staticjinja Documentation

Release 0.3.0

Ceasar Bautista

August 19, 2016

Contents

staticjinja is a library for easily deploying static sites using the Jinja2 template engine.

Most static site generators are cumbersome to use. Nevertheless, when deploying a static website that could benefit from factored out data or modular HTML pages (especially convenient when prototyping), a templating engine can be invaluable. Jinja2 is an extremely powerful tool in this regard.

staticjinja is designed to be lightweight, easy-to-use, and highly extensible, enabling you to focus on simply making your site.

```
$ mkdir templates
$ vim templates/index.html
$ staticjinja watch
Building index.html...
Templates built.
Watching 'templates' for changes...
Press Ctrl+C to stop.
```

CHAPTER 1

User Guide

This part of the documentation focuses on step-by-step instructions for getting the most of staticjinja.

1.1 Quickstart

Eager to get started? This page gives a good introduction for getting started with staticjinja.

1.1.1 Installation

staticjinja supports Python 2.6, 2.7, 3.3 and 3.4.

Installing staticjinja is simple with pip:

```
$ pip install staticjinja
```

1.1.2 Rendering templates

If you're just looking to render simple data-less templates, you can get up and running with the following shortcut:

```
$ staticjinja build
 Rendering index.html...
```

This will recursively search `./templates` for templates (any file whose name does not start with `.` or _) and build them to `.`.

To monitor your source directory for changes, and recompile files if they change, use `watch`:

```
$ staticjinja watch
 Rendering index.html...
 Watching 'templates' for changes...
 Press Ctrl+C to stop.
```

1.1.3 Configuration

`build` and `watch` each take 3 options:

- `--srcpath` - the directory to look in for templates (defaults to `./templates`);
- `--outpath` - the directory to place rendered files in (defaults to `.`);

- `--static` - the directory (or directories) within `srcpath` where static files (such as CSS and JavaScript) are stored. Static files are copied to the output directory without any template compilation, maintaining any directory structure. This defaults to `None`, meaning no files are considered to be static files. You can pass multiple directories separating them by commas: `--static="foo,bar/baz,lorem"`.

More advanced configuration can be done using the staticjinja API, see *Using Custom Build Scripts* for details.

1.2 Advanced Usage

This document covers some of staticjinja's more advanced features.

1.2.1 Using Custom Build Scripts

The command line shortcut is convenient, but sometimes your project needs something different than the defaults. To change them, you can use a build script.

A minimal build script looks something like this:

```
from staticjinja import make_site


if __name__ == "__main__":
    site = make_site()
    # enable automatic reloading
    site.render(use_reloader=True)
```

To change behavior, pass the appropriate keyword arguments to `make_site`.

- To change which directory to search for templates, set `searchpath="searchpath_name"` (default is `./templates`).
- To change the output directory, pass in `outpath="output_dir"` (default is `.`).
- To add Jinja extensions, pass in `extensions=[extension1, extension2, ...]`.
- To change which files are considered templates, subclass the `Site` object and override `is_template`.
- To change where static files (such as CSS or JavaScript) are stored, set `staticpaths=["mystaticfiles/"]` (the default is `None`, which means no files are considered to be static files). You can pass multiple directories in the list: `staticpaths=["foo/", "bar/"]`. You can also specify singly files to be considered as static: `staticpaths=["favicon.ico"]`.

Finally, just save the script as `build.py` (or something similar) and run it with your Python interpreter.

```
$ python build.py
Building index.html...
Templates built.
Watching 'templates' for changes...
Press Ctrl+C to stop.
```

1.2.2 Loading data

Some applications render templates based on data sources (e.g. CSVs or JSON files).

The simplest way to supply data to templates is to pass `make_site()` a mapping from variable names to their values (a "context") as the `env_globals` keyword argument.

```
if __name__ == "__main__":
    site = staticjinja.make_site(env_globals={
        'greeting':'Hello world!',
    })
    site.render()
```

Anything added to this dictionary will be available in all templates:

```
<!-- templates/index.html -->
<h1>{{greeting}}</h1>
```

If the context needs to be different for each template, you can restrict contexts to certain templates by supplying `make_site()` a sequence of regex-context pairs as the `contexts` keyword argument. When rendering a template, staticjinja will search this sequence for the first regex that matches the template's name, and use that context to interpolate variables. For example, the following code block supplies a context to the template named "index.html":

```
from staticjinja import make_site

if __name__ == "__main__":
    context = {'knights': ['sir arthur', 'sir lancelot', 'sir galahad']}
    site = make_site(contexts=[('index.html', context)])
    site.render()
```

```
<!-- templates/index.html -->
<h1>Knights of the Round Table</h1>
<ul>
{% for knight in knights %}
    <li>{{ knight }}</li>
{% endfor %}
</ul>
```

If contexts needs to be generated dynamically, you can associate filenames with functions that return a context ("context generators"). Context generators may either take no arguments or the current template as its sole argument. For example, the following code creates a context with the last modification time of the template file for any templates with an HTML extension:

```
import datetime
import os

from staticjinja import make_site


def date(template):
    template_mtime = os.path.getmtime(template.filename)
    date = datetime.datetime.fromtimestamp(template_mtime)
    return {'template_date': date.strftime('%d %B %Y')}

if __name__ == "__main__":
    site = make_site(
        contexts=[('.*.html', date)],
    )
    site.render()
```

By default, staticjinja uses the context of the first matching regex if multiple regexes match the name of a template. You can change this so that staticjinja combines the contexts by passing `mergecontexts=True` as an argument to `make_site()`. Note the order is still important if several matching regex define the same key, in which case the last regex wins. For example, given a build script that looks like the following code block, the context of the `index.html` template will be `{'title': 'MySite - Index', 'date': '05 January 2016'}`.

```
import datetime
import os

from staticjinja import make_site


def base(template):
    template_mtime = os.path.getmtime(template.filename)
    date = datetime.datetime.fromtimestamp(template_mtime)
    return {
        'template_date': date.strftime('%d %B %Y'),
        'title': 'MySite',
    }


def index(template):
    return {'title': 'MySite - Index'}

if __name__ == "__main__":
    site = make_site(
        contexts=[('.*.html', base), ('index.html', index)],
        mergecontexts=True,
    )
    site.render()
```

1.2.3 Filters

Filters modify variables. staticjinja uses Jinja2 to process templates, so all the standard Jinja2 filters are supported. To add your own filters, simply pass `filters` as an argument to `make_site()`.

```
filters = {
    'hello_world': lambda x: 'Hello world!',
    'my_lower': lambda x: x.lower(),
}

if __name__ == "__main__":
    site = staticjinja.make_site(filters=filters)
    site.render()
```

Then you can use them in your templates as you would expect:

```
<!-- templates/index.html -->
{% extends "_base.html" %}
{% block body %}
<h1>{{'' | hello_world}}</h1>
<p>{{'THIS IS AN EXAMPLE WEB PAGE.' | my_lower}}</p>
{% endblock %}
```

1.2.4 Compilation rules

Sometimes you'll find yourself needing to change how a template is compiled. For instance, you might want to compile files with a `.md` extension as Markdown, without needing to put jinja syntax in your Markdown files.

To do this, just write a handler by registering a regex for the files you want to handle, and a compilation function (a "rule").

```
import os

from staticjinja import make_site

# Custom MarkdownExtension
from extensions import MarkdownExtension


def get_post_contents(template):
    with open(template.filename) as f:
        return {'post': f.read()}


# compilation rule
def render_post(env, template, **kwargs):
    """Render a template as a post."""
    post_template = env.get_template("_post.html")
    head, tail = os.path.split(post_template.name)
    post_title, _ = tail.split('.')
    if head:
        out = "%s/%s.html" % (head, post_title)
        if not os.path.exists(head):
            os.makedirs(head)
    else:
        out = "%s.html" % (post_title, )
    post_template.stream(**kwargs).dump(out)


if __name__ == "__main__":
    site = make_site(extensions=[
        MarkdownExtension,
    ], contexts=[
        ('.*.md', get_post_contents),
    ], rules=[
        ('.*.md', render_post),
    ])
    site.render(use_reloader=True)
```

Note the rule we defined at the bottom. It tells staticjinja to check if the filename matches the `.*.md` regex, and if it does, to compile the file using `render_post`.

Now just implement `templates/_post.html`...

```
<!-- templates/_post.html -->
{% extends "_base.html" %}
{% block content %}
<div class="post">
{% markdown %}
{{ post }}
{% endmarkdown %}
</div>
{% endblock %}
```

This would allow you to drop Markdown files into your `templates` directory and have them compiled into HTML.

Note: You can grab MarkdownExtension from http://silas.sewell.org/blog/2010/05/10/jinja2-markdown-extension/.

CHAPTER 2

API Documentation

If you are looking for information on a specific function, class, or method, this part of the documentation is for you.

2.1 Developer Interface

This part of the documentation covers staticjinja's API.

2.1.1 Main Interface

All of staticjinja's functionality can be accessed by this function, which returns an instance of the `Site` object.

staticjinja.**make_site**(*searchpath='templates'*, *outpath='.'*, *contexts=None*, *rules=None*, *encoding='utf8'*, *extensions=None*, *staticpaths=None*, *filters=None*, *env_globals=None*, *env_kwargs=None*, *mergecontexts=False*)

Create a `Site` object.

Parameters

- **searchpath** – A string representing the absolute path to the directory that the Site should search to discover templates. Defaults to `'templates'`.

  If a relative path is provided, it will be coerced to an absolute path by prepending the directory name of the calling module. For example, if you invoke staticjinja using `python build.py` in directory `/foo`, then *searchpath* will be `/foo/templates`.

- **outpath** – A string representing the name of the directory that the Site should store rendered files in. Defaults to `'.'`.

- **contexts** – A list of *(regex, context)* pairs. The Site will render templates whose name match *regex* using *context*. *context* must be either a dictionary-like object or a function that takes either no arguments or a single `jinja2.Template` as an argument and returns a dictionary representing the context. Defaults to `[]`.

- **rules** – A list of *(regex, function)* pairs. The Site will delegate rendering to *function* if *regex* matches the name of a template during rendering. *function* must take a `jinja2.Environment` object, a filename, and a context as parameters and render the template. Defaults to `[]`.

- **encoding** – A string representing the encoding that the Site should use when rendering templates. Defaults to `'utf8'`.

- **extensions** – A list of Jinja extensions that the `jinja2.Environment` should use. Defaults to `[]`.
- **staticpaths** – List of directories to get static files from (relative to searchpath). Defaults to `None`.
- **filters** – A dictionary of Jinja2 filters to add to the Environment. Defaults to `{}`.
- **env_globals** – A mapping from variable names that should be available all the time to their values. Defaults to `{}`.
- **env_kwargs** – A dictionary that will be passed as keyword arguments to the jinja2 Environment. Defaults to `{}`.
- **mergecontexts** – A boolean value. If set to `True`, then all matching regex from the contexts list will be merged (in order) to get the final context. Otherwise, only the first matching regex is used. Defaults to `False`.

Classes

class staticjinja.**Site**(*environment*, *searchpath*, *outpath*, *encoding*, *logger*, *contexts=None*, *rules=None*, *staticpaths=None*, *mergecontexts=False*)

The Site object.

Parameters

- **environment** – A `jinja2.Environment`.
- **searchpath** – A string representing the name of the directory to search for templates.
- **contexts** – A list of *regex, context* pairs. Each context is either a dictionary or a function that takes either no argument or or the current template as its sole argument and returns a dictionary. The regex, if matched against a filename, will cause the context to be used.
- **rules** – A list of *regex, function* pairs used to override template compilation. *regex* must be a regex which if matched against a filename will cause *function* to be used instead of the default. *function* must be a function which takes a Jinja2 Environment, the filename, and the context and renders a template.
- **encoding** – The encoding of templates to use.
- **logger** – A logging.Logger object used to log events.
- **staticpaths** – List of directory names to get static files from (relative to searchpath).
- **mergecontexts** – A boolean value. If set to `True`, then all matching regex from the contexts list will be merged (in order) to get the final context. Otherwise, only the first matching regex is used. Defaults to `False`.

get_context(*template*)

Get the context for a template.

If no matching value is found, an empty context is returned. Otherwise, this returns either the matching value if the value is dictionary-like or the dictionary returned by calling it with *template* if the value is a function.

If several matching values are found, the resulting dictionaries will be merged before being returned if mergecontexts is True. Otherwise, only the first matching value is returned.

Parameters **template** – the template to get the context for

get_dependencies(*filename*)

Get a list of files that depends on the file named *filename*.

Parameters **filename** – the name of the file to find dependencies of

get_rule(*template_name*)
Find a matching compilation rule for a function.

Raises a `ValueError` if no matching rule can be found.

Parameters **template_name** – the name of the template

get_template(*template_name*)
Get a `jinja2.Template` from the environment.

Parameters **template_name** – A string representing the name of the template.

is_ignored(*filename*)
Check if a file is an ignored file.

Ignored files are neither rendered nor used in rendering templates.

A file is considered ignored if it or any of its parent directories are prefixed with an `'.'`.

Parameters **filename** – the name of the file to check

is_partial(*filename*)
Check if a file is a partial.

Partial files are not rendered, but they are used in rendering templates.

A file is considered a partial if it or any of its parent directories are prefixed with an `'_'`.

Parameters **filename** – the name of the file to check

is_static(*filename*)
Check if a file is a static file (which should be copied, rather than compiled using Jinja2).

A file is considered static if it lives in any of the directories specified in `staticpaths`.

Parameters **filename** – the name of the file to check

is_template(*filename*)
Check if a file is a template.

A file is a considered a template if it is neither a partial nor ignored.

Parameters **filename** – the name of the file to check

render(*use_reloader=False*)
Generate the site.

Parameters **use_reloader** – if given, reload templates on modification

render_template(*template*, *context=None*, *filepath=None*)
Render a single `jinja2.Template` object.

If a Rule matching the template is found, the rendering task is delegated to the rule.

Parameters

- **template** – A `jinja2.Template` to render.
- **context** – Optional. A dictionary representing the context to render *template* with. If no context is provided, *`get_context()`* is used to provide a context.
- **filepath** – Optional. A file or file-like object to dump the complete template stream into. Defaults to to `os.path.join(self.outpath, template.name)`.

render_templates(*templates*, *filepath=None*)
Render a collection of `jinja2.Template` objects.

Parameters

- **templates** – A collection of Templates to render.
- **filepath** – Optional. A file or file-like object to dump the complete template stream into. Defaults to to `os.path.join(self.outpath, template.name)`.

templates
Generator for templates.

class staticjinja.**Reloader**(*site*)
Watches `site.searchpath` for changes and re-renders any changed Templates.

Parameters **site** – A *Site* object.

event_handler(*event_type*, *src_path*)
Re-render templates if they are modified.

Parameters

- **event_type** – a string, representing the type of event
- **src_path** – the path to the file that triggered the event.

should_handle(*event_type*, *filename*)
Check if an event should be handled.

An event should be handled if a file in the searchpath was modified.

Parameters

- **event_type** – a string, representing the type of event
- **filename** – the path to the file that triggered the event.

watch()
Watch and reload modified templates.

CHAPTER 3

Contributor Guide

If you want to contribute to the project, this part of the documentation is for you.

3.1 Contributing

staticjinja is under active development, and contributions are more than welcome!

3.1.1 Get the Code

staticjinja is actively developed on GitHub, where the code is always available.

You can clone the public repository:

```
git clone git://github.com/Ceasar/staticjinja.git
```

Once you have a copy of the source, you can embed it in your Python package, or install it into your site-packages easily:

```
$ python setup.py install
```

3.1.2 How to Help

1. Check for open issues or open a fresh issue to start a discussion around a feature idea or a bug.
2. Fork the repository on GitHub to start making your changes to the **master** branch (or branch off of it).
3. Send a pull request and bug the maintainer until it gets merged and published. :) Make sure to add yourself to AUTHORS.

3.2 Authors

staticjinja is written and maintained by Ceasar Bautista and various contributors:

3.2.1 Development Lead

- Ceasar Bautista <cbautista2010@gmail.com>

3.2.2 Patches and Suggestions

- Dominic Rodger (dominicrodger)
- Filippo Valsorda
- Alexey (rudyrk)
- Jacob Lyles
- Jakub Zalewski
- NeuronQ
- Eduardo Rivas (jerivas)
- Bryan Bennett (bbenne10)
- Anuraag Agrawal (anuraaga)
- saschalalala
- Tim Best (timbest)

3.3 Changelog

3.3.1 0.3.3

- Enable users to direct pass dictionaries instead of context generator in Site and make_site() for contexts that don't require any logic.
- Introduces a `mergecontexts` parameter to Site and make_site() to direct staticjinja to either use all matching context generator or only the first one when rendering templates.

3.3.2 0.3.2

- Allow passing keyword arguments to jinja2 Environment.
- Use `shutil.copy2` instead of `shutil.copyfile` when copying static resources to preserve the modified time of files which haven't been modified.
- Make the Reloader handle "created" events to support editors like Pycharm which save by first deleting then creating, rather than modifying.
- Update easywatch dependency to 0.0.3 to fix an issue that occurs when installing easywatch 0.0.2.
- Make `--srcpath` accept both absolute paths and relative paths.
- Allow directories to be marked partial or ignored, so that all files inside them can be considered partial or ignored. Without this, developers would need to rename the contents of these directories manually.
- Allow users to mark a single file as static, instead of just directories.

3.3.3 0.3.1

- Add support for filters so that users can define their own Jinja2 filters and use them in templates:

```
    filters = {
        'filter1': lambda x: "hello world!",
        'filter2': lambda x: x.lower()
    }
    site = staticjinja.make_site(filters=filters)
```

- Add support for multiple static directories. They can be passed as a string of comma-separated names to the CLI or as a list to the Renderer.
- "Renderer" was renamed to "Site" and the Reloader was moved staticjinja.reloader.

3.3.4 0.3.0

- Add a command, `staticjinja`, to handle the simple case of building context-less templates.
- Add support for copying static files from the template directory to the output directory.
- Add support for testing, linting and checking the documentation using `tox`.

3.3.5 0.2.0

- Add a `Reloader` class.
- Add `Renderer.templates`, which refers to the lists of templates available to the `Renderer`.
- Make `Renderer.get_context_generator()` private.
- Add `Renderer.get_dependencies(filename)`, which gets every file that depends on the given file.
- Make `Renderer.render_templates()` require a list of templates to render, *templates*.

S

E

G

I

M

R

S

T

W