实验报告

> 2020010971 王麒杰

1 单元测试

1.1 检测函数

在`app/checkers/user.py`中，我使用了小学期学过的`python`正则表达式进行了输入参数检测。如：

```
# username
    if 'username' not in content.keys():
        return ('username', False)
    length = len(content['username'])
    if length < 5 or length > 12:
        return 'username', False
    if not re.match('^[a-zA-Z]+[0-9]+$', content['username']):
        return 'username', False
```

1.2 测试用例

- 设计思路：

在`tests/test_basic.py`中，共设计了31个样例，其中29个为错误样例，2个为正确样例。考虑到了字段缺失、字段过长或过短、字段内字符种类或数量错误、字段内字符

顺序错误、`mobile`区号或手机号错误、`url`协议错误/顶级域名错误/域名中`-`符号置于首尾等。每一个样例前都有注释标注当前样例的错误种类，如：

```
# url err: protocol
    def test_register_params_check_21(self):
        content = {}
        content['username'] = 'wangqj20'
        content['password'] = '233ha-HAHA'
        content['nickname'] = 'wangluli'
        content['url'] = 'hop://wangqj20.mails.tsinghua.edu.cn'
        content['mobile'] = '+86.189700485332'
        self.assertEqual(register_params_check(content), ("ok", True))
```

- 实现思路：

每一个样例都使用对`content`内字段赋值的形式构造。

- 测试用例列表：

测试用例名称	测试用例注释（错误原因）
`test_register_params_check_00`	nonetype
`test_register_params_check_01`	empty
`test_register_params_check_02`	lack username
`test_register_params_check_03`	lack password
`test_register_params_check_04`	lack nickname
`test_register_params_check_05`	lack url
`test_register_params_check_06`	lack mobile
`test_register_params_check_07`	username err: short
`test_register_params_check_08`	username err: long
`test_register_params_check_09`	username err: no letter
`test_register_params_check_10`	username err: no number
`test_register_params_check_11`	username err: sequence
`test_register_params_check_12`	password err: short
`test_register_params_check_13`	password err: long
`test_register_params_check_14`	password err: lack type
`test_register_params_check_15`	password err: illegal symbol
`test_register_params_check_16`	mobile err: erea short
`test_register_params_check_17`	mobile err: erea long

`test_register_params_check_18`	mobile err: number short
`test_register_params_check_19`	mobile err: number long
`test_register_params_check_20`	mobile err: lack point
`test_register_params_check_21`	url err: protocol
`test_register_params_check_22`	url err: domain symbol
`test_register_params_check_23`	url err: number top domain
`test_register_params_check_24`	url err: http long
`test_register_params_check_25`	url err: https long
`test_register_params_check_26`	url err: other symbol
`test_register_params_check_27`	url err: no point
`test_register_params_check_28`	magic number err: negative
`test_register_params_check_ok`	ok
`test_register_params_check_ok2`	ok2

- 覆盖率

设计的31种测试用例基本覆盖了可能出现的情况，但仍有少数情况未能覆盖到，如未能覆盖`password`的每一种顺序错误，而是挑选了其中一种作为错误样例。`app\checkers\user.py`的覆盖率达到了99%。

```
----------------------------------------------------------------------
---
Ran 31 tests in 0.035s

FAILED (failures=29)
Coverage:
Name                     Stmts   Miss Branch BrPart  Cover
-----------------------------------------------------------
app\__init__.py             19      9      4      1    57%
app\checkers\user.py        39      1     36      0    99%
app\utils\config.py         12      8      4      0    38%
-----------------------------------------------------------
TOTAL                       70     18     44      1    82%
HTML version: file://D:\VscodeFile\Python\Software\test_report/inde
x.html
```

2 集成测试

2.1 实现思路

模仿`test_api.py`中的示例代码，通过构造注册、登录、登出用户数据，并调用`current_app.test_client().post()`及`current_app.test_client().patch()`传参至接口，并获取`response`，检查返回值`response.data`，与期望输出进行比对。其中登出操作出现过错误，但发现可以通过登录时返回的`jwt`字段获取`token`，登出时添加包含t`oken`的请求头即可成功通过`wraperlogout`。其中正确登入登出的代码如下：

```
def test_login_correct(self):
        """
        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
        """
        data = {'username': 'test', 'password': 'test'}
        response = current_app.test_client().patch(
            url_for('user.login'),
            json=data
        )
        json_data = json.loads(response.data)

        # 可以读取到返回的'jwt'和'userId'
        # self.assertEqual(json_data['jwt'], None)
        # self.assertEqual(json_data['userId'], None)

        token = json_data['jwt']
        self.assertEqual(json_data['username'], "test")
        self.assertEqual(json_data['nickname'], "test")
        self.assertEqual(response.status_code, 200)

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

3 端到端测试

3.1 实现思路

模仿`EXAMPLE`，使用`self.client.get()`获取当前网页，使用`self.client.find_element_by_xpath()`函数通过在浏览器使用f12开发者工具获取的xpath值找到待操作网页组件对应的`element`，或使用`self.client.find_element_by_class_name()`函数通过`class`属性找到对应`element`，接着通过`.click()`、`.send_keys()`等操作对`element`进行操作。在通过`xpath`寻找`element`时，出现因页面跳转时加载速度慢导致无法找到的情况，于是在每个`test`中加入了`time.sleep(0.2)`等待页面加载完成。对`MuiInputBase-input`元素操作时，发现`clear()`无法清除其中文字，但发现可用`.send_keys(Keys.CONTROL, 'a', Keys.DELETE)`实现清除。

其中一个`test`代码如下：

```
def test_web_forum_update(self):
        """
        test3: 更新帖子标题为：Hello World！，帖子内容为：你好。
        """
        from selenium.webdriver.common.keys import Keys
        time.sleep(0.2)

        self.client.find_element_by_xpath(
            '//*[@id="post-main"]/div[2]/span[2]/span/a[1]').click()
        self.client.find_element_by_class_name(
            'MuiInputBase-input').send_keys(Keys.CONTROL, 'a', Keys.DELETE)
        self.client.find_element_by_class_name(
            'MuiInputBase-input').send_keys('Hello World!')
        self.client.find_element_by_name('textarea').send_keys(
            Keys.CONTROL, 'a', Keys.DELETE)
        self.client.find_element_by_name('textarea').send_keys('你好')
        self.client.find_element_by_xpath(
            '//*[@id="root"]/div/div[3]/div/div/div[3]/button').click()
        # time.sleep(3)
```

4 Docker部署

4.1 实现思路

*Docker*部署部分的实现思路大致为按照文档中给出的`django`实例项目，编写`Dockerfile`与`docker-compose.yml`文件，修改`nginx`的`.conf`文件，之后执行`docker-compose up`。其中`mysql`部分出现较多问题，通过在`docker-compose.yml`文件中`app`的`command`部分添加`python manage.py init_db`指令，在`requirements.txt`添加`mysql-connector-python`依赖并`pip`，以及修改`config.yaml`中的`MYSQL`解决。

4.2 体会

感觉部署部分相比于前面几个部分比较困难，可能主要原因是此前没使用过服务器，导致并不是很会在服务器端`debug`。不过得益于这次作业，让我既熟悉了服务器的一些操作，又了解了使用`docker`将应用部署至服务器的基本方法及所需的几种文件的语法。