django-staticfiles Documentation

Release 1.2.1

Jannis Leidel

Sep 27, 2017

Contents

This is a Django app that provides helpers for serving static files.

Django developers mostly concern themselves with the dynamic parts of web applications – the views and templates that render new for each request. But web applications have other parts: the static media files (images, CSS, Javascript, etc.) that are needed to render a complete web page.

For small projects, this isn't a big deal, because you can just keep the media somewhere your web server can find it. However, in bigger projects – especially those comprised of multiple apps – dealing with the multiple sets of static files provided by each application starts to get tricky.

That's what `staticfiles` is for:

> Collecting static files from each of your Django apps (and any other place you specify) into a single location that can easily be served in production.

The main website for django-staticfiles is github.com/jezdez/django-staticfiles where you can also file tickets.

Note: django-staticfiles is now part of Django (since 1.3) as `django.contrib.staticfiles`.

The django-staticfiles 0.3.X series will only receive security and data los bug fixes after the release of django-staticfiles 1.0. Any Django 1.2.X project using django-staticfiles 0.3.X and lower should be upgraded to use either Django >= 1.3's staticfiles app or django-staticfiles >= 1.0 to profit from the new features and stability.

You may want to chose to use django-staticfiles instead of Django's own staticfiles app since any new feature (additionally to those backported from Django) will be released first in django-staticfiles.

CHAPTER 1

Installation

- Use your favorite Python packaging tool to install `staticfiles` from PyPI, e.g.:

```
pip install django-staticfiles
```

  You can also install the in-development version of django-staticfiles with `pip install django-staticfiles==dev`.

- Added `"staticfiles"` to your `INSTALLED_APPS` setting:

```
INSTALLED_APPS = [
    # ...
    "staticfiles",
]
```

- Set your `STATIC_URL` setting to the URL that handles serving static files:

```
STATIC_URL = "/static/"
```

- In development mode (when `DEBUG = True`) the `runserver` command will automatically serve static files:

```
python manage.py runserver
```

- Once you are ready to deploy all static files of your site in a central directory (`STATIC_ROOT`) to be served by a real webserver (e.g. Apache, Cherokee, Lighttpd, Nginx etc.), use the `collectstatic` management command:

```
python manage.py collectstatic
```

  See the webserver's documentation for descriptions how to setup serving the deployment directory (`STATIC_ROOT`).

- (optional) In case you use Django's admin app, make sure the `ADMIN_MEDIA_PREFIX` setting is set correctly to a subpath of `STATIC_URL`:

```
ADMIN_MEDIA_PREFIX = STATIC_URL + "admin/"
```

CHAPTER 2

Differences to `django.contrib.staticfiles`

Features of `django-staticfiles` which Django's `staticfiles` **doesn't** support:

- Runs on Django 1.2.X.
- *`STATICFILES_EXCLUDED_APPS`* settings – A sequence of dotted app paths that should be ignored when searching for static files.
- *`STATICFILES_IGNORE_PATTERNS`* settings – A sequence of glob patterns of files and directories to ignore when running `collectstatic`.
- Legacy 'media' dir file finder – a staticfiles finder that supports the location for static files that a lot of 3rd party apps support (`staticfiles.finders.LegacyAppDirectoriesFinder`).

See the *Settings* docs for more information.

CHAPTER 3

Contents

Management Commands

collectstatic

Collects the static files from all installed apps and copies them to the *STATICFILES_STORAGE*.

Duplicate file names are resolved in a similar way to how template resolution works. Files are initially searched for in *STATICFILES_DIRS* locations, followed by apps in the order specified by the INSTALLED_APPS setting.

Some commonly used options are:

--noinput Do NOT prompt the user for input of any kind.

-i PATTERN or --ignore=PATTERN Ignore files or directories matching this glob-style pattern. Use multiple times to ignore more.

-n or --dry-run Do everything except modify the filesystem.

-l or --link Create a symbolic link to each file instead of copying.

--no-default-ignore Don't ignore the common private glob-style patterns 'CVS', '.*' and '*~'.

-c or --clear

New in version 1.1.

Clear the existing files before trying to copy or link the original file.

--no-post-process

New in version 1.1.

Don't call the *post_process()* method of the configured *STATICFILES_STORAGE* storage backend.

For a full list of options, refer to the collectstatic management command help by running:

```
$ python manage.py collectstatic --help
```

findstatic

Searches for one or more relative paths with the enabled finders:

```
$ python manage.py findstatic css/base.css admin/js/core.css
/home/special.polls.com/core/media/css/base.css
/home/polls.com/core/media/css/base.css
/home/polls.com/src/django/contrib/admin/media/js/core.js
```

By default, all matching locations are found. To only return the first match for each relative path, use the `--first` option:

```
$ python manage.py findstatic css/base.css --first
/home/special.polls.com/core/media/css/base.css
```

This is a debugging aid; it'll show you exactly which static file will be collected for a given path.

runserver

Overrides the core `runserver` command if the `staticfiles` app is installed (in `INSTALLED_APPS`) and adds automatic serving of static files and the following new options.

`--nostatic`

Use the `--nostatic` option to disable serving of static files with the `staticfiles` app entirely. This option is only available if the `staticfiles` app is in your project's `INSTALLED_APPS` setting.

Example usage:

```
django-admin.py runserver --nostatic
```

`--insecure`

Use the `--insecure` option to force serving of static files with the `staticfiles` app even if the `DEBUG` setting is `False`.

> **Warning:** By using this you acknowledge the fact that it's **grossly inefficient** and probably **insecure**.
>
> This is only intended for local development, should **never be used in production** and is only available if the `staticfiles` app is in your project's `INSTALLED_APPS` setting.

Example usage:

```
django-admin.py runserver --insecure
```

Helpers

Context processors

The `static` context processor

`context_processors.`**`static`**`()`

This context processor adds the `STATIC_URL` into each template context as the variable `{{ STATIC_URL }}`. To use it, make sure that `'staticfiles.context_processors.static'` appears somewhere in your `TEMPLATE_CONTEXT_PROCESSORS` setting.

Remember, only templates rendered with a `RequestContext` will have acces to the data provided by this (and any) context processor.

Template tags

static

`templatetags.staticfiles.`**`static`**`()`

New in version 1.1.

Uses the configued *`STATICFILES_STORAGE`* storage to create the full URL for the given relative path, e.g.:

```
{% load staticfiles %}
<img src="{% static "css/base.css" %}" />
```

The previous example is equal to calling the `url` method of an instance of *`STATICFILES_STORAGE`* with `"css/base.css"`. This is especially useful when using a non-local storage backend to deploy files to a CDN.

get_static_prefix

`templatetags.static.`**`get_static_prefix`**`()`

If you're not using `RequestContext`, or if you need more control over exactly where and how `STATIC_URL` is injected into the template, you can use the `get_static_prefix` template tag instead:

```
{% load static %}
<img src="{% get_static_prefix %}images/hi.jpg" />
```

There's also a second form you can use to avoid extra processing if you need the value multiple times:

```
{% load static %}
{% get_static_prefix as STATIC_PREFIX %}

<img src="{{ STATIC_PREFIX }}images/hi.jpg" />
<img src="{{ STATIC_PREFIX }}images/hi2.jpg" />
```

get_media_prefix

`templatetags.static.`**`get_media_prefix`**`()`

Similar to *`get_static_prefix()`* but uses the `MEDIA_URL` setting instead.

Storages

StaticFilesStorage

class storage.**StaticFilesStorage**

A subclass of the FileSystemStorage storage backend that uses the *STATIC_ROOT* setting as the base file system location and the *STATIC_URL* setting respectively as the base URL.

post_process(*paths*, ***options*)

New in version 1.1.

This method is called by the *collectstatic* management command after each run and gets passed the paths of found files, as well as the command line options.

The *CachedStaticFilesStorage* uses this behind the scenes to replace the paths with their hashed counterparts and update the cache appropriately.

CachedStaticFilesStorage

class storage.**CachedStaticFilesStorage**

New in version 1.1.

A subclass of the *StaticFilesStorage* storage backend which caches the files it saves by appending the MD5 hash of the file's content to the filename. For example, the file css/styles.css would also be saved as css/styles.55e7cbb9ba48.css.

The purpose of this storage is to keep serving the old files in case some pages still refer to those files, e.g. because they are cached by you or a 3rd party proxy server. Additionally, it's very helpful if you want to apply far future Expires headers to the deployed files to speed up the load time for subsequent page visits.

The storage backend automatically replaces the paths found in the saved files matching other saved files with the path of the cached copy (using the *post_process()* method). The regular expressions used to find those paths (storage.CachedStaticFilesStorage.cached_patterns) by default cover the @import rule and url() statement of Cascading Style Sheets. For example, the 'css/styles.css' file with the content

```
@import url("../admin/css/base.css");
```

would be replaced by calling the url() method of the CachedStaticFilesStorage storage backend, ultimatively saving a 'css/styles.55e7cbb9ba48.css' file with the following content:

```
@import url("/static/admin/css/base.27e20196a850.css");
```

To enable the CachedStaticFilesStorage you have to make sure the following requirements are met:

- the *STATICFILES_STORAGE* setting is set to 'staticfiles.storage.CachedStaticFilesStorage'
- the DEBUG setting is set to False
- you use the staticfiles static() template tag to refer to your static files in your templates
- you've collected all your static files by using the *collectstatic* management command

Since creating the MD5 hash can be a performance burden to your website during runtime, staticfiles will automatically try to cache the hashed name for each file path using Django's caching framework. If you want to override certain options of the cache backend the storage uses, simply specify a custom entry in the CACHES setting named 'staticfiles'. It falls back to using the 'default' cache backend.

Static file development view

staticfiles.views.**serve**(*request*, *path*)

This view function serves static files in development.

> **Warning:** This view will only work if DEBUG is True.
>
> That's because this view is **grossly inefficient** and probably **insecure**. This is only intended for local development, and should **never be used in production**.

This view is automatically enabled by runserver (with a DEBUG setting set to True). To use the view with a different local development server, add the following snippet to the end of your primary URL configuration:

```
from django.conf import settings

if settings.DEBUG:
    urlpatterns += patterns('staticfiles.views',
        url(r'^static/(?P<path>.*)$', 'serve'),
    )
```

Note, the begin of the pattern (r'^static/') should be your STATIC_URL setting.

URL patterns helper

staticfiles.urls.**staticfiles_urlpatterns**()

> **Warning:** This helper function will only work if DEBUG is True and your STATIC_URL setting is neither empty nor a full URL such as http://static.example.com/.

Since configuring the URL patterns is a bit finicky, there's also a helper function that'll do this for you.

This will return the proper URL pattern for serving static files to your already defined pattern list. Use it like this:

```
from staticfiles.urls import staticfiles_urlpatterns

# ... the rest of your URLconf here ...

urlpatterns += staticfiles_urlpatterns()
```

Settings

django.conf.settings.**STATIC_ROOT**

Default "" (Empty string)

The absolute path to the directory that contains static content after using *collectstatic*.

Example: "/home/example.com/static/"

When using the *collectstatic* management command this will be used to collect static files into, to be served under the URL specified as *STATIC_URL*.

This is a **required setting** to use *collectstatic* – unless you've overridden *STATICFILES_STORAGE* and are using a custom storage backend.

> **Warning:** This is not a place to store your static files permanently under version control!
>
> You should do that in directories that will be found by your *STATICFILES_FINDERS* (by default, per-app `'static'` subdirectories, and any directories you include in *STATICFILES_DIRS* setting). Files from those locations will be collected into *STATIC_ROOT*.

See also *STATIC_URL*.

`django.conf.settings.`**`STATIC_URL`**

Default `None`

URL that handles the files served from `STATIC_ROOT` and used by `runserver` in development mode (when `DEBUG = True`).

Example: `"/site_media/static/"` or `"http://static.example.com/"`

It must end in a slash if set to a non-empty value.

See also *STATIC_ROOT*.

`django.conf.settings.`**`STATICFILES_DIRS`**

Default `()`

This setting defines the additional locations the staticfiles app will traverse if the `FileSystemFinder` finder is enabled, e.g. if you use the *collectstatic* or *findstatic* management command or use the static file serving view.

This should be set to a list or tuple of strings that contain full paths to your additional files directory(ies) e.g.:

```
STATICFILES_DIRS = (
    "/home/special.polls.com/polls/static",
    "/home/polls.com/polls/static",
    "/opt/webfiles/common",
)
```

In case you want to refer to files in one of the locations with an additional namespace, you can **OPTIONALLY** provide a prefix as `(prefix, path)` tuples, e.g.:

```
STATICFILES_DIRS = (
    # ...
    ("downloads", "/opt/webfiles/stats"),
)
```

Example:

Assuming you have *STATIC_URL* set `'/static/'`, the *collectstatic* management command would collect the stats files in a `'downloads'` subdirectory of *STATIC_ROOT*.

This would allow you to refer to the local file `'/opt/webfiles/stats/polls_20101022.tar.gz'` with `'/static/downloads/polls_20101022.tar.gz'` in your templates, e.g.:

```
<a href="{{ STATIC_URL }}downloads/polls_20101022.tar.gz">
```

`django.conf.settings.`**`STATICFILES_IGNORE_PATTERNS`**

Default `()`

This setting defines patterns to be ignored by the *collectstatic* management command.

This should be set to a list or tuple of strings that contain file or directory names and may include an absolute file system path or a path relative to *STATIC_ROOT*, e.g.:

```
STATICFILES_IGNORE_PATTERNS = (
    "*.txt",
    "tests",
    "css/*.old",
    "/opt/webfiles/common/*.txt",
    "/opt/webfiles/common/temp",
)
```

New in version 1.2.

django.conf.settings.**STATICFILES_EXCLUDED_APPS**

Default `()`

A sequence of app paths that should be ignored when searching for static files, e.g.:

```
STATICFILES_EXCLUDED_APPS = (
    'annoying.app',
    'old.company.app',
)
```

django.conf.settings.**STATICFILES_STORAGE**

Default `'staticfiles.storage.StaticFileStorage'`

The file storage engine to use when collecting static files with the *collectstatic* management command.

django.conf.settings.**STATICFILES_FINDERS**

Default `('staticfiles.finders.FileSystemFinder', 'staticfiles.finders.AppDirectoriesFinder')`

The list of finder backends that know how to find static files in various locations.

The default will find files stored in the *STATICFILES_DIRS* setting (using `staticfiles.finders.FileSystemFinder`) and in a `static` subdirectory of each app (using `staticfiles.finders.AppDirectoriesFinder`)

One finder is disabled by default: `staticfiles.finders.DefaultStorageFinder`. If added to your *STATICFILES_FINDERS* setting, it will look for static files in the default file storage as defined by the `DEFAULT_FILE_STORAGE` setting.

Note: When using the `AppDirectoriesFinder` finder, make sure your apps can be found by staticfiles. Simply add the app to the `INSTALLED_APPS` setting of your site.

Static file finders are currently considered a private interface, and this interface is thus undocumented.

To ease the burden of upgrading a Django project from a non-`staticfiles` setup, the optional finder backend `staticfiles.finders.LegacyAppDirectoriesFinder` is shipped as part of `django-staticfiles`.

When added to the *STATICFILES_FINDERS* setting, it'll enable `staticfiles` to use the `media` directory of the apps in `INSTALLED_APPS`, similarly `staticfiles.finders.AppDirectoriesFinder`.

This is especially useful for 3rd party apps that haven't been switched over to the `static` directory instead. If you want to use both `static` **and** `media`, don't forget to have `staticfiles.finders.AppDirectoriesFinder` in the *STATICFILES_FINDERS*, too.

Changelog

v1.2.1 (2012-02-16)

- Backported a change from Django trunk that prevents openening too many files at once when running the `collectstatic` management command.

v1.2 (2012-02-12)

- Added `STATICFILES_IGNORE_PATTERNS` setting to globally ignore files when running the `collectstatic` management command.
- Refactored `CachedFilesMixin` and management command to only post process the collected files if really needed.
- Added support for URL fragment to the `CachedStaticFilesStorage`.
- Stopped using versiontools again as it caused installation time issues.

v1.1.2 (2011-08-25)

- Fixed a minor bug in how django-appconf was used.

v1.1.1 (2011-08-22)

- Fixed resolution of relative paths in `CachedStaticFilesStorage`.
- Started to use django-appconf and versiontools.

v1.1 (2011-08-18)

- Pulled all changes from upstream Django:
  - `static` template tag to refer to files saved with the `STATICFILES_STORAGE` storage backend. It'll use the storage `url` method and therefore supports advanced features such as serving files from a cloud service.
  - `CachedStaticFilesStorage` which caches the files it saves (when running the `collectstatic` management command) by appending the MD5 hash of the file's content to the filename. For example, the file `css/styles.css` would also be saved as `css/styles.55e7cbb9ba48.css`
  - Added a `staticfiles.storage.staticfiles_storage` instance of the configured `STATICFILES_STORAGE`.
  - `--clear` option for the management command which clears the target directory (by default `STATIC_ROOT`) before collecting
  - Stop trying to show directory indexes in the included `serve` view.
  - Correctly pass kwargs to the URL patterns when using the static URL patterns helper.

- Use sys.stdout in management command, not self.stdout which was only introduced in a later Django version.
- Refactored AppSettings helper class to be only a proxy for Django's settings object instead of a singleton on its own.
- Updated list of supported Django versions: 1.2.X, 1.3.X and 1.4.X
- Updated list of supported Python versions: 2.5.X, 2.6.X and 2.7.X

v1.0.1 (2011-03-28)

- Fixed an encoding related issue in the tests.
- Updated tox configuration to use 1.3 release tarball.
- Extended docs a bit.

v1.0 (2011-03-23)

Note: `django-staticfiles` is a backport of the staticfiles app in Django contrib. If you're upgrading from `django-staticfiles` < `1.0`, you'll need to make a few changes. See changes below.

- Renamed `StaticFileStorage` to `StaticFilesStorage`.
- Application files should now live in a `static` directory in each app (previous versions of `django-staticfiles` used the name `media`, which was slightly confusing).
- The management commands `build_static` and `resolve_static` are now called `collectstatic` and `findstatic`.
- The settings `STATICFILES_PREPEND_LABEL_APPS` and `STATICFILES_MEDIA_DIRNAMES` were removed.
- The setting `STATICFILES_RESOLVERS` was removed, and replaced by the new `STATICFILES_FINDERS` setting.
- The default for `STATICFILES_STORAGE` was renamed from `staticfiles.storage.StaticFileStorage` to `staticfiles.storage.StaticFilesStorage`
- If using `runserver` for local development (and the setting `DEBUG` setting is `True`), you no longer need to add anything to your URLconf for serving static files in development.

v0.3.4 (2010-12-25)

- Minor documentation update.

v0.3.3 (2010-12-23)

Warning: django-staticfiles was added to Django 1.3 as a contrib app.

The django-staticfiles 0.3.X series will only receive security and data los bug fixes after the release of django-staticfiles 1.0. Any Django 1.2.X project using django-staticfiles 0.3.X and lower should be upgraded to use either Django 1.3's staticfiles app or django-staticfiles >= 1.0 to profit from the new features and stability.

You may want to chose to use django-staticfiles instead of Django's own staticfiles app since any new feature (additionally to those backported from Django) will be released first in django-staticfiles.

- Fixed an issue that could prevent the `build_static` management command to fail if the destination storage doesn't implement the `listdir` method.
- Fixed an issue that caused non-local storage backends to fail saving the files when running `build_static`.

v0.3.2 (2010-08-27)

- Minor cosmetic changes
- Moved repository back to Github: http://github.com/jezdez/django-staticfiles

v0.3.1 (2010-08-21)

- Added Sphinx config files and split up README.

  Documetation now available under django-staticfiles.readthedocs.org

v0.3.0 (2010-08-18)

- Added resolver API which abstract the way staticfiles finds files.
- Added staticfiles.urls.staticfiles_urlpatterns to avoid the catch-all URLpattern which can make top-level urls.py slightly more confusing. From Brian Rosner.
- Minor documentation changes
- Updated testrunner to work with Django 1.1.X and 1.2.X.
- Removed custom code to load storage backend.

v0.2.0 (2009-11-25)

- Renamed build_media and resolve_media management commands to build_static and resolve_media to avoid confusions between Django's use of the term "media" (for uploads) and "static" files.
- Rework most of the internal logic, abstracting the core functionality away from the management commands.
- Use file system storage backend by default, ability to override it with custom storage backend
- Removed –interactive option to streamline static file resolving.
- Added extensive tests
- Uses standard logging

v0.1.2 (2009-09-02)

- Fixed a typo in settings.py
- Fixed a conflict in build_media (now build_static) between handling non-namespaced app media and other files with the same relative path.

v0.1.1 (2009-09-02)

- Added README with a bit of documentation :)

v0.1.0 (2009-09-02)

- Initial checkin from Pinax' source.
- Will create the STATIC_ROOT directory if not existent.

Python Module Index

S

Index