Evaluating JavaScript

Introduction

Playwright scripts run in your Playwright environment. Your page scripts run in the browser page environment. Those environments don't intersect, they are running in different virtual machines in different processes and even potentially on different computers.

The page.evaluate() API can run a JavaScript function in the context of the web page and bring results back to the Playwright environment. Browser globals like `window` and `document` can be used in `evaluate`.

```
const href = await page.evaluate(() => document.location.href);
```

If the result is a Promise or if the function is asynchronous evaluate will automatically wait until it's resolved:

```
const status = await page.evaluate(async () => {
  const response = await fetch(location.href);
  return response.status;
});
```

Different environments

Evaluated scripts run in the browser environment, while your test runs in a testing environments. This means you cannot use variables from your test in the page and vice versa. Instead, you should pass them explicitly as an argument.

The following snippet is **WRONG** because it uses the variable directly:

```
const data = 'some data';
const result = await page.evaluate(() => {
  // WRONG: there is no "data" in the web page.
  window.myApp.use(data);
});
```

The following snippet is **CORRECT** because it passes the value explicitly as an argument:

```
const data = 'some data';
// Pass |data| as a parameter.
const result = await page.evaluate(data => {
```

```
  window.myApp.use(data);
}, data);
```

Evaluation Argument

Playwright evaluation methods like page.evaluate() take a single optional argument. This argument can be a mix of Serializable values and JSHandle instances. Handles are automatically converted to the value they represent.

```
// A primitive value.
await page.evaluate(num => num, 42);

// An array.
await page.evaluate(array => array.length, [1, 2, 3]);

// An object.
await page.evaluate(object => object.foo, { foo: 'bar' });

// A single handle.
const button = await page.evaluateHandle('window.button');
await page.evaluate(button => button.textContent, button);

// Alternative notation using JSHandle.evaluate.
await button.evaluate((button, from) => button.textContent.substring(from),
5);

// Object with multiple handles.
const button1 = await page.evaluateHandle('window.button1');
const button2 = await page.evaluateHandle('window.button2');
await page.evaluate(
    o => o.button1.textContent + o.button2.textContent,
    { button1, button2 });

// Object destructuring works. Note that property names must match
// between the destructured object and the argument.
// Also note the required parenthesis.
await page.evaluate(
    ({ button1, button2 }) => button1.textContent + button2.textContent,
    { button1, button2 });

// Array works as well. Arbitrary names can be used for destructuring.
// Note the required parenthesis.
await page.evaluate(
    ([b1, b2]) => b1.textContent + b2.textContent,
    [button1, button2]);

// Any mix of serializables and handles works.
await page.evaluate(
    x => x.button1.textContent + x.list[0].textContent + String(x.foo),
    { button1, list: [button2], foo: null });
```

Init scripts

Sometimes it is convenient to evaluate something in the page before it starts loading. For example, you might want to setup some mocks or test data.

In this case, use page.addInitScript() or browserContext.addInitScript(). In the example below, we will replace `Math.random()` with a constant value.

First, create a `preload.js` file that contains the mock.

```
// preload.js
Math.random = () => 42;
```

Next, add init script to the page.

```
import { test, expect } from '@playwright/test';
import path from 'path';

test.beforeEach(async ({ page }) => {
  // Add script for every test in the beforeEach hook.
  // Make sure to correctly resolve the script path.
  await page.addInitScript({ path: path.resolve(__dirname,
'../mocks/preload.js') });
});
```

Alternatively, you can pass a function instead of creating a preload script file. This is more convenient for short or one-off scripts. You can also pass an argument this way.

```
import { test, expect } from '@playwright/test';

// Add script for every test in the beforeEach hook.
test.beforeEach(async ({ page }) => {
  const value = 42;
  await page.addInitScript(value => {
    Math.random = () => value;
  }, value);
});
```

Previous
Downloads

Next
Events

Learn

- Getting started
- Playwright Training
- Learn Videos
- Feature Videos

Community

- Stack Overflow
- Discord
- Twitter
- LinkedIn

More

- GitHub
- YouTube
- Blog
- Ambassadors