Selenium - passos iniciais

Jailma Januário da Silva

Leonardo Massayuki Takuno

Resumo

Os objetivos desta parte são: (I) Entender sobre testes em navegadores web. (II) Aprender a instalar o arcabouço Selenium. (III) Aprender o que é um driver de navegador. (IV) Realizar testes com navegadores usando Selenium Webdriver.

Introdução

O Selenium é um arcabouço bastante utilizado para testes de software direcionados para os navegadores web mais utilizados, como o Google Chrome, Mozilla Firefox e Internet Explorer. O projeto Selenium disponibiliza três ferramentas em sua página inicial, as quais denominam-se Selenium Webdriver, Selenium IDE e Selenium Grid. Cada ferramenta tem um propósito específico, características e vantagens e desvantagens em sua utilização. Este texto apresenta o Selenium Webdriver, utilizando a linguagem Python para a automação de testes.

A escolha do Selenium é motivada por ser uma ferramenta que auxilia na criação de scripts de testes automatizados e, também, é uma ferramenta bastante conhecida na indústria. Além disso, é uma ferramenta open source, e por esse motivo, sempre recebe atualizações para as principais linguagens de programação e navegadores do mercado.

Configuração do Selenium Webdriver

Segundo a SELENIUM (2021), o suporte da automação de testes para navegadores é feito por meio do *Webdriver*, o qual é uma API e protocolo que define uma linguagem para controlar o comportamento dos navegadores web. Cada navegador necessita de uma implementação específica do *Webdriver*, a esse componente dá-se o nome de *driver*. O driver, portanto, é o componente responsável por facilitar a comunicação entre o Selenium e o navegador.

Para configurar o Selenium para executar de maneira integrada com a linguagem Python siga as seguintes etapas:

1. Instale a biblioteca Selenium para o Python através do comando:

   pip install selenium

2. Instale o webdriver manager que permite a utilização de drivers de diferentes navegadores.

   pip install webdriver-manager

Primeiros scripts com Selenium webdriver

Uma vez configurado o Selenium webdriver, e o webdriver-manager, está tudo pronto para criar o primeiro script. Observe o arquivo test_selenium_chrome.py conforme a Codificação 7.1.

O primeiro passo é realizar a importação da biblioteca selenium e importar o objeto webdriver. Em seguida, o script inicia uma sessão no navegador, que neste exemplo é o Google Chrome, e isso significa que o browser será iniciado e será aberto. A última instrução informa qual é o endereço URL para o qual o navegador deve realizar a requisição. Como resultado, este script inicia o navegador na página indicada pela URL 'http://google.com.br'.

Codificação 7.1. test_selenium_chrome.py

```
from selenium import webdriver
from webdriver_manager.chrome import ChromeDriverManager
```

```
from selenium.webdriver.chrome.service import Service

servico = Service(ChromeDriverManager().install())
navegador = webdriver.Chrome(service=servico)
navegador.get("https://google.com")

input()
```

Fonte: do autor, 2022.

De maneira semelhante, observe a Codificação 7.2 para iniciar uma sessão com o navegador Firefox.

Codificação 7.2. test_selenium_firefox.py

```
from selenium import webdriver
from webdriver_manager.firefox import GeckoDriverManager
from selenium.webdriver.firefox.service import Service as FirefoxService

navegador = webdriver.Firefox(service=FirefoxService(GeckoDriverManager().install()))
navegador.get('http://google.com.br')
```

Fonte: do autor, 2022.

Manipulando o navegador

Como se pode perceber, pelos exemplos anteriores, o selenium webdriver controla o navegador através de comandos de scripts. Esta seção apresenta alguns recursos que podem ser utilizados para coleta de informações sobre o título da página, bem como poder obter a URL atual ou até mesmo obter o código fonte da página atual. Além disso, é possível utilizar instruções que controlam o comportamento do browser, tal como maximizar a janela do navegador, encerrar a janela atual, ou até mesmo, encerrar todas as janelas. Observe a Codificação 7.3 para identificar alguns elementos de manipulação do navegador.

Codificação 7.3. test_selenium_firefox.py

```
from selenium import webdriver
```

```
from webdriver_manager.firefox import GeckoDriverManager
from selenium.webdriver.firefox.service import Service as FirefoxService

navegador = webdriver.Firefox(service=FirefoxService(GeckoDriverManager().install()))
navegador.get('http://google.com.br')
navegador.maximize_window()
print('título:', navegador.title)
print('endereço URL:', navegador.current_url)
print('código fonte:', navegador.page_source[:100])
navegador.close()
```

Fonte: do autor, 2022.

De acordo com a Codificação 7.3, o script executa as seguintes operações:

- Iniciar o navegador Firefox;
- Acessar a página do Google pela URL 'http://google.com.br';
- Maximizar a tela;
- Imprimir o título da página;
- Imprimir o endereço URL;
- Imprimir uma substring (com 100 caracteres) do código fonte da página atual;
- Encerrar a janela do navegador.

Ao executar o código, perceba que o Firefox será iniciado pelo selenium webdriver, conforme apresenta a Figura 7.1.

Figura 7.1. Navegador Firefox iniciado pelo selenium webdriver

Fonte: do autor, 2022.

Após a abertura, o navegador maximiza por um certo tempo e finalmente encerra-se. Ao final da execução pode-se perceber as informações que foram impressas no console, como apresenta a Figura 7.2.

Figura 7.2. Resultado da execução do script de Codificação 7.3

Fonte: do autor, 2022.

Interação com os elementos de web

Os testes para um sistema web consistem em realizar interações com os elementos da interface que estão na página HTML, os quais incluem botão, caixa de textos, listas, tabelas entre outros. Para páginas pequenas e simples é bem fácil identificar os elementos de web, no entanto, para páginas de sistemas reais, o código HTML gerado é, na maioria dos casos, bastante complexo (PEIXOTO, 2018).

Para identificar detalhes dos elementos, muitas vezes, é preciso inspecioná-los. Isto é importante para que se possa informar ao Selenium Webdriver qual é o componente alvo da interação, e qual será a ação que aquele componente deve executar durante os testes de unidade. Abra o seu navegador, neste exemplo usou-se o Firefox, no endereço do Google ('http://www.google.com.br'), então selecione o campo de busca da página e clique com o botão direito. Após isso, escolha a opção 'Inspecionar' (ou *Inspect*, em inglês), conforme apresenta a Figura 7.3.

Figura 7.3. Inspecionar o campo de busca da página do Google

Fonte: do autor, 2022.

Observe que o navegador apresentou o código-fonte da página, e o campo de busca que foi selecionado é marcado em destaque, conforme apresenta a Figura 7.4. Perceba que o elemento em destaque é do tipo **<input>** com alguns atributos, dentre os quais destaca-se o atributo name, pois este é um dos principais atributos utilizados pelo Selenium para localizar um determinado elemento.

Figura 7.4. Código-fonte do elemento inspecionado

Fonte: do autor, 2022.

Segundo PEIXOTO (2018), o Selenium contém meios de identificar elementos na página, os quais são denominados de localizadores (do inglês, *locators*). Esses localizadores são passados como parâmetro para os métodos utilizados para encontrar elementos por meio dos atributos:

- id;
- name;
- classname;
- css;
- link text;
- partial link text;
- xpath;
- tagname.

Observe que pela Figura 7.4 o campo de busca que é um elemento do tipo **<input>**, possui o atributo **name** com valor igual a **'q'**, que será utilizada no exemplo apresentado na Codificação 7.4. A instrução **find_element** é um método do webdriver que devolve um objeto do tipo WebElement, o qual é atribuído à variável **campo_de_busca**. Ao final, a propriedade **tag_name** do

campo de busca é impresso para console, que apresenta como resultado a palavra 'textarea'.

Codificação 7.4. test_google_firefox.py

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.firefox.service import Service as FirefoxService
from webdriver_manager.firefox import GeckoDriverManager

navegador = webdriver.Firefox(service=FirefoxService(GeckoDriverManager().install()))
navegador.get('http://google.com.br')
navegador.maximize_window()
campo_de_busca = navegador.find_element(By.NAME, 'q')
print(campo_de_busca.tag_name)  # textarea
```

Fonte: do autor, 2022.

Agora, para escrever um texto no campo de busca utilize o método **send_keys()** do driver passando como parâmetro a string que se deseja realizar a busca, e para realizar a busca, utilize o método **submit()**, observe a Codificação 7.5.

Codificação 7.5. test_google_firefox.py

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.firefox.service import Service as FirefoxService
from webdriver_manager.firefox import GeckoDriverManager

navegador = webdriver.Firefox(service=FirefoxService(GeckoDriverManager().install()))
navegador.get('http://google.com.br')
navegador.maximize_window()
campo_de_busca = navegador.find_element(By.NAME, 'q')
```

```
campo_de_busca.send_keys('Selenium')
campo_de_busca.submit()
```

Fonte: do autor, 2022.

A execução do programa consiste em abrir o navegador Firefox, navegar até a página do Google no link 'http://google.com.br', em seguida selecionar o campo de busca e digitar o texto **Selenium** e, por fim, submeter a consulta ao servidor, conforme apresenta a Figura 7.5 e Figura 7.6.

Figura 7.5. Escrevendo no campo de busca do Google

Fonte: do autor, 2022.

Figura 7.6. Resultado da busca da palavra Selenium do Google

Fonte: do autor, 2022.

Para localizar elementos web

Como se pode perceber até o momento, para interagir com um componente em uma página, deve-se primeiramente encontrá-lo. O selenium utiliza os chamados localizadores para buscar esses elementos web. Como mencionado anteriormente, existem localizadores de elementos web, os quais estão detalhados de acordo com a Tabela 7.2.

Tabela 7.2. Localizadores

Fonte: do autor, 2022.

Ações com mouse e teclados

Em testes de aplicações web, é importante simular ações do usuário utilizando entrada e saída por meio de ações de mouse e teclado. Esta seção apresenta algumas dessas ações.

Mouse

O mouse pode executar operações e ações tais como click, drag, mover entre outras. Estas ações são aplicadas sobre os elementos web. Por exemplo, é possível criar um script para obter a localização de um link, e acionar esse link por meio do click do mouse.

Para ilustrar essa situação, observe a Codificação 7.6, que é um arquivo html que renderiza uma página web contendo um link para a página de busca do Google. Em seguida, crie o arquivo test_exemplo_link.py de acordo com a Codificação 7.7.

Codificação 7.6. exemplo_click.html

```
<html>
<head>
   <title>Titulo</title>
</head>
<body>
   <a href="http://www.google.com.br"> Google </a>
</body>

</html>
```

Fonte: do autor, 2022.

Codificação 7.7. test_exemplo_click.py

```
import os
from selenium.webdriver import Chrome
from selenium.webdriver.common.by import By
from webdriver_manager.chrome import ChromeDriverManager
```

```
from selenium.webdriver.chrome.service import Service as ChromeService

chrome = Chrome(service=ChromeService(ChromeDriverManager().install()))

local_path = f'file:///{os.path.dirname(os.path.realpath(__file__))}/'
chrome.get(f'{local_path}exemplo_click.html')

ancora = chrome.find_element(By.LINK_TEXT, 'Google')
ancora.click()
```

Fonte: do autor, 2022.

A Codificação 7.7 inicialmente importa a biblioteca os, para obter informações do arquivo html local, e também, classes do arcabouço Selenium para interação com o browser e localização dos elementos web. Após isso, o script utiliza o navegador Google Chrome para realizar a abertura do arquivo exemplo_click.html. Em seguida, o script localiza o link por meio do texto 'Google', e atribui o elemento web à variável ancora, que em seguida aciona o método click(), simulando o click do mouse.

Considere outro exemplo, suponha que o seu script deve localizar um botão e posicionar o mouse e acionar o botão direito do mouse. Isto pode ser feito com a função context_click(web_element). Para isso é preciso utilizar o objeto da classe ActionChain que automatiza interações de baixo nível de mouse e teclado. Para isso, observe a Codificação 7.8, que renderiza um botão, e ao ser acionado, leva à página web do youtube.

Em seguida, observe a Codificação 7.9, que contém o código Python de exemplo para realizar a ação de mover o mouse em cima do botão e acionar o botão direito do mouse, apresentando o menu suspenso.

Codificação 7.8. ex_botao_para_youtube.html

```
<html>
```

```
<head><title>Titulo</title></head>
<body>
   <form method="get" action="https://www.youtube.com/">
      <button type="submit">youtube</button>
   </form>

</body>

</html>
```

Fonte: do autor, 2022.

Codificação 7.9. test_exemplo_clique_direito.py

```
import os
from selenium import webdriver
from selenium.webdriver.common.by import By
from webdriver_manager.chrome import ChromeDriverManager
from selenium.webdriver.chrome.service import Service as ChromeService

chrome = webdriver.Chrome(service=ChromeService(ChromeDriverManager().install()))
local_path = f'file://{os.path.dirname(os.path.realpath(__file__))}/'
chrome.get(f'{local_path}ex_botao_para_youtube.html')

botao = chrome.find_element(By.TAG_NAME, 'button')
webdriver.ActionChains(chrome).context_click(botao).perform()
```

Fonte: do autor, 2022.

Observe a Codificação 7.9, que foi instanciado um objeto **ActionChains**, passando para o construtor o driver do Google Chrome, e executou-se o método **context_click** sobre o elemento web **botao**, localizado na linha anterior pela tag **button**.

Outra ação pertinente para os testes em página web com interação com o mouse é a ação de duplo clique, que é executada pelo método **double_click**() da classe **ActionChains**. Observe a Codificação 7.10 para observar esta situação.

Codificação 7.10. test_exemplo_duplo_clique.py

```
import os
from selenium import webdriver
from selenium.webdriver.common.by import By
from webdriver_manager.chrome import ChromeDriverManager
from selenium.webdriver.chrome.service import Service as ChromeService

chrome = webdriver.Chrome(service=ChromeService(ChromeDriverManager().install()))

local_path = f'file:///{os.path.dirname(os.path.realpath(__file__))}/'
chrome.get(f'{local_path}ex_botao_para_youtube.html')

botao = chrome.find_element(By.TAG_NAME, 'button')
webdriver.ActionChains(chrome).double_click(botao).perform()
```

Fonte: do autor, 2022.

Testes com Selenium

Esta seção apresenta um exemplo de testes de unidade utilizando o Selenium e o Pytest. Assim como as partes anteriores, os exemplos serão construídos de maneira organizada em pastas com a intenção de caminhar para um projeto de sistema. Observe na Figura 7.7, a estrutura de pastas e arquivos para este exemplo.

Figura 7.7. Teste com selenium

Fonte: do autor, 2022.

Para o arquivo aluno.html, observe a Codificação 7.12, que contém um formulário com campos para o nome do aluno e e-mail, e também, um botão de login.

Codificação 7.12. aluno.html

```
<html>
<title>Cadastro de aluno</title>
<body>
<form id="FormAluno">
<input name="nome_aluno" placeholder="nome do aluno" type="text"/><br/>
<input name="email" placeholder="e-mail" type="email"/><br/>
<input name="login" type="submit" value="Login"/>
</form>
</body>

</html>
```

Fonte: do autor, 2022.

Para o arquivo conftest.py, observe a Codificação 7.13, que contém uma fixture que inicia o navegador Google Chrome. E após isso, a função acessa o arquivo aluno.html que se encontra no path local do projeto. A instrução **yield** permite que a fixture seja utilizada em um determinado teste, e após o uso, o navegador é encerrado pelo método **close()**. É importante enfatizar que o nome da fixture é **form_aluno**, pois, ele será utilizado apenas para os testes de navegação dos elementos web do arquivo aluno.html.

Codificação 7.13. conftest.py

```
# arquivo conftest.py
import os
from pytest import fixture
```

```
from selenium import webdriver

from webdriver_manager.chrome import ChromeDriverManager
from selenium.webdriver.chrome.service import Service as ChromeService

@fixture
def form_aluno():
    chrome = webdriver.Chrome(service=ChromeService(ChromeDriverManager().install()))
    pasta = os.path.dirname(os.path.realpath(__file__))
    local_path = f'file://{pasta}/../app/template/'
    chrome.get(f'{local_path}aluno.html')
    yield chrome
    chrome.close()
```

Fonte: do autor, 2022.

Para o arquivo test_form_aluno.py, de acordo com a Codificação 7.14, observe a primeira função **test_localizar_campo_nome**, que recebe uma fixture **form_aluno** como parâmetro. Após isso, foi possível localizar o campo nome pelo localizador **NAME** com valor 'nome_aluno', o qual foi obtido por meio do método **find_element()**. Observou-se que o **tag_name** é um componente do tipo input, e possui atributos **placeholder** igual a 'nome do aluno' e **type** igual a 'text', que foram obtidos pelo método **get_attribute()**. As funções para testar o campo de e-mail e o botão login acontecem de maneira semelhante. A função **test_localizar_form_aluno** utiliza o localizador ID com valor **'FormAluno'**, e verifica que o **tag_name** é do tipo form. E por fim, a função **test_verificar_titulo_do_form** apenas verifica que o valor da propriedade title é igual a 'Cadastro de aluno'.

Codificação 7.14. test_form_aluno.py

```
# arquivo test_form_aluno.py
```

```
from selenium.webdriver.common.by import By

def test_localizar_campo_nome(form_aluno):
    campo_nome = form_aluno.find_element(By.NAME, 'nome_aluno')
    assert campo_nome.tag_name == 'input'
    assert campo_nome.get_attribute('placeholder') == 'nome do aluno'
    assert campo_nome.get_attribute('type') == 'text'

def test_localizar_campo_email(form_aluno):
    campo_email = form_aluno.find_element(By.NAME, 'email')
    assert campo_email.tag_name == 'input'
    assert campo_email.get_attribute('placeholder') == 'e-mail'
    assert campo_email.get_attribute('type') == 'email'

def test_localizar_campo_login(form_aluno):
    campo_login = form_aluno.find_element(By.NAME, 'login')
    assert campo_login.tag_name == 'input'
    assert campo_login.get_attribute('type') == 'submit'
    assert campo_login.get_attribute('value') == 'Login'

def test_localizar_form_aluno(form_aluno):
    form_aluno = form_aluno.find_element(By.ID, 'FormAluno')
    assert form_aluno.tag_name == 'form'

def test_verificar_titulo_do_form(form_aluno):
    assert form_aluno.title == 'Cadastro de aluno'
```

Fonte: do autor, 2022.

Para o arquivo pytest.ini observe a Figura 7.8, este é um arquivo importante para que o pytest consiga encontrar os arquivos de testes dentro do projeto.

Figura 7.8. arquivo pytest.ini

Fonte: do autor, 2022.

Considerações finais

Ferramentas de testes para web têm um papel importante na garantia de qualidade de software. Como se pode perceber, o selenium tem uma linguagem simples e que permite realizar testes em todos os navegadores mais relevantes da indústria.

Esta parte apresentou de forma muito introdutória o selenium webdriver, e tratou de ações de obtenção de informações sobre os navegadores, tais como obter o título, obter a URL, e até maximizar a janela do navegador, e ainda, mostrou os conceitos utilizados para realizar a navegação de elementos web por meio de localizadores.

Além disso, apresentou-se um exemplo de como combinar o arcabouço pytest com o selenium para realizar testes sobre os elementos web de uma página estática. A próxima parte, está planejada para aprofundar os conhecimentos de como interagir ainda mais com os elementos web por meio do selenium. Ainda há muito o que aprender.

Vamos praticar?

1) Para este exercício, crie um arquivo exercicio01.html conforme o código a seguir:

```
<html>
<title>Exercício 01</title>
<body>
<p class="content">O conteúdo do site vem aqui</p>
</form>
</body>
```

</html>

exercicio01.html

Pede-se:

a) Escreva um script que leia o arquivo local **exercicio01.html**, mostre o título da página.
b) Após finalizar o item a), mostre o conteúdo do parágrafo, utilizando o localizador **TAG_NAME**.
c) Após finalizar o item b), mostre o conteúdo do parágrafo, utilizando o localizador **CSS_SELECTOR** com o seletor '**p.content**'.
d) Criar um arquivo chamado test_exercicio01.py que realiza o teste de unidade dos elementos web do arquivo local **exercicio01.html.**

Referências

PYTEST. **Documentação versão de python 3.7+**. 2015. Disponível em: <https://docs.pytest.org/en/7.1.x/index.html>. Acesso em: 11 jun. 2022.

PYTHON. **Documentação versão de python 3.10.5.** The python standard library. Development tools. unittest.mock - mock object library. Versão em inglês. Disponível em: <https://docs.python.org/3/library/unittest.mock.html>. Acesso: 08 jul. 2022.

PEIXOTO, R. **Selenium webdriver:** Descomplicando testes automatizados com Java. Casa do Código - Livros para o programador. 2018.

RAGHAVENDRA. **Python testing with selenium**: learn to implement different testing techniques using selenium webdriver. India, Dharwad Karnataka: Appres(R), 2021.

SALE, D. **Testing python**: applying unit testing, TDD, BDD, and accepting testing. Wiley, 2014.

SELENIUM. **Documentação** - o projeto selenium de automação de navegadores. The selenium umbrella. 2021. Versão online. Disponível em: <https://www.selenium.dev/pt-br/documentation/>. Acesso: 09 jul. 2022.