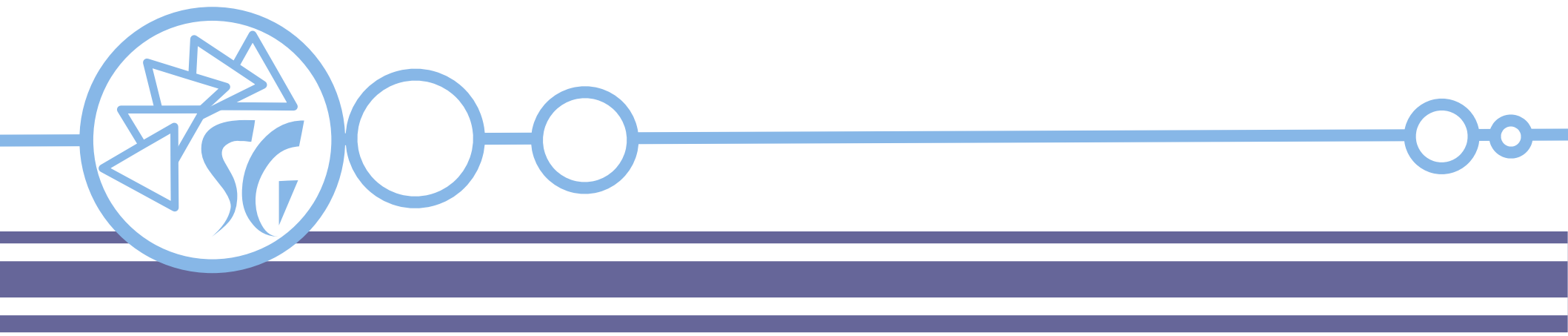

Selenium Web Driver

Ing. Simone Giustetti

www.giustetti.net

Selenium Web Driver

Selenium Web Driver pilota un browser installato localmente o su macchina remota.

Pensato per sviluppatori.

Consiste di una API ad oggetti, + bindings per il linguaggio di programmazione, che esegue comandi sul browser.

Installare Selenium Web Driver

Installare una libreria per legare Selenium al linguaggio di programmazione utilizzato.

La procedura di installazione dipende dal linguaggio di programmazione.

Installare Web Driver per JavaScript

Web Driver per JavaScript si appoggia a **NodeJS**.

Web Driver è distribuito sotto forma di pacchetto per NodeJS.

```
npm install selenium-webdriver
```

Web Driver deve essere incluso nelle dipendenze del progetto aggiungendo al file package.json:

```
"selenium-webdriver": "^4.11.1"
```

Creare un Nuovo Progetto JavaScript

- Creare una nuova cartella per il progetto.
- Aprire la finestra del terminale e spostarsi nella cartella appena creata.
- Inizializzare il nuovo progetto:

  npm init

- Accettare tute le opzioni predefinite premendo il tasto “Invio”. Verrà creato il file *package.json*.
- Aggiungere in coda come dipendenza la versione di Selenium Web Driver.

Il File package.json

```
{
   "name": "demoseleniumproject",
   "version": "1.0.0",
   "description": "",
   "main": "index.js",
   "scripts":  {
      "test": "echo \"Error: no test specified\" && exit
1"
   },
   "author": "",
   "license": "ISC",
   "dependencies": {
      "selenium-webdriver": "^4.11.1"
   }
}
```

Importare le Librerie di Collegamento

Nel file JavaScript del test aggiungere le righe:

```
// Import the driver
const { Builder, By, until } = require( "selenium-
webdriver" );
...

// Declare the test method
async function loginTest()  {
   // Launch the browser
   let driver = await new
      Builder().forBrowser( "firefox" ).build();
   ...
}
```

Eseguire un Test in JavaScript

Per lanciare ed eseguire un test si fa ricorso al run-time di NodeJS:

```
node <test>.js
```

NodeJS si occupa di copiare nella cartella del test tutti i prerequisiti, creare il pacchetto finito e, infine, eseguire il test.

Codici di Ritorno di un Test

Al termine dell'esecuzione, NodeJS rende 2 informazioni:

- Il tempo richiesto per eseguire il test;
- Un codice di ritorno.

Il codice di ritorno può assumere 2 valori:

- **0**: Test eseguito con successo;
- **1**: Errore.

Identificare un Elemento di una Pagina

Esistono 8 modi (locator) per identificare un elemento di una pagina web in Selenium:

- Un selettore CSS.
- L'attributo **class** di un elemento.
- L'attributo **id** di un elemento.
- L'attributo **name** di un elemento.
- Il testo visibile di un elemento.
- Parte del testo visibile di un elemento.
- Il nome di un marcatore (tag) HTML.
- Una espressione xpath.

Selettore CSS

```
// Launch the browser
let driver = await new
    Builder().forBrowser( 'firefox' ).build();

...

// Select element through a CSS selector
const loc = await
    driver.findElement( By.css( '#fname' ));

...
```

Nome di una Classe

```
// Launch the browser
let driver = await new
    Builder().forBrowser( 'firefox' ).build();

...

// Select element by class name
const loc = await
    driver.findElement( By.className( 'information' ));

...
```

Attributo id di un Elemento

```
// Launch the browser
let driver = await new
    Builder().forBrowser( 'firefox' ).build();

...

// Select element by id
const loc = await
    driver.findElement( By.id( 'username' ));

...
```

Attributo name di un Elemento

```
// Launch the browser
let driver = await new
    Builder().forBrowser( 'firefox' ).build();

...

// Select element by name
const loc = await
    driver.findElement( By.name( 'password' ));

...
```

Testo di un link

```
// Launch the browser
let driver = await new
    Builder().forBrowser( 'firefox' ).build();

...

// Select link by its displayed text
const loc = await
    driver.findElement( By.linkText( 'Homepage' ));

...
```

Testo Parziale di un link

```
// Launch the browser
let driver = await new
   Builder().forBrowser( 'firefox' ).build();

...

// Select link by its displayed text
const loc = await
   driver.findElement( By.partialLinkText( 'Official
      Page' ));

...
```

Viene selezionato il primo elemento nel caso esistano più candidati.

Il Nome di un Marcatore HTML

```
// Launch the browser
let driver = await new
    Builder().forBrowser( 'firefox' ).build();

...

// Select element by tag name
const loc = await
    driver.findElement( By.tagName( 'a' ));

...
```

Deprecato in Selenium 4. Utilizzare un selettore CSS.

Xpath

```
// Launch the browser
let driver = await new
   Builder().forBrowser( 'firefox' ).build();


...
// Select element by absolute xpath
const loc01 = await
   driver.findElement( By.xPath(
      '/html/form/input[1]' ));
// Select element by relative xpath
const loc02 = await
   driver.findElement( By.xPath(
      '//input[@value='f']' ));
const loc02 = await
   driver.findElement( By.xPath(
      '//input[@name='cognome']' ));
```

Relative Locator

Un relative locator consente di identificare un elemento in base alla posizione relativa ad un altro.

```
let emailLocator =
   locateWith(By.tagName( 'input' )).above(By.id( 'password' ));
let pwdLocator =
   locateWith(By.tagName( 'input' )).below(By.id( 'email' ));
let cancelLocator =
   locateWith(By.tagName( 'button' )).toLeftOf(By.id( 'submit' ));
let submitLocator =
  locateWith(By.tagName( 'button' )).toRightOf(By.id( 'cancel' ));
// 50px radius
let emailLocator =
   locateWith(By.tagName( 'input' )).near(By.id( 'lbl-email' ));
// Multiple relative locators
let submitLocator =
   locateWith(By.tagName( 'button' )).below(By.id( 'email'
   )).toRightOf(By.id( 'cancel' ));
```

Trovare un Elemento in una Pagina

Il metodo **findElement** unito ad un locator consente di trovare ed identificare un elemento di una pagina HTML.

Quando esistono più elementi identificabili attraverso il locator, findElement punta sempre al primo trovato.

Trovare un Elemento di un Sottoinsieme

Il metodo **findElement** può essere usato per trovare un elemento in un insieme di elementi.

```
// Select a subset of page elements
const fruits =
   await driver.findElement(By.id( 'fruits' ));
// Select an element from the subset
const fruit =
   fruits.findElement(By.className( 'tomatoes' ));
```

Iterare tra gli Elementi di un Sottoinsieme

Il metodo **findElements** unito ad un locator consente di identificare un insieme di elementi di una pagina HTML.

Dato un insieme di elementi è necessario iterare per far riferimento ad uno specifico.

```
// Get all the elements available with tag 'p'
let elements =
    await driver.findElements(By.css( 'p' ));
// Cycle all elements of the set
for( let e of elements )  {
    console.log( await e.getText());
}
```

Iterare tra i Figli di un Elemento

Dato un elemento, iterando, si può trovare il riferimento ad un suo figlio specifico.

```
// Get the 1st <div> element of a page
let element =
   await driver.findElement(By.css( 'div' ));
// Get all child paragraphs of the 1st <div>
let elements =
   await element.findElements(By.css( 'p' ));
for( let e of elements )  {
   console.log( await e.getText());
}
```

Localizzare l’Elemento Attivo

Si dice “attivo” l’elemento di una pagina HTML che al momento della ricerca ha il focus.

Il metodo **activeElement** consente di ottenere il riferimento all’elemento attivo.

```
// Get title of current active element
let attr =
    await driver.switchTo().activeElement().
    getAttribute( 'title' );
console.log( `${attr}` );
```

Le Azioni

Le azioni consentono di interagire con gli elementi di una pagina HTML simulando l'intervento umano.

Esistono 5 azioni:

- clear (Elementi contenti testo modificabile);
- click (Per ogni elemento);
- select (Per le liste);
- send keys (Elementi contenti testo modificabile);
- submit (Per le maschere);

Clear

Simula l'inizializzazione di un controllo che contenga testo. L'elemento viene pulito e portato al valore predefinito.

```
// Navigate to URL
await driver.get(
   'http://localhost/dev/presenze/index.php' ));
// Reinit the login field
await
   driver.findElement(By.name( 'username' )).clear();
```

Click

Simula il click del mouse sull'elemento selezionato.

```
// Navigate to URL
await driver.get( 'http://www.giustetti.net' ));
// Click the login link
driver.findElement(By.linkText( 'log in' )).click();
```

Send Keys

Simula la digitazione di testo nell'elemento selezionato.

```
// Navigate to URL
await driver.get(
    'http://localhost/dev/presenze/index.php' ));
// Fill-in the login and password fields
await
driver.findElement(By.name( 'username' )).sendKeys(
    'PAOLO' );
await
driver.findElement(By.name( 'password' )).sendKeys(
    'PAOLO' );
```

Submit

Simula l’invio dei dati contenuti in una maschera (Form HTML) al web server.

A partire da **Selenium 4**, il metodo non è più supportato. Utilizzare il metodo **click** sul pulsante della maschera per simulare l’invio dei dati.

Ottenere Informazioni dal Browser

È possibile ottenere informazioni circa la pagina visitata in modo da eseguire controlli e verifiche.

Il metodo **getCurrentUrl** rende l'indirizzo della pagina visitata.

Il metodo **getTitle** rende il titolo della pagina visitata.

Ottenere Informazioni sugli Elementi di Pagina

È possibile ottenere informazioni circa gli elementi che costituiscono una pagina web ed il loro stato.

Il metodo **isDisplayed** rende un valore booleano uguale a true se l'elemento interrogato è stato disegnato dal browser ed è visibile.

Il metodo **isEnabled** rende un valore booleano uguale a true se l'elemento interrogato è utilizzabile al momento del controllo.

Ottenere Informazioni sugli Elementi di Pagina

Il metodo **isSelected** rende un valore booleano uguale a true se l'elemento interrogato è selezionato al momento del controllo.

Il metodo **getTagName** rende una stringa uguale al valore del marcatore dell'elemento.

Il metodo **getRect** rende un oggetto contenente posizione e dimensioni dell'elemento. Il formato dell'oggetto è: Coord X, Coord Y, Altezza, Larghezza.

Ottenere Informazioni sugli Elementi di Pagina

Il metodo **getCssValue** rende il valore impostato per l'attributo CSS dell'elemento. La tipologia di dato resa varia in base all'attributo interrogato. Il nome dell'attributo deve essere passato come parametro al metodo.

Il metodo **getText** rende una stringa contenente il testo mostrato per l'elemento, ad esempio un collegamento.

Ottenere Informazioni sugli Elementi di Pagina

Il metodo **getAttribute** rende il valore impostato per l'attributo HTML dell'elemento. La tipologia di dato resa varia in base all'attributo interrogato. Il nome dell'attributo deve essere passato come parametro al metodo.

Simulare i Pulsanti di Navigazione

Esistono metodi che emulano la pressione dei pulsanti di navigazione del browser:

```
// Open URL
await driver.get( 'http://www.giustetti.net' );
// Go back one page
await driver.navigate().back();
// Go forward one page
await driver.navigate().forward();
// Refresh
await driver.navigate().refresh();
```

Best Practices

Usare il **locator più adatto**: Evita di aggiornare continuamente i test a seguito di modifiche agli elementi di una pagina. Usare i locator CSS per gli elementi creati dinamicamente.

Organizzare i test secondo una priorità. Testare le funzionalità critiche da subito.

Organizzare i test perché funzionino con combinazioni diverse di dati.

Best Practices

Separare gli script di test dai locator per facilitare la manutenzione dei test.

Incorporare istruzioni di attesa per gestire i tempi di caricamento delle pagine e simulare problemi dovuti ad un collegamento lento.

Documentare i test.

Informazioni & Licenze