# **项目开发过程整理**

## 项目

项目名：《。。。》

开发方向：主要为H5页面开发，涉及APP与设备的交互，云端数据的交互。

项目开发团队：伟伟小分队

Connecting...

## 开发预备

1. **项目相关资料的阅读与熟悉**

项目背景

项目大体流程：厂商需求——UI稿设计——H5页面开发（含开发自测试）——H5页面与设备联调——测试——代码打包接入Hilink

项目设计规范（华为Hilink规范）：

* Hilink相关UI设计

gitea地址：http://192.168.65.156:3000/chenwei/Hilink-weiwei/src/branch/master/Z-UI规范

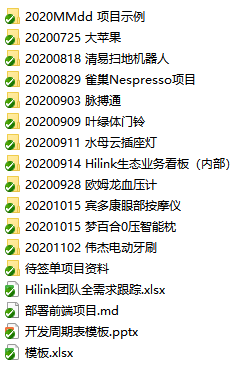
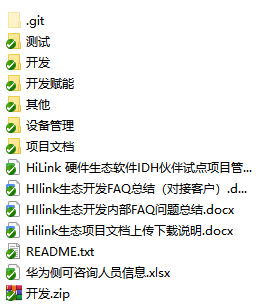
文件路径：Hilink-report\Z-UI规范

* 代码设计

1. **团队开发及测试的规范性问题（分别提供相关文档）**

开发部分：项目代码基本结构、团队代码风格统一、根据实际情况按要求执行开发步骤

注：团队的整体项目必须有清晰的资料管理，形成一个围绕HIlink建设的生态树，记录各个环节的进展、问题、收货等，为后续项目各个环节的进行奠定基础。例如如下文档结构：



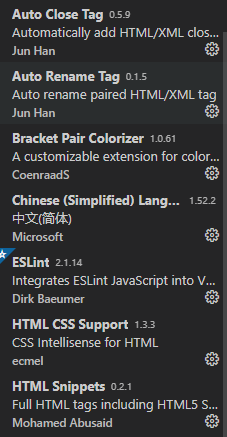
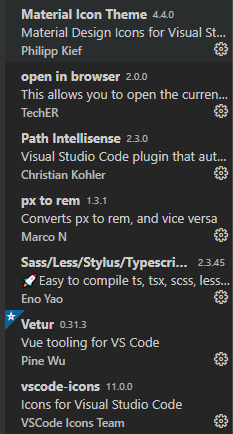
## 实际开发

开发使用的基本技术栈：VUE框架

开发工具：VSCode（推荐）、gitea(类似github的代码管理工具)

### **VSCode插件及配置**

#### （1）我使用的插件

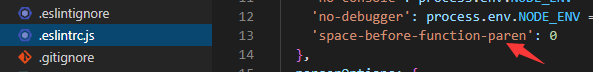
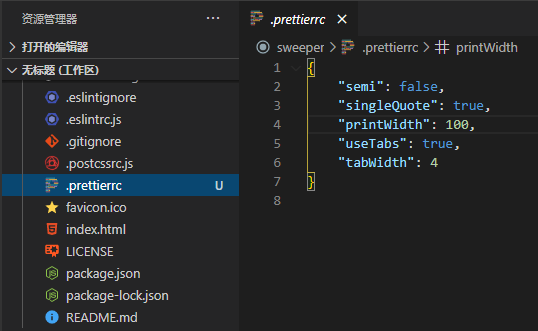
 

#### （2）插件冲突问题：

这里因为ESLint语法检查经常因为编码风格的问题（编码规范符合，不影响代码），影响开发，所以暂时把它给关闭了。

开启ESLint后，因为我使用的Prettier代码格式化工具，会产生格式冲突。所以针对两个插件做了相应的配置。（试过在settings.json中做配置，但是总是无效.....emmmm，只能这样了）。

注：不同人的编码风格不同，具体配置可能就不一样哦！



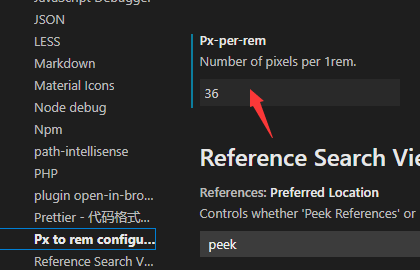
调整：不使用prettier，使用eslint 和vetur。取消保存自动格式化（使用默认的 Shift+Alt+F格式化然后Ctrl+S保存）。通常我们也在.eslintrc.js中做相应的代码格式设置。

#### （3）配置tips:

**① 移动端适配插件 px to rem**

开发时只需要用px单位即可，会自动转换成rem，你也可以选中数值Ctrl+Z进行px 和 rem之间的转换

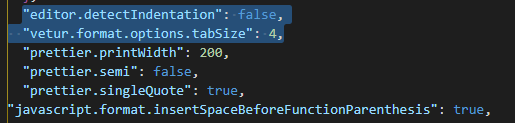
转换的比例单位根据开发需求设为了 36



不太推荐使用哈：

* 开发过程中，单位转换避免不了出现无线小数的出现，会出现一定的误差。
* 需要更改数值的时候，尤其是要根据上下文CSS的数值进行综合计算时，这么多小数，别说计算了，看着都头疼。

**② 每行代码缩进 4个空格（settings.json）**



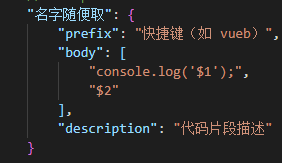
**③保存自动格式化（settings.json）**



**④ 自定义代码片段（Vue.js）**

直接上图...

我们只需通过以下操作建立一个自定义代码段文件（名字自定义。最好是XXXX.json）



这里我准备了两个，一个帮助我更加清晰地理解vue的template。一个帮助我快速生成简洁实用的vue模板代码。

上图！

    //   vue.js基础模板

    "vue-basic template": {

        "prefix": "vueB",

        "body": [

            "<!-- $1 -->",

            "<template>",

            "   <div>",

            "       $3",

            "   </div>",

            "</template>",

            "",

            "<script>",

            "// import 导入其他文件",

            "",

            "export default {",

            "   // import引入的组件需要注入到对象",

            "   components: {},",

            "   data() {",

            "       // 这里存放数据",

            "       return {$4",

            "       };",

            "   },",

            "   // 监听属性",

            "   computed: {},",

            "   // 监控data中的数据变化",

            "   watch: {},",

            "   // 方法集合",

            "   methods: {$5",

            "   },",

            "   // 生命周期 - 创建完成（可以访问当前this实例）",

            "   created() {",

            "   },",

            "   // 生命周期 - 挂载完成（可以访问DOM元素）",

            "   mounted() {$6",

            "   },",

            "   // 生命周期 - 创建之前",

            "   beforeCreate() {},",

            "   // 生命周期 - 挂载之前",

            "   beforeMount() {},",

            "   //生命周期 - 更新之前",

            "   beforeUpdate() {},",

            "   // 生命周期 - 更新之后",

            "   updated() {},",

            "   // 生命周期 - 销毁之前",

            "   beforeDestroy() {},",

            "   // 生命周期 - 销毁完成",

            "   destroyed() {},",

            "   // 如果页面有keep-alive缓存功能，这个函数会触发",

            "   activated() {}",

            "}",

            "</script>",

            "",

            "<style lang='less' scoped>",

            "</style>"

        ],

        "description": "vue 基础开发 模板"

    },

    //   vue.js实用开发模板

    "vue-using template": {

        "prefix": "vueU",

        "body": [

            "<!-- $1 -->",

            "<template>",

            "    <div>",

            "    Hello Vue$2",

            "    </div>",

            "</template>",

            "",

            "<script>",

            "",

            "export default {",

            "    data () {",

            "        return {$3}",

            "    },",

            "    computed: {},",

            "    watch: {},",

            "    methods: {$4},",

            "    created () {},",

            "    mounted () {}",

            "}",

            "</script>",

            "",

            "<style lang='less' scoped>",

            "</style>",

            "",

        ],

        "description": "vue 实用开发 模板"

    }

**⑤ eslinct.js语法配置修改（根据个人习惯来，可以不做修改直接用eslinct默认语法哦！）**

推荐配置：

const isPrd = process.env.NODE\_ENV === 'production';

module.exports = {

    root: true,

    parserOptions: {

        parser: 'babel-eslint'

    },

    env: {

        browser: true,

    },

    extends: [

        'plugin:vue/essential'

    ],

    // required to lint \*.vue files

    plugins: [

        'vue'

    ],

    // add your custom rules here

    rules: {

        // allow debugger during development

        'no-debugger': isPrd ? 2 : 0,

        'no-alert': isPrd ? 2 : 0,

        'semi': [2, 'always'],

        'spaced-comment': [2, 'always'],

        'no-trailing-spaces': 2,

        'array-bracket-spacing': [2, 'never'],

        'comma-spacing': [2, { 'before': false, 'after': true }],

        'space-infix-ops': 2,

        'space-before-function-paren': 2,

        'key-spacing': [2, { 'beforeColon': false, 'afterColon': true }],

        "keyword-spacing": 2,

        'indent': [2, 4]

    }

}

关于这块内容，可以看看如下链接：

ESlinct: <https://cloud.tencent.com/developer/doc/1078>

Eslinct与prettier区别：<https://blog.csdn.net/qq_21567385/article/details/109136668>

Connecting...

### **Npm 及Git 的安装（详见官网即可）**

这里推荐把镜像源换成淘宝镜像哦，npm => cnpm 。

git的话，为了方便可以使用小乌龟（Tortoise）。开发过程中不太推荐使用Git Bash方式提交代码，最好还是在vscode中做相关操作吧。

### **项目基本结构**

先安装最新版vue脚手架（npm install -g @vue/cli）

所有项目我们使用的都是vue-cli2x，这里我想用vue-cli3x进行尝试，嘻嘻

使用vue脚手架搭建项目基本结构（搭建时的具体需求按项目需要实施），推荐使用 vue ui 命令进入可视面板进行搭建（不易出错，一目了然）

一般搭建过程：

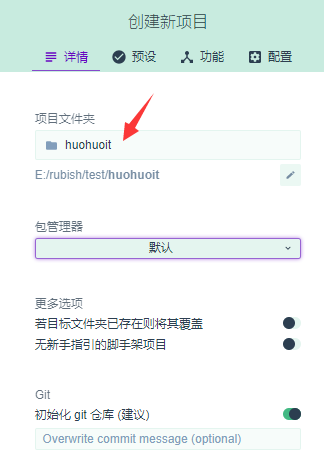
项目创建路径



路径选择完成后点击当前页面如下按钮



创建项目名（完成后下一步）



为项目选择一套搭建过的项目预设（这里我手动配置一次）



选择基本插件功能

其实Vuex根据实际情况来使用，我比较推荐使用嘻嘻。

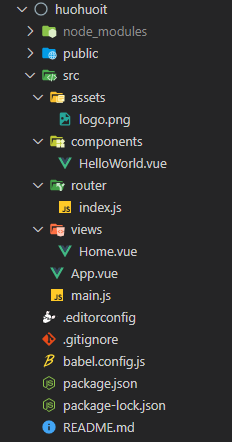
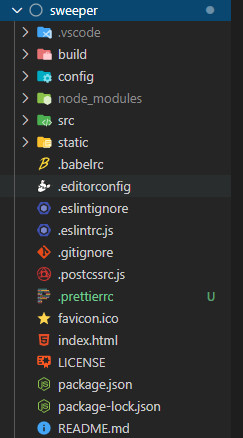
项目配置

语法检查配置

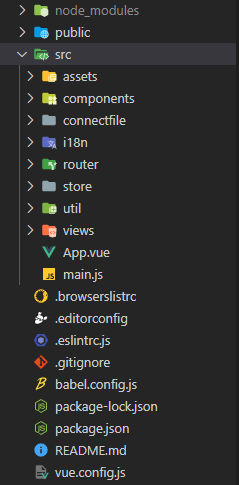


箭头这里有个ESLint + Prettier，后续待尝试，不知道是不是解决了两者的冲突问题（是的话那就不需要做那么多冲突处理了）

然后现在项目初始结构是左边这样的（记得把里面的关于初始界面的东西都删了，我们刚开发的时候，App.vue界面的样式没有删除，导致后续出现了一些bug（好低级哈哈哈哈）。一定要删除干净）。当时我们的结构却是右边这样的，是脚手架版本（vue-cli3以下版本）和初始配置的插件的原因。（踩了好多坑emmmmm!）

趁热来看看一个完整的项目目录结构（暂时不看打包配置哦！）



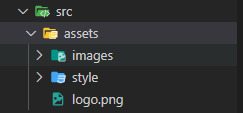
### **目录文件解析**

这里我们主要来看 src 下的文件夹

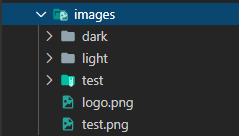
#### （1）assets 资源目录

记住：这里的资源会被webpack构建

assets文件夹下，我主要放的是图片images和CSS样式style



**① images文件**



文件的命令，模块的区分跟实际项目的需求来定，本文就针对我所做的项目来举例分析说明吧。

项目需求上APP界面需要有正常模式和暗黑模式（也就是夜间模式，当然可以类比开发其他更多的模式），每个模式下的字体、背景、图片的颜色和透明度都不同，所以这里图片分成两个子文件夹（dark和light）来放（方便开发和查看）

**Tips：**

* **图片模块化：**当然在图片较多的情况下，非常推荐在子文件夹里面继续根据某个功能模块将图片进行细分。
* **test文件：**还有一些图片可能各个模式可以共用，这里我就把这类图片放在test文件夹下了。
* 外面的零碎图片暂时就用来做开发测试啦，后面要删掉的。

**图片命名技巧：**

* + 不过，为了避免后续开发图片修改和维护，一定要注意，dark和light中对应的图片命名一定要相同（如果你有疑惑，往后看完暗黑模式，就明白咯）。
  + 另外，有些图片可能内含文字，在后面的国际化中，图片的英文版也是必须的，这时候如何做呢？举例：图片----1.png，英文图片----1en.png；图片----test.png，英文图片----testen.png。这里可能你会疑惑，写成test-en.png或者testEn.png不是更好吗。这里我作简单解释：图片比较多的情况下，很有可能用一行代码遍历出一大列图片出来，而我们的图片不是通过传统的src路径来引入，是通过class类名的方式（原因看暗黑模式），此时，如果这个class类名可以像数组下标一样，遍历出来，那是不是很方便呢。而，我们给图片命名的时候，最好跟类名保持一致哦。

先过目一下吧：

.hi-point19 {

  background: url('../images/point/19.png') center/contain no-repeat;

}

.hi-point20 {

  background: url('../images/point/20.png') center/contain no-repeat;

}

.hi-point1en {

    background: url('../images/point-en/1en.png') center/contain no-repeat;

}

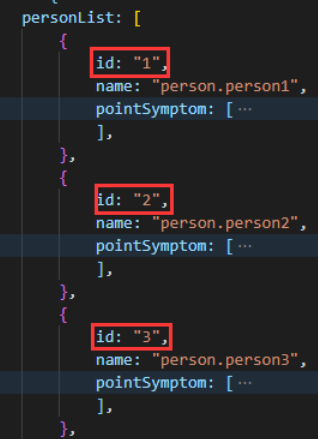
.hi-point2en {

    background: url('../images/point-en/2en.png') center/contain no-repeat;

}

上代码（这里的变量item.id就是数组项的id，直接可以对应hi-icon后加有序数字）：

:class="`hi-icon${item.id}`"



再看看英文的class类名如何处理？

:class="`hi-p1${entype}`"

变量处理--计算属性

        enctype () {

            return this.$i18n.locale === 'zh' ? '' : 'en';

        }

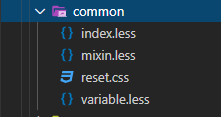
还能放在一起处理哦，直接解决了一大串图片的展示和中英文切换问题，是不是很快？

:class="`hi-point${item.id}${entype}`"

这里的变量名是自定义的哈

**② style文件**

**初版探索：**



* reset.css：这里放的是公共的初始化css文件

以及三个Less处理文件（这里只展示核心内容）

* variable.less：样式规范表----基本变量



* mixin.less：样式函数封装



* Index.less：less引用入口文件



Less文件封装完毕，那么我们该怎么引入呢？在我们需要使用到variable.less文件内的变量的时候，必须先引入该 index.js 文件，不然会报错。（路径根据实际位置，不要写错哦）

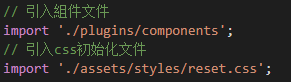


哪个vue文件里要使用，就在哪个文件引入，这样是没有问题的。

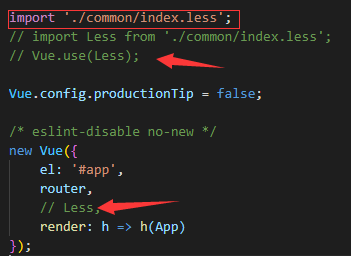
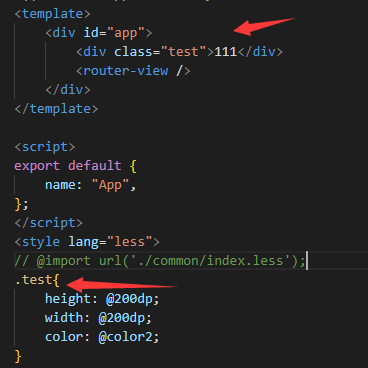
但是，我们的开发基本上每个页面的样式都需要使用less封装的变量，这样每个页面都需要引入一次，太麻烦了。

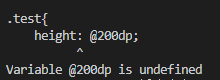
先来探索一下

1. 我们引入项目中的全局文件的时候，一般都会在 main.js中这么做



我尝试着也这么引入上面的less，并在页面做测试

，

发现会报错，，也就是无法这么引入。我又尝试了这几种方法看有什么效果

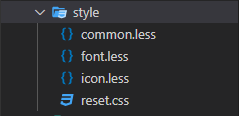
1. import from + Vue.use()
2. import from +直接挂载
3. @import url()

前两种方法还是报同样的错误，第三种方法可以正常显示。注意。第三种方法我是直接在App.vue中引入的（区别于在其他vue文件下），因为我想试试在这里引入，对其他页面有没有效果。不过发现其他页面还是无法使用Less变量。

解决办法：

1. 将项目所有页面的css放在同一个css文件下，比如common.css
2. 在App.vue内引入less（index.less）和这个common.css。不过推荐直接把less先引入到common.css，然后在App.vue下引入这个common.css。

**推荐模板：**



* reset.css：初始化CSS文件
* common.less：处理通用颜色（改一处就可以改变项目中所有使用到的地方，这里看情况使用哦，如果没做多模式切换的话，非常非常推荐）和带有数值的CSS属性（为了处理移动端的像素适配问题）

例如：

@size: 36;

@white: #ffffff;

@white10: rgba(255, 255, 255, .1); //暗黑模式-页面底部按钮背景色

@white20: rgba(255, 255, 255, .2); //暗黑模式-分割线

@white38: rgba(255, 255, 255, .38);

@white60: rgba(255, 255, 255, .60); //暗黑模式-内容描述性字体颜色

@white86: rgba(255, 255, 255, .86); //暗黑模式-头部标题、卡片标题、弹框主体内容（除弹框标题和按钮以外）、标题性文字

@white90: rgba(255, 255, 255, .9); //暗黑模式-弹框标题颜色、提示框内容颜色

@black: #000000; // 暗黑模式-界面背景色

@black3: rgba(0, 0, 0, .03);

@black10: rgba(0, 0, 0, .1);

@black20: rgba(0, 0, 0, .2);

@black30: rgba(0, 0, 0, .3);

@black60: rgba(0, 0, 0, .6);

@black90: rgba(0, 0, 0, .9);

@F7F7F7: #F7F7F7; // 普通模式背景色

@262626: #262626; // 背景色: 暗黑模式-卡片、弹框

@007DFF: #007DFF; // 普通模式字体颜色

@3F97E9: #3F97E9; // 暗黑模式文本颜色-弹框（确认、取消、被选中时间）、按钮、链接文本

@E64548: #E64548; // 暗黑模式字体颜色-删除

@404040: #404040; // 暗黑模式提示框背景色

@007DFF: #007DFF; //暗黑模式-功能选中背景色（加10%透明

@006CDE: #006CDE; //暗黑模式-控件高亮背景色（进度条、switch、按钮背景色）

.h(@dp) {

    height: unit(@dp/@size, rem);

}

.w(@dp) {

    width: unit(@dp/@size, rem);

}

.lh(@dp) {

    line-height: unit(@dp/@size, rem);

}

.padding(@t, @r, @b, @l) {

    padding: unit(@t/@size, rem) unit(@r/@size, rem) unit(@b/@size, rem) unit(@l/@size, rem);

}

.margin(@t, @r, @b, @l) {

    margin: unit(@t/@size, rem) unit(@r/@size, rem) unit(@b/@size, rem) unit(@l/@size, rem);

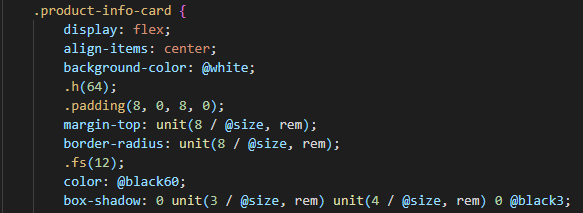
}

.fs(@dp) {

    font-size: unit(@dp/@size, rem);

}

实际使用：



* font.less：这个文件的存在 ==>暗黑模式。这里全部是涉及字体、背景等与颜色有关的class类，哪里需要就在哪里加上这个类名。注意，这里的普通类名与 .theme-dark下对应的类名必须相同，这样，在切换到暗黑模式下，类名会自动切换成对应暗黑模式的类名，妈妈再也不用担心我为了做暗黑模式适配在每个页面下多写 .theme-dark下的颜色class类了。其实这里也突出一个思想：将所有CSS样式抽离出来单独放在一个文件夹

这些代码就不再需要咯



看一些比较核心的例子吧：

// 一级文本

.fontColor90 {

    color: rgba(0, 0, 0, .9);

}

// 二级文本

.fontColor60 {

    color: rgba(0, 0, 0, .6);

}

// 页面背景色

.pageColorBg {

    background-color: #F7F7F7;

}

// 卡片背景色 及 阴影

.cardColorBg {

    background-color: #ffffff;

    box-shadow: 0 3/36rem 6/36rem 0 rgba(0, 0, 0, .03);

}

// 控件高亮色 高亮状态的点击效果 （滑杆等）

.controlColorOn {

    color: #007DFF;

}

/\* 分割线 \*/

.split-left {

    border-left: 0.25/36rem solid rgba(0, 0, 0, .2);

}

/\* 分割线 \*/

// 特殊文本颜色 （按钮等）

.font007DFF {

    background-color: #007DFF;

}

// 状态文字颜色

.color3F56EA {

    color: #3F56EA;

}

.color01AAEF {

    color: #01AAEF;

}

.color00BFC9 {

    color: #00BFC9;

}

.theme-dark {

    .fontColor90 {

        color: rgba(255, 255, 255, .86);

    }

    .fontColor60 {

        color: rgba(255,255,255,.6);

    }

    .pageColorBg{

        background-color: #000000;

    }

    .cardColorBg {

        background-color: #262626;

        box-shadow: 0 3/36rem 6/36rem 0 rgba(255, 255, 255, .03);

    }

    .controlColorOn {

        color: #006CDF;

    }

    .split-left {

        border-left: 0.25/36rem solid rgba(255, 255, 255, .2);

    }

    .font007DFF {

        background-color: #3F97E9;

    }

    .color3F56EA {

        color: #4056EB;

    }

    .color01AAEF {

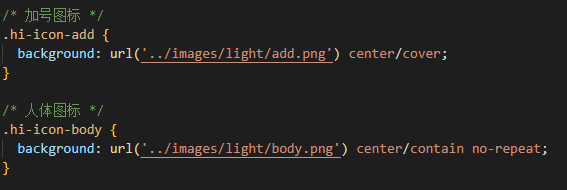
        color: #007ACC;

    }

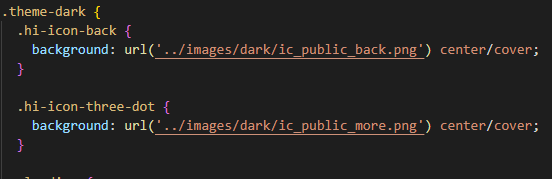
}

* icon.less：图片切换的核心（主要针对正常模式和暗黑模式的切换），所有图标图片都会写入这里的class，方便模式图片切换。

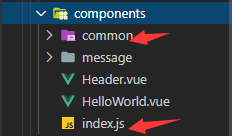
正常模式：



暗黑模式：



#### （2）components组件文件



common 内放的都是公共组件，index.js是基础组件的自动化全局注册配置

###### ① 全局注册组件

该项目中的所有组件采用了自动化全局注册的方式进行注册（如何注册主要根据项目的需求来定），该方式省去了很多繁琐的引入注册过程。

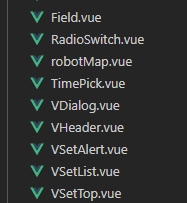
关于基础组件的自动化全局注册，参考官方文档

<https://cn.vuejs.org/v2/guide/components-registration.html#%E5%9F%BA%E7%A1%80%E7%BB%84%E4%BB%B6%E7%9A%84%E8%87%AA%E5%8A%A8%E5%8C%96%E5%85%A8%E5%B1%80%E6%B3%A8%E5%86%8C>

<https://github.com/chrisvfritz/vue-enterprise-boilerplate/blob/master/src/components/_globals.js>

新建一个components文件夹

所有组件只需要在该文件夹下编写.vue文件即可



（关键）项目在 文件夹根目录下的index.js里做的配置代码如下：

// 自定义组件

//

const requireComponents = require.context('./common', false, /\.vue$/)

// console.log(requireComponents.prototype);

function toFirstUpper (str) {

  return str.charAt(0).toUpperCase() + str.slice(1)

}

const install = Vue => {

  // requireComponents.keys()----["./VButton.vue"]

  requireComponents.keys().forEach(filename => {

    const componentConfig = requireComponents(filename) // 组件实例

    // filename ---- ./VButton.vue

    // 进行转化得到 VButton,另外防止有小写，进行首字母转化

    const componentName = toFirstUpper(

      filename.replace(/\.\//, '').replace(/\.\w+$/, '')

    )

    // 全局注册组件

    Vue.component(componentName, componentConfig.default || componentConfig)

  })

}

// 确保是正常环境

if (typeof window !== 'undefined' && window.Vue) {

  install(window.Vue)

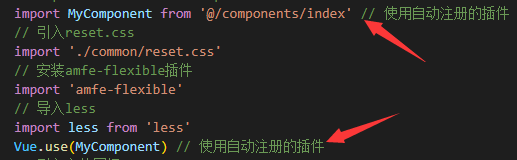
}

export default {

  install

}

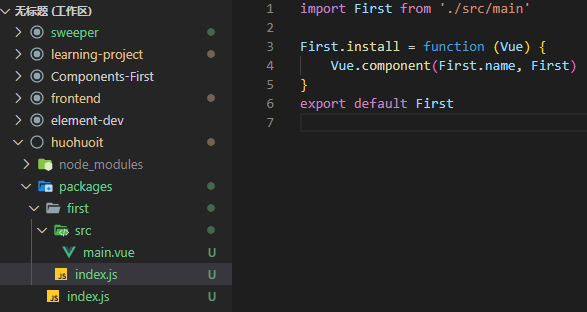
同时不要忘记在 main.js（最好单独建立一个文件夹，作为注册和使用组件的配置） 中引入



但是在搭建组件库时，不推荐都用全局注册。全局注册往往是不够理想的。比如，如果你使用一个像 webpack 这样的构建系统，全局注册所有的组件意味着即便你已经不再使用一个组件了，它仍然会被包含在你最终的构建结果中。这造成了用户下载的 JavaScript 的无谓的增加。（按需加载，按需引入的问题）

###### ② 局部注册组件（按需导入）

模仿element-ui，以如下文件格式开发组件。为组件添加 install 方法，用于**按需导入**，并暴露出来（这样会比较麻烦，但是性能好很多，且非常利于开发组件库）。



编辑 packages/index.js文件实现组件的全局注册。定义install方法

该部分配置详细例子请看如下代码：

// 导入单个组件

import First from './first/index'

// 以数组的结构保存组件，便于遍历

const components = [First]

// 定义 install 方法

const install = function(Vue) {

  if (install.installed) return

  // 遍历并注册全局组件

  components.map((component) => {

    Vue.component(component.name, component)

  })

}

if (typeof window !== 'undefined' && window.Vue) {

  install(window.Vue)

}

export default {

  // 导出的对象必须具备一个 install 方法

  install,

  // 组件列表

  First,

}

引入注册(一般是新建文件夹放一个js文件)

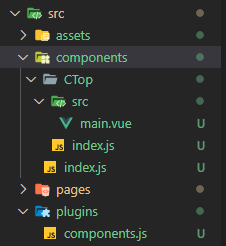
import First from './../packages/first/index'

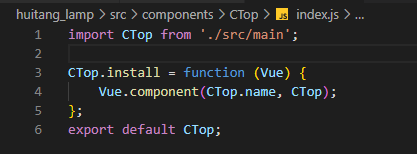
Vue.use(First)

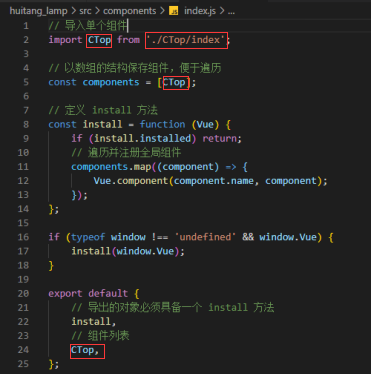
可参考：<https://www.cnblogs.com/luoluo-snow/p/11636943.html>

###### ③ 详细案例参考

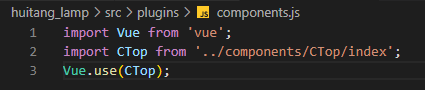
如下示例（非原项目）：components中放组件和index.js文件，每个组件单独建立一个文件夹，如下目录结构。







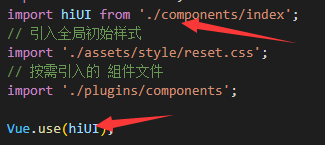
另外单独建立一个plugins文件夹，建立components.js文件，用来按需导入已注册的组件（不推荐这一步，多余）



注意：记得在main.js中引入这个components.js文件



如果在项目开发过程中，components中的所有组件都有用到，按需导入就显得有些多余，此时不用单独新建plugins文件夹，再引入components.js文件。而是直接在main.js中引入components文件根目录下的index.js文件。（HiUI为自定义名，别忘记在下面加上vue.use）

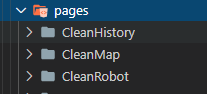


#### （3）image图片文件夹

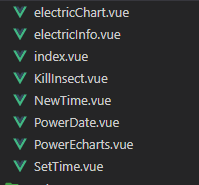


#### （4）pages项目页面文件夹

每个单独页面用单独的文件夹

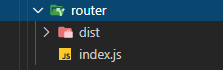


当然，如果项目比较小，页面及页面类比较少，就没必要建多余的文件夹啦！



#### （5）router路由文件夹

做路由配置。dist文件夹为开发模式构建（暂不清楚）

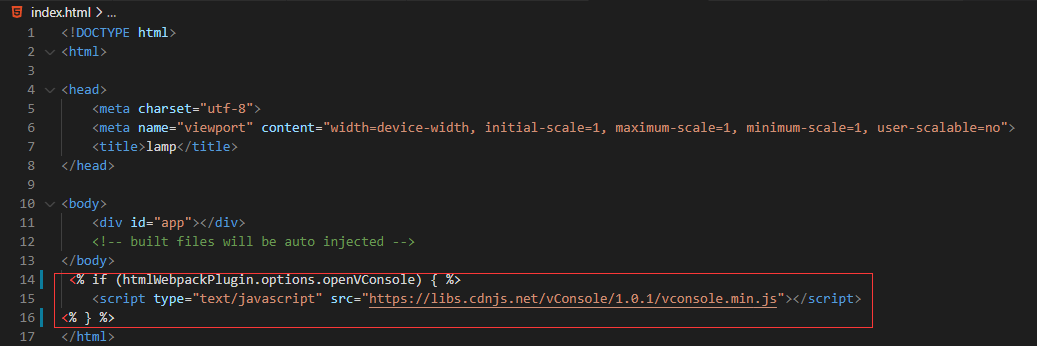


#### （6）App.vue 根组件

#### （7） main.js 入口js文件

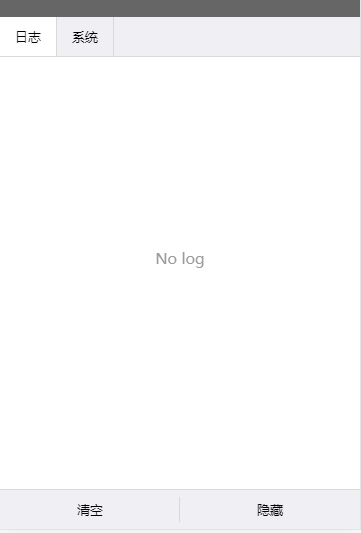
#### （8）index.html

我们做的是一个单页面项目。Vue项目运行npm run build后会生成一个dist文件夹，我们一般都是把这个文件夹部署到服务器上。dist文件夹里边有一个static文件和一个index.html页面，这个index就是最后单页面的最终文件。

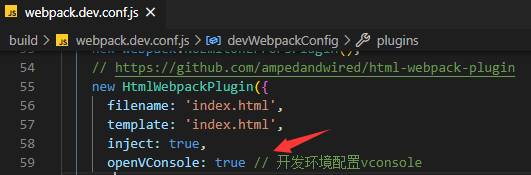


注意到这三行代码

1. 模板语法：<% if (这里做渲染判断) {%>需要渲染的内容<%} %>
2. script标签：这里引入了一个外部JS。（因为我们做的是移动端项目，需要在手机设备上做调试，查看移动端的控制台。）



需要注意的是，我们是在开发模式才需要这个面板，所以要在开发模式下做相应的配置



### **页面开发问题**

#### （1）设置界面

**① 设置列表的开发问题**

其中一部分主体内容的第一版（基本写法）

      <!-- 基本设置 -->

      <div class="basicSet">

        <span class="nextPart">基本设置</span>

        <div class="itemSet">

          <div class="typeSet">

            <span>设备名称</span>

            <span class="rightIcon"></span>

          </div>

          <div class="typeSet">

            <span>设备位置</span>

            <span class="anotherItem">默认房间</span>

            <span class="rightIcon"></span>

          </div>

          <div class="typeSet">

            <span>设备信息</span>

            <span class="rightIcon"></span>

          </div>

          <div class="typeSet">

            <span>设备网络信息</span>

            <span class="anotherItem">DASDADADAS</span>

            <span class="rightIcon"></span>

          </div>

          <div class="typeSet">

            <span>自动升级</span>

            <span class="rightIcon"></span>

          </div>

        </div>

第二版（v-for遍历写法）

      <!-- 基本设置 -->

      <div class="basicSet">

        <span class="nextPart">基本设置</span>

        <div class="itemSet" v-for="item in basicSetList" :key="item.id">

          <div class="typeSet" @click="basicSetRouter(item)">

            <span>{{ item.name }}</span>

            <span class="anotherItem">{{ item.note }}</span>

            <span class="rightIcon"></span>

          </div>

        </div>

第三版（组件化）

                <!-- 基本设置 -->

                <v-set-list

                    :moduleName="basicName"

                    :setList="basicSetList"

                    @doClick="doClick($event)"

                ></v-set-list>

**② 页面滚动条的问题**

参考这篇博客： <https://blog.csdn.net/huohuoit/article/details/108206317>

参考诚哥的建议，后面使用了 better-scroll 插件对页面滚动进行了优化

参考地址https://cloud.tencent.com/developer/article/1605081

#### （2）布局问题

首先明确一点，移动端页面布局最好用**弹性布局**。开发的初期，我并未完全使用弹性盒子进行布局，导致样式代码量大且复杂。

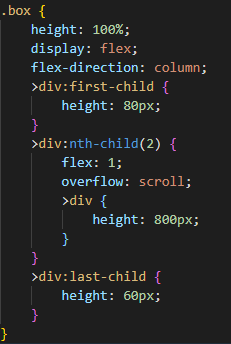
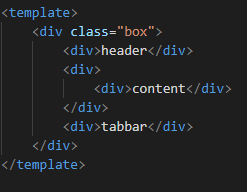
1. 例一

上面的页面滚动条问题，页面的的顶部是一个固定导航栏，而这个部分多出来的空间，是由我给页面主体content加了padding-top多出来的，后续出现的滚动条问题，也是这个引起，所以做了上述相对繁琐的处理。

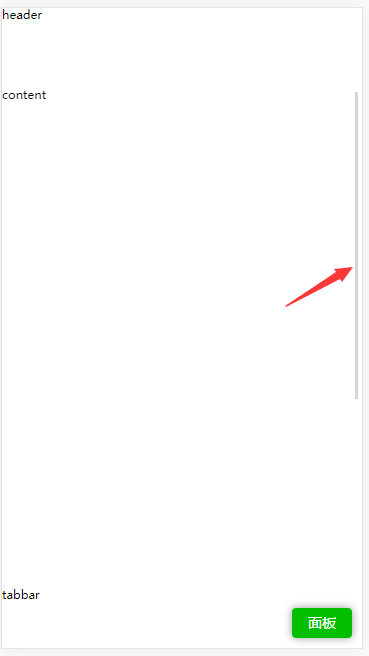
而在这里如果我们使用弹性盒子，只需要给弹性容器设置display:flex。

注意：我们要实现的一个布局是头部固定导航栏 + 中间页面主体 + 底部固定tabbar，方向是纵向的，而默认是横向的，所以要利用flex的另一个属性flex-direction：column变向。

弹性子元素，我们只需要给上下两边一个高度，中间flex:1即可（**三列布局**）

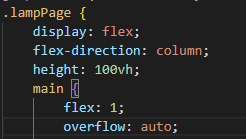
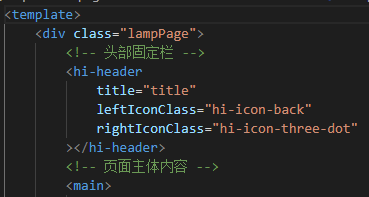


再给需要出现滚动条的地方加上overflow: scroll，页面效果如下



这样就可以很快处理上面的滚动条问题了。

也可以参考这个例子（我把overflow:scroll改成auto）：



#### （3）H5移动端安卓/IOS兼容问题

**① 日期格式问题**

一般在安卓中我们会这样格式一个日期

   new Date(data, 'yyyy-MM-dd')

推荐：但是在IOS系统下，是不支持“-”的，页面中会报错，做兼容的时候需要换成“/”

    new Date(data, 'yyyy/MM/dd')

也可以支持如下两种格式：

    new Date(data.replace(/-/g, '/'))

    new Date(2020,11,16,15,02,33)

**6、暗黑模式**

#### （1）本质

样式切换

重点还是样式，每切换一个模式，对应的都是样式的变化，注意包括背景色、字体色、图标、图片的变化。

#### （2）关键步骤

1. app.vue文件下做全局切换设置
2. 图标和图片都以样式的方式引入，暗黑模式的就用theme-dark父元素包裹
3. 处理每个页面有色彩样式的元素（只需要改变其色彩），处理办法就是用theme-dark父元素包裹这些新的色彩样式

#### （3）一键全局切换模式呢？

只需要在app.vue文件下，根元素上动态绑定一个class。

在created生命周期控制当前显示的是哪种模式。下面代码我们在这里给出了整个页面暗黑模式下的背景色。关键步骤如下红色框框标注：

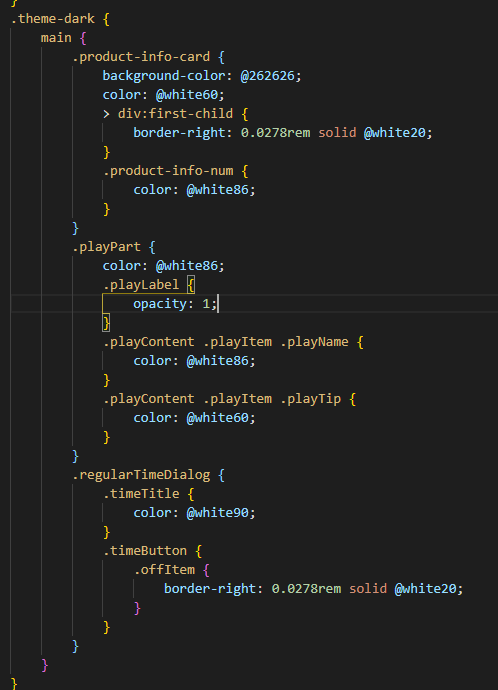


注：该代码为Hilink接口方法的处理，调用该接口，可在手机端通过模式设置，切换页面的模式。

this.isDark = window.hilink && window.hilink.getDarkMode() === 2;

#### （4）其他页面处理

在其他的页面上，我们只需要改变拥有色彩的元素的CSS。注意，这些CSS此时都是theme-dark的子元素，且都是符合暗黑模式UI规范的CSS。如下：

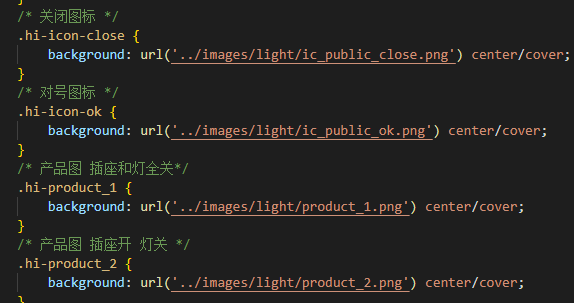


#### （5）图片图标的处理

我们已经在根组件切换了暗黑模式，相应地在根元素（id=app）上添加了一个theme-dark样式。如果此时图片图标也能基于这个新加的样式改变成暗黑模式的，就可以达到切换的效果。

处理如下：

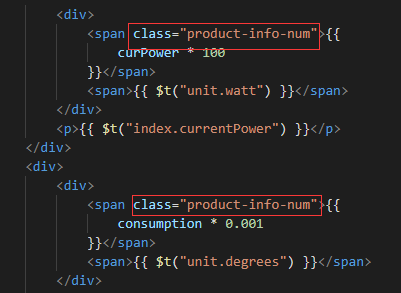
① 将所有图标和图片通过样式的方法引入到页面中（这些样式可以放在assets/styles/icon.less）



② 正常模式的不再处理，暗黑模式的图标图片用theme-dark包裹（改变了样式层级）



页面上只需要这样使用即可



**7、国际化**

#### （1）插件：vue-i18n

详细内容可见官网：<http://kazupon.github.io/vue-i18n/introduction.html>

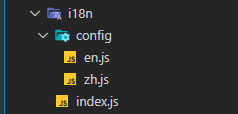
#### （2）中英文切换

**① 安装**

cnpm install vue-i18n

**② 使用配置**

推荐文件结构：src目录下新建文件夹i18n



index.js文件配置：

import Vue from 'vue';

import VueI18n from 'vue-i18n';

import zh from './config/zh';

import en from './config/en';

Vue.use(VueI18n);

const i18n = new VueI18n({

    locale: 'zh',

    messages: {

        zh,

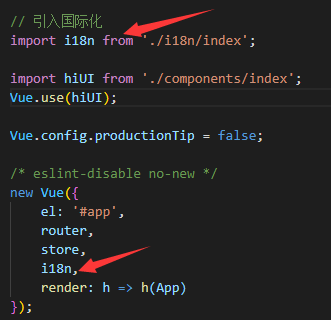
        en

    }

});

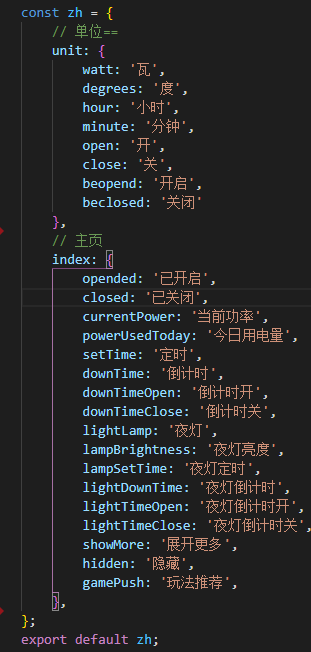
export default i18n;

注意：别忘了在入口文件 main.js中引入并挂载

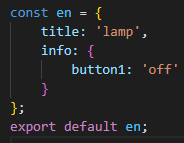


中文配置文件 zh.js：

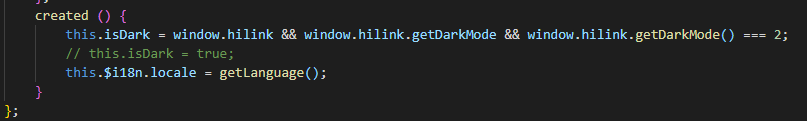
推荐单位、页面、特定功能模块的变量放在同一个对象中



英文配置文件 en.js：



一般都是一键全部中英文切换，类似暗黑模式，可以在App.vue中做处理：



**③ 页面使用举例**

推荐在中文页面全部开发完之后，将代码中涉及中文的部分全部用 zh.js 中配置的键值对替换

HTML中替换变量：

:leftText="$t(outlet.leftText)"

 <p class="playName">{{ $t(item.name) }}</p>

HTML中替换常量：

 :leftText="$t('index.setTime')"

 <p>{{ $t("index.currentPower") }}</p>

JS中替换常量：

lampCountDown: {

                leftText: this.$t("index.lightDownTime")

            },

this.outlet.leftText = this.$t("index.closed");

this.countDown.title = this.outlet.flag

                    ? this.$t("index.downTimeOpen")

                    : this.$t("index.downTimeClose");

对于列表的中文数据处理，最好如下处理：



