pyDHTMLParser

Release 2.2.2

Mar 16, 2017

Contents

Python version of DHTMLParser DOM HTML/XML parser.

This version is actually much more advanced, D version is kinda unupdated.

CHAPTER 1

What is it?

DHTMLParser is a lightweight HTML/XML parser created for one purpose - quick and easy picking selected tags from DOM.

It can be very useful when you are in need to write own “guerilla” API for some webpage, or a scrapper.

If you want, you can also create HTML/XML documents more easily than by joining strings.

CHAPTER 2

How it works?

The module have just one important function; *parseString()*. This function takes a string and returns a Document Object Model made of linked `HTMLElement` objects (see bellow).

When you call *parseString()*, the string argument is cut into pieces and then evaluated. Each piece is checked and if it looks like it could be HTML element, then it is put into `HTMLElement` object and proper attributes are set (`HTMLElement.__istag` and so on).

Every following element is put into `HTMLElement.childs` list of this element, until proper closing element is found by simple stack mechanism.

Elements with parameters are parsed and parameters are extracted into `HTMLElement.params` property.

Result is array of single linked trees (you can make double linke by calling *makeDoubleLinked()*), which is then encapsulated in a blank `HTMLElement` container, which holds the whole DOM in its `HTMLElement.childs` property.

This container can be then queried using `HTMLElement.find()`, `HTMLElement.findB()`, `HTMLElement.wfind()` and `HTMLElement.match()` methods.

XML

This module is intended mainly for parsing HTML. If you want to parse XML and you don't want parser to guess nonpair tags from source, just set global module property `NONPAIR_TAGS` to blank list.

There is also `cip` argument of *parseString()* function, which makes parameters of the HTML/XML tags case sensitive.

CHAPTER 3

Package content

dhtmlparser API

Most important function here is *parseString()*, which is used to process string and create Document Object Model.

class dhtmlparser.**StateEnum**

content = 0

tag = 1

parameter = 2

comment = 3

dhtmlparser.**first**(*inp_data*)

Return first element from *inp_data*, or raise StopIteration.

Note: This function was created because it works for generators, lists, iterators, tuples and so on same way, which indexing doesn't.

Also it have smaller cost than list(generator)[0], because it doesn't convert whole *inp_data* to list.

Parameters **inp_data** (*iterable*) – Any iterable object.

Raises StopIteration – When the *inp_data* is blank.

dhtmlparser.**parseString**(*txt*, *cip=True*)

Parse string *txt* and return DOM tree consisting of single linked HTMLElement.

Parameters

- **txt** (*str*) – HTML/XML string, which will be parsed to DOM.

- **cip** (*bool, default True*) – Case Insensitive Parameters. Use special dictionary to store `HTMLElement.params` as case insensitive.

Returns Single conteiner HTML element with blank tag, which has whole DOM in it's `HTMLElement.childs` property. This element can be queried using `HTMLElement.find()` functions.

Return type obj

`dhtmlparser.`**makeDoubleLinked**(*dom*, *parent=None*)

Standard output from *dhtmlparser* is single-linked tree. This will make it double-linked.

Parameters

- **dom** (*obj*) – `HTMLElement` instance.
- **parent** (*obj, default None*) – Don't use this, it is used in recursive call.

`dhtmlparser.`**removeTags**(*dom*)

Remove all tags from *dom* and obtain plaintext representation.

Parameters **dom** (*str, obj, array*) – str, HTMLElement instance or array of elements.

Returns Plain string without tags.

Return type str

Submodules

HTMLElement class

This class can be used for parsing or for creating DOM manually.

DOM building

If you want to create DOM from HTMLElements, you can use one of theese four constructors:

```
HTMLElement()
HTMLElement("<tag>")
HTMLElement("<tag>", {"param": "value"})
HTMLElement("tag", {"param": "value"}, [HTMLElement("<tag1>"), ...])
```

Tag or parameter specification parts can be omitted:

```
HTMLElement("<root>", [HTMLElement("<tag1>"), ...])
HTMLElement(
   [HTMLElement("<tag1>"), ...]
)
```

Examples

Blank element

```
>>> from dhtmlparser import HTMLElement
>>> e = HTMLElement()
>>> e
<dhtmlparser.HTMLElement instance at 0x7fb2b39ca170>
>>> print e

>>>
```

Usually, it is better to use `HTMLElement("")`.

Nonpair tag

```
>>> e = HTMLElement("<br>")
>>> e.isNonPairTag()
True
>>> e.isOpeningTag()
False
>>> print e
<br>
```

Notice, that closing tag wasn't automatically created.

Pair tag

```
>>> e = HTMLElement("<tag>")
>>> e.isOpeningTag() # this doesn't check if tag actually is paired, just if it looks␣
↪like opening tag
True
>>> e.isPairTag()    # this does check if element is actually paired
False
>>> e.endtag = HTMLElement("</tag>")
>>> e.isOpeningTag()
True
>>> e.isPairTag()
True
>>> print e
<tag></tag>
```

In short:

```
>>> e = HTMLElement("<tag>")
>>> e.endtag = HTMLElement("</tag>")
```

Or you can always use string parser:

```
>>> e = d.parseString("<tag></tag>")
>>> print e
<tag></tag>
```

But don't forget, that elements returned from parseString() are encapsulated in blank "root" tag:

```
>>> e = d.parseString("<tag></tag>")
>>> e.getTagName()
''
```

```
>>> e.childs[0].tagToString()
'<tag>'
>>> e.childs[0].endtag.tagToString() # referenced thru .endtag property
>>> e.childs[1].tagToString() # manually selected entag from childs - don't use this
'</tag>'
'</tag>
```

Tags with parameters

Tag (with or without <>) can have as dictionary as second parameter.

```
>>> e = HTMLElement("tag", {"param":"value"})  # without <>, because normal text can
↪'t have parameters
>>> print e
<tag param="value">
>>> print e.params  # parameters are accessed thru .params property
{'param': 'value'}
```

Tags with content

You can create content manually:

```
>>> e = HTMLElement("<tag>")
>>> e.childs.append(HTMLElement("content"))
>>> e.endtag = HTMLElement("</tag>")
>>> print e
<tag>content</tag>
```

But there is also easier way:

```
>>> print HTMLElement("tag", [HTMLElement("content")])
<tag>content</tag>
```

or:

```
>>> print HTMLElement("tag", {"some": "parameter"}, [HTMLElement("content")])
<tag some="parameter">content</tag>
```

HTMLElement class API

Quoter submodule

This module provides ability to quote and unquote strings using backslash notation.

dhtmlparser.quoter.**unescape**(*inp*, *quote=""*)

Unescape *quote* in string *inp*.

Example usage:

```
>> unescape('hello \"')
'hello "'
```

Parameters

- **inp** (*str*) – String in which *quote* will be unescaped.
- **quote** (*char, default "*) – Specify which character will be unescaped.

Returns Unescaped string.

Return type str

dhtmlparser.quoter.**escape**(*inp*, *quote=""*)

Escape *quote* in string *inp*.

Example usage:

```
>>> escape('hello "')
'hello \"'
>>> escape('hello \"')
'hello \\"'
```

Parameters

- **inp** (*str*) – String in which *quote* will be escaped.
- **quote** (*char, default "*) – Specify which character will be escaped.

Returns Escaped string.

Return type str

SpecialDict class

class dhtmlparser.specialdict.**SpecialDict**(**args*, ***kwargs*)

Bases: collections.OrderedDict

This dictionary stores items case sensitive, but compare them case INsensitive.

clear()

get(*k*, *d=None*)

has_key(*key*)

iteritems(**args*, ***kwargs*)

iterkeys(**args*, ***kwargs*)

itervalues(**args*, ***kwargs*)

keys(**args*, ***kwargs*)

items(**args*, ***kwargs*)

values(**args*, ***kwargs*)

CHAPTER 4

Interactive example

```
>>> import dhtmlparser as d
>>> dom = d.parseString("""
... <root>
...  <element name="xex" />
... </root>
... """)
>>> print dom
<dhtmlparser.HTMLElement instance at 0x240b320>
>>> dom.getTagName()  # blank, container element
''
```

DOM tree now in memory looks like this:

```
dom == <dhtmlparser.HTMLElement instance at 0x240b320>
 |- .getTagName() == ""
 |- .isTag()      == False
 |- .params       == ""
 |- .openertag    == None
 |- .endtag       == None
 `- .childs       == [<dhtmlparser.HTMLElement instance at 0x2403b90>, <dhtmlparser.
↪HTMLElement instance at 0x2403ab8>, <dhtmlparser.HTMLElement instance at 0x240b050>,
↪ <dhtmlparser.HTMLElement instance at 0x240b248>]
      |
      |- .childs[0]       == <dhtmlparser.HTMLElement instance at 0x2403b90>
      |  |- .getTagName() == "\n"
      |  |- .isTag()      == False
      |  |- .params       == {}
      |  |- .openertag    == None
      |  |- .endtag       == None
      |  `- .childs       == []
      |
      |- .childs[1]         == <dhtmlparser.HTMLElement instance at 0x2403ab8>
      |  |- .getTagName()   == "root"
      |  |- .isTag()        == True
      |  |- .isEndTag()     == False
```

```
      |  |- .isOpeningTag() == True
      |  |- .params         == {}
      |  |- .openertag      == None
      |  |- .endtag         == <dhtmlparser.HTMLElement instance at 0x240b050>
      |  `- .childs         == [<dhtmlparser.HTMLElement instance at 0x2403c68>,
↪<dhtmlparser.HTMLElement instance at 0x2403d88>, <dhtmlparser.HTMLElement instance␣
↪at 0x2403ea8>]
      |     |
      |     |- .childs[0]       == <dhtmlparser.HTMLElement instance at 0x2403c68>
      |     |  |- .getTagName() == '\n '
      |     |  |- .isTag()      == False
      |     |  |- .params       == {}
      |     |  |- .openertag    == None
      |     |  |- .endtag       == None
      |     |  `- .childs       == []
      |     |
      |     |- .childs[1]         == <dhtmlparser.HTMLElement instance at 0x2403d88>
      |     |  |- .getTagName()   == 'element'
      |     |  |- .isTag()        == True
      |     |  |- .isNonPairTag() == True
      |     |  |- .params         == {'name': 'xex'}
      |     |  |- .openertag      == None
      |     |  |- .endtag         == None
      |     |  `- .childs         == []
      |     |
      |     `- .childs[2]       == <dhtmlparser.HTMLElement instance at 0x2403ea8>
      |        |- .getTagName() == '\n'
      |        |- .isTag()      == False
      |        |- .params       == {}
      |        |- .openertag    == None
      |        |- .endtag       == None
      |        `- .childs       == []
      |
      |- .childs[2]       == <dhtmlparser.HTMLElement instance at 0x240b050>
      |  |- .getTagName() == 'root'
      |  |- .isTag()      == True
      |  |- .isEndTag()   == True
      |  |- .params       == {}
      |  |- .openertag    == <dhtmlparser.HTMLElement instance at 0x2403ab8>
      |  |- .endtag       == None
      |  `- .childs       == []
      |
      `- .childs[3]       == <dhtmlparser.HTMLElement instance at 0x240b248>
         |- .getTagName() == '\n'
         |- .isTag()      == False
         |- .params       == {}
         |- .openertag    == None
         |- .endtag       == None
         `- .childs       == []
```

In interactive shell, we can easily verify the tree:

```
>>> dom.childs[1].getTagName()
'root'
>>> dom.childs[1].childs
[<dhtmlparser.HTMLElement instance at 0x2403c68>, <dhtmlparser.HTMLElement instance␣
↪at 0x2403d88>, <dhtmlparser.HTMLElement instance at 0x2403ea8>]
```

and so on..

Now, let say, that you know there is HTML element named `element` and we want to get it, but we don't know where it is. In that case `HTMLElement.find()` will help us:

```
>>> dom.find("element")
[<dhtmlparser.HTMLElement instance at 0x2403d88>]
```

Or when we don't know name of the element, but we know that he has `"name"` parameter (`HTMLElement.params`) set to `"xex"`:

```
>>> dom.find("", fn = lambda x: "name" in x.params and x.params["name"] == "xex")
[<dhtmlparser.HTMLElement instance at 0x2403d88>]
```

Or we want only `<element>` tags with `name="xex"` parameters:

```
>>> dom.find("element", {"name": "xex"})
[<dhtmlparser.HTMLElement instance at 0x2403d88>]
>>> dom.find("element", {"NAME": "xex"})  # parameter names (not values!) are case
↪insensitive by default
[<dhtmlparser.HTMLElement instance at 0x2403d88>]
```

Sources

Source codes can be found at GitHub; https://github.com/Bystroushaak/pyDHTMLParser

Installation

pyDHTMLParser is hosted at pypi, so you can install it using pip:

```
pip install pyDHTMLParser
```

Unittests

Almost everything should be tested. You can run tests using script `run_tests.sh` which can be found at the root of the project:

```
$ ./run_tests.sh
============================= test session starts ==============================
platform linux2 -- Python 2.7.6, pytest-2.8.2, py-1.4.30, pluggy-0.3.1
rootdir: /home/bystrousak/Plocha/Dropbox/c0d3z/python/libs/pyDHTMLParser, inifile:
plugins: cov-1.8.1
collected 68 items

tests/test_escapers.py ..
tests/test_htmlelement_find.py ..........
tests/test_htmlelement_functions.py ..
tests/test_htmlelement_getters.py ............
tests/test_htmlelement_mult_param.py ....
tests/test_htmlelement_one_param.py ......
tests/test_htmlelement_setters.py ...
```

```
tests/test_module.py .................
tests/test_specialdict.py ............

========================== 68 passed in 0.10 seconds ==========================
```

CHAPTER 5

Confused?

If you don't understand how to use it, look at examples in `./examples/`.

If you have questions, you can write me an email to: `bystrousak```@kitakitsune.org`

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

d

Index