04 o que não te contaram sobre django

Settings

Existem algumas fraquezas em usar um conjunto de settings globais para todos os ambientes.

3 hacks para melhorar seu settings.py

`PROJECT_PATH`

`settings_local.py`

`auto debug`

PROJECT_PATH

A criação dinamica de um PROJECT_PATH facilita a execução em multiplos ambientes

```
import os

PROJECT_PATH = os.path.abspath(os.path.split(__file__)[0])
...
MEDIA_ROOT = os.path.join(PROJECT_PATH,'media')
```

settings_local.py

se existir `settings_local.py` executa ela

```
# ultima coisa no settings
try:
    execfile(PROJECT_PATH+'/settings_local.py')
    #print 'Usando configuracao LOCAL'
except IOError:
    #print 'Usando configuracao PADRAO'
    pass
```

auto debug

O menos aconselhavel das 3 modificações

```
import socket

# Set DEBUG = True if on the production server
if socket.gethostname() == 'your.domain.com':
    DEBUG = False
else:
    DEBUG = True

TEMPLATE_DEBUG = DEBUG
```

Bonus hack

no `settings.py`

```
TEMPLATE_TAGS = (
        'projeto.foo.templatetags.foobar',
        'projeto.bar.templatetags.foobar',
)
```

autoload template_tags

no __init__.py

```
from django.conf import settings
from django.template import add_to_builtins
#import django.template.loader

try:
    for lib in settings.TEMPLATE_TAGS:
        add_to_builtins(lib)
except AttributeError:
    pass
```

Costumizando o Admin

criar o diretorio de templates especial pro admin

```
mkdir -p templates/admin

cd templates/admin
```

base_site.html

```
{% extends "admin/base.html" %}

{% block title %}{{ title }} | Admin da pizzaria {% endblock %}

{% block branding %}
<h1 id="site-name" > Pizzaria</h1>
{% endblock %}
```

existe ainda um {% block extrastyle %} para colocar styles de css

OBS: incluir tambem o <style> e </style>

Applicações para conhecer

`django-config` Gerencia varias configurações

`django-debug-toolbar` Fantastico!

`haystack` Facilita search

`sphinx` Grande ferramenta de documentção

`celery` Controle de tarefas assincronamente

`fabric` Deploy e gerenciamento remoto

`gunicorn` Servidor WSGI

`varnish` Solucao para cache

debugando

usando o pdb

```
import pdb; pdb.set_trace()
```

alguns comandos

```
w(here)
c(ont)
n(ext)
s(tep)
l(ist)
a(rgs)
b(reak)
p(rint)
u(p)
```

Stand alone scripts

```
#!/usr/bin/python
#coding:utf8

# local onde esta o settings
SETTINGS_PATH = "/home/aluno/pizza"
import sys, os
sys.path.append(SETTINGS_PATH)

from django.core.management import setup_environ
import settings
setup_environ(settings)

# a partir daqui coisas como
# from pizza.entrega.models import Cliente funciona
# funcionam
```

virutalenv

Permite que voce use diversas versões de pacotes na mesma maquina

Criando

```
virtualenv --no-site-packages ambiente
```

Ativação

```
source ambiente/bin/activate
```

para desativar

```
deactivate
```

Instalar pacotes

Atenção sem o sudo

```
pip install django
```

Decoradores

Recebe uma função como parametro e retorna uma função

Exemplo

```
def decorador(func):
    func.tipo = "decorada"
    return func

@decorador
def foo(valor):
    print valor
# python antigo
# foo = decorador(foo)
```

usando

```
>>> foo("oi")
oi

>>> foo.tipo
'decorada'
```

funções dentro de funções

```
def decorador(func):
    def nova_f(*args):
        print "iniciando", func.__name__
        func(*args)
        print "terminando"
    return nova_f
```

executando

```
>>> foo("oi")
iniciando  foo
oi
terminando
>>> print foo.__name__
'nova_f'
```

decoradores com parametros

```
@decorador("legal")
def foo(valor):
    print valor
```

Fabrica de decoradores

```
def decorador(tipo):
    def fabrica(func):
        def nova_f(*args):
            print "tipo ", tipo
            print "iniciando ", func.__name__
            func(*args)
            print "terminando"
        return nova_f
    return fabrica
```

python < 2.4

```
foo = decorador("legal")(foo)
```

decoradores como Classes

```
class decorador(object):
    def __init__(self, func):
        self.f = func

    def __call__(self,*args):
        print "iniciando "
        self.f(*args)
        print "terminando"
```

Decoradores importantes

metodos de classe

```
Foo.bar("larari")
```

@staticmethod

```
class Foo(object):
    @staticmethod
    def bar(cls, valor):
        print valor

>>> Foo.bar("larari")
larari
```

Propriedades

```
>>> f = Foo()
>>> f.bar
'algo'
```

@property

```
class Foo(object):
    @property
    def bar(self):
        return "algo"
```

Cuidado

```
>>> f.bar = 123
AttributeError: can't set attribute
```

getters e setters

Não precisa para atributos simples como:

```
class Foo():
    dia = 11
>>> f = Foo()
>>> f.dia
11
>>> f.dia = 12
>>> f.dia
12
```

Para atributos complexos

```
from datetime import datetime

class Foo(object):
    data = datetime.now()

    def get_dia(self):
        return self.data.day

    def set_dia(self, dia):
        self.data = self.data.replace(day=dia)

    dia = property(get_dia,set_dia)
```

Uso

```
f = Foo()
f.data
datetime.datetime(2009, 6, 20, 13, 44, 22, 463668)
f.dia
20
f.dia = 21
f.dia
21
f.data
datetime.datetime(2009, 6, 21, 13, 44, 22, 463668)
```

do django

```
from django.contrib.auth.decorators
                                import login_required

@login_required
def foo(request):
    ...
```

Permissões para usar o admin

```
from django.contrib.admin.views.decorators
                        import staff_member_required

@staff_member_required
def foo(request):
    ...
```

permisso permisso

Usuario logado e o resto?

No admin

```
can_delete_entrega
```

na view

```
if request.user.has_perm('aluno.delete_entrega'):
    pass
```

Signals

Sinais não são o que voce pensa

Eles são:

Sincronos

Rolam na mesma thread

Exemplo de uso

modificar um valor antes de salvar

```
from django.db.models import signals

class Foo(models.Model):
    validade = models.DateTimeField()

def marca_val(sender,instance,**kwargs):
    if not instance.validade:
        instance.validade = "2012-01-01"
signals.pre_save.connect(marca_val, sender=Foo)
```

No shell

```
>>>from larari.models import Foo
>>>a = Foo()
>>>a.validade

>>> a.save()
>>> a.validade
'2012-01-01'
#ou datetime.datetime(2012, 1, 1, 0, 0)
```

DB Signals

pre_save, post_save, pre_delete, post_delete Requer o parametro sender

enviam sender, instance e outros(post_save manda o created)

m2m_changed

alem destes manda ainda action veja a documentação

http://docs.djangoproject.com/en/dev/ref/signals/

Signals de gerencia

```
post_syncdb

request_started

request_finished

got_request_exception

connection_created
```

Signals que so rolam em testes

`template_rendered`

Seus Signals

```
from django.dispatch import Signal

compra_pronta = Signal(providing_args=["c_id"])

class Compra():
    def finalizar_compra(self):
        ...
        compra_pronta.send(sender=self,c_id=self.id)
```

Ouvindo seus signals

```
def compra_callback(sender,c_id,**kwargs):
    pass

compra_pronta.connect(compra_callback)
```

cache

```
CACHE_BACKEND = 'memcached://127.0.0.1:11211/'

# tem que fazer no shell antes:
# ./manage.py createcachetable [cache_table_name]
CACHE_BACKEND = 'db://my_cache_table'


CACHE_BACKEND = 'file:///var/tmp/django_cache'

CACHE_BACKEND = 'locmem://'

CACHE_BACKEND = 'dummy://' # NAO CACHEIA
```

middlewares

```
MIDDLEWARE_CLASSES = (
    # tem que ser o primeiro
    'django.middleware.cache.UpdateCacheMiddleware',
    ...
    # tem que ser o ultimo
    'django.middleware.cache.FetchFromCacheMiddleware',
)
CACHE_MIDDLEWARE_SECONDS  = 90 #segundos
CACHE_MIDDLEWARE_KEY_PREFIX = "" # somente multiplos sites
```

Tipos de cache

por view

template fragment cache

API de baixo nivel

Tipos de cache

por view

template fragment cache

API de médio nivel

por view

```
from django.views.decorators.cache import cache_page

@cache_page( 90 ) #segundos
def foo(request):
    ...
```

ou direto na urls.py

urls.py

```
from django.views.decorators.cache import cache_page

urlpatterns = ('',
    (r'^foo/$', cache_page(foo, 90)),
)
```

template fragment cache

legal demais e simples demais

```
{% load cache %}
{% cache 500 barra_menu %}
    .. barra_menu ..
{% endcache %}
```

Mas

E para usuários logados?

Valores dinamicos?

Cache por usuário

```
{% load cache %}
{% cache 500 barra request.user.username %}
    ... barra do usuario ...
{% endcache %}
```

API de médio nivel

```
>>> from django.core.cache import cache
>>> timeout = 5
>>> cache.set("chave","objeto",timeout)
>>> cache.get("chave")
"objeto"
>>> cache.get("chave")
None
```

Mais metodos

```
cache.clear()

cache.add(chave,valor)
cache.get(chave,valor_padrao)
cache.delete(chave)
# django 1.2
cache.set_many({chave1:val1, chave2:val2})
cache.get_many([chave1,chave2,chave3])
cache.delete_many([chave1,chave2,chave3])
```

Cache para contadores

```
cache.set("contador", 1)
cache.incr("contador")

cache.incr("contador",5)

cache.decr("contador")
```

Cache entre o django e o browser

```
from django.views.decorators.vary import vary_on_headers

@vary_on_headers('User-Agent')
def foo(request):
    ...
```

Cookies

```
from django.views.decorators.vary import vary_on_cookie

@vary_on_cookie
def foo(request):
    ...
```

private

```
from django.views.decorators.cache import cache_control

@cache_control(private=True)
def foo(request):
    ...
```

ou forca cache

```
from django.views.decorators.cache import cache_control

@cache_control(must_revalidate=True, max_age=3600)
def foo(request):
    ...
```

sem cache

```
from django.views.decorators.cache import never_cache

@never_cache
def foo(request):
    ...
```

Gerando feeds

urls.py

```
from foo.bar.views import BlogFeed, AtomBlogFeed
from django.contrib.syndication.views import feed

feeds = {
    'rss': BlogFeed,
    'atom': AtomBlogFeed,
}


...
(r'^feeds/(?P<url>.*)/$', 'feed', {'feed_dict': feeds}),
...
```

views

```
from django.contrib.syndication.feeds import Feed
from django.utils.feedgenerator import Atom1Feed

class BlogFeed(Feed):
    title = "Foo"
    link = "/"
    def items(self):
        return Post.get_open()[:10]


class AtomBlogFeed(BlogFeed):
    feed_type = Atom1Feed
    subtitle = BlogFeed.description
    author_name="me"
```

Extra fields

```
def item_link(self, item):
    return "/post/%s"% item.slug

def item_pubdate(self, item):
    return item.published_at

def item_author_name(self, item):
    return item.author
```

Dicas

sempre valida o feed

atom é mais chato

Custom everything

models fields

form fields

widgets

template tags

template filters

custom model fields

antes de mais nada leia a documentação

```
class TrueCharField(models.Field):
    def db_type(self, connection):
        return 'char(100)'

class ModeloBatuta(models.Model):
    fixo = TrueCharField()
```

Coisas a observar

`__metaclass__ = models.SubfieldBase`

Para tipos costumizados de objetos python

`db_type`

Da o formato em SQL básico

`to_python`

Formata do formado to SQL para python

`get_prep_value`

Formata o contrario de python pra SQL

exemplo complexo

```
class MesAnoField(models.DateField):
    __metaclass__ = models.SubfieldBase
```

```
def value_to_string(self, obj):
    return self._get_val_from_obj(obj)
```

de sql para python

```
def to_python(self,value):
    if not value:
        return
    if isinstance(value,str):
        return value
    if isinstance(value,unicode):
        return value
    return '%s/%s' % (value.month,value.year)
```

de python para sql

```
def get_prep_value(self, value):
    if value:
        mes,ano = map(int,value.split('/'))
        return date(ano,mes,1)
```

custom form fields

Basicamente sobreescrever o clean

```
class BrFloatField(forms.FloatField):
    widget = BrFloatWidget # vemos a seguir

    def clean(self,value):
        value = value.replace(',','.')
        return super(BrFloatField,self).clean(value)
```

exemplo prático

```
def clean(self, value):
    if value in EMPTY_VALUES:
        return None
    try:
        int(value)
    except (ValueError, TypeError):
        raise ValidationError(u'Código de Cliente inválido')

    try:
        value = Cliente.objects.get(id=value)
    except self.model.DoesNotExist:
        raise ValidationError(u'Cliente não existe')
    return value
```

custom widgets

Quando cresce o numero de clientes fica claro que isso não funciona por culpa do html

```
class ClienteForm(forms.Form):
    cliente = forms.ModelChoiceField(Cliente.objects.all())
# FUNCIONA MAS NAO FUNCIONA PARA MUITOS CLIENTES
```

```
<select>
    ...milhares de <option>s
</select>
```

Alternativa

fazer um widget(representação html do campo) mais apropriado

Simples

subclasseia uma widget proxima e muda o render

exemplo

```
class BrFloatWidget(forms.fields.TextInput):
  def render(self, name, value, attrs=None):
    if value:
        if type(value) is float:
            value = '%.2f' % value
            value = value.replace('.',',')
    else:
        value = '0,00'

    return super(BrFloatWidget,self).render(name, value, attrs)
```

Só pra lembrar

```
class BrFloatField(forms.FloatField):
    widget = BrFloatWidget # <-----

    def clean(self,value):
        value = value.replace(',','.')
        return super(BrFloatField,self).clean(value)
```

widgets mais complexos

class Media

```
class OneToManySearchInput(forms.SelectMultiple):
    class Media:
        css = {
            'all': ('/static/jquery.autocomplete.css',)
        }
        js = (
            '/static/jquery.bgiframe.min.js',
            '/static/jquery.ajaxQueue.js',
            '/static/jquery.autocomplete.js'
        )
```

no template

```
{{ form.media }}
<form>
    {{ form }}
</form>
```

e usar marksafe

```
def render(self, name, value, attrs=None):
    return mark_safe(u"""
        <script>........
    """)
```

Custom template tags e filters

```
mkdir entrega/templatetags
cd entrega/templatetags
```

criar um arquivo

footags.py

```
from django import template

register = template.Library()

@register.filter
def foo(val):
    return len(val)
```

no template

```
{% load footags %}

foo tem tamanho {{ texto_qualquer|foo }}
```

Ou mais prático

```
from django.template.defaultfilters import floatformat
from django import template

register = template.Library()

@register.filter
def brfloatformat(value,arg=-1):
    value = floatformat(value,arg)
    return value.replace('.',',')
brfloatformat.is_safe = True
```

mark_safe

se volta algum html que é seguro marca com

```
from django.utils.safestring import mark_safe

@register.filter
def foo(val):
    return mark_safe("<big>%s</big>"% len(val))
foo.needs_autoescape = True
```

custom template tags

```
agora é {% agora "%Y-%m-%d" %}
```

tem que criar um parser e um node

parser

```
def agora_parser(parser, token):
    try:
        tag_name, format_string = token.split_contents()
    except ValueError:
        # nao tem dois parametros
        raise template.TemplateSyntaxError
    # nao tem aspas no segundo parametro
    if not (format_string[0] == format_string[-1]
                and format_string[0] in ("'", '"')):
        raise template.TemplateSyntaxError
    return AgoraNode(format_string[1:-1])
```

Node

```
class AgoraNode(template.Node):
    def __init__(self, format_string):
        self.format_string = format_string
    def render(self, context):
        return datetime.datetime.now().strftime(self.format_string)
```

Registra

```
@register.tag(name="agora")
def agora_parser(parser, token):
    ....
```

agora mais simples

```
{% lista_comentarios post %}
```

```
<dl>
    <dt>fulano</dt>
    <dd>maior legal</dd>
    <dt>ciclano</dt>
    <dd>não gostei</dd>
</dl>
```

Cria o método

```
def lista_comentarios(post):
    comments =  post.comentario_set.all()
    return {"comentarios":comments}
```

comentarios.html

dentro de algum `/templates`

```
<dl>
    {% for comment in comentarios %}
    <dt>{{comment.autor}}</dt>
    <dd>{{comment.texto}}</dd>
    {% endfor %}
</dl>
```

dai registra

```
@register.inclusion_tag('comentarios.html')
def lista_comentarios(post):
    comments =  post.comentario_set.all()
    return {"comentarios":comments}
```

block tags

```
{% mimimi %}O dia foi lindo{% endmimimi %}
```

```
O dia foi mimimi
```

parser e node

```
def do_mimimi(parser, token):
    nodelist = parser.parse(('endmimimi',))
    parser.delete_first_token()
    return MimimiNode(nodelist)
```

node

```
class MimimiNode(template.Node):
    def __init__(self, nodelist):
        self.nodelist = nodelist
    def render(self, context):
        output = self.nodelist.render(context)
        return output.replace("lindo", "mimimi")
```