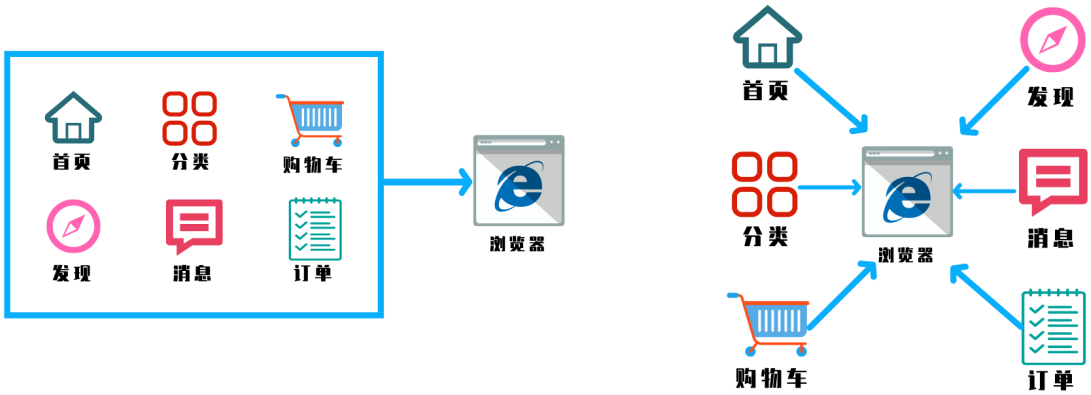
全面搞定微前端

- 课程安排：
  - 本周二（`SystemJS`原理 + single-spa实战）
  - 本周四（从零手写single-spa）
  - 下周二 （`qiankun`实战 + `qiankun`源码剖析）
  - 下周四 （`webpack`联邦模块实现微前端 （原理剖析） + `EMP`实战）

一.微前端能干什么?

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**!



1.微前端解决的问题

- 不同团队（技术栈不同），同时开发一个应用
- 每个团队开发的模块都可以独立开发，独立部署
- 实现增量迁移

2.如何实现微前端

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模块。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。 （子应用需要暴露固定的钩子 bootstrap、mount、unmount）

- `iframe`、`webComponent`
- 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)
- 2020年`EMP`基于module Federation，接入成本低， 解决第三方依赖包问题

二.`SystemJS`

`SystemJS` 是一个通用的模块加载器，它能在浏览器上动态加载模块。微前端的核心就是加载微应用，我们将应用打包成模块，在浏览器中通过`SystemJS`来加载模块。

1.搭建React开发环境

```
npm init -y
npm install webpack webpack-cli webpack-dev-server babel-loader
@babel/core @babel/preset-env @babel/preset-react html-webpack-plugin
-D
npm install react react-dom
```

`webpack.config.js`

```
const HtmlWebpackPlugin = require('html-webpack-plugin');
const path = require('path');
module.exports = () => {
    return {
        mode: 'development',
        output: {
            filename: 'index.js',
            path: path.resolve(__dirname, 'dist'),
            libraryTarget: ''
        },
        module: {
            rules: [{
                test: /\.js$/,
                use: { loader: 'babel-loader' },
                exclude: /node_modules/
            }]
        },
        plugins: [
            new HtmlWebpackPlugin({
                template: './public/index.html'
            })
        ]
    }
}
```

`.babelrc`

```
{
    "presets": [
        "@babel/preset-env",
        "@babel/preset-react"
    ]
}
```

`package.json`

```
"scripts": {
    "dev": "webpack serve"
}
```

2.`SystemJS`模块打包

```
"scripts": {
    "dev": "webpack serve",
    "build": "webpack --env production" // 增加环境变量区分环境
}
```

```
const HtmlWebpackPlugin = require('html-webpack-plugin');
const path = require('path');
module.exports = (env) => {
    return {
        mode: 'development',
        output: {
            filename: 'index.js',
            path: path.resolve(__dirname, 'dist'),
            libraryTarget: env.production ? 'system' : '' // 指定
SystemJS模块
        },
        module: {
            rules: [{
                test: /\.js$/,
                use: { loader: 'babel-loader' },
                exclude: /node_modules/
            }]
        },
        plugins: [
            !env.production && new HtmlWebpackPlugin({ // 不生成html
                template: './public/index.html'
            }),
        ].filter(Boolean),
```

```
        externals: env.production ? ['react', 'react-dom'] : [] // 不
打包React
    }
}
```

> 打包出的结果就是`SystemJS`的格式

3.浏览器加载模块

```
<script type="systemjs-importmap">
{
    "imports":{

"react":"https://cdn.bootcdn.net/ajax/libs/react/17.0.2/umd/react.pro
duction.min.js",
        "react-dom":"https://cdn.bootcdn.net/ajax/libs/react-
dom/17.0.2/umd/react-dom.production.min.js"
    }
}
</script>
<div id="root"></div>
<script
src="https://cdn.bootcdn.net/ajax/libs/systemjs/6.10.1/system.min.js"
></script>
<script>
    System.import('./index.js')
</script>
```

4.手写`SystemJS`原理

> 加载system模块后，会自动调用register方法将依赖和回调函数传入

```
System.register(["react","react-dom"], function(__WEBPACK_DYNAMIC_EXPORT__, __system_context__) {
    var __WEBPACK_EXTERNAL_MODULE_react__ = {};
    var __WEBPACK_EXTERNAL_MODULE_react_dom__ = {};
    Object.defineProperty(__WEBPACK_EXTERNAL_MODULE_react__, "__esModule", { value: true });
    Object.defineProperty(__WEBPACK_EXTERNAL_MODULE_react_dom__, "__esModule", { value: true });
    return {
        setters: [···
        ],
        execute: function() {···
        }
    };
});
```

```
function SystemJS(){}
function load(id){
    return new Promise((resolve,reject)=>{
        const script = document.createElement('script');
        script.src = id;
        script.async = true;
        document.head.appendChild(script);
```

```
        script.addEventListener('load',function(){
            let _lastRegister = lastRegister;
            lastRegister = undefined;
            if (!_lastRegister) {
                resolve([
                    [],
                    function(_export) {
                        return {
                            execute: function(_export) {
                                let obj = getGlobalLastPro();
                                _export(obj)
                            }
                        }
                    }
                ]); // 不是system.js 给默认值
            }
            resolve(_lastRegister); // 文件加载完毕后，会将
System.register的参数回传回来
        });
    })
}
let lastRegister;
SystemJS.prototype.import = function (id) {
    return new Promise((resolve,reject)=>{
        const lastSepIndex = location.href.lastIndexOf('/');
        const baseUrl = location.href.slice(0,lastSepIndex + 1);
        if(id.startsWith('./')){
            resolve(baseUrl + id.slice(2))
        }
    }).then(id=>{
        return load(id).then(()=>{
            // todo..
        })
    })
}
let lastRegister;
SystemJS.prototype.register = function (deps,declare) {
    lastRegister = [deps,declare]
}
let System = new SystemJS();
System.import('./index.js');
```

依赖加载

```
SystemJS.prototype.import = function (id) {
    // ...
    let e;
    return load(id).then((registration) => {
        function _export(result) {
```

```
            console.log(result)
        }
        let declared = registration[1](_export);
        e = declared.execute
        return [registration[0], declared.setters];
    }).then((instantiation) => { // 加载文件后加载依赖文件
        return Promise.all(instantiation[0].map((dep, i) => {
            var setter = instantiation[1][i];
            return load(dep).then(r => {
                let p = getGlobalLastPro();
                setter(p); // 将属性赋值给webpack中的变量
            })
        }))
    }).then(() => {
        e()
    })
}
```

> 对比window上新增的属性，返回新添加的属性

```
let globalMap = new Set()
let saveGlobalPro = () => {
    for (let p in window) {
        globalMap.add(p)
    }
}
saveGlobalPro();
let getGlobalLastPro = () => {
    let result;
    for (let p in window) {
        if (globalMap.has(p)) continue;
        result = window[p]
        result.default = result
        result.__useDefault = true;
    }
    return result
}
```

实现模块递归加载

```
function createLoad(id) {
    let e;
    return load(id).then((registration) => {
        // 加载文件后会将依赖和对应的回调传递过来
        function _export(key) {
            console.log(key)
        }
        let declared = registration[1](_export); // 获取函数的结果
        e = declared.execute;
```

```
            return [registration[0], declared.setters];
        }).then((deps) => {
            return Promise.all(deps[0].map((dep, i) => {
                let setter = deps[1][i];
                return createLoad(dep).then(() => {
                    let p = getGlobalLastPro();
                    setter(p);
                })
            }))
        }).then(() => {
            e();
        })
    }
```

三.SingleSpa实战

> single-spa是一个实现微前端架构的框架

通过脚手架创建应用

```
npm install create-single-spa -g
```

1.base基座项目

```
create-single-spa base
```

```
C:\Users\test1\Desktop\project>create-single-spa base
? Select type to generate single-spa root config
? Which package manager do you want to use? yarn
? Will this project use Typescript? No
? Would you like to use single-spa Layout Engine No
? Organization name (can use letters, numbers, dash or underscore) zf
Initialized empty Git repository in C:/Users/test1/Desktop/project/base/.git/
```

`zf-root-config.js`

```
import { registerApplication, start } from "single-spa";
registerApplication({
  name: "@single-spa/welcome",  // 应用名字
  app: () =>                    // 加载的应用
    System.import(
      "https://unpkg.com/single-spa-welcome/dist/single-spa-
welcome.js"
    ),

  activeWhen: ["/"],            // 路径匹配
});
start({
  urlRerouteOnly: true, // 全部使用SingleSpa中的reroute管理路由
});
```

- `registerApplication`注册应用并加载应用
- `start`启动应用

`index.ejs`

```
<!-- 当加载基座时,需要导入single-spa -->
<script type="systemjs-importmap">
    {
      "imports": {
        "single-spa": "https://cdn.jsdelivr.net/npm/single-
spa@5.9.0/lib/system/single-spa.min.js"
      }
    }
</script>
<!-- 对SingleSpa实现预加载 -->
<link rel="preload" href="https://cdn.jsdelivr.net/npm/single-
spa@5.9.0/lib/system/single-spa.min.js" as="script">
<!-- 需要systemJS模块化和amd解析 -->
<script
src="https://cdn.jsdelivr.net/npm/systemjs@6.8.3/dist/system.min.js">
</script>
<script
src="https://cdn.jsdelivr.net/npm/systemjs@6.8.3/dist/extras/amd.min.
js"></script>

<script>
    System.import('@zf/root-config'); // 加载基座应用
</script>
```

2.Vue应用

```
create-single-spa vue-app
```

```
? Please pick a preset: Manually select features
? Check the features needed for your project: Choose Vue version, Babel, Router
? Choose a version of Vue.js that you want to start the project with 3.x (Preview)
? Use history mode for router? (Requires proper server setup for index fallback in production) Yes
? Where do you prefer placing config for Babel, ESLint, etc.? In dedicated config files
? Save this as a preset for future projects? No
```

`vue.config.js`

```
module.exports = {
    devServer:{
        port: 3000
    }
}
```

基座中注册应用

```
registerApplication({
    name: "@zf/vue",
    app: () => System.import("@zf/vue"),
    activeWhen: location => location.pathname.startsWith('/vue'),
});
```

3.React应用

```
create-single-spa react-app
```

```
C:\Users\test1\Desktop\project>create-single-spa react-app
? Select type to generate single-spa application / parcel
? Which framework do you want to use? react
? Which package manager do you want to use? yarn
? Will this project use Typescript? No
? Organization name (can use letters, numbers, dash or underscore) zf
? Project name (can use letters, numbers, dash or underscore) react
Initialized empty Git repository in C:/Users/test1/Desktop/project/react-app/.git/
```

`zf-react.js`

```
import React from "react";
import ReactDOM from "react-dom";
import singleSpaReact from "single-spa-react";
import Root from "./root.component";

const lifecycles = singleSpaReact({ // 通过single-spa-react生成对应的生命周期
  React,
  ReactDOM,
  rootComponent: Root,
  errorBoundary(err, info, props) {
    // Customize the root error boundary for your microfrontend here.
    return null;
  },
});
```

```
export const { bootstrap, mount, unmount } = lifecycles; // 导出约定好的协议
```

> 更改项目启动端口

```
wepack-serve --port 4000
```

> 基座中注册React应用

```
registerApplication({
    name: "@zf/react",
    app: () => System.import("@zf/react"),
    activeWhen: location => location.pathname.startsWith('/react'),
// 当路径为/react时开始加载
});
```

```
<script type="systemjs-importmap">
    {
      "imports": {
        "@zf/root-config": "//localhost:9000/zf-root-config.js",
        "@zf/react":"//localhost:4000/zf-react.js" // 添加importMap
      }
    }
</script>
```

> 配置React路由系统

```
import { BrowserRouter as Router, Route, Link, Switch, Redirect }
from 'react-router-dom'
import Home from './components/Home.js'
import About from './components/About.js'

export default function Root(props) {
  return <Router basename="/react">
    <div>
      <Link to="/">Home React</Link>
      <Link to="/about">About React</Link>
    </div>
    <Switch>
      <Route path="/"  exact={true} component={Home}></Route>
      <Route path="/about" component={About}></Route>
      <Redirect to="/"></Redirect>
    </Switch>
  </Router>
}
```

> 在基座中配置`importMap`

```
{
    "react-router-dom"":https://cdn.bootcdn.net/ajax/libs/react-router-dom/5.2.0/react-router-dom.min.js
}
```

打包时需要配置路由的`externals`属性

```
externals:['react-router-dom']
```

4.`Parcel`应用

```
create-single-spa parcel
```

```
? Please pick a preset: Manually select features
? Check the features needed for your project: Choose Vue version, Babel, Router
? Choose a version of Vue.js that you want to start the project with 2.x
? Use history mode for router? (Requires proper server setup for index fallback in production)
? Where do you prefer placing config for Babel, ESLint, etc.? In dedicated config files
? Save this as a preset for future projects? No
```

```
<template>
  <div id="app">
    <div id="nav">
      <router-link to="/react">React应用</router-link> |
      <router-link to="/vue">Vue应用</router-link>
    </div>
    <router-view/>
  </div>
</template>
```

`vue.config.js`

```
module.exports = {
    chainWebpack:config=> config.externals(['vue','vue-router']),
    devServer:{
        port:5000
    }
}
```

基座中配置`importMap`

```
 {
     "imports": {
         "vue":"https://cdn.bootcdn.net/ajax/libs/vue/2.6.13/vue.js",
         "vue-router":"https://cdn.bootcdn.net/ajax/libs/vue-router/3.5.2/vue-router.js"
     }
 }
```

- react中使用`Parcel`组件

```
import Parcel from 'single-spa-react/parcel'
<Parcel config={System.import('@zf/parcel')}></Parcel>
```

- vue中使用`Parcel`组件

```
<Parcel :config="config" :mountParcel="mountRootParcel"></Parcel>
<script>
import Parcel from 'single-spa-vue/dist/esm/parcel';
import {mountRootParcel} from 'single-spa'
export default defineComponent({
  components:{
    Parcel
  },
  setup() {
    return {
      config:window.System.import('@zf/parcel'),
      mountRootParcel
    }
  },
})
</script>
```

5.应用间通信

- 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

```
import { registerApplication, start } from "single-spa";
// 公共数据
let customProps = { school: 'zf' }
registerApplication({
    name: "@zf/react",
    app: () => System.import("@zf/react"),
    activeWhen: location => location.pathname.startsWith('/react'),
    customProps
});
registerApplication({
    name: "@zf/vue",
    app: () => System.import("@zf/vue"),
    activeWhen: location => location.pathname.startsWith('/vue'),
    customProps
});
start({
    urlRerouteOnly: true,
```

```
});
```

> 可以在基座中通过共享容器的方式进行通信，实现应用之间的通信