Python を Web 上で使うには HOWTO

リリース *2.7ja1*

Guido van Rossum
Fred L. Drake, Jr., editor

2011 年 12 月 25 日

Python Software Foundation
Email: docs@python.org

目次

1 **低レベルから見て** ii
1.1 Common Gateway Interface . . . iii
1.2 mod_python . . . v
1.3 FastCGI と SCGI . . . vi
1.4 mod_wsgi . . . vii

2 **後ろに下って**: WSGI vii
2.1 WSGI サーバ . . . viii
2.2 事例研究: MoinMoin . . . viii

3 Model-view-controller ix

4 web **サイトの構成要素** ix
4.1 テンプレート . . . x
4.2 データの永続性 . . . xi

5 **フレームワーク** xii
5.1 いくつかの著名なフレームワーク . . . xii

Author Marek Kubica

概要

このドキュメントは Python を web 向けに組み込む方法を示します。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。web サイトを作るのに Python を利用することは可能でしたが、それは退屈な作業でした。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。この話題に対する一般的な入門はこのドキュメントだけで扱いきれないほど広いわけではありませんが、ここでは最も人気のあるライブラリの簡単な概要のみを扱います。

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ているとは限りません。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。

1 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です。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、そのため動的な 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 コードを単純な方法で実行できません– 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決めるために膨大な数の試みがなされてきましたが、言及すべきものはわずかです。

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

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 – そのためこのインターフェースは負荷が少な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ます、そのため、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 ページに Python での CGI に関した追加情報を取り上げています。

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、この短かく単純な 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!"
```

このコードは `.py` または `.cgi` 拡張子をつけたファイルに書く必要があります、どちらを使うかは web サーバの設定に依存します。web サーバの設定によっては、ファイルは `cgi-bin` フォルダ内にある必要があるかもしれません、これはセキュリティ上の理由のためです。

`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、いくつかの重要なデータをユーザにさらけ出すリスクにもなりえます。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– エンドユーザは漠然とした “Internal Server Errors” をブラウザで見ることを好みません。

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らないでしょ。そのままで動作するか調べることは可能です、もし動作しない場合、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、チケットを埋めて Python サポートを得るということができるでしょう。

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 です、Apache を簡単に説明すると – 最も多くの人が利用しているもので、ほとんど全てのシステムでシステムのパッケージ管理ソフトを利用して簡単にインストールできます。ただし lighttpd も注目を集め始めていて、さらにパフォーマンスの面で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ソフト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全くありません。

- Apache ではチュートリアル Dynamic Content with CGI を参照できます、これには全てが書かれています。多くの場合には `+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 を使う必要があります、これは直接的な方法で設定できます。結局のところ、`cgi.assign` を適切に設定することになります。

CGI スクリプトでの一般的な問題

CGI を利用しようとすると、しばしば小さないらいらを生むことになります、その中でおそらく経験するものの一つは実行時するためにこれらのスクリプトの取得を試みるときに起きます。これによって一見正しいスクリプトが期待どおりに動かないことはよくあります、これにはいくつかの小さな隠れた理由があるので、突き止めるのが困難です。

いくつかの理由:

- Python スクリプトが実行可能であると示されていない。CGI スクリプトが実行不可能な場合、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 スクリプトに対しては適切に実行するために `+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します。
- 行末は Unix 形式である必要があります。これは、web サーバが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の実行を試みるため重要となります。これは Windows の行末 (Carriage Return & Line Feed、または CRLF) とよく混同されます、そのためファイルを Unix 行末 (Line Feed のみ LF) に変換する必要があります。これは FTP 経由でテキストモードではなくバイナリ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、より望ましい方法は単にテキストエディタで Unix 行末で保存することです。多くの適切なエディタはこれをサポートしています。

- web サーバはファイルを読めるようにしなければいけないので、パーミッションが適切になっているか確認してください。しばしば、サーバは `www-data` ユーザ、グループとして実行します、そのためファイルのオーナーシップを変更するか、 `chmod a+r your_script.py` を使いファイルを全世界から読み込み可能にするといったこと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いけません。web サーバの設定を確認して下さい、おそらくそこに間違いがあります。

- shebang 内のインタプリタへのパス (`#!/usr/bin/env python`) は正確でないといけません。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` が無い場合失敗します。Python がどこにインストールされているか知っている場合、そのパスを使うこともできます。`whereis python` と `type -p python` なども 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。一旦わかってしまえば、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 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、しかしいくつかのエディタは UTF-8 ファイルに対してもそれ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 を書かないようにしてください。

- *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。利用できないようにして下さい。

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。彼らはたいてい最初に mod_python が `mod_php` と等価だと考えて、利用しようとします。しかし実際にはそうではありません。mod_python は Apache プロセスにインタプリタを埋め込まないため、毎回のリクエストで起動せずスピードアップします。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 でそれと等価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開始し、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。このため web サーバ全体が遅くなります。別の問題は mod_python は `libpython` が特定のバージョンに対してリンクされることです、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 サーバに制限されるため 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、それらは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、もちろん Python を含んだ、任意の言語で書くことができます。言語は web サーバとの通信を扱うライブラリを持っている必要があるだけです。

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 にも適用できます、そのため後者のみを書くことになるでしょう。

最近では FastCGI を直接呼ぶことはありません。`mod_python` のように WSGI アプリケーションのみが利用されてます。

参考:

- FastCGI, SCGI, and Apache: Background and Future には FastCGI と SCGI のコンセプトが mod_python よりどうして優れているかが検討されています。

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()
```

これは単純な WSGI アプリケーションですが、まず flup をインストールする必要があります、flup は低レベルの FastCGI アクセスを取り扱います。

参考:

setting up Django with FastCGI にドキュメントがあります、この内の多くは WSGI 互換フレームワークやライブラリで再利用できます。代わりに `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 低レベルなゲートウェイから脱するための試みです。WSGI アプリケーションをどうにかして利用可能にするには多くの場合 FastCGI、SCGI、mod_python が利用されるますが、mod_wsgi は WSGI アプリケーションを直接 Apache web サーバに埋め込みます。この方法をとることによる恩恵はホストの WSGI アプリケーションのために特別にデザインされたものと比べてより簡単な WSGI アプリケーションを利用できることです– ホストの WSGI アプリケーションに対するグルーコードを持つ他の低レベルな方法 (先ほど述べた flup のような) と違って。mod_wsgi の欠点は Apache web サーバに制限されているということです、他の web サーバは mod_wsgi の独自実装が必要となります。

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し、動作としては FastCGI に似ています。FastCGI と比較すると、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。

2 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, **PEP 333** または略して WSGI は現在のところ Python で web プログラミングをする最良の方法です。programmers writing framework として卓越している一方で、一般の人は直接接点を持つ必要はありません。しかし、web 開発フレームワークとして選択する場合に、WSGI を選ぶことは素晴しい考えです。

WSGI を使う上での大きな利益は、統一性です。WSGI と互換性のあるプログラムであれば– これはフレームワークが WSGI をサポートしているということを意味し、プログラムは WSGI ラッパーを持つ全ての web サーバインターフェースで利用可能になります。つまりユーザが mod_python か FastCGI どちら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るための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。例えばセッション管理 (連続したリクエストの中でユーザを同定を行います、HTTP は状態を維持しないので、リクエストが同じユーザのものであるということがわかります) を書く代わりに、ミドルウェアを取ってきて繋ぐだけで、既存の機能を当てにできます。圧縮についても同じです – HTML を gzip で圧縮してサーバの帯域を節約したい場合。ミドルウェアを組み込むだけで行うことができます。認証もミドルウェアで簡単に解決できる問題です。

一般的には – WSGI は複雑に思えるかもしれませんが、WSGI は web サイトを書く際に起きうる多くの問題の解決法を持っているため、学習の初期段階を実りあるものにしてくれます。

2.1 WSGI サーバ

CGI や mod_python などに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。これらのサーバの一つに `flup` があります、これは前に述べましたが FastCGI、SCGI と AJP を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ます、それらは気軽に置き換えることができます。

いくつものサーバが利用可能ですから、Python の web アプリケーションはほとんどどこでも利用できます。これは他の web テクニックと比べたときの Python の大きな利点です。

参考:

WSGI wiki は全ての WSGI に関連したコードに対して素晴しい概観を与えてくれます、ここには WSGI をサポートするアプリケーション **全て** が利用できるサーバの広大なリスト WSGI servers が含まれています。

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:

- `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? 長く存在している WSGI を使わない、Python で書かれた 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。現在では WSGI をサポートしていますが、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。いまやこれら全ては

WSGI と既存のゲートウェイを使って可能になりました。FastCGI 上では `flup` を使って実行できますし、スタンドアロンサーバとして実行するには `wsgiref` を使うことになります。

3 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 は技術的なものというよりかは、構造的なものです、多くの web フレームワークはこのモデルを使って開発者がプログラムに構造を持ちこむことを助けています。大きな web アプリケーションはたくさんのコードを持っているので、最初からプログラムに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 (他の言語でもかまいません、MVC は Python 特有のものではないので) のユーザであっても、構造に既に馴染んでいるため、存在しているコードを簡単に理解できるようになります。

MVC は三要素からできています:

- *model* これは変更されるデータのことです。たいていの Python のフレームワークでこの要素は object-relational マッパーを利用したクラスで表現されます。つまり宣言部分はここに集約されます。
- *view* この要素はモデルのデータをユーザに表示する仕事を行います。典型的にはこの要素はテンプレートで表現されます。
- *controller* これはユーザとモデルの間のレイヤーです。controller はユーザの動作 (特定の URL を開く等) に反応して、必要に応じてモデルにデータを変更するよう伝えます。

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。

ノート: 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、MVC パターンを使った、データロジック (model) とインタラクションロジック (controller) とテンプレート (view) を分離した web サイトを作成することはありふれています。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– MVC に反すると大混乱を生むことになります。

参考:

Wikipedia の英語版には Model-View-Controller pattern の記事があります、この記事には様々な言語での web フレームワークの長大なリストが含まれています。

4 web サイトの構成要素

web サイトは複雑な構造物です、そのため web サイト開発者の保守作業を助けるためにツールが作られました。それらのツールは Python 固有のものではないので、他のプログラミング言語に対しても存在します。もちろん、開発者はそれらのツールを使うことを強制されることはありませんし、たいていの場合において「最

良」のツールも存在しません、しかし、開発者が利用できる道具は膨大にあるので、どれかを選ぶ前に情報を得ておくことは価値のあることです、

参考:

ここで述べたものよりも多くの組み合わせ可能な要素が書かれています。Python wiki にはこれらの要素についてのページ Web Components 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。最初は便利ですが、そうすることでコードが保守不可能となる恐れがあります。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 はこのような単純なテンプレートを含んでいます:

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

Python の標準ライブラリには、 `string.Template` を通して利用できるより高度なテンプレートも含まれています、しかし、HTML テンプレートには Python の *for* や *if* のような条件やループなどが使えることが必要です。そのため、 **テンプレートエンジン** が必要になります。

いまや、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は平文のプログラミング言語を利用します、この言語はとても限定的なので簡単に学ぶことができます、一方で他のものには XML を利用し、テンプレートの出力が正当な XML であることが保証されているものがあります。いくつかの framework は独自のテンプレートエンジンを含んでいたり、特定のエンジンを使うことを推奨しています。もしどれがいいかわからない場合には、使ってみるといいでしょう。

ノート: Python はほんとうに様々なテンプレートエンジンを持っていますが、たいていの場合に手製のテンプレートシステムを使うことは理にかないません。全てのテンプレートシステムを評価するのは不毛ですから、最も人気のあるものを探すのに時間を割いた方がよいでしょう。いくつかのフレームワークは独自のテンプレートエンジンを持っていたり、推奨しているものがあります。これらを選ぶのが賢いでしょう。

人気のあるテンプレートエンジンに含まれるものは:

- Mako
- Genshi
- Jinja

参考:

たくさんの様々なテンプレートエンジンが、それぞれに注目を分けあっています、これは Python でのエンジン作成が簡単なためです。wiki の Templating には巨大な今も増え続けるエンジンが列挙されています。

4.2 データの永続性

データの永続性 *(data persistence)* は複雑に聞こえますが、単にデータを蓄積するだけです。このデータはブログのエントリであったり、掲示板の投稿であったり、wiki ページのテキストであったりします。例のごとく web サーバに情報をたくわえるには様々な方法があります。

しばしば MySQL や PostgreSQL のようなリレーショナル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。それらは SQL と呼ばれる言語を利用した **照会** *(queried)* です。一般的な Python プログラマは SQL をあまり好みません、彼らはオブジェクトで動作する方を好みます。ORM と呼ばれる技術を使うことでデータベース内の Python オブジェクトを節約できます。ORM は内部でオブジェクト指向的アクセスを SQL コードに変換し、ユーザはそのことを意識せずに済みます。多くの framework は ORMs を利用し、とてもうまく動作します.

第 2 の可能性はハードディスクに保存されたファイルを利用することです (しばしば、フラットファイルと呼ばれます)。これはとても簡単ですが、高速ではありません。SQLite と呼ばれる小さなデータベースエンジンもあります、これは Python に `sqlite` モジュールとしてバンドルされていて、ファイル一つだけを利用します。このデータベースは ORM 経由でオブジェクトを保存するのに利用できて、依存関係がありません。小さなサイトに対しては SQLite で十分です。ただ、データをファイルシステムに保存する方法はこれだけではありません。普通の平文のテキストファイルで十分な場合もあります。

第 3 の最も利用されない方法はオブジェクト指向データベースと呼ばれています。このデータベースは、OR マッピングによって行との間に作成されるリレーションの代わりに、**実際のオブジェクト** を保存します。この方法にはほとんど全てのオブジェクトを直接的な方法で保存できるという利点があります、これはリレーショナルデータベースでいくつかのオブジェクトが ORMs で表現するのが困難であるのと対照的です。

framework はしばしばユーザにどの方法を選べばいいか、ヒントを与えてくれます。たいていの場合、一つだけの方法を必要とする特別な条件がある場合がない限りは、それに従うことはいい考えです。

参考:

- Persistence Tools 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 は Python での最も強力な OR マッパーで、Elixir は利用しやすいものにしてくれます
- 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

5 フレームワーク

web サイトは簡単に大きくなりうるので、開発者がそれらのサイトを簡単に扱えるよう助けるものはフレームワークと呼ばれます。最も知られたフレームワークは Ruby on Rails ですが、Python も独自のフレームワークを持っていて、それらは Rails からヒントを得たものであったり、Rails より以前から存在していたものであったりします。

web フレームワークのアプローチには二つの方法があります: 最小主義アプローチと全てを含む包括的アプーチ (しばしば *full-stack* と呼ばれます) です。包括的なフレームワークは作業を始めるのに必要なもの全てが揃っています、テンプレートエンジン、いくつかのデータベースへの保存、アクセス方法やその他の機能。それらは他の多くユーザによって広く利用されていて、本やチュートリアルといった形式で詳細なドキュメントが準備されているので、多くのユーザはそれらを利用して順調に進めることができます、他の web フレームワークは最小主義アプローチをとり、できるだけ自由に変えられるようにして、ユーザがどれが最良かを選ぶ自由を残せるようにしています。

ユーザの多数は包括的フレームワークを利用することで順調に進めることができます。それらのフレームワークは全てを備えているので、ユーザは単に飛び乗ってコートを書くことができます。それらには制限がある一方で、申し分なくやりたいと思った内容の 80% は埋めてくれます。それらは多様な要素から成っていて、それぞれの要素ができるだけうまく協調できるようにデザインされています。

Python で書かれた web フレームワークが大多数存在することは、実際にそれらを書くことが容易だということを実証しています。Python で書かれた web アプリケーションで最も知られているものは Zope で、これは大きなフレームワークの一種とみなすことができます。しかし、いまではほとんど忘れられていますが、Zope は単なるフレームワークではありません、利用者の多くが新しいものに移行したため、それらについてこれ以上記述する必要はないでしょう。

5.1 いくつかの著名なフレームワーク

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し、その必要さえありません、なぜならほとんどのフレームワークは特別なものではありませんし、それらにできることは人気のあるものでできます。

Django

Django はスクラッチから書かれた、とてもうまく協調する、いくつかの要素が強く結びついてできたフレームワークです。ORM を含んでいてとても強力である上に、単純に利用でき、ブラウザからデータベース上のデータを編集できる優秀な管理インターフェースを持っています。テンプレートエンジンはテキストベースで動作し、Python を書けないページデザイナーにとっても利用しやすいようにデザインされています。テンプレート継承やフィルタ (Unix のパイプのように動作します) と呼ばれるものもサポートしています。Django は RSS フィードの作成や、ほぼ Python コード無しで web サイトを作ることができるようなジェネリックビューといった、多くの役に立つ機能をバンドルしています。

Django には、Django を利用して多くのサイトを作成してきた大きく、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アドオンプロジェクトもたくさんあります。この内容の一部は Django の素晴らしい online documentation と Django book によります。

ノート: Django は MVC スタイルフレームワークですが、Django は構成要素に対して異なる名前で読んでいます、このことは Django FAQ に詳しい記述があります。

TurboGears

Python の他に人気あるフレームワークは TurboGears です。これは既存の要素と継ぎ目ををなくすためグルーコードを使ってそれらを組み合わせるアプローチをとっています。TurboGears はユーザが自由に要素を選べるようにしていて、ORM は制限が多い代わり使いやすいもの、複雑だが強力なもの、に切り替えて使うことができます。テンプレートエンジンにも同様のことがいえます。TurboGears の強力な点は、構成要素が他のプロジェクトでも TurboGears への依存無しに簡単に利用できるということです、例えば TurboGears を支えている web サーバ CherryPy もそうです。

ドキュメントが TurboGears wiki にあり、スクリーンキャストへのリンクもあります。TurboGears には関連した質問のほとんどに答えられる、活動的なユーザコミュニティがあります。入門に向いた TurboGears book も出版されています。

次の TurboGears のメジャーバージョン version 2.0 での計画では、Pylons と呼ばれる別の柔軟なフレームワークによって提供される、より柔軟な基本要素に切り替えることになっています。

他の著名なフレームワーク

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全てではありません、人気では少し劣りますが、言及するに値するフレームワークがいくつかあります。

そのうちの一つは既に言及しましたが Zope です、このフレームワークはずいぶんと長い期間にわたって存在しています。Zope 2.x はどちらかというと Pythonic でないとされてきましたが、新しい Zope 3.x ではその変更を試みて、Python プログラマから受け入れられ始めています。これらの努力は既に結果をみせ始めていて、Repoze と呼ばれる Zope と WSGI を結びつけるプロジェクトや Grok と呼ばれる Zope の熟慮された要素を「普通の」Python プログラマが使えるようにするためのプロジェクトがあります。

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。ほとんど全ての要素は交換することができるため、それぞれの要素について、ドキュメントを利用するこが避けられません、なぜなら各必要条件を満す Pylons の組み合わせの可能性は本当にたくさんあるのです。Pylons は WSGI を扱いやすくするための広範囲なツールである、Paste を基にして組まれています。

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

参考:

Python wiki には広大な web frameworks のリストがあります。

多くのフレームワークは独自のメーリングリストや IRC チャンネルを持っています、プロジェクトの web サイトからそれらを探して下さい。また、一般的な話題のために “Python in the Web” IRC チャンネルも #python.web と呼ばれる freenode 上にあります。

索引

Python Enhancement Proposals
    PEP 333, vii