

## ElementHandle class

ElementHandle represents an in-page DOM element.

### Signature:

```
export declare class ElementHandle<ElementType extends Element = Element> extends
  JSHandle<ElementType>
```

Extends: [JSHandle](#)<ElementType>

## Remarks

ElementHandles can be created with the [Page.\\$\(\)](#) method.

```
const puppeteer = require('puppeteer');

(async () => {
  const browser = await puppeteer.launch();
  const page = await browser.newPage();
  await page.goto('https://example.com');
  const hrefElement = await page.$('a');
  await hrefElement.click();
  // ...
})();
```

ElementHandle prevents the DOM element from being garbage-collected unless the handle is [disposed](#).

ElementHandles are auto-disposed when their origin frame gets navigated.

ElementHandle instances can be used as arguments in [Page.\\$eval\(\)](#) and [Page.evaluate\(\)](#) methods.

If you're using TypeScript, ElementHandle takes a generic argument that denotes the type of element the handle is holding within. For example, if you have a handle to a `<select>` element, you can type it as

`ElementHandle<HTMLSelectElement>` and you get some nicer type checks.

The constructor for this class is marked as internal. Third-party code should not call the constructor directly or create subclasses that extend the `ElementHandle` class.

## Methods

Method	Modifiers	Description
<a href="#">\$(selector)</a>		Runs <code>element.querySelector</code> within the page. If no element matches the selector, the return value resolves to <code>null</code> .
<a href="#">\$\$\$(selector)</a>		Runs <code>element.querySelectorAll</code> within the page. If no elements match the selector, the return value resolves to <code>[]</code> .
<a href="#">\$\$eval(selector, pageFunction, args)</a>		This method runs <code>document.querySelectorAll</code> within the element and passes it as the first argument to <code>pageFunction</code> . If there's no

		<p>element matching <code>selector</code>, the method throws an error. If <code>pageFunction</code> returns a Promise, then <code>frame.\$\$eval</code> would wait for the promise to resolve and return its value.</p>
<a href="#"><code>\$\$eval(selector, pageFunction, args)</code></a>		<p>This method runs <code>document.querySelector</code> within the element and passes it as the first argument to <code>pageFunction</code>. If there's no element matching <code>selector</code>, the method throws an error. If <code>pageFunction</code> returns a Promise, then <code>frame.\$\$eval</code> would wait for the promise to resolve and return its value.</p>
<a href="#"><code>\$x(expression)</code></a>		<p>The method evaluates the XPath expression relative to the <code>elementHandle</code>. If there are no such elements, the method will resolve to an empty array.</p>
<a href="#"><code>asElement()</code></a>		
<a href="#"><code>boundingBox()</code></a>		<p>This method returns the bounding box of the element (relative to the main frame), or <code>null</code> if the element is not visible.</p>
<a href="#"><code>boxModel()</code></a>		<p>This method returns boxes of the element, or <code>null</code> if the element is not visible.</p>
<a href="#"><code>click(options)</code></a>		<p>This method scrolls element into view if needed, and then uses <a href="#">Page.mouse</a> to click in the center of the element. If the element is detached from DOM, the method throws an error.</p>
<a href="#"><code>contentFrame()</code></a>		<p>Resolves to the content frame for element handles referencing iframe nodes, or <code>null</code> otherwise</p>
<a href="#"><code>focus()</code></a>		<p>Calls <a href="#">focus</a> on the element.</p>
<a href="#"><code>hover()</code></a>		<p>This method scrolls element into view if needed, and then uses <a href="#">Page.mouse</a> to hover over the center of the element. If the element is detached from DOM, the method throws an error.</p>
<a href="#"><code>isIntersectingViewport()</code></a>		<p>Resolves to true if the element is visible in the current viewport.</p>
<a href="#"><code>press(key, options)</code></a>		<p>Focuses the element, and then uses <a href="#">Keyboard.down()</a> and <a href="#">Keyboard.up()</a>.</p>
<a href="#"><code>screenshot(options)</code></a>		<p>This method scrolls element into view if needed, and then uses <a href="#">Page.screenshot()</a> to take a screenshot of the element. If the element is detached from DOM, the method throws an error.</p>
<a href="#"><code>select(values)</code></a>		<p>Triggers a <code>change</code> and <code>input</code> event once all the provided options have been selected. If there's no <code>&lt;select&gt;</code> element matching <code>selector</code>, the method throws an error.</p>
<a href="#"><code>tap()</code></a>		<p>This method scrolls element into view if needed, and then uses <a href="#">Touchscreen.tap()</a> to tap in the center of the element. If the element is detached from DOM, the method throws an error.</p>
<a href="#"><code>type(text, options)</code></a>		<p>Focuses the element, and then sends a <code>keydown</code>, <code>keypress/input</code>, and <code>keyup</code> event for each character in the text. To press a special key, like <code>Control</code> or <code>ArrowDown</code>, use <a href="#">ElementHandle.press()</a>.</p>

[uploadFile\(filePaths\).](#)

This method expects `elementHandle` to point to an [input element](#).