代码的天敌就是代码量

H2 微前端

H3 起源

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 MPA 和 SPA 两者的优势结合起来，构建出一个相对完善的微前端架构方案呢?

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。

工程价值

公司内部使用

优点：

1. 独立开发和部署
2. 大型单页应用无限扩展
3. 不限技术栈

4. 快速整合业务
5. 多团队协作

缺点：

1. 体验有折损
2. 维护成本变高 （版本的升级，公共组件的公用）
3. 管理版本复杂、依赖复杂
4. 开发体验不太友好 （业务域模块开发）
5. 粒度不宜太小

H3 区别

微前端	Widget / 业务组件
架构体系。用来实现大型Web应用	以库(外联/npm)的形式实现复用
生产方式	生产工具
通过隔离机制实现技术栈无关	**需要人工解决依赖和冲突问题**
单独构建 \ 单独发布 \ 热升级	整体构建 \ 整体发布
体系化治理，可控性强	可控性差
主从关系(路由映射、消息机制)	相互无关
微应用是产品的子集(粒度大)	通用功能(粒度小)
变化快	变化小
若干微应用的组合	“外挂”

H3 微前端架构实践中的问题

H4 构建时组合 VS 运行时组合

H4 JS Entry vs HTML Entry

H4 样式隔离

H5 Shadow DOM

基于 Web Components 的 Shadow DOM 能力（内外完全没联系），我们可以将每个子应用包裹到一个 Shadow DOM 中，保证其运行时的样式的绝对隔离。

但 Shadow DOM 方案在工程实践中会碰到一个常见问题，比如我们这样去构建了一个在 Shadow DOM 里渲染的子应用：

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

由于子应用的样式作用域仅在 shadow 元素下，那么一旦子应用中出现运行时越界跑到外面构建 DOM 的场景，必定会导致构建出来的 DOM 无法应用子应用的样式的情况。

比如 sub-app 里调用了 antd modal 组件，由于 modal 是动态挂载到 document.body 的，而由于 Shadow DOM 的特性 antd 的样式只会在 shadow 这个作用域下生效，结果就是弹出框无法应用到 antd 的样式。解决的办法是把 antd 样式上浮一层，丢到主文档里，但这么做意味着子应用的样式直接泄露到主文档了。gg...

H5 CSS Module? BEM?

社区通常的实践是通过约定 css 前缀的方式来避免样式冲突(人肉不推荐)，即各个子应用使用特定的前缀来命名 class，或者直接基于 css module 方案写样式。对于一个全新的项目，这样当然是可行，但是通常微前端架构更多的目标是解决**存量/遗产** 应用的接入问题。**很显然遗产应用通常是很难有动力做大幅改造的**。

最主要的是，约定的方式有一个无法解决的问题，假如子应用中使用了三方的组件库，三方库在写入了大量的**全局样式**的同时又不支持定制化前缀？比如 a 应用引入了 antd 2.x，而 b 应用引入了 antd 3.x，两个版本的 antd 都写入了全局的 `.menu class`，但又彼此不兼容怎么办？

微前端只能做到子应用之间是不会相互干扰的，父应用一般做的很少，就只有左侧菜单个顶部导航栏，很少会有跟子应用之间有样式之间的冲突，如果有的话就把父应用的权重提高就行了。

H5 Dynamic Stylesheet !

解决方案其实很简单，我们只需要在应用切出/卸载后，同时卸载掉其样式表即可，原理是浏览器会对所有的样式表的插入、移除做整个 CSSOM 的重构，从而达到 插入、卸载 样式的目的。这样即能保证，在一个时间点里，只有一个应用的样式表是生效的。

上文提到的 HTML Entry 方案则天生具备样式隔离的特性，因为应用卸载后会直接移除去 HTML 结构，从而自动移除了其样式表。

比如 HTML Entry 模式下，子应用加载完成的后的 DOM 结构可能长这样：

```
<html>

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

</html>
```

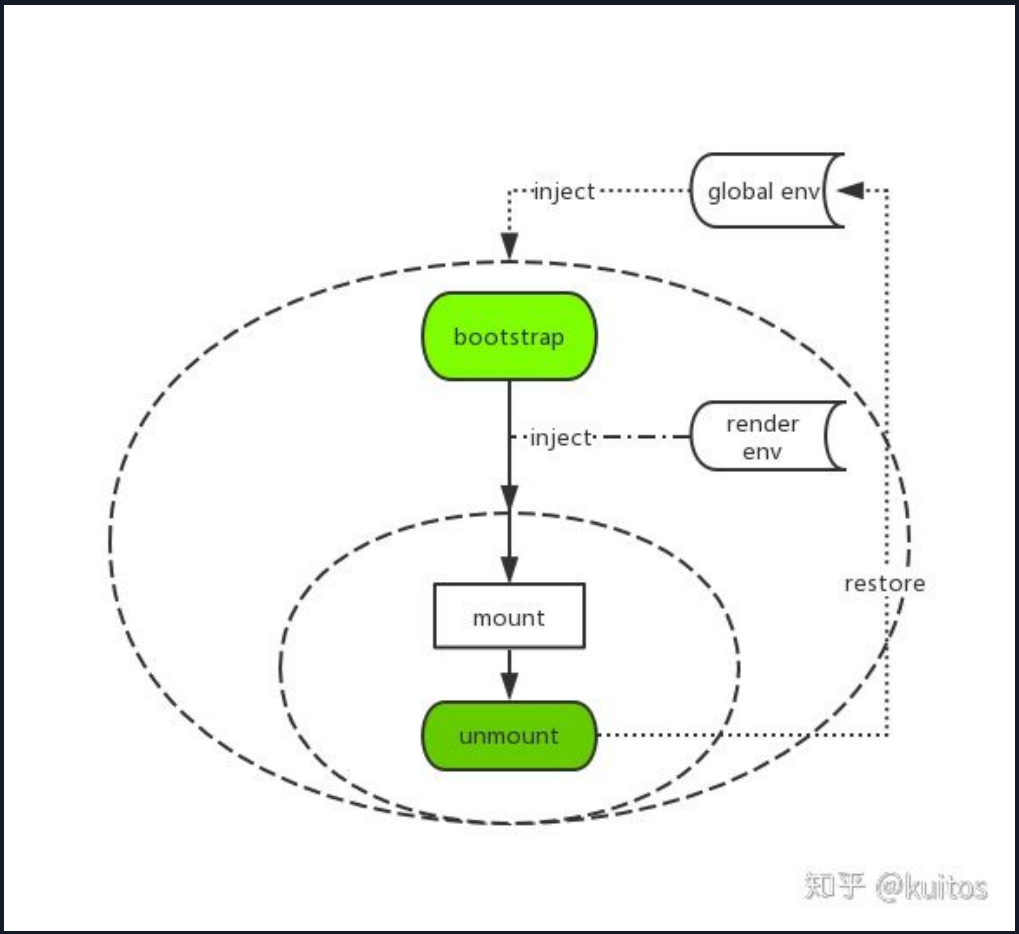
当子应用被替换或卸载时，`subApp` 节点的 innerHTML 也会被复写，`//alipay.com/subapp.css` 也就自然被移除样式也随之卸载了。

H4 JS 隔离

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更棘手，社区的普遍玩法是给一些全局副作用加各种前缀从而避免冲突。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出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：



即在应用的 bootstrap 及 mount 两个生命周期开始之前分别给全局状态打下快照，然后当应用切出/卸载时，将状态回滚至 bootstrap 开始之前的阶段，确保应用对全局状态的污染全部清零。而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远不止这些，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。

H3 参考文档

蚂蚁 有知(乾坤) 沙盒内容

基于 qiankun 的微前端最佳实践（万字长文） - 从 0 到 1 篇

微前端架构模板

微服务的JavaScript框架 single-spa

乾坤文档

一些关于微前端的文章

d2峰会_自己找微前端视频有三篇