郭又宁 Almee Kuo

以系统化抽象力，
驱动业务规模化增长与产研效能闭环

资深体验架构师 | 系统型产品设计

SaaS　企业级

jzalmee@gmail.com



关于郭又宁



> 真正的平台能力，不是解决一个问题，
> 而是让同类问题，从此不再需要重新解决

从化学工程跨界体验设计，这段经历让我养成一个习惯：先拆解问题的结构，再进入设计。

我参与并主导了任务自动化平台、资源调度系统可扩展框架、多场景 CRM 工作台、B 端设计系统治理等项目，形成一套可复用的设计路径：**识别问题本质 → 抽象可复用结构 → 让系统持续演进**

复杂 B 端专精　0-1 平台搭建　设计系统治理与规模化　用户研究与画像抽象　跨团队机制建设

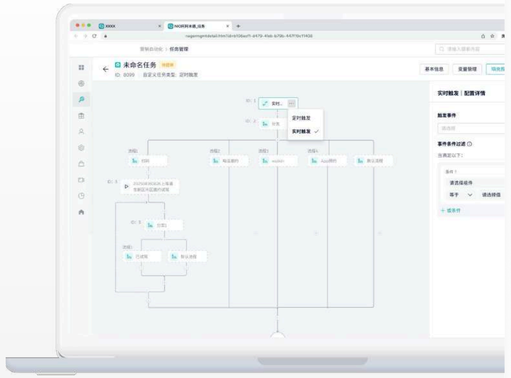
我设计的不是单一功能/页面，而是复杂系统如何被人理解、运转与规模化

1

任务自动化平台
重构

以「限制自由度」换取「确定性」
提升配置成功率

主设计师 | 模块化 场景抽象

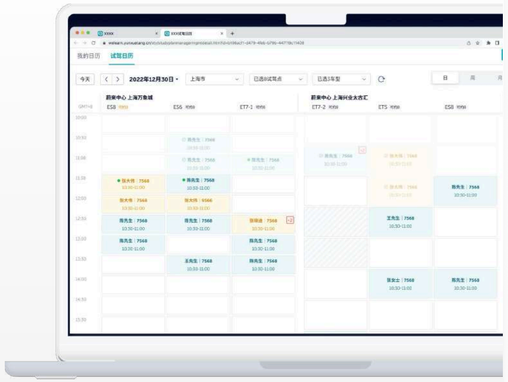


2

资源调度
可扩展框架

真正的平台能力，
让同类问题不再重复解决

设计组长 | 0 → 1 平台化 场景抽象

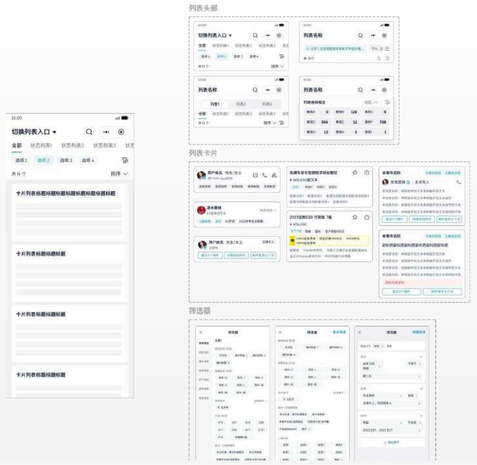


3

B端移动
设计系统治理

真正的设计治理，
是让规范可以自己运转

设计组长 | 规范化 跨产设研职能协作

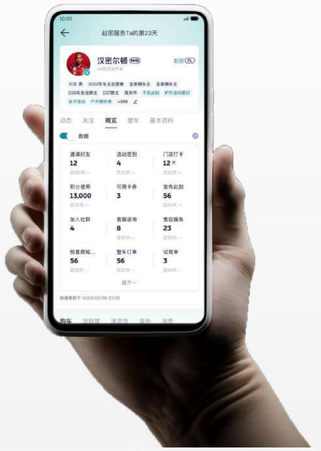


4

多场景CRM
用户工作台

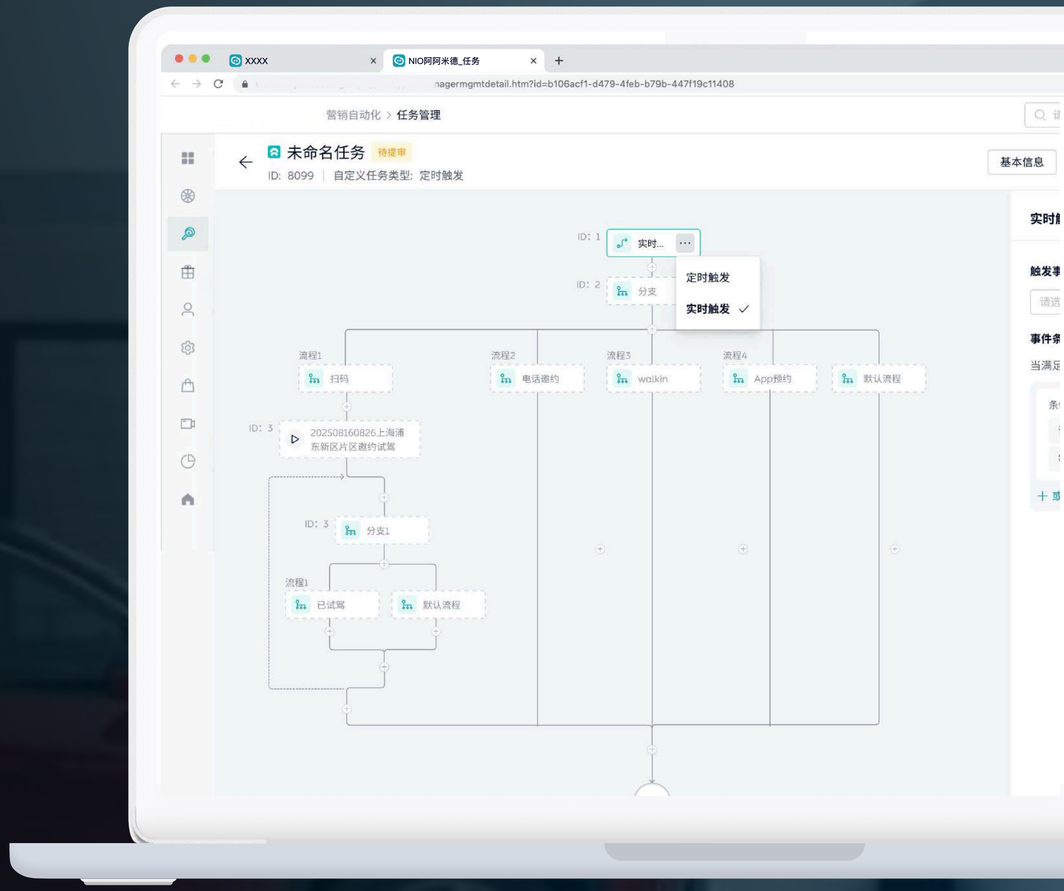
让售前售后运营从「靠经验判断」
进化成「靠系统感知」

设计组长 | 信息数据重构 用户测试



任务自动化平台重构

让工作流搭建，
从高度依赖经验进化成可扩展的系统能力



2周 → 3天
上线效率

-30%
新人上手时间

1 → 3个
支援品牌

60%
场景覆盖率

赋能场景
3品牌App、B端、企微

项目时长
3次双周迭代

主设计师
PM*2、RD*5

工作流搭建画布的体验问题：过度自由却不易操作及高报错率高

为什么，不局部优化
Problem Statement

背景
问题

蔚来B端平台承接3个品牌、跨企微／App／Web的营销任务协同，业务规模每季扩张。

平台越灵活，系统越脆弱

关键
判断

当「用户、产研、协作」三个视角问题，指向同个根源系统 缺乏结构性约束，我知道这 不能仅「局部优化」解决

以下三个视角印证这一判断：

	表象	核心原因
用户	操作依赖经验，效率随业务增长递减	**系统没有给用户「正确路径」的引导**
产研	框架缺乏约束，功能堆砌引发系统脆弱	**开发在没有边界的画布里无法建立结构**
协作	多方需求拉锯，高压复杂风险3重困境	**没有人知道「谁的判断是基准」**

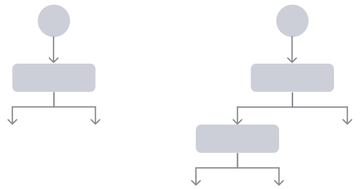
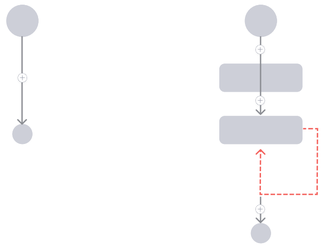
限制自由度，换取成功率

当平台越灵活，
系统反而越脆弱

维护成本暴增、新人学习成本高、
操作成功率下降、跨团队无法协作

我们评估过保留开放式并优化的方案，但因为
无法拓展「循环式流程能力」，所以放弃了

我选择限制自由
来换取成功率与扩展性

分类	开放式画布 | 原版	封闭式画布 | 新版
抽象		
操作方式	重拖拽，手动对齐反覆	**重点击，精准有度**
流程查看	控件摆放混乱，需「阅读」	**结构清晰，能快速「浏览」**
开发成本	渲染动效要求高，还原度低	**控件类型明确，还原度高**
擴展性	控件连接方式单一无法拓展	**连接方式有度易管理**

3个设计决策，让工作流搭建，从「依赖经验者」变成「依赖系统」

关键决策
Key Process

STEP 1：依赖「工作流 → 系统流」抽象

抽象业务模型，构建触发原因与对象

针对管理职「操作依赖经验、效率随业务增长递减」问题，
抽象 4大业务线共性流程，提炼3类触发模式，
将「隐性经验」转化为「可复用的系统路径」。

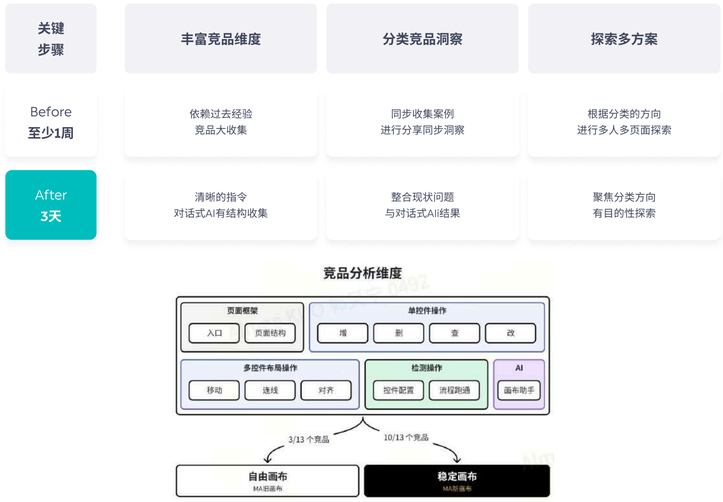


STEP 2：依赖「交互特征」抽象

对话式 AI ，提升5倍交互决策效率

用对话式AI系统性拆解 13个 竞品的画布结构与控件逻辑：

- 将原需人工翻阅 2周 的分析，压缩至 3天
- 从团队分工降至 1人 作业
- **稳定画布在「操作效率」与「还原度」上都优于自由画布**



STEP 3：依赖「操作控件」抽象

限制操作自由度，提升配置成功率

开放式画布虽灵活，但无法约束用户的操作路径，
错误只能在执行后发现。因此选择封闭式添加：每个节点强制定义起点与流向，配合空值检查，**将报错前移至配置阶段**。



佈局框架｜清晰流程順序，不中断不遮蔽當前所需信息为原则

1 提高画布可视区

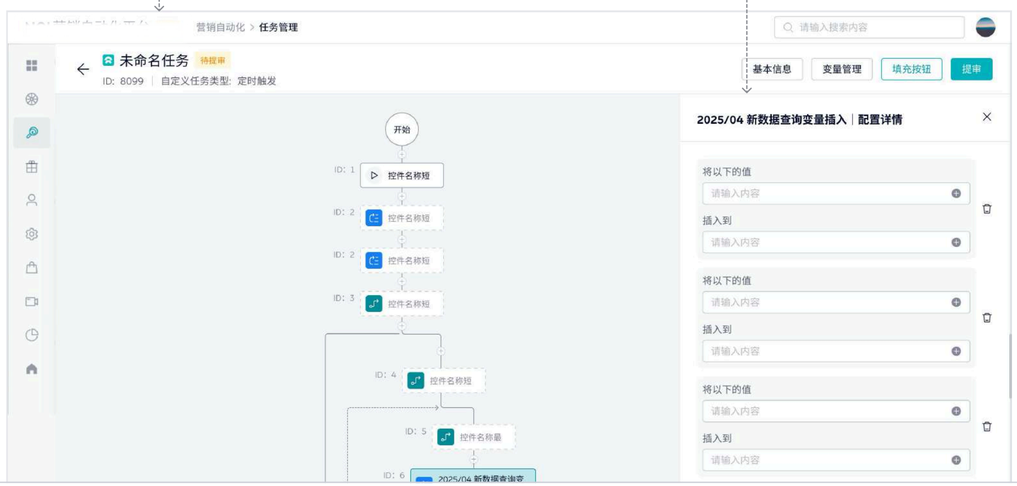
从用研识别控件配置使用频次高于添加控件，
来释放更大操作区，承载更多控件与流程配置

原来表单宽度收窄至屏幕1/3
去除断点操作的朦层

Before



After



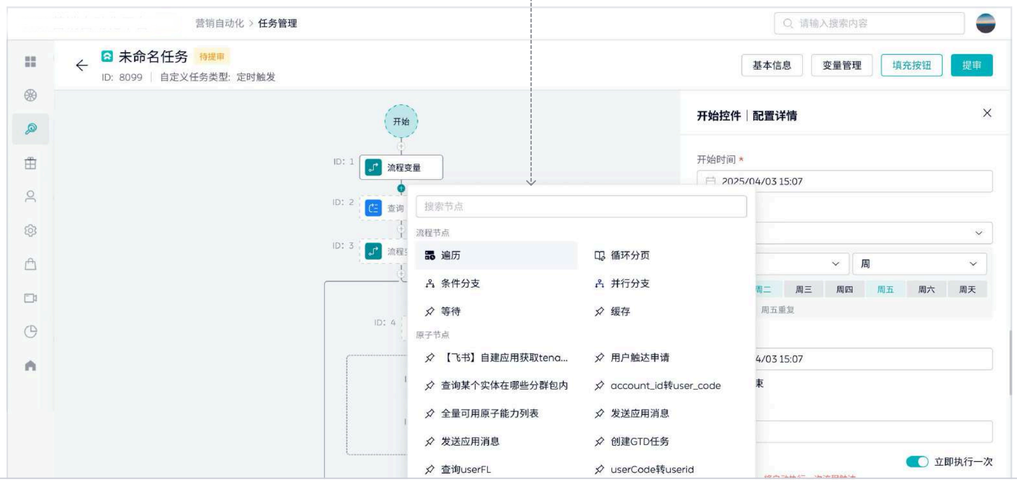
2 提高配置操作流畅度

移除拖拽交互方式，调整为点即添加，并刷新表单

Before



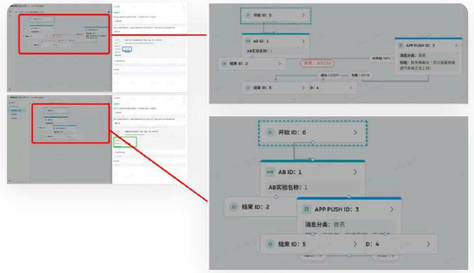
After



控件模组｜抽象逻辑，拓展有理有据

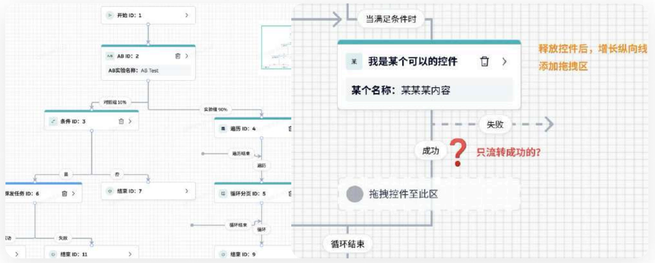
1 控件互相遮蔽问题

不易回溯整体流程配置，不易编辑调整流程



2 原交互逻辑「无法承载」新后端逻辑

循环且重复性单流程特性的控件，在原交互无法跑通





旧控件，需阅读理解

串形控件（旧）

控件上包含成功与失败连接线
（每新增一个连接就会产生一个分支，依此类推）

新控件，从「轮廓、添加控件方式」来识别

串形控件（新）

添加时按顺序上下连接，
不含连接线。
整体流程仅一条主线
适用于「顺着走」的流程

内嵌套控件

添加时不断向内堆叠，
包含循环类连接线。
保持唯一主流程
适用于「重复走」的流程

多流程控件

添加时不断向旁边扩展，
包含分支类连接线。
在唯一流程中衍生多个分支。
适用于「分岔走」的流程

任务自动化平台重构影响力

画布优化

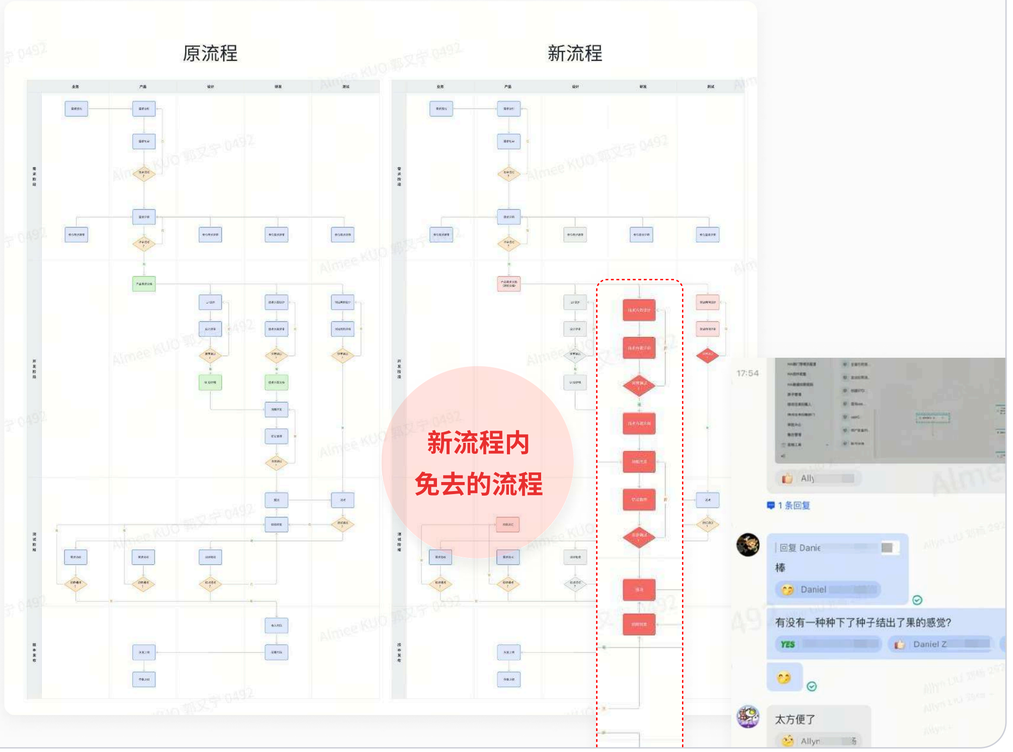
体系化效率
赋能团队
公开分享



5 倍

上线效率

原：一个业务需求，需2个迭代满足
新：一个业务需求，需0.5/1个迭代



✓ **服务业务从 2个 → 5个；1个品牌至3个品牌**

迭代后的操作逻辑、配置方式提升需求满足
新人同类产品上手时间缩短30%

✓ **控件配置化，新业务接入不需重新开发，直接复用框架即扩展**

✓ **画布体验优化**

流程配置过程，单控件减少至少4个步骤，且降低反复调整次数

✓ **3类控件交互拓展逻辑**

产品团队据业务需求，从3类里进行排列组合配置，快速上线

资源调度可扩展框架

真正的平台能力，不是解决一个问题，
而是让同类问题，从此不再需要重新解决

85 → 15 天
新场景接入效率

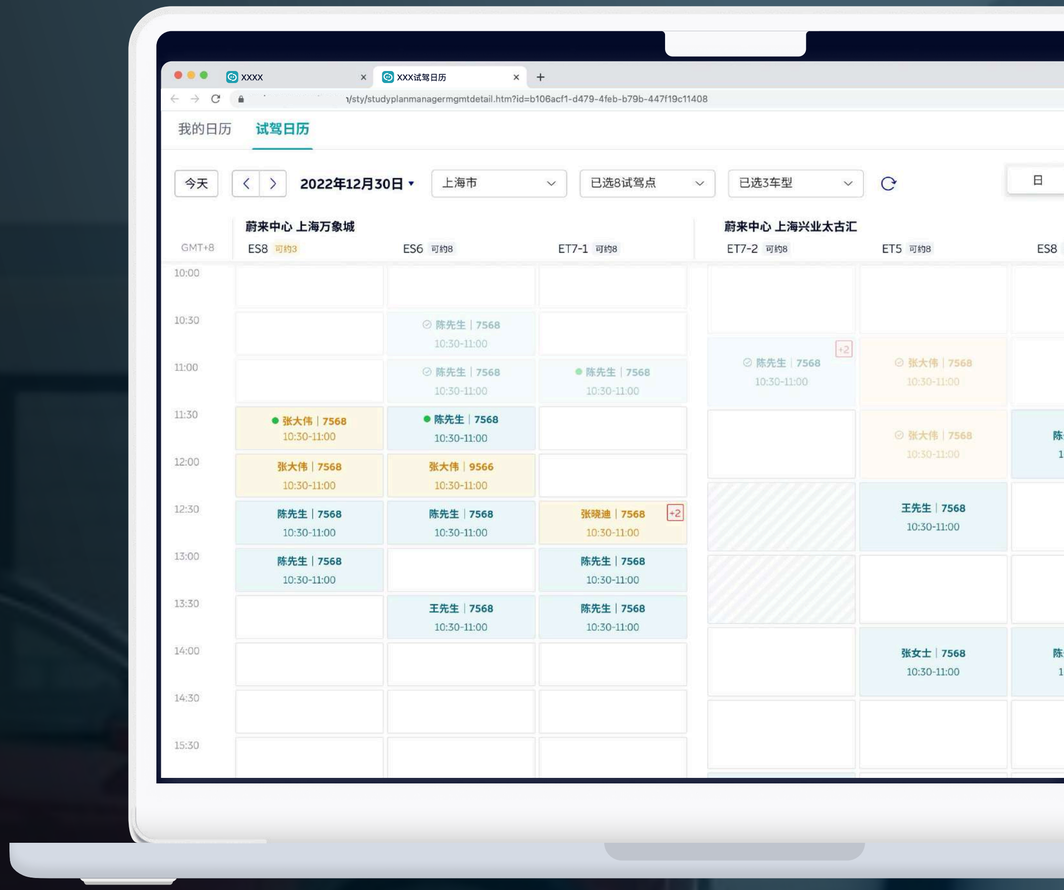
1 → 4 个
支援场景

+50 %
框架组件複用效率

赋能场景
试驾、交付、排班、维修

项目时长
搭建期8周｜8次双周迭代

主设计师
PM*1、UX*1、UI*2、RD*3



识别多场景，业务割裂与资源损耗共同问题，降低重复建设

核心原因

我们的核心问题是 资源冲突管理

如果使用现成产品（它们解决的是时间管理问题），仅解决单一试驾预约问题；其他业务需退让，或新场景重新开发。

关键判断

自研框架，我们解决的是蔚来所有 资源调度场景 的底层逻辑。
这是个关于「解决问题的范围」的设计判断，不只是技术选型。

	表象	本质
线上客服	试驾车被重复预约，用户等待时间过长	同资源在同时间被多门店挣抢
管理者	排班看不清忙闲，影响人力调度	人力资源可视化与调配没有统一语言
交付专员	交付时间撞单，人员支援不及时	时间、空间、人、车，4个维度没有共同框架

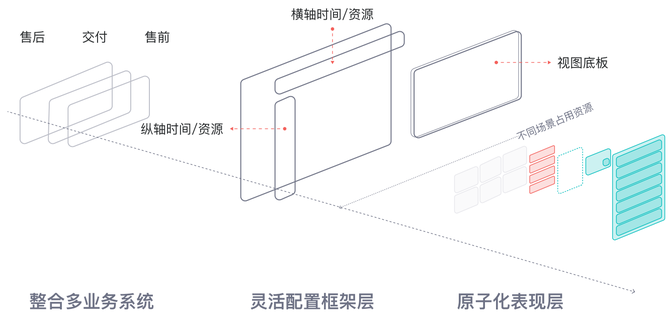
提炼底层资源公式，原子化结构支撑业务扩张

关键过程
Key Process

STEP 1：原子抽象

抽象问题，4 个场景，1 个本质

试驾 · 排班 · 维修 · 交付的底层冲突一致：
资源占用 = 时间 × 空间 × 人 × 物
这个公式是整个设计框架的起点。



STEP 2：原子堆叠

3 层架构设计，支撑 4 个业务场景

① 整理「车辆、人力、时段」，资源为系统可识别的原子单位

② 灵活配置框架
横轴时间 × 纵轴资源，两轴独立配置

③ 原子化设计原则
先综述后细节、 材质颜色区分忙闲、角标显示总量、
不堆叠



STEP 3：堆叠规则

主动解构视图逻辑，消除协作摩擦

将 UX 层级拆解为开发可直接对照的组件结构：
底板 → 视图类型 → 筛选器 → 单元格 → 事件卡片。
每层交互关系明确定义，**还原度问题在沟通前就解决**。



适用于任何高复杂度B端系统 4 步骤

「 先厘清人与场景，再定义工具边界，最后才进入设计。」

先定义	识别	建立	最后才是：交互与UI
真正要优化的是什么	哪些是高频关键链路	可扩展但有限度的框架	打磨体验细节 + 呈现可交付的方案

项目拆解
时间规划

从1v1访谈
提炼场景痛点

提炼设计原则
激发可能性

汇报立项
产品业务团队

拆解颗粒度
获取开发资源

赋能多场景
维修、试驾、排班、交付

精准效率迭代
场景细化、屏效优化



可套用至：0→1 项目，为项目落地指引；1→1+项目，为缺漏补查清单

资源调度可扩展框架影响力

框架可复用性被真实业务验证 85天→15天，它说明：

框架的抽象层级是对的
足够稳定可以被复用
足够灵活可以被扩展

这是平台型设计最难做到的事。

✓ **日历框架提炼与落地规范**

App/Web端日历规范化，赋能产品经理与开发落地

✓ **4大场景落地** （App/Web端）

试驾日历：提高试驾车使用率
排班日历：效率调配跨门店人员支援
交付日历：调配交付量与调配人员
车模日历：调配单量与技师人员

视图原子化

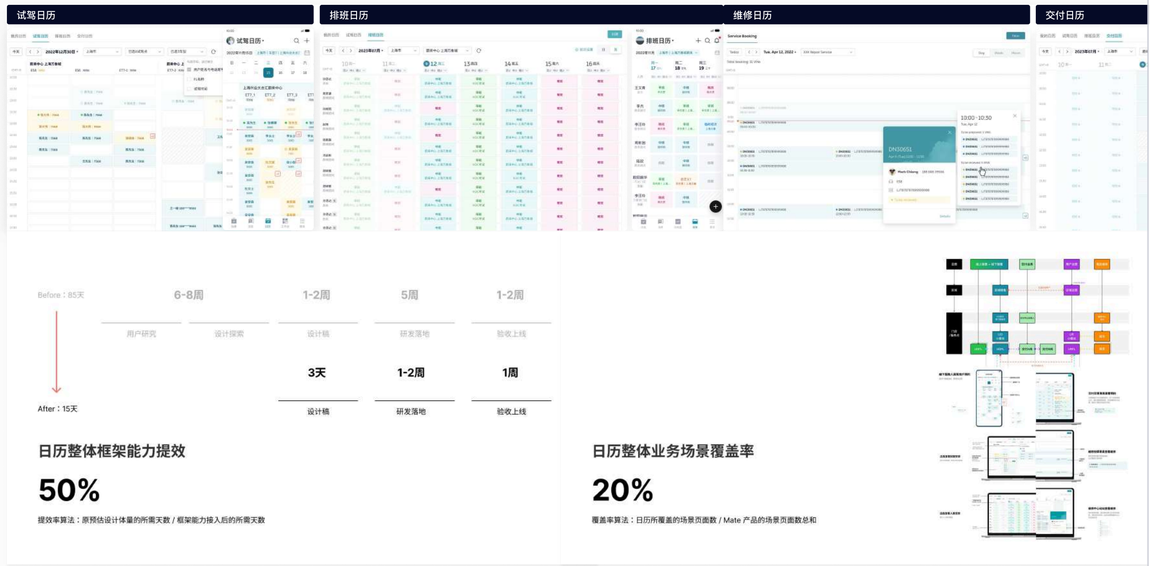
推进产品化
推进开发
赋能业务



85→15 天

日历框架原子化，
新场景接入从平均85天缩短至15天

从单一试驾场景，扩展至排班、维修、交付，覆盖率提升说明框架的可复用性被验证了。



B端移动设计系统治理

最贵的解决方案不一定是最好的
真正的治理，是让规范可以自己运转

3.4 → 4.0
用户满意度

-30 %
设计/开发工时

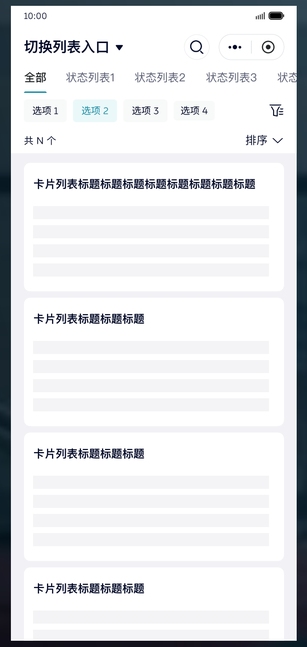
100 人
跨团队推动

15 套
沉淀场景模版

赋能场景
车企售前转化售后全场景

项目时长
6周重构 | 12次双周迭代

主设计师
UX*1、UI*2



列表头部



列表卡片



筛选器



组件库完成度 90%，体验还是割裂

关键判断

设计系统存在，不代表它在被使用

蔚来有设计系统、组件库、Figma文件，原因不是设计系统做得不好
而是：组件是原子，但业务需要的是分子

核心原因

设计系统提供了原子，却 没有告诉人怎么组合

以下分析来印证此观点

失效原因	本质
组件存在 ≠ 团队会使用	设计师面对业务场景，不知道该用哪个组件
缺少业务语义	原子组件没有场景化说明，设计师只能靠猜
缺少场景化指引	同一个操作目标，4条业务线各自有一套做法
PM/RD无法理解	规范文件只有设计师看得懂，产研无法自行对齐

不再 0 → 1，打破组件库「建设容易维护难」窘境

当时团队

第一反应是要**建立「业务自己的设计系统」**

关键判断

重构解决的是系统问题，但 失效的根源是使用问题

我决定：在原有系统上叠加业务应用层

以下三个视角印证这一判断：

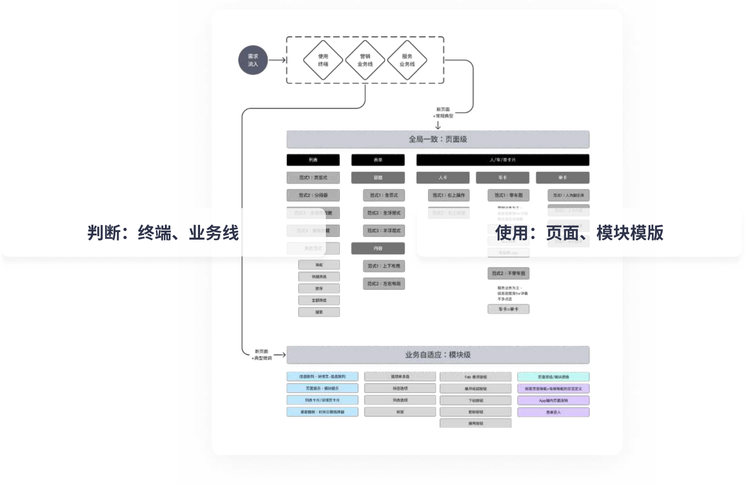
	重構	疊加业务应用层
产研	能解决当前的设计系统过于灵活问题	**在原有基础上，说明到位**
风险	全部推倒重来，周期半年起步 业务线开发需要同步配合迁移，无法正常交付	**不影响现状开发，计划有度的迭代上线**
收益	三个月后，可能会回到原点	**从 MVP 扩大规模，拉通并优化使用方式**

推动 100人 改变工作习惯，比设计本身难 10倍。我做了 3件事：

STEP 1：建立判断逻辑与机制

避免全面重构，典型页面模块抽象

全面对齐成本极高，优先切入影响面最大的地方：
先排查「高频页面」与「跨业务线重复出现」的标准操作流程
确立一致性基线后，再按 交互→视觉 的顺序向外扩展，
每个阶段都有可验收的交付边界。

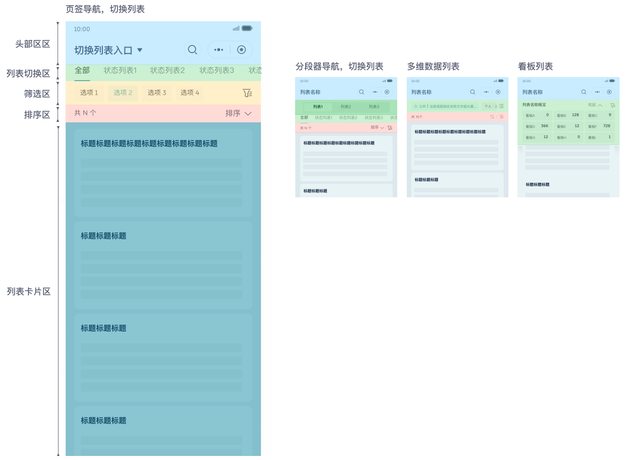


STEP 2：建立组合规则

降低导入成本，规范「自带说明书」

加入**「模组封装层」**后，设计师不需要理解底层逻辑，
照路径走就能做出一致结果。

每个模组容器定义了完整使用路径：**入口 → 接续操作 → 可替换内容 → 回显**，解决了原系统「先懂逻辑才能用」的门槛问题。



STEP 3：大规模覆盖实际工作场景

组织 Design All Hands 提高接受度

规范治理的终点是：没有设计师说明下也能运转。
通过跨业务线 **Design All Hands**，
用**「典型页面 + 常用模组范式」**替代抽象规则，
通过设计稿（设计师）、文档（非设计师）的方式传递，
让规范从被动接收变为主动共识。



B端移动设计系统治理影响力

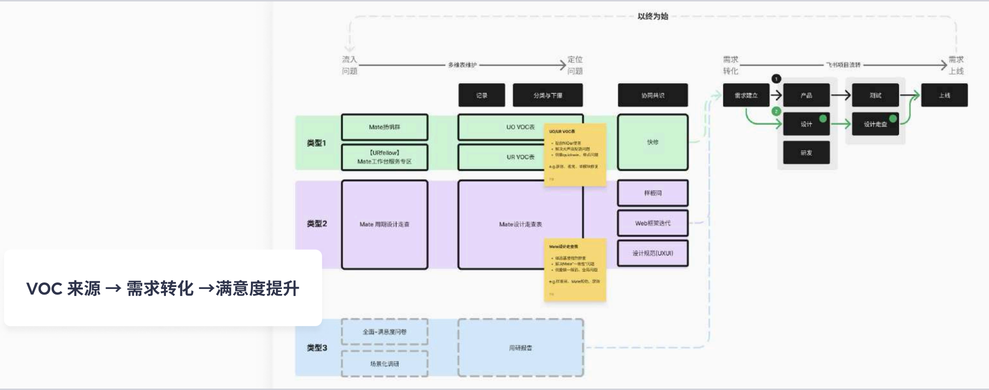
✓ **这个治理框架，3个层面闭环：**
用户体验对齐、产研效能提升、设计资产沉淀

这是设计系统治理真正的价值，不只是规范文件；
最终输出PM / RD可直接执行的多品牌、App、Web规范细则，小型需求设计师只需走查验收，不必全程介入。

用户满意度数据治理

体验闭环

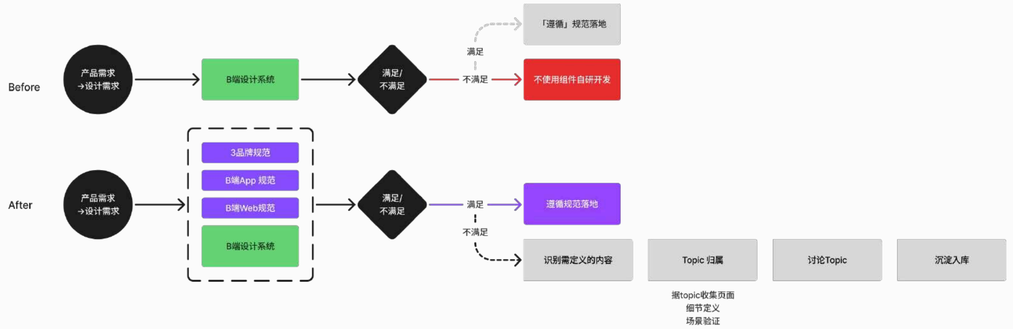
下钻提炼Voc
至逐条体验优化
用户满意度 3.4 → 4.0



协作工作流治理

效能闭环

落实规范使用
共识组件模块
工时 −30%



设计系统治理

资产闭环

品牌规范应用
App/Web端模块组件拓展
沉淀 15+ 套场景模版



多场景
CRM用户工作台

让售前售后运营从「靠经验判断」
进化成「靠系统感知」

+70%
全覆盖场景增长

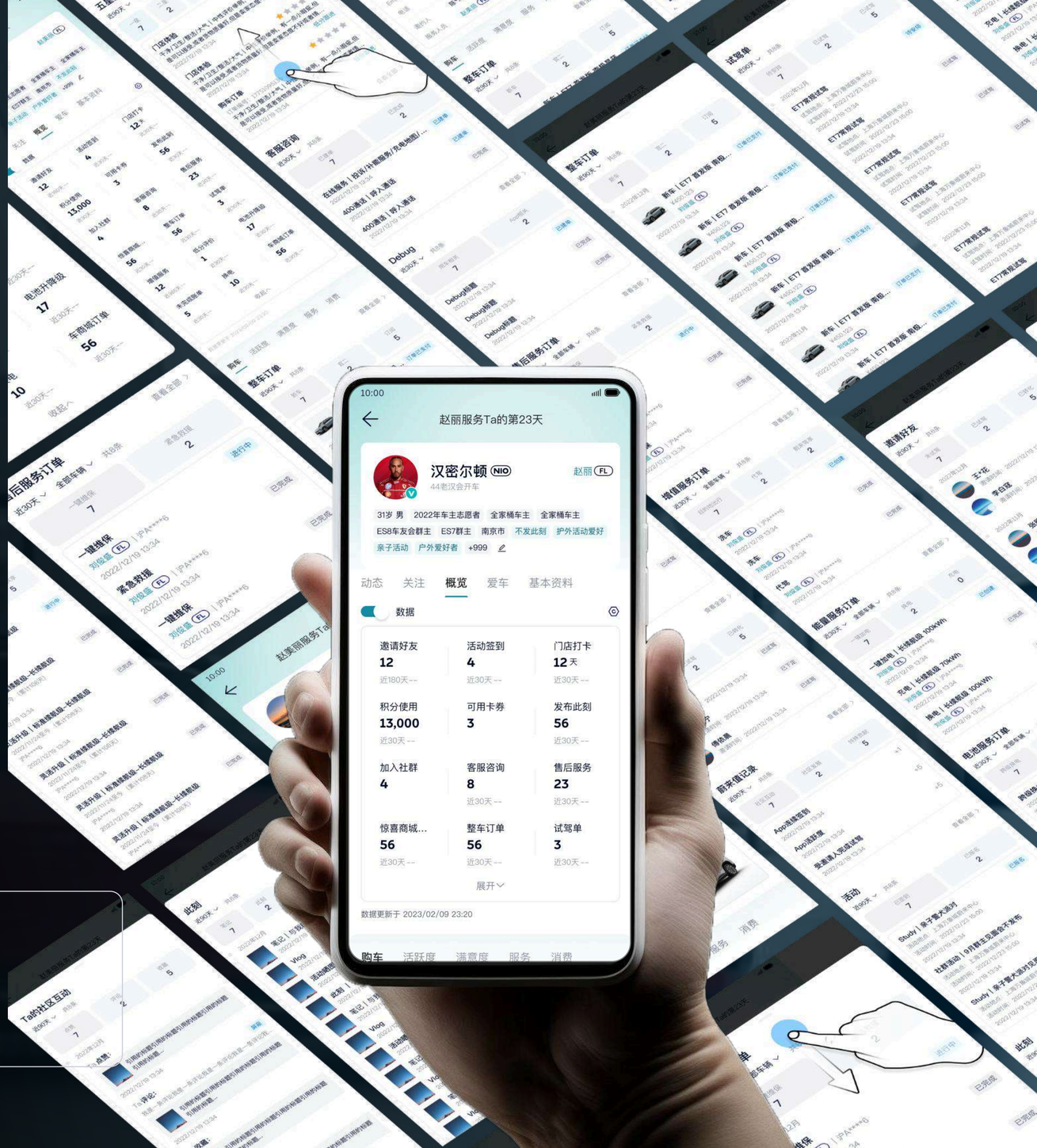
+40%
数据可视化占比提升

200%
迭代效率提升

赋能场景
满意度跟进、活跃度运营、售后服务管理

项目时长
6周重构
12次双周迭代

主设计师
PM*1、UX*1、
UI*2、RD*3



增量的平铺信息，导致查找信息低效

为什么，高密度服务信息无法快速决策
Problem Statement

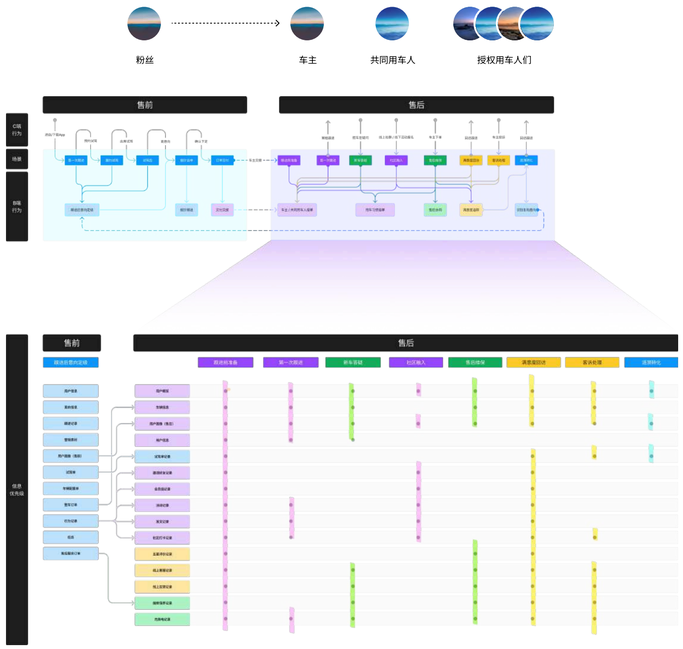
背景
问题

蔚来售后服务管家同时跟进多位车主，信息分散在不同入口，
服务动作靠人工记忆和经验判断，效率随用户量增长急速下降。

关键
判断

当三个视角的问题都指向同一个根源 系统没有帮员工「看清用户状态」，我知道这不能只靠「加字段」解决

	表象	本质
服务者	操作依赖经验，跟进靠记忆	系统没有帮员工识别「用户状态」
产研	字段平铺，信息拓展性差	字段平铺结构，无法支撑售后非线性场景
协作	多人服务同一用户，信息不同步	没有统一的服务状态语言



信息降噪与场景化提效：从「搜索信息」到「数据驱动决策」的转变

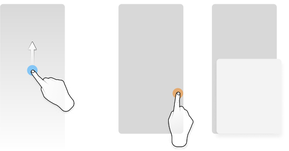
售前是线性的；售后是动态、多线程的。

原有平铺结构失效

补字段无法解决：
多场景并行/动态优先级
所以放弃了。

我选择重构框架

分类	解决的问题
信息层	什么值得被注意
结构层	什么需要优先判断
交互层	如何降低扫描成本
表现层	如何提升理解速度

分类	原版	新版
页面结构	 长轴下滑	 长轴下滑 + 浮层结构
信息获取	信息堆叠，翻找式，效率低	概览优先，按需下钻
可扩展性	字段堆叠，难以拓展	模块化，场景可独立迭代
状态感知	无，靠人工判断	系统自动推断运营优先级

3个设计决策，让服务效率从「依赖记忆」变成「依赖系统」

关键过程
Key Process

STEP 1：前置，高优信息

重构框架，匹配非线性服务场景

原平铺框架无法支撑多场景并行，因此重构为**双层**结构：首层**快速感知用户状态**，二层据**场景深入明细**，让员工的**注意力跟着场景走**，而不是跟着字段找。



STEP 2：高亮，重点信息

信息可视化，让系统替代人工判断

信息太多不是问题，看不出重点才是问题。

- 定义表现优先级：**标签（状态）> 数值 > 字段明细**
- 系统自动推断运营优先级，并适时介入

员工一眼知道该做什么，不需要自己判断。



STEP 3：下钻，详细信息

信息规范化，建立跨场景统一语言

原系统格式不一，关键信息需要复制到其他系统才能查询

- 重新定义规范：**多用标签、数值、关键词构句**，少用描述和赘词。
- 建立**看板和筛选器**结构：让购车、满意度、活跃度、服务、消费五大场景共用同一套信息语言。



多场景
CRM用户工作台
影响力

70 %

场景覆盖增长

过去用户信息多半堆积在「转化前数据」



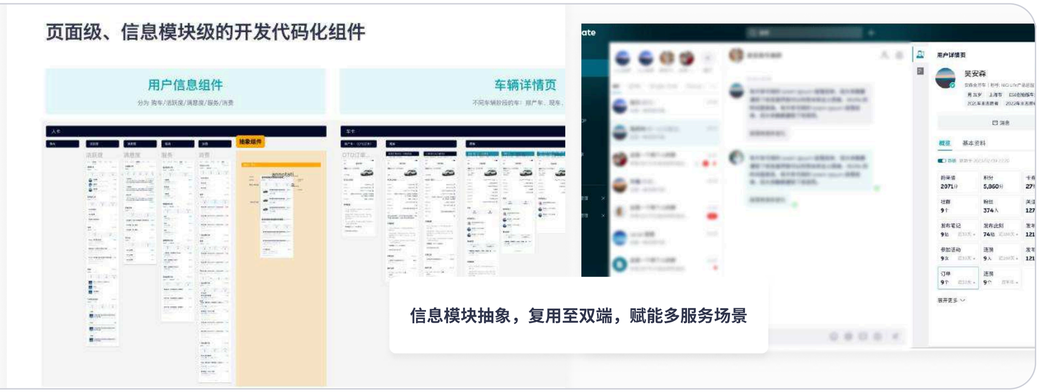
40 %

数据可视化提升



200 %

迭代效率提升



✓ **提炼服务小管家用户画像**

丰富并迭代NIO B端体验模型

✓ **用户详情页 App/Web双端落地**

提高用户服务效率

✓ **沉淀用户信息组件、车辆信息组件**

场景覆盖购车、满意度、活跃度、服务、消费

✓ **设计特征被其他业务复用**

总分组提升信息传递效率

我设计的不是单一功能/页面，而是复杂系统如何被人理解、运转与规模化

设计决策思路 → 落地路径

设计方法

1

识别问题本质，不是优化页面，

而是判断：

是流程问题？信息问题？系统问题？协作问题？

2

抽象可复用结构避免解决单点需求，而是建立：

场景模型、信息层级、组件能力、平台规则

3

让系统可持续演进，最终目标不是设计完成，而是：

可扩展、可协作、可运营、可持续迭代

核心能力

平台化

建立统一设计框架、降低跨业务重复建设、提升系统一致性

场景抽象

从单一需求抽象为通用模型、识别高频核心路径、支撑多场景扩展

信息重构

重组复杂信息优先级、建立动态信息结构、提升运营判断效率

组件化

模块拆分与复用、降低设计与开发成本、支撑快速迭代

系统思维

迭代前后台协同设计、用户流 × 数据流 × 运营流整合、兼顾业务与体验

策略落地

推动复杂方案上线、跨团队协作与对齐、从概念走向规模化落地

实际案例

任务自动化平台重构

平台化 场景抽象

让工作流搭建，从高度依赖经验进化成可扩展的系统能力

资源调度可扩展框架

平台化 组件化 策略落地

真正的平台能力，不是解决一个问题，而让同类问题从此不再需要重新解决

多场景CRM用户工作台

场景抽象 信息数据重构 系统思维

让售前售后运营从「靠经验判断」进化成「靠系统感知」

B端设计系统治理

组件化 策略落地

最贵的解决方案不一定是最好的，真正的治理，是让规范可以自己运转