

# 开发文档

👍 建议在开发部署前阅读此文档



## 前言

相信在经过一段时间的学习之后，对Python，数据库，服务器，计算机网络这些概念都有了一定程度的了解了，那是时候开始我们项目的开发了，在开发之前，我们需要做一些准备工作....

love & peace

## 搭建开发远程服务器

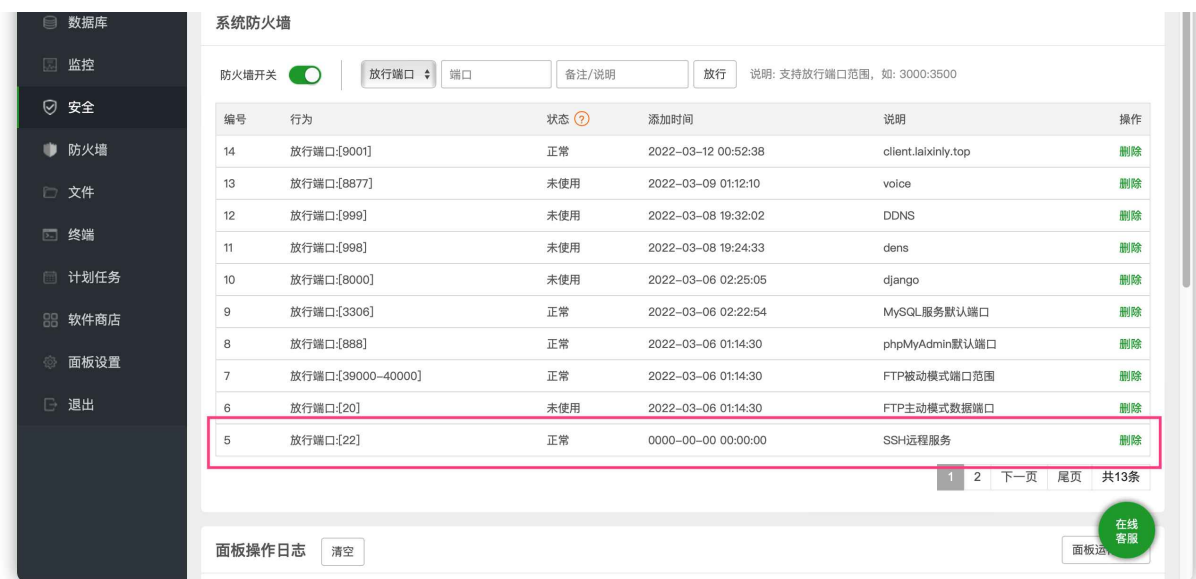
在技术栈文档中我们提到，部分项目对环境的依赖性比较强，比如人脸项目，比如图像识别项目，这些库中有一部分的代码是采用c++或者Java进行编写的，其中引用的动态库可能只能在服务器环境上成功调用，对于开发会造成一点点👉的影响，因此远程开发也是非常重要的。

我们采用的远程开发工具为Gateway，Gateway中包含了Pycharm的全部功能以及Git的支持，因此从界面上操作是与Pycharm99%重合的。👌

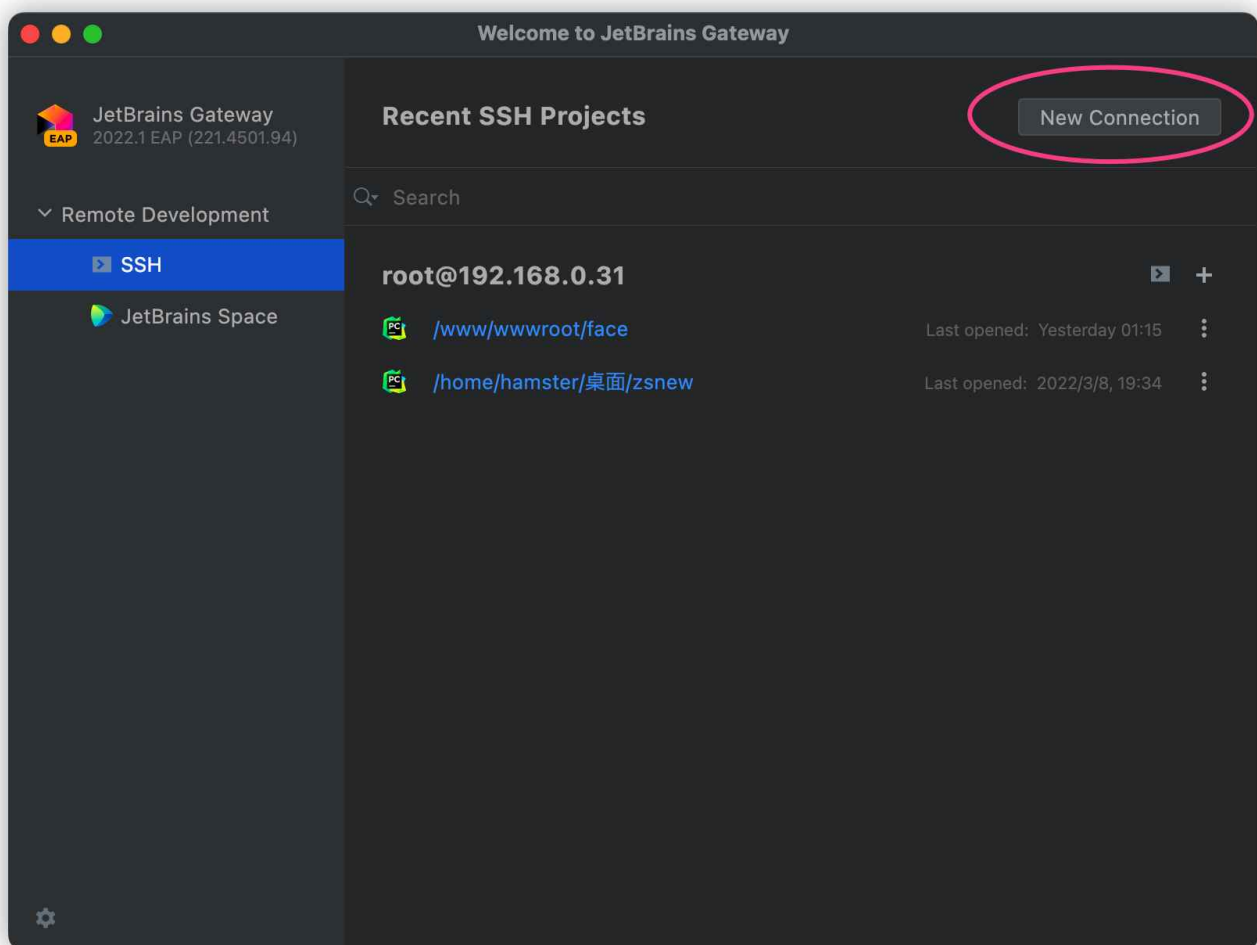
## 配置Gateway

Gateway是依赖于ssh来实现服务的，因此首先我们需要在服务器或宝塔面板里将ssh 22端口开放

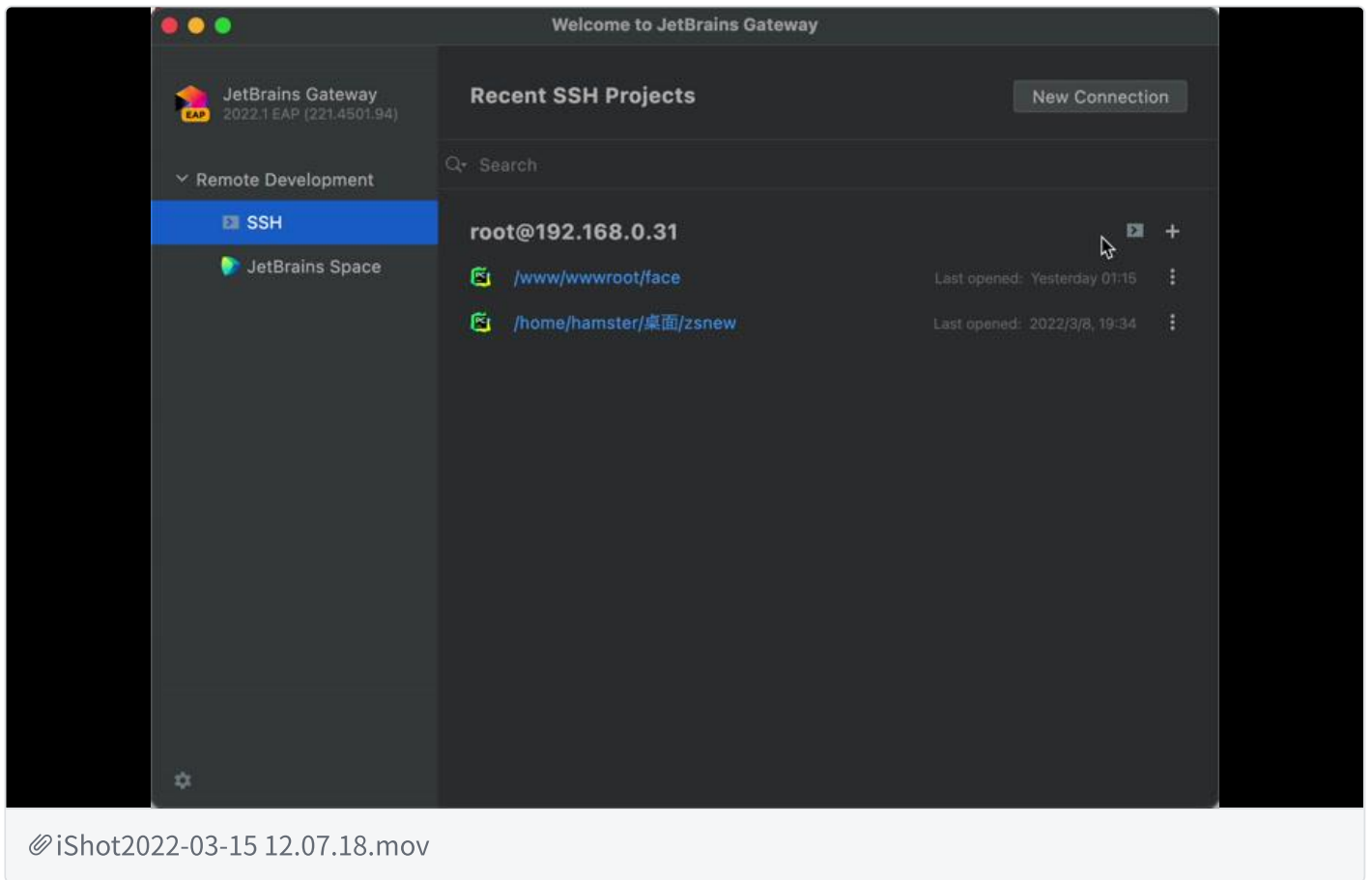




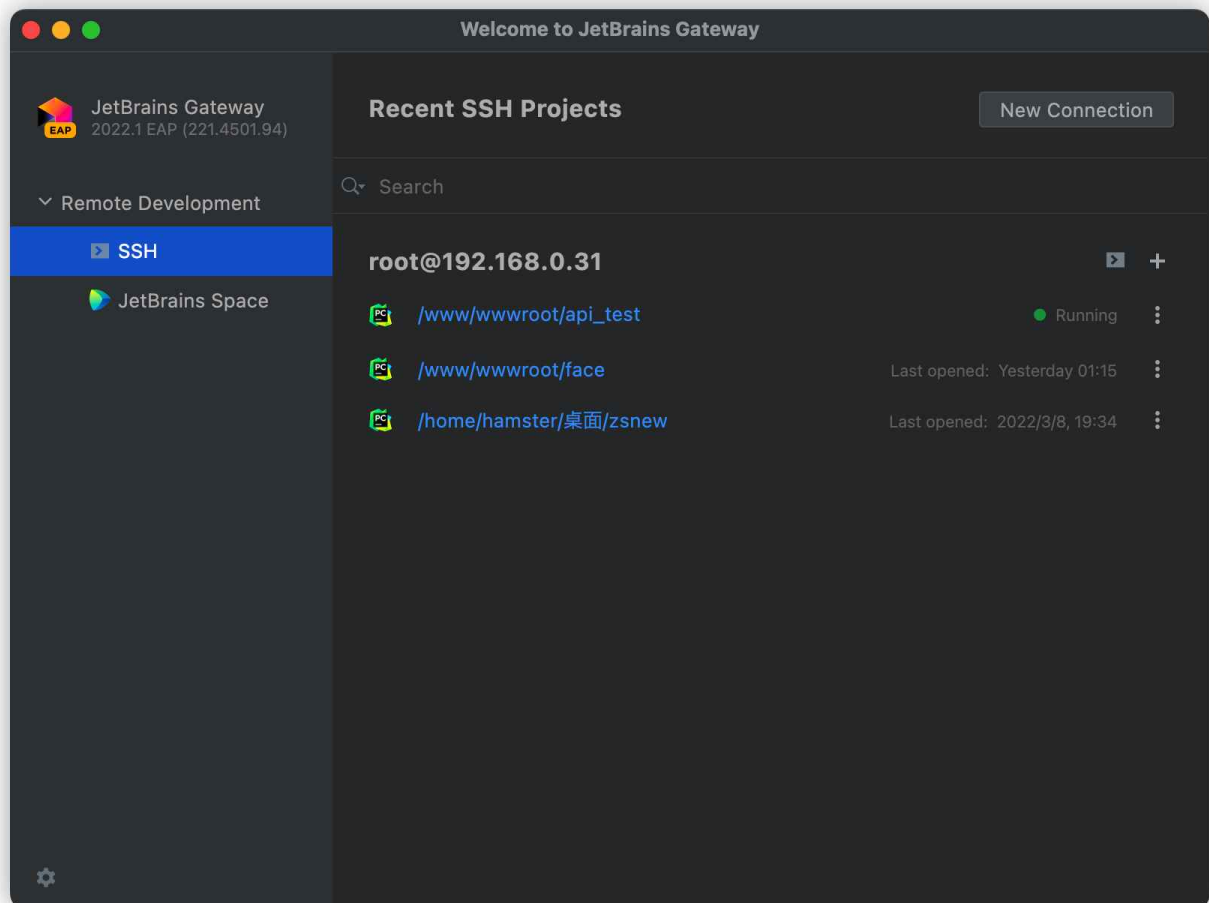
配置完成后就可以打开Gateway进行服务器的连接



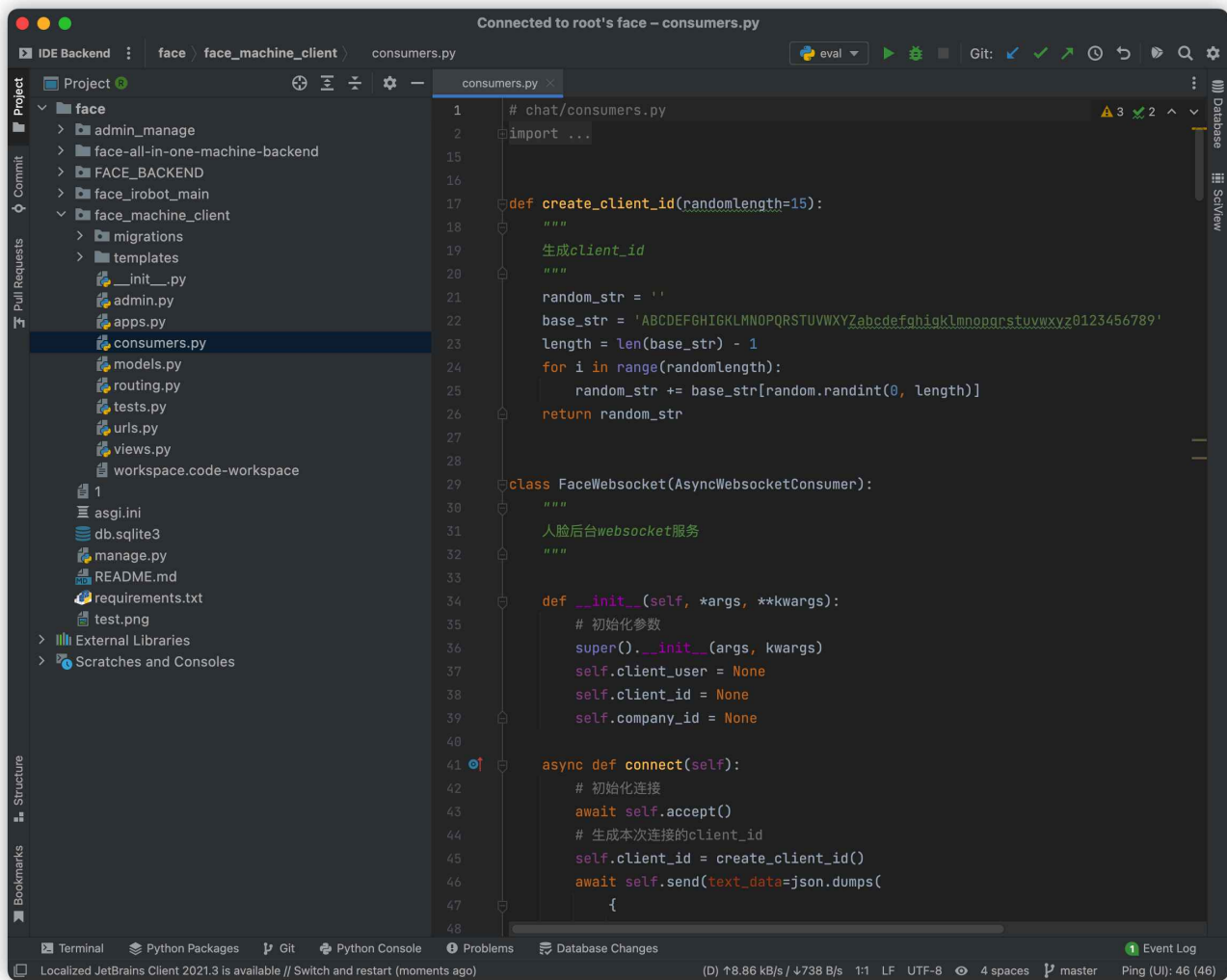
我们主要需要填入服务器的IP地址以及root账号与密码。



选择PycharmIDE进行安装，等待安装完成自动运行。



成功安装后便可以在软件中看到映射的项目了，点击打开项目。



！ 注意，可能Gateway会出现找不到Python版本的情况，需要我们手动绑定Python

## 服务器Python与GCC配置

后端采用centos系统下的宝塔面板进行服务的部署，在部署前如果需要进行调试需要对服务器进行环境的搭建。

主要分为两步，安装Python3.9，增加软连接。第二步为升级GCC版本进行Dlib的编译安装。

## 安装Python

采用源码编译安装的方式。

先安装一些依赖的包：

Ruby

```
1 [root@centos7 ~]$ yum install gcc openssl-devel bzip2-devel libffi-devel -y
2 # 下载python源码包
3 [root@centos7 ~]$ wget
   https://registry.npmmirror.com/-/binary/python/3.9.9/Python-3.9.9.tgz
```

提取文件:

Python

```
1 [root@centos7 ~]$ tar -zxvf Python-3.9.9.tgz
```

准备从源代码编译Python:

Apache

```
1 cd Python-3.9.9 && ./configure prefix=/usr/local/python3
2
3 ## 等待完成
4
5 make && make install
```

提示:

SQL

```
1 WARNING: The script easy_install-3.9 is installed in '/usr/local/python3/bin'
  which is not on PATH.
2 Consider adding this directory to PATH or, if you prefer to suppress this
  warning, use --no-warn-script-location.
3 WARNING: The scripts pip3 and pip3.9 are installed in '/usr/local/python3/bin'
  which is not on PATH.
4 Consider adding this directory to PATH or, if you prefer to suppress this
  warning, use --no-warn-script-location.
```

提示我们安装的脚本不在路径中, 下面创建链接。

Bash


```
1 vim ~/.bash_profile
2 source ~/.bash_profile
```

在vim ~/.bash\_profile文件中加入

Apache

```
1 alias python3=/usr/local/python3/bin/python3
2 alias pip3=/usr/local/python3/bin/pip3
```

在终端中进行测试, 查看python与pip版本是否正确。

 如果出现pip3无法打开的情况

no pip3 in 或者 bash: pip3: 未找到命令...\_划船的使者的博客-CSDN博客\_pip3未找到命令

1、whereis python3[wcy@localhost ~]\$ whereis python3python3: /usr/bin/python3  
/usr/local/python3[wcy@localhost ~]\$ cd /usr/local/python3[wcy@localhost python3]\$ lsbin include li...

 [https://blog.csdn.net/weixin\\_42185136/article/details/90636750](https://blog.csdn.net/weixin_42185136/article/details/90636750)

## 升级GCC

在终端中:

SQL

```
1 yum install centos-release-scl
2 yum install devtoolset-7-gcc*
```

设置devtoolset-7 为默认的gcc编译器

Bash

```
1 scl enable devtoolset-7 bash
2 vim /etc/profile
3 source /opt/rh/devtoolset-7/enable
4
5 gcc --v
6 which gcc
7 # gcc version 7.3.1 20180303 (Red Hat 7.3.1-5) (GCC)
8 # 看到gcc 为 7.3.1则为安装完成
```

## 编译安装Dlib

完成上面的步骤后开始安装Dlib

在终端中输入

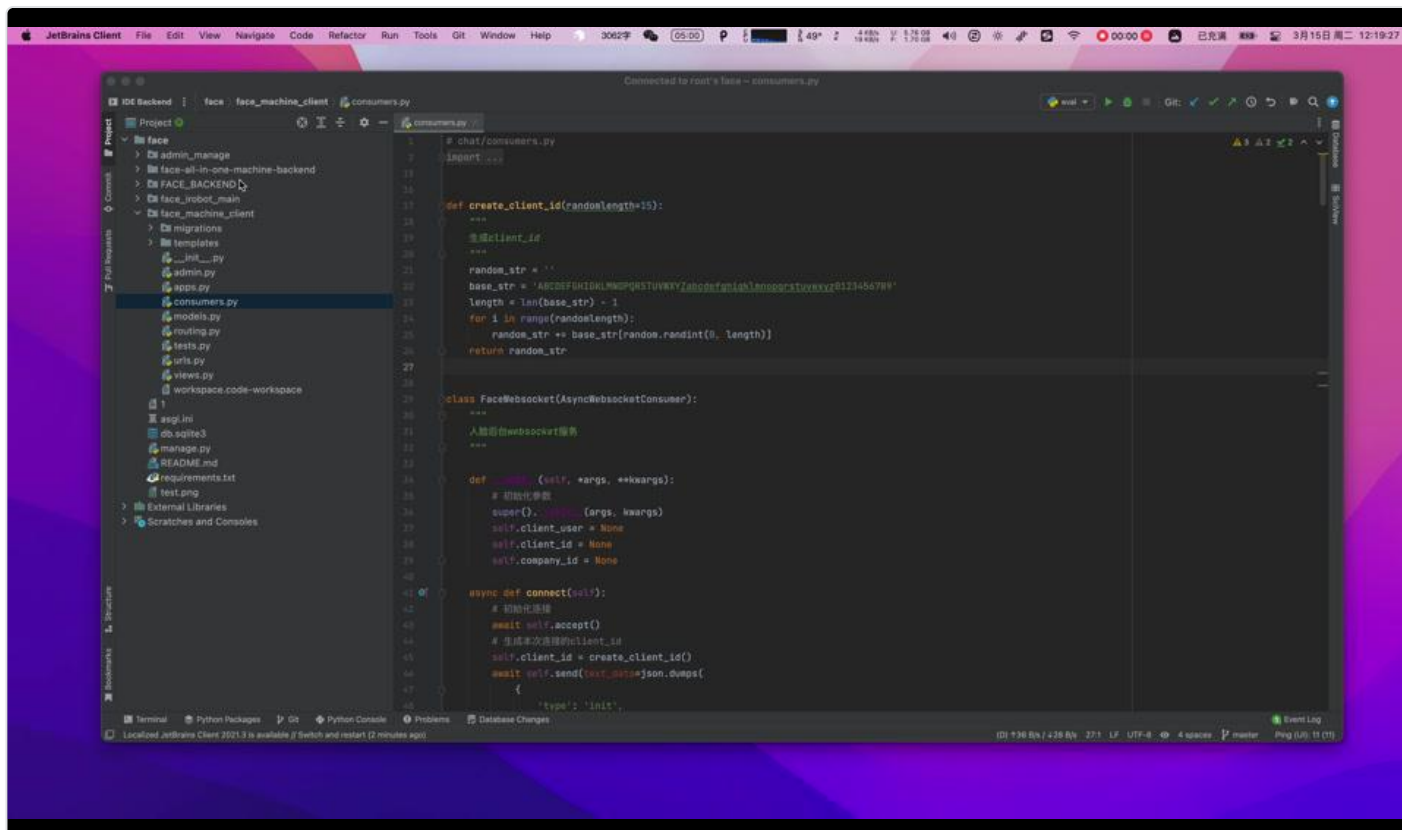
Bash

```
1 export CC=/usr/local/bin/gcc
2 export CXX=/usr/local/bin/g++
3
4 pip3 install dlib
```

! 编译安装dlib的时间较长，等待一下就好，摸鱼time

## 手动绑定Python

在Gateway设置中，重新指定Python路径与版本

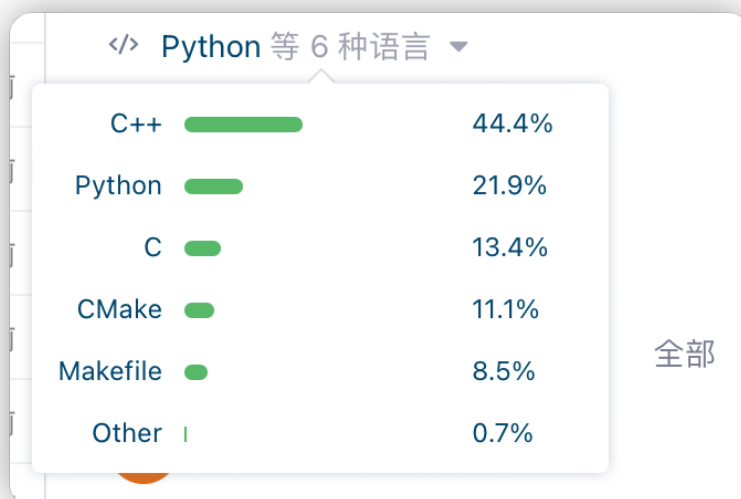


iShot2022-03-15 12.19.26.mov

## 搭建seetaface环境

### 人脸机后台简介

对于人脸机后台的开发，用到的语言比例可以参考一下下图



可以看到在项目中,c++在代码占据了不小的分量,因此在部署服务器环境中,需要学习到一些编译以及动态库引用的知识,操作都是偏向基础项的,因此难度并不大。

人脸机项目中,人脸识别的算法部分采用了开源的Seetaface作为算法基础,在Seetaface的基础上进行更多功能的开发,因此对于c++的基础语法的了解也是必须的。

github.com

<https://github.com/SeetaFace6Open/index>

## 模型下载

模型可以在seetaface的官网下载:

模型文件:

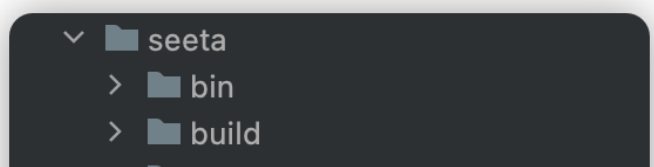
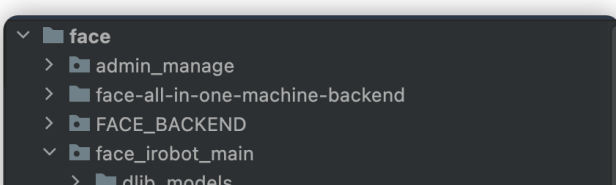
Part I: [Download](#) code: `ngne`, including: `age_predictor.csta`,  
`face_landmarker_pts5.csta`, `fas_first.csta`, `pose_estimation.csta`,  
`eye_state.csta`, `face_landmarker_pts68.csta`, `fas_second.csta`,  
`quality_lbn.csta`, `face_detector.csta`, `face_recognizer.csta`,  
`gender_predictor.csta`, `face_landmarker_mask_pts5.csta`,  
`face_recognizer_mask.csta`, `mask_detector.csta`.

Part II: [Download](#) code: `t6j0`, including: `face_recognizer_light.csta`.

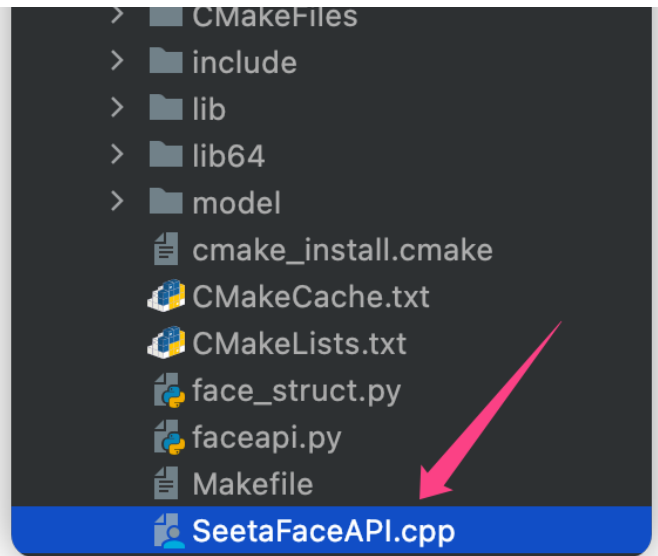
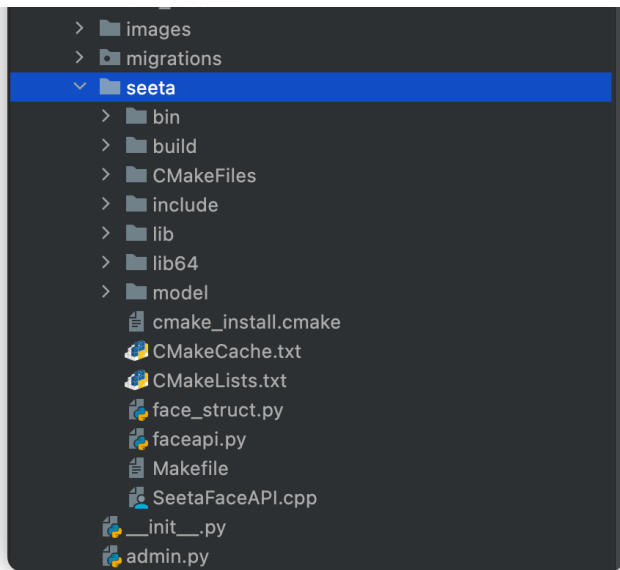
人脸识别采用 `face_recognizer_light.csta` 作为后端与消费机模型。

## 编译

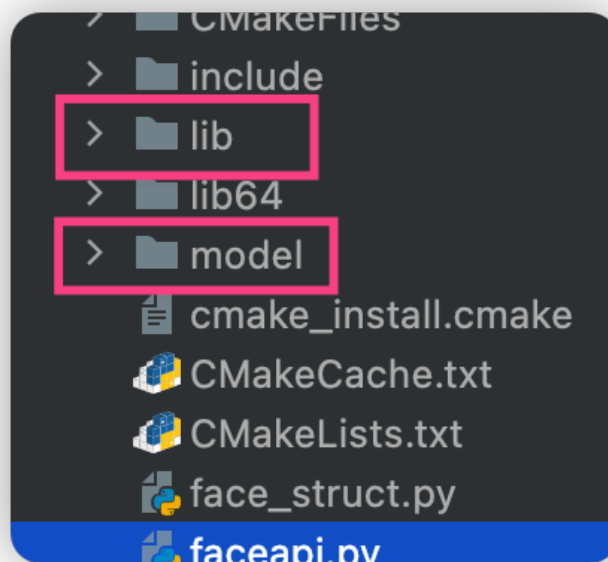
进入Django项目下,到目录face\_irobot\_main下,找寻到seeta文件夹进入







检查SeetaFaceAPI.cpp文件，与lib与model文件夹下是否已经放置好需要的基础库与模型文件



## 开始编译

打开终端，cd到face\_irobot\_main文件夹

在终端中输入

## Bash

```
1 cd seeta
2 mkdir build && cd build
3 cmake ..
4 make
```

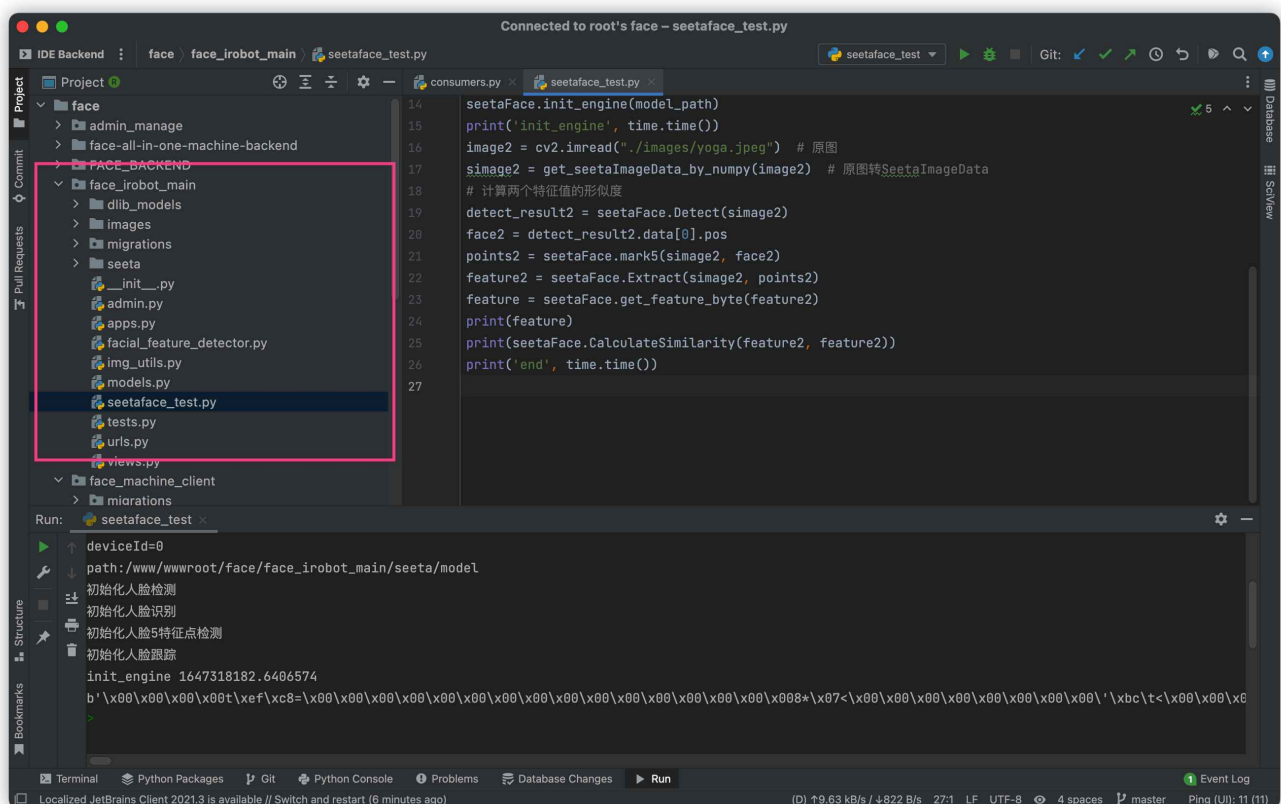
将在lib文件夹下产生libSeetaFaceAPI.so文件

## Bash

```
1 sudo echo ${项目路径}/face_irobot_main/seeta/lib/ >
  /etc/ld.so.conf.d/seetaface6.conf
2 sudo ldconfig
```

## 测试代码运行状态

在人脸后端项目中，在face\_irobot\_main中有一个seetaface\_test.py文件，运行环境搭建好后运行，如果在下面的Run中正确显示信息，则证明开发服务器搭建完成。



## Git使用

在多人协作的项目中，Git工具的使用是必不可少的，一方面采用Git可以快速方便的同步我们的代码，另一方面我们也可以随时查看其他成员对于项目代码的更改，更好的进行更新与debug。

由于github网络经常抽风,(特别是即使在科学上网后)。因此我们采用国内的gitee进行代码的管理

## gitee码云完整使用教程(部署与克隆)\_柯晓楠的博客-CSDN博客\_gitee使用教程

1.创建仓库登录码云 <https://gitee.com/>创建一个仓库2.使用git在本地初始化(1)新建一个目录，存放下载下来的项目，我在D盘新建了一个“gitspace”文件夹，用来存放下载下来的项目(2)进入刚刚新建的文件夹，即…

<https://blog.csdn.net/p445098355/article/details/104766195>

非常的好学好用好看的在线Git教程

## Learn Git Branching

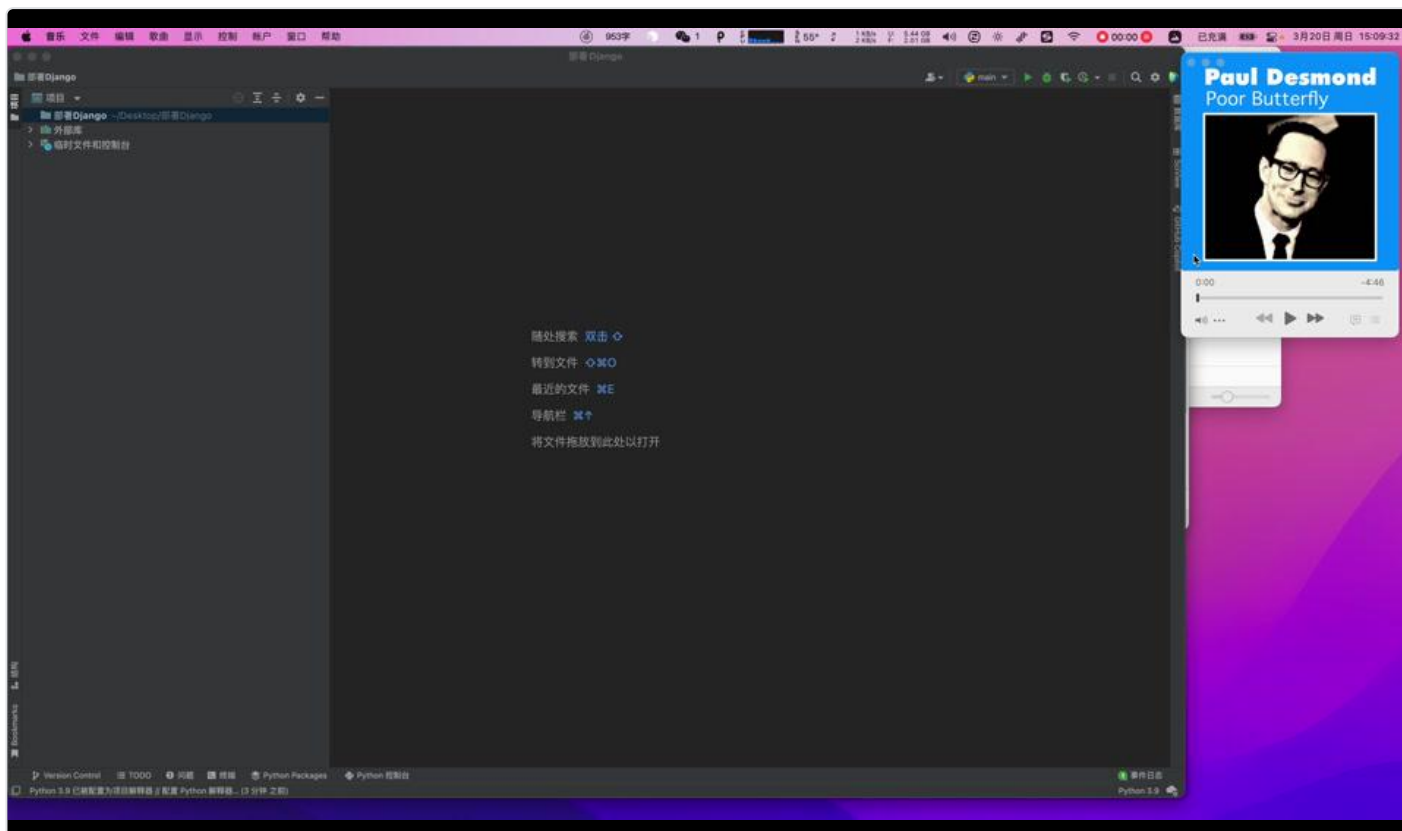
An interactive Git visualization tool to educate and challenge!

<https://learngitbranching.js.org/>

! 千万千万千万，不要force-push，无论什么情况下都不要强制推送

## 搭建Django项目

### 完整视频



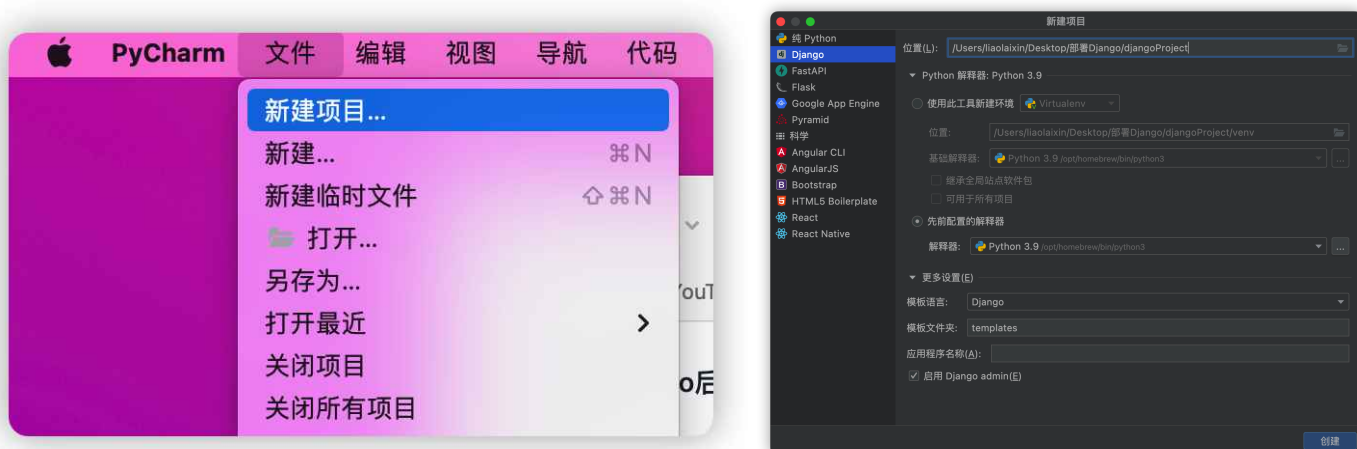
deploy.mp4

## 创建Django项目

在菜单栏中选择新建项目，在位置中选择相应的位置

在本地开发时可直接选择本地的Python解释器

等待新窗口出现即创建成功



## 一些必须更改的设置

设置debug模式为True, ALLOWED\_HOSTS = ['\*']

### Python

```
1  DEBUG = True
2  ALLOWED_HOSTS = ['*']
```

## 取消本地跨域调试限制

### Nginx

```
1  MIDDLEWARE = [
2      'django.middleware.security.SecurityMiddleware',
3      'django.contrib.sessions.middleware.SessionMiddleware',
4      'django.middleware.common.CommonMiddleware',
5      # 'django.middleware.csrf.CsrfViewMiddleware', <-----注释这一行
6      'django.contrib.auth.middleware.AuthenticationMiddleware',
7      'django.contrib.messages.middleware.MessageMiddleware',
8      'django.middleware.clickjacking.XFrameOptionsMiddleware',
9  ]
```

## 设置服务器跨域django-corsheaders

首先安装

### Python

```
1  pip3 django-corsheaders
```

将corsheaders放在所有模块的上方，并将图中的中间件放在相应的位置

Python

```
1 'corsheaders.middleware.CorsMiddleware',
```

```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
    'corsheaders',  
    'User_manage',  
    'Company_detail',  
    'User_detail',  
    'Search_Test'  
]  
  
MIDDLEWARE = [  
    'django.middleware.security.SecurityMiddleware',  
    'django.contrib.sessions.middleware.SessionMiddleware',  
    'corsheaders.middleware.CorsMiddleware',  
    'django.middleware.common.CommonMiddleware',  
    # 'django.middleware.csrf.CsrfViewMiddleware',  
    'django.contrib.auth.middleware.AuthenticationMiddleware',  
    'django.contrib.messages.middleware.MessageMiddleware',  
    'django.middleware.clickjacking.XFrameOptionsMiddleware',  
]
```

在setting.py文件底部加上

## HTTP

```
1
2
3  CORS_ALLOW_CREDENTIALS = True
4  CORS_ORIGIN_ALLOW_ALL = True
5  CORS_ORIGIN_WHITELIST = ()
6
7  CORS_ALLOW_METHODS = (
8      'DELETE',
9      'GET',
10     'OPTIONS',
11     'PATCH',
12     'POST',
13     'PUT',
14     'VIEW',
15 )
16
17 CORS_ALLOW_HEADERS = (
18     'accept',
19     'accept-encoding',
20     'authorization',
21     'content-type',
22     'dnt',
23     'origin',
24     'user-agent',
25     'x-csrftoken',
26     'x-requested-with',
27 )
```

## 本地开发设置数据库为路径下sqlite数据库

### Bash

```
1  DATABASES = {
2      'default': {
3          'ENGINE': 'django.db.backends.sqlite3',
4          'NAME': BASE_DIR / 'db.sqlite3',
5      }
6  }
```

## 服务器开发设置数据库为MySQL数据库

Bash

```
1 DATABASES = {  
2     'default': {  
3         'ENGINE': 'django.db.backends.mysql',  
4         'NAME': 'stock_db',  
5         'USER': '',  
6         'PASSWORD': '',  
7         'HOST': '120.24.211.49',  
8         'PORT': '3306',  
9     },  
10  
11 }
```

设置时区（非常重要!!!）

YAML

```
1 LANGUAGE_CODE = 'en-us'  
2  
3 TIME_ZONE = 'Asia/Shanghai'  
4  
5 USE_I18N = True  
6  
7 USE_L10N = True  
8  
9 USE_TZ = False
```

进行数据库的合并，第一次运行指令为创建默认数据库

在项目文件夹中打开终端，在终端中输入

Bash

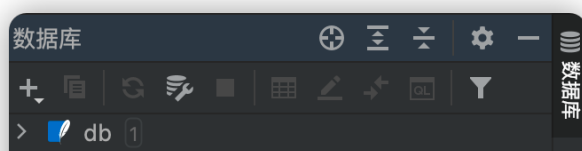
```
1 python3 manage.py migrate
```

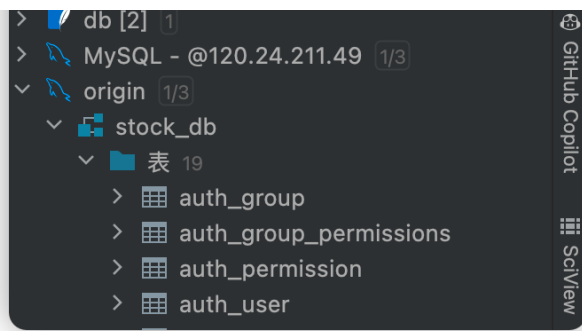
大多数终端中关于Django的指令都是围绕manage.py展开

如 合并数据库 python3 manage.py migrate

生成数据表 python3 manage.py makemigrations [模块名]

如果是设置了本地sqlite数据库,则会在本地生成一个sqlite3数据库





如果是设置在服务器MySQL数据库，则会在服务器中创建Django的默认表名以及默认的数据

如果采用MySQL,首先需要安装pymysql库来进行Django,需要在Django项目中采用pymysql来管理MySQL数据库

安装pymysql

在\_\_init\_\_.py文件中加入下面语句即可

Python

```
1 import pymysql
2
3 pymysql.install_as_MySQLdb()
```

## 创建模块

📌 Django的思想类似于黑胶储物架，当你需要这个模块的时候，就将这个模块放进setting.py文件的INSTALLED\_APPS中，模块的名称就对应模块文件夹的名称。

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'Pull_data',
]
```

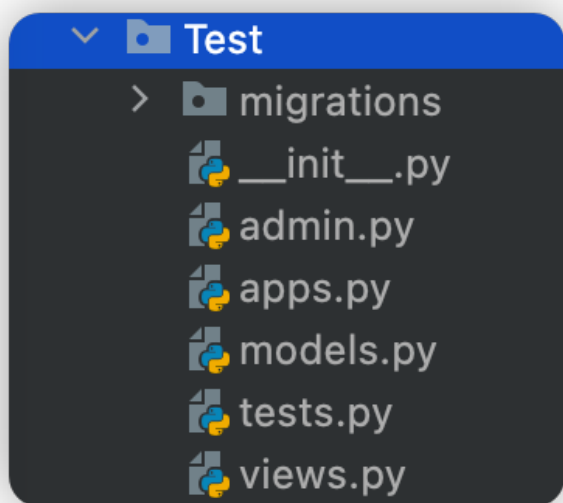


在项目终端中输入

Nginx

```
1 python3 manage.py startapp [Test] <--- 模块名称
```

输入指令后默认会生成一个以模块名称为名字的文件夹，里面有模块对应的基础文件。



## 增加urls文件

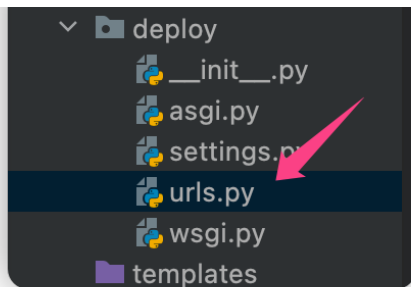
Django中的路由管理是由urls.py文件来进行管理的。

我们采用分模块负责各自的urls，也就是接口路径的方法来进行管理。

在项目主文件夹下管理urls.py

deploy ~/Desktop/部署

```
from django.contrib import admin
```



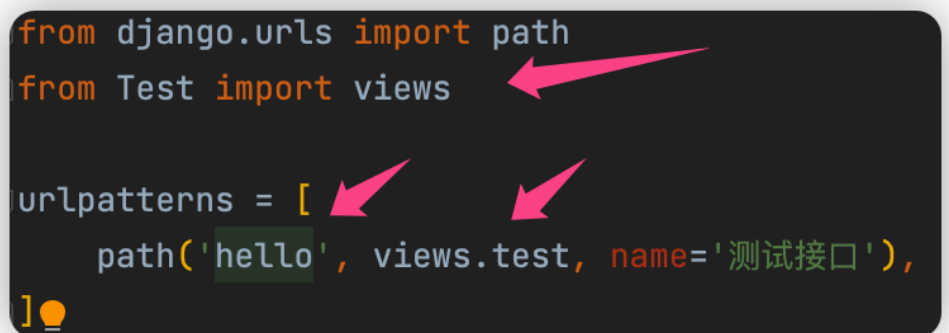
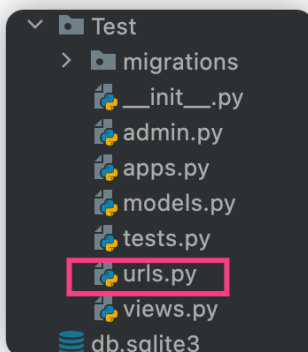
引入include库，便于分模块管理，如右图在admin/下新开path，第一个冒号对应的是服务器域名后对应的接口路径，采用include则表明api/v1/test后的路径部分由Test模块下的urls文件进行管理。

在模块文件夹下新建urls.py文件,在文件中添加路径

打开新建的urls.py文件，首先引入path库，接着我们需要引入模块的views文件，views文件主要内容就是为我们的接口函数。

在下方添加urlpatterns，参考右图，第一个为本模块在主路径（api/v1/test）后的子路径

第二个则为对应的接口函数



## 编写接口

在模块的views.py文件夹内编写接口

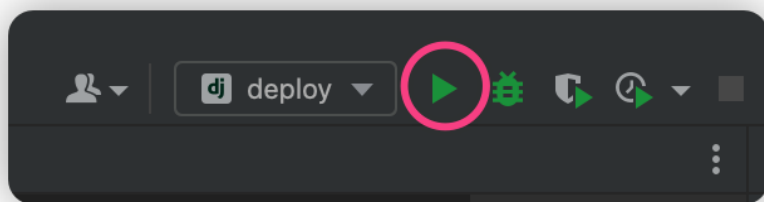
如图所示，首先定义接口函数的名称，对应的为上图路径中的接口函数名称，在传参为request，用于捕获请求，最后return指通常采用JsonResponse来进行json数据的封装。

```
from django.http import JsonResponse

def test(request):
    return JsonResponse({'msg': 'hello world'})
```

## 测试接口

### 运行Django服务器

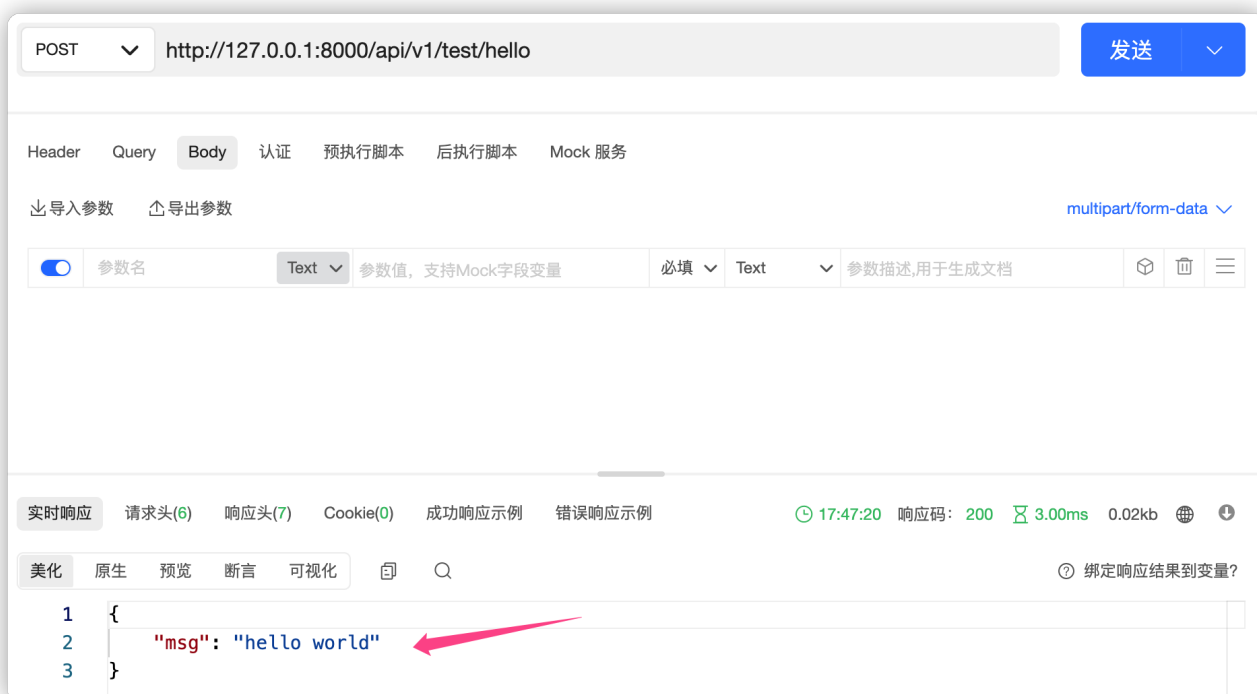


```
/opt/homebrew/bin/python3 /Users/liaolaixin/Desktop/部署
Performing system checks...

Watching for file changes with StatReloader
System check identified no issues (0 silenced).
March 21, 2022 - 17:44:42
Django version 3.2.10, using settings 'deploy.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

可以采用apipost来进行测试

根据上面的路径，我们可以通过<http://127.0.0.1:8000/api/v1/test/hello>来测试接口是否正常。



可以看到接口返还正常。

## 增加模块数据库

Django的一大方便之处就是集成了非常好用的ORM来帮助我们去管理数据库

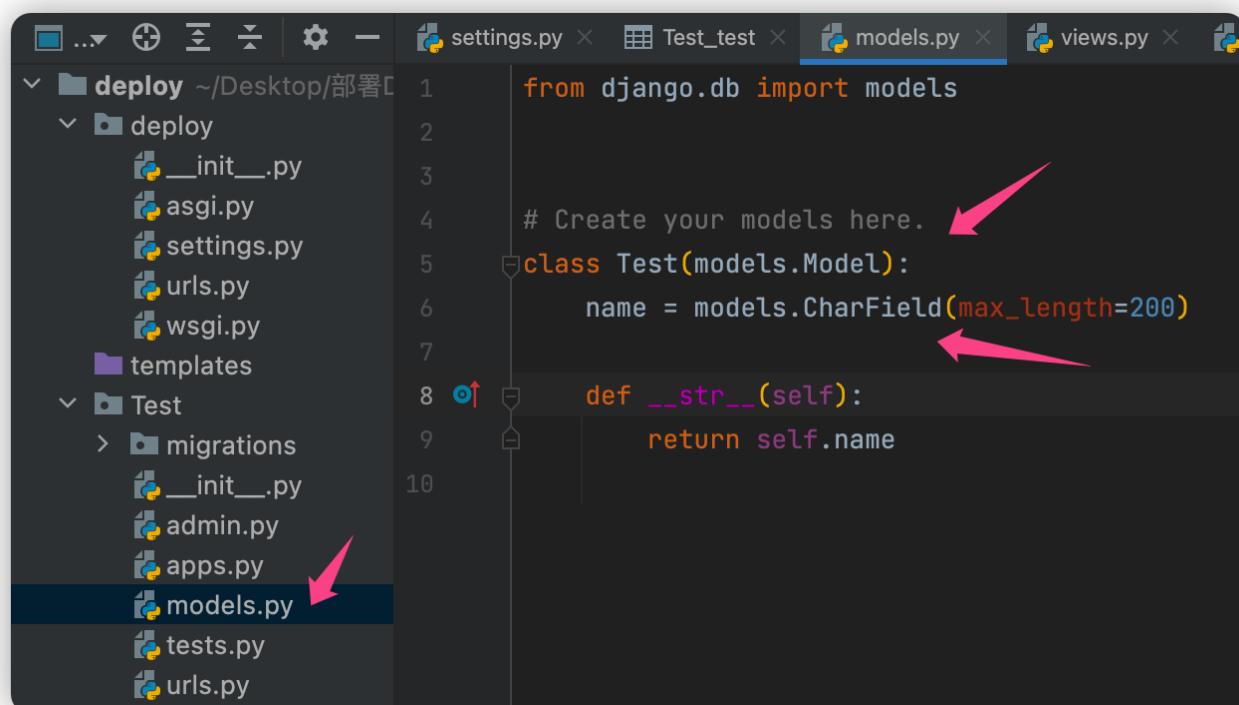
每个模块的数据库模型储存在models.py文件中，如下图

我们采用class 【表名】作为一个表来进行管理，如下图就会创建一个名为Test的表

class中的内容，左侧则为表里的字段名称，右侧为该字段的属性。

如图，创建了一个name字段，是属于字符串字段，设置max\_length最长长度为200

📌 在sqlite中，max\_length属性无效，在MySQL数据库中生效，注意一下这个哈，踩过坑...



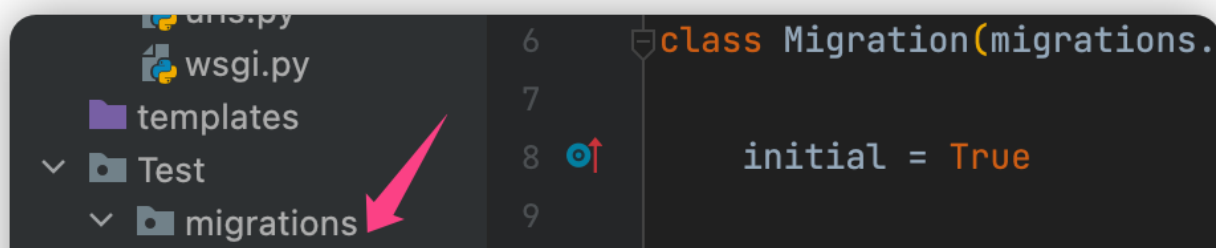
编写完成后，采用合并的方式，将模型通过Django来创建到数据库中

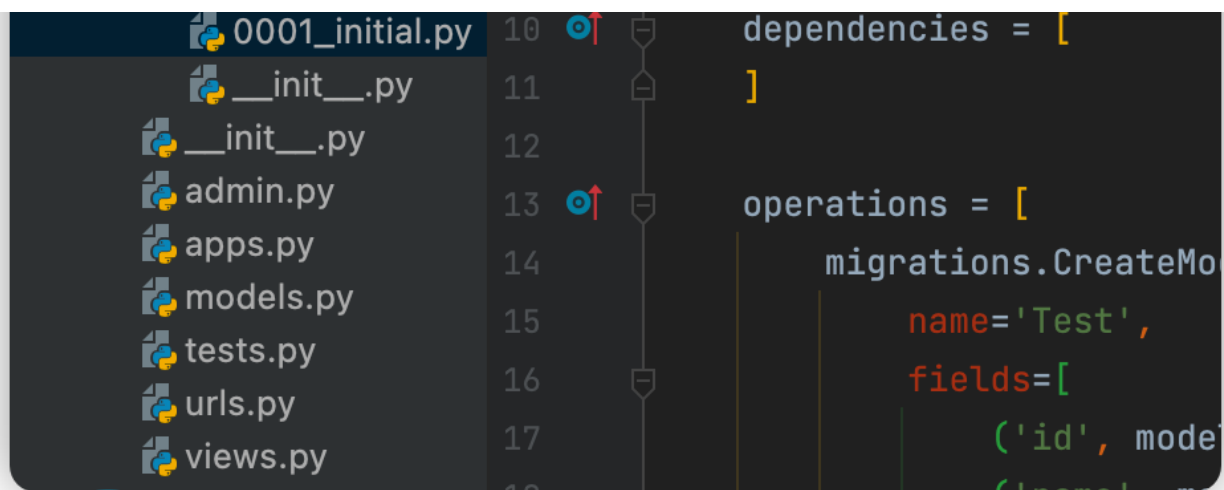
Python

```
1 python3 manage.py makemigrations [模块名]
```

models属于哪一个模块，就填入那个模块的文件夹名。

成功后应该会在模块的migrations文件夹中看到更改



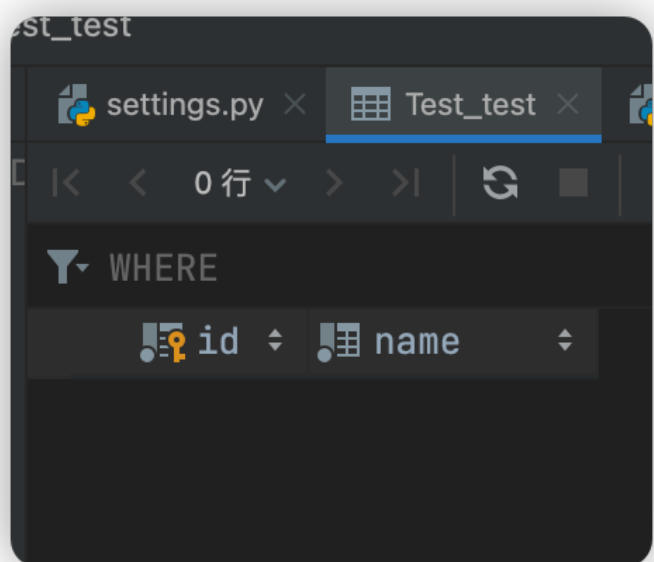
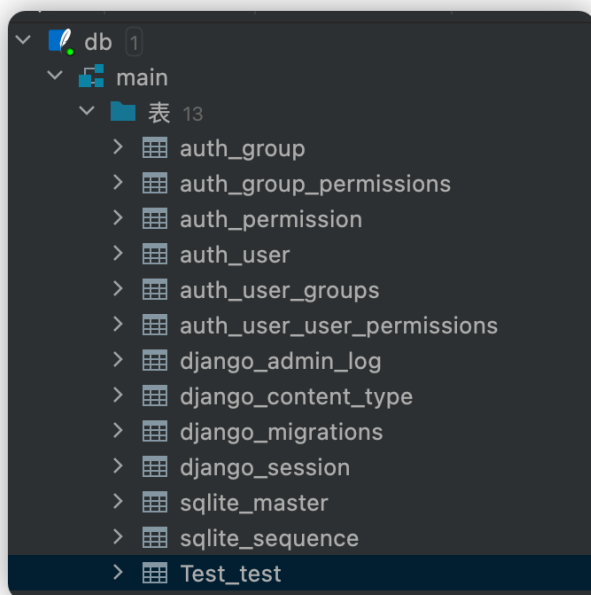


如果正常则进行合并到数据库的操作

Python

```
1 python3 manage.py migrate
```

成功后即可在数据库中看到对应的表以及字段



Django ORM调用数据库

## Django ORM常用操作介绍(新手必看)-Django社区,Django中文网,django教程,Django!

Django开发过程中对表(model)的增删改查是最常用的功能之一，本文介绍笔者在使用model 操作过程中遇到的一些操作

<https://www.django.cn/article/show-15.html>

### Django ORM反向生成模型

Bash

```
1 默认default
2  python3 manage.py inspectdb
3  指定到应用中的models
4  python3 manage.py inspectdb > app/models.py
5  指定某个数据库中某张表
6  python3 manage.py inspectdb --database default table_name > app/models.py
```

2022.3.21