



Technologia i rozwiązania

Selenium i testowanie aplikacji Receptury

Wydanie II



Unmesh Gundecha

[PACKT] open source*
PUBLISHING community experience distilled

Tytuł oryginału: Selenium Testing Tools Cookbook, Second Edition

Tłumaczenie: Radosław Meryk

ISBN: 978-83-283-3411-3

Copyright © Packt Publishing 2015

First published in the English language under the title
'Selenium Testing Tools Cookbook - Second Edition - (9781784392512)'

Polish edition copyright © 2017 by Helion SA
All rights reserved.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz Wydawnictwo HELION dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz Wydawnictwo HELION nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: helion@helion.pl
WWW: <http://helion.pl> (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
<http://helion.pl/user/opinie/seler2>
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)

Spis treści

O autorze	7
O recenzentach	9
Przedmowa	11
Rozdział 1. Od czego zacząć?	17
Wprowadzenie	17
Konfigurowanie środowiska tworzenia testów Selenium WebDriver dla języka Java z Eclipse i Maven	18
Wykorzystanie narzędzia Ant do egzekucji testów Selenium WebDriver	26
Konfigurowanie programu Microsoft Visual Studio do tworzenia testów Selenium WebDriver	29
Konfigurowanie frameworka Selenium WebDriver dla Pythona i Ruby	32
Konfiguracja programu Internet Explorer Driver Server	36
Konfiguracja sterownika ChromeDriver przeglądarki Google Chrome	40
Konfiguracja sterownika Microsoft WebDriver dla przeglądarki Microsoft Edge	43
Rozdział 2. Wyszukiwanie elementów	47
Wprowadzenie	47
Wykorzystywanie narzędzi przeglądarki do badania elementów i struktury strony	48
Wyszukiwanie elementu za pomocą metody findElement	54
Wyszukiwanie wielu elementów za pomocą metody findElements	59
Wyszukiwanie linków	60
Wyszukiwanie elementów na podstawie nazwy znacznika	61
Wyszukiwanie elementów z wykorzystaniem XPath	62
Wyszukiwanie elementów na podstawie selektorów CSS	70
Lokalizowanie elementów na podstawie tekstu	74
Wyszukiwanie elementów na podstawie zaawansowanych selektorów CSS	76
Wykorzystanie selektorów jQuery	78

Rozdział 3. Obsługa elementów HTML	81
Wprowadzenie	81
Automatyzowanie pól tekstowych, obszarów tekstowych i przycisków	82
Sprawdzanie tekstu elementu	84
Sprawdzanie atrybutów elementu i wartości CSS	86
Automatyzowanie rozwijanych menu i list	87
Zaznaczanie opcji w elementach Select	90
Zaznaczanie wybranych opcji na rozwijanych menu i listach	92
Automatyzacja przełączników i grup opcji	95
Automatyzowanie pól wyboru	97
Praca z obiektami WebTable	99
Rozdział 4. Korzystanie z Selenium API	101
Wprowadzenie	101
Sprawdzanie istnienia elementu	102
Sprawdzanie stanu elementu	103
Wykorzystanie API Advanced User Interactions do obsługi zdarzeń myszy i klawiatury	104
Wykonywanie dwukrotnych kliknięć elementów	106
Wykonywanie operacji „przeciągnij i upuść”	108
Obsługa menu kontekstowych	109
Wykonywanie kodu JavaScript	111
Przechwytywanie zrzutów ekranu za pomocą Selenium WebDriver	113
Maksymalizowanie okna przeglądarki	115
Obsługa plików cookie sesji	115
Korzystanie z mechanizmów nawigacji przeglądarki	117
Korzystanie ze zdarzeń frameworka WebDriver	118
Rozdział 5. Synchronizacja testów	123
Wprowadzenie	123
Synchronizacja testów z niejawnym oczekiwaniem	124
Synchronizacja testów z jawnym oczekiwaniem	125
Synchronizacja testów z niestandardowymi warunkami oczekiwanymi	127
Synchronizacja testów z wykorzystaniem klasy FluentWait	130
Rozdział 6. Alerty, ramki i okna	133
Wprowadzenie	133
Obsługa prostych okien alertów JavaScript	133
Obsługa okien confirm i prompt	136
Identyfikowanie i obsługa ramek	139
Ramki iframe	144
Identyfikowanie i obsługa okien potomnych	146
Identyfikowanie i obsługa okien na podstawie tytułu	148
Identyfikowanie i obsługa wyskakujących okien na podstawie zawartości	150

Rozdział 7. Testowanie sterowane danymi	153
Wprowadzenie	153
Tworzenie testów sterowanych danymi z wykorzystaniem JUnit	155
Tworzenie testów sterowanych danymi z wykorzystaniem frameworka TestNG	159
Odczytywanie danych testowych z pliku CSV z wykorzystaniem JUnit	161
Odczytywanie danych testowych z pliku Excela z wykorzystaniem JUnit i Apache POI	164
Tworzenie testów sterowanych danymi z wykorzystaniem NUnit	167
Tworzenie testów sterowanych danymi z wykorzystaniem MSTEST	171
Tworzenie testów sterowanych danymi w Ruby z wykorzystaniem Roo	175
Tworzenie testów sterowanych danymi w Pythonie z wykorzystaniem DDT	179
Rozdział 8. Korzystanie ze wzorca Page Object	181
Wprowadzenie	181
Korzystanie z klasy PageFactory do udostępniania elementów na stronie	182
Korzystanie z klasy PageFactory do udostępniania operacji na stronie	186
Korzystanie z klasy LoadableComponent	189
Implementacja zagnieżdżonych egzemplarzy klasy Page Object	192
Implementacja modelu Page Object w środowisku .NET	197
Implementacja modelu Page Object w Pythonie	200
Implementacja modelu Page Object w Ruby z wykorzystaniem gemu page-object	203
Rozdział 9. Rozszerzanie Selenium	205
Wprowadzenie	205
Tworzenie klas rozszerzeń dla tabel webowych	206
Tworzenie rozszerzenia dla widżetu tab biblioteki jQueryUI	210
Implementacja rozszerzenia klasy WebElement	
w celu ustawiania wartości atrybutów elementów	214
Implementacja rozszerzenia interfejsu WebElement w celu wyróżniania elementów	216
Utworzenie mapy obiektów dla testów Selenium	217
Przechwytywanie zrzutów ekranu elementów w Selenium WebDriver	223
Porównywanie obrazów w Selenium	224
Mierzenie wydajności z wykorzystaniem API Navigation Timing	229
Rozdział 10. Testowanie aplikacji webowych HTML5	233
Wprowadzenie	233
Automatyzowanie odtwarzacza wideo HTML5	234
Automatyzowanie interakcji z elementem HTML5 canvas	237
Web storage — testowanie lokalnego magazynu	239
Web storage — testowanie pamięci sesji	241
Czyszczenie magazynu lokalnego i magazynu sesji	243

Rozdział 11. BDD — tworzenie oprogramowania sterowane zachowaniami	245
Wprowadzenie	245
Wykorzystanie technik BDD w Javie za pomocą biblioteki Cucumber-JVM i frameworka Selenium WebDriver	246
Wykorzystanie technik BDD w środowisku .NET za pomocą biblioteki SpecFlow.NET i frameworka Selenium WebDriver	255
Wykorzystanie biblioteki Capybara, Cucumber i frameworka Selenium WebDriver w Ruby	264
Wykorzystanie biblioteki Behave i frameworka Selenium WebDriver w Pythonie	267
Rozdział 12. Integracja z innymi narzędziami	271
Wprowadzenie	271
Konfigurowanie systemu ciągłej integracji Jenkins	272
Wykorzystanie systemów Jenkins i Maven do egzekucji testów Selenium WebDriver w środowisku ciągłej integracji	274
Wykorzystanie systemu Ant do uruchamiania testów Selenium WebDriver	279
Wykorzystanie systemów Jenkins i Ant do uruchamiania testów Selenium WebDriver w środowisku ciągłej integracji	281
Automatyzacja aplikacji z niewebowymi interfejsami użytkownika z wykorzystaniem Selenium WebDriver oraz AutoIt	286
Automatyzacja aplikacji z niewebowymi interfejsami użytkownika z wykorzystaniem Selenium WebDriver oraz Sikuli	293
Rozdział 13. Testy dla wielu przeglądarek	297
Wprowadzenie	297
Konfigurowanie programu Selenium Grid Server do pracy w trybie równoległego uruchamiania testów	299
Dodawanie węzłów do siatki Selenium Grid w celu testowania różnych przeglądarek	301
Tworzenie i wykonywanie skryptu Selenium równoległe z TestNG	304
Tworzenie i równoległe wykonywanie skryptu Selenium z wykorzystaniem języka Python	310
Wykorzystanie narzędzi w chmurze do testowania różnych przeglądarek i uruchamiania testów w chmurze	313
Uruchamianie testów w trybie headless z wykorzystaniem PhantomJS	316
Rozdział 14. Testowanie aplikacji w przeglądarkach mobilnych	319
Wprowadzenie	319
Konfigurowanie Appium do testowania aplikacji mobilnych	320
Testowanie mobilnych aplikacji webowych w systemie iOS z wykorzystaniem Appium	322
Testowanie mobilnych aplikacji webowych w systemie Android z wykorzystaniem Appium	325
Skorowidz	331

Korzystanie z Selenium API

Oto zagadnienia, jakie zostaną omówione w tym rozdziale.

- Sprawdzanie istnienia elementu.
- Sprawdzanie stanu elementu.
- Wykorzystanie API zaawansowanych interakcji użytkownika (ang. *Advanced User Interactions*) do obsługi zdarzeń myszy i klawiatury.
- Wykonywanie podwójnych kliknięć elementu.
- Wykonywanie operacji „przeciągnij i upuść”.
- Obsługa menu kontekstowych.
- Wykonywanie kodu JavaScript.
- Przechwytywanie zrzutów ekranu za pomocą Selenium WebDriver.
- Maksymalizowanie okna przeglądarki.
- Obsługa plików cookie sesji.
- Obsługa mechanizmów nawigacji przeglądarki.
- Korzystanie ze zdarzeń klasy WebDriver.

Wprowadzenie

Selenium WebDriver implementuje kompleksowe API do pracy z elementami webowymi, które umożliwia wykonywanie zaawansowanych interakcji z użytkownikiem, takich jak złożone zdarzenia związane z myszą i klawiaturą, wykonywanie kodu JavaScript, przechwytywanie zrzutów ekranu i tak dalej.

W tym rozdziale omówię sposób wykorzystania tych własności do budowania kroków testów o różnym poziomie złożoności. Lektura rozdziału pomoże również w przezwycięzeniu niektórych typowych problemów, które mogą powstawać podczas tworzenia testów bazujących na frameworku Selenium WebDriver.

Sprawdzanie istnienia elementu

Selenium WebDriver nie implementuje metody Selenium RC `isElementPresent()`, która sprawdza, czy element istnieje na stronie. Metoda jest przydatna do budowania niezawodnego testu, który mógłby sprawdzić istnienie elementu przed wykonaniem na nim jakiegokolwiek działania.

W tej recepturze napiszę metodę podobną do metody `isElementPresent()`.

Jak to zrobić?

Aby zaimplementować metodę `isElementPresent()`, wykonaj następujące czynności.

1. Utwórz metodę `isElementPresent()` w module dostępnym do testów i umieść w niej następujący kod.

```

{{Private boolean isElementPresent(By by)
    try {
        driver.findElement(by);
        return true;
    } catch {} (NoSuchElementException e)
        return false;
    }
}

```

2. Teraz utwórz kod testu, który wywołuje metodę `isElementPresent()`. Metoda będzie sprawdzać, czy żądany element istnieje na stronie. Jeśli będzie znaleziony, zostanie wykonane kliknięcie elementu, w przeciwnym razie test nie przejdzie. Można to zrobić w następujący sposób.

```

@Test
Public void testIsElementPresent() {}
    // Sprawdzenie, czy element spełniający kryteria lokalizatora istnieje na stronie
    if (isElementPresent(By.name("airbags"))) {
        // Utwórz reprezentację pola wyboru i zaznacz je
        WebElement airbag = driver.findElement(By.name("airbags"));
        if (!airbag.isSelected()) {
            airbag.click();
        }
    } else {
        fail("Pole wyboru poduszki powietrznej nie istnieje!!");
    }
}

```


Jak to działa?

Metoda `isElementPresent()` pobiera argument lokalizatora z wykorzystaniem egzemplarza klasy `By`. Następnie wywołuje metodę `findElement()`. Jeśli element nie zostanie znaleziony, będzie zgłoszony wyjątek `NoSuchElementException`. Dzięki użyciu bloku `try` i `catch` metoda `isElementPresent()` zwróci `true`, gdy element zostanie znaleziony. W tym przypadku nie zostanie zgłoszony wyjątek. W przeciwnym razie zwróci `false`, a metoda `findElement()` zgłosi wyjątek `NoSuchElementException`.

Zobacz też

- Receptura „Sprawdzanie stanu elementu”.

Sprawdzanie stanu elementu

Wielokrotnie zdarza się, że próba kliknięcia elementu lub wprowadzenia tekstu w polu zawodzi ze względu na to, że element jest wyłączony lub istnieje w modelu DOM, ale jest ukryty na stronie. To powoduje zgłoszenie błędu i niepowodzenie testu. Zbudowanie wiarygodnych testów, które można uruchomić w trybie nienadzorowanym, wymaga utworzenia w przepływie testów solidnego mechanizmu obsługi błędów i wyjątków.

Powstające problemy można obsłużyć, sprawdzając stan elementów. Interfejs `WebElement` udostępnia następujące metody sprawdzania stanu elementu.

Metoda	Przeznaczenie
<code>isEnabled()</code>	Metoda sprawdza, czy element jest aktywny. Zwraca <code>true</code> , jeśli jest aktywny, i <code>false</code> w przypadku przeciwnym.
<code>isSelected()</code>	Metoda sprawdza, czy element (przełącznik, pole wyboru i tak dalej) jest zaznaczony. Zwraca <code>true</code> , jeśli jest zaznaczony, i <code>false</code> w przypadku przeciwnym.
<code>isDisplayed()</code>	Metoda sprawdza, czy element jest wyświetlany.

W tej recepturze zastosuję niektóre z wymienionych metod w celu sprawdzenia stanu elementów i obsługi błędów, jeśli to możliwe.

Jak to zrobić?

Utworzymy test, w ramach którego sprawdzimy, czy na stronie zostało zaznaczone pole wyboru włączające reflektor LED. To pole wyboru będzie włączone lub wyłączone, w zależności od wcześniej wybranych opcji. Przed zaznaczeniem tego pola wyboru upewnimy się, czy jest ono aktywne. Można to zrobić w następujący sposób.

```

@Test
public void testElementIsEnabled() {
    // Tworzenie reprezentacji pola wyboru w postaci obiektu WebElement z wykorzystaniem atrybutu name
    WebElement ledheadlamp = driver.findElement(By.name("ledheadlamp"));
    // Sprawdzenie, czy element jest aktywny przed jego zaznaczeniem
    if (ledheadlamp.isEnabled()) {
        // Sprawdzenie, czy pole wyboru już jest zaznaczone. W przeciwnym razie zaznaczenie go
        if (!ledheadlamp.isSelected()) {
            ledheadlamp.click();
        }
    } else {
        fail("Pole wyboru reflektora LED jest nieaktywne!!");
    }
}

```

Jak to działa?

Zaznaczamy pole wyboru, sprawdzając dwa stany elementu; po pierwsze, czy pole to jest aktywne, i po drugie, czy już nie jest zaznaczone. Do tego celu możemy użyć funkcji `isEnabled()` interfejsu `WebElement`, która zwraca `true`, jeśli element jest aktywny, lub `false`, gdy jest nieaktywny. Jeśli pole wyboru jest nieaktywne, test zakończy się niepowodzeniem. Kiedy nie sprawdzimy tego warunku, test prawdopodobnie zgłosi wyjątek informujący o tym, że obiekt jest nieaktywny, tak jak pokazano poniżej.

```

// Sprawdzenie, czy element jest aktywny przed jego zaznaczeniem
if (ledheadlamp.isEnabled()) {
    // Sprawdzenie, czy pole wyboru jest już zaznaczone. W przeciwnym razie zaznaczenie pola wyboru
    if (!ledheadlamp.isSelected()) {
        ledheadlamp.click();
    }
} else {
    fail("Pole wyboru reflektora LED jest nieaktywne!!");
}

```

Wykorzystanie API Advanced User Interactions do obsługi zdarzeń myszy i klawiatury

API *Advanced User Interactions* frameworka Selenium WebDriver pozwala wykonywać zarówno proste, jak i złożone operacje obsługi zdarzeń klawiatury i myszy. Klasa `Actions` udostępnia tak złożone operacje jak „przeciągnij i upuść”, czy przytrzymanie klawisza, a następnie wykonanie operacji myszą. API umożliwia także budowanie złożonych łańcuchów zdarzeń imitujących działania ręcznie wykonywane przez użytkownika.

W celu utworzenia złożonych działań obejmujących grupę innych działań klasa `Actions` implementuje wzorzec „budowniczy”.

W tej recepturze użyję klasy `Actions` do utworzenia łańcucha zdarzeń wymaganych do zaznaczenia wierszy w tabeli.

Jak to zrobić?

Utwórzmy test, który zaznacza wiele wierszy z różnych pozycji w tabeli za pomocą klawisza `Ctrl` (w komputerach Mac `Command`). Możemy wybrać wiele wierszy: zaznaczamy pierwszy wiersz, przytrzymujemy klawisz `Ctrl` (`Command` na komputerach Mac), a następnie wybieramy inny wiersz i zwalniamy klawisz `Ctrl` (`Command`). Takie działanie spowoduje zaznaczenie żądanych wierszy z tabeli, tak jak pokazano w poniższym fragmencie kodu.

```
@Test
public void testRowSelectionUsingControlKey() {

    List<WebElement> tableRows = driver.findElements
        (By.xpath("//table[@class='iceDatTbl']/tbody/tr"));

    // Zaznacz drugi i czwarty wiersz w tabeli za pomocą klawisza Control.
    // Wiersze rozpoczynają się od indeksu 0
    Actions builder = new Actions(driver);
    builder.click(tableRows.get(1))
        .keyDown(Keys.CONTROL)
        .click(tableRows.get(3))
        .keyUp(Keys.CONTROL)
        .build().perform();

    // Sprawdzenie, czy w tabeli są zaznaczone dwa wiersze
    List<WebElement> rows = driver.findElements
        (By.xpath("//div[@class='icePnlGrp
        exampleBox']/table[@class='iceDatTbl']/tbody/tr"));
    assertEquals(2, rows.size());
}
```

W komputerach z systemem Mac OS X należy użyć składni klawisza `Command` zamiast składni klawisza `Control`. Oto przykład.

```
Actions builder = new Actions(driver);
builder.click(tableRows.get(1)).keyDown(Keys.COMMAND)
    .click(tableRows.get(3)).keyUp(Keys.COMMAND).perform();}
```

Jak to działa?

Należy utworzyć egzemplarz klasy `Actions`, przekazując do konstruktora egzemplarz klasy `driver` w następujący sposób.

```
Actions builder = new Actions(driver);
```

Zbudujemy łańcuch zdarzeń, które należy wykonać, aby zaznaczyć wiersze. Będzie to wymagało wykonania operacji `click()` na pierwszym wierszu, następnie przytrzymania klawisza *Ctrl* (*Command* w systemie Mac) przy użyciu operacji `keyDown()`, kliknięcia wiersza końcowego, a następnie zwolnienia klawisza *Ctrl* (*Command*) przez wywołanie `keyUp()`. Klasa `Actions` oferuje różne metody do wykonywania operacji z klawiaturą i myszą.

```
Actions builder = new Actions(driver);
builder.click(tableRows.get(1)).keyDown(Keys.CONTROL)
        .click(tableRows.get(3)).keyUp(Keys.CONTROL)
        .build().perform();
```

Możemy utworzyć złożone działania, które wykonamy przy użyciu wywołania metody `perform()` klasy `Actions`.

Klasa `Keys` reprezentuje wszystkie nietekstowe klawisze na klawiaturze, na przykład klawisz *Ctrl*, klawisz *Shift*, klawisze funkcyjne i tak dalej. W poprzednim przykładzie użyliśmy wywołania `keyDown(Keys.CONTROL)` w celu naciśnięcia i przytrzymania klawisza *Ctrl* (*Command*) aż do zakończenia następnej operacji.

Działania mogą nie uruchamiać się poprawnie w przypadku elementów, które nie są widoczne lub są nieaktywne. Przed użyciem opisanych zdarzeń należy zadbać o to, aby elementy były widoczne i aktywne.

Zobacz też

- Receptura „Wykonywanie dwukrotnych kliknięć elementów”.

Wykonywanie dwukrotnych kliknięć elementów

W aplikacjach webowych są elementy, które do wykonania niektórych czynności wymagają dwukrotnych kliknięć. Przykładowo dwukrotne kliknięcie wiersza tabeli powoduje wyświetlenie nowego okna. API `Advanced User Interaction` dostarcza metodę umożliwiającą wykonywanie dwukrotnych kliknięć.

W tej recepturze do wykonywania operacji dwukrotnego kliknięcia użyję klasy `Actions`.

Jak to zrobić?

Utwórzmy test lokalizujący element, dla którego zaimplementowano zdarzenie dwukrotnego kliknięcia. Dwukrotne kliknięcie tego elementu spowoduje zmianę jego koloru.

```

package com.secookbook.examples.chapter04;

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.By;
import org.openqa.selenium.interactions.Actions;
import static org.junit.Assert.*;
import org.junit.Test;

public class DoubleClickTest {
    @Test
    public void testDoubleClick() throws Exception {
        WebDriver driver = new ChromeDriver();
        driver.get("http://cookbook.seleniumacademy.com/DoubleClickDemo.html");

        try {
            WebElement message = driver.findElement(By.id("message"));

            // Sprawdzenie, czy element jest niebieskiego koloru
            assertEquals("rgba(0, 0, 255, 1)", message.
                ↪getCssValue("background-color"));

            Actions builder = new Actions(driver);
            builder.doubleClick(message).perform();

            // Sprawdzenie, czy element jest żółtego koloru
            assertEquals("rgba(255, 255, 0, 1)", message.
                ↪getCssValue("background-color"));
        } finally {
            driver.quit();
        }
    }
}

```

Jak to działa?

Aby wykonać dwukrotne kliknięcie elementu, należy wywołać metodę `doubleClick()` klasy `Actions`. W celu wywołania tej metody trzeba utworzyć egzemplarz klasy `Actions`, tak jak pokazano poniżej.

```
Actions builder = new Actions(driver);
```

Metoda `doubleClick()` wymaga elementu, na którym zostanie zainicjowane zdarzenie dwukrotnego kliknięcia. Metodę `doubleClick()` można wywołać, przekazując element w następujący sposób.

```
builder.doubleClick(message).perform();
```

Zobacz też

- Receptura „Wykorzystanie API Advanced User Interactions do obsługi zdarzeń myszy i klawiatury”,
- Receptura „Wykonywanie operacji «przeciągnij i upuść»”.

Wykonywanie operacji „przeciągnij i upuść”

Selenium WebDriver, za pośrednictwem klasy `Actions`, implementuje polecenie Selenium RC `dragAndDrop`. Klasa `Actions`, jak pokazywałem we wcześniejszych recepturach, obsługuje zaawansowane interakcje użytkownika, takie jak wyzwalanie zdarzeń myszy i klawiatury. Za pośrednictwem tej klasy można budować proste lub złożone łańcuchy zdarzeń.

W tej recepturze wykorzystam klasę `Actions` do wykonywania operacji „przeciągnij i upuść”.

Jak to zrobić?

Zaimplementujemy test, który za pomocą klasy `Actions` wykona na stronie operację „przeciągnij i upuść”.

```
@Test
public void testDragDrop() {}
    Driver.get ("http://cookbook.seleniumacademy.com/DragDropDemo.html");

    WebElement source = driver.findElement(By.id("draggable"));
    WebElement target = driver.findElement(By.id("droppable"));

    Actions builder = new Actions(driver);
    builder.dragAndDrop(source, target).perform();
    assertEquals("Upuszczono!", target.getText());
}
```

Jak to działa?

Aby przeciągnąć element na inny element i go na nim upuścić, trzeba zlokalizować oba elementy i przekazać je do metody `dragAndDrop()` klasy `Actions`. Żeby wywołać tę metodę, należy utworzyć egzemplarz klasy `Actions` w następujący sposób.

```
Actions builder = new Actions(driver);
```

Metoda `dragAndDrop()` wymaga elementu źródłowego oraz elementu docelowego, na który element źródłowy zostanie przeciągnięty i upuszczony. Metodę `dragAndDrop()` można wywołać w następujący sposób.

```
builder.dragAndDrop(source, target).perform();
```

W czasie powstawania tej książki framework Selenium WebDriver nie obsługiwał operacji „przeciągnij i upuść” dla HTML5. Więcej informacji na ten temat można znaleźć pod adresem <https://code.google.com/p/selenium/issues/detail?id=3604>. Istnieje obejście umożliwiające symulację własności „przeciągnij i upuść” w aplikacjach webowych bazujących na HTML5. Więcej szczegółów na ten temat można znaleźć pod adresem <https://gist.github.com/rcorreia/2362544>.

Zobacz też

- Receptura „Wykorzystanie API Advanced User Interactions do obsługi zdarzeń myszy i klawiatury”,
- Receptura „Wykonywanie dwukrotnych kliknięć elementów”.

Obsługa menu kontekstowych

Menu kontekstowe (znane także jako menu skrótów, wyskakujące menu lub menu podręczne) to lista poleceń wyświetlanych na stronie WWW w odpowiedzi na kliknięcie elementu prawym przyciskiem myszy. Przykładowo pod adresem <http://bit.ly/1CAV05I> zamieszczono kod korzystający z wtyczki jQuery contextMenu, który wyświetla menu edycji, gdy użytkownik kliknie przycisk prawym przyciskiem myszy.

Klasa Actions frameworka Selenium WebDriver dostarcza metodę contextClick(), która pozwala wykonać operację kliknięcia prawym przyciskiem myszy. W tej recepturze pokażę, w jaki sposób zautomatyzować interakcje z menu kontekstowymi.

Jak to zrobić?

Spróbujmy zaimplementować test, który otworzy menu kontekstowe i, korzystając z klasy Actions, wybierze jedną z opcji.

```
@Test
public void testContextMenu() {}
    WebElement clickMeElement =
        driver.findElement(By.cssSelector("div.context-menuone.Box.menu-1"));
    WebElement editMenuItem =
        driver.findElement(By.cssSelector("li.context-menuitem.icon-edit"));

    Actions builder = new Actions(driver);
    builder.contextClick(clickMeElement)
        .moveToElement(editMenuItem)
        .click()
        .perform();

    WebDriverWait wait = new WebDriverWait(driver, 10);
```

```

Alert alert = wait.until(ExpectedConditions.alertIsPresent());
assertEquals("clicked: edit", alert.getText());
alert.dismiss();
}

```

Jak to działa?

Do otwarcia menu kontekstowego w odpowiedzi na kliknięcie prawym przyciskiem myszy służy metoda `contextClick()` klasy `Actions`. Poniższy kod wykonuje kliknięcie prawym przyciskiem myszy podanego elementu.

```

WebElement clickMeElement =
    driver.findElement(By.cssSelector("div.context-menu-one.
box.menu-1"));
WebElement editMenuItem =
    driver.findElement(By.cssSelector("li.context-menuitem.
icon-edit"));

Actions builder = new Actions(driver);
builder.contextClick(clickMeElement)
    .moveToElement(editMenuItem)
    .click()

    .perform();

```

Następnie można przejść do żądanego elementu — w tym przypadku jest nim element ``, który reprezentuje jedną z opcji menu. Aby wykonać akcję menu, wywoływana jest metoda `click()`. W tym przykładzie wywołanie metody `click()` powoduje wyświetlenie alertu, a test sprawdza komunikat w oknie alertu.

Co dalej?

W poprzednim przykładzie test wyszukuje pozycję menu, a następnie wykonuje operację kliknięcia. Czasami w menu są dostępne klawisze skrótów. Za pośrednictwem klasy `Actions` można wykorzystać kombinację zdarzeń myszy i naciśnięć klawiszy w celu otwarcia żądanej opcji menu. Przykładowo klawiszem skrótu dla pozycji menu *Edit* jest „e”. Możemy utworzyć menu kontekstowe, a następnie wysłać kombinację klawiszy `Alt+E` w następujący sposób.

```

@Test
public void testContextMenuWithKeys() {
    WebElement clickMeElement =
        driver.findElement(By.cssSelector("div.context-menu-one.box.menu-1"));

    Actions builder = new Actions(driver);
    builder.contextClick(clickMeElement)
        .sendKeys(Keys.chord(Keys.ALT, "e"))
        .perform();
}

```



```

WebDriverWait wait = new WebDriverWait(driver, 10);

Alert alert = wait.until(ExpectedConditions.alertIsPresent());
assertEquals("clicked: edit", alert.getText());
alert.dismiss();
}

```

Zobacz też

- Receptura „Wykorzystanie API Advanced User Interactions do obsługi zdarzeń myszy i klawiatury”.

Wykonywanie kodu JavaScript

API Selenium WebDriver zapewnia możliwość wykonywania w oknie przeglądarki kodu JavaScript. Jest to bardzo przydatna funkcja, w przypadku gdy testy muszą wykonywać operacje na stronie przy użyciu JavaScript. Dzięki temu API frameworka Selenium WebDriver może być wykorzystane również do testowania kodu JavaScript po stronie klienta. Selenium WebDriver oferuje interfejs `JavascriptExecutor`, który może służyć do uruchamiania dowolnego kodu JavaScript w kontekście przeglądarki.

W tej recepturze pokażę, w jaki sposób używać interfejsu `JavascriptExecutor` do uruchamiania kodu JavaScript. W tej książce zamieszczono różne receptury, w których wykorzystano interfejs `JavascriptExecutor` do wykonywania niektórych zaawansowanych operacji, które nie są jeszcze obsługiwane przez Selenium WebDriver.

Jak to zrobić?

Utwórzmy test, który wywołuje kod JavaScript w celu zwrócenia tytułu i liczby linków (tzn. liczby tagów `Anchor`). Aby zwrócić tytuł strony, można również wykorzystać metodę `driver.getTitle()`. Poniżej przedstawiono przykładowy kod takiego działania.

```

@Test
public void testJavaScriptCalls() throws Exception {
    WebDriver driver = new ChromeDriver();
    driver.get("http://www.google.com");
    try {
        JavascriptExecutor js = (JavascriptExecutor) driver;

        String title = (String) js.executeScript("return document.title");
        assertEquals("Google", title);

        long links = (Long) js

```

```

        .executeScript("var links = document.
            ↪getElementsByTagName('A'); return links.length");
    assertEquals(42, links);
} finally {
    driver.quit();
}
}

```

Jak to działa?

Zrzutowanie egzemplarza WebDriver na interfejs JavascriptExecutor pozwala wykonać w Selenium WebDriver kod JavaScript.

```
JavascriptExecutor js = (JavascriptExecutor) driver;
```

W poniższym przykładzie uruchomiono pojedynczy wiersz kodu JavaScript, który zwraca tytuł strony wyświetlanej przez sterownik. Interfejs JavascriptExecutor dostarcza metodę executeScript(), do której należy przekazać kod JavaScript.

```
String title = (String) js.executeScript("return document.title");
```

Podczas zwracania wartości z kodu JavaScript trzeba użyć słowa kluczowego return. Wynik metody executeScript() trzeba zrzutować na podstawie typu zwróconej wartości. Dla wartości dziesiętnych można użyć typu Double, niedziesiętne wartości liczbowe wymagają użycia typu Long, natomiast dla wartości logicznych należy stosować typ Boolean. Jeśli kod JavaScript zwraca element HTML, można wykorzystać typ WebElement. Dla wartości tekstowych należy skorzystać z typu String. Jeśli jest zwracana lista obiektów, można zastosować dowolny z wymienionych typów, w zależności od typu obiektów na liście. W przeciwnym razie zostanie zwrócona wartość null.

W poniższym przykładzie w celu pobrania liczby linków na stronie uruchomiono wielowierszowy kod JavaScript.

```
long links = (Long) js.executeScript("var links =
    document.getElementsByTagName('A'); return links.length");
```

Co dalej?

Do kodu JavaScript uruchamianego za pomocą metody executeScript() mogą być przekazywane argumenty. W poniższym przykładzie chcemy ustawić wartość elementu. Wewnątrz kodu JavaScript zostanie wykorzystana specjalna tablica arguments, tak jak pokazano w poniższym kodzie.

```
js.executeScript("document.getElementById('name').value =
    arguments[0]", "Jan");
```

Przechwytywanie zrzutów ekranu za pomocą Selenium WebDriver

Selenium WebDriver dostarcza interfejs `TakesScreenshot`, który umożliwia przechwytywanie zrzutów ekranu stron internetowych. Pomaga to podczas uruchamiania testu, ponieważ pokazuje, co dokładnie się stało, gdy został zgłoszony wyjątek lub wystąpił błąd podczas uruchamiania. Można również przechwytywać zrzuty ekranu podczas weryfikacji stanu elementu, wartości wyświetlanych wewnątrz elementów lub stanu po zakończeniu działania.

Przechwytywanie zrzutów ekranu pomaga także w weryfikacji takich obszarów jak układ strony czy sposób wyrównania pól, kiedy porównywane są zrzuty ekranu wykonane podczas działania testów z ilustracjami bazowymi.

W tej recepturze użyję interfejsu `TakesScreenshot` do wykonania zrzutu ekranu testowanej strony WWW.

Jak to zrobić?

Utwórzmy test, który otworzy aplikację testową i wykona zrzut ekranu strony w formacie **PNG** (ang. *Portable Network Graphics*), tak jak pokazano w poniższym przykładowym kodzie.

```
@Test
public void testTakesScreenshot() throws Exception {
    File scrFile = ((TakesScreenshot) driver)
        .getScreenshotAs(OutputType.FILE);
    FileUtils.copyFile(scrFile, new File("target/main_page.png"));
}
```

Jak to działa?

Interfejs `TakesScreenshot` dostarcza metodę `getScreenshotAs()`, która pozwala przechwytać zrzut ekranu strony wyświetlanej wewnątrz egzemplarza obiektu `driver`. W poniższym przykładzie użyliśmy wartości `OutputType.FILE` jako argumentu metody `getScreenshotAs()`. Dzięki temu przechwycony zrzut ekranu zostanie zwrócony w pliku.

```
File scrFile =
    ((TakesScreenshot)driver).getScreenshotAs(OutputType.FILE);
```

Obiekt `file` zwrócony przez metodę `getScreenshotAs()` możemy zapisać za pomocą metody `copyFile()` klasy `FileUtils` należącej do klasy `org.apache.commons.io.FileUtils`.

Interfejs `TakesScreenshot` wykonuje zrzuty ekranu z wykorzystaniem API przeglądarki. Sterownik `HtmlUnit` nie obsługuje interfejsu `TakesScreenshot`.

Co dalej?

Klasa `OutputType` oferuje wiele sposobów prezentowania wyników wykonania zrzutów ekranu za pomocą metody `getScreenshotAs()`. W poprzednim przykładzie widzieliśmy zrzut ekranu zapisany w pliku. Zrzuty ekranu mogą być również przechwytywane w tekstowym formacie Base64 lub w formacie surowych bajtów. W poniższym przykładzie wykonano zrzut ekranu w formacie ciągu Base64.

```
String base64 =
    ((TakesScreenshot)driver).getScreenshotAs(OutputType.BASE64);
```

Przechwytywanie zrzutów ekranu w środowisku RemoteWebDriver (Selenium Grid)

Gdy testy są uruchamiane z wykorzystaniem interfejsu `RemoteWebDriver` lub `Selenium Grid`, nie można wykonywać zrzutów ekranu, ponieważ interfejs `TakesScreenshot` nie jest zaimplementowany w klasie `RemoteWebDriver`.

Możemy jednak użyć klasy `Augmenter`, która dodaje interfejs `TakesScreenshot` do egzemplarza `RemoteWebDriver`, tak jak pokazano w poniższym przykładzie kodu.

```
driver = new Augmenter().augment(driver);
File scrFile =
    ((TakesScreenshot)driver).getScreenshotAs(OutputType.FILE);
FileUtils.copyFile(scrFile, new File("c:\\tmp\\screenshot.png"));
```

Klasa `Augmenter` ulepsza klasę `RemoteWebDriver` poprzez dodanie do niej różnych interfejsów, w tym interfejsu `TakesScreenshot`.

```
driver = new Augmenter().augment(driver);
```

Po wykonaniu takiego rozszerzenia możemy użyć interfejsu `TakesScreenshot` z klasy `RemoteWebDriver` do przechwytywania ekranu.

Zobacz też

- Receptura „Przechwytywanie zrzutów ekranu elementów w Selenium WebDriver” w rozdziale 9., „Rozszerzanie Selenium”.
- Receptura „Porównywanie obrazów w Selenium” w rozdziale 9., „Rozszerzanie Selenium”.

Maksymalizowanie okna przeglądarki

W Selenium WebDriver brakowało polecenia `windowMaximize()` z API Selenium RC. Jednakże od wersji 2.21 Selenium WebDriver obsługuje maksymalizowanie okna przeglądarki.

W tej krótkiej recepturze pokażę, jak zmaksymalizować okno przeglądarki.

Przygotuj się

Utwórz nowy test, który zbuduje egzemplarz WebDriver, przejdzie do witryny i wykona kilka podstawowych działań i weryfikacji.

Jak to zrobić?

Aby zmaksymalizować okno przeglądarki, trzeba wywołać metodę `maximize()` interfejsu `Window` klasy `driver`. Dodaj drugi wiersz kodu z poniższego fragmentu, ten za instrukcją definiującą egzemplarz klasy `FirefoxDriver`.

```
driver = new FirefoxDriver();
driver.manage().window().maximize();
```

Obsługa plików cookie sesji

W witrynach internetowych używane są pliki cookie do przechowywania preferencji użytkownika, informacji o logowaniu oraz różnych innych szczegółów dotyczących klienta. Selenium WebDriver API zapewnia różne metody zarządzania plikami cookie podczas testowania. Za pomocą tych metod możemy odczytywać wartości plików cookie, dodawać pliki cookie oraz usuwać takie pliki podczas testowania. Może to służyć do testowania reakcji aplikacji podczas wykonywania działań z plikami cookie. Interfejs `WebDriver.Options` udostępnia następujące metody zarządzania plikami cookie.

Metoda	Opis
<code>addCookie(Cookie cookie)</code>	Dodaje plik cookie.
<code>getCookieNamed(String name)</code>	Zwraca plik cookie o podanej nazwie.
<code>getCookies()</code>	Zwraca wszystkie pliki cookie dla bieżącej domeny.
<code>deleteCookieNamed(String name)</code>	Usuwa plik cookie o podanej nazwie.
<code>deleteCookie(Cookie cookie)</code>	Usuwa plik cookie.
<code>deleteAllCookies()</code>	Usuwa wszystkie pliki cookie dla bieżącej domeny.

W tej recepturze pokażę, jak odczytywać pliki cookie i sprawdzać ich wartość.

Przygotuj się

Utwórz nowy test, który zbuduje egzemplarz WebDriver, przejdzie do witryny i wykona kilka podstawowych działań i weryfikacji.

Jak to zrobić?

Utwórzmy test, który odczytuje plik cookie i sprawdza jego wartość, tak jak pokazano w poniższym przykładzie kodu.

```
@Test
public void testCookies() {}
    Driver.get ("http://demo.magentocommerce.com/");

    // Utworzenie reprezentacji rozwijanego menu wyboru języka w postaci egzemplarza klasy Select
    Select language = new Select(driver.findElement(By
        .id("select-language")));

    // Sprawdzenie, czy domyślnie zaznaczona opcja to język polski
    assertEquals("Polski", language.getFirstSelectedOption().getText());

    // Zmienna storeCookie powinna mieć wartość null
    Cookie storeCookie = driver.manage().getCookieNamed("store");
    assertEquals(null, storeCookie);

    // Zaznacz opcję przy użyciu metody SelectByVisibleText
    language.selectByVisibleText("English");

    // Zmienna storeCookie powinna zawierać informację o wybranym języku
    storeCookie = driver.manage().getCookieNamed("store");
    assertEquals("english", storeCookie.getValue());
}
```

Jak to działa?

Interfejs `WebDriver.Options` dostarcza różne metody do dodawania, czytania, modyfikowania i usuwania plików cookie. W tym przykładzie, gdy zmieniamy język sklepu, do przechowywania preferencji językowych używany jest plik cookie. Ten plik cookie oraz jego wartość możemy odczytać w następujący sposób.

```
Cookie storeCookie = driver.manage().getCookieNamed("store");
assertEquals("english", storeCookie.getValue());
```

Wywołaliśmy metodę `getCookieNamed()`, przekazując nazwę pliku cookie. Metoda zwraca egzemplarz obiektu `Cookie`. Obiekt `Cookie` zawiera różne metody czytania wartości, domeny i tak dalej.

Korzystanie z mechanizmów nawigacji przeglądarki

Przeglądarki zapewniają wiele metod nawigacji umożliwiających dostęp do stron WWW z historii przeglądarki. Pozwalają również na odświeżanie bieżącej strony za pomocą przycisków *Wstecz*, *Dalej* oraz *Odśwież* na pasku narzędzi okna przeglądarki. API Selenium WebDriver zapewnia dostęp do tych przycisków za pomocą różnych metod interfejsu `WebDriver.Navigation`. Przy użyciu tych metod możemy testować zachowanie aplikacji.

Metoda	Opis
<code>back()</code>	Metoda powoduje przejście do poprzedniej strony w historii przeglądarki.
<code>forward()</code>	Metoda powoduje przejście do następnej strony w historii przeglądarki.
<code>refresh()</code>	Metoda powoduje ponowne załadowanie bieżącej strony.
<code>to(String url)</code>	Metoda ładuje podany adres URL w bieżącym oknie przeglądarki.
<code>to(java.net.URL url)</code>	

W tej recepturze zaprezentuję metody nawigacji w przeglądarce.

Przygotuj się

Utwórz nowy test, który zbuduje egzemplarz klasy `WebDriver`, przejdzie do witryny i wykona kilka podstawowych działań i weryfikacji.

Jak to zrobić?

Utwórzmy test, który wywoła różne metody nawigacji i sprawdza zachowanie aplikacji. Jego kod zamieszczono poniżej.

```
@Test
public void testNavigation() {
    driver.get("http://www.google.com");

    // Utworzenie reprezentacji pola tekstowego wyszukiwania
    WebElement searchField = driver.findElement(By.name("q"));
    searchField.clear();

    // Wprowadzenie słowa kluczowego do wyszukiwania i przesłanie ządania
    searchField.sendKeys("selenium webdriver");
    searchField.submit();
    WebElement resultLink = driver.findElement(By.linkText("Selenium WebDriver"));
    resultLink.click();
}
```

```

new WebDriverWait(driver, 10).until(ExpectedConditions
    .titleIs("Selenium WebDriver"));

assertEquals("Selenium WebDriver", driver.getTitle());

driver.navigate().back();

new WebDriverWait(driver, 10).until(ExpectedConditions
    .titleIs("selenium webdriver - Szukaj w Google"));

assertEquals("selenium webdriver - Szukaj w Google", driver.getTitle());

driver.navigate().forward();

new WebDriverWait(driver, 10).until(ExpectedConditions
    .titleIs("Selenium WebDriver"));

assertEquals("Selenium WebDriver", driver.getTitle());

driver.navigate().refresh();

new WebDriverWait(driver, 10).until(ExpectedConditions
    .titleIs("Selenium WebDriver"));

assertEquals("Selenium WebDriver", driver.getTitle());
}

```

Jak to działa?

Interfejs `WebDriver.Navigation` dostarcza metody `back()` i `forward()` do ładowania stron z historii przeglądarki. Metody te reprezentują przyciski *Wstecz* i *Dalej* dostępne w dowolnej przeglądarce WWW. Możemy również odświeżyć lub ponownie załadować stronę poprzez wywołanie metody `refresh()`.

Korzystanie ze zdarzeń frameworka WebDriver

Selenium WebDriver zawiera klasę `EventFiringWebDriver`, która nasłuchuje różnych zdarzeń zachodzących podczas wykonywania testu. Przykładowo zdarzenia są wyzwalane przy wejściu na stronę, gdy zostanie wykonane kliknięcie elementu albo kiedy zostanie zmieniona wartość. W poniższej tabeli zestawiono listę wszystkich zdarzeń, które możemy śledzić podczas wykonywania testu.

Zdarzenie	Opis
BeforeNavigateTo	Metoda jest wywoływana przed wywołaniem metod <code>get(String url)</code> lub <code>navigate().to(String url)</code> .
afterNavigateTo	Metoda jest wywoływana po wywołaniu metod <code>get(String url)</code> lub <code>navigate().to(String url)</code> .
beforeNavigateBack	Metoda jest wywoływana przed wywołaniem metody <code>navigate().back()</code> .
afterNavigateBack	Metoda jest wywoływana po wywołaniu metody <code>navigate().back()</code> .
beforeNavigateFor	Metoda jest wywoływana przed wywołaniem metody <code>navigate().forward()</code> .
afterNavigateForward	Metoda jest wywoływana po wywołaniu metody <code>navigate().forward()</code> .
beforeFindBy	Metoda jest wywoływana przed wywołaniem następujących metod: <code>WebDriver.findElement(...)</code> , <code>WebDriver.findElements(...)</code> , <code>WebElement.findElement(...)</code> , <code>WebElement.findElements(...)</code> .
afterFindBy	Metoda jest wywoływana po wywołaniu następujących metod: <code>WebDriver.findElement(...)</code> , <code>WebDriver.findElements(...)</code> , <code>WebElement.findElement(...)</code> , <code>WebElement.findElements(...)</code> .
beforeChangeValueOf	Metoda jest wywoływana przed wywołaniem metod <code>WebElement.clear()</code> lub <code>WebElement.sendKeys(...)</code> .
afterChangeValueOf	Metoda jest wywoływana po wywołaniu metod <code>WebElement.clear()</code> lub <code>WebElement.sendKeys(...)</code> .
beforeClickOn	Metoda jest wywoływana przed wywołaniem metody <code>WebElement.click()</code> .
afterClickOn	Metoda jest wywoływana po wywołaniu metody <code>WebElement.click()</code> .
beforeScript	Metoda jest wywoływana przed metodą <code>RemoteWebDriver.executeScript(java.lang.String, java.lang.Object[])</code> .
afterScript	Metoda jest wywoływana po metodzie <code>RemoteWebDriver.executeScript(java.lang.String, java.lang.Object[])</code> .
onException	Metoda ta jest wywoływana zawsze wtedy, gdy ma być zgłoszony wyjątek.

Wymienionych procedur obsługi zdarzeń można użyć do wykonania dodatkowych poleceń. Przykładowo przed wprowadzeniem wartości w polu tekstowym możemy wyczyścić istniejącą wartość lub przechwycić rzut ekranu nawet wtedy, gdy `WebDriver` zgłosi wyjątek. Można to zrobić, wykonując następujące czynności.

- Utwórz własną klasę nasłuchiwania zdarzeń. Do tej klasy możesz dodać kod, który zostanie uruchomiony po zgłoszeniu specyficznych zdarzeń.

- Za pomocą egzemplarza WebDriver utwórz egzemplarz klasy EventFiringWebDriver.
- Zarejestruj klasę słuchacza zdarzeń w egzemplarzu klasy EventFiringWebDriver.

Klasę słuchacza zdarzeń można utworzyć na dwa sposoby:

- przez zaimplementowanie interfejsu WebDriverEventListener,
- przez rozszerzenie klasy AbstractWebDriverEventListener.

W tej recepturze pokażę, w jaki sposób używać klasy EventFiringWebDriver do nasłuchiwania zdarzeń WebDriver.

Przygotuj się

Utwórz nowy test, który zbuduje egzemplarz WebDriver, przejdzie do witryny i wykona kilka podstawowych działań oraz weryfikacji.

Jak to zrobić?

Na początek zdefiniujemy klasę nasłuchiwania zdarzeń poprzez zaimplementowanie interfejsu WebDriverEventListener w następujący sposób.

```
package com.secookbook.examples.chapter04;

import org.openqa.selenium.*;
import org.openqa.selenium.support.events.WebDriverEventListener;

public class MyListener implements WebDriverEventListener {
    public void beforeChangeValueOf(WebElement element, WebDriver
driver) {
        element.clear();
    }
}
```

Następnie utworzymy test, który używa klasy EventFiringWebDriver.

```
package com.secookbook.examples.chapter04;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.support.events.EventFiringWebDriver;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;

public class EventFiringTest {
    private WebDriver driver;
```

```

@Before
public void setUp() throws Exception {
    driver = new FirefoxDriver();
}

@Test
public void testEventFiringWebDriver() throws Exception {

    EventFiringWebDriver eventDriver =
        new EventFiringWebDriver(driver);
    MyListener myListener = new MyListener();
    eventDriver.register(myListener);
    eventDriver.get("http://bit.ly/1DbdhsW");
    eventDriver.findElement(By.id("q"))
        .sendKeys("Selenium i testowanie aplikacji. Receptury");
}

@After
public void tearDown() throws Exception {
    driver.quit();
}
}

```

Co dalej?

Dodajemy do klasy odbiornika zdarzeń jedną dodatkową procedurę obsługi zdarzenia, która przechwytuje zrzut ekranu w momencie zgłaszania wyjątku.

```

public void onException(Throwable exception, WebDriver driver) {
    try {
        if
            (driver.getClass().getName().equals("org.openqa.selenium.remote.Re
            moteWebDriver")) {
                driver = new Augmenter().augment(driver);
            }
            File scrFile = ((TakesScreenshot)
            driver).getScreenshotAs(OutputType.FILE);
            FileUtils.copyFile(scrFile, new
            File("target/screenshots/error.png"));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```


Skorowidz

A

ADB, 326
adnotacja
 @CacheLookup, 186
 @DataProvider, 160
 @FindBy, 186
 @Test, 161
 @When, 253
 FindsBy, 199
adres
 docelowo, 59
 URL, 47
AJAX, 186, 229
alert, 133, 135
 confirm, 136
 JavaScript, 134
Android Debug Bridge, 326
Ant, 26, 27, 271, 272, 273, 279,
 281
 konfiguracja, 29, 281
Apache Ant, *Patrz:* Ant
Apache Maven, *Patrz:* Maven
API Advanced User Interactions,
 104
API Apache POI, 165
aplikacja mobilna, 319, 323, 325
Appium, 319, 320
Apple Instruments, 322
argument
 –hubHost, 302
 –role, 302

atrybut, 65
 class, 57, 61
 Class, 71
 DataSource, 174
 handle, 147, 148, 149
 id, 56, 57, 61, 71, 140
 innerText, 75, 85
 name, 57, 61, 72, 140, 146
 readonly, 86
 TestCaseSource, 170
 textContent, 75, 85
 title, 148, 150
 value, 189
 wartość częściowa, 73
 wyszukiwanie, 66, 71, 72
AutoIt, 286, 287, 290, 291

B

baner, 139
BDD, 245, 246
 w środowisku .NET, 255
Behave, 246, 267, 269
Behavior Driven Development,
 Patrz: BDD
biblioteka, 18
 Appium, 323
 Appium Java Client, 325
 AutoItX, 291
 jQuery, 78, 80, 129, 210
 jQuery Mobile, 320
 kliencka, 33, 34, 36

OpenCSV, 162
Pythona, 180
SpecFlow.NET, 255, 256
unittest, 180
BrowserStack, 313

C

Capybara, 246, 264, 265
CDN, 80
chmura, 297, 313
 Sauce Labs, *Patrz:* Sauce Labs
ChromeDriver, 40, 42
CI, 26
Content Delivery Network,
 Patrz: CDN
CSS, 47, 70
Cucumber-JVM, 246, 248, 253,
 264
 konfiguracja, 254

D

dekorator @ddt, 180
dokument XML drzewo, 62
DOM, 48, 54, 63, 68
 korzeń, *Patrz:* korzeń
 odpytywanie, 124
 węzeł, *Patrz:* węzeł
drzewo modelu DOM,
 Patrz: DOM
dziecko, 64

E

Eclipse, 18, 25
 konfigurowanie z Maven, 19
 element
 canvas, 237
 checkbox, 97
 dwukrotne kliknięcie, 107
 form, 83, 84
 HTML, 206
 input, 83
 li, 110
 nadrzędny, 62
 potomny, 58, 62
 przeciągnij i upuść, 108
 select, 88, 206
 sprawdzanie stanu, 103
 table, 99
 td, 99, 209
 tr, 99, 209
 video, 234, 236
 etykieta tekstowa, 92
 Excel, 164, 172

F

Firebug, 49, 51
 Firefox, 36, 75
 narzędzia programistyczne, 49
 Flash, 286, 296
 formularz, 83
 funkcja, *Patrz też*: metoda
 ControlClick, 290
 ControlSetText, 290
 DataProvider, 159
 normalize-space, 75
 text, 74
 UI Automation, 322
 WinWaitActive, 290
 XPath, 66

G

gem
 Cucumber, 265
 DeepTest, 299
 page-object, 203, 204
 Roo, 175, 178

generator akcji, 239
 Google Chrome narzędzia
 programistyczne, 51
 Google Docs, 178
 grupa opcji, 95, 96, 97
 GUI, 293

H

historijka użytkownika, 246,
 264, 267
 HTML, 47, 48, 49
 HTML5, 233, 237, 239

I

IDE, 18
 IntelliJ IDEA, 18
 interfejs
 Alert, 134, 135, 139
 IWebDriver, 199
 Java Image, 223
 JavascriptExecutor, 111,
 112
 jQuery API, 79
 localStorage, 239
 Navigation Timing, 230
 Selenium API, 325
 sessionStorage, 241, 243
 TakesScreenshot, 113, 114,
 223, 224
 Timeouts, 125
 timing, 230
 użytkownika, 133
 graficzny, *Patrz*: GUI
 natywny, 286
 niewebowy, 293
 WebDriver.Navigation, 117
 WebDriver.Options, 116
 WebDriver.TargetLocator,
 140, 142
 WebDriverEventListener,
 120
 WebElement, 58, 82, 83, 95,
 206, 214, 216
 Window, 115
 Internet Explorer Driver
 Server, 36, 38

InternetExplorerDriver, 36

J

JavaScript, 111, 129
 Jenkins, 26, 272, 274, 281
 Build Trigger, 278
 kompilacja, 278
 konfiguracja, 281
 środowisko rozproszonego
 budowania i testowania,
 274
 wyświetlanie wyników, 278
 język
 Gherkin, 246, 247
 Python, *Patrz*: Python
 Ruby, *Patrz*: Ruby
 XPath, *Patrz*: XPath
 jQuery, 210
 jQuery File Upload, 287
 JUnit, 22, 155, 254

K

klasa
 AbstractWebDriverEvent
 ↳Listener, 120
 Actions, 105, 108, 109
 Augmenter, 114
 BufferedReader, 164
 By, 55, 60, 61, 99
 strategia lokalizacji, 55
 Canvas, 237
 CompareUtil, 225, 226, 228
 DataProvider, 159
 DesiredCapabilities, 301
 driver, 115
 EdgeDriver, 43
 EventFiringWebDriver,
 118, 120
 ExpectedCondition, 127
 implementacja
 niestandardowa, 128
 ExpectedConditions, 125,
 127, 136
 FirefoxDriver, 115
 FluentWait, 130
 GoogleSearchTest.java, 23

- InternetExplorerDriver, 39
 - IOSDriver, 325
 - Java, 215
 - JavascriptExecutor, 79, 217, 234, 236, 243
 - jQueryUITab, 210, 212, 213
 - Keys, 106
 - LoadableComponent, 189, 190
 - ObjectMap, 218, 219
 - OutputType, 114
 - PageFactory, 183, 185
 - PhantomJSDriver, 318
 - PixelGrabber, 226
 - Properties, 221
 - RemoteWebDriver, 40, 301
 - RunCukesTest, 254
 - Select, 87, 88, 90, 92, 94, 238
 - SimpleDDT, 157
 - SpreadsheetData, 165, 167
 - uruchamiająca, 155
 - WebDriverWait, 125, 126, 136
 - WebElement, 54, 94
 - WebElementExtender, 215, 216, 239
 - WebTable, 209
 - WebTable.java, 206
 - WrapsDriver, 224
 - zagnieżdżanie, 193
 - klawiatura, 106
 - klawisz skrótu, 110
 - klucz clickCount, 243
 - kod
 - HTML, *Patrz:* HTML kod
 - JavaScript, 111, 112
 - kombinator rodzeństwa, 76
 - koncentrator, 299, 300, 301
 - Sauce Labs, 315
 - kontrolka, 295
 - Flash, 286
 - Flex, 286
 - jQuery, 205
 - listy, 88
 - menu, 87
 - Silverlight, 286
 - korzeń, 62
 - kwerenda XPath,
 - Patrz:* wyrażenie XPath
- ## L
- link, 111
 - adres docelowy, *Patrz:* adres docelowy
 - wyszukiwanie, 60
 - lista
 - jednokrotnego wyboru, 94
 - rozwijana, 90
 - wartość oczekiwana, 92
 - wielokrotnego wyboru, 88, 89, 94
- ## M
- magazyn
 - lokalny, 241, 243
 - sesji, 241, 243
 - mapa obiektów, 217, 218
 - bazująca na XML, 221
 - Maven, 18, 25, 26, 271, 272, 274, 323
 - menedżer pakietów NuGet, *Patrz:* NuGet
 - menu, 90
 - Drawing Tool, 238
 - kontekstowe, 109, 110
 - nawigacji, 139
 - podręczne, *Patrz:* menu kontekstowe
 - rozwijane, 87, 88, 89, 92, 94
 - metoda, *Patrz też:* funkcja
 - accept, 136
 - addCookie, 115
 - alert, 134
 - alertIsPresent, 136
 - back, 117, 118
 - captureElementBitmap, 239
 - captureElementPicture, 223, 224
 - clear, 82, 83
 - click, 82, 83, 84, 95, 96, 106
 - CompareImage, 226
 - contextClick, 109, 110
 - copyFile, 113
 - cssSelector, 70, 213
 - deleteAllCookies, 115
 - deleteCookie, 115
 - deleteCookieNamed, 115
 - deselectByIndex, 90
 - deselectByVisibleText, 89
 - dismiss, 138, 139
 - doubleClick, 107
 - dragAndDrop, 108
 - driver.close, 148
 - driver.getPageSource, 152
 - driver.getTitle, 111
 - driver.getWindowHandles, 149, 150, 151
 - driver.switchTo.alert, 135, 136
 - driver.switchTo.default
 - ↪Content, 142
 - driver.switchTo.frame, 140, 142, 145
 - driver.switchTo.window, 147
 - executeScript, 112, 236, 243
 - ExpectedConditions.title
 - ↪Contains, 126
 - find, 79
 - findElement, 54, 56, 58, 103, 125, 128, 145
 - findElements, 54, 58, 59, 100, 213
 - forward, 117, 118
 - get, 119
 - getAllSelectedOptions, 94
 - getAttribute, 85, 86
 - getCellData, 209
 - getCellEditor, 209
 - getCookieNamed, 115, 116
 - getCookies, 115
 - getCSSValue, 86, 87
 - getFirstSelectedOption, 94
 - getLocator, 221
 - GetLocator, 222
 - getOptions, 89, 92
 - getProducts, 197
 - getProperty, 221
 - getScreenshotAs, 113, 114, 224
 - getText, 84, 85, 94, 135, 209
 - Google.new, 178
 - highlightElement, 216, 217
 - highlightElements, 216

- metoda
 - implicitlyWait, 125
 - initElements, 185, 186
 - InitElements, 199
 - injectQuery, 80
 - injectQueryIfNeeded, 80
 - isDisplayed, 103
 - isElementPresent, 102, 103
 - isEnabled, 103, 104
 - isMultiple, 88
 - isSelected, 97, 98, 103
 - jQueryLoaded, 80
 - keyDown, 106
 - keyUp, 106
 - linkText, 60, 61
 - moveByOffset, 239
 - name, 72
 - navigate, 119
 - nth-child, 76
 - partialLinkText, 60, 61
 - pause, 236
 - play, 236
 - przesłanie, 192
 - refresh, 117, 118
 - RemoteWebDriver.execute
 - Script, 119
 - searchInStore, 197
 - selectByIndex, 90
 - selectByValue, 89, 238
 - selectByVisible, 94
 - selectByVisibleText, 89
 - sendKeys, 82, 83, 139, 214
 - setAttribute, 215
 - setBrowserName, 301
 - setPlatform, 301
 - setVersion, 301
 - size, 209
 - submit, 82, 83, 84
 - tagName, 61, 62, 209
 - TestContext, 174
 - testData, 164, 167
 - testDropdown, 91, 93
 - to, 117
 - wait.until, 127
 - WebDriver.findElement, 119
 - WebDriver.findElements, 119
 - WebElement.clear, 119
 - WebElement.click, 119
 - WebElement.findElement, 119
 - WebElement.findElements, 119
 - WebElement.sendKeys, 119
 - windowMaximize, 115
 - xpath, 65, 69
 - Microsoft Edge, 43
 - Microsoft Internet Explorer
 - narzędzia programistyczne, 53
 - Microsoft WebDriver Server, 43, 45
 - model
 - COM, 291
 - DOM, *Patrz:* DOM
 - moduł
 - DDT, 179, 180
 - nose, 299, 312
 - MSTEST, 171, 172, 174, 299
 - mysz, 106, 237
 - dwukrotne kliknięcie, 106, 107
 - przyciski prawy, 109
- ## N
- narzędzie ciągłej integracji, *Patrz:* CI
 - NetBeans, 18
 - NuGet, 29, 32
 - NUnit, 167, 168
 - NUnit Test Adaptor for Visual Studio, 263
- ## O
- obiekt
 - Cookie, 116
 - DataTable, 174
 - mapa, *Patrz:* mapa obiektów
 - Page, 260
 - płynnego oczekiwania, 131
 - WebElement, 206, 209
 - window.performance, 230, 231
 - oczekiwanie
 - bazujące na klasie FluentWait, 130
 - jawne, 125, 127
 - na aktualizację wartości atrybutu elementu, 128
 - na widoczność elementu, 129
 - na zdarzenia w modelu DOM, 129
 - niejawne, 124, 125
 - niestandardowe, 127
 - odtwarzacz wideo, 234, 236
 - okno
 - confirm, 136, 137, 139
 - identyfikacja, 146
 - modalne, 134
 - podrzędne, 139
 - potomne, 139, 146, 147
 - prompt, 137, 138, 139
 - przeglądarki
 - maksymalizowanie, 115
 - wyskakujące, 133, 150
 - zamykanie, 148
 - operator
 - +, 77
 - >, 76
 - unii, 68
 - oprogramowanie
 - sterowane testami, *Patrz:* TDD
 - sterowane zachowaniem, *Patrz:* BDD
 - oś XPath, 68
- ## P
- Page Object, 181, 185, 189
 - implementacja
 - w Pythonie, 200, 202
 - w Ruby, 203
 - w środowisku .NET, 197
 - zagnieżdżanie, 192
 - PhantomJS, 316, 318
 - plik
 - .xls, 165
 - .xlsx, 165
 - build.xml, 27, 29, 281
 - conf.xml, 310
 - cookie, 115, 116, 239
 - CSV, 162, 164
 - definicji kroku, 259, 263, 264, 268

OpenDialogHandler.exe, 290
pom.xml, 18, 21, 25, 316
POM.xml, 323
przesyłanie na serwer, 287
specyfikacji, 258
wideo, 236

pole

- tekstowe, 82
- wyboru, 103
 - lokalizowanie, 98
 - zaznaczanie, 98

polecenie clean, 26

porównywanie obrazów, 224, 226, 227

potomek, 64, 69

- wyszukiwanie, 76, 77

predykat, 65, 67

protokół

- JSON, 320, 322, 326
- JSON WebDriver, 38

przeglądarka

- historia, 118
- nawigacja, 117
- okno, 115
- przechowywane dane, 239

przełącznik, 95, 96

przestrzeń nazw java.awt.Image, 226

przodek, 64, 69

przycisk, 82, 83, 144

pseudoklasa, 76, 77

- :active, 77
- :checked, 78
- :disabled, 78
- :enabled, 78
- :focus, 77
- :hover, 77
- contains, 75

PUnit, 299

Python, 32, 179, 180, 200, 202, 267, 299, 310, 312

R

ramka

- HTML, 139
- inline, 144, 145

refaktoryzacja, 187, 189

rodzeństwo, 64, 69

- kombinator, *Patrz:*
 - kombinator rodzeństwa
 - wyszukiwanie, 77

rodzic, 64

Roo, 175, 177, 178

RSpec, 264

Ruby, 32, 175, 203, 264, 299

runner, *Patrz:* klasa uruchamiająca

S

Sauce Connect, 315

Sauce Labs, 313

scenariusz, 253

selektor

- CSS, 51, 52, 70, 74, 76, 100, 186
- jQuery, 78
- or, 73
- XPath, 51, 52, 100, 186

Selenium, 205, 224, 264, 271, 286, 297

Selenium Grid, 298

Selenium Grid Server, 299, 300

Selenium Hub, 301

- węzeł, 301, 302, 303, 304

Selenium Server, 299

Selenium WebDriver, 17

serwer

- ciągłej integracji, 272
- HTTP, 322, 326

Sikuli, 293, 295

Silverlight, 286, 296

sluchacz zdarzeń, 120

SpecFlow.NET, 246

statystyka czasowa, 230

Ś

ścieżka, 64

- bezwzględna, 65
- względna, 65, 71

środowisko zintegrowane tworzenia oprogramowania, *Patrz:* IDE

T

tabela, 99, 209

- webowa, 205
- wyświetlanie danych, 99

tablica, 164

tag

- frame, 139
- frameset, 139
- iframe, 144

tag Anchor, 111

TDD, 245

tekstu wyszukiwanie, 74, 75

test, 25

- akceptacyjny, 245
- automatyzacja, 123, 181, 297
- bazujący na danych, 154, 155, 159, 167, 171, 174, 175, 177, 179
- BDD/ATDD, 248
- funkcjonalny, 316
- jednostkowy, 304
- na platformie .NET, 29, 197
- NUnit, 263
- synchronizacja, 124, 125, 127
- tworzenie Ruby, 34
- uruchamianie
 - automatyczne, 284
 - CI, 26
 - z wiersza polecenia, 26
- w chmurze, 313
- w środowisku ciągłej integracji, 281
- wykonywanie równoległe, 298, 299, 304, 305, 306, 308, 310, 312

Test Driven Development, *Patrz:* TDD

TestNG, 22, 155, 159, 160, 299, 304, 306, 309

testowanie sterowane danymi, 153, 154, 155, 159, 167, 171, 174, 175, 177, 179

Twitter, 144

U

uwierzytelnianie, 291, 293

W

Web storage, 239
 węzeł, 62, 64, 69
 widżet, 210, 213, 214
 Wiki Confluence, 178
 WinAnt, 27, 279
 wirtualizacja, 297
 wydajność, 229, 297
 wyjątek, 119

- ElementNotVisibleException, 84
- NoAlertPresentException, 136
- NoSuchElementException, 54, 58, 103, 130
- NoSuchWindowException, 148

 wyrażenie XPath, 62, 65
 wzorzec Page Object, *Patrz:* Page Object

X

XPath, 62, 68

- oś, *Patrz:* oś XPath
- wyrażenie, *Patrz:* wyrażenie XPath

Z

zdarzenie, 118, 128

- afterChangeValueOf, 119
- afterClickOn, 119
- afterFindBy, 119
- afterNavigateBack, 119
- afterNavigateForward, 119
- afterNavigateTo, 119
- afterScript, 119
- beforeChangeValueOf, 119
- beforeClickOn, 119
- beforeFindBy, 119
- beforeNavigateBack, 119
- beforeNavigateFor, 119
- BeforeNavigateTo, 119
- beforeScript, 119
- onException, 119
- onsubmit, 84
- sluchacz, 120

 znacznik, 61

- frameset, 139, 144
- iframe, 139, 144

 znak

- \$=, 73
- *, 68
- *=, 73
- „, 73

., 64, 75
 .., 64
 /, 64
 //, 64
 @, 65, 68
 [], 65
 ^ =, 73
 |, 68
 +, 77
 >, 76
 kropka, *Patrz:* znak .
 podwójna kropka,
 Patrz: znak ..
 podwójny ukośnik,
 Patrz: znak //
 ukośnik, *Patrz:* znak /
 zrzut ekranu, 113, 114, 223

Ż

żądanie, 47

PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION



- 1. ZAREJESTRUJ SIĘ**
- 2. PREZENTUJ KSIĄŻKI**
- 3. ZBIERAJ PROWIZJĘ**

Zmień swoją stronę WWW
w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

<http://program-partnerski.helion.pl>

Selenium i testowanie aplikacji

Receptury. Wydanie II

Projektanci nowoczesnych aplikacji internetowych muszą obecnie sprostać wielu wyzwaniom. Oprogramowanie, które tworzą, musi działać wydajnie i bezbłędnie w różnych przeglądarkach, na różnych platformach, na różnych urządzeniach. Wobec rosnących wymagań testowanie aplikacji staje się bardzo ważnym etapem jej rozwoju. Jako że zaprojektowanie i przeprowadzenie takich testów jest dość złożonym zadaniem, warto zapewnić sobie sprawdzone narzędzie umożliwiające automatyzację testów aplikacji. Takim narzędziem z pewnością jest Selenium Web Driver — framework służący do automatyzacji przeglądarek internetowych.

W tej książce przedstawiono zaawansowane techniki testowania aplikacji internetowych za pomocą Selenium Web Driver i narzędzi pokrewnych. Zaprezentowano skuteczne i efektywne metodyki testowania aplikacji przeznaczonych dla komputerów stacjonarnych, przeglądarek mobilnych i działających w środowisku rozproszonym. Opisano tu również wzorce projektowe, takie jak testy bazujące na danych, obiekty stron i mapy obiektów. Ponadto omówiono techniki rozszerzania frameworka Selenium i dostosowywania go do szczególnych działań. W książce znalazło się ponad 80 receptur, które z pewnością okażą się przydatne podczas automatyzacji testów aplikacji.

Nowoczesne testowanie: Selenium Web Driver!



W książce znajdziesz:

- konfigurację Selenium i narzędzi współpracujących
- techniki i strategie lokalizacji
- stosowanie Selenium do tworzenia i synchronizacji testów
- techniki Behavior-Driven Development (BDD)
- testowanie aplikacji w wielu różnych przeglądarkach, w tym w środowiskach mobilnych

Unmesh Gundecha od kilkunastu lat specjalizuje się w rozwijaniu i testowaniu oprogramowania. Brał udział w licznych projektach automatyzacji testów funkcjonalnych przy użyciu standardowych i niestandardowych frameworków automatyzacji testów w połączeniu z wiodącymi narzędziami. Obecnie jest architektem testów w międzynarodowej firmie w Pune w Indiach.

[PACKT] open source
PUBLISHING community experience distilled

Helion

księgarnia internetowa



<http://helion.pl>

zamówienia telefoniczne



0 801 339900



0 601 339900

Informatyka w najlepszym wydaniu

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
<http://helion.pl>

Sprawdź najnowsze promocje:
● <http://helion.pl/promocje>
Książki najchętniej czytane:
● <http://helion.pl/bestsellery>
Zamów informacje o nowościach:
● <http://helion.pl/novosci>

sięgnij po WIĘCEJ



KOD KORZYSCI

ISBN 978-83-283-3411-3



9 788328 334113

cena: 59,00 zł