



微前端就是将不同的功能按照不同的维度拆分成多个子应用。通过主应用来加载这些子应用。

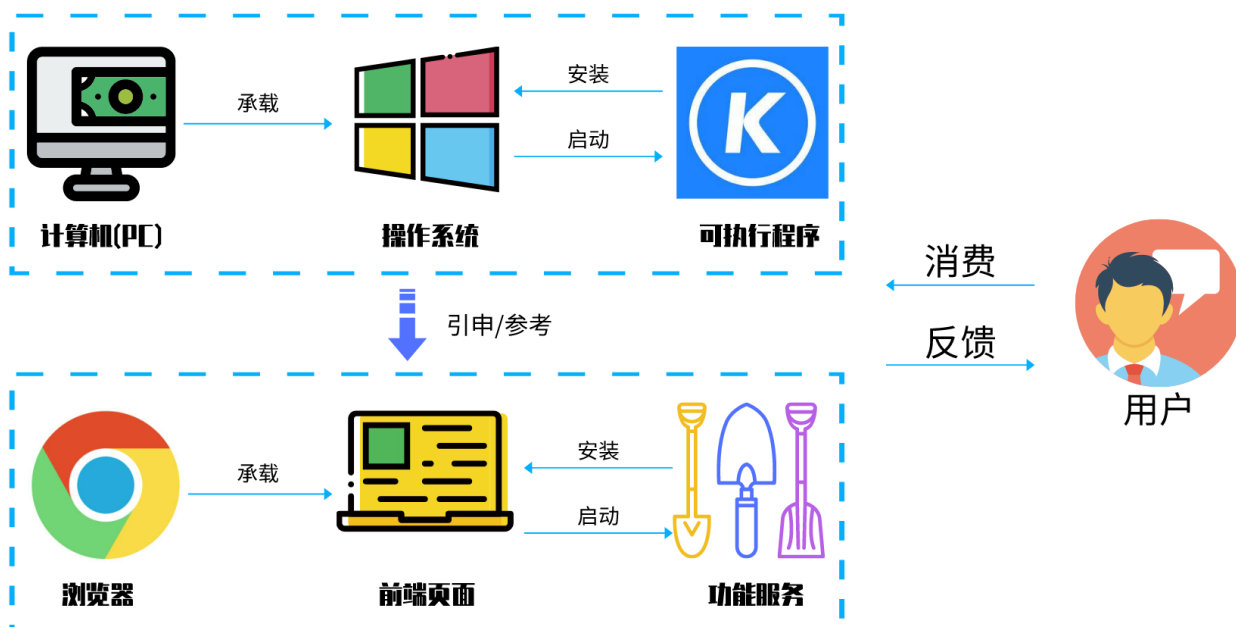
微前端的核心在于**拆**, 拆完后在**合**!

Why?为什么去使用他?

- 不同团队间开发同一个应用技术栈不同怎么破?
- 希望每个团队都可以独立开发, 独立部署怎么破?
- 项目中还需要老的应用代码怎么破?

我们是不是可以将一个应用划分成若干个子应用, 将子应用打包成一个个的lib。当路径切换时加载不同的子应用。这样每个子应用都是独立的, 技术栈也不用做限制了! 从而解决了前端协同开发问题

How?怎样落地微前端?



2018年 Single-SPA诞生了， `single-spa` 是一个用于前端微服务化的 JavaScript 前端解决方案 (本身没有处理样式隔离， `js` 执行隔离) 实现了路由劫持和应用加载

2019年 `qiankun` 基于Single-SPA, 提供了更加开箱即用的 API (`single-spa` + `sandbox` + `import-html-entry`) 做到了，技术栈无关、并且接入简单 (像 `i frame` 一样简单)

总结：子应用可以独立构建，运行时动态加载,主子应用完全解耦，技术栈无关，靠的是协议接入 (子应用必须导出 `bootstrap`、`mount`、`unmount`方法)

这里先回答大家肯定会问的问题：

这不是 `iframe` 吗？

- 如果使用 `iframe` ， `iframe` 中的子应用切换路由时用户刷新页面就尴尬了。

应用通信:

- 基于URL来进行数据传递，但是传递消息能力弱
- 基于 `CustomEvent` 实现通信
- 基于props主子应用间通信
- 使用全局变量、 `Redux` 进行通信

公共依赖:

- `CDN` - `externals`
- `webpack` 联邦模块

二. SingleSpa 实战

1). 构建子应用

```
1 vue create spa-vue
2 npm install single-spa-vue
```

sh

```
1 import singleSpaVue from 'single-spa-vue';
2 const appOptions = {
3   el: '#vue',
4   router,
5   render: h => h(App)
6 }
7 // 在非子应用中正常挂载应用
8 if(!window.singleSpaNavigate){
9   delete appOptions.el;
10  new Vue(appOptions).$mount('#app');
11 }
12 const vueLifeCycle = singleSpaVue({
13   Vue,
14   appOptions
15 });
16 // 子应用必须导出 以下生命周期 bootstrap、mount、unmount
17 export const bootstrap = vueLifeCycle.bootstrap;
18 export const mount = vueLifeCycle.mount;
19 export const unmount = vueLifeCycle.unmount;
20 export default vueLifeCycle;
```

js

```
1 const router = new VueRouter({
2   mode: 'history',
3   base: '/vue',
4   routes
5 })
```

js

配置子路由基础路径

2). 配置库打包

js

```

1  module.exports = {
2      configureWebpack: {
3          output: {
4              library: 'singleVue',
5              libraryTarget: 'umd'
6          },
7          devServer:{
8              port:10000
9          }
10     }
11 }

```

将子模块打包成类库

3).主应用搭建

html

```

1  <div id="nav">
2      <router-link to="/vue">vue项目</router-link>
3      <div id="vue"></div>
4  </div>

```

将子应用挂载到 id="vue" 标签中

js

```

1  import Vue from 'vue'
2  import App from './App.vue'
3  import router from './router'
4  import ElementUI from 'element-ui';
5  import 'element-ui/lib/theme-chalk/index.css';
6  Vue.use(ElementUI);
7  const loadScript = async (url)=> {
8      await new Promise((resolve,reject)=>{
9          const script = document.createElement('script');
10         script.src = url;
11         script.onload = resolve;
12         script.onerror = reject;
13         document.head.appendChild(script)
14     });
15 }
16 import { registerApplication, start } from 'single-spa';
17 registerApplication(
18     'singleVue',
19     async ()=>{

```

```

20         await loadScript('http://localhost:10000/js/chunk-vendors.js');
21         await loadScript('http://localhost:10000/js/app.js');
22         return window.singleVue
23     },
24     location => location.pathname.startsWith('/vue')
25 )
26 start();
27 new Vue({
28     router,
29     render: h => h(App)
30 }).$mount('#app')
31

```

4).动态设置子应用 publicPath

```

1     if(window.singleSpaNavigate){
2         __webpack_public_path__ = 'http://localhost:10000/'
3     }

```

js

三. qiankun 实战

1).主应用编写

```

1     <el-menu :router="true" mode="horizontal">
2         <el-menu-item index="/">首页</el-menu-item>
3         <el-menu-item index="/vue">vue应用</el-menu-item>
4         <el-menu-item index="/react">react应用</el-menu-item>
5     </el-menu>
6     <router-view v-show="$route.name"></router-view>
7     <div v-show="!$route.name" id="vue"></div>
8     <div v-show="!$route.name" id="react"></div>

```

html

2).注册子应用

```

1     import {registerMicroApps,start} from 'qiankun'
2     const apps = [
3         {
4             name:'vueApp',
5             entry:'//localhost:10000',

```

js

```

6         container: '#vue',
7         activeRule: '/vue'
8     },
9     {
10        name: 'reactApp',
11        entry: '//localhost:20000',
12        container: '#react',
13        activeRule: '/react'
14    }
15 ]
16 registerMicroApps(apps);
17 start();

```

3).子Vue应用

```

1  let instance = null;
2  function render(){
3      instance = new Vue({
4          router,
5          render: h => h(App)
6      }).$mount('#app')
7  }
8  if(window.__POWERED_BY_QIANKUN__){
9      __webpack_public_path__ = window.__INJECTED_PUBLIC_PATH_BY_QIANKUN__;
10 }
11 if(!window.__POWERED_BY_QIANKUN__){render()}
12 export async function bootstrap(){}
13 export async function mount(props){render();}
14 export async function unmount(){instance.$destroy();}

```

js

打包配置

```

1  module.exports = {
2      devServer:{
3          port:10000,
4          headers:{
5              'Access-Control-Allow-Origin': '*'
6          }
7      },
8      configureWebpack:{
9          output:{
10              library: 'vueApp',
11              libraryTarget: 'umd'

```

js

```

12     }
13   }
14 }

```

4).子React应用

```

1   import React from 'react';
2   import ReactDOM from 'react-dom';
3   import './index.css';
4   import App from './App';
5   function render() {
6     ReactDOM.render(
7       <React.StrictMode>
8         <App />
9       </React.StrictMode>,
10    document.getElementById('root')
11  );
12  }
13  if(!window.__POWERED_BY_QIANKUN__){
14    render()
15  }
16  export async function bootstrap() {}
17  export async function mount() {render();}
18  export async function unmount() {
19    ReactDOM.unmountComponentAtNode(document.getElementById("root"));
20  }
21

```

js

重写 react 中的 webpack 配置文件 (config-overrides.js)

```

1   yarn add react-app-rewired --save-dev

```

```

1   module.exports = {
2     webpack: (config) => {
3       config.output.library = `reactApp`;
4       config.output.libraryTarget = "umd";
5       config.output.publicPath = 'http://localhost:20000/';
6       return config
7     },
8     devServer: function (configFunction) {
9       return function (proxy, allowedHost) {
10        const config = configFunction(proxy, allowedHost);

```

js

```

11         config.headers = {
12             "Access-Control-Allow-Origin": "*",
13         };
14         return config;
15     };
16 },
17 };

```

配置 .env 文件

```

1 PORT=20000
2 WDS_SOCKET_PORT=20000

```

React路由配置

```

1 import { BrowserRouter, Route, Link } from "react-router-dom"
2 const BASE_NAME = window.__POWERED_BY_QIANKUN__ ? "/react" : "";
3 function App() {
4     return (
5         <BrowserRouter basename={BASE_NAME}>
6             <Link to="/">首页</Link>
7             <Link to="/about">关于</Link>
8             <Route path="/" exact render={() => <h1>hello home</h1>}></Route>
9             <Route path="/about" render={() => <h1>hello about</h1>}></Route>
10         </BrowserRouter>
11     );
12 }

```

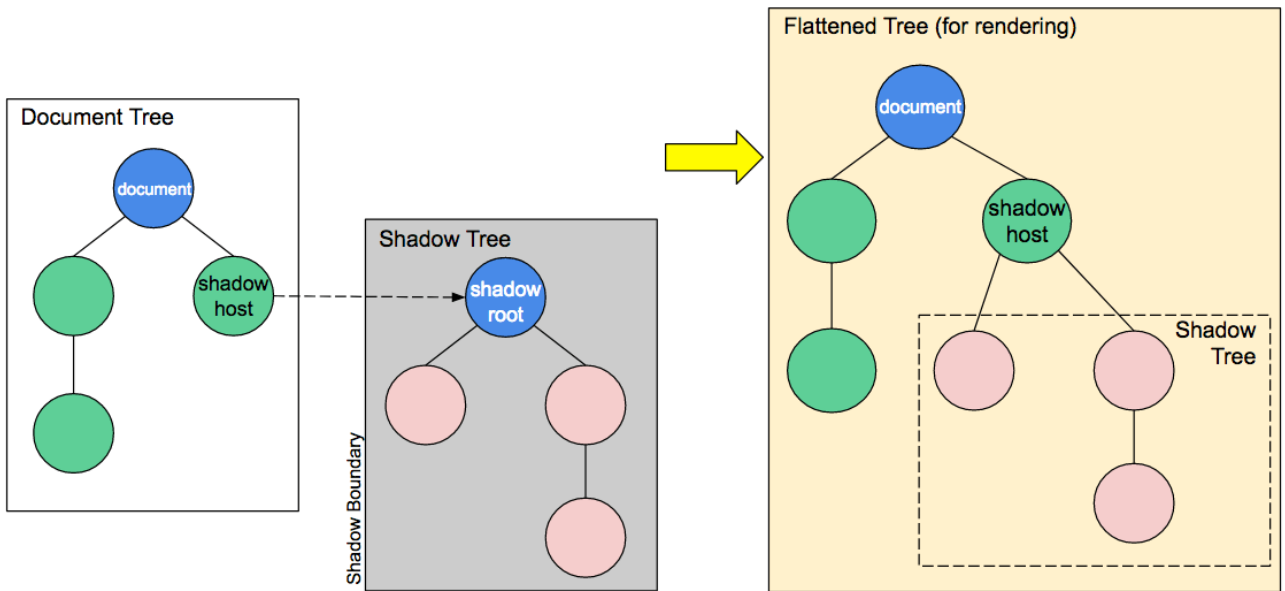
四. CSS 隔离方案

子应用之间样式隔离：

- Dynamic Stylesheet 动态样式表，当应用切换时移除老应用样式，添加新应用样式

主应用和子应用之间的样式隔离：

- BEM (Block Element Modifier) 约定项目前缀
- CSS-Modules 打包时生成不冲突的选择器名
- Shadow DOM 真正意义上的隔离
- css-in-js



```

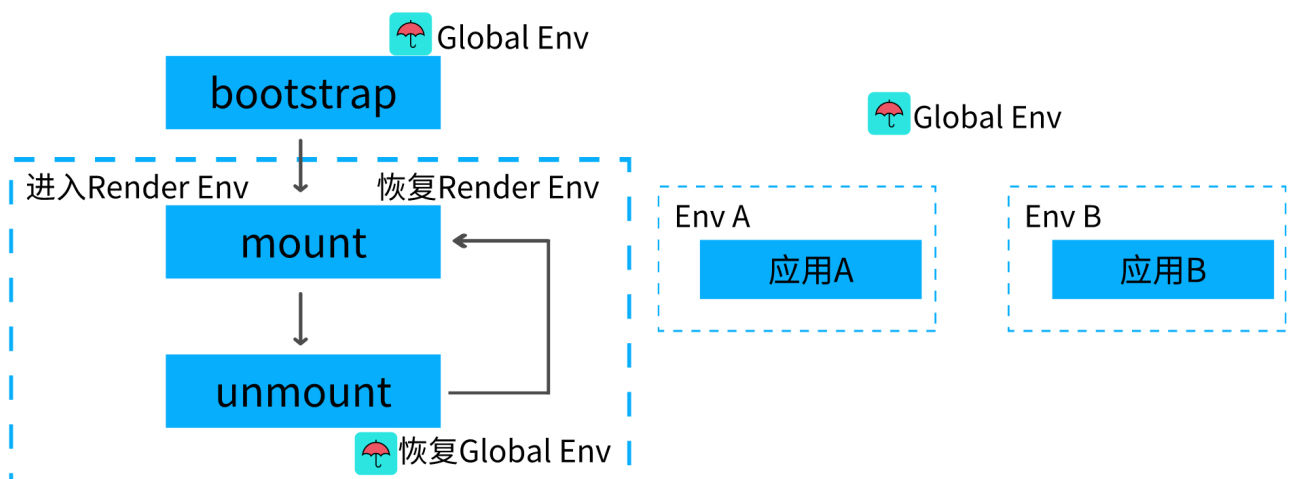
1  let shadowDom = shadow.attachShadow({ mode: 'open' });
2  let pElement = document.createElement('p');
3  pElement.innerHTML = 'hello world';
4  let styleElement = document.createElement('style');
5  styleElement.textContent = `
6    p{color:red}
7  `;
8  shadowDom.appendChild(pElement);
9  shadowDom.appendChild(styleElement)

```

js

shadow DOM 可以实现真正的隔离机制

五. JS 沙箱机制



当运行子应用时应该跑在内部沙箱环境中

- 快照沙箱，在应用沙箱挂载或卸载时记录快照，在切换时依据快照恢复环境 (无法支持多实例)
- Proxy 代理沙箱,不影响全局环境

1).快照沙箱

- 1.激活时将当前window属性进行快照处理
- 2.失活时用快照中的内容和当前window属性比对
- 3.如果属性发生变化保存到 `modifyPropsMap` 中，并用快照还原window属性
- 4.在次激活时，再次进行快照，并用上次修改的结果还原window

```
1      class SnapshotSandbox {
2          constructor() {
3              this.proxy = window;
4              this.modifyPropsMap = {}; // 修改了那些属性
5              this.active();
6          }
7          active() {
8              this.windowSnapshot = {}; // window对象的快照
9              for (const prop in window) {
10                 if (window.hasOwnProperty(prop)) {
11                     // 将window上的属性进行拍照
12                     this.windowSnapshot[prop] = window[prop];
13                 }
14             }
15             Object.keys(this.modifyPropsMap).forEach(p => {
16                 window[p] = this.modifyPropsMap[p];
17             });
18         }
19         inactive() {
20             for (const prop in window) { // diff 差异
21                 if (window.hasOwnProperty(prop)) {
22                     // 将上次拍照的结果和本次window属性做对比
23                     if (window[prop] !== this.windowSnapshot[prop]) {
24                         // 保存修改后的结果
25                         this.modifyPropsMap[prop] = window[prop];
26                         // 还原window
27                         window[prop] = this.windowSnapshot[prop];
28                     }
29                 }
30             }
31         }
32     }
```

js

```

31     }
32 }

1 let sandbox = new SnapshotSandbox();
2 ((window) => {
3     window.a = 1;
4     window.b = 2;
5     window.c = 3
6     console.log(a,b,c)
7     sandbox.inactive();
8     console.log(a,b,c)
9 })(sandbox.proxy);

```

js

快照沙箱只能针对单实例应用场景,如果是多个实例同时挂载的情况则无法解决,只能通过 proxy 代理沙箱来实现

2).Proxy 代理沙箱

```

1 class ProxySandbox {
2     constructor() {
3         const rawWindow = window;
4         const fakeWindow = {}
5         const proxy = new Proxy(fakeWindow, {
6             set(target, p, value) {
7                 target[p] = value;
8                 return true
9             },
10            get(target, p) {
11                return target[p] || rawWindow[p];
12            }
13        });
14        this.proxy = proxy
15    }
16 }
17 let sandbox1 = new ProxySandbox();
18 let sandbox2 = new ProxySandbox();
19 window.a = 1;
20 ((window) => {
21     window.a = 'hello';
22     console.log(window.a)
23 })(sandbox1.proxy);
24 ((window) => {
25     window.a = 'world';

```

js

```
26     console.log(window.a)
27   })(sandbox2.proxy);
```

每个应用都创建一个proxy来代理window，好处是每个应用都是相对独立，不需要直接更改全局window属性！

← 从零手写Vue3原理

从零实现微前端框架 →