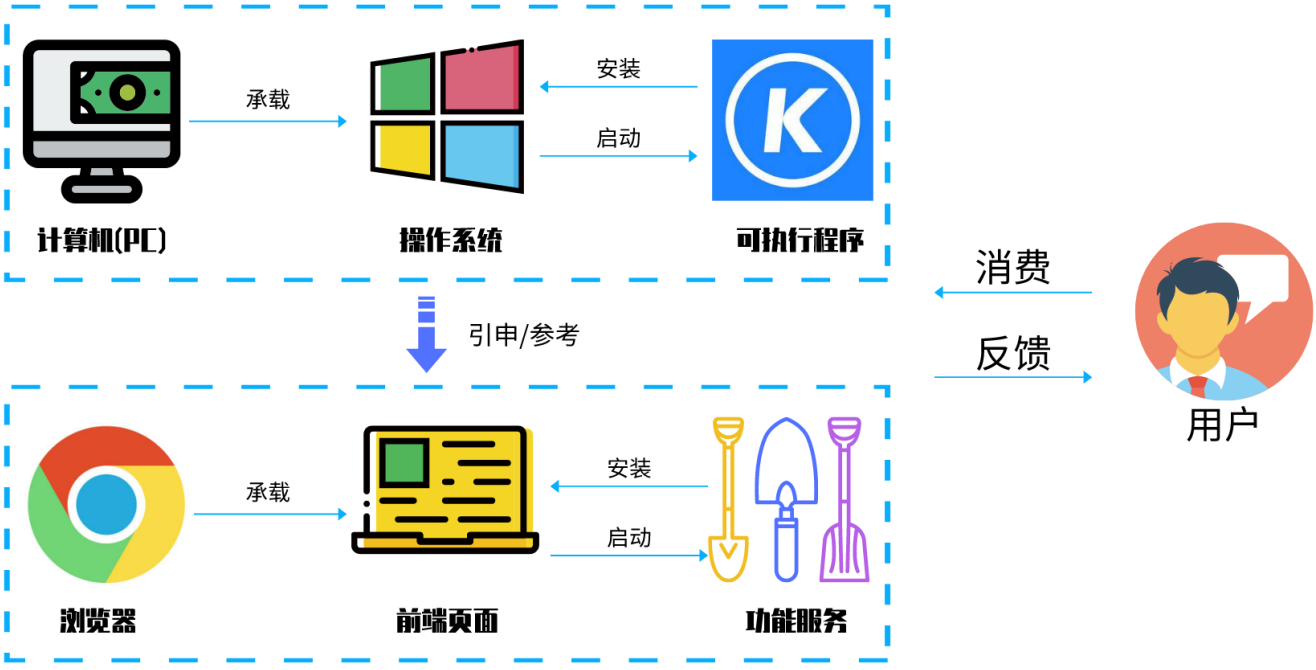
# 微前端从实战到源码

## 一.为什么需要微前端?

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，实现分而治之!**




## 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。**

# 3.实现微前端技术方案

- 采用何种方案进行应用拆分?
- 采用何种方式进行应用通信?
- 应用之间如何进行隔离?

## 1）iframe

- 微前端的最简单方案，通过iframe加载子应用。
- 通信可以通过postMessage进行通信。
- 完美的沙箱机制自带应用隔离。

> 缺点：用户体验差 （弹框只能在iframe中、在内部切换刷新就会丢失状态）

## 2）Web Components

- 将前端应用程序分解为自定义 HTML 元素。
- 基于`CustomEvent`实现通信

- Shadow DOM天生的作用域隔离

> 缺点：浏览器支持问题、学习成本、调试困难、修改样式困难等问题。

### 3）single-spa

- single-spa 通过路由劫持实现应用的加载（采用SystemJS），提供应用间公共组件加载及公共业务逻辑处理。**子应用需要暴露固定的钩子 bootstrap、mount、unmount）接入协议。**
- 基于props主子应用间通信
- 无沙箱机制，需要实现自己实现JS沙箱以及CSS沙箱

> 缺点：学习成本、无沙箱机制、需要对原有的应用进行改造、子应用间相同资源重复加载问题。

### 4）Module federation

- 通过模块联邦将组件进行打包导出使用
- 共享模块的方式进行通信
- 无CSS沙箱和JS沙箱

> 缺点：需要webpack5。

## 二.SystemJS剖析

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。

```
 const newMapUrl = {}
 function processScripts() {

Array.from(document.querySelectorAll('script')).
forEach(script => {
     if (script.type === 'systemjs-importmap') {
       const imports =
JSON.parse(script.innerHTML).imports;
       // 映射包的路径
       Object.entries(imports).forEach(([key,
value]) => newMapUrl[key] = value);
     }
   })
 }
function load(id) {
  return new Promise((resolve, reject) => {
    const script =
document.createElement('script');
    id = newMapUrl[id] || id
    script.src = id;
    script.async = true;
    document.head.appendChild(script);
```

```
    script.addEventListener('load', function ()
{
      let _lastRegister = lastRegister;
      lastRegister = undefined;
      resolve(_lastRegister);
    })
  })
}
let lastRegister;
let globalMap = new Set()
let saveGlobalPro = () => {
  for (let p in window) {
    globalMap.add(p)
  }
}
saveGlobalPro();
function getGlobalLastPro() {
  let result;
  for (let p in window) {
    if (globalMap.has(p)) continue;
    result = window[p]
  }
  return result
}
class SystemJS {
  import(id) {
    // 1.将script脚本进行解析
```

```
    return Promise.resolve(processScripts())
      .then(() => {
      // 2.获取最后一个斜杠的位置 + 文件名
      const lastSepIndex =
location.href.lastIndexOf('/');
      const baseUrl = location.href.slice(0,
lastSepIndex + 1);
      if (id.startsWith('./')) {
        return baseUrl + id.slice(2);
      }
    })
      .then(id => {
      // 根据路径加载文件
      let execute;
      return load(id)
        .then(registeration => {
        function _export(result) { }
        let declared = registeration[1]
(_export);
        execute = declared.execute;
        // 依赖列表，将在的依赖结果调用setters进行设置
        return [registeration[0],
declared.setters]
      })
      .then(([registration, setters]) => {
        return
Promise.all(registration.map((dep, i) => {
```

```
            const setter = setters[i];// 加载依赖
            return load(dep).then(() => {
              let p = getGlobalLastPro();
              setter(p);
            })
          }))
        })
          .then(() => {
          execute()
        })
      })
    }
    register(deps, declare) {
      // 执行打包的模块
      lastRegister = [deps, declare];
    }
}
const System = new SystemJS()
```

# 三.single-spa实战

## 1.安装脚手架

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

```
# 创建子项目
single-spa application / parcel
# 用于跨应用共享JavaScript逻辑的微应用
in-browser utility module (styleguide, api
cache, etc)
# 创建基座容器
> single-spa root config
```

> 生成基座项目，用于加载子应用

# 2.生成react子应用

```
create-single-spa react-project
```

## 1）配置路由

```
npm install react-router-dom
```

```
import { BrowserRouter as Router, Route, Link,
Routes }
  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>
      <Routes>
          <Route path="/" element={<Home />} />
          <Route path="/about" element={<About />}
/>
      </Routes>
    </Router>
}
```

## 2）webpack.config.js配置

```
delete defaultConfig.externals; // 关闭externals
return merge(defaultConfig, {
  devServer:{
    port:3001 // 修改端口
  }
});
```

## 3）注册子应用

```
<script type="systemjs-importmap">
    {
      "imports": {
        "@jw/root-config": "//localhost:9000/jw-root-config.js",
        "@jw/react":"//localhost:3001/jw-react.js"
      }
    }
</script>
```

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

# 3.生成vue子应用

```
create-single-spa vue-project
```

## 1）vue.config.js配置

```
const { defineConfig } = require('@vue/cli-service')
module.exports = defineConfig({
  transpileDependencies: true,
  publicPath: 'http://localhost:3002',
  devServer: {
    port: 3002
  },
  chainWebpack: (config) => {
    if (config.plugins.has("SystemJSPublicPathWebpackPlugin")) {

 config.plugins.delete("SystemJSPublicPathWebpackPlugin");
    }
  }
})
```

> 修改baseURL

```
const router = createRouter({
  history: createWebHistory('/vue'),
  routes
})
```

## 2）注册子应用

```
<script type="systemjs-importmap">
    {
      "imports": {
        "@jw/root-config": "//localhost:9000/jw-root-config.js",
        "@jw/react":"//localhost:3001/jw-react.js",
        "@jw/vue":"//localhost:3002/js/app.js"
      }
    }
</script>
```

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

# 4.生产parcel应用

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

```
const { defineConfig } = require('@vue/cli-service')
module.exports = defineConfig({
  transpileDependencies: true,
  devServer: {
    port: 8000
  },
  chainWebpack: (config) => {
    if (config.plugins.has("SystemJSPublicPathWebpackPlugin")) {

 config.plugins.delete("SystemJSPublicPathWebpackPlugin");
    }
  }
})
```

注册Parcel应用

```
<script type="systemjs-importmap">
    {
      "imports": {
        "@jw/root-config": "//localhost:9000/jw-
root-config.js",
        "@jw/react":"//localhost:3001/jw-
react.js",
        "@jw/vue":"//localhost:3002/js/app.js",

"@jw/parcel":"//localhost:8000/js/app.js"
      }
    }
</script>
```

## 1）在react应用中加载

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

## 2）在vue应用中加载

```
<template>
  <nav>
    <router-link to="/">Home</router-link> |
    <router-link to="/about">About</router-link>
    <Parcel :config="config"
:mountParcel="mountRootParcel"></Parcel>
  </nav>
  <router-view />
</template>
<script setup>
import Parcel from 'single-spa-vue/parcel'
import { mountRootParcel } from 'single-spa'
const config = System.import("@jw/parcel");
</script>
```

## 5.应用通信

```
const state = { age: 30 }
registerApplication({
  name: "@jw/react",
  app: () => System.import('@jw/react'),
  activeWhen: (location) =>
location.pathname.startsWith('/react'),
  customProps:state
});
registerApplication({
  name: "@jw/vue",
  app: () => System.import("@jw/vue"),
  activeWhen: location =>
location.pathname.startsWith('/vue'),
  customProps:state
});
```

```
export function bootstrap(props) {
  return lifecycles.bootstrap(props);
}
export function mount(props) {
  return lifecycles.mount(props);
}
export function unmount(props) {
  return lifecycles.unmount(props);
}
```