django-load Documentation

Release 1.0.0

Jonas Obrist

January 07, 2017

Contents

Contents:

CHAPTER 1

Python API

1.1 django_load.core

django_load.core.**load**(*modname*, *verbose=False*, *failfast=False*)
Loads modules from all installed apps.

Parameters

- **modname** (*string*) – Name of the module (eg: `'plugins'`)
- **verbose** (*boolean*) – If true, the loading will be verbose. Useful for debugging.
- **failfast** (*boolean*) – If true, the loading will not surpress exceptions.

Return type None

django_load.core.**iterload**(*modname*, *verbose=False*, *failfast=False*)
Same as *load()* but returns an iterator of the loaded modules.

Parameters

- **modname** (*string*) – Name of the module (eg: `'plugins'`)
- **verbose** (*boolean*) – If true, the loading will be verbose. Useful for debugging.
- **failfast** (*boolean*) – If true, the loading will not surpress exceptions.

Return type iterator of modules

django_load.core.**load_object**(*import_path*)
Loads an object from an 'import_path', like in MIDDLEWARE_CLASSES and the likes.

Import paths should be: "mypackage.mymodule.MyObject". It then imports the module up until the last dot and tries to get the attribute after that dot from the imported module.

If the import path does not contain any dots, a TypeError is raised.

If the module cannot be imported, an ImportError is raised.

If the attribute does not exist in the module, a AttributeError is raised.

Parameters **import_path** (*string*) – The path to the object to load, eg `'mypackage.mymodule.MyObject'`

Return type object

django_load.core.**iterload_object**(*import_paths*)
Same as *load_object()* but for multiple import paths at once.

Parameters **import_paths** (*iterable of strings*) – An iterable containing import_paths for *load_object()*.

Return type iterator of objects

CHAPTER 2

Examples

2.1 load

Let's assume your app wants to load all `plugins.py` files from the installed apps, to allow those apps to extend your application. You can achieve this like this:

```
from django_load.core import load

load('plugins')
```

2.2 iterload

Now let's say you want to do the same, but actually do something with those modules, more specific, find all objects in those modules, that are subclasses of `BasePlugin` and call our `do_something` function with those objects:

```
from django_load.core import iterload

for module in iterload('plugins'):
    for name in dir(module):
        obj = getattr(module, name)
        if issubclass(obj, BasePlugin):
            do_something(obj)
```

2.3 iterload_objects

You could also have a setting called `MY_APP_PLUGINS` which contains import paths similar to `MIDDLEWARE_CLASSES`. You want to load those plugins and call the `do_something` function with them:

```
from django_load.core import iterload_objects
from django.conf import settings

for obj in iterload_objects(settings.MY_APP_PLUGINS):
    do_something(obj)
```

2.4 load_object

If you only want to load a single object, you can do that too. Let's say you want to load `MyObject` from the `mypackage.mymodule` module:

```
from django_load.core import load_object

obj = load_object('mypackage.mymodule.MyObject')
```

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

d

D

I

L