打包

Django项目打包

默认都在项目根目录下

固定开发依赖第三方包，生成依赖文件

```
$ pip freeze > requirements
```

setup参数 https://packaging.python.org/guides/distributing-packages-using-setuptools/#setup-args

setup.py

```
import setuptools

setuptools.setup(
    name="mammoth",
    version="0.1.1",
    author="wayne",
    author_email="wayne@magedu.com",
    description="Mammoth Devops Management",
    url="https://www.magedu.com",
    classifiers=[
        "Programming Language :: Python :: 3",
        "License :: OSI Approved :: MIT License",
        "Operating System :: OS Independent",
    ],
    package_dir={"": "."},
    packages=setuptools.find_packages(), # 找当前路径下的包
    python_requires=">=3.6",
    data_files=[('', ['requirements'])], # data_files=['requirements']
    py_modules=['manage'] # 去掉.py
)
```

py_modules 中的manage.py 按道理不需要，为了部署测试方便，一起打包。

install_requires https://packaging.python.org/discussions/install-requires-vs-requirements/

MANIFEST.in 也可以用来控制文件或目录是否打包。参考 https://packaging.python.org/guides/using-manifest-in/#using-manifest-in。

源码打包如下

```
$ python setup.py sdist --formats=gztar
```

pyenv

如果Linux系统安装多个项目，为防止版本冲突，建议安装多版本工具，并使用虚拟环境。

OS：CentOS 8

安装Pyenv

```
# yum install git

本地编译Python解释器需要
# yum -y install gcc make patch gdbm-devel openssl-devel sqlite-devel readline-devel zlib-devel bzip2-devel

python依赖
# yum install python-devel

mysqlclient依赖，必须，mysqlclient需要
# yum install mysql-devel

# yum install libffi-devel

# useradd python
# echo python | passwd python --stdin

# su - python

进入python用户执行以下操作
$ curl -L https://github.com/pyenv/pyenv-installer/raw/master/bin/pyenv-installer | bash
$ vim ~/.bashrc
export PATH="/home/wayne/.pyenv/bin:$PATH"
eval "$(pyenv init -)"
eval "$(pyenv virtualenv-init -)"

有必要就更新一下
$ pyenv update

$ pyenv install 3.8.8 -vv
```

libffi-devel为了解决，安装mysqlclient、uWSGI时，如果出现`ModuleNotFoundError: No module named '_ctypes'`，缺少ffi库。如果出现该问题，需要重新编译安装Python解释器。

pip国内镜像

可选

```
准备pip配置文件
$ mkdir ~/.pip

$ vim ~/.pip/pip.conf
[global]
index-url = https://mirrors.aliyun.com/pypi/simple/

[install]
trusted-host=mirrors.aliyun.com
```

虚拟环境

```
$ pyenv virtualenv 3.8.8 v388
```

项目部署

项目源代码文件其实部署在哪里都行，就是Python源代码文件而已。uWSGI和Django等需要安装在虚拟环境中。为了减少不必要的麻烦，都部署在同一个环境中。

```
$ mkdir -p ~/projects
$ cd ~/projects
```

上传源代码到projects目录解压

```
$ tar xf mammoth-0.1.1.tar.gz
$ ln -sv mammoth-0.1.1 mammoth
'mammoth' -> 'mammoth-0.1.1'
$ cd mammoth
$ pyenv virtualenv 3.8.8 v388
$ pyenv local v388
(v388) [python@localhost mammoth]$
```

安装项目依赖包

```
安装mysql前一定要yum安装下面的依赖
# yum install mysql-devel
```

```
$ pip list
$ pip install -r requirements
```

如果出现`ModuleNotFoundError: No module named '_ctypes'`，缺少ffi库，需要libffi-devel，并重新编译安装Python

```
# yum install libffi-devel

$ pyenv uninstall 3.8.8
$ pyenv install 3.8.8
$ pyenv virtualenv 3.8.8 v388
```

修改Django配置

```
$ sed -i -e 's/DEBUG =.*/DEBUG = False/' -e 's/ALLOWED_HOSTS.*/ALLOWED_HOSTS = ["*"]/' mammoth/settings.py
```

```
DEBUG = False

ALLOWED_HOSTS = ["*"]

数据库配置
```

运行测试

```
$ python manage.py runserver 0.0.0.0:8000
```

启动过程中如果卡住了，可能是网络阻塞导致的，极有可能是数据库访问有问题。

使用`http://192.168.142.130:8000/login/`测试，有反应，就是正常的。

uWSGI部署

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

应用程序由符合WSGI规范的 Django 框架负责，WSGI Server谁来做？谁来调用Django这个Application呢?

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 Http + Django部署

```
$ uwsgi --wsgi-file mtest1/wsgi.py --http :8000 --stats :8001 --stats-http
```

- 长选项名也是配置项名
- --wsgi-file mtest1/wsgi.py，指定wsgi Application的入口
- http :8000，在0.0.0.0的8000端口，支持HTTP协议访问
- --stats :8001 --stats-http，状态查看

测试`http://192.168.142.130:8000/login`

stats能够显示服务器状态值。--stats-http选项可以使用http访问这个值。
安装uwsgitop获取这个stat值。注意使用这个命令**不要**使用--stats-http选项。

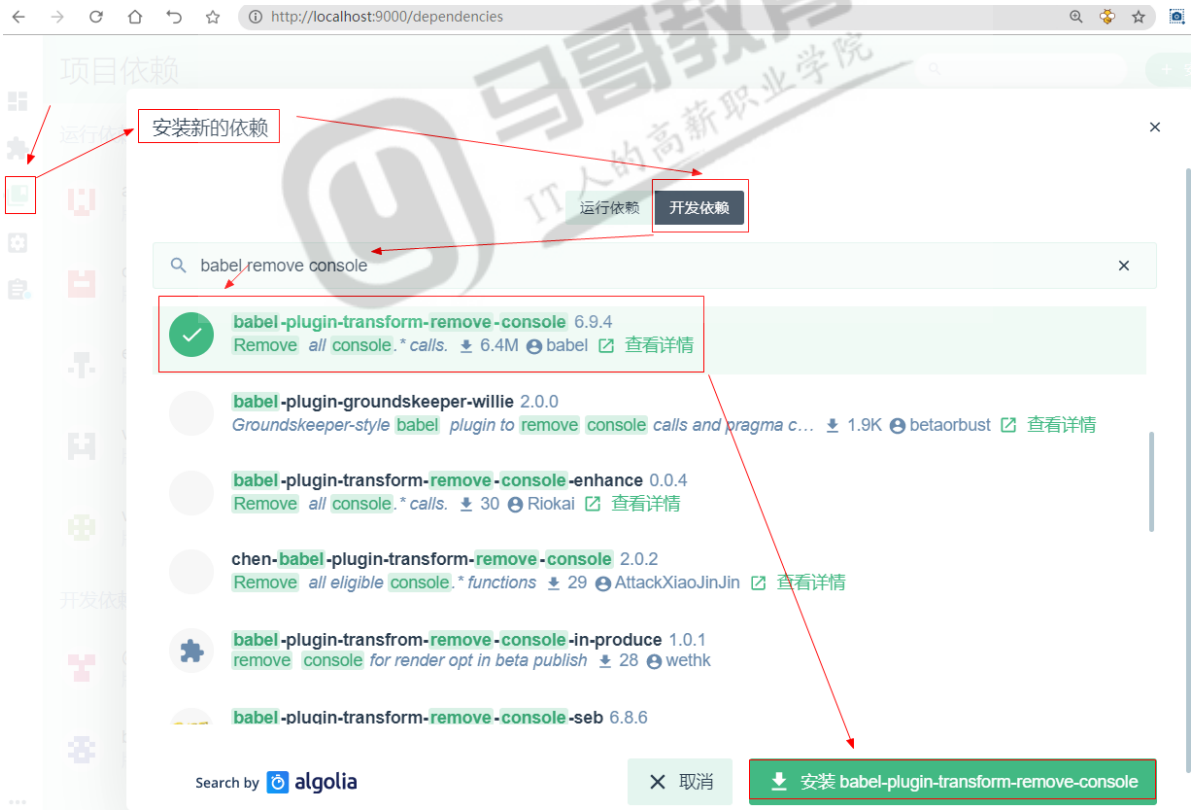
```
$ pip install uwsgitop

$ uwsgi --http :8000 --wsgi-file blog/wsgi.py --stats :8001
$ uwsgitop --frequency 10 127.0.0.1:8001
```

Vue打包

移除console输出

前端项目直接为生产环境编译，报错

开发中，有大量的console控制台输出，不能发布到生产环境的代码中。



配置参考 https://babeljs.io/docs/en/babel-plugin-transform-remove-console/

```
module.exports = {
  presets: ['@vue/cli-plugin-babel/preset'],
  plugins: [
    'transform-remove-console',
    [
      'component',
      {
        libraryName: 'element-ui',
        styleLibraryName: 'theme-chalk'
      }
    ]
  ]
}
```

重新编译后，编译通过。

但是，重启启动项目调试，发现开发环境也没有控制台输出了。这个插件的配置影响生产和开发环境。

在Nodejs中，全局变量process表示当前node进程。

process.env可以获取当前环境变量对象。

webpack通过选项--mode=production或者--mode=development来设置`process.env.NODE_ENV`

mode参考 https://www.webpackjs.com/concepts/mode/

babel.config.js

```
// 判断，生产环境才加载移除控制台输出插件
const prodPlugins = []
if (process.env.NODE_ENV === 'production') {
  prodPlugins.push("transform-remove-console")
}

module.exports = {
  presets: [
    '@vue/cli-plugin-babel/preset'
  ],
  plugins: [
    [
      'component',
      {
        libraryName: 'element-ui',
        styleLibraryName: 'theme-chalk'
      }
    ],
    ...prodPlugins
  ]
}
```

重启前端Server，使其能重新读取babel.conf.js的配置。

Webpack配置

Webpack生产环境和开发环境配置不同，当前项目中vue.config.js就是webpack的配置文件。

vue.config.js目前如下

```
module.exports = {
  devServer: {
    proxy: {
      '/api/v1': {
        target: 'http://localhost:8000',
        changeOrigin: true,
        pathRewrite: { '^/api/v1': '' }
      }
    }
  }
}
```

这里面只是配置了devServer，这是Vue中配置Webpack的DevServer。也可以使用configureWebpack或chainWebpack来配置Webpack。

Vue配置中使用configureWebpack选项对Webpack配置https://cli.vuejs.org/zh/config/#configurewebpack

Vue配置中也可以使用chainWebpack选项对Webpack配置

- https://cli.vuejs.org/zh/config/#chainwebpack
- https://cli.vuejs.org/zh/guide/webpack.html#%E9%93%BE%E5%BC%8F%E6%93%8D%E4%BD%9C-%E9%AB%98%E7%BA%A7
- https://github.com/Yatoo2018/webpack-chain/tree/zh-cmn-Hans

```
module.exports = {
  chainWebpack: config => {
    // 从这里开始
  }
}
```

`config.when(条件，真对应函数，假对应函数)` 参考 https://github.com/neutrinojs/webpack-chain#conditional-configuration

`config.devServer.proxy({})` 参考https://github.com/neutrinojs/webpack-chain#config-devserver-shorthand-methods

```
module.exports = {
  chainWebpack: config => {
    config.when(process.env.NODE_ENV === 'production',
      // 为生产环境修改配置...
      config => { },
      // 为开发环境修改配置...
      config => {
        config.devServer.proxy({
          '/api/v1': {
            target: 'http://localhost:8000',
            changeOrigin: true,
            pathRewrite: { '^/api/v1': '' }
```

```
          }
        })
      }
    )
  }
}
```

修改为chainWebpack配置 https://github.com/Yatoo2018/webpack-chain/tree/zh-cmn-Hans#%E9%85%8D%E7%BD%AE%E6%8F%92%E4%BB%B6-%E4%BF%AE%E6%94%B9%E5%8F%82%E6%95%B0

html-webpack-plugin插件配置

html-webpack-plugin插件，此插件已经被安装过了

插件参考 点击

```
参考
https://cli.vuejs.org/zh/guide/webpack.html#%E4%BF%AE%E6%94%B9%E6%8F%92%E4%BB%B6%E9%80%89%E9%A1%B9
```

通过`vue-cli-service inspect --mode production`或`vue-cli-service inspect --mode development`查看webpack配置


```
module.exports = {
  chainWebpack: config => {
    config.when(process.env.NODE_ENV === 'production',
      // 为生产环境修改配置...
      config => { },
      // 为开发环境修改配置...
      config => {
        config.devServer.proxy({
          '/api/v1': {
            target: 'http://localhost:8000',
            changeOrigin: true,
            pathRewrite: { '^/api/v1': '' }
          }
```

```
        })
        console.log('~~~~~~~~~~~~~~~~~~~')
        config.plugin('html').tap(args => {
          console.log(args);
          return args
        })
        console.log('~~~~~~~~~~~~~~~~~~~')
      }
    )
  }
}
```

```
$ vue-cli-service inspect --mode development
~~~~~~~~~~~~~~~~~~~

[
  {
    title: 'mammoth',
    templateParameters: [Function: templateParameters],
    template:
'E:\\ClassProjects\\frontprojects\\mammoth\\public\\index.html'
  }
]
~~~~~~~~~~~~~~~~~~~
```

通过.tap(args => {})，args[0]就是该插件的选项参数，参数如下

```
    /* config.plugin('html') */
    new HtmlWebpackPlugin(
      {
        title: 'mammoth',
        templateParameters: function () { /* omitted long function */ },
        template:
'E:\\ClassProjects\\frontprojects\\mammoth\\public\\index.html'
      }
    ),
```

```
module.exports = {
  chainWebpack: config => {
    config.when(process.env.NODE_ENV === 'production',
      // 为生产环境修改配置...
      config => {
        config.plugin('html').tap(args => {
          args[0].title = '猛犸运维系统平台'
          return args
        })
      },
      // 为开发环境修改配置...
      config => {
        config.devServer.proxy({
          '/api/v1': {
            target: 'http://localhost:8000',
            changeOrigin: true,
            pathRewrite: { '^/api/v1': '' }
```

```
          }
        })
        console.log('~~~~~~~~~~~~~~~~~~')
        config.plugin('html').tap(args => {
          // args[0] 选项参数
          args[0].title = '猛犸运维系统平台-dev'
          return args
        })
        console.log('~~~~~~~~~~~~~~~~~~')
      }
    )
  }
}
```

CSS和JS优化

本项目由于可能完全在内外使用，所以并不能把部分依赖的CSS、JS文件改到CDN，从而减小打包文件大小。

部署

nginx安装

淘宝提供的nginx

```
安装依赖
# yum install gcc openssl-devel pcre-devel -y

# tar xf tengine-1.2.3
# cd tengine-1.2.3
# ./configure --help | grep wsgi

# ./configure          # 第一步
# make && make install # 第二步、第三步

默认安装位置,这个路径下面的html
# cd /usr/local/nginx/

查看模块
# sbin/nginx -m
测试配置文件
# sbin/nginx -t
```

编译安装过程中，已经看到安装了fastcgi、uwsgi模块。

/usr/local/nginx/是默认安装路径，其子目录

- html默认部署目录
- sbin下面是nginx运行文件
- conf配置目录

http部署

nginx配置

```
    server {
        listen       80;
        server_name  localhost;

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

        location ^~ /api/v1/ {
            rewrite ^/api/v1(/.*) $1 break;
            proxy_pass http://127.0.0.1:8000;
        }

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

`^~ /api/v1/` 左前缀匹配
`rewrite ^/api/v1(/.*) $1 break;` 重写请求的path

```
# pwd
/usr/local/nginx

# sbin/nginx -t
# sbin/nginx -s reload
```

访问`http://192.168.142.130/`，可以看到登录页了。

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。

使用uwsgi协议的命令行写法如下

```
$ uwsgi --socket :9000 --wsgi-file blog/wsgi.py
```

在nginx中配置uwsgi

http://nginx.org/en/docs/http/ngx_http_uwsgi_module.html

```
    server {
        listen       80;
        server_name  localhost;

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

        location ^~ /api/v1/ {
            rewrite ^/api/v1(/.*) $1 break;
```

```
            #proxy_pass http://127.0.0.1:8000;
            uwsgi_pass 127.0.0.1:9000;
            include    uwsgi_params;
        }

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

重新装载nginx配置文件，成功运行。

```
# sbin/nginx -t
# sbin/nginx -s reload
```

uWSGI配置文件

项目源码部署目录/home/python/sources/mammoth

虚拟环境目录/home/python/projects/mammoth

注意，这两个目录其实可以不一样。

Django官方文档参考 https://docs.djangoproject.com/en/3.2/howto/deployment/wsgi/uwsgi/

uWSGI的命令选项就是配置项。https://uwsgi-docs.readthedocs.io/en/latest/Options.html

配置项		说明
socket=127.0.0.1:9000	-s, --socket	使用uwsgi协议通信
chdir=/home/python/sources/mammoth		Django项目根目录
wsgi-file = mammoth/wsgi.py		指定wsgi.py
module = mammoth.wsgi:application		指定wsgi模块
master = True	-M, --master	启用master进程管理工作进程
processes = 4	-p，--processes	启用工作进程数
threads = 2		控制工作进程中的线程数
pidfile=/tmp/mammoth-master.pid		创建进程pid文件
daemonize=/var/log/uwsgi/mammoth.log		后台
vacuum=True		退出清理垃圾

默认uwsgi启动单进程单线程

```
$ touch ~/uwsgi.ini
```

uwsgi.ini配置如下

```
[uwsgi]
socket=127.0.0.1:9000
chdir=/home/python/sources/mammoth
module=mtest1.wsgi:application
master=True
processes = 4
pidfile=/tmp/mammoth-master.pid
daemonize=/var/log/uwsgi/mammoth.log
```

~/uwsgi.ini在家目录

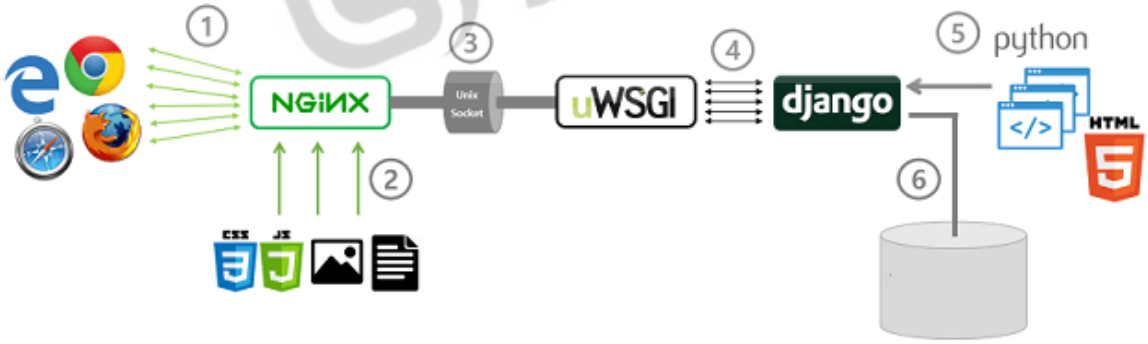
~/sources/mammoth 是源代码

~projects/web 是v388的Python虚拟环境，安装了Django、uwsgi、DRF等包

```
$ cd ~/projects/web
$ python -V
Python 3.8.8

$ uwsgi --ini ~/uwsgi.ini
[uWSGI] getting INI configuration from /home/python/uwsgi.ini
```

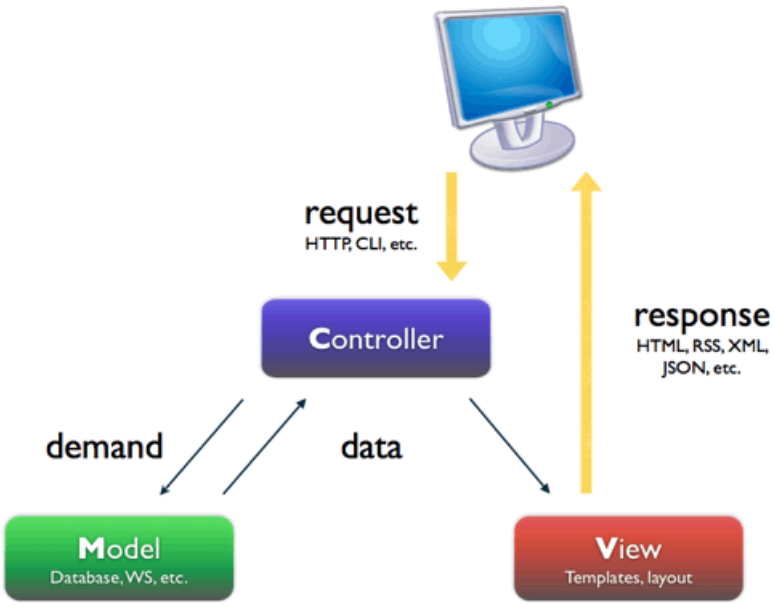
uwsgi协议 https://uwsgi-docs.readthedocs.io/en/latest/Protocol.html

部署图

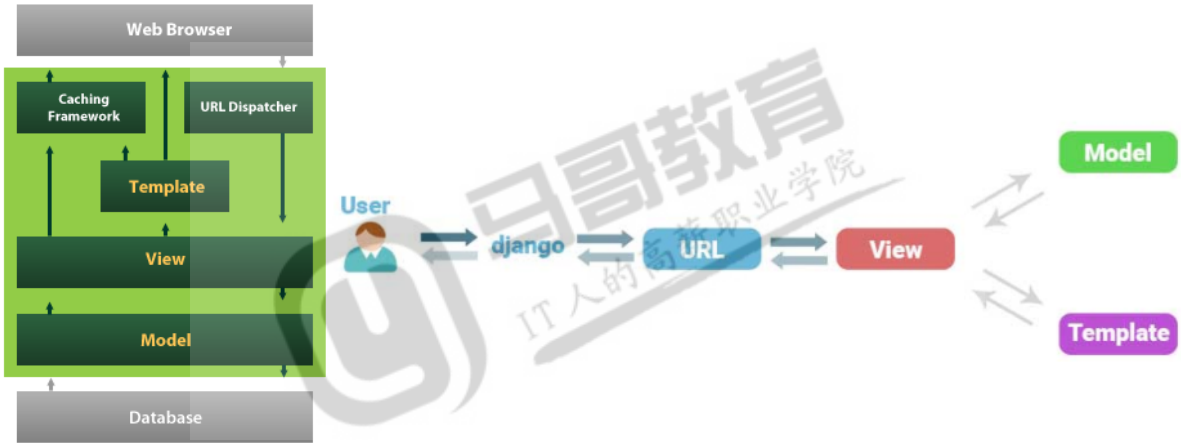


1. 浏览器通过互联网HTTP协议访问NGINX
2. 静态内容（图片、JS、CSS、文件）都由Nginx负责提供WEB服务
3. Nginx配置代理。可以是Http和Socket通信。本次使用uwsgi协议
4. uWSGI服务程序提供uwsgi协议的支持，将从Nginx发来的请求封装后调用WSGI的Application。这个Application可能很复杂，有可能是基于Django框架编写。这个程序将获得请求信息。
5. 通过Django的路由，将请求交给视图函数（类）处理，可能需要访问数据库的数据，也可能使用了模板。最终数据返回给浏览器。

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。这其实也是视图层的部分功能

- View层

  Django完成URL映射后，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。

- 控制器

  Django内部的中间件实际上完成了控制功能，最后通过控制URL分发请求