Django 1.6 最佳实践: 如何设置django项目的设置(settings.py)和部署文件(requirements.txt)

在Django 1.6中的settings.py中可以修改130多项设置, 但大多数都继承自默认值. 设置是在web服务器启动时首次载入的, 服务器重启时重新载入, 因此, 程序员们应尽量避免修改正式服务器上使用的settings.py文件. 以下是一些我们应当遵循的原则:

所有的设置文件应当进行版本管理

不要重复自己 (don't repeat yourself)

妥善保存关键信息

之前, 我们采用的方法是, 不将设置文件放入git库中, 每个开发人员本地有一份自己的设置文件. 但我们发现这样做是错误的. 因为:

在debug之前, 我们可能已经花费了大量精力去模拟正式服务器上出现的错误, 但最终发现这是由于正式服务器的settings文件设置和本地不同而 出现的问题. 这时你的心情会是怎样?

当你在开发django项目时, 发现并修复了一个bug. 当将这一commit push到服务器后, 你突然发现这一bug的出现完全是因为你修改了本地的 settings文件而产生的, 而由于你的push, 又导致了服务器的宕机. 这时你又会是怎样的感受?

每个人都会从另一个程序员那里拷贝/黏贴settings文件内容, 这难道不是违反了"不要重复自己"的原则吗?

正式由于这些问题, 所以我们现在采用不一样的设置方式. 我们首先建立一个基本的设置文件, 然后将开发和正式部署的设置文件分离成不同的模块, 但 这些模块都继承自同一个基本设置文件:

1. 使用分离式的设置文件

django项目建立时, 会自动生成settings.py文件. 为了实现分离式的设置文件, 我们首先删除settings.py, 然后建立settings目录:

settings/

\_\_init\_\_.py

base.py

local.py

test.py

production.py

...

设置文件 目的

base.py 基本设置文件, 在各个环境中都相同的设置可以放入其中.

local.py 当在开发时使用的设置文件. 可以设置开发时的选项, 包括DEBUG, log的等级, 是否开启例如 django-debug-toolbar等开发工具等.

test.py 运行test时的配置, 包括test runners, in-memory数据库定义和log设置等.

production.py 当部署到正式服务器上所用的设置.

我们可以使用以下命令使用这些不同的设置文件:

python manage.py shell --settings=mysite.settings.local

python manage.py runserver --settings=mysite.settings.local

当然如果你熟悉 DJANGO\_SETTINGS\_MODULE 和 PYTHONPATH 的话, 也可以事先设置好 DJANGO\_SETTINGS\_MODULE 和 PYTHONPATH 环境变量, 这样做的好处就是你不必使用--settings了.

如果你对virtualenv有深入的了解的话, 也可以在postactivate脚本中设置 DJANGO\_SETTINGS\_MODULE 和 PYTHONPATH.

local.py的例子

# settings/local.py

from .base import \*

DEBUG = True

TEMPLATE\_DEBUG = DEBUG

EMAIL\_BACKEND = 'django.core.email.backends.console.EmailBackend'

DATABASES = {

"defaults": {

"ENGINE": "django.db.backends.postgresql\_psycopg2",

"NAME": "weiguda",

"USER": "",

"PASSWORD": "",

"HOST": "localhost",

"PORT": "",

}

}

INSTALLED\_APPS += ("debug\_toolbar",)

有时, 一个开发人员的配置文件可能与另一个不同, 这时, 我们可以在settings目录中新建local\_name.py:

# settings/local\_name.py

from .local import \*

# 设置不同的配置

CACHE\_TIMEOUT = 30

2. 将关键信息和设置文件分离

将SECTET\_KEY, AWS key文件, API key文件等关键信息放入设置文件中也是违反基本原则的. 因为:

配置环境不同时关键信息会改变, 程序却不会.

关键信息不是程序.

关键信息应当是隐蔽的, 如果储存在了版本管理系统中, 则任何有权访问该版本库的用户都能获知这些关键信息.

许多PAAS服务无法为每台服务器编辑设置文件, 即使可以, 这也是不正确的做法.

环境变量

为了避免以上的问题, 我们使用环境变量 (environment variables) 来储存这些关键信息, (需要注意的是, apache不支持环境变量, 我们会在下面讲到). 使用环境变量储存关键信息有以下好处:

将关键信息从代码中移除, 这样你就可以安心的将所有文件放入版本管理系统中.

每个开发人员都拥有一样的local.py文件.

在部署django项目时, 不需要修改程序代码.

大多数PAAS都推荐这一方法, 并提供了方便的设置方法, 因此容易实现.

设置环境变量

在使用bash的Mac或Linux中设置环境变量比较容易, 你只需要将以下代码加入.bashrc, .bash\_profile, 或.profile其中之一即可. 如果多个项目 使用相同的API, 并且关键信息都不同时, 可以将以下代码加入到virtualenv的bin/activate脚本中:

$ export SOME\_SECRET\_KEY=654-3jgwg-4r3-2t4h-76jk

$ export ANOTHER\_SECRET\_KEY=y5y-5jk8-75i5h-5g4/.-,o.

如果使用的是windows系统, 则设置稍微复杂一点. 如果使用cmd.exe, 你必须使用setx命令一个一个的设置, 一个较好的方式是使用virtualenv的 bin/activate.bat

> setx OME\_SECRET\_KEY=654-3jgwg-4r3-2t4h-76jk

PowerShell是Windows Vista及以上自带的shell, 它比cmd.exe强大得多. 因此可以使用PowerShell来设置环境变量:

# 为用户User设置

[Environment]::SetEnvironmentVariable("SOME\_SECRET\_KEY", "654-3jgwg-4r3-2t4h-76jk", "User")

# 为全局设置

[Environment]::SetEnvironmentVariable("SOME\_SECRET\_KEY", "654-3jgwg-4r3-2t4h-76jk", "Machine")

如果你使用virtuanenvwrapper, 那么可以使用virtualenvwrapper的pre-virtualenv设置环境变量, 这样可能会更方便.

如果你使用PAAS, 则请参阅不同的PAAS提供的设置方法.

获取环境变量

在设置好环境变量后, 我们来看如何在django的settings代码中获取这些关键信息:

# 在settings/production.py顶部

import os

SOME\_SECRET\_KEY = os.environ["SOME\_SECRET\_KEY"]

在以上代码中, 如果SOME\_SECRET\_KEY无法被获取到的话, 就会出现KeyError错误, 导致django项目无法启动. 这很好, 但KeyError没有提供更有 用的信息, 导致debug的困难, 因此, 我们在base.py(希望你还记得这是哪个文件)加入以下function, 为我们提供哪个关键信息无法获取的信息:

# settings/base.py

import os

# 通常你不应该从django引入任何代码, 但ImproperlyConfigured是个例外

from django.core.exceptions import ImproperlyConfigured

def get\_env\_variable(var\_name):

try:

return os.environ[var\_name]

except KeyError:

error\_msg = "Set the %s environment variable" % var\_name

raise ImproperlyConfigured('error\_msg')

然后修改之前的production.py:

# 在settings/production.py顶部

import os

SOME\_SECRET\_KEY = get\_env\_variable('SOME\_SECRET\_KEY')

此时, 当你没有设置SOME\_SECRET\_KEY环境变量时, 系统会提示错误信息, 告诉你是哪个环境变量没有设置.

无法使用环境变量时

当我们使用apache时, 我们会发现, django无法使用环境变量. 这时, 我们推荐将关键信息储存在JSON格式的文件中, 已达到将关键信息和代码分离的 目的. 首先我们可以创建secrets.json文件:

{

"FILENAME": "secrets.json",

"SOME\_SECRET\_KEY": "654-3jgwg-4r3-2t4h-76jk"

}

在settings中使用该文件:

# settings/base.py

import json

# 通常你不应该从django引入任何代码, 但ImproperlyConfigured是个例外

from django.core.exceptions import ImproperlyConfigured

# 读取json文件

with open("secrets.json") as f:

secrets = json.loads(f.read())

def get\_secret(setting, secrets=secrets):

try:

return secrets[setting]

except KeyError:

error\_msg = "Set the {0} environment variable".format(setting)

raise ImproperlyConfigured('error\_msg')

SOME\_SECRET\_KEY = get\_secret('SOME\_SECRET\_KEY')

3. 使用不同的部署文件(requirements.txt)

部署文件(requirements.txt)中储存的是该django项目的依赖库, 一般使用pip freeze --local生成. 本着"只安装需要的模块"的原则, 不同的设 置文件, 应当对应不同的requirements.txt文件. 就像分离式的settings文件一样, 我们使用分离式的requirements文件. 首先我们删除前文中提到的repository\_root中的requirements.txt, 然后建立requirements目录:

requirements/

base.txt

local.txt

production.txt

在base.txt中, 储存的是所有开发环境中都会用到的依赖库, 例如:

Django==1.6.5

psycopg2==2.5.3

South==0.8.4

在local.txt中, 储存的是本地开发时用到的依赖库:

# 导入base.txt中的依赖库

-r base.txt

coverage==3.7.1

django-debug-toolbar==1.2

当重新配置本地开发环境时, 可以使用以下代码安装依赖库:

pip install -r requirements/local.txt

4. 设置文件中的文件路径

我们强烈反对将绝对路径写入设置文件中, 因为如果将绝对路径写入设置文件中的话, 遇到不同的环境, 就需要重新修改, 给开发和部署带来了许多麻烦 和不确定性.

我们推荐使用Unipath来设置media, static和templates的 路径. 这样无论部署测试环境如何变化, media和templates文件夹的设置都不会有问题.

# settings/base.py 顶部

from unipath import Path

BASE\_DIR = Path(\_\_file\_\_).ancestor(3)

MEDIA\_ROOT = BASE\_DIR.child("media")

STAIC\_ROOT = BASE\_DIR.child("static")

TEMPLATES\_DIRS = (

BASE\_DIR.child("templates"),

)