KOCA 微前端指南

KOCA 微前端介绍

微前端是一种多个团队通过独立发布功能的方式来共同构建现代化 web 应用的技术手段及方法策略。KOCA 微前端使用 qiankun 微前端框架。

微前端架构具备以下几个核心价值：

- 技术栈无关
  主框架不限制接入应用的技术栈，微应用具备完全自主权。

- 独立开发、独立部署
  微应用仓库独立，前后端可独立开发，部署完成后主框架自动完成同步更新。

- 增量升级
  在面对各种复杂场景时，我们通常很难对一个已经存在的系统做全量的技术栈升级或重构，而微前端是一种非常好的实施渐进式重构的手段和策略。

创建微前端项目

使用最新版的 koca-cli 创建项目，选择微前端框架：

微前端目录结构

以微应用 admin-basic(**app_name=admin-basic**)为例：

```
micro-front                          微前端工程目录 ${workspaceFolder}
    ├─ package.json                  项目依赖模块，构建命令：serve，serve.main 等
    ├─ vue.config.js                 构建 VUE 单应用配置
    ├─ vue.main.config.js            构建 VUE 主应用配置
    ├─ vue.app.config.js             构建 VUE 微应用配置
    ├─ package.json                  项目依赖模块，构建脚本，serve.main，build.main
    ├─ public                        主应用 public 目录
    │   ├─ admin-basic               *** 微应用 public 目录为 public/$app_name/
    │   │   ├─ index.html            *** 微应用入口 index 文件
```

```
    │   │   └── app_config.js          *** 微应用配置文件
    │   ├── micro_apps.js              $$$ 微应用定义文件
    │   └── index.html                 主应用入口 index 文件
    ├── build                          项目构建的脚本，配置等
    │   ├── externals.js               需要外部化的库,按需要增加
    │   ├── external-details.js        需要外部化的库的详细描述,按需要增加
    │   └── utils.js                   项目构建使用的函数定义
    ├── src                            源文件目录
    │   ├── main.ts                    主应用入口 ts
    │   ├── microapp.ts                *** 微应用入口 ts，即微应用的 main.ts
    │   ├── router/projects.ts         主应用包含的页面，自动生成
    │   ├── router/admin-basic.ts      *** 微应用包含的页面，自动生成
    │   ├── config/lazy-libs.ts        延迟加载的依赖库
```

说明：新增微应用时，请仿照 admin-basic 在 package.json 中增加构建命令，
增加 public/app_name 目录，在 public/micro_apps.js 中定义应用。

微前端编译打包

编译打包脚本定义在 package.json 中，开发人员根据需要以 admin-basic 为例增加构建脚本。如果主应用和微应用部署在一起，编译打包时可以不指定 VUE_APP_IP 和 VUE_APP_PORT，如下图的 build.studio-basic。

- serve.main -- 开发 serve 主应用
- build.main – 编译打包主应用
- serve.app-name -- 开发 serve 微应用
- build.app-name – 编译打包微应用
- approutes – 生成微应用路由

```
"approutes": "node build/projects_routes.js --app",
"approutes.admin-basic": "node build/projects_routes.js --app admin-basic /views/",

"serve.main": "cross-env VUE_APP_SPA=qiankun VUE_CLI_SERVICE_CONFIG_PATH=./vue.main.config.js vue-cli-service
"build.main": "cross-env VUE_APP_SPA=qiankun VUE_CLI_SERVICE_CONFIG_PATH=./vue.main.config.js vue-cli-service

"serve.admin-basic": "cross-env VUE_APP_NAME=admin-basic VUE_APP_IP=localhost VUE_APP_PORT=7001 VUE_CLI_SERVI
"build.admin-basic": "cross-env VUE_APP_NAME=admin-basic VUE_APP_PORT=7001 VUE_CLI_SERVICE_CONFIG_PATH=./vue.

"serve.studio-basic": "cross-env VUE_APP_NAME=studio-basic VUE_APP_IP=localhost VUE_APP_PORT=7002 VUE_CLI_SER
"build.studio-basic": "cross-env VUE_APP_NAME=studio-basic VUE_CLI_SERVICE_CONFIG_PATH=./vue.app.config.js vu
```

微应用路由生成：vue.main.config.js 会生成微应用路由文件./src/router/app_name.ts，用户也可以手动执行命令生成路由：yarn approutes app_name /views/。

开发注意事项

- **微应用页面路径和名称命名规范**：src/projects/**app-name/views**/**/FileName.vue，FileName 和组件名称保持一致，文件路径和访问的 URL 保持一致。如：src/projects/admin-basic/views/user/UserInfoManager.vue。

- 路由的懒加载是按源代码目录树处理的，菜单树尽量和源代码目录树保持一致用户体验更好。
- httpClient 通过 this.$http 访问。如：

```
this.tableProp.loading = true;
this.$http.post(this.apiList.queryRoleInfoSimple, {}).then((res) => {
  this.tableProp.loading = false;
```

- Vuex 由主应用提供，微应用通过 this.$app, this.$user, this.$dict 等访问，如：

```
if (item) {
  item = this.$dict.translate("ID_TYPE", item);
}
```

- 微应用不允许引入主应用的 store，如下面的例子微应用中不允许使用。

```
// 微应用不允许
import { app$ } from "szkingdom.yf.koca-template/lib/store";
```

- 当不能通过 this 访问主应用 Vue 的绑定的对象时，可以类似这样使用：

```
244   const httpClient = Vue.prototype.$http;
```

- 微应用可以定义自己需要的全局变量，但是微应用之间注意名称不要冲突，如 public/admin-basic/app_config.js：

```
1  // ---- 微应用配置文件，命名 APP_NAME_CONIFG，如
2  window.ADMIN_BASIC_CONIFG = {
3    SETTING: {
4      AA: "aa",
5      BB: "bb",
6    }
7  };
8
```

- 微应用不应该引入 kui 或 element-ui 来个性化，容易造成重复包含、打包体积过大、内存占用增加问题。Kui 组件如果不满足需要可以给前端组提需求增加或修改相关组件。
- 微应用修改全局样式应该是少量的，尤其不能再次全量引入 kui/element-ui 的样式！各个微应用的界面主题风格应该和主应用保持一致，由主应用个性化主题。如果确实有需要覆盖全局样式，可以放在 src/assets/scss/app_reset.scss 中，由微应用入口文件引入。

```
TS microapp.ts ×
11
12  import "./assets/scss/app_reset.scss";
13
```

框架升级到微前端框架

1. 删除 szkingdom.yf.template 依赖库（npm uninstall szkingdom.yf.template）。

2. 添加 szkingdom.yf.micro-template 依赖库（npm install szkingdom.yf.micro-template）。
3. 以 node_modules/szkingdom.yf.micro-template/package.json 为基准，合并 package.json。
4. 修改 koca-package.json，原型改为微前端框架，原型版本和上一部安装的版本一致（package.json 中查看安装的版本号）。

5. 将这些目录或文件复制到项目目录：

node_modules/szkingdom.yf.micro-template	前端项目目录
public/admin-basic	public/admin-basic
tsconfig.json	tsconfig.json
build	build
src/microapp.ts	src/microapp.ts
src/router/static.ts	src/router/static.ts
vue.config.js	vue.config.js
vue.app.config.js	vue.app.config.js
vue.main.config.js	vue.main.config.js

修改代码满足微应用

开发人员根据开发注意事项修改代码。

url.config.js 修改举例：

1. url.config.js 定义无前缀的 api 接口

```
export const ApiList = {
  // user
  queryUserDetail: "user/detail", // 人员信息查询（详情）
  queryUserInfoSimple: "user/query-simple",  // 人员信息查询（简约）
  addUserInfo: "user/add", // 人员信息新增
  updateUserInfo: "user/update", // 人员信息修改
  saveUserRole: "user/role/save", // 人员角色保存
  saveUserOrg: "user/org/save", // 人员机构保存
  userFreezeUrl: "user/freeze", // 人员冻结
  userUnfreezeUrl: "user/unfreeze", // 人员解冻
```

```
  userCancelUrl: "user/cancel",  // 人员注销
};
```

2. vue 文件获取有前缀的 apiList

```
import { Component, Vue, Watch } from "vue-property-decorator";

import { ApiList } from "./url.config";
import { getApiList } from "szkingdom.yf.koca-template/lib/utils/routes";

@Component({
  name: "UserInfoManager",
})
export default class UserInfoManager extends Vue {
  // computed
  get apiList() {
    return getApiList(this.$app.apiPrefix.USER_URL_PREFIX, ApiList);
  }
}
```

3. template 使用有前缀的 apiList

```
<kui-page
  slot="aside"
  ref="page1"
  title="$t[dict.xitong_zidian]"
  :is-block="false"
  remote-pagi
  :query-url="apiList.queryDictInfoPage"
  :add-url="apiList.addDictInfo"
  :upd-url="apiList.updateDictInfo"
  :del-url="apiList.deleteDictInfo"
  :form-prop="formProp"
```

微应用定义自己的 store 举例

1. src/store/modules/legacy.ts

```
import { VuexModule, Module, Mutation, Action } from "vuex-module-
decorators";

export interface ILegacyState {
  publicPath: string;
}

@Module({ name: "legacy", namespaced: true })
export class LegacyModule extends VuexModule implements ILegacyState {
  publicPath = "";

  @Mutation
  setPublicPath(entry: string) {
    this.publicPath = entry;
  }
}
```

2. src/store/index.ts

```
import Vuex from "vuex";
import { getModule } from "vuex-module-decorators";
import { LegacyModule, ILegacyState } from "./modules/legacy";

export interface IRootState {
  app: ILegacyState;
}

const store = new Vuex.Store<IRootState>({
  modules: {
    legacy: LegacyModule,
  },
});

export const legacy$ = getModule(LegacyModule, store);

export default store;
```

微应用定义举例 public/micro_apps.js

```
// 微应用定义，为空数组表示没有定义
var KOCA_MICRO_APPS = [
  {
    sysId: 1,              // 系统 id，匹配字典用
    name: "admin-basic",
    entry: "http://localhost:7001/admin-basic/",
    container: "#admin-basic-container",
    prefixPath: "/admin-basic",
  },
  {
    sysId: 1,              // 系统 id，匹配字典用
    name: "studio-basic",
    entry: "http://localhost:7002/studio-basic/",
    container: "#studio-basic-container",
    prefixPath: "/studio-basic",
  }
];
```

常见问题处理

问题：页面正常，没有微应用加载请求

如果调试工具检查网路请求发现没有微应用加载请求，问题应该是主应用和微应用是同一前端工程，主应用打包时没有排除掉相应的目录。修改主应用打包配置 vue.main.config.js，排除微应用的目录就可以了。如下图中，admin-basic, studio-basic 这两个目录都作为微应用部署，主应用打包配置文件里就需要配置这两个目录。

```
57
58  // 生成projects下模块的路由映射
59  // 配置主应用不包含哪些模块
60  const excludes = [
61    "admin-basic",
62    "studio-basic",
63  ];
64
65  // views 页面目录, 如：/pages/, /views/, ""
66  const views = "/views/";
67  generate_projects_routes(views, excludes);
68
```

问题：没有加载微应用页面

1．目前有三个 vue.config，其中 vue.config.js 是没有微应用改造的单应用配置，用 yarn serve 启动。Vue.mian.js 是微前端构建主应用配置，用 yarn serve.main 启动。Vue.app.config.js 是微前端构建微应用的配置，用 yarn serve.app_name 启动。开发人员需要**确保执行了正确的启动脚本**。

2．vue cli 如果发现端口好被占用会使用下一个可用的端口，开发人员也需要检查监听端口是否和配置的端口一致。

3．确保微应用的路由生成正确并和菜单的路径一致。vue.app.config.里有生成路由定义文件：src/router/app_name.ts。如果页面路径不符合命名规范，**没有 views 目录，会导致生成空的数组**。

```
33  // views 页面目录, 如：/pages/, /views/, ""
34  const views = "/views/";
35  generate_app_routes(app_name, views);
```

微应用入口文件 src/microapp.ts 里有打印出路由的代码，也可以放开调试。

```
19  const routes = require(`@/router/${process.env.VUE_APP_NAME}`).app_routes;
20  routes.push({path: "*", component: NotFoundPage});
21  // 查看生成的路由对不对
22  // console.log("routes", routes);
```

问题：微应用首页没有编译

1．检查构建脚本和微应用定义：**文档中 admin-basic 是配置举例的微应用名称。自己增加微应用配置时请检查是否有哪里没有改到。**如增加名称为 legacy 的微应用构建脚本，这里 build.legacy 没有指定 IP 和 PORT，表示微应用打包后和主应用部署在一起：

```
    "serve.legacy": "cross-env VUE_APP_NAME=legacy VUE_APP_IP=localhost
VUE_APP_PORT=7001 VUE_CLI_SERVICE_CONFIG_PATH=vue.app.config.js vue-
cli-service serve",
    "build.legacy": "cross-env VUE_APP_NAME=legacy
VUE_CLI_SERVICE_CONFIG_PATH=vue.app.config.js vue-cli-service build --
report",
```

相应的微应用定义文件(public/micro_apps.js)要和构建脚本的名称、IP 和 PORT 保存一致。

```
// 微应用定义，为空数组表示没有定义
var KOCA_MICRO_APPS = [
  {
    name: "legacy",
    // entry: "http://localhost:7001/legacy/",  // 开发配置
    entry: "/legacy/",  // 部署配置
    container: "#legacy-container",
    prefixPath: "/legacy",
  }
];
```

2．注意微应用的 entry 入口地址带有后缀 **app_name/**。单独访问微应用排查问题需要用有后缀的 entry 地址，例如：http://localhost:7001/legacy/。

问题：微应用页面 keep-alive 有问题

Keep-alive 是根据组件定义的名称来缓存的。由于 koca 菜单定义没有组件名称配置，主应用没法知道路由的组件名称，我们需要**约定俗成，按照命名规范命名路径和文件名，代码才能取到正确的组件名称**。

问题：微应用没有按菜单目录延迟加载

应该是微应用配置文件 vue.app.config.js 不是最新的，**注意不能配置 transpileDependencies**。请比较是否如下：

```
/* eslint-disable */
// @ts-check
const path = require("path");
const PreloadPlugin = require("@vue/preload-webpack-plugin");
const utils = require("./build/utils");
const externals = require("./build/externals");
const { generate_app_routes } = require("./build/projects_routes");

const app_name = process.env.VUE_APP_NAME;
const outputDir = path.resolve(__dirname, "dist/" + app_name);
const port = process.env.VUE_APP_PORT;
const use_cdn = false;
const is_app = true;
const is_prod = process.env.NODE_ENV === "production";
// 默认不使用 swc 编译，需要时放开
const use_swc = false;

// **** 微应用需要有明确的地址，不然主用加载资源是会 404:
// **** 如果微应用和主应用部署在同一个 nginx 下的同一个 IP 和端口，
// publicPath 可以配置为`/${app_name}`
// const publicPath = `//localhost:${port}/${app_name}`;
const publicPath = utils.build_publicPath();

const exclude_libs = [
  "vue",
  "vue-router",
  "vuex",
  "vue-i18n",
  "element-ui",
```

```
  "axios",
];

// views 页面目录，如：/pages/, /views/, ""
const views = "/views/";
generate_app_routes(app_name, views);

// 拷贝 venders 依赖库
utils.copy_venders(`public/${app_name}`, exclude_libs);

module.exports = {
  pages: {
    index: {
      entry: "src/microapp.ts",
      template: `public/${app_name}/index.html`,
    }
  },
  devServer: {
    port,
    headers: {
      "Access-Control-Allow-Origin": "*",
    },
  },
  publicPath,
  outputDir,
  lintOnSave: true,
  runtimeCompiler: false,
  productionSourceMap: false,
  css: {
    extract: false,
    sourceMap: !is_prod,
  },

  // 自定义 webpack 配置
  configureWebpack: {
    externals,
    // qiankun 要求把子应用打包成 umd 库格式
    output: {
      library: `${app_name}-[name]`,
      libraryTarget: "umd",
      jsonpFunction: `webpackJsonp_${app_name}`,
    },
  },

  chainWebpack: config => {
    config.plugins.delete("prefetch-index");

    utils.config_alias(config);
    utils.config_html(config, use_cdn, publicPath, "html-index", exclude_libs);
    utils.config_copy(config, use_cdn, outputDir, is_app, exclude_libs);
    utils.config_zip(config, is_prod);

    // 目前 swc 不支持 vue 文件，所以 vue 文件调试有问题，VUE 文件要把 js，ts 独立成一个文件就没问
题。
    // 如：<script lang="ts" src="./ElementPage.ts"></script>
    // 或：<script lang="js" src="./ElementPage.js"></script>
    utils.config_swc(config, use_swc);
  }
};
```

另外微服务入口文件 src/microapp.ts 文件里 const routes 必须这样写：

```
const routes = require(`@/router/${process.env.VUE_APP_NAME}`).app_routes;
```

问题：内存占用过高

1．首先要明确微前端应该是**主重微轻。**如果微应用再加载一遍 element-ui 和 kui 必然导致加载完成后内存开销很大，所以不建议微应用对 kui/element-ui 或主题做个性化。个性化主题，个性化组件库，加载组件库都应该放到主应用。

2．公共静态资源 cache：除了业务代码以外，前端还会有一些公共静态资源，例如 Vue 的 is 文件、ElementUI 的资源文件、 Echart 资源文件，以及一些图片文件，第三方 JS 库，UI 库等。 对于这些文件，是所有工程所共享的，假如这些文件分散在各个工程里，既没必要，也容易导致不 同工程依赖文件不统一。另外也会重复加载这些文件，浪费网络带宽和静态服务器存储空间，没 有意义。因此，设一个公共静态路径，通过配置，当需要加载这些资源时，直接去指定 nginx 静态服务器里加载，并做长时间 cache，可以提高访问效率和性能。 微应用之间用到相同的第三方 js 库，可以将第三方库 CDN 化延迟加载，最好的是使用的时侯才加载。具体技术细节和应用举例请参考 Confluence 分享：http://192.168.0.220:8090/pages/viewpage.action?pageId=27568460

3．使用 loadsh 时注意不要全量引入。

关于 lodash 用法，只有最后的两种方法才能起到减小包体积的效果：

```
xx: import * as _ from 'lodash';          # 全部引入，错误用法
xx: import _ from 'lodash';               # 全部引入，错误用法
xx: import { has } from 'lodash';         # 全部引入，错误用法

ok: import get from 'lodash.get';         # 需要安装 lodash.get，不建议。
ok: import _find from 'lodash/find';      # 只使用 find，建议这种
```

统一起起见，建议全部替换为 lodash-es （最好用这种！！！）

```
yarn add lodash-es && yarn add -D @types/lodash-es
```

用法：

```
import { random, get, set, find, has } from "lodash-es";
```

或：

```
import {
    random as _random,
    get as _get,
    set as _set,
    find as _find,
    has as _has
} from "lodash-es";
```

或：

```
import random from "lodash-es/random";
```

或：

```
import _random from "lodash-es/random";
```

微应用打包结果检查

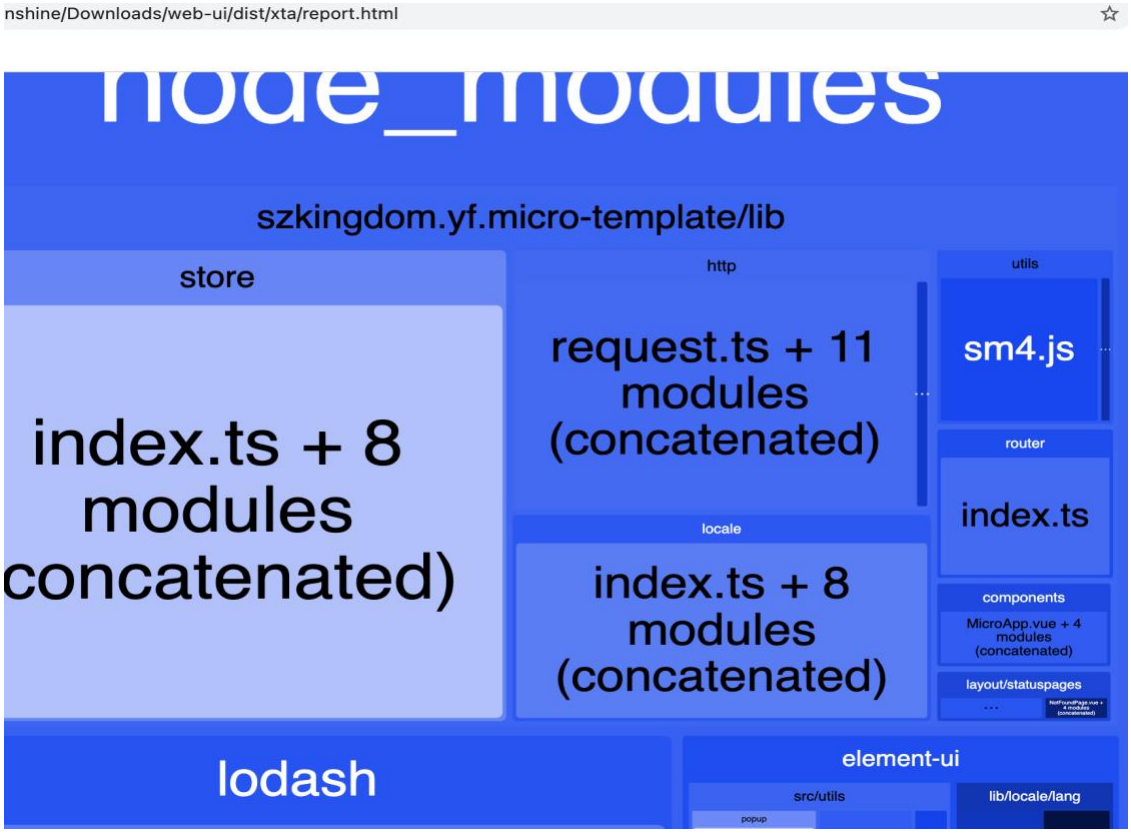
（1）微应用打包是按目录树分块的，需要观察打包的 js 文件名是否按目录树分块。（2）微应用不能引入主应用的 Vuex 定义的变量，观察打包报告文件 dist/APP_NAME/report.html 可以确定有没有引入了主应用的$app, $user，$dict 等。（3）观察引入第三依赖库的大小，看看是否有可以优化的地方，比如 lodash 是不是全量引入了，是不是引入了 moment.js 等。

一、多数 js 文件应该按 src/router/APP_NAME.ts 文件里指定的 webpackChunkName 命名。

```
ıt/maintain/Maintain", component: () => import(/* webpackChunkName: "xta_account_maintain" */
```

```
-zsh                                                                    ⌥⌘1
-zsh ⌘1
-zsh ⌘2
-zsh ⌘3

File                                                    Size        Gzipped

dist/xta/venders/tippy.js/6.2.7/tippy-bundle.umd.min.js     23.51 KiB   7.88 KiB
dist/xta/js/index.b52f5cbc.js                               120.83 KiB  29.52 KiB
dist/xta/js/chunk-vendors.a56c5686.js                       118.35 KiB  41.11 KiB
dist/xta/js/xta_account_adjust~xta_account_check~xta_accoun 100.52 KiB  28.96 KiB
t_maintain~xta_account_manage_apply~xta_account_manag~418fb
6e1.d6d5d7ff.js
dist/xta/js/chunk-0d823a87.f8a1d80e.js                      93.78 KiB   28.33 KiB
dist/xta/js/chunk-7bf587ea.278c4771.js                      90.47 KiB   27.38 KiB
dist/xta/js/xta_account_manage_apply_update_invAppAcct.2ee1 82.49 KiB   14.97 KiB
378e.js
dist/xta/js/xta_account_manage_apply.8c754fa9.js            80.07 KiB   17.27 KiB
dist/xta/js/xta_account_check.5ab3bc2a.js                   78.19 KiB   16.97 KiB
dist/xta/js/xta_account_adjust.447a00e0.js                  76.77 KiB   16.43 KiB
dist/xta/js/xta_account_manage_apply_update.0ed57b6a.js     69.26 KiB   14.81 KiB
dist/xta/js/xta_account_adjust~xta_account_check~xta_accoun 52.06 KiB   9.91 KiB
t_manage_apply~xta_account_manage_apply_update~xta_ac~210be
e4c.bb30ff34.js
dist/xta/js/xta_account_manage_apply_update_invAcctCrs.31fd 47.80 KiB   9.41 KiB
911f.js
dist/xta/js/xta_account_manage_apply_update_invAppAcctAnti. 39.86 KiB   10.69 KiB
11458bcb.js
dist/xta/venders/numeral/2.0.6/numeral.js                   32.81 KiB   6.81 KiB
dist/xta/js/xta_account_adjust~xta_account_check~xta_accoun 19.45 KiB   6.51 KiB
t_maintain~xta_account_manage_apply~xta_account_manag~b7e25
656.1e324fc5.js
dist/xta/js/xta_account_maintain.121d584b.js                16.32 KiB   4.36 KiB
dist/xta/js/chunk-48c54546.9a81d4e0.js                      16.28 KiB   10.60 KiB
dist/xta/js/xta_account_manage_apply_update_commonComp.9402 5.82 KiB    2.17 KiB
3fc3.js
dist/xta/js/chunk-020229b6.41a4df3c.js                      2.47 KiB    1.00 KiB
dist/xta/app_config.js                                      0.69 KiB    0.28 KiB

Images and other types of assets omitted.

 DONE  Build complete. The dist/xta directory is ready to be deployed.
 INFO  Check out deployment instructions at https://cli.vuejs.org/guide/deployment.html

✨  Done in 23.21s.
```

二、下图中微应用打包报告包含了 szkngdom.yf.micro-template/lib/store 是不对的：



NGINX 配置举例

```
# mkdir /usr/local/var/www/kace
# cp -R dist/* /usr/local/var/www/kace/
#
# cp docs/kace.nginx.conf /usr/local/etc/nginx/servers/
# rm -r -f /usr/local/var/log/nginx/*
# brew services restart nginx
gzip off;
gzip_static on;

brotli off;
brotli_static on;

server {
    listen 9000;
    server_name localhost;
    charset UTF-8;


    access_log  /usr/local/var/log/nginx/kace.access.log main;
    error_log   /usr/local/var/log/nginx/kace.error.log warn;

    # MockServer 代理所有 mock 请求
    location ^~ /mock/ {
      proxy_pass http://localhost:1088;
      rewrite ^/mock(.*)$ $1;
    }

    # 处理代理/legacy/ajax_fm.do
    location ^~ /legacy/ {
      proxy_pass http://192.168.8.212:8110;
      rewrite ^/legacy(.*)$ $1;
    }

    # API Request 代理所有业务请求, 去掉前缀 api
    location ^~ /api/ {
      proxy_pass http://192.168.8.212:8110;
      rewrite ^/api(.*)$ $1;
    }

    # Web Resources 静态页面
    location / {
        root /usr/local/var/www/kace;
        index index.html index.htm;

        # No Cache 这些静态文件不缓存
        location ~* (index.html|micro_apps.js|app_config.js|config.json)$ {
          add_header Last-Modified $date_gmt;
          add_header Cache-Control 'no-store, no-cache';
          if_modified_since off;
          expires off;
          etag off;
        }

        try_files $uri $uri/ /index.html;
    }
}
```