django-hosts Documentation

Release 3.0

Jannis Leidel and contributors

Nov 20, 2017

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.

You can find the full docs here: django-hosts.rtfd.org

Then configure your Django site to use the app:

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

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

6. 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.

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'),
)
```

In your templates you can use the *`host_url()`* template tag to reverse a URL the way you're used to it with Django's url template tag:

```
{% load hosts %}
<a href="{% host_url 'homepage' host 'www' %}">Home</a> |
<a href="{% host_url 'account' host 'wildcard' request.user.username %}">Your Account
↪</a> |
```

Since the template tag will always automatically fall back to your default host (as defined by *`DEFAULT_HOST`*) you can leave off the `host` parameter as well.

You can even *override the url tag* that comes with Django to simplify reversing URLs in your templates:

```
<a href="{% url 'homepage' %}">Home</a> |
<a href="{% url 'account' host 'wildcard' request.user.username %}">Your Account</a> |
```

On the Python side of things like your views you can easily do the same as with Django's own reverse function. Simply use the *reverse()* function for that:

```
from django.shortcuts import render
from django_hosts.resolvers import reverse

def homepage(request):
    homepage_url = reverse('homepage', host='www')
    return render(request, 'homepage.html', {'homepage_url': homepage_url})
```

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).

django.conf.settings.**HOST_SCHEME** (*optional*)
: The scheme to prepend host names with during reversing, e.g. when using the *`host_url()`* template tag. Defaults to `'//'`.

django.conf.settings.**HOST_PORT** (*optional*)
: New in version 1.0.

  The port to append to host names during reversing, e.g. when using the *`host_url()`* template tag. Defaults to `''` (empty string).

django.conf.settings.**HOST_SITE_TIMEOUT** (*optional*)
: The time to cache the host in the default cache backend, in seconds, when using the *`cached_host_site()`* callback. Defaults to `3600`.

CHAPTER 4

More docs

4.1 Template tags

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

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 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' host '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.

Changed in version 0.5.

You can override the used default scheme with the *HOST_SCHEME* setting.

Changed in version 1.0.

You can override the individually used scheme with the *scheme* parameter.

4.1.1 Override the default url template tag

New in version 1.0.

In case you don't like adding `{% load hosts %}` to each and every template that you reverse an URL in you can automatically override the url template tag that is built into Django by adding `'django_hosts.templatetags.hosts_override'` to the `TEMPLATES['OPTIONS']['builtins']` list.

It won't hurt to have some `{% load hosts %}` in some templates and the *host_url()* template tag will also still work. But that will at least enable the use of templates in 3rd party apps, for example.

4.1.2 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'),
)
```

4.1.3 Host and URL pattern parameters

If your host pattern contains an parameter (or keyed parameter), 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+).users', 'path.to.user_urls', name='user-area'),
)
```

you can also easily pass parameters to the *host_url()* template tag:

```
{% load hosts %}

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

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

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

4.1.4 Changing the scheme individually

It's not only possible to define the scheme in the hostconf but also on a case-by-case basis using the template tag:

```
{% load hosts %}

<a href="{% host_url 'user-dashboard' host 'user-area' username='johndoe' scheme
↪'https' %}">John's dashboard</a>
<a href="//help.example.com/faq/">FAQ</a>
```

Which will be rendered (with a *PARENT_HOST* of `'example.com'` and a *HOST_SCHEME* setting defaulting to `'//'`) as:

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

4.1.5 Storing the url in a context variable

New in version 0.4.

If you'd like to retrieve a URL without displaying it, you can save the result of the template tag in a template variable and use it later on, e.g.:

```
{% load hosts %}

{% host_url 'homepage' as homepage_url %}
<a href="{{ homepage_url }}" title="Go back to {{ homepage_url }}">Home</a>
```

4.2 Python helpers

4.2.1 `hosts` and `patterns`

When defining hostconfs you need to use the `patterns` and `host` helpers

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

The function to define the list of hosts (aka hostconfs), 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=''*, *scheme=None*, *port=None*)

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'),
    host(r'admin', 'urls.admin', name='admin', scheme='https://'),
)
```

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
- **`scheme`** (*`str`*) – the scheme to prepend host names with during reversing, e.g. when using the host_url() template tag. Defaults to *`HOST_SCHEME`*.
- **`port`** – the port to append to host names during reversing, e.g. when using the host_url() template tag. Defaults to *`HOST_PORT`*.

`add_prefix`(*prefix=''*)

Adds the prefix string to a string-based urlconf.

4.2.2 `reverse_host` and `reverse`

If you want to reverse the hostname or the full URL or a view including the scheme, hostname and port you'll need to use the `reverse` and `reverse_host` helper functions (or its lazy cousins).

`django_hosts.resolvers.`**`reverse`**(*viewname*, *args=None*, *kwargs=None*, *prefix=None*, *current_app=None*, *host=None*, *host_args=None*, *host_kwargs=None*, *scheme=None*, *port=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.resolvers import reverse
>>> reverse('about')
'//www.example.com/about/'
>>> reverse('about', host='www')
'//www.example.com/about/'
>>> reverse('repo', args=('jezdez',), host='www', scheme='git', port=1337)
'git://jezdez.example.com:1337/repo/'
```

You can set the used port and scheme in the host object or override with the paramater named accordingly.

The host name can be left empty to automatically fall back to the default hostname as defined in the *DEFAULT_HOST* setting.

Parameters

- **viewname** – the name of the view
- **args** – the arguments of the view
- **kwargs** – the keyed arguments of the view
- **prefix** – the prefix of the view urlconf
- **current_app** – the current_app argument
- **scheme** – the scheme to use
- **port** – the port to use
- **host** – the name of the host
- **host_args** – the host arguments
- **host_kwargs** – the host keyed arguments

Raises **django.core.urlresolvers.NoReverseMatch** – if no host or path matches

Return type the fully qualified URL with path

`django_hosts.resolvers.`**`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.resolvers import reverse_host
>>> reverse_host('with_username', args=('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

django_hosts.resolvers.**reverse_host_lazy**(**args*, ***kw*)
The lazy version of the *reverse_host()* function to be used in class based views and other module level situations

django_hosts.resolvers.**reverse_lazy**(**args*, ***kw*)
The lazy version of the *reverse()* function to be used in class based views and other module level situations

4.2.3 HostSiteManager model manager

class django_hosts.managers.**HostSiteManager**(*field_name=None*, *select_related=True*)
A model manager to limit objects to those associated with a site.

Parameters

- **field_name** – the name of the related field pointing at the Site model, or a series of relations using the field1__field2__field3 notation. Falls back to looking for 'site' and 'sites' fields.
- **select_related** – a boolean specifying whether to use select_related() when querying the database

Define a manager instance in your model class with one of the following notations:

```
on_site = HostSiteManager()  # automatically looks for site and sites
on_site = HostSiteManager("author__site")
on_site = HostSiteManager("author__blog__site")
on_site = HostSiteManager("author__blog__site",
                          select_related=False)
```

Then query against it with one of the manager methods:

```
def home_page(request):
    posts = BlogPost.on_site.by_request(request).all()
    return render(request, 'home_page.html', {'posts': posts})
```

by_id(*site_id=None*)
Returns a queryset matching the given site id. If not given this falls back to the SITE_ID setting.

Parameters **site_id** – the ID of the site

Return type QuerySet

by_request(*request*)
Returns a queryset matching the given request's site attribute.

Parameters **request** (HttpRequest) – the current request

Return type QuerySet

by_site(*site*)
Returns a queryset matching the given site.

Parameters **site** (Site) – a site instance

Return type QuerySet

4.3 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.

4.3.1 Included callbacks

`django-hosts` includes the following callbacks for use with the Django contrib app `django.contrib.sites`:

`django_hosts.callbacks.`**`host_site`**(*request*, **args*, ***kwargs*)

A callback function which uses the `django.contrib.sites` contrib app included in Django to match a host to a `Site` instance, setting a `request.site` attribute on success.

Parameters

- **`request`** – the request object passed from the middleware
- **`*args`** – the parameters as matched by the host patterns
- **`**kwargs`** – the keyed parameters as matched by the host patterns

It's important to note that this uses *`reverse_host()`* behind the scenes to reverse the host with the given arguments and keyed arguments to enable a flexible configuration of what will be used to retrieve the `Site` instance – in the end the callback will use a `domain__iexact` lookup to get it.

For example, imagine a host conf with a username parameter:

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

settings.PARENT_HOST = 'example.com'

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

When requesting this website with the host `jezdez.example.com`, the callback will act as if you'd do:

```
request.site = Site.objects.get(domain__iexact='jezdez.example.com')
```

..since the result of calling *`reverse_host()`* with the username `'jezdez'` is `'jezdez.example.com'`.

Later, in your views, you can nicely refer to the current site as `request.site` for further site-specific functionality.

`django_hosts.callbacks.`**`cached_host_site`**(*request*, **args*, ***kwargs*)

A callback function similar to *`host_site()`* which caches the resulting `Site` instance in the default cache backend for the time specfified as *`HOST_SITE_TIMEOUT`*.

Parameters

- **`request`** – the request object passed from the middleware
- **`*args`** – the parameters as matched by the host patterns
- **`**kwargs`** – the keyed parameters as matched by the host patterns

4.4 FAQ

4.4.1 Does django-hosts work with the Django Debug Toolbar?

Yes, django-hosts works with Django Debug toolbar with the only limitation that the toolbar's middleware has to be come *after* django-hosts' `HostsRequestMiddleware` middleware, e.g.:

```
MIDDLEWARE = (
    'django_hosts.middleware.HostsRequestMiddleware',
    # your other middlewares..
    'debug_toolbar.middleware.DebugToolbarMiddleware',
    'django_hosts.middleware.HostsResponseMiddleware',
)
```

Also, you have to install django-debug-toolbar 0.9.X or higher.

4.5 Changelog

4.5.1 3.0 (2017-11-20)

- **BACKWARD-INCOMPATIBLE** Dropped support for Django < 1.11.
- Confirmed support for Django 1.11 and Python 3.6 (no code changes were required). Added support for Django 2.0.

4.5.2 2.0 (2016-07-25)

- **BACKWARD-INCOMPATIBLE** Removed the `HostsMiddleware`, deprecated in django-hosts 1.0.
- **BACKWARD-INCOMPATIBLE** Removed the `on` argument of the `{% host_url %}` template tag, deprecated in django-hosts 1.0.
- Added support for Django 1.10.

4.5.3 1.4 (2016-01-21)

- **BACKWARD-INCOMPATIBLE** Dropped support for Python 3.2.
- Removed the last leftover from Django 1.5 support code.
- Clarified license in favor of Jazzband members.

4.5.4 1.3 (2015-12-15)

- **BACKWARD-INCOMPATIBLE** Dropped support for Django 1.7 as it doesn't receive security releases anymore.
- Added support for Django 1.9.
- Moved repo to https://github.com/jazzband/django-hosts
- Moved tests to https://travis-ci.org/jazzband/django-hosts
- Start to use setuptools_scm for easier versioning.

4.5.5 1.2 (2015-05-06)

- **BACKWARD-INCOMPATIBLE** Dropped support for Django 1.6 as it doesn't receive any security releases anymore.
- **BACKWARD-INCOMPATIBLE** Removed deprecated `django_hosts.reverse` module as it incorrectly shadowed the *`django_hosts.resolvers.reverse()`* function that was added in version 1.0. This is a earlier deprecation than planned, apologies for the inconvenience.
- Added support for Django 1.8.

4.5.6 1.1 (2015-01-05)

- Improved handling of allowed hosts by returning the 400 response directly.

4.5.7 1.0 (2014-12-29)

- **BACKWARD-INCOMPATIBLE** Moved `django_hosts.reverse.reverse_full` to *`django_hosts.resolvers.reverse()`* and `django_hosts.reverse.reverse_host` to *`django_hosts.resolvers.reverse_host()`*. This is a cleanup process to easier map Django's features and normalize the call signatures. The old functions are now pending deprecation and will be removed in the 1.2 release.
- **BACKWARD-INCOMPATIBLE** Dropped support for Django 1.5 as it doesn't receive any security releases anymore and 1.4 since its very soon going to lose it's LTS status.
- Moved repo to https://github.com/jazzband/django-hosts
- Extended testing setup to Python 3.4 and Django 1.7.
- Optionally allow setting the port per host and using the *`HOST_PORT`* setting.
- Refactored *`host_url()`* template tag to closer follow Django's own url template tag. This includes:
  - the renaming of the `on` argument to `host` (`on` will be removed in the 1.2 release)
  - the use of the Django>1.5 url template tag syntax that requires the view name (and the host name) to be quoted unless it's meant to be a template context variable

    Old:

    ```
    {% host_url homepage on www %}
    ```

    New:

    ```
    {% host_url 'homepage' host 'www' %}
    ```

  - the ability to automatically fallback to the host as defined in the *`DEFAULT_HOST`* setting when no `host` name is passed
  - a new optional `scheme` parameter to override the resulting URL's scheme individually
  - a new optional `port` parameter to override the resulting URL's port individually
  - a new ability to override Django's built-in url template tag by setting the `HOST_OVERRIDE_URL_TAG` setting to `True`
- Added *`reverse_lazy()`* and *`reverse_host_lazy()`* for use in import time situations such as class based views.

- Split the `django_hosts.middleware.HostsMiddleware` middleware into two piece to enable the use of the `request.host` parameter in other middlewares. See the installation instruction for the new setup.
- Rely on a few more built-ins in Django instead of writing them ourselves.
- Moved the test suite to use the py.test runner instead of Django's own test runner.
- Updated the *FAQ* to explain how to use Django's full page caching middleware with Django<1.7 and fixed the entry about the compatibility to the Debug Toolbar.
- Extended the tests to be close to 100% test coverage.
- Added tox configuration for easy local tests.
- Added a few Django 1.7 system checks (for the `ROOT_HOSTCONF` and `DEFAULT_HOST` settings).

4.5.8 0.6 (2013-06-17)

- Support for Django 1.5.x and Python > 3.2.
- Dropped support for Python 2.5 and Django 1.3.
- Optionally allow setting the scheme per host instead of only using the *`HOST_SCHEME`* setting.

4.5.9 0.5 (2012-08-29)

- Fixed host reversing when the `PARENT_HOST` equals
- Added *`HOST_SCHEME`* setting to be able to override the default URL scheme when reversing hosts.

4.5.10 0.4.2 (2012-02-14)

- Removed a unneeded installation time requirement for Django <= 1.4.
- Removed the use of versiontools due to unwanted installation time side effects.
- Refactored tests slightly.

4.5.11 0.4.1 (2011-12-23)

- Added *`cached_host_site()`* callback which stores the matching `Site` instance in the default cache backend (also see new *`HOST_SITE_TIMEOUT`* setting).
- Throw warning if django-debug-toolbar is used together with the `django_hosts` and the order of the `MIDDLEWARE_CLASSES` setting isn't correct.
- Added CI server at https://ci.enn.io/job/django-hosts/

4.5.12 0.4 (2011-11-04)

- Added ability to *save the result* of *`host_url()`* template tag in a template context variable.

4.5.13 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 https://django-hosts.readthedocs.io
- Stopped preventing the name 'default' for hosts.

4.5.14 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.

4.5.15 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.

4.5.16 0.1.1 (2011-05-30)

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

4.5.17 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.

Python Module Index

d

Index