html5-quickstart

Apr 12, 2023

Contents

A fast implementation of the HTML 5 parsing spec for Python. Parsing is done in C using a variant of the gumbo parser. The gumbo parse tree is then transformed into an lxml tree, also in C, yielding parse times that can be **a thirtieth** of the html5lib parse times. That is a speedup of **30x**. This differs, for instance, from the gumbo python bindings, where the initial parsing is done in C but the transformation into the final tree is done in python.

CHAPTER 1

Installation

1.1 Unix

On a Unix-y system, with a working C99 compiler, simply run:

```
pip install --no-binary lxml html5-parser
```

It is important that lxml is installed with the `--no-binary` flag. This is because without it, lxml uses a static copy of libxml2. For html5-parser to work it must use the same libxml2 implementation as lxml. This is only possible if libxml2 is loaded dynamically.

You can setup html5-parser to run from a source checkout as follows:

```
git clone https://github.com/kovidgoyal/html5-parser && cd html5-parser
pip install --no-binary lxml 'lxml>=3.8.0' --user
python setup.py develop --user
```

1.2 Windows

On Windows, installation is a little more involved. There is a 200 line script that is used to install html5-parser and all its dependencies on the windows continuous integration server. Using that script installation can be done by running the following commands in a Visual Studio 2015 Command prompt:

```
python.exe win-ci.py install_deps
python.exe win-ci.py test
```

This will install all dependencies and html5-parser in the `sw` sub-directory. You will need to add `sw\bin` to `PATH` and `sw\python\Lib\site-packages` to `PYTHONPATH`. Or copy the files into your system python's directories.

CHAPTER 2

Quickstart

To use html5-parser in your code, after installing it simply do:

```python
from html5_parser import parse
from lxml.etree import tostring
root = parse(some_html)
print(tostring(root))
```

See the `html5_parser.parse()` function documentation for more details on parsing. To learn how to use the parsed lxml tree in your program, see the lxml tutorial.

CHAPTER 3

XHTML

html5-parser has the ability to parse XHTML documents as well. It will preserve namespace information even for namespaces not defined in the HTML 5 spec. You can ask it to treat the input html as possibly XHTML by using the `maybe_xhtml` parameter to the *html5_parser.parse()* function. For example:

```
<p xmlns:n="my namespace"><n:tag n:attr="a" />
```

becomes

```
<html>
    <head/>
    <body>
        <p xmlns:n="my namespace">
            <n:tag n:attr="a"/>
        </p>
    </body>
</html>
```

This is useful when try to parse a XHTML document that is not well-formed and so cannot be parsed by a regular XML parser.

CHAPTER 4

API documentation

The API of html5-parser is a single function, `parse()`.

`html5_parser.parse`(*html*, *transport_encoding=None*, *namespace_elements=False*, *tree-builder=u'lxml'*, *fallback_encoding=None*, *keep_doctype=True*, *maybe_xhtml=False*, *return_root=True*, *line_number_attr=None*, *sanitize_names=True*, *stack_size=16384*, *fragment_context=None*)

Parse the specified `html` and return the parsed representation.

Parameters

- **`html`** – The HTML to be parsed. Can be either bytes or a unicode string.
- **`transport_encoding`** – If specified, assume the passed in bytes are in this encoding. Ignored if `html` is unicode.
- **`namespace_elements`** – Add XML namespaces when parsing so that the resulting tree is XHTML.
- **`treebuilder`** – The type of tree to return. Note that only the lxml treebuilder is fast, as all other treebuilders are implemented in python, not C. Supported values are:
  - lxml – the default, and fastest
  - lxml_html – tree of lxml.html.HtmlElement, same speed as lxml (new in *0.4.10*)
  - etree (the python stdlib `xml.etree.ElementTree`)
  - dom (the python stdlib `xml.dom.minidom`)
  - soup – BeautifulSoup, which must be installed or it will raise an `ImportError`
- **`fallback_encoding`** – If no encoding could be detected, then use this encoding. Defaults to an encoding based on system locale.
- **`keep_doctype`** – Keep the <DOCTYPE> (if any).
- **`maybe_xhtml`** – Useful when it is unknown if the HTML to be parsed is actually XHTML. Changes the HTML 5 parsing algorithm to be more suitable for XHTML. In particular handles self-closed CDATA elements. So a `<title/>` or `<style/>` in the HTML will not

completely break parsing. Also preserves namespaced tags and attributes even for namespaces not supported by HTML 5 (this works only with the `lxml` and `lxml_html` treebuilders). Note that setting this also implicitly sets `namespace_elements`.

- **`return_root`** – If True, return the root node of the document, otherwise return the tree object for the document.
- **`line_number_attr`** – The optional name of an attribute used to store the line number of every element. If set, this attribute will be added to each element with the element's line number.
- **`sanitize_names`** – Ensure tag and attributes contain only ASCII alphanumeric charactes, underscores, hyphens and periods. This ensures that the resulting tree is also valid XML. Any characters outside this set are replaced by underscores. Note that this is not strictly HTML 5 spec compliant, so turn it off if you need strict spec compliance.
- **`stack_size`** – The initial size (number of items) in the stack. The default is sufficient to avoid memory allocations for all but the largest documents.
- **`fragment_context`** – the tag name under which to parse the HTML when the html is a fragment. Common choices are `div` or `body`. To use SVG or MATHML tags prefix the tag name with `svg:` or `math:` respectively. Note that currently using a non-HTML fragment_context is not supported. New in *0.4.10*.

CHAPTER 5

Comparison with html5lib

Before doing the actual comparison, let me say that html5lib is a great project. It was a pioneer of HTML 5 parsing and I have used it myself for many years. However, being written in pure python, it cannot help but be slow.

5.1 Benchmarks

There is a benchmark script named benchmark.py that compares the parse times for parsing a large (~ 5.7MB) HTML document in html5lib and html5-parser. The results on my system (using python 3) show a speedup of **37x**. The output from the script on my system is:

```
Testing with HTML file of 5,956,815 bytes
Parsing 100 times with html5-parser
html5-parser took an average of: 0.397 seconds to parse it
Parsing 10 times with html5-parser-to-soup
html5-parser-to-soup took an average of: 1.685 seconds to parse it
Parsing 10 times with html5lib
html5lib took an average of: 13.906 seconds to parse it
Parsing 10 times with BeautifulSoup-with-html5lib
BeautifulSoup-with-html5lib took an average of: 12.683 seconds to parse it
Parsing 10 times with BeautifulSoup-with-lxml
BeautifulSoup-with-lxml took an average of: 3.826 seconds to parse it

Results are below. They show how much faster html5-parser is than each
specified parser. Note that there are two additional considerations:
what the final tree is and whether the parsing supports the HTML 5
parsing algorithm. The most apples-to-apples comparison is when the
final tree is lxml and HTML 5 parsing is supported by the parser being
compared to. Note that in this case, we have the largest speedup. In
all other cases, speedup is less because of the overhead of building
the final tree in python instead of C or because the compared parser
does not use the HTML 5 parsing algorithm or both.

Parser            |Tree              |Supports HTML 5   |Speedup (factor)  |
```

(continues on next page)

(continued from previous page)

```
================================================================================
html5lib          |lxml              |yes               |35                |
soup+html5lib     |BeautifulSoup     |yes               |8                 |
soup+lxml.html    |BeautifulSoup     |no                |2                 |
```

There is further potential for speedup. Currently the gumbo subsystem uses its own data structures to store parse results and these are converted to libxml2 data structures in a second pass after parsing completes. By modifying gumbo to use libxml2 data structures directly, there could be significant speed and memory usage gains.

5.2 XML namespace handling

html5lib has truly horrible handling of namespaces. There is even a source-code file in it named `_ihatexml.py`. Compare the result of parsing and pretty printing the following simple HTML fragment (pretty printing is done via lxml in both cases).

```
<p>xxx<svg><image xlink:href="xxx"></svg><p>yyy
```

With **html5lib**:

```
<html:html xmlns:html="http://www.w3.org/1999/xhtml">
    <html:head/>
    <html:body>
        <html:p>xxx<ns0:svg xmlns:ns0="http://www.w3.org/2000/svg"><ns0:image␣
↪xmlns:ns1="http://www.w3.org/1999/xlink" ns1:href="xxx"/></ns0:svg></html:p>
        <html:p>yyy</html:p>
    </html:body>
</html:html>
```

With **html5-parser**:

```
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:xlink="http://www.w3.org/1999/xlink">
    <head/>
    <body>
        <p>xxx<svg xmlns="http://www.w3.org/2000/svg"><image xlink:href="xxx"/></svg>
↪</p>
        <p>yyy</p>
    </body>
</html>
```

While both outputs are technically correct, the output produced via html5-parser is much easier to read and much closer to what an actual human would write. In particular, notice the unnecessary use of prefixes in the html5lib output, as well as the ugly `ns0` anonymous prefix for the svg namespace.

html5-parser also has the ability to optionally preserve namespace information even for namespaces not defined in the HTML 5 standard. See the *XHTML* section for more information.

CHAPTER 6

Safety and correctness

The HTML parser is based on the gumbo parser which has undergone a Google security review and been tested on 2.5 billion pages from the Google cache. In addition, html5-parser passes (almost) all the tests from the html5lib test suite.

Finally, html5-parser is compiled with `-pedantic-errors -Wall -Werror` and the test suite, consisting of thousands of tests, is run using the address and undefined behavior sanitizers. Continuous integration testing is done on three major OSes and four different compilers.

Index