


可能是你见过最完善的微前端解决方案



kuitos
qiankun author / mobxjs team

关注他

1,655 人赞同了该文章

Techniques, strategies and recipes for building a modern web app with multiple teams using different JavaScript frameworks. — Micro Frontends

前言

TL;DR

想跳过技术细节直接看怎么实践的同学可以拖到文章底部，直接看最后一节。

目前社区有很多关于微前端架构的介绍，但大多停留在概念介绍的阶段。而本文会就某一个具体的类型场景，着重介绍微前端架构可以带来什么价值以及具体实践过程中需要关注的技术决策，并辅以具体代码，从而能真正意义上帮助你构建一个生产可用的微前端架构系统。

而对于微前端的概念感兴趣或不熟悉的同学，可以通过搜索引擎来获取更多信息，如 [知乎上的相关内容](#)，本文不再做过多介绍。

▲ 赞同 1655 ▼ ● 134 条评论 ↗ 分享 ❤ 喜欢 ★ 收藏 📄 申请转载 ...

两个月前 Twitter 曾爆火；
身我们今天不做过多评论(后面可能会写篇文章来回顾一下)，有兴趣的同学可以通过[这篇文章](#)了解一二。



微前端的价值

微前端架构具备以下几个核心价值：

- 技术栈无关 主框架不限制接入应用的技术栈，子应用具备完全自主权
- 独立开发、独立部署 子应用仓库独立，前后端可独立开发，部署完成后主框架自动完成同步更新
- 独立运行时 每个子应用之间状态隔离，运行时状态不共享

微前端架构旨在解决单体应用在一个相对长的时间跨度下，由于参与的人员、团队的增多、变迁，从一个普通应用演变成一个巨石应用(Frontend Monolith)后，随之而来的应用不可维护的问题。这类问题在企业级 Web 应用中尤其常见。

针对中后台应用的解决方案

- 单实例：即同一时刻，只有一个子应用被展示，子应用具备一个完整的应用生命周期。通常基于 url 的变化来做子应用的切换。
- 多实例：同一时刻可展示  子应用。通常使用 Web Components 方案来做子应用 ，子应用更像一个业务组件而不是应用。

本文将着重介绍单实例场景下的微前端架构实践方案（基于 [single-spa](#)），因为这个场景更贴近大部分中后台应用。

行业现状

传统的云控制台应用，几乎都会面临业务快速发展之后，单体应用进化成巨石应用的问题。为了解决产品研发之间各种耦合的问题，大部分企业也都会有自己的解决方案。笔者于17年底，针对国内外几个著名的云产品控制台，做过这样一个技术调研：

产品	架构(截止 2017-12)	实现技术
google cloud	纯 SPA	主 portal angularjs，部分页面 angular(ng2)。
aws	纯 MPA 架构	首页基于 angularjs。各系统独立域名。
七牛	SPA & MPA 混合架构	入口 dashboard 及 个人中心模块为 spa，使用同一 portal 模块（Angularjs(1.5.10) + webpack）。其他模块自治，或使用不同版本 portal，或使用其他技术栈。
又拍云	纯 SPA 架构	基于 angularjs 1.6.6 + ui-bootstrap。控制台内容较简单。
ucloud	纯 SPA 架构	angularjs 1.3.12

知乎 @kuitos

MPA 方案的优点在于 部署简单、各应用之间硬隔离，天生具备技术栈无关、独立开发、独立部署的特性。缺点则也很明显，应用之间切换会造成浏览器重刷，由于产品域名之间相互跳转，流程体验上会存在断点。

SPA 则天生具备体验上的优势，应用直接无刷新切换，能极大的保证多产品之间流程操作串联时的流程性。缺点则在于各应用技术栈之间是强耦合的。

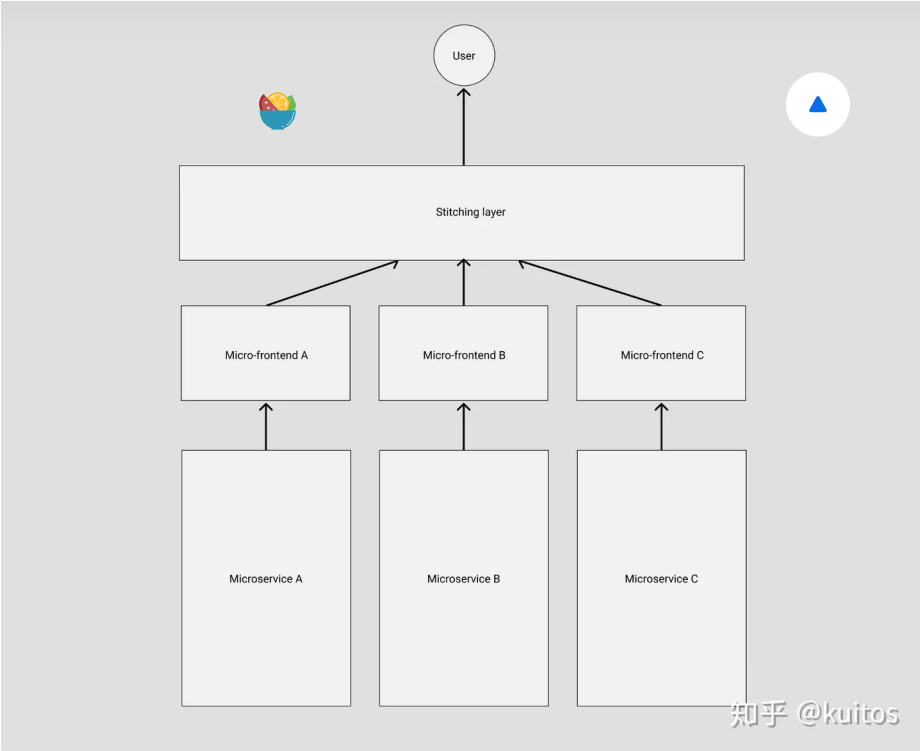
那我们有没有可能将 M 呢？

jsconf china 2016 大会上，ucloud 的同学分享了他们的基于 angularjs 的方案（[单页应用“联邦制”实践](#)），里面提到的“联邦制”概念很贴切，可以认为是早期的基于耦合技术栈的微前端架构实践。

微前端架构实践中的问题

可以发现，微前端架构的优势，正是 MPA 与 SPA 架构优势的合集。即保证应用具备独立开发权的同时，又有将它们整合到一起保证产品完整的流程体验的能力。

这样一套模式下，应用的架构就会变成：



Stitching layer 作为主框架的核心成员，充当调度者的角色，由它来决定在不同的条件下激活不同的子应用。因此主框架的定位则仅仅是：**导航路由 + 资源加载框架**。

而具体要实现这样一套架构，我们需要解决以下几个技术问题：

路由系统及 Future State

我们在一个实现了微前端内核的产品中，正常访问一个子应用的页面时，可能会有这样一个链路：



此时浏览器的地址可能是 `https://app.alipay.com/subApp/123/detail`，想象一下，此时我们手动刷新一下浏览器，会发生什么情况？

由于我们的子应用都是 lazy load 的，当浏览器重新刷新时，主框架的资源会被重新加载，同时异步 load 子应用的静态资源，由于此时主应用的路由系统已经激活，但子应用资源可能还没有完全加载完毕，从而导致路由注册表里发现没有能匹配子应用 `/subApp/123/detail` 的规则，这时候就会导致跳 NotFound 页或者直接路由报错。

解决思路也很简单，我们需要设计这样一套路由机制：

主框架配置子应用的路由为 `subApp: { url: '/subApp/**', entry: './subApp' }`，则当浏览器的地址为 `/subApp/abc` 时，框架需要先加载 `entry` 资源，待 `entry` 资源加载完毕，确保子应用的路由系统注册进主框架之后，再去由子应用的路由系统接管 `url change` 事件。同时，在子应用路由切出时，主框架需要触发相应的 `destroy` 事件，子应用在监听到该事件时，调用自己的卸载方法卸载应用，如 `React` 场景下 `destroy = () => ReactDOM.unmountAtNode(container)`。

要实现这样一套机制，我们可以自己去劫持 `url change` 事件从而实现自己的路由系统，也可以基于社区已有的 `ui router library`，尤其是 `react-router` 在 `v4` 之后实现了 `Dynamic Routing` 能力，我们只需要复写一部分路由发现的逻辑即可。这里我们推荐直接选择社区比较完善的相关实践 `single-spa`。

App Entry

解决了路由问题后，主框架与子应用集成的方式，也会成为一个需要重点关注的技术决策。

构建时组合 VS 运行时组合

微前端架构模式下，子应用打包的方式，基本分为两种：

方案	特点
构建时	子应用通过 Package Registry (可以是 npm package，也可以是 git tags 其他方式) 的方式，与主应用一起打包发布。
运行时	子应用自己构建打包，主应用运行时动态加载子应用资源。

两者的优缺点也很明显：

方案	优点	缺点
构建时	主应用、子应用之间可以做打包优化，如依赖共享等	子应用与主应用之间产品工具链耦合。工具链也是技术栈的一部分。 子应用每次发布依赖主应用重新打包发布
运行时	主应用与子应用完全技术栈无关	

很显然，要实现真正的技术栈无关跟独立部署两个核心目标，大部分场景下我们需要使用运行时加载子应用这种方案。

JS Entry vs HTML Entry

在确定了运行时载入的方案后，另一个需要决策的点是，我们需要子应用提供什么形式的资源作为渲染入口？

`JS Entry` 的方式通常是子应用将资源打成一个 `entry script`，比如 `single-spa` 的 `example` 中的方式。但这个方案的限制也颇多，如要求子应用的所有资源打包到一个 `js bundle` 里，包括 `css`、图片等资源。除了打出来的包可能体积庞大之外的问题之外，资源的并行加载等特性也无法利用上。

`HTML Entry` 则更加灵活，直接将子应用打出来 `HTML` 作为入口，主框架可以通过 `fetch html` 的方式获取子应用的静态资源，同时将 `HTML document` 作为子节点塞到主框架的容器中。这样不仅可以极大的减少主应用的接入成本，子应用的开发方式及打包方式基本上也不需要调整，而且可以天然地解决子应用之间样式隔离的问题(后面提到)。想象一下这样一个场景：

```
<body>
  <main id="root"></main>
</body>
// 子应用入口
ReactDOM.render(<App/>, document.getElementById('root'))
```

如果是 JS Entry 方案，主框架需要在子应用加载之前构建好相应的容器节点(比如这里的 "#root" 节点)，不然子应用加载时会因为找不到 container 报错。但问题在于，主应用并不能保证子应用使用的容器节点为某一特定标记元素。而 HTML Entry 的方案则天然能解决这一问题，保留子应用完整的环境上下文，从而确保子应用有良好的开发体验。

HTML Entry 方案下，主框架注册子应用的方式则变成：

```
framework.registerApp('subApp1', { entry: '//abc.alipay.com/index.html'})
```

本质上这里 HTML 充当的是应用静态资源表的角色，在某些场景下，我们也可以将 HTML Entry 的方案优化成 Config Entry，从而减少一次请求，如：

```
framework.registerApp('subApp1', { html: '', scripts: ['//abc.alipay.com/index.
```

总结一下：

App Entry	优点	缺点
HTML Entry	1. 子应用开发、发布完全独立 2. 子应用具备与独立应用开发时一致的开发体验	1. 多一次请求，子应用资源解析消耗转移到运行时 2. 主子应用不处于同一个构建环境，无法利用 bundler 的一些构建期的优化能力，如公共依赖抽取等
JS Entry	主子应用使用同一个 bundler，可以方便做构建时优化	1. 子应用的发布需要主应用重新打包 2. 主应用需为每个子应用预留一个容器节点，且该节点 id 需与子应用的容器 id 保持一致 3. 子应用各类资源需要一起打成一个 bundle，资源加载效率变低

模块导入

微前端架构下，我们需要(参考 single-spa)，部署、独立部署两种模式同时支持的需求，使得我们只能选择 umd 这种兼容性的模块格式打包我们的子应用。如何在浏览器运行时获取远程脚本中导出的模块引用也是一个需要解决的问题。

通常我们第一反应的解法，也是最简单的解法就是与子应用与主框架之间约定好一个全局变量，把导出的钩子引用挂载到这个全局变量上，然后主应用从这里面取生命周期函数。

这个方案很好用，但是最大的问题是，主应用与子应用之间存在一种强约定的打包协议。那我们是否能找出一种松耦合的解决方案呢？

很简单，我们只需要走 umd 包格式中的 global export 方式获取子应用的导出即可，大体的思路是通过给 window 变量打标记，记住每次最后添加的全局变量，这个变量一般就是应用 export 后挂载到 global 上的变量。实现方式可以参考 [systemjs global import](#)，这里不再赘述。

应用隔离

微前端架构方案中有两个非常关键的问题，有没有解决这两个问题将直接标志你的方案是否真的生产可用。比较遗憾的是此前社区在这个问题上的处理都会不约而同选择“绕道”的方式，比如通过主子应用之间的一些默认约定去规避冲突。而今天我们会尝试从纯技术角度，更智能的解决应用之间可能冲突的问题。

知乎

首发于
kuitos 的后脑勺

由于微前端场景下，不同技术栈的子应用会被集成到同一个运行时中，所以我們必須在框架層確保各個子應用之間不會出現樣式互相干擾的問題。

Shadow DOM?



針對 "Isolated Styles" 這個問題，如果不考慮瀏覽器兼容性，通常第一個浮現到我們腦海里的方案會是 Web Components。基於 Web Components 的 Shadow DOM 能力，我們可以將每個子應用包裹到一個 Shadow DOM 中，保證其運行時的樣式的絕對隔離。

但 Shadow DOM 方案在工程實踐中會碰到一個常見問題，比如我們這樣去構建了一個在 Shadow DOM 里渲染的子應用：

```
const shadow = document.querySelector('#hostElement').attachShadow({mode: 'open'})
shadow.innerHTML = '<sub-app>Here is some new text</sub-app><link rel="stylesheet" href="subapp.css">
```

由於子應用的樣式作用域僅在 shadow 元素下，那麼一旦子應用中出現運行時越界跑到外面構建 DOM 的場景，必定會導致構建出來的 DOM 無法應用子應用的樣式的情況。

比如 sub-app 里調用了 antd modal 組件，由於 modal 是動態掛載到 document.body 的，而由於 Shadow DOM 的特性 antd 的樣式只會在 shadow 這個作用域下生效，結果就是彈出框無法應用到 antd 的樣式。解決的辦法是把 antd 樣式上浮一層，丟到主文檔里，但這麼做意味著子應用的樣式直接泄露到主文檔了。gg...

CSS Module? BEM?

社區通常的實踐是通過約定 css 前綴的方式來避免樣式衝突，即各個子應用使用特定的前綴來命名 class，或者直接基於 css module 方案寫樣式。對於一個全新的項目，這樣當然是可行，但是通常微前端架構更多的目標是解決存量/遺產應用的接入問題。很顯然遺產應用通常是很难有動力做大幅改造的。

最主要的是，約定的方式有一個無法解決的問題，假如子應用中使用了三方的組件庫，三方庫在寫入了大量的全局樣式的同時又不支持定制化前綴？比如 a 應用引入了 antd 2.x，而 b 應用引入了 antd 3.x，兩個版本的 antd 都寫入了全局的 .menu class，但又彼此不兼容怎麼辦？

Dynamic Stylesheet !

解決方案其實很簡單，我們可以在子應用的 HTML 中動態插入樣式表，這樣每個子應用都會對所有的樣式表的插入，即能保證，在一個時間點里，只有一個應用的樣式表是生效的。

上文提到的 HTML Entry 方案則天生具備樣式隔離的特性，因為應用卸載後會直接移除去 HTML 結構，從而自動移除了其樣式表。

比如 HTML Entry 模式下，子應用加載完成的後 DOM 結構可能長這樣：

```
<html>

  <body>
    <main id="subApp">
      // 子應用完整的 html 結構
      <link rel="stylesheet" href="//alipay.com/subapp.css">
      <div id="root">....</div>
    </main>
  </body>

</html>
```

當子應用被替換或卸載時，subApp 節點的 innerHTML 也會被復寫，//alipay.com/subapp.css 也就自然被移除樣式也隨之卸載了。

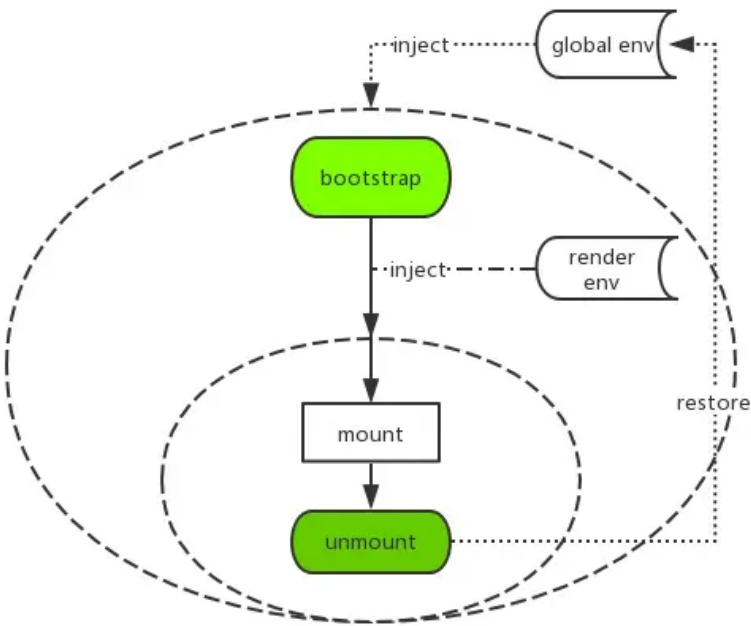
知乎

首发于
kuitos 的后脑勺

解决了样式隔离的问题后，有一个更关键的问题我们还没有解决：如何确保各个子应用之间的全局变量不会互相干扰，从而保证每个子应用之间的软隔离？

这个问题比样式隔离的问题棘手，社区的普遍玩法是给一些全局副作用加各种前缀，避免冲突。但其实我们都明白，这种通过团队间的“口头”约定的方式往往低效且易碎，所有依赖人为约束的方案都很难避免由于人的疏忽导致的线上 bug。那么我们是否有可能打造一个好用的且完全无约束的 JS 隔离方案呢？

针对 JS 隔离的问题，我们独创了一个运行时的 JS 沙箱。简单画了个架构图：



知乎 @kuitos

即在应用的 bootstrap 出/卸载时，将状态回滚
当应用二次进入时则再恢复至 mount 前的状态的，从而确保应用在 remount 时拥有跟第一次 mount 时一致的全局上下文。

当然沙箱里做的事情还不止这些，其他的还包括一些对全局事件监听的劫持等，以确保应用在切出之后，对全局事件的监听能得到完整的卸载，同时也会在 remount 时重新监听这些全局事件，从而模拟出与应用独立运行时一致的沙箱环境。

蚂蚁的微前端落地实践

自去年年底伊始，我们便尝试基于微前端架构模式，构建出一套全链路的面向中后台场景的产品接入平台，目的是解决不同产品之间集成困难、流程割裂的问题，希望接入平台后的应用，不论使用哪种技术栈，在运行时都可以通过自定义配置，实现不同应用之间页面级别的自由组合，从而生成一个千人千面的个性化控制台。

目前这套平台已在蚂蚁生产环境运行半年多，同时接入了多个产品线的 40+ 应用、4+ 不同类型的技术栈。过程中针对大量微前端实践中的问题，我们总结出了一套完整的解决方案：



在内部得到充分的技术验证和线上考验之后，我们决定将这套解决方案开源出来！

qiankun - 一套完整的微前端解决方案

github.com/umijs/qiankun...

取名 qiankun，意为统一。我们希望通过 qiankun 这种技术手段，让你能很方便的将一个巨石应用改造成一个基于微前端架构的系统，并且不再需要去关注各种过程中的技术细节，做到真正的开箱即用和生产可用。

对于 umi 用户我们也提供了配套的 qiankun 插件 [@umijs/plugin-qiankun](#)，以便于 umi 应用能几乎零成本的接入 qiankun。

最后欢迎大家点赞使用提出宝贵的意见。

Probably the most complete micro-frontends solution you ever met .
可能是你见过的最完善的微前端架构解决方案。

招聘

最后打个招聘广告，[蚂蚁金服体验技术部招聘前端啦！](#) 要求 P6 及以上！有兴趣的同学可以发简历到


youzhi.lk@antfin.com

youzhi.lk@antfin.com

youzhi.lk@antfin.com

编辑于 2020-03-28 12:00

[前端架构](#) [前端框架](#)

发布一条带图评论吧

134 条评论

默认 最新

- TIKUN

2019-08-16

回复 33
- 

小问

前 七牛前端架构, 当时我们就想过要做一个这样的东西, 但碍于业务和 js 沙盒尚... 就没实施😓

2019-08-16

回复 13
- 

阿斯蓝

其实像antd这样的UI library, 都提供了getContainer这样的方法可以指定像modal这种组件的挂载位置。

像styled-components、typestyle这样的css-in-js library, 也提供了方法可以指定动态生成的style挂载的位置。

考虑到这两点, web components (shadow dom) 这种解决方案是不是就完全可行而且也能算低代价了呢?

比如像这个框架: [GitHub - micro-zoe/micro-app: A minimalist solution for building micro front-end applications.](#) 一种用于构建微前端应用的极简方案

2021-08-27

回复 5
- 

茶山小旋风

这周把10+业务系统接入了乾坤 看了两天框架源码, 蚂蚁的开源repo果然质量很高 变量命名和备注都巨清晰 🤔

2019-08-16

回复 3
- 

前端逗比

老的系统应用通信可以分享一下, 可以搞篇文章分享一下

2020-03-11

回复 喜欢
- 

茶山小旋风

是

2019-08-16

回复 喜欢
- 展开其他 2 条回复 >
- 

YJiz

这个样式隔离方案的前提是每个子应用独享一个页面? 如果多个子应用同时存在于一个页面, 样式冲突问

2019-08-16

...
- 

辉卫无敌

而且主应用和子应用的样式冲突还是存在吧

2021-01-15

回复 1
- 

失礼

D2 演讲里提到的阿里云的例子就是这样的, 不知道是不是让所有项目都统一了版本或者阿里云本身样式库的 class 就是 hash 化的或者支持用 id 来做了 scope 或者直接用做了代理, 这种场景全部由根应用去代理。当然我想的这些可能都可以用来解决这个问题。

2020-03-13

回复 喜欢
- 展开其他 1 条回复 >
- 

ChasLui

乾坤里用的proxy, ie兼容和iframe嵌套什么时候能解决呀

2019-08-16

回复 6
- 

kuitos

作者

issue 去 github 提吧..

2019-08-16

回复 3
- 

zry754331875

...


知乎

首发于
kuitos 的后脑勺

掘金 <https://mduffy.com/jd/article...>

2019-08-21


回复 3

山野都有雾灯

样式冲突中，如果主应用和子应用存在样式冲突如何解决呢？

2021-01-19


回复 1

huang鱼

自己解决，样式加前缀之类的，我们是给主应用的样式都加了前缀，因为主应用内容相对比较少，加起来很方便

2021-02-20

回复 1

Torn

我想问一下，如果是基于运行时构建，为什么js-entry的方式，主子应用在同一bundler呢？

2020-11-23


回复 1

失业青年

同问

2020-11-28


回复 喜欢

赤晨

小白请教下，qiankun怎么处理子应用之间的逻辑复用呢，例如有store要被两个子应用去使用。是通过父应用来传递？还是把这部分抽离成单独的包让子应用项目分部import？另外公共部分最终是通过window来挂载吗？

2020-08-16

回复 1

礼行-leason

你这里的逻辑复用是指状态共享吧，根本在应用间通信，官方又提供解决方案

2020-09-10

回复 喜欢

点击查看全部评论 >



发布一条带图评论吧



kuitos 的后脑勺
记录一些编程过程



前端外刊评论
关注前端技术，探寻深邃思想，<https://qianduan.group>

推荐阅读

可能是你见过最完善的微前端解决方案

Techniques, strategies and recipes for building a modern web app with multiple teams using different JavaScript frameworks.
—— Micro Frontends前言TL;DR 想跳过技术细节直接看怎...
阿里云云栖... 发表于程序员进修...

聊聊微前端的原理和实践

本文首发于 vivo互联网技术 微信公众号 链接：
<https://mp.weixin.qq.com/s/2qH9q>
作者：Tan Xin本文对微前端的概念和场景进行科普，介绍一些主流的微前端的实现库及其用法...
2020L... 发表于互联网技术...

如何接地气地接入微前端？

一、前言微前端，这个概念已经在国内不止一次的登上各大热门话题，它所解决的问题也很明显，这几个微前端所提到的痛点在我们团队所维护的项目中也是非常凸显。但我始终认为，一个新的技术...
阿里云云栖号



如何接地气地接入微前端？
阿里开发者