pyramid$_webpack$

Release 0.1.0

October 23, 2016

A Pyramid extension for managing assets with Webpack.

Code lives here: https://github.com/stevearc/pyramid_webpack

CHAPTER 1

User Guide

1.1 Getting Started

These instructions will get you up and running with a minimal Pyramid app and a basic webpack configuration.

1.1.1 Set up the Pyramid app

Skip this section if you already have a Pyramid server up and running. Set up a virtualenv and create a new Pyramid project:

```
virtualenv venv
source venv/bin/activate
pip install pyramid
pcreate -s starter hello_world
cd hello_world
pip install -e .
```

You should be able to run the server with `pserve development.ini` and see it working.

1.1.2 Install and configure pyramid_webpack

We're also going to install pyramid_jinja2 as the templating engine.

```
pip install pyramid_jinja2 pyramid_webpack
```

Add the following to your development.ini file:

```
# If this option already exists, append the values
pyramid.includes =
    pyramid_jinja2
    pyramid_webpack
jinja2.extensions =
    pyramid_webpack.jinja2ext:WebpackExtension

# Reloads file changes and requests wait while webpack is compiling
webpack.debug = True
# Directory containing the webpack bundles. Relative to your package root.
webpack.bundle_dir = webpack/bundles
# File containing the webpack stats. Relative to your package root.
webpack.stats_file = webpack/stats.json
```

1.1.3 Set up webpack

You will need to have Node installed and in your PATH for the following steps.

```
npm init
npm install --save-dev webpack webpack-bundle-tracker babel babel-loader
```

Put the following into `webpack.config.js`

```
var path = require("path")
var BundleTracker = require('webpack-bundle-tracker')

module.exports = {
  context: __dirname,

  entry: './assets/js/index',

  output: {
      path: path.resolve('./hello_world/webpack/bundles/'),
      filename: "[name]-[hash].js",
  },

  plugins: [
    new BundleTracker({filename: './hello_world/webpack/stats.json'}),
  ],

  module: {
    loaders: [
      {
        test: /\.js$/,
        exclude: /node_modules/,
        loader: 'babel-loader'
      },
    ],
  },

  resolve: {
    modulesDirectories: ['node_modules'],
    extensions: ['', '.js']
  },
}
```

Create a javascript file to be built by webpack:

```
mkdir -p assets/js/
echo "var n = document.createElement('h1'); n.innerText = 'Javascript loaded'; document.body.appendC
```

1.1.4 Running everything

Run the Pyramid server with:

```
pserve --reload development.ini
```

Run webpack with:

```
./node_modules/.bin/webpack --config webpack.config.js -d --progress --colors --watch
```

1.1.5 Using in templates

To render a bundle inside a Chameleon template, we're going to call `get_bundle` directly. Create a file called `hello_world/templates/index.pt` and add the following:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Example</title>
  </head>

  <body>
    <script type="text/javascript" tal:repeat="asset request.webpack().get_bundle('main')" src="${ass
  </body>
</html>
```

Then change the renderer in `hello_world/views.py` to be `templates/index.pt`. When you reload the webpage it should now say "Javascript Loaded".

To render a bundle in Jinja2, make a template called `hello_world/templates/index.jinja2` and add the following:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Example</title>
  </head>

  <body>
    {% webpack 'main' %}
      <script type="text/javascript" src="{{ ASSET.url }}"></script>
    {% endwebpack %}
  </body>
</html>
```

Then change the renderer in `hello_world/views.py` to be `templates/index.jinja2`. When you reload the webpage it should now say "Javascript Loaded".

1.2 Configuration

1.2.1 Options

webpack.debug

Argument: bool, inherits, default `False`

If True the server will re-read the stats file for each request and requests will block while webpack is running. See *webpack.timeout* to configure how long the requests will wait for webpack.

webpack.static_view

Argument: bool, default `True`

If True, pyramid_webpack will automatically set up the static view(s) for you. Disable this if you want to call `add_static_view` yourself (e.g. if you need to pass in custom parameters).

webpack.bundle_dir

Argument: str

The directory that contains the compiled webpack bundles. This may be in three forms:

- `raw_relative_path` - This path will be relative to your root project package
- `package:relative_path` - This is a path relative to a package location
- `/absolute/path` - An absolute path on disk

You will almost always need to supply a *bundle_dir*, but if you are using an external webserver for `webpack.static_view_name`, then you don't need to provide it.

webpack.static_view_name

Argument: str, default `webpack-DEFAULT`

This will be the `name` argument passed to add_static_view.

webpack.stats_file

Argument: str, default `webpack-stats.json`

The location of the webpack stats file generated by the webpack-bundle-tracker plugin. This path may be in the same three formats as `webpack.bundle_dir`.

webpack.timeout

Argument: float, inherits, default `0`

Requests will block for this many seconds while waiting for webpack to finish compiling (if `webpack.debug = True`). A value of `0` will wait indefinitely.

webpack.ignore

Argument: list, inherits, default `*.hot-update.js, *.map`

When getting a bundle, ignore chunks that match these patterns. Uses glob matching.

webpack.ignore_re

Argument: list, inherits

When getting a bundle, ignore chunks that match these patterns. Uses PCRE matching.

webpack.configs

Argument: list

List of names of other webpack configurations to load. See the section below for more detail.

1.2.2 Multiple Configs

If you have multiple instances of webpack running and generating bundles, you can load them as well by giving them names and adding them to the `webpack.configs` option. You can then configure those instances by prefixing the same options in the config file with `webpack.<name>.`. For example:

```
webpack.debug = True
webpack.bundle_dir = webpack/bundles
webpack.stats_file = webpack/stats.json
webpack.configs =
    other

webpack.other.bundle_dir = webpack/other/bundles
webpack.other.stats_fie = webpack/other/stats.json
```

For any of the options that are marked as `inherits` (for example, `webpack.debug`), it will default to whatever value was provided to the default configuration. For example, the value of `webpack.other.debug` in the above example will default to `True` because `webpack.debug = True`.

For information on how to render bundles from different configs, see the docs on *Rendering bundles into templates*.

1.2.3 Static View Examples

Here we'll go over a couple of example configurations for the asset static views and how they differ.

The simplest version can be used in development or production, and will serve the static assets using pyramid:

```
webpack.bundle_dir = webpack/bundles
```

If you're running in production and want to serve the assets from a CDN, you can instead use a static_view_name:

```
# Don't need a webpack.bundle_dir
webpack.static_view_name = //my.cdn.com/
```

And if you want to full control over how the static views are set up, you can disable them:

```
webpack.static_view = False
```

And set it up yourself:

```
for config_name, state in config.registry.webpack.iteritems():
    # You should use state.static_view_path as the path.
    # That value is used to generate the urls via request.static_url()
    config.add_static_view(name="//my.cdn.com/" + config_name,
                           path=state.static_view_path,
                           cache_max_age=64000)
```

1.3 Rendering bundles into templates

1.3.1 Jinja2

Rendering bundles into jinja2 uses the `webpack` tag.

```
{% webpack 'main' %}
  <script type="text/javascript" src="{{ ASSET.url }}"></script>
{% endwebpack %}
```

Inside of the `webpack` block you will have access to an `ASSET` variable that has a `url`, `name`, and `path`. The text inside of the block will be repeated once per chunk that is in the bundle.

To use a different webpack config, prefix the name of the bundle with that config name and a colon:

```
{% webpack 'other_config:mybundle' %}
  <script type="text/javascript" src="{{ ASSET.url }}"></script>
{% endwebpack %}
```

And if you would like to filter the bundle by one or more file extensions, you can pass that in as a second argument (space delimited string).

```
{% webpack 'mybundle', '.js .js.gz' %}
  <script type="text/javascript" src="{{ ASSET.url }}"></script>
{% endwebpack %}
```

1.3.2 Chameleon

Chameleon templates should just make a call directly to the `get_bundle()` method.

```
<script type="text/javascript"
  tal:repeat="asset request.webpack().get_bundle('main')"
  src="${asset.url}">
</script>
```

To use a different webpack config, pass in the name of that config to `request.webpack()`:

```
<script type="text/javascript"
  tal:repeat="asset request.webpack('other_config').get_bundle('main')"
  src="${asset.url}">
</script>
```

And if you would like to filter the bundle by one or more file extensions, you can pass them in as the second argument to `get_bundle()`:

```
<script type="text/javascript"
  tal:repeat="asset request.webpack().get_bundle('main', ['.js', '.js.gz'])"
  src="${asset.url}">
</script>
```

1.4 Changelog

1.4.1 0.1.0 - 2016/10/23

- Initial release

CHAPTER 2

API Reference

2.1 pyramid_webpack package

2.1.1 Submodules

pyramid_webpack.jinja2ext module

Jinja2 extension for pyramid_webpack

class `pyramid_webpack.jinja2ext.`**`WebpackExtension`**(*environment*)

Bases: `jinja2.ext.Extension`

Extension for jinja2.

Examples

```
{% webpack 'main' %}
    <link rel="stylesheet" type="text/css" href="{{ ASSET.url }}">
{% endwebpack %}
```

```
{% webpack 'other_config:main', 'js', 'js.gz' %}
    <script type="text/javascript" src="{{ ASSET.url }}"></script>
{% endwebpack %}
```

`identifier` **= 'pyramid_webpack.jinja2ext.WebpackExtension'**

`parse`(*parser*)

`tags` **= set([u'webpack'])**

2.1.2 Module contents

pyramid_webpack

class `pyramid_webpack.`**`StaticResource`**(*path*)

Bases: `object`

Wrapper around a filepath or asset path

classmethod **`create`**(*path*, *root_package*)

Create a StaticResource, setting the package if needed

open()
Open a stream object to the resource data

class pyramid_webpack.**Webpack**(*request*, *name='DEFAULT'*)
Bases: `object`

Wrapper object for the public webpack API

get_bundle(*bundle_name*, *extensions=None*)
Get all the chunks contained in a bundle

stats
Load and cache the webpack stats file

class pyramid_webpack.**WebpackState**(*settings*, *root_package_name='pyramid_webpack'*, *name='DEFAULT'*)
Bases: `object`

Wrapper for all webpack configuration and cached data

load_stats(*cache=None*, *wait=None*)
Load and cache the webpack-stats file

pyramid_webpack.**get_webpack**(*request*, *name='DEFAULT'*)
Get the Webpack object for a given webpack config.

Called at most once per request per config name.

pyramid_webpack.**includeme**(*config*)
Add pyramid_webpack methods and config to the app

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index