计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2020,29(9):126−130 [doi: 10.15888/j.cnki.csa.007616]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

微前端化微应用管理控制台①

刘一田, 曹一鸣

(南京南瑞信息通信科技有限公司, 南京 210003)
通讯作者: 刘一田, E-mail: liuyitian@sgepri.sgcc.com.cn

摘　要: 微服务架构下开发时技术栈分离提升了开发效率以及运行时服务编排能力, 但微应用数量倍增导致微应用管理控制台集成复杂度加大, 跨微应用之间通信及操作交互体验差. 本文给出一种微前端解决方案, 结合典型微前端框架、管理控制台事件总线、微应用动态路由通信、微应用运行时上下文资源分离等技术机制, 提高了管理控制台柔性与运行效率, 降低了开发和运维成本, 较好地实现了微应用管理控制台管理优化目标. 经试验评估, 柔性增强且代价更小, 并在实际项目中验证了解决方案的有效性.

关键词: 微前端; 管理控制台事件总线; 微应用生命周期

引用格式: 刘一田,曹一鸣.微前端化微应用管理控制台.计算机系统应用,2020,29(9):126–130. http://www.c-s-a.org.cn/1003-3254/7616.html

Micro Frontends-Based Micro-Applications Management Console

LIU Yi-Tian, CAO Yi-Ming

(Nanjing NARI Information and Communication Technology Co. Ltd., Nanjing 210003, China)

Abstract: Technology stack separation during development under the microservice architecture improves development efficiency and runtime service orchestration capabilities. However, the multiplication of micro-applications results in the increase of integration complexity of micro application management console and the poor experience of communication and operation interaction across micro-applications. This paper presents a micro frontends solution, combined with typical micro frontends framework, management console eventbus, micro-applications routing communication, separation of context resources in runtime and other technical mechanisms, the flexibility and operating efficiency of the management console is improved, the development and operation and maintenance costs are reduced, and the management optimization objective of the management console is better realized. Through the test evaluation, the flexibility is enhanced and the cost is smaller, the effectiveness of the solution is verified in the actual project.

Key words: micro frontends; management console eventbus; micro-applications life cycle

1　引言

传统的服务管理控制台和应用门户在面临业务快速发展之后, 出现了单体应用进化成巨石应用的问题, 随着微服务架构的广泛应用, 业务应用被划分为更小颗粒度的微应用, 不同的微应用可以采用不同的技术栈实现, 更好地提升了业务开发效率和需求响应能力, 从企业级管理需求角度, 管理控制台需要聚合微应用并提供管理入口, 支持租户按需定制 SaaS 层应用门户, 对已在线上运行的项目, 支持低成本地接入微应用门户, 而不需要对现有开发和部署流程做大量兼容改造; 支持不同技术栈微应用分布式开发部署和运行时上下文资源分离, 但允许开发阶段微应用之间良好联调, 能

① 基金项目: 南京南瑞信息通信科技有限公司科技项目 (5246DR200014)
Foundation item: Science and Technology Project of Nanjing NARI Information and Communication Technology Co. Ltd. (5246DR200014)
收稿时间: 2020-02-22; 修改时间: 2020-03-24; 采用时间: 2020-04-03; csa 在线出版时间: 2020-09-04

保证单页应用的操作流程不中断和体验流畅性. 目前, 微应用管理控制台及业务服务微应用的前端技术栈实现是主流的 Vue、React、Angular 等前端框架, 为了解决上述问题, 主流解决方案有 3 种: (1) 基于 iframe 的页面嵌入. 将不同的微应用的入口页面通过 iframe 嵌入到门户菜单中. 集成方式简单, 缺点是页面切换无法保持路由状态, 父子应用通信不稳定, 存在潜在性能瓶颈. (2) 基于 Nginx 导航路由. 基于 Nginx 中微应用路由前缀的不同将请求路由到不同的微应用. 优点是支持微应用的技术栈独立, 缺点是页面跳转有时会闪屏, 动态扩展需变更 Nginx 配置. (3) 基于微前端[1]Single-SPA 框架[2]. 即通过由独立交付的多个微应用动态组成整体管理控制台的架构模式. 将前端应用分解成一些更小、更简单的能够独立开发、测试、部署的小块, 从用户角度仍然是无缝管理体验. 管理控制台应用异步加载嵌入其内部的微应用的入口文件, 动态创建微应用装载点. 优点是支持微应用的开发技术栈独立, 缺点是微应用之间开发联调复杂. 文献 [3] 描述的微前端应用以 JS 文件入口方式实现微应用加载, 这种方式会导致微应用之间运行时资源上下文混淆, 增加了开发和维护的复杂度.

综合上述问题及方案, 结合项目中微应用数量不断迭代递增、父子微应用通信、微应用资源按需加载和卸载、运行时上下文资源分离等需求, 本文给出了微前端化管理管理控制台的解决方案. (1) 提供基于管理控制台消息总线的路由注册机制, 管理各个微应用的生命周期, 传递路由消息. (2) 定义微应用渲染入口规范, 提供微应用注册接入机制. (3) 提供微应用加载工具库, 监听微应用地址端口路由变化并快速加载或卸载微应用, 加载时读取微应用配置并为微应用的实例提供门户占位装载点, 卸载时反向移除微应用相关资源.

2 微前端化微应用管理控制台设计

微前端管理控制台由管理门户和微服务对应的管理微应用组成, 如图 1 所示. 它们共享公共依赖库以实现共享数据和资源依赖, 进而降低开发管理成本, 管理控制的权限刷新令牌采用 REDIS 集群存储, 访问令牌基于本地缓存存储[4]. 本文中讨论的微应用基于 Node.js 环境部署运行, 文献 [5] 论证了前端应用基于 Node.js 环境在微应用系统运行期的相对优势.

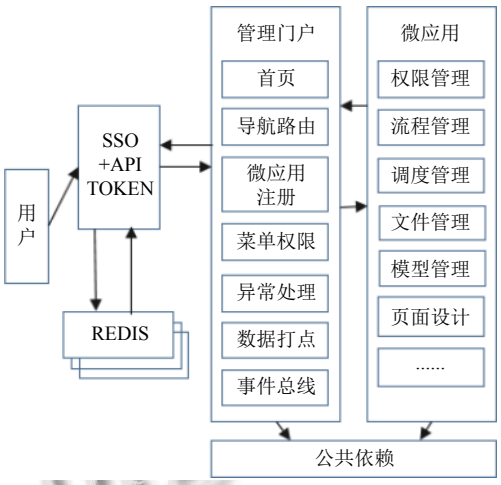


图 1 微前端管理控制台框架

微前端化管理控制台的交互过程及关键处理机制包含分为以下 7 个方面.

(1) 用户登录. 用户访问管理门户首页, 管理门户鉴别是否已存储已登录的本地令牌, 如无则跳转到单点登录页面.

(2) 令牌存储. 登录成功后分别记录用户访问令牌到本地缓存, 记录刷新令牌到 REDIS 中, 其中 REDIS 采用集群部署方式.

(3) 管理门户微应用启动时加载所管理的微应用配置. 配置包括微应用名称、页面入口地址和渲染方法引用, 认证登录成功后根据配置跳转到默认微应用页面.

(4) 微应用注册及加载. 微应用注册时, 采用 HTML 作为集成入口, 避免 JS 入口方式资源未完全加载而异步构建子微应用容器节点, 可能导致渲染失败等问题, 微应用运行加载时, 管理门户微应用注册模块通过提取 HTML 获取当前子微应用的静态资源, 同时将微应用 HTML 文档作为子模板节点添加到管理门户的页面模板容器中, 继而触发微应用页面渲染.

(5) 管理路由事件. 提供导航路由模块, 通过监听不同微应用的启动端口侦测浏览器地址变化事件, 在所管理的微应用路由触发或切换时, 在缓存中保留切换前的微应用完整的环境上下文状态, 动态加载目标路由微应用, 待再次路由返回时还原之前状态的微应用上下文运行时资源, 确保微应用切换时良好交互体验.

(6) 微应用模块跨应用依赖. 为了更好地管理微应用, 增加微应用的生命周期事件监听及相应处理函数, 并通过 Webpack 的 umd 打包格式中的全局导出模式

导出微应用的生命周期事件监听[6], 从而获取微应用的内置模块导出和跨微应用模块导入.

(7) 微应用资源边界上下文分离. 监听微应用的加载和卸载事件, 在门户路由系统跨微应用切换时, 在监听的路由事件中重新加载或卸载微应用上下文中的 JS 脚本和样式表等资源.

3 微前端化微应用管理控制台实现

图 1 中微前端化管理控制台由管理门户微应用和众多业务管理微应用组成, 管理门户采用 Vue2+ElementUI[7] 实现, 其中管理门户微应用作为父应用负责其它微应用的注册和卸载, 管理门户微应用目录结构如图 2 所示, 微应用基于模块化工程方式构建[8], 父应用基于 HTML5 规范的历史记录对象状态变更机制触发微应用路由的切换, 在父应用视图模板中置入子微应用的宿主容器, 作为子微应用在父应用容器中的占位符. 并需要实现以下 3 种框架机制: (1) 微应用注册、加载和卸载机制; (2) 微应用通信及交互机制; (3) 微应用资源共享和运行时上下文资源分离机制.

```
管理控制台
    |--- 权限管理
    |     |--- 本地资源
    |           |--- 内部公共资源
    |--- 模型管理
    |     |--- 本地资源
    |           |--- 内部公共资源
    |--- 流程管理
    |     |--- 本地资源
    |           |--- 内部公共资源
    |--- … …
    |--- 外部公共资源库
```

图 2 微前端化微应用管理控制台工程目录

3.1 实现微应用注册和卸载机制

由于控制台管理采用延迟加载微应用模式, 当浏览器重新加载时, 管理控制台的微应用资源也会重新被异步再次加载, 由于此时门户的路由导航已经触发微应用生命周期启动阶段事件, 但微应用运行时上下文 JS 脚本和样式等静态资源不能确保已经全部加载和正常解析完毕, 间接导致了路由记录缓存里找不到相对应的导航规则, 最终导致路由切换时出现异常而无法正常展示

因此, 注册微应用时, 需要基于管理控制台前端事件总线, 和父子微应用生命周期不同阶段钩子事件实现父子微应用及子微应用之间的注册、加载、卸载等交互, 管理微应用运行时上下文状态缓存, 微应用注册过程的形式化描述如图 3 所示.

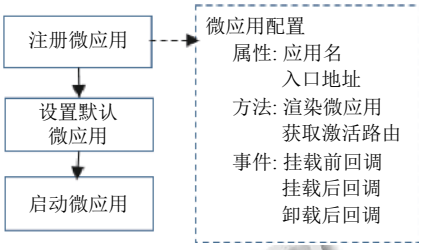


图 3 微应用注册过程

首先, 在父应用中根据微应用配置规约定义每个子微应用的加载入口页面 HTML 资源属性、渲染及路由方法、微应用生命周期事件等; 父微应用需要先加载每个子微应用的入口资源, 待入口资源加载完毕, 确保子微应用的路由系统注册进父微应用之后, 再由子微应用内部的路由系统接管路由改变事件. 同时在子微应用路由卸载时, 父微应用触发相应的销毁事件, 子微应用在监听到该事件时, 调用自己的卸载方法卸载自身运行时上下文资源; 其次, 设置默认的子微应用作为父应用的默认路由规则, 对默认微应用进行预加载, 并定义默认微应用加载完成后的回调事件以支持父微应用完成后的业务定制扩展; 最后, 启动全局路由, 根据全局路由规则切换微应用上下文资源、渲染微应用展现和卸载清理.

3.2 微应用通信及交互机制

微应用之间的通信及交互分为父子微应用、微应用内部、微应用之间等 3 种通信及交互机制. 其中父子微应用及微应用之间的交互基于管理控制台事件总线实现, 如图 4 所示.

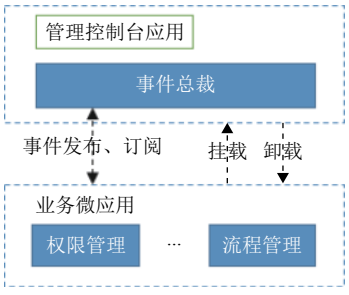


图 4 跨微应用通信及交互

3.2.1 父子微应用通信

(1) 父微应用传递子微应用的方式. 父微应用中注

册子微应用时, 在子微应用的配置中加入消息实体属性, 通过该属性值的修改, 动态将消息通知给子微应用.

(2) 子微应用传递父微应用的方式. 子微应用中导出挂载前回调事件, 在事件参数中, 将子微应用待传递的参数对象置入, 发布到父微应用的事件总线上, 父微应用监听到消息后, 从其事件总线上获取子微应用的数据对象. 形式化描述如图 5 所示.

```
// 父应用通知子应用
let app1Data = {// 定义传入子微应用的对象
        data:{ hasToken: false },
        fn:[{name: “msg”}]}
registerMicroApps(//微应用注册配置定义
    [{ name: "app1",entry: "//ip:port",
     render, activeRule: genActiveRule("/app1")
     props:app1Data //待传递对象}])
```

```
// 子微应用传递信息给父微应用
export async function bootstrap(props = {}) {
Array.isArray(props.fn) && props.fn.map(i => {
    Vue.prototype[i.name] = i[i.name]
}); }
```

图 5 父子微应用通信机制

3.2.2 微应用内部及之间通信

定义基于微应用管理门户主窗体的对象全局变量, 在子微应用内部匹配切换路由时, 路由管理通过 Vue Router 实现, 路由设置为 HTML5 历史模式, 如果侦听到路由中包含了全局变量中指定的微应用路由, 则进行跨微应用切换, 否则进行微应用内部页面的路由切换.

当源微应用准备发消息给目标微应用时, 通过父微应用的事件消息总线进行指定主题事件的订阅和发布, 当源微应用完成指定操作后要跳转到目标微应用时, 则通过浏览器对象模型提供的历史记录对象的状态改变方法, 传入目标微应用名称和入口参数, 触发微应用之间路由的强制跳转和消息通信传递.

3.3 微应用资源共享和运行时上下文分离

3.3.1 实现微应用资源及数据共享机制

(1) 运行时数据共享. 全局的数据一般会存储在父微应用中, 然后子微应用可以直接共享, 比如: 当前登录用户及隶属组织、国际和本地化配置、API 令牌等, 简单的数据共享可以直接挂载在门户微应用主窗体上即可, 为了让每个子微应用使用全局服务和模块内服务一致, 通过在父微应用中实例化这些服务, 然后在每个子微应用的入口模块中将该实例引用设置为子微应用的全局变量, 方便让子微应用业务开发人员共享使用这些公共服务数据.

(2) 开发环境公共依赖共享. 开发环境建立公共资源的目的是减少开发时的重复定义, 并避免多微应用修改带来的管理维护难题, 因此, 本文定义了图 2 中的目录规范以规约公共资源定义及引用, 在微应用部署时, 抽取微应用公共依赖库避免类库重复打包, 减少打包体积, 从而有效提升运行时效率.

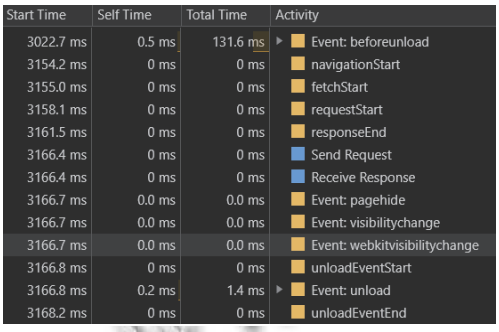
3.3.2 实现运行时微应用上下文分离

微应用的运行时上下文资源模型可以表示为一个五元组, 形式化简述为 APPCXT=<JS,CSS,BOM,T,R>, 其中 JS 和 CSS 分别表示微应用的静态资源, BOM 是浏览器对象模型, T 是页面模板, R 表示当前微应用路由, 运行时上下文表示了当前路由下 BOM 负责基于页面模板将 JS 和 CSS 资源解析为可交互的页面, 为了确保路由切换后的 BOM 性能最优和内存占用最小化, 在微应用之间导航切换时, 通过监听不同微应用端口的地址改变, 在微应用生命周期的启动及装载两个阶段的事件触发时, 记录下每个微应用的实时上下文资源状态, 当微应用通过路由导航切换出去时, 将微应用上下文状态还原至当前微应用生命周期开始之前的阶段状态, 以避免微应用切换时历史微应用资源未及时卸载导致的内存泄露等问题. 当微应用被再次导航回来时, 即时通过提取和加载之前缓存微应用上下文状态进行上下文恢复, 这样微应用切换上下文时就有效实现了运行时 JS 等资源的分离. 为了保证不同微应用切换后样式展示效果能动态无缝切换生效, 在微应用加载时采用了基于 HTML 入口资源的方式, 在微应用卸载后, 浏览器的特性本身会同时卸载掉该微应用的运行时样式表, 从而实现了微应用运行时样式表的动态插入和移除, 保证了切换微应用上下文后的微应用运行时上下文状态一致性和资源分离.

4 案例实践与评估

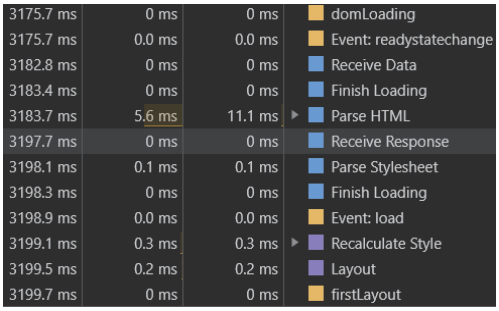
为验证本文微前端化微应用管理控制台方案, 在国网某省公司调控云公共服务管理平台测试环境中做了案例部署和评估. 案例环境搭建在 3 台 8 核 32 GB 内存的浪潮服务器 (SA5248L) 上, 网关、服务注册、工作流、权限、模型驱动、文件管理、页面设计等微服务均采用容器化部署, 容器副本数均设置为 3, 管理控制台各微应用前端管理微应用采用独立容器化部署, 容器副本数均设置为 2, 所有微应用在容器中基于 Node.js

环境运行, 其中工作流微应用基于 Angular 技术栈实现, 其它微应用基于 Vue 技术栈实现. 部署实例以 K8S 部署的 Docker 容器运行. 测试评估点分别从不同前端技术栈的微前端化集成、微应用通信及交互、运行时上下文资源分离、性能等方面进行验证. 浏览器采用 Chrome78, 通过 Chrome 开发工具的性能分析工具记录, 跨微应用路由切换时, 卸载源微应用的打点记录日志如图 6 所示, 加载目标微应用打点日志如图 7 所示.



Start Time	Self Time	Total Time	Activity
3022.7 ms	0.5 ms	131.6 ms	Event: beforeunload
3154.2 ms	0 ms	0 ms	navigationStart
3155.0 ms	0 ms	0 ms	fetchStart
3158.1 ms	0 ms	0 ms	requestStart
3161.5 ms	0 ms	0 ms	responseEnd
3166.4 ms	0 ms	0 ms	Send Request
3166.4 ms	0 ms	0 ms	Receive Response
3166.7 ms	0.0 ms	0.0 ms	Event: pagehide
3166.7 ms	0.0 ms	0.0 ms	Event: visibilitychange
3166.7 ms	0.0 ms	0.0 ms	Event: webkitvisibilitychange
3166.8 ms	0 ms	0 ms	unloadEventStart
3166.8 ms	0.2 ms	1.4 ms	Event: unload
3168.2 ms	0 ms	0 ms	unloadEventEnd

图 6 跨微应用路由切换之卸载源微应用



3175.7 ms	0 ms	0 ms	domLoading
3175.7 ms	0.0 ms	0.0 ms	Event: readystatechange
3182.8 ms	0 ms	0 ms	Receive Data
3183.4 ms	0 ms	0 ms	Finish Loading
3183.7 ms	5.6 ms	11.1 ms	Parse HTML
3197.7 ms	0 ms	0 ms	Receive Response
3198.1 ms	0.1 ms	0.1 ms	Parse Stylesheet
3198.3 ms	0 ms	0 ms	Finish Loading
3198.9 ms	0.0 ms	0.0 ms	Event: load
3199.1 ms	0.3 ms	0.3 ms	Recalculate Style
3199.5 ms	0.2 ms	0.2 ms	Layout
3199.7 ms	0 ms	0 ms	firstLayout

图 7 跨微应用路由切换之加载目标微应用

在监测过程中, 管理控制台在 Chrome 浏览器中耗费 177 ms 左右完成了路由切换, 切换过程贯穿了预定义的前后微应用生命周期加载和卸载等钩子事件, 操作过程符合用户预期体验, DOM 结构、静态资源和微应用上下文相适配, 无页面死链现象, 反复压测下无浏览器内存泄露现象, 符合预定义的测试验证目标.

在实验过程中, 采用的容器化微应用分布式部署方式有一定技术难度, 亟需通过自动化向导部署方式提升运维便捷性.

5 结束语

本文研究了微前端的微应用注册和卸载、微应用通信和交互、微应用上下文资源分离等技术, 在此基础上, 设计了微前端化管理控制台解决方案, 给出了基于微应用生命周期不同阶段事件钩子的微应用注册和卸载机制、基于管理控制台事件总线的微应用通信与交互机制、跨应用共享变量事件注入及 HTML 入口的资源共享、运行时上下文资源分离机制等创新点, 阐述了该方案的架构设计及关键实现技术. 最后, 以国家电网某省公司调控云公共服务管理平台微应用集成为背景, 给出了微前端化微应用管理控制台的应用验证评估, 评估结果、效率优化及生产试运行实践表明, 该框方案提升了微服务架构下微应用管理的开发效率, 实现了管理控制台微应用即插即用机制和跨微应用之间业务操作连续性, 提高了国网调控云公共服务管理平台应用服务水平. 后续将针对遗留问题持续改进优化该解决方案.

参考文献

1 Geers M. Micro FrontEnds: Extending the microservice idea to frontend development. https://micro-frontends.org/.

2 Single-Spa. https://github.com/single-spa/single-spa.

3 杨彩芳. 基于微前端的旅行社业务支撑系统的设计与实现 [硕士学位论文]. 北京: 北京邮电大学, 2019.

4 刘一田, 林亭君, 刘士进. 柔性微服务安全访问控制框架. 计算机系统应用, 2018, 27(10): 70–74. [doi: 10.15888/j.cnki.csa.006568]

5 单茹楠. 基于微服务化前端架构的研究与实现 [硕士学位论文]. 武汉: 武汉大学, 2018.

6 Webpack. https://webpack.js.org/configuration/output.

7 Element. https://element.eleme.cn/#/zh-CN.

8 周俊鹏. 前端工程化: 体系设计与实践. 北京: 电子工业出版社, 2018.