

借助 Python 开源生态打造企业级 自动化测试框架 (HttpRunner)

演讲人：李隆 (debugtalk)





主题概要

- 1、背景介绍
- 2、`HttpRunner` 设计思路
- 3、`HttpRunner` 核心特性
- 4、`HttpRunner` 实践案例
- 5、Q & A

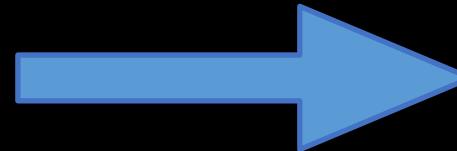
1、背景介绍





背景介绍

- 1、业务高速发展，需求迭代频繁
- 2、普遍具有功能回归测试、性能测试、持续集成、线上监控等需求



效率低

成本高

投入产出比低

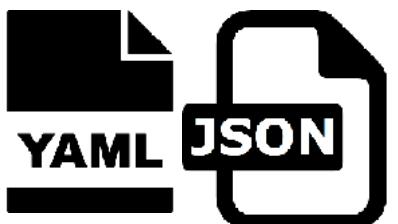


背景介绍



自动化测试 ?

基于 Python 的自动化测试生态



{
x} JSON Schema

django

Flask
web development,
one drop at a time

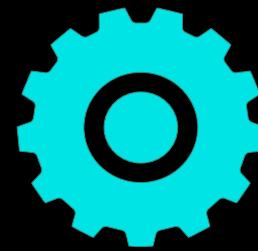
Jinja
Template Engine



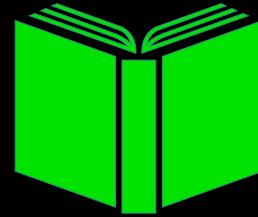


背景介绍

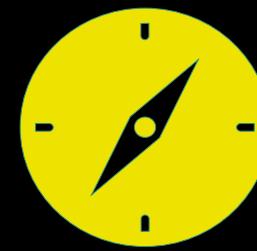
传统自动化测试方法普遍存在的问题：



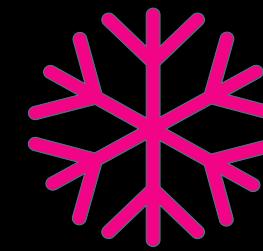
工具多 & 杂



学习成本高



团队协作难



维护成本高



目标：提升投入产出比

少投入

高收益

- 工具开发 & 维护
 - 学习使用成本
 - 编写 & 录制测试用例
 - 测试用例管理 & 维护
- 实现自动化回归测试
 - 性能测试脚本复用
 - 兼具持续集成、线上监控
 - 辅助手工测试：自定义生成特定业务数据

2、HttpRunner 设计思路



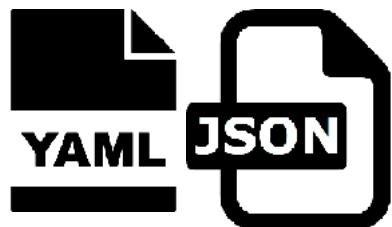


设计思路

- 如何快速实现从0到1?
- 如何避免闭门造车?
- 如何兼具自动化、性能测试、持续集成、线上监控?
- 如何保障工具（框架）质量?
- 如何实现协作管理?
- 如何将框架设计得更优雅?

设计思路 - 如何快速实现从0到1?

充分复用开源项目



{ } JSON Schema



福 Jinja



Jenkins



明显优势:

- 减少开发量
- 保障稳定性
- 降低学习成本



{...} swagger



设计思路 - 如何快速实现从0到1?

遵循行业标准



{x} JSON Schema

https://json-schema.org

Specification Learn Implementations Discussion

JSON Schema

The current version is **draft-07**.

JSON Schema is a vocabulary that allows you to **annotate** and validate JSON documents.

Advantages

JSON Schema	JSON Hyper-Schema
<ul style="list-style-type: none">• Describes your existing data format(s).• Provides clear human- and machine- readable documentation.• Validates data which is useful for:<ul style="list-style-type: none">◦ Automated testing.◦ Ensuring quality of client submitted data.	<ul style="list-style-type: none">• Make any JSON format a hypermedia format with no constraints on document structure• Allows use of URI Templates with instance data• Describe client data for use with links using JSON Schema.• Recognizes collections and collection items.

Main Interface



All of Requests' functionality can be accessed by these 7 methods. They all return an instance of the **Response** object.

requests.request(method, url, **kwargs)

Constructs and sends a **Request**.

[source]

Parameters: • **method** – method for the new **Request** object.

- **url** – URL for the new **Request** object.
- **params** – (optional) Dictionary or bytes to be sent in the query string for the **Request**.
- **data** – (optional) Dictionary or list of tuples [(key, value)] (will be form-encoded), bytes, or file-like object to send in the body of the **Request**.
- **json** – (optional) A JSON serializable Python object to send in the body of the **Request**.
- **headers** – (optional) Dictionary of HTTP Headers to send with the **Request**.
- **cookies** – (optional) Dict or CookieJar object to send with the **Request**.

设计思路 - 如何避免闭门造车?

拥抱开源生态

HttpRunner 中文使用手册

介绍

设计理念

逻辑流程图

HttpRunner / HttpRunner

Code Issues 79 Pull requests 1 Projects 3 Wiki Insights Settings

One-stop solution for HTTP(S) testing. <http://cn.htrunner.org/>

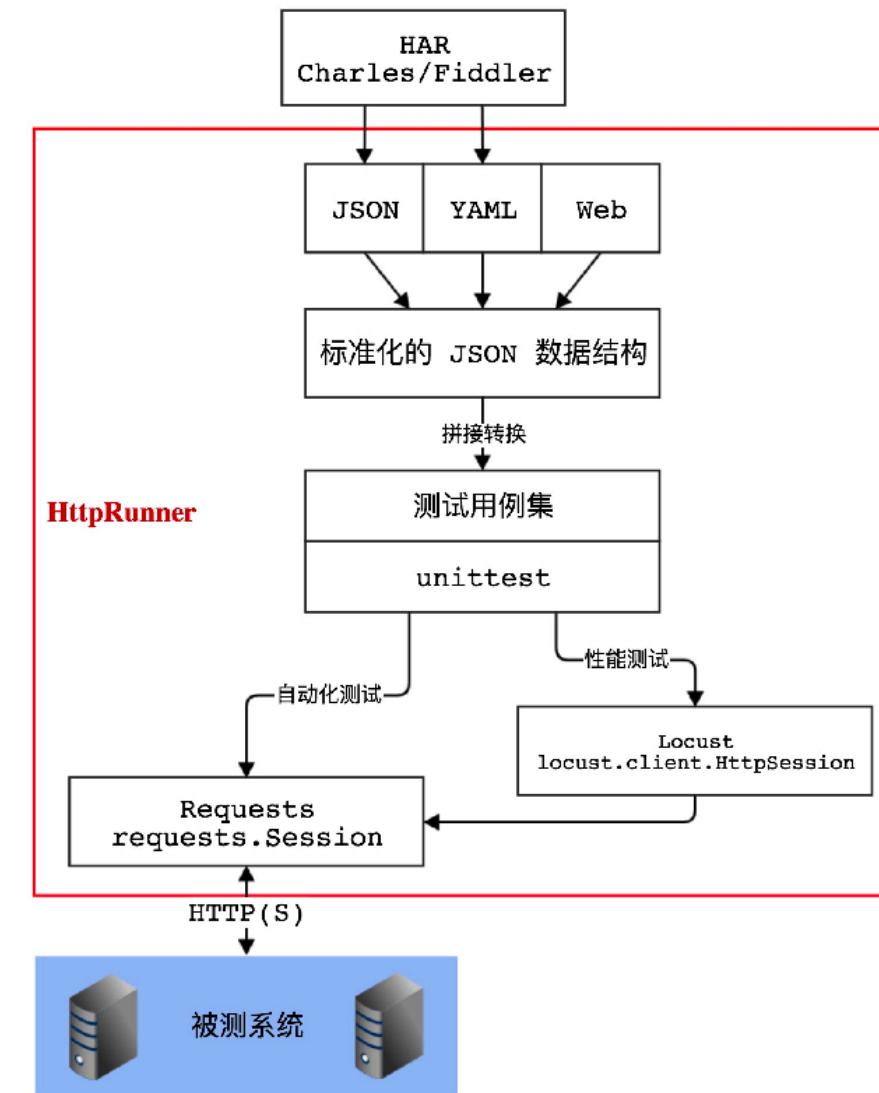
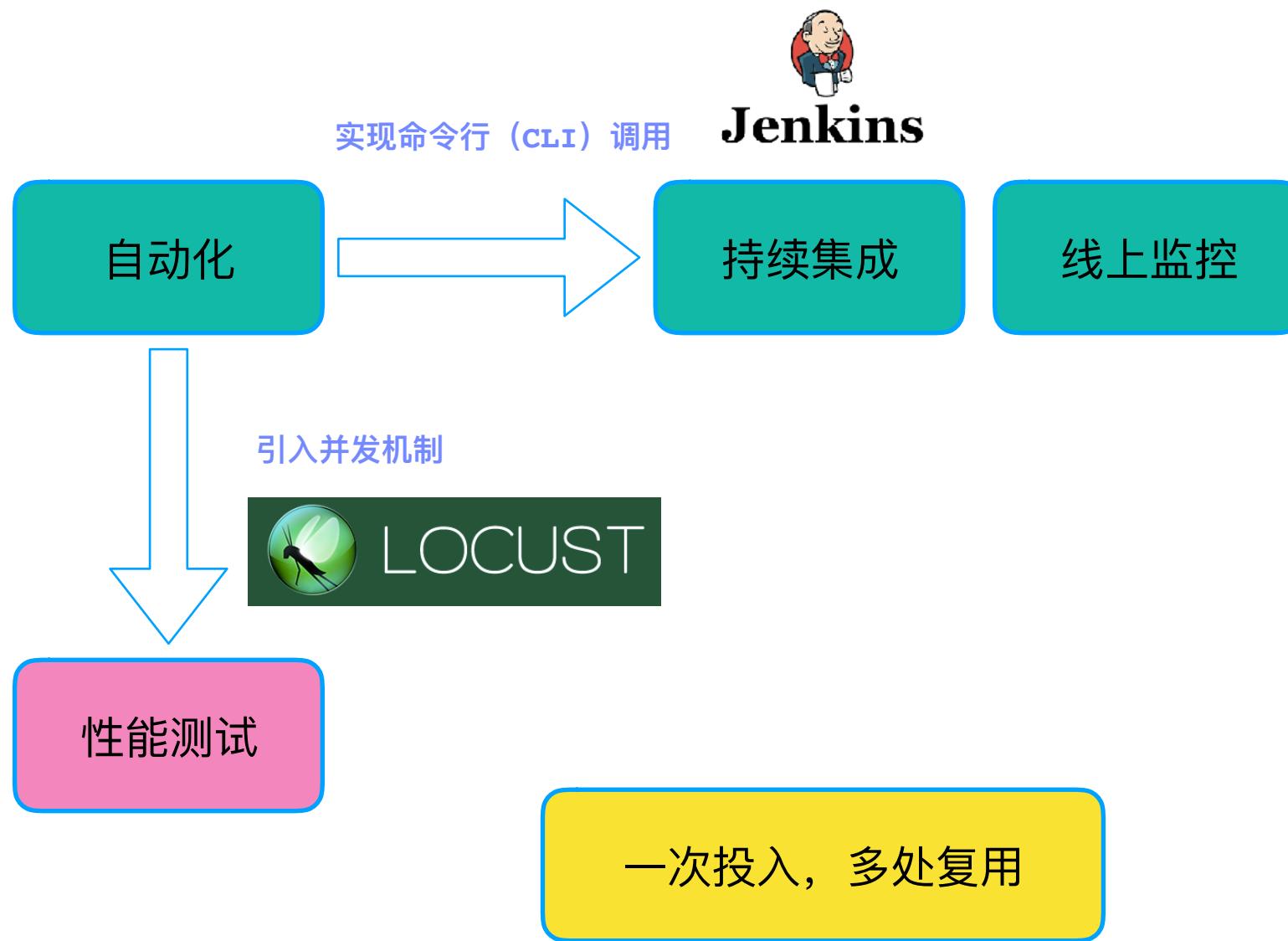
api-test locustio performance-testing locust httpstest testrunner htmltestrunner load-testing Manage topics

774 commits 4 branches 42 releases 5 contributors MIT

日期	标题	日期	作者
2017-06-20	ApiTestEngine 演进之路 (0) 开发未动，测试先行		
2017-06-18	接口自动化测试的最佳工程实践 (ApiTestEngine)		
2017-09-09	约定大于配置：ApiTestEngine实现热加载机制	2018-05-12	HttpRunner 实现 hook 机制
2017-08-27	ApiTestEngine 集成 Locust 实现更好的性能测试体验	2018-03-25	HttpRunner 再议参数化数据驱动机制
2017-08-20	ApiTestEngine QuickStart	2018-02-16	HttpRunner 实现参数化数据驱动机制
2017-08-05	How to install a package from Github that has other github depende	2018-02-08	HttpRunner 通过 skip 机制实现对测试用例的分组执行控制
2017-07-17	ApiTestEngine 演进之路 (4) 测试用例中实现 Python 函数的调用	2017-12-23	HttpRunner 的测试用例分层机制
2017-07-11	ApiTestEngine 演进之路 (3) 测试用例中实现 Python 函数的定义	2017-12-13	HttpRunner 的结果校验器优化
2017-07-07	ApiTestEngine 演进之路 (2) 探索优雅的测试用例描述方式	2017-11-14	HttpRunner 支持 HAR 意味着什么？
2017-06-28	300行Python代码打造实用接口测试框架	2017-11-08	ApiTestEngine 正式更名为 HttpRunner
2017-06-22	ApiTestEngine 演进之路 (1) 搭建基础框架	2017-11-06	ApiTestEngine，不再局限于API的测试

博客驱动开发

设计思路 - 如何兼具自动化、性能测试、持续集成、线上监控？



设计思路 - 如何保障工具质量?

测试驱动开发

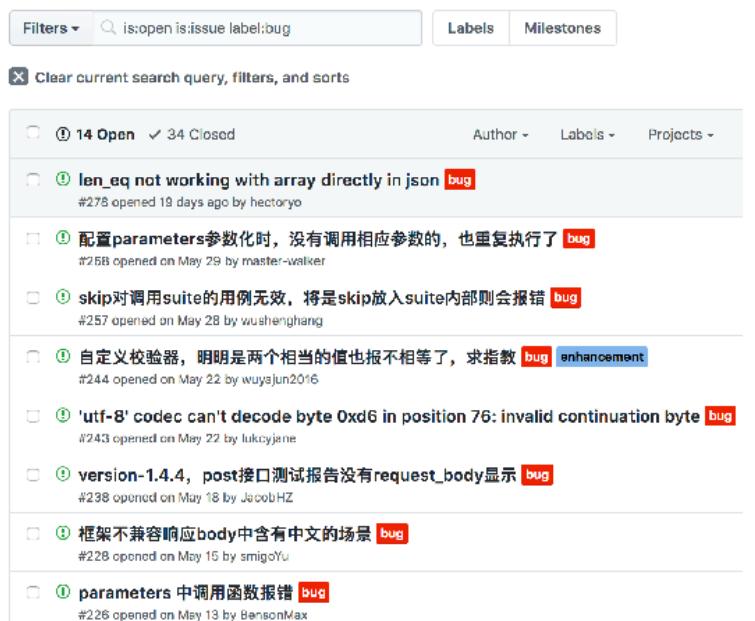
- Custom Mock Server
- httpbin.org by kennethreitz
- Postman Echo services

开源社区反馈

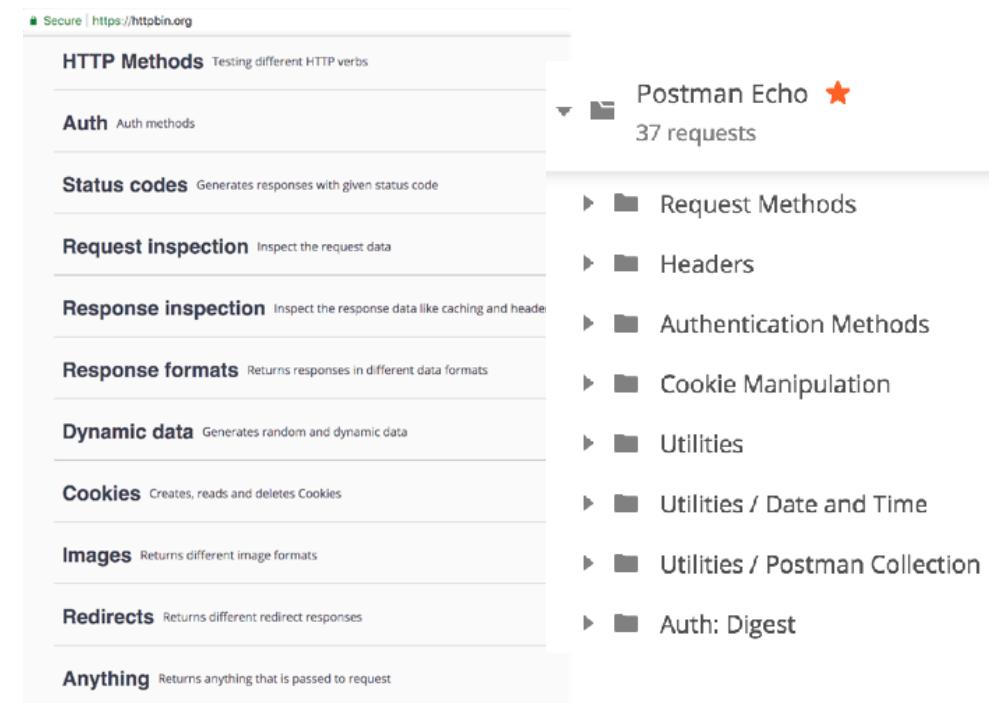
- Github issues
- TesterHome

HttpRunner

license MIT build passing coverage 81% pypi v1.5.12 python 2.7 | 3.4 | 3.5 | 3.6 | 3.7



The screenshot shows the GitHub Issues page for the HttpRunner repository. The search bar at the top contains the query "is:open is:issue label:bug". Below the search bar, there are filters for "Filters", "Labels", and "Milestones". The main list displays several open issues, each with a title, a brief description, and a link to the issue page. Some issues are labeled with "bug" or "enhancement".



The screenshot shows the Postman Echo API documentation. At the top, it says "Secure | https://httpbin.org". Below that, it lists various testing features: "HTTP Methods", "Auth", "Status codes", "Request inspection", "Response inspection", "Response formats", "Dynamic data", "Cookies", "Images", "Redirects", and "Anything". To the right, there is a sidebar titled "Postman Echo" with a star icon, showing a tree view of "37 requests" categorized under "Request Methods", "Headers", "Authentication Methods", "Cookie Manipulation", "Utilities", "Utilities / Date and Time", "Utilities / Postman Collection", and "Auth: Digest".

设计思路 - 如何实现协作管理?

约定大于配置

- 复用 Requests interface
- 统一测试用例描述形式 (JSON/YAML)
- JSON Schemas

```
import requests # make sure this is installed
from jsonschema import validate
from jsonschema.exceptions import ValidationError

schema = requests.get('http://schema.httprunner.org/json/v1.json').json()

test_input = {} # Whatever needs to be validated.

try:
    validate(test_input, schema)
except ValidationError:
    print 'It is not a valid collection'
else:
    print 'It is a valid collection'
```

```
[{"test": {"name": "create user which does not exist", "request": {"url": "http://127.0.0.1:5000/api/users/1000", "method": "POST", "headers": {"device_sn": "9TN6O2Bn1vzfzbF", "token": "kEoKIu6SRPTX3IZA"}, "json": {"name": "user1", "password": "123456"}}, "assert": {"$": [{"status_code": 201}], "$content.success": true}}]
```

```
→ demo git:(master) ✘ hrun --pretty test1.json
Start to prettyfy JSON file: test1.json
success: test1.pretty.json
```

```
{
    "$schema": "http://json-schema.org/draft-06/schema",
    "$id": "https://github.com/HttpRunner/schemas/blob/master/json/v1.json",
    "title": "HttpRunner Schemas",
    "description": "HttpRunner Schemas",
    "type": "array",
    "items": {
        "oneOf": [
            {"$ref": "#/definitions/config"},
            {"$ref": "#/definitions/test"}
        ]
    },
    "definitions": {
        "config": {
            "$id": "#/definitions/config",
            "description": "testset config, used for public configuration",
            "type": "object",
            "minProperties": 1,
            "maxProperties": 1,
            "properties": {
                "config": {
                    "$ref": "#/definitions/config_dict"
                }
            },
            "required": ["config"]
        },
        "config_dict": {
            ...
        },
        "test": {
            "$id": "#/definitions/test",
            "description": "testcase description",
            "type": "object",
            "minProperties": 1,
            "maxProperties": 1,
            "properties": {
                ...
            }
        }
    }
}
```

设计思路 - 如何实现协作管理?

```
import unittest
import requests

class FirstTestcase(unittest.TestCase):

    def test_get_token(self):
        url = "http://127.0.0.1:5000/api/get-token"
        method = "POST"
        headers = {
            "user_agent": "iOS/10.3",
            "device_sn": "9TN602Bn1vzfzbF",
            "os_platform": "ios",
            "app_version": "2.8.6"
        }
        json = {
            "sign": "19067cf712265eb5426db8d3664026c1cceaa02b9"
        }

        resp = requests.request(method, url, headers=headers, json=json)
        status_code = resp.status_code
        token = resp.json()["token"]

        assert status_code == 200
        assert len(token) == 16

    def test_create_user_which_does_not_exist(self):
        url = "http://127.0.0.1:5000/api/users/1000"
        method = "POST"
        headers = {
            "device_sn": "9TN602Bn1vzfzbF",
            "token": "kEoKIu6SRPTX3IZA"
        }
        json = {
            "name": "user1",
            "password": "123456"
        }

        resp = requests.request(method, url, headers=headers, json=json)
        status_code = resp.status_code
        success = resp.json()["success"]

        assert status_code == 201
        assert success is True
```

信息量等价

重复

迥异

简洁

规范

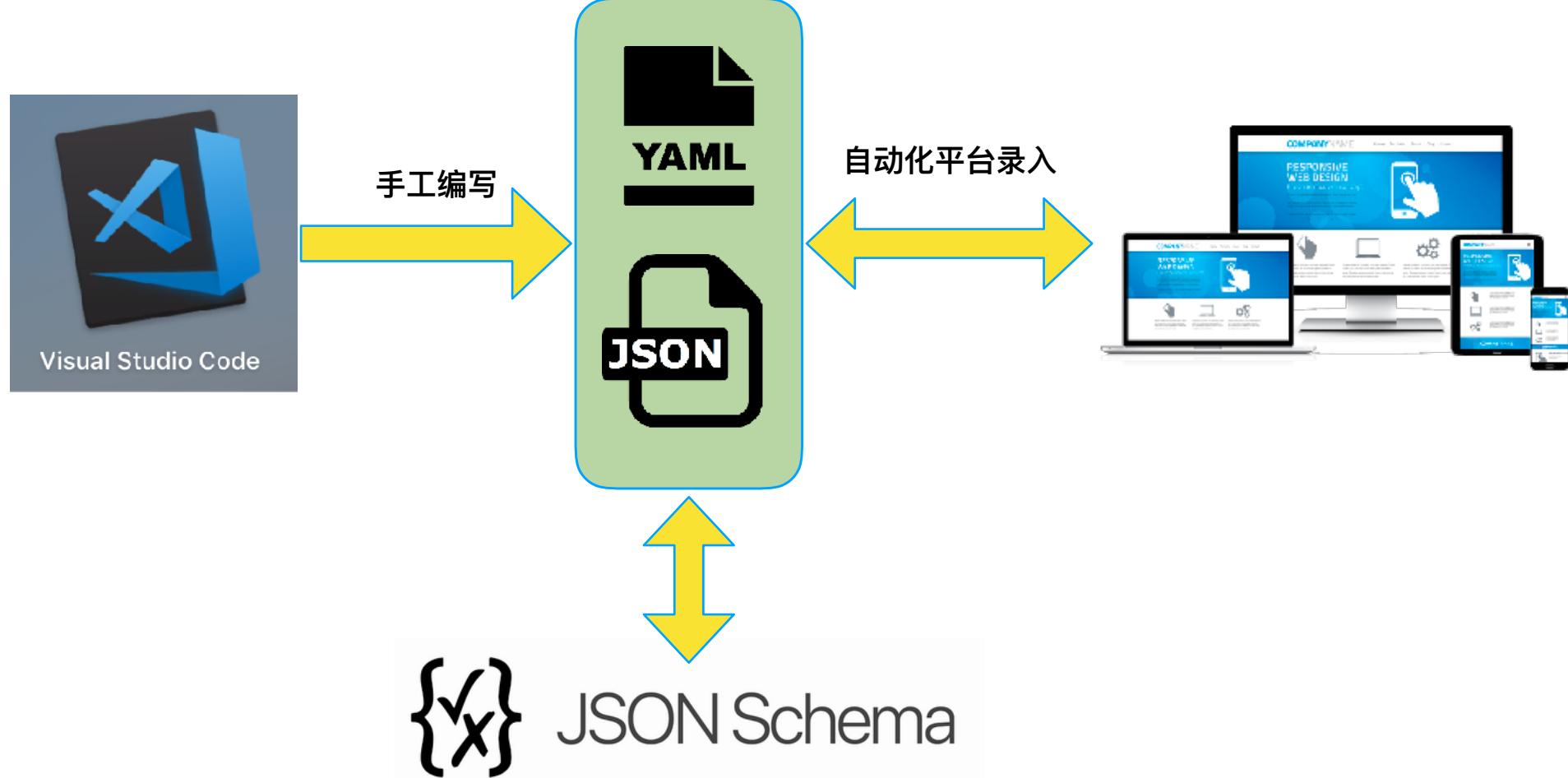
```
- test:
    name: get token
    request:
        url: http://127.0.0.1:5000/api/get-token
        method: POST
        headers:
            user_agent: "iOS/10.3",
            device_sn: "9TN602Bn1vzfzbF",
            os_platform: "ios",
            app_version: "2.8.6"
        json:
            sign: 19067cf712265eb5426db8d3664026c1cceaa02b9
    extract:
        - token: content.token
    validate:
        - eq: [status_code, 200]
        - len_eq: [content.token, 16]

- test:
    name: create user which does not exist
    request:
        url: http://127.0.0.1:5000/api/users/1000
        method: POST
        headers:
            device_sn: "9TN602Bn1vzfzbF",
            token: "kEoKIu6SRPTX3IZA"
        json:
            name: "user1"
            password: "123456"
    validate:
        - eq: [status_code, 201]
        - eq: [content.success, true]
```

设计思路 - 如何实现协作管理?

脚本与平台无缝切换

- 统一存储和管理
- 平台服务化



设计思路 - 如何将框架设计得更优雅?

借鉴优秀（开源）项目

debugtalk.py

测试用例模板语言
(JSON/YAML)

参数化机制
(parameters)

环境变量管理 (.env)

测试用例分层管理



设计思路 - 如何将框架设计得更优雅?

遵循 Unix 哲学

“Write programs that do one thing and do it well.”

“Write programs to work together”

prepare

initializer

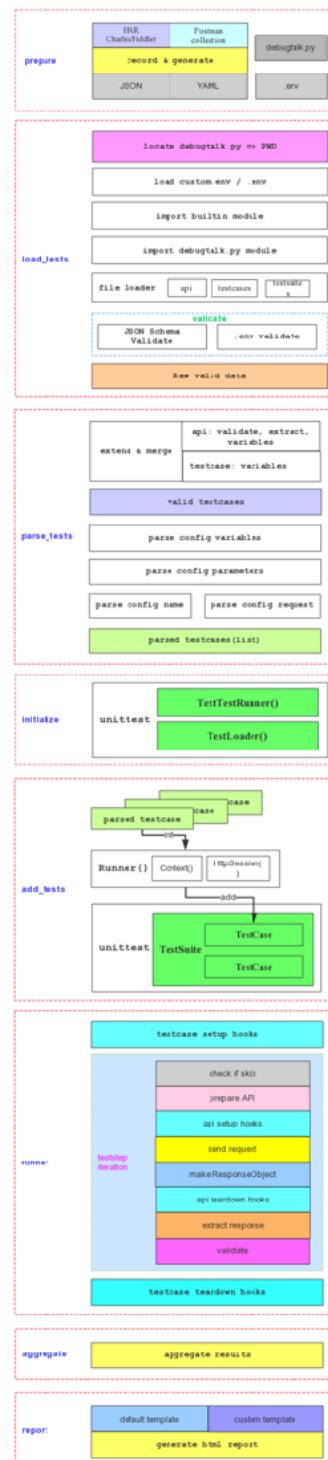
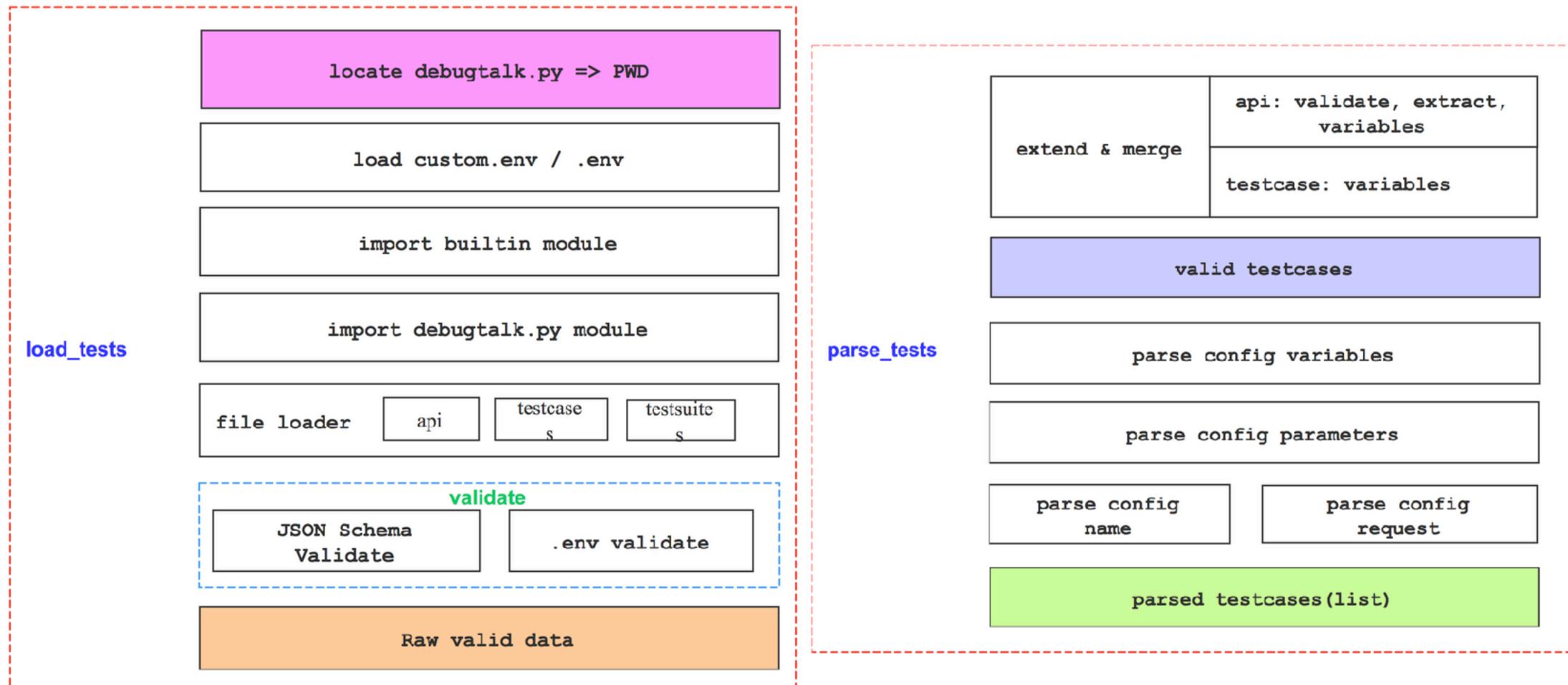
loader

runner

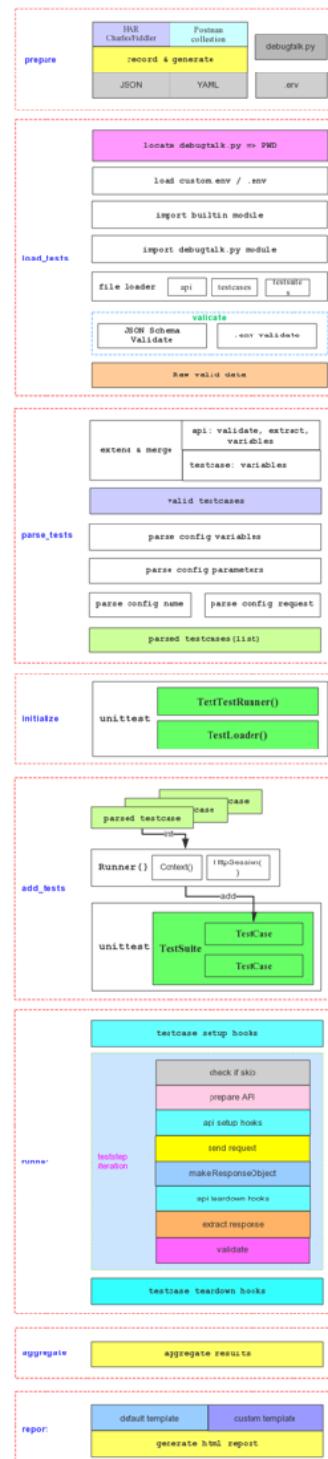
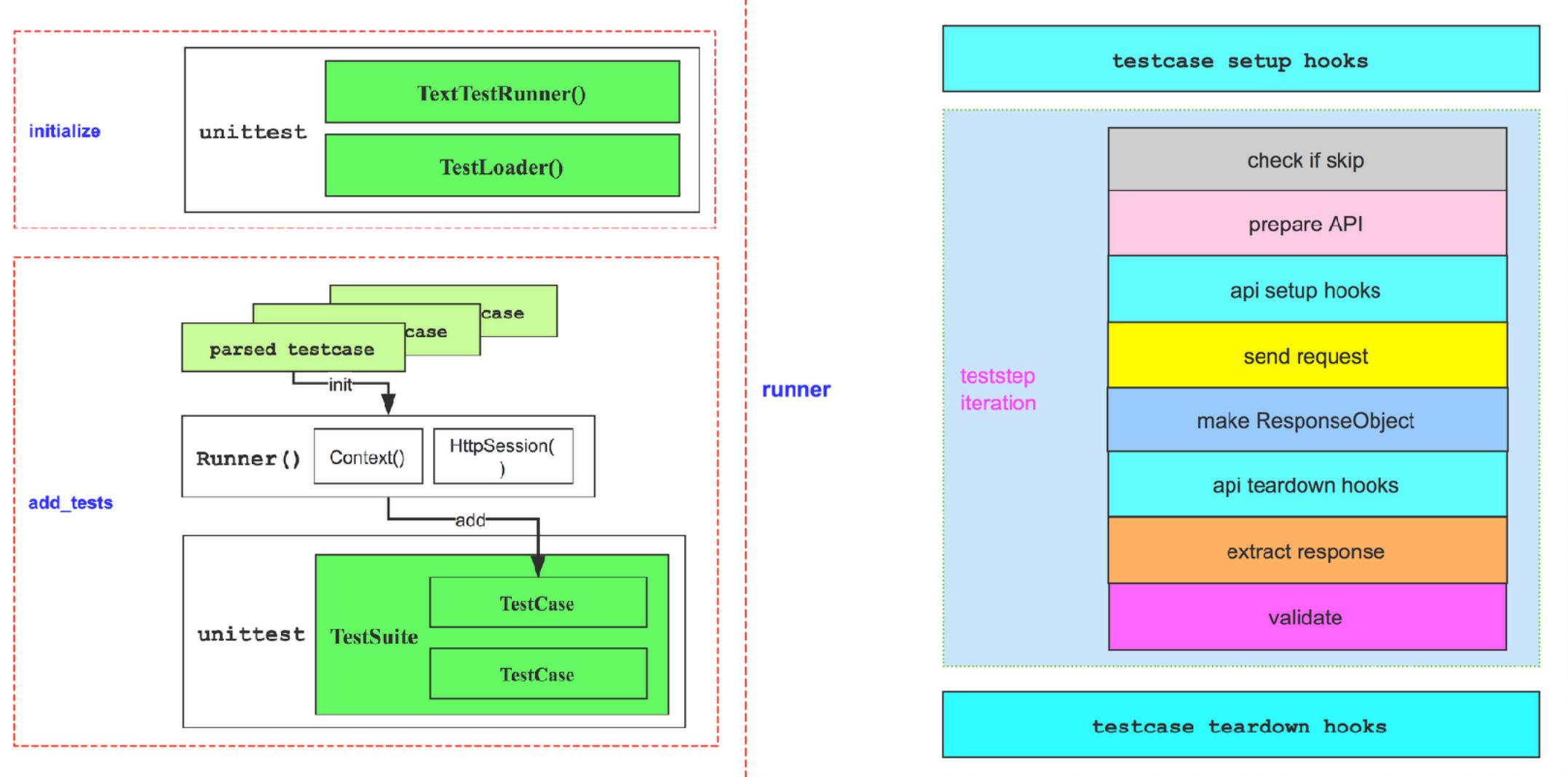
parser

report

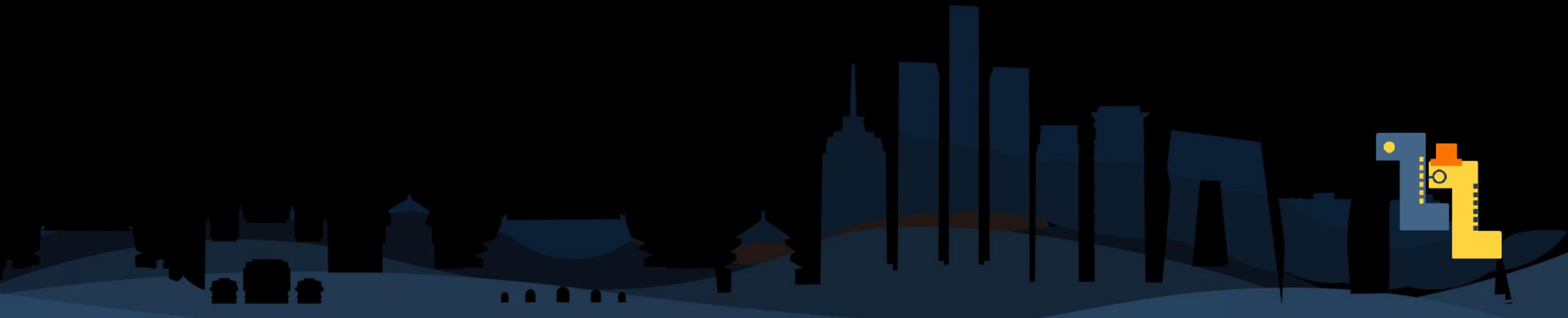
逻辑流程图



逻辑流程图



3、HttpRunner 核心特性





HttpRunner 是什么？

简单易用

功能强大

接口自动化测试工具

核心特性

抓包录制 & 生成用例

Postman 转换生成用例

Swagger 转换生成用例

YAML/JSON 用例格式

JSON Schema

数据驱动机制

测试用例分组执行控制

setup & teardown hooks

动态计算(*debugtalk.py*)

性能测试

CLI 调用

可扩展测试报告(*Jinja2*)

测试用例分层机制

加载环境变量(*.env*)

...

核心特性 - 测试准备



export

HAR

har2case



postman2case



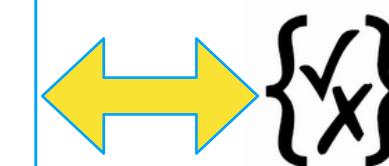
swagger2case



自动化平台录入/手工编写



录制生成测试用例



JSON Schema

核心特性 - 测试脚本优化

```
- config:  
  name: "smoketest for CRUD users."  
  parameters:  
    - os_platform: ["ios", "android"]  
  variables:  
    - user_agent: 'ios/10.3'  
    - device_sn: ${gen_random_string(15)}  
    - os_platform: 'ios'  
    - app_version: '2.8.6'  
  request:  
    base_url: http://127.0.0.1:5000  
    headers:  
      Content-Type: application/json  
      device_sn: $device_sn
```

结果提取
extract

结果校验
validate

数据驱动
parameters

用例集全局定义

```
- test:  
  name: get token  
  request:  
    url: /api/get-token  
    method: POST  
    headers:  
      user_agent: $user_agent  
      device_sn: $device_sn  
      os_platform: $os_platform  
      app_version: $app_version  
    json:  
      sign: ${get_sign($user_agent, $device_sn, $os_platform, $app_version)}  
  extract:  
    - token: content.token  
  setup_hooks:  
    - ${setup_hook_prepare_kwargs($request)}  
  teardown_hooks:  
    - ${teardown_hook_sleep_N_secs($response, 2)}  
  validate:  
    - {"check": "status_code", "comparator": "eq", "expect": 200}  
    - len_eq: ["content.token", 16]
```

setup_hooks

teardown_hooks

变量定义
variables

变量引用
\$var

函数调用
\${func(\$a, 9)}

```
- test:  
  name: create user which does not exist  
  request:  
    url: /api/users/1000  
    method: POST  
    headers:  
      token: $token  
    json:  
      name: "user1"  
      password: "123456"  
  validate:  
    - eq: ["status_code", 201]  
    - {"check": "content.success", "comparator": "eq", "expect": true}
```

核心特性 - 测试脚本优化

函数定义
debugtalk.py

```
import hashlib
import hmac
import random
import string

SECRET_KEY = "DebugTalk"

def gen_random_string(str_len):
    random_char_list = []
    for _ in range(str_len):
        random_char = random.choice(string.ascii_letters + string.digits)
        random_char_list.append(random_char)

    random_string = ''.join(random_char_list)
    return random_string

def get_sign(*args):
    content = ''.join(args).encode('ascii')
    sign_key = SECRET_KEY.encode('ascii')
    sign = hmac.new(sign_key, content, hashlib.sha1).hexdigest()
    return sign
```

核心特性 - 测试执行

```
[→ httprunner git:(master) ✘ hrun tests/httpbin/basic.yml --log-level debug
INFO Loading environment variables from /Users/debugtalk/MyProjects/HttpRunner-dev/HttpRunner/tests/.env
INFO Loaded variable: UserName
INFO Loaded variable: Password
INFO Loaded variable: PROJECT_KEY
INFO Start to run testcase: basic test with httpbin
user-agent
DEBUG call hook: ${setup_hook_prepare_kwargs($request)}
INFO GET /user-agent
DEBUG request kwargs(raw): {}
DEBUG processed request:
> GET http://httpbin.org/user-agent
> kwargs: {'timeout': 120}
DEBUG ===== request details =====
url : 'http://httpbin.org/user-agent'
method : 'GET'
headers : {'User-Agent': 'python-requests/2.18.4', 'Accept-Encoding': 'gzip, deflate', 'Accept': '*/*', 'Connection': 'keep-alive'}
start_timestamp : 1535087147.133355
body : None
DEBUG ===== response details =====
status_code : 200
headers : {'Connection': 'keep-alive', 'Server': 'gunicorn/19.9.0', 'Date': 'Fri, 24 Aug 2018 05:05:47 GMT', 'Content-Type': 'application/json'}
content_size : 45
response_time_ms : 528.63
elapsed_ms : 521.792
encoding : None
content : b'(\n "user-agent": "python-requests/2.18.4"\n)\n'
content_type : 'application/json'
ok : True
url : 'http://httpbin.org/user-agent'
reason : 'OK'
cookies : {}
text : '(\n "user-agent": "python-requests/2.18.4"\n)\n'
json : {'user-agent': 'python-requests/2.18.4'}
```

```
Ran 2 tests in 0.924s
-----
```

OK

```
INFO Start to render Html report ...
INFO Generated Html report: /Users/debugtalk/MyProj
```

```
Ran 1 test in 0.532s
-----
```

```
OK
DEBUG ===== Variables & Output =====
Type | Variable : Value
----- | -----
Var | HTTPBIN_SERVER : http://127.0.0.1:3458
Var | SECRET_KEY : DebugTalk
Var | BASE_URL : http://127.0.0.1:5000
Var | demo_default_request : {"base_url": "$BASE_URL", "headers": {"content-type": "application/json"}}

-----
```

```
DEBUG No html report template specified, use default.
INFO Start to render Html report ...
DEBUG render data: {'success': True, 'stat': {'testsRun': 1, 'failures': 0, 'errors': 0, 'skipped': 0, 'expectedFailures': 0, 'unexpectedFailures': 3203}, 'platform': {'httprunner_version': '1.5.11', 'python_version': 'CPython 3.6.5+', 'platform': 'Darwin-17.6.0-x86_64-i386-64bit!'}, 'data': {'start_at': 1535087147.132546, 'duration': 0.5316638946533203}, 'records': [{"url": "http://httpbin.org/user-agent", "method": "GET", "headers": {"User-Agent": "python-requests/2.18.4", "Accept-Encoding": "gzip, deflate", "Accept": "*/*", "Connection": "keep-alive"}, "status_code": 200, "headers": {"Connection": "keep-alive", "Server": "gunicorn/19.9.0", "Date": "Fri, 24 Aug 2018 05:05:47 GMT"}]}]
```

核心特性 - 测试报告

Request and Response data

Test Report:

Summary

START AT	2018-08-24 13:08:18		
DURATION	3.566 seconds		
PLATFORM	HttpRunner 1.5.11	Cython 3.6.5+	Darwin-10.14.5
TOTAL	SUCCESS	FAILED	ERROR
9	9	0	0

Details

basic test with httpbin

base_url	http://httpbin.org/ params		
TOTAL: 9	SUCCESS: 9	FAILED: 0	ERROR: 0
Status	Name		Response Time
success	headers		564.02 ms
success	user-agent		303.29 ms
success	get without params		261.03 ms
success	get with params in url		264.09 ms
success	get with params in params field		290.97 ms
success	set cookie		568.02 ms
success	extract cookie		257.49 ms
success	post data		295.98 ms
success	validate content length		743.18 ms

Request:

url	http://httpbin.org/user-agent
method	GET
headers	User-Agent: python-requests/2.18.4 Accept-Encoding: gzip, deflate Accept: */* Connection: keep-alive
start_timestamp	1535087298.8629591
body	None

Response:

status_code	200
headers	Connection: keep-alive Server: gunicorn/19.9.0 Date: Fri, 24 Aug 2018 05:08:19 GMT Content-Type: application/json Content-Length: 45 Content-Encoding: gzip Via: 1.1 gunicorn Access-Control-Allow-Origin: *

Validators:

	check	comparator	expect value	actual value
encoding	None	status_code	eq	200
content	{ "us" }	content.user-agent	startswith	python-requests

Statistics:

content_size(bytes)	45
response_time(ms)	303.29
elapsed(ms)	301.897

核心特性 - 性能测试

```
[→ httprunner git:(master) ✘ locusts -f skypixel.yml --processes 4
INFO    Loading environment variables from /Users/debugtalk/MyProjects/HttpRunner-dev/HttpRunner/.env
[2018-10-14 00:41:40,940] Leos-MacBook-Pro.local/INFO/locust.main: Starting web monitor at *:8089
[2018-10-14 00:41:40,946] Leos-MacBook-Pro.local/INFO/locust.main: Starting Locust 0.8.1
[2018-10-14 00:41:40,947] Leos-MacBook-Pro.local/INFO/locust.main: Starting Locust 0.8.1[2018-10-14 00:41:40,947] Leos-MacBook-Pro.local/INFO/locust.ma
in: Starting Locust 0.8.1

[2018-10-14 00:41:40,948] Leos-MacBook-Pro.local/INFO/locust.main: Starting Locust 0.8.1
[2018-10-14 00:41:40,949] Leos-MacBook-Pro.local/INFO/locust.main: Starting Locust 0.8.1
[2018-10-14 00:41:40,952] Leos-MacBook-Pro.local/INFO/locust.runners: Client 'Leos-MacBook-Pro.local_0e1aba43bd2aaafc88ff28bc2b6ddd9ae' reported as ready.
Currently 1 clients ready to swarm.
[2018-10-14 00:41:40,952] Leos-MacBook-Pro.local/INFO/locust.runners: Client 'Leos-MacBook-Pro.local_1f6cda5ab3b7270dbc0e643268a60f27' reported as ready.
Currently 2 clients ready to swarm.
[2018-10-14 00:41:40,952] Leos-MacBook-Pro.local/INFO/locust.runners: Client 'Leos-MacBook-Pro.local_6610254251722056_0e912410-7241-4
y. Currently 3 clients ready to swarm.
[2018-10-14 00:41:40,952] Leos-MacBook-Pro.loca
y. Currently 4 clients ready to swarm.
```

The image shows a screenshot of the Locust web interface. At the top right, there is a green header bar with the text "HOST https://www.skypixel.com", "STATUS READY", and "SLAVES 4". The main content area has a dark green background. On the left, there is a logo of a green locust and the word "LOCUST". In the center, the text "Start new Locust swarm" is displayed in white. Below it, there are two input fields: "Number of users to simulate" with the value "100" and "Hatch rate (users spawned/second)" with the value "10". At the bottom right, there is a green button labeled "Start swarming".



HttpRunner 还是什么？

不仅仅是自动化测试工具

通用的自动化
测试解决方案

融合最佳工程
实践

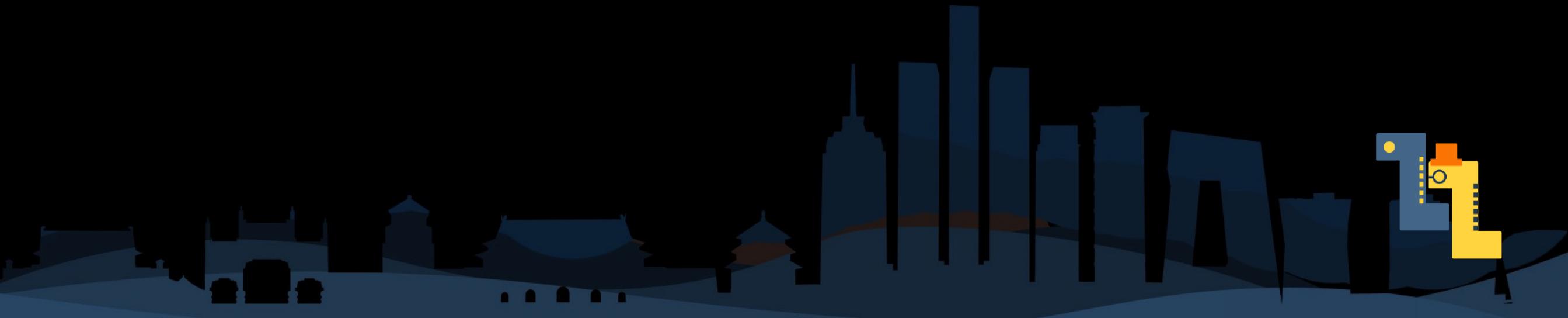
打造接口自动化测试生态



基于 HttpRunner 的接口测试平台：

- [HttpRunnerManager](#)
- [FastRunner](#)
- [SECO](#)
- [testcenter](#)
- [ApiTestWeb](#)

4、HttpRunner 实践案例

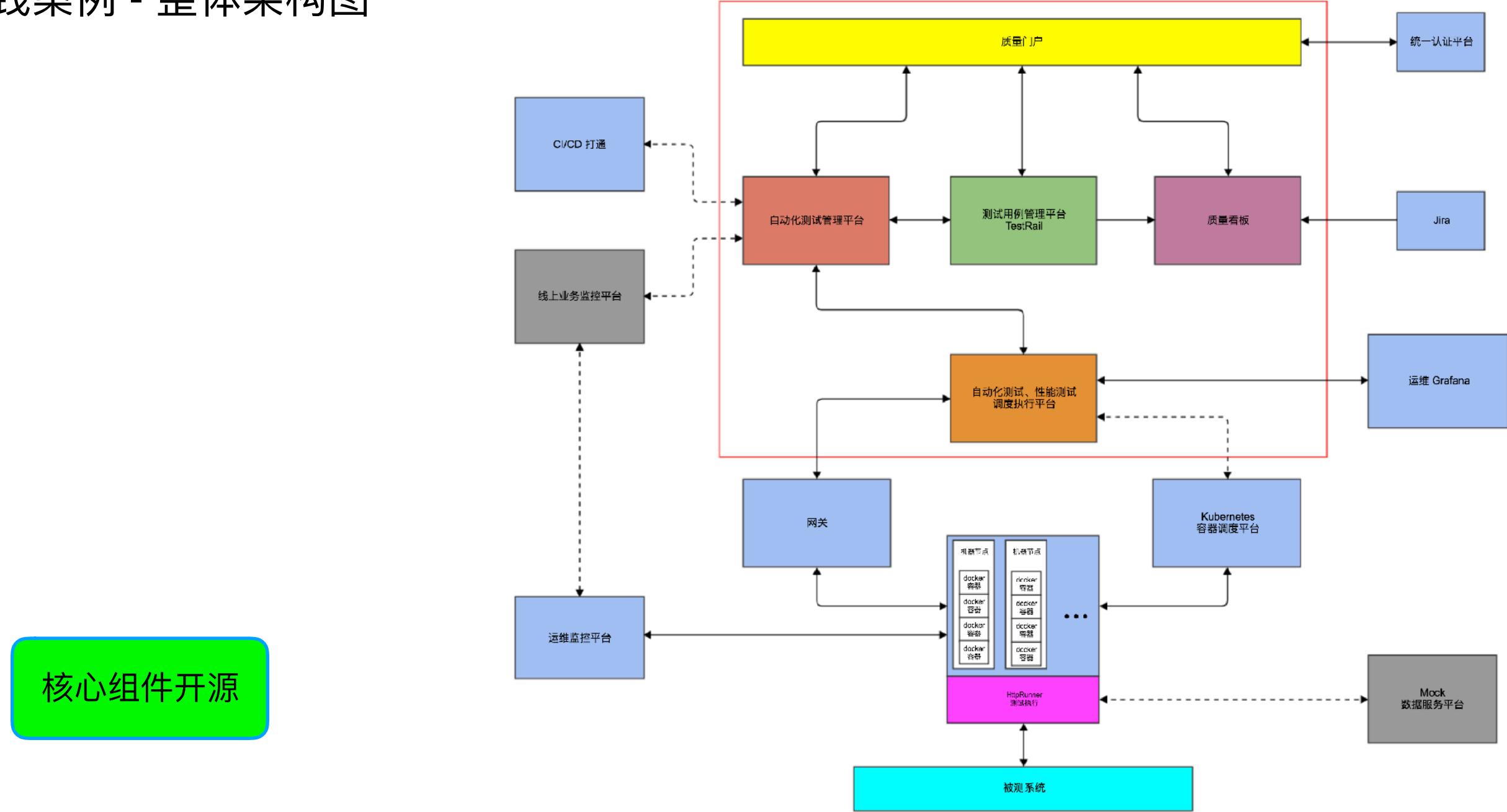




实践案例

- 整体架构图
- 自动化测试平台
- 打通功能测试平台（TestRail）
- 性能测试平台

实践案例 - 整体架构图



实践案例 - 自动化测试平台

接口查看 + 编辑接口 树状视图

common

项目名称	demo	
接口描述	登录用户中心	
接口定义	login_account	
接口参数		
GET	https://[REDACTED]/login	params

request	headers		
	data	backUrl	\$backUrl
variables	appId	appId	\$appId
	backUrl	http://[REDACTED]/repair/index&locale=zh_CN	
	appId	store	
extract			
hooks			
validate			

在线编辑 & 查看

接口查看 + 编辑接口 表格视图

common

项目名称	demo	
接口描述	登录用户中心	
接口定义	login_account	
接口参数		
GET	https://[REDACTED]/login	params

```
1: {
2:   "test": {
3:     "request": {
4:       "params": {
5:         "backUrl": "$backUrl",
6:         "appId": "$appId"
7:       },
8:       "headers": {},
9:       "url": "https://[REDACTED]/login",
10:      "method": "GET"
11:    },
12:    "variables": [
13:      {
14:        "backUrl": "http://[REDACTED]/repair/index&locale=zh_CN"
15:      },
16:      {
17:        "appId": "store"
18:      }
19:    ],
20:    "extract": [],
21:    "def": "login_account()", // 可以直接在这里调用自定义的步骤
22:    "validate": []
23:  }
24: }
```

实践案例 - 自动化测试平台

common

项目名称	demo
接口描述	登录用户中心
接口定义	login_account
接口参数	方法参数,\$xxx

GET https://account.dbeta.me/login 发送 保存

data

Key	Type	Value
backUrl	string	\$backUrl - +
appId	string	\$appId - +
	string	- +

request

key	type	value
cookies	string	- +

测试结果 | 测试通过

request_header	Connection: "keep-alive", Accept: "*/*", User-Agent: "python-requests/2.19.1", Accept-Encoding: "gzip, deflate"
request_body	"None"
response_header	Server: "openresty", Content-Length: "1539", ETag: "W/"5b7ecb5d-a08\"", Content-Type: "text/html", Connection: "keep-alive", Last-Modified: "Thu, 23 Aug 2018 14:57:33 GMT", Content-Encoding: "gzip", Date: "Fri, 31 Aug 2018 11:40:22 GMT"
response_body	b'<!DOCTYPE html><html><head><meta charset=utf-8><title>DJ...</title><meta name=keywords content="DJI, aerial photography, aerial filming, drone, UAV, camera gimbal, quadcopter"><meta name=description conte ... TOO LARGE ... ccount-cdn dbeta.me/pc/static/js/vendor.0ed151126d339edbf707.js></script><script type=text/javascript src=https://account-cdn.dbeta.me/pc/static/js/app.d42b2b57632a7b58ec4d.js></script></body></html>'

在线调试接口

实践案例 - 自动化测试平台

用例组装

接口
拖动接口至右边测试用例列表组成测试用例。
test1 api luguo.he
ddd api luguo.he
test_api api luguo.he
test_api_copy api luguo.he
test_jj api luguo.he
test api luguo.he
test_copy api luguo.he
获取配置文件 api luguo.he
defabcabc api luguo.he
origizationService api luguo.he
origizationService api luguo.he

测试用例
从左侧接口列表拖动接口组成测试用例。
test testcase luguo.he
test testcase luguo.he
test testcase luguo.he
获取配置文件 testcase luguo.he

测试场景
从左侧接口列表/测试用例列表拖动组成测试场景。
ddd api luguo.he
test testcase luguo.he



测试用例分层管理

拖拽组装生成测试场景

实践案例 - 打通功能测试平台 (TestRail)

The screenshot shows the 'Edit Test Case' screen for a test case titled 'demo-api-test'. The 'Type' dropdown is set to 'Functional'. A dropdown menu for 'UI Automation' status is open, with the option '未编写' (Not Written) highlighted and selected. Other options in the dropdown include '不能自动化' (Cannot Automate) and '已编写' (Written).

The screenshot shows the 'demo-api-test' test case page. The 'Edit' button is highlighted with a red box. A success message at the top states 'Successfully updated the test case.'

This screenshot displays a custom web application for managing automation cases. It includes fields for '项目名称' (Project Name), '用例描述' (Case Description), '用例定义' (Definition), '用例参数' (Parameters), 'worker' (Worker), 'TestRail ID' (TestRail ID), and 'base url' (Base URL). A red box highlights the 'TestRail ID' field, which contains the value '61946'. The background shows a sidebar with navigation links like 'DJI官网', '接口定义', '测试用例', '测试场景', and '测试数据'.

关联功能测试用例和自动化
测试用例

自动化率统计

实践案例 - 打通功能测试平台 (TestRail)

The screenshot shows the TestRail interface for a project named 'TEST_演示项目'. At the top, there are several tabs: Overview, Todo, Milestones, Test Runs & Results (selected), Test Suites & Cases, Reports, and 导入导出用例. Below the tabs is a toolbar with icons for email, attachments, reports, rerun, edit, suite, and trigger automation tasks. A red box highlights the '触发自动化任务' button. To the right of the toolbar is a yellow box labeled '触发执行'.

Below the toolbar is a pie chart summary:

- 2 Passed (67% set to Passed)
- 0 Partially Passed (0% set to Partially Passed)
- 0 Blocked (0% set to Blocked)
- 0 Retest (0% set to Retest)
- 0 Failed (0% set to Failed)

The chart indicates 67% passed, 1 / 3 untested (33%).

Below the summary is a table titled 'Test Cases (3)'. The columns are ID, Title, Assigned To, and Status. The data is as follows:

ID	Title	Assigned To	Status
T2796...	演示		Passed
T2796...	sds		Passed
T2796...	demo-api-test		Untested

触发执行

The screenshot shows a web-based task management system. The URL is https://q.djicorp.com/autotest/task_list/1/?type_data=testrail_testrun_1726. The page has a sidebar with '项目列表' and '任务列表'. The main content area is titled '任务列表' and includes search fields for '任务名称' (testrail_testrun_) and '创建人'. A table lists tasks with columns: 任务名称, 运行状态(完成/失败/正在运行/等待运行), 测试结果(通过/不通过), 类型, and 创建人.

任务名称	运行状态(完成/失败/正在运行/等待运行)	测试结果(通过/不通过)	类型	创建人
demo	2/0/0/0	2/0	testrail_testrun_1726	leo.lee
demo_演示	leo.lee	运行完成	测试通过	2018-07-06 21:00:39
demo_sds	leo.lee	运行完成	测试通过	2018-07-06 21:00:39

查看执行结果

实践案例 - 性能测试平台

启动任务

选择项目
官网静态页面

选择目标
Release

选择压力机
AWS-VG

设置任务名称
官网静态页面 - Release - 7/6/2018, 9:11:12 PM (AWS-VG)

[启动任务](#) [查看脚本](#) [创建项目](#)

查看性能测试任务对应的
自动化测试用例

用例内容

序号	名称	类型	权重	创建者	创建时间
1	首页	api	100	■■■	2018年6月26日 10:03
2	访问404	testcase	5	■■■	2018年7月1日 16:50

用例组装



比例权重配置

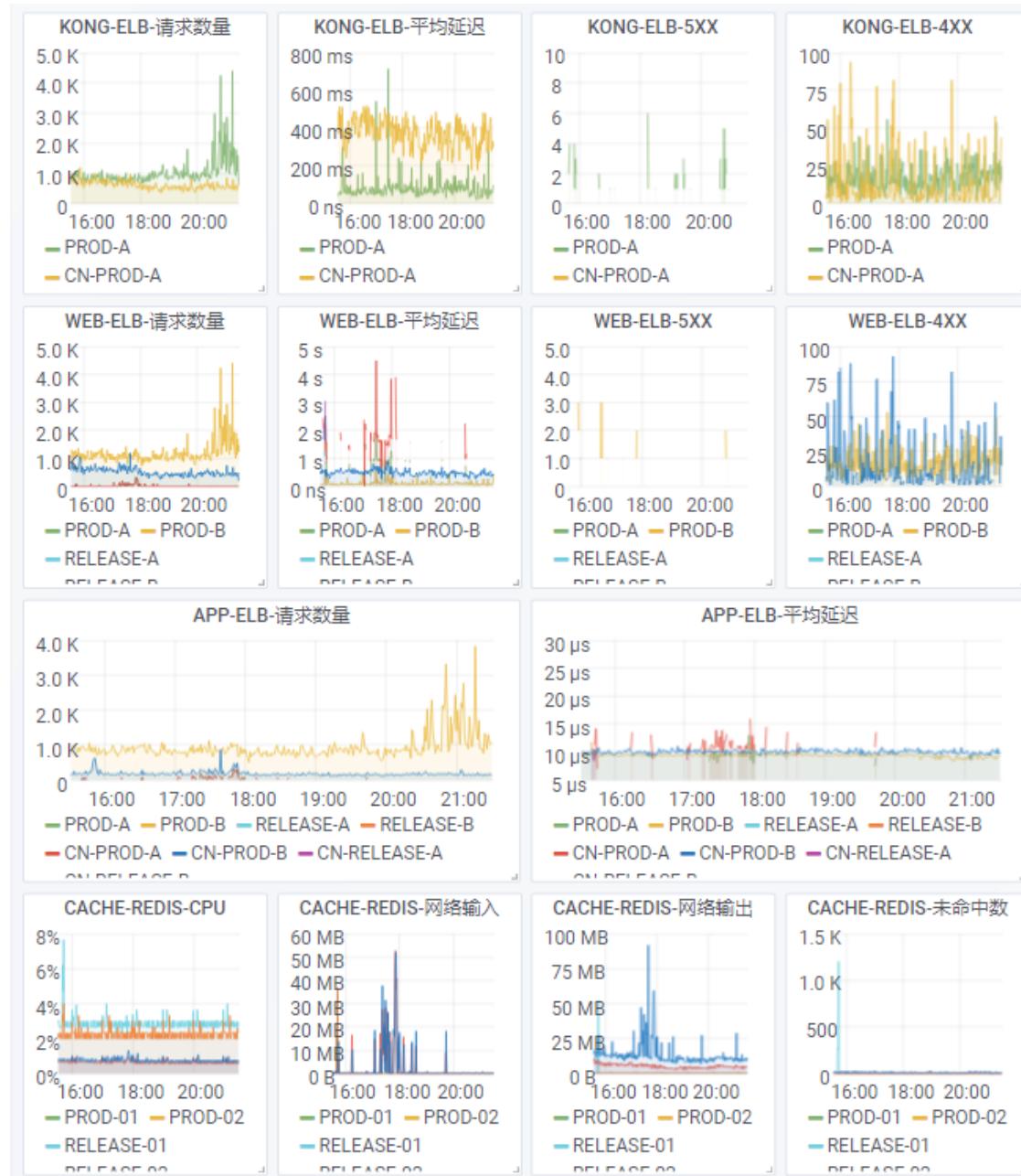
实践案例 - 性能测试平台



智能加压

性能数据
Grafana

New Relic



THANK YOU

You can find me here:

- Blog: <http://debugtalk.com>
- GitHub: <https://github.com/debugtalk>
- 微信公众号: DebugTalk

