



無料電子ブック

学習

Django

Free unaffiliated eBook created from
Stack Overflow contributors.

#django

.....	1
1: Django	2
.....	2
.....	2
Examples	3
.....	3
Django	5
.....	6
.....	7
Python 3.3	7
Python 2	7
.....	7
virtualenvwrapper	8
pyenv + pyenv-virtualenv	8
.....	9
Hello World	9
Docker	10
.....	10
.....	11
.....	11
Nginx	12
.....	12
2: ArrayField - PostgreSQL	14
.....	14
.....	14
Examples	14
ArrayField	14
ArrayField	14
containsArrayField	14
ArrayField	15

contained_by.....	15
3: CookiecutterDjango.....	16
Examples.....	16
Cookiecutterdjango.....	16
4: Django Rest Framework.....	18
Examples.....	18
API.....	18
5: Django-CachingRedis.....	20
.....	20
Examples.....	20
django-redis-cache.....	20
django-redis.....	20
6: Django.....	22
.....	22
Examples.....	22
Django.....	22
7: Django.....	23
.....	23
Examples.....	24
python-social-auth.....	24
Django Allauth.....	27
8: DjangoCRUD.....	30
Examples.....	30
CRUD.....	30
9: django.....	35
.....	35
Examples.....	35
Django.....	35
10: F.....	36
.....	36
.....	36

Examples.....	36
.....	36
.....	36
.....	37
11: Formsets.....	38
.....	38
Examples.....	38
Formset.....	38
12: HStoreField - - PostgreSQL.....	40
.....	40
Examples.....	40
HStoreField.....	40
HStoreField.....	40
.....	40
.....	41
contains.....	41
13: JSONField - PostgreSQL.....	42
.....	42
.....	42
.....	42
Examples.....	42
JSONField.....	42
Django 1.9.....	42
JSONField.....	42
.....	43
.....	43
.....	43
JSONField.....	43
14: RangeFields - PostgreSQL.....	44
.....	44
Examples.....	44
.....	44

RangeField.....	44
.....	44
contains.....	44
contained_by.....	45
.....	45
None.....	45
.....	45
15: URL.....	46
Examples.....	46
Django.....	46
URLDjango 1.9.....	48
16:	50
Examples.....	50
Querysets `as_manager`.....	50
select_related.....	50
.....	51
17:	53
.....	53
Examples.....	53
.....	53
Q.....	54
ManyToManyFieldn + 1.....	54
.....	54
.....	55
ForeignKeyn + 1.....	56
.....	56
.....	57
DjangoSQL.....	57
QuerySet.....	58
F.....	58
18:	60

.....	60
Examples.....	60
.....	60
views.py.....	60
urls.py.....	60
.....	60
views.py.....	60
book.html.....	61
.....	61
app / models.py.....	61
app / views.py.....	61
app / templates / app / pokemon_list.html.....	62
app / templates / app / pokemon_detail.html.....	62
app / urls.py.....	62
.....	63
app / views.py.....	63
app / templates / app / pokemon_form.html.....	63
app / templates / app / pokemon_confirm_delete.html.....	64
app / models.py.....	64
.....	64
DjangoCreateView.....	65
1.....	66
19:	67
.....	67
Examples.....	67
.DEBUG.....	67
.....	67
.....	69
20:	70
Examples.....	70

Jenkins 2.0+ Pipeline Script.....	70
Jenkins 2.0+ Pipeline ScriptDocker Containers.....	70
21:	72
Examples.....	72
CBVdjango-filter.....	72
22:	73
Examples.....	73
.....	73
.....	73
.....	74
+MQ.....	75
23:	77
Examples.....	77
XSS.....	77
.....	78
CSRF.....	79
24:	81
Examples.....	81
.....	81
.....	81
.....	81
25:	83
Examples.....	83
MySQL / MariaDB.....	83
PostgreSQL.....	84
sqlite.....	85
.....	85
.....	86
26:	88
Examples.....	88
.....	88
.....	89

27:	90
.....	90
Examples.....	90
PythonPdb.....	90
Django.....	91
"assert False".....	93
.....	93
28:	94
Examples.....	94
.....	94
.....	95
.....	95
.....	96
.....	97
{extend}{include}{blocks}.....	97
.....	97
.....	98
29:	100
Examples.....	100
.....	100
.....	100
Node.....	101
30:	104
.....	104
Examples.....	104
[]Hello World Equivalent.....	104
31:	105
Examples.....	105
ModelForm.....	105
Django.....	105
views.pymodelForm.....	105
Django.....	107

.....	108
32:	111
Examples.....	111
.....	111
.....	111
33:	113
Examples.....	113
>>/.....	113
django.....	114
34:	115
.....	115
.....	115
Examples.....	115
.....	115
IP.....	116
.....	117
Django 1.10.....	117
35:	119
.....	119
Examples.....	119
.....	119
36:	120
.....	120
Examples.....	120
.....	120
.....	120
.....	121
Django DB.....	123
.....	124
.....	125
.....	125
URL.....	125

.....	125
.....	126
Meta.....	126
.....	126
.....	126
.....	127
UUID.....	129
.....	129
37:	131
.....	131
.....	132
Examples.....	132
.....	132
BinaryField.....	135
CharField.....	135
DateTimeField.....	135
.....	135
38:	137
.....	137
Examples.....	137
Queryset.....	137
.....	137
GROUP BY ... COUNT / SUM Django ORM.....	138
39:	140
Examples.....	140
.....	140
username `email` username`.....	143
Django.....	145
.....	147
.....	148
40:	150
Examples.....	150

-	150
Django	151
Django	152
.....	154
.....	155
41:	157
Examples	157
Syslog	157
Django	158
42:	160
.....	160
.....	160
Examples	160
.....	160
.....	161
.....	162
43:	163
.....	163
.....	163
Examples	164
.....	164
/.....	164
pre_save.....	165
.....	165
44:	167
.....	167
Examples	167
.....	167
.....	167
settings.py.....	167
.....	167
.....	168

.....	168
.....	169
.....	170
Noop.....	171
.....	172
.....	172
.....	172
45:	174
Examples.....	174
.....	174
.....	175
ManyToMany.....	175
46:	176
.....	176
Examples.....	176
.....	176
.....	176
47:	178
.....	178
Examples.....	178
.....	178
.....	179
.....	180
.....	181
.....	181
.....	181
.....	182
CharField.....	182
48:	184
Examples.....	184
.....	184
CSSJS.....	185

.....	186
views.py	187
urls.py	187
forms.py	187
admin.py	188
49:	189
.....	189
.....	189
Examples	189
.....	189
.....	190
manage.pydjango-admin	191
.....	191
50:	193
Examples	193
.....	193
.....	193
BASE_DIR	193
.....	194
settings.py	194
.....	195
1	195
2	196
.....	196
.....	196
JSON	197
DATABASE_URL	198
51:	199
Examples	199
.....	199
52:	200

Examples.....	200
GunicornDjango.....	200
Heroku.....	200
fabfile.py.....	201
Heroku Django.....	202
Django Nginx + Gunicorn + Supervisor on LinuxUbuntu.....	202
NGINX.....	203
GUNICORN.....	204
.....	204
apache / nginx.....	205
53:	206
.....	206
Examples.....	206
2.....	206
.....	208

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: [django](#)

It is an unofficial and free Django ebook created for educational purposes. All the content is extracted from [Stack Overflow Documentation](#), which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official Django.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

1: Djangoをいめる

Djangoは、「デッドラインをつのためのWebフレームワーク」としてしています。「Djangoにより、よりかつよりないコードでよりいWebアプリケーションをすることがになります。それはMVCアーキテクチャとることができます。そのコアにはのものがあります

- とテストのためのでスタンドアロンのWebサーバー
- HTMLフォームとデータベースのにしたをできるフォームのシリアライゼーションとシステム
- オブジェクトプログラミングからされたのをするテンプレートシステム
- リクエストのさまざまなでしてカスタムをできるミドルウェアクラスをサポートする、いくつかのキャッシュメソッドのいずれかをできるキャッシングフレームワーク
- アプリケーションのコンポーネントがされたをしてにイベントをすることをするディスパッチャシステム
- Djangoのコンポーネントをさまざまなにするなどのシステム
- DjangoモデルインスタンスのXMLおよび/またはJSONをしてみむことができるシリアライゼーションシステム
- テンプレートエンジンのをするためのシステム
- Pythonのユニットテストフレームワークにみまれたインタフェース

バージョン

バージョン	
1.11	2017-04-04
1.10	2016-08-01
1.9	2015-12-01
1.8	2015-04-01
1.7	2014-09-02
1.6	2013-11-06
1.5	2013-02-26
1.4	2012-03-23
1.3	2011-03-23
1.2	2010-05-17
1.1	2009-07-29

バージョン	
1.0	2008-09-03

Examples

プロジェクトの

DjangoはPythonに基づくWebフレームワークです。 Djangoの**1.11**のリリースがインストールされているPython **2.7**、 **3.4**、 **3.5**または**3.6**がです。 `pip`がであるとする、インストールはのコマンドをするのとじくらいです。 にすバージョンをすると、 `django`のバージョンがインストールされることにしてください。

```
$ pip install django
```

のバージョンの`django`をインストールするには、バージョンが`django 1.10.5`であるとして、 のコマンドをします。

```
$ pip install django==1.10.5
```

DjangoをしてされたWebアプリケーションは、 Djangoプロジェクトにするがあります。 `django-admin` コマンドをして、 のディレクトリでしいプロジェクトをすることができます

```
$ django-admin startproject myproject
```

ここで、 `myproject` はプロジェクトをにするで、 、 、 およびアンダースコアでできます。

これにより、 のプロジェクトがされます。

```
myproject/  
  manage.py  
  myproject/  
    __init__.py  
    settings.py  
    urls.py  
    wsgi.py
```

アプリケーションをするには、 サーバーをします

```
$ cd myproject  
$ python manage.py runserver
```

サーバーがしているので、 Webブラウザで `http://127.0.0.1:8000/` にアクセスしてください。 のページがされます。

It worked!

Congratulations on your first Django-powered page.

Of course, you haven't actually done any work yet. Next, start your first app by running `python manage.py startapp [app_label]`.

You're seeing this message because you have `DEBUG = True` in your Django settings file and you haven't configured any URLs. Get to work!

デフォルトでは、`runserver` コマンドは、ポート 8000 IP でサーバーをします。このサーバーは、コードをするとにします。しかし、しいファイルをするは、でサーバーをするがあります。

サーバーのポートをするは、コマンドラインとしてします。

```
$ python manage.py runserver 8080
```

サーバーのIPをするは、ポートとにそれをします。

```
$ python manage.py runserver 0.0.0.0:8000
```

`runserver` はデバッグビルドとローカルテストのためのものです。なサーバープログラム *Apache* などとは、にプロダクションですがあります。

Django アプリケーションをする

Django プロジェクトには、の `apps` がまれてい `apps`。これは、プロジェクトをでなモジュールにするなです。アプリケーションをするには、あなたのプロジェクトフォルダ `manage.py` にし、`startapp` コマンドをしますにじて *myapp* をします。

```
python manage.py startapp myapp
```

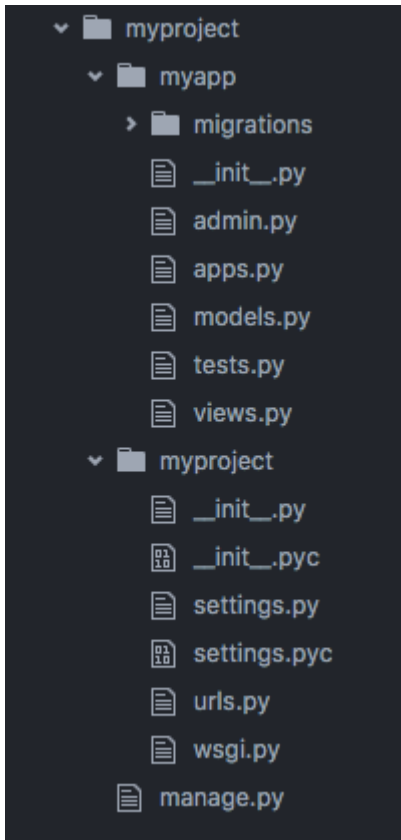
これにより、*myapp* フォルダといくつかのなファイル `models.py` と `views.py` などがされます。

Django に *myapp* をさせるために、`settings.py` それをしてください

```
# myproject/settings.py

# Application definition
INSTALLED_APPS = [
    ...
    'myapp',
]
```

Django プロジェクトのフォルダはみにわせてすることができます。フォルダーのりしをけるため、プロジェクトフォルダーのが `/src` にされることがあります。なフォルダはのようになります。



Djangoの

django-adminは、Djangoにしているコマンドラインツールです。Djangoプロジェクトを始めるためのなコマンドがいくつかしています。このコマンドは`./manage.py`とじですが、プロジェクトディレクトリにあるはありません。 `DJANGO_SETTINGS_MODULE` をするがあります。

Django プロジェクトはDjangoファイルをもPythonコードベースです。Djangoは、 `django-admin startproject NAME` コマンドをとってプロジェクトをすることができます。プロジェクトには、トップレベルに`manage.py`というファイルがあり、`urls.py`というルートURLファイルがあります。

`manage.py`は`django-admin`プロジェクトのバージョンで、そのプロジェクトでコマンドをすることができます。たとえば、プロジェクトをローカルでするには、 `python manage.py runserver` します。プロジェクトはDjangoアプリでされています。

Django アプリケーションは、モデルファイルデフォルトでは`models.py` と、アプリケーションのURLやビューなどのそののファイルをもPythonパッケージです。アプリケーションは、 `django-admin startapp NAME` コマンドでできますこのコマンドはプロジェクトディレクトリからするがあります。アプリをプロジェクトのにするには、 `settings.py` `INSTALLED_APPS` リストにそのアプリがまれているがあり `settings.py`。あなたがなをした、Djangoはのようなものうプリインストールされ、そのアプリケーションのいくつかのアプリケーションがしてを、あなたのために。アプリケーションはのDjangoプロジェクトでできます。

Django ORMは`models.py`でされたデータベースモデルをすべてし、それらのモデルクラスについてデータベーステーブルをします。これをうには、まず、 `settings.py` `DATABASES` をしてデータベースをセットアップし `settings.py`。に、 **データベースモデル** をしたら、 `python manage.py makemigrations` し、いて `python manage.py migrate` をして、モデルについてデータベースのスキーマ

をまたはします。

なこんにちはの。

ステップ**1**すでにDjangoがインストールされているは、このをスキップできます。

```
pip install Django
```

ステップ**2**しいプロジェクトをします

```
django-admin startproject hello
```

これにより、`hello`というのフォルダがされ、のファイルがされます。

```
hello/
├─ hello/
│   ├── __init__.py
│   ├── settings.py
│   ├── urls.py
│   └─ wsgi.py
└─ manage.py
```

インサイドステップ**3** `hello` モジュールむフォルダ `__init__.py` ファイルをするにはば `views.py`

```
hello/
├─ hello/
│   ├── __init__.py
│   ├── settings.py
│   ├── urls.py
│   ├── views.py  <- here
│   └─ wsgi.py
└─ manage.py
```

のをしてください

```
from django.http import HttpResponse

def hello(request):
    return HttpResponse('Hello, World')
```

これはビューとされます。

ステップ**4** `hello/urls.py` をのようになります。

```
from django.conf.urls import url
from django.contrib import admin
from hello import views

urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url(r'^$', views.hello)
```

```
] 
```

ビュー `hello()` をURLにリンクします。

ステップ5サーバをします。

```
python manage.py runserver
```

ステップ6

ブラウザーで `http://localhost:8000/` をすると、がされます。

こんにちは

にはではありませんが、プロジェクトを「」ですることをくおめします。とは、のバージョンのPythonとのモジュールをし、オペレーティングシステムのネイティブPythonやのプロジェクトとしコンピュータでしないコンテナ ディレクトリです。

のプロジェクトごとになるをすることで、さまざまなバージョンのPythonでさまざまなDjangoプロジェクトをでき、することなくのをできます。

Python 3.3

Python 3.3には、の `venv` モジュールがまれています。、これは `pyvenv` としてびすことができます。`pyvenv` コマンドができないでは、 `python3 -m venv` としてモジュールをびすことでじにアクセスできます。

をするには

```
$ pyvenv <env-folder>
# Or, if pyvenv is not available
$ python3 -m venv <env-folder>
```

Python 2

Python 2をしているは、に `pip` からのモジュールとしてインストールできます。

```
$ pip install virtualenv
```

わりに `virtualenv` コマンドをしてをします。

```
$ virtualenv <env-folder>
```

のバージョン

がされました。それをするには、するでにすることができます。

のPythonバージョンを 'にする'には、

Linuxのような

```
$ source <env-folder>/bin/activate
```

Windowsのような

```
<env-folder>\Scripts\activate.bat
```

これにより、がアクティブであることをすプロンプトがされます。 (<env-folder>) \$

これ、 `pip` をしてインストールされたものは、システムではなく `env` フォルダにインストールされます。

のままにするには、 `deactivate`

```
(<env-folder>) $ deactivate
```

わりに **virtualenvwrapper** をう

virtualenvwrapper をって、`virtualenv` のとアクティベーションをににしたり、コードからしたりすることもえられます

```
# Create a virtualenv
mkvirtualenv my_virtualenv

# Activate a virtualenv
workon my_virtualenv

# Deactivate the current virtualenv
deactivate
```

わりに **pyenv + pyenv-virtualenv**

のPythonバージョンをうがあるでは、`virtualenv` と `pyenv-virtualenv` のをけることができます。

```
# Create a virtualenv for specific Python version
pyenv virtualenv 2.7.10 my-virtual-env-2.7.10
```

```
# Create a virtualenv for active python version
pyenv virtualenv venv34

# Activate, deactivate virtualenv
pyenv activate <name>
pyenv deactivate
```

virtualenvsをするは、[postactivateスクリプト](#)でPYTHONPATHとDJANGO_SETTINGS_MODULEをするとです。

```
#!/bin/sh
# This hook is sourced after this virtualenv is activated

# Set PYTHONPATH to isolate the virtualenv so that only modules installed
# in the virtualenv are available
export PYTHONPATH="/home/me/path/to/your/project_root:$VIRTUAL_ENV/lib/python3.4"

# Set DJANGO_SETTINGS_MODULE if you don't use the default `myproject.settings`
# or if you use `django-admin` rather than `manage.py`
export DJANGO_SETTINGS_MODULE="myproject.settings.dev"
```

プロジェクトパスをする

プロジェクトパスをベース<env-folder>あるな.projectファイルのにすることもしばしばにちます。これをうと、をアクティブにするたびにアクティブディレクトリがされたパスにされます。

<env-folder>/.projectというのしいファイルをしします。ファイルのは、プロジェクトディレクトリのパスにするがあります。

```
/path/to/project/directory
```

に、をしします source <env-folder>/bin/activateまたはworkon my_virtualenvをしします。は /path/to/project/directoryます。

ファイルの**Hello World**の

このは、DjangoでHello Worldページをするためののをしています。これは、django-admin startproject exampleコマンドがにフォルダとファイルのをし、プロジェクトをするためにずしもそのがというわけではないことにくでしょう。

1. file.pyというfile.pyファイルをしします。
2. そのファイルにのコードをコピーしてりけます。

```
import sys

from django.conf import settings

settings.configure(
```

```

DEBUG=True,
SECRET_KEY='thisisthesecretkey',
ROOT_URLCONF=__name__,
MIDDLEWARE_CLASSES=(
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
),
)

from django.conf.urls import url
from django.http import HttpResponse

# Your code goes below this line.

def index(request):
    return HttpResponse('Hello, World!')

urlpatterns = [
    url(r'^$', index),
]

# Your code goes above this line

if __name__ == "__main__":
    from django.core.management import execute_from_command_line

    execute_from_command_line(sys.argv)

```

3. ターミナルにき、このコマンドで `python file.py runserver` コマンドをします。
4. ブラウザをき、 127.0.0.18000 にみます。

Dockerをサポートするデプロイにしたプロジェクト。

デフォルトのDjangoプロジェクトテンプレートはうまくいきますが、コードをデプロイすると `devop` がプロジェクトにをくなどしてしまいます。あなたができることは、あなたのリポジトリにするがあるりのものとソースコードをけることです。

[GitHub](#) でな Django プロジェクトテンプレートをつけることができます。

プロジェクトの

```

PROJECT_ROOT
├── devel.dockerfile
├── docker-compose.yml
├── nginx
│   └── project_name.conf
├── README.md
├── setup.py
└── src
    ├── manage.py
    └── project_name
        └── __init__.py

```



```

└─ service
  ├── __init__.py
  ├── settings
  │   ├── common.py
  │   ├── development.py
  │   ├── __init__.py
  │   └── staging.py
  ├── urls.py
  └── wsgi.py

```

はしたい `service` のディレクトリ `service`、はじできるように、すべてのプロジェクトのおかげため Dockerfile すべてののプロジェクトで。との、すでにここににされています。

のファイルの
のをする

ドッカーファイル

だけが Docker をするというですべてのがしているわけではありません。これは、
`devel.dockerfile` `devel.dockerfile` であるがあります

```

FROM python:2.7
ENV PYTHONUNBUFFERED 1

RUN mkdir /run/service
ADD . /run/service
WORKDIR /run/service

RUN pip install -U pip
RUN pip install -I -e .[develop] --process-dependency-links

WORKDIR /run/service/src
ENTRYPOINT ["python", "manage.py"]
CMD ["runserver", "0.0.0.0:8000"]

```

のだけでビルドの Docker キャッシュがされます。のをするだけでみます。

する

Docker のは、にローカルでするのサービスがあるにです。 `docker-compose.yml`

```

version: '2'
services:
  web:
    build:
      context: .
      dockerfile: devel.dockerfile
    volumes:
      - "./src/{{ project_name }}:/run/service/src/{{ project_name }}"
      - "./media:/run/service/media"
    ports:

```

```
- "8000:8000"
depends_on:
  - db
db:
  image: mysql:5.6
  environment:
    - MYSQL_ROOT_PASSWORD=root
    - MYSQL_DATABASE={{ project_name }}
nginx:
  image: nginx
  ports:
    - "80:80"
  volumes:
    - "./nginx:/etc/nginx/conf.d"
    - "./media:/var/media"
  depends_on:
    - web
```

Nginx

あなたのはなりにいものでなければならぬので、からNginxをいいたいとっています。に、nginxのファイルのをします。

```
server {
    listen 80;
    client_max_body_size 4G;
    keepalive_timeout 5;

    location /media/ {
        autoindex on;
        alias /var/media/;
    }

    location / {
        proxy_pass_header Server;
        proxy_set_header Host $http_host;
        proxy_redirect off;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Scheme $scheme;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-SSL on;
        proxy_connect_timeout 600;
        proxy_read_timeout 600;
        proxy_pass http://web:8000/;
    }
}
```

```
$ cd PROJECT_ROOT
$ docker-compose build web # build the image - first-time and after requirements change
$ docker-compose up # to run the project
$ docker-compose run --rm --service-ports --no-deps # to run the project - and be able to use PDB
$ docker-compose run --rm --no-deps <management_command> # to use other than runserver commands, like makemigrations
```

```
$ docker exec -ti web bash # For accessing django container shell, using it you will be  
inside /run/service directory, where you can run ./manage shell, or other stuff  
$ docker-compose start # Starting docker containers  
$ docker-compose stop # Stopping docker containers
```

オンラインでDjangoをいめるをむ <https://riptutorial.com/ja/django/topic/200/django>をいめる

2: ArrayField - PostgreSQLのフィールド

- `from django.contrib.postgres.fields import ArrayField`
- クラス `ArrayField` base_field、size = None、** options
- `FooModel.objects.filter(array_field_name__contains = [オブジェクト、to、check])`
- `FooModel.objects.filter(array_field_name__contained_by = [オブジェクト、to、check])`

sizeパラメータはPostgreSQLにされますが、PostgreSQLはそれをしません。

ArrayFieldをするは、 [Postgresqlのドキュメント](#) からこのをえておいてください。

ヒントはセットではありません。のをすることは、データベースのったのとなります。となるアイテムごとにをつのテーブルをすることをしてください。これはがになり、のにしてよりスケーラビリティがします。

Examples

なArrayField

PostgreSQL ArrayFieldをするには、ArrayFieldに、のとしてフィールドとしてするデータをすがあります。のをするので、FloatFieldをしFloatField。

```
from django.db import models, FloatField
from django.contrib.postgres.fields import ArrayField

class Book(models.Model):
    ratings = ArrayField(FloatField())
```

ArrayFieldのサイズの

```
from django.db import models, IntegerField
from django.contrib.postgres.fields import ArrayField

class IceCream(models.Model):
    scoops = ArrayField(IntegerField() # we'll use numbers to ID the scoops
                        , size=6) # our parlor only lets you have 6 scoops
```

sizeパラメータをすると、それはpostgresqlにされ、それを付けてします。したがって、postgresqlコンソールをしてのscoopsフィールドに7つのをすることはです。

containsをむArrayFieldのメンバシップのクエリ

このクエリは、チョコレートスcoopとバニラスcoopをつすべてのコーンをしします。

```
VANILLA, CHOCOLATE, MINT, STRAWBERRY = 1, 2, 3, 4 # constants for flavors
choco_vanilla_cones = IceCream.objects.filter(scoops__contains=[CHOCOLATE, VANILLA])
```

models.py ファイルから IceCream モデルをインポートすることをしないでください。

また、django は ArrayField のインデックスをしないことに ArrayField。それらをするは、インデックスがになります。このインデックスは、ファイルで RunSQL をびすことででするがあります。

ArrayField のネスト

あなたはができ ArrayField すことによって、S を ArrayField それがだとして base_field。

```
from django.db import models, IntegerField
from django.contrib.postgres.fields import ArrayField

class SudokuBoard(models.Model):
    numbers = ArrayField(
        ArrayField(
            models.IntegerField(),
            size=9,
        ),
        size=9,
    )
```

contained_by をむリストのをむすべてのモデルをする

このクエリは、ミントスクープまたはバニラスクープのいずれかを つすべてのコーンをします。

```
minty_vanilla_cones = IceCream.objects.filter(scoops__contained_by=[MINT, VANILLA])
```

オンラインで ArrayField - PostgreSQL のフィールドをむ

<https://riptutorial.com/ja/django/topic/1693/arrayfield-----postgresqlのフィールド>

3: CookiecutterでDjangoをうには

Examples

Cookiecutterを使ったdjangoプロジェクトのインストールと

Cookiecutterをインストールするためのはのとおります。

- ピップ
- virtualenv
- PostgreSQL

あなたのプロジェクトのvirtualenvをし、それをにしてください

```
$ mkvirtualenv <virtualenv name>
$ workon <virtualenv name>
```

にCookiecutterをインストールする

```
$ pip install cookiecutter
```

ディレクトリを、プロジェクトをきたいフォルダにします。のコマンドをしてdjangoプロジェクトをします。

```
$ cookiecutter https://github.com/pydanny/cookiecutter-django.git
```

このコマンドは、cookiecutter-djangoリポジトリでcookiecutterをし、プロジェクトのをするようにします。のあとに[かっこ]でされているデフォルトをするには、もせずに「enter」をします。

```
project_name [project_name]: example_project
repo_name [example_project]:
author_name [Your Name]: Atul Mishra
email [Your email]: abc@gmail.com
description [A short description of the project.]: Demo Project
domain_name [example.com]: example.com
version [0.1.0]: 0.1.0
timezone [UTC]: UTC
now [2016/03/08]: 2016/03/08
year [2016]: 2016
use_whitenoise [y]: y
use_celery [n]: n
use_mailhog [n]: n
use_sentry [n]: n
use_newrelic [n]: n
use_opbeat [n]: n
windows [n]: n
use_python2 [n]: n
```

プロジェクトオプションについては、 [オフィシャルドキュメント](#)をしてください。プロジェクトがセットアップされました。

オンラインでCookiecutterでDjangoをうにはをむ

<https://riptutorial.com/ja/django/topic/5385/cookiecutter>でdjangoをうには-

4: Django Rest Framework

Examples

シンプルなベアボーンのみ API

のようなモデルがあるとすると、Django REST Framework「DRF」によってされるなベアボーンのみ APIをしてランニングをします。

models.py

```
class FeedItem(models.Model):
    title = models.CharField(max_length=100, blank=True)
    url = models.URLField(blank=True)
    style = models.CharField(max_length=100, blank=True)
    description = models.TextField(blank=True)
```

シリアライザは、Djangoモデルこのは`FeedItem` からすべてのをし、JSONにする`FeedItem`です。Djangoでフォームクラスをするのによくています。があれば、これはとてもです。

serializers.py

```
from rest_framework import serializers
from . import models

class FeedItemSerializer(serializers.ModelSerializer):
    class Meta:
        model = models.FeedItem
        fields = ('title', 'url', 'description', 'style')
```

views.py

DRFは、さまざまなユースケースをするための**くのビュークラス**をします。このでは、**のみ API**のみをするため、よりな**ビューセット**やするビューをするのではなく、DRFの`ListAPIView`サブクラスをします。

このクラスのは、データをシリアライザにリンクし、レスポンスオブジェクトにすべてまとめてラップすることです。

```
from rest_framework import generics
from . import serializers, models

class FeedItemList(generics.ListAPIView):
    serializer_class = serializers.FeedItemSerializer
    queryset = models.FeedItem.objects.all()
```

urls.py

ルートがDRFビューにいていることをしてください。

```
from django.conf.urls import url
from . import views

urlpatterns = [
    ...
    url(r'path/to/api', views.FeedItemList.as_view()),
]
```

オンラインでDjango Rest Frameworkをむ <https://riptutorial.com/ja/django/topic/7341/django-rest-framework>

5: Django-Caching バックエンドでの Redis の

django-redis-cache または django-redis をすることは、キャッシュされたすべてのアイテムをするためのソリューションです。Redis を `SESSION_ENGINE` としてセットアップすることは `SESSION_ENGINE`、なの1つは、キャッシュをしのように、のキャッシュを `SESSION_ENGINE` としてすること `SESSION_ENGINE`。これはにのドキュメントののトピックですが、そのがまれています。

に `settings.py` をしてください

```
SESSION_ENGINE = "django.contrib.sessions.backends.cache"
```

Examples

django-redis-cache をう

バックエンドキャッシュユーティリティとしてRedisをするがあるのは、[django-redis-cache](#) パッケージです。

このでは、すでに[Redisサーバー](#)がしていることをとしています。

```
$ pip install django-redis-cache
```

`CACHES` オブジェクトをインクルードするように `settings.py` をします [キャッシングにするDjangoのドキュメント](#) を。

```
CACHES = {
    'default': {
        'BACKEND': 'redis_cache.RedisCache',
        'LOCATION': 'localhost:6379',
        'OPTIONS': {
            'DB': 0,
        }
    }
}
```

django-redis をう

バックエンドキャッシュユーティリティとしてRedisをするがあるのは、[django-redis](#) パッケージです。

このでは、すでに[Redisサーバー](#)がしていることをとしています。

```
$ pip install django-redis
```

`CACHES` オブジェクトをインクルードするように `settings.py` をします [キャッシングにするDjangoの](#)

ドキュメントを。

```
CACHES = {
    'default': {
        'BACKEND': 'django_redis.cache.RedisCache',
        'LOCATION': 'redis://127.0.0.1:6379/1',
        'OPTIONS': {
            'CLIENT_CLASS': 'django_redis.client.DefaultClient',
        }
    }
}
```

オンラインでDjango-CachingバックエンドでのRedisのをむ

<https://riptutorial.com/ja/django/topic/4085/django-cachingバックエンドでのredisの>

6: Django コマンドラインから。

DjangoはWebアプリケーションですが、コマンドラインアプリやスクリプトにもできるでいやすいORMがあります。できる2つのなるアプローチがあります。はカスタムコマンドをし、もう1つはスクリプトのにDjangoをします。

Examples

Django コマンドラインから。

あなたがdjangoプロジェクトをセットアップし、ファイルがmainというのアプリにあるとすれば、これはあなたのコードをします

```
import os, sys

# Setup environ
sys.path.append(os.getcwd())
os.environ.setdefault("DJANGO_SETTINGS_MODULE", "main.settings")

# Setup django
import django
django.setup()

# rest of your imports go here

from main.models import MyModel

# normal python code that makes use of Django models go here

for obj in MyModel.objects.all():
    print obj
```

は、のように入ります。

```
python main/cli.py
```

オンラインでDjangoコマンドラインから。をむ <https://riptutorial.com/ja/django/topic/5848/django>
コマンドラインから-

7: Djangoとソーシャルネットワーク

パラメーター

	ありますか
いくつかの	Django-Allauthがほとんどのをとっているにするな。そのオプションについては、 をし てください。
ACCOUNT_AUTHENTICATION_METHOD= "username"または "email"または "username_email"	するログインをします。ユーザーがユーザー、メールアドレス、またはそのをしてログインするかどうかをします。これを「メール」にすると、 ACCOUNT_EMAIL_REQUIRED = True
ACCOUNT_EMAIL_CONFIRMATION_EXPIRE_DAYS = 3	メールメールのをします。
ACCOUNT_EMAIL_REQUIRED= False	サインアップには、ユーザーはメールアドレスをきすがあります。これは ACCOUNT_AUTHENTICATION_METHODとしてわれま す
ACCOUNT_EMAIL_VERIFICATION=「オプション」	サインアップにメールのをします。「」、「オプション」、または「なし」のいずれかをします。「」にすると、メールアドレスがされるまでユーザーはログインできなくなります。のメールアドレスでのログインをするには、「オプション」または「なし」をします。「オプション」のはメールメールはされますが、「なし」のはメールメールはされません。
ACCOUNT_LOGIN_ATTEMPTS_LIMIT= 5	ログインにした。このをえると、ユーザーはされた ACCOUNT_LOGIN_ATTEMPTS_TIMEOUT ログインできなくなります。これはallauth ログインビューをしますが、Djangoのログインがにされるのをしません。
ACCOUNT_LOGOUT_ON_PASSWORD_CHANGE= False	ユーザーがパスワードをまたはしたのにログアウトするかどうかをします。

	ありますか
SOCIALACCOUNT_PROVIDERS= dict	プロバイダのをむ。

Examples

なpython-social-auth

python-social-authは、ソーシャルとメカニズムをするフレームワークです。くのソーシャルバックエンド**Facebook**、Twitter、Github、LinkedInなど

インストール

まず、python-social-authパッケージをインストールするがあります。

```
pip install python-social-auth
```

githubからコードをダウンロードしてください。これをあなたのrequirements.txtファイルにするといでしょう。

CONFIGURING settings.py

settings.pyのadd

```
INSTALLED_APPS = (  
    ...  
    'social.apps.django_app.default',  
    ...  
)
```

バックエンドの

AUTHENTICATION_BACKENDSには、するバックエンドがまれており、なものだけをくがあります。

```
AUTHENTICATION_BACKENDS = (  
    'social.backends.open_id.OpenIdAuth',  
    'social.backends.google.GoogleOpenId',  
    'social.backends.google.GoogleOAuth2',  
    'social.backends.google.GoogleOAuth',  
    'social.backends.twitter.TwitterOAuth',  
    'social.backends.yahoo.YahooOpenId',  
    ...  
    'django.contrib.auth.backends.ModelBackend',  
)
```

あなたのプロジェクト settings.py はまだ AUTHENTICATION_BACKENDS フィールドをとっていないかもしれません。そのは、フィールドをします。ユーザ/パスワードによるログインをするので 'django.contrib.auth.backends.ModelBackend', をしてはいけません。

たとえば、FacebookやLinkedin Backendsをするは、APIキーをするがあります

```
SOCIAL_AUTH_FACEBOOK_KEY = 'YOURFACEBOOKKEY'
SOCIAL_AUTH_FACEBOOK_SECRET = 'YOURFACEBOOKSECRET'
```

そして

```
SOCIAL_AUTH_LINKEDIN_KEY = 'YOURLINKEDINKEY'
SOCIAL_AUTH_LINKEDIN_SECRET = 'YOURLINKEDINSECRET'
```

あなたはFacebookのとLinkedinののneddedキーをできます。 [ここでは](#)、APIキーとキーをするためのなリストとそれぞれのをることができます。

にするはにしておくがあります。 [ここで](#)つスタックオーバーフローの。 [このチュートリアル](#)は、のにちます。

TEMPLATE_CONTEXT_PROCESSORSはリダイレクト、バックエンドなどのにちますが、のもののみがです。

```
TEMPLATE_CONTEXT_PROCESSORS = (
    ...
    'social.apps.django_app.context_processors.backends',
    'social.apps.django_app.context_processors.login_redirect',
    ...
)
```

Django 1.8では、のようにTEMPLATE_CONTEXT_PREPROCESSORSをすることはされていませんでした。これがてはまるは、 TEMPLATESにします。あなたはこのようなかをなければなりません

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, "templates")],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
                'social.apps.django_app.context_processors.backends',
                'social.apps.django_app.context_processors.login_redirect',
            ],
        },
    ],
]
```

カスタムユーザの

カスタムユーザモデルをしていて、そのユーザとけたいは、のをしてくださいまだsettings.pyにあります

```
SOCIAL_AUTH_USER_MODEL = 'somepackage.models.CustomUser'
```

CustomUserは、デフォルトのUserからまたはするモデルです。

CONFIGURING urls.py

```
# if you haven't imported include make sure you do so at the top of your file
from django.conf.urls import url, include

urlpatterns = patterns('',
    ...
    url(' ', include('social.apps.django_app.urls', namespace='social'))
    ...
)
```

に、なモデルをするためにデータベースをするがあります。

```
./manage.py migrate
```

にたちはぶことができます

いくつかのテンプレートでは、のようなものをするがあります

```
<a href="{% url 'social:begin' 'facebook' %}?next={{ request.path }}">Login with
Facebook</a>
<a href="{% url 'social:begin' 'linkedin' %}?next={{ request.path }}">Login with
Linkedin</a>
```

のバックエンドをするは、バックエンドで 'facebook'をするだけです。

ユーザーのログアウト

ログインしたは、ログアウトするをしたいとかもしれません。ログインテンプレートがされたテンプレートのくに、のタグをします。

```
<a href="{% url 'logout' %}">Logout</a>
```

または

```
<a href="/logout">Logout</a>
```

urls.py ファイルをurls.pyようなコードでしたいとうでしょう

```
url(r'^logout/$', views.logout, name='logout'),
```

に、のようなコードでviews.pyファイルをします。

```
def logout(request):
    auth_logout(request)
```



```
return redirect('/')
```

Django Allauthをう

のすべてのプロジェクトでは、Django-Allauthはセットアップがなものでしたが、これにされるものではありません。

- 50のソーシャルネットワーク
- ローカルアカウントとソーシャルアカウントののサインアップ
- のソーシャルアカウント
- ソーシャルアカウントのオプション
- メールアドレスののメールアドレス、プライマリの
- パスワードれたれEメールアドレスフロー

あなたのをすことにがあれば、Django-Allauthは、システムのプロセスとをするためののをいます。

のでは、Django 1.10+

セットアップ

```
pip install django-allauth
```

settings.pyファイルで、のをいます。

```
# Specify the context processors as follows:
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                # Already defined Django-related contexts here

                # `allauth` needs this from django. It is there by default,
                # unless you've devilishly taken it away.
                'django.template.context_processors.request',
            ],
        },
    },
]

AUTHENTICATION_BACKENDS = (
    # Needed to login by username in Django admin, regardless of `allauth`
    'django.contrib.auth.backends.ModelBackend',

    # `allauth` specific authentication methods, such as login by e-mail
    'allauth.account.auth_backends.AuthenticationBackend',
)

INSTALLED_APPS = (
    # Up here is all your default installed apps from Django
```

```
# The following apps are required:
'django.contrib.auth',
'django.contrib.sites',

'allauth',
'allauth.account',
'allauth.socialaccount',

# include the providers you want to enable:
'allauth.socialaccount.providers.google',
'allauth.socialaccount.providers.facebook',
)

# Don't forget this little dude.
SITE_ID = 1
```

のsettings.pyファイルのをurls.py、urls.pyファイルにします。あなたのyourapp/urls.pyやProjectName/urls.pyすることができます。、はProjectName/urls.pyをむ。

```
urlpatterns = [
    # other urls here
    url(r'^accounts/', include('allauth.urls')),
    # other urls here
]
```

include('allauth.urls')するだけで、あなたにのURLをでします

```
^accounts/ ^ ^signup/$ [name='account_signup']
^accounts/ ^ ^login/$ [name='account_login']
^accounts/ ^ ^logout/$ [name='account_logout']
^accounts/ ^ ^password/change/$ [name='account_change_password']
^accounts/ ^ ^password/set/$ [name='account_set_password']
^accounts/ ^ ^inactive/$ [name='account_inactive']
^accounts/ ^ ^email/$ [name='account_email']
^accounts/ ^ ^confirm-email/$ [name='account_email_verification_sent']
^accounts/ ^ ^confirm-email/(?P<key>[-:\w]+)/$ [name='account_confirm_email']
^accounts/ ^ ^password/reset/$ [name='account_reset_password']
^accounts/ ^ ^password/reset/done/$ [name='account_reset_password_done']
^accounts/ ^ ^password/reset/key/(?P<uidb36>[0-9A-Za-z]+)-(?P<key>.+)/$
[name='account_reset_password_from_key']
^accounts/ ^ ^password/reset/key/done/$ [name='account_reset_password_from_key_done']
^accounts/ ^social/
^accounts/ ^google/
^accounts/ ^twitter/
^accounts/ ^facebook/
^accounts/ ^facebook/login/token/$ [name='facebook_login_by_token']
```

に、python ./manage.py migrateをして、Django-allauthのをデータベースにコミットします。

どおり、したソーシャルネットワークをしてアプリにログインするには、ネットワークのソーシャルアカウントのをするがあります。

Django localhost:8000/admin にログインし、ソーシャルアカウントのをしてソーシャルSocial Applicationsにある。

ソーシャルアプリケーションのセクションにするためのをするには、プロバイダにアカウントがありません。

できるものとできるもののなについては、「[」ページ](#)をしてください。

オンラインでDjangoとソーシャルネットワークをむ

<https://riptutorial.com/ja/django/topic/4743/django>とソーシャルネットワーク

8: DjangoのCRUD

Examples

****もなCRUDの****

これらのがわからないは、[わりにこちらからすることをしてください](#)。これらのはスタックオーバーフローのドキュメントからることにしてください。

```
django-admin startproject myproject
cd myproject
python manage.py startapp myapp
```

myproject / settings.pyアプリをインストールする

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'myapp',
]
```

myappディレクトリにurls.pyというファイルをし、のビューでしてください。

```
from django.conf.urls import url
from myapp import views

urlpatterns = [
    url(r'^$', views.index, name='index'),
]
```

のurls.py ファイルをのでします。

```
from django.conf.urls import url
from django.contrib import admin
from django.conf.urls import include
from myapp import views

urlpatterns = [
    url(r'^$', views.index, name='index'),
    url(r'^myapp/', include('myapp.urls')),
    url(r'^admin/', admin.site.urls),
]
```

myappディレクトリにtemplatesというのフォルダをします。に、**templates**ディレクトリにindex.htmlというのファイルをします。のでしてください。

```

<!DOCTYPE html>
<html>
<head>
    <title>myapp</title>
</head>
<body>
    <h2>Simplest Crud Example</h2>
    <p>This shows a list of names and lets you Create, Update and Delete them.</p>
    <h3>Add a Name</h3>
    <button>Create</button>
</body>
</html>

```

views.py ファイルをすることでできる **index.html** をするビューもです。

```

from django.shortcuts import render, redirect

# Create your views here.
def index(request):
    return render(request, 'index.html', {})

```

あなたは、あなたが りむべきをっています。のステップは、モデルをすることです。これはなもなですので、 **models.py** フォルダーにのコードをしてください。

```

from __future__ import unicode_literals

from django.db import models

# Create your models here.
class Name(models.Model):
    name_value = models.CharField(max_length=100)

    def __str__(self): # if Python 2 use __unicode__
        return self.name_value

```

これにより、**Name** オブジェクトのモデルがされ、これをコマンドラインからのコマンドでデータベースにします。

```

python manage.py createsuperuser
python manage.py makemigrations
python manage.py migrate

```

Django がするがいくつかされます。これらはテーブルをセットアップし、Django のビューから **admin** データベースにアクセスできるスーパーユーザーをします。それについてえば、たちのしいモデルをビューですることができます。 **admin.py** にき、のコードをしてください。

```

from django.contrib import admin
from myapp.models import Name
# Register your models here.

admin.site.register(Name)

```

コマンドラインにると、 `python manage.py runserver` コマンドでサーバをスピンアップできます。

[http:// localhost8000 /](http://localhost8000/)にアクセスし、あなたのアプリをることができるはずです。に、 [http:// localhost8000 / admin](http://localhost8000/admin/)にして、プロジェクトにをしてください。ログインしてMYAPPテーブルのにをすると、このではになりましたので、100であることをしてください。

にアクセスするには、どこかにするがあります。 **views.py**のインデックスをして、すべてのNameオブジェクトをデータベースからりします。

```
from django.shortcuts import render, redirect
from myapp.models import Name

# Create your views here.
def index(request):
    names_from_db = Name.objects.all()
    context_dict = {'names_from_context': names_from_db}
    return render(request, 'index.html', context_dict)
```

index.htmlファイルをののようにします。

```
<!DOCTYPE html>
<html>
<head>
    <title>myapp</title>
</head>
<body>
    <h2>Simplest Crud Example</h2>
    <p>This shows a list of names and lets you Create, Update and Delete them.</p>
    {% if names_from_context %}
        <ul>
            {% for name in names_from_context %}
                <li>{{ name.name_value }} <button>Delete</button>
                <button>Update</button></li>
            {% endfor %}
        </ul>
    {% else %}
        <h3>Please go to the admin and add a Name under 'MYAPP'</h3>
    {% endif %}
    <h3>Add a Name</h3>
    <button>Create</button>
</body>
</html>
```

これはCRUDでのみみをしています。 **myapp**ディレクトリにforms.pyファイルをします。のコードをします。

```
from django import forms
from myapp.models import Name

class NameForm(forms.ModelForm):
    name_value = forms.CharField(max_length=100, help_text = "Enter a name")

    class Meta:
        model = Name
        fields = ('name_value',)
```

index.htmlをののようにします。

```

<!DOCTYPE html>
<html>
<head>
    <title>myapp</title>
</head>
<body>
    <h2>Simplest Crud Example</h2>
    <p>This shows a list of names and lets you Create, Update and Delete them.</p>
    {% if names_from_context %}
        <ul>
            {% for name in names_from_context %}
                <li>{{ name.name_value }} <button>Delete</button>
<button>Update</button></li>
            {% endfor %}
        </ul>
    {% else %}
        <h3>Please go to the admin and add a Name under 'MYAPP'</h3>
    {% endif %}
    <h3>Add a Name</h3>
    <form id="name_form" method="post" action="/">
        {% csrf_token %}
        {% for field in form.visible_fields %}
            {{ field.errors }}
            {{ field.help_text }}
            {{ field }}
        {% endfor %}
        <input type="submit" name="submit" value="Create">
    </form>
</body>
</html>

```

に、のようにして**views.py**をします。

```

from django.shortcuts import render, redirect
from myapp.models import Name
from myapp.forms import NameForm

# Create your views here.
def index(request):
    names_from_db = Name.objects.all()

    form = NameForm()

    context_dict = {'names_from_context': names_from_db, 'form': form}

    if request.method == 'POST':
        form = NameForm(request.POST)

        if form.is_valid():
            form.save(commit=True)
            return render(request, 'index.html', context_dict)
        else:
            print(form.errors)

    return render(request, 'index.html', context_dict)

```

サーバーをすると、Cでのがしたアプリケーションのバージョンがするはずです。

TODOはとをします

オンラインでDjangoのCRUDをむ <https://riptutorial.com/ja/django/topic/7317/djangoのcrud>

9: djangoのをリセットする

き

Djangoアプリケーションをするには、をクリーンアップしてリセットするだけでくのをすることができます。

Examples

Djangoのをリセットするのデータベースをし、しいものとしてする

データベースの/データベースにSQLiteをしているは、このファイルをしてください。 MySQL / Postgresまたはのデータベースシステムをしているは、データベースをして、しいデータベースをするがあります。

appフォルダのにあるmigrationsフォルダにある "init.py"ファイルのすべてのファイルをするがあります。

、フォルダは

```
/your_django_project/your_app/migrations
```

データベースとファイルをしたので、めてdjangoプロジェクトをするときとじように、のコマンドをしてください。

```
python manage.py makemigrations
python manage.py migrate
```

オンラインでdjangoのをリセットするをむ <https://riptutorial.com/ja/django/topic/9513/djangoのをリセットする>

10: F

き

Fは、DjangoがPythonオブジェクトをして、データベースのモデルフィールドやきののをPythonメモリにプルするなくするです。これにより、はのレースをし、モデルフィールドについてをフィルタリングすることができます。

- `django.db.models`からのインポートF

Examples

の

がわからないは、 [このQAの](#)をしてください。

のコードはのとなるがあります。

```
article = Article.objects.get(pk=69)
article.views_count += 1
article.save()
```

`views_count`が1337にしい、このようなクエリがします。

```
UPDATE app_article SET views_count = 1338 WHERE id=69
```

2つのクライアントがにこのにアクセスした、がこることがあり、2のHTTPリクエストがされるということです。`Article.objects.get(pk=69)`にするに`article.save()`。したがって、のは`views_count = 1337`をち、して`views_count = 1338`をデータベースにしますが、には1339です。

これをするには、`F()`をします。

```
article = Article.objects.get(pk=69)
article.views_count = F('views_count') + 1
article.save()
```

、これはそのようなクエリになります

```
UPDATE app_article SET views_count = views_count + 1 WHERE id=69
```

クエリーセットの

`id 51`のすべてのから2つのアップフォートをしたいとしましょう。

これをPythonでのみうと、`N`クエリがされます。`N`はクエリセットのアーティクルのです。

```
for article in Article.objects.filter(author_id=51):
    article.upvotes -= 2
    article.save()
# Note that there is a race condition here but this is not the focus
# of this example.
```

もしてのをPythonによってループしたり、アップボントをらしたり、されたをデータベースにし
たりするのではなく、のがありましたか

`F()` をすると、1つのクエリでそれをできます。

```
Article.objects.filter(author_id=51).update(upvotes=F('upvotes') - 2)
```

これは、のSQLクエリでできます。

```
UPDATE app_article SET upvotes = upvotes - 2 WHERE author_id = 51
```

なぜこれはいですか

- Pythonがをうわりに、データベースにをします。これは、そのようなクエリをうためにさ
れています。
- なをるためになデータベースクエリのをにします。

フィールドのをする

`F()` をして、モデルフィールドのルックアップ/をするために、モデルフィールドの `+`、`-`、`*` など
をできます。

- モデルをのようになります。

```
class MyModel(models.Model):
    int_1 = models.IntegerField()
    int_2 = models.IntegerField()
```

- ここで、`int_1`と`int_2`フィールドがこのをたす`MyModel`テーブルのすべてのオブジェクトをす
るとします`int_1 + int_2 >= 5`。 `annotate()` と `filter()` をすると、のようになります。

```
result = MyModel.objects.annotate(
    diff=F(int_1) + F(int_2)
).filter(diff__gte=5)
```

`result` にはのすべてのオブジェクトがまれるようになりました。

このでは`Integer`フィールドをしています、このメソッドはをできるすべてのフィールドでしま
す。

オンラインでFをむ <https://riptutorial.com/ja/django/topic/2765/f-->

11: Formsets

- `NewFormSet = formset_factory(SomeForm, extra=2)`
- `formset = NewFormSet(initial=[{'some_field': 'フィールド', 'other_field': 'そののフィールド'}])`

Examples

され、ユニットされたデータをつ**Formset**

`Formset`は、データグリッドのように、1ページにのフォームをレンダリングするです。この `ChoiceForm`は、`sort`のにけられているがあります。は、どののでもですか

appname/forms.py

```
from django import forms
class ChoiceForm(forms.Form):
    choice = forms.CharField()
    pub_date = forms.DateField()
```

あなたのでは、することができます `formset_factory`るコンストラクタ `Form`パラメータにそのよう `ChoiceForm`このと `extra` フォーム/フォームのなフォームをレンダリングするがあり、あなたはをループすることができますどののく `formset`ただのようなオブジェクトをそのの。

フォームセットがデータでされていないには、にしいの `extra + 1`し、フォームセットがされているには、プリント `initialized + extra`extraののフォームの。

appname/views.py

```
import datetime
from django.forms import formset_factory
from appname.forms import ChoiceForm
ChoiceFormSet = formset_factory(ChoiceForm, extra=2)
formset = ChoiceFormSet(initial=[
    {'choice': 'Between 5-15 ?',
     'pub_date': datetime.date.today(),}
])
```

`formset.print_form.as_table`のように `formset object`をこのようにループすると、

Output in rendered template

```
<tr>
<th><label for="id_form-0-choice">Choice:</label></th>
<td><input type="text" name="form-0-choice" value="Between 5-15 ?" id="id_form-0-choice" /></td>
</tr>
<tr>
<th><label for="id_form-0-pub_date">Pub date:</label></th>
<td><input type="text" name="form-0-pub_date" value="2008-05-12" id="id_form-0-pub_date">
```

```
</td>
</tr>
<tr>
<th><label for="id_form-1-choice">Choice:</label></th>
<td><input type="text" name="form-1-choice" id="id_form-1-choice" /></td>
</tr>
<tr>
<th><label for="id_form-1-pub_date">Pub date:</label></th>
<td><input type="text" name="form-1-pub_date" id="id_form-1-pub_date" /></td>
</tr>
<tr>
<th><label for="id_form-2-choice">Choice:</label></th>
<td><input type="text" name="form-2-choice" id="id_form-2-choice" /></td>
</tr>
<tr>
<th><label for="id_form-2-pub_date">Pub date:</label></th>
<td><input type="text" name="form-2-pub_date" id="id_form-2-pub_date" /></td>
</tr>
```

オンラインでFormsetsをむ <https://riptutorial.com/ja/django/topic/6082/formsets>

12: HStoreField - へのマッピング - PostgreSQLのフィールド

- `FooModel.objects.filter(field_name__key_name = 'する')`

Examples

HStoreFieldのセットアップ

まず、`HStoreField`させるためにセットアップをうがあります。

1. `django.contrib.postgres`があなたの`INSTALLED\_APPS`にあることをしてください
2. に`HStoreExtension`をします。 `CreateModel`または`AddField`のに`HStoreExtension`をすることをれないでください。

```
from django.contrib.postgres.operations import HStoreExtension
from django.db import migrations

class FooMigration(migrations.Migration):
    # put your other migration stuff here
    operations = [
        HStoreExtension(),
        ...
    ]
```

モデルにHStoreFieldをする

->このではまず`HStoreField`をして`HStoreField`。

`HStoreField`をするためのパラメータはありません。

```
from django.contrib.postgres.fields import HStoreField
from django.db import models

class Catalog(models.Model):
    name = models.CharField(max_length=200)
    titles_to_authors = HStoreField()
```

しいモデルインスタンスをする

ネイティブのPythonにをマッピングして`create()`し`create()`。

```
Catalog.objects.create(name='Library of Congress', titles_to_authors={
    'Using HStoreField with Django': 'CrazyPython and la comunidad',
    'Flabbergeists and thingamajigs': 'La Artista Fooista',
    'Pro Git': 'Scott Chacon and Ben Straub',
})
```

キーの

```
Catalog.objects.filter(titles__Pro_Git='Scott Chacon and Ben Straub')
```

containsをう

dictオブジェクトをキーワードとしてfield_name__containsます。

```
Catalog.objects.filter(titles__contains={
    'Pro Git': 'Scott Chacon and Ben Straub'})
```

SQL `@>`にします。

オンラインでHStoreField - へのマッピング - PostgreSQLのフィールドをむ

<https://riptutorial.com/ja/django/topic/2670/hstorefield----へのマッピング----postgresqlのフィールド>

13: JSONField - PostgreSQLのフィールド

- JSONField**オプション
- DjangoのJSONFieldにPostgres JSONBカラムにデータをします。これはPostgres 9.4でのみです。
- JSONFieldは、よりなスキーマがなにJSONFieldです。たとえば、データのをわずにキーをしたいや、すべてのオブジェクトがしをっていないなどです。
- あなたは、キーをしてデータをしているは、ののフィールドのわりにすることをJSONFieldとして、sのわりにJSONField々かなりることができます。

クエリの

クエリをさせることができます。たとえば、リストにがするは、2つのアンダースコアとクエリをします。

のアンダースコアでクエリをすることをれないでください。

Examples

JSONFieldの

Django 1.9で

```
from django.contrib.postgres.fields import JSONField
from django.db import models

class IceCream(models.Model):
    metadata = JSONField()
```

にじて、の**optionsすることができます。

あなたのsettings.py INSTALLED_APPSに'django.contrib.postgres'をしなければならい
ことにしてください

JSONFieldにデータをむオブジェクトをする

ネイティブのPythonでデータをしますえば、list、dict、str、None、boolなど。

```
IceCream.objects.create(metadata={
    'date': '1/1/2016',
```



```
'ordered by': 'Jon Skeet',
'buyer': {
    'favorite flavor': 'vanilla',
    'known for': ['his rep on SO', 'writing a book']
},
'special requests': ['hot sauce'],
})
```

にJSONFieldをするJSONFieldについては、「」のをしてください。

トップレベルのデータのクエリ

```
IceCream.objects.filter(metadata__ordered_by='Guido Van Rossum')
```

にネストされたデータの

チョコレートきながしたすべてのアイスクリームコーンをしてください

```
IceCream.objects.filter(metadata__buyer__favorite_flavor='chocolate')
```

せのについては、「」のをしてください。

にするデータのクエリ

はインデックスルックアップとしてされます。

```
IceCream.objects.filter(metadata__buyer__known_for__0='creating stack overflow')
```

せのについては、「」のをしてください。

JSONFieldによる

Djangoでは、JSONFieldですることはまだサポートされていません。しかし、jSQLbのPostgreSQLをつてRawSQLをしようことができます

```
from django.db.models.expressions import RawSQL
RatebookDataEntry.objects.all().order_by(RawSQL("data->>%s", ("json_objects_key",)))
```

このでは、data['json_objects_key']というのJSONFieldのdata['json_objects_key']をJSONField data。

```
data = JSONField()
```

オンラインでJSONField - PostgreSQLのフィールドをむ

<https://riptutorial.com/ja/django/topic/1759/jsonfield-----postgresqlのフィールド>

14: RangeFields - PostgreSQLのフィールドのグループ

- `from django.contrib.postgres.fields import *` `RangeField`
- `IntegerRangeField`**オプション
- `BigIntegerRangeField`** options
- `FloatRangeField`** options
- `DateTimeRangeField`**オプション
- `DateRangeField`** options

Examples

モデルにのフィールドをめる

Pythonには3の`RangeField`あります。 `IntegerField`、 `BigIntegerField`、 および`FloatField`です。 それらは`psycopg2 NumericRange`され`psycopg2`が、をネイティブPythonタプルとしてくれます。 がまれ、がされます。

```
class Book(models.Model):
    name = CharField(max_length=200)
    ratings_range = IntegerRange()
```

RangeFieldの

1. あなたの`INSTALLED_APPS` `'django.contrib.postgres'`をしてください
2. `psycopg2`インストールする

フィールドをむモデルの

`NumericRange`わりにPythonタプルとしてをするがで`NumericRange`。

```
Book.objects.create(name='Pro Git', ratings_range=(5, 5))
```

`NumericRange`メソッド

```
Book.objects.create(name='Pro Git', ratings_range=NumericRange(5, 5))
```

`contains`をう

このクエリは、が3のすべてのをします。

```
bad_books = Books.objects.filter(ratings_range__contains=(1, 3))
```

contained_byをする

このクエリは、06のをつすべてのをします。

```
all_books = Book.objects.filter(ratings_range_contained_by=(0, 6))
```

オーバーラップの

このクエリは、するすべてのを6から10にします。

```
Appointment.objects.filter(time_span__overlap=(6, 10))
```

Noneをしてをさない

このクエリは、4のをつすべてのをします。

```
maybe_good_books = Books.objects.filter(ratings_range__contains=(4, None))
```

```
from datetime import timedelta

from django.utils import timezone
from psycopg2.extras import DateTimeTZRange

# To create a "period" object we will use psycopg2's DateTimeTZRange
# which takes the two datetime bounds as arguments
period_start = timezone.now()
period_end = period_start + timedelta(days=1, hours=3)
period = DateTimeTZRange(start, end)

# Say Event.timeslot is a DateTimeRangeField

# Events which cover at least the whole selected period,
Event.objects.filter(timeslot__contains=period)

# Events which start and end within selected period,
Event.objects.filter(timeslot__contained_by=period)

# Events which, at least partially, take place during the selected period.
Event.objects.filter(timeslot__overlap=period)
```

オンラインでRangeFields - PostgreSQLのフィールドのグループをむ

<https://riptutorial.com/ja/django/topic/2630/rangefields-----postgresqlのフィールドのグループ>

15: URLルーティング

Examples

Djangoがリクエストをする

Djangoは、URLパスをビューにルーティングすることによってリクエストをします。ビューは、クライアントにをすがあります。なるURLは、なるビューによってされます。リクエストをのビューにルーティングするために、DjangoはURLまたはURLconfをべます。デフォルトプロジェクトテンプレートは<myproject>/urls.py URLconfをし<myproject>/urls.py。

あなたのURLconfは、`urlpatterns` `django.conf.urls.url()` インスタンスのリストである`urlpatterns` いうのをするpythonモジュールでなければなりません。`url()` インスタンスは、なくとも、URLとすると、ビューまたはのURLconfのいずれかのターゲットをするがあります。URLパターンがビューをターゲットにしているは、あとでパターンをにするためのをけることをおめします。

なをてみましょう

```
# In <myproject>/urls.py

from django.conf.urls import url

from myapp.views import home, about, blog_detail

urlpatterns = [
    url(r'^$', home, name='home'),
    url(r'^about/$', about, name='about'),
    url(r'^blog/(?P<id>\d+)/$', blog_detail, name='blog-detail'),
]
```

このURLconfは3つのURLパターンをします。すべてがview、`home`、`about` および`blog-detail`です。

- `url(r'^$', home, name='home'),`

にはアンカー'^'がまれ、すぐにアンカー'\$'がきます。このパターンは、URLパスがのであるとし、`myapp.views`された`home`ビューに`myapp.views`ます。

- `url(r'^about/$', about, name='about'),`

このにはアンカーがき、そのに`about/`、およびアンカーのリテラルがきます。これはURL `/about/`とし、`about`ビューにルーティングされます。すべてのでないURLは/`でまるので、Djangoはのスラッシュをにします。`

- `url(r'^blog/(?P<id>\d+)/$', blog_detail, name='blog-detail'),`

このはしです。のパターンとに、アンカーとリテラル`blog/`します。の`(?P<id>\d+)`は、キャプチャグループとされます。キャプチャグループは、そののように、のをキャプチャし、Djangoはキャ

プチャしたをとしてビューにします。

キャプチャグループのは `(?P<name>pattern)` です。 `name` はグループのをします。これは、Djangoがをビューにすためにするでもあります。パターンは、グループによってするをします。

この、は `id`、その `blog_detail` パラメータのけなければならない `id`。パターンは `\d+` です。 `\d` はパターンがだけにすることをします。 `+` は、パターンが1つのとしなければならないことをします。

いくつかのなパターン

パターン	のためにされる	マッチ
<code>\d+</code>	<code>id</code>	1つの
<code>[\w-]+</code>	スラグ	1つの、アンダースコアまたはダッシュ
<code>[0-9]{4}</code>	い	4つの、0から9
<code>[0-9]{2}</code>	い の	2つの、0から9
<code>[^/]+</code>	パスセグメント	スラッシュのもの

`blog-detail` パターンのキャプチャグループのにはリテラル `/` とアンカーがきます。

なURLはのとおりです。

- `/blog/1/` # passes `id='1'`
- `/blog/42/` # passes `id='42'`

なURLはのとおりです。

- `/blog/a/` # `'a'` does not match `'\d'`
- `/blog//` # no characters in the capturing group does not match `'+'`

Djangoは、らがでされているじでURLパターンを `urlpatterns` のパターンがじURLにするは、これがです。えは

```
urlpatterns = [
    url(r'blog/(?P<slug>[\w-]+)/$', blog_detail, name='blog-detail'),
    url(r'blog/overview/$', blog_overview, name='blog-overview'),
]
```

のURLconfでは、2のパターンにできません。パターンはURL `/blog/overview/` としますが、`blog_overview` ビューをびすわりに、URLはまず `blog-detail` パターンとし、`blog_detail` ビューを `slug='overview'` びし `slug='overview'`。

URL `/blog/overview/`が`blog_overview`ビューにルーティングされるようにするには、パターンを`blog-detail`パターンのにするがあります。

```
urlpatterns = [
    url(r'blog/overview/$', blog_overview, name='blog-overview'),
    url(r'blog/(?P<slug>[\w-]+)/$', blog_detail, name='blog-detail'),
]
```

なアプリケーションのURLをするDjango 1.9

`app_name`をして、URLをにするようにアプリのURLconfをします。

```
# In <myapp>/urls.py
from django.conf.urls import url

from .views import overview

app_name = 'myapp'
urlpatterns = [
    url(r'^$', overview, name='overview'),
]
```

これにより、ルートURLconfにアプリケーションがまれている、アプリケーションのが`'myapp'`にされます。なアプリのユーザーは、URLをむのをうはありません。

```
# In <myproject>/urls.py
from django.conf.urls import include, url

urlpatterns = [
    url(r'^myapp/', include('myapp.urls')),
]
```

なアプリケーションは、アプリケーションのをしてURLをリバースできるようになりました。

```
>>> from django.urls import reverse
>>> reverse('myapp:overview')
'/myapp/overview/'
```

rootのURLconfは、`namespace`パラメータでインスタンスのをできます。

```
# In <myproject>/urls.py
urlpatterns = [
    url(r'^myapp/', include('myapp.urls', namespace='mynamespace')),
]
```

アプリケーションのとインスタンスのをしてURLをできます。

```
>>> from django.urls import reverse
>>> reverse('myapp:overview')
'/myapp/overview/'
>>> reverse('mynamespace:overview')
```

```
'/myapp/overview/'
```

インスタンスは、にされていない、デフォルトでアプリケーションになります。

オンラインでURLルーティングをむ <https://riptutorial.com/ja/django/topic/3299/urlルーティング>

16: カスタムマネージャとクエリセット

Examples

Querysetsと`as\_manager`メソッドをしてマネージャをする

Django managerは、djangoモデルがデータベースにいわせるインターフェイスです。ほとんどのdjangoクエリでされるobjectsフィールドは、にdjangoによってされたデフォルトのマネージャですこれは、カスタムマネージャをしないにのみされます。

なぜカスタムマネージャ/クエリセットをするのでしょうか

たちのコードベースにのクエリをくのけ、わりにえやすいをとってそれらをする。どのバージョンがみやすいかをでめます

- すべてのアクティブなユーザーのみをする `User.objects.filter(is_active=True)` VS `User.manager.active()`
- `User.objects.filter(is_active=True).filter(is_doctor=True).filter(specialization='Dermatology')` VS `User.manager.doctors.with_specialization('Dermatology')`

もうつのメリットは、`psychologists`が`dermatologists`であるとした、たちはマネージャでにクエリーをし、それをするということです。

は、カスタムのであるManagerすることによってされたQuerySetとしてas_managerを。

```
from django.db.models.query import QuerySet

class ProfileQuerySet(QuerySet):
    def doctors(self):
        return self.filter(user_type="Doctor", user__is_active=True)

    def with_specializations(self, specialization):
        return self.filter(specializations=specialization)

    def users(self):
        return self.filter(user_type="Customer", user__is_active=True)

ProfileManager = ProfileQuerySet.as_manager
```

のようにモデルにします

```
class Profile(models.Model):
    ...
    manager = ProfileManager()
```

モデルでmanagerをすると、モデルのobjectsはもうされません。

すべてのクエリにして**select_related**

キーをつモデル

これらのモデルでします

```
from django.db import models

class Book(models.Model):
    name= models.CharField(max_length=50)
    author = models.ForeignKey(Author)

class Author(models.Model):
    name = models.CharField(max_length=50)
```

`book.author.name`にアクセスするとします

ビューで

私たちはのこをうことができました。

```
books = Book.objects.select_related('author').all()
```

しかし、これはドライではありません。

カスタムマネージャー

```
class BookManager(models.Manager):

    def get_queryset(self):
        qs = super().get_queryset()
        return qs.select_related('author')

class Book(models.Model):
    ...
    objects = BookManager()
```

python 2.xでは`super`へのびしをするがあります

たちはビューでするがあります

```
books = Book.objects.all()
```

テンプレート/ビューでののはわれません。

カスタムマネージャをする

にしばしば、`published`フィールドのようなものをつモデルをうことがあります。このようなフィールドは、オブジェクトをするときほとんどにされるため、のようなをします。

```
my_news = News.objects.filter(published=True)
```

あまりにもも。カスタムマネージャーをしてこれらのにできるので、のようができます。

```
my_news = News.objects.published()
```

これはのよりもみやすく、みやすいです。

appディレクトリにファイル `managers.py` をし、しい `models.Manager` クラスをします

```
from django.db import models

class NewsManager(models.Manager):

    def published(self, **kwargs):
        # the method accepts **kwargs, so that it is possible to filter
        # published news
        # i.e: News.objects.published(insertion_date__gte=datetime.now)
        return self.filter(published=True, **kwargs)
```

このクラスをするには、モデルクラスの `objects` プロパティをし `objects`。

```
from django.db import models

# import the created manager
from .managers import NewsManager

class News(models.Model):
    """ News model
    """
    insertion_date = models.DateTimeField('insertion date', auto_now_add=True)
    title = models.CharField('title', max_length=255)
    # some other fields here
    published = models.BooleanField('published')

    # assign the manager class to the objects property
    objects = NewsManager()
```

すぐニュースをのようになにできます

```
my_news = News.objects.published()
```

さらにフィルタリングをすることもできます。

```
my_news = News.objects.published(title__icontains='meow')
```

オンラインでカスタムマネージャとクエリセットをむ <https://riptutorial.com/ja/django/topic/1400/>
カスタムマネージャとクエリセット

17: クエリーセット

き

Querysetは、に、Modelからしたオブジェクトのリストであり、データベースクエリのコンパイルによってされます。

Examples

スタンドアロンモデルでのなクエリ

ここでは、いくつかのテストクエリをするためにするなモデルをします。

```
class MyModel(models.Model):
    name = models.CharField(max_length=10)
    model_num = models.IntegerField()
    flag = models.NullBooleanField(default=False)
```

id / pkが4ののモデルオブジェクトをする

IDが4のアイテムがなく、のアイテムがあるは、がスローされます。

```
MyModel.objects.get(pk=4)
```

すべてのモデルオブジェクト

```
MyModel.objects.all()
```

flagがTrueされているモデルオブジェクト

```
MyModel.objects.filter(flag=True)
```

model_numが25よりきいモデルオブジェクト

```
MyModel.objects.filter(model_num__gt=25)
```

「Cheap Item」 nameのflagとFalseされたflagつモデルオブジェクト

```
MyModel.objects.filter(name="Cheap Item", flag=False)
```

のモデルname とを

```
MyModel.objects.filter(name__contains="ch")
```

のモデル_{name} をしない

```
MyModel.objects.filter(name__icontains="ch")
```

Qオブジェクトによるクエリ

モデルの

```
class MyModel(models.Model):
    name = models.CharField(max_length=10)
    model_num = models.IntegerField()
    flag = models.NullBooleanField(default=False)
```

Qオブジェクトをして、クエリでAND、ORをできます。たとえば、flag=True または model_num>15 すべてのオブジェクトがであるとします。

```
from django.db.models import Q
MyModel.objects.filter(Q(flag=True) | Q(model_num__gt=15))
```

のことは、あなたがやる**AND**の他に、WHERE flag=True OR model_num > 15にされます。

```
MyModel.objects.filter(Q(flag=True) & Q(model_num__gt=15))
```

Qオブジェクトは~をもって**NOT**クエリをすることにもします。々がっているすべてのオブジェクトをしたいとしましょうflag=False **AND** model_num!=15たちはどうなります

```
MyModel.objects.filter(Q(flag=True) & ~Q(model_num=15))
```

filter() でQオブジェクトと "normal"パラメータをする、Qオブジェクトがにるがあります。のクエリは、flagがTrueまたは15より大きいモデルにされているモデルと、"H"でまるをします。

```
from django.db.models import Q
MyModel.objects.filter(Q(flag=True) | Q(model_num__gt=15), name__startswith="H")
```

Qオブジェクトは、filter、exclude、getなどのキーワードをるルックアップでできます。getをもってうときは、オブジェクトを1つだけすか、MultipleObjectsReturnedがするようにしてください。

ManyToManyFieldのクエリをらすn + 1の

```
# models.py:
class Library(models.Model):
    name = models.CharField(max_length=100)
    books = models.ManyToManyField(Book)

class Book(models.Model):
```

```
title = models.CharField(max_length=100)
```

```
# views.py
def myview(request):
    # Query the database.
    libraries = Library.objects.all()

    # Query the database on each iteration (len(author) times)
    # if there is 100 libraries, there will have 100 queries plus the initial query
    for library in libraries:
        books = library.books.all()
        books[0].title
        # ...

    # total : 101 queries
```

でManyToManyFieldフィールドであるフィールドにアクセスするがあることがわかっているは、ManyToManyField prefetch_relatedをしてください。

```
# views.py
def myview(request):
    # Query the database.
    libraries = Library.objects.prefetch_related('books').all()

    # Does not query the database again, since `books` is pre-populated
    for library in libraries:
        books = library.books.all()
        books[0].title
        # ...

    # total : 2 queries - 1 for libraries, 1 for books
```

prefetch_relatedは、フィールドでもできます。

```
# models.py:
class User(models.Model):
    name = models.CharField(max_length=100)

class Library(models.Model):
    name = models.CharField(max_length=100)
    books = models.ManyToManyField(Book)

class Book(models.Model):
    title = models.CharField(max_length=100)
    readers = models.ManyToManyField(User)
```

```
# views.py
def myview(request):
    # Query the database.
    libraries = Library.objects.prefetch_related('books', 'books__readers').all()

    # Does not query the database again, since `books` and `readers` is pre-populated
    for library in libraries:
        for book in library.books.all():
            for user in book.readers.all():
```

```

        user.name
        # ...

# total : 3 queries - 1 for libraries, 1 for books, 1 for readers

```

しかし、クエリーセットがされると、フェッチされたデータはデータベースをすることなくすることができません。のでは、のクエリをします。

```

# views.py
def myview(request):
    # Query the database.
    libraries = Library.objects.prefetch_related('books').all()
    for library in libraries:
        for book in library.books.filter(title__contains="Django"):
            print(book.name)

```

Django 1.7でされたPrefetchオブジェクトをして、をすることができます

```

from django.db.models import Prefetch
# views.py
def myview(request):
    # Query the database.
    libraries = Library.objects.prefetch_related(
        Prefetch('books', queryset=Book.objects.filter(title__contains="Django")
    ).all()
    for library in libraries:
        for book in library.books.all():
            print(book.name) # Will print only books containing Django for each library

```

ForeignKeyフィールドのクエリをらすn + 1

Djangoクエリーセットは、なでされます。えば

```

# models.py:
class Author(models.Model):
    name = models.CharField(max_length=100)

class Book(models.Model):
    author = models.ForeignKey(Author, related_name='books')
    title = models.CharField(max_length=100)

# views.py
def myview(request):
    # Query the database
    books = Book.objects.all()

    for book in books:
        # Query the database on each iteration to get author (len(books) times)
        # if there is 100 books, there will have 100 queries plus the initial query
        book.author
        # ...

# total : 101 queries

```

のコードは、`django`がののためにデータベースにいわせるようにします。これはであり、のクエリしかたないほうがよい。

で`ForeignKey`フィールドにアクセスするがあることがわかっているは、`ForeignKey select_related`をしてください。

```
# views.py
def myview(request):
    # Query the database.
    books = Books.objects.select_related('author').all()

    for book in books:
        # Does not query the database again, since `author` is pre-populated
        book.author
        # ...

    # total : 1 query
```

`select_related`はフィールドでもできます。

```
# models.py:
class AuthorProfile(models.Model):
    city = models.CharField(max_length=100)

class Author(models.Model):
    name = models.CharField(max_length=100)
    profile = models.OneToOneField(AuthorProfile)

class Book(models.Model):
    author = models.ForeignKey(Author, related_name='books')
    title = models.CharField(max_length=100)
```

```
# views.py
def myview(request):
    books = Book.objects.select_related('author')\
        .select_related('author__profile').all()

    for book in books:
        # Does not query database
        book.author.name
        # or
        book.author.profile.city
        # ...

    # total : 1 query
```

Django クエリーセットのSQLをする

`query`セットの`query`は、あなたのクエリーのSQLとのをします。

```
>>> queryset = MyModel.objects.all()
>>> print(queryset.query)
SELECT "myapp_mymodel"."id", ... FROM "myapp_mymodel"
```

これは、デバッグでのみするがあります。されたクエリはバックエンドではありません。したがって、パラメータはにされず、SQLインジェクションにしてになります。クエリは、データベースバックエンドででないもあります。

QuerySetからとのレコードをする

のオブジェクトをするには

```
MyModel.objects.first()
```

のオブジェクトをするには

```
MyModel.objects.last()
```

フィルタをするのオブジェクト

```
MyModel.objects.filter(name='simple').first()
```

フィルタをするのオブジェクト

```
MyModel.objects.filter(name='simple').last()
```

Fオブジェクトによるなクエリ

Fオブジェクトは、モデルフィールドまたはきのをします。モデルフィールドのをし、にデータベースからPythonメモリにプルするなしに、モデルフィールドをしてデータベースをすることができます。 - F

クエリでのフィールドのをするがあるときはいつでもF()オブジェクトをするのがです。それでは、F()オブジェクトはもするものではなく、クエリーセットのでびすことはできません。それらは、じクエリーセットでフィールドのをするためにされます。

えは、えられたモデル...

```
SomeModel(models.Model):  
    ...  
    some_field = models.IntegerField()
```

...ユーザは、F()をとってのようにフィルタリングしながら、idフィールドのをすることによって、some_fieldがid 2のオブジェクトをできます。

```
SomeModel.objects.filter(some_field=F('id') * 2)
```

F('id')はじインスタンスのidをするだけです。DjangoはそれをとってするSQLをします。この、これによくたか


```
SELECT * FROM some_app_some_model
WHERE some_field = ((id * 2))
```

`F()` がないければ、これはSQLまたはPythonでのフィルタリングこれはにオブジェクトがたくさんあるのパフォーマンスをさせる `F()` でされます。

- フィルタはモデルのフィールドをできます
- `F`
- `TinyInstance`からの

`F()` クラスから

のクエリオブジェクトへのをできるオブジェクト。 - `F`ソース

これは `TinyInstance`からのをてのからたものです。

オンラインでクエリーセットをむ <https://riptutorial.com/ja/django/topic/1235/クエリーセット>

18: クラスベースのビュー

CBVをする、それぞれのジェネリッククラスにしてどのメソッドをきでできるかをにるがあることがよくあります。 [このページ](#)のdjangoドキュメントには、すべてのメソッドがフラットされたジェネリッククラスと、できるクラスがリストされています。

さらに、 [Classy Class Based View](#)の Web サイトでは、らしいインタラクティブなインターフェイスでじをしています。

Examples

クラスベースのビュー

クラスベースのビューでは、ビューをなものにするためにをてることができます。

なページについては、されるテンプレートをいてなものはありません。 [TemplateView](#)をしてくださいテンプレートをするだけです。ジョブがしました。。

views.py

```
from django.views.generic import TemplateView

class AboutView(TemplateView):
    template_name = "about.html"
```

urls.py

```
from django.conf.urls import url
from . import views

urlpatterns = [
    url('^about/', views.AboutView.as_view(), name='about'),
]
```

URLにAboutViewをしなにしてください。これは、びしであることがされるため、 `as_view()` すものです。

コンテキストデータ

によっては、テンプレートにもうしがなもあります。たとえば、ユーザーのページヘッダーに、ログアウトリンクのにプロフィールへのリンクがあるようにしたいとします。このようなは、 `get_context_data` メソッドをして `get_context_data` 。

views.py

```
class BookView(DetailView):
    template_name = "book.html"

    def get_context_data(self, **kwargs):
        """ get_context_data let you fill the template context """
        context = super(BookView, self).get_context_data(**kwargs)
        # Get Related publishers
        context['publishers'] = self.object.publishers.filter(is_active=True)
        return context
```

スーパークラスで`get_context_data`メソッドをひすがあり、デフォルトのコンテキストインスタンスがされます。このディクショナリにするアイテムはすべてテンプレートでできます。

book.html

```
<h3>Active publishers</h3>
<ul>
    {% for publisher in publishers %}
        <li>{{ publisher.name }}</li>
    {% endfor %}
</ul>
```

リストビューとビュー

テンプレートビューはなページではうまくいくので、`get_context_data`ってすべてのテンプレートビューをすることができますが、をビューとしてするよりも`get_context_data`いでしょう。

[ListView](#)と[DetailView](#)を

app / models.py

```
from django.db import models

class Pokemon(models.Model):
    name = models.CharField(max_length=24)
    species = models.CharField(max_length=48)
    slug = models.CharField(max_length=48)
```

app / views.py

```
from django.views.generic import ListView, DetailView
from .models import Pokemon
```

```
class PokedexView(ListView):
    """ Provide a list of Pokemon objects """
    model = Pokemon
    paginate_by = 25

class PokemonView(DetailView):
    model = Pokemon
```

それは、モデルのすべてのオブジェクトとのビューをリストするビューをするためになすべてです。リストにもページがられています。のものがなは、 `template_name` できます。デフォルトでは、モデルからされます。

app / templates / app / pokemon_list.html

```
<!DOCTYPE html>
<title>Pokedex</title>
<ul>{% for pokemon in pokemon_list %}
    <li><a href="{% url 'app:pokemon' pokemon.pk %}">{{ pokemon.name }}</a>
    &ndash; {{ pokemon.species }}
</ul>
```

コンテキストには、 `object_list` という2つのオブジェクトのリストとモデルの2つのbuildここでは `pokemon_list` ます。あなたがリストをページしたは、のリンクとのリンクもするがあります。[Paginator](#) オブジェクトはそれをけることができます。コンテキストデータでもできます。

app / templates / app / pokemon_detail.html

```
<!DOCTYPE html>
<title>Pokemon {{ pokemon.name }}</title>
<h1>{{ pokemon.name }}</h1>
<h2>{{ pokemon.species }} </h2>
```

とじように、コンテキストには `object` と `pokemon` にモデルオブジェクトがされます。 `pokemon` はモデルからしています。

app / urls.py

```
from django.conf.urls import url
from . import views

app_name = 'app'
urlpatterns = [
    url(r'^pokemon/$', views.PokedexView.as_view(), name='pokedex'),
    url(r'^pokemon/(?P<pk>\d+)/$', views.PokemonView.as_view(), name='pokemon'),
]
```

このスニペットでは、ビューのURLはキーをしてされています。としてスラッグをすることもできます。これは、えやすい、よりいたのURLをします。ただし、モデルにはslugというのフィールドがです。

```
url(r'^pokemon/(?P<slug>[A-Za-z0-9_-]+)/$', views.PokemonView.as_view(), name='pokemon'),
```

フィールドはslugしないは、することができます slug_fieldにDetailViewのフィールドをすように。

ページネーションには、ページパラメータをするか、URLにページをします。

フォームとオブジェクトの

オブジェクトをするためのビューをくことはにです。フォームをするがあります。フォームをするがあります。アイテムをするか、エラーがしたフォームをすがあります。 [なビューのいずれか](#)をしないうり。

app / views.py

```
from django.core.urlresolvers import reverse_lazy
from django.views.generic.edit import CreateView, UpdateView, DeleteView
from .models import Pokemon

class PokemonCreate(CreateView):
    model = Pokemon
    fields = ['name', 'species']

class PokemonUpdate(UpdateView):
    model = Pokemon
    fields = ['name', 'species']

class PokemonDelete(DeleteView):
    model = Pokemon
    success_url = reverse_lazy('pokedex')
```

CreateViewとUpdateViewは、2つの、 model、 およびfieldsありfields。 デフォルトでは、どちらも '_form'ののいたモデルについてテンプレートをしします。 template_name_suffixでのみができます。 DeleteViewは、オブジェクトをするにメッセージをしします。

UpdateViewとDeleteViewでオブジェクトをフェッチするがあります。それらはDetailViewと同じメソッドをし、URLからをし、オブジェクトフィールドをしします。

app / templates / app / pokemon_form.html

```
<form action="" method="post">
    {% csrf_token %}
    {{ form.as_p }}
    <input type="submit" value="Save" />
</form>
```

`form` は、なすすべてのフィールドをむ `form` がまれています。ここでは、 `as_p` ためにフィールドごと
にがされます。

app / templates / app / pokemon_confirm_delete.html

```
<form action="" method="post">
    {% csrf_token %}
    <p>Are you sure you want to delete "{{ object }}"?</p>
    <input type="submit" value="Confirm" />
</form>
```

`csrf_token` タグは、にする `django` ののためにです。フォームをしている URL が / をしている URL と
じであるため、アクションはのままです。

リストとと同じものをする、モデルには2つのがっています。まず、`create` と `update` は、リダイレク
ト URL がつかないとをうでしょう。それはポケモンモデルに `get_absolute_url` をすることででき
ます。2のは、がのあるをしなことです。これをするには、をするのがもなです。

app / models.py

```
from django.db import models
from django.urls import reverse
from django.utils.encoding import python_2_unicode_compatible

@python_2_unicode_compatible
class Pokemon(models.Model):
    name = models.CharField(max_length=24)
    species = models.CharField(max_length=48)

    def get_absolute_url(self):
        return reverse('app:pokemon', kwargs={'pk':self.pk})

    def __str__(self):
        return self.name
```

クラスデコレータは、すべてが Python 2 のでにすることをします。

の

views.py

```

from django.http import HttpResponseRedirect
from django.views.generic import View

class MyView(View):
    def get(self, request):
        # <view logic>
        return HttpResponseRedirect('result')

```

urls.py

```

from django.conf.urls import url
from myapp.views import MyView

urlpatterns = [
    url(r'^about/$', MyView.as_view()),
]

```

Django ドキュメントの»

Django クラスベースのビュー—CreateView の

クラスベースのビューでは、モデルから CRUD ビューをするのはにです。くの、みみの Django はではないか、まれず、の CRUD ビューをロールバックするがあります。そのような、CBV はにです。

CreateView クラスには、モデル、するフィールド、URL という3つのものです。

```

from django.views.generic import CreateView
from .models import Campaign

class CampaignCreateView(CreateView):
    model = Campaign
    fields = ('title', 'description')

    success_url = "/campaigns/list"

```

がすると、ユーザーは `success_url` リダイレクトされます。また、メソッドのすることができます `get_success_url` わりにしてし `reverse` または `reverse_lazy` URL をします。

このビューのテンプレートをするがあります。テンプレートのは、 `<app name>/<model name>_form.html` のでするがあります。モデルはにするがあります。たとえば、のアプリが `dashboard`、のビューでは、 `dashboard/campaign_form.html` というのテンプレートをするがあります。

テンプレートでは、 `form` には `form` がまれます。テンプレートのサンプルコードはのとおりです。

```

<form action="" method="post">
    {% csrf_token %}
    {{ form.as_p }}
    <input type="submit" value="Save" />
</form>

```

は、URLパターンにビューをします。

```
url('^campaign/new/$', CampaignCreateView.as_view(), name='campaign_new'),
```

URLにアクセスすると、したフィールドのフォームがされます。すると、データでモデルの新しいインスタンスをしてしようとしします。すると、ユーザーはURLにリダイレクトされます。エラーがすると、エラーメッセージとにフォームがされます。

1つのビュー、のフォーム

1つのDjangoビューでのフォームをするなをにします。

```
from django.contrib import messages
from django.views.generic import TemplateView

from .forms import AddPostForm, AddCommentForm
from .models import Comment

class AddCommentView(TemplateView):

    post_form_class = AddPostForm
    comment_form_class = AddCommentForm
    template_name = 'blog/post.html'

    def post(self, request):
        post_data = request.POST or None
        post_form = self.post_form_class(post_data, prefix='post')
        comment_form = self.comment_form_class(post_data, prefix='comment')

        context = self.get_context_data(post_form=post_form,
                                         comment_form=comment_form)

        if post_form.is_valid():
            self.form_save(post_form)
        if comment_form.is_valid():
            self.form_save(comment_form)

        return self.render_to_response(context)

    def form_save(self, form):
        obj = form.save()
        messages.success(self.request, "{} saved successfully".format(obj))
        return obj

    def get(self, request, *args, **kwargs):
        return self.post(request, *args, **kwargs)
```

オンラインでクラスベースのビューをむ <https://riptutorial.com/ja/django/topic/1220/クラスベースのビュー>

19: コンテキストプロセッサ

コンテキストプロセッサをして、テンプレートのどこにでもアクセスをします。

をする、または `dict` したいの `s` が、そのにこれらのをし `TEMPLATE_CONTEXT_PROCESSORS`。

Examples

コンテキストプロセッサをしてテンプレートの **.DEBUG** にアクセスする

myapp/context_processors.py

```
from django.conf import settings

def debug(request):
    return {'DEBUG': settings.DEBUG}
```

in settings.py

```
TEMPLATES = [
    {
        ...
        'OPTIONS': {
            'context_processors': [
                ...
                'myapp.context_processors.debug',
            ],
        },
    ],
]
```

またはバージョン < 1.9 の

```
TEMPLATE_CONTEXT_PROCESSORS = (
    ...
    'myapp.context_processors.debug',
)
```

のテンプレートでは、に

```
{% if DEBUG %} .header { background:#f00; } {% endif %}
{{ DEBUG }}
```

コンテキストプロセッサをして、すべてのテンプレートののブログエントリにアクセスする

`models.py` ファイルにされた `Post` というモデルがあり、ブログをみ、 `date_published` フィールドをとっていると `date_published` ます。

ステップ1 コンテキストプロセッサをく

context_processors.py というのファイルをアプリケーションディレクトリにまたはします

```
from myapp.models import Post

def recent_blog_posts(request):
    return {'recent_posts': Post.objects.order_by('-date_published')[0:3],} # Can change
    numbers for more/fewer posts
```

ステップ2 ファイルにコンテキストプロセッサをする

TEMPLATES の settings.py ファイルにしいコンテキストプロセッサをしてください

```
TEMPLATES = [
    {
        ...
        'OPTIONS': {
            'context_processors': [
                ...
                'myapp.context_processors.recent_blog_posts',
            ],
        },
    ],
]
```

Django の 1.9 よりのバージョンでは、これは `TEMPLATE_CONTEXT_PROCESSORS` をって settings.py settings.py でいました。

ステップ3 テンプレートにコンテキストプロセッサをする

のブログエントリーを々のビューをしてもうすはありませんどのテンプレートでも recent_blog_posts をしてください。

たとえば、home.html、のへのリンクをむサイドバーをすることができます

```
<div class="blog_post_sidebar">
    {% for post in recent_blog_posts %}
        <div class="post">
            <a href="{{post.get_absolute_url}}">{{post.title}}</a>
        </div>
    {% endfor %}
</div>
```

またはblog.htmlでのなをすることができます

```
<div class="content">
    {% for post in recent_blog_posts %}
        <div class="post_detail">
            <h2>{{post.title}}</h2>
            <p>Published on {{post.date_published}}</p>
        </div>
    {% endfor %}
</div>
```

```

        <p class="author">Written by: {{post.author}}</p>
        <p><a href="{{post.get_absolute_url}}">Permalink</a></p>
        <p class="post_body">{{post.body}}</p>
    </div>
{% endfor %}
</div>

```

テンプレートの

コンテキストプロセッサをして、グループメンバーシップまたはのクエリ/ロジックについてテンプレートをします。これにより、の/のユーザーは1つのテンプレートとなグループをしてのテンプレートをできます。

myapp / context_processors.py

```

def template_selection(request):
    site_template = 'template_public.html'
    if request.user.is_authenticated():
        if request.user.groups.filter(name="some_group_name").exists():
            site_template = 'template_new.html'

    return {
        'site_template': site_template,
    }

```

コンテキストプロセッサをにします。

テンプレートでは、コンテキストプロセッサでされたをします。

```
{% extends site_template %}
```

オンラインでコンテキストプロセッサをむ <https://riptutorial.com/ja/django/topic/491/コンテキストプロセッサ>

20: ジェンキンスとのな

Examples

Jenkins 2.0+ Pipeline Script

のJenkinsバージョン2.xには、なCIタスクをのジョブをせずにし、ビルド/テストをにバージョンできる「Build Pipeline Plugin」がしています。

これは "パイプライン"タイプのジョブでインストールすることも、プロジェクトがGithubでホストされているは、 "GitHub Organization Folder Plugin"をしてにジョブをすることができます。

Djangoサイトのためのなは、サイトのされたPythonモジュールだけをインストールするがあります。

```
#!/usr/bin/groovy

node {
    // If you are having issues with your project not getting updated,
    // try uncommenting the following lines.
    //stage 'Checkout'
    //checkout scm
    //sh 'git submodule update --init --recursive'

    stage 'Update Python Modules'
    // Create a virtualenv in this folder, and install or upgrade packages
    // specified in requirements.txt; https://pip.readthedocs.io/en/1.1/requirements.html
    sh 'virtualenv env && source env/bin/activate && pip install --upgrade -r requirements.txt'

    stage 'Test'
    // Invoke Django's tests
    sh 'source env/bin/activate && python ./manage.py runtests'
}
```

Jenkins 2.0+ Pipeline Script、 Docker Containers

に、 Dockerコンテナをし、でテストをするパイプラインスクリプトのをします。 エントリポイントは、 `manage.py` または `invoke / fabric` いずれかであり、 `runtests` コマンドがであると `runtests` ます。

```
#!/usr/bin/groovy

node {
    stage 'Checkout'
    checkout scm
    sh 'git submodule update --init --recursive'

    imageName = 'mycontainer:build'
    remotes = [
        'dockerhub-account',
```

```
]

stage 'Build'
def djangoImage = docker.build imageName

stage 'Run Tests'
djangoImage.run('', 'runtests')

stage 'Push'
for (int i = 0; i < remotes.size(); i++) {
    sh "docker tag ${imageName} ${remotes[i]}/${imageName}"
    sh "docker push ${remotes[i]}/${imageName}"
}
}
```

オンラインでジェンキンスとのなをむ <https://riptutorial.com/ja/django/topic/5873/ジェンキンスとのな>

21: ジャンゴフィルター

Examples

CBVでdjango-filterをする

django-filterは、ユーザのについてDjango QuerySetsをフィルタリングするためのシステムです。[このドキュメント](#)では、モデルとしてベースのビューでしています。

```
from django.db import models

class Product(models.Model):
    name = models.CharField(max_length=255)
    price = models.DecimalField()
    description = models.TextField()
    release_date = models.DateField()
    manufacturer = models.ForeignKey(Manufacturer)
```

フィルタはのようになります。

```
import django_filters

class ProductFilter(django_filters.FilterSet):
    name = django_filters.CharFilter(lookup_expr='iexact')

    class Meta:
        model = Product
        fields = ['price', 'release_date']
```

これをCBVするには、ListViewの`get_queryset()`をオーバーライドし、フィルタされた`querset`ます。

```
from django.views.generic import ListView
from .filters import ProductFilter

class ArticleListView(ListView):
    model = Product

    def get_queryset(self):
        qs = self.model.objects.all()
        product_filtered_list = ProductFilter(self.request.GET, queryset=qs)
        return product_filtered_list.qs
```

`f.qs` ページングなど、ビューのフィルタリングされたオブジェクトにアクセスすることは`f.qs`。これにより、フィルタリングされたオブジェクトのリストがページされます。

オンラインでジャンゴフィルターをむ <https://riptutorial.com/ja/django/topic/6101/ジャンゴフィルター>

22: スーパーバイザでセロリをべる

Examples

セロリの

セロリ

1. インストール - `pip install django-celery`
- 2.
3. なプロジェクトの。

```
- src/
- bin/celery_worker_start # will be explained later on
- logs/celery_worker.log
- stack/__init__.py
- stack/celery.py
- stack/settings.py
- stack/urls.py
- manage.py
```

4. `celery.py` ファイルを `stack/stack/` フォルダにします。

```
from __future__ import absolute_import
import os
from celery import Celery
os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'stack.settings')
from django.conf import settings # noqa
app = Celery('stack')
app.config_from_object('django.conf:settings')
app.autodiscover_tasks(lambda: settings.INSTALLED_APPS)
```

5. あなたの `stack/stack/__init__.py` にのコードをしてください

```
from __future__ import absolute_import
from .celery import app as celery_app # noqa
```

6. タスクをし、`@shared_task()` などのようにマークします。

```
@shared_task()
def add(x, y):
    print("x*y={}".format(x*y))
```

7. セルライトを「で」らせる

```
celery -A stack worker -l info
```

 みたい

のスーパーバイザ

1. セロリのをするスクリプトをします。アプリにスクリプトをします。にをし

```
stack/bin/celery_worker_start
```

```
#!/bin/bash

NAME="StackOverflow Project - celery_worker_start"

PROJECT_DIR=/home/stackoverflow/apps/proj/proj/
ENV_DIR=/home/stackoverflow/apps/proj/env/

echo "Starting $NAME as `whoami`"

# Activate the virtual environment
cd "${PROJECT_DIR}"

if [ -d "${ENV_DIR}" ]
then
    . "${ENV_DIR}bin/activate"
fi

celery -A stack --loglevel='INFO'
```

2. しくしたスクリプトにをする

```
chmod u+x bin/celery_worker_start
```

3. スーパーバイザをインストールしますスーパーバイザがすでにインストールされているはこのテストをスキップします

```
apt-get install supervisor
```

4. あなたのスーパーバイザのファイルをして、セロリをします。

```
/etc/supervisor/conf.d/stack_supervisor.conf
```

```
[program:stack-celery-worker]
command = /home/stackoverflow/apps/stack/src/bin/celery_worker_start
user = polsha
stdout_logfile = /home/stackoverflow/apps/stack/src/logs/celery_worker.log
redirect_stderr = true
environment = LANG = en_US.UTF-8,LC_ALL = en_US.UTF-8
numprocs = 1
autostart = true
autorestart = true
startsecs = 10
stopwaitsecs = 600
priority = 998
```

5. みりとの

```
sudo supervisorctl reread
    stack-celery-worker: available
sudo supervisorctl update
    stack-celery-worker: added process group
```


6. コマンド

```
sudo supervisorctl status stack-celery-worker
stack-celery-worker      RUNNING    pid 18020, uptime 0:00:50
sudo supervisorctl stop stack-celery-worker
stack-celery-worker: stopped
sudo supervisorctl start stack-celery-worker
stack-celery-worker: started
sudo supervisorctl restart stack-celery-worker
stack-celery-worker: stopped
stack-celery-worker: started
```

セレブとセロリ+ラビットMQ

Celeryでは、メッセージ・パッシングをするためにブローカが必要です。私たちはRabbitMQをしています。セットアップがで、うまくサポートされているからです。

のコマンドをしてrabbitmqをインストールします。

```
sudo apt-get install rabbitmq-server
```

インストールがしたら、ユーザーをし、ホストをしてアクセスをします。

```
sudo rabbitmqctl add_user myuser mypassword
sudo rabbitmqctl add_vhost myvhost
sudo rabbitmqctl set_user_tags myuser mytag
sudo rabbitmqctl set_permissions -p myvhost myuser ".*" ".*" ".*"
```

サーバーをするには

```
sudo rabbitmq-server
```

々はpipでセロリをインストールすることができます

```
pip install celery
```

あなたのDjango settings.pyファイルでは、ブローカーURLはのようになります

```
BROKER_URL = 'amqp://myuser:mypassword@localhost:5672/myvhost'
```

セロリのをする

```
celery -A your_app worker -l info
```

このコマンドは、Celeryワーカーがdjangoアプリケーションでされたタスクをするようにします。

Supervisorは、UnixプロセスをしてできるようにするPythonプログラムです。クラッシュしたプ

ロセスをすることもできます。たちはセロリのがにっていることをするためにそれをします。

まず、スーパーバイザのインストール

```
sudo apt-get install supervisor
```

スーパーバイザconf.d/etc/supervisor/conf.d/your_proj.confにyour_proj.confファイルをしします。

```
[program:your_proj_celery]
command=/home/your_user/your_proj/.venv/bin/celery --app=your_proj.celery:app worker -l info
directory=/home/your_user/your_proj
numprocs=1
stdout_logfile=/home/your_user/your_proj/logs/celery-worker.log
stderr_logfile=/home/your_user/your_proj/logs/low-worker.log
autostart=true
autorestart=true
startsecs=10
```

ファイルがされてされると、supervisorctlコマンドをしてしいプログラムをSupervisorにできます。に、Supervisorに/etc/supervisor/conf.dディレクトリのまたはされたプログラムをすようにします

```
sudo supervisorctl reread
```

それにくをするようにすることによって、

```
sudo supervisorctl update
```

たちのプログラムがされると、いなく、、、またはのがながます。

```
sudo supervisorctl status
```

セロリインスタンスをするには

```
sudo supervisorctl restart your_proj_celery
```

オンラインでスーパーバイザでセロリをべるをむ <https://riptutorial.com/ja/django/topic/7091/スーパーバイザでセロリをべる>

23: セキュリティ

Examples

クロスサイトスクリプティングXSS

XSSは、HTMLまたはJSコードをページにすることでされます。は、[クロスサイトスクリプティングとは](#)をしてください。

このをぐため、デフォルトではDjangoはテンプレートのをすをエスケープします。

のをえると、

```
context = {
    'class_name': 'large' style="font-size:4000px",
    'paragraph': (
        "<script type=\"text/javascript\">alert('hello world!');</script>"),
}
```

```
<p class="{{ class_name }}">{{ paragraph }}</p>
<!-- Will be rendered as: -->
<p class="large" style="font-size: 4000px">&lt;script&gt;alert(&#39;hello
world!&#39;);&lt;/script&gt;</p>
```

してにレンダリングしたいHTMLをむがある、にであるとわなければなりません

```
<p class="{{ class_name|safe }}">{{ paragraph }}</p>
<!-- Will be rendered as: -->
<p class="large" style="font-size: 4000px">&lt;script&gt;alert(&#39;hello
world!&#39;);&lt;/script&gt;</p>
```

すべてののをむブロックがある、ローカルでエスケープのをにすることができます

```
{% autoescape off %}
<p class="{{ class_name }}">{{ paragraph }}</p>
{% endautoescape %}
<!-- Will be rendered as: -->
<p class="large" style="font-size: 4000px"><script>alert('hello world!');</script></p>
```

また、テンプレートのをなものとしてマークすることもできます。

```
from django.utils.safestring import mark_safe

context = {
    'class_name': 'large' style="font-size:4000px",
    'paragraph': mark_safe(
        "<script type=\"text/javascript\">alert('hello world!');</script>"),
}
```

```
<p class="{ class_name }">{ paragraph }</p>
<!-- Will be rendered as: -->
<p class="large" style="font-size: 4000px"><script>alert('hello
world!');</script></p>
```

`format_html`などのDjangoユーティリティの中には、すでにとマークされたをすものもあります。

```
from django.utils.html import format_html

context = {
    'var': format_html('<b>{}</b> {}'.format('hello', '<i>world!</i>')),
}
```

```
<p>{{ var }}</p>
<!-- Will be rendered as -->
<p><b>hello</b> &lt;i>world!&lt;/i></p>
```

クリックジャツキ

Clickjackingは、Webユーザーをクリックして、ユーザーがクリックしたこととものをクリックするというのあるテクニックです。 [もっとしくる](#)

クリック `XFrameOptionsMiddleware` をにするには、ミドルウェアクラスに `XFrameOptionsMiddleware` をします。あなたがそれをしなかった、これはにそこにあるはずです。

```
# settings.py
MIDDLEWARE_CLASSES = [
    ...
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
    ...
]
```

このミドルウェアは、にされていないか、にされていないりににされているはきされない、すべてののに 'X-Frame-Options' ヘッダーをします。デフォルトでは「SAMEORIGIN」にされています。これをするには、`X_FRAME_OPTIONS` をします。

```
X_FRAME_OPTIONS = 'DENY'
```

ビューでデフォルトのをきすることができます。

```
from django.utils.decorators import method_decorator
from django.views.decorators.clickjacking import (
    xframe_options_exempt, xframe_options_deny, xframe_options_sameorigin,
)

xframe_options_exempt_m = method_decorator(xframe_options_exempt, name='dispatch')

@xframe_options_sameorigin
def my_view(request, *args, **kwargs):
    """Forces 'X-Frame-Options: SAMEORIGIN'."""
    return HttpResponse(...)
```

```
@method_decorator(xframe_options_deny, name='dispatch')
class MyView(View):
    """Forces 'X-Frame-Options: DENY'."""

@xframe_options_exempt_m
class MyView(View):
    """Does not set 'X-Frame-Options' header when passing through the
    XFrameOptionsMiddleware.
    """
```

クロスサイトリクエスト **CSRF**

ワンクリックまたはセッションライドともばれ、CSRFまたはXSRFとされるクロスサイトリクエストは、Webサイトがするユーザーからされていないコマンドがされる、Webサイトのののであるのです。 [もっとしくる](#)

CSRFをにするには、ミドルウェアクラスに `CsrfViewMiddleware` をします。このミドルウェアはデフォルトでになっています。

```
# settings.py
MIDDLEWARE_CLASSES = [
    ...
    'django.middleware.csrf.CsrfViewMiddleware',
    ...
]
```

このミドルウェアは、のクッキーにトークンをします。がでないメソッド `GET`、`HEAD`、`OPTIONS`、`TRACE`のメソッドをするはに、`csrfmiddlewaretoken` フォームデータまたは `X-CSRFToken` ヘッダーとしてされるトークンとするがあります。これにより、をしたクライアントがCookieのであり、されたセッションでもあることがされます。

HTTPS してがわれると、なのチェックがになります。 `HTTP_REFERER` ヘッダーがののホストまたは `CSRF_TRUSTED_ORIGINS` ホスト [1.9](#)の としない、はされます。

POST メソッドをするフォームは、CSRF トークンをテンプレートにめるがあります。 `{% csrf_token %}` テンプレートタグはのフィールドをし、レスポンスにCookieがされていることをします

```
<form method='POST'>
{% csrf_token %}
...
</form>
```

`@csrf_exempt` デコレータをすると、CSRF にしてではない々のビューをすることができます。

```
from django.views.decorators.csrf import csrf_exempt

@csrf_exempt
def my_view(request, *args, **kwargs):
    """Allows unsafe methods without CSRF protection"""
    return HttpResponse(...)
```

されていませんが、このビューがCSRFにしていなければ、`CsrfViewMiddleware`をにすることができます。この、`@csrf_protect`デコレータをして々のビューをすることができます。

```
from django.views.decorators.csrf import csrf_protect

@csrf_protect
def my_view(request, *args, **kwargs):
    """This view is protected against CSRF attacks if the middleware is disabled"""
    return HttpResponse(...)
```

オンラインでセキュリティをむ <https://riptutorial.com/ja/django/topic/2957/セキュリティ>

24: データベーストランザクション

Examples

アトミックトランザクション

デフォルトでは、Djangoはただちにデータベースへのコミットをします。のコミットにがすると、データベースがましくないになるがあります。

```
def create_category(name, products):
    category = Category.objects.create(name=name)
    product_api.add_products_to_category(category, products)
    activate_category(category)
```

のシナリオでは、

```
>>> create_category('clothing', ['shirt', 'trousers', 'tie'])
-----
ValueError: Product 'trousers' already exists
```

ズボンをカテゴリにしようとするにがします。こので、カテゴリはすでにされており、シャツがされています。

なカテゴリとをむコードは、コードをして `create_category()` メソッドをもうびすにでするがあります。そうしないと、するカテゴリがされます。

`django.db.transaction` モジュールをすると、のデータベースを [アトミックトランザクション](#) にできます。

[a]のデータベース。すべてがするか、もこらない。

のシナリオにすると、これは [デコレータ](#) としてできます。

```
from django.db import transaction

@transaction.atomic
def create_category(name, products):
    category = Category.objects.create(name=name)
    product_api.add_products_to_category(category, products)
    activate_category(category)
```

または、 [コンテキストマネージャ](#) をして

```
def create_category(name, products):  
    with transaction.atomic():  
        category = Category.objects.create(name=name)  
        product_api.add_products_to_category(category, products)  
        activate_category(category)
```

、トランザクションのどのがすると、データベースのはコミットされません。

オンラインでデータベーストランザクションをむ <https://riptutorial.com/ja/django/topic/5555/データベーストランザクション>

25: データベースのセットアップ

Examples

MySQL / MariaDB

DjangoはMySQL 5.5をサポートしています。

いくつかのパッケージがインストールされていることをしてください

```
$ sudo apt-get install mysql-server libmysqlclient-dev
$ sudo apt-get install python-dev python-pip          # for python 2
$ sudo apt-get install python3-dev python3-pip        # for python 3
```

Python MySQLドライバの1つ `mysqlclient` はDjangoのです

```
$ pip install mysqlclient      # python 2 and 3
$ pip install MySQL-python    # python 2
$ pip install pymysql         # python 2 and 3
```

データベースエンコーディングはDjangoですることはできませんが、データベースレベルでするがあります。 `my.cnf`または `/etc/mysql/mariadb.conf/*.*.cnf` で `default-character-set` をし、エンコーディングをします。

```
[mysql]
#default-character-set = latin1      #default on some systems.
#default-character-set = utf8mb4    #default on some systems.
default-character-set = utf8

...

[mysqld]
#character-set-server = utf8mb4
#collation-server = utf8mb4_general_ci
character-set-server = utf8
collation-server = utf8_general_ci
```

MySQLまたはMariaDBのデータベース

```
#myapp/settings/settings.py

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'DB_NAME',
        'USER': 'DB_USER',
        'PASSWORD': 'DB_PASSWORD',
        'HOST': 'localhost', # Or an IP Address that your database is hosted on
        'PORT': '3306',
        #optional:
        'OPTIONS': {
```

```

        'charset' : 'utf8',
        'use_unicode' : True,
        'init_command': 'SET '
            'storage_engine=INNODB,'
            'character_set_connection=utf8,'
            'collation_connection=utf8_bin'
            #'sql_mode=STRICT_TRANS_TABLES,'      # see note below
            #'SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED',
    },
    'TEST_CHARSET': 'utf8',
    'TEST_COLLATION': 'utf8_general_ci',
}
}

```

OracleのMySQLコネクタをしている、ENGINEはのようになります。

```
'ENGINE': 'mysql.connector.django',
```

データベースをするときは、エンコードとをするがあります。

```
CREATE DATABASE mydatabase CHARACTER SET utf8 COLLATE utf8_bin
```

MySQL 5.7およびしくインストールされたMySQL 5.6では、sql_modeオプションのデフォルトにはSTRICT_TRANS_TABLESがまれています。このオプションは、にデータがたりてられた、をエラーにエスカレートします。Djangoは、データSTRICT_TRANS_TABLESまたはSTRICT_ALL_TABLESのいずれかをくために、MySQLのモードをにすることをくします。/etc/my.cnfへのをにするにはsql-mode = STRICT_TRANS_TABLES

PostgreSQL

いくつかのパッケージがインストールされていることをしてください

```

sudo apt-get install libpq-dev
pip install psycopg2

```

PostgreSQLのデータベース

```

#myapp/settings/settings.py

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'myprojectDB',
        'USER': 'myprojectuser',
        'PASSWORD': 'password',
        'HOST': '127.0.0.1',
        'PORT': '5432',
    }
}

```

いバージョンでは、django.db.backends.postgresql_psycopg2することもできます。

Postgresqlをする、いくつかのにアクセスできます

Modelfields

```
ArrayField          # A field for storing lists of data.
HStoreField          # A field for storing mappings of strings to strings.
JSONField            # A field for storing JSON encoded data.
IntegerRangeField    # Stores a range of integers
BigIntegerRangeField # Stores a big range of integers
FloatRangeField      # Stores a range of floating point values.
DateTimeRangeField   # Stores a range of timestamps
```

sqlite

sqliteはDjangoのデフォルトです。 はいため、ではしないでください。

```
#myapp/settings/settings.py

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': 'db/development.sqlite3',
        'USER': '',
        'PASSWORD': '',
        'HOST': '',
        'PORT': '',
    },
}
```

フィクスチャは、データベースのデータです。 のデータをすでになっているときにもなは、
dumpdata コマンドをうこと dumpdata

```
./manage.py dumpdata > databasedump.json          # full database
./manage.py dumpdata myapp > databasedump.json      # only 1 app
./manage.py dumpdata myapp.mymodel > databasedump.json # only 1 model (table)
```

これにより、jsonファイルがされます。 このファイルは、

```
./manage.py loaddata databasedump.json
```

ファイルをせずに loaddata をする、 Djangoはアプリケーションの fixtures フォルダまたは
FIXTURE_DIRS でされるディレクトリのリストを FIXTURE_DIRS 、わりにそのコンテンツをします。

```
/myapp
  /fixtures
    myfixtures.json
    morefixtures.xml
```

なファイルは JSON, XML or YAML

JSONの

```
[
  {
    "model": "myapp.person",
    "pk": 1,
    "fields": {
      "first_name": "John",
      "last_name": "Lennon"
    }
  },
  {
    "model": "myapp.person",
    "pk": 2,
    "fields": {
      "first_name": "Paul",
      "last_name": "McCartney"
    }
  }
]
```

YAMLの

```
- model: myapp.person
  pk: 1
  fields:
    first_name: John
    last_name: Lennon
- model: myapp.person
  pk: 2
  fields:
    first_name: Paul
    last_name: McCartney
```

XMLの

```
<?xml version="1.0" encoding="utf-8"?>
<django-objects version="1.0">
  <object pk="1" model="myapp.person">
    <field type="CharField" name="first_name">John</field>
    <field type="CharField" name="last_name">Lennon</field>
  </object>
  <object pk="2" model="myapp.person">
    <field type="CharField" name="first_name">Paul</field>
    <field type="CharField" name="last_name">McCartney</field>
  </object>
</django-objects>
```

ジャンゴカサンドラエンジン

- インストール `pip $ pip install django-cassandra-engine`
- `settings.py` ファイルの `INSTALLED_APPS` への `INSTALLED_APPS =`
`['django_cassandra_engine']`
- Cange データベースの Standart

スタンダード

```
DATABASES = {
    'default': {
        'ENGINE': 'django_cassandra_engine',
        'NAME': 'db',
        'TEST_NAME': 'test_db',
        'HOST': 'db1.example.com,db2.example.com',
        'OPTIONS': {
            'replication': {
                'strategy_class': 'SimpleStrategy',
                'replication_factor': 1
            }
        }
    }
}
```

Cassandraはしいユーザ—cqlshをします

```
DATABASES = {
    'default': {
        'ENGINE': 'django_cassandra_engine',
        'NAME': 'db',
        'TEST_NAME': 'test_db',
        'USER_NAME': 'cassandradb',
        'PASSWORD': '123cassandra',
        'HOST': 'db1.example.com,db2.example.com',
        'OPTIONS': {
            'replication': {
                'strategy_class': 'SimpleStrategy',
                'replication_factor': 1
            }
        }
    }
}
```

```
}
```

オンラインでデータベースのセットアップをむ <https://riptutorial.com/ja/django/topic/4933/データベースのセットアップ>

26: データベースルーター

Examples

データベースルーティングファイルの

Djangoでのデータベースをするには、 `settings.py` それぞれをしてください

```
DATABASES = {
    'default': {
        'NAME': 'app_data',
        'ENGINE': 'django.db.backends.postgresql',
        'USER': 'django_db_user',
        'PASSWORD': os.environ['LOCAL_DB_PASSWORD']
    },
    'users': {
        'NAME': 'remote_data',
        'ENGINE': 'django.db.backends.mysql',
        'HOST': 'remote.host.db',
        'USER': 'remote_user',
        'PASSWORD': os.environ['REMOTE_DB_PASSWORD']
    }
}
```

`dbouters.py` ファイルをして、データベースのクラスごとにどのモデルをするかをします。たとえば、 `remote_data` にされているリモートデータのは、のようにします。

```
class DbRouter(object):
    """
    A router to control all database operations on models in the
    auth application.
    """
    def db_for_read(self, model, **hints):
        """
        Attempts to read remote models go to remote database.
        """
        if model._meta.app_label == 'remote':
            return 'remote_data'
        return 'app_data'

    def db_for_write(self, model, **hints):
        """
        Attempts to write remote models go to the remote database.
        """
        if model._meta.app_label == 'remote':
            return 'remote_data'
        return 'app_data'

    def allow_relation(self, obj1, obj2, **hints):
        """
        Do not allow relations involving the remote database
        """
        if obj1._meta.app_label == 'remote' or \
            obj2._meta.app_label == 'remote':
```

```

        return False
    return None

def allow_migrate(self, db, app_label, model_name=None, **hints):
    """
    Do not allow migrations on the remote database
    """
    if model._meta.app_label == 'remote':
        return False
    return True

```

に、 settings.py dbrouter.pyをし settings.py

```

DATABASE_ROUTERS = ['path.to.DbRouter', ]

```

コードになるデータベースをする

のobj.save() メソッドはデフォルトのデータベースをします。データベースルーターをするは、
db_for_writeされたデータベースをします。をしてきすることができます。

```

obj.save(using='other_db')
obj.delete(using='other_db')

```

に、のために

```

MyModel.objects.using('other_db').all()

```

オンラインでデータベースルーターをむ <https://riptutorial.com/ja/django/topic/3395/データベースルーター>

27: デバッグ

Pdb

Pdbは、Pdbプロンプトに`globals()`または`locals()`をそれぞれすることによって、`globals()`スコープまたはローカルスコープのものをすべてすることもできます。

Examples

PythonデバッガPdbの

もなDjangoのデバッグツールは、Pythonライブラリのである`pdb`です。

ビュースクリプト

な`views.py`スクリプトを試みましょう

```
from django.http import HttpResponse

def index(request):
    foo = 1
    bar = 0

    bug = foo/bar

    return HttpResponse("%d goes here." % bug)
```

サーバーをするコンソールコマンド

```
python manage.py runserver
```

インデックスページをみようとすると、Djangoが`ZeroDivisionError`をげることにはわかですが、コードでバグがいとわれるは、にはになるがあります。

ブレークポイントの

いにも、たちはそのバグをするためにブレークポイントをすることができます

```
from django.http import HttpResponse

# Pdb import
import pdb

def index(request):
    foo = 1
    bar = 0
```



```
# This is our new breakpoint
pdb.set_trace()

bug = foo/bar

return HttpResponse("%d goes here." % bug)
```

pdbをしてサーバーをするコンソールコマンド

```
python -m pdb manage.py runserver
```

すぐロードブレイクポイントがシェルでプロンプトPdbをすると、ブラウザがになります。

pdbシェルによるデバッグ

シェルをしてスクリプトとやりとりすることによって、そのビューをデバッグするです。

```
> ../views.py(12)index()
-> bug = foo/bar
# input 'foo/bar' expression to see division results:
(Pdb) foo/bar
*** ZeroDivisionError: division by zero
# input variables names to check their values:
(Pdb) foo
1
(Pdb) bar
0
# 'bar' is a source of the problem, so if we set it's value > 0...
(Pdb) bar = 1
(Pdb) foo/bar
1.0
# exception gone, ask pdb to continue execution by typing 'c':
(Pdb) c
[03/Aug/2016 10:50:45] "GET / HTTP/1.1" 200 111
```

のでは、たちのOKをし、にじてしていることがわかります。

pdbループをめるには、シェルにqをするだけです。

Djangoデバッグツールバーの

まず、 [django-debug-toolbar](#)をインストールするがあります

```
pip install django-debug-toolbar
```

settings.py

に、それをプロジェクトにインストールされているアプリケーションにみみますが、してください。デバッグツールバーのようなアプリケーションやミドルウェアには、のsettings.pyファイルをするをおsettings.pyます。

```
# If environment is dev...
DEBUG = True

INSTALLED_APPS += [
    'debug_toolbar',
]

MIDDLEWARE += ['debug_toolbar.middleware.DebugToolbarMiddleware']
```

デバッグツールバーもなファイルにしているので、なアプリもめてください。

```
INSTALLED_APPS = [
    # ...
    'django.contrib.staticfiles',
    # ...
]

STATIC_URL = '/static/'

# If environment is dev...
DEBUG = True

INSTALLED_APPS += [
    'debug_toolbar',
]
```

によっては、 settings.py INTERNAL_IPS も settings.py があり settings.py

```
INTERNAL_IPS = ('127.0.0.1', )
```

urls.py

urls.py では、ドキュメントがするように、のスニペットでデバッグツールバーのルーティングをにすることができます。

```
if settings.DEBUG and 'debug_toolbar' in settings.INSTALLED_APPS:
    import debug_toolbar
    urlpatterns += [
        url(r'^__debug__/', include(debug_toolbar.urls)),
    ]
```

インストールにツールバーのをする

```
python manage.py collectstatic
```

つまり、デバッグツールバーがプロジェクトのページにされ、SQL、ファイル、などのさまざまながされます。

HTML

また、 django-debug-toolbar は、にレンダリングするために、 *Content-type* の text/html 、 <html> 、および <body> タグをとします。

は、すべてのをしたが、デバッグツールバーはまだレンダリングされていない [この""](#)のをしてそれをしようとしします。

"assert False"をうと、

に、コードにのをします。

```
assert False, value
```

このがされたときにdjangoがエラーメッセージとしてされたでAssertionErrorをさせます。

これがビューまたはビューからびされたコードでし、DEBUG=Trueがされているは、くのデバッグをむでなスタックトレースがブラウザにされま。

あなたがしたときにラインをすることをれないでください

デバッガをうわりに、よりくのドキュメント、テスト、ロギング、アサーションをくことをする

デバッグにはとががかかります。

デバッガでバグをするのではなく、コードをするためにをやすことをしてください。

- [テスト](#)をしてします。PythonとDjangoには、みみのテストフレームワークがされています。これは、デバッガをってでうよりもはるかにくコードをテストするためにできます。
- あなたの、クラス、モジュールのためのなドキュメントをく。 [PEP 257](#)と[GoogleのPython Style Guide](#)はいドキュメントストリングをくためのいをしています。
- [ロギング](#)をして、およびデプロイにプログラムからをします。
- なでコードにassertイオンをassert あいまいさをらし、がしたときにそれをする。

ボーナスドキュメントとテストをみわせるための[doctest](#)をく

[オンラインでデバッグをむ](#) <https://riptutorial.com/ja/django/topic/5072/デバッグ>

28: テンプレート

Examples

ビューコンテキストでされたには、カッコをしてアクセスできます。

あなたの `views.py`

```
class UserView(TemplateView):
    """ Supply the request user object to the template """

    template_name = "user.html"

    def get_context_data(self, **kwargs):
        context = super(UserView, self).get_context_data(**kwargs)
        context.update(user=self.request.user)
        return context
```

`user.html`

```
<h1>{{ user.username }}</h1>

<div class="email">{{ user.email }}</div>
```

ドットはのものにアクセスします

- ・ オブジェクトのプロパティ。たとえば、 `user.username` は `{{ user.username }}`
- ・ 、 えば `request.GET["search"]` は `{{ request.GET.search }}`
- ・ のないメソッド `users.count()` は `{{ user.count }}`

テンプレートはをるメソッドにアクセスできません。

また、をテストしてループバックすることもできます。

```
{% if user.is_authenticated %}
    {% for item in menu %}
        <li><a href="{{ item.url }}">{{ item.name }}</a></li>
    {% endfor %}
{% else %}
    <li><a href="{% url 'login' %}">Login</a>
{% endif %}
```

URLは、 `{% url 'name' %}` をしてアクセスします。ここで、は `urls.py` にします。

```
{% url 'login' %} - /accounts/login/
{% url 'user_profile' user.id %} - URLのはにされます
{% url next %} - URLはにすることができます
```

クラスベースのビューでのテンプレート

カスタムでテンプレートにデータをすことができます。

あなたの `views.py`

```
from django.views.generic import TemplateView
from MyProject.myapp.models import Item

class ItemView(TemplateView):
    template_name = "item.html"

    def items(self):
        """ Get all Items """
        return Item.objects.all()

    def certain_items(self):
        """ Get certain Items """
        return Item.objects.filter(model_field="certain")

    def categories(self):
        """ Get categories related to this Item """
        return Item.objects.get(slug=self.kwargs['slug']).categories.all()
```

あなたの `item.html` なリスト

```
{% for item in view.items %}
<ul>
    <li>{{ item }}</li>
</ul>
{% endfor %}
```

また、データののプロパティをすることもできます。

モデル `Item` に `name` フィールドがあるとします

```
{% for item in view.certain_items %}
<ul>
    <li>{{ item.name }}</li>
</ul>
{% endfor %}
```

ベースのビューでのテンプレート

のように、ベースのビューでテンプレートをできます。

```
from django.shortcuts import render

def view(request):
    return render(request, "template.html")
```

テンプレートをすは、のようになります。

```
from django.shortcuts import render

def view(request):
    context = {"var1": True, "var2": "foo"}
    return render(request, "template.html", context=context)
```

に、 `template.html` では、をのようによきます。

```
<html>
{% if var1 %}
    <h1>{{ var2 }}</h1>
{% endif %}
</html>
```

テンプレートフィルタ

Djangoのテンプレートシステムには、タグとフィルタがみまれています。これはテンプレートであり、コンテンツをのでレンダリングします。パイプでのフィルタをでき、とにフィルタにをけることができます。

```
{{ "MAINROAD 3222"|lower }}      # mainroad 3222
{{ 10|add:15 }}                  # 25
{{ "super"|add:"glue" }}         # superglue
{{ "A7"|add:"00" }}              # A700
{{ myDate | date:"D d M Y" }}    # Wed 20 Jul 2016
```

なみみフィルターのリストは、 <https://docs.djangoproject.com/en/dev/ref/templates/builtins/#ref-templates-builtins-filters> にあります。

カスタムフィルタの

のテンプレートフィルタをするには、`app`フォルダに `templatetags` というのフォルダをします。に、 `__init__.py` し、フィルタをむファイルをファイルにします。

```
#!/myapp/templatetags/filters.py
from django import template

register = template.Library()

@register.filter(name='tostring')
def to_string(value):
    return str(value)
```

にフィルタをするには、テンプレートにロードするがあります。

```
#templates/mytemplate.html
{% load filters %}
{% if customer_id|tostring = customer %} Welcome back {% endif %}
```

トリック

フィルタははシンプルにえますが、それはいくつかのすてきなことをすることをにします

```
{% for x in ""|ljust:"20" %}Hello World!{% endfor %}      # Hello World!Hello World!Hel...
{{ user.name.split|join:"_" }} ## replaces whitespace with '_'
```

については、[テンプレートタグ](#)をしてください。

がテンプレートでびされないようにする

オブジェクトがテンプレートコンテキストにさらされると、そののないメソッドができます。これは、これらの「ゲッター」であるにです。しかし、これらのによってデータがされたり、があったりするとなることがあります。テンプレートライターをしているはありますが、はのをしていないか、ってったをってびすをとっているかもしれません。

のモデルをえます。

```
class Foobar(models.Model):
    points_credit = models.IntegerField()

    def credit_points(self, nb_points=1):
        """Credit points and return the new points credit value."""
        self.points_credit = F('points_credit') + nb_points
        self.save(update_fields=['points_credit'])
        return self.points_credit
```

これをいなくテンプレートにいた

```
You have {{ foobar.credit_points }} points!
```

これにより、テンプレートがびされるたびにポイントのがえます。あなたはそれにかないかもしれません。

これをぐには、のあるメソッドにして`alters_data`を`True`にするがあります。これは、テンプレートからそれらをびすことがになります。

```
def credit_points(self, nb_points=1):
    """Credit points and return the new points credit value."""
    self.points_credit = F('points_credit') + nb_points
    self.save(update_fields=['points_credit'])
    return self.points_credit
credit_points.alters_data = True
```

{extend}、{include}、{blocks}の

- **{extends}** としてえられたテンプレートをのテンプレートのとします。 `{% extends 'parent_template.html' %}`。

- **{block} {endblock}** これはテンプレートのセクションをするためにされるので、のテンプレートがこのテンプレートをする、にかれたHTMLコードをきえることができます。ブロックはそれでされます。 `{% block content %} <html_code> {% endblock %}`。
- **{include}** これはのテンプレートにテンプレートをしします。インクルードされたテンプレートはのコンテキストをけるので、カスタムもできます。な `{% include 'template_name.html' %}`、の `{% include 'template_name.html' with variable='value' variable2=8 %}`

ガイド

すべてのコードにしてのレイアウトをつフロントエンドサイドコードをしていて、すべてのテンプレートにしてコードをりしたくないとします。 Djangoはそうするためのビルドみのタグをします。

じレイアウトをする3つのテンプレートをつ1つのブログWebサイトがあるとします。

```
project_directory
..
templates
  front-page.html
  blogs.html
  blog-detail.html
```

1 base.html ファイルをし、

```
<html>
<head>
</head>

<body>
  {% block content %}
  {% endblock %}
</body>
</html>
```

2 blog.html ように、

```
{% extends 'base.html' %}

{% block content %}
  # write your blog related code here
{% endblock %}

# None of the code written here will be added to the template
```

ここではレイアウトをして、HTMLレイアウトがblog.html ファイルでできるようにしました。 `{% block %}` コンセプトはテンプレートのです。これにより、あなたのをすべてむな "スケルトン"テンプレートがきできるブロックをします。

3あなたの3つのテンプレートのすべてがじHTML divをとっているとします。これはのあるをしします

.3かれるわりに、しいテンプレート`posts.html`します。

blog.html

```
{% extends 'base.html' %}

{% block content %}
    # write your blog related code here
    {% include 'posts.html' %} # includes posts.html in blog.html file without passing any
data
    <!-- or -->
    {% include 'posts.html' with posts=postdata %} # includes posts.html in blog.html file
with passing posts data which is context of view function returns.
{% endblock %}
```

オンラインでテンプレートをむ <https://riptutorial.com/ja/django/topic/588/テンプレート>

29: テンプレートタグとフィルタ

Examples

カスタムフィルタ

フィルタをすると、にをできます。このは**0**または**1**のをとることができます。はのとおりです。

```
{{ variable|filter_name }}
{{ variable|filter_name:argument }}
```

これはになので、フィルタはすることができます

```
{{ variable|filter_name:argument|another_filter }}
```

Pythonにされた、のはのようになります

```
print(another_filter(filter_name(variable, argument)))
```

このでは、Modelインスタンスまたはクラスまたは`verbose_name`にされるカスタムフィルタ
`verbose_name`を`verbose_name`します。が`True`されているは、モデルのな、またはそのなのがされ`True`
。

```
@register.filter
def verbose_name(model, plural=False):
    """Return the verbose name of a model.
    `model` can be either:
    - a Model class
    - a Model instance
    - a QuerySet
    - any object referring to a model through a `model` attribute.

    Usage:
    - Get the verbose name of an object
      {{ object|verbose_name }}
    - Get the plural verbose name of an object from a QuerySet
      {{ objects_list|verbose_name:True }}
    """
    if not hasattr(model, '_meta'):
        # handle the case of a QuerySet (among others)
        model = model.model
    opts = model._meta
    if plural:
        return opts.verbose_name_plural
    else:
        return opts.verbose_name
```

なタグ

カスタムテンプレートタグをするもなは、 `simple_tag` をすること `simple_tag`。これらはセットアップするのがにです。はタグになりますが、はトークンでまれたスペースをいてでられた "トークン" になります。キーワードもサポートしています。

たちのをするなタグがあります

```
{% useless 3 foo 'hello world' foo=True bar=baz.hello|capfirst %}
```

`foo` と `baz` のようなコンテキストとしましょう

```
{'foo': "HELLO", 'baz': {'hello': "world"}}
```

たちは、このようなレンダリングににたないタグをいたいとしましょう

```
HELLO;hello world;bar:World;foo:True<br/>
HELLO;hello world;bar:World;foo:True<br/>
HELLO;hello world;bar:World;foo:True<br/>
```

きのは3りされます3がのきです。

タグのはのようになります。

```
from django.utils.html import format_html_join

@register.simple_tag
def useless(repeat, *args, **kwargs):
    output = ';'.join(args + ['{}:{}'.format(*item) for item in kwargs.items()])
    outputs = [output] * repeat
    return format_html_join('\n', '{}<br/>', ((e,) for e in outputs))
```

`format_html_join` では、なHTMLとして `<br/>` をマークすることができますが、`outputs` のはマークしません。

Nodeをしたなカスタムタグ

によっては、`filter` や `simple_tag` にしてすぎることもあります。これをする、コンパイルとレンダラーをするがあります。

このでは、テンプレートタグ `verbose_name` をのでします。

<code>{% verbose_name obj %}</code>	モデルのな
<code>{% verbose_name obj 'status' %}</code>	フィールド "ステータス" のな
<code>{% verbose_name obj plural %}</code>	のモデルのな
<code>{% verbose_name obj plural capfirst %}</code>	のモデルをにしたな

<code>{% verbose_name obj 'foo' capfirst %}</code>	フィールドのとの
<code>{% verbose_name obj field_name %}</code>	からのフィールドのな
<code>{% verbose_name obj 'foo' add: '_bar' %}</code>	フィールド "foo_bar"のな

なタグでこれをうことができないは、`plural`と`capfirst`はでもでもなく、"キーワード"であるということです。らかにそれらを`'plural'`と`'capfirst'`としてすことができますが、これらののフィールドとするがあります。 `{% verbose_name obj 'plural' %}`は、「の`obj`な」または「`obj.plural`な」を`obj.plural`ますか

まずコンパイルをしましょう

```
@register.tag(name='verbose_name')
def do_verbose_name(parser, token):
    """
    - parser: the Parser object. We will use it to parse tokens into
      nodes such as variables, strings, ...
    - token: the Token object. We will use it to iterate each token
      of the template tag.
    """
    # Split tokens within spaces (except spaces inside quotes)
    tokens = token.split_contents()
    tag_name = tokens[0]
    try:
        # each token is a string so we need to parse it to get the actual
        # variable instead of the variable name as a string.
        model = parser.compile_filter(tokens[1])
    except IndexError:
        raise TemplateSyntaxError(
            "'{%}' tag requires at least 1 argument.".format(tag_name))

    field_name = None
    flags = {
        'plural': False,
        'capfirst': False,
    }

    bits = tokens[2:]
    for bit in bits:
        if bit in flags.keys():
            # here we don't need `parser.compile_filter` because we expect
            # 'plural' and 'capfirst' flags to be actual strings.
            if flags[bit]:
                raise TemplateSyntaxError(
                    "'{%}' tag only accept one occurrence of '{%}' flag".format(
                        tag_name, bit))
            flags[bit] = True
            continue
        if field_name:
            raise TemplateSyntaxError((
                "'{%}' tag only accept one field name at most. {} is the second "
                "field name encountered."
            ).format(tag_name, bit))
        field_name = parser.compile_filter(bit)
```

```
# VerboseNameNode is our renderer which code is given right below
return VerboseNameNode(model, field_name, **flags)
```

レンダラー

```
class VerboseNameNode(Node):

    def __init__(self, model, field_name=None, **flags):
        self.model = model
        self.field_name = field_name
        self.plural = flags.get('plural', False)
        self.capfirst = flags.get('capfirst', False)

    def get_field_verbose_name(self):
        if self.plural:
            raise ValueError("Plural is not supported for fields verbose name.")
        return self.model._meta.get_field(self.field_name).verbose_name

    def get_model_verbose_name(self):
        if self.plural:
            return self.model._meta.verbose_name_plural
        else:
            return self.model._meta.verbose_name

    def render(self, context):
        """This is the main function, it will be called to render the tag.
        As you can see it takes context, but we don't need it here.
        For instance, an advanced version of this template tag could look for an
        `object` or `object_list` in the context if `self.model` is not provided.
        """
        if self.field_name:
            verbose_name = self.get_field_verbose_name()
        else:
            verbose_name = self.get_model_verbose_name()
        if self.capfirst:
            verbose_name = verbose_name.capitalize()
        return verbose_name
```

オンラインでテンプレートタグとフィルタをむ <https://riptutorial.com/ja/django/topic/1305/テンプレートタグとフィルタ>

30: ビュー

き

ビューまたはビューは、にWebリクエストをけり、WebレスポンスをすPythonです。 -Djangoドキュメンテーション -

Examples

□ シンプルビュー—Hello World Equivalent

HTMLの "Hello World"テンプレートにするになビューをしましょう。

1. これをうには、 `my_project/my_app/views.py` ここではビューを `my_project/my_app/views.py` ますにし、のビューをします。

```
from django.http import HttpResponse

def hello_world(request):
    html = "<html><title>Hello World!</title><body>Hello World!</body></html>"
    return HttpResponse(html)
```

2. このビューをびすには、 `my_project/my_app/urls.py` URLパターンをするがあります。

```
from django.conf.urls import url

from . import views

urlpatterns = [
    url(r'^hello_world/$', views.hello_world, name='hello_world'),
]
```

3. サーバーをする `python manage.py runserver`

`http://localhost:8000/hello_world/`、たちのテンプレートhtmlがブラウザにされます。

オンラインでビューをむ <https://riptutorial.com/ja/django/topic/7490/ビュー>

31: フォーム

Examples

ModelFormの

サブクラスすることで、のモデルクラスからのModelFormをModelForm

```
from django import forms

class OrderForm(forms.ModelForm):
    class Meta:
        model = Order
        fields = ['item', 'order_date', 'customer', 'status']
```

Djangoフォームをゼロからするウィジェットを

フォームは、モデルとので、 `django.forms.Form` をサブクラスすることでできます。
`CharField`、 `URLField`、 `IntegerField`などのさまざまなフィールドオプションをできます。
なフォームをすると、のようになります。

```
from django import forms

class ContactForm(forms.Form):
    contact_name = forms.CharField(
        label="Your name", required=True,
        widget=forms.TextInput(attrs={'class': 'form-control'}))
    contact_email = forms.EmailField(
        label="Your Email Address", required=True,
        widget=forms.TextInput(attrs={'class': 'form-control'}))
    content = forms.CharField(
        label="Your Message", required=True,
        widget=forms.Textarea(attrs={'class': 'form-control'}))
```

ウィジェットはDjangoのHTMLユーザータグので、フォームフィールドのカスタムhtmlをレンダリングするためにできますここでされたにしてテキストボックスがレンダリングされる

`attrs`は、フォームのレンダリングされたhtmlにそのままコピーされるです。

```
content.render("name", "Your Name")
```

```
<input title="Your name" type="text" name="name" value="Your Name" class="form-control" />
```

views.pyのについてmodelFormのフィールドをする

たちがのようなモデルをっていれば、

```

from django.db import models
from django.contrib.auth.models import User

class UserModuleProfile(models.Model):
    user = models.OneToOneField(User)
    expired = models.DateTimeField()
    admin = models.BooleanField(default=False)
    employee_id = models.CharField(max_length=50)
    organisation_name = models.ForeignKey('Organizations', on_delete=models.PROTECT)
    country = models.CharField(max_length=100)
    position = models.CharField(max_length=100)

    def __str__(self):
        return self.user

```

そして、このモデルをのようにするモデルは、

```

from .models import UserModuleProfile, from django.contrib.auth.models import User
from django import forms

class UserProfileForm(forms.ModelForm):
    admin = forms.BooleanField(label="Make this User
Admin", widget=forms.CheckboxInput(), required=False)
    employee_id = forms.CharField(label="Employee Id ")
    organisation_name = forms.ModelChoiceField(label='Organisation
Name', required=True, queryset=Organizations.objects.all(), empty_label="Select an Organization")
    country = forms.CharField(label="Country")
    position = forms.CharField(label="Position")

    class Meta:
        model = UserModuleProfile
        fields = ('admin', 'employee_id', 'organisation_name', 'country', 'position',)

    def __init__(self, *args, **kwargs):
        admin_check = kwargs.pop('admin_check', False)
        super(UserProfileForm, self).__init__(*args, **kwargs)
        if not admin_check:
            del self.fields['admin']

```

フォームのMetaクラスの中には、フォームフィールドをするためにforms.pyからフォームをするにできる**init**がされていますまたはそのアクション。これについてはでします。

したがって、このフォームはユーザーですることができ、フォームのメタクラスにされているすべてのフィールドがです。しかし、ユーザーをするときにじフォームをするは、フォームのフィールドをしたくないはどうすればよいでしょうか

いくつかのロジックについてフォームをし、バックエンドからフィールドをするときに、にのをすることができます。

```

def edit_profile(request, user_id):
    context = RequestContext(request)
    user = get_object_or_404(User, id=user_id)
    profile = get_object_or_404(UserModuleProfile, user_id=user_id)
    admin_check = False
    if request.user.is_superuser:
        admin_check = True

```



```

# If it's a HTTP POST, we're interested in processing form data.
if request.method == 'POST':
    # Attempt to grab information from the raw form information.
    profile_form =
UserProfileForm(data=request.POST,instance=profile,admin_check=admin_check)
    # If the form is valid...
    if profile_form.is_valid():
        form_bool = request.POST.get("admin", "xxx")
        if form_bool == "xxx":
            form_bool_value = False
        else:
            form_bool_value = True
        profile = profile_form.save(commit=False)
        profile.user = user
        profile.admin = form_bool_value
        profile.save()
        edited = True
    else:
        print profile_form.errors

# Not a HTTP POST, so we render our form using ModelForm instance.
# These forms will be blank, ready for user input.
else:
    profile_form = UserProfileForm(instance = profile,admin_check=admin_check)

return render_to_response(
    'usermodule/edit_user.html',
    {'id':user_id, 'profile_form': profile_form, 'edited': edited, 'user':user},
    context)

```

このとおり、したフォームをしたなをしました。がフォームをしたとき、は`True`か`False`をむ`admin_check`をしました。

```

profile_form = UserProfileForm(instance = profile,admin_check=admin_check)

```

すぐいたフォームにいたら、**init**でここからす`admin_check`パラメータをキャッチしようとしていることが`admin_check`ます。が`False`の、フォームから`admin`フィールドをしてするだけです。これはモデルのフォームなので、モデルのフィールドは`null`にはできません。フォームのにフィールドがあるかどうかをするだけです。そうでないは、ビューののコードのビューコードで`False`にします。

```

form_bool = request.POST.get("admin", "xxx")
if form_bool == "xxx":
    form_bool_value = False
else:
    form_bool_value = True

```

Django フォームによるファイルアップロード

まず、`settings.py` ファイルに`MEDIA_ROOT`と`MEDIA_URL`をするがあります

```

MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
MEDIA_URL = '/media/'

```

ImageField であるもありますので、そのようなのはライブラリ `pip install pillow` を `pip install pillow`。そうしないと、のエラーがします。

```
ImportError: No module named PIL
```

ピローは、もはやされていないPythonイメージングライブラリであるPILのフォークです。ピローはPILとがあります。

Djangoには、`FileField`と`ImageField`という2つのフィールドがサーバーにアップロードされます。の2つのフィールドをフォームにするです

forms.py

```
from django import forms

class UploadDocumentForm(forms.Form):
    file = forms.FileField()
    image = forms.ImageField()
```

views.py

```
from django.shortcuts import render
from .forms import UploadDocumentForm

def upload_doc(request):
    form = UploadDocumentForm()
    if request.method == 'POST':
        form = UploadDocumentForm(request.POST, request.FILES) # Do not forget to add:
        request.FILES
        if form.is_valid():
            # Do something with our files or simply save them
            # if saved, our files would be located in media/ folder under the project's base
            folder

            form.save()
        return render(request, 'upload_doc.html', locals())
```

upload_doc.html

```
<html>
  <head>File Uploads</head>
  <body>
    <form enctype="multipart/form-data" action="" method="post"> <!-- Do not forget to
add: enctype="multipart/form-data" -->
      {% csrf_token %}
      {{ form }}
      <input type="submit" value="Save">
    </form>
  </body>
</html>
```

フィールドのとモデルへのコミットユーザーのメールの

Djangoにはユーザパスワードをするためのフォームがされています。そのは[SetPasswordForm](#)です。

ただし、ユーザのメールをするフォームはありません。フォームをしくするをするには、のがです。

のでは、のチェックをいます。

- メールはにされました。メールのやメールのチンパンジーのがなはにです。
- メールとメールはじです。フォームにはメールの2つのフィールドがあるため、はエラーがしにくいものです。

そしてに、しいメールをユーザオブジェクトにしますユーザのメールをします。 `__init__()`メソッドにはユーザオブジェクトがであることにしてください。

```
class EmailChangeForm(forms.Form):
    """
    A form that lets a user change set their email while checking for a change in the
    e-mail.
    """
    error_messages = {
        'email_mismatch': _("The two email addresses fields didn't match."),
        'not_changed': _("The email address is the same as the one already defined."),
    }

    new_email1 = forms.EmailField(
        label=_("New email address"),
        widget=forms.EmailInput,
    )

    new_email2 = forms.EmailField(
        label=_("New email address confirmation"),
        widget=forms.EmailInput,
    )

    def __init__(self, user, *args, **kwargs):
        self.user = user
        super(EmailChangeForm, self).__init__(*args, **kwargs)

    def clean_new_email1(self):
        old_email = self.user.email
        new_email1 = self.cleaned_data.get('new_email1')
        if new_email1 and old_email:
            if new_email1 == old_email:
                raise forms.ValidationError(
                    self.error_messages['not_changed'],
                    code='not_changed',
                )
        return new_email1

    def clean_new_email2(self):
        new_email1 = self.cleaned_data.get('new_email1')
        new_email2 = self.cleaned_data.get('new_email2')
        if new_email1 and new_email2:
            if new_email1 != new_email2:
                raise forms.ValidationError(
                    self.error_messages['email_mismatch'],
```

```

        code='email_mismatch',
    )
    return new_email2

def save(self, commit=True):
    email = self.cleaned_data["new_email1"]
    self.user.email = email
    if commit:
        self.user.save()
    return self.user

def email_change(request):
    form = EmailChangeForm()
    if request.method=='POST':
        form = Email_Change_Form(user,request.POST)
        if form.is_valid():
            if request.user.is_authenticated:
                if form.cleaned_data['email1'] == form.cleaned_data['email2']:
                    user = request.user
                    u = User.objects.get(username=user)
                    # get the proper user
                    u.email = form.cleaned_data['email1']
                    u.save()
                    return HttpResponseRedirect("/accounts/profile/")
            else:
                return render_to_response("email_change.html", {'form':form},
                                           context_instance=RequestContext(request))

```

オンラインでフォームをむ <https://riptutorial.com/ja/django/topic/1217/フォーム>

32: フォームウィジェット

Examples

シンプルなテキストウィジェット

もなウィジェットのはカスタムテキストです。たとえば、`<input type="tel">`をするには、`TextInput`をサブクラスし、`input_type`を`'tel'`するがあります。

```
from django.forms.widgets import TextInput

class PhoneInput(TextInput):
    input_type = 'tel'
```

ウィジェット

`MultiWidget`をすると、のウィジェットでされるウィジェットをできます。

```
from datetime import date

from django.forms.widgets import MultiWidget, Select
from django.utils.dates import MONTHS

class SelectMonthDateWidget(MultiWidget):
    """This widget allows the user to fill in a month and a year.

    This represents the first day of this month or, if `last_day=True`, the
    last day of this month.
    """

    default_nb_years = 10

    def __init__(self, attrs=None, years=None, months=None, last_day=False):
        self.last_day = last_day

        if not years:
            this_year = date.today().year
            years = range(this_year, this_year + self.default_nb_years)
        if not months:
            months = MONTHS

        # Here we will use two `Select` widgets, one for months and one for years
        widgets = (Select(attrs=attrs, choices=months.items()),
                   Select(attrs=attrs, choices=((y, y) for y in years)))
        super().__init__(widgets, attrs)

    def format_output(self, rendered_widgets):
        """Concatenates rendered sub-widgets as HTML"""
        return (
            '<div class="row">'
            '<div class="col-xs-6">{</div>'
            '<div class="col-xs-6">{</div>'
            '</div>'
        )
```

```

).format(*rendered_widgets)

def decompress(self, value):
    """Split the widget value into subwidgets values.
    We expect value to be a valid date formatted as `%Y-%m-%d`.
    We extract month and year parts from this string.
    """
    if value:
        value = date(*map(int, value.split('-')))
        return [value.month, value.year]
    return [None, None]

def value_from_datadict(self, data, files, name):
    """Get the value according to provided `data` (often from `request.POST`)
    and `files` (often from `request.FILES`, not used here)
    `name` is the name of the form field.

    As this is a composite widget, we will grab multiple keys from `data`.
    Namely: `field_name_0` (the month) and `field_name_1` (the year).
    """
    datalist = [
        widget.value_from_datadict(data, files, '{}_{}'.format(name, i))
        for i, widget in enumerate(self.widgets)]
    try:
        # Try to convert it as the first day of a month.
        d = date(day=1, month=int(datelist[0]), year=int(datelist[1]))
        if self.last_day:
            # Transform it to the last day of the month if needed
            if d.month == 12:
                d = d.replace(day=31)
            else:
                d = d.replace(month=d.month+1) - timedelta(days=1)
    except (ValueError, TypeError):
        # If we failed to recognize a valid date
        return ''
    else:
        # Convert it back to a string with format `%Y-%m-%d`
        return str(d)

```

オンラインでフォームウィジェットをむ <https://riptutorial.com/ja/django/topic/1230/フォームウィジェット>

33: プロジェクトの

Examples

リポジトリ>プロジェクト>サイト/

ソースの requirements と deployment tools えた Django プロジェクト。これは、Django の 2 つのスクー
プからののに基づいています。らは テンプレート をしています

```
repository/  
  docs/  
  .gitignore  
  project/  
    apps/  
      blog/  
        migrations/  
        static/ #( optional )  
          blog/  
            some.css  
        templates/ #( optional )  
          blog/  
            some.html  
        models.py  
        tests.py  
        admin.py  
        apps.py #( django 1.9 and later )  
        views.py  
      accounts/  
        #... ( same as blog )  
      search/  
        #... ( same as blog )  
    conf/  
      settings/  
        local.py  
        development.py  
        production.py  
      wsgi  
      urls.py  
    static/  
    templates/  
  deploy/  
    fabfile.py  
  requirements/  
    base.txt  
    local.txt  
  README  
  AUTHORS  
  LICENSE
```

ここで、 apps フォルダと conf フォルダには、 user created applications とプロジェクトの core configuration folder がそれぞれ user created applications。

project ディレクトリの static および templates フォルダには、 project でグローバルにされているフ

ファイルとhtml markupファイルがまれています。

また、すべてのアプリケーションフォルダのblog、accounts、searchは、staticフォルダとtemplatesフォルダがほとんどまれてstaticあります。

djangoアプリケーションのファイルとテンプレートファイルの

staticおよびtemplatesフォルダには、アプリのフォルダがまれているがありますex. blogこれはの
をするためにされるので、々はのようにファイルを/blog/base.htmlではなく/base.html 々がし
、をしているファイルのさをしします。

blogやsearchアプリケーションのtemplatesフォルダには、がbase.html ファイルがまれています。
views ファイルをすると、アプリケーションはどのファイルをレンダリングするのかします。

```
(Project Structure)
.../project/
  apps/
    blog/
      templates/
        base.html
    search/
      templates/
        base.html

(blog/views.py)
def some_func(request):
    return render(request, "/base.html")

(search/views.py)
def some_func(request):
    return render(request, "/base.html")

## After creating a folder inside /blog/templates/(blog) ##

(Project Structure)
.../project/
  apps/
    blog/
      templates/
        blog/
          base.html
    search/
      templates/
        search/
          base.html

(blog/views.py)
def some_func(request):
    return render(request, "/blog/base.html")

(search/views.py)
def some_func(request):
    return render(request, "/search/base.html")
```

オンラインでプロジェクトのをむ <https://riptutorial.com/ja/django/topic/4299/プロジェクトの>

34: ミドルウェア

き

Djangoのミドルウェアは、コードがにフックして、Djangoのまたはをするためのフレームワークです。

にめるに、ミドルウェアを`settings.py` `MIDDLEWARE_CLASSES` リストにするがあります。 Djangoがしいプロジェクトをするときにするデフォルトのリストはのとおりです

```
MIDDLEWARE_CLASSES = [  
    'django.middleware.security.SecurityMiddleware',  
    'django.contrib.sessions.middleware.SessionMiddleware',  
    'django.middleware.common.CommonMiddleware',  
    'django.middleware.csrf.CsrfViewMiddleware',  
    'django.contrib.auth.middleware.AuthenticationMiddleware',  
    'django.contrib.auth.middleware.SessionAuthenticationMiddleware',  
    'django.contrib.messages.middleware.MessageMiddleware',  
    'django.middleware.clickjacking.XFrameOptionsMiddleware',  
]
```

これらはすべて、すべてのリクエスト `views.py` ビューコードにすると `process_response` コールバックの、バージョン1.10よりになされるすべてのです。 [クロスサイトリクエストcsrf](#) トークンをするなど、さまざまなことをいます。

らかのミドルウェアがリダイレクトをうと、そののすべてのミドルウェアがしてされないため、はです。また、ミドルウェアがcsrfトークンがそこにあることをしているは、CsrfViewミドルウェアのにするCsrfViewMiddlewareます。

Examples

リクエストにデータをする

Djangoでは、ビューでののために、のデータをリクエストにするのがとてもです。たとえば、リクエストのMETAのサブドメインをし、ミドルウェアをしてののプロパティとしてアタッチすることができます。

```
class SubdomainMiddleware:  
    def process_request(self, request):  
        """  
        Parse out the subdomain from the request  
        """  
        host = request.META.get('HTTP_HOST', '')  
        host_s = host.replace('www.', '').split('.')  
        request.subdomain = None  
        if len(host_s) > 2:  
            request.subdomain = host_s[0]
```

ミドルウェアをしてデータをリクエストにすると、しくされたデータにさらにアクセスすることができます。ここでは、されたサブドメインをして、がアプリケーションにアクセスしているようなものをします。このアプローチは、すべてのインスタスが1つのインスタスをしているワイルドカードサブドメインをつDNSでされ、アプリケーションにアクセスするユーザーがアクセスポイントにするスキンプージョンをとするにです。

```
class OrganizationMiddleware:
    def process_request(self, request):
        """
        Determine the organization based on the subdomain
        """
        try:
            request.org = Organization.objects.get(domain=request.subdomain)
        except Organization.DoesNotExist:
            request.org = None
```

ミドルウェアをおいにさせるは、がであることにしてください。リクエストの、するミドルウェアをのにくことがましいでしょう。

```
MIDDLEWARE_CLASSES = [
    ...
    'myapp.middleware.SubdomainMiddleware',
    'myapp.middleware.OrganizationMiddleware',
    ...
]
```

IPアドレスでフィルタリングするミドルウェア

パス

あなたがそれをつていないは、ののアプリケーションにミドルウェアフォルダをするがあります

```
yourproject/yourapp/middleware
```

フォルダミドルウェアは、**settings.py**、**urls**、**templates** ...とじフォルダにするがあります。

アプリケーションがこのフォルダをできるように、ミドルウェアフォルダに**init.py**ファイルをすることをれないでください

ミドルウェアクラスをむのフォルダをつわりに、あなたのを1つのファイル、
`yourproject/yourapp/middleware.py`にれることも `yourproject/yourapp/middleware.py`。

Second ミドルウェアをする

これでカスタムミドルウェアのファイルをするがあります。このでは、ユーザのIPアドレスについてユーザをフィルタするミドルウェアがであるとして、**filter_ip_middleware.py**というファイルをしします。

```
#yourproject/yourapp/middleware/filter_ip_middleware.py
from django.core.exceptions import PermissionDenied

class FilterIPMiddleware(object):
    # Check if client IP address is allowed
    def process_request(self, request):
        allowed_ips = ['192.168.1.1', '123.123.123.123', etc...] # Authorized ip's
        ip = request.META.get('REMOTE_ADDR') # Get client IP address
        if ip not in allowed_ips:
            raise PermissionDenied # If user is not allowed raise Error

        # If IP address is allowed we don't do anything
        return None
```

3 ミドルウェアを 'settings.py' にする

settings.py ので `MIDDLEWARE_CLASSES` をし、ミドルウェアをするがあります のにしてください。それはのようすべきです

```
MIDDLEWARE_CLASSES = (
    'django.middleware.common.CommonMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    # Above are Django standard middlewares

    # Now we add here our custom middleware
    'yourapp.middleware.filter_ip_middleware.FilterIPMiddleware'
)
```

すべてのクライアントからのすべてのリクエストがカスタムミドルウェアをびし、カスタムコードをします。

グローバルにする

をしたクライアントにのがないときに、データベースのオブジェクトをしようとするみをするロジックをしたとします。そのようなは、カスタム `ConfictError(detailed_message)` ます。

このエラーがしたときに、 [HTTP 409Confict](#) ステータスコードをすようにします。、このをさせるのあるビューではなく、ミドルウェアとしてこれをします。

```
class ConfictErrorHandlingMiddleware:
    def process_exception(self, request, exception):
        if not isinstance(exception, ConflictError):
            return # Propagate other exceptions, we only handle ConflictError
        context = dict(confict_details=str(exception))
        return TemplateResponse(request, '409.html', context, status=409)
```

Django 1.10 ミドルウェアのしいスタイルをする

Django 1.10 では、 `process_request` と `process_response` がマージされたしいミドルウェアスタイル

がされました。

この新しいスタイルでは、ミドルウェアはのびしコードをすびしコードです。、はミドルウェアファクトリであり、はのミドルウェアです。

ミドルウェア・ファクトリは、ミドルウェア・スタックののミドルウェア、またはスタックののするとビューをのとしてります。

ミドルウェアはをのとしてけり、に**HttpResponse**します。

新しいスタイルのミドルウェアがどのようにするかをすもいは、おそらくのあるミドルウェアをするをすことです。

```
class MyMiddleware:

    def __init__(self, next_layer=None):
        """We allow next_layer to be None because old-style middlewares
        won't accept any argument.
        """
        self.get_response = next_layer

    def process_request(self, request):
        """Let's handle old-style request processing here, as usual."""
        # Do something with request
        # Probably return None
        # Or return an HttpResponse in some cases

    def process_response(self, request, response):
        """Let's handle old-style response processing here, as usual."""
        # Do something with response, possibly using request.
        return response

    def __call__(self, request):
        """Handle new-style middleware here."""
        response = self.process_request(request)
        if response is None:
            # If process_request returned None, we must call the next middleware or
            # the view. Note that here, we are sure that self.get_response is not
            # None because this method is executed only in new-style middlewares.
            response = self.get_response(request)
        response = self.process_response(request, response)
        return response
```

オンラインでミドルウェアをむ <https://riptutorial.com/ja/django/topic/1721/>ミドルウェア

35: メタドキュメンテーションのガイドライン

これはDjangoのPythonの "MetaDocumentation Guidelines"です。

これらはなるであり、ではありません。しないか、いたいことがにあるは、ここでかをにしてください。

Examples

サポートされていないバージョンにはにするはありません

サポートされていないバージョンのDjangoをですることはほとんどありません。もしかがそうしたとしても、そのバージョンにがするかどうかをすることはのであるにありません。

をすると、サポートされていないバージョンのについてすることはです。

1.6

こののブロックは、のがDjango <1.6をしないのでにたない。

1.8

こののブロックは、のがDjango <1.8をしないのでにたない。

これはトピックにもされます。このをいているで、サポートされているバージョンのクラスベースのビューのは1.3-1.9です。これは、にはAll versionsとなすことができAll versions。これにより、しいバージョンがリリースされるたびに、サポートされるすべてのトピックをアップグレードすることもされます。

サポートされているバージョンはのとおりです 1.8 ¹ 1.9 ² 1.10 ¹

1. セキュリティ、データのバグ、バグのクラッシュ、しくされたのな、いバージョンのDjangoのなどがあります。
2. セキュリティおよびデータのバグ。

オンラインでメタドキュメンテーションのガイドラインをむ

<https://riptutorial.com/ja/django/topic/5243/メタ-ドキュメンテーションのガイドライン>

36: モデル

き

なケースでは、モデルはのデータベーステーブルにマップされるPythonクラスです。クラスのは、のにマップされ、クラスのインスタンスは、データベースのをします。モデルは `django.db.models.Model` からされ、データベースのをおよびフィルタリングするためのAPIをします。

のモデルをする

Examples

のモデルをする

モデルは、アプリケーションのサブディレクトリの `models.py` ファイルにされています。
`django.db.models` モジュールの `Model` クラスは、モデルをするのにしたクラスです。え

```
from django.db import models

class Book(models.Model):
    title = models.CharField(max_length=100)
    author = models.ForeignKey('Author', on_delete=models.CASCADE,
related_name='authored_books')
    publish_date = models.DateField(null=True, blank=True)

    def __str__(self): # __unicode__ in python 2.*
        return self.title
```

モデルのは、データベースのをします。

- `title` は100のテキストです
- `author` はのモデル/テーブルこのは `Author` とのをす `ForeignKey` ですとしてのみされます。
`on_delete` は、するオブジェクト `Author` がされたに、データベースでオブジェクトの `on_delete` します。 `django 1.9` `on_delete` は2のとしてできるので、 `django 2` ではのであり、ちにそれをうことをおめします。いバージョンでは、デフォルトの `CASCADE` ます。
- `publish_date` はをします。 `null` と `blank` が `True` にされて、フィールドではないことをしますつまり、でしたり、にしておくことができます。

とに `__str__` メソッドをすると、これは、デフォルトではなく、にじて `string` としてされるのタイトルをします。

をデータベースにする

しいモデルをしたり、のモデルをしたりしたら、のためにマイグレーションをし、したデータベ

ースにマイグレーションをするがあります。これは、Djangoにみまれているシステムをしてうことができます。プロジェクトのルートディレクトリにあるときにmanage.pyユーティリティをする

```
python manage.py makemigrations <appname>
```

のコマンドは、アプリケーションのmigrationsサブディレクトリになスクリプトをします。
<appname>パラメータをすると、 settings.pyのINSTALLED_APPSにされているすべてのアプリケーションがされます。なは、をできます。

にをせずになをするには、--dry-runオプションをします。

```
python manage.py makemigrations --dry-run
```

をするには

```
python manage.py migrate <appname>
```

のコマンドは、のでされたスクリプトをし、にデータベースをします。

のデータベースのモデルがされた、なをうには、のコマンドがです。

```
python manage.py migrate --run-syncdb
```

Djangoはデフォルトで<appname>_<classname>というのテーブルをします。いつかあなたはそれをいたくないのです。デフォルトのをするは、 Metaクラスのdb_tableをしてテーブルをアナウンスすることができます。

```
from django.db import models

class YourModel(models.Model):
    parms = models.CharField()
    class Meta:
        db_table = "custom_table_name"
```

のによってされるSQLコードをするには、のコマンドをします。

```
python manage.py sqlmigrate <app_label> <migration_number>
```

Django> 1.10

しいmakemigrations --checkオプションは、マイグレーションのないモデルがされたときにコマンドがゼロのできるようにします。

のについては、「 」をしてください。

をつモデルの

1の

```

from django.db import models

class Author(models.Model):
    name = models.CharField(max_length=50)

#Book has a foreignkey (many to one) relationship with author
class Book(models.Model):
    author = models.ForeignKey(Author, on_delete=models.CASCADE)
    publish_date = models.DateField()

```

もなオプション。をしたいであればどこでもできます

の

```

class Topping(models.Model):
    name = models.CharField(max_length=50)

# One pizza can have many toppings and same topping can be on many pizzas
class Pizza(models.Model):
    name = models.CharField(max_length=50)
    toppings = models.ManyToManyField(Topping)

```

には、これはのテーブルをしてされます。そして、 `ManyToManyField` は、フォームでされるモデルにくがあります。えは `Appointment` あります `ManyToManyField` とばれる `Customer`、 `Pizza` あり `Toppings` ようにしています。

Through クラスをしたの

```

class Service(models.Model):
    name = models.CharField(max_length=35)

class Client(models.Model):
    name = models.CharField(max_length=35)
    age = models.IntegerField()
    services = models.ManyToManyField(Service, through='Subscription')

class Subscription(models.Model):
    client = models.ForeignKey(Client)
    service = models.ForeignKey(Service)
    subscription_type = models.CharField(max_length=1, choices=SUBSCRIPTION_TYPES)
    created_at = models.DateTimeField(default=timezone.now)

```

このようにして、には2つのエンティティのについてよりくのメタデータをできます。されるように、クライアントは、いくつかのサブスクリプションタイプをしてのサービスにサブスクライブすることができます。このののいは、それがM2Mにしいインスタンスをすることで、がショートカットはできません `pizza.toppings.add(topping)` のわりに、 クラスのしいオブジェクトをしなければならないが、 `Subscription.objects.create(client=client, service=service, subscription_type='p')`

のでは `through tables` は `JoinColumn`、 `Intersection table` または `mapping table`

11の


```
class Employee(models.Model):
    name = models.CharField(max_length=50)
    age = models.IntegerField()
    spouse = models.OneToOneField(Spouse)

class Spouse(models.Model):
    name = models.CharField(max_length=50)
```

これらのフィールドは、2つのモデルののみをつにします。

な Django DB クエリ

Django ORMはなで、SQLクエリーをですることなく、データベースからデータをしてりすことができます。

のモデルをしましょう

```
class Author(models.Model):
    name = models.CharField(max_length=50)

class Book(models.Model):
    name = models.CharField(max_length=50)
    author = models.ForeignKey(Author)
```

のコードをdjangoアプリケーションにし、 `migrate` コマンドをしてデータベースがされるようにして Djangoシェルをする

```
python manage.py shell
```

これは、のPythonシェルをしますが、するDjangoライブラリをインポートすることで、なにをすることができます。

にしたモデルをインポートすることからめますこれはファイル `models.py` われているとしています

```
from .models import Book, Author
```

のクエリをします。

```
>>> Author.objects.all()
[]
>>> Book.objects.all()
[]
```

とブックオブジェクトをできます

```
>>> hawking = Author(name="Stephen hawking")
>>> hawking.save()
>>> history_of_time = Book(name="history of time", author=hawking)
>>> history_of_time.save()
```

または`create`をしてモデルオブジェクトをし、1のコードです

```
>>> wings_of_fire = Book.objects.create(name="Wings of Fire", author="APJ Abdul Kalam")
```

これでクエリをできるようになりました

```
>>> Book.objects.all()
[<Book: Book object>]
>>> book = Book.objects.first() #getting the first book object
>>> book.name
u'history of time'
```

クエリに`where`をしましょう

```
>>> Book.objects.filter(name='nothing')
[]
>>> Author.objects.filter(name__startswith='Ste')
[<Author: Author object>]
```

ののについてのをるには

```
>>> book = Book.objects.first() #getting the first book object
>>> book.author.name # lookup on related model
u'Stephen hawking'
```

Stephen Hawkingのルックアップブックがしたすべてのをするには、

```
>>> hawking.book_set.all()
[<Book: Book object>]
```

`_set`は、きにされるです。つまり、フィールドがBookモデルにある`book_set`に、`author`オブジェクトの`book_set`をしてすべてのができます。

なアンマネージテーブル。

Djangoをしているあるで、すでにされたテーブルやデータベースビューとしたいとかかもしれません。このような、Djangoはによってテーブルをするはありません。これをするには、モデルのMetaクラスに`managed = False`を1つだけするがあります。

に、データベースビューとするモデルをするのをします。

```
class Dummy(models.Model):
    something = models.IntegerField()

    class Meta:
        managed = False
```

これは、のようにSQLでされたビューにマップできます。

```
CREATE VIEW myapp_dummy AS
SELECT id, something FROM complicated_table
WHERE some_complicated_condition = True
```

このモデルをしたら、のモデルとじようにできます。

```
>>> Dummy.objects.all()
[<Dummy: Dummy object>, <Dummy: Dummy object>, <Dummy: Dummy object>]
>>> Dummy.objects.filter(something=42)
[<Dummy: Dummy object>]
```

なモデル

モデルは、オブジェクトにするデータだけでなく、よりくのをすることができます。をて、それがのにつのかをてみましょう

```
from django.db import models
from django.urls import reverse
from django.utils.encoding import python_2_unicode_compatible

@python_2_unicode_compatible
class Book(models.Model):
    slug = models.SlugField()
    title = models.CharField(max_length=128)
    publish_date = models.DateField()

    def get_absolute_url(self):
        return reverse('library:book', kwargs={'pk':self.pk})

    def __str__(self):
        return self.title

    class Meta:
        ordering = ['publish_date', 'title']
```

キー

`get_absolute_url` メソッドで `self.pk` をしていることにくかもしれません。 `pk` フィールドは、モデルのキーへのエイリアスです。また、もし `django` がつからなければ、にプライマリキーをします。これはすることがなく、のモデルにキーをしてにできるようにします。

URL

されているのは `get_absolute_url` です。あなたがをっているなら、これで、あなたは URL タグ、などをらずにリンクをすることができます。に `book.get_absolute_url` をびすと、しいリンクがられます。ボーナスとして、`django` のあなたのオブジェクトは、ボタン "サイトのビュー" をします。

`__str__` メソッドをすると、オブジェクトをするがあるときにそのオブジェクトをできます。たとえば、のでは、テンプレートのブックへのリンクをすることは、`<a href="{`

`book.get_absolute_url }}`"}>{{ book }}です。ポイントまでまっすぐ。このメソッドは、のドロップダウンにされているものえは、キーなどもします。

クラスデコレータでは、python 2で`__str__`と`__unicode__`メソッドをできますが、python 3でははしません。のバージョンでアプリケーションをすることがされるは、そのです。

スラッグフィールド

スラッグフィールドは`char`フィールドにいますが、シンボルはないです。デフォルトでは、`_`、アンダースコアまたはハイフンのみです。たとえば、`url`のようならしいをとってオブジェクトをしたいにです。

Meta クラス

`Meta`クラスでは、アイテムのコレクションについてさらにくのをできます。ここでは、ののみがされています。えは、`ListView`オブジェクトです。ソートにするフィールドのリストがです。ここでは、はによってにソートされ、がじはタイトルによってソートされます。

そののは`verbose_name`と`verbose_name_plural`です。デフォルトでは、モデルのからされ、うまくいくはずです。しかし、はなる`"s"`をにけるだけで、によってはにしたいかもしれません。

モデルオブジェクトがフェッチされると、にされたクラスのインスタンスになります。そのため、フォームやシリアライザDjango Rest Frameworkなどでのメソッドにアクセスできます。

Pythonプロパティをすると、さまざまなによってデータベースにされないのをすためのエレガントなです。

```
def expire():
    return timezone.now() + timezone.timedelta(days=7)

class Coupon(models.Model):
    expiration_date = models.DateField(default=expire)

    @property
    def is_expired(self):
        return timezone.now() > self.expiration_date
```

ほとんどの、クエリセットのでデータをうことができますが、モデルプロパティとしてののは、クエリのでにできないにはです。

さらに、プロパティはPythonクラスでされており、スキーマのとしてはされていないため、にはできません。

モデルのをする

モデルオブジェクトのがめるプレゼンテーションをするには、`Model.__str__()`メソッドまたは

Python2では`Model.__unicode__()`をするがあり`Model.__str__()`。このメソッドは、モデルのインスタンスで`str()`をびすたびにびされ`str()`モデルがテンプレートでされるなど。ここにがあります

1. ブックモデルをします。

```
# your_app/models.py

from django.db import models

class Book(models.Model):
    name = models.CharField(max_length=50)
    author = models.CharField(max_length=50)
```

2. モデルのインスタンスをし、データベースにします。

```
>>> himu_book = Book(name='Himu Mama', author='Humayun Ahmed')
>>> himu_book.save()
```

3. インスタンスにして`print()`をします。

```
>>> print(himu_book)
<Book: Book object>
```

<BookBookオブジェクト>は、デフォルトのであり、たちのけにはならない。これをするには、`__str__`メソッドをしましょう。

```
from django.utils.encoding import python_2_unicode_compatible

@python_2_unicode_compatible
class Book(models.Model):
    name = models.CharField(max_length=50)
    author = models.CharField(max_length=50)

    def __str__(self):
        return '{} by {}'.format(self.name, self.author)
```

`python_2_unicode_compatible`デコレータはあなたのコードは、このデコレータコピーのpython 2とをつようにしたいにのみです`__str__`する`__unicode__`を。 `django.utils.encoding`からインポートし `django.utils.encoding`。

`print`をbookインスタンスとびます

```
>>> print(himu_book)
Himu Mama by Humayun Ahmed
```

ずっといい

は、モデルが`ForeignKeyField`および`ManyToManyField`フィールドの`ModelForm`でされるにもされます。

モデルミックスイン

じケースでは、なるモデルがのライフサイクルでじフィールドとじをつことができます。コードのをたずにこれらのをうことができます。クラスをするわりに、 **mixin**のデザインパターンは、いくつかのメソッドとをするまたは、いくつかのことをべる ことができます。をてみましょう

```
class PostableMixin(models.Model):
    class Meta:
        abstract=True

    sender_name = models.CharField(max_length=128)
    sender_address = models.CharField(max_length=255)
    receiver_name = models.CharField(max_length=128)
    receiver_address = models.CharField(max_length=255)
    post_datetime = models.DateTimeField(auto_now_add=True)
    delivery_datetime = models.DateTimeField(null=True)
    notes = models.TextField(max_length=500)

class Envelope(PostableMixin):
    ENVELOPE_COMMERCIAL = 1
    ENVELOPE_BOOKLET = 2
    ENVELOPE_CATALOG = 3

    ENVELOPE_TYPES = (
        (ENVELOPE_COMMERCIAL, 'Commercial'),
        (ENVELOPE_BOOKLET, 'Booklet'),
        (ENVELOPE_CATALOG, 'Catalog'),
    )

    envelope_type = models.PositiveSmallIntegerField(choices=ENVELOPE_TYPES)

class Package(PostableMixin):
    weight = models.DecimalField(max_digits=6, decimal_places=2)
    width = models.DecimalField(max_digits=5, decimal_places=2)
    height = models.DecimalField(max_digits=5, decimal_places=2)
    depth = models.DecimalField(max_digits=5, decimal_places=2)
```

モデルをクラスにするには、のMetaクラスでabstract=Trueとするがあります。 Djangoはモデルのテーブルをデータベースにしません。ただし、 EnvelopeとPackageのモデルでは、するテーブルがデータベースにされます。

さらに、のモデルではいくつかのモデルがになります。したがって、コードのりしをぐために、これらのメソッドをミックスインにすることができます。たとえば、 PostableMixinにをするメソッドをすると、そののからアクセスできます。

```
class PostableMixin(models.Model):
    class Meta:
        abstract=True

    ...
    ...

    def set_delivery_datetime(self, dt=None):
        if dt is None:
            from django.utils.timezone import now
            dt = now()

        self.delivery_datetime = dt
```

```
self.save()
```

このメソッドは、どもにしてのようになります。

```
>> envelope = Envelope.objects.get(pk=1)
>> envelope.set_delivery_datetime()

>> pack = Package.objects.get(pk=1)
>> pack.set_delivery_datetime()
```

UUIDキー

デフォルトでは、モデルはインクリメントキーをします。これは、あなたにのキー1,2,3をえるでしょう。

モデルにさなをえて、モデルになるキータイプをできます。

UUIDはユニバーサルユニークなです。これは32のランダムでIDとしてできます。 PostgreSQLですると、uuidデータ、それのはchar32にされます。

```
import uuid
from django.db import models

class ModelUsingUUID(models.Model):
    id = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
```

されたキーは7778c552-73fc-4bc4-8bf9-5a2f6f7b7f47になります

モデルのは2つのことができます。

- のクラス「モデルミックスイン」のを
- のテーブルをつのモデル

マルチテーブルのは、フィールドに1つのテーブルをし、モデルごとに1つのテーブルをします。

```
from django.db import models

class Place(models.Model):
    name = models.CharField(max_length=50)
    address = models.CharField(max_length=80)

class Restaurant(Place):
    serves_hot_dogs = models.BooleanField(default=False)
    serves_pizza = models.BooleanField(default=False)
```

2つのテーブルをします.1つはPlace、もう1つはRestaurantで、OneToOneフィールドをにしてフィールドにPlaceします。

これは、レストランオブジェクトをするたびにplacesテーブルにのクエリがになることにしてください。

オンラインでモデルをむ <https://riptutorial.com/ja/django/topic/888/モデル>

37: モデルフィールドリファレンス

パラメーター

パラメータ	
ヌル	trueの、のはデータベースにnullとしてされます。
ブランク	trueの、フィールドはフォームではありません。フィールドがの、Djangoはデフォルトのフィールドをします。
	このフィールドのとして2iterableをにできます。されている、フィールドはのドロップダウンとしてレンダリングされます。 [('m', 'Male'), ('f', 'Female'), ('z', 'Prefer Not to Disclose')] オプションをグループするには、 [('Video Source', ((1, 'YouTube'), (2, 'Facebook'))), ('Audio Source', ((3, 'Soundcloud'), (4, 'Spotify')))]
db_column	デフォルトでは、djangoはデータベースカラムのフィールドをします。これをしてカスタムをする
db_index	True、データベースのこのフィールドにインデックスがされます
db_tablespace	このフィールドのインデックスにするテーブルスペース。 このフィールドは、データベースエンジンがサポートしているにのみされ、そうでないはされます。
デフォルト	このフィールドのデフォルト。、またはびしなオブジェクトにすることができます。なデフォルトリスト、セット、ディクショナリのは、びしをするがあります。とののため、ラムダをすることはできません。
な	False、フィールドはモデルまたはModelFormされません。デフォルトはTrueです。
error_messages	このフィールドにされるデフォルトのエラーメッセージをカスタマイズするためにします。はで、キーはエラーをし、はメッセージです。デフォルトのキーエラーメッセージはnull、blank、invalid、invalid_choice、uniqueおよびunique_for_dateです。のエラーメッセージは、カスタムフィールドによってされることがあります。
ヘルプテキスト	フィールドをするテキスト。ユーザーをします。HTMLはされます。
on_delete	ForeignKeyでされるオブジェクトがされると、Djangoはon_deleteでされたSQLのをエミュレートします。これは、ForeignKeyとOneToOneFieldのフィールドの2のです。のフィールドにはこのはありません。

パラメータ	
primary_key	True、このフィールドはキーになります。 Djangoはキーをにします。カスタムプライマリーキーをするにのみです。モデルごとに1つのプライマリーキーしかてません。
ユニークな	True、このフィールドにしたをするとエラーがします。これはデータベースレベルのであり、なるユーザーインターフェイスブロックではありません。
unique_for_date	をDateFieldまたはDateTimeFieldにすると、じまたはのにするがあるとエラーがします。
unique_for_month	unique_for_date にていますが、チェックはそのにされています。
unique_for_year	unique_for_date と、はにされます。
verbose_name	さまざまなでdjangoがする、フィールドのフレンドリやモデルのフォームにラベルをするなど。
バリデーター	このフィールドのバリデータのリスト。

- あなたがとじたらあなたのフィールドをくことができます
- モデルクラスのをオーバーライドできます。もなのはsave()です

Examples

フィールド

フィールドのをにします。

フィールド

にキーにされるインクリメント。

```
from django.db import models

class MyModel(models.Model):
    pk = models.AutoField()
```

モデルは、デフォルトでプライマリーフィールド id とばれますをします。したがって、プライマリーのでモデルにidフィールドをするはありません。

BigIntegerField

フィッティング-9223372036854775808に 9223372036854775807 8 Bytes。

```
from django.db import models

class MyModel(models.Model):
    number_of_seconds = models.BigIntegerField()
```

IntegerField

IntegerFieldは、-2147483648から2147483647 4 Bytes までのをするためにされます。

```
from django.db import models

class Food(models.Model):
    name = models.CharField(max_length=255)
    calorie = models.IntegerField(default=0)
```

defaultパラメータはではありません。しかし、デフォルトをしますとです。

PositiveIntegerField

IntegerFieldとていますが、またはゼロ0であるがあります。 PositiveIntegerFieldは、0 2147483647 4 Bytes のをするためにされます。これは、にでなければならぬとなり。例えば、カロリーでべをしている、それはであってはなりません。このフィールドは、によってのをします。

```
from django.db import models

class Food(models.Model):
    name = models.CharField(max_length=255)
    calorie = models.PositiveIntegerField(default=0)
```

defaultパラメータはではありません。しかし、デフォルトをしますとです。

SmallIntegerField

SmallIntegerFieldは、-3276832767 2 Bytes のをするためにされます。このフィールドは、ではないにちます。

```
from django.db import models

class Place(models.Model):
    name = models.CharField(max_length=255)
    temperature = models.SmallIntegerField(null=True)
```

PositiveSmallIntegerField

SmallIntegerFieldは、032767 2 Bytes のをためにされます。 SmallIntegerFieldとじように、このフィールドはがくなく、にのにするのにです。例えば、それはできないをすることができます

```
from django.db import models

class Staff(models.Model):
    first_name = models.CharField(max_length=255)
    last_name = models.CharField(max_length=255)
    age = models.PositiveSmallIntegerField(null=True)
```

`PositiveSmallIntegerField`はとしてですが、これはEnumをするジャンゴのです。

```
from django.db import models
from django.utils.translation import gettext as _

APPLICATION_NEW = 1
APPLICATION_RECEIVED = 2
APPLICATION_APPROVED = 3
APPLICATION_REJECTED = 4

APPLICATION_CHOICES = (
    (APPLICATION_NEW, _('New')),
    (APPLICATION_RECEIVED, _('Received')),
    (APPLICATION_APPROVED, _('Approved')),
    (APPLICATION_REJECTED, _('Rejected')),
)

class JobApplication(models.Model):
    first_name = models.CharField(max_length=255)
    last_name = models.CharField(max_length=255)
    status = models.PositiveSmallIntegerField(
        choices=APPLICATION_CHOICES,
        default=APPLICATION_NEW
    )
    ...
```

にじてクラスまたはモジュールとしてをすることは、それらをするいです。がフレンジリーななしでフィールドにされると、がまれます。

DecimalField

の10。PythonでDecimalインスタンスでされます。IntegerFieldとそのとはなり、このフィールドには2つのがあります。

1. `DecimalField.max_digits` にされる。これは`decimal_places`でなければならないことにしてください。
2. `DecimalField.decimal_places` をするのです。

3まで99をするは、`max_digits=5`と`decimal_places=3`するがあります。

```
class Place(models.Model):
    name = models.CharField(max_length=255)
    atmospheric_pressure = models.DecimalField(max_digits=5, decimal_places=3)
```

BinaryField

これはバイナリデータのにされるなフィールドです。 バイトのみをけけます。 データはに base64シリアルされます。

バイナリデータをするので、このフィールドはフィルタでできません。

```
from django.db import models

class MyModel(models.Model):
    my_binary_data = models.BinaryField()
```

CharField

CharFieldは、されたさのテキストをするためにされます。 のでは、128のテキストをフィールドにすることができます。 これよりもいをすると、エラーがします。

```
from django.db import models

class MyModel(models.Model):
    name = models.CharField(max_length=128, blank=True)
```

DateTimeField

DateTimeFieldは、のをするためにされます。

```
class MyModel(models.Model):
    start_time = models.DateTimeField(null=True, blank=True)
    created_on = models.DateTimeField(auto_now_add=True)
    updated_on = models.DateTimeField(auto_now=True)
```

DateTimeFieldは、2つのオプションのパラメータがあります。

- `auto_now_add`は、オブジェクトのにフィールドのをのdatetimeにします。
- `auto_now`は、フィールドがされるたびにフィールドのをのdatetimeにします。

これらのオプションと `default` パラメータはにです。

キー

ForeignKeyフィールドは、モデルの `many-to-one` をするためにされます。 ほとんどののフィールドとじように、ながです。 のは、 `car` と `owner` のをしています。

```
from django.db import models

class Person(models.Model):
    GENDER_FEMALE = 'F'
    GENDER_MALE = 'M'
```

```
GENDER_CHOICES = (
    (GENDER_FEMALE, 'Female'),
    (GENDER_MALE, 'Male'),
)

first_name = models.CharField(max_length=100)
last_name = models.CharField(max_length=100)
gender = models.CharField(max_length=1, choices=GENDER_CHOICES)
age = models.SmallIntegerField()

class Car(model.Model)
    owner = models.ForeignKey('Person')
    plate = models.CharField(max_length=15)
    brand = models.CharField(max_length=50)
    model = models.CharField(max_length=50)
    color = models.CharField(max_length=50)
```

フィールドのものは、モデルがするクラスです。2のは`on_delete`です。のバージョンでは、このはではありませんが、Django 2.0ではです。のデフォルトは、のようにされます。

```
class Car(model.Model)
    owner = models.ForeignKey('Person', on_delete=models.CASCADE)
    ...
```

これにより、オーナーがPersonモデルからされたときに、モデルからCarオブジェクトがされます。これがデフォルトのです。

```
class Car(model.Model)
    owner = models.ForeignKey('Person', on_delete=models.PROTECT)
    ...
```

これにより、なくとも1つのCarオブジェクトにしているPersonオブジェクトがされなくなります。PersonオブジェクトをするCarオブジェクトはすべてにするがあります。そして、Personオブジェクトをすることができます。

オンラインでモデルフィールドリファレンスをむ <https://riptutorial.com/ja/django/topic/3686/モデルフィールドリファレンス>

38: モデル

き

は、モデルからしたオブジェクトの々のおよび/またはグループのにするのをにするメソッドです。

Examples

、、、Querysetからの

```
class Product(models.Model):
    name = models.CharField(max_length=20)
    price = models.FloatField()
```

すべてののをするには

```
>>> from django.db.models import Avg, Max, Min, Sum
>>> Product.objects.all().aggregate(Avg('price'))
# {'price__avg': 124.0}
```

すべてののをするには

```
>>> Product.objects.all().aggregate(Min('price'))
# {'price__min': 9}
```

すべてののをするには

```
>>> Product.objects.all().aggregate(Max('price'))
# {'price__max': 599 }
```

すべてののをするには

```
>>> Product.objects.all().aggregate(Sum('price'))
# {'price__sum': 92456 }
```

のをえる

```
class Category(models.Model):
    name = models.CharField(max_length=20)

class Product(models.Model):
    name = models.CharField(max_length=64)
    category = models.ForeignKey(Category, on_delete=models.PROTECT)
```

カテゴリのをするには

```
>>> categories = Category.objects.annotate(Count('product'))
```

これは、されたインスタンスに<field_name>__count をします。

```
>>> categories.values_list('name', 'product__count')
[('Clothing', 42), ('Footwear', 12), ...]
```

キーワードをして、のカスタムをできます。

```
>>> categories = Category.objects.annotate(num_products=Count('product'))
```

クエリーセットでフィールドをすることができます

```
>>> categories.order_by('num_products')
[<Category: Footwear>, <Category: Clothing>]

>>> categories.filter(num_products__gt=20)
[<Category: Clothing>]
```

GROUP BY ... COUNT / SUM Django ORM

々は、 annotate() 、 values() 、 order_by() および django.db.models の Count して、 Django ORM で GROUP BY ... COUNT または GROUP BY ... SUM SQL のクエリをできます Sum メソッドをします。

々のモデルはのようにならう

```
class Books(models.Model):
    title = models.CharField()
    author = models.CharField()
    price = models.FloatField()
```

GROUP BY ... COUNT

- Books テーブルには、のごとにいくつかのブックオブジェクトがするかをカウントしたいとします。

```
result = Books.objects.values('author')
                        .order_by('author')
                        .annotate(count=Count('author'))
```

- result は、 author と count 2つのをつクエリーセットがまわてい count 。

author	count
OneAuthor	5
OtherAuthor	2
...	...

GROUP BY ... SUM

- Books テーブルにするのすべてののをしたいとします。

```
result = Books.objects.values('author')
                    .order_by('author')
                    .annotate(total_price=Sum('price'))
```

- result は、 author と total_price 2つのをつ total_price がまれています。

author	total_price
OneAuthor	100.35
OtherAuthor	50.00
...	...

オンラインでモデルをむ <https://riptutorial.com/ja/django/topic/3775/モデル>

39: ユーザーモデルのまたは

Examples

プライマリログインフィールドとしてメールをするカスタムユーザモデル。

models.py

```
from __future__ import unicode_literals
from django.db import models
from django.contrib.auth.models import (
    AbstractBaseUser, BaseUserManager, PermissionsMixin)
from django.utils import timezone
from django.utils.translation import ugettext_lazy as _

class UserManager(BaseUserManager):
    def _create_user(self, email, password, is_staff, is_superuser, **extra_fields):
        now = timezone.now()
        if not email:
            raise ValueError('users must have an email address')
        email = self.normalize_email(email)
        user = self.model(email=email,
                           is_staff=is_staff,
                           is_superuser=is_superuser,
                           last_login=now,
                           date_joined=now,
                           **extra_fields)
        user.set_password(password)
        user.save(using=self._db)
        return user

    def create_user(self, email, password=None, **extra_fields):
        user = self._create_user(email, password, False, False, **extra_fields)
        return user

    def create_superuser(self, email, password, **extra_fields):
        user = self._create_user(email, password, True, True, **extra_fields)
        return user

class User(AbstractBaseUser, PermissionsMixin):
    """My own custom user class"""

    email = models.EmailField(max_length=255, unique=True, db_index=True,
        verbose_name=_('email address'))
    date_joined = models.DateTimeField(auto_now_add=True)
    is_active = models.BooleanField(default=True)
    is_staff = models.BooleanField(default=False)

    objects = UserManager()

    USERNAME_FIELD = 'email'
    REQUIRED_FIELDS = []

    class Meta:
        verbose_name = _('user')
```

```

        verbose_name_plural = _('users')

    def get_full_name(self):
        """Return the email."""
        return self.email

    def get_short_name(self):
        """Return the email."""
        return self.email

```

forms.py

```

from django import forms
from django.contrib.auth.forms import UserCreationForm
from .models import User

class RegistrationForm(UserCreationForm):
    email = forms.EmailField(widget=forms.TextInput(
        attrs={'class': 'form-control', 'type': 'text', 'name': 'email'}),
        label="Email")
    password1 = forms.CharField(widget=forms.PasswordInput(
        attrs={'class': 'form-control', 'type': 'password', 'name': 'password1'}),
        label="Password")
    password2 = forms.CharField(widget=forms.PasswordInput(
        attrs={'class': 'form-control', 'type': 'password', 'name': 'password2'}),
        label="Password (again)")

    '''added attributes so as to customise for styling, like bootstrap'''
    class Meta:
        model = User
        fields = ['email', 'password1', 'password2']
        field_order = ['email', 'password1', 'password2']

    def clean(self):
        """
        Verifies that the values entered into the password fields match
        NOTE : errors here will appear in 'non_field_errors()'
        """
        cleaned_data = super(RegistrationForm, self).clean()
        if 'password1' in self.cleaned_data and 'password2' in self.cleaned_data:
            if self.cleaned_data['password1'] != self.cleaned_data['password2']:
                raise forms.ValidationError("Passwords don't match. Please try again!")
        return self.cleaned_data

    def save(self, commit=True):
        user = super(RegistrationForm, self).save(commit=False)
        user.set_password(self.cleaned_data['password1'])
        if commit:
            user.save()
        return user

#The save(commit=False) tells Django to save the new record, but dont commit it to the
database yet

class AuthenticationForm(forms.Form): # Note: forms.Form NOT forms.ModelForm
    email = forms.EmailField(widget=forms.TextInput(
        attrs={'class': 'form-control', 'type': 'text', 'name': 'email', 'placeholder': 'Email'}),
        label='Email')
    password = forms.CharField(widget=forms.PasswordInput(

```

```

        attrs={'class':'form-control','type':'password', 'name':
'password','placeholder':'Password'})),
        label='Password')

class Meta:
    fields = ['email', 'password']

```

views.py

```

from django.shortcuts import redirect, render, HttpResponseRedirect
from django.contrib.auth import login as django_login, logout as django_logout, authenticate
as django_authenticate
#importing as such so that it doesn't create a confusion with our methods and django's default
methods

from django.contrib.auth.decorators import login_required
from .forms import AuthenticationForm, RegistrationForm

def login(request):
    if request.method == 'POST':
        form = AuthenticationForm(data = request.POST)
        if form.is_valid():
            email = request.POST['email']
            password = request.POST['password']
            user = django_authenticate(email=email, password=password)
            if user is not None:
                if user.is_active:
                    django_login(request,user)
                    return redirect('/dashboard') #user is redirected to dashboard
        else:
            form = AuthenticationForm()

    return render(request,'login.html',{'form':form,})

def register(request):
    if request.method == 'POST':
        form = RegistrationForm(data = request.POST)
        if form.is_valid():
            user = form.save()
            u = django_authenticate(user.email = user, user.password = password)
            django_login(request,u)
            return redirect('/dashboard')
    else:
        form = RegistrationForm()

    return render(request,'register.html',{'form':form,})

def logout(request):
    django_logout(request)
    return redirect('/')

@login_required(login_url ="/")
def dashboard(request):
    return render(request, 'dashboard.html',{})

```

settings.py

```

AUTH_USER_MODEL = 'myapp.User'

```

```

from django.contrib import admin
from django.contrib.auth.admin import UserAdmin as BaseUserAdmin
from django.contrib.auth.models import Group
from .models import User

class UserAdmin(BaseUserAdmin):
    list_display = ('email', 'is_staff')
    list_filter = ('is_staff',)
    fieldsets = ((None,
                  {'fields': ('email', 'password')}), ('Permissions', {'fields': ('is_staff',)})),
    add_fieldsets = ((None, {'classes': ('wide',), 'fields': ('email', 'password1',
'password2')})),)
    search_fields = ('email',)
    ordering = ('email',)
    filter_horizontal = ()

admin.site.register(User, UserAdmin)
admin.site.unregister(Group)

```

usernameとして`email`をい、`username`フィールドをしてください

usernameフィールドをりき、emailをのユーザとしてするは、AbstractBaseUserわりにAbstractUserしたカスタムUserモデルをするがあります。、usernameとemailはAbstractUserでされており、それらをきすることはできません。つまり、AbstractUserされたすべてのフィールドをするがあります。

```

from django.contrib.auth.models import (
    AbstractBaseUser, PermissionsMixin, BaseUserManager,
)
from django.db import models
from django.utils import timezone
from django.utils.translation import gettext_lazy as _

class UserManager(BaseUserManager):

    use_in_migrations = True

    def _create_user(self, email, password, **extra_fields):
        if not email:
            raise ValueError('The given email must be set')
        email = self.normalize_email(email)
        user = self.model(email=email, **extra_fields)
        user.set_password(password)
        user.save(using=self._db)
        return user

    def create_user(self, email, password=None, **extra_fields):
        extra_fields.setdefault('is_staff', False)
        extra_fields.setdefault('is_superuser', False)
        return self._create_user(email, password, **extra_fields)

    def create_superuser(self, email, password, **extra_fields):
        extra_fields.setdefault('is_staff', True)
        extra_fields.setdefault('is_superuser', True)

```

```

        if extra_fields.get('is_staff') is not True:
            raise ValueError('Superuser must have is_staff=True.')
        if extra_fields.get('is_superuser') is not True:
            raise ValueError('Superuser must have is_superuser=True.')

    return self._create_user(email, password, **extra_fields)

class User(AbstractBaseUser, PermissionsMixin):
    """PermissionsMixin contains the following fields:
        - `is_superuser`
        - `groups`
        - `user_permissions`
    You can omit this mix-in if you don't want to use permissions or
    if you want to implement your own permissions logic.
    """

    class Meta:
        verbose_name = _("user")
        verbose_name_plural = _("users")
        db_table = 'auth_user'
        # `db_table` is only needed if you move from the existing default
        # User model to a custom one. This enables to keep the existing data.

    USERNAME_FIELD = 'email'
    """Use the email as unique username."""

    REQUIRED_FIELDS = ['first_name', 'last_name']

    GENDER_MALE = 'M'
    GENDER_FEMALE = 'F'
    GENDER_CHOICES = [
        (GENDER_MALE, _("Male")),
        (GENDER_FEMALE, _("Female")),
    ]

    email = models.EmailField(
        verbose_name=_("email address"), unique=True,
        error_messages={
            'unique': _(
                "A user is already registered with this email address"),
        },
    )
    gender = models.CharField(
        max_length=1, blank=True, choices=GENDER_CHOICES,
        verbose_name=_("gender"),
    )
    first_name = models.CharField(
        max_length=30, verbose_name=_("first name"),
    )
    last_name = models.CharField(
        max_length=30, verbose_name=_("last name"),
    )
    is_staff = models.BooleanField(
        verbose_name=_("staff status"),
        default=False,
        help_text=_(
            "Designates whether the user can log into this admin site."
        ),
    )

```

```

is_active = models.BooleanField(
    verbose_name=_("active"),
    default=True,
    help_text=_(
        "Designates whether this user should be treated as active. "
        "Unselect this instead of deleting accounts."
    ),
)
date_joined = models.DateTimeField(
    verbose_name=_("date joined"), default=timezone.now,
)

objects = UserManager()

```

Django ユーザモデルをににする

たちの `UserProfile` クラス

`OneToOne` とデフォルトの `User` モデルのを `UserProfile` モデルクラスをします。

```

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

class UserProfile(models.Model):
    user = models.OneToOneField(User, related_name='user')
    photo = FileField(verbose_name=_("Profile Picture"),
                      upload_to=upload_to("main.UserProfile.photo", "profiles"),
                      format="Image", max_length=255, null=True, blank=True)
    website = models.URLField(default='', blank=True)
    bio = models.TextField(default='', blank=True)
    phone = models.CharField(max_length=20, blank=True, default='')
    city = models.CharField(max_length=100, default='', blank=True)
    country = models.CharField(max_length=100, default='', blank=True)
    organization = models.CharField(max_length=100, default='', blank=True)

```

Django のシグナル

Django Signals をして、ちにしい `UserProfile` し、 `User` オブジェクトをします。これは、じファイルの `UserProfile` モデルクラスのに `UserProfile` か、きなにくことができます。あなたがしくそれをするのとじように、はにしません。

```

def create_profile(sender, **kwargs):
    user = kwargs["instance"]
    if kwargs["created"]:
        user_profile = UserProfile(user=user)
        user_profile.save()
post_save.connect(create_profile, sender=User)

```

`inlineformset_factory` への `inlineformset_factory`

あなたの `views.py` は、のようなものがあります

```

from django.shortcuts import render, HttpResponseRedirect

```

```

from django.contrib.auth.decorators import login_required
from django.contrib.auth.models import User
from .models import UserProfile
from .forms import UserForm
from django.forms.models import inlineformset_factory
from django.core.exceptions import PermissionDenied
@login_required() # only logged in users should access this
def edit_user(request, pk):
    # querying the User object with pk from url
    user = User.objects.get(pk=pk)

    # prepopulate UserProfileForm with retrieved user values from above.
    user_form = UserForm(instance=user)

    # The sorcery begins from here, see explanation https://blog.khophi.co/extending-django-
    user-model-userprofile-like-a-pro/
    ProfileInlineFormset = inlineformset_factory(User, UserProfile, fields=('website', 'bio',
    'phone', 'city', 'country', 'organization'))
    formset = ProfileInlineFormset(instance=user)

    if request.user.is_authenticated() and request.user.id == user.id:
        if request.method == "POST":
            user_form = UserForm(request.POST, request.FILES, instance=user)
            formset = ProfileInlineFormset(request.POST, request.FILES, instance=user)

            if user_form.is_valid():
                created_user = user_form.save(commit=False)
                formset = ProfileInlineFormset(request.POST, request.FILES,
instance=created_user)

                if formset.is_valid():
                    created_user.save()
                    formset.save()
                    return HttpResponseRedirect('/accounts/profile/')

            return render(request, "account/account_update.html", {
                "noodle": pk,
                "noodle_form": user_form,
                "formset": formset,
            })
        else:
            raise PermissionDenied

```

たちのテンプレート

それから、あなたのテンプレート `account_update.html` すべてをきます

```

{% load material_form %}
<!-- Material form is just a materialize thing for django forms -->
<div class="col s12 m8 offset-m2">
    <div class="card">
        <div class="card-content">
            <h2 class="flow-text">Update your information</h2>
            <form action="." method="POST" class="padding">
                {% csrf_token %} {{ noodle_form.as_p }}
                <div class="divider"></div>
                {{ formset.management_form }}
                {{ formset.as_p }}
                <button type="submit" class="btn-floating btn-large waves-light waves-effect"><i

```



```

class="large material-icons">done</i></button>
    <a href="#" onclick="window.history.back(); return false;" title="Cancel"
class="btn-floating waves-effect waves-light red"><i class="material-icons">history</i></a>

    </form>
</div>
</div>
</div>

```

Django UserProfileをProのようにしたスニペットの

カスタムユーザモデルの

Djangoのみみ`User`モデルは、あるのプロジェクトにはずしもしていません。いくつかのサイトで、たとえばユーザーのわりにメールアドレスをするほうがあります。

カスタマイズした`User`モデルをプロジェクトファイルの`AUTH_USER_MODEL`にすると、デフォルトの`User`モデルをきできます。

```
AUTH_USER_MODEL = 'myapp.MyUser'
```

にする`advised`だということにしてください `AUTH_USER_MODEL`のをまたはするに、 `manage.py migrate`めて。 Djangoのののために。

たとえば、ブログでは、のがのユーザーではなくメールアドレスでログインできるようにするため、メールアドレスを`USERNAME_FIELD`としてカスタムの`User`モデルをします。

```

from django.contrib.auth.models import AbstractBaseUser

class CustomUser(AbstractBaseUser):
    email = models.EmailField(unique=True)

    USERNAME_FIELD = 'email'

```

`AbstractBaseUser`をすることにより、した`User`モデルをすることができ`User`。 `AbstractBaseUser`は、`User`モデルのコアをします。

Django `manage.py createsuperuser` コマンドがなのフィールドをることができるように、`REQUIRED_FIELDS`をすることができます。これはDjangoののではありません

```

class CustomUser(AbstractBaseUser):
    ...
    first_name = models.CharField(max_length=254)
    last_name = models.CharField(max_length=254)
    ...
    REQUIRED_FIELDS = ['first_name', 'last_name']

```

Djangoののにするためには、`is_active`、`get_full_name()`、`get_short_name()`をするがあります。

```
class CustomUser(AbstractBaseUser):
```

```
...
is_active = models.BooleanField(default=False)
...
def get_full_name(self):
    full_name = "{0} {1}".format(self.first_name, self.last_name)
    return full_name.strip()

def get_short_name(self):
    return self.first_name
```

また、カスタムする `UserManager` あなたのための `User` Django がすることをに作るモデル、`create_user()` と `create_superuser()`

```
from django.contrib.auth.models import BaseUserManager

class CustomUserManager(BaseUserManager):
    def create_user(self, email, first_name, last_name, password=None):
        if not email:
            raise ValueError('Users must have an email address')

        user = self.model(
            email=self.normalize_email(email),
        )

        user.set_password(password)
        user.first_name = first_name
        user.last_name = last_name
        user.save(using=self._db)
        return user

    def create_superuser(self, email, first_name, last_name, password):
        user = self.create_user(
            email=email,
            first_name=first_name,
            last_name=last_name,
            password=password,
        )

        user.is_admin = True
        user.is_active = True
        user.save(using=self.db)
        return user
```

ユーザーモデルの

`User` モデルをするプロジェクト および `AUTH_USER_MODEL` がされた では、 `AUTH_USER_MODEL` はしません。

たとえば、カスタマイズした `User` モデルを `User` してブログの `Post` モデルをするは、のようなカスタム `User` モデルをするがあります。

```
from django.conf import settings
from django.db import models

class Post(models.Model):
    author = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.CASCADE)
```

オンラインでユーザーモデルのまたはをむ <https://riptutorial.com/ja/django/topic/1209/ユーザーモデルのまたは>

40: ユニットテスト

Examples

テスト - な

これは、しいDjangoプロジェクトのにするドキュメントをんだことをとしています。プロジェクトのメインアプリケーションのがtdテストのであるとします。のテストをするには、test_view.py というのファイルをし、のをコピーしてコピーします。

```
from django.test import Client, TestCase

class ViewTest(TestCase):

    def test_hello(self):
        c = Client()
        resp = c.get('/hello/')
        self.assertEqual(resp.status_code, 200)
```

このテストは、

```
./manage.py test
```

それはもにするでしようのようなエラーがされます。

```
Traceback (most recent call last):
  File "/home/me/workspace/td/tests_view.py", line 9, in test_hello
    self.assertEqual(resp.status_code, 200)
AssertionError: 200 != 404
```

なぜそれがこるのですかたちはそれについてのビューをしていないのでだからそれをやりましよう。 views.pyというのファイルをし、のコードをします

```
from django.http import HttpResponse
def hello(request):
    return HttpResponse('hello')
```

に、のようにurls pyをして/ hello /にマップします

```
from td import views

urlpatterns = [
    url(r'^admin/', include(admin.site.urls)),
    url(r'^hello/', views.hello),
    ....
]
```

テストをもうやりして ./manage.py test again and viola !!

```
Creating test database for alias 'default'...
```

```
.
```

```
-----  
Ran 1 test in 0.004s
```

```
OK
```

Djangoモデルをにテストする

クラスをする

```
from django.db import models

class Author(models.Model):
    name = models.CharField(max_length=50)

    def __str__(self):
        return self.name

    def get_absolute_url(self):
        return reverse('view_author', args=[str(self.id)])

class Book(models.Model):
    author = models.ForeignKey(Author, on_delete=models.CASCADE)
    private = models.BooleanField(default=False)
    publish_date = models.DateField()

    def get_absolute_url(self):
        return reverse('view_book', args=[str(self.id)])

    def __str__(self):
        return self.name
```

テスト

```
from django.test import TestCase
from .models import Book, Author

class BaseModelTestCase(TestCase):

    @classmethod
    def setUpClass(cls):
        super(BaseModelTestCase, cls).setUpClass()
        cls.author = Author(name='hawking')
        cls.author.save()
        cls.first_book = Book(author=cls.author, name="short_history_of_time")
        cls.first_book.save()
        cls.second_book = Book(author=cls.author, name="long_history_of_time")
        cls.second_book.save()

class AuthorModelTestCase(BaseModelTestCase):
    def test_created_properly(self):
        self.assertEqual(self.author.name, 'hawking')
        self.assertEqual(True, self.first_book in self.author.book_set.all())
```

```

def test_absolute_url(self):
    self.assertEqual(self.author.get_absolute_url(), reverse('view_author',
args=[str(self.author.id)]))

class BookModelTestCase(BaseModelTestCase):

    def test_created_properly(self):
        ...
        self.assertEqual(1, len(Book.objects.filter(name__startswith='long')))

    def test_absolute_url(self):
        ...

```

いくつかの

- `created_properly` テストは、`django` モデルのプロパティをするためにされます。それらは、デフォルト、`file_upload_paths` などをしたのキャッチアップにちます。
- `absolute_url` はではないかもしれませんが、URL パスをするとはバグをぐことができました
- はにモデルでされたすべてのメソッドのテストケースをします `mock` オブジェクトなどをして
- の `BaseModelTestCase` をすることで、なテストをにうためにモデルになをすることができます

に、わしいときは、テストをく。なのはにをうことによってえられ、くれたコードはなトラブルをきこすことはありません。

Djangoビューでのアクセスのテスト

tl; dr 2つのユーザーオブジェクト `user` と `another_user` をするクラスをします。のモデルをし、3つの `Client` インスタンスをします。

- `self.client` ブラウザにログインしている `user` す
- `self.another_client` `another_user` のクライアントをす
- `self.unlogged_client` ログインのをす

これらの3つのクライアントオブジェクトからすべてのあなたのURLとURLにアクセスし、あなたがするをします。では、`private` のユーザーがまたは `public` でもることができるのいずれかの `Book` オブジェクトのをしました。

```

from django.test import TestCase, RequestFactory, Client
from django.core.urlresolvers import reverse

class BaseViewTestCase(TestCase):

    @classmethod
    def setUpClass(cls):
        super(BaseViewTestCase, cls).setUpClass()
        cls.client = Client()
        cls.another_client = Client()
        cls.unlogged_client = Client()
        cls.user = User.objects.create_user(
            'dummy', password='dummy'
        )

```

```

cls.user.save()
cls.another_user = User.objects.create_user(
    'dummy2', password='dummy2'
)
cls.another_user.save()
cls.first_book = Book.objects.create(
    name='first',
    private = True
)
cls.first_book.readers.add(cls.user)
cls.first_book.save()
cls.public_book = Template.objects.create(
    name='public',
    private=False
)
cls.public_book.save()

def setUp(self):
    self.client.login(username=self.user.username, password=self.user.username)
    self.another_client.login(username=self.another_user.username,
password=self.another_user.username)

"""
    Only cls.user owns the first_book and thus only he should be able to see it.
    Others get 403(Forbidden) error
"""
class PrivateBookAccessTestCase(BaseViewTestCase):

    def setUp(self):
        super(PrivateBookAccessTestCase, self).setUp()
        self.url = reverse('view_book',kwargs={'book_id':str(self.first_book.id)})

    def test_user_sees_own_book(self):
        response = self.client.get(self.url)
        self.assertEqual(200, response.status_code)
        self.assertEqual(self.first_book.name,response.context['book'].name)
        self.assertTemplateUsed('myapp/book/view_template.html')

    def test_user_cant_see_others_books(self):
        response = self.another_client.get(self.url)
        self.assertEqual(403, response.status_code)

    def test_unlogged_user_cant_see_private_books(self):
        response = self.unlogged_client.get(self.url)
        self.assertEqual(403, response.status_code)

"""
    Since book is public all three clients should be able to see the book
"""
class PublicBookAccessTestCase(BaseViewTestCase):

    def setUp(self):
        super(PublicBookAccessTestCase, self).setUp()
        self.url = reverse('view_book',kwargs={'book_id':str(self.public_book.id)})

    def test_user_sees_book(self):
        response = self.client.get(self.url)
        self.assertEqual(200, response.status_code)
        self.assertEqual(self.public_book.name,response.context['book'].name)

```

```

self.assertTemplateUsed('myapp/book/view_template.html')

def test_another_user_sees_public_books(self):
    response = self.another_client.get(self.url)
    self.assertEqual(200, response.status_code)

def test_unlogged_user_sees_public_books(self):
    response = self.unlogged_client.get(self.url)
    self.assertEqual(200, response.status_code)

```

データベースとテスト

Djangoはテストにデータベースをして、テストでデータベースをにできるようにしますが、デフォルトではのデータベースでします。あるテストでのデータベースのは、のテストではされません。たとえば、ののテストがします。

```

from django.test import TestCase
from myapp.models import Thing

class MyTest(TestCase):

    def test_1(self):
        self.assertEqual(Thing.objects.count(), 0)
        Thing.objects.create()
        self.assertEqual(Thing.objects.count(), 1)

    def test_2(self):
        self.assertEqual(Thing.objects.count(), 0)
        Thing.objects.create(attr1="value")
        self.assertEqual(Thing.objects.count(), 1)

```

のテストでデータベースオブジェクトをするは、テストケースの`setUp`メソッドでデータベースオブジェクトをするか、さらに、djangoプロジェクトでフィクスチャをしたは、のようにフィクスチャをみむことができます

```

class MyTest(TestCase):
    fixtures = ["fixture1.json", "fixture2.json"]

```

デフォルトでは、djangoはアプリの`fixtures`ディレクトリにある`fixtures`チャをしています。さらにディレクトリは`FIXTURE_DIRS`をとってできます

```

# myapp/settings.py
FIXTURE_DIRS = [
    os.path.join(BASE_DIR, 'path', 'to', 'directory'),
]

```

のようにモデルをしたとします。

```

# models.py
from django.db import models

```



```
class Person(models.Model):
    """A person defined by his/her first- and lastname."""
    firstname = models.CharField(max_length=255)
    lastname = models.CharField(max_length=255)
```

それで、あなたの.jsonのフィクスチャはのようになります

```
# fixture1.json
[
  { "model": "myapp.person",
    "pk": 1,
    "fields": {
      "firstname": "Peter",
      "lastname": "Griffin"
    }
  },
  { "model": "myapp.person",
    "pk": 2,
    "fields": {
      "firstname": "Louis",
      "lastname": "Griffin"
    }
  },
]
```

テストデータベースをする

テストのスピードを上げるために、コマンドにテストデータベースをするようすることができます
また、テストのたびにされたりされることをぐため。これは、のようにkeepdbまたはshorthand -
k フラグをとってうことができます

```
# Reuse the test-database (since django version 1.8)
$ python manage.py test --keepdb
```

されるテストのをする

テストランナーがどのモジュールをするべきかをするによって、 `manage.py test` によってされるテストをすることができます

```
# Run only tests for the app names "appl"
$ python manage.py test appl

# If you split the tests file into a module with several tests files for an app
$ python manage.py test appl.tests.test_models

# it's possible to dig down to individual test methods.
$ python manage.py test appl.tests.test_models.MyTestCase.test_something
```

たくさんのテストをしたいは、ファイルのパターンをすことができます。たとえば、モデルにするテストのみをします。

```
$ python manage.py test -p test_models*
```

```
Creating test database for alias 'default'...
.....
-----
Ran 115 tests in 3.869s

OK
```

に、`--failfast`をして、のにテストスイートをすることができます。このをすると、スイートでしたなエラーをくできます。

```
$ python manage.py test appl
...F..
-----
Ran 6 tests in 0.977s

FAILED (failures=1)

$ python manage.py test appl --failfast
...F
=====
[Traceback of the failing test]
-----
Ran 4 tests in 0.372s

FAILED (failures=1)
```

オンラインでユニットテストをむ <https://riptutorial.com/ja/django/topic/1232/ユニットテスト>

41: ロギング

Examples

Syslog サービスへのロギング

Djangoがローカルまたはリモートsyslogサービスにログをするようにすることはです。ここでは、PythonみみのSysLogHandlerをします。

```
from logging.handlers import SysLogHandler
LOGGING = {
    'version': 1,
    'disable_existing_loggers': True,
    'formatters': {
        'standard': {
            'format' : "[YOUR PROJECT NAME] [%asctime)s] %(levelname)s [%s] %(message)s",
            'datefmt' : "%d/%b/%Y %H:%M:%S"
        }
    },
    'handlers': {
        'console': {
            'class': 'logging.StreamHandler',
        },
        'syslog': {
            'class': 'logging.handlers.SysLogHandler',
            'formatter': 'standard',
            'facility': 'user',
            # uncomment next line if rsyslog works with unix socket only (UDP reception
disabled)
            #'address': '/dev/log'
        }
    },
    'loggers': {
        'django':{
            'handlers': ['syslog'],
            'level': 'INFO',
            'disabled': False,
            'propagate': True
        }
    }
}

# loggers for my apps, uses INSTALLED_APPS in settings
# each app must have a configured logger
# level can be changed as desired: DEBUG, INFO, WARNING...
MY_LOGGERS = {}
for app in INSTALLED_APPS:
    MY_LOGGERS[app] = {
        'handlers': ['syslog'],
        'level': 'DEBUG',
        'propagate': True,
    }
LOGGING['loggers'].update(MY_LOGGERS)
```

Djangoのログ

には、DjangoはPythonロギングシステムをします。プロジェクトのロギングをするにはくのがあ
ります。ここにベースがあります

```
LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'formatters': {
        'default': {
            'format': "[%(asctime)s] %(levelname)s [%(name)s:%(lineno)s] %(message)s",
            'datefmt': "%Y-%m-%d %H:%M:%S"
        },
    },
    'handlers': {
        'console': {
            'level': 'INFO',
            'class': 'logging.StreamHandler',
            'formatter': 'default'
        },
    },
    'loggers': {
        'django': {
            'handlers': ['console'],
            'propagate': True,
            'level': 'INFO',
        },
    },
}
```

フォーマッタ

のためにされたときにログのをするためにすることができます。それぞれのフォーマッタにキー
をすることによって、くのフォーマッタをできます。に、ハンドラをするときにフォーマッタが
されます。

ハンドラー

ログをするをするためにできます。のでは、それらはstdoutとstderrにられます。さまざまなハン
ドラクラスがあります。

```
'rotated_logs': {
    'class': 'logging.handlers.RotatingFileHandler',
    'filename': '/var/log/my_project.log',
    'maxBytes': 1024 * 1024 * 5, # 5 MB
    'backupCount': 5,
    'formatter': 'default'
    'level': 'DEBUG',
},
```

これにより、 filename targetedされたfilenameログがされfilename。このでは、のサイズが5 MB
にしたときにしいログファイルがされますいものはmy_project.log.1にがされます。の5つのファ
イルはアーカイブにされます。

```
'mail_admins': {  
    'level': 'ERROR',  
    'class': 'django.utils.log.AdminEmailHandler'  
},
```

これは、ログを `ADMINS` でされたユーザーに email でします。レベルがするようにされている `ERROR` レベルでこれだけのログ、 `ERROR` メールでされます。これは、プロダクションサーバーでなエラー 50x についてのをるためににです。

のハンドラは Django でできます。なリストについては、するをおみください。フォーマッタとに、じプロジェクトにのハンドラをし、それぞれなるキーをすることができます。ハンドラはのロガーでできます。

ロガー

`LOGGING` では、モジュールなので、ロギングレベル、するハンドラなどをします。

オンラインでロギングをむ <https://riptutorial.com/ja/django/topic/1231/ロギング>

42: なビュー

き

なビューは、オブジェクトの、、、またはテンプレートのなど、あらかじめされたのアクションをするビューです。

なビューは、なタスクをするためににでされているビューとするがあります。すると、ビューをするがあり、ビューをプログラムするがあるとえます。

なビューは、にされたくのタスクをするに、くのできます。

これらは、なビューはにされたタスクをはるかにすることをしていきますからすべてをプログラミングするのではなく、のがすでにあなたのためにプログラミングしたものをします。これは、バックグラウンドでのプロセスではなく、プロジェクトのにもっとすることができるため、くのがあります。

だから、あなたはいつもそれらをうべきですかいいえ、あなたのタスクがかなりされているオブジェクトのみみ、、、りしのタスクがいり、それらはをなさないでしょう。1つののビューをししない、になタスクをするためにすべてのメソッドをオーバーライドすることはをなさないかもしれません。ここでなビューをおめします。

ただし、このがなビューがたくさんあるや、タスクがのビューのみタスクにする、ビューは、よりなものにするためになものです。

Examples

ビューとビュー

オブジェクトをするためのビューの。コメントやをいて、15のコードがです。

```
# imports
from django.shortcuts import render_to_response
from django.http import HttpResponseRedirect

from .models import SampleObject
from .forms import SampleObjectForm

# view function
def create_object(request):

    # when request method is 'GET', show the template
    if request.method == GET:
        # perform actions, such as loading a model form
        form = SampleObjectForm()
        return render_to_response('template.html', locals())
```

```
# if request method is 'POST', create the object and redirect
if request.method == POST:
    form = SampleObjectForm(request.POST)

    # save object and redirect to success page if form is valid
    if form.is_valid:
        form.save()
        return HttpResponseRedirect('url_to_redirect_to')

    # load template with form and show errors
    else:
        return render_to_response('template.html', locals())
```

じタスクをするための「クラスベースのなビュー」のじタスクをするには、7のコードしかありません。

```
from django.views.generic import CreateView

from .models import SampleObject
from .forms import SampleObjectForm

class CreateObject(CreateView):
    model = SampleObject
    form_class = SampleObjectForm
    success_url = 'url_to_redirect_to'
```

ビューのカスタマイズ

のは、タスクがにタスクであるにのみします。たとえば、ここになコンテキストをしないでください。

よりなをりましょう。テンプレートにページタイトルをしたいとします。ビューでは、これはの
ように1するだけでします。

```
def create_object(request):
    page_title = 'My Page Title'

    # ...

    return render_to_response('template.html', locals())
```

なビューではこれをするのがよりしくなりますまたは。クラスベースであるため、のをるために
クラスのメソッドの1つまたはをオーバーライドするがあります。このでは、クラスの
`get_context_data`メソッドをのようオーバーライドするがあります。

```
class CreateObject(CreateView):
    model = SampleObject
    form_class = SampleObjectForm
    success_url = 'url_to_redirect_to'

    def get_context_data(self, **kwargs):

        # Call class's get_context_data method to retrieve context
```

```
context = super().get_context_data(**kwargs)

context['page_title'] = 'My page title'
return context
```

ここでは、1つではなく4つのコードをするがあります。なくとも、したいののコンテキストです。

ミックスインをしたなビュー

ミックスインとみわせると、ジェネリックビューののパワーががります。 `mixin`は、ビュークラスによってされるメソッドをつ、あなたがしたのクラスです。

すべてのビューでの `'page_title'`がテンプレートにされるようにしたいとします。ビューをするたびに `get_context_data`メソッドをオーバーライドするわりに、このメソッドでミックスインをし、このミックスインからするビューをできます。それはよりもです。

```
# Your Mixin
class CustomMixin(object):

    def get_context_data(self, **kwargs):

        # Call class's get_context_data method to retrieve context
        context = super().get_context_data(**kwargs)

        context['page_title'] = 'My page title'
        return context

# Your view function now inherits from the Mixin
class CreateObject(CustomMixin, CreateView):
    model = SampleObject
    form_class = SampleObjectForm
    success_url = 'url_to_redirect_to'

# As all other view functions which need these methods
class EditObject(CustomMixin, EditView):
    model = SampleObject
    # ...
```

これのしさは、コードがなビューのほとんどのよりもずっとされていることです。のタスクのにあるロジックは、1かに1つしかありません。また、さまざまなオブジェクトをいてにじタスクをするビューがあるは、をにできます

オンラインでなビューをむ <https://riptutorial.com/ja/django/topic/9452/なビュー>

43:

パラメーター

クラス/メソッド	なぜ
UserProfileクラス	UserProfileクラスは、Djangoのデフォルトユーザーモデルをします。
create_profileメソッド	create_profileメソッドは、Django Userモデルのpost_saveシグナルがされるたびにされます。

、。

Djangoシグナルは、にのタスクまたはのモデルやなどをアプリにします。

これらのをすると、がリリースされたにしたアクションをします。

例えば、いつでもいい Djangoのユーザーがされ、ユーザーモデルは、のようないのparamsで、をしsender=User、あなたがにこのには、したのへののリスニングをターゲットにすることができ、しいユーザーの。

のでは、Userオブジェクトがされたに、UserProfileオブジェクトをすることをしています。したがって、UserモデルデフォルトのDjango User Modelからのpost_saveをpost_saveくことによって、しいUserがされたにUserProfileオブジェクトをします。

Djangoドキュメンテーションは、なすべてのなシグナルにするなドキュメントをしています。

しかし、のは、をすることがなとなるなをにすることです。

「きなをもって、きなをう」らがらしいのだからあなたのアプリケーションやプロジェクトにをらばってしまうことがかもしれません。まあ、しないでください。らはクールだから、にかぶすべてのなのためのにはなりません。

は、いつものように、すべてではありません。ログイン/ログアウト、はらしいです。ユーザーモデルなどのをつモデル。

あなたのアプリケーションのすべてのモデルのためのシグナルをすることは、あるでされ、Djangoシグナルのスパースなのアイデアをにちかします。

Djangoの2つのスコープについてシグナルをしないでください

- シグナルはの1つのモデルにし、そのモデルのメソッドの1つにできます。おそらくsave()によってびされsave()。

- このは、カスタムモデルマネージャメソッドでできることができます。
- はのビューにし、そのビューにできます

のにをしてもありません。

- は、のモデルにをえるがあります。
- じをのアプリからし、のでじでしたいとします。
- モデルのにキャッシュをにしたいとします。
- あなたはコールバックをとするしいシナリオがあり、シグナルをするにそれをするのはありません。たとえば、サードパーティアプリケーションのモデルの`save()`または`init()`についてかをトリガーしたいとします。サードパーティのコードをすることはできません。がなもあります。したがって、シグナルがコールバックのトリガーとなります。

Examples

ユーザプロフィールの

このは、[ProのようなDjangoユーザプロフィール](#)からしたスニペットです

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

class UserProfile(models.Model):
    user = models.OneToOneField(User, related_name='user')
    website = models.URLField(default='', blank=True)
    bio = models.TextField(default='', blank=True)

def create_profile(sender, **kwargs):
    user = kwargs["instance"]
    if kwargs["created"]:
        user_profile = UserProfile(user=user)
        user_profile.save()
    post_save.connect(create_profile, sender=User)
```

シグナルをIするためのなる

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

class UserProfile(models.Model):
    user = models.OneToOneField(User, related_name='user')
    website = models.URLField(default='', blank=True)
    bio = models.TextField(default='', blank=True)

@receiver(post_save, sender=UserProfile)
def post_save_user(sender, **kwargs):
    user = kwargs.get('instance')
    if kwargs.get('created'):
        ...
```

pre_saveのまたはであるかどうかをべる

pre_saveをすることで、データベースのsaveアクションがのオブジェクトのやしいオブジェクトのにするものかどうかをできます。

これをするためには、モデルオブジェクトのをチェックすることができます

```
@receiver(pre_save, sender=User)
def pre_save_user(sender, instance, **kwargs):
    if not instance._state.adding:
        print ('this is an update')
    else:
        print ('this is an insert')
```

saveがされるpre_saveに、pre_saveがされ、のpre_saveされます。

- this is an update アクションからされたアクションののthis is an updateです。
- this is an insert アクションがアクションからしたのthis is an insertです。

このメソッドは、のデータベースクエリをとしないことにしてください。

モデルへの

Djangoのシグナルはになクラスにされているため、サブクラスされたモデルはすぐにじシグナルにされません。

このモデルとをる

```
class Event(models.Model):
    user = models.ForeignKey(User)

class StatusChange(Event):
    ...

class Comment(Event):
    ...

def send_activity_notification(sender, instance: Event, raw: bool, **kwargs):
    """
    Fire a notification upon saving an event
    """

    if not raw:
        msg_factory = MessageFactory(instance.id)
        msg_factory.on_activity(str(instance))
    post_save.connect(send_activity_notification, Event)
```

モデルでは、をサブクラスにでして、をけないようにするがあります。

```
post_save.connect(send_activity_notification, StatusChange)
post_save.connect(send_activity_notification, Comment)
```

Python 3.6では、いくつかのクラスメソッドをして、このバインディングをするクラスをすることが出来ます。

```
class Event(models.Model):

    @classmethod
    def __init_subclass__(cls, **kwargs):
        super().__init_subclass__(**kwargs)
        post_save.connect(send_activity_notification, cls)
```

オンラインでをむ <https://riptutorial.com/ja/django/topic/2555/>

44:

- gettextメッセージ
- gettext、
- ugettextメッセージ
- ungettext、
- pgettextコンテキスト、メッセージ
- npgettextコンテキスト、、、
- gettext_lazyメッセージ
- gettext_lazy、 =なし
- ugettext_lazyメッセージ
- ungettext_lazy、 =なし
- pgettext_lazyコンテキスト、メッセージ
- npgettext_lazyコンテキスト、、、 =なし
- gettext_noopメッセージ
- ugettext_noopメッセージ

Examples

セットアップ

settings.py

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

USE_I18N = True # Enable Internationalization
LANGUAGE_CODE = 'en' # Language in which original texts are written
LANGUAGES = [ # Available languages
    ('en', _("English")),
    ('de', _("German")),
    ('fr', _("French")),
]

# Make sure the LocaleMiddleware is included, AFTER SessionMiddleware
# and BEFORE middlewares using internationalization (such as CommonMiddleware)
MIDDLEWARE_CLASSES = [
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.locale.LocaleMiddleware',
    'django.middleware.common.CommonMiddleware',
]
```

をなものとしてマークする

ののステップは、 をとしてマークすることです。これは `gettext` の1つをしています「[」セクション](#)

ンを。たとえば、モデルのをにします。

```
from django.utils.translation import ugettext_lazy as _
# It is common to import gettext as the shortcut `_` as it is often used
# several times in the same file.

class Child(models.Model):

    class Meta:
        verbose_name = _("child")
        verbose_name_plural = _("children")

    first_name = models.CharField(max_length=30, verbose_name=_("first name"))
    last_name = models.CharField(max_length=30, verbose_name=_("last name"))
    age = models.PositiveSmallIntegerField(verbose_name=_("age"))
```

__() カプセルされたすべてののは、とマークされるようになりました。には、されたになく、カプセルされたとしてにされますまだができないため。

の

このは、をするのにです。ほとんどの、をとなして、プロジェクトののをしたいだけです。したがって、これはのでされています。

レイジーノンレイジー

をすると、はすぐにされます。

```
>>> from django.utils.translation import activate, ugettext as _
>>> month = _("June")
>>> month
'June'
>>> activate('fr')
>>> _("June")
'juin'
>>> activate('de')
>>> _("June")
'Juni'
>>> month
'June'
```

をする、はにされるときにのみします。

```
>>> from django.utils.translation import activate, ugettext_lazy as _
>>> month = _("June")
>>> month
<django.utils.functional.lazy.<locals>.__proxy__ object at 0x7f61cb805780>
>>> str(month)
'June'
>>> activate('fr')
>>> month
<django.utils.functional.lazy.<locals>.__proxy__ object at 0x7f61cb805780>
```

```
>>> "month: {}".format(month)
'month: juin'
>>> "month: %s" % month
'month: Juni'
```

のようなには、をすることができます。

- `_("some string")` がされたときにがにならないがされていない
- のは、モデルやフォームフィールドなどのクラスなどでのみされます。

テンプレートによる

テンプレートでをにするには、 `i18n` ライブラリをみむがあります。

```
{% load i18n %}
```

なは `trans` テンプレートタグでわれます。

```
{% trans "Some translatable text" %}
{# equivalent to python `gettext("Some translatable text")` #}
```

`trans` テンプレートタグはコンテキストをサポートしています

```
{% trans "May" context "month" %}
{# equivalent to python `pgettext("May", "month")` #}
```

にブレースホルダをめるには、のようにします。

```
_("My name is {first_name} {last_name}").format(first_name="John", last_name="Doe")
```

`blocktrans` テンプレートタグをすることができます

```
{% blocktrans with first_name="John" last_name="Doe" %}
    My name is {{ first_name }} {{ last_name }}
{% endblocktrans %}
```

もちろん、 `"John"` と `"Doe"` ではなく、とフィルタをできます。

```
{% blocktrans with first_name=user.first_name last_name=user.last_name|title %}
    My name is {{ first_name }} {{ last_name }}
{% endblocktrans %}
```

`first_name` と `last_name` がすでにコンテキストにあるは、 `with` をすることもできます。

```
{% blocktrans %}My name is {{ first_name }} {{ last_name }}{% endblocktrans %}
```

ただし、「トップレベル」のコンテキストのみをできます。 これはしません

```
{% blocktrans %}
    My name is {{ user.first_name }} {{ user.last_name }}
{% endblocktrans %}
```

これはに、がファイルのプレースホルダとしてされているためです。

blocktrans テンプレートタグはもけれます。

```
{% blocktrans count nb=users|length %}
    There is {{ nb }} user.
{% plural %}
    There are {{ nb }} users.
{% endblocktrans %}
```

に、`gettext` ライブラリになく、`gettext` をしてテンプレートタグにすことができます。

```
{{ site_name|default:_("It works!") }}
{% firstof var1 var2 _("translatable fallback") %}
```

これは、`gettext` をするいくつかののみ `django` テンプレートシステムですが、これは `gettext` ではありません。 `_("It works!")` にされた `default` としてテンプレートタグ `_("It works!")` とに、その、`gettext` をされ、`name` としてされることになる `"name"` になりますとしてされます。

の

をするには、ファイルをするがあります。そのために、`django` はコマンド `makemessages` ます。

```
$ django-admin makemessages -l fr
processing locale fr
```

のコマンドは、インストールされたアプリでとマークされたすべてのをし、フランスのためにアプリごとに1つのファイルをしします。たとえば、`gettext` をむ `app myapp` が1つしかない、`myapp/locale/fr/LC_MESSAGES/django.po` というファイルがされます。このファイルはのようになります。

```
# SOME DESCRIPTIVE TITLE
# Copyright (C) YEAR THE PACKAGE'S COPYRIGHT HOLDER
# This file is distributed under the same license as the PACKAGE package.
# FIRST AUTHOR <EMAIL@ADDRESS>, YEAR
#
#, fuzzy
msgid ""
msgstr ""
"Project-Id-Version: PACKAGE VERSION\n"
"Report-Msgid-Bugs-To: \n"
"POT-Creation-Date: 2016-07-24 14:01+0200\n"
"PO-Revision-Date: YEAR-MO-DA HO:MI+ZONE\n"
"Last-Translator: FULL NAME <EMAIL@ADDRESS>\n"
"Language-Team: LANGUAGE <LL@li.org>\n"
"Language: \n"
"MIME-Version: 1.0\n"
```



```

"Content-Type: text/plain; charset=UTF-8\n"
"Content-Transfer-Encoding: 8bit\n"

#: myapp/models.py:22
msgid "user"
msgstr ""

#: myapp/models.py:39
msgid "A user already exists with this email address."
msgstr ""

#: myapp/templates/myapp/register.html:155
#, python-format
msgid ""
"By signing up, you accept our <a href=\"%%(terms_url)s\" "
"target=_blank>Terms of services</a>."
msgstr ""

```

まず、プレースホルダをするがあります。その、をします。 `msgid` はあなたのコードでとマークされたです。 `msgstr` あなたはののをくがあります。

にプレースホルダがまれているは、それらにもめるがあります。たとえば、のメッセージをのうにします。

```

#: myapp/templates/myapp/register.html:155
#, python-format
msgid ""
"By signing up, you accept our <a href=\"%%(terms_url)s\" "
"target=_blank>Terms of services</a>."
msgstr ""
"En vous inscrivant, vous acceptez nos <a href=\"%%(terms_url)s\" "
"target=_blank>Conditions d'utilisation</a>"

```

ファイルがしたら、`.po` ファイルを `.mo` ファイルにコンパイルするがあります。これは、`compilemessages` コマンドをびすことによってわれます。

```
$ django-admin compilemessages
```

それで、ができるようになりました。

コードをしたときにファイルをするには、`django-admin makemessages -l fr` します。これにより、`.po` ファイルがされ、のがされ、しいがされます。されたはコメントでききできます。すべての `.po` ファイルをするには、`django-admin makemessages -a` します。`.po` ファイルがされたら、`django-admin compilemessages` して `.mo` ファイルをすることをれないでください。

Noop ユースケース

(u) `gettext_noop` では、にすることなくをとしてマークすることができます。

なは、けのメッセージをでしたいが、それをクライアントにされたでしたいです。 `gettext` にをすることはできますが、そのはごとにわるため、なとしてはされません。。

```
# THIS WILL NOT WORK AS EXPECTED
import logging
from django.contrib import messages

logger = logging.getLogger(__name__)

error_message = "Oops, something went wrong!"
logger.error(error_message)
messages.error(request, _(error_message))
```

エラーメッセージは`.po`ファイルにはされず、でするがあることをえておくがあります。これをす
るには、`gettext_noop`をし`gettext_noop`。

```
error_message = ugettext_noop("Oops, something went wrong!")
logger.error(error_message)
messages.error(request, _(error_message))
```

すぐ`"Oops, something went wrong!"`されると`.po`ファイルでされ、になります。エラーはききのため
にでされます。

のとし

あいまいな

`makemessages`は、のためにつけたが、すでにするとしているとえるかもしれません。 `.po`ファイル
にのような`fuzzy`コメントをけてマークするとします

```
#: templates/randa/map.html:91
#, fuzzy
msgid "Country"
msgstr "Länderinfo"
```

がしいか、それをするためにしたとしても、あなたが`fuzzy`コメントをしないうり、あなたのプロジ
ェクトをするためにはされません。

の

`makemessages`は、プレーンテキストからPythonコードまでさまざまなファイルをしますが、これ
らのののをつのあるすべてののにうようにはされていません。ほとんどの、のでうまくしますが、
のようなをしているは、

```
translation = _("firstline"
"secondline"
"thirdline")
```

それはのための`firstline`だけをうでしよう。これをするには、なときにのをしないようにします
。

オンラインでをむ <https://riptutorial.com/ja/django/topic/2579/>

45: の

Examples

スルーモデルで

```
class Skill(models.Model):
    name = models.CharField(max_length=50)
    description = models.TextField()

class Developer(models.Model):
    name = models.CharField(max_length=50)
    skills = models.ManyToManyField(Skill, through='DeveloperSkill')

class DeveloperSkill(models.Model):
    """Developer skills with respective ability and experience."""

    class Meta:
        order_with_respect_to = 'developer'
        """Sort skills per developer so that he can choose which
        skills to display on top for instance.
        """
        unique_together = [
            ('developer', 'skill'),
        ]
        """It's recommended that a together unique index be created on
        `(developer, skill)`. This is especially useful if your database is
        being access/modified from outside django. You will find that such an
        index is created by django when an explicit through model is not
        being used.
        """

    ABILITY_CHOICES = [
        (1, "Beginner"),
        (2, "Accustomed"),
        (3, "Intermediate"),
        (4, "Strong knowledge"),
        (5, "Expert"),
    ]

    developer = models.ForeignKey(Developer, models.CASCADE)
    skill = models.ForeignKey(Skill, models.CASCADE)
    """The many-to-many relation between both models is made by the
    above two foreign keys.

    Other fields (below) store information about the relation itself.
    """

    ability = models.PositiveSmallIntegerField(choices=ABILITY_CHOICES)
    experience = models.PositiveSmallIntegerField(help_text="Years of experience.")
```

にのインデックスをすることをおめします `(developer, skill)`。これは、データベースがのdjangoからアクセス/されているににです。このようなインデックスは、なスルーモデルがされていないときにdjangoによってされることがわかります。

なの。

```
class Person(models.Model):
    name = models.CharField(max_length=50)
    description = models.TextField()

class Club(models.Model):
    name = models.CharField(max_length=50)
    members = models.ManyToManyField(Person)
```

ここでは、クラブがくの`Person`とメンバーをち、`Person`がのなる`Club`のメンバーとなるをします。

2つのモデルしかしていませんが、`django`はにデータベースに3つのテーブルをします。これらは`myapp_person`、`myapp_club`、および`myapp_club_members`です。`Django`はに`myapp_club_members (club_id, person_id)`にユニークなインデックスをします。

ManyToManyフィールドの

のからこのモデルをします。

```
class Person(models.Model):
    name = models.CharField(max_length=50)
    description = models.TextField()

class Club(models.Model):
    name = models.CharField(max_length=50)
    members = models.ManyToManyField(Person)
```

トムとビルをナイトクラブにえる

```
tom = Person.objects.create(name="Tom", description="A nice guy")
bill = Person.objects.create(name="Bill", description="Good dancer")

nightclub = Club.objects.create(name="The Saturday Night Club")
nightclub.members.add(tom, bill)
```

クラブにはがいますか

```
for person in nightclub.members.all():
    print(person.name)
```

あなたにえるだろう

```
Tom
Bill
```

オンラインでのをむ <https://riptutorial.com/ja/django/topic/2379/>の

46:

き

タイムゾーンはしばしばにとってです。 Djangoはタイムゾーンをいやすくするために、いくつかのらしいユーティリティをしています。

プロジェクトがのタイムゾーンでされているでも、データベースにUTCとしてデータをして、のケースをすることをおめします。のタイムゾーンでしている、データをUTCとしてするがあります。

Examples

タイムゾーンのサポートをにする

にまず、 settings.py ファイルで `USE_TZ = True` をし settings.py。 また、にデフォルトのタイムゾーンを `TIME_ZONE` など `TIME_ZONE='UTC'`。 [ここにタイムゾーンのなりリストをします](#)。

`USE_TZ False`で、 `TIME_ZONE` Djangoは、すべてのをするためにするタイムゾーンになります。とき `USE_TZ` になっている、 `TIME_ZONE` Djangoはテンプレートのをするためにされますと、フォームにしたをするために、デフォルトのタイムゾーンです。

タイムゾーンのサポートがになっていると、 djangoは `datetime` データをタイムゾーン `UTC` としてデータベースにします

セッションのタイムゾーンをする

Pythonの `datetime.datetime` オブジェクトには、タイムゾーンをするためにされる `tzinfo` があります。がされている、そのオブジェクトはされているとみなされます。がされていないは、Naiveとみなされます。

タイムゾーンがナীবであることをするには、 `.is_naive()` と `.is_aware()` します。

settings.py ファイルに `USE_TZ` になっている、デフォルトの `TIME_ZONE` が settings.py されているり、 `datetime` にタイムゾーンがされ settings.py

このデフォルトのタイムゾーンは、によってはながありますが、のタイムゾーンでユーザーをするは、になががあります。これをするには、ミドルウェアをするがあります。

```
import pytz

from django.utils import timezone

# make sure you add `TimezoneMiddleware` appropriately in settings.py
class TimezoneMiddleware(object):
```

```

"""
Middleware to properly handle the users timezone
"""

def __init__(self, get_response):
    self.get_response = get_response

def __call__(self, request):
    # make sure they are authenticated so we know we have their tz info.
    if request.user.is_authenticated():
        # we are getting the users timezone that in this case is stored in
        # a user's profile
        tz_str = request.user.profile.timezone
        timezone.activate(pytz.timezone(tz_str))
    # otherwise deactivate and the default time zone will be used anyway
    else:
        timezone.deactivate()

    response = self.get_response(request)
    return response

```

しいことがいくつかあります。ミドルウェアとそれがチェックアウトしないのについては、[ドキュメントの](#)を。 `__call__` では、タイムゾーンデータのをしています。まず、このユーザーのタイムゾーンデータがあることをするために、ユーザーがされていることをします。これがわかったら、 `timezone.activate()` をしてユーザーセッションのタイムゾーンをアクティブにします。たちがっているタイムゾーンのを `datetime` でできるものにするために、 `pytz.timezone(str)` をし `pytz.timezone(str)` 。

は、テンプレートで `datetime` オブジェクトにアクセスすると、データベースの 'UTC' から、ユーザーがいるのににされます。 `datetime` オブジェクトにアクセスするだけで、のミドルウェアがされているとしてタイムゾーンがされますしく。

```
{{ my_datetime_value }}
```

ユーザーのタイムゾーンがされているかどうかをきめかくしたいは、をしてください。

```

{% load tz %}
{% localtime on %}
    {# this time will be respect the users time zone #}
    {{ your_date_time }}
{% endlocaltime %}

{% localtime off %}
    {# this will not respect the users time zone #}
    {{ your_date_time }}
{% endlocaltime %}

```

このは、*Django 1.10*でのみすることにしてください。 1.10よりの *django* をサポートするには、[MiddlewareMixin](#)をてください

オンラインでをむ <https://riptutorial.com/ja/django/topic/10566/>

47:

パラメーター

django-admin コマンド	
makemigrations <my_app>	my_app マイグレーションをする
makemigrations	すべてのアプリケーションのを
makemigrations --merge	すべてのアプリケーションのを
makemigrations --merge <my_app>	my_appのを
makemigrations --name <migration_name> <my_app>	migration_nameというのmy_appをし migration_name
migrate <my_app>	my_appのをデータベースにする
migrate	すべてののをデータベースにする
migrate <my_app> <migration_name>	migration_nameまたはする
migrate <my_app> zero	my_appですべてのをmy_app
sqlmigrate <my_app> <migration_name>	きのSQLをします。
showmigrations	すべてのアプリケーションのすべてのをし す
showmigrations <my_app>	my_appのすべてのをしmy_app

Examples

の

Djangoはマイグレーションをして、モデルにえたをデータベースにします。ほとんどの、djangoはそれらをすることができます。

をするには、をします。

```
$ django-admin makemigrations <app_name>
```

これにより、 app_name migration サブモジュールにマイグレーション・ファイルがされます。のは

0001_initial.py とばれ、もう1つは 0002_、 0003、 にされます。

<app_name> をすると、すべての INSTALLED_APPS がされます。

をデータベースにするには、のコマンドをします。

```
$ django-admin migrate <app_name>
```

すべてのをするには、のコマンドをします。

```
$ django-admin showmigrations app_name
app_name
[X] 0001_initial
[X] 0002_auto_20160115_1027
[X] 0003_somemodel
[ ] 0004_auto_20160323_1826
```

- [X] は、がデータベースにされたことをします
- [] は、マイグレーションがデータベースにされなかったことをします。 django-admin migrate をしてそれをする

をにすこともできます。これは、を migrate command すことでできます。ののリスト django-admin showmigrations されているをと、

```
$ django-admin migrate app_name 0002 # Roll back to migration 0002
$ django-admin showmigrations app_name
app_name
[X] 0001_initial
[X] 0002_auto_20160115_1027
[ ] 0003_somemodel
[ ] 0004_auto_20160323_1826
```

々、Djangoによってされたはではありません。これは、 データをうににてはまります。

たとえば、あなたはそのようなモデルをっています

```
class Article(models.Model):
    title = models.CharField(max_length=70)
```

このモデルにはすでにのデータがあり、では SlugField をし SlugField

```
class Article(models.Model):
    title = models.CharField(max_length=70)
    slug = models.SlugField(max_length=70)
```

フィールドをするためにをしました、 title にじてのすべてののslugをします。

もちろん、あなたはのそのようなことをすることができます

```
$ django-admin shell
```

```
>>> from my_app.models import Article
>>> from django.utils.text import slugify
>>> for article in Article.objects.all():
...     article.slug = slugify(article.title)
...     article.save()
...
>>>
```

しかし、あなたはあなたのすべてのつまり、あなたのオフィスのデスクトップ、ラップトップ、などでこれをうががあります。あなたのはにそうしなければなりません。あなたはステージングにそれについてえるがあります。ライブ。

これをにうために、たちはでそれをいます。にのをします。

```
$ django-admin makemigrations --empty app_name
```

これにより、のファイルがされます。それをくと、がまれています。ので0023_article_slugというがけられ、こののが0024_auto_20160719_1734ます。ファイルにはのようになります。

```
# -*- coding: utf-8 -*-
# Generated by Django 1.9.7 on 2016-07-19 15:34
from __future__ import unicode_literals

from django.db import migrations
from django.utils.text import slugify

def gen_slug(apps, schema_editor):
    # We can't import the Article model directly as it may be a newer
    # version than this migration expects. We use the historical version.
    Article = apps.get_model('app_name', 'Article')
    for row in Article.objects.all():
        row.slug = slugify(row.name)
        row.save()

class Migration(migrations.Migration):

    dependencies = [
        ('hosting', '0023_article_slug'),
    ]

    operations = [
        migrations.RunPython(gen_slug, reverse_code=migrations.RunPython.noop),
        # We set `reverse_code` to `noop` because we cannot revert the migration
        # to get it back in the previous state.
        # If `reverse_code` is not given, the migration will not be reversible,
        # which is not the behaviour we expect here.
    ]
```

の

がされると、Djangoはのをdjango_migrationsテーブルにします。

のスキーマののと

あなたのアプリににモデルとデータベーステーブルがあり、がない。まず、アプリのをします。

```
python manage.py makemigrations your_app_label
```

されたをする

```
python manage.py migrate --fake-initial
```

すべてのアプリケーションですべてのをする

```
python manage.py migrate --fake
```

のアプリの

```
python manage.py migrate --fake core
```

のファイル

```
python manage.py migrate myapp migration_name
```

ファイルのカスタム

makemigrations --name <your_migration_name> オプションをすると、されたをするわりにマイグレーションのをけることができます。

```
python manage.py makemigrations --name <your_migration_name> <app_name>
```

のの

き

にはがし、がにわってしてしまうことがあります。これはくのシーンでするがありますが、チームで1つのアプリをするときににするがあります。

ソースをしている、なのがします。に、ブランチフィーチャごとのがされているにします。このシナリオでは、 name と address をつ Reporter というモデルをし address 。

これで2のがフィーチャーをしているので、とも Reporter モデルののコピーをします。Aはファイル 0002_reporter_age.py ファイルのとなる age をします。Bは、 bank_account する bank_account フィールドをし 0002_reporter_bank_account 。これらのがコードをしてをしようとすると、のがしました。

このは、これらのがしモデル Reporter するためにします。さらに、しいファイルはとも 0002 でま

ります。

のマージ

それをうにはいくつかのがあります。はされるです

1. もなは、makemigrationsコマンドを--mergeフラグですることです。

```
python manage.py makemigrations --merge <my_app>
```

これにより、のをするしいがされます。

2. このなファイルがでなでされない、するをすることもできます。その、のmakemigrationsコマンドをしてしいをうことができます。カスタムがかれているは、のようなmigrations.RunPython、このをいてするがあります。

CharFieldをキーにする

に、discographyとばれるアプリケーションので、これがあなたのモデルであるとしましょう。

```
from django.db import models

class Album(models.Model):
    name = models.CharField(max_length=255)
    artist = models.CharField(max_length=255)
```

さて、アーティストのわりにForeignKeyをいたいとっています。これはややなプロセスで、いくつかのステップでするがあります。

ステップ1、ForeignKeyにしいフィールドをし、それをヌルとしてマークしますリンクのモデルもまれています。

```
from django.db import models

class Album(models.Model):
    name = models.CharField(max_length=255)
    artist = models.CharField(max_length=255)
    artist_link = models.ForeignKey('Artist', null=True)

class Artist(models.Model):
    name = models.CharField(max_length=255)
```

...こののためのをします。

```
./manage.py makemigrations discography
```

ステップ2、しいフィールドにをします。これをうには、のをするがあります。

```
./manage.py makemigrations --empty --name transfer_artists discography
```

こののがしたら、レコードをリンクするために `RunPython` を1つする必要があります。この、のようになります。

```
def link_artists(apps, schema_editor):
    Album = apps.get_model('discography', 'Album')
    Artist = apps.get_model('discography', 'Artist')
    for album in Album.objects.all():
        artist, created = Artist.objects.get_or_create(name=album.artist)
        album.artist_link = artist
        album.save()
```

データがしいフィールドにされたので、しい `artist_link` フィールドをして、にすべてをそのまますることができます。または、しクリーンアップをいたいは、2つのマイグレーションをするがあります。

あなたのののために、のフィールド `artist` をしたいでしょう。2のでは、しいフィールド `artist_link` を `artist` します。

これは、Djangoがしくをできるように、のステップでされます。

オンラインでをむ <https://riptutorial.com/ja/django/topic/1200/>

48:

Examples

リスト

のモデルのなmyblogアプリがあるとしましょう

```
from django.conf import settings
from django.utils import timezone

class Article(models.Model):
    title = models.CharField(max_length=70)
    slug = models.SlugField(max_length=70, unique=True)
    author = models.ForeignKey(settings.AUTH_USER_MODEL, models.PROTECT)
    date_published = models.DateTimeField(default=timezone.now)
    is_draft = models.BooleanField(default=True)
    content = models.TextField()
```

Django Adminの「リスト」は、のモデルのすべてのオブジェクトをリストするページです。

```
from django.contrib import admin
from myblog.models import Article

@admin.register(Article)
class ArticleAdmin(admin.ModelAdmin):
    pass
```

デフォルトでは、オブジェクトのをするために、モデルの__str__() メソッドまたはPython2のは__unicode__() unicode __unicode__() をします。これは、あなたがそれをきしなれば、すべてのオブジェクトというのリストをすることをします。このをするには、__str__() メソッドをします。

```
class Article(models.Model):
    def __str__(self):
        return self.title
```

さて、あなたのすべてののは、のをつががありますし、"オブジェクト"よりもです。

ただし、このリストにのデータをすることもできます。そのためには、list_display

```
@admin.register(Article)
class ArticleAdmin(admin.ModelAdmin):
    list_display = ['__str__', 'author', 'date_published', 'is_draft']
```

list_displayは、モデルフィールドとプロパティにされません。あなたのModelAdminメソッドでもあります

```
from django.forms.utils import flatatt
```

```

from django.urls import reverse
from django.utils.html import format_html

@admin.register(Article)
class ArticleAdmin(admin.ModelAdmin):
    list_display = ['title', 'author_link', 'date_published', 'is_draft']

    def author_link(self, obj):
        author = obj.author
        opts = author._meta
        route = '{}_{}_change'.format(opts.app_label, opts.model_name)
        author_edit_url = reverse(route, args=[author.pk])
        return format_html(
            '<a{}>{}</a>', flatatt({'href': author_edit_url}), author.first_name)

    # Set the column name in the change list
    author_link.short_description = "Author"
    # Set the field to use when ordering using this column
    author_link.admin_order_field = 'author__firstname'

```

ページのCSSスタイルとJSスクリプト

シンプルなCustomerモデルがあるとします。

```

class Customer(models.Model):
    first_name = models.CharField(max_length=255)
    last_name = models.CharField(max_length=255)
    is_premium = models.BooleanField(default=False)

```

Djangoにし、フィールドをfirst_nameとlast_nameします

```

@admin.register(Customer)
class CustomerAdmin(admin.ModelAdmin):
    list_display = ['first_name', 'last_name', 'is_premium']
    search_fields = ['first_name', 'last_name']

```

これをすると、フィールドがリストページにされ、デフォルトのプレースホルダ"キーワード"がされます。しかし、そのプレースホルダを「で」にしたいはどうすればよいでしょうか

カスタムJavascriptファイルをadmin Mediaすことでこれをうことができます

```

@admin.register(Customer)
class CustomerAdmin(admin.ModelAdmin):
    list_display = ['first_name', 'last_name', 'is_premium']
    search_fields = ['first_name', 'last_name']

    class Media:
        #this path may be any you want,
        #just put it in your static folder
        js = ('js/admin/placeholder.js', )

```

ブラウザデバッグツールバーをして、DjangoがこのバーにしたIDまたはクラスをつけて、jsコードをくことができます

```
$(function () {  
    $('#searchbar').attr('placeholder', 'Search by name')  
})
```

Mediaクラスでは、オブジェクトをむCSSファイルをすることもできます

```
class Media:  
    css = {  
        'all': ('css/admin/styles.css',)  
    }
```

たとえば、`first_name`のをのがあります。

デフォルトでは、Djangoは`list_display`のテーブルをし、すべての`<td>`タグは`field-`
`'list_display_name'`ようなcssクラスをちますが、このは`field_first_name`

```
.field_first_name {  
    background-color: #e6f2ff;  
}
```

JSやいくつかのCSSスタイルをしてのをカスタマイズしたいは、ブラウザのデバッグツールでid
とコンポーネントのクラスをチェックすることができます。

きなテーブルをするキーの

デフォルトでは、Djangoは`ForeignKey`フィールドを`<select>`としてレンダリングします。これは、
されるテーブルにもものエントリがあると、ページのロードがにくなるがあります。のエントリ
しかないでも、すべてのエントリのからのエントリをすのはにです。

これのためのになモジュールは[django-autocomplete-light](#) DALです。これにより、`<select>`フイ
ールドのわりにオートコンプリートフィールドをすることができます。

Ville:	Paris (75) Par Parcy-et-Tigny (02) Parfondeval (02) Parfondru (02) Parfouru-sur-Odon (14) Parville (27) Pardines (63)
Quartier:	
Code postal:	
Adresse:	

views.py

```
from dal import autocomplete

class CityAutocomp(autocomplete.Select2QuerySetView):
    def get_queryset(self):
        qs = City.objects.all()
        if self.q:
            qs = qs.filter(name__istartswith=self.q)
        return qs
```

urls.py

```
urlpatterns = [
    url(r'^city-autocomp/$', CityAutocomp.as_view(), name='city-autocomp'),
]
```

forms.py

```
from dal import autocomplete

class PlaceForm(forms.ModelForm):
    city = forms.ModelChoiceField(
        queryset=City.objects.all(),
        widget=autocomplete.ModelSelect2(url='city-autocomp')
    )

    class Meta:
        model = Place
```

```
fields = ['__all__']
```

admin.py

```
@admin.register(Place)
class PlaceAdmin(admin.ModelAdmin):
    form = PlaceForm
```

オンラインでをむ <https://riptutorial.com/ja/django/topic/1219/>

49: コマンド

き

コマンドは、Djangoプロジェクトまたはとなるデータベースでアクションをできるでなスクリプトです。さまざまなデフォルトコマンドにえて、のコマンドをくこともです

のPythonスクリプトとして、コマンドフレームワークをすると、なセットアップがにわれます。

コマンドは、のいずれかからびすことができます。

- `django-admin <command> [options]`
- `python -m django <command> [options]`
- `python manage.py <command> [options]`
- `./manage.py <command> [options]` `manage.py`にがある `chmod +x manage.py`

Cronでコマンドをするには

```
*/10 * * * * pythonuser /var/www/dev/env/bin/python /var/www/dev/manage.py <command> [options]
> /dev/null
```

Examples

コマンドのと

Djangoでコマンドラインやのサービスユーザ/リクエストがされていないをとってアクションをするには、`management commands`で`management commands`。

Djangoモジュールは、にじてインポートできます。

コマンドにして、々のファイルをするがあり`myapp/management/commands/my_command.py`
`management`ディレクトリと`commands`ディレクトリにはの`__init__.py`ファイルがです

```
from django.core.management.base import BaseCommand, CommandError

# import additional classes/modules as needed
# from myapp.models import Book

class Command(BaseCommand):
    help = 'My custom django management command'

    def add_arguments(self, parser):
        parser.add_argument('book_id', nargs='+', type=int)
        parser.add_argument('author', nargs='+', type=str)

    def handle(self, *args, **options):
        bookid = options['book_id']
        author = options['author']
```

```
# Your code goes here

# For example:
# books = Book.objects.filter(author="bob")
# for book in books:
#     book.name = "Bob"
#     book.save()
```

class **command**は、 **BaseCommand**またはそのサブクラスの1つをするためにです。

コマンドのは、それをむファイルのです。のでコマンドをするには、プロジェクトディレクトリでのコマンドをします。

```
python manage.py my_command
```

コマンドのにはモジュールのインポートのためにかかることにしてください。したがって、によっては、 `management commands`わりに`daemon`プロセスをすることをおめし `management commands`。

コマンドの

のコマンドのリストをする

なコマンドのリストは、のでできます。

```
>>> python manage.py help
```

あなたがコマンドをしていないか、オプションのをしているなら、このように`-h`をうことができます

```
>>> python manage.py command_name -h
```

ここで`command_name`はあなたのむコマンドです。これはコマンドのヘルプテキストをします。

```
>>> python manage.py runserver -h
>>> usage: manage.py runserver [-h] [--version] [-v {0,1,2,3}]
                                [--settings SETTINGS] [--pythonpath PYTHONPATH]
                                [--traceback] [--no-color] [--ipv6] [--nothreading]
                                [--noreload] [--nostatic] [--insecure]
                                [addrport]

Starts a lightweight Web server for development and also serves static files.

positional arguments:
  addrport              Optional port number, or ipaddr:port

optional arguments:
  -h, --help            show this help message and exit
  --version             show program's version number and exit
  -v {0,1,2,3}, --verbosity {0,1,2,3}
                        Verbosity level; 0=minimal output, 1=normal output,
```

```

--settings SETTINGS      2=verbose output, 3=very verbose output
                          The Python path to a settings module, e.g.
                          "myproject.settings.main". If this isn't provided, the
                          DJANGO_SETTINGS_MODULE environment variable will be
                          used.
--pythonpath PYTHONPATH  A directory to add to the Python path, e.g.
                          "/home/djangoprojects/myproject".
--traceback              Raise on CommandError exceptions
--no-color               Don't colorize the command output.
--ipv6, -6              Tells Django to use an IPv6 address.
--nothreading            Tells Django to NOT use threading.
--noreload               Tells Django to NOT use the auto-reloader.
--nostatic               Tells Django to NOT automatically serve static files
                          at STATIC_URL.
--insecure               Allows serving static files even if DEBUG is False.

```

なコマンドのリスト

manage.pyのわりにdjango-adminをう

manage.pyをりき、わりにdjango-adminコマンドをすることができます。これをうには、manage.pyがでうがあります

- PYTHONPATHにプロジェクトパスをする
- DJANGO_SETTINGS_MODULEをする

```

export PYTHONPATH="/home/me/path/to/your_project"
export DJANGO_SETTINGS_MODULE="your_project.settings"

```

これはpostactivateスクリプトでこれらのできるvirtualenvでにです。

django-adminコマンドは、ファイルシステムのどこにいてもできるというがあります。

みみコマンド

Djangoには、python manage.py [command]か、manage.pyに+xがあるにのみ./manage.py [command]をしてのコマンドが./manage.py [command]ます。は、もにされるもののいくつかです

なすべてのコマンドのリストをする

```
./manage.py help
```

Djangoサーバをlocalhost8000でします。ローカルテストに

```
./manage.py runserver
```

プロジェクトのDjangoがプリロードされたでPythonまたはインストールされているはipythonコンソールをしますこれをわずにPythonでプロジェクトのにアクセスしようとするします。

```
./manage.py shell
```

モデルにえたについて、しいデータベースファイルをします。 [を](#)

```
./manage.py makemigrations
```

のをのデータベースにします。

```
./manage.py migrate
```

プロジェクトのテストスイートをします。 [ユニットテスト](#)をしてください

```
./manage.py test
```

プロジェクトのすべてのファイルを `STATIC_ROOT`、 `STATIC_ROOT` されたフォルダにして、
`STATIC_ROOT` できるようにします。

```
./manage.py collectstatic
```

スーパーユーザーのをします。

```
./manage.py createsuperuser
```

したユーザーのパスワードをします。

```
./manage.py changepassword username
```

[なコマンドのなリスト](#)

[オンラインでコマンドをむ](#) <https://riptutorial.com/ja/django/topic/1661/コマンド>

Examples

タイムゾーンの

Djangoが`settings.py`ファイルでタイムゾーンをすることができます。

```
TIME_ZONE = 'UTC' # use this, whenever possible
TIME_ZONE = 'Europe/Berlin'
TIME_ZONE = 'Etc/GMT+1'
```

ここになタイムゾーンのリストがあります

Windowsでするは、システムのタイムゾーンとじにするがあります。

Djangoにタイムゾーンのをさせたくない

```
USE_TZ = False
```

Djangoのベストプラクティスでは、UTCをしてデータベースにをすることができます。

Webサイトが1つのタイムゾーンでのみなでも、データをUTCにデータベースにすることをめします。なはDSTです。くのでは、DSTのシステムがあり、はとにします。でしているは、がしたに2、エラーがするがあります。

<https://docs.djangoproject.com/en/stable/topics/i18n/timezones/>

へのアクセス

すべてのがしたら、コードでそのをすることになります。これをうには、ファイルにのインポートをします。

```
from django.conf import settings
```

に、`settings`モジュールのとしてにアクセスできます。たとえば、のようになり`settings`。

```
if not settings.DEBUG:
    email_user(user, message)
```

アプリのをするためにBASE_DIRをする

アプリケーションのパスをハードコードするのはいえです。コードがなるマシンでシームレスにできるように、にURLをすることができます。これをするもいは、このようなをすることです

```
import os
BASE_DIR = os.path.dirname(os.path.dirname(__file__))
```

に、この`BASE_DIR`をして、のすべてのをします。

```
TEMPLATE_PATH = os.path.join(BASE_DIR, "templates")
STATICFILES_DIRS = [
    os.path.join(BASE_DIR, "static"),
]
```

々。これにより、することなくなるマシンでコードをできるようになります。

しかし、`os.path`はしです。たとえば、モジュールが`project.settings.dev`、のようにするがあります。

```
BASE_DIR = os.path.dirname(os.path.dirname(os.path.dirname(__file__)))
```

のとして、`unipath`モジュール `pip install unipath` できますをするがあります。

```
from unipath import Path

BASE_DIR = Path(__file__).ancestor(2) # or ancestor(3) if using a submodule

TEMPLATE_PATH = BASE_DIR.child('templates')
STATICFILES_DIRS = [
    BASE_DIR.child('static'),
]
```

をしてサーバーでをする

をすることは、[The Twelve-Factor App](#)にされているように、にじてアプリのをするためにくされています。

デプロイメントでがされるがあるため、アプリケーションのソースコードをりげたり、アプリケーションファイルやソースコードリポジトリのにをせずにをするのはにいです。

Djangoでは、メインはプロジェクトのフォルダの`settings.py`にあります。シンプルなPythonファイルなので、ライブラリのPythonの`os`モジュールをしてにアクセスすることができますそしてなデフォルトをつことさえできます。

settings.py

```
import os

SECRET_KEY = os.environ.get('APP_SECRET_KEY', 'unsafe-secret-key')

DEBUG = bool(os.environ.get('DJANGO_DEBUG', True) == 'False')
```



```
ALLOWED_HOSTS = os.environ.get('DJANGO_ALLOWED_HOSTS', '').split()

DATABASES = {
    'default': {
        'ENGINE': os.environ.get('APP_DB_ENGINE', 'django.db.backends.sqlite3'),
        'NAME': os.environ.get('DB_NAME', 'db.sqlite'),
        'USER': os.environ.get('DB_USER', ''),
        'PASSWORD': os.environ.get('DB_PASSWORD', ''),
        'HOST': os.environ.get('DB_HOST', None),
        'PORT': os.environ.get('DB_PORT', None),
        'CONN_MAX_AGE': 600,
    }
}
```

Djangoではデータベースをすることができるので、sqlite3をマシンでうことができますそして、ソースシステムにコミットするためにはseditがデフォルトです。これはですがおめできません

アプリのデータベース、キューイングシステム、キャッシュなどのバックアップサービスは、dev / prodパリティがなの1つです。 [12のアプリケーション - /プロダクトパリティ](#)

データベースにDATABASE_URLパラメータをするは、 [するを](#)してください。

のをする

Djangoのデフォルトのプロジェクトレイアウトは、つのsettings.pyしsettings.py。これは、のようになるとです。

```
myprojectroot/
  myproject/
    __init__.py
    settings/
      __init__.py
      base.py
      dev.py
      prod.py
      tests.py
```

これにより、、、テストなど、さまざまなですることができます。

デフォルトのレイアウトからこのレイアウトにするとき、のsettings.pyはsettings/base.pyなりsettings/base.py。のすべてのサブモジュールがfrom .base import *まるsettings/base.pyを"サブクラス"するとき。たとえば、 settings/dev.pyはのようになり settings/dev.py。

```
# -*- coding: utf-8 -*-
from .base import * # noqa

DEBUG = True
INSTALLED_APPS.extend([
    'debug_toolbar',
])

EMAIL_BACKEND = 'django.core.mail.backends.console.EmailBackend'
INTERNAL_IPS = ['192.168.0.51', '192.168.0.69']
```

1

django-admin コマンドがしくするためには、DJANGO_SETTINGS_MODULE デフォルトは myproject.settings をするがあり myproject.settings。 は、 myproject.settings.dev し myproject.settings.dev。 プロダクションでは、 myproject.settings.prod し myproject.settings.prod。 virtualenv をするは、 postactivate スクリプトでするのがです。

```
#!/bin/sh
export PYTHONPATH="/home/me/django_projects/myproject:$VIRTUAL_ENV/lib/python3.4"
export DJANGO_SETTINGS_MODULE="myproject.settings.dev"
```

DJANGO_SETTINGS_MODULE によってされていないモジュールをに --settings は、 django-admin --settings オプションをうことができます

```
django-admin test --settings=myproject.settings.tests
```

2

あなたがしておきたいは DJANGO_SETTINGS_MODULE デフォルトので myproject.settings、 あなたはにえることができ settings あなたにインポートをくことで、ロードするモジュールを __init__.py ファイル。

のでは、 __init__.py することでじがられます。

```
from .dev import *
```

のファイルの

ファイルは、ファイルのとするがあります。は、 のをしてむをしてください。

```
django project
├── config
│   ├── __init__.py
│   ├── requirements
│   │   ├── base.txt
│   │   ├── dev.txt
│   │   ├── test.txt
│   │   └── prod.txt
│   └── settings
└── manage.py
```

base.txt ファイルでは、すべてののでされるをします。

```
# base.txt
```

```
Django==1.8.0
psycopg2==2.6.1
jinja2==2.8
```

そののすべてのファイルでは、`-r base.txt` ベースを `-r base.txt`、のになのをします。

```
# dev.txt
-r base.txt # includes all dependencies in `base.txt`

# specific dependencies only used in dev env
django-queryinspect==0.1.0
```

```
# test.txt
-r base.txt # includes all dependencies in `base.txt`

# specific dependencies only used in test env
nose==1.3.7
django-nose==1.4
```

```
# prod.txt
-r base.txt # includes all dependencies in `base.txt`

# specific dependencies only used in production env
django-queryinspect==0.1.0
gunicorn==19.3.0
django-storages-redux==1.3
boto==2.38.0
```

に、をインストールします。 `dev env` の `pip install -r config/requirements/dev.txt`

JSON ファイルをしてデータをす

Git や SVN などの VCS をする、リポジトリがされているかプライベートであるかにかかわらずバージョンされてはならないのデータがあります。

これらのデータの中には、`SECRET_KEY` とデータベースパスワードがあります。

これらのをバージョンからすなは、プロジェクトのルートに `secrets.json` というファイルをするこ
とです [このアイデアについては、"Two Scoops of Django" にします](#)

```
{
  "SECRET_KEY": "N4HE:AMk:.Ader5354DR453TH8SHTQr",
  "DB_PASSWORD": "v3ry53cr3t"
}
```

それをリストにします `.gitignore` for git

```
*.py[co]
*.sw[po]
*~
/secrets.json
```

に、のを `settings` モジュールにします。

```
import json
import os
from django.core.exceptions import ImproperlyConfigured

with open(os.path.join(BASE_DIR, 'secrets.json')) as secrets_file:
    secrets = json.load(secrets_file)

def get_secret(setting, secrets=secrets):
    """Get secret setting or fail with ImproperlyConfigured"""
    try:
        return secrets[setting]
    except KeyError:
        raise ImproperlyConfigured("Set the {} setting".format(setting))
```

に、をのようにします。

```
SECRET_KEY = get_secret('SECRET_KEY')
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgres',
        'NAME': 'db_name',
        'USER': 'username',
        'PASSWORD': get_secret('DB_PASSWORD'),
    },
}
```

クレジット [Djangoの2つのスクリプトDaniel Roy GreenfeldとAudrey Roy GreenfeldによるDjango 1.8のベストプラクティス2015 2スクリプトプレスISBN 978-0981467344](#)

からの **DATABASE_URL** の

HerokuなどのPaaSサイトでは、のパラメータホスト、ポート、ユーザー、パスワードなどではなく、のURLとしてデータベースをけるのがです。

モジュール `dj_database_url` あり、DjangoにデータベースをするのにしたPythonに **DATABASE_URL** をにします。

```
import dj_database_url

if os.environ.get('DATABASE_URL'):
    DATABASES['default'] =
        dj_database_url.config(default=os.environ['DATABASE_URL'])
```

オンラインでもむ <https://riptutorial.com/ja/django/topic/942/>

51: バックエンド

Examples

メールバックエンド

Djangoのデフォルトは、`username`と`password`フィールドで`username`。メールバックエンドは、`email`と`password`についてユーザーをし`password`。

```
from django.contrib.auth import get_user_model

class EmailBackend(object):
    """
    Custom Email Backend to perform authentication via email
    """
    def authenticate(self, username=None, password=None):
        user_model = get_user_model()
        try:
            user = user_model.objects.get(email=username)
            if user.check_password(password): # check valid password
                return user # return user to be authenticated
        except user_model.DoesNotExist: # no matching user exists
            return None

    def get_user(self, user_id):
        user_model = get_user_model()
        try:
            return user_model.objects.get(pk=user_id)
        except user_model.DoesNotExist:
            return None
```

このバックエンドを`AUTHENTICATION_BACKENDS`にします。

```
# settings.py
AUTHENTICATION_BACKENDS = (
    'my_app.backends.EmailBackend',
    ...
)
```

オンラインでバックエンドをむ <https://riptutorial.com/ja/django/topic/1282/バックエンド>

Examples

GunicornでDjangoアプリケーションをする

1. ガンコンをインストールする

```
pip install gunicorn
```

2. djangoプロジェクトフォルダmanage.pyがあるフォルダとしフォルダから、のコマンドをして、のdjangoプロジェクトをgunicornでします

```
gunicorn [projectname].wsgi:application -b 127.0.0.1:[port number]
```

--env オプションをして、をロードするパスをすることができます

```
gunicorn --env DJANGO_SETTINGS_MODULE=[projectname].settings [projectname].wsgi
```

または-D オプションをしてデーモンプロセスとしてする

3. gunicornがにすると、コンソールにのがされます

```
Starting gunicorn 19.5.0
```

```
Listening at: http://127.0.0.1:[port number] ([pid])
```

.... そののガーコンサーバーにする

Herokuでする

1. [Heroku Toolbelt](#)をダウンロードしてください。
2. Djangoアプリケーションのソースのルートにします。あなたはtkがです
3. heroku create [app_name] ます。アプリをししないと、Herokuはランダムにします。アプリのURLはhttp://[app name].herokuapp.com
4. Procfile というのテキストファイルをします。にをれないでください。

```
web: <bash command to start production server>
```

ワーカープロセスがあるは、することもできます。ののをします worker-name:
<bash command to start worker>

5. requirements.txtをします。

- をしているは、 pip freeze > requirements.txt し pip freeze > requirements.txt
- それのは、[をしてください](#)。なPythonパッケージをでリストすることもできますが、この

チュートリアルではありません。

6. です

1. `git push heroku master`

HerokuはデプロイメントをうためにgitリポジトリまたはDropboxフォルダがです。わりに、heroku.com GitHubリポジトリからリロードをすることもできますが、このチュートリアルではしません。

2. `heroku ps:scale web=1`

これにより、ウェブの「dynos」のが1になります。あなたはダイノスについてもっととることができます。

3. `heroku open`か、`http://app-name.herokuapp.com` `heroku open`します

ヒント `heroku open`は、デフォルトブラウザのherokuアプリへのURLをきます。

7. アドオンをします。 Djangoアプリケーションを、Herokuでされているデータベースを「アドオン」としてバインドするようにするがあります。このではこれについてはしませんが、Herokuでデータベースをするのパイプラインのもあります。

シンプルなりモート **fabfile.py**

*Fabric*は、*Python2.5-2.7*ライブラリとコマンドラインツールで、アプリケーションのデプロイメントやシステムのためのSSHのをします。コマンドラインからの*Python*をすることができます。

`pip install fabric`インストールファブリックをしてファブリックを `pip install fabric` ルートディレクトリに `fabfile.py`をします。

```
#myproject/fabfile.py
from fabric.api import *

@task
def dev():
    # details of development server
    env.user = # your ssh user
    env.password = #your ssh password
    env.hosts = # your ssh hosts (list instance, with comma-separated hosts)
    env.key_filename = # pass to ssh key for github in your local keyfile

@task
def release():
    # details of release server
    env.user = # your ssh user
    env.password = #your ssh password
    env.hosts = # your ssh hosts (list instance, with comma-separated hosts)
    env.key_filename = # pass to ssh key for github in your local keyfile

@task
def run():
    with cd('path/to/your_project/')
```

```
with prefix('source ../env/bin/activate'):
# activate venv, suppose it appear in one level higher
# pass commands one by one
run('git pull')
run('pip install -r requirements.txt')
run('python manage.py migrate --noinput')
run('python manage.py collectstatic --noinput')
run('touch reload.txt')
```

ファイルをするには、に `fab` コマンドをし `fab`。

```
$ fab dev run # for release server, `fab release run`
```

githubのsshキーをすることはできず、ログインとパスワードはでするだけですが、`fabfile`はキーと同じようにします。

Heroku Django スターターテンプレートをする。

HerokuであなたのDjango Webサイトをホストするのは、Heroku Django Starter Templateをしてプロジェクトをできます

```
django-admin.py startproject --template=https://github.com/heroku/heroku-django-
template/archive/master.zip --name=Procfile YourProjectName
```

ファイル、データベース、Gunicornなどのプロダクションのと、WhiteNoiseによるDjangoのファイルのをえています。これはあなたのをします。それはHerokuでのホスティングのためのオールレディです。このテンプレートのにあなたのウェブサイトをするだけです

Herokuでこのテンプレートをするには

```
git init
git add -A
git commit -m "Initial commit"

heroku create
git push heroku master

heroku run python manage.py migrate
```

それでおしまい

Django デプロイメントの。 Nginx + Gunicorn + Supervisor on LinuxUbuntu

3つのなツール。

1. nginxフリー、オープンソース、HTTPサーバとリバースプロキシ、。
2. gunicorn - 'Green Unicorn'はUNIXのPython WSGI HTTPサーバですサーバのにです。
3. スーパーバイザー - ユーザーがUNIXライクなオペレーティングシステムのくのプロセスをおよびできるようにするクライアント/サーバシステム。あなたがアプリやシステムクラ

ツシュに、あなたのdjango /セロリ/セロリカムなどをします。

にするために、あなたのアプリが/home/root/app/src/ディレクトリにあるとして、rootユーザーをしますただし、あなたのアプリケーションにのユーザーをするがあります。また、は/home/root/app/env/ pathにあります。

NGINX

nginxからめましょう。nginxがまだマシンにないは、`sudo apt-get install nginx`。でnginxディレクトリ/etc/nginx/sites-enabled/yourapp.conf しいファイルをするがあります。default.confというのファイルがあるはします。

ソケットファイルをしてサービスをしようとするnginx confファイルのペローコード。でガンコーンのがあります。ソケットファイルは、nginxとgunicornの इसलिएにここでされます。また、ポートをしてうこともできます。

```
# your application name; can be whatever you want
upstream yourappname {
    server          unix:/home/root/app/src/gunicorn.sock fail_timeout=0;
}

server {
    # root folder of your application
    root            /home/root/app/src/;

    listen          80;
    # server name, your main domain, all subdomains and specific subdomains
    server_name     yourdomain.com *.yourdomain.com somesubdomain.yourdomain.com

    charset         utf-8;

    client_max_body_size              100m;

    # place where logs will be stored;
    # folder and files have to be already located there, nginx will not create
    access_log      /home/root/app/src/logs/nginx-access.log;
    error_log       /home/root/app/src/logs/nginx-error.log;

    # this is where your app is served (gunicorn upstream above)
    location / {
        uwsgi_pass  yourappname;
        include     uwsgi_params;
    }

    # static files folder, I assume they will be used
    location /static/ {
        alias       /home/root/app/src/static/;
    }

    # media files folder
    location /media/ {
        alias       /home/root/app/src/media/;
    }
}
```

GUNICORN

、たちのGUNICORNスクリプトは、サーバーでdjangoアプリケーションをするがあります。まず、ガンコンをインストールして、に `pip install gunicorn`。

```
#!/bin/bash

ME="root"
DJANGODIR=/home/root/app/src # django app dir
SOCKFILE=/home/root/app/src/gunicorn.sock # your sock file - do not create it manually
USER=root
GROUP=webapps
NUM_WORKERS=3
DJANGO_SETTINGS_MODULE=yourapp.yoursettings
DJANGO_WSGI_MODULE=yourapp.wsgi
echo "Starting $NAME as `whoami`"

# Activate the virtual environment
cd $DJANGODIR

source /home/root/app/env/bin/activate
export DJANGO_SETTINGS_MODULE=$DJANGO_SETTINGS_MODULE
export PYTHONPATH=$DJANGODIR:$PYTHONPATH

# Create the run directory if it doesn't exist
RUNDIR=$(dirname $SOCKFILE)
test -d $RUNDIR || mkdir -p $RUNDIR

# Start your Django Gunicorn
# Programs meant to be run under supervisor should not daemonize themselves (do not use --
daemon)
exec /home/root/app/env/bin/gunicorn ${DJANGO_WSGI_MODULE}:application \
    --name root \
    --workers $NUM_WORKERS \
    --user=$USER --group=$GROUP \
    --bind=unix:$SOCKFILE \
    --log-level=debug \
    --log-file=--
```

gunicornのスクリプトをできるようにするには、モードをにするがあります

```
sudo chmod u+x /home/root/app/src/gunicorn_start
```

、あなただけのあなたのgunicornサーバーをすることができるようになります `./gunicorn_start`

にべたように、たちはスーパーバイザーがしたときにアプリケーションをさせたいとしています。
。スーパーバイザがまだ `sudo apt-get install supervisor` でサーバにインストールされていない `sudo apt-get install supervisor`。

にスーパーバイザーをインストールします。に、`.conf`メインディレクトリのファイルを `/etc/supervisor/conf.d/your_conf_file.conf`

ファイルの

```
[program:yourappname]
command = /home/root/app/src/gunicorn_start
user = root
stdout_logfile = /home/root/app/src/logs/gunicorn_supervisor.log
redirect_stderr = true
```

クイックブリーフ、に `[program:yourappname]` がです。これはたちのになります。また、`stdout_logfile` は、ログがアクセスとエラーのにされるファイルです。

それがしたら、にしいファイルをしたことをえるがあります。これをうために、なるUbuntuのバージョンのためのなるプロセスがあります。

Ubuntu version 14.04 or lesserのは、のコマンドをしてください

`sudo supervisorctl reread` ->スーパーバイザカタログのすべてのファイルをみします。これはのよ
うにされます **yourappnameavailable**

`sudo supervisorctl update` ->スーパーバイザーをしくされたファイルにします。 **yourappname** プ
ロセスグループをしてください

Ubuntu 16.04

```
sudo service supervisor restart
```

あなたのアプリがしくしているかどうかをするために

```
sudo supervisorctl status yourappname
```

これがされます

```
yourappname RUNNING pid 18020, uptime 0:00:50
```

こののライブデモンストレーションをするには、この[ビデオ](#)をてください。

apache / nginxをせずにローカルにデプロイする

コンテンツをするためにApache / Nginxをするためのプロダクションデプロイメントの。したが
って、`DEBUG`がfalseの、およびメディアのはロードされません。ただし、アプリケーションで
Apache / Nginxサーバーをしなくても、コンテンツをデプロイメントにみむことができます。

```
python manage.py runserver --insecure
```

これはローカルLANなどのみをしており、`staticfiles`はしないでください`staticfiles`アプリケ
ーションがプロジェクトの`INSTALLED_APPS`にあるにのみできます。

オンラインでをむ <https://riptutorial.com/ja/django/topic/2792/>

53: タスクセロリ

Celeryは、バックグラウンドジョブやスケジューリングされたジョブをし、Djangoとによくできるタスクキューです。セロリには、メッセージブローカーと呼ばれるかが、びしからワーカーにメッセージをすがあります。このメッセージブローカーはredis、rabbitmq、またはDjango ORM / dbでもかまいませんが、これはされるではありません。

サンプルをめるに、セロリをするがあります。セロリをするには、メインアプリでsettings.py ファイルとしてcelery_config.py ファイルをします。

```
from __future__ import absolute_import
import os
from celery import Celery
from django.conf import settings

# broker url
BROKER_URL = 'redis://localhost:6379/0'

# Indicate Celery to use the default Django settings module
os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'config.settings')

app = Celery('config')
app.config_from_object('django.conf:settings')
# if you do not need to keep track of results, this can be turned off
app.conf.update(
    CELERY_RESULT_BACKEND=BROKER_URL,
)

# This line will tell Celery to autodiscover all your tasks.py that are in your app folders
app.autodiscover_tasks(lambda: settings.INSTALLED_APPS)
```

メインアプリの__init__.py ファイルでセロリのアプリケーションをインポートします。このような

```
# -*- coding: utf-8 -*-
# Not required for Python 3.
from __future__ import absolute_import

from .celery_config import app as celery_app # noqa
```

celery workerをするには、manage.pyがあるレベルでこのコマンドをします。

```
# pros is your django project,
celery -A proj worker -l info
```

Examples

2つのをするな

するには

1. セロリ `pip install celery` セロリを `pip install celery`
2. セロリをするにかう

```
from __future__ import absolute_import, unicode_literals

from celery.decorators import task

@task
def add_number(x, y):
    return x + y
```

`.delay()` メソッドをすると、これをにできます。

`add_number.delay(5, 10)` と10は、`add_number` です

がをしたかどうかをべるには、`delay` メソッドによってされたオブジェクトにして `.ready()` をできます。

のをフェッチするには、オブジェクトの `.result` をします。

```
async_result_object = add_number.delay(5, 10)
if async_result_object.ready():
    print(async_result_object.result)
```

オンラインでタスクセロリをむ <https://riptutorial.com/ja/django/topic/5481/タスク-セロリ->

クレジット

S. No		Contributors
1	Djangoをいめる	A. Raza , Abhishek Jain , Aidas Bendoraitis , Alexander Tyapkov , Ankur Gupta , Anthony Pham , Antoine Pinsard , arifin4web , Community , e4c5 , elbear , ericdwang , ettanany , Franck Dernoncourt , greatwolf , ilse2005 , Ivan Semochkin , J F , Jared Hooper , John , John Moutafis , JRodDynamite , Kid Binary , knbk , Louis , Luis Alberto Santana , Ixaer , maciek , McAbra , MiniGunnR , mnoronha , Nathan Osman , naveen.panwar , nhydock , Nikita Davidenko , nouλλdλzελQ , Rahul Gupta , rajarshig , Ron , ruddra , sarvajeetsuman , shacker , ssice , Stryker , techydesigner , The Brewmaster , Thereissoupinmyfly , Tom , WesleyJohnson , Zags
2	ArrayField - PostgreSQLのフィールド	Antoine Pinsard , e4c5 , nouλλdλzελQ
3	Cookiecutterで Djangoをうには	Atul Mishra , nouλλdλzελQ , OliPro007 , RamenChef
4	Django Rest Framework	The Brewmaster
5	Django-CachingバックエンドでのRedisの	Majid , The Brewmaster
6	Djangoコマンドラインから。	e4c5 , OliPro007
7	Djangoとソーシャルネットワーク	Aidas Bendoraitis , aisflat439 , Carlos Rojas , Ivan Semochkin , Rexford , Simplans
8	DjangoのCRUD	aisflat439 , George H.
9	djangoのをリセットする	Cristus Cleetus
10	F	Antoine Pinsard , John Moutafis , Linville , Omar Shehata , RamenChef , Roald Nefs
11	Formsets	naveen.panwar

12	HStoreField - へのマッピング - PostgreSQLのフィールド	CrazyPythou
13	JSONField - PostgreSQLのフィールド	Antoine Pinsard , Daniil Ryzhkov , Matthew Schinckel , noufAlAzar , Omar Shehata , techydesigner
14	RangeFields - PostgreSQLのフィールドのグループ	Antoine Pinsard , CrazyPythou
15	URLルーティング	knbk
16	カスタムマネージャとクエリセット	abidibo , knbk , sudshekhar , Trivial
17	クエリーセット	Antoine Pinsard , Brian Artschwager , Chalist , coffee-grinder , DataSwede , e4c5 , Evans Murithi , George H. , John Moutafis , Justin , knbk , Louis Barranqueiro , Maxime Lorant , MicroPyramid , nima , ravigadila , Sanyam Khurana , The Brewmaster
18	クラスベースのビュー	Antoine Pinsard , Antwane , coffee-grinder , e4c5 , gkr , knbk , maciek , masnun , Maxime Lorant , nicorellius , pleasedontbelong , Pureferret
19	コンテキストプロセッサ	Antoine Pinsard , Brian Artschwager , Dan Russell , Daniil Ryzhkov , fredley
20	ジェンキンスとのな	pnovotnak
21	ジャンゴフィルタ	4444 , Ahmed Atalla
22	スーパーバイザでセロリをべる	RéÑjith , sebb
23	セキュリティ	Antoine Pinsard , knbk
24	データベーストランザクション	Ian Clark
25	データベースのセットアップ	Ahmad Anwar , Antoine Pinsard , Evans Murithi , Kid Binary , knbk , Ixxer , Majid , Peter Mortensen
26	データベースルータ	fredley , knbk

27	デバッグ	Antoine Pinsard , Ashutosh , e4c5 , Kid Binary , knbk , Sayse , Udi
28	テンプレート	Adam Starrh , Alasdair , Aniket , Antoine Pinsard , Brian H. , coffee-grinder , doctorsherlock , fredley , George H. , gkr , lser , Stephen Leppik , Zags
29	テンプレートタグとフィルタ	Antoine Pinsard , irakli khitarishvili , knbk , Medorator , naveen.panwar , The_Cthulhu_Kid
30	ビュー	ettanany , HorsePunchKid , John Moutafis
31	フォーム	Aidas Bendoraitis , Antoine Pinsard , Daniel Rucci , ettanany , George H. , knbk , NBajanca , nicorellius , RamenChef , rumman0786 , sudshekhar , trpt4him
32	フォームウィジェット	Antoine Pinsard , ettanany
33	プロジェクトの	Antoine Pinsard , naveen.panwar , nicorellius
34	ミドルウェア	AlvaroAV , Antoine Pinsard , George H. , knbk , lser , nhydock , Omar Shehata , Peter Mortensen , Trivial , William Reed
35	メタドキュメンテーションのガイドライン	Antoine Pinsard
36	モデル	Aidas Bendoraitis , Alireza Aghamohammadi , alonisser , Antoine Pinsard , aquasan , Arpit Solanki , atomh33ls , coffee-grinder , DataSwede , ettanany , Gahan , George H. , gkr , Ivan Semochkin , Jamie Cockburn , Joey Wilhelm , kcrk , knbk , Linville , lser , maazza , Matt Seymour , MuYi , Navid777 , nhydock , noufalyzaz , pbaranay , PhoebeB , Rana Ahmed , Saksow , Sanyam Khurana , scriptmonster , Selcuk , SpiXel , sudshekhar , techydesigner , The_Cthulhu_Kid , Utsav T , waterproof , zurfyx
37	モデルフィールドリファレンス	Burhan Khalid , Husain Basrawala , knbk , Matt Seymour , Rod Xavier , scriptmonster , techydesigner , The_Cthulhu_Kid
38	モデル	Ian Clark , John Moutafis , ravigadila
39	ユーザーモデルのまたは	Antoine Pinsard , Jon Clements , mnoronha , Raito , Rexford , rigdonmr , Rishabh Agrahari , Roald Nefs , techydesigner , The_Cthulhu_Kid
40	ユニットテスト	Adrian17 , Antoine Pinsard , e4c5 , Kim , Matthew Schinckel , Maxime Lorant , Patrik Stenmark , SandroM , sudshekhar , Zags
41	ロギング	Antwane , Brian Artschwager , RamenChef

42	なビュー	nikolas-berlin
43		Antoine Pinsard , e4c5 , Hetdev , John Moutafis , Majid , nhydock , Rexford
44		Antoine Pinsard , dmvrtx
45	の	Antoine Pinsard , e4c5 , knbk , Kostronor
46		William Reed
47		Antoine Pinsard , engineercoding , Joey Wilhelm , knbk , MicroPyramid , ravigadila , Roald Nefs
48		Antoine Pinsard , coffee-grinder , George H. , Ivan Semochkin , no q1YdzpQ , ssice
49	コマンド	Antoine Pinsard , aquasan , Brian Artschwager , HorsePunchKid , Ivan Semochkin , John Moutafis , knbk , Ixxer , MarZab , Nikolay Fominyh , pbaranay , ptim , Rana Ahmed , techydesigner , Zags
50		allo , Antoine Pinsard , Brian Artschwager , fredley , J F , knbk , Louis , Louis Barranqueiro , Ixxer , Maxime Lorant , NBajanca , Nils Werner , ProfSmiles , RamenChef , Sanyam Khurana , Sayse , Selcuk , SpiXel , ssice , sudshekhar , Tema , The Brewmaster
51	バックエンド	knbk , Rahul Gupta
52		Antoine Pinsard , Arpit Solanki , CodeFanatic23 , I Am Batman , Ivan Semochkin , knbk , Ixxer , Maxime S. , MaxLunar , Meska , no q1YdzpQ , rajarshig , Rishabh Agrahari , Roald Nefs , Rohini Choudhary , sebb
53	タスクセロリ	iankit , Mevin Babu