# 乾坤

qiankun的相关学习地址：

1.【官方文档地址】

<https://qiankun.umijs.org/zh/guide>

2.【GitHub地址】

<https://github.com/umijs/qiankun>

3.【快速在qiankun基础上搭建的使用demo，GitHub开源】

<https://github.com/a1029563229/micro-front-template>

## 入门教程

### 路由模式选择

react-router，angular-router，vue-router 这三种路由，都支持 hash 和 history 模式，微应用使用不同的模式在 qiankun 中略有差别。

#### activeRule 使用 location.pathname 区分微应用

主应用使用 location.pathname 来区分微应用时，微应用可以是 hash 和 history 模式。

注册微应用时 activeRule 这样写即可：

registerMicroApps([

{

name: 'app',

entry: 'http://localhost:8080',

container: '#container',

activeRule: '/app',

},

]);

#### 微应用路由模式设置区别

1. 当微应用是 history 模式时，设置路由 base 即可
2. 当微应用是 hash 模式时，三种路由的表现不一致

| **路由** | **主应用跳转/app/#/about** | **特殊配置** |
| --- | --- | --- |
| vue-router | 响应 about 路由 | 无 |
| react-router | 不响应 about 路由 | 无 |
| angular-router | 响应 about 路由 | 需要设置 --base-href |

angular 应用在 package.json 里面设置 --base-href：

- "start": "ng serve",

+ "start": "ng serve --base-href /angular9",

- "build": "ng build",

+ "build": "ng build --base-href /angular9",

打包部署后，angular 微应用可以被主应用访问。但是独立访问时，懒加载的路由会报错，路径不正确。这里有两个解决办法：

方法1：修改 public-path.js 为：

\_\_webpack\_public\_path\_\_ = window.\_\_POWERED\_BY\_QIANKUN\_\_ ? window.\_\_INJECTED\_PUBLIC\_PATH\_BY\_QIANKUN\_\_ : `http://${ip}:${port}/`; // 填写你的实际部署地址

方法2：修改打包命令，并且将微应用部署在 angular9 目录：

- "build": "ng build",

+ "build": "ng build --base-href /angular9 --deploy-url /angular9/",

#### activeRule 使用 location.hash 区分微应用

当微应用都是 hash 模式时可以使用 hash 区分微应用，主应用的路由模式不限。

注册微应用时 activeRule 需要这样写：

const getActiveRule = hash => location => location.hash.startsWith(hash);

registerMicroApps([

{

name: 'app-hash',

entry: 'http://localhost:8080',

container: '#container',

activeRule: getActiveRule('#/app-hash'),

// 这里也可以直接写 activeRule: '#/app-hash'，但是如果主应用是 history 模式或者主应用部署在非根目录，这样写不会生效。

},

]);

react-router 和 angular-router 微应用需要设置 activeRule 的值为路由的 base ，写法同 history 模式。

vue-router 的 hash 模式下不支持设置路由的 base，需要额外新建一个空的路由页面，将其他所有路由都作为它的 children：

const routes = [

{

path: '/app-vue-hash',

name: 'Home',

component: Home,

children: [

// 其他的路由都写到这里

]

}

]

#### 同时存在多个微应用时

如果一个页面同时展示多个微应用，需要使用 loadMicroApp 来加载。

如果这些微应用都有路由跳转的需求，要保证这些路由能互不干扰，需要使用 momery 路由。vue-router 使用 abstract 模式，react-router 使用 memory history 模式，angular-router 不支持。

## 项目开启操作

### 主应用

主要步骤：

1. 注册微应用
2. 为每一个微应用提供dom容器
3. 启动主应用

#### 安装qiankun

$ yarn add qiankun # 或者 npm i qiankun -S

#### 在主应用中注册微应用

1. import { registerMicroApps, start } from 'qiankun';
2. registerMicroApps([
3. {
4. name: 'react app', // app name registered
5. entry: '//localhost:7100',
6. container: '#yourContainer',
7. activeRule: '/yourActiveRule',
8. },
9. {
10. name: 'vue app',
11. entry: { scripts: ['//localhost:7100/main.js'] },
12. container: '#yourContainer2',
13. activeRule: '/yourActiveRule2',
14. },
15. ]);
16. start();

当微应用信息注册完之后，一旦浏览器的 url 发生变化，便会自动触发 qiankun 的匹配逻辑，所有 activeRule 规则匹配上的微应用就会被插入到指定的 container 中，同时依次调用微应用暴露出的生命周期钩子。

如果微应用不是直接跟路由关联的时候，你也可以选择手动加载微应用的方式：

import { loadMicroApp } from 'qiankun';

loadMicroApp(

{

name: 'app',

entry: '//localhost:7100',

container: '#yourContainer',

}

);

### 微应用

微应用不需要额外安装任何其他依赖即可接入 qiankun 主应用。

1. 入口js文件平级目录下增加public-path.js文件；
2. 入口js文件中引入public-path.js，修改并导出qiankun定义的三个钩子函数：bootstrap、mount、unmount；
3. 使用history路由，并且路由base值要和activeRule匹配规则保持一致；
4. 修改webpack配置，允许开发环境跨域及umd打包；

#### 导出响应的生命周期钩子

微应用需要在自己的入口 js (通常就是你配置的 webpack 的 entry js) 导出 bootstrap、mount、unmount 三个生命周期钩子，以供主应用在适当的时机调用。

/\*\*

\* bootstrap 只会在微应用初始化的时候调用一次，下次微应用重新进入时会直接调用 mount 钩子，不会再重复触发 bootstrap。

\* 通常我们可以在这里做一些全局变量的初始化，比如不会在 unmount 阶段被销毁的应用级别的缓存等。

\*/

export async function bootstrap() {

console.log('react app bootstraped');

}

/\*\*

\* 应用每次进入都会调用 mount 方法，通常我们在这里触发应用的渲染方法

\*/

export async function mount(props) {

ReactDOM.render(<App />, props.container ? props.container.querySelector('#root') : document.getElementById('root'));

}

/\*\*

\* 应用每次 切出/卸载 会调用的方法，通常在这里我们会卸载微应用的应用实例

\*/

export async function unmount(props) {

ReactDOM.unmountComponentAtNode(props.container ? props.container.querySelector('#root') : document.getElementById('root'));

}

/\*\*

\* 可选生命周期钩子，仅使用 loadMicroApp 方式加载微应用时生效

\*/

export async function update(props) {

console.log('update props', props);

}

## 乾坤API说明

### initGlobalState(state)

定义全局状态，并返回通信方法，建议在主应用使用，微应用通过 props 获取通信方法。

* onGlobalStateChange: (callback: OnGlobalStateChangeCallback, fireImmediately?: boolean) => void， 在当前应用监听全局状态，有变更触发 callback，fireImmediately = true 立即触发 callback
* setGlobalState: (state: Record<string, any>) => boolean， 按一级属性设置全局状态，微应用中只能修改已存在的一级属性
* offGlobalStateChange: () => boolean，移除当前应用的状态监听，微应用 umount 时会默认调用

示例：

主应用：

import { initGlobalState, MicroAppStateActions } from 'qiankun';

//父应用的初始state

// Vue.observable是为了让initialState变成可响应：https://cn.vuejs.org/v2/api/#Vue-observable。

let state = Vue.observable({

user: {},

});

// 初始化 state

const actions: MicroAppStateActions = initGlobalState(state);

actions.onGlobalStateChange((state, prev) => {

// state: 变更后的状态; prev 变更前的状态

console.log(state, prev);

});

actions.setGlobalState(state);

actions.offGlobalStateChange();

微应用：

// 从生命周期 mount 中获取通信方法，使用方式和 master 一致

export function mount(props) {

props.onGlobalStateChange((state, prev) => {

// state: 变更后的状态; prev 变更前的状态

console.log(state, prev);

});

props.setGlobalState(state);

}

* Vue.observable是为了让父应用的state变成可响应式，如果不用Vue.observable包一层，它就只是一个纯粹的object，子应用也能获取到，但会失去响应式，**意味着数据改变后，页面不会更新**。
* getGlobalState方法，这个是**有争议**的，大家在github上有讨论：[github.com/umijs/qiank…](https://github.com/umijs/qiankun/pull/729" \t "_blank)。

一方面，作者认为getGlobalState不是必须的，onGlobalStateChange其实已经够用。

另一方面，笔者和其他提pr的同学觉得有必要提供一个getGlobalState的api，理由是get方法更方便使用，子应用有需求是不需一直监听stateChange事件，它只需要在首次mount时通过getGlobalState初始化一次即可。在这里，笔者先坚持己见让父应用下发一个getGlobalState的方法。

# 乾坤前身（single-spa）

## Single-spa简介

1. [Applications](https://www.bookstack.cn/read/single-spa-5.x-zh/building-applications.md)，每个应用程序本身就是一个完整的 SPA (某种程度上)。 每个应用程序都可以响应 url 路由事件，并且必须知道如何从 DOM 中初始化、挂载和卸载自己。 传统 SPA 应用程序和 Single SPA 应用程序的主要区别在于，它们必须能够与其他应用程序共存，而且它们没有各自的 html 页面。

例如，React 或 Angular spa 就是应用程序。 当激活时，它们监听 url 路由事件并将内容放在 DOM上。 当它们处于非活动状态时，它们不侦听 url 路由事件，并且完全从 DOM 中删除。

1. 一个 [single-spa-config](https://www.bookstack.cn/read/single-spa-5.x-zh/$configuration)配置, 这是html页面和向Single SPA注册应用程序的JavaScript。每个应用程序都注册了三件东西
   * A name
   * A function (加载应用程序的代码)
   * A function (确定应用程序何时处于活动状态/非活动状态)

## Single-spa安装

### 全局安装

1. npm install --global create-single-spa
2. # or
3. yarn global add create-single-spa

### 本地安装

1. npm init single-spa
2. # or
3. npx create-single-spa
4. # or
5. yarn create single-spa

注意：

这将触发CLI提示，询问你将创建或更新哪种项目。create-single-spa **会安装在当前目录**，因此如果从头开始，请确保创建一个空目录。