Pylons とその仲間たち
～フレームワークのためのメタフレームワーク～

2007年12月12日

トライアックス株式会社
林　淳哉 junya@triax.jp

TOC

- 自己紹介
- 自分の開発スタイルをフレームワーク化する
- Pylons
- WSGI
- プロジェクトのひな形を見る
- まとめ

自己紹介

- 自己紹介
- Python との付き合いは？
- Pylons との付き合いは？

翻訳プロジェクト

- http://wiki.pylonshq.com/display/pylonsja/

今日お話しすることと、お話しないこと

- 深く話すこと
  - Pylons ってなんぞや？
  - WSGI
  - オレ様フレームワークの作り方
- あまり深く話さないこと
  - ORマッパー
  - フォーム
  - テンプレート言語

- TurboGears 2 は知りません！

Pylons の好きなところ

- スタンダードと柔軟性
- 名前空間
  - c とか g とか h とか・・・
- 自分の開発スタイルをパッケージ化
- ドキュメントが充実
- エラー画面でデバッグ

Pylons の大変なところ・気をつけるところ

- いろんなパッケージに依存
  - ドキュメントが分散
  - バージョンの不整合
- デバッグ画面はセキュリティリスク

Starting at the End

自分の開発スタイルを フレームワーク化する

Triax WEB開発フレームワーク

- ver 0.0.1
- 先週の土曜日に社内リリース

- プロジェクト作成から、管理画面が出来上がるまでのデモです。
  - http://jp.youtube.com/watch?v=6UeYyyOE_JE

特徴

- Pylons をベースにした開発環境
- 管理画面の自動生成
- プロジェクトのひな型を自動生成
  - paster create –t trcms foo
- カスタムコマンド
  - $ paster triaxapp news
  - $ paster customshell
  - $ paster form foo

開発スタイルのフレームワーク化（1）

- MyCMS を作ってみる
- setuptools でパッケージを作成
  - setup.py を書く
  - python setup.py develop
  - パッケージの設定を記述

開発スタイルのフレームワーク化（2）

- プロジェクトのひな型を作成
- テンプレート
  - templates/default_projects/
- 変数置換
  - +package+ : package 名
  - +egg+ : egg 名
  - ${hoge}: 変数で置換

```
hoge.ini_tmpl
+egg+.egg-info
+egg+.egg-info/hoge.ini_tmpl
+package+
+package+/__init__.py_tmpl
+package+/hoge.py_tmpl
```

開発スタイルのフレームワーク化（２－１）

```
class MyCmsTemplate(Template):
    _template_dir = 'templates/default_project'
    summary = ‘MyCMS application template'
    egg_plugins = ['Pylons',
                   'WebHelpers’, ‘MyCMS’]
```

開発スタイルのフレームワーク化（3）

- カスタムコマンドを作成
  - paste.script.command:Command を継承

```
class MyCommand(Command):
    summary = __doc__.splitlines()[0]
    usage = '¥n' + __doc__
    min_args = 1
    max_args = 1
    group_name = 'pylons‘
    default_verbosity = 3
    parser = Command.standard_parser(simulate=True)
    parser.add_option('--no-test',
                      action='store_true',
                      dest='no_test',
                      help="Don't create the test; just the controller")

    def command(self):
        <後略>
```

開発スタイルのフレームワーク化（4）

- setup.py にentry_point を記述

```
setup(
    <中略>
    entry_points="""
    [paste.paster_create_template]
    mycms = mycms.templates:MyCmsTemplate

    [paste.paster_command]
    mycommand=mycms.commands:MyCommand
    """,
)
```

開発スタイルのフレームワーク化（4）

- egg を作成して、配布
  - python setup.py bdist_egg
- 自分の開発スタイルを再利用

```
$ easy_install MyCMS-0.0.1-py2.5.egg
$ paster create –t mycms othercms
```

まとめ

- Pylons のひな形をベースに、WSGIによりかかりながら、好きなパッケージを使って自分の開発スタイルをフレームワーク化できる。
- テンプレートを登録できるので、開発スタイルの再利用が簡単

- 本当の土台は・・・Paste
- Paste のアプリケーション実装がPylons

小さな巨人

PYLONS

Pylons とは

- Pylons
  - URL: http://www.pylonshq.com/
  - 現在の最新版: 0.9.6.1
  - 特徴
    - WSGI 標準に準拠した初めてのフレームワーク
    - 極端なまでの柔軟性
    - TurboGears 2.0 はPylons上で動く
  - 開発者
    - Ben Bangert, James Gardner, Philip Jenvey

問題です

問題： Pylons のソースコードは何ファイルあるでしょう？
（Django は281個、35,069行）

答え: 27個
（3635行）

- なぜ小さくてすむのか・・・・
- それは、WSGIアプリケーションだから。

Pylons とその仲間たち

WSGI Protocol

Routes

SQLAlchemy

AuthKit

Elixir

Beaker

FormEncode

Mako

Pylons

Genshi

ToscaWidgets

Nose

WebHelpers

setuptools

Paste

Pylons の役割（1）- WSGIアプリケーション

- WSGI アプリケーション
- 設定ファイル
- コントローラ
- エラー処理
- StackedObjectProxy 変数（どこでも使える）
  - c : テンプレートと共有するオブジェクト
  - g : グローバルオブジェクト
  - cache : cache オブジェクト
    - 普通は、beaker_cache デコレータを使う
  - request : リクエストオブジェクト
  - session : セッションオブジェクト

Pylons の役割（2） - コマンドによる開発支援

- プロジェクトの雛型を作成する
  - paster create –t pylons mycms
- Controller の雛型を作成する
  - paster controller foo
- Python Shell 環境を提供する
  - paster shell

まとめ

- WSGI準拠
- 小さなパッケージ
- たくさんの外部パッケージに依存
  - 使いたいものだけ使う
  - Pylons じゃなくても使う

- いろんなものを活用するための、
  いい感じの場所を提供している

Python 標準のWeb Server Gateway Interface

WSGI

WSGI って何？

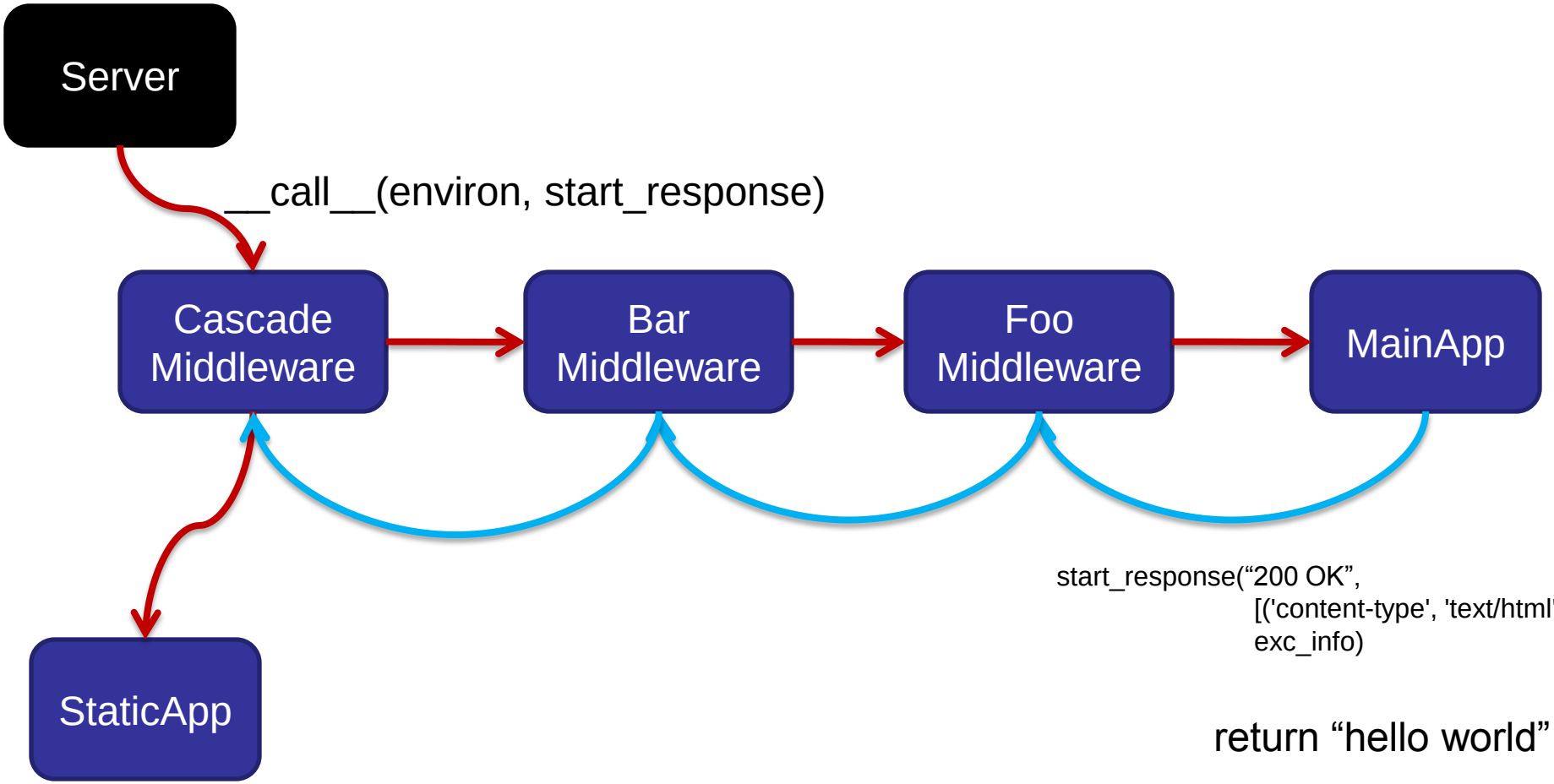
- WSGI（ウィスキー）
- Web Server Gateway Interface
- PEP333
- Web Server と Application をつなぐ仕様
- 登場人物
  - Server
  - Application
  - Middleware

Plagger やDecorator を思い浮かべてください。

WSGI による application リレー（1）

```
def make_app():
    app = MainApp()
    app = FooMiddleware(app)
    app = BarMiddleware(app)
    static_app = StaticApp()
    app = Cascade([static_app, app])
    return app
```

WSGI による application リレー



Server

- environ に値を格納
  - CGI環境変数: PATH_INFO など
  - wsgi 変数: wsgi.version など
- application を呼び出す
  - app(environ, start_response)
- request を受けてresponseを返す

Application

- environ を見てresponse
- 2つの引数を受け取るcallable オブジェクト
  - environ
  - start_response
- start_response でヘッダーを出力

MiddleWare

- 2つの引数を受け取るcallable オブジェクト
  - environ
  - start_response
- 初期化時にapplication を受け取る
- できること
  - environ に値やオブジェクトを格納
  - application の代わりにresponse
  - response をapplication に委譲

WSGI のメリット

- サーバ・アプリケーション間の仕様を定義することで、再利用性を支援
- Middleware でルースカップリング

- Middleware で楽をして、開発したい部分に注力できる

Middleware の例

- Paste: ConfigMiddleware
  - 設定ファイルの情報をenviron に格納
- Paste: Cascade
  - 複数のアプリを連結
  - 404 は次のアプリへ
- Routes: RoutesMiddleware
  - match 情報を格納
    - environ['wsgiorg.routing_args']
    - environ['routes.route']

詳しくは・・・

PEP333-ja：

http://wiki.pylonshq.com/display/pylonsja/PEP333-ja

Pylons で作るとどんな感じ？

プロジェクトのひな形を見る

Welcome to Pylons

$ easy_install Pylons

$ paster create –t pylons mycms

development.ini を 1行コメントアウト

$ paster serve development.ini

ひな形プロジェクトの構成は？

- アプリ記述の3点セット
  - foo/controllers
  - foo/models
  - foo/templates
- 重要なファイル
  - development.ini
  - foo/config/
    - middleware.py : Application 本体
    - routing.py : URLマッピング
    - environment.py : 設定ファイルの読み込み、設定
  - foo/lib/
    - base.py : 外部パッケージはここを通して使います
    - helpers.py : 便利ツール。WebHelpers + α
    - app_globals.py : グローバル変数

config / middleware.py

- Application の entry point
- WSGI Application の作成
- Middleware をリレー

```
def make_app():
    app = PylonsApp()
    app = FooMiddleware(app)
    app = BarMiddleware(app)
    static_app = StaticApp()
    app = Cascade([static_app, app])
    return app
```

config / environment.py

- 設定ファイルを読み込む
- paste.deploy:ConfigMiddleware
  - environ[‘paste.config’]
  - paste.deploy: CONFIG
- 設定をどこからでも読める

config / routing.py

- Routes
- RoutesMiddleware
  - environ['wsgiorg.routing_args']
  - PylonsApp 内で利用
    - environ['pylons.routes_dict‘]
    - コントーラーやアクションの情報

lib / base.py

- コントローラーで必ず読み込むファイル
- StackedObjectProxy 変数（どこでも使える）
  - c : テンプレートと共有するオブジェクト
  - g : グローバルオブジェクト
  - cache : cache オブジェクト
    - 普通は、beaker_cache デコレータを使う
  - request : リクエストオブジェクト
  - session : セッションオブジェクト
- オブジェクトをオーバーライド可能

lib / helpers.py

- 便利な関数群
- 主に、WebHelpers
- 自前で定義も可能
- h で参照

lib / app_globals.py

- リクエストごとではなく、アプリケーション実行中有効なグローバル変数を定義
  - リクエストで値を書き換えると、書き換えた値が残る
- g で参照

（おまけ）lib / vars.py

- 定数はここにおいてます（Triax 流）
- v で参照
  - v.gender.MALE
  - v.gender[‘MALE’]

```
class Gender(Variable):
    MALE = 1
    FEMAIL = 2
    labels = {
            MALE : u”男性”,
            FEMAIL: u”女性”,
    }
gender = Gender()
```

```
class Variable(object, DictMixin):
    def __getitem__(self, key):
        try:
            return self.labels[key]
        except KeyError:
            return u"“
    def keys(self):
        return self.labels.keys()
    def __iter__(self):
        return self.labels.__iter__()
    def iteritems(self):
        return self.labels.iteritems()
```

時間内に終わったでしょうか！？

まとめ

まとめ

- 土台は Paste
- Paste のアプリケーション実装がPylons
- Pylons のプロジェクトのひな形は、パッケージと同じくらい大切
- Pylons のApplication の中身はMiddlewareだらけ・・・WSGIをうまく活用している
- Pylons をベースに、自分スタイルのフレームワークを作ることもできる

Triax で働きたいヒト募集中！！ http://triax.jp/recruit/

ありがとうございました！