Using Django in Non-Standard Ways

Eric Florenzano

PyCon, Atlanta, GA,
February 19, 2010

Why give this talk?

Why give this talk?

- “I tried to bring Django into my workplace, but we would have had to customize things so much that it wouldn’t be worth it.”

Why give this talk?

- “I got fed up with the restrictive template langauge. It’s too bad since I liked all the other stuff, but now I’m back in PHP.”

Non-Standard Django

- Two Main Categories
- Choosing alternatives to what Django offers
- Using bits of Django in other contexts

The main thing

- It’s not as hard as you think it’s going to be

Choosing Alternatives to what Django offers

1. Using Jinja2 With Django

Using Jinja2

- What is it?
  - An alternative templating system
  - It makes different tradeoffs from Django's template system
  - Sometimes Jinja2 makes more sense

Using Jinja2

- We created an app called django_ext
- It mirrors Django's layout exactly
- But we swap out Django's template rendering ideas for Jinja2's equivalent

Using Jinja2 (Cont'd)

```
from django.shortcuts import render_to_response
```

```
from django_ext.shortcuts import render_to_response
```

Using Jinja2 (Cont'd)

```
# IMPORTS
from django.conf import settings
from django.http import HttpResponse
from django.template.context import get_standard_processors
from jinja2 import FileSystemLoader, Environment

# ONE INSTANTIATION
env = Environment(
    loader=FileSystemLoader(settings.TEMPLATE_DIRS)
)

# ONE 5-LINE FUNCTION
def render_to_response(tmpl, dct, req=None,
                       mimetype=settings.DEFAULT_CONTENT_TYPE):
    for processor in get_standard_processors():
        dct.update(processor(request))
    rendered = env.get_template(tmpl).render(**dct)
    return HttpResponse(rendered, mimetype=mimetype)
```

Using Jinja2 (Cont'd): Imports

```
from django.conf import settings

from django.http import HttpResponse
from django.template.context import \
    get_standard_processors
from jinja2 import FileSystemLoader
from jinja2 import Environment
```

Using Jinja2 (Cont'd): Instantiation

```
ld = FileSystemLoader(settings.TEMPLATE_DIRS)
env = Environment(loader=ld)
```

Using Jinja2 (Cont'd): One 5-Line Function

```
def render_to_response(tmpl, d, req=None,
    m=settings.DEFAULT_CONTENT_TYPE):
    for processor in get_std_processors():
        dct.update(processor(request))
    rendered = env.get_tmpl(tmpl).render(**d)
    return HttpResponse(rendered, mimetype=m)
```

That's pretty much it!

Note: in Django 1.2, you can just write a template loader.

What about my apps?

- Truth is, some of them will work with Jinja2, and some of them won't
- If they won't, you don't need to throw them away...
- They're free code that you can use to go 95% of the way
- Just modify it to use Jinja2

Choosing Alternatives to what Django offers

2. Not using django.contrib.auth

Why not use auth?

- When using it will be more difficult than not using it
- When using it will make your code less straightforward than not using it

e.g. Writing a Facebook App

Facebook App Basics

- You don't render HTML to the user
- You render FBML, which Facebook assembles and renders for the user
  - User → Facebook → Django → Facebook → User
- No such thing as user registration
  - You're given a unique id for each user

How to tackle this?

- Didn't even attempt to make it fit the **django.contrib.auth** paradigm
- We just plain don't use the **User** model
- Created a tiny app with one model whose PK is the Facebook User ID
- Wrote one decorator function to redirect to an authorization page if not auth'd

Time Taken

- About 45 minutes to create our custom app
- About 1 hour to convert the few apps that we needed to using our new model
- Note we didn't discard the apps

Advantages

- Straightforward code with clear intent
- Didn't waste time trying to shoehorn Facebook User IDs into the username field of the User model
- Reduced overhead (no need for any of the auth or sessions machinery)
- Still get to use the rest of the Django stack

Disadvantages

- Had to write some of our own stuff
- Had to modify apps

Choosing Alternatives to what Django offers

3. Not Using the ORM

Why not use the ORM?

- Integrating with complex legacy databases
- Using a database that the ORM doesn't support
  - e.g. Talking to a non-relational database
- You're not talking to a "database"

Wait a minute... Database backends are pluggable!

- You won't have time
- It might not make sense
  - e.g. writing a backend for Cassandra

Real-world example

- Service for accessing/modifying data
- Based on Pylons
- Speaks HTTP + JSON
- Talks to a PostgreSQL database
- Written before Django was introduced to this workplace

Web Service Example

- curl -d '{"s": "bloons"} -H 'content-type: application-json' http://server-name/metaflip/games/get_game_by_slug
- {"tag": "eb5d4e50c49bc832", "name": "Bloons", "approved": true, "width": 640, ...}
- Have existing clients written to use this

Now we want to create a Django app that wants to use this data

We've got a choice...

- Don't use Django
- Use Django

  - Create models for these objects, and maintain them when we change the database, etc.

  OR

  - Just use what we've already got

How that app looks:

```
games/
    __init__.py
    urls.py
    views.py
    models.py # Intentionally Empty
    tests.py
    context_processors.py
    templatetags/
        __init__.py
        games_tags.py
```

Here's a view:

```
from my_lib import games_client as g

def play(request, game_slug=None):
    game = g.get_game_by_slug(game_slug)
    if not game:
        raise Http404
    return render_to_response(
        'games/play.html',
        {'game': game},
    )
```

What about the admin?

- We already had one, written in Pylons
- If not we would have had to write one
  - Annoying: yes, show-stopping: no

Other similar services

- High score leaderboards
- Achievements
- Game plays

Using bits of Django in Other Contexts

1. Using Django's Forms in Pylons

Using Forms in Pylons

- Initial implementation: No form library
  - Parsed the POST and validated in-line
- Looked at other alternatives
  - FormEncode
  - FormBuild
  - WTForm (actually tried using this)

Decided we liked Django's Best

- How do we make this work?
  - Turn off Django's I18N handling
  - New Form base class to coerce WebOb (Unicode)MultiDicts into a QueryDict
  - A Genshi wrapper to allow the form's HTML through to the template

What about settings?

- That was how we turned off I18N
- (We don't need I18N yet)
- No messing with environment vars, etc.

```
from django.conf import settings
settings.configure(USE_I18N=False)
```

In Total:

- 60 Lines of glue code
- And now we get to use all the nice validation and form display that Django affords us

Using bits of Django in Other Contexts

2. Using Django's ORM Stand-Alone

Why use ORM Stand-Alone?

- Probably because you like the API more than other solutions out there
- Maybe you already have Django apps that define models, and you want to use them outside of a web context

Steps to make this work

- Make sure the app with models is on your python path
- Call settings.configure with your DB info
- (Optionally copy manage.py to your proj.)
- Import your models and use them

Using WSGI Middleware with Django

WSGI Middleware

- For some reason, most Django users don't use it
- It's easy to use with Django
- Start by looking at Repoze

Repoze Examples

- **repoze.bitblt** - Automatically scales images
- **repoze.squeeze** - Merges JS/CSS automatically based on statistical analysis
- **repoze.profile** - Aggregates Python profiling data across all requests, and provides an HTML UI for viewing the data

Typical .wsgi File

```
import os,sys
sys.path.append('/usr/local/django')
os.environ['DJANGO_SETTINGS_MODULE'] = \
    'mysite.settings'

from django.core.handlers.wsgi import \
    WSGIHandler

application = WSGIHandler()
```

Middleware .wsgi File

```
import os,sys
sys.path.append('/usr/local/django')
os.environ['DJANGO_SETTINGS_MODULE' = \
    'mysite.settings'

from django.core.handlers.wsgi import \
    WSGIHandler
from repoze.profile.profiler import \
    AccumulatingProfileMiddleware as P

application = P(
    WSGIHandler(),
    log_filename='/tmp/profile.log'
)
```

Other Cool Non-Standard Stuff

- **YardBird** - IRC using Django's URL mapping to match messages and views to handle the callbacks
- **Djng** - Microframework built on Django
- **Jngo** - Single-File Django CMS

Questions?

- Twitter: @ericflo
- http://www.eflorenzano.com/
- http://mochimedia.com/jobs.html