msva / lua-htmlparser

Branch: master ▾ lua-htmlparser / README.md Find file Copy path

msva badges 5bcd0ab on 9 Apr

4 contributors

111 lines (93 sloc) 5.91 KB

build passing coverage 29% License LGPL+

LuaRock "htmlparser"

Parse HTML text into a tree of elements with selectors

Install

Htmlparser is a listed LuaRock. Install using LuaRocks: `luarocks install htmlparser`

Dependencies

Htmlparser depends on Lua 5.1-5.3 or LuaJIT, which provides 5.1-compatible ABI. To be able to run the tests, lunitx also comes along as a LuaRock

Usage

Start off with

```
local htmlparser = require("htmlparser")
```

Then, parse some html:

```
local root = htmlparser.parse(htmlstring)
```

Optionally, you can pass loop-limit value (integer). This value means the deepness of the tree, after which parser will give up. Default value is 1000. Also, global variable `htmlparser_looplimit` is supported (while this optional argument takes priority over global value)

The input to parse may be the contents of a complete html document, or any valid html snippet, as long as all tags are correctly opened and closed. Now, find specific contained elements by selecting:

```
local elements = root:select(selectorstring)
```

Or in shorthand:

```
local elements = root(selectorstring)
```

This wil return a list of elements, all of which are of the same type as the root element, and thus support selecting as well, if ever needed:

```
for _,e in ipairs(elements) do
        print(e.name)
        local subs = e(subselectorstring)
        for _,sub in ipairs(subs) do
                print("", sub.name)
        end
```

```
end
```

The root element is a container for the top level elements in the parsed text, i.e. the `<html>` element in a parsed html document would be a child of the returned root element.

Selectors

Supported selectors are a subset of jQuery's selectors:

- `"*"` all contained elements
- `"element"` elements with the given tagname
- `"#id"` elements with the given id attribute value
- `".class"` elements with the given classname in the class attribute
- `"[attribute]"` elements with an attribute of the given name
- `"[attribute='value']"` equals: elements with the given value for the given attribute
- `"[attribute!='value']"` not equals: elements without the given attribute, or having the attribute, but with a different value
- `"[attribute|='value']"` prefix: attribute's value is given value, or starts with given value, followed by a hyphen (-)
- `"[attribute*='value']"` contains: attribute's value contains given value
- `"[attribute~='value']"` word: attribute's value is a space-separated token, where one of the tokens is the given value
- `"[attribute^='value']"` starts with: attribute's value starts with given value
- `"[attribute$='value']"` ends with: attribute's value ends with given value
- `":not(selectorstring)"` elements not selected by given selector string
- `"ancestor descendant"` elements selected by the `descendant` selector string, that are a descendant of any element selected by the `ancestor` selector string
- `"parent > child"` elements selected by the `child` selector string, that are a child element of any element selected by the `parent` selector string

Selectors can be combined; e.g. `".class:not([attribute]) element.class"`

Element type

All tree elements provide, apart from `:select` and `()`, the following accessors:

Basic

- `.name` the element's tagname
- `.attributes` a table with keys and values for the element's attributes; `{}` if none
- `.id` the value of the element's id attribute; `nil` if not present
- `.classes` an array with the classes listed in element's class attribute; `{}` if none
- `:getcontent()` the raw text between the opening and closing tags of the element; `""` if none
- `.nodes` an array with the element's child elements, `{}` if none
- `.parent` the element that contains this element; `root.parent` is `nil`

Other

- `.index` sequence number of elements in order of appearance; root index is `0`
- `:gettext()` the complete element text, starting with `"<tagname"` and ending with `"/>"` or `"</tagname>"`
- `.level` how deep the element is in the tree; root level is `0`
- `.root` the root element of the tree; `root.root` is `root`
- `.deepernodes` a Set containing all elements in the tree beneath this element, including this element's `.nodes`; `{}` if none
- `.deeperelements` a table with a key for each distinct tagname in `.deepernodes`, containing a Set of all deeper element nodes with that name; `{}` if none
- `.deeperattributes` as `.deeperelements`, but keyed on attribute name
- `.deeperids` as `.deeperelements`, but keyed on id value
- `.deeperclasses` as `.deeperelements`, but keyed on class name

Limitations

- Attribute values in selector strings cannot contain any spaces
- The spaces before and after the `>` in a `parent > child` relation are mandatory
- `<!` elements (including doctype, comments, and CDATA) are not parsed; markup within CDATA is *not* escaped
- Textnodes are no separate tree elements; in `local root = htmlparser.parse("<p>line1<br />line2</p>")`, `root.nodes[1]:getcontent()` is `"line1<br />line2"`, while `root.nodes[1].nodes[1].name` is `"br"`
- No start or end tags are implied when omitted. Only the void elements should not have an end tag
- No validation is done for tag or attribute names or nesting of element types. The list of void elements is in fact the only part specific to HTML

Examples

See `./doc/sample.lua`

Tests

See `./tst/init.lua`

License

LGPL+; see `./doc/LICENSE`