

杭州电子科技大学

硕士学位论文

题 目： 面向工业设备远程运维的
低代码开发平台设计

研 究 生 黄盼盼

专 业 电子与通信工程

指导教师 陈科明 副教授

企业导师 曾佳

完成日期 2022 年 5 月

杭州电子科技大学硕士学位论文

面向工业设备远程运维的低代码
开发平台设计

研 究 生： 黄盼盼

指导教师： 陈科明 副教授

2022 年 5 月

**Dissertation Submitted to Hangzhou Dianzi University
for the Degree of Master**

**Design of low-code development platform for
remote operation and maintenance of
industrial equipment**

Candidate: Huang Pan Pan

Supervisor: Associate Prof. Chen Keming

May, 2022

杭州电子科技大学

学位论文原创性声明和使用授权说明

原创性声明

本人郑重声明： 所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不含任何其他个人或集体已经发表或撰写过的作品或成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。

申请学位论文与资料若有不实之处，本人承担一切相关责任。

论文作者签名：黄璐

日期： 2022 年 5 月 31 日

学位论文使用授权说明

本人完全了解杭州电子科技大学关于保留和使用学位论文的规定，即：研究生在校攻读学位期间论文工作的知识产权单位属杭州电子科技大学。本人保证毕业离校后，发表论文或使用论文工作成果时署单位名称仍然为杭州电子科技大学。学校有权保留送交论文的复印件，允许查阅和借阅论文；学校可以公布论文的全部或部分内容，可以允许采用影印、缩印或其它复制手段保存论文。（保密论文在解密后遵守此规定）

论文作者签名：黄璐

日期： 2022 年 5 月 31 日

指导教师签名：陈科明

日期： 2022 年 5 月 31 日

摘 要

相比于传统开发模式，低代码开发模式可以有效节约开发时间，减轻 IT 用人压力，提升系统敏捷性，因此被广泛关注。现今，市场上的低代码平台功能多聚焦于 workflow、业务报表以及数据分析，面向的是企业内部业务流程管理 BPM 应用开发。工业设备远程运维系统相较于 BPM，寻求的是工控组态、设备实时监控、设备远程联动、设备故障报警等硬件设备与软件系统的对接功能，目前尚且缺乏这类低代码开发平台。

本文自主研发了一套面向工业设备远程运维的低代码开发平台，利用低代码技术为工业硬件设备与运维软件系统之间架设桥梁，打通了软硬件交互。内部将复杂的底层框架和基础设施抽象为图形化界面，使用者利用拖放型组件以及可视化配置就可以快速搭建工业设备远程运维系统。

在低代码搭建应用的过程中，鼠标框选是核心功能，业界称为选区。针对目前市场上的低代码平台，通常采用矩形包含算法或 AABB 包围盒算法计算选区，其方法存在精度较低的问题。而工业设备具有复杂多样性，更准确高效的选区方案能使操作体验有较大程度提升。为此本文创新性的将计算机物理学中的 CD 算法和仿生学 ACO 算法应用到低代码领域，设计并实现了基于混杂 CD 算法和改进 ACO 的选区方案。通过改进混杂 CD 算法生成 BVH 树，提升包围盒紧密度进而提升选区精度。利用改进 ACO 算法优化 BVH 树的相交检测，使得算法更精确的同时也保证了效率。测试数据表明本方案准确率较现有方案平均提升 36.4%，响应时长平均缩短 25.13%，平均帧率可以达到 54.8fps。

针对目前市场上的低代码平台普遍缺少数据缓冲层，大数据量频繁更新存在卡顿问题，而工业设备运维中对数据的监控有实时性要求，且数据量较大。为此本文设计并实现了基于 MMH3 算法和改进 Diff 算法的缓冲层方案。首先使用 MMH3 算法计算文件缓存摘要，并对 MMH3 算法进行优化。然后使用 Diff 算法设计增量更新渲染方案，并对 Diff 算法进行改进。测试数据表明，使用本方案平台更新渲染时网络传输的数据量可以减少 42.21%，更新节点数量可以减少 44.3%。

本文设计的低代码平台已经在实际项目中得到了应用。基于此平台，已经开发了锯床物联网运维系统、包装机械设备物联网运维系统、工业缝纫机物联网在线监控运维系统等。统计数据显示，采用本文设计的低代码开发平台进行工业设备远程运维系统开发，可以有效减少开发人力投入、加快研发周期、提高项目交付速度。

关键词：低代码、远程运维、选区、数据缓冲、蚁群算法、MMH3

ABSTRACT

Compared with the traditional development mode, the low-code development model can effectively save development time, reduce the pressure on IT personnel, and improve system agility, so it has been widely concerned. Nowadays, the functions of low-code platforms on the market mostly focus on workflow, business reports and data analysis, and are oriented to the development of BPM, platform within enterprises. Compared with BPM, industrial equipment remote operation and maintenance system seeks for the docking function between hardware equipment and software system, such as industrial control configuration, real-time monitoring of equipment, remote linkage of equipment, equipment failure alarm, etc. At present, there is still a lack of such a low-code development platform.

This thesis independently developed a low-code development platform for remote operation and maintenance of industrial equipment, using low-code technology to build a bridge between industrial hardware equipment and operation and maintenance software systems, and opened up the interaction between software and hardware. The complex underlying framework and infrastructure is abstract into a graphical interface internally, and users can quickly build a remote operation and maintenance system for industrial equipment by using drag-and-drop components and visual configuration.

For the low-code platform on the market at present, rectangular inclusion algorithm or AABB algorithm is usually used to calculate the selection area, which has the problem of low accuracy. While industrial equipment is complex and diverse, a more accurate and efficient selection scheme can greatly improve the operation experience. Therefore, this thesis innovatively applies the CD algorithm and ACO algorithm in computational physics to the Low-code field, and designs and implements a selection scheme based on hybrid CD algorithm and improved ACO. By improving the hybrid CD algorithm to generate BVH tree, the compactness of bounding box is improved, and then the selection accuracy is improved. The improved ACO algorithm is used to optimize the BVH tree intersection detection, which makes the algorithm more accurate and ensures the efficiency. Test data show that the accuracy of this scheme is increased by 36.4% on average, the response time is shortened by 25.13% on average, and the average frame rate can reach 54.8fps.

At present, the low-code platform in the market generally lacks data buffer layer, and the large amount of data is frequently updated, so there is a stuck problem. However, the real-time monitoring of data onto the operation and maintenance of industrial equipment requires a large amount of data. Therefore, this thesis designs and implements a buffer layer scheme based on MMH3 algorithm and

improved Diff algorithm. Firstly, the MMH3 algorithm is used to calculate the file cache digest, and the MMH3 algorithm is optimized. Then, the Diff algorithm is used to design the Difference update rendering scheme, and the diff algorithm is improved. The test results show that when the platform of this scheme is used to update, the amount of data transmitted by the network decreases by 42.21%, and the number of update nodes decreases by 44.3%.

The low-code platform designed in this thesis has been applied in practical projects. Based on this platform, the Internet of Things operation and maintenance system of sawing machine, the Internet of Things operation and maintenance system of packaging machinery and equipment, and the Internet of Things online monitoring and maintenance system of industrial sewing machine have been developed. Statistics show that using the low-code development platform designed in this thesis to develop the remote operation and maintenance system of industrial equipment can effectively reduce the investment of development manpower, speed up the development cycle and improve the project delivery speed.

Keywords: Low-code, Remote operation, Electoral district, Data buffer layer , Ant colony optimization, MMH3

目 录

摘 要.....	I
ABSTRACT.....	II
第一章 绪 论.....	1
1.1 研究背景与意义.....	1
1.2 工业设备远程运维的前景与挑战.....	3
1.3 低代码平台的国内外研究现状.....	4
1.4 论文工作及章节安排.....	7
第二章 面向工业设备远程运维的低代码平台设计.....	8
2.1 需求分析.....	8
2.2 总体流程设计.....	8
2.3 平台架构模型设计.....	9
2.3.1 客户端微前端模型设计.....	10
2.3.2 服务端微服务模型设计.....	12
2.3.3 数据库设计.....	13
2.4 平台核心模块设计.....	15
2.4.1 图形元素编辑动作模块.....	15
2.4.1.1 图形元素设计.....	15
2.4.1.2 图形元素组织.....	17
2.4.1.3 基本操作设计.....	18
2.4.1.4 层级与嵌套设计.....	20
2.4.1.5 动作事件设计.....	20
2.4.2 组态设备交互模块.....	21
2.4.3 数据通信模块.....	23
2.4.4 功能设计器模块.....	25
2.4.4.1 流程设计器.....	25
2.4.4.2 表单设计器.....	28
2.4.4.3 表格设计器.....	29
2.5 系统测试.....	30
2.5.1 搭建测试环境.....	30
2.5.2 测试内容与结果.....	31
2.6 本章小结.....	33

第三章 基于混杂 CD 算法和改进 ACO 的选区方案.....	35
3.1 低代码编辑器选区方案分析.....	35
3.1.1 基于矩形包含算法的选区方案.....	35
3.1.2 基于 AABB 包围盒算法的选区方案.....	36
3.1.3 优化方向分析.....	37
3.2 基于混杂 CD 算法和改进 ACO 的选区设计与实现.....	38
3.2.1 总体方案设计.....	38
3.2.2 基于混杂 CD 算法的 BVH 树.....	39
3.2.2.1 混杂 CD 算法及其改进设计	39
3.2.2.2 自动判别包围盒方法(ADBB)实现	40
3.2.2.3 BVH 树的更新优化实现	42
3.2.3 基于改进 ACO 的 BVH 树相交检测算法.....	43
3.2.3.1 BVH 树相交检测的问题	43
3.2.3.2 ACO 算法及其改进	44
3.2.3.3 基于改进 ACO 的包围盒树相交检测算法实现	46
3.3 实验测试.....	47
3.3.1 混杂 CD 算法的对比分析	47
3.3.1.1 改进混杂 CD 算法的 BVH 树构建速度对比测试.....	47
3.3.1.2 改进混杂 CD 算法紧密度对比测试	48
3.3.2 改进 ACO 的包围盒树相交检测算法对比分析	49
3.3.2.1 改进 ACO 的包围盒树相交检测效率对比测试	49
3.3.2.2 改进 ACO 算法寻优对比测试	50
3.3.3 选区方案整体性能对比分析.....	52
3.3.3.1 选区方案响应时间对比测试.....	52
3.3.3.2 选区方案准确率对比测试.....	53
3.3.3.3 选区方案帧率对比测试.....	53
3.4 本章小结.....	54
第四章 基于 MMH3 算法和改进 Diff 算法的缓冲层方案.....	55
4.1 低代码平台缓冲层方案分析.....	55
4.2 方案整体流程设计.....	55
4.3 基于 MMH3 算法的缓存摘要模型.....	57
4.3.1 缓存摘要模型设计	57
4.3.2 MMH3 算法及优化策略.....	58
4.3.3 优化的 MMH3 算法设计.....	59
4.4 基于改进 Diff 算法的差量更新渲染方案.....	61

4.4.1 增量更新渲染方案设计	61
4.4.2 Diff 算法及改进策略	61
4.4.3 改进的 Diff 算法设计	63
4.5 实验测试.....	66
4.5.1 基于 MMH3 算法的缓存摘要模型测试.....	66
4.5.2 基于改进 Diff 算法的增量更新渲染方案测试	69
4.5.3 缓冲层方案整体性能测试.....	71
4.6 本章小结.....	72
第五章 面向工业设备远程运维的低代码平台应用案例.....	73
5.1 锯床物联网运维系统.....	73
5.2 包装机械物联网运维系统.....	75
5.3 工业缝纫机物联网在线监控运维系统.....	77
5.4 本章小结.....	79
第六章 总结与展望.....	80
6.1 工作总结.....	80
6.2 工作展望.....	80
致谢.....	81
参考文献.....	82
附 录.....	86

第一章 绪 论

1.1 研究背景与意义

随着新一代信息技术的不断突破，势必引发一场全球新一轮科技技术革命与行业变革的大潮，在数字经济与实体经济高度融合发展的背景下，工业设备生产方式由传统的标准化、批量生产逐渐向数字经济时代的智能化、信息化、网络化过渡^[1]。中国经济现已迈入增速换挡时期，市场的数字化建设需求呈现出急剧增长的态势。据海比研究院报告的统计数据显示：2021 年我国企业数字服务市场规模仍保持高速增长趋势，总市场规模达到 3.6 万亿元，2020 至 2025 年市场复合增长率达到 20%，预计 2022 年总市场规模将达到 4 万亿。专家们普遍认为，我国数字经济发展逐渐步入下半场，推动数字产业化和产业数字化的不断深化，助力数字经济与实体经济的深度融合成为数字中国建设的主旋律。

我国企业数字化发展已经历了四个阶段：部门信息化、企业级信息化、互联网化、数字化，如图 1.1 所示。从技术演进方向来看，企业实现数字化转型的主要方式无非是两类，一类是由软件程序员用纯代码研发数字化系统，要求开发人员从一砖一瓦开始建起，这对于企业的研发能力、技术水平等都有较高的要求；另一类途径则是通过第三方力量助其数字化转型。2020 年是一个分水岭，2020 年之前的技术方案多为纯代码编程，但随着数字化转型需求激增，软件技术人员出现了大量缺口。据国家统计局、工信部统计，我国 2020 年网民规模为 9.4 亿人，移动互联网用户为 13.2 亿户，而软件从业人员规模仅为 677.5 万人^[2]。在供需关系无法平衡的情况下，以低代码为主导的研发模式将成为企业数字化转型的核心引擎。

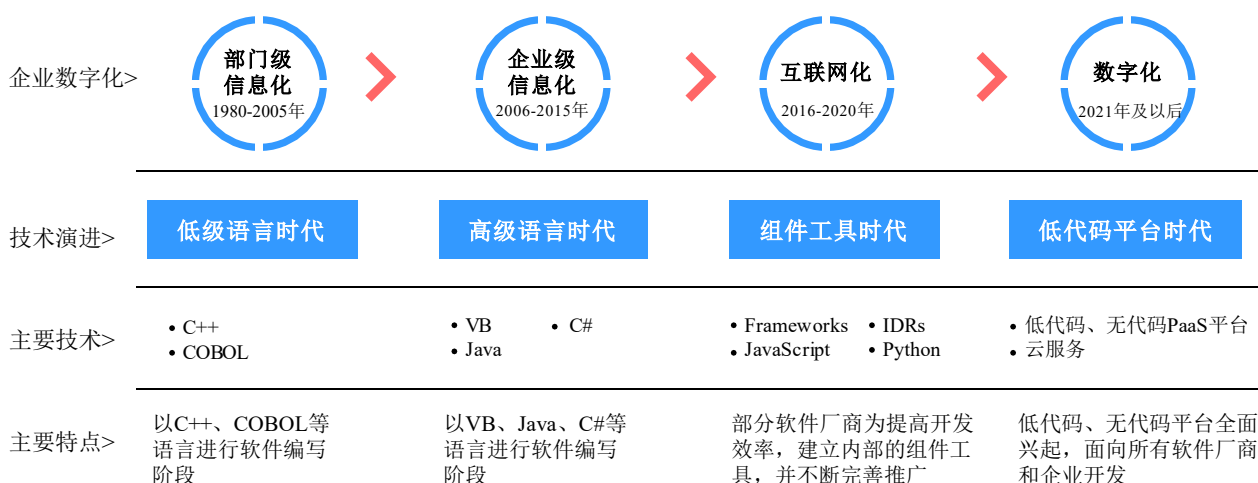


图 1.1 企业数字化发展历程

运维作为传统企业的重要管理环节，数字化、信息化必然是不可或缺模块。随着低代

码技术的兴起，为建设工业设备远程运维带来了新的解决方案。在工业设备领域其服务是依托于各种各样的物理设备上，传统人工运维的形式，需要消耗大量人工成本。基于语言编程建设线上运维平台的形式，则往往需要专业的软件研发人员，并且对研发人员的要求较高，需要兼备软硬件知识，部署和运维应用流程也十分繁琐。在这种背景下，低代码技术日趋成熟，低代码开发平台即能够以最少的手写代码和设置，快速开发应用、配置和部署业务的应用程序^[3]，这是一种全新的开发模式，低代码平台开发流程与传统软件项目开发流程对比如下图 1.2 所示。

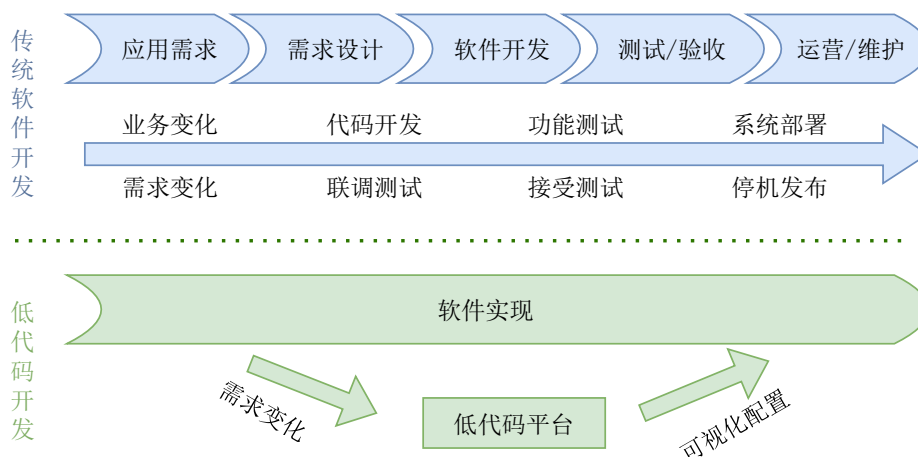


图 1.2 传统软件开发与低代码开发流程对比图

低代码开发平台是一个可视化集成开发环境，为编程提供一个可视化图形操作界面，这样就可以迅速开发应用，同时降低了传统的编程工作量^[4]。使得非 IT 专业人士可以采用拖放应用程序内部组件的方式研发 Web 应用，而不需要了解或者只了解较少的平台编程语言和开发知识，使用友好的可视化图形操作界面来拖拽组件搭建应用内部模块，就能完成页面构建与功能测试。相比于传统软件开发模式，低代码开发的具体优势如下表 1.1 所示。

表 1.1 低代码开发模式与传统开发模式对比及其优势

	传统开发模式	低代码开发模式	低代码开发模式的优势
编程方式	纯代码编程	图形化拖、拉、拽	工作量小、开发门槛低
测试方式	测试、修复代码 Bug	测试、很少有修复环节	套用封装代码，省去测试修复环节
开发模式	紧耦合串行开发	松耦合并行开发	程序全生命周期可视化编程
部署方式	不同环境分别部署	一键部署到多种环境	部署简便

未来在研发领域中，低代码开发将会是主要的技术发展方向，在新产业互联网时代，低代码开发技术将得到迅猛发展。按美国权威机构 Forrester 的预测，2022 年该赛道市场支出预计将高达 210 亿美元^[5]，据 Gartner 预测，到了 2024 年有 65% 的应用软件都会以低代码的形式完成；75% 的大型企业都将会采用低代码开发工具来开发应用程序^[6]。依据海比研究院的数

据统计, 2021 年中国低代码的市场已经达到了 28.5 亿元。从 2021 年到 2025 年, 市场复合增长率为 49.5%, 中国的低代码市场在 2025 年有望攀高至 142.2 亿元, 如图 1.3 所示。整体呈现持续扩张的良好态势^[7]。随着低代码平台日趋成熟, 其能力已经足够承载工业设备领域对远程运维平台的需求。日后, 低代码开发平台将成为工业设备厂商实现降本增效、数字化发展的重要方式。

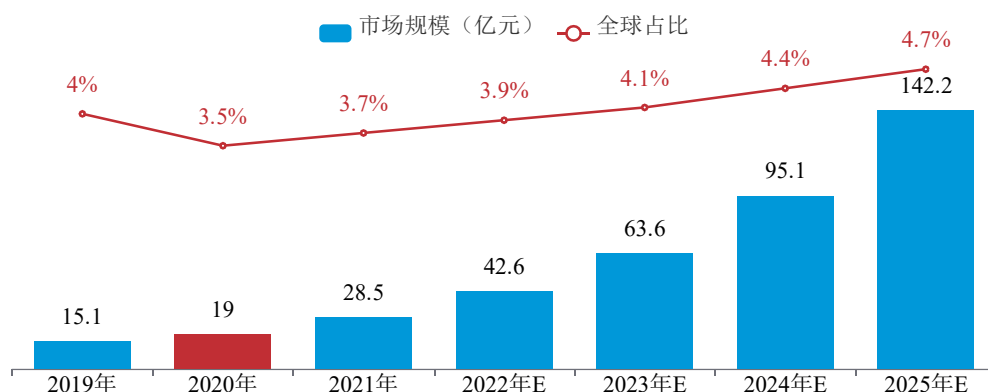


图 1.3 中国低/无代码市场规模及其发展趋势

目前业界已经有了不少优秀的低代码厂商, 但是却缺乏面向工业设备远程运维的低代码平台。当前低代码平台追求的是更通用, 企图使用一种平台解决各行各业的应用需求, 功能多聚焦于 workflow、业务报表以及数据分析, 面向的是企业内部业务流程管理(Business Process Management, BPM)平台的开发。但是工业设备领域的专业性要求更强, 需要的核心功能是硬件设备与软件系统的对接功能, 目前的低代码平台并不能适用于该领域, 所以研发一种针对性更强, 面向工业设备远程运维的低代码开发平台是尤为重要的, 也是十分有意义的。

1.2 工业设备远程运维的前景与挑战

从大环境来看, 国家“十四五”提出信息化的建设指标, 其中明确指出将大数据、人工智能、云计算应用赋能到各个产业当中去, 推动行业的信息化水平^[8]。所以传统行业的数字化、信息化已经进入到一个不可逆转的快速更新进程当中, 尤其是垂直领域的头部行业。在国家政策的大力推动下, 运维作为传统企业的重要管理环节, 智慧化信息化必然是不可或缺的模块。

企业的数字化转型计划是后互联网时代一直绕不开的课题。工业互联网时代, 设备制造企业快速变化的激烈竞争市场环境下, 向数字化制造转型, 是赢得未来的必由之路^[9]。打造工业设备互联网远程运维平台, 是制造业转型升级的关键。依靠远程运维系统能够做到对工艺流程远程监控, 远程设备诊断与线上维保, 对世界各地的设备进行实时监测和记录, 实现故障的提前预警, 对整个售后、维修数据进行记录、查询、分析, 并实现对配件的管理。随着“新基建”时代的到来, 建设一套智能化的工业设备远程运维系统显得愈来愈有必要。

目前, 工业设备运维已经有了一些常用的解决方案, 但随着智能运维系统的需求激增, 工业设备运维也面临了一些挑战:

(1) 业界数据采集监控系统多是 C/S 架构：当前普遍采用的数据采集和监控系统模式是，通过下位机采集数据，数据展示和人机交互由上位机完成，人机交互完全依赖于 Wincc，组态王等组态软件，C/S 架构就决定了其对运行系统有苛刻的要求，往往需要针对 Windows、Linux、MacOS 等系统进行独立开发，也限制了其跨端能力。

(2) 工业设备运维应用产能欠缺：工业设备领域从设备到应用的链路相对来说比较长，存在着很深的行业壁垒，加之该领域的 SAAS 厂商寥寥无几，导致设备制造厂商不得不独立开发运维系统。然而由于软件研发人员欠缺，整体应用研发产能不足。

(3) 工业设备运维应用迭代频繁：对于物联网行业，应用变动频繁，生产工艺流程、设备运行状态等时常需要调整，对迭代速度有一定要求，但是嵌入式开发工程师，设备 OT 工程师等熟悉业务的工程师却不懂软件技术。

(4) 可复用部分较多且项目规模大：对于工业物联网应用，相似度很高，大多数都是对设备的监控，设备数据的呈现，内容较简单，项目规模大。随着从单一业务逐渐向丰富能力发展，应用体积越来越庞大，使用以往的单体应用模式，各功能模块耦合在一起，代码迭代和维护都有一定难度。

反观低代码平台的使用场景，低代码正是为跨端、敏捷开发、快速迭代而生，使用低代码技术，可以使得了解业务的工程师快速高效地搭建运维平台。但当前业界低代码平台提供的可视化数据搭建并不能完全满足工业设备的专业需求，在工业设备远程运维中，核心功能在于对产线设备的监控，专业的组态建模、数据监控等，这是目前的低代码平台无法做到的。考虑到物联网运维平台的体量一般比较庞大，现有的单体应用架构的低代码平台也并不适合，所以，建立一个面向工业设备领域，降低技术门槛、基于 Web 的、可以快速响应，功能复用的低代码平台成为亟待解决的严峻问题。

本文从工业设备远程运维的低代码领域切入，设计实现了一种面向工业设备远程运维的低代码开发平台，并着重从软件设计，低代码平台选区方案，更新策略等角度探讨了如何提升低代码平台性能，助力工业设备企业借助低代码技术获得红利。

1.3 低代码平台的国内外研究现状

近几年，企业面临着互联网数字化变革带来的巨大压力，想要迅速应对产业变革和赶超对手，在高端信息技术人员短缺的状况下，低代码研发得到了企业的关注。低代码设计平台 (Low-Code Development Platform, LCDP)，即使用少许编码或者无需撰写代码就能够迅速建立应用程序的开发平台。低代码的概念衍生历程如图 1.4 所示，此概念最初可追溯到 1982 年出版的《无程序员的应用程序开发》著作中^[10]，由 James Martin 撰写。书中指出每台台式计算机所配备的程序员数量在骤减，这就导致了未来多数的台式电脑需要在无程序员配备的情况下独立运转。2014 年，“低代码”一词才由国际知名的研究机构 Forrester 所正式提出^[11,12]。2021 年我国逐渐形成完整的低代码、无代码生态体系。低代码开发平台，这一允许非技术用户实现自主研发的预想逐渐落地。

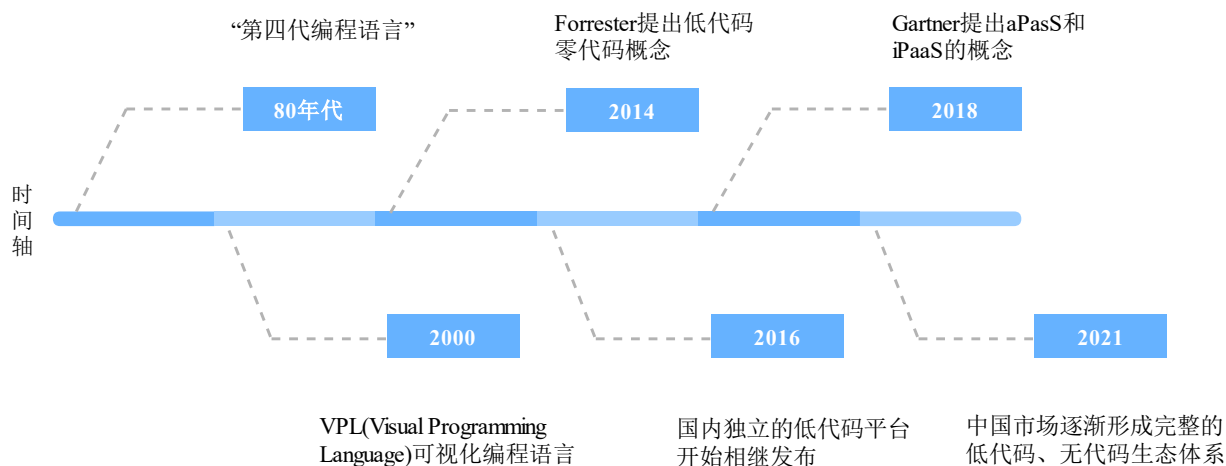


图 1.4 低代码概念衍生历程

随着低代码技术的蓬勃发展，大批平台商加入到低代码行业的角逐中，竞争格局如图 1.5 所示。明星象限的平台商，在产品竞争力、市场表现上较为突出，其技术实力较强，已得到了行业的认可；而金牛象限的平台商，其市场份额较大，运营模式稳固，能保证稳定的资金流；潜力象限的平台商，则具备较强的产品竞争力，尽管现阶段市场份额还较低，但大多数均是因为刚入局，处在尚未发力阶段，所以具有较高的成长潜力；孵化象限的平台商，则多是新成立的平台商，积累少，发展时间较短，尚需时日孵化成长。从低代码平台商的竞争格局来看，国外的低代码产品研发已经历经较长的研究过程，实现了一定的技术沉淀和落地经验。从早期萌芽阶段、探索阶段，到现在的巨头整合阶段，整体发展已经较为成熟，我国低代码技术相对起步较晚，但近几年在疫情、市场、资本的多重驱动下迈进了高速发展期，传统开发的思维枷锁被进一步打破，逐步涌现出一些行业佼佼者。

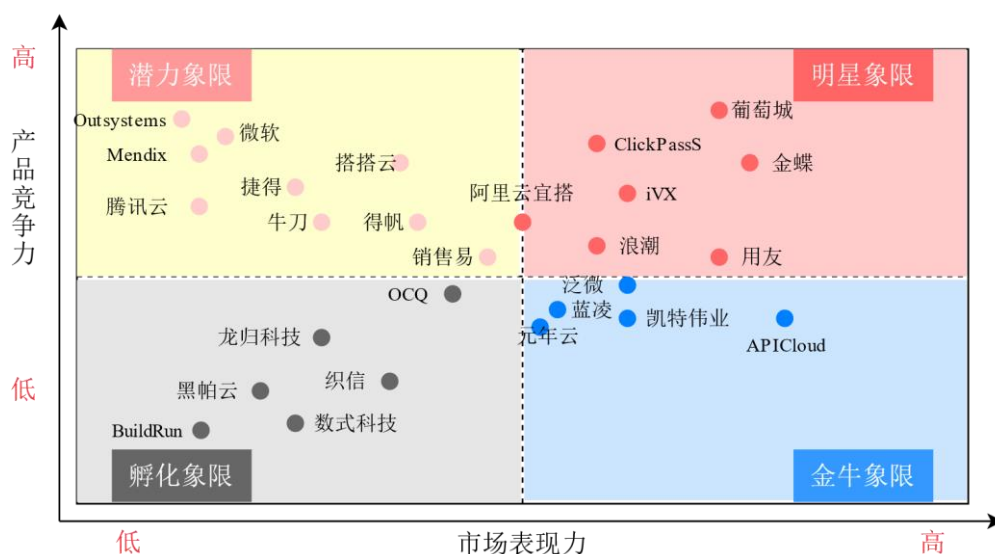


图 1.5 低代码平台的竞争格局分析

分析低代码平台的竞争格局，在国外 OutSystem、Mendix，Microsoft PowerApps 处于市场领先地位。OutSystem 允许开发桌面和移动应用程序，可以在云中或本地基础设施中运行。

具有基本内置功能，提供面向企业级研发、部署上线和管理全流程应用工作套件^[13-15]。但是由于采用了较早的技术，使得 IDE 接口陈旧，不方便操作。Mendix 不需要编写任何代码^[16]，所有功能都可以通过拖放功能访问，同时业务人员可以与程序员实时协作^[17]。提供一个可视化开发工具，可以帮助重用各种组件，从而加快研发过程^[18]。但对于业务人员来说，需要掌握一定的编程语言知识和大量的专业技术研发相关知识。所以，对不了解研发技术的业务人员，不适合采用 Mendix 作为构建软件应用的工具。Microsoft PowerApps，是微软的一款低代码产品，能够支持的应用场景比较广泛，支持拖放功能，并为用户提供一组模板，允许重用已开发的控件^[19,20]。用户可以在构建应用程序时遵循模型驱动或者使用 Canvas 的方法。但是，其对数据文件的要求比较苛刻，对于复杂数据，处理效率很低，对于普通业务人员，上手成本很高。从架构模式和实现技术来说，国外低代码平台多采用 C/S 架构模式，使用时要事先安装软件，况且软件汉化程度有限，技术生态在我国也不活跃。

我国低代码平台商中 iVX、阿里云宜搭、金蝶属于明星象限，综合表现最佳。iVX 是国内通用零代码开发平台的代表，主打不需要任何代码，就可以开发应用并自动生成可以独立部署的源代码^[21,22]，借助 iVX 开发平台的组件拖放和事件配置，可以快速完成各种应用程序搭建。由于主打的是零代码，所以在生成复杂应用时配置项极其复杂，教学内容较少，非技术人员想要完全上手还会有一些麻烦。阿里宜搭，是一款低代码的企业应用搭建平台^[23]。平台内置页面设计、数据收集和分析、业务规则定义三大核心功能，对标企业 BPM 应用。用户通过简单地拖拉拽操作、可视化配置即可实现应用构建^[24]。但是，它最大的问题在于，目前仍处在上升发展阶段，产品、文档、技术支持、培训和售后体系仍在持续完善中，而且依赖于钉钉，使用场景受限。金蝶，属于应用平台即服务系统(Application Platform as a Service, APaaS)，用户可以在无代码的情况下，通过表单表格设计器快速搭建企业级表单应用。内置 28 种标准 ERP 领域模型、624 种模型元素，可实现快速交付。在架构模式和实现技术上，我国低代码平台多采用 B/S 两层架构，两端采用单体模式，由于物料多为工整块状结构，所以选区技术上多采用矩形包含算法或 AABB 包围盒算法。

综上所述，目前国内市场上的低代码平台功能多集中于工作流程、业务报表和统计数据分折，重点面向的是公司内部业务流程管理 BPM 平台的研发。以钉钉宜搭低代码平台为例，据阿里公开的数据显示，目前的用户群体大多数集中在行政、HR、财务等角色^[25]。而对于工业设备远程运维系统而言，其相较于 BPM 平台，寻求的是工控组态、设备实时监控、设备远程联动、设备故障报警等硬件设备与软件系统的对接功能，对标的用户角色是对设备较为熟悉的硬件嵌入式等工程师，目前尚且缺乏这类低代码开发平台。从实现技术来看，B/S 两层架构更符合国内生态，但单体架构随着版本迭代体积将变得越来越臃肿，所以本文采用微前端-微服务架构。使用 B/S 架构在数据量大时稍显卡顿，所以本文在浏览器与服务器之间加入了数据缓冲层。由于面向工业设备运维，平台物料复杂多样，为了提升选区精度，本文设计使用混杂 CD 算法和改进 ACO 进行选区。

1.4 论文工作及章节安排

本文从工业设备远程运维领域实际需求切入,将低代码、智能化运维与工业设备相结合,并以微服务、微前端,数据缓冲技术,选区方案作为实现基础,详细描述了一种面向工业设备远程运维的低代码平台的设计与实现。

在团队协作中本人所承担并完成的工作内容为:参与整体架构设计提出微前端-微服务架构,在前端层面将 `iframe` 容器化,微件化,路由分发相结合设计混合微前端架构模型,后端使用 `Service Mesh` 服务网络的模式,运用容器技术以 `Kubernetes` 作为基础设施,结合服务网格化实施服务治理;设计并完成低代码编辑器画布交互功能,重点研究选区方案,针对目前低代码平台选区精度较低问题,提出基于混杂 `CD` 算法和改进 `ACO` 算法的选区方案,将 `CD` 算法和 `ACO` 算法应用到低代码领域,通过改进混杂 `CD` 算法生成 `BVH` 树,提升包围盒紧密度进而提升选区精度。利用改进 `ACO` 算法优化 `BVH` 树的相交检测,使得算法更精确的同时也保证了效率;设计并完成缓冲层方案,针对画布交互中数据同步的性能问题,在存储方式上采用改进 `MMH3` 算法的生成缓存摘要,在渲染方案上采用改进 `Diff` 算法进行增量更新渲染,从根本上解决问题,降低数据同步处理开销,优化平台整体性能。

全文共分为六章,文章的组织结构如下:

第一章:绪论。简要介绍了面向工业设备远程运维的低代码平台的研究背景和意义,分析了工业设备远程运维的前景与面临的挑战,然后对目前市场上的低代码平台研究现状进行讨论,指出了当前低代码平台应用在工业设备远程运维领域的欠缺,进而强调了设计实现面向工业设备远程运维的低代码开发平台的必要性和重要性,以及平台整体的创新之处。最后,就各个章节的研究重点和文章的总体组织结构进行了阐述。

第二章:面向工业设备远程运维的低代码平台设计。首先对平台需求进行分析,设计总体流程,在架构层面提出微前端-微服务架构模型。然后,对平台核心模块进行详细设计,最后测试系统性能。

第三章:基于混杂 `CD` 算法和改进 `ACO` 的选区方案。首先对目前选区方案进行分析,针对现有选区方案中存在的问题,设计了一种混杂 `CD` 算法和改进 `ACO` 的选区方案,详细阐述了选区方案的设计思路以及混杂 `CD` 算法和 `ACO` 算法的改进流程,最后评估方案整体性能。

第四章:基于 `MMH3` 算法和改进 `Diff` 算法的缓冲层方案。首先对面向工业设备远程运维的低代码平台中的缓冲层方案进行分析,讨论了建设缓冲层的必要性。然后设计总体流程,提出基于 `MMH3` 算法的缓存摘要模型,并对 `MMH3` 算法进行优化。其次,提出基于改进 `Diff` 算法的增量更新渲染方案,并对 `Diff` 算法进行改进,最后进行测试分析。

第五章:应用案例。介绍应用本平台搭建的实际运维系统,展示运行结果,验证平台整体的完备性和优越性。

第六章:总结与展望。首先,对面向工业设备运维的低代码平台全文及其创新成果进行总结,并思考平台中可能出现的新问题与改进思路 and 方向,最后对后续研究目标进行了展望。

第二章 面向工业设备远程运维的低代码平台设计

本章对面向工业设备远程运维的低代码平台进行整体设计。首先，针对目前工业设备远程运维领域发展现状和低代码平台面临的缺陷，对平台需求进行分析。其次，对平台整体流转过程进行了设计。针对单体应用体积臃肿、维护困难，可拓展性差等问题，设计了一种微前端-微服务软件架构方案。然后，详细地设计了平台的核心模块，最后测试系统性能。

2.1 需求分析

针对当前市面上低代码平台并不贴合工业设备运维领域、现有运维系统维护困难、应用体积臃肿等问题，本文实现的面向工业设备远程运维的低代码平台，需要满足以下几点要求：

(1) 软件架构的合理性。保证敏捷开发，快速迭代，可以从容应对需求变更，工业工艺流程变更，设备升级等场景。

(2) 组态搭建。用户可以通过拖拽建模的方式搭建组态界面，并完成设备的数据绑定。可以通过搭建的可视化图形界面远程下发设备指令。

(3) 底层设计器引擎支持。包括运维系统所需要的设计器工具，用户可以通过可视化配置的形式，进行 workflow 搭建和表单表格搭建。

(4) 达到企业级平台性能指标。保证良好的用户体验，页面稳定美观，搭建过程流畅，页面更新渲染迅速。

2.2 总体流程设计

低代码平台旨在快速构建应用程序。使用低代码平台搭建应用的整体流程如下图 2.1 所示。首先，用户进入登录界面，登录成功后进入到应用控制台。用户可以通过控制台管理数据、管理应用、配置权限等，控制台同时也集成了常用的研发辅助工具。通过控制台中创建应用或编辑应用入口，可以进入到低代码编辑器界面。低代码编辑器是构建应用的核心，其中内置了大量的图形元素和行业模板，用户在使用过程中，可以将左侧工具栏中的图形元素采用拖拽的方式拖到中央画布中，右侧为配置栏，提供整体配置和为图形元素配置事件、属性参数等。

在页面搭建过程中，用户通过组件的拖拉拽，按照对业务的需求完成页面布局、模块组装以及各种样式调整。同时平台提供了预览功能，用户可以实时预览当前页面的效果通过预览适当地调试，当预览界面符合页面需求时，就可以点击发布按钮，发布应用。平台提供一键发布功能，可以快速发布应用到测试环境进行测试，同时发布成功后控制台将生成一条该应用的状态信息，完成整体业务流程闭环。



图 2.1 低代码平台搭建应用总流程

下面对低代码平台中的相关概念进行必要解释：

(1) 控制台：控制台是本平台提供的 Web 控制界面，用于管理数据资源，应用编辑，以及用户权限配置。

(2) 低代码编辑器：承载着低代码平台的核心功能，包括拖拽建模、编排、组件配置、画布渲染等。支持编辑、保存、实时预览、发布应用流程，同时支持设备数据绑定，设备指令事件绑定。此外，即提供了无代码模式，也提供低代码模式，有一定 JavaScript 基础的使用者可根据业务需求完善应用逻辑的丰富度和复杂度。由于其功能多，打磨精细难，也是低代码平台建设最耗时的地方。

(3) 业务流程设计器：将各种业务管理活动抽象成各种形式的业务流程节点，通过业务流程设计器能够采用可视化拖拽设计的方法实现业务流程节点的连接。还可实现业务流程触发、流转、审批、消息通知等操作。

(4) 表单设计器：表单设计器将业务中常用表单组件进行抽象，使用者可以通过可视化配置的方式快速创建表单并进行数据源绑定。

(5) 表格设计器：凭借表格设计器强悍的报表设计能力，使用者能够在线上完成报表的设计，表格设计器支持各类 Excel 报表设计，类 Word 文档类报表，常规报表等报表类创建。

2.3 平台架构模型设计

面向工业设备远程运维的低代码平台是一个完整的系统，客户端设计，服务端设计，数据库设计等都需要进行充分考虑，平台架构图如下图 2.2 所示。

整体架构由上至下分为四层：客户端、数据缓冲层、服务端，数据库，客户端采用了微前端架构，将一个应用划分成若干独立的子应用来执行。数据缓冲层是连接客户端与服务端的中间层，旨在将低代码搭建页面生成的配置文件数据等进行缓冲。服务端也对应的采用了

微服务架构。通过分解职责，将任务分割为若干微服务，每一项都在相应的单独进程中独立运行，服务与服务之间相互分离。数据库相应使用通用数据库与分布式数据库结合的结构设计。

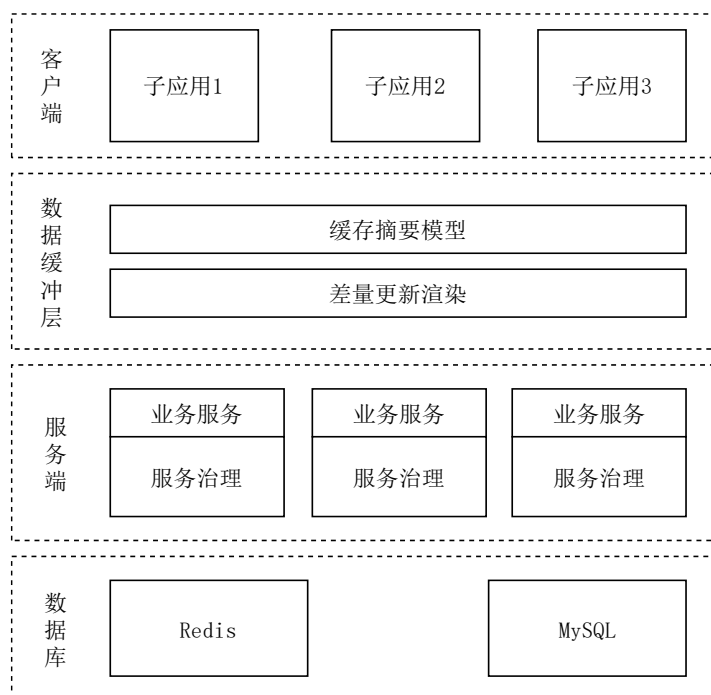


图 2.2 面向工业设备远程运维的低代码平台架构图

2.3.1 客户端微前端模型设计

微前端不是一个框架或库，而是一种前端架构的设计思路，要实现微前端，核心就是隔离团队代码^[26]，杨彩芳^[27]采用了微件化的方式，借助 Webpack 打包工具，独立构建组件和应用，产生各个单独的 chunk 文件，并在构建后将产生的 chunk 文件再进行归类。使用懒加载的方式在运行时加载各个模块，这种方案适合使用同一种框架且业务通信频繁的场景，但它有严格的限制，就是必须使用同一个框架构建。李学森^[28]和胡如乐^[29]等人采用了路由分发式，通过 HTTP 服务器转发实现任务调度，将原有的巨石前端应用改造成了各个相互独立的子服务，每个前端子服务有高度的独立性，可以独立迭代更新。但是这种方案在路由跳转的过程中，经常需要刷新页面，重新加载资源文件，白屏等待时间是不可忽略的应用用户体验问题。Daojiang W^[30]采用了 iframe 容器化的方案，通过嵌入不同的 iframe 容器实现业务拆分且相互隔离，利用 iframe 构建微前端框架，即能够使得各个项目之间都保持着高度的独立性，还能够有效的防止不同项目的 JavaScript、层叠样式表(Cascading Style Sheets, CSS)之间的交互影响，每个项目都不需要部署到同一个域名上，而且可以通过引入第三方的方式，将一系列的 API 对外公布，使第三方基于这套 API 进行开发。但是这种方式的缺点也很明显，因为应用是独立的，所以不能在生成时抽取公用的依赖，搜索引擎优化(Search Engine Optimization, SEO)支持度也较差。

综上可知，目前的微前端架构方案都或多或少存在了不可忽视的问题，路由分发式，在路由跳转的过程中，会出现白屏过程；微件化方案必须统一框架；iframe 容器化无法提取公共依赖，且对 SEO 支持度较差。所以本文开拓性的提出一种组合式微前端架构方案，有效解决了以上痛点。考虑到系统的鲁棒性和实际应用场景，本文认为混合技术是最合适的解决方案，在本文的具体实施中混合使用了路由分发、iframe、微件化的方式，或者说是某一种方案的变种，因为本平台同时具备了这几种方案的特点。

在本系统微前端的架构设计中采用基座模式^[31]如图 2.3 所示。基座模式是由一个主应用和一系列业务子应用构成的系统，并由主应用管理其他子应用。本文将基座主应用作为系统的统一入口，通过前端路由，将用户路由到相应页面。基座应用管理子应用行为，在应用划分上，按照功能划分为基础功能与低代码编辑器两部分，基础功能包含了权限管理，数据源管理，应用管理等功能。在微前端架构技术方案选型上，基础功能部分依赖相同的公共基建，所以采用微件化架构。低代码编辑器作为一个全新的独立宿主环境，即可以独立运行，又可以集成到系统中，且与基础功能模块不共享依赖，所以更适合 iframe 容器化架构。这将比简单使用单一架构有更好的用户体验。

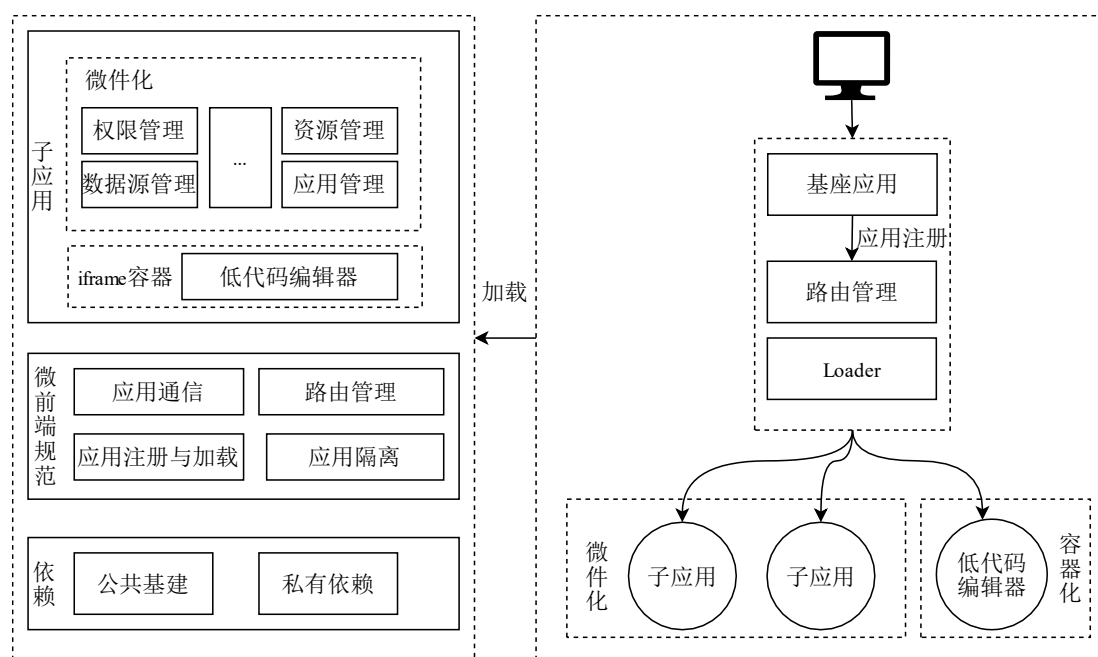


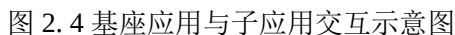
图 2.3 客户端微前端架构示意图

由上图 2.3 基座应用与子应用交互示意图所示，整个方案非常清晰明确，按照业务功能将路由按级别进行了拆分，平台由基座提供入口，加载各个子应用。

（1）基座应用设计

对于基座应用，整体使用 Vue 框架设计，Vue 是一种渐进式的由数据驱动视图更新的 JS 框架^[32]，使用 MVVM(Model-View-View-Model)初始化数据，采用发布订阅模式达到数据响应式的目的。基座应用不涉及任何业务功能，仅负责整体架构运转，资源加载，全局管理。在

按照功能对子应用进行划分，每个子应用对应自己的页面和路由信息，其中基础功能使用微应用架构，并维护一个统一的公共库和 **store** 层，通过顶级 **store** 应用间可以进行数据共享及通信。低代码编辑器使用 **iframe** 接入，拥有自己独立的公共库。低代码编辑器和其他应用使用基于 **postMessage** 和浏览器原生封装的通信模块进行应用间通信和数据共享，如下图 2.4 所示。



单体架构由于随着业务拓展以及用户数量的增加,已无法再采用单纯横向扩展方式来提高网络系统承受能力,也无法适应物联网系统中大量的设备连接需求,所以微服务架构被广泛应用于物联网平台的构建。然而,微服务虽然解决了由于传统单体架构过紧相互耦合、整体研发模式、整体部署而造成的开发迭代困难和性能瓶颈等弊端,但同时也产生了若干问题。微服务细颗粒化会造成服务管理非常复杂,对于如何解决服务治理中复杂困难的问题^[33],目

前业界已经有了较为完善的方案，张玄弋^[34]等人已经使用 Spring Cloud 服务治理框架实现了各个微服务的自动化注册和发现，李新^[35]使用了 Dubbo 架构，实现了透明化和高效率的远程过程调度(Remote Procedure Call, RPC)以及面向服务的结构管理(Service-Oriented Architecture, SOA)。但不管使用 Spring Cloud 还是 Dubbo 框架，实现服务治理时，都需要引入所有接入服务的组件，再从业务服务进行暴露，并同时消费相应的服务，这样就大大增加了服务治理与业务服务之间的耦合程度^[33]，进而提高了服务开发的费用。

在本系统服务架构设计中，采用 Service Mesh 服务网格的模式，运用容器技术以 Kubernetes 作为基础设施，把网络代理技术运用于服务治理中作为基础设施的边车，并结合服务网格化实施服务治理。使用 Istio 服务治理平台中的组件实现了服务发现功能，从流量控制和容错策略设置两个方面对微服务进行容错设计。设置预选策略和优选策略进行服务调度和迁移，同时使用服务端 Mixer 组件采集服务运行中的遥测数据信息来实现服务监控功能，系统架构图如下图 2.5 所示。

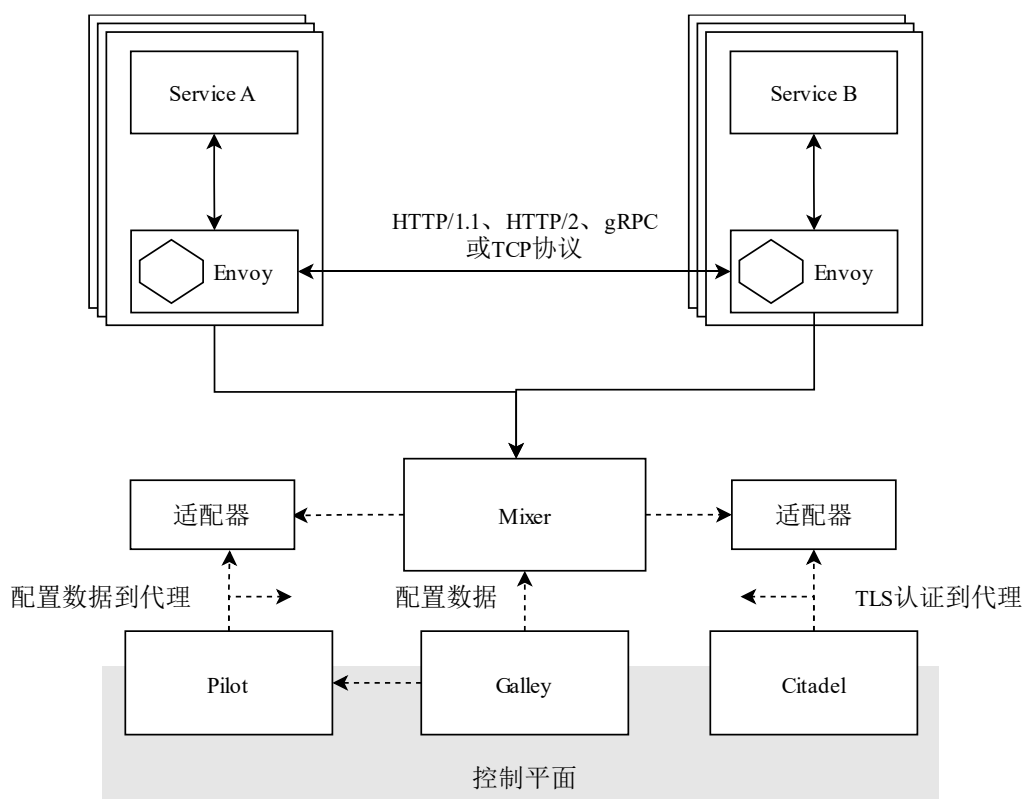


图 2.5 微服务架构设计图

2.3.3 数据库设计

数据库设计是系统设计的一个关键点，本文为每个服务都建设了自己的独立数据库，只有其自身才能使用。微服务相互之间通过服务调度，主、从表的方式来实现数据信息资源共享。本文数据库采用微服务的设计方案，系统将不具备依赖或耦合的服务分为一组，每组内的多个服务具备依赖关系，而不同组之间的任意服务都不具备依赖关系。平台中的核心数据

库及其表名称和对应实现的功能说明如下表 2.1 所示。

表 2.1 核心数据库及其功能表

数据库名称	数据库表名称	表功能
基础数据库	base_client	客户端表
	base_dict_biz	业务字典表
	base_log_api	接口日志表
	base_log_error	错误日志表
	base_log_usual	通用日志表
	base_menu	菜单表
	base_notice	通知公告表
	base_oss	对象存储表
	base_role	角色表
	base_role_menu	角色菜单关联表
	base_scope	权限表
	base_user	用户表
低代码服务数据库	code_datasource	数据源配置表
	code_param	参数表
	code_process_leave	流程业务表
	code_report_file	报表文件表
	code_configuration	组态配置表
	code_image	存储图库图片
服务数据搜集数据库	pod	集群 pod 表
	pod_service	pod 与服务实例关联表
	pod_node	系统节点与 pod 关联表
	pod_resource	pod 资源表
	service	服务实例表
	service_config	服务配置表
	service_instance_config	服务实例与服务配置关联表

2.4 平台核心模块设计

低代码平台的关键在于使用拖拽搭建的方式进行界面开发，其中低代码编辑器是其主要的功能，界面如下图 2.6 所示。

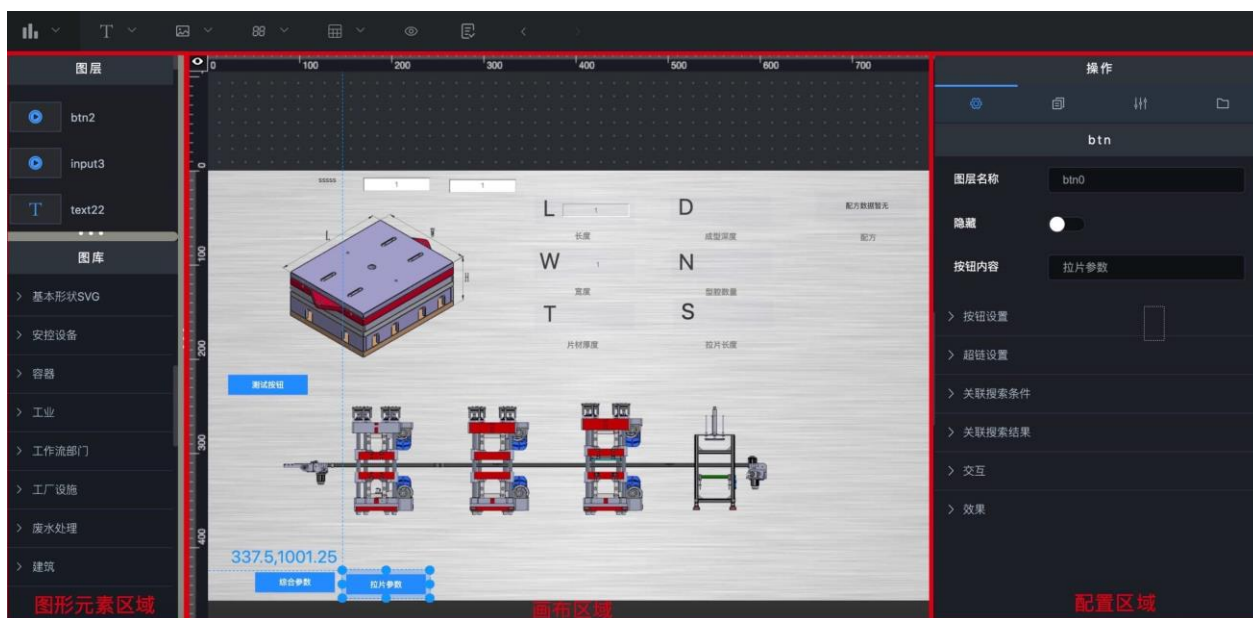


图 2.6 低代码编辑器界面

使用者拖拽的每一个组件称为图形元素。由此可知，低代码开发的页面是由一个一个图形元素组成的，所以图形元素是贯穿整个低代码开发的主体，它们为用户操作时真正的对象，可以说图形元素是低代码编辑器系统的基建部分。本平台相比于其他低代码平台的优势就在于，面向的是工业设备运维领域，所以在系统的设计上与普通的低代码平台有所不同，提供了更适合工业设备远程运维领域的专业模块。为了更贴合物联网场景，在图元设计上内置了大量工业可缩放矢量图形 (Scalable Vector Graphics, SVG) 元素。为了更生动形象的表示整个工艺流转状态，内置了动态 Canvas 数据条。此外，针对工业设备运维领域，设置了设备数据模拟配置功能，提供了设备真实动态数据的实时接入。同时也支持设备指令下发事件，满足了用户通过模拟建模与真实设备实时联动的需求。

2.4.1 图形元素编辑动作模块

2.4.1.1 图形元素设计

低代码页面是由各种基本图形元素构成的，选定合适的图形元素格式是系统运行的基础。在前端的图形化技术中，主要分为 Canvas 绘图和 SVG 绘图两种。Canvas 绘图规定了一种可以使用 JavaScript 脚本绘制的画布，在使用 HTML 标签定义了一个容器之后，可以使用脚本获取到容器通过编程技术生成图形。SVG 绘图技术是一种使用可扩展标记语言 (eXtensible Markup Language, XML) 定义的可缩放矢量图形，也就是说相比于 Canvas 技术 SVG 技术绘图

是由大量的 XML 标签堆砌而成的，基于它的矢量特性，可以保障在缩放或改变尺寸时图形质量不失真。黄华梅^[36]等人和张丽^[37]采用了 SVG 绘图技术绘制图元，这种方式保证了图形元素矢量缩放，但是在大量应用时，DOM 体积将变得很臃肿，所以针对图形元素的应用场景，合理选择绘图方式是很重要的，本项目中图元设计分为两部分，元件库和图库，包括内容如下图 2.7、图 2.8 所示。

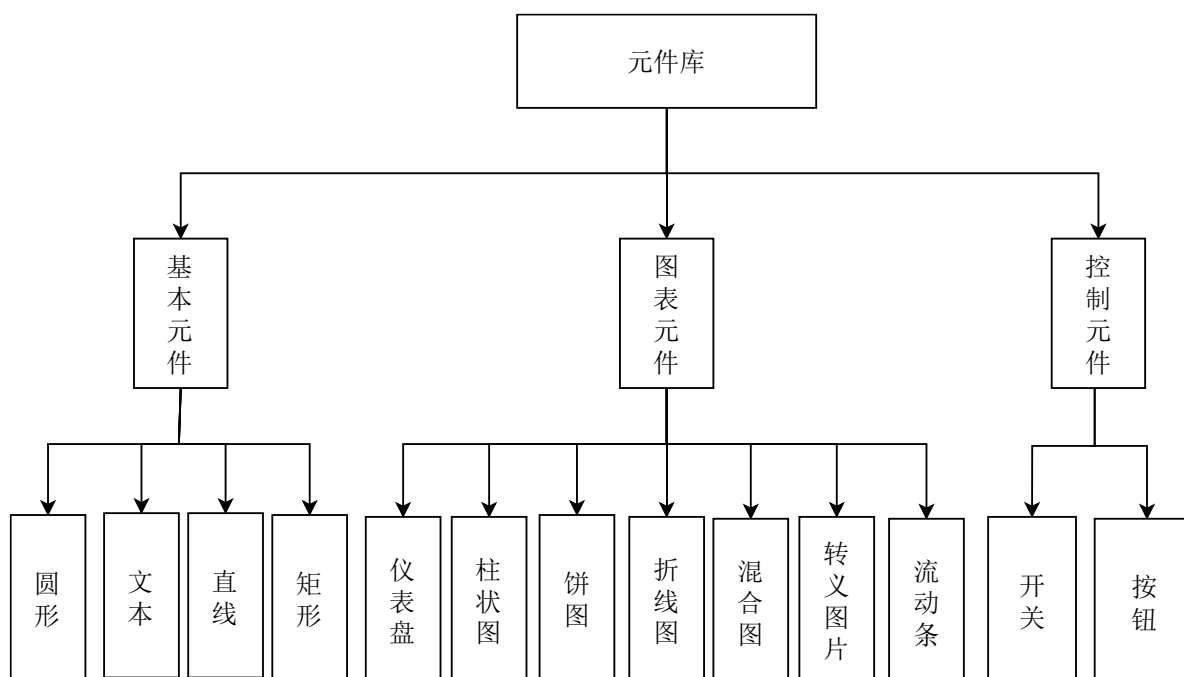


图 2.7 元件库组织图

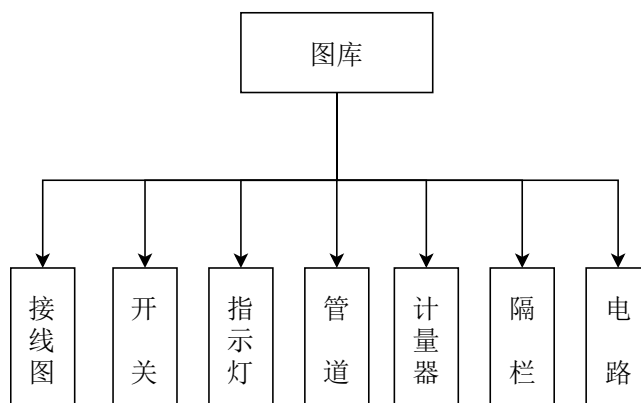


图 2.8 图库组织图

其中元件库中的基本元件要求伸缩不失真，所以采用 SVG 绘图，图表元件基于 Canvas 进行绘图，控制元件导入 SVG^[38]格式图片。图库中也对图形的矢量性有要求，所以采用 SVG 格式的图片。

本项目中的图表组件与动效数据流都是基于 Canvas 绘图技术绘制的，如下图 2.9 所示。

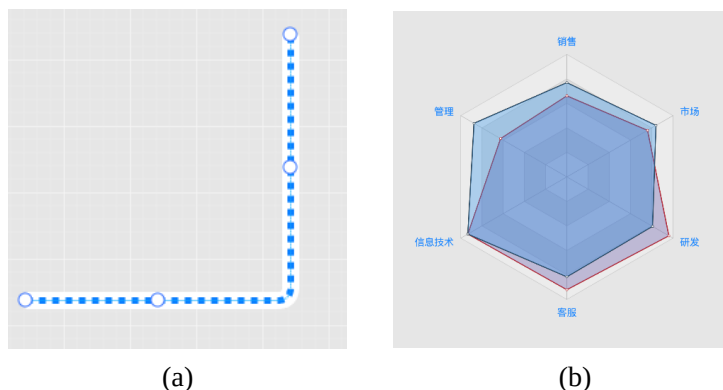


图 2.9 基于 Canvas 绘图技术实现的图形元素
(a)动效数据流 (b)雷达图表

2.4.1.2 图形元素组织

本文在描述图形元素过程中定义了一种领域特定语言(Domain-Specific Language, DSL), DSL 语言用于表示某个领域中的问题以及构建对应的解决方案^[39]。在低代码开发平台中,用户通过拖拽搭建的方式操作图形化组件搭建页面。如何用语言描述组件数据,相对位置大小,类型结构等信息是十分重要的。设计一门使用自然语言描述的 DSL 不是一件容易的事情,为此我们使用数据描述的 DSL。本平台使用 JSON Schema 来最终表示页面配置,同时作为传输时的结构。JSON Schema 主要用于描述 JSON 数据,通常 JSON 中不同的 key-value 数值,有 string 类型也有 object 等数据类型。针对不同的应用场景,约定 JSON 的数据所包含的字段、字段值、以及字段值的类型甚至依赖关系,需要使用规范来描述这些约束的关系。而这也是 JSON Schema 所存在的意义。

但是在图形展示时,JSON 作为字符串解析起来相对不便,所以在展示时考虑到使用其他的组织结构,散乱的对象结构势必会增大解析难度和研发人员的工作量,不同的组织结构的表现也会带来不同的效益和问题,所以选取一个适合的组织结构是至关重要的,考虑到图形元素应该是具有语义性的,我们最终选择树形结构,树形结构是一种嵌套结构,这种结构可以很大程度的维护逻辑关系,结合元素本身的语义信息,这样做可以有效的提升效率。但是使用树形结构,不可避免的要处理位置组织变动而触发的更新问题,因为当元素的逻辑语义变更时,相应的树形结构的位置信息也有进行更新,当发生频繁变动时,带来了额外的维护成本^[40],正因为如此在第四章中我们提出了采用 Diff 算法对新旧树形结构进行差异化对比,借鉴了 Vue 框架中虚拟 DOM 树的 Diff 思想,实现了改进的 Diff 算法,改进后的 Diff 算法具有速度上的优越性,异步更新解决了频繁变动结构所带来的效率降低问题。

为了增强图形元素代码的可复用度,本文基于 JavaScript 类创建了一个生成对象的模板 baseClass 如下图 2.10 所示。其中包含了各类图形元素的公共特征和共有属性信息,通过 JavaScript 对象的原型链,在 baseClass 原型上定义了基本配置对象 config、颜色的配置对象 colorOption、表格的列配置对象 tableOption、字典配置对象 dicOption、加载图片素材库配置

对象 `imgOption`，这些对象又对应了各自的派生对象与派生方法，当创建子类时，子类通过原型链回溯可以访问到父类对应的方法，也可以根据自身需要进行修改。

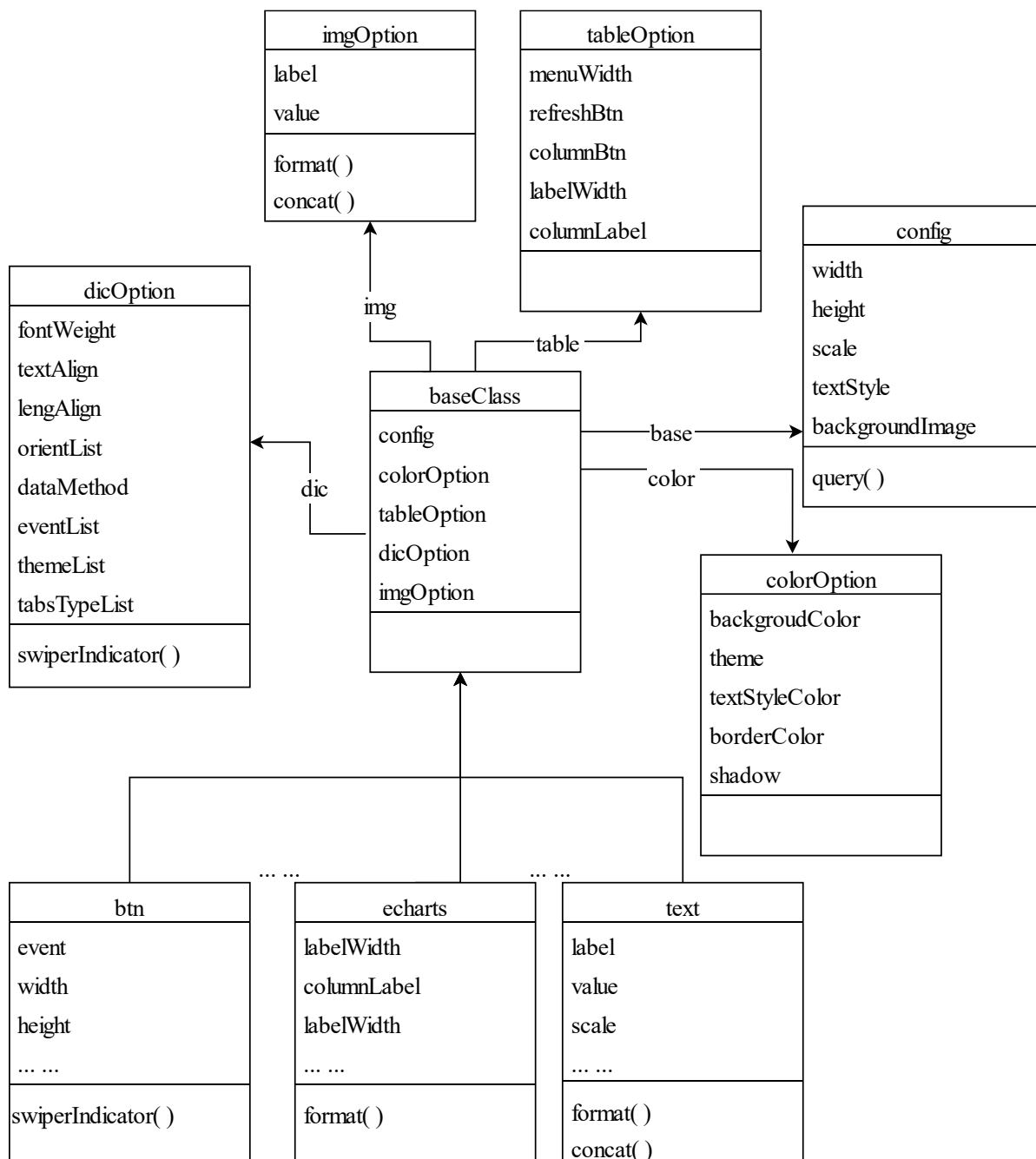


图 2.10 配置项类图

2.4.1.3 基本操作设计

对于图形元素要考虑到它的复用能力，以及操作时用户的使用习惯。为了提高系统的复用能力，增加了对图形元素的拷贝、粘贴功能。在系统的实现过程中，使用栈结构对页面活跃元素进行维护，栈结构是一个可以在一端执行接入和移除动作的特殊线性表。拷贝的过程是将图形元素压入栈的过程，而粘贴的过程则是从栈顶出栈的过程，过程如下图 2.11 所示。

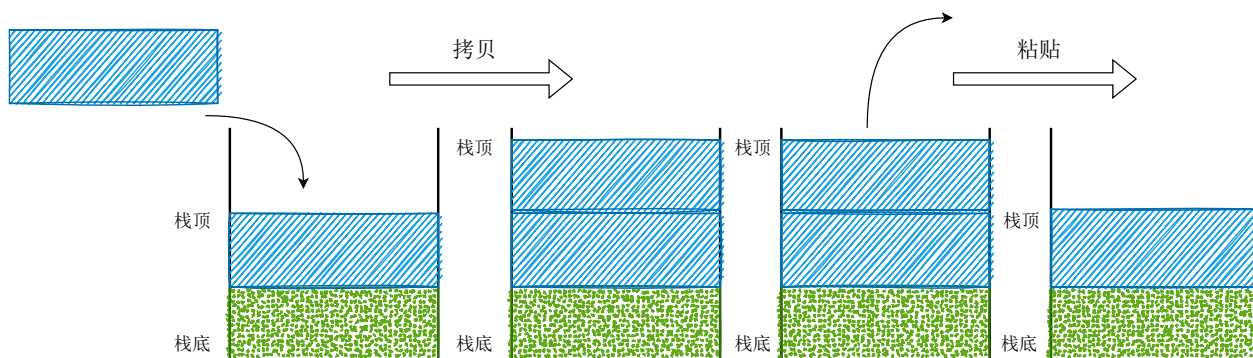


图 2.11 拷贝粘贴过程图

拷贝粘贴的算法流程可以概括为，首先寻找父类，然后将要拷贝的节点进行深拷贝，同时生成一个标识图形元素的唯一标识 UUID，最后将拷贝的节点挂载到父类上，高亮元素切换为拷贝的图形元素。

考虑系统的容错能力，需要支持一种局部删除的方法，在本系统中通过 active 变量维护选中的图形元素，active 采用数组结构，因为它可能是一个独立的元素也可能选中的是一个图层或者嵌套结构，对于一个元素来说系统实现了删除方法，对于一个图层或者嵌套结构，系统也实现了批量删除的方法。实现删除的核心思路是，首先获取选中的对象，通过图形元素的 UUID 在页面活跃元素列表中找到对应的元素，获取其父元素，从父元素维护的子元素列表中调用原型上的 splice 方法进行删除，然后初始化高亮元素。

对于计算机操作人员来说，使用快捷键操作可以提升工作效率，系统需要对常用的快捷键组合提供支持，本文通过对键盘快捷键操作的绑定实现了这一功能，支持的键码表如下表 2.2 所示，需要注意的是运行环境本身有自己的快捷键规则，为了避免与运行系统的快捷键规则冲突，在实现快捷键功能时应首先禁用运行环境的默认快捷键。

表 2.2 keyCode 键码表

按键	键码	按键	键码
Spacebar	32	Z	90
Delete	46	C	67
Control	17	V	86
BackSpace	8	...	...

具体过程为：

- (1) 在页面初始化的 created 生命周期中，执行监听函数。
- (2) 当用户按下键盘中的按键时，监听函数的反馈。
- (3) 使用 if-else 流程判断语句，根据 keyCode 键码值来判断用户按下的具体按键。
- (4) 调用相应的函数。

2.4.1.4 层级与嵌套设计

低代码的初衷是将页面进行扁平化处理,尽量减少嵌套和层次堆叠,但是在实际操作中,嵌套和堆叠是无法避免的,所以实现层级管理和嵌套也是重要的一部分,早期我们的技术方案是使用网格布局的方式,优势在于用户可以灵活简单的进行拖拽操作,但是嵌套和层次的实现上过于复杂且效果也不甚理想,经过技术调研与尝试,探索出了以自然流布局为基础的层级嵌套解决方案。在设计堆叠嵌套机制时,本文使用自然流布局进行元素的层级设置,增大了布局的灵活性。通过排序的方式改变图形元素的位置从而实现图形元素的层级与嵌套。底层使用了 VueDraggable 拖拽排序库,通过库函数将图形元素进行排序,从而达到实现嵌套的目的。

2.4.1.5 动作事件设计

本平台涉及的动作事件主要分三种,远程上传事件、鼠标事件、拖放动作。

(1) 远程上传事件

虽然系统中已经内置了丰富的图形元素,但是在一个系统中囊括所有图元也是不现实且会影响系统性能的,所以为了保证系统的可拓展性,开放了元素上传的接口,以满足一些特定领域或者特定环境下的建模需求。为方便地管理所有图形元素,将图形资源统一上传到云端平台,在项目中所提交上传的所有图形均由阿里云提供的对象存储业务(Object Storage Service,OSS)进行统一处理,并对外开发了 API 接口,图片提交后将返回图片存放在服务器上的路径。

具体流程为:

- (1) 前端向后端发送请求,获取 OSS 配置数据。
- (2) 调用前端文件上传 API,请求 OSS 提供的上传接口。
- (3) 上传完成后,OSS 返回文件存放在服务器上的路径。
- (4) 将返回的路径存放到图库列表,并进行回显。
- (5) 请求后端接口更新图库列表数据。

(2) 鼠标事件

为了使用户的操作尽可能的灵活,系统对图形元素提供了多种鼠标事件,例如鼠标左右键点击、鼠标悬停、鼠标进入元素等。本文提出了一种基于回调函数的事件绑定机制,在顶层父类中定义根据鼠标移动事件自动触发的抽象函数 Mousemove,通过 Mousemove 函数可以实时获取到鼠标的位置信息,利用位置信息获取鼠标停靠的图形元素,然后在每一个图形元素上绑定了相应的事件函数,通过鼠标事件去调用相应的回调函数。本文还由事件委托的机制设计了语义型鼠标事件绑定逻辑,在绑定时传入要绑定的事件回调函数,和鼠标事件行为的描述^[40],行为描述参考了链式调用的形式,通过“.”操作符连接,出于减少内存消耗的考虑,使用了事件委托机制将元素事件代理到外层元素上。

(3) 拖放动作

拖放是低代码平台最核心的功能，也是一种很常见的特性。一个典型的拖拽动作可以描述为：首先选定一个可拖拽的元件，并将其拖拽到一个可放置的元件内，期间鼠标不松开，之后释放鼠标。在 HTML5 标准中，规定所有元素标签都能够被拖放，HTML5 标准目前已经被大多数浏览器支持，兼容性良好。所以，项目中的拖放功能基于 HTML5 的拖放 API 来实现。

在平台的图形元素拖放过程中，由于需要得到元素的位置信息，所以本文建立了如下图 2.12 所示的坐标轴，按下鼠标后，被拖放元素的位置坐标可以用 ox 和 oy 变量描述。鼠标指针在被按下时的位置坐标则用 mx 和 my 变量保存。在拖放过程中，鼠标指针的实时位置由事件对象自定义属性 $event.mx$ 和 $event.my$ 保存。当算出这三对坐标之后，便能够动态地求出拖放元素的实时位置坐标，其 X 轴位置坐标值为 $ox+event.mx-mx$ ， Y 轴则为 $oy+event.my-my$ 。当鼠标被松开后，事件对象同时释放。

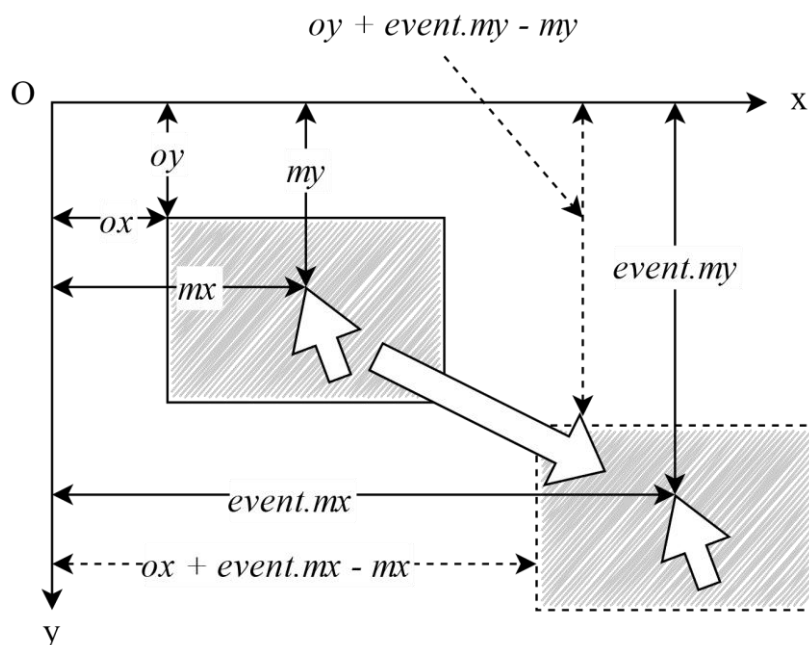


图 2.12 坐标轴与位置定义

一个完整的拖放过程可以归纳为以下步骤：

步骤 1：初始化拖放对象，将其设置为可拖放元件。

步骤 2：先取得被拖放的元件的引用指针，然后设定成绝对定位元素，再界定宽高属性，定义事件对象标准化函数，事件对象标准化函数用于处理浏览器兼容性和统计鼠标指针的 X 轴距离 mx 和 Y 轴距离 my 。

步骤 3：定义对鼠标事件的处理函数，并获得当前所拖放元素的坐标信息(ox, oy)，之后注册鼠标的移动事件处理函数和松开鼠标事件处理函数。

2.4.2 组态设备交互模块

本平台的面向对象是工业设备远程运维领域，与普通的低代码平台的区别除了提供了大

量专业领域的组件外,还提供了针对设备的多种功能,其中下发指令是重要的核心功能之一。通常物联网运维平台的结构如图 2.13 所示,首先在硬件层进行数据的采集,通过数据层对数据进行处理和过滤,最后将设备的实时数据下发到前端的展示页面^[41]。目前物联网运维系统中通过虚拟建模下发指令的研究较少,这就没有实现整个系统的闭环,在本项目中实现了基于低代码平台对设备进行指令下发的功能,实现了整个系统的逻辑闭环。

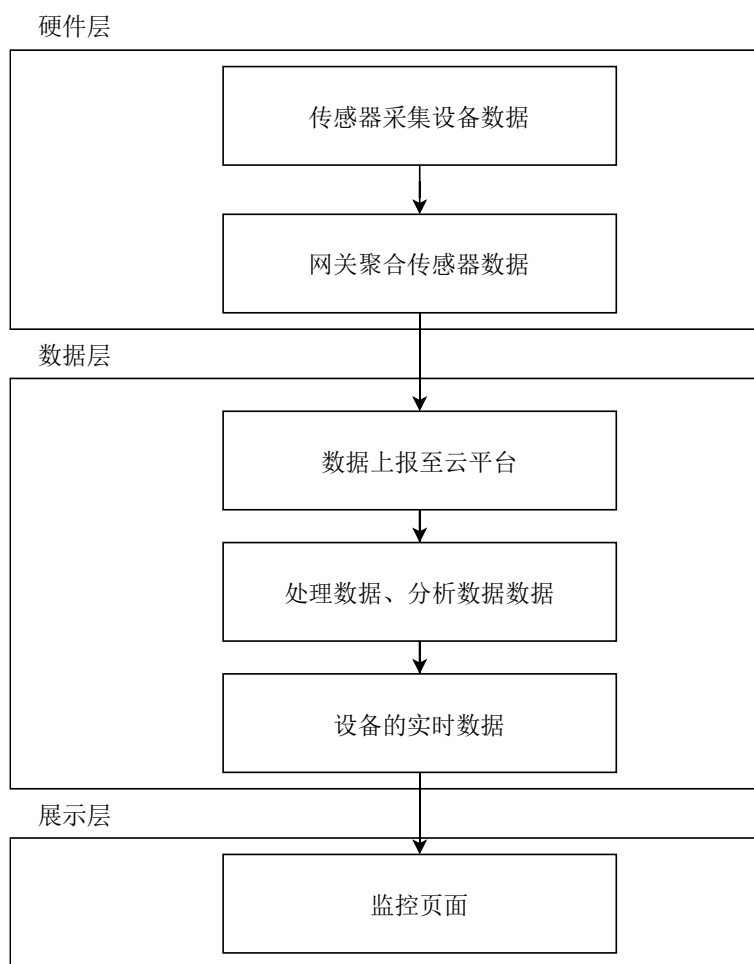


图 2.13 物联网运维流程图

指令的下发功能,底层支持是智能网关,智能网关充当了下层数据传输的通信枢纽,向上与远程上报云平台进行连接,向下与硬件物理设备连接。在本文中使用了云平台实现数据管理,智能网关上层可与云平台的数据交互指令完成上报下发操作,下层部分则集成多种不同的接口,使得本地 PLC、传感器等设备可以进行数据交互。

项目中使用的是 OneNET 物联网云平台^[42],实现了设备的接入,设备数据的上报和设备指令的下发功能,在低代码平台中,通过绑定 OneNET 的监控点,可以将虚拟建模图形元素和真实的物联网设备进行关联。同样,指令的下发可以在按钮图形元素上,关联真实的 OneNET 设备,配置点击事件。下发指令使用 HTTP 连接中的 POST 请求下发到 OneNET 提供的接口上,OneNET 提供的与设备指令下发相关的 API 列表如下表 2.3 所示,通过 OneNET

的中转，将指令下发到智能网关，网关解析指令并传输给下挂的 PLC 控制器，用于控制设备硬件。

表 2.3 OneNET 平台设备命令相关 API 说明

API 名称	请求方式	说明
下发指令	POST	平台主动传送数据信息至设备
查询命令状态	GET	用于查询某条命令的执行情况
查询命令响应	GET	设备收到命令时需要设备发送命令响应
查询设备历史记录命令	GET	用于查看设备历史数据

2.4.3 数据通信模块

系统使用了 B/S 架构，在现代 B/S 架构设计中，通常采取前后端分离开发的模式，为了保证两端协同开发，当后端依赖接口还没有开发完成时，前端可以在接口协议确定后，采用模拟数据，这样可以有效提升系统开发效率。基于这样的思想，对于工业设备来说，当底层数据支撑还没有开发完成时，为了测试页面展示效果，或者展示原型系统，也可以采用模拟数据的形式，本系统在底层内置了图表等的初始化数据，用户也可以通过配置菜单，配置模拟数据，如图 2.14 所示。



图 2.14 模拟数据配置

在工业设备运维领域，最重要的就是对设备进行监控，在专业领域页面建模接入真实设备的过程，称之为组态，低代码平台使得组态过程可以通过拖拽的形式与设备进行无缝对接，基于 OneNET 云平台，可以支持多种终端协议，轻松管理各地设备通信，平台从用户的真实应用场景出发，将设备厂商的产品或项目场景在云端进行数字化展示，一个产品或一个场景

下允许挂接多个联网设备，在低代码搭建平台只需要给对应的图形元素配置数据源即可实现设备的动态数据接入如图 2.15 所示。

图 2.15 动态数据接入

在组态页面数据通信机制方面，已经有很多的成熟的方案，冷波^[43]使用了 Ajax(Asynchronous Javascript And XML)技术进行通信，但是在组态系统上对监控数据的实时性是有较高要求的，传统方案本质上是请求/应答消息对^[44]，仍然会耗费信息资源，即时性也不强。本项目的数据模块采用了 HTTP 协议和 WebSocket 协议^[45,46]，并以浏览器作为载体实现了服务端与浏览器端的双向数据实时推送功能。

数据通信模型结构如下图 2.16 所示，在系统中使用分治的思想，对于实时性要求不高的数据使用 HTTP 协议，例如用户信息等。对于实时性要求较高的数据，例如设备状态，事件报警信息等使用 WebSocket 协议通信，实现实时通信。服务器一方面通过查询 SQL 语句与数据库进行交互；另一方面通过 WebSocket、HTTP 协议与前端显示页面进行数据通信。

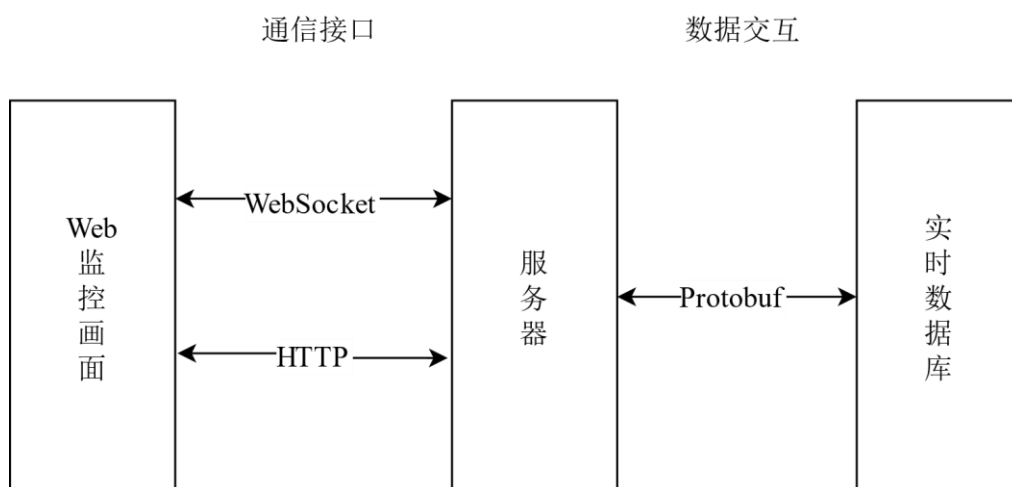


图 2.16 数据通信模型结构

2.4.4 功能设计器模块

2.4.4.1 流程设计器

流程设计器是低代码开发平台的核心底层支持引擎，用于构建流程、避免重复性劳动，这是低代码的典型应用场景，流程设计器的主要作用就是根据不同业务系统为其提供可视化、可配置的流程模型，使得业务系统可以快速完成流程的创建、部署、发起、审批。本系统流程设计器模块基于 BladX Workflow 进行开发,底层基于 Flowable 流程引擎,实现了符合 BPMN 2.0 流程标准的业务流程设计器。其体系结构如下图 2.17 所示：

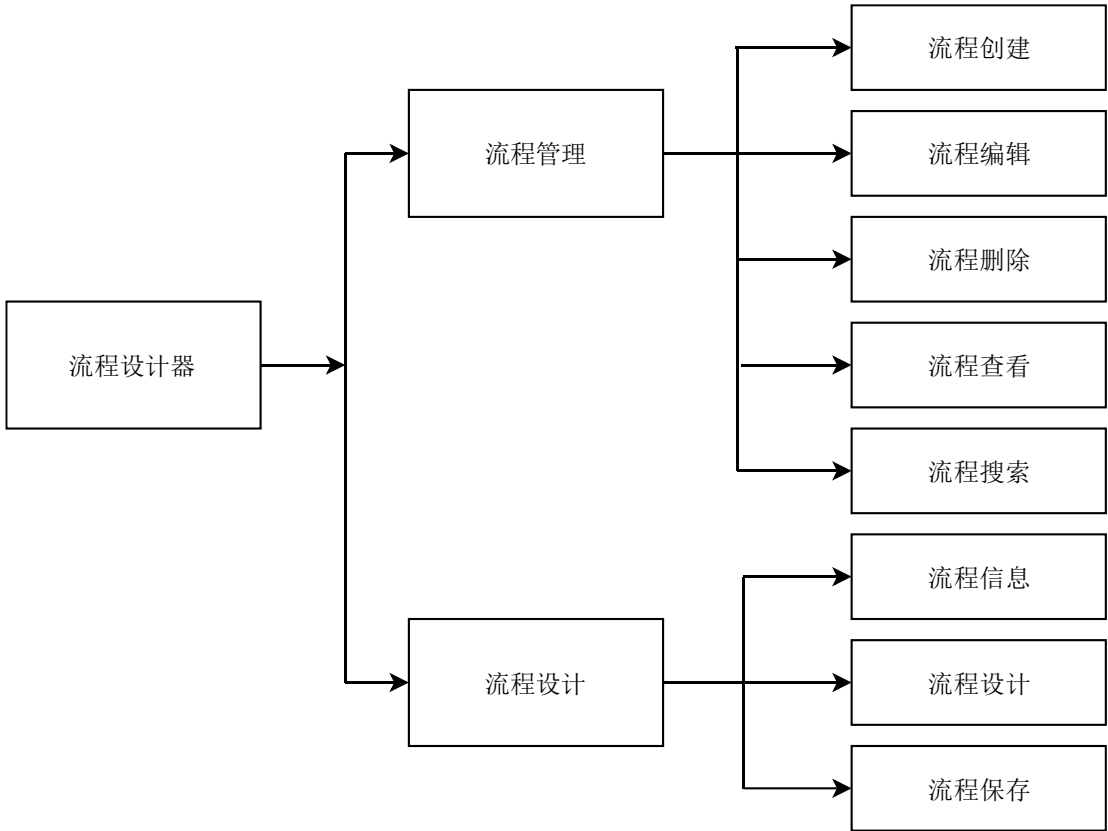


图 2.17 流程设计器体系结构图

流程设计器的使用流程为：

首先用户进入流程管理页，使用搜索功能查询是否已经具有适合的流程模型，如不具有适合的流程模型则新建流程模型，填写对应模型表单信息，完成流程的新建。

- (1) 成功新建模型后将自动跳转到模型设计器，用户可通过模型设计器可视化拖拽进行模型构建，同时设置各个节点组件的相关属性信息，完成流程模型的绘制。
- (2) 点击保存按钮进行模型保存，模型保存后路由自动跳转到流程管理界面。
- (3) 点击部署按钮对模型进行部署，完成流程设计。

具体流程模型活动图如图 2.18 所示：

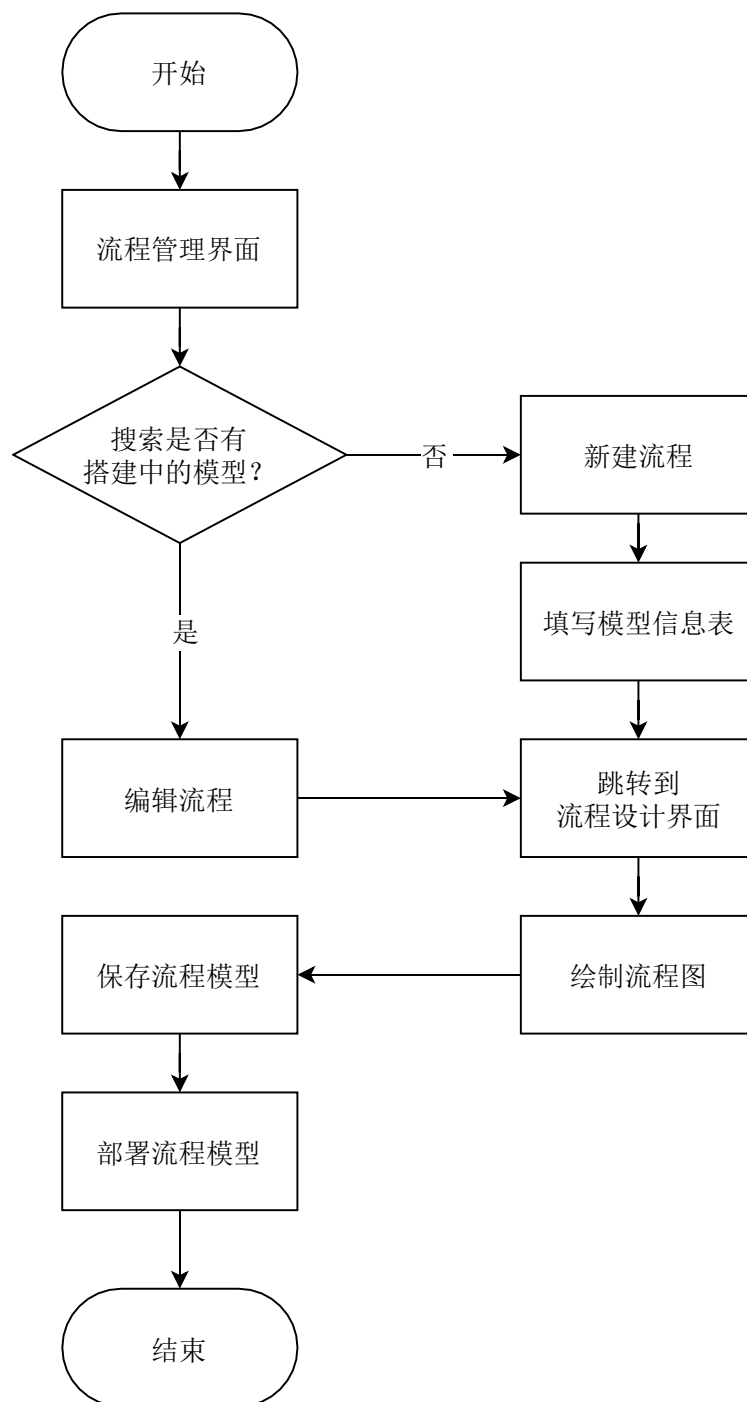


图 2.18 流程模型活动图

流程设计器分为流程的管理，流程的设计两部分。在流程管理页面，展示了保存的所有流程模型，用户可以对已有的流程模型进行编辑、删除、查看和搜索界面。本系统中流程管理功能的实现主要依托于 `ModelController` 主控制函数，通过 `ModelController` 函数调用相应功能接口，核心类图如图 2.19 所示，主函数提供了操作流程模型的具体接口方法，比如通过调用 `getList` 函数进行按条件查询、`deployModelByKey` 根据流程模型的唯一 key 值部署模型。

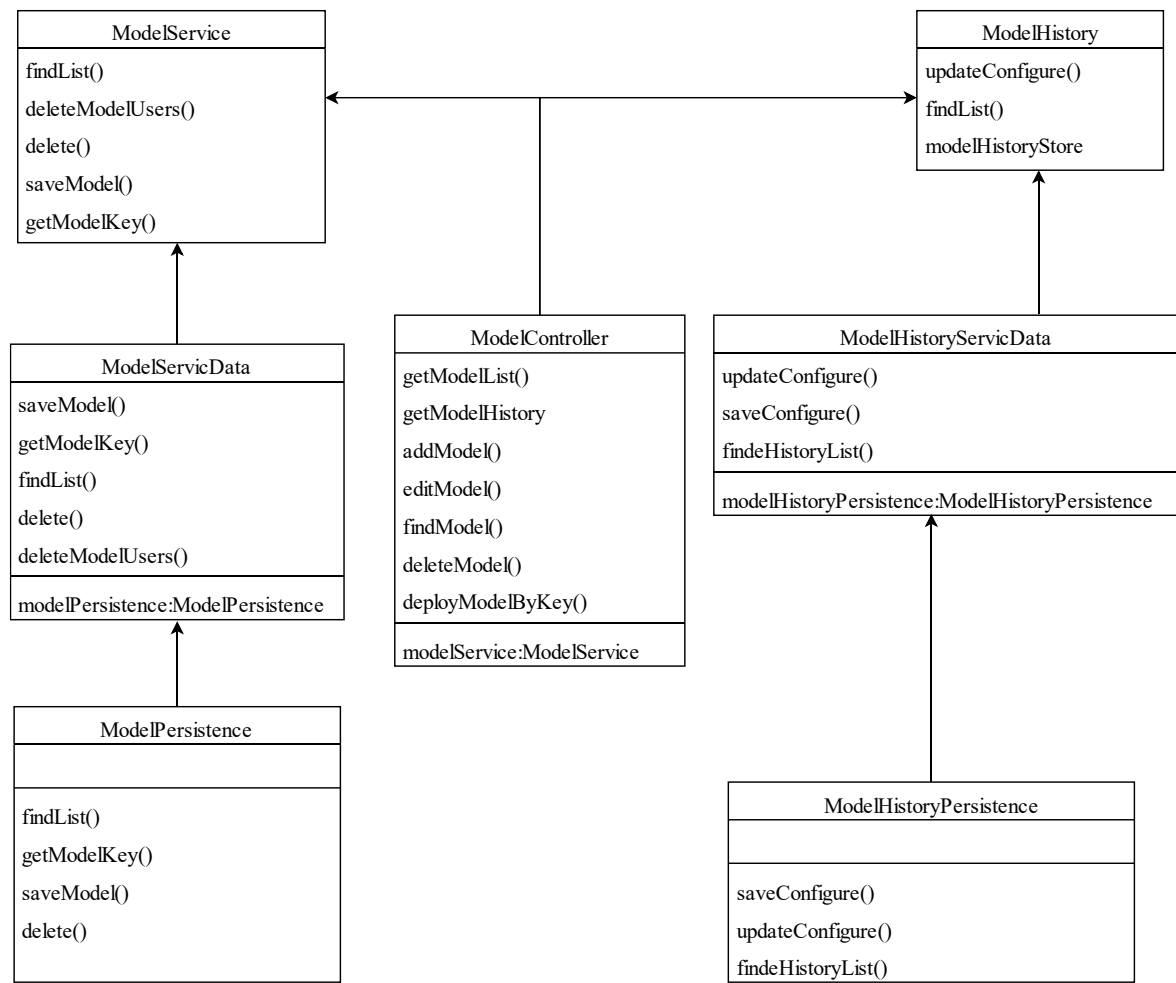


图 2.19 流程管理核心类图

在流程管理界面，可以新建流程和导入流程，在创建流程模型时要填写模型名称、模型 key 值和对模型的描述信息如图 2.20 所示。导入流程模型时流程模型的格式为 .bpmn 或 .bpmn20.xml 扩展名的 BPMN XML 定义文件，如下图 2.21 所示。

The screenshot shows a web form titled "创建一个新的业务流程模型" (Create a new business process model). It includes a text input for "模型名称" (Model Name), another for "模型key" (Model Key), and a larger text area for "描述" (Description). At the bottom, there are two buttons: "取消" (Cancel) and "创建新模型" (Create New Model).

图 2.20 创建流程模型操作

The screenshot shows a web form titled "导入一个流程模型" (Import a process model). It features a dashed box for dragging and dropping a file, with the instruction "请浏览或拖拽 .bpmn 或 .bpmn20.xml 扩展名的 BPMN XML 定义文件" (Please browse or drag and drop BPMN XML definition files with .bpmn or .bpmn20.xml extensions). Below the box is a "选择文件" (Select File) button and a "取消上传" (Cancel Upload) button. A "x 取消" (Cancel) link is in the top right corner.

图 2.21 导入流程模型操作

流程设计器的另一部分则是流程设计器，界面如图 2.22 所示，本系统的流程设计功能主

要使用 BladX Workflow 技术进行开发，底层基于 Flowable 流程引擎。本流程设计器不仅具有重量轻的优点，而且具有可扩展性，便捷的二次编辑功能降低了开发难度，同时渲染速度也具有优势。



图 2.22 流程设计器界面

流程设计器的整个配置都是在 flowConfig.js 文件中进行定义，通过设置 flowConfig.js 可以配置流程扩展方法和流程节点扩展方法、定义监听反馈、操作方式等。流程设计界面主要提供了流程设计、流程信息修改、流程信息保存、流程模型保存、流程模型取消等功能。流程设计功能使用可视化拖拽建模的方式，用户通过拖拽左侧工具栏中的组件，然后配置流程节点信息，可以快速搭建并展示流程模型，大大缩减了开发时间和工作量。

2.4.4.2 表单设计器

同流程设计器一样，表单设计器同样包含表单管理界面和表单设计界面两部分这里不再赘述，表单管理界面提供新建表单、操作配置、弹框配置、按钮配置、编辑表单，删除表单功能。用户从表单管理界面中的新增表单或者编辑表单按钮进入表单设计界面如图 2.23 所示。

本系统的表单设计功能基于 vue-form-making 进行开发，内置了常用的表单组件，数据绑定等都通过可视化配置的方式，使用者无需具有软件编程经验，即可快速搭建出合适的表单。其底层使用了饿了么前端开发团队推出的 UI 框架 Element-UI。此外，本设计器内置了 i18n 国际化解决方案，通过可视化配置的方式搭建表单，一键预览，一键生成导入导出配置，支持表单验证，所有配置保存在一个 JSON 文件中，导入导出的格式均使用 JSON 数据格式，并提供自定义组件，使得用户搭建表单时更加灵活。

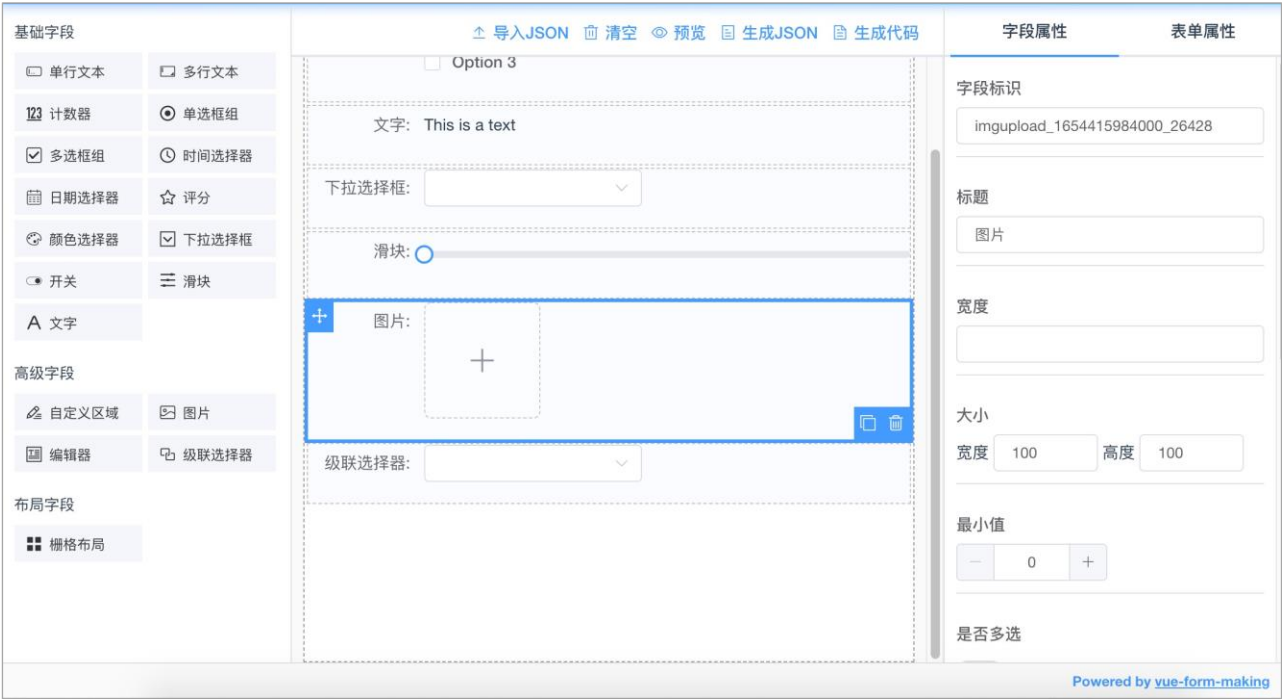


图 2.23 表单设计器界面

2.4.4.3 表格设计器

同表单设计器与流程设计器一样，表格设计器同样包含表格管理界面和表格设计界面两部分，表格管理界面提供新建表格、搜索表格、设计表格、预览表格、批量删除表格功能。用户从表格管理界面的新增表格和设计表格按钮进入表格设计界面。不同于表单设计器，表格设计器不是 UI 组件的组合，而是报表的形式如图 2.24 所示。本系统中表格设计功能基于 UReport2 引擎进行开发，它是一款高效的报表引擎，通过对单元格的迭代，可以迅速地生成多种形式的复杂中式报表。

在表格设计器中，单元格之间是存在相互依赖关系的，任意一个单元格都可以设定它的左、上父格。单元格的父格子不是必须设置项，在缺省情况下，自动判别其上父格是与它同处一列且相距最近的上方单元格，而其左父格就是与它同处一行且距离最近的左侧的单元格。为了更灵活的处理表格，实现了手动变更某一特定单元格的左、上父格功能。此外，单元格具有自动展开的功能，当数据量增多超出格子默认大小时自动展开，对于父格子而言，展开时会连带内部的子格子一起展开。如果子格与父格绑定的数据集同属一个，那么子格将受到父格数据的约束。同时，子格绑定的数据集中对应字段的数据在展开时，也会带动其中的子格一起展开。这时，如果当前的子格子的父格和其同处于一行或者一列，就会导致父格子自适应的被拉大。利用上述的一套自适应扩充的方式对单元格进行迭代，就可以制作出各种繁琐报表。

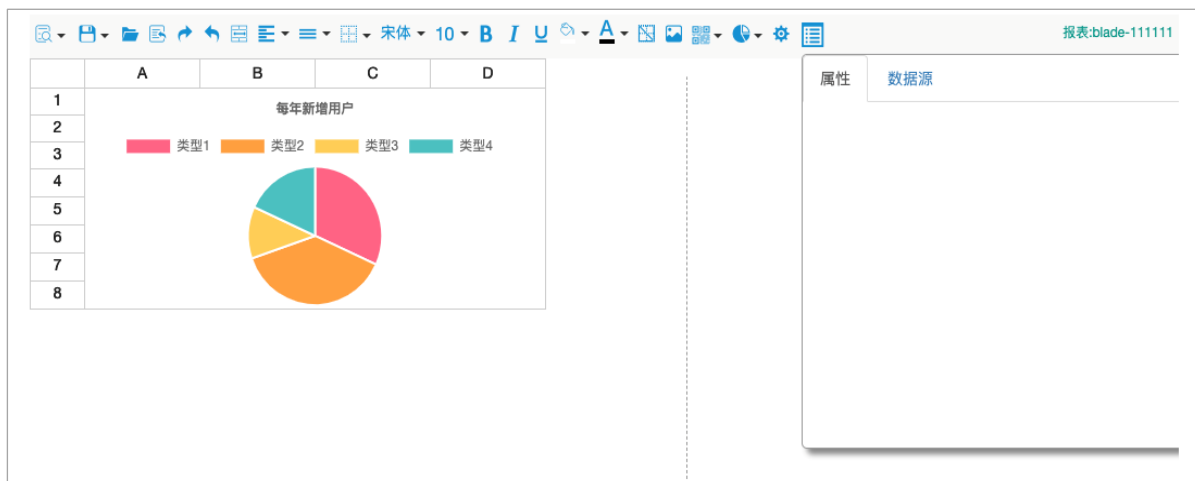


图 2.24 表格设计界面

2.5 系统测试

2.5.1 搭建测试环境

本次测试将使用三台同时提供了 Kubernetes 集群和 Istio 服务网格环境的服务器，其中两台 Centos 8.4 操作系统的云服务器，用于集群 Node 节点，具体配置如下表 2.4，完成服务部署后，使用市场主流浏览器，对界面进行访问，以测试系统的兼容性。在 Istio 系统中部署了 Promethues、Jaeger、Servicegraph、Grafana 等组件，测试服务治理功能。

表 2.4 服务器配置和软件环境

设备与软件	处理器型号	内存	操作系统	软件环境
云服务器	Intel Xeon Cooper Lake(3.4GHz/3.8Ghz)	8G	CentOS 8.4	Kubernetes1.20.1
				Istio 1.10.2
				MySQL 8.0.28
云服务器	Intel Xeon Cooper Lake(3.4GHz/3.8Ghz)	8G	CentOS 8.4	Kubernetes1.20.1
				Istio 1.10.2
				MySQL 8.0.28
云服务器	Intel Xeon Cooper Lake(3.4GHz/3.8Ghz)	4G	CentOS 8.0	Kubernetes1.20.1
				Istio 1.10.2
				MySQL 8.0.28
浏览器客户端	2.6 GHz Intel Core i7	8G	macOS 版本	Chrome 94.0.992
			10.15.7	Safari 13.1.3
浏览器客户端	3.3 GHz Interl Core i5	8G	Window 7	Internet Explorer 11
				Mozilla Firfox97.0.1

2.5.2 测试内容与结果

(1) 应用间交互测试

本系统微前端部分使用组合式应用路由方案设计，该方案设计的核心内容是“主从”思路，将巨石应用拆分为各个微小应用，包括一个基座应用和若干子应用。测试重点在于基座应用与子应用，子应用与子应用的交互，测试项如下表 2.5 所示。

表 2.5 应用间交互测试表

测试项/操作	预期结果	测试结果
打包部署	微件成功打成互不侵入的独立包	通过
加载基座应用	容器应用成功部署在应用内部 页面成功加载并显示 加载时长<200ms 页面按钮可操作 后台无报错提示 页面流畅无卡顿	通过
加载子应用	子应用成功注册、加载 白屏等待时长<1s 页面按钮均可操作 后台无报错提示 页面流畅无卡顿	通过
子应用路由跳转	前端成功构建路由表 各子应用可成功从主应用跳转，也可互相跳转 后台无报错信息 不在路由列表的页面成功路由到 400/404 页面	通过
子应用加载公共基建	共享的公共基建可以为所以子应用所应用	通过
子应用通信交互	能成功进行子应用与子应用间通信 能成功进行子应用与父应用通信 能成功进行网络通信	通过
应用间隔离	应用代码完全隔离互不干扰	通过

(2) 兼容性测试

对于客户端界面，由于历史原因各个浏览器应用的内核及所支持的页面编程语言标准有所不同，显示效果可能无法达到理想效果。通常情况下，系统的兼容性需求是，不管是使用何种浏览器访问本网站或本系统，均应确保其显示效果一致。兼容性问题也是决定整个系统

性能的关键问题，同时在测试阶段也一定要考虑。此次测评，使用了市面上的主流浏览器对系统进行兼容性测评如下图 2.25 所示。由结果显示，兼容性效果均达到预期。

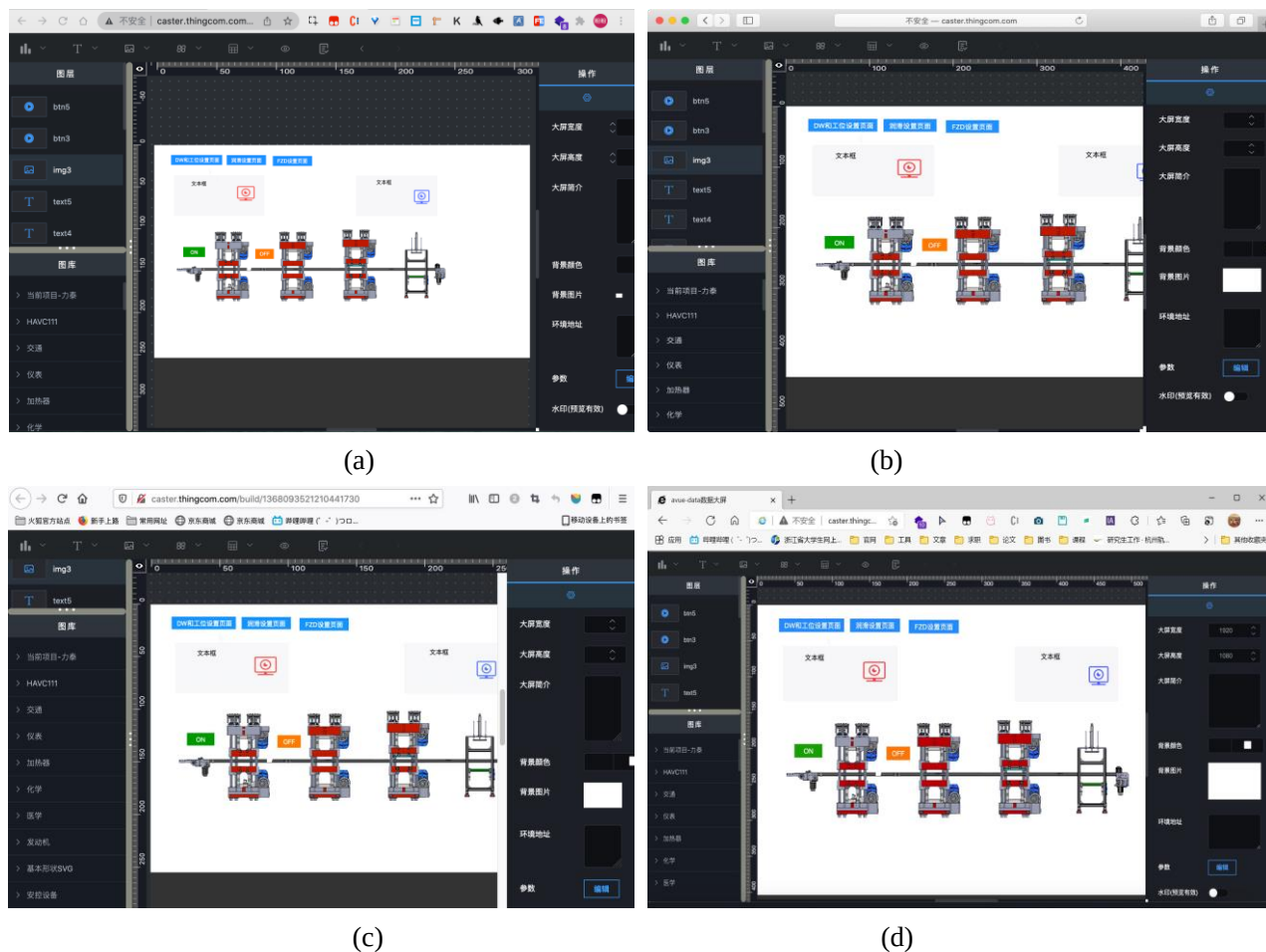


图 2.25 兼容性测试结果

(a)Chrome 浏览器 (b) Safari 浏览器
(c) Mozilla Firefox 浏览器 (d)Internet Explorer11 浏览器

(3) 页面性能测试

页面性能是决定用户体验的重要因素，本系统采用 Google 的 RAIL 模型评估页面性能，RAIL 是一项可量化的、面向用户的性能测试标准^[47]，将从 Response（响应）、Idle（浏览器空置状态）、Load（加载）和 Animation（动画）四个方面去评估一个页面的性能，性能指标如下表 2.6 所示：

表 2.6 RAIL 性能指标

名称	描述	指标
Response	用户从输入到响应的时长	$\leq 100\text{ms}$
Animation	每一帧的渲染时长	$\leq 16\text{ms}$
Idle	最大化主线程的空闲时间	$< 50\text{ms}$
Load	传输内容到页面可交互的时间	$< 5\text{s}$

本次测试使用 LightHouse 性能评估工具对本系统进行评估，为确保测试数据有效性，分别在 MacOS 操作系统和 Windows 操作系统测试 100 次并取平均值，统计得到如下图 2.26 所示测试结果。图中横轴代表测试次数，纵轴代表该次测试中消耗的平均时间。由测试结果可知，系统页面性能满足 RAIL 性能指标，页面性能良好。

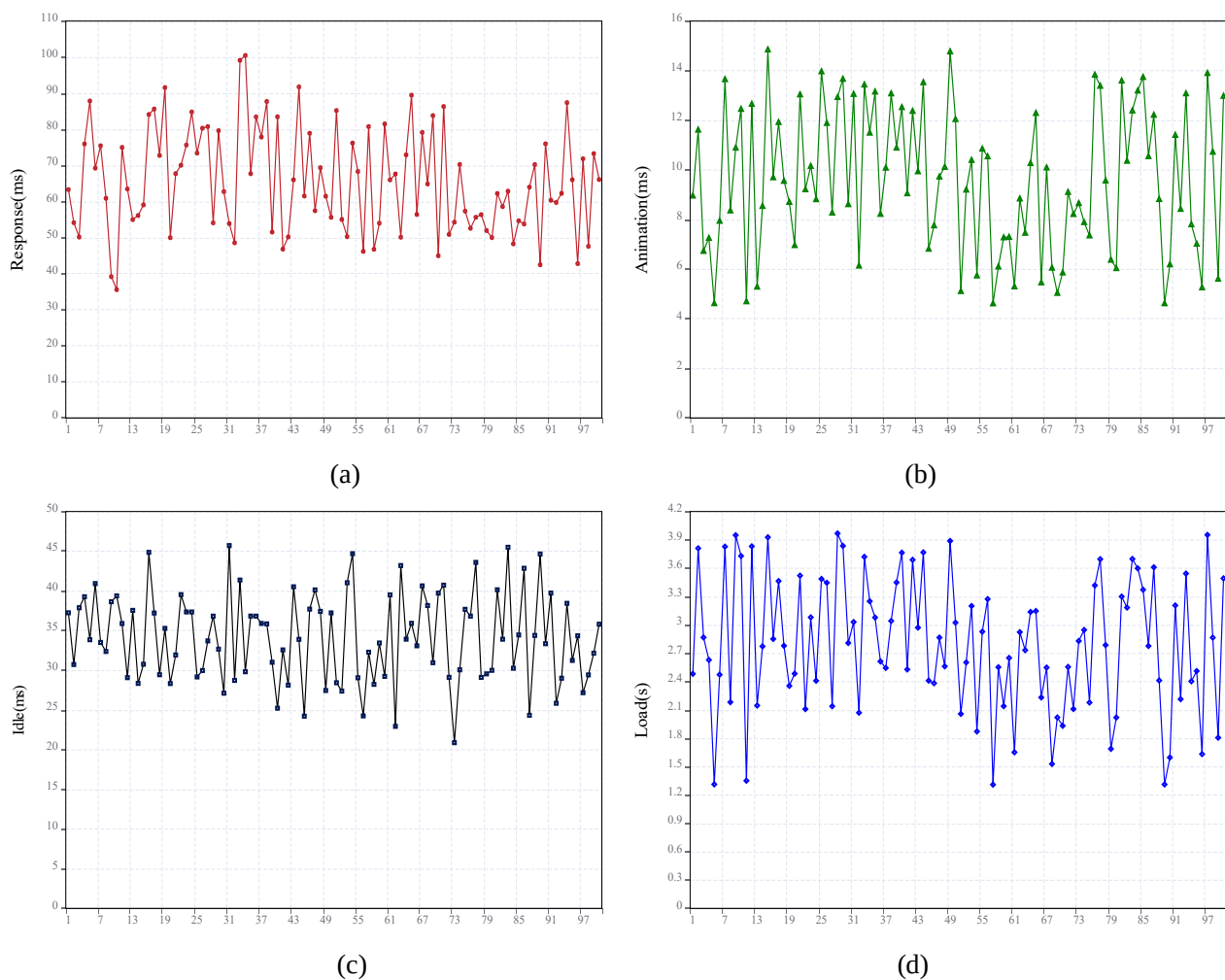


图 2.26 页面性能测试结果

(a)Response 测试结果 (b)Animation 测试结果
(c)Idle 测试结果 (d)Load 测试结果

2.6 本章小结

本章针对面向工业设备远程运维的低代码平台，首先对项目需求进行总体剖析，提出了一套低代码平台整体设计方案，在架构模型上提出微前端-微服务模型，解决了目前单体应用体积大且扩展性差的问题。然后就平台的核心模块给出了详细的设计方案，在本章最后对系统进行测试，证明其有效性和稳定性。

本文设计的面向工业设备远程运维的低代码平台有如下优势：

(1) 更合理的软件架构，使用微前端-微服务架构将应用程序分割成小的独立的部分，每部分都可以独立开发、测试和部署，增强系统的可维护性和可拓展性。

(2) 简洁的拖拽式开发，允许业务用户高度灵活性的创建自己的应用程序，从而有效降低企业应用开发成本，快速完成项目交付。面向工业设备远程运维领域，整合物联网方案，内置丰富行业图形元素和行业模板，从设备数据导入、数据监控、异常上报、指令下发实现了完整业务闭环。

(3) 完善的功能支撑，平台集成了流程引擎、表单引擎、报表引擎三大引擎，实现了界面化流程的建模，使得流程设计变得简单和可操作，通过简易配置即可自动生成各种类型的报表、表格。

第三章 基于混杂 CD 算法和改进 ACO 的选区方案

依据前文所述，在建设低代码平台时，低代码编辑器是核心模块。在低代码编辑器中搭建全过程都涉及拖拽选区交互。对于工业设备来说，具有复杂多样且数量大的特点，所以选区的精度和效率直接决定着面向工业设备远程运维的低代码平台的使用体验与系统性能。本章针对目前的选区方案精确性不足，大数据量操作页面卡顿的问题，创新性的将计算机物理学中的碰撞检测(Collision detection, CD)算法和仿生学蚁群算法(Ant Colony Optimization, ACO)应用到低代码平台中，提出基于混杂 CD 算法和改进 ACO 算法的选区方案。首先通过改进混杂 CD 算法生成 BVH 树，然后再使用改进 ACO 算法进行选区相交检测，最后设计对比实验测试改进方向的有效性和方案整体性能。

3.1 低代码编辑器选区方案分析

选区在涉及拖拽的交互场景下应用的十分普遍，也是十分基础的需求。即使用者可以通过鼠标勾画出一个矩形选区，选中选区内的图形，方便之后的移动、变形、删除等操作。所以如何判断图形元素是否在选区内是实现选区方案的核心，业界常用的方案有两种：基于矩形包含算法的选区方、基于 AABB 包围盒算法的选区方案。

3.1.1 基于矩形包含算法的选区方案

基于矩形包含算法的选区方案核心思路是判断选区和图形元素是否是包含关系，当选区完全包含图形元素时，图形元素被选中如下图 3.1 所示。只要图形元素区域矩形的所有点都在框选区矩形的范围内，就满足包含关系。图形元素使用一个轴向对齐的外接矩形近似代替。

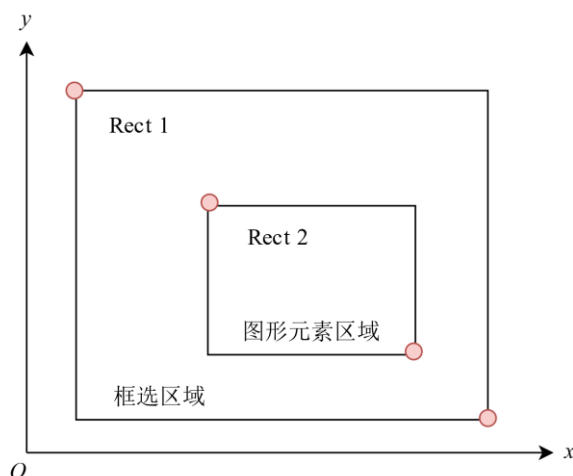


图 3.1 基于矩形包含算法的选区方案示意图

算法核心思路: 获取框选区域 Rect1 与图形元素区域 Rect2 的左上角与右下角坐标信息，

找出极值点 x_{min} , x_{max} , y_{min} , y_{max} , 判断时只需要对比:

$$\begin{aligned} R_1 x_{min} &< R_2 x_{min} \\ R_1 y_{min} &< R_2 y_{min} \\ R_1 x_{max} &> R_2 x_{max} \\ R_1 y_{max} &> R_2 y_{max} \end{aligned} \quad (3.1)$$

如满足式(3.1)则确定选中图形元素。这种方案带来的问题很明显, 使用者必须完全选中图形, 使用过程中用户体验不佳, 当图形元素大小超出画布一屏时会使得选区操作变得极其困难, 并且对图形元素的位置信息没有做缓存处理, 大数据量卡顿明显。

3.1.2 基于 AABB 包围盒算法的选区方案

基于 AABB 包围盒算法的选区方案。这种方案是目前低代码平台使用最为广泛的, 相较于基于矩形包含算法的选区方案, 其引入了计算机图形学中包围盒的概念。

在图形学中包围盒的思想是用简单几何特征的图形代替复杂几何特征的图形, 在判断碰撞时只判断包围盒是否碰撞^[48]。常用的包围盒模型有: 圆形包围盒(Sphere)^[49], 轴向对齐包围盒(Axis-aligned bounding box, AABB)^[50], 方向包围盒(Oriented bounding box, OBB)^[51], 几种包围盒模型如下图 3.2 所示。

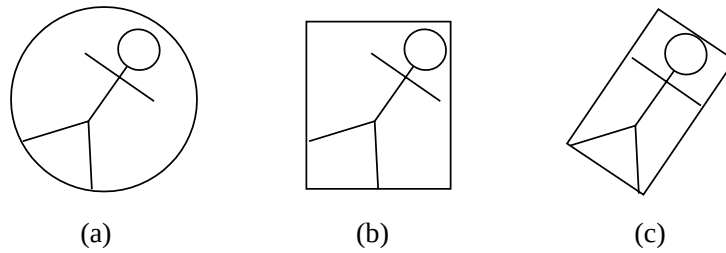


图 3.2 常用包围盒类型

(a)圆形包围盒(Sphere) (b)轴向包围盒(AABB) (c)方向包围盒(OBB)

在该方案中主要思路就是用一个形状规则的 AABB 包围盒去包裹图形元素, 然后判断框选区域和图形元素的 AABB 包围盒是否相交, 相交时选中图形元素如下图 3.3 所示。

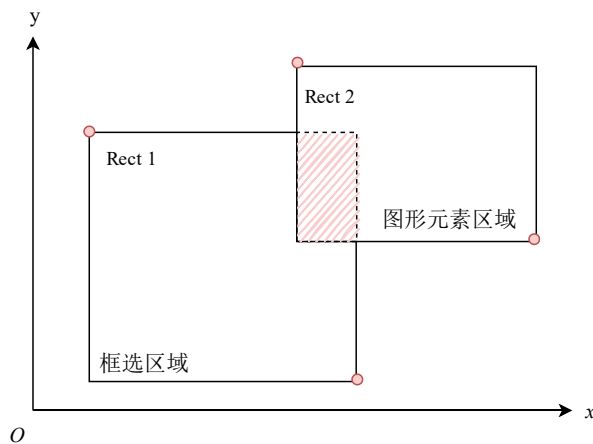


图 3.3 基于 AABB 包围盒算法的选区方案示意图

算法核心思路：分别做 Rect1 与 Rect2 在 XY 轴方向的投影，记 Rect1 在 X 轴方向最小值为 $R_1x_{\min}$ ，最大值为 $R_1x_{\max}$ ，在 Y 轴方向最小值 $R_1y_{\min}$ ，最大值 $R_1y_{\max}$ ，Rect2 同理。两区域重叠区域如图 3.3 所示，若 Rect1 与 Rect2 在两个坐标轴的投影发生重叠则判断选中，在二维空间中只需要满足：

$$\begin{aligned} R_1x_{\min} &> R_2x_{\max} \\ R_1y_{\min} &> R_2y_{\max} \\ R_2y_{\min} &> R_1y_{\max} \\ R_2x_{\min} &> R_1x_{\max} \end{aligned} \quad (3.2)$$

若满足式(3.2)则确定选中该图形元素。这种方案解决了图形元素超出一屏框选困难的问题，但是同时也带来了新的问题，在当前低代码平台中，元素多为形状规则的表单块与卡片，所以问题没有凸显。但本文设计的是面向工业设备的低代码开发平台，其中涉及很多不规则形状的设备零件，从而使用这种方案无法保证选区精度，很容易误判，即选区虽然与图形元素的 AABB 包围盒相交但由于形状不规则导致 AABB 包围盒不够紧密，所以并没有真正的框中图形本身，这很容易使得用户在操作过程中产生困惑，降低使用感。此外，同样对图形元素的位置信息都没有做缓存处理，存在大数据量卡顿问题。

3.1.3 优化方向分析

针对以上两种方案的弊端，本文在设计选区方案时，优化设计方向主要为以下两点：

(1) 构建 BVH 树提升选区精度

目前低代码平台中使用的普遍为单一 AABB 包围盒算法，这种算法虽然简单高效，但对图形元素包围的紧密性太差，这是导致精度不足的直接原因。本文认为单一使用某一种包围盒包裹图形元素达不到选区方案的精度需求，所以使用构建 BVH 树的方式包裹图形元素。通过三层分层的方式供后续相交检测进行预处理、粗检测、精细检测，这种分层的方式在低代码领域鲜少得到应用，究其原因，在低代码平台中运行环境依赖浏览器，构建 BVH 树势必增大了计算复杂度和性能开销，所以 BVH 树技术不能直接应用于低代码平台中。本文在混杂 CD 算法的基础上对其作出了改进，在分层时不固定使用某一种包围盒类型而是通过自动判别的方式，降低复杂度和算法耗时。

(2) 引入 ACO 算法提高相交检测效率

选区过程本质上是判断鼠标框选区域与图形元素是否相交，采用 BVH 树的方式在相交检测时，势必也要分层检测，这无疑降低了检测速度。为了解决这一问题，在本文相交检测中加入了 ACO 优化算法，使得相交检测问题转化为在二维空间中特征对搜索最优解的问题，从而提高了检测速度。但是，引入 ACO 的 BVH 树相交检测算法在提高检测效率的同时，也会出现过早陷入局部最优和收敛速度慢等问题。于是在本文中对 ACO 算法作出了改进，通过动态设置挥发因子的方式提升了算法性能。

3.2 基于混杂 CD 算法和改进 ACO 的选区设计与实现

3.2.1 总体方案设计

由 3.1 小节的方案分析，本文的选区方案分为三阶段：预处理阶段、粗检测阶段、精细检测阶段。预处理阶段为图形元素构建简单 AABB 包围盒；粗检测阶段通过检测预处理阶段构建的简单 AABB 包围盒是否与选区相交，过滤掉离选区距离较远的图形元素；精细检测阶段通过混杂 CD 算法构建 BVH 树，然后通过改进 ACO 求得相交区域中的子节点内特征对与选区内特征对的最优距离，进而判断其与选区是否相交。整体流程如图 3.4 所示。

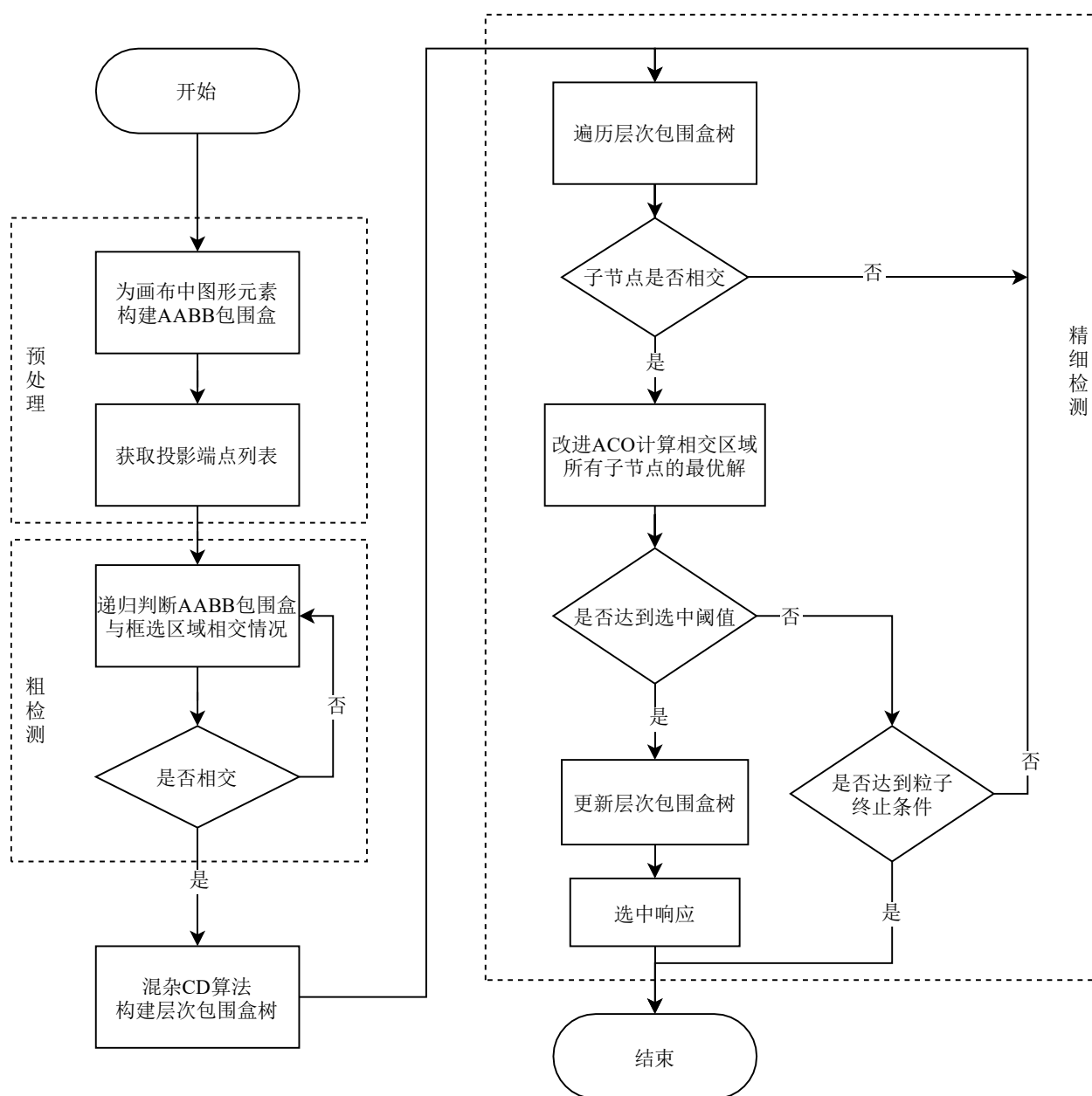


图 3.4 整体流程设计图

3.2.2 基于混杂 CD 算法的 BVH 树

BVH 树是一种结合了层次树和包围盒技术的概念。算法的基本思想是，通过 BVH 树将图形元素按层次划分成微小部分，并在此基础上，不断地构造出小部分的包围盒，再用树形层次结构将图形元素的各个小部分进行连接如图 3.5 所示^[52]。分层的树形结构中的根节点被用于存放图形元素最外面的包围盒，图形元素分割成的微小部分的包围盒被子节点存放起来，然后子节点继续递归划分，直至达到一定的终止条件。

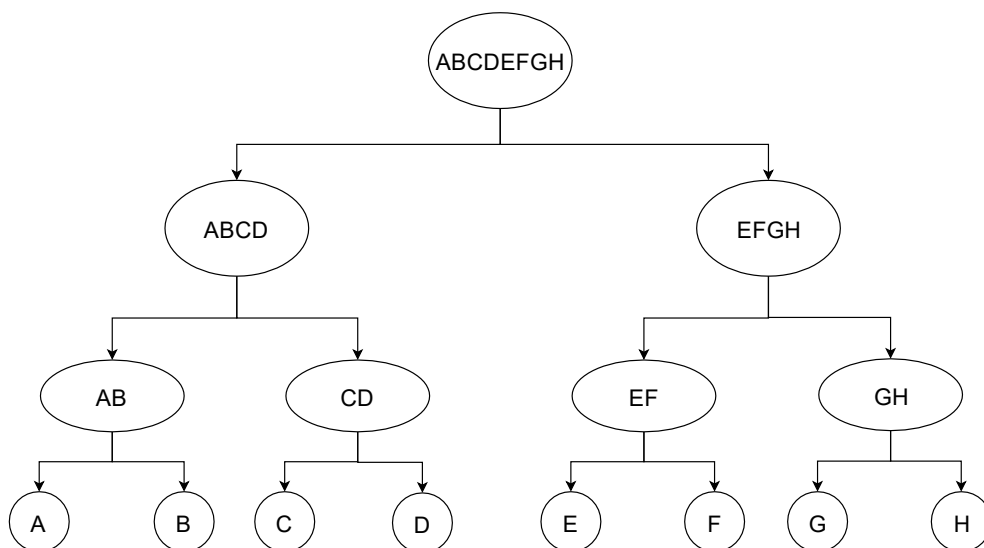


图 3.5 树形层次结构图

3.2.2.1 混杂 CD 算法及其改进设计

混杂 CD 算法的核心原理是使用多种包围盒技术构造 BVH 树，目前这种方式多用于计算机模拟与计算机游戏的碰撞检测中，结合本文低代码选区过程，本质上属于碰撞检测的范畴。李健^[53]等人固定使用圆形包围盒结合 AABB 和 OBB 包围盒，构建了两层包围盒层次树应用于农业信息化领域；李玉虎^[54]等人固定使用 AABB 和 OBB 包围盒构建了四层包围盒层次树应用于材料力学；Baiqiang G^[55]等人固定使用 AABB 和 OBB 包围盒构建了三层包围盒层次树应用于虚拟现实领域。

在本文中应用领域为低代码研发，所以受到浏览器的单线程限制。由前文所述越紧密的包围盒其计算越复杂，本文认为如果使用固定的包围盒类型构建层次树，则对性能有较大影响，且失去了一定灵活性。所以本文设计了自动判别包围盒方法。将通过 BVH 树划分的图形元素微小部分，构建其包围盒时采用智能判别方法，设定安全角，当包裹部分方向轴与标准轴夹角小于安全角时构建 AABB 包围盒，大于安全角时使用 OBB 包围盒。

此外，得益于浏览器便捷的存储接口，运用面向对象的思想本文对 BVH 树的更新过程进行了优化，由于所有的图形元素变换操作都为旋转、平移，缩放，三种或者三种的组合，所以想到用 cache 对图形变换信息存储到 Web 数据库优化更新树过程。

因此，使用混杂 CD 算法构建、遍历，更新 BVH 树的整体流程如图 3.6 所示。

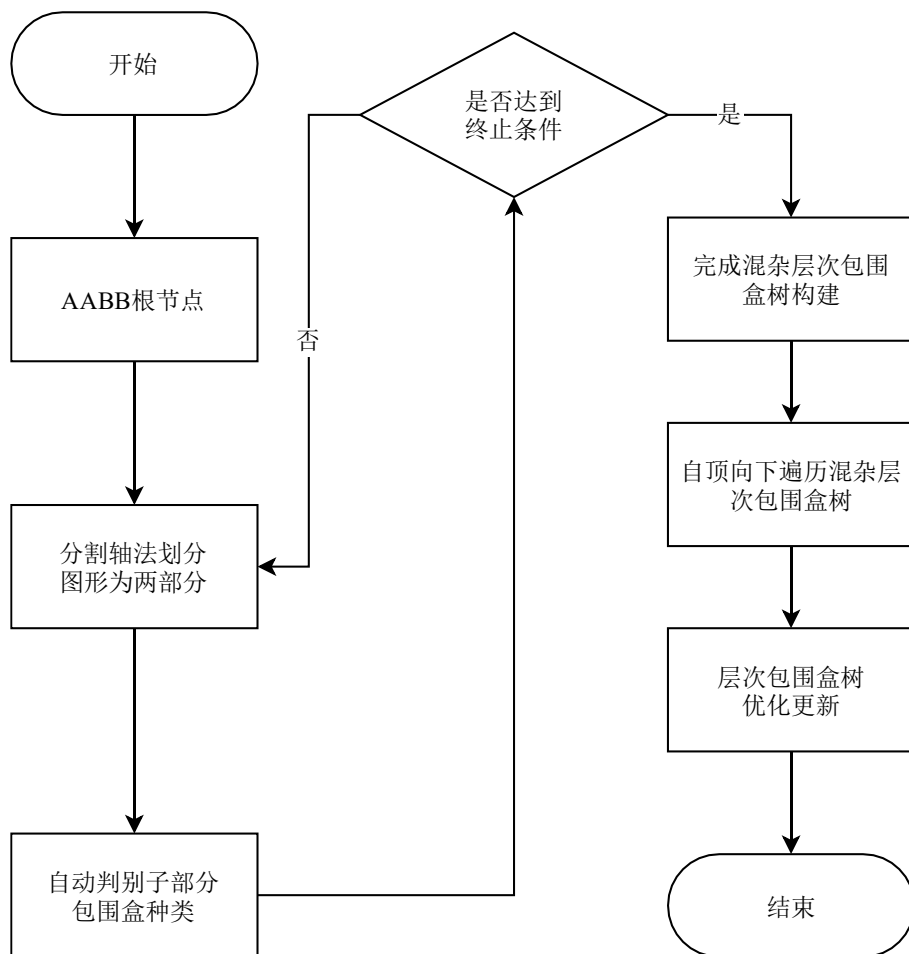


图 3.6 改进混杂 CD 算法整体流程图

下面将对流程中的两个改进点：自动判别包围盒方法和 BVH 树更新优化进行实现。

3.2.2.2 自动判别包围盒方法(ADBB)实现

低代码平台框选过程如下图 3.7 所示。可见对于鼠标绘制的选区天然形成一个轴对齐矩形。

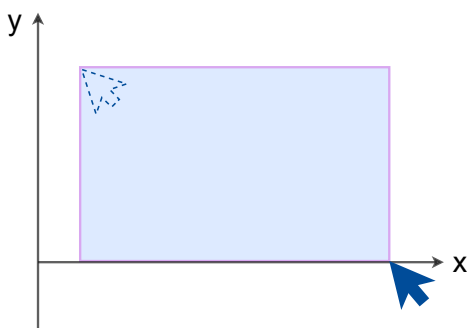


图 3.7 低代码平台框选过程示意图

所以，本文对于框选对象固定使用单一 AABB 包围盒，通过计算鼠标的起点与终点可以快速构建出包围区域，进而确定 $x_{\min}$ ， $x_{\max}$ ， $y_{\min}$ ， $y_{\max}$ 得到 AABB 包围盒的矩形区域：

$$R = \{(x, y) | x_{\min} \leq x \leq x_{\max}, y_{\min} \leq y \leq y_{\max}\} \quad (3.3)$$

其中 $(x_{\max}, y_{\max})$, $(x_{\min}, y_{\min})$ 表示选区在各个轴上的投影最大值和最小值。

本文认为, 对于低代码平台画布中的图形元素, 单纯固定一种包围盒方式并不能达到有效包围, 原因在于 OBB 相对于 AABB, 增加了对图形元素的旋转角度描述, 从而提升紧密性, 然而 OBB 计算中涉及相对复杂的数学计算, 所以耗时较长。

针对以上问题, 本文设计了一种自动判别包围盒方法以下记做 ADBB(Automatic Discrimination Bounding Box), 构建过程如图 3.8 所示, 通过设置安全角的方式程序自动选用合适的包围盒, 当图形元素旋转角度小于安全角, 则使用 AABB 包围盒, 当图形元素旋转角度大于安全角, 则使用 OBB 包围盒。

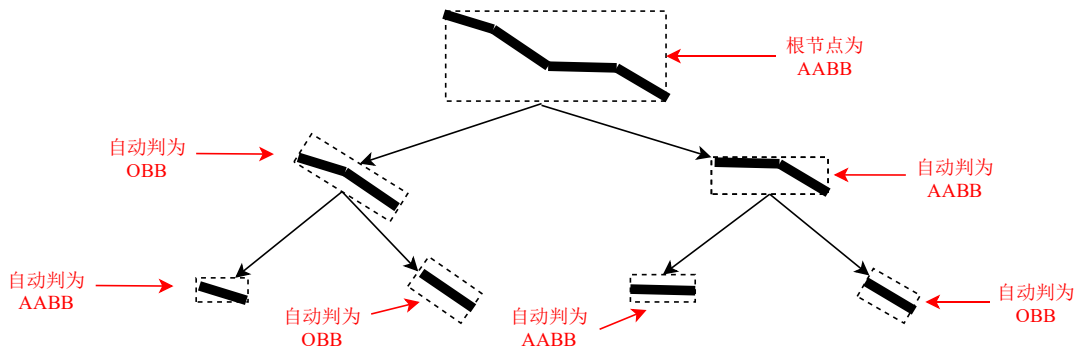


图 3.8 ADBB 构建包围盒树的过程

具体步骤如下:

步骤 1: 预设安全角 ρ ;

步骤 2: 求出图形主方向;

首先设 X, Y 为随机变量, C 为常数, K 为正整数, 图形二阶矩为:

$$\begin{cases} M_{20} = \sum_i \sum_j i^2 \cdot V(i, j) \\ M_{02} = \sum_i \sum_j j^2 \cdot V(i, j) \\ M_{11} = \sum_i \sum_j i \cdot j \cdot V(i, j) \end{cases} \quad (3.4)$$

其中 $V(i, j)$ 表示图像在 (i, j) 点上的灰度值。

通过二阶矩求图形主方向:

$$\theta = \frac{1}{2} \arctan \left(\frac{2b}{a-c} \right) \quad (3.5)$$

$$a = \frac{M_{20}}{M_{00}} - x_c^2 \quad b = \frac{M_{11}}{M_{00}} - x_c y_c \quad c = \frac{M_{02}}{M_{00}} - y_c^2 \quad (3.6)$$

步骤 3: 构建协方差矩阵 C ;

$$C = \begin{pmatrix} \text{cov}(x, x) & \text{cov}(x, y) \\ \text{cov}(y, x) & \text{cov}(y, y) \end{pmatrix} \quad (3.7)$$

$$\text{cov}(x, y) = E[(X_i - u_x)(Y_i - u_y)] \quad (3.8)$$

步骤 4: 求解特征值, 特征向量, 并归一化处理为 e_1, e_2 ;

步骤 5: 求出方向轴 e_1, e_2 与标准坐标轴的夹角;

步骤 6: 比较 ρ 、 ρ_1 、 ρ_2 若在安全角范围内, 则使用步骤 2 中的主方向, 先计算 AABB 包围盒中心, 构建 AABB 包围盒; 若超出安全角范围则继续进行步骤 7 构建 OBB 包围盒;

步骤 7: 确定 e_1, e_2 坐标下包围盒四个顶点;

设由 e_1, e_2 构建的坐标系为 UV 坐标系, 将基本 XY 空间的点转到 UV 空间;

$$\begin{pmatrix} U_i \\ V_i \end{pmatrix} = \begin{pmatrix} | & | \\ e_1 & e_2 \\ | & | \end{pmatrix} \begin{pmatrix} X_i \\ Y_i \end{pmatrix} \quad (3.9)$$

求出 UV 空间的 AABB 包围盒, 求出最大值最小值, 并设四个顶点位置坐标为;

$$(U_j, V_j)^T \quad (3.10)$$

步骤 8: 将包围盒四个顶点坐标转到 xy 坐标系, 画出包围盒;

$$\begin{pmatrix} X_i \\ Y_i \end{pmatrix} = \begin{pmatrix} - & e_1 & - \\ - & e_2 & - \end{pmatrix} \begin{pmatrix} U_i \\ V_i \end{pmatrix} \quad (3.11)$$

3.2.2.3 BVH 树的更新优化实现

针对本文 BVH 树的使用场景, 相比于更新时直接删除替换节点, 设置节点信息缓存的方式可以有效提升更新效率, 具体步骤为:

步骤 1: 在图形元素数据结构中添加旋转矩阵和缩放平移向量字段;

步骤 2: 当图形元素发生旋转、缩放或者平移时, 计算旋转矩阵、平移矩阵和缩放矩阵;

(1) 计算旋转矩阵^[56]在本平台中旋转操作规定为基于包围盒最下点进行旋转, 设旋转角为 α , 对应的旋转矩阵为 $R(\alpha)$, 旋转前, 动坐标系的标准正交基在坐标系中表示为:

$$i = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad j = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \quad (3.12)$$

旋转后, 动坐标系的标准正交基在静止坐标系中表示为:

$$\begin{cases} R(\alpha)i = R(\alpha) \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} \cos \alpha \\ \sin \alpha \end{pmatrix} \\ R(\alpha)j = R(\alpha) \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} -\sin \alpha \\ \cos \alpha \end{pmatrix} \end{cases} \quad (3.13)$$

旋转矩阵为：

$$R(\alpha) = \begin{pmatrix} \cos \alpha & -\sin \alpha \\ \sin \alpha & \cos \alpha \end{pmatrix} \quad (3.14)$$

(2) 计算平移矩阵^[57]，假设沿 X 轴平移 t_x ，沿 Y 轴平移 t_y 距离可得：

$$\begin{matrix} x' \\ y' \end{matrix} = \begin{matrix} x \\ y \end{matrix} + \begin{matrix} t_x \\ t_y \end{matrix} \quad (3.15)$$

用齐次矩阵表示为：

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \quad (3.16)$$

平移矩阵为：

$$T(t_x, t_y) = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix} \quad (3.17)$$

(3) 计算缩放矩阵^[58]，监听鼠标事件可以获得缩放倍数，假设 X 轴方向缩放为 s_x ， Y 轴方向缩放倍数为 s_y ，矩阵可以表示为：

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} s_x & 0 \\ 0 & s_y \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \quad (3.18)$$

缩放矩阵为：

$$S(s_x, s_y) = \begin{pmatrix} s_x & 0 \\ 0 & s_y \end{pmatrix} \quad (3.19)$$

步骤 3：根据旋转矩阵、平移矩阵和缩放矩阵重新计算包围盒顶点；

$$\begin{pmatrix} X'_i \\ Y'_i \end{pmatrix} = R \cdot T \cdot S \begin{pmatrix} X_i \\ Y_i \end{pmatrix} \quad (3.20)$$

步骤 4：更新 cache。

3.2.3 基于改进 ACO 的 BVH 树相交检测算法

3.2.3.1 BVH 树相交检测的问题

在传统的 BVH 树相交检测算法中，首先要对图形进行预处理，然后进行粗粒度的检测，再去除明显不相交的部分。在此基础上，将剩余图形构建成 BVH 树，并通过相交检测遍历至叶节点，从而实现精细检测。通常的精细检测阶段，是叶子节点间进行相交检测。但在，实际大型检测中这种方式检测效率很低。因此，就可以在 BVH 树构建阶段以多个三角面组成的图形部分结构作为最后的叶子节点而不是单一三角面，然后用随机相交检测算法进行求解。

利用随机相交检测算法将待检测图形进行特征对采样，由所采集的特征对构成搜索空间。在 2D 空间中的优化问题可表述为：

$$\begin{aligned} \min f(x) &= f(x_1, x_2, \dots, x_n) \\ a_i &\leq x_i \leq b_i \quad \text{s.t.} \quad i=1, 2, \dots, n \end{aligned} \quad (3.21)$$

其可行解是空间中的特征点对，因此判断是否相交，只需要在空间中搜索可能相交的特征对，即判断空间解集合中是否存在一对特征对满足最小距离小于阈值。解决多目标的最优解问题通常采用智能优化算法，比如粒子群算法、蚁群算法、狼群算法等。其中蚁群算法具备天然的并行性，我们构建的层次包围盒是二叉树结构，所以使用蚁群算法可以对子树同时遍历，从而加快检测速度。

3.2.3.2 ACO 算法及其改进

ACO 算法的基本思想是模拟蚂蚁在交流觅食时的行为。首先寻到食物的蚂蚁会沿路产生信息素，而后到的蚂蚁将有更大的概率选定有这些物质的路径，并再次释放信息素。路径越短，信息素浓度也就越高，直至所有蚂蚁都选定了最短路径。

算法过程如下：

- (1) 初始化种群和参数：蚂蚁数量 m ，蚂蚁编号 k ，时刻 t ，城市数 n ，信息素挥发系数 ρ 。初始化时每条边的信息素量都相同，所以 $\tau_{ij}(0) = C$ ， $\Delta\tau_{ij}(0) = 0$ 。
- (2) 将各只蚂蚁放置到各顶点，禁忌表 tabu_k 为对应的顶点。
- (3) 取一只蚂蚁，计算反映蚂蚁由城市 i 转移到城市 j 启发程度的启发式因子 $\eta_{ij}(t)$ 和时刻 t 蚂蚁 k 由城市 i 转移到城市 j 的转移概率 $P_{ij}^k(t)$ ：

$$\eta_{ij}(t) = \frac{1}{d_{ij}} \quad (3.22)$$

$$P_{ij}^k(t) = \begin{cases} \frac{[\tau_{ij}(t)]^\alpha [\eta_{ij}(t)]^\beta}{\sum_{s \in J_k(i)} [\tau_{is}(t)]^\alpha [\eta_{is}(t)]^\beta}, & j \in J_k(i) \\ 0, & j \notin J_k(i) \end{cases} \quad (3.23)$$

其中， d_{ij} 为城市 (i, j) 之间的距离； α 为信息素的相对重要程度； β 为启发式因子的相对重要程度； $J_k(i)$ 为蚂蚁 k 下一步允许选择的集合。

- (4) 按轮盘赌的方式确定下一个顶点，同时更新禁忌表，并重新计算转移概率。
- (5) 判断是否已经遍历所有顶点 1 次，是则进行下一步，否则重复 4。
- (6) 计算该只蚂蚁留在各边的信息素量：

$$\Delta\tau_{ij}^k = \begin{cases} \frac{Q}{L_k}, & \text{若蚂蚁 } k \text{ 在本次周游中经过边 } (i, j) \\ 0 & \end{cases} \quad (3.24)$$

其中, Q 为正常数, L_k 代表蚂蚁 k 在本次周游中所行走路径的总长度。

(7) 重复 3~6, 直到 m 只蚂蚁都周游完毕。

(8) 计算各边的信息素增量 $\Delta\tau_{ij}^k$ 和信息素量 $\tau_{ij}(t+n)$ 。

$$\Delta\tau_{ij} = \sum_{k=1}^m \Delta\tau_{ij}^k \quad (3.25)$$

$$\tau_{ij}(t+n) = (1-\rho) \cdot \tau_{ij}(t) + \Delta\tau_{ij} \quad (3.26)$$

(9) 记录本次迭代的路径, 更新当前的最优路径, 并清空禁忌表。

(10) 判断是否已达到了所预定的迭代步数, 或者是否出现了停滞现象。如果是则算法结束, 并输出当前最优路径, 否则跳转到 2, 进入下一次迭代。

ACO 算法流程如图 3.9 所示:

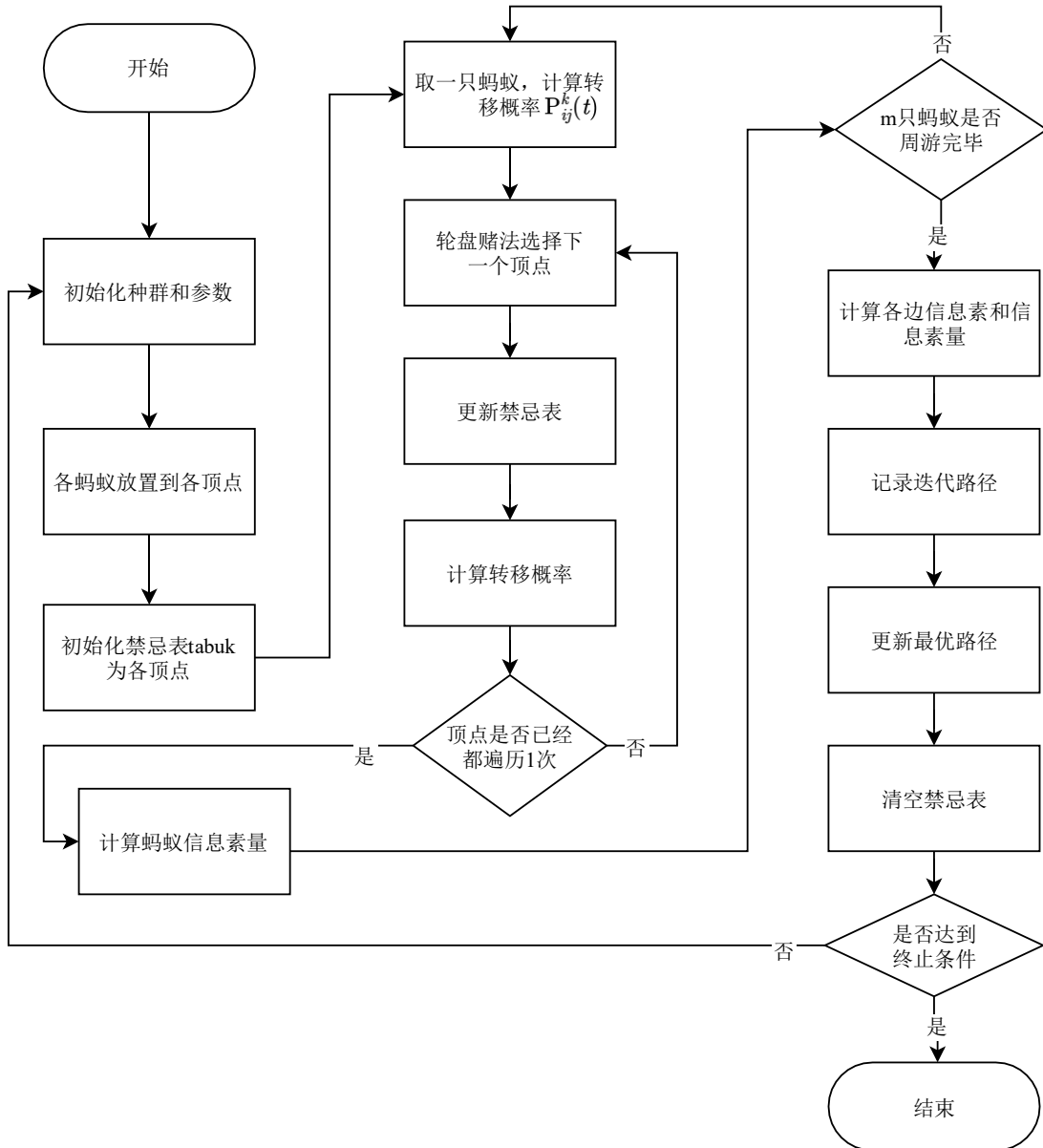


图 3.9 ACO 算法流程图

信息素挥发系数的高低直接决定各条路径上信息量的多少，也影响算法全局的收敛速度以及对最优解的搜寻能力。上述 ACO 算法的问题在于，信息素挥发系数的值是固定的，设定值越大，路径越容易被重新选取，越容易发生局部收敛。设定值越小，则会提高算法的搜寻效率，但会导致收敛速度下降。所以，在算法的早期阶段，要用较大的 ρ ，来迅速的寻找优势的路径，而在算法的后期，则要用更小的 ρ 扩展全局的搜索能力。

所以本文提出，改进的自适应 ρ 与迭代次数 u 的拉普拉斯分布变化，如式(3.27)：

$$\rho(k, \mu) = \frac{1}{2\sigma} e^{-\frac{|k-\mu|}{\sigma}} \quad (3.27)$$

$$\sigma = \frac{\sum_{k=1}^m \tau_k}{\mu} \quad (3.28)$$

其中 σ 为每完成一次搜索后，较差蚂蚁与最优蚂蚁信息素浓度的方差。

本文运用统计概率学中拉普拉斯分布的分布特征，将其运用于 ρ 的动态变化概率模型，实现自适应变化。在算法初期因为挥发系数较小，所以尽可能保留了各路径上的信息素浓度，并引导蚂蚁朝浓度较大的路径上移动。改进算法的优势在于，在早期控制 ρ 在较小范围，以防止搜索过程停滞而陷入局部收敛，随后增加 ρ ，从而增大了搜索可行解的空间，最后再减小 ρ ，从而引导更多蚂蚁靠近最优解。

3.2.3.3 基于改进 ACO 的包围盒树相交检测算法实现

由 3.2.1 总体设计，相交检测分为两个阶段，粗检测，精细检测。将上节的改进 ACO 算法应用于精细检测阶段，通过对粗检测阶段筛选出的可能相交的图形元素特征采样，运用改进 ACO 算法计算最优路径的距离，进而根据距离确定图形是否被选中。算法具体步骤如下：

步骤 1：粗检测阶段，根据 3.2.2 小节建立的混杂 BVH 树，确定根节点 AABB 包围盒是否相交，如果不相交则直接排除检测图形元素，若相交则进入下一步；

步骤 2：自顶向下对待检测图形元素层次包围盒进行遍历，判断叶子节点是否与选区叶子节点相交；

步骤 3：若相交，则进入精细检测阶段。对两叶子节点之间的部分图形元素进行特征点采样组成二维搜索空间。不相交则排除检测图形元素；

步骤 4：初始化种群 m 和禁忌表以及相关参数，初始化信息素量 $\tau_{ij(t)}$ 、 $\eta_{ij}(t)$ 启发式因子；初始化蚂蚁位置，并将其放置到搜索空间；

步骤 5：根据转移概率 $P_{ij}^k(t)$ 移动蚂蚁，迭代每一个顶点，更新禁忌表；

步骤 6：计算每只蚂蚁的信息素量，计算各边的信息素增量 $\Delta\tau_{ij}$ 、自适应信息素挥发因子 $\rho(k, \mu)$ ，进而求得各边信息素量 $\tau_{ij}(t+n)$ ；

步骤 7：记录迭代路径，更新最优路径，清空禁忌表，确定是否满足迭代终止条件，即最优路径是否满足确定相交的阈值，满足则作出相交响应，不满足则跳转到步骤 4 继续迭代。

3.3 实验测试

在本节中将搭建实验平台，设置对比试验测试本节中提出的算法改进与整体方案的性能。测试的主要硬件信息为 macOS Catalina 操作系统，以及 2.6 GHz 六核 Intel Core i7 处理器。

3.3.1 混杂 CD 算法的对比分析

为验证本文提出的改进混杂 CD 算法的有效性，本节从 BVH 树的构建速度，包围紧密度，更新节点耗时三方面对其进行测试，并设置对照实验。

3.3.1.1 改进混杂 CD 算法的 BVH 树构建速度对比测试

本文对混杂 CD 算法的 BVH 树构建过程改进在于，加入了自动判别包围盒(ADBB)类型。因为图形元素的基元数量与图形元素真实性呈正相关，基元数量越多，越逼近图形元素真实形状，BVH 树的度数越高，耗时越长。所以本节使用图形的基元三角面作为测试变量，分别使用 ADBB 包围盒的混杂 CD 算法，固定 AABB-AABB-OBB 包围盒的混杂 CD 算法，固定 AABB-OBB-OBB 包围盒的混杂 CD 算法对具有不同数量三角面的同一图形构建 BVH 树，耗时对比如图 3.10 所示，具体测试数据如表 3.1 所示。

表 3.1 构建 BVH 树耗时实验测试结果

方法	三角面数量					
平均耗时(ms)	100	200	400	800	1600	3200
AABB-OBB-OBB 构建 BVH 树	32	54	81	118	207	416
AABB-AABB-OBB 构建 BVH 树	7	21	32	59	98	225
ADBB 构建 BVH 树	8	19	35	53	85	177

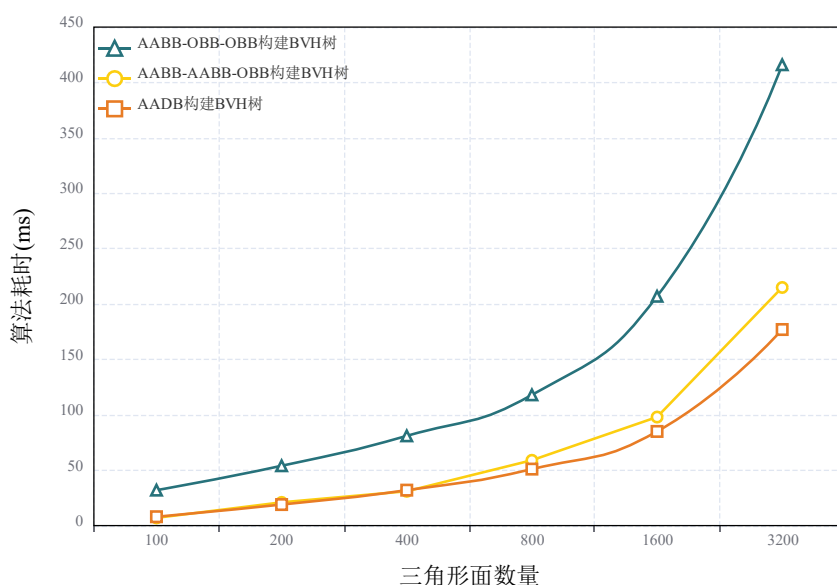


图 3.10 构建 BVH 树耗时对比图

由测试结果可以看出固定 AABB-OBB-OBB 包围盒的混杂 CD 算法耗时过长,这是由于构建 OBB 包围盒相比于构建 AABB 包围盒算法要更复杂。在三角片数较少的情况下,固定 AABB-AABB-OBB 包围盒的混杂 CD 算法有一定优势,但是紧密度较差,紧密度对比将在 3.3.1.2 小节详细讨论。且随着三角片数量增多,这种固定的包围盒的类型的缺点也逐渐显示出来,第三层开始全部使用 OBB 包围盒失去了灵活性且增加了运算复杂度。综合来看,在构建 BVH 树阶段,本文设计的自动判别包围盒(ADBB)种类的混合 CD 算法更加优越。

3.3.1.2 改进混杂 CD 算法紧密度对比测试

紧密度为包围盒与实际图形之间的空隙,空隙越大紧密度越差,反之空隙越小紧密度越好,本文使用 BVH Inspector 测试集对 AABB 包围盒树、AABB-AABB-OBB 包围盒树,本文改进混杂 CD 包围盒树进行紧密度测试,测试图形如图 3.11 所示:



图 3.11 紧密度测试图形

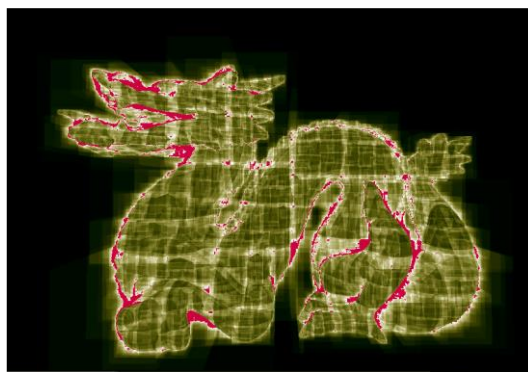
测试结果如图 3.12 所示,图中红色区域代表最后构建的 BVH 树与实际图形元素间的空隙。由测试结果可知,单一 AABB 构建的 BVH 树紧密性最差;AABB-AABB-OBB 构建的 BVH 树紧密性较好,但是在第二层使用固定 AABB 包围盒还是丢失了一些包裹精度;本文改进混杂 CD 算法构建的 BVH 树紧密度最佳,从而证明了改进的有效性。



(a)



(b)



(c)

图 3.12 紧密度测试结果

(a)AABB 包围盒树紧密度 (b) AABB-AABB-OBb 包围盒树紧密度

(c) 改进混杂 CD 包围盒树紧密度

3.3.2 改进 ACO 的包围盒树相交检测算法对比分析

为验证本文在包围盒相交检测算法中的改进效果，本节首先从宏观上设计了相交检测效率对比实验，证明了改进 ACO 算法运用于相交检测上的优越性。然后再从微观上设计了改进 ACO 算法寻优对比实验，进一步证明了本节中对 ACO 算法改进的有效性。

3.3.2.1 改进 ACO 的包围盒树相交检测效率对比测试

本文在选区方案中的相交检测算法上的改进在于加入了 ACO 算法，提升了检测效率。为验证本文算法效果，本节设置了对比试验，分别对比未使用 ACO 算法的相交检测、使用基本 ACO 算法的相交检测，使用本文改进 ACO 算法的相交检测，三种相交检测的检测效率。

实验模型如下图 3.13 所示。

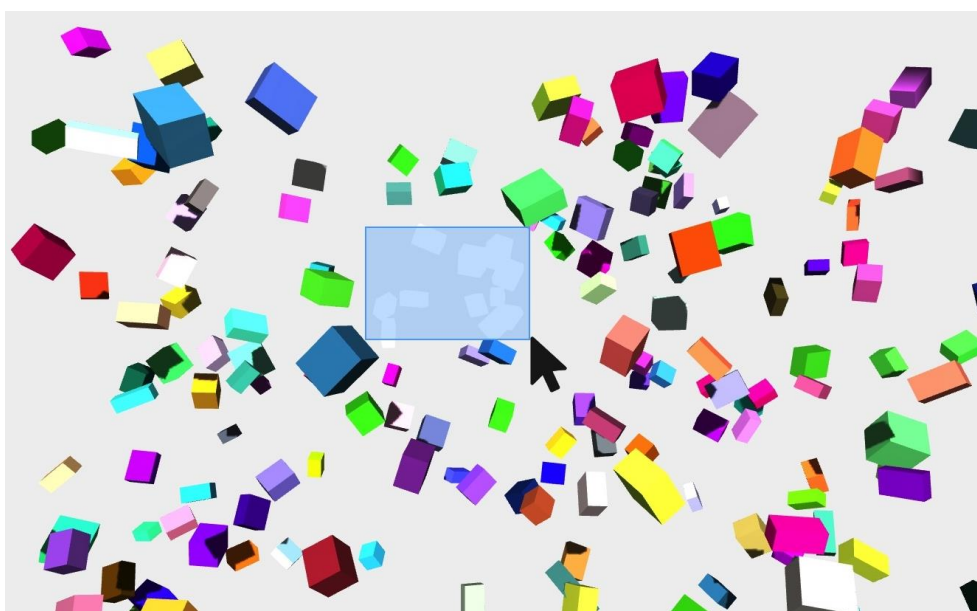


图 3.13 相交检测效率测试模型图(局部)

模型中在画布中加入了 200 个图形元素作为检测对象，使用 WebGL API 着色器语法生成矩形选区，模拟用户框选过程，三种算法的检测效率如图 3.14 所示，图中 X 轴为检测时间，Y 轴为检测效率。由测试结果可知，本文改进 ACO 算法的检测效果更好，在 500ms 左右就可以完成检测，所以后面的线是平直的，说明改进是有效的。

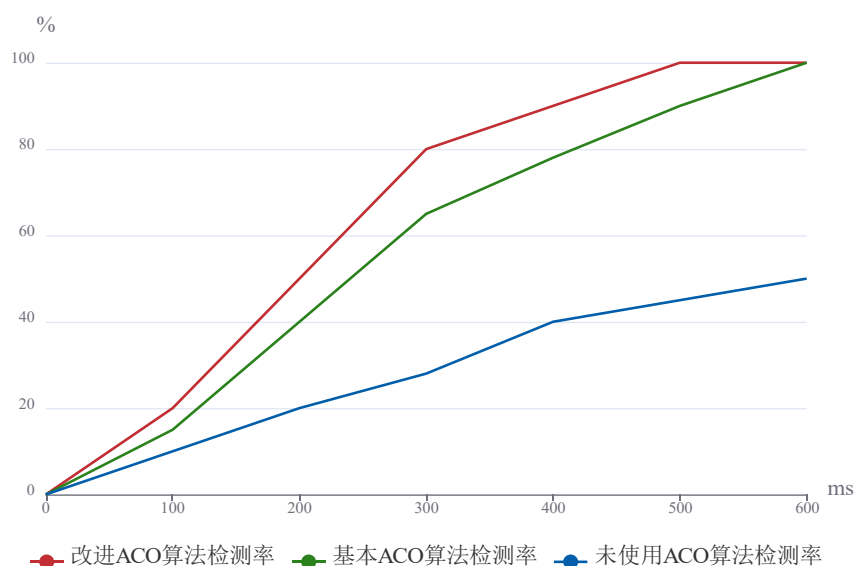


图 3.14 相交检测效率测试结果

3.3.2.2 改进 ACO 算法寻优对比测试

为了进一步验证本文改进 ACO 在相交检测中对 BVH 树节点中特征对之间最优距离的作用，本节分别采用多峰标准测试函数支架台函数(Holder Table)，碗装函数(Sphere Function)来对改进 ACO 算法、基本 ACO 算法进行对比测试，通过最优解误差来考察算法精确度，通过平均迭代次数来考察收敛性。

测试函数表达式如下：

Holder Table 函数：
$$f_1(x) = - \left| \sin(x_1) \cos(x_2) e^{\left| 1 - \frac{\sqrt{x_1^2 + x_2^2}}{\pi} \right|} \right|$$

Sphere 函数：
$$f_2(x) = \sum_{i=1}^n x_i^2$$

函数参数设置如下表 3.2、表 3.3 所示：

表 3.2 测试函数参数设置

函数	搜索空间	维度	理论最优解
$f_1(x)$	[-10,10]	50	0
$f_2(x)$	[-3,3]	50	0

表 3.3 基本 ACO 与改进 ACO 参数设置

ACO	m	α	β	初始 ρ	Q	$\max(\mu)$
基本 ACO	50	1.5	4	0.1	1	200
改进 ACO	50	1.5	4	0.1	1	200

根据检测函数可以绘制出如图 3.15 所示图像, 采用相同的参数, 对每个函数进行 20 次检测并取平均值, 以消除算法的随机误差对结果的影响, 最终测试结果如下表 3.4 和表 3.5 所示。

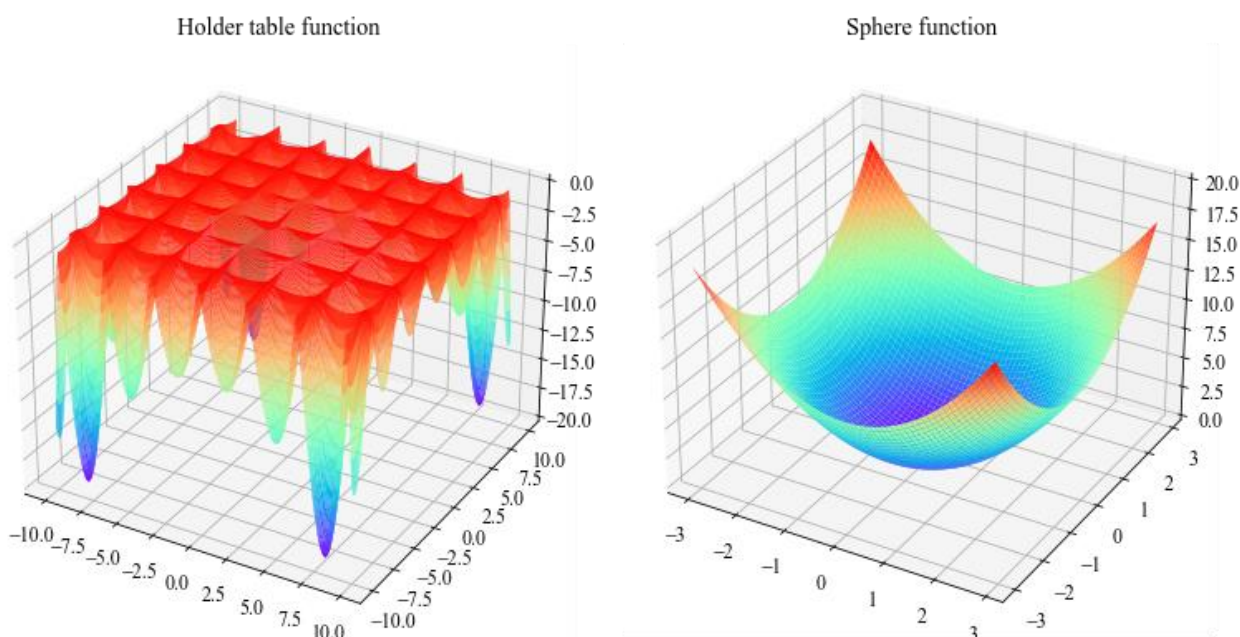


图 3.15 Holder Table、Sphere 的函数图像

表 3.4 Holder Table 测试结果

算法	最优值	最差值	标准差	最小迭代次数	运行时间/ms
ACO	0.067	0.173	0.107	154	324
改进 ACO	1.476e-04	0.067	2.31e-04	103	197

表 3.5 Sphere 测试结果

算法	最优值	最差值	标准差	最小迭代次数	运行时间/ms
ACO	0.035	0.185	0.136	116	289
改进 ACO	2.962e-04	0.087	6.2e-04	98	204

上述两个表格反映改进 ACO 与基本 ACO 对于各测试函数的求解结果。可以看出, 改进后的 ACO, 迭代次数更少且运行时间更短, 收敛精度较高。这是由于动态的调整了信息素挥发系数, 早期值较小进而避免了陷入局部收敛, 然后逐渐增大进而扩大解空间, 后期逐渐减

小指引更多蚂蚁靠近最优解。由此说明，本文对 ACO 算法的改进效果是较好的。

3.3.3 选区方案整体性能对比分析

为了测试本章提出的基于混杂 CD 算法和改进 ACO 的选区方案整体性能，在本节中将对 3.1 节所述目前低代码平台采用的选区方案：基于矩形包含算法的选区方案，基于 AABB 包围盒算法的选区方案与本章设计的选区方案进行性能对比测试。分别测试选区方案的响应时间、准确率、平均帧率。

3.3.3.1 选区方案响应时间对比测试

在低代码平台中，框选交互中当检测到选中时常伴随着属性变化例如高亮等。在本次测试中，测试用例选用 300 个动态图形节点，分别测试了交互过程中图形属性变化的平均响应时间，显然响应越迅速给用户的体验越好。测试结果如下表 3.6 所示，由测试结果做出的统计曲线如下图 3.16 所示，其中横轴代表图形元素个数，纵轴代表该方案的响应时长。

表 3.6 响应时长测试结果

选区方案响应时长(ms)	图形元素数量					
	50	100	150	200	250	300
矩形包含算法选区	152	378	586	829	1237	1678
AABB 包围盒算法选区	96	297	428	682	955	1245
本文混杂 CD 和 ACO 算法选区	117	350	421	555	781	954

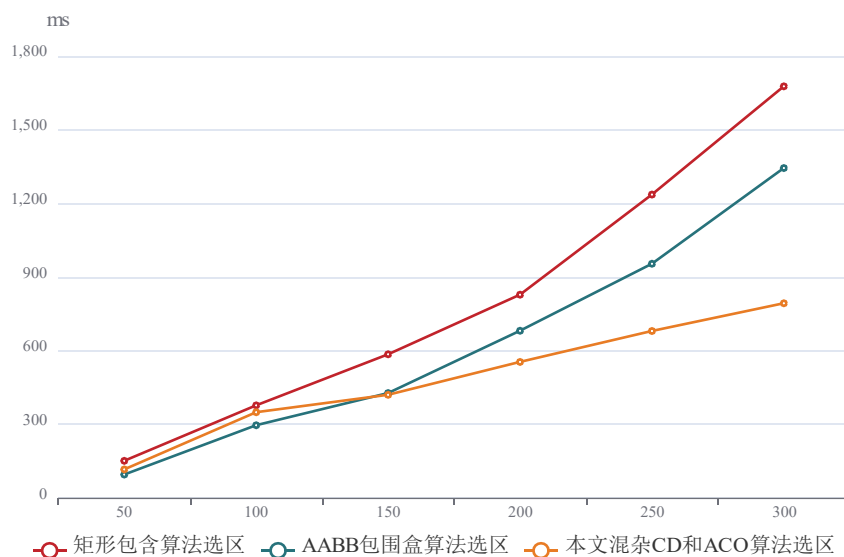


图 3.16 选区方案响应时长对比结果

三种方案随着加入图元的数量增加，响应时长逐渐增长。在图形元素数量较少时，三种方案差别不大，AABB 包围盒方案由于其相交检测算法简单，所以前期表现出更好的响应速度，所以这种方案适用于图形元素较少且对准确率要求不高的使用场景。而本文设计的选区

方案在图形元素数据量较大时响应速度表现优越，这得益于构建了混杂 CD 算法的 BVH 树，采用预处理，粗检测，精细检测的方式对图形做了过滤，更新时合理设计了缓存结构，相交检测时使用改进 ACO 算法进行了优化。结合本平台使用场景，工业设备远程运维涉及的页面图形元素多，数据量大，所以本方案适用于面向工业设备运维的低代码场景。综合来看，本文较 AABB 包围盒方案响应时长可以平均缩短 25.13%。

3.3.3.2 选区方案准确率对比测试

在涉及图形交互的系统中，准确率是另一个判断一个系统好坏的重要指标，也是本文设计混杂 CD 算法的初衷。本次测试通过测试 500 个图元在选区交互中框选的准确率，结果如图 3.17 所示，其中横轴代表图元个数，纵轴代表对应方案准确率。

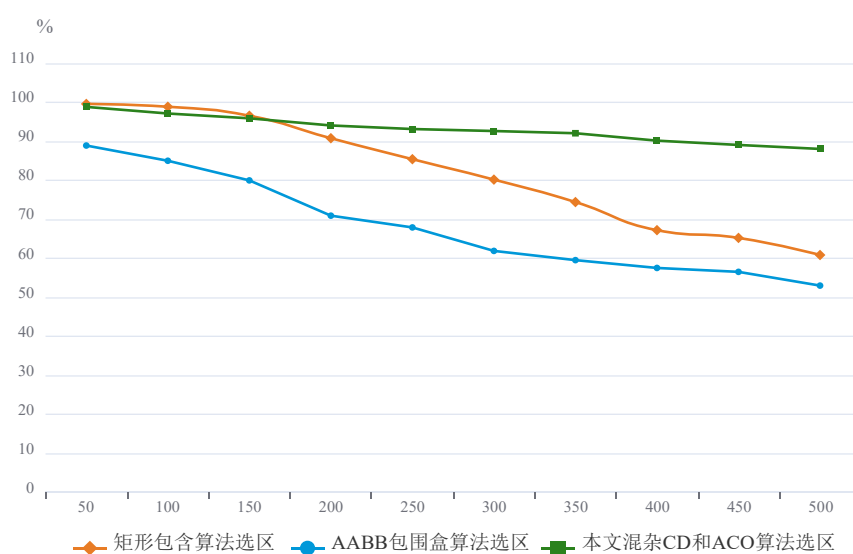


图 3.17 选区方案准确率对比

结果显示，矩形包含算法的选区方案在少量图形元素时准确率更高，因为它只有完全框选才判定选中，但是性能较差，当图形元素数量较大时，矩形包含算法由于图形元素的层层叠加准确率快速下降。本文的选区方案由于使用了混杂 CD 算法构建 BVH 树，改进 ACO 算法在精细检测阶段检测相交，相较于使用单一 AABB 包围盒算法有较大的精度提升，虽然还是由于包围盒技术丢失了一些精度，但是换取了较大的性能提升。综合来看，本文的选区方案相比于矩形包含算法的选区方案准确率平均提升 36.4%。而且更适用于数据量大，精度要求高的工业设备运维领域。

3.3.3.3 选区方案帧率对比测试

对于低代码平台是否能保证流畅性也是衡量系统的重要指标，页面的卡顿会严重影响用户体验，帧率是测试流程性的指标，也称为刷新率，表示的是每秒钟画面更新的次数。在选区交互过程中涉及动画效果，在帧率达到 50~60fps 的时候，动画会非常流畅，当帧率介于在 30fps 与 50fps 之间时，对于敏感程度高的用户来说，会察觉到轻微卡顿，而帧率小于 30fps 时，用户在操作过程中就会出现非常明显的卡顿和不适。在本次测试中利用

requestAnimationFrame API 分别计算了在三种选区方案下实时刷新率并求得了区间平均值结果如表 3.7 所示, 根据测试结果可以作出图 3.18 所示图表, 其中横轴代表图形元素个数区间, 纵轴代表该方案在对应区间的平均帧率。

结果显示, 本方案在流畅性上表现更优越平均帧率可以达到 54.8fps, 这得益于设置了图形信息缓存, 在图形变化时可以局部更新。

表 3.7 帧率测试结果

选区方案帧率(fps)	图形元素数量			
	100	200	400	800
矩形包含算法选区	56.422	55.039	44.428	39.158
AABB 包围盒算法选区	55.073	49.623	39.152	30.158
本文混杂 CD 和 ACO 算法选区	60	58.639	55.428	50.316

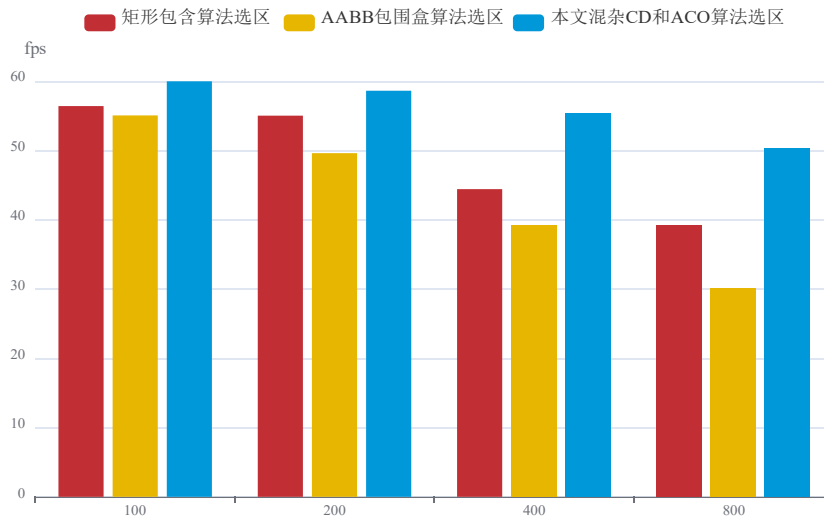


图 3.18 选区方案帧率对比

3.4 本章小结

本章设计并实现了一种基于混杂 CD 算法和改进 ACO 的选区方案。首先, 对现有的低代码平台选区方案进行分析。然后, 针对目前低代码平台选区方案准确率不足, 面对大数据量图形节点效率低下的问题, 设计了使用混杂 CD 的 BVH 树, 并对混杂 CD 算法做了改进。构建阶段使用自动判断包围盒方法构建 BVH 树, 在图形发生变化时使用缓存的变换信息快速更新树。在相交检测阶段, 设计了基于 ACO 算法的 BVH 树相交检测算法, 使用改进 ACO 算法对相交检测中的精细检测阶段进行优化, 将相交问题转换为二维空间搜索最优解问题, 有效提升检测效率。最后, 从响应时间、准确率、帧率三个方面设置对比测试, 数据表明本方案较现有方案准确率平均提升 36.4%, 在框选大数量图元时响应时长平均缩短 25.13%, 平均帧率可以达到 54.8fps, 从而验证了本方案是一种兼顾精度与性能的方案。针对面向工业设备的低代码平台图元数据量大, 准确率要求高的特点, 应用本方案具有显著的优越性。

第四章 基于 MMH3 算法和改进 Diff 算法的缓冲层方案

在第三章中讨论了如何在画布中更准确高效地选区，并在微观上缓存了图形元素信息，从而提升系统性能。然而在画布交互过程中，如何进行数据同步也是决定系统性能优劣的关键。依据前文所述，面向工业设备远程运维的低代码开发平台，设备需要实时监控，且涉及的数据量较大。现有的低代码平台大数据量交互会造成卡顿，用户体验较差，这是由于缺少对数据的缓冲处理，所以本章从数据缓冲的角度出发，提出了基于 MMH3(Murmurhash3)算法和改进 Diff 算法的缓冲层方案。宏观上，在客户端与服务端之间建立数据缓冲层，以实现有效数据过滤，进而提升系统性能。

4.1 低代码平台缓冲层方案分析

本平台为了达到轻量化且跨终端的目的抛弃了 C/S 架构，使用了 B/S 架构，这也就意味着用户操作的平台是在浏览器端，随之产生的问题也较为明显，在前后端分离结构中，前端与后端的交互形式仅仅是网络通信，这种方式势必会造成运行效率上的限制，网络通信请求与响应是需要处理时间的，而对于面向工业设备远程运维的低代码平台来说，涉及频繁大量的拖拽操作，同时设备的数据也是在实时刷新的，图形元素的数据量也很大，这就带来了两个问题，如果不将操作变动上传到服务器，很可能造成用户意外退出，操作未及时保存，导致使用体验较差；如果实时全量更新，进行服务器同步，就没办法保证用户的响应感，而用户的满意度是影响服务质量要求的重要制约因素。随之带来的性能影响也很大，同时也增加了服务器存储的压力。

考虑到以上问题，做数据缓冲是优化的首要方向，本项目中创新性的加入了数据缓冲层，对比传统交互方式，此结构抽离出来一个缓冲层对用户的操作及数据进行存储，由于直接存储在本地，大大节省了时间开销，优化了用户体验。因为数据是直接本地存储的，即便是用户偶然间退出，也不会导致数据丢失，为了确保跨终端使用，需要适时与服务器数据进行同步。

4.2 方案整体流程设计

在建设数据缓冲层时，关键点有两个：

(1) 存储方案的设计

系统的整体架构模型是 B/S 架构，所以对终端是不做限制的，那么面对多终端环境如果单纯的以文件名字存储，当首次进入页面时，系统无法判断本地存储中的缓存还是否有效，是否在其他终端已经修改过了，所以本文设计了一种基于 MMH3 散列算法的缓存摘要模型，当保存时在前端根据文件内容散列生成文件摘要，再次登录时系统可以根据文件摘要判断文

件内容是否有修改。

(2) 更新渲染方案设计

低代码平台存储的文件大多为页面的配置文件，所以获取本地存储之后和数据变动之后如何更新渲染也是问题之一，本文参考了 Vue 框架异步渲染的思路，设计了基于 Diff 算法的异步更新方案，当根据文件摘要判断配置有修改了之后，系统并不会将配置文件完全替换掉，而是使用 Diff 算法将两个文件进行对比，只更新更改过的部分。

经过前文分析，整体方案设计流程如下图 4.1 所示。

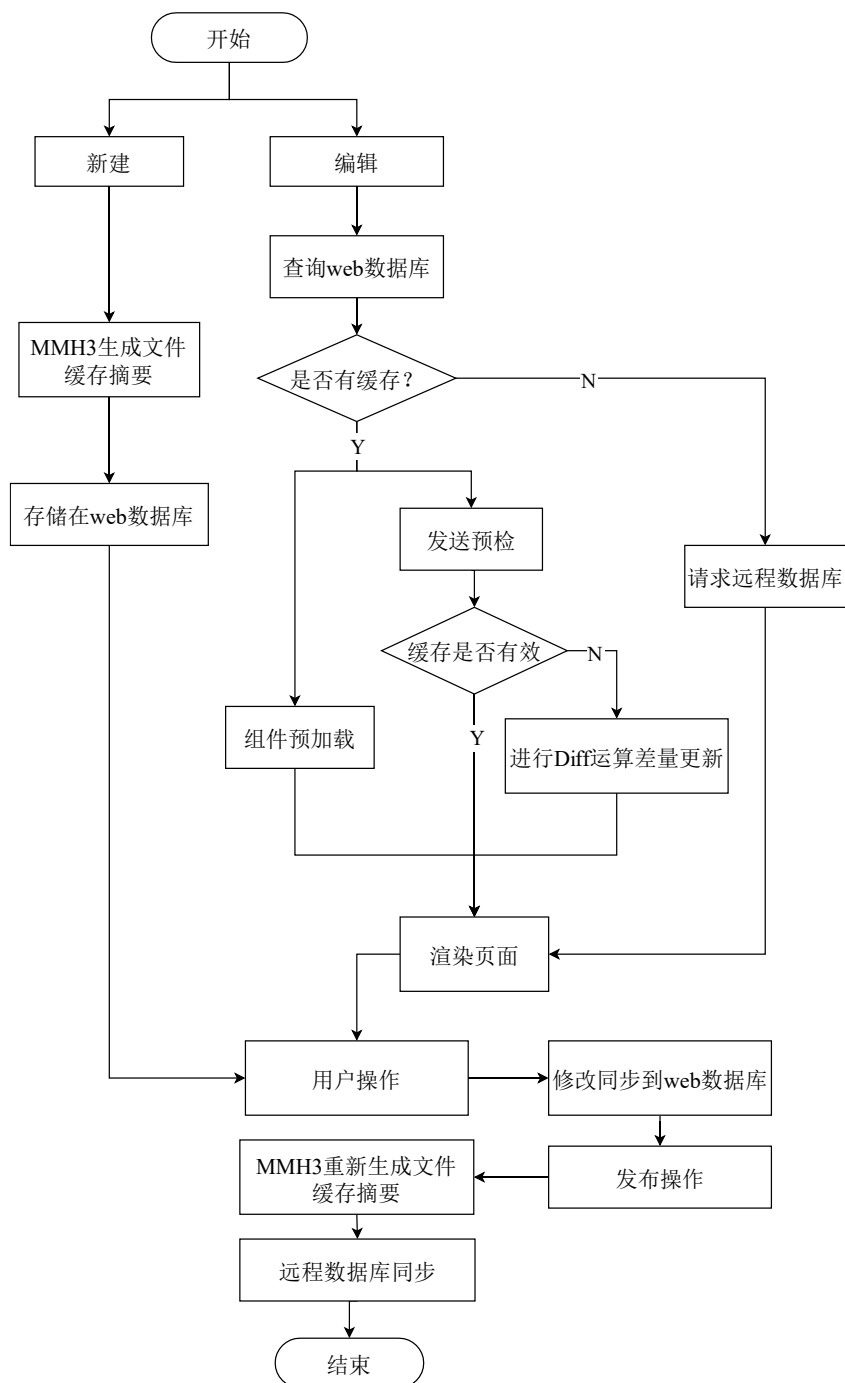


图 4.1 数据缓冲层设计流程图

具体流程为：

步骤 1：引入文件缓存摘要模型，通过优化的 MMH3 算法，根据文件内容生成一个缓存摘要，并使用 Web 数据库将文件以及缓存摘要对应储存在本地。

步骤 2：根据缓存摘要发送预检请求，判断文件是否有修改，即本地存储是否有效。

步骤 3：缓存有效时，直接使用本地存储渲染页面。失效时请求新的文件，同时根据本地存储进行预加载组件，请求数据返回时，通过改进的 Diff 算法进行新旧文件对比，更改的部分进行增量更新，并将新文件缓存在 Web 数据库。

步骤 4：当用户拖拽组件生成新的页面并点击发布按钮时，重新生成新的缓存摘要，更新本地数据库数据，并与远程数据库进行同步。

4.3 基于 MMH3 算法的缓存摘要模型

4.3.1 缓存摘要模型设计

由 4.1 小节所述，本方案引入了文件缓存摘要模型如下图 4.2 所示，通过缓存摘要将用户请求路由到本地资产上，以便实现 Web 资源的分布式缓存。缓存摘要按词义理解即是一种标识文件的方式^[59]，根据缓存摘要可以做到对文件数据的版本管理，当摘要不变时，文件内容也没有变，所以可以直接应用缓存数据，当文件摘要改变时需要拉取远程的最新文件，这样做的好处在于，可以充分利用 Web 缓存的优势，虽然预检的时候也发送了一次请求，但是仅仅发送的是缓存摘要，而缓存摘要只是一小段字符串，和直接传输整个文件相比，性能上有很大提升，在文件内容未修改时，直接使用本地缓存，省去了重新进行信息检索、下载文件的开销，可以有效的优化用户体验。

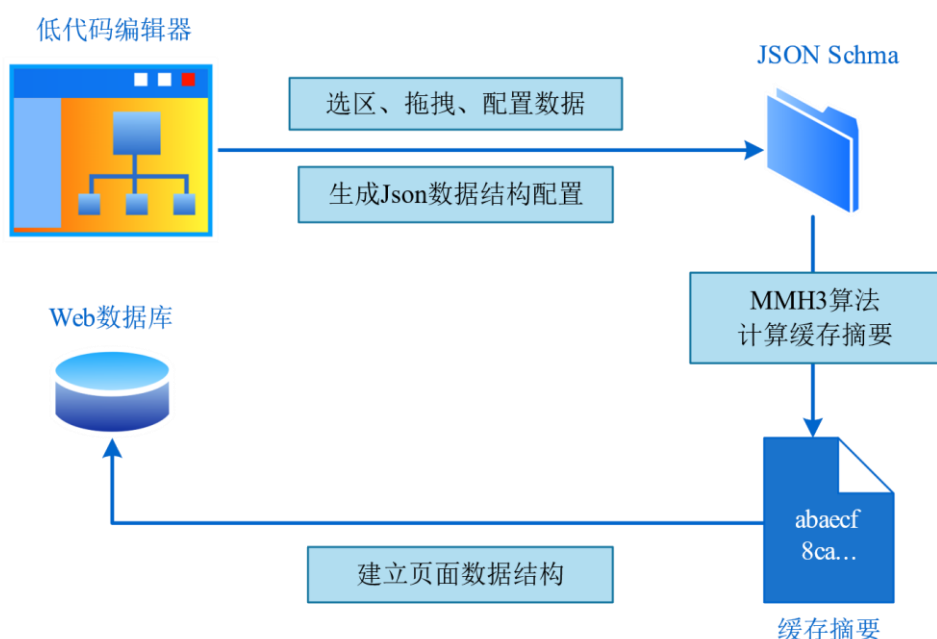


图 4.2 缓存摘要模型

根据方案中缓存摘要的功能特点和实际应用场景，可以想到通过散列算法去计算摘要，因此本小节首先阐述散列算法的基本原理，然后剖析了常见的散列算法，并基于分析结论最终确定了本方案首先采用 MMH3 算法，然后再基于 TS 语言特性实现 MMH3 算法，并进行了优化。最后通过实验测试，证明了该优化方案的效果。

4.3.2 MMH3 算法及优化策略

MMH3 算法实质上是一种散列算法，也就是所谓的摘要算法或哈希(Hash)算法^[60]，其核心原理是使用 Hash 函数来获取数据的指纹信息，并将其转化为一个长度固定的数据串，即当数据被哈希函数处理后，就会得到唯一的哈希值如图 4.3，该哈希值称为数字摘要，并且如果该数字摘要一致，那么说明它的数据是一致的。目前应用较广的散列算法有 MD5、SHA 系列算法(SHA-1、SHA-2)及其变种等等。

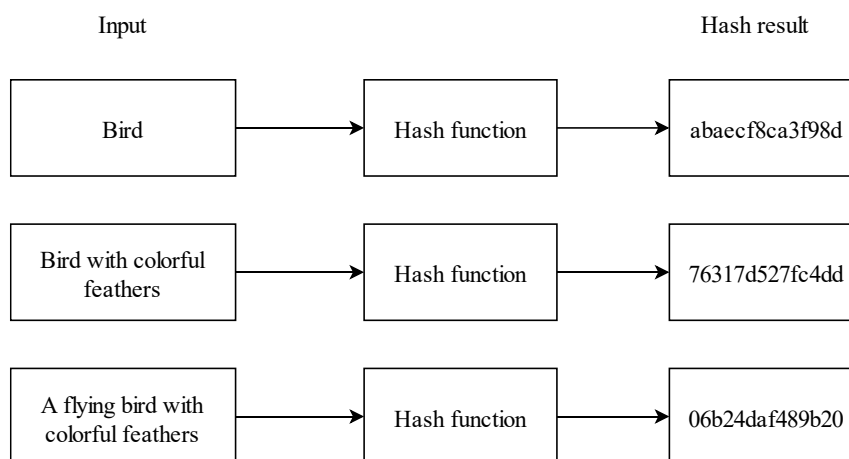


图 4.3 散列算法模型示意图

分析本方案的使用场景，系统仅仅需要一个摘要去识别文件是否有修改。其对安全性要求不高，但是对算法速度有较高要求，因为速度直接和用户体验感挂钩，所以需求是一种具有唯一性并且速度优越的算法。经过调研与测试，最终选定使用 MMH3 算法，它与系统需求契合度较高。

MMH3 是一种非加密散列算法，不像有安全性要求的 MD5 等算法，它不追求非可逆向破解，只要求高效率和低碰撞率。这种不加密的算法速度会快几十倍^[61]，适用基于散列进行查找的场景。

标准 MMH3 算法核心流程如下：

- (1) 设置常量（这些常量是经过大量卡方测试和雪崩测试选出的最优解）

$$c_1 = 0xcc9e2d51, c_2 = 0x1b873593;$$

$$r_1 = 15, r_2 = 13; m = 5, n = 0xe6546b64$$

- (2) 对输入的数据每四个字节分为一组；

- (3) 每组数据赋值给变量 K ；

- (4) 计算 K 值：

$$\begin{aligned}
K &= K * c_1 \\
K &= K \text{ rot by } r_1 \\
K &= K * c_2
\end{aligned} \tag{4.1}$$

(5) 在函数输入参数中，除了需要 *hash* 的数据，还需要设置随机数种子 *seed*。

seed 为一个随机数，这里定义 $h = seed$ ；

(6) 计算 *h* 值：

$$\begin{aligned}
h &= h \wedge K \\
h &= h \text{ rot by } r_2 \\
h &= h * m + n
\end{aligned} \tag{4.2}$$

(7) 迭代字符串每组重复执行 3, 4, 5, 6；

(8) 计算最终输出值（下面的常量内容都是经验值）：

$$\begin{aligned}
h &= h \gg 16 \\
h^* &= 0x85ebca6b \\
h^\wedge &= h \gg 13 \\
h^* &= 0xc2b2ae35 \\
h^\wedge &= h \gg 16
\end{aligned} \tag{4.3}$$

目前对 MMH3 算法的实现语言多是 JAVA, C++, GO 语言等，而我们的使用场景是在前端生成缓存摘要，所以使用前端语言对 MMH3 算法进行实现是十分必要的，最终选定使用 TypeScript 语言^[62]。由于它并不限于 32 位整数，相反，因为它在内部将整数表示为双精度浮点，因此，它仅被限制为 52 位整数（尾数长度），而 52 位足以包含 48 位的信息，并且由于浮点运算的灵活性，它能够对各种指数/移位位置执行此操作。因此，它能够包含 32 位和 16 位数字相乘的结果，而结果的低位不丢失最低的 16 位，结果的高位部分不会丢失接下来的 16 位，后者是由于浮点的灵活性，结果的高位部分没有使用最低的 16 位（浮点的指数部分相对较大）接下来的 48 位存储在浮点尾数中，直到代码中最高的 32 位被清除。与使用 16 位成员的乘法来模拟 32 位乘法的算法相比，这种方法可以用更少的操作来模拟 32 位成员的乘法，也正因为如此 32 位操作数的加法可以更简单地执行。

4.3.3 优化的 MMH3 算法设计

标准 MMH3 算法的核心思路就是不断进行左移运算，通过位运算提升了运算速度。由 4.3.2 小节所述，TypeScript 中采用了 IEEE754 标准的 64 位双精度浮点数规范，如表 4.1 所示。在内存中，64 位浮点数的具体表现形式为：第 0 位表示符号位，决定数的正负；1 至 11 位表示指数位，决定数值的具体大小；12 至 63 位表示小数位即有效数字位，决定数值的精度。在 IEEE 754 规范中表示，第 1 位的有效数字预设值始终为 1，并且它不包含在 64 位浮点数之中。所以，有效数字总是 1.xx...xx 的形式，其中 xx...xx 的部分被保存在 64 位浮点数当

中，最大的可能为 52 位。TypeScript 所提供的有效数字长度最长为 53 个二进制位，如下图 4.4 所示。

表 4.1 64 位双精度浮点数规范说明

位数	描述
第 0 位	符号位，以 s 表示，0 代表正数，1 代表负数
第 1 位到第 11 位	储存指数部分，以 e 表示
第 12 位到第 63 位	储存小数部分（即有效数字），以 f 表示

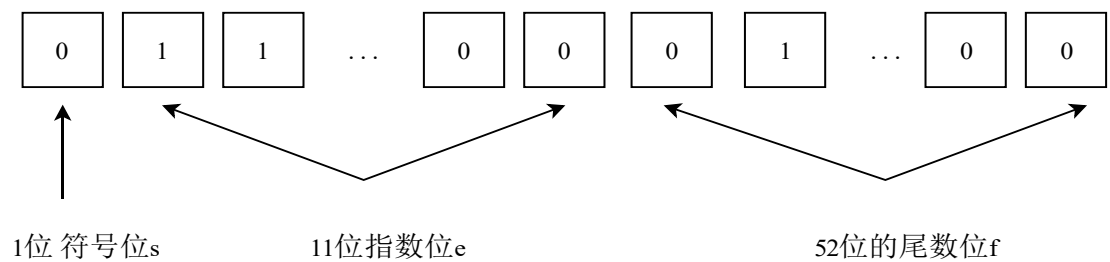


图 4.4 TypeScript 提供的有效位图示

使用 TypeScript 语言乘法结果可以包含 32 位和 16 位数字相乘，这样结果的高部分就没有使用最低的 16 位（浮点的指数部分相对较大）接下来的 48 位存储在浮点尾数中，直到代码中最高的 32 位被清除。基于此对 MMH3 算法进行优化，标准 MMH3 算法和优化过的算法关键实现代码对比如表 4.2 所示：

表 4.2 标准 MMH3 算法与优化 MMH3 算法关键表达式对比

标准 MMH3 算法实现	优化的 MMH3 算法实现
c1 = 0xcc9e2d51	c1_low = 0x2d51 c1_high = 0xcc9e0000
c2 = 0x1b873593	c2_low = 0x3593 c2_high = 0x1b870000
K*= c1	K = (c1_high * K 0) + (c1_low * K);
K =K rot by r1	K= (K << 15) (K >>> 17)
k *= c2;	K = (c2_high * K 0) + (c2_low * K)
h = h rot by r2	h= (h << 13) (h >>> 19)
h *= 0x85ebca6b	h = (0x85eb0000 * h 0) + (0xca6b * h)
h*= 0xc2b2ae35	h = (0xc2b20000 * h 0) + (0xae35 * h)

4.4 基于改进 Diff 算法的增量更新渲染方案

4.4.1 增量更新渲染方案设计

在低代码开发平台中，页面的搭建到渲染的流程是：首先用户通过可视化界面拖拽图形组件到画布中；然后进行自定义搭建与配置；搭建完成后点击发布，生成一个页面；数据变动重新渲染页面。

其中之所以可以生成页面，是因为系统底层做了解析，根据用户拖拽产生的图形信息转换为 JSON Schema，最终传输的是一个 JSON 格式的文件。随着页面复杂度增加，这个文件会越来越大，所以我们使用了缓存摘要模型。但仅从设置缓存的方式去优化，也无法达到相对理想的效果，因为虽然节省了传输文件的时间，但是拿到文件之后系统是根据 JSON 文件中描述的节点信息进行全量更新的，也就是说当一个页面的一部分并没有修改，还是全部重新进行了加载。针对以上问题本文创建了一种增量更新的方式如图 4.5 所示。不论文件是否有修改，都先拿到缓存进行组件预加载，然后使用改进后的 Diff 算法对比新旧配置文件，将改变的部分进行异步更新，最后进行页面的渲染，经过用例测试，这种方式可以有效缩短更新渲染时长，提升软件使用体验。

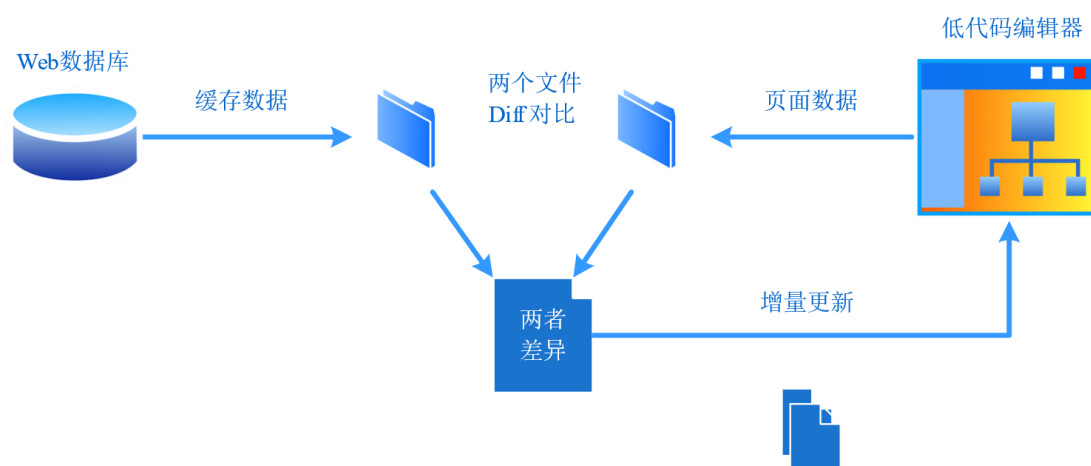


图 4.5 增量更新渲染模型

4.4.2 Diff 算法及改进策略

对比字符串结构显然没有对比树形结构来的高效，所以我们的思路是将 JSON 字符串转化为树形结构，这样需要解决的核心问题就转变成如何对比两颗树形结构的差异。由一个树转化成另一个树是一个复杂的，并且值得深入研究的课题。传统的 Diff 算法是递归比较树中的节点。

传统 Diff 算法流程^[63]如下图 4.6 所示：

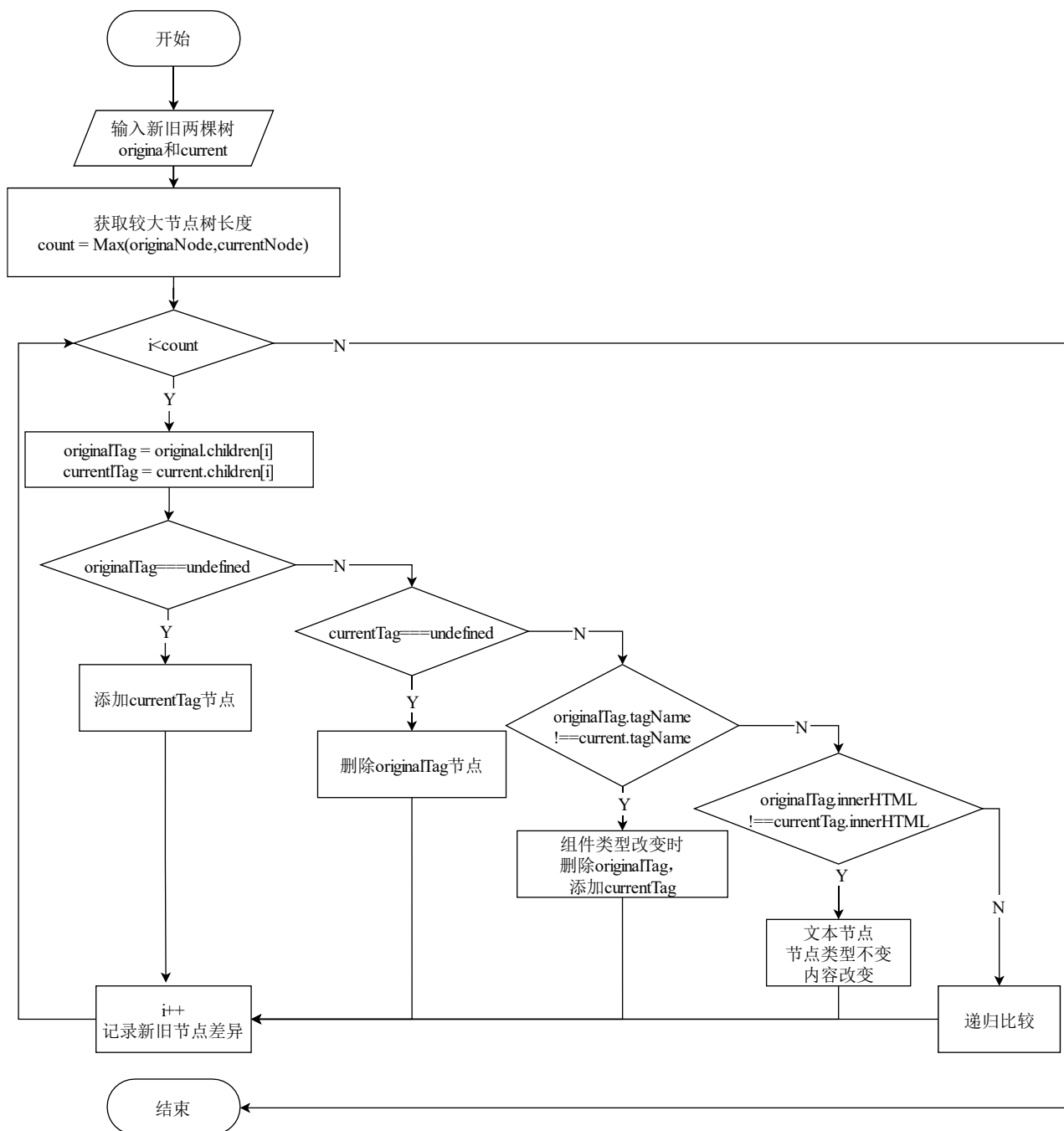


图 4.6 传统 Diff 算法流程图

核心流程可以分为以下几个步骤：

步骤 1：输入两棵树形结构。

步骤 2：获取两棵树中节点长度较大的值。

步骤 3：采用深度优先遍历算法遍历两棵树。

步骤 4：判断是否有新增节点，并记录新旧差异。

步骤 5：判断是否有删除节点，并记录新旧差异。

步骤 6：如果节点相同，判断参数与属性是否一致。

步骤 7: 如果相同, 循环递归其子节点, 重复步骤 3。

步骤 8: 输出两棵树的记录信息。

传统 Diff 算法的示意图如图 4.7 所示。根据图 4.6 所示算法流程, 对 A 节点对比流程为:

$$A \rightarrow E \quad A \rightarrow D \quad A \rightarrow B \quad A \rightarrow C \quad A \rightarrow A$$

由此可见, 传统 Diff 算法的效率低下, 算法的复杂度高达 $O(n^3)$, 其中的 n 代表的是树形结构中的节点个数, $O(n^3)$ 的复杂度性能消耗是非常大的, 显然这种算法是无法应用到本方案中的。因为方案的出发点就是节省时间, 使用传统的 Diff 算法在现代 CPU 中比较 1000 个节点至少需要一秒钟, 这与我们的初衷相悖, 所以需要一种有效的改进方案。

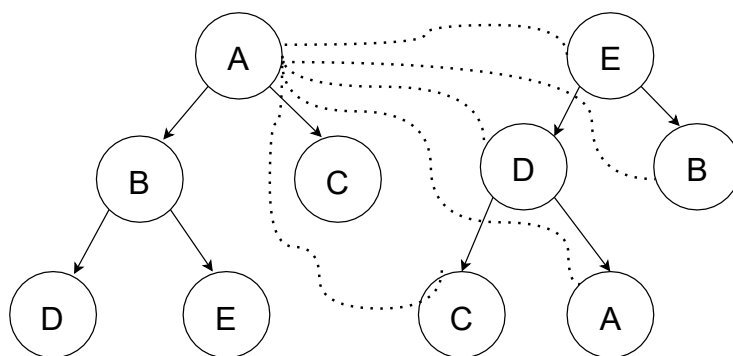


图 4.7 传统 Diff 算法示意图

4.4.3 改进的 Diff 算法设计

根据 4.4.2 小节分析, 使用传统的 Diff 算法没有办法达到性能要求, 因此需要增加新的对比策略, 设计新的对比流程, 寻求一种降低 $O(n^3)$ 复杂度的方案。本文借鉴了 Vue 框架对虚拟 DOM 的 Diff 算法的实现思路, 设计了一套应用于数据缓冲层的改进的 Diff 算法。

传统 Diff 算法之所以复杂度较高, 是因为每一个节点都与另一棵树的所有节点进行了比较, 因此考虑到是否可以借鉴 Vue 框比较虚拟 DOM 时的思路, 只进行同层级的比较, 忽略跨级操作。基于此本文做了大胆尝试:

(1) 只对比同层级的节点, 忽略跨层级节点, 如下图 4.8 所示。

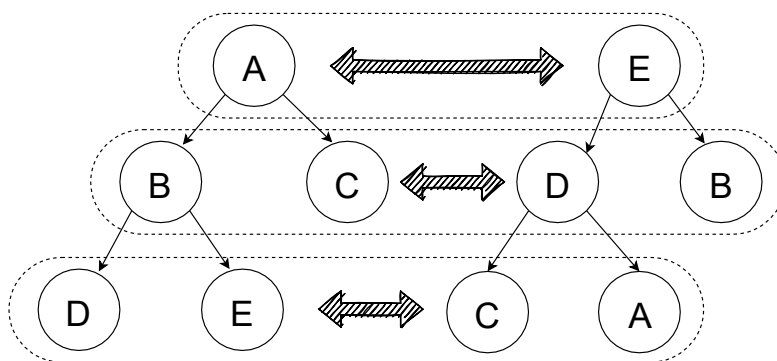


图 4.8 同级比较节点示意图

(2) 在对比过程中，采用循环由两端向中间逐渐收拢的形式，如下图 4.9 所示。

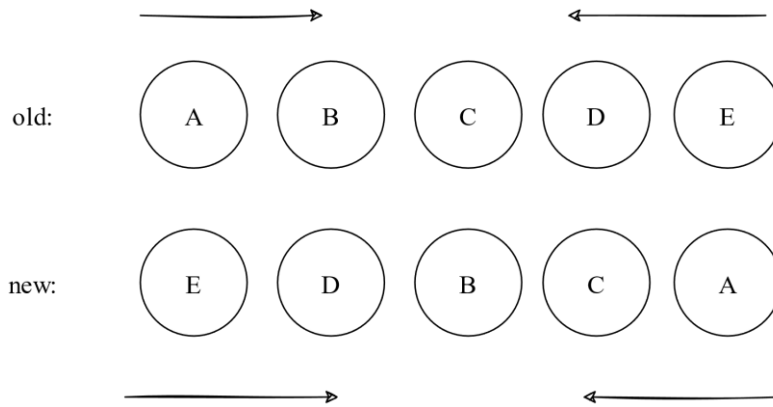


图 4.9 两边收拢循环示意图

在 Vue Diff 的基础上，结合本项目的具体实现，我们发现，在配置树上有一些固定的静态节点，而这一部分节点不应参与到 Diff 的过程中，所以基于 Vue Diff 的思路我们做了一些修改，给静态的节点设置了 **tag**，在算法过程中增加了对静态节点的判断，当遇到静态节点时直接跳过对比。

为了实现两边收拢循环定义了多个指针标识旧节点列表的起始节点、旧节点列表的终止节点、遍历到的旧节点起始，遍历到的旧节点终止、新节点列表的起始节点、新节点列表的终止节点、遍历到的新节点起始，遍历到的新节点终止。

具体说明如下表 4.3 所示。

表 4.3 改进 Diff 算法指针定义

指针名称	描述
OS (OldStarVnode)	旧节点列表的起始节点
OE (OldEndVnode)	旧节点列表的终止节点
OSIndex (OldStarVnodeIndex)	遍历到的旧节点起始
OEIndex (OldEndVnodeIndex)	遍历到的旧节点终止
NS (NewStarVnode)	新节点列表的起始节点
NE (NewEndVnode)	新节点列表的终止节点
NSIndex (NewStarVnodeIndex)	遍历到的新节点起始
NEIndex (NewEndVnodeIndex)	遍历到的新节点终止

算法的核心流程如图 4.10 所示：

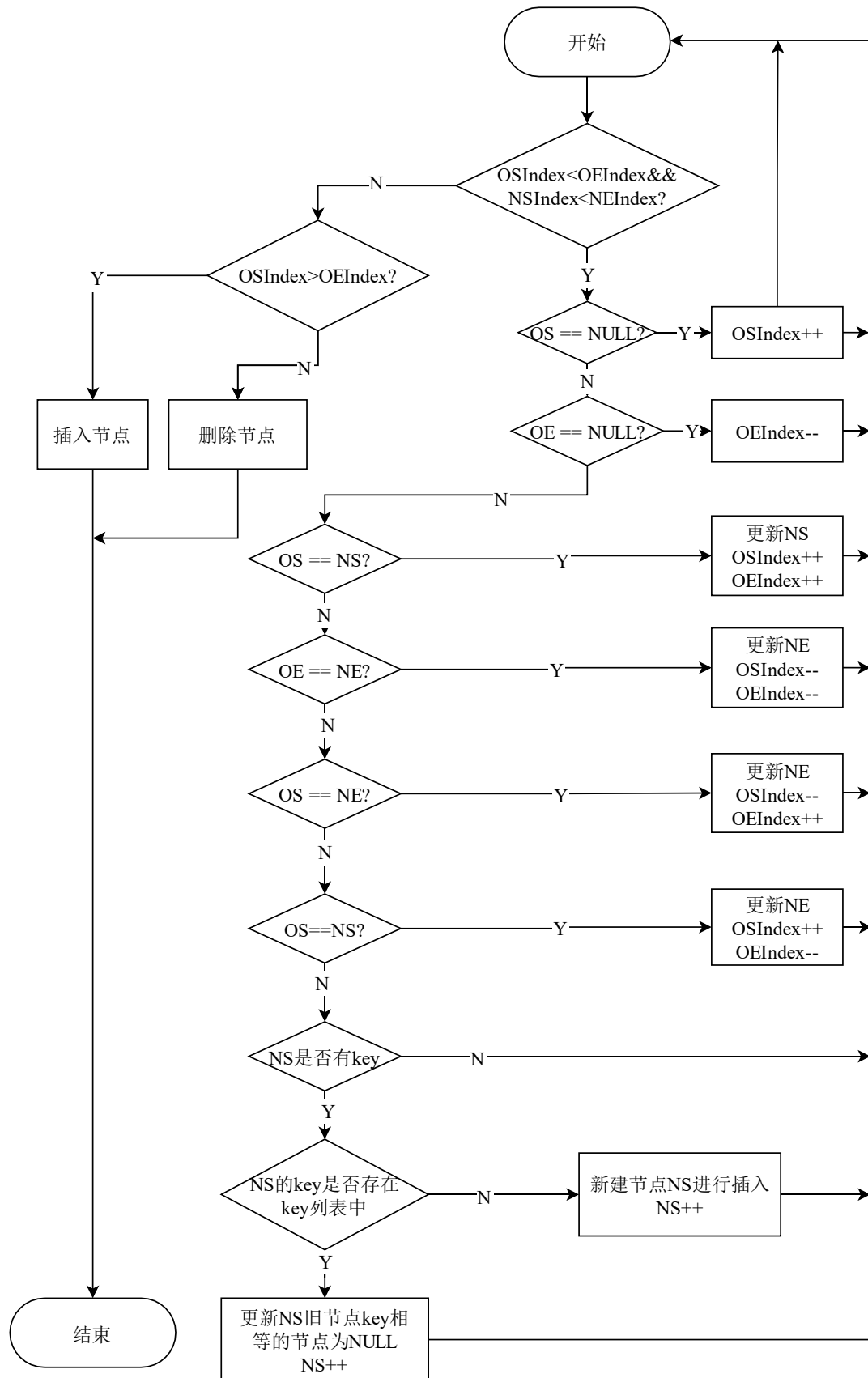


图 4.10 改进 Diff 算法核心流程图

核心算法步骤可以描述为：

步骤 1：首先确认遍历过程是否已完成，若尚未遍历，则进行步骤 2，否则若遍历完成了旧节点列表，则未遍历过的将设置为新节点并加入到节点列表中，若遍历完成了新节点列表，则从节点列表里移除未遍历的旧节点。

步骤 2：先确定旧节点的 OS 和 OE 是否为空，若为空，则跳过该节点，然后继续从步骤 1 开始；否则，继续步骤 3。

步骤 3：两两对比 OS，OE，NS，NE，如果相同则更新节点，同时将指针移动指向下一个节点，然后继续从步骤 1 开始；否则继续步骤 4。

步骤 4：首先确认 NS 中是不是有 key 值，有 key 值则确定 NS 是否能够在旧节点列表里面找到 key 相同的节点进行更新；否则创建 NS 并且插入节点列表。

4.5 实验测试

为验证前文方案设计的有效性，本节将对缓冲层方案中的两个核心，即 4.3 节提出的基于 MMH3 算法的缓存摘要模型，和 4.4 节中基于改进 Diff 算法的增量更新渲染方案进行测试，并通过设置对比试验，测试算法的优越性，最后对方案整体进行性能测试。

本次实验使用的测试设备信息如下：

- (1) 操作系统：macOS Catalina 版本 10.15.7
- (2) 处理器：2.6 GHz 六核 Intel Core i7
- (3) 浏览器：Microsoft Edge 版本 94.0.992.47
- (4) 渲染引擎：WebKit

4.5.1 基于 MMH3 算法的缓存摘要模型测试

为验证改进 MMH3 算法的效果，本节使用 SMhasher 测试集对优化过的 MMH3 算法（以下记作 MMHTS）进行测试。

(1) 健全性测试

健全性测试用来测试散列函数是否工作，它的输入是否会产生一些副作用，包括算法是否有在要散列的键的开始或结束之前读取字节这样的不健全行为，是否处理了键值的所有部分，是否生成了给定相同键的相同散列值。经测试，MMHTS 算法通过了 SMHasher 健全性测试集合。

(2) 算法性能测试

算法性能测试，测试哈希函数消耗输入的速度有多快，完成一个哈希需要多少个周期数。使用散列函数为大量数据生成唯一标识符的应用程序最关心散列函数的批量速度，而使用散列函数实现短键高效散列表的应用程序最关心散列键所需的总时间，SMHasher 测试集合将对以上两点都进行测试。对大量数据的测试，SMHasher 测试分为 0-7 组，测试哈希函数完全哈希 262144 字节键的速度。对于小数量键，SMHash 测试生成大量短键，并测量对它们进行大

量哈希所需的 CPU 周期数，然后跟踪所有运行中最快的时间。根据测试数据可以作出如图 4.11 和图 4.12 的统计图表：

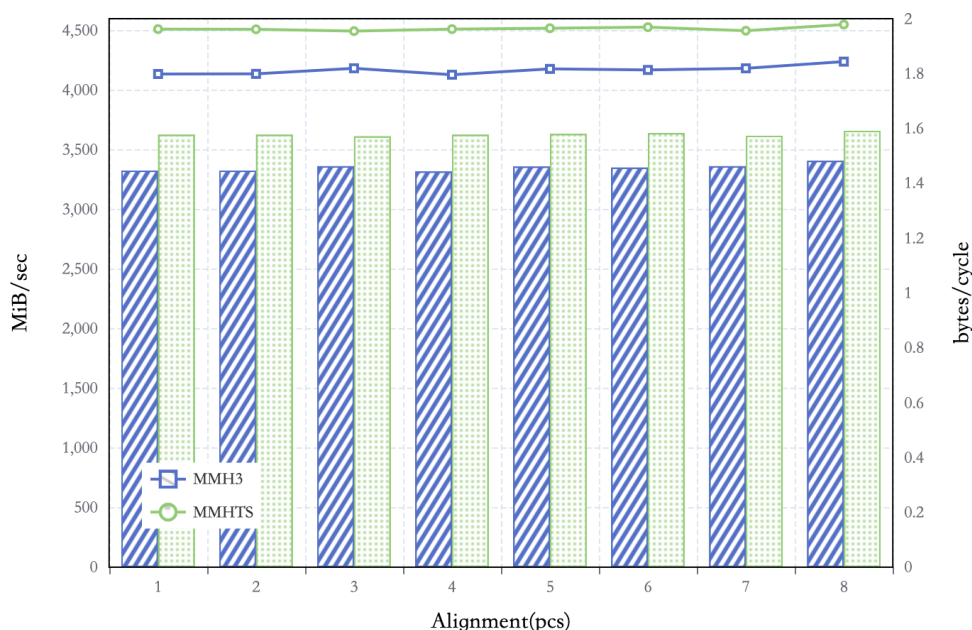


图 4.11 大数据量键批量速度测试结果

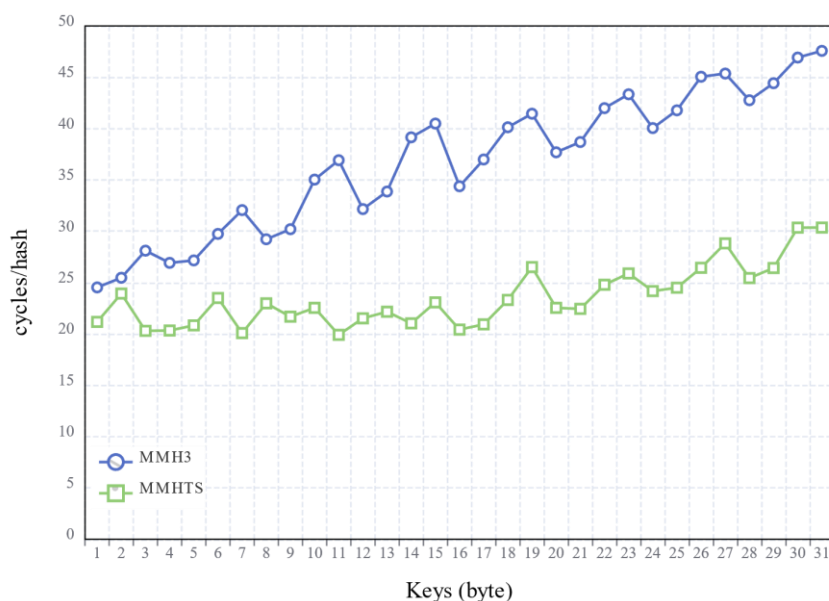


图 4.12 小数据量键散列速度测试结果

在图 4.11 中横轴代表各个分组，左纵轴代表用 262144 字节键对 0-7 组进行批量键速度测试的平均值，越高性能越好，对应柱状图；右纵轴代表每个 CPU 指令循环下处理的字节数，对应折线图。在图 4.12 中横轴代表生成的微小键数量，纵轴代表 1-31 字节键的小键速度测试的平均值代表每个哈希运算消耗的 CPU 指令循环个数，越小性能越好。

如上图 4.11 和图 4.12 的性能测试结果可知，MMHTS 算法速度更快，在散列大数据量

键时，每个 CPU 指令循环下处理的字节数更多，相同字节数速度可以提升 15.2%。在小数据量键下，每个哈希运算消耗的 CPU 指令循环个数更少，速度可以提升 24.7%，相比于标准 MMH3 算法性能更优越。

(3) 差异性测试

差异性测试用来测试近似的但并不完全相同的键产生相同的散列值的频率是多少，导致哈希函数产生相同值的两个键之间的最小差异是什么对于 m 位键中所有可能的 n 位子集，生成 1000 个随机键对，这些键对只有那些 n 位不同，然后观察它们是否散列成相同的值。如果生成了相同值，这些 n 位往往会在散列函数内部相互抵消，并产生比预期更多的冲突。SMHasher 测试所有 5 位差分用于 64 位键，所有 4 位差分用于 128 位键，所有 3 位差分用于 256 位键。微分的数量非常大(数百万级)，每个微分做 1000 个测试，所以在 32 位散列值的情况下，很可能会看到随机碰撞。正因为如此，SMHasher 报告的碰撞指发生一次以上，由于看到一次以上碰撞的可能性很小，所以散列函数的优劣可以依据此来判断。经过 SMHasher 测试集测试 MMHTS 算法在查分测试中出现 0 次碰撞。差分测试中出现 0 次碰撞说明优化后算法极少产生冲突。

(4) 雪崩测试

雪崩测试散列函数的稳定性，是否所有关键位都同样影响所有哈希位，翻转一个关键位是否会导致所有哈希位翻转。确定所有关键位是否平等的影响所有哈希位，对于 n 位散列函数，SMHasher 测试集生成 200 万个 $n*2$ 位键，并翻转每个键的每个位，比较键的原始散列和翻转位键的散列。它计算每个输出位的翻转和预期的 50/50 概率之间相差多少百分比，并输出最坏偏差，一般来说低于 1.00%的值都是可接受的。测试结果如图 4.13 所示。

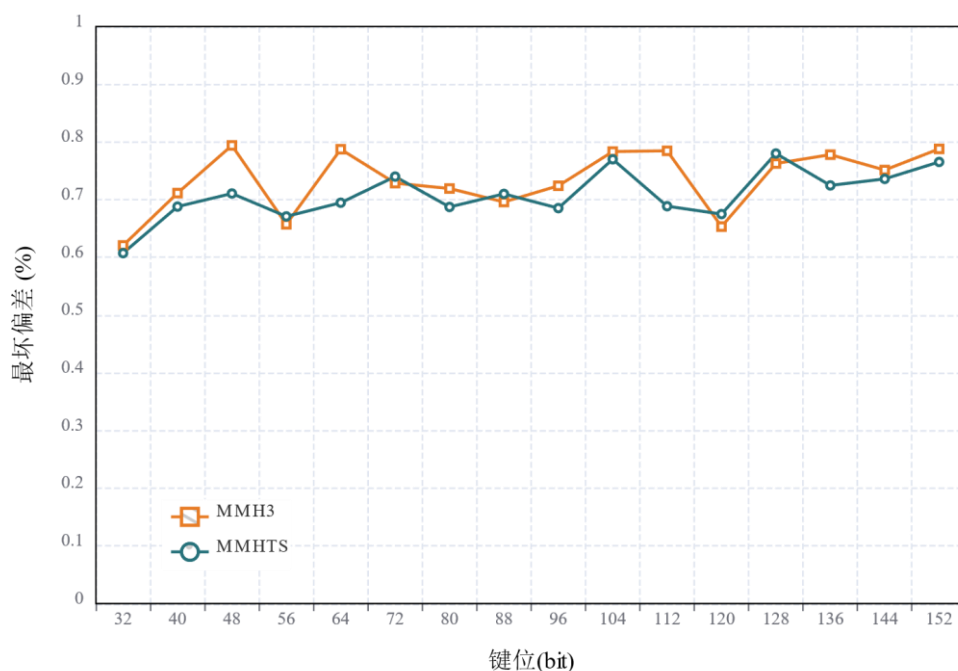


图 4.13 标准 MMH3 算法与 MMHTS 算法雪崩测试对比

在图 4.13 中，横轴代表测试的键位数，纵轴代表最大偏差，0% 表示完美的 50/50 的概率，100% 表示输出位从未翻转或总是翻转，越接近 0% 算法性能越好，由测试结果可知改进后的算法在稳定性上并没有受到太大影响，可以应用到实际项目中。

本次实验分别从健全性、性能、差异性、稳定性（雪崩）四个方面对优化后的 MMH3 算法进行测试，算法通过了 SMHasher 测试集合，优势明显。所以此模型可以应用于低代码平台，相比标准 MMH3 算法在性能上表现出更显著的优越性。

4.5.2 基于改进 Diff 算法的增量更新渲染方案测试

为验证本章节对 Diff 算法改进的效果，本小节同时从横向和纵向上设计了对比实验。为模拟用户在使用低代码平台的操作过程，逐渐分批次的在画布中投放静态节点和动态节点，然后分别测试传统 Diff 算法和改进 Diff 算法的运行时长并计算均值，测试结果如下图 4.14 所示。X 轴代表静态节点数量，Y 轴代表动态节点数量，Z 轴代表算法运行时长。

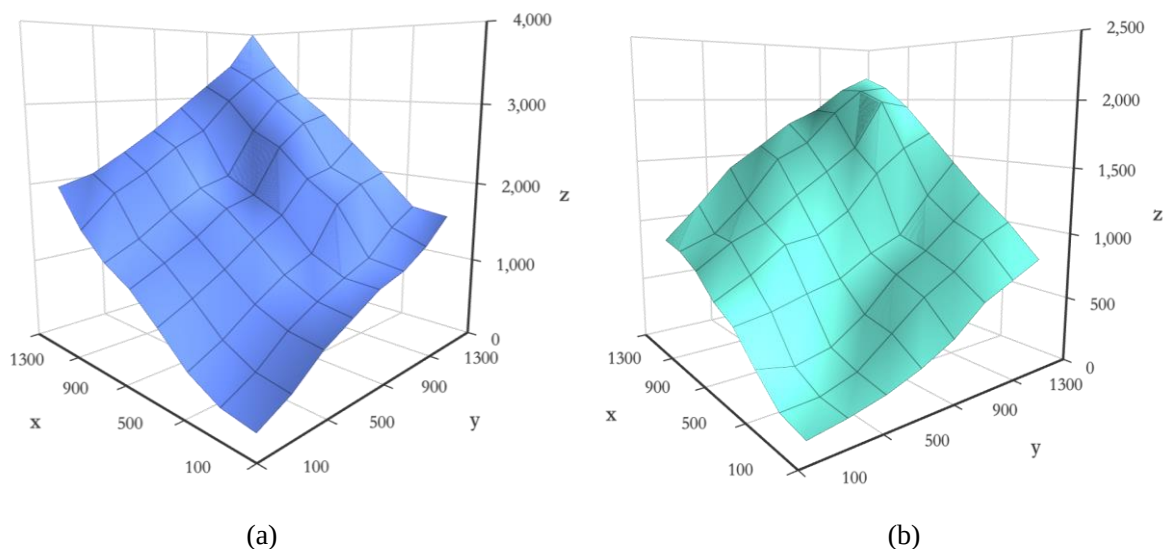


图 4.14 传统 Diff 算法和改进 Diff 算法运行时长测试结果
(a)传统 Diff 算法运行时长 (b)改进 Diff 算法运行时长

由测试结果可以看出，在相同的静态节点数和动态节点数情况下，改进 Diff 算法的运行时长更短，所以性能更好。随着静态节点和动态节点的增加，传统 Diff 算法图形越来越陡峭，这是由于没有设置静态节点的判断，而且对比节点时检测了所有节点，耗时直线上升；改进 Diff 算法则较为平缓，因为设置了静态节点指针在对比时直接跳过，对比过程中只对比同级节点，运行时长平均缩短 31.4%。

为了进一步验证改进 Diff 算法在增量更新渲染方案上的效果，本节使用 Github 开源的测试平台 js-framework-benchmark 作为基准测试。基准测试将创建一个包含随机条目的大表，并分别测试使用传统 Diff 算法的增量更新方案和使用本文改进 Diff 算法的增量更新方案在更新时的渲染时长。

计算渲染时长时可以借助浏览器测试工具 DevTools 进行测试，界面如图 4.15 所示。这

样就可以分析哪一种增量更新方案速度更快。性能监视器工具显示三个性能指标，并且提供了其他指标，CPU 使用率表示网页使用的 CPU 百分比；JS 堆大小表示页面上 JavaScript 程序使用的内存量；DOM 节点表示浏览器中跨选项卡的 DOM 节点(个)；JS 事件侦听器表示浏览器中跨选项卡的 JavaScript 事件侦听器(数)；文档表示浏览器上的文档对象数(选项卡)；文档框架表示浏览器中跨选项卡显示的文档(数)；布局/秒表示浏览器引擎每秒构造页面布局次数；样式重新计算/秒表示浏览器引擎每秒计算页面的 CSS 样式次数。

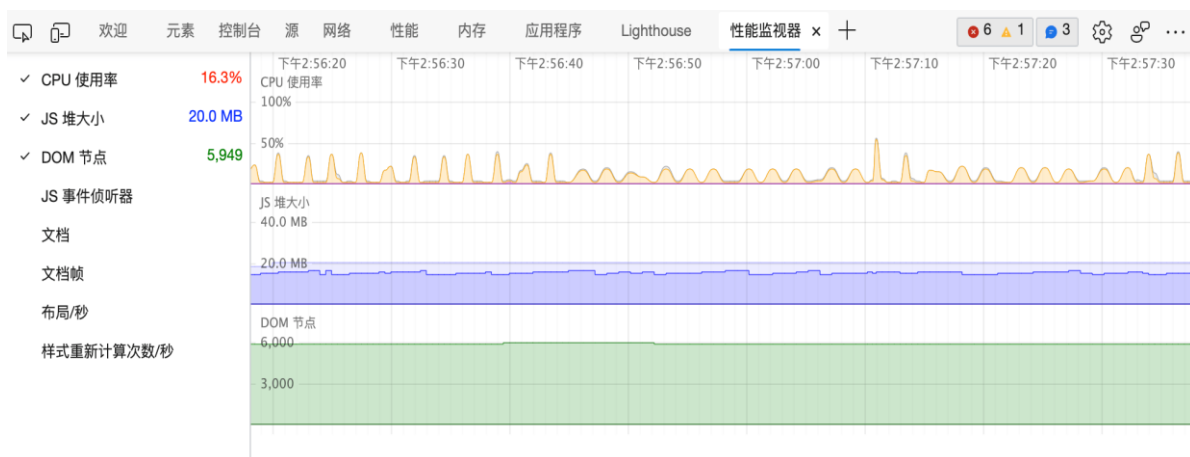


图 4.15 Microsoft Edge 开发者工具的性能监视器

具体测试内容及用例设置如表 4.4 所示：

表 4.4 测试用例与内容描述

测试项编号	测试内容描述
测试项 1	替换所有行：替换表中所有 1000 行的所需时间。
测试项 2	部分更新：为包含 10000 行的表每 10 行更新文本所需时间。
测试项 3	选择行：响应行上的单击而突出显示行所需时间。
测试项 4	交换行：交换 1000 行表中的 2 行的时间。
测试项 5	移除行：为具有 1000 行的表移除一行所需的时间。
测试项 6	创建多行：持续时间创建 10000 行。
测试项 7	清除行：清除包含 10000 行的表的持续时间。

结合本次研究内容特点，需要对基准测试进行一些修改，原先基准测试是通过创建操作一个大表格进行测试，本平台实际操作的是各种封装的组件，我们对这里进行了替换，替换成了自研的组件库渲染。为保证测试结构科学准确，所以每次测试需要至少运行 10 次，取平均值。此外，js-framework-benchmark 基准测试涵盖的是较为广泛和基础的测试内容，而本次研究是应用在特定领域的，也就是低代码平台，所以在基础测试之上，增设了 3 项适应场景的复杂操作，用来测试改进后的算法，新增项目操作如下：

- (1) 删除头节点，中间节点和尾节点。
- (2) 交换中间层第一个节点和倒数第一个节点，替换头节点。
- (3) 中间层插入复杂表格节点，删除一级子节点。

本次 10 个测试项的测试结果如下表 4.5 所示。测试结果展示了使用 Diff 算法实现的增量更新渲染传统方案，和本文使用改进 Diff 算法实现的增量更新渲染方案对比。

表 4.5 改进 Diff 算法的增量更新方案性能测试结果

	Grp1	Grp2	Grp3	Grp4	Grp5	Grp6	Grp7	Grp8	Grp9	Grp10
传统方案	1156.	1250.	1025.	873.4	1053.	1292.	983.8	1068.	1122.	1038.
	431	445	746	45	336	327	23	252	097	649
本方案	281.5	235.9	461.2	525.3	756.6	353.1	527.6	276.7	279.7	352.0
	26	92	64	46	78	84	58	45	44	93
时间差(ms)	874.9	1014.	564.4	348.0	296.6	939.1	456.1	791.5	842.3	685.7
	05	453	82	99	58	43	65	07	53	19
性能提升(%)	75.65	81.12	55.03	39.85	28.16	72.67	46.36	74.09	75.07	66.02

由结果可知，改进的 Diff 算法能够在增量更新方案中运行稳定。分析测试数据结果可知，使用改进算法的增量更新方案相比于使用原始算法在渲染性能上有显著提升，在部分更新的场景下效果最明显，性能最高可提升 81%，这是由于本方案的增量更新尽可能的保留了未变更的节点，且在对比时时间复杂度更小。在低代码平台中，有大量的配置节点是未变更的，所以此方案适用于低代码平台。

4.5.3 缓冲层方案整体性能测试

为验证本章所述的缓冲层方案在面向工业设备远程运维的低代码平台中的效果，本节对缓冲层整体性能进行对比测试，测试操作页面图形元素 200 次，分别计算使用缓冲层方案和未使用缓冲层方案平均发送的请求数、发送的请求数据包大小、接收数据包大小、页面刷新率和更新节点的个数并求均值，测试结果如表 4.6 所示。

表 4.6 缓存层方案整体性能测试结果

	平均发送的请求数据包 大小(kB)	平均接收数据包 大小(kB)	页面刷新率 (fps)	平均更新节点 个数(个)
未使用本方案	3.2	6.7	48	4726
使用本方案	2.6	2.3	54	2628

由测试结果可以看出，使用本方案构建数据缓冲层可以有效的提升系统性能，使用本方

案数据网络传输的数据量减少 42.21%，这是因为使用了改进 MMH3 算法计算了缓存摘要，当缓存没有发生变化时不传输数据。同时，使用改进 Diff 算法使得平均更新节点数量减少 44.3%，进而平台性能更优秀，平均刷新率更高，用户操作更流畅。

4.6 本章小结

本章设计并实现了一种数据缓冲层方案。首先介绍了目前应用软件采用的普遍结构，并指出本平台应用现有结构所带来的不足：运行效率较低；用户体验较差；无法保证响应感；服务器存储压力较大。从而设计了加入数据缓冲层的三层软件结构。然后围绕实现过程中核心的两个问题：如何存储和如何更新渲染，进行了深入讨论。在存储方式上设计了基于 MMH3 算法的缓存摘要模型，在实现过程中对算法进行了优化，并通过 SMHasher 测试集对优化后算法加以检测，数据表明优化后的算法对大数据量键速度可以提升 15.2%，小数据量键速度可以提升 24.7%，证实了算法优化的效果。在渲染方案上设计了基于 Diff 算法的增量更新渲染方案，并对传统 Diff 算法进行改进，然后设置对比试验测试改进的 Diff 算法的效果，结果显示改进 Diff 算法运行速度更快，平均缩短 31.4%。之后测试使用改进 Diff 的增量更新方案，结果显示使用改进后的 Diff 算法更新渲染性能最高可提升 81%。最后对整体方案进行了测试，测试数据表明加入缓冲层后平台更新渲染时网络传输的数据量可以减少 42.21%，更新节点数量可以减少 44.3%，整体性能优秀。

第五章 面向工业设备远程运维的低代码平台应用案例

本文前述章节对面向工业设备远程运维的低代码平台进行了设计和实现，并重点研究混杂 CD 算法和改进 ACO 算法用于提升选区精度、MMH3 算法和改进 Diff 算法用于解决大数据量更新渲染卡顿的问题，平台整体已经具备了搭建企业应用的基础能力且具有诸多优势。传统软件应用开发采用手工编写代码的方式，本平台采用可视化图形拖拽的方式，大大减少了工作量，降低了研发门槛；传统开发组态界面过程中相似的功能往往需要重复研发，本平台使用封装好的行业模板以及行业标准元器件图元库搭建 Web 组态界面，对物料进行了有效复用，提高了开发效率。

本平台在实际应用中已经覆盖了很多工业领域，其中与浙江省丽水市缙云县某锯床装备企业、浙江省瑞安市某机械企业、浙江省台州市某缝纫机企业的合作项目最为典型。其中使用本平台搭建的锯床物联网运维系统，企业统计数据显示 1 年时间内节约成本超 5 万元；使用本平台搭建的包装机械物联网运维系统项目周期历经 2 个月，而传统开发方式的软件定制商评估工期至少需要 3 个月，项目周期缩短 33.3%；使用本平台搭建的工业缝纫机物联网在线运维系统中，包含有 100 余页面和超过 50 张表，使用本平台从立项到交付上线仅花费了 5 个月，相较于传统的纯代码开发，节省了约 60% 的人月数，交付速度提升 3 倍。

5.1 锯床物联网运维系统

浙江省丽水市缙云县某锯床装备企业是一个传统的装备制造厂商，企业整体的数字化水平较低，设备运维还停留在人工运维的阶段，运维成本较高；锯床监控参数时常需要变更，企业缺乏软件技术人员，担心与传统外包软件公司在需求变更时对接困难，况且软件外包开发工期较长。

本平台为其提供了有效的解决方案，基于本平台开发了锯床物联网运维系统，实现了对锯床装备的远程运维。功能上实现了对锯床数据的采集和在线监测，具体效果，如下图 5.1、图 5.2 所示。

基于本平台低代码搭建应用具有显著优势。即使企业缺少高级软件工程师也可以独立进行参数调整与系统更新迭代，如图 5.3 所示。使用平台提供的大量工业场景模板，企业可以迅速搭建数据可视化监控大屏，在短短一个月的时间就完成了开发测试上线全流程。使用本文低代码平台有效的减少了研发人数和研发天数。同时，低代码开发模式也降低了对开发人员技术水平要求，降低了人员日均工资值，进而从整体上降低总成本。根据企业内部统计数据显示，使用本平台搭建的锯床物联网运维系统在 1 年时间内节约成本超 5 万元。



图 5.1 锯床分布数据大屏

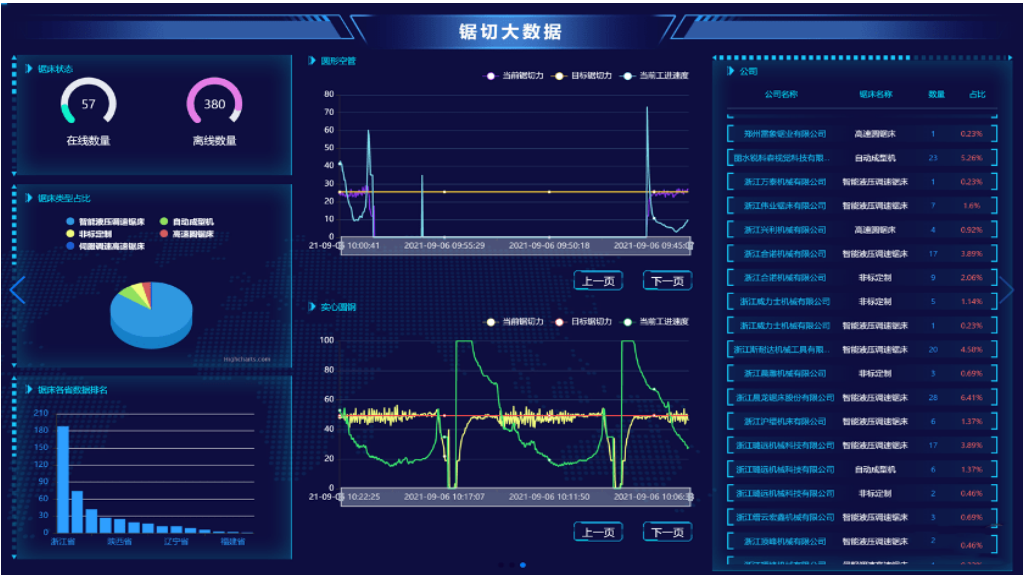


图 5.2 锯切大数据监控大屏



图 5.3 锯床装备视频监控

5.2 包装机械物联网运维系统

浙江省瑞安市某机械企业是包装机械生产厂商，企业有原始的组态系统，但是基于 C/S 架构，升级维护困难，相似内容得不到复用，建模也不方便，故障频出。企业管理平台较为分散，数据统计大屏、运维管理平台等都由不同的 Web 软件提供商提供，很难集中管理和维护。

本平台为包装机械物联网运维提供了有效的解决方案，基于本平台建设的包装机械物联网运维系统。实现了对塑料热成型机的生产过程组态建模，效果如图 5.4、图 5.5 所示。并进行实时在线监测，效果如图 5.6 所示。根据企业的系统重构规划，软件定制商评估项目周期至少为三个月，而使用本平台仅仅历时了两个月，就完成了项目交付上线，项目周期缩短 33.3%。

得益于本设计第三章所述的基于混杂 CD 算法和改进 ACO 的选区方案，实现了对机械装备的精细化建模。得益于本平台微服务-微前端架构，使得轻松方便的对企业老旧系统与新系统进行了集成如图 5.7 所示，整个流程都通过低代码拖拽的方式，操作十分简单。在设备零件逐渐增加时，通过 MMH3 算法和改进 Diff 算法建设的缓存层，使得系统运行速度和性能都得到了保证。依靠本平台，不熟悉软件研发但是对工艺流程十分熟悉的硬件嵌入式等工程师经过短期培训也完全可以参与建设线上运维系统。

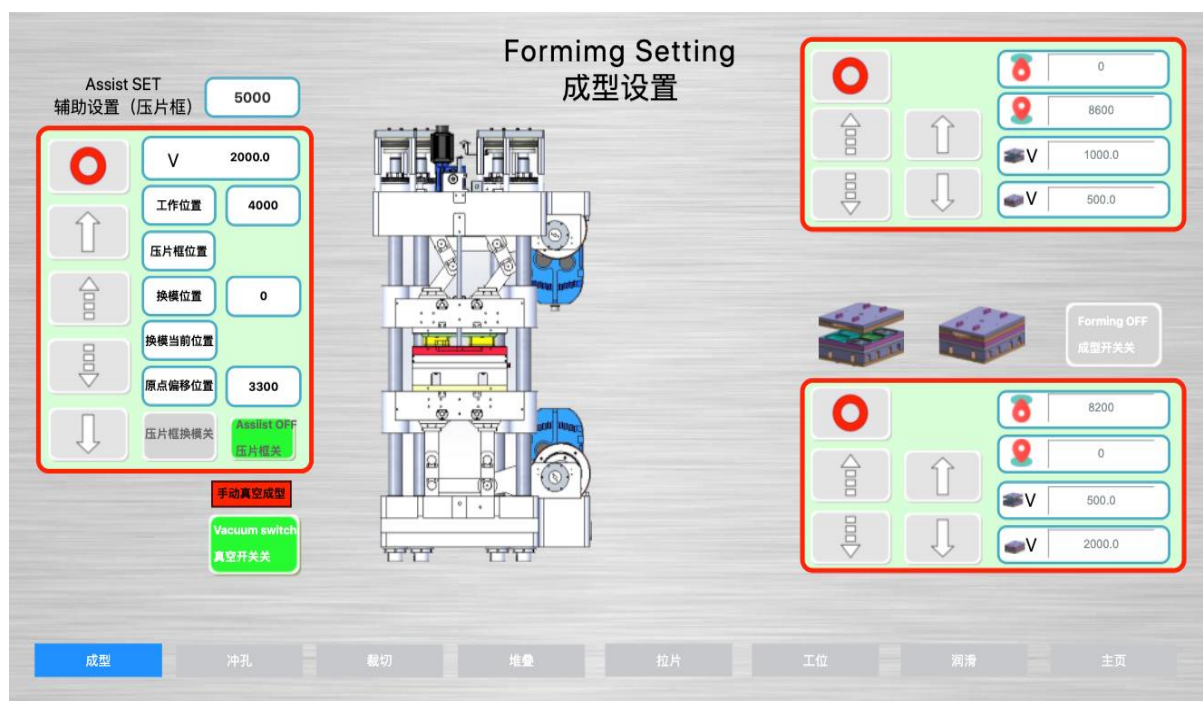


图 5.4 成型过程组态建模

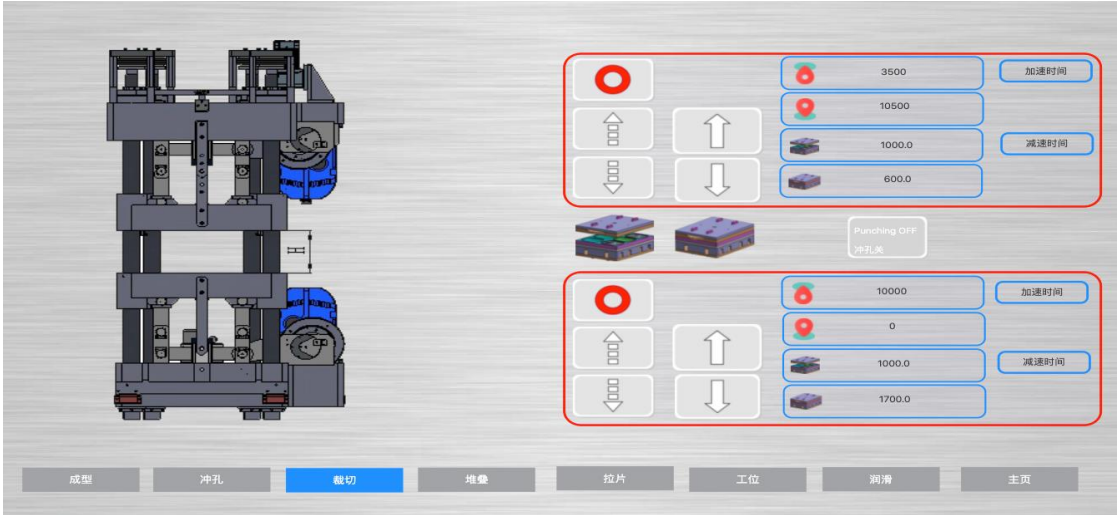


图 5.5 裁切过程组态建模

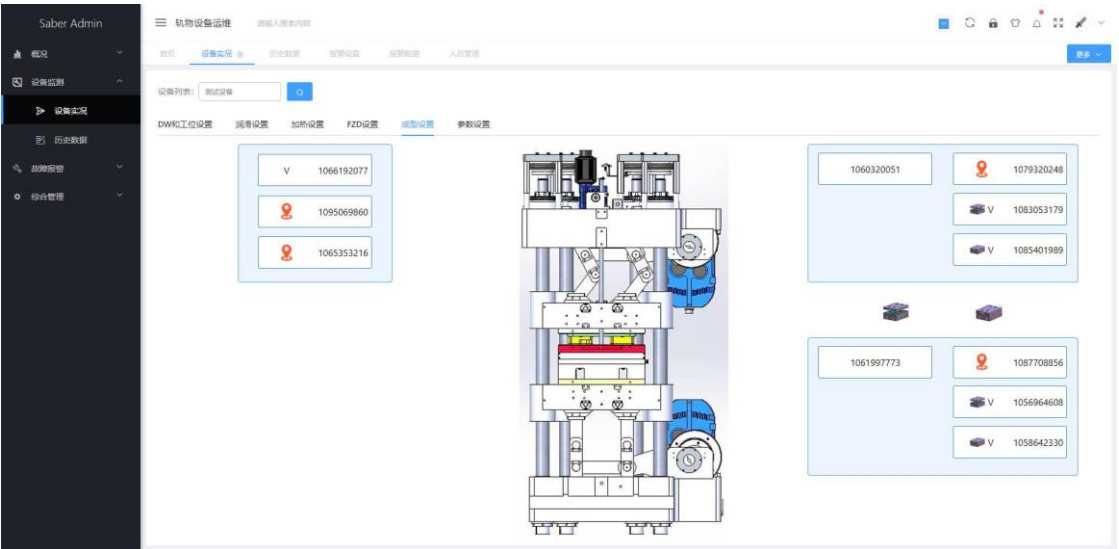


图 5.6 数据汇总界面

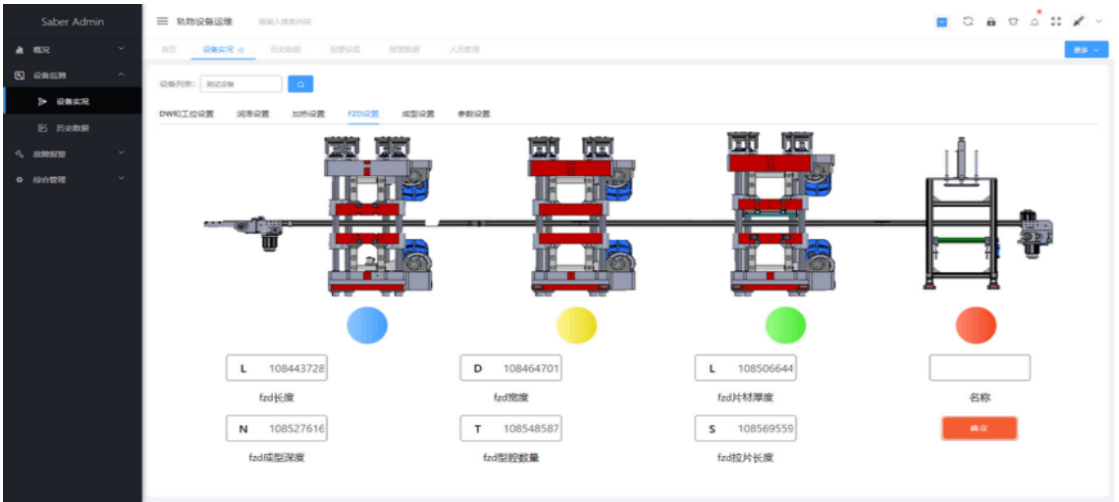


图 5.7 组态系统与运维系统集成界面

5.3 工业缝纫机物联网在线监控运维系统

浙江省台州市某缝纫机企业,是全球化的缝制设备生产厂商。监控运维上遇到的问题是,设备数量庞大,监控界面卡顿,时常崩溃。运维平台工作量大,但是技术含量低,占用了大量软件技术人员。而且缺乏统一的设备对接移动端,安卓平台,IOS 平台,微信小程序都是不同的代码进行的实现,当系统升级时,要对应分别升级多个平台,工作量较大。

本平台为工业缝纫机物联网在线监控运维提供了有效的解决方案,首先运用低代码技术无需高级软件工程师即搭建了企业在线监控运维系统,使得高级技术人员从繁重的重复工作中的得以脱身去关注技术含量更高的方面。使用本平台快速便捷的实现了对缝纫机数据的实时监控如图 5.8 所示。同时依赖本平台可视化搭建页面功能以及表单设计器表格设计器与 workflow 设计器,通过可配置的方式完成设备管理、统计分析、综合管理功能,得到效果如图 5.9 所示。整个项目是运维监控与 BPM 的整合,包含有 100 余页面和超过 50 张表,从项目立项到交付上线仅花费了 5 个月,人月数为 8,相比传统研发 12.8 人月,节省了约 60%的人月数,交付速度提升了 3 倍。

得益于本设计第四章所述的基于 MMH3 算法和改进 Diff 算法的缓冲层方案,当数据量较大时监控界面数据也可以流畅的刷新展示。由于本平台发布时生成了 H5 网址,所以通过 H5 技术将移动端对接功能嵌入到了安卓外壳、IOS 外壳和微信小程序内,在更新迭代时只需要在低代码研发平台发布 H5,三个平台将自动完成同步更新,移动端界面如图 5.10 所示。



图 5.8 缝纫机物联网大数据平台

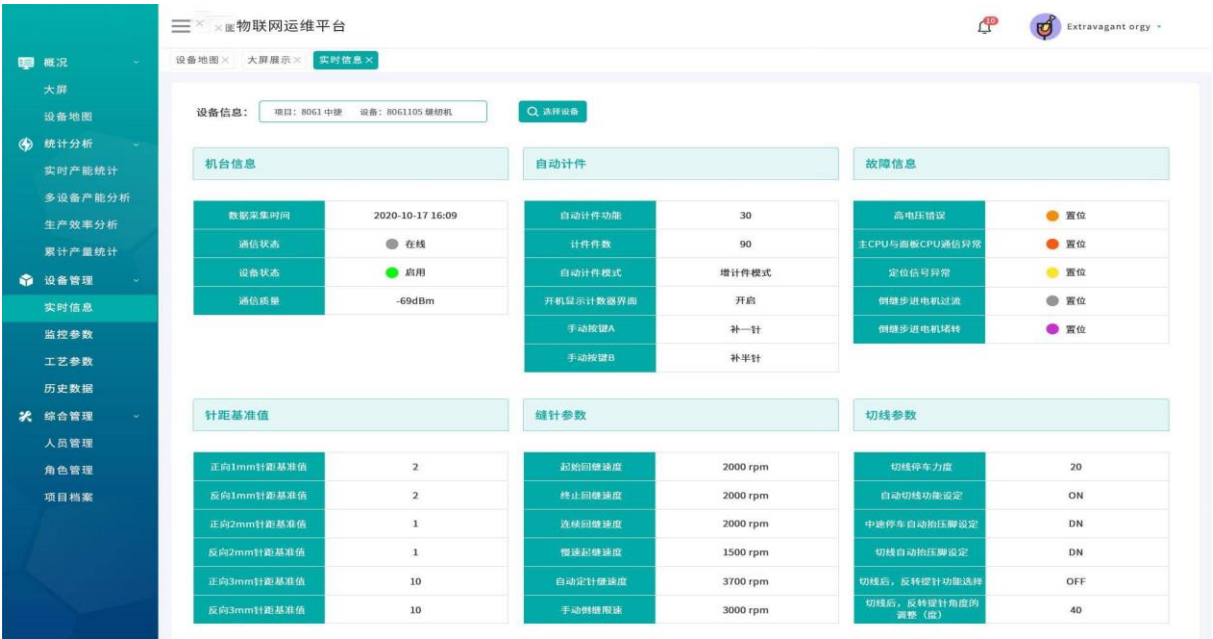


图 5.9 缝机物联网运维系统

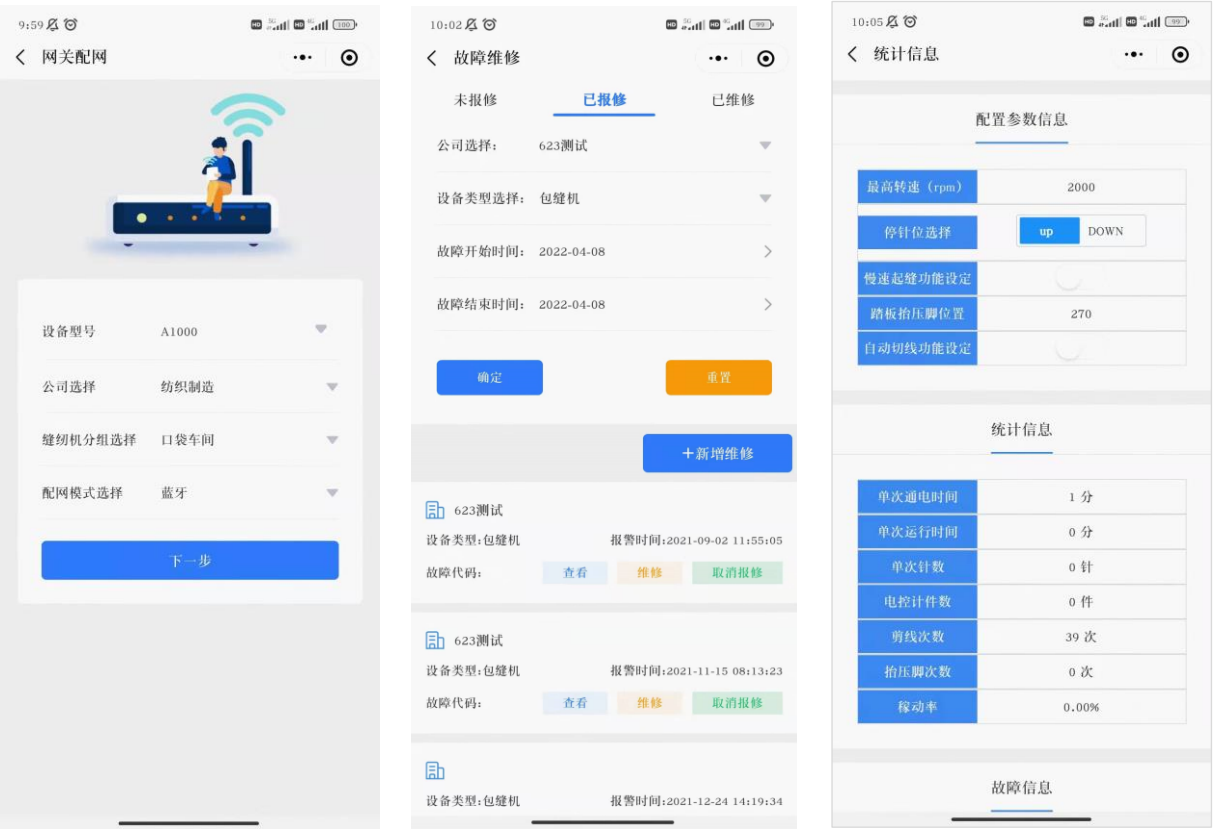


图 5.10 移动端设备对接界面

5.4 本章小结

在本章中介绍了在实际应用中使用本平台搭建的案例，并对案例在监控运维中遇到的问题进行讨论，分析了本平台应用在工业制造行业运维中的优势，综合来看本平台的优势可以归纳如下：

(1) 减少工作量，降低开发门槛。使用图形可视化界面拖动组件即可搭建应用。

(2) 开发效率高。使用封装好的行业模板以及行业标准元器件图形库可以快速搭建 Web 组态界面。常用物料可以形成组件资产，高效复用。

(3) 快速响应，迅速迭代。内部使用微服务-微前端架构，可以方便的集成老旧系统，当工艺流程或设备等变更时，只需要使用可视化配置的方式就可以完成系统迭代发布更新。

(4) 更精确的选区方案，搭建更加高效。使用改进混杂 CD 算法构建 BVH 树满足工业场景对选区精度的要求，基于改进 ACO 算法进行相交检测，使得搭建更高效。

(5) 更优秀的系统性能。内部建设了数据缓冲层，使用改进 MMH3 算法快速生成缓存摘要，改进 Diff 算法增量更新页面，提升用户体验。

第六章 总结与展望

6.1 工作总结

低代码技术的快速发展使得工业设备远程运维领域低代码构建与部署已成为可能。论文设计并实现了一种面向工业设备远程运维的低代码开发平台。平台成功落地到实际项目中，证明了其能有效的帮助企业快速构建企业级运维平台，实现商业价值。

本文所完成的主要工作内容如下：

(1) 自主研发了一套面向工业设备远程运维的低代码开发平台，利用低代码技术为工业硬件设备与运维软件系统之间架设桥梁。内部将复杂的底层框架和基础设施抽象化为图形化界面，使用者利用拖放型组件以及可视化配置就可以快速搭建工业设备远程运维系统。

(2) 提出基于混杂 CD 算法和改进 ACO 的选区方案，实现了高效准确的选区。详细讨论了方案的设计思路，与算法的改进方向，最后通过实验测试，验证方案的可行性和准确率的有效提升。

(3) 提出在远程服务器与客户端之间架设数据缓冲层。设计并实现了基于 MMH3 算法和 Diff 算法的缓冲层方案，针对方案设计提出基于 MMH3 算法计算文件缓存摘要、基于改进 Diff 算法的增量更新渲染方案，并对 MMH3 算法进行性能优化，结合应用场景对 Diff 算法进行改进，最后设计测试用例对本方案进行测试验证了方案的可行性和改进的有效性。

6.2 工作展望

由于作者的所学与精力限制，本平台尚存在着若干缺陷与不足之处，下面是对本平台设计过程的一点反思，希望在今后版本中有所提升：

(1) 针对生成代码包的使用场景

本论文中讨论的是非软件技术人员的低代码平台使用场景，并未提供代码包的生成，未来考虑到依然会有相当一部分软件技术人员使用本平台，将提供代码包生成的功能。

(2) 移动端集成

本论文中平台是 B/S 架构，所以可以使用 H5 的方式集成到移动端，但是 H5 集成在用户体验上较原生环境要差一些，后续将针对 APP，微信小程序等社交集成场景，提供原生支持。

(3) 个性化定制

本论文中所述低代码开发平台还未开发个性化定制服务，任何通用型软件都无法满足用户所有的个性化需求，所以后续将添加相应个性化服务接口，针对用户需求提供标准、量身定制的交付完全贴合业务需求的应用。

致谢

指缝太宽，时光太瘦，三年的研究生生涯转眼已经接近尾声，将近二十年的求学路，其中酸甜苦辣能与老师同学一同分享，是我的幸运，能在杭电结束学生时代，是我的荣幸。硕士研究生三年的工作、学习和生活经历，短暂却又充实，更使我受益良多，也将成为我认识中宝贵的精神财富。

在此要特别感谢我的人生导师陈科明副教授，陈老师从论文的选题、构思到最终定稿的各个环节都给予了细心指引与教导。陈老师严谨的治学态度、敏锐的学术思维、精益求精的工作态度以及丰富的工程实践经验是我终生学习的楷模。在此毕业之际，谨向我最敬爱的陈老师致以最诚挚的敬意和衷心的感谢。感谢我的同门徐志杰、吴文楷、刘剑、张百军、邢豪蔚、胡晓双。感谢你们给我鼓励，和我一路并肩向前；感谢我的师姐王奕怡和师兄周涛在学习和工作上提供的帮助和指引。

感谢我的父母，二十余载的疼爱与教育，全世界都在催着我长大，你们却总能捧我在手心，即便在几千公里外的外省求学，也不辞辛苦的为我奔波，为我付出，一直用最纯粹的爱来爱着我。虽然我只是一个平凡的女儿，但你们是世界上最好的父母。感谢我的男朋友蔡坤，感谢他在学业和工作上敦促我向前。感谢他平日的照顾和包容，在焦虑失意时给我安慰，在最艰难的时期一直在我身旁，有你的日子平淡也浪漫。

感谢论文评阅和答辩委员会的各位老师，感谢您们在百忙中对本文的审阅，并恳请提出宝贵的意见和批评。

最后，感谢一路磕绊却从没放弃的自己。

愿我们伟大的祖国早日战胜新冠疫情。

参考文献

- [1] 梁正, 李瑞. 数字时代的技术-经济新范式及全球竞争新格局[J]. 科技导报, 2020, 38(14): 142-147.
- [2] 朱金凤. 大道至简, 以快智胜——Mendix 助力中国企业赢在数字时代[J]. 电气时代, 2021(02): 11-13.
- [3] Bock Alexander C, Frank U. Low-Code Platform[J]. Business & Information Systems Engineering, 2021, 63(6): 46-47.
- [4] Ullrich C, Lata T, Geyer-Klingenberg J. Celonis studio - A low-code development platform for citizen developers[J]. CEUR Workshop Proceedings, 2021, 172(78): 2973.
- [5] Rymer R. The Forrester Wave™: Low-Code Development Platforms For AD&D Pros Q4 2017[R]. Massachusetts: Forrester Research, 2017: 21.
- [6] Paul V, Yefim N, Kimihiko I. Magic Quadrant for Enterprise Low-Code Application Platforms [R]. Stamford Connecticut: Gartner Group, 2020: 76-78+121.
- [7] 石菲. 低代码开发占领新常态市场[J]. 中国信息化, 2021(02): 11.
- [8] 任保平, 豆渊博. “十四五”时期新经济推进我国产业结构升级的路径与政策[J]. 经济与管理评论, 2021, 37(01): 10-22.
- [9] 付宇涵, 王丹, 柴雯, 王庆瑜. 中国工业互联网发展历程与展望[J]. 科技导报, 2021, 39(12): 116-120.
- [10] Raquel S, Óscar García P, Francisco F, Raul P. Low-Code as Enabler of Digital Transformation in Manufacturing Industry[J]. Applied Sciences, 2019, 10(1): 11-12.
- [11] Alamin M, Malakar S, Uddin G, et al. An Empirical Study of Developer Discussions on Low-Code Software Development Challenges[J]. ACM, 2021, 92(3): 28-29, 34.
- [12] Yifei Z, Ryan W. Toward low LDPC-code floors: a case study[J]. IEEE Transactions on Communications, 2009, 57(6): 1566-1573.
- [13] Salgueiro J, Ribeiro F, Metrôlho J. Best Practices for OutSystems Development and Its Influence on Test Automation[C]//World Conference on Information Systems and Technologies. Springer, Cham, 2021: 85-95.
- [14] 红山. 2021 低代码厂商 TOP35[J]. 互联网周刊, 2021(15): 16-17.
- [15] Martins R, Caldeira F, Sa F, et al. An overview on how to develop a low-code application using OutSystems[C]//International Conference on Smart Technologies in Computing, Electrical and Electronics (ICSTCEE). 2020, 20: 173-176.

- [16] 佟伟. 低代码软件开发平台 Mendix 落地中国市场[J]. 现代制造, 2021(02): 16-17.
- [17] Frank B. Mendix 低代码平台助力安全设计[J]. 软件和集成电路, 2022(01): 70-71.
- [18] Anirban G. Mendix low-code PaaS adds Industry Clouds for key business sectors[J]. InfoWorld.com, 2021(02): 76-77.
- [19] Simon B. Business analysts: Go no-code with Microsoft PowerApps[J]. InfoWorld.com, 2017(06): 9-11.
- [20] Soultan D, Bragg M, Thompson T, Hoover A. PO-1708 In-House Incident Learning System for FMEA Using Microsoft Sharepoints and Powerapps[J]. Radiotherapy and Oncology, 2021, 161(S1): 34-35.
- [21] Marcus W. The Rise of Low-Code Software [J]. Engineering, 2020, 6(09): 960-964.
- [22] 李争辉. 企业级无代码开发效率变革赋能业务增长[J]. 软件和集成电路, 2021(08): 60-61.
- [23] 毛德欢, 诸葛斌. 基于宜搭低代码开发高校财务管理系统[J]. 2021, 44(12): 70-74.
- [24] 郭卫丹. 低代码开发为构建“智慧校园生态圈”助力[J]. 中国信息技术教育, 2021(09): 72-77.
- [25] 刘宗凡, 邱元阳, 等. 深入评析低代码开发平台[J]. 中国信息技术教育, 2021(11): 80-86.
- [26] Peltonen S, Mezzalana L, Taibi D. Motivations benefits and issues for adopting Micro-Frontends: A Multivocal Literature Review[J]. Information and Software Technology, 2021, 136: 106571.
- [27] 杨彩芳. 基于微前端的旅行社业务支撑系统的设计与实现[D]. 北京: 北京邮电大学, 2019: 41-52.
- [28] 李学森, 李昭玥. 基于微前端架构的数字运营一点通平台[J]. 金融电子化, 2021(01): 71-72.
- [29] 胡如乐, 张亮, 张倩, 等. 基于微服务基础组件的前端开发技术应用研究[J]. 电子测试, 2020(20): 84-86.
- [30] Daojiang W, DongMing Y, Huan Z, Ye W, Daocheng H, Qiwen D. A Novel Application of Educational Management Information System based on Micro Frontends[J]. Procedia Computer Science, 2020, 176: 1567-1576.
- [31] 方燕琼, 唐升卫, 顾博川, 等. 适用于智能配电网前端微服务的跨系统集成平台优化[J]. 环境技术, 2021, 39(04): 142-146.
- [32] ZhiDong Z. Proceedings of 2019 Chinese Intelligent Automation Conference[J]. Springer-Verlag Singapore Pte Ltd, 2020, 586: 683-690.
- [33] 沈林强. 基于服务网格的物联网平台研究与实现[D]. 杭州: 浙江理工大学, 2019: 44-59.
- [34] 张玄弋, 张立斌. 智慧机场物联网平台架构设计[J]. 物联网技术, 2021, 11(10): 91-96.
- [35] 李新. 基于微服务架构的宽带缴费服务系统的设计与实现[D]. 北京: 北京交通大学,

- 2021: 32-41.
- [36] 黄华梅, 杨信廷, 杨宝祝, 等. 基于 AJAX 和 SVG 的组态软件 Web 发布模型[J]. 计算机工程与设计, 2020, 31(11): 2629-2633.
- [37] 张丽. 基于 Web 组态的实验室安全实时监测系统设计与实现[D]. 武汉: 华中师范大学, 2019: 46-51.
- [38] 王惠. 基于 SVG 技术的网页可视化视觉符号交互系统设计[J]. 现代电子技术, 2020, 43(16): 135-137+141.
- [39] 李超亚. 基于领域特定语言的智能数据可视化引擎[D]. 杭州: 浙江大学, 2016: 32-36.
- [40] 庾龙鑫. 基于 Web 的系统工程建模平台前端图形展示层的架构设计[D]. 杭州: 浙江大学, 2020: 22-25.
- [41] 李杰, 徐启禄. 基于云平台的城市轨道交通智能运维系统设计与应用[J]. 城市轨道交通研究, 2021, 24(08): 213-217.
- [42] 付亦文, 辛业薇, 白利宾, 等. 基于 OneNET 的 ZH 型强磁选机水压监测系统设计与实现[J]. 矿冶工程, 2020, 40(2): 67-69.
- [43] 冷波. 基于 B/S Web 架构组态监控平台的应用[J]. 自动化应用, 2019(09): 72-74.
- [44] 叶忠文, 黄鹏, 施金金. 基于 WebSocket 的 Web 实时通信系统[J]. 火力与指挥控制, 2014, 39(S1): 181-183.
- [45] 李垚, 张宇, 张惠樑. WebSocket 协议在 IoT 领域的应用优势[J]. 自动化应用, 2021(01): 58-60.
- [46] 樊天威. 基于 WebSocket 与 SVG 的在线组态监控系统开发[D]. 南京: 南京大学, 2018: 21-49.
- [47] 张永福. Web 前端开发中性能优化方案设计探讨[J]. 信息系统工程, 2020(11): 31-32.
- [48] 燕楠, 林支慨. 基于包围盒与碰撞的模型填充算法研究[J]. 软件工程, 2020, 25(03): 5-8.
- [49] 靳雁霞, 程琦甫, 张晋瑞, 齐欣, 马博, 贾瑶. 融合 DNN 与 AABB—圆形包围盒自碰撞检测[J]. 中国图象图形学报, 2020, 25(08): 1674-1683.
- [50] 干智峰. 基于 AABB 包围盒的双机械臂避碰技术的研究[J]. 科技视界, 2020(17): 95-98.
- [51] 张敏, 郑哲, 林君强. 基于 OBB 包围盒的图形检测算法[J]. 闽江学院学报, 2020, 41(05): 74-82.
- [52] 曹晶. 基于混合层次包围盒的碰撞检测算法的研究[D]. 西安: 西安科技大学, 2014: 39-58.
- [53] 李健, 王明月, 姚汝婧, 曹春玲, 胡雅婷. 大数据背景下混合层次包围盒碰撞检测算法的优化[J]. 吉林大学学报(理学版), 2017, 55(03): 673-678.
- [54] 李玉虎, 王宗彦. 基于混合层次包围盒碰撞算法的改进[J]. 华东交通大学学报, 2019, 36(06): 112-118.
- [55] Baiqiang G, Qiuping D. An improved optimal algorithm for collision detection of hybrid

- hierarchical bounding box[J]. *Evolutionary Intelligence*, 2021(prepublish): 12-16.
- [56] 张超. 二维图形连续变换算法分析及实现[J]. *现代计算机（专业版）*, 2007(11): 43-46.
- [57] 王姗姗. 几何形状约束下的二维图像变形[D]. 上海: 华东师范大学, 2009: 46-58.
- [58] 杜丽美, 李艳玲, 侯慧玲. 矩阵理论在图形变换中的应用[J]. *电子测试*, 2016(9): 63-64.
- [59] Rachmad F, Anton Y, Sunardi S. Penyusunan File Fingerprint untuk Berkas Jpeg/exif dengan Hash Function SHA512 dan Algoritma Boyer-Moore String Matching[J]. *Jurnal Edukasi dan Penelitian Informatika (JEPIN)*, 2020(1): 1-6.
- [60] 李夏梦, 潘广贞. 基于消息摘要算法第五版和 IDEA 的混合加密算法[J]. *科学技术与工程*, 2017, 17(9): 233-238.
- [61] 潘斌, 田雨, 李鹏飞, 周天舒, 李劲松. 图像加密技术在临床信息交换中的应用[J]. *中国数字医学*, 2014, 9(09): 63-65+69.
- [62] Hart G. Transcribing the Typescript[J]. *Contemporary Literature*, 2021, 61(2): 66-69.
- [63] Philip B. A survey on tree edit distance and related problems[J]. *Theoretical Computer Science*, 2004, 337(1): 839-844.

附 录

作者在读期间发表的学术论文及参加的科研项目

参加的科研项目：

- [1] 面向锯切装备产业提升的“智能一代”技术研究及产业化，浙江省科技计划项目（KYZ044321014），2020.07-2022.12
- [2] 缝纫机工业互联网平台开发，横向课题（KYH043121031M），2021.04-2022.03

发表和录用的学术论文：

- [1] 黄盼盼. 基于多平台的微远程组态监控模型[J]. 信息技术, 2021. （已发表）

申请的专利：

- [1] 陈科明,黄盼盼,蔡坤,邢豪蔚. 一种基于Web数据库的数据持久化存储方法[P]. 浙江省: CN114254235A, 2022-03-29. （已公开，2/4，导师第一发明人）