基于微服务架构的后台管理系统提效体系综述

1.引言

随着互联网技术的不断发展和成熟，各类线上运营的产品都依赖于强大且复杂的后台数据系统的支撑，其中包含运营数据，订单数据等各类业务数据的增删改查。后台数据管理平台作为支撑用户端产品运作的核心技术平台承担着提升效率的关键责任。传统的管理平台在面临日渐复杂的业务使用每个业务一个系统的方案存在着管理混乱、开发成本高效率低，使用者在多套后台系统间反复切换操作成本高效率的现状。为了解决以上问题，从企业级需求管理角度出发，后台管理系统需要聚合微应用并提供管理入口。目前，具备聚合微应用能力的后台管理系统实现方案主要有 3 种：1）

通过网页标签 iframe 实现，这种方式实现成本较低，但存在无法直接获取子应用路由，直接通过 URL 直接定位到子应用菜单的成本较高。2）基于 Nginx 路由实现，这种方式解决 iframe 无法通过 URL 直接定位子应用菜单的问题，但是在技术实现上，新增路由扩展需要频繁修改配置文件，操作成本较高，不易扩展。3) 基于现代前端框架 React、Vue、Angular 等实现 SPA 应用。这种方式完美解决了 iframe 方案动态 URL 的问题和 Nignx 方案中扩展路由的问题。在系统架构层面由传统的 MVC 架构升级为 MVVM 架构，实现先后端逻辑解耦，进一步提高系统的稳定性和鲁棒性。

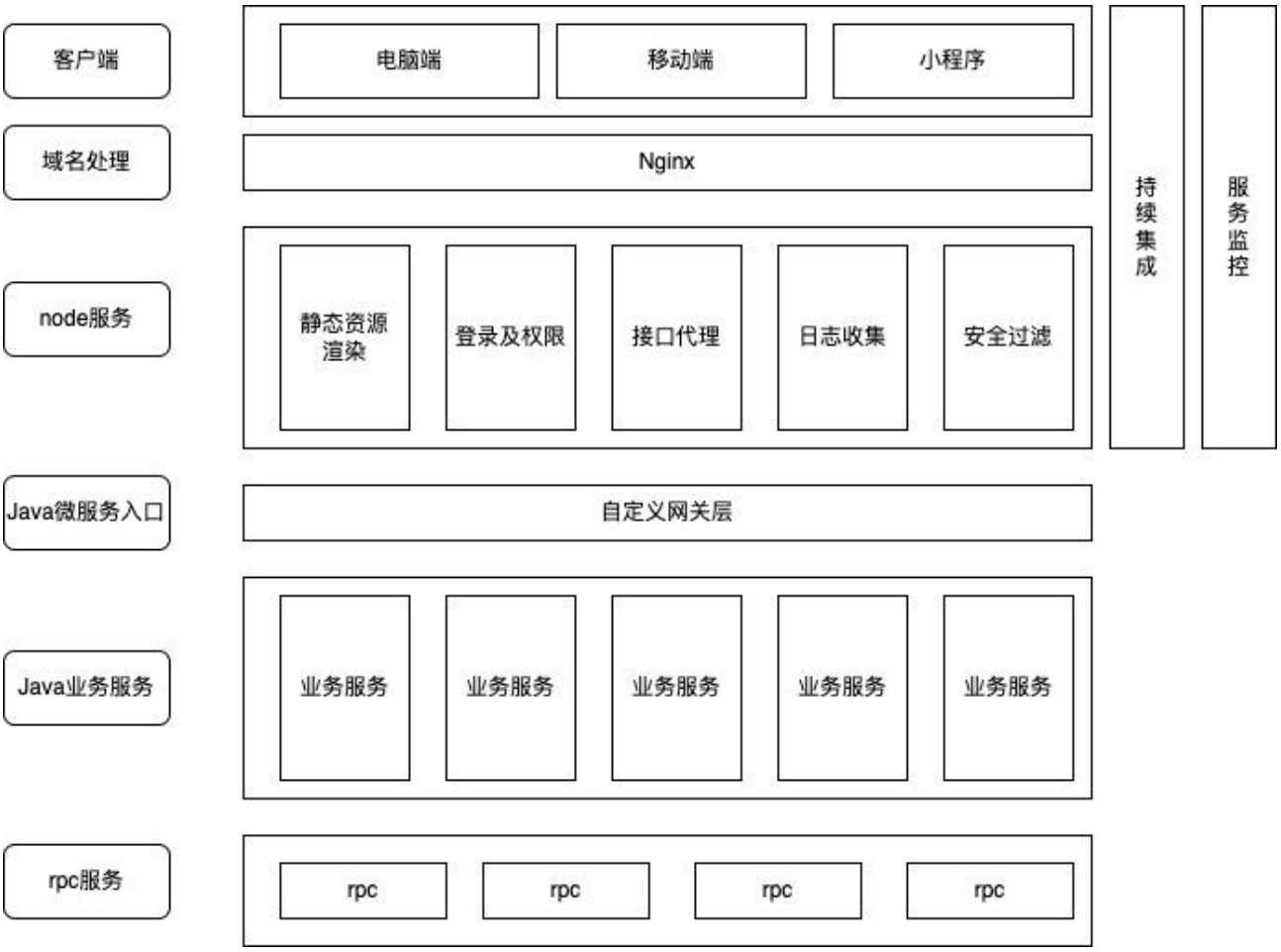
基于以上微服务管理系统实现，在后续的迭代和应用的过程中，如何能够在操作层面降低使用者的学习成本提升使用效率，在开发层面降低开发者的开发成本和系统错误是后台管理系统目前的主要目标。本文将用户需求划分为 3 类：1）常规表查询、常规表单数据插入及修改、常规表数据删除。此类需求，可以通过 no-code 平台覆盖，预置页面的交互规范，通过配置动态控制页面的操作行为，实现对数据库单表的增删改查界面化操作，操作简便，前后端数据关联，效率极高，缺点是：可扩展能力较弱，仅不支持数据库跨库查询，表单内容无法支持联动等复杂交互。2）相对个性化表查询、相对个性化表单数据插入及修改等。此类需求，可以通过 low-code 平台覆盖，通过迭代过程中积累丰富的物料库内容，通过配置对物料组件和模版进行组合拼装，能够覆盖 no-code 平台无法基于页面扩展的弊端，使交互更具有多样性，缺点是：前期需要对物料进行积累，前期开发成本较高，且无法直接操作数据库。2）完全定制化的需求需要开发人员进行编码时，可采用 pro-code 方案，利用编辑器插件提供有效高频的编写提示，提高开发效率，满足多样性需求，缺点是：开发成本较高。

综合以上的实现方案和逻辑，结合系统中微应用数量不断增加，基座与应用数据通信，微应用资源动态加载，对于后台系统的边界操作与敏捷开发的需求，本文给出以构建以 SPA 为基础的微服务后台关系系统应用，融合若干子应用系统的大型后台管理框架方案。以提升开发和使用效率，降低开发和使用成本为目的，本文提出基于微服务架构的后台管理系统的提效方案。

2.微服务架构的后台管理体系

2.1MVVM 架构

由于后端服务出于解耦业务逻辑的角度出发，采用分布式微服务部署方案，所以传统的 MVC 架构托管前端文件无法友好的支持，所以本文采用 MVVM 架构通过 nodejs 实现前后端分离。架构如下图：

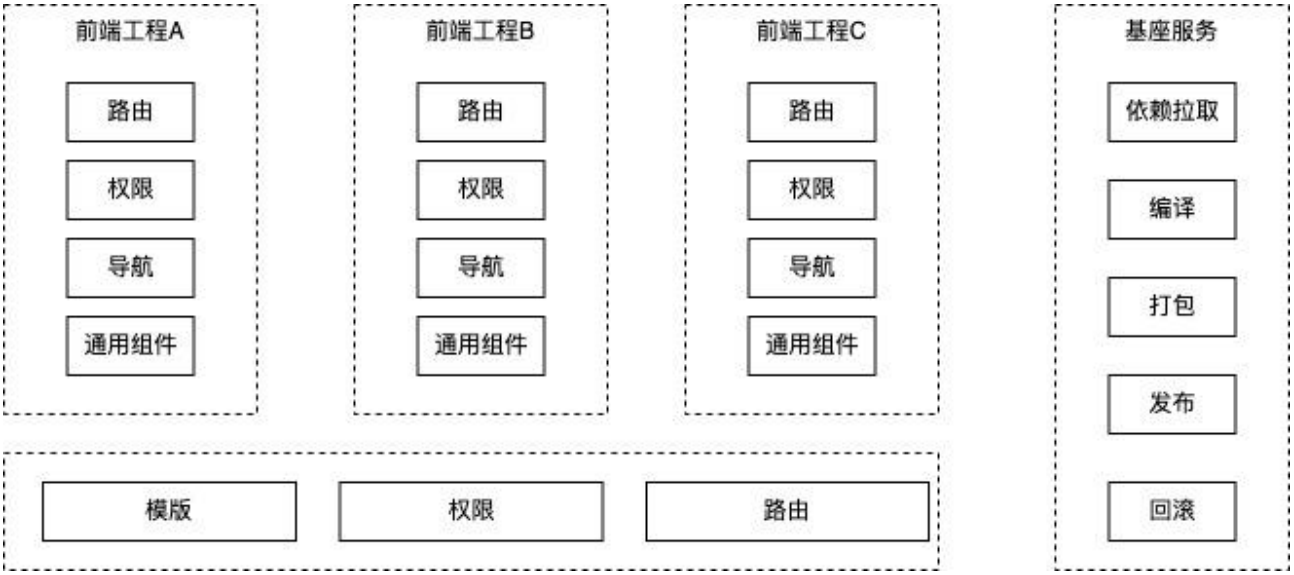


本文通过 Node 服务实现前后端分离，Node 服务职责主要静态资源托管、登录和权限的控制、接口透传、日志记录、web 安全过滤，上游对接客户端请求，下游对接业务微服务。java 微服务自定义

网关层负责通过接口路径对服务进行分发以及接口权限控制。Java 业务服务负责完成业务逻辑处理的单一职责。rpc 服务负责对数据库数据公共逻辑的加工和处理。

2.2 微前端架构

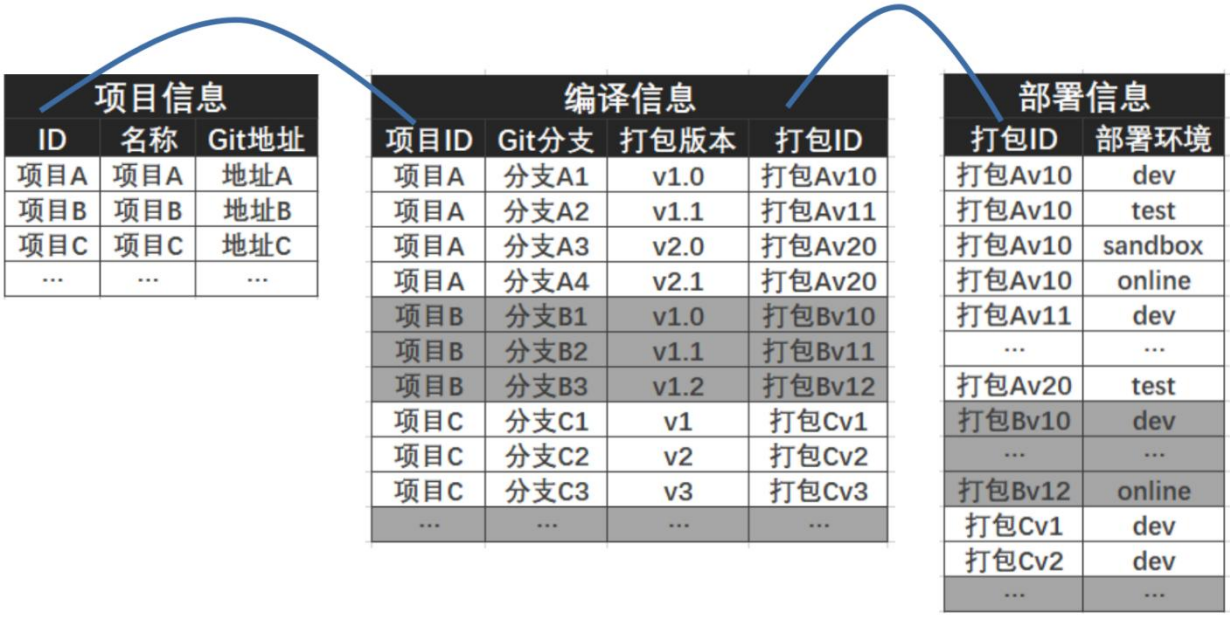
前文中的微服务架构实现了前后端逻辑解耦，前后端分别迭代上线，服务间无直接耦合关联关系。基于微服务架构，前端工程面临着独立维护多项目静态资源的挑战。如何能够完成静态资源测试及生产环境的依赖拉取、构建、部署、回滚、操作记录等成为主要问题。本文提出了以下微前端架构



：

每个前端工程的骨架结构、路由、权限组件、导航组件、公用组件是通过脚手架统一构建生成的，因此每个前端工程都具备标准目录结构，以降低开发人员对项目维护和开发的成本。下游服务对模版的维护、系统权限的控制、路由的分发处理通过 node 服务和 java 网关服务联合处理，这个 node 服务职责在：维护静态资源模版文件、控制项目页面基级别和按钮级别的权限管理、对 browserRouter 路由进行分发处理；java 网关服务职责在：对接口请求做微服务分发处理、对接口级别的权限进行控制。基座服务用来对前端工程进行统一的持续集成操作，包含拉取项目最新的依赖包，对项目在适合环境进行编译、打包、构建，生成静态资源包，静态资源包中涉及 js、css、img 文件内容直接通过 ftp 发布到 cdn 服务器，打包生成的 html 文件则需要传送到 node 服务进行统一的维护。

2.3 基座服务能力



项目信息

ID	名称	Git地址
项目A	项目A	地址A
项目B	项目B	地址B
项目C	项目C	地址C
...	...	...

编译信息

项目ID	Git分支	打包版本	打包ID
项目A	分支A1	v1.0	打包Av10
项目A	分支A2	v1.1	打包Av11
项目A	分支A3	v2.0	打包Av20
项目A	分支A4	v2.1	打包Av20
项目B	分支B1	v1.0	打包Bv10
项目B	分支B2	v1.1	打包Bv11
项目B	分支B3	v1.2	打包Bv12
项目C	分支C1	v1	打包Cv1
项目C	分支C2	v2	打包Cv2
项目C	分支C3	v3	打包Cv3
...	...	...	...

部署信息

打包ID	部署环境
打包Av10	dev
打包Av10	test
打包Av10	sandbox
打包Av10	online
打包Av11	dev
...	...
打包Av20	test
打包Bv10	dev
...	...
打包Bv12	online
打包Cv1	dev
打包Cv2	dev
...	...

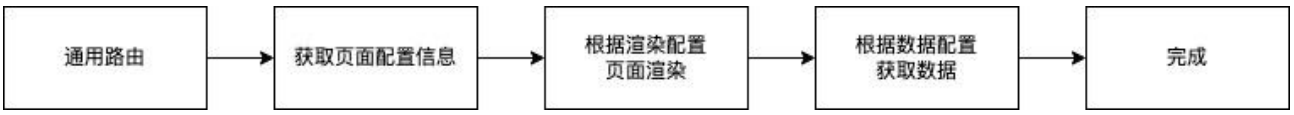
基座服务打包过程中，会面临多版本构建冲突、多部署环境等问题，此处我们采用一下解决办法：

如上如所示，我们讲编译内容拆分为三个表：项目信息表、编译信息表、部署信息表。项目信息表存储前端项目的属性信息，包含：存储项目 ID、名称、git 地址。编译信息表存储项目编译过程中需要缓存的信息，包含：存储项目信息表中关联的项目 ID、git 分支信息、包版本信息、打包 ID，部署信息表存储项目部署过程中需要缓存的信息，包含项目打包 ID、部署的环境。按照上述方式，即可以解决多前端项目打包构建多版本维护和部署多环境的需求。

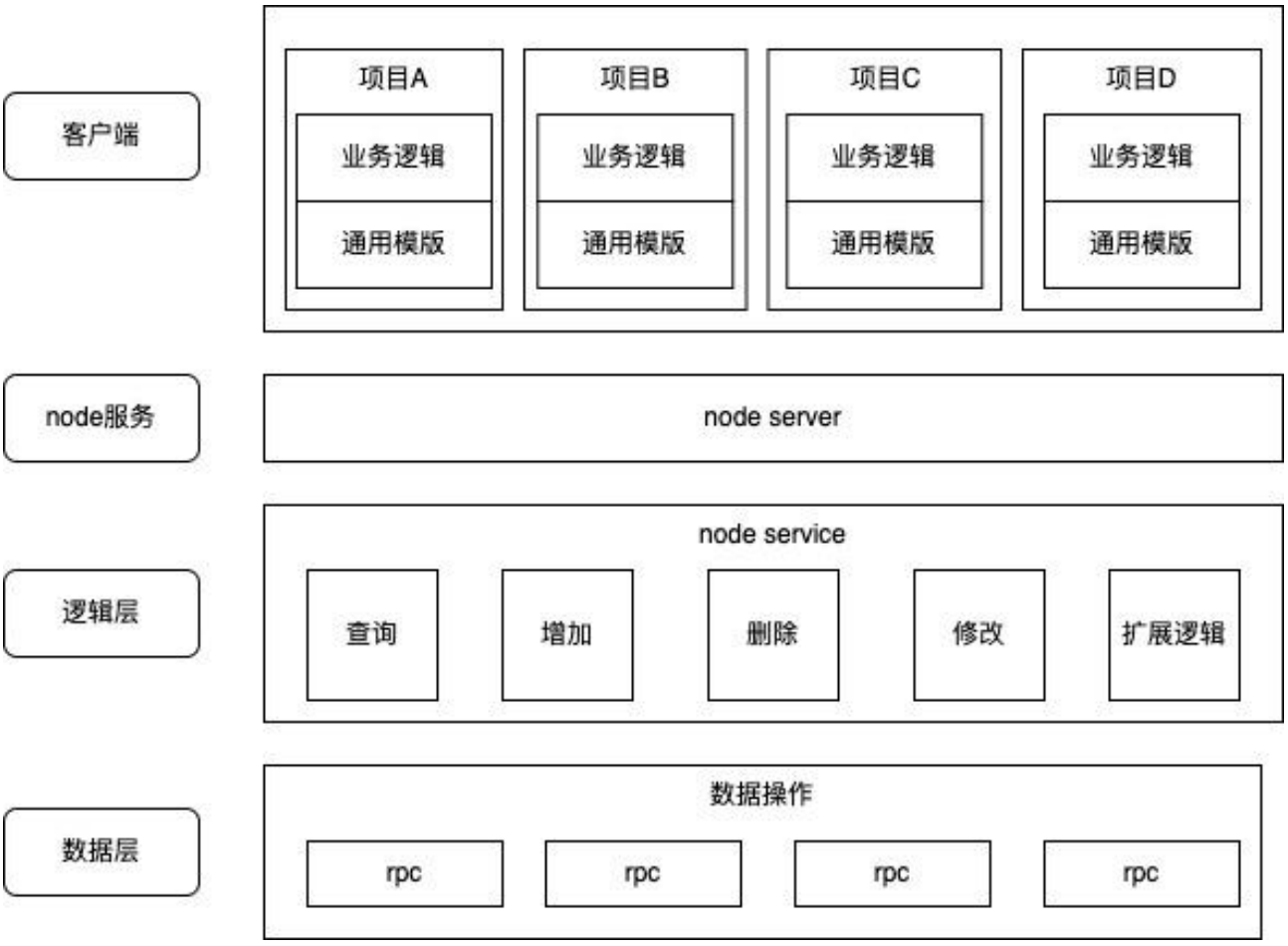
3.提效体系

3.1 no-code 提效平台

基于微服务架构的后台管理系统架构，本文提出了 100%覆盖需求的提效体系，针对基本的数据单表查询与写入操作需求，可以使用 no-code 提效平台，该平台基于用户后台操作提供基本的对数据库单表数据的查询和编辑操作界面。用户在配置列表页面时，配置信息分为两部分： 数据配置信息、页面配置信息。我们用列表页配置举例：用户需要先配置要查询的数据库表字段，根据表字段配置信息系统会生成 sql 语句对表字段进行实时查询，根据用户要查询的表字段，进一步配置页面展示和查询字段。流程如下图：



按上述操作，生成 json 数据后，提交到 no-code 体系平台数据库页面配置信息表中。no-code 提效平台架构如下图：



最上层为客户端预置页面，每个前端工程后台项目均包含通用页面模版引擎用于对 no-code 提效平台生成的配置信息进行渲染。下游服务为 node 服务，用于静态资源处理和接口请求代理。逻辑层使用 nodejs 创建 server，用于响应来自客户端的请求，对下游 rpc 服务进行查询，对返回页面配置信息数据进行拦截后进行必要的数据结构修正，使得响应给客户端的数据更加友好有利于提升客户端

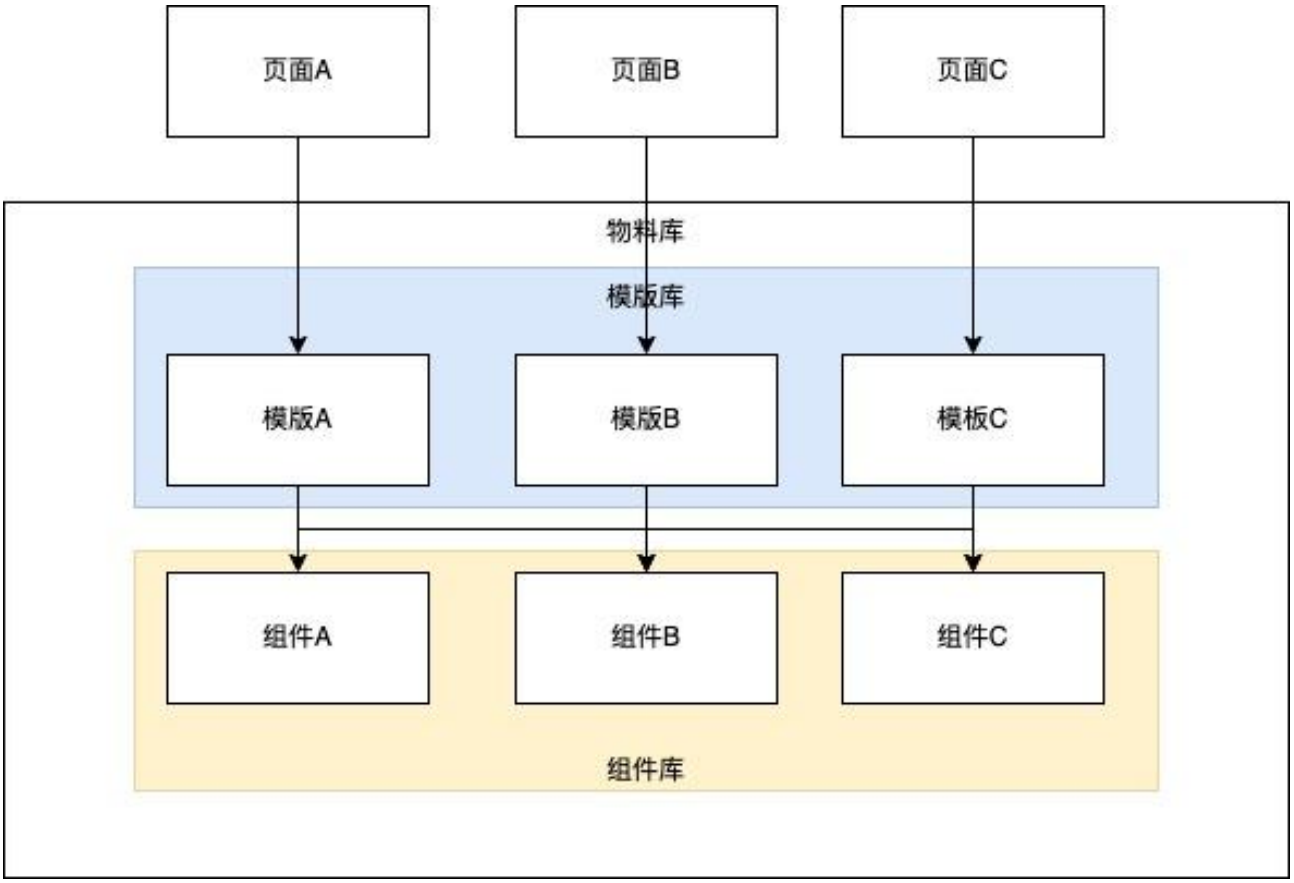
渲染速度。客户端通用模版在收到逻辑层响应的数据后，对数据进行解析后渲染，并发起数据查询请求完成页面数据展示。

No-code 提效平台可以覆盖 50%的用户后台操作需求，在通用模版配置的基础上，按照用户的操作习惯确定交互规范，不需要重复开发页面，用户直接进行配置即可生成对应的页面。并且可以实现前后端数据联动，页面直出数据，也降低了服务端开发成本。缺点是：不支持数据库跨库查询，且交互无法扩展。所以为了解决以上问题，本文引入了 low-code 提效平台。

3.2 low-code 提效平台

Low-code 提效平台不绑定数据库数据，旨在扩展页面交互和数据交互的灵活性，low-code 提效平台提出物料库中包含：组件库、模板库的概念。

结构如下图：

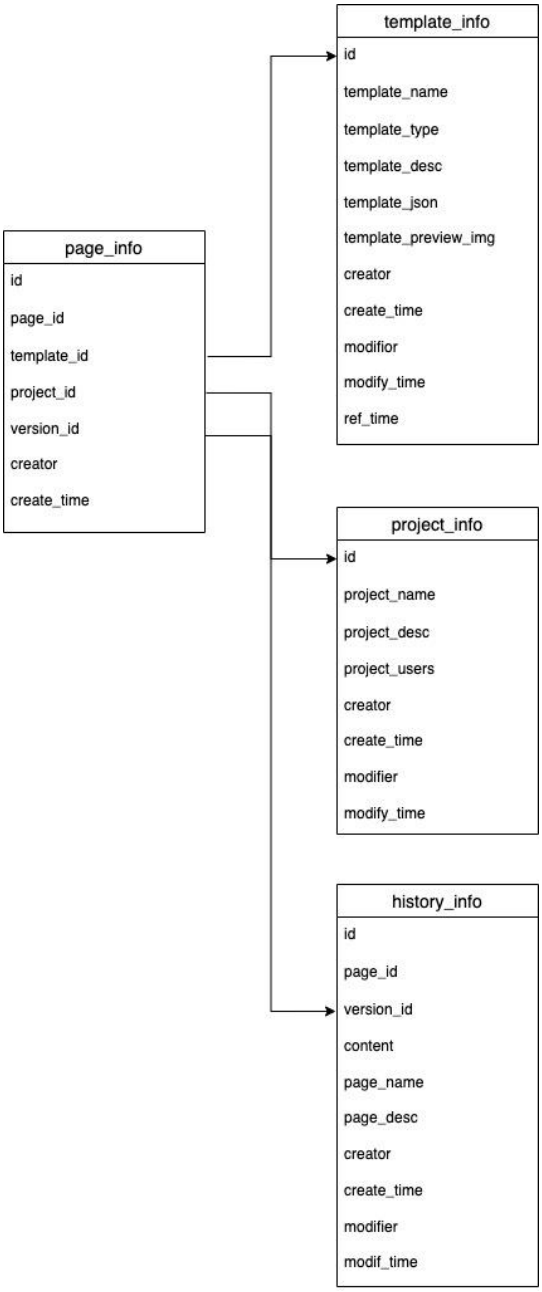


页面通过继承模版生成，模板库中包含若干相对通用的逻辑与若干组件组合形成的模版，组件库中包含若干相对通用的业务组件模块。

物料库依赖于跟随迭代的积累逐渐丰富后，维护开发成本逐渐降低，前期在物料库中的物料无法满足现有需求的情况下，需要开发人员按照规则创建组件和模版以达到丰富物料的目的。物料组件和模板通过对 npm 多包管理工具进行开发和迭代，物料库中的物料通过前端 npm 包管理服务进行版本维护。项目中如果需要使用新物料时，需要拉去最新版本的物料资源完成更新。

数据方面，控制页面的渲染和请求依然依赖于用户在 low-code 配置平台配置生成的 json 结构。json 结构分为两部分，一部分存储页面渲染规则，另一部分用于存储数据拉取规则，不同于 no-code 平台 ，在数据存储部分 low-code 平台存储的是 api 的数据结构而不是数据库表结构，如此设计的目的是为了增加服务对数据处理的灵活性和扩展性。

Low-code 平台本身的数据存储分为四个表：页面信息表（page_info）用于存储页面相关信息；模板信息表（template_info）用于存储模板相关信息；项目信息表（project_info）用于存储项目相关信息，一个项目中会包含多个页面；操作历史信息表（history_info）用于存储用户操作记录相关信息。四张表的关联关系如下图：



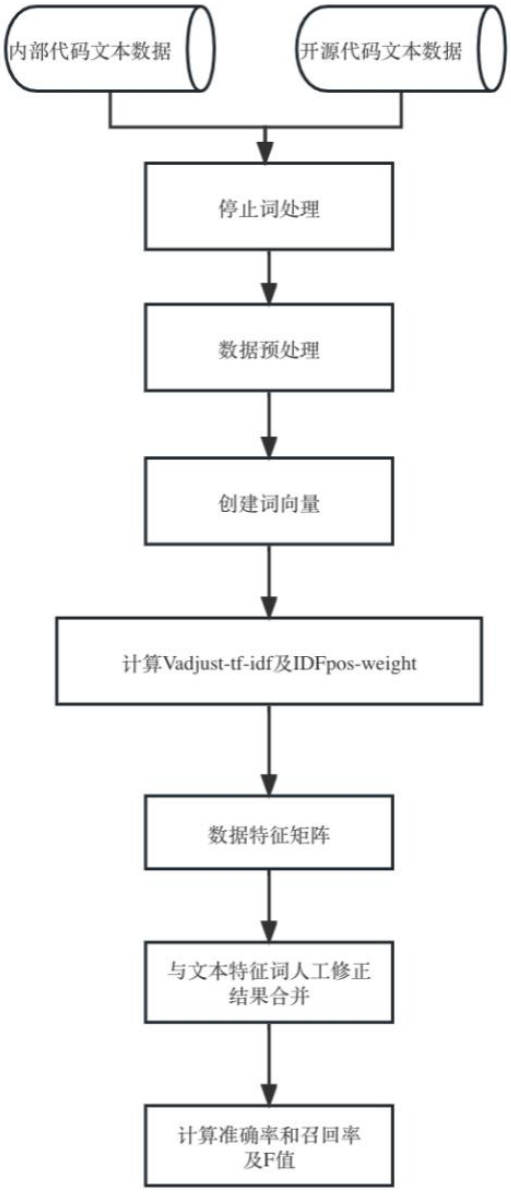
由开发者开发的模版通过 low-code 提效平台模版模块将模版信息录入 template_info 表中。用户创建项目后，基于现有的模版创建页面，将页面相关配置存储到 page_info 表中。同时，数据会在 history_info 表中进行备份。

3.3 pro-code 提效工具

通过 no-code 平台和 low-code 平台基本可以覆盖中后台 90%以上的需求，但实际过程中依然会存在一些需要完全基于现有交互定制化的操作。针对此类需求，通常需要开发人员进行手动编码。基于此类需要手动编码的场景，本文提出使用 pro-code 编码提效工具对开发过程进行提效处理。

pro-code 产品形态是一个编辑器插件。本文只描述基于 vscode 编辑插件，此处称此工具为：无界，下文都使用该称呼。用开发人员在进行开发的时候，无界会根据键入的关键词给出有效的 suggestion 提示，减少开发过程中复制和粘贴等重复编码的操作。

推荐算法的核心 思路如下图：



模型的训练数据包含团队私有项目工程以及 github 一些开源工程，对数据进行分词及停止词的处理后，对数据进行预处理创建词向量，根据 tf-idf 模版构建模型参数，回归模型的准确率和精确度以及 F 值。在后续的应用中开发者键入的关键词后模型根据计算出的权重进行排序和展示，此处会增加一

部分自定义词袋，当用户键入固定的关键词后，无界会给出自定义词袋预置好的高频代码片段作为 suggesion 的内容。

Pro-code 模块本文使用无界解决了平台配置无法覆盖到的个性化需求，从开发者层面提供了便捷的提效方式。

4.结语

本文介绍了微前端架构在微服务架构层面上的应用及最佳实践思路。重点介绍了基于微前端架构的背景下，包含关联库表的 no-code 平台、高扩展性的 low-code 平台、开发过程中 pro-code 工具的提效体系，全方位多角度覆盖现有中后台体系需求，效率提升达 70%以上，有效的降低了用户的操作成本和开发人员的开发成本。为其他系统的开发提供了重要的经验和参考思路。对于后续，如何更好的进一步丰富工具的能力，提升各阶段工具的延展性依然值得我们进一步探索。

引用文献：

[1]王菊雅.基于 Ant Design Pro 的物流系统前端开发与用户体验优化研究[J].软件,2024,45(04):8-10.

[1]刘一田,曹一鸣.微前端化微应用管理控制台[J].计算机系统应用,2020,29(09):126-130.DOI:10.15888/j.cnki.csa.007616.

[1]张浩洋,顾丹鹏.微前端架构在数据管理平台的实践与应用[J].现代计算机,2023,29(21):107-110.

[1]马雄.基于微服务架构的系统设计与开发[D].南京邮电大学,2017.

[1]洪华军,吴建波,冷文浩.一种基于微服务架构的业务系统设计与实现[J].计算机与数字工程,2018,46(01):149-154.

[1]武宁.代码展示方法、装置、电子设备及存储介质[P].北京市:CN202310770454.6,2023-09-29.