0 环境配置

```
# 创建python3.8环境
conda create -n python38 python3.8.3

# 进入项目，创建venv
virtualenv -p "D:/Anaconda/envs/python38/python.exe" venv
# venv激活
【记得每次进入之后都要激活！！，自己看现在的python环境就知道了】
source ./venv/Scripts/activate

# 安装指定依赖
pip install -r requirements.txt

【记得vscode切换一下编译器环境！！】

# 填充fake数据
python manage.py init_db
python manage.py runserver

# 下载与本地chrome版本匹配的chromedriver，建议放置与chrome同目录下
# 将chrome.exe 和 chromedriver.exe路径进行修改

# 运行所有测试
python manage.py test
```

console输出正常应该如下图所示：

```
hhhhzz@LAPTOP-0MSEQ3OU MINGW64 /d/大四课程/软工/清软论坛单元练习 (master)
$ python manage.py test
test_login (test_api.APITestCase)
TODO: 使用错误的信息进行登录，检查返回值为失败 ... ok
test_logout (test_api.APITestCase)
TODO: 未登录直接登出 ... ok
test_register (test_api.APITestCase)
Example: 使用错误信息进行注册，检查返回值为失败 ... ok
test_register_params_check (test_basic.BasicTestCase) ... ok
test_web (test_e2e.SeleniumTestCase)
EXAMPLE: 使用测试用户进行登录 ... ok

----------------------------------------------------------------------
Ran 5 tests in 2.944s

OK
Coverage:
Name                        Stmts   Miss Branch BrPart  Cover
-------------------------------------------------------------
app\__init__.py                19      9      4      1    57%
app\checkers\user.py            2      1      0      0    50%
app\controllers\hello.py       22     20      0      0     9%
app\controllers\user.py        59     51     14      2    14%
app\services\user.py           34     32      4      0     5%
app\utils\config.py            12      8      4      0    38%
app\utils\jwt.py               27     24      2      0    10%
app\utils\middleware.py        11      7      4      1    33%
-------------------------------------------------------------
TOTAL                         186    152     32      4    19%
HTML version: file://D:\大四课程\软工\清软论坛单元练习\test_report/index.html
(venv)
```

1 代码风格测试

1.1 完善flake8配置

官方文档：flake8

一般的代码风格测试工具都会与语言版本相关，所以需要根据要求首先安装好指定的语言环境 **python3.8**

```
# 进入虚拟环境！！
source ./venv/Scripts/activate

# 根据当前环境安装对应版本的flake8
python -m pip install flake8

# 检查版本是否匹配
flake8 --version
```

文件配置`.flake8`

```
[flake8]
# TODO: 补充flake8配置文件

# 忽略且仅忽略.git, __pycache__, venv文件夹
exclude =
    .git,
    __pycache__,
    venv

# app/services/post.py, app/services/user.py, tests/test_e2e.py,
tests/test_api.py忽略E501错误
per-file-ignores =
    app/services/post.py:E501
    app/services/user.py:E501
    tests/test_e2e.py:E501
    tests/test_api.py:E501
```

代码风格检查（不会进行修复~，后续会有工具进行修复）

```
flake8 .
```

显示结果如下：

```
hhhhzz@LAPTOP-0MSEQ3OU MINGW64 /d/大四课程/软工/清软论坛单元练习 (master)
$ flake8 .
.\driver.py:21:15: W292 no newline at end of file
.\manage.py:10:1: F401 'sys' imported but unused
.\manage.py:14:4: E225 missing whitespace around operator
.\manage.py:14:34: E231 missing whitespace after ','
.\manage.py:27:1: E302 expected 2 blank lines, found 1
.\manage.py:43:1: E302 expected 2 blank lines, found 1
.\manage.py:47:156: E261 at least two spaces before inline comment
.\manage.py:51:1: E305 expected 2 blank lines after class or function definition, found 1
.\test.py:6:1: E302 expected 2 blank lines, found 1
.\test.py:7:3: E111 indentation is not a multiple of 4
.\test.py:12:7: E111 indentation is not a multiple of 4
.\app\controllers\user.py:126:58: W292 no newline at end of file
.\app\services\user.py:8:1: E302 expected 2 blank lines, found 1
.\app\services\user.py:11:42: E225 missing whitespace around operator
.\app\services\user.py:22:17: E128 continuation line under-indented for visual indent
.\app\services\user.py:23:17: E128 continuation line under-indented for visual indent
.\app\services\user.py:34:53: E225 missing whitespace around operator
.\app\services\user.py:34:78: E225 missing whitespace around operator
.\app\services\user.py:40:35: W292 no newline at end of file
.\app\utils\config.py:18:1: E305 expected 2 blank lines after class or function definition, found 1
.\app\utils\jwt.py:4:1: F401 'hashlib' imported but unused
.\app\utils\jwt.py:14:1: W293 blank line contains whitespace
.\app\utils\jwt.py:32:1: E302 expected 2 blank lines, found 1
.\app\utils\jwt.py:47:1: E302 expected 2 blank lines, found 1
.\app\utils\middleware.py:5:1: E302 expected 2 blank lines, found 1
.\tests\test_api.py:5:16: E401 multiple imports on one line
.\tests\test_api.py:9:1: E302 expected 2 blank lines, found 1
.\tests\test_api.py:41:5: E303 too many blank lines (2)
.\tests\test_api.py:45:27: E231 missing whitespace after ':'
.\tests\test_api.py:65:1: E305 expected 2 blank lines after class or function definition, found 1
.\tests\test_api.py:66:20: W292 no newline at end of file
.\tests\test_e2e.py:32:38: W605 invalid escape sequence '\P'
.\tests\test_e2e.py:32:52: W605 invalid escape sequence '\G'
.\tests\test_e2e.py:32:59: W605 invalid escape sequence '\C'
.\tests\test_e2e.py:32:66: W605 invalid escape sequence '\A'
.\tests\test_e2e.py:32:78: W605 invalid escape sequence '\c'
.\tests\test_e2e.py:35:16: W605 invalid escape sequence '\P'
.\tests\test_e2e.py:35:30: W605 invalid escape sequence '\G'
.\tests\test_e2e.py:35:37: W605 invalid escape sequence '\C'
.\tests\test_e2e.py:35:44: W605 invalid escape sequence '\A'
.\tests\test_e2e.py:35:56: W605 invalid escape sequence '\c'
(venv)
```

可以看到的确忽略了指定文件的代码风格检查，以及忽略了指定文件的指定错误；并且该工具在代码提示上指定了明确的某一行代码，以及是什么样的问题，还是很不错的~

1.2 完善格式化脚本lint.sh

- autopep8
- autoflake
- isort

```
# TODO: add autopep8 here.
autopep8 --recursive --in-place --aggressive --exclude venv,.git,__pycache__ .

# TODO: add autoflake here.
autoflake --recursive --in-place --exclude venv,.git,__pycache__ .

# TODO: add isort here.
isort .

# TODO: add flake8 here.
flake8 .
```

注意，需要加上`--recursive`参数才能都对该项目所有文件进行格式化处理

> -r, --recursive： run recursively over directories; must be used with --in-place or --diff

```
hhhhzz@LAPTOP-0MSEQ3OU MINGW64 /d/大四课程/软工/清软论坛单元练习 (master)
$ chmod +x lint.sh
(venv)
hhhhzz@LAPTOP-0MSEQ3OU MINGW64 /d/大四课程/软工/清软论坛单元练习 (master)
$ ./lint.sh
(venv)
hhhhzz@LAPTOP-0MSEQ3OU MINGW64 /d/大四课程/软工/清软论坛单元练习 (master)
$ echo $?
0
(venv)
```

注意，还需要排除对应文件修改，特别是`venv`，发现autoflake在修改了`venv`之后出现包循环import的问题，如下图所示。可以重装环境并且正确配置好`lint.sh`解决

```
hhhhzz@LAPTOP-0MSEQ3OU MINGW64 /d/大四课程/软工/清软论坛单元练习 (master)
  File "D:\大四课程\软工\清软论坛单元练习\venv\lib\site-packages\pip\_internal\cli\cmdoptions.py", line 24, in <module
>
    from pip._internal.cli.parser import ConfigOptionParser
  File "D:\大四课程\软工\清软论坛单元练习\venv\lib\site-packages\pip\_internal\cli\parser.py", line 12, in <module>
    from pip._internal.configuration import Configuration, ConfigurationError
  File "D:\大四课程\软工\清软论坛单元练习\venv\lib\site-packages\pip\_internal\configuration.py", line 20, in <module>

    from pip._internal.exceptions import (
  File "D:\大四课程\软工\清软论坛单元练习\venv\lib\site-packages\pip\_internal\exceptions.py", line 13, in <module>
    from pip._vendor.requests.models import Request, Response
  File "D:\大四课程\软工\清软论坛单元练习\venv\lib\site-packages\pip\_vendor\requests\__init__.py", line 45, in <modul
e>
    from .exceptions import RequestsDependencyWarning
  File "D:\大四课程\软工\清软论坛单元练习\venv\lib\site-packages\pip\_vendor\requests\exceptions.py", line 9, in <modu
le>
    from .compat import JSONDecodeError as CompatJSONDecodeError
ImportError: cannot import name 'JSONDecodeError' from 'pip._vendor.requests.compat' (D:\大四课程\软工\清软论坛单元练
习\venv\lib\site-packages\pip\_vendor\requests\compat.py)
(venv)
```

可以看到，配置之后在vscode源代码管理**更改区**，有着被格式化的文件，并且使用`echo $?`指令返回0，证明了执行`./lint.sh`没有出错。

过程中发现运行`autopep8`出现gbk编码错误，如下图所示：

```
hhhhzz@LAPTOP-0MSEQ3OU MINGW64 /d/大四课程/软工/清软论坛单元练习 (master)
$ autopep8 .
Traceback (most recent call last):
  File "D:\Anaconda\envs\python38\lib\runpy.py", line 194, in _run_module_as_main
    return _run_code(code, main_globals, None,
  File "D:\Anaconda\envs\python38\lib\runpy.py", line 87, in _run_code
    exec(code, run_globals)
  File "D:\大四课程\软工\清软论坛单元练习\venv\Scripts\autopep8.exe\__main__.py", line 7, in <module>
  File "D:\大四课程\软工\清软论坛单元练习\venv\lib\site-packages\autopep8.py", line 4519, in main
    args = parse_args(argv[1:], apply_config=apply_config)
  File "D:\大四课程\软工\清软论坛单元练习\venv\lib\site-packages\autopep8.py", line 3864, in parse_args
    parser = read_config(args, parser)
  File "D:\大四课程\软工\清软论坛单元练习\venv\lib\site-packages\autopep8.py", line 4015, in read_config
    if config.read([os.path.join(parent, fn)
  File "D:\Anaconda\envs\python38\lib\configparser.py", line 696, in read
    self._read(fp, filename)
  File "D:\Anaconda\envs\python38\lib\configparser.py", line 1016, in _read
    for lineno, line in enumerate(fp, start=1):
UnicodeDecodeError: 'gbk' codec can't decode byte 0xb9 in position 94: illegal multibyte sequence
(venv)
```

经查阅资料，找到相应位置，将`encoding=encoding`一行强制更换为`encoding='utf-8'`即可。

2 单元测试

2.1 补充基础函数

字段检查模块：简单地判断字典是否存在对应key值，其中如果`magic_number`为空，补充默认值0

用户账号检查模块：

- 长度

- 都包含字母和数字，使用`isalnum`，`isdigit`，`isalpha`
- 找到第一个字母和数字分界，分割前后字串，判断前字串为纯字母且不全是大写或小写，判断后字串为数字

用户密码检查模块：使用正则表达式判断

- 首先限制只能有数字、大小写字母、指定特殊字符，限制长度

```
pattern1 = r"^[A-Za-z0-9\-_\*\^]{8,15}$"
```

- 保证数字、大小写、特殊字符都存在

```
pattern2 = r"""^(?![A-Za-z0-9]+$)(?![a-z0-9\-_\*\^]+$)
(?![A-Za-z\-_\*\^]+$)(?![A-Z0-9\-_\*\^]+$)"""
```

用户手机号检查模块：

- 判断以`+`号开头，并且存在字符`.`
- 以`.`为分界分割字串，得到`areaCode`和`phoneNumber`
- 分别对`areaCode`和`phoneNumber`进行长度和纯数字检查

用户url检查模块：

- 判断存在字符`/`
- 以字符`/`分割协议与域名
- 判断协议段是否为`https://`或者`http://`
- 判断域名长度是否合理，并且包含`.`
- 根据`.`进行字串分割，得到多个标签序列
- 判断最后一个标签是否为纯数字
- 用正则依次对每个标签序列进行检查

幸运数字检查模块：

- 判断非负以及为`int`类型

2.2 补充单元测试

测例1（有效等价类）：

找了一种一般情况，覆盖掉主要路径

```
content = {
    'username': 'asdA12312',
    'password': '123nasi_Asd',
    'nickname': 'huangzhengchao',
    'url': 'https://hzcportfolio.dora.run',
    'mobile': '+86.019801280171',
    'magic_number': 7
}
self.assertEqual(register_params_check(content), ("ok", True))
```

测例2和3：

首先对username进行一个检测，根据边界值分析，造一个长度为5，然后都不符合字母在前、数字在后、包含大小写地规则；其余字段无需变更，因为也不会检查到。

```
content = {
    'username': '1a2vc',
    'password': '123nasi_Asd',
    'nickname': 'huangzhengchao',
    'url': 'https://hzcportfolio.dora.run',
    'mobile': '+86.019801280171',
}
self.assertEqual(register_params_check(content), ("username", False))
```

```
content = {
    'username': 'aaaaaaaa',
    'password': '123nasi_Asd',
    'nickname': 'huangzhengchao',
    'url': 'https://hzcportfolio.dora.run',
    'mobile': '+86.019801280171',
}
self.assertEqual(register_params_check(content), ("username", False))
```

测例4:

对password进行检测，首先保证username正确，这里也找了一个边界，然后password也找了一个边界，纯符号且只有8位

```
content = {
    'username': 'HzC88',
    'password': ',,,,,,,,',
    'nickname': 'huangzhengchao',
    'url': 'https://hzcportfolio.dora.run',
    'mobile': '+86.019801280171',
}
self.assertEqual(register_params_check(content), ("password", False))
```

测例5、6、7:

对mobile进行检测，首先也保证username和password都正确，都对长度取了一个上界，mobile主要对格式进行了检查，第一个对含有`+`号进行检测，第二个对区号为2位数字进行检测，第三个对手机号为12位数字进行检测

```
content = {
    'username': 'Riccardo8888',
    'password': '_^*ABSa1239__88',
    'nickname': 'huangzhengchao',
    'url': 'https://hzcportfolio.dora.run',
    'mobile': '86019801280171',
}
self.assertEqual(register_params_check(content), ("mobile", False))
```

```
content = {
    'username': 'Riccardo8888',
    'password': '_^*ABSa1239__88',
    'nickname': 'huangzhengchao',
    'url': 'https://hzcportfolio.dora.run',
    'mobile': '+1xs.019801280171',
}
self.assertEqual(register_params_check(content), ("mobile", False))
```

```
content = {
    'username': 'Riccardo8888',
    'password': '_^*ABSa1239__88',
    'nickname': 'huangzhengchao',
    'url': 'https://hzcportfolio.dora.run',
    'mobile': '+98.019801280xd1',
}
self.assertEqual(register_params_check(content), ("mobile", False))
```

测例8、9:

对url进行检测，保证username、password、mobile正确，且username和password取一种中间值，第一个url主要是对协议名检测；第二个主要是对域名字段检查，检查最后一个tag不能为纯数字；第三个主要对不能以`-`开头或结尾检测

```
content = {
    'username': 'yduAab17',
    'password': 'asiiQWE867_*',
    'nickname': 'huangzhengchao',
    'url': 'file://软工作业',
    'mobile': '+99.128301280171',
}
self.assertEqual(register_params_check(content), ("url", False))
```

```
content = {
    'username': 'asdbjhA432',
    'password': 'asuiASyui_^1',
    'nickname': 'huangzhengchao',
    'url': 'https://www.bai-du.123.999',
    'mobile': '+99.128301280171',
}
self.assertEqual(register_params_check(content), ("url", False))
```

```
content = {
    'username': 'asdbjhA432',
    'password': 'asuiASyui_^1',
    'nickname': 'huangzhengchao',
    'url': 'https://www.-google.123.com.asd31232d1241789m',
    'mobile': '+99.128301280171',
}
self.assertEqual(register_params_check(content), ("url", False))
```

测例10:

对magic_number不能为非负数进行测试

```
content = {
    'username': 'aA1111',
    'password': '1A1A1A1aA__^',
    'nickname': 'huangzhengchao',
    'url': 'https://1-23.12-3.123.12C',
    'mobile': '+99.128301280171',
    'magic_number': -1
}
self.assertEqual(register_params_check(content), ("magic_number", False))
```

测试结果

Module	*statements*	*missing*	*excluded*	*branches*	*partial*	*coverage*
app__init__.py	19	9	0	4	1	57%
app\checkers\user.py	68	11	0	48	10	82%
app\utils\config.py	12	8	0	4	0	38%
Total	**99**	**28**	**0**	**56**	**11**	**74%**

coverage.py v6.4.4, created at 2022-10-03 12:20 +0800

参数说明如下：

- `statements`：总的有效代码行数
- `missing`：未执行的代码行数
- `branches`：总分支数
- `coverage`：覆盖率

3 集成测试

3.1 登录测试

测试1：使用错误的信息进行登录，检查返回值为失败。

在初始化时只向数据库加入了`username="test",password="test"...`的用户数据，所以构造一个错误信息检测即可，预期会返回`not found`以及500状态码

```
data = {"username":"123", "password": "21321"}

response = current_app.test_client().patch(
    url_for('user.login'),
    json=data,
)
json_data = json.loads(response.data)
self.assertEqual(json_data['message'], "not found")
self.assertEqual(response.status_code, 500)
```

测试2：使用正确的信息进行登录，检查返回值为成功；进行登出，检查返回值为成功。

使用`test`去登录即可，预期返回的`username`为`test`，然后状态码为200。之后需要用到返回的`jwt`加入到`logout`请求的头部当中，这样才能正常登出

```
data = {"username":"test", "password": "test"}
response = current_app.test_client().patch(
    url_for('user.login'),
    json=data,
)
json_data = json.loads(response.data)
self.assertEqual(json_data['username'], "test")
self.assertEqual(response.status_code, 200)

jwt = json_data['jwt']
response = current_app.test_client().patch(
    url_for('user.warperlogout'),
    json=data,
    headers={'Authorization':jwt}
)
json_data = json.loads(response.data)
self.assertEqual(json_data['message'], "ok")
self.assertEqual(response.status_code, 200)
```

3.2 注册测试

测试1： 使用正确的信息进行注册，检查返回值为成功；使用正确注册信息进行登录，检查返回值为成功

这里使用了单元测试中第一个测例进行测试，预期返回 `ok` 和状态码200；

登录之后可以查看返回的用户名是否匹配，状态码是否为200；

```
data = {
    'username': 'asdA12312',
    'password': '123nasi_Asd',
    'nickname': 'huangzhengchao',
    'url': 'https://hzcportfolio.dora.run',
    'mobile': '+86.019801280171',
    'magic_number': 7
}
response = current_app.test_client().post(
    url_for('user.register_user'),
    json=data
)
json_data = json.loads(response.data)
self.assertEqual(json_data['message'], "ok")
self.assertEqual(response.status_code, 200)

response = current_app.test_client().patch(
    url_for('user.login'),
    json={
        'username':'asdA12312',
        'password':'123nasi_Asd'
    }
)
json_data = json.loads(response.data)
self.assertEqual(json_data['username'], "asdA12312")
self.assertEqual(response.status_code, 200)
```

3.3 登出测试

测例1：未登录直接登出

没有登录的话，是不知道`token`信息的，那么登出理论上会报401错误

```
response = current_app.test_client().patch(
    url_for('user.warperlogout'),
)
json_data = json.loads(response.data)
self.assertEqual(json_data['message'], "User must be authorized.")
self.assertEqual(response.status_code, 401)
```

Module	*statements*	*missing*	*excluded*	*branches*	*partial*	*coverage*
app__init__.py	19	9	0	4	1	57%
app\checkers\user.py	68	22	0	48	22	62%
app\controllers\login_required.py	8	5	0	2	0	50%
app\controllers\user.py	59	41	0	14	3	34%
app\services\user.py	34	21	0	4	0	39%
app\utils\config.py	12	8	0	4	0	38%
app\utils\jwt.py	27	11	0	2	1	59%
app\utils\middleware.py	11	3	0	4	1	73%
Total	**238**	**120**	**0**	**82**	**28**	**51%**

coverage.py v6.4.4, created at 2022-10-03 16:50 +0800

4 端到端测试

Xpath参考

端到端测试的实现思路：

首先在`release`环境下找到实现的最短路径，通过chrome调试工具找到每一个组件在DOM树下的位置，用`xpath`进行表示，当然如果该组件有`id`，那么用`id`表示会更方便

任务1：登录后发帖，发帖标题为：Hello World，发帖内容为：你好！

最短路径：点击发帖、输入标题、输入内容、点击发表

```
time.sleep(1)
self.client.find_element_by_xpath(
    "//*[@id='root']/div/div[3]//*[@class='MuiButtonGroup-root']/a").click()
time.sleep(1)
self.client.find_element_by_xpath(
    "//*[@id='root']/div/div[3]//*[@class='MuiInputBase-input MuiInput-
input']").send_keys('Hello World')
time.sleep(1)
self.client.find_element_by_xpath(
    "//*[@id='root']/div/div[3]/div/div/div[2]//textarea").send_keys('你好！')
time.sleep(1)
self.client.find_element_by_xpath(
    "//*[@id='root']/div/div[3]/div/div/div[3]/button").click()
```

任务2：更新帖子标题为：Hello World！，帖子内容为：你好。

最短路径：点击编辑、给标题输入一个`！`、给内容输入`BACK_SPACE`以及一个`。`、点击发表

```
time.sleep(1)
self.client.find_element_by_xpath(
    "//*[@id='root']/div/div[3]/div/div[2]/div[2]/span[2]/span/a[1]").click()
time.sleep(1)
self.client.find_element_by_xpath(
    "//*[@id='root']/div/div[3]//*[@class='MuiInputBase-input MuiInput-
input']").send_keys('!')
time.sleep(1)
self.client.find_element_by_xpath(
    "//*
[@id='root']/div/div[3]/div/div/div[2]//textarea").send_keys(Keys.BACK_SPACE+'。'
)
time.sleep(1)
self.client.find_element_by_xpath(
    "//*[@id='root']/div/div[3]/div/div/div[3]/button").click()
```

任务3：回复刚才的帖子，回复内容为：你好!

最短路径：点击回复、输入内容为`你好!`、点击发表

```
time.sleep(1)
self.client.find_element_by_xpath(
    "//*[@id='root']/div/div[3]/div/div[2]/div[2]/span[2]/span/a[2]").click()
time.sleep(1)
self.client.find_element_by_xpath(
    "//*[@id='root']/div/div[3]/div/div/div[2]//textarea").send_keys('你好! ')
time.sleep(1)
self.client.find_element_by_xpath(
    "//*[@id='root']/div/div[3]/div/div/div[3]/button").click()
```

任务4：退出登录

最短路径：点击右上角的用户名、点击登出

```
time.sleep(1)
self.client.find_element_by_xpath(
    "//*[@id='root']/div/header/div/span/a").click()
time.sleep(1)
self.client.find_element_by_xpath(
    "//*[@id='root']/div/div[3]/div/div[6]/button").click()
time.sleep(2)
```

注：为更好看清过程，每一步都加了延时

5 测试结果

Module	statements	missing	excluded	branches	partial	coverage
app__init__.py	19	9	0	4	1	57%
app\checkers\post.py	2	1	0	0	0	50%
app\checkers\reply.py	2	1	0	0	0	50%
app\checkers\user.py	68	11	0	48	10	82%
app\controllers\hello.py	22	18	0	0	0	18%
app\controllers\login_required.py	8	5	0	2	0	50%
app\controllers\post.py	110	70	0	36	14	37%
app\controllers\user.py	59	38	0	14	4	40%
app\services\post.py	112	62	0	24	5	46%
app\services\user.py	34	18	0	4	0	47%
app\utils\config.py	12	8	0	4	0	38%
app\utils\jwt.py	27	11	0	2	1	59%
app\utils\middleware.py	11	3	0	4	1	73%
Total	**486**	**255**	**0**	**142**	**36**	**50%**

coverage.py v6.4.4, created at 2022-10-04 10:49 +0800

6 Docker部署

服务器信息：

- 实例ID：lhins-npkkudfy
- 实例IP：101.43.165.141
- 用户名：ubuntu
- 密码：aSKLiM5M^kcK2v

清软论坛网址：

- 主页面：http://101.43.165.141:8000/
- 后端api接口：http://101.43.165.141:8000/api/v1

6.1 服务器端安装Docker

Docker安装

启动Docker

```
sudo systemctl enable docker
sudo systemctl start docker
```

建立docker用户组

```
sudo groupadd docker     #添加docker用户组
sudo gpasswd -a $USER docker     #将登陆用户加入到docker用户组中
newgrp docker     #更新用户组
docker ps    #测试docker命令是否可以使用sudo正常使用
```

6.2 编写Dockerfile文件

Dockerfile是一个用来**构建镜像**的文本文件，文本内容包含了一条条构建镜像所需的指令和说明。

```
FROM python:3.8.3

RUN mkdir /code
WORKDIR /code
RUN pip install pip -U -i https://pypi.tuna.tsinghua.edu.cn/simple
ADD requirements.txt /code/
RUN pip install -r requirements.txt -i https://pypi.tuna.tsinghua.edu.cn/simple
ADD . /code/
```

- 将主机上当前目录挂载到container`/code`目录下，此目录将包含当前主机上`.`中的代码；
- WORKDIR：切换到镜像中的指定路径，设置工作目录。WORKDIR 指令为 Dockerfile 中跟随它的任何 RUN、CMD、ENTRYPOINT、COPY、ADD 指令设置工作目录
- `ADD`：将主机上的文件添加到容器内指定目录下

  例如`ADD . /code/`将所有代码都添加到`/code`目录下

6.3 编写docker-compose.yml文件

compose文件是一个定义服务`services`、网络`networks`、卷`volumes`的yaml文件

```
version: "3"
services:
  app:
    container_name: app
    restart: always
    build: .
    command: >
      bash -c
      "gunicorn --timeout=30 --workers=4 --bind :8000 manage:app"
    volumes:
      - .:/code
      - static-volume:/code/app/static
    expose:
      - "8000"
    depends_on:
      - db
    networks:
      - web_network
      - db_network
  db:
    container_name: mysql
    image: mysql:5.7
    volumes:
      - "~/mysql:/var/lib/mysql"
    expose:
      - "3306"
    restart: always
    environment:
      - MYSQL_ROOT_PASSWORD=2019011463
      - MYSQL_DATABASE=thss
```

```
      - TZ=Asia/Shanghai
    command: ['mysqld', '--character-set-server=utf8mb4', '--collation-
server=utf8mb4_unicode_ci']
    networks:
      - db_network

  nginx:
    container_name: nginx
    restart: always
    image: nginx:latest
    ports:
      - "8000:8000"
    volumes:
      - static-volume:/code/app/static
      - ./nginx:/etc/nginx/conf.d
    depends_on:
      - app
    networks:
      - web_network

networks:
  web_network:
    driver: bridge
  db_network:
    driver: bridge

volumes:
  static-volume:
```

- 通过`container_name: xxx`可以设置运行时容器名;
- 通过volumes可以将一些资源持久化存储

6.4 编写nginx配置文件

```
server {
  listen 8000; # 监听端口
  server_name localhost; # 绑定ip

  # 静态文件服务
  location /static/ {
    autoindex on;
    alias /code/app/static/;
  }

  # 代理
  location / {
    proxy_pass http://app:8000;
  }

  location = /api/v1/ {
    proxy_pass http://app:8000/apidocs/#;
  }
}
```

- 其中为了让用户输入`ip:port/api/v1`可以访问后端api接口信息，为了不与`location /`冲突，需要调整location的优先级，这里使用了最高优先级的精确匹配。（nginx的location优先级）

6.5 实验体会

这是第一次接触Docker部署，很久之前就听说过Docker，但一直不知道是什么，有什么用，通过这次实验对Docker有了一个初步认识，也对镜像、容器、仓库的概念更加熟悉，也体会到Docker的便利、轻量化。

同时也学会了一些工程化操作，比如将所有依赖写入到一个`requirements.txt`中，也利于使用Docker自己构建镜像。

也更加理解了反向代理的概念以及优势，学会了如何去设置代理。

也了解了如何进行volume挂载，虽然还有很多不理解的地方，今后也会更加深入学习Docker。

6.6 Tricks

```
#删除mysql(需要root权限)
sudo rm -rf mysql

#列出正在运行的容器
docker ps

#列出所有容器
docker ps -a

#查看数据卷
docker volume ls

#查看docker磁盘使用情况
docker system df

#后台启动项目
docker-compose up -d

#关闭项目
docker-compose down
```