django-hooks Documentation

Release 0.2.0

Esteban Castro Borsani

November 29, 2015

Contents

A modular plugin system for django apps.

There are 3 kinds of hooks:

- TemplateHook: Third-party apps will be able to insert their own code (text/html) into an app template.
- FormHook: Third-party apps will be able to insert Forms in an app view.
- ViewHook: This is deprecated in favor of FormHook
- SignalHook: Connect or emit a signal by its name/id. This is the same as Django signals except that they don't need to be pre-defined.

This documentation is divided into different parts. I recommend that you get started with the Installation and then head over to the Usage.

CHAPTER 1

User's Guide

1.1 Installation

1.1.1 Pip

Latest django-hooks can be installed via pip:

```
$ pip install django-hooks
```

1.1.2 Django Project

To load the templatetags add `hooks` into `settings.INSTALLED_APPS`.

Example:

```
# settings.py

INSTALLED_APPS = [
    # ...

    'hooks'
]
```

1.2 Hooks

1.2.1 TemplateHook

Adding a hook-point in `main_app`'s template:

```
# my_main_app/templates/_base.html

{% load hooks_tags %}

<!DOCTYPE html>
<html>
  <head>
    #...
```

```
    {% hook 'within_head' %}

    #...
  </head>
</html>
```

Tip: Here we are adding a *hook-point* called `within_head` where *third-party* apps will be able to insert their code.

Creating a hook listener in a `third_party_app`:

```
# third_party_app/template_hooks.py

from django.template.loader import render_to_string
from django.utils.html import mark_safe, format_html


# Example 1
def css_resources(context, *args, **kwargs):
    return mark_safe(u'<link rel="stylesheet" href="%s/app_hook/styles.css">' % settings.STATIC_URL)


# Example 2
def user_about_info(context, *args, **kwargs):
    user = context['request'].user
    return format_html(
        "<b>{name}</b> {last_name}: {about}",
        name=user.first_name,
        last_name=user.last_name,
        about=mark_safe(user.profile.about_html_field)  # Some safe (sanitized) html data.
    )


# Example 3
def a_more_complex_hook(context, *args, **kwargs):
    # If you are doing this a lot, make sure to keep your templates in memory (google: django.templat
    return render_to_string(
        template_name='templates/app_hook/head_resources.html',
        context_instance=context
    )


# Example 4
def an_even_more_complex_hook(context, *args, **kwargs):
    articles = Article.objects.all()
    return render_to_string(
        template_name='templates/app_hook/my_articles.html',
        dictionary={'articles': articles, },
        context_instance=context
    )
```

Registering a hook listener in a `third_party_app`:

```
# third_party_app/apps.py

from django.apps import AppConfig


class MyAppConfig(AppConfig):
```

```
    name = 'myapp'
    verbose_name = 'My App'

    def ready(self):
        from hooks.templatehook import hook
        from third_party_app.template_hooks import css_resources

        hook.register("within_head", css_resources)
```

Tip: Where to register your hooks:

Use `AppConfig.ready()`: docs and example

1.2.2 FormHook

Creating a hook-point:

```
# main_app/formhooks.py

from hooks.formhook import Hook

MyFormHook = Hook()
UserFormHook = Hook(providing_args=['user'])
```

Adding a hook-point to the main app view:

```
# main_app/views.py

from main_app import formhooks


# Example 1
def my_view(request):
    if request.method == 'POST':
        form_hook = formhooks.MyFormHook(data=request.POST)

        if form_hook.is_valid():
            form_hook.save()
            redirect('/')
    else:
        form_hook = formhooks.MyFormHook()

    return response('my_view.html', {'form_hook': form_hook})


# Example 2
def user_profile_update(request):
    if request.method == 'POST':
        user_form = UserForm(data=request.POST, instance=request.user)

        # Hook listeners will receive the user and populate the
        # initial data (or instance if a ModelForm is used) accordingly,
        # or maybe even query the data base.
        user_form_hook = formhooks.UserFormHook(user=request.user, data=request.POST)

        if all([user_form.is_valid(), user_form_hook.is_valid()]):  # Avoid short-circuit
```

```
            new_user = user_form.save()
            user_form_hook.save(new_user=new_user)  # They may receive extra parameter when saving
            redirect('/')
    else:
        user_form = MyForm(instance=request.user)
        user_form_hook = formhooks.UserFormHook(user=request.user)

    return response('user_profile_update.html', {'user_form': user_form, 'user_form_hook': user_form_
```

Displaying the forms:

```
# main_app/templates/my_view.html

{% extends "main_app/_base.html" %}

{% block title %}My forms{% endblock %}

{% block content %}
    <h1 class="headline">My forms</h1>

    <form action="." method="post">
        {% csrf_token %}

        {% for f in form_hook %}
            {{ f }}
        {% endfor %}

        <input type="submit" value="Save" />
    </form>
{% endblock %}
```

Creating a hook-listener in a third-party app:

Tip: Hooks listeners are just regular django forms or model forms

```
# third_party_app/forms.py

from django import forms
from third_party_app.models import MyUserExtension


# Example 1
class MyRegularForm(forms.Form):
    """"""
    # ...


# Example 2
class MyUserExtensionForm(forms.ModelForm):

    class Meta:
        model = MyUserExtension
        fields = ("gender", "age", "about")

    def __init__(user=None, *args, **kwargs):
        try:
            instance = MyUserExtension.objects.get(user=user)
        except MyUserExtension.DoesNotExist:
```

```
            instance = None

        kwargs['instance'] = instance
        super(MyUserExtensionForm, self).__init__(*args, **kwargs)

    def save(new_user, *args, **kwargs):
        self.instance.user = new_user
        super(MyUserExtensionForm, self).save(*args, **kwargs)
```

Registering a hook-listener:

```
# third_party_app/apps.py

from django.apps import AppConfig


# Example
class MyAppConfig(AppConfig):

    name = 'myapp'
    verbose_name = 'My App'

    def ready(self):
        from main_app.formhooks import MyFormHook, UserFormHook
        from third_party_app.forms import MyRegularForm, MyUserExtensionForm

        MyFormHook.register(MyRegularForm)
        UserFormHook.register(MyUserExtensionForm)
```

1.2.3 SignalHook

Tip: Best practices:

- Always *document* the signals the app will send, include the parameters the receiver should handle.
- Send signals from views, only.
- Avoid sending signals from plugins.
- Try to avoid signal-hell in general. It's better to be *explicit* and call the functions that would've handle the signal otherwise. Of course, this won't be possible when there are plugins involved.

Connecting a hook-listener:

```
# third_party_app/urls.py

from third_party_app.viewhooks import myhandler
from hooks import signalhook

signalhook.hook.connect("my-signal", myhandler)
```

Sending a signal:

```
# send from anywhere, app-hook, main-app... view, model, form...

from hooks import signalhook
```

```
responses = signalhook.hook.send("my-signal", arg_one="hello", arg_two="world")
responses = signalhook.hook.send("another-signal")
```

Tip: `SignalHook` uses django signals under the hood, so you can do pretty much the same things.

CHAPTER 2

API Reference

If you are looking for information on a specific function, class or method, this part of the documentation is for you.

2.1 API

2.1.1 hooks.formhook Module

HookFactory Object

class `hooks.formhook.`**`HookFactory`**(*instances*)

Hook factory, provide some short-cuts to make a sequence of forms behave as a single form. This is used by *Hook*

Parameters **`instances`** (*list*) – Sequence of callables, usually `django.forms.Form` or `django.forms.ModelForm`

`__iter__`()

Forms iterator

Yield Form instance

`is_valid`()

Validate all the forms

Returns The result of validating all the forms

Return type bool

`save`(**args*, ***kwargs*)

Save all the forms

Parameters

- **`*args`** – Positional arguments passed to the forms
- **`**kwargs`** – Keyword arguments passed to the forms

Returns Sequence of returned values by all the forms as tuples of (instance, result)

Return type list

Hook Object

class hooks.formhook.**Hook**(*providing_args=None*)
Container of forms

Parameters **providing_args** (*list*) – A list of the arguments this hook can pass along in a __call__()

__call__(**args*, ***kwargs*)
Call all registered forms

Parameters

- **prefix** (*str*) – Prefix for the forms to avoid clashing of fields, it must be of the form text_%d. Defaults to hook%d
- ***args** – Positional arguments passed to the forms
- ****kwargs** – Keyword arguments passed to the forms

Returns Factory to handle all the forms as they were one

Return type *HookFactory*

register(*form*)
Register form

Parameters **form** (*callable*) – The form, usually django.forms.Form or django.forms.ModelForm

unregister(*form*)
Remove form from registry

Parameters **form** (*callable*) – The previously registered form

2.1.2 hooks.signalhook Module

Hook Object

class hooks.signalhook.**Hook**
A dynamic-signal dispatcher. Should be used through *hook*

thread-safety: it's not thread safe, this may change in the future if a RLock is added around _registry operations. In the meanwhile, you should register/connect/disconnect at import time (global scope) to ensure thread-safety, doing it in the AppConfig.ready() method is safe

connect(*name*, *func*, *sender=None*, *dispatch_uid=None*)
Connects a function to a hook. Creates the hook (name) if it does not exists

Parameters

- **name** (*str*) – The hook name
- **func** (*callable*) – A function reference used as a callback
- **sender** (*class*) – Optional sender __class__ to which the func should respond. Default will match all
- **dispatch_uid** (*str*) – Optional unique id, see django.dispatch.Signal for more info

disconnect(*name*, *func*, *dispatch_uid=None*)
Disconnects a function from a hook

Parameters

- **name** (*str*) – The hook name
- **func** (*callable*) – A function reference registered previously
- **dispatch_uid** (*str*) – optional unique id, see `django.dispatch.Signal` for more info.

register(*name*)
Register a new hook. Not required (see `connect()` method)

Parameters **name** (*str*) – The hook name

Returns Django signal

Return type `django.dispatch.Signal`

send(*name*, *sender=None*, ***kwargs*)
Sends the signal. Return every function response that was hooked to hook-name as a list: [(func, response),]

Parameters

- **name** (*str*) – The hook name
- **sender** (*class*) – Optional sender __class__ to which registered callback should match (see `connect()` method)

Returns Signal responses as a sequence of tuples (func, response)

Return type list

hook Singleton

`hooks.signalhook.`**hook = <hooks.signalhook.Hook object>**
A dynamic-signal dispatcher. Should be used through *hook*

thread-safety: it's not thread safe, this may change in the future if a RLock is added around _registry operations. In the meanwhile, you should register/connect/disconnect at import time (global scope) to ensure thread-safety, doing it in the AppConfig.ready() method is safe

2.1.3 hooks.templatehook Module

TemplateHook Object

class `hooks.templatehook.`**TemplateHook**(*providing_args=None*)
A hook for templates. This can be used directly or through the *Hook* dispatcher

Parameters **providing_args** (*list*) – A list of the arguments this hook can pass along in a `__call__()`

register(*func*)
Register a new callback

Parameters **func** (*callable*) – A function reference used as a callback

unregister(*func*)
Remove a previously registered callback

Parameters **func** (*callable*) – A function reference that was registered previously

unregister_all()
Remove all callbacks

Hook Object

class hooks.templatehook.**Hook**
Dynamic dispatcher (proxy) for *TemplateHook*

register(*name*, *func*)
Register a new callback. When the name/id is not found a new hook is created under its name, meaning the hook is usually created by the first registered callback

Parameters

- **name** (*str*) – Hook name
- **func** (*callable*) – A func reference (callback)

unregister(*name*, *func*)
Remove a previously registered callback

Parameters

- **name** (*str*) – Hook name
- **func** (*callable*) – A function reference that was registered previously

unregister_all(*name*)
Remove all callbacks

Parameters **name** (*str*) – Hook name

2.1.4 hooks.templatetags.hooks_tags Module

Template tags

hooks.templatetags.hooks_tags.**hook_tag**(*context*, *name*, **args*, ***kwargs*)
Hook tag to call within templates

Parameters

- **context** (*dict*) – This is automatically passed, contains the template state/variables
- **name** (*str*) – The hook which will be dispatched
- ***args** – Positional arguments, will be passed to hook callbacks
- ****kwargs** – Keyword arguments, will be passed to hook callbacks

Returns A concatenation of all callbacks responses marked as safe (conditionally)

Return type str

Helpers

hooks.templatetags.hooks_tags.**template_hook_collect**(*module*, *hook_name*, **args*, ***kwargs*)
Helper to include in your own templatetag, for static TemplateHooks

Example:

```
    import myhooks
    from hooks.templatetags import template_hook_collect

    @register.simple_tag(takes_context=True)
    def hook(context, name, *args, **kwargs):
        return template_hook_collect(myhooks, name, context, *args, **kwargs)
```

Parameters

- **module** (*module*) – Module containing the template hook definitions
- **hook_name** (*str*) – The hook name to be dispatched
- ***args** – Positional arguments, will be passed to hook callbacks
- ****kwargs** – Keyword arguments, will be passed to hook callbacks

Returns A concatenation of all callbacks responses marked as safe (conditionally)

Return type str

CHAPTER 3

Additional Notes

Design notes, legal information and changelog are here for the interested.

3.1 Changelog

3.1.1 0.2.0

- New: Proper docs
- Improvement: *FormHook.save()* returns a list of (form, result) instead of just the results

3.1.2 0.1.4

- Drops Django 1.7 support (mostly because it does not supports Python 3.5)
- Adds Python 3.5 support
- New: FormHook
- Deprecated: VieHook in favor of FormHook

3.1.3 0.1.3

- Fixes missing templatetags auto-escaping
- Drops support for Django 1.4, 1.5 and 1.6

3.1.4 0.1.2

- Fixes missing *hooks.templatetags* in setup.py

3.1.5 0.1.1

- Fixes missing viewhook.context

3.1.6 0.1.0

- Initial release

3.2 License

h

Symbols

C

D

H

I

R

S

T

U