

AW自动化框架指导书

自动化测试介绍

自动化测试的优点：

- **提高效率：**自动化测试可以大大减少手动测试的时间和努力，特别是在回归测试和大规模测试中。一旦测试脚本被编写和验证，它们可以在任何时间被重复执行，无需额外的成本。
- **提高质量：**自动化测试可以减少由于人为错误导致的测试遗漏和错误。它可以确保每次都按照相同的步骤和顺序执行测试，从而提高测试的准确性和一致性。
- **提高覆盖率：**自动化测试可以更容易地覆盖更多的测试场景，包括边界条件和异常情况，这在手动测试中可能很难或很耗时去做。
- **快速反馈：**自动化测试可以快速提供反馈，帮助开发人员及时发现和修复问题。这对于持续集成/持续部署（CI/CD）流程尤其重要。
- **降低成本：**虽然自动化测试的初期投资可能较大（例如，购买测试工具，编写和维护测试脚本等），但长期来看，它可以降低软件开发的总体成本，特别是在大型和长期的项目中。
- **提高可靠性：**自动化测试可以在每次代码更改后执行，确保代码的更改没有引入新的错误。
- **支持敏捷和DevOps实践：**自动化测试是敏捷开发和DevOps实践的关键组成部分，它支持频繁和持续的软件交付。
- **可重复性：**自动化测试可以无限次地重复执行，这对于验证修复的缺陷或执行回归测试非常有用。
- **并行执行：**自动化测试可以在多个设备或平台上并行执行，这可以大大缩短测试时间。
- **性能测试：**自动化测试可以模拟大量用户并发访问，用于性能、负载和压力测试。
- **可编程：**自动化测试允许你使用编程语言来编写复杂的测试逻辑，这在手动测试中是无法实现的。
- **持续监控：**自动化测试可以作为持续监控的一部分，定期检查系统的功能和性能。
- **提高团队士气：**通过减少重复和繁琐的手动测试工作，自动化测试可以提高测试团队的士气，并让他们有更多的时间专注于更有挑战性和价值的任务。
- **文档化测试用例：**自动化测试脚本本身就是一种形式的文档，它详细描述了测试的步骤和预期结果。
- **支持快速迭代：**在敏捷开发和持续交付的环境中，自动化测试可以快速验证新的特性和改进，支持快速迭代。

自动化测试的缺点：

- **高昂的初始成本：**自动化测试工具的购买成本可能很高，而且编写和维护自动化测试脚本也需要投入大量的时间和资源。因此，自动化测试的初始成本可能会比手动测试高。

- **技术要求高：**编写和维护自动化测试脚本需要一定的编程知识和技能。这可能会增加团队的培训成本，或者需要聘请具有这些技能的新员工。
- **不适合所有测试：**并非所有的测试都适合自动化。例如，探索性测试、用户体验测试、可用性测试等通常需要人工进行，因为它们需要人的直觉、创造力和主观判断。
- **维护成本高：**随着软件的变化和演进，自动化测试脚本可能需要频繁地更新和维护。如果没有良好的测试管理和维护策略，这可能会导致维护成本的增加。
- **可能产生误报：**自动化测试可能会因为各种原因（如网络延迟、测试环境问题等）产生误报，这可能会浪费团队的时间去调查和解决这些非问题。
- **依赖于UI的测试脚本脆弱：**如果自动化测试脚本过于依赖于用户界面，那么任何UI的改变都可能导致测试脚本失败。这就需要测试人员花费大量的时间去更新和维护这些脚本。

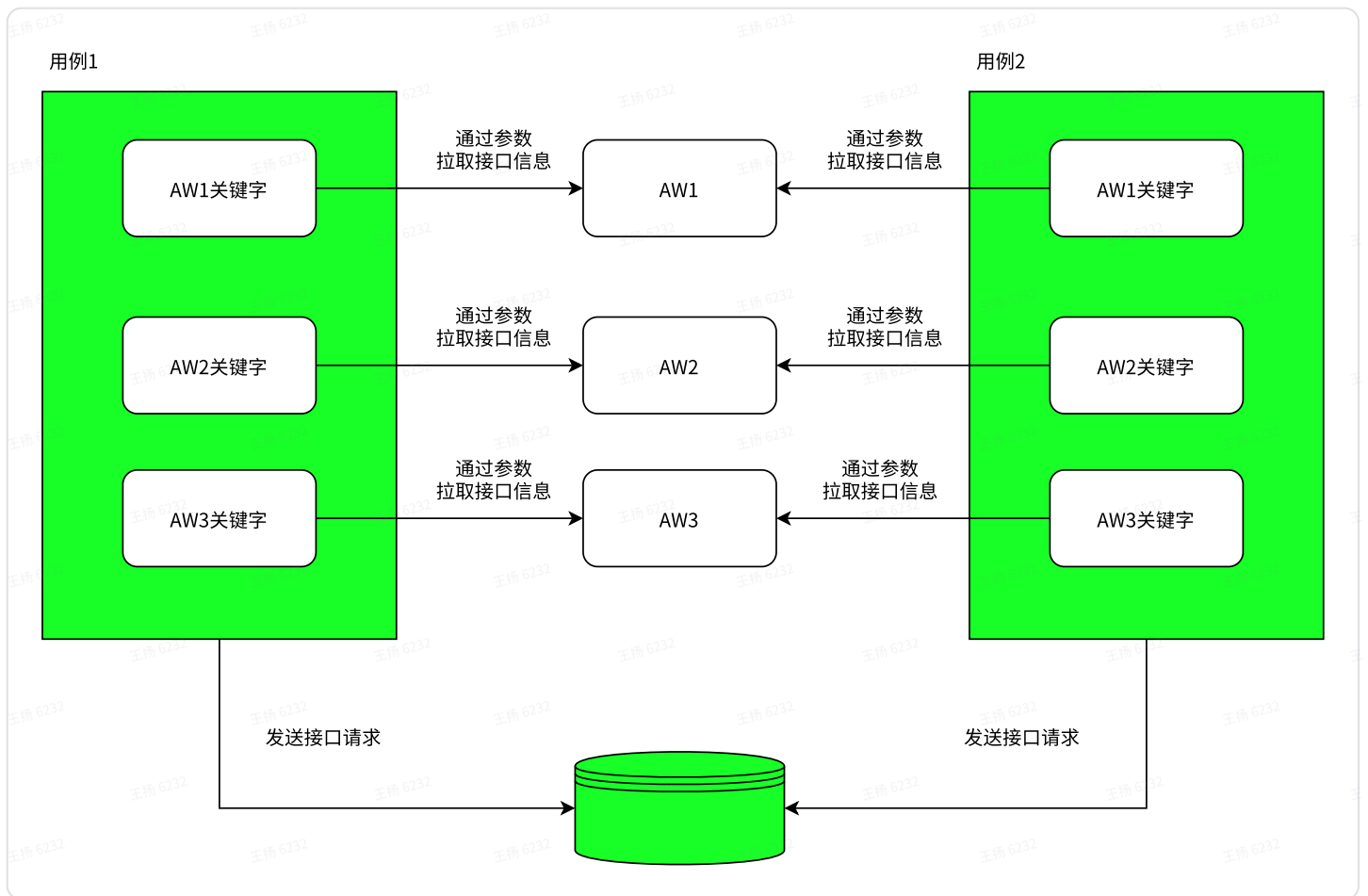
因此，在决定是否以及如何进行自动化测试时，需要权衡其优点和缺点，并根据项目的具体需求和条件做出决策。

框架介绍

基本原理及快速入门

概述

1. 接口自动化用例的实现思想：是利用了模板和编排的思想实现，包括：用例是用AW关键字进行编排的（是YAML模板），AW是固定的YAML的模板
2. 每一个AW的YAML模板，封装了单接口所有的属性，包括：接口请求方式、URL地址、所属模块、前置、后置、断言、参数提取等等，支持外部传递不同的参数，返回不同的内容出去
3. 用例包含了使用AW关键字和参数来拉取AW yaml文件里面的信息，组装为用例发送请求，进行结果校验，完成自动化测试



用例

用例YAML样例

下面黄色部分是调用AW

用例yaml的命名规范是: test_xxx.yaml

```
1 # -*- coding: utf-8 -*-
2 - case: 发票中台-查询可开票金额-单订单
3   epic: 业财
4   feature: 发票中台
5   story: 发票中台
6   severity: P0
7   tag: lxt
8   variables:
9     payeeCode: orderlxt001 # 业务单号
10  steps:
11    - name: 发票中台-查询可开票金额
12      module: invoice-center
13      AWFunc: 查询可开票金额
14      aw_params: {'bizCodes': ["$payeeCode"]}
15  teardown:
16    sqls:
17
```

```

18 - case: 发票中台-查询可开票金额-多订单
19 epic: 业财
20 feature: 发票中台
21 story: 发票中台
22 severity: P0
23 tag: lxt
24 variables:
25     payeeCode1: orderlxt001 # 业务单号
26     payeeCode2: orderlxt002 # 业务单号
27 steps:
28     - name: 发票中台-查询可开票金额
29       module: invoice-center
30       AWFunc: 查询可开票金额
31       aw_params: {'bizCodes': ['$payeeCode1', '$payeeCode2']}
32 teardown:
33 sqls:
34
35 - case: 发票中台-查询可开票金额-申请列表
36 epic: 业财
37 feature: 发票中台
38 story: 发票中台
39 severity: P0
40 variables:
41 steps:
42     - name: 发票中台-查询可开票金额-申请列表
43       module: invoice-center
44       AWFunc: 自营开票申请列表
45       aw_params: {'sellerSubject': '开工智合（江苏）数字技术有限公司'}
46 teardown:
47 sqls:

```

python用例编写

python用例的脚本还是需要写到case目录下面

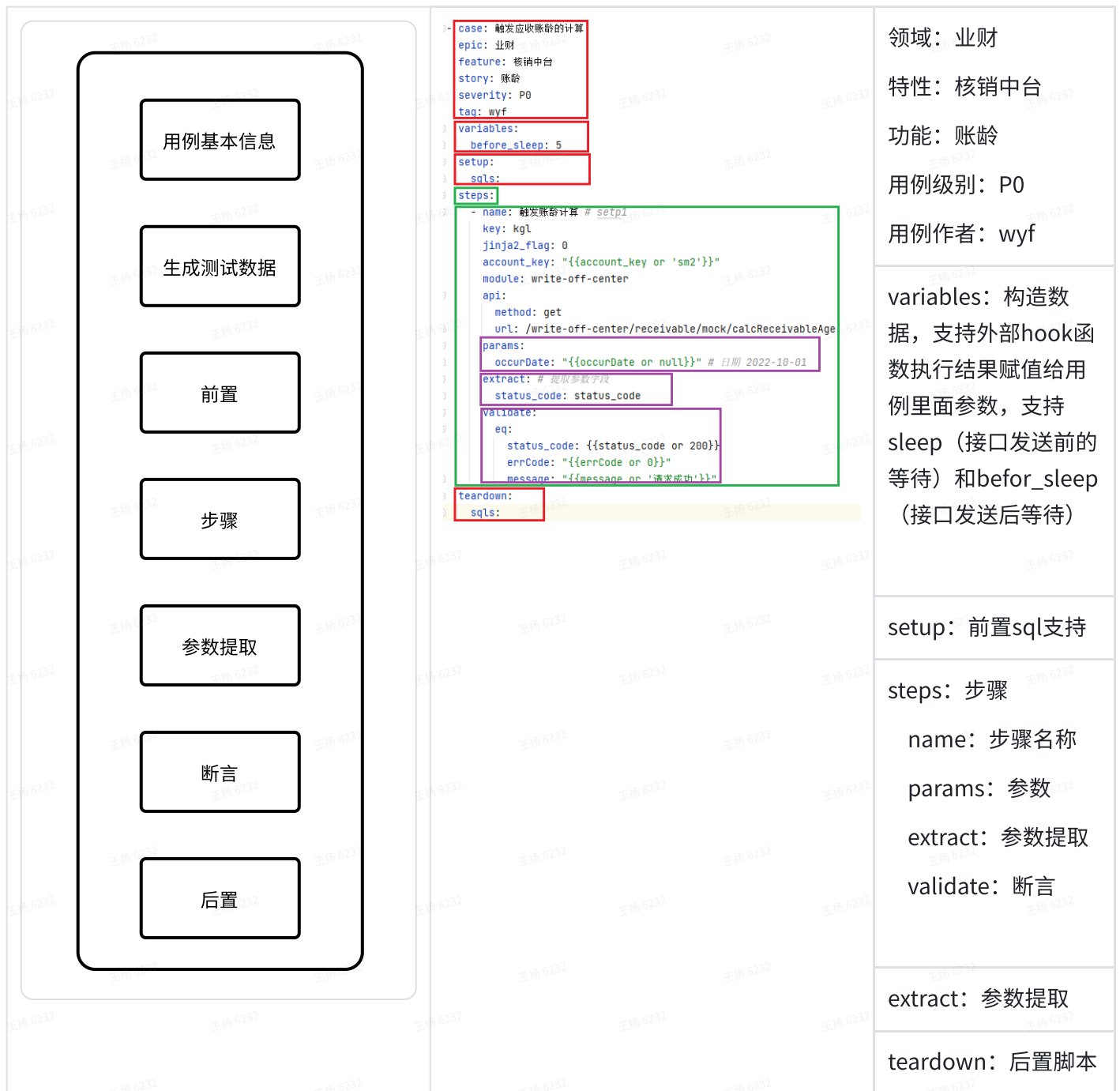
```

1 import pytest
2
3 from utils.base.myMetaClass import MyMetaClass
4
5 class Test_invoice_center(metaclass=MyMetaClass):
6
7     case_yml_list = [
8         # 发票中台
9         "/data/kgf/invoicecenter/test_佣金发票申请.yaml",
10        "/data/kgf/invoicecenter/test_居间佣金退票申请.yaml",
11        "/data/kgf/invoicecenter/test_查询可开票金额.yaml",

```

AW

AW结构图



AW的YAML样例

下面黄色部分是AW的主体，步骤部分

AW的命名规范是aw_xxxx.yaml

```

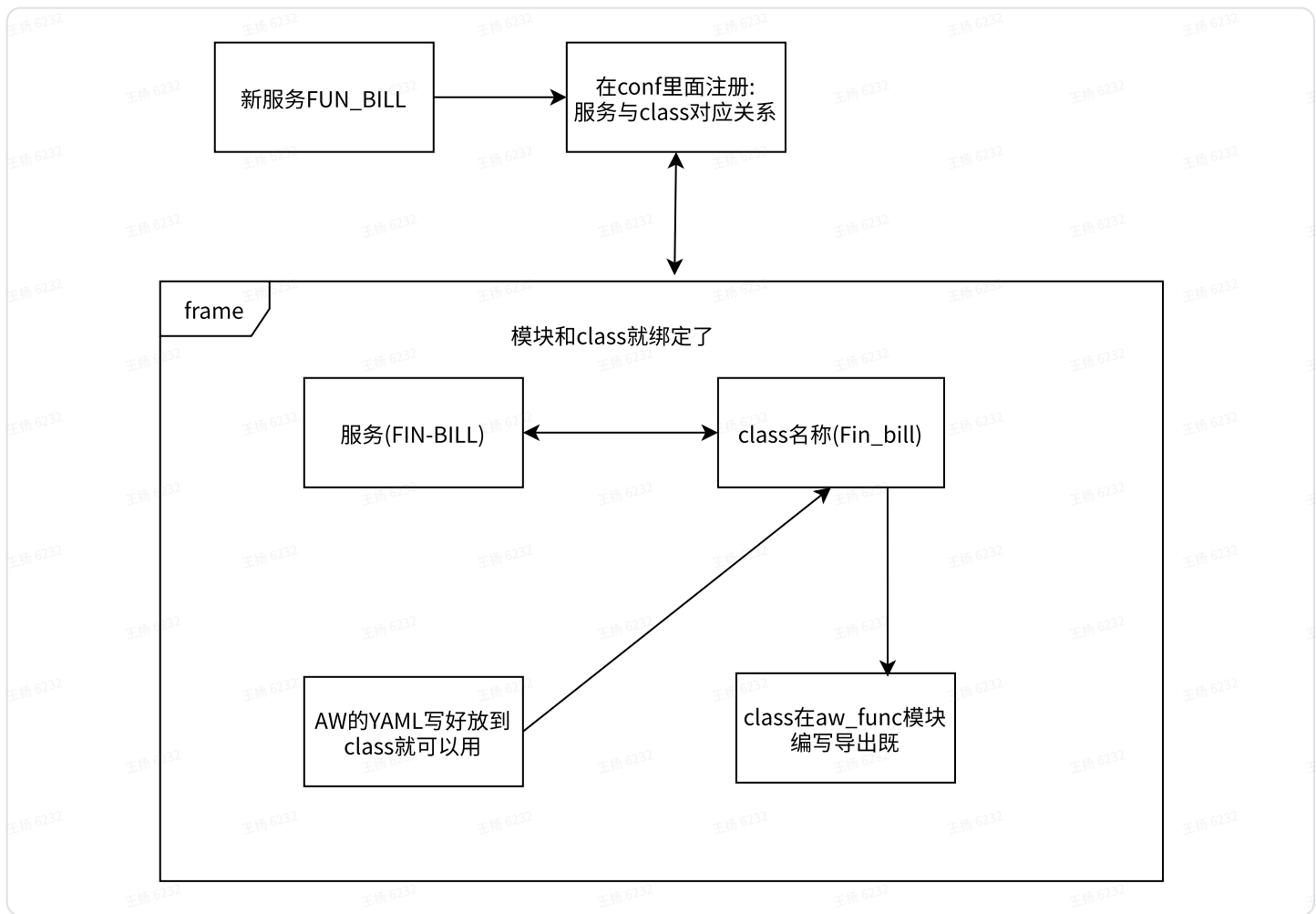
2 epic: 业财
3 feature: 发票中台
4 story: 发票申请
5 severity: P0
6 tag: lxt
7 variables: # 测试数据生成
8 steps:
9   - name: 查询可开票金额 #
10     key: kgl
11     jinja2_flag: 0
12     account_key: lxt
13     module: invoice-center
14     extract: # 数据提取, 这里可以提取接口返回的applyCode
15     api:
16       method: get
17       url: /invoice-center/api/invoice/getCanInvoiceAmount
18       params:
19         bizCodes: {{ bizCodes|myDefault(0) }}
20       validate:
21         eq:
22           status_code: {{ status_code|myDefault(200) }}
23           errCode: {{ errCode|myDefault(200) }}
24           message: {{ message|myDefault(200) }}
25     setup:
26       sqls:
27     teardown:
28       sqls:

```

AW注册分两步:

- 服务注册: 每个服务只需要注册一次
- AW注册: 每个AW YAML写完之后, 注册后就可以使用

图解



服务注册

只需在conf.yml文件里面注册即可，如下图，只需要写模块名称加冒号

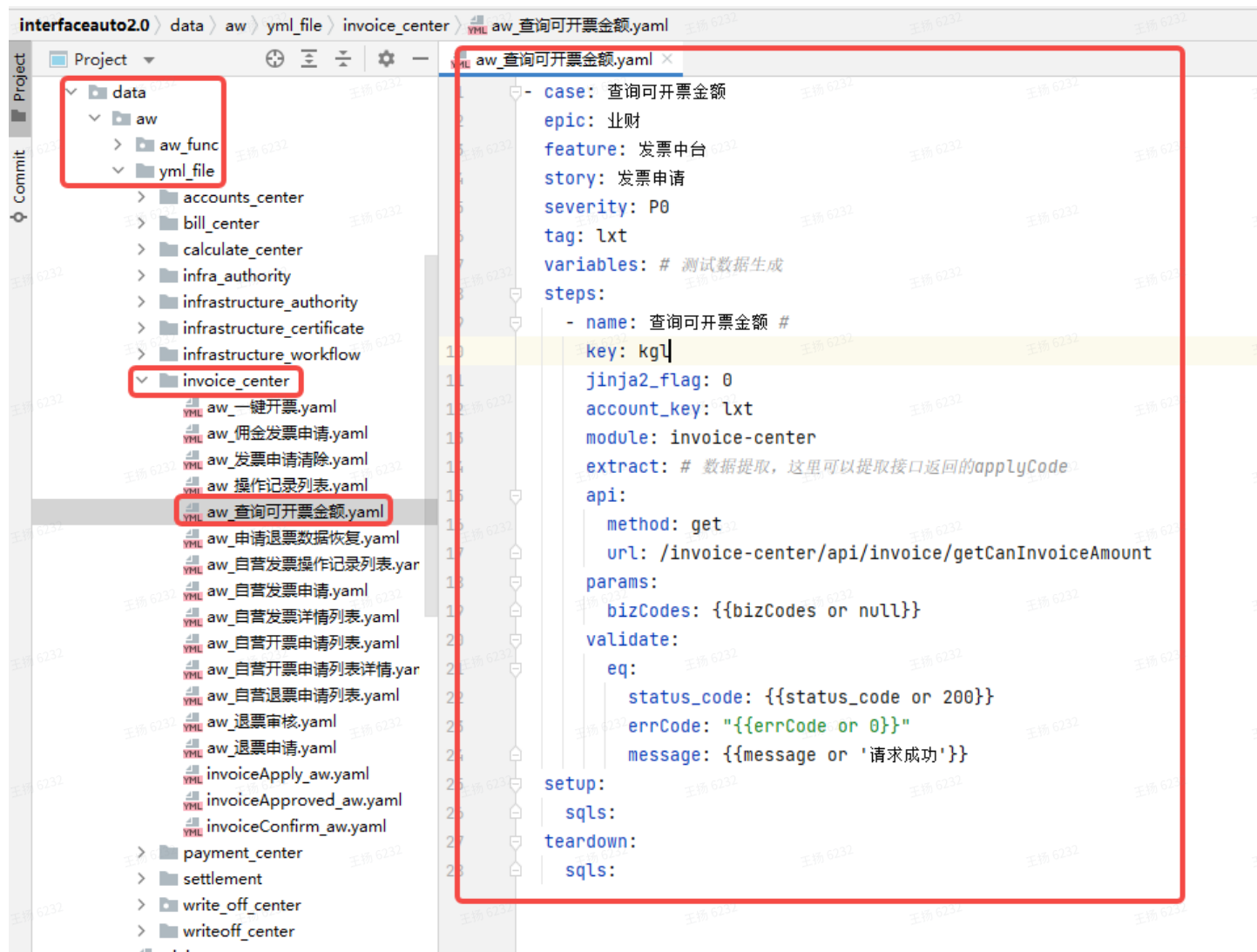
```

BASE:
test:
  url: "https://api-si
# log等级 warning error
log_level: "info"
# 扩展名
log_extension: ".log"
# 模块名与 AW 里面的类属性
module:
  finance-settlement:
  infrastructure-authority: Infrastructure_authority #效能-权限、认证
  infrastructure-workflow: Infrastructure_workflow #效能-流程引擎
  payment-center: #业财支付中心
  writeoff-center: Writeoff_center
  infrastructure-certificate: Infrastructure_certificate #效能-注册相关
  infra-authority: Infra_authority #效能-认证权限相关
  invoice-center: Invoice_center
  calculate-center: Calculate_center #计费中台
  bill-center: Bill_center #账单中台
  write-off-center: #留空会自动生成: Write_off_center, 替换下划线, 首写字母
  accounts-center: Accounts_center #账款中台
  account-info: Account_info #效能-用户
  ocr-recognize: Ocr_recognize #OCR自定义模版识别
  infrastructure-integration: Infrastructure_integration #查询省市区域编码
  merchant-cts: Merchant_cts #商户管理
  demo-pass: Demo_pass
  modify-password: Modify_password #旧密码修改密码
  send-verifycode: Send_verifycode #忘记密码发送验证码
  user-center: #效能-用户中心
  infrastructure-metadata: Infrastructure_metadata #元数据后台管理
  infrastructure-sign: Infrastructure_sign #电子签约
  finance-billing:
  finance-custody: #平台-二清系统
  finance-invoice: #发票服务
  finance-gateway: #财务网关
  infrastructure-license:
  infrastructure-message:
  infrastructure-impexp:
  infrastructure-console:
  infrastructure-admin:
  finance-evidence:
  risk-controlling: #金融风控
  infrastructure-licen

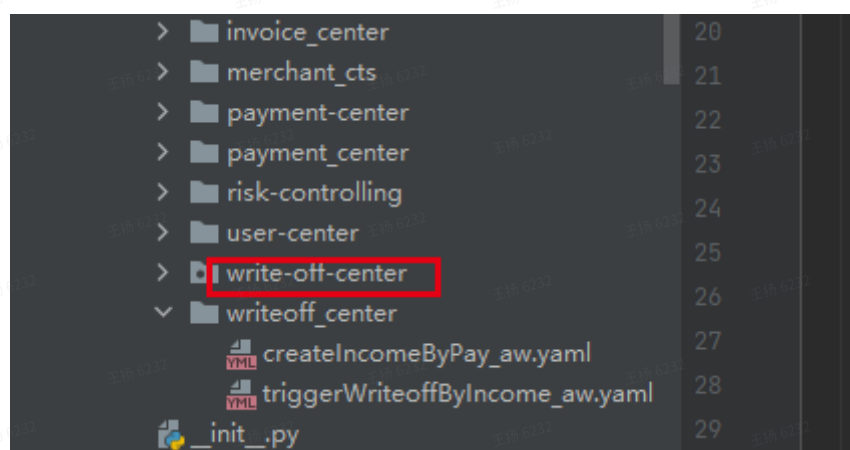
```

AW注册。

- 创建服务的aw存放目录, 建议在下图的目录里面以服务名称中线改为_的方式命名目录



aw文件所在的目录名称必须和模块名称一致



前提：aw文件名字是aw_xxxx.yaml，aw的调用名字就是xxx

HAW

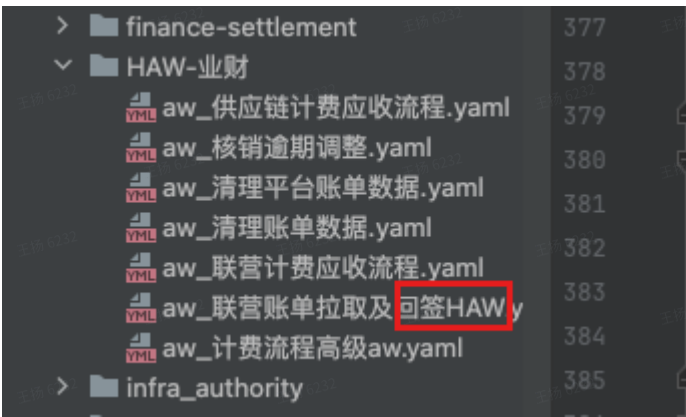
概述

高阶AW则是指更加抽象和复杂的关键字，它们可能组合了多个普通AW来完成一个更复杂的测试场景或流程。例如，“登录并验证用户信息”可以是一个高阶AW，它包含了登录操作和验证用户信息的多个普通AW。高阶AW的使用可以使测试用例更具可读性、模块化和复用性。

因此，高阶AW和普通AW的区别在于抽象级别和复杂度。普通AW涵盖了基本的测试步骤，而高阶AW则提供了更高级别的操作和组合。在编写关键字驱动接口自动化测试时，合理使用普通AW和高阶AW可以帮助提高测试用例的可维护性和可扩展性。

建议：高阶aw最好是从低阶aw里面思考复用性较高的，需要抽象的步骤，建立高阶aw，可以大幅提高场景用例的实现效率

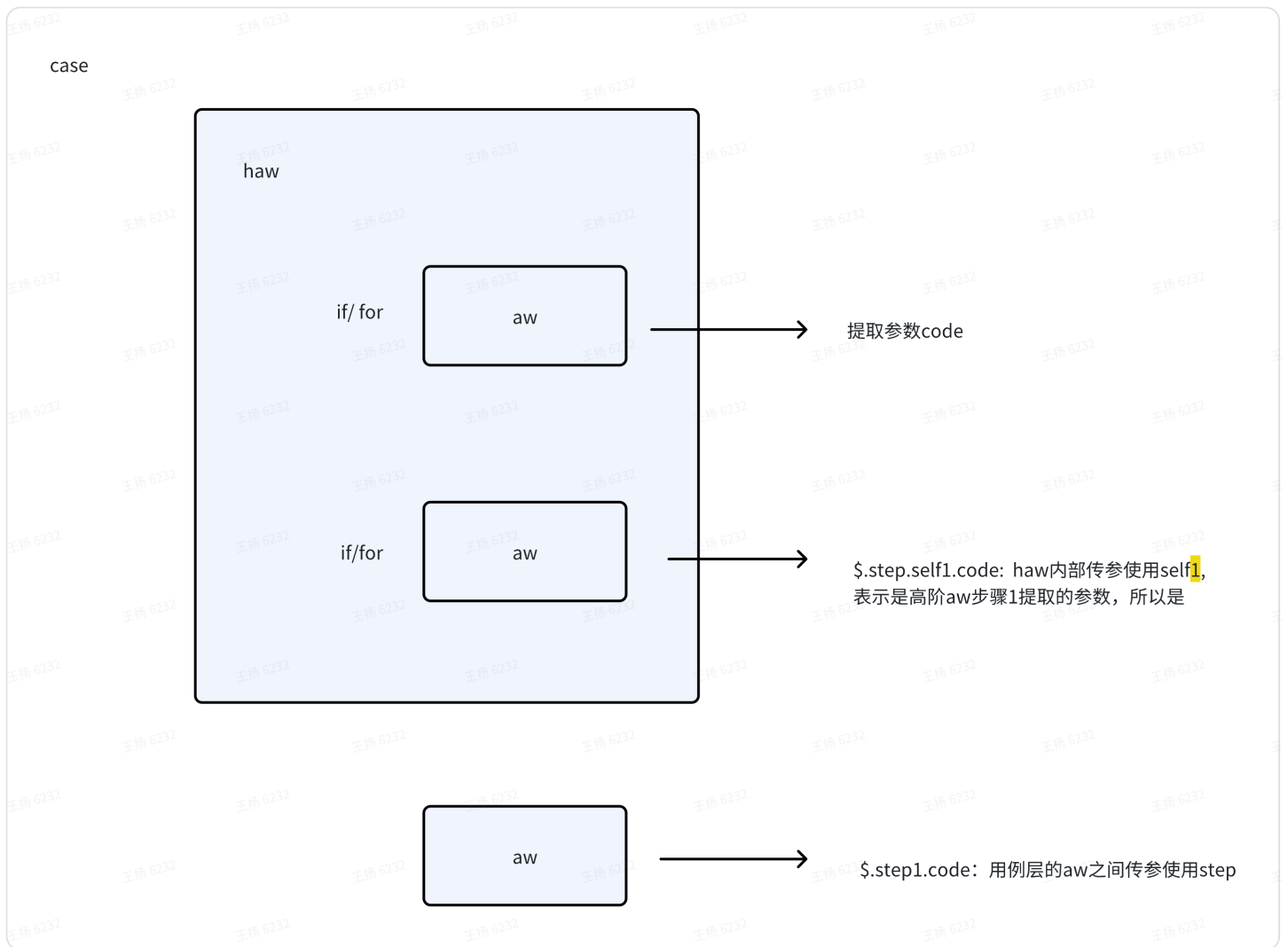
HAW的命名建议在后面增加HAW标识



> finance-settlement	377
> HAW-业财	378
aw_供应链计费应收流程.yaml	379
aw_核销逾期调整.yaml	380
aw_清理平台账单数据.yaml	381
aw_清理账单数据.yaml	382
aw_联营计费应收流程.yaml	383
aw_联营账单拉取及回签HAW.y	384
aw_计费流程高级aw.yaml	385
> infra_authority	

高阶aw的图解

我们的高阶aw比传统的高阶aw更加具有灵活性，主要得益于我们高阶aw支持if和for循环，所以可以根据传递的参数动态的生成高阶aw的行为



从上图可以看出, haw支持内部步骤之间传递参数, 关键字是self, 而haw在用例层是一个步骤, 和普通的aw是一样的, 所以和之前的参数传递是一样的, 使用step1来获取

下面是一段对话: 对haw的解释可能更加接地气

在用例层, haw是一个步骤, 只会是一个步骤

haw里面的循环只是增加了haw步骤里面的子步骤, 不是用例步骤

haw可以理解为一个场景, 或者多个场景, 自己理解下

宋辉寰(大豆)

一个用例可以包含多个haw, 或者多个普通aw

haw在用例层就是一个aw

你也可以把多个不相关的接口 封装到一个haw里面, 场景自己理解 (已编辑)

高阶aw与普通aw的对比

普通aw的实现方式

和之前的实现是一样的

```
1 # 作废账单
2 - case: test_背靠背
3   epic: 账单中台
4   feature: 联营
5   story: 背靠背
6   severity: P0
7   tag: wyf
8   ddt:
9   - test_env:
10       case_name:
11         open: 0 # 1 开, 0 关
12       # 公共字段
13       tenantId: 10001 # 租户ID 联营 为10001
14       account_key: sm1
15       accountId: 1523583931985948805 # 审批流被转交人的accountId
16
17       # 对客逻辑合同
18       # LYK-2023-599553-0001 1007900000015879255
19
20       # 对商
21       # 上海商策信息科技有限公司 LYS-2023-003517-0008 子订单号
22       1001100000046100503 逻辑合同编号 1007900000015879168
23
24       # 众能联合数字技术有限公司 LYS-2023-032603-0282 子订单号
25       1001100000046100618 逻辑合同编号 1007900000015879159
26
27       CustomerContractBizCode: 1007900000015879255 # 对客逻辑合同编号 (替换3)
28       js_orderCode: 'LYK-2023-599553-0001'
29
30       MerchantContractBizCode1: 1007900000015879159 # 对商逻辑合同编号 (替换5)
31       MerchantContractBizCode2: 1007900000015879168 # 对商逻辑合同编号 (替换5)
32
33   steps:
34     # 清除 供应链账单中台 账单数据
35     - name: 清除账单中台账单数据 # setp38
36       module: bill-center
37       AWFunc: 清除账单中台账单数据
38       aw_flag: 1
39       aw_params: { 'orderCode': '$CustomerContractBizCode',
```

```

39         'js_orderCode': '$js_orderCode',
40         'account_key': '$account_key'
41     }
42
43     # 清除 供应链账单中台 账单数据
44     - name: 清除账单中台账单数据 # setp38
45       module: bill-center
46       AWFunc: 清除账单中台账单数据
47       aw_flag: 1
48       aw_params: { 'orderCode': '$MerchantContractBizCode1',
49                   'account_key': '$account_key',
50                   "merchant_name": "众能联合数字技术有限公司"
51               }
52
53     # 清除 供应链账单中台 账单数据
54     - name: 清除账单中台账单数据 # setp38
55       module: bill-center
56       AWFunc: 清除账单中台账单数据
57       aw_flag: 1
58       aw_params: { 'orderCode': '$MerchantContractBizCode2',
59                   'account_key': '$account_key',
60                   "merchant_name": "上海商策信息科技有限公司"
61               }

```

高阶aw实现方式

用例层的步骤减少了，只有一个步骤，在高阶aw里面进行了实现

用例层

```

1  # 作废账单1
2  - case: test_背靠背
3    epic: 账单中台
4    feature: 联营
5    story: 背靠背
6    severity: P0
7    tag: wyf
8    ddt:
9      - test_env:
10         case_name:
11         open: 0 # 1 开, 0 关
12
13         Haw_bill_code: {
14             CustomerContractBizCode: {
15                 coed: "1007900000015879255", # dddd
16                 js_orderCode: "LYK-2023-599553-0001",

```

```

17     account_key: "sm1"
18 },
19 MerchantContractBizCodes: [{
20     code: 10079000000015879159,
21     merchant_name: "众能联合数字技术有限公司",
22     access_token: "sm1"
23 }, {
24     code: 10079000000015879168,
25     merchant_name: "上海商策信息科技有限公司",
26     access_token: "sm1"
27 }]
28 }
29 steps:
30     # 清除 供应链账单中台 账单数据
31     - name: 清除账单中台账单数据 # setp38
32       module: HAW-业财
33       AWFunc: 清理平台账单数据
34       aw_flag: 1
35       aw_params: {
36         Haw_bill_code: $Haw_bill_code
37     }

```

HAW具体实现

通过传递的参数，动态生成需要的步骤

```

1 # 作废账单
2 - case: 作废平台账单
3   epic: 账单中台
4   feature: 联营
5   story: 作废平台账单
6   severity: P0
7   tag: wyf
8   variables:
9     # 参数参考
10    Haw_bill_code: {
11      CustomerContractBizCode: { # 对客商户
12        coed: "10079000000016034449", # 对客逻辑合同id
13        js_orderCode: "LYK-2023-599553-0004", # 对客逻辑合同code
14        account_key: "sm1" # 登录的账号
15      },
16      MerchantContractBizCodes: [ { # 对商商户
17        code: 10079000000016034349, # 对商逻辑合同id
18        merchant_name: "众能联合数字技术有限公司", # 对商商户名称
19        access_token: "sm1" # 登录的账号
20      }, {

```

```

21 #         code: 1007900000016034357,      # 对商逻辑合同id
22 #         merchant_name: "上海商策信息科技有限公司",    # 对商商户名称
23 #         access_token: "sm1"    # 登录的账号
24 #     } ]
25 # }
26
27 steps:
28
29 # 清除 供应链账单中台 账单数据
30 {% if Haw_bill_code.CustomerContractBizCode %}
31 - name: 清除账单中台账单数据 # setp38
32   module: bill-center
33   AWFunc: 清除账单中台账单数据
34   aw_flag: 1
35   aw_params: { 'orderCode': {{ Haw_bill_code.CustomerContractBizCode.coed
36 },
37               'js_orderCode': {{
38 Haw_bill_code.CustomerContractBizCode.js_orderCode }},
39               'account_key': {{
40 Haw_bill_code.CustomerContractBizCode.account_key }}
41           }
42 {% endif %}
43
44 # 清除 供应链账单中台 账单数据
45 {% if Haw_bill_code.MerchantContractBizCodes %}
46 {% for Merchant in Haw_bill_code.MerchantContractBizCodes %}
47 - name: 清除账单中台账单数据 # setp38
48   module: bill-center
49   AWFunc: 清除账单中台账单数据
50   aw_flag: 1
51   aw_params: { 'orderCode': {{ Merchant.code }},
52               'account_key': {{ Merchant.account_key }},
53               "merchant_name": {{ Merchant.merchant_name }} # 删除背靠背结
54 算单信息
55           }
56 {% endfor %}
57 {% endif %}

```

最佳实践建议

- 1、建议在HAW里面增加参数的样例，并备注好每个参数，包括是否必须，如果可选就代码可以使用默认值，对于复杂的业务会简化传参，对于其他领域需要快速使用你的aw很有作用：比如可以写一个最简传参样例（下游的人使用），和一个最全传参样例（自己本领域测试使用）
- 2、HAW开始可以写一个正常的，对于一些比较偏的场景如果编写付出代价比较大的，可以后续有时间在优化。

下面的案例都可以极大的减少用例的篇幅和复杂度，都可以从几百行到上千行，缩短到100行以内。
最终极大的简化的用例的编写难度和减少重复工作量。

计费极简案例：

主要是给需要用到计费数据的其他领域，快速生成计费数据使用，降低跨领域自动化学习成本，提高测试效率

下面是联营一个客户对应两个商户接单（2个主订单的极简案例）

```
1 # -*- coding: utf-8 -*-
2 # 正常场景
3 - case: test_背靠背
4   epic: 账单中台
5   feature: 联营
6   story: 背靠背
7   severity: P0
8   tag: wyf
9   ddt:
10 - test_env:
11     case_name: 极简计费
12     open: 1 # 1 开, 0 关
13
14 steps:
15 - name: 联营计费应收流程 # setp1
16   module: HAW-业财
17   AFunc: 联营计费应收流程
18   aw_flag: 1
19   aw_params: { hawCalculate: {
20     public: { # 公共参数
21       account_key: "sm1", # 登录的账号
22     },
23     calculateDatas: [
24       # 1
25       { # 计费数据
26         contractBizCode: "1007900000015879255", # 逻辑合同id
27         before_sleep: 8, # 清除D0应收脏数据的前置等待,第一个合同行需入参
28         # pushSleep: 5, # 推应收接口的等待时间,最后一个合同行需入参
29         bizRowCodeDatas: [
30           { # 订单行合同行数据 -对客
31             bizCode: "1001100000046100503", # 订单编号
32             bizRowCode: "1001100000046100505", # 订单行编号
33             contractBizRowCode: "1007900000015879258", # 逻辑合同行编号# 服务项
34             # 目编码 列表
35             userIdentity: "CUSTOMER", # 客商标识 CUSTOMER - 客 MERCHANT - 商
```


码

项目编码 列表

商

目编码

目编码 列表

服务项目编码

需入参

目编码 列表

```
serviceItemCodeDatas: [ { # 服务项目数据
  serviceItemCode: "1735272976364916812-20231214-001", # 服务项目编
}
},
{ # 订单行合同行数据 -对客
  bizCode: "10011000000046100618", # 订单编号
  bizRowCode: "10011000000046100620", # 订单行编号
  contractBizRowCode: "10079000000015879258", # 逻辑合同行编号# 服务
  userIdentity: "CUSTOMER", # 客商标识 CUSTOMER - 客 MERCHANT -
  serviceItemCodeDatas: [ { # 服务项目数据
    serviceItemCode: "1735274794906079291-20231214-001", # 服务项
  } ]
},
]
},
{ # 订单行合同行数据 -对商
  contractBizCode: "10079000000015879168", # 逻辑合同id
  before_sleep: 8, # 清除D0应收脏数据的前置等待,第一个合同行需入参
  pushSleep: 5, # 推应收接口的等待时间 最后一个合同行需入参
  bizRowCodeDatas: [ { # 订单行合同行数据
    bizCode: "10011000000046100503", # 订单编号
    bizRowCode: "10011000000046100505", # 订单行编号
    contractBizRowCode: "10079000000015879169", # 逻辑合同行编号# 服务项
  } ]
  userIdentity: "MERCHANT", # 客商标识 CUSTOMER - 客 MERCHANT - 商
  serviceItemCodeDatas: [ { # 服务项目数据
    serviceItemCode: "MERCHANT_1735272976364916812-20231214-001",
    calculateSleep: 5, # 跑完计费接口的等待时间 最后一个服务项目需入参
  } ]
} ]
},
# 2
{ # 订单行合同行数据 -对商
  contractBizCode: "10079000000015879159", # 逻辑合同id
  before_sleep: 8, # 清除D0应收脏数据的前置等待,第一个合同行
  pushSleep: 5, # 推应收接口的等待时间 最后一个合同行需入参
  bizRowCodeDatas: [ { # 订单行合同行数据
    bizCode: "10011000000046100618", # 订单编号
    bizRowCode: "10011000000046100620", # 订单行编号
    contractBizRowCode: "10079000000015879160", # 逻辑合同行编号# 服务项
```

```

74     userIdentity: "MERCHANT", # 客商标识 CUSTOMER - 客 MERCHANT - 商
75     serviceItemCodeDatas: [ { # 服务项目数据
76         serviceItemCode: "MERCHANT_1735274794906079291-20231214-001",
# 服务项目编码
77         calculateSleep: 5, # 跑完计费接口的等待时间 最后一个服务项目需入参
78     } ]
79 } ]
80 } ]
81 } }

```

计费复杂传参：

主要是满足计费专业测试使用

```

1 # 正常场景
2 - case: test_2002联营计费全流程HAW_干租模版_sm
3 epic: 账单中台
4 feature: 联营
5 story: 周期账单出账相关
6 severity: P0
7 tag: sm
8 ddt:
9 - test_env:
10     case_name: test_2002联营计费全流程HAW_干租模版_sm
11     open: 1 # 1 开, 0 关
12
13     serviceItemCodeEnterEventTime: ${getCalculateTime(-10,'top')} # 进场时间
14     serviceItemCodeEnterCreatedAt: ${getCalculateTime(-10,'created')} # 进场创
建时间
15     serviceItemCodeExitEventTime: ${getCalculateTime(0,'mid')} # 进场时间
16     serviceItemCodeExitCreatedAt: ${getCalculateTime(0,'created')} # 进场创建时
间
17
18
19     serviceItemCodePauseEventTime: ${getCalculateTime(-8,'top')} # 暂停时间
20     serviceItemCodePauseCreatedAt: ${getCalculateTime(-8,'created')} # 暂停创
建时间
21     serviceItemCodeRecoveryEventTime: ${getCalculateTime(-5,'top')} # 恢复时间
22     serviceItemCodeRecoveryCreatedAt: ${getCalculateTime(-5,'created')} # 恢复
创建时间
23
24     MachineTeamOccurDate1: ${getSystemDay(-10)} # 台班单业务日
25     MachineTeamSubmitTime1: ${getCalculateTime(-10,'created')} # 台班单提交时
间
26
27     serviceItemCodeStartCalculateDate: ${getSystemDay(-11)} # 跑计费开始时间

```

间

目编码 列表

MACHINE_TEAM

计价规则

码

```
28     serviceItemCodeEndCalculateDate: ${getSystemDay(-1)} # 跑计费结束时间
29     serviceItemCodePushStartDate: ${getSystemDay(-10)} # 推应收开始时间
30     serviceItemCodePushEndDate: ${getSystemDay(0)} # 推应收结束时间
31     steps:
32         - name: 联营计费应收流程 # setp1
33           module: HAW-业财
34           AWFunc: 联营计费应收流程
35           aw_flag: 1
36           aw_params: { hawCalculate: {
37             public: { # 公共参数
38               account_key: "sm3", # 登录的账号
39               startCalculateDate: $serviceItemCodeStartCalculateDate, # 跑计费开始时
40               endCalculateDate: $serviceItemCodeEndCalculateDate, # 跑计费结束时间
41               pushStartDate: $serviceItemCodePushStartDate, # 推应收开始时间
42               pushEndDate: $serviceItemCodePushEndDate, # 推应收结束时间
43             },
44             calculateDatas: [ { # 计费数据
45               contractBizCode: "10079000000016839175", # 逻辑合同id
46               before_sleep: 8, # 清除D0应收脏数据的前置等待,第一个合同行需入参
47               # pushSleep: 5, # 推应收接口的等待时间,最后一个合同行需入参
48               bizRowCodeDatas: [
49                 { # 订单行合同行数据 -对客
50                   bizCode: "10011000000048427701", # 订单编号
51                   bizRowCode: "10011000000048427703", # 订单行编号
52                   contractBizRowCode: "10079000000016839178", # 逻辑合同行编号# 服务项目
53                   userIdentity: "CUSTOMER", # 客商标识 CUSTOMER - 客 MERCHANT - 商
54                   rentalModeEnum: "DRY", # 租赁模式 DRY-干租, WET-湿租, RANK-趟次
55                   periodUnitEnum: "DAY", # 工期, 天 DAY, 月 MONTH, 趟 RANK, 台班
56                   rentCalcPriceRuleType: "2", # 租金计价规则类型
57                   rentCalcPriceRule: '{"perPrice":1000,"monthPrice":18000}', # 租金
58                   taxRate: 6, # 税率 如6% 则入参 6
59                   fareCalcPriceRuleType: "FIXED_PER_PRICE", # 运费计价规则类型
60                   fareCalcPriceRule: '{"perPrice":900}', # 运费计价规则
61                   farePerPrice: 900, # 运费单价
62                   overtimeCalcPriceRule: '{"perPrice":200}', # 加班计价规则 湿租传
63                   overtimePerPrice: 200, # 加班费价格 湿租传
64                   serviceItemCodeDatas: [ { # 服务项目数据
65                     serviceItemCode: "17598600081262903332-20240220-001", # 服务项目编
66                     enterEventTime: $serviceItemCodeEnterEventTime, # 进场时间
67                     exitEventTime: $serviceItemCodeExitEventTime, # 退场时间
68                     pauseEventTime: $serviceItemCodePauseEventTime, # 暂停时间
69                     recoveryEventTime: $serviceItemCodeRecoveryEventTime, # 恢复时间
```

时间

编号

目编码 列表

MACHINE_TEAM

计价规则

服务项目编码

```
70     enterCreatedAt: $serviceItemCodeEnterCreatedAt, # 进场创建时间
71     exitCreatedAt: $serviceItemCodeExitCreatedAt, # 退场创建时间
72     pauseCreatedAt: $serviceItemCodePauseCreatedAt, # 暂停创建时间
73     recoveryCreatedAt: $serviceItemCodeRecoveryCreatedAt, # 恢复创建
74
75     enterBusinessCode: "enterCode_SM27701", # 进场业务单号
76     enterBusinessPaperCode: "enterPaperCode_SM27701", # 进场纸质单据
77
78     exitBusinessCode: "exitCode_SM27701", # 退场业务单号
79     exitBusinessPaperCode: "exitPapeCode_SM27701", # 退场纸质单据编号
80     stopBusCode: "BT2024041000000001", # # 暂停恢复业务单号
81   } ]
82 } ]
83 { # 订单行合同行数据 -对商
84   contractBizCode: "1007900000016839202", # 逻辑合同id
85   before_sleep: 8, # 清除D0应收脏数据的前置等待,第一个合同行需入参
86   pushSleep: 5, # 推应收接口的等待时间 最后一个合同行需入参
87   bizRowCodeDatas: [ { # 订单行合同行数据
88     bizCode: "10011000000048427701", # 订单编号
89     bizRowCode: "10011000000048427703", # 订单行编号
90     contractBizRowCode: "1007900000016839203", # 逻辑合同行编号# 服务项
91
92     userIdentity: "MERCHANT", # 客商标识 CUSTOMER - 客 MERCHANT - 商
93     rentalModeEnum: "DRY", # 租赁模式 DRY-干租 , WET-湿租 , RANK-趟次
94     periodUnitEnum: "DAY", # 工期, 天 DAY ,月 MONTH ,趟 RANK, 台班
95     ; 不传则按计费老逻辑取计量数据
96     rentCalcPriceRuleType: "2", # 租金计价规则类型
97     rentCalcPriceRule: '{"perPrice":1000,"monthPrice":18000}', # 租金
98
99     taxRate: 13, # 税率 如6% 则入参 6
100     fareCalcPriceRuleType: "FIXED_PER_PRICE", # 运费计价规则类型
101     fareCalcPriceRule: '{"perPrice":900}', # 运费计价规则
102     farePerPrice: 900, # 运费单价
103     overtimeCalcPriceRule: "", # 加班计价规则 湿租传
104     overtimePerPrice: "", # 加班费价格 湿租传
105     serviceItemCodeDatas: [ { # 服务项目数据
106       serviceItemCode: "MERCHANT_1759860081262903332-20240220-001",
107
108       calculateSleep: 5, # 跑完计费接口的等待时间 最后一个服务项目需入参
109     } ]
110   } ]
111 } ]
112 }
```

存在haw内部传参的示例：见下面黄底文字部分

```
1 # 作废账单
2 - case: HW逾期计算
3 epic: 账单中台
4 feature: 逾期
5 story: 逾期
6 severity: P0
7 tag: wyf
8 variables:
9 # 供应链例子
10 # Haw_Overdue: {
11 #   public: {
12 #     bissline: 1, # 业务线, 1供应链, 2联营
13 #     account_key: "sm1", # 登录账号
14 #     order_code: "1625692803776679997", # 根订单号
15 #     gyl_order_code: '1075446809123119104', # 供应链子订单号
16 #     settle_type: 3, # 结算方式
17 #     settleRatio: '0.7', # 结算比例
18 #     account_period: 30, # 账期
19 #     end_account_period: 30, # 尾款账期
20 #     reconCycle: 2, # 对账周期
21 #     Overdue: '2022-11-25', # 逾期跑定时任务时间
22 #     mockEnterDate: '2022-04-25', # 模拟退场时间
23 #     mockExitDate: '2022-09-25', # 模拟进场时间
24 #   },
25 #   devices: [{
26 #     # 供应链需要下面参数
27 #     gyl_order_code: 1075446809123119104, # 供应链订单号
28 #     # 联营需要下面参数
29 #     subOrderCode: 1001100000046100618, # 子订单号
30 #     parentOrderCode: 1735134240088051794, # 根订单号
31 #     ystzs: [
32 #       # 1
33 #       {
34 #         occurDate: '2022-05-01', sendTime: '2022-05-01
35 #         08:00:00', sheetCode: sheetCode_tz_wyf_011,
36 #         sheetType: 1, sheetSubType: 1, amount: 100,
37 #       },
38 #       {
39 #         occurDate: '2022-05-31', sendTime: '2022-05-31
40 #         08:00:00', sheetCode: sheetCode_tz_wyf_0211,
41 #         sheetType: 1, sheetSubType: 1, amount: 100,
42 #       },
43 #       # 2
44 #       {
45 #         occurDate: '2022-06-01', sendTime: '2022-06-01
46 #         08:00:00', sheetCode: sheetCode_tz_wyf_031,
```

```

44 #      sheetType: 1,sheetSubType: 1,amount: 100,
45 #      },
46 #      {
47 #      occurDate: '2022-06-30',sendTime: '2022-06-30
08:00:00',sheetCode: sheetCode_tz_wyf_041,
48 #      sheetType: 1,sheetSubType: 1,amount: 100,
49 #      },
50 #      # 3
51 #      {
52 #      occurDate: '2022-07-01',sendTime: '2022-07-01
08:00:00',sheetCode: sheetCode_tz_wyf_051,
53 #      sheetType: 1,sheetSubType: 1,amount: 100,
54 #      },
55 #      {
56 #      occurDate: '2022-07-31',sendTime: '2022-07-31
08:00:00',sheetCode: sheetCode_tz_wyf_061,
57 #      sheetType: 1,sheetSubType: 1,amount: 100,
58 #      },
59 #      # 4
60 #      {
61 #      occurDate: '2022-08-01',sendTime: '2022-08-01
08:00:00',sheetCode: sheetCode_tz_wyf_071,
62 #      sheetType: 1,sheetSubType: 1,amount: 100,
63 #      },
64 #      {
65 #      occurDate: '2022-08-31',sendTime: '2022-08-31
08:00:00',sheetCode: sheetCode_tz_wyf_081,
66 #      sheetType: 1,sheetSubType: 1,amount: 100,
67 #      },
68 #      # 5
69 #      {
70 #      occurDate: '2022-09-01',sendTime: '2022-09-01
08:00:00',sheetCode: sheetCode_tz_wyf_091,
71 #      sheetType: 1,sheetSubType: 1,amount: 100,
72 #      },
73 #      {
74 #      occurDate: '2022-09-30',sendTime: '2022-09-30
08:00:00',sheetCode: sheetCode_tz_wyf_092,
75 #      sheetType: 1,sheetSubType: 1,amount: 100,
76 #      },
77 #      ]
78 #      }],
79 #      bills: [{
80 #      jobDate: "2022-07-01", 'del': 1 # 拉周期账单, del 是提前清理重置之
前的账单
81 #      },{
82 #      jobDate: "2022-09-01", # 拉周期账单

```

```

83 # }, {
84 #     jobDate: "2022-11-01", # 拉周期账单
85 # }
86 # ]
87 # }
88 steps:
89
90 # 查询分表分库下标, 给下面步骤使用
91 - name: 自营计费入核销中台-查询下标(应收、预收删除
92   module: write-off-center
93   aw_flag: 1
94   AWFunc: 根据订单号查询分表下标及清理数据
95   aw_params: { 'orderCode': '{{ Haw_Overdue.public.order_code }}',
96               'account_key': '{{ Haw_Overdue.public.account_key }}' }
97
98 # 创建结算档案
99 {% if Haw_Overdue.public.settle_type %}
100 - name: 档案创建消息
101   module: bill-center
102   AWFunc: 档案创建消息
103   aw_flag: 1
104   aw_params: {
105     {% if Haw_Overdue.public.bissline == 1 %}
106     'orderCode': '{{ Haw_Overdue.public.gyl_order_code }}',
107     {% else %}
108     'orderCode': '{{ Haw_Overdue.public.order_code }}',
109     {% endif %}
110     'billingDate': '{{ Haw_Overdue.public.billingDate or 1 }}',
111     'accountPeriod': '{{ Haw_Overdue.public.account_period }}',
112     'endAccountPeriod': '{{ Haw_Overdue.public.end_account_period }}',
113     'settleType': '{{ Haw_Overdue.public.settle_type }}',
114     'settleRatio': '{{ Haw_Overdue.public.settleRatio }}',
115     'busLine': {{ Haw_Overdue.public.busLine }},
116     'delete': 'dossier',
117     'account_key': '{{ Haw_Overdue.public.account_key }}',
118     'plaTradeId': '{{ Haw_Overdue.public.order_code }}',
119     'reconCycle': {{ Haw_Overdue.public.reconCycle }},
120     {% if Haw_Overdue.public.mockExitDate %}
121     'car_finish_date': '{{ Haw_Overdue.public.mockExitDate }}',
122     {% endif %}
123   }
124 {% endif %}
125
126 # 创建应收调整 供应链||联营
127 {% for device in Haw_Overdue.devices %}
128 # 供应链应收调整

```

```

130     {% if device.ystzs %}
131     {% for ystz in device.ystzs %}
132     - name: 应收调整
133       module: write-off-center
134       aw_flag: 1
135       AWFunc: 自营应收单据
136       aw_params: { 'orderCode': '{{ device.gyl_order_code }}',
137                   'occurDate': '{{ ystz.occurDate }}',
138                   'sendTime': '{{ ystz.sendTime }}',
139                   'sheetType': 1,
140                   'sheetSubType': 2,
141                   'sheetCode': '{{ ystz.sheetCode }}',
142                   'amount': {{ ystz.amount }},
143                   'account_key': '{{ Haw_Overdue.public.account_key }}' }
144     {% endfor %}
145     {% endif %}
146     # 联营应收调整
147     {% if device.yl_ystzs %}
148     {% for ystz in device.yl_ystzs %}
149     - name: 计费应收-租金-运费 # setp1
150       module: write-off-center
151       AWFunc: 应收调整_联营
152       aw_params: {
153         busLine: {{ystz.busLine or 2}}, #业务线,1:自营, 2:联营
154         targetBody: '{{ystz.targetBody or 1}}', #目标主体,1:对客单
155         orderCode: '{{Haw_Overdue.public.order_code or null}}', #逻辑合同编号
156         occurDate: '{{ystz.occurDate or null}}', #费用发生日期,年月日
157         sendTime: '{{ystz.sendTime or null}}', #发送时间
158         'sheetType': 1,
159         sheetSubType: 1, #单据子类型,1:租金 2:运费
160         sheetCode: '{{ystz.sheetCode or null}}', #单据编号 必须唯一
161         amount: {{ystz.amount or 0}}, #单据金额
162         taxRate: 1, #税率(调整单有),1:6%,2:9%,3:13%,4:1%,5:0%
163         remark: '{{ystz.remark or null}}', #说明信息,如:调整原因
164         subOrderCode: '{{device.subOrderCode or null}}', #子订单号
165         parentOrderCode: '{{device.parentOrderCode or null}}', #主订单号 (不是根订单号)
166       }
167     {% endfor %}
168     {% endif %}
169     {% endfor %}
170
171     # 生成给逾期使用的周期账单的过程数据
172     {% for bill in Haw_Overdue.bills %}

```



```

173 - name: 逾期周期过程数据
174 module: write-off-center
175 AWFunc: 逾期周期过程数据
176 aw_params: { "orderCode": '{{ Haw_Overdue.public.order_code }}',
177               "jobDate": "{{ bill.jobDate }}",
178               {% if bill.del %}
179               'del': {{ bill.del }},
180               {% endif %}
181               {% if Haw_Overdue.public.mockEnterDate %}
182               "mockEnterDate": "{{ Haw_Overdue.public.mockEnterDate }}",
183               {% endif %}
184               {% if Haw_Overdue.public.mockExitDate %}
185               "mockExitDate": "{{ Haw_Overdue.public.mockExitDate }}",
186               {% endif %}
187           }
188 {% endfor %}
189
190 # 触发逾期生成
191 - name: 触发逾期数据生成
192 module: write-off-center
193 AWFunc: 计算应收逾期
194 aw_params: { 'order_code': '{{ Haw_Overdue.public.order_code }}',
195               'settle_type': {{ Haw_Overdue.public.settle_type }},
196               'occurDate': '{{ Haw_Overdue.public.Overdue }}',
197               'account_key': '{{ Haw_Overdue.public.account_key }}',
198               'index': '$.step.self1.index',
199               {% if Haw_Overdue.public.mockEnterDate %}
200               mockEnterDate: '{{ Haw_Overdue.public.mockEnterDate }}',
201               {% endif %}
202               {% if Haw_Overdue.public.mockExitDate %}
203               mockExitDate: '{{ Haw_Overdue.public.mockExitDate }}',
204               {% endif %}
205           }

```

用例篇

本篇幅会对用例编写的方方面面进行介绍, 用例YML可以分为4块

- **用例基本信息:** 用例基本信息主要是体现用例标题, 所属功能模块, 用例级别等等基本信息
- **用例前置:** 用例前置的关键字是setup, 目前支持sql执行, 可以在用例执行之前做一些数据库的初始化(update, delete)
- **用例步骤:** 每一个步骤就是一个AW, 步骤由AW编排而成, 通过编排, 完成流程场景测试
- **用例后置:** 用例后置和前置一样, 可以执行sql, 处理用例执行后的数据清理工作

用例基本信息

项目对各字段分别进行了解释

```
1 # -*- coding: utf-8 -*-
2 - case: 自营电子普票退票申请-审核 # 用例标题
3 epic: 业财 # 业务线
4 feature: 发票中台 # 领域
5 story: 自营发票申请 # 功能特性
6 severity: P0 #用例级别
7 tag: lxt #用例编写人
```

用例前置

如下面示例, 前置的关键字是**setup**, 子关键字是**sql**, 目前只支持**sql**

里面可以直接放**sql**, 放的里面的**sql**语句要比**sql**关键字有一定的**缩进**

下面的`{% %}` 和`{{ }}` 是**jinja2**的语法, 可以自行说些, 后面也会专门介绍下一些常用的语法和案例

```
1 ## -*- coding: utf-8 -*-
2 - case: 档案创建消息
3 ... (省略)
4 steps:
5 ... (省略)
6 setup:
7 sqls:
8     {% if delete == 'dossier' %}
9     delete from billing_center.js_settle_dossier where order_code = {{
10 orderCode/myDefault(200) }};
11     delete from billing_center.js_settle_dossier_change where order_code =
12     {{ orderCode/myDefault(200) }};
13     {% endif %}
```

用例步骤

前面讲过用例的步骤是AW的拼装:

项目的用例包含一个aw: **退票申请**, 这个aw必须是在**invoice-center**注册时对应的class里面进行注册的, 如下图

```
1 # -*- coding: utf-8 -*-
2 - case: 退票申请-电子普票
```

```

3 epic: 业财
4 feature: 发票中台
5 story: 居间-佣金
6 severity: P0
7 tag: lxt
8 variables:
9   applyCode: "FA10018000000001740038" # 开票申请编码
10  customerId: "10002000000002655087" # 退票人customer编码
11 steps:
12   - name: 退票申请-电子普票
13     module: invoice-center
14     AWFunc: 退票申请
15     aw_params: {"applyCode": "$applyCode",
16                'customerId': '$customerId'}
17   teardown:
18   sqls: UPDATE invoice_center.invoice_apply set invoice_apply.invoice_status
19         = "30" where apply_code= "FA10018000000001740038";
20         UPDATE fbs.fbs_bill_info SET fbs_bill_info.commission_invoice_status =
21         "2" WHERE bill_code = "ZD-1585539868400947200" AND rent_amount = 201.00;

```

用例后置

- 用例后置与前置原理是一样的, 不同的地方是在用例接口请求及断言之后执行后置
- 用例的后置执行不受用例步骤执行失败的影响, 必然会执行, 主要是为了不影响下次自动化批跑
- 用例步骤的案例里面有用例后置, 关键字是: **teardown**

数据生成

数据生成是我认为的使用这个框架的最佳实践方式, 主要思想是把用例中调用的所有AW里面的参数提取到数据生成里面, 进行批注, 方便其他人参考学习

数据生成的关键字: **variables**, 如下例

```

1 - case: 用例
2 epic: 业财
3 feature: 清分结算
4 story: 核销结算
5 severity: P0
6 tag: wyf
7 variables: # 测试数据生成
8   user: settlementRealTime
9   pwd: ${uuid_1("aa")}
10  # 这个变量是id name, phone
11  sql: select id, name, phone from infrastructure_authority.prime_account
12  where id = '1481667533810475086';

```

用例编写最佳实践1

用例yaml

1. 场景流程: 下面的用例中包含了3个aw, 主要完成的是退票申请与退票审核的流程
2. aw是退票申请数据恢复

```
1 # -*- coding: utf-8 -*-
2 - case: 自营电子普票退票申请-审核
3   epic: 业财
4   feature: 发票中台
5   story: 自营发票申请
6   severity: P0
7   tag: lxt
8   variables:
9     bizCode1: orderlxt008 # 业务单号
10    invoice_rent: 200 # 已开票租金
11    not_invoice_freight: 200 # 订单未开票运费
12    applyCode: FA10018000000001938008
13    customerId: "10002000000002655087" # 退票人customer编码
14    departmentCode: "Z00456"
15    managerCode: "001535"
16    refundApplicant: "小明"
17    bizType: '02'
18   steps:
19     - name: 自营退票申请-退票申请数据恢复
20       module: invoice-center
21       AWFunc: 申请退票数据恢复
22       aw_params: {'orderCode': '$bizCode1',
23                  'invoice_rent': '$invoice_rent',
24                  'not_invoice_freight': '$not_invoice_freight'}
25     - name: 自营退票申请-电子普票
26       module: invoice-center
27       AWFunc: 退票申请
28       aw_params: {"applyCode": "$applyCode",
29                  'customerId': '$customerId',
30                  'bizType': '$bizType',
31                  'departmentCode': '$departmentCode',
32                  'managerCode': '$managerCode',
33                  'refundApplicant': '$refundApplicant'})
34     - name: 自营退票审核-电子普票
35       module: invoice-center
36       AWFunc: 退票审核
37       aw_params: {"applyCode": "$applyCode",
```

38

`'refundCode': '${step2.refundCode}']})`

aw的yaml

aw1:

主要作用是完成了数据清理

```
1 # -*- coding: utf-8 -*-
2 - case: 申请退票数据恢复
3   epic: 业财
4   feature: 发票中台
5   story: 退票申请
6   severity: P0
7   tag: lxt
8   variables: # 测试数据生成
9
10  steps:
11    - name: 申请退票数据恢复
12      key: kgl
13      jinja2_flag: 0
14      account_key: "{{account_key or 'lxt'}}"
15      module: invoice-center
16      extract:
17        api:
18          method: post
19          url: /invoice-center/v1/object/450020/list
20        params:
21          objectId: 450020
22          gridApiKey: invoiceRefundList
23          current: 1
24          size: 10
25          orders: [ ]
26          conditions: { }
27          query: [ ]
28          apiKeys:
29            - id
30            - name
31            - applyCode
32            - bizType
33            - invoiceApply.sellerSubject
34            - refundType
35            - invoiceApply.invoiceType
36            - invoiceApply.buyerCustomerId
37            - invoiceApply.buyerSubject
38            - invoiceApply.managerName
```

```

39         - invoiceApply.invoiceAmount
40         - refundStatus
41         - refundReason
42         - createTime
43         - refundApplicant
44         - departmentName
45         - refundTime
46         - approve
47         - refundNo
48     validate:
49         eq:
50             status_code: {{ status_code/myDefault(200) }}
51             errCode: {{ errCode/myDefault(0) }}
52             message: {{ message/myDefault(请求成功) }}
53     aw_teardown:
54         sqls:
55             delete from invoice_center.invoice_refund where apply_code =(select
invoice_apply_biz.apply_code from invoice_center.invoice_apply_biz where
invoice_apply_biz.biz_code = {{ orderCode/myDefault(请求成功) }});
56             UPDATE invoice_center.invoice_detail SET invoice_detail.invoice_status =
30 where apply_code = (select invoice_apply_biz.apply_code from
invoice_center.invoice_apply_biz where invoice_apply_biz.biz_code = {{
orderCode/myDefault(请求成功) }});
57     teardown:
58         sqls:

```

aw2

发起退票退票申请

```

1  # -*- coding: utf-8 -*-
2  - case: 发票退票申请
3    epic: 业财
4    feature: 发票中台
5    story: 退票申请
6    severity: P0
7    tag: lxt
8    variables: # 测试数据生成
9        sleep: 1
10   steps:
11       - name: 退票申请 #
12         key: kgl
13         jinja2_flag: 0
14         account_key: lxt
15         module: invoice-center
16         extract:

```

```

17     refundCode: data.refundCode # {"data":{"refundCode":"xxx"}}
18   api:
19     method: post
20     url: /invoice-center/api/invoice/refund
21   params:
22     applyCode: {{ applyCode/myDefault(请求成功) }} # 开票申请编码
23     bizType: {{ bizType/myDefault(请求成功) }} # 业务类型佣金01 自营租赁费02
24     customerId: {{ customerId/myDefault(请求成功) }} # 退票人customer编码
25     {% if bizType == '02' %}
26     departmentCode: {{ departmentCode/myDefault(请求成功) }} # 部门编码
27     departmentName: {{ departmentName/myDefault(请求成功) }} # 部门名称
28     managerCode: {{ managerCode/myDefault(请求成功) }} # 客户经理编码
29     managerName: {{ managerName/myDefault(请求成功) }} # 客户经理名称
30     refundApplicant: {{ refundApplicant/myDefault(请求成功) }} # 申请人
31     {% endif %}
32     refundReason: {{ refundReason/myDefault(请求成功) }} # 退票原因
33     refundRemark: {{ refundRemark/myDefault(请求成功) }} # 退票备注
34     refundType: {{ refundType/myDefault(请求成功) }} # 退票类型1:红字发票 2作废
发票
35   validate:
36     eq:
37       status_code: {{ status_code/myDefault(请求成功) }}
38       errCode: {{ errCode/myDefault(请求成功) }}
39       message: {{ message/myDefault(请求成功) }}
40       refundStatus: {{ refundStatus/myDefault(请求成功) }}

```

aw3就省略了

AW篇

前面提到AW是对单个接口或者多个接口的一个封装, 通常我们是封装单个接口, 然后在用例中调用多个AW来组装成用例.

前面快速入门对于一个典型的例子进行了说明, 这里会对一个AW的全部内容展开一个说明

AW图解

aw_根据订单号查询分表下标及清理核销表.yaml x run_api.py x

```
1 ## -*- coding: utf-8 -*-
2 - case: 根据订单号查询分表下标
3   epic: 业财
4   feature: 核销中台
5   story: 分表下标查询
6   severity: P0
7   tag: wyf
8   variables:
9     sleep: 1
10 steps:
11   - name: 根据订单号查询分表下标 # setp1
12     key: kgl
13     jinja2_flag: 0
14     account_key: "{{account_key or 'sm2'}}"
15     headers:
16     module: write-off-center
17     extract:
18       index: data # 提取下标给后面的步骤使用 {'step1': {'index': 3}}
19     api:
20       method: get
21       url: /write-off-center/writeoff/getTableSuffix
22       params:
23         orderCode: "{{orderCode or null}}" # 订单号
24     validate:
25       eq:
26         status_code: {{status_code or 200}}
27         errCode: "{{errCode or 0}}"
28         message: "{{message or '请求成功'}}"
29       sql:
30     aw_teardown: <1 key>
31     teardown:
32       sqls:
```

aw基本信息: 主要有用到的case, feature, story

数据构造: 可以调用外部函数构造数据, 还可以使用sleep与before_sleep在接口请求前后暂停,来满足业务需要

请求账号

消息头: 会添加到最终请求的消息头里面

接口所属的服务

数据提取: 提取response里面的数据,给后续的步骤使用

接口请求信息: method, url, params data,files等

断言: 支持response断言与sql断言 具体可以见断言篇

aw teardown: aw的后置, 在aw执行完后立即执行的sql脚本

用例后置: 用例执行完成后执行的后置sql脚本

典型yaml

```
1 ## -*- coding: utf-8 -*-
2 - case: 根据订单号查询分表下标
3   epic: 业财
4   feature: 核销中台
5   story: 分表下标查询
6   severity: P0
7   tag: wyf
8   variables:
9     sleep: 1
10 steps:
11   - name: 根据订单号查询分表下标 # setp1
12     key: kgl
13     jinja2_flag: 0
14     account_key: "{{account_key|myDefault(sm2)}}"
15     headers:
16     module: write-off-center
17     extract:
18       index: data # 提取下标给后面的步骤使用 {'step1': {'index': 3}}
```



```

19     api:
20         method: get
21         url: /write-off-center/writeoff/getTableSuffix
22     params:
23         orderCode: {{orderCode|myDefault(0)}} # 订单号
24     validate:
25         eq:
26             status_code: {{status_code|myDefault(0)}}
27             errCode: {{errCode|myDefault(0)}}
28             message: {{message|myDefault(0)}}
29     sql:
30     aw_tearardown:
31     sqls:
32         delete from write_off_center.hx_msg_event where order_code =
33         {{order_code|myDefault(0)}};
34         delete from write_off_center.ad_advance where order_code =
35         {{order_code|myDefault(0)}};
36         .....
37     teardown:
38     sqls:

```

基本信息

由于历史原因,目前AW的基本信息和用例基本信息是一样的, 因为AW是直接拷贝的用例模板修改的. 这块不是很重要, 如果觉得需要优化后续再优化

数据构造

典型例子

接口前后sleep

```

1  ## -*- coding: utf-8 -*-
2  - case: 根据订单号查询分表下标
3  epic: 业财
4  feature: 核销中台
5  story: 分表下标查询
6  severity: P0
7  tag: wyf
8  variables:
9      sleep: 1 # aw接口请求之后sleep
10     before_sleep: 0.1 # 接口请求之前sleep
11  steps:
12      - name: 根据订单号查询分表下标 # setp1
13        ...省略

```

调用外部函数构造数据

对于一些复杂的数据, 我们需要调用外部数据来构造数据

约定: `${}` 里面包裹的是hook函数

变量引用用 `$`

```
1  ## -*- coding: utf-8 -*-
2  - case: 资金流水导入
3    epic: 业财
4    feature: 账款中台
5    story: 资金流水导入
6    severity: P0
7    tag: raj
8    variables: # 测试数据生成
9      fileurl: {{fileurl or '/data/kg1/accountscenter/银行资金流水导入模板.xls'}}
10     file_name: ${file_name("${fileurl")}
11     file_rb: ${file_rb1("${fileurl")}
12   steps:
13     - name: 资金流水导入 # setp1
14       key: kg1
15       jinja2_flag: 0
16       headers: {'terminal': 'WEBPC'}
17       account_key: "{{account_key or 'raj'}}" #测试是使用的账号, 使用时请修改为自己
使用的测试账号
18       module: accounts-center
19       api:
20         method: post
21         url: /accounts-center/boss/account/flow/import
22       files:
23         file:
24           - '$file_name'
25           - $file_rb
26           - application/vnd.ms-excel
27       data:
28         bizType: '01'
29       validate:
30         eq:
31           status_code: {{status_code|myDefault(0)}}
32           errCode: {{errCode|myDefault(0)}}
33           message: {{message|myDefault(0)}}
```

utils/base/hook.py

```
1 #!/usr/bin/env ppython
2 # -*-coding:utf-8 -*-
3 """
4 @author: wuyanfeng
5 @contact: wuyanfeng@znlh.com
6 @software: PyCharm
7 @file: hook.py
8 @time: 2022/11/8 17:19
9 """
10 import os
11 import json
12 import uuid
13
14 from config.conf import current
15
16
17 class Generating_test_data:
18     '''构造自定义的测试数据'''
19     @classmethod
20     def uuid_1(cls,name=None):
21         return str(uuid.uuid1())
22
23     @classmethod
24     def file_name(cls, fileurl):
25         '''通过路径获取文件名字'''
26         file_name = os.path.basename(fileurl)
27         return file_name
28
29     @classmethod
30     def data_json(cls, data):
31         data = json.dumps(data)
32         print('data:',data)
33         return data
34
35     @classmethod
36     def file_rb(cls,fileurl):
37         with open(current + fileurl , "rb") as f:
38             filerb = f.read()
39             print("file_rb, fileurl",filerb)
40             return filerb.decode('utf-8')
41
42
43     @classmethod
44     def file_rb1(cls,fileurl):
45         return open(current + fileurl , "rb")
```

断言篇

说明: 目前断言都是在AW里面进行断言(用例级的断言不推荐, 当然也是支持的), 每个AW执行完立即进行断言

断言的关键字是: **validate**

SQL断言

sql断言的关键字是sql关键字是

```
1 - case: 触发应收账款的计算
2   ... (省略)
3   steps:
4     - name: 触发账龄计算 # setp1
5       ... (省略)
6       validate:
7         sql:
```

对比原理

说明: 最终是由两个字典进行对比: 实际结果字典, 和预期结果字典.

如下图, 预期结果key的数量小于等于实际结果

注意: 在yaml里面字典里面的key后面的冒号后面, 必须有空格才被解析为字典{ 'a' :xx}

实际结果{'a': xx, 'b': xx, 'c': xx}

预期结果{'a': xx, 'b': xx}

具体场景

SQL与SQL结果比较

SQL的查询结果都是字典

```

1      ... 省略
2      validate:
3      sql:
4          -- select a.id as 账号,
5              a.phone as phone
6              from infrastructure_authority.prime_account
7              where a.id = '{{id}}'; # 返回结果{'账号': 'wyf',
'phone': '15355558888'}
8
9          -- select
10             a.phone as phone
11             from table
12             where a.id = '{{id}}'; # 返回结果{'phone': '15355558888'}

```

预期结果是字典

```

1      ... 省略
2      validate:
3      sql:
4          -- select a.id as 账号,
5              a.phone as phone
6              from infrastructure_authority.prime_account
7              where a.id = '$user'; # 返回结果{'账号': 'wyf',
'phone': '15355558888'}
8          -- { '账号': 'wyf' } # 字典

```

预期结果是空字典

```

1      ... 省略
2      validate:
3      sql:
4          -- select a.id as 账号,
5              a.phone as phone
6              from infrastructure_authority.prime_account
7              where a.id = '$user'; # 返回结果{'账号': 'wyf',
'phone': '15355558888'}
8          -- { } # 空字典

```

预期结果是列表

```

1      ... 省略
2      validate:

```

```

3      sql:
4      - - select a.id as 账号,
5          a.phone as phone,
6          a.platform as 平台,
7          a.customer_id as 客户身份
8          from infrastructure_authority.prime_account a left join
9          infrastructure_authority.prime_customer b on a.customer_id = b.id
10         where a.phone = '15380754163' order by a.id;
11      # 预期返回的是一个列表, 和上面的查询sql的顺序需要保持一致
12      - [{ '账号': 10002000000002654484, 'phone': '15380754163', '平台': 1 },
13        { '账号': 1481667533810475086, 'phone': '15380754163', '平台': 1 },
14        { '账号': 1483359604875272228, 'phone': '15380754163', '平台': 2 },
15        { '账号': 1485450701529247793, 'phone': '15380754163', '平台': 2 },
16        { '账号': 1485450701529247794, 'phone': '15380754163', '平台': 2 },
17        { '账号': 1485450701529247795, 'phone': '15380754163', '平台': 2 },
18        { '账号': 1486173689203486783, 'phone': '15380754163', '平台': 3 } ]

```

基于hook的返回作为断言

当逾期或者实际结果, 通过数据库查询无法满足时, 需要编写hook函数返回字典的方式继续对比如下标黄的内容

```

1  ## -*- coding: utf-8 -*-
2  - case: 触发应收账款龄的计算
3    epic: 业财
4    feature: 核销中台
5    story: 逾期
6    # 幂等性判断:
7    idempotent: # 对比下面两个值相等, 认为是幂等, 不会执行此步骤
8      - select order_code from write_off_center.re_overdue where order_code
9        = 'wyf_order_zl_0051' order by created_at desc ;
10     - { 'order_code': '{{order_code}}' }
11  variables:
12    before_sleep: 5
13    # 逾期结果, 通过函数返回字段, 在下面进行断言
14    re_overdue: ${re_overdue('{{order_code}}', '{{occurDate}}', '{{index}}', env)}
15  steps:
16    - name: 触发逾期计算 # setp1
17      key: kgl
18      jinja2_flag: 0
19      account_key: "{{account_key or 'sm2'}}"
20      module: write-off-center
21      api:
22        method: get
23        url: /write-off-center/receivable/mock/calcReceivableOverdue

```

```

24     params:
25         occurDate: "{{occurDate or null}}" # 日期 定时任务时间
26     validate:
27         sql:
28             {% if order_code and settle_type in (3,4) %}
29             -- select income_amount,
30                 not_income_amount,
31                 overdue_amount30,
32                 overdue_amount60,
33                 overdue_amount90,
34                 overdue_amount_other,
35                 not_overdue_amount,
36                 overdue_amount_total
37             from write_off_center.re_overdue
38             where order_code = '{{order_code}}'
39             order by created_at desc;
40             - $re_overdue
41             {% endif %}
42     teardown:
43         sqls:

```

Response断言

支持等于eq和in

1. eq就是response里面的某个字段与实际结果相等
2. In 实际结果字符串在response里面, in可以用||分开多个字符串片段

eq&in

对比原理

对比response里面的json字段(通过jsonpath获取)与预期结果(一般是通过用例动态传递进来)的对比

in是把返回的json当成字符串, 这里会忽略双引号和单引号,

'status_code' 和 "status_code" 是一样的, 忽略引号

使用场景

1. eq:下面例子中,比对的是{"data":[1,2,3]}返回response的列表中的第一个的值是否是1, jsonpath的表达式是\$.data.0, 这里取值需要写data.0即可
 - a. 增加了对提取结果的长度len的短语 len(data):2
2. In:下面例子中可以看出, in是把response的返回当成了一个完整的字符串, 我们可以判断它是否包含我们预期的字符串, 下面例子判断了 'status_code': 20 和 'message': '异常信息: accountId 和jobNumber参数不能同时` 是否在response里面, 显然是在的.

```

1 response: {"data": [1, 2, 3]} # eq: response
2 response: {'status_code': 200, 'errCode': 'A010004', 'message': '异常信息: accountId和jobNumber参数不能同时为空', 'data': null} # in的response

```

```

1 ## -*- coding: utf-8 -*-
2 - case: 查询结算档案
3 ... (省略)
4 steps:
5   - name: 查询结算档案 # setp1
6   ... (省略)
7   validate:
8     eq:
9       status_code: {{status_code or 200}}
10      errCode.name: "jum"
11      len(data): {{num}}
12      obj_in(data): {{num}}
13 ## -*- coding: utf-8 -*-
14 - case: 查询结算档案
15 ... (省略)
16 steps:
17   - name: 查询结算档案 # setp1
18   ... (省略)
19   validate:
20     eq:
21       data.0: 1 # data.0是response里面的取的值, 1是逾期结果
22       # 'status_code': 200和'message': '请求成功'是预期结果, 判断在response里面是
23       in: "'status_code': 200 || 'message': '异常信息: accountId和jobNumber参数不能同时存在'"

```

gt<


```
6      errCode: "{{errCode or 0}}"
7      message: "{{message or '请求成功'}}"
8      gt:
9      status_code: {{status_code or 200}}
10     lt:
11     status_code: {{status_code or 200}}
12
13     deleteOld: {{deleteOld(myDefault(true))}} # 是否清除历史数据
14     validate:
15     eq:
16     status_code: {{status_code or 200}}
17     errCode: "{{errCode or 0}}"
18     message: "{{message or '请求成功'}}"
19     gt:
20     status_code: {{status_code or 190}} # 实际值大于预期
21     lt:
22     status_code: {{status_code or 210}} # 实际值小于预期
23
24 Document 1/1 > Item 1/1 > steps: > Item 1/1
25
26 sts passed: 1 of 1 test - 2 sec 451 ms
27
28 3-12-27 18:38:21,896 - INFO - run_api.py - run_api - [181]: 替换变量之后的data: {'orderCode': '1007900000015879255', 'deleteOld': True, 'message': '请求成功', 'status_code': 200}
29 3-12-27 18:38:21,897 - INFO - run_api.py - run_api - [208]: 请求的方法: post
30 3-12-27 18:38:21,897 - INFO - run_api.py - run_api - [209]: 请求的url: /write-off-center/msgevent/checkAndReRunByOrder
31 3-12-27 18:38:21,897 - INFO - run_api.py - run_api - [214]: 发送请求的数据: {'orderCode': '1007900000015879255', 'deleteOld': True, 'message': '请求成功', 'status_code': 200}
32 3-12-27 18:38:24,328 - INFO - run_api.py - run_api - [247]: resp: {'status_code': 200, 'errCode': '0', 'message': '请求成功'}
33 3-12-27 18:38:24,328 - INFO - run_api.py - run_api - [248]: TID:2adbec8e5f242d98ddfb2ef6916ef5a.154.17036735023549021
34 3-12-27 18:38:24,328 - INFO - run_api.py - run_api - [110]: 当前步骤是1, 提取step参数: {'step1': {}}
35 3-12-27 18:38:24,328 - INFO - run_api.py - run_api - [110]: 当前步骤是1, 提取step参数: {'step1': {}}
36 3-12-27 18:38:24,329 - INFO - assertUtil.py - assertUtil - [72]: eq: status_code actual value is 200
37 3-12-27 18:38:24,329 - INFO - assertUtil.py - assertUtil - [73]: eq: status_code expected value is 200
38 3-12-27 18:38:24,329 - INFO - assertUtil.py - assertUtil - [72]: eq: errCode actual value is 0
39 3-12-27 18:38:24,330 - INFO - assertUtil.py - assertUtil - [73]: eq: errCode expected value is 0
40 3-12-27 18:38:24,330 - INFO - assertUtil.py - assertUtil - [72]: eq: message actual value is 请求成功
41 3-12-27 18:38:24,330 - INFO - assertUtil.py - assertUtil - [73]: eq: message expected value is 请求成功
42 3-12-27 18:38:24,330 - INFO - assertUtil.py - assertUtil - [99]: gt: status_code actual value is 200
43 3-12-27 18:38:24,330 - INFO - assertUtil.py - assertUtil - [100]: gt: status_code expected value is 190
44 3-12-27 18:38:24,330 - INFO - assertUtil.py - assertUtil - [126]: lt: status_code actual value is 200
45 3-12-27 18:38:24,330 - INFO - assertUtil.py - assertUtil - [127]: lt: status_code expected value is 210
46 3-12-27 18:38:24,330 - INFO - run_api.py - run_api - [477]: test case passed
47
48 process finished with exit code 0
```

底层实现:

```
1 ... (省略) # eq:
2     if eqs:
3         for k, v in eqs.items():
4             a_v = jsonpath(resp, f"${k}") [0]
5             assert a_v == v
6
7 ... (省略) # in
8     resp_str = json.dumps(resp, ensure_ascii=False).replace("'", '"')
9     ins = data.get("in").split("||")
10    log.info(f"in: actual value is {resp_str}")
11    log.info(f"in: expected is {ins}")
12    if ins:
13        for inn in ins:
14            inn = inn.replace("'", '"')
15            assert inn in resp_str
16    return True
```

Like

比如apaas是通过name进行筛选的，可以使用like进行断言，下面是使用"呵%"（含义和数据库的一样）查询的，我们使用下面的方式进行断言：`data.records.name: '呵%'`

```
1 {
2     "status_code": 200,
3     "errCode": "0",
4     "message": "请求成功",
5     "data": {
6         "current": 1,
7         "apiKeys": [
8             "birthday",
9             "leader",
10            "leader.name",
11            "code",
12        ],
13        "orders": [{"createdAt": 1}],
14        "records": [
15            {
16                "name": "呵*",
17                "id": "1072542749105483776",
18                "age": 0,
19                "__typename": "customer",
20                "text": "呵呵",
21                "jobNumber": "[003705]熊哲",
22            },
23            {
24                "name": "呵*",
25                "id": "1072542749105483776",
26                "age": 0,
27                "__typename": "customer",
28                "text": "呵呵",
29                "jobNumber": "[003705]熊哲",
30            },
31        ],
32        "total": 1,
33        "size": 10,
34    },
35 }
```

```
1 ## -*- coding: utf-8 -*-
2 - case: apaas-list接口
3     epic: 低码平台
```

```
4   feature: 低码平台list查询接口
5   story: 低码平台list查询接口
6   severity: P0
7   tag: raj
8   steps:
9   - name: 低码平台list查询接口 # setp1
10     key: farm
11     jinja2_flag: 0
12     account_key: "{{account_key or 'raj'}}" #接口使用账号, 调用请修改为自己的测试
    账号
13     module: demo-pass
14     extract:
15     api:
16       method: post
17       url: /{{serviceid}}/v1/object/{{objectId}}/list
18     params:
19       objectId: {{objectId or null}}
20       gridApiKey: "{{gridApiKey or null}}"
21       current: {{current or null}}
22       size: {{size or null}}
23       orders: {{orders or [ ] }}
24       conditions: {{conditions or { } }}
25       query: {{query or [ ] }}
26       apiKeys: {{apiKeys or [ ] }}
27     validate:
28       eq:
29         status_code: {{status_code or 200}}
30         errCode: "{{errCode or 0}}"
31       likes:
32         data.records: "{{query}}"
```

```
1 - case: 低码平台list查询接口-query-like查询
2   epic: 效能
3   feature: 低码平台list查询接口
4   story: 低码平台list查询接口
5   severity: P0
6   tag: raj
7   variables:
8   steps:
9   - name: 低码平台list查询接口-query-like查询 # setp1
10     module: demo-pass
11     AWFunc: apaaslist(env,params ={
12       "serviceid":"demo-paas",
13       "objectId":"101",
14       "gridApiKey":"customerList",
```

```

15     "current":1,
16     "size":10,
17     "orders":[ ],
18     "conditions":{ },
19     "query":[{"name":"name","op":"like","val":"呵%"}],
20     "apiKeys":
["id","code","name","phone","idCard","workflowState","areaCode","age","leader",
"leader.name","leader.phone","standbyPhone","_buttons"]])

```

日志:

```

'action': 'delete', 'icon': 'DeleteFilled'}, {'apiKey': 'view', 'label': '详情', 'action': 'drawer', 'icon': 'icon_s_edit'}]], 'total': 1,
2023-02-09 20:36:19,006 - INFO - run_api.py - run_api - 【99】: 当前步骤是1, 提取step参数:{'step1': {}}
2023-02-09 20:36:19,008 - INFO - run_api.py - run_api - 【99】: 当前步骤是1, 提取step参数:{'step1': {}}
2023-02-09 20:36:19,008 - INFO - assertUtil.py - assertUtil - 【43】: eq: actual value is {'status_code': 200, 'errCode': '0', 'message': '请:
'leader', 'leader.name', 'code', 'address', 'gender', 'idCard', 'verified', 'telephone', '_buttons', 'areaCode', 'contactList', 'createdBy.n
'standbyPhone', 'name', 'createdBy.accountExt.jobNumber', 'id', 'leader.phone', 'age'], 'orders': [{'createdAt': 1}], 'records': [{'birthday
'address': '{"cityAdCode":"610100","adCode":"610112","address":"经济技术开发区凤城九路南侧白桦林居38幢3单元3层30303号","city":"西安市","cityCode":"0:
.344948,"locationDetail":"陕西小百合艺术(经开校区)","longitude":108.937602,"province":"陕西省'}], 'gender': None, 'idCard': '142730199309052213',
'contactList': [], 'balance': 100.0, 'phone': '18720991832', 'workflowState': None, 'standbyPhone': '18620999', 'createdBy': {'name': '熊哲',
'003705', '__typename': 'accountExt', 'text': None, 'name': None}, 'id': '1495942991146250620', '__typename': 'accountInfo', 'text': '熊哲'
'__typename': 'customer', 'text': '呵呵', 'jobNumber': '[003705]熊哲', '_buttons': [{'apiKey': 'edit', 'label': '编辑', 'action': 'edit', 'ico
'action': 'delete', 'icon': 'DeleteFilled'}, {'apiKey': 'view', 'label': '详情', 'action': 'drawer', 'icon': 'icon_s_edit'}]]}, 'total': 1,
2023-02-09 20:36:19,008 - INFO - assertUtil.py - assertUtil - 【44】: eq: expected value is {'status_code': 200, 'errCode': '0'}
2023-02-09 20:36:19,015 - INFO - assertUtil.py - assertUtil - 【87】: like: actual value is 呵呵
2023-02-09 20:36:19,015 - INFO - assertUtil.py - assertUtil - 【88】: like: expected value is {'name': 'name', 'op': 'like', 'val': '呵%'}
2023-02-09 20:36:19,016 - INFO - run_api.py - run_api - 【419】: >>【end:-----

```

数据驱动篇

数据生成目前支持3种

1. 直接写死:
2. 通过hook函数返回的数据:
3. 通过sql查询得到的数据:
4. DDT

DDT篇

使用ddt, 可以在一条用例里面, 通过不同的数据生成不同的用例

实例如下, 下面yaml里面的黄色部分就是ddt的内容, 生成2条用例

- variables里面如果有变量, 会作为全局变量, 给每条DDT生成的用例使用
- ddt里面的数据的变量, 会被加入到variables里面, ddt相当于局部变量, 会覆盖variables已有的同名的全局变量

```

1  ## -*- coding: utf-8 -*-
2
3

```

```

4 - case: 查询可开票金额
5 epic: 业财
6 feature: 核销中台
7 story: 查询可开票金额
8 severity: P0
9 tag: ctt
10 ddt:
11 -
12 test_env:
13     case_name: 预收时间不是当天, 删除失败
14     open: 0 # 1 开, 0 关
15     orderCodeList: cttorder2022110201 #订单号
16 -
17 test_env:
18     case_name: 预收时间不是当天, 删除失败
19     open: 0 # 1 开, 0 关
20     orderCodeList: cttorder2022110201 #订单号
21 variables:
22     orderCodeList: cttorder2022110201 #订单号
23 steps:
24     - name: 查询可开票金额 # setp1
25       module: write-off-center
26       AWFunc: 查询可开票金额(env,params ={
27         "orderCodeList":"$orderCodeList"})
28
29 teardown:

```

ddt添加用例开关与用例标题字段

如下标黄内容, open开关主要是为了方便接口自动化调试使用.

```

1 ddt:
2 - test_env:
3     case_name: 预收时间不是当天, 删除失败
4     open: 0 # 1 开, 0 关
5     orderCode1: order_ly_wyf001 #订单号
6     incomeType: 1 # 收款方式, 1. 微信 2. 支付宝 3. 银行转账
7     amount1: 100 # 来款金额
8     amount2: 100 # 来款金额
9     incomeFlow: incomeFlow_ly_wyf001 #来款流水号
10    incomeTime: 2022-01-03 08:00:00 #来款时间
11    sendTime: 2022-01-03 08:00:00 #发送时间
12    sendTime1: 2022-01-03 08:00:00 #发送时间
13    occurDate: 2022-01-03 #业务发生日
14    busLine: '2' # 联营2, 自营1
15    # 断言

```

```
16     errCode: 'F140020'
17     message: '预收时间不是当天!'
```

ddt_file篇

ddt_file的思想是使用等价类的设计方法, 使用预置好的测试因子, 自动化的生成测试用例的过程, 进一步简化了编写用例的过程

等价类生成用例思想解释

```
1  # 接口的入参的因子的集合
2  a = {
3      'serviceid': [[1, 2, 3, 4, 5, 6, 7], ["#", "%", "&"], [{}, {'code': '405',
4      'message': '字段错误'}]],
5      'objectId': [[11, 'null', 22, 23], ["$"], [{ 'code': '406', 'message': '字段错
6      误'}]],
7      'gridApiKey': [['aa'], ["null"], []],
8      'current': [['bb', 'cc', 'dd'], ['$$'], []],
9  }
10 # 生成的正常用例
11 {'serviceid': 1, 'objectId': 11, 'gridApiKey': 'aa', 'current': 'bb'}
12 {'serviceid': 2, 'objectId': None, 'gridApiKey': 'aa', 'current': 'cc'}
13 {'serviceid': 3, 'objectId': 22, 'gridApiKey': 'aa', 'current': 'dd'}
14 {'serviceid': 4, 'objectId': 23, 'gridApiKey': 'aa', 'current': 'bb'}
15 {'serviceid': 5, 'objectId': 11, 'gridApiKey': 'aa', 'current': 'bb'}
16 {'serviceid': 6, 'objectId': 11, 'gridApiKey': 'aa', 'current': 'bb'}
17 {'serviceid': 7, 'objectId': 11, 'gridApiKey': 'aa', 'current': 'bb'}
18 {'serviceid': {}, 'objectId': {'code': '406', 'message': '字段错误'},
19   'gridApiKey': None, 'current': None}
20 {'serviceid': {'code': '405', 'message': '字段错误'}, 'objectId': None,
21   'gridApiKey': None, 'current': None}
22 # 异常用例
23 {'serviceid': '#', 'objectId': 11, 'gridApiKey': 'aa', 'current': 'bb'}
24 {'serviceid': 1, 'objectId': '$', 'gridApiKey': 'aa', 'current': 'bb', 'code':
25   '406', 'message': '字段错误'}
26 {'serviceid': 1, 'objectId': 11, 'gridApiKey': 'null', 'current': 'bb'}
27 {'serviceid': 1, 'objectId': 11, 'gridApiKey': 'aa', 'current': '$$'}
28 {'serviceid': '%', 'objectId': 11, 'gridApiKey': 'aa', 'current': 'bb', 'code':
29   '405', 'message': '字段错误'}
30 {'serviceid': '&', 'objectId': 11, 'gridApiKey': 'aa', 'current': 'bb'}
31 # 无异常断言信息的异常因子, 建议都对应都有对应的预期结果
32 null无异常断言信息
```

30 \$\$无异常断言信息

31 &无异常断言信息

ddt文件

如下, 嵌套列表中, 第一个列表是正常输入因子, 第二个是异常输入因子, 第三个是异常输入因子的特殊断言(可以没有)

```
1 # -*- coding: utf-8 -*-
2 #params: [[], # 正常值
3           [], # 异常值
4           [] # 异常值对应的错误提示, 或者错误码, 列表的下标和异常因子下标对应
5 'serviceid': ["demo-paas"],
6               [],
7               []
8 'objectId': ["101"],
9              ['-1', "-2"], # 异常用例
10              [{}, {'errorCode': '500', 'message': 'tenantId objectId -1 is not
exists'}]] # 异常用例尽量是个他一个断言
11 'gridApiKey': ["customerList"],
12               [],
13               []
14 'current': [[1],
15             [],
16             []
17 'size': [[100],
18          [],
19          []
20 'orders': [[["createdAt":0]],
21            [],
22            []
23 'conditions': [{"searchPc": "客户"}],
24               [],
25               []
26 'query': [[[]],
27           [],
28           []
29 'apiKeys': [{"id", "code", "name", "phone", "idCard",
30              "workflowState", "areaCode", "age", "leader", "leader.name",
31              "leader.phone", "standbyPhone", "_buttons", "createdAt"}],
32            [],
33            []
```

用例模板文件

用例模板主要是编写需要调用的aw的编排, 主要的作用是参数, 参数化的自动化与因子的key保持一致

```
1 # -*- coding: utf-8 -*-
2 - case: 低码平台list查询接口-conditions-searchPc查询
3 epic: 效能
4 feature: 低码平台list查询接口
5 story: 低码平台list查询接口
6 severity: P0
7 tag: raj
8 ddt_file: '/data/apaas_data/demo.yml' # 有ddt_file时, ddt无效
9 ddt: # 无效
10 - test_env:
11     case_name: conditions查询
12     open: 0 # 1 开, 0 关
13 variables:
14     errCode: '' # 全局变量, 会被ddt里面的传参替换, 正常用例一般不传, 取默认值
15     message: '' # 全局变量, 会被ddt里面的传参替换, 正常用例一般不传, 取默认值
16 steps:
17     - name: 低码平台list查询接口-conditions-searchPc查询 # setp1
18       module: demo-pass
19       AFunc: apaaslist(env,params ={
20         "serviceid":"$serviceid",
21         "objectId":"$objectId",
22         "gridApiKey":"$gridApiKey",
23         "current":$current,
24         "size":$size,
25         "orders":$orders,
26         "conditions":$conditions,
27         "query":$query,
28         "apiKeys":$apiKeys,
29         "errCode":'$errCode',
30         "message":'$message'
31       })
```

aw文件

```
1 ## -*- coding: utf-8 -*-
2 - case: apaas-list接口
3 epic: 低码平台
4 feature: 低码平台list查询接口
5 story: 低码平台list查询接口
6 severity: P0
7 tag: raj
8 steps:
```


账号

```
- name: 低码平台list查询接口 # setp1
key: farm
jinja2_flag: 0
account_key: "{{account_key or 'raj'}}" #接口使用账号, 调用请修改为自己的测试

module: demo-pass
extract:
api:
  method: post
  url: /{{serviceid}}/v1/object/{{objectId}}/list
  params:
    objectId: {{objectId or null}}
    gridApiKey: "{{gridApiKey or null}}"
    current: {{current or null}}
    size: {{size or null}}
    orders: {{orders or [ ] }}
    conditions: {{conditions or { } }}
    query: {{query or [ ] }}
    apiKeys: {{apiKeys or [ ] }}
    {% if unMaskingApiKeys %}
    unMaskingApiKeys: {{unMaskingApiKeys or ["*"] }}
    {% endif %}
  validate:
    eq:
      status_code: {{status_code or 200}}
      errCode: "{{errCode or 0}}"
      {% if errCode == '' %} # 异常有值, 不会走这里
      query:
        data.records: "{{query}}"
      conditions:
        data.records: "{{conditions}}"
      orders:
        data.records: "{{orders}}"
      {% endif %}
```

用例文件

```
1 # !/usr/bin/env_ppython
2 # -*-coding:utf-8 -*-
3 ""
4 @author: renajian
5 @contact: renajian1@znlh.com
6 @software: PyCharm
7 @file: test_contractList.py
```

```

8 @time: 2023/01/31 17:13
9 """
10 import pytest
11
12 from utils.base.myMetaClass import MyMetaClass
13
14 class Test_apaaS(metaclass=MyMetaClass):
15
16     case_yaml_list = [
17
18         '/data/usefulness/demo-paas/test_apaaS_list_files.yaml', #apaas列表查询
        接口
19     ]
20
21 if __name__ == '__main__':
22     pytest.main(["-sv", "test_contractList.py"])

```

跑单条用例

当ddt_file里面用例很多时, 有一条用例失败, 此时如果只想跑失败的单条用例, 可以通过添加debug关键字, 只跑包含debug关键字的用例, 方便调试失败的用例

debug的具体规则如下:

1. 如果存在几条用例都有debug标记时, 只有第一条用例会被执行
2. debug在执行用例时, 关键字会被还原为没有加关键字的形态, 不影响用例执行
3. 一次调试尽量是在了解等价类生成用例规则基础上, 在关键路径上的某一个参数添加debug关键字标记, 即可执行到这条用例
4. debug调试完成后, 需要去掉debug标志, 不然正式批跑不会跑其他用例
5. 如果入参是列表[1]或字典{"a":1}, debug是需要把列表变为"元组", (1):debug, ("a":1):debug, 其他的参数都是直接在后面加 ":debug" 即可:"null":debug, 1:debug, false:debug

```

1 # -*- coding: utf-8 -*-
2 #params: [[], # 正常值
3          [], # 异常值
4          [] # 异常值对应的错误提示, 或者错误码
5
6 # 支持下面几种debug
7 'serviceid': [ ["demo-paas":debug, "null":debug, (1):debug, ("a":1):debug,
8                 1:debug, false:debug],
9                [],
10               []]

```

自动生成ddt_file.yaml文件

data特殊处理

在少量情况下，如果两个接口强依赖，第二个接口的发送请求的data和第一个接口返回的data高度相似，而且字段数量很多的情况，我们可以直接拿第一个接口的data作为参数，对于差异的地方进行替换，就要用到下面的方法

```
1 #第一个接口的步骤
2 .....
3 - name: 编辑整备单列表界面，选择清洁工程师-王强，点击领取整备单
4   module: service-center
5   AWFunc: 选择清洁工程师后点击领取整备单
6   aw_params: {
7     'account_key': '$account_key',
8     'data': '$.step2.data',
9     "data1": {
10       "serviceStaffName": "王强",
11       "__typename": "serviceEngineer",
12       "text": "王强",
13       "id": "1486173689203486781",
14       "name": "王强"
15     },
16     "data2": {
17       "checkStatus": "0",
18       "__typename": "preparationCheck",
19       "id": "2000050000136109676",
20       "serviceEngineer": {
21         "serviceStaffName": "王强",
22         "__typename": "serviceEngineer",
23         "text": "王强",
24         "id": "1486173689203486781",
25         "name": "王强"
26       }
27     }
28   }
29 第二个接口的yaml
30
31 - case: 选择清洁工程师后点击领取整备单
32   epic: 供应链
33   feature: 选择清洁工程师后点击领取整备单
34   story: 选择清洁工程师后点击领取整备单
35   severity: P0
36   tag: wq
37   # 全局变量
38   variables:
```

```

39     data||data.preparationCheck: {{ data1 }}
40     data||data.preparationClean: {{ data2 }}
41     steps:
42         - name: 选择清洁工程师后点击领取整备单
43           key: gyl_sit_stj
44           module: service-center
45           jinja2_flag: 0
46           account_key: {{ account_key|myDefault('wq_stj') }}
47           # 参数提取
48           extract:
49             api:
50               method: post
51               url: /service-center/v1/object/10400005
52             params:
53               endpoint: 2
54               data: {{ data }}
55               layout: "apply"
56               scene: "accept"
57           validate:
58             eq:
59               status_code: 200
60               errCode: "0"
61               message: '请求成功'

```

上面表示：发给后端的参数中有一个data的json中有2个地方需要替换jsonpath指向的是\$.data.preparationCheck和data.preparationClean, 替换为用例里面传递进来的data1和data2

幂等篇

幂等的使用场景: 如果流程很长, 前面有些步骤, 如果是存在可用的历史数据(如果数据不存在, 就执行步骤, 构造数据), 想跳过这个步骤, 可以使用幂等功能

关键字是: idempotent

1. 幂等的具体内容是在AW里面实现的, idempotent 长度为偶数的列表, 奇偶相比, 比如下标0==1, 2==3, 3==4 ...
2. 幂等开关在用例里面判断, 开关默认关闭

AW实现

实例1:

函数结果返回值与, 目标字典对比, 逻辑就是python的`==`

```

1  ## -*- coding: utf-8 -*-

```

```

2 - case: 触发应收账款龄的计算
3 epic: 业财
4 feature: 核销中台
5 story: 逾期
6 # 幂等性判断:
7 idempotent: # 对比下面两个值相等, 认为是幂等, 不会执行此步骤
8     - ${re_overdue('{{order_code}}','{{occurDate}}','{{index}}',env)}
9     - {'order_code': '{{order_code}}'}
10 ...

```

实例2:

写死的值相等, 就是必然是跳过

```

1 ## -*- coding: utf-8 -*-
2 - case: 触发应收账款龄的计算
3 epic: 业财
4 feature: 核销中台
5 story: 逾期
6 # 幂等性判断:
7 idempotent: # 对比下面两个值相等, 认为是幂等, 不会执行此步骤
8     - 1
9     - 1
10 ...

```

实例3:

sql的返回值与目标值比较

```

1 - case: 触发应收账款龄的计算
2 epic: 业财
3 feature: 核销中台
4 story: 逾期
5 # 幂等性判断:
6 idempotent: # 对比下面两个值相等, 认为是幂等, 不会执行此步骤
7 - select order_code from write_off_center.re_overdue where order_code =
  'wyf_order_zl_0051' order by created_at desc ;
8     - {'order_code': '{{order_code}}'}
9 ...

```

幂等开关:

用例里面设置开关为1, 就是判断幂等

```

1  ...
2  - name: 触发逾期数据生成
3    module: write-off-center
4    idempotent: 0 # 1 判断幂等性, 0或者不写, 不判断幂等性
5    AWFunc: 计算应收逾期(env,
6      params ={'order_code': '$bizCode',
7        'index': '$.step4.index',
8        'settle_type': '$settle_type',
9        'occurDate': '$occurDate_zl_data',
10       'account_key': '$account_key'})
11  teardown:

```

参数传递篇

aw的参数传递的主要思想是, 用例yaml的参数可以原样传递到接口请求里面, 这块其实非常重要的内容下面列举下主要的传参的场景:

```

1  1. 传参的路径:
2    用例yaml-->aw的yaml渲染-->接口的请求参数
3  2. 典型传递样例 (左边是用例里面编写的格式--yaml里面的格式, 右边是接口请求里面的内容--
   python字典的格式)
4    null -> None
5    "" -> ""
6    0 -> 0
7    '0' -> '0'
8    ' ' -> ' '
9    false -> False
10   "不传" -> 取默认值
11   参数不传递到aw yaml里面的话, 也是取默认值
12

```

支持上面的方式需要在aw里面写过滤器, 如下

```

1  billIndex: {{ billIndex|myDefault(11111) }} # 这个用法和{{ a or b }}分开用,
2                                             #新的建议使用myDefault, or不支持过
   滤器原样输出

```

1

低码、APAAS测试思路

低码测试需要结合页面，存在一个痛点就是页面覆盖是有限的，那么就必须结合接口来增加覆盖度，这样才能够更好的起到底层测试支撑上层业务的测试效果

1. 页面测试：主要覆盖业务基本流程，对于异常或者是业务层千奇百怪的使用，页面测试无法覆盖
2. 接口测试：接口测试刚好弥补页面测试无法覆盖到的内容，所以这里重点考虑接口测试带来的价值
 - 接口测试对正常流程，可以在页面正常流程覆盖基础上面，快速增加
 - 对于异常入参的覆盖需要思考，形成异常参数集合，后续对于异常用例会进一步自动化生成用例——这里有一个需要解决的问题就是异常接口的返回需要规范和结构化，这块可以在初步与开发形成共识后，通过测试驱动
 - 异常值有一个例外，对于边界值的异常覆盖需要收到添加
3. 后续效能的自动化用例会非常多，所以，需要按接口去统计绩效，数量会作为辅助参考内容，形成经验数据。

快速生成AW

说明

- 根据抓包的data的json，快速生产aw，比如下面是抓包的data
- 请求参数中如果存在列表嵌套字典，还支持自动生成for循环
- 只支持生产params里面的内容，url、method等其他用例信息都需要自己手动填写

```
1 抓包获取data数据：
2 JsDossier = {
3     "public": [{
4         "dossier": 1,
5         "account_key": "sm2"
6     }, {
7         "dossier": 2,
8         "account_key": "sm3"
9     }],
10    "logicContractId": None,
11    "logicContractType": 1,
12    "tag": "tag_070c5dda0953",
13    "billingDate": 0,
14    "progressPayment": {
15        "ratio": "-1",
16        "periodNum": 0,
17        "periodType": 0,
18        "periodTypeUnit": ""
19    },
```

```
20     "finalPayment": {
21         "ratio": "ratio_0c2b8d3b4d64",
22         "periodNum": 0,
23         "periodType": 0,
24         "periodTypeUnit": ""
25     },
26     "settlementPolicyExt": {
27         "monthDefine": {
28             "type": 0,
29             "typeUnit": "typeUnit_be80c4e5258b"
30         },
31         "paymentMode": {
32             "type": 0,
33             "typeUnit": "typeUnit_713f14a25d67"
34         },
35         "settlementMode": {
36             "type": 2,
37             "typeUnit": ""
38         }
39     },
40     "signedProjectStatus": 0,
41     "updateTime": "2024-05-18 09:52:24",
42     "bizType": "2",
43     "reconCycle": 0
44 }
```

代码

以项目里面的代码为最终代码

```
1 from jinja2 import Environment, Template
2
3 def generate_template_str(data, indent=0, path='', short_format=False,
4 key_counter=None):
5     """
6     递归生成模板字符串
7     :param data: 字典或列表数据
8     :param indent: 当前缩进级别
9     :param path: 当前访问路径, 用于模板变量引用
10    :param short_format: 是否使用短格式 (不带前缀路径)
11    :param key_counter: 用于记录键值出现次数的字典
12    :return: 模板字符串
13    """
14    template_lines = [] # 用于存储生成的模板行
15    indent_str = ' ' * indent # 根据当前缩进级别生成对应的缩进字符串
```



```

15
16     if key_counter is None:
17         key_counter = {} # 初始化键值计数器字典
18
19     if isinstance(data, dict): # 如果是字典类型
20         for key, value in data.items(): # 遍历字典的每个键值对
21             if isinstance(value, (dict, list)): # 如果值是字典或列表, 递归处理
22                 template_lines.append(f'{indent_str}{key}:')
23                 template_lines.extend(generate_template_str(value, indent + 1,
24 f'{path}.{key}' if path else key, short_format, key_counter))
25             else:
26                 short_key = key # 初始化短键名
27                 if short_format: # 如果启用短格式
28                     if key not in key_counter:
29                         key_counter[key] = 0 # 初始化计数器
30                     else:
31                         key_counter[key] += 1 # 增加计数器
32                     short_key = f'{key}{key_counter[key]}' # 更新键名, 添加
33 后缀数字
34                 var = f'{path}.{key}' if not short_format else short_key # 确定
35 模板变量使用的完整路径或短路径
36                 # 添加生成模板行
37                 template_lines.append(f'{indent_str}{key}: {{{{
38 {var}|myDefault({repr(value)}) }}}}')
39
40     elif isinstance(data, list): # 如果值是列表类型
41         template_lines.append(f'{indent_str}{{% for item in {path} %}}') # 添加
42  for循环开始行
43         template_lines.append(f'{indent_str}- ') # 列表项的缩进
44         if all(isinstance(i, dict) for i in data): # 检查列表中的所有项是否都是字
45 典
46             template_lines.extend(generate_template_str(data[0], indent + 1,
47 'item', short_format, key_counter))
48         # 添加for循环结束行
49         template_lines.append(f'{indent_str}{{% endfor %}}')
50
51     return template_lines # 返回生成的模板行列表
52
53 def main(data, root_name, short_format=False):
54     """
55     主函数, 生成模板并进行渲染
56     :param data: 输入数据
57     :param root_name: 根路径名称
58     :param short_format: 是否启用短格式
59     """
60     # 生成模板字符串

```

```

54     template_lines = generate_template_str(data, path=root_name,
short_format=short_format)
55     params_content = '\n'.join(template_lines) # 将模板行连接成字符串
56
57     # YAML模板字符串
58     yaml_template = f"""
59 ## -*- coding: utf-8 -*-
60 - case: 联营结算档案mq
61     epic: 账单中台
62     feature: 同步结算档案, 联营结算档案mq
63     story: 同步结算档案, 联营结算档案mq
64     severity: P0
65     tag: sm
66     variables:
67         sleep: {{{ sleep|myDefault(0) }}}
68     steps:
69         - name: 联营结算档案mq # Step1
70           key: kgl
71           jinja2_flag: 0
72           account_key: "{{$account_key or 'sm2'}}"
73           module: bill-center
74           extract:
75             api:
76               method: post
77               url: /billing-center/mock/bill/js/dossier/joinJsDossierConsumer
78             params:
79 {{{ params_content | indent(8, True) }}}
80             validate:
81               eq:
82                 status_code: {{{status_code or 200}}}
83                 errCode: "{{$errCode or 0}}"
84                 message: "{{$message or '请求成功'}}"
85                 data: {{{data or True}}}
86           setup:
87             sqls:
88               {% if delete == 'dossier' %}
89                 delete from billing_center.js_settle_dossier where order_code =
'{{$logicContractId or null}}';
90                 delete from billing_center.js_settle_dossier_change where order_code =
'{{$logicContractId or null}}';
91               {% endif %}
92           aw_teardown:
93             sqls:
94               {% if car_finish_date %}
95                 update billing_center.js_settle_dossier set car finish_date =
'{{$car_finish_date}}' where order_code = '{{$logicContractId}}';
96               {% endif %}

```

```
97 """
98
99 # 定义辅助函数
100 def myDefault(value, default):
101     # 用于在模板中返回默认值的自定义过滤器
102     return value if value is not None else default
103
104 # 渲染模板
105 env = Environment()
106 env.filters['myDefault'] = myDefault # 注册自定义过滤器
107 template = env.from_string(yaml_template) # 从字符串加载模板
108 output = template.render(params_content=params_content) # 渲染模板
109
110 # 打印结果
111 print(output)
112
113 if __name__ == '__main__':
114     # 示例数据
115     JsDossier = {
116         "public": [{
117             "dossier": 1,
118             "account_key": "sm2"
119         }, {
120             "dossier": 2,
121             "account_key": "sm3"
122         }],
123         "logicContractId": None,
124         "logicContractType": 1,
125         "tag": "tag_070c5dda0953",
126         "billingDate": 0,
127         "progressPayment": {
128             "ratio": "-1",
129             "periodNum": 0,
130             "periodType": 0,
131             "periodTypeUnit": ""
132         },
133         "finalPayment": {
134             "ratio": "ratio_0c2b8d3b4d64",
135             "periodNum": 0,
136             "periodType": 0,
137             "periodTypeUnit": ""
138         },
139         "settlementPolicyExt": {
140             "monthDefine": {
141                 "type": 0,
142                 "typeUnit": "typeUnit_be80c4e5258b"
143             },
```

```

144         "paymentMode": {
145             "type": 0,
146             "typeUnit": "typeUnit_713f14a25d67"
147         },
148         "settlementMode": {
149             "type": 2,
150             "typeUnit": ""
151         }
152     },
153     "signedProjectStatus": 0,
154     "updateTime": "2024-05-18 09:52:24",
155     "bizType": "2",
156     "reconCycle": 0
157 }
158
159 # 执行主函数，使用短格式
160 main(JsDossier, 'JsDossier', short_format=True)
161 # 执行主函数，使用完整格式
162 main(JsDossier, 'JsDossier')

```

用法

上面的代码支持2种格式的aw输出：

main(JsDossier, 'JsDossier', short_format=True) # 带前缀
main(JsDossier, 'JsDossier') # 不带前缀，默认

带前缀输出结果：

```

1  ## -*- coding: utf-8 -*-
2  - case: 联营结算档案mq
3    epic: 账单中台
4    feature: 同步结算档案, 联营结算档案mq
5    story: 同步结算档案, 联营结算档案mq
6    severity: P0
7    tag: sm
8    variables:
9      sleep: {{ sleep|myDefault(0) }}
10   steps:
11     - name: 联营结算档案mq # Step 1
12       key: kgl
13       jinja2_flag: 0
14       account_key: {{ account_key or 'sm2' }}
15       module: bill-center
16       extract:

```

```

17     api:
18         method: post
19         url: /billing-center/mock/bill/js/dossier/joinJsDossierConsumer
20     params:
21         public:
22             {% for item in JsDossier.public %}
23             -
24                 dossier: {{ item.dossier|myDefault(1) }}
25                 account_key: {{ item.account_key|myDefault('sm2') }}
26             {% endfor %}
27         logicContractId: {{ JsDossier.logicContractId|myDefault(None) }}
28         finalPayment:
29             ratio: {{
30 JsDossier.finalPayment.ratio|myDefault('ratio_0c2b8d3b4d64') }}
31             periodNum: {{ JsDossier.finalPayment.periodNum|myDefault(0) }}
32         settlementPolicyExt:
33             monthDefine:
34                 type: {{
35 JsDossier.settlementPolicyExt.monthDefine.type|myDefault(0) }}
36                 typeUnit: {{
37 JsDossier.settlementPolicyExt.monthDefine.typeUnit|myDefault('typeUnit_be80c4e5
38 258b') }}
39             paymentMode:
40                 type: {{
41 JsDossier.settlementPolicyExt.paymentMode.type|myDefault(0) }}
42                 typeUnit: {{
43 JsDossier.settlementPolicyExt.paymentMode.typeUnit|myDefault('typeUnit_713f14a2
44 5d67') }}
45         validate:
46             eq:
47                 status_code: {{ status_code or 200 }}
48                 errCode: {{ errCode or 0 }}
49                 message: {{ message or '请求成功' }}
50                 data: {{ data or True }}
51     setup:
52         sqls:
53             {% if delete == 'dossier' %}
54             delete from billing_center.js_settle_dossier where order_code = {{
55 logicContractId or null }};
56             delete from billing_center.js_settle_dossier_change where order_code =
57 {{ logicContractId or null }};
58             {% endif %}
59         aw_teardown:
60             sqls:
61                 {% if car_finish_date %}
62                 update billing_center.js_settle_dossier set car_finish_date = {{
63 car_finish_date }} where order code = {{ logicContractId }};

```

不带前缀的输出:

```
1  ## -*- coding: utf-8 -*-
2  - case: 联营结算档案mq
3  epic: 账单中台
4  feature: 同步结算档案, 联营结算档案mq
5  story: 同步结算档案, 联营结算档案mq
6  severity: P0
7  tag: sm
8  variables:
9      sleep: {{ sleep|myDefault(0) }}
10 steps:
11     - name: 联营结算档案mq # Step 1
12       key: kgl
13       jinja2_flag: 0
14       account_key: {{ account_key or 'sm2' }}
15       module: bill-center
16       extract:
17       api:
18         method: post
19         url: /billing-center/mock/bill/js/dossier/joinJsDossierConsumer
20       params:
21         public:
22           {% for item in public %}
23           -
24             dossier: {{ dossier|myDefault(1) }}
25             account_key: {{ account_key|myDefault('sm2') }}
26           {% endfor %}
27         logicContractId: {{ logicContractId|myDefault(None) }}
28         finalPayment:
29           ratio: {{ ratio|myDefault('ratio_0c2b8d3b4d64') }}
30           periodNum: {{ periodNum|myDefault(0) }}
31         settlementPolicyExt:
32           monthDefine:
33             type: {{ type|myDefault(0) }}
34             typeUnit: {{ typeUnit|myDefault('typeUnit_be80c4e5258b') }}
35           paymentMode:
36             type: {{ type1|myDefault(0) }}
37             typeUnit: {{ typeUnit1|myDefault('typeUnit_713f14a25d67') }}
38       validate:
39         eq:
40           status_code: {{ status_code or 200 }}
41           errCode: {{ errCode or 0 }}
```

```

42         message: {{ message or '请求成功' }}
43         data: {{ data or True }}
44     setup:
45         sqls:
46             {% if delete == 'dossier' %}
47             delete from billing_center.js_settle_dossier where order_code = {{
48             logicContractId or null }};
49             delete from billing_center.js_settle_dossier_change where order_code =
50             {{ logicContractId or null }};
51             {% endif %}
52         aw_tearardown:
53         sqls:
54             {% if car_finish_date %}
55             update billing_center.js_settle_dossier set car_finish_date = {{
56             car_finish_date }} where order code = {{ logicContractId }};
57             {% endif %}

```

注意事项

除了下面黄底的部分是根据入参自动化生成的不怎么需要修改，其他部分都是参考，需要大家修改成实际的值，包括url和方法等等

```

1  ## -*- coding: utf-8 -*-
2  - case: 联营结算档案mq
3  epic: 账单中台
4  feature: 同步结算档案, 联营结算档案mq
5  story: 同步结算档案, 联营结算档案mq
6  severity: P0
7  tag: sm
8  variables:
9  sleep: {{ sleep|myDefault(0) }}
10 steps:
11     - name: 联营结算档案mq # Step 1
12       key: kgl
13       jinja2_flag: 0
14       account_key: {{ account_key or 'sm2' }}
15       module: bill-center
16       extract:
17       api:
18           method: post
19           url: /billing-center/mock/bill/js/dossier/joinJsDossierConsumer
20       params:
21           public:
22               {% for item in public %}
23               -

```

```

24     dossier: {{ dossier|myDefault(1) }}
25     account_key: {{ account_key|myDefault('sm2') }}
26     {% endfor %}
27 logicContractId: {{ logicContractId|myDefault(None) }}
28 finalPayment:
29     ratio: {{ ratio|myDefault('ratio_0c2b8d3b4d64') }}
30     periodNum: {{ periodNum|myDefault(0) }}
31 settlementPolicyExt:
32     monthDefine:
33         type: {{ type|myDefault(0) }}
34         typeUnit: {{ typeUnit|myDefault('typeUnit_be80c4e5258b') }}
35     paymentMode:
36         type: {{ type1|myDefault(0) }}
37         typeUnit: {{ typeUnit1|myDefault('typeUnit_713f14a25d67') }}
38 validate:
39     eq:
40         status_code: {{ status_code or 200 }}
41         errCode: {{ errCode or 0 }}
42         message: {{ message or '请求成功' }}
43         data: {{ data or True }}
44 setup:
45     sqls:
46         {% if delete == 'dossier' %}
47         delete from billing_center.js_settle_dossier where order_code = {{
logicContractId or null }};
48         delete from billing_center.js_settle_dossier_change where order_code =
{{ logicContractId or null }};
49         {% endif %}
50 aw_tearardown:
51     sqls:
52         {% if car_finish_date %}
53         update billing_center.js_settle_dossier set car_finish_date = {{
car_finish_date }} where order code = {{ logicContractId }};
54         {% endif %}

```

FAQ

遇到的问题和经验大家共同维护补充

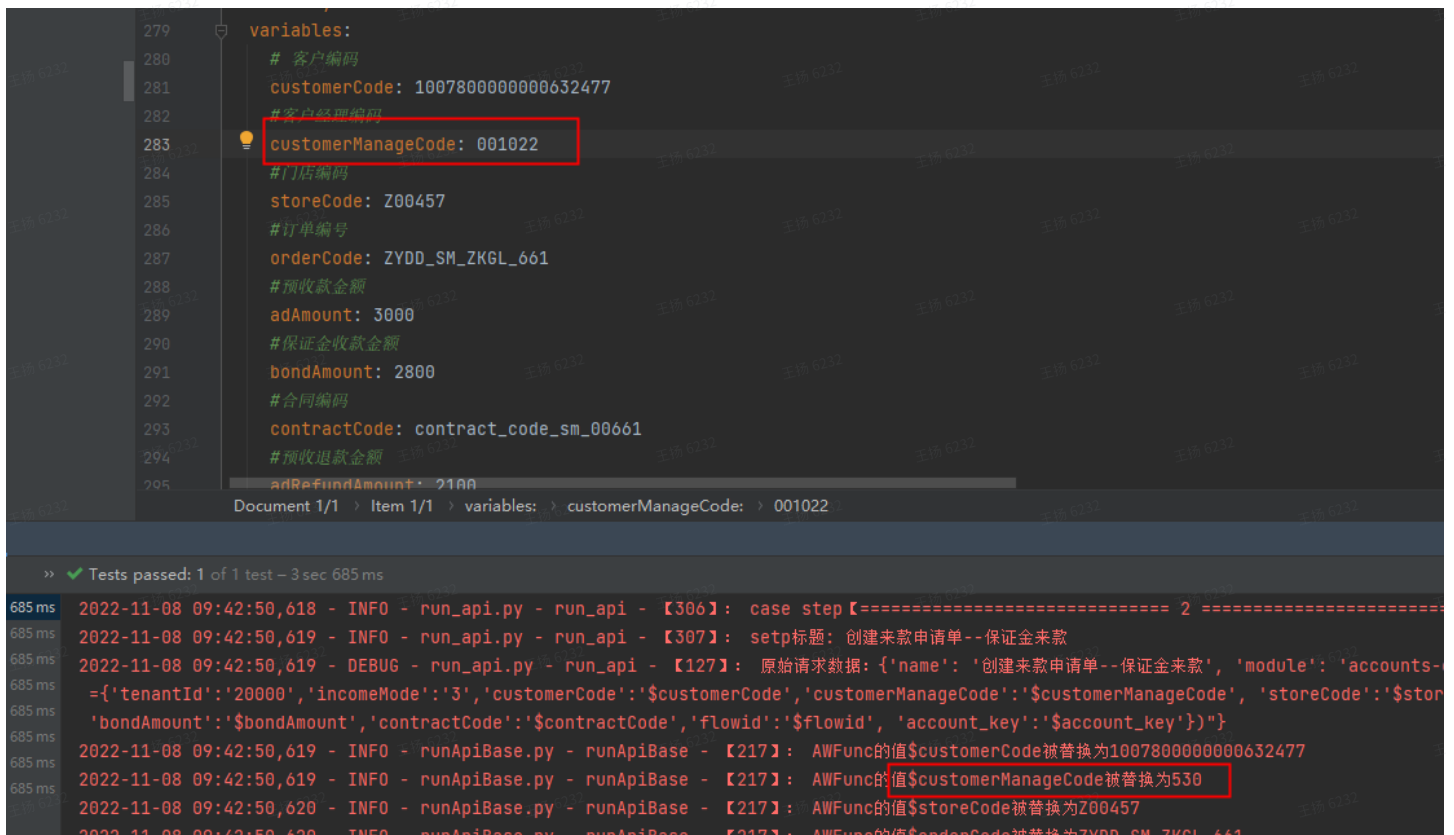
【问题描述】 当同一个用例yaml文件下存在>2条case，每条case第一步清除所有数据，case使用数据一致时，偶现 第一条case接口返回成功，但数据未落库成功，此时第二条case执行第一步清除数据时，未清除到第一条case执行后残留数据。

【解决方法】 可在第一条case加sleep等待 或 加sql查库表数据断言



【问题描述】用例/aw yaml文件中 全局变量 variables 中 变量值 为 0开头时数字，yaml解析出现问题。

【解决方法】变量值为数字，且0开头时，打上引号，解析准确



```
280 # 客户编码
281 customerCode: 1007800000000632477
282 # 客户经理编码
283 customerManageCode: '001022'
284 # 门店编码
285 storeCode: Z00457
286 # 订单编号
287 orderCode: ZYDD_SM_ZKGL_661
288 # 预收款金额
289 adAmount: 3000
290 # 保证金收款金额
291 bondAmount: 2800
292 # 合同编码
293 contractCode: contract_code_sm_00661
294 # 预收退款金额
295 adRefundAmount: 2100
Document 1/1 › Item 1/1 › variables: › adAmount: › 3000

center_sm.py
Tests passed: 1 of 1 test - 3 sec 746 ms
3 sec 746 ms 2022-11-08 09:43:43,085 - INFO - run_api.py - run_api - 【306】: case step【===== 2 =====】
3 sec 746 ms 2022-11-08 09:43:43,085 - INFO - run_api.py - run_api - 【307】: setp标题: 创建来款申请单--保证金来款
3 sec 746 ms 2022-11-08 09:43:43,085 - DEBUG - run_api.py - run_api - 【127】: 原始请求数据: {'name': '创建来款申请单--保证金来款', 'module': 'accounts-center', 'AWF
={ 'tenantId': '20000', 'incomeMode': '3', 'customerCode': '$customerCode', 'customerManageCode': '$customerManageCode', 'storeCode': '$storeCode', 'order
'bondAmount': '$bondAmount', 'contractCode': '$contractCode', 'flowId': '$flowId', 'account_key': '$account_key' }" }
2022-11-08 09:43:43,086 - INFO - runApiBase.py - runApiBase - 【217】: AWFunc的值$customerCode被替换为1007800000000632477
2022-11-08 09:43:43,086 - INFO - runApiBase.py - runApiBase - 【217】: AWFunc的值$customerManageCode被替换为001022
2022-11-08 09:43:43,086 - INFO - runApiBase.py - runApiBase - 【217】: AWFunc的值$storeCode被替换为Z00457
2022-11-08 09:43:43,086 - INFO - runApiBase.py - runApiBase - 【217】: AWFunc的值$orderCode被替换为ZYDD_SM_ZKGL_661
2022-11-08 09:43:43,087 - INFO - runApiBase.py - runApiBase - 【217】: AWFunc的值$bondAmount被替换为2800
```

【headers典型案例】案例

```
1 # -*- coding: utf-8 -*-
2 - case: 手机号修改密码
3 epic: 效能
4 feature: 手机号修改密码
5 story: 修改密码
6 severity: P0
7 tag: mr
8 variables:
9 phone: "13488325270" #手机号
10 newPassword: "123456" #新密码
11 verifyCode: "888888" #验证码
12 codeId: "111111" #验证码CodeId
13 sql: select id as isssdd ....
14 sql: select id ....
15 steps:
16 - name: 忘记密码验证码验证手机号
17   module: user-center
18   AWFunc: 忘记密码验证码验证手机号 (env,
19     params ={'phone': '$phone',
20     'verifyCode': '$verifyCode',
21     'codeId': '$codeId'})
22 - name: 手机号修改密码
23   module: user-center
24   AWFunc: 手机号修改密码 (env,
25     params ={'phone': '$phone',
26     'newPassword': '$newPassword',
27     'Set_Cookie': '$.step1.Set_Cookie'})
```

AW1 YAML

```
1 # -*- coding: utf-8 -*-
2 - case: 忘记密码验证码验证手机号
3   epic: 效能
4   feature: 手机号验证
5   story: 手机号验证
6   severity: P0
7   tag: mr
8   variables: # 测试数据生成
9     sleep: 0
10  steps:
11    - name: 忘记密码验证码验证手机号 #
12      key: gyl
13      jinja2_flag: 0
14      account_key: mr
15      module: user-center
16      extract:
17        'Set_Cookie': Set-Cookie # 变量不支持中杠 -
18      api:
19        method: post
20        url: /infrastructure-certificate/forget/password/verify/mobilePhone
21      params:
22        phone: "{{phone or null}}"
23        verifyCode: "{{verifyCode or null}}"
24        codeId: "{{codeId or null}}"
25      validate:
26        eq:
27          data: {{data or null}}
28          errCode: "{{errCode or 0}}"
29          message: {{message or '请求成功'}}
```

AW2 YMAL

```
1 # -*- coding: utf-8 -*-
2 - case: 手机号修改密码
3   epic: 效能
4   feature: 手机号修改密码
5   story: 修改密码
6   severity: P0
7   tag: mr
8   variables: # 测试数据生成
9     sleep: 0
```

```

10  steps:
11    - name: 手机号修改密码 #
12      key: gyl
13      jinja2_flag: 0
14      account_key: mr
15      module: user-center
16      headers: {'Cookie': '{{ Set_Cookie }}'}
17      extract:
18        #sql: select apply_code from invoice_center.invoice_apply_biz where
        biz_code = '{{order_code}}';
19      api:
20        method: post
21        url: /infrastructure-certificate/forget/password/phone/modify/pwd
22        params:
23          phone: "{{phone or null}}"
24          newPassword: "{{newPassword or null}}"
25        validate:
26          eq:
27            data: {{data or null}}
28            errCode: "{{errCode or 0}}"
29            message: {{message or '请求成功'}}

```

【sql断言常见问题：key后面缺少空格】

逾期结果是字典时，key后面缺少空格，导致解析的key不对

在yaml里面字典里面的key后面的冒号后面，必须有空格才被解析为字典{ 'a' :xx}

如果写成{a:xx}, 会被解析为 {'a:xx': None}, 实际想要的结果是{a: xx}

【false的参数传递案例】

```

feature: 核销中台
story: 是否存在应收数据
severity: P0
tag: wyf
variables:
  orderCode: cttorder2022110201 #订单号
  data: false
steps:
  - name: 自营计费入核销中台-查询下标 # setp2
    module: write-off-center
    AWFunc: 根据订单号查询分表下标及清理数据(env, params = {"orderCode": "$orderCode"})
  - name: 是否存在应收数据 # setp1
    module: write-off-center
    AWFunc: 是否存在应收数据(env, params = {"orderCode": "$orderCode",
      "data": $data
    })
teardown:

```

Document 1/1 > Item 1/1 > steps: > Item 1/1 > AWFunc: > 是否存在应收数据(env, params ...

✓ Tests passed: 1 of 1 test - 356 ms

```

2023-05-06 14:43:18,086 - INFO - run_api.py - run_api - 【399】: 历史生成测试数据: {'orderCode': 'cttorder2022110201', 'data': False}
2023-05-06 14:43:18,086 - INFO - run_api.py - run_api - 【419】: 【---1---】引用用例setp标题: 是否存在应收数据
2023-05-06 14:43:18,086 - INFO - run_api.py - run_api - 【170】: 替换变量之后的data: {'orderCode': 'cttorder2022110201'}
2023-05-06 14:43:18,087 - INFO - run_api.py - run_api - 【197】: 请求的method: get
2023-05-06 14:43:18,087 - INFO - run_api.py - run_api - 【198】: 请求的url: /write-off-center/writeoff/existReceivable
2023-05-06 14:43:18,087 - INFO - run_api.py - run_api - 【203】: 发送请求的data: {'orderCode': 'cttorder2022110201'}
2023-05-06 14:43:18,426 - INFO - run_api.py - run_api - 【236】: resp: {'status_code': 200, 'errCode': '0', 'message': '请求成功', 'data': {
2023-05-06 14:43:18,426 - INFO - run_api.py - run_api - 【103】: 当前步骤是1, 提取step参数: {'step1': {}}
2023-05-06 14:43:18,426 - INFO - run_api.py - run_api - 【103】: 当前步骤是1, 提取step参数: {'step1': {}}
2023-05-06 14:43:18,427 - INFO - assertUtil.py - assertUtil - 【47】: eq: status_code actual value is 200
2023-05-06 14:43:18,427 - INFO - assertUtil.py - assertUtil - 【48】: eq: status_code expected value is 200
2023-05-06 14:43:18,427 - INFO - assertUtil.py - assertUtil - 【47】: eq: errCode actual value is 0
2023-05-06 14:43:18,427 - INFO - assertUtil.py - assertUtil - 【48】: eq: errCode expected value is 0
2023-05-06 14:43:18,427 - INFO - assertUtil.py - assertUtil - 【47】: eq: message actual value is 请求成功
2023-05-06 14:43:18,427 - INFO - assertUtil.py - assertUtil - 【48】: eq: message expected value is 请求成功
2023-05-06 14:43:18,427 - INFO - assertUtil.py - assertUtil - 【47】: eq: data actual value is False
2023-05-06 14:43:18,428 - INFO - assertUtil.py - assertUtil - 【48】: eq: data expected value is False
2023-05-06 14:43:18,428 - INFO - run_api.py - run_api - 【447】: >> [End...

```

【自动审批业务单据】

在发起审批流的接口，可以把审批的hook加到aw_teardown下面：

```

1  ## -*- coding: utf-8 -*-
2  - case: 联营对客账单回签
3    epic: 账单中台
4    feature: 回签
5    story: 联营对客账单回签
6    severity: P0
7    tag: sm
8  # variables:
9    steps:
10     - name: 联营对客账单回签 # setp1
11       key: finance
12       jinja2_flag: 0
13       account_key: "{{account_key or 'sm2'}}"
14       module: bill-center
15       extract:
16         token: token

```

```

17     api:
18         method: post
19         url: /billing-center/v1/object/410030
20         params: {
21             "layout": "signBack",
22             "data": {
23                 "code": {{ code|myDefault(0) }},
24                 "backAmount": {{ backAmount|myDefault(0) }},
25                 "backFile": "[{"relativePath": "public/img/H01/2024/04/24/feba529f2f04fcea53c463a15fcaa3f.png", "absolutePath": "https://oss-sit.kaigongla.cn/public/img/H01/2024/04/24/feba529f2f04fcea53c463a15fcaa3f.png", "name": "2.png"}]",
26                 "__typename": "jsBill",
27                 "includeOldShouldAmount": "0",
28                 "backRemark": "",
29                 "backNotIncomeAmount": 0,
30                 "billAmount": {{ billAmount|myDefault(0) }},
31                 "deleted": {{ deleted|myDefault(none) }},
32                 "busLine": "2",
33                 "name": {{ name|myDefault(none) }},
34                 "backDisputeFlag": "0",
35                 "id": {{ id|myDefault(none) }},
36                 "text": {{ text|myDefault(none) }},
37                 "oldShouldAmount": {{ oldShouldAmount|myDefault(none) }},
38                 "workflowId": {{ workflowId|myDefault(none) }},
39                 "objectId": {{ objectId|myDefault(none) }}
40             },
41             "endpoint": 1,
42             "scene": "signBack"
43         }
44     validate:
45         eq:
46             status_code: {{status_code or 200}}
47             errCode: "{{errCode or 0}}"
48             message: "{{message or '请求成功'}}"
49     aw_teardown:
50         {% if signature %}
51         funs:
52             - ${approval_pass("${code}", "${Approver}", "$token")}
53         {% endif %}

```

然后通过传递参数去调用他即可

```
1 - name: 联营对客账单回签 # setp1
```

```

2  module: bill-center
3  AWFunc: 联营对客账单回签
4  aw_params: {
5      "code": '$.step1.billCode',    # 要审批的业务单号, 比如账单号, 订单号等
6      "backAmount": '$.step1.billAmount',
7      "billAmount": '$.step1.billAmount',
8      "signature": '1',    # 开关, 审批就传一个1, 不审批就会忽略
9      "Approver": 'Testing_Department',    # 审批人, 随便写一个组织结构里面存在的中文名字即可
10     'account_key': '$account_key'
11 }

```

【提取结果中，存在类json的字符串，方便使用，可以设置转译为json的开关】

```

1  - case: 押金创建预支付单
2    epic: 业财
3    feature: 支付中台
4    story: 创建预支付单
5    severity: P0
6    tag: lxt
7    variables: # 测试数据生成
8    steps:
9      - name: 押金创建预支付单 #
10        key: kgl
11        jinja2_flag: 0
12        account_key: lxt
13        module: payment-center|
14        extract: # 数据提取, 这里可以提取接口返回的payOrderCode
15          payOrderCode: data.payOrderCode|
16        api:
17          method: post
18          url: /payment-center/v1/api/pay/createPrePayOrder
19          params:
20            paymentType: {{paymentType or 1}} # 支付渠道 1、微信 2、
21            paymentScene: {{paymentScene or 1}} # 支付场景 1、app
22            actualAmount: {{pay_amount}} # 实际支付金额, 目前与账单金

```

则response结果假设是这样的：

```

1  {
2    "platformOrderNo": "1816109407391563791",
3    "useAddress": "{ \"adCode\": \"320114\", \"address\": \"江苏省南京市雨花台区南京南
  站\", \"city\": \"南京市\", \"cityAdCode\": \"320100\", \"district\": \"雨花台区
  \", \"latitude\": \"31.968750\", \"locationDetail\": \"南京南站

```



```
\",\\"longitude\\":\\"118.798039\\",\\"province\\":\\"江苏省\\",\\"provinceCode\\":\\"320000\\"}",  
4   "projectAddress": "{\\"adCode\\":\\"320114\\",\\"address\\":\\"江苏省南京市雨花台区南京南站\\",\\"city\\":\\"南京市\\",\\"cityAdCode\\":\\"320100\\",\\"district\\":\\"雨花台区\\",\\"latitude\\":\\"31.968750\\",\\"locationDetail\\":\\"南京南站\\",\\"longitude\\":\\"118.798039\\",\\"province\\":\\"江苏省\\",\\"provinceCode\\":\\"320000\\"}",  
5   "projectName": "自动化测试7220841702484279296",  
6   "projectId": "1264755540771008512"  
7 }
```

假设你提取的是上面的全部内容，到data里面，加了|后，useAddress，projectAddress后面的就不是字符串，而是json格式了