jinja2-webpack

Release 0.1.4

Sep 26, 2017

Contents

CHAPTER 1

Overview

docs	
tests	
package	

Integration of webpack with jinja2

- Free software: BSD license

Description

Plugin for jinja2 which enables you to reference assets generated by webpack in your Jinja2 templates, as well as an optional scan feature which allows you to push references to assets from your Jinja2 templates back into webpack for it to build them.

Installation

```
pip install jinja2-webpack
```

Documentation

https://jinja2-webpack.readthedocs.io/

Development

To run the all tests run:

```
tox
```

Note, to combine the coverage data from all the tox environments run:

Windows	`set PYTEST_ADDOPTS=--cov-append` `tox`
Other	`PYTEST_ADDOPTS=--cov-append tox`

CHAPTER 2

Installation

At the command line:

```
pip install jinja2-webpack
```

CHAPTER 3

Usage

You will configure webpack to write a JSON manifest of built files with webpack-manifest-plugin.

This project can then be configured to load the manifest, and allow you to reference the templates from your jinja2 templates by using the filter syntax:

```
{{ "entry_name" | webpack }}
```

So for a JS file reference you might use syntax like:

```
<script src="{{ "test_js_bundle" | webpack }}"></script>
```

CHAPTER 4

Setting up webpack

Add webpack-manifest-plugin as a dependency and configure it into your webpack.config.js:

```
var ManifestPlugin = require('webpack-manifest-plugin');

module.exports = {
    ...
    plugins: [
        ...,
        new ManifestPlugin({
            fileName: './webpack-manifest.json',
        })
    ]
}
```

CHAPTER 5

Auto-scanning jinja2 templates for entries

It is also possible to scan your jinja2 templates to force webpack to process assets that you reference from them.

This would enable you to reference static pngs from within your jinja2 templates.

The project comes with a command line tool called jinja2-webpack-scan which can be used to generate a JS file that requires() all of the assets you reference from your jinja2 templates. This will cause webpack to process them and add them to the manifest.

Example running it on a template with:

```
{{ "./image.png" | webpack }}
```

Would produce a JS reference file with a line like:

```
require("./image.png");
```

And it takes the path of the entry into account, so you can do relative imports within your templates such as:

```
{{ "../pngs/image.png" | webpack }}
```

And it will resolve the path relative to where you store the output file:

```
require("project/pngs/image.png")
```

Example usage:

```
jinja2-webpack-scan \
    -o webpack-asset-entries.js \
    -d 'templates/' \
    --root 'project/' \
    'templates/*.jinja2'
```

And then configure webpack so it will build those things:

```
modules.exports = {
    ...,
    entries: {
        ...,
        assets: './project/webpack-asset-entries.js'
    }
}
```

CHAPTER 6

Raw integration with jinja2

To use jinja2-webpack in a project:

```
from jinja2 import Environment
from jinja2_webpack import Environment as WebpackEnvironment
from jinja2_webpack.filter import WebpackFilter


jinja2_env = Environment(loader=FileSystemLoader(['.']))
webpack_env = WebpackEnvironment(manifest='webpack-manifest.json')
jinja2_env.filters['webpack'] = WebpackFilter(webpack_env)
```

CHAPTER 7

Pyramid

See: pyramid-jinja2-webpack.

CHAPTER 8

Reference

jinja2_webpack

class jinja2_webpack.**Asset**(*filename*, *url*)
Asset class. Might someday expose file access here too so can render assets inline. For now the url is the interesting attribute

exception jinja2_webpack.**AssetNotFoundException**
Thrown when an asset cannot be found, can be disabled by settings errorOnInvalidReference = False

class jinja2_webpack.**Environment**(***kwargs*)
The webpack environment class. Loads the manifest and allows it to be accessed. Settings:

- **manifest - default "webpack-manifest.json"** Path to the WebpackManifest file
- **errorOnInvalidReference - default True** True if exception should be thrown when you try to resolve an invalid asset reference
- **publicRoot - default /static/pack** The public path to prepend to all asset URLs

identify_assetspec(*spec*)
Lookup an asset from the webpack manifest. The asset spec is processed such that you might reference an entry with or without the extension.

Will raise an AssetNotFoundException if the errorOnInvalidReference setting is enabled and the asset cannot be found.

Note that all files must be have globally unique names, due to a limitation in the way that WebpackManifestPlugin writes the data.

register_renderer(*extension*, *renderer*)
Register a new renderer function to the environment

render_asset(*asset*)
Render an asset to a URL or something more interesting, by looking up the extension in the registered renderers

CHAPTER 9

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

Bug reports

When reporting a bug please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Documentation improvements

jinja2-webpack could always use more documentation, whether as part of the official jinja2-webpack docs, in docstrings, or even on the web in blog posts, articles, and such.

Feature requests and feedback

The best way to send feedback is to file an issue at https://github.com/JDeuce/python-jinja2-webpack/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that code contributions are welcome :)

Development

To set up *python-jinja2-webpack* for local development:

1. Fork python-jinja2-webpack (look for the "Fork" button).
2. Clone your fork locally:

```
git clone git@github.com:your_name_here/python-jinja2-webpack.git
```

3. Create a branch for local development:

```
git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

4. When you're done making changes, run all the checks, doc builder and spell checker with tox one command:

```
tox
```

5. Commit your changes and push your branch to GitHub:

```
git add .
git commit -m "Your detailed description of your changes."
git push origin name-of-your-bugfix-or-feature
```

6. Submit a pull request through the GitHub website.

Pull Request Guidelines

If you need some code review or feedback while you're developing the code just make the pull request.

For merging, you should:

1. Include passing tests (run `tox`)[1].
2. Update documentation when there's new API, functionality etc.
3. Add a note to `CHANGELOG.rst` about the changes.
4. Add yourself to `AUTHORS.rst`.

Tips

To run a subset of tests:

```
tox -e envname -- py.test -k test_myfeature
```

[1] If you don't have all the necessary python versions available locally you can rely on Travis - it will run the tests for each change you add in the pull request.
It will be slower though ...

CHAPTER 10

Authors

- Josh Jaques - https://jdeuce.net

CHAPTER 11

Changelog

0.1.4 (2017-06-03)

- Separate jinja2_webpack.pyramid into pyramid_jinja2_webpack

0.1.0 (2017-05-28)

- First release on PyPI.

CHAPTER 12

Indices and tables

- genindex
- modindex
- search

Python Module Index

j

Index