`qiankun`实战

- 简单：任意 `js` 框架均可使用。微应用接入像使用接入一个 `iframe` 系统一样简单，但实际不是 `iframe`。
- 完备：几乎包含所有构建微前端系统时所需要的基本能力，如 样式隔离、`js` 沙箱、预加载等。
- 生产可用：已在蚂蚁内外经受过足够大量的线上系统的考验及打磨，健壮性值得信赖。

一.主应用搭建

> 主应用我们采用react作为基座

```
create-react-app base
yarn add react-router-dom qiankun
```

渲染父应用导航

```
import {BrowserRouter as Router,Link} from 'react-router-dom'
function App() {
  return (
    <div className="App">
        <Router>
          <Link to="/vue">vue应用</Link>
          <Link to="/react">react应用</Link>
        </Router>
        {/* 路由切换时 应用渲染到这里 */}
        <div id="container"></div>
    </div>
  );
}
export default App;
```

> 接入`React`和`Vue`微应用 `registerApps.js`

```
import { registerMicroApps, start } from 'qiankun';
const loader = (loading) => {
    console.log(loading)
}
registerMicroApps([{
    name: 'vueApp',
    entry: '//localhost:20000',
    container: '#container',
    activeRule: '/vue',
    loader
}, {
    name: 'reactApp',
    entry: '//localhost:30000',
    container: '#container',
    activeRule: '/react',
    loader
}], {
    beforeLoad: () => {
        console.log('beforeLoad')
    },
    beforeMount: () => {
        console.log('beforeMount')
    },
    adterMount: () => {
        console.log('adterMount')
    },
    beforeUnmount: () => {
        console.log('beforeUnmount')
    },
    afterUnmount: () => {
        console.log('afterUnmount')
    }
})
start();
```

二.`Vue`微应用

```
vue create m-vue
```

```
? 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
```

改造`Vue`项目配置文件

```
module.exports = {
    publicPath:'http://localhost:20000', // 静态资源路径统一为2000端口
    devServer:{
        port: 20000, // 端口20000
        headers:{
            'Access-Control-Allow-Origin':'*' // 允许跨域
        }
    },
    configureWebpack: {
        output: {
          library: 'm-vue',
          libraryTarget: 'umd', // 把微应用打包成 umd 库格式
        },
     },
}
```

> 导出接入协议

```
import { createApp } from 'vue'
import App from './App.vue'
import routes from './router'
import { createRouter, createWebHistory } from 'vue-router'
let app = null;
let router = null;
let history = null;

function render(props = {}) {
    const { container } = props;
    history = createWebHistory('/vue');
    router = createRouter({
        history,
        routes
    })
    app = createApp(App)
    app.use(router).mount(container ? container.querySelector('#app')
: '#app')
}
if (!window.__POWERED_BY_QIANKUN__) {
    render() // 让子应用可以独立运行
}
export async function bootstrap() {
    // 提供启动方法
    console.log('vue3 app bootstraped')
}
export async function mount(props) {
    console.log('vue3 app mount');
    render(props)
}
export async function unmount() {
    app.unmount();
    app = null;
    router = null;
    history.destroy();
}
```

三.React微应用

```
create-react-app m-react
```

改造`react`项目配置文件

```
npm i -D @rescripts/cli
```

`.rescriptsrc.js`

```
module.exports = {
    webpack: (config) => {
        config.output.library = `m-react`;
        config.output.libraryTarget = 'umd';
        config.output.publicPath = 'http://localhost:30000/'
        return config;
    },
    devServer: (config) => {
        config.headers = {
            'Access-Control-Allow-Origin': '*',
        };
        return config;
    },
};
```

> 更改启动端口

```
PORT=30000
WDS_SOCKET_PORT=30000
```

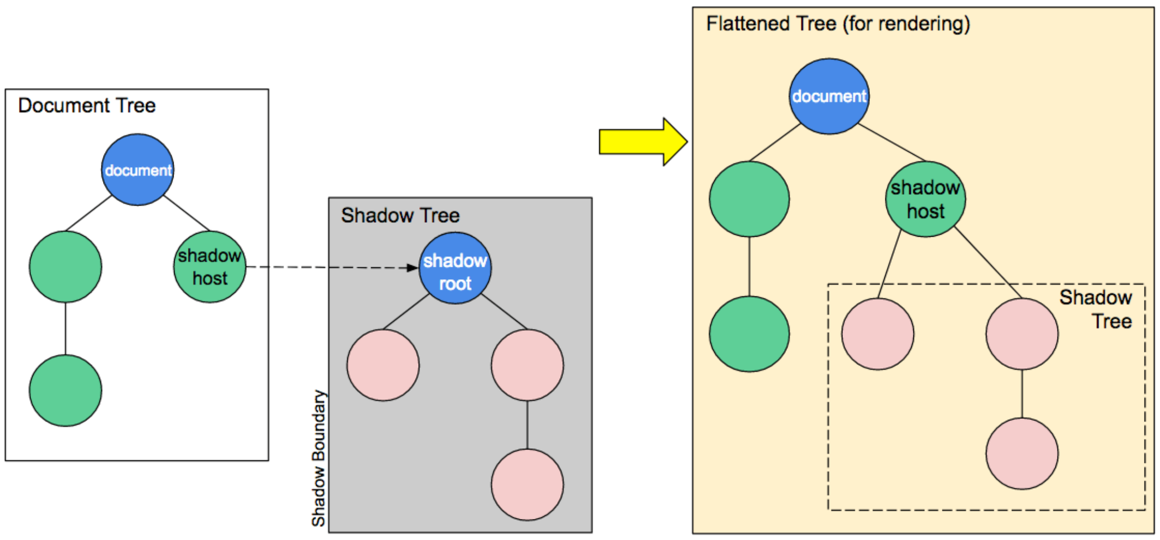
> 导出接入协议

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
function render(props = {}) {
  const { container } = props;
  ReactDOM.render(<App />, container ?
container.querySelector('#root') : document.querySelector('#root'));
}
if (!window.__POWERED_BY_QIANKUN__) {
  render();
}
export async function bootstrap() {
  console.log(' react app bootstraped');
}
export async function mount(props) {
  console.log('props from main framework');
  render(props);
}
export async function unmount(props) {
  const { container } = props;
  ReactDOM.unmountComponentAtNode(container ?
container.querySelector('#root') : document.querySelector('#root'));
}
```

```
start({
  sandbox: {
    strictStyleIsolation: true,  // 启用shadowDOM
    experimentalStyleIsolation:true // 增加一个特殊的选择器规则来限定其影
响范围
  }
});
```

四.`qiankun`中`CSS`隔离方案

- 子应用之间的样式隔离：`Dynamic Stylesheet`切换应用时将老应用样式移除
- 主应用和子应用之间的样式隔离：
  - `BEM(Block Element Modifier)`规范
  - `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` 不在推荐使用



```
const appContent = `<div id="qiankun">
```

```
        <div>hello world</div>
        <style>div{color:red}</style>
</div>`; // 好比qiankun中获取的html，我们拿到后包裹了一层
const containerElement = document.createElement('div');
containerElement.innerHTML = appContent;

const appElement = containerElement.firstChild // 拿出第一个儿子，中的内容
const { innerHTML } = appElement;

appElement.innerHTML = '';
let shadow = appElement.attachShadow({ mode: 'open' }); // 将父容器变为shadowDOM
shadow.innerHTML = innerHTML; // 将内容插入到shadowDOM中
document.body.appendChild(appElement);
```

```
start({
    sandbox:{
        strictStyleIsolation:true, // shadowDOM
        experimentalStyleIsolation:true // 实验性语法
    }
});
```

五.`qiankun`中`JS`隔离方案

1.`ProxySandbox`

支持Proxy时，可以创建一个对象代理window，子应用的操作都在代理window上。避免子应用污染全局变量

```
class ProxySandbox {
    constructor() {
        let fakeWindow = {} // 根据window创建个代理proxy
        this.sandboxRunning = true; // 沙箱是否正在运行
        const rawWindow = window
        const proxy = new Proxy(fakeWindow, {
            get(target, p) {
                return target[p] || rawWindow[p];
            },
            set: (target, p, value) => {
                if (this.sandboxRunning) {
                    target[p] = value;
                }
                return true
            }
        });
        this.proxy = proxy;
    }
    active() { // 沙箱激活
        this.sandboxRunning = true;
    }
    inactive() { // 沙箱失活
        this.sandboxRunning = false;
    }
}
let sandbox = new ProxySandbox();
```

2.`SnapshotSandbox`

给Window对象拍照，当更改时记录对`window`对象的修改。失活后还原window对象

```
class SnapshotSandbox {
    constructor() {
        this.proxy = window;
        this.modifyPropsMap = {}; // 修改了那些属性
    }
    active() {
        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];
        });
    }
    inactive() {
        for (const prop in window) { // diff 差异
            if (window.hasOwnProperty(prop)) {
                // 将上次拍照的结果和本次window属性做对比
                if (window[prop] !== this.windowSnapshot[prop]) {
                    // 保存修改后的结果
                    this.modifyPropsMap[prop] = window[prop];
                    // 还原window
                    window[prop] = this.windowSnapshot[prop];
                }
            }
        }
    }
}
```

六.应用之间通信

- 直接通过props来进行通信 （single-spa）
- 通过`qiankun`中提供的`initGlobalState`

```
const state = {
    name:'zf',
    age:12
}
const actions = initGlobalState(state);
actions.onGlobalStateChange((state,prev)=>{ // 基座中监听事件变化
    console.log(state,prev);
});
```

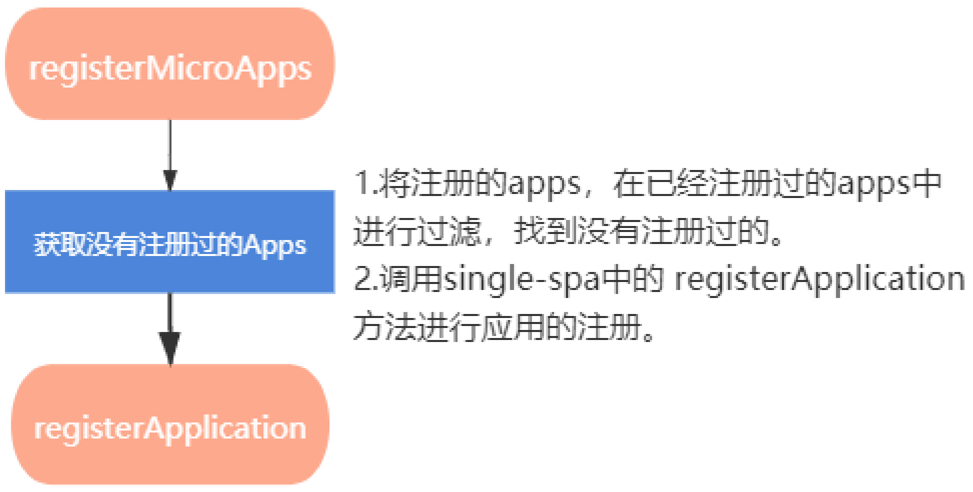
```
props.onGlobalStateChange((prev, next) => {}) // 子组件设置状态
props.setGlobalState({name:'jw',age:18}); // 子组件中更改状态
```

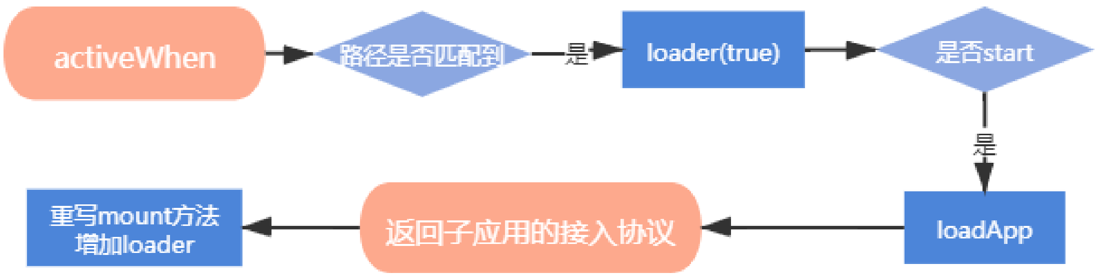
七.公共组件

```
import logo from './logo.svg';
import './App.css';
import { loadMicroApp } from 'qiankun';
import React,{useEffect} from 'react'
function App() {
  let containerRef = React.createRef();
  let microApp;
  useEffect(()=>{
    microApp = loadMicroApp({
      name: 'vue-parcel',
      entry: '//localhost:40000',
      container: containerRef.current,
    });
  })
  return (
    <div className="App">
      <div ref={containerRef}></div>
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <p>
          Edit <code>src/App.js</code> and save to reload.
        </p>
        <a
          className="App-link"
          href="https://reactjs.org"
          target="_blank"
          rel="noopener noreferrer"
        >
          Learn React
        </a>
      </header>
    </div>
  );
}
export default App;
```

`qiankun`原理剖析

1.解析`registerMicroApps`

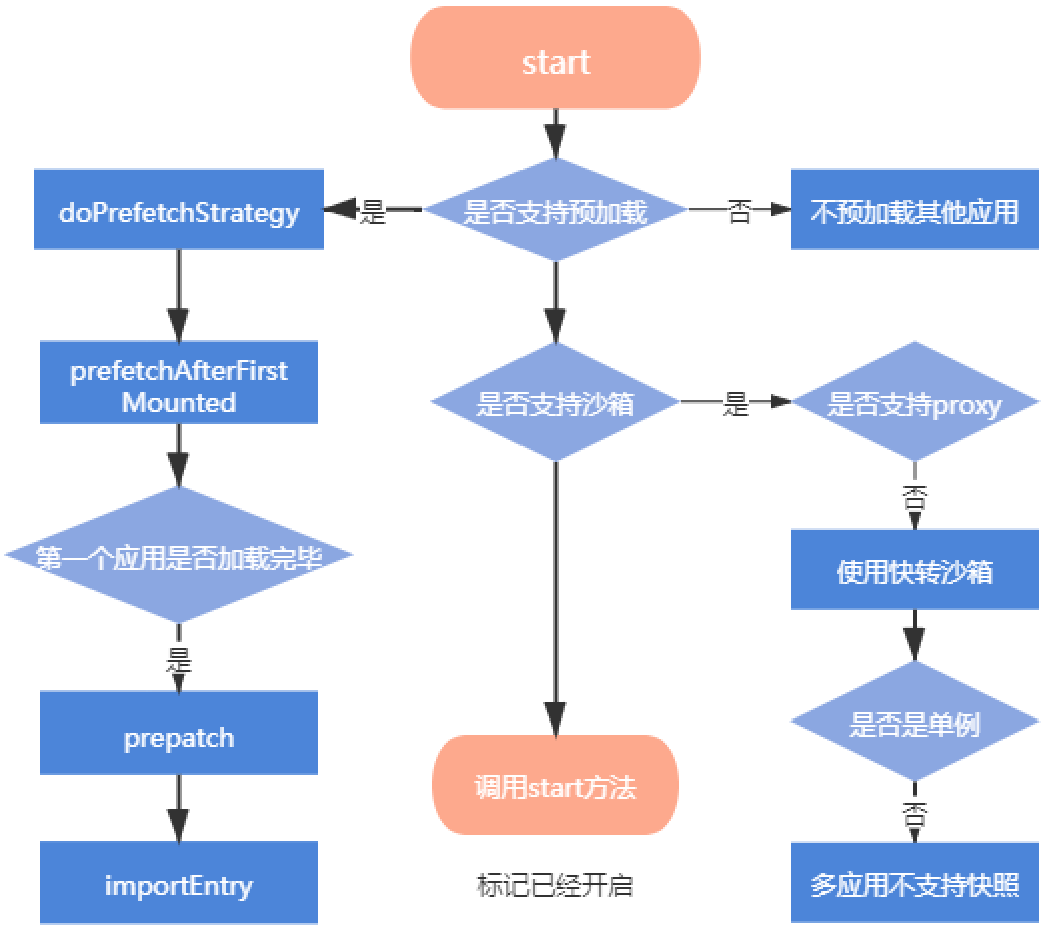

registerMicroApps
获取没有注册过的Apps
registerApplication
1.将注册的apps，在已经注册过的apps中进行过滤，找到没有注册过的。
2.调用single-spa中的 registerApplication方法进行应用的注册。



```
export function registerMicroApps<T extends ObjectType>(
  apps: Array<RegistrableApp<T>>, // 需要注册的应用
  lifeCycles?: FrameworkLifeCycles<T>, // 对应的生命周期
) {
  // 防止注册重复的应用 ，直接过滤掉重复的
  const unregisteredApps = apps.filter((app) =>
!microApps.some((registeredApp) => registeredApp.name === app.name));
  microApps = [...microApps, ...unregisteredApps];
  unregisteredApps.forEach((app) => { // 将需要注册的新应用，循环依次注册
    const { name, activeRule, loader = noop, props, ...appConfig } =
app;
```

```
    registerApplication({ // singleSpa 的应用注册函数
      name,
      app: async () => {
        loader(true); // 设置loadinhg
        await frameworkStartedDefer.promise; // 等待start方法被调用

        const { mount, ...otherMicroAppConfigs } = (
          // 加载应用， 获取生命周期钩子
          await loadApp({ name, props, ...appConfig },
frameworkConfiguration, lifeCycles)
        )();
        return { // 调用mount时可以
          mount: [async () => loader(true), ...toArray(mount), async
() => loader(false)],
          ...otherMicroAppConfigs,
        };
      },
      activeWhen: activeRule,
      customProps: props,
    });
  });
}
```

2.解析start方法



```
export function start(opts: FrameworkConfiguration = {}) {
  // 是否支出预加载, 是否支持单例模式,  是否支持沙箱  opts是用户传入的其他参数
  frameworkConfiguration = { prefetch: true, singular: true, sandbox:
true, ...opts };
  const {
    prefetch,
    sandbox,
    singular,
    urlRerouteOnly = defaultUrlRerouteOnly,
    ...importEntryOpts // 将用的参数放到importEntryOpts 选项中
  } = frameworkConfiguration;

  if (prefetch) { // 是否需要预先加载, 默认需要
    doPrefetchStrategy(microApps, prefetch, importEntryOpts); // 做预
先加载策略
  }
  // 是否需要启动沙箱
  if (sandbox) {
    if (!window.Proxy) { // 如果不支持proxy 则退到快照沙箱
      console.warn('[qiankun] Miss window.Proxy, proxySandbox will
degenerate into snapshotSandbox');
      frameworkConfiguration.sandbox = typeof sandbox === 'object' ?
{ ...sandbox, loose: true } : { loose: true };
      if (!singular) { // 如果不是单例模式, 还不支持window.proxy 则会报错
        console.warn(
          '[qiankun] Setting singular as false may cause unexpected
behavior while your browser not support window.Proxy',
        );
      }
    }
  }

  startSingleSpa({ urlRerouteOnly }); // 启动主应用,调用single-spa的
start方法
  started = true; // 表示开启
  frameworkStartedDefer.resolve(); // start成功后让promise变为成功态
}
```

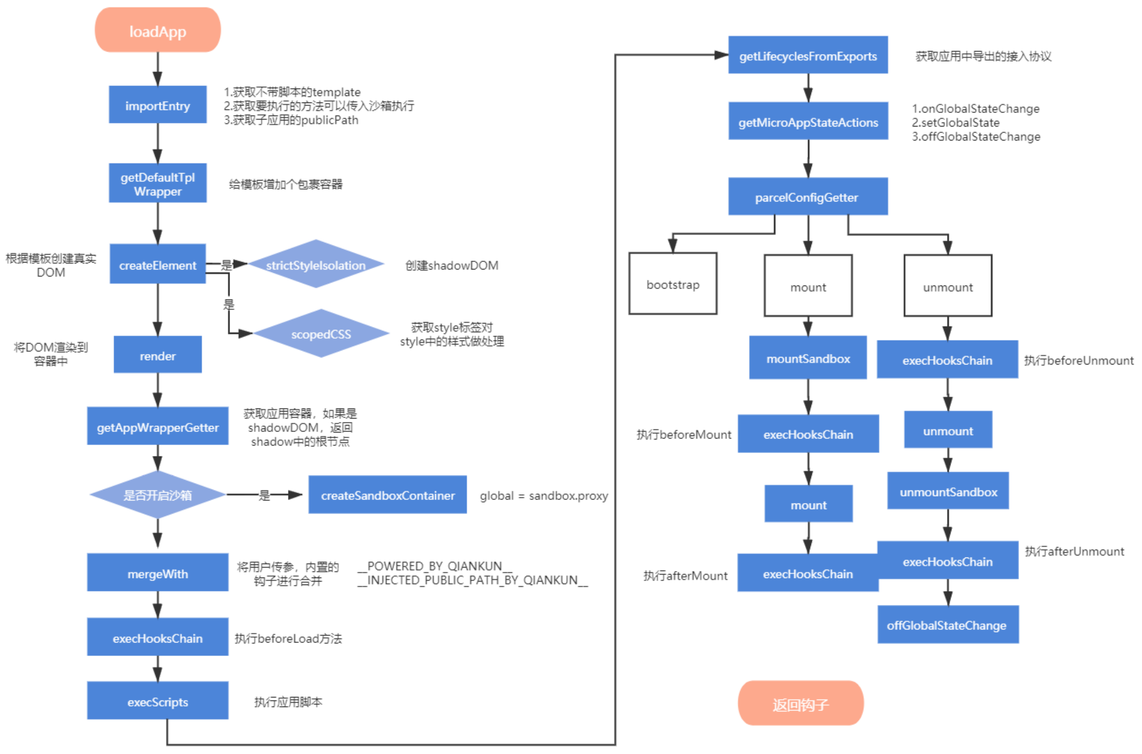
3.`prefetch`为true的情况

```
function prefetchAfterFirstMounted(apps: AppMetadata[], opts?:
ImportEntryOpts): void {
  window.addEventListener('single-spa:first-mount', function
listener() {
    // 监听第一个应用的挂在事件
    const notLoadedApps = apps.filter((app) => getAppStatus(app.name)
=== NOT_LOADED); // 过滤所有没加载的app
```

```
    if (process.env.NODE_ENV === 'development') {
      const mountedApps = getMountedApps();
      console.log(`[qiankun] prefetch starting after ${mountedApps}
mounted...`, notLoadedApps);
    }
    // 没加载的app全部去预先加载
    notLoadedApps.forEach(({ entry }) => prefetch(entry, opts));
    // 移除监听的事件
    window.removeEventListener('single-spa:first-mount', listener);
  });
}
```

> 这里会等待第一个应用加载完毕后，调用`importEntry`获取获取其他`app`资源

4.`loadApp`实现



当`start`后开始去加载应用

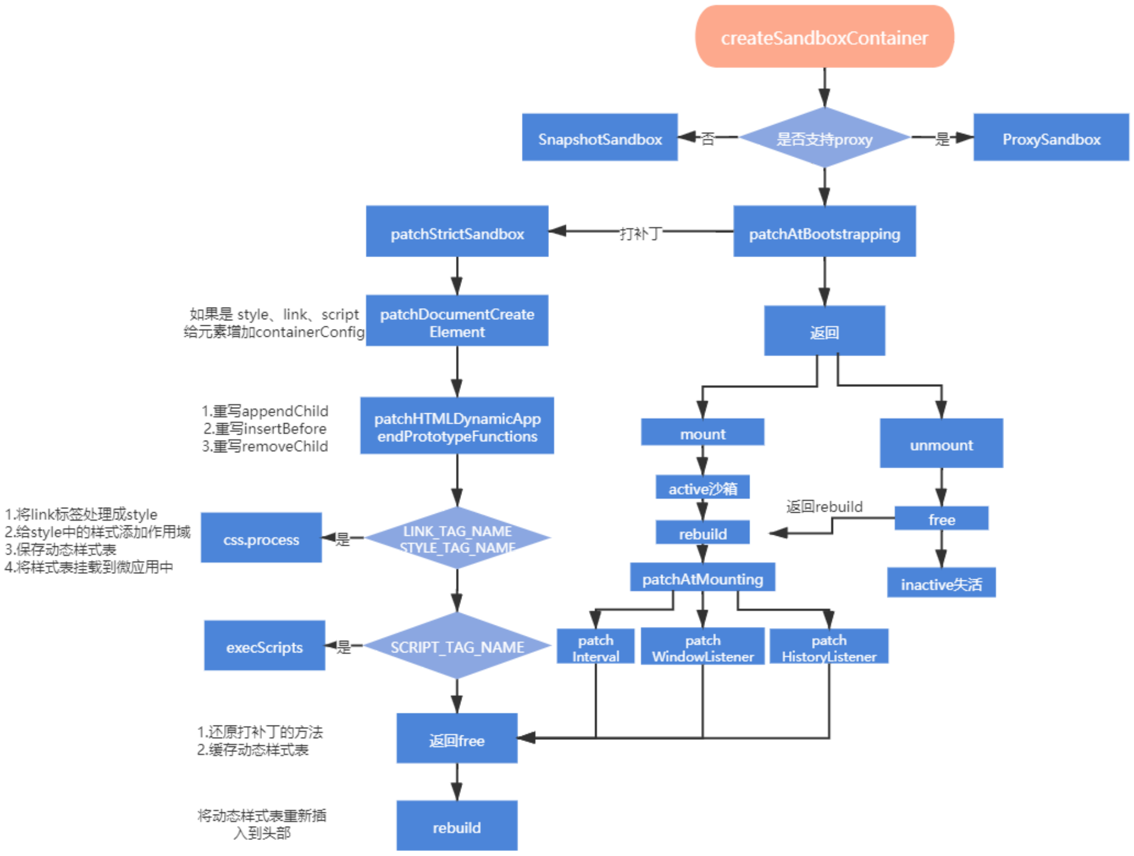
```
export async function loadApp<T extends ObjectType>(
  app: LoadableApp<T>,
  configuration: FrameworkConfiguration = {},
  lifeCycles?: FrameworkLifeCycles<T>,
): Promise<ParcelConfigObjectGetter> {
  const { entry, name: appName } = app; // 获取要加载的应用和应用的名字
  const appInstanceId = `${appName}_${+new
Date()}_${Math.floor(Math.random() * 1000)}`;

  const markName = `[qiankun] App ${appInstanceId} Loading`;
  if (process.env.NODE_ENV === 'development') {
    performanceMark(markName);
  }

  const { singular = false, sandbox = true, excludeAssetFilter,
...importEntryOpts } = configuration;

  // get the entry html content and script executor  通过路径获取html
  // template 就是注释掉脚本后的html
  // execScripts 要执行的脚本，  可以增添沙箱
  // assetPublicPath 子应用的publicPath
  const { template, execScripts, assetPublicPath } = await
importEntry(entry, importEntryOpts);

  // as single-spa load and bootstrap new app parallel with other
apps unmounting
  // (see https://github.com/CanopyTax/single-
spa/blob/master/src/navigation/reroute.js#L74)
  // we need wait to load the app until all apps are finishing
unmount in singular mode
  if (await validateSingularMode(singular, app)) {  // 等待所有应用卸载
完毕 在进行挂载
    await (prevAppUnmountedDeferred &&
prevAppUnmountedDeferred.promise);
  }
  // 将模板内容，外面包裹一个div
  const appContent = getDefaultTplWrapper(appInstanceId, appName)
(template);

  // 沙箱处理样式隔离 shadowdom
  const strictStyleIsolation = typeof sandbox === 'object' &&
!!sandbox.strictStyleIsolation;
  // 是否启用作用域css
  const scopedCSS = isEnableScopedCSS(sandbox);
  // 创建shadowDOM或者作用域样式
  let initialAppWrapperElement: HTMLElement | null = createElement(
    appContent,
    strictStyleIsolation,
    scopedCSS,
    appName,
  );
  const initialContainer = 'container' in app ? app.container :
undefined; // 初始化容器
  const legacyRender = 'render' in app ? app.render : undefined; //
遗留的render方法

  const render = getRender(appName, appContent, legacyRender); // 返回

  // 第一次加载设置应用可见区域 dom 结构
  // 确保每次应用加载前容器 dom 结构已经设置完毕  渲染
  render({ element: initialAppWrapperElement, loading: true,
container: initialContainer }, 'loading');

  // 获取包裹容器，可能是shadowDOM的容器
  const initialAppWrapperGetter = getAppWrapperGetter(
    appName,
    appInstanceId,
    !!legacyRender,
    strictStyleIsolation,
    scopedCSS,
    () => initialAppWrapperElement,
  );

  let global = window;
  let mountSandbox = () => Promise.resolve();
  let unmountSandbox = () => Promise.resolve();
  const useLooseSandbox = typeof sandbox === 'object' &&
!!sandbox.loose;
  let sandboxContainer;
  if (sandbox) { // 使用沙箱
    sandboxContainer = createSandboxContainer(
      appName,
      // FIXME should use a strict sandbox logic while remount, see
https://github.com/umijs/qiankun/issues/518
      initialAppWrapperGetter,
      scopedCSS,
      useLooseSandbox,
      excludeAssetFilter,
    );
    // 用沙箱的代理对象作为接下来使用的全局对象
    global = sandboxContainer.instance.proxy as typeof window;
    mountSandbox = sandboxContainer.mount; // 将沙箱mount和unmount保存
起来
    unmountSandbox = sandboxContainer.unmount;
  }

  // 给quankun的钩子上增加属性
  const {
    beforeUnmount = [],
    afterUnmount = [],
    afterMount = [],
    beforeMount = [],
    beforeLoad = [],
  } = mergeWith({}, getAddOns(global, assetPublicPath), lifeCycles,
(v1, v2) => concat(v1 ?? [], v2 ?? []));
  // 执行beforeLoad方法 转化成链
  await execHooksChain(toArray(beforeLoad), app, global);

  // get the lifecycle hooks from module exports
  const scriptExports: any = await execScripts(global, sandbox &&
!useLooseSandbox); // 在沙箱中执行脚本指定上下文
  const { bootstrap, mount, unmount, update } =
getLifecyclesFromExports( // 获得子应用的生命周期
    scriptExports,
    appName,
    global,
    sandboxContainer?.instance?.latestSetProp,
  );

  // 绑定事件监听功能
  const { onGlobalStateChange, setGlobalState, offGlobalStateChange
}: Record<string, CallableFunction> =
    getMicroAppStateActions(appInstanceId);

  // FIXME temporary way
  const syncAppWrapperElement2Sandbox = (element: HTMLElement | null)
=> (initialAppWrapperElement = element);

  const parcelConfigGetter: ParcelConfigObjectGetter =
(remountContainer = initialContainer) => {
    let appWrapperElement: HTMLElement | null =
initialAppWrapperElement;
    const appWrapperGetter = getAppWrapperGetter(
      appName,
      appInstanceId,
      !!legacyRender,
      strictStyleIsolation,
      scopedCSS,
      () => appWrapperElement,
    );

    const parcelConfig: ParcelConfigObject = {
      name: appInstanceId,
      bootstrap,
      mount: [
        async () => {
          if (process.env.NODE_ENV === 'development') {
            const marks = performanceGetEntriesByName(markName,
'mark');
            // mark length is zero means the app is remounting
            if (marks && !marks.length) {
              performanceMark(markName);
            }
          }
        },
        async () => { // 单例模式只能挂在一个应用
          if ((await validateSingularMode(singular, app)) &&
prevAppUnmountedDeferred) {
```

```
            return prevAppUnmountedDeferred.promise;
          }

          return undefined;
        },
        // 添加 mount hook, 确保每次应用加载前容器 dom 结构已经设置完毕
        async () => {
          const useNewContainer = remountContainer !==
initialContainer;
          if (useNewContainer || !appWrapperElement) {
            // element will be destroyed after unmounted, we need to
recreate it if it not exist
            // or we try to remount into a new container
            appWrapperElement = createElement(appContent,
strictStyleIsolation, scopedCSS, appName);
            syncAppWrapperElement2Sandbox(appWrapperElement);
          }

          render({ element: appWrapperElement, loading: true,
container: remountContainer }, 'mounting');
        },
        mountSandbox, // 挂载沙箱
        // exec the chain after rendering to keep the behavior with
beforeLoad
        async () => execHooksChain(toArray(beforeMount), app,
global), // 执行beforeMount链式调用
        async (props) => mount({ ...props, container:
appWrapperGetter(), setGlobalState, onGlobalStateChange }),
        // finish loading after app mounted
        async () => render({ element: appWrapperElement, loading:
false, container: remountContainer }, 'mounted'), // 挂载完毕 loading为
false
        async () => execHooksChain(toArray(afterMount), app, global),
// 执行afterMount
        // initialize the unmount defer after app mounted and resolve
the defer after it unmounted
        async () => {
          if (await validateSingularMode(singular, app)) { // 单例的话
添加一个promise
            prevAppUnmountedDeferred = new Deferred<void>();
          }
        },
        async () => {
          if (process.env.NODE_ENV === 'development') {
            const measureName = `[qiankun] App ${appInstanceId}
Loading Consuming`;
            performanceMeasure(measureName, markName);
          }
        },
      ],
      unmount: [
        async () => execHooksChain(toArray(beforeUnmount), app, global), // 执行bueforeUnmount
        async (props) => unmount({ ...props, container: appWrapperGetter() }), // 调用unmount
        unmountSandbox, // 卸载沙箱
        async () => execHooksChain(toArray(afterUnmount), app, global), // 执行afterUnmount
        async () => {
          render({ element: null, loading: false, container: remountContainer }, 'unmounted'); // 渲染卸载完毕
          offGlobalStateChange(appInstanceId); // 关闭全局监听事件
          // for gc
          appWrapperElement = null;
          syncAppWrapperElement2Sandbox(appWrapperElement);
        },
        async () => {
          if ((await validateSingularMode(singular, app)) && prevAppUnmountedDeferred) {
            prevAppUnmountedDeferred.resolve(); // 单例卸载后，可以挂另一个
          }
        },
      ],
    };
    if (typeof update === 'function') {
      parcelConfig.update = update; // 添加update方法
    }
    return parcelConfig;
  };
  return parcelConfigGetter;
}
```

- 通过`importEntry`方法拉取子应用
- 在拉取的模板外面包一层`div`,增加`css`样式隔离`shadowdom`、`scopedCSS`
- 将模板进行挂载
- 创建`js`沙箱 ,获得沙箱开启和沙箱关闭方法
- 合并出`beforeUnmount`、`afterUnmount`、`afterMount`、`beforeMount`、`beforeLoad`方法。增加`qiankun`标识
- 依次调用`beforeLoad`方法
- 在沙箱中执行脚本， 获取子应用的生命周期`bootstrap`、`mount`、`unmount`、`update`
- 格式化子应用的`mount`方法和`unmount`方法。

- 在mount执行前挂载沙箱、依次执行`beforeMount`，之后调用mount方法，将全局通信方法传入。mount方法执行完毕后执行`afterMount`
- unmount方法会优先执行`beforeUnmount`钩子，之后开始卸载
- 增添一个`update`方法

5.`createSandbox`实现



createSandboxContainer
是否支持proxy
否
SnapshotSandbox
是
ProxySandbox
patchAtBootstrapping
打补丁
patchStrictSandbox
如果是 style、link、script
给元素增加containerConfig
patchDocumentCreate Element
1.重写appendChild
2.重写insertBefore
3.重写removeChild
patchHTMLDynamicAppendPrototypeFunctions
1.将link标签处理成style
2.给style中的样式添加作用域
3.保存动态样式表
4.将样式表挂载到微应用中
css.process
是
LINK_TAG_NAME
STYLE_TAG_NAME
execScripts
是
SCRIPT_TAG_NAME
1.还原打补丁的方法
2.缓存动态样式表
返回free
将动态样式表重新插入到头部
rebuild
返回
mount
active沙箱
rebuild
patchAtMounting
patch Interval
patch WindowListener
patch HistoryListener
unmount
返回rebuild
free
inactive失活