はじめに

近年、Webアプリケーション開発で用いるプログラミング言語として、Lightweight Language（以降LL）と呼ばれるスクリプト言語が人気を博しています。本稿では、そのスクリプト言語の中から**Python**とWebアプリケーションフレームワークの**Django**（ジャンゴと読む）を紹介します。

Pythonの大きな特徴として、「言語仕様が小さくシンプルであり、簡潔で読みやすいアプリケーションを作れる」という点が挙げられます。DjangoはPythonの簡潔さをうまく活かし、シンプルかつ本格的な開発ができるWebアプリケーションフレームワークです。本稿ではこのDjangoによるアプリケーション開発の基本をチュートリアル形式で説明します。

対象読者

- PythonによるWebアプリケーション開発に興味がある方
- 日頃、Perl、Ruby、PHP、Java、C#などPython以外のプログラミング言語で開発している方
- Webアプリケーションの開発をこれから学ぶ方、もしくは学びはじめたばかりの方。

必要な環境

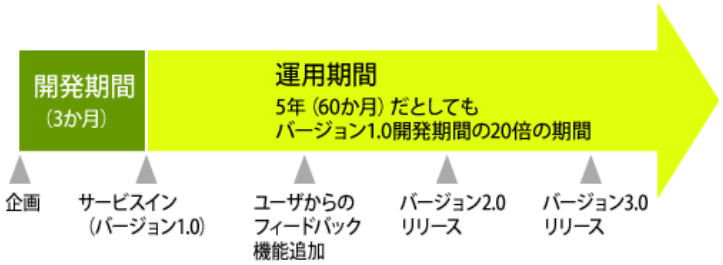
- Windows XP
- Python 2.6.2
- Django 1.1 Beta
- SQLite3

執筆時点（2009年6月29日）では、Djangoの正式バージョンは1.0.2ですが、間もなく1.1がリリースされる模様です。本稿では1.1Betaを使用します。

システムのライフサイクル

プログラミング言語を選択する際の視点として、開発生産性やとっかかりの良さだけに目が行きがちです。もちろんその2つはとても重要な要素ですが、システムはバージョン1.0をリリースしたらそれでおしまいということはほとんどなく、その後も機能追加、修正を実施していくことがほとんどです。むしろ、**ライフサイクルでみるとバージョン1.0の開発期間よりも、リリースされてからの時間の方が圧倒的に長い**と言えるでしょう。それを示したのが図1です。

図1：システムのライフサイクル



バージョン1.0からの時間の方が圧倒的に長く、企業ではジョブローテーションなどで人が入れ替わることを考えると、**作られたプログラムが読みやすいかという点、すなわち保守性の高さは言語選択において重要なポイント**であると言えます。

各スクリプト言語の思想・哲学

プログラミング言語は、それぞれに言語思想・哲学を持っています。例えばPerlは、TIMTOWTDI（there's more than one way to do it - あることをするのにいくつものやり方がある）という哲学を持っています。Rubyもその哲学を引き継ぎ、「多様性は善（Diversity is Good）」という哲学を持っています。すなわちPerlとRubyは言語のもともとの思想として、一つのことを実現するのに、プログラマによりさまざまな書き方があることを善とし、奨励するというスタンスであり、当然その言語を使用して開発されたプログラムは、規約などを定めない限り、開発する人によりさまざまに異なってくるということになります。

一方、Pythonは言語自身の機能をできるだけ小さくおさえ、ユーザーがいつも必要とする最小限の機能のみを提供するという思想で言語が設計されています。これより、Pythonでは同一の仕事をするプログラムは、大体どれも同じようなコードにおさまるという特徴を持ちます。これがPythonで書かれたコードの読みやすさ、保守性の高さに直結します。このようなことから、筆者は、エンタープライズのウェブサービスのプログラミング言語としてPythonに注目しています。

GoogleのPython採用

世界的なIT企業であるGoogleでは使用するプログラミング言語を3つ定めています。それはC++、Java、Pythonの3つです。Googleのサービス、社内ツールの多くがPythonで実装され、大規模なシステムで活用されています。代表的なサービスとして、YouTubeの大部分がPythonで実装されています。また2008年4月にGoogleがリリースしたクラウドサービスである**Google App Engine**の言語としてPythonが採用されており、Djangoも同梱されています（Google App Engineに同梱されているバージョンは0.96ですが、Google App Engine上で全ての機能が使用できるわけではありません）。

Googleのような世界的レベルの企業が、Pythonを積極的に活用しているのも、Pythonがエンタープライズのシステムで用いるのに効果的な言語であることを認められているからではないでしょうか。欧米ではGoogleをはじめとしてPythonの良さが既に認められ、さまざまなシステムやサービスで使用されています。

日本国内でのPython人気

Python Hack-a-thonの第1回の開催が、8月に予定されています。75名募集のところが、募集開始と同時にあっという間に定員に達し、キャンセル待ちが20人以上発生している状態です。このようなことからも、日本国内でのPython人気が高まってきていることが伺えるのではないでしょうか。

Hack-a-thon（ハッカソン）とは、ある開発テーマの技術に興味のあるプログラマーたちが、ノートPC持参で集まり、ソフトウェアをハックしたり、プログラムをコーディングしたりして、最後に開発したアプリケーションやサービスをプレゼンするというイベントです。日本でもここ最近で活発に開催されるようになってきています。

Djangoの紹介

いよいよ本稿の主役であるDjangoについて紹介します。Djangoはもともとは、アメリカの新聞社の社内ツールとして2003年頃に開発され、2005年7月にBSDライセンスのもとで、オープンソースとして公開されました。

Djangoのコアの部分は、データモデルとリレーショナルデータベース（以下、RDB）との間を仲介するO/Rマッパー、正規表現を活用したURLディスパッチャ、リクエスト要求を処理するビュー、データを表示するテンプレートシステムなどから構成され、Webアプリケーションフレームワークとして、フルスタックの機能を搭載しています。

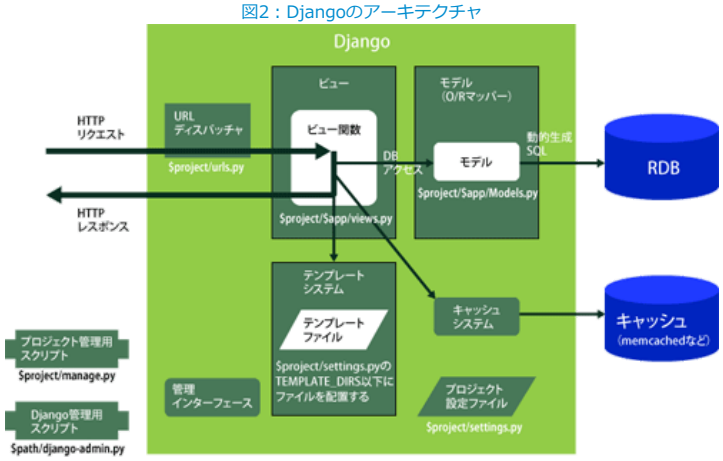
「The Web framework for pefectionists with deadlines（デッドラインにいる完璧主義者のためのウェブフレームワーク）」とDjango本家サイトのトップタイトルにも記載されているように、忙しい時でも、素早く簡潔できれいなアプリケーションが構築できることを目的にしています。

Djangoの本家サイトにも多数の実績が掲載されており、豊富な稼働実績を持っているフレームワークです。

Djangoのアーキテクチャ全体像

まずは、Djangoの全体像について説明します。Djangoのアーキテクチャを図2に示しました。

図2：Djangoのアーキテクチャ



MTVフレームワーク

図2にコントローラがないことに違和感を感じられた方もいらっしゃると思います。Djangoプロジェクトでは、MVC（Model View Controller）のControllerを「View」、Viewを「Template」と呼び、MTV（Model View Template）フレームワークと称しています。これは、Djangoプロジェクトでは「どのデータを提示するか」をViewの役割と解釈し「どう見せるか」をTemplateの役割と解釈しているためです。MTVとMVCで呼び名は違いますが、全体動作の流れとしては大きな変わりはなく、解釈の違いだけであるので、あまり気にするところではないと思われます。

モデル（Model）

モデルはDjangoで用意されているO/R（Object/Relational Database Mapper）マッパーが担当します。O/RマッパーはPythonクラスとしてモデルを定義するだけで、シンプルで使い勝手の良いデータベースへのインターフェイスを使用できるようにします。

対話型インターフェイスが用意されており、O/Rマッパーの動作を逐次確認できるので、すばやく使い方を習得することができます。またDjangoのモデルは、電子メール型や電話番号型など、RDBのフィールド型だけでは表現できないフィールドも用意しているので、より実際のシステム仕様に近づいたモデルを表現することができます。Google App EngineのEntityモデルもDjangoのモデルに近いものを採用しています。

テンプレート（Template）

Djangoのテンプレートシステムは、デザインとコンテンツ、Pythonコードの分離を目指したテンプレートエンジンです。テンプレート記述が複雑になることを防ぐため、あえてPythonとは別のテンプレート専用言語を採用しています。

ビュー（View）

ビュー関数は、HTTPリクエストを引数にとり、HTTPレスポンスを返す関数です。ユーザーに「どのデータを提示するか」を決定するロジックを記述します。

コントローラ（Controller）

Controllerは、URLのパスとそれに割り当てられ呼び出されるビュー関数のマッピングを記述するURLディスパッチ機構が担当します。このURLディスパッチ機構は正規表現を活用していて、フレームワークの仕様にしばられない柔軟なURL設計を実現できることが特徴です。

その他のDjangoの特徴的な機能

MVTフレームワークの機能以外にもDjangoは有用な機能を搭載しています。それらの特徴的な機能についていくつか説明します。

キャッシュシステム

Djangoは、memcachedや、その他のキャッシュフレームワークにデータを手軽にキャッシュできる仕組みを用意しています。この機構を利用することにより不特定多数のアクセスに耐えうるシステムを簡単に構築することができます。

国際化

Djangoは、多言語のアプリケーションに対応しています。設定ファイルに言語とタイムゾーンを指定します。

管理インターフェイス

モデル定義をもとにそのモデルのCRUD（Create、Read、Update、Delete）用の管理画面を提供する機能です。

Webサービスでは一般に、そのサービスで提供される表向きの機能とは別に、データを管理するためのバックエンド（バックオフィス）用の画面が必要となります。これらの画面はほとんどの場合、決まり切った定型的な画面となりますが、実運用で使えるものを作るとなると意外に手間がかかります。Djangoはこのバックエンド用画面を簡単に提供することを目的としています。

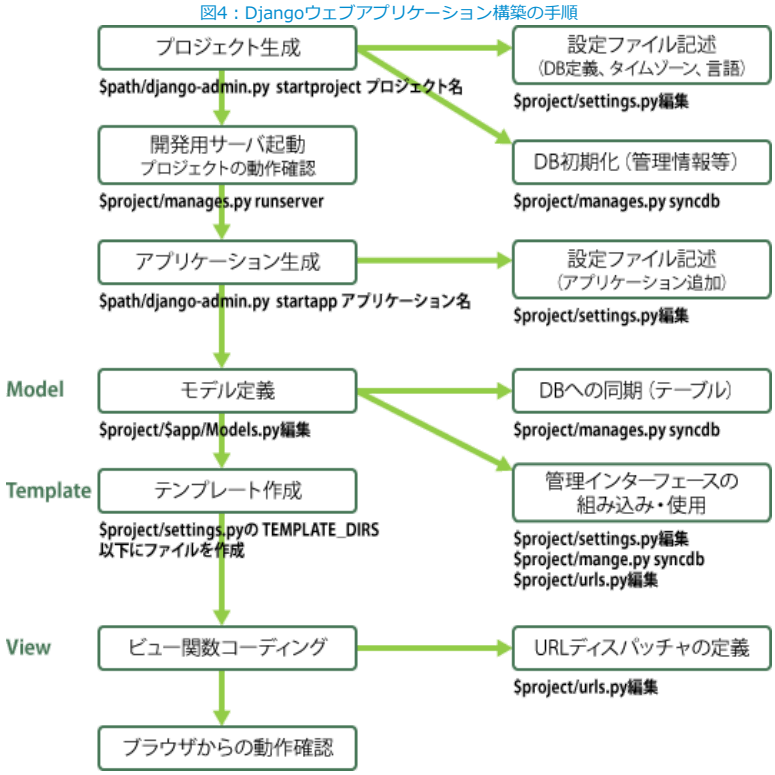
管理インターフェイスはを実運用でも十分に使用できるレベルのものであり、カスタマイズも容易です。Ruby on Railsのscaffoldとは違い、ソースコードを自動生成せず、モデル定義から動的に画面を生成するのも特徴の1つです。図3が管理インターフェイスのログイン後トップメニュー画面イメージです。

図3：管理インターフェイスのログイン後トップメニュー画面

Djangoによるアプリケーション開発の基本的な手順

図4に本稿で紹介する開発手順を示しました。以降では、この手順に従って説明を進めていきます。

図4：Djangoウェブアプリケーション構築の手順



インストール

それでは前準備として、PythonとDjangoをインストールしてみましょう。

Pythonのインストール

Python の公式サイトの**ダウンロードページ**から、該当プラットフォームのバイナリをダウンロードします。バージョンは、Djangoがサポートしている2.6.2（執筆現在）をダウンロード（Python 2.6.2 Windows installer）します。ダウンロードした「Python-2.6.2.msi」をダブルクリックすると、インストーラが開始しますので、以降はインストーラに従いインストールを進めます。インストールフォルダは「C:¥Python26」としました。インストール完了後、環境変数のPATHに以下を追加設定してください。

［リスト1］ 環境変数PATHの設定

```
C:¥Python26;C:¥Python26¥Tools¥Scripts
```

Djangoのインストール

次にDjangoをインストールします。**Djangoの公式サイトのダウンロードページ**からパッケージングされたファイルをダウンロードします。ダウンロードしたファイルを任意のフォルダに展開します（著者は「C:¥temp」以下に展開しました）。DOSプロンプトで、展開したフォルダの直下に移動し、以下のコマンドを実行します。

［リスト2］ Djangoのインストール

```
python setup.py install
```

以下の場所に、Djangoのフォルダとファイルがコピーされます。

- C:¥Python26¥Lib¥site-packages
- C:¥Python26¥Scripts¥django-admin.py

LinuxやMacOSの場合もインストール方法もほぼ変わりません。Django本家サイトの**インストールガイド**やDjango和訳ドキュメントの**インストールガイド**などを参照してください。

Djangoのインストール確認

DOSプロンプトで、Pythonを実行し、Pythonの対話型インターフェイスから次のように入力して確認します。

［リスト3］ Djangoのインストール確認

```
> python
Python 2.6.2 (r262:71605, Apr 14 2009, 22:40:02) [MSC v.1500 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> import django
>>> django.VERSION
(1, 1, 0, 'beta', 1)
```

Djangoのバージョンが正しく表示されれば、インストールが正常に完了しています。

構築するアプリケーションの概要とチュートリアルの目標

PythonとDjangoのインストールが終わったところで、次にプロジェクトとアプリケーションの作成に進みます。本稿のチュートリアルでは、簡単なECサイトを題材とします。具体的には、ECサイトで販売する商品を中心とした情報を登録・表示するアプリケーションを構築することによって、Djangoの基本機能の使い方を習得することを目標とします。

プロジェクトの作成

次にDjangoのプロジェクトを作成します。任意のフォルダ（著者は「C:¥codezine」フォルダ）で、以下のコマンドを実行します。今回はECサイトを構築するので「ecsite」というプロジェクト名とします。

［リスト4］ Djangoプロジェクトの作成

```
> django-admin.py startproject ecsite
```

コマンド実行が完了すると「ecsite」というフォルダが作成され（「C:¥codezine¥ecsite」）フォルダ内には、以下の表に示した4つのファイルが作成されます。

初めに生成される4つのファイル

ファイル名	説明
__init__.py	このディレクトリが Python パッケージであることを Python に知らせるための空のファイル
manage.py	Django プロジェクトに対するさまざまな操作を行うための管理コマンド用ユーティリティ
settings.py	Django プロジェクトの設定ファイル。DB接続定義や言語/タイムゾーンなど基本的な設定を記述する
urls.py	Django プロジェクトの URL設計を定義する。正規表現を用いてURLとビューのマッピングを定義する。

開発サーバの起動・動作確認

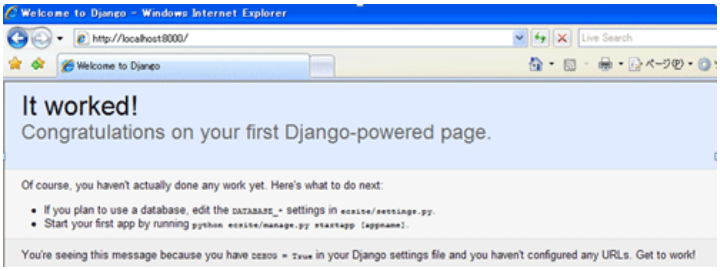
プロジェクトの作成を確認してみましょう。Djangoに同梱されている開発用スタンドアロンWebサーバを起動します。DOSプロンプトでプロジェクトのフォルダから（著者の場合「C:¥codezine¥ecsite」）以下のコマンドを実行します。

［リスト5］ 開発用サーバの起動

```
> manage.py runserver
```

ブラウザを立ち上げ「http://127.0.0.1:8000/」にアクセスしてください。以下の画面が表示されれば、プロジェクトの作成は成功しています。

図5：プロジェクト作成時トップ画面の確認



setting.pyの編集（SQLite接続設定）

次に、データベースの接続、言語、タイムゾーンの設定をします。プロジェクトフォルダ直下にあるsettings.py（著者環境の場合「C:¥codezine¥ecsite¥settings.py」）を次のように編集します。

［リスト6］ settings.py

```
DATABASE_ENGINE = 'sqlite3'     # 'postgresql_psycopg2', 'postgresql', 'mysql', 'sqlite3' or 'oracle'.
import os
DATABASE_NAME = os.path.dirname(__file__) + os.sep + 'ecsite.sqlite'
DATABASE_USER = ''              # Not used with sqlite3.
DATABASE_PASSWORD = ''          # Not used with sqlite3.
DATABASE_HOST = ''              # Set to empty string for localhost. Not used with sqlite3.
DATABASE_PORT = ''              # Set to empty string for default. Not used with sqlite3.

TIME_ZONE = 'Asia/Tokyo'
LANGUAGE_CODE = 'ja'
```

SQLite3を使用する場合、DATABASE_ENGINEに「sqlite3」、DATABASE_NAMEには、データベースファイル名を指定します。上記では、Pythonコードによって、動的にファイルパスを生成しています（著者環境の場合「C:¥codezine¥ecsite¥ecsite.sqlite」という名前でデータベースファイルが作成されます）。SQLite3を使用する場合、DATABASE_USER、DATABASE_PASSWORD、DATABASE_HOST、DATABASE_PORTの指定は必要ありません。タイムゾーン、言語はデフォルト指定では、それぞれ「America/Chicago」「en-us」になっていますが、それぞれ「Asia/Tokyo」「ja」に書き換えます。

プロジェクト管理情報のDBへの同期

settings.pyを編集して使用するDBの設定などが完了したところで、DBを初期化します。プロジェクトフォルダの直下で以下のコマンドを実行します。

［リスト7］ DjangoプロジェクトのDB初期化

```
> manage.py syncdb
```

実行すると、DB上にDjangoの管理用テーブルが作成されます。実行中のプロンプトでDjangoのスーパーユーザーを作るか否かを聞いてくるので「yes」と打ち込んでください。その後、管理者ユーザーIDとメールアドレス、パスワードも聞いてきますので入力します。このユーザーは、後に管理インターフェイスでログインする際に使用するユーザーなので忘れないようにしておいてください。

［リスト8］ DB初期化時のプロンプト表示

```
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: admin
E-mail address: user@example.com
Password:
Password (again):
Superuser created successfully.
Installing index for auth.Permission model
Installing index for auth.Message model
```

DBの確認（syncdb実行結果の確認）

データベースを初期化したところで、作成したDBの中身を確認しましょう。SQLiteのDBを参照するには、SQLite3の実行ファイルを取得する必要があります。SQLiteのサイトの**ダウンロードページ**で「sqlite-3.6.14.tar.gz」（執筆時の最新）を取得し、ファイルを解凍して「sqlite3.exe」を環境変数PATHが通っているフォルダにコピーしてください。その後プロジェクトフォルダに移動し、次のように実行します。

［リスト9］ SQLite3の実行

```
> sqlite3 ecsite.sqlite
```

上記のecsite.sqliteの個所には、DBファイル名を指定します。本稿ではsettings.pyの DATABASE_NAME パラメータに指定したファイル名です。

SQLite3 のプロンプトが立ち上がったら、以下のコマンドを打ち込んでください。Create Database文や作成したテーブルのCreateテーブル文を参照することができます。SQLiteのプロンプトではSQLを実行したり、テーブルを作成したり、データベースを直接操作することができます。

［リスト10］ SQLite3のダンプ

```
> .dump
```

Djangoでは「manage.py dbshell」とオプションを指定することでsettings.pyに設定したDBを直接操作できるのですが、Django1.1Betaの場合、Windowsではうまく動作しないのでSQLite3の実行ファイルを直接起動してデータベースファイルにアクセスするか、SQLite3用アクセス用のツールなどを使用する必要があります。筆者は、環境変数PATHで指定されたフォルダにSQLite3の実行ファイルを配置し「manage.py dbshell」を実行したのですが、以下のエラーが出力され、操作できませんでしたので本稿では直接sqlite3のコマンドを実行します（※2009年7月13日現在のレポジトリのメイントランクから取得した最新ソースの場合、正常にdbshellが実行できました。1.1の正式版ではバグは修正されるようです）。

［リスト11］ manage.pyからのDB接続

```
> manage.py dbshell
Error: You appear not to have the 'sqlite3' program installed or on your path.
```

アプリケーションの開発

プロジェクトを作成したら、次にアプリケーションを作成します。

アプリケーション作成コマンド実行

アプリケーションを作成するには、プロジェクトフォルダ（著者の場合「C:¥codezine¥ecsite」）で、以下のコマンドを実行します。ここでは商品ページを表示するためのアプリケーションなので、名前を「itempage」とします。

［リスト12］ アプリケーションの作成

```
> manage.py startapp itempage
```

コマンドの実行が完了すると、プロジェクトフォルダの直下に「itempage」というフォルダが作成され、以下の表に示した4つのファイルが作成されます。

アプリケーションを作成すると生成される4つのファイル

ファイル名	説明
__init__.py	このディレクトリが Python パッケージであることを Python に知らせるための空のファイル
models.py	モデル定義をするためのファイル
tests.py	UnitTestを記述するためのファイル
views.py	ビュー関数を定義するためのファイル

アプリケーションの追加設定

アプリケーションを作成したら、Djangoに認識させるためにプロジェクトフォルダの「settings.py」を編集し、 INSTALLED_APPS パラメータにアプリケーション名である「ecsite.itempage」を追加します。アプリケーション名は「プロジェクトサイト名.アプリケーション名」となります。名前はPythonのパッケージ名と一致します。

［リスト13］ settings.py

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

モデルの作成

アプリケーションを作成したら、いよいよモデルを作成します。モデルは基本的にデータベースの1テーブルと対応し、フィールドはDBテーブルの1カラムにマッピングします。

モデルのコーディング

ここでは商品のクラスをコーディングします。アプリケーションフォルダにある「models.py」を次のように編集します。ファイルはUTF-8形式で保存してください。

［リスト14］ models.pyの編集（Itemクラスのコーディング）

```
# -*- coding: utf-8 -*-
from django.db import models

# Create your models here.
class Item(models.Model):
    item_code = models.CharField(u'商品コード',max_length=256,unique=True)
    item_name = models.CharField(u'商品名',max_length=256)
    price = models.PositiveIntegerField(u'価格')
    start_date = models.DateField(u'掲載開始日',null=True);

    def __unicode__(self):
        return self.item_code

    class Meta:
        db_table = 'item'
```

モデルクラスは、 django.db.models.Model クラスを継承して作成します。ここでは、 Item クラスは、商品コード（ item_code ）、商品名（ item_name ）、価格（ price ）、掲載開始日（ start_date ）を属性として持つこととします。

各フィールドは型を表現するクラスによって定義され、そのコンストラクタにさまざまな制約を定義することができます。例えばitem_codeフィールドは文字列型なので、 CharField クラスによって定義され、コンストラクタに最大文字バイト数が256バイト（ max_length=256 ）、テーブル内でユニーク（ unique=True ）という制約を定義しています。

__unicode__ メソッドはインスタンスの文字列表現を返します。ここでは、商品コード（ item_code ）を返します。モデルに対応するテーブル名は、インナークラスMetaの db_table フィールドに名前を指定します。このフィールドを定義しなくても、「アプリケーション名_モデル名」というのがデフォルトのテーブル名（本稿の場合「itempage_item」）になりますが、アプリケーション名が入らないテーブル名の方が一般的と思われますので、ここではテーブル名を明示的に指定します。

モデルとDBの同期

次に作成したモデルをDBに反映します。プロジェクトフォルダにて以下のコマンドを実行します。

［リスト15］ モデルとDBの同期

```
> manage.py syncdb
```

「Creating table item」と画面に表示されたら成功です。SQLite3 で早速テーブルを確認してみましょう。

［リスト16］ SQLite3への同期確認

```
> sqlite3 ecsite.sqlite
SQLite version 3.6.13
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> .dump

 (……省略……)
CREATE TABLE "item" (
    "id" integer NOT NULL PRIMARY KEY,
    "item_code" varchar(256) NOT NULL UNIQUE,
    "item_name" varchar(256) NOT NULL,
    "price" integer unsigned NOT NULL,
    "start_date" date
);
 (……省略……)
```

コマンドラインからモデルを操作する

モデルを作成し、DBと同期すると、Djangoのユーティリティを通じてPythonのシェルからモデルを操作できるようになります。プロジェクトフォルダにて以下のコマンドを実行します。

［リスト17］ コマンドラインからのモデル操作

```
> manage.py shell
```

起動したシェル上で次のように打ち込みます。

［リスト18］ コマンドラインからのitemモデル操作

```
>>> from itempage.models import Item
>>> item = Item()
>>> item.item_name='item1'
>>> item.price = 1000
>>> item.save()
```

SQLite3でDBを直接確認してみましょう。

［リスト19］ SQLite3上のデータ作成確認

```
> sqlite3 ecsite.sqlite
sqlite> select * from item;
1|item-00001|item1|1000|
```

1件のレコードが作成されているのが確認できました。

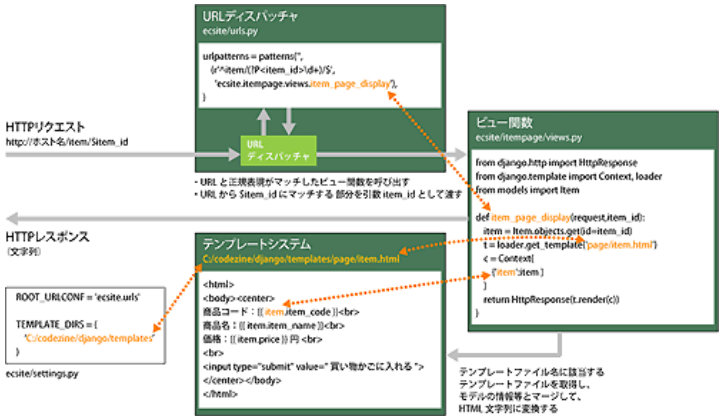
モデルをHTMLに表示する

データを作成したところで、DBからデータを取得しHTMLに表示する処理を作成します。

URLディスパッチャ、テンプレート、ビュー関数の関連

モデルを作成したところで、作成したモデルをHTMLに表示する機能を実装します。URLディスパッチャ、ビュー関数、テンプレートシステムの関係を図6に示しました。

図6：URLディスパッチャ・ビュー関数・テンプレートシステムの関係



図中に記載されている設定ファイルの記述、ソースコードについては順に説明していきます。

テンプレートの作成

まずは、モデルを表示するためのテンプレートを作成します。

settings.pyの編集

テンプレートファイルはDjangoのディレクトリ構造と切り離して設定することができます。「settings.py」ファイルの TEMPLATE_DIRS パラメータに、テンプレートを配置するルートフォルダを指定します。この値はカンマ区切りで複数指定することができます。ここでは「C:/codezine/django/templates」をテンプレート用のルートフォルダとして設定します。

［リスト20］ settings.py TEMPLATE_DIRSパラメータの編集

```
TEMPLATE_DIRS = (
    'C:/codezine/django/templates',
)
```

テンプレートファイルの作成

次に、テンプレートファイルを作成します。ファイルはsettings.pyの TEMPLATE_DIRS パラメータに指定した「C:/codezine/django/templates」の下にさらにpageフォルダを作成し「item.html」というファイル名（フルパス「C:/codezine/django/templates/page/item.html」）で作成します。ファイルの中身は次のとおりです。商品コード（item_code）、商品名（item_name）、価格（price）を表示するだけの簡単なページです。

［リスト21］ item.html

```
<html>
<head>
   <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
   <title>{{ item.item_name }}</title>
<head>
<body>
<center>
商品コード：{{ item.item_code }}<br>
商品名：{{ item.item_name }}<br>
価格：{{ item.price }}円<br>
<br>
<input type="submit" value="買い物かごに入れる">
</center>
</body>
</html>
```

テンプレートファイル内の「{{ item.item_name }}」と二重の中カッコになっている箇所に注目してください。ここはテンプレートがビュー関数で動的に変換される場所になります。ここでは商品コード、商品名、価格が表示時に動的に変換されます。

URLディスパッチャの定義

URLによるビュー関数のディスパッチを定義するには、プロジェクトフォルダ「c:¥codezine¥ecsite」の「urls.py」を編集します。

［リスト22］ urls.pyの編集

```
urlpatterns = patterns('',
    (r'^item/(?P<item_id>¥d+)/$', 'ecsite.itempage.views.item_page_display'),
)
```

urlpatterns 変数内の patterns 関数の第1引数に正規表現でパスを定義し、第2引数に対応するビュー関数の名前を指定します。「r'^item/(?P<item_id>¥d+)/$'」は、「item/数字/行末コード」を表します。数字の部分の「(?P<item_id>¥d+)」は、マッチした部分をitem_idという名前の引数でビュー関数に渡すことを意味します。第2引数の「'ecsite.itempage.views.item_page_display'」は「ecsite.itempage.views」パッケージのビュー関数「 item_page_display 」を呼ぶことを意味します。ビュー関数「 item_page_display 」の実際のコードは以降で説明します。

ビュー関数の定義

最後にビュー関数をコーディングします。アプリケーションフォルダのviews.pyに item_page_display 関数を定義します。

item_page_display 関数の引数は、HTTPリクエストを表す HttpRequest オブジェクトと、URLで指定された item_id の2つです。 item_id 引数は、URLディスパッチャの定義で説明したようにURLで指定された item_id が渡されます。ビュー関数では、引数 item_id に該当する Item オブジェクトをO/Rマッパーの APIを取得し、テンプレートとマージして、レスポンスを表す HttpResponse オブジェクトを返します。HttpResponseのコンストラクタには、画面に表示するHTMLの文字列を設定します。

［リスト23］ views.py

```
from django.http import HttpResponse
from django.template import Context, loader
from models import Item

def item_page_display(request,item_id):
    # item_idに該当するオブジェクトを取得する
    item = Item.objects.get(id=item_id)
    # テンプレートを取得して、モデルの値とマージする
    t = loader.get_template('page/item.html')
    c = Context(
        {'item':item }
    )
    # HTTP Responseを返す。
    return HttpResponse(t.render(c))
```

item_page_display 関数内で使われている処理の簡単な説明を以下の表に示します。

キャプション

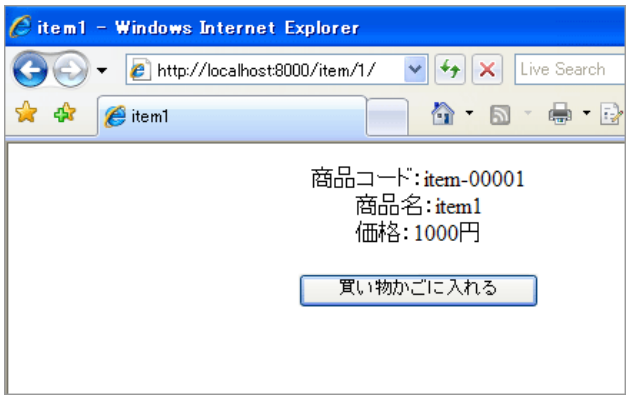
処理	説明

Item.objects.get(id=item_id)	Modelクラスのobjects属性は、オブジェクトマネージャと呼ばれ、DBから指定した条件に合ったオブジェクトや集合を取得するメソッドをもっています。getメソッドは、引数に指定された検索条件に合致する単一のオブジェクトを取得します。
loader.get_template('page/item.html')	loaderクラスのget_templateメソッドは引数で指定されたテンプレートファイルを表したインスタンスを返します。テンプレートファイルのパスは、settings.pyファイルの「TEMPLATE_DIRS」に設定したフォルダからの相対パスを指定します。
Context({'item':item })	Contextクラスには、テンプレートに渡すデータを辞書形式で指定します。ここでは「item」という名前でItemクラスのインスタンスを設定しています。
t.render(c)	Templateクラスのrenderメソッドは引数にContextクラスのインスタンスを受取り、テンプレート文字列とContextの情報をマージして文字列として返します。

商品ページの表示

商品画面表示のためのそれぞれの設定、ファイル作成がひととおり終わったところでブラウザから「http://localhost:8000/item/1/」にアクセスしてみましょう。図7のような画面が表示されれば成功です。

図7：商品画面の表示



まとめ

Django の最初のステップ（モデル作成、HTML画面の表示）について駆け足で説明してきましたが、いかがでしたでしょうか。Djangoは、簡単に簡潔できれいなアプリケーションを書くための仕組みが数多く用意されていて、快適にアプリケーションを開発することができます。前編のチュートリアルだけでも、ひととおり動くWebアプリケーションが10分もあれば開発できることがご理解いただけたかのではないかと思います。後編では、モデルAPIや、その他Djangoが用意している便利な機能を使用して、アプリケーションを拡張していきます。