



EBook Gratuito

APPENDIMENTO protractor

Free unaffiliated eBook created from
Stack Overflow contributors.

#protractor

Sommario

Di.....	1
Capitolo 1: Iniziare con il goniometro	2
Osservazioni.....	2
Versioni.....	2
Examples.....	2
Installazione e configurazione del goniometro (su Windows).....	2
Primo test usando il goniometro.....	3
Scrivi un test del goniometro.....	4
Test di corsa selettivi.....	5
Test in sospenso.....	6
combinazioni.....	6
Goniometro: test E2E per applicazioni angolari aziendali.....	6
Capitolo 2: Controllo del flusso e promesse	9
introduzione.....	9
Examples.....	9
Capire il flusso di controllo.....	9
Capitolo 3: Explicit attende con browser.wait ()	10
Examples.....	10
browser.sleep () vs browser.wait ().....	10
Capitolo 4: File di configurazione del goniometro	11
introduzione.....	11
Examples.....	11
Semplice file di configurazione - Chrome.....	11
File di configurazione con funzionalità - Chrome.....	11
file di configurazione shardTestFiles - Chrome.....	11
emule multi-capability del file di configurazione - chrome.....	12
Capitolo 5: Goniometro Debugger	14
Sintassi.....	14
Osservazioni.....	14
Examples.....	14

Utilizzo di browser.pause ()	14
Utilizzando browser.debugger ()	15
Capitolo 6: Individuazione degli elementi	17
introduzione	17
Parametri	17
Examples	17
Localizzatori specifici per goniometro (per applicazioni angolari)	17
Localizzatore vincolante	17
Esempio	17
Localizzatore di binding esatto	18
Esempio	18
Localizzatore di modelli	18
Esempio	18
Localizzatore di testo pulsante	18
Esempio	19
Localizzatore di testo pulsante parziale	19
Localizzatore di ripetitori	19
Esempio	20
Localizzatore ripetitore esatto	20
Esempio	20
Localizzatore di testo e CSS	20
Esempio	21
Opzioni di ricerca	21
Esempio	21
Localizzatore CSS profondo	21
Esempio	22
Nozioni di base sul localizzatore	22
Capitolo 7: Oggetti di pagina	24
introduzione	24
Examples	24
Primo oggetto della pagina	24

Capitolo 8: Selettori CSS	26
Sintassi.....	26
Parametri.....	26
Osservazioni.....	26
Examples.....	27
Scorciatoie di selezione selettore CSS \$ e \$\$.....	27
Introduzione ai locatori.....	28
Seleziona l'elemento con un valore dell'attributo HTML esatto.....	28
Seleziona l'elemento con un attributo HTML che contiene un valore specificato.....	28
Capitolo 9: Selettori XPath in goniometro	29
Examples.....	29
Selezione di un elemento DOM utilizzando il rapportatore.....	29
Selezione di elementi con attributi specifici.....	29
Per classe	29
Per id	30
Altri attributi	30
Capitolo 10: Test di app non angolari con Goniometro	32
introduzione.....	32
Examples.....	32
Modifiche necessarie per testare l'applicazione non angolare con Goniometro.....	32
Titoli di coda	33

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: [protractor](#)

It is an unofficial and free protractor ebook created for educational purposes. All the content is extracted from [Stack Overflow Documentation](#), which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official protractor.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capitolo 1: Iniziare con il goniometro

Osservazioni

Il **goniometro** è un framework di test end-to-end per le applicazioni AngularJS.

Il goniometro è un wrapper (costruito in alto) attorno a Selenium WebDriver, quindi contiene tutte le funzionalità disponibili nel Selenium WebDriver. Inoltre, il goniometro fornisce alcune nuove strategie e funzioni di localizzazione che sono molto utili per automatizzare l'applicazione AngularJS. Gli esempi includono cose come: `waitForAngular`, `By.binding`, `By.repeater`, `By.textarea`, `By.model`, `WebElement.all`, `WebElement.evaluate`, ecc.

Versioni

Versione	Rilascio dati
0.0.1	2016/08/01

Examples

Installazione e configurazione del goniometro (su Windows)

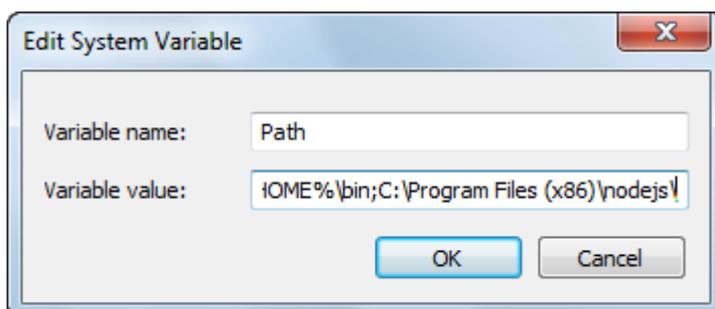
Requisiti: il goniometro richiede l'installazione delle seguenti dipendenze prima dell'installazione:

- Java JDK 1.7 o successivo
- Node.js v4 o versione successiva

Installazione:

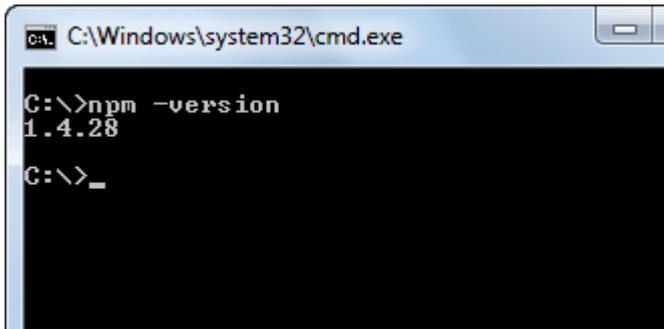
Scarica e installa Node.js da questo URL: <https://nodejs.org/en/>

Per vedere se l'installazione di Node.js ha esito positivo, puoi andare e controllare le variabili di ambiente. Il 'Percorso' sotto Variabili di sistema verrà automaticamente aggiornato.



Puoi anche controllare lo stesso digitando il comando `npm -version` nel prompt dei comandi che ti

darà la versione installata.



```
C:\Windows\system32\cmd.exe
C:\>npm -version
1.4.28
C:\>_
```

Ora il goniometro può essere installato in due modi: a livello locale o globale.

Possiamo installare il rapportatore in una cartella specificata o in un percorso di directory del progetto. Se installiamo in una directory di progetto, ogni volta che eseguiamo, dovremmo eseguire solo da quella posizione.

Per installare localmente nella directory del progetto, accedere alla cartella del progetto e digitare il comando

```
npm install protractor
```

Per installare Protractor esegui globalmente il comando:

```
$ npm install -g protractor
```

Questo installerà due strumenti da riga di comando, `protractor` e `webdriver-manager`.

Esegui il `protractor --version` per assicurarsi che il goniometro sia stato installato con successo.

`webdriver-manager` viene utilizzato per scaricare i file binari del driver del browser e avviare il server di selenio.

Scarica i binari del driver del browser con:

```
$ webdriver-manager update
```

Avviare il server selenio con:

```
$ webdriver-manager start
```

Per scaricare il driver di internet explorer, eseguire il comando `webdriver-manager update --ie` nel prompt dei comandi. Questo scaricherà IEDriverServer.exe nella cartella del selenio

Primo test usando il goniometro

Il goniometro necessita solo di due file per eseguire il primo file di test, di specifiche (codice di test) e il file di configurazione. Il file spec contiene il codice di test e l'altro contiene i dettagli di configurazione come il percorso del file spec, i dettagli del browser, l'url del test, i parametri del

framework ecc. Per scrivere il primo test forniremo solo l'indirizzo del server di selenio e il percorso del file spec. Gli altri parametri come il browser , timeout, il framework verrà prelevato ai valori predefiniti.

Il browser predefinito per Protractor è Chrome.

conf.js - File di configurazione

```
exports.config = {
  seleniumAddress: 'http://localhost:4444/wd/hub',
  specs: ['spec.js']
};
```

spec.js - File Spec (codice di test)

```
describe('first test in protractor', function() {
  it('should verify title', function() {
    browser.get('https://angularjs.org');

    expect(browser.getTitle()).toEqual('AngularJS - Superheroic JavaScript MVW Framework');
  });
});
```

seleniumAddress - Percorso del server su cui è in esecuzione il server webdriver.

specifiche : un elemento dell'array che contiene il percorso dei file di test. I percorsi multipli possono essere specificati da valori separati da virgola.

descrivere - Sintassi dal framework [Jasmine](#) . describe sintassi

Scrivi un test del goniometro

Aprire una nuova riga di comando o una finestra di terminale e creare una cartella pulita per il test.

Il goniometro ha bisogno di due file da eseguire, un file spec e un file di configurazione.

Iniziamo con un semplice test che naviga nell'esempio dell'elenco di cose da fare nel sito Web di AngularJS e aggiunge un nuovo elemento di todo all'elenco.

Copia quanto segue in `spec.js`

descrivere ('angularjs homepage todo list', function () {it ('dovrebbe aggiungere un todo', function () {browser.get (' <https://angularjs.org> ');

```
element(by.model('todoList.todoText')).sendKeys('write first protractor test');
element(by.css('[value="add"]')).click();

var todoList = element.all(by.repeater('todo in todoList.todos'));
expect(todoList.count()).toEqual(3);
expect(todoList.get(2).getText()).toEqual('write first protractor test');

// You wrote your first test, cross it off the list
```

```
todoList.get(2).element(by.css('input')).click();
var completedAmount = element.all(by.css('.done=true'));
expect(completedAmount.count()).toEqual(2);});});
```

Test di corsa selettivi

Il goniometro può eseguire selettivamente gruppi di test usando `fdescribe ()` invece di `describe ()`.

```
fdescribe('first group', ()=>{
  it('only this test will run', ()=>{
    //code that will run
  });
});
describe('second group', ()=>{
  it('this code will not run', ()=>{
    //code that won't run
  });
});
```

Il goniometro può eseguire in modo selettivo test all'interno di gruppi usando `fit ()` anziché `it ()`.

```
describe('first group', ()=>{
  fit('only this test will run', ()=>{
    //code that will run
  });
  it('this code will not run', ()=>{
    //code that won't run
  });
});
```

Se non c'è `adattamento ()` all'interno di un `file fdescribe ()`, quindi verrà eseguito ogni `esso ()`. Tuttavia, un `fit ()` bloccherà `()` chiamate all'interno della stessa descrizione `()` o `fdescribe ()`.

```
fdescribe('first group', ()=>{
  fit('only this test will run', ()=>{
    //code that will run
  });
  it('this code will not run', ()=>{
    //code that won't run
  });
});
```

Anche se `fit ()` si trova in una descrizione `()` anziché in un `fdescribe ()`, verrà eseguito. Inoltre, qualsiasi `esso ()` all'interno di un `fdescribe ()` che non contiene un `adattamento ()` verrà eseguito.

```
fdescribe('first group', ()=>{
  it('this test will run', ()=>{
    //code that will run
  });
  it('this test will also run', ()=>{
    //code that will also run
  });
});
describe('second group', ()=>{
```

```
it('this code will not run', ()=>{
  //code that won't run
});
fit('this code will run', (){
  //code that will run
});
});
```

Test in sospeso

Il goniometro consente di impostare i test come in sospeso. Ciò significa che il goniometro non eseguirà il test, ma produrrà invece:

```
Pending:
1) Test Name
Temporarily disabled with xit
```

Oppure, se disabilitato con `xdescribe ()`:

```
Pending:
1) Test Name
No reason given
```

combinazioni

- Un `xit ()` all'interno di `xdescribe ()` emetterà la risposta `xit ()`.
- Un `xit ()` all'interno di un `fdescribe ()` sarà comunque considerato come in sospeso.
- Un `fit ()` all'interno di `xdescribe ()` verrà comunque eseguito e nessun test in sospeso produrrà alcun risultato.

Goniometro: test E2E per applicazioni angolari aziendali

Installazione e impostazione del goniometro

Passaggio 1 : scarica e installa NodeJS da qui. Assicurati di avere l'ultima versione del nodo. Qui, sto usando il nodo v7.8.0. Sarà necessario installare il Java Development Kit (JDK) per eseguire il selenio.

Passaggio 2 : apri il terminale e digita il seguente comando per installare a livello globale il goniometro.

```
npm install -g protractor
```

Questo installerà due strumenti come il goniometro e il gestore di webdriver. È possibile verificare l'installazione del goniometro seguendo il seguente comando: `protractor -version`. Se Goniometro viene installato correttamente, il sistema visualizzerà la versione installata (es. Versione 5.1.1). In caso contrario dovrai ricontrollare l'installazione. Passaggio 3: aggiornare il gestore di webdriver per scaricare i file binari necessari.

Passo 4: il seguente comando avvia un Selenium Server. Questo passaggio eseguirà il web driver manager in background e ascolterà qualsiasi test eseguito tramite goniometro.

start di webdriver-manager È possibile visualizzare informazioni sullo stato del server su <http://localhost:4444/wd/hub/static/resource/hub.html>.

Scrittura del primo test case usando il goniometro:

Prima di passare alla scrittura del test case, dobbiamo preparare due file che sono file di configurazione e file spec.

Nel file di configurazione:

```
//In conf.js
exports.config = {
  baseUrl: 'http://localhost:8800/adminapp',
  seleniumAddress: 'http://localhost:4444/wd/hub',
  specs: ['product/product_test.js'],
  directConnect : true,
  capabilities :{
    browserName: 'chrome'
  }
}
```

Comprensione di base delle terminologie utilizzate nel file di configurazione:

baseUrl - Un URL di base per la tua applicazione in prova.

seleniumAddress - Per connettersi a un server Selenium che è già in esecuzione.

specifiche - Posizione del file spec

directConnect : true - Per connettersi direttamente al browser Driver.

Funzionalità : se si esegue il test su un singolo browser, utilizzare l'opzione relativa alle funzionalità. Se stai provando su più browser, usa l'array multiCapabilities.

Puoi trovare più opzioni di configurazione da [qui](#) . Hanno descritto tutta la terminologia possibile con la sua definizione.

Nel file Spec:

```
//In product_test.js

describe('Angular Enterprise Boilerplate', function() {
  it('should have a title', function() {
    browser.get('http://localhost:8800/adminapp');
    expect(browser.getTitle()).toEqual('Angular Enterprise Boilerplate');
  });
});
```

Comprensione di base delle terminologie utilizzate nel file spec:

Di default, Protractor usa il framework jasmine per la sua interfaccia di test. la sintassi 'describe' e 'it' proviene dal framework jasmine. Puoi saperne di più da qui. Esecuzione del primo test:

Prima di eseguire il test case assicurati che il tuo gestore di webdriver e la tua applicazione funzionino in diverse schede del tuo terminale.

Ora, esegui il test con:

```
Protractor app/conf.js
```

Dovresti vedere il browser Chrome che si apre con l'URL della tua applicazione e si chiude da solo. L'output del test dovrebbe essere 1 test, 1 asserzione, 0 guasti.

Bravo! Esegui con successo il tuo primo test case.

Leggi Iniziare con il goniometro online: <https://riptutorial.com/it/protractor/topic/933/iniziare-con-il-goniometro>

Capitolo 2: Controllo del flusso e promesse

introduzione

Goniometro / WebDriverJS ha questo meccanismo chiamato [Controllo del flusso](#) - è una coda interna di promesse, mantiene organizzata l'esecuzione del codice.

Examples

Capire il flusso di controllo

Considera il seguente test:

```
it('should test something', function() {
  browser.get('/dashboard/');

  $("#myid").click();
  expect(element(by.model('username')).getText()).toEqual('Test');

  console.log("HERE");
});
```

Nel seguente test, quando si esegue `console.log()` e si vede `HERE` sulla console, nessuno dei comandi del goniometro dalle righe precedenti è stato eseguito. Questo è un comportamento completamente *asincrono*. I comandi sono rappresentati come promesse e sono stati inseriti nel flusso di controllo che eseguirà e risolverà le promesse in sequenza, una per una.

Vedi di più su [Promises e il flusso di controllo](#).

Leggi [Controllo del flusso e promesse online](#):

<https://riptutorial.com/it/protractor/topic/8580/controllo-del-flusso-e-promesse>

Capitolo 3: Explicit attende con `browser.wait()`

Examples

`browser.sleep()` vs `browser.wait()`

Quando si tratta di gestire il problema della temporizzazione, è allettante e facile inserire un `browser.sleep(<timeout_in_milliseconds> "rapido")` e andare avanti.

Il problema è che un giorno fallirebbe. Non esiste una regola golden / generica su quale sleep timeout impostare e, quindi, a un certo punto a causa di rete o prestazioni o altri problemi, potrebbe essere necessario più tempo per caricare una pagina o elemento per diventare visibile ecc. Inoltre, la maggior parte di il tempo, si finirebbe ad aspettare più di quanto si dovrebbe effettivamente.

`browser.wait()` d'altra parte funziona diversamente. Fornisci una [funzione di Condizione prevista](#) per Protractor / WebDriverJS da eseguire e attendi che il risultato della funzione venga valutato su true. *Il goniometro eseguirà continuamente la funzione e si fermerà una volta che il risultato della funzione sarà valutato su vero o se è stato raggiunto un timeout configurabile.*

Esistono più condizioni previste incorporate, ma puoi anche crearne e utilizzarne uno personalizzato (esempio [qui](#)).

Leggi [Explicit attende con browser.wait\(\)](#) online:

<https://riptutorial.com/it/protractor/topic/8297/explicit-attende-con-browser-wait--->

Capitolo 4: File di configurazione del goniometro

introduzione

Il file di configurazione contiene informazioni utilizzate da Protractor per eseguire lo script di test. Qui proverò a dare alcune varianti diverse.

Examples

Semplice file di configurazione - Chrome

```
var config = {};  
var timeout = 120000;  
  
config.framework = 'jasmine2';  
config.allScriptsTimeout = timeout;  
config.getPageTimeout = timeout;  
config.jasmineNodeOpts.isVerbose = true;  
config.jasmineNodeOpts.defaultTimeoutInterval = timeout;  
config.specs = ['qa/**/*.Spec.js'];  
config.browserName = 'chrome';  
  
exports.config = config;
```

File di configurazione con funzionalità - Chrome

```
var config = {};  
var timeout = 120000;  
  
config.framework = 'jasmine2';  
config.allScriptsTimeout = timeout;  
config.getPageTimeout = timeout;  
config.jasmineNodeOpts.isVerbose = true;  
config.jasmineNodeOpts.defaultTimeoutInterval = timeout;  
config.specs = ['qa/**/*.Spec.js'];  
config.capabilities = {  
  browserName: 'chrome',  
  'chromeOptions': {  
    'args': ['start-minimized', 'window-size=1920,1080']  
  }  
};  
  
exports.config = config;
```

file di configurazione shardTestFiles - Chrome

Questa configurazione consente di eseguire i file spec totali in due istanze del browser in parallelo. Aiuta a ridurre il tempo di esecuzione complessivo del test. Cambia le maxInstances in

base alle tue necessità.

Nota : assicurati che i tuoi test siano indipendenti.

```
var config = {};  
var timeout = 120000;  
  
config.framework = 'jasmine2';  
config.allScriptsTimeout = timeout;  
config.getPageTimeout = timeout;  
config.jasmineNodeOpts.isVerbose = true;  
config.jasmineNodeOpts.defaultTimeoutInterval = timeout;  
config.specs = ['qa/**/*.Spec.js'];  
config.capabilities = {  
  browserName: 'chrome',  
  shardTestFiles: true,  
  maxInstances: 2,  
  'chromeOptions': {  
    'args': ['start-minimized', 'window-size=1920,1080']  
  }  
};  
  
exports.config = config;
```

emule multi-capability del file di configurazione - chrome

```
var config = {};  
var timeout = 120000;  
  
config.framework = 'jasmine2';  
config.allScriptsTimeout = timeout;  
config.getPageTimeout = timeout;  
config.jasmineNodeOpts.isVerbose = true;  
config.jasmineNodeOpts.defaultTimeoutInterval = timeout;  
config.specs = ['qa/**/*.Spec.js'];  
config.multiCapabilities = [{  
  browserName: 'chrome',  
  shardTestFiles: true,  
  maxInstances: 2,  
  'chromeOptions': {  
    'args': ['start-minimized', 'window-size=1920,1080']  
  }  
},  
{  
  browserName: 'chrome',  
  shardTestFiles: true,  
  maxInstances: 1,  
  'chromeOptions': {  
    'args': ['show-fps-counter=true'],  
    'mobileEmulation': {  
      'deviceName': 'Apple iPhone 6'  
    }  
  }  
}  
];  
  
exports.config = config;
```

Leggi File di configurazione del goniometro online:

<https://riptutorial.com/it/protractor/topic/9745/file-di-configurazione-del-goniometro>

Capitolo 5: Goniometro Debugger

Sintassi

- `browser.pause ()`
- `browser.debugger ()`

Osservazioni

Questa sezione spiega come possiamo eseguire il debug dei test del goniometro.

Examples

Utilizzo di `browser.pause ()`

Il metodo `pause ()` è una delle soluzioni più semplici che consente a Protractor di eseguire il debug del codice, per poterlo utilizzare è necessario aggiungerlo nel codice in cui si desidera sospendere l'esecuzione. Una volta eseguita l'esecuzione è in pausa:

1. Puoi usare `c` (tipo C) per andare avanti. Prestare attenzione durante l'utilizzo, è necessario scrivere questo comando senza alcun ritardo in quanto si potrebbe ottenere errore di timeout dalla libreria delle asserzioni se si è ritardato a premere `c`.
2. Digitare `repl` per accedere alla modalità interattiva. La modalità interattiva viene utilizzata per inviare comandi del browser direttamente per aprire l'istanza del browser. Ad esempio in modalità interattiva è possibile emettere un comando come questo:

```
> element (by.css ('#username')).getText ()
> NoSuchElementError: No element found using locator: by.username("#username")
```

L'output di avviso del comando precedente viene visualizzato direttamente lì, che ti consente di conoscere la correttezza del comando.

Nota: se hai aperto Chrome Dev Tools, devi chiuderli prima di continuare il test perché ChromeDriver non può funzionare quando gli strumenti Dev sono aperti.

3. Esci dalla modalità di debug usando `CTRL+C`, puoi uscire dalla modalità di debug usando il classico comando `CTRL + C`.

```
it ('should pause when we use pause method', function () {
  browser.get ('/index.html');

  var username = element (by.model ('username'));
  username.sendKeys ('username');
  browser.pause ();

  var password = element (by.model ('password'));
```

```
password.sendKeys('password');
browser.pause();
});
```

4. Premere `d` per continuare con la successiva istruzione debugger

Utilizzando `browser.debugger ()`

È possibile utilizzare `browser.debugger ()` per interrompere l'esecuzione. È possibile inserirlo in qualsiasi posizione nel codice e interromperà l'esecuzione dopo quella riga finché non si impone di continuare.

Nota: per eseguire i test in modalità debugger devi emettere un comando come questo:

```
`protractor debug <configuration.file.js>`
```

Inserisci `c` per avviare l'esecuzione e continuare dopo il punto di interruzione o immettere `next command`. The passi vicino comando alla riga successiva nel flusso di controllo.

Il debugger utilizzato in Protractor utilizza il [debugger del nodo](#) e sospende l'esecuzione in modo asincrono. Ad esempio, nel codice sottostante il `browser.debugger ()` verrà richiamato quando `username.sendKeys ('username')` è stato eseguito.

Nota: poiché si tratta di attività asincrona, è necessario aumentare il timeout predefinito delle specifiche altrimenti verrà generata un'eccezione di timeout predefinita!

```
it('should pause when we use pause method', function () {
  browser.get('/index.html');

  var username = element(by.model('username'));
  username.sendKeys('username');
  browser.debugger();

  var password = element(by.model('password'));
  password.sendKeys('password');
});
```

Si può entrare nella modalità `repl` inserendo il comando-

```
debug > repl
> element(by.model('abc')).sendKeys('xyz');
```

Questo eseguirà il comando `sendKeys` come operazione successiva, quindi reinserirà il debugger.

Uno può cambiare il `Port no.` vogliono eseguire il debug dei propri script semplicemente passando la porta al metodo `debugger-`

```
browser.debugger(4545); //will start the debugger in port 4545
```

Il metodo `debugger ()` inietta un lato client dal goniometro al browser ed è possibile eseguire pochi

comandi nella console del browser per recuperare gli elementi. Uno degli esempi per utilizzare lo script lato client è:

```
window.clientSideScripts.findInputs('username');
```

Leggi Goniometro Debugger online: <https://riptutorial.com/it/protractor/topic/3910/goniometro-debugger>

Capitolo 6: Individuazione degli elementi

introduzione

Per poter interagire con una pagina, devi dire a Protractor esattamente quale elemento cercare. Le basi utilizzate per selezionare gli elementi sono i locator. Il goniometro, oltre a includere i selettori generici del selenio, ha anche locatori angolari specifici che sono più robusti e persistenti ai cambiamenti. Tuttavia, a volte, anche in un'applicazione Angolare, devono essere utilizzati locatori regolari.

Parametri

Parametro	Dettaglio
selettore	Una stringa che specifica il valore del selettore (dipende dal localizzatore)

Examples

Localizzatori specifici per goniometro (per applicazioni angolari)

Questi locatori dovrebbero essere usati come priorità quando possibile, perché sono più persistenti ai cambiamenti in un'applicazione e quindi ai localizzatori basati su css o xpath, che possono facilmente rompersi.

Localizzatore vincolante

Sintassi

```
by.binding('bind value')
```

Esempio

vista

```
<span>{{user.password}}</span>  
<span ng-bind="user.email"></span>
```

Locator

```
by.binding('user.password')  
by.binding('user.email')
```

Supporta anche le corrispondenze parziali

```
by.binding('email')
```

Localizzatore di binding esatto

Simile al `binding`, tranne le partite parziali non sono consentite.

Sintassi

```
by.exactBinding('exact bind value')
```

Esempio

vista

```
<span>{{user.password}}</span>
```

Locator

```
by.exactBinding('user.password')  
by.exactBinding('password') // Will not work
```

Localizzatore di modelli

Seleziona un elemento con una [direttiva del modello angolare](#)

Sintassi

```
by.model('model value')
```

Esempio

vista

```
<input ng-model="user.username">
```

Locator

```
by.model('user.username')
```

Localizzatore di testo pulsante

Seleziona un pulsante in base al suo testo. Dovrebbe essere usato solo se il testo del pulsante non dovrebbe cambiare spesso.

Sintassi

```
by.buttonText('button text')
```

Esempio

vista

```
<button>Sign In</button>
```

Locator

```
by.buttonText('Sign In')
```

Localizzatore di testo pulsante parziale

Simile a `buttonText`, ma consente corrispondenze parziali. Dovrebbe essere usato solo se il testo del pulsante non dovrebbe cambiare spesso.

Sintassi

```
by.partialButtonText('partial button text')
```

Esempio

vista

```
<button>Register an account</button>
```

Locator

```
by.partialButtonText('Register')
```

Localizzatore di ripetitori

Seleziona un elemento con una [direttiva ripetitore angolare](#)

Sintassi

```
by.repeater('repeater value')
```

Esempio

vista

```
<tbody ng-repeat="review in reviews">
  <tr>Movie was good</tr>
  <tr>Movie was ok</tr>
  <tr>Movie was bad</tr>
</tbody>
```

Locator

```
by.repeater('review in reviews')
```

Supporta anche le corrispondenze parziali

```
by.repeater('reviews')
```

Localizzatore ripetitore esatto

Simile al `repeater` , ma non consente corrispondenze parziali

Sintassi

```
by.exactRepeater('exact repeater value')
```

Esempio

vista

```
<tbody ng-repeat="review in reviews">
  <tr>Movie was good</tr>
  <tr>Movie was ok</tr>
  <tr>Movie was bad</tr>
</tbody>
```

Locator

```
by.exactRepeater('review in reviews')
by.exactRepeater('reviews') // Won't work
```

Localizzatore di testo e CSS

Un localizzatore CSS esteso in cui è anche possibile specificare il contenuto del testo dell'elemento.

Sintassi

```
by.cssContainingText('css selector', 'text of css element')
```

Esempio

vista

```
<ul>
  <li class="users">Mike</li>
  <li class="users">Rebecca</li>
</ul>
```

Locator

```
by.cssContainingText('.users', 'Rebecca') // Will return the second li only
```

Opzioni di ricerca

Seleziona un elemento con una [direttiva di opzioni angulari](#)

Sintassi

```
by.options('options value')
```

Esempio

vista

```
<select ng-options="country.name for c in countries">
  <option>Canada</option>
  <option>United States</option>
  <option>Mexico</option>
</select>
```

Locator

```
by.options('country.name for c in countries')
```

Localizzatore CSS profondo

Localizzatore CSS che si estende nel [DOM ombra](#)

Sintassi

```
by.deepCss('css selector')
```

Esempio

vista

```
<div>
  <span id="outerspan">
    <"shadow tree">
      <span id="span1"></span>
      <"shadow tree">
        <span id="span2"></span>
      </>
    </>
  </div>
```

Locator

```
by.deepCss('span') // Will select every span element
```

Nozioni di base sul localizzatore

I localizzatori di per sé non restituiscono un elemento che può essere interagito con in Goniometro, sono semplicemente istruzioni che indicano Goniometro come trovare l'elemento.

Per accedere all'elemento stesso, usa questa sintassi:

```
element(locator);
element.all(locator);
```

Nota: l'elemento (i) non è effettivamente accessibile fino a quando non viene eseguita un'azione su di esso - cioè, il goniometro andrà effettivamente a recuperare l'elemento quando un'azione sull'elemento viene chiamata `getText()`.

Se vuoi selezionare solo un elemento usando un localizzatore, usa l' `element`. Se il tuo localizzatore punta a più elementi, l' `element` restituirà il primo trovato. `element` restituisce un `ElementFinder`.

Se vuoi selezionare più elementi usando un localizzatore, `element.all` restituirà tutti gli elementi trovati. `element.all` restituisce un `ElementArrayFinder` e ogni elemento dell'array è accessibile tramite diversi metodi, ad esempio la funzione `map`.

```
element.all(locator).map(function(singleElement) {
    return singleElement.getText();
});
```

Localizzatori di catene

È possibile collegare più localizzatori per selezionare un elemento in un'applicazione complessa. Non è possibile collegare direttamente gli oggetti `locator`, è necessario collegare gli `ElementFinders`:

```
element(by.repeater('movie in movies')).element(by.linkText('Watch Frozen on Netflix'))
```

Non c'è limite al numero di catene che puoi usare; alla fine, riceverai comunque un singolo `ElementFinder` o `ElementArrayFinder`, a seconda dei localizzatori.

Leggi Individuazione degli elementi online:

<https://riptutorial.com/it/protractor/topic/10825/individuazione-degli-elementi>

Capitolo 7: Oggetti di pagina

introduzione

Gli oggetti Page è un modello di progettazione che si traduce in meno duplicati di codice, facilità di manutenzione e maggiore leggibilità.

Examples

Primo oggetto della pagina

```
/* save the file in 'pages/loginPage'
var LoginPage = function(){

};

/*Application object properties*/
LoginPage.prototype = Object.create({}, {
  userName: {
    get: function() {
      return browser.driver.findElement(By.id('userid'));
    }
  },
  userPass: {
    get: function() {
      return browser.driver.findElement(By.id('password'));
    }
  },
  submitBtn: {
    get: function() {
      return browser.driver.findElement(By.id('btnSubmit'));
    }
  }
});

/* Adding functions */
LoginPage.prototype.login = function(strUser, strPass) {
  browser.driver.get(browser.baseUrl);
  this.userName.sendKeys(strUser);
  this.userPass.sendKeys(strPass);
  this.submitBtn.click();
};

module.exports = LoginPage;
```

Usiamo il nostro file oggetto prima pagina nel nostro test.

```
var LoginPage = require('../pages/loginPage');
describe('User Login to Application', function() {
  var loginPage = new LoginPage();

  beforeEach(function() {
    loginPage.login(browser.params.userName, browser.params.userPass);
```

```
});  
  
it('and see a success message in title', function() {  
    expect(browser.getTitle()).toEqual('Success');  
});  
  
});
```

Leggi Oggetti di pagina online: <https://riptutorial.com/it/protractor/topic/9747/oggetti-di-pagina>

Capitolo 8: Selettori CSS

Sintassi

- `by.css` ('css selettore')
- `by.id` ('id')
- `by.model` ('modello')
- `by.binding` ('vincolante')

Parametri

Parametro	Dettagli
css selettore	Un selettore css come <code>' .class-name '</code> per selezionare l'elemento sulla base del nome della classe
id	Id dell'elemento dom
modello	Modello utilizzato per l'elemento dom
rilegatura	Nome del legame che viene utilizzato per il collegamento a determinati elementi

Osservazioni

Come scrivere i selettori CSS?

Gli attributi più importanti per scrivere i selettori di CSS sono la classe e l'id di dom. Per un'istanza se un dom HTML si presenta come nell'esempio seguente:

```
<form class="form-signin">
  <input type="text" id="email" class="form-control" placeholder="Email">
  <input type="password" id="password" class="form-control" placeholder="Password">
  <button class="btn btn-block" id="signin-button" type="submit">Sign in</button>
</form>
```

Quindi per selezionare il campo di inserimento dell'email, puoi scrivere il selettore css nel seguente modo:

1. **Usando il nome della classe** : il nome della classe nel selettore css inizia con un carattere speciale (punto). Il selettore css per quello sarà come questo `.form-control` .

```
by.css('.form-control')
```

Poiché la classe di `form-control` della `form-control` è condivisa da entrambi gli elementi di input, ciò solleva una preoccupazione di duplicità nei localizzatori. Quindi, in tale situazione, se id è

disponibile, si dovrebbe sempre preferire l'uso di id invece del nome della classe.

2. **Utilizzo dell'ID** : l'id nel selettore CSS inizia con il carattere speciale # (hash). Quindi il selettore css usando id per l'elemento di input email verrà scritto come di seguito:

```
by.css('#email')
```

3. **Usando più nomi di classi** : Se l'elemento dom ha più classi, allora puoi con una combinazione di classi come selettore di CSS. Per esempio se l'elemento dom è così:

```
<input class="username-class form-control">
// css selector using multiple classes
by.css('.username-class.form-control')
```

4. **Usare il nome del tag con altri attributi** : l'espressione generale per scrivere il selettore css usando il nome del tag e altri attributi è `tagname[attribute-type='attribute-value']` . Quindi, seguendo l'espressione, il locatore di css per il pulsante di accesso può essere formato in questo modo:

```
by.css("button[type='submit']") //or
by.css("button[id='signin-button']")
```

Examples

Scorciatoie di selezione selettore CSS \$ e \$\$

L'API protractor consente ai localizzatori di elementi CSS di utilizzare la [notazione shortcut](#) jQuery-like `$()` .

Localizzatore di elementi CSS normale :

```
element(by.css('h1.documentation-text[ng-bind="title"]'));
element(by.css('[ng-click="submit"]));
```

Collegamento `$()` Localizzatore elemento CSS :

```
$('#h1.documentation-text[ng-bind="title"]');
$('#[ng-click="submit"]');
```

Per trovare più elementi sotto un localizzatore usa la [notazione di collegamento](#) `$$ ()` .

Localizzatore di elementi CSS normale :

```
element.all(by.css('h1.documentation-text[ng-bind="title"]'));
element.all(by.css('[ng-click="submit"]));
```

Collegamento `$$ ()` Localizzatore elemento CSS :

```
$$('h1.documentation-text[ng-bind="title"]');  
$$('[ng-click="submit"]');
```

Introduzione ai locatori

Un localizzatore in Goniometro viene utilizzato per eseguire azioni sugli elementi dom HTML. I locator più comuni e migliori utilizzati in Protractor sono css, id, model e binding. Ad esempio i locatori comunemente usati sono:

```
by.css('css-selector')  
by.id('id')
```

Seleziona l'elemento con un valore dell'attributo HTML esatto

Per selezionare un elemento con un attributo HTML esatto usa il modello del localizzatore css [\[attributo = valore\]](#)

```
//selects the first element with href value '/contact'  
element(by.css('[href="/contact"]'));  
  
//selects the first element with tag option and value 'foo'  
element(by.css('option[value="foo"]'));  
  
//selects all input elements nested under the form tag with name attribute 'email'  
element.all(by.css('form input[name="email"]'));
```

Seleziona l'elemento con un attributo HTML che contiene un valore specificato

Per selezionare un elemento con un attributo HTML che contiene un valore specificato usa il pattern css locator [\[attributo * = valore\]](#)

```
//selects the first element with href value that contains 'cont'  
element(by.css('[href*="cont"]'));  
  
//selects the first element with tag h1 and class attribute that contains 'fo'  
element(by.css('h1[class*="fo"]'));  
  
//selects all li elements with a title attribute that contains 'users'  
element.all(by.css('li[title*="users"]'));
```

Leggi Selettori CSS online: <https://riptutorial.com/it/protractor/topic/1524/selettori-css>

Capitolo 9: Selettori XPath in goniometro

Examples

Selezione di un elemento DOM utilizzando il rapportatore

Oltre al selettore CSS, al modello e al binding, il goniometro può anche individuare gli elementi usando xpath View

```
<ul>
<li><a href='http://www.google.com'>Go to google</a></li>
</ul>
```

Codice

```
var googleLink= element(by.xpath('//ul/li/a'));
expect(element.getText()).to.eventually.equal('Go to google','The text you mention was not found');
```

Selezione di elementi con attributi specifici

I selettori XPath possono essere utilizzati per selezionare elementi con attributi specifici, come classe, id, titolo ecc.

Per classe

Vista:

```
<div class="HakunaMatata"> Hakuna Matata </div>
```

Codice:

```
var theLionKing= element(by.xpath('//div[@class="HakunaMatata"]'));
expect(theLionKing.getText()).to.eventually.equal('Hakuna Matata', "Text not found");
```

Tuttavia, un elemento può avere più classi. In questi casi, è possibile utilizzare la soluzione alternativa 'contains'

Vista:

```
<div class="Hakuna Matata"> Hakuna Matata </div>
```

Codice:

```
var theLionKing= element(by.xpath('//div[contains(@class,"Hakuna")]'));
```

```
expect (theLionKing.getText()).to.eventually.equal('Hakuna Matata', "Text not found");
```

La parte di codice sopra riportata restituirà elementi contenenti sia 'class = "HakunaMatata" "che" class = "Hakuna Matata" ". Se il testo di ricerca fa parte di un elenco separato da spazi, è possibile utilizzare la seguente soluzione alternativa:

```
var theLionKing= element (by.xpath('//div[contains(concat(' ',normalize-space(@class),' '), 'Hakuna')]'));  
expect (theLionKing.getText()).to.eventually.equal('Hakuna Matata', "Text not found");
```

Per id

L'ID rimane il localizzatore più semplice e preciso che può essere utilizzato per selezionare un elemento.

Vista:

```
<div id="HakunaMatata">Hakuna Matata</div>
```

Codice:

```
var theLionKing= element (by.xpath('//div[@id="HakunaMatata"]'));  
expect (theLionKing.getText()).to.eventually.equal('Hakuna Matata', "Text not found");
```

Come con le classi, la funzione contiene può essere utilizzata per trovare un elemento contenente il testo specificato.

Altri attributi

Trovare un elemento con un determinato attributo **titolo**

vista

```
<div title="Hakuna Matata">Hakuna Matata</div>
```

Codice

```
var theLionKing= element (by.xpath('//div[@title="Hakuna Matata"]'));  
expect (theLionKing.getText()).to.eventually.equal('Hakuna Matata', "Text not found");
```

Selezione di un elemento con un testo specifico

vista

```
<div class="Run Simba Run">Run Simba</div>
```

Codice

```
var runSimba= element(by.xpath('//div[text()="Run Simba"]'));
```

Come con altre ricerche basate sul testo, la funzione `contains()` può essere utilizzata per selezionare elementi con testo () contenente la corrispondenza richiesta.

vista

```
<div class="Run Simba Run">Run Simba,run</div>
```

Codice

```
var runSimba= element(by.xpath('//div[contains(text(),"Run Simba")]'));  
expect(runSimba.getText()).to.eventually.equal('Run Simba, run', "Text not found"); //true
```

Selezione di un elemento con un attributo nome specifico

vista

```
<input type="text" name="FullName"></input>
```

Codice

```
var fullNameInput= element(by.xpath('//input[@name="FullName"]'));  
fullNameInput.sendKeys("John Doe");
```

Leggi Selettori XPath in goniometro online: <https://riptutorial.com/it/protractor/topic/7205/selettori-xpath-in-goniometro>

Capitolo 10: Test di app non angolari con Goniometro

introduzione

Il goniometro è realizzato per testare le applicazioni angolari. Tuttavia, è ancora possibile testare applicazioni non angolari con Goniometro, se necessario.

Examples

Modifiche necessarie per testare l'applicazione non angolare con Goniometro

Utilizza `browser.driver` anziché `driver`

Utilizza `browser.driver.ignoreSynchronization = true`

Motivo : il goniometro attende che i componenti angolari vengano caricati completamente su una pagina Web prima che inizi l'esecuzione. Tuttavia, poiché le nostre pagine non sono spigolose, il goniometro continua ad aspettare che venga caricato "angolare" fino a quando il test non riesce con il timeout. Quindi, dobbiamo dire esplicitamente al goniometro di non aspettare "angolare"

Leggi [Test di app non angolari con Goniometro online](https://riptutorial.com/it/protractor/topic/8830/test-di-app-non-angolari-con-goniometro):

<https://riptutorial.com/it/protractor/topic/8830/test-di-app-non-angolari-con-goniometro>

Titoli di coda

S. No	Capitoli	Contributors
1	Iniziare con il goniometro	Bhoomi Bhalani , Community , Devmati Wadikar , Manuli Piyalka , olyv , Peter Stegnar , Praveen , Priyanshu Shekhar , SilentLupin , sonhu , Stephen Leppik
2	Controllo del flusso e promesse	alecxe
3	Explicit attende con <code>browser.wait ()</code>	alecxe
4	File di configurazione del goniometro	Barney
5	Goniometro Debugger	Devmati Wadikar , Priyanshu Shekhar , Ram Pasala , Sakshi Singla , Stephen Leppik
6	Individuazione degli elementi	Sébastien Dufour-Beauséjour
7	Oggetti di pagina	Barney , Suresh Salloju
8	Selettori CSS	alecxe , Droogans , leon , Priyanshu Shekhar , sonhu
9	Selettori XPath in goniometro	Shubhang
10	Test di app non angolari con Goniometro	Sakshi Singla