一种基于微组件架构的前端低代码平台

张钢岭[1] 陈 聪[1] 郭凌云[1]
ZHANG Gangling CHEN Cong GUO Lingyun

摘 要 针对传统软件开发面临开发周期长、技术门槛高、投入成本大等问题，提出一种基于微组件架构的前端低代码平台解决方案。通过将前端应用拆分为独立的、可复用的微组件，大幅降低开发难度和周期，并允许非专业开发者通过拖放和配置的方式快速构建前端应用。提高了开发效率，同时保证了系统的灵活性和可扩展性。

关键词 低代码；微组件；Single-Spa；JSON Schema；Low-Code

doi：10.3969/j.issn.1672-9528.2025.03.039

0 引言

数字化时代，各类应用程序的需求与日俱增。然而，传统应用程序的开发流程通常开发周期长、成本高、资源消耗大，不适合数字化时代快速迭代，降本增效的要求，对于业务创新和团队效率提出了挑战。为了解决这一问题，低代码平台应运而生。低代码平台[1]作为一种创新性的工具，以其简化和加速软件开发的能力，以及灵活可扩展的特性引起了广泛关注。

本文旨在探讨一种现实可行的前端低代码平台。通过详细介绍其组成，以及实现要点。向读者展示低代码平台需要解决的技术问题，并横向对比各种解决方案。为研究人员、开发人员和企业决策者提供深入了解和评估低代码平台的基础，以推动企业更快速、更灵活的数字化转型。

1 低代码平台分类

从开发方式区分，低代码平台分为自动化代码生成平台和可视化拖拽式平台。它们在工作原理、开发方式和使用群体等方面存在一些区别。

1.1 自动化代码生成平台

这类低代码平台通过解析开发人员提供的元数据、配置或其他规则，自动生成应用程序的部分代码。这种平台旨在提高开发效率、减少手动编写代码的工作量，并确保生成的代码符合一致性和最佳实践。

优点：

（1）快速生成代码：通过自动生成代码模板，开发人员可以快速生成初始代码，节省开发时间和精力。

（2）一致性和可维护性：生成的代码模板遵循一致的结构和规范，提高了代码的可读性和可维护性。

（3）可配置性：自动化代码生成平台提供了丰富的配置选项，开发人员可以根据项目需求和业务规则来进行定制。通过配置选项，开发人员可以定义实体关系、字段属性、业务逻辑、界面布局等方面的规则，从而生成符合特定需求的代码。

缺点：

（1）缺乏灵活性：生成的代码模板可能无法满足所有的开发需求，需要进行额外的自定义编辑和扩展。

（2）学习曲线高：开发人员需要理解生成代码模板的低代码平台的工作原理和使用方法，可能需要一定的学习成本。

1.2 可视化拖拽式平台

这类低代码平台通过可视化界面和拖拽操作来进行应用程序的开发。业务人员可以通过拖拽组件、定义属性和配置事件等方式来构建应用程序的用户界面和逻辑。

优点：

（1）可视化开发：通过直观的可视化界面和拖拽操作，业务人员可以快速构建应用程序，无须编写代码。

（2）即时反馈：可视化拖拽低代码平台通常提供即时预览功能，开发人员可以实时查看应用程序的外观和行为。

（3）降低技术门槛：相较于传统的编码开发方式，可视化拖拽低代码平台降低了对编程知识的依赖，使更多人可以参与应用程序的开发。

缺点：

（1）限制性：可视化拖拽式低代码平台通常提供了一组预定义的组件和操作，开发人员需要在这些限定的选项中

1. 中国航空工业集团公司西安航空计算技术研究所
陕西西安 710066

进行选择。这可能会限制业务人员的创造力和灵活性，同时无法满足复杂应用程序的开发需求。

（2）学习成本高：尽管可视化拖拽式低代码平台旨在简化开发过程，但对于新的业务人员或团队来说，仍然需要一定的学习曲线和培训成本。了解平台的界面、组件和工作流程可能需要一段时间，并且需要熟悉平台的约束和限制。

（3）性能和定制化的挑战：可视化拖拽式低代码平台生成的代码可能不够高效，存在性能瓶颈。此外，对于需要高度定制化和优化的应用程序，可能需要深入了解平台的底层机制，并使用自定义代码进行调整。这可能会增加开发人员的工作量和复杂性。

相较而言，可视化拖拽式平台因其直观易用、灵活定制、所见即所得等特性，更符合企业快速迭代的需要。因此，本文将介绍一种可视化拖拽式前端低代码平台。

2 低代码平台

本文将深入探讨一种可视化拖拽式低代码的核心原理和关键技术。在此之前，本节先简单介绍可视化拖拽式低代码的关键组成部分，以帮助读者建立此类平台的直观认识。

（1）可视化设计器：是可视化拖拽式低代码平台的关键组件。它提供了一个直观的界面，开发人员可以在其中进行应用程序的界面设计和布局。设计器通常包含一个丰富的组件库，其中包含各种预定义的界面组件，如按钮、表单字段、列表等。开发人员可以通过拖拽和配置操作，将这些组件放置在设计区域，并进行布局和样式设置。

（2）属性编辑器：用于配置和编辑界面组件的属性。当开发人员选择一个组件时，属性编辑器会显示该组件的属性列表。通过属性编辑器，开发人员可以设置组件的文本内容、样式、行为等。属性编辑器通常提供了友好的界面和可视化的选项，使属性配置更加直观和便捷。

（3）数据模型生成器：是可视化拖拽式低代码平台中的核心组成部分。允许开发人员定义数据模型的结构和约束。通过定义模型的属性、类型、格式以及验证规则，开发人员可以确保所创建的应用程序符合预期的数据结构和数据质量标准。

（4）数据模型解析器：是可视化拖拽式低代码平台中的核心组成部分。负责解析数据模型，动态生成前端界面，并根据用户定义的验证规则，验证用户输入的合法性和完整性。

通过对可视化拖拽式低代码组成部分的介绍，不难发现此类低代码的关键一方面取决于是否有足够丰富的组件，组件越丰富，此类低代码可以适配的业务范围越广。另一方面取决于使用什么样的模型存储设计好的系统，模型应该具有良好的解释性，扩展性和可验证性。

基于以上两点，本文介绍了一种可视化拖拽式低代码的实现，其框架设计如图 1 所示。

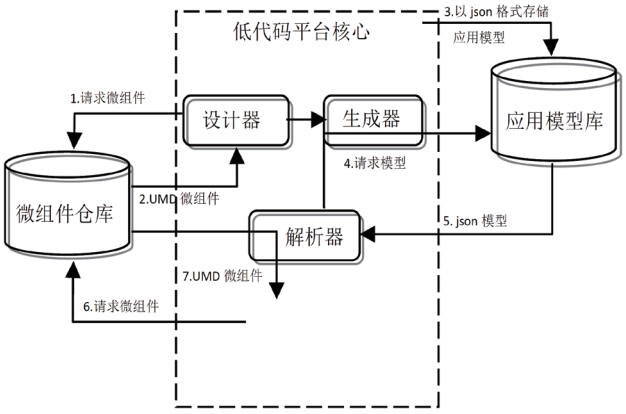

图 1 低代码平台基础框架

如前文所说，拖拽式低代码平台通常提供预设的组件和模型，这些组件和模型虽然易于使用，但也限制了平台的灵活性和可扩展性。因此，如何动态扩展组件，满足业务自由度的需求，是拖拽式低代码平台必须首要解决的问题。本文提供了一种组件和平台解耦，平台支持可热插拔动态获取远程组件的低代码设计方案。下面，本文将从微组件、如何渲染微组件、微组件与基座如何通信三部分介绍低代码平台。

2.1 微组件

微组件是采用微前端 [2] 技术实现的单个组件。它与平台完全分离，但又能集成到平台中组成系统的一部分。本文对比几种当下流行的微前端框架，如表 1 所示。

表 1 微前端框架对比

	Single-spa[3]	Qiankun	Micro-app
技术栈	技术栈无关，支持多种前端框架	技术栈无关，支持多种前端框架	webcomponent[4]
拆分粒度	组件级，页面级	页面级	组件级
Js 隔离	支持	支持	支持
样式隔离	支持	支持	支持
公共模块重用	支持	不支持	不支持

不同的微前端框架各有优缺点，在选用这些框架时，应该根据具体的产品特性和项目需求来选择。一个优秀的拖拽式低代码平台应该具备以下特性。

（1）能够支持多种前端框架。不同的团队可以根据项目的需求和团队的熟悉度选择最适合的前端框架。

（2）支持组件级的前端单元复用。拖拽式低代码平台的优势就在于为用户提供一组功能完备的组件，方便用户快速搭建出满足业务需要的系统。

（3）减少重复模块的加载。出于性能考虑，平台应该维护公共模块，减少不必要的模块下载。同时，相同的模块版本也有助于统一前端风格和功能。

出于以上几点考虑，Single-spa 微前端框架更符合拖拽式低代码平台对于微组件的需要。选定微前端框架后，在实现微组件的过程中，还应该注意一些细节。

（1）组件采用 UMD[5] 规范。通用模块定义规范（universal module definition, UMD），旨在提供一种同时兼容 AMD、CMD 和 CommonJS[6] 规范的代码模块。UMD 模块的目标是让同一个代码模块可以在上述 3 种 JavaScript 模块规范中都能够正常运行。

（2）共享通用模块。微组件通常依赖第三方模块，比如前端框架（如 React[7], Vue[8]）、第三方工具（如 axios，lodash）等。如果每个微组件的产出物都包含所有的依赖模块，势必造成相同模块重复加载的问题（比如每个微组件都要加载 Vue），一方面降低页面响应速度，另一面也容易造成样式和行为不一致。Single-spa 推荐使用 SystemJS[9] 作为前端模块加载器。SystemJS 采用 importMap 加载机制来管理模块路径映射，一方面使得模块的引用更加简洁和可维护。另一方面支持动态加载模块，允许在运行时按需加载模块。同时也确保同一个模块的不同版本，在浏览器中只缓存一份，避免了不必要的数据加载。但 SystemJS 默认只支持 System.register 类型的模块，而本文在前面提到，为了保证微组件的可移植性，规定微组件采用 UMD 模块类型。如果直接使用 SystemJS 加载 UMD 模块类型的组件，会导致加载异常。因此，在加载之前，需要将 UMD 模块类型的微组件转换成 System.register 模块类型。SystemJS 官方给出了一组工具，可以将其他类型的模块转换成 System.register 模块类型，以确保模块可以正常加载。这组工具是不会自动加载的，需要用户根据实际需要加载所需的转换工具。因此，低代码平台需要在页面入口首先加载 UMD 模块类型的转换工具。至此，通过引入 SystemJS 及其 importMap 加载机制，实现了前端通用模块的管理和维护，提升了页面加载效率和响应速度。

2.2 渲染微组件

微组件技术使得组件开发者可以自由选择技术栈开发满足业务需要的前端组件，然后将其发布成 UMD 模块并上传到组件库。低代码平台再通过 URL 地址动态加载组件到浏览器。那么，浏览器在完成微组件加载后，又如何动态使用微组件呢？

一种常用的方法是使用 new Function 构建一个函数的运行沙箱环境，将组件模块的导出对象挂载到事先准备好的目标对象，然后再使用前端框架的动态组件渲染技术将组件渲染到页面中。但这种方法有一定的局限性，即微组件和基座必须使用相同的前端框架，这就限定了组件开发者的技术栈选择范围，不符合微组件框架无关性的要求。

本文使用 Single-spa parcel 来解决动态渲染微组件的问题。Single-spa parcel 是一种构造框架无关组件的实现方法。它具有以下特性：

（1）独立性：每个 Parcel 都是独立的，它不依赖于 Single-spa 应用的生命周期，可以独立加载和卸载。

（2）动态加载：Parcel 可以在运行时动态加载到应用中，无须重启应用。

一个 parcel 本质上就是一个 js 对象，但需要具备一组函数（bootstrap, mount, unmount）以便在特定的时间节点执行组件启动，挂载，卸载任务。为了支持不同前端框架，Single-spa 提供了一组框架相关工具，一方面极大地简化从框架组件转化为 parcel 的过程。另一方面内置 Parcel 组件，用来渲染微组件。

现在，组件开发者使用自己熟悉的技术栈开发组件，并使用 Single-spa 提供的工具将其封装为 parcel 组件，然后将组件打包成满足 UMD 规范的微组件。然后，低代码平采用远程拉取的方式将微组件加载到基座，再通过 Single-spa 的给定工具动态渲染微组件到页面。

2.3 数据通信

当将微组件添加到低代码平台后，微组件不应该被看作是一个独立的单元，而是整个应用系统的一部分。在实际应用场景中，微组件之间以及微组件和基座之间涉及大量的协作和数据共享。因此，低代码平台必须考虑微组件与基座之间的数据通信问题。

数据通信的最简单解决方案是实现一个全局的事件总线，微组件和基座都可以向该事件总线发布和订阅消息。当然，事件总线也有一些缺点，因为业务逻辑分散在不同组件中，造成业务逻辑的处理变得碎片化。在阅读和维护代码时，需要不断寻找事件的发布者和订阅者，导致业务理解和维护成本变大。

前面提到，本文使用 Single-spa 的 parcel 封装和渲染微组件。而 parcel 在封装组件时，可以将一组用户自定义的 props 传递给组件。基于此，本文设计一套消息发布订

阅机制来实现微组件和基座的通信。相较事件总线，本机制具有业务逻辑集中，易于维护的优点。数据通信机制由三部分组成。

（1）全局状态（Global State）。它用于存储所有微组件的状态。每当全局状态发生变化，都会通知所有的微组件。每一个微组件在全局状态中都对应一个唯一的 key，在接收到变更通知后，会比较当前状态与全局状态中 key 对应的值，如果不一致则更新组件。

（2）组件订阅机制（onGlobalStateChange）。作为 parcel props 的一部分传递给微组件，它接收一个函数作为参数。当微组件加载到页面中以后（即上文提到的 mount 回调），首先将全局状态订阅函数作为参数传递给 onGlobalStateChange，并调用执行 onGlobalStateChange，如此则将微组件注册为全局状态的订阅者。

（3）全局状态变更发布机制（setGlobalState）。它同样作为 parcel props 的一部分传递给微组件。当微组件需要通知全局状态更新当前组件状态时，就会调用 setGlobalState，并将最新状态作为参数。而依赖当前微组件的其他微组件或者基座，就可以通过订阅机制接收到最新的状态，进而自我更新。

基于此通信机制，在解耦组件与基座的同时，各组件以及基座之间也可以实现数据通信。从而保证了组件独立开发，但又可以相互合作构成完整的业务系统。

3 Json Schema

当用户基于组件和配置设计好系统之后，需要将系统转换成模型，以便持久化存储。用来描述模型的语法应该具备以下特性。

（1）它应该是一个被广泛接受的标准。使用标准化的格式可以减少误解和沟通成本，使得不同开发者和工具能够更容易地理解和使用数据结构。

（2）它要有足够的灵活性和可扩展性。这样才能覆盖不同的场景。

（3）它要具备可验证性。这样才能保证生成的模型是满足业务需要的，在投产之后，才不会出现各种因为用户数据不符合业务规则造成的问题。

（4）它要易于维护。这有助于减少维护成本。

结合以上要求和前端数据多采用 json 格式的事实，本文将采用 JSON Schema[10] 作为模型语法。JSON Schema 是一种用于描述 JSON 数据结构的工具，它允许开发者定义数据的格式、结构和验证规则。通过使用 JSON Schema，用户可以确保 JSON 数据符合预期的格式。同时，JSON Schema 是一个被广泛接受的标准，有丰富的文档和社区支持。有许多工具和库（如验证器、生成器）都已经实现了对 JSON Schema 的支持，这将降低开发低代码平台的难度。

4 结语

低代码（Low-Code）是一种软件开发方法，通过图形化界面和最少的手动编码，帮助开发人员和非技术人员快速构建应用程序。这种方法使得开发过程更加高效，并降低了对专业技术知识的依赖。随着人工智能和机器学习技术的发展，未来的低代码平台将可能集成更多智能化功能，如自动代码生成、智能建议等。同时，低代码将逐步进入更多行业和领域，如医疗、金融、制造等，推动数字化转型。

参加文献：

[1] 韦青 , 赵健 , 王芷 , 等 . 实战低代码 [M]. 北京 : 机械工业出版社 , 2021.

[2] GEERS M. 微前端实战 [M]. 颜宇 , 周轶 , 张兆阳 , 译 . 北京 : 清华大学出版社 , 2022.

[3] 王佳琪 . 微前端之道：从理论到实践 [M]. 北京 : 清华大学出版社 , 2024.

[4] PATEL S K. Web Component 实战 [M]. 范洪春 , 邵锋 , 何语萱 , 等译 . 北京 : 电子工业出版社 , 2015.

[5] FLANAGAN D. JavaScript 权威指南 [M]. 7 版 . 李松峰 , 译 . 北京 : 机械工业出版社 , 2020.

[6] SIMPSON K. 你不知道的 JavaScript: 中卷 [M]. 赵望野 , 梁杰 , 译 . 北京 : 人民邮电出版社 , 2016.

[7] 亚历克斯·班克斯 , 伊夫·波尔切洛 . React 学习手册 [M]. 2 版 . 安道 , 译 . 北京 : 中国电力出版社 , 2021.

[8] 张益珲 , 曹艳琴 . 循序渐进 Vue.js 3.x 前端开发实战 [M]. 北京 : 清华大学出版社 , 2023.

[9] 尼古拉斯·贝瓦夸 . 精通模块化 JavaScript[M]. 回晓 , 杨蓉 , 陈立伸 , 等译 . 北京 : 电子工业出版社 , 2020.

[10] 汤姆·马尔斯 . JSON 实战 [M]. 邵钊 , 译 . 北京 : 人民邮电出版社 , 2018.

【作者简介】

张钢岭（1983—），男，安徽亳州人，硕士研究生，研究方向：云计算、前端工程化、嵌入式软件。

（收稿日期：2024-11-15）