pytest_server设计文档及接入指南

1 项目背景

- 项目目的

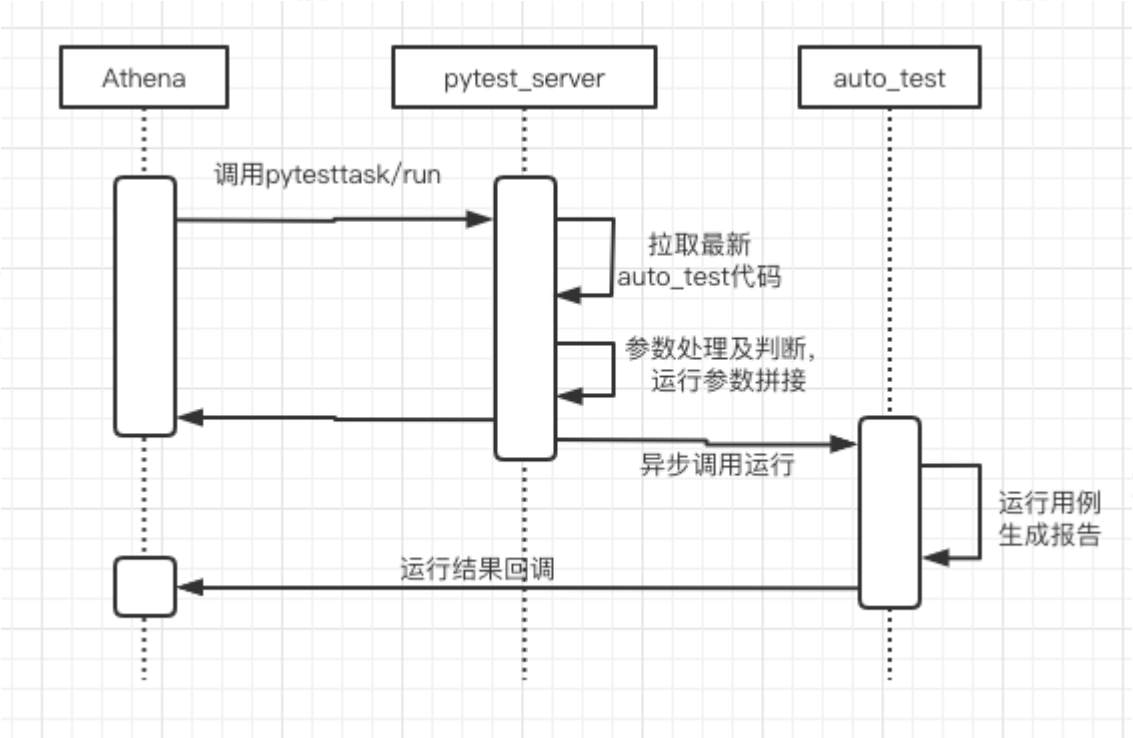
1. sls多个team正在使用auto_test代码仓库做自动化测试，积累了上万case。之前使用独立的jenkins(http://10.143.204.54:8080/)去做测试任务管理及报告展示。遗憾的是，独立的jenkins不能很好地契合当前SSC的CI / CD流程。
2. 关于自动化测试如何在CI/CD中发挥作用，目前通用的做法是由DMS触发Athena的自动化任务，在业务项目构建后立即触发相应的测试，并计算自动化测试的覆盖率。
3. auto_test如果要接入到Athena中，需要做一点改造。改造的方案有2种，1种是利用jenkins套娃，另1种是自建一个服务，就是pytest_server。经过一系列讨论，最终确认用自建服务的方式

- 文档编写目的

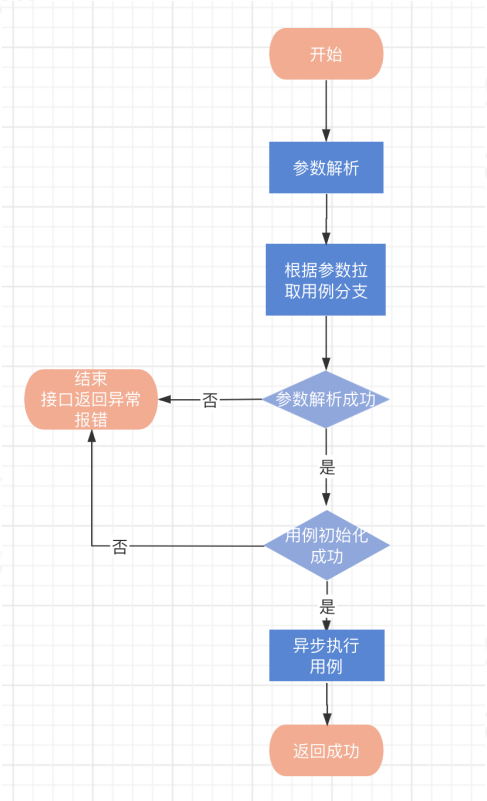
1. 做这个项目的调研的时候，使用了一些pytest的高级用法，甚至改了pytest插件去实现项目目的。这些需要提出来让后续维护者或使用者避坑
2. 部署及运行流程和公司的通用项目不太一致，在本文档中标出
3. 一些写死的规则 及 使用的注意事项也写在本文档中标出
4. 第4，5部分是服务维护指引 及 业务项目组接入指南

2 时序图

时序图



pytest_server：pytesttask/run流程图：



3 详细设计

3.1 接口设计

1. 任务运行接口：

pytest_server提供case运行接口：/taskmanage/pytesttask/run

API描述	Case 任务执行
调用方式	HTTP(POST)，application/json

请求	{
	"project_id":1,//项目id
	"batch_id":"xxx",//batch_id
	"task_id":"xxx",//task_id
	"env":"",//环境, test,staging,uat,fte,小写
	"pytest_cmd":["-s","-v",""],//pytest的运行命令
	"fte_info":{
	"lls_api_grpc_url": "lcos-grpc-test-sg.grpc-gateway-sg2-test.devops-sz.i.shopee.io:80",
	"lcos_api_http_url": "http://localhost:80",
	"lls_api_http_url": "http://localhost:80"
	},
	"cid":"sg",//小写，不传默认sg
	"branch_name":"test"
	}

响应	{
	"retcode":0,//0：成功，1：失败
	"message":"success"//返回信息，成功时**success，失败时失败原因。**
	}

Pytest方收到任务执行请求。命令校验是一个同步过程，命令执行是一个异步的过程。目前FTE还用不起来

2．回调接口

平台提供case执行结果录入接口：/casemanage/autocase/pytest-report

API描述	case执行报告录入
调用方式	HTTP(POST)，application/json

请求	{
	"project_id":1,//项目id
	"batch_id":"xxx",//batch_id
	"task_id":"xxx",//task_id
	"env":"",//环境
	"result":"",//执行结果,成功success, 失败：具体描述
	"data":{
	"passed":1,//成功
	"failed":1,//失败
	"broken":1,//阻塞？
	"skipped":1,//跳过
	"unknown":1,//未知
	"link":"http://xxx"//报告链接
	}
	}

响应	{
	"retcode":0,//0：成功，1：失败
	"message":"success"//返回信息，成功时**success，失败时失败原因。**
	}

3.2 自动化用例改造设计

1) pytest-metadata插件改造

详细的插件代码见： https://github.com/pytest-dev/pytest-metadata

该插件的用途为设置被测项目的元数据并展示在报告文件中。引入这个插件是为了解决环境变量无法在运行命令动态传入auto_test。从上面的接口文档可以看出：

project_id 、batch_id、task_id 等字段，在结果回传的时候，要传给 Athena。另外 env 和 cid(即 country) 需要在跑之前设置为环境变量（参考jenkins设置）。

环境变量的设置，要在自动化脚本运行之前。因此在auto_test 的任意一个代码行去设置 都来不及，只能通过这种外部插件，在test collection 及 test session 启动之前，就将环境变量设置起来。

优雅一点的做法：是自己开发一个类metadata的插件，然后安装。以下做法是偷懒做法：

pip 安装 pytest-metadata 最新版本后，去修改 site-packages/pytest_metadata/plugin.py的代码，增加如下代码行：

（由于是通过直接改依赖包的代码方式，建议不要随便升级服务器上的这个插件，代码会被覆盖掉）

```
@pytest.hookimpl(tryfirst=True)
def pytest_configure(config):
    config._metadata = {
        "Python": platform.python_version(),
        "Platform": platform.platform(),
        "Packages": {
            "pytest": pytest.__version__,
            "py": py.__version__,
            "pluggy": pluggy.__version__,
        },
    }
    config._metadata.update({k: v for k, v in config.getoption("metadata")})
    config._metadata.update(json.loads(config.getoption("metadata_from_json")))
        # ----------  ---------------------------------------------
    for k, v in config.getoption("metadata"):
        os.environ[k]=v
    for k, v in json.loads(config.getoption("metadata_from_json")).items():
        os.environ[k]=v
        # ---------------------------------------------------------------------
    plugins = dict()
    for plugin, dist in config.pluginmanager.list_plugin_distinfo():
        name, version = dist.project_name, dist.version
        if name.startswith("pytest-"):
            name = name[7:]
        plugins[name] = version
    config._metadata["Plugins"] = plugins

    for provider in CONTINUOUS_INTEGRATION:
        [
            config._metadata.update({var: os.environ.get(var)})
            for var in provider
            if os.environ.get(var)
        ]

    if hasattr(config, "workeroutput"):
        config.workeroutput["metadata"] = config._metadata

    config.hook.pytest_metadata(metadata=config._metadata)
```

PS: 插件的改造是一次性的，改造完成，服务起来后，使用者不需要关注。服务维护者要知道这个坑点。

2) conftest改造

conftest.py 改造的核心点1：在auto_test的子项目目录中添加 conftest.py 文件。如 lcs/conftest.py lls/conftest.py regression/conftest.py

conftest.py 改造的核心点2：下面代码行的前3行， 更改 auto_test/ 目录为 Python 运行目录。

conftest.py 改造的核心点3：所有case运行完成后回调Athena ，回传测试结果 。 在 pytest_terminal_summary中完成，该插件是在case运行完成后才会执行，适合用来做发送 seatalk 及回传报告这种操作

以上3点，缺一不可

```
import os
import sys
sys.path.append((os.path.abspath(os.path.join(os.path.dirname(__file__), '../'))))
import time
from freight_recon.libs.http_api.publicBaseApi import PublicBaseApi
from freight_recon.basic_services_settings import TEST_ENV, IS_SEND_SEATALK, TEST_REPORT_URL, JOB_NAME


def pytest_terminal_summary(terminalreporter, exitstatus, config):
    '''''''

    content = '[]:' + JOB_NAME + '\n'
    content = content + "" + os.getenv("env", "test") + "\n"
```

```
    total = terminalreporter._numcollected
    passed = len([i for i in terminalreporter.stats.get('passed', []) if i.when != 'teardown'])
    content = content + "passed:" + str(passed) + "\n"
    failed = len([i for i in terminalreporter.stats.get('failed', []) if i.when != 'teardown'])
    content = content + "failed:" + str(failed) + "\n"
    error = len([i for i in terminalreporter.stats.get('error', []) if i.when != 'teardown'])
    broken = len([i for i in terminalreporter.stats.get('broken', []) if i.when != 'teardown'])
    content = content + "error:" + str(error) + "\n"
    rerun = len([i for i in terminalreporter.stats.get('rerun', []) if i.when != 'teardown'])
    content = content + "rerun:" + str(rerun) + "\n"
    skipped = len([i for i in terminalreporter.stats.get('skipped', []) if i.when != 'teardown'])
    content = content + "skipped:" + str(skipped) + "\n"
    unknown = len([i for i in terminalreporter.stats.get('unknown', []) if i.when != 'teardown'])

    selected = passed + skipped + error + failed
    content = content + "selected:" + str(selected) + "\n"
    if selected > 0:
        success_percent = passed / selected * 100
        content = content + "%.2f" % success_percent + "%\n"
    else:
        success_percent = 0
        content = content + "%.2f" % success_percent + "%\n"

    # terminalreporter._sessionstarttime
    duration = (time.time() - terminalreporter._sessionstarttime)/60
    content = content + 'total times:%.2f' % duration + ' \n'
    content = content + ":" + TEST_REPORT_URL + "\n"
    # print("content" + content)

    end_flag = False
    #
    if selected == total and total > 0:
        end_flag = True
    # , masterflag
    if total == 0 and selected > 0:
        end_flag = True

    if IS_SEND_SEATALK.lower() == 'true' and selected > 0 and end_flag:
        data = init_push_webhook_data(content)
        data2 = init_push_webhook_data2(content)
        push_webhook_seatalk('HoZlmGTwTrKp9_1Cr5AgHQ', data)        # finance
        # push_webhook_seatalk('ilzj5dIQRJOiWSklPXfQfg', data2)

    if os.getenv("project_id") and selected > 0 and end_flag:
        if config.getoption("htmlpath"):
            htmlpath = "report" + config.getoption("htmlpath").split('report')[-1]
        else:
            htmlpath = "report/"
        result = {"result_flag":"Success.", "passed": passed, "failed": failed, "error": error, "broken":
broken,
                  "rerun": rerun, "skipped": skipped, "unknown": unknown}
        other_data = {
            "project_id": os.getenv("project_id", ""),
            "batch_id": os.getenv("batch_id", ""),
            "task_id": os.getenv("task_id", ""),
            "report_path": htmlpath
        }
        callback_data = init_callback_data(other_data, TEST_ENV, result)
        report_callback_athena(callback_data)
```

3.3 报告存储及展示设计

报告选型：Allure / pytest-html

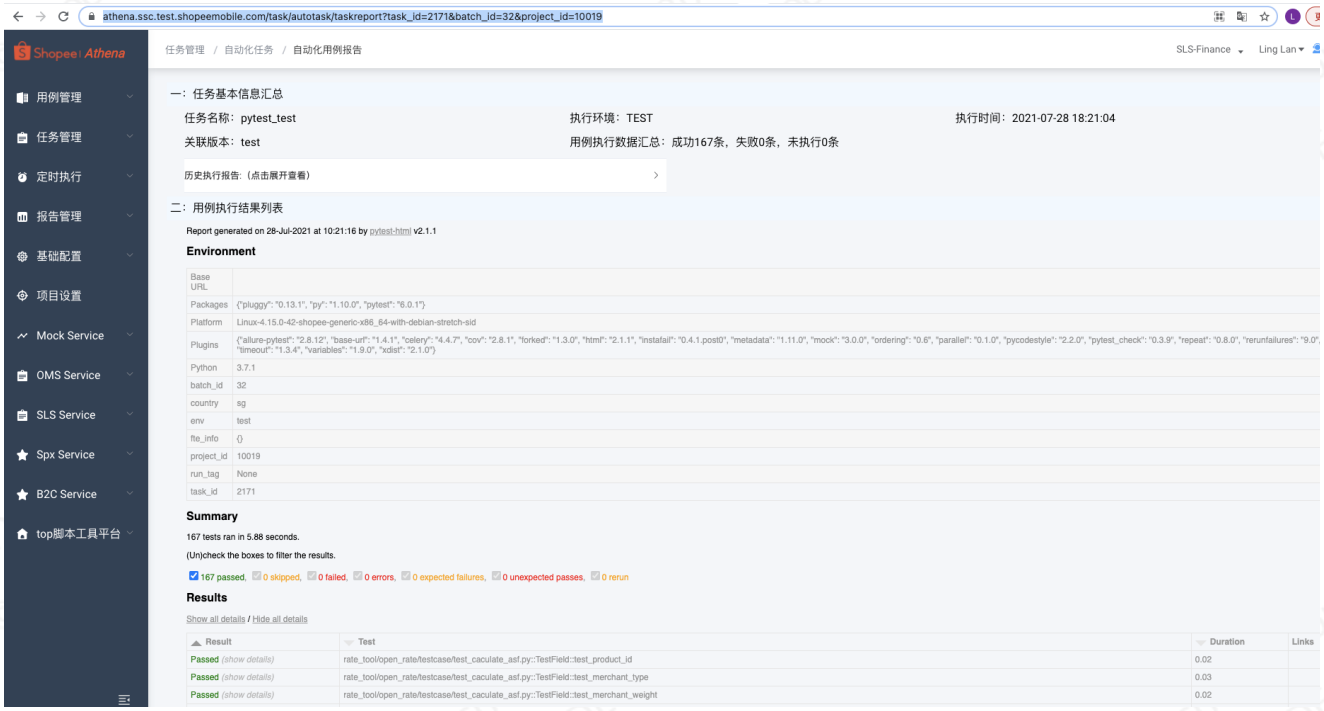
Allure：报告美观，占用空间大(一个完整的lls-admin的运行报告是25+M)。

pytest-html：报告较丑，占用空间小，约为Allure的1/20。

pytest_server所部署的机器，磁盘大小为50G。目前已使用50%。在没有扩容需求前，选择了pytest-html

用pytest-html生成的报告，是一个简单的html文件。直接使用nginx做了一个静态文件服务器： http://10.143.204.54:8000/report/

以此链接为例：http://10.143.204.54:8000/report/freight_recon/2171/33_1627469713.html 目前会内嵌到Athena的报告页面



3.4 并行执行任务评估

由于目前该台机器上有配 jenkins的任务。jenkins自己做资源调度时，最高是2-3个任务同时运行。

直接用pytest_server时，资源会抢占jenkins的调度资源。所以同时运行的任务应该最多不超过5个。

如果用pytest-xdist多进程运行case，建议不能超过 12个进程同时进行。

4 服务维护及用例运行排错

4.1 服务器地址

10.143.204.54（访问权限申请 gac上申请）

1. 该服务器8080端口为 jenkins提供服务，此jenkins亦是用来运行自动化用例
2. 该服务器8088端口为 pytest_server 提供服务
3. 该服务器8000 端口为静态资源服务器nginx提供服务，即测试报告的展示

该服务器为一个虚拟机实例，所有服务的搭建由自己搭建，而不是申请的公司的jenkins pipline，原因是auto_test在建项目之初就用的python3，pytest_server也用python3开发。公司的pipline没有兼容python3。

由于没有pipline，所以如果pytest_server有代码的更新，需要手动去 /home/ld-sgdev/ling_lan/pytest_server 去执行 git pull 拉代码。然后用启动命令启动。

nginx 配置的location映射位置为：

```
server {
        listen  8000;
        server_name localhost;
        location / {
                root html;
                index index.html index.html;
        }
        location /report/ {
                root  /home/ld-sgdev/ling_lan/;
                autoindex on;
        }
}
```

nginx.conf 所在位置为：/etc/nginx/nginx.conf

4.2 启动命令

cd /home/ld-sgdev/ling_lan/pytest_server

nohup python3 manage.py runserver 0.0.0.0:8088 --noreload &

4.3 运行日志查看及排错

pytest_server客户端日志在 /home/ld-sgdev/ling_lan/pytest_server/nohup.log，一些未正常运行的请求，可以在这里查看日志

auto_test 运行日志在 /home/ld-sgdev/ling_lan/pytest_server/xxx.log 如果在上面没有找到报错，再在具体的项目运行日志里查看

4.4 日志及报告的维护及清理

随着越来越多的项目在athena运行，产生的测试报告会越来越多。报告存储路径： /home/ld-sgdev/ling_lan/report/

如后续有需要，可以设置定时任务去清理过期报告

5 自动化用例接入及运行注意事项

5.1 新项目改造 auto_test

假如你是auto_test的使用者，想要接入 Athena 的 任务运行，需要做如下修改

1) 添加及修改conftest

见 3.2-2)

另： conftest 的引用关系尽量不跨项目引用，避免各项目间代码相互影响。大家在copy的时候需要改一下代码的引用关系

2) 修改pytest_server (可以找 ling.lan@shopee.com 做相关修改)

找到 pytest_server/pytest_server/views.py， 找到如下代码行

添加auto_test下的项目目录 如"lfs" "lls"与 Athena project_id的map关系

```
DIR_PROJECTID_MAP = {
        "lls": 18,
        "lcs": 13,
        "freight_recon": 10019,
        "shopee_logistic_open": 10025,
        "regression": 10024
}
```

3）修改Athena相关的代码配置，使用pytest 运行自动化任务（可以找 yongsheng.zhu@shopee.com 做相关修改）

5.2 运行语句

在Athena建自动化任务时，要配置auto_test的cmdline，及待测的分支。

配置Athena 运行的cmdline时，一定要以上面的项目目录作为运行目录的起始目录，且一定至少要写一个目录，如：["-sq", "freight_recon/rate_tool/open_rate/testcase/","-n=2"] or ["-sq", "-m", "smoke", "freight_recon/"]，代码中做了 project_id 与 目录的对应关系校验，避免别的项目组越权执行本项目组的case

6 待优化项及扩展项

git地址

https://git.garena.com/shopee/bg-logistics/qa/pytest_server.git

优化项1：

pytest_server中 auto_test 最新代码的拉取使用的是一个shell 命令行，后续可改成用 python-git 去完成

优化项2：

pytest-html的报告有些许难看，在磁盘空间允许的情况下，也是可以改成allure的

扩展项1：

如果需要将case上传至Athena，建议在pytest_server里做扩展，用 --collect-only 命令可以完成所有用例的收集，不需要每个目录去解析

扩展项2：

随着接入的任务越来越多，一台虚拟机的运行资源肯定不够大家使用。如有需要，可以借着 pytest-xdist 做个扩展，多申请几台网络相通的 slave 机器专门做执行