ExecutionContext.evaluateHandle() method

Signature:

```
evaluateHandle<HandleType extends JSHandle | ElementHandle = JSHandle>(pageFunction:
EvaluateHandleFn, ...args: SerializableOrJSHandle[]): Promise<HandleType>;
```

Parameters

Parameter	Type	Description
pageFunction	EvaluateHandleFn	a function to be evaluated in the `executionContext`
args	SerializableOrJSHandle[]	argument to pass to the page function

Returns:

Promise<HandleType>

A promise that resolves to the return value of the given function as an in-page object (a JSHandle).

Remarks

The only difference between `executionContext.evaluate` and `executionContext.evaluateHandle` is that `executionContext.evaluateHandle` returns an in-page object (a JSHandle). If the function passed to the `executionContext.evaluateHandle` returns a Promise, then `executionContext.evaluateHandle` would wait for the promise to resolve and return its value.

Example 1

```
const context = await page.mainFrame().executionContext();
const aHandle = await context.evaluateHandle(() => Promise.resolve(self));
aHandle; // Handle for the global object.
```

Example 2

A string can also be passed in instead of a function.

```
// Handle for the '3' * object.
const aHandle = await context.evaluateHandle('1 + 2');
```

Example 3

JSHandle instances can be passed as arguments to the `executionContext.* evaluateHandle`:

```
const aHandle = await context.evaluateHandle(() => document.body);
const resultHandle = await context.evaluateHandle(body => body.innerHTML, *
aHandle);
console.log(await resultHandle.jsonValue()); // prints body's innerHTML
await aHandle.dispose();
await resultHandle.dispose();
```