django-assets Documentation

Release 0.12

Michael Elsdörfer

Feb 23, 2018

Contents

CHAPTER 1

Jinja2 support

`django-assets` strives to offer full support for the Jinja2 template language.

A Jinja2 extension is available as `webassets.ext.jinja2.AssetsExtension`. It will provide a `{% assets %}` tag that functions pretty much like the Django template version, except inheriting the more expressive syntax of Jinja. For example, filters may be specified as tuples:

```
{% assets filters=("coffeescript", "jsmin") ... %}
```

More exhaustive documentation of the Jinja2 tag can be here.

1.1 Installation

How you enable the Jinja2 extension depends on how you are integrating Jinja with Django. For example:

- If you are using Coffin, you don't have to do anything at all: The extension will be available at the moment `django-assets` is added to `INSTALLED_APPS`.
- If you are creating your Jinja2 environment manually, you can simply use its `extensions` parameter and specify `webassets.ext.jinja2.AssetsExtension`.

However, there is a minor difficulty if you intend to use the management command to manually build assets: Since that step involves parsing your templates, the command needs to know what other Jinja2 extensions you are using to successfully do so. Because there is no "one way" to integrate Jinja and Django, it can't determine the extensions you are using all by itself. Instead, it expects you to specify the `ASSETS_JINJA2_EXTENSIONS` setting. In most cases, you would simply to something like:

```
ASSETS_JINJA2_EXTENSIONS = JINJA2_EXTENSIONS
```

i.e. aliasing it to the actual setting you are using.

Again, if you are using Coffin, you may disgard this step as well, since your Coffin environment will automatically be used.

CHAPTER 2

Settings

There are a bunch of values which you can define in your Django `settings` module to modify the behaviour of `webassets`.

Note: This document places those values inside the `django_assets.settings` module. This is irrelevant. To change the values, you need to define them in your project's global settings.

`django_assets.settings.`**`ASSETS_ROOT`**
: The base directory to which all paths will be relative to, unless `load_paths` are given, in which case this will only serve as the output directory.

  In the url space, it is mapped to `urls`.

  By default, `STATIC_ROOT` will be used for this, or the older `MEDIA_ROOT` setting.

`django_assets.settings.`**`ASSETS_URL`**
: The url prefix used to construct urls for files in `directory`.

  To define url spaces for other directories, see `url_mapping`.

  By default, `STATIC_URL` will be used for this, or the older `MEDIA_URL` setting.

`django_assets.settings.`**`ASSETS_DEBUG`**
: Enable/disable debug mode. Possible values are:

  `False` Production mode. Bundles will be merged and filters applied.

  `True` Enable debug mode. Bundles will output their individual source files.

  "merge" Merge the source files, but do not apply filters.

`django_assets.settings.`**`ASSETS_AUTO_BUILD`**
: Controls whether bundles should be automatically built, and rebuilt, when required (if set to `True`), or whether they must be built manually be the user, for example via a management command.

  This is a good setting to have enabled during debugging, and can be very convenient for low-traffic sites in production as well. However, there is a cost in checking whether the source files have changed, so if you care about performance, or if your build process takes very long, then you may want to disable this.

By default automatic building is enabled.

django_assets.settings.**ASSETS_URL_EXPIRE**
If you send your assets to the client using a *far future expires* header (to minimize the 304 responses your server has to send), you need to make sure that assets will be reloaded by the browser when they change.

If this is set to `True`, then the Bundle URLs generated by webassets will have their version (see `Environment.versions`) appended as a querystring.

An alternative approach would be to use the `%(version)s` placeholder in the bundle output file.

The default behavior (indicated by a `None` value) is to add an expiry querystring if the bundle does not use a version placeholder.

django_assets.settings.**ASSETS_VERSIONS**
Defines what should be used as a Bundle `version`.

A bundle's version is what is appended to URLs when the `url_expire` option is enabled, and the version can be part of a Bundle's output filename by use of the `%(version)s` placeholder.

Valid values are:

timestamp The version is determined by looking at the mtime of a bundle's output file.

hash (default) The version is a hash over the output file's content.

False, None Functionality that requires a version is disabled. This includes the `url_expire` option, the `auto_build` option, and support for the %(version)s placeholder.

Any custom version implementation.

django_assets.settings.**ASSETS_MANIFEST**
A manifest persists information about the versions bundles are at.

The Manifest plays a role only if you insert the bundle version in your output filenames, or append the version as a querystring to the url (via the `url_expire` option). It serves two purposes:

- Without a manifest, it may be impossible to determine the version at runtime. In a deployed app, the media files may be stored on a different server entirely, and be inaccessible from the application code. The manifest, if shipped with your application, is what still allows to construct the proper URLs.
- Even if it were possible to determine the version at runtime without a manifest, it may be a costly process, and using a manifest may give you better performance. If you use a hash-based version for example, this hash would need to be recalculated every time a new process is started.

Valid values are:

"cache" (default) The cache is used to remember version information. This is useful to avoid recalculating the version hash.

"file:{path}" Stores version information in a file at {path}. If not path is given, the manifest will be stored as `.webassets-manifest` in `Environment.directory`.

"json:{path}" Same as "file:{path}", but uses JSON to store the information.

False, None No manifest is used.

Any custom manifest implementation.

django_assets.settings.**ASSETS_CACHE**
Controls the behavior of the cache. The cache will speed up rebuilding of your bundles, by caching individual filter results. This can be particularly useful while developing, if your bundles would otherwise take a long time to rebuild.

Possible values are:

`False` Do not use the cache.

`True` (default) Cache using default location, a `.webassets-cache` folder inside `directory`.

custom path Use the given directory as the cache directory.

`django_assets.settings.`**`ASSETS_CACHE_FILE_MODE`**
Controls the mode of files created in the cache. The default mode is 0600. Follows standard unix mode. Possible values are any unix mode, e.g.:

`0660` Enable the group read+write bits

`0666` Enable world read+write bits

`django_assets.settings.`**`ASSETS_JINJA2_EXTENSIONS`**
This is needed in some cases when you want to use `django-assets` with the Jinja 2 template system. It should be a list of extensions you are using with Jinja 2, using which it should be possible to construct a Jinja 2 environment which can parse your templates. For more information, see *Jinja2 support*.

`django_assets.settings.`**`ASSETS_MODULES`**
`django-assets` will automatically look for `assets.py` files in each application, where you can register your bundles. If you want additional modules to be loaded, you can define this setting. It expects a list of importable modules:

```
ASSETS_MODULES = [
    'myproject.assets'
]
```

django-assets helps you to integrate webassets into your Django application.

CHAPTER 3

Quickstart

First, add `django_assets` to your `INSTALLED_APPS` setting:

```
INSTALLED_APPS = (
    ...,
    'django_assets',
)
```

Create an `assets.py` file inside your *application* directory. This is where you define your assets, and like Django's `admin.py` files, they will automatically be picked up:

```
from django_assets import Bundle, register
js = Bundle('common/jquery.js', 'site/base.js', 'site/widgets.js',
            filters='jsmin', output='gen/packed.js')
register('js_all', js)
```

Note: Make sure your `assets.py` is inside a Django *application*, not in the *project*. That is, the path might be something like `my_project/my_application/assets.py`.

If you want to define assets in a different place, you can use the *ASSETS_MODULES* setting.

Finally, include the bundle you defined in the appropriate place within your templates:

```
{% load assets %}
{% assets "js_all" %}
    <script type="text/javascript" src="{{ ASSET_URL }}"></script>
{% endassets %}
```

`django-assets` will now automatically merge and compress your bundle's source files the first time the template is rendered, and will automatically update the compressed file every time a source file changes. If *ASSETS_DEBUG* is enabled, then each source file will be outputted individually instead.

3.1 Templates only

If you prefer, you can also do without defining your bundles in code, and simply define everything inside your template:

```
{% load assets %}
{% assets filters="jsmin", output="gen/packed.js", "common/jquery.js", "site/base.js",
↪ "site/widgets.js" %}
    <script type="text/javascript" src="{{ ASSET_URL }}"></script>
{% endassets %}
```

You can also pass in `depends` through templatetags with a slightly modified comma-delimated syntax, e.g. `depends="myfile.js,path/to/file.js"`.

3.2 The management command

`django-assets` also provides a management command, `manage.py assets`. It can be used to manually cause your bundles to be rebuilt:

```
$ ./manage.py assets build
Building asset: cache/site.js
Building asset: cache/ie7.js
Building asset: cache/site.css
```

Note that this is more difficult if you are defining your bundles within your templates, rather than in code. You then need to use the `--parse-templates` option, so the `build` command can find the bundles.

3.3 `staticfiles` integration

`django-assets` can integrate with Django's django.contrib.staticfiles.

3.4 Jinja2 support

See *Jinja2 support* if you want to use `django-assets` with the Jinja2 templating language.

3.5 Settings

See *Settings* for on overview of Django configuration values.

CHAPTER 4

webassets documentation

For further information, have a look at the complete webassets documentation, and in particular, the following topics:

- All about bundles
- Builtin filters
- Custom filters
- CSS compilers
- FAQ

Python Module Index

d

Index

A

D