The Story of App-Loading

Aymeric Augustin - DjangoCon Europe 2014

From the Kunst- und Wunderkammer, Schloss Ambras, Innsbruck, Austria.
Hello, I’m Aymeric

Core Developer since October 2011

- Time zones
- Python 3
- Transactions
- App loading

Chief Technical Officer since January 2014

- Original & new auto parts
- Pure player
- We’re hiring!
What is app loading?

"'cause you know, it's the first time I hear about it."
App loading

1. A step in Django’s initialization sequence
   - Import all models
   - Populate a cache

2. A project that aims at improving this step
   - Provide extensibility
   - Make it more robust
“add support for custom app_label and verbose_name”
“add support for custom app_label and verbose_name”
Timeline

- Feb ’07 — Joseph Kocherhans posts a proof of concept
- Mar ’07 — Vinay Sajip starts a series of patches
- Mar ’08 — State of Django keynote and sprints at PyCon
- May ’10 — Arthur Koziel participates in GSoC
- Aug ’10 — Jannis Leidel maintains the branch
- Aug ’12 — Preston Holmes picks up the flag
- Jun ’13 — Marc Tamlyn investigates
- Dec ’13 — Aymeric Augustin reboots the project
Original proposal

# myproject/settings.py

from django.conf.apps import app

INSTALLED_APPS = (  
    # Old syntax - backwards compatible
    'django.contrib.auth',

    # New syntax - extensible
    app('custom.auth',  
        app_label='myauth',  
        verbose_name='My cool auth app'),
)

Final implementation

# myproject/apps.py

from django.apps import AppConfig

class MyAuthConfig(AppConfig):  
    name = 'custom.auth'
    label = 'myauth'
    verbose_name = "My cool auth app"

# myproject/settings.py

INSTALLED_APPS = [  
    # Old format - backwards compatible
    'django.contrib.auth',  
    # New format - points to an application configuration class
    'myproject.apps.MyAuthConfig',
]
Initialization sequence

```python
import threading, time
from django.core import signals
from django.dispatch import receiver

state = threading.local()

@receiver(signals.request_started)
def save_start_time(sender, **kwargs):
    state.start_time = time.time()

@receiver(signals.request_finished)
def print_request_duration(sender, **kwargs):
    try:
        duration = time.time() - state.start_time
    except Exception as exc:
        print("Exception: {}").format(exc)
    else:
        print("Request took {:.3f} seconds".format(duration))
```
Initialization sequence

$ django-admin.py runserver

Request took 0.032 seconds
Request took 0.023 seconds
Request took 0.020 seconds

$ gunicorn myproject.wsgi

Exception: 'thread._local' object has no attribute 'start_time'
Request took 0.020 seconds
Request took 0.020 seconds

Bugs that only happen in production: the best kind of bugs!
Why does Django have an app loading process?

I wasn’t there, but I love version control speleology.
Here’s a model

```python
# monsters/models.py

from django.conf import settings
from django.db import models

class Monster(models.Model):
    name = models.CharField(max_length=100, unique=True)
    slayer = models.ForeignKey(settings.AUTH_USER_MODEL,
                                related_name="monsters_slain")

    def __str__(self):
        return self.name
```
The ORM does the rest

```
$ django-admin.py shell

>>> from django.contrib.auth.models import User

>>> heracles = User.objects.get(first_name="Heracles")

>>> heracles.monsters_slain.all()
[<Monster: Nemean Lion>,
 <Monster: Lernaean Hydra>,
 <Monster: Stymphalian Birds>]
```
Let’s find out

# monsters/models.py

from django.conf import settings
from django.db import models

class Monster(models.Model):
    raise RuntimeError("You shall not know about us!")

    name = models.CharField(max_length=100, unique=True)
    slayer = models.ForeignKey(settings.AUTH_USER_MODEL,
                                related_name="monsters_slain")

    def __str__(self):
        return self.name
In a Python shell

$ django-admin.py shell

Traceback (most recent call last):
  File "./manage.py", line 11, in <module>
    execute_from_command_line(sys.argv)
  File "./django/core/management/__init__.py", line 399, in execute_from_command_line
    utility.execute()
  File "./django/core/management/__init__.py", line 392, in execute
    self.fetch_command(subcommand).run_from_argv(self.argv)
  File "./django/core/management/base.py", line 242, in run_from_argv
    self.execute(*args, **options.__dict__)
  File "./django/core/management/base.py", line 285, in execute
    output = self.handle(*args, **options)
  File "./django/core/management/base.py", line 415, in handle
    return self.handle_noargs(**options)
  File "./django/core/management/commands/shell.py", line 70, in handle_noargs
  get_models()
  File "./django/db/models/loading.py", line 232, in get_models
    self._populate()
  File "./django/db/models/loading.py", line 75, in _populate
    self.load_app(app_name, True)
  File "./django/db/models/loading.py", line 99, in load_app
    models = import_module('%s.models' % app_name)
... (stack trace depends on the version of Python here)
  File "./dceu14al/monsters/models.py", line 7, in <module>
    class Monster(models.Model):
  File "./dceu14al/monsters/models.py", line 9, in Monster
    raise RuntimeError("You shall not know about us!")
RuntimeError: You shall not not know about us!

tl;dr — "shell" calls AppCache.get_models() in django.db.models.loading. That method reads INSTALLED_APPS and imports each application’s models module.
In a development server

$ django-admin.py runserver

Traceback (most recent call last):
  File "~/django/utils/autoreload.py", line 93, in wrapper
    fn(*args, **kwargs)
  File "~/django/core/management/commands/runserver.py", line 101, in inner_run
    self.validate(display_num_errors=True)
  File "~/django/core/management/base.py", line 310, in validate
    num_errors = get_validation_errors(s, app)
  File "~/django/core/management/validation.py", line 34, in get_validation_errors
    for (app_name, error) in get_app_errors().items():
  File "~/django/db/models/loading.py", line 196, in get_app_errors
    self._populate()
  File "~/django/db/models/loading.py", line 75, in _populate
    self.load_app(app_name, True)
  File "~/django/db/models/loading.py", line 99, in load_app
    models = import_module('%s.models' % app_name)
... (stack trace depends on the version of Python here)
  File "~/dceu14al/monsters/models.py", line 7, in <module>
    class Monster(models.Model):
  File "~/dceu14al/monsters/models.py", line 9, in Monster
    raise RuntimeError("You shall not know about us!")
RuntimeError: You shall not know about us!

tl;dr — “runserver” calls AppCache.get_app_errors() in django.db.models.loading in order to validate models. That method also imports each application’s models module.
In a production server

```
$ gunicorn dceu14al.wsgi
```

Traceback (most recent call last):
File "~/django/core/handlers/base.py" in get_response
  response = wrapped_callback(request, *callback_args, **callback_kwargs)
File "~/djang...try: return self.admin_view(view, cacheable)(*args, **kwars)
File "~/djang...view_func(request, *args, **kwars)
File "~/djang...func(request, *args, **kwars)
File "~/djang...cache_user = SimpleLazyObject(lambda: get_user(request))
File "~/djang...user_id = request.session[SESSION_KEY]
File "~/djang...cached_user = auth.get_user(request)
File "~/djang...user = auth.get_user(request)
File "~/djang...session[key]
File "~/djang...session[key]
File "~/djang...session_cache = self_load()
File "~/djang...session/db" in _get_session
File "~/djang...manager.py" in load
File "~/djang...models/query.py" in get
File "~/djang...models/query.py" in clone
```

File "~/django/db/models/query.py" in clause
  self.add_q(where_part, used_aliases)
File "~/django/db/models/sql/query.py" in add_q
  current_negated=kwargs)
```

File "~/django/db/models/sql/query.py" in build_filter
  allow_explicit_fk=True)
File "~/django/db/models/sql/query.py" in setup_joins
  names, opts, allow_many, allow_explicit_fk)
File "~/django/db/models/sql/query.py" in names_to_path
  field, model, direct, m2m =
  opts.get_field_by_name(name)
File "~/django/db/models/options.py" in get_field_by_name
  cache = self.init_name_map()
File "~/django/db/models/options.py" in init_name_map
  for model in
  self.get_all_related_m2m_objects_with_model():
    File "~/django/db/models/options.py" in get_all_related_m2m_objects_with_model
      cache = self._fill_related_many_to_many_cache()
    File "~/django/db/models/options.py" in _fill_related_many_to_many_cache
      for klass in get_models(only_installed=False):
        File "~/django/db/models/options.py" in self_populate
      File "~/django/db/models/loading.py" in load_models
        models = import_module('%s.models' % app_name)
    ... (stack trace depends on the version of Python here)
    File "~/dceu14al-monsters/models.py" in <module>
```

```
tl;dr — When the auth middleware accesses the session, Django loads it from the database. Twelve frames deeper, the ORM calls AppCache.get_models() in django.db.models.loading.
```
```
Enter `django.db.models.loading.AppCache`

- `_populate()` imports the models module of each app.
- Importing models sets up relations and reverse-relations.
- This needs to be done before using the ORM.
- Model classes register themselves with the app cache.
- This provides introspection of apps and models.
- Global state galore :-(

19
The app cache doesn’t make a list of models. Models submit themselves to the app cache.
Hardest bug

$ git log --reverse | egrep -B8 '#1796\b' | egrep 'Author|Date'

Author: Malcolm Tredinnick <malcolm.tredinnick@gmail.com>
Date:   Sun Jun 25 14:49:34 2006 +0000
Author: Malcolm Tredinnick <malcolm.tredinnick@gmail.com>
Date:   Mon Jun 26 09:01:07 2006 +0000
Author: Malcolm Tredinnick <malcolm.tredinnick@gmail.com>
Date:   Mon Jun 26 16:20:58 2006 +0000
Author: Malcolm Tredinnick <malcolm.tredinnick@gmail.com>
Date:   Thu Jul  6 13:25:12 2006 +0000
Author: Malcolm Tredinnick <malcolm.tredinnick@gmail.com>
Date:   Tue Aug  1 21:15:41 2006 +0000
Author: Malcolm Tredinnick <malcolm.tredinnick@gmail.com>
Date:   Sat Feb 17 01:38:03 2007 +0000
Author: Malcolm Tredinnick <malcolm.tredinnick@gmail.com>
Date:   Fri Aug 17 17:23:15 2007 +0000
Author: Aymeric Augustin <aymeric.augustin@m4x.org>
Date:   Mon Dec 30 11:12:40 2013 +0100
I shit you not

```bash
$ git log \
  | tr ' ' '\n' \
  | sed -n 's/#\([0-9][0-9]*\).*$/\1/p'
  | sort
  | uniq -c
  | sort
  | tail -n 7

8 1796   # Bug: app-cache initialization
11 627    # Refactoring: create django.contrib.admin
13 19385  # Feature: add multi-column joins
15 2333   # Feature: add the test framework
16 15300  # Batch: pull translations
21 5106   # Refactoring: create DatabaseOperations
43 17822  # Batch: update and push translations
```
Interesting behavior

# monsters/models.py

from django.db import models

class Monster(models.Model):
    evil = True

print(Monster.evil)  # True
Interesting behavior

```python
# monsters/models.py

from django.db import models

class Monster(models.Model):
    evil = True

Original = Monster

class Monster(models.Model):
    evil = False

print(Monster.evil)  # True
print(Monster is Original)  # True
```
Not even an accident

```python
# django/db/models/base.py

class ModelBase(type):
    def __new__(cls, name, bases, attrs):
        # ...

        return new_class

    # June 26th, 2006 — in 919df8b8 / r3212:
    # Fixed another path where imports were creating two
    # instances of a model's class. Refs #1796, #2232.

    return get_model(new_class._meta.app_label, name)
```
Other scary bedtime stories

- AppCache is a singleton based on the Borg pattern:

  ```python
class AppCache(object):
    __shared_state = {}

    def __init__(self):
        self.__dict__ = self.__shared_state
  ```

- For thread-safety, `_populate()` takes the import lock.

- To avoid import loops, `_populate()` is re-entrant.

- `_populate()` deals with dependencies by postponing imports that fail and retrying them at the end.
What’s new in Django 1.7?

“I hope you broke all my code for a good reason.”
App-loading reloaded - goals

1. Allow apps without a models module or package
2. Provide verbose_name for the admin
3. Provide a reliable initialization signal
App-loading reloaded - non-goals

4. Allow changing app_label in case of name clashes

5. Allow changing db_prefix

6. Clarify the app cache population sequence

7. Enforce consistency between the contents of INSTALLED_APPS and the AppCache
App-loading reloaded - non-goals

8. Provide APIs to discover things at arbitrary Python or filesystems subpaths inside apps

9. AppAdmin — #7497

10. App-specific settings in application objects

11. Order of INSTALLED_APPS — #21018

12. Support deploying multiple copies of an app — most likely not worth solving, if possible at all
Apps without models

$ find . -empty -name models.py -exec git rm {} \;

$ git commit -m "Look ma, no models!"
Providing verbose_name

# rock_n_roll/apps.py

from django.apps import AppConfig

class RockNRollConfig(AppConfig):
    name = 'rock_n_roll'
    verbose_name = "Rock 'n' roll"

# rock_n_roll/__init__.py

default_app_config = 'rock_n_roll.apps.RockNRollConfig'
Start-up hook

# rock_n_roll/apps.py

from django.apps import AppConfig

class RockNRollConfig(AppConfig):
    name = 'rock_n_roll'

    def ready(self):
        from . import signals
        print("Signals registered! Rock on!")

# rock_n_roll/__init__.py

default_app_config = 'rock_n_roll.apps.RockNRollConfig'
Changing app_label

# core/billing/apps.py

from django.apps import AppConfig

class CoreBillingConfig(AppConfig):
    name = 'core.billing'
    label = 'core_billing'

# web/billing/apps.py

from django.apps import AppConfig

class WebBillingConfig(AppConfig):
    name = 'web.billing'
    label = 'web_billing'
App registry population

• Population is triggered explicitly when Django starts.

• Request handling: `get_wsgi_application()` calls `django.setup()` before returning the WSGI callable.

• Management commands: `execute_from_command_line()` calls `django.setup()` just after parsing the command line.

• Standalone scripts: they must call `django.setup()` — **backwards-incompatible.**
Consistency

• Apps must have unique labels — **backwards-incompatible**.

• Models must be imported with the same Python path — **backwards-incompatible**.

• Models must be defined in an installed application or provide their app_label explicitly — **deprecation path**.

• Models mustn’t be imported before their application — **deprecation path**.
Order of INSTALLED_APPS

If several applications provide the same resource, the earliest application in INSTALLED_APPS wins.

- Management commands — backwards-incompatible.
- Static files
- Templates
- Template tags and filters libraries
- Translations
```
from django.apps import AppConfig

AppConfig.name          # Python path ('django.contrib.admin')
AppConfig.label          # Arbitrary identifier ('admin')
AppConfig.verbose_name   # Human-readable name ("Administration")
AppConfig.path           # Filesystem path.

AppConfig.module         # Root module of the application.
AppConfig.models_module  # Models module, may be None.

AppConfig.get_models()   # -> Iterable of Model classes.

AppConfig.get_model(model_name)  # -> Model class or LookupError.

AppConfig.ready()        # -> Start-up hook.
```
from django.apps import apps

apps.get_app_configs()
# -> iterable of AppConfig instances.

apps.get_app_config(app_label)
# -> AppConfig instance or LookupError.

apps.get_model(app_label, model_name)
# -> Model class or LookupError.

apps.ready
# -> True when the registry is populated.

apps.is_installed(app_name)
# -> True if an application with that name exists.
Upgrade tips

Not all errors are hard to fix.
RuntimeError: App registry isn't ready yet.

• Defer or refactor code that depends on the app registry at import time.
  • ugettext —> ugettext_lazy
  • get_user_model() —> AUTH_USER_MODEL
  • functools.lru_cache

• Move startup code to AppConfig.ready().

• Add django.setup() in standalone scripts.
ImportError: cannot import name ...

- Look for a circular import involving Apps.populate().
- Check if your third-party packages support Django 1.7.
- Have a tree of dependencies between your applications.
- Cache on first use instead of computing at import time.
Dynamic models and other app loading hijackings

- You’re going to have a bad time porting from 1.6 to 1.7.
- You’re going to have a worse time supporting both.
- Check tickets with the “app-loading” keyword in Trac.
- Django 1.7’s explicit app-loading design will hopefully make creative hacks more viable!
  - Take advantage of AppConfig.ready().
One more thing

Django’ design philosophies 101: loose coupling
Django 1.6 project template

# project_name/urls.py

from django.conf.urls import patterns, include, url

from django.contrib import admin

admin.autodiscover()

urlpatterns = patterns('',
    # Examples:
    # url(r'^$', '{{ project_name }}.views.home', name='home'),
    # url(r'^blog/', include('blog.urls')),

    url(r'^admin/', include(admin.site.urls)),
)

not an URL.
Django 1.7 project template

# project_name/urls.py

from django.conf.urls import patterns, include, url
from django.contrib import admin

urlpatterns = patterns('',
    # Examples:
    # url(r'^$', '{{ project_name }}.views.home', name='home'),
    # url(r'^blog/', include('blog.urls')),
    url(r'^admin/', include(admin.site.urls)),
)

The admin’s default AppConfig performs autodiscovery.
too complicated; didn’t listen

https://docs.djangoproject.com/en/1.7/ref/applications/
Thank you!

Questions?

From the Kunst- und Wunderkammer, Schloss Ambras, Innsbruck, Austria.