Day 12 - 编写日志列表页

41次阅读

MVVM模式不但可用于Form表单，在复杂的管理页面中也能大显身手。例如，分页显示Blog的功能，我们先把后端代码写出来：

在apis.py中定义一个Page类用于存储分页信息：

```
class Page(object):
    def __init__(self, item_count, page_index=1, page_size=10):
        self.item_count = item_count
        self.page_size = page_size
        self.page_count = item_count // page_size + (1 if item_count % page_size > 0 else 0)
        if (item_count == 0) or (page_index < 1) or (page_index > self.page_count):
            self.offset = 0
            self.limit = 0
            self.page_index = 1
        else:
            self.page_index = page_index
            self.offset = self.page_size * (page_index - 1)
            self.limit = self.page_size
        self.has_next = self.page_index < self.page_count
        self.has_previous = self.page_index > 1
```

在urls.py中实现API：

```
def _get_blogs_by_page():
    total = Blog.count_all()
    page = Page(total, _get_page_index())
    blogs = Blog.find_by('order by created_at desc limit ?,?', page.offset, page.limit)
    return blogs, page

@api
@get('/api/blogs')
def api_get_blogs():
    blogs, page = _get_blogs_by_page()
    return dict(blogs=blogs, page=page)
```

返回模板页面：

```
@view('manage_blog_list.html')
@get('/manage/blogs')
def manage_blogs():
    return dict(page_index=_get_page_index(), user=ctx.request.user)
```

模板页面首先通过API：GET /api/blogs?page=?拿到Model：

```
{
    "page": {
        "has_next": true,
        "page_index": 1,
        "page_count": 2,
        "has_previous": false,
        "item_count": 12
    },
    "blogs": [...]
}
```

然后，通过Vue初始化MVVM：

```
<script>
function initVM(data) {
    $('#div-blogs').show();
    var vm = new Vue({
        el: '#div-blogs',
        data: {
            blogs: data.blogs,
            page: data.page
        },
        methods: {
            previous: function () {
                gotoPage(this.page.page_index - 1);
            },
            next: function () {
                gotoPage(this.page.page_index + 1);
            },
            edit_blog: function (blog) {
```

```
                location.assign('/manage/blogs/edit/' + blog.id);
            }
        }
    });
}

$(function() {
    getApi('/api/blogs?page={{ page_index }}', function (err, results) {
        if (err) {
            return showError(err);
        }
        $('#div-loading').hide();
        initVM(results);
    });
});
</script>
```

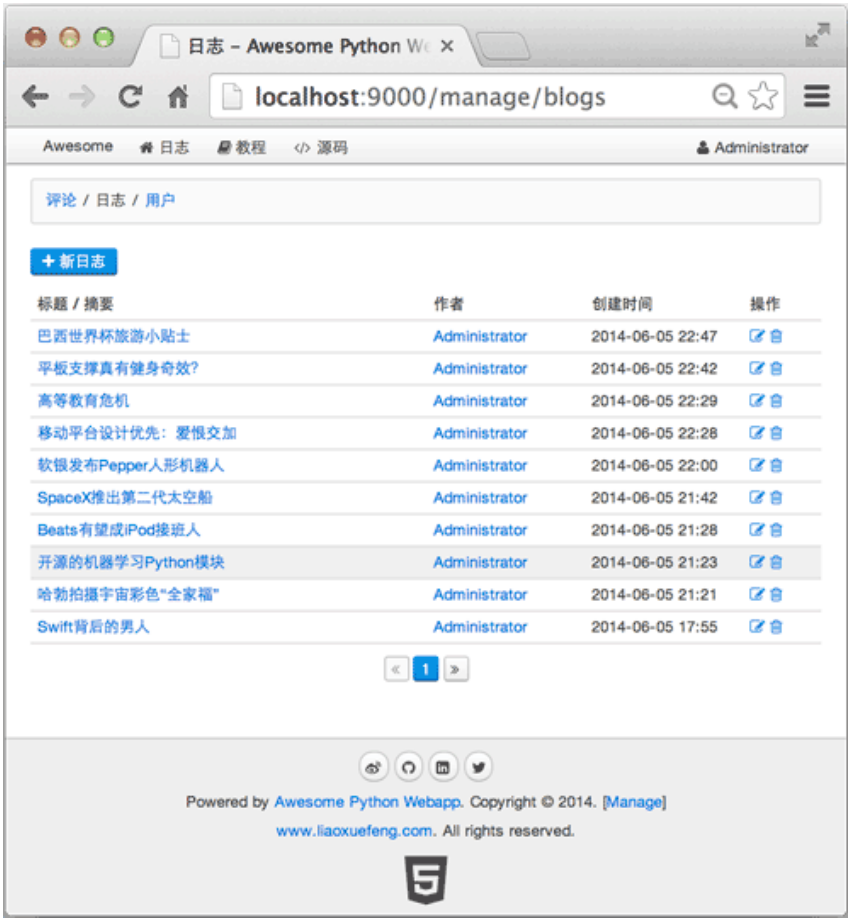
View的容器是#div-blogs，包含一个table，我们用v-repeat可以把Model的数组blogs直接变成多行的<tr>：

```
<div id="div-blogs" class="uk-width-1-1" style="display:none">
    <table class="uk-table uk-table-hover">
        <thead>
            <tr>
                <th class="uk-width-5-10">标题 / 摘要</th>
                <th class="uk-width-2-10">作者</th>
                <th class="uk-width-2-10">创建时间</th>
                <th class="uk-width-1-10">操作</th>
            </tr>
        </thead>
        <tbody>
            <tr v-repeat="blog: blogs" >
                <td>
                    <a target="_blank" v-attr="href: '/blog/'+blog.id" v-text="blog.name"></a>
                </td>
                <td>
                    <a target="_blank" v-attr="href: '/user/'+blog.user_id" v-text="blog.user_name"></a>
                </td>
                <td>
                    <span v-text="blog.created_at.toDateTime()"></span>
                </td>
                <td>
                    <a href="#0" v-on="click: edit_blog(blog)"><i class="uk-icon-edit"></i>
                </td>
            </tr>
        </tbody>
    </table>
    <div class="uk-width-1-1 uk-text-center">
        <ul class="uk-pagination">
            <li v-if="! page.has_previous" class="uk-disabled"><span><i class="uk-icon-angle-double-left"></i></span></li>
            <li v-if="page.has_previous"><a v-on="click: previous()" href="#0"><i class="uk-icon-angle-double-left"></i></a></li>
            <li class="uk-active"><span v-text="page.page_index"></span></li>
            <li v-if="! page.has_next" class="uk-disabled"><span><i class="uk-icon-angle-double-right"></i></span></li>
            <li v-if="page.has_next"><a v-on="click: next()" href="#0"><i class="uk-icon-angle-double-right"></i></a></li>
        </ul>
    </div>
</div>
```

往Model的blogs数组中增加一个Blog元素，table就神奇地增加了一行；把blogs数组的某个元素删除，table就神奇地减少了一行。所有复杂的Model-View的映射逻辑全部由MVVM框架完成，我们只需要在HTML中写上v-repeat指令，就什么都不用管了。

可以把v-repeat="blog: blogs"看成循环代码，所以，可以在一个<tr>内部引用循环变量blog。v-text和v-attr指令分别用于生成文本和DOM节点属性。

完整的Blog列表页如下：


日志 - Awesome Python W
localhost:9000/manage/blogs
Awesome
日志
教程
源码
Administrator
评论 / 日志 / 用户
+ 新日志
标题 / 摘要
作者
创建时间
操作
巴西世界杯旅游小贴士 Administrator 2014-06-05 22:47
平板支撑真有健身奇效? Administrator 2014-06-05 22:42
高等教育危机 Administrator 2014-06-05 22:29
移动平台设计优先：爱恨交加 Administrator 2014-06-05 22:28
软银发布Pepper人形机器人 Administrator 2014-06-05 22:00
SpaceX推出第二代太空船 Administrator 2014-06-05 21:42
Beats有望成iPod接班人 Administrator 2014-06-05 21:28
开源的机器学习Python模块 Administrator 2014-06-05 21:23
哈勃拍摄宇宙彩色"全家福" Administrator 2014-06-05 21:21
Swift背后的男人 Administrator 2014-06-05 17:55
« 1 »
Powered by Awesome Python Webapp. Copyright © 2014. [Manage]
www.liaoxuefeng.com. All rights reserved.

Day 13 - 提升开发效率

83次阅读

现在，我们已经把一个Web App的框架完全搭建好了，从后端的API到前端的MVVM，流程已经跑通了。

在继续工作前，注意到每次修改Python代码，都必须在命令行先Ctrl-C停止服务器，再重启，改动才能生效。

在开发阶段，每天都要修改、保存几十次代码，每次保存都手动来这么一下非常麻烦，严重地降低了我们的开发效率。有没有办法让服务器检测到代码修改后自动重新加载呢？

Django的开发环境在Debug模式下就可以做到自动重新加载，如果我们编写的服务器也能实现这个功能，就能大大提升开发效率。

可惜的是，Django没把这个功能独立出来，不用Django就享受不到，怎么办？

其实Python本身提供了重新载入模块的功能，但不是所有模块都能被重新载入。另一种思路是检测www目录下的代码改动，一旦有改动，就自动重启服务器。

按照这个思路，我们可以编写一个辅助程序pymonitor.py，让它启动wsgiapp.py，并时刻监控www目录下的代码改动，有改动时，先把当前wsgiapp.py进程杀掉，再重启，就完成了服务器进程的自动重启。

要监控目录文件的变化，我们也无需自己手动定时扫描，Python的第三方库watchdog可以利用操作系统的API来监控目录文件的变化，并发送通知。我们先用easy_install安装：

```
$ easy_install watchdog
```

利用watchdog接收文件变化的通知，如果是.py文件，就自动重启wsgiapp.py进程。

利用Python自带的subprocess实现进程的启动和终止，并把输入输出重定向到当前进程的输入输出中：

```
#!/usr/bin/env python
import os, sys, time, subprocess

from watchdog.observers import Observer
from watchdog.events import FileSystemEventHandler

def log(s):
    print '[Monitor] %s' % s

class MyFileSystemEventHander(FileSystemEventHandler):
    def __init__(self, fn):
        super(MyFileSystemEventHander, self).__init__()
        self.restart = fn

    def on_any_event(self, event):
        if event.src_path.endswith('.py'):
            log('Python source file changed: %s' % event.src_path)
            self.restart()

command = ['echo', 'ok']
process = None

def kill_process():
```

```
    global process
    if process:
        log('Kill process [%s]...' % process.pid)
        process.kill()
        process.wait()
        log('Process ended with code %s.' % process.returncode)
        process = None

def start_process():
    global process, command
    log('Start process %s...' % ' '.join(command))
    process = subprocess.Popen(command, stdin=sys.stdin, stdout=sys.stdout, stderr=sys.stderr)

def restart_process():
    kill_process()
    start_process()

def start_watch(path, callback):
    observer = Observer()
    observer.schedule(MyFileSystemEventHander(restart_process), path, recursive=True)
    observer.start()
    log('Watching directory %s...' % path)
    start_process()
    try:
        while True:
            time.sleep(0.5)
    except KeyboardInterrupt:
        observer.stop()
    observer.join()

if __name__ == '__main__':
    argv = sys.argv[1:]
    if not argv:
        print('Usage: ./pymonitor your-script.py')
        exit(0)
    if argv[0]!='python':
        argv.insert(0, 'python')
    command = argv
    path = os.path.abspath('.')
    start_watch(path, None)
```

一共50行左右的代码，就实现了Debug模式的自动重新加载。用下面的命令启动服务器：

```
$ python pymonitor.py wsgiapp.py
```

或者给pymonitor.py加上可执行权限，启动服务器：

```
$ ./pymonitor.py wsgiapp.py
```

在编辑器中打开一个py文件，修改后保存，看看命令行输出，是不是自动重启了服务器：

```
$ ./pymonitor.py wsgiapp.py
[Monitor] Watching directory /Users/michael/Github/awesome-python-webapp/www...
[Monitor] Start process python wsgiapp.py...
...
INFO:root:application (/Users/michael/Github/awesome-python-webapp/www) will start at 0.0.0.0:9000...
[Monitor] Python source file changed: /Users/michael/Github/awesome-python-webapp/www/apis.py
[Monitor] Kill process [2747]...
[Monitor] Process ended with code -9.
[Monitor] Start process python wsgiapp.py...
...
INFO:root:application (/Users/michael/Github/awesome-python-webapp/www) will start at 0.0.0.0:9000...
```

现在，只要一保存代码，就可以刷新浏览器看到效果，大大提升了开发效率。

Day 14 - 完成Web App

78次阅读

在Web App框架和基本流程跑通后，剩下的工作全部是体力活了：在Debug开发模式下完成后端所有API、前端所有页面。我们需要做的事情包括：

对URL/manage/进行拦截，检查当前用户是否是管理员身份：

```
@interceptor('/manage/')
def manage_interceptor(next):
    user = ctx.request.user
    if user and user.admin:
        return next()
    raise seeother('/signin')
```

后端API包括：

- 获取日志：GET /api/blogs
- 创建日志：POST /api/blogs
- 修改日志：POST /api/blogs/:blog_id
- 删除日志：POST /api/blogs/:blog_id/delete
- 获取评论：GET /api/comments
- 创建评论：POST /api/blogs/:blog_id/comments
- 删除评论：POST /api/comments/:comment_id/delete
- 创建新用户：POST /api/users
- 获取用户：GET /api/users

管理页面包括：

- 评论列表页：GET /manage/comments
- 日志列表页：GET /manage/blogs
- 创建日志页：GET /manage/blogs/create
- 修改日志页：GET /manage/blogs/
- 用户列表页：GET /manage/users

用户浏览页面包括：

- 注册页：GET /register
- 登录页：GET /signin
- 注销页：GET /signout

- 首页：GET /
- 日志详情页：GET /blog/:blog_id

把所有的功能实现，我们第一个Web App就宣告完成！

Day 15 - 部署Web App

90次阅读

作为一个合格的开发者，在本地环境下完成开发还远远不够，我们需要把Web App部署到远程服务器上，这样，广大用户才能访问到网站。

很多做开发的同学把部署这件事情看成是运维同学的工作，这种看法是完全错误的。首先，最近流行DevOps理念，就是说，开发和运维要变成一个整体。其次，运维的难度，其实跟开发质量有很大的关系。代码写得垃圾，运维再好也架不住天天挂掉。最后，DevOps理念需要把运维、监控等功能融入到开发中。你想服务器升级时不中断用户服务？那就得在开发时考虑到这一点。

下面，我们就来把awesome-python-webapp部署到Linux服务器。

搭建Linux服务器

要部署到Linux，首先得有一台Linux服务器。要在公网上体验的同学，可以在Amazon的AWS申请一台EC2虚拟机（免费使用1年），或者使用国内的一些云服务器，一般都提供Ubuntu Server的镜像。想在本地部署的同学，请安装虚拟机，推荐使用VirtualBox。

我们选择的Linux服务器版本是Ubuntu Server 12.04 LTS，原因是apt太简单了。如果你准备使用其他Linux版本，也没有问题。

Linux安装完成后，请确保ssh服务正在运行，否则，需要通过apt安装：

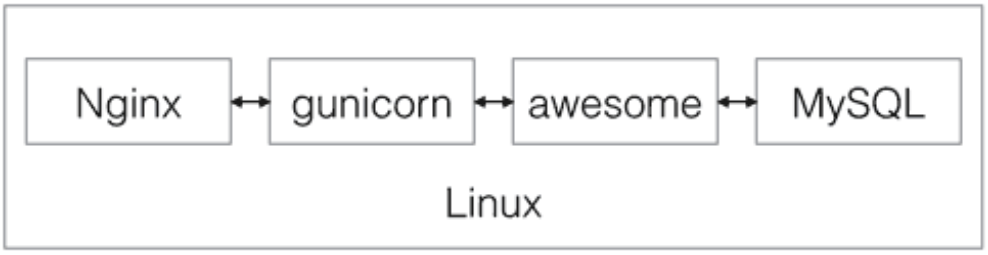
```
$ sudo apt-get openssh-server
```

有了ssh服务，就可以从本地连接到服务器上。建议把公钥复制到服务器端用户的.ssh/authorized_keys中，这样，就可以通过证书实现无密码连接。

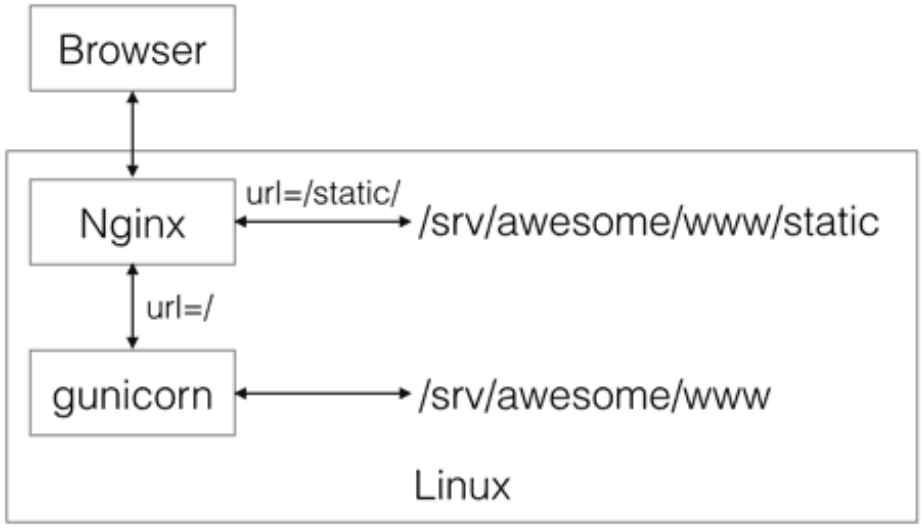
部署方式

在本地开发时，我们可以用Python自带的WSGI服务器，但是，在服务器上，显然不能用自带的这个开发版服务器。可以选择的WSGI服务器很多，我们选gunicorn：它用类似Nginx的Master-Worker模式，同时可以提供gevent的支持，不用修改代码，就能获得极高的性能。

此外，我们还需要一个高性能Web服务器，这里选择Nginx，它可以处理静态资源，同时作为反向代理把动态请求交给gunicorn处理。gunicorn负责调用我们的Python代码，这个模型如下：



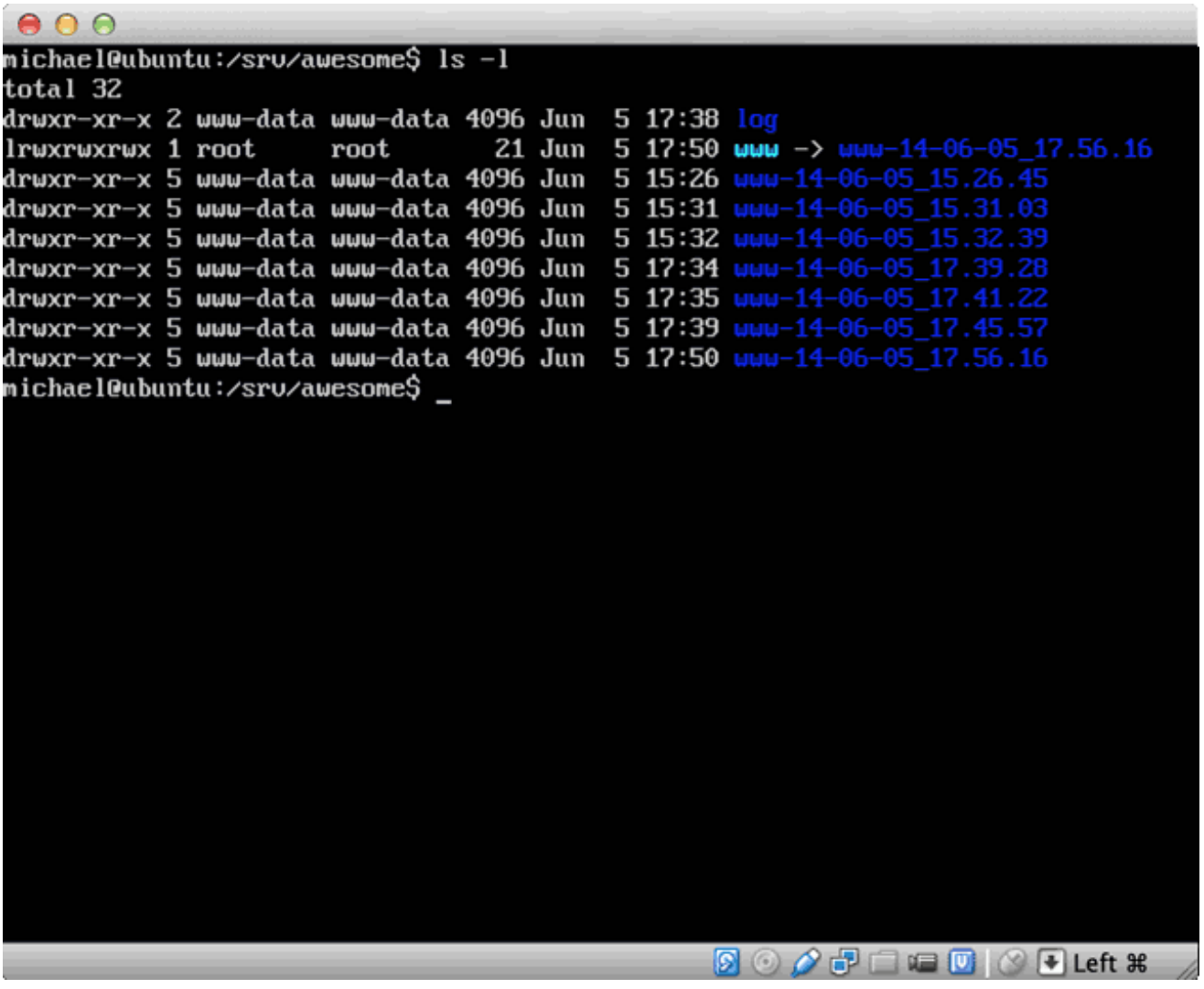
Nginx负责分发请求：



在服务器端，我们需要定义好部署的目录结构：

```
/
+- srv/
   +- awesome/       <-- Web App根目录
      +- www/        <-- 存放Python源码
      |  +- static/  <-- 存放静态资源文件
      +- log/        <-- 存放log
```

在服务器上部署，要考虑到新版本如果运行不正常，需要回退到旧版本时怎么办。每次用新的代码覆盖掉旧的文件是不行的，需要一个类似版本控制的机制。由于Linux系统提供了软链接功能，所以，我们把www作为一个软链接，它指向哪个目录，哪个目录就是当前运行的版本：



而Nginx和gunicorn的配置文件只需要指向www目录即可。

Nginx可以作为服务进程直接启动，但gunicorn还不行，所以，Supervisor登场！Supervisor是一个管理进程的工具，可以随系统启动而启动服务，它还时刻监控服务进程，如果服务进程意外退出，Supervisor可以自动重启服务。

总结一下我们需要用到的服务有：

- Nginx：高性能Web服务器+负责反向代理；
- gunicorn：高性能WSGI服务器；
- gevent：把Python同步代码变成异步协程的库；
- Supervisor：监控服务进程的工具；
- MySQL：数据库服务。

在Linux服务器上用apt可以直接安装上述服务：

```
$ sudo apt-get install nginx gunicorn python-gevent supervisor mysql-server
```

然后，再把我们自己的Web App用到的Python库安装了：

```
$ sudo apt-get install python-jinja2 python-mysql.connector
```

在服务器上创建目录/srv/awesome/以及相应的子目录。

在服务器上初始化MySQL数据库，把数据库初始化脚本schema.sql复制到服务器上执行：

```
$ mysql -u root -p < schema.sql
```

服务器端准备就绪。

部署

用FTP还是SCP还是rsync复制文件？如果你需要手动复制，用一次两次还行，一天如果部署50次不但慢、效率低，而且容易出错。

正确的部署方式是使用工具配合脚本完成自动化部署。Fabric就是一个自动化部署工具。由于Fabric是用Python开发的，所以，部署脚本也是用Python来编写，非常方便！

要用Fabric部署，需要在本机（是开发机器，不是Linux服务器）安装Fabric：

```
$ easy_install fabric
```

Linux服务器上不需要安装Fabric，Fabric使用SSH直接登录服务器并执行部署命令。

下一步是编写部署脚本。Fabric的部署脚本叫fabfile.py，我们把它放到awesome-python-webapp的目录下，与www目录平级：

```
awesome-python-webapp/
+- fabfile.py
+- www/
+- ...
```

Fabric的脚本编写很简单，首先导入Fabric的API，设置部署时的变量：

```
# fabfile.py
import os, re
from datetime import datetime
```

```
# 导入Fabric API:
from fabric.api import *

# 服务器登录用户名:
env.user = 'michael'
# sudo用户为root:
env.sudo_user = 'root'
# 服务器地址，可以有多个，依次部署:
env.hosts = ['192.168.0.3']

# 服务器MySQL用户名和口令:
db_user = 'www-data'
db_password = 'www-data'
```

然后，每个Python函数都是一个任务。我们先编写一个打包的任务：

```
_TAR_FILE = 'dist-awesome.tar.gz'

def build():
    includes = ['static', 'templates', 'transwarp', 'favicon.ico', '*.py']
    excludes = ['test', '.*', '*.pyc', '*.pyo']
    local('rm -f dist/%s' % _TAR_FILE)
    with lcd(os.path.join(os.path.abspath('.'), 'www')):
        cmd = ['tar', '--dereference', '-czvf', '../dist/%s' % _TAR_FILE]
        cmd.extend(['--exclude=\'%s\'' % ex for ex in excludes])
        cmd.extend(includes)
        local(' '.join(cmd))
```

Fabric提供local('...')来运行本地命令，with lcd(path)可以把当前命令的目录设定为lcd()指定的目录，注意Fabric只能运行命令行命令，Windows下可能需要Cgywin环境。

在awesome-python-webapp目录下运行：

```
$ fab build
```

看看是否在dist目录下创建了dist-awesome.tar.gz的文件。

打包后，我们就可以继续编写deploy任务，把打包文件上传至服务器，解压，重置www软链接，重启相关服务：

```
_REMOTE_TMP_TAR = '/tmp/%s' % _TAR_FILE
_REMOTE_BASE_DIR = '/srv/awesome'

def deploy():
    newdir = 'www-%s' % datetime.now().strftime('%y-%m-%d_%H.%M.%S')
    # 删除已有的tar文件:
    run('rm -f %s' % _REMOTE_TMP_TAR)
    # 上传新的tar文件:
    put('dist/%s' % _TAR_FILE, _REMOTE_TMP_TAR)
    # 创建新目录:
    with cd(_REMOTE_BASE_DIR):
        sudo('mkdir %s' % newdir)
    # 解压到新目录:
    with cd('%s/%s' % (_REMOTE_BASE_DIR, newdir)):
        sudo('tar -xzvf %s' % _REMOTE_TMP_TAR)
    # 重置软链接:
    with cd(_REMOTE_BASE_DIR):
        sudo('rm -f www')
        sudo('ln -s %s www' % newdir)
        sudo('chown www-data:www-data www')
        sudo('chown -R www-data:www-data %s' % newdir)
    # 重启Python服务和nginx服务器:
    with settings(warn_only=True):
        sudo('supervisorctl stop awesome')
        sudo('supervisorctl start awesome')
```

```
        sudo('/etc/init.d/nginx reload')
```

注意run()函数执行的命令是在服务器上运行，with cd(path)和with lcd(path)类似，把当前目录在服务器端设置为cd()指定的目录。如果一个命令需要sudo权限，就不能用run()，而是用sudo()来执行。

配置Supervisor

上面让Supervisor重启gunicorn的命令会失败，因为我们还没有配置Supervisor呢。

编写一个Supervisor的配置文件awesome.conf，存放到/etc/supervisor/conf.d/目录下：

```
[program:awesome]
command     = /usr/bin/gunicorn --bind 127.0.0.1:9000 --workers 1 --worker-class gevent wsgiapp:application
directory   = /srv/awesome/www
user        = www-data
startsecs   = 3

redirect_stderr         = true
stdout_logfile_maxbytes = 50MB
stdout_logfile_backups  = 10
stdout_logfile          = /srv/awesome/log/app.log
```

配置文件通过[program:awesome]指定服务名为awesome，command指定启动gunicorn的命令行，设定gunicorn的启动端口为9000，WSGI处理函数入口为wsgiapp:application。

然后重启Supervisor后，就可以随时启动和停止Supervisor管理的服务了：

```
$ sudo supervisorctl reload
$ sudo supervisorctl start awesome
$ sudo supervisorctl status
awesome                  RUNNING    pid 1401, uptime 5:01:34
```

配置Nginx

Supervisor只负责运行gunicorn，我们还需要配置Nginx。把配置文件awesome放到/etc/nginx/sites-available/目录下：

```
server {
    listen      80; # 监听80端口

    root       /srv/awesome/www;
    access_log /srv/awesome/log/access_log;
    error_log  /srv/awesome/log/error_log;

    # server_name awesome.liaoxuefeng.com; # 配置域名

    # 处理静态文件/favicon.ico:
    location /favicon.ico {
        root /srv/awesome/www;
    }

    # 处理静态资源:
    location ~ ^\/static\/.*$ {
        root /srv/awesome/www;
    }

    # 动态请求转发到9000端口(gunicorn):
    location / {
        proxy_pass       http://127.0.0.1:9000;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header Host $host;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }
}
```

然后在/etc/nginx/sites-enabled/目录下创建软链接：

```
$ pwd
/etc/nginx/sites-enabled
$ sudo ln -s /etc/nginx/sites-available/awesome .
```

让Nginx重新加载配置文件，不出意外，我们的awesome-python-webapp应该正常运行：

```
$ sudo /etc/init.d/nginx reload
```

如果有任何错误，都可以在/srv/awesome/log下查找Nginx和App本身的log。如果Supervisor启动时报错，可以在/var/log/supervisor下查看Supervisor的log。

如果一切顺利，你可以在浏览器中访问Linux服务器上的awesome-python-webapp了：

如果在开发环境更新了代码，只需要在命令行执行：

```
$ fab build
$ fab deploy
```

自动部署完成！刷新浏览器就可以看到服务器代码更新后的效果。

友情链接

嫌国外网速慢的童鞋请移步网易和搜狐的镜像站点：

http://mirrors.163.com/

http://mirrors.sohu.com/

Day 16 - 编写移动App

112次阅读

网站部署上线后，还缺点啥呢？

在移动互联网浪潮席卷而来的今天，一个网站没有上线移动App，出门根本不好意思跟人打招呼。

所以，awesome-python-webapp必须得有一个移动App版本！

开发iPhone版本

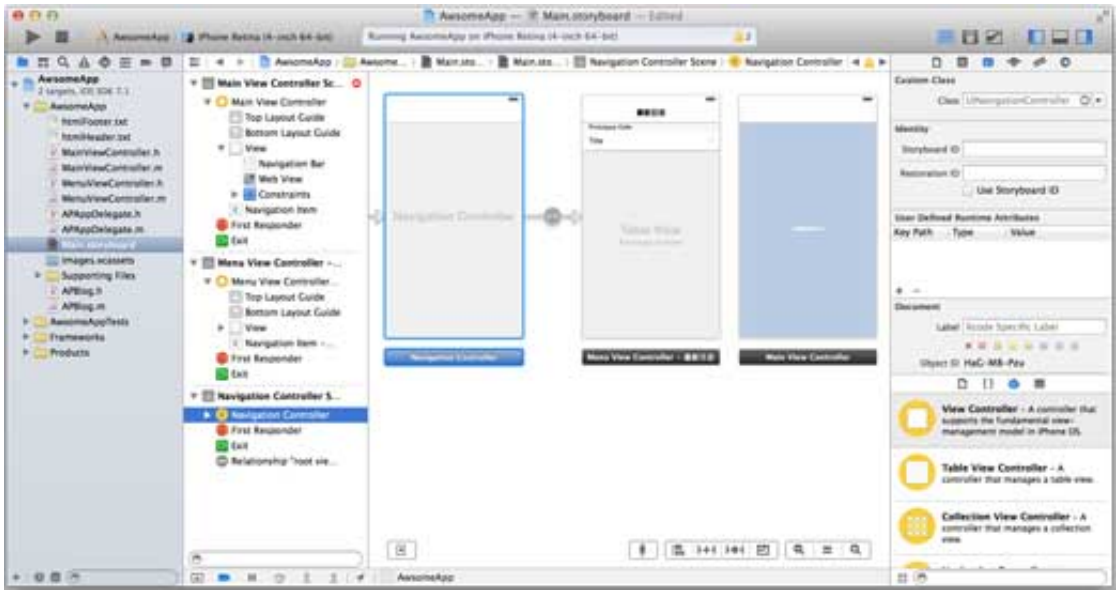
我们首先来看看如何开发iPhone App。前置条件：一台Mac电脑，安装XCode和最新的iOS SDK。

在使用MVVM编写前端页面时，我们就能感受到，用REST API封装网站后台的功能，不但能清晰地分离前端页面和后台逻辑，现在这个好处更加明显，移动App也可以通过REST API从后端拿到数据。

我们来设计一个简化版的iPhone App，包含两个屏幕：列出最新日志和阅读日志的详细内容：

只需要调用API：/api/blogs。

在XCode中完成App编写：



由于我们的教程是Python，关于如何开发iOS，请移步Develop Apps for iOS。

点击下载iOS App源码。

如何编写Android App？这个当成作业了。
