

TurboGearsで作るWebアプリケーション

Python Wrokshop the Edge 2007
TurboGearsハンズオンセミナー 資料
(c) 柴田 淳 / Webcore株式会社

今回のハンズオンセミナーでは、TurboGearsを使って簡単なWeb掲示板を作ってみましょう。

TurboGearsでWebアプリケーションを作るには、以下のような手順を踏みます。

1. コマンドラインツール(tg-admin)を使いプロジェクトを作る
2. 設定ファイルの編集をする(必要があれば)
3. モデルやコントローラのクラス、テンプレートを作る
4. Webブラウザなどでテストをする
5. 3~4を完成するまで繰り返す

CGIのような旧来的な方法を使った場合だと、どのような手順で開発を進めるのかと言うところから考えないとなりません。しかし、TurboGearsのようなWebアプリケーションフレームワークの多くでは、開発の手順が決められているため、作りたいアプリケーションを素早く作り始めることができます。見通しが立つので、開発のモチベーションが維持しやすいのも利点の一つです。

プロジェクトの作成

TurboGearsでWebアプリケーションを作るときには、まず**プロジェクト**を作ります。プロジェクトは、TurboGearsで開発をするときの基本単位となります。プロジェクトの実体は、ソースコードや設定ファイルなどをディレクトリ(フォルダ)にまとめたものです。では実際に、プロジェクトを作ってみましょう。

プロジェクトを作るには、tg-adminというコマンドを使います。

```
$ tg-admin quickstart <-- このコマンドを入力, 以降はリターンのみ
Enter project name: tgbbs
Enter package name [tgbbs]:
Do you need Identity (usernames/passwords) in this project? [no]
Selected and implied templates:
TurboGears#tgbase      tg base template
TurboGears#turbogears  web framework
:
:
```

入力事項を入力し終わると、tg-admin が必要なファイルを書き出し、課程がログとして表示されます。

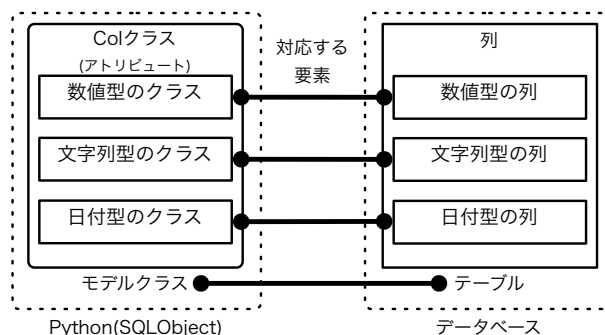
プロジェクトの作成が終わると、プロジェクト名と同名のディレクトリができています。シェルでプロジェクトディレクトリに移動してみましょう。移動したら「start-tgbbbs.py」というスタートアップ用スクリプトを起動してみましょう。Linuxなどであればスクリプトをコマンドとして入力します。Windowsなら、スクリプトファイルをダブルクリックします。Webブラウザを使って、「<http://127.0.0.1:8080/>」というURLにアクセスしましょう。

モデルの作成

さて、いよいよTurboGearsを使ったWeb掲示板の開発に取りかかります。まずは、掲示板の投稿を保存するために**モデル**を定義します。掲示板の投稿をデータベースに保存し、必要に応じて取り出すための仕組みをモデルと呼びます。

TurboGearsでは、標準ではSQLObjectと呼ばれるPythonのライブラリを使ってデータベースとのやりとりを行います。SQLObjectはO/Rマップと呼ばれるライブラリです。O/Rマップを使うと、SQLiteのようなリレーショナルデータベースとの間で通信を行う際に、SQLという問い合わせ言語を使うことなく、データの書き込みや取り出しが行えます。

SQLObjectでは、クラスを定義してテーブルとの**マッピング**を表現します。テーブルとマッピングするクラスは、SQLObjectというクラスを継承して定義します。また、テーブルではデータを保存するための**列**(Row, ローとも呼ばれます)を定義しますが、SQLObjectではクラスにアトリビュートを定義してテーブルの列をPythonのデータ型との関連性を表現します。



テキストエディタでmodel.pyというファイルを開いてください。そうしたら、以下のコードを書き込んでモデルの定義をします。ここでは、掲示板の書き込みのタイトル、投稿者の名前、本文を保存するテーブルを定義しています。

```
class Article(SQLObject):
    body = UnicodeCol()
    title = UnicodeCol()
    author = UnicodeCol()
```

モデルクラスができたなら、「tg-admin sql create」というコマンドを入力し、データベースのテーブルを作ります。

テストデータの投入

次に、シェルを使ってテストデータを投入してみます。シェルでプロジェクトディレクトリをカレントディレクトリにして「tg-admin shell」と入力してください。すると、Pythonのインタラクティブシェルが起動します。この状態で、TurboGearsの開発でよく利用するライブラリがインポートされています。model.pyに定義したモデルクラスもインポート済みですので、すぐに利用できます。

インタラクティブシェルを使って、モデルにテスト用のデータを投入してみましょう。

```
>>> Article(title='test1', author='name1', body='body1'*10)
>>> Article(title='test2', author='name2', body='body2'*10)
```

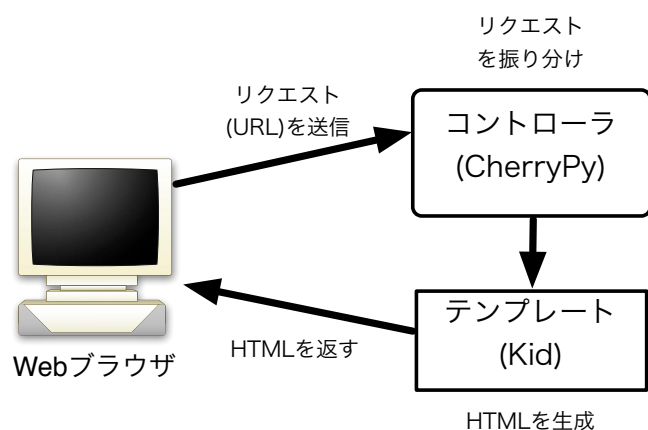
このコードは、model.pyに定義されたArticleというクラスのインスタンスを生成しています。純粋なPythonのコードですが、裏側ではデータベースとのやりとりが行われています。データベースの処理をするとき、SQLをほとんど意識せずに処理が行える。これがSQLObjectのようなO/Rマッパーの魅力です。次に、登録したデータを取り出してみましよう。登録したデータを取り出すためには、クラスのメソッドを使います。

```
>>> print [a.id for a in Article.select()]
[1, 2]
>>> print Article.get(1)
<Article 4 body="u'body1body1body1...'...">
```

コントローラとテンプレートの作成

次に、データをWebに表示するためにコントローラとテンプレートを作成します。

TurboGearsので「コントローラ」の役割を担っているのは、CherryPyです。CherryPyを使うと、WebブラウザなどからのリクエストをPythonのクラスで受け取り、処理をすることができるようになります。CherryPy自体がWebサーバとして稼働しますので、ApacheなどのWebサーバを別途立てる必要がありません。



TurboGearsのビュー(テンプレート)を担当するのはKidというテンプレートエンジンです。HTMLのタグの内部に、動的に置き換える要素を指定するルール(ロジック)を埋め込むことで、HTMLなどのテキストの一部を動的に置き換え、出力します。Kidのテンプレートは独立したテキストファイルとして設置します。Pythonのコードと、HTMLに埋め込むデザイン的な要素が分離されるため、効率よくWebアプリケーションを開発することができます。また、Kidのロジック部

分はHTMLのタグ内部に記述するため、テンプレートファイルをDreamweaverのようなHTML編集専用のツールで編集することもできます。

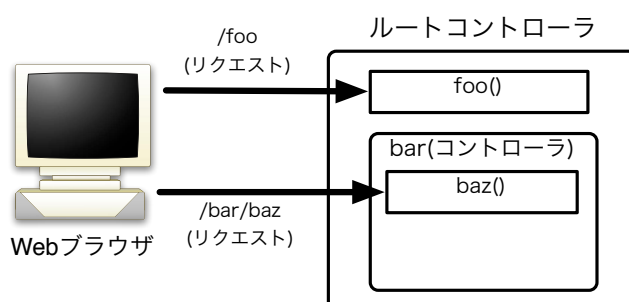
Webアプリケーション側でどのテンプレートを使うかの指定はコントローラ側で行います。コントローラのクラスに定義したメソッドに、関数デコレータの形でテンプレートのパスを指定します。テンプレート側では、コントローラから受け取ったオブジェクトなどを元にして、HTMLなどを出力します。

コントローラの作成

初期プロジェクトにcontrollers.pyというスクリプトファイルがあります。このファイルを拡張することで、コントローラを作ってゆきます。controllers.pyというファイルを、エディタで開いてみてください。

ファイルには、一つクラスが定義されているはずです。このクラスが、Webブラウザのリクエストを受けて応答を返すコントローラの定義です。このクラスはルートコントローラ(RootController)と呼ばれるクラスを継承しています。ルートコントローラは、Webアプリケーションの「ルート(最上位)」に位置するコントローラで、ルートのリクエストがこのコントローラに渡されます。TurboGearsでWebアプリケーションを作るときには、かならずこのルートコントローラを一つ作ります。

TurboGearsでは、URLをスラッシュで区切った階層を分割して処理を行うコントローラメソッドを探し出す、という仕組みでURLとコントローラの対応が決まります。たとえば、「/foo」というリクエストはルートコントローラの「foo()」というメソッドが受け取ります。メソッドが存在しない場合はエラーとなります。



ルートコントローラを出発点にして、アトリビュートの階層構造をたどってリクエストが渡されてゆきます。

まず、必要なモジュールをインポートするために、ソースの先頭に以下のコードを書き込みます。

```
# coding=utf-8
from turbogears import controllers, expose, flash # この行は入力不要
from turbogears import redirect
from model import Article # モデルクラスをimportする
```

次に、ルートコントローラに以下のメソッドを定義しましょう。インデントに注意してください。

index()というメソッドは、URLにスラッシュだけでアクセスしたときの「インデックスアクセス」を受け取る特殊なメソッドです。

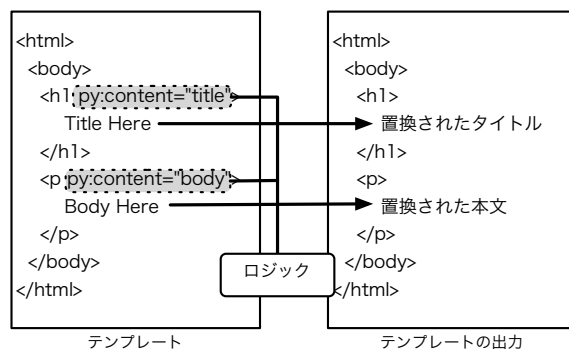
```
@expose(template="tgbbs.templates.bbslist")
def index(self):
    art_list = [] # 投稿一覧を表示するリストを定義
    for a in Article.select(): # DBから投稿一覧を取り出し、ループ
        art_list.append(a) # DBから取り出した投稿をリストに追加
    return { 'art_list':art_list }
```

テンプレートの作成

次に、HTMLを表示するテンプレートを作ります。

tg-adminで作ったプロジェクトでは、いくつかのテンプレートがあらかじめ作られています。controllers.pyなどがあるディレクトリに「templates」というディレクトリがあり、ここに「.kid」という拡張子のテンプレートが何種類か見えるはずですが、これが、TurboGearsで利用するテンプレートです。

「welcome.kid」というファイルをテキストエディタで開いてみてください。これは、コントローラのindex()メソッドで利用しているテンプレートです。ごく普通のHTMLには見えませんが、よく見ると「py:extends」や「py:replace」という見慣れない記法があるのが分かります。これが、Kidのロジックを表記している部分です。



Kidでは、「py:xxx」というアトリビュートを使ってロジックを指定します。xxxに相当する部分には何種類かの「Kidの命令」を記入します。どのようなロジックを埋め込むかによって、いろいろな命令を指定します。命令の後ろにはイコールに続けてクオーテーションを使って文字列を記入します。文字列の部分には、Pythonの式を記入します。どのような命令を使うかによって、どのような式を記入するかが変わってきます。

アトリビュート領域を使ってロジックを記述する他、HTML内部にPythonのコードを埋め込むこともできます。

TurboGearsのテンプレートは、controllers.pyと同じ階層にある「templates」というディレクトリにまとまっています。ここに、bbslist.kidというテキストファイルを作ります。テンプレートの文字コードはUTF-8にします。

TurboGesarsのテンプレートでは、コントローラの戻り値として渡された辞書が変数として利用できます。辞書のキーと同名の変数に、キーの値が代入された形で利用できま

す。index()メソッドでは「art_list」というキーに投稿一覧のリストを返していました。テンプレート側では、art_listという変数に投稿一覧のリストが渡されてくるわけです。

welcome.kidの内容をコピーして、bodyタグの内部を以下のように書き換えます。

```
<h1>投稿一覧</h1>
<div py:for="article in art_list">
  <h3 py:content="article.title">タイトル</h3>
  <div style="color: gray"
    py:content="article.author">
    投稿者の名前
  </div>
  <p py:content="article.body">
    投稿本文
  </p>
</div>
```

このテンプレートでは、コントローラから渡ってきたオブジェクトを元にループを組み、投稿の内容を表示しています。

- ・ py:content

タグの内部を、イコールに続くPython の式が返す結果でエレメント全体を置き換えます。

- ・ py:replace

タグ自体を含め、イコールに続くPython の式が返す結果でエレメント全体を置き換えます。

- ・ py:for

指定した式を元に、エレメントの内部を繰り返します。

- ・ py:attrs

イコール以下に辞書を指定し、アトリビュート(hrefやsrcのような)の値を動的に置き換えます。リンクや画像を動的に生成したい場合に利用します。

- ・ py:if

イコールに続く条件式が成り立つときだけ、タグ自体を含むエレメントを出力します。条件が成り立たない場合はエレメントを出力しません。

コントローラメソッドとテンプレートができれば、プロジェクトを起動します。その後、「<http://127.0.0.1:8080/>」というURLにアクセスしてみてください。先ほど投入したテストデータがちゃんと表示できていればOKです。

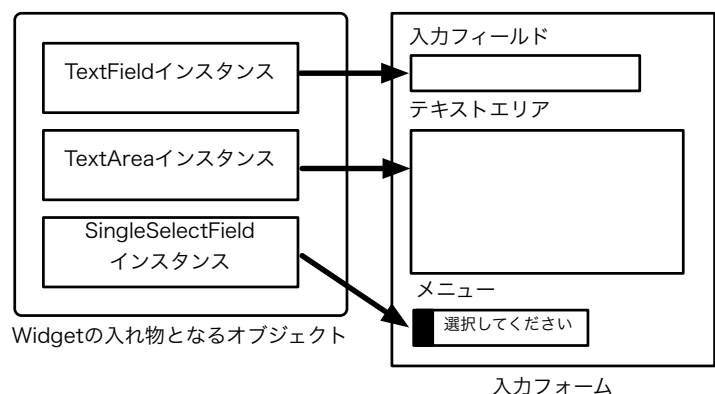
新規投稿フォームの作成

投稿を表示する仕組みはできましたので、次に投稿用のフォームを作ります。

TurboGearsでは、フォームを作る際にはWidgetと呼ばれる仕組みを使います。Widgetとはフォームの部品を抽象化したPythonのクラスです。フォームの表示や遷移のコントロール、バリデーションチェックなどを担当しています。HTMLを生書きしてフォームを作るより、ずっと手軽に高度なWebアプリケーションに求められる遷移を作成できます。

Widgetは「turbogears.widgets」というパッケージの下に登録されています。テキストフィールドやラジオボタンなど、基本的なフォームを作るためのWidgetが内蔵されています。

Widget用のクラスには、大きく分けて二つの種類があります。一つは、フォームの部品となるWidgetです。もう一種類は、フォーム部品の「入れ物」となるWidgetです。フォームの部品となるWidgetクラスのインスタンスを複数作り、この入れ物となるクラスに登録します。テンプレートには入れ物となるクラスを渡して表示します。



まずはフォームを表示するためにWidgetのオブジェクトを作りましょう。controllers.pyのクラス定義の前に、以下のコードを書いてください。

```
# ウィジェットをimportする
from turbogears.widgets import TableForm, TextField, TextArea

# widgetを定義する
title = TextField(name="title", label=u"タイトル")
author = TextField(name="author", label=u"名前")
body = TextArea(name="body", label=u"本文")

article_form = TableForm()
# フォームのフィールドにwidgetを指定する
article_form.fields = [title, author, body]
```

次にフォームを表示するコントローラメソッドとテンプレートを作ります。ルートコントローラクラスに、以下のメソッドを定義します。

```
@expose(template="tgbbs.templates.form")
def show_form(self, id=None):
    value = {}
    return { 'form':article_form, 'action':'add_article',
            'value':None }
```

フォームの投稿を受け付けるメソッドを定義します。show_form()メソッドの返す辞書の、actionというキーに指定されたメソッド名を使います。また、投稿の内容はメソッドの引数として受け取ります。CherryPyがフォームからのリクエストを分解し、引数として渡してくれるのです。

```
@expose()
def add_article(self, title, author, body):
    a = Article(title=title, author=author, body=body)
    return redirect('/')
```

最後に、フォームを表示するテンプレートを作ります。先ほど作ったbbslist.kidをコピーして、form.kidというファイルを作ります。bodyの内部を以下のように書き換えます。

```
<h1>投稿をする</h1>
<div py:replace="form.display(action=action,value=value)" />
```

ここまでできたら、再度プロジェクトを起動します。「http://127.0.0.1/show_form」というURLにアクセスし、フォームが表示できるかどうか試してみてください。フォームが表示できたら、実際に投稿をしてみてください。

バリデータの指定

これで一通りWeb掲示板の機能が調いました。しかし、問題があります。

今の状態では、フォームに何も入力しない場合でも掲示板への投稿ができてしまうのです。タイトルのない投稿、本文のない投稿をされては困ります。そこで、ここでは入力値のチェック機能を付けてみようと思います。

TurboGearsでは、バリデータという仕組みを使って入力値のチェックを行います。バリデーションチェック専用のデコレータと、Widgetを組み合わせることで、とても簡単に入力値のチェックと再入力機能を持った高度なWebアプリケーションが作れます。

add_article()メソッド定義の直前に以下のコードを記入してください。

```
from turbogears import validate, error_handler, validators
@expose() # この行はそのまま
@turbogears.error_handler(show_form)
@turbogears.validate(form=article_form,
                    validators={'title':validators.NotEmpty(),
                               'author':validators.NotEmpty(),
                               'body':validators.NotEmpty(),})
```

最初のデコレータでは、エラーが起こったときにフォームの再表示をするためのコントロールメソッドを指定しています。次のデコレータでは、値の入力に利用するフォームと、値チェックの方法を指定しています。ここでは、すべて「validate.NotEmpty()」を指定して、入力がない場合はエラーとして扱うように指定しています。

実際にフォームで空のまま投稿をしてみてください。投稿が行われず、エラーの表示を伴ってフォームが再表示されるはずですよ。

課題

時間の余裕がある方は、以下の課題に取り組んでみてください。

タイトルのデフォルトを表示

フォームを表示するとき、タイトルにデフォルトの文字列を表示するようにしてみてください。

投稿者名が空だった場合にデフォルトの文字列を登録するように

投稿フォームの名前欄を「空」でも投稿できるようにします。空だった場合には「anonymous」のような文字列を登録するように書き換えてみてください。

個別の書き込みを表示できるように

投稿全体のリストだけでなく、個別の書き込みを表示できるようにしてみてください。