posthtml-parser

Parse HTML/XML to PostHTML AST. More about PostHTML

Install

NPM install

```
$ npm install posthtml-parser
```

Usage

Input HTML

```
<a class="animals" href="#">
    <span class="animals__cat"
        style="background:
        url(cat.png)">Cat</span>
</a>
```

```
import { parser } from 'posthtml-parser'
import fs from 'fs'
```

```
const html = fs.readFileSync('path/to/
        input.html', 'utf-8')

console.log(parser(html)) // Logs a
        PostHTML AST
```

input HTML

```
<a class="animals" href="#">
    <span class="animals__cat"
        style="background:
        url(cat.png)">Cat</span>
</a>
```

Result PostHTMLTree

```
[{
    tag: 'a',
    attrs: {
        class: 'animals',
        href: '#'
    },
    content: [
        '\n    ',
        {
            tag: 'span',
            attrs: {
                class: 'animals__cat',
                style: 'background:
        url(cat.png)'
            },
            content: ['Cat']
        },
        '\n'
```

`sourceLocations`

Type: `Boolean` Default: `false` Description: *If set to true, AST nodes will have a* `location` *property containing the* `start` *and* `end` *line and column position of the node.*

`recognizeNoValueAttribute`

Type: `Boolean` Default: `false` Description: *If set to true, AST nodes will recognize attribute with no value and mark as* `true` *which will be correctly rendered by* `posthtml-render` *package*

License

MIT


```
    ]
}]
```

PostHTML AST Format

Any parser being used with PostHTML should return a standard PostHTML Abstract Syntax Tree (AST). Fortunately, this is a very easy format to produce and understand. The AST is an array that can contain strings and objects. Any strings represent plain text content to be written to the output. Any objects represent HTML tags.

Tag objects generally look something like this:

```
{
    tag: 'div',
    attrs: {
        class: 'foo'
    },
    content: ['hello world!']
}
```

Tag objects can contain three keys. The `tag` key takes the name of the tag as the value. This can include custom tags. The optional `attrs` key takes an object with key/value pairs representing the attributes of the html tag. A boolean attribute has an empty string as its value. Finally, the optional `content` key takes an array as its value, which is a PostHTML AST. In this manner, the AST is a tree that should be walked recursively.

Options

directives

Type: `Array` Default: `[{name: '!doctype', start: '<', end: '>'}]` Description: *Adds processing of custom directives. Note: The property* `name` *in custom directives can be* `String` *or* `RegExp` *type*

xmlMode

Type: `Boolean` Default: `false` Description: *Indicates whether special tags (*`<script>` *and* `<style>`*) should get special treatment and if "empty" tags (eg.* `<br>`*) can have children. If false, the content of special tags will be text only. For feeds and other XML content (documents that don't consist of HTML), set this to true.*

decodeEntities

Type: `Boolean` Default: `false` Description: *If set to true, entities within the document will be decoded.*

lowerCaseTags

Type: `Boolean` Default: `false` Description: *If set to true, all tags will be lowercased. If* `xmlMode` *is disabled.*

lowerCaseAttributeNames

Type: `Boolean` Default: `false` Description: *If set to true, all attribute names will be lowercased. This has noticeable impact on speed.*

recognizeCDATA

Type: `Boolean` Default: `false` Description: *If set to true, CDATA sections will be recognized as text even if the* `xmlMode` *option is not enabled. NOTE: If* `xmlMode` *is set to* `true` *then CDATA sections will always be recognized as text.*

recognizeSelfClosing

Type: `Boolean` Default: `false` Description: *If set to true, self-closing tags will trigger the* `onclosetag` *event even if* `xmlMode` *is not set to* `true`*. NOTE: If* `xmlMode` *is set to* `true` *then self-closing tags will always be recognized.*