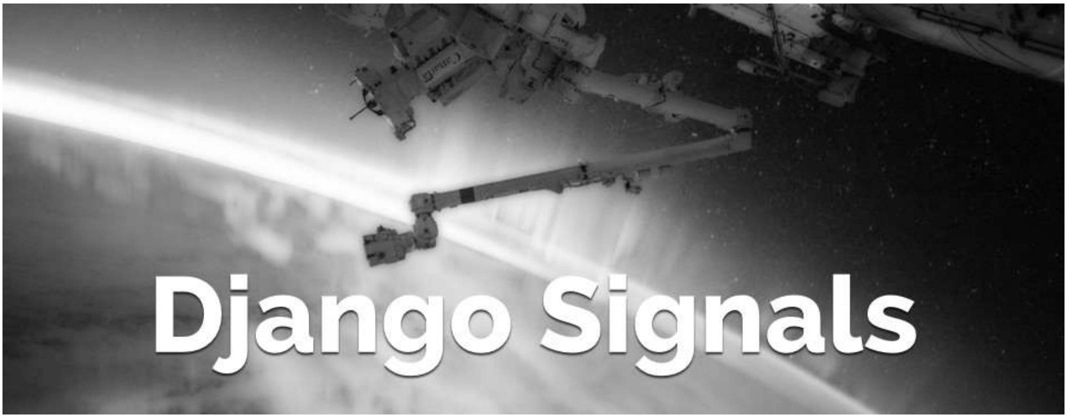
By **Vitor Freitas**

I'm a passionate software developer and researcher from Brazil, currently living in Finland. I write about Python, Django and Web Development on a weekly basis. Read more.

TUTORIAL

How to Create Django Signals

Jul 28, 2016 7 minutes read 55 comments 107,190 views



(Picture: https://unsplash.com/photos/NuE8Nu3otjo)

The Django Signals is a strategy to allow decoupled applications to get notified when certain events occur. Let's say you want to invalidate a cached page everytime a given model instance is updated, but there are several places in your code base that this model can be updated. You can do that using **signals**, hooking some pieces of code to be executed everytime this specific model's save method is trigged.

Another common use case is when you have extended the Custom Django User by using the Profile strategy through a one-to-one relationship. What we usually do is use a "signal dispatcher" to listen for the User's **post_save** event to also update the Profile instance as well. I've covered this case in another post, which you can read here: How to Extend Django User Model.

In this tutorial I will present you the built-in signals and give you some general advices about the best practices.

When Should I Use It?

Before we move forward, know *when you should use it*:

- When many pieces of code may be interested in the same events;
- When you need to interact with a decoupled application, e.g.:
  - A Django core model;
  - A model defined by a third-party app.

How it works?

If you are familiar with the **Observer Design Pattern**, this is somewhat how Django implements it. Or at least serves for the same purpose.

There are two key elements in the signals machinary: the *senders* and the *receivers*. As the name suggests, the *sender* is the one responsible to dispatch a signal, and the *receiver* is the one who will receive this signal and then do something.

A *receiver* must be a function or an instance method which is to receive signals.

A *sender* must either be a Python object, or None to receive events from any sender.

The connection between the *senders* and the *receivers* is done through “signal dispatchers”, which are instances of **Signal**, via the **connect** method.

The Django core also defines a **ModelSignal**, which is a subclass of **Signal** that allows the sender to be lazily specified as a string of the `app_label.ModelName` form. But, generally speaking, you will always want to use the **Signal** class to create custom signals.

So to receive a signal, you need to register a *receiver* function that gets called when the signal is sent by using the **Signal.connect()** method.

Usage

Let’s have a look on the `post_save` built-in signal. Its code lives in the `django.db.models.signals` module. This particular signal fires right after a model finish executing its **save** method.

```
from django.contrib.auth.models import User
from django.db.models.signals import post_save

def save_profile(sender, instance, **kwargs):
    instance.profile.save()

post_save.connect(save_profile, sender=User)
```

In the example above, `save_profile` is our *receiver* function, `User` is the *sender* and `post_save` is the *signal*. You can read it as: Everytime when a `User` instance finalize the execution of its `save` method, the `save_profile` function will be executed.

If you supress the **sender** argument like this: `post_save.connect(save_profile)` , the `save_profile` function will be executed after any Django model executes the **save** method.

Another way to register a signal, is by using the `@receiver` decorator:

```
def receiver(signal, **kwargs)
```

The **signal** parameter can be either a **Signal** instance or a list/tuple of **Signal** instances.

See an example below:

```
from django.contrib.auth.models import User
from django.db.models.signals import post_save
from django.dispatch import receiver

@receiver(post_save, sender=User)
def save_profile(sender, instance, **kwargs):
    instance.profile.save()
```

If you want to register the receiver function to several signals you may do it like this:

```
@receiver([post_save, post_delete], sender=User)
```

Where Should the Code Live?

Depending on where you register your application's signals, there might happen some side-effects because of importing code. So, it is a good idea to avoid putting it inside the **models** module or in application root module.

The Django documentation advices to put the signals inside your app config file. See below what I usually do, considering a Django app named **profiles**:

profiles/signals.py:

```
from django.contrib.auth.models import User
from django.db.models.signals import post_save
from django.dispatch import receiver

from cmdbox.profiles.models import Profile

@receiver(post_save, sender=User)
def create_user_profile(sender, instance, created, **kwargs):
    if created:
        Profile.objects.create(user=instance)

@receiver(post_save, sender=User)
def save_user_profile(sender, instance, **kwargs):
    instance.profile.save()
```

profiles/app.py:

```
from django.apps import AppConfig
from django.utils.translation import ugettext_lazy as _

class ProfilesConfig(AppConfig):
    name = 'cmdbox.profiles'
    verbose_name = _('profiles')

    def ready(self):
        import cmdbox.profiles.signals  # noqa
```

profiles/__init__.py:

```
default_app_config = 'cmdbox.profiles.apps.ProfilesConfig'
```

In the example above, just importing the **signals** module inside the **ready()** method will work because I'm using the `@receiver()` decorator. If you are connecting the **receiver function** using the **connect()** method, refer to the example below:

profiles/signals.py:

```
from cmdbox.profiles.models import Profile

def create_user_profile(sender, instance, created, **kwargs):
    if created:
        Profile.objects.create(user=instance)

def save_user_profile(sender, instance, **kwargs):
    instance.profile.save()
```

profiles/app.py:

```
from django.apps import AppConfig
from django.contrib.auth.models import User
from django.db.models.signals import post_save
from django.utils.translation import ugettext_lazy as _

from cmdbox.profiles.signals import create_user_profile, save_user_profile

class ProfilesConfig(AppConfig):
    name = 'cmdbox.profiles'
    verbose_name = _('profiles')

    def ready(self):
        post_save.connect(create_user_profile, sender=User)
        post_save.connect(save_user_profile, sender=User)
```

profiles/__init__.py:

```
default_app_config = 'cmdbox.profiles.apps.ProfilesConfig'
```

Note: The **profiles/__init__.py** bits are not required if you are already referring to your AppConfig in the `INSTALLED_APPS` settings.

Django Built-in Signals

Here you can find a list of some useful built-in signals. It is not complete, but those are the most common.

Model Signals

django.db.models.signals.**pre_init**:

```
receiver_function(sender, *args, **kwargs)
```

django.db.models.signals.**post_init**:

```
receiver_function(sender, instance)
```

django.db.models.signals.**pre_save**:

```
receiver_function(sender, instance, raw, using, update_fields)
```

django.db.models.signals.**post_save**:

```
receiver_function(sender, instance, created, raw, using, update_fields)
```

django.db.models.signals.**pre_delete**:

```
receiver_function(sender, instance, using)
```

django.db.models.signals.**post_delete**:

```
receiver_function(sender, instance, using)
```

django.db.models.signals.**m2m_changed**:

```
receiver_function(sender, instance, action, reverse, model, pk_set, using
```

Request/Response Signals

django.core.signals.**request_started**:

```
receiver_function(sender, environ)
```

django.core.signals.**request_finished**:

```
receiver_function(sender, environ)
```

django.core.signals.**got_request_exception**:

```
receiver_function(sender, request)
```

If you want to see the whole list, click here to access the official docs.

Related Posts

django signals

Share this post

Name

atodorov • a year ago • edited

Hi Vitor,
do you have any advise for applications that provide custom signals to which the consumer of this app can connect an arbitrary handler (notify about new user registrations in this particular case). I'm wondering whether to tell my users to connect their handlers in local_settings.py or have them create another app inside their projects and connect the signals in their AppConfig.ready() methods ?

^^^ and have their local_settings.py append the newly created app to INSTALLED_APPS

17 ^ | ⌄ • Reply • Share ›

Umut Çağdaş Coşkun • 3 years ago

I found your blog about a week ago (I guess via ImportPython newsletter). Then I begin to follow in feedly, I love your articles. Please keep writing!

15 ^ | ⌄ • Reply • Share ›

Vitor Freitas Mod ➦ Umut Çağdaş Coşkun • 3 years ago

Thank you very much for your comment, Umut! :-)
It is good to know people are enjoying and perhaps learning something new as well!

8 ^ | ⌄ • Reply • Share ›

Murat Jumashev ➦ Vitor Freitas • a year ago • edited

Absolutely enjoying! Your examples are priceless.

Reply • Share ›

Python Fanboy ➦ Umut Çağdaş Coşkun • 3 years ago

yeah!

1 • Reply • Share ›

myFatherIsKing • a year ago

Thanks Vitor.

Something so basic yet missing from the docs. You're a lifesaver. (I'm using this for Django==2.0)

4 • Reply • Share ›

Lance Wade Moore • a year ago

As usual Vitor is consulting the official Django docs and using best practices. This is my "go to" blog for anything django, and in many cases just as good or better than the official documentation!

1 • Reply • Share ›

christian • a year ago

Hi Vitor,

Your blog is definitely my favourite for django related tutorials!

Quick question:
Is it possible to set up some sort of asynchronous notification system using signals and receivers? For example, notify the logged in users when a model is updated?

1 • Reply • Share ›

Alejandro Hurtado • 2 years ago

How to send session of User signals?

1 ^ | v • Reply • Share ›

Alejandro Hurtado ➦ Alejandro Hurtado • 2 years ago

I did it with Middleware

1 ^ | v • Reply • Share ›

Jean-Christophe Lavocat • 3 years ago

I had some side effects going on from signals placed in the wrong fie. Thanks for your explanation Vitor.

1 ^ | v • Reply • Share ›

Vitor Freitas Mod ➦ Jean-Christophe Lavocat • 3 years ago

This happened to me once :-) And that was how I found the right place to put the signals
Thanks Jean!

^ | v • Reply • Share ›

Rishikesh Agrawani • a month ago • edited

Vitor, thank you very much for the guide.

^ | v • Reply • Share ›

Sekure Grean • 3 months ago

What is the purpose of { if created } block inside the create_user_profile function

^ | v • Reply • Share ›

Martin Quinta • 3 months ago

if your post_save save another model, i recommend use "if created and not raw" for prevent fixtures problems.

^ | ∨ • Reply • Share ›

Bapan Biswas • 3 months ago

creating profile using signal is good but how can I populate the other attributes of the profile in the same call

^ | ∨ • Reply • Share ›

Taukir Alam • 5 months ago

signal is asynchronous or synchronous ?

^ | ∨ • Reply • Share ›

gamesbook • 6 months ago

Is it possible to return messages created in the signal back to the user (in the normal admin interface)?

^ | ∨ • Reply • Share ›

Tushar Gupta • 8 months ago

Really amazing guides posted by you sir. Helps a lot. Please keep writing and I would appreciate if you could show some direction on the use of customised products in e commerce, as in how to add them and link them to cart.

^ | ∨ • Reply • Share ›

Gonzalo Amadio • 9 months ago

I am trying to use this, but instead the normal user profile. I have some models, and depending on the type of user, I create one or other type of associated user.

Basicalli I have entities/models.py , where I have Person and Company classes, and the signals.py is something like:

```
```

from django.db.models.signals import post_save
from django.dispatch import receiver
from django.conf import settings
from . import models as entities_models

@receiver(post_save, sender=settings.AUTH_USER_MODEL)
def user_create_profile(sender, instance

see more

Reply • Share ›

Marcelo Ribeiro da Silva • a year ago

Hello Victor,

The signals from my application are being listened to only within model.py. When I try to organize in separate files the signal does not return the functions of the signals.py file and nothing happens, not even error messages. I'm using Django 2.1.

Here's how my files are.

app.py
=====

from django.apps import AppConfig

class myAppConfig (AppConfig):
name = 'myapp'

def ready (self):

see more

Reply • Share ›

Divya • a year ago

what about when update and call signal???

Reply • Share ›

^ | ⌄ • Reply • Share ›

Mohammad Aqib • a year ago

Hi
If a certain condition exists then the signal not to save. How I can do that?

^ | ⌄ • Reply • Share ›

Carolina Londoño Agudelo • a year ago

Hi Victor,

in the profiles/app.py of the first example, the name of the class "class ProfilesConfig(AppConfig)" makes reference to the app name or to the Profiles Model?

Thank you for the post!

^ | ⌄ • Reply • Share ›

Gaurav Yadav • a year ago

Hi can we use sender (user) instance to check on type of users if the user type has been defined in an custom user model earlier. (Suppose type1 and type2)

@receiver(post_save, sender=settings.AUTH_USER_MODEL)
def create_and_save_profile(sender, instance, **kwargs):
If created:
If instance.user_type == 'type1':
Profile.user.create(user = instance)
instance.profile.save()
Elif instance.user_type == 'type2':
Profile2.user.create(user = instance)
instance.profile2.save()

^ | ⌄ • Reply • Share ›

郭东北 • a year ago

郭宗北 • a year ago

I could imagine there might be some beginners like myself, who can understand how to create Signals and use decorator receiver(), but just do not know the correct way to refer app_config into INSTALLED_APPS.

Assuming that you already got a app named 'websiteapp', and the apps.py asf:

class MyConfig(AppConfig):
name = 'websiteapp'

def ready(self):
import websiteapp.signals

Your default INSTALLED_APP should be asf:

INSTALLED_APPS = [
'django.contrib.admin',
[...]
'websiteapp',
]

The 'websiteapp' should be override as 'websiteapp.apps.MyConfig'. Then your signal will get working, no matter whether using decorator or not.

Thank you Vitor!

• Reply • Share ›

Kaushal Kumar • 2 years ago • edited

Thanks Vitor.
I have created UserConfig exactly like the given tutorial and added #default_app_config = 'users.app.UserConfig' to __init__.py. Signal

and receiver are working fine but now no warning, error or log is visible on console. And no request is also visible on console like

```
[08/Jan/2018 18:10:29] "GET
/users/verify/rp/8e07cd2d96bd4340a3d8687
HTTP/1.1" 301 0
```

^ | v • Reply • Share ›

Kaushal Kumar → Kaushal Kumar • 2 years ago

Solved the issue.

^ | v • Reply • Share ›

Ashfaque Chowdhury • 2 years ago

Hi! How do u pass a request object to the signal? let's say I store the logged-in user's database in a django session variable and I wanna use a model signal and wanna save the model for that database connection only using 'using(request["session])' on model's save method.How do I best go about it?

Thanks in advance.

^ | v • Reply • Share ›

Hugo Pineda • 2 years ago

It's hard to believe that the django docs are missing such an important step about the app config and init.py part. Thank you for this post!

^ | v • Reply • Share ›

Yul Gurrea • 2 years ago

Sorry i have to rephrase my question. What if i have two or more foreignkeys in a signal? How do i create these instances?

^ | v • Reply • Share ›

Yul Gurrea • 2 years ago

What if i have two senders and with different models each? how do i create instances of it?

Reply • Share ›

Yuriy Rabeshko • 2 years ago

Thank you Vitor!! I try to connect signal whole evening without any effect. Simply one line of code was separating me from success, I forgot to add the `default_app_config` to `__init__.py`)
Thanks a lot again !!

Reply • Share ›

Subin Shrestha • 2 years ago

i have a problem when i try to create profile from admin site.
from profile add user option i have created new user and when i try to save profile this gives me "Profile with this User already exists." errors as shown in screen shot. so what should i do to solve this problem.
Thanks!!!

Reply • Share ›

Luca • 2 years ago

Thanks for the tips!

I know this is a year old tutorial, but just thought you could update it to follow Django's documentation suggestion: https://docs.djangoproject....

It says:
"default_app_config allows applications that predate Django 1.7 such as django.contrib.admin to opt-in to AppConfig features without requiring users to update their INSTALLED_APPS.

New applications should avoid default_app_config. Instead they should require the dotted path to the appropriate AppConfig subclass to be configured explicitly in INSTALLED_APPS."

Cheers :)

Reply • Share ›

sohamnavadiya • 2 years ago

Very informative and nice explanation. Keep it up and share more tutorial like this.

Reply • Share ›

Kest lv • 2 years ago

Definitely one of the best Django related blogs! Thank you for your input, Vitor

Reply • Share ›

mtphmtph • 2 years ago

great tut,

about the "where shoulld the code live"
which example is more efficient ?

thank you for sharing.

^ | ˅ • Reply • Share ›

Rafael Capucho • 2 years ago

Hi Vitor, I noticed an advice in the Django Docs that says:

"Your ready() method will run during startup of every management command. For example, even though the test database configuration is separate from the production settings, manage.py test would still execute some queries against your production database!" - source: https://docs.djangoproject....

Is it safe to start importing every signal in ready()? thanks

^ | ˅ • Reply • Share ›

Vitor Freitas Mod → Rafael Capucho • 2 years ago

Hi Rafael!

It's safe to connect the receivers

Subscribe to our Mailing List

Receive updates from the Blog!

Your email address

SUBSCRIBE

Popular Posts

How to Extend Django User Model

How to Setup a SSL Certificate on Nginx for a Django Application

How to Deploy a Django Application to Digital Ocean