Document: parseHTMLUnsafe() static method

- Syntax
  - Parameters
  - Return value
  - Exceptions
- Description
  - Security considerations
- Specifications
- Browser compatibility
- See also

{{APIRef("DOM")}}

> [!WARNING] This method parses its input as HTML, writing the result into the DOM. APIs like this are known as injection sinks, and are potentially a vector for cross-site scripting (XSS) attacks, if the input originally came from an attacker.
>
> You can mitigate this risk by always passing `TrustedHTML` objects instead of strings and enforcing trusted types. See Security considerations for more information.

> [!NOTE] {{domxref("Document/parseHTML_static", "Document.parseHTML()")}} should almost always be used instead of this method — on browsers where it is

> supported — as it always removes XSS-unsafe HTML entities.

The **`parseHTMLUnsafe()`** static method of the {{domxref("Document")}} object is used to parse HTML input, optionally filtering unwanted HTML elements and attributes, in order to create a new {{domxref("Document")}} instance.

Syntax

```
Document.parseHTMLUnsafe(input)
Document.parseHTMLUnsafe(input, options)
```

Parameters

- `input`
  - : A {{domxref("TrustedHTML")}} instance or a string defining HTML to be parsed.
- `options` {{optional_inline}}
  - : An options object with the following optional parameters:
    - `sanitizer` {{optional_inline}}
      - : A {{domxref("Sanitizer")}} or {{domxref("SanitizerConfig")}} object which defines what elements of the input will be allowed or removed. This can also be a string with the value `"default"`, which applies a `Sanitizer` with the (XSS-safe) default sanitizer configuration. If not specified, no sanitizer is used.
        Note that if you're using the same configuration multiple times, it's expected to be more efficient to use a `Sanitizer` and modify it when you need to.

Return value

A {{domxref("Document")}}.

Exceptions

- {{jsxref("TypeError")}}
  - : This is thrown if:

- `html` is passed a string when Trusted Types are enforced by a CSP and no default policy is defined.
- `options.sanitizer` is passed a:
  - {{domxref("SanitizerConfig")}} that isn't valid. For example, a configuration that includes both "allowed" and "removed" configuration settings.
  - string that does not have the value `"default"`.
  - value that is not a {{domxref("Sanitizer")}}, {{domxref("SanitizerConfig")}}, or string.

Description

The **`parseHTMLUnsafe()`** static method can be used to create a new {{domxref("Document")}} instance, optionally filter out unwanted elements and attributes. The resulting `Document` will have a content type of "text/html", a character set of UTF-8, and a URL of "about:blank".

The input HTML may include declarative shadow roots. If the string of HTML defines more than one declarative shadow root in a particular shadow host then only the first {{domxref("ShadowRoot")}} is created — subsequent declarations are parsed as {{htmlelement("template")}} elements within that shadow root.

`parseHTMLUnsafe()` doesn't perform any sanitization by default. If no sanitizer is passed as a parameter, all HTML entities in the input will be injected.

Security considerations

The suffix "Unsafe" in the method name indicates that it does not enforce removal of all XSS-unsafe HTML entities (unlike {{domxref("Document/parseHTML_static", "Document.parseHTML()")}}). While it can do so if used with an appropriate sanitizer, it doesn't have to use an effective sanitizer, or any sanitizer at all! The method is therefore a possible vector for cross-site scripting (XSS) attacks, where potentially unsafe strings provided by a user are injected into the DOM without first being sanitized.

You should mitigate this risk by always passing {{domxref("TrustedHTML")}} objects instead of strings, and enforcing trusted types using the `require-trusted-types-for` CSP directive. This ensures that the input is passed through a transformation function, which has the chance to sanitize the input to remove potentially dangerous markup (such as {{htmlelement("script")}} elements and event handler attributes), before it is injected.

Using `TrustedHTML` makes it possible to audit and check that sanitization code is effective in just a few places, rather than scattered across all your injection sinks. You should not need to pass a sanitizer to the method when using `TrustedHTML`.

If for any reason you can't use `TrustedHTML` (or even better, `setHTML()`) then the next safest option is to use `setHTMLUnsafe()` with the XSS-safe default sanitizer configuration.

Specifications

{{Specifications}}

Browser compatibility

{{Compat}}

See also

- {{domxref("Document.parseHTML_static", "Document.parseHTML()")}}
- {{domxref("Element.setHTML()")}} and {{domxref("Element.setHTMLUnsafe()")}}
- {{domxref("ShadowRoot.setHTML()")}} and {{domxref("ShadowRoot.setHTMLUnsafe()")}}
- {{domxref("DOMParser.parseFromString()")}} for parsing HTML or XML into a DOM tree
- HTML Sanitizer API

□□□ #2
□ □□□ □□□ 2026-10-31 03:27:53 UTC
□ □□□ □□□ 2026-10-31 03:27:53 UTC