

MTSC 2021 中国互联网测试开发大会  
TESTING SUMMIT CONFERENCE CHINA 2021

# 质量无界·测绘未来

QUALITY UNBOUNDED MAPPING FUTURE

深圳站

2021.11.19-20 | 深圳登喜路国际大酒店

主办方: TesterHome



**MTSC 2021** 中国互联网测试开发大会 **深圳站**  
TESTING SUMMIT CONFERENCE CHINA 2021

# 低代码测试引擎与造数平台的 实践与探索

虎牙——王玺

主办方: TesterHome

# 目 录

从测试造数开始

低代码革命之路

实践与探索

总结与展望

# 从测试造数开始



## 虎牙造数某场景：需要一个佩戴粉丝徽章的账号



3 种协议 (HTTP TARS RTMP)

5 个关联的业务团队

7 项步骤

9 次以上的接口请求

灌水 社区里满大街的测试平台都是接口类型的，为啥没有数据工厂的平台？

底层贫困人员 · 2 月前 · 最后由 雨 回复于 2 月前 · 1668 次阅读

背景

目录

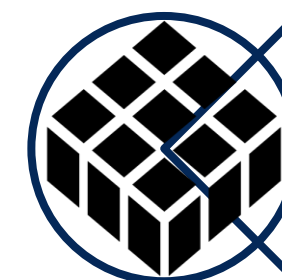
在社区上看到了挺多优秀的测试平台，但大多都是以接口为核心的平台，比较困惑的是，为啥类似的造数据平台都没有？造数据也是业务测试的痛点啊，有一些流程挺长的，涉及上下游的，为了验证一个功能，需要花时间造数据去验证；有时候开发在开发功能的时候，也是需要测试进行造数据。

3、这方面更多还是体力活（一个一个脚本写下来），能做创新的点不多，受众相比接口测试这些也不广，所以大家可能分享欲望也不是太强。

目前造数工具这块其实还是基于公司定制化的东西。并没有多少做成通用的，都是小众化的，无法独立。

我有给团队做一个简单的造数和工具平台，主要是做其他自动化测试的副产物，在代码层面也可以根据注释自动生成前端页面。建议不要想太复杂，一切从解决实际用途出发

大佬，大部分造数工具都是基于流程性的接口用例造数吗？有没有别的方法



没有开源的平台



强业务，难以通用



依赖于接口自动化



场景复用局限性



01

## 数据

1. 测试数据构造成本高
2. 测试数据**丰富度不足**

02

## 工具

1. 面对复杂业务流程**缺乏流程性、系统性**
2. **多样化的场景类型**传统接口工具不足以支撑

03

## 脚本

1. 复杂场景的代码编写，对测试人员要求高
2. 造数脚本**可复用性差**
3. 脚本的后期**可维护性差**

04

## 协作

1. 测试数据**重经验性**
2. 测试团队的重复开发

思路	方法	优点	缺点	常见工具
基于GUI构造	手动、UI自动化	1. 不光是在造数据，本质上还是一次端到端的测试 2. 熟悉业务即可，简单易操作	1. 造数效率低 <b>2. 成本高，稳定性差</b> 3. 依赖性强，前端	Selenium
线上-线下	SQL、流量两类	1. 数据真实性高 2. 数据量大	1. 需要一些工具实现，难度更大 2. 需要脱敏 <b>3. 不一定有需要的数据</b>	goreplay
API调用	脚本，自动化	1. 数据可靠 2. 可以由自动化用例改造过来 3. 不依赖前端	1. 学习成本高 <b>2. 复用性差</b> 3. 效率低	Postman、jmeter
SQL生成	脚本，自动化	1. 效率高	1. 编写难度大，局限性高 <b>2. 跨团队合作困难</b> 3. 数据丰富性差 4. 表变动后需要同步修改	Datafaker
综合平台	集成以上方法	1. 适配多种场景	1. 需要很大的工程开发量 2. 不一定通用	.....



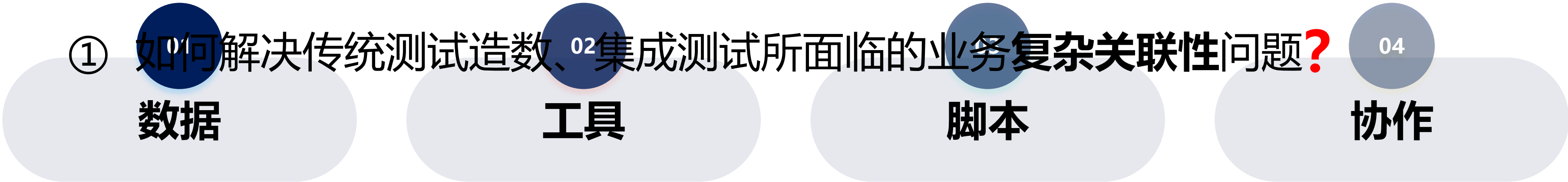
## Code less

- 操作以UI交互为主
- 在接口、自动化工具上实现
- 可视化编排

## Pure code

- 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
Code less	<div>1. 纯ui操作，易上手</div> <div>2. 人员成本低</div>	<div>1. 复杂场景的新功能拓展不易</div> <div>2. 复杂业务流程支持不佳</div>
Pure code	<div>1. 灵活性极高，适配各种场景</div>	<div>1. 脚本维护成本高</div> <div>2. 对QA人员要求高</div>



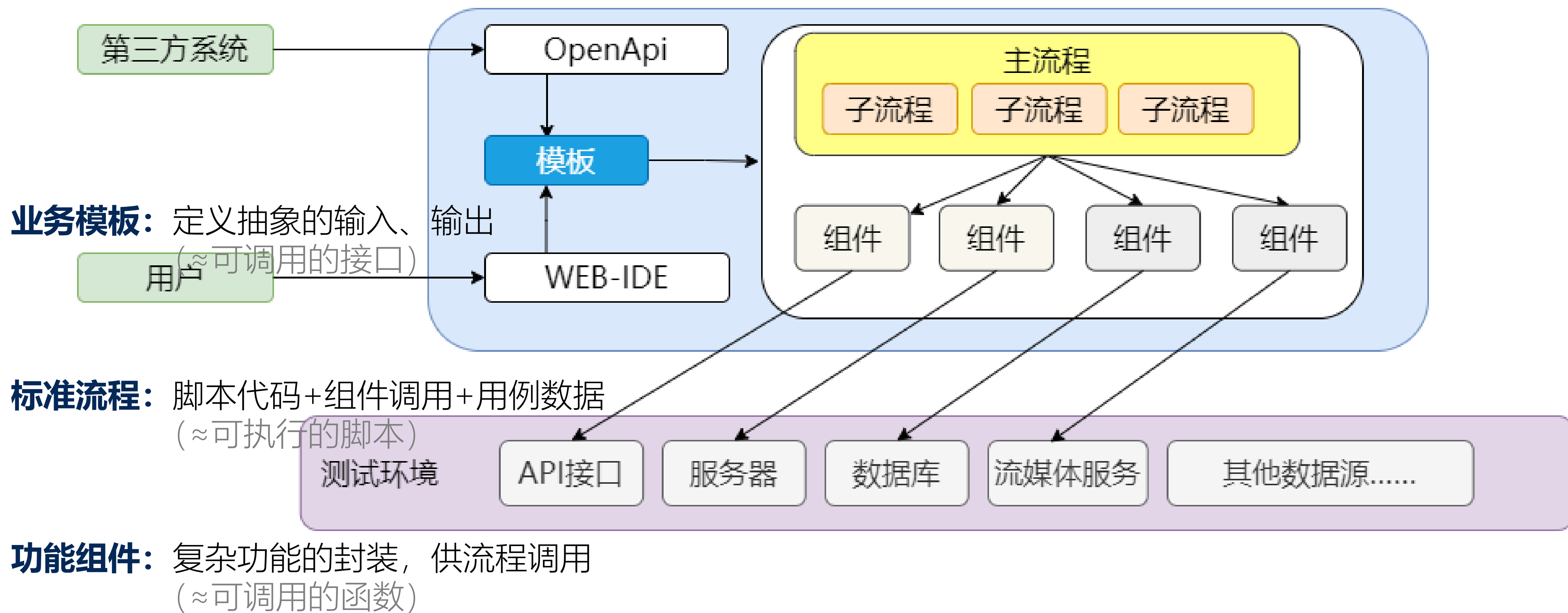
- ② 现代化的测试工具要如何低成本适配更多样的通信协议，中间件？
- 1. 测试数据丰富度不足
  - 2. 多样化的场景类型
  - 1. 测试数据丰富度不足
  - 2. 造数脚本可复用性差
- 中间件？
1. 测试数据丰富度不足
2. 多测试的复用难度高
1. 测试数据丰富度不足
2. 造数脚本可复用性差

③ 测试领域code less, low code 与pure code之争，灵活与规范的抉择？

Question: 测试领域的测试工具要如何低成本适配灵活与规范的抉择？



# 低代码革命之路





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:

▼ dev2

★ 类型:

▼ dev3

★ 类型:

▼ dev4

★ 类型:

基本设置

\* 接口名称:

\* 选择分类:

\* 接口路径:

Tag:

状态:

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

返回 编辑

地址

测试 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YAPI接口平台上录入

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

★ 流程

1

2

3

4

5

6

```
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] =====f1ow 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": "T", "url": "https://mock/463/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": 156, "Access-Control-Allow-Origin": ["123456"], "Access-Control-Allow-Credentials": ["true"], "Date": ["Tue, 28 Sep 2021 03:50:39 GMT"], "Server": ["APISIX/2.5.6"]}, "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
```

J	K
~service->apiUserRegisterGET	
demo_pwd	demo_type
\${pwd}	\${userType}
\${pwd}	\${userType}
\${pwd}	\${userType}

声明式调用组件  
自动生成接口参数  
即时编译，实时日志



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

模板输入参数

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

模板输出参数

参数	类型
phone	String
password	String
uid	String

Parameters

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

保存快照

执行

根据手机号注册用户(RegisterByPhone)

参数编辑

执行结果

结果

序号	id	phone ②	password ②	uid ②	状态
1	17349697				成功

日志

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: 1193

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

文本对比(格式化)

结束流程

从脚本不停机热更新，动态生成接口与文档、默认值、校验

## 功能管理

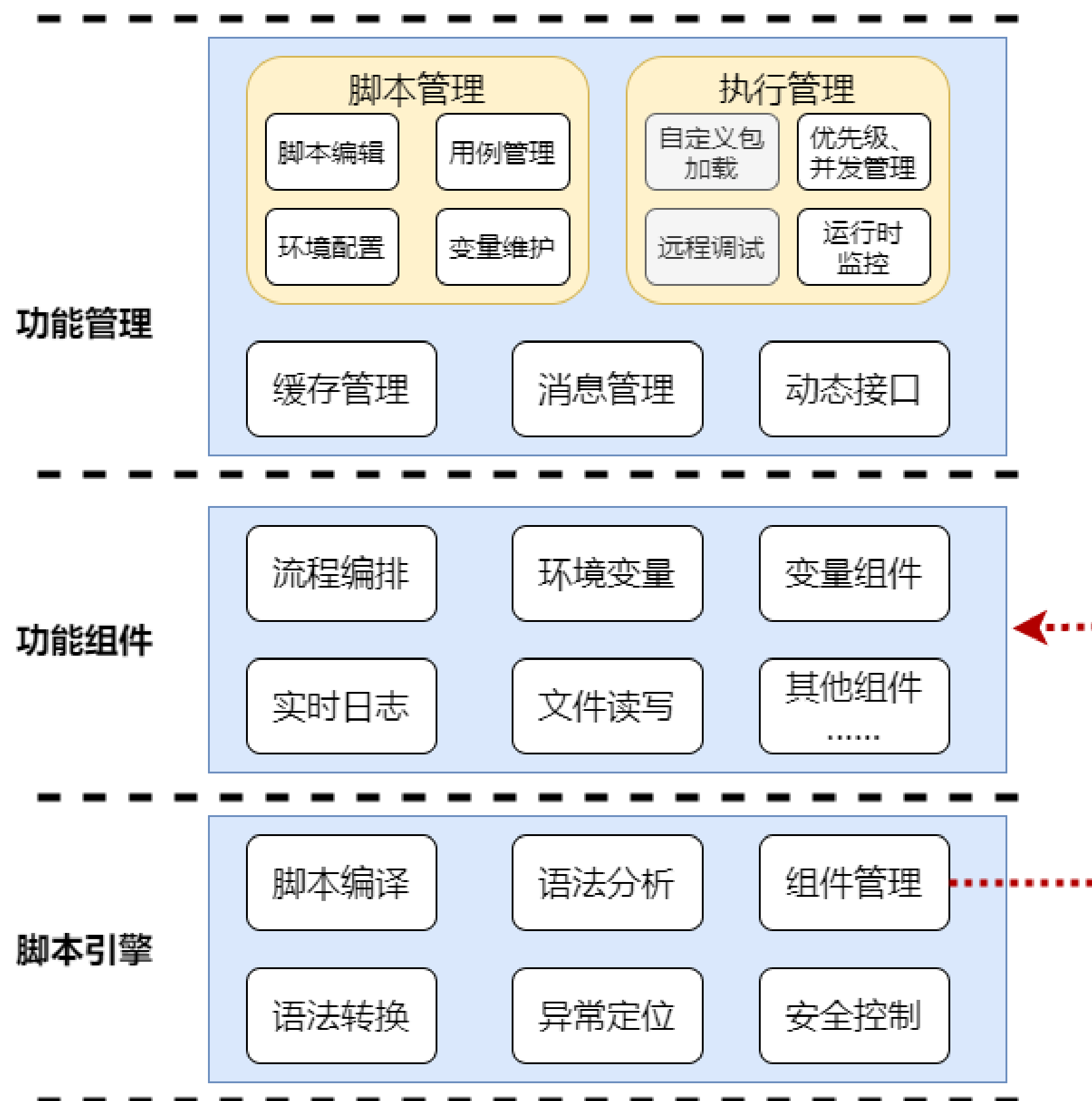
- 脚本管理
- 执行管理

## 功能组件

- 系统核心组件
- 其他功能组件：RPC、推流、DB.....

## 执行引擎

- 脚本引擎
- 组件系统





**Question:** 现代化的测试工具要如何 **低成本** | **适配** 更多样的通信协议，中间件？

**Answer:** 组件（插件）系统

**Question:** 现代化的测试工具要如何 **低成本** | **适配** 更多样的通信协议，中间件？

**Answer:** 使用成本低+开发成本低

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**



核心理念：命令式编程——》声明式编程

声明式Http请求

```
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



★ 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

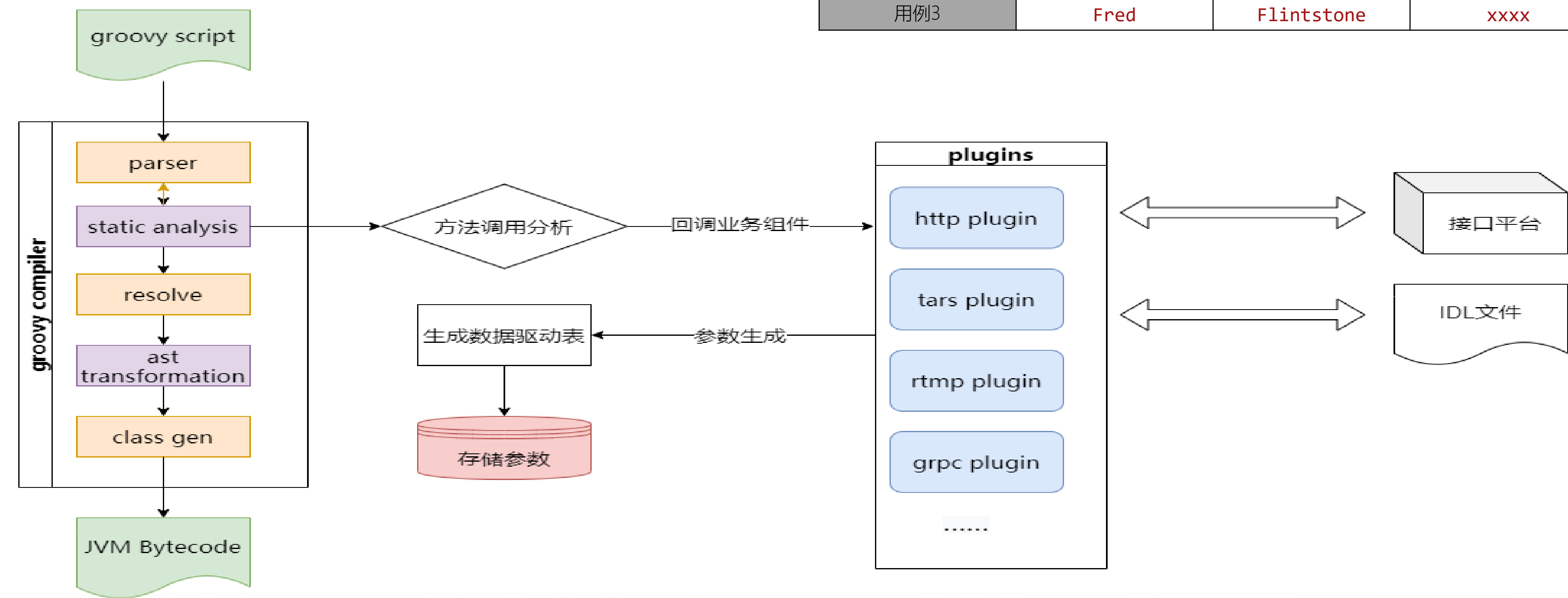
	A	B	C	D	E	F	H	I	J	K	
1				流程用例	用例描述	username	userType	HTTP: user-service->apiUserRegisterGET			
2						demo_username	demo_pwd	demo_type			
3	运行	保存	×	PPTDemo_common_user	注册一个普通用户	user-090931	0	\${username}	\${pwd}	\${userType}	
4	运行	保存	×	PPTDemo_vip_user	注册一个vip用户	user-090932	1	\${username}	\${pwd}	\${userType}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

自动生成参数

如何从声明式的调用分析出请求参数?

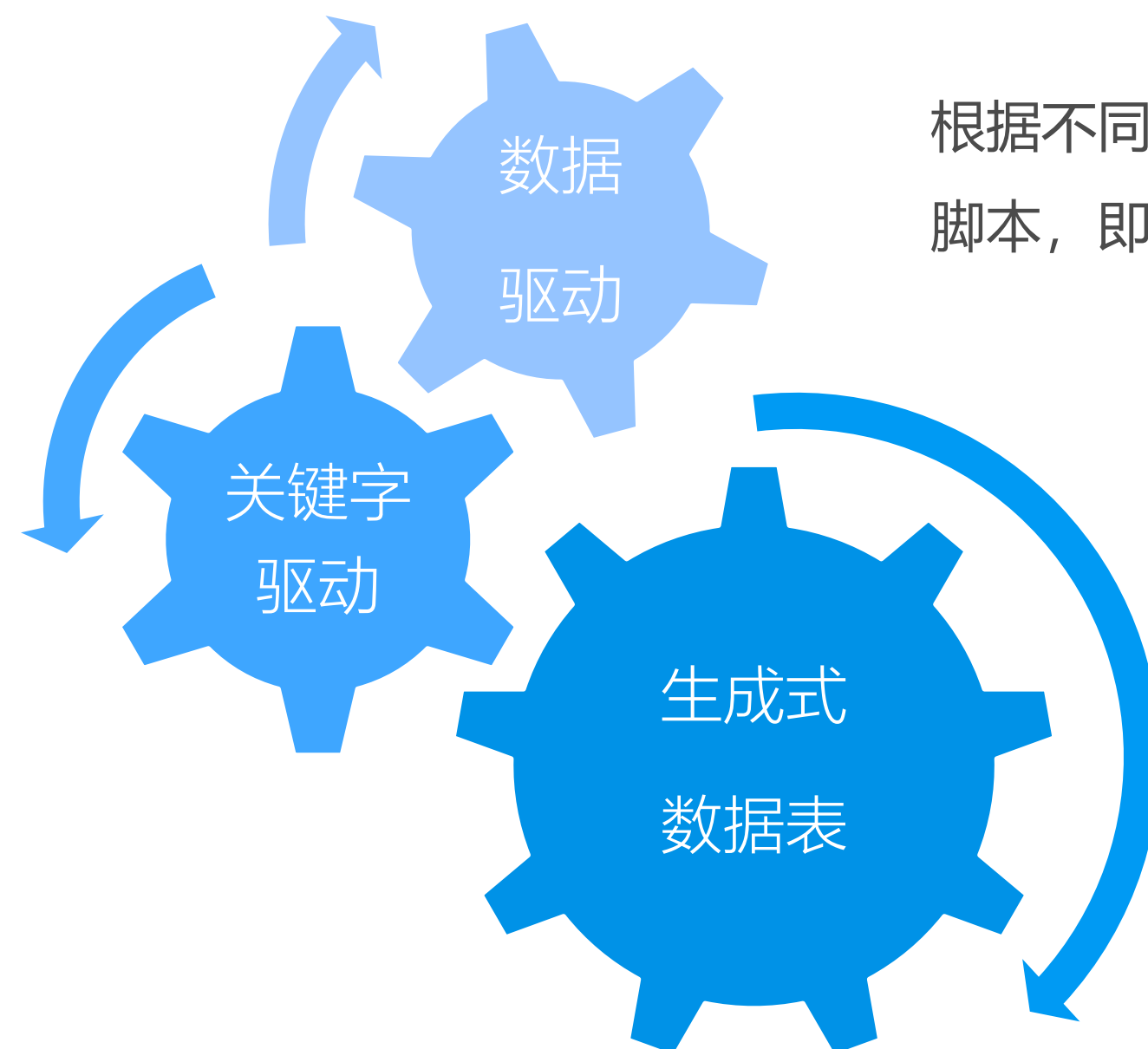
	demo_username	demo_upwd	demo_type
用例1	Fred	Flintstone	xxxx
用例2	Foo	Bar	xxxx
用例3	Fred	Flintstone	xxxx

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





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**

由系统帮你完成可预知的数据定义。使得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**

## Json提取

The screenshot shows the 'JSON Extractor' configuration window. It includes fields for 'Name' (set to 'JSON Extractor'), 'Comments', and 'Apply to' (with 'Main sample only' selected). There are also sections for 'Names of created variables', 'JSON Path expressions', 'Match No.' (set to 0 for Random), 'Compute concatenation var (suffix \_ALL)' (unchecked), and 'Default Values'.

## 正则匹配

The screenshot shows the 'Regular Expression Extractor' configuration window. It includes fields for 'Name' (set to 'Regular Expression Extractor'), 'Comments', and 'Apply to' (with 'Main sample only' selected). There are also sections for 'Field to check' (with 'Body' selected), 'Name of created variable', 'Regular Expression', 'Template (\$i\$ where i is capturing group number, starts at 1)', 'Match No.' (set to 0 for Random), 'Default Value', and a checkbox for 'Use empty default value'.

## SQL查询

The screenshot shows the 'SQL Query' configuration window. It includes a 'Query Type' dropdown (set to 'Select Statement') and a 'Query' text area. Below the query area, there are sections for 'Parameter values', 'Parameter types', 'Variable names', 'Result variable name', 'Query timeout', 'Limit ResultSet', and 'Handle ResultSet' (set to 'Store as String').



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

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

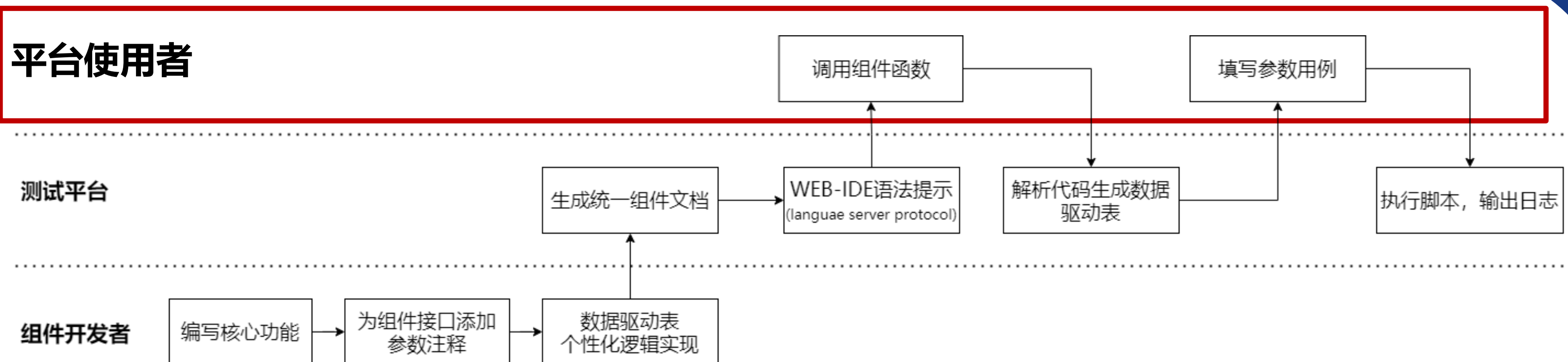
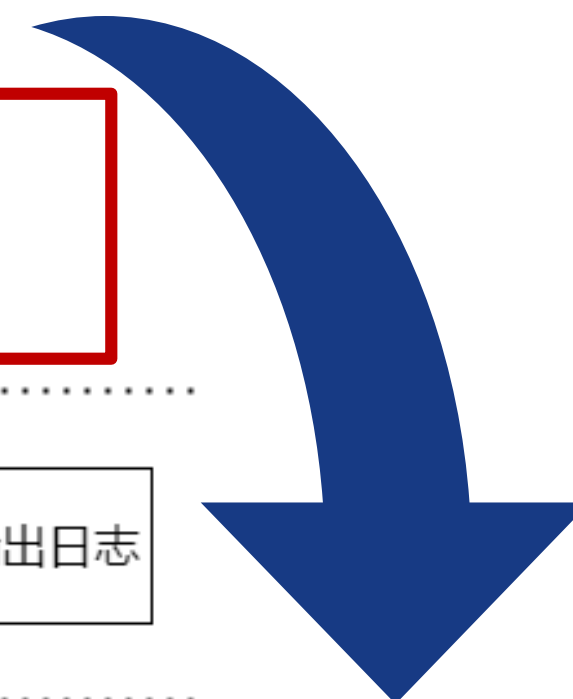
}

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



**核心理念：** 函数调用+数据驱动表替代部分界面配置，复杂度下移，降低使用成本

复杂度下移



后端开发成本（功能实现、文档编写）



开发过程中注释注解自动生成使用文档

前端开发成本（界面开发、文档编写）



函数的调用，IDE自动生成语法提示

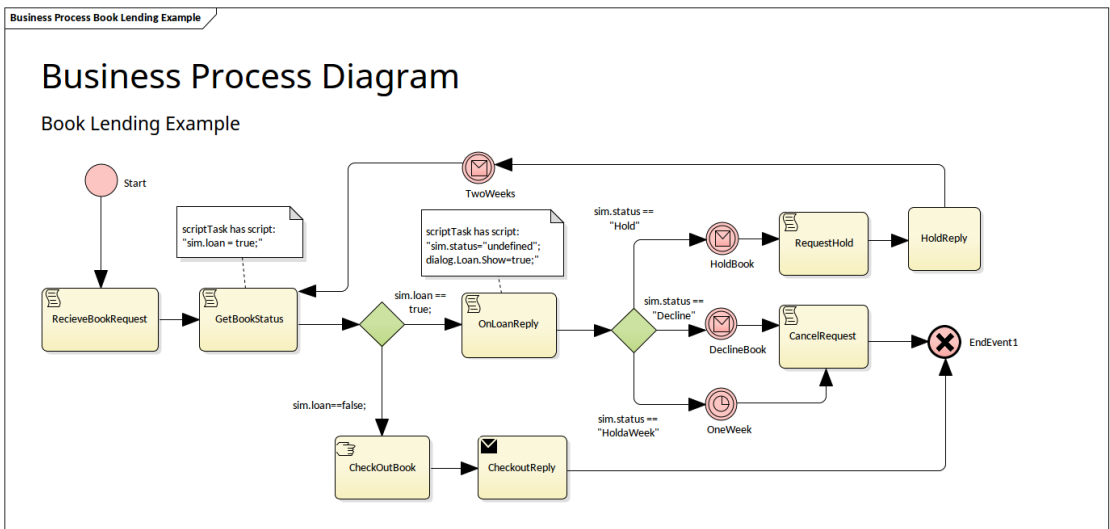
用户学习成本（功能学习、使用学习）



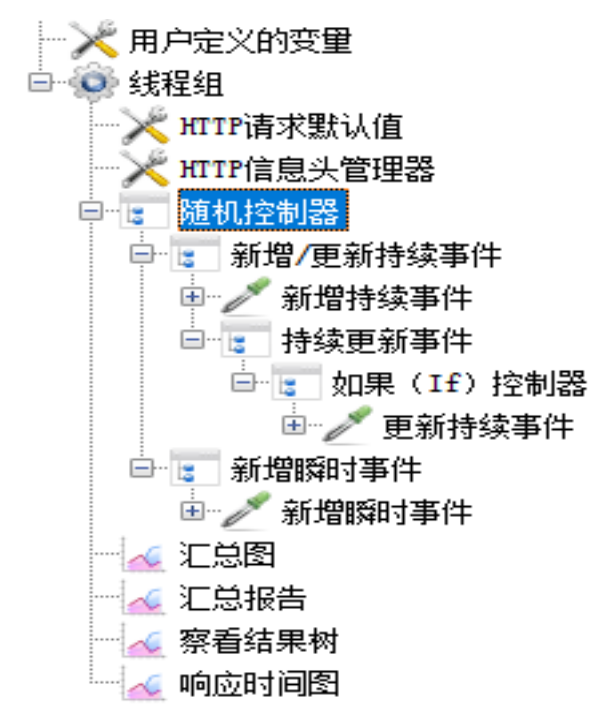
统一的函数调用学习



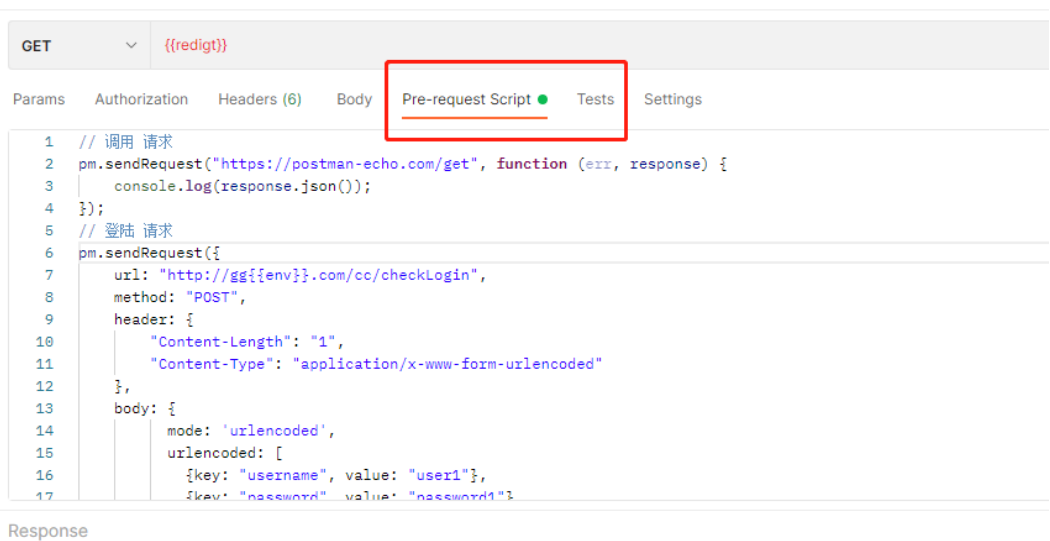
## 工作流引擎界面拖拽式编排



## Jmeter控制器编排



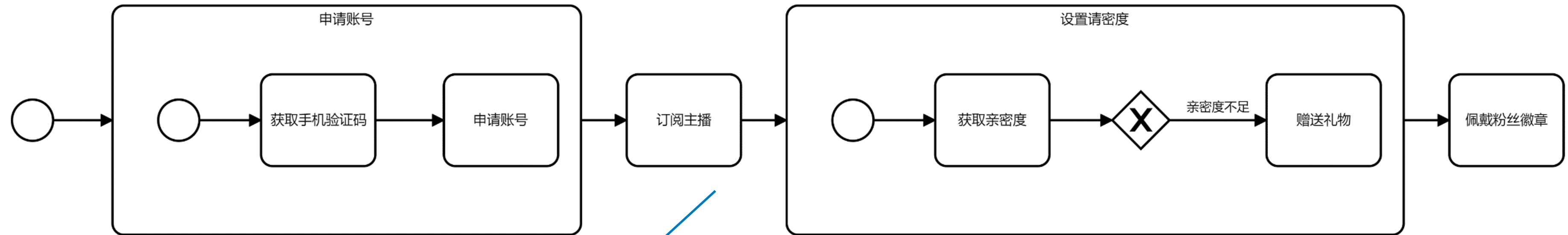
## Postman接口前后置编排



## HttpRunner钩子函数式编排

```
1 - config:
2   name: basic test with httpbin
3   request:
4     base_url: http://127.0.0.1:3458/
5   setup_hooks:
6     - ${hook_print(setup_testset)}
7   teardown_hooks:
8     - ${hook_print(teardown_testset)}
9
10 - test:
11   name: get headers
12   times: 2
13   request:
```

## workflow引擎编排样例



## 低代码编排样例

```
class 获取佩戴粉丝徽章的用户{
  @override
  void test(){
    pf.flow.run("申请账号")
    pf.http.request("订阅主播")
    pf.flow.run("设置亲密度")
    pf.tars.request("佩戴粉丝徽章")
  }
}
```

```
class 设置亲密度{
  @override
  void test(){
    pf.http.request("获取亲密度")
    if("${获取亲密度.亲密度}" < target){
      pf.http.request("赠送礼物")
    }
  }
}
```

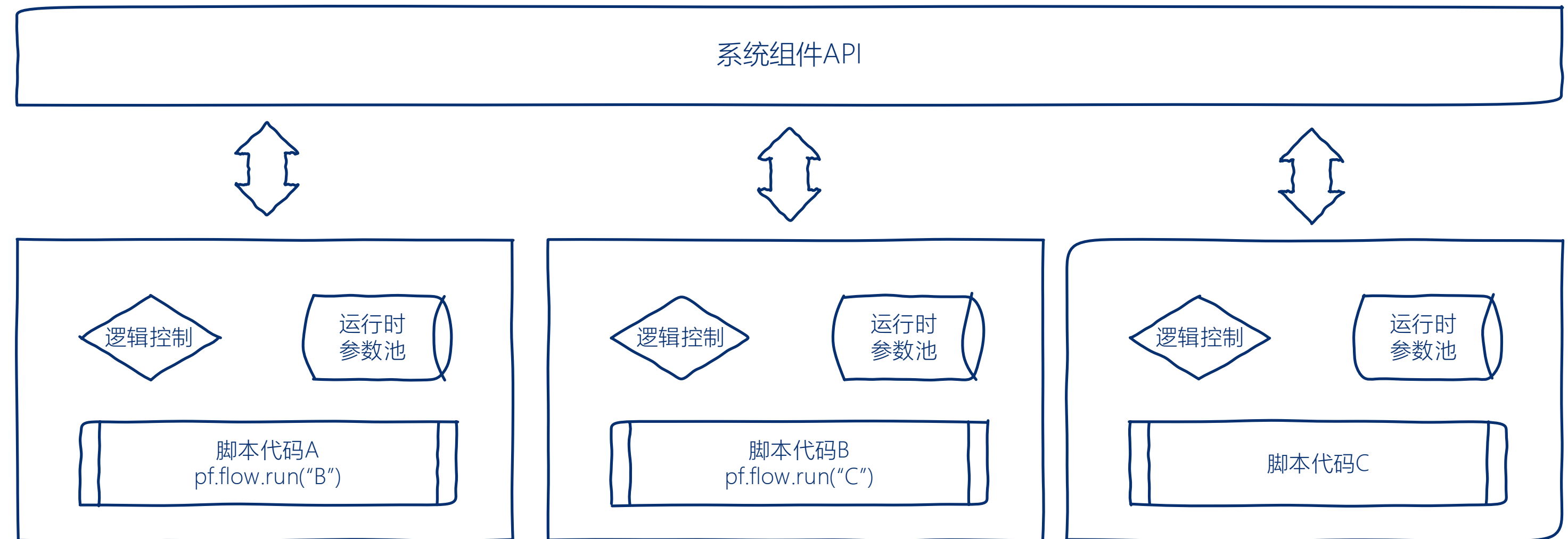


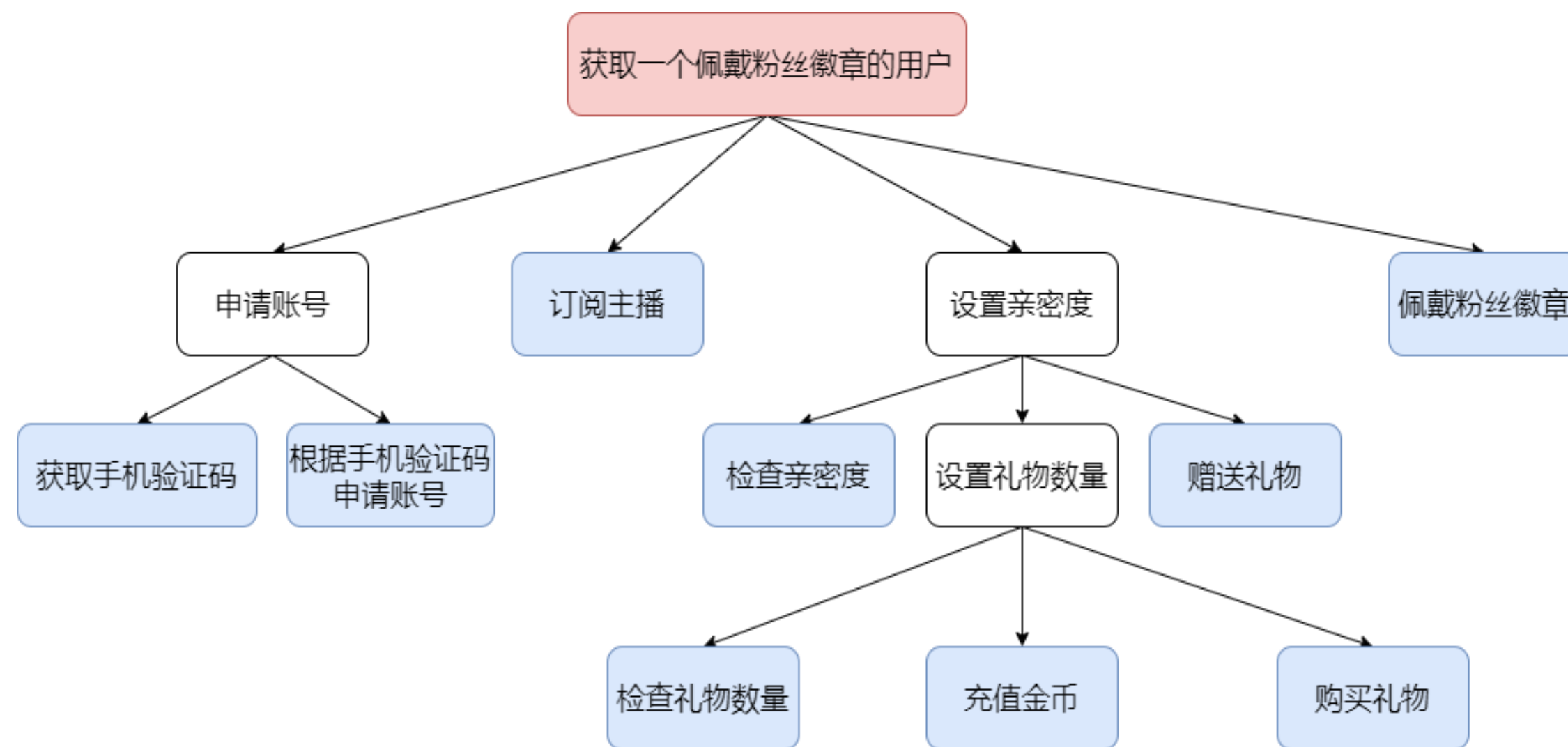
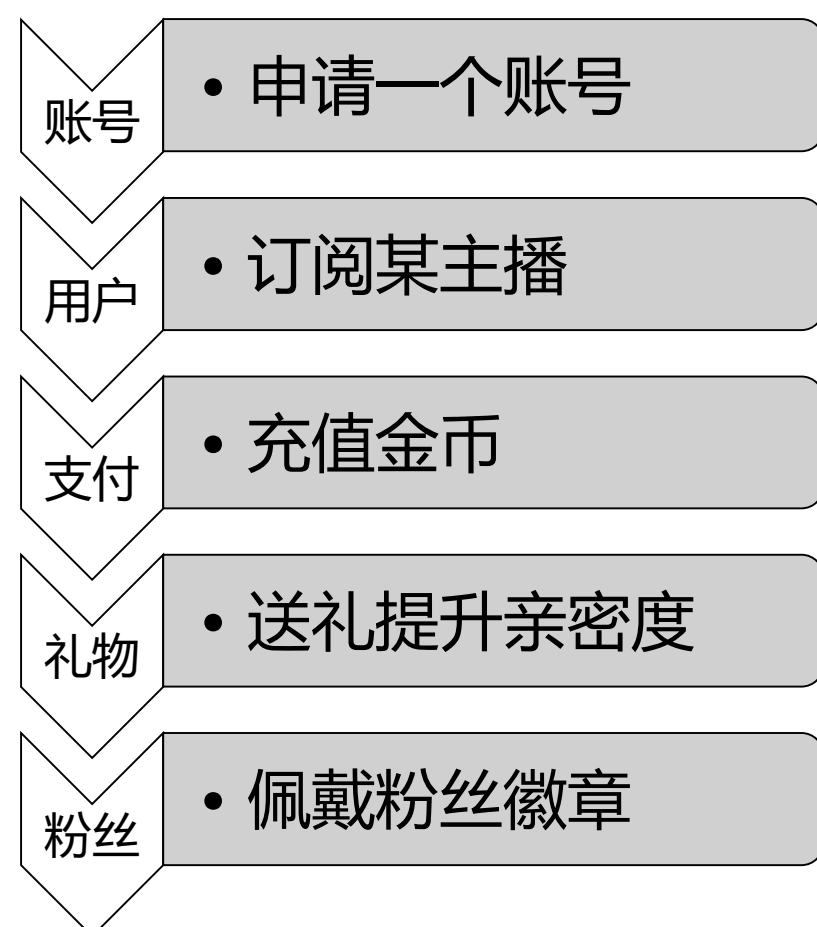
**Question:** 如何对不同的人写的、风格迥异的代码进行编排?

```
class A extends Flow {  
    void test() {  
        pf.flow.run("B")  
    }  
}
```

```
class B extends Flow {  
    void test() {  
        pf.flow.run("C")  
    }  
}
```

```
class C extends Flow {  
    void test() {  
    }  
}
```





多数场景都可以抽象成执行逻辑为 **DFS** 的 **多叉树** 结构

主流程：树的根节点，目标业务场景

父流程：父节点的流程，发起调用的流程

祖先流程：同等于树的祖先节点

单一流程：树的叶子节点，不包含任何子流程的纯净流程

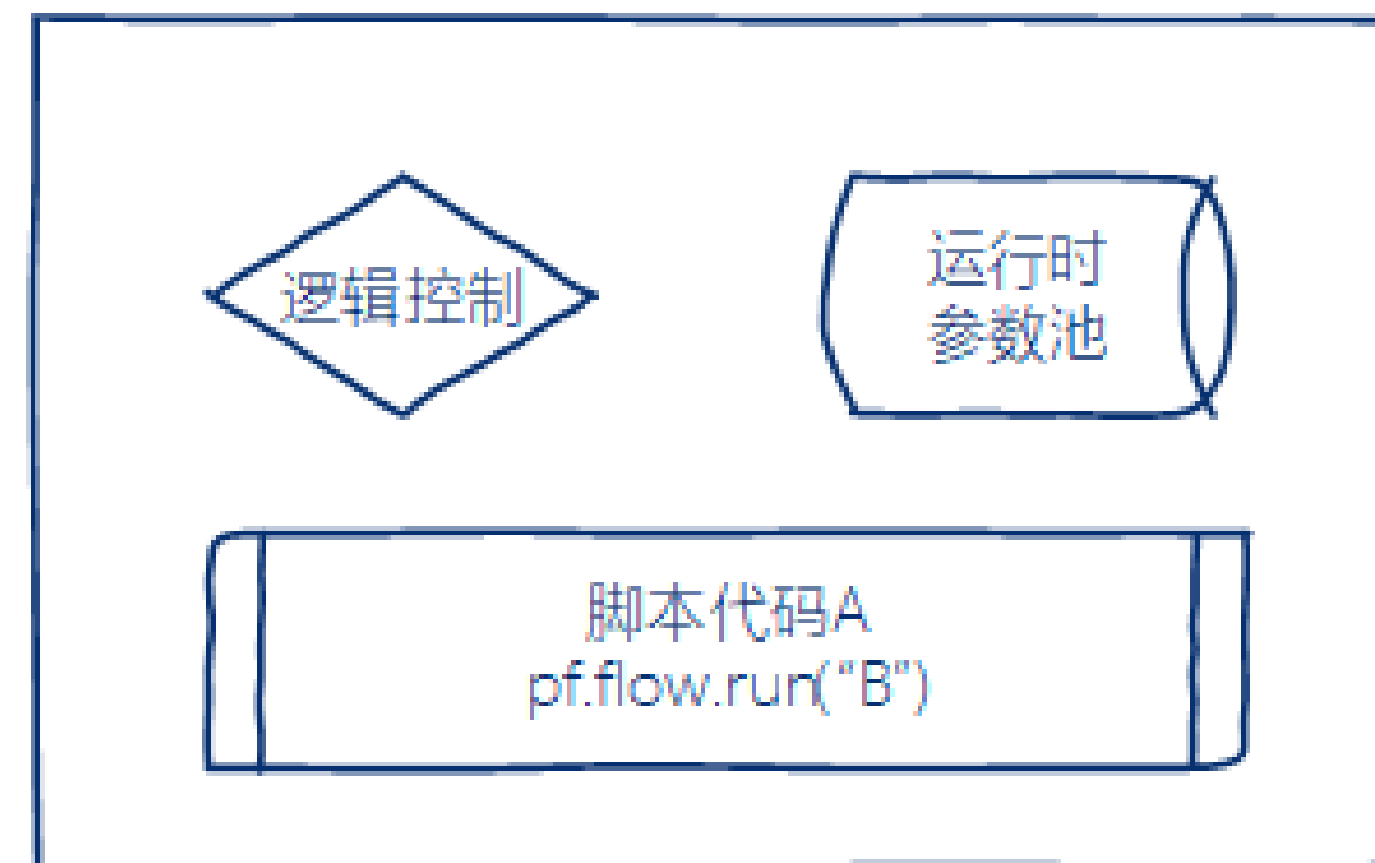
子流程：子节点的流程，被调用的流程

子孙流程：同等于树的子孙节点



**Question：**多人协作时，我们对子流程进行调用，如果部分接口参数不可修改，那么如何复用？如果修改参数，如何做到可维护？

**Answer：**运行时，无侵入的修改子流程参数  
(全局变量、局部变量)



```
pf.flow.run("registerUser")
def resp=pf.variable.get("registerUser", "resp")
if(resp?.code==0){
    pf.flow.runWithVariable("Recharge",["user":"${registerUser?.resp?.uid}"])
}else if(resp?.code=4){
    pf.flow.runWithVariable("registerUser",["user":"User-"+UUID.randomUUID()])
}
```

## 子流程调用：声明式运行一个流程

`pf.flow.run("registerUser")` +

	registerUser	user	type
用例1	注册普通用户	User1	Common
用例2	注册VIP用户	User2	Vip
用例3	注册超级管理员	User3	admin

## 条件判断：if else编排

```
pf.flow.run("registerUser")
if(("${registerUser.resp.code}"==0){
    pf.flow.run("Recharge")
}else if("${registerUser.resp.code}"=4){
    pf.flow.runWithVariable("registerUser",["user":"User"+UUID.randomUUID()])
}
```

## 循环调用：for while编排

```
for( i in 1..10){
    pf.flow.run("registerUserAuto")
}
```



## 流程中断：停止当前的流程，并不执行后续

```
pf.flow.stop("金币数额已经满足要求") //正常的中断
assert false: "xxx异常"                //异常中断
throw new Exception("xxx异常")          //异常中断
```

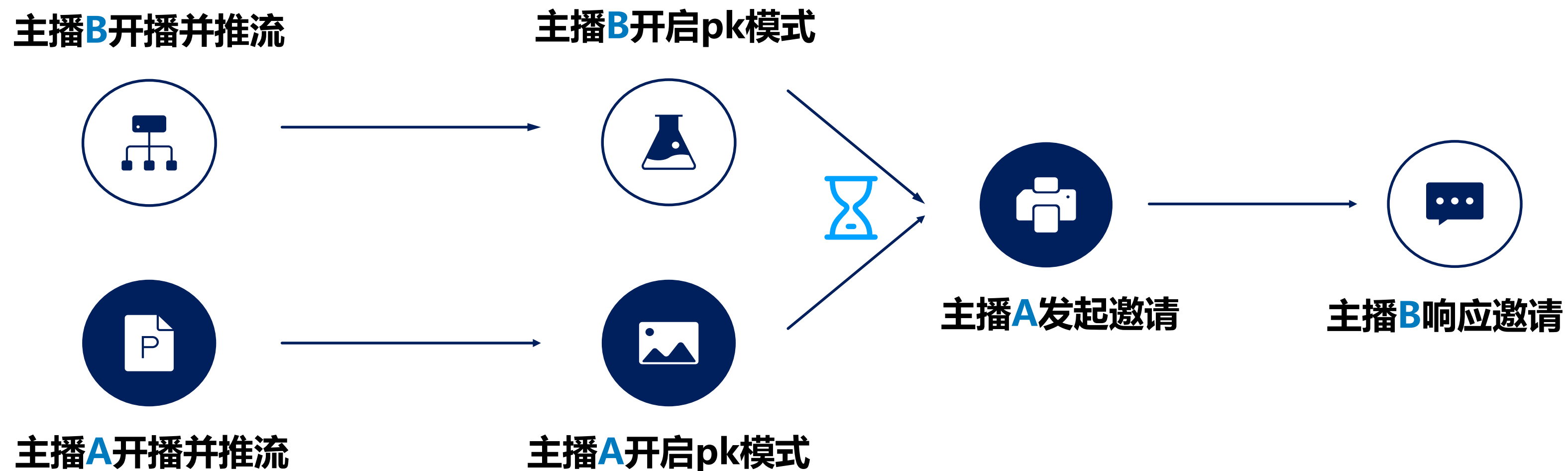
## 失败跳过：子流程的异常不影响主流程后续执行

```
pf.flow.run("BeginLive")
pf.flow.skipOnError {
    pf.flow.run("CheckLiveStatus")
}
```

## 失败重试：子流程失败后再次执行

```
pf.flow.run("BeginLive")
pf.flow.retryOnError(10,10000,{          //尝试10次，每次休眠10s
    pf.flow.run("CheckLiveStatus")
})
```

**Example:** 需要两个处于PK模式下的直播间，需要并行，需要阻塞等待





## 串行

```
pf.flow.runByAlias("BeginLiveAndPushStream", "BeginLive_A") //开播并推流
pf.flow.runByAlias("BeginLiveAndPushStream", "BeginLive_B")
pf.flow.runByAlias("StartPKMode", "Start_A") //开启pK模式
pf.flow.runByAlias("StartPKMode", "Start_B")
pf.flow.run("InviteMultiPK") //邀请
pf.flow.run("ResponseMultiPKInvite") //接收
```

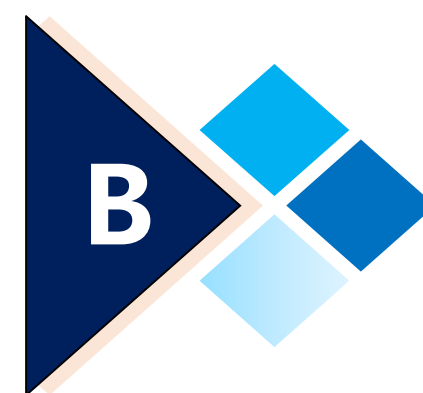
## 并行

```
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")
    }
)
pf.flow.run("InviteMultiPK") //邀请
pf.flow.run("ResponseMultiPKInvite") //接收
```



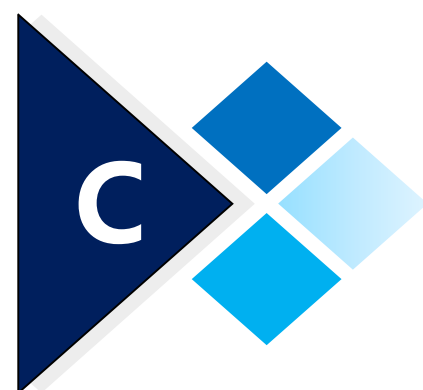
## 更灵活的编排方式

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



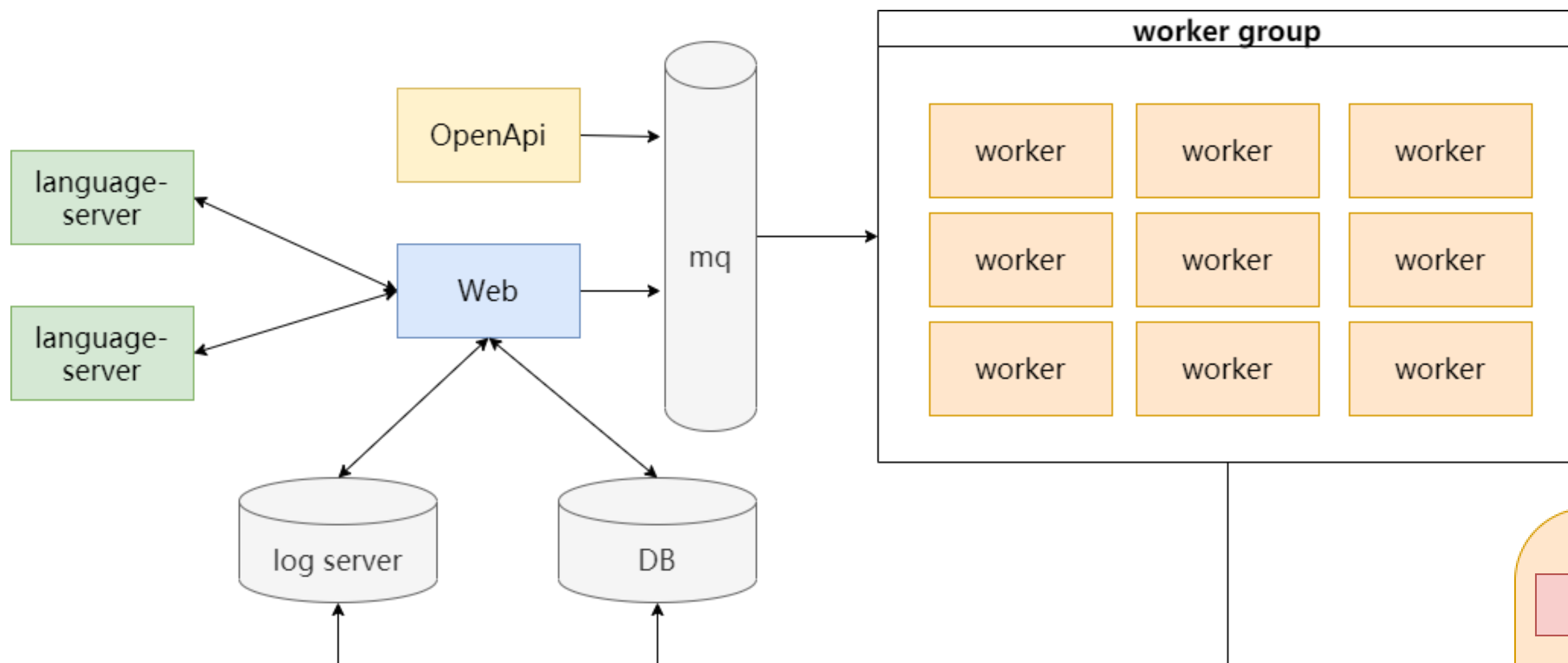
## 统一的流程模型

系统层面：方便后续功能拓展  
脚本层面：可维护行极大提升



# 实践与探索



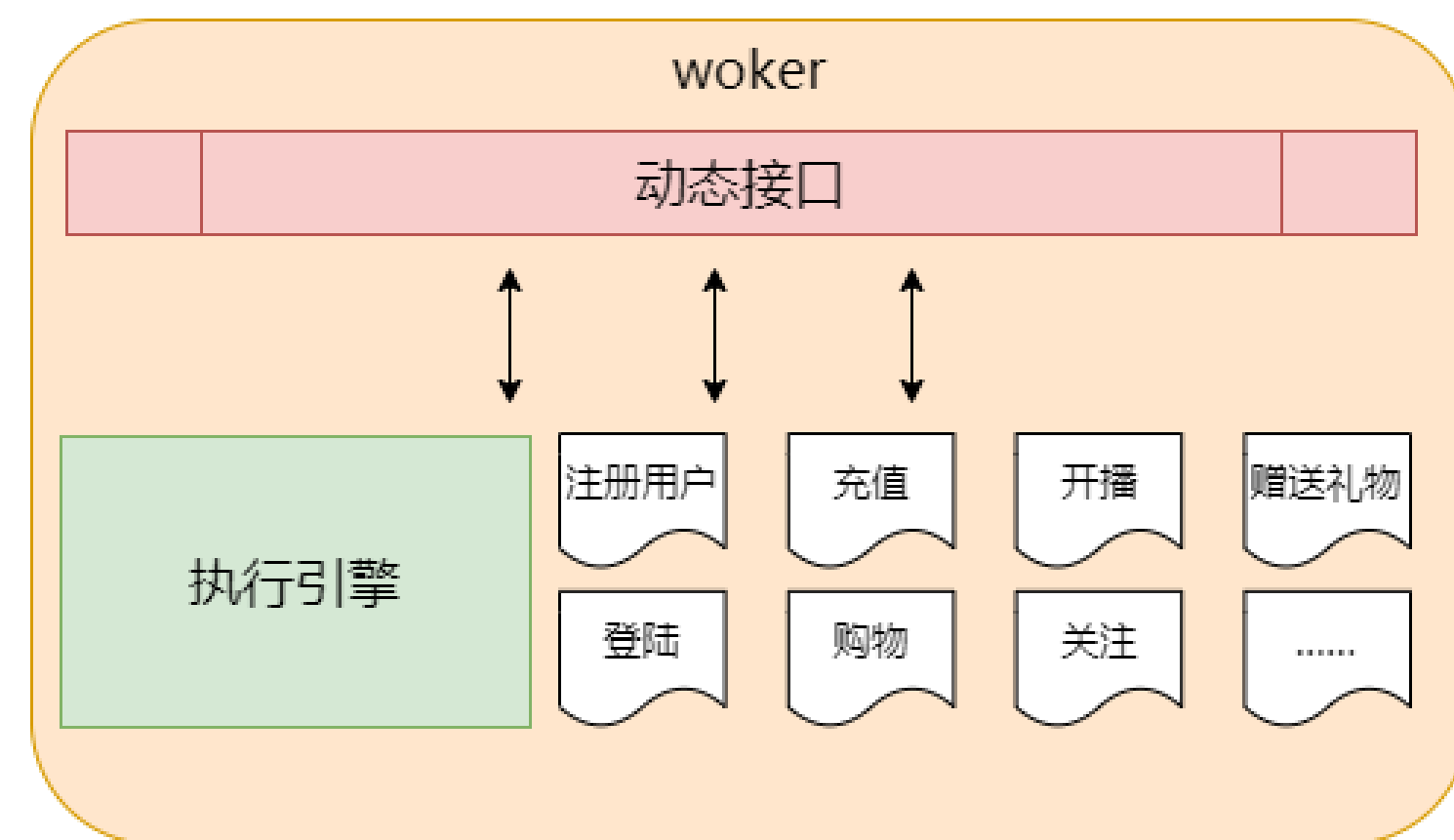


**Web:** 前端，负责脚本创建、编辑等工作

**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:** 对外提供自动化接口





## 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

## 高占用，长耗时的任务

- ① 消耗主机资源，影响系统稳定
- ② 发布导致长时间执行任务中断

## 01 编译期

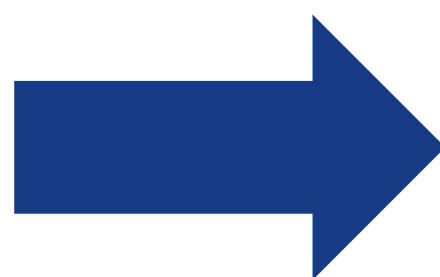
- 对高危操作进行检查
- 代码注入 (SafePoint)

## 02 运行时

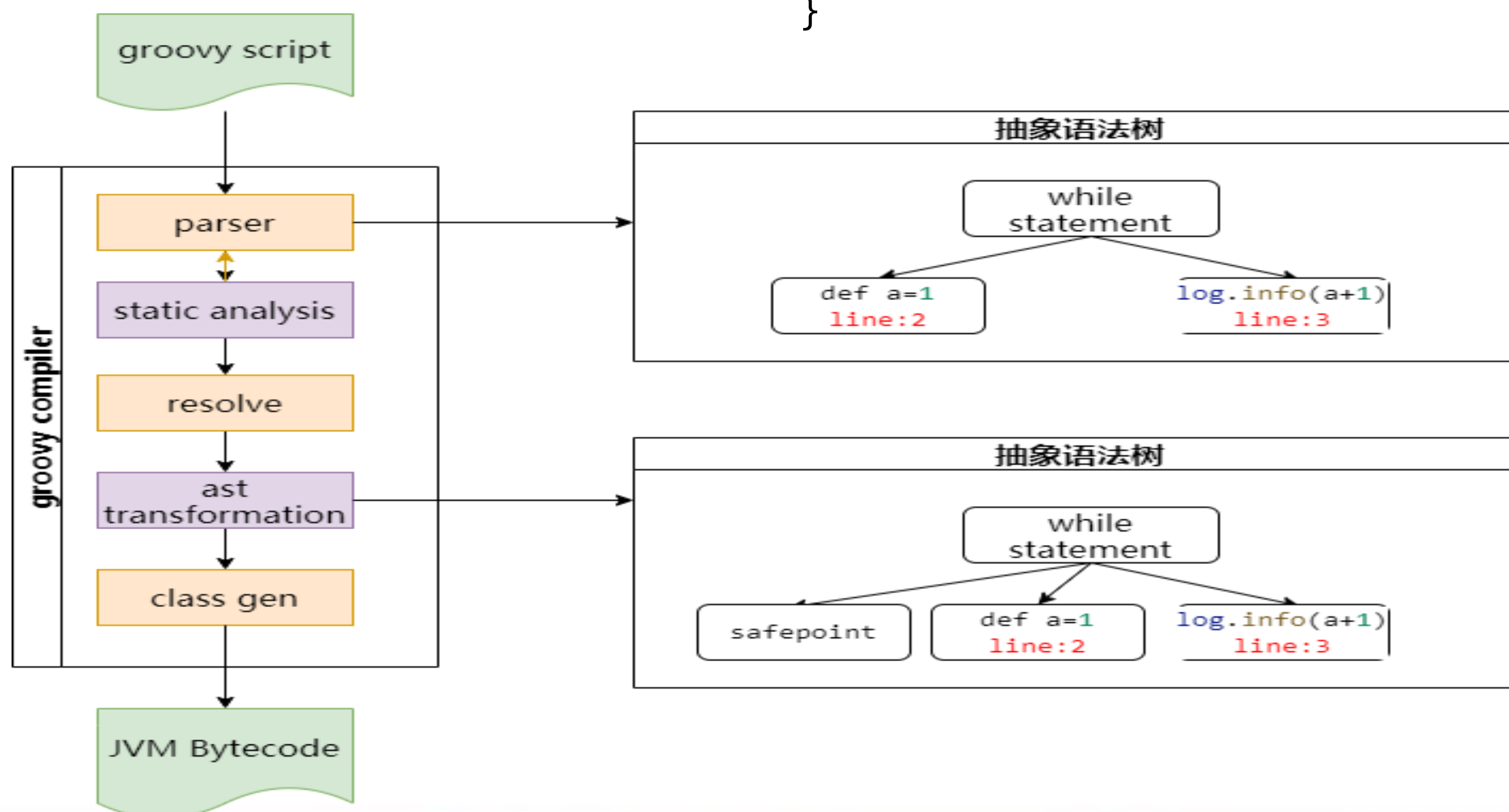
- 运行时资源限制
- 多进程



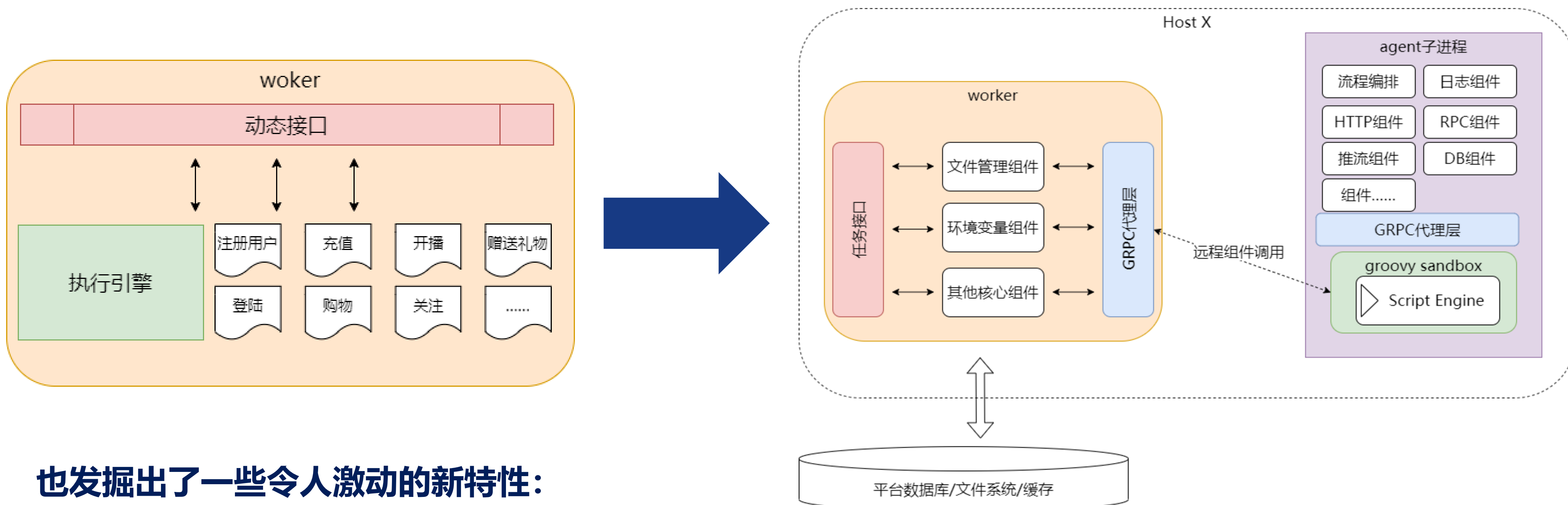
```
while(true){  
  def a=1  
  log.info(a+1)  
}
```



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







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;
```

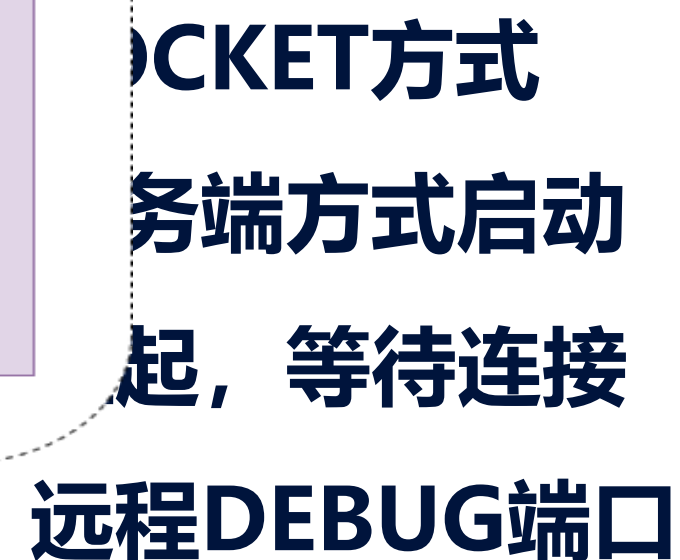
低代码Debug

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

Serverless

多语言混合编排

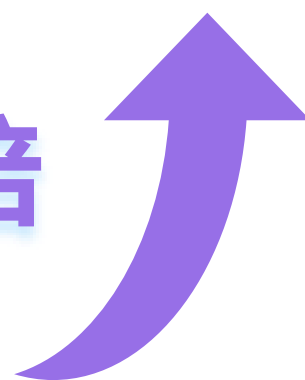
- 远程调试不能影响线上服务——子进程启动，不影响其他脚本
- 本地拥有编译后的平台代码——Agent内包含定制脚本编译器
- 部分代码不想暴露给使用者——部分组件通过GRPC代理层调用，屏蔽实现细节





## 造数效率

50倍



- 效率从过去手动造数分钟级降低到秒级
- 月均调用次数800w+
- 支持huya、nimo等多个产品线
- 测试、开发、产品都在用

## 开发效率

50%

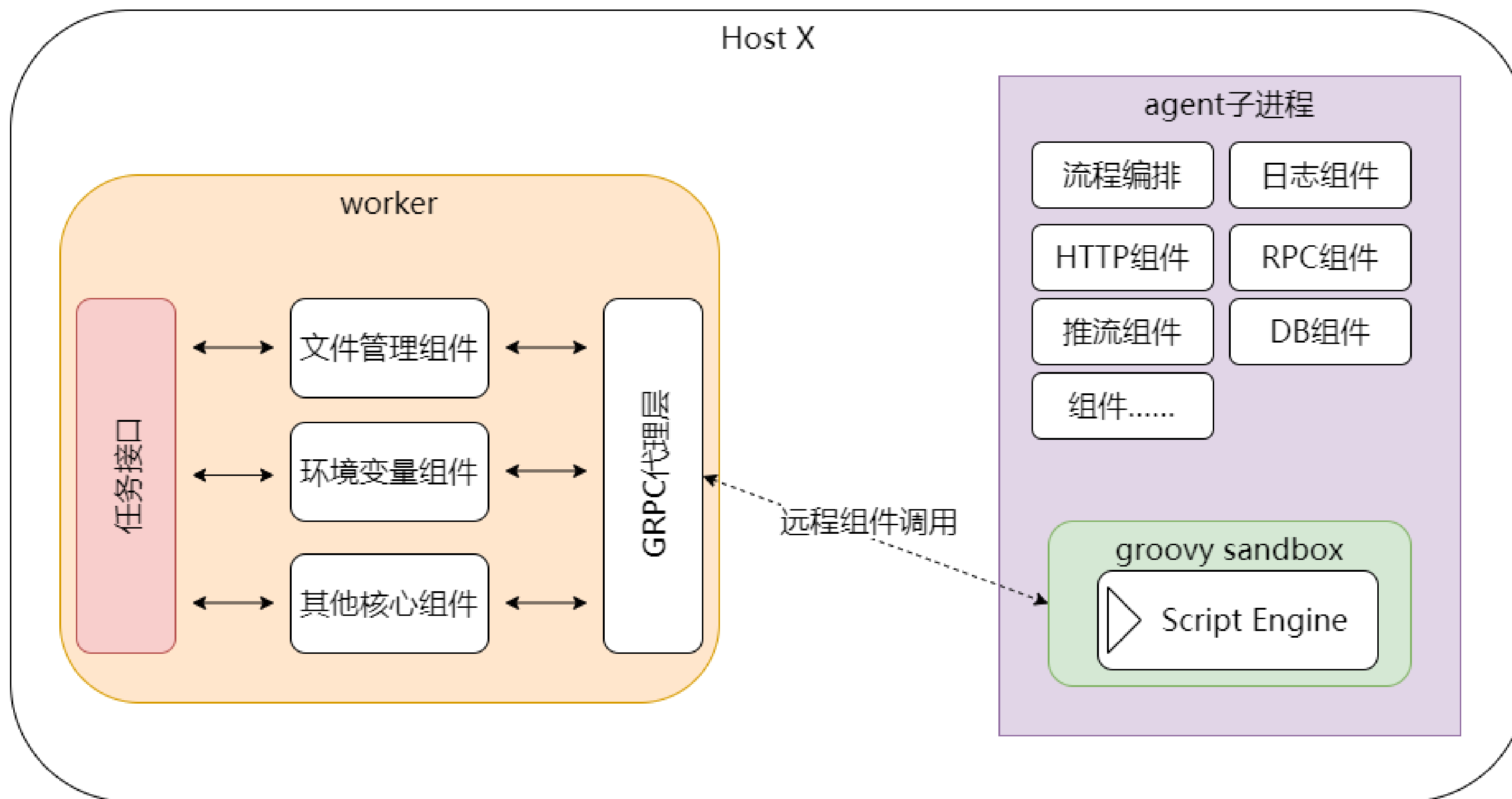


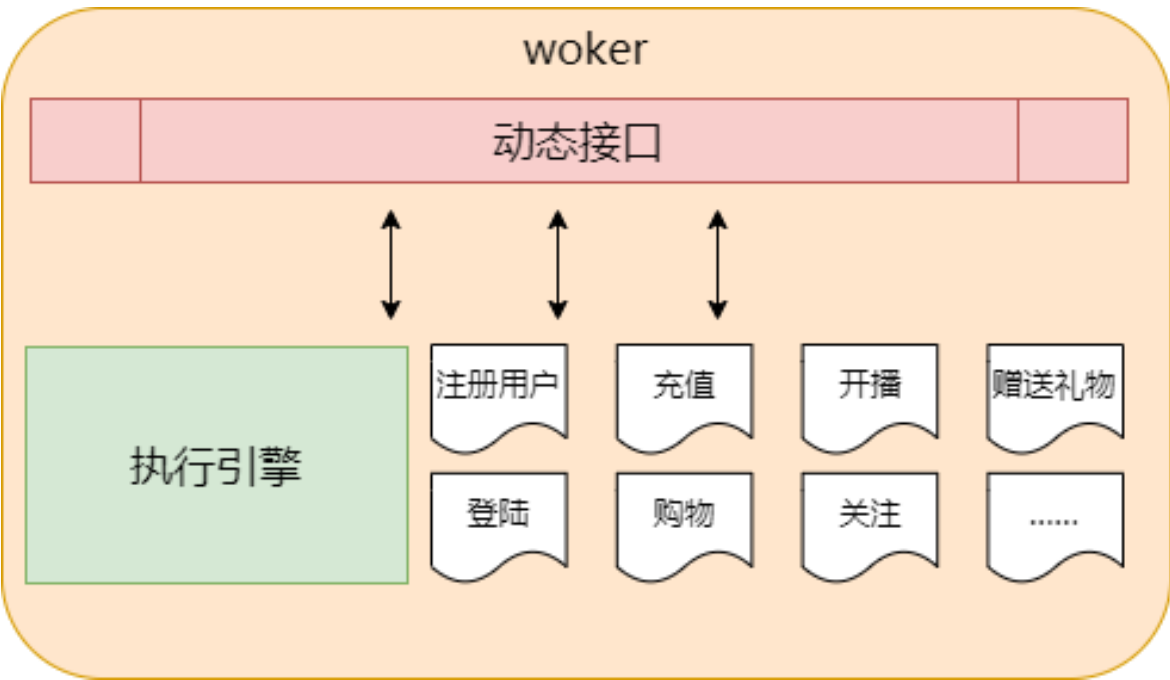
- 单个场景造数自动化数个小时缩短至几十分钟  
(业务沟通+开发+测试+部署+后期运维)
- 1个实习生, 1个月完成50+的造数场景  
(一半以上的时间耗费在业务沟通上)

# 总结与展望

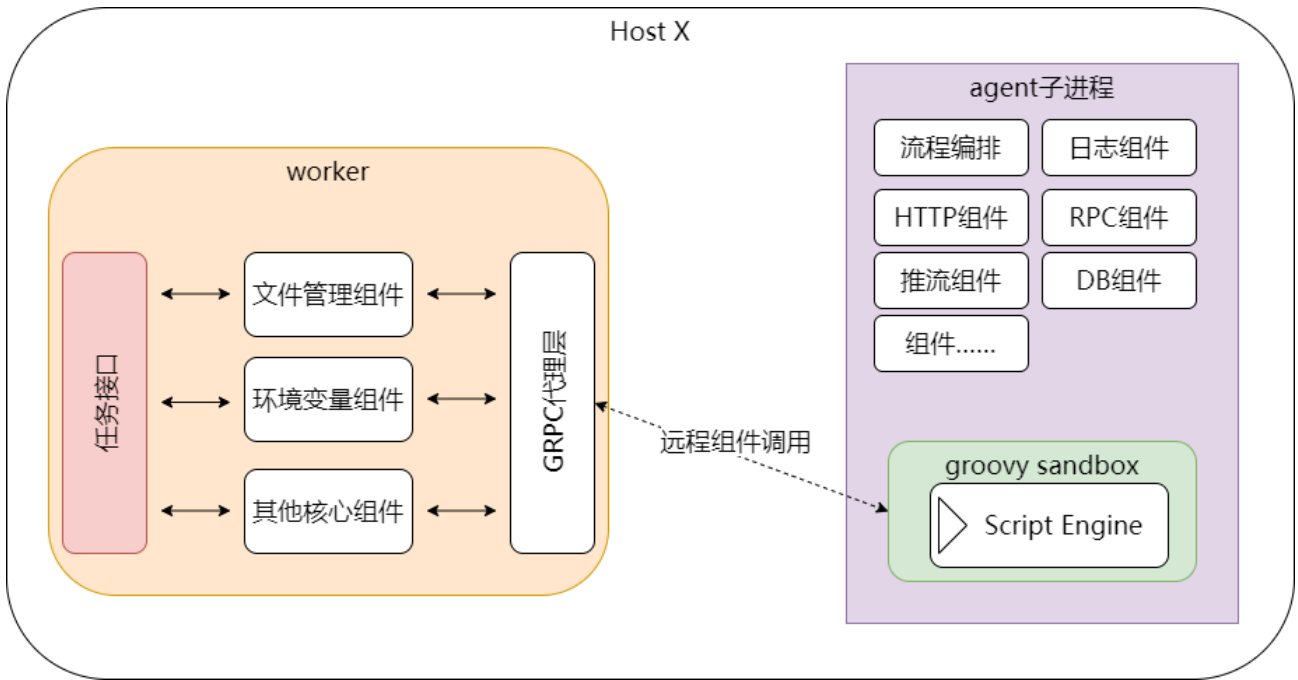




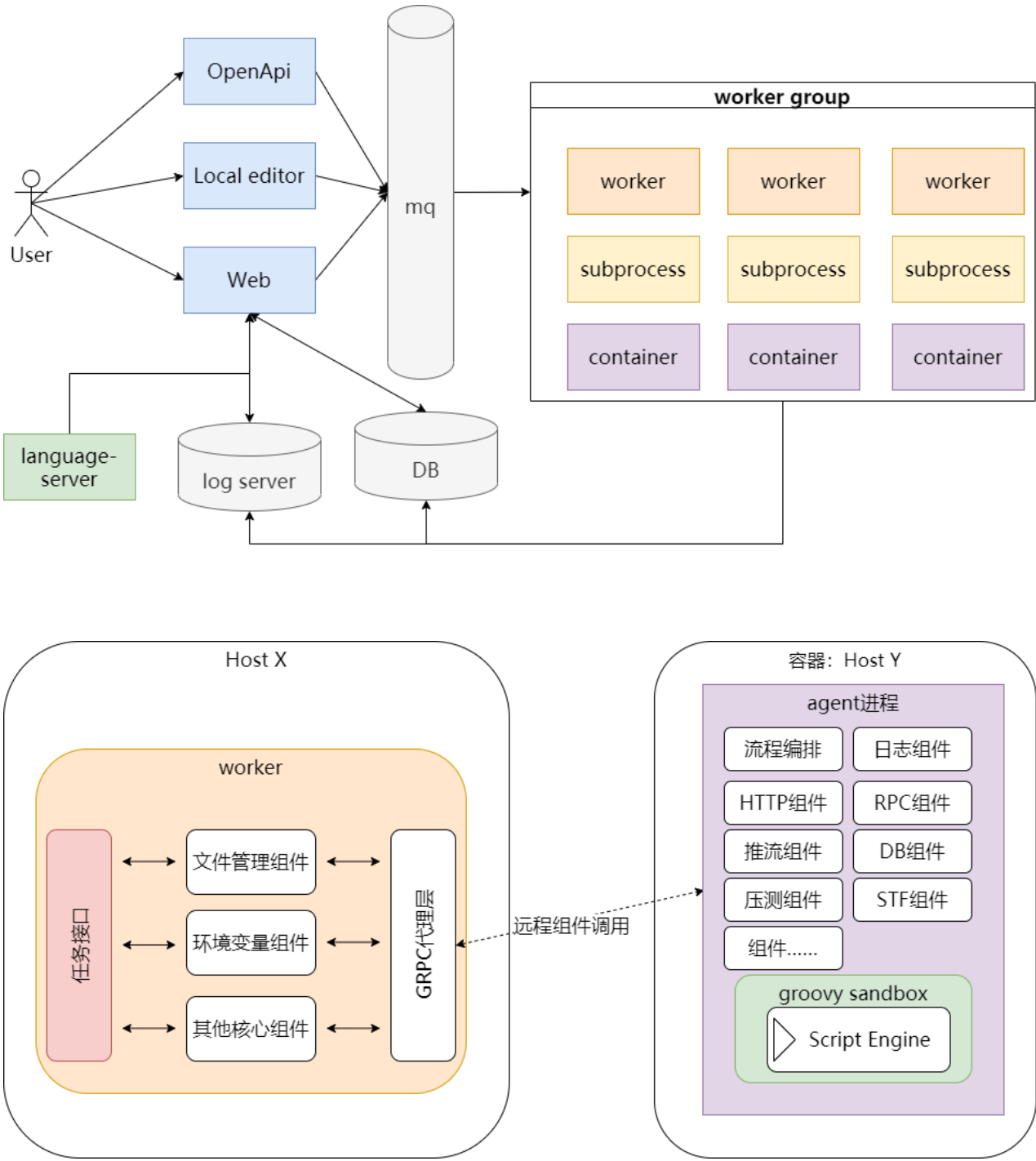




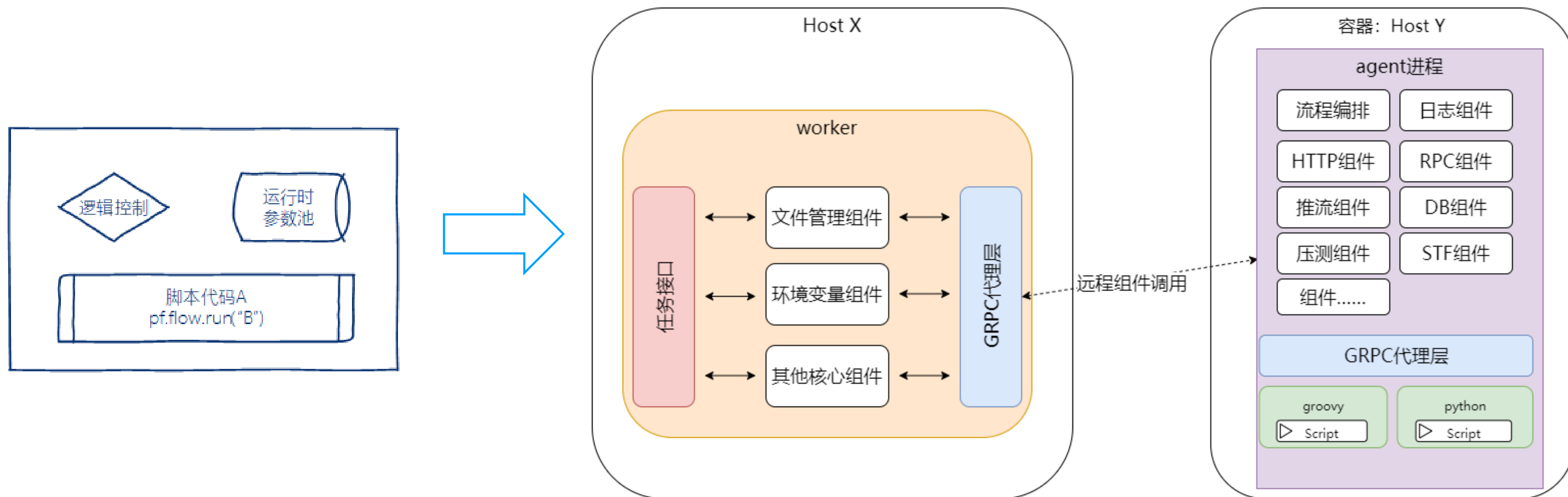
多线程



多线程+多进程



多线程+多进程+serverless

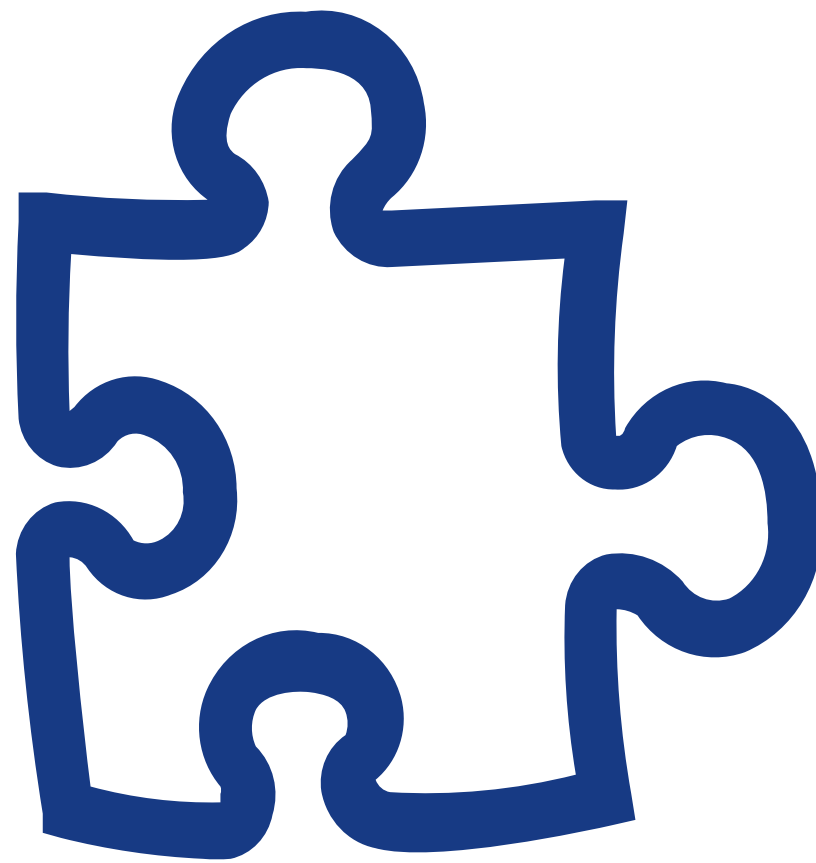






效率

低代码编排  
生成式表驱动



标准

标准流程模型  
统一的组件模型



开放

完全原生的代码  
开放式的组件接入



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

云游戏平台YOWA上线

低代码测试引擎技术交流(712523537)



谢谢  
THANKS

