单元测试与应用部署

《软件工程》 清华大学软件学院 2022-2023秋

作业内容

只需要修改带TODO标注的部分

- 测试部分【15分】
  - 代码风格测试【2分】
  - 单元测试【6分】
  - 集成测试【4分】
  - 端到端测试【3分】
  - 作业文档
- Docker应用部署【5分】

作业提交

禁止修改无TODO标注的文件以及添加新文件

```
# 提交清小软_2020000000.zip文件
python zip.py 清小软 2020000000
```

- 修改无TODO标注文件

```
MODIFIED FILES NOT IN ALLOW LIST ARE NOT ALLOW!!!
These are modified files not allowed:
- app/configs.py
```

- 添加新文件

```
NEW FILES ARE NOT ALLOW!!!
These are new files you added:
- new.txt
```

环境搭建

推荐编辑器：vscode

```
# python3.8
virtualenv venv
source venv/bin/activate
# https://mirrors.tuna.tsinghua.edu.cn/help/pypi/
pip install -r requirements.txt
```

```
# 填充Fake数据
python manage.py init_db
python manage.py runserver
# 可在http://localhost:5000/apidocs/查看swagger文档
#
# Serving Flask app "app" (lazy loading)
# Environment: dev
# Debug mode: on
# Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
# Restarting with stat
# Debugger is active!
# Debugger PIN: 135-486-532
#
# 可以使用用户名：test，密码：test登录
```

环境搭建

下载并配置浏览器driver路径，推荐chrome浏览器

在浏览器中输出chrome://version/查看当前chrome版本，在 https://chromedriver.chromium.org/downloads找到兼容的driver版本下载，解压后置于drivers目录，并修改`tests/test_e2e.py`中第31，33行的路径为本机的实际路径

提交时请撤销第31行，33行的修改

```
# 运行所有测试
python manage.py test
# 查看命令文档
python manage.py test --help
# 筛选运行测试
python manage.py test -f (test_api|test_basic|test_e2e)
```

环境搭建

```
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
Name                         Stmts   Miss Branch BrPart  Cover
-------------------------------------------------------------
app/__init__.py                 18      9      4      1    55%
app/checkers/user.py             2      1      0      0    50%
app/controllers/hello.py        20     18      0      0    10%
app/controllers/user.py         52     44     14      2    15%
app/services/user.py            35     33      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                          177    144     32      4    20%
HTML version: file:///home/wangao/learn/SimpleBBS/test_report/index.html
```

代码风格

为什么要统一代码风格

- 提高代码可读性与代码质量
- 减少团队沟通成本
- 美学享受，心情愉快

常用工具

- flake8：https://flake8.pycqa.org/en/latest/
- autoflake：https://github.com/PyCQA/autoflake
- autopep8：https://pypi.org/project/autopep8/
- isort：https://github.com/PyCQA/isort

代码风格测试

- flake8 检查代码是否符合规范

```
flake8 .
```

- autoflake & autopep8 自动修复不符合规范的代码

```
autoflake .
autopep8 .
```

- isort 对引入依赖进行排序

```
isort .
```

上述工具均支持配置文件，如.flake8，.isort.cfg

https://flake8.pycqa.org/en/latest/user/configuration.html

https://flake8.pycqa.org/en/latest/user/options.html

代码风格测试

- 提供lint.sh自动格式化代码
  - 添加autopep8执行命令
  - 添加autoflake执行命令
  - 添加isort执行命令
  - 添加flake8执行命令
  - 补全flake8配置文件.flake8
- 评测时执行lint.sh检测是否正确格式化且符合flake8格式

```
chmod +x lint.sh
./lint.sh
echo $?
```

单元测试

参考文档https://docs.python.org/3/library/unittest.html

- 补全`register_params_check`函数，评测时通过所有测例即通过【**测试驱动开发**】
- 对`register_params_check`补充单元测试，**覆盖率**达到即通过

```
python manage.py test -f test_basic
```

```
test_register_params_check (test_basic.BasicTestCase) ... *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\utils\config.py        12      8      4      0    38%
-----------------------------------------------------------
TOTAL                      33     18      8      1    49%
HTML version: file://D:\projects\SimpleBBS\test_report/index.html
```

单元测试

```
import unittest
# 测试以类的形式组织，继承unittest.TestCase
# 测试以函数的形式运行，以test_命名
class TestStringMethods(unittest.TestCase):
  def test_example(self):
    self.assertEqual('foo'.upper(), 'FOO')
    self.assertTrue('FOO'.isupper())
    self.assertFalse('Foo'.isupper())
    with self.assertRaises(TypeError):
      raise TypeError

# 测试runner
if __name__ == '__main__':
  unittest.main()
```

Command line runner

```
python -m unittest test.py
coverage run test.py
coverage report
```

单元测试

https://coverage.readthedocs.io/en/latest/faq.html#q-why-do-the-bodies-of-functions-show-as-executed-but-the-def-lines-do-not

Coverage report: 49%

coverage.py v6.4.4, created at 2022-09-19 09:20 +0800

Module	statements	missing	excluded	branches	partial	coverage
app__init__.py	19	9	0	4	1	57%
app\checkers\user.py	2	1	0	0	0	50%
app\utils\config.py	12	8	0	4	0	38%
Total	**33**	**18**	**0**	**8**	**1**	**49%**

coverage.py v6.4.4, created at 2022-09-19 09:20 +0800

Coverage for **app\checkers\user.py**: 50%

2 statements | 1 run | 1 missing | 0 excluded | 0 partial

« prev ^ index » next coverage.py v6.4.4, created at 2022-09-19 09:07 +0800

```
1  # -*- coding: utf-8 -*-
2
3  def register_params_check(content):
4      """
5      TODO: 进行参数检查
6      """
7      return "ok", True
```

集成测试

也叫组装测试或联合测试。在单元测试的基础上，将所有模块按照设计要求（如根据结构图）组装成为子系统或系统，进行集成测试。

为什么要进行集成测试

- 一些模块虽然能够单独地工作，但并不能保证连接起来也能正常的工作
- 一些局部反映不出来的问题，在全局上很可能暴露出来

在后续课程中，老师也会着重介绍集成测试，同学们可以先通过这次作业练习一下如何进行集成测试，也可将集成测试用于后续的大作业中保障大作业应用的正确性。

集成测试

https://flask.palletsprojects.com/en/2.2.x/testing/

- 补全`tests/test_api.py`TODO部分

```
python manage.py test -f test_api
```

- 评测时按照顺序正确测试对应路由即通过

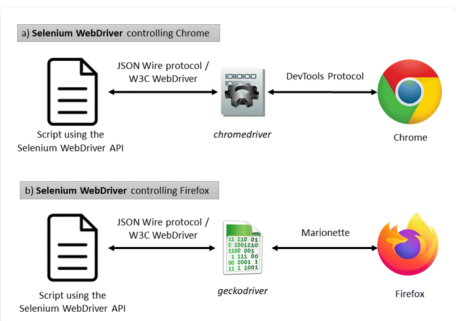
```
def test_register(self):
  """
  Example: 使用错误信息进行注册，检查返回值为失败
  """
  pass
  """
  TODO: 使用正确的信息进行注册，检查返回值为成功
  TODO: 使用正确注册信息进行登录，检查返回值为成功
  """
```

端到端测试

为什么进行端到端测试

- 通过添加比其他测试方法（如单元和功能测试）更详细的测试案例，扩大测试范围。
- 通过运行基于终端用户行为的测试用例，确保应用程序的正确执行。
- 通过自动化关键用户路径缩短应用测试周期。
- 通过减少测试软件的时间，降低构建和维护软件的总体成本。
- 有助于可预测地、可靠地检测错误。

Selenium是一个用于Web应用程序测试的工具，直接运行在浏览器中，模拟真正的用户操作，测试应用程序功能，验证用户的实际需求。



端到端测试

https://selenium-python.readthedocs.io/

- 补全`tests/test_e2e.py`TODO部分

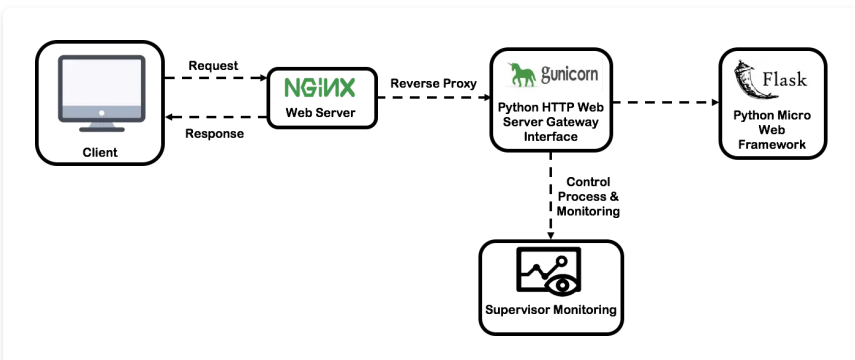
```
python manage.py test -f test_e2e
```

- Selenium

```
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.common.by import By
driver = webdriver.Chrome('drivers/chromedriver.exe')
driver.get("http://www.baidu.com")
assert "百度" in driver.title
# https://selenium-python.readthedocs.io/locating-elements.html
elem = driver.find_element(By.NAME, "wd")
elem.clear()
elem.send_keys("软件工程")
elem.send_keys(Keys.RETURN)
time.sleep(10)
driver.close()
```

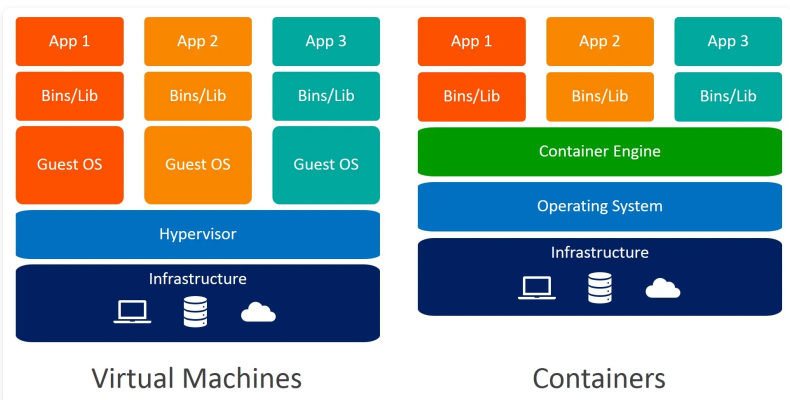
应用部署

- 生产服务器需使用Nginx做反向代理服务器
  - Nginx安装：http://nginx.org/en/docs/install.html
  - 相关博客：https://juejin.im/post/6844903944267759624
- WSGI服务器
  - Python程序在生产环境中要托管在WSGI服务器之下
  - gunicorn -w4 myapp:app



Docker

- 环境配置繁琐，部分依赖库之间耦合严重，交接沟通成本大
- 一旦服务器崩溃需要重新部署，或者需要迁移到新的集群，需要大量时间
- 针对不同的系统要设置不同的开机自启脚本
- 难以快速多机部署和横向扩容
- 存在更换系统或者升级系统版本后项目无法启动的可能
- 使用传统虚拟机资源消耗过大，启动慢



Docker

安装参考链接：https://yeasy.gitbooks.io/docker_practice/install/

国内拉取docker hub image速度较慢，可以考虑使用各类镜像加速

- 科大：https://docker.mirrors.ustc.edu.cn
- 网易云：https://hub-mirror.c.163.com
- 设置镜像加速：https://yeasy.gitbook.io/docker_practice/install/mirror

Docker

- Image 镜像

  特殊文件系统，提供运行时所需程序、库、资源、配置等文件 构建后不会改变

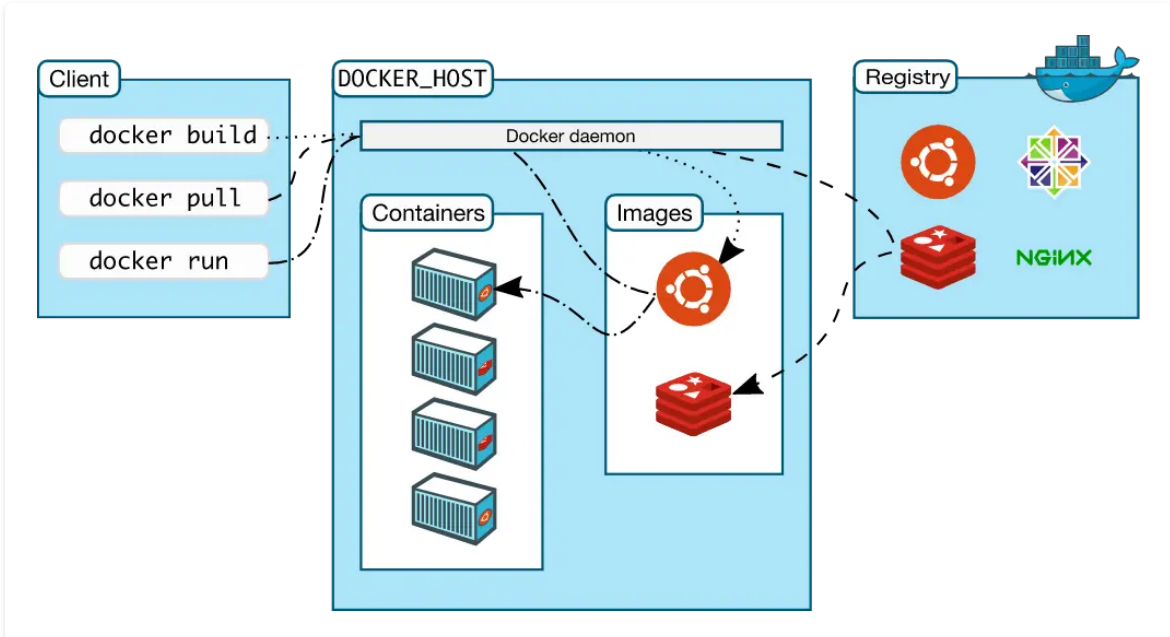
- Container 容器

  容器是镜像运行时的实体，可以被创建、启动、停止、删除 与镜像的关系可以类比为OOP中的实例与类

- Repository 仓库

  镜像的集中存储分发 一个镜像包括名称和标签，默认标签为latest。例如ubuntu:18.04

Docker


Client
docker build
docker pull
docker run
DOCKER_HOST
Docker daemon
Containers
Images
Registry
NGINX

Docker

https://docs.docker.com/engine/reference/commandline/docker/

```
# 查看所有正在运行的容器
docker ps
# 查看所有镜像
docker images
# 以bash方式进入容器内部交互
docker exec -it django-docker sh
# 查看容器日志
docker logs -f <container>
```

Dockerfile

https://yeasy.gitbook.io/docker_practice/image/build

```
FROM nginx
RUN echo '<h1>Hello, Docker!</h1>' > /usr/share/nginx/html/index.html
```

- 构建镜像

```
docker build -t nginx:v3 .
docker images | grep nginx
```

- 运行容器

```
docker run -d --name docker-test -p 2333:80 nginx:v3
docker ps
```

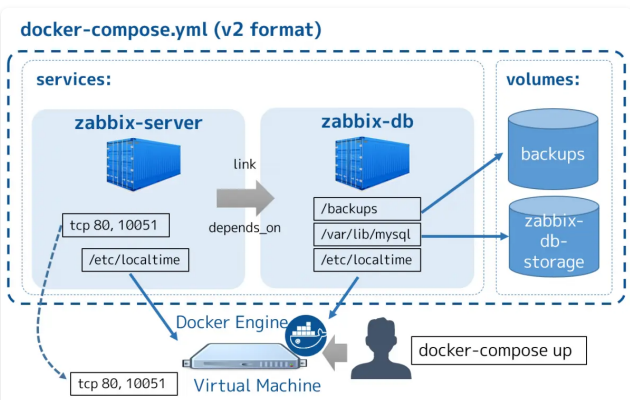
DockerCompose

https://docs.docker.com/compose/

实现对于多个Docker容器编排。其定位是“定义和运行多个Docker容器的应用”。允许用户通过docker-compose.yml模板文件来定义一组关联的容器构成项目。

有两个重要的概念：

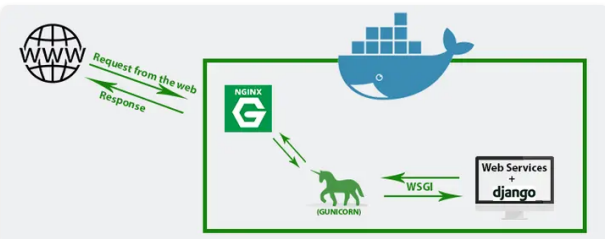
- Service 服务：一个应用的容器
- Project 项目：一组容器组成的业务单元



实例项目

https://github.com/PowerfooI/django-docker-tutorial

```
django-docker-tutorial
├── Dockerfile            构建app镜像
├── README.md
├── collected_static      静态文件
├── config
├── django_app            app代码
├── docker-compose.yml    定义应用组成
├── manage.py
└── requirements.txt
```



实例项目

- App容器

```
FROM python:3.7

ENV PYTHONUNBUFFERED 1

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/
```

实例项目

- App服务

```
app:
  restart: always
  build: .
  command: >
    bash -c
    "python3 manage.py collectstatic --no-input &&
    python3 manage.py migrate &&
    gunicorn --timeout=30 --workers=4 --bind :8000 django_app.wsgi:application"
  volumes:
    - .:/code
    - static-volume:/code/static
  expose:
    - "8000"
  depends_on:
    - db
  networks:
    - web_network
    - db_network
```

实例项目

- MySQL服务

```
db:
  image: mysql:5.7
  volumes:
    - "./mysql:/var/lib/mysql"
  env_file: .env
  expose:
    - "3306"
  restart: always
  environment:
    - MYSQL_DATABASE=${DB_NAME}
    - MYSQL_ROOT_PASSWORD=${DB_PASSWORD}
  networks:
    - db_network
```

实例项目

- Nginx服务

```
nginx:
  restart: always
  image: nginx:latest
  ports:
    - "8001:8000"
  volumes:
    - static-volume:/code/static
    - ./config/nginx:/etc/nginx/conf.d
  depends_on:
    - app
  networks:
    - web_network
```

- Nginx配置

```
server {
  location /static/ {
    autoindex on;
    alias /code/collected_static/;
  }
}
```

实例项目

- 将目录中的 .env_example 文件复制一份，重命名为 .env，补全 .env 文件中数据库名和数据库密码，用于 MySQL 镜像的配置使用
- 将第2步中补全的数据库名和密码填写到 django_app/settings.py 的 DATABASES 设置当中，可以全局搜索 [DB_NAME] 和 [DB_PASSWORD] 可快速定位
- 执行命令 docker-compose up 启动项目
  - 首次启动会先下载所需的基础镜像，请耐心等待
  - 可能会出现连接不上数据库的问题，是因为 MySQL 镜像初次使用需要一定的初始化过程
  - Django 会重试连接数据库，等 MySQL 初始化完成后可完成连接
- 在 8001 端口访问启动的应用

实例项目

- 查看运行中的容器：

```
docker ps
```

- 进入容器进行交互

```
docker exec -it django-docker-tutorial-app-1 sh
```

- 停止项目：

```
docker-compose down
```

Thanks!

Q&A