案例回顾

增加课程

```yaml
config:
  name: 添加课程
  verify: false
  base_url: http://120.55.190.222:7080
  export:
    - course_id
    - cookie

teststeps:
-
  name: 登录
  testcase: testcase/login.yml

-
  name: 添加课程
  request:
    method: POST
    url: /api/mgr/sq_mgr/
    data:
      action: add_course
      data: '{ "name": "初中化学", "desc": "初中化学课程", "display_idx": "1"}'
    cookies:
      sessionid: ${cookie}
  extract:
    course_id: body.id
  validate:
    - eq: ['status_code',200,]
    - eq: ['body.retcode',0]
```


修改课程

```yaml
config:
  name: 修改课程
  verify: false
  base_url: http://120.55.190.222:7080
  export:
    - course_id
    - cookie

teststeps:
  - name: 添加课程
    testcase: testcase/add_course.yml

  - name: 修改课程
    request:
      method: PUT
```

```
      url: /api/mgr/sq_mgr/
      data:
        action: modify_course
        id: ${course_id}
        newdata: '{"name": "初中计算机", "desc": "初中计算机课程", "display_idx":
"1"}'
      cookies:
        sessionid: ${cookie}
    validate:
      - eq: [ 'status_code',200, ]
      - eq: [ 'body.retcode',0 ]
```

删除课程

```
config:
  name: 删除课程
  verify: false
  base_url: http://120.55.190.222:7080

teststeps:
  - name: 添加并修改课程
    testcase: testcase/update_suite.yml

  - name: 删除课程
    request:
      method: DELETE
      url: /api/mgr/sq_mgr/
      data:
        action: delete_course
        id: $course_id
      cookies:
        sessionid: ${cookie}
    validate:
      - eq: [ 'status_code',200, ]
      - eq: [ 'body.retcode',0 ]
```


用例变量

httprunner中，可以用变量替换测试步骤中的某些值。

变量按照类型划分有：配置变量(config variables) 环境变量（env variables）和 参数变量(parameter variables)

按照作用域划分有：测试步骤变量(step variables) 和 全局配置变量(config variables)，步骤优先级大于用例优先级

配置变量(config variables)

在config中定义的变量为用例全局变量作用域为当前用例，

```
config:
  name: 登录测试用例
  verify: false
  base_url: http://120.55.190.222:7080
  variables:
    username: auto        #定义变量
    password: sdfsdfsdf
teststeps:
-
  name: 用户名密码登录
  request:
    url: /api/mgr/loginReq
    data:
      username: $username     #引用变量   也可以写成${变量名}
      password: $password
    method: POST
  extract:  #提取当前步骤变量
    cookie: cookies.sessionid
  validate:
    -  eq:
         - status_code
         - 200
    -  eq:
         - body.retcode
         - 0
```

如果变量需要从其他数据源读取，那么可以在 `debugtalk.py` 中定义函数返回键值对变量

```
def login_variables():
    return {'user':'auto','psw':'sdfsdfsdf',"code":0}
```

```
config:
  name: 登录测试用例
  verify: false
  base_url: http://120.55.190.222:7080
  variables: ${login_variables()}

teststeps:
-
  name: 用户名密码登录
  request:
    url: /api/mgr/loginReq
    data:
      username: $user  #引用变量   也可以写成${变量名}
      password: $psw
    method: POST
  extract:  #提取当前步骤变量
    cookie: cookies.sessionid
  validate:
    -  eq:
         - status_code
```

```
        - 200
    -  eq:
        - body.retcode
        - $code
```

测试步骤变量 (step variables)

teststeps中定义的则为当前步骤变量，作用域仅限当前步骤

```
config:
  name: 登录测试用例
  verify: false
  base_url: http://120.55.190.222:7080
  variables: ${login_variables()}
#  parameters:  #数据驱动参数
#    username-password-code: ${get_login_data()}
teststeps:
-
  name: 用户名密码登录
  variables:
    user: auto
    psw: sdfsdfsdf666
  request:
    url: /api/mgr/loginReq
    data:
      username: $user  #引用变量  也可以写成${变量名}
      password: $psw
    method: POST
  extract:  #提取当前步骤变量
    cookie: cookies.sessionid
  validate:
    -  eq:
        - status_code
        - 200
    -  eq:
        - body.retcode
        - $code
```


如果步骤中的变量名和用例变量名相同，步骤变量优先级高于用例变量。

步骤变量不支持函数返回值模式 即不支持 `variables: ${login_variables()}` 这种形式

环境变量（env variables)

对于1些环境数据，平时不方便卸载用例中进行维护的，我们可以放在环境变量文件中，方式是保存在项目根目录下的.env文件（如果文件没有就创建，文件名就是`.env`不要忘记点）

定义环境变量.env

```
user=auto
psw=sdfsdfsdf
```

使用环境变量

```
teststeps:
-
  name: 用户名密码登录
  variables:
    user: ${ENV(user)}
    psw:  ${ENV(psw)}
```

参数化（数据驱动）

参数变量(parameter variables)

参数变量的用途就是作为数据驱动被使用

独立参数&直接指定参数列表

```
config:
  name: 登录测试用例
  verify: false
  base_url: http://120.55.190.222:7080
  parameters:
    username:
    - auto
    - auto
    password:
    - sdfsdfsdf
    - sdfsdfsdf
teststeps:
-
  name: 用户名密码登录
  request:
    url: /api/mgr/loginReq
    data:
      username: $username  #引用变量  也可以写成${变量名}
      password: $password
    method: POST
  extract:  #提取当前步骤变量
    cookie: cookies.sessionid
  validate:
    -  eq:
         - status_code
         - 200
    -  eq:
         - body.retcode
         - 0
```


实际的参数数量等于 username的参数个数*password参数的个数，即笛卡尔积

关联参数&直接指定参数列表

如上例，参数之间存在关联（用户名和密码），应该以关联方式传递参数

```
config:
  name: 登录测试用例
  verify: false
  base_url: http://120.55.190.222:7080
  parameters:
    username-password:
    - ['auto','sdfsdfsdf']
    - ['auto','sdfsdfsdf']
teststeps:
-
  name: 用户名密码登录
  request:
    url: /api/mgr/loginReq
    data:
      username: $username  #引用变量  也可以写成${变量名}
      password: $password
    method: POST
  extract:  #提取当前步骤变量
    cookie: cookies.sessionid
  validate:
    -  eq:
         - status_code
         - 200
    -  eq:
         - body.retcode
         - 0
-
  name: 列出课程
  request:
    cookies:
      sessionid: $cookie
    url: /api/mgr/sq_mgr/
    params:
      action: list_course
      pagenum: 1
      pagesize: 20
    method: GET
  extract:
    id: body.retlist[0].id
  validate:
      - eq:
          - status_code
          - 200
      - eq:
          - body.retcode
          - 0
```

关联参数 & 引用自定义函数

对于没有现成参数列表，或者需要更灵活的方式动态生成参数的情况，可以通过在 debugtalk.py 中自定义函数生成参数列表，并在 YAML/JSON 引用自定义函数的方式。

编写获取参数的函数，如：

```
# debugtalk.py
def get_login_data():
    return [
        {'username':'auto','password':'sdfsdfsdf',"code":0},
        {'username': 'auto', 'password': 'sdfsdfsdf1', "code": 1},
        {'username': 'auto1', 'password': 'sdfsdfsdf', "code": 1},
    ]
```

返回参数的格式为`[{key:value},{key2:value}...]`

参数变量(parameter variables) 部分修改为，参数名1-参数名2-参数名3: ${get_login_datat()} , 可以理解为${函数返回值}

```
config:
  name: 登录测试用例
  verify: false
  base_url: http://120.55.190.222:7080
#  variables: ${login_variables}
  parameters:  #数据驱动参数
    username-password-code: ${get_login_data()}
teststeps:
-
  name: 用户名密码登录
  request:
    url: /api/mgr/loginReq
    data:
      username: $username  #引用变量  也可以写成${变量名}
      password: $password
    method: POST
  extract:  #提取当前步骤变量
    cookie: cookies.sessionid
  validate:
    -  eq:
         - status_code
         - 200
    -  eq:
         - body.retcode
         - $code
```


初始化与清除机制

在hr中实现初始化与清除机制的方式是采用hook 机制，可以在请求前和请求后调用钩子函数。

目前HR3只支持步骤层面的hook,用例层面的未实现。

步骤层面

1.定义钩子函数

```
# debugtalk.py

def hook_setup():
    with open('setup.txt','w',encoding='utf-8') as f:
        f.write('执行初始化')
def hook_teardown():
    with open('teardown.txt', 'w', encoding='utf-8') as f:
        f.write('执行清除')
```

2.测试步骤引用钩子函数

```
config:
  name: 登录测试用例
  verify: false
  base_url: http://120.55.190.222:7080
  variables: ${login_variables()}
teststeps:
-
  name: 用户名密码登录
  setup_hooks:
    - ${hook_setup()}
  teardown_hooks:
    - ${hook_teardown()}
  request:
    url: /api/mgr/loginReq
    data:
      username: $user  #引用变量  也可以写成${变量名}
      password: $psw
    method: POST
  extract:  #提取当前步骤变量
    cookie: cookies.sessionid
  validate:
    -  eq:
         - status_code
         - 200
    -  eq:
         - body.retcode
         - 0
```


实战案例

debugtalk.py

```
def delete_course(_id,sessionid):
    url = 'http://120.55.190.222:7080/api/mgr/sq_mgr/'
    payload = {
        "action": "delete_course",
        "id": _id
    }
    cookies={
        'sessionid': sessionid
    }
    resp = requests.delete(url,data=payload,cookies=cookies)
    return resp.json()
```

update_course.yml

```
config:
  name: 修改课程
  verify: false
  base_url: http://120.55.190.222:7080
  export:
    - course_id
    - cookie

teststeps:
  -
    name: 添加课程
    testcase: testcase/add_course.yml
  -
    name: 修改课程
    request:
      method: PUT
      url: /api/mgr/sq_mgr/
      data:
        action: modify_course
        id: ${course_id}
        newdata: '{"name": "初中计算机", "desc": "初中计算机课程", "display_idx":
"1"}'
      cookies:
        sessionid: ${cookie}
    validate:
      - eq: [ 'status_code',200, ]
      - eq: [ 'body.retcode',0 ]

    teardown_hooks:
      - ${delete_course($course_id,$cookie)}
```

add_course.yml

```
config:
  name: 添加课程
  verify: false
  base_url: http://120.55.190.222:7080
  export:
    - course_id
    - cookie

teststeps:
```

```
-
  name: 登录
  testcase: testcase/login.yml

-
  name: 添加课程
  request:
    method: POST
    url: /api/mgr/sq_mgr/
    data:
      action: add_course
      data: '{ "name": "初中化学", "desc": "初中化学课程", "display_idx": "1"}'
    cookies:
      sessionid: ${cookie}
  extract:
    course_id: body.id
  validate:
    - eq: ['status_code',200,]
    - eq: ['body.retcode',0]
```

用例层面

hr3技术层面未支持用例hook，我们可以通过变通的方式实现初始化与清除

比如将初始化清除的步骤写进测试用例，在测试步骤中引用测试用例即可

初始化用例文件 setup.yaml

```
config:
  name: 初始化
  verify: false
  base_url: http://120.55.190.222:7080
  variables: ${login_variables()}
  export:
    - cookie
teststeps:
-
  name: 登录
  request:
    url: /api/mgr/loginReq
    data:
      username: $user  #引用变量  也可以写成${变量名}
      password: $psw
    method: POST
  extract:  #提取当前步骤变量
    cookie: cookies.sessionid
```

清除用例文件 teardown.yml

```
config:
  name: 清除
  verify: false
  base_url: http://120.55.190.222:7080
teststeps:
```

```
-
  name: 登出
  request:
    url: /api/mgr/logoutreq
    method: GET
  validate:
    - eq:
        - status_code
        - 200
    - eq:
        - body.retcode
        - 0
```

测试用例部分 demo.yml

```
config:
  name: 列出课程测试
  verify: false
  base_url: http://120.55.190.222:7080
teststeps:
-
  name: 初始化
  testcase: testcases/setup.yml
-
  name: 列出课程
  request:
    cookies:
      sessionid: $cookie
    url: /api/mgr/sq_mgr/
    params:
      action: list_course
      pagenum: 1
      pagesize: 20
    method: GET
  extract:
    id: body.retlist[0].id
  validate:
      - eq:
          - status_code
          - 200
      - eq:
          - body.retcode
          - 0
-
  name: 清除
  testcase: testcases/teardown.yml
```