arne brodowski

Write your own ReStructuredText-Writer

Some time ago I decided to use ReStructuredText (http://docutils.sf.net) as the universal Format for nearly all content on my website. This includes not only all pages but also all blog entries like this one you are currently reading.

Soon after I had implemented the Django backend for my website and started hacking together some HTML and CSS, I felt the need to manipulate the generated HTML from the docutils package. Don't get me wrong, there's nothing wrong with the HTML docutils created, but I'm a fan of very minimalistic but still semantic markup.

Let me give you a few examples. While producing content for my website I make sure, that I only have one `<h1>` per page, which is the title of the page.

Docutils creates the following HTML:

```
<h1 class="title">Document Title</h1>
```

But for me the extra class is redundant, because the h1 is unique on my pages. The output I want is:

```
<h1>Document Title</h1>
```

Doctils also wraps every section of the document into an extra div with an id to make it possible to provide anchor-links to specific sections. This is a nice-to-feature but I don't like it because it's not minimalistic, it creates too much extra-markup for me.

Different people may have different needs for the generated HTML markup, so if you are like me and want to change the HTML produced by docutils then read on and see how I've done it.

Docutils Internals

Docutils can generate different output formats. These formats not only include HTML4 and LaTeX but also a specialized HTML output for the Python Enhancement Proposals (PEP) and for the S5 HTML slideshow system.

The components which produce the output are called **Writers** and inherit from `docutils.writers.Writer` (or any existing Writer).

Everything besides HTML output is out the scope of this blog entry, so if we want to produce a modified HTML writer, than we should inherit from `docutils.writers.html4css1.Writer`. This writer has an attribute `translator_class` which assigns a Translator-class to the writer. The Translator is build by inheriting from `docutils.nodes.NodeVisitor` which implements the Visitor pattern for document tree traversals defined by Gamma, Helm, Johnson and Vlissides in *Design Patterns: Elements of Reusable Object-Oriented Software* (Addison-Wesley, 1995). Here is an excerpt from the docstring:

> Each node class has corresponding methods, doing nothing by default; override individual methods for specific and useful behaviour. The *dispatch_visit()* method is called by *Node.walk()* upon entering a node. *Node.walkabout()* also calls the *dispatch_departure()* method before exiting a node.
>
> The dispatch methods call "`visit_` + node class name" or "`depart_` + node class name", resp.
>
> This is a base class for visitors whose `visit_...` & `depart_...` methods should be implemented for *all* node types encountered (such as for *docutils.writers.Writer* subclasses). Unimplemented methods will raise exceptions.

To change the generated HTML output we will have to write our own HTML-Translator class which will inherit from the default `docutils.writers.html4css1.HTMLTranslator`.

Writing Code

We start by subclassing the Writer class to specify our own Translator class:

```
from docutils.writers import html4css1

class MyHTMLWriter(html4css1.Writer):
"""
This docutils writer will use the MyHTMLTranslator class below.

"""
def __init__(self):
    html4css1.Writer.__init__(self)
    self.translator_class = MyHTMLTranslator
```

The next step is to provide our own Translator by subclassing the `html4css1.HTMLTranslator` class.

```
class MyHTMLTranslator(html4css1.HTMLTranslator):
"""
This is a translator class for the docutils system.
It will produce a minimal set of html output.
(No extry divs, classes oder ids.)

"""
```

That's enough 'infrastructure', every code written from now on (in the `MyHTMLTranslator` class will have direct impact on the produced HTML output.

As the excerpt from the docstring above already mentioned, the Translator class will have a number of methods starting with either `visit_` or `depart_`. These methods will will be called while the document tree (created by the docutils ReStruncturedText parser) is traversed and will add stuff to the output. The Translator has an attribute `body` where the content for the output is stored. Additionally there is a `context` attribute, which can be used to store values between the visit- and depart-methods without writing to the `body`. An example below will hopefully make this clearer.

As an example here are the original `visit_block_quote` and `depart_block_quote` methods of the `html4css1.HTMLTranslator`:

```
def visit_block_quote(self, node):
    self.body.append(self.starttag(node, 'blockquote'))

def depart_block_quote(self, node):
    self.body.append('</blockquote>\n')
```

This turns an indented paragraph in the ReStructuredText source into a `<blockquote>` element in the HTML output. Everything inside the blockquote (the text) is further processed as a pragraph node.

Every paragraph is processed by these methods:

```
def visit_paragraph(self, node):
    if self.should_be_compact_paragraph(node):
        self.context.append('')
    else:
        self.body.append(self.starttag(node, 'p', ''))
        self.context.append('</p>\n')

def depart_paragraph(self, node):
    self.body.append(self.context.pop())
```

There are two things to note here: First the *_paragraph-methods use the `context` attribute with `append()` and `pop()` as a stack to maintain a state between the visit and the depart method. The `visit_paragraph` calls `should_be_compact_paragraph` to decide if a <p> tag is appropriate for this paragraph or not. Depending on this decision a starttag (<p>) may be written to the `body` and the correct closetag is appended to the `context` so that `depart_paragraph` can append it to the `body`. This mechanism is needed, because the text in the paragraph may contain further elements, which are processed by other methods. If you use the `context` to store values please ensure that you use `context.pop()` exactly as often as you used `context.append()` so that other methods are not confused by values left-over in the stack.

Now comes our first modification to the HTML output. The html4css1 Writer will happily compact parapgraphs inside the blockquote element, if it contains only a single paragraph, but XHTML 1.0 Strict requires that the text inside a <blockquote> tag is always wrapped in a <p> tag. To achive this, we add a method `should_be_compact_paragraph` to the `MyHTMLTranslator` class defined above:

```
# add this to the imports: from docutils import nodes

def should_be_compact_paragraph(self, node):
    if(isinstance(node.parent, nodes.block_quote)):
        return 0
    return html4css1.HTMLTranslator.should_be_compact_paragraph(self, node)
```

Another example of a modification, which I made in the MyHTMLTranslator class is the handling of document sections. As I explained above, I don't need a div around every section, so let's have a look at the original code:

```
def visit_section(self, node):
    self.section_level += 1
    self.body.append(
        self.starttag(node, 'div', CLASS='section'))

def depart_section(self, node):
    self.section_level -= 1
    self.body.append('</div>\n')
```

And here is my modified version, which don't add the divs around the sections:

```
def visit_section(self, node):
    self.section_level += 1

def depart_section(self, node):
    self.section_level -= 1
```

As you can see, I've just removed the parts, where the div is added to the `body`.

Using the Writer with Django

To use my custom ReStructuredText Writer on my Django powered website I started with the bundled templatetag. Django provides a `restructuredtext` templatetag in the `django.contrib.markup` package, which looks like this (some linebreaks added for better readability, marked with a \):

```
def restructuredtext(value):
try:
    from docutils.core import publish_parts
except ImportError:
    if settings.DEBUG:
        raise template.TemplateSyntaxError, \
            "Error in {% restructuredtext %} filter: \
            The Python docutils library isn't installed."
    return force_unicode(value)
else:
    docutils_settings = getattr(settings, "RESTRUCTUREDTEXT_FILTER_SETTINGS", {})
    parts = publish_parts(source=smart_str(value), writer_name="html4css1",\
```

```
            settings_overrides=docutils_settings)
    return mark_safe(force_unicode(parts["fragment"]))
```

Only a single line needs to be changed to hook the custom Writer into the system. In the second line from the bottom the docutils function `publish_parts` is called and gets passed an `writer_name` argument. The only change needed is to replace the `writer_name="html4css1` argument with `writer=MyHTMLWriter()`:

```
parts = publish_parts(source=smart_str(value), writer=MyHTMLWriter(),\
            settings_overrides=docutils_settings)
```

Of course I don't modified the templatefilter in the Django source, instead I made a copy of the code and put it in an templatetag file somewhere in my project.

Now everytime I load the the templatetag in a template and apply the modified restructedtext filter to some variable the custom Writer is used and produces exactly the HTML markup I want.

Veröffentlicht von Arne Brodowski am 11. Sept. 2009, 16:08 in css, django, html, rest, reststructuredtext, semantic, writer.

Kommentare

Thanks for documenting this!

Geschrieben von Seth 6 Stunden, 55 Minuten nach Veröffentlichung des Blog-Eintrags am 11. Sept. 2009, 23:04. Antworten (#comment-form)

Nice! Thank you very much, I'll need this soon.

Geschrieben von Stefan (http://stefanimhoff.de/) 1 Tag, 17 Stunden nach Veröffentlichung des Blog-Eintrags am 13. Sept. 2009, 09:59. Antworten (#comment-form)

Really nice article, can't wait to begin to play with my own one. Thnx a lot.

Geschrieben von Wu (http://blog.e-shell.org/) 3 Tage, 1 Stunde nach Veröffentlichung des Blog-Eintrags am 14. Sept. 2009, 17:49. Antworten (#comment-form)

Quite a great article!
I just need my own rest writer!

Geschrieben von 机票网 (http://www.flight2.net/) 3 Tage, 18 Stunden nach Veröffentlichung des Blog-Eintrags am 15. Sept. 2009, 11:07. Antworten (#comment-form)

Great article, just a small suggestion: Use a monospaced font for your code blocks, they're not aligning up properly.

Geschrieben von Sykora (http://lucentbeing.com/) 1 Monat nach Veröffentlichung des Blog-Eintrags am 16. Okt. 2009, 12:16. Antworten (#comment-form)

font-family is set to Courier on the code blocks, maybe this is not enough for all Browser-OS combination, I will fix it.

Thanks for pointing this out.

Geschrieben von Arne (http://www.arnebrodowski.de/blog/) 1 Monat nach Veröffentlichung des Blog-Eintrags am 16. Okt. 2009, 12:39. Antworten (#comment-form)

This was realy useful. BTW the Django tag returns parts["fragment"] which detaches the first heading from the document. parts["html_body"] seems much more useful.

Geschrieben von Peter 3 Monate nach Veröffentlichung des Blog-Eintrags am 16. Dez. 2009, 18:26. Antworten (#comment-form)

I add a simple rest2html method to python-creole witch create a clean html code from ReStructuredText.

I used some code from here ;) Thanks!

More info: http://code.google.com/p/python-creole/wiki/rest2html

Geschrieben von jedie (http://github.com/jedie/python-creole) 1 Jahr, 10 Monate nach Veröffentlichung des Blog-Eintrags am 4. Aug. 2011, 16:52. Antworten (#comment-form)

Suche