pyramid_frontendDocumentation

Release 0.4.2.dev

Scott Torborg

February 13, 2016

Contents

Scott Torborg - Cart Logic

`pyramid_frontend` provides:

- Theme / template handling.
- Theme switching based on the `request`.
- Theme configuration.
- Theme stacking (one theme can inherit from another).
- Image filtering / serving.
- Asset handling and compilation.
- Uses Mako, PIL, require.js, and LESS.

CHAPTER 1

Contents

1.1 Quick Start

1.1.1 Install

Install with pip:

```
$ pip install pyramid_frontend
```

1.1.2 Integrate with a Pyramid App

Include pyramid_frontend, by calling `config.include('pyramid_frontend')` or adding pyramid_frontend to `pyramid.includes`.

Configure the following settings:

- `pyramid_frontend.compiled_asset_dir`
- `pyramid_frontend.original_image_dir`
- `pyramid_frontend.processed_image_dir`
- `pyramid_frontend.compile`

1.1.3 Add Themes

Register at least one theme, using the `config.add_theme(theme)` directive. You can also pass a dotted string (e.g. `myapp.themes.foo.FooTheme`) which will be resolved relative to the calling module.

Other possible mechanisms for theme registration which may be added later are a setuptools entrypoint or a settings key.

- Themes are subclasses of the `pyramid_frontend.Theme` class.
- Class attributes or properties can be set for resource configuration.
- Paths are interpreted as relative to the directory of the module containing the class definition.
- An example:

```
class MyTheme(Theme):
    template_dir = ...
    static_dir = ...
    image_filters = {
        'detail': ...
        'thumb': ...
    }
    assets = {
        'main-js': ('static/js/main.js', 'requirejs'),
        'main-less': ('static/css/main.less', 'less'),
    }
```

1.1.4 Use a Theme

Configure your application to use a theme, with one of the following methods:

- Specify the `pyramid_frontend.theme` setting key.
- Call `config.set_theme_strategy(func)` with a function that will return the theme to use.
- An example:

```
def mobile_theme_strategy(request):
    if request.is_mobile and not request.session.get('use_desktop'):
        return 'my-mobile-theme'
    else:
        return 'my-desktop-theme'
```

Inside your app, specify a .html or .txt renderer. It will be rendered using the currently active theme (or call the theme strategy function to determine which theme to use).

The `request` object has a few added methods.

- `request.asset_tag(key)` - Generate an asset tag (either a script tag or stylesheet tag, or some combination thereof) for a corresponding asset key. In production, this will point to a concatenated / minified file.
- `request.image_url(name, original_ext, filter_key)` - Generate a URL for an image as processed by the specified filter chain.
- `request.image_tag(name, original_ext, filter_key, **kwargs)` - Generate an img tag for an image as processed by the specified filter chain.
- `request.image_original_path(name, original_ext)` - Return the filesystem path to the original file for this image.
- `request.theme` is a reified property on `request` - Return the theme instance that will be used to serve this request.

1.1.5 Compile Assets

When using in production, call `pcompile production.ini` to generate static assets, or call `pyramid_frontend.compile(registry.settings)`.

1.2 Advanced Usage

1.2.1 Theme Inheritance

Themes can stack on top of another theme by subclassing them.

Templates

An inheriting theme's templates will layer on top of the superclass theme's templates. The renderer will attempt to resolve templates to the child-most class first, then traverse up the inheritance chain.

Inside a template, you can refer to files with the prefix `super:` to make the filename resolve in the theme that is being inherited from.

Image Filters

An inheriting theme's image filters will layer on top of the superclass theme's image filters. If an image filter of the same name is specified, the child class will override the superclass.

Assets

An inheriting theme's asset entry points will layer on top of the super class theme's entry points. If an entry point of the same name is specified, the child class will override the superclass.

1.2.2 Static Files

Each theme has exactly one static file directory. It will be served up at an underscore-prefixed path corresponding to the theme's key.

1.2.3 Asset Compilation

The `assets` dict attribute maps entry point names to a tuple of URL paths and asset type.

In development, simply call `request.asset_tag(key)` to generate an asset tag.

In production, assets must be compiled before that call. The asset compilation step does the following for each entry point in each theme:

- Resolve the entry point path to a filesystem path.
- Collect static dirs from the theme and superclasses for use in resolving references during the compilation process.
- Compile the asset by calling a `Compiler` instance with the theme and the asset entry point.
- Save the result to a file in `pyramid_frontend.compiled_asset_dir` with a filename based on the sha1 of the contents. - Collect all filenames for compiled files, mapping entry point name to filename.
- Write the filename to a file with a path like `<compiled asset dir>/<theme key>/<entry point>.map`.

For normal usage, you can compile assets simply with:

```
$ pcompile production.ini
```

Other options which can be useful are:

```
$ pcompile --no-minify production.ini
```

Print debugging output:

```
$ pcompile -vv production.ini
```

It's also possible to programmatically call the asset compilation step (for example, for embedding in other deployment tools), with the `compile()` function.

`pyramid_frontend.compile.`**`compile`**(*registry*, *minify=True*)
Compile static assets for all themes which are registered in `registry`.

1.3 API Reference

class `pyramid_frontend.theme.`**`Theme`**(*settings*)
Represents a collection of templates, static files, image filters, and configuration corresponding to a particular visual theme (or "skin") used by the application.

New themes are created by subclassing from this class. When passed to `config.add_theme()`, The subclass will be instantiated with the application's `settings` dict and prepared for use.

`assets` = {}

`cache_args` = None

`cache_impl` = None

`compile`(*minify=True*)

`compiled_asset_path`(*key*)

`image_filters` = []

`includes` = []

`keyed_static_dirs`

`lookup`

`lookup_nofilters`

`opt`(*key*, *default=<object object>*)

classmethod **`qualify_path`**(*path*)

`stacked_assets`

`stacked_image_filters`

`stacked_includes`

`static`(*path*)

`static_dir` = 'static'

`static_url_to_filesystem_path`(*url*)
Given a URL of the structure /_<theme key>/<path>, locate the static dir which corresponds to the theme key and re-qualify the <path> to that directory.

`template_dir` = 'templates'

template_dirs

classmethod traverse_attributes(*name*, *qualify_paths=False*)

pyramid_frontend.theme.**add_theme**(*config*, *cls*)
A Pyramid config directive to initialiaze and register a theme for use.

pyramid_frontend.theme.**default_theme_strategy**(*request*)
The default theme selection strategy: just checks the pyramid_frontend.theme settings key.

pyramid_frontend.theme.**includeme**(*config*)

pyramid_frontend.theme.**set_theme_strategy**(*config*, *strategy_func*)
A Pyramid config directive to set a customized theme-selection strategy for each request.

pyramid_frontend.theme.**theme**(*request*)
The theme instance that should be used for this request. This property is both lazily-evaluated and reified.

pyramid_frontend.assets.**asset_tag**(*request*, *key*, ***kwargs*)
Request method to render an HTML fragment containing tags which reference the supplied entry point. This will dispatch to the appropriate tag rendering function based on context and entry point type.

pyramid_frontend.assets.**includeme**(*config*)

class pyramid_frontend.images.**FilterChain**(*suffix*, *filters=()*, *extension='png'*, *width=None*, *height=None*, *no_thumb=False*, *pad=False*, *crop=False*, *crop_whitespace=False*, *background='white'*, *enlarge=False*, ***saver_kwargs*)
A chain of image filters (a.k.a. "pipeline") used to process images for a particular display context.

basename(*name*, *original_ext*)

run(*dest_path*, *image_data*)

run_chain(*image_data*)

write(*dest_path*, *filtered*)

exception pyramid_frontend.images.**MissingOriginal**(*path*, *chain*)

pyramid_frontend.images.**save_image**(*settings*, *name*, *original_ext*, *f*)

pyramid_frontend.images.**save_to_error_dir**(*settings*, *name*, *f*)
Save a questionable image (could not be verified by PIL) to a penalty box for investigation.

pyramid_frontend.images.**check**(*f*)
Given a file object, check to see if the contents is a valid image. If so, return the file format. Otherwise, raise exceptions.

class pyramid_frontend.compile.**ConsoleHandler**(*stream=None*)
A subclass of StreamHandler which behaves in the same way, but colorizes the log level before formatting it.

colors = {'INFO': '\x1b[1;32m', 'WARNING': '\x1b[1;33m', None: '\x1b[0m', 'CRITICAL': '\x1b[1;31m', 'ERROR': '

emit(*record*)
Emit a record.

If a formatter is specified, it is used to format the record, but before the formatter is applied the loglevel is colored.

pyramid_frontend.compile.**compile**(*registry*, *minify=True*)
Compile static assets for all themes which are registered in registry.

pyramid_frontend.compile.**configure_logging**(*verbosity*)
Configure logging for use with the asset compilation command.

pyramid_frontend.compile.**main**(*args=['/home/docs/checkouts/readthedocs.org/user_builds/pyramid-frontend/envs/latest/bin/sphinx-build', '-b', 'latex', '-D', 'language=en', '-d', '_build/doctrees', '.', '_build/latex']*)

Main entry point for the executable which compiles assets.

1.4 Contributing

Patches and suggestions are strongly encouraged! GitHub pull requests are preferred, but other mechanisms of feedback are welcome.

pyramid_frontend has a comprehensive test suite with 100% line and branch coverage, as reported by the excellent `coverage` module. To run the tests, simply run in the top level of the repo:

```
$ tox
```

This will also ensure that the Sphinx documentation builds correctly, and that there are no PEP8 or Pyflakes warnings in the codebase.

Any pull requests should preserve all of these things.

CHAPTER 2

Indices and Tables

- genindex
- modindex

p

A

B

C

D

E

F

I

K

L

M

O

P

Q

R

S

T

W