selenose Documentation

Release 1.3

ShiningPanda

October 20, 2014

Selenose provides a set of Selenium related plugins/tasks for nose/django-jenkins developed by ShiningPanda.

The use of these plugins/tasks is detailed bellow, but let's have a look on the *installation process* first.

CHAPTER 1

Installation

On most UNIX-like systems, you'll probably need to run these commands as root or using `sudo`.

Install selenose using setuptools/distribute:

```
$ easy_install selenose
```

Or pip:

```
$ pip install selenose
```

It can take a while as Selenium server's jar is downloaded on the fly during installation.

If you plan to use django-jenkins, note that Django 1.4+ is required (support for in-browser testing frameworks).

CHAPTER 2

Nose

Selenose provides two Selenium related plugins for nose:

- *Selenium Server Plugin* starts a Selenium Server before running tests, and stops it at the end of the tests.
- *Selenium Driver Plugin* provides a Selenium Web Driver to the tests.

2.1 Selenium Server Plugin

This plugin starts a Selenium Server before running tests, and stops it at the end of the tests.

To enable it, add `--with-selenium-server` to the nose command line:

```
$ nose --with-selenium-server
```

You can also add the `with-selenium-server` option under the `nosetests` section of the configuration file (`setup.cfg`, `~/.noserc` or `~/nose.cfg`):

```
[nosetests]
with-selenium-server = true
```

Options for Selenium Server can be found by downloading its jar and typing:

```
$ java -jar /path/to/seleniumserver/libs/selenium-server-standalone-X.X.X.jar -h
```

Most common options are:

- `-port <nnnn>`: the port number the Selenium Server should use (default 4444),
- `-log <logFileName>`: writes lots of debug information out to a log file,
- `-debug`: enable debug mode.

To set the server options, add a `selenium-server` section to the configuration file (`setup.cfg`, `~/.noserc` or `~/nose.cfg`). Option names are obtained by removing the initial dash, for instance to run:

```
$ java -jar selenium-server-standalone-X.X.X.jar -debug -log selenium-server.log
```

Add the following options to the configuration:

```
[selenium-server]
debug = true
log = selenium-server.log
```

In your test, just create a new `Remote` Web Driver calling the server and that's it:

```
import nose
import unittest

from selenium import webdriver

class TestCase(unittest.TestCase):

    def test(self):
        driver = webdriver.Remote(desired_capabilities=webdriver.DesiredCapabilities.FIREFOX)
        try:
            driver.get('http://www.google.com')
            # Your test here...
        finally:
            driver.quit()

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

2.2 Selenium Driver Plugin

This plugin provides a Selenium Web Driver to Selenium tests.

2.2.1 Flag Selenium tests

This plugin only provides Web Drivers to Selenium test. To declare a Selenium test:

- Either make your test case inherit from `selenose.cases.SeleniumTestCase`,
- Or set a `enable_selenium_driver` flag to `True`:

```
class TestCase(unittest.TestCase):
    enable_selenium_driver = True
```

2.2.2 Enable the plugin

To enable this plugin, add `--with-selenium-driver` on the nose command line:

```
$ nose --with-selenium-driver
```

You can also add the `with-selenium-driver` option under the `nosetests` section to the configuration file (`setup.cfg`, `~/.noserc` or `~/nose.cfg`):

```
[nosetests]
with-selenium-driver = true
```

But enabling it is not enough, a *Web Driver environment* is also required.

2.2.3 Web Driver environment

An environment declares all the necessary parameters to create a new Web Driver.

To create a new environment `sample`, add a `selenium-driver:sample` section to the configuration file (`setup.cfg`, `~/.noserc` or `~/nose.cfg`) with at least a `webdriver` option:

```
[selenium-driver:sample]
webdriver = firefox
```

This `webdriver` option defines the Web Driver to use. Here are the available values:

- **`chrome` for Chrome, allowing the following options in configuration:**
  - `executable_path` (optional): path to `chromedriver` executable,
  - `port` (optional),
  - `desired_capabilities` (optional),
- **`firefox` for Firefox, allowing the following options in configuration:**
  - `timeout` (optional),
- **`ie` for Internet Explorer, allowing the following options in configuration:**
  - `port` (optional),
  - `timeout` (optional),
- **`remote` to delegate to a Selenium Server (started by *Selenium Server Plugin*?), allowing the following options in configura**
  - `command_executor` (required): url of the server (`http://127.0.0.1:4444/wd/hub` by default),
  - `desired_capabilities` (required): the desired browser, it could be the lower case field name of `selenium.webdriver.DesiredCapabilities` such as `firefox`, `htmlunitwithjs`... or a comma separated key/value list such as `browserName=firefox,platform=ANY`.

To enable an environment, add `--selenium-driver` on the nose command line:

```
$ nose --with-selenium-driver --selenium-driver=sample
```

You can also add the `selenium-driver` option under the `nosetests` section to the configuration file (`setup.cfg`, `~/.noserc` or `~/nose.cfg`):

```
[nosetests]
with-selenium-driver = true
selenium-driver = sample

[selenium-driver:sample]
webdriver = firefox
```

Selenose also provides a set of predefined but overridable environments:

```
[selenium-driver:chrome]
webdriver = chrome

[selenium-driver:ie]
webdriver = ie

[selenium-driver:firefox]
webdriver = firefox

[selenium-driver:remote-htmlunit]
webdriver = remote
desired_capabilities = htmlunit

[selenium-driver:remote-htmlunitwithjs]
```

```
webdriver = remote
desired_capabilities = htmlunitwithjs

[selenium-driver:remote-opera]
webdriver = remote
desired_capabilities = opera

[selenium-driver:remote-...]
webdriver = remote
desired_capabilities = ...
```

2.2.4 Writing tests

The Web Driver is directly available with `self.driver` and there is no need to cleanup after use, selenose will do it for you:

```python
import nose

from selenose.cases import SeleniumTestCase

class TestCase(SeleniumTestCase):

    def test(self):
        self.driver.get('http://www.google.com')
        # Your test here...

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

2.3 Combining Server & Driver

To combine a Selenium Server and a Selenium Driver plugin, just enable them both: the `command_executor` option of the `remote` Web Driver will know the correct value to reach the Selenium Server.

Django Jenkins

Selenose provides two Selenium related tasks for django-jenkins:

- *Selenium Server Task* starts a Selenium Server before running tests, and stops it at the end of the tests.
- *Selenium Driver Task* provides a Selenium Web Driver to the tests.

Note that Django 1.4+ support for in-browser testing frameworks is required.

3.1 Selenium Server Task

This task starts a Selenium Server before running tests, and stops it at the end of the tests.

To enable it, edit your `settings.py` and append `selenose.tasks.selenium_server` to `JENKINS_TASKS`:

```
JENKINS_TASKS = [
    # Other tasks...
    'selenose.tasks.selenium_server',
]
```

If this setting does not exist yet, do not forget to create it with the default tasks:

```
JENKINS_TASKS = [
    'django_jenkins.tasks.run_pylint',
    'django_jenkins.tasks.with_coverage',
    'django_jenkins.tasks.django_tests',
    'selenose.tasks.selenium_server',
]
```

Options for Selenium Server are the same than for the nose *Selenium Server Plugin*. Set them in a `setup.cfg` located in the current working directory, for instance:

```
[selenium-server]
debug = true
log = selenium-server.log
```

You can also specify the path to the configuration file with the `--selenose-config` option on the `manage.py jenkins` command line:

```
$ python manage.py jenkins --help
[...]
  selenose.tasks.selenium_server:
    --selenose-config=SELENOSE_CONFIGS
```

```
                        Load selenose configuration from config file(s). May
                        be specified multiple times; in that case, all config
                        files will be loaded and combined.
```

In your tests, just create a new `Remote` Web Driver calling the server and that's it:

```
from django.test import LiveServerTestCase

from selenium import webdriver

class TestCase(LiveServerTestCase):

    @classmethod
    def setUpClass(cls):
        cls.driver = webdriver.Remote(desired_capabilities=webdriver.DesiredCapabilities.FIREFOX)
        super(BaseTestCase, cls).setUpClass()

    @classmethod
    def tearDownClass(cls):
        super(BaseTestCase, cls).tearDownClass()
        cls.driver.quit()

    def test(self):
        driver.get(self.live_server_url)
```

3.2 Selenium Driver Task

This task provides a Selenium Web Driver to Selenium tests.

To enable it, edit your `settings.py` and append `selenose.tasks.selenium_driver` to `JENKINS_TASKS`:

```
JENKINS_TASKS = [
    # Other tasks...
    'selenose.tasks.selenium_server',
]
```

If this setting does not exist yet, do not forget to create it with the default tasks:

```
JENKINS_TASKS = [
    'django_jenkins.tasks.run_pylint',
    'django_jenkins.tasks.with_coverage',
    'django_jenkins.tasks.django_tests',
    'selenose.tasks.selenium_driver',
]
```

But enabling this task is not enough, a *Web Driver environment* is also required.

The *Web Driver environment* are defined in a `setup.cfg` located in the current working directory, for instance:

```
[selenium-driver:sample]
webdriver = firefox
```

You can also specify the path to the configuration file containing the environments with the `--selenose-config` option on the `manage.py jenkins` command line:

```
$ python manage.py jenkins --help
[...]
```

```
  selenose.tasks.selenium_driver:
    --selenose-config=SELENOSE_CONFIGS
                        Load selenose configuration from config file(s). May
                        be specified multiple times; in that case, all config
                        files will be loaded and combined.
    --selenium-driver=SELENIUM_DRIVER
                        Enable the provided environment.
```

To enable an environment, use the `--selenium-driver` option on the `manage.py jenkins` command line:

```
$ python manage.py jenkins --selenium-driver=sample
```

Then the Web Driver is directly available in you tests with `self.driver` and there is no need to cleanup after use, selenose will do it for you:

```
from selenose.cases import LiveServerTestCase

class TestCase(LiveServerTestCase):

    def test(self):
        self.driver.get(self.live_server_url)
        # Your test here...
```

3.3 Combining Server & Driver

To combine a Selenium Server and a Selenium Driver task, just enable them both in the settings: the `command_executor` option of the `remote` Web Driver will know the correct value to reach the Selenium Server.

```
JENKINS_TASKS = [
    # Other tasks...
    'selenose.tasks.selenium_server',
    'selenose.tasks.selenium_driver',
]
```

Tips

When writing tests, it's convenient to start a Selenium Server manually to reduce setup time when running tests. To do so, execute:

```
$ selenium-server
Starting... done!

Quit the server with CONTROL-C.
```

Then type `CONTROL-C` or `CTRL-BREAK` to stop the server.

In this case, run your tests neither with the *Selenium Server Plugin* not with the *Selenium Server Task*.