django-hitcount Documentation

Release 1.2

Damon Timm

October 26, 2015

Contents

Django-Hitcount allows you to track the number of hits/views for a particular object.

CHAPTER 1

Overview

Django-Hitcount allows you to track the number of hits (views) for a particular object. This isn't meant to be a full-fledged tracking application or a real analytic tool; it's just a basic hit counter.

How one tracks a "hit" or "view" of a web page is not such a simple thing as it might seem. That's why folks rely on Google Analytics or similiar tools. It's tough! This is a simple app with some settings and features that should suit the basic needs of smaller sites.

It comes ready to track hits with a `HitCountDetailView` and a `HitCountJSONView` (to use the out-of-the-box JavaScript method, you will need jQuery – although writing your own JavaScript implementation is not hard).

1.1 Requirements and Compatibility

The 1.2.x series currently supports Django >= 1.7.x and corresponding versions of Python also supported by Django (including Python 3). Development of django-hitcount follows Django's Supported Version Policy and testing for older versions of Django/Python will be removed as time marches on.

Note: If you are running a Django 1.4, 1.5, or 1.6 stick with the django-hitcount v1.1.1. If you are running a Django version pre-1.4 you can try django-hitcount v0.2 (good luck!).

CHAPTER 2

Installation and Usage

Install django-hitcount:

```
pip install django-hitcount
```

Add django-hitcount to your `INSTALLED_APPS`:

```
# settings.py
INSTALLED_APPS = (
    ...
    'hitcount'
)
```

View the additional settings section for a list of the django-hitcount settings that are available.

For a working implementation, you can view the example project on Github.

2.1 Counting Hits

The main business-logic for evaluating and counting a *Hit* is done in `hitcount.views.HitCountMixin.hit_count()`. You can use this class method directly in your own Views or you can use one of the Views packaged with this app.

- *HitCountJSONView*: a JavaScript implementation which moves the business-logic to an Ajax View and hopefully speeds up page load times and eliminates some bot-traffic
- *HitCountDetailView*: which provides a wrapper from Django's generic `DetailView` and allows you to process the Hit as the view is loaded

2.1.1 HitCountMixin

This mixin can be used in your own class-based views or you can call the `hit_count()` class method directly. The method takes two arguments, a `HttpRequest` and `HitCount` object it will return a namedtuple: `UpdateHitCountResponse(hit_counted=Boolean, hit_message='Message'). hit_counted` will be `True` if the hit was counted and `False` if it was not. `hit_message` will indicate by what means the Hit was either counted or ignored.

It works like this.

```
from hitcount.models import HitCount
from hitcount.views import HitCountMixin
```

```
# first get the related HitCount object for your model object
hit_count = HitCount.objects.get_for_object(your_model_object)

# next, you can attempt to count a hit and get the response
# you need to pass it the request object as well
hit_count_response = HitCountMinxin. hit_count(request, hit_count)

# your response could look like this:
# UpdateHitCountResponse(hit_counted=True, hit_message='Hit counted: session key')
# UpdateHitCountResponse(hit_counted=False, hit_message='Not counted: session key has active hit')
```

To see this in action see the views.py code.

2.1.2 HitCountJSONView

The `hitcount.views.HitCountJSONView` can be used to handle an AJAX POST request. Django-hitcount comes with a bundled jQuery plugin for speeding up the `$.post` process by handling the retrieval of the CSRF token for you.

If you wish to use the `HitCountJSONView` in your project you first need to update your `urls.py` file to include the following:

```
# urls.py
urlpatterns = [
    ...
    url(r'hitcount/', include('hitcount.urls', namespace='hitcount')),
]
```

Next, you will need to add the JavaScript Ajax request to your template. To do this, use the `{% get_hit_count_js_variables for post as [var_name] %}` template tag to get the `ajax_url` and `hitcount_pk` for your object. The `hitcount_pk` is needed for POST-ing to the `HitCountJSONView`.

Here is an example of how all this might work together with the bundled jQuery plugin. It is taken from the example project and the jQuery can be modified to suit your needs. In the example below it simply updates the template with the `HitCountJSONView` response after the Ajax call is complete.

```
{% load staticfiles %}
<script src="{% static 'hitcount/jquery.postcsrf.js' %}"></script>

{% load hitcount_tags %}
{% get_hit_count_js_variables for post as hitcount %}
<script type="text/javascript">
jQuery(document).ready(function($) {
  // use the template tags in our JavaScript call
  $.postCSRF("{{ hitcount.ajax_url }}", { hitcountPK : "{{ hitcount.pk }}" })
    .done(function(data){
      $('<i />').text(data.hit_counted).attr('id','hit-counted-value').appendTo('#hit-counted');
      $('#hit-response').text(data.hit_message);
  }).fail(function(data){
      console.log('POST failed');
      console.log(data);
  });
});
</script>
```

2.1.3 HitCountDetailView

The `HitCountDetailView` can be used to do the business-logic of counting the hits by setting `count_hit=True`. See the views section for more information about what else is added to the template context with this view.

Here is an example implementation from the example project:

```
from hitcount.views import HitCountDetailView

class PostCountHitDetailView(HitCountDetailView):
    modal = Post        # your model goes here
    count_hit = True    # set to True if you want it to try and count the hit
```

Note: Unlike the JavaScript implementation (above), this View will do all the HitCount processing *before* the content is delivered to the user; if you have a large dataset of Hits or exclusions, this could slow down page load times. It will also be triggered by web crawlers and other bots that may not have otherwise executed the JavaScript.

2.2 Displaying Hits

There are different methods for *displaying* hits:

- *Template Tags*: provide a robust way to get related counts
- Views: allows you to wrap a class-based view and inject additional context into your template
- *Models*: can have a generic relation to their respective `HitCount`

2.2.1 Template Tags

For a more granular approach to viewing the hits for a related object you can use the `get_hit_count` template tag.

```
# remember to load the tags first
{% load hitcount_tags %}

# Return total hits for an object:
{% get_hit_count for [object] %}

# Get total hits for an object as a specified variable:
{% get_hit_count for [object] as [var] %}

# Get total hits for an object over a certain time period:
{% get_hit_count for [object] within ["days=1,minutes=30"] %}

# Get total hits for an object over a certain time period as a variable:
{% get_hit_count for [object] within ["days=1,minutes=30"] as [var] %}
```

2.2.2 Views

The `hitcount.views.HitCountDetailView` extends Django's generic `DetailView` and injects an additional context variable `hitcount`.

```
{# the primary key for the hitcount object #}
{{ hitcount.pk }}

{# the total hits for the object #}
{{ hitcount.total_hits }}
```

If you have set `count_hit=True` (see: *HitCountDetailView*) two additional variables will be set.

```
{# whether or not the hit for this request was counted (true/false) #}
{{ hitcount.hit_counted }}

{# the message form the UpdateHitCountResponse #}
{{ hitcount.hit_message }}
```

2.2.3 Models

Note: You are not *required* to do anything specific with your models; django-hitcount relies on a `GenericForeignKey` to create the relationship to your model's `HitCount`.

If you would like to add a reverse lookup in your own model to its related `HitCount` you can utilize the `hitcount.models.HitCountMixin`.

```
from django.db import models

from hitcount.models import HitCountMixin

# here is an example model with a GenericRelation
class MyModel(models.Model, HitCountMixin):
    pass

# you would access your hit_count like so:
my_model = MyModel.objects.get(pk=1)
my_model.hit_count.hits                  # total number of hits
my_model.hit_count.hits_in_last(days=7) # number of hits in last seven days
```

CHAPTER 3

Additional Settings

There are a few additional settings you can use to customize django-hitcount and are set in your `settings.py` file.

3.1 HITCOUNT_KEEP_HIT_ACTIVE

This is the number of days, weeks, months, hours, etc (using a `timedelta` keyword argument), that an `Hit` is kept **active**. If a `Hit` is **active** a repeat viewing will not be counted. After the **active** period ends, however, a new `Hit` will be recorded. You can decide how long you want this period to last and it is probably a matter of preference.:

```
# default value
HITCOUNT_KEEP_HIT_ACTIVE = { 'days': 7 }
```

3.2 HITCOUNT_HITS_PER_IP_LIMIT

Limit the number of **active** `Hits` from a single IP address. 0 means that it is unlimited.:

```
# default value
HITCOUNT_HITS_PER_IP_LIMIT = 0
```

3.3 HITCOUNT_EXCLUDE_USER_GROUP

Exclude `Hits` from all users in the specified user groups. By default, this is set to an empty list (all users counted). In the example, below, it will exclude all your 'Editors'.:

```
# example value, default is empty tuple
HITCOUNT_EXCLUDE_USER_GROUP = ( 'Editor', )
```

3.4 HITCOUNT_KEEP_HIT_IN_DATABASE

This setting is used with the `hitcount_cleanup` management command and specifies a `timedelta` within which to keep/save `Hits`. Any `Hit` older than the time specified will be removed for the `Hits` table.:

```
# default value
HITCOUNT_KEEP_HIT_IN_DATABASE = { 'days': 30 }
```

CHAPTER 4

Management Commands

If you would like to periodically prune your stale `Hits` you can do so by running the the management command `hitcount_cleanup`.:

```
./manage.py hitcount_cleanup
```

The command relies on the setting `HITCOUNT_KEEP_HIT_IN_DATABASE` to determine how far back to prune. See the additional settings section for more information.

CHAPTER 5

Example Project

There is an example project the demonstrates the functionality of this app. It's fairly easy to get this working using the Django development server. Be sure to run this inside your own `virtualenv` (but who doesn't, these days?!).

```
$ git clone git@github.com:thornomad/django-hitcount.git
$ cd django-hitcount/example_project
$ pip install -r requirements.txt   # sqlite requires pytz
$ python manage.py migrate          # will load some data fixtures for you
$ python manage.py createsuperuser  # for access to the admin portion
$ python manage.py runserver        # should be all set!
```

When you are ready to work on your own site, check out the Installation and Usage and Additional Settings sections.

CHAPTER 6

Contribution and Testing

I would love to make it better. Please fork, branch, and push.

Please make new features/improvements against the develop branch. If you are patching a bug or providing a fix of some sort that can be made against the master branch. For larger features, please create your own feature branch first before you make the pull request.

Note: You can safely ignore the `devel` branch which is old and stale but has something in it I can't remember why I'm saving it. Call me a hoarder.

6.1 Testing

You can run the tests by installing the requirements and then executing `runtests.py`:

```
$ pip install -r tests/requirements.txt
$ ./runtests.py     # against your currently installed version of Django
$ tox               # against the entire array of Django/Python versions
```

This method using `py.test` for test discovery and will also run *flake8* for code formatting. If you would like to use Django's own test runner you can execute:

```
$ ./runtests.py --django
```

CHAPTER 7

Additional Authors and Thanks

I've had some help and I'm very grateful! You can always look at the contributors on GitHub to get a clearer picture. This doesn't include everyone and if I missed someone let me know I will add it.

Thanks goes to:

- Basil Shubin and his work at django-hitcount-headless as well as his Russian translations
- ariddell for putting the *setup.py* package together for me

CHAPTER 8

Changelog

8.1 Version 1.2:

- added `hitcount.models.HitCountMixin` to provide a reverse lookup property to a model's `HitCount`
- deprecated `hitcount.views_update_hit_count()` and moved the business logic into `hitcount.views.HitCountMixin.hit_count()`
- deprecated `hitcount.views.update_hit_count_ajax()` and replaced with class-based view `hitcount.views.HitCountJSONView`
- deprecated `static/hitcount-jquery.js` and replaced with `static/jquery.postcsrf.js` (a more generic way to handle the Ajax POST CSRF fun-party)
- updated Django and Python version testing/support (>=1.7 as of Oct 2015)
- updated example_project to use new views and jQuery plugin
- updated tests to rely on the example_project

8.2 Version 1.1.1:

- fixed `session_key` returning `None` #40 (>=1.8.4)
- removed requirement for *SESSION_SAVE_EVERY_REQUEST*
- removed *patterns* for urls.py (>=1.9)
- updated management command, using `BaseCommand` instead of `NoArgsCommand` (>=1.9)
- added `TEMPLATES` to *conftest.py*

8.3 Version 1.1.0:

- added tests (lots of them)
- added documentation
- support for Django 1.4.x - 1.8.x
- support for Python 3.x
- created an example project

- squashed bugs
- released to pip
- more, I'm sure!

Note: if you are upgrading from version 0.2 (it's so old!) the `HitCount.object_pk` was changed from a `CharField` to a `PositiveIntegerField`. You will have to manually fix this in your database after upgrading.

CHAPTER 9

Issues

Use the GitHub issue tracker for django-hitcount to submit bugs, issues, and feature requests.