部署

Django打包

构建setup.py文件

```
from distutils.core import setup
import glob

setup(name='blog',
      version='1.0',
      description='magedu blog',
      author='Wayne',
      author_email='wayne@magedu.com',
      url='http://www.magedu.com/python',
      packages=['blog', 'post', 'user'],
      py_modules=['manage'], # 可以不打包manage.py
      data_files=glob.glob('templates/*.html') + ['requirements']
     )
```

```
# 应用程序的根目录下打包
$ pip freeze > requirements
$ python setup.py sdist --formats=gztar # gz
```

在Linux系统中创建一个python虚拟环境目录，使用

```
$ pyenv virtualenv 3.5.3 blog353
$ pyenv local blog353
$ pip list
$ tar xf blog-1.0.tar.gz
$ mv blog-1.0 blog # Django应用部署目录
$ cd blog
# yum install python-devel mysql-devel # CentOS需要root权限yum安装，mysqlclient依赖
$ pip install -r requirements

$ pip list
Package     Version
----------- -------
bcrypt      3.1.4
cffi        1.11.5
Django      1.11
mysqlclient 1.3.13
pip         10.0.1
pycparser   2.18
PyJWT       1.6.4
pytz        2018.5
setuptools  28.8.0
```

```
simplejson  3.16.0
six         1.11.0

$ sed -i -e 's/DEBUG.*/DEBUG = False/' -e 's/ALLOWED_HOSTS.*/ALLOWED_HOSTS = ["*"]/'
blog/settings.py # 修改Django配置
$ python manage.py runserver 0.0.0.0:8000 # 测试
```

使用`http://192.168.142.135:8000/post?page=2&size=2`成功返回数据，说明Django应用成功

至此，Django应用部署完成。Django带了个开发用Web Server，生成环境不用，需要借助其它Server。

注意：ALLOWED_HOSTS = ["*"] 这是所有都可以访问，生产环境应指定具体可以访问的IP，而不是所有。

WSGI

Web Server Gateway Interface，是Python中定义的Web Server与应用程序的接口定义。

应用程序有WSGI 的 Django框架负责，WSGI Server谁来做？

uWSGI 项目

uWSGI是一个C语言的项目，提供一个WEB服务器，它支持WSGI协议，可以和Python的WSGI应用程序通信。

官方文档 https://uwsgi-docs.readthedocs.io/en/latest/

uWSGI可以直接启动HTTP服务，接收HTTP请求，并调用Django应用。

安装

```
$ pip install uwsgi
$ uwsgi --help
```

uWSGI + Django部署

在Django项目根目录下，运行`$ uwsgi --http :8000 --wsgi-file blog/wsgi.py --stats :8001`，使用下面的链接测试

http://192.168.142.135:8000/
http://192.168.142.135:8000/post/?page=1&size=2

运行正常。

React项目打包

rimraf 递归删除文件，rm -rf

`$ npm install rimraf --save-dev`

在package.json中替换

`"build": "rimraf dist && webpack -p --config webpack.config.prod.js"`

在webpack.config.prod.js中增加css的loader，否则编译可能出错

```
    module: {
        rules: [
            {
                test: /\.js$/,
                exclude: /node_modules/, // 不编译第三方包
                use: [
                    { loader: 'react-hot-loader/webpack' },
                    { loader: 'babel-loader' }
                ]
            },
            {
                test: /\.css$/,
                exclude: /node_modules/,
                use: [ 'style-loader', 'css-loader' ]
            },
            {
                test: /\.less$/,
                use: [
                    { loader: "style-loader" },
                    { loader: "css-loader" },
                    { loader: "less-loader" }
                ]
            }
        ]
    },
```

`$ npm run build` 编译，成功。查看项目目录中的dist目录。

nginx uwsgi 部署

tengine安装

淘宝提供的nginx

```
# yum install gcc openssl-devel pcre-devel -y

# tar xf tengine-1.2.3
# ./configure --help | grep wsgi
# ./configure # 第一步
# make && make install # 第二步、第三步
# cd /usr/local/nginx/ # 默认安装位置
```

编译安装过程中，已经看到安装了fastcgi、uwsgi模块。

nginx配置

```
    server {
        listen       80;
```

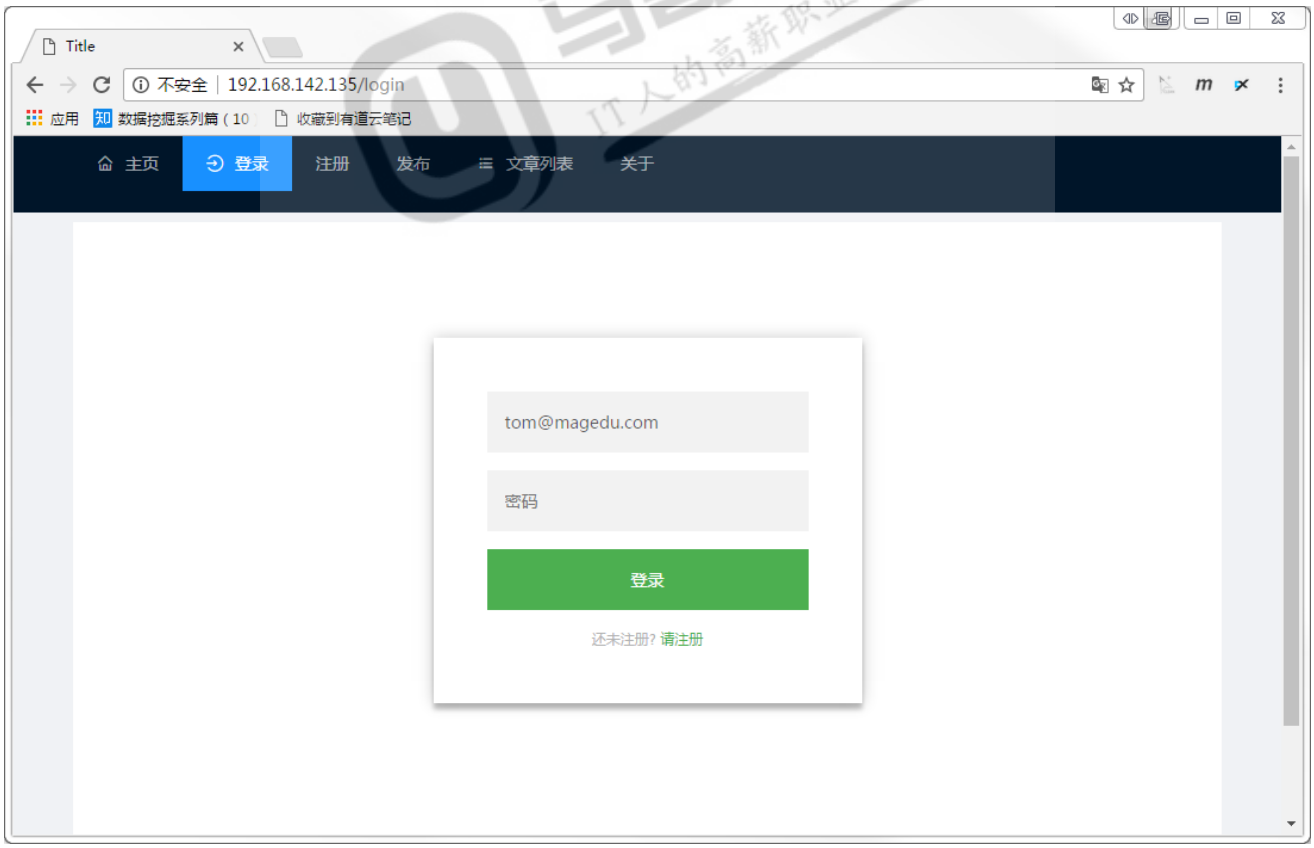
```
        server_name  localhost;

        #charset koi8-r;
        #access_log  logs/host.access.log  main;

        location ^~ /api/ {
            rewrite ^/api(/.*) $1 break;
            proxy_pass http://127.0.0.1:8000;
        }

        location / {
            root   html;
            index  index.html index.htm;
        }
    }
```

`^~ /api/` 左前缀匹配
`rewrite ^/api(/.*) $1 break;` 重写请求的path

访问 `http://192.168.142.135/` ，可以看到nginx的首页了。

部署博客前端系统

将编译好的前端项目文件index.htm复制到/usr/local/nginx/html， app-xxx.js复制到/usr/local/nginx/html/assets。刷新首页，一切从此开始。



uwsgi部署

目前nginx和uWSGI直接使用的是HTTP通信，效率低。改为使用uwsgi通信。

本次pyenv的虚拟目录是/home/python/magedu/projects/web。其下放uWSGI的配置文件blog.ini。

```
[uwsgi]
socket = 127.0.0.1:9000
chdir = /home/python/magedu/projects/web/blog
wsgi-file = blog/wsgi.py
```

socket = 127.0.0.1:9000 使用uwsgi协议通信
chdir = /home/python/magedu/projects/web/blog Django项目根目录
wsgi-file = blog/wsgi.py 指定App文件，blog下wsgi.py

故意把端口该为9000，是让配置错的人看不到数据，从而发现问题。

Django项目目录/home/python/magedu/projects/web/blog，其下放blog项目

```
(blog353) [python@node web]$ pwd
/home/python/magedu/projects/web
(blog353) [python@node web]$ vim blog.ini
(blog353) [python@node web]$ cat blog.ini
[uwsgi]
socket = 127.0.0.1:9000
chdir = /home/python/magedu/projects/web/blog
wsgi-file = blog/wsgi.py
$ uwsgi blog.ini
```

在nginx中配置uwsgi

```
    server {
        listen       80;
        server_name  localhost;

        #charset koi8-r;
        #access_log  logs/host.access.log  main;

        location ^~ /api/ {
            rewrite ^/api(/.*) $1 break;
            #proxy_pass http://127.0.0.1:8000
            include    uwsgi_params;
            uwsgi_pass 127.0.0.1:9000;
        }

        location / {
            root   html;
            index  index.html index.htm;
        }
    }
```

重新装载nginx配置文件，成功运行。

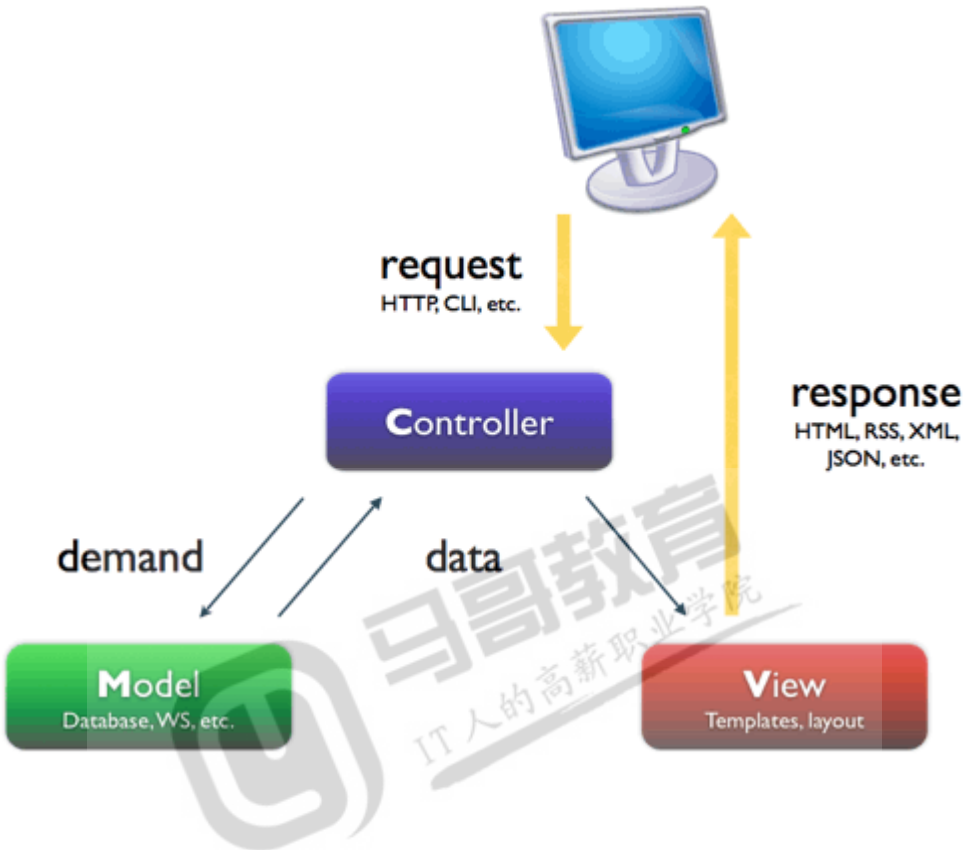
至此，前后端分离的开发、动静分离的部署的博客项目大功告成。

参看 https://uwsgi-docs.readthedocs.io/en/latest/WSGIquickstart.html

uwsgi协议 https://uwsgi-docs.readthedocs.io/en/latest/Protocol.html

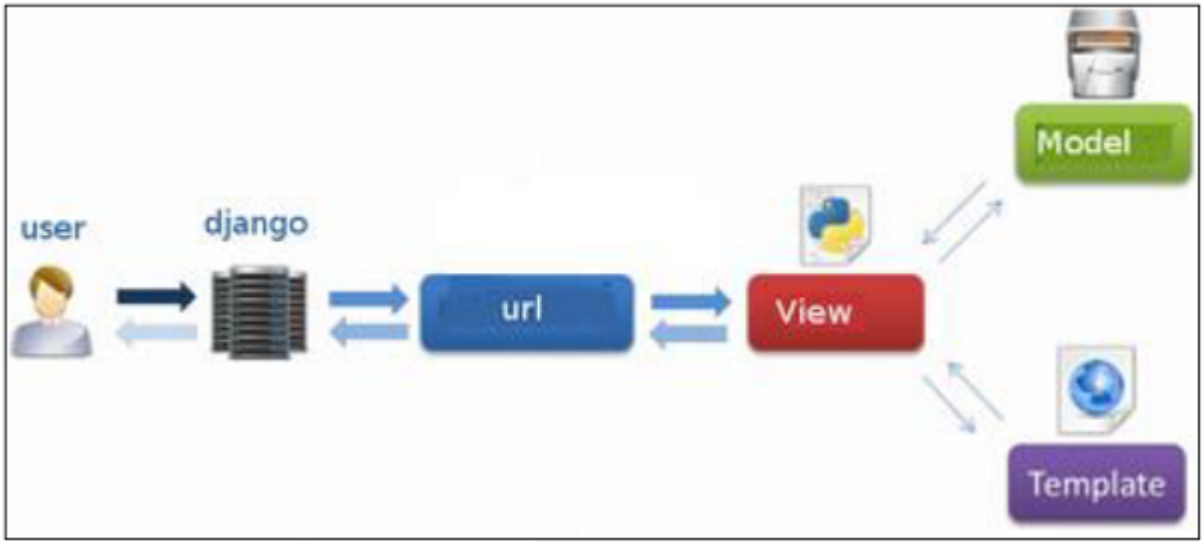
---

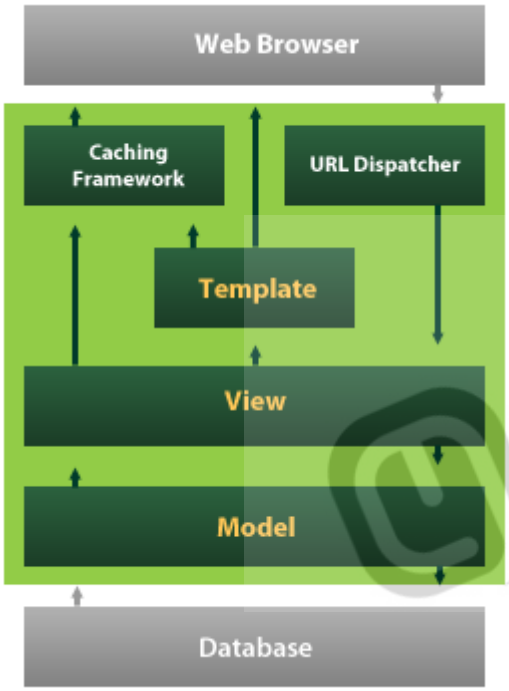
Django



MVC设计模式

Controller控制器：负责接收用户请求，调用Model完成数据，调用view完成对用户的响应
Model模型：负责业务数据的处理
View视图：负责用户的交互界面





Model层
ORM 建立对象关系映射，提供数据库操作

Template层
负责数据的可视化，使用HTML、CSS等构成模板，将数据应用到模板中，并返回给浏览器。

View层
Django完成URL映射，把请求交给view层的视图函数处理，调用Model层完成数据，如有必要调用Template层响应客户端，如果不需要，直接返回数据。