集成 EXTJS 联合 JSP 与 Vue3 的混合架构设计

邱明辰
南银法巴消费金融有限公司　江苏南京　210000

摘　要：为了解决企业级应用中因 EXTJS 停更引发的生态缺失、技术负债及人才断层等问题，为企业级应用累积了一套低成本渐进演化升级的迭代思路。本研究聚焦于企业级 Web 前端可视化集成，提出一种实现 EXTJS 联合 JSP 动态加载 VUE3 组件构建前端用户界面的配置方法，其具以下优势。（1）相比传统的 iframe 嵌入，本文所述的配置方法加载效率优异，可提高用户访问速度，跨框架状态同步延迟≤ 50ms。（2）在高交互模块，如数据可视化上卷下钻要求高页面采用 Vue3 动态加载，可实现实时交互与动画效果；（3）在开发成本方面，较全量重构节省 70% 成本，升级周期缩短至全量重构方案的 1/3，显著降低企业技术升级的成本与时间投入。（4）本文所述架构具备高度灵活性，支持动态切换组件加载策略，能够根据不同业务场景启用或禁用 Vue 模块。由此可见，该技术拥有广阔的应用空间。

关键词：JSP；Vue3；集成 EXTJS

前言：

EXTJS，通常被简称为 Ext，在 2006 年发布，是一个功能丰富的 JavaScript 框架。其基于纯 Htaml/CSS ＋ JS 技术，多用于前端用户界面的创建。由于 EXTJS 具有用户界面元素丰富、跨浏览器兼容性良好、面向对象且可扩展、响应速度高效、文档与示例丰富等优势，EXTJS 联合 JSP 曾在企业级用户界面构建中占主导地位，使部分应用对其产生了依赖 [1]。

2023 年 EXTJS 停止维护，对大量存量基于 EXTJS 联合 JSP 构建的系统造成了严重的冲击，使之面对技术断代的风险 [2]。但如采用新型框架进行重构，不但会对业务连续性造成影响，亦会为公司造成经济压力 [3]。在此背景下，须要一种方法，既能支撑存量 EXTJS 联合 JSP 构建的应用稳定运行，无缝集成如 VUE3 等现代前端框架组件的优势，也可从旧有系统中剥离出符合现代用户界面构建范式要求的子应用，进行架构解耦 [4]。

针对上述挑战，本文提出基于微前端的渐进式重构方案：通过构建 EXTJS 与 Vue3 的混合架构，在保留存量系统核心功能的同时，利用 Qiankun 微前端框架实现现代组件的无缝集成。该方案聚焦三层关键技术：首先设计沙箱隔离机制解决跨框架冲突，其次开发标准化桥接层实现 EXTJS 动态加载 Vue3 子应用，最后重构构建流程与部署体系。实践证明，该方法可有效降低的迁移成本，使传统系统获得模块化扩展能力，为企业级应用的技术演进提供可行路径。

1 整体方案设计

对企业级应用进行改造时，业务对技术的要求有二：（1）成本；（2）业务连续性。因此架构设计须满足下述要求：（1）实现前后端分离解耦；（2）新老技术栈无缝集成；（3）在确保存量业务稳定性的基础上，引入现代化能力。

表 1 为主应用与子应用的技术栈与职责边界，可知，本文所述配置方法延用了 EXTJS 联合 JSP 渲染的主应用，以此承载对存量业务逻辑，如：列表、路由、列表以及表单提交等；Vue3 子应用则主要用于支撑复杂度高的交互性模块。此外，还利用配置驱动的方法，实现了对于前端资源（Vue 组件）与后端服务（JSP）的分离部署，落地了部分前后端分离的微场景，减少了前端开发对 jsp 渲染模式的依赖。构成了以配置驱动的“双轨制”混合架构。

表 1　主应用与子应用的技术栈与职责边界

应用类型	技术栈	职责边界
主应用	EXTJS + JSP	路由管理、基础表单、列表展示等
子应用	Vue3 + Webpack	数据可视化穿透、高级的筛选器等

2 关键技术

本文所述配置方法的关键技术，有以下几点。1）动态组件加载引擎：基于现代前端框架的代码拆分能力，实现了 Vue3 及其他现代化框架组件的按需加载，精准适配不同业务场景对技术栈的动态需求。2）跨框架上下文通信：依托微前端引擎与 EXTJS–JSP 架构的深度集成，打通 EXTJS 与

Vue3 的状态同步机制，支持参数传递、数据联动等核心功能，保障跨框架数据交互的高效与准确。3）沙箱编译环境：通过 Webpack 定制化编译链，实现 Vue 组件在 EXTJS–JSP 体系下的独立编译与隔离运行，有效规避多框架融合时的逻辑与样式冲突。4）请求中间件：利用 Nginx 静态服务器进行资源代理与跨域处理，突破 B/S 架构的同源策略限制，确保运行时资源的无缝集成与稳定加载。

3 配置步骤

3.1 规划“混合”式工程目录结构，配置 EXTJS +JSP 访问 Vue3 目录的策略。

将传统 Java–web 项目改造为“Java 后端 + 局部微前端化”的混合架构。

图 1 为改造前架构，由表 1 可知改造前结构以 Java 后端源码，配套 Maven、Docker 等基础配置。改造后形成“混合”式工程结构，关键是在 Java 后端工程目录结构中布置了前端桥接层。

具体而言就是源码与编译输出的目录中均新增了 FedClient/ 目录（前端桥接层）；其中内置了微前端加载框架核心文件和异步模块加载配置文件；而原有 Java 后端代码、Maven 配置则保持不变。 然后 Java–web 项目构建输出的 WAR 包中会自动包含 webapp/ 目录中的内容，即前端桥接层 FedClient/ 目录便会在被 Tomcat 服务器的资源映射后允许被访问（如 http://localhost:8080/FedClient/）。

即：源码与编译输出的目录中均新增了 FedClient/ 目录（前端桥接层）；其中内置了微前端加载框架核心文件和异步模块加载配置文件；而原有 Java 后端代码、Maven 配置则保持不变。 Java–web 项目构建输出的 WAR 包中会自动包含 webapp/ 目录中的内容，即前端桥接层 FedClient/ 目录便会在被 Tomcat 服务器的资源映射后允许被访问。

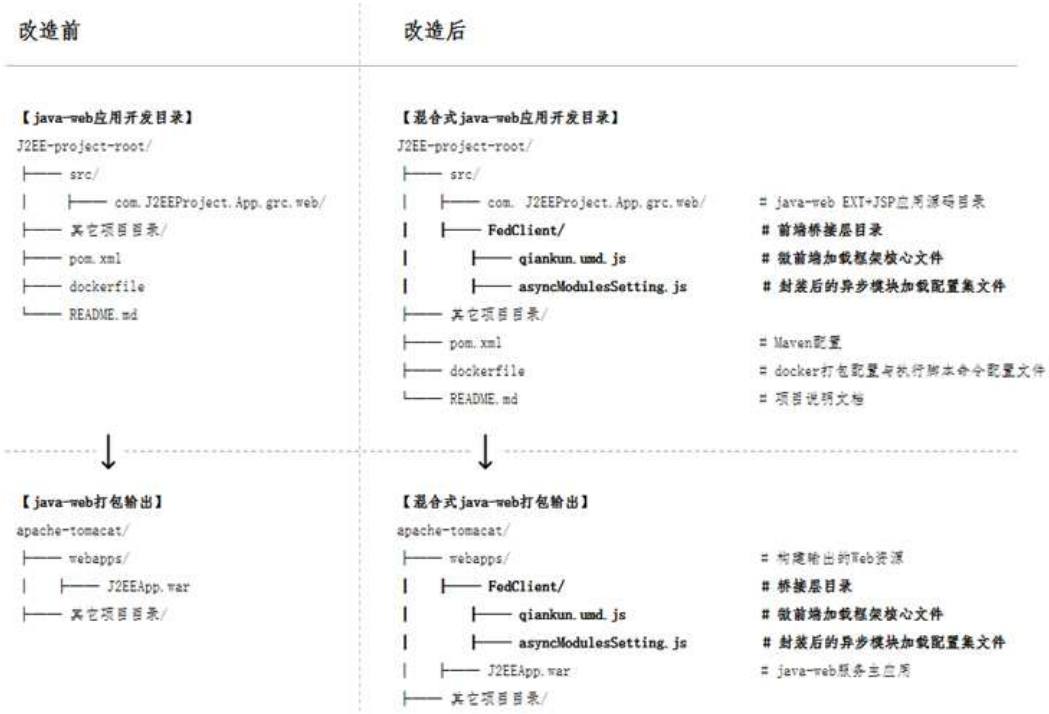

图 1 “混合”式 Java–web 工程目录结构

本文所述配置方法新增可独立开发、部署的 vue3 独立前端应用工程目录，用以开发、编译和打包可供 EXYJS 远程加载的 Vue3 组件。

最终该阶段的产出在成功具备了现代前端模块灵活加载、独立开发、独立部署能力的同时，还保留了 Java 后端的成熟生态，让项目具备更灵活的前后端协同能力。与此同时，从架构设计的扩展性来看，子应用承载的现代 SPA 模式前端框架也可根据实际情况平替成非 Vue3 框架（如 react 或 angualr）。

3.2 配置改造 EXTJS +JSP 工程，并引入”桥接层”配置跨框架通信机制，实现混合式渲染表 2 为 EXTJS–JSP 主界面入口文件微前端集成改造要点。

表 2 EXTJS–JSP 主界面入口文件微前端集成改造要点

改造 EXTJS 联合 JSP 工程源码 – 主界面入口 jsp 文件	
引入微前端引擎文件，作为集成 VUE3 应用的桥接层核心。	<script src=”/FedClient/qiankun.umd.min.js”></script>
封装后的异步模块加载配置集文件作为加载 Vue3 应用的配置驱动核心。	<script src=”/FedClient/asyncModulesSetting.js”></script>
HTML 中布置 Vue3 应用的 DOM 容器作为预加载 Vue3 资源的初始化。	<div data–controller=”app_FedClient”> 插槽
在入口 jsp 文件 </body> 标签外的 javascript 环境中进行预加载的调用。其中”asyncExt“是自定义的全局对象，作为桥阶层核心管理对象。	asyncExt.useFedClient(‘’).then(instance => {console.log(为预加载成功’); }, subExport => { console.log(‘重复创建 :’ , subExport); });

采用 UMD 规范引入微前端引擎“qiankun”，凭借 loadMicroApp 模式加载方式手动按需加载 Vue3 子应用。再对微场景的完整生命周期进行再次封装，使 Promise 配置管理 Vue3 异步加载的生命周期管理，增强代码可读性。通过自定义 DOM 属性配置”data–controller“进行组件容器的按需创建与展示位置进行控制。其中最为关键的是在 asyncModulesSetting.js 文件中实现了一款可进行配置驱动 EXTJS 联合 JSP 动态加载 VUE3 组件的全局“桥接层”静态类”asyncExt”。表 3 为 EXTJS–JSP 工程加载 Vue3 子应用的桥接层模块设计。

表 3 EXTJS–JSP 工程加载 Vue3 子应用的桥接层模块设计

布置 EXTJS 联合 JSP 工程加载 Vue3 子应用的桥接层 – asyncModulesSetting.js 文件	
A: 桥阶层单例工厂创建模块	asyncExt.module(‘FedClientFactory’ , options => { ... });
B: 桥阶层初始化配置模块	asyncExt.module(‘MicroModuleSetConfig’ , (microModule,props) => { ... })
C:Vue3 组件加载配置模块	asyncExt.module(‘useFedClient’ , options => { ... })

为落地配置驱动，须通过 asyncModulesSetting.js 文件提供的 asyncExt.module 方法进行注册三个核心模块。在 ExtJs 上下文中进行使用：通过 A: 桥阶层单例工厂创建模块可创建“桥阶层”对象单例，通过 B: 桥阶层初始化配置模块配置初始化后，任何地方调用均可获得。而 C:Vue3 组件加载配置模块用于在 Extjs 框架打开路由或面板时按需调用，动态获取 Vue3 组件进行共同构建用户界面。

3.3 Vue3 构建工具 Webpack 环境配置改造

在开发环境配置中 [H2.1]，通过 Webpack 设置静态资源目录并启用 Gzip 压缩，建立 9110 端口的本地服务，开启热更新功能；同时配置代理将 /JavaWeb 路径请求转发至 Tomcat 服务（8063 端口），通过修改主机头和添加 Access–Control–Allow–Origin: * 响应头解决跨域问题。为规避跨框架冲突 [H1.1]，采用沙箱编译链配置：JS 文件通过 Babel 转译并排除 node_modules 依赖，CSS 文件启用模块化处理并采用 [local]__[hash:base64:5] 哈希格式生成隔离类名，同时通过 DefinePlugin 注入 __SANDBOX__ 全局标识。在生产环境构建配置 [H3.1] 中，设置资源公共路径为 /FedClient/dist/，对 JS/CSS 文件添加 8 位完整哈希命名，图片资源采用 [name].[hash]–[ext] 格式；定义子应用名为 "Export"，以 UMD 模块格式输出，并声明 webpackJsonp_Export 全局变量供 ExtJS 调用。

3.4 Nginx 跨域解决方案

通过配置 Nginx 服务 [H4.1] 监听 9120 端口并绑定 localhost 域名，建立双路径代理：将根路径 / 代理至 J2EE 应用服务（8080 端口），同时将 /FedClient/dist/ 路径代理至 Vue3 应用资源地址；关闭代理重定向响应头，配置自定义错误页处理 50x 系列错误。最终实现通过 http://127.0.0.1:9120/ 统一访问由 ExtJS 动态加载 Vue3 组件构建的前端界面，彻底解决跨域问题。

结语：本研究针对 EXTJS 停止维护后企业级 Web 应用面临的技术断代风险，提出了一种创新的混合架构设计，实现了 EXTJS–JSP 传统架构与 Vue3 现代框架的无缝集成。通过动态组件加载引擎、跨框架通信机制、沙箱编译环境等关键技术，成功构建配置驱动的“双轨制”混合架构，保留存量系统的业务稳定性，显著提升了高交互模块的用户体验，降低 70% 的全量重构成本。由此可见，本文所述架构具有一定的推广应用价值。

参考文献：

[1] 徐荣荣 , 邓雾恒 . 基于 ExtJS 的云南省高速公路基础参数管理系统的设计与实现 [J]. 中国管理信息化 ,2022,25(02):184–186.

[2] 吴建平 .ExtJS+SSH2 框架在高职院校科研信息管理系统中的应用研究 [J]. 现代职业教育 ,2021,(43):92–93.

[3] 邢飞燕 , 沈春梅 , 施佺 . 基于 ExtJS 的高校同等学力研究生信息管理系统设计与实现 [J]. 淮北职业技术学院学报 ,2021,20(03):97–10226.

[4] 李超 . 基于 ExtJS 的风电场监测系统设计与实现 [J]. 江苏科技信息 ,2020,37(06):38–41.

作者简介：邱明辰（1988—），男，汉族，南京，本科，研究方向为信息技术。