Django 最初のアプリケーション 1 (環境設定～モデルの作成)

[Django][Python][次]

Python の概要も分かり易い .

・The Django Book

・http://docs.djangoproject.com/en/dev/intro/tutorial01/#intro-tutorial01

を参考にサンプルアプリケーションを作成してみる

Django がインストールされている

・Django インストール

import django が python インタプリタから行えること

```
# python
Python 2.6.2 (r262:71600, Jun 13 2009, 02:28:29)
[GCC 4.1.2 20070626 (Red Hat 4.1.2-13)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import django
```

Help を確認

```
>>> help(django)
Help on package django:

NAME
    django

FILE
    /usr/local/lib/python2.6/site-packages/django/__init__.py

PACKAGE CONTENTS
    bin (package)
    conf (package)
    contrib (package)
       :
```

プロジェクトの作成

■ django-admin.py コマンドの実行

・Django を最初に利用する場合、初期化処理を行う必要がある
・コードを置こうと思っているディレクトリで、以下のコマンドを実行 (mysite ディレクトリが作成される)

```
# django-admin.py startproject mysite
```

・

プロジェクト名に、django や test は避けること

・django-admin.py は、Django を setpu.py でインストールしたシステムパス (/usr/local/bin のような) にある。

```
# whereis django-admin.py
django-admin: /usr/local/bin/django-admin.py
```

■作成されるファイル

・startproject は以下のファイルを作成

```
# tree
.
`-- mysite
    |-- __init__.py
    |-- manage.py
    |-- settings.py
    `-- urls.py
```

ファイル	概要
____ init ____ .py	このディレクトリがパッケージだと Python に伝える空のファイル
manage.py	プロジェクトと対話するためのコマンドライン ユーティリティ
settings.py	プロジェクトの設定ファイル
urls.py	プロジェクトの URL 宣言

開発サーバー

・mysite ディレクトリへ入って、以下を実行

```
# python manage.py runserver
Validating models...
0 errors found

Django version 1.0.2 final, using settings 'mysite.settings'
Development server is running at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

・開発サーバーを起動すると、Python で書かれた軽量サーバーが http://127.0.0.1:8000/ で起動する

■起動ポート、IP アドレスを変更する

・デフォルトで開発サーバーはポート 8000 で起動するが、ポートを指定して起動することもできる。

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

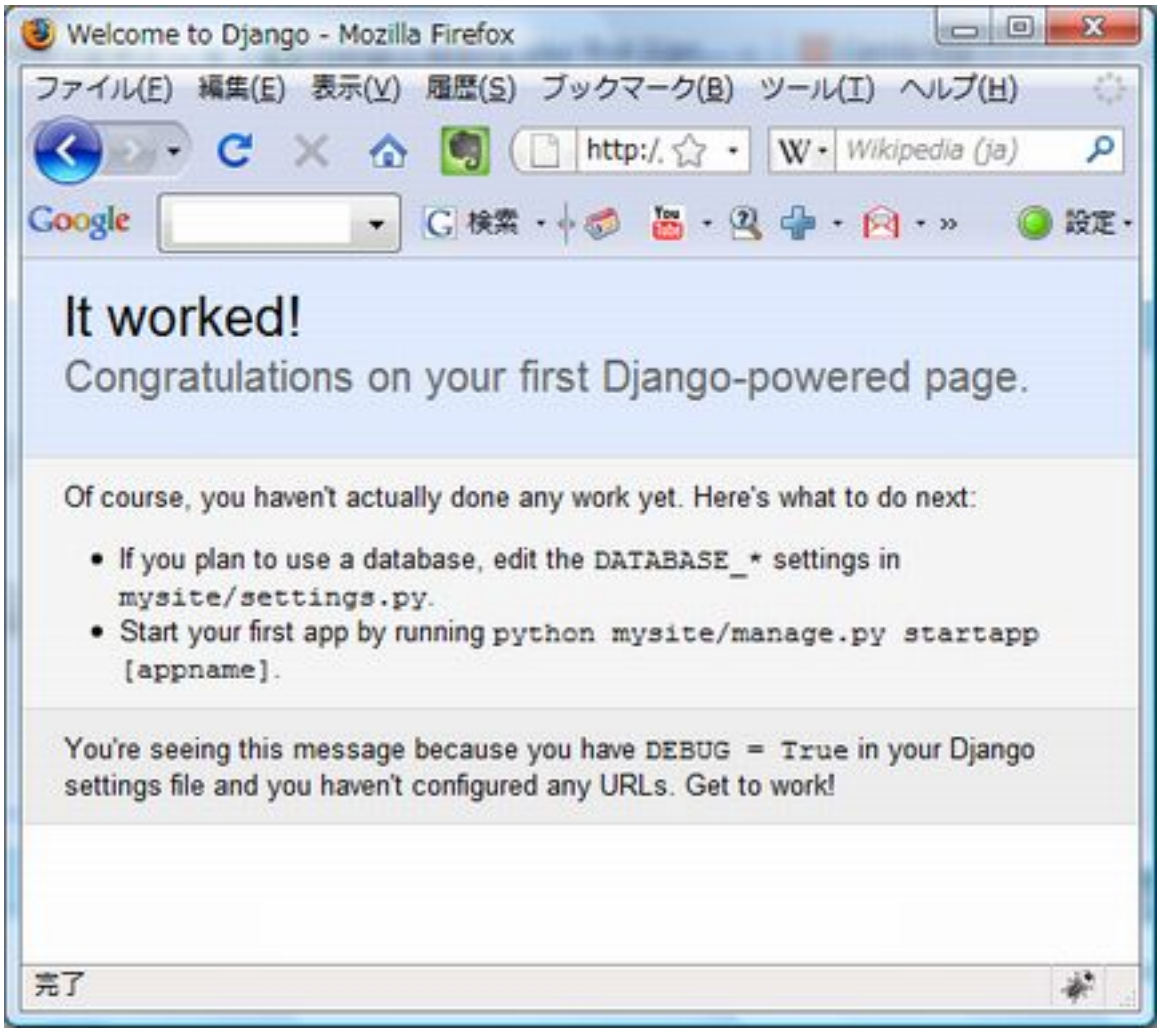
・IP アドレスはローカルホストしかリッスンしていないようだが、指定することができる

```
# python manage.py runserver 192.168.24.14:8080
Validating models...
0 errors found

Django version 1.0.2 final, using settings 'mysite.settings'
Development server is running at http://192.168.24.14:8080/
```

```
Quit the server with CONTROL-C.
```

起動



データベースの設定

・データベースの接続設定に応じて、setting.py を編集する。

設定項目	内容
DATABASE_ENGINE	'postgresql_psycopg2'、'mysql'、'sqlite3' その他
DATABASE_NAME	データベース名。SQLite の場合、絶対パス名存在しない場合、初回自動的に作成される。
DATABASE_USER	データベースユーザー名。SQLite の場合不要
DATABASE_PASSWORD	データベースパスワード。SQLite の場合不要
DATABASE_HOST	データーベースのホスト。設定しない場合、物理的に同じマシンとみなされる。SQLite では不要

■ SQLite は、Pytho2.5 以降に含まれる

・SQLite を使用する場合、DATABASE_ENGINE に sqlite3 を指定する
・Python2.5 以降に含まれるため、他にインストール等不要

SQLite を指定した場合準備は不要。データベースファイルは必要になったときに作成される

■ PostgreSQL を利用してみる

・PostgreSQL 8.3.5 インストール
・PostgresSQL 8.3.5 起動と停止
・PostgreSQL pgAdmin のインストール

```
DATABASE_ENGINE = 'postgresql_psycopg2'
DATABASE_NAME = 'testdb'
DATABASE_USER = 'postgres'
DATABASE_PASSWORD = '******'
DATABASE_HOST = ''
DATABASE_PORT = '5432'
```

■ psycopg Python-PostgreSQL Database Adapter のインストール

http://www.initd.org/pub/software/psycopg/
http://www.initd.org/pub/software/psycopg/psycopg2-2.0.11.tar.gz

・次のステップで、以下を行うとエラー

```
# python manage.py syncdb
  :
raise ImproperlyConfigured("Error loading psycopg2 module: %s" % e)
```

django.core.exceptions.ImproperlyConfigured: Error loading psycopg2 module: No module named psycopg2

psycopg のインストール

```
# wget http://www.initd.org/pub/software/psycopg/psycopg2-2.0.11.tar.gz
# tar zxvf psycopg2-2.0.11.tar.gz
```

・解凍してできたディレクトリで、以下を実行

```
# python setup.py install
```

・インポートできるか？

```
#python
>>> import psycopg2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/usr/local/lib/python2.6/site-packages/psycopg2/__init__.py", line 60, in <module>
    from _psycopg import BINARY, NUMBER, STRING, DATETIME, ROWID
ImportError: libpq.so.5: cannot open shared object file: No such file or directory
```

・libpq.so が見つからないエラー

```
# LD_LIBRARY_PATH=/usr/local/pgsql/lib:$LD_LIBRARY_PATH
```

OK

■ syncdb の実行

PostgreSQL を起動しておく

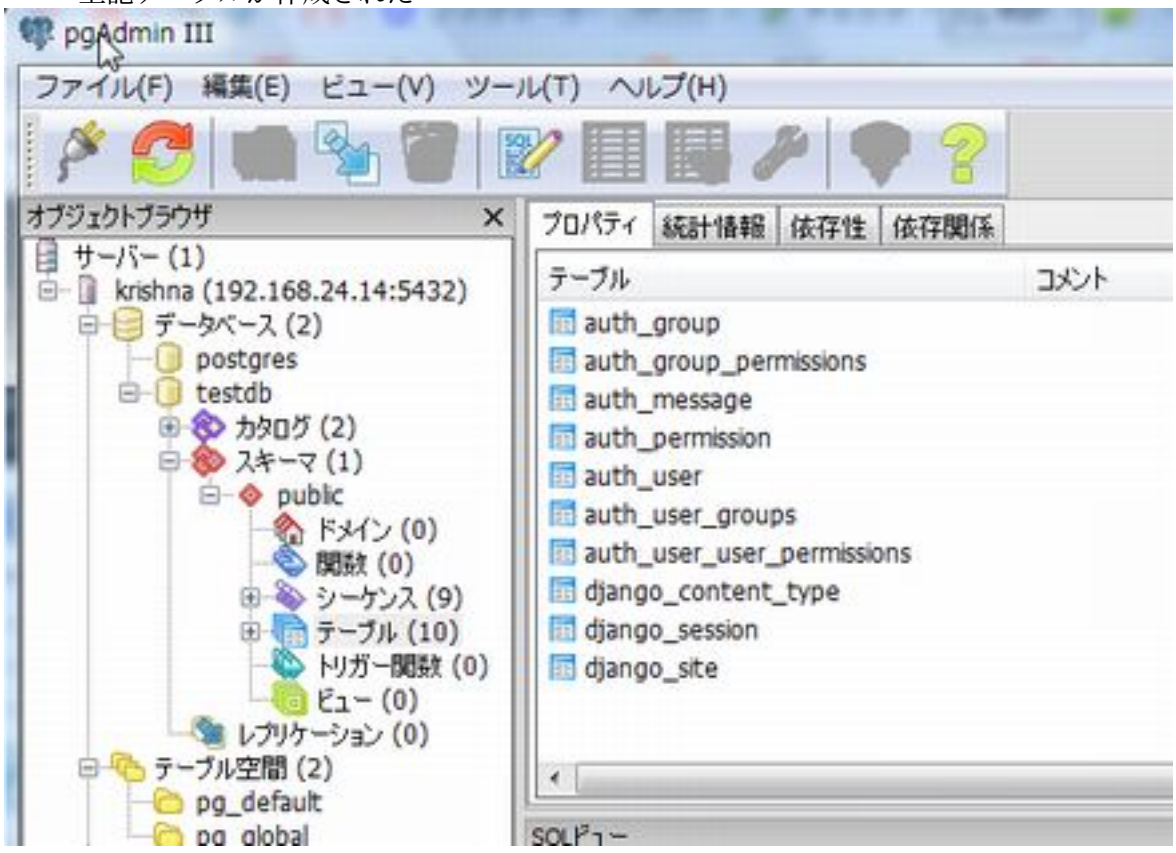
```
#pg_ctl start
```

syncdb コマンドの実行

```
# python manage.py syncdb
Creating table auth_permission
Creating table auth_group
Creating table auth_user
Creating table auth_message
Creating table django_content_type
Creating table django_session
Creating table django_site

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 'root'):
Error: That e-mail address is invalid.
E-mail address: piroto@typea.info
Password:
Password (again):
Superuser created successfully.
Installing index for auth.Permission model
Installing index for auth.Message model
```

・上記テーブルが作成された



INSTALLED_APPS 設定

・INSTALLED_APPS 変数は、この Django インスタンスからアクティベートされるすべての Django アプリケーションで保持される

```
INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
)
```

モデルの生成

■アプリケーションの作成

・ここまでで、プロジェクト環境は作成された
・Django のアプリケーションは、Python のパッケージ

プロジェクトは設定のコレクションであり、アプリケーションは特定の Web サイト。プロジェクトは複数のアプリケーションを含み、アプリケーションは複数のプロジェクトに属すことができる

サンプルアプリケーション (polls) の作成

```
# python manage.py startapp polls
```

作成されるファイル

```
# tree ./polls
./polls
|-- __init__.py
|-- models.py
 -- views.py
```

■モデルの作成

・Django Web アプリケーションでは、モデルを定義する
・単純なサンプルアプリケーション (poll) では、2 つのモデル polls、choices を作成する
・poll は、質問と発表日、choice は選択テキストと同意への投票の 2 つのフィールドを持つ
・/polls/models.py に記述

```
from django.db import models

class Poll(models.Model):
        question = models.CharField(max_length=200)
        pub_date = models.DateField('date published')

class Choice(models.Model):
        poll = models.ForeignKey(Poll)
        choice = models.CharField(max_length=200)
        votes = models.IntegerField()
```

■モデルの開始

■ settings.py を以下のように編集

```
INSTALLED_APPS = (
    'django.contrib.auth',
```

```
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'mysite.polls'
)
```

■ DDL の確認～テーブルの生成

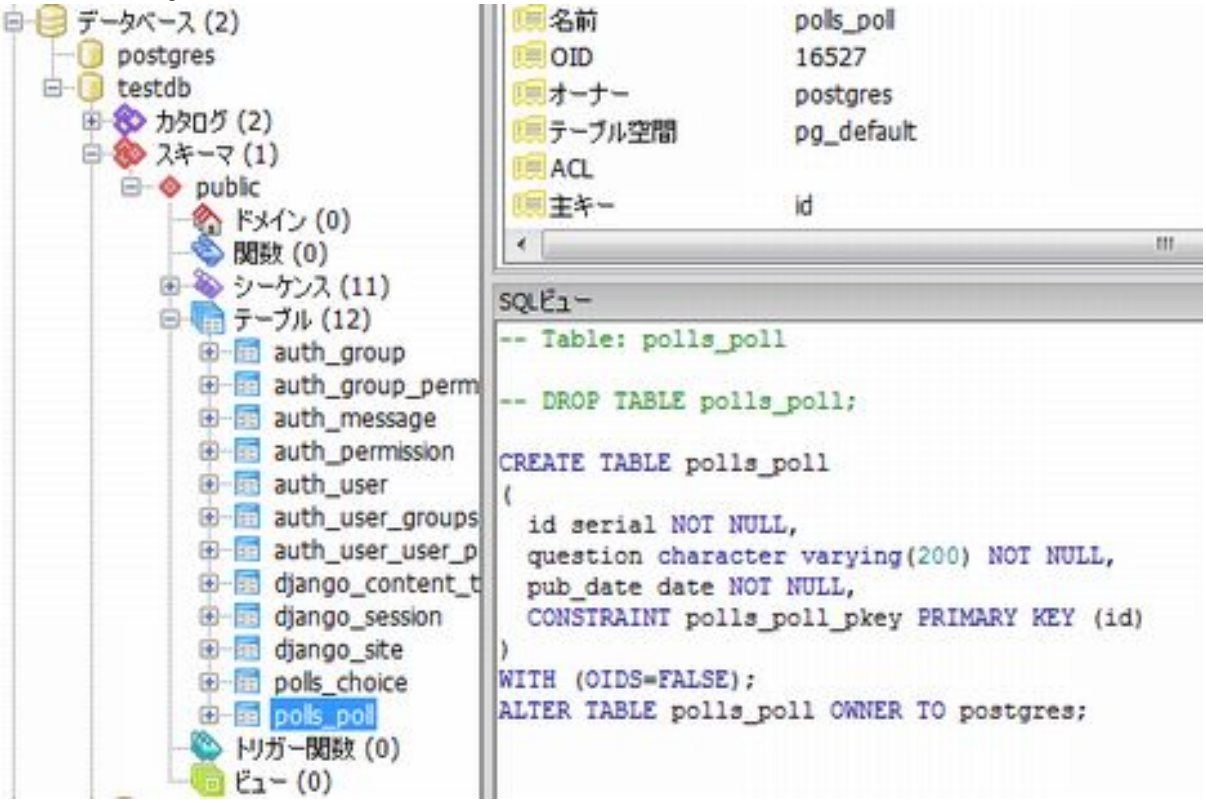
・Django は、polls を含む mysite を認識しているので、以下のコマンドを実行
・データベースを起動しておく

DDL を発行し、テーブルを作成する

```
# python manage.py sql polls
BEGIN;
CREATE TABLE "polls_poll" (
    "id" serial NOT NULL PRIMARY KEY,
    "question" varchar(200) NOT NULL,
    "pub_date" date NOT NULL
)
;
CREATE TABLE "polls_choice" (
    "id" serial NOT NULL PRIMARY KEY,
    "poll_id" integer NOT NULL REFERENCES "polls_poll" ("id") DEFERRABLE   INITIALLY DEFERRED,
    "choice" varchar(200) NOT NULL,
    "votes" integer NOT NULL
)
;
COMMIT;
```

テーブルの生成

```
# python manage.py syncdb
Creating table polls_poll
Creating table polls_choice
Installing index for polls.Choice model
```



API から操作してみる

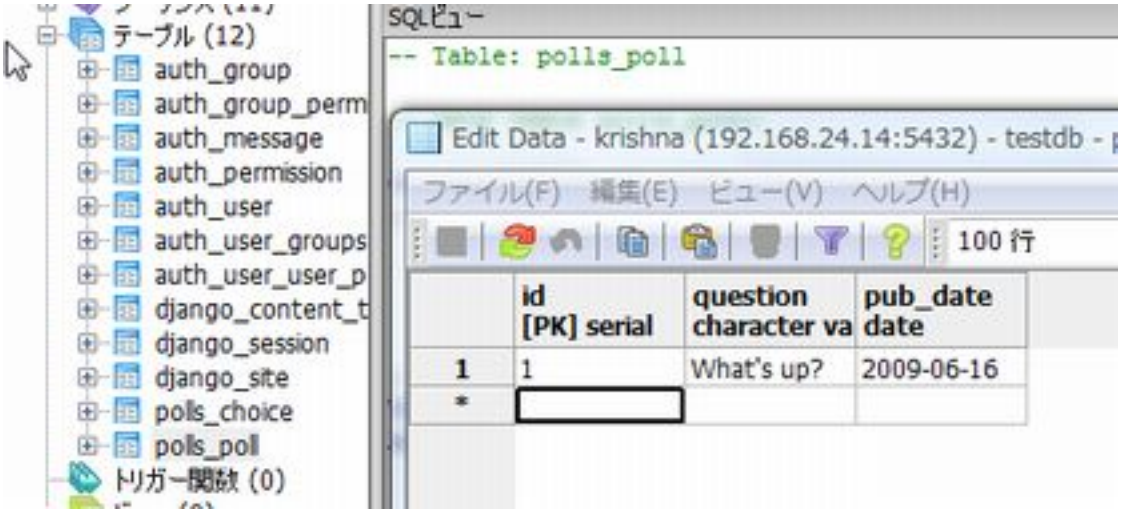
・manage.py を指定することで、プロジェクト環境が設定される
　・mysite を sys.path に設定
　・DJANGO_SETTINGS_MODULE 環境変数の設定

```
# python manage.py shell
```

Shell から database API を利用

```
>>> from mysite.polls.models import Poll,Choice # モデルとクラスのインポート
# まだ polls は存在していない
>>> Poll.objects.all()
[]
#poll を生成
>>> import datetime
>>> p = Poll(question="What's up?",pub_date=datetime.datetime.now())
>>> p.save()
>>> p.id
1L
#Python の属性を経由してデータベースのカラムにアクセス
>>> p.question
"What's up?"
>>> p.pub_date
datetime.datetime(2009, 6, 16, 8, 54, 31, 757771)
>>> p.save()
# データベース上の poll を表示
>>> Poll.objects.all()
[<Poll: Poll object>]
>>>
```

データベース上にデータが作成されている



■____ unicode ____メソッドの追加

```
>>> Poll.objects.all()
[<Poll: Poll object>]
```

・この出力では、このオブジェクトを表現するには役に立っていないので、polls/models.py のクラスに____ unicode ____メソッドを追加する

```
class Poll(models.Model):
        ⋮
        def __unicode__(self):
```

```
                return self.question
class Choice(models.Model):
        :
        def __unicode__(self):
                return self.choice
```

・出力がこうなる

```
>>> Poll.objects.all()
[<Poll: What's up?>]
```

■カスタムメソッドの追加

・python 通常のメソッドを追加して、簡単なデモ

```
import datetime
class Poll(models.Model):
        :
        def was_published_today(self):
                return self.pub_date == datetime.date.today()


>>> from mysite.polls.models import Poll,Choice

#Django は高機能な lookup API を提供
>>> Poll.objects.filter(id=1)
[<Poll: What's up?>]
>>> Poll.objects.filter(question__startswith='What')
[<Poll: What's up?>]
>>> p = Poll.objects.get(id=1)

# カスタムメソッドが動作することを確認
>>> p.was_published_today()
True

#INSERT ステートメントの記述なしに、create の呼び出しで Choice オブジェクトを生成する
# 外部キー制約を指定したので choice_set が有効
>>> p.choice_set.create(choice='Not much',votes=0)
<Choice: Not much>
>>> p.choice_set.create(choice='The sky',votes=0)
<Choice: The sky>
>>> c = p.choice_set.create(choice='Just hacking again',votes=0)
>>> Choice.objects.filter(poll__pub_date__year=2009)
[<Choice: Just hacking again>, <Choice: The sky>, <Choice: Not much>]
>>> c = p.choice_set.filter(choice__startswith='Just hacking')
>>> c.delete()
```

API は自動的にリレーションシップ (参照整合性制約) をフォローする。アンダースコアの 2 重続きは、リレーションシップを区別する

[次]