



GOPS 2022
Shenzhen



深圳站

GOPS 全球运维大会

2022 - XOps 风向标

中国·深圳

指导单位：



主办单位：



时间：2022年4月21-22日

一站式测试平台

——低代码与云原生之路

王玺——虎牙

目录

Contents

1 从测试造数开始

2 低代码之路

3 一站式探索

4 云原生之路

5 总结与展望

/01

从测试造数开始



测试造数场景

功能测试前置条件：需要一个**用户账号**，关注了主播A，并且拥有5级粉丝牌



3 * 应用协议
5 * 关联的业务团队
7 * 步骤
9 * 接口请求



痛点分析

01

数据

1. 手工构造数据耗时
占比**>20%**
2. 贴合业务场景的数据**构造难**

02

工具

1. 面对复杂业务流程,
缺乏系统性抽象
2. **多样化的场景(协议)**
类型传统接口工具
不足以支撑

03

脚本

1. 对测试人员要求高
2. 脚本**可复用性差**
3. 脚本**可维护性差**

04

协作

1. 测试数据**重经验性**
2. 测试服务化? **开发**
+部署+运维
3. 测试团队的**重复开**
发



Codeless

- 操作以UI交互为主
- 在接口、自动化工具上实现
- 可视化编排

Coding

- 操作以编码为主
- 统一的框架下进行编码
- 纯代码的逻辑编排

	优势	劣势
Codeless	<ol style="list-style-type: none">1. 纯ui操作，易上手2. 人员成本低	<ol style="list-style-type: none">1. 复杂场景的新功能拓展不易2. 复杂业务流程支持不佳
Coding	<ol style="list-style-type: none">1. 灵活性极高，适配各种场景	<ol style="list-style-type: none">1. 脚本维护成本高2. 对QA人员要求高

/02

低代码与云原生之路



总体思路——造数流程拆解

原子性操作

- HTTP接口请求
- RPC接口调用
- 数据库查询
- 数据库插入
- 随机生成
- 数据包截取、解析
-

流程编排

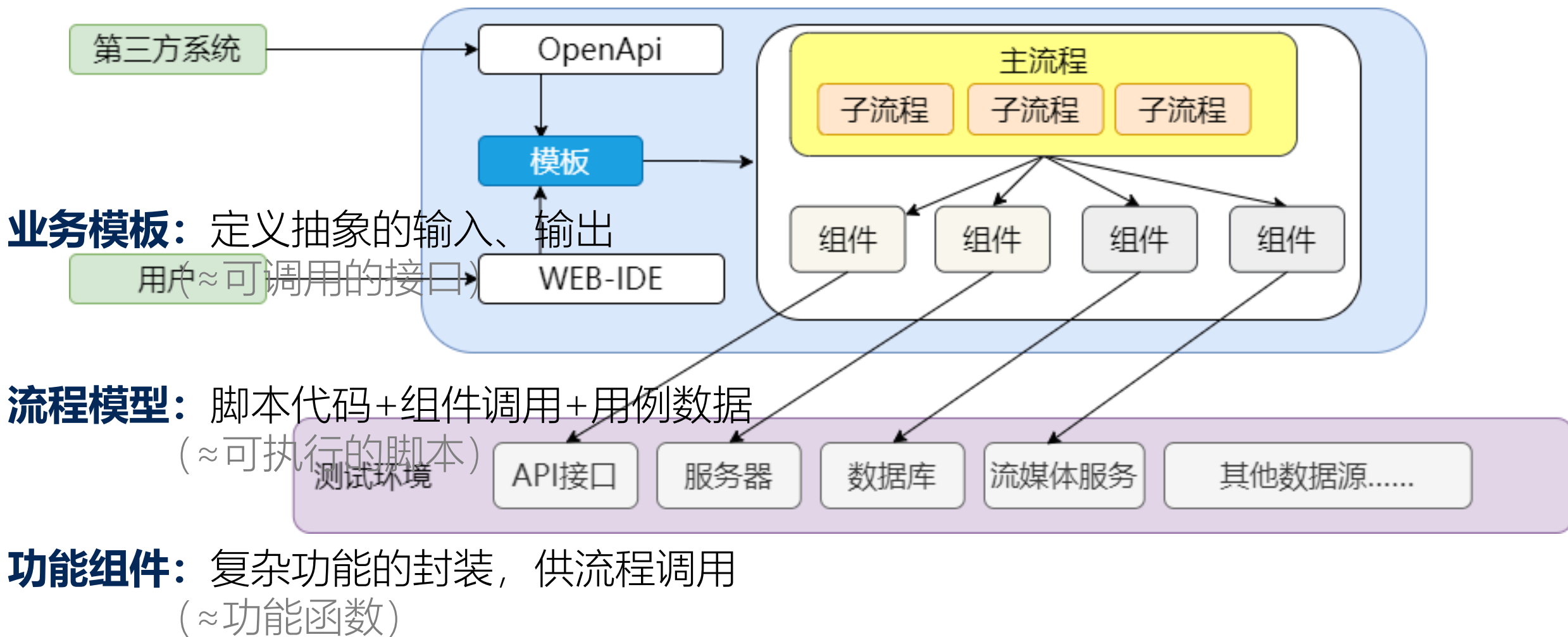
- 并行、串行
- 循环调用
- 选择判断
- 延时
- 异常跳过
-

易用性封装

- 提取入参、出参
- 枚举参数 (单选、多选)
- 参数默认值、参数校验
- 批量执行
- 界面调用 & 接口调用
-

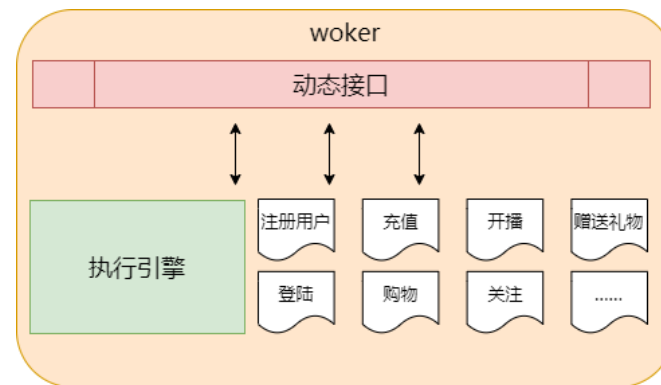
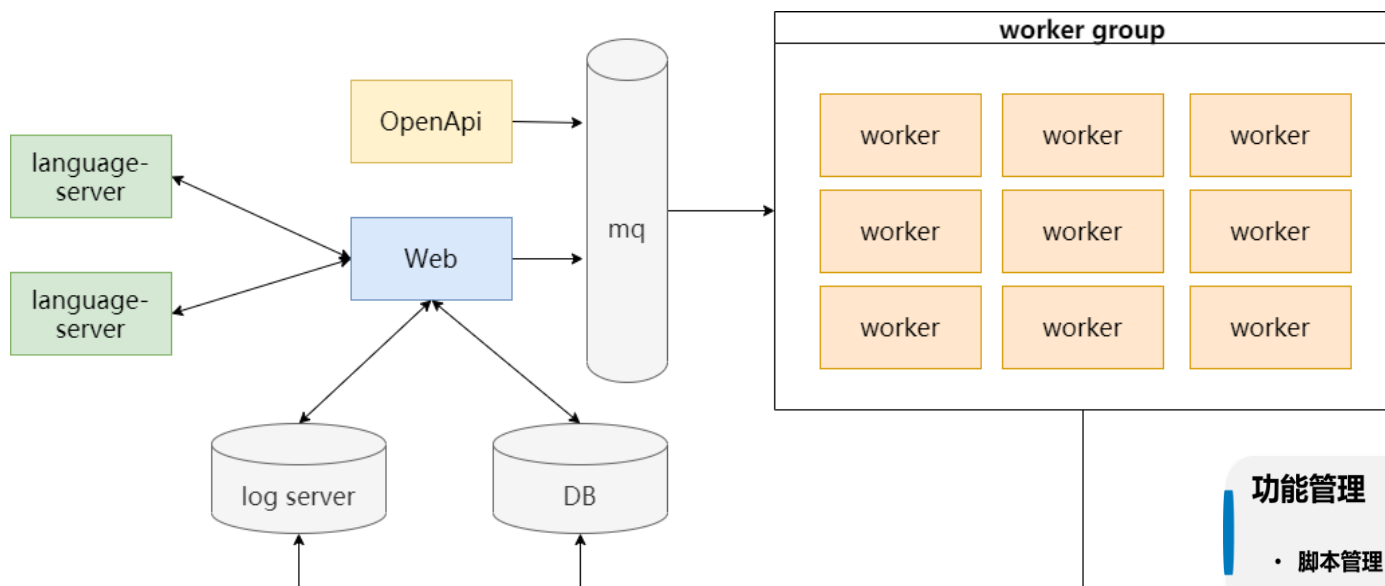


总体思路——调用流程





总体思路——早期测试引擎设计



- Web:** 前端，负责脚本创建、编辑等工作
- Language-server:** 为WEB-IDE提供语法分析等服务
- Worker:** 任务执行节点，无状态水平扩容
- OpenAPI:** 对外提供接口

功能管理

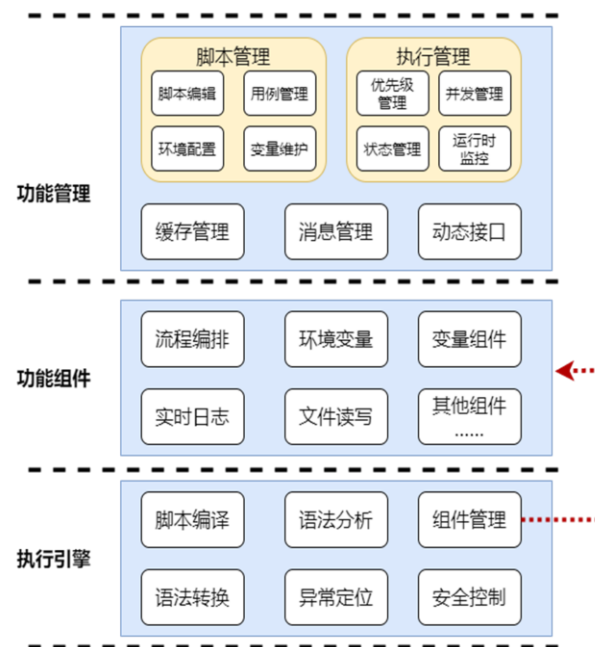
- 脚本管理
- 执行管理

功能组件

- 系统核心组件
- 其他功能组件

执行引擎

- 脚本引擎
- 组件系统





使用演示

任务管理

数据准备

任务列表

模板管理

业务流程

组件管理

信息维护

项目测试

模板入参测试相关 ☆

nimo

nimotest ☆

账号

注册用户并修改密码 (自动生成手机号) ☆

注册用户并修改密码 (用户自行输入手机号) ☆

SetHyc222 ☆

账号关系

虎牙币

设置虎牙币 ☆

虎牙币查询 ☆

减少虎牙币 ☆

增加虎牙币 ☆

金豆 / 银豆

包裹礼物

贵族

贵族1

用户是否为贵族 ☆

贵族g

用户是否为贵族1 ☆

用户是否为贵族-sjy ☆

贵族判断 ☆

guizu12

用户是否为贵族123 ☆

贵族*

用户是否为贵族2333 ☆

订单

注：动图内容仅为演示时创建的测试数据



使用演示——组件配置录入

录入HTTP接口

录入RPC接口

录入数据库配置

数据库组件详情

★ 名称:

▼ dev

★ 类型:

MySQL数据库

▼ dev2

★ 类型:

▼ dev3

★ 类型:

▼ dev4

★ 类型:

基本设置

* 接口名称:

demo_registerUser

* 选择分类:

公共分类

* 接口路径 ②:

GET

/api/userRegister

Tag:

请选择 tag

状态:

未完成

请求参数设置

添加Query参数

demo_username

必需

参数示例

demo_pwd

必需

参数示例

demo_type

必需

参数示例

Query

Headers

返回

编辑

测试

编辑

测试

编辑

测试

编辑

测试

编辑

添加一个数据库配置

上传IDL接口文件

在接口平台上录入



使用演示——流程编写，用例录入

编写流程脚本

录入流程用例

调试执行

★ 流程名称: PPTDemo

流程编辑 流程用例 执行用例 使用文档 快捷键列表 保存

```
9
10 @Override
11 void test() {
12     //注册用户-http
13     def resp = pf.http.requestForObj("user-service", "apiUserRegisterGET")
14     assert resp?.code == 0: "用户注册失败: ${resp}"
15     pf.variable.set("uid", resp?.data.uid)
16     log.info("用户uid: {}", "${uid}")
17 }
18
```

日志控制台 文本对比(格式化) 结束流程

日志 69007 x

```
2021-09-28 11:50:39:606 [INFO] =====flow PPTDemo start=====
2021-09-28 11:50:39:612 [INFO] http接口【user-service-apiUserRegisterGET】开始请求
2021-09-28 11:50:39:743 [DEBUG] http[user-service][apiUserRegisterGET]Request: {"headers":{"Accept":["*/*"],"Content-Type":["application/json"],"User-Agent":[""],"url":["https://192.168.1.100:8080/api/userRegister?demo_username=user-090931&demo_pwd=123456&demo_type=0","type":null]}
2021-09-28 11:50:39:747 [DEBUG] http[user-service][apiUserRegisterGET]Response: {"headers":{"Content-Type":["application/json; charset=utf-8"],"Content-Length":["123456"],"Access-Control-Allow-Origin":["*"],"Access-Control-Allow-Credentials":["true"],"Date":["Tue, 28 Sep 2021 03:50:39 GMT"],"Server":["APISIX/2.10.0"],"statusCode":["OK"],"statusCodeValue":["200]}
2021-09-28 11:50:39:750 [INFO] http接口【user-service-apiUserRegisterGET】请求成功
2021-09-28 11:50:39:754 [INFO] 用户uid: 123456
```

pf.http.requestForObj("user", "userRegister")

pf含义为public function, 即**组件方法**

声明式调用组件
接口参数 (表头), 按需填写用例
即时编译, 实时日志



使用演示——模板提取并使用

提取输入输出参数

界面手动调用

接口调用

根据手机号注册用户(RegisterByPhone)

模板描述: 1、
2、
3、

任务名称: 请输入
(自动)

环境: 研测

执行次数: 1

日志等级: info

执行方法: 串

Api文档: 同步

最后一次执行

POST /api_sync/RegisterByPhone 根据手机号注册用户

1、用户自行输入手机号
2、用户输入密码
3、用户注册

参数	类型	描述
password	String	密码
phone	String	手机号
appld	String	应用

参数	类型
phone	String
password	String
uid	String

Parameters

Name	Description
------	-------------

参数编辑

执行结果

结果

序号	id	phone	password	uid	状态
1	17349697				成功

日志

文本对比(格式化)

结束流程

```
2021-09-28 12:08:52:402 [INFO] =====flow Register start=====
2021-09-28 12:08:52:418 [INFO] 手机号码是: 19
2021-09-28 12:08:52:420 [INFO] http接口【】开始请求
2021-09-28 12:08:52:646 [INFO] http接口【】请求成功
2021-09-28 12:08:52:649 [INFO] http接口【】开始请求
2021-09-28 12:08:52:837 [INFO] http接口【】请求成功
2021-09-28 12:08:52:838 [INFO] 登录生成的uid: 119
2021-09-28 12:08:52:841 [INFO] http接口【】开始请求
2021-09-28 12:08:52:931 [INFO] http接口【】请求成功
2021-09-28 12:08:52:932 [INFO] 修改密码成功
```

保存快照

执行

系统从脚本中解析参数，按需提取输入输出参数并记录类型、默认值、校验



低代码组件——使用方式（配置+调用）

Json提取

JSON Extractor

Name:

Comments:

Apply to:

☐ Main sample and sub-samples ☒ Main sample only ☐ Sub-samples only ☐ JMeter

Names of created variables:

JSON Path expressions:

Match No. (0 for Random):

Compute concatenation var (suffix _ALL): ☐

Default Values:

正则匹配

Regular Expression Extractor

Name:

Comments:

Apply to:

☐ Main sample and sub-samples ☒ Main sample only ☐ Sub-samples only ☐ JMeter Variable Name to use

Field to check

☒ Body ☐ Body (unescaped) ☐ Body as a Document ☐ Response Headers ☐ Request Headers

Name of created variable:

Regular Expression:

Template (\$i\$ where i is capturing group number, starts at 1):

Match No. (0 for Random):

Default Value: ☐ Use empty default value

SQL查询

SQL Query

Query Type:

Query:

1

Parameter values:

Parameter types:

Variable names:

Result variable name:

Query timeout:

Limit ResultSet:

Handle ResultSet:



```
def resp = pf.http.requestForString("user-service", "re
def title= pf.json.path(resp,"$.store.book[0].title")//
pf.regex.matchAll(title,"(?<=body).+(?=hu)")//regex
pf.mysql.execute("db1","select * from xxx")//sql
```

```
// }
@Override
void test() {
}

// @Override
// void after() {
```



低代码组件——生成式数据表驱动

传统编写一个http请求需要传入哪些参数？

```
axios({  
  method: 'post',  
  url: '/user/12345',  
  data: {  
    firstName: 'Fred',  
    lastName: 'Flintstone'  
  },  
  Headers: {  
    "agent": "xxxx"  
  },  
  Proxy: {  
    host: '127.0.0.1',  
    port: 9000  
  }  
});
```

header url query path body proxy.....

弊端：

脚本内代码繁琐，不易理解

多套测试环境，增加代码复杂度

多人协作，脚本难以复用



低代码组件——生成式数据表驱动

核心理念：命令式编程——》声明式编程

```
pf.http.requestForObj("user","userRegister")
```

+

	demo_username	demo_pwd	demo_type
注册普通用户	Fred	Flintstone	xxxx
注册VIP用户	Foo	Bar	xxxx
注册超级管理员	Fred	Flintstone	xxxx



自动生成参数

★ 流程名称: PPTDemo

流程编辑 流程用例

表头筛选

日志等级: debug

环境: dev

表格内搜索

	A	B	C	D	E	F	H	I	J	K
1				流程用例	用例描述	username	userType	HTTP: user-service->apiUserRegisterGET		
2								demo_username	demo_pwd	demo_type
3	运行	保存	×	PPTDemo_common_user	注册一个普通用户	user-090931	0	\${username}	\${pwd}	\${userType}
4	运行	保存	×	PPTDemo_vip_user	注册一个vip用户	user-090932	1	\${username}	\${pwd}	\${userType}
5	运行	保存	×	PPTDemo_black_user	注册一个黑名单用户	user-090933	2	\${username}	\${pwd}	\${userType}
6	运行	保存	×							

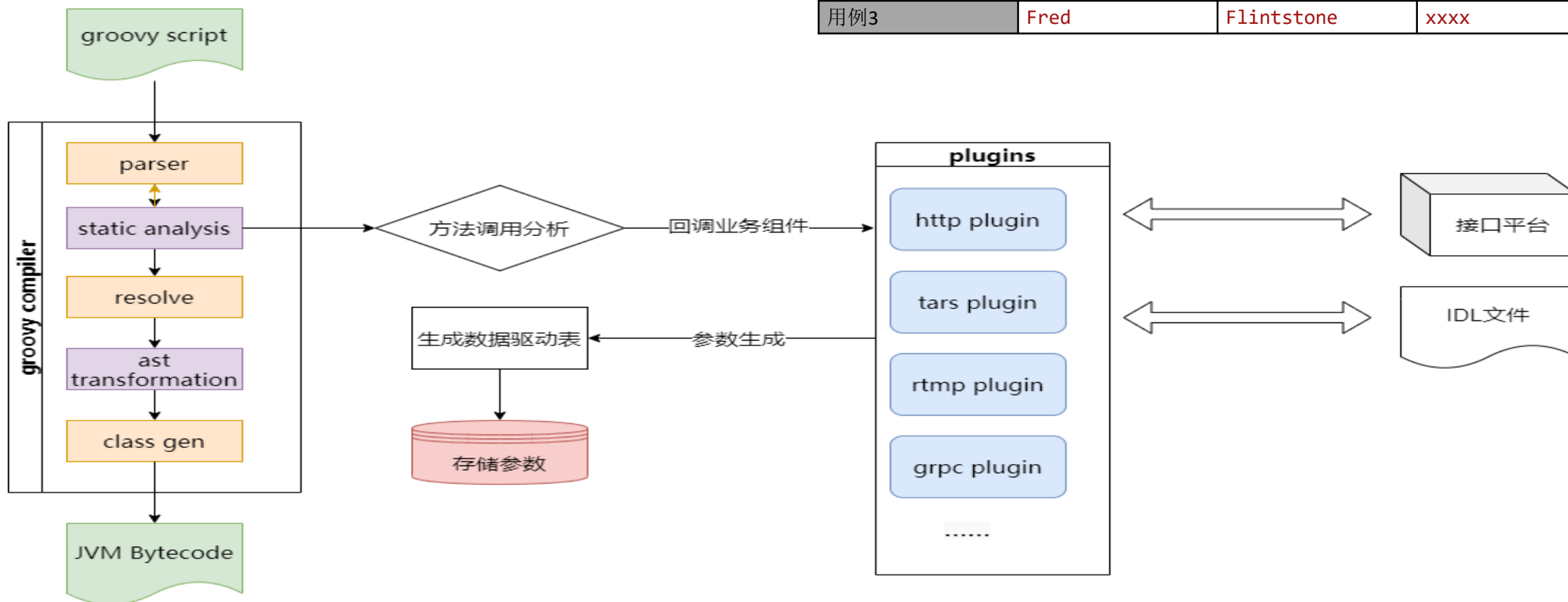


低代码组件——编译期的回调设计

如何从声明式的调用分析出请求参数?

```
pf.http.requestForObj("user","userRegister")
```

	demo_username	demo_upwd	demo_type
用例1	Fred	Flintstone	xxxx
用例2	Foo	Bar	xxxx
用例3	Fred	Flintstone	xxxx



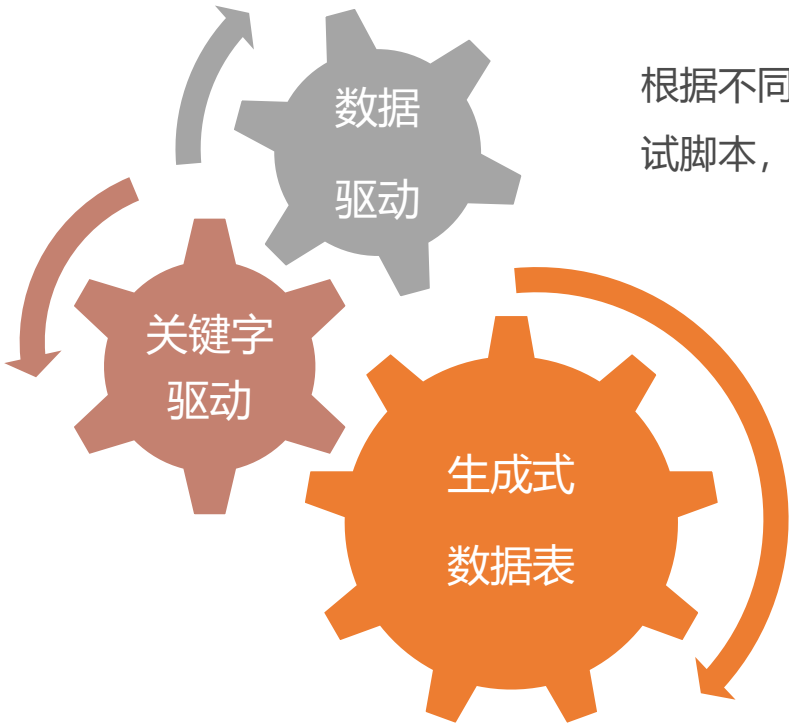


低代码组件——从数据驱动到关键字驱动

```
pf.http.requestForObj("user","userRegister")
```

	demo_username	demo_pwd	demo_type
注册普通用户	Fred	Flintstone	xxxx
注册VIP用户	Foo	Bar	xxxx
注册超级管理员	Fred	Flintstone	xxxx

在数据驱动的基础上，通过**关键行为（关键字）**拓展测试用例的模式，从而达到一个由数据和关键字驱动整个测试的效果



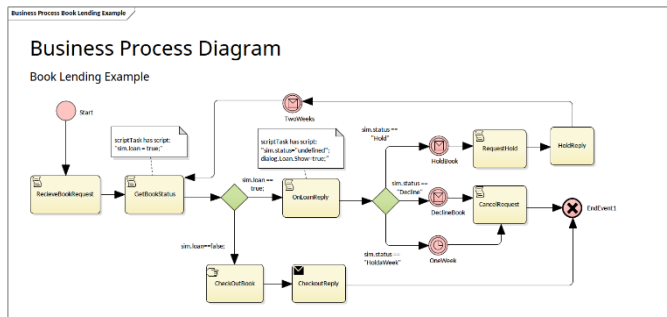
根据不同的测试环境，测试用例，运行同一测试脚本，即**测试逻辑与测试数据的分离**

由系统自动完成参数化。测试同学通过声明式的编程，从而**更专注于当前测试逻辑的完成**

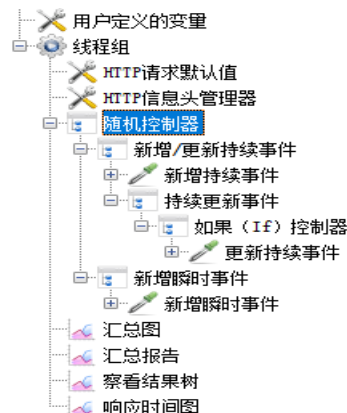


低代码编排——常见的一些流程编排模式

Workflow引擎界面拖拽式编排



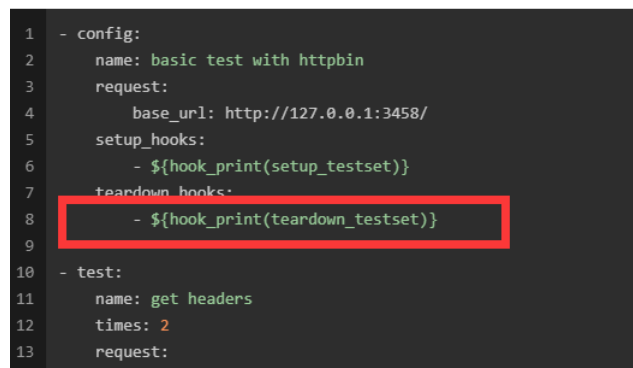
Jmeter控制器编排



Postman接口前后置编排



HttpRunner钩子函数式编排





编排——低代码流程编排带来的改变

传统流程编排样例

主播B开播并推流

主播B开启pk模式



主播A发起邀请

主播B响应邀请

主播A开播并推流

主播A开启pk模式

- 复杂流程编排缺乏灵活、拓展
- 执行过程黑盒，难以debug
- 新功能的开发成本高
- 对 workflow 界面的适应学习成本

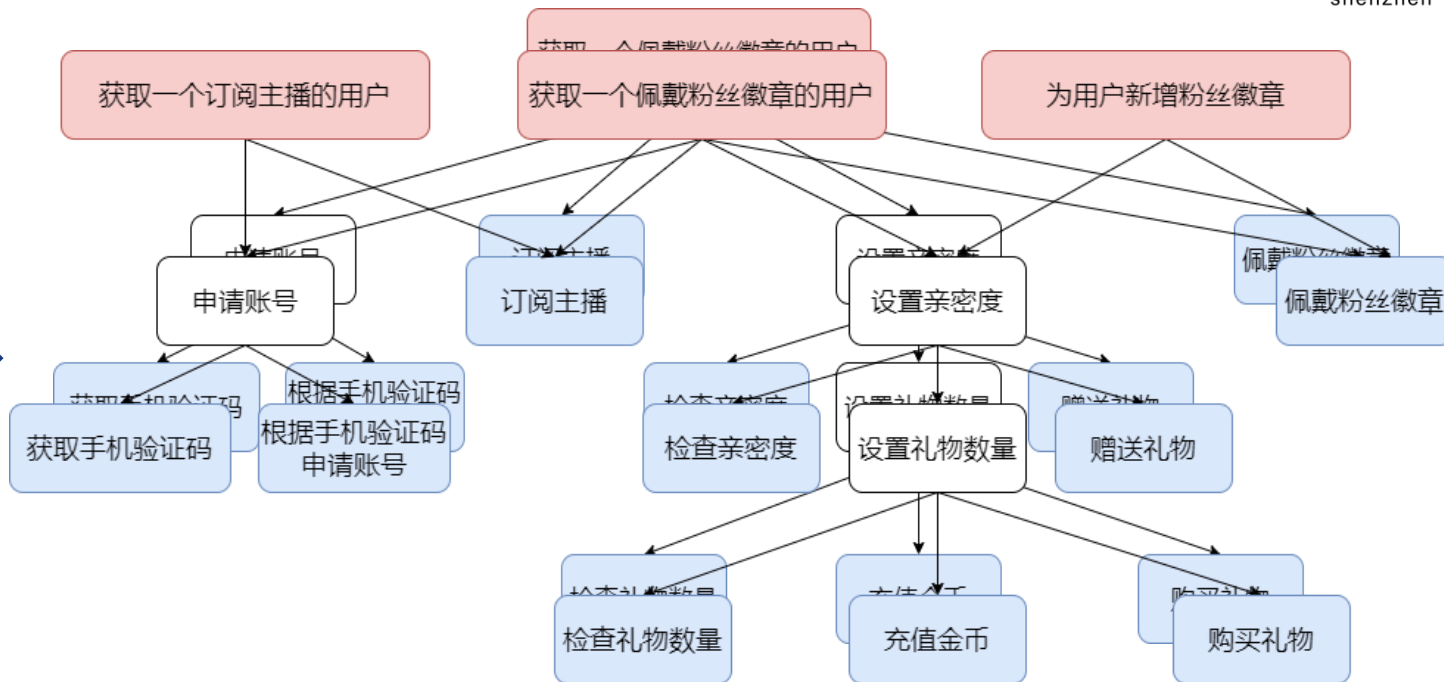
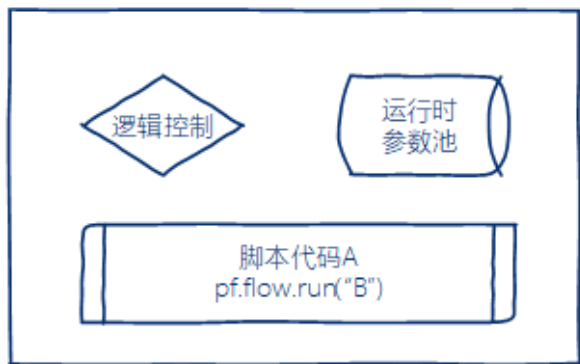
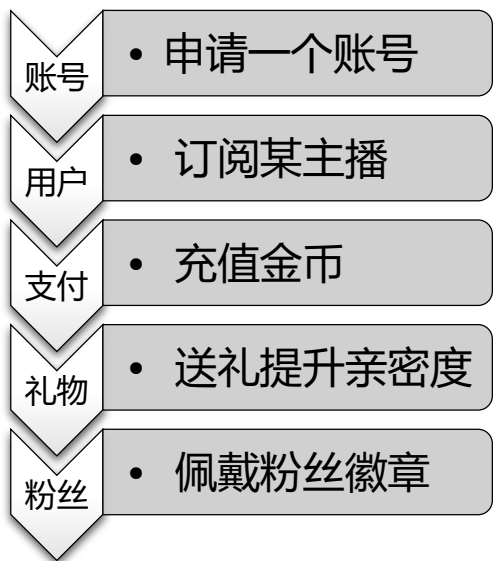
低代码编排样例

```
pf.flow.parallel(  
  {  
    pf.flow.runByAlias("BeginLiveAndPushStream", "BeginLive_A")  
    pf.flow.runByAlias("StartPKMode", "Start_A") },  
  {  
    pf.flow.runByAlias("BeginLiveAndPushStream", "BeginLive_B")  
    pf.flow.runByAlias("StartPKMode", "Start_B") })  
if("${Start_A.status}"=="${Start_B.status}==0"){  
  pf.flow.run("InviteMultiPK") //邀请  
  pf.flow.run("ResponseMultiPKInvite") //接收  
}
```

- 原生代码编排，简单易上手
- 代码执行，可以打日志or调试
- 更低的组件开发成本
- 会写代码就能上手



低代码编排——标准流程模型相关



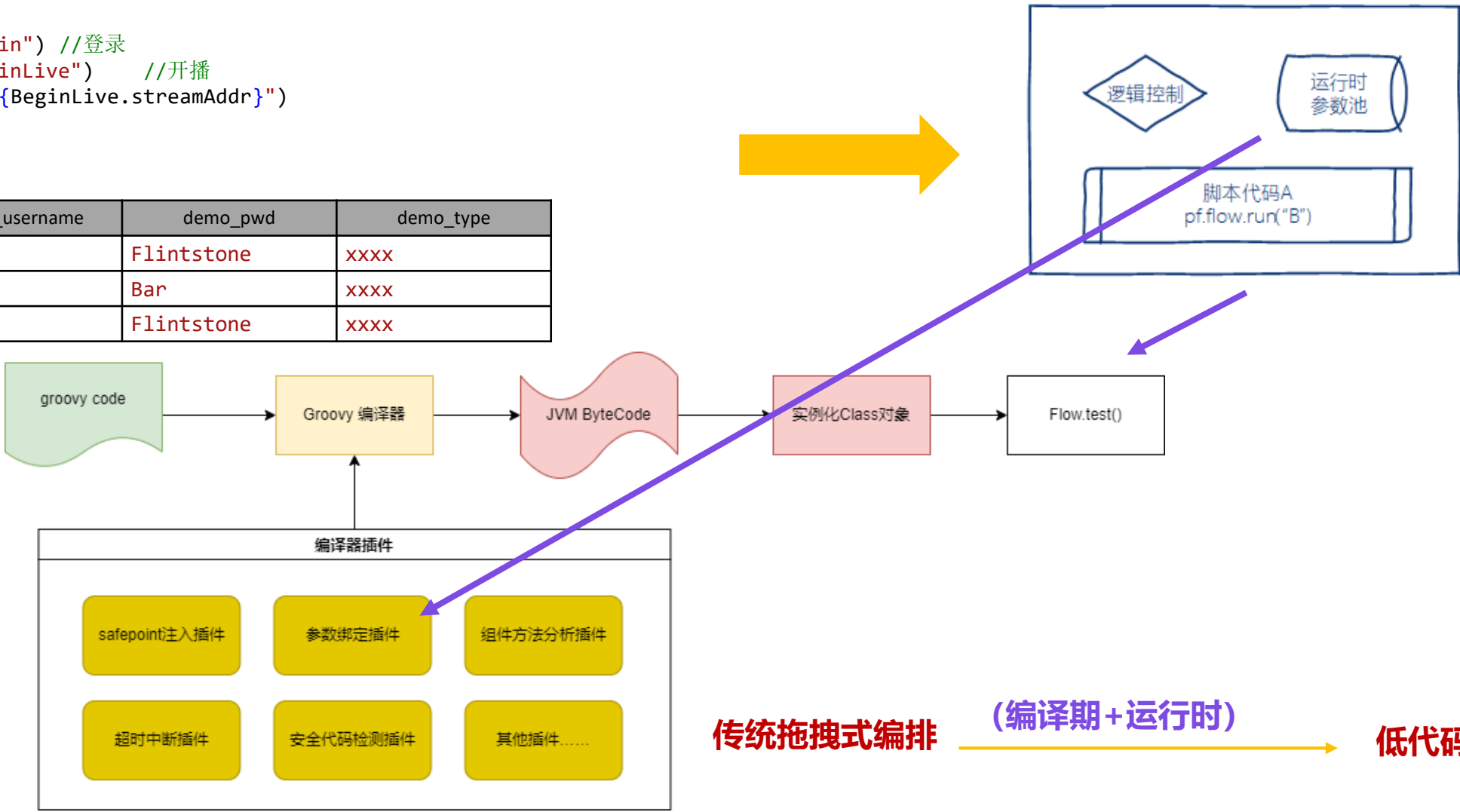
复杂流程由**基础流程**组合而成，
流程间相互**复用**，打造贴合业务的**流程生态库**



低代码编排——标准流程模型对业务逻辑的封装

```
class BeginLiveAndPushStream extends Flow {  
    @Override  
    void test() {  
        pf.http.run("Login") //登录  
        pf.flow.run("BeginLive") //开播  
        pf.rtmp.push("${BeginLive.streamAddr}")  
        //调用推流组件  
    }  
}
```

	demo_username	demo_pwd	demo_type
普通用户开播	Fred	Flintstone	xxxx
VIP用户开播	Foo	Bar	xxxx
黑名单用户开播	Fred	Flintstone	xxxx





低代码编排——更深度的流程控制能力

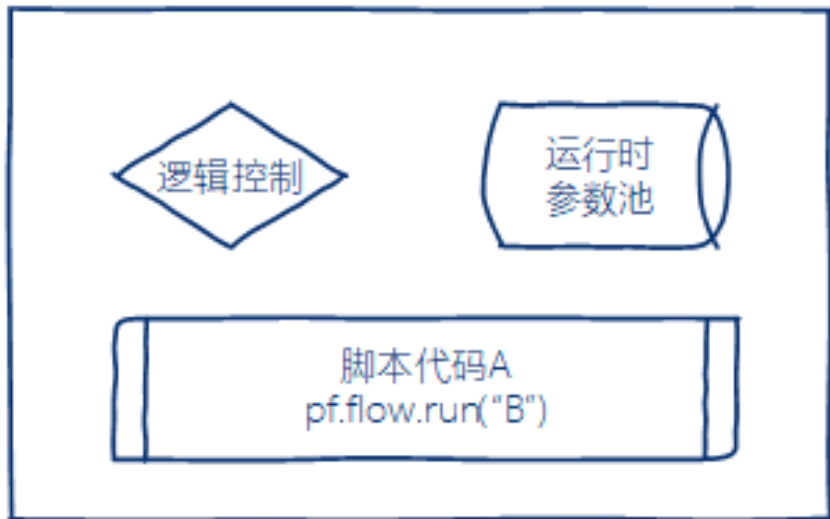
代码脚本在多人协作遇到的痛点

- **逻辑与数据耦合**：多场景下难以复用，参数化的习惯难以养成
- **统一的编排**：个人编码习惯不同，代码风格迥异，难以进行

```
pf.flow.run("BeginLive")
def streamAddr = pf.variable.get("BeginLive", "streamAddr")
//调用推流组件
if(streamAddr){
    pf.rtmp.pushImage(streamAddr)
}
for (def uid in [123,456,788,960]){
    pf.flow.runWithVar("BeginLive", [{"phone": uid}])
} //循环给多人开播
```

①代码逻辑与数据的分离

③无侵入地修改子流程参数



	BeginLive	phone	pwd
用例1	用户A开播	1881959	123
用例2	用户B开播	1881958	456

②静态分析代码后自动完成参数化

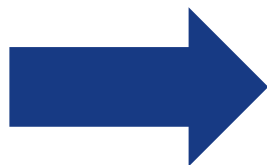
★ 流程名称: BeginLive

	A	B	C	D	E	F	G	H	I	J
1				流程用例	用例描述	Flow: GetUdbToken	phone	pwd		TAF: liveui->HUYA.liveui->
2						GetUdbToken				In BeginLiveReq tReq
3	运行	保存	×	BeginLive_success		GetUdbToken_common	18819592020	abcd1234		"ISubSid": 0,
4	运行	保存	×	BeginLive_pk1	pk1用户开播	GetUdbToken_common	18819592020	abcd1234		"ISubSid": 0,
5	运行	保存	×	BeginLive_pk2	pk2用户开播	GetUdbToken_common	18853290289	abcd1234		"ISubSid": 0,
6	运行	保存	×	BeginLive_wfe_test_6672	wfe_test_6672开播	GetUdbToken_common	18888886672	123456		"ISubSid": 0,

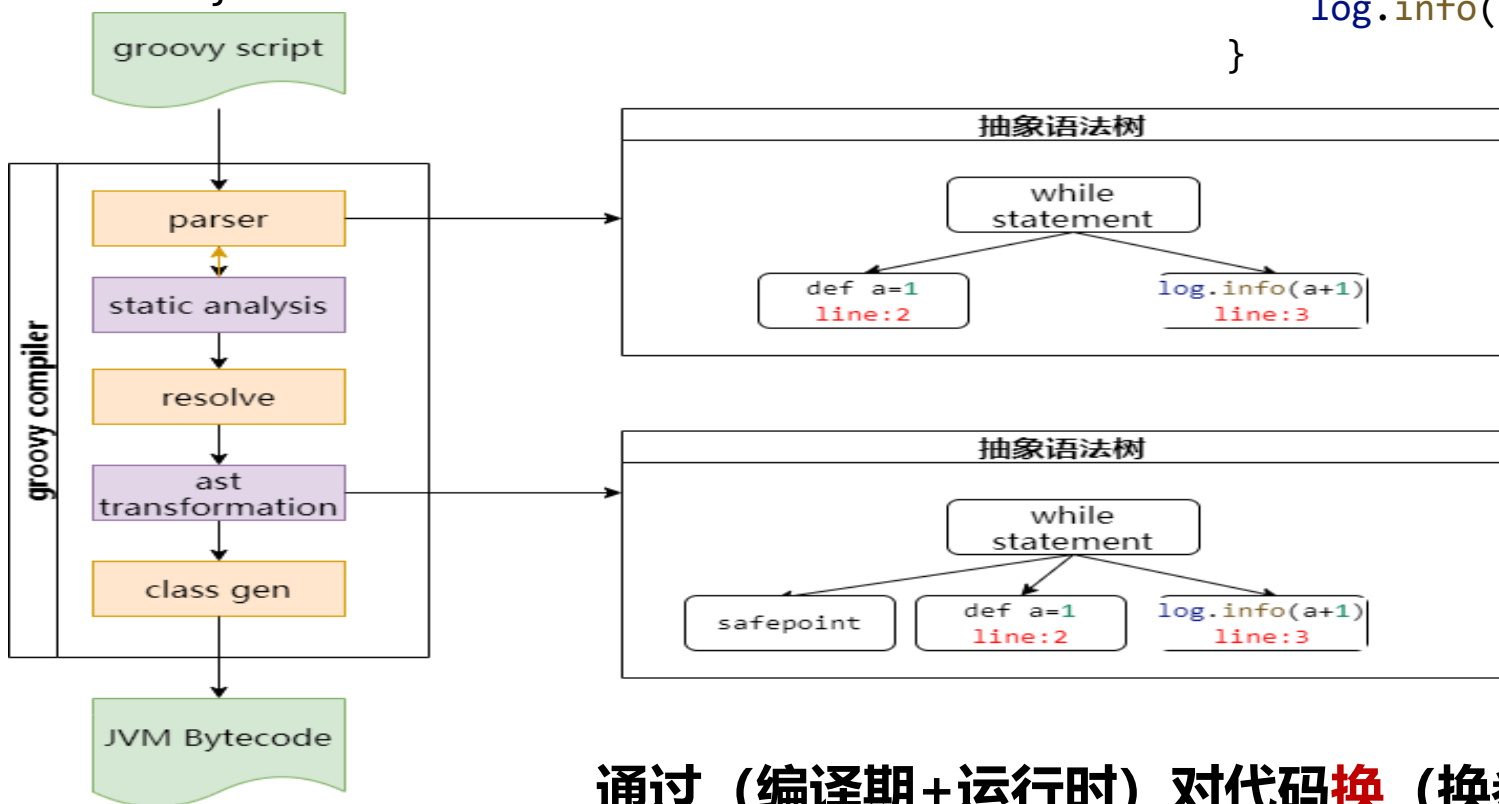


低代码编排——“编译期” OR “运行时” 代码注入

```
while(true){  
  def a=1  
  log.info(a+1)  
}
```



```
while (true) {  
  if (Condition.shouldStop(Thread.currentThread())) {  
    Thread.currentThread().wait();  
  }  
  def a=1  
  log.info(a+1)  
}
```



稳定性需求

安全性保障

参数可替换

逻辑可修改

流程可编排

通过（编译期+运行时）对代码**换**（换参数）、**排**（编排流程）、**改**（改逻辑）

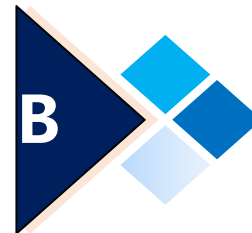


低代码编排——低代码编排小结



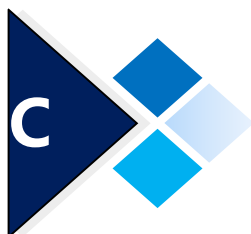
更灵活的编排方式

使用原生代码，自由度极高。
例如失败重试，循环，判断等



无侵入式流程控制

父流程可以在不改动子流程情况下完成对运行参数，运行逻辑的控制



封装实现细节

声明式调用，隐藏实现细节。
测试只需要专注测试逻辑即可



统一的流程模型

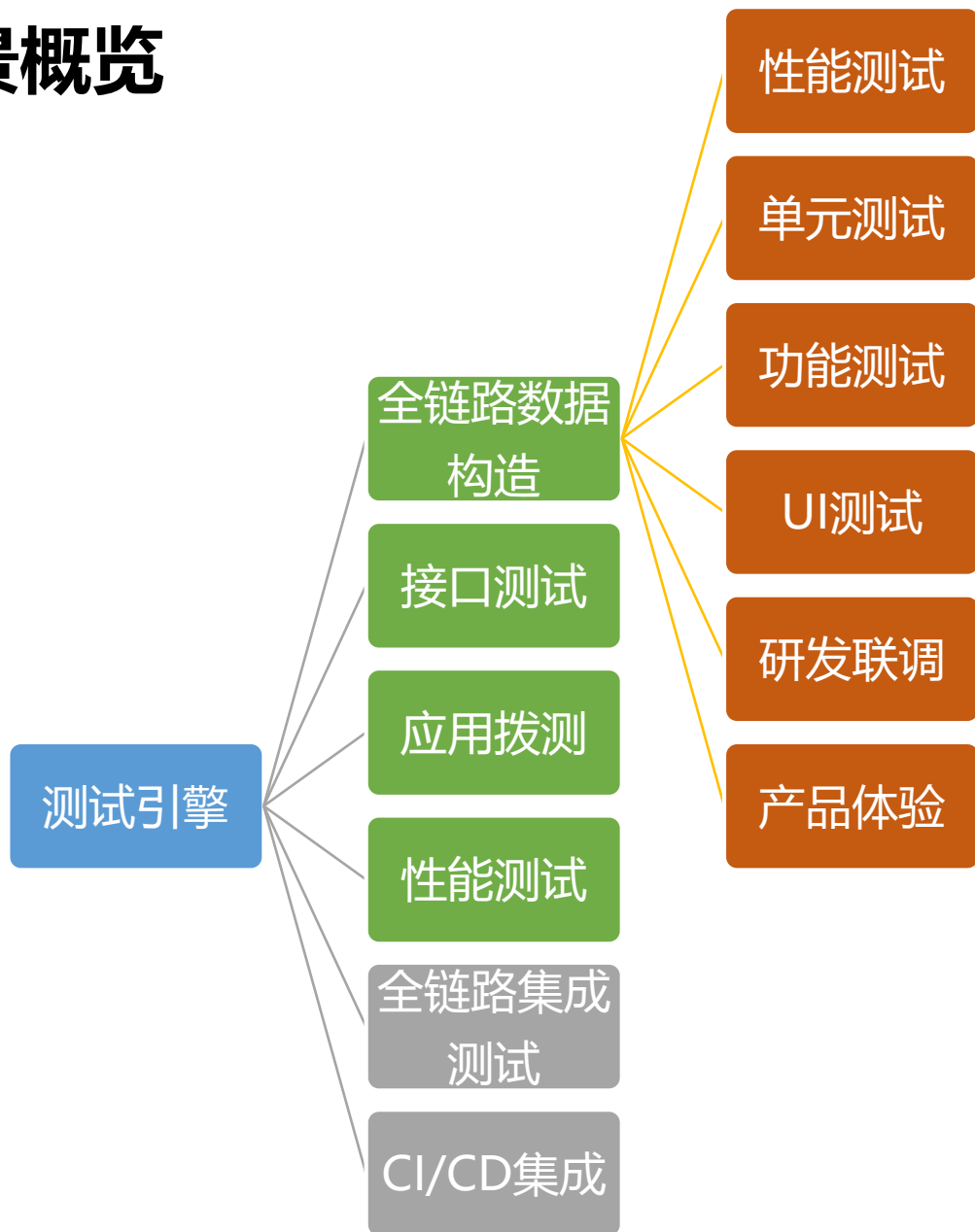
系统层面：方便后续功能拓展
脚本层面：可维护行极大提升

/03

一站式探索



一站式应用场景概览





复杂场景造数——获取N级粉丝徽章

(粉丝徽章升级或降级) 获取N级粉丝徽章并佩戴 (N>0) (GetNBadgeAndUsed)

参数编辑

执行结果

模板描述 (粉丝徽章升级或降级) 获取N级粉丝徽章并佩戴 (N>0)

任务名称:

(自动生成, 可不填)

环境:

执行次数:

日志等级:

执行方法: ☒ 串行 ☐ 并行

Api文档: [同步](#) | [异步](#)

最后一次执行

GetNBadge.targetBadgeLevel

targetBadgeLeve

文本

保存

POST

/api_sync/GetNBadgeAndUsed (粉丝徽章升级或降级) 获取N级粉丝徽章并佩戴 (N>0)

(粉丝徽章升级或降级) 获取N级粉丝徽章并佩戴 (N>0)

模板输入参数

参数

类型

描述

uid

String

用户uid

targetBadgeLevel

String

目标粉丝徽章等级

pid

String

主播uid

模板输出参数

参数

类型

描述

uid

String

用户uid

pid

String

主播uid (徽章id)

curBadgeLevel

String

当前粉丝徽章等级

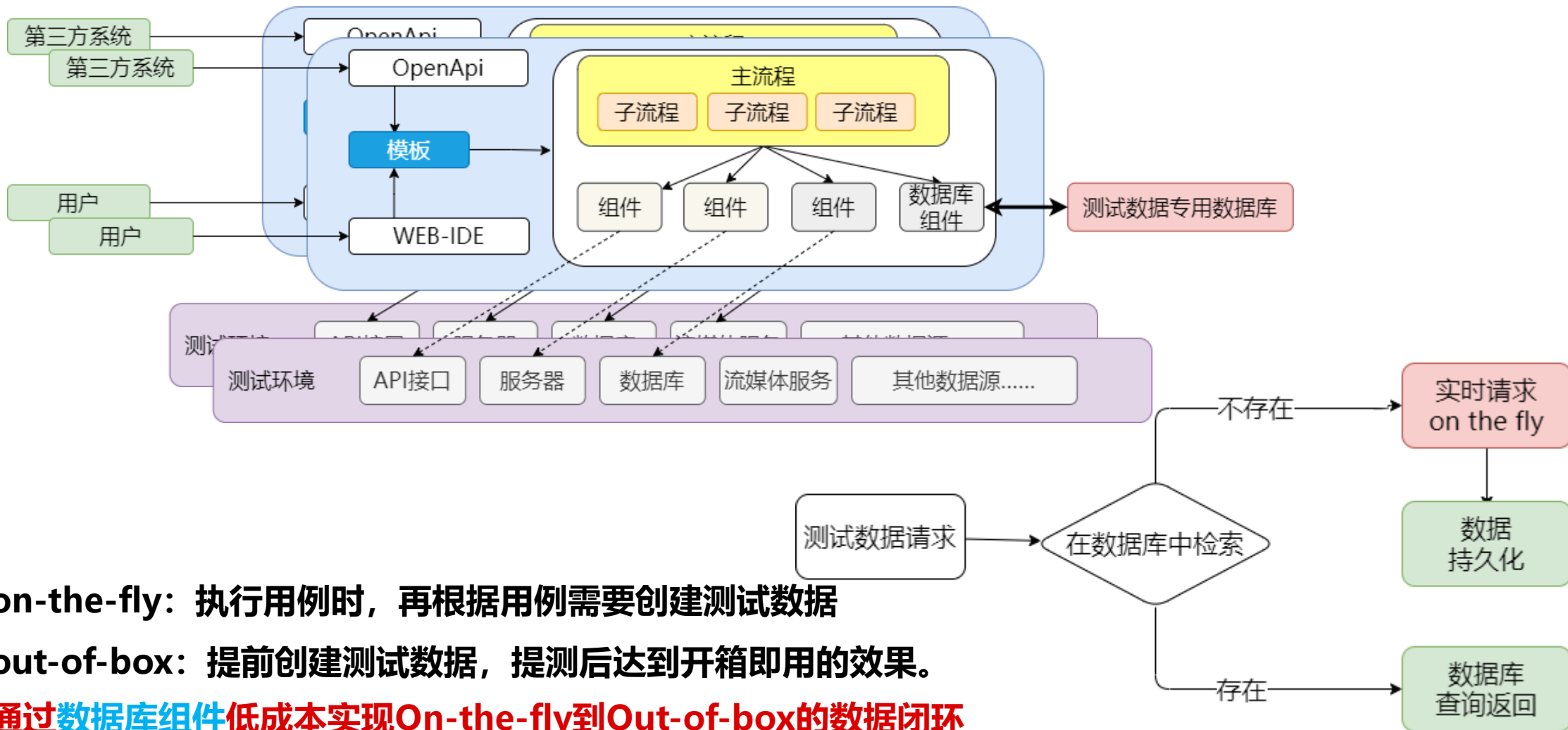
targetBadgeLevel

String

目标粉丝徽章等级



造数从On-the-fly到Out-of-box





接口自动化——用户登录接口

①接口调用+断言

②数据表驱动，编写用例

★ 流程名称: LoginServDemo

流程编辑

流程用例



```
1 package flow
2 // flow desc:
3 class LoginServDemo extends Flow {
4
5     @Override
6     void before() {
7         pf.variable.declare("user")
8     }
9
10    @Override
11    void test() {
12        def resp=pf.http.requestForObj("LoginServ","loginPOST")
13        assert resp?.ret==0
14    }
15 }
```

★ 流程名称: LoginServDemo

流程编辑

流程用例

表头筛选

日志等级: debug

环境: dev

表格内搜索

	A	B	C	D	E	F	G
1				流程用例	用例描述	user	HTTP: LoginServ->login POST
2							root
3	运行	保存	×	LoginServDemo_comm	普通用户登录	user_comm	"user": "\${user}", "pwd": "123456", "app": 17891
4	运行	保存	×	LoginServDemo_vip	VIP用户登录	user_vip	"user": "\${user}", "pwd": "123456", "app": 17891
5	运行	保存	×	LoginServDemo_black	黑名单用户登录	user_black	"user": "\${user}", "pwd": "123456", "app": 17891
6	运行	保存	×				

root

```
{
  "user": "${user}",
  "pwd": "123456",
  "app": 17891
}
```

声明接口



编写断言



录入用例

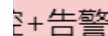


自动化任务



稳定性

2. 案例二：流程脚本+用例+定时执行+数据上报+自





性能测试

未专项开展前，测试同学自主实现的一个简单的压测脚本

```
void test
def t @Override
def t void test() {
//自
def resp=pf.http.requestForObj("LoginServ","loginPOST")
Exec assert resp?.ret==0
for t
    executorService.execute(new Runnable() {
        void run() {
            try {
                pf.taf.requestForObj("HiCompare", "JMG.F
            } catch (Exception e) {
                log.error("", e)
            }
        }
    })
}
executorService.shutdown();
executorService.awaitTermination(10, TimeUnit.HOURS)
```

*最大并发持续时长: 120

分钟

500

0

0 300 600 900 1200 1500 1800 2100 2400 2700 3000 3300 3600 3900 4200 4500 4800 5100

预计压测总时长 (包含压力增长时长): 02:18:00

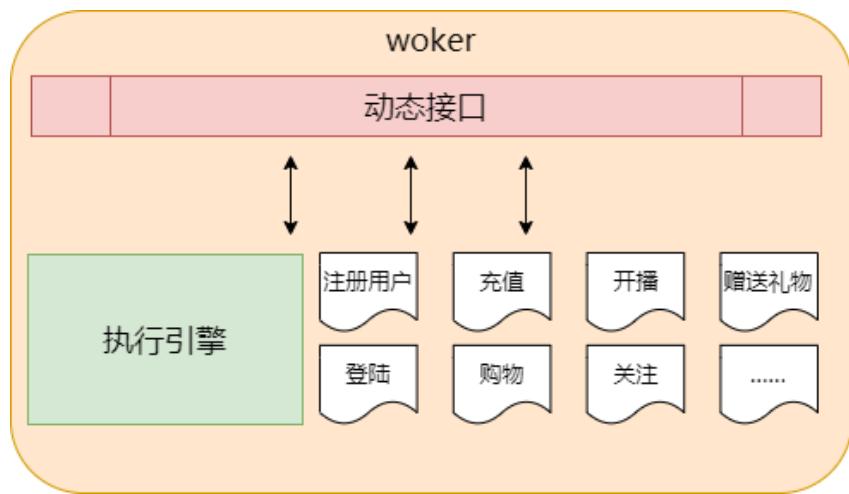
标准流程模型让一份脚本同时实现全链路造数+接口测试+应用拨测+性能测试

/04

云原生之路



线程隔离模式遇到一些问题



- 直播推流
- 压测脚本
- WebSocket推送
- 其他长时间执行的测试任务

用例间隔离

① 脚本代码对系统的影响

② 对系统资源的占用

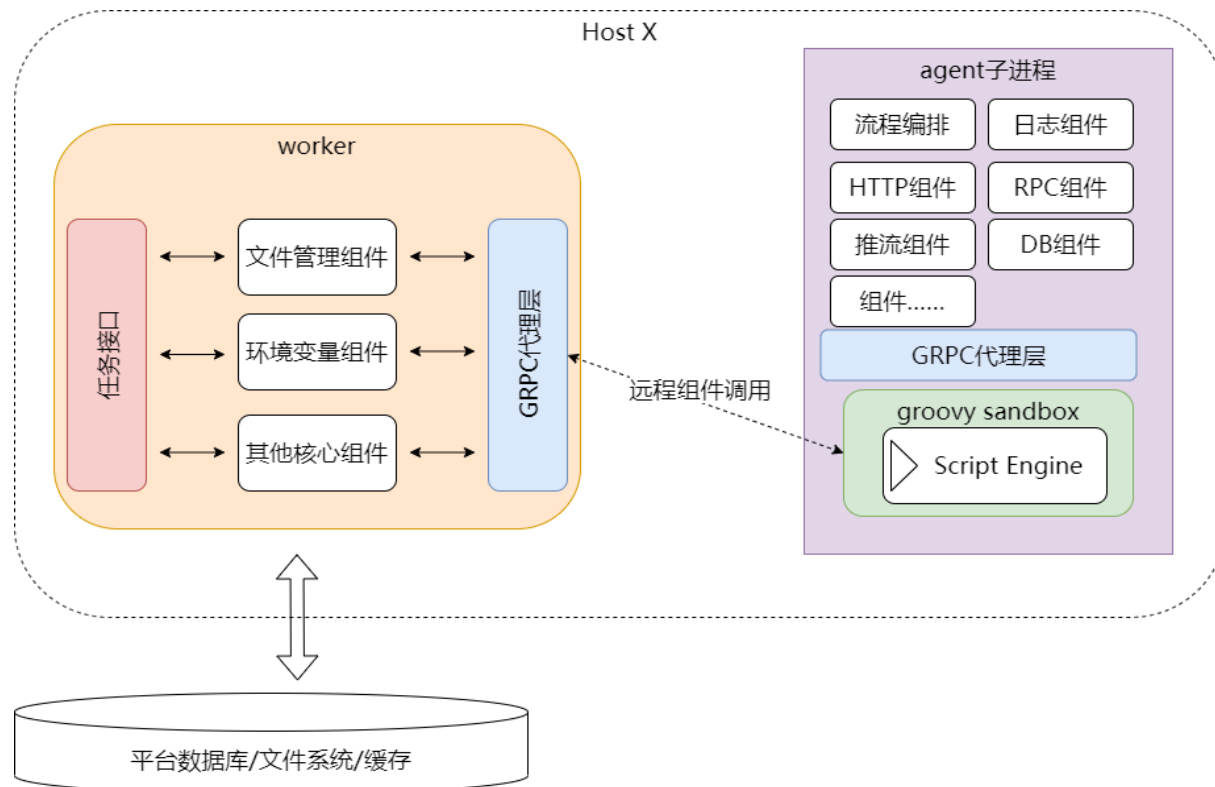
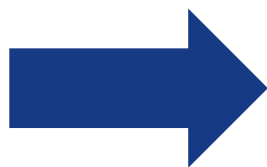
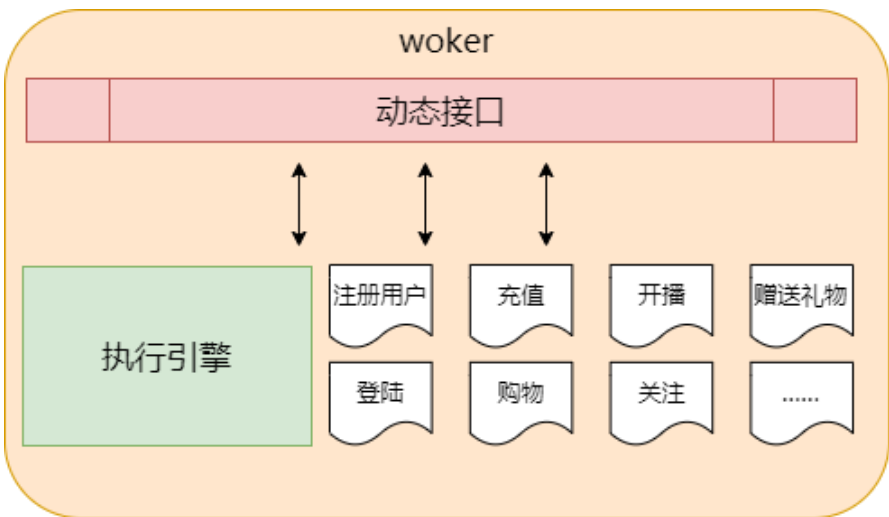
系统对用例的影响

① 主机资源，影响任务稳定行

② 发布导致长时间执行任务中断



从线程隔离到进程隔离的重构



也发掘出了一些令人激动的新特性:

动态类库加载

```
java -cp xxxxxx.jar;
```

低代码Debug

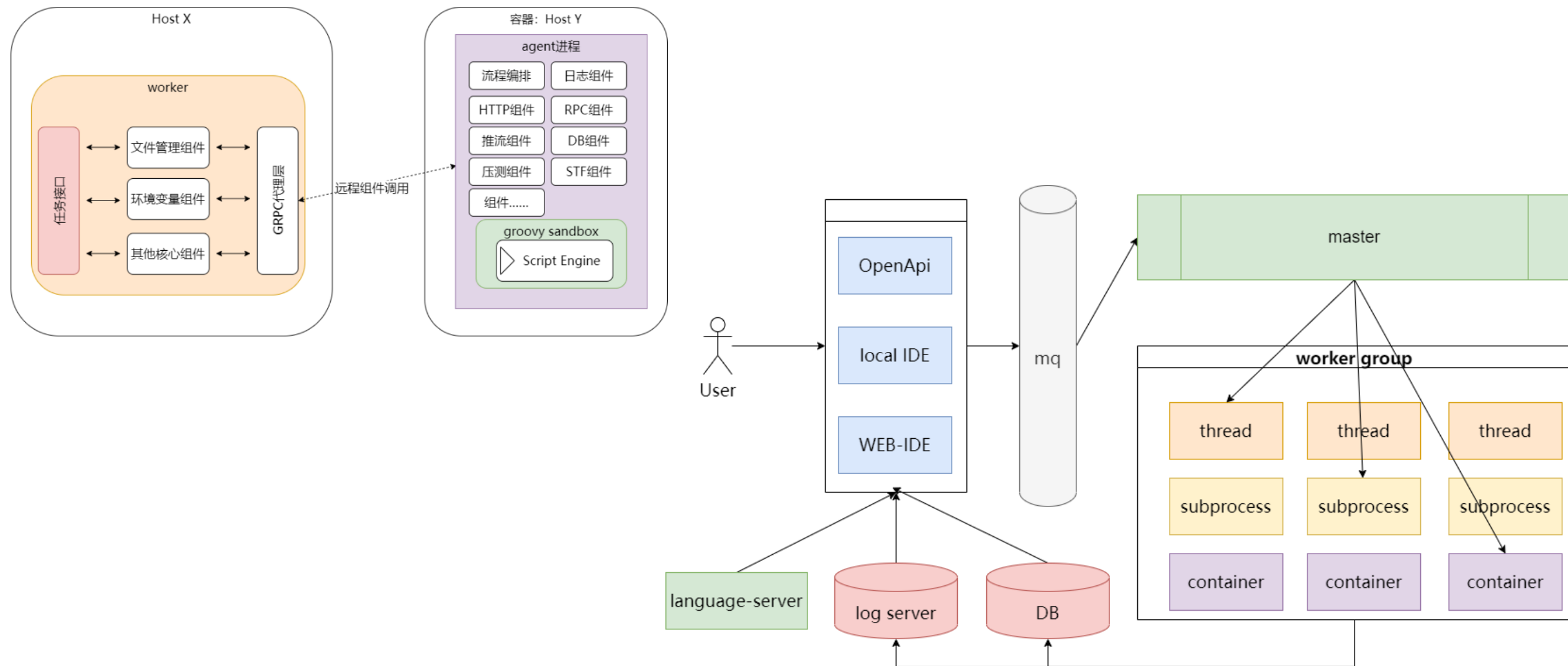
```
jdwp=transport=dt_socket,address=*:8080
```

Serverless

多语言混合编排



向ServerLess Job演进

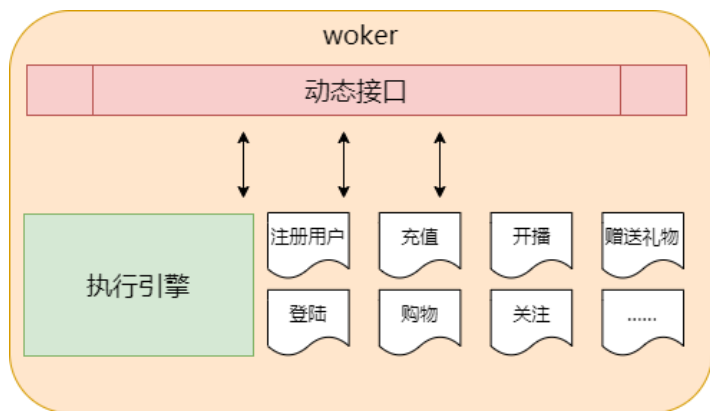




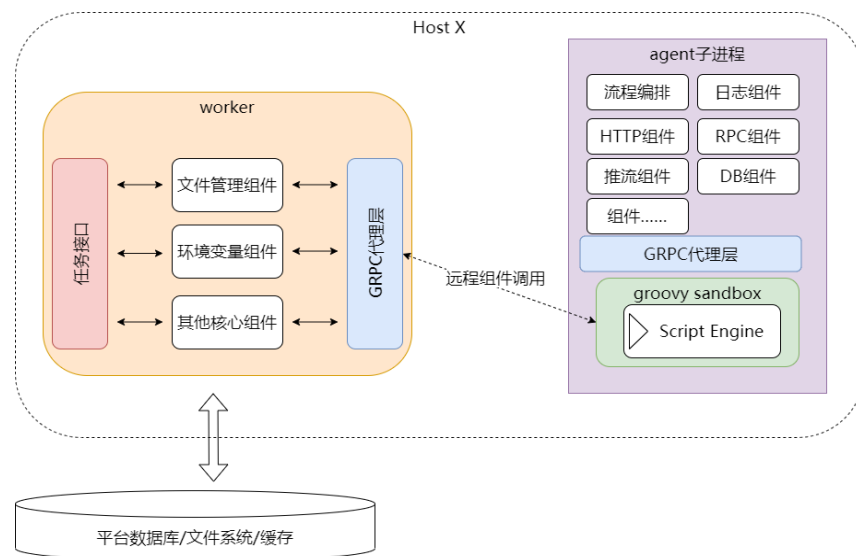
演进之路



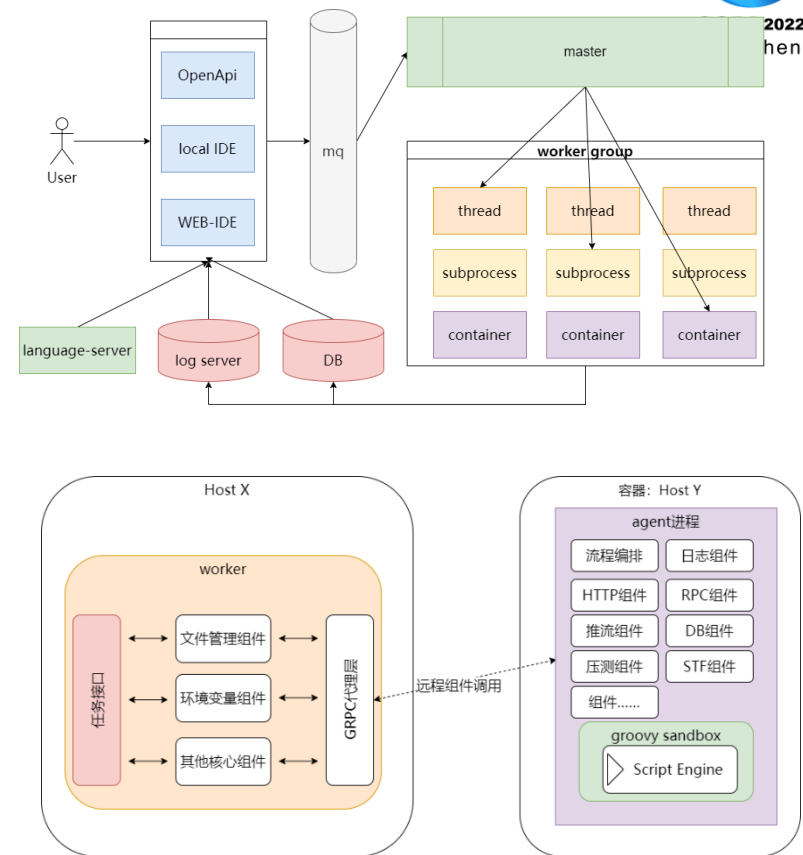
2022
hen



线程隔离



进程隔离



镜像/容器隔离

/05

总结与展望



体系概览

可观测

metric
上报/告警

log采集

trace注入

任务调度

状态管理

资源管理

场景接入

全链路
数据构造

接口测试

应用拨测

性能测试

全链路
集成测试

CI/CD
集成

开发&使用

web-ide

本地调试

语言
服务器

ide插件

类库加载

运行时
profiling

低代码生成

环境配置

组件配置

模板
接口抽象

动态
代码生成

动态
界面生成

动态
接口生成

部署&运行

源码托管

脚本编译

异常定位

语法
分析转换

语法级
安全控制

多语言
混合编排

serverless运行

负载均衡

自动部署

线程级
隔离&运行

节点注册

进程级
隔离&运行

镜像/容器
隔离&运行



工具&平台的价值落点?



效率

语言语法层面抽象、低
代码生成



标准

定义标准流程模型、组
件模型



开放

原生语言语法



低代码测试引擎技术交流(712523537)

虎牙一直践行“技术驱动内容”，
在实时内容创作与直播互动技术上持续创新，
为业务赋能

发布直播数字人和小程序开放平

4K超分、AI打点实时回放功能上

推出虚实同台互动直播



Thanks

开放运维联盟

高效运维社区

DevOps 时代

荣誉出品