Python を Web 上で使うには

リリース *2.7.17*

Guido van Rossum
and the Python development team

12 月 24, 2019

Python Software Foundation
Email: docs@python.org

目 次

著者 Marek Kubica

概要

このドキュメントは Python を web 向けに使う方法を示します。Python を web サーバとともに利用するためのいくつかの方法や web サイトを開発する際の一般的なプラクティスを示します。

web サイトのコンテンツをユーザが作るということに焦点を当てた ”Web 2.0” の提起以来 Web プログラミングは人気の話題となっています。Python を使って web サイトを作れましたが、それはやや退屈な作業でした。そのため、開発者がサイトを速く、頑強に作るのを助けるために多くのフレームワークと補助ツー

ルが作成されました。この HOWTO では動的なコンテンツを作成するために Python と web サーバを結合させるいくつかの方法について述べます。この話題は非常に広過ぎてドキュメント 1 つだけでは網羅しきれないため、この HOWTO を完全な入門書とするつもりはありません。ここでは最も人気のあるライブラリの簡単な概要を述べます。

参考:

この HOWTO は Python を web 上で使う方法の概要を扱おうとしていますが、HOWTO が常に望みどおりに最新の状況を伝えられるわけではありません。Python での web 開発は急速に発展しています、そのため Wiki ページ Web Programming に多くの最新の開発に関する内容があるでしょう。

第1章　低レベルから見て

ユーザが web サイトを訪れた時、ブラウザはサイトの web サーバとコネクションを形成します (これは **リクエスト** と呼ばれます)。サーバはファイルシステム上からファイルを探し出し、ユーザのブラウザにそれを送り返し、ブラウザが表示します (これが **レスポンス** です)。これが基礎となるプロトロル、HTTP のおおまかな動作です。

さて、動的な web サイトはファイルシステム上のファイルではなく、リクエストが来たときに web サーバが実行するプログラムの上に作られていて、このプログラムがユーザへ返すコンテンツを **生成** します。それらは掲示板の投稿を表示したり、メールを表示したり、ソフトウェアの設定や現在時刻の表示などの様々な便利なことができます。これらのプログラムはサーバがサポートするあらゆる言語で書くことができます。ほとんどのサーバが Python をサポートしているので、動的な web サイトを作成するのに Python を利用することは簡単です。

ほとんどの HTTP サーバは C や C++ で書かれていて、これらは Python コードを直接は実行できず、サーバとプログラムの間にブリッジが必要です。これらのブリッジやインターフェースはプログラムがサーバとどうやりとりするかを定めます。これまでに最良のインターフェースを作成するために膨大な数の試みがなされてきましたが、触れる価値のあるものはわずかです。

全ての web サーバが全てのインターフェースをサポートしているわけではありません。多くの web サーバは古い、現在では撤廃されたインターフェースのみをサポートしています; しかし、多くの場合にはサードパーティーモジュールを利用して新しいインターフェースをサポートするように拡張できます。

1.1 Common Gateway Interface

一般に ”CGI” と呼ばれるこのインターフェースは最も古く、ほとんどの web サーバでサポートされ、すぐに使うことができます。CGI を利用して web サーバと通信するプログラムはリクエスト毎に起動される必要があります。そのため毎回のリクエストは新しい Python インタプリタを起動します – このため起動にいくらか時間がかかります – そしてこのインターフェースは負荷が低い状況にのみ向いています。

CGI の利点は単純だということです – CGI を利用するプログラムを書くのは 3 行のコードを書くだけです。しかし、この単純さは後で高くつきます; 開発者を少ししか助けてくれません。

CGI プログラムを書くことは可能ではありますが、もはや推奨されません。*WSGI* (詳しくは後で述べます) では CGI をエミュレートするプログラムを書くことができるので、よりよい選択肢が選べない場合には CGI としてプログラムを実行できます。

参考:

Python の標準ライブラリには簡素な CGI プログラムを作成するのを助けるいくつかのモジュールが含まれています:

- `cgi` – CGI スクリプトでのユーザ入力を扱います

- `cgitb` – CGI アプリケーションの中でエラーが発生した場合に、 ”500 Internal Server Error” メッセージの代わりに親切なトレースバックを表示します

Python wiki では CGI scripts にあるページに Python での CGI に関して追加情報を取り上げています。

1.1.1 CGI をテストするための単純なスクリプト

CGI が web サーバで動くかどうかを調べるのに、この短く単純な CGI プログラムが利用できます:

```
#!/usr/bin/env python
# -*- coding: UTF-8 -*-

# enable debugging
import cgitb
cgitb.enable()

print "Content-Type: text/plain;charset=utf-8"
print

print "Hello World!"
```

web サーバの設定に依存して、このコードを `.py` もしくは `.cgi` 拡張子をつけたファイルに書く必要があります。それに加えて、セキュリティ上の理由のため、ファイルは `cgi-bin` フォルダ内に置く必要があるかもしれません。

`cgitb` 行が何なのか疑問に思うかもしれません。この行は、クラッシュしてブラウザで ”Internal Server Error” と表示する代わりに、親切なトレースバックを表示できるようにします。これはデバッグの際に便利ですが、いくつかの機密データをユーザにさらけ出すリスクにもなりえます。そういった理由で、スクリプトを完成品として利用する準備ができたら `cgitb` は使ってはいけません。さらに、**常に** 例外を捕捉し、適切なエラーページを表示するようにしなければいけません – エンドユーザは得体の知れない ”Internal Server Errors” をブラウザで見ることを好みません。

1.1.2 自身のサーバで CGI を立ち上げる

自身の web サーバを持っていない場合には、この内容は当てはまりません。そのままで動作するか調べることは可能で、もし動作しない場合は、とにかく web サーバの管理者と話し合う必要があります。大きなホストである場合、チケットに記入して Python サポートを求められるでしょう。

あなた自身が管理者であるか、自身のコンピュータで試すためにインストールしたい場合には自分自身で設定する必要があります。異なる設定オプションを持つ web サーバがたくさんあるため、CGI の設定法はひとつではありません。現在最も広く使われている web サーバは Apache HTTPd 、略して Apache です。Apache はほぼ全てのシステムにパッケージ管理ツールを使って簡単にインストールできます。lighttpd はもう一つの選択肢で、パフォーマンスがより優れているといわれています。多くのシステムでこのサーバはパッケージ管理ツールを利用してインストールできるので、web サーバを手動でコンパイルする必要はないでしょう。

- Apache ではチュートリアル Dynamic Content with CGI を参照でき、これには全てが解説されています。ほとんど場合には `+ExecCGI` を設定すれば十分です。このチュートリアルはよくでくわす可能性のある落し穴についても書かれています。

- lighttpd では CGI module を使う必要があり、この設定は分かりやすいです。結局のところ、`cgi.assign` を適切に設定することになります。

1.1.3 CGI スクリプトでの一般的な問題

CGI を利用していると、スクリプトを走らせようとするときにちょっといらいらすることがときどきあります。あるときは一見正しいスクリプトが期待どおりに動かないことがあり、気付くのが難しい小さな隠れた問題が原因だったりします。

いくつかの潜在的な問題は次の通りです:

- Python スクリプトが実行可能でない。CGI スクリプトが実行可能でないとき、多くの web サーバは実行しユーザに出力を送る代わりに、ユーザがダウンロードできるようにします。CGI スクリプトが Unix 系 OS で適切に実行されるために +x ビットが設定される必要があります。`chmod a+x your_script.py` を使うことで問題は解決するでしょう。
- Unix 系システムでは、プログラムファイルの行末は Unix 形式でなければなりません。web サーバはスクリプト最初の行 (shebang と呼ばれます) を調べ、そこで指定されたプログラムを実行しようとするため、ここが重要なのです。これは Windows の行末 (Carriage Return & Line Feed、または CRLF) によって簡単に混乱させられるので、ファイルの行末を Unix の行末 (Line Feed, LF のみ) に変換する必要があります。これは FTP 経由でバイナリモードではなくテキストモードでファイルをアップロードすると自動的に行なわれますが、単に Unix 行末で保存するようテキストエディタに指示する方が望ましいです。ほとんどのエディタはこの機能をサポートしています。
- web サーバはファイルが読めなければならないので、パーミッションが適切になっているか確認する必要があります。Unix 系システムでは、サーバはしばしば `www-data` ユーザ、`www-data` グループのパーミッションで実行されているので、ファイルの所有権を変更したり、`chmod a+r your_script.py` を使いファイルを誰からでも読み込み可能にしたりするのは試す価値があります。
- web サーバはアクセスを試みているファイルが CGI スクリプトであるということを知っていなければなりません。web サーバが CGI スクリプトに対し、ある特定のファイル拡張子を持つことを要求するよう設定されていないか設定を確認して下さい。
- Unix 系システムでは、 shebang にあるインタプリタへのパス (`#!/usr/bin/env python`) は正確でなければなりません。この行は Python を見つけるために `/usr/bin/env` を呼び出しますが、`/usr/bin/env` が無いか Python が web サーバのパスに無い場合は失敗します。Python がどこにインストールされているか分かっている場合、そのフルパスを使うこともできます。`whereis python` コマンドと `type -p python` コマンドも Python がどこにインストールされたかを探すのに役立つでしょう。一旦パスが分かれば、shebang 行をそれに応じて変更できます: `#!/usr/bin/python`。
- ファイルは BOM (Byte Order Mark) を含んでいてはいけません。BOM は UTF-16 と UTF-32 エンコーディングのバイト順を決定するのに利用されますが、あるエディタは UTF-8 ファイルにも BOM を書き込むことがあります。BOM は shebang 行に影響するので、エディタが BOM を書き込まないようにしてください。
- web サーバが *mod_python* を使っている場合、 `mod_python` が問題となることがあります。`mod_python` はそれ自身で CGI スクリプトを扱うことができますが、そのことが問題の原因となることがあります。

1.2 mod_python

PHP から来た人達はしばしば、Python を web 上で利用する方法を把握するのに苦労します。彼らが最初に考えるのはたいてい mod_python のことで、それは mod_python が mod_php と同等のものだと考えるからです。しかし実際にはたくさんの相違点があります。mod_python が行なうことは Apache プロセスへのインタプリタの埋め込みで、そのためリクエストごとに Python インタプリタを起動させる必要が無いので、リクエストに対する速度が向上します。一方で PHP でよくやるような「HTML への Python の埋め込み」とはかけ離れています。Python でそれと同等なことをするのはテンプレートエンジンです。mod_python 自身はより強力で Apache 内部に対してより多くのアクセスを提供します。CGI をエミュレートし、JSP に似た「HTML への Python 埋め込み」である "Python Server Pages" モードで動作でき、全てのリクエストを一つのファイルで受け付けて何を実行するを決める "Publisher" を持っています。

しかし、mod_python はいくつかの問題も抱えています。PHP インタプリタと違い Python インタプリタはファイル実行時にキャッシュを利用するため、ファイルの変更時にはアップデートするには web サーバ全体を再起動する必要があります。もう一つの問題は基本的なコンセプトにあります – Apache はリクエストを扱うためにいくつかの子プロセスを起動し、不幸にも全ての子プロセスが Python インタプリタ全体を、利用しない場合であっても、読み込む必要があるのです。このせいで web サーバ全体が遅くなります。別の問題は mod_python は特定のバージョンの libpython に対してリンクされるため、mod_python を再コンパイルせずに古いバージョンから新しいバージョンに切り替える (例えば 2.4 から 2.5) ことはできません。さらに mod_python は Apache web サーバとも結び付いているため mod_python 用に書かれたプログラムは他の web サーバで簡単に動かすことはできません。

これらが新しくプログラムを書く際に mod_python を避けるべき理由です。いくつかの状況では mod_python を利用するのはよいアイデアでしょうが、WSGI は mod_python 下でも WSGI プログラムを同様に動かせます。

1.3 FastCGI と SCGI

FastCGI と SCGI は CGI のパフォーマンス上の問題を別の方法で解決しようという試みです。web サーバにインタプリタを組み込む代わりに、バックグラウンドで長時間実行されるプロセスを生成します。さらに web サーバ上にはいくつかのモジュールがあり、それらは web サーバとバックグラウンドプロセスが「話す」ことを可能にします。バックグラウンドプロセスはサーバと独立しているため、 Python を含んだ、任意の言語で書くことができます。言語に必要なのは web サーバとの通信を扱うライブラリだけです。

FastCGI と SCGI の違いはささいなもので、SCGI は基本的に "simpler FastCGI" です。しかし、SCGI をサポートする web サーバは限定されているため、多くの人々は代わりに同様に動作する FastCGI を利用します。SCGI に適用されるほとんど全てのものは FastCGI にも適用できるので、 FastCGI だけについて解説します。

最近では FastCGI を直接使うことはありません。mod_python のように WSGI アプリケーションの配置のためだけに FastCGI が利用されています。

1.3.1 FastCGI のセットアップ

web サーバに応じて特別なモジュールが必要となります。

- Apache には mod_fastcgi と mod_fcgid の両方があります。mod_fastcgi が最初に作られましたが、非フリーとして扱われるという、いくつかのライセンスの問題があります。mod_fcgid はより小さく、前者と互換性があります。このうちのどちらかを Apache から読み込む必要があります。
- lighttpd は自身に FastCGI module を含んでいて、SCGI module も同様に含んでいます。
- nginx も FastCGI をサポートしています。

一旦モジュールをインストールして設定したら、以下の WSGI アプリケーションを使ってテストできます:

```
#!/usr/bin/env python
# -*- coding: UTF-8 -*-

from cgi import escape
import sys, os
from flup.server.fcgi import WSGIServer

def app(environ, start_response):
    start_response('200 OK', [('Content-Type', 'text/html')])

    yield '<h1>FastCGI Environment</h1>'
    yield '<table>'
    for k, v in sorted(environ.items()):
        yield '<tr><th>%s</th><td>%s</td></tr>' % (escape(k), escape(v))
    yield '</table>'

WSGIServer(app).run()
```

This is a simple WSGI application, but you need to install flup first, as flup handles the low level FastCGI access.

参考:

setting up Django with WSGI にドキュメントがあり、その多くは WSGI 互換フレームワークやライブラリで再利用できます。manage.py の部分を変更するだけで、ここで使った例を利用できます。Django もほとんど同様のことを行います。

1.4 mod_wsgi

mod_wsgi は低レベルなゲートウェイから脱するための試みです。FastCGI、SCGI、mod_python は主に WSGI アプリケーションを配置するために使われると仮定し、mod_wsgi は WSGI アプリケーションを直接 Apache web サーバに埋め込むために開発が開始されました。mod_wsgi は WSGI アプリケーションをホストするために特別に設計されています。これにより WSGI アプリケーションの配置は、グルーコードが必要な他の低レベルな手法を使った配置よりも非常に簡単になります。mod_wsgi の欠点は Apache web サーバに制限されてしまうことです; 他のサーバではそれ用の mod_wsgi の実装が必要になるでしょう。

mod_wsgi は 2 つのモードをサポートします: 埋め込みモード (embeded mode) は Apache プロセスとデーモンプロセスを統合するもので、 FastCGI に似ています。FastCGI と違って、 mod_wsgi はそれ自身がワーカープロセスを取り扱うので管理が楽になります。

第2章　後ろに下って: WSGI

WSGI について何度も言及してきたため、なにか重要そうに感じたでしょう。実際に重要なので、ここで説明します。

Web Server Gateway Interface 、略して WSGI は **PEP 333** で定義されていて、現在のところ Python で web プログラミングをする最良の方法です。WSGI はフレームワークを書くプログラマにとっては優れている一方、普通の web プログラマが直接触れる必要の無いものです。web 開発のフレームワークを選ぶときに、WSGI をサポートしているもの選ぶことは素晴しい考えです。

WSGI の大きな利点は、アプリケーションプログラミングインターフェース (API) が統一されることです。プログラムが WSGI と互換性があれば – これは外部から見たときにフレームワークが WSGI をサポートしているということになります – そのプログラムは WSGI ラッパーのある全ての web サーバインターフェースを通して利用できます。ユーザが mod_python や FastCGI や mod_wsgi のうちどれを利用しているかを気にせずにすみます – WSGI を使うことで任意ゲートウェイインターフェース上で動作するようになります。Python 標準ライブラリには、テストのために利用できる小さな web サーバである、独自の WSGI サーバ `wsgiref` が含まれています。

WSGI の本当に卓越している機能はミドルウェアです。ミドルウェアとはプログラムに様々な機能性を加えられるレイヤーのことを指します。無数の ミドルウェア が利用可能になっています。例えば自前でセッション管理を書く (HTTP はステートレスなプロトコルなので、複数の HTTP リクエストを単一のユーザに関連付けるために、アプリケーションがセッションによって状態を作成し管理しなければなりません) 代わりに、セッション管理をしてくれるミドルウェアをダウンロードし、動作させ、アプリケーション独自の部分をコーディングするだけです。圧縮についても同じです – HTML の gzip 圧縮をする既存のミドルウェアがあって、これでサーバの帯域を節約できます。認証も既存のミドルウェアで簡単に解決できる問題です。

WSGI は複雑に思えるかもしれませんが、学習の始めの段階は非常に価値ものになります。というのも WSGI とそれに関連したミドルウェアには、 web サイトを開発している中で起きる多くの問題の解決策が備わっています。

2.1 WSGI サーバ

CGI や mod_python などの様々な低レベルゲートウェイに接続するためのコードを *WSGI* **サーバ** と呼びます。こういったサーバの一つに `flup` があり、これは FastCGI、SCGI に加えて AJP もサポートしています。これらのサーバのいくつかは `flup` のように Python で書かれていますが、C で書かれたものもあり、それらは気軽に置き換えることができます。

多くのサーバが既に利用可能なので、 Python の web アプリケーションはほとんどどこにでも配置できます。これは他の web テクノロジーと比べたときの Python の大きな利点の 1 つです。

参考:

WSGI homepage では WSGI 関係のコードの素晴しい概要が読め、 WSGI をサポートする **全ての** アプリケーションが利用できる WSGI サーバ の広大なリストがあります。

標準ライブラリに含まれる WSGI をサポートするモジュールに興味が湧いたかもしれません、すなわちこのモジュールのことです:

- `wsgiref` – WSGI のためのいくつかの小さなユーティリティとサーバ

2.2 事例研究: MoinMoin

WSGI は web アプリケーションプログラマに何をもたらしてくれるのでしょうか? WSGI を使わずに Python で書かれた昔からある web アプリケーションをみてみましょう。

最も広く使われている wiki ソフトウェアパッケージの一つに MoinMoin があります。これは 2000 年に作られたため、WSGI より 3 年ほど先行していました。古いバージョンでは CGI、mod_python、FastCGI、スタンドアロンで動作するためには別々のコードが必要でした。

現在では WSGI をサポートしています。WSGI を使うことで、グルーコードを追加せずに WSGI に準拠したサーバに MoinMoin を配置できるようになりました。WSGI 対応前のバージョンとは違って、 MoinMoin の開発者が全く知らない WSGI サーバを MoinMoin に含めることができます。

第3章　モデル・ビュー・コントローラ (Model-View-Controller)

MVC という用語は「フレームワーク *foo* は MVC をサポートしています」というような文句でよく聞きます。MVC は特定の API というよりは、コードの総合的な構造についての話です。多くの web フレームワークはこのモデルを使い、開発者がプログラムに構造を与えることを助けています。大きな web アプリケーションはコード量が増えるので、最初からプログラムによい構造を持たせることはよい考えです。そうすることで、他のフレームワークのユーザであっても (MVC は Python 特有のものではないので、他の言語であっても)、既に MVC 構造に馴染んでいるため、コードを簡単に理解できます。

MVC は 3 つの構成要素からできています:

- **モデル** (model) 。これは表示したり変更されたりするデータのことです。Python のフレームワークでは、この要素は object-relational マッパーが利用するクラスで表現されます。
- **ビュー** (view) 。この構成要素の仕事は、モデルのデータをユーザに表示することです。典型的にはこの要素はテンプレートで実装されます。
- **コントローラ** (controller) 。これはユーザとモデルの間にあるレイヤーです。コントローラは (ある特定の URL を開くというような) ユーザの動作に反応し、必要に応じてモデルにデータを変更するよう伝え、ビューのコードに何を表示するかを伝えます。

MVC を複雑なデザインパターンだと考える人もいるかもしれませんが、実際はそうではありません。Python で使われているのは、それがきれいで保守可能な web サイトを作成するのに便利だということがわかっているからです。

注釈: 全ての Python フレームワークが明示的に MVC をサポートしているわけではありませんが、データロジック (モデル) とユーザとのインタラクションロジック (コントローラ) とテンプレート (ビュー) を分離して、 MVC パターンを使っている web サイトを作成することは普通のことです。その理由はテンプレートに不必要な Python コードを書かないことが重要なことだからです – そのようなコードは MVC に反する動作をして大混乱を生み、それを理解して修正することがいっそう難しくなります。

参考:

Wikipedia の英語版には Model-View-Controller pattern についての記事があります。この記事には様々なプログラミング言語での web フレームワークの長大なリストが載っています。

第4章　web サイトの要素

web サイトは複雑な構造物なので、 web 開発者がコードを書きやすく、保守しやすくする助けとなるツールが作られてきました。このようなツールは、全ての言語の全ての web フレームワークについて存在します。開発者はこれらのツールを強制的に使わされるわけではないし、「最良の」ツールはたいていは存在しません。しかし、これらの利用可能なツールは web サイトの開発プロセスを非常に単純にしてくれるので、学ぶ価値はあります。

参考:

このドキュメントで述べられるよりも多くの要素があります。Python wiki には Web Components と呼ばれる、これらの要素についてのページがあります。

4.1 テンプレート

HTML と Python コードを混在させることは、いくつかのライブラリを利用することで可能になります。最初は便利なのですが、こうしてしまうとコードが恐ろしく保守不可能となります。これがテンプレートが存在する理由です。テンプレートは、最も単純な場合には、単にプレースホルダーを持つ HTML ファイルとなります。プレースホルダーを埋めた後に HTML はユーザのブラウザに送信されます。

Python には既に単純なテンプレートを作る 2 つの手段があります:

```
>>> template = "<html><body><h1>Hello %s!</h1></body></html>"
>>> print template % "Reader"
<html><body><h1>Hello Reader!</h1></body></html>

>>> from string import Template
>>> template = Template("<html><body><h1>Hello ${name}</h1></body></html>")
>>> print template.substitute(dict(name='Dinsdale'))
<html><body><h1>Hello Dinsdale!</h1></body></html>
```

単純でないモデルのデータに基づいて複雑な HTML を生成するために、たいてい Python の *for* や *if* のような条件分岐や反復といった構造が必要とされます。**テンプレートエンジン** はこのような複雑さを持つテンプレートをサポートします。

Python には多くのテンプレートエンジンがあり、*framework* とともに、または独立に利用できます。いくつかのテンプレートエンジンでは、利用用途を狭めて学びやすくしてあるプレーンテキストのプログラミング言語を定義しています。他のテンプレートエンジンでは XML を利用しているものもあり、テンプレートの出力が常に有効な XML になるよう保証されています。これ以外にも非常に多くの種類があります。

独自のテンプレートエンジンを提供する *frameworks* もあれば、ある特定のテンプレートエンジンを推奨するものもあります。別のテンプレートエンジンを使う理由が無ければ、フレームワークが提供もしくは推奨するものを使うのがよい考えです。

人気のあるテンプレートエンジンには次のようなものがあります:

- Mako
- Genshi
- Jinja

参考:

たくさんのテンプレートエンジンが注目を集めようと競っています。というのも、 Python でテンプレートエンジンを作るのが非常に簡単だからです。wiki の Templating ページには、大量の今も増え続けるテンプレートエンジンの一覧が載っています。上で挙げた 3 つのテンプレートエンジンは "次世代の" テンプレートエンジンと見なされていて、始めるのに良いものです。

4.2 データの永続化

データの永続化 *(data persistence)* とは非常に複雑に聞こえますが、単にデータを保存するだけです。このデータはブログエントリのテキストだったり、掲示板の投稿だったり、wiki ページのテキストだったりします。もちろん、 web サーバに情報を保存するたくさんの様々な方法があります。

しばしば MySQL や PostgreSQL のような関係データベースエンジンが利用されます。それは数百万エントリに及ぶ非常に大きなデータベースを優れたパフォーマンスで扱うことができるためです。SQLite と呼ばれる小さなデータベースエンジンもあり、これは `sqlite3` モジュールによって Python にバンドルされていて、ファイルを一つだけしか使いません。これ以外の依存関係はありません。小さめのサイトに対しては SQLite で十分です。

関係データベースには SQL と呼ばれる言語を利用して **問い合わせ** *(query)* ます。一般的に Python プログラマは SQL があまり好きではなく、オブジェクトで動作する方を好みます。ORM (Object Relational Mapping) と呼ばれる技術を使うことで、 Python オブジェクトをデータベースに保存できます。ORM は内部でオブジェクト指向的アクセスを SQL コードに変換するので、ユーザはそのことを意識せずに済みます。ほとんどの *framework* は ORMs を利用し、とてもうまくいっています.

第 2 の可能性は通常のプレーンテキストのファイル (しばしば "フラットファイル" と呼ばれます) で保存することです。この方法は単純なサイトでは非常に簡単ですが、 web サイトが保存されたデータへの更新がたくさん行われる場合、ちゃんと動くようにするのは難しくなってきます。

第 3 の可能性はオブジェクト指向データベースです ("オブジェクトデータベース" とも呼ばれます)。このデータベースは、プログラムの実行中のメモリ上での構成とほぼ同じ形式でオブジェクトのデータを保存します。(対照的に、 ORM はオブジェクトデータをテーブルの行と行どうしの関係として保存します。) オブジェクトを直接保存することには、ほぼ全てのオブジェクトを単純な方法で保存できるという優位点があり、これは表現するのが非常に難しいオブジェクトがある関係データベースには無い特徴です。

framework はしばしばどのデータストレージ方法を選べばよいかのヒントを与えてくれます。代わりとなるストレージ機構を使った方がアプリケーションの特別な要件によりよく合うのでない限り、普通はフレームワークが推奨しているデータストアを使い続けるのがよい考えです。

参考:

- Persistence Tools にはファイルシステムにデータを保存する方法が列挙されています、これらのモジュールの内のいくつかは標準ライブラリの一部です
- Database Programming はデータ保存の方法を選ぶのを助けてくれます

- SQLAlchemy, the most powerful OR-Mapper for Python, and Elixir, which makes SQLAlchemy easier to use
- SQLObject は別の人気のある OR マッパーです
- ZODB と Durus の二つはオブジェクト指向データベースです

第5章　フレームワーク

web サイトを動かすコードを作成するプロセスは様々なサービスを提供するコードを書く作業を伴います。あるサービスを提供するコードは、当該の web サイトの複雑さや目的に関わらず、たいてい同じような動作をします。共通のソリューションを抽象化し再利用可能なコードにすると、 web 開発で ”フレームワーク” と呼ばれるものが生み出されます。おそらく最も知られている web 開発のフレームワークは Ruby on Rails ですが、 Python には独自のフレームワークがあります。その中には部分的に Rails にインスパイアされたり、アイディアを拝借したものもありましたが、多くのものは Rails よりずっと前から存在していました。

もともと Python の web フレームワークには、 web サイトを開発するのに必要な全てのサービスを 1 つにまとめて、巨大な集約されたツールの集まりを作る傾向がありました。どの 2 つの web フレームワークも互換性はありませんでした: あるフレームワーク用に開発したプログラムは、かなりの再設計作業をしないと他のフレームワークには配置できませんでした。この状況が ”ミニマリスト” web フレームワークの開発につながり、そういったフレームワークは Python コードと http プロトコルの間のやり取りをするツールだけを提供し、他の全てのサービスを別々のコンポーネントとしてフレームワーク上に追加するものでした。場当たり的な標準が開発され、制限はあるものの、異なるテンプレートエンジンを交換可能にする標準のような、フレームワークどうしの互換性も生まれました。

WSGI の出現から Python の web フレームワークの世界は、 WSGI 標準に基づいた互換性に向かって進化していきました。(最も複雑な web サイトを配置するのに必要な全てのツールを提供する) ”フルスタック” であれ、ミニマリストであれ、その中間であれ、今や多くの web フレームワークが複数のフレームワークで使える再利用可能なコンポーネントで構築されています。

大多数のユーザはおそらくコミュニティが活発な ”フルスタック” フレームワークを選びたいでしょう。これらのフレームワークはきちんとドキュメントが書かれ、機能を完全に揃えた web サイトを最小の時間で作る簡単な道が提供されている傾向があります。

5.1 いくつかの注目に値するフレームワーク

膨大な数のフレームワークがあるので、全てについて記述することは不可能です。その代わりに最も人気のあるもののいくつかに簡単に触れます。

5.1.1 Django

Django は、一から書かれた非常に上手く連携するいくつかの要素を固く結び付けて構成されたフレームワークです。Django には非常に強力でシンプルに使える ORM があり、ブラウザでデータベースのデータを編集できる優れたオンラインの管理インターフェースがあります。テンプレートエンジンはテキストベースで Python が書けないページデザイナーにも使えるよう設計されています。このテンプレートエンジン

はテンプレートの継承と (Unix のパイプのように動作する) フィルタをサポートしています。Django には RSS フィードや汎用ビュー (generic view) のようなたくさんの使いやすい機能が付属しているため、 Python コードをほとんど書かずに web サイトを作ることができるようになっています。

Django には、多くの web サイトを作成してきたメンバーによる、大きな国際的なコミュニティがあります。Django の通常の機能を拡張するたくさんのアドオンプロジェクトもあります。これらは Django の素晴しい オンラインドキュメント と Django book による部分もあります。

注釈: Django は MVC スタイルのフレームワークですが、 Django は構成要素に対して別の名前を付けています。これについては Django FAQ で解説されています。

5.1.2 TurboGears

他の人気がある Python のフレームワークは TurboGears です。TurboGears は、既存のコンポーネントをグルーコードで組み合わせ、無理なく連携する使い勝手を持たせるという手法をとっています。TurboGeers にはユーザがコンポーネントを選べる柔軟性があります。例えば、 ORM とテンプレートエンジンはデフォルトとは異なるパッケージに変えられます。

ドキュメントが TurboGears documentation にあり、スクリーンキャストへのリンクもあります。TurboGears にも活発なユーザコミュニティがあり、関連した質問のほとんどに答えてくれます。入門に向いた TurboGears book も出版されています。

TurboGears の最新版の version 2.0 では、よりいっそう WSGI サポートとコンポーネントに基づくアーキテクチャに向けて進んでいます。TurboGears 2 は Pylons というコンポーネントに基づく人気の web フレームワークの WSGI スタックを基礎としています。

5.1.3 Zope

Zope フレームワークは ”古い原始の” フレームワークの 1 つです。Zope2 の現在の姿は、固く結合されたフルスタックなフレームワークです。Zope の最も興味深い機能の 1 つは、本体に固く結合されている ZODB (Zope Object Database) と呼ばれる強力なオブジェクトデータベースです。この固く結合されている性質のために、 Zope はいくぶん孤立したエコシステムが形成されることとなりました: Zope 用に書かれたコードは Zope 以外の場所ではあまり使えず、逆向きも同様です。この問題を解決するために Zope 3 への取り組みが始まりました。Zope 3 では、より綺麗に分離されたコンポーネントの集合体として Zope を再設計します。この取り組みは WSGI 標準が考案されるより前に始まりましたが、 Repoze プロジェクトによって Zope 3 のための WSGI サポートが提供されています。Zope のコンポーネントはそれまで何年もプロダクションで使われていて、 Zope 3 プロジェクトではそれまでより広い Python コミュニティに向けて、それらのコンポーネントが利用できるようになります。Zope コンポーネントに基づく Zope とは別の Grok というフレームワークさえあります。

Zope は Plone によって利用されているインフラストラクチャでもあります。Plone は利用可能な中でもっとも強力で人気のあるコンテンツ管理システム (content management system, CMS) の一つです。

5.1.4 他の注目に値するフレームワーク

もちろんこれらの二つだけが利用できるフレームワークの全てではありません。ここで触れておく価値のあるフレームワークは他にもたくさんあります。

既に述べた別のフレームワークに Pylons がありました。Pylons は TurboGears とよく似ていますが、柔軟性に磨きがかかっていて、利用するコストが少々高くつきます。ほぼ全ての要素は交換することができるため、個々のコンポーネント全てについてドキュメントを利用する必要があり、そしてコンポーネントはたくさんあります。Pylons は Paste という WSGI を扱いやすい大量のツールの集合体に基づいて構築されています。

これで全てが述べられたわけではありません。最新の情報は Python wiki で見つけることができます。

参考:

Python wiki には広大な web フレームワーク のリストがあります。

ほとんどのフレームワークは自身のメーリングリストや IRC チャンネルを持っているので、プロジェクトの web サイトからそれらを探して下さい。

索引