django-staticpreprocessor Documentation

Release 0.1.0

Luke Pomfrey

June 07, 2013

CONTENTS

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

It was written at Titan Entertainment Group to enable us to pre-compile sass/less files, and handlebars templates before deployment to remove the need to install node/ruby apps on the server.

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

ONE

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 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',
)
```

1.2.2 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, 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(),
    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`.

PYTHON MODULE INDEX