

实验五：软件测试与修复 实验报告

姓名：张路远

学号：221220144

邮箱：3501334609@qq.com

实验项目：天气软件

编程语言：Python

编辑器：Vscode

测试工具：Unittest

覆盖率检查工具：Coverage

单元测试报告

天气服务模块 `weather_service.py`

天气服务模块通过调用和风天气提供的api获取指定城市的天气，单元测试主要检查其能否成功调用api并解析和传递数据，同时能否应对可能的异常情况，包含如下12条测试：

```
#### test_get_current_weather_valid
```

该测试使用模拟（mock）技术，模拟一个固定的返回结果，检查`getCurrentWeather`方法能否正确解析从API返回的数据。测试验证了当API返回有效的天气数据时，`getCurrentWeather`是否能够正确地提取和返回天气信息。

```
#### test_get_current_weather_invalid_city
```

此测试使用一个无效的查询城市（例如：无锡），目的是检查`getCurrentWeather`方法在遇到非法城市名时，是否能够捕获错误并正确地进行处理。例如，可能返回默认值或抛出适当的错误消息。

```
#### test_get_current_weather_api_error
```

该测试通过模拟一个错误码500的API返回，检查`getCurrentWeather`方法

是否能正确地处理API错误，并返回预设的默认值。这帮助确保当API返回服务器错误时，程序能够稳定运行，并不会中断。

test_get_current_weather_network_error

此测试模拟网络错误（例如无法连接到API），目的是验证`getCurrentWeather`方法在网络故障情况下是否能捕获异常并返回默认的天气数据。这确保了当遇到网络问题时，程序的健壮性不会受到影响。

test_get_current_weather_value_error

该测试模拟`getCurrentWeather`方法在解析API返回的JSON数据时遇到值错误（如数据格式问题）。通过这种方式，可以确保即使在数据解析失败的情况下，程序仍能返回默认的天气数据。

test_get_current_weather_no_rain

该测试验证当API返回的数据表明没有降雨时，`getCurrentWeather`方法是否能够正确地将`is_rainy`标记为`False`。通过提供一个天气描述为“晴”的响应，测试确保方法正确地识别并处理天气状态。

test_get_current_weather_is_windy

这个测试用于检查`getCurrentWeather`方法是否能正确识别并返回风力信息。当API返回的数据表明风力等级大于等于3时（例如4级风），测试验证`is_windy`属性是否设置为`True`，确保程序能够准确地识别强风天气。

test_get_hourly_forecast_valid

该测试检查`getHourlyForecast`方法在API返回有效数据时的行为。它模拟了一个返回未来两小时温度的有效API响应，并验证`getHourlyForecast`方法是否能正确地返回这两小时的温度数据。

test_get_hourly_forecast_api_error

此测试通过模拟API返回错误码500来验证`getHourlyForecast`方法的错误处理机制。当API返回错误时，测试确保方法能正确处理，并返回预设的默认温度数据。

test_get_hourly_forecast_network_error

该测试模拟网络错误（例如无法连接到API），检查`getHourlyForecast`方法是否能在网络不可用时返回默认的温度数据。这确保了网络问题时，程序能够稳定运行，不会因连接失败而崩溃。

```
### test_get_hourly_forecast_value_error
```

此测试模拟`getHourlyForecast`方法在解析API返回的JSON数据时发生值错误，确保程序在遇到数据格式问题时能够正确地处理异常，并返回默认的温度数据。

```
### test_get_hourly_forecast_not_enough_data
```

该测试通过模拟一个返回数据量不足的API响应，验证`getHourlyForecast`方法是否能在没有足够的逐小时温度数据时，返回默认的温度数据。它确保即使API返回的数据不足，程序仍能正常运行并提供合理的默认值。

在项目文件夹中新建 `./tests` ，然后创建 `test_weather_service.py` ，编写测试代码如下

```
import unittest
from unittest.mock import patch
from weather import Weather
from weather_service import WeatherService

class TestWeatherService(unittest.TestCase):
    def setUp(self):
        self.weather_service = WeatherService()

    @patch('requests.get')
    def test_get_current_weather_valid(self, mock_get):
        mock_response = {
            "code": "200",
            "now": {
                "temp": "25",
                "humidity": "60",
                "windScale": "3",
                "text": "晴",
                "windDir": "东风"
            }
        }
        mock_get.return_value.json.return_value = mock_response
        result = self.weather_service.getCurrentWeather("南京")
        self.assertIsInstance(result, Weather)
        self.assertEqual(result.temperature, 25)
        self.assertEqual(result.humidity, 60)
```

```

        self.assertTrue(result.is_windy)
        self.assertFalse(result.is_rainy)

@patch('requests.get')
def test_get_current_weather_invalid_city(self, mock_get):
    with self.assertRaises(KeyError):
        self.weather_service.getCurrentWeather("无锡")

@patch('requests.get')
def test_get_current_weather_api_error(self, mock_get):
    mock_response = {"code": "500"}
    mock_get.return_value.json.return_value = mock_response
    result = self.weather_service.getCurrentWeather("南京")
    self.assertEqual(result.temperature, 20)

@patch('requests.get')
def test_get_current_weather_network_error(self, mock_get):
    mock_get.side_effect = Exception("Network error")
    result = self.weather_service.getCurrentWeather("南京")
    self.assertEqual(result.temperature, 20)

@patch('requests.get')
def test_get_current_weather_value_error(self, mock_get):
    mock_get.return_value.json.side_effect = ValueError(" ")
    result = self.weather_service.getCurrentWeather("南京")
    self.assertEqual(result.temperature, 20)

@patch('requests.get')
def test_get_current_weather_no_rain(self, mock_get):
    mock_response = {
        "code": "200",
        "now": {
            "temp": "20",
            "humidity": "50",
            "windScale": "3",
            "text": "晴",
            "windDir": "北风"
        }
    }

```

```

    }
    mock_get.return_value.json.return_value = mock_respon
    result = self.weather_service.getCurrentWeather("广州")
    self.assertFalse(result.is_rainy)

@patch('requests.get')
def test_get_current_weather_is_windy(self, mock_get):
    mock_response = {
        "code": "200",
        "now": {
            "temp": "20",
            "humidity": "50",
            "windScale": "4",
            "text": "风大",
            "windDir": "北风"
        }
    }
    mock_get.return_value.json.return_value = mock_respon
    result = self.weather_service.getCurrentWeather("上海")
    self.assertTrue(result.is_windy)

def test_get_hourly_forecast_valid(self):
    with patch('requests.get') as mock_get:
        mock_response = {
            "code": "200",
            "hourly": [
                {"temp": "22"},
                {"temp": "23"}
            ]
        }
        mock_get.return_value.json.return_value = mock_re
        result = self.weather_service.getHourlyForecast("
        self.assertEqual(result, [22, 23])

@patch('requests.get')
def test_get_hourly_forecast_api_error(self, mock_get):
    mock_response = {"code": "500"}
    mock_get.return_value.json.return_value = mock_respon

```

```

        result = self.weather_service.getHourlyForecast("南京")
        self.assertEqual(result, [20, 21])

    @patch('requests.get')
    def test_get_hourly_forecast_network_error(self, mock_get):
        mock_get.side_effect = Exception("Network error")
        result = self.weather_service.getHourlyForecast("南京")
        self.assertEqual(result, [20, 21])

    @patch('requests.get')
    def test_get_hourly_forecast_value_error(self, mock_get):
        mock_get.return_value.json.side_effect = ValueError(" ")
        result = self.weather_service.getHourlyForecast("南京")
        self.assertEqual(result, [20, 21])

    @patch('requests.get')
    def test_get_hourly_forecast_not_enough_data(self, mock_get):
        mock_response = {
            "code": "200",
            "hourly": [{"temp": "22"}]
        }
        mock_get.return_value.json.return_value = mock_response
        result = self.weather_service.getHourlyForecast("广州")
        self.assertEqual(result, [20, 21])

if __name__ == "__main__":
    unittest.main()

```

测试结果如下：

```
测试结果 终端 端口 GITLENS SQL CONSOLE 调试控制台 问题 输出
>>>PYTHON-EXEC-OUTPUT
Received test ids from temp file.
test_get_current_weather_api_error (test_weather_service.TestWeatherService.test_get_current_weather_api_error) ... 数据解析错误: API 返回错误码: 500
ok
test_get_current_weather_invalid_city (test_weather_service.TestWeatherService.test_get_current_weather_invalid_city) ... ok
test_get_current_weather_is_windy (test_weather_service.TestWeatherService.test_get_current_weather_is_windy) ... ok
test_get_current_weather_network_error (test_weather_service.TestWeatherService.test_get_current_weather_network_error) ... 未知错误: Network error
ok
test_get_current_weather_no_rain (test_weather_service.TestWeatherService.test_get_current_weather_no_rain) ... ok
test_get_current_weather_valid (test_weather_service.TestWeatherService.test_get_current_weather_valid) ... ok
test_get_current_weather_value_error (test_weather_service.TestWeatherService.test_get_current_weather_value_error) ... ok
test_get_hourly_forecast_api_error (test_weather_service.TestWeatherService.test_get_hourly_forecast_api_error) ... 数据解析错误: JSON parse error
ok
test_get_hourly_forecast_network_error (test_weather_service.TestWeatherService.test_get_hourly_forecast_network_error) ... 数据解析错误: API 返回错误码: 500
未知错误: Network error
ok
test_get_hourly_forecast_not_enough_data (test_weather_service.TestWeatherService.test_get_hourly_forecast_not_enough_data) ... ok
test_get_hourly_forecast_valid (test_weather_service.TestWeatherService.test_get_hourly_forecast_valid) ... ok
test_get_hourly_forecast_value_error (test_weather_service.TestWeatherService.test_get_hourly_forecast_value_error) ... 数据解析错误: JSON parse error
<<<PYTHON-EXEC-OUTPUT
ok
-----
Ran 12 tests in 0.023s

OK
Finished running tests!
```

Running Tests for Workspace(s): c:\Users\ZY\O...
test_get_current_weather_api_error
test_get_current_weather_invalid_city
test_get_current_weather_is_windy
test_get_current_weather_network_error
test_get_current_weather_no_rain
test_get_current_weather_valid
test_get_current_weather_value_error
test_get_hourly_forecast_api_error
test_get_hourly_forecast_network_error
test_get_hourly_forecast_not_enough_data
test_get_hourly_forecast_valid
test_get_hourly_forecast_value_error
> 38个较旧的结果

使用 `coverage` 工具可以方便地检查测试的语句覆盖率

```
> coverage run -m unittest tests.test_weather_service
> coverage html
```

File ▲	statements	missing	excluded	coverage
tests\test_weather_service.py	79	1	0	99%
weather_service.py	51	2	0	96%
weather.py	12	1	0	92%
Total	142	4	0	97%

coverage.py v7.6.9, created at 2024-12-16 15:21 +0800

`coverage` 报告天气服务模块的语句覆盖率达到96%，相关模块总体语句覆盖率为97%

用户设置模块 `user_settings.py`

用户设置模块处理用户对查询城市、用户名等相关设置的修改和保存，并支持程序加载时读取保存的设置，单元测试主要检查其能否正确保存和读取设置和应对用户行为，包含如下10条测试：

test_load_config_valid

该测试验证`loadConfig()`方法在配置文件存在且内容有效的情况下是否能够正常加载配置。通过模拟`open()`和`json.load()`，测试确保`loadConfig()`能够正确地从配置文件读取配置并返回正确的内容。此测试确保当配置文件格式正确时，应用能够加载到期望的配置。

test_load_config_file_not_found

该测试模拟配置文件不存在的情况（`FileNotFoundError`），检查`loadConfig()`方法是否能够处理文件缺失的错误，并返回默认配置。此测试确保当配置文件丢失时，应用能够恢复到初始状态，而不会导致程序崩溃。

test_load_config_invalid_json

该测试模拟配置文件损坏的情形（如文件格式不合法），使用`json.JSONDecodeError`来模拟JSON解析失败的情况。测试验证了`loadConfig()`方法在遇到无效的JSON时是否能够返回默认配置，从而保证程序的稳定性。

test_save_config

该测试验证`saveConfig()`方法是否能够正确地将配置信息保存到文件中。测试创建一个临时文件作为配置文件，并检查是否成功保存并加载了指定的配置内容。通过删除临时文件来清理资源，确保每次测试环境的一致性。

test_get_default_config

该测试验证`getDefaultConfig()`方法是否能够正确地返回默认配置。通过与预期的默认配置进行比较，确保当没有用户配置或需要恢复到初始状态时，能够返回标准的默认设置。

test_open_settings_dialog

该测试模拟打开`SettingsDialog`并修改用户名的操作，确保通过界面输入的用户名能够正确地保存。测试了`SettingsDialog`的交互功能，检查是否能够成功从用户界面获取并更新配置。

test_save_settings

该测试模拟在`SettingsDialog`中点击保存按钮后的操作，验证了用户设置的

保存功能。通过`mock_save_config`检查是否调用了`saveConfig()`，并确保保存后的用户名更新正确。

test_reset_to_default

该测试验证恢复默认设置的功能。模拟了在`SettingsDialog`中修改用户名后，点击“恢复默认设置”按钮，确保`resetToDefaultSettings()`方法能够恢复到默认配置，并且`saveConfig()`被正确调用以保存默认配置。

test_switch_content

该测试验证`SettingsDialog`中切换不同设置页面的功能。通过调用`switchContent()`方法切换到不同的页面，并检查`content_stack.currentIndex()`是否正确更新，确保设置界面能够顺利切换。

test_cancel_dialog

该测试验证`SettingsDialog`中的取消操作。模拟点击取消按钮（`reject()`），并确保设置窗口关闭，`isVisible()`返回`False`，保证用户取消操作时，窗口正确关闭，且不会保存更改。

在`./tests`中创建`test_user_settings.py`，编写测试代码如下

```
import json
import unittest
import tempfile
from unittest.mock import mock_open, patch, MagicMock
from user_settings import UserSettings, SettingsDialog
from PyQt5.QtCore import Qt
from PyQt5.QtWidgets import QApplication

class TestUserSettings(unittest.TestCase):

    @classmethod
    def setUpClass(cls):
        cls.app = QApplication([])

    @classmethod
    def tearDownClass(cls):
        cls.app.quit()
```

```

def setUp(self):
    self.user_settings = UserSettings()

    # 测试 loadConfig() 方法: 正常加载配置
    @patch('builtins.open', return_value=MagicMock(read=MagicMock(return_value='{"username": "test_user"}'))))
    @patch('json.load', return_value={"username": "test_user"})
    def test_load_config_valid(self, mock_json_load, mock_open):
        config = self.user_settings.loadConfig()
        self.assertEqual(config, {"username": "test_user"})

    # 测试 loadConfig() 方法: 配置文件不存在
    @patch('builtins.open', side_effect=FileNotFoundError)
    def test_load_config_file_not_found(self, mock_open):
        config = self.user_settings.loadConfig()
        self.assertEqual(config, self.user_settings.getDefaultConfig())

    # 测试 loadConfig() 方法: 配置文件损坏
    @patch('builtins.open', return_value=MagicMock(read=MagicMock(return_value='{username: "test_user",}'))))
    @patch('json.load', side_effect=json.JSONDecodeError("Expecting value", "", 0))
    def test_load_config_invalid_json(self, mock_json_load, mock_open):
        config = self.user_settings.loadConfig()
        self.assertEqual(config, self.user_settings.getDefaultConfig())

    # 测试 saveConfig() 方法: 保存配置
    def test_saveConfig(self):
        with tempfile.NamedTemporaryFile(delete=False) as temp_file:
            temp_file_name = temp_file.name
            user_settings = UserSettings()
            user_settings.settings_file = temp_file_name

```

```

        config = {
            "username": "testuser",
            "selectedCity": "北京",
            "alerts": [True, False],
            "charac": "八云紫"
        }
        user_settings.saveConfig(config)
        saved_config = user_settings.loadConfig()
        self.assertEqual(saved_config, config)
        import os
        os.remove(temp_file_name)

# 测试 getDefaultConfig() 方法
def test_get_default_config(self):
    default_config = self.user_settings.getDefaultConfig()

    self.assertEqual(default_config, {
        "username": "默认用户",
        "selectedCity": "南京",
        "alerts": [False, False],
        "charac": "浅色"
    })

# 测试 SettingsDialog: 打开窗口并设置用户名
@patch('user_settings.UserSettings.loadConfig', return_value={"username": "test_user"})
def test_open_settings_dialog(self, mock_load_config):
    dialog = SettingsDialog(self.user_settings)
    dialog.username_edit.setText("new_user")
    dialog.saveSettings()
    self.assertEqual(dialog.username_edit.text(), "new_user")

# 测试 SettingsDialog: 点击保存按钮后更新配置
@patch('user_settings.UserSettings.saveConfig')
@patch('user_settings.UserSettings.loadConfig', return_value={"username": "old_user"})
def test_save_settings(self, mock_load_config, mock_save_config):

```

```

e_config):
    dialog = SettingsDialog(self.user_settings)
    dialog.username_edit.setText("new_user")
    dialog.saveSettings()
    mock_save_config.assert_called_once()
    self.assertEqual(dialog.username_edit.text(), "new_
user")

# 测试 SettingsDialog: 恢复默认设置
@patch('user_settings.UserSettings.saveConfig')
def test_reset_to_default(self, mock_save_config):
    dialog = SettingsDialog(self.user_settings)
    dialog.username_edit.setText("changed_user")
    dialog.resetToDefaultSettings()
    self.assertEqual(dialog.username_edit.text(), "默认
用户")
    mock_save_config.assert_called_once_with(dialog.user_
settings.getDefaultConfig())

# 测试 SettingsDialog: 切换菜单并加载不同页面
def test_switch_content(self):
    dialog = SettingsDialog(self.user_settings)
    dialog.switchContent(0) # 切换到第一个页面
    self.assertEqual(dialog.content_stack.currentIndex
(), 0) # 个人账户页面

# 测试 SettingsDialog: 点击取消按钮
def test_cancel_dialog(self):
    dialog = SettingsDialog(self.user_settings)
    dialog.reject() # 模拟取消操作
    self.assertFalse(dialog.isVisible()) # 确保窗口关闭

if __name__ == "__main__":
    unittest.main()

```

测试结果如下：

```
测试结果 终端 端口 GITLENS SQL CONSOLE 调试控制台 问题 输出
>>>PYTHON-EXEC-OUTPUT
Received test ids from temp file.
test_cancel_dialog (test_user_settings.TestUserSettings.test_cancel_dialog) ... ok
test_get_default_config (test_user_settings.TestUserSettings.test_get_default_config) ... ok
test_load_config_file_not_found (test_user_settings.TestUserSettings.test_load_config_file_not_found) ... ok
test_load_config_invalid_json (test_user_settings.TestUserSettings.test_load_config_invalid_json) ... ok
test_load_config_valid (test_user_settings.TestUserSettings.test_load_config_valid) ... ok
test_open_settings_dialog (test_user_settings.TestUserSettings.test_open_settings_dialog) ... ok
test_reset_to_default (test_user_settings.TestUserSettings.test_reset_to_default) ... ok
test_saveConfig (test_user_settings.TestUserSettings.test_saveConfig) ... ok
test_save_settings (test_user_settings.TestUserSettings.test_save_settings) ... ok
test_switch_content (test_user_settings.TestUserSettings.test_switch_content) ... ok
<<<PYTHON-EXEC-OUTPUT

-----
Ran 10 tests in 0.819s

OK
Finished running tests!
```

Running Tests for Workspace(s): c:\Users\Zly\OneDrive\SE\M...

- test_cancel_dialog
- test_get_default_config
- test_load_config_file_not_found
- test_load_config_invalid_json
- test_load_config_valid
- test_open_settings_dialog
- test_reset_to_default
- test_saveConfig
- test_save_settings
- test_switch_content

> 39 个较旧的结果

使用coverage 检查的语句覆盖率

File ▲	statements	missing	excluded	coverage
tests\test_user_settings.py	75	1	0	99%
user_settings.py	128	3	0	98%
Total	203	4	0	98%

coverage.py v7.6.9, created at 2024-12-16 15:44 +0800

coverage 报告用户设置模块的语句覆盖率达到98%，相关模块总体语句覆盖率为98%

集成测试报告

自顶向下集成测试

程序顶层为 `weather_app.py`，自顶向下集成测试主要检查其能否与支持其工作的各模块正确交互，包含如下5组测试：

```
### test_initial_ui_elements
```

该测试验证`WeatherApp`主界面上各个UI元素是否都已正确创建并且能够显示。测试过程中检查了所有关键标签（如城市标签、温度标签、湿度风力标签等）是否都已初始化，并且不会为空。此测试的目的是确保用户界面组件的基本存在和正常工作，为后续的交互和数据显示做好准备。

```
### test_weather_data_update
```

该测试确保`WeatherService`模块与`WeatherApp`界面之间的数据传递是

有效的。测试过程中通过模拟天气服务返回的数据，更新了界面中的天气信息（如温度、湿度、天气描述等）。最终通过断言验证了UI上显示的数据与从`WeatherService`获取的数据是否一致，确保数据的正确传递和展示。

test_user_settings_load_and_update

该测试验证用户设置模块（`UserSettings`）的加载和保存功能。在测试中，首先创建了一个模拟的用户设置文件，并确保程序能够正确加载该文件中的内容。然后通过更新设置并保存，测试了`UserSettings`模块是否能够正确更新配置文件，并确保界面显示了更新后的设置。此测试涉及到了多个模块的协同工作，包括文件系统、设置加载和保存，以及与UI的交互。

test_weather_alert

该测试验证`NotificationService`模块是否能够根据天气条件正确显示预警信息。在测试中，模拟了一个风雨交加的天气情况，测试了`NotificationService`能否根据这些条件生成并显示适当的预警信息。通过确保预警信息被正确显示在界面上，验证了`NotificationService`与`WeatherApp`的协同工作是否顺畅。此测试强调了数据输入（天气信息）与输出（UI更新）的整合。

test_save_and_load_user_settings

该测试检查`UserSettings`模块的保存和加载功能是否能够正常工作。通过修改并保存新的用户设置，接着加载保存后的设置并验证其正确性，确保了配置文件的读写操作与程序的其他模块（如设置对话框和UI）能有效协同工作。此测试通过操作文件系统、设置保存和加载的过程，验证了模块间的协作性。

在 `./tests` 中创建 `alltest_top2but.py`，编写测试代码如下

```
import json
import os
import unittest
from PyQt5.QtWidgets import QApplication
from weather_app import WeatherApp
from user_settings import UserSettings
from weather_service import WeatherService
from notification_service import NotificationService
from weather import Weather
import sys

class TestWeatherAppIntegration(unittest.TestCase):
```

```

@classmethod
def setUpClass(cls):
    """在所有测试开始之前运行一次，初始化应用程序和主界面。"""
    cls.app = QApplication(sys.argv)
    cls.weather_app = WeatherApp()
    cls.weather_app.start()
    cls.weather_service = WeatherService()
    cls.user_settings = UserSettings()
    cls.notification_service = NotificationService(cls.
weather_app.alert_label)

    def test_initial_ui_elements(self):
        """测试各个标签是否存在"""
        city_label = self.weather_app.city_label
        self.assertIsNotNone(city_label, "City label should
exist.")
        grt_label = self.weather_app.grt_label
        self.assertIsNotNone(grt_label, "Greeting label sho
uld exist.")
        tmp_label = self.weather_app.tmp_label
        self.assertIsNotNone(tmp_label, "Temperature label
should exist.")
        hum_wid_label = self.weather_app.hum_wid_label
        self.assertIsNotNone(hum_wid_label, "Humidity and w
ind label should exist.")
        foc_label = self.weather_app.foc_label
        self.assertIsNotNone(foc_label, "Focus label should
exist.")
        wth_label = self.weather_app.wth_label
        self.assertIsNotNone(wth_label, "Weather descriptio
n label should exist.")
        alert_label = self.weather_app.alert_label
        self.assertIsNotNone(alert_label, "Alert label shou
ld exist.")
        charac_label = self.weather_app.charac_label
        self.assertIsNotNone(charac_label, "Character label
should exist.")

```

```

def test_weather_data_update(self):
    """测试天气数据是否能够从 WeatherService 正确获取并显示。"""
    # 模拟从天气服务获取的天气数据
    weather = self.weather_service.getCurrentWeather("南京")

    # 更新界面上的天气信息
    self.weather_app.tmp_label.setText(f"{weather.temperature}°")
    self.weather_app.hum_wid_label.setText(f"{weather.get_wind_info()}级 | 湿度{weather.humidity}%")
    self.weather_app.foc_label.setText(f"{weather.forecast[0]}° {weather.forecast[1]}°")
    self.weather_app.wth_label.setText(weather.weather)

    # 验证数据是否正确显示
    self.assertEqual(self.weather_app.tmp_label.text(), f"{weather.temperature}°")
    self.assertEqual(self.weather_app.hum_wid_label.text(), f"{weather.get_wind_info()}级 | 湿度{weather.humidity}%")
    self.assertEqual(self.weather_app.wth_label.text(), weather.weather)

def test_user_settings_load_and_update(self):
    """测试用户设置是否能够加载并在界面中正确反映。"""
    # 无配置文件时模拟初始的配置文件内容
    self.settings_file = "user_settings.json"
    self.user_settings = UserSettings()
    if not os.path.exists(self.settings_file):
        settings = {
            "username": "user_test",
            "selectedCity": "深圳",
            "alerts": [True, True],
            "charac": "风见幽香"
        }

```



```

        with open(self.settings_file, "w") as file:
            json.dump(settings, file, indent=4)

# 直接读取配置文件中的内容
with open(self.settings_file, "r") as file:
    expected_settings = json.load(file)

# 加载用户设置
settings = self.user_settings.loadConfig()

# 验证加载的设置是否与文件中的实际内容匹配
self.assertEqual(settings["username"], expected_settings["username"])
self.assertEqual(settings["selectedCity"], expected_settings["selectedCity"])
self.assertEqual(settings["alerts"], expected_settings["alerts"])
self.assertEqual(settings["charac"], expected_settings["charac"])

# 更新设置
new_settings = {
    "username": "new_user",
    "selectedCity": "上海",
    "alerts": [False, False],
    "charac": "八云紫"
}

self.user_settings.saveConfig(new_settings)

# 验证设置是否已更新
with open(self.settings_file, "r") as file:
    updated_settings = json.load(file)

self.assertEqual(updated_settings["username"], "new_user")
self.assertEqual(updated_settings["selectedCity"], "上海")

```

```

        self.assertEqual(updated_settings["alerts"], [False, False])
        self.assertEqual(updated_settings["charac"], "八云紫")

    def test_weather_alert(self):
        """测试天气预警功能是否正确工作。"""
        # 模拟一个有风和降水的天气情况
        weather = Weather(
            temperature=22,
            humidity=70,
            is_windy=True,
            is_rainy=True,
            wind_direction="东风",
            wind_level="4级",
            forecast=[20, 21],
            weather="雷阵雨"
        )

        # 发送天气预警
        self.notification_service.sendWeatherAlert(weather)

        # 检查是否显示了正确的预警信息
        self.assertTrue(self.weather_app.alert_label.isVisible())
        self.assertIn("大风预警", self.weather_app.alert_label.text())
        self.assertIn("降水预警", self.weather_app.alert_label.text())

    def test_save_and_load_user_settings(self):
        """测试设置保存和加载功能。"""
        # 修改用户设置
        new_settings = {
            "username": "user_test",
            "selectedCity": "深圳",
            "alerts": [True, True],
            "charac": "风见幽香"

```

```

    }
    self.user_settings.saveConfig(new_settings)

    # 重新加载配置
    loaded_settings = self.user_settings.loadConfig()

    # 验证保存的设置是否正确加载
    self.assertEqual(loaded_settings["username"], "user_test")
    self.assertEqual(loaded_settings["selectedCity"], "深圳")
    self.assertEqual(loaded_settings["alerts"], [True, True])
    self.assertEqual(loaded_settings["charac"], "风见幽香")

    @classmethod
    def tearDownClass(cls):
        """在所有测试结束后执行清理工作。"""
        cls.weather_app.stop()

if __name__ == "__main__":
    unittest.main()

```

测试结果如下：

The screenshot shows a test runner interface with two main panels. The left panel displays the output of a Python test run, including test names, durations, and results. The right panel shows a list of tests that were run, with a summary of the results.

Test Results (Left Panel):

```

>>>PYTHON-EXEC-OUTPUT
Received test ids from temp file.
[27732:22944:1218/183554.153:ERROR:rarecc_manager.cpp(923)] UserJsonFile::Load failed
[27732:22944:1218/183554.153:ERROR:rarecc_manager.cpp(674)] Load rarecc.dat failed:
test_initial_ui_elements (alltest_top2but.TestWeatherAppIntegration.test_initial_ui_elements)
测试各个标签是否存在 ... ok
test_save_and_load_user_settings (alltest_top2but.TestWeatherAppIntegration.test_save_and_load_us
er_settings)
测试设置保存和加载功能。 ... ok
test_user_settings_load_and_update (alltest_top2but.TestWeatherAppIntegration.test_user_settings_
load_and_update)
测试用户设置是否能够加载并在界面中正确反映。 ... ok
test_weather_alert (alltest_top2but.TestWeatherAppIntegration.test_weather_alert)
测试天气预警功能是否正确工作。 ... ok
test_weather_data_update (alltest_top2but.TestWeatherAppIntegration.test_weather_data_update)
测试天气数据是否能够从 WeatherService 正确获取并显示。 ... ok
-----
Ran 5 tests in 4.475s

OK
<<<PYTHON-EXEC-OUTPUT
Finished running tests!

```

Running Tests (Right Panel):

```

Running Tests for Workspace(s): c:\Users\Zly\O...
✓ test_initial_ui_elements
✓ test_save_and_load_user_settings
✓ test_user_settings_load_and_update
✓ test_weather_alert
✓ test_weather_data_update
> 40 个较旧的结果

```

模糊测试报告

工具选择

Hypothesis作为一种模糊测试工具，可以自动生成输入内容，测试策略灵活，且易与**pytest**结合使用

安装截图

```
PS C:\Users\Zly\OneDrive\SE\MyWeather> pip install hypothesis
Looking in indexes: https://mirror.nju.edu.cn/pypi/web/simple
Collecting hypothesis
  Downloading https://mirror.nju.edu.cn/pypi/web/packages/66/cb/44fe7e78c3cfbc01f905b3b252eff6396e2f2e8e88b2d27b5140a6ac474/hypothesis-6.122.3-py3-none-any.whl (475 kB)
    475.7/475.7 kB 15.0 MB/s eta 0:00:00
Requirement already satisfied: attrs>=22.2.0 in c:\users\zly\anaconda3\lib\site-packages (from hypothesis) (23.1.0)
Requirement already satisfied: sortedcontainers<3.0.0,>=2.1.0 in c:\users\zly\anaconda3\lib\site-packages (from hypothesis) (2.4.0)
Installing collected packages: hypothesis
Successfully installed hypothesis-6.122.3
```

测试代码

项目中风险最大的模块当属天气服务模块 `weather_service.py`，因为它调用了api，且需要从网络获取数据，并解析数据获取所需信息

```
### test_get_weather_normal
```

该测试验证`WeatherService`模块在正常情况下的功能是否正常。通过模拟一个有效的天气API响应，确保`WeatherService`能够正确解析返回的数据并将其转化为一个`Weather`对象。测试过程中，模拟的数据包括温度、湿度、风力、天气描述等信息，并通过断言验证了`Weather`对象的属性是否与模拟数据一致。

```
### test_get_weather_empty_response
```

该测试验证`WeatherService`如何处理API返回空数据的情况。通过模拟一个空响应，确保`WeatherService`能够返回一个包含默认值的`Weather`对象，而不会崩溃或出现异常。

在 `./tests` 中创建 `mhstest_weather_service.py`，编写测试代码如下

```
import unittest
from unittest.mock import Mock, patch
from hypothesis import given, strategies as st
from weather_service import WeatherService
from weather import Weather
import json

# 模拟的API返回数据
```

```

def mock_get_weather_data(city, params):
    """模拟API返回的天气数据"""
    mock_response = Mock()
    # 设置模拟的返回状态码
    mock_response.status_code = 200 # 模拟一个成功的响应
    # 模拟 `raise_for_status()` 方法，不抛出异常
    mock_response.raise_for_status = Mock()
    # 设置模拟的响应内容
    mock_response.json.return_value = {
        "code": "200",
        "now": {
            "temp": "20", # 温度
            "humidity": "50", # 湿度
            "windScale": "3", # 风力
            "text": "晴", # 天气描述
            "windDir": "东风", # 风向
        },
        "hourly": [
            {"temp": "19"}, # 未来1小时的温度
            {"temp": "21"} # 未来2小时的温度
        ]
    }

    return mock_response


class TestWeatherService(unittest.TestCase):

    # 测试WeatherService正常数据处理
    @given(st.text())
    def test_get_weather_normal(self, city):
        weather_service = WeatherService()

        # 使用patch模拟API请求，返回模拟数据
        with patch('requests.get', side_effect=mock_get_weather_data):
            weather = weather_service.getCurrentWeather(city)

```

```

        # 检查返回的Weather对象的各个属性
        self.assertIsInstance(weather, Weather)
        self.assertIsInstance(weather.temperature, int)
        self.assertIsInstance(weather.humidity, int)
        self.assertIsInstance(weather.is_windy, bool)
        self.assertIsInstance(weather.is_rainy, bool)
        self.assertIsInstance(weather.wind_direction, str)

        self.assertIsInstance(weather.wind_level, str)
        self.assertIsInstance(weather.forecast, list)
        self.assertIsInstance(weather.weather, str)
        self.assertEqual(weather.weather, "晴") # 根据
模拟数据检查

```

```

# 测试WeatherService应对API返回空数据
@given(st.text())
def test_get_weather_empty_response(self, city):
    # 模拟API返回空数据
    def mock_get_weather_data_empty(city):
        return {}

    weather_service = WeatherService()

    # 使用patch模拟API请求, 返回空数据
    with patch('requests.get', side_effect=mock_get_weather_data_empty):
        weather = weather_service.getCurrentWeather(city)

```

```

        # 检查返回的Weather对象是否是默认值
        self.assertIsInstance(weather, Weather)
        self.assertEqual(weather.temperature, 20) # 默认值

        self.assertEqual(weather.humidity, 50) # 默认值
        self.assertFalse(weather.is_windy) # 默认值
        self.assertFalse(weather.is_rainy) # 默认值
        self.assertEqual(weather.weather, "未知") # 默认值

```

```
if __name__ == "__main__":  
    unittest.main()
```

运行测试代码，发现一处缺陷

另一个重要模块是 `user_settings.py`，因为涉及到本地 `json` 文件的读写

```
### test_load_default_config_when_file_not_exists
```

该测试验证`UserSettings`模块在配置文件不存在时，是否能够正确加载默认配置。通过确保配置文件未存在或损坏，`UserSettings`应返回预定义的默认设置，如用户名、城市、预警设置和看姑娘等字段的默认值。

```
### test_save_and_load_config
```

该测试使用`Hypothesis`生成不同的输入数据，验证`UserSettings`是否能够正确保存和加载配置。测试覆盖了用户的用户名、城市选择、预警设置以及看姑娘的选择，通过断言确保保存后的数据能够被正确加载，并与原始输入数据一致。

```
### test_save_and_load_individual_keys
```

该测试验证`UserSettings`在保存和加载配置的过程中，是否能正确处理单独的配置项。通过保存并加载每个独立配置项（如用户名、城市、预警设置和看姑娘），测试确保每个配置项能被准确保存和读取。

```
### test_reset_to_default
```

该测试验证`UserSettings`是否能够恢复默认配置。通过保存一个自定义配置后，再将配置恢复为默认值，测试确保配置能够恢复到初始默认状态，并且所有字段都符合默认配置。

在 `./tests` 中创建 `mhtest_user_settings.py`，编写测试代码如下

```
import unittest  
from hypothesis import given, strategies as st  
from user_settings import UserSettings  
import json  
import os  
  
class TestUserSettings(unittest.TestCase):
```

```

def setUp(self):
    """在每个测试前清除配置文件，确保测试的独立性。"""
    self.settings_file = "user_settings.json"
    if os.path.exists(self.settings_file):
        os.remove(self.settings_file)

def tearDown(self):
    """清理测试后的设置文件。"""
    if os.path.exists(self.settings_file):
        os.remove(self.settings_file)

def test_load_default_config_when_file_not_exists(self):
    """测试配置文件不存在时，加载默认配置。"""
    user_settings = UserSettings()
    config = user_settings.loadConfig()

    default_config = user_settings.getDefaultConfig()
    self.assertEqual(config, default_config)

@given(
    st.text(), # 用户名
    st.text(), # 城市
    st.lists(st.booleans(), min_size=2, max_size=2), #
alerts
    st.text() # 看板娘
)
def test_save_and_load_config(self, username, city, alerts, charac):
    """测试保存和加载配置功能。"""
    user_settings = UserSettings()

    # 创建模拟配置
    config = {
        "username": username,
        "selectedCity": city,
        "alerts": alerts,

```



```

        "charac": charac
    }

    # 保存配置
    user_settings.saveConfig(config)

    # 加载并验证配置
    loaded_config = user_settings.loadConfig()

    self.assertEqual(config["username"], loaded_config["username"])
    self.assertEqual(config["selectedCity"], loaded_config["selectedCity"])
    self.assertEqual(config["alerts"], loaded_config["alerts"])
    self.assertEqual(config["charac"], loaded_config["charac"])

    @given(
        st.text(min_size=1), # 用户名
        st.text(min_size=1), # 城市
        st.lists(st.booleans(), min_size=2, max_size=2), # alerts
        st.text(min_size=1) # 看板娘
    )
    def test_save_and_load_individual_keys(self, username, city, alerts, charac):
        """测试保存和加载单个配置项。"""
        user_settings = UserSettings()

        # 保存单个配置项
        user_settings.saveConfig({
            "username": username,
            "selectedCity": city,
            "alerts": alerts,
            "charac": charac
        })

```

```

        # 验证每个键的加载
        self.assertEqual(user_settings.loadUserName(), user
name)
        self.assertEqual(user_settings.loadSelectedCity(),
city)
        self.assertEqual(user_settings.loadAlerts(), alert
s)
        self.assertEqual(user_settings.loadCharac(), chara
c)

    def test_reset_to_default(self):
        """测试恢复到默认配置。"""
        user_settings = UserSettings()

        # 创建并保存一个自定义配置
        custom_config = {
            "username": "CustomUser",
            "selectedCity": "北京",
            "alerts": [True, True],
            "charac": "八云紫"
        }
        user_settings.saveConfig(custom_config)

        # 重置为默认配置
        user_settings.saveConfig(user_settings.getDefaultCo
nfig())

        # 验证配置是否恢复为默认
        default_config = user_settings.getDefaultConfig()
        self.assertEqual(user_settings.loadConfig(), default_
t_config)

if __name__ == "__main__":
    unittest.main()

```

运行测试代码，未发现任何缺陷

缺陷：非法查询城市

在测试天气服务模块条目 `test_get_weather_normal` 时发生如下错误：

```
File "c:\Users\Zly\OneDrive\SE\MyWeather\weather_service.py", line 15, in getCurrentWeather
    city = self.city_id[city] # 默认南京
    |         |         | ~~~~~^
KeyError: ''
Falsifying example: test_get_weather_normal(
    self=<mhtest_weather_service.TestWeatherService testMethod=test_get_weather_normal>,
    city='',
)
```

测试条目用随机生成的文本代替城市名尝试获取天气，在 `weather_service.py` 第15行，使用预存字典将城市名转城市ID这一步，出现了 `KeyError: ''`

这暴露的缺陷为**程序在城市名转码这一步上没有对非法的查询城市做应对**

崩溃复现

手动改写配置文件中储存的查询城市，然后启动程序

```
{
    "username": "new_user",
    "selectedCity": "",
    "alerts": [
        false,
        false
    ],
    "charac": "\u516b\u4e91\u7d2b"
}
```

然而，崩溃没有出现。原因为城市为空字段时，我的程序初始化载入本地配置文件失败，会直接启用默认设置，因此在查询天气时规避了问题

再次修改查询城市，改成不支持的非法城市，则程序无法正常启动

```
{
    "username": "new_user",
    "selectedCity": "\u4e0a\u6d77",
    "alerts": [
        false,
        false
    ],
}
```

```

    "charac": "\u516b\u4e91\u7d2b"
}

```

```

PS C:\Users\Zly\OneDrive\SE\MyWeather> python main.py
libpng warning: iCCP: known incorrect sRGB profile
Traceback (most recent call last):
  File "C:\Users\Zly\OneDrive\SE\MyWeather\main.py", line 7, in <module>
    weather_app = WeatherApp()
                  ^^^^^^^^^^^
  File "C:\Users\Zly\OneDrive\SE\MyWeather\weather_app.py", line 20, in __init__
    self.initUI()
  File "C:\Users\Zly\OneDrive\SE\MyWeather\weather_app.py", line 56, in initUI
    self.refresh()
  File "C:\Users\Zly\OneDrive\SE\MyWeather\weather_app.py", line 119, in refresh
    weather = self.weather_service.getCurrentWeather(self.city)
              ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\Zly\OneDrive\SE\MyWeather\weather_service.py", line 15, in getCurrentWeather
    city = self.city_id[city] # 默认南京
           ~~~~~^~~~~
KeyError: '上'

```

植入缺陷1：取消网络请求失败时的应对

调用api时，若出现error则进行处理。这里我们故意取消对网络请求失败的处理，不返回默认值，而是直接退出程序，作为植入的缺陷

```

except requests.RequestException as e:
    print(f"网络请求失败：{e}")
    exit() # 植入的缺陷
except ValueError as e:
    print(f"数据解析错误：{e}")
except Exception as e:
    print(f"未知错误：{e}")

# 如果发生错误，返回默认值
return Weather(
    ... # 默认值
)

```

崩溃复现

启动程序后，断设备的网络连接，然后点击刷新按钮

```

PS C:\Users\Zly\OneDrive\SE\MyWeather> python main.py
libpng warning: iCCP: known incorrect sRGB profile
[31348:33524:1218/203218.337:ERROR:rarecc_manager.cpp(923)] UserJsonFile::Load failed
[31348:33524:1218/203218.337:ERROR:rarecc_manager.cpp(674)] Load rarecc.dat failed:
用户设置已更新: {'username': 'new_user', 'selectedCity': '南京', 'alerts': [False, False], 'charac': '八云紫'}
网络请求失败: HTTPConnectionPool(host='devapi.qweather.com', port=443): Max retries exceeded with url: /v7/weather/now?location=101190101&key=1065a96d69544fd3aa192c26b5eac905 (Caused by SSLError(SSLError(8, '[SSL: UNEXPECTED_EOF_WHILE_READING] EOF occurred in violation of protocol (_ssl.c:1006)'))))

```

植入缺陷2：取消加载配置失败时的应对

若本地配置丢失或损坏，我的程序会自动启动默认配置。这里我们故意取消这一应对，改为直接退出，作为植入缺陷

```
def loadConfig(self):
    """加载用户配置文件，若文件不存在或损坏则返回默认配置。"""
    try:
        with open(self.settings_file, "r") as file:
            return json.load(file)
    except (FileNotFoundError, json.JSONDecodeError):
        exit() # 植入的缺陷
        # return self.getDefaultConfig()
```

崩溃复现：

恶意破坏配置文件，然后尝试启动程序，直接无法启动，且无任何反馈

```
● PS C:\Users\Zly\OneDrive\SE\MyWeather> python main.py
○ PS C:\Users\Zly\OneDrive\SE\MyWeather> █
```

程序修复报告

与大模型对话来修复缺陷。首先我们给大模型提供天气服务模块的源码，并要求其跟随我们的节奏逐个修复前两个缺陷；然后再给出用户设置模块的源码，要求其修复最后一个缺陷：

```
# User:
如下是一个调用外部api获取天气信息的模块代码，其中存在一些缺陷。你先阅读代码，现在不必指出缺陷。稍后我会逐一指出缺陷，届时再尝试修复缺陷。源代码如下：
import requests
from weather import Weather

... # weather_service.py 的源码

# GPT:
```

我已经阅读了代码，了解了它的结构和功能。请指出你认为存在的缺陷，我将一一帮你修复。

修复缺陷1

User:

如你所见，这个查询程序只支持五个城市，但我没有对不支持的非法城市做拦截。请你修复这个缺陷，当查询的城市不在字典中时，在终端输出必要的提示信息，然后默认采用北京进行查询。只需要给出有修改的代码，不需要给出全部代码

GPT:

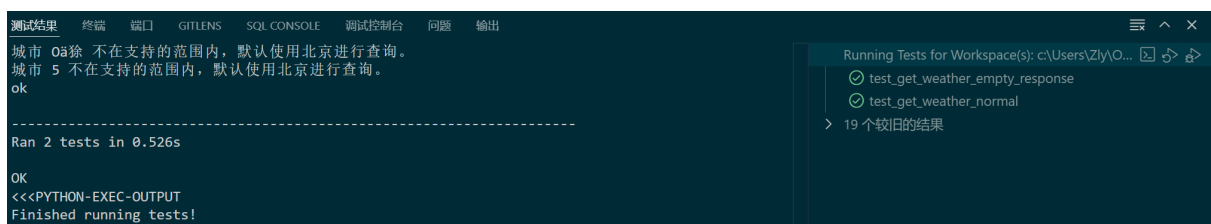
```
def getCurrentWeather(self, city):  
    # 城市名转代码  
    if city not in self.city_id:  
        print(f"城市 {city} 不在支持的范围内，默认使用北京进行查  
询。")  
        city = "北京" # 默认使用北京
```

```
        city = self.city_id[city]
```

后续代码保持不变...

这段代码会检查传入的城市是否在支持的城市列表中。如果不在，则输出提示并将城市默认设置为北京

直接采用GPT的修改方案，效果如下（重测模糊测试）、



修复缺陷2

User:

第二个缺陷：网络请求失败时，程序会直接调用exit()来结束。但我现在希望改变这一行为：网络请求失败后，间歇1秒再次尝试。总计尝试三次后失败，则在控

制台输出必要的提示信息，然后返回默认值

```
# GPT:
...
    # 重试次数
    retries = 3
    for attempt in range(retries):
        try:
            ...
        except requests.RequestException as e:
            print(f"网络请求失败，尝试重新连接 ({attempt + 1}/
{retries}): {e}")
            time.sleep(1) # 间隔1秒后重试
            ...

    # 如果重试三次仍然失败，输出提示信息并返回默认值
    print("网络请求失败，已达到最大重试次数，返回默认值。")

    # 返回默认值
    return Weather(
        ...
    )
```

主要修改点：

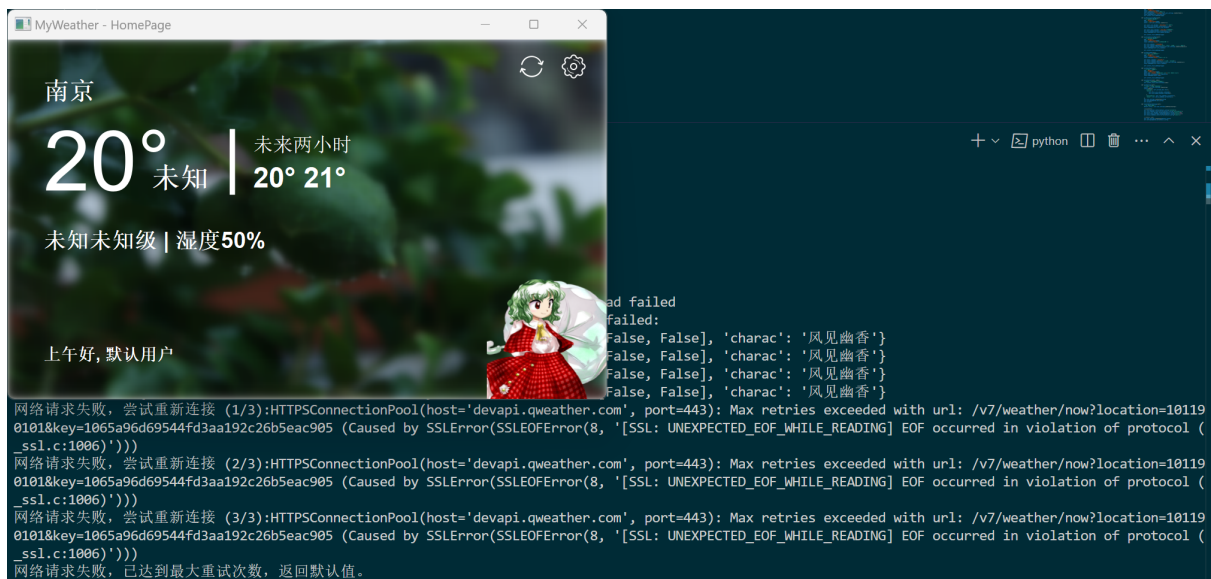
添加了 `retries` 变量来控制最大重试次数（设为3次）。

使用 `for` 循环来进行重试，失败后等待1秒（`time.sleep(1)`）。

如果请求三次都失败，则输出提示信息，并返回默认值。

这样，程序在遇到网络请求失败时会重试3次，并且在三次失败后输出提示并返回默认值。

直接采用GPT的修改方案，效果如下（断网情况下刷新天气）



修复缺陷3

User:

这个模块的任务已经完成。如下是另一个模块：用户设置模块。可以看到，当配置文件缺失或损坏时，程序直接退出了。我希望其return self.getDefaultConfig()而不是直接exit，同时在控制台输出必要的信息

... # user_settings.py 的部分源码

GPT:

```
def loadConfig(self):
```

```
    """加载用户配置文件，若文件不存在或损坏则返回默认配置。"""
```

```
    try:
```

```
        with open(self.settings_file, "r") as file:
```

```
            return json.load(file)
```

```
    except (FileNotFoundError, json.JSONDecodeError) as e:
```

```
        print(f"配置文件加载失败：{e}。将使用默认配置。")
```

```
        return self.getDefaultConfig()
```

主要修改点：

捕获异常后不退出程序：原先的 exit() 被移除，改为输出错误信息，并返回默认配置。

输出提示信息：当配置文件加载失败时，程序会在控制台输出一条说明，告知用户配置文件加载失败，并且会使用默认配置。

这样，在配置文件缺失或损坏时，程序不会直接退出，而是会继续运行并使用默认配置。

直接采用GPT的修改方案（其实就是我植入缺陷前的原本方案），效果如下（删除配置文件后从尝试启动）



软件运行说明

运行环境

python – 3.11.7

requests – 2.31.0

PyQt5 – 5.15.10

运行方式

1. 从 `.py` 文件启动

在项目文件夹启动终端并配置好运行环境，然后使用**python**打开 `main.py`

```
> pip install requests PyQt5
> python main.py
```

2. 从 `.exe` 程序启动

双击运行 `./main/main.exe` 即可

报告结束