asphalt-templating

Release 2.0.1.post3

Oct 01, 2017

Contents

This Asphalt framework component provides a standardized interface for a number of different templating renderers:

- Django
- Jinja2
- Mako
- Tonnikala
- Tornado

Additional backends may be provided through third party plugins.

CHAPTER 1

Configuration

To configure a template renderer for your application, you need to choose a backend and then specify any necessary configuration values for it. The following backends are provided out of the box:

- `django`
- `jinja2`
- `mako`
- `tonnikala`
- `tornado`

Other backends may be provided by other components.

Once you've selected a backend, see its specific documentation to find out what configuration values you need to provide, if any. Configuration values are expressed as constructor arguments for the backend class:

```
components:
  templating:
    backend: mako
    package_paths:
      - myapp.somepackage/templates
```

This configuration publishes a `asphalt.templating.api.TemplateRenderer` lazy resource named `default` using the Mako backend, accessible as `ctx.mako`. The renderer will look for templates in the `templates` subdirectory of the `myapp.somepackage` package.

The same can be done directly in Python code as follows:

```
class ApplicationComponent(ContainerComponent):
    async def start(ctx: Context):
        self.add_component('templating', backend='mako',
                           package_paths=['myapp.somepackage/templates'])
        await super().start()
```

1.1 Multiple renderers

If you need to configure multiple renderers, you can do so by using the `renderers` configuration option:

```
components:
  templating:
    renderers:
      django:
        package_paths:
          - myapp.somepackage/templates/django
      jinja2:
        package_name: myapp.somepackage
        package_path: templates/jinja2
      mako:
        package_paths:
          - myapp.somepackage/templates/mako
      tonnikala:
        package_paths:
          - myapp.somepackage/templates/tonnikala
      tornado:
        package_path: myapp.somepackage/templates/tornado
      foobar:
        backend: jinja2
        context_attr: foo
        package_name: myapp.somepackage
        package_path: templates/jinja2
```

The above configuration creates 5 lazy resources of type `asphalt.templating.api.TemplateRendererProxy`:

- `django` as `ctx.django`
- `jinja2` as `ctx.jinja2`
- `mako` as `ctx.mako`
- `tonnikala` as `ctx.tonnikala`
- `tornado` as `ctx.tornado`
- `foobar` as `ctx.foo`

CHAPTER 2

Using template renderers

Using renderers is quite straightforward. For example, to render a Jinja2 template named `templatefile.html`:

```
async def handler(ctx):
    text = ctx.jinja2.render('templatefile.html', somevariable='foo')
```

This example assumes a configuration with a Jinja2 renderer and a Jinja2 template file named `templatefile.html` in the designated template directory.

To directly render a template string:

```
async def handler(ctx):
    text = ctx.jinja2.render_string('This is foo: {{ foo }}', somevariable='foo')
```

CHAPTER 3

Writing new renderer backends

If you wish to implement an alternate method of template rendering, you can do so by subclassing the `TemplateRenderer` class. There is only one method implementors must override:

- `render()`

If you want your renderer to be available as a backend for `TemplatingComponent`, you need to add the corresponding entry point for it. Suppose your serializer class is named `AwesomeRenderer`, lives in the package `foo.bar.awesome` and you want to give it the alias `awesome`, add this line to your project's `setup.py` under the `entry_points` argument in the `asphalt.templating.renderers` namespace:

```
setup(
    # (...other arguments...)
    entry_points={
        'asphalt.templating.renderers': [
            'awesome = foo.bar.awesome:AwesomeRenderer'
        ]
    }
)
```

CHAPTER 4

Version history

This library adheres to Semantic Versioning.

2.0.1 (2017-06-04)

- Added compatibility with Asphalt 4.0

2.0.0 (2017-04-11)

- **BACKWARD INCOMPATIBLE** Migrated to Asphalt 3.0
- **BACKWARD INCOMPATIBLE** Renamed the `asphalt.templating.util` module to `asphalt.templating.utils`
- Renderer resources are now added to the context using their actual types as well
- The Mako renderer now skips filesystem checks by default in production mode

1.0.0 (2015-05-21)

- Initial release
- API reference