pyquery Documentation

Release 1.2.10.dev0

Olivier Lauzanne

June 08, 2015

Contents

pyquery allows you to make jquery queries on xml documents. The API is as much as possible the similar to jquery. pyquery uses lxml for fast xml and html manipulation.

This is not (or at least not yet) a library to produce or interact with javascript code. I just liked the jquery API and I missed it in python so I told myself "Hey let's make jquery in python". This is the result.

The project is being actively developped on a git repository on Github. I have the policy of giving push access to anyone who wants it and then to review what he does. So if you want to contribute just email me.

Please report bugs on the github issue tracker.

CHAPTER 1

Quickstart

You can use the PyQuery class to load an xml document from a string, a lxml document, from a file or from an url:

```
>>> from pyquery import PyQuery as pq
>>> from lxml import etree
>>> import urllib
>>> d = pq("<html></html>")
>>> d = pq(etree.fromstring("<html></html>"))
>>> d = pq(url=your_url)
>>> d = pq(url=your_url,
...        opener=lambda url, **kw: urlopen(url).read())
>>> d = pq(filename=path_to_html_file)
```

Now d is like the $ in jquery:

```
>>> d("#hello")
[<p#hello.hello>]
>>> p = d("#hello")
>>> print(p.html())
Hello world !
>>> p.html("you know <a href='http://python.org/'>Python</a> rocks")
[<p#hello.hello>]
>>> print(p.html())
you know <a href="http://python.org/">Python</a> rocks
>>> print(p.text())
you know Python rocks
```

You can use some of the pseudo classes that are available in jQuery but that are not standard in css such as :first :last :even :odd :eq :lt :gt :checked :selected :file:

```
>>> d('p:first')
[<p#hello.hello>]
```

CHAPTER 2

Full documentation

2.1 Attributes

Using attribute to select specific tag In attribute selectors, the value should be a valid CSS identifier or quoted as string:

```
>>> d = pq("<option value='1'><option value='2'>")
>>> d('option[value="1"]')
[<option>]
```

You can play with the attributes with the jquery API:

```
>>> p = pq('<p id="hello" class="hello"></p>')('p')
>>> p.attr("id")
'hello'
>>> p.attr("id", "plop")
[<p#plop.hello>]
>>> p.attr("id", "hello")
[<p#hello.hello>]
```

Or in a more pythonic way:

```
>>> p.attr.id = "plop"
>>> p.attr.id
'plop'
>>> p.attr["id"] = "ola"
>>> p.attr["id"]
'ola'
>>> p.attr(id='hello', class_='hello2')
[<p#hello.hello2>]
>>> p.attr.class_
'hello2'
>>> p.attr.class_ = 'hello'
```

2.2 CSS

You can play with css classes:

```
>>> p.addClass("toto")
[<p#hello.hello.toto>]
>>> p.toggleClass("titi toto")
[<p#hello.hello.titi>]
```

```
>>> p.removeClass("titi")
[<p#hello.hello>]
```

Or the css style:

```
>>> p.css("font-size", "15px")
[<p#hello.hello>]
>>> p.attr("style")
'font-size: 15px'
>>> p.css({"font-size": "17px"})
[<p#hello.hello>]
>>> p.attr("style")
'font-size: 17px'
```

Same thing the pythonic way ('_' characters are translated to '-'):

```
>>> p.css.font_size = "16px"
>>> p.attr.style
'font-size: 16px'
>>> p.css['font-size'] = "15px"
>>> p.attr.style
'font-size: 15px'
>>> p.css(font_size="16px")
[<p#hello.hello>]
>>> p.attr.style
'font-size: 16px'
>>> p.css = {"font-size": "17px"}
>>> p.attr.style
'font-size: 17px'
```

2.3 Using pseudo classes

2.3.1 :button

Matches all button input elements and the button element:

```
>>> from pyquery import PyQuery
>>> d = PyQuery(('<div><input type="button"/>'
...              '<button></button></div>'))
>>> d(':button')
[<input>, <button>]
```

2.3.2 :checkbox

Matches all checkbox input elements:

```
>>> from pyquery import PyQuery
>>> d = PyQuery('<div><input type="checkbox"/></div>')
>>> d('input:checkbox')
[<input>]
```

2.3.3 :checked

Matches odd elements, zero-indexed:

```
>>> from pyquery import PyQuery
>>> d = PyQuery('<div><input checked="checked"/></div>')
>>> d('input:checked')
[<input>]
```

2.3.4 :child

right is an immediate child of left

2.3.5 :contains()

Matches all elements that contain the given text

```
>>> from pyquery import PyQuery
>>> d = PyQuery('<div><h1/><h1 class="title">title</h1></div>')
>>> d('h1:contains("title")')
[<h1.title>]
```

2.3.6 :descendant

right is a child, grand-child or further descendant of left

2.3.7 :disabled

Matches all elements that are disabled:

```
>>> from pyquery import PyQuery
>>> d = PyQuery('<div><input disabled="disabled"/></div>')
>>> d('input:disabled')
[<input>]
```

2.3.8 :empty

Match all elements that do not contain other elements:

```
>>> from pyquery import PyQuery
>>> d = PyQuery('<div><h1><span>title</span></h1><h2/></div>')
>>> d(':empty')
[<h2>]
```

2.3.9 :enabled

Matches all elements that are enabled:

```
>>> from pyquery import PyQuery
>>> d = PyQuery('<div><input value="foo" /></div>')
>>> d('input:enabled')
[<input>]
```

2.3.10 :eq()

Matches a single element by its index:

```
>>> from pyquery import PyQuery
>>> d = PyQuery('<div><h1 class="first"/><h1 class="last"/></div>')
>>> d('h1:eq(0)')
[<h1.first>]
>>> d('h1:eq(1)')
[<h1.last>]
```

2.3.11 :even

Matches even elements, zero-indexed:

```
>>> from pyquery import PyQuery
>>> d = PyQuery('<div><p></p><p class="last"></p></div>')
>>> d('p:even')
[<p>]
```

2.3.12 :file

Matches all input elements of type file:

```
>>> from pyquery import PyQuery
>>> d = PyQuery('<div><input type="file"/></div>')
>>> d('input:file')
[<input>]
```

2.3.13 :first

Matches the first selected element:

```
>>> from pyquery import PyQuery
>>> d = PyQuery('<div><p class="first"></p><p></p></div>')
>>> d('p:first')
[<p.first>]
```

2.3.14 :gt()

Matches all elements with an index over the given one:

```
>>> from pyquery import PyQuery
>>> d = PyQuery('<div><h1 class="first"/><h1 class="last"/></div>')
>>> d('h1:gt(0)')
[<h1.last>]
```

2.3.15 :header

Matches all header elelements (h1, ..., h6):

```
>>> from pyquery import PyQuery
>>> d = PyQuery('<div><h1>title</h1></div>')
>>> d(':header')
[<h1>]
```

2.3.16 :hidden

Matches all hidden input elements:

```
>>> from pyquery import PyQuery
>>> d = PyQuery('<div><input type="hidden"/></div>')
>>> d('input:hidden')
[<input>]
```

2.3.17 :image

Matches all image input elements:

```
>>> from pyquery import PyQuery
>>> d = PyQuery('<div><input type="image"/></div>')
>>> d('input:image')
[<input>]
```

2.3.18 :input

Matches all input elements:

```
>>> from pyquery import PyQuery
>>> d = PyQuery(('<div><input type="file"/>'
...              '<textarea></textarea></div>'))
>>> d(':input')
[<input>, <textarea>]
```

2.3.19 :last

Matches the last selected element:

```
>>> from pyquery import PyQuery
>>> d = PyQuery('<div><p></p><p class="last"></p></div>')
>>> d('p:last')
[<p.last>]
```

2.3.20 :lt()

Matches all elements with an index below the given one:

```
>>> from pyquery import PyQuery
>>> d = PyQuery('<div><h1 class="first"/><h1 class="last"/></div>')
>>> d('h1:lt(1)')
[<h1.first>]
```

2.3.21 :odd

Matches odd elements, zero-indexed:

```
>>> from pyquery import PyQuery
>>> d = PyQuery('<div><p></p><p class="last"></p></div>')
>>> d('p:odd')
[<p.last>]
```

2.3.22 :parent

Match all elements that contain other elements:

```
>>> from pyquery import PyQuery
>>> d = PyQuery('<div><h1><span>title</span></h1><h1/></div>')
>>> d('h1:parent')
[<h1>]
```

2.3.23 :password

Matches all password input elements:

```
>>> from pyquery import PyQuery
>>> d = PyQuery('<div><input type="password"/></div>')
>>> d('input:password')
[<input>]
```

2.3.24 :pseudo

Translate a pseudo-element.

Defaults to not supporting pseudo-elements at all, but can be overridden by sub-classes

2.3.25 :radio

Matches all radio input elements:

```
>>> from pyquery import PyQuery
>>> d = PyQuery('<div><input type="radio"/></div>')
>>> d('input:radio')
[<input>]
```

2.3.26 :reset

Matches all reset input elements:

```
>>> from pyquery import PyQuery
>>> d = PyQuery('<div><input type="reset"/></div>')
>>> d('input:reset')
[<input>]
```

2.3.27 :selected

Matches all elements that are selected:

```
>>> from pyquery import PyQuery
>>> d = PyQuery('<select><option selected="selected"/></select>')
>>> d('option:selected')
[<option>]
```

2.3.28 :submit

Matches all submit input elements:

```
>>> from pyquery import PyQuery
>>> d = PyQuery('<div><input type="submit"/></div>')
>>> d('input:submit')
[<input>]
```

2.3.29 :text

Matches all text input elements:

```
>>> from pyquery import PyQuery
>>> d = PyQuery('<div><input type="text"/></div>')
>>> d('input:text')
[<input>]
```

2.4 Manipulating

You can also add content to the end of tags:

```
>>> d = pq('<p class="hello" id="hello">you know Python rocks</p>')
>>> d('p').append(' check out <a href="http://reddit.com/r/python"><span>reddit</span></a>')
[<p#hello.hello>]
>>> print(d)
<p class="hello" id="hello">you know Python rocks check out <a href="http://reddit.com/r/python"><spa
```

Or to the beginning:

```
>>> p = d('p')
>>> p.prepend('check out <a href="http://reddit.com/r/python">reddit</a>')
[<p#hello.hello>]
>>> print(p.html())
check out <a href="http://reddit.com/r/python">reddit</a>you know ...
```

Prepend or append an element into an other:

```
>>> d = pq('<html><body><div id="test"><a href="http://python.org">python</a> !</div></body></html>')
>>> p.prependTo(d('#test'))
[<p#hello.hello>]
>>> print(d('#test').html())
<p class="hello" ...
```

Insert an element after another:

```
>>> p.insertAfter(d('#test'))
[<p#hello.hello>]
>>> print(d('#test').html())
<a href="http://python.org">python</a> !
```

Or before:

```
>>> p.insertBefore(d('#test'))
[<p#hello.hello>]
>>> print(d('body').html())
<p class="hello" id="hello">...
```

Doing something for each elements:

```
>>> p.each(lambda i, e: pq(e).addClass('hello2'))
[<p#hello.hello.hello2>]
```

Remove an element:

```
>>> d = pq('<html><body><p id="id">Yeah!</p><p>python rocks !</p></div></html>')
>>> d.remove('p#id')
[<html>]
>>> d('p#id')
[]
```

Remove what's inside the selection:

```
>>> d('p').empty()
[<p>]
```

And you can get back the modified html:

```
>>> print(d)
<html><body><p/></body></html>
```

You can generate html stuff:

```
>>> from pyquery import PyQuery as pq
>>> print(pq('<div>Yeah !</div>').addClass('myclass') + pq('<b>cool</b>'))
<div class="myclass">Yeah !</div><b>cool</b>
```

Remove all namespaces:

```
>>> d = pq('<foo xmlns="http://example.com/foo"></foo>')
>>> d
[<{http://example.com/foo}foo>]
>>> d.remove_namespaces()
[<foo>]
```

2.5 Traversing

Some jQuery traversal methods are supported. Here are a few examples.

You can filter the selection list using a string selector:

```
>>> d = pq('<p id="hello" class="hello"><a/></p><p id="test"><a/></p>')
>>> d('p').filter('.hello')
[<p#hello.hello>]
```

It is possible to select a single element with eq:

```
>>> d('p').eq(0)
[<p#hello.hello>]
```

You can find nested elements:

```
>>> d('p').find('a')
[<a>, <a>]
>>> d('p').eq(1).find('a')
[<a>]
```

Breaking out of a level of traversal is also supported using end:

```
>>> d('p').find('a').end()
[<p#hello.hello>, <p#test>]
>>> d('p').eq(0).end()
[<p#hello.hello>, <p#test>]
>>> d('p').filter(lambda i: i == 1).end()
[<p#hello.hello>, <p#test>]
```

2.6 pyquery – PyQuery complete API

class pyquery.pyquery.**PyQuery**(**args*, ***kwargs*)

The main class

class Fn

Hook for defining custom function (like the jQuery.fn):

```
>>> fn = lambda: this.map(lambda i, el: PyQuery(this).outerHtml())
>>> PyQuery.fn.listOuterHtml = fn
>>> S = PyQuery(
...   '<ol>   <li>Coffee</li>   <li>Tea</li>   <li>Milk</li>   </ol>')
>>> S('li').listOuterHtml()
['<li>Coffee</li>', '<li>Tea</li>', '<li>Milk</li>']
```

PyQuery.**addClass**(*value*)

Alias for *add_class()*

PyQuery.**add_class**(*value*)

Add a css class to elements:

```
>>> d = PyQuery('<div></div>')
>>> d.add_class('myclass')
[<div.myclass>]
>>> d.addClass('myclass')
[<div.myclass>]
```

PyQuery.**after**(*value*)

add value after nodes

PyQuery.**append**(*value*)

append value to each nodes

PyQuery.**appendTo**(*value*)

Alias for *append_to()*

PyQuery.**append_to**(*value*)

append nodes to value

PyQuery.**base_url**
Return the url of current html document or None if not available.

PyQuery.**before**(*value*)
insert value before nodes

PyQuery.**children**(*selector=None*)
Filter elements that are direct children of self using optional selector:

```
>>> d = PyQuery('<span><p class="hello">Hi</p><p>Bye</p></span>')
>>> d
[<span>]
>>> d.children()
[<p.hello>, <p>]
>>> d.children('.hello')
[<p.hello>]
```

PyQuery.**clone**()
return a copy of nodes

PyQuery.**closest**(*selector=None*)

```
>>> d = PyQuery(
...  '<div class="hello"><p>This is a '
...  '<strong class="hello">test</strong></p></div>')
>>> d('strong').closest('div')
[<div.hello>]
>>> d('strong').closest('.hello')
[<strong.hello>]
>>> d('strong').closest('form')
[]
```

PyQuery.**contents**()
Return contents (with text nodes):

```
>>> d = PyQuery('hello <b>bold</b>')
>>> d.contents()
['hello ', <Element b at ...>]
```

PyQuery.**each**(*func*)
apply func on each nodes

PyQuery.**empty**()
remove nodes content

PyQuery.**encoding**
return the xml encoding of the root element

PyQuery.**end**()
Break out of a level of traversal and return to the parent level.

```
>>> m = '<p><span><em>Whoah!</em></span></p><p><em> there</em></p>'
>>> d = PyQuery(m)
>>> d('p').eq(1).find('em').end().end()
[<p>, <p>]
```

PyQuery.**eq**(*index*)
Return PyQuery of only the element with the provided index:

```
>>> d = PyQuery('<p class="hello">Hi</p><p>Bye</p><div></div>')
>>> d('p').eq(0)
[<p.hello>]
>>> d('p').eq(1)
[<p>]
>>> d('p').eq(2)
[]
```

PyQuery.**extend**(*other*)
Extend with anoter PyQuery object

PyQuery.**filter**(*selector*)
Filter elements in self using selector (string or function):

```
>>> d = PyQuery('<p class="hello">Hi</p><p>Bye</p>')
>>> d('p')
[<p.hello>, <p>]
>>> d('p').filter('.hello')
[<p.hello>]
>>> d('p').filter(lambda i: i == 1)
[<p>]
>>> d('p').filter(lambda i: PyQuery(this).text() == 'Hi')
[<p.hello>]
>>> d('p').filter(lambda i, this: PyQuery(this).text() == 'Hi')
[<p.hello>]
```

PyQuery.**find**(*selector*)
Find elements using selector traversing down from self:

```
>>> m = '<p><span><em>Whoah!</em></span></p><p><em> there</em></p>'
>>> d = PyQuery(m)
>>> d('p').find('em')
[<em>, <em>]
>>> d('p').eq(1).find('em')
[<em>]
```

PyQuery.**hasClass**(*name*)
Alias for *has_class()*

PyQuery.**has_class**(*name*)
Return True if element has class:

```
>>> d = PyQuery('<div class="myclass"></div>')
>>> d.has_class('myclass')
True
>>> d.hasClass('myclass')
True
```

PyQuery.**height**(*value=<NoDefault>*)
set/get height of element

PyQuery.**hide**()
remove display:none to elements style

```
>>> print(PyQuery('<div style="display:none;"/>').hide())
<div style="display: none"/>
```

PyQuery.**html**(*value=<NoDefault>*, ***kwargs*)
Get or set the html representation of sub nodes.

Get the text value:

```
>>> d = PyQuery('<div><span>toto</span></div>')
>>> print(d.html())
<span>toto</span>
```

Extra args are passed to `lxml.etree.tostring`:

```
>>> d = PyQuery('<div><span></span></div>')
>>> print(d.html())
<span/>
>>> print(d.html(method='html'))
<span></span>
```

Set the text value:

```
>>> d.html('<span>Youhou !</span>')
[<div>]
>>> print(d)
<div><span>Youhou !</span></div>
```

PyQuery.**insertAfter**(*value*)
Alias for *insert_after()*

PyQuery.**insertBefore**(*value*)
Alias for *insert_before()*

PyQuery.**insert_after**(*value*)
insert nodes after value

PyQuery.**insert_before**(*value*)
insert nodes before value

PyQuery.**is_**(*selector*)
Returns True if selector matches at least one current element, else False:

```
>>> d = PyQuery('<p class="hello"><span>Hi</span></p><p>Bye</p>')
>>> d('p').eq(0).is_('.hello')
True
```

```
>>> d('p').eq(0).is_('span')
False
```

```
>>> d('p').eq(1).is_('.hello')
False
```

PyQuery.**items**(*selector=None*)
Iter over elements. Return PyQuery objects:

```
>>> d = PyQuery('<div><span>foo</span><span>bar</span></div>')
>>> [i.text() for i in d.items('span')]
['foo', 'bar']
>>> [i.text() for i in d('span').items()]
['foo', 'bar']
>>> list(d.items('a')) == list(d('a').items())
True
```

PyQuery.**make_links_absolute**(*base_url=None*)
Make all links absolute.

PyQuery.**map**(*func*)
Returns a new PyQuery after transforming current items with func.

func should take two arguments - 'index' and 'element'. Elements can also be referred to as 'this' inside of func:

```
>>> d = PyQuery('<p class="hello">Hi there</p><p>Bye</p><br />')
>>> d('p').map(lambda i, e: PyQuery(e).text())
['Hi there', 'Bye']

>>> d('p').map(lambda i, e: len(PyQuery(this).text()))
[8, 3]

>>> d('p').map(lambda i, e: PyQuery(this).text().split())
['Hi', 'there', 'Bye']
```

PyQuery.**nextAll**(*selector=None*)
Alias for *next_all()*

PyQuery.**next_all**(*selector=None*)

```
>>> h = '<span><p class="hello">Hi</p><p>Bye</p><img scr=""/></span>'
>>> d = PyQuery(h)
>>> d('p:last').next_all()
[<img>]
>>> d('p:last').nextAll()
[<img>]
```

PyQuery.**not_**(*selector*)
Return elements that don't match the given selector:

```
>>> d = PyQuery('<p class="hello">Hi</p><p>Bye</p><div></div>')
>>> d('p').not_('.hello')
[<p>]
```

PyQuery.**outerHtml**()
Alias for *outer_html()*

PyQuery.**outer_html**()
Get the html representation of the first selected element:

```
>>> d = PyQuery('<div><span class="red">toto</span> rocks</div>')
>>> print(d('span'))
<span class="red">toto</span> rocks
>>> print(d('span').outer_html())
<span class="red">toto</span>
>>> print(d('span').outerHtml())
<span class="red">toto</span>

>>> S = PyQuery('<p>Only <b>me</b> & myself</p>')
>>> print(S('b').outer_html())
<b>me</b>
```

PyQuery.**parents**(*selector=None*)

```
>>> d = PyQuery('<span><p class="hello">Hi</p><p>Bye</p></span>')
>>> d('p').parents()
[<span>]
>>> d('.hello').parents('span')
[<span>]
```

```
>>> d('.hello').parents('p')
[]
```

PyQuery.**prepend**(*value*)
prepend value to nodes

PyQuery.**prependTo**(*value*)
Alias for *prepend_to()*

PyQuery.**prepend_to**(*value*)
prepend nodes to value

PyQuery.**prevAll**(*selector=None*)
Alias for *prev_all()*

PyQuery.**prev_all**(*selector=None*)

```
>>> h = '<span><p class="hello">Hi</p><p>Bye</p><img scr=""/></span>'
>>> d = PyQuery(h)
>>> d('p:last').prev_all()
[<p.hello>]
>>> d('p:last').prevAll()
[<p.hello>]
```

PyQuery.**remove**(*expr=<NoDefault>*)
Remove nodes:

```
>>> h = '<div>Maybe <em>she</em> does <strong>NOT</strong> know</div>'
>>> d = PyQuery(h)
>>> d('strong').remove()
[<strong>]
>>> print(d)
<div>Maybe <em>she</em> does  know</div>
```

PyQuery.**removeAttr**(*name*)
Alias for *remove_attr()*

PyQuery.**removeClass**(*value*)
Alias for *remove_class()*

PyQuery.**remove_attr**(*name*)
Remove an attribute:

```
>>> d = PyQuery('<div id="myid"></div>')
>>> d.remove_attr('id')
[<div>]
>>> d.removeAttr('id')
[<div>]
```

PyQuery.**remove_class**(*value*)
Remove a css class to elements:

```
>>> d = PyQuery('<div class="myclass"></div>')
>>> d.remove_class('myclass')
[<div>]
>>> d.removeClass('myclass')
[<div>]
```

PyQuery.**remove_namespaces**()
Remove all namespaces:

```
>>> doc = PyQuery('<foo xmlns="http://example.com/foo"></foo>')
>>> doc
[<{http://example.com/foo}foo>]
>>> doc.remove_namespaces()
[<foo>]
```

PyQuery.**replaceAll**(*expr*)
Alias for *replace_all()*

PyQuery.**replaceWith**(*value*)
Alias for *replace_with()*

PyQuery.**replace_all**(*expr*)
replace nodes by expr

PyQuery.**replace_with**(*value*)
replace nodes by value:

```
>>> doc = PyQuery("<html><div /></html>")
>>> node = PyQuery("<span />")
>>> child = doc.find('div')
>>> child.replace_with(node)
```

[ ] >>> print(doc) <html></html>

PyQuery.**root**
return the xml root element

PyQuery.**show**()
add display:block to elements style

```
>>> print(PyQuery('<div />').show())
<div style="display: block"/>
```

PyQuery.**siblings**(*selector=None*)

```
>>> h = '<span><p class="hello">Hi</p><p>Bye</p><img scr=""/></span>'
>>> d = PyQuery(h)
>>> d('.hello').siblings()
[<p>, <img>]
>>> d('.hello').siblings('img')
[<img>]
```

PyQuery.**text**(*value=<NoDefault>*)
Get or set the text representation of sub nodes.

Get the text value:

```
>>> doc = PyQuery('<div><span>toto</span><span>tata</span></div>')
>>> print(doc.text())
toto tata
```

Set the text value:

```
>>> doc.text('Youhou !')
[<div>]
>>> print(doc)
<div>Youhou !</div>
```

PyQuery.**toggleClass**(*value*)
Alias for *toggle_class()*

PyQuery.**toggle_class**(*value*)
Toggle a css class to elements

```
>>> d = PyQuery('<div></div>')
>>> d.toggle_class('myclass')
[<div.myclass>]
>>> d.toggleClass('myclass')
[<div>]
```

PyQuery.**val**(*value=<NoDefault>*)
Set the attribute value:

```
>>> d = PyQuery('<input />')
>>> d.val('Youhou')
[<input>]
```

Get the attribute value:

```
>>> d.val()
'Youhou'
```

PyQuery.**width**(*value=<NoDefault>*)
set/get width of element

PyQuery.**wrap**(*value*)
A string of HTML that will be created on the fly and wrapped around each target:

```
>>> d = PyQuery('<span>youhou</span>')
>>> d.wrap('<div></div>')
[<div>]
>>> print(d)
<div><span>youhou</span></div>
```

PyQuery.**wrapAll**(*value*)
Alias for *wrap_all()*

PyQuery.**wrap_all**(*value*)
Wrap all the elements in the matched set into a single wrapper element:

```
>>> d = PyQuery('<div><span>Hey</span><span>you !</span></div>')
>>> print(d('span').wrap_all('<div id="wrapper"></div>'))
<div id="wrapper"><span>Hey</span><span>you !</span></div>

>>> d = PyQuery('<div><span>Hey</span><span>you !</span></div>')
>>> print(d('span').wrapAll('<div id="wrapper"></div>'))
<div id="wrapper"><span>Hey</span><span>you !</span></div>
```

PyQuery.**xhtml_to_html**()
Remove xhtml namespace:

```
>>> doc = PyQuery(
...         '<html xmlns="http://www.w3.org/1999/xhtml"></html>')
>>> doc
[<{http://www.w3.org/1999/xhtml}html>]
>>> doc.xhtml_to_html()
[<html>]
```

2.7 Scraping

PyQuery is able to load an html document from a url:

```
>>> pq(your_url)
[<html>]
```

By default it uses python's urllib.

If requests is installed then it will use it. This allow you to use most of requests parameters:

```
>>> pq(your_url, headers={'user-agent': 'pyquery'})
[<html>]

>>> pq(your_url, {'q': 'foo'}, method='post', verify=True)
[<html>]
```

2.8 `pyquery.ajax` – PyQuery AJAX extension

You can query some wsgi app if WebOb is installed (it's not a pyquery dependencie). IN this example the test app returns a simple input at */* and a submit button at */submit*:

```
>>> d = pq('<form></form>', app=input_app)
>>> d.append(d.get('/'))
[<form>]
>>> print(d)
<form><input name="youyou" type="text" value=""/></form>
```

The app is also available in new nodes:

```
>>> d.get('/').app is d.app is d('form').app
True
```

You can also request another path:

```
>>> d.append(d.get('/submit'))
[<form>]
>>> print(d)
<form><input name="youyou" type="text" value=""/><input type="submit" value="OK"/></form>
```

If restkit is installed, you are able to get url directly with a HostProxy app:

```
>>> a = d.get(your_url)
>>> a
[<html>]
```

You can retrieve the app response:

```
>>> print(a.response.status)
200 OK
```

The response attribute is a WebOb Response

2.8.1 Api

2.9 Tips

2.9.1 Making links absolute

You can make links absolute which can be usefull for screen scrapping:

```
>>> d = pq(url=your_url, parser='html')
>>> d('form').attr('action')
'/form-submit'
>>> d.make_links_absolute()
[<html>]
```

2.9.2 Using different parsers

By default pyquery uses the lxml xml parser and then if it doesn't work goes on to try the html parser from lxml.html. The xml parser can sometimes be problematic when parsing xhtml pages because the parser will not raise an error but give an unusable tree (on w3c.org for example).

You can also choose which parser to use explicitly:

```
>>> pq('<html><body><p>toto</p></body></html>', parser='xml')
[<html>]
>>> pq('<html><body><p>toto</p></body></html>', parser='html')
[<html>]
>>> pq('<html><body><p>toto</p></body></html>', parser='html_fragments')
[<p>]
```

The html and html_fragments parser are the ones from lxml.html.

2.10 Testing

If you want to run the tests that you can see above you should do:

```
$ git clone git://github.com/gawel/pyquery.git
$ cd pyquery
$ python bootstrap.py
$ bin/buildout install tox
$ bin/tox
```

You can build the Sphinx documentation by doing:

```
$ cd docs
$ make html
```

2.11 Future

- SELECTORS: done
- ATTRIBUTES: done
- CSS: done

- HTML: done
- MANIPULATING: missing the wrapInner method
- TRAVERSING: about half done
- EVENTS: nothing to do with server side might be used later for automatic ajax
- CORE UI EFFECTS: did hide and show the rest doesn't really makes sense on server side
- AJAX: some with wsgi app

2.12 News

2.12.1 1.2.10 (unreleased)

- Fixed #98: contains act like jQuery

2.12.2 1.2.9 (2014-08-22)

- Support for keyword arguments in PyQuery custom functions
- Fixed #78: items must take care or the parent
- Fixed #65 PyQuery.make_links_absolute() no longer creates 'href' attribute when it isn't there
- Fixed #19. `is_()` was broken.
- Fixed #9. `.replaceWith(PyQuery element)` raises error
- Remove official python3.2 support (mostly because of 3rd party semi-deps)

2.12.3 1.2.8 (2013-12-21)

- Fixed #22: Open by filename fails when file contains invalid xml
- Bug fix in .remove_class()

2.12.4 1.2.7 (2013-12-21)

- Use pep8 name for methods but keep an alias for camel case method. Eg: remove_attr and removeAttr works Fix #57
- .text() now return an empty string instead of None if there is no text node. Fix #45
- Fixed #23: removeClass adds class attribute to elements which previously lacked one

2.12.5 1.2.6 (2013-10-11)

README_fixt.py was not include in the release. Fix #54.

2.12.6 1.2.5 (2013-10-10)

cssselect compat. See https://github.com/SimonSapin/cssselect/pull/22

tests improvments. no longer require a eth connection.

fix #55

2.12.7 1.2.4

Moved to github. So a few files are renamed from .txt to .rst

Added .xhtml_to_html() and .remove_namespaces()

Use requests to fetch urls (if available)

Use restkit's proxy instead of Paste (which will die with py3)

Allow to open https urls

python2.5 is no longer supported (may work, but tests are broken)

2.12.8 1.2.3

Allow to pass this in .filter() callback

Add .contents() .items()

Add tox.ini

Bug fixes: fix #35 #55 #64 #66

2.12.9 1.2.2

Fix cssselectpatch to match the newer implementation of cssselect. Fixes issue #62, #52 and #59 (Haoyu Bai)

Fix issue #37 (Caleb Burns)

2.12.10 1.2.1

Allow to use a custom css translator.

Fix issue 44: case problem with xml documents

2.12.11 1.2

PyQuery now use cssselect. See issue 43.

Fix issue 40: forward .html() extra arguments to `lxml.etree.tostring`

2.12.12 1.1.1

Minor release. Include test file so you can run tests from the tarball.

2.12.13 1.1

fix issues 30, 31, 32 - py3 improvements / webob 1.2+ support

2.12.14 1.0

fix issues 24

2.12.15 0.7

Python 3 compatible

Add __unicode__ method

Add root and encoding attribute

fix issues 19, 20, 22, 23

2.12.16 0.6.1

Move README.txt at package root

Add CHANGES.txt and add it to long_description

2.12.17 0.6

Added PyQuery.outerHtml

Added PyQuery.fn

Added PyQuery.map

Change PyQuery.each behavior to reflect jQuery api

CHAPTER 3

More documentation

First there is the Sphinx documentation here. Then for more documentation about the API you can use the jquery website. The reference I'm now using for the API is ... the color cheat sheet. Then you can always look at the code.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

p

S

T

V

W

X