Selenium 2 WebDriver

Automatyczne testy funkcjonalne

Piotr Tynecki

Politechnika Białostocka

Wydział Informatyki

Białystok 2014

Co to jest Selenium i do czego służy?

- **Selenium** to popularne narzędzie do automatyzowania operacji wykonywanych przez przeglądarkę.

- Głównym zastosowaniem Selenium są testy aplikacji webowych, w szczególności ich frontentu.

- Za pomocą tego narzędzia możemy pokryć testami bardziej frontendową część aplikacji, np. kod JavaScript - coś czego zwykłe testy nie są w stanie obsłużyć.

http://docs.seleniumhq.org/

Co to jest Selenium 2 Webdriver i do czego służy?

- **Selenium 2 Webdriver** to zewnętrzna biblioteka, która łączy w sobie sterownik do danej przeglądarki oraz serwer Selenium.

- Webdriver dostarcza użytkownikowi gotowe API pozwalające na interakcję z przeglądarką.

- Dostarczony interfejs jest minimalistyczny i bardzo prosty w użyciu.

http://docs.seleniumhq.org/projects/webdriver/

„Środowisko anaboliczne” dla Selenium

Selenium obsługuje:

- **przeglądarki internetowe**: Firefox, IE, Safari, Opera i Google Chrome*.
- **systemy operacyjne**: Windows, OS X, Linux i Solaris.
- **języki programowania**: C#, Java, Perl, PHP, Python i Ruby.

Selenium Core napisano w JavaScript, zatem (w teorii) powinien działać również z innymi OS i przeglądarkami.

* Google Chrome Driver - https://code.google.com/p/selenium/wiki/ChromeDriver

http://python.org/

http://selenium-python.readthedocs.org/

Pobieranie i instalacja Selenium

Tworzenie środowiska wirtualnego pb:
virtualenv pb

Aktywacja środowiska wirtualnego pb:
source pb/bin/activate

Instalacja pakietu selenium:
pip install selenium

Pobieranie ChromeDrive:
wget http://chromedriver.storage.googleapis.com/2.9/chromedriver_linux64.zip

Rozpakowanie ChromeDrive:
unzip chromedriver_linux64.zip

Przykład 1 - asercja (zawieranie)

```python
# -*- coding: utf-8 -*-

import os

from selenium import webdriver


class SeleniumTest:
    def __init__(self, src, drive_path):
        self.driver = webdriver.Chrome(executable_path=drive_path)

        self.driver.get(src)

    def __del__(self):
        self.driver.close()

    def assert_title(self, title):
        assert title in self.driver.title


if __name__ == "__main__":
    drive_path = "%s/chromedriver" % os.getcwd()

    st = SeleniumTest("http://www.python.org", drive_path)
    st.assert_title("Python")
```

Uruchomienie:

(pb)katharsis@Toshi:~/workspace/pb$ python selenium-test-1.py

Przykład 2 - nawigowanie

Otwarcie strony

```
driver.get(url)
```

Drag and drop

```
element = driver.find_element_by_name("source")
target = driver.find_element_by_name("target")

from selenium.webdriver import ActionChains

action_chains = ActionChains(driver)
action_chains.drag_and_drop(element, target)
```

Przełączanie się pomiędzy oknami

```
driver.switch_to_window("windowName")
```

Przełączanie się pomiędzy ramkami

```
driver.switch_to_frame("frameName")
driver.switch_to_default_content()
```

Przełączanie się pomiędzy oknami popup

```
alert = driver.switch_to_alert()
```

Przykład 3 - lokalizowanie elementów

Lokalizowanie elementu po atrybucie id

Kod HTML

```
<html>
 <body>
  <form id="loginForm">
   <input name="username" type="text" />
   <input name="password" type="password" />
   <input name="continue" type="submit" value="Login" />
  </form>
 </body>
<html>
```

Kod Pythona

```
login_form = driver.find_element_by_id('loginForm')
```

W przypadku braku elementu wystąpi wyjątek **NoSuchElementException**.

Przykład 3 - lokalizowanie elementów c.d.

Lokalizowanie elementu po atrybucie class

```
fb_button = driver.find_element_by_class_name('facebook_button')
```

Lokalizowanie elementu po atrybucie name

```
username = driver.find_element_by_name('username')
```

Lokalizowanie elementu po nazwie tagu

```
continue_btn = driver.find_element_by_name('continue')
```

Lokalizowanie elementu selektorem xpath

```
login_form = driver.find_element_by_xpath("/html/body/form[1]")
login_form = driver.find_element_by_xpath("//form[1]")
login_form = driver.find_element_by_xpath("//form[@id='loginForm']")
```

1) Ścieżka absolutna

2) Pierwszy element w HTML DOM

3) Element, którego atrybut `id` równy jest `loginForm`

Przykład 4 - wypełnianie, wybieranie i klikanie formatek

Lokalizowanie obiektu, wypełnienie obiektu i kliknięcie obiektu

```
passwd_input = driver.find_element_by_id("id_passwd")
element.send_keys("admin1")

driver.find_element_by_id("submit").click()
```

Lokalizowanie obiektu `select`, zawartych w nim obiektów `option` i wypisywanie zawartości ich atrybutu `value`

```
element = driver.find_element_by_xpath("//select[@name='name']")

all_options = element.find_elements_by_tag_name("option")

for option in all_options:
    print "Value is: %s" % option.get_attribute("value")
```

Alternatywny sposób pracy z obiektem `select`

```
from selenium.webdriver.support.ui import Select

select = Select(driver.find_element_by_name("name"))

select.select_by_index(index)
select.select_by_visible_text("text")
select.select_by_value(value)
```

Przykład 5 - ciasteczka

Dodawanie i wypisywanie ciasteczek (dla danego URL)

```
cookie = {"key": "value"})

driver.add_cookie(cookie)

all_cookies = driver.get_cookies()

for cookie_name, cookie_value in all_cookies.items():
    print "%s -> %s" % (cookie_name, cookie_value)
```

Przykład 6 - oczekiwanie

Oczekiwanie na pojawienie się obiektu w DOM (AJAX)

```
# -*- coding: utf-8 -*-

from selenium import webdriver

from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions

from selenium.webdriver.common.by import By

import os

driver = webdriver.Chrome(executable_path="%s/chromedriver" % os.getcwd())
driver.get("http://python.org/")

try:
    element = WebDriverWait(driver, 10).until(
        expected_conditions.presence_of_element_located((
            By.ID,
            "myDynamicElement"
        ))
    )
finally:
    driver.quit()
```

Skrypt w ciągu 10 sekund, co 500 milisekund, przeszukuje DOM pod kątem naszego kryterium.

Selenium + unittest

Test wyszukiwarki dostępnej na pl.python.org

```python
# -*- coding: utf-8 -*-

import os

import unittest

from selenium import webdriver

from selenium.webdriver.common.keys import Keys


class SeleniumTest(unittest.TestCase):
    def setUp(self):
        self.driver = webdriver.Chrome(executable_path="%s/chromedriver" % os.getcwd())

    def test_search_in_pl_python_org(self):
        self.driver.get("http://pl.python.org/")

        self.assertIn("Python", self.driver.title)

        elem = self.driver.find_element_by_id("what")
        elem.send_keys("PyCon PL")
        elem.send_keys(Keys.RETURN)

        self.assertIn("PyCon PL", self.driver.title)

    def tearDown(self):
        self.driver.quit()


if __name__ == "__main__":
    unittest.main()
```

Chcąc wykorzystać potencjał modułu `unittest`, klasa bazowa musi dziedziczyć z `unittest.TestCase`, oraz posiadać metody `setUp` i `tearDown`.

Selenium + JavaScript

Wykonywanie kodu JavaScript (jQuery)

```
height = driver.execute_script("return document.body.scrollHeight")
```

Selenium i zrzuty ekranów

Wykonywanie zrzutu ekranu

```
driver.save_screenshot('screenshot.png')
```

WebDriver API

http://selenium-python.readthedocs.org/api.html

DZIĘKUJĘ