Webbrowser-Automatisierung mit Python und Selenium WebDriver

Andi Albrecht

FrOSCon 2012, Python-Track
http://froscon.org

26. August 2012

Andi Albrecht – @andialbrecht

- Erster Python-Kontakt vor ca. 10 Jahren als Studentische Hilfskraft bei der DFG
- Aktuell: Anwendungsentwickler für Webapplikationen bei ProUnix in Bonn
- Entwicklung und Pflege mittlerer und großer Systeme
- OpenSource: Rietveld Code Review Tool, python-sqlparse, CrunchyFrog, ...
- Mitglied der Python-User-Group pyCologne

Übersicht

Selenium & WebDriver

- **Selenium** – Set an Tools zur Browser-Automatisierung.
- **WebDriver**
  - ist eine API, um Browser anzusteuern
  - ersetzt Selenium RC (seit Selenium 2.0)
  - WebDriver API ist W3C-Spezifikation (Status: Working Draft)
  - HtmlUnit, Firefox, IE, Chrome, Opera, iPhone, Android, ...
- Homepage: `http://seleniumhq.org/`
- Aktuelle Version (21. August 2012): 2.25.0 (Python 2.24.0)

selenium-Python-Modul, Installation

selenium-Python-Modul, Installation

```
pip install selenium
```

selenium-Python-Modul, Installation

```
pip install selenium
```

- Python-Module
- Firefox-Driver
- Treiber für Chrome und andere Browser brauchen noch ein bisschen mehr Setup.

selenium-Python-Modul, Erste Schritte

```
$ python
Python 2.7.1+ (r271:86832, Apr 11 2011, 18:13:53)
>>> from selenium import webdriver
>>> browser = webdriver.Firefox()
>>> browser.get('http://google.com')
>>> browser.find_element_by_tag_name('title')
<selenium.webdriver.remote.webelement.WebElement ...>
```

selenium-Python-Modul, Erste Schritte

```
$ python
Python 2.7.1+ (r271:86832, Apr 11 2011, 18:13:53)
>>> from selenium import webdriver
>>> browser = webdriver.Firefox()
>>> browser.get('http://google.com')
>>> browser.find_element_by_tag_name('title')
<selenium.webdriver.remote.webelement.WebElement ...>
```

selenium-Python-Modul, Erste Schritte

```
$ python
Python 2.7.1+ (r271:86832, Apr 11 2011, 18:13:53)
>>> from selenium import webdriver
>>> browser = webdriver.Firefox()
>>> browser.get('http://google.com')
>>> browser.find_element_by_tag_name('title')
<selenium.webdriver.remote.webelement.WebElement ...>
```

selenium-Python-Modul, Erste Schritte

```
$ python
Python 2.7.1+ (r271:86832, Apr 11 2011, 18:13:53)
>>> from selenium import webdriver
>>> browser = webdriver.Firefox()
>>> browser.get('http://google.com')
>>> browser.find_element_by_tag_name('title')
<selenium.webdriver.remote.webelement.WebElement ...>
```

selenium-Python-Modul, Erste Schritte

```
$ python
Python 2.7.1+ (r271:86832, Apr 11 2011, 18:13:53)
>>> from selenium import webdriver
>>> browser = webdriver.Firefox()
>>> browser.get('http://google.com')
>>> browser.find_element_by_tag_name('title')
<selenium.webdriver.remote.webelement.WebElement ...>
```

API – Browsersteuerung

```
# Browser initialisieren
from selenium import webdriver
browser = webdriver.Firefox()

# URL aufrufen
browser.get('http://example.com')

# Seite neu laden
browser.refresh()

# History
browser.back()
browser.forward()

# Screenshot erstellen
browser.get_screenshot_as_file('/tmp/screenshot.png')

# Browser schließen
browser.quit()
```

API – Elemente finden

```
# Seitentitel
browser.title  # --> 'Example'

# Elemente finden
browser.find_element_by_class_name('clsname')
browser.find_element_by_css_selector('div.myclass p')
browser.find_element_by_id('myid')
browser.find_element_by_link_text('Klick hier')
browser.find_element_by_name('email')
browser.find_element_by_partial_link_text('hier')
browser.find_element_by_tag_name('a')
browser.find_element_by_xpath('//*/body')
browser.find_elements_by_*
```

API – Elemente ansprechen

```
# Beispiel: Formular ausfüllen und absenden
form = browser.find_element_by_id('myform')
input = form.find_element_by_name('email')
input.clear()
input.send_keys('foo@example.com')
input.submit()  # entspricht Enter im Input-Feld

# Beispiel: Link klicken
link = browser.find_element_by_link_text('Anmelden')
link.click()

# Beispiel: Elemente untersuchen
elem = browser.find_element_by_class_name('hidden')
elem.is_displayed() # --> False
browser.find_element_by_link_text('Show hidden').click()
elem.is_displayed() # --> True
elem.location       # --> {u'x': 870, u'y': 50}
elem.size           # --> {'height': 29, 'width': 90}
elem.tag_name       # --> 'div'
elem.text           # --> 'text content of div'
```

API – JavaScript

```
browser.execute_script('alert("Hello World!")')
```

Web-Applikationen testen

```
1   #!/usr/bin/env python
2
3   import unittest
4
5   from selenium import webdriver
6
7
8   class TestTitle(unittest.TestCase):
9
10      def setUp(self):
11          # Toggle comments to test with a different browser
12          self.driver = webdriver.Chrome()
13          #self.driver = webdriver.Firefox()
14          #self.driver = wevdriver.Ie()
15
16      def tearDown(self):
17          self.driver.close()
18
19      def test_title_tag(self):
20          self.driver.get('http://google.com')
21          title_tag = self.driver.find_element_by_tag_name('title')
22          self.assertEqual(title_tag.text, 'Google')
23
24
25  if __name__ == '__main__':
26      unittest.main()
```

Wait!

- Browser-Kommandos warten nicht (click, submit, JavaScript, ...)
- eigene Warteschleifen sind erforderlich (und sinnvoll!)

Wait!

- Browser-Kommandos warten nicht (click, submit, JavaScript, ...)
- eigene Warteschleifen sind erforderlich (und sinnvoll!)

```
from selenium.webdriver.support.wait import WebDriverWait

[...]

elem = browser.find_element_by_id('save')
elem.click()  # sendet POST, lädt die Seite neu
WebDriverWait(browser, 3).until(
    lambda x: x.find_element_by_tag_name('body'))
```

Page-Pattern

```
class LoginPage(object):

    def login_as(self, username, password):
        # some magic
        return HomePage(self.browser)

    def login_expect_exception(self, username, password):
        # ....
        return LoginPage(self.browser)

    def get_exception(self):
        # ....
        return "that happened"
```

Page-Pattern

```
class LoginPage(object):

    def login_as(self, username, password):
        # some magic
        return HomePage(self.browser)

    def login_expect_exception(self, username, password):
        # ....
        return LoginPage(self.browser)

    def get_exception(self):
        # ....
        return "that happened"
```

- The public methods represent the services that the page offers
- Try not to expose the internals of the page
- Generally don't make assertions
- Methods return other PageObjects
- Need not represent an entire page
- Different results for the same action are modelled as different methods

Quelle: http://code.google.com/p/selenium/wiki/PageObjects

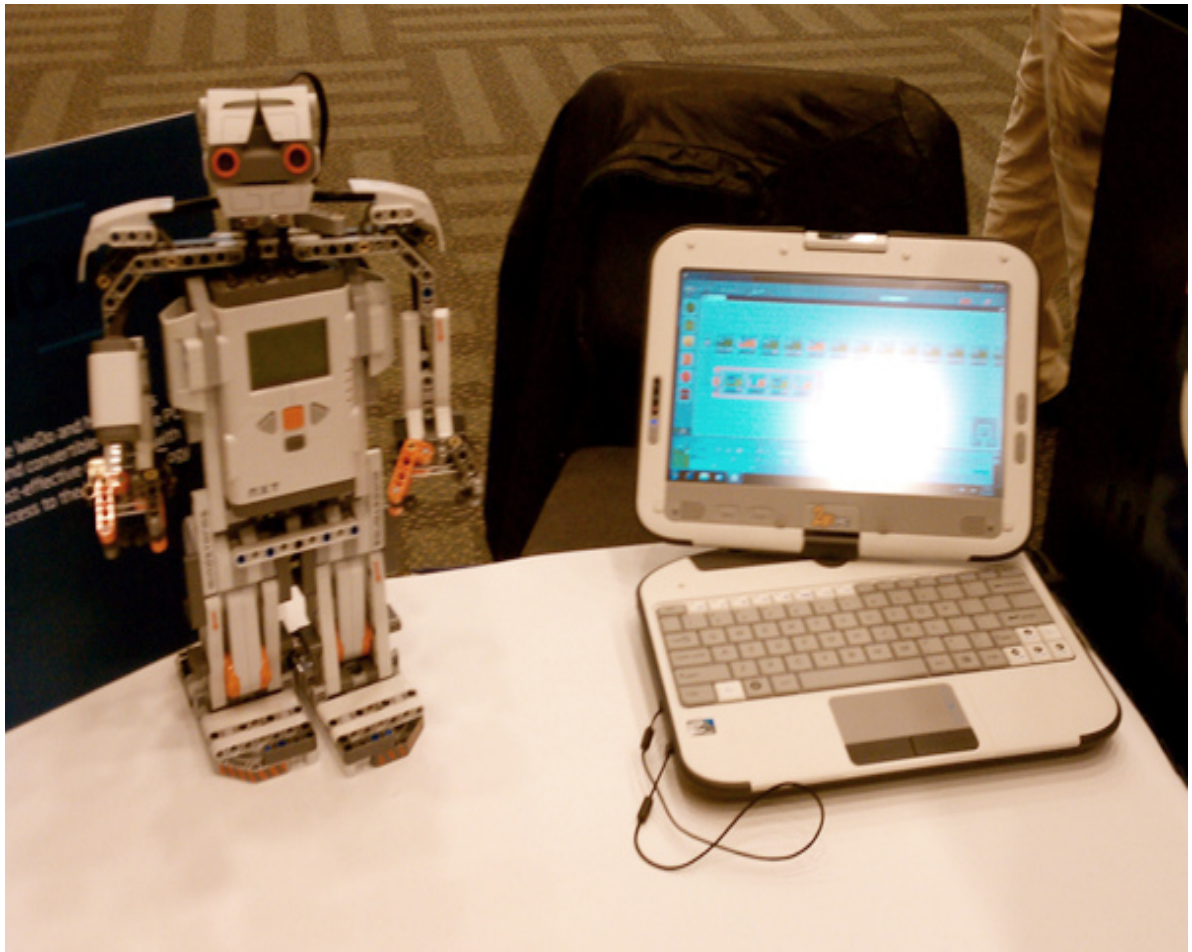
`sst` – Selenium Simple Test

- Wrapper um `selenium`-Modul
- Einfache API, für Testentwickler gedacht¡
- Parametrisierbare Tests
- Screenshots bei Fehlern
- Headless-Mode out-of-the-box (benötigt xvfb)
- Homepage: `http://testutils.org/sst/`

sst – Selenium Simple Test

```
1   from sst.actions import *
2
3   go_to('http://froscon.org')
4   assert_title_contains('FrOSCon')
5   click_link(get_element(tag='a', text='Program'))
6   click_link(get_element(tag='a', text='program of lectures'))
7   assert_title_contains('froscon2012: FrOSCon Program')
8   click_link(get_element(tag='a', text_regex='Day 2.*26$'))
9   click_link(get_element(tag='a', text='>'))
10  click_link(get_element(tag='a', text_regex='.*Selenium.*'))
11  click_link(get_element(tag='a', text='Click here'))
12  set_radio_value('event_feedback_rating_2')
13  write_textfield(
14      'event_feedback_comment',
15      'Yay! I\'ve learned to automate feedback for my next FrOSCon talk!')
16  take_screenshot('/home/andi/Arbeitsfläche/feedback.png')
17  raw_input('Done.')
```

Demo, Fragen?



Quelle: Classmates PC w Programmable Robot, by chang_sen, on Flickr

http://www.flickr.com/photos/senchang/5000312557/

cologne

Danke!

Links

Python-Modul
: `http://pypi.python.org/pypi/selenium/`

Selenium-Homepage
: `http://seleniumhq.org/`

pyCologne
: `http://pycologne.de`, Treffen jeden 2. Mittwoch

Kontakt

E-Mail `albrecht.andi@gmail.com`

Twitter `@andialbrecht`

Homepage `andialbrecht.de`