

Djangoを最速でマスターする part2

Django Python3

この記事は最終更新日から1年以上が経過しています。

進捗

2017/11/3 現在

第1章 完成

第2章 完成

第3章 完成

今回やること

前回の記事で作ったDjangoアプリを拡張させて行きます。

これからの拡張の具体的な内容としては、

	(前半) Django自体の説明	(後半) Django自体の説明 + DockerやMySQLなどとの連携
1章	Django Debug Toolbarの導入	DjangoアプリをDocker上で動かしてみる
2章	ログイン機能の実装	MySQLを使用する
3章	Hijack機能の実装	SQL文の効率化と高速化 (Django組み込み関数を使う)
4章	カスタムtag・カスタムfilter	SQL文の効率化と高速化 (自作関数を使う)
5章	Form	Ajaxを使った高度な通信
6章	自動テストを書こう	(追加予定あり)

といった感じで進めて行こうと思います。

前後半かけて、重箱のすみをつついたような部分まで解説することを目指します。

一回目の記事とは違い、**随時更新**という形で公開していこうと思っています。

なので、**説明のリクエストとかを受けますので、希望があればコメントでお知らせください。**

時間と能力の可能な限り対応します。

また、章の内容によっては作っているサイトのデータの的に合わない場合があるので、その時はドキュメントを詳しく説明します。

1章 Django Debug Toolbarの導入

Django自体の機能とは関係ないですが、これがあるのとないのでは開発の効率が全然違ってくるので導入することをオススメします。

* 公式ドキュメント

上の公式ドキュメントに従いながら導入して行きます。

まずはインストールしてきます。

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

以下をurls.pyに追加してください（前回分とのマージ結果は[Githubのレポジトリ](#)を参考にしてください！）

urls.py

```
from django.conf import settings
from django.conf.urls import include, url

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

settings.py で DEBUG=True となっている時だけDebug Toolbarが現れるようになります。

settings.pyでDjangoにインストールしたアプリを教えてあげましょう。

settings.py

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
```

```
'django.contrib.messages',  
'django.contrib.staticfiles', # これがあることを確認（なければ  
'debug_toolbar', # 追加部分  
'manager',  
]
```

Debug Toolbarをミドルウェアの中に設置します。

TODO

おそらくMIDDLEWAREを変更しただけでいいはずなのですが、MIDDLEWARECLASSにも追加しないとDebug Toolbarが現れてくれなかった。

その説明を書きたい（でもなぜかわからないので、分かる方、ご教授ください）。

settings.py

```
MIDDLEWARE = [  
    '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',
'django.middleware.security.SecurityMiddleware',
'debug_toolbar.middleware.DebugToolbarMiddleware', # 追加
]

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',
    'debug_toolbar.middleware.DebugToolbarMiddleware', # 追加
]
```

あとは Debug Toolbarが現れるように settings.py に以下を追加します。

settings.py

```
# Debug Toolbar
```

```
DEBUG = True
```

```
if DEBUG:

    INTERNAL_IPS = ['127.0.0.1', 'localhost']

def custom_show_toolbar(request):
    return True

DEBUG_TOOLBAR_PANELS = [
    'debug_toolbar.panels.timer.TimerPanel',
    'debug_toolbar.panels.request.RequestPanel',
    'debug_toolbar.panels.sql.SQLPanel',
    'debug_toolbar.panels.templates.TemplatesPanel',
    'debug_toolbar.panels.cache.CachePanel',
    'debug_toolbar.panels.logging.LoggingPanel',
]

DEBUG_TOOLBAR_CONFIG = {
    'INTERCEPT_REDIRECTS': False,
    'SHOW_TOOLBAR_CALLBACK': custom_show_toolbar,
    'HIDE_DJANGO_SQL': False,
    'TAG': 'div',
    'ENABLE_STACKTRACES': True,
}
```

これで設定は完了です。

あとは Debug Toolbarのstaticファイルを落としてきます。

master directory（`manage.py` が置いてあるディレクトリを勝手にこう呼んでます。以下同様。）で以下を実行してstaticファイルを落としてください。

```
$ python manage.py collectstatic
```

そうすると、`assets` というディレクトリができる（僕がみた説明ではstaticの中にできていましたが、そうなるように設定してるっぽかったので、多分デフォルトではassetsの中にできるんじゃないかと思います）ので、その中にある `debug_toolbar` というフォルダをごっそりstatic配下にコピーしましょう。

`assets`ディレクトリは不要なので、コピーしたあとは消してしまいましょう。

以上でDjango Debug Toolbarの設定が完了しました。実際に表示されるか見てみましょう。

Hide ▶

Time

CPU: 6596.83ms
(6480.69ms)

Request

WorkerListView

SQL

1001 queries in 94.77ms

Templates

worker_list.html

Cache

0 calls in 0.00ms

Logging

0 messages

Edit Help

Show 20 entries

Search:

ID	名前	性別	誕生日
1	person0	0	Dec. 31, 1979, 3 p.m.
2	person1	0	Feb. 1, 1981, 3 p.m.
3	person2	0	March 2, 1982, 3 p.m.
4	person3	0	April 3, 1983, 3 p.m.
5	person4	0	May 4, 1984, 3 p.m.
6	person5	0	June 5, 1985, 3 p.m.
7	person6	0	July 6, 1986, 3 p.m.
8	person7	0	Aug. 7, 1987, 3 p.m.
9	person8	0	Sept. 8, 1988, 3 p.m.
10	person9	0	Oct. 9, 1989, 3 p.m.
11	person10	0	Nov. 10, 1990, 3 p.m.
12	person11	0	Dec. 11, 1991, 3 p.m.
13	person12	0	Jan. 12, 1992, 3 p.m.
14	person13	0	Feb. 13, 1993, 3 p.m.
15	person14	0	March 14, 1994, 3 p.m.
16	person15	0	April 15, 1995, 3 p.m.
17	person16	0	May 16, 1996, 3 p.m.
18	person17	0	June 17, 1997, 3 p.m.
19	person18	0	July 18, 1998, 3 p.m.
20	person19	0	Aug. 19, 1999, 3 p.m.

Showing 1 to 20 of 1,000 entries

こんな感じに表示できました。

/worker_list/ のページを見ていますが、表示にかなりの時間がかかっていることがわかります。

SQLが1001回も投げられていて、表示に6.48秒もかかっています（クソサイトですね！！笑）

SQLのところをクリックして、実際どんなクエリが投げられているのか見てみましょう。

(多く投げられていそうなところの+をクリックすると中身が見れます)

The screenshot shows the Django Debug Toolbar's SQL panel. The title is "SQL queries from 1 connection". It displays a query that was executed 1001 times, resulting in 1000 duplicates. The query is a SELECT statement from the 'manager_person' table, filtered by 'id = 201'. The panel also shows the connection details, the view being rendered ('worker_list.html'), and the template name ('worker_list.html'). The right sidebar shows the toolbar's navigation menu with options like Time, Request, SQL, Templates, Cache, and Logging.

Query	Timeline	Time (ms)	Action
SELECT ... FROM "manager_person"		0.81	[Sel] [Exp]
SELECT "manager_person"."id", "manager_person"."name", "manager_person"."birthday", "manager_person"."sex", "manager_person"."address_from", "manager_person"."current_address", "manager_person"."email" FROM "manager_person" WHERE "manager_person"."id" = '201'		0.22	[Sel] [Exp]

Connection: default
/Users/kentaroooura/Library/Mobile Documents/com-apple-CloudDocs/dev/django/manager_project/manager/views.py in get(18):
return render(self.request, self.template_name, context)

```
27         {% for worker in workers %}
28         <tr>
29             <td>{{worker.id}}</td>
30             <td>{{worker.person.name}}</td>
31             <td>{{worker.person.sex}}</td>
32             <td>{{worker.person.birthday}}</td>
33         </tr>
34         {% endfor %}
```

/Users/kentaroooura/Library/Mobile Documents/com-apple-CloudDocs/dev/django/manager_project/manager/templates/worker_list.html

Query	Timeline	Time (ms)	Action
SELECT ... FROM "manager_person" WHERE "manager_person"."id" = '202'		0.13	[Sel]

後半に説明する予定なので、今回は省略しますが、あまりにも遅いので解決策だけ紹介します。

views.py でクエリを作成するところを以下のように変更してください。

views.py

```
class WorkerListView(TemplateView):
    template_name = "worker_list.html"

    def get(self, request, *args, **kwargs):
        context = super(WorkerListView, self).get_context_data

        workers = Worker.objects.all().select_related('person')
        context['workers'] = workers
```

```
return render(self.request, self.template_name, contex
```

これでリロードして見ましょう。

今度は投げられるクエリの数が増え、表示までの時間は0.25秒に短縮されましたね！

典型的なN+1問題という感じですね！

2章 ログイン機能の実装

ウェブアプリには欠かせないログイン機能を実装していきます。

* 公式ドキュメント

`models.py` を根本的に結構いじるので、エラーが出るのを防ぐため、前回作成したマイグレーションファイルとデータベースを消去しておきましょう。

（今回はPersonをログインできるようにして、色々変更しているため、特別です。でも、データベースを一掃したいときなど、たまにあるかもしれないので、やり方を紹介します。）

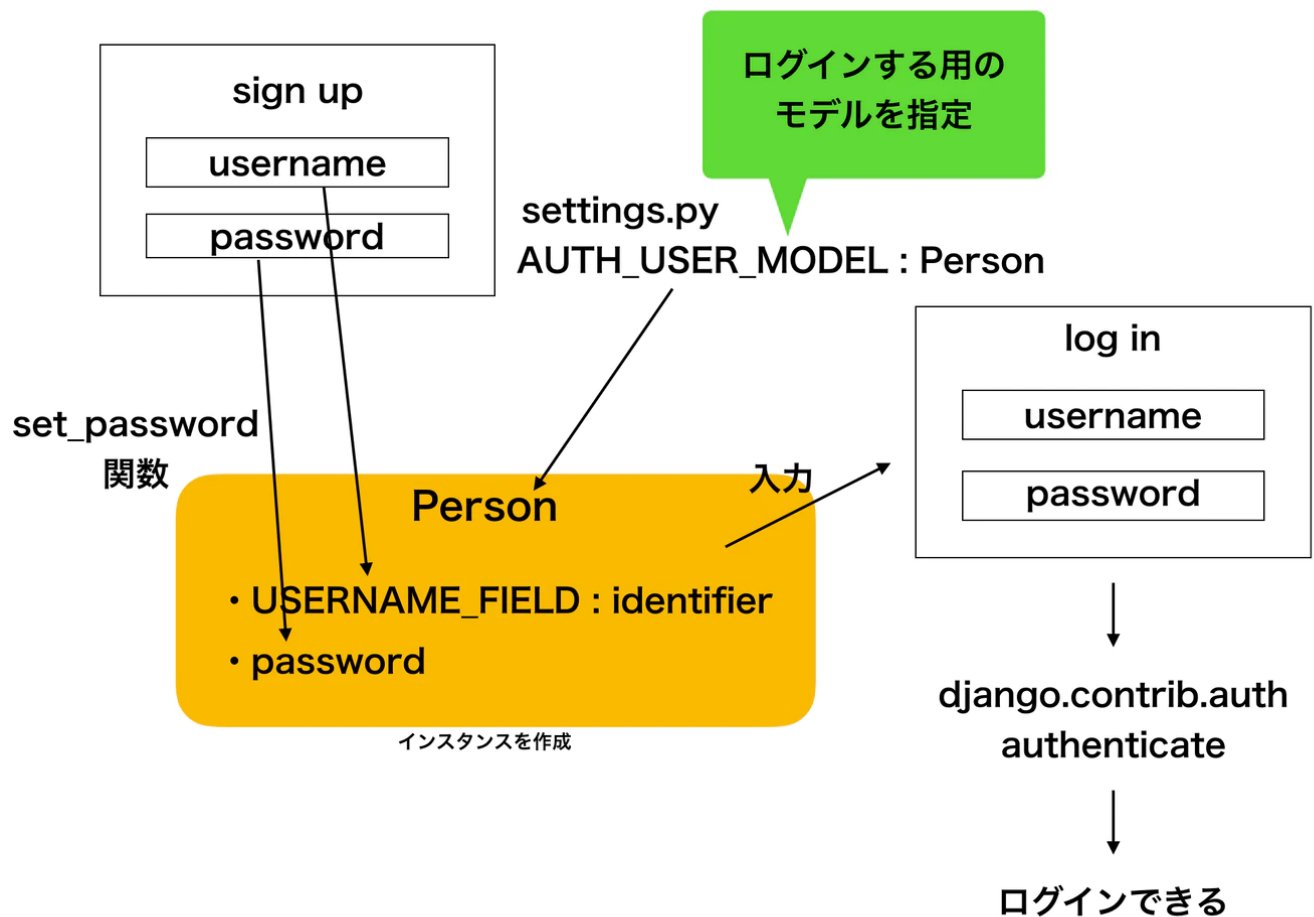
とは言っても、Djangoの標準ではsqlite3を使っているので、ファイルを削除するだけで大丈夫です！

manager/migrations/ 配下の `__init__.py` 以外のファイルを全部と、 `db.sqlite3` というファイルを消してください。

これで完了です！

早速ログイン機能を実装していきましょう！

概要図



概要は上の図を参考にしてください！

ログイン機能の実装

まずはPersonモデルを変えていきます。

コードは重要なところをピックアップして書いているので、全体が見たい方は僕のGitHubのレポジトリを見てください！

* GitHubのコードはこちら

models.py

```
from django.contrib.auth.models import AbstractBaseUser
from manager.managers import PersonManager

class Person(AbstractBaseUser): #1
    objects = PersonManager() # 2

    identifier = models.CharField(max_length=64, unique=True, |
    name = models.CharField(max_length=128)
    email = models.EmailField()

    is_active = models.BooleanField(default=True) # 必要です！

    USERNAME_FIELD = 'identifier' # 4
```

- 1) ログイン用に、AbstractBaseUserを継承します
- 2) Person.objects.create()の時にもちゃんと作れるように定義します（PersonManagerは下で書きます）
- 3) アカウント名のカラムを追加します

4) usernameというカラムがないので、代わりに identifier
を使ってね、ということをDjangoに伝えます

PersonManagerを書きます。

manager/managers.py というファイルを作ってください。

managers.py

```
from django.contrib.auth.models import BaseUserManager
```

```
from django.utils import timezone
```

```
class PersonManager(BaseUserManager):
```

```
    def create_user(self, identifier, email, password=None, **kwargs):
```

```
        if not email:
```

```
            raise ValueError('Users must have an email address')
```

```
        email = PersonManager.normalize_email(email)
```

```
        person = self.model(
```

```
            identifier=identifier,
```

```
            email=email,
```

```
            **kwargs
```

```
        )
```

```
        person.set_password(password)
```

```
person.save(using=self._db)
```

```
return person
```

続いて `views.py` でログイン用のページの処理を書いていきます。

`views.py`

```
from django.contrib.auth.views import login
```

```
from django.contrib.auth import authenticate
```

```
class CustomLoginView(TemplateView):
```

```
    template_name = "login.html"
```

```
    def get(self, _, *args, **kwargs):
```

```
        if self.request.user.is_authenticated():
```

```
            return redirect(self.get_next_redirect_url())
```

```
        else:
```

```
            kwargs = {'template_name': 'login.html'}
```

```
            return login(self.request, *args, **kwargs)
```

```
    def post(self, _, *args, **kwargs):
```

```
        username = self.request.POST['username']
```

```
        password = self.request.POST['password']
```

```
        user = authenticate(username=username, password=password)
```

```
        if user is not None:
```



```
        login(self.request, user)
        return redirect(self.get_next_redirect_url())
    else:
        kwargs = {'template_name': 'login.html'}
        return login(self.request, *args, **kwargs)

def get_next_redirect_url(self):
    redirect_url = self.request.GET.get('next')
    if not redirect_url or redirect_url == '/':
        redirect_url = '/worker_list/'
    return redirect_url
```

1) ここが一番のキモです！ userを認証しています。
この authenticate 関数が何をしているか少し説明します。

Djangoのコードを見て見ましょう。

django/contrib/auth/backends.py に authenticate 関数が定義されています。

backends.py

```
def authenticate(self, request, username=None, password=None,
    if username is None:
```

```
        username = kwargs.get(UserModel.USERNAME_FIELD) # 2
    try:
        user = UserModel._default_manager.get_by_natural_key(u
    except UserModel.DoesNotExist:
        # Run the default password hasher once to reduce the t
        # difference between an existing and a nonexistent use
        UserModel().set_password(password)
    else:
        if user.check_password(password) and self.user_can_auth
            return user
```

このコード、テクってますよね！笑

if と else の間に try-except を挟んでますが、どう処理されるかわかりづらいですよ。

簡単に言えば、 if にひかかってても else の処理は受けるっていうのがここでのテクニックなのですが、自分で簡単な関数を作ったりして確かめてください！

2) username のカラムがないときは USERNAME_FIELD に定義されてるカラムを取りに行く

3) username をkeyにしてオブジェクトを取得している

4) パスワードが正しいかcheckしている

これらを有効にするために、 `setting.py` に以下を追加します！

僕がちゃんと目を通してないだけかもしれませんが、なんかドキュメントとかstack overflowとかに書かれていなくて、結構困った記憶があります。

そういうときは、djangoの元コードをみて、どういう処理が走っているかを確認すると解決しますよ！

`settings.py`

```
# user authentication
```

```
AUTHENTICATION_BACKENDS = (  
    'django.contrib.auth.backends.ModelBackend',  
)
```

```
AUTH_USER_MODEL = 'manager.Person'
```

あとは `urls.py` にlogin用のurlを定義して、 `login.html` も追加しています。

長くなるので、ファイルは[GitHubレポジトリ](#)を確認してください！

マイグレーションファイルとデータファイルを一掃したので、作り直します。

master directory から以下を実行します。

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

では、ちゃんとログインできるか確かめてみましょう！
一人Personを使ってログインしてみます。

```
$ python manage.py shell
```

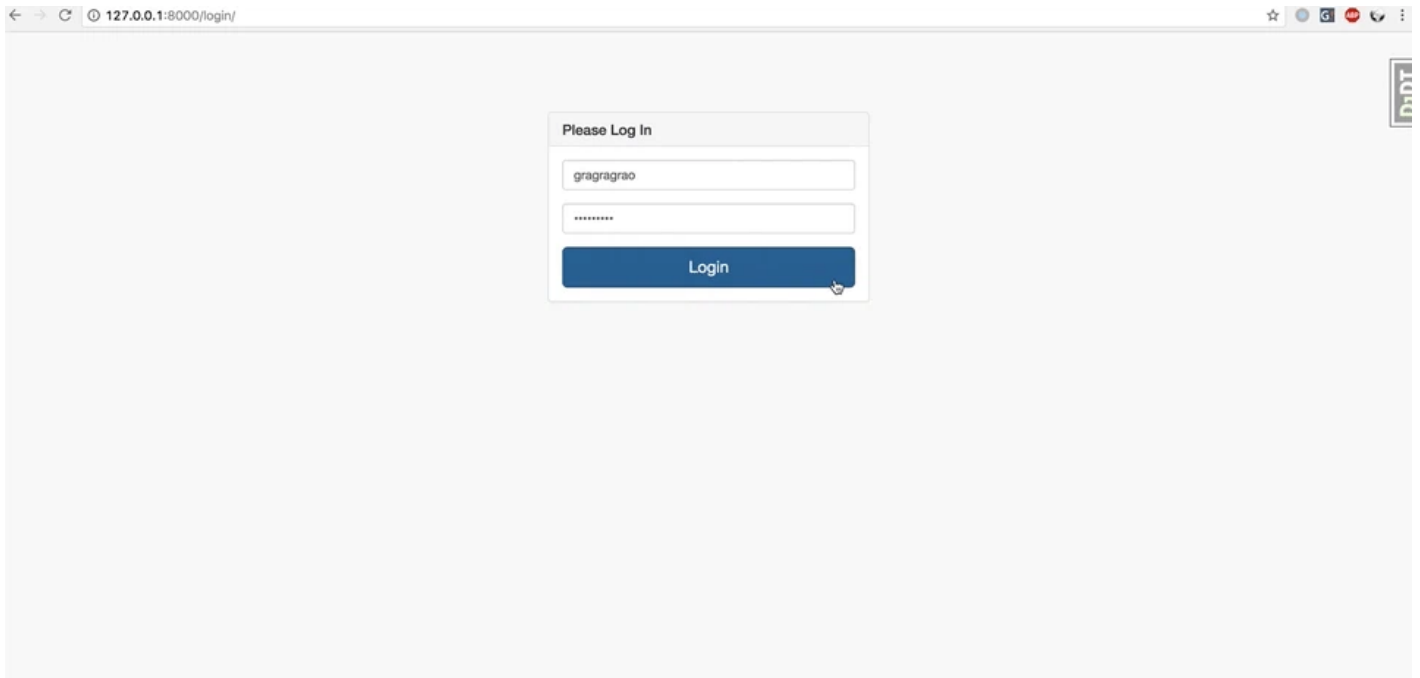
```
# from manager.models import *
# import datetime
# person = Person(identifier="gragragrao", name="gragragrao", )
# person.set_password("grao_pass")
# person.save()
```

これでログイン用のpersonができました！

user名は gragragrao 、パスワードは grao_pass ですね。

ではログインできるか確かめてみましょう！

（ WorkerListView も少し変更しているので注意してください。）



ログアウト用のエンドポイントを作成してこの章を終わりにしたいと思います。

（登録用のページはformが絡むのであとで作成しようと思います。）

ログアウト

```
<ul class="nav" id="side-menu">
  <li><a href="/worker_list/"><i class="fa fa-bar-chart" aria-l
  <li><a href="/logout/"><i class="fa fa-bar-chart" aria-hidde
</ul>
```



views.py

```
from django.contrib.auth import logout

def logout_view(request):
    logout(request)
    return redirect('/login/')
```

urls.py

```
from django.contrib.auth.decorators import login_required # ㇏

urlpatterns = [
    url(r'^logout/', manager_view.logout_view),
    url(r'^worker_list/', login_required(manager_view.WorkerLi
]
```



これで2章は終わりです。

複雑だったので、うまくいかないなどあればコメントをください。

第3章 Hijack機能の実装

この章では、Hijack機能を実装していきます。

自分が管理者なら、登録しているユーザーとしてログインしたい場面があると思います。

そんなとき、登録ユーザーのパスワードを知らなくてもログインできるのがHijack機能です。

簡単なのでさっと実装しましょう！

ドキュメントに従います。

settings.py

```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',
```

```
'django.contrib.sessions',
'django.contrib.messages',
'django.contrib.staticfiles',
'debug_toolbar',
'manager',
'hijack',
'compat',
]

# hijack
HIJACK_LOGIN_REDIRECT_URL = '/worker_list/'
HIJACK_LOGOUT_REDIRECT_URL = '/worker_list/'
HIJACK_ALLOW_GET_REQUESTS = True

HIJACK_USE_BOOTSTRAP = True
```

INSTALLED_APPS に hijack と、依存性のある compat を追加します。

[django-adminのconfigページ](#)に色々設定が書いてあります。

設定	説明	Default

設定	説明	Default
HIJACK_DISPLAY_WARNING	hijackしていることを示す黄色いバーを表示するかどうか	True
HIJACK_USE_BOOTSTRAP	Bootstrapに最適化するかどうか	False
HIJACK_URL_ALLOWED_ATTRIBUTES	userがどの属性を通してHijackされることができるか	下記参照
HIJACK_AUTHORIZE_STAFF	is_staff=True のuserが他の is_staff=False の userに対してhijackできるか	False
HIJACK_AUTHORIZE_STAFF_TO_HIJACK_STAFF	is_staff=True のuserが他の is_staff=True の userに対してhijackできるか	False
HIJACK_LOGIN_REDIRECT_URL	hijackした時にどのURLにリダイレクトされるか	settings.LOGIN_REDIRECT_URL
HIJACK_LOGOUT_REDIRECT_URL	hijackをreleaseした時にどのURLにリダイレクトされるか	settings.LOGIN_REDIRECT_URL
HIJACK_AUTHORIZATION_CHECK	hijackできる権限を決める関数	'hijack.helpers.is_authorized' (下記参照)
HIJACK_ALLOW_GET_REQUESTS	hijackがGETできるかどうか	False

HIJACK_URL_ALLOWED_ATTRIBUTES について

Default: ('user_id', 'email', 'username')

Djangoが捌けるURLは以下のようにになっています。

```
^hijack/ ^email/(?P<email>[^\@]+\@[^\@]+\.[^\@]+)/$ [name='login_w  
^hijack/ ^username/(?P<username>.*)/$ [name='login_with_userna  
^hijack/ ^(?P<user_id>[\w-]+)/$ [name='login_with_id']
```

つまり、email のフィールドが example@example.com の personに対してなら、 /hijack/email/example.com/ で Hijackできるんですね。

Django-hijackの中のコードを見ましたが、このDefault以外のカラムは使えないです。email だけでしかHijackできないようにしたい！とかなら、 ('email') に設定すればできます (カラムを減らすことはできて増やすことはできない感じですね)。

HIJACK_AUTHORIZATION_CHECK について

Default: 'hijack.helpers.is_authorized_default'

Defaultの関数はこんな感じです↓

```
def is_authorized_default(hijacker, hijacked):  
    if hijacker.is_superuser:  
        return True  
  
    if hijacked.is_superuser:  
        return False  
  
    if hijacker.is_staff and hijack_settings.HIJACK_AUTHORIZE_!:  
        if hijacked.is_staff and not hijack_settings.HIJACK_AU!  
            return False  
        return True  
  
    return False
```



Personにhijackで使われる以下の属性を追加していきます。

models.py

```
class Person(AbstractBaseUser):
    # hijack機能の実装に必要
    is_admin = models.BooleanField(default=False)
    is_staff = models.BooleanField(default=True)
    is_superuser = models.BooleanField(default=False)
```

urlに以下を追記します。

urls.py

```
url(r'^hijack/', include('hijack.urls')),
```

base.html の header に {% load hijack_tags
%} と <link rel="stylesheet" type="text/css"
href="{% static 'hijack/hijack-styles.css' %}"
> , body の直下に {% hijack_notification %} を追加
します。

このあたりの実装がわかりにくい人は、僕の[GitHubコード](#)を
参照してください。

実装はこれで以上になります。

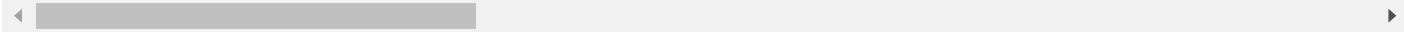
ちゃんと動くか確かめてみましょう。

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

でモデルの変更をDBに適応して、以下のよう
に `is_superuser=True` のPersonを作っていきます（これが
管理者という想定です）。

```
$ python manage.py shell

# from manager.models import *
# import datetime
# person = Person(identifier='grao_super', name='grao_super', )
# person.set_password('grao_pass')
# person.save()
```



これで完成です。

`is_superuser=False` のPersonがない場合は適宜作って
ください。

これからhijackするPersonは、 id=1,
email='example@example.com' (uniqueな値) とします。

この場合、 is_superuser=True のPerson(上で作った
grao_super)でログインしたあと、 hijackのURLを打てば
hijackできます。

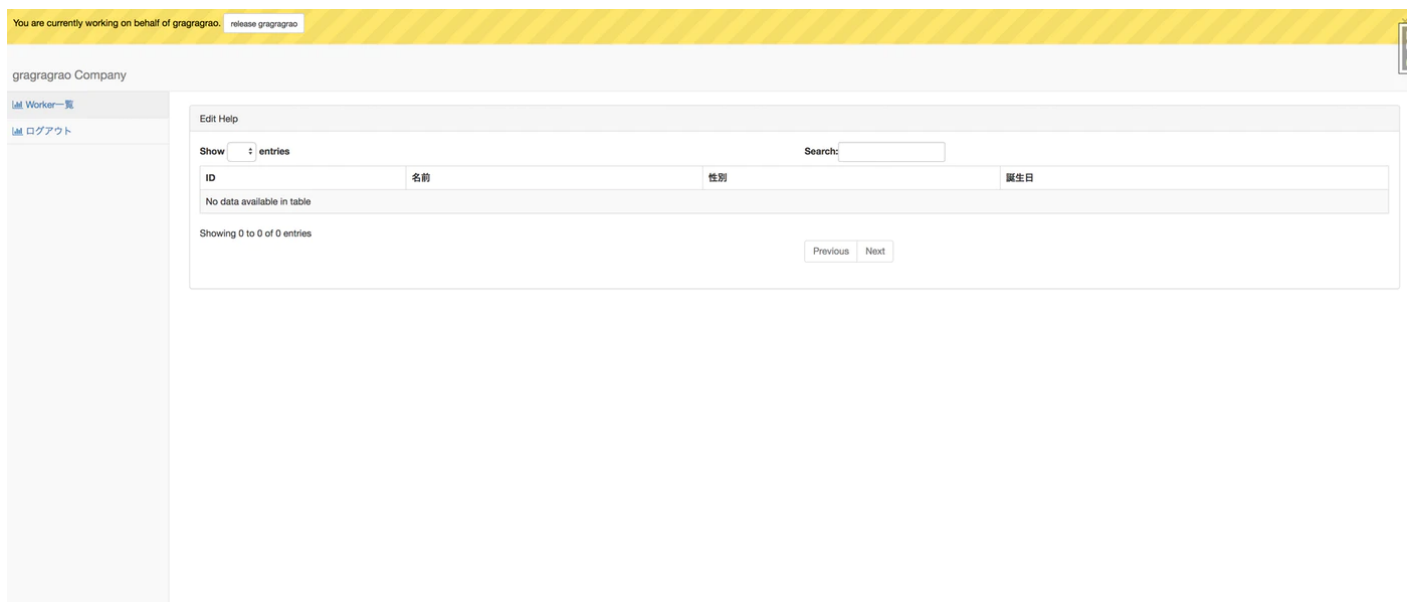
```
# python manage.py runserver 8080
```

ログインして、

```
http://localhost:8080/hijack/1/
```

```
http://localhost:8080/hijack/email/example@example.com/
```

にリクエストを送ると、以下のようにhijackできます。



release gragrgrao をクリックすると、hijackを解除できます。

以上です。