今日学习目标

能够使用接口自动化测试框架封装Dubbo接口

现有问题

远程调用的 7个 dubbo接口 存在的问题：

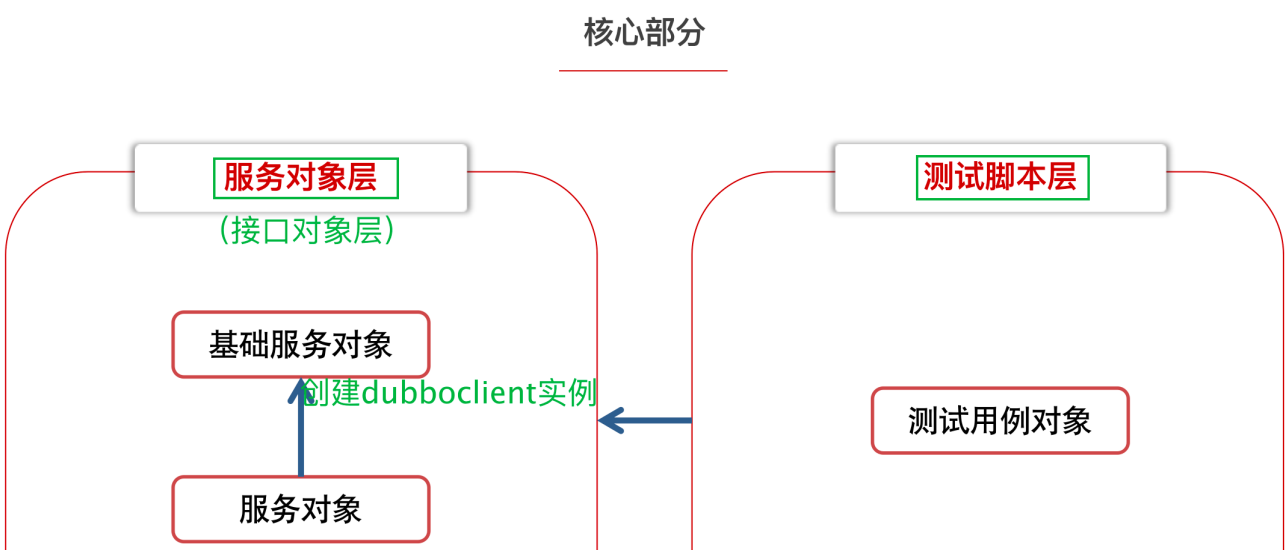
1. **代码有 大量冗余**
2. **测试接口时，除了要给 测试数据之外，还需要 指定 服务名、方法名**
3. **传参时，除了要考虑测试数据外，还要分析是否要添加 class 字段 及 对应数据。**
4. **返回的数据类型统一为 string（不具体）**

封装目标

1. **只关心：测试数据、响应结果**
2. **返回的结果 分别为 不同的 具体类型。**

Dubbo接口自动化测试框架

==核心模块==



基础服务对象封装

```python
from dubboclient import DubboClient

class BaseService(object):
    def __init__(self):
        self.dubbo_client = DubboClient("211.103.136.244", 6502)
```

==服务对象封装==

会员服务封装

```python
"""
类名：MemberService，继承于 BaseService
实例属性：
    服务名称：service_name，赋值为 'MemberService'
实例方法：
    def __init__(self):
        # 先调父类__init__()，再添加实例属性 service_name

    def find_by_telephone(self, telephone):
        # 功能：根据手机号查询会员信息
        # :param telephone: 手机号
        # :return: 1. 会员存在，返回会员信息 2. 会员不存在，返回None

    def find_member_count_by_months(self, data_list):
        # 功能：根据日期统计会员数
        # :param date_list: 日期列表，格式如：["2021.7"]
        # :return: 返回列表，列表元素为对应月份的会员数，如：[10]

    def add(self, info): 添加会员
        # 功能：添加会员
        # :param info: 会员信息的字典格式数据，参考接口文档填入字段数据，手机号需要唯一
        #              如：{"fileNumber":"D0001", "name":"李白", "phoneNumber":"13020210002"}
        # :return: 添加成功返回True， 添加失败返回False
验证结果：
        # 1. 实例化对象
        # 2. 通过实例对象调用实例方法
        # 2.1 根据手机号查询会员信息
        # 2.2 根据日期统计会员数
        # 2.3 添加会员
"""
import json
from day02.base_service import BaseService

# 将 会员服务 封装成 会员服务类
class MemberService(BaseService):
    def __init__(self):
        super().__init__()  # 调用父类 init 方法
        self.service_name = "MemberService"
```

```
    def find_by_telephone(self, tel):
        resp = self.dubbo_client.invoke(self.service_name, "findByTelephone", tel)
        if resp == "null":
            return None
        else:
            # 作用：将 string类型的 数据，还原回成 字典 或 列表 数据。
            return json.loads(resp)

    def find_member_count_by_months(self, months):
        resp = self.dubbo_client.invoke(self.service_name, "findMemberCountByMonths", months)
        # 作用：将 string类型的 数据，还原回成 字典 或 列表 数据。
        return json.loads(resp)

    def add(self, info):
        """
        :param info: 代表 用户 传入的 测试数据，没有 class 元素
        :return:
        """
        # 如果 class 已经存在，覆盖原有class值；如果不存在 class，新增一组 元素到 字典中。
        info["class"] = "com.itheima.pojo.Member"

        # 3. 调用 invoke 传 服务名、方法名、实参。得响应结果
        resp = self.dubbo_client.invoke(self.service_name, "add", info)
        if resp == "null":
            return True
        else:
            return False

if __name__ == '__main__':
    ms = MemberService()
    resp = ms.find_by_telephone("13020210001")
    print("响应结果 =", resp)
    print("type(resp) =", type(resp))

    print("=" * 66)

    months = ["2021-6"]
    ms = MemberService()
    resp = ms.find_member_count_by_months(months)
    print("响应结果 =", resp)
    print("type(resp) =", type(resp))

    print("&" * 66)

    # 准备 add 方法，所需要的数据
    info = {"id": 911, "name": "杜甫", "phoneNumber": "13048379041"}
    ms = MemberService()
    resp = ms.add(info)
    print("响应结果 =", resp)
    print("type(resp) =", type(resp))
```

预约设置服务封装

```
"""
类名：OrderSettingService，继承于 BaseService
实例属性：
    服务名称：service_name，赋值为 'OrderSettingService'
实例方法：
    def __init__(self):
        # 先调父类__init__()，再添加实例属性 service_name

    def add(self, date_list):
        # 功能：添加预约设置
        # :param date_list:
        #     1.  日期列表，如：[{"orderDate":"2021-09-20 16:45:12","number":20}]
        #     2. 日期格式为："2021-09-20 16:45:12"，必须包括时分秒
        # :return: 设置成功返回True，  设置失败返回False

    def get_order_setting_by_month(self, date):
        # 功能：按月统计预约设置信息
        # :param date: 日期，如："2021-08"
        # :return: 列表，指定月份的预约信息

    def edit_number_by_date(self, info): 根据日期修改预约设置数量
        # 功能：根据日期修改预约设置数量
        # :param info:
        #     1. 预约设置的字典格式数据，参考接口文档填入字段数据
        #     2. 如：{"orderDate":"2021-09-19 17:45:12","number":15}
        #     3. 日期格式为："2021-09-19 17:45:12"，必须包括时分秒
        #     4. 添加 "class":"com.itheima.pojo.OrderSetting"
        # :return: 修改成功返回 True，  修改失败返回 False
验证结果：
        # 1. 实例化对象
        # 2. 通过实例对象调用实例方法
        # 2.1 添加预约设置
        # 2.2 按月统计预约设置信息
        # 2.3 根据日期修改预约设置数量
"""
import json

from day02.base_service import BaseService


# 封装 预约设置服务类
class OrderSettingService(BaseService):
    def __init__(self):
        super().__init__()
        self.service_name = "OrderSettingService"

    def add(self, date_list):
        # 功能：添加预约设置
        # :param date_list:

        #     1. 日期列表，如：[{"orderDate":"2021-09-20 16:45:12","number":20}]
```

```
        #     2. 日期格式为: "2021-09-20 16:45:12", 必须包括时分秒
        # :return: 设置成功返回 True,  设置失败返回 False
        resp = self.dubbo_client.invoke(self.service_name, "add", date_list)
        if resp == "Failed":
            return False
        else:
            return True

    def get_order_setting_by_month(self, month):
        # 功能: 按月统计预约设置信息
        # :param months: 日期, 如: "2021-08"
        # :return: 列表, 指定月份的预约信息
        resp = self.dubbo_client.invoke(self.service_name, "getOrderSettingByMonth", month)
        if resp == "Failed":
            return None
        else:
            return json.loads(resp)

    def edit_number_by_date(self, date):
        # 功能: 根据日期修改预约设置数量
        # :param info:
        #     1. 预约设置的字典格式数据, 参考接口文档填入字段数据
        #     2. 如: {"orderDate":"2021-09-19 17:45:12","number":15}
        #     3. 日期格式为: "2021-09-19 17:45:12", 必须包括时分秒
        #     4. 添加 "class":"com.itheima.pojo.OrderSetting"
        # :return: 修改成功返回 True,  修改失败返回 False
        date["class"] = "com.itheima.pojo.OrderSetting"

        # 3. 调用 invoke 传 服务名、方法名、实参。得响应结果
        resp = self.dubbo_client.invoke(self.service_name, "editNumberByDate", date)
        if resp == "Failed":
            return False
        else:
            return True


if __name__ == '__main__':
    oss = OrderSettingService()

    # 准备 add 方法, 所需要的数据
    info = [{"orderDate": "2021-05-18", "number": 346}]

    resp = oss.add(info)
    print("响应结果 =", resp)
    print("type(resp) =", type(resp))

    print("============== 按月统计预约设置信息 ===========")
    oss = OrderSettingService()
    # 月份
    months = "2021.02"
    resp = oss.get_order_setting_by_month(months)
    print("响应结果 =", resp)

    print("type(resp) =", type(resp))
```

```
    print("============== 根据日期修改预约设置数量 ===========")
    # 日期 和 设置数据
    date = {"orderDate": "2021-06-15 16:99:77", "number": 120}
    oss = OrderSettingService()
    resp = oss.edit_number_by_date(date)
    print("响应结果 =", resp)
    print("type(resp) =", type(resp))
```

用户服务封装

```
"""
类名：UserService，继承于BaseService
实例属性：
    服务名称：service_name，赋值为'UserService'
实例方法：
    def __init__(self):
        # 先调父类__init__()，再添加实例属性 service_name

    def find_by_username(self, username):
        # 功能：根据用户名查询用户信息
        # :param username: 用户名
        # :return: 1. 如果用户存在，返回用户信息 2. 如果不存在，返回 None

验证结果：
        # 1. 实例化对象
        # 2. 通过实例对象调用实例方法
"""
import json

from day02.base_service import BaseService


# 封装 用户服务类
class UserService(BaseService):
    def __init__(self):
        super().__init__()

    def find_by_user_name(self, name):
        # 3. 调用 invoke 传 服务名、方法名、实参。得响应结果
        resp = self.dubbo_client.invoke("UserService", "findByUsername", name)
        if resp == "null":
            return None
        else:
            return json.loads(resp)

if __name__ == '__main__':
    # 管理员用户名

    name = "李白"
```

```
    us = UserService()
    resp = us.find_by_user_name(name)
    print("响应结果 =", resp)
    print("type(resp) =", type(resp))
```

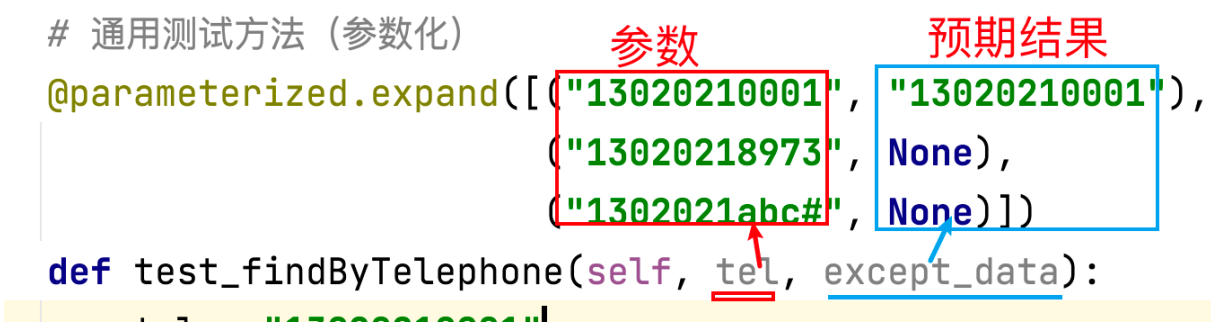
==测试用例对象封装==

```
import unittest

# 借助 unittest 框架，封装测试类，从 TestCase 继承
from day02.py02_会员服务类封装设计 import MemberService

class TestFindByTelephone(unittest.TestCase):
    ms = None

    @classmethod
    def setUpClass(cls) -> None:
        # 创建MemberService实例
        cls.ms = MemberService()

    def test01_tel_exists(self):
        tel = "13020210001"
        resp = self.ms.find_by_telephone(tel)
        print("手机号存在 =", resp)

        self.assertEqual("13020210001", resp.get("phoneNumber"))

    def test02_tel_not_exists(self):

        tel = "13020218973"
        resp = self.ms.find_by_telephone(tel)
        print("手机号不存在 =", resp)

        self.assertEqual(None, resp)

    def test03_tel_has_special_char(self):

        tel = "1302021abc#"
        resp = self.ms.find_by_telephone(tel)
        print("手机号含有字母特殊字符 =", resp)

        self.assertEqual(None, resp)
```

参数化

1. 导包 from parameterized import parameterized
2. 在 通用测试方法上一行，@parameterized.expand()

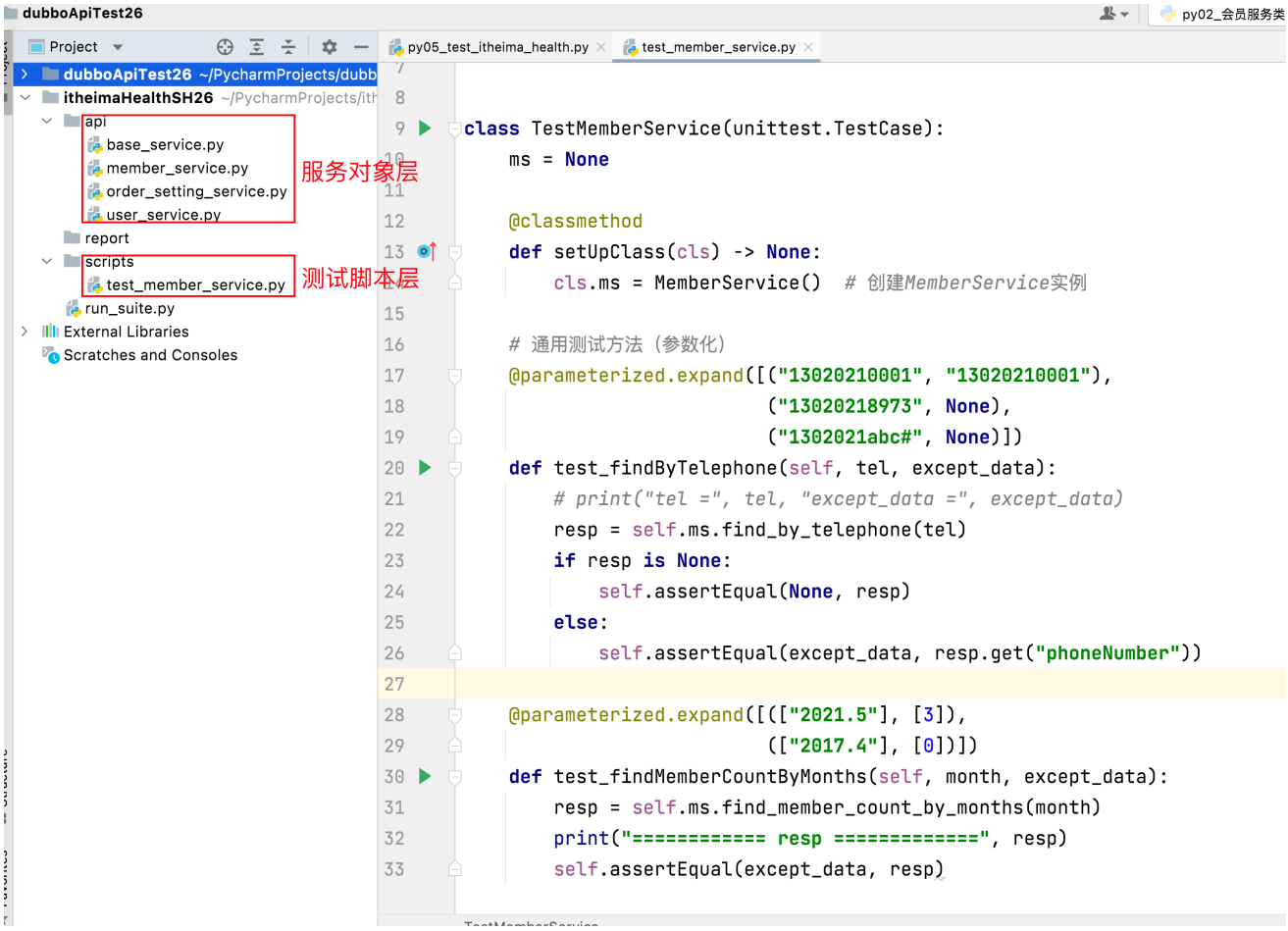
3. 给 expand() 传入 [(),(),()] 类型的数据。
4. 修改 通用测试方法，添加形参，个数、顺序与 () 数据一致。
5. 在 通用测试方法 使用形参




```
import unittest
from day02.py02_会员服务类封装设计 import MemberService
from parameterized import parameterized

# 借助 unittest 框架，封装测试类，从 TestCase 继承
class TestMemberService(unittest.TestCase):
    ms = None

    @classmethod
    def setUpClass(cls) -> None:
        cls.ms = MemberService()  # 创建MemberService实例

    # 通用测试方法（参数化）
    @parameterized.expand([("13020210001", "13020210001"),
                           ("13020218973", None),
                           ("1302021abc#", None)])
    def test_findByTelephone(self, tel, except_data):
        # print("tel =", tel, "except_data =", except_data)
        resp = self.ms.find_by_telephone(tel)
        if resp is None:
            self.assertEqual(None, resp)
        else:
            self.assertEqual(except_data, resp.get("phoneNumber"))

    @parameterized.expand([(["2021.5"], [3]),
                           (["2017.4"], [0])])
    def test_findMemberCountByMonths(self, month, except_data):
        resp = self.ms.find_member_count_by_months(month)
        print("============ resp =============", resp)
        self.assertEqual(except_data, resp)

    # def test01_tel_exists(self):
    #     tel = "13020210001"
    #     resp = self.ms.find_by_telephone(tel)
    #     print("手机号存在 =", resp)
    #
    #     self.assertEqual("13020210001", resp.get("phoneNumber"))
    #
    # def test02_tel_not_exists(self):
```

```
    #     tel = "13020218973"
    #     resp = self.ms.find_by_telephone(tel)
    #     print("手机号不存在 =", resp)
    #
    #     self.assertEqual(None, resp)
    #
    # def test03_tel_has_special_char(self):
    #     tel = "1302021abc#"
    #     resp = self.ms.find_by_telephone(tel)
    #     print("手机号含有字母特殊字符 =", resp)
    #
    #     self.assertEqual(None, resp)
```

接口自动化框架封装



测试报告

```
# 导包
import unittest
from htmltestreport import HTMLTestReport
```

```
# 创建 suite 实例
from scripts.test_member_service import TestMemberService

suite = unittest.TestSuite()

# 添加测试用例
suite.addTest(unittest.makeSuite(TestMemberService))

# 创建 HTMLTestReport 类对象
runner = HTMLTestReport("./report/传智健康测试报告.html", description="描述", title="标题")

# 调用 run() 传入 suite
runner.run(suite)
```

作业

1. 完成每日反馈!!!
2. 复习本天课程内容，完成《接口测试-第12天-作业.md》中习题。