Класс BeaconWrapper

Класс представляет собой адаптер для работы с сервисом мониторинга. Используется для отслеживания состояния сети.

Атрибуты класса BeaconWrapper:

Имя атрибута	**Тип**	**Описание**
selfaddress	public : string	Адрес, под которым адаптер был зарегистрирован на сервисе мониторинга
beacon_adapter_cycletime	public : int	Интервал опроса адаптером сервиса мониторинга
beacon	public : string	Адрес сервиса мониторинга
backend_port	public : int	Порт, под которым адаптер регистрируется на сервисе мониторинга
seterr	public : bool	Наличие ошибок регистрации на сервисе мониторинга
geterr	public : bool	Наличие ошибок забора данных на сервисе мониторинга
state	public : string	Состояние сервиса мониторинга
setterEndpoint	public : string	Удалённый ресурс, на котором регистрируется адаптер в сервисе мониторинга
addresses	public : Dict<string, string>	Список адресов сервисов, за состоянием которых следит адаптер

Методы класса BeaconWrapper:

Имя метода	**Описание**
__init__(string, int, string, [string])	Конструирует объект адаптера заданного сервиса мониторинга к сервису с заданными портом. Адаптер отслеживает адреса переданных типов узлов.
beacon_setter()	Осуществляет регистрацию узла на сервисе мониторинга.
beacon_getter()	Осуществляет отслеживание состояния других сервисов.

Модульное тестирование

Производится тестирование класса BeaconWrapper, с использованием методики тестирования – разбиение на уровне класса на категории по функциональности. Категория объединяет в себе методы класса, выполняющие близкую по смыслу функциональность.

Методы класса можно разбить на две категории по функциональности:

- Методы инициализации – создание адаптера;
- Методы операций с адаптером – оповещение сервиса мониторинга о состоянии сервиса, получение данных о других сервисах;

Категория 1 – тестирование методов инициализации

Название теста	TestCreation
Тестируемый метод	TestCreation.__init__
Описание теста	Проверка создания адаптера. Сервис мониторинга развёрнут по

	указанному адресу.
Степень важности ошибки	Фатальная
Ожидаемый результат	Создан адаптер к сервису мониторинга.
Результат	Тест пройден

Категория 2 – тестирование методов работы с адаптером

Название теста	TestSetterGood
Тестируемый метод	TestCreation.beaconSetter
Описание теста	Проверка, что адаптер регистрирует тестовый сервис на сервисе мониторинга. Ошибок соединения нет.
Степень важности ошибки	Фатальная
Ожидаемый результат	Адаптер переходит в состояние ‘stateNormal’. На сервисе мониторинга создана запись с адресом и именем, соответствующими тестовому сервису.
Результат	Тест пройден

Название теста	TestSetterBad
Тестируемый метод	TestCreation.beaconSetter
Описание теста	Проверка, что адаптер корректно определяет отсутствие соединения с сервисом мониторинга. Для имитации разрыва соединения, изменяется адрес сервиса мониторинга.
Степень важности ошибки	Фатальная
Ожидаемый результат	Адаптер переходит в состояние ‘stateError’. На сервисе мониторинга через заданный период неактивности запись о тестовом сервисе удаляется.
Результат	Тест пройден

Название теста	TestGetterGood
Тестируемый метод	TestCreation.beaconGetter
Описание теста	Проверка, что адаптер отслеживает адреса других сервисов через сервис мониторинга. Создан дополнительный тестовый сервис.
Степень важности ошибки	Фатальная
Ожидаемый результат	Адрес запрашиваемого сервиса.
Результат	Тест пройден

Название теста	TestGetterBad
Тестируемый метод	TestCreation.beaconGetter
Описание теста	Проверка, что адаптер отслеживает исчезновение отслеживаемых сервисов с сервиса мониторинга. Дополнительный тестовый сервис отключён от сервиса мониторинга.
Степень важности ошибки	Фатальная
Ожидаемый результат	Адаптер переходит в состояние ‘stateError’.
Результат	Тест пройден

Исходные тексты тестовых драйверов на языке Python:

```
from common.common import BeaconWrapper
from time import sleep
import requests, subprocess, sys, time

beacon = 'http://localhost:1666'
self_port = 1667
self_addr = '127.0.0.1:' + str(self_port)
badadr = 'http://localhost:1650'

name = 'test_service'
target = 'test_target1'
targets = {target}

bw = None
bw2 = None
def TestCreation():
  print ('Running TestCreation...')
  global bw
  bw = BeaconWrapper(beacon, self_port, 'services/' + name, targets)
  assert bw != None, 'TestCreation failed'
  print('TestCreation passed')

def TestSetterGood():
  print ('Running TestSetterGood...')
  bw.beacon_setter()
  time.sleep(3)
  assert requests.get(beacon + '/services').json()[name][self_addr]['state'] == bw.stateNormal,
'TestSetterGood failed'
  assert bw.state == bw.stateNormal, 'TestSetterGood failed'
  print('TestSetterGood passed')

def TestSetterBad():
  print ('Running TestSetterBad...')
  bw.beacon = badadr
  time.sleep(30)

  assert bw.state == bw.stateError
  try:
    st = requests.get(beacon + '/services').json()[name][self_addr]['state']
  except:
    bw.beacon = beacon
    print('TestSetterBad passed')
    return
  bw.beacon = beacon
  raise 'TestSetterBad failed'

def TestGetterGood():
  print ('Running TestGetterGood...')
  global bw2
  bw2 = BeaconWrapper(beacon, self_port + 1, 'services/' + target, {})
  bw2.beacon_setter()
  bw.beacon_getter()
```

```
    time.sleep(10)
    assert bw[target] == 'http://127.0.0.1:'+str(self_port+1), 'TestGetterGood failed'
    print('TestGetterGood passed')

def TestGetterBad():
    print ('Running TestGetterBad...')
    global bw2
    bw2.beacon = badadr
    time.sleep(40)
    print (bw.state)
    assert bw.state == bw.stateError, 'TestGetterBad failed'
    print('TestGetterBad passed')


DETACHED_PROCESS = 8
def run(str):
    subprocess.Popen('py ' + str, creationflags=DETACHED_PROCESS, close_fds=True)

if __name__ == '__main__':
    run('beacon_backend/beacon_backend.py 1666')
    time.sleep(5)

    TestCreation()
    TestSetterGood()
    TestSetterBad()
    TestGetterGood()
    TestGetterBad()
```

Выводы по результатам модульного тестирования

Возможно использование класса BeaconWrapper, все тесты пройдены успешно.