Selenium

Testování webových aplikací

Michal Hořejšek, vedoucí týmu vývoje, michal.horejsek@firma.seznam.cz

SEZNAM.CZ

Klasické jednotkové a integrační testy

```
def test_list_clients_contracts(api, client, contract):
    res = api.client.list_contracts(client.id)
    assert len(res) == 1
    assert res[0] == contract
```

Testování response objektu

```
def test_client_persons_suggest(test_app, client):
    data = {
        'query': client.surname,
    }
    response = test_app.json('/klient/vyhledavani', data)

    assert response.status_code == 200
    assert response[‘clients'][0] == client
```

Jasmínové testy

```
describe 'Restaurant', () ->
    restaurant = new models.Restaurant
        name: 'Delicious dream'
        lastUpdate: '2013-01-01 18:30'

    describe '#getPrintalbeLastUpdate()', () ->
        it 'should return date as pretty string', () ->
            res = restaurant.getPrintalbeLastUpdate()
            assert.equal res, '1. ledna v 18:30'
```

** Ve skutešnosti to je test naší webovky, podle které chodíme na oběd. Když budete v Praze kolem Anděla, zkuste lunchtimeandel.cz ;-)*

Jak ale udělat integrační test webové aplikace?

Selenium

„Selenium automates browsers. That's it!“

Jednoduše z kódu lze ovládat prohlížeč a testovat tak webovou aplikaci jako celek.

Instalace Selenia

Velice jednoduché přes PyPI:

```
# pip install selenium
```

Ukázka Selenia

```
from selenium import webdriver

driver = webdriver.Chrome()

driver.get('http://www.seznam.cz')

print driver.title

driver.quit()
```

Virtuální Xka

- Možnost spouštět na serveru bez Xek
- Dobré i na osobním počítači – běží na pozadí

- Existuje několik backendů: Xvfb, Xephyr, Xvnc
- Xvfb je X server
- Xvnc je X server s displejem s možností vzdáleného připojení

Instalace Xek

Nejprve backend, například Xvfb:

```
# apt-get install xvfb
```

Poté API pro Python:

```
# pip install pyvirtualdisplay
```

Ukázka s virtuálním displayem

```
from pyvirtualdisplay import Display

display = Display(size=(800, 400))
display.start()

# ...

display.stop()
```

Prohlížeč na Debian serveru

- **Tři možnosti – Iceweasel, Firefox, Chrome**
- **Nejlepší zkušenosti s Chrome, funguje nejspolehlivěji**

Selenium server

- **Možnost více prohlížečů**
- **Možnost více operačních systémů**
- **Možnost rozložit zátěž na více strojů**

Selenium server

```
url = 'http://127.0.0.1:4444/wd/hub‘
desire = webdriver.DesiredCapabilities.FIREFOX.copy()

driver = webdriver.Remote(
    command_executor=url,
    desired_capabilities=desire,
)
```

Selenium – co nabízí

- Procházení stránek
- Vyhledávání elementů
- Zkoumání elementů
- Klikání
- Práce s okny a alerty
- Práce s cookies
- Screenshoty

- Nic víc

Selenium – co neumí

- Neposkytuje informace o requestu
- Nelze jednoduše vyplňovat formuláře
- Do výjimek nedává rozumné hlášky
- Neumí vyhledávat text
- Neobsahuje nic pro testy

webdriverwrapper

```
# pip install webdriverwrapper
```

Řeší právě zmíněné nedostatky…

- **Formuláře**
- **Vyjimky**
- **Stahování souborů**
- **Testy**
- **Zkratky**

Formuláře – klasicky

```
from selenium.webdriver.common.keys import Keys

input = driver.find_element_by_name('q')
input.send_keys('selenium‘ + Keys.TAB)

submit = driver.find_element_by_id('hledej‘)
submit.click()
```

Formuláře – s wrapperem

```
driver.get_elm('inet-f').fill_out_and_submit({
    'q': 'selenium',
})
```

- **Není třeba řešit rozdíly mezi formuářovými prvky**
- **Lze posílat Pythoní typy**
- **Po vyplnění zruší focus pro onchange event**
- **Zvládne i „moderní“ skryté checkboxy**

Formuláře – select

```
elm = driver.find_element_by_id('select')
elm = webdriver.support.select.Select(elm)
elm.deselct_all()
```

S wrapperem je select automaticky `Select`.

Smysluplné výjimky

```
>>> driver.go_to('http://www.seznam.cz‘)
>>> driver.get_elm('resultCount‘)
NoSuchElementException: Message: u'No element <*
id=resultCount> found at http://www.seznam.cz/’
```

- **Hned je vidět, zda se vůbec test dostal na správnou stránku a nevyhučelo to někde dříve**
- **Je vidět, co se hledalo, pokud se někde vyhledává dynamicky (třeba formuláře)**

Stažení souboru

```
f = driver.get_elm('link').download_file()
f.method
f.status_code
f.headers
f.data
```

- **Zvládne odkazy i formuláře**
- **Vytváří request s cookies**
- **Možnost ověřit si hlavičky, status kódy, co se Seleniem normálně nelze**
- **Podobně metoda** `download_url(url=None)`

Info o síti

- **Neexistuje a nikdy nebude**
- **Možné řešení pomocí proxy – v testu si z ní tahat potřebné informace**

Práce s okny, taby

```
driver.switch_to_window(window_name)
```

- **Selenium umožňuje pouze podle názvu (v JS)**
- **Ve wrapperu je navíc možnost podle** `title` **a** `url`

Práce s okny, taby

```
driver.close()
```

- **Selenium umožňuje zavřít pouze aktuální okno**
- **Ve wrapperu je navíc metoda** `close_window`
- **Případně také** `close_other_windows`

```
driver.close_window(window_name|title|url)
```

Práce s okny, taby

- **Alert je blokující stejně jako pro uživatele**
- **Musí se do wrapperu přepnout a *odklepnout***

```
from selenium.webdriver.common.alert import Alert
Alert(driver).accept()
```

- **Wrapper má zkratku** `get_alert`

```
driver.get_alert().accept()
```

Zkratky – go_to

- **Parametry pro cestu, doménu a query**
- **Doména (či absolutní adresa) se musí určit pouze na začátku**
- **Možnost poslat slovník query se hodí především pro testy či přehlednost**

```
driver.go_to(domain='seznam.cz')
driver.go_to('search')  # Nemusim psat znovu seznam.cz
driver.go_to(query={'q': 'param'})
```

Zkratky – `get_elm(s)`

- **Především nemusím psát časté a dlouhé** `find_element_by_id`
- **Podporuje parent element, takže stačí napsat jednou**

```
elm = driver.find_element_by_id('id')
elm = elm.find_elements_by_tag_name(‘a')
```

vs.

```
elm = driver.get_elms(tag_name=‘a', parent_id='id')
```

StaleElementException

- **Instance elementu se po změně DOMu nezmění. Zůstává reference na starý element.**

```
elm = driver.get_elm('q')
elm.send_keys('aa')
time.sleep(5)  # JS smaze nahradi element za novy.
elm.send_keys('bb')  # Exception
```

Zkratky – `click`

- **Funguje stejně jako `get_elm`, navíc však provede `click`**
- **Pokud takových elementů je více, klikne na ten první**

```
elm = driver.find_element_by_id('id')
elm = elm.find_element_by_tag_name(‘a')
elm.click()
```

vs.

```
elm = driver.click(tag_name=‘a', parent_id='id')
```

Zkratky – `contains_text`

- **Selenium vůbec nepodporuje hledání textu**
- **Implementováno pomocí XPath**
- **Možnost vyřadit některé elementy ze stránky atributem `data-selenium-not-search` (typycky debug)**
- **Samozřejmě tu je i `find_element_by_text`**

XPath

- **Velice mocný nástroj**
- **Info na http://www.w3schools.com/xpath/xpath_intro.asp**
- **V Chrome konzoli přes `$x(...)`**
- **Používat však výjimečně na složité cesty (pomalejší)**

```
//select[@name="country.id"]/option[@value=“cr"]
```

```
//div[contains(text(), "xxx")]
```

Implicitní a explicitní wait

- **Při kliknutí Selenium implicitně čeká, až se provede request**
- **Funguje to však pouze u klasických odkazů, nikoliv u elementů obsluhující JavaScript**
- **U JS se musí případné čekání provést ručně**

```
from selenium.webdriver.support.wait import WebDriverWait
WebDriverWait(driver, timeout=10).until(callback)

def callback(driver):
    return driver.find_element_by_id('id')
```

Wait s wrapperem

- **Čekání na konkrétní element je velmi časté a proto na to je speciální metoda**

```
driver.wait_for_element(id_='id')
```

- **Pokud je potřeba něco specifičtějšího, metoda** `wait` **je zkratka pro** `WebDriverWait`

```
driver.wait().until(callback)
```

Implicitní wait

- Lze nastavit implicitní wait i pro metody find_element*
- Znamená to však pomalejší testy
- Čeká se i tam, kde se čekat nemusí (například při smazání elementu)

Testování se Seleniem

- `WebdriverTestCase`
- **Vytváří driver**
- **Samo ověřuje chyby (chybové stránky a zprávy)**
- **Samo dokáže ověřit JS chyby**
- **Zaručuje běh v hlavním okně (přepnutí z popupů)**
- **Možný debug (čekání po testech)**
- **Zkratky**
- **Užitečné dekorátory**

WebdriverTestCase

```
class TestCase(WebdriverTestCase):
    domain = 'https://www.google.com'
    instances_of_driver = ONE_INSTANCE_PER_TESTCASE

    def _get_driver(self):
        return Chrome()

    def testDoodle(self):
        self.click('gbqfsb')
        self.contains_text('Doodles‘)
```

Dekorátory

- **Pro jednodušší práci užitečné dekorátory**
- **Některé testy pak stačí passnout – dekorátory zařídí vše potřebné**

```
@GoToPage(‘http://www.google.com')
@ShouldBeOnPage('doodles/AllDoodles')
def testDoodle(self):
    self.click('gbqfsb')
    self.contains_text('Doodles')
```

Dekorátory – zpracování chyb

- **Po každém testu se zkoumá stránka, zda neobsahuje neočekávané chyby**
- **Stránku 4xx, 5xx vyhledává podle CSS třídy** `error-page` **a v něm nadpis tag** `h1`**.**
- **Chybové hlášky jsou hledány podle CSS třídy** `error` **(atribut** `error`**).**

Dekorátory – zpracování chyb

```
@utils.suClient
@utils.ShouldBeErrorPage(403)
@utils.GoToPage(‘/uzivatel/admin')
def testClientHasNotPermission(self):
    pass

@utils.ShouldBeError('exists')
def testAlreadyExists(self):
    self.get_elm('userUsername_form').fill_out_and_submit({
        'item.username': 'admin',
    })
```

Zpracování chyb

Hledání chyb v JavaScriptu automaticky s tímto snipetem:

```
<script type="text/javascript">
    window.jsErrors = [];
    window.onerror = function(errorMessage) {
        window.jsErrors[window.jsErrors.length] =
errorMessage;
    }
</script>
```

Zpracování chyb

- **Ověřování chyb je pouze na konci testu**
- **Není možné automaticky testovat na všech stránkách**
- **Také to nemusí být žádoucí**
- **Proto metoda** `check_errors`

Další dekorátory

`CanBeError`

- **Funguje jako** `ShouldBeError`
- **Vhodné pro testy, kde může ale i nemusí nastat chyba**

`ShouldBeInfo`

- **Stejné jako u** `ShoulBeError`
- **Vyhledává však CSS třídy** `info` **(atribut** `info`**)**
- **Vhodné pro testování „Podařilo se úspěšně uložit“**

Debugování

- **Zastavit po každém testu lze třídní proměnnou** `wait_after_test`
- **Zastavit na konkrétním místě lze pomocí metody** `break_point`
- **Zalogovat (pomocí logging) lze pomocí metody** `debug`

Vždy aktivní hlavní okno

- Na konci každého testu se vrátí do hlavního okna
- Není potřeba ručně řešit s try-finally

Zkratky

TestCase obsahuje zkratky na driver:

```
def testMethod(self):
    self.click(...)
    self.get_elm(...)
    self.contains_text(...)
    self.wait_for_element(...)
    self.switch_to_window(...)
    ...
```

Co se plánuje ve wrapperu

- Dekorátor na určování v kterém prohlížeči vyzkoušet (podpora Selenia gridu)
- Podpora pytestu
- Důmyslnější Fuzzy
- Refaktoring pro snažší přetěžování

Page Objects

- Rozhraní mezi stránkou a testovacím kódem
- Představuje seznam akcí na stránce, které uživatel může provést
- Netahá do testů kód stránky (v testu se nesahá na API Selenia)

K čemu to je dobré?

- Zvýší přehlednost a zjednoduší kód testu
- Rychlejší úprava testů
- Omezení duplicitního kódu
- Každá funkce má svoje jedno místo

Ukázka – page object

```
class Homepage(object):
    def __init__(self, driver, location):
        self.driver = driver
        self.driver.get(location)

    def is_loaded(self):
        return self.driver.title == u'Seznam - Najdu tam, co neznám'

    def search_for(self, search_term):
        self.driver.get_elm('inet-f').fill_out_and_submit({
            'q': search_term,
        })

    def get_title(self):
        return self.driver.title
```

Ukázka – využití v testu

```
class TestSearch(unittest.TestCase):
    def setUp(self):
        self.driver = webdriver.Chrome()
        self.hp = Homepage(self.driver, "http://www.seznam.cz")

    def test_search(self):
        self.assertTrue(self.hp.is_loaded())
        self.hp.search_for("auto")
        self.assertTrue("auto" in self.hp.get_title())

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

Rozsáhlost takového objektu

- **Jedna stránka nebo její část (stránku pak komponovat)**
- **Příliš velké objekty jsou nepřehledné => lepší rozdělit na menší celky**

Více info o Page Objects

- https://wiki.mozilla.org/QA/Execution/Web_Testing/Docs/Automation/StyleGuide

Děkuji za pozornost

Michal Hořejšek, vedoucí týmu vývoje, michal.horejsek@firma.seznam.cz