

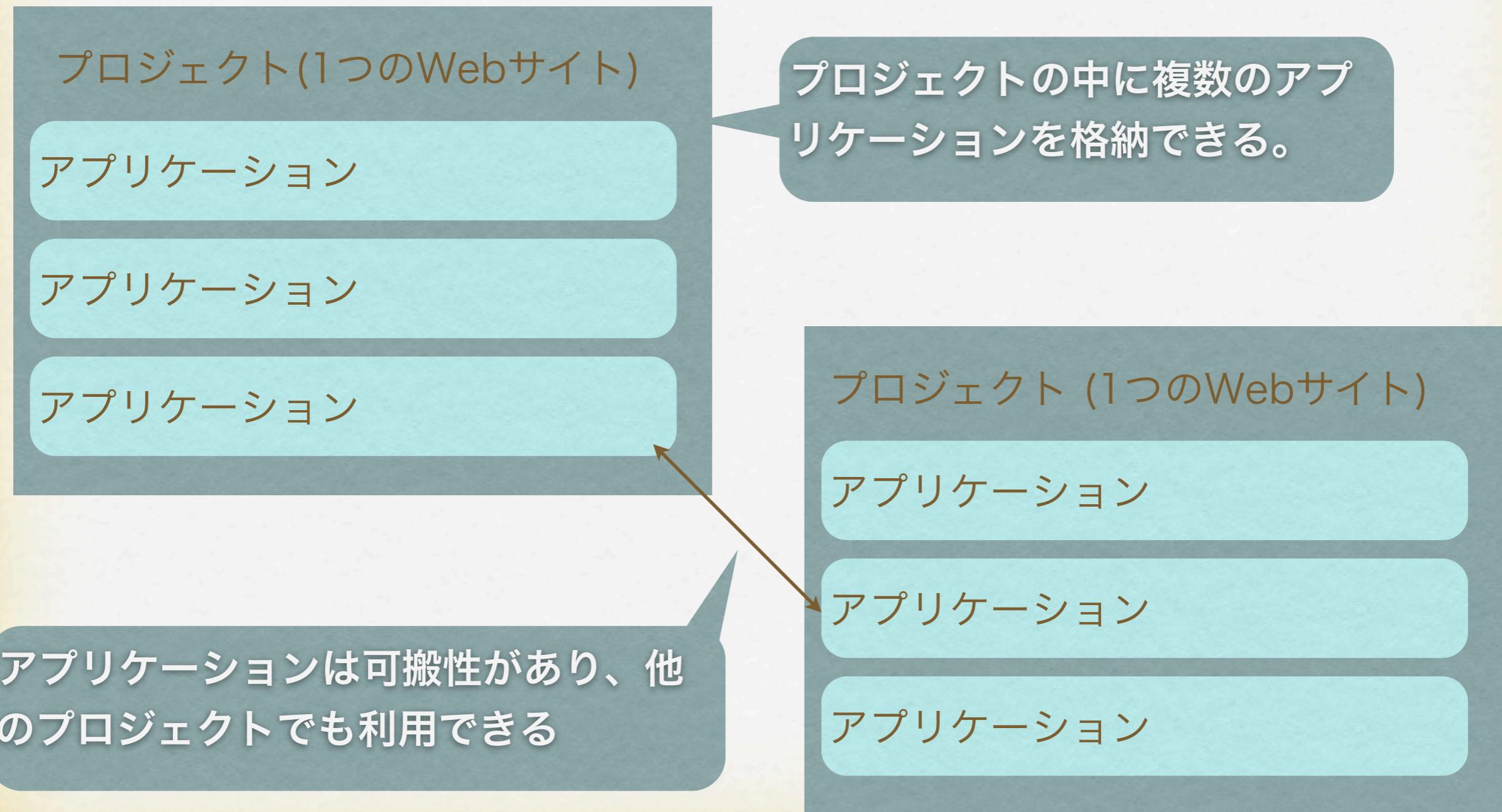
Django

入門

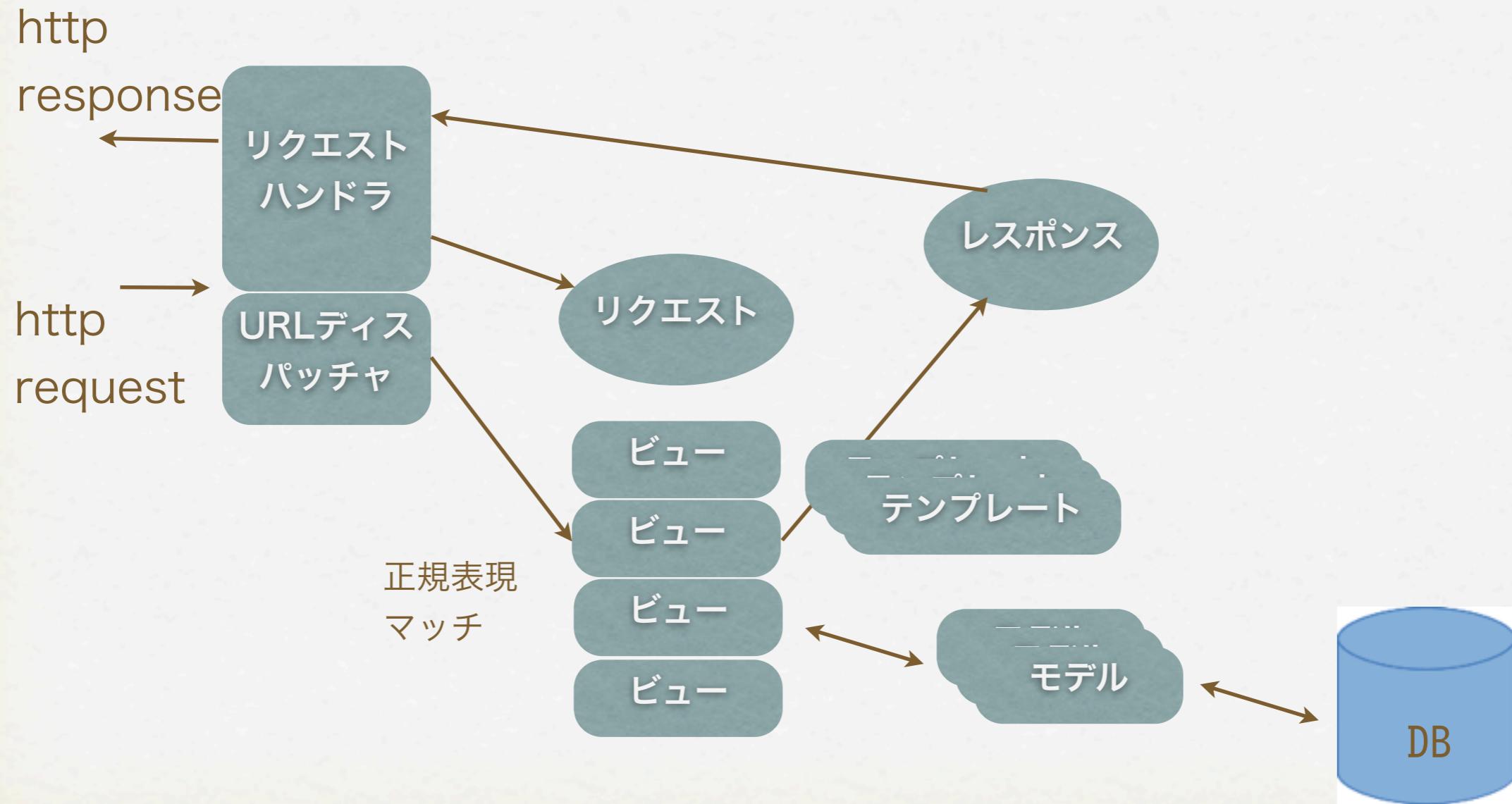
Django

- ※ Pythonを用いたWebフレームワーク
- ※ Webフレームワークからデータアクセス、テスト支援までを含んだ「オールインワン」フレームワーク
- ※ 今回の資料はDjango 1.5.1をベースにしています。

Djangoのアプリケーション構造



基本的な処理の流れ



インストール

- * Pythonをインストールしておく。
Django 1.5.1でサポートされるPythonは2.6.5 - 2.7
- * Python pipを用いたインストール例(Debian系)
`sudo apt-get install python-pip`
`sudo pip install Django==1.5.1`

インストールの確認

* Pythonで確認する

```
shanai@shanai-desktop:~/oss/django$ python
Python 2.7.3 (default, Sep 26 2012, 21:51:14)
[GCC 4.7.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import django
>>> print(django.get_version())
1.5.1
>>>
```

備忘録

サンプルアプリケーション

プロジェクトの作成

- * 以下のコマンドでプロジェクトを作成

```
django-admin.py startproject myproject
```

▲
プロジェクト名

(djangoやtestなどのPythonパッケージ名とぶつかる名前を避ける)

作成されるファイル

```
Shisei-no-MacBook-Pro:django shanai$ find myproject
myproject
myproject/manage.py
myproject/myproject
myproject/myproject/__init__.py
myproject/myproject/settings.py
myproject/myproject/urls.py
myproject/myproject/wsgi.py
```

manage.py	Pythonで書かれた管理ツール https://docs.djangoproject.com/en/1.5/ref/django-admin/
myproject/__init__.py	空のファイル。Pythonがパッケージとして認識するために必要。
myproject/settings.py	構成ファイル
myproject/urls.py	urlディスパッチャのためのファイル
myproject/wsgi.py	WSGI互換サーバ用

動作の確認

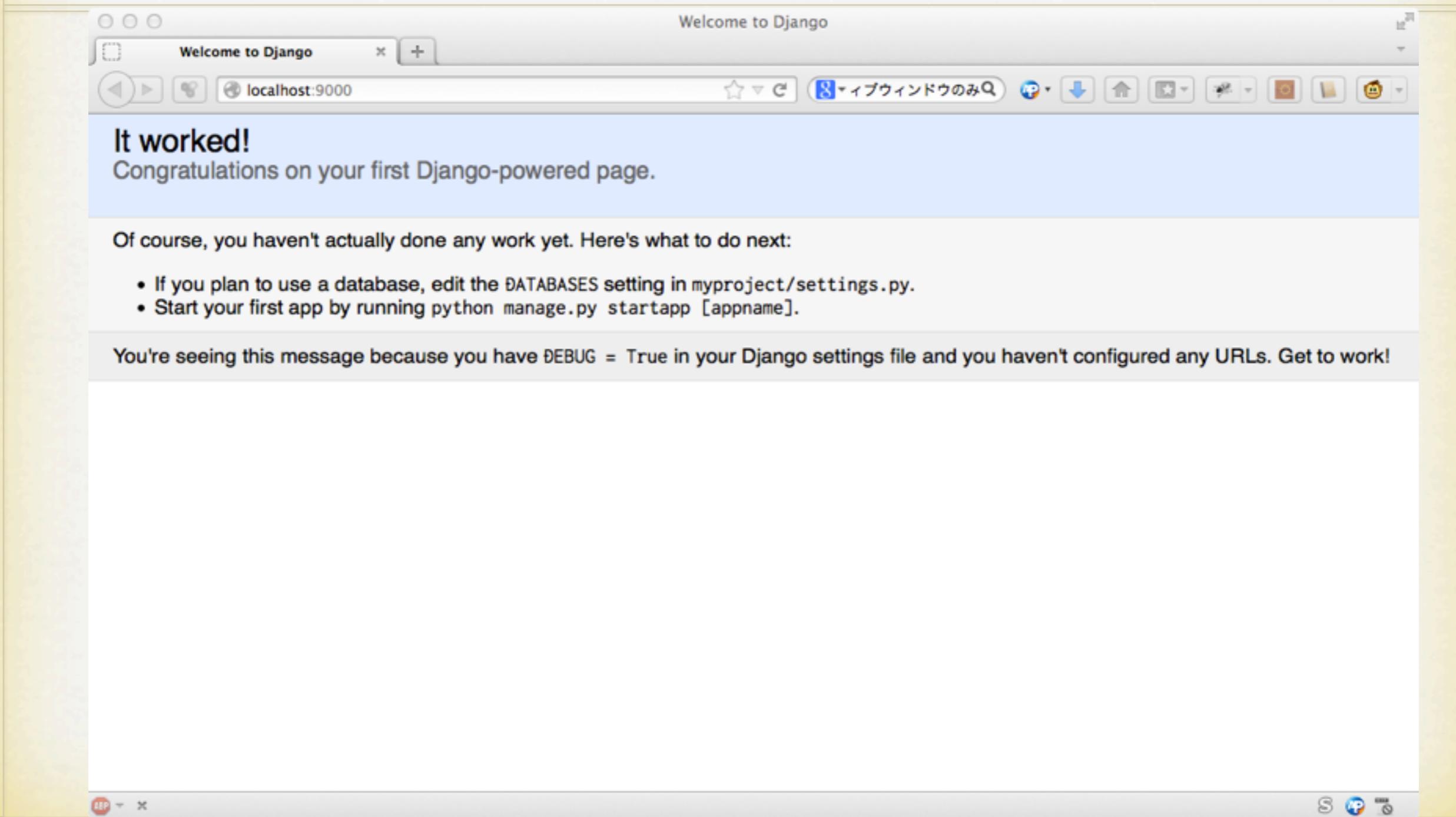
以下をmyproject直下で実行

```
./manage.py runserver 0.0.0.0:9000
```

0.0.0.0はバインドアドレス。9000はポートの指定(省略すると8000)
Pythonで書かれた軽量な開発用のサーバが使用される。

<https://docs.djangoproject.com/en/1.5/ref/django-admin/#django-admin-runserver>

動作の確認



データベースの設定

- * settings.pyのDATABASESに設定する

```
DATABASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.',  
        'NAME': '',  
        # The following settings are not used with sqlite3:  
        'USER': '',  
        'PASSWORD': '',  
        'HOST': '',  
        'PORT': ''  
    }  
}
```

SQLiteを使う場合は、ENGINEとNAMEだけ設定すれば良い。

```
'ENGINE': 'django.db.backends.sqlite3',  
'NAME': 'db',
```

データを保存する場所。この場合はdbディレクトリ

TimeZone/LANGUAGE_CODEの設

＊ settings.pyに設定

```
TIME_ZONE = 'Asia/Tokyo'
```

```
LANGUAGE_CODE = 'ja-JP'
```

<http://www.postgresql.org/docs/8.1/static/datetime-keywords.html#DATETIME-TIMEZONE-SET-TABLE>

標準アプリケーション

- * 標準でアプリケーションが幾つか登録されている
(settings.pyのINSTALLED_APPS)。

```
INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    # Uncomment the next line to enable the admin:
    # 'django.contrib.admin',
    # Uncomment the next line to enable admin documentation:
    # 'django.contrib.admindocs',
)
```

これらを使用するに先立ってDBを初期化する必要がある。

DBの初期化

```
shanai@shanai-desktop:~/django/mysite$ ./manage.py syncdb
Creating tables ...
Creating table auth_permission
Creating table auth_group_permissions
...
You just installed Django's auth system, which means you don't have any superusers defined.
Would you like to create one now? (yes/no): yes
Username (leave blank to use 'shanai'): admin
Email address: ruimo.uno@gmail.com
Password:
Password (again):
Superuser created successfully.
Installing custom SQL ...
Installing indexes ...
Installed 0 object(s) from 0 fixture(s)
```

認証システム用にパスワードを入力。

アプリケーションの作成

ここまでプロジェクトが設定できたので、アプリケーションを作成していきます。

アプリケーションの作成

```
./manage.py startapp memo
```

アプリケーション名

```
myproject shanai$ find memo  
memo  
memo/__init__.py  
memo/models.py  
memo/tests.py  
memo/views.py
```

アプリケーションはPythonパッケージなので、他のプロジェクトでも使用できる。

myproject

 myproject

 memo

プロジェクト共通のサイト全体の
設定など

今回作成

アプリケーションの登録

- * myproject/settings.pyのINSTALLED_APPSに登録する

```
INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'memo',
    # Uncomment the next line to enable the admin:
    # 'django.contrib.admin',
    # Uncomment the next line to enable admin documentation:
    # 'django.contrib.admindocs',
)
```

モデルの作成

- * memo/models.pyを作成する。

```
# coding=utf-8
```

```
from django.db import models
```

```
class Memo(models.Model):
```

```
    memo = models.CharField(max_length=200)
```

```
    create_date = models.DateTimeField('作成日')
```

```
class Category(models.Model):
```

```
    memo = models.ForeignKey(Memo)
```

```
    category_name = models.CharField(max_length=80)
```

フィールドは、
XXXFieldで作成

max指定はバリデー
ションでも使用される

名前を付けると
ビューで使用できる

外部キー指定

Memo
memo
create_date

1 ————— n

Category
category_name

SQLの確認

* SQLの確認をする(実行はされない)

```
Shisei-no-MacBook-Pro:myproject shanai$ ./manage.py sql memo  
BEGIN;  
CREATE TABLE "memo_memo" (  
    "id" integer NOT NULL PRIMARY KEY,  
    "memo" varchar(200) NOT NULL,  
    "create_date" datetime NOT NULL  
)  
;  
CREATE TABLE "memo_category" (  
    "id" integer NOT NULL PRIMARY KEY,  
    "memo_id" integer NOT NULL REFERENCES "memo_memo" ("id"),  
    "category_name" varchar(80) NOT NULL  
)  
;  
COMMIT;
```

使用しているDBに合った
SQLが出力される。

サロゲートキーが付与される。

外部キーのフィールドには_idが付与される。

SQLの実行

- * syncdbを実行すると、登録アプリケーションのDBが同期される。

```
Shisei-no-MacBook-Pro:myproject shanai$ ./manage.py syncdb
Creating tables ...
Creating table memo_memo
Creating table memo_category
Installing custom SQL ...
Installing indexes ...
Installed 0 object(s) from 0 fixture(s)
```

モデルの動作確認

- * manage.py shellを用いると、mysite/settings.pyをインポートした状態で対話シェル(REPL)を開始する。

手で、

```
DJANGO_SETTINGS_MODULE=mysite.settings manage.py
```

と実行するのと同じ。

シェルによる確認

```
Shisei-no-MacBook-Pro:myproject shanai$ ./manage.py shell
Python 2.7.2 (default, Oct 11 2012, 20:14:37)
[GCC 4.2.1 Compatible Apple Clang 4.0 (tags/Apple/clang-418.0.60)] on
darwin
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> from memo.models import Memo, Category
>>> from django.utils import timezone
>>> m = Memo(memo="肉を買う", create_date=timezone.now())
>>> m.save()
>>> m.id
1
>>> m.memo
'肉を買う'
>>> Memo.objects.all()
[<Memo: Memo object>]
>>>
```

Memoに__unicode__()メソッドが無いため。内容
が分かりにくい。

文字列への変換

```
class Memo(models.Model):
    memo = models.CharField(max_length=200)
    create_date = models.DateTimeField('作成日')
    def __unicode__(self):
        return "Memo [" + self.memo + "]"]
```

一度シェルを抜ける必要がある。

```
>>> from memo.models import Memo, Category
>>> Memo.objects.all()
[<Memo: Memo [肉を買う]>]
```

その他の操作

```
>>> Memo.objects.filter(id=1)
[<Memo: Memo [肉を買う]>]
>>> Memo.objects.filter(memo__startswith='肉')
[<Memo: Memo [肉を買う]>]
>>> from django.utils import timezone
>>> current_year = timezone.now().year
>>> Memo.objects.get(create_date__year=current_year)
<Memo: Memo [肉を買う]>
>>> Memo.objects.get(id=2)
Traceback (most recent call last):
  File "<console>", line 1, in <module>
    File "/Library/Python/2.7/site-packages/django/db/models/base.py",
line 143, in get
    return self._get_by_natural_key(*args, **kwargs)
  File "/Library/Python/2.7/site-packages/django/db/models/query.py",
line 389, in get
    (self.model._meta.object_name, kwargs))
DoesNotExist: Memo matching query does not exist. Lookup parameters were
{'id': 2}
```

条件指定

無いレコード
の検索

その他の操作

```
>>> from memo.models import Memo, Category  
>>> memo = Memo.objects.get(pk=1)  
>>> memo.category_set.all()  
[]  
>>> cat = memo.category_set.create(category_name='買い物')  
>>> cat.memo  
<Memo: Memo [肉を買う]>  
>>> memo.category_set.all()  
[<Category: Category [買い物]>]  
>>> Category.objects.filter(memo_create_date__year=2013)  
[<Category: Category [買い物]>]
```

pk指定。1件のみ

関連をたどる

関連テーブルのレコード追加

モデルへのメソッド追加

```
import datetime
from django.db import models
from django.utils import timezone

class Memo(models.Model):
    memo = models.CharField(max_length=200)
    create_date = models.DateTimeField('作成日')
    def __unicode__(self):
        return "Memo [" + self.memo + "]"
    def was_created_recently(self):
        return self.create_date >= timezone.now() - datetime.timedelta(days=7)
```

最近1週間以内に作成されたメモかどうか。

```
>>> from memo.models import Memo
>>> memo = Memo.objects.get(pk=1)
>>> memo.was_created_recently()
True
```

サイト管理機能

まだモデルを作成しただけですが、Djangoのサイト管理機能を使うと、モデルの編集アプリケーションを簡単に構築できます。

サイト管理機能

- * 標準でモデルを保守するための機能が入っている。コメントアウトされているのでsettings.pyを変更する。

```
INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    # Uncomment the next line to enable the admin:
    'django.contrib.admin',
    # Uncomment the next line to enable admin documentation:
    # 'django.contrib.admindocs',
    'polls',
)
```

初期設定

- * syncdbを実行する。

```
Shisei-no-MacBook-Pro:myproject shanai$ ./manage.py syncdb
Creating tables ...
Creating table django_admin_log
Installing custom SQL ...
Installing indexes ...
Installed 0 object(s) from 0 fixture(s)
```

- * mysite/urls.pyを編集する。

```
from django.contrib import admin
admin.autodiscover()

urlpatterns = patterns('',
    ...
    url(r'^admin/', include(admin.site.urls)),
)
```

管理対象の追加

- * 管理サイトにアプリケーションを追加するには、対象アプリケーションの中にadmin.pyを作成して(今回はmemo/admin.py)、サーバを再起動する。

```
from django.contrib import admin  
from memo.models import Memo  
  
admin.site.register(Memo)
```

モデルMemoを管理対象に追加する。

管理サイトの利用

- * /adminを開いて、導入時に指定したユーザ/パスワードを指定。



管理サイトの利用

The screenshot shows two browser windows demonstrating the Django admin interface.

Left Window (List View):

- Title: 変更する memo を選択 | Django サイト管理
- Address bar: localhost:9000/admin/Memo/Memos/
- Content:
 - Django 管理サイト
 - ようこそ shanai. パスワードの変更 / ログアウト
 - ホーム > Memo > Memos
 - 変更する memo を選択
 - 操作: -----
 - 実行
 - 1個中0個選択
 - Memo
 - Memo [肉を買う]
 - 1 memo

Right Window (Detail View):

- Title: memo を変更 | Django サイト管理
- Address bar: localhost:9000/admin/Memo/Memos/1/
- Content:
 - Django 管理サイト
 - ようこそ shanai. パスワードの変更 / ログアウト
 - ホーム > Memo > Memos > Memo [肉を買う]
 - memo を変更
 - Memo: 肉を買う
 - 作成日: 日付: 2013-05-22 今日 | 月
 - 時刻: 14:31:02 現在 | ○
 - *削除**
 - 保存してもう一つ追加
 - 保存して編集を続ける
 - 保存**

Annotations:

- A green speech bubble points to the list view window with the text "一覧表示" (List View).
- A green speech bubble points to the detail view window with the text "詳細表示" (Detailed View).
- A large green speech bubble at the bottom left points to the "Memo" field in the list view with the text "モデルで名前を付けておいたので日本語で表示されている" (The name was given in the model, so it is displayed in Japanese).

管理サイトの利用

The screenshot shows two windows of the Django Admin interface:

- Left Window (Top):** A modal titled "変更する memo を選択 | Django サイト管理". It contains a dropdown menu set to "-----", a "実行" (Execute) button, and a list of three items: "Memo", "Memo [野菜を買う]", and "Memo [肉を買う]". Below the list, it says "2 memos".
 - A green checkmark message at the bottom states: "memo "Memo [野菜を買う]" を追加しました。"
- Right Window (Bottom):** The main Django Admin dashboard titled "サイト管理 | Django サイト管理". It lists four models: "Auth", "Users", "グループ", "Memo", "Memos", "Sites", and "サイト". Each model has "追加" (Add) and "変更" (Change) buttons.

Two callout bubbles are present:

- A blue bubble on the right side points to the "追加" (Add) button in the "Memo" section of the left window, with the text "一件追加" (Add one).
- A blue bubble on the right side points to the "History" section in the right window, with the text "ヒストリー" (History).

管理サイトのカストマイズ

* カラム表示順変更

```
class MemoAdmin(admin.ModelAdmin):
    fields = ['create_date', 'memo']

admin.site.register(Memo, MemoAdmin)
```

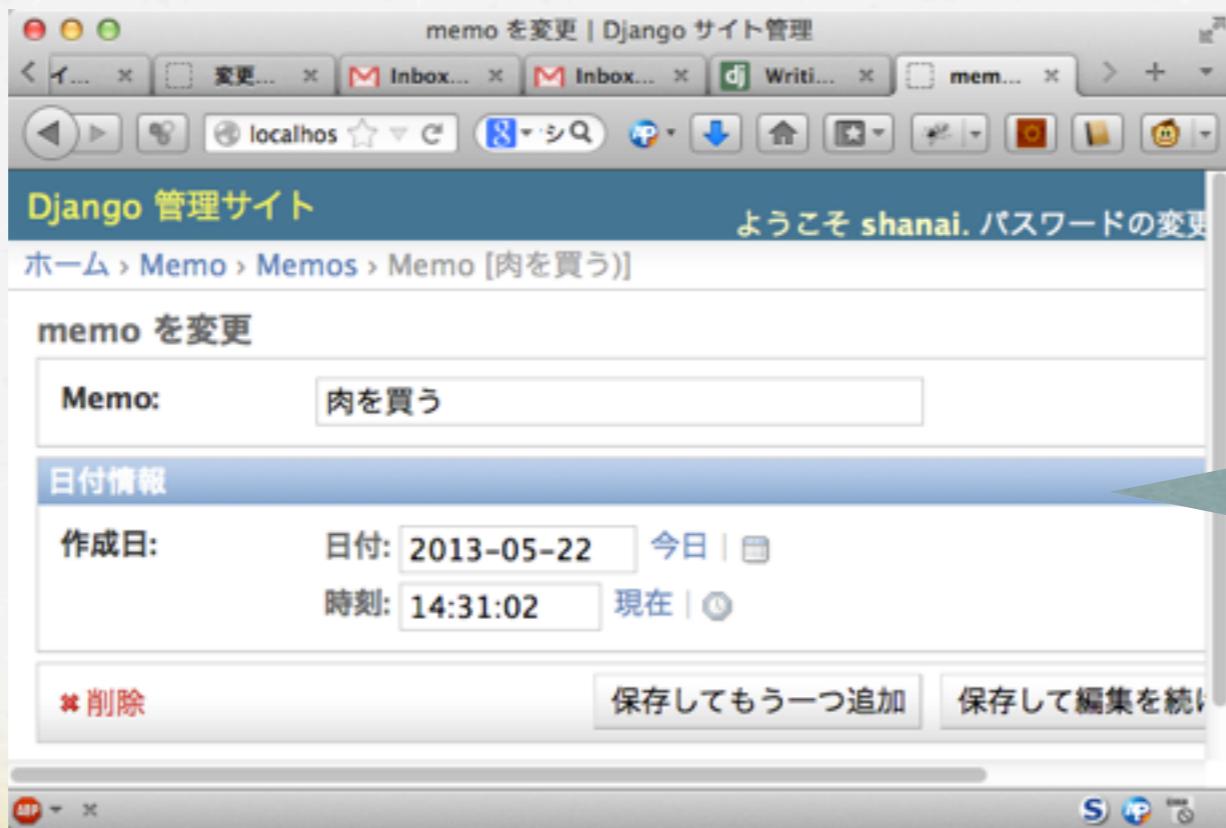


作成日が先に来る。

管理サイトのカストマイズ

* グルーピング

```
class MemoAdmin(admin.ModelAdmin):
    fieldsets = [
        (None, {'fields': ['memo']}),
        ('日付情報', {'fields': ['create_date']}),
```

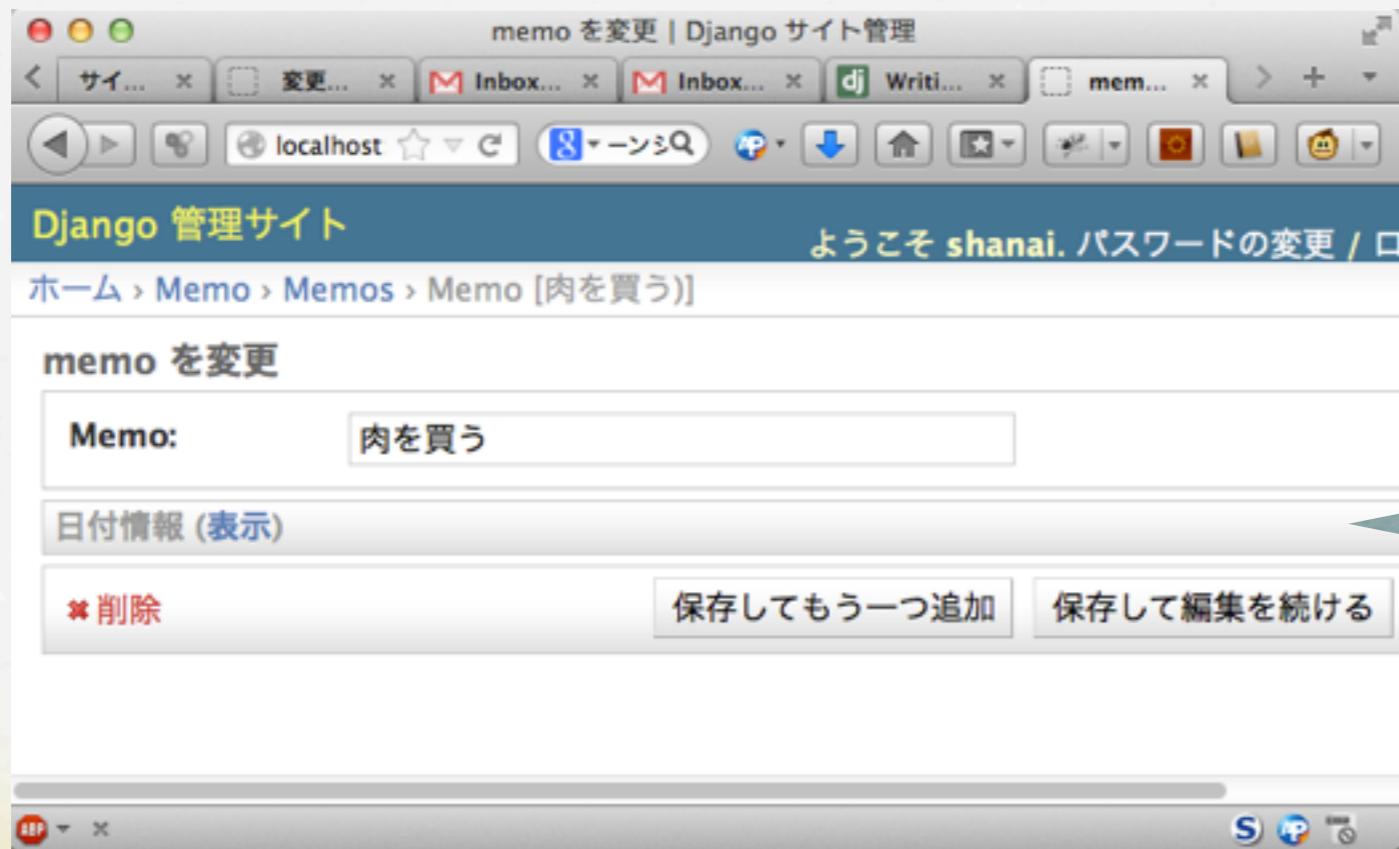


グルーピングされる。

管理サイトのカストマイズ

* CSS指定

```
class MemoAdmin(admin.ModelAdmin):
    fieldsets = [
        (None, {'fields': ['memo']}),
        ('日付情報', {'fields': ['create_date'], 'classes': ['collapse']}),
```



たたむことが出来るようになる。

関連モデルの編集

- * 関連するモデルと一緒に編集できる。

```
from django.contrib import admin
from memo.models import Memo, Category

class CategoryInline(admin.StackedInline):
    model = Category

class MemoAdmin(admin.ModelAdmin):
    fieldsets = [
        (None, {'fields': ['memo']}), 
        ('日付情報', {'fields': ['create_date'], 'classes': ['collapse']}),
    ]
    inlines = [CategoryInline]

admin.site.register(Memo, MemoAdmin)
```

関連モデルの編集

The screenshot shows the Django admin interface for the 'memo' model. The top navigation bar indicates the current view is 'memo を変更 | Django サイト管理'. The main content area is titled 'memo を変更' and shows a memo entry: '野菜を買う'. Below this, there is a 'Category' section with a table listing categories and their names. The first row shows 'Category: Category [買い物]' and 'Category name: 買い物'. A blue callout bubble points to this row with the text: 'Memoの編集画面で、Categoryを編集できる。' (You can edit the Category in the Memo's edit interface).

Category	Category name
Category: Category [買い物]	Category name: 買い物
Category: #2	Category name:
Category: #3	Category name:
Category: #4	Category name:

Memoの編集画面で、Categoryを編集できる。

関連モデルの編集

```
class CategoryInline(admin.TabularInline):  
    model = Category  
    extra = 0
```



一覧表示の変更



一覧表示の変更

```
class MemoAdmin(admin.ModelAdmin):  
    ...  
    inlines = [CategoryInline]  
    list_display = ('create_date', 'memo', 'was_created_recently')
```

The screenshot shows the Django Admin interface for the 'Memo' model. The title bar says '変更する memo を選択 | Django サイト管理'. The browser tab is 'localhost:9000/admin/memo/'. The main content area is titled 'Django 管理サイト' and 'ようこそ shanai. パスワードの変更 / ログアウト'. Below that, it shows the navigation path 'ホーム > Memo > Memos'. A button 'memo を追加 +' is visible. The main table has columns: '操作:' (Operations), '作成日' (Create Date), 'Memo', and 'Was created recently'. There are two rows of data:

操作:	作成日	Memo	Was created recently
<input type="checkbox"/>	2013年5月23日6:51:51	野菜を買う	True
<input type="checkbox"/>	2013年5月22日14:31:02	肉を買う	True

Below the table, it says '2 memos'.

全てのカラムが表
示されるように
なったが、見にく
い。

一覧表示の変更

memoのラベルも日本語で表示されるように

```
class Memo(models.Model):  
    memo = models.CharField('備忘録', max_length=200)  
    create_date = models.DateTimeField('作成日')  
    def __unicode__(self):  
        return "Memo [" + self.memo + "]"  
    def was_created_recently(self):  
        return self.create_date >= timezone.now() - datetime.timedelta(days=7)  
    was_created_recently.admin_order_field = 'create_date'  
    was_created_recently.boolean = True  
    was_created_recently.short_description = '最近のメモ'
```

was_created_recentlyのソート順を、create_dateカラムと同じにし、内容としては"True"という文字列ではなくチェックマークにする。
タイトルを「最近のメモ」にする。

一覧表示の変更

変更する memo を選択 | Django サイト管理

d Writing your first Django app, p... × 案 変更する memo を選択 | Django ... × +

localhost:9000/admin/ postolo AP ↴ ホーム フォルダ ログアウト

Django 管理サイト ようこそ shanai. パスワードの変更 / ログアウト

ホーム > Memo > Memos

memo を追加 +

変更する memo を選択

操作: ----- 実行 2個の内ひとつも選択されていません

	備忘録	最近のメモ
<input type="checkbox"/> 作成日	野菜を買う	野菜を買う
<input type="checkbox"/> 2013年5月23日6:51:51	肉を買う	肉を買う
<input type="checkbox"/> 2013年5月22日14:31:02		

2 memos

ABP × S AP

一覧表示の変更

```
class MemoAdmin(admin.ModelAdmin):  
    ...  
    list_display = ('create_date', 'memo', 'was_created_recently')  
    list_filter = ['create_date']
```

The screenshot shows the Django Admin interface for a 'Memo' model. The page title is '変更する memo を選択 | Django サイト管理'. The URL is 'localhost:9000/admin/memo/'. The main content area is titled 'Django 管理サイト' and shows a list of 'Memos'. There are two entries: '2013年5月23日6:51:51' and '2013年5月22日14:31:02'. To the right of the list is a sidebar with a 'memo を追加' button and a 'フィルター' section. The 'フィルター' section has a heading '作成日 で絞り込む' and a dropdown menu with options: 'いつでも', '今日', '過去 7 日間', '今月', and '今年'. A callout bubble points to the 'list_filter' field in the code above, indicating that this configuration adds a filter to the sidebar.

フィルタが付加される。

検索機能の追加

```
class MemoAdmin(admin.ModelAdmin):  
    ...  
    list_filter = ['create_date']  
    search_fields = ['memo']
```

変更する memo を選択 | Django サイト管理

dj Writing your first Django app, p... × 変更する memo を選択 | Django ... × +

localhost:9000/admin/ postolo

Django 管理サイト ようこそ shanai. パスワードの変更 / ログアウト

ホーム > Memo > Memos

変更する memo を選択

操作: ----- 実行 2個の内ひとつも選択されません

	備忘録	最近のメモ
<input type="checkbox"/> 作成日	野菜を買う	野菜を買う
<input type="checkbox"/> 2013年5月23日6:51:51	肉を買う	肉を買う
<input type="checkbox"/> 2013年5月22日14:31:02		

2 memos

いつでも
今日
過去 7 日間
今月
今年

検索機能が追加された。

レコード絞り込み

```
class MemoAdmin(admin.ModelAdmin):  
    ...  
    search_fields = ['memo']  
    date_hierarchy = 'create_date'
```

The screenshot shows the Django Admin interface for a 'Memo' model. The title bar says '変更する memo を選択 | Django サイト管理'. The browser tab is 'localhost:9000/admin/memo/'. The main content area is titled 'Django 管理サイト' and 'ようこそ shanai. パスワードの変更 / ログアウト'. Below it, the URL is 'ホーム > Memo > Memos'. A large green callout bubble points to the search bar, which contains the text '野菜' and '(全 2 件)'. To the right of the search bar is a filter sidebar with the heading 'フィルター' and options like '作成日で絞り込む', 'いつでも', '今日', '過去 7 日間', '今月', and '今年'. The main table lists two items: one for '野菜' on '2013 5月22日' and another for '野菜を買う' on '2013年5月23日6:51:51'. The table has columns for '操作', '作成日', '備忘録', and '最近のメモ'.

絞り込み機能

ビュー

モデルの動作を管理サイト機能で確認したので
ビューを作成します。

ビューの作成

memo/views.py

```
# coding=utf-8  
  
from django.http import HttpResponse  
  
def index(request):  
    return HttpResponse("備忘録へようこそ")
```

ビューは、リクエストを受け取り、レスポンスを返すメソッドとして実装する。

memo/urls.py

```
from django.conf.urls import patterns, url  
from memo import views  
  
urlpatterns = patterns(''  
    url(r'^$', views.index, name='index')  
)
```

正規表現を使って、urlのパターンと、ビューのメソッドを対応付ける。

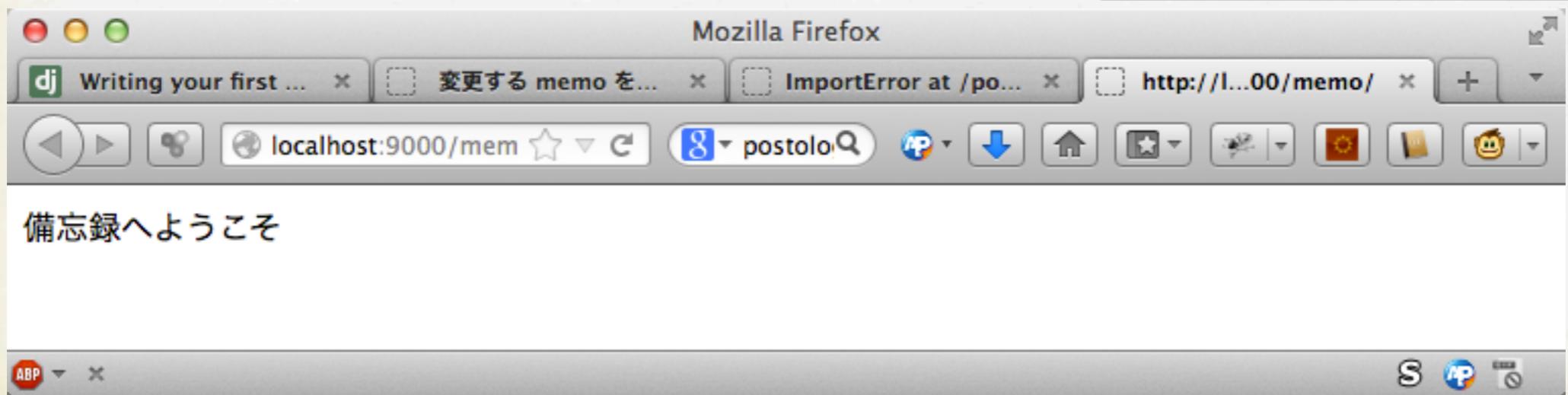
ビューの作成

myproject/urls.py

```
urlpatterns = patterns('',
...
    url(r'^admin/', include(admin.site.urls)),
    url(r'^memo/', include('memo.urls')),
)
```

/memo

サイト全体のURLマッピング。/memo/で始まるものをmemoアプリケーションへ。



urlパターン

- * Djangoは上から順にパターンにマッチするか調べて、マッチするものを採用する。ここではパラメータ、ドメイン名はマッチ対象にならない。

`http://ruimo.com/myapp/`

`http://ruimo.com/myapp/?page=3`

どちらも、`myapp/`というパターンにマッチする。

urlパターン

- * 正規表現にurlがマッチすると、viewに指定されたメソッドが呼び出される。
- * この時、引数としてHttpRequestオブジェクトと、正規表現でキャプチャされたものが渡される。
- * また任意の追加引数を渡すことができる。
- * パターンに名前を付けるとテンプレートで使用する時に便利。

詳細表示ビュー

memo/views.py

```
def detail(request, memo_id):
    return HttpResponse(u"備忘録%s" % memo_id)
```

memo/urls.py

```
urlpatterns = patterns('',
    url(r'^$', views.index, name='index'),
    url(r'^(?P<memo_id>\d+)/$', views.detail, name='detail'),
)
```

%による書式化を行う場合、uを頭に付けて

Unicode文字列にしないとエラーになることがある。

¥d+にマッチした部分をキャプチャして、memo_idという変数に設定する。これによりdetailメソッドのmemo_idに渡る。

URL処理の流れ

- * /memo/1/がリクエストされる。
- * settings.pyのROOT_URLCONFを参照。
ROOT_URLCONF = 'myproject.urls'
- * myproject.urlsを参照し、上から一致するものを探す。

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

パターンに\$が入っていない
ことに注意。URLの最初の部
分にマッチする。

includeメソッドは、マッチした部分を捨て、URLの残り部分
(1/)を、memo.urlsに渡す。

URL処理の流れ

- * memo/urls.pyを先頭から調べる。
url(r'^(?P<memo_id>\d+)/\$', views.detail,
name='detail'),
- * 1/の中の1の部分がmemo_idに渡る。
- * views.detail(request, 1)が実行される。

テンプレート

テンプレート

- * ビューで直接htmlを生成して返すこともできるが、ロジックとビューが混在して分かりにくくなる。
- * テンプレートを用いると、ロジックを分離することができる。

テンプレートエンジン

- * デフォルトで2つのテンプレートエンジンが登録されている。

settings.py

```
TEMPLATE_LOADERS = (
    'django.template.loaders.filesystem.Loader',
    ''django.template.loaders.app_directories.Loader',
    #     'django.template.loaders.eggs.Loader',
)
```

- * `django.template.loaders.app_directories.Loader`は、`templates`というサブディレクトリを、`INSTALLED_APPS`の中から探し出す。

テンプレート

- * memoの下にtemplatesディレクトリを作成し、その下にmemoディレクトリを作つて、テンプレートを格納する。つまり、index.htmlというテンプレートのパスは以下のようになる。
memo/templates/memo/index.html
- * アプリケーションからは、memo/index.htmlでアクセスする。

テンプレート

memo/views.py

```
from django.http import HttpResponseRedirect  
from django.template import Context, loader  
  
from memo.models import Memo  
  
def index(request):  
    latest_memo_list = Memo.objects.order_by(' -create_date')[:5]  
    template = loader.get_template('memo/index.html')  
    context = Context({  
        'latest_memo_list': latest_memo_list,  
    })  
    return HttpResponseRedirect(template.render(context))
```

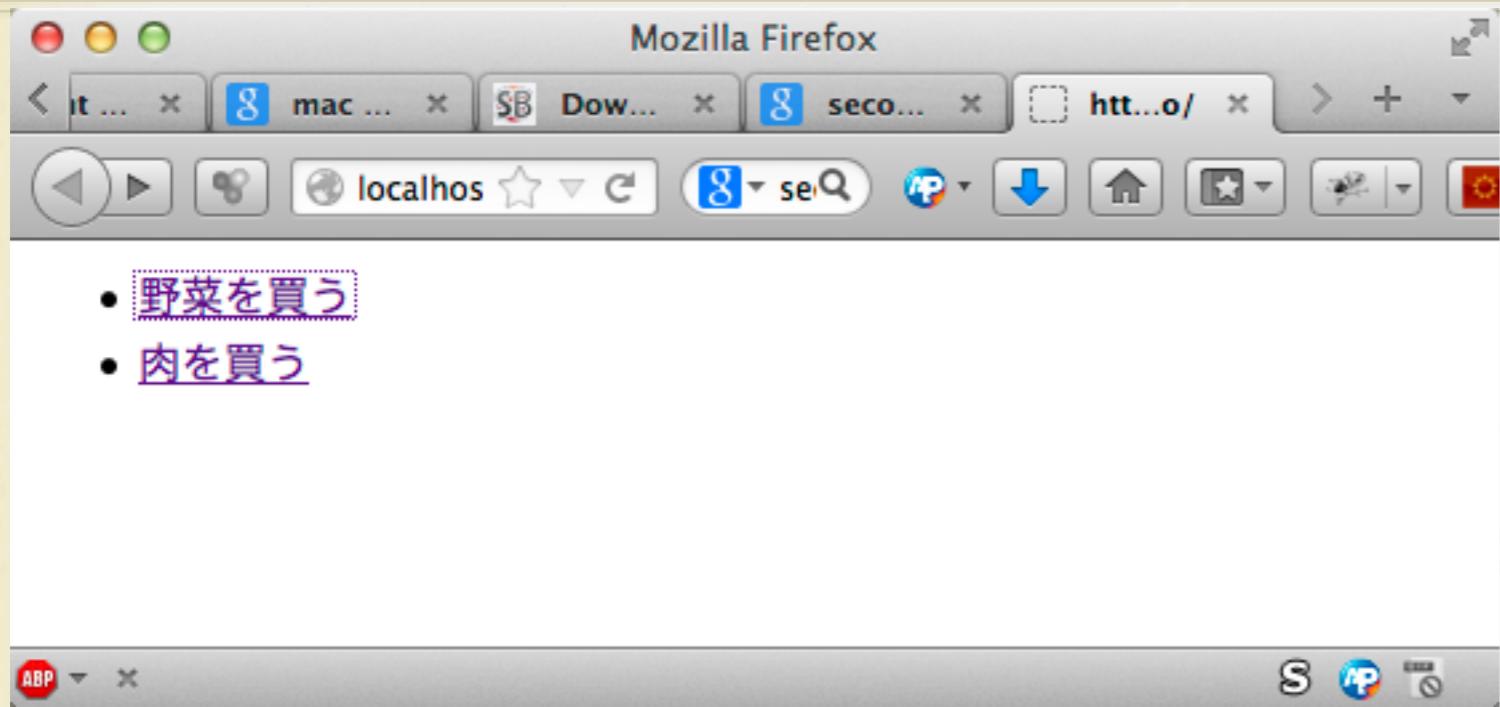
テンプレートへのデータの受け渡しは、Contextを用いる。

テンプレート

memo/templates/memo/index.html

```
{% if latest_memo_list %}  
  <ul>  
    {% for memo in latest_memo_list %}  
      <li><a href="/memo/{{ memo.id }}/">{{ memo.memo }}</a></li>  
    {% endfor %}  
  </ul>  
{% else %}  
  <p>備忘録が1つもありません。</p>  
{% endif %}
```

テンプレート



renderを使うと
もっと簡単に書ける。

```
from django.http import HttpResponseRedirect
from django.shortcuts import render

from memo.models import Memo

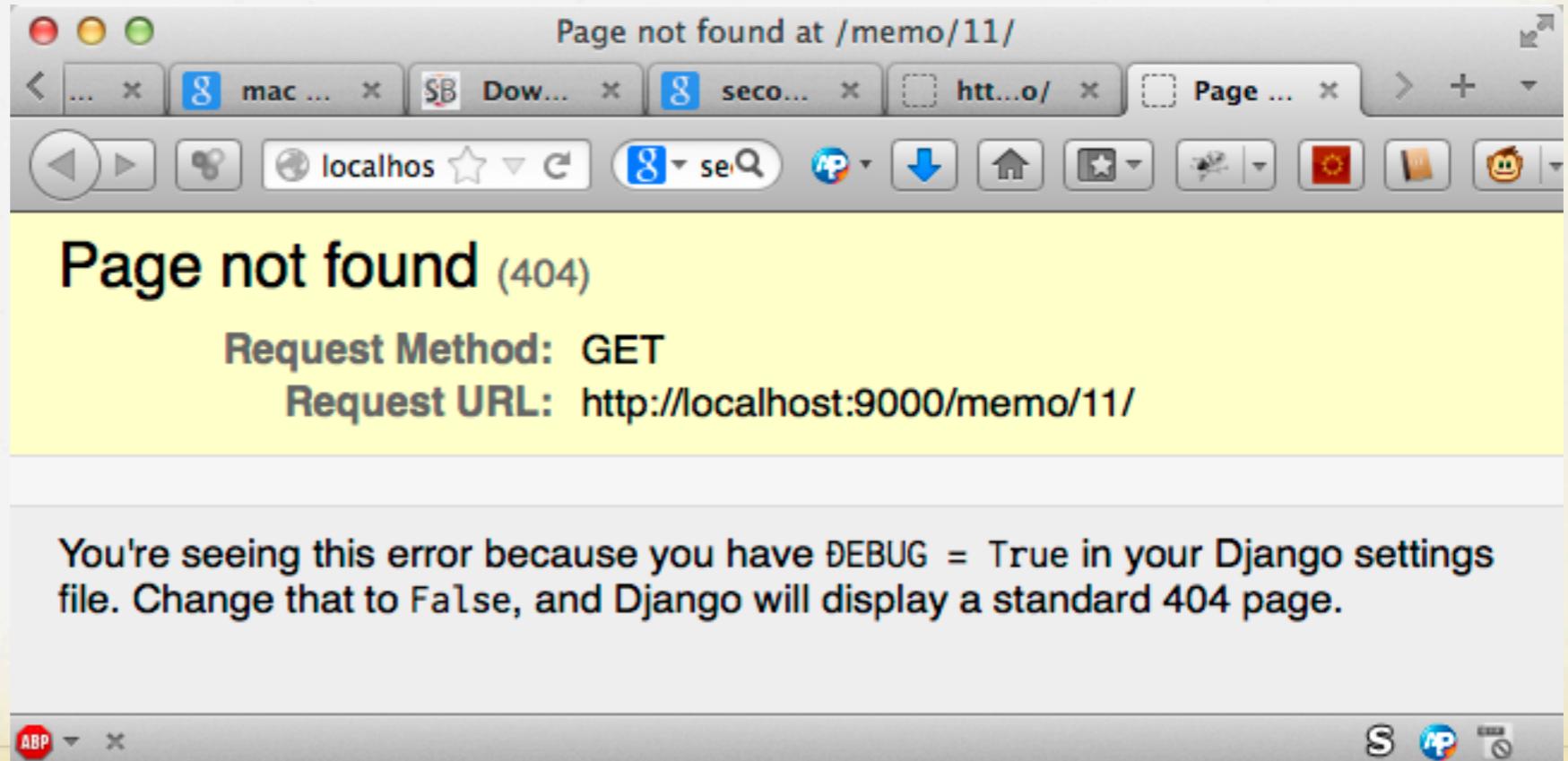
def index(request):
    latest_memo_list = Memo.objects.all().order_by('-create_date')[:5]
    context = {'latest_memo_list': latest_memo_list}
    return render(request, 'memo/index.html', context)
```

エラーページ

404エラー

```
from django.http import Http404
...
def detail(request, memo_id):
    try:
        memo = Memo.objects.get(pk=memo_id)
    except Memo.DoesNotExist:
        raise Http404
    return render(request, 'memo/detail.html', {'memo': memo})
```

ビューで404ページを表示するには、Http404をスローする。



404エラー

- * ショートカットを使うともっと短かく書ける。

```
from django.shortcuts import render, get_object_or_404
...
def detail(request, memo_id):
    memo = get_object_or_404(Memo, pk=memo_id)
    return render(request, 'memo/detail.html', {'memo': memo})
```

- * `get_object_or_404`は、レコードが無ければ
Http404例外をスローする。

404ページ

- * Http404が投げられた場合、rootのurls.pyに指定されたhandler404が使用される。
- * handler404が無い場合は、
`django.views.defaults.page_not_found()`が使
用される。
- * これらはsettings.pyでDEBUG=Falseの時に有
効。

なお、デフォルトのsettings.pyでは、ALLOWED_HOSTSが空
なので、Debug=Falseにすると全てのリクエストが500エラーに
なるので注意。開発時は'localhost'を入れておく。

404ページ

- * ページのデザインのみを変更したい場合は、テンプレートディレクトリのトップに404.htmlを置く。
- * ビューを使用して返したい場合は、rootの urls.pyで
handler404 = 'myproject.views.my404'
などのようにビューを指定する。
- * なおurls.pyのパターンいずれにもマッチしない場合にも404ページが返る。

500ページ

- * アプリケーションで例外がスローされて、アプリケーション内で処理されないと表示される。
- * ページデザインを変更したい場合は、テンプレートディレクトリのトップに500.htmlを置く。
- * ビューを使って返したい場合は、rootのurls.pyにhandler500を用意する。

テンプレートの詳細

テンプレート詳細

- ※ ドットによる表記は、まずディクショナリとして解決を試みる。
- ※ 上記で解決できない場合は属性としてルックアップする。
- ※ 上記で解決できない場合はリストの添字としてルックアップする。

URLの逆変換

- * 以下のように書くと、urlとビューの対応を変更しようとすると、テンプレートを書き直さなければならなくなる。

```
<a href="/memo/  
{{ memo.id }}/">{{ memo.memo }}</a>
```

単に変数を出力するだけの場合は、`{{...}}`で指定できる。
出力内容はエスケープされる。

URLの逆変換

- * 以下のように書くとビューからURLに逆変換できる。

```
<a href="{% url 'detail' memo.id %}">{{ memo.memo }}</a>
```

- * この名前は、urls.pyの以下から来ている。

```
url(r'^(?P<memo_id>\d+)/$', views.detail,  
name='detail'),
```

ただし、このような名前付けだと複数のアプリケーションで衝突する可能性がある。これを解決するには名前空間を用いる。

URLの逆変換

* 名前空間定義

myproject/urls.py

```
url(r'^memo/', include('memo.urls', namespace="memo")),
```

templates/memo/index.html

```
<a href="{% url 'memo:detail' memo.id %}">{{ memo.memo }}</a>
```

フォーム送信

メモ作成ページ

memo/urls.py

```
url(r'^create_start$', views.create_memo_start, name='create_memo_start'),
```

memo/views.py

```
def create_memo_start(request):
    return render(request, 'memo/create_memo.html')
```

memo/create_memo.html

<h1>備忘録の作成</h1>

```
{% if error_message %}<p><strong>{{ error_message }}</strong></p>{% endif %}

<form action="{% url 'memo:create_memo' %}" method="post">
  {% csrf_token %}
    <label for="memo">メモ</label>
    <input id="memo" type="text" name="memo" >
<input type="submit" value="作成" />
</form>
```

CSRF脆弱性対策のためのタグを出力する。

メモ作成ページ

memo/urls.py

```
url(r'^create$', views.create_memo, name='create_memo'),
```

memo/views.py

```
def create_memo(request):
    Memo(
        memo = request.POST['memo'],
        create_date = timezone.now()
    ).save()
    return HttpResponseRedirect(reverse('memo:index'))
```

メッセージ

メッセージ

- * リクエストをまたぐメッセージの出力(Railsのflash記憶)。

memo/views.py

```
def create_memo(request):
    Memo(
        memo = request.POST['memo'],
        create_date = timezone.now()
    ).save()
    messages.success(request, 'メモを追加しました')
    return HttpResponseRedirect(reverse('memo:index'))
```

リダイレクト先のページに出力される。

メッセージ

templates/memo/index.html

```
{% if messages %}  
<ul class="messages">  
  {% for message in messages %}  
    <li{% if message.tags %} class="{{ message.tags }}"{% endif %}>  
      {{ message }}  
    </li>  
  {% endfor %}  
</ul>  
{% endif %}
```

テスト

単体テスト

- * デフォルトで、tests.pyが用意されている。

```
from django.test import TestCase

class SimpleTest(TestCase):
    def test_basic_addition(self):
        """
        Tests that 1 + 1 always equals 2.
        """
        self.assertEqual(1 + 1, 2)
```

TestCaseのサブクラスを探して実行する。

実行

```
device-ea2fd6:myproject shanai$ ./manage.py test memo
Creating test database for alias 'default'...
```

```
Ran 1 test in 0.001s
```

```
OK
```

```
Destroying test database for alias 'default'...
```

テスト用のデータベースが用意される。

ビューのテスト

- * DjangoにはWebクライアントが用意されている。

shellで実行した例：

```
>>> from django.test.utils import setup_test_environment  
>>> setup_test_environment()  
>>> from django.test.client import Client  
>>> client = Client()  
>>> response = client.get('/  
Not Found: /  
>>> response.status_code  
404  
>>> from django.core.urlresolvers import reverse  
>>> response = client.get(reverse('memo:index'))  
>>> response.status_code  
200
```

このClientには、TestCaseを継承したクラス内であれば、
self.clientで取得できる。とはいえたまり複雑なことはできないの
で、Seleniumを使う。

テストサーバ

```
from django.test import TestCase
from django.test import LiveServerTestCase
from selenium.webdriver.firefox.webdriver import WebDriver
import selenium.webdriver.support.ui as ui

class MySeleniumTests(LiveServerTestCase):
    @classmethod
    def setUpClass(cls):
        cls.selenium = WebDriver()
        super(MySeleniumTests, cls).setUpClass()

    @classmethod
    def tearDownClass(cls):
        cls.selenium.quit()
        super(MySeleniumTests, cls).tearDownClass()
```

テスト前にDjangoサーバを起動し、テスト後にシャットダウンする。

Seleniumとの統合

- * Python用WebDriverのインストール。

```
sudo pip install selenium
```

live_server_urlでサーバのurl(ドメイン名まで)が取得できる。

```
def test_create_memo(self):  
    self.selenium.get('%s%s' % (self.live_server_url, '/memo/create_start'))  
    username_input = self.selenium.find_element_by_id("memo")  
    username_input.send_keys(u'テストメモ')  
    self.selenium.find_element_by_xpath('//input[@value="作成"]').click()  
  
    wait = ui.WebDriverWait(self.selenium, 10)  
    wait.until(lambda d: d.title == 'Memo list')  
    message = self.selenium.find_element_by_xpath('//li[@class="success"]')  
    self.assertHTMLEqual(u'メモを追加しました', message.text)  
    memo = self.selenium.find_element_by_xpath('//ul/li/a')  
    self.assertHTMLEqual(u'テストメモ', memo.text)
```

HTMLEqualを使うと、スペースの有無やタグ内の属性順序の違いを無視してくれる。

まとめ

- ※ 非常にシンプルな構造を持つフレームワーク
- ※ モデルは、ORマッパであるが抽象度が低いので学習コストが低い(関連は外部キーをそのまま使用)。
- ※ コントローラは正規表現のみ。
- ※ ロジックはビューとモデルに実装。

Railsとの比較(弱み)

- * モジュール管理

デフォルトではモジュール管理(複数バージョンの共存)に難がある。別途virtualenvなどを使用する必要があつて煩雑。

- * Functional/Integrationテスト

Railsと比較すると支援機能が若干薄い。

- * アセットパイプライン

Railsのアセットパイプラインは標準では提供されていない。

Railsとの比較(弱み)

- * CSRFトークンが無い時の動作は403エラー
Railsの場合はセッションがクリアされるだけ。外部サービスとの連携で制限が発生するかもしれない。
- * 情報が少ない。
WebDriverのドキュメントなど、普通に「ソース読んでね」と書かれている。
- * 開発環境(IDE)に乏しい。
PyCharm(JetBrain)が使えるかもしれない(未評価)

Railsとの比較(強み)

- * 単純で学習コストが低い
RailsのActiveRecordは、ある程度複雑なケースになると結構難しい。
- * 環境の構築が簡単
Rubyは、環境の構築が結構難しい(OpenSSLなど)。
- * Seleniumとの統合が進んでいる
別にサーバを立ち上げたりせず、テストクラス内でテスト環境が完結する。
- * Fixtureの指定をテストクラスごとに行える。

Railsとの比較(強み)

- * モジュールに分けることで再利用をし易い。特に大規模の開発ではRailsよりも管理し易いと思われる。
- * Railsよりも暗黙の規約(routes内の指定など)が少なく学習コストが低い。
- * 管理画面の作成が非常に強力。割り切りができるなら、単純なCRUDアプリケーションを低成本で開発可能。Railsのscaffoldは、そのままでは、ほとんど使い物にならない。