Pylons ユーザのための Pyramid 移行ガイド

Author: Nozomu Kaneko

Publication: Django & Pyramid Con JP 2012
(in conjunction with Pycon JP 2012)

Date: 2012-09-16

お前誰よ

- 金子望 (Twitter: @knzm2011)
- 勤務先: トライアックス株式会社
- Pylons を使い始めて5年ぐらい
  - Python 歴もだいたい同じくらい
  - 入社後に Python を使い始めて、気がついたら CMS フレームワークを作ってた
- 翻訳とか
  - Pythonドキュメント日本語翻訳プロジェクト
  - PEP 333: Python Web Server Gateway Interface v1.0
  - Pylons 0.9.7 ドキュメント
  - Pylons プロジェクトドキュメント / Pyramid ドキュメント
- Pylons Project JP の中の人
- 実は Pyramid はあまり使ってない (これから...)

アジェンダ

- 第1部 Pyramid の概要
  - Pyramid FAQ
  - Pylons に何が起こったか
  - Pylons 1.x の何が悪かったのか
  - Pyramid の特徴
- 第2部 移植の方針
  - Pylons 1.0 プロジェクトを Pyramid に移植すべき3つの理由
  - Pylons 1.0 プロジェクトを Pyramid に移植すべきでない理由
  - 移植方法
  - 移植における注意点
  - 移植デモ
- 第3部 Pylons と Pyramid の比較

第1部 Pyramid の概要

Pyramid FAQ

- Pyramid と Pylons の関係は?
  - repoze.bfg と Pylons 1.0 という 2 つのウェブフレームワークが合流して Pyramid ができた
    - コードベースは repoze.bfg
    - Pyramid を開発しているプロジェクトの名前は Pylons プロジェクト
  - ウェブフレームワークとしての Pylons もまだ現役

- repoze.bfg とは何ですか?
  - repoze = Zope 由来のコンポーネントを WSGI アプリケーションで利用できるようにしたコンポーネント集
  - repoze.bfg = repoze のコンポーネントを再構成したフレームワーク

Pyramid FAQ

- Python 3 で動きますか?
  - Pyramid 1.3 から Python 3.2 以上で動きます (thanks to @aodag)

- Pylons 1.0 プロジェクトを Pyramid に移植すべきですか?
  - 後で詳しく説明します

Pylons に何が起こったか

- Pylons 1.0 までは順調に開発が進んだ
  - 特に、周辺ライブラリも含めた Pylons スタックは WSGI ベースのフレームワークとして代表的な存在に
- 拡張性に問題があることが分かった
  - Ben Bangert による blog 記事 (2010年11月): “Why Extending Through Subclassing (a framework’s classes) is a Bad Idea”
- Pylons 2 の開発を進めるうちに repoze.bfg と似てきたことで、コードベースと開発コミュニティをマージする方向へとシフト
- 2011年1月31日 Pyramid 1.0 リリース

Pylons 1.x の何が悪かったのか

- サブクラス化によるフレームワークの拡張
  - フレームワークを改良したくても、すべての主要なメソッドは事実上の API として凍結されていた
    - Pylons では、フレームワークの提供するベースクラスをサブクラス化することでプロジェクトを作成 (BaseController, BaseWSGIApp)
    - ユーザはカスタマイズが必要なメソッドを自由にオーバーライドする
  - 過度の継承によるパフォーマンスの問題
  - フレームワーク内の親クラスに依存するため単体テストしづらい
  - 多重継承による奇妙な衝突が発生
- StackedObjectProxy の使用
  - スレッドローカルかつアプリケーション固有のグローバル変数
  - 時に混乱の原因となる

Pyramid の特徴

- Pylons と repoze.bfg それぞれに由来する豊富な機能 (※)
  - ルーティング: URLディスパッチ or トラバーサル
  - データベースエンジン: SQLAlchemy or ZODB
  - テンプレートエンジン: Mako or Chameleon
  - scaffold
  - インタラクティブデバッガー
  - アクセス制御: ACL
  - フック
- これまでの開発で得られた教訓
  - 徹底したテストコード
  - 徹底したドキュメンテーション
  - サブクラス化に頼らない拡張方法

※ フレームワークが乱立することを防ぐため、Pyramid ではフレームワーク内である程度の機能の重複があることは想定内とされている

第2部 移植の方針

Pylons 1.0 プロジェクトを Pyramid に移植すべき3つの理由

- Pylons 1.0.x は「レガシー」扱いで今後はメンテナンスのみ
- Pyramid の方が (色々な意味で) 強力
  - 特に、拡張性が非常に高い
- Pyramid for Pylons Users が公開された (2012-06-12)
  - 翻訳済み: http://docs.pylonsproject.jp/projects/pyramid_cookbook-ja/en/latest/pylons/index.html

Pylons 1.0 プロジェクトを Pyramid に移植すべきでない理由

- Pyramid 自体が開発中
  - 例) 1.3.2 で Mako テンプレートの継承ができなくなるバグがあり、修正版 (1.3.3) がリリースされるまで 3 ヶ月近くまともに使えなかった
  - 全体的に Pylons ユーザ向けの機能はまだ弱い
- 情報が少ない
  - Pylons (特に大規模プロジェクト) からの移行に関してはあまり情報がない
  - 日本では Pyramid ユーザ自体が少ない (~30人ぐらい?)
  - Pylons Project JP http://www.pylonsproject.jp/ にぜひ参加を (宣伝)

移植する? しない?

- 注: 個人の感想です
  - Pyramid と Pylons は内部がかなり違うので、移植はそれなりに覚悟が必要
  - まずは新規のプロジェクトで Pyramid を試してみる
  - 既存のプロジェクトを移植する場合は、今のうちから計画を準備しておく

移植の戦略 (1) ゼロから Pyramid で書き直す

- あまり変更せずに再利用可能
  - モデル, テンプレート, 静的ファイル
- 変更が必要
  - コントローラ, ルーティング, グローバル変数

移植の戦略 (1) ゼロから Pyramid で書き直す

BeforeRender イベントを使ってレンダラーグローバル変数を追加する例:

```
from pyramid.events import subscriber
from pyramid.events import BeforeRender
from pyramid.threadlocal import get_current_request

from mypylonsproject.lib import helpers

@subscriber(BeforeRender)
def add_renderer_globals(event):
    event["h"] = helpers
    request = event.get("request")
    if request is None:
        request = get_current_request()
    event["c"] = request.tmpl_context
```

移植の戦略 (2) 共存させつつ徐々に移植する

- 一度に 1 つずつ URL を移植する
  - 移植された URL -> Pyramid
  - 移植されていない URL -> Pylons
- 選択肢
  - A) mod_rewrite を使用する
    - Pyramid と Pylons の両方のアプリケーションが別プロセスで実行される
  - B) INI ファイルの中で paste.cascade を設定する
    - 最初に片方のアプリケーションを実行してみて、“Not Found” が返る場合にはもう片方のアプリケーションを試す
    - Pylons が静的 ファイルを返すのと同じ方法
  - C) Pyramid のビューで Pylons アプリケーションをラップする

移植の戦略 (2) 共存させつつ徐々に移植する

NotFound ビューを使用する例:

```
from mypylonsproject import thepylonsapp

class LegacyView(object):
   def __init__(self, app):
      self.app = app
   def __call__(self, request):
      return request.get_response(self.app)

if __name__ == '__main__':
  legacy_view = LegacyView(thepylonsapp)
  config = Configurator()
  config.add_notfound_view(legacy_view)
  # ... rest of config ...
```

移植における注意点

- 複数のアプリケーションを同時に実行する際に調整が必要な箇所
  - データベース接続、セッション、データファイルなど
- Pyramid アプリケーションを Python 2 と 3 のどちらで書くかは重要
  - Python 3
    - Pyramid は Python 3 対応済み
    - Pyramid が必要とするライブラリもほとんどは Python 3 対応している
    - 一部のライブラリは Python 3 対応していない
      - PIL, Paste, WebHelpers, FormEncode, Pylons
  - Python 2
    - Pyramid は Python 2.6 以上で動作
    - 多くのライブラリで Python 2.5 以下は徐々にサポートが打ち切られている
  - レガシー Pylons アプリケーションで使用しているライブラリのバージョンが古いと、Pyramid と共存することが難しくなる

実際にやってみた

- Pylons の QuickWiki チュートリアルを Pyramid に移植
  - 「ゼロから Pyramid で書き直す」パターン
- https://bitbucket.org/knzm/quickwiki-pyramid/wiki/PatchList

第3部 Pylons と Pyramid の比較

Pylons 流のアプリケーション開発が、Pyramid に移行することでどう変わるか

paster コマンド -> p* コマンド

Pylons	Pyramid	備考
paster create	pcreate	オプション -t => -s
paster serve	pserve	
paster shell	pshell	初期設定される変数が違う
paster setup-app	initialize_App_db	App はアプリケーション名

scaffold

- 新しくプロジェクトを開始する場合
  - Pylons: paster create -t <テンプレート名>
  - Pyramid: pcreate -s <scaffold 名>

- 標準 scaffold
  - alchemy: URL ディスパッチ、SQLAlchemy
  - starter: URL ディスパッチ、データベースなし
  - zodb: トラバーサル、ZODB

ディレクトリレイアウト

alchemy scaffold でプロジェクトを作成した場合

```
PyramidApp/
  ├──── development.ini, production.ini
  ├──── setup.py, setup.cfg
  ├──── pyramidapp/
  │       ├──── __init__.py, models.py, views.py, tests.py
  │       ├──── templates/
  │       ├──── static/
  │       └──── scripts/
  └──── PyramidApp.egg-info/
```

- 変更点
  - controllers, config, lib がなくなった
  - views が増えた
  - public が static に名前が変わった
  - アプリケーションの設定が __init__.py に集約された

main 関数

- アプリケーションを返すトップレベルの関数
  - Pylons: pylonsapp/config/middleware.py の make_app 関数
  - Pyramid: pyramidapp/__init__.py の main 関数
- Pyramid の main 関数は Pylons の middleware.py, environment.py, routing.py の内容を含む
- Pyramid では Configurator を使って Pylons よりも簡潔に設定が行える
- WSGI ミドルウェアはありません

main 関数

Pyramid の main 関数の例:

```
from pyramid.config import Configurator

def main(global_config, **settings):
    """ This function returns a Pyramid WSGI application.
    """
    config = Configurator(settings=settings)
    config.add_static_view('static', 'static', cache_max_age=3600)
    config.add_route('home', '/')
    config.scan()
    return config.make_wsgi_app()
```

ルーティングとビュー

- Pylons の場合
  - ルーティングの登録は map.connect() で行う
    - map.connect('/article/{id}', controller='article', action='show')
  - controllers ディレクトリの同名のコントローラが呼ばれる
  - コントローラは BaseController を継承したクラス
- Pyramid の場合
  - ルーティングの登録とビューの登録が分かれている
    - ルーティングの登録
      - config.add_route('article_page', 'article/{id}')
    - ビューの登録
      - config.add_view() または @view_config デコレータ
  - ルーティングは必ず名前を持つ
  - ビューは任意の callable オブジェクトを指定できる
    - 関数、クラス、__call__ メソッドを 実装したインスタンス

view_config のパラメータ

- 述語引数
  - route_name
  - context
  - request_method
  - request_param
  - match_param
  - custom_predicates
- 非述語引数
  - renderer
  - permission
  - http_cache

リソース

- 「トラバース」というルーティング方式を使う場合に重要な概念
- URL ディスパッチ (Pylons と同様のルーティング方式) を使う場合は、
  あまり意識しなくていい
  - 重要なのは root リソースのみ
- 使い方
  - ビジネスロジックの定義
  - セキュリティ

リソースの使用例

```
class Resource(object):
    __acl__ = [
        (Allow, Everyone, 'view'),
        (Allow, 'group:editors', 'edit'),
        ]

    def __init__(self, request):
        self.request = request

config = Configurator(settings=settings, root_factory=Resource)
```

個別のルーティングで使うリソースを指定する場合

```
config.add_route('abc', '/abc', factory=Resource)
```

特殊グローバル変数

pylons.request

- ビュー関数の中では request 引数
- クラスベースのビューメソッドの中では self.request
- テンプレートの中では、request あるいは req
- それ以外の場所では pyramid.threadlocal.get_current_request()
  を呼び出して request を取得可能
  (pshell やユニットテストなどで request オブジェクトが渡ってこない場合)

pylons.response

- Pyramid にはグローバルなレスポンスオブジェクトはない

特殊グローバル変数

pylons.tmpl_context, pylons.c

- テンプレートに変数を渡したい場合、ビューから dict を返すとレンダラー経由でテンプレートに渡る
- ※過去に request.tmpl_context が導入されたが、後に廃止された

特殊グローバル変数

pylons.app_globals

- 最も近い等価物は request.registry か request.registry.settings
- レジストリは Pyramid によって内部で使用されるシングルトン
- request.registry.settings は通常アプリケーションの設定が格納される

特殊グローバル変数

pylons.url (h.url_for)

- request が URL 生成のためのメソッドを持っている
  - request.route_url()
  - request.static_url()
  - request.resource_url()

特殊グローバル変数

pylons.session

- request.session
- pyramid_beaker 拡張を有効にするか、Configurator に session_factory を渡す

```
from pyramid.session import UnencryptedCookieSessionFactoryConfig
session_factory = UnencryptedCookieSessionFactoryConfig('secret')
config = Configurator(session_factory=session_factory)
```

pylons.cache

- Pyramid はキャッシュ機能を内蔵していない
- pyramid_beaker 拡張を使用する

HTTP エラーとリダイレクト

Pylons の場合 (コントローラの中で):

```
abort(404)   # Not Found
abort(500)   # Internal server error
redirect(url("section1"))   # リダイレクト
```

Pyramid の場合 (ビューの中で):

```
raise exc.HTTPNotFound()           # Not Found
return exc.HTTPNotFound()          # 戻り値として返すこともできる
raise exc.HTTPInernalServerError()  # Internal server error
raise exc.HTTPFound(request.route_url("section1"))   # リダイレクト
```

HTTNotFound と HTTPForbidden は Pyramid 内部でも発生する

例外ビュー

特定の例外が起きたときに呼び出されるビューのこと。

```
class ValidationFailure(Exception):
    def __init__(self, msg):
        self.msg = msg

@view_config(route_name='home')
def home(request):
    raise ValidationFailure('some error')

@view_config(route_name='home', context=ValidationFailure)
def failed_validation(exc, request):
    return Response('Failed validation: %s' % exc.msg)
```

静的ファイル

- Pyramid で静的ファイルを返す標準の方法

```
config.add_static_view('static', 'static', cache_max_age=3600)
```

  - 静的 URLにプレフィックス “/static” が付く
  - トップレベルのファイル URL を返せない
- どうするか
  - favicon.ico

```
<link rel="shortcut icon"
href="${request.static_url('pyramidapp:static/favicon.ico')}" />
```

  - robots.txt

```
Alias /robots.txt /var/www/static/norobots.txt
```

静的ファイル

- 高度な使い方
  - 複数のパスを設定する
    - config.add_static_view(name='images', path='static/images')
    - config.add_static_view(name='css', path='static/css')
    - config.add_static_view(name='js', path='static/js')
  - 外部の静的メディアサーバを使う
    - config.add_static_view(name='http://staticserver.com/', path='static')
    - name 引数の値を設定で切り替えることも
- pyramid_assetviews というパッケージを使うとトップレベルのファイル URL を簡単に設定できる (らしい)

```
config.include("pyramid_assetviews")
config.add_asset_views("static", ["robots.txt", "favicon.ico"])
```

asset spec

- パッケージ内のファイルを参照する方法
- パッケージ名と相対パスをコロンで繋げる
  - 例: my.package:static/baz.css
- テンプレートと静的ファイルのパスを指定する箇所で使用できる

セッション

pyramid_beaker を使うと Pylons と同じように設定できる

- ini ファイル

```
session.type = file
session.data_dir = %(here)s/data/sessions/data
session.lock_dir = %(here)s/data/sessions/lock
session.key = akhet_demo
session.secret = 0cb243f53ad865a0f70099c0414ffe9cfcfe03ac
```

- main 関数

```
config.include("pyramid_beaker")
```

おわりに

まとめ

- Pyramid に移行するメリットはある
  - 最新のライブラリへの追従
  - 拡張性
- 既存の Pylons アプリケーションの移植は慎重に考える必要あり
- 移植の戦略
  - ゼロから Pyramid で書き直す
  - 共存させつつ徐々に移植する

Pyramid の拡張方法

※時間の関係で今回は割愛しました (いずれどこかで発表したい)

- 設定ディレクティブ
- ビューマッパー
- リクエストファクトリ
- イベントシステム
- tween (Pyramid 内の WSGI ミドルウェアのようなもの)
- Zope コンポーネントアーキテクチャ (ZCA)

情報源

- Pylons Project JP http://www.pylonsproject.jp/
- facebookグループ pylonsproject.jp
- Pyramid ドキュメント (翻訳)
  http://docs.pylonsproject.jp/projects/pyramid-doc-ja/en/latest/
- Pylons ユーザのための Pyramid (翻訳)
  http://docs.pylonsproject.jp/projects/pyramid_cookbook-ja
  /en/latest/pylons/index.html

Thank you!

ご清聴ありがとうございました