



第8届 AI+ Development  
Digital Summit

# AI+研发数字峰会

拥抱AI 重塑研发

11月14-15日 | 深圳





# 2026 AI+ PRODUCT INNOVATION SUMMIT

01.16-17 · ShangHai

## AI+产品创新峰会



### Track 1: AI 产品战略与创新设计

从0到1的AI原生产品构建

论坛1: AI时代的用户洞察与需求发现

论坛2: AI原生产品战略与商业模式重构

论坛3: Agentic AI产品创新与交互设计

### 2-hour Speech: 回归本质



用户洞察的第一性

——2小时思维与方法论工作坊

在数字爆炸、AI迅速发展的时代，  
仍然考验“看见”的“同理心”



### Track 2: AI 产品开发与工程实践

从1到10的工程化落地实践

论坛1: 面向Agent智能体的产品开发

论坛2: 具身智能与AI硬件产品

论坛3: AI产品出海与本地化开发

### Panel 1: 出海前瞻



“出海避坑地图”圆桌对话

——不止于翻译:

AI时代的出海新范式



### Track 3: AI 产品运营与智能演化

从10到100的AI产品运营

论坛1: AI赋能产品运营与增长黑客

论坛2: AI产品的数据飞轮与智能演化

论坛3: 行业爆款AI产品案例拆解

### Panel 2: 失败复盘



为什么很多AI产品“叫好不叫座”?

——从伪需求到真价值:

AI产品商业化落地的关键挑战

智能重构产品 数据驱动增长  
Reinventing Products with Intelligence, Driven by Data

80+

业界大咖

汇聚产品大咖的真知灼见

40+

创新案例

开拓全新AI+产品视角

10+

分论坛

涵盖产品到商业增长的全链路

800+

听众规模

共同探索产品的智能重构

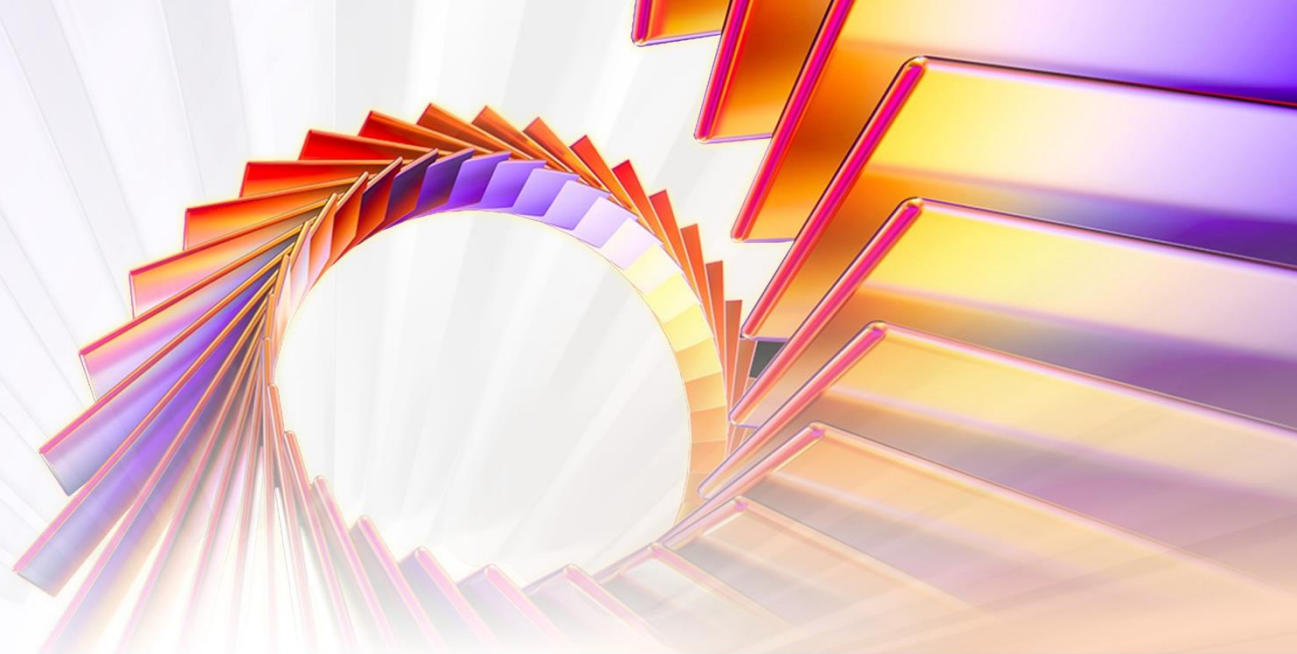


扫码查看会议详情

**AiDD** 8<sup>th</sup> 2025 | 11月14-15日 | 深圳

# 第8届 AI+ Development AI+研发数字峰会

拥抱AI 重塑研发



## 接口测试Agent在财付通 (微信支付) 的探索实践

余春龙 | 腾讯



## 余春龙

腾讯金融科技 支付清算测试负责人

---

腾讯T12技术专家/资深架构师，在互联网领域超15年工作经验，先后主导过广告/电商/智能风控等多个大型复杂业务的架构设计、AI应用工作，目前在财付通/支付质量中心探索AI大模型的落地，全面推进质量工作的智能化。

# 目录

## CONTENTS

- I. 业界接口测试自动化的常见思路
- II. 支付领域接口测试面临的几个难点
- III. 接口测试Agent设计思路
- IV. 接口测试元数据建模
- V. 资金安全方法论在接口异常测试上的应用
- VI. 总结与下一步计划

# PART 01

## 业界接口测试自动化的常见思路

# ► 背景 – 我们为什么要重点搞接口测试？

## 1. 测试左移的频谱（测试与开发的协作模式）



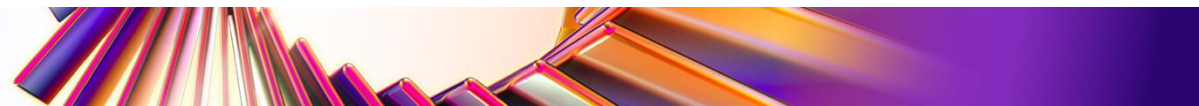
## 2. 接口测试的定位



(1) 接口测试，这里仅指“单接口”测试，不包括“多接口编排”。接口测试介于“单元测试”和“集成测试”之间，是开发在日常自测中使用最广的测试方式。

(2) 在大型分布式系统中，尤其微服务拆分很细的情况下，微服务的接口测试替代了大部分传统单体应用的单元测试，即质量与效率之间取得一个平衡

(3) 所以，接口测试工具的用户主要是“开发”，不是“测试”，赋能开发，提升开发自测效率



| 大类     | 常见思路          | 问题                                                                                                                                                                                                                                         |
|--------|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 非AI方法  | 1. 流量录制与回放    | 1. 只能做回归测试，新接口不行<br>2. 需要非常完善的微服务框架 + 公共函数库 + 中间件，要能mock所有外部依赖（包括下游接口/DB/KV/本地Linux函数，比如时间戳、随机数...）<br>3. 数据安全（现网敏感数据不能直接在测试环境回放，各种加密/签名/脱敏）<br>4. 边界场景在现网长时间没流量，异常场景现网没有<br>5. 录制过程可能影响现网稳定性                                              |
|        | 2. 流量 => 用例   | 从数据源来说，总共就这几类：<br>接口契约 + 流量 + 文档 + 代码 + 专家经验<br><br>但普遍面临以下几类问题：<br>(1) 文档质量问题，接口契约/需求文档/技术文档，信息“缺”，“歧义”<br>(2) 大模型幻觉：尤其是让大模型读复杂的需求文档/技术文档，幻觉都很严重，不太可行，完全不可控<br>(3) 从代码->用例，本身存在悖论。代码本身如果有bug，生成的用例也有bug<br>(4) 如果只是生成文本用例，文本 => 代码，还需要人写。 |
|        | 3. 接口契约 => 用例 |                                                                                                                                                                                                                                            |
|        | 4. 需求文档 => 用例 |                                                                                                                                                                                                                                            |
| 基于AI方法 | 5. 业务代码 => 用例 |                                                                                                                                                                                                                                            |



## PART 02

# 支付领域接口测试面临的几个难点

# ► 支付领域做接口测试自动化，面临哪些问题？

除了前面，基于AI的方法面临的通用问题：文档质量 + 模型幻觉 之外，我们还面临以下几类典型问题：

## 1. 前置造数困难

大部分接口都有前置依赖，单接口无法直接测试，比如：

- 要测支付接口，依赖下单接口，下单接口又依赖用户创建接口；
- 要测退款接口，依赖支付接口；
- 要测分期付款/借款接口，依赖支付接口；
- 要测分期付款/还款接口，依赖借款接口；

## 2. 测试场景想“全”了吗？

测试领域的灵魂拷问之一，对于支付/金融来说，这个问题尤为被关注

## 3. 异常用例生成

异常有哪些分类？异常能被穷举吗？



# ▶ 支付领域做接口测试自动化，面临哪些问题？

## 4. 增量生成

很多场景，不是新增接口，而是修改老接口。  
接口修改之后，新生成的一批用例不能直接覆盖老的用例，  
如何自动diff，自动合并？

## 5. DB断言自动生成

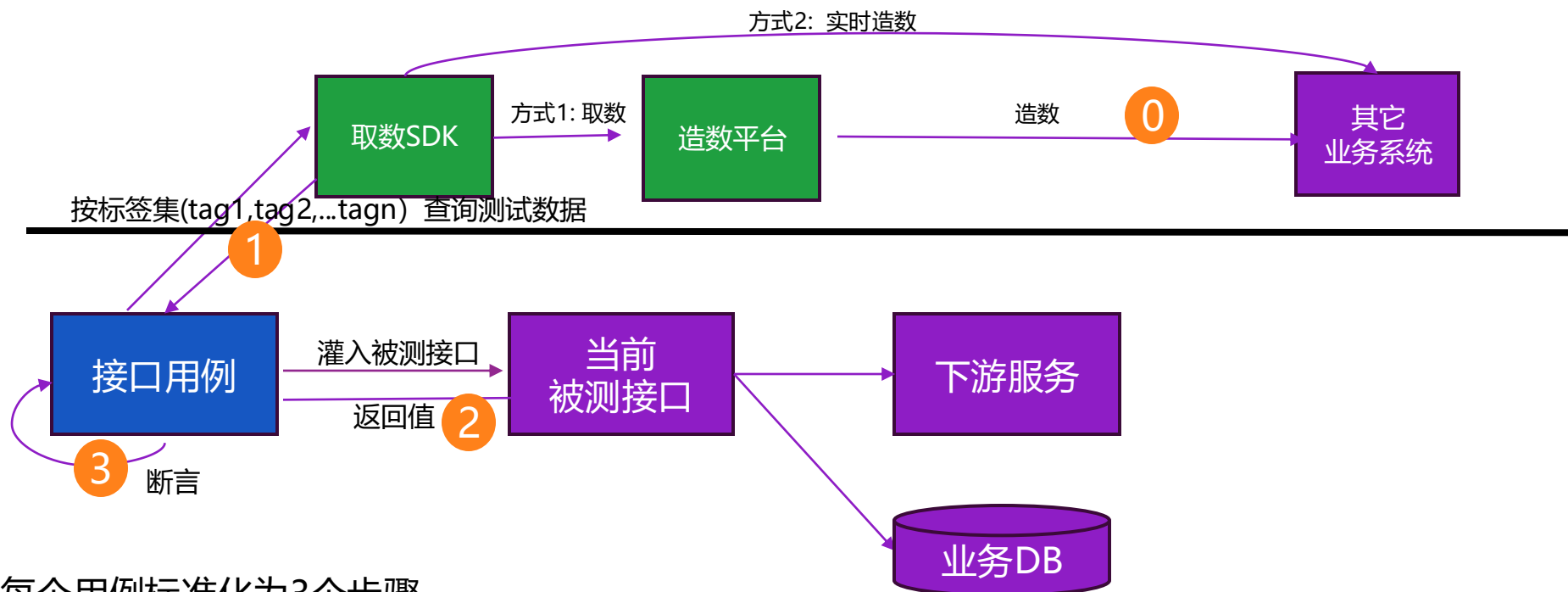
DB断言的准确度能做到多少？



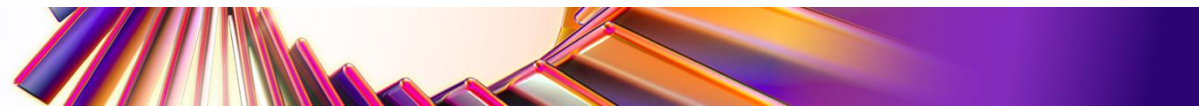
## PART 03

# 接口测试Agent设计思路

## 措施1 – 通过造数平台，解耦被测接口和其前置依赖

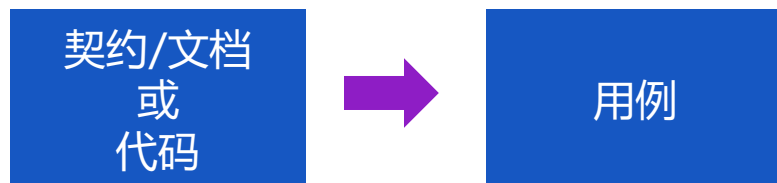


1. 每个用例标准化为3个步骤  
(1) 取数 (2) 发请求 (3) 断言
2. 造数，分为提前造和实时造  
(1) 提前造，即屯数，基于master，适用于前置依赖稳定的场景  
(2) 实时造，即早即用，基于特性分支，适用于前置依赖不稳定、一起变更的场景
3. 无论提前造，还是实时造，对于用例来说，不感知。用例感知的是一个一个的“数据对象”
4. 造数本身的AI化，不在此项目考虑
5. 开发/测试的分工：测试负责造，开发负责用

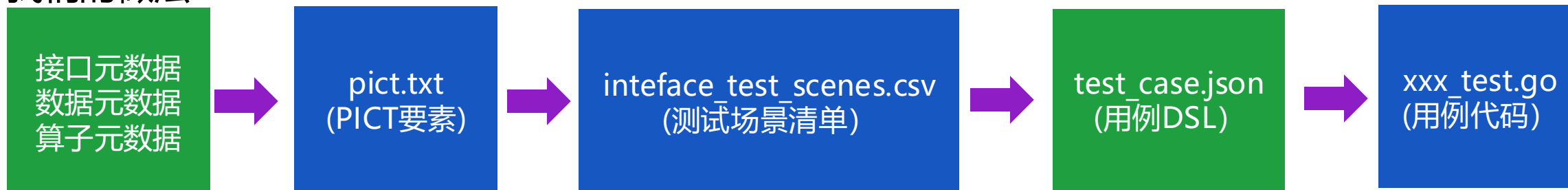


## 措施2 – workflow拆解 + 元数据建模，精确控制模型生成质量

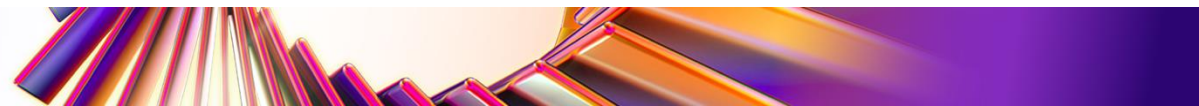
业界做法：



我们的做法：



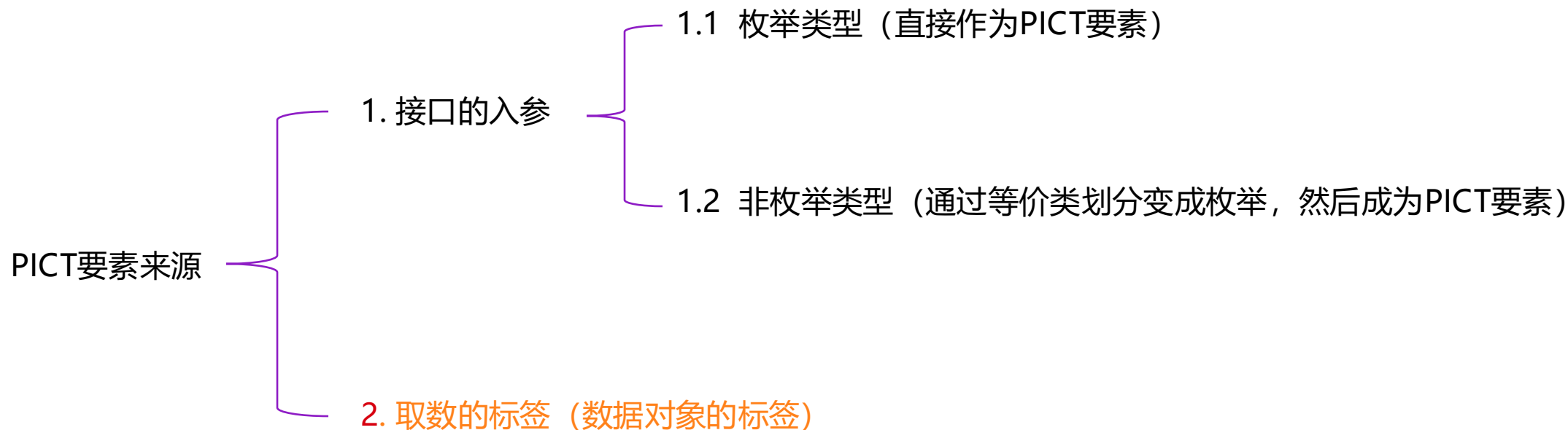
1. 我们没有直接从文档/代码出发，而是定义了相关的元数据结构，精确描述所需的业务信息。再想各种办法，去智能填充这个结构，相当于给模型加了一个“围栏”，减少漫无边界的生成
2. 通过工程流拆解，把用例的生成过程，拆解的足够细
3. 最终用例是Go代码，但并没有直接生成Go。而是先用json定义一个用例的DSL语言，同样是给模型加了一个“围栏”，让其生成的东西，足够可控。





## 措施3 – PICT测试方法论与造数相结合，保证测试场景“全”

PICT是微软发明的一个经典的测试场景分析的方法，但是只有接口的出/入参无法确定所有的PICT要素，要把所造的数据的种类也纳入PICT考虑范围



备注：接口的返回字段、取数的返回字段，不用作为PICT要素



## ► 措施3 – 2个度量手段，反向验证用例是否“全”

度量手段1:  
现网流量 -> 接口的场景列表

step1: 通过现网流量，归纳出每个接口有多少种参数的排列组合（即测试场景）

step2: 测试用例场景数  $\geq$  现网场景数

度量手段2:  
函数调用链-> 接口的行覆盖率

step1: 扫描代码，得到每个接口的静态函数调用链

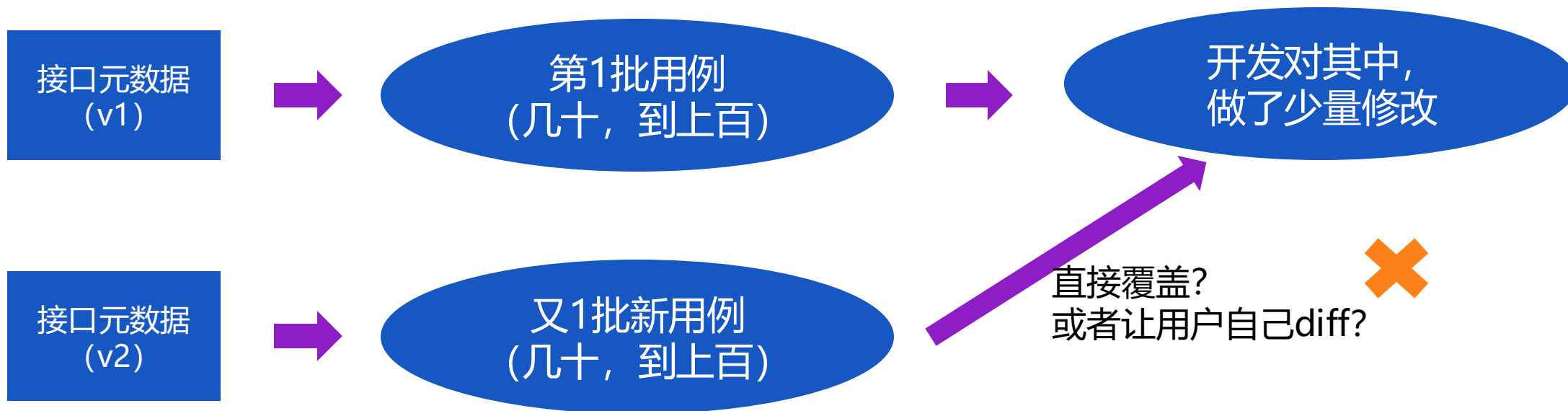
step2: 用例跑完之后，精确计算调用链上每个函数的行覆盖率

覆盖率低，不代码用例不全，可能有其他原因：

- (1) 某些代码废弃了，永远走不到
- (2) 某些公共函数的某些分支，被其他接口使用，不被当前接口使用



## 措施4 – 增量生成



当用户修改了接口之后，需要知道具体修改了什么，是新增字段值、还是新增字段、或者接口没动改了业务逻辑？

哪些用例需要新增，哪些用例需要修改？



## ► 措施4 – 增量生成 – 拆解修改场景，逐场景做对应方案

### 修改场景

1. 接口没动，改了内部逻辑

2. 新增字段值

3. 新增字段

4. 2和3混合

### 对应解决办法

用例保持不动

1. PICT生成的新/老场景列表，做diff，找出哪些新增场景，新增对应用例

1. PICT生成的新/老场景列表，做diff，找出哪些新增场景，新增对应用例  
2. 老用例，统一修改发包接口；

类似上面处理



# PART 04

## 接口测试元数据建模

# 接口元数据

接口契约或者接口文档，包含的信息还是太少了，不足以给大模型来生成用例。  
我们在接口契约的基础上，定义了“接口元数据”的数据模型，精确描述一个接口的完整信息。

## 接口元数据分3个部分

p1: 前置依赖数据的描述

p2: 每个字段的【生成方式、生成规则、等价类划分】

p3: 测试场景PICT语言

## 数据元数据

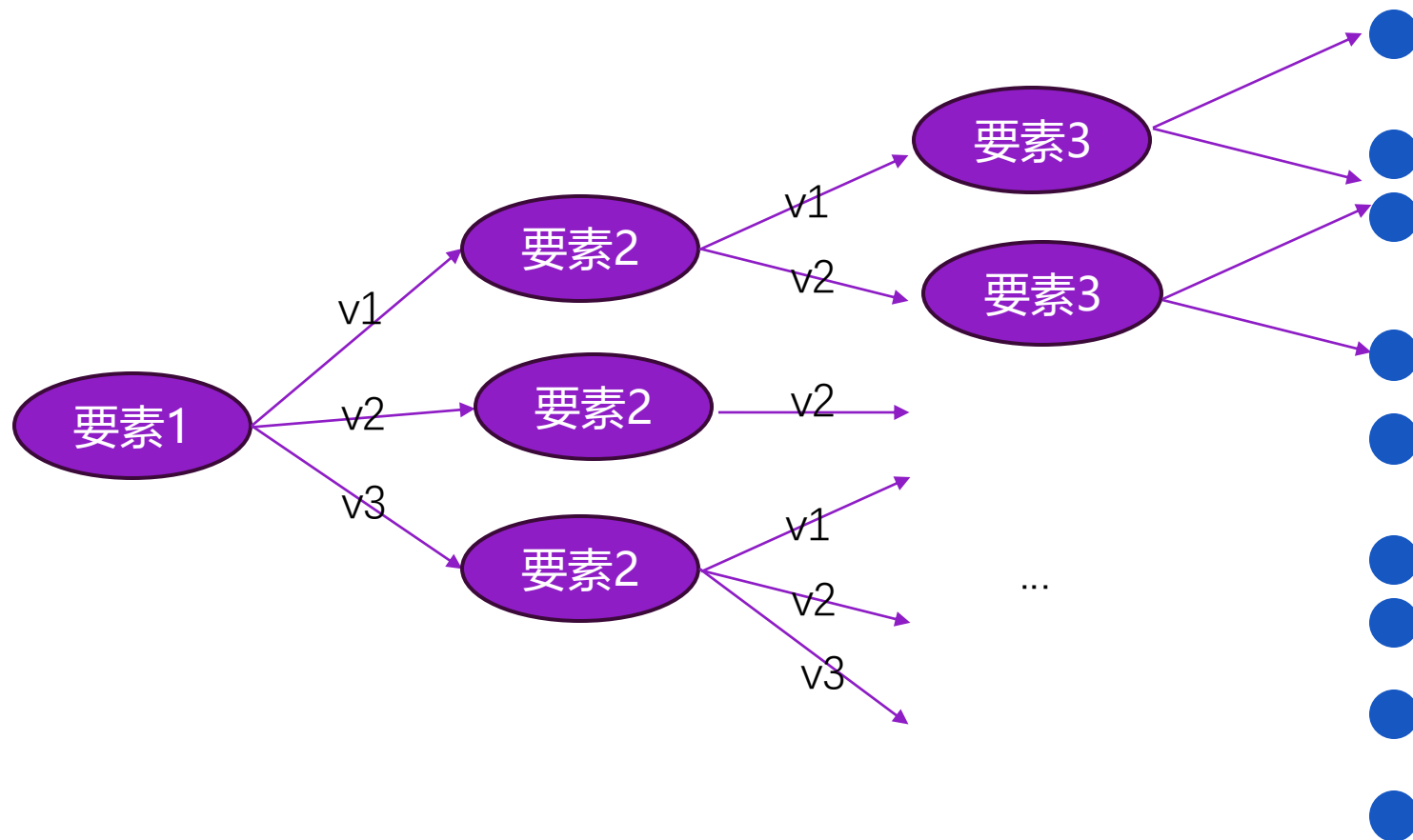
| 字段名(举例)       | 生成方式                     | 生成规则                              | 等价类划分                          |
|---------------|--------------------------|-----------------------------------|--------------------------------|
| serial_number | 计算<br>(这里要引入算子与DSL计算表达式) | 随机值: rand(1,100)                  | 空                              |
| xxx_state     | 计算                       | enum(0,1,2)                       | 同生成规则                          |
| xxx_sign      | 计算                       | md5(p1 + p2 + ..)                 | 空                              |
| age           | 计算                       | range(0,200)                      | [age < 0, age <= 18, age > 18] |
| timestamp     | 计算                       | currentTime()                     | [白天, 晚上]                       |
| bank_number   | 计算                       | 正则表达式xxx                          | 空                              |
| xxx_state     | 造数平台                     | \$payment_receipt.state<br>(造数平台) | [0,1,2]                        |
| user.uid      | 造数平台                     | \$payment_receipt.user.uid        | [小贷、联合贷]                       |

| PICT关键要素                                                                             | 组合粒度 | 备注                           |
|--------------------------------------------------------------------------------------|------|------------------------------|
| 借贷模式: 助贷, 联合贷                                                                        | 全组合  |                              |
| 分期数: 分3期, 分6期, 分12期                                                                  | 成对组合 |                              |
| 营销减免: 无营销, 部分营销, 全额营销                                                                | 成对组合 |                              |
| 逾期状态: 未逾期, 连续逾期, 间隔逾期                                                                | 全组合  |                              |
| 逾期资方: 无资方逾期, 单资方逾期, 多资方逾期                                                            | 成对组合 | 联合贷单资方逾期: 某个还款计划小贷先还清然后逾期    |
| 罚息金额: 无罚息, 罚息为0, 罚息1分, 罚息>1分                                                         | 成对组合 | 某期还款计划逾期后单日计的罚息金额            |
| 还款时间: 借当日还, 14天内还款, 15天后还款                                                           | 成对组合 |                              |
| 还款类型: 按期还款, 提前结清, 退款还款                                                               | 全组合  |                              |
| 还款金额: 部分还款, 足额还款                                                                     | 成对组合 | 随机范围可以使用: 0、1、x/2、x-1 (x为待还) |
| { 借贷模式, 逾期状态, 还款类型 } @ 3                                                             |      |                              |
| 部分 2                                                                                 |      |                              |
| 部分 3                                                                                 |      |                              |
| IF [分期数] = "分3期" AND [逾期状态] = "间隔逾期" THEN [还款类型] <> "提前结清";                          |      |                              |
| IF [借贷模式] = "助贷" THEN [逾期资方] <> "多资方逾期";                                             |      |                              |
| IF [逾期状态] = "未逾期" THEN [逾期资方] = "无资方逾期" AND [罚息金额] = "无罚息" AND [还款优惠] = "无优惠";       |      |                              |
| IF [逾期状态] <> "未逾期" THEN [逾期资方] <> "无资方逾期" AND [罚息金额] <> "无罚息" AND [还款时间] = "15天后还款"; |      |                              |
| IF [借贷模式] = "联合贷" AND [逾期资方] = "单资方逾期" THEN [罚息金额] = "罚息为0";                         |      |                              |
| IF [还款类型] = "按期还款" THEN [还款时间] = "15天后还款";                                           |      |                              |
| IF [还款类型] = "提前结清" THEN [还款金额] = "足额还款";                                             |      |                              |

PICT语言的3个部分:

- (1) PICT要素
- (2) 要素之间的组合关系
- (3) 要素之间的约束关系

## ▶ 接口元数据 - PICT的可视化展示（决策树）

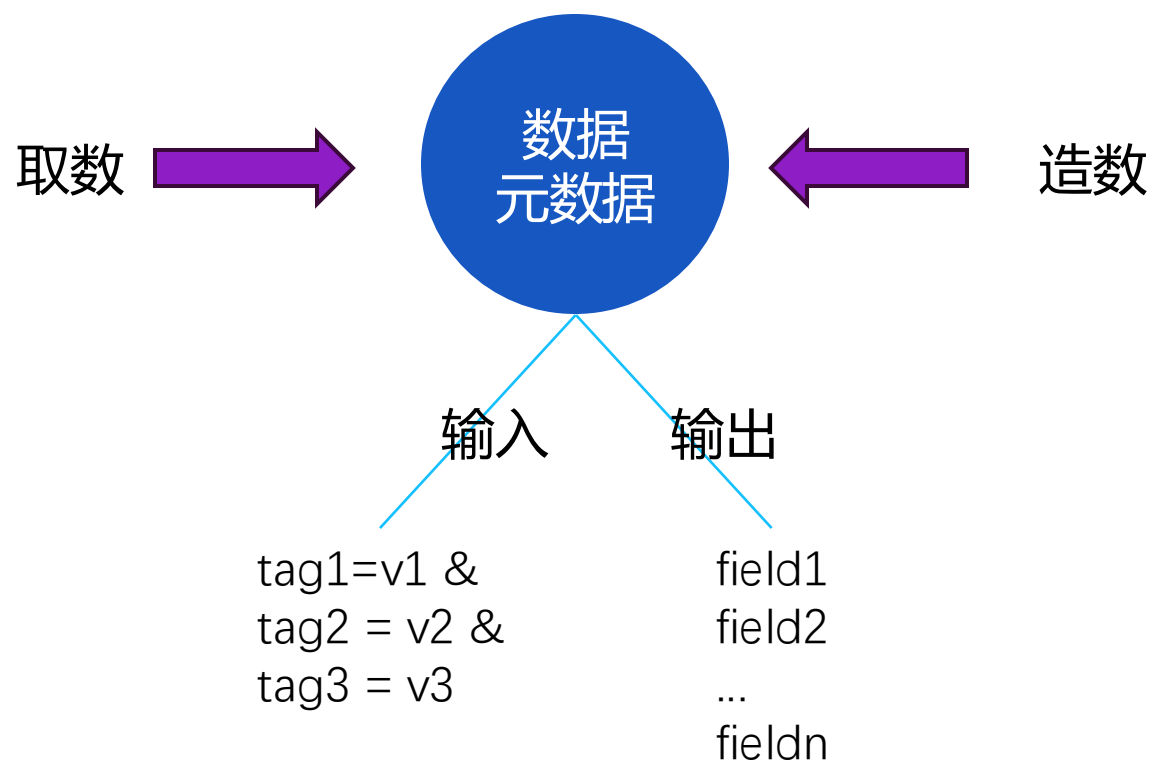


每个接口，一棵决策树，表达这个接口的所有测试场景  
树的1个叶子节点 = 1个测试用例（测试场景）



## ► 数据元数据 – 对造数结果建模，而不是对过程建模

无论是哪种造数过程（实时造数，还是造数平台提前造），造数结果的数据模型是一致的。这里，我们定义数据元数据，对造数结果建模。对于用例来说，它只管取数，不管数是如何造出来的。



数据元数据的种类（按业务领域划分）

- 用户
- 商户
- 支付单
- 退款单
- ...



# ▶ 数据元数据 – 和DB的表Schema不是一回事

## 1. 关注点不一样

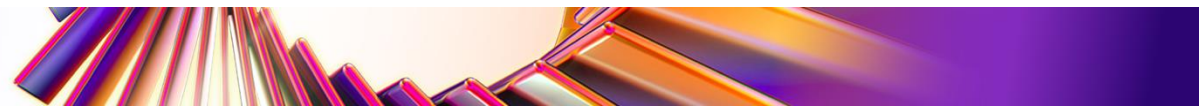
对于一个被测接口来说，它不关注上游业务系统的DB Schema，关注的是“如何获取上游数据，填充对应的被测接口参数”

## 2. 结构不一样

1个数据对象可能包含多个表Schema的数据，是一个嵌套结构，反映的是多个前置API的调用结果“累积”，比如“借据”嵌套包含“用户”

## 3. 内容不一样

造数的字段 = DB字段 + 接口返回字段，而不是仅DB字段



## 1. 为什么需要算子?

很多复杂的接口参数，其计算逻辑在接口契约上反映不出来，需要额外的信息来表达其计算规则

## 2. 抽象1套算子库，表达接口参数的计算逻辑

(1) 算子库，维护常用的计算逻辑：

- 四则运算；
- 随机数、时间戳；
- 加/解密、签名；
- ID生成（订单ID、xxx事件ID）；
- 3元表达式；
- ...

(2) 通过基本算子之间的组合，即函数表达式，表达参数的计算逻辑。例子：

```
- name: sms_flag
  type: string
  length: null
  required: false
  description: null
  value: condition.Ternary($busi_sms_flag==1, "cft", "")
```



## ▶ 接口元数据 - 智能填充

手工填写接口元数据，工作量比较大，这里探索了各种智能填充方式

### 1. 基于现网流量

1. 枚举类型，pb里面没有定义枚举值，从现网流量归纳枚举值，自动补全
2. 参数值之间的约束关系

### 2. 基于pb的备注/注释

用pb中的注释，自动补全接口参数的元数据

### 3. 数据元数据与接口参数的智能匹配

用数据元数据，自动补全接口参数的生成规则

### 4. 算子元数据与接口参数的智能匹配

用算子元数据，自动补全接口参数的生成规则

### 5. 基于数据字典

从历史接口元数据中，提取字段；  
新接口，有相同/类似命名的参数，自动采用之前的字段和元数据

智能填充能填70%字段，剩下30%还是需要人手工补



## ▶ 接口元数据 - 智能填充（案例）

```
// 授权书url
// {{limit: 1-255, required:true}}
optional string auth_url = 10;
// memo信息
// {{limit: 0-128, required:false}}
optional string memo = 11;
// 是否出境 1:是 2:否
// 目前仅支持不出境场景
// {{limit: 2-2, required:true}}
optional int32 go_abroad = 12;
```

```
- name: uid
  type: int64
  length: 64
  required: true
  description: 用户唯一标识
  source: data_platform
  rule: $loan.user.uid 自动填充
```

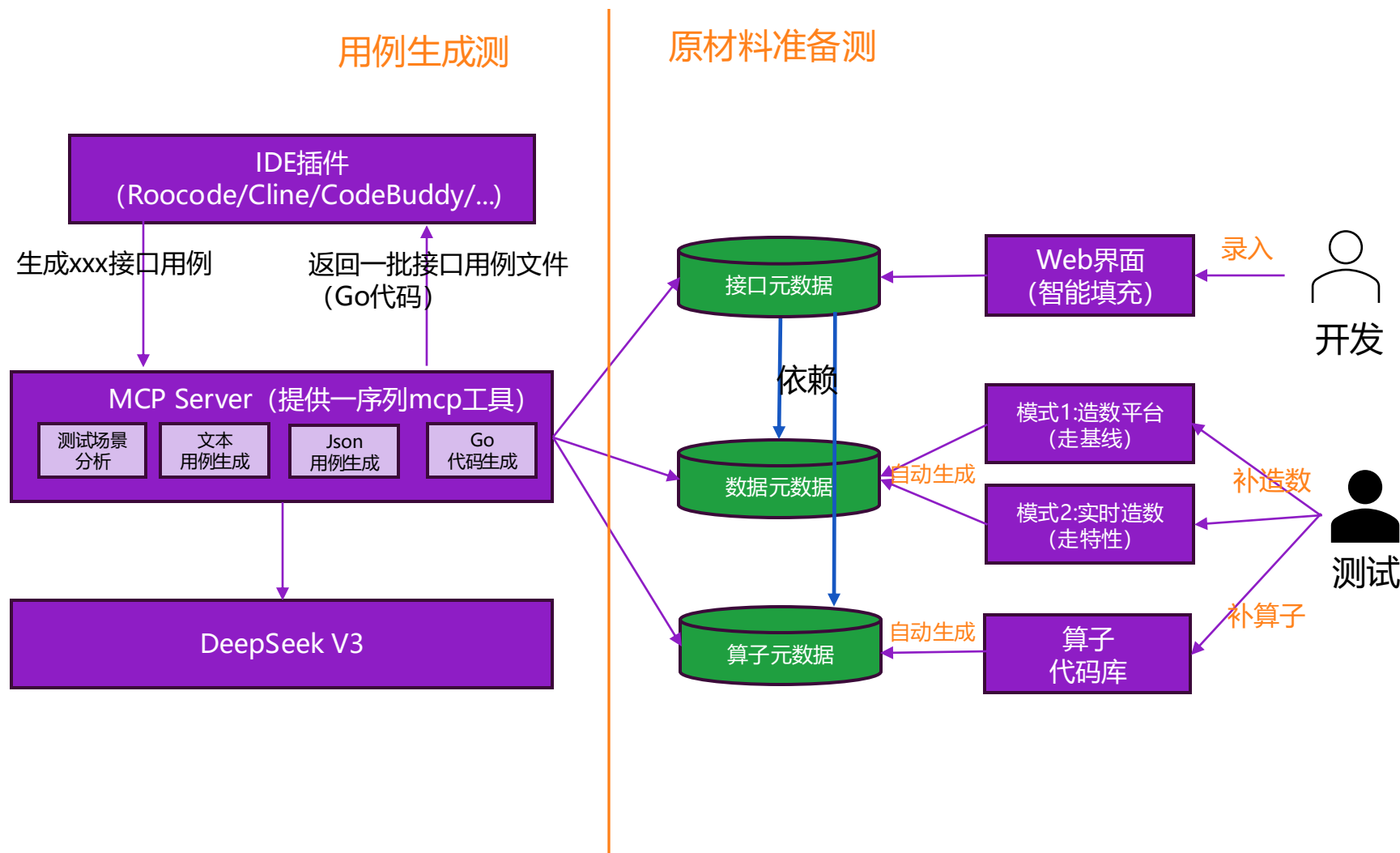
```
- name: uin
  type: string
  length: 0
  required: false
  description: 微信支付用户uin
  source: data_platform
  rule: $loan.user.uin 自动填充
```

```
- name: data_ver
  type: int64
  length: 64
  required: true
  description: 用户账户版本号
  source: data_platform
  rule: $loan.user.data_ver 自动填充
```

```
- name: contract_no
  type: string
  length: 64
  required: true
  description: 用户合同编号
  source: data_platform
  rule: $loan.user.contract_no 自动填充
```

```
- name: event_id
  type: string
  required: true
  description: 事件ID
  source: compute
  rule: generator.AssetCreditEventId()
```

# 系统整体架构



## PART 05

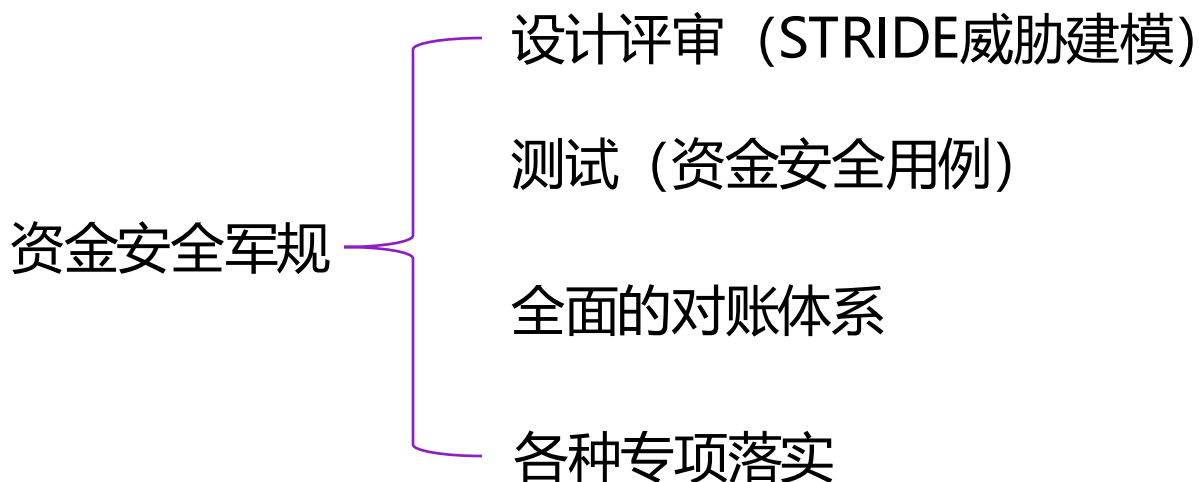
# 资金安全方法论在接口异常测试上的应用

## ► 背景 – 资金安全方法论

1. 金融领域都非常重视资金安全

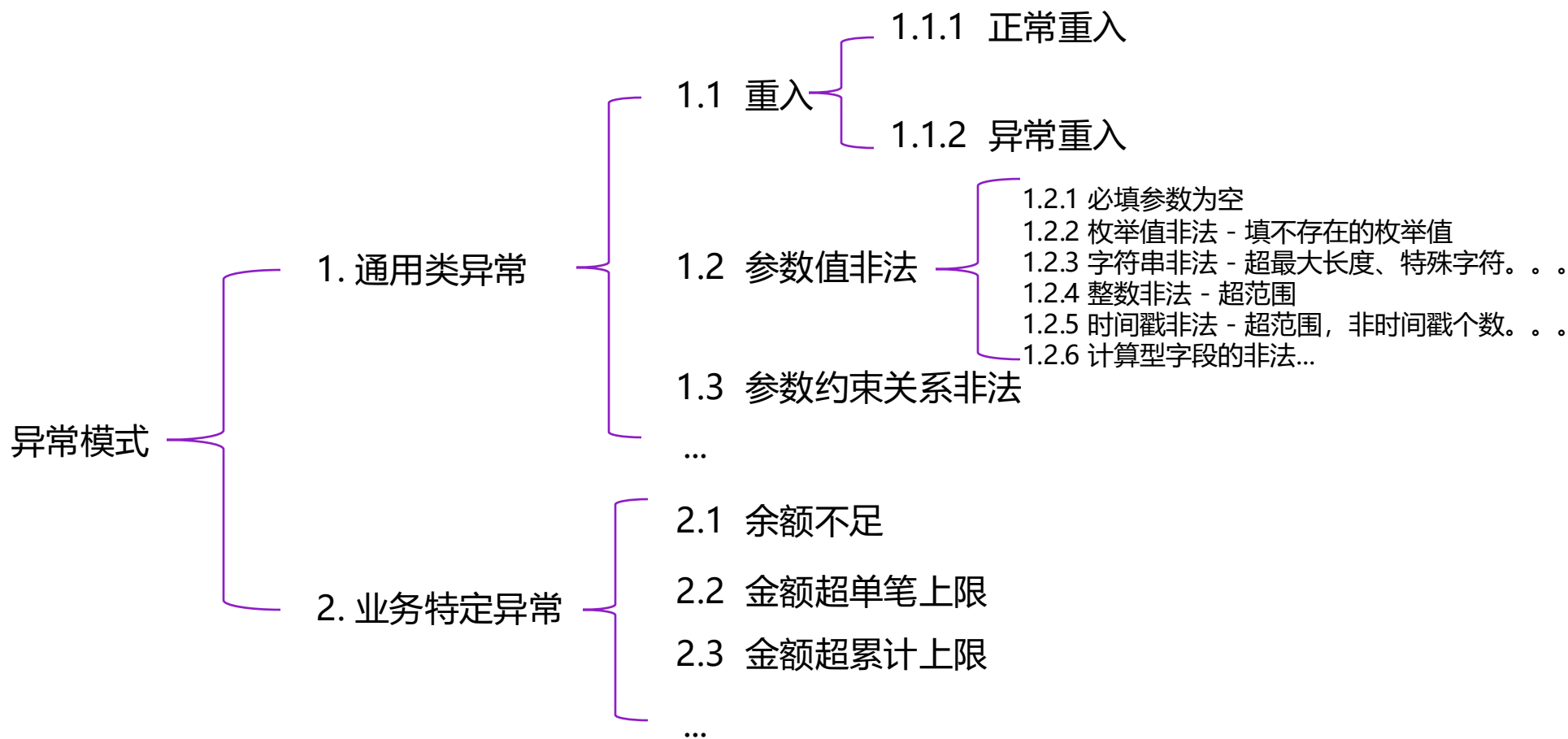
资金安全的朴素定义：任何bug（业务/技术）导致的资金损失（长款、短款）

2. 资金安全不是光靠测试解决，而是贯穿研发的整个生命周期

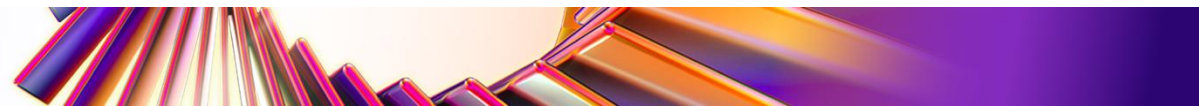


3. 资金安全军规来自长期的工程经验沉淀，是一个长长的Check List，把这个List里面，能够用测试用例来检验的，逐个拿出来变成一种异常模式，自动化





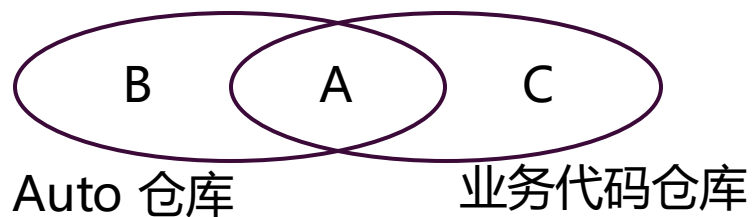
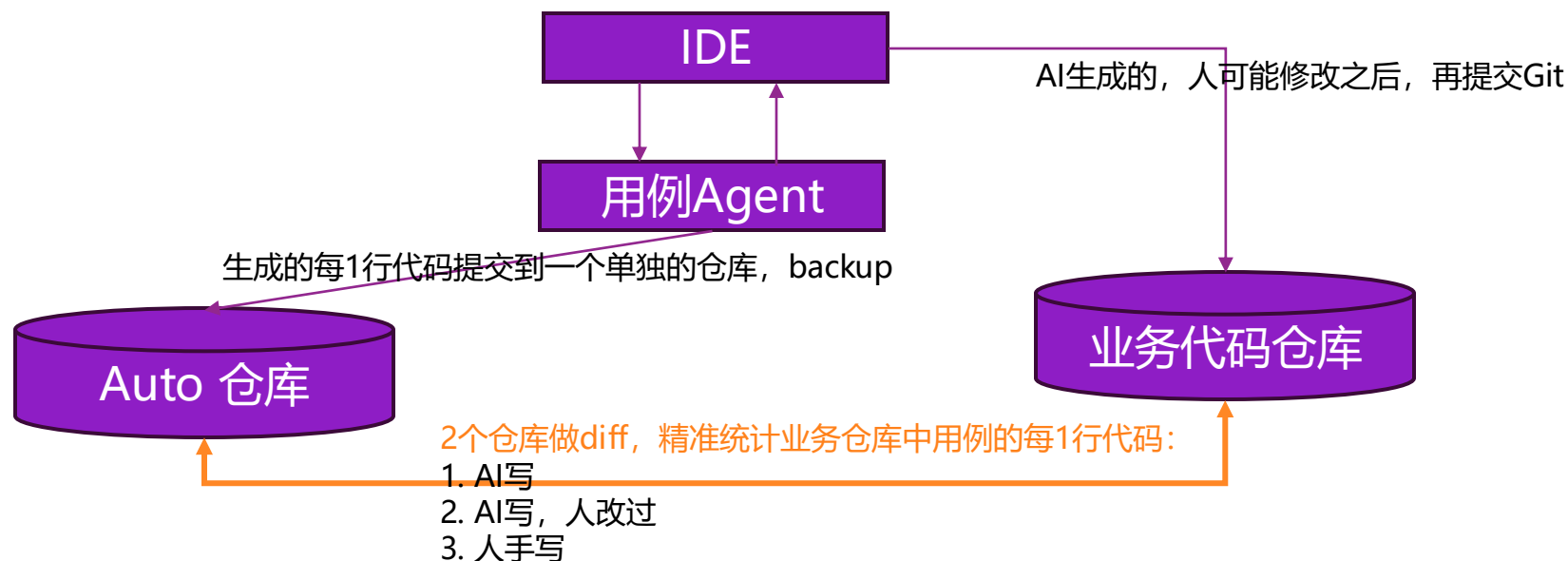
从通用到特定，从简单到复杂，一类类拆解，逐个自动化



# PART 06

## 总结与下一步计划

# 最终效果：AI代码采纳率(by行) 达到90%以上



$$\text{AI代码采纳率} = A / (A + B)$$



- 想清楚效果衡量指标，如何证明做的好，如何精确度量做的“准”、做的“全”？  
AI采纳率？  
接口测试用例，多少算“全”，质量标准是什么？
- 数据质量不好（接口契约、需求文档、技术文档），AI再强也没用。  
该人补的地方，还是得笨办法，一点点补
- 想减少幻觉，尽量做 workflow 拆解。拆解的越细，让AI执行，准确度越高
- 很多开发工作，不用AI模型，用传统算法也能做。但用AI，开发效率会极大提升



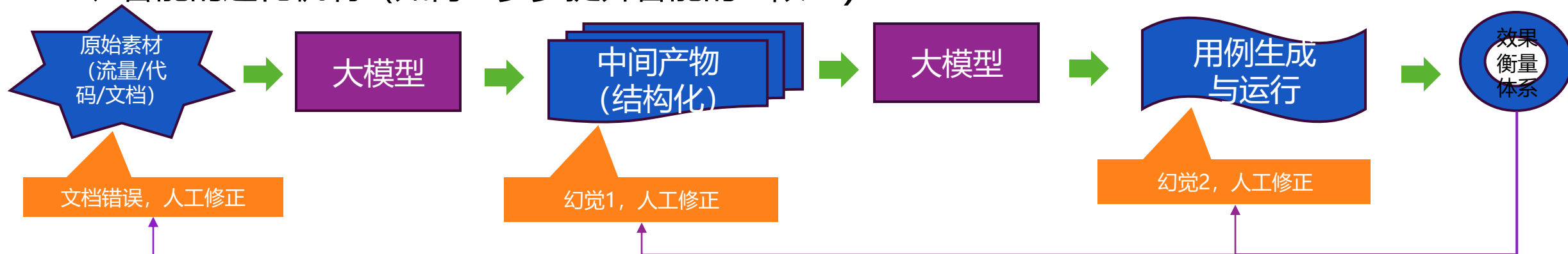
## ▶ 下一步计划

### 一、DB断言/造数的智能编排

### 二、知识工程（人工沉淀 + 研发规范 + AI知识抽取）

- (1) 数据字典/数据治理：同1个字段会出现在不同接口参数、DB表、造数中，名字要一样
- (2) 接口参数之间的约束关系
- (3) 取数标签之间的约束关系
- (4) 算子库
- (5) DB字段的状态机

### 三、智能的进化机制（如何一步步提升智能的上限？）



- (1) 大模型的幻觉无法彻底消除，所以一定有人工修正环节。关键是要把这种修正，沉淀下来，并且能持续优化模型生成效果
- (2) 效果衡量体系很重要：效果不好，是幻觉导致、还是数据质量差、还是上下文超长，如何快速找到原因并持续进化



# 科技生态圈峰会 + 深度研习

——1000+ 技术团队的选择



NiDD 峰会

NiDD 峰会 上海站

AI+研发数字峰会

时间: 2026.05.22-23

NiDD 峰会 北京站

AI+研发数字峰会

时间: 2026.08.21-22

NiDD 峰会 深圳站

AI+研发数字峰会

时间: 2026.11.20-21



AiDD峰会详情

NiDD × K AI+产品峰会

NiDD × K 上海站

AI+产品创新峰会

时间: 2026.01.16-17

NiDD × K 北京站

AI+产品创新峰会

时间: 2026.08.23-24



产品峰会详情



# 2026 AI+ PRODUCT INNOVATION SUMMIT

01.16-17 · ShangHai

## AI+产品创新峰会



### Track 1: AI 产品战略与创新设计

从0到1的AI原生产品构建

论坛1: AI时代的用户洞察与需求发现

论坛2: AI原生产品战略与商业模式重构

论坛3: Agentic AI产品创新与交互设计

### 2-hour Speech: 回归本质



用户洞察的第一性

——2小时思维与方法论工作坊

在数字爆炸、AI迅速发展的时代，  
仍然考验“看见”的“同理心”



### Track 2: AI 产品开发与工程实践

从1到10的工程化落地实践

论坛1: 面向Agent智能体的产品开发

论坛2: 具身智能与AI硬件产品

论坛3: AI产品出海与本地化开发

### Panel 1: 出海前瞻



“出海避坑地图”圆桌对话

——不止于翻译:

AI时代的出海新范式



### Track 3: AI 产品运营与智能演化

从10到100的AI产品运营

论坛1: AI赋能产品运营与增长黑客

论坛2: AI产品的数据飞轮与智能演化

论坛3: 行业爆款AI产品案例拆解

### Panel 2: 失败复盘



为什么很多AI产品“叫好不叫座”?

——从伪需求到真价值:

AI产品商业化落地的关键挑战

智能重构产品 数据驱动增长  
Reinventing Products with Intelligence, Driven by Data

80+

业界大咖

汇聚产品大咖的真知灼见

40+

创新案例

开拓全新AI+产品视角

10+

分论坛

涵盖产品到商业增长的全链路

800+

听众规模

共同探索产品的智能重构



扫码查看会议详情



第8届 AI+ 研发数字峰会  
AI+ Development Digital Summit

# 感谢聆听!

扫码领取会议PPT资料

