一文让你了解微前端的现状

* Florian Rappl
* 盖磊
* 万佳

发布于：2020 年 8 月 4 日 14:03

在前端 Web 开发中，微前端（microfrontends）是一个备受争议的话题。它是否值得开发者采用？面对如此之多的[神奇案例](https://blog.bitsrc.io/11-popular-misconceptions-about-micro-frontends-d5daecc92efb" \t "_blank)，人们无法否认微前端正日益流行这个事实。本文将探究微前端的具体使用场景和使用群体 ，并给出能快速轻松上手的现有解决方案。

什么是微前端？

微前端将大规模的后端系统切分为很多面向前端的微服务，力图实现一定程度的改进。

这里的主要问题是， 各个部分总是作为一个整体被使用和体验的。

用户体验（UX）是由前端直接负责的，因为后端系统从来不会被直接整体访问。

该问题存在多种解决方案。最简单的做法是将现有 API 的数据交换模型替换为 HTML 输出。只需一个超链接即可实现服务（视图）间的跳转。尽管这种解决方案是有效的，但缺点是在很多情况下并不能提供用户所需的 UX。



分布式 Web 应用的演进

显然，要将那些各自独立的较小 UI 部分聚合为一个整体的前端，需要的是一些更为复杂的解决方案。这应视为分布式 Web 应用演进的下一步。

一个重要的问题是，如何理解微前端与组件和模块的关系。事实上，组件、模块、微前端等概念，都是以构建单元的形式实现可重用性和所承担的功能。它们之间的唯一差别是所处的层级不同，具体而言：

* 组件是底层 UI 库的构建单元；
* 模块是相应运行时的构建单元；
* 软件包是依赖解析器的构建单元；
* 微前端是当前应用的构建单元。

如上，微前端可视为组成人体的各个器官，软件包则对应于组成各器官的细胞，而模块就是分子，组件对应于原子。

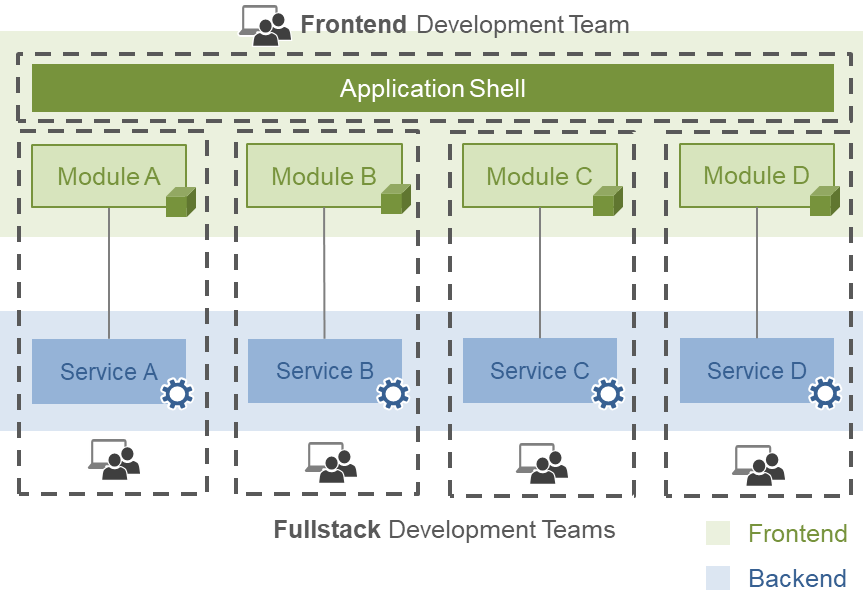
为什么要使用微前端？

使用微前端的原因多种多样， 常见的原因多是技术性的 ，但往往有现实的商业用例（或者提升 UX 的用例）在背后提供支持。

究其根本，微前端解决方案可提供如下特性：

* 单个前端部分可独立开发、测试和部署；
* 无需重新构建即可添加、移除或替换单个前端部分；
* 不同的前端部分可使用不同的技术构建。

由此可见，微前端主要实现了解耦。在应用到达一定规模后，微前端就有了用武之地。其中一个潜在优点是， 它支持组织分割为更多团队，乃至创建更小的全栈团队。



微前端对全栈团队的支持

微前端在如下场景中将更发挥更大作用：

* 多个团队贡献同一个前端
* 一些独立的部分需由特定的用户或团队激活、 终止激活或 roll out
* 需支持外部开发人员对 UI 进行扩展
* UI 的特性集每日 / 每周都在增长，并不会影响系统其它部分
* 不论应用如何增长， 都需要维持恒定的开发速度
* 支持不同团队使用不同的开发工具

微前端的使用者

目前，微前端得到越来越多企业的积极采纳，下面给出部分最新列表：

* DAZN
* Elsevier
* entando
* Fiverr
* Hello Fresh
* IKEA
* Bit.dev
* Microsoft
* Open Table
* OpenMRS
* Otto
* SAP
* Sixt
* Skyscanner
* smapiot
* Spotify
* Starbucks
* Thalia
* Zalando
* ZEISS

…… 不胜枚举！

各个企业所采用的方法当然各有千秋，但是大家的目标是一致的 。



使用微前端技术的企业（由 Luca Mezzalira 提供）

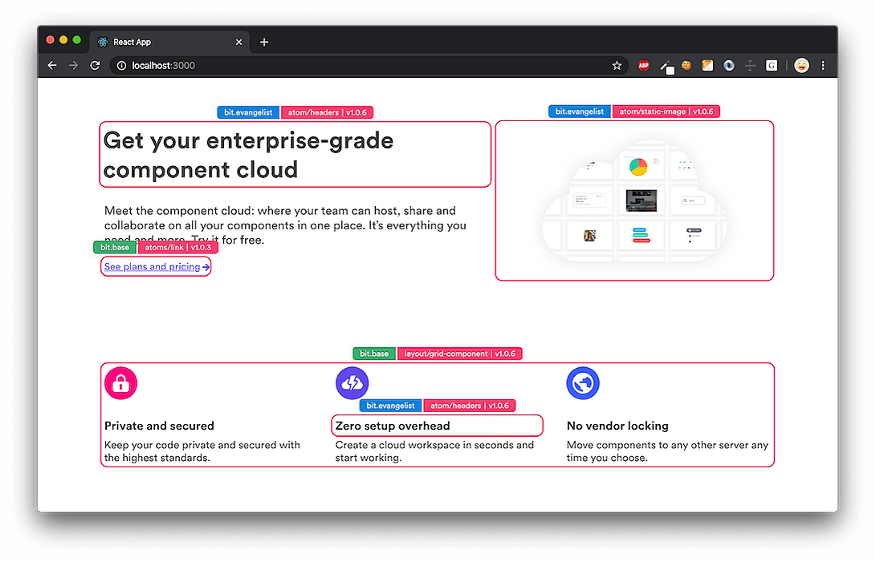
企业列表正不断增长，从 ThoughtWorks、HLC 等咨询公司，到 SalesPad、Apptio 等 SaaS 服务提供商。还有更多传统企业已经押注微前端，典型实例就是德国的隐形巨头 Hoffmann 集团。

Hoffmann 集团很好地展示了微前端并不需要多么大型的团队，也不需要占用多少内部资源。该集团与多家服务提供商有业务往来，这是其选择微前端的一个重要因素。

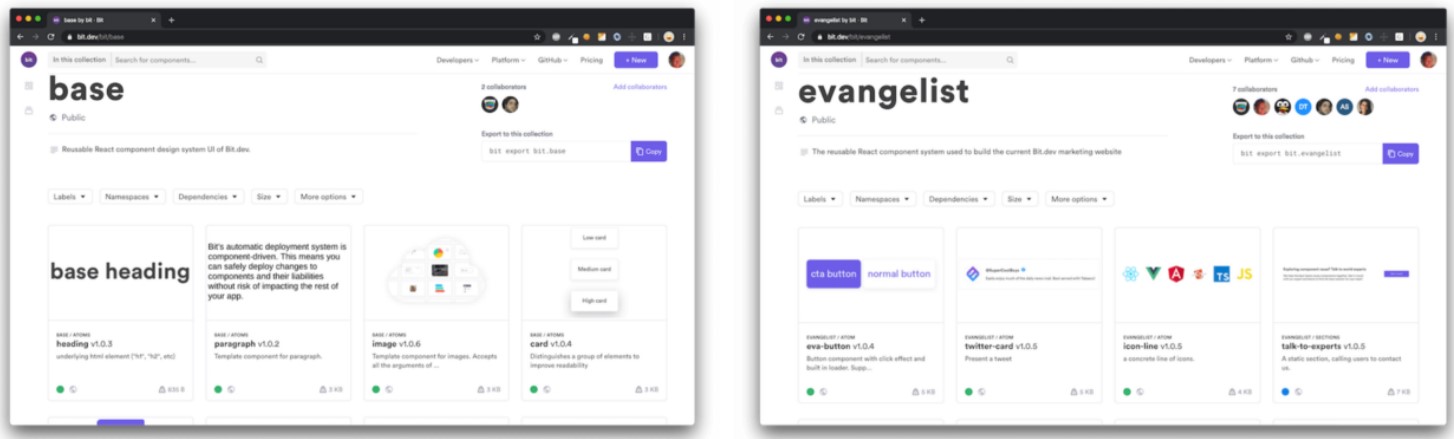
实例：微前端及所使用的组件

[Bit.dev](https://bit.dev/)平台及其市场营销网站均使用 React 组件构建，并且由 Bit 自身维护。

用户在浏览[网站](https://bit.dev/" \t "_blank)查看其“原生服务”时，可停留在不同组件上。点击位于组件上方的名字，即可查看组件详情，进而安装到用户项目中。



构建该页面的组件，基于GitHub 上两个不同的代码库，“ [base-ui](https://github.com/teambit/base-ui)”（ [在Bit 上的访问位置](https://bit.dev/bit/base-ui)）和“ [evangelist](https://github.com/teambit/evangelist)”（ [在Bit 上的访问位置](https://bit.dev/bit/evangelist))。



[base-ui 代码库](https://bit.dev/bit/base-ui)使用 Bit 发布，实现设计系统。 [evangelist 代码库](https://bit.dev/bit/evangelist)用于市场营销页面，其中使用了 base-ui 提供的一些组件，以在不同 MF 之间保持统一的观感。

在此， [Bit](https://bit.dev/)不仅用于自动交付特性，而且用来在不同微前端间维护一致的 UI。

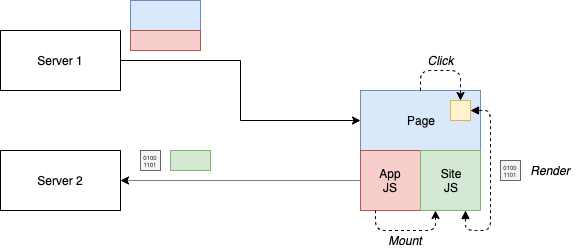
如何构建微前端？

这个 问题没有确切答案。和微服务一样，并不存在适用于所有人的单一方法，也没有已确立的业界标准。

相比微服务实现，微前端不仅在实现细节上存在差异，而且在所有的细枝末节上均有所不同。因此需要区分主要使用场景。一些服务端框架也支持客户端组装，反之依然。

客户端框架

客户端微前端的可选择范围很广。其中部分支持服务端渲染。



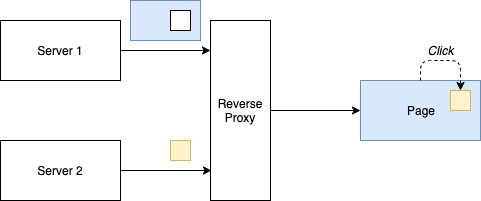
客户端构成

下列框架实现了这种（ 或类似 的）模式：

* [Piral](https://github.com/smapiot/piral)
* [Open Components](https://github.com/opencomponents/oc)
* [qiankun](https://github.com/umijs/qiankun)
* [Luigi](https://github.com/SAP/luigi)
* [Frint.js](https://github.com/frintjs/frint)

服务端框架

服务端框架有多种选项。但其中一些只是用于 express 的软件库或框架；还有一些以服务形式提供，需加载到用户的基础架构中。



服务端构成

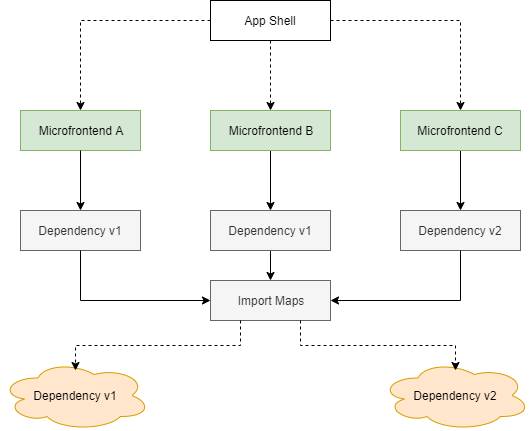
下列框架实现这种（ 或类似 的）模式：

* [Mosaic](https://www.mosaic9.org/)
* [PuzzleJs](https://github.com/puzzle-js/puzzle-js)
* [Podium](https://podium-lib.io/)
* [Micromono](https://github.com/lsm/micromono)

Helper 库

还可考虑一些帮助（helper）库。这些帮助库或是提供共享依赖、路由事件的基础架构，或是将不同的微前端及其生命周期组织起来 。

下例通过 Import Maps 或打包特定 Chunk [实现对共享依赖的处理](https://blog.bitsrc.io/sharing-dependencies-in-micro-frontends-9da142296a2b)。



使用 Import Maps 共享依赖。

下面的库可用于削减模板代码：

* [Module Federation](https://github.com/module-federation)
* [Siteless](https://www.npmjs.com/package/siteless)
* [Single SPA](https://github.com/single-spa)
* [Postal.js](https://github.com/postaljs/postal.js)
* [EventBus](https://github.com/krasimir/EventBus)

微服务的下一步发展

虽然 有些人觉得 Module Federation 之类的帮助库很好用 ，但多数人还是会继续用原来的解决方案 。好的一面是，有很多不受大厂商控制的框架可以用来轻松编写代码 。但至少从技术上看，微前端依然缺少便于解决方案互通的通用标准。

另一个问题是，微前端的社区接受度和采用率仍显不足。

尽管微前端模式已经有一定知名度，但是社区中大多数人仍对其存疑。

究其原因，其一是微服务被视为一种后端设计的最佳实践和标准，但并未当作是一种新的， 可用于特定场景的工具。显然这并不是人们当初想的那样，所以微前端也不应该被视为灵丹妙药。

小结

微前端现有解决方案的可用数量及其在全球许多项目中的用途都发出了强烈的信号：微前端已经随时可以使用！我建议，在实际开始大型 / 生产级项目之前， 先考察各种模式和解决方案。

我想了解大家的观点及原因，对微前端持喜爱、可容忍态度，还是弃若敝屣？