# cheerio

##### Fast, flexible & lean implementation of core jQuery designed specifically for the server.

   

[中文文档 (Chinese Readme)](https://github.com/cheeriojs/cheerio/wiki/Chinese-README) ```js const cheerio = require('cheerio'); const $ = cheerio.load('

## Hello world

'); $('h2.title').text('Hello there!'); $('h2').addClass('welcome'); $.html(); //=>

## Hello there!

``` ## Note We are currently working on the 1.0.0 release of cheerio on the `main` branch. The source code for the last published version, `0.22.0`, can be found [here](https://github.com/cheeriojs/cheerio/tree/aa90399c9c02f12432bfff97b8f1c7d8ece7c307). ## Installation `npm install cheerio` ## Features \*\*❤ Familiar syntax:\*\* Cheerio implements a subset of core jQuery. Cheerio removes all the DOM inconsistencies and browser cruft from the jQuery library, revealing its truly gorgeous API. \*\*ϟ Blazingly fast:\*\* Cheerio works with a very simple, consistent DOM model. As a result parsing, manipulating, and rendering are incredibly efficient. \*\*❁ Incredibly flexible:\*\* Cheerio wraps around [parse5](https://github.com/inikulin/parse5) parser and can optionally use @FB55's forgiving [htmlparser2](https://github.com/fb55/htmlparser2/). Cheerio can parse nearly any HTML or XML document. ## Cheerio is not a web browser Cheerio parses markup and provides an API for traversing/manipulating the resulting data structure. It does not interpret the result as a web browser does. Specifically, it does \_not\_ produce a visual rendering, apply CSS, load external resources, or execute JavaScript which is common for a SPA (single page application). This makes Cheerio \*\*much, much faster than other solutions\*\*. If your use case requires any of this functionality, you should consider browser automation software like [Puppeteer](https://github.com/puppeteer/puppeteer) and [Playwright](https://github.com/microsoft/playwright) or DOM emulation projects like [JSDom](https://github.com/jsdom/jsdom). ## API ### Markup example we'll be using: ```html

* Apple
* Orange
* Pear

``` This is the HTML markup we will be using in all of the API examples. ### Loading First you need to load in the HTML. This step in jQuery is implicit, since jQuery operates on the one, baked-in DOM. With Cheerio, we need to pass in the HTML document. This is the \_preferred\_ method: ```js // ES6 or TypeScript: import \* as cheerio from 'cheerio'; // In other environments: const cheerio = require('cheerio'); const $ = cheerio.load('

...'); $.html(); //=>

...``` Similar to web browser contexts, `load` will introduce ``, ``, and `` elements if they are not already present. You can set `load`'s third argument to `false` to disable this. ```js const $ = cheerio.load('

...', null, false); $.html(); //=> '

...' ``` Optionally, you can also load in the HTML by passing the string as the context: ```js $('ul', '

...'); ``` Or as the root: ```js $('li', 'ul', '

...'); ``` If you need to modify parsing options for XML input, you may pass an extra object to `.load()`: ```js const $ = cheerio.load('

...', { xml: { normalizeWhitespace: true, }, }); ``` The options in the `xml` object are taken directly from [htmlparser2](https://github.com/fb55/htmlparser2/wiki/Parser-options), therefore any options that can be used in `htmlparser2` are valid in cheerio as well. When `xml` is set, the default options are: ```js { xmlMode: true, decodeEntities: true, // Decode HTML entities. withStartIndices: false, // Add a `startIndex` property to nodes. withEndIndices: false, // Add an `endIndex` property to nodes. } ``` For a full list of options and their effects, see [domhandler](https://github.com/fb55/DomHandler) and [htmlparser2's options](https://github.com/fb55/htmlparser2/wiki/Parser-options). #### Using `htmlparser2` Cheerio ships with two parsers, `parse5` and `htmlparser2`. The former is the default for HTML, the latter the default for XML. Some users may wish to parse markup with the `htmlparser2` library, and traverse/manipulate the resulting structure with Cheerio. This may be the case for those upgrading from pre-1.0 releases of Cheerio (which relied on `htmlparser2`), for those dealing with invalid markup (because `htmlparser2` is more forgiving), or for those operating in performance-critical situations (because `htmlparser2` may be faster in some cases). Note that "more forgiving" means `htmlparser2` has error-correcting mechanisms that aren't always a match for the standards observed by web browsers. This behavior may be useful when parsing non-HTML content. To support these cases, `load` also accepts a `htmlparser2`-compatible data structure as its first argument. Users may install `htmlparser2`, use it to parse input, and pass the result to `load`: ```js // Usage as of htmlparser2 version 6: const htmlparser2 = require('htmlparser2'); const dom = htmlparser2.parseDocument(document, options); const $ = cheerio.load(dom); ``` If you want to save some bytes, you can use Cheerio's \_slim\_ export, which always uses `htmlparser2`: ```js const cheerio = require('cheerio/lib/slim'); ``` ### Selectors Cheerio's selector implementation is nearly identical to jQuery's, so the API is very similar. #### \$( selector, [context], [root] ) `selector` searches within the `context` scope which searches within the `root` scope. `selector` and `context` can be a string expression, DOM Element, array of DOM elements, or cheerio object. `root` is typically the HTML document string. This selector method is the starting point for traversing and manipulating the document. Like jQuery, it's the primary method for selecting elements in the document. ```js $('.apple', '#fruits').text(); //=> Apple $('ul .pear').attr('class'); //=> pear $('li[class=orange]').html(); //=> Orange ``` ##### XML Namespaces You can select with XML Namespaces but [due to the CSS specification](https://www.w3.org/TR/2011/REC-css3-selectors-20110929/#attribute-selectors), the colon (`:`) needs to be escaped for the selector to be valid. ```js $('[xml\\:id="main"'); ``` ### Rendering When you're ready to render the document, you can call the `html` method on the "root" selection: ```js $.root().html(); //=> // // //

//

* Apple

//

* Orange

//

* Pear

//// // ``` If you want to render the [`outerHTML`](https://developer.mozilla.org/en-US/docs/Web/API/Element/outerHTML) of a selection, you can use the `html` utility functon: ```js cheerio.html($('.pear')); //=>

* Pear

``` You may also render the text content of a Cheerio object using the `text` static method: ```js const $ = cheerio.load('This is *content*.'); cheerio.text($('body')); //=> This is content. ``` ### Plugins Once you have loaded a document, you may extend the prototype or the equivalent `fn` property with custom plugin methods: ```js const $ = cheerio.load('Hello, **world**!'); $.prototype.logHtml = function () { console.log(this.html()); }; $('body').logHtml(); // logs "Hello, **world**!" to the console ``` If you're using TypeScript, you should add a type definition for your new method: ```ts declare module 'cheerio' { interface Cheerio { logHtml(this: Cheerio): void; } } ``` ### The "DOM Node" object Cheerio collections are made up of objects that bear some resemblance to [browser-based DOM nodes](https://developer.mozilla.org/en-US/docs/Web/API/Node). You can expect them to define the following properties: - `tagName` - `parentNode` - `previousSibling` - `nextSibling` - `nodeValue` - `firstChild` - `childNodes` - `lastChild` ## Screencasts [https://vimeo.com/31950192](https://vimeo.com/31950192) > This video tutorial is a follow-up to Nettut's "How to Scrape Web Pages with Node.js and jQuery", using cheerio instead of JSDOM + jQuery. This video shows how easy it is to use cheerio and how much faster cheerio is than JSDOM + jQuery. ## Cheerio in the real world Are you using cheerio in production? Add it to the [wiki](https://github.com/cheeriojs/cheerio/wiki/Cheerio-in-Production)! ## Sponsors Does your company use Cheerio in production? Please consider [sponsoring this project](https://github.com/cheeriojs/cheerio?sponsor=1)! Your help will allow maintainers to dedicate more time and resources to its development and support.        ## Backers [Become a backer](https://github.com/cheeriojs/cheerio?sponsor=1) to show your support for Cheerio and help us maintain and improve this open source project.        ## Special Thanks This library stands on the shoulders of some incredible developers. A special thanks to: \*\*• @FB55 for node-htmlparser2 & CSSSelect:\*\* Felix has a knack for writing speedy parsing engines. He completely re-wrote both @tautologistic's `node-htmlparser` and @harry's `node-soupselect` from the ground up, making both of them much faster and more flexible. Cheerio would not be possible without his foundational work \*\*• @jQuery team for jQuery:\*\* The core API is the best of its class and despite dealing with all the browser inconsistencies the code base is extremely clean and easy to follow. Much of cheerio's implementation and documentation is from jQuery. Thanks guys. \*\*• @visionmedia:\*\* The style, the structure, the open-source"-ness" of this library comes from studying TJ's style and using many of his libraries. This dude consistently pumps out high-quality libraries and has always been more than willing to help or answer questions. You rock TJ. ## License MIT