django-staticpreprocessor Documentation

Release 0.3.0

Luke Pomfrey

August 13, 2015

Contents

django-staticpreprocessor is a Django app to simplify the pre-processing of static assets.

staticpreprocessor can be used for building assets such as less/sass/scss files, and handlebars and other JS templates.

Static files needing pre-processing are collected, in a similar manner to Django's staticfiles collection process, into a pre-selected directory. They are then operated on by processors to generate the required files which will then be collected by collectstatic.

CHAPTER 1

Contents

1.1 Installation

You can grab django-staticpreprocessor from PyPI:

```
$ pip install django-staticpreprocessor
```

Add `staticpreprocessor` to your `INSTALLED_APPS`.

Create a directory to hold your pre-compiled static assets, set the `STATIC_PREPROCESSOR_ROOT` setting, add add it to `STATICFILES_DIRS`:

```
STATIC_PREPROCESSOR_ROOT = '/path/to/rawstatic/'
STATICFILES_DIRS = (
    ...
    STATIC_PREPROCESSOR_ROOT,
    ...
)
```

1.2 Usage

Add `staticpreprocessor` to your `INSTALLED_APPS`.

Create a directory to hold your pre-compiled static assets, set the *`STATIC_PREPROCESSOR_ROOT`* setting, add add it to `STATICFILES_DIRS`:

```
STATIC_PREPROCESSOR_ROOT = '/path/to/processedstatic/'
STATICFILES_DIRS = (
    ...
    STATIC_PREPROCESSOR_ROOT,
    ...
)
```

1.2.1 Quickstart

- Add your required finders to the *`STATIC_PREPROCESSOR_FINDERS`* list.
- Add your required processors to the *`STATIC_PREPROCESSOR_PREPROCESSORS`* list.
- Run the `preprocess_static` management command.

1.2.2 Finders

Finders are exactly the same in concept as staticfiles finders. `staticpreprocessor` comes with several.

class `staticpreprocessor.finders.`**`FileSystemFinder`**
Analagous to the similarly-named `staticfiles` finder, the `FileSystemFinder` collects all files from the directories named in the `STATIC_PREPROCESSOR_DIRS` setting.

class `staticpreprocessor.finders.`**`AppDirectoriesFinder`**
Again, this is analagous to the `AppDirectoriesFinder` in staticfiles, with the exception that rather than collecting files from the `/static/` directory under each app, files are collected from `/rawstatic/`.

In order to use the finders they should be added to the *`STATIC_PREPROCESSOR_FINDERS`* setting, e.g.:

```
STATIC_PREPROCESSOR_DIRS = (
    os.path.join(os.path.dirname(__file__), 'rawstatic/'),
)
STATIC_PREPROCESSOR_FINDERS = (
    'staticpreprocessor.finders.FileSystemFinder',
    'staticpreprocessor.finders.AppDirectoriesFinder',
)
```

1.2.3 Processors

Processors are the classes that do the actual work of pre-processing your static files.

Processors can be specified in the `STATIC_PREPROCESSORS_PROCESSORS` setting as either dotted-paths, or otherwise, if a tuple or list is given it will be taken as the dotted path to the processor and a dictionary of keyword arguments, e.g.:

```
from staticpreprocessor.contrib.processors.less import LessProcessor
from staticpreprocessor.contrib.processors.sass import SassProcessor
from staticpreprocessor.processors import CommandListProcessor

STATIC_PREPROCESSOR_PROCESSORS = (
    'staticpreprocessor.contrib.processors.HandlebarsProcessor',
    LessProcessor,
    SassProcessor(),
    ('staticpreprocessor.processors.CommandListProcessor',
     dict(extensions=['.txt'], command='echo {input} > {output}')),
    CommandListProcessor(
        extensions=['.txt'], command='echo {input} > {output}'),
)
```

There are several base processor classes in `staticpreprocessor.processors` that can be extended and used:

class `staticpreprocessor.processors.`**`BaseProcessor`**
This is the base processor implementation that defines the most basic functionality of a processor, namely, the following methods:

`get_file_list`(*self*, ***kwargs*)
Returns the list of files to be operated on by the processor.

`handle`(*self*, ***kwargs*)
this is the main method that processes the static files.

And the following attributes:

storage
The storage class to use. Defaults to the default staticpreprocessor storage.

extensions
The file extensions to target, e.g. `.txt`, `.css` as a `list` or `tuple`. Setting to `None` will cause the processor to operate on all file extensions

exclude_match
A glob-type expression. Any files matching this pattern will be excluded from processing by this processor.

exclude_regex
An un-compiled regex string. Any files matching this pattern will be excluded from processing by this processor.

include_match
A glob-type expression. Any files *NOT* matching this pattern will be excluded from processing by this processor.

include_regex
An un-compiled regex string. Any files *NOT* matching this pattern will be excluded from processing by this processor.

class staticpreprocessor.processors.**BaseListProcessor**
`BaseListProcessor` extends *BaseProcessor* and allows the entire collected file list to be processed using the `handle_list` method.

Methods:

handle_list(*self*, *file_list*, ** *kwargs*)
`file_list` is the list of all files found to be handled in bulk.

Attributes:

remove_processed_files
If this is `True` (the default), the processor will remove the processed files after processing.

class staticpreprocessor.processors.**BaseFileProcessor**
`BaseFileProcessor` extends *BaseListProcessor*, with the `handle_file` method being called once for every file in the collected file list.

Methods:

handle_file(*self*, *file*, ***kwargs*)
Is repeatedly called, with `file` being a single file from the collected file list.

Attributes:

remove_processed_files
If this is `True` (the default), the processor will remove the processed files after processing.

class staticpreprocessor.processors.**CommandProcessorMixin**
The `CommandProcessorMixin` provides command running functionality via the envoy package.

Methods:

get_command(*self*, ***kwargs*)
Returns the command to be run. By default this is the *command* attribute formatted with **kwargs. **kwargs contains any keyword arguments passed to the class, along with *input* which is generally the space-separated list of files to be operated on, and *output* which is the *output* attribute passed through the class' storage *path* method.

run_command(*self*, *input*, ***kwargs*)
Runs the command returned by *get_command()*.

input should generally be a space separated list of files to be processed. If *require_input* is *True*, the default, and input is empty the command will not be run.

If the return value of the command run is not in the list *expected_return_codes* then this method will raise *RuntimeError*.

Attributes:

command
The command line string to be run. By default this will be formatted by the *get_command()* method so string formatting sequences can be used, e.g.: `cat {input} > {output}`.

output
A path to an output file. This will be passed through `storage.path` so it may be relative to *STATIC_PREPROCESSOR_ROOT*.

expected_return_codes
A list of return codes that are acceptable for the run process to return. Defaults to `[0]`.

require_input
Whether or not we should require input in order to run the command. Defaults to `True`.

class `staticpreprocessor.processors.`**CommandListProcessor**
Extends *BaseListProcessor* and *CommandProcessorMixin*. The specified command is run with *input* being the space-separated list of filenames generated by `get_file_list()`.

class `staticpreprocessor.processors.`**CommandFileProcessor**
Extends *BaseListProcessor* and *CommandProcessorMixin*. The specified command is run on each filename generated by `get_file_list()` in turn, with *input* being the filename.

All attributes on processor classes are overridden by any keyword arguments passed to `__init__`.

Contrib Processors

There are several processors included in the `staticpreprocessor.contrib.processors` module.

class `handlebars.`**HandlebarsProcessor**
Processes all `.handlebars` files into `handlebars_templates.js`.

class `sass.`**SassProcessor**
Processes all `.sass` and `.scss` files into `sass_styles.css`.

class `less.`**LessProcessor**
Processes all `.less` files into `less_styles.css`.

1.2.4 `preprocess_static` Management Command

Once you've added your finders and processors to your settings file, you can run the `preprocess_static` management command.

This will find all of your raw static files, collect them into *STATIC_PREPROCESSOR_ROOT* and run your processors on them.

By default, `preprocess_static` will empty the target directory before processing, to prevent this from happending pass the `--no-clear` argument to the command.

1.2.5 Settings

staticpreprocessor.conf.**STATIC_PREPROCESSOR_ROOT**
The directory to collect the pre-processed static files in. This must be defined.

staticpreprocessor.conf.**STATIC_PREPROCESSOR_STORAGE**
Default: 'staticpreprocessor.storage.StaticPreprocessorFileStorage'

The path to the storage class used to store pre-processed files. You shouldn't need to change this unless you want to use some form of cloud storage etc.

staticpreprocessor.conf.**STATIC_PREPROCESSOR_FINDERS**
Default: []

The list of finders to use to collect files to be pre-processed. these are run in order, with files collected by one finder being overwritten by files with the same name found by other finders. Should contain dotted-paths to finders.

Example:

```
STATIC_PREPROCESSOR_FINDERS = [
    'staticpreprocessor.finders.FileSystemFinder',
]
```

staticpreprocessor.conf.**STATIC_PREPROCESSOR_PROCESSORS**
Default: []

The list of processors to run against the collected files. These may be specified as dotted-paths or classes/class instances.

Example:

```
from staticpreprocessor.contrib.processors.less import LessProcessor
from staticpreprocessor.contrib.processors.sass import SassProcessor
from staticpreprocessor.processors import CommandListProcessor

STATIC_PREPROCESSOR_PROCESSORS = (
    'staticpreprocessor.contrib.processors.HandlebarsProcessor',
    LessProcessor,
    SassProcessor(),
    CommandListProcessor(
        extensions=['.txt'], command='echo {input} > {output}'),
)
```

staticpreprocessor.conf.**STATIC_PREPROCESSOR_DIRS**
Default: []

The list of directories that the *FileSystemFinder* will look for files in.

1.3 Examples

In all the following examples the settings shown should be used in order to have the *preprocess_static* command produce the desired result.

1.3.1 Compiling all sass stylesheets and handlebar templates

To compile all less files into `styles.css` and all handlebars templates into `handlebars_templates.js`:

```
# settings.py
import os

STATIC_PREPROCESSOR_ROOT = os.path.join(
    os.path.dirname(__file__), 'processedstatic/')
STATIC_PREPROCESSOR_FINDERS = [
    'staticpreprocessors.finders.AppDirectoriesFinder',
    'staticpreprocessors.finders.FileSystemFinder',
]
STATIC_PREPROCESSOR_PROCESSORS = [
    'staticpreprocessor.contrib.processors.less.LessProcessor',
    'staticpreprocessor.contrib.processors.handlebars.HandlebarsProcessor',
]
```

1.3.2 Compiling Less files to multiple targets

To compile `less/responsive.less` to `css/responsive.css` and `less/unresponsive.less` to `css/unresponsive.css`:

```
# settings.py
import os
from staticpreprocessor.contrib.processors.less import LessProcessor

STATIC_PREPROCESSOR_ROOT = os.path.join(
    os.path.dirname(__file__), 'processedstatic/')
STATIC_PREPROCESSOR_FINDERS = [
    'staticpreprocessors.finders.AppDirectoriesFinder',
    'staticpreprocessors.finders.FileSystemFinder',
]
STATIC_PREPROCESSOR_PROCESSORS = [
    LessProcessor(
        include_match='less/unresponsive.less',
        output='css/unresponsive.less'
    ),
    LessProcessor(
        include_match='less/responsive.less',
        output='css/responsive.css',
    )
]
```

1.3.3 Compiling multiple handlebar template groups

To compile all templates in `groupa` directories into `handlebar_groupa.js` and all templates in `groupb` into `handlebar_groupb.js`:

```
# settings.py
import os
from staticpreprocessor.contrib.processors.handlebars import HandlebarsProcessor

STATIC_PREPROCESSOR_ROOT = os.path.join(
    os.path.dirname(__file__), 'processedstatic/')
STATIC_PREPROCESSOR_FINDERS = [
    'staticpreprocessors.finders.AppDirectoriesFinder',
    'staticpreprocessors.finders.FileSystemFinder',
]
```

```
STATIC_PREPROCESSOR_PROCESSORS = [
    HandlebarsProcessor(
        include_regex=r'^groupa/.*',
        output='handlebar_groupa.js',
    ),
    HandlebarsProcessor(
        include_match='groupb/*',
        output='handlebar_groupb.js',
    )
]
```

S

S