一.为什么需要微前端?

What?什么是微前端?

> 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?

1. 不同团队间开发同一个应用技术栈不同怎么破?
2. 希望每个团队都可以独立开发，独立部署怎么破?
3. 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 基于Single-SPA, 提供了更加开箱即用的 API (+ sandbox +) 做到了，技术栈无关、并且接入简单(像i 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

SingleSpa 实战

1.构建子应用(基于 vue)

> 创建项目并安装 single-spa-vue

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

> main.js

```
import Vue from 'vue'
import App from './App.vue'
import router from './router'
import singleSpaVue from 'single-spa-vue'

Vue.config.productionTip = false

const appOptions = {
  el: "#vue" , // 挂载到父应用中的 id 为 vue 的标签中
  router,
  render: h => h(App)
}

// 如果是父应用引用我
if(window.singleSpaNavigate){
    __webpack_public_path__ = 'http://localhost:10001/'; // 动态设置子应用路径前缀
 }
 // 子应用独立开发
if(!window.singleSpaNavigate){
  delete appOptions.el;
  new Vue(appOptions).$mount('#app');
}
// 包装一个vue微前端服务对象
const vueLifecycle = singleSpaVue({
  Vue,
  appOptions
})
// 子应用必须导出一下生命周期，相当于协议接入，子应用定义好协议，父应用会调用这些方法
export const bootstrap = vueLifecycle.bootstrap // 启动时
export const mount = vueLifecycle.mount // 挂载时
export const unmount = vueLifecycle.unmount // 卸载时
export default vueLifecycle
```

> router目录下 index.js

```
const router = new VueRouter({
  mode: 'history',
  // base: process.env.BASE_URL,
  base: '/vue', // 子应用基路径（根父应用registerApplication里面
location 里面配的路径一致），不改的话，子应用会默认相对父应用基路径
  routes
})
```

2.配置库打包

> vue.config.js

```
module.exports = {
    configureWebpack:{
        output:{ // 将子模块打包成类库
            library: 'singleVue', // 打包类库的名字
            libraryTarget:'umd' //
window.singleVue.bootstrap/mount/unmount 打包将 export
bootstrap/mount/unmount方法挂载在 window 上
        },
        devServer:{
            port: 10001
        }
    }
}
```

3.构建主应用（基于 vue ）

> app.vue，将子应用挂载到 id="vue"便签中

```
<template>
  <div id="app">
    <router-link to='/vue'>加载 vue 应用</router-link>
    <div id="vue"></div>
  </div>
</template>
```

> main.js

```
import Vue from 'vue'
import App from './App.vue'
import router from './router'
import { registerApplication, start } from 'single-spa';

Vue.config.productionTip = false

const loadScript = async (url) => {
  return new Promise((resolve,reject) => {
    let script = document.createElement('script');
    script.src = url;
    script.onload = resolve;
    script.onerror = reject;
    document.head.appendChild(script);
  })
}
// singleSpa 缺陷 不够灵活 不能动态加载js 文件
// 样式不隔离 没有js 沙箱机制
registerApplication('myVueApp',
  async () => {
    console.log('加载模块');
    // systemJS
    await loadScript('http://localhost:10001/js/chunk-vendors.js');
    await loadScript('http://localhost:10001/js/app.js');
    return window.singleVue; // 就能拿到 bootstrap mount unmount 方法
  },
  location => location.pathname.startsWith('/vue') // 用户切换到/vue
的路径下，需要加载刚才定义的子应用
);

start(); // 启动

new Vue({
  router,
  render: h => h(App)
}).$mount('#app')
```

快照沙箱

```
// 如果应用加载，刚开始我加载 A 应用 window.a,B应用（window.a）
        // 单应用切换，沙箱创造一个干净的环境给这个子应用使用，当切换时可以选择
丢弃属性和恢复属性
```

```
        // JS 沙箱 proxy

        class SnapshotSandbox {
            constructor (){
                this.proxy = window;
                this.modifyPropsMap = {} // 修改了哪些属性
                this.active();
            }
            active() { // 激活
                this.windowSnapshot = {} // window 对象的快照
                for(const prop in window){
                    if(window.hasOwnProperty(prop)){
                        // 将 window 上的属性进行拍照
                        this.windowSnapshot[prop] = window[prop];
                    }
                }
                Object.keys(this.modifyPropsMap).forEach(p => {
                    window[p] = this.modifyPropsMap[p]; // 修改了的属
性挂载在 window 上
                })
            }
            inactive(){ // 失活
                for(const prop in window){ // diff 差异
                    if(window.hasOwnProperty(prop)){
                        if(window[prop] !==
this.windowSnapshot[prop]){
                            // 保存修改的结果
                            this.modifyPropsMap[prop] =
window[prop];
                            // 还原 window
                            window[prop] =
this.windowSnapshot[prop];
                        }
                    }
                }
            }
        }
        // 应用的运行，从开始到结束，切换后不会影响全局
        let sandBox = new SnapshotSandbox();
        ((window)=>{
            window.a = 1;
            window.b = 2;
            window.c = 3;
            console.log(window.a,window.b,window.c); // 1 2 3
            sandBox.inactive();
            console.log(window.a,window.b,window.c); // undefind
undefind undefind
```

```
            sandBox.active();
            console.log(window.a,window.b,window.c);  // 1 2 3
        })(sandBox.proxy);
        // 如果是多个子应用就不能使用这种方式了，需要使用es6的 proxy
        // 代理沙箱可以实现多应用沙箱，把不同的应用用不同的代理来处理
```

Proxy 代理沙箱

```
class ProxySandbox {
    constructor(){
        const rawWindow = window;
        const fakeWindow = {};
        const proxy = new Proxy(fakeWindow,{
            set(target,p,value){
                target[p] = value;
                return true;
            },
            get(target,p){
                return target[p] || rawWindow[p];
            }
        });
        this.proxy = proxy;
    }
}
let sandbox1 = new ProxySandbox();
let sandbox2 = new ProxySandbox();
window.a = 1;
((window) => {
    window.a = 'hello';
    console.log(window.a); // hello
})(sandbox1.proxy);
((window) => {
    window.a = 'world';
    console.log(window.a); // world
})(sandbox2.proxy);
```

qiankun实战(官网： https://qiankun.umijs.org/zh)

构建主应用（基于 vue）

> 创建项目并安装qiankun

```
vue create qiankun-base
$ yarn add qiankun  # or npm i qiankun -S
```

> main.js

```
import Vue from 'vue'
import App from './App.vue'
import router from './router'
import ElementUI from 'element-ui';
import 'element-ui/lib/theme-chalk/index.css';
Vue.use(ElementUI);
import
{registerMicroApps,start,initGlobalState,MicroAppStateActions} from
'qiankun';

let state = {a:1,b:[{
  name:'king',
  age:18
}]}
// 初始化 state
const actions = initGlobalState(state);

actions.onGlobalStateChange((state,prev) => {
  // state:变更后的状态,  prev:变更前的状态
  console.log('主=>',state,prev)
})

actions.setGlobalState(state);

const apps = [
  {
    name:'vueApp', // 应用的名字
    entry:'//localhost:8888', // 默认会加载这个html 解析里面的js 动态的执
行 (子应用必须支持跨域) fetch
    container:'#vue', // 容器名
    activeRule:'/vue', // 激活的路径
    props:{a:1}

  },
  {
    name:'reactApp',
    entry:'//localhost:20000', // 默认会加载这个html 解析里面的js 动态的
```

```
执行 （子应用必须支持跨域）fetch
    container:'#react',
    activeRule:'/react',
  }
]
registerMicroApps(apps); // 注册应用
start({
  prefetch:false // 取消预加载
});// 开启

new Vue({
  router,
  render: h => h(App)
}).$mount('#app')
```

构建子应用（基于 vue）

> 创建项目

```
vue create qiankun-vue
```

> main.js

```
import Vue from 'vue'
import App from './App.vue'
import router from './router'

Vue.config.productionTip = false


let instance = null;
function render(props){
  instance = new Vue({
    router,
    render: h => h(App)
  }).$mount('#app') // 这里是挂载到自己的 html 中，主应用会拿到这个挂载后的
html，将其插入进去
}
// 动态添加publicpath 路径，主要解决了子应用动态载入的脚本、样式、图片等地址不
正确的问题
if(window.__POWERED_BY_QIANKUN__){
  __webpack_public_path__ =
window.__INJECTED_PUBLIC_PATH_BY_QIANKUN__;
}
// 独立运行子应用，便于开发调试
if(!window.__POWERED_BY_QIANKUN__){
  render();
}

// 子应用定义好协议
export async function bootstrap(props){

}
export async function mount(props){
  console.log(props)
  let state = {a:1,b:[{
    name:'jake',
    age:18
  }]}
  props.onGlobalStateChange((state,prev)=>{
     // state: 变更后的状态; prev 变更前的状态
     console.log('子=>',state, prev);
  })
  props.setGlobalState(state);
  render(props);
}
export async function unmount(props){
  instance.$destroy();
}
```

> router目录下 index.js

```
const router = new VueRouter({
  mode: 'history',
  // base: process.env.BASE_URL,
  base: '/vue', // 子应用基路径（根父应用registerApplication里面
location 里面配的路径一致），不改的话，子应用会默认相对父应用基路径
  routes
})
```

配置库打包

> vue.config.js

```
module.exports = {
    configureWebpack:{
        output:{ // 将子模块打包成类库
            library: 'singleVue', // 打包类库的名字
            libraryTarget:'umd' //
window.singleVue.bootstrap/mount/unmount 打包将 export
bootstrap/mount/unmount方法挂载在 window 上
        },
        devServer:{
            port: 8888,
            headers:{
                'Access-Control-Allow-origin':'*' // 主应用加载子应用资
源是通过 fetch 方式请求，需要允许跨域
            }
        },
    }
}
```

构建子应用（react）

> 创建项目并安装react-app-rewired（react-app-rewired是一个不用 eject 暴露的自定义 webpack配置的包，参考资料：https://juejin.im/post/6844904016581754888）

```
react-create-app qiankun-react
$ yarn add react-app-rewired  # or npm i react-app-rewired -S
```

> package.json

```
  "scripts": {
    "start": "react-app-rewired start",
    "build": "react-app-rewired build",
    "test": "react-app-rewired test",
    "eject": "react-app-rewired eject"
  }
```

> 在项目根目录下新建config-overrides.js 文件，自定义 webpack 配置

```
module.exports = {
    webpack:(config) => {
        config.output.library = 'reactApp';
        config.output.libraryTarget = 'umd';
        config.output.publicPath = 'http://localhost:20000/'; // 主
应用加载子应用打包后的资源路劲前缀，例
如：http://localhost:20000/static/js/bundle.js
        return config;
    },
    devServer:(configFunction) => {
        return function(proxy,allowedHost){
            const config = configFunction(proxy,allowedHost);
            // config.port = 20000;
            config.headers = {
                "Access-Control-Allow-origin":'*' // 主应用加载子应用资
源是通过 fetch 方式请求，需要允许跨域
            }
            return config;
        }
    }
}
```

> 新建.env文件，配置全局的环境变量

```
PORT = 20000
WDS_SOCKET_PORT = 20000
```

> index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import * as serviceWorker from './serviceWorker';



// If you want your app to work offline and load faster, you can
change
// unregister() to register() below. Note this comes with some
pitfalls.
// Learn more about service workers: https://bit.ly/CRA-PWA
serviceWorker.unregister();

// 独立运行子应用，便于开发调试
if(!window.__POWERED_BY_QIANKUN__){
  render();
}

function render () {
  ReactDOM.render(
    <React.StrictMode>
      <App />
    </React.StrictMode>,
    document.getElementById('root')
  );
}
// 只需要导出几个方法就行
export async function bootstrap () { }
export async function mount () {
  render();
}
export async function unmount () {
  ReactDOM.unmountComponentAtNode(document.getElementById('root'));
}
```