47 | 前端架构：前端架构有哪些核心问题？

winter 2019-05-18

00:00 08:00

讲述：winter 大小：6.42M

你好，我是 winter

今天我们来谈谈架构。

在传统桌面软件开发中，架构师是一种通过设计架构保证团队能够良好分工和有序工作的岗位。

在工程领域，我们凡是要做点什么事儿，都会有明确的目的性，这个目的性，一定是为了完成生产服务业务的。

为什么桌面软件开发需要架构师和架构设计呢？因为桌面软件开发具有高度的复杂性，如果没有架构，就没法分解成互相耦合低的模块来分工。

所以一般来说，架构是为了分工而存在的。但是到了前端领域，这个问题是否还存在呢？答案是，不存在。

前端是个天然按照页面解耦的技术，在多页面架构中，页面的复杂度大约刚好适合一个人的工作量。（所以，我们的结论是，前端根本不需要架构设计。当然，我这句话是开玩笑的。）

前端不存在分工问题，但是在多人协同时，仍然要解决质量和效率的问题，这就需要组件化了。除此之外还有前端特有的兼容性问题，也是需要从架构的角度去解决的。

对于一些追求极致的团队来说，会挑战“单页面应用”，通过单页面应用来提升用户体验，单页面应用的升级版本是谷歌提出的 PWA，PWA 既是业务方案也是技术方案，在技术层面，它近乎苛刻地规定了网页的各方面的体验标准。

前端领域还有一个特有的生态：框架，第一代前端框架（如 jQuery, PrototypeJS）重点解决了兼容问题和 API 的易用性问题，在现代浏览器普及之后，这些问题逐渐变得不存在或者不重要，所以第二代前端框架（如 Vue，Angular，React）重点解决了组件化问题。选择合适的框架，可以节约架构的成本，还能够享受社区资源。

本节课，我会围绕前端架构的几个核心问题，为你介绍前端架构工作。

首先我们来讲讲组件化

组件化

组件化讲起来是个非常简单的概念，前端主要的开发工作是 UI 开发，而把 UI 上的各种元素分解成组件，规定组件的标准，实现组件运行的环境就是组件化了。

现行的组件化方案，目前有五种主流选择：

- Web Component；
- Vue；
- React；
- Angular；
- 自研。

Web Component 是 W3C 推行的规范，理论上是未来的选项；但是实际上这份标准的状态堪忧，Shadow DOM 的设计比较复杂，一般的前端掌握起来都比较困难。

此外，CSS 也比较难以应用，需要依靠 CSS Houdini。目前来说，我还没有看到那个前端团队实际在使用 Web Component 作为组件化方案。当然，它的优势也非常明显：不需要任何额外的运行时支持，就能在现代浏览器环境运行，也可以跟 HTML 无缝结合。

Vue 是目前最受欢迎的框架（从 github star 来看），由华人程序员尤小右开发和维护。它有两个主要特点，一个是比较符合原本的 JS/CSS/HTML 书写习惯；另一个是它绑定了 MVVM 模式，直接确定了 UI 架构，通过 DSL 的支持，数据交互非常简洁。

React 是 Facebook 推行的新一代 Web 框架。它利用 JSX 模式，把 HTML、CSS 和 JS 都放进了 JS 文件中，对于不喜欢 CSS 和 HTML 的前端工程师来说，是很理想的。它还可以迁移到 React Native，直接编写简单的客户端应用。

Angular 是 Google 推出的 Web 框架，它是比较标准的 MVVM 模式。Angular 曾经因为大版本兼容性而饱受诟病，目前它的核心竞争力是与 TypeScript 结合得较好。

上面是我对几种方案的简单介绍。但是实际上，我们做技术选型时的主要依据是团队的现状，开发移动端还是桌面端、是否跟 Native 结合、团队成员的技能分布都是需要考虑的因素，这些框架本身的特点，目前我认为仅仅是一种偏好选项，而不是关键因素。

兼容性和适配性

前端开发的特有问题就是兼容性，到了移动时代，需要面对不同的机型，我们又需要解决适配性问题。

兼容性问题到 2011 年左右都是前端的主旋律，但是在之后，随着现代浏览器的逐渐普及，兼容性问题逐渐减小，所以我们这里就不多谈兼容性问题了。

适配问题主要适配的是屏幕的三个要素：

- 单位英寸像素数（Pixel Per Inch，PPI）：现实世界的一英寸内像素数，决定了屏幕的显示质量
- 设备像素比率（Device Pixel Ratio，DPR）：物理像素与逻辑像素（px）的对应关系
- 分辨率（Resolution）：屏幕区域的宽高所占像素数

在当前环境下，分辨率适配可以使用 vw 单位解决，DPR 适配则需要用到 CSS 的 viewport 规则来控制缩放比例解决，而 PPI 主要影响的是文字，可以采用 media 规则来适配。

单页应用

前文已经讲过，前端架构的解耦问题不大，因为页面是天然解耦的，但是，大家都知道，浏览器加载 HTML 时是会有白屏过程的，对追求极致体验的团队来说，希望能够进一步提升体验，于是就有了“单页应用（SPA）”的概念。

单页应用是把多个页面的内容实现在同一个实际页面内的技术，因为失去了页面的天然解耦，所以就要解决耦合问题。也就是说，我们要在一个“物理页面”内，通过架构设计来实现若干个“逻辑页面”。

逻辑页面应该做到独立开发和独立发布，一种思路是，每个逻辑页面一个 JS，用一个 SPA 框架加载 JS 文件。

从交互的角度，这并不困难，但是，这里还有一个隐性需求，保持前进后退历史。

一般来说，前进后退历史使用 URL 的 Hash 部分来控制，但是 onhashchange 事件并没有提供前进或者后退信息，目前还没有完美的解决方案，只能牺牲一部分体验。实现单页应用的逻辑页面发布需要改造发布系统，在工程上，这也是一个比较大的挑战。

扩展前端新边界

除了解决现实问题，我认为前端架构的职责还包括扩展前端的边界，所以前端架构还包含了很多 Native 开发任务：如客户端和前端结合的方案 Weex 和 React Native；如前端和图形学结合的方案 GCanvas；如前端的 3D 框架 Three.js，这些都是试图用架构的手段赋予前端新的能力的尝试。

这些具体的尝试涉及很多领域知识，我这里就不做详细介绍了，但是如果你成为了一个前端架构师，我希望你也把“拓展前端边界”当做团队的核心目标之一。

总结

今天我从宏观的角度介绍了前端架构相关的知识，我重点介绍了“组件化”“适配性”“单页应用”三个前端架构需要解决的核心问题，组件化在社区有很多现成的方案，我们需要做的主要工作是框架选型。适配性需要用到 CSS 的几种特性：vw 单位、viewport 规则和 media 规则，单页应用重点是逻辑页面解耦、独立开发和发布和保持前进后退历史。

最后留一个思考问题，你所在的团队有前端架构师吗？如果有的话，他的工作职责是什么？