PO 前端项目交接

交接人： 李顺

对接人：黄德辉

Contents

[一、PO 框架代码交接 3](#_Toc14189079)

[1项目初始化 3](#_Toc14189080)

[2 VS code 配置 3](#_Toc14189081)

[3目录说明 5](#_Toc14189082)

[4 src目录说明 5](#_Toc14189083)

[5 main.js 文件 6](#_Toc14189084)

[6 api 模块 6](#_Toc14189085)

[7 Vuex 模块 7](#_Toc14189086)

[8 Router 模块。 8](#_Toc14189087)

[9 Components 模块 9](#_Toc14189088)

[10 Plugin 模块 9](#_Toc14189089)

[11 Utils/Filters 模块 9](#_Toc14189090)

[12 Utils/Mixins 模块 9](#_Toc14189091)

[13 Directive 模块 9](#_Toc14189092)

[14页面组件模块 10](#_Toc14189093)

[15项目中使用静态变量 11](#_Toc14189094)

[16环境变量 12](#_Toc14189095)

[二、SST 组件库代码交接 13](#_Toc14189096)

[1 组件库实现原理 13](#_Toc14189097)

[2 组件库目录结构 14](#_Toc14189098)

[3 src 目录 14](#_Toc14189099)

[4 packages 目录 14](#_Toc14189100)

[5 views 目录 15](#_Toc14189101)

[6 组件开发流程 15](#_Toc14189102)

[7 组件文档编写流程 16](#_Toc14189103)

[8 打包组件库 17](#_Toc14189104)

[9 其他项目使用组件库 17](#_Toc14189105)

[三、业务交接 19](#_Toc14189106)

[1、Order Request 模块： 19](#_Toc14189107)

[2、Pending Tasks 模块 19](#_Toc14189108)

[3、Uncompleted Purchase 模块 19](#_Toc14189109)

[4、Vendor 模块 19](#_Toc14189110)

[5、 PO Summary 20](#_Toc14189111)

[6、 Assign Package 20](#_Toc14189112)

[7、Pending PO 20](#_Toc14189113)

[8、 Receiver Address 20](#_Toc14189114)

[9、Inquiry Setting 20](#_Toc14189115)

[10、Receiving 21](#_Toc14189116)

[11、 Ready for Pick Up 21](#_Toc14189117)

## 一、PO 框架代码交接

### 1项目初始化

vue create sst-vue-po(项目名)

### 2 VS code 配置

#### 2.1安装 vetur、perttier 、eslint插件

#### 2.2 vs code 项目语法风格配置

##### 2.2.1 根目录下新建 .vscode/settings.json

{

"eslint.alwaysShowStatus": true,

"eslint.autoFixOnSave": true,

"eslint.validate": [

"javascript",

"javascriptreact",

{

"language": "vue",

"autoFix": true

}

],

"files.autoSave": "onFocusChange"

}

##### 2.2.2 项目代码风格配置（参照.eslintrc: <https://eslint.vuejs.org/rules/>. prettierrc: <https://prettier.io/docs/en/configuration.html> ）

参考： .eslintrc

module.exports = {

root: true,

env: {

node: true

},

extends: ['plugin:vue/essential', 'plugin:vue/strongly-recommended','plugin:vue/recommended', '@vue/prettier'],

rules: {

'no-console': process.env.NODE\_ENV === 'production' ? 'error' : 'off',

'no-debugger': process.env.NODE\_ENV === 'production' ? 'error' : 'off',

"vue/html-self-closing": ["error", {

"html": {

"void": "always",

"normal": "always",

"component": "always"

},

"svg": "always",

"math": "always"

}]

},

parserOptions: {

parser: 'babel-eslint'

}

}

.prettierrc

{

"semi": false,

"singleQuote": true

}

##### 2.2.3项目 icon 更改， vue.config.js 添加

pwa需要安装依赖： vue add @vue/pwa node-sass sass-loader

pwa: {

iconPaths: {

favicon32: 'favicon.ico'

}

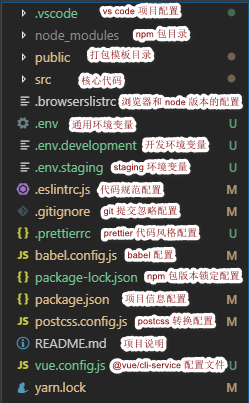
}

更多配置参考 <https://cli.vuejs.org/zh/config/>

### 3目录说明

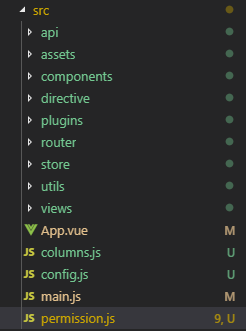
项目目录由 vue-cli 脚手架生成，在开发前应参考上面提到的 eslint 和 vscode 的配置规范开发。

* src 为主要的开发目录
* public 目录存放的是打包时用到的 .html 模板和项目中用的静态文件。
* .env[.xxx] 文件特定的环境文件， 存放对应环境存放的变量。
* vue.config.js 为 @vue/vue-service 预设配置。



### 4 src目录说明

src目录为源码存放目录，以 main.js 为入口文件， 按照使用场景分别包含： api模块、 assets 静态文件模块、公共组件模块、公共指令模块、公共插件模块(如 element-ui)、路由模块、vuex 模块、utils 模块、页面组件模块、公司信息config 模块、路由权限permission.js模块、表格列配置模块以及一个页面主入口组件。



### 5 main.js 文件

该文件为项目的入口文件，负责引用上述模块，主要逻辑是生成一个 Vue 实例，并挂载到 #app DOM 节点上。（建议该文件只做引用和实例生成，不编写任何逻辑代码）

### 6 api 模块

api 模块基于 axios 工具，封装了一套 async/await 的接口模式。底层实现文件为 utils/http.js。应用层按照业务模块分别封装在 api目录下。如 orderRequest 模块对用的文件就是 api/PoOrderRequest.js 文件。每一个接口使用 export const xxx 单独向外暴露。如下：

export const addImageToRequestItem = (args, header) => {

return http.post(`/admin/po/addImageToRequestItem`, args, true, header)

}

整个 api 模块通过 api/index.js 向外暴露出所有的接口, 并且用 $api 这个变量挂载到 vue 实例上。

import \* as common from './PoCommon'

const api = {

common

}

Object.defineProperty(api, 'install', {

writable: false,

enumerable: false,

configurable: false,

value: (Vue, Option) => {

Vue.$api = api

Vue.prototype.$api = api

}

})

export default api

当在业务代码中需要调用接口时使用async/ await方式：

async getReadyLinkPoBox() {

let { error, message, data } = await this.$api.box.findReadyLinkPoBox()

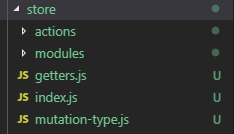
if (error) return this.$message.error(message)

this.readyLinkPoBox = data

},

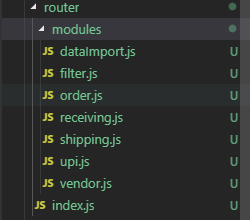
### 7 Vuex 模块

Vuex 模块放置在 store 目录下，包含5个文件和目录，同 api 模块相似， vuex 同样遵循业务划分不同的模块，module目录下为各个模块的 state、mutations、actions的定义，actions 目录下是各个模块actions的具体实现。getters.js 文件中定义了一些全局的 getter 属性，如user、sideBar。mutation-type.js 中定义了所有的 mutation 常量。index.js中引入了各个模块和getters 并向外暴露出一个vuex 实例， 最终在 main.js中挂载到 vue 实例上。



### 8 Router 模块。

#### 8.1 router 目录。

由 modules目录 和 index.js 文件构成， 在 modules 目录下按照页面菜单逻辑划分了各个路由模块。

每一个路由模块向外暴露一个路由对象。对象中需要提供包含以下部分

export default {

path: '/poDataImport',

component: Layout,

redirect: '',

name: 'poDataImport',

meta: { title: 'Data Import', icon: 'example' },

children: [

{

path: '',

name: 'dataImport',

component: () => import('@/views/poDataImport/'),

meta: { title: 'PO Data Import' }

}

]

}

所有的路由模块由 index.js 文件引用，存放在 asyncRouterMap 变量, 而 index.js 文件向外暴露一个router 实例，最终挂载到vue 实例上面。需要注意的是在index.js 中， 向外暴露的 router 实例 routes 属性的值并不是 asyncRouterMap 对象，这是因为该对象需要在验证权限之后才会重新添加到 routes 上面。

#### 8.2路由权限控制。

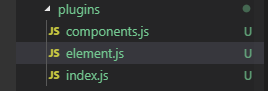
在 src/permission.js 中， 使用router.beforeEach 方法每一次进入页面的时候都根据当前的用户信息来重新触发一次 listPoMenu 接口，生成最新的页面路由表。从而达到每次在admin端重新配置了路由权限，前端页面再次刷新的时候也会同步数据。

### 9 Components 模块

该模块存放项目中有必要作为全局使用的组件，例如 page 组件。每个组件为一个目录, 该目录下为index.vue， 或者每一个组件为单独的 xx-xx.vue。 无论是目录或者文件， 组件的名称应该遵守驼峰或者 xx-xx 的规则。为了防止 mian.js 中单个的引用所有的全局组件会导致 main.js 变得臃肿， 建议全局组件放在 plugin/components.js 中引用。

### 10 Plugin 模块

该模块主要存放需要全局使用的 UI 组件库的挂载逻辑， element.js 中存放了 Element-ui 组件的挂载以及 Element css 重写样式的引入。 而在 components.js 中挂在了一些其他内部使用的公共组件库。



### 11 Utils/Filters 模块

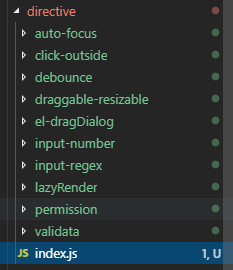
在 main.js 文件中还挂载了一些全局的filter 方法，包含了 formatDate 时间格式处理、currency 金额格式处理等方法，同时 在 filters.js 中用变量 $filters 挂载到了 vue 的原型链上，在编程式使用时可以用 this.$filters.currency(val) 使用

### 12 Utils/Mixins 模块

mixins是为vue全局混入模块。在 uitls/mixins/index.js 中，向全局混入了 print(用来实现打印功能)， twoNum(格式化数字)两个方法和 tableHeight、user 两个computed 属性， 这些方法和属性可以再任何页面组件中不经过声明使用。

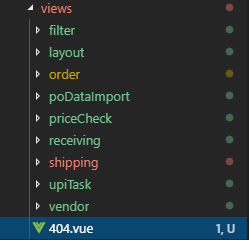
### 13 Directive 模块

该模块按照指令划分目录， 最终由 index.js 文件向外暴露。最终使用v-auto-focus 可以让组件内的 input 组件自动获取焦点。使用 v-click-outside 实现鼠标点击到组件外部时触发自定义方法。v-debounce 用来实现事件的节流、v-draggable-resizable/v-el-dragDialog 实现组件的可拖拽可放大缩小、v-input-number/v-input-regex 实现input 组件的value验证、v-lazy-render 实现表格行的懒加载效果、v-permission 实现按钮的权限控制、v-validate实现数值验证。

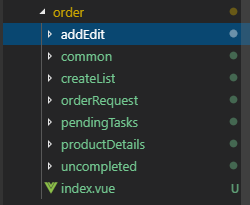


### 14页面组件模块

views 目录存放所有的页面组件。该目录下按照菜单模块划分目录.



以order 目录为例，说明一下每个目录下的结构：

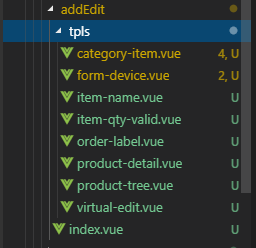


#### 14.1每个目录下应该有一个 index 文件为当前菜单的页面入口。

#### 14.2每一个二级路由单独为一个目录。

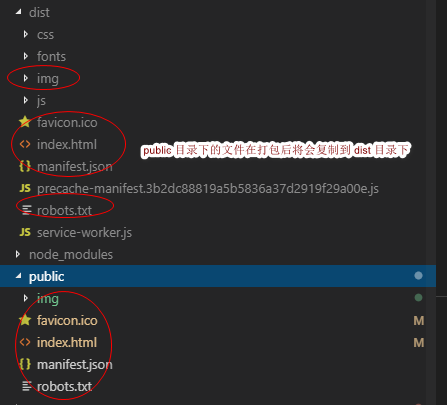
#### 14.3如需要，common 目录存放当前菜单下公用的组件、工具。

#### 14.4单独作为二级路由目录的组件中如需再次抽离组件，可以在目录下建立 tpls 目录，存放仅在该目录下使用到的公共组件和工具



### 15项目中使用静态变量

在[目录说明](#目录说明)中说明了 public 目录中存放了打包模板index.html 文件，此外项目中使用到的静态文件也会放到这个文件当中。 这是因为当使用 npm/yarn run build 的时候除了会向 index.html 文件插入依赖，其他public 文件内的内容会原封不动的复制到打包后的目录。



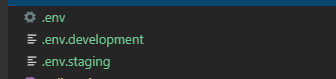
因为这个原因，项目中的静态文件的根目录是 dist/ 也就等同于 public/, 所以项目中引用静态文件可以用 public为根目录

<img src="/img/boxN.png" class="iconX" />

<img src="/img/boxF.png" class="iconX" />

### 16环境变量

项目中需要根据环境来区分api\_url、img\_url、login\_url 等变量的值，可以替换根目录下的 .env[.xx] 文件来制定环境变量



环境文件只包含环境变量的 “键=值”：

VUE\_APP\_API\_ROOT=http://webdev05.skystartrade.com:4009

VUE\_APP\_LOGIN\_URL=http://webdev05.skystartrade.com:9480/a/login

VUE\_APP\_IMAGE\_URL=https://imagetest.skystartrade.com/po/

为了区分环境在使用 yarn/npm 运行脚本的时候应该使用 --mode区分环境。

"scripts": {

"dev": "vue-cli-service serve",

"qa": "vue-cli-service serve --mode qa",

"staging": "vue-cli-service serve --mode staging"

}

#### 17更多参照：

项配置参照 <https://cli.vuejs.org/zh/>

语法规范参照 <https://eslint.vuejs.org/>

<https://cn.vuejs.org/v2/style-guide/>

vue 语法参照 <https://vuejs.org/>

vue-router <https://router.vuejs.org/zh/>

vuex <https://vuex.vuejs.org/zh/>

## 二、SST 组件库代码交接

### 1 组件库实现原理

#### 1.1组件库实现：

组件库目的是为Vue 添加全局的功能， 在组件库中使用如下几种方式。

* 添加全局资源
* 添加全局方法或者属性
* 通过全集混入来添加一些组件选项
* 添加 Vue 实例方法， 通过把他们添加到 Vue.prototype 上面实现

#### 1.2 使用插件

通过全局方法 vue.use() 使用插件。 （Vue.use 会自动阻止多次注册相同插件）

import Sst from './sst'

Vue.use(Sst)

new Vue({

})

#### 1.3 插件的开发

Vue 插件暴露出一个 install 方法， 这个方法的第一个参数是 Vue 构造器， 第二个参数是可选对象。

export default install = (Vue, options) => {

Vue.GlobalMethods = () => {}

Vue.directive('my-directive', {})

Vue.component('my-component', '<div>com</div>'),

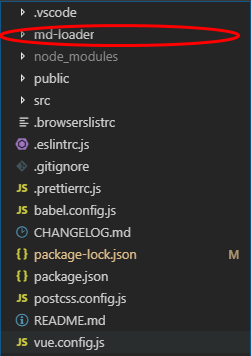
Vue.mixin({})

Vue.prototype.$xx = () => {}

}

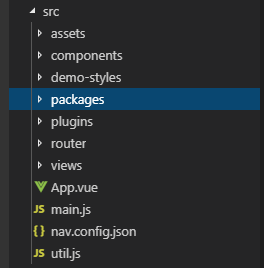
### 2 组件库目录结构

同 [PO项目的目录说明](#目录说明) 一样，由于组件库也使用 vue-cli@3 脚手架生成，目录结构基本相同， 但是多了一个 md-loader 模块， 这个模块是专门用来加载 .md 结尾的 webpack loader, 项目中所有markdown 文件都会通过这个 loader 打包。



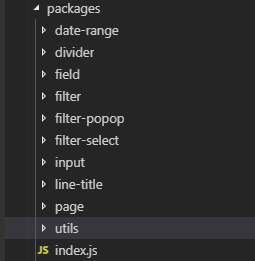
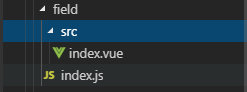
### 3 src 目录

因为本项目专注于组件开发，src 目录下并没有vuex 模块、permission模块等。 其中demo-styles 负责渲染组件文档样式，packages 存放开发组件、main.js 为项目的入口文件、nav.config.json 文件是 文档页面路由生成的依赖、util.js 为工具库文件。

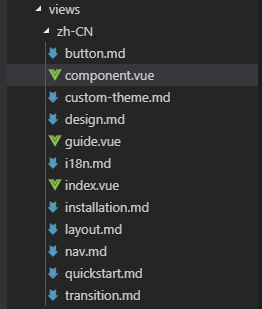
### 4 packages 目录

该目录存放组件源码。 每一个组件以一个组件目录存在， 每一个组件目录中又分为一个src/index.vue 文件和一个index.js 文件，前者用来存放组件源码， 后者用来编写install 和 export default 。所有的组件在 packages/index.js 中引用。

### 5 views 目录

同 PO 项目相似，views 目录同样用来编写页面组件。 不同的是这里的页面组件还包含了 .md 文件。 每一个 .md 文件是一个 页面组件。其中的编写格式遵循 markdown 语法和 [vuepress markdown 语法](https://vuepress.vuejs.org/zh/guide/markdown.html)。



### 6 组件开发流程

* 在packages 目录中新增一个组建目录和文件例如 sst-field、sst-field/src/index.vue、sst-fiels/index.js
* 在 sst-field 目录中添加一下代码。（这里的 install 函数是为了单文件暴露需要）

import Field from './src/index.vue'

Field.install = Vue => {

Vue.component(Field.name, Field)

}

export default Field

* 在 packages/index.js 文件中添加 sst-field 组件

import Field from './field'

const components = [

Field,

…

]

### 7 组件文档编写流程

经过上述步骤一个组件已经成功初始化，后续的组件测试则需要文档页面的支持。一个组件文档的编写通过下面几个步骤。（以 sst-field 组件为例）

* 在 nav.config.json 中新增 sst-field 的组件路由。Path 为组件文档的指向，这里设置为 ‘/field’ 意味着文档的实际地址为 views/zh-CN/field.md

"groups": [

{

"groupName": "Basic",

"list": [

{

"path": "/field",

"title": "输入框"

}

]

}

]

* 新建 views/zh-CN/field.md 文件
* 在 views/zh-CN/field.md文件中编写组件文档，并测试组件。一个初始化文档中会包含 以 :::demo 为开始符号， ::: 为结束符号的demo 片段， 在这个片段可以编写 组件 demo。 在组件中如需编写 vue 代码可以使用 以 ```html 为开始， ``` 为结束的 vue 代码块。

**# 输入框**

输入框组件

**## 基础用法**

:::demo 输入框文档

```html

<sst-field />

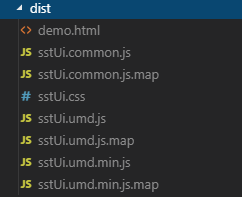
```

:::

* 运行项目， 查看组件呈现形式 yarn/npm run serve

### 8 打包组件库

运行yarn/npm run lib 将会在 根目录下生成一个dist 目录。 里面包含打包压缩后的 common规范和umd规范的js 文件,在其他项目使用时，根据其他项目的使用规范引用相对应的js 文件。



### 9 其他项目使用组件库

上一步中可以看到已经打包出了 .css 文件和.js 文件， 如果在 jsp 这种单文件中可以分别引入 .css 文件和.js 文件就可以了。 但是在 其他的 vue 项目中引用情况会有些不同。

* 项目依赖于 vue-cli@2 的情况。

1. Vue-cli@2 项目的静态文件存放在根目录的static 目录， 这时可以把 打包后的组件库的 sstUi.css 和 sstUi.common.js 这两个文件放到 static 目录下。
2. 找到 html-webpack-plugin 插件中配置的 index.html/index.tpl 模板(一般这个文件存放在根目录下)。在这个模板中手动插入 sstUi.css 依赖和sstUi.common.js 依赖。

<link rel="stylesheet" href="/sstUi.css">

<script src="/sstUi.common.js"></script> // 这行放在body 下

* 项目依赖于 vue0cli@3 的情况。

1. Vue-cli@3 项目的静态文件存放在根目录的public目录， 这时可以把 打包后的组件库的 sstUi.css 和 sstUi.common.js 这两个文件放到 public目录下。
2. 在public/index.html这个模板中手动插入 sstUi.css 依赖和sstUi.common.js 依赖。

<link rel="stylesheet" href="/sstUi.css">

<script src="/sstUi.common.js"></script> // 这行放在body 下

## 三、业务交接

### 1、Order Request 模块：

#### 1.1 Order Request 列表搜索逻辑

#### 1.2 Order Request订单的新建、修改、删除逻辑。

#### 1.3 Labels 逻辑

#### 1.4 PO Item QTY 数量计算逻辑。

### 2、Pending Tasks 模块

#### 2.1 Pending Tasks 列表搜索逻辑

#### 2.1 Pending Tasks 的 Approve 操作逻辑

#### 2.3 Pending Tasks 操作列逻辑(删除、记录、编辑)

### 3、Uncompleted Purchase 模块

#### 3.1 列表搜索逻辑

#### 3.2 操作列逻辑(修改、删除)

### 4、Vendor 模块

#### 4.1 Vendor 列表搜索展示

#### 4.2 Purchased History 和 Products Count 列表的跳转

#### 4.3 Vendor item 的新增

#### 4.4 Vendor item 修改编辑

#### 4.5 Vendor link Product 的逻辑

#### 4.6 Purchased Item 页面搜索与展示

### 5、 PO Summary

#### 5.1 Package detail 页面

#### 5.2 Shipping No. (Tracking No. 和 Barcode)

#### 5.3 Ready for Pick Up

#### 5.4 On Hold

#### 5.5 Mistake/Change 逻辑

#### 5.6 PO Item 编辑功能

#### 5.7 PO Item 删除功能

#### 5.8 PO Item Invoice 打印功能

### 6、 Assign Package

#### 6.1 Add Box 功能

#### 6.2 Assign Box 功能

#### 6.3 编辑 Box 内容(数量改变， 箱子交换功能， 改变 isFull 状态)

#### 6.4 Link PO 逻辑

### 7、Pending PO

#### 7.1 Pending Reason 逻辑

#### 7.2 Assign 逻辑

### 8、 Receiver Address

#### 8.1 address 增删改功能

### 9、Inquiry Setting

#### 9.1 Inquiry Setting 的筛选与修改

### 10、Receiving

#### 10.1 Receied PO 列表搜索与展示

#### 10.2 Mistake/Change 展示

#### 10.3 Mistake/Change 邮件发送功能

#### 10.4 Package Detail Print

#### 10.5 Package Detail inspection

#### 10.6 Inspection Detail (Add item from product or unknown item)

#### 10.7 Inspection Package 状态修改

#### 10.8 Issues item 数量、类型与 received QTY 之间的关系

#### 10.9 Inspection item checked 状态与 package inspect Completed 状态之间的逻辑

#### 10.10 Inspection item image

#### 10.11 改变为 restock 状态

### 11、 Ready for Pick Up

#### 11.1 pick Up 列表搜索与展示

#### 11.2 pick Up Location 更改

#### 11.3 Picked Up 状态的更改

#### 11.4 Print 功能