicking the "close" button in the titlebar of a popup_
↪window or tab.
self.close()

# Delete all cookies in the scope of the session.
self.delete_all_cookies()

# Deletes a single cookie with the given name.
self.delete_cookie('my_cookie')

# Dismisses the alert available.
self.dismiss_alert()

# Double click element.
self.double_click(css="#el")

# Execute JavaScript scripts.
self.execute_script("window.scrollTo(200,1000);")

# Setting width and height of window scroll bar.
self.window_scroll(width=300, height=500)

# Setting width and height of element scroll bar.
self.element_scroll(css=".class", width=300, height=500)

# open url.
self.open("https://www.baidu.com")

# Gets the text of the Alert.
self.get_alert_text

# Gets the value of an element attribute.
self.get_attribute(css="#el", attribute="type")

# Returns information of cookie with ``name`` as an object.
self.get_cookie()

# Returns a set of dictionaries, corresponding to cookies visible in the_
↪current session.
self.get_cookies()

# Gets the element to display, The return result is true or false.
self.get_display(css="#el")

# Get a set of elements
self.get_element(css="#el", index=0)

```

(continues on next page)

(continued from previous page)

```

# Get element text information.
self.get_text(css="#e1")

# Get window title.
self.get_title

# Get the URL address of the current page.
self.get_url

# Set browser window maximized.
self.max_window()

# Mouse over the element.
self.move_to_element(css="#e1")

# open url.
self.open("https://www.baidu.com")

# Quit the driver and close all the windows.
self.quit()

# Refresh the current page.
self.refresh()

# Right click element.
self.right_click(css="#e1")

# Saves a screenshots of the current window to a PNG image file.
self.screenshots() # Save to HTML report
self.screenshots('/Screenshots/foo.png') # Save to the specified directory

# Saves a element screenshot of the element to a PNG image file.
self.element_screenshot(css="#id") # Save to HTML report
self.element_screenshot(css="#id", file_path='/Screenshots/foo.png') # Save_
→to the specified directory

"""
Constructor. A check is made that the given element is, indeed, a SELECT tag.
→If it is not,
then an UnexpectedTagNameException is thrown.
<select name="NR" id="nr">
    <option value="10" selected="">10 dollar</option>
    <option value="20">20 dollar</option>
    <option value="50">50 dollar</option>
</select>
"""
self.select(css="#nr", value='20')
self.select(css="#nr", text='20 dollar')
self.select(css="#nr", index=2)

# Set browser window wide and high.
self.set_window(100, 200)

# Submit the specified form.
self.submit(css="#e1")

```

(continues on next page)

(continued from previous page)

```

# Switch to the specified frame.
self.switch_to_frame(css="#el")

# Returns the current form machine form at the next higher level.
# Corresponding relationship with switch_to_frame () method.
self.switch_to_frame_out()

# Switches focus to the specified window.
# This switches to the new windows/tab (0 is the first one)
self.switch_to_window(1)

# Operation input box.
self.type(css="#el", text="selenium")

# Implicitly wait.All elements on the page.
self.wait(10)

# Setting width and height of window scroll bar.
self.window_scroll(width=300, height=500)

```

## 1.4.6 Keys

Sometimes we need to use the keyboard, For example: enter, “backspace”, “TAB”, “ctrl/command + a”, ctrl/command + c and so on.

*seldom* provides a set of keyboard operations.

### Demo

```

import seldom

class Test(seldom.TestCase):

    def test_key(self):
        self.open("https://www.baidu.com")

        self.Keys(css="#kw").input("seldommm")

        self.Keys(id_="kw").backspace()

        self.Keys(id_="kw").input("github")

        self.Keys(id_="kw").select_all()

        self.Keys(id_="kw").cut()

        self.Keys(id_="kw").paste()

        self.Keys(id_="kw").enter()

if __name__ == '__main__':
    seldom.main()

```

## 1.5 Advanced Usage

### 1.5.1 Random Test Data

*seldom* provides a method for randomly capturing test data.

#### Demo

```
import seldom
from seldom import testdata

class YouTest(seldom.TestCase):

    def test_case(self):
        """a simple test case """
        word = testdata.get_word()
        self.open("https://www.baidu.com")
        self.type(id="kw", text=word)
        self.click(css="#su")
        self.assertInTitle(word)

if __name__ == '__main__':
    seldom.main()
```

Get a random word by *get\_word()* and search for that word.

#### More method

- first\_name()
- last\_name()
- username()
- get\_birthday()
- get\_date()
- get\_digits()
- get\_email()
- get\_float()
- get\_now\_time()
- get\_past\_time()
- get\_future\_time()
- get\_past\_datetime()
- get\_future\_datetime()
- get\_int()
- get\_int32()
- get\_int64()
- get\_md5()
- get\_uuid()

- `get_word()`
- `get_words()`
- `get_phone()`

## 1.5.2 Data-driven Best Practices

If you automate a function when the test data is different but the steps are the same, you can use parameterization to save test code.

### @data

method of parameterizing test cases.

```
import seldom
from seldom import data

class BaiduTest(seldom.TestCase):

    @data([
        (1, 'seldom'),
        (2, 'selenium'),
        (3, 'unittest'),
    ])
    def test_baidu(self, _, keyword):
        """
        used parameterized test
        """
        self.open("https://www.baidu.com")
        self.type(id_="kw", text=keyword)
        self.click(css="#su")
        self.assertTitle(keyword+"_" + _)
```

### @data\_class

Parameterizes a test class by setting attributes on the class.

```
import seldom
from seldom import data_class

@data_class(
    ("keyword", "assert_tile"),
    [("seldom", "seldom_"),
     ("python", "python_")]
)
class YouTest(seldom.TestCase):

    def test_case(self):
        """a simple test case """
        self.open("https://www.baidu.com")
        self.type(id_="kw", text=self.keyword)
        self.click(css="#su")
        self.assertTitle(self.assert_tile)
```

### CSV data file

*seldom* support for parameterization of CSV files.

*data.csv* file contents:

username	password
admin	admin123
guest	guest123

```
import seldom
from seldom import file_data

class YouTest(seldom.TestCase):

    @file_data("data.csv", line=2)
    def test_login(self, username, password):
        """a simple test case """
        print(username)
        print(password)
        # ...
```

- file: The name of the CSV file.
- line: Start reading line 1 by default.

### Excel data file

*seldom* support for parameterization of *excel* files.

```
import seldom
from seldom import file_data

class YouTest(seldom.TestCase):

    @file_data("data.xlsx", sheet="Sheet1", line=2)
    def test_login(self, username, password):
        """a simple test case """
        print(username)
        print(password)
        # ...
```

- file : The name of the Excel file.
- sheet: Excel sheet name, default to *sheet1*.
- line : Start reading line 1 by default.

### JSON data file

*seldom* support for parameterization of *JSON* files.

json file:

```
{
  "login1": [
    ["admin", "admin123"],
    ["guest", "guest123"]
  ],
  "login2": [
    {
```

(continues on next page)

(continued from previous page)

```

        "username": "Tom",
        "password": "tom123"
    },
    {
        "username": "Jerry",
        "password": "jerry123"
    }
]
}

```

Note: 'login1' and 'login2' are called in the same way. The difference is that the former is more concise while the latter is easier to read.

python code:

```

import seldom
from seldom import file_data

class YouTest(seldom.TestCase):

    @file_data("data.json", key="login1")
    def test_login(self, username, password):
        """a simple test case """
        print(username)
        print(password)
        # ...

```

- file : The name of the JSON file..
- key: Specifies the key of the dictionary. By default, parsing the entire JSON file is not specified.

### YAML file data

*seldom* support for parameterization of *YAML* files.

data.yaml file:

```

login1:
  - - admin
    - admin123
  - - guest
    - guest123
login2:
  - username: Tom
    password: tom123
  - username: Jerry
    password: jerry123

```

Like JSON usage, YAML is much more compact to write.

python code:

```

import seldom
from seldom import file_data

class YouTest(seldom.TestCase):

```

(continues on next page)

(continued from previous page)

```
@file_data("data.yaml", key="login1")
def test_login(self, username, password):
    """a simple test case """
    print(username)
    print(password)
    # ...
```

- file : The name of the YAML file.
- key: Specifies the key of the dictionary. By default, parsing the entire YAML file is not specified.

### ddt library

Seldom supports third party parameterized libraries:[ddt](#).

installation:

```
> pip install ddt
```

Create the test file *test\_data.json*:

```
{
  "test_data_1": {
    "word": "seldom"
  },
  "test_data_2": {
    "word": "unittest"
  },
  "test_data_3": {
    "word": "selenium"
  }
}
```

In *seldom* use *ddt*.

```
import seldom
from ddt import ddt, file_data

@ddt
class YouTest(seldom.TestCase):

    @file_data("test_data.json")
    def test_case(self, word):
        """a simple test case """
        self.open("https://www.baidu.com")
        self.type(id_="kw", text=word)
        self.click(css="#su")
        self.assertInTitle(word)

if __name__ == '__main__':
    seldom.main()
```

See the ddt documentation for more usage:<https://ddt.readthedocs.io/en/latest/example.html>



### 1.5.3 Page Objects Design Patterns

poium Is *Page objects* design pattern best practice.

installation:

```
> pip install poium==1.0.0
```

Use ‘seldom’ and ‘poium’ together

```
import seldom
from poium import Page, Element

class BaiduPage(Page):
    """baidu page"""
    search_input = Element(id_="kw")
    search_button = Element(id_="su")

class BaiduTest(seldom.TestCase):
    """Baidu serach test case"""

    def test_case(self):
        """
        A simple test
        """
        page = BaiduPage(self.driver)
        page.get("https://www.baidu.com")
        page.search_input = "seldom"
        page.search_button.click()
        self.assertInTitle("seldom")

if __name__ == '__main__':
    seldom.main()
```

### 1.5.4 Automatic Email

Demo

```
import seldom
from seldom import SMTP

# ...

if __name__ == '__main__':
    seldom.main()
    smtp = SMTP(user="send@126.com", password="abc123", host="smtp.126.com")
    smtp.sendmail(to="receive@mail.com", subject='Email title')
```

- subject: Email title, default: *Seldom Test Report*.
- to: Addressee email, Add multiple recipients commas ‘,’ to separate.

### 1.5.5 Use Case Dependencies

While it is not recommended to write dependent use cases, there are times when you can't completely avoid them.

#### depend

*depend* Decorators are used to set dependent use cases.

```
import seldom
from seldom import depend

class TestDepend(seldom.TestCase):

    def test_001(self):
        # ...

    @depend("test_001")
    def test_002(self):
        # ...

    @depend("test_002")
    def test_003(self):
        # ...

if __name__ == '__main__':
    seldom.main(debug=True)
```

*test\_002* depends on *test\_001*, and *test\_003* depends on *test\_002*.

#### if\_depend

Customize the state of the use case, and the dependent use case chooses whether to skip.

```
import seldom
from seldom import if_depend

class TestIfDepend(seldom.TestCase):
    Test001 = True

    def test_001(self):
        self.open("https://www.baidu.com")
        TestIfDepend.Test001 = False # Change Test001 to False

    @if_depend("Test001")
    def test_002(self):
        self.open("http://news.baidu.com/")

if __name__ == '__main__':
    seldom.main(debug=True)
```

### 1.5.6 Use case classification label

This function is implemented in seldom version 2.4.0.

#### demo

```
# test_label.py
import seldom
from seldom import label

class MyTest(seldom.TestCase):

    @label("base")
    def test_label_base(self):
        self.assertEqual(1+1, 2)

    @label("slow")
    def test_label_slow(self):
        self.assertEqual(1, 2)

    def test_no_label(self):
        self.assertEqual(2+3, 5)

if __name__ == '__main__':
    # seldom.main(debug=True, whitelist=["base"]) # whitelist
    seldom.main(debug=True, blacklist=["slow"]) # blacklist
```

If you only run the use cases labeled *base*, set the whitelist.

If you only want to block the use cases labeled *slow*, set a blacklist.

## 1.6 Other

### 1.6.1 Runs On More Browsers

*seldom* supports running automated tests on different browsers

```
import seldom

# ...

if __name__ == '__main__':
    seldom.main(browser="chrome") # chrome, The default
    seldom.main(browser="firefox") # firefox browser
    seldom.main(browser="opera") # opera browser
    seldom.main(browser="edge") # edge browser
    seldom.main(browser="safari") # safari browser
```

### 1.6.2 Mobile Web Mode

*seldom* also supports the Mobile Web model.

```
import seldom

# ...

if __name__ == '__main__':
    seldom.main(browser="iPhone 6") # iPhone 6
```

Type of device supported:

```
PHONE_LIST = [
    'iPhone 5', 'iPhone 6', 'iPhone 7', 'iPhone 8', 'iPhone 8 Plus',
    'iPhone X', 'Pixel 2', 'Pixel XL', "Galaxy S5"
]
PAD_LIST = ['iPad', 'iPad Pro']
```

## 1.6.3 Headless Mode

Firefox and Chrome support ‘headless’ mode, Enable headless mode for browsing.

```
import seldom
from seldom import ChromeConfig

#...

if __name__ == '__main__':
    ChromeConfig.headless = True
    seldom.main(browser="chrome")
```

The Firefox browser is configured similarly.

## 1.6.4 Browser configuration

In order to meet the personalized requirements, such as disabling the browser plug-in, setting the browser proxy, etc. So, open up these capabilities with the arguments of the ChromeConfig class.

```
import seldom
from seldom import ChromeConfig
from selenium.webdriver import ChromeOptions

# ...

if __name__ == '__main__':
    chrome_options = ChromeOptions()
    chrome_options.add_argument('--ignore-certificate-errors')
    ChromeConfig.options = chrome_options
    seldom.main(browser="chrome")
```

## 1.6.5 Selenium Grid

1. Install the Java environment
2. More configuration, [Selenium Server](#).

```
> java -jar selenium-server-standalone-3.141.59.jar

12:30:37.138 INFO [GridLauncherV3.parse] - Selenium server version: 3.141.59, ↵
↵ revision: e82be7d358
12:30:37.204 INFO [GridLauncherV3.lambda$buildLaunchers$3] - Launching a standalone_↵
↵ Selenium Server on port 4444
2020-10-10 12:30:37.245:INFO::main: Logging initialized @301ms to org.seleniumhq.
↵ jetty9.util.log.StdErrLog
```

(continues on next page)

(continued from previous page)

```
12:30:37.417 INFO [WebDriverServlet.<init>] - Initialising WebDriverServlet
12:30:37.497 INFO [SeleniumServer.boot] - Selenium Server is up and running on port_
↪ 4444
```

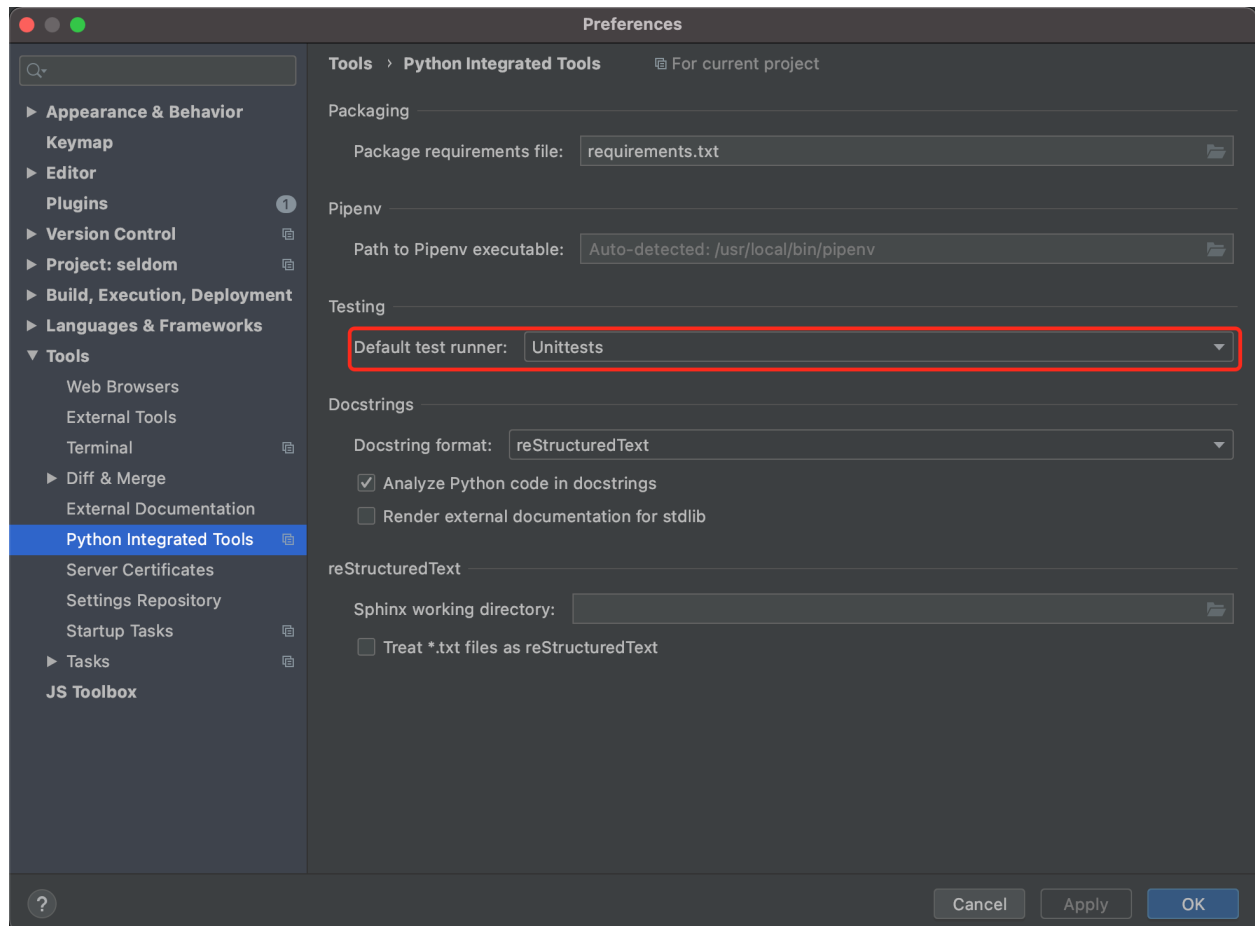
```
import seldom
from seldom import ChromeConfig

# ...
if __name__ == '__main__':
    ChromeConfig.command_executor = "http://127.0.0.1:4444/wd/hub"
    seldom.main(browser="chrome")
```

- More configuration, [Selenium Grid doc](#).

## 1.6.6 Run the test in PyCharm

1. Configure the test case to run through unittest.



2. Select the test class or use case execution in the file.

Warning: Running the browser that the use case opens requires manual closing, and *seldom* does not do the use case closing action.



## 1.7 HTTP Interface Testing

*seldom* has many advantages in doing interface testing.

- Support HTML/XML test reports
- Support parameterization
- Support generating random data

*seldom 2.0* added support for automated testing of HTTP interfaces..

*Seldom* compatible [Requests](#) API.

seldom	requests
self.get()	requests.get()
self.post()	requests.post()
self.put()	requests.put()
self.delete()	requests.delete()

### 1.7.1 Seldom VS Request+unittest

Let's take a look at how unittest + requests automate interfaces:

```

import unittest
import requests

class TestAPI(unittest.TestCase):

    def test_get_method(self):
        payload = {'key1': 'value1', 'key2': 'value2'}
        r = requests.get("http://httpbin.org/get", params=payload)
        self.assertEqual(r.status_code, 200)

if __name__ == '__main__':
    unittest.main()

```

This is actually pretty neat. The same use case, implemented in *seldom*.

```
# test_req.py
import seldom

class TestAPI(seldom.TestCase):

    def test_get_method(self):
        payload = {'key1': 'value1', 'key2': 'value2'}
        self.get("http://httpbin.org/get", params=payload)
        self.assertStatusCode(200)

if __name__ == '__main__':
    seldom.main()
```

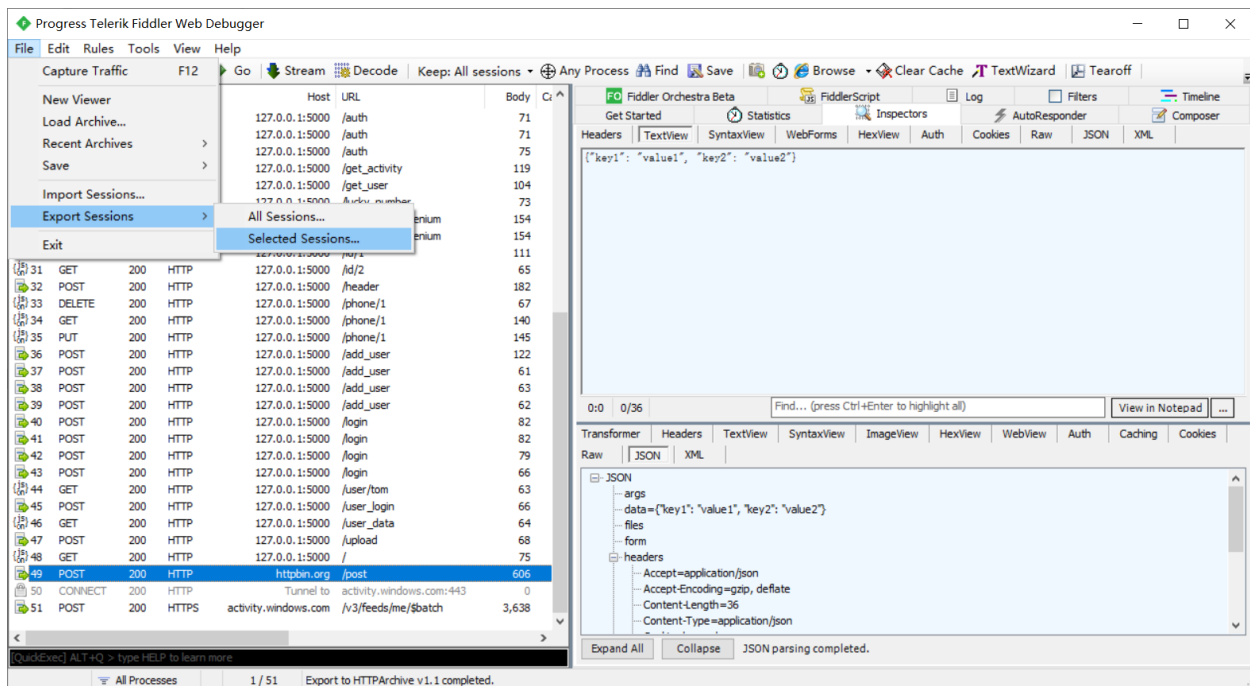
The advantages of *seldom* are assertions, logging, and reporting.

## 1.7.2 HAR TO CASE

For those unfamiliar with the Requests library, writing interface test cases through *seldom* can still be a bit difficult. Thus, Cement provided the order for *har* file to turn *case*.

First, open the Fiddler tool to grab the packet and select a particular request.

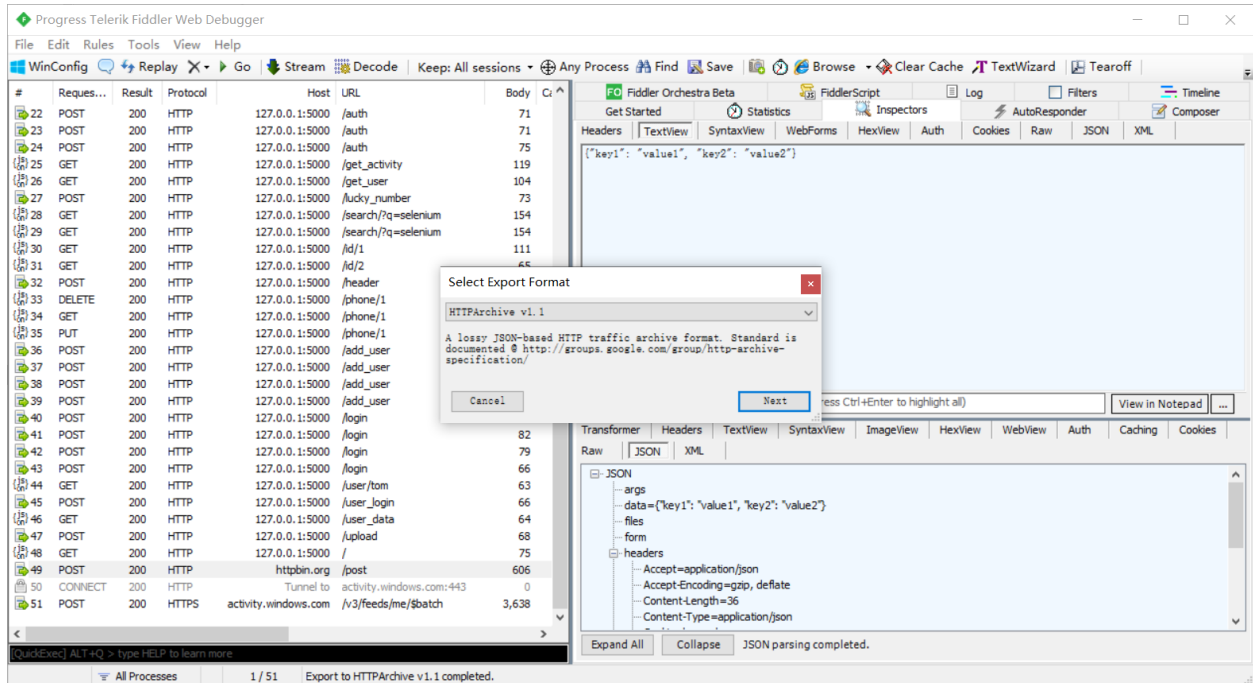
Then, select the menu bar: *file -> Export Sessions -> Selected Sessions...*



Select the file format to export.

Click on the *next* save as the *demo.har* file.

Finally, the script file *demo.py* is converted by '*seldom -h2c*'.



```
> seldom -h2c .\demo.har
.\demo.py
2021-06-14 18:05:50 [INFO] Start to generate testcase.
2021-06-14 18:05:50 [INFO] created file: D:\.\demo.py
```

*demo.py* file.

```
import seldom

class TestRequest(seldom.TestCase):

    def start(self):
        self.url = "http://httpbin.org/post"

    def test_case(self):
        headers = {"User-Agent": "python-requests/2.25.0", "Accept-Encoding": "gzip,
        ↳deflate", "Accept": "application/json", "Connection": "keep-alive", "Host":
        ↳httpbin.org", "Content-Length": "36", "Origin": "http://httpbin.org", "Content-Type
        ↳": "application/json", "Cookie": "lang=zh"}
        cookies = {"lang": "zh"}
        self.post(self.url, json={"key1": "value1", "key2": "value2"},
        ↳headers=headers, cookies=cookies)
        self.assertStatusCode(200)

if __name__ == '__main__':
    seldom.main()
```





(continued from previous page)

```
class TestAPI(seldom.TestCase):

    def test_assert_json(self):
        payload = {'name': 'tom', 'hobby': ['basketball', 'swim']}
        self.get("http://httpbin.org/get", params=payload)
        assert_json = {'args': {'hobby': ['swim', 'basketball'], 'name': 'tom'}}
        self.assertJSON(assert_json)
```

### Running logs

```
test_get_method (test_req.TestAPI) ...
----- Request -----
url: http://httpbin.org/get          method: GET
----- Response -----
type: json
{'args': {'hobby': ['basketball', 'swim'], 'name': 'tom'}, 'headers': {'Accept': '*/'
→, 'Accept-Encoding': 'gzip, deflate', 'Host': 'httpbin.org', 'User-Agent': 'python-
→requests/2.22.0', 'X-Amzn-Trace-Id': 'Root=1-608a896d-48fac4f6139912ba01d2626f'},
→'origin': '183.178.27.36', 'url': 'http://httpbin.org/get?name=tom&hobby=basketball&
→hobby=swim'}}
Assert data has not key: headers
Assert data has not key: origin
Assert data has not key: url
ok

-----
Ran 1 test in 1.305s

OK
```

*seldom* will also prompt you for fields that have not been asserted.

### assertPath

‘assertPath’ is an assertion method based on ‘jmespath’, very powerful.

jmespath:<https://jmespath.org/specification.html>

The interface returns the result:

```
{
  "args": {
    "hobby":
      ["basketball", "swim"],
    "name": "tom"
  }
}
```

Assertion using PATH:

```
import seldom

class TestAPI(seldom.TestCase):

    def test_assert_path(self):
        payload = {'name': 'tom', 'hobby': ['basketball', 'swim']}
```

(continues on next page)

(continued from previous page)

```
self.get("http://httpbin.org/get", params=payload)
self.assertPath("name", "tom")
self.assertPath("args.hobby[0]", "basketball")
```

### assertSchema

Sometimes you don't care what the data itself is, but you need to assert the type of the data. 'assertSchema' is an assertion method based on 'JSONSchema'.

jsonschema: <https://json-schema.org/learn/>

The interface returns the result:

```
{
  "args": {
    "hobby":
      ["basketball", "swim"],
    "name": "tom",
    "age": "18"
  }
}
```

Assertion using *assertSchema*:

```
import seldom

class TestAPI(seldom.TestCase):

    def test_assert_schema(self):
        payload = {"hobby": ["basketball", "swim"], "name": "tom", "age": "18"}
        self.get("/get", params=payload)
        schema = {
            "type": "object",
            "properties": {
                "args": {
                    "type": "object",
                    "properties": {
                        "age": {"type": "string"},
                        "name": {"type": "string"},
                        "hobby": {
                            "type": "array",
                            "items": {
                                "type": "string"
                            }
                        }
                    }
                }
            }
        },
        self.assertSchema(schema)
```

Again, the assertions provided by *seldom* are very flexible and powerful.

## 1.7.5 Interface Data Dependency

In scenario testing, we need to call the next interface using data from the previous interface.

### Sample 1

```
import seldom

class TestRespData(seldom.TestCase):

    def test_data_dependency(self):
        """
        Test for interface data dependencies
        """
        headers = {"X-Account-Fullname": "bugmaster"}
        self.get("/get", headers=headers)
        self.assertStatusCode(200)

        username = self.response["headers"]["X-Account-Fullname"]
        self.post("/post", data={'username': username})
        self.assertStatusCode(200)
```

`self.response` Used to record the result returned by the last interface, just use it.

### Sample 2

Defining common modules

```
# common.py
from seldom import HttpRequest

class Common(HttpRequest):

    def get_login_user(self):
        """
        Call the interface to get the user name
        """
        headers = {"X-Account-Fullname": "bugmaster"}
        self.get("http://httpbin.org/get", headers=headers)
        user = self.response["headers"]["X-Account-Fullname"]
        return user
```

Create classes that inherit *HttpRequest* class calls using Http request methods 'get/post/put/delete'.

Referencing public modules

```
import seldom
from common import Common

class TestRequest(seldom.TestCase):

    def start(self):
        self.c = Common()

    def test_case(self):
        # get_login_user()
        user = self.c.get_login_user()
        print(user)
        self.post("http://httpbin.org/post", data={'username': user})
        self.assertStatusCode(200)
```

(continues on next page)

(continued from previous page)

```
if __name__ == '__main__':
    seldom.main(debug=True)
```

### 1.7.6 Data-Driver

*seldom* has a strong data-driven nature and is very convenient for interface testing.

#### @data

```
import seldom
from seldom import data

class TestDDT(seldom.TestCase):

    @data([
        ("key1", 'value1'),
        ("key2", 'value2'),
        ("key3", 'value3')
    ])
    def test_data(self, key, value):
        """
        Data-Driver Tests
        """
        payload = {key: value}
        self.post("/post", data=payload)
        self.assertEqual(200)
        self.assertEqual(self.response["form"][key], value)
```

#### @file\_data

data file:

```
{
  "login": [
    ["admin", "admin123"],
    ["guest", "guest123"]
  ]
}
```

code file:

```
import seldom
from seldom import file_data

class TestDDT(seldom.TestCase):

    @file_data("data.json", key="login")
    def test_data(self, username, password):
        """
        Data-Driver Tests
        """
        payload = {username: password}
        self.post("http://httpbin.org/post", data=payload)
```

(continues on next page)

(continued from previous page)

```
self.assertStatusCode(200)
self.assertEqual(self.response["form"][username], password)
```

More like data files(csv/excel/yaml), [View](#)

## 1.7.7 Random Test Data

SELDOM provides a method of randomly generating test data to generate some commonly used data.

```
import seldom
from seldom import testdata

class TestAPI(seldom.TestCase):

    def test_data(self):
        phone = testdata.get_phone()
        payload = {'phone': phone}
        self.get("http://httpbin.org/get", params=payload)
        self.assertPath("args.phone", phone)
```

For more types of test data, [View](#)

## 1.8 Database Operation

seldom supports simple operations of SQLite3 and MySQL database.

sqlite3	MySQL
execute_sql()	execute_sql()
query_sql()	query_sql()
delete()	delete()
insert()	insert()
select()	select()
update()	update()
init_table()	init_table()
close()	close()

### 1.8.1 Connecting DB

#### Connect to SQLite3 database

```
from seldom.db_operation import SQLiteDB

db = SQLiteDB(r"D:\learnAPI\db.sqlite3")
```

#### Connect to MySQL database

1. Install the PyMySQL driver

```
> pip install pymysql
```

## 2. connect to databases

```
from seldom.db\_operation import MySQLDB

db = MySQLDB(host="127.0.0.1", port="3306", user="root", password="123",
database="db_name")
```

### 1.8.2 Operation Method

- execute\_sql

The SQL statement was executed, but no result was returned.

```
db.execute_sql("INSERT INTO table_name (id, name) VALUES (1, 'tom') ")
db.execute_sql("UPDATE table_name SET name = 'jack' WHERE id=1")
db.execute_sql("DELETE FROM table_name WHERE id = 1")
```

- query\_sql

The query SQL statement is executed and the query result is returned.

```
ret = db.query_sql("select * from table_name")
print(ret)
```

- delete

Delete table data.

```
db.delete(table="user", where={"id":1})
```

- insert

Insert a data.

```
data = {'id': 1, 'username': 'admin', 'password': "123"},
db.insert(table="user", data=data)
```

- select

Query data in the table.

```
result = db.select(table="user", where={"id":1, "name": "tom"})
print(result)
```

- update

Update table data.

```
db.update(table="user", data={"name":"new tom"}, where={"name": "tom"})
```

- init\_table

Bulk inserts, clearing table data before inserting.

```
datas = {
    'api_event': [
        {'id': 1, 'name': 'Redmi PRO launch1'},
        {'id': 2, 'name': 'Redmi2 PRO launch'},
```

(continues on next page)

(continued from previous page)

```
        {'id': 3, 'name': 'Redmi3 PRO launch'},
        {'id': 4, 'name': 'Redmi4 PRO launch'},
        {'id': 5, 'name': 'Redmi5 PRO launch'},
    ],
    'api_guest': [
        {'id': 1, 'real_name': 'alen'},
        {'id': 2, 'real_name': 'jack'},
        {'id': 3, 'real_name': 'tom'},
    ]
}

db.init_table(datas)
```

- close

Close the database connection.

```
db.close()
```



## CHAPTER 2

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`