Page.$$eval() method

This method runs `Array.from(document.querySelectorAll(selector))` within the page and passes the result as the first argument to the `pageFunction`.

Signature:

```
$$eval<ReturnType>(selector: string, pageFunction: (elements: Element[], ...args:
unknown[]) => ReturnType | Promise<ReturnType>, ...args: SerializableOrJSHandle[]):
Promise<WrapElementHandle<ReturnType>>;
```

Parameters

Parameter	Type	Description
selector	string	the selector to query for
pageFunction	(elements: Element[], ...args: unknown[]) => ReturnType | Promise<ReturnType>	the function to be evaluated in the page context. Will be passed the result of `Array.from(document.querySelectorAll(selector))` as its first argument.
args	SerializableOrJSHandle[]	any additional arguments to pass through to `pageFunction`.

Returns:

Promise<WrapElementHandle<ReturnType>>

The result of calling `pageFunction`. If it returns an element it is wrapped in an ElementHandle, else the raw value itself is returned.

Remarks

If `pageFunction` returns a promise `$$eval` will wait for the promise to resolve and then return its value.

Example 1

```
// get the amount of divs on the page
const divCount = await page.$$eval('div', divs => divs.length);

// get the text content of all the `.options` elements:
const options = await page.$$eval('div > span.options', options => {
  return options.map(option => option.textContent)
});
```

If you are using TypeScript, you may have to provide an explicit type to the first argument of the `pageFunction`. By default it is typed as `Element[]`, but you may need to provide a more specific sub-type:

Example 2

```
// if you don't provide HTMLInputElement here, TS will error
// as `value` is not on `Element`
await page.$$eval('input', (elements: HTMLInputElement[]) => {
  return elements.map(e => e.value);
});
```

The compiler should be able to infer the return type from the `pageFunction` you provide. If it is unable to, you can use the generic type to tell the compiler what return type you expect from `$$eval`:

Example 3

```
// The compiler can infer the return type in this case, but if it can't
// or if you want to be more explicit, provide it as the generic type.
const allInputValues = await page.$$eval<string[]>(
 'input', (elements: HTMLInputElement[]) => elements.map(e => e.textContent)
);
```