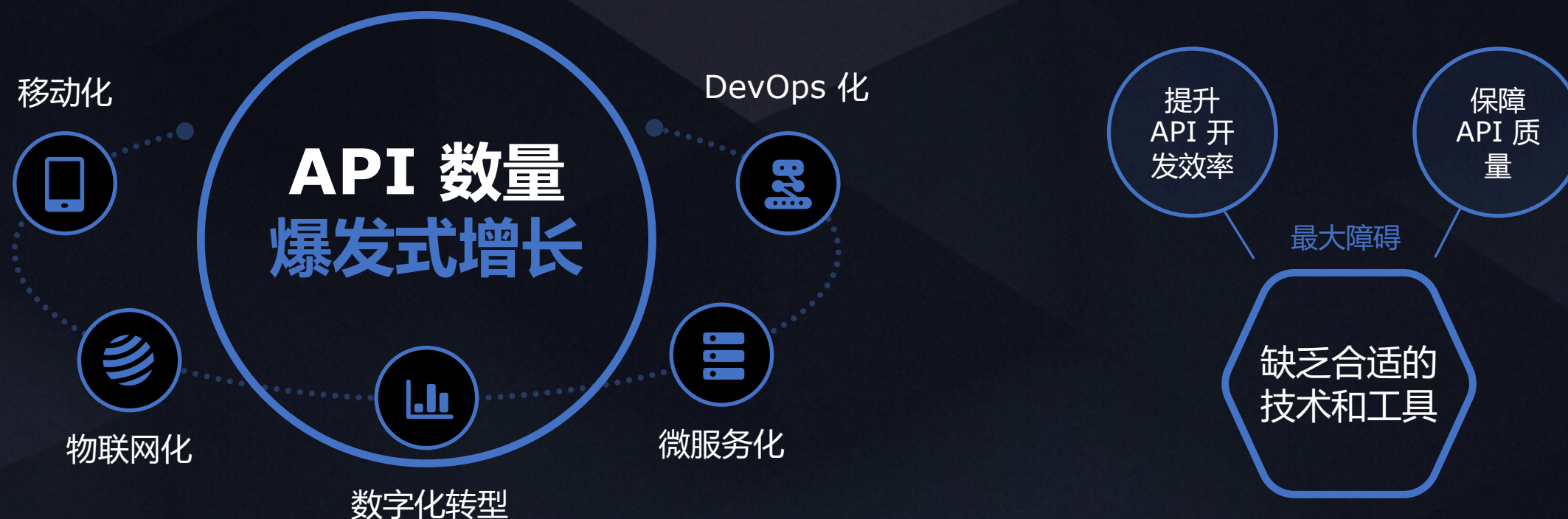


Apifox

API 文档、API 调试、API Mock、API 自动化测试

一体化协作平台

行业情况



常用解决方案

API 设计者

Swagger
API 文档设计



后端开发

Postman
API 开发调试



Mock.js
API 数据Mock



前端开发

JMeter
API 自动化测试
API 压力测试



测试人员

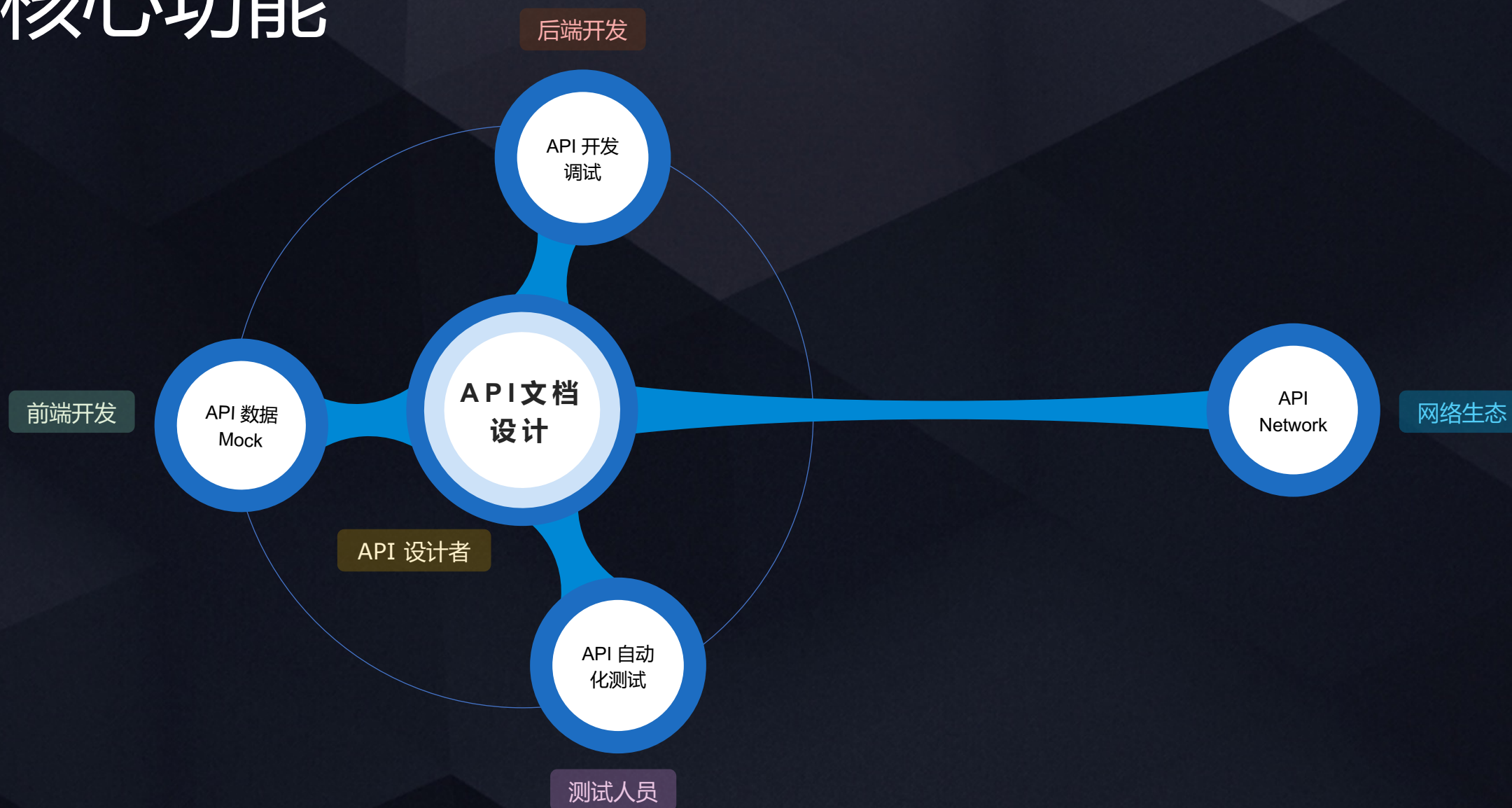
存在的问题

1. 开发人员在 Swagger 定义好文档后，接口调试的时候还需要去 Postman 再定义一遍。
2. 前端开发 Mock 数据的时候又要用 Mock 工具定义一遍，还需要手动设置 Mock 规则。
3. 测试人员需要去 JMeter 再定义一遍。
4. 前端根据 RAP Mock 出来的数据开发完，后端根据 Swagger 定义的接口文档开发完，各都测试通过了，本以为可以马上上线，结果一对接发现各种问题：
 1. 开发过程中接口变更了，只修改了 Swagger，但是没有及时同步修改 RAP。
 2. 后端开发的接口数据类型和文档不一致，肉眼难以发现问题。
5. 同样，测试在 JMeter 写好的测试用例，真正运行的时候也会发现各种不一致。
6. 时间久了，各种不一致会越来越严重。

Apifox

= Postman + Swagger + Mock + JMeter

核心功能





API 文档

1. 可视化的接口文档管理
2. 支持数据结构定义，多接口复用相同数据结构
3. 接口文档完全遵循 OpenAPI 规范
4. 支持在线分享接口文档



API 调试

1. Postman 有的功能Apifox 都有，并且更好用
2. 自动校验数据结构，自动发现接口返回异常



API 自动化测试

1. 完善的接口场景测试功能
2. 可视化的断言、提取变量功能
3. 支持自定义前置/后置脚本，脚本语法 100% 兼容 Postman
4. 支持调用其他编程语言代码



API 数据 Mock

1. 零配置即可 Mock 出非常人性化的数据
2. 内置 Mock.js 规则引擎
3. 支持自定义期望，灵活配置根据不同参数值返回不同数据内容



CI 持续集成

1. 支持命令行方式运行接口测试 (Apifox CLI)
2. 支持集成 Jenkins 等持续集成工具



数据库操作

1. 支持读取数据库数据，作为接口请求参数使用
2. 支持读取数据库数据，用来校验(断言)接口请求是否成功



自动生成代码

1. 根据接口/模型定义，自动生成各种语言/框架的业务代码和接口请求代码
2. 支持等 130 种编程语言及框架
3. 支持自定义代码模板，满足各种个性化的需求



数据导入/导出

1. 支持导出 OpenAPI (Swagger)、Markdown、Html 等数据格式
2. 支持导入 OpenAPI (Swagger)、Postman 等 10 多种数据格式



团队协作

1. 接口数据云端同步，实时更新
2. 成熟的团队/项目权限管理，满足各类企业的需求

解决的问题

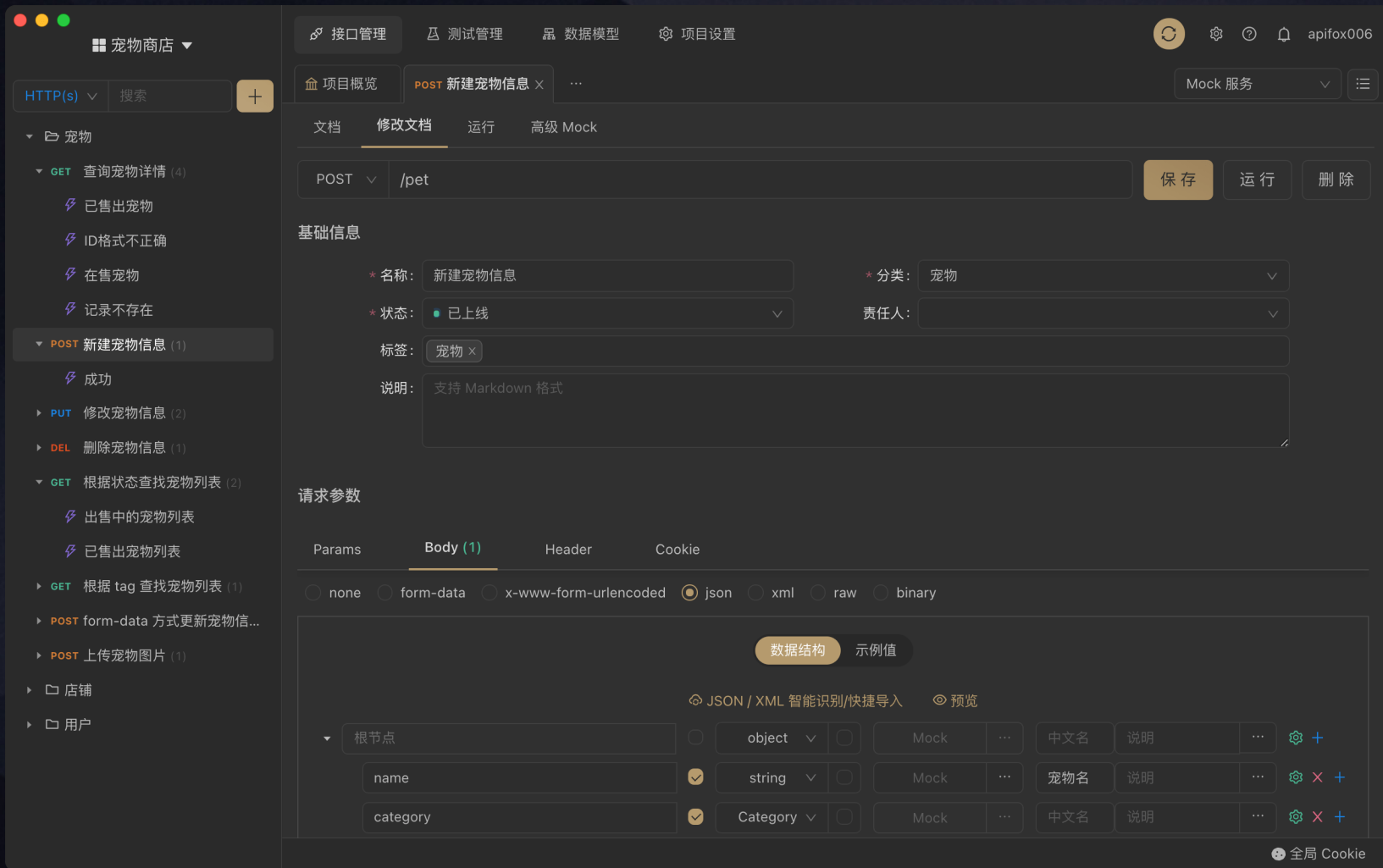
1. 一套系统、一份数据，解决多个系统之间的数据同步问题。
2. 只要定义好接口文档，接口调试、数据 Mock、接口测试即可直接使用，无需再次定义。
3. 接口文档和接口开发调试使用同一个工具，接口调试完成后即可保证和接口文档定义完全一致。
4. 高效、及时、准确！

最佳实践

1. **前端（或后端）**：在 **Apifox** 上定好接口文档初稿。
2. **前后端**：一起评审、完善接口文档，定好接口用例。
3. **前端**：使用系统根据接口文档自动生成的 Mock 数据进入开发，无需手写 mock 规则。
4. **后端**：使用接口用例 调试开发中接口，只要所有接口用例调试通过，接口就开发完成了。如开发过中接口有变化，调试的时候就自动更新了文档，零成本的保障了接口维护的及时性。
5. **后端**：每次调试完一个功能就保存为一个接口用例。
6. **测试人员**：直接使用接口用例测试接口。
7. 所有接口开发完成后，**测试人员**（也可以是**后端**）使用集合测试功能进行多接口集成测试，完整测试整个接口调用流程。
8. **前后端** 都开发完，前端从Mock 数据切换到正式数据，联调通常都会非常顺利，因为前后端双方都完全遵守了接口定义的规范。

接口文档设计

接口文档设计



The screenshot displays the Apifox web application interface for defining a REST API. The interface is organized into several sections:

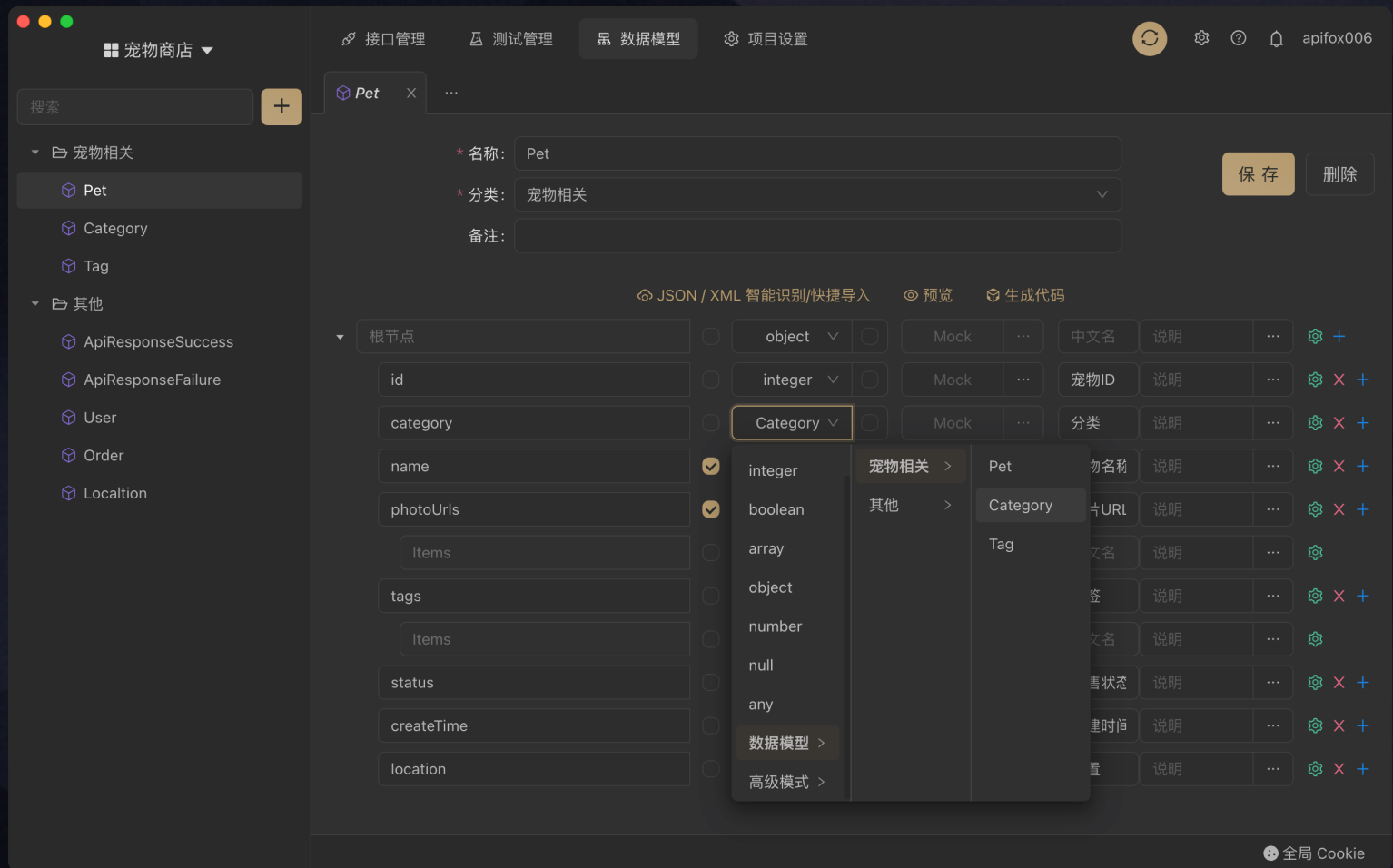
- Header:** Contains navigation tabs for '接口管理' (API Management), '测试管理' (Test Management), '数据模型' (Data Model), and '项目设置' (Project Settings). The current project is 'apifox006'.
- Left Sidebar:** Shows a project tree for '宠物商店' (Pet Store) with various API endpoints listed, such as '查询宠物详情' (Query Pet Details) and '新建宠物信息' (Create Pet Information).
- Main Editor:**
 - Project Overview:** Displays the selected API endpoint: 'POST /pet'.
 - Basic Information:**
 - 名称 (Name):** 新建宠物信息
 - 分类 (Category):** 宠物
 - 状态 (Status):** 已上线
 - 责任人 (Responsible Person):** (Empty field)
 - 标签 (Tags):** 宠物
 - 说明 (Description):** 支持 Markdown 格式
 - 请求参数 (Request Parameters):**
 - Params:** (Empty)
 - Body (1):** Selected tab showing the request body structure.
 - Header:** (Empty)
 - Cookie:** (Empty)
- Body Structure:**
 - Format:** json (Selected)
 - Root Node:** 根节点
 - Structure:**
 - name:** string
 - category:** Category

1.完全可视化

2.零学习成本

3.遵循 OpenAPI 规范

数据模型



1. 完全可视化
2. 支持模型之间嵌套引用
3. 支持 JSON/XML 智能导入
4. 遵循 JSON Schema 规范
5. 支持 oneOf、allOf 等高级组合模式

接口文档

The Apifox interface displays a project named '宠物商店' (Pet Shop). The left sidebar lists various API endpoints, including '查询宠物详情' (Query Pet Details), '新建宠物信息' (New Pet Information), '修改宠物信息' (Modify Pet Information), '删除宠物信息' (Delete Pet Information), '根据状态查找宠物列表' (Find Pet List by Status), '根据 tag 查找宠物列表' (Find Pet List by Tag), 'form-data 方式更新宠物信息' (Update Pet Information by form-data), '上传宠物图片' (Upload Pet Image), '根据状态查询宠物库存数' (Query Pet Inventory by Status), '下单购买宠物' (Place Order to Buy Pet), '查询订单详情' (Query Order Details), '删除订单' (Delete Order), '新建用户' (New User), 'List 方式批量创建用户' (Batch Create Users by List), 'Array 方式批量创建用户' (Batch Create Users by Array), '登录' (Login), '查询用户' (Query User), and '退出登录' (Logout).

The main panel shows the 'POST /pet' endpoint, which is '已上线' (Online). It includes a '基础信息' (Basic Information) section with '宠物' (Pet) as the category. The 'Mock' section shows a table with columns '名称' (Name), '来源' (Source), 'URL / 参数' (URL / Parameters), and '操作' (Action). The table contains one entry: '成功 (200)' (Success (200)) from '接口 Response' (API Response) at 'http://127.0.0.1:4523/mock/396612/pet', with a '快捷调试' (Quick Debug) button. The '请求参数' (Request Parameters) section shows 'Body 参数 (application/json)' (Body Parameters (application/json)) with a table of parameters: 'name' (string, 宠物名), 'category' (object {2}, Category), 'id' (integer, 分类ID), and 'name' (string, 分类名). The '数据结构' (Data Structure) section shows a table with columns '名称' (Name), '来源' (Source), 'URL / 参数' (URL / Parameters), and '操作' (Action). The table contains one entry: '成功 (200)' (Success (200)) from '接口 Response' (API Response) at 'http://127.0.0.1:4523/mock/396612/pet', with a '快捷调试' (Quick Debug) button.

The Apifox interface displays the details of the '查询宠物详情' (Query Pet Details) endpoint, which is 'GET /pet/{petId}' and '已发布' (Published). The '请求参数' (Request Parameters) section shows a table with columns '参数名' (Parameter Name), '位置' (Location), '必填' (Required), '示例值' (Example Value), and '说明' (Description). The table contains one entry: 'petId' (path, 是, 宠物 ID). The '返回 Response' (Return Response) section shows the '成功 (200)' (Success (200)) status with '内容格式: JSON' (Content Format: JSON). The response body is an 'object {8}' (Pet) with the following structure:

- id: integer (宠物ID)
- category: object {2} (分类)
 - id: integer (分类ID)
 - name: string (分类名)
- name: string (宠物名称)
- photoUrls: array[string] (照片URL)
- tags: array[object]{2} (Tag)
 - id: integer
 - name: string (标签名)
- status: string (销售状态) **枚举值**: 'available': 在售 'pending': 待上架 'sold': 已售
- createTime: string (创建时间)
- location: object {3} (位置)

接口用例/接口调试

接口用例



宠物商店

HTTP(s) 搜索 +

宠物

GET 查询宠物详情 (4)

已售出宠物

ID格式不正确

在售宠物

记录不存在

POST 新建宠物信息 (1)

成功

PUT 修改宠物信息 (2)

DEL 删除宠物信息 (1)

GET 根据状态查找宠物列表 (2)

出售中的宠物列表

已售出宠物列表

GET 根据 tag 查找宠物列表 (1)

POST form-data 方式更新宠物信...

POST 上传宠物图片 (1)

店铺

用户

接口管理 测试管理 数据模型 项目设置

项目概览 GET 查询宠物详情 查询宠物详情 (已...)

Mock 服务

发送 保存 删除

GET http://127.0.0.1:4523/mock/396612 /pet/{id}

查询宠物详情 (已售出宠物)

Params (1) Body Header Cookie 前置操作 后置操作 (3) 生成代码

启用	参数名	参数值	说明
☒	id	1	宠物 ID

Path 参数

返回 Response

成功 (200)

返回 Body 返回 Cookie 返回 Header (7) 控制台 (2) 实际请求

状态码: 200 OK 耗时: 27 ms 大小: 223 B

✖ Response 数据结构校验失败:

1. \$.name 应当是 string 类型
2. \$.status 应当是预设定的枚举值之一 (枚举值: available、pending、sold)

断言结果:

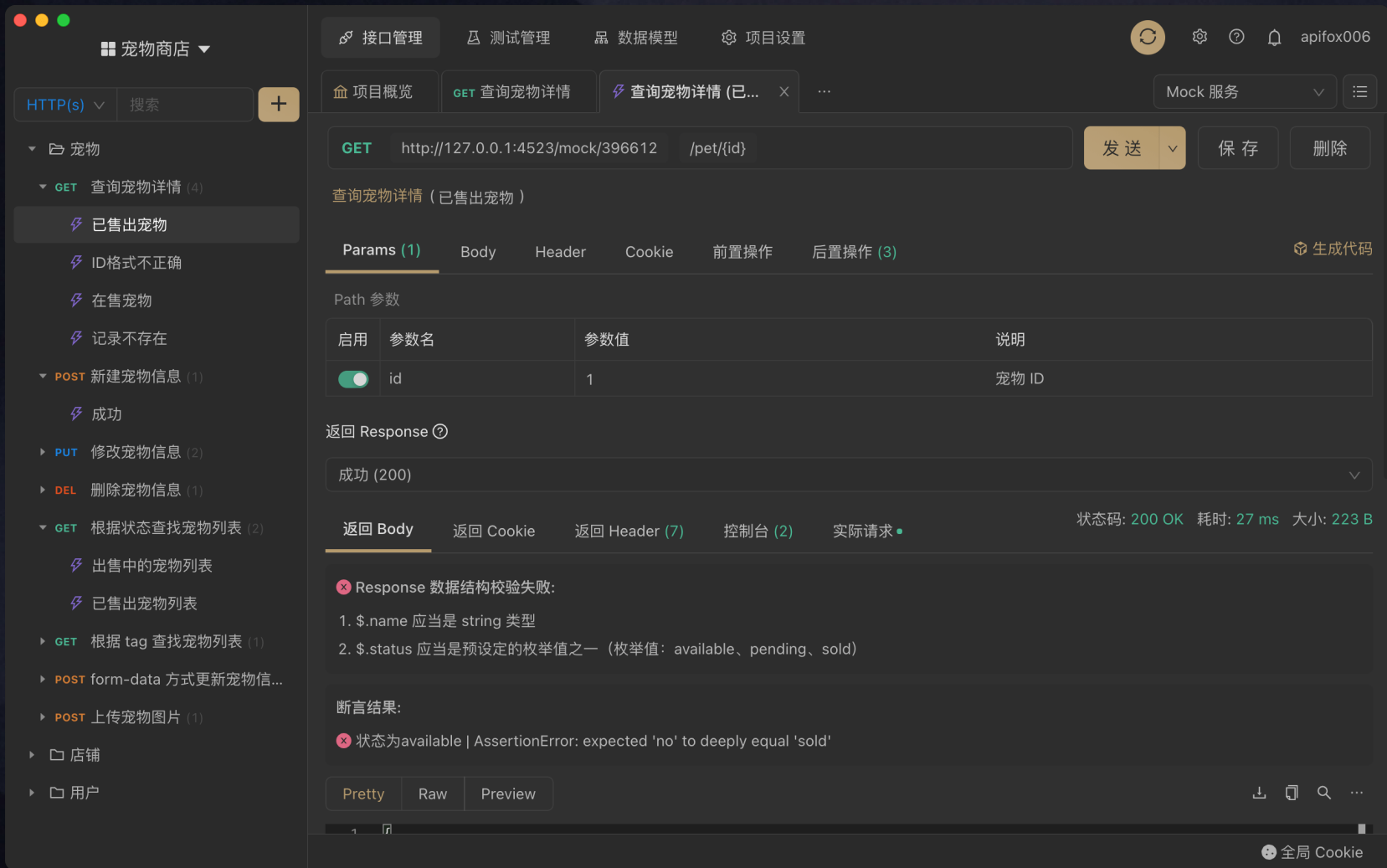
✖ 状态为available | AssertionError: expected 'no' to deeply equal 'sold'

Pretty Raw Preview

全局 Cookie

1. 一个接口多个用例
2. 自动跟随接口变更

接口调试



宠物商店

HTTP(s) 搜索 +

宠物

GET 查询宠物详情 (4)

已售出宠物

ID格式不正确

在售宠物

记录不存在

POST 新建宠物信息 (1)

成功

PUT 修改宠物信息 (2)

DEL 删除宠物信息 (1)

GET 根据状态查找宠物列表 (2)

出售中的宠物列表

已售出宠物列表

GET 根据 tag 查找宠物列表 (1)

POST form-data 方式更新宠物信...

POST 上传宠物图片 (1)

店铺

用户

接口管理 测试管理 数据模型 项目设置

项目概览 GET 查询宠物详情 查询宠物详情 (已... x ...

Mock 服务

发送 保存 删除

GET http://127.0.0.1:4523/mock/396612 /pet/{id}

查询宠物详情 (已售出宠物)

Params (1) Body Header Cookie 前置操作 后置操作 (3) 生成代码

Path 参数

启用	参数名	参数值	说明
☒	id	1	宠物 ID

返回 Response

成功 (200)

返回 Body 返回 Cookie 返回 Header (7) 控制台 (2) 实际请求

状态码: 200 OK 耗时: 27 ms 大小: 223 B

✖ Response 数据结构校验失败:

- \$.name 应当是 string 类型
- \$.status 应当是预设定的枚举值之一 (枚举值: available、pending、sold)

断言结果:

✖ 状态为available | AssertionError: expected 'no' to deeply equal 'sold'

Pretty Raw Preview

1

全局 Cookie

Postman 有的功能 Apifox 基本都有

环境变量、全局变量、前置/后置脚本、Cookie/Session 全局共享等...

环境变量/全局参数

全局

① 全局变量

② 全局参数

环境

正式环境

Mock服务器

+ 新建环境

正式环境

* 环境名称: 正式环境 共用

服务 (前置URL)

服务名	前置URL
默认服务 默认	https://www.baidu.com

+ 添加服务

环境变量

变量名	远程值	本地值	批量修改
token	iNBYWjJYUrGffkRUziiK	iNBYWjJYUrGffkRUziiK	X
变量名			

关闭 保存

全局

① 全局变量

② 全局参数

环境

正式环境

Mock服务器

+ 新建环境

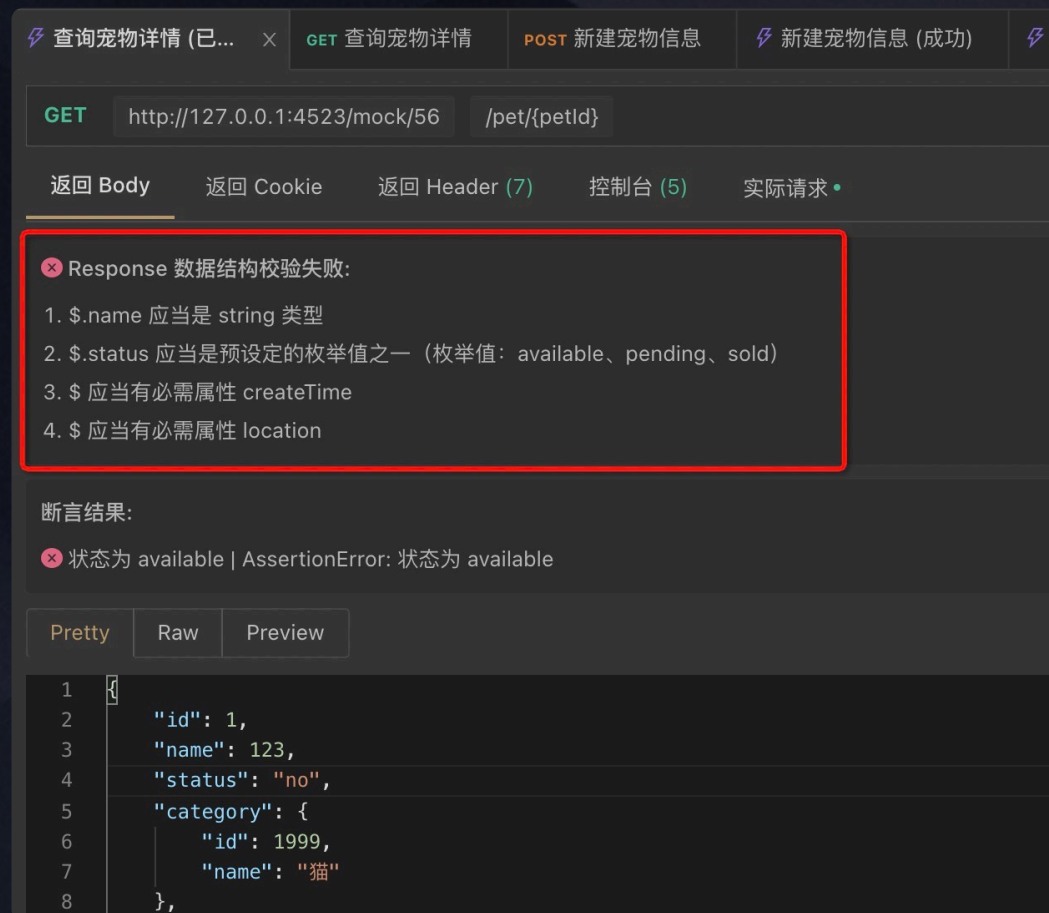
全局参数

Header (1) Cookie Query Body

参数名	必填	默认值	默认启用	说明	批量修改
X-AuthToken	☐	{{token}}	☒		X
添加参数	☐	建议填变量, 如 {{token}}	☒		

关闭 保存

自动校验接口数据



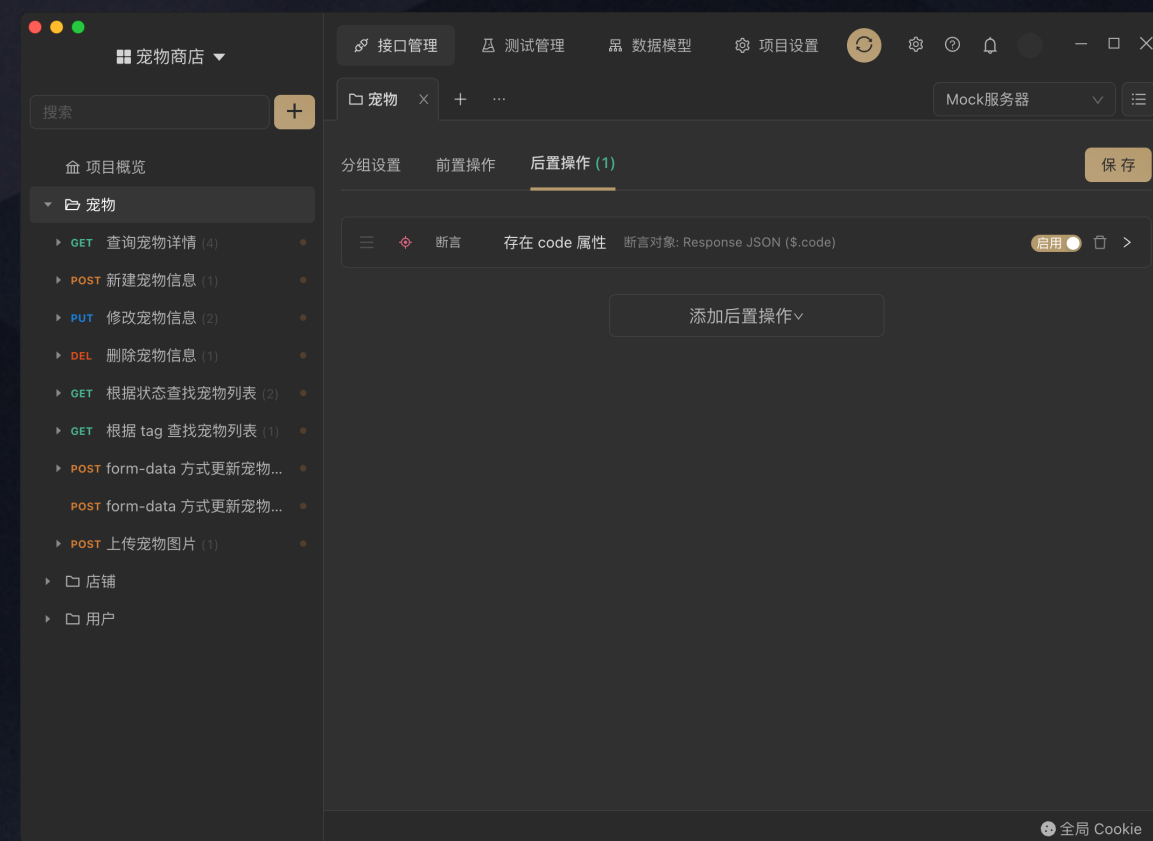
1. 根据数据结构自动校验

2. 完整的 JSON Schema 校验

前置操作/后置操作



针对单个接口



针对整个分组

断言

Params (1)
Body
Headers (1)
Cookies
前置操作
后置操作 (1)
生成代码

三
断言
\$.data.status 等于 sold
断言对象: Response JSON (\$.data.status)

断言名称: 可不填

断言对象: Response JSON
XML(类XML)会智能转 JSON

* 提取表达式 ? : \$.data.status

断言: 等于 sold

返回 Body
返回 Cookie
返回 Header (7)
控制台
实际请求
状态码: 200 OK 耗时: 33 ms 大小: 240 B

Response 数据结构校验通过!

断言结果:

\$.data.status 等于 sold | AssertionError: \$.data.status 等于 sold

Pretty
Raw
Preview

```

1  {
2    "id": 2,
3    "category": {
4      "id": 1999,
5      "name": "猫"
6    },
7    "name": "Hello Kitty",
8    "photoUrls": [
9      "http://dummyimage.com/500x500"
10   ],
11   "tags": [
12     {
13       "id": 1,
14       "name": "Cat"
15     }
16   ],
17   "status": "available"
18 }
```

提取变量

GET http://127.0.0.1:4523/mock/56 /pet/{petId} 发送 保存 删除

Params (1) Body Headers (1) Cookies 前置操作 后置操作 (1) 生成代码

提取变量 petId 临时变量 来源: Response JSON (\$.id)

* 变量名称:

petId

变量类型:

临时变量

提取来源:

Response JSON

XML(类XML)会智能转 JSON

* 提取表达式

\$.id

添加后置操作

返回 Response

成功 (200)

返回 Body 返回 Cookie 返回 Header (7) 控制台 (1) 实际请求 状态码: 200 OK 耗时: 13 ms 大小: 223 B

"已设置临时变量【petId】，值为【1】"

1.可视化

2.JSON Path 提取

数据库操作

GET http://127.0.0.1:4523/mock/56 /pet/{petId} 发送 保存 删除

数据库操作

数据库读取 pet 信息

主数据库

SELECT * FROM pets WHERE id={{petId}}

✕

* 操作名称:

数据库读取 pet 信息

* 数据库连接:

主数据库

▼

☰

* SQL:

SELECT * FROM pets WHERE id={{petId}}

✎

控制台打印结果

🔇

提取结果到变量

变量名	变量类型	JSON Path 表达式	✕
dbPetInfo	临时变量	[\$[0]]	✕
变量名	请选择	如: \$[0].name, 默认为: \$	

添加后置操作

▼

返回 Response

成功 (200)

▼

返回 Body

返回 Cookie

返回 Header (7)

控制台 (1)

实际请求

"数据库运行结果"

▼

```
{
  0: {
    "id": 1
    "name": "kitty"
    "status": "available"
  }
}
```

1. 读取数据库数据

2. 写入数据库数据

自定义脚本

Params (1)BodyHeaders (1)Cookies前置操作后置操作 (1)生成代码

自定义脚本try { // jar 示例, 调用 cn.apifox.Base64EncodeDemo.jar // 实际命令行执行的命令为: java -jar cn.apifox.Base64Encod...

```
1 try {
2   // jar 示例, 调用 cn.apifox.Base64EncodeDemo.jar
3   // 实际命令行执行的命令为: java -jar cn.apifox.Base64EncodeDemo.jar abc
4   const jarResult = fox.execute('cn.apifox.Base64EncodeDemo.jar', ['abc']);
5   console.log('jar 运行结果', jarResult);
6
7   // php 示例, 调用 test.php
8   const param1 = { a: 1, b: 2 };
9   // 注意: json 格式数据作为参数时, 需要使用 JSON.stringify 对参数进行序列化
10  // 实际命令行执行的命令为: php test.php '{"a":1,"b":2}'
11  const phpResultString = fox.execute('test.php', [JSON.stringify(param1)]);
12  // 注意: 返回数据为 json 格式字符串时, 可使用 JSON.parse 反序列化
13  const phpResult = JSON.parse(phpResultString);
14  console.log('php 运行结果', phpResult);
15 } catch (e) {
16   console.error(e.message);
17 }
```

代码片段:
获取环境变量
设置环境变量
获取临时变量
设置临时变量
请求接口
Response 状态码为 200
Response Body 包含字符串
指定字符串
Response Body JSON
值检查
Response Body 字符串
检查

添加后置操作

返回 Response
记录不存在 (404)

返回 Body 返回 Cookie 返回 Header (7) 控制台 (2) 实际请求 • 状态码: 200 OK 耗时: 36 ms 大小: 2 B

"jar 运行结果" "YWJj"

"php 运行结果" {
 "a" : 2
 "b" : 4
}

- 1.语法 100% 兼容 Postman
- 2.支持运行其他任何语言代码

javascript、java、python、php、js、
BeanShell、go、shell、ruby、lua ...

Mock 数据

零配置 Mock 接口数据

```
{
  "username": "黎勇",
  "phone": "13247278136",
  "age": 23,
  "avatar": "http://dummyimage.com/100x100",
  "description": "广高六被严",
  "location": {
    "province": "江西省",
    "city": "丽江市",
    "address": "安徽省日喀则地区石狮市"
  },
  "registerTime": 72567224950,
  "createAt": "1998-07-24 04:05:01",
  "registerIp": "83.18.192.180"
}
```

Apifox

```
{
  "username": "Lorem in in esse",
  "phone": "anim",
  "age": 65335358,
  "avatar": "deserunt aute pariatur sed",
  "description": "in",
  "location": {
    "province": "enim",
    "city": "Duis non dolor",
    "address": "deserunt cillum consecter"
  },
  "registerTime": 94673859,
  "createAt": "in aute magna",
  "registerIp": "culpa enim"
}
```

同类工具

智能 Mock 数据

1. 根据接口定义里的数据结构、数据类型，自动生成 mock 规则。
2. 内置智能 mock 规则库，根据字段名、字段数据类型，智能优化自动生成的 mock 规则。
3. 可自动识别出图片、头像、用户名、手机号、网址、日期、时间、时间戳、邮箱、省份、城市、地址、IP 等字段，从而 Mock 出非常人性化的数据。
4. 支持自定义规则库，满足各种个性化需求。支持使用 正则表达式、通配符 来匹配字段名自定义 mock 规则。

自定义 Mock 规则

* 名称: Pet

* 分类: 宠物相关

备注:

保存 删除

JSON / XML 智能识别/快捷导入 预览 生成代码

根节点		object		Mock	...	中文名	说明	...	
id		integer		Mock	...	宠物ID	说明	...	
category		Category		@ctitle		中文标题			
name	✓	string		@cword		中文词组			
photoUrls	✓	array		@cparagraph		中文大段文本			
Items		string		@csentence		中文句子			
tags		array		@cname		中文姓名			
Items		Tag		@cfirst		中文姓			
status		string		@clast		中文名			
createTime		string		@image		图片链接			
location		Localtion		Mock	...	创建时间	说明	...	
				Mock	...	位置	说明	...	

- 1.支持 Mock.js 语法
- 2.扩展身份证、国内手机号等常用规则

高级 Mock

项目概览

GET 查询宠物详情 ×

🔗 查询宠物详情 (在...

🔗 查询宠物详情 (已...

● POST 新建宠物信息

...

Mock服务器

▼

👁

⚙

文档

修改文档

运行

高级 Mock

+

新建期望

启用 ②	名称	创建人	操作
☒	异常状态宠物	petstore	查看 / 修改 快捷调试 删除
☒	在售宠物	petstore	查看 / 修改 快捷调试 删除
☒	记录不存在	petstore	查看 / 修改 快捷调试 删除
☐	ID格式不正确	petstore	查看 / 修改 快捷调试 删除

修改

* 期望名称

异常状态宠物

期望条件

	参数位置	参数名	参数值
≡	path	petid	1
	▼		

返回数据

☒ JSON ☐ Raw

② 支持 Mock.js 语法

```
1 {
2   "id": 1,
3   "name": 123,
4   "status": "no",
5   "category": {
6     "id": 1999,
7     "name": "猫"
8   },
9   "photoUrls": [
10    "http://dummyimage.com/500x500"
11  ],
12  "tags": [
13    "tag1",
14    "tag2"
15  ]
16 }
```

🔗 格式化

更多设置 ▼

自动化测试

自动化测试

宠物商店

测试用例

测试套件

测试报告

接口管理

测试管理

数据模型

项目设置

apifox006

< 返回

删除

编辑

主流程测试

分类: 默认分类

优先级: P2

创建人: apifox006

最后修改人: apifox006

创建时间: 23 天前

最后修改时间: 几秒前

测试步骤

测试数据

测试报告

已选 8 项

移除

步骤名称	绑定	操作
☒ GET 查询宠物详情 (已售出宠物)	已绑定	设置 移除
☒ GET 查询宠物详情 (ID格式不正确)	已绑定	设置 移除
☒ GET 查询宠物详情 (在售宠物)	已绑定	设置 移除
☒ POST 新建宠物信息 (成功)	已绑定	设置 移除
☒ PUT 修改宠物信息 (成功)	已绑定	设置 移除
☒ DELETE 删除宠物信息 (成功)	已绑定	设置 移除
☒ POST 上传宠物图片 (成功)	未绑定	设置 移除
☒ POST 上传宠物图片 (上传宠物图片)	未绑定	设置 移除

+ 添加步骤

* 运行环境: Mock 服务

测试数据: ☒ 停用

循环次数: 1

线程数: 2

间隔停顿: 0 毫秒

☐ 保存变量变化值

☐ 使用全局 Cookie

☐ 保存 Cookie 到全局

运行

保存

持续集成

导出

全局 Cookie

从【接口用例】导入

宠物 全选

全部 (0)

宠物 (13)

2

店铺 (0)

用户 (0)

GET 查询宠物详情

☒ 已售出宠物

☒ ID格式不正确

☐ 在售宠物

☐ 记录不存在

POST 新建宠物信息

☐ 成功

PUT 修改宠物信息

☐ 成功

☐ 参数有误

DELETE 删除宠物信息

☐ 成功

GET 根据状态查找宠物列表

已选 2 个接口用例

导入模式: ☐ 复制 ☐ 绑定

取消

确定

测试数据

[< 返回](#)

删除编辑

主流程测试

分组：

优先级：P2

创建人：petstore

最后修改人：apifox006

创建时间：1 年前

最后修改时间：2 天前

测试步骤测试数据测试报告

默认数据

添加数据集(行)批量编辑导入/导出

正式环境

Mock服务器

数据集名称	id	name	status	createTi...	添加变量(列)
数据集-1	1	dog	available	2021-10-11	
数据集-2	2	cat	sold	2021-10-11	
数据集-3	3	kitty	sold	2021-10-13	

第 1-3 条/总共 3 条 < 1 > 20 条/页

自动化测试

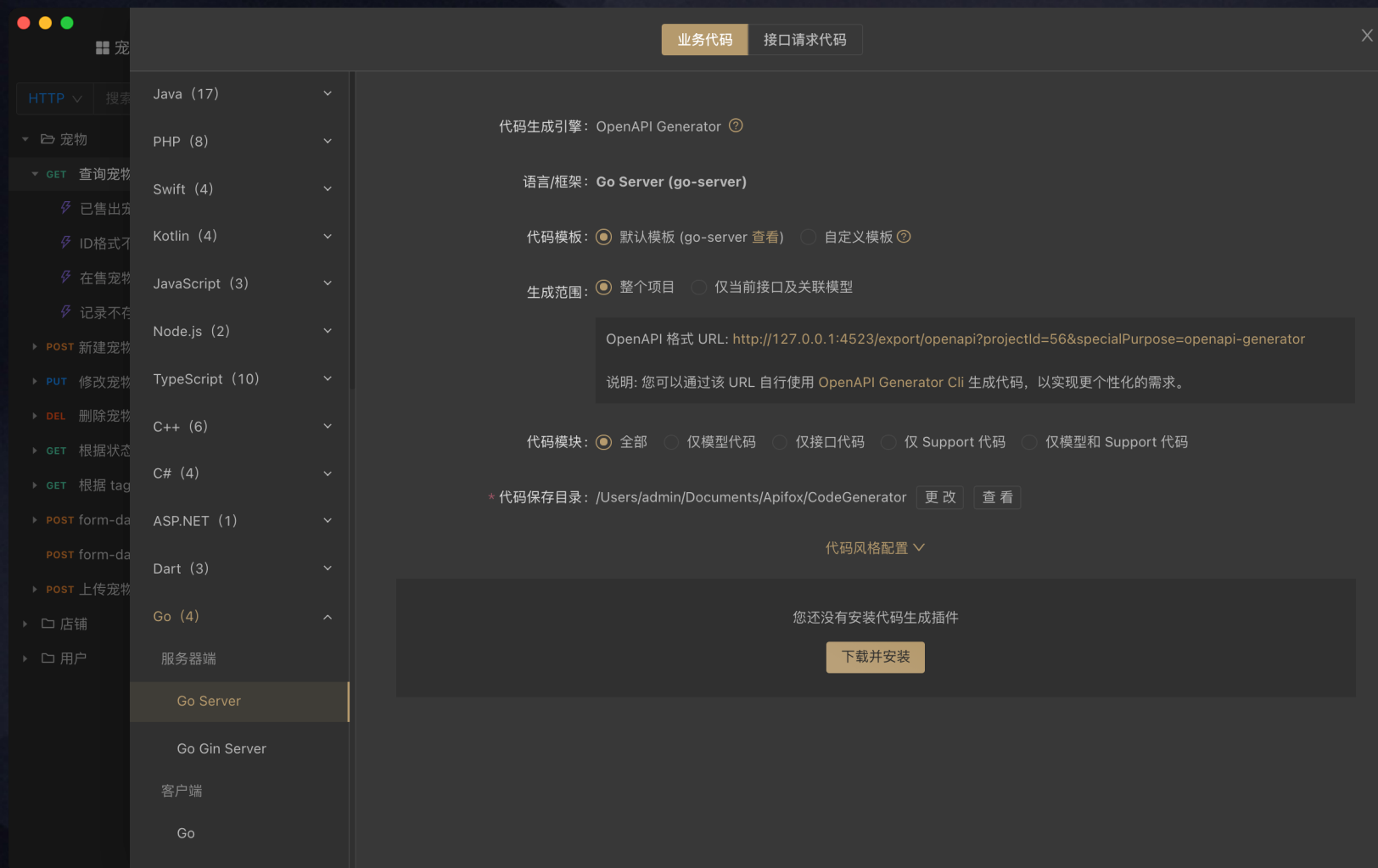


自动化测试



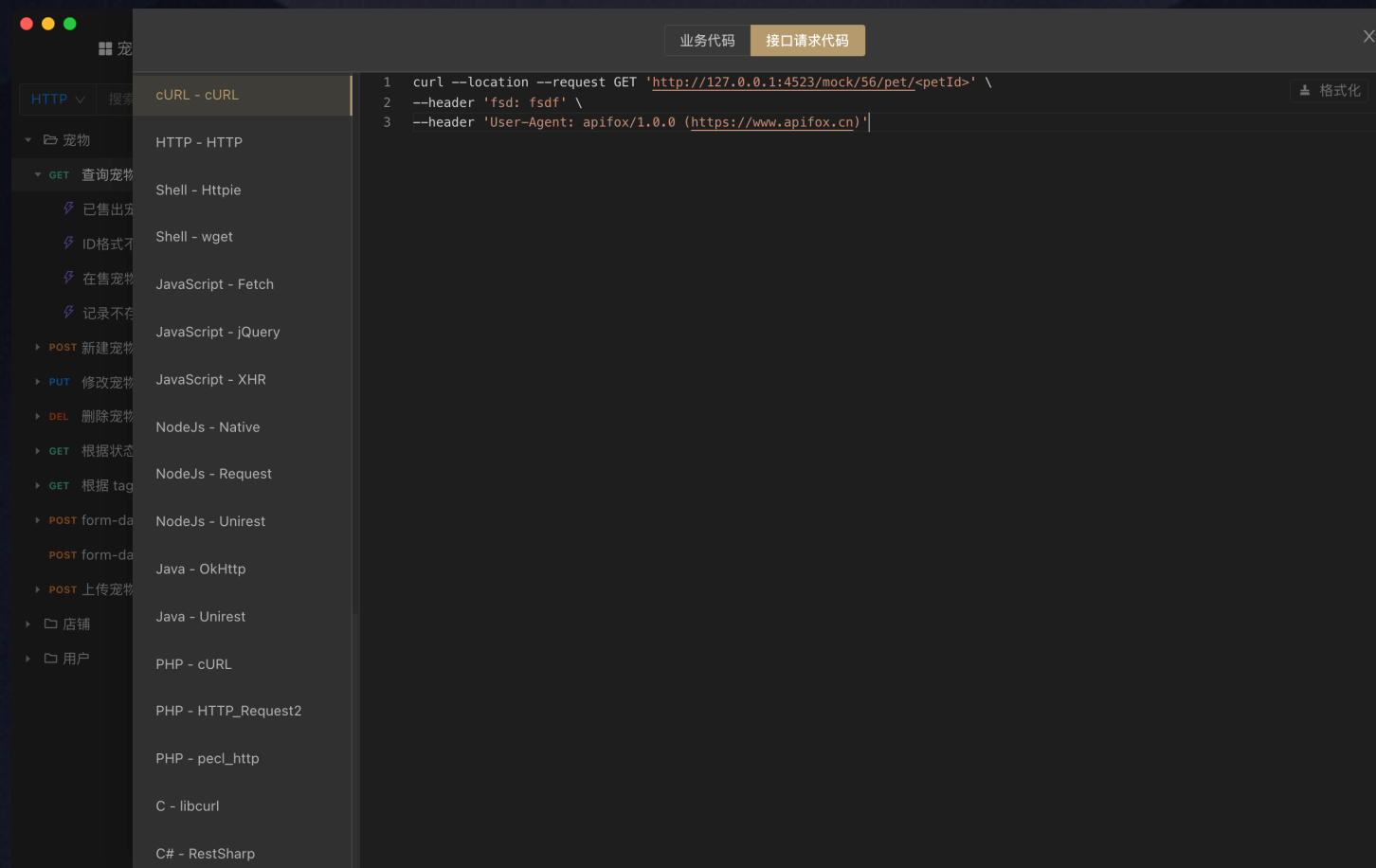
其他特性

生成业务代码



1. 根据接口/模型定义，自动生成各种语言/框架的业务、模型代码。
2. 支持 TypeScript、Java、Go、Swift、ObjectiveC、Kotlin、Dart、C++、C#、Rust 等 130 种语言及框架。

生成接口请求代码



自动生成代码

- 根据接口/模型定义，自动生成各种语言/框架的业务代码和接口请求代码。
- 支持 TypeScript、Java、Go、Swift、ObjectiveC、Kotlin、Dart、C++、C#、Rust 等 130 种语言及框架。
- 支持自定义代码模板，自动生成符合自己团队的架构规范的代码，满足各种个性化的需求。

支持 CI/CD

- 支持命令行方式运行接口测试 (Apifox CLI)。
- 支持集成 Jenkins 等持续集成工具。

```
$ apifox run examples/sample.apifox-cli.json -r cli,html
```

数据导入/导出

- 支持导出 OpenAPI (Swagger)、Markdown、Html 等数据格式。
- 支持导入 OpenAPI (Swagger)、Postman、HAR、RAP2、JMeter、YApi、Eolinker、RAML、DOClever、Apizza、DOCWAY、ShowDoc、I/O Docs、WADL、Google Discovery 等数据格式。

团队协作

- 接口数据云端同步，实时更新。
- 成熟的团队/项目权限管理，支持管理员、普通成员、只读成员等角色设置，满足各类企业的需求。

Thanks.

Apifox.cn

节省研发团队的每一分钟