django-hosts Documentation

Release 0.3

Jannis Leidel and contributors

December 29, 2014

Contents

This Django app routes requests for specific hosts to different URL schemes defined in modules called "hostconfs".

For example, if you own `example.com` but want to serve specific content at `api.example.com` and `beta.example.com`, add the following to a `hosts.py` file:

```
from django_hosts import patterns, host

host_patterns = patterns('path.to',
    host(r'api', 'api.urls', name='api'),
    host(r'beta', 'beta.urls', name='beta'),
)
```

This causes requests to `{api,beta}.example.com` to be routed to their corresponding URLconf. You can use your `urls.py` as a template for these hostconfs.

Patterns are evaluated in order. If no pattern matches, the request is processed in the usual way, ie. using the standard `ROOT_URLCONF`.

The patterns on the left-hand side are regular expressions. For example, the following `ROOT_HOSTCONF` setting will route `foo.example.com` and `bar.example.com` to the same URLconf.

```
from django_hosts import patterns, host

host_patterns = patterns('',
    host(r'(foo|bar)', 'path.to.urls', name='foo-or-bar'),
)
```

CHAPTER 1

Installation

First, install the app with your favorite package manager, e.g.:

```
pip install django-hosts
```

Alternatively, use the repository on Github.

Then configure your Django site to use the app:

1. Add `'django_hosts'` to your `INSTALLED_APPS` setting.
2. Add `'django_hosts.middleware.HostsMiddleware'` to your `MIDDLEWARE_CLASSES` setting.
3. Create a module containing your default host patterns, e.g. in the `hosts.py` file next to your `urls.py`.
4. Set the `ROOT_HOSTCONF` setting to the dotted Python import path of the module containing your host patterns, e.g.:

   ```
   ROOT_HOSTCONF = 'mysite.hosts'
   ```

5. Set the `DEFAULT_HOST` setting to the *name* of the host pattern you want to refer to as the default pattern. It'll be used if no other pattern matches or you don't give a name to the `host_url` template tag.

`django-hosts` uses versiontools to manage version numbers following **PEP 386**.

CHAPTER 2

Usage

Patterns being regular expressions allows setups to feature dynamic (or "wildcard") host schemes:

```
from django.conf import settings
from django_hosts import patterns, host

host_patterns = patterns('',
    host(r'www', settings.ROOT_URLCONF, name='www'),
    host(r'(\w+)', 'path.to.custom_urls', name='wildcard'),
)
```

Here, requests to `www.example.com` will be routed as normal but a request to `admin.example.com` is routed to `path.to.custom_urls`.

As patterns are matched in order, we placed `www` first as it otherwise would have matched against `\w+` and thus routed to the wrong destination.

Alternatively, we could have used negative lookahead, given the value of the `ROOT_URLCONF` setting:

```
from django_hosts import patterns, host

host_patterns = patterns('',
    host(r'(?!www)\w+', 'path.to.custom_urls', name='wildcard'),
)
```

CHAPTER 3

Settings

`django.conf.settings.`**`ROOT_HOSTCONF`** (*required*)
: The dotted Python import path of the module containing your host patterns. Similar to `ROOT_URLCONF`.

`django.conf.settings.`**`DEFAULT_HOST`** (*required*)
: The *name* of the host pattern you want to refer to as the default pattern. Used if no other host pattern matches or no host name is passed to the `host_url()` template tag.

`django.conf.settings.`**`PARENT_HOST`** (*optional*)
: The parent domain name to be *appended to the reversed domain* (e.g. using the `host_url()` template tag).

CHAPTER 4

Contents

4.1 Per-host callbacks

Parsing the host from `request.get_host()` and lookup its corresponding object instance (e.g. site) in every view violates DRY. If these dynamic hosts had a lot of views this would become particularly unwieldy.

To remedy this, you can optionally specify a callback method to be called if your host matches.

Simply define a callback function:

```
from django.shortcuts import get_object_or_404
from django.contrib.auth.models import User

def custom_fn(request, username):
    request.viewing_user = get_object_or_404(User, username=username)
```

..and pass it as the `callback` paramter to the `host` object:

```
from django.conf import settings
from django_hosts import patterns, host

host_patterns = patterns('',
    host(r'www', settings.ROOT_URLCONF, name='www'),
    host(r'(?P<username>\w+)', 'path.to.custom_urls',
         callback='path.to.custom_fn', name='with-callback'),
)
```

This example avoids the duplicated work in every view by attaching a `viewing_user` instance to the request object. Views referenced by the "dynamic" URLconf can now assume that this object exists.

The custom method is called with the `request` object and any named captured arguments, similar to regular Django url processing.

Callbacks may return either `None` or an `HttpResponse` object.

- If it returns `None`, the request continues to be processed and the appropriate view is eventually called.
- If a callback returns an `HttpResponse` object, that `HttpResponse` is returned to the client without any further processing.

Note: There are a few things to keep in mind when using the callbacks:

- Callbacks are executed with the URLconf set to the second argument in the `host_patterns` list. For example, in the example above, the callback will be executed with the URLconf as `path.to.custom_urls` and not the default URLconf.

- This can cause problems when reversing URLs within your callback as they may not be "visible" to `django.core.urlresolvers.reverse()` as they are specified in (eg.) the default URLconf.
- To remedy this, specify the `urlconf` parameter when calling `reverse()`.
- When using dynamic hosts based on user input, ensure users cannot specify names that conflict with static subdomains such as "www" or their subdomain will not be accessible.
- Don't forget to add `handler404` and `handler500` entries for your custom URLconfs.

4.1.1 Included callbacks

`django-hosts` includes the following callbacks:

4.2 Template tags

`django_hosts.templatetags.hosts.`**`host_url`**(*view_name*[, *view_args*, *view_kwargs*], *host_name*[, *host_args*, *host_kwargs*])

Now if you want to actually refer to the full URLs in your templates you can use the included `host_url` template tag. So imagine having a host pattern of:

```
from django.conf import settings
from django_hosts import patterns, host

host_patterns = patterns('',
    host(r'admin', settings.ROOT_URLCONF, name='our-admin'),
)
```

and a `ROOT_URLCONF` of:

```
from django.conf.urls.defaults import patterns, url

urlpatterns = patterns('mysite.admin',
    url(r'^dashboard/$', 'dashboard', name='dashboard'),
)
```

then this example will create a link to the admin dashboard:

```
{% load hosts %}

<a href="{% host_url dashboard on our-admin %}">Admin dashboard</a>
```

which will be rendered as:

```
<a href="//admin/dashboard/">Admin dashboard</a>
```

Note: The double slash at the beginning of the href is an easy way to not have to worry about which scheme (http or https) is used. Your browser will automatically choose the currently used scheme. If you're on `https://mysite.com/` a link with an href of `//mysite.com/about/` would actually point to `https://mysite.com/about/`.

For more information see the The protocol-relative URL article by Paul Irish or the appropriate section in RFC 3986.

4.2.1 Fully qualified domain names (FQDN)

In case you want to append a default domain name to the domain part of the rendered URL you can simply set the `PARENT_HOST`, e.g:

```
PARENT_HOST = 'example.com'
```

This would render the link of the previous section as:

```
<a href="//admin.example.com/dashboard/">Admin dashboard</a>
```

Alternatively – in case you don't want to append this parent domain to all URLs you can also spell out the domain in the host pattern:

```
from django.conf import settings
from django_hosts import patterns, host

host_patterns = patterns('',
    host(r'admin\.example\.com', settings.ROOT_URLCONF, name='admin'),
)
```

If your host pattern contains an argument (or key argument), like:

```
from django.conf import settings
from django_hosts import patterns, host

host_patterns = patterns('',
    host(r'www', settings.ROOT_URLCONF, name='homepage'),
    host(r'(\w+)', 'path.to.support_urls', name='wildcard'),
    host(r'(?P<username>\w+)', 'path.to.user_urls', name='user-area'),
)
```

you can also easily pass arguments to the `host_url()` template tag:

```
{% load hosts %}

<a href="{% host_url user-dashboard on user-area username='johndoe' %}">John's dashboard</a>
<a href="{% host_url faq-index on wildcard 'help' %}">FAQ</a>
```

Which will be rendered (with a `PARENT_HOST` of `'example.com'`) as:

```
<a href="//johndoe.example.com/">John's dashboard</a>
<a href="//help.example.com/faq/">FAQ</a>
```

4.3 Reference

4.3.1 `hosts` and `patterns`

`django_hosts.defaults.`**`patterns`**(*prefix*, **args*)

The function to define the list of hosts (aka host confs), e.g.:

```
from django_hosts import patterns

host_patterns = patterns('path.to',
    (r'www', 'urls.default', 'default'),
    (r'api', 'urls.api', 'api'),
)
```

Parameters

- **prefix** (*str*) – the URLconf prefix to pass to the host object
- ***args** – a list of `hosts` instances or an iterable thereof

class `django_hosts.defaults.`**host**(*regex*, *urlconf*, *name*, *callback=None*, *prefix=''*)

The host object used in host conf together with the `django_hosts.defaults.patterns()` function, e.g.:

```
from django_hosts import patterns, host

host_patterns = patterns('path.to',
    host(r'www', 'urls.default', name='default'),
    host(r'api', 'urls.api', name='api'),
)
```

Parameters

- **regex** (*str*) – a regular expression to be used to match the request's host.
- **urlconf** (*str*) – the dotted path of a URLconf module of the host
- **callback** (*callable or str*) – a callable or the dotted path of a callable to be used when matching has happened
- **prefix** (*str*) – the prefix to apply to the `urlconf` parameter

add_prefix(*prefix=''*)

Adds the prefix string to a string-based urlconf.

4.3.2 Reversal with `reverse_host` and `reverse_full`

`django_hosts.reverse.`**reverse_full**(*host*, *view*, *host_args=None*, *host_kwargs=None*, *view_args=None*, *view_kwargs=None*)

Given the host and view name and the appropriate parameters, reverses the fully qualified URL, e.g.:

```
>>> from django.conf import settings
>>> settings.ROOT_HOSTCONF = 'mysite.hosts'
>>> settings.PARENT_HOST = 'example.com'
>>> from django_hosts.reverse import reverse_full
>>> reverse_full('www', 'about')
'//www.example.com/about/'
```

Parameters

- **host** – the name of the host
- **view** – the name of the view

Host_args the host arguments

Host_kwargs the host keyed arguments

View_args the arguments of the view

View_kwargs the keyed arguments of the view

Return type fully qualified URL with path

django_hosts.reverse.**reverse_host**(*host*, *args=None*, *kwargs=None*)

Given the host name and the appropriate parameters, reverses the host, e.g.:

```
>>> from django.conf import settings
>>> settings.ROOT_HOSTCONF = 'mysite.hosts'
>>> settings.PARENT_HOST = 'example.com'
>>> from django_hosts.reverse import reverse_host
>>> reverse_host('with_username', 'jezdez')
'jezdez.example.com'
```

Parameters **name** – the name of the host as specified in the hostconf

Args the host arguments to use to find a matching entry in the hostconf

Kwargs similar to args but key value arguments

Raises **django.core.urlresolvers.NoReverseMatch** if no host matches

Return type reversed hostname

4.3.3 HostSiteManager model manager

4.4 Changelog

4.4.1 0.3 (2011/09/30)

- Consolidated reversal internals.
- Removed unfinished support for the Django Debug Toolbar.
- Added a custom callback which uses Django's sites app to retrieve a `Site` instance matching the current host, setting `request.site`.
- Extended tests dramatically (100% coverage).
- Added docs at http://django-hosts.rtfd.org
- Stopped preventing the name 'default' for hosts.

4.4.2 0.2.1 (2011/05/31)

- Fixed issue related to the `PARENT_HOST` setting when used with empty host patterns.
- Stopped automatically emulating hosts in debug mode.

4.4.3 0.2 (2011/05/31)

- **BACKWARDS INCOMPATIBLE** Renamed the package to `django_hosts`

  Please change your import from:

  ```
  from hosts import patterns, hosts
  ```

  to:

  ```
  from django_hosts import patterns, hosts
  ```

- **BACKWARDS INCOMPATIBLE** Changed the data type that the `django_hosts.patterns` function returns to be a list instead of a SortedDict to follow conventions of Django's URL patterns. You can use that for easy extension of the patterns, e.g.:

```
from django_hosts import patterns, host
from mytemplateproject.hosts import host_patterns

host_patterns += patterns('',
    host('www2', 'mysite.urls.www2', name='www2')
)
```

- Extended tests to have full coverage.
- Fixed prefix handling.

4.4.4 0.1.1 (2011/05/30)

- Fixed docs issues.
- Use absolute imports where possible.

4.4.5 0.1 (2011/05/29)

- Initial release with middleware, reverse and templatetags.

CHAPTER 5

Issues

For any bug reports and feature requests, please use the Github issue tracker.

CHAPTER 6

Thanks

Many thanks to the folks at playfire for releasing their django-dynamic-subdomains app, which was the inspiration for this app.

d

A

D

H

P

R