错误重试

问题引入

- 程序或用例的某些步骤，由于一些时效性的问题，可能执行失败，一旦失败则抛出异常，终止执行。例如：
  1. 创建redis/db连接的时候失败，若无重试机制，则判定本次请求失败，缺乏健壮性
  2. 异步请求的结果更新有时效性，若第一次查询失败则判定为失败，用例的失败率会增高
  3. 某些不稳定场景的测试，比如对接第三方接口，由于网络或环境的问题，导致请求本身不稳定
- 引入失败重试机制，并手动控制重试次数，重试间隔等，提高代码或自动化的健壮性

场景案例

- 若请求一个get接口，必须得到200，才能进行后续的操作，但该接口偶尔会返回非200

```
def add():
    # 模拟随机获取到的http返回码
    a = random.randint(200, 201)
    print("当前返回码是", a)
    assert a == 200

# 有概率执行失败，一旦失败，则终止
add()
```

通过try解决

通过方法封装来实现重试

```
def add():
    # 模拟随机获取到的http返回码
    a = random.randint(200, 201)
    print("当前返回码是", a)
    assert a == 200

# 改造函数
def do_1(retry=10):
    # 循环指定次数
    for i in range(retry):
        try:
            # 模拟随机返回码
            a = random.randint(200, 202)
            # 如果返回码异常，则抛出异常
            if a != 200:
```

```
                raise IOError("code is not 200, try again", a)
            else:
                # 如果返回码是200，则返回
                print("get expect result", a)
                return a
        except Exception as e:
            # 捕获刚刚抛出的异常
            print(e)

# 指定重试次数
do_1(retry=5)
```

```
get expect result 200
```

```
200
```

通过装饰器

用函数不太方便，单个实现尚可。若用例/程序中需要大量使用，则无法满足

```
import logging, random
from functools import wraps

# 定义装饰器，带参数则是三层嵌套
def retry_test(func):
    # 复制元信息
    @wraps(func)
    def wrapped(*args, **kwargs):
        last_raised = None
        # 重试次数，写死
        RETRIES_LIMIT = 3
        for i in range(RETRIES_LIMIT):
            try:
                return func(*args, **kwargs)
            except Exception as e: # 捕获异常
                print(e)
                last_raised = e
        if last_raised:
            raise last_raised

    return wrapped
```

```
@retry_test
def add():
    # 模拟随机获取到的http返回码
    a = random.randint(200, 202)
    print("当前返回码是", a)
    assert a == 200

add()
```

```
当前返回码是 201

当前返回码是 201

当前返回码是 200
```

带参数装饰器

```
from functools import wraps
import time, random


def do_retry(retry_times=1, delay_seconds=0):
    """通过参数指定重试次数，重试的间隔时间"""
    def decorator(func):
        @wraps(func)
        def wrapper(*args, **kwargs):
            err = None
            print(retry_times)
            # retry_times = retry_times + 1
            # 当重试次数大于0，一直执行循环
            while retry_times > 0:
                try:
                    err = None
                    # 执行函数，并返回结果
                    return func(*args, **kwargs)
                except Exception as e:
                    # 捕获异常
                    err = e
                    if delay_seconds > 0:
                        time.sleep(delay_seconds)
                    # 重试次数-1
                    retry_times -= 1
            # 如果err不为空，则抛出异常
            if err:
                raise err
        return wrapper
    return decorator
```

```
@do_retry(retry_times=10)
def add():
    # 模拟随机获取到的http返回码
    a = random.randint(200, 300)
    print("当前返回码是", a)
    assert a == 200


add()
```

```
---------------------------------------------------------------------------

UnboundLocalError                         Traceback (most recent call last)

<ipython-input-7-fb6a849f6d8a> in <module>
     43
     44
---> 45 add()
```

```
<ipython-input-7-fb6a849f6d8a> in wrapper(*args, **kwargs)
     10         def wrapper(*args, **kwargs):
     11             err = None
---> 12             print(retry_times)
     13             # retry_times = retry_times + 1
     14
```

```
UnboundLocalError: local variable 'retry_times' referenced before assignment
```

- 思考题：为什么闭包里面不能使用外部变量呢？`print(retry_times)` 报错

```
from functools import wraps
import time, random

# 当对于装饰器参数，仅仅是使用，而不改变，则不会报错
# 当需要对装饰器参数进行改变时，则会报错：referenced before assignment
def do_retry(retry_times=1, delay_seconds=0):
    def decorator(func):
        @wraps(func)
        def wrapper(*args, **kwargs):
            # 输出，而并没有操作
            print(retry_times)
            # 实际操作
            retry_times = retry_times + 1
            return func(*args, **kwargs)
```

```
        return wrapper

    return decorator


@do_retry(retry_times=10)
def add():
    # 模拟随机获取到的http返回码
    a = random.randint(200, 300)
    print("当前返回码是", a)
#     assert a == 200

add()
```

1. 现象解释：当在wrapper的局部作用于里面操作**不可变类型**的数据时，会生成新的局部变量，但是新生成的局部变量retry_times在使用时还没来得及初始化，因此会提示找不到变量。
2. 如何解决？将不可变参数用可变容器封装即可，即：闭包下如何捕获变量

```
from functools import wraps
import time, random

# 当对于装饰器参数，仅仅是使用，而不改变，则不会报错
# 当需要对装饰器参数进行改变时，则会报错：referenced before assignment
def do_retry(retry_times=1, delay_seconds=0):
    temp_dict = {
        'retry_times': retry_times,
        'delay_seconds': delay_seconds
    }
    def decorator(func):
        @wraps(func)
        def wrapper(*args, **kwargs):
            # 通过可变容器(字典)来获取参数
            retry_times = temp_dict.get('retry_times')
            print(retry_times)
            return func(*args, **kwargs)
        return wrapper

    return decorator


@do_retry(retry_times=4)
def add():
    # 模拟随机获取到的http返回码
    a = random.randint(200, 300)
    print("当前返回码是", a)
#     assert a == 200

add()
```

```
4
当前返回码是 203
```

优化版本

```python
from functools import wraps
import time, random


def do_retry(retry_times=1, delay_seconds=0):
    """通过参数指定重试次数，重试的间隔时间"""
    temp_dict = {
        'retry_times': retry_times,
        'delay_seconds': delay_seconds
    }
    def decorator(func):
        @wraps(func)
        def wrapper(*args, **kwargs):
            err = None
            retry_times = temp_dict.get('retry_times')
            # retry_times = retry_times + 1
            # 当重试次数大于0，一直执行循环
            while retry_times > 0:
                try:
                    err = None
                    # 执行函数，并返回结果
                    return func(*args, **kwargs)
                except Exception as e:
                    # 捕获异常
                    err = e
                    if delay_seconds > 0:
                        time.sleep(delay_seconds)
                    # 重试次数-1
                    retry_times -= 1
            # 如果err不为空，则抛出异常
            if err:
                raise err
        return wrapper
    return decorator


@do_retry(retry_times=10)
def add():
    # 模拟随机获取到的http返回码
    a = random.randint(200, 210)
    print("当前返回码是", a)
```

```
    assert a == 200


add()
```

```
当前返回码是 209
当前返回码是 207
当前返回码是 203
当前返回码是 207
当前返回码是 203
当前返回码是 200
```

最终版本

```
from functools import wraps
import time, random

def do_retry(retry_times=1, delay_seconds=0):
    """通过参数指定重试次数，重试的间隔时间"""
    temp_dict = {
        'retry_times': 3 if retry_times < 0 or retry_times > 5 else
retry_times,
        'delay_seconds': 2 if delay_seconds < 0 or delay_seconds > 5 else
delay_seconds
    }
    def decorator(func):
        @wraps(func)
        def wrapper(*args, **kwargs):
            err = None
            retry_times = temp_dict.get('retry_times')
            while retry_times > 0:
                try:
                    err = None
                    return func(*args, **kwargs)
                except Exception as e:
                    err = e
                    if delay_seconds > 0:
                        time.sleep(delay_seconds)
                    retry_times -= 1
            if err:
                raise err
        return wrapper
    return decorator


@do_retry(retry_times=4)
def add():
```

```
    # 模拟随机获取到的http返回码
    a = random.randint(200, 205)
    print("当前返回码是", a)
    assert a == 200


add()
```

```
当前返回码是 200
```

第三方库

有两个第三方库，针对重试情况实现了装饰器：tenacity 和 retrying

- pip install tenacity
- pip install retring

```
from retrying import retry
@retry
def add():
    # 模拟随机获取到的http返回码
    a = random.randint(200, 205)
    print("当前返回码是", a)
    assert a == 200
add()
```

```
当前返回码是 201
当前返回码是 202
当前返回码是 203
当前返回码是 205
当前返回码是 201
当前返回码是 205
当前返回码是 204
当前返回码是 204
当前返回码是 202
当前返回码是 200
```

使用说明：

1. stop_max_attempt_number：用来设定最大的尝试次数，超过该次数就停止重试
2. stop_max_delay：比如设置成10000，那么从被装饰的函数开始执行的时间点开始，到函数成功运行结束或者失败报错中止的时间点，只要这段时间超过10秒，函数就不会再执行了
3. wait_fixed：设置在两次retrying之间的停留时间
4. wait_random_min和wait_random_max：用随机的方式产生两次retrying之间的停留时间
5. wait_exponential_multiplier和wait_exponential_max：以指数的形式产生两次retrying之间的停留时间，产生的值为2^previous_attempt_number * wait_exponential_multiplier，

previous_attempt_number是前面已经retry的次数，如果产生的这个值超过了wait_exponential_max的大小，那么之后两个retrying之间的停留值都为wait_exponential_max。这个设计迎合了exponential backoff算法，可以减轻阻塞的情况。

6. 我们可以指定要在出现哪些异常的时候再去retry，这个要用retry_on_exception传入一个函数对象：

```
from retrying import retry
@retry(stop_max_attempt_number=10, stop_max_delay=2000, wait_fixed=1000)
def add():
    # 模拟随机获取到的http返回码
    a = random.randint(200, 300)
    print("当前返回码是", a)
    assert a == 200
add()
```

推荐tenacity

官方文档：https://tenacity.readthedocs.io/en/latest/

- 简单场景应用

```
from tenacity import retry, stop_after_attempt, wait_fixed
# stop_after_attempt指定重试次数
# wait_fixed指定重试间隔，单位秒
@retry(stop=stop_after_attempt(3), wait=wait_fixed(1))
def add():
    # 模拟随机获取到的http返回码
    a = random.randint(200, 300)
    print("当前返回码是", a)
    assert a == 200
add()
```

- 重试间隔指数递增

```
from tenacity import retry, stop_after_attempt, wait_fixed, wait_exponential
import random
# 重试时间间隔满足：2^n * multiplier，n为重试次数，但最多间隔10秒
@retry(stop=stop_after_attempt(3), wait=wait_exponential(multiplier=1,
max=10))
def add():
    # 模拟随机获取到的http返回码
    a = random.randint(200, 201)
    print("当前返回码是", a)
    assert a == 200
add()
```

- 特殊场景

1. requests获取网页出错
2. 解析JSON出错
3. info_dict字典里面没有data这个key

```
from tenacity import retry, stop_after_attempt, wait_fixed,
retry_if_exception_type
from json.decoder import JSONDecodeError
import json
@retry(stop=stop_after_attempt(3),
retry=retry_if_exception_type(JSONDecodeError))
def add():
    # 模拟随机获取到的http返回码
    # {"name": "dcs", "age": None}
    print("ok")
    a = json.loads('''{"name": "dcs", "age": None}''')
    print("当前返回码是", a)
add()
```

自动化用例场景

消息发送场景

1. 理清消息推送过程
2. 手动测试，关注：数据库/redis/celery
3. 练习：完成一个自动化用例，通过前端来校验结果

```
from tsms.tsms_web import TsmsWeb
from tsms.tsms_base import Tsmstest
import unittest
from time import sleep
from tenacity import retry
class TestWeb(unittest.TestCase):
    @classmethod
    def setUpClass(cls):
        cls.tb = TsmsWeb()
        cls.ts = Tsmstest()
        # 登录
        cls.tb.login_c()
        assert cls.tb.is_login()

    @classmethod
    def tearDownClass(cls):
        cls.tb.close()

    def test_add_sign(self):
        # 调接口
        phone = self.ts.gen_phones(1)
        data = {"sign_id": 424, "temp_id": 180, "mobiles": phone}
```

```
        self.ts.req_post('message', data)
        assert self.ts.status_code == 200
        assert isinstance(self.ts.json["uuid"], str)
        # 查前端
        send_list = self.tb.reg_send()
        real_res = self.tb.get_res(phone[0], send_list)
        assert real_res[1] == phone[0]
        assert real_res[3] == "success", "期待返回{}，实际返回
{}".format("success", real_res[3])
        assert real_res[4] == "1"

if __name__ == '__main__':
    # 执行本suite
    unittest.main(argv=['ignored', '-v'], exit=False)
```

失败重试

练习：针对异步请求结果的获取，进行失败重试/重新获取

```
from tsms.tsms_web import TsmsWeb
from tsms.tsms_base import Tsmstest
import unittest
from time import sleep
from tenacity import retry, stop_after_attempt, wait_exponential
import logging
logging.basicConfig(level=logging.INFO, format='%(asctime)-16s %(levelname)-8s
%(message)s')


class TestWeb(unittest.TestCase):
    @classmethod
    def setUpClass(cls):
        cls.tb = TsmsWeb()
        cls.ts = Tsmstest()
        # 登录
        cls.tb.login_c()
        assert cls.tb.is_login()

    @classmethod
    def tearDownClass(cls):
        cls.tb.close()


    @retry(stop=stop_after_attempt(5), wait=wait_exponential(multiplier=1,
max=10))
    def check_web(self, phone):
        logging.info("start check web")
        send_list = self.tb.reg_send()
        real_res = self.tb.get_res(phone, send_list)
```

```
        assert real_res[1] == phone
        assert real_res[3] == "success", "期待返回{}，实际返回
{}".format("success", real_res[3])
        assert real_res[4] == "1"

    def test_add_sign(self):
        # 调接口
        phone = self.ts.gen_phones(1)
        data = {"sign_id": 424, "temp_id": 180, "mobiles": phone}
        self.ts.req_post('message', data)
        assert self.ts.status_code == 200
        assert isinstance(self.ts.json["uuid"], str)
        # 查前端
        self.check_web(phone[0])

if __name__ == '__main__':
    # 执行本suite
    unittest.main(argv=['ignored', '-v'], exit=False)
```

```
2019-10-26 22:46:11,773 INFO     [获取页面token]:
ImFhZGNiMDE0MWJiYWJlOGIwNWJjYzg2NWJkMjcxNjc2NDY3ZmU1YjUi.EJXtsw.ezUVMaCacwDjZ0
mTRnBB88wzoaU
2019-10-26 22:46:11,926 INFO     [现在登录页面是]: http://127.0.0.1:5001/user/dcs
2019-10-26 22:46:11,940 INFO     登录成功
test_add_sign (__main__.TestWeb) ... 2019-10-26 22:46:11,941 INFO     [当前被调
用方法是]: gen_phones
2019-10-26 22:46:11,942 INFO     [num is]: 1
2019-10-26 22:46:11,942 INFO     [执行结果为]: ['17049642058']
2019-10-26 22:46:11,943 INFO     [当前被调用方法是]: req_post
2019-10-26 22:46:11,944 INFO     [url_type is]: message
2019-10-26 22:46:11,944 INFO     [data is]: {'sign_id': 424, 'temp_id': 180,
'mobiles': ['17049642058']}
2019-10-26 22:46:11,945 INFO     [当前请求的地址是]:
http://127.0.0.1:5001/v1/message
2019-10-26 22:46:11,946 INFO     [发送内容是]: {'sign_id': 424, 'temp_id': 180,
'mobiles': ['17049642058']} <class 'dict'>
2019-10-26 22:46:12,121 INFO     [返回码是:] 200
2019-10-26 22:46:12,122 INFO     [返回内容是]: {'uuid': '5aa6e3c4-f7ff-11e9-
92de-acde48001122'}
2019-10-26 22:46:12,124 INFO     [执行结果为]: {'uuid': '5aa6e3c4-f7ff-11e9-
92de-acde48001122'}
2019-10-26 22:46:12,125 INFO     start check web
2019-10-26 22:46:12,126 INFO     [现在要正则匹配的页面是]:
http://127.0.0.1:5001/user/dcs/history
2019-10-26 22:46:13,149 INFO     start check web
2019-10-26 22:46:13,150 INFO     [现在要正则匹配的页面是]:
http://127.0.0.1:5001/user/dcs/history
ok
```

```
----------------------------------------------------------------------
Ran 1 test in 1.405s

OK
```

```
```