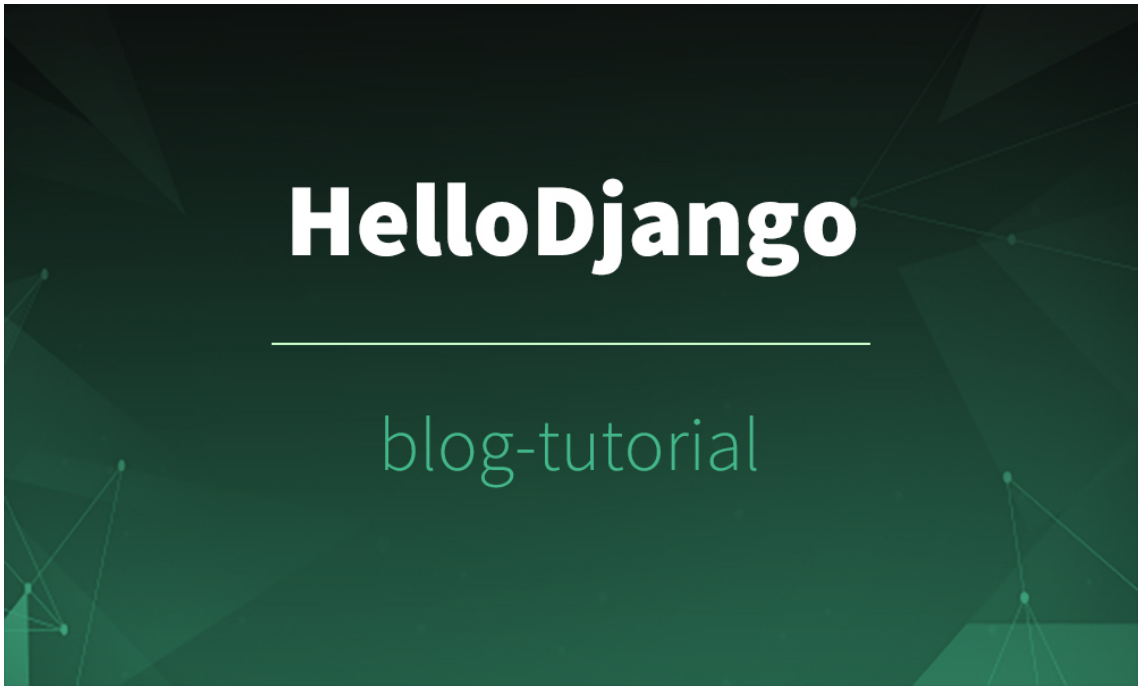
# 是时候让大家看看你用django写出来的博客了（内含部署教程视频）




作者：HelloGitHub-**追梦人物**

> 文中涉及的示例代码，已同步更新到 HelloGitHub-Team 仓库

博客的基础功能已经开发的差不多了，虽然还有很多地方可以完善，但我们还是希望早点把博客部署到服务器上，让他人可以通过外网访问。至于有待完善的地方，可以等部署完后一点点地迭代和改进。现在就让我们来把博客部署到服务器上吧！

↓↓↓ **视频在这里** ↓↓↓

> 作者亲自录制的真机环境演示部署全过程，再不成功你打我！
>
> B 站演示（阿里云 CentOS 7 系统）观看地址：https://www.bilibili.com/video/av68020610/

**注意**：本文的每一个步骤都在真实环境下验证无误。除非你知道自己在做什么，否则建议每一步均严格按照教程的指导来，这样能保证你顺利完成部署。

## 部署前准备

我们将使用比较流行的 Nginx + Gunicorn 的方式将 django 开发的博客部署到自己的服务器，让别人能够通过域名访问你的博客。至于 Nginx、Gunicorn 是什么暂时放到一边，读完本教程后你就会知道它们的作用和使用方法了。

为了部署我们的博客，需要满足以下两个条件：

1. 最好有一台可以通过外网访问的服务器（如果没有的话可以在自己的电脑上建一台虚拟机，通过虚拟 ip 访问）。
2. 最好有一个域名（如果没有的话，则只能使用 ip 访问，且不能开启 HTTPS）。

## 配置服务器

本教程使用的本地环境为 Windows 10，服务器环境为 CentOS 7（64 位）。如果你的环境和我的有所差异（比如 Ubuntu）导致一些命令无法执行，将这些命令转换为你所在环境的命令执行即可，重点在于理解每一步的作用，这样无论在何种环境你都能成功地完成部署，而不是机械地复制粘贴命令。

### 远程登录到服务器

服务器通常位于云端，需要使用远程登录工具登录后才能对服务器进行操作。我使用的是 Xshell，Windows 下百度 Xshell 下载安装即可，软件对学校和个人用户是免费的。

如何远程登录到服务器这里就不赘述了，相信你参考网上的一些教程肯定能够顺利登录。假如你和我一样使用 Xshell 的话，这里有一篇很详细的教程可以参考：教你怎么使用 xshell 远程连接 linux 服务器。

### 创建一个超级用户

顺利连接到远程服务器了，如果是一台全新服务器的话，通常我们是以 root 用户登录的。在 root 下部署代码不够安全，最好是建一个新用户（如果你已经以非 root 用户登录的话可以跳过这一步）。下面的一些列命令将创建一个拥有超级权限的新用户（把 yangxg 替换成你自己想要的用户名，我这里取我的名字拼音 yangxg）：

```
# 在 root 用户下运行这条命令创建一个新用户，yangxg 是用户名
# 因为我叫杨学光，所以我取的用户名是 yangxg
# 选择一个你喜欢的用户名，不一定非得和我的相同
root@server:~# adduser yangxg

# 为新用户设置密码
# 注意在输密码的时候不会有字符显示，不要以为键盘坏了，正常输入即可
root@server:~# passwd yangxg

# 把新创建的用户加入超级权限组
root@server:~# usermod -aG wheel yangxg

# 切换到创建的新用户
root@server:~# su - yangxg

# 切换成功，@符号前面已经是新用户名而不是 root 了
yangxg@server:$
```

新用户创建并切换成功了。如果是新服务器的话，最好先更新一下系统，避免因为版本太旧而给后面安装软件带来麻烦。运行下面的两条命令：

```
yangxg@server:$ sudo yum update
yangxg@server:$ sudo yum upgrade
```

## 更新 SQLite3

为了方便，我们博客使用了 SQLite3 数据库，django 2.2 要求 SQLite3 数据库版本在 3.8.3 以上，而 CentOS 7 系统自带版本低于 django 2.2 所要求的最低版本，所以首先来更新 SQLite3 的版本。

> **注意**
>
> 有可能你使用的服务器系统发行版 SQLite3 已经高于 3.8.3，这一步就可以跳过。如何查看 SQLite3 的版本呢？请执行 sqlite3 --version

首先登陆到 sqlite 的官方下载地址，查看最新发布的版本，截止到本教程完成时，其最新版本为 3.29.0，找到该版本的源码压缩包，复制其下载链接，然后通过 wget 命令下载到服务器（我一般习惯将源码放在 ~/src 目录下。）

```
# 创建 src 目录并进到这个目录
yangxg@server:$ mkdir -p ~/src
yangxg@server:$ cd ~/src

# 下载 sqlite3 源码并解压安装
yangxg@server:$ wget https://sqlite.org/2019/sqlite-autoconf-3290000.tar.gz
yangxg@server:$ tar zxvf sqlite-autoconf-3290000.tar.gz
yangxg@server:$ cd sqlite-autoconf-3290000
yangxg@server:$ ./configure
yangxg@server:$ make
yangxg@server:$ sudo make install
```

> 小贴士：
>
> 如果 wget 命令不存在，使用 sudo yum install -y wget 安装即可。

至此 SQLite3 更新完毕，接下来安装 Python3。

## 安装 Python3 和 Pipenv

CentOS 7 自带的 Python 发行版为 2.7，因此需要安装 Python3，为了兼容性，我们安装 Python 3.6.4。

首先安装可能的依赖：

```
yangxg@server:$ sudo yum install -y openssl-devel bzip2-devel expat-devel gdbm-devel readline-devel sqlite-devel
```

然后下载 Python 3.6.4 的源码并解压：

```
yangxg@server:$ cd ~/src
yangxg@server:$ wget https://www.python.org/ftp/python/3.6.4/Python-3.6.4.tgz
yangxg@server:$ tar -zxvf Python-3.6.4.tgz
```

最后编译安装：

```
yangxg@server:$ cd Python-3.6.4
yangxg@server:$ ./configure LD_RUN_PATH=/usr/local/lib LDFLAGS="-L/usr/local/lib" CPPFLAGS="-I/usr/local/include"
yangxg@server:$ make LD_RUN_PATH=/usr/local/lib
yangxg@server:$ sudo make install
```

注意这里安装 Python 时，Python 会依赖 SQLite3 的库，所以在 configure 时通过 LD_RUN_PATH 指定依赖的搜索目录（因为我们之前更新了 SQLite3 的版本，指定依赖搜索目录确保使用新的 SQLite3 依赖库），另外两个参数作用类似。

然后输入 python3.6 -V 和 pip3.6 -V 命令测试安装结果，输出版本号说明安装成功了。

有了 pip，就可以安装 Pipenv 了：

```
yangxg@server:$ sudo pip3.6 install pipenv
```

> 小贴士：
>
> 如果以上命令报错，可能是因为 pip3.6 安装在当前用户的 bin 路径下（/usr/local/bin/），使用 pip3.6 install pipenv --users 命令也把 Pipenv 安装到当前用户的 bin 路径下就可以了。

## 部署代码

接下来开始准备部署代码，让我们的博客应用在服务上跑起来，这和在本地开发时的过程是一模一样的。不过为了将应用部署到服务器上，我们首先要对项目做一点配置，打开 settings.py，找到 `ALLOWED_HOSTS`，将其修改为：

```
blogproject/settings.py

ALLOWED_HOSTS = ['127.0.0.1', 'localhost ', '.zmrenwu.com']
```

指定了 `ALLOWED_HOSTS` 的值后，django 将只允许通过指定的域名访问我们的应用，比如这里只允许通过 127.0.0.1，localhost 以及 zmrenwu.com 和其任意子域名（域名前加一个点表示允许访问该域名下的子域名）访问（即 HTTP 报文头部中 Host 的值必须是以上指定的域名，通常你在浏览器输入域名访问网站时，Host 的值就会被设置为网站的域名），这样可以避免 HTTP Host 头攻击。

Django 项目中会有一些 CSS、JavaScript 等静态文件，为了能够方便地让 Nginx 处理这些静态文件的请求，我们把项目中的全部静态文件收集到一个统一的目录下，这个目录通常位于 django 项目的根目录，并且命名为 static。为了完成这些任务，需要在项目的配置文件里做一些必要的配置：

```
blogproject/settings.py

# 其他配置...

STATIC_URL = '/static/'
# 加入下面的配置
STATIC_ROOT = os.path.join(BASE_DIR, 'static')
```

`STATIC_ROOT` 即指定静态文件的收集路径，这里指定为 BASE_DIR（项目根目录，在 settings.py 文件起始处定义）下的 static 文件夹。

现在的关键是把代码传到服务器上来了，这里我们使用 git。首先安装 git：

```
yangxg@server:$ sudo  yum install -y git
```

将代码上传到 GitHub 等代码托管平台，这样我们就可以方便地把代码拉取到服务器了。Git 和 GitHub 的使用相信你已经很熟悉了，这里就不赘述过程。如果不知道如何使用地话可以自行百度相关教程。**注意数据库文件不要上传！**

我通常喜欢把应用代码放在 ~/apps/ 目录下，先来设置一下服务器的文件结构，用于存放应用代码等相关文件：

```
# 在用户目录下创建 apps 目录并进入
yangxg@server:$ mkdir -p ~/apps
yangxg@server:$ cd ~/apps

# 拉取博客代码
yangxg@server:$ git clone https://github.com/HelloGitHub-Team/HelloDjango-blog-tutorial.git
```

然后进入到项目根目录，安装项目依赖：

```
yangxg@server:$ cd ~/apps/HelloDjango-blog-tutorial

yangxg@server:$ pipenv install --deploy --ignore-pipfile
```

这里指定 `--deploy` 参数，Pipenv 将只会安装 Pipfile 中 [packages] 下指定的依赖。因为我们现在是在线上环境进行部署，仅用于开发环境的相关依赖我们并不需要。

`--ignore-pipfile` 将会使 Pipenv 从 Pipfile.lock 文件中安装项目依赖。Pipfile.lock 记录了项目依赖的精确信息，从这里读取依赖信息能够确保依赖信息被无意中修改或者破坏而使得运行环境因为依赖包的缘故出现不可预料的问题。

Pipenv 会自动帮我们创建虚拟环境，然后将项目依赖安装到虚拟环境下。

然后创建一下数据库：

```
yangxg@server:$ pipenv run python manage.py migrate
```

启动开发服务器：

```
yangxg@server:$ pipenv run python manage.py runserver 0.0.0.0:8000
```

这里我们启动开发服务器时指定了服务器运行的 ip 和端口，这将允许通过公网 ip 的 8000 端口访问我们的博客。

访问 ip:8000，可以看到访问成功（其中 ip 为你服务器的公网 ip）。

## 使用 Gunicorn

Django 官方文档强调使用 runserver 开启的开发服务器仅用于开发测试，不建议用于生产环境。所以我们使用流行的 Gunicorn 来启动可以用于线上环境的服务器。

首先进入到项目根目录，安装 Gunicorn：

```
yangxg@server:$ pipenv install gunicorn
```

由于我们在服务端修改安装了 gunicorn，代码中 Pipfile 文件和 Pipfile.lock 文件会被更新，因此别忘了把改动同步到本地，具体做法可以自行学习，以下是一个参考：

```
# 服务端提交代码
yangxg@server:$ git add Pipfile Pipfile.lock
yangxg@server:$ git commit -m "add gunicorn dependency"
yangxg@server:$ git push

# 本地拉取代码
git pull
```

回到线上服务器，在项目根目录，执行下面的命令启动服务：

```
yangxg@server:$ pipenv run gunicorn blogproject.wsgi -w 2 -k gthread -b 0.0.0.0:8000
```

来解释一下各个参数的含义。

`-w 2 表示启动 2 个 worker 用于处理请求（一个 worker 可以理解为一个进程），通常将 worker 数目设置为 CPU 核心数的 2-4 倍。

`-k gthread` 指定每个 worker 处理请求的方式，根据大家的实践，指定为 `gthread` 的异步模式能获取比较高的性能，因此我们采用这种模式。

`-b 0.0.0.0:8000`，将服务绑定到 8000 端口，运行通过公网 ip 和 8000 端口访问应用。

访问 ip:8000（ip 为你服务器的公网 ip），应用成功访问了，但是我们看到样式完全乱了。别急，这不是 bug！此前我们使用 django 自带的开发服务器，它会自动帮我们处理静态样式文件，但是 Gunicorn 并不会帮我们这么做。因为处理静态文件并不是 Gunicorn 所擅长的事，应该将它交给更加专业的服务应用来做，比如 Nginx。

## 启动 Nginx 服务器

Nginx (engine x) 是一个高性能的 HTTP 和反向代理 web 服务器，它的功能非常多，这里我们主要用它来处理静态文件以及将非静态文件的请求反向代理给 Gunicorn。

当我们访问一个博客文章详情页面时，服务器会接收到下面两种请求：

- 显示文章的详情信息，这些信息通常保存在数据库里，因此需要调用数据库获取数据。
- 图片、css、js 等存在服务器某个文件夹下的静态文件。

对于前一种请求，博客文章的数据需要借助 django 从数据库中获取，Nginx 处理不了，它就会把这个请求转发给 运行在 Gunicorn 服务中的 django 应用，让 django 去处理。而对于后一种静态文件的请求，只需要去这些静态文件所在的文件夹获取，Nginx 就会代为处理，不再麻烦 django。

用 django 去获取静态文件是很耗时的，但 Nginx 可以很高效地处理，这就是我们要使用 Nginx 的原因。

首先安装 Nginx：

```
yangxg@server:$ sudo yum install epel-release -y
yangxg@server:$ sudo yum install nginx -y
```

运行下面的命令启动 Nginx 服务：

```
yangxg@server:$ sudo systemctl start nginx
```

在浏览器输入 ip（不输入端口则默认为 80 端口，Nginx 默认在 80 端口监听请求），看到 Nginx 的欢迎界面说明 Nginx 启动成功了。



## 配置 Nginx

Nginx 的配置位于 /etc/nginx/nginx.conf 文件中，你可以打开这个文件看看里面的内容，下面是一些关键性的配置：

```
user nobody nobody;
...
http {
    # Load modular configuration files from the /etc/nginx/conf.d directory.
    # See http://nginx.org/en/docs/ngx_core_module.html#include
    # for more information.
    include /etc/nginx/conf.d/*.conf;

    server {
        listen       80 default_server;
        listen       [::]:80 default_server;
        server_name  _;
        root         /usr/share/nginx/html;

        # Load configuration files for the default server block.
        include /etc/nginx/default.d/*.conf;

        location / {
        }
    }
}
```

首先是这个 user 配置，用于指定 Nginx 进程运行时的用户和组（分别为第一个和第二个参数），为了防止可能的权限问题，我们改成当前系统用户（我的用户名是 yangxg，所属组 yangxg，记得改成你自己服务器中运行的用户和组，修改完后记得保存文件内容）：

```
user yangxg yangxg;
```

然后在 http 配置下有一个 server 模块，server 模块用于配置一个虚拟服务，使这个虚拟服务监听指定的端口和域名。你可以配置多个 server，这样就会启动多个虚拟服务，用于监听不同端口，或者是同一个端口，但是不同的域名，这样你就可以在同一服务器部署多个 web 应用了。

这个 server 的配置我们下面会详细讲解，再来看看 server 下的 include，include 会将指定路径中配置文件包含进来，这样便于配置的模块化管理，例如我们可以把不同 web 应用的配置放到 /etc/nginx/conf.d/ 目录下，这样 nginx 会把这个目录下所有以 .conf 结尾的文件内容包含到 nginx.conf 的配置中来，而无需把所有配置都堆到 nginx.conf 中，使得配置文件十分臃肿。

我们来配置博客应用，上面说了，为了模块化管理，我们将配置写到 /etc/nginx/conf.d/ 目录下。先在服务器的 conf.d 目录下新建一个配置文件，我把它叫做 HelloDjango-blog-tutorial.conf。写入下面的配置内容：

```
server {
    charset utf-8;
    listen 80;
    server_name hellodjango-blog-tutorial-demo.zmrenwu.com;

    location /static {
        alias /home/yangxg/apps/HelloDjango-blog-tutorial/static;
    }

    location / {
        proxy_set_header Host $host;
        proxy_pass http://127.0.0.1:8000;
    }
}
```

首先我们配置了一个虚拟服务，编码方式为 utf-8，监听于 80 端口。

服务的域名为 hellodjango-blog-tutorial-demo.zmrenwu.com，所以来自这个域名的请求都会被这个服务所处理。

所有URL 匹配 /static 的请求均由 Nginx 处理，alias 指明了静态文件的存放目录，这样 Nginx 就可以在这个目录下找到请求的文件返回给客户端。

其它请求转发给运行在本机 8000 端口的应用程序处理，我们会在这个端口启动 Gunicorn 用于处理 Nginx 转发过来的请求。

重启 nginx 使得配置生效：

```
yangxg@server:$ sudo systemctl restart nginx
```

## 关闭 DEBUG 模式，收集静态文件

开发环境下，django 为了调试方便，会将 settings.py 文件中的 DEBUG 选项配置为 True，这样如果程序运行出错，调试信息将一览无余，这在开发时很方便，但部署到线上就会带来巨大安全隐患，所以我们把 DEBUG 选项设置为 False，关闭调试模式，在本地将 settings.py 中的 DEBUG 为：

```
DEBUG=False
```

**线上服务器更新最新的代码**，然后运行命令收集静态文件到之前配置的 STATIC_ROOT 目录下：

```
yangxg@server:$ pipenv run python manage.py collectstatic
```

然后使用 Gunicorn 启动服务。

```
yangxg@server:$ pipenv run gunicorn blogproject.wsgi -w 2 -k gthread -b 127.0.0.1:8000
```

现在，访问配置的域名 hellodjango-blog-tutorial-demo.zmrenwu.com（改成你自己在 Nginx 中配置的域名），可以看到博客成功部署！

## 管理 Gunicorn 进程

现在 Gunicorn 是我们手工启动的，一旦我们退出 shell，服务器就关闭了，博客无法访问。就算在后台启动 Gunicorn，万一哪天服务器崩溃重启了又得重新登录服务器去启动，非常麻烦。为此使用 Supervisor 来管理 Gunicorn 进程，这样当服务器重新启动或者 Gunicorn 进程意外崩溃后，Supervisor 会帮我们自动重启 Gunicorn。

先按 Ctrl + C 停止刚才启动的 Gunicorn 服务进程。

首先安装 Supervisor 注意这里使用的是**系统自带的 pip2**，因为截至本教程书写时 Supervisor 还不支持 Python3，不过这并不影响使用。

```
yangxg@server:$ pip install supervisor
```

为了方便，我一般会设置如下的目录结构（位于 ~/etc 目录下）来管理 Supervisor 有关的文件：

```
~/etc

├── supervisor
│   ├── conf.d
│   └── var
│       ├── log
└── supervisord.conf
```

其中 supervisord.conf 是 Supervior 的配置文件，它会包含 conf.d 下的配置。var 目录下用于存放一些经常变动的文件，例如 socket 文件，pid 文件，log 下则存放日志文件。

首先来建立上述的目录结构：

```
yangxg@server:$ mkdir -p ~/etc/supervisor/conf.d
yangxg@server:$ mkdir -p ~/etc/supervisor/var/log
```

然后进入 ~/etc 目录下生成 Supervisor 的配置文件：

```
yangxg@server:$ cd ~/etc
yangxg@server:$ echo_supervisord_conf > supervisord.conf
```

修改 supervisor.conf，让 Supervisor 进程产生的一些文件生成到上面我们创建的目录下，而不是其默认指定的地方。

首先找到 [unix_http_server] 版块，将 file 设置改为如下的值：

```
[unix_http_server]
file=/home/yangxg/etc/supervisor/var/supervisor.sock
```

即让 socket 文件生成在 ~/etc/supervisor/var/ 目录下。注意 supervisor 不支持将 ~ 展开为用户 home 目录，所以要用绝对路径指定。

类似的修改 [supervisord] 板块下的 logfile 和 pidfile 文件的路径，还有 user 改为系统用户，这样 supervisor 启动的进程将以系统用户运行，避免可能的权限问题：

```
logfile=/home/yangxg/etc/supervisor/var/log/supervisord.log
pidfile=/home/yangxg/etc/supervisor/var/supervisord.pid
user=yangxg
```

还有 [supervisorctl] 板块下：

```
serverurl=unix:///home/yangxg/etc/supervisor/var/supervisor.sock
```

[include] 版块，将 /home/yangxg/etc/supervisor/conf.d/ 目录下所有以 .ini 结尾的文件内容包含到配置中来，这样便于配置的模块化管理，和之前 Nginx 配置文件的处理方式是类似的。

```
files = /home/yangxg/etc/supervisor/conf.d/*.ini
```

然后我们到 conf.d 新建我们博客应用的配置：

```
[program:hellodjango-blog-tutorial]
command=pipenv run gunicorn blogproject.wsgi -w 2 -k gthread -b 127.0.0.1:8000
directory=/home/yangxg/apps/HelloDjango-blog-tutorial
autostart=true
autorestart=unexpected
user=yangxg
stdout_logfile=/home/yangxg/etc/supervisor/var/log/hellodjango-blog-tutorial-stdout.log
stderr_logfile=/home/yangxg/etc/supervisor/var/log/hellodjango-blog-tutorial-stderr.log
```

说一下各项配置的含义：

[program:hellodjango-blog-tutorial] 指明运行应用的进程，名为 hellodjango-blog-tutorial。

- command 为进程启动时执行的命令。

- directory 指定执行命令时所在的目录。
- autostart 随 Supervisor 启动自动启动进程。
- autorestart 进程意外退出时重启。
- user 进程运行的用户，防止权限问题。
- stdout_logfile，stderr_logfile 日志输出文件。

启动 Supervisor

```
yangxg@server:$ supervisord -c ~/etc/supervisord.conf
```

-c 指定 Supervisr 启动时的配置文件。

进入 supervisorctl 进程管理控制台：

```
yangxg@server:$ supervisorctl -c ~/etc/supervisord.conf
```

执行 update 命令更新配置文件并启动应用。

浏览器输入域名，可以看到服务已经正常启动了。

## 使用 CDN 加快 Bootstrap 和 jQuery 的加载速度

我们的项目使用了 Bootstrap 和 jQuery，这两个文件我们是从本地加载的。如果服务器性能比较差的话，加载需要耗费很长的时间，网站打开的速度就变得无法忍受。我们使用 CDN 来加快加载速度。具体来说，替换 base.html 的几个静态文件的加载标签：

```
base.html

- <link rel="stylesheet" href="{% static 'blog/css/bootstrap.min.css' %}">
- <script src="{% static 'blog/js/jquery-2.1.3.min.js' %}"></script>
- <script src="{% static 'blog/js/bootstrap.min.js' %}"></script>
+ <link href="https://cdn.bootcss.com/bootstrap/3.3.7/css/bootstrap.min.css" rel="stylesheet">
+ <script src="https://cdn.bootcss.com/jquery/2.1.3/jquery.min.js"></script>
+ <script src="https://cdn.bootcss.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
```

本地修改代码后，将代码同步到线上服务器，执行下面的命令重启 hellodjango-blog-tutorial 应用进程：

```
yangxg@server:$ supervisorctl -c ~/etc/supervisord.conf restart hellodjango-blog-tutorial
```

这样网站访问的速度将大大提升！

## 总结

部署步骤很多且繁杂，因为每个环境都不一样，因此部署是最容易出错的步骤，一定要搞清楚每一步的作用，这样在遇到问题时，才能针对性地去解决，如果只知道一味地复制粘贴命令，而不知道自己在干嘛，那么一旦出错将束手无策。

## 部署过程自动化

在整个部署过程中我们运行了十几条命令，手动输入了 N 个字符。如果每次更新代码都要远程连接到服务器执行这些命令的话将变得非常麻烦。接下来的教程我们将介绍使用 Fabric 自动化整个部署过程。写好部署脚本后，只需要执行一条命令，就可以非常方便地自动完成整个部署。

---

**『讲解开源项目系列』**——让对开源项目感兴趣的人不再畏惧、让开源项目的发起者不再孤单。跟着我们的文章，你会发现编程的乐趣、使用和发现参与开源项目如此简单。欢迎留言联系我们、加入我们，让更多人爱上开源、贡献开源～



分类: HelloDjango





« 上一篇： 讲解开源项目：让你成为灵魂画手的 JS 引擎：Zdog
» 下一篇： 可能是 Python 中最火的第三方开源测试框架 pytest