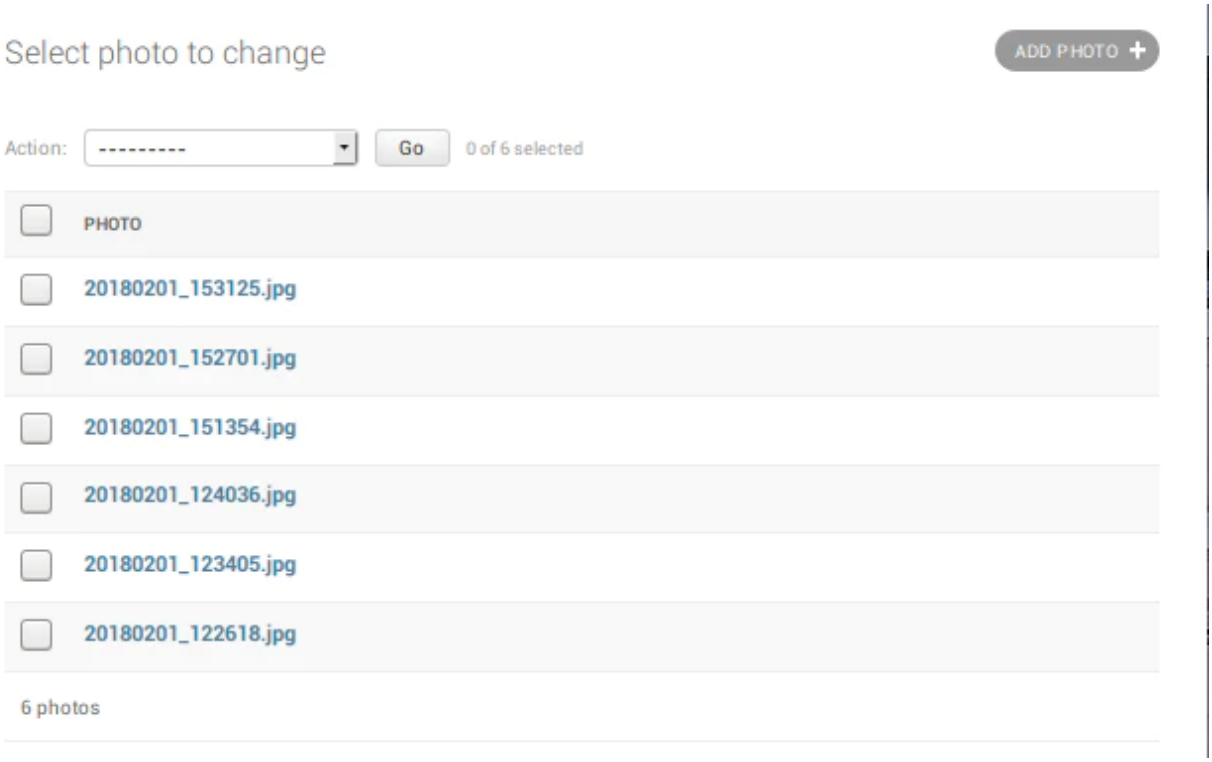
Djangoで特定のModelに対するadmin画面をカスタマイズする

Python Django template admin

Djangoはadminページがとても良くできているので、データ入力もadminだけで済ませるのではないかと考えた。特に入力を少数の、管理者権限を与えてもセキュリティー的な問題のないスタッフだけで行うような場合、わざわざデータ入力ページをadminの外に作るのは、車輪の再発明ではないだろうか（実は、元プログラムは私自身の蔵書管理のために作ったので、当然スタッフは私一人しかいない）。

ところが、admin画面はどんなModelにでも適用できるように汎用的に作られているから、逆に、かゆい所には手が届かない。例えば、次の画面である。



これは、以下で定義されたPhotoというModelのリストである。

リスト1

```
class Photo(models.Model):
    image = models.ImageField(upload_to='photos', null=True)
    comment = models.CharField(max_length=1024, blank=True, nu
    box = models.ForeignKey(Box,
        on_delete=models.SET_NULL, blank=True, null=True)
    upload_time = models.DateTimeField(auto_now=True)
    thumbnail = ImageSpecField(source='image',
        processors=[ResizeToFill(64,64)],
        format="JPEG",
        options={'quality': 50}
```

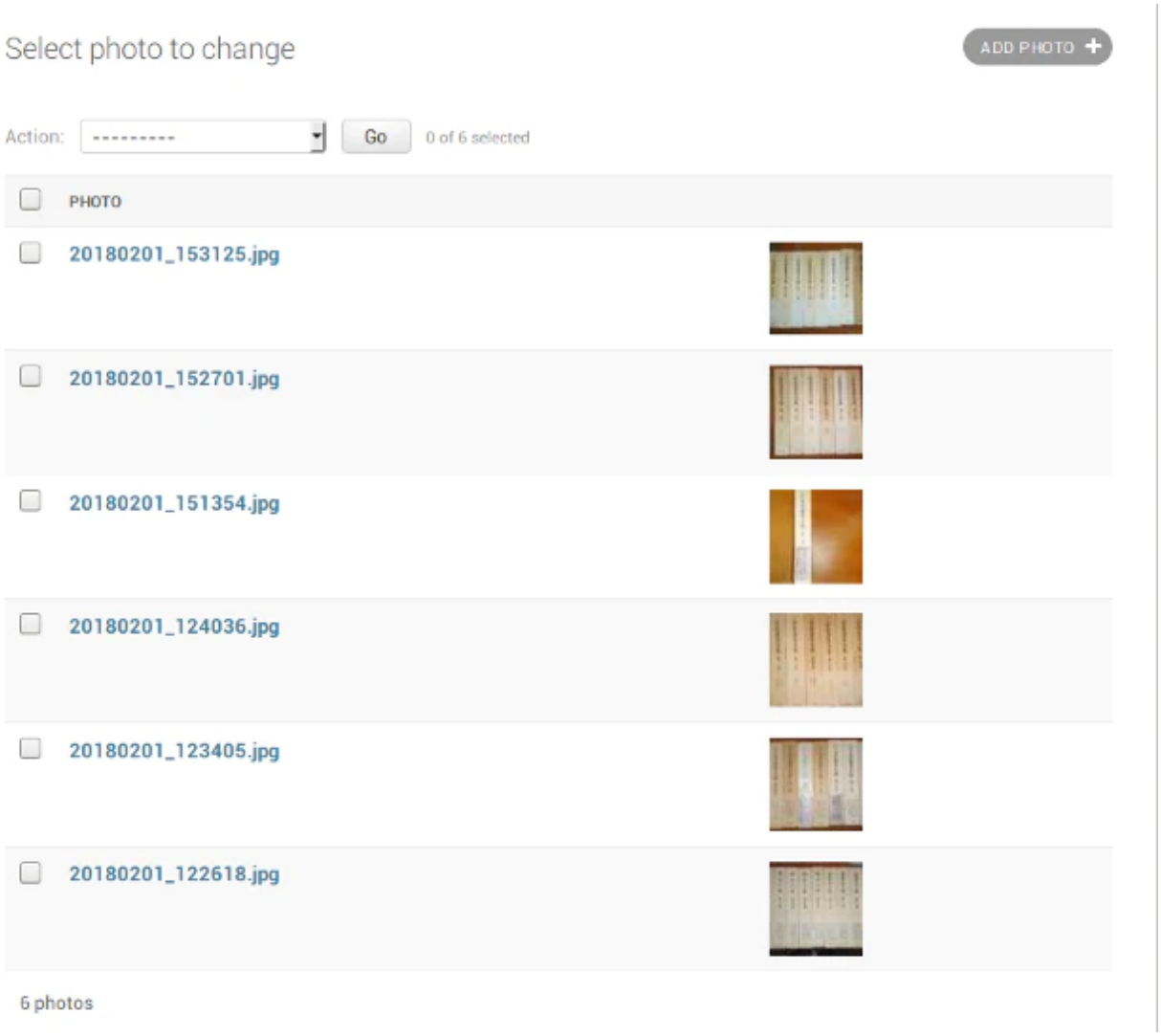
```
        )
    ...
```

Photoは、Imagekitライブラリを使ってthumbnailという縮小画像を作るので、リスト中に次の図のようにthumbnail画像を表示したい。さらに、縮小画像をクリックすると元画像が別ウインドウに表示されるようにしたい。

やりたいことは、たったこれだけである。ここでは、2種類の方法を紹介する。

- Templateをカスタマイズする方法
- ModelAdminをカスタマイズする方法

当初Templateをカスタマイズするしかないと思っていたのだが、ModelAdminをカスタマイズする方法でもできることが分かった。明らかに後者の方がスマートで簡潔だが、前者にはカスタムタグの用法を把握できるメリットがあるので、時間経過に従って前者を先に並べた。なお、ここで使っているDjangoのバージョンは2.0.2、Pythonは3.5.3である。



Templateをカスタマイズする方法

この例では、プロジェクトの中にアプリケーション「bib」を作成し、bibアプリケーションに「Photo」、「Box」、「Book」の3種類のモデルがある。このために、adminには3種類のモデルに対応した3種類のリストが存在する。tumbnail

が必要なのはPhotoだけなので、Photoのリストだけをカスタマイズしなければならない。

ちなみに、Bookは個々の蔵書の情報、Boxは蔵書を収納しているダンボール箱の情報、Photoは1個の箱に収められた書籍を撮影した写真（1個の箱に対して複数の写真がある）である。1個の箱に20冊から40冊程度入っていて、その箱が30箱ほどある。写真を見ながら箱に収められた書籍情報（Book）を少しづつ作って行くのが目的である。

特定のModelに対するTemplate

adminのTemplateのカスタマイズの方法は、Djangoサイトの「Overriding admin templates」という項目に説明がある。

それによれば、adminのビルトインTemplateは、

```
{DJANGO_DIR}/contrib/admin/templates/admin
```

の中に格納されている。ここで、{DJANGO_DIR}は、Djangoのインストールディレクトリを表わしている。これをオーバー

ライドするためには、

```
{PROJECT_DIR}/{APP}/templates/admin
```

に同じ名前のTemplateファイルを置けば良い。ここで、{PROJECT_DIR}は当該プロジェクトのディレクトリ、{APP}はアプリケーション名を表わす。ところが、これでは全てのModelに対するTemplateを上書きしてしまう。特定のModelだけを上書きしたい場合には、

```
{PROJECT_DIR}/{APP}/templates/admin/{APP}/{MODEL}
```

にTemplateファイルを置かなければならない。なお、{MODEL}はモデル名をあらわす。

リストのTemplate

adminにおける各Modelに対するリストは、`change_list.html` という名前である。従って、今回は

```
{DJANGO_DIR}/contrib/admin/templates/admin/change_list.html
```

を

```
{PROJECT_DIR}/bib/templates/admin/bib/photo/change_list.html
```

にコピーして編集することにする。

しかし、これだけでは済まなかった。それは、以下に一部を示すchange_list.htmlの中に、 {% result_list cl %} というカスタムタグが含まれているからである。

リスト2

```
{% block result_list %}
  {% if action_form and actions_on_top and cl.show_admin_actio
    {% admin_actions %}
  {% endif %}
  {% result_list cl %}
  {% if action_form and actions_on_bottom and cl.show_admin_ac
    {% admin_actions %}
  {% endif %}
{% endblock %}
```

カスタムタグ

リスト2の中で、 `{% result_list cl %}` という記述がある。これは、カスタムタグと呼ばれる。

Djangoのテンプレート記述言語では、 `{% tag %}` のようなものをテンプレートタグと呼ぶ。タグには、 `{% for %}` 、 `{% if %}` 、 `{% else %}` 、 `{% endif %}` などのように、あらかじめ定義された20種類ほどの組み込みタグがあるが、それ以外に、カスタムタグを定義することができる。 `{% result_list %}` は、adminのカスタムタグなので、

```
{DJANGO_DIR}/contrib/admin/templatetags/admin_list.py
```

の中でリスト3のように定義されている。なお、タグの中の2番目の引数 `cl` は、カスタムタグを定義する関数（タグと同じ名前の関数）に渡される引数である。リスト3は、 `admin_list.py` の中の定義関数の部分を示したものである。

リスト3

```
@register.inclusion_tag("admin/change_list_results.html")
def result_list(cl):
    """
    Display the headers and data list together.
    """
    headers = list(result_headers(cl))
    num_sorted_fields = 0
    for h in headers:
        if h['sortable'] and h['sorted']:
            num_sorted_fields += 1
    return {'cl': cl,
            'result_hidden_fields': list(result_hidden_fields(
            'result_headers': headers,
            'num_sorted_fields': num_sorted_fields,
            'results': list(results(cl))}
```

この関数定義部分の意味は、Templateの中で{% result_list cl %}というタグに出会うと、その位置に、

```
admin/change_list_results.html
```

というサブTemplateを挿入し、上記関数の戻り値であるディクショナリを、サブTemplateに渡すということである。

カスタムタグの定義

カスタムタグの定義の方法は、Djangoサイトの「独自のテンプレートタグを記述する」という項目に説明がある。

アプリケーションの中で、カスタムタグを定義する場合には、

```
{PROJECT_DIR}/{APP}/templatetags
```

ディレクトリの中に定義ファイル（Pythonファイル）を置く。その際、ユーザ定義ファイル以外に、 `__init__.py` という空のファイルを置いておく必要がある。

何をしなければならないか

当初の目的のために、新たに作成したファイルを以下にあげる。実質的には２個のファイルだけである。

リスト4

```
{PROJECT_DIR}
+-- bib
    +-- templatetags
    |   +-- __init__.py
    |   +-- admin_list_bib_photo.py
    +-- templates
        +-- admin
            +-- bib
                +-- photo
                    +-- change_list.html
```

カスタムタグの定義ファイル

カスタムタグを定義するファイル

admin_list_bib_photo.py を以下に示す。

リスト5（admin_list_bib_photo.py）

```
rom django.contrib.admin.templatetags import admin_list
from django.template import Library
from django.utils.html import format_html

register = Library()

@register.inclusion_tag("admin/change_list_results.html")
def result_list_photo(cl):
```

```
    """
    Recall result_list() in admin_list.py
    """
    rl = admin_list.result_list(cl)
    rl['result_headers'].append({
        "text": '',
        "sortable": True,
        "sorted": True,
        "ascending": True,
        "sort_priority": None,
        "url_primary": None,
        "url_remove": None,
        "url_toggle": None,
        "class_attrib": None,
    })
    results = rl['results']
    qs = cl.result_list
    tmpl = '<td><img src="{}" onClick="window.open(\'{}\', \'{
    for i in range(len(results)):
        basename = os.path.basename(qs[i].image.url)
        last_tag = format_html(
            tmpl, qs[i].thumbnail.url, qs[i].image.url, basena
        results[i].append(last_tag)
    return rl
```

新しいカスタムタグは result_list_photo という名前にした。やっていることは、最初に元のカスタムタグ {% result_list %} の定義関数を呼び出して、戻り値のディクショナリを取得し、そのディクショナリのうちの２個のキーに必要なオブジェクトを追加しているだけである。

result_list_photo 関数に与えるclという引数は、

```
{DJANGO_DIR}/contrib/admin/views/main.py
```

で定義されている ChangList という名前のクラスのインスタンスである。

関数の戻り値であるディクショナリのうち、 results というキーに、ChangeListのresult_list変数から取得した値を使って、tumbnailのimgタグを追加している。

result_headers に追加している情報は、リストのヘッダ部分の表示に関するものである。

以上の記述の詳細は、元のタグ {% result_list %} を定義しているファイル

```
{DJANGO_DIR}/contrib/admin/templatetags/admin_list.py
```

を読んで頂ければ、お分かりになるだろう。

Templateファイル（change_list.html）

当初、上で紹介した `change_list.html` をまるごとコピーして2行分を変更した方法を紹介していたが、たった2行の変更箇所のために89行もある元ファイルをコピーするのは効率が悪い。こういうときのために、Djangoのテンプレート言語には、 `{% block %}` タグが用意されている。

blockタグは元ファイルの様々な場所に入れておくことができ、その部分だけを上書きすることができる。これを使った新しい `change_list.html` が以下である。

リスト7（新しい'change_list.html'の全文）

```
{% extends "admin/change_list.html" %}
{% load admin_list admin_list_bib_photo %}

{% block result_list %}
    {% if action_form and actions_on_top and cl.show_admin_act
```

```
    {% result_list_photo cl %}
    {% if action_form and actions_on_bottom and cl.show_admin_
{% endblock %}
```

1行目の {% extends "admin/change_list.html" %} は、上書きされる元ファイルを表す。ファイル名が同じなのでパスが重ならないように注意しなければならない（この場合は幸い admin/bib/photo/change_list.html なのでOK）。

元ファイルには result_list という名前のブロックが5行分定義されている。そのブロックをそのままコピーして、 {% result_list cl %} タグだけを {% result_list_photo cl %} タグに置き換えた（ブロック名とタグ名の両方に'result_list'が使われているが、両者は別物）。

ModelAdminをカスタマイズする方法

上記のTemplateをカスタマイズする方法は、Template言語を練習するためには良いが、もっと簡単な方法がないかと探していたら、ModelAdminをカスタマイズする方法で可能なことが分かった。分かってみれば、とても簡潔だった。

リスト8（bib/models.pyの一部）

```
class PhotoAdmin(admin.ModelAdmin):
    # リスト画面
    list_display = ('photo', 'thumbnail')

    # 以下の2行は詳細画面のため
    readonly_fields = ('thumbnail', )
    fields = ('image', 'thumbnail', 'comment', 'box')

    # list_displayに対応して、これが必要になる
    def photo(self, instance):
        return instance

    def thumbnail(self, instance):
        tmpl = '<img src="{}" onClick="window.open(\'{}\', \'{
        basename = os.path.basename(instance.image.url)
        return format_html(
            tmpl, instance.thumbnail.url, instance.image.url,
```

{APP}/models.py に上のPhotoAdminを加えれば良い。これに伴い、 {APP}/admin.py を以下のように変更する。

```
admin.site.register(Photo, PhotoAdmin)
```

第2引数として、PhotoAdminを付け加えるのである（importも必要）。

PhotoAdminには、リスト画面だけでなく、詳細画面のカスタマイズも付け加えている。

ModelAdmin.list_display

ModelAdminのlist_display属性は、リスト画面に表示する項目をタブルで記述する。第2要素（'thumbnail'）が、PhotoAdminクラスの `thumbnail` 関数と関係付けられる（Photoクラスの `thumbnail` 属性ではない）。

`thumbnail` 関数の第2引数は、Photoクラスのインスタンスなので、 `Photo.thumbnail` と `Photo.image` を使って `img` タグを作成し、戻り値とする。

javascriptの `window.open` 関数の第2引数は、windowの名前である。この名前を付けることにより、同じ画像が重複して表示されることがない。

list_displayの第1要素は、Photoのインスタンスを戻す関数名（'photo'）である。 `PhotoAdmin.photo` を別途、定義することが必要である。

ModelAdmin.fields

ModelAdminのfields属性は、詳細画面のform（編集、削除）に表示される項目をカスタマイズする。
ModelAdmin.list_displayに似ているが、そのままではカスタマイズした関数を追加できず、元クラスの属性を登録するだけである。従って通常は、属性を追加するのではなく、編集させたくない属性をformから除去するために使用する。

ただ、その規則には例外があって、readonly_fieldsに登録した関数だけは追加できる。これにより、list_displayで追加した関数（'thumbnail'）をそのまま利用できた。

おわりに

縮小画像をadminのリスト中に表示したいという極めて些細な目的だったが、調べなければならない情報がたくさんあったので、少しわずらわしい作業だった。結果的に、後から追加した「ModelAdminをカスタマイズする方法」が、当初の目的のためには最適な方法だった。いずれにしても編集箇所が極めて少ないので、最初から分かっていれば大した作業ではなかったのだが、読んで頂いた方の助けになったら幸いである。

変更履歴

2019-08-28: Djangoのドキュメントへのリンクをv2.0からv2.2のものに変更した。元プログラムを作成した経緯を付け加えた

2018-02-10: `change_list.html` をブロックレベルで上書きする方法を追加した

2018-02-14: 「ModelAdminをカスタマイズする方法」を追加した

2018-07-26: 「ハッシュ」という表現を、Pythonの習慣に従って「ディクショナリ」に改めた