Django výpisky

© 2010 Petr Nohejl, aktualizace: 16. 6. 2010

# Instalace

* instalace Python
* odstranění staré verze Djanga
* instalace Djanga příkazem python setup.py
* ověření správné instalace příkazem import django v interpretu
* přidání C:\Python26;C:\Python26\Lib\site-packages\django\bin; do systémové proměnné PATH

# Vytvoření projektu

Příkaz django-admin.py startproject myproject vytvoří v aktuálním adresáři projekt s názvem myproject. Projekt nesmí mít název, shodující se s nějakou build-in komponentou Pythonu a Djanga. Kód není vhodné z hlediska bezpečnosti ukládat do kořenového adresáře webserveru (/var/www). Toto se u Djanga, narozdíl od PHP, nedělá.

Nový projekt obsahuje soubory:

* \_\_init\_\_.py – vytvoření package
* manage.py – utilita pro správu projektu
* settings.py – nastavení projektu
* urls.py – deklarace URL adres

## Manage.py

* manage.py runserver – spustí vývojový webserver na adrese <http://127.0.0.1:8000/>, není vhodný pro ostrý provoz (lepší použít Apache), při vytvoření nového souboru v projektu je server nutno restartovat
* manage.py runserver 1.2.3.4:8080 – spustí vývojový webserver na dané IP a portu
* manage.py syncdb – vytvoří všechny potřebné tabulky, indexy a příkazy (které ještě neexistují) v databázi pro všechny aplikace, definované v proměnné INSTALLED\_APPS, (provede příkazy z manage.py sqlall myapp), při první synchronizaci aplikace django.contrib.auth rovněž nabídne vytvoření účtu superuživatele
* manage.py createsuperuser – vytvoří účet superuživatele
* manage.py validate – kontroluje chyby v definici modelů aplikací
* manage.py sql myapp – vypíše SQL kód pro vytvoření tabulek (CREATE TABLE) databáze aplikace
* manage.py sqlcustom myapp – vypíše SQL kód pro všechny uživatelské příkazy
* manage.py sqlclear myapp – vypíše SQL kód pro odstranění tabulek (DROP TABLE) databáze aplikace
* manage.py sqlindexes myapp – vypíše SQL kód pro vytvoření indexů (CREATE INDEX) databáze aplikace
* manage.py sqlall myapp – kombinace příkazů sql, sqlcustom, sqlindexes
* manage.py shell – spustí interpret a nastaví prostředí projektu (přidá myproject do systémové proměnné PATH a nastaví proměnnou prostředí DJANGO\_SETTINGS\_MODULE)

## Settings.py

* DATABASE\_ENGINE – typ databáze ('postgresql\_psycopg2', 'postgresql', 'mysql', 'sqlite3', 'oracle')
* DATABASE\_NAME – název databáze nebo absolutní cesta k sqlite3 ('D:/myproject/sql.db'), pokud soubor sqlite3 neexistuje, bude vytvořen při první synchronizaci, u databází PostgreSQL nebo MySQL je třeba databázi vytvořit příkazem CREATE DATABASE jmeno\_databaze;
* DATABASE\_USER – databázové uživatelské jméno (v sqlite3 se nepoužívá)
* DATABASE\_PASSWORD – databázové heslo (v sqlite3 se nepoužívá)
* DATABASE\_HOST – adresa databázového serveru, pokud je databáze na stejném serveru jako Django, nechá se řetězec prázdný (v sqlite3 se nepoužívá)
* DATABASE\_PORT – port databázového serveru
* TEMPLATE\_DIRS – cesta k adresářům se šablonami HTML
* INSTALLED\_APPS – seznam aplikací
  + django.contrib.auth – autentifikační systém
  + django.contrib.contenttypes – framework pro typy obsahu
  + django.contrib.sessions – framework pro sessions
  + django.contrib.sites – framework pro spravování několika samostatných webů v jedné instanci Djanga
  + django.contrib.admin – administrační rozhraní
  + myproject.myapp – vlastní aplikace

# Vytvoření aplikace

Příkaz python manage.py startapp myapp vytvoří v aktuálním adresáři (pro zjednodušení nastavíme adresář projektu myproject) aplikaci s názvem myapp. Důsledkem zjednodušení bude vzájemné propojení aplikace a projektu.

Nová aplikace obsahuje soubory:

* \_\_init\_\_.py – vytvoření package
* models.py – definice modelů (popis dat), struktura databáze s dodatečnými metadaty, jeden model je jedna databázová tabulka
* views.py – definice pohledů (šablony webové stránky)

## Models.py

Modely popisují data aplikace (strukturu databáze). Ctí princip DRY (Don’t repeat yourself). Každý model je jedna databázová tabulka a je potomkem třídy django.db.models.Model. Proměnné modelu definují jednotlivá databázová pole – instance třídy Field. Pojmenováním pole Field se definuje strojové jméno, přes které se bude přistupovat k hodnotě pole a v databázi bude využito k pojmenování sloupce databázové tabulky. Volitelně lze v prvním argumentu instance Field definovat human-readable jméno pole.

from django.db import models

import datetime

class Poll(models.Model): #definice modelu

question = models.CharField(max\_length=200) #textove pole o delce 200 znaku

pub\_date = models.DateTimeField('date published') #datum s human-readable jmenem

def \_\_unicode\_\_(self): #definice unicode metody

return self.question

def was\_published\_today(self): #definice vlastni metody

return self.pub\_date.date() == datetime.date.today()

#human-readable pojmenovani metody

was\_published\_today.short\_description = 'Publikovano dnes?'

class Choice(models.Model): #definice dalsiho modelu

poll = models.ForeignKey(Poll) #cizi klic

choice = models.CharField(max\_length=200) #textove pole o delce 200 znaku

votes = models.IntegerField() #celociselne pole

def \_\_unicode\_\_(self): #definice unicode metody

return self.choice

Unicode metody jsou užitečné při práci v interaktivním režimu pro výpis (např. >>> Poll.objects.all()) a používají se v automaticky generovaném administračním rozhraní.

Modely je potřeba po vytvoření aktivovat příkazem manage.py syncdb. Poté se vytvoří databázové schéma aplikace a databázové API pro přístup k datům. Název tabulky se automaticky generuje z názvu aplikace a názvu modelu. Primární klíč 'id' se definuje automaticky. Název cizího klíč se automaticky generuje z názvu pole a suffixu '\_id'.

## Urls.py

V adresáři s projektem v souboru urls.py v proměnné urlpatterns jsou deklarovány URL adresy. Pro určitý regulární výraz, odpovídající nějaké URL adrese, se zavolá odpovídající funkce z views.py. Příklad volání: detail(request=<HttpRequest object>, poll\_id='23') pro regulární výraz r'^polls/(?P<poll\_id>\d+)/$', přičemž část textu v závorkách se odešle jako argument do view funkce a '?P<poll\_id>' definuje název, kterým se zachycená část pojmenuje. Tyto regulární výrazy neprohledávají parametry GET, POST, a ani názvy domén. Při požadavku na http://www.example.com/myapp/?page=3, bude opět hledat myapp/.

from django.conf.urls.defaults import \*

from django.contrib import admin

admin.autodiscover()

urlpatterns = patterns('',

(r'^polls/$', 'myproject.myapp.views.index'),

(r'^polls/(?P<poll\_id>\d+)/$', 'myproject.myapp.views.detail'),

(r'^polls/(?P<poll\_id>\d+)/results/$', 'myproject.myapp.views.results'),

(r'^polls/(?P<poll\_id>\d+)/vote/$', 'myproject.myapp.views.vote'),

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

)

Každá aplikace může mít definovány svoje vlastní urls.py, které se pak importují do hlavního urls.py projektu.

# urls.py projektu

from django.conf.urls.defaults import \*

from django.contrib import admin

admin.autodiscover()

urlpatterns = patterns('',

(r'^polls/', include('myproject.myapp.urls')),

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

)

# urls.py aplikace

from django.conf.urls.defaults import \*

# prvni argument je zkratka k views, aby se nemusela opakovat v kazdem callbacku

urlpatterns = patterns('myproject.myapp.views',

(r'^$', 'index'),

(r'^(?P<poll\_id>\d+)/$', 'detail'),

(r'^(?P<poll\_id>\d+)/results/$', 'results'),

(r'^(?P<poll\_id>\d+)/vote/$', 'vote'),

)

## Views.py

Příklad základního view:

from django.http import HttpResponse

def index(request):

return HttpResponse("Hello, world. You're at the poll index.")

def detail(request, poll\_id):

return HttpResponse("You're looking at poll %s." % poll\_id)

def results(request, poll\_id):

return HttpResponse("You're looking at the results of poll %s." % poll\_id)

def vote(request, poll\_id):

return HttpResponse("You're voting on poll %s." % poll\_id)

Každé view vrací objekt HttpResponse, nebo výjimku (např. Http404). Příklad jednoduchého view, které něco dělá:

from django.http import HttpResponse

from myproject.myapp.models import Poll

# zobrazi poslednich 5 serazenych otazek z ankety

def index(request):

latest\_poll\_list = Poll.objects.all().order\_by('-pub\_date')[:5]

output = ', '.join([p.question for p in latest\_poll\_list])

return HttpResponse(output)

View s použitím HTML šablony [1]:

from django.template import Context, loader

from django.http import HttpResponse

from myproject.myapp.models import Poll

# zobrazi poslednich 5 serazenych otazek z ankety v HTML sablone

def index(request):

latest\_poll\_list = Poll.objects.all().order\_by('-pub\_date')[:5]

t = loader.get\_template('myapp/index.html')

c = Context({

'latest\_poll\_list': latest\_poll\_list,

})

return HttpResponse(t.render(c))

View s použitím HTML šablony a funkce render\_to\_response():

from django.shortcuts import render\_to\_response

from myproject.myapp.models import Poll

# zobrazi poslednich 5 serazenych otazek z ankety v HTML sablone

def index(request):

latest\_poll\_list = Poll.objects.all().order\_by('-pub\_date')[:5]

return render\_to\_response('myapp/index.html', {'latest\_poll\_list': latest\_poll\_list})

View s voláním chyby 404 [2]:

from django.shortcuts import render\_to\_response

from django.http import Http404

from myproject.myapp.models import Poll

# zobrazi detail ankety s osetrenim na chybu 404

def detail(request, poll\_id):

try:

p = Poll.objects.get(pk=poll\_id)

except Poll.DoesNotExist:

raise Http404

return render\_to\_response('myapp/detail.html', {'poll': p})

View s voláním chyby 404 a funkce get\_object\_or\_404(), která přijímá jako první argument model Djanga, a na dalších pozicích libovolný počet argumentů, které se předají funkci get(). Pokud se podle dodaných parametrů objekt nenalezne, vyvolá se vyjímka Http404. Existuje také funkce get\_list\_or\_404(), která používá metodu filter(). Funkce vyvolá výjimku Http404 pokud je seznam prázdný.

from django.shortcuts import render\_to\_response, get\_object\_or\_404

from myproject.myapp.models import Poll

# zobrazi detail ankety s osetrenim na chybu 404

def detail(request, poll\_id):

p = get\_object\_or\_404(Poll, pk=poll\_id)

return render\_to\_response('myapp/detail.html', {'poll': p})

View pro zpracování formuláře [3]. Z objektu request.POST můžeme vytahovat odeslaná data s pomocí jejich jména. Hodnoty z request.POST se vrací vždy v podobě řetězce. Podobně funguje i request.GET. Funkce reverse() přesměrovává na určité view.

from django.shortcuts import get\_object\_or\_404, render\_to\_response

from django.http import HttpResponseRedirect, HttpResponse

from django.core.urlresolvers import reverse

from django.template import RequestContext

from myproject.myapp.models import Choice, Poll

def vote(request, poll\_id):

p = get\_object\_or\_404(Poll, pk=poll\_id)

try:

selected\_choice = p.choice\_set.get(pk=request.POST['choice'])

except (KeyError, Choice.DoesNotExist):

# zobrazime formular znovu, tentokrat s chybovou hlaskou

return render\_to\_response('myapp/detail.html', {

'poll': p,

'error\_message': "Nevybral(-a) jste zadnou anketni volbu.",

}, context\_instance=RequestContext(request))

else:

selected\_choice.votes += 1

selected\_choice.save()

# Po kazdem uspesnem zpracovani POST dat je treba vyvolat

# HttpResponseRedirect. Tato praktika zabrani dvojimu

# odeslani (a uzlozeni) dat z formulare v situacich, kdy

# uzivatel klikne na tlacitko "Zpet" ve svem webovem

# prohlizeci.

return HttpResponseRedirect(reverse('myproject.myapp.views.results', args=(p.id,)))

View s použitím HTML šablony a funkce render\_to\_response() [4]:

from django.shortcuts import get\_object\_or\_404, render\_to\_response

from myproject.myapp.models import Choice, Poll

def results(request, poll\_id):

p = get\_object\_or\_404(Poll, pk=poll\_id)

return render\_to\_response('myapp/results.html', {'poll': p})

## Šablony

Cesta k šablonám se nastavuje v souboru settings.py v proměnné TEMPLATE\_DIRS. Každá aplikace má svůj adresář se šablonami. Příklad HTML šablony myapp/index.html [1]:

{% if latest\_poll\_list %}

<ul>

{% for poll in latest\_poll\_list %}

<li>{{ poll.question }}</li>

{% endfor %}

</ul>

{% else %}

<p>No polls are available.</p>

{% endif %}

Příklad HTML šablony myapp/detail.html [2]:

<h1>{{ poll.question }}</h1>

<ul>

{% for choice in poll.choice\_set.all %}

<li>{{ choice.choice }}</li>

{% endfor %}

</ul>

Šablona pro chybu 404 se ukládá do kořenového adresáře se šablonami do souboru 404.html. View 404 se vyvolá, pokud není nalezena shoda v regulárních výrazech URLconf. Lze použít výchozí view 404. V Debug režimu se view 404 nepoužije. Zobrazuje se traceback. Stejným způsobem lze definovat šablonu pro chybu 500 (chyba serveru). Ta je vyvolána v případě runtime chyby. Šablona se ukládá do souboru 500.html.

Příklad HTML šablony myapp/results.html [4]:

<h1>{{ poll.question }}</h1>

<ul>

{% for choice in poll.choice\_set.all %}

<li>{{ choice.choice }} - {{ choice.votes }} vote{{ choice.votes|pluralize }}</li>

{% endfor %}

</ul>

## Formuláře

Příklad HTML formuláře myapp/detail.html. Po odeslání formuláře se zavolá view Vote [3].

<h1>{{ poll.question }}</h1>

{% if error\_message %}<p><strong>{{ error\_message }}</strong></p>{% endif %}

<form action="/polls/{{ poll.id }}/vote/" method="post">

{% csrf\_token %}

{% for choice in poll.choice\_set.all %}

<input type="radio" name="choice" id="choice{{ forloop.counter }}" value="{{ choice.id }}" />

<label for="choice{{ forloop.counter }}">{{ choice.choice }}</label><br />

{% endfor %}

<input type="submit" value="Vote" />

</form>

# Databázové API

>>> from myproject.myapp.models import Poll, Choice

>>> import datetime

# vypise seznam vsech anket modelu Poll

>>> Poll.objects.all()

# vytvoreni noveho zaznamu ankety

>>> p = Poll(question="Tak co?", pub\_date=datetime.datetime.now())

>>> p.save()

# vypis jednotlivych poli modelu Poll

>>> p.id

>>> p.question

>>> p.pub\_date

# uprava pole

>>> p.pub\_date = datetime.datetime(2007, 4, 1, 0, 0)

>>> p.save()

# vyhledavani v databazi

>>> Poll.objects.filter(id=1)

>>> Poll.objects.filter(question\_\_startswith='Tak')

>>> Poll.objects.get(pub\_date\_\_year=2007)

>>> Poll.objects.get(id=1)

# vyhledavani podle primarniho klice a volani vlastni metody

>>> p = Poll.objects.get(pk=1)

>>> p.was\_published\_today()

# vytvoreni noveho zaznamu odpovedi, pridani do sady odpovedi a vlozeni do databaze

>>> c = p.choice\_set.create(choice='Just hacking again', votes=0)

# pristup k souvisejicim objektum

>>> c.poll

>>> p.choice\_set.all()

>>> p.choice\_set.count()

# pokrocile filtrovani

>>> Choice.objects.filter(poll\_\_pub\_date\_\_year=2007)

# smazani zaznamu

>>> c.delete()

# Administrační rozhraní

Pro aktivaci administračního rozhraní je potřeba přidat 'django.contrib.admin' do seznamu INSTALLED\_APPS, provést synchronizaci databáze a správně upravit urls.py. Admin pak běží na adrese <http://127.0.0.1:8000/admin/>. Pro přidání správy aplikace myapp do administračního rozhraní je potřeba vytvořit soubor admin.py v adresáři aplikace.

from myproject.myapp.models import Poll

from django.contrib import admin

admin.site.register(Poll) #pridani modelu Poll

## Vzhled administračních formulářů

Formuláře jsou automaticky generovány podle modelů. Zobrazení administračních formulářů lze ovlivňovat. Prohození políček formuláře:

# administratorsky objekt

class PollAdmin(admin.ModelAdmin):

fields = ['pub\_date', 'question']

admin.site.register(Poll, PollAdmin) #pridani modelu Poll

Formuláře lze rozdělit na skupiny. Libovolnému poli lze přiřadit HTML třídu (class), např. collapse pro zabalení skupiny:

# administratorsky objekt

class PollAdmin(admin.ModelAdmin):

fieldsets = [

(None, {'fields': ['question']}),

('Date information', {'fields': ['pub\_date'], 'classes': ['collapse']}),

]

admin.site.register(Poll, PollAdmin) #pridani modelu Poll

Zobrazení jednotlivých polí v libovolném pořadí ve výpisu všech záznamů modelu (change list):

list\_display = ('question', 'pub\_date', 'was\_published\_today')

Filtr ve výpisu záznamů modelu:

list\_filter = ['pub\_date']

Vyhledávání ve výpisu záznamů modelu:

search\_fields = ['question']

Hierarchická navigace podle data:

date\_hierarchy = 'pub\_date'

## Přidání souvisejících objektů

Přidání souvisejícího objektu Choice lze provést pomocí funkce admin.site.register(), nebo můžeme související objekt Choice spravovat přímo v administraci Poll. Pro zobrazení souvisejícího modelu můžeme využít řádkového zobrazení admin.StackedInline nebo tabulkového zobrazení admin.TabularInline.

from myproject.myapp.models import Poll

from myproject.myapp.models import Choice

from django.contrib import admin

# radkove zobrazeni souvisejiciho modelu

class ChoiceInline(admin.StackedInline):

model = Choice #objekt modelu

extra = 3 #zobrazi 3 pole

# administratorsky objekt

class PollAdmin(admin.ModelAdmin):

inlines = [ChoiceInline] #pridani souvisejiciho objektu

admin.site.register(Poll, PollAdmin) #pridani modelu Poll

admin.site.register(Choice) #pridani modelu Choice

## Úprava vzhledu prostředí

Pro úpravu vzhledu administračního prostředí si vytvoříme vlastní šablony. Cestu k šablonám musíme nastavit v souboru settings.py v proměnné TEMPLATE\_DIRS:

TEMPLATE\_DIRS = (

"D:/DJANGO/myproject/templates ", #adresar se sablonami

)

Zkopírujeme například /Python26/Lib/site-packages/django/contrib/admin/templates/base\_site.html do D:/DJANGO/myproject/templates/admin/base\_site.html. Nyní můžeme šablonu libovolně upravovat.