单元测试和 Docker 部署

李祁

October 25, 2020

1 单元测试

1.1 测试驱动开发的体会

测试驱动开发要求我们先编写测试代码, 然后编写功能代码以通过测试. 通过测试来推动整个开发的进行. 因为在编写测试代码的的时候会重点考虑一些边界条件和非法输入, 因此依靠其编写出来的功能代码更加高质量, 更加鲁棒. 此外, 有测试代码驱动写出的功能代码也更加简洁易用.

1.2 设计思路

因为实现逻辑中基本没有复杂的逻辑判断和循环, 所以测试的重点主要在输入合法性上. 首先编写一个符合规范的测试用例, 确保功能函数返回正确. 然后在此基础上修改各个参数, 检查其是否抛出异常. 对于每一个参数都会检查其

- 是否存在
- 类型是否正确
- 长度是否符合规定
- 若是字符串, 是否有不允许的字符出现
- 语义要求 (如邮箱, 身份证格式等)

1.3 实现思路

在实现测试代码时将每一个测试用例都用一个函数来实现, 使得语义和目的更加清晰. 在实现功能代码时则将每一个检测都和测试用例对应起来, 以确保功能代码能通过测试.

1.4 测试用例列表

1.4.1 PostCheckTest

字典是否存在:

1. test_no_content_case

基础样例:

1. test_base_case

标题长度及类型:

1. test_title_edge_length_case
2. test_no_title_or_not_str_case
3. test_too_short_or_too_long_title_case

内容长度及类型:

1. test_content_edge_length_case
2. test_no_content_or_not_str_case
3. test_too_short_or_too_long_content_case

1.4.2 ReplyCheckTest

字典是否存在:

1. test_no_content_case

基础测例:

1. test_base_case

内容长度及类型:

1. test_no_content_or_not_str
2. test_content_edge_length_case
3. test_too_long_or_too_short_content_case

无 Id, Id 不是 int 或者为负数:

1. test_no_replyId_case
2. test_replyId_not_int_or_neg_case

1.4.3 UserCheckTest

字典是否存在:

1. test_no_content_case

基础测例:

1. test_base_case

用户名长度及类型:

1. test_username_edge_case
2. test_username_too_long_or_too_short
3. test_no_username_or_wrong_type_case

密码长度及类型:

1. test_password_edge_length_case
2. test_no_password_or_wrong_type_case
3. test_password_too_long_or_too_short_case

密码字符合规性测试:

1. test_password_no_num_case
2. test_password_no_upper_case
3. test_password_no_lower_case
4. test_password_invalid_character

昵称长度及类型:

1. test_no_nickname_or_wrong_type
2. test_nickname_edge_length_case
3. test_nickname_too_long_or_too_short_case

身份证长度及类型:

1. test_no_document_number_or_wrong_type
2. test_document_number_wrong_length

身份证地址合法性检测:

1. test_document_number_address_not_number

身份证生日合法性检测:

1. test_document_number_birthday_not_number
2. test_document_number_02_29
3. test_document_number_invalid_date
4. test_document_number_less_than_18

身份证顺序码检测:

1. test_document_number_wrong_check_code

身份证校验码:

1. test_document_number_order_not_number

电话长度及类型:

1. test_no_mobile_or_wrong_type
2. test_wrong_mobile_length_case

邮箱长度及类型:

1. test_no_email_or_wrong_type
2. test_email_wrong_at_number

邮箱域内部分检查:

1. test_inhost_invalid_character
2. test_inhost_too_long

邮箱域名检查:

1. test_host_too_long
2. test_host_no_dot
3. test_host_dot_with_no_content
4. test_host_invalid_character
5. test_host_pure_number_in_last
6. test_host_hypen

1.5 测试结果及分析

测试通过率见图 1, 测试覆盖率见图 2. 因为功能代码与测试代码基本对应, 所以测试覆盖率能达到 100%, 也很容易定位出错的位置, 修改后可以全部通过.

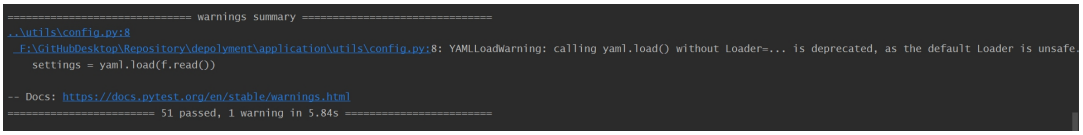

Figure 1: 通过率

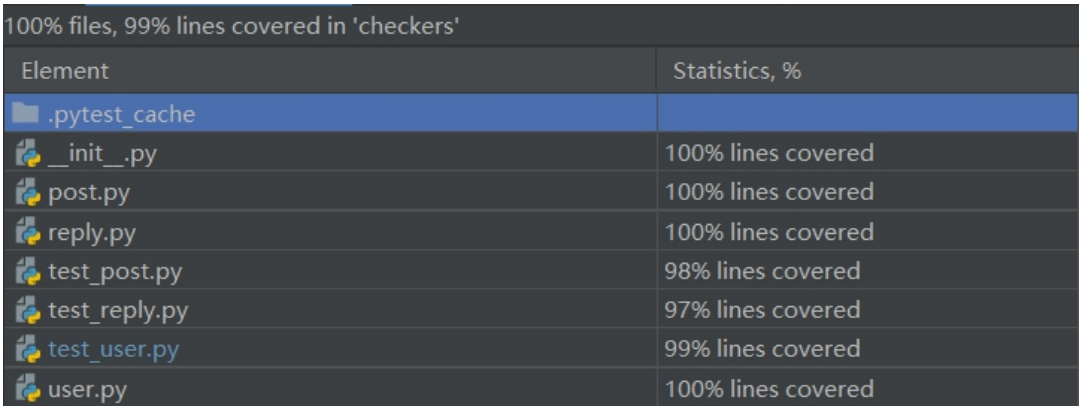

Figure 2: 覆盖率

2 Docker 部署

2.1 实现思路

1. 使用 DockerFile 创建 app 镜像, 所需要的环境依赖全部写入到 requirements.txt 中, 在编写 DockerFile 的时候只需要运行 `RUN pip install -r requirments.txt`即可.
2. 使用 docker-compose.yml 来编排容器, 其中 app 运行 Flask 服务, 基础 image 为 Python3.8, db 运行数据库服务, image 为 mysql:5.7, nginx 运行网络服务器, image 为 nginx:latest.
3. nginx 向外暴露 8000 端口, 对内监听 80 端口. app 在内网暴露 5000 端口, db 在内网暴露 3306 端口.
4. 在构建时, 会先启动 db 容器, 再启动 app 容器, 最后再启动 nginx 容器.
5. nginx 和 app 在一个同一个 bridge 网络中, app 和 db 在同一个 bridge 网络中, 通过 nginx 所在的容器无法直接访问数据库容器.
6. 将 mysql 容器内的`/var/lib/mysql`挂在到了宿主机的`/home/ubuntu/mysql`目录. 将数据库结构定义写在了`db/model.sql`中并将`db`挂载到了`/docker-entrypoint-initdb.d`中进行数据库的初始化.

2.2 体会

Docker 部署方便快捷, 易于迁移. 在部署成功后通过简单的命令就可以运行起一个复杂的项目, 是项目部署的利器.

2.3 运行方式

运行`docker-compose -d --build`即可