Page.evaluateHandle() method

Signature:

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

Parameters

Parameter	Type	Description
pageFunction	EvaluateHandleFn	a function that is run within the page
args	SerializableOrJSHandle[]	arguments to be passed to the pageFunction

Returns:

Promise<HandlerType>

Remarks

The only difference between page.evaluate and `page.evaluateHandle` is that `evaluateHandle` will return the value wrapped in an in-page object.

If the function passed to `page.evaluteHandle` returns a Promise, the function will wait for the promise to resolve and return its value.

You can pass a string instead of a function (although functions are recommended as they are easier to debug and use with TypeScript):

Example 1

```
const aHandle = await page.evaluateHandle('document')
```

Example 2

JSHandle instances can be passed as arguments to the `pageFunction`:

```
const aHandle = await page.evaluateHandle(() => document.body);
const resultHandle = await page.evaluateHandle(body => body.innerHTML, aHandle);
console.log(await resultHandle.jsonValue());
await resultHandle.dispose();
```

Most of the time this function returns a JSHandle, but if `pageFunction` returns a reference to an element, you instead get an ElementHandle back:

Example 3

```
const button = await page.evaluateHandle(() => document.querySelector('button'));
// can call `click` because `button` is an `ElementHandle`
await button.click();
```

The TypeScript definitions assume that `evaluateHandle` returns a `JSHandle`, but if you know it's going to return an `ElementHandle`, pass it as the generic argument:

```
const button = await page.evaluateHandle<ElementHandle>(...);
```