

Interação com elementos da página web

Jailma Januário da Silva

Leonardo Massayuki Takuno

Resumo

Os objetivos desta parte são: (I) Revisar os tipos de localizações. (II) Aprender a interagir com elementos de formulários como comboboxes, links, botões e campos de textos. (III) Aprender a utilizar o objeto da classe Select para acessar listas. (IV) Aprender a interagir e verificar o estado de um radio button. (V) Realizar um preenchimento automático de formulários de cadastro.

Introdução

O texto base da parte 7 apresentou conceitos iniciais sobre a utilização do Selenium, que é um arcabouço de testes para aplicações web. Como visto anteriormente, o Selenium Webdriver é uma coleção de ferramentas de código aberto específicas para construir scripts automatizados. Existem drivers específicos para os mais diversos navegadores, os quais incluem Google Chrome, Mozilla Firefox e Internet Explorer, entre outros. Esses drivers devem ser obtidos por meio da realização de download do site do fabricante, e após isso colocado em alguma pasta que tenha um PATH configurado no sistema operacional.

Algumas operações podem ser realizadas utilizando o Selenium Webdriver, tais como localizar elementos web, pelo nome, id, tag name entre outros, e após isso enviar algum tipo de ação, tal como enviar uma ação de click do

mouse, ou uma ação do teclado, ou até mesmo, realizar manipulações do navegador, como maximizar.

Este texto trabalha ainda com a interação, inicialmente revisando os localizadores, enfatizando alguns dos localizadores que não foram utilizados até o momento, e ainda, trabalha a interação com outros tipos de elementos web que não foram cobertos ainda, os quais incluem formulários, caixas de textos, caixas de checagem (do inglês, *checkbox*), listas, entre outros.

Localizadores

O Selenium utiliza os localizadores para realizar a busca e assim conseguir aplicar algum tipo de ação. Por exemplo, para acionar o clique do mouse sobre um elemento botão, deve primeiro encontrar o elemento botão por meio de algum tipo de localizador, como por exemplo TAG_NAME, com esse elemento em mãos, executar o método para acionar o clique do mouse. Os localizadores encontram-se na Tabela 8.1.

Tabela 8.1. Localizadores



Fonte: do autor, 2022.

As seções seguintes são baseadas e adaptadas do tutorial online intitulado Selenium with Python (MUTHUKADAN, 2018).

Localizando elementos pelo ID

O ID é um dos principais localizadores, portanto, utilize este localizador quando se conhece o atributo ID. Caso o ID não for encontrado pela função

find_element, o Selenium lançará uma exceção do tipo NoSuchElementException.

Codificação 8.1. pagina_exemplo_01.html

```
<html>
<body>
  <p>Digite o seu nome:</p>
  <form id="form_dados">
    <input name="nome" type="text">
    <input name="sobrenome" type="text">
    <button type="submit">Enviar</button>
  </form>
</body>

</html>
```

Fonte: do autor, 2022.

Para localizar o elemento form da Codificação 8.1 utiliza-se a instrução a seguir:

```
form = navegador.find_element(By.ID, "form_dados")
```

Localizando elementos pelo NAME

O método find_element() devolve o primeiro elemento web que possua o atributo NAME que foi passado como parâmetro. Caso não encontre, o método find_element() gera uma exceção NoSuchElementException. Para localizar os elementos nome e sobrenome da Codificação 8.1 utiliza-se a instrução a seguir:

```
nome = navegador.find_element(By.NAME, "nome")
```

```
sobrenome = navegador.find_element(By.NAME, "sobrenome")
```

Localizando elementos pelo TAG_NAME

O método find_element() devolve o primeiro elemento web que possua o atributo TAG_NAME que foi passado como parâmetro. Caso não encontre, o

método `find_element()` gera uma exceção `NoSuchElementException`. Para localizar o elemento parágrafo (`<p>`) e botão Enviar da Codificação 8.1 utiliza-se a instrução a seguir:

```
paragrafo = navegador.find_element(By.TAG_NAME, "p")
botao = navegador.find_element(By.TAG_NAME, "button")
```

Localizando elementos pelo XPATH

XPath é uma linguagem para localizar elementos em um XML. Como o HTML pode ser implementado como um XML (XHTML), o Selenium permite a localização de elementos web utilizando o XPath. Para localizar o elemento form da Codificação 8.1 utiliza-se uma das formas a seguir:

1. Caminho absoluto
2. Primeiro elemento form do HTML
3. O atributo form com atributo id com valor form_dados.

```
form = navegador.find_element(By.XPATH, "/html/body/form[1]")
form = navegador.find_element(By.XPATH, "//form[1]")
form = navegador.find_element(By.XPATH, "//form[@id='form_dados']")
```

O elemento nome pode ser localizado da seguinte forma:

```
nome = navegador.find_element(By.XPATH,
"//form[input/@name='nome']")
nome = navegador.find_element(By.XPATH,
"//form[@id='form_dados']/input[1]")
nome = navegador.find_element(By.XPATH, "//input[@name='nome']")
```

Por ora, estes são exemplos básicos o suficiente para iniciar a busca de elementos utilizando XPath. Junto com este conteúdo procure o conteúdo em Saiba mais, lá existem algumas sugestões de leituras para aprofundamento sobre o conteúdo de XPath.

Localizando Hiperlinks pelo texto

O método `find_element()` com o localizador `LINK_TEXT` devolve o primeiro elemento âncora que possua o texto do link que foi passado como parâmetro. Caso o localizador seja `PARTIAL_LINK_TEXT`, basta passar uma substring do texto do link para que o método `find_element()` devolva o primeiro elemento âncora que satisfaça a condição de busca. Caso não encontre, o método `find_element()` gera uma exceção `NoSuchElementException`.

Codificação 8.2. pagina_exemplo_02.html

```
<html>
<body>
  <a href="http://www.youtube.com">Vídeos no youtube</a>
  <a href="http://www.google.com">Pesquisa no google</a>
</body>

</html>
```

Fonte: do autor, 2022.

Para localizar os elementos da Codificação 8.2 utiliza-se a instrução a seguir:

```
ancora_google = navegador.find_element(By.LINK_TEXT, "Pesquisa no
google")
```

```
ancora_youtube = navegador.find_element(By.PARTIAL_LINK_TEXT,
"youtube")
```

Localizando elementos pelo CLASS_NAME

O método `find_element()` com o localizador `CLASS_NAME` devolve o primeiro elemento que possua o atributo class name que foi passado como parâmetro. Caso não encontre, o método `find_element()` gera uma exceção `NoSuchElementException`.

Codificação 8.3. pagina_exemplo_03.html

```
<html>
<body>
  <p class="conteudo">O conteúdo vem aqui</p>
</body>
```

</html>

Fonte: do autor, 2022.

Para localizar os elementos da Codificação 8.3 utiliza-se a instrução a seguir:

```
conteudo = navegador.find_element(By.CLASS_NAME, "conteudo")
```

Localizando elementos pelo CSS_SELECTOR

O método `find_element()` com o localizador `CSS_SELECTOR` devolve o primeiro elemento que possua o atributo seletor css que foi passado como parâmetro. Caso não encontre, o método `find_element()` gera uma exceção `NoSuchElementException`. Para localizar os elementos da Codificação 8.3 utiliza-se a instrução a seguir:

```
conteudo = navegador.find_element(By.CSS_SELECTOR, "p.conteudo")
```

Interações

Esta seção, que foi baseada no livro Selenium WebDriver: Descomplicando testes automatizados com Java (PEIXOTO, 2018), e adaptado para a linguagem Python, trata de alguns elementos bastante comuns em formulários de cadastros em geral. Esses elementos incluem listas, que é o termo conhecido como *combobox*, opções de seleção, também conhecidos como *radio buttons* e caixas de checagem, ou *checkbox*, além de links e campos de textos que foram vistos anteriormente.

Interações com listas

As listas são elementos bastante conhecidos e utilizados em formulários web. A tag para esse tipo de elemento é o `<select>`. Um `select` contém vários elementos de tag `<option>`, as quais são as opções que devem ser escolhidas na página. Observe a Codificação 8.4 para um exemplo utilizando uma lista.

Codificação 8.4. pagina_exemplo_04.html

```
<html>
<body>
  <p class="conteudo">O conteúdo vem aqui</p>
  <select name="opcoes">
    <option value="1">C/C++</option>
    <option value="2" selected>Python</option>
    <option value="3">Java</option>
    <option value="4">C#</option>
    <option value="5">PHP</option>
    <option value="6">JavaScript</option>
  </select>
</body>
</html>
```

Fonte: do autor, 2022.

Para interagir com as listas, é preciso seguir com as seguintes etapas:

- Identificar o elemento select;
- Escolher do elemento <option>, que podem ser realizadas por três métodos:
 - por índice;
 - por valor; ou
 - por texto visível.

Codificação 8.5. exemplo_04.py

```
import os
```

```
import time
```

```
from selenium.webdriver import Chrome
```

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

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

```
from webdriver_manager.chrome import ChromeDriverManager
```

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

```
servico = Service(ChromeDriverManager().install())
chrome = Chrome(service=servico)

local_path = f'file://{os.path.dirname(os.path.realpath(__file__))}/'
chrome.get(f'{local_path}pagina_exemplo_04.html')

select_element = chrome.find_element(By.NAME, 'opcoes')
select_object = Select(select_element)
select_object.select_by_index(1)
time.sleep(0.8)
select_object.select_by_value('3')
time.sleep(0.8)
select_object.select_by_visible_text('PHP')
time.sleep(0.8)
select_object.select_by_visible_text('Python')
```

Fonte: do autor, 2022.

Observe a Codificação 8.5, em que o script inicia o Google Chrome com o arquivo html local. Após isso, obtém-se o elemento select, a localização é feita por NAME com valor 'opcoes'. De posse do elemento web, o script cria uma instância da classe Select. A partir daí, é possível selecionar por índice, utilizando o método `select_by_index()`, ou selecionar por valor, utilizando o método `select_by_value()`, e então, pelo texto visível, utilizando o método `select_by_visible_text()`.

Interações com *radio buttons* e *checkbox*

Tanto o *radio button* quanto o *checkbox* são definidos pela tag `<input>`, porém, cada um tem um **type** diferente, conforme os exemplos abaixo.

```
<input type="radio" id="Python" name="linguagem" value="Python">
<input type="checkbox" id="Python" name="linguagem" value="Python">
```

Como exemplo, observe a Codificação 8.6.

Codificação 8.6. pagina_exemplo_05.html

```
<html>
<body>
  <p>Linguagem web favorita:</p>
  <input type="radio" id="C/C++" name="linguagem" value="C/C++">
  <label for="C/C++">C/C++</label><br>
  <input type="radio" id="Python" name="linguagem" value="Python">
  <label for="Python">Python</label><br>
  <input type="radio" id="Java" name="linguagem" value="Java">
  <label for="Java">Java</label><br>
  <input type="radio" id="C#" name="linguagem" value="C#">
  <label for="C#">C#</label><br>
  <input type="radio" id="javascript" name="linguagem"
value="JavaScript">
  <label for="JavaScript">JavaScript</label>
</body>
</html>
```

Fonte: do autor, 2022.

Observe que todos os radio buttons estão agrupados pelo localizador NAME de valor linguagem. Então para selecionar uma opção radio button, deve-se obter a lista de elementos, e por meio de uma estrutura de repetição procurar o elemento que deseja realizar a interação. Observe a Codificação 8.7, que contém um script que abre o arquivo pagina_exemplo_05.html localmente, e seleciona a opção Python.

Codificação 8.7. pagina_exemplo_05.html - 2

```
import os

from selenium.webdriver import Chrome
from selenium.webdriver.common.by import By
from selenium.webdriver.support.select import Select
```

```

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

servico = Service(ChromeDriverManager().install())
chrome = Chrome(service=servico)

local_path = f'file://{os.path.dirname(os.path.realpath(__file__))}'
chrome.get(f'{local_path}pagina_exemplo_05.html')

linguagens = chrome.find_elements(By.NAME, 'linguagem')
for linguagem in linguagens:
    if linguagem.get_attribute('value') == 'Python':
        linguagem.click()

for linguagem in linguagens:
    if linguagem.is_selected():
        print(linguagem.get_attribute('value'))

```

Fonte: do autor, 2022.

Observe pela Codificação 8.7, que todos os elementos com o localizador NAME de valor linguagem foram obtidos pelo método **find_elements()** e atribuiu a lista à variável linguagens. Após isso, o script procurou na lista o elemento cujo atributo value é igual à string 'Python'. Para isso, utilizou-se o método **get_attribute()**. Na sequência, executou-se novamente a iteração sobre a lista de linguagens para verificar qual foi o elemento selecionado e caso houver alguma linguagem selecionada imprime o valor do atributo value no console. Para verificar qual é o elemento selecionado utilizou-se o método **is_selected()**.

Preenchimento de cadastro automatizado

Esta seção apresenta como realizar um preenchimento automático de um formulário de cadastro. Para isso, será utilizado como exemplo, o cadastro de um usuário na rede social Facebook. O script realizará os seguintes passos:

- Acessar a página do Facebook pelo link <http://www.facebook.com>
- Preencher o formulário de cadastro, com dados fictícios.

Observe a Codificação 8.8 que utiliza o Google Chrome para acessar a página do Facebook e depois realizar o preenchimento do formulário de cadastro.

Codificação 8.8. cadastro_facebook.py

```
import time
```

```
from selenium import webdriver
```

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

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

```
from webdriver_manager.chrome import ChromeDriverManager
```

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

```
servico = Service(ChromeDriverManager().install())
```

```
driver = webdriver.Chrome(service=servico)
```

```
driver.maximize_window()
```

```
driver.implicitly_wait(0.5)
```

```
driver.get("https://www.facebook.com/")
```

```
bt_create_new_acc = driver.find_element(By.XPATH, "//a[text()='Create  
new account']")
```

```
bt_create_new_acc.click()
```

```
first_name = driver.find_element(By.NAME, "firstname")
```

```
first_name.send_keys('José')
```

```
last_name = driver.find_element(By.NAME, "lastname")
```

```
last_name.send_keys('Silva')
```

```
email = driver.find_element(By.NAME, "reg_email__")
```

```
email.send_keys('ze@email.com')
```

```
email_conf = driver.find_element(By.NAME, "reg_email_confirmation__")  
email_conf.send_keys('ze@email.com')
```

```
passwd = driver.find_element(By.NAME, "reg_passwd__")  
passwd.send_keys('1234')
```

```
month = driver.find_element(By.ID, "month")  
sel = Select(month)  
sel.select_by_index(11)
```

```
day = driver.find_element(By.ID, "day")  
sel = Select(day)  
sel.select_by_index(11)
```

```
year = driver.find_element(By.ID, "year")  
sel = Select(year)  
sel.select_by_value('1995')
```

```
sex_male = driver.find_element(By.XPATH, "///label[text()='Male']")  
sex_male.click()
```

```
botao_sign_up = driver.find_element(By.XPATH, "///button[text()='Sign  
Up']")  
botao_sign_up.click()
```

```
time.sleep(25)
```

```
error_msg = driver.find_element(By.ID, "reg_error_inner")  
print(error_msg.text)
```

Fonte: do autor, 2022.

Como os dados que o script preencheu no formulário são fictícios, o Facebook não vai permitir o cadastro. Ao acionar o botão Sign Up, o sistema do Facebook apresenta uma mensagem de erro como mostra a Figura 8.1.

Figura 8.1. Formulário de cadastro do Facebook



Fonte: do autor, 2022.

Considerações finais

Esta parte aprofundou conceitos relacionados aos localizadores, com vários exemplos simples de serem reproduzidos. Em seguida, o texto trabalha com interações de elementos web tais como listas, *radio buttons* e *checkbox*, os quais são importantes para automatizar os testes de sistemas web. Por fim, o texto inclui um exemplo para preenchimento automático de um formulário de cadastro da rede social Facebook, a qual utilizou todos os conceitos aprendidos nesta aparte e na anterior. Ainda existem alguns tópicos a serem cobertos pelo curso, porém, estes elementos de testes automatizados integrando Selenium Webdriver e ferramentas de testes de unidade como o pytest e o unittest compõe um conjunto de ferramentas que auxiliam, e muito, o trabalho de testes automatizados realizados na fase de desenvolvimento e, também, na fase de teste e garantia de qualidade.

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, S. **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.

MUTHUKADAN, B. **Selenium with python**. 2018. Disponível em: <<https://selenium-python.readthedocs.io/index.html>>. Acesso: 10 jul. 2022.