staticky

Release 0.2

September 08, 2015

Contents

1 Overview **1**
- 1.1 Quick Example . . . 1

2 Staticky Environment **3**
- 2.1 Manifest . . . 3

3 Rules **7**
- 3.1 Cat . . . 7
- 3.2 Less . . . 7
- 3.3 Mako . . . 7
- 3.4 Sass . . . 8
- 3.5 Creating New Rules . . . 8

4 Changes **9**
- 4.1 0.2 (TBD) . . . 9
- 4.2 0.1.3 (Jun 13 2014) . . . 9
- 4.3 0.1.2 (Jun 5 2014) . . . 9
- 4.4 0.1.1 (Jun 4 2014) . . . 9
- 4.5 0.1 (Jun 4 2014) . . . 9

CHAPTER 1

Overview

staticky is a library that handles compiling collections of "static" files, such as for webapps.

It takes a set of directories, and a set of rules, and uses them to generate a set of output files, which can be exposed through several interfaces such as a wsgi application, or be installed to a target destination.

It has a few design goals:

- Be able to pull files from several different locations, to allow imported static files to be laid out modularly.
- Allow for the creation of custom third party rules.

Future planned improvements also include:

- Integration into web frameworks such as Django or Flask.
- Suitability for generating entirely static sites.
- Improved multiprocess behavior (probably using file locking)

Warning: staticky is alpha software, and its API is subject to change from release to release!

1.1 Quick Example

Registration:

```
from staticky import StaticFiles

staticfiles = StaticFiles(input=["path/to/static"])

staticfiles.register('staticky.rules.sass:Sass', input='{name}.scss', output='{name}.css')
staticfiles.register('staticky.rules.sass:Sass', input='{name}.sass', output='{name}.css')

staticfiles.register_post('staticky.post.uglify:UglifyJS', name='{name}.js')
staticfiles.register_post('staticky.post.uglify:UglifyCSS', name='{name}.css')
```

As a View:

```
from webob import Request

request = webob.Request.blank("/file.css")
response = staticfiles.as_view(request)
```

As a WSGI App:

```
application = staticfiles.as_wsgi
```

Pre-Compile (for deployment):

```
staticfiles.install("/var/www/htdocs")
```

CHAPTER 2

Staticky Environment

2.1 Manifest

The manifest describes what files are copied by the *install()* method, or is visible via the *as_view()* and *as_wsgi()* methods.

2.1.1 Manifest Rules

A manifest is an ordered collection of rules. The last matching rule determines if the file is to be included, and each rule either includes matching files, or excludes matching files.

The determination of whether a rule includes or excludes paths is whether the type character is '+' or '-'. The determination of whether a rule matches a path is given by a shell-like glob, in the following format:

Pattern	Description
`*`	Matches everything.
`**/`	Subdirectory matching
`/**`	Parent directory matching
`?`	Matches a single character.
`[chars]`	Matches any character in *chars*
`[!chars]`	Matches any character not in *chars*

If a pattern does not contain a slash, then the pattern will only be checked against the file's name relative to its directory. If is not, then a full path will need to be specified, or use of the `**` directory matching will need to be used.

When a manifest is read, it goes through several transformations, and each of the resulting formats may be used for the manifest.

The first, is as a filename, in which case the file will be opened and used as a sequence of strings.

The second, a sequence of strings, is converted to a sequence of tuples. Each string that is blank or starts with # is ignored, and all other strings are expected to be formatted as "type pattern", where type is the type character, and pattern is the shell-like glob pattern. For example, `"+ *.txt"`.

The final format, before being converted into its internal format, is a list of tuples in the form of (typecharacter, pattern). For example, `('+', '*.txt')`.

class `staticky.`**`StaticFiles`**(*input*, *workdir=None*, *pathvar=None*, *manifest=None*, *index='index.html'*, *meta_class='staticky.meta.sqlite:MetaData'*)

Create a StaticFiles object, which defines a staticky environment.

The location of source files indicated by `input`, and may either be a path, or a list of a paths. If a list is given, priority increases with later paths.

The staging area is specified by `workdir`, and if it is omitted or `None`, a temporary directory will be used that will be cleaned up on destruction of the object.

If `manifest` is provided, it controls what files are visible via *`open()`*, *`as_wsgi()`*, *`as_view()`* or *`install()`*, although *`install()`* can override the manifest specified here. For information on the expected format, see the *Manifest* documentation.

The `index` keyword, which defaults to "index.html" may be used to indicate which file should be used if a directory (with a trailing slash) is requested when using *`as_wsgi()`* or *`as_view()`*.

as_view(*request*, *path=None*)
Call as a view, expecting a `WebOb Request` object `request` and returning a `WebOb Response` object.

as_wsgi(*environ*, *start_response*)
Call as a wsgi application.

The bound method may be passed to anything expecting a wsgi application, eg `make_server('', 8080, staticfiles.as_wsgi)`.

cleanup()
Clean up any temporary files and closes any opened resources.

This method is automatically called on object destruction.

install(*destdir*, *gzip=False*, *clean=False*, *include_post=True*, *manifest=None*)
Copy the built tree of files to `destdir`. If a `manifest` provided here, or when the `StaticFiles` object was constructed, it will be used to control what files are actually copied over.

If the `gzip` flag is provided, then each files generated will also be gzip compressed with .gz appended to each filename.

If the `clean` flag is provided, then the destination directory will be wiped before installation occurs.

Any post-processing rules will be called, unless `include_post` is provided with a false value.

open(*path*, *mode='r'*, *force=False*, *include_post=False*, *job=None*)
Opens a built file for reading or writing.

If the contents of a file are overwritten, changes are likely to be lost the next time the file is built, such as if modifications are detected in a source file.

register(*rule*, *args={}*, *input=None*, *output=None*)
Register a rule, where `rule` is either a callable or a string in form of `MODULE:CLASS`.

If `rule` is a string representing a class, then `args` will be passed to it when instantiated as keyword arguments, allowing for rule-specific configuration. It is expected that this class is itself callable, and will be called to build files the rule is register for.

If `rule` is already a callable, it will be used directly as the callable for building files. For more information about existing rules and the construction of new rules see the Rules documentation.

`input` and `output` are strings that describe what filenames the rule matches. The wildcard parts of the url are indicated using curly brackets (e.g. `"{filename}.ext"`) and the matchable text can be changed by appending a colon and a regular expression (e.g. `"{filename:[a-z]}.ext"`). Wildcard names must be included in both the `input` and the `output` strings, as this determines how names are mapped. For example, if `input` is `"{filename}.in"` and `output` is `"{filename}.out"`, a file named hello.out would be generated if a filename called hello.in is found.

register_post(*rule*, *args={}*, *name='{f}'*, *priority=0*)
Register a post-processing rule, where `rule` follows the same form as used with *`register()`*.

Unlike the *register()* method, the input and output are implicitly the same, and are specified via `name`, and the built file is a transformation of an existing file.

priority determines what order post-processing rules are evaluated in, so that multiple post-processing rules can be chained on files in a consistent order.

CHAPTER 3

Rules

3.1 Cat

Usage:

```
staticfiles.register('staticfiles.rules.cat:Cat', input='{name}.{ext}.cat', output='{name}.{ext}')
```

This rule concatenates several additional files together from the contents of a singular input file, where each file to be concatenated is each provided on their own line in order. Any line that is blank or starts with # is ignored.

This rule takes no parameters.

3.2 Less

Usage:

```
staticfiles.register('staticfiles.rules.less:Less', input='{name}.less', output='{name}.css', **para
```

This runs lessc to produce CSS files from LESS files.

Parameters:

- include
- lessc

3.3 Mako

Usage:

```
staticfiles.register('staticfiles.rules.mako:Mako', input='{name}.mako', output='{name}.html', **para
```

This calls mako to produce HTML (or other files) from Mako templates.

Parameters:

- include
- template_factory

3.4 Sass

Usage:

```
staticfiles.register('staticfiles.rules.sass:Sass', input='{name}.{ext:sass|scss}', output='{name}.cs
```

This rule runs sass to produce CSS files from sass or scss files.

Parameters:

- cache_dir
- include
- sass

3.5 Creating New Rules

Rules are simple callable classes. Their constructors take any parameters as named keywords. They are invoked as a callable to actually perform the rule operation using the following parameters: * context * input file * output file

Context contains several useful methods, including:

open(*path*, *mode*)
: Open a file. If it does not exist, it will be generated if any rule is capable of generating it, and if it is not currently being generated.

  This is a shortcut for calling the builtin open() with resolve()'s return value.

resolve(*name*)
: Return the real path on the filesystem for a path. If it does not exist, it will be generated if any rule is capable of generating it, and if it is not currently being generated.

CHAPTER 4

Changes

4.1 0.2 (TBD)

- Rewrite building process, so that outputs can be used as inputs for other rules.
- Add manifest specification to describe what files to include as installed/served output
- Introduce rule for Jinja templates.
- Introduce rule for yuglify compressor.
- Split existing compressors into own rules.
- Introduce some basic tests and documentation.

4.2 0.1.3 (Jun 13 2014)

- Make paths passed to rules more consistent.
- Packaging updates.

4.3 0.1.2 (Jun 5 2014)

- Sanitize urls.
- Introduce simple LESS rule.

4.4 0.1.1 (Jun 4 2014)

- Packaging fixes.

4.5 0.1 (Jun 4 2014)

- First initial release.

A

as_view() (staticky.StaticFiles method), 4
as_wsgi() (staticky.StaticFiles method), 4

C

cleanup() (staticky.StaticFiles method), 4

I

install() (staticky.StaticFiles method), 4

O

open() (built-in function), 8
open() (staticky.StaticFiles method), 4

R

register() (staticky.StaticFiles method), 4
register_post() (staticky.StaticFiles method), 4
resolve() (built-in function), 8

S

StaticFiles (class in staticky), 3