

前端利用 Vue.js 开发项目小结

——以互联网小贷风控系统为例

李鸿文 2017/8/1

培训内容：

- 1.开发环境搭建
- 2.关于 webpack 模板的目录结构说明
- 3.npm 常用命令
- 4.webpack 结合 git 用法
- 5.Vue.js 介绍和基本用法
- 6.Vue-router 路由用法
- 7.axios 基本用法及解决 ajax 跨域请求配置
- 8.Element ui 基本用法介绍
- 9.分享互联网小贷风控系统开发心得

1.开发环境搭建

1.1、环境准备

1.1.1 安装软件：

1)从 node.js 官网下载并安装 node 输入 node -v 查看版本

2)npm 包管理器，是集成在 node 中的，所以，直接输入 npm -v 查看版本

1.1.2 安装构建工具和脚手架：

1) npm install webpack -g //安装 webpack 输入 webpack -v 查看版本

2) npm install -g vue-cli //安装 vue 脚手架 输入 vue -V 查看版本

注意，此时 V 是大写

1.2 构建项目:

安装完成后，即可运行命令：

```
$ vue init <template-name> <project-name>
```

<template-name>：模板名称，运行 npm list 会显示有什么模板可以选择；

<project-name>：需要命名的项目名称；

例如运行：

1.vue init webpack my-project //在工作目录下生成项目 my-project

2.npm install //在项目目录初始化 npm

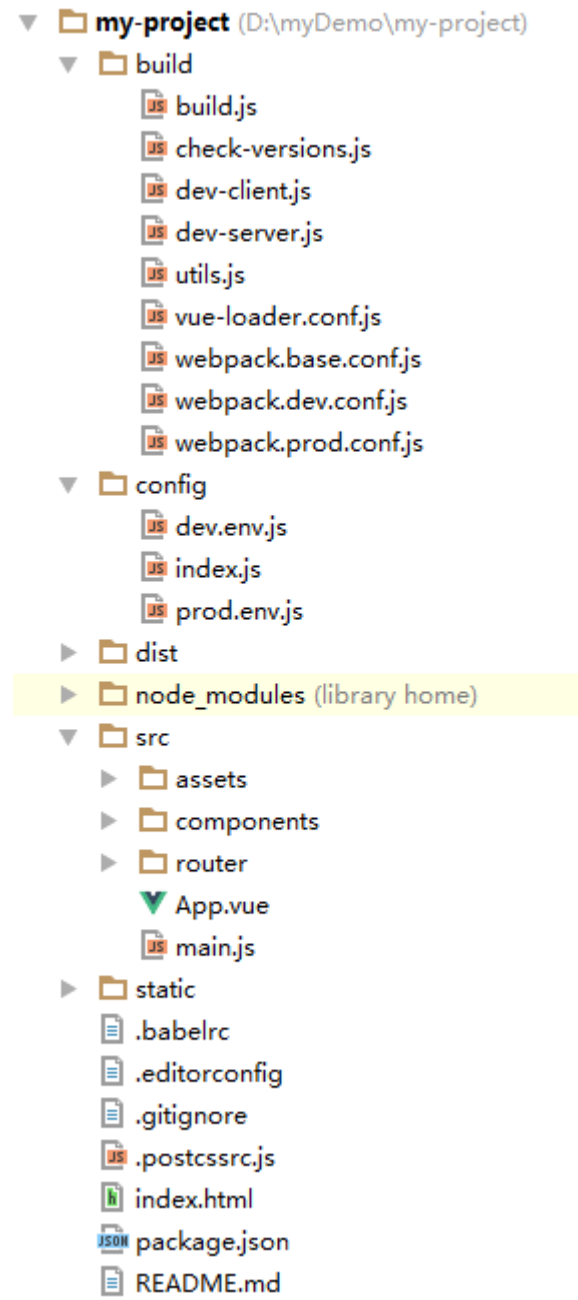
1.3 启动和打包:

1. npm run dev //启动开发

2. npm run build //打包发布

2. 关于 webpack 模板的目录结构说明

2.1 生成的目录结构：



2.2 目录结构说明：

- | --build //webpack 相关代码文件夹
 - | | --build.js //生产环境结构代码
 - | | --check-version.js //检查 node、npm、等版本
 - | | --dev-client.js //热加载相关代码
 - | | --dev-server.js //本地服务器
 - | | --utils.js //构建工具
 - | | --webpack.base.conf.js //webpack 基本配置
 - | | --webpack.dev.conf.js //webpack 开发环境配置
 - | | --webpack.prod.conf.js //webpack 生产环境配置
- | --config //项目开发环境配置
 - | | --dev.env.js //开发环境变量
 - | | --index.js //项目基本配置 (proxyTable:{ //配置请求代理})
 - | | --dev.env.js //开发环境变量
 - | | --prod.env.js //生产环境变量
- | --dist //执行 npm run build, 生成打包发布的目录
- | --node_modules //初始化 npm install, 生成的依赖包目录 (注意, 不要提交到 svn !)
- | --src //项目源代码目录
 - | | --components //组件目录
 - | | --assets //Vue 默认 logo 目录
 - | | --router //路由目录
 - | | --APP.vue //默认组件, 入口文件
 - | | --main.js //程序入口文件, 引用、加载各种组件
- | --static //静态文件目录, 比如: CSS、图片、等等静态文件
- | --index.html //入口文件

3. npm 常用命令

3.1、npm 包安装模式

》》 本地安装：package 会被下载到当前所在目录，也只能在当前目录下使用。

》》 全局安装：package 会被下载到特定的系统目录下，安装的 package 能够在所有目录下使用。

3.2、安装模块

1) 本地安装，如 grunt-cli

npm install grunt-cli：安装包 grunt-cli，默认会安装最新的版本

npm install grunt-cli@"0.1.9"：安装 0.1.9 版本的 grunt-cli

npm install grunt-cli --save 或 npm install grunt-cli -S：安装包 grunt-cli 并将信息将加入到 package.json 文件的 dependencies（生产阶段的依赖）

npm install grunt-cli --save-dev 或 npm install grunt-cli -D:安装包 grunt-cli 并将包信息写入 package.json 文件的 devDependencies(开发阶段的依赖)配置中，这样代码提交到 github 时，就不用提交 node_modules 这个文件.

npm install grunt-cli --save-optional 或 npm install grunt-cli -O:安装包 grunt-cli 并将信息将加入到 optionalDependencies（可选阶段的依赖）

模块的依赖都被写入了 package.json 文件后，他人打开项目的根目录（项目开发、内部团队合作），使用 npm install 命令可以根据 dependencies 配置安装所有的依赖包

2) 全局安装，如 npm

`npm install -g npm` :全局安装 npm

`npm install -g npm@2.14.14` :安装指定的 npm 版本, 同时也是降低和更新 npm 版本的方法 ;

全局安装模块的目录一般为 :

`c:\User\Administrator\AppData\Roaming\npm\node_modules\`下 ;

全局安装的模块, 在代码中直接通过 `require()`的方式是没有办法调用到的。全局的安装是供命令行使用的, 就好像全局安装了 `vmarket` 后, 就可以在命令行中直接运行 `vm` 命令。也就是可以直接在 `cmd` 的命令行中使用。

3.3、卸载模块

比如 `grunt-cli`

`npm uninstall grunt-cli`: 卸载包 `grunt-cli`

`npm uninstall grunt-cli@"0.1.9"`: 卸载 0.1.9 版本的 `grunt-cli`

3.4、更新模块

比如 `grunt-cli`,全局 npm

`npm update` #升级当前目录下的项目的所有模块

`npm update grunt-cli` 更新

`npm update -g npm@3.14.14`, 安装指定的 npm 版本, 同时也是更新 npm 版本的方法 ;

3.5、查看模块

比如 `grunt-cli`

`npm ls grunt-cli` : 查看特定包 `grunt-cli` 的信息

`npm info grunt-cli` : 查看详细的输出信息 (包括作者、版本、依赖等)。

npm list #列出已安装模块

npm show grunt-cli #显示模块详情

3.6 实例：

引入 element ui <http://element.eleme.io/1.3/#/zh-CN/component/installation>

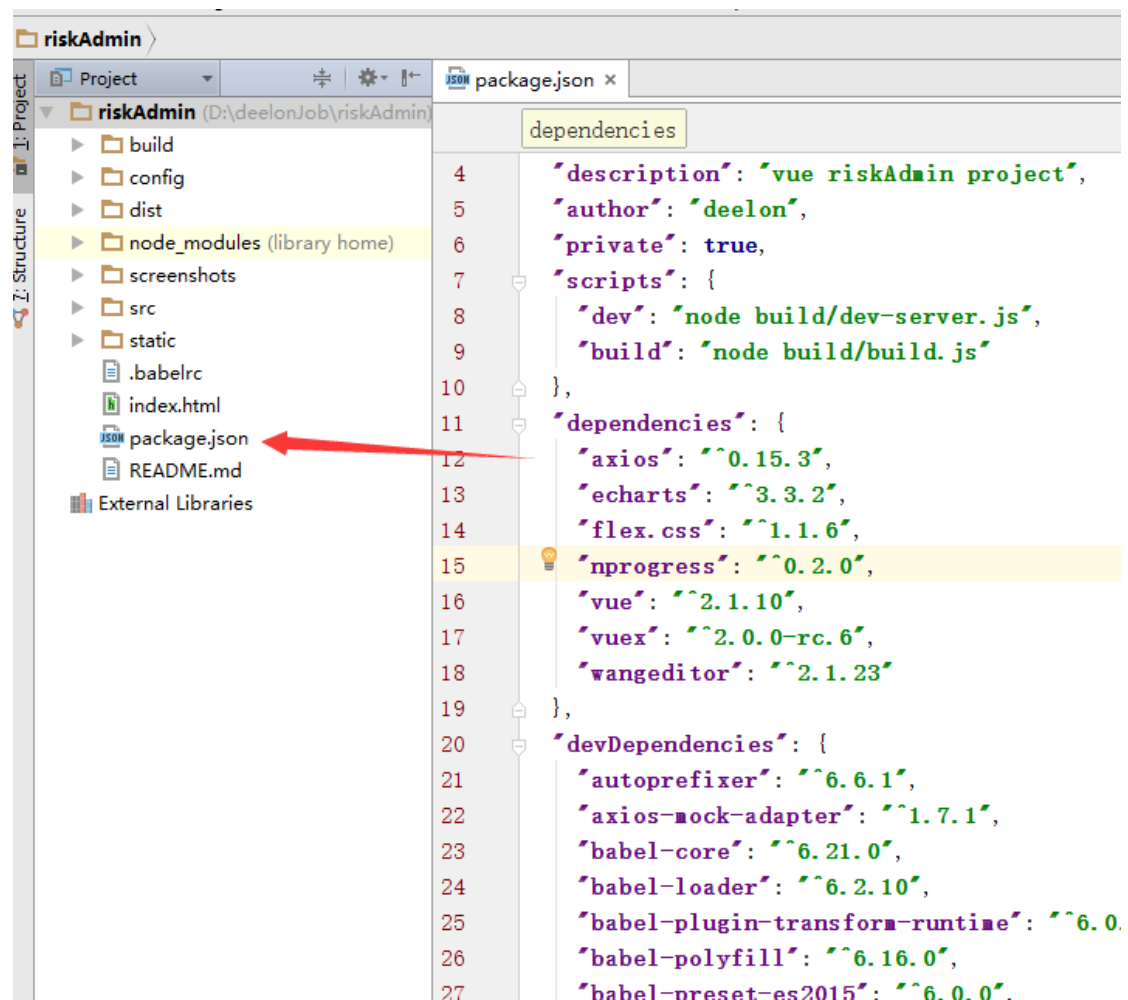
npm 安装，命令如下：

npm i element-ui -S //简写

同样，引入 axios，——http ajax 请求组件，命令如下：

npm install axios --save

安装包并将信息保持到项目的 package.json 文件中，见下图：



【延伸】：

-S, --save 安装包信息将加入到 dependencies（生产阶段的依赖），如：

npm install gulp --save 或 npm install gulp -S

package.json 文件的 dependencies 字段：

```
"dependencies": {  
  "gulp": "^3.9.1"  
}
```

-D, --save-dev 安装包信息将加入到 devDependencies（开发阶段的依赖），所以开发阶段一般使用它，如：

npm install gulp --save-dev 或 npm install gulp -D

package.json 文件的 devDependencies 字段：

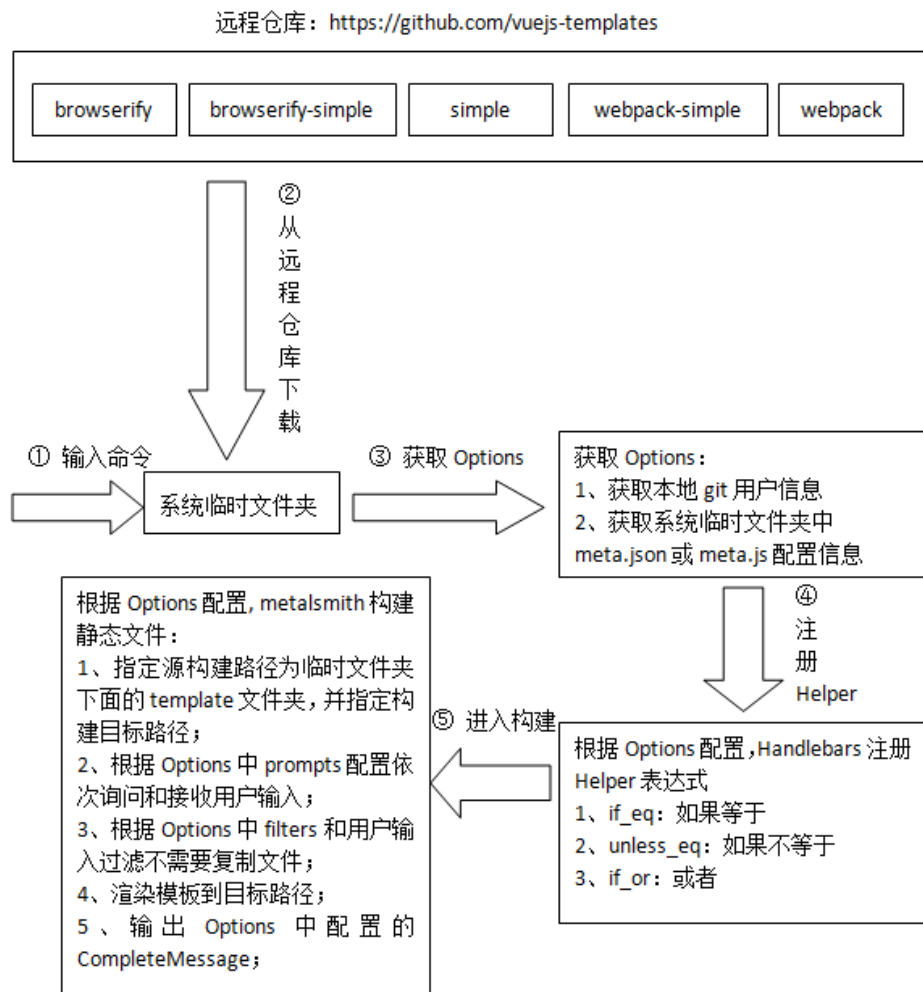
```
"devDependencies": {  
  "gulp": "^3.9.1"  
}
```


4. webpack 结合 git 用法

基于 webpack+vue-cli 构建项目的原理图，不难看出：

构建步骤

vue-cli中命令init最复杂，其它命令很简单，下面用图表示下vue init构建步骤：



5. Vue.js 介绍和基本用法

学习一个新技术，必须要清楚两个 W，"What && Why"。

"XX 是什么？”，

"为什么要使用 XX，或者说 XX 有什么好处”，

最后才是"XX 怎么使用”。

前端技术发展很快，来势汹汹~已经跟不上了~

前端构建工具 Gulp / browserify/ webpack / npm ?

- node , 是javascript语言的环境和平台 ,
- npm , bower 是一类 , 包管理 ,
- webpack , browserify , 是一类 , javascript预编译模块的方案 ,
- requirejs , seajs , 是一类 , 基于commonjs , amd , cmd , umd 之类的模块类包加载方案的框架 ,
- grunt , gulp , 前端工具 , 合并、压缩、编译 sass/less , browser 自动载入资源 ,
- react , angular , vue , backbone , 是一类 , mvc , mvvm , mvp 之类的前端框架 ,
- jquery , zepto , prototype , 是一类 , 前端 DOM , BOM 类库 ,
- ext , yui , kissy , dojo , 是一类 , 前端应用组件 ,
- underscore , 函数式编程库。

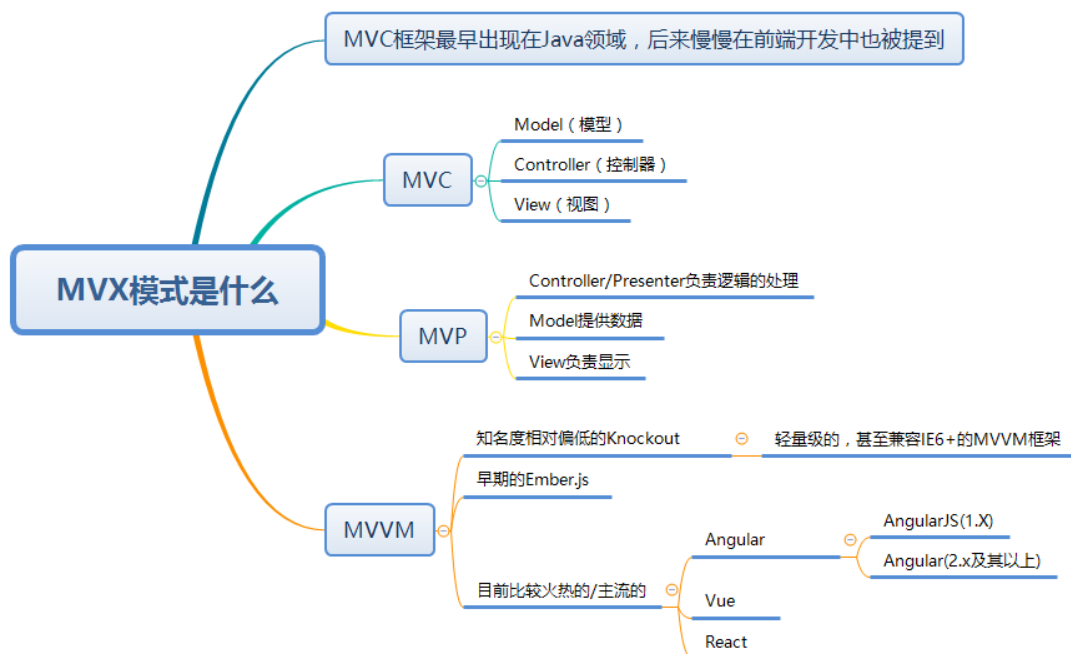
然后, 找喜欢的上。

发布于 2016-07-28

上图：知乎网友的整理，仅供参考。

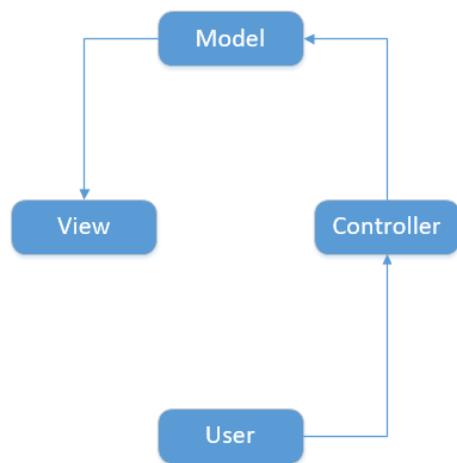
5.1 MVX 模式是什么？

下图是我参考《VUE.JS 权威指南》一书的理解，画的思维导图：



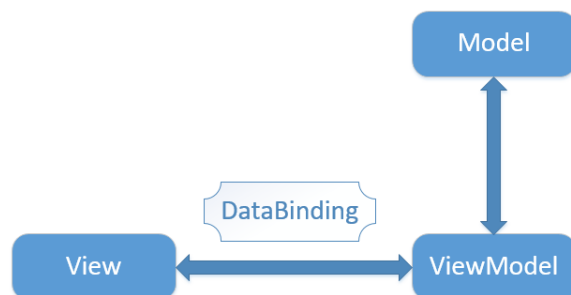
》MVC 模式通讯方式：用户（User）通过 Controller 来操作 Model 已达到 View

的变化。



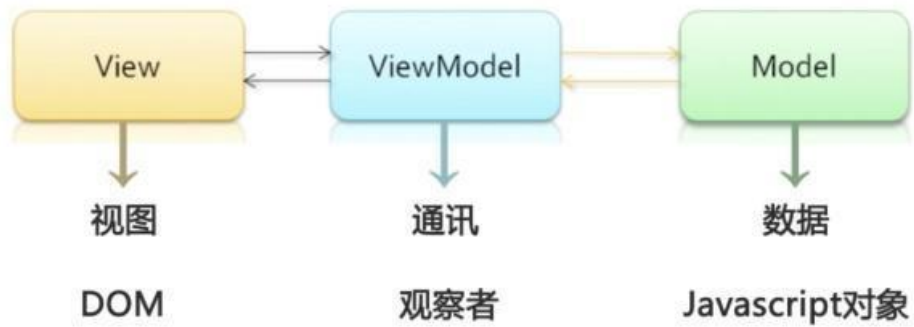
》MVP 模式：MVP 是从经典的 MVC 模式演变而来，它们的基本思路有相通的地方：Controller/Presenter 负责逻辑的处理，Model 提供数据，View 负责显示。

》MVVM 模式：相比前面两种模式，MVVM 只是把 MVC 的 Controller 和 MVP 的 Presenter 改成了 ViewModel。View 的变化会自动更新到 ViewModel，ViewModel 的变化也会自动同步到 View 上显示。



MVVM 全称是 Model - ViewModel - View 的简称

MVVM框架



Model 对应的是数据，JavaScript 对象。

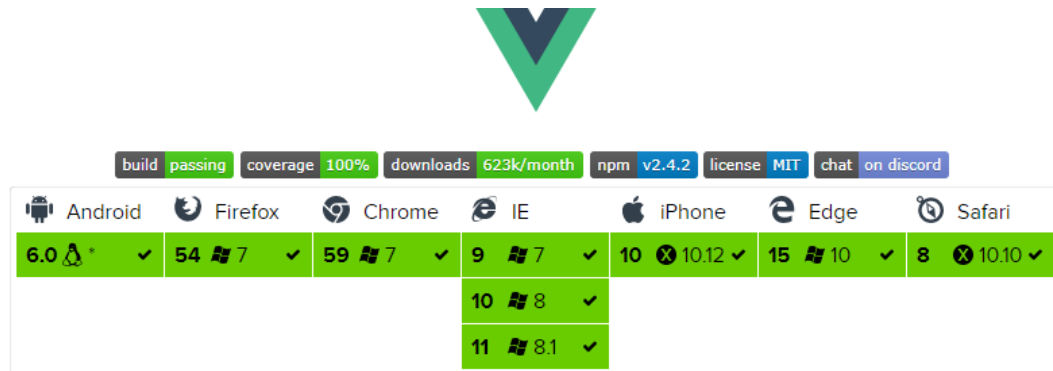
View 对应的是视图，也就是 DOM。

ViewModel 是连接 View 和 Model 的中间件，在 MVVM 下，View 和 Model 是不能直接通讯的，当数据变化（用户操作视图），ViewModel 都能监听到变化，实现了双向绑定。

5.2 Vue.js 是什么

官网：<https://cn.vuejs.org/v2/guide/>

Vue 的兼容性，见下图：



5.3. vue / angular /react 区别

对比主流框架：<https://cn.vuejs.org/v2/guide/comparison.html>

Vue 跟 react 使用 Virtual DOM，组件化的视图组件，将注意力集中保持在核心库，而将路由和全局状态管理分离出来（交给相关的库）

Vue 和 AngularJS (1.x 版本)，语法很相似（如 v-if 和 ng-if），借鉴了 AngularJS

5.4 基本用法

一、基础语法

1、v-model(绑定数据)

2、v-for（循环）

3、v-on（绑定事件）

4、data（数据）

5、methods（方法）

6、v-if（不满足条件的话则不会出现在 dom 中）

7、v-show（不满足条件，则样式会设置成隐藏 display:none;）

```
<a @click="doSomething"></a>
```

5.5 理解生命周期图

<http://cn.vuejs.org/v2/guide/instance.html#生命周期图示>

理解，参考《Vue.js 权威指南》一书，

》》 init: 在实例开始初始化时同步调用。此时数据观察、事件和 Watcher 都尚未初始化。

》》 created: 实例创建之后同步调用。此时已建立：数据绑定、计算属性、方法、Watcher/事件回调。但是还没有开始 DOM 编译，\$el 还不存在。

》》 beforeCompile: 在编译开始前调用。

》》 compiled: 在编译结束后调用。此时所有的指令已经生效，因而数据的变化将触发 DOM 更新。但是不担保\$el 已插入文档。

》》 ready: 在编译结束和\$el 第一次插入文档之后调用，如在第一次 attached 钩子之后调用。注意，必须是由 Vue 插入（如 vm.\$appendTo()等方法或指令更新）才触发 ready 钩子的。

》》 attached: vm.\$el 插入调用。必须是由指令或实例方法（如\$appendTo()）插入，直接操作 vm.\$el 不会触发这个钩子。

》》 detached: 在 vm.\$el 从 DOM 中删除时调用。

》》 beforeDestroy: 在开始销毁实例时调用。

》》 destroyed: 在实例被销毁之后调用。

目前，在风控系统项目开发中，只用上了 created()和 mounted()，实例代码：

```
export default {
  components: {...},
  data() {
    return {'id': ''...}
  },
  methods: {...},
  created() {
    this.$bus.$off('H_ruleSetInfo');
    this.$bus.$on('H_ruleSetInfo', (params)=>{
      this.render(params);
    });
    this.$bus.$off('H_RULESE_showLoading');
    this.$bus.$on('H_RULESE_showLoading', ()=>{
      this.showLoading();
    });
    this.$bus.$off('H_RULESE_hideLoading');
    this.$bus.$on('H_RULESE_hideLoading', ()=>{
      this.stopLoading();
    });
  },
  mounted() {
    var _this = this;
    _this.id = this.$route.query.id;
    _this.getInfo();
  }
}
</script>
```


5.6 如何理解组件？模块化？



下图是 vue 官网对组件的理解：

什么是组件？

组件 (Component) 是 Vue.js 最强大的功能之一。组件可以扩展 HTML 元素，封装可重用的代码。在较高层面上，组件是自定义元素，Vue.js 的编译器为它添加特殊功能。在有些情况下，组件也可以是原生 HTML 元素的形式，以 `is` 特性扩展。

-----Talk is cheap,show me the code-----

组件化——前端中“组件化”这个词，在 UI 这一层通常指“标签化”，也就是把大块的
业务界面，拆分成若干小块，然后进行组装。

代码示例：

```
1 <!--2017/3/22 接口测试联调 lh-->
2 <template>
3   <section>
4     <!--规则模型配置-->
5     <el-row...>
39
40     <div class='h-boxLoading' v-loading='boxLoading'>
41       <!--列表信息 改为: 组件内循环才满足接口和交互需要-->
42       <ruleSetList v-bind:my-data='result'></ruleSetList>
43       <!--配置弹窗-->
44       <ruleSettingDialog></ruleSettingDialog>
45       <!--编辑弹窗-->
46       <rulSetEditDialog></rulSetEditDialog>
47     </div>
48
49     <el-row class='h-scoreSet-btnBox'...>
55   </section>
56 </template>
57
58 <script>
59   | import ruleSetList from './ruleSet.list.vue' //引入规则模型配置--列表组件
60   import ruleSettingDialog from './ruleSet.settingDialog.vue' //引入配置规则弹窗
61   import rulSetEditDialog from './ruleSet.list.dialog.vue' //引入列表编辑弹窗
62   import productList from '../api/productList' //引入规则模型配置Api
63   export default {
64     components: {
65       ruleSetList,
66       ruleSettingDialog,
67       rulSetEditDialog
68     },
69     data() {
```

```
<ruleSetList v-bind:my-data="result"></ruleSetList>
import ruleSetList from './ruleSet.list.vue' //引入列表组件
```

看下 jsp 做静态页面生成的时候, 就可以利用 jsp 的<jsp:include page="xxx.jsp"/> 指令进行引入公共组件。

再或者, 跟 thinkphp 模板中 include 引入其他模板文件, 如:

```
<include file="Public/header" /> // 包含头部模版 header
```

模块化——所谓的模块化开发就是封装细节, 提供使用接口, 彼此之间互不影响, 每个模块都是实现某一特定的功能。如: CommonJS 和 AMD。

两者比较理解：

》》》 模块化：主要针对 Javascript，模块化开发的基础就是函数。

》》》 组件化：侧重 UI 层面，由 html、css、js 构成的独立文件。

6. Vue-router 路由的使用

官网 API : <https://router.vuejs.org/zh-cn/installation.html>

6.1 NPM 安装 : npm install vue-router

在如果在一个模块化工程中使用它，必须要通过 `Vue.use()` 明确地安装路由功能：

```
import Vue from 'vue'
import VueRouter from 'vue-router'
Vue.use(VueRouter)
```

下面贴上 demo 代码：



```
1 import Vue from 'vue'
2 import Router from 'vue-router'
3 import Hello from '@/components/Hello'
4 import demo from '../components/pages/table.vue'
5
6 Vue.use(Router)
7
8 export default new Router({
9   routes: [
10     {
11       path: '/',
12       name: 'Hello',
13       component: Hello
14     },
15     {
16       path: '/demo',
17       name: 'demo',
18       component: demo
19     }
20   ]
21 })
```

6.2 路由的跳转方式和路由传参：

》使用<router-link>标签，相当于<a>标签，其中 to 属性等于 href

```
<router-link to="/main" class="project_name">{{ProjectName}}</router-link>
```

》使用 js 跳转：

```
this.$router.push({ path: '/productList' });
```

》路由传参数跳转：

```

handleEditRule(row) {
  //var _data = this.$Utils.dataClone(row);
  var _data = {id:row.id};
  //this.$bus.$emit('ruleDialogShow', '编辑', _data, 'edit');
  this.$router.push({ path: '/rule/edit', query: { type:'edit', id:row.id}});
},

```

》》 目标页面接受参数方法：

```

},
mounted() {
  this.id = this.$route.query.id;
  this.getInfo();
}

```

导航钩子 :vue-router 提供的导航钩子主要用来拦截导航, 让它完成跳转或取消。

```

const router = new VueRouter({ ... })
router.beforeEach((to, from, next) => {
  // ...
})

```

每个钩子方法接收三个参数：

- to: Route: 即将要进入的目标 **路由对象**
- from: Route: 当前导航正要离开的路由
- next: Function: 一定要调用该方法来 resolve 这个钩子。执行效果依赖 next 方法的调用参数。
 - next(): 进行管道中的下一个钩子。如果全部钩子执行完了, 则导航的状态就是 confirmed (确认的)。
 - next(false): 中断当前的导航。如果浏览器的 URL 改变了 (可能是用户手动或者浏览器后退按钮), 那么 URL 地址会重置到 from 路由对应的地址。
 - next('/') 或者 next({ path: '/' }): 跳转到一个不同的地址。当前的导航被中断, 然后进行一个新的导航。

确保要调用 next 方法, 否则钩子就不会被 resolved。

使用场景：如判断用户是否登录跳转和权限页面跳转控制。参考 main.js

7. axios 基本用法及解决 ajax 跨越请求配置

Github :

7.1 用法简单，看文档~

Performing a GET request

```
// Make a request for a user with a given ID
axios.get('/user?ID=12345')
  .then(function (response) {
    console.log(response);
  })
  .catch(function (error) {
    console.log(error);
  });

// Optionally the request above could also be done as
axios.get('/user', {
  params: {
    ID: 12345
  }
})
  .then(function (response) {
    console.log(response);
  })
  .catch(function (error) {
    console.log(error);
  });
```

Performing a POST request

```
axios.post('/user', {
  firstName: 'Fred',
  lastName: 'Flintstone'
})
  .then(function (response) {
    console.log(response);
  })
  .catch(function (error) {
    console.log(error);
  });
```

7.2 解决 ajax 跨域问题

在 config 目录下的 index.js 配置 proxyTable 参数

```
dev: {
  env: require('./dev.env'),
  port: 8069,
  assetsSubDirectory: 'static',
  assetsPublicPath: '/',
  proxyTable: { //配置请求代理
    '/dlsys': {
      target: 'http://192.168.16.211:84',
      changeOrigin: true,
      pathRewrite: {
        '^/dlsys': '/dlsys'
      }
    },
    '/dlapi': {
      target: 'http://192.168.16.211:84',
      changeOrigin: true,
      pathRewrite: {
        '^/dlapi': '/dlapi'
      }
    },
    '/dlbiz': {
      target: 'http://192.168.16.211:84',
      changeOrigin: true,
      pathRewrite: {
        '^/dlbiz': '/dlbiz'
      }
    }
  },
  // CSS Sourcemaps off by default because relative paths are "buggy"
  // with this option, according to the CSS-Loader README
  // (https://github.com/webpack/css-loader#sourcemaps)
  // In our experience, they generally work as expected,
  // just be aware of this issue when enabling this option.
  cssSourceMap: true
}
```

上面的代理，其实 Apache 反向代理差不多

```
69 #
70 <VirtualHost *:85>
71     ServerAdmin www.lhw.com
72
73     DocumentRoot "E:/lhwBuild/md"
74     ServerName www.lhw.com
75
76     ErrorLog "logs/portal-front-error.log"
77     CustomLog "logs/portal-front-access.log" common
78
79     <Directory "E:/lhwBuild/md">
80         Options Indexes FollowSymLinks
81         AllowOverride None
82         Order allow,deny
83         Allow from all
84     </Directory>
85
86     ProxyPass /mingdao http://192.168.16.208:8182/mingdao
87
88     ProxyPassReverse /mingdao http://192.168.16.208:8182/mingdao
89
90
91 </VirtualHost>
```

others :

1、vue-resources 有 jsonp 的方法

//1.jsonp 不能发 post 请求，不管是否跨域，只要用 jsonp 方式就只能是 get，
因为本质是 script 方式加载的。

2、后端 response header 设置，

Access-Control-Allow-Origin: http://xxx.com，允许来自 xxx 的跨域

3、现代的浏览器，跨域优先考虑 Cross-Origin-Resource-Sharing. IE 11 就全面支持了。

8.Element UI 基本用法介绍

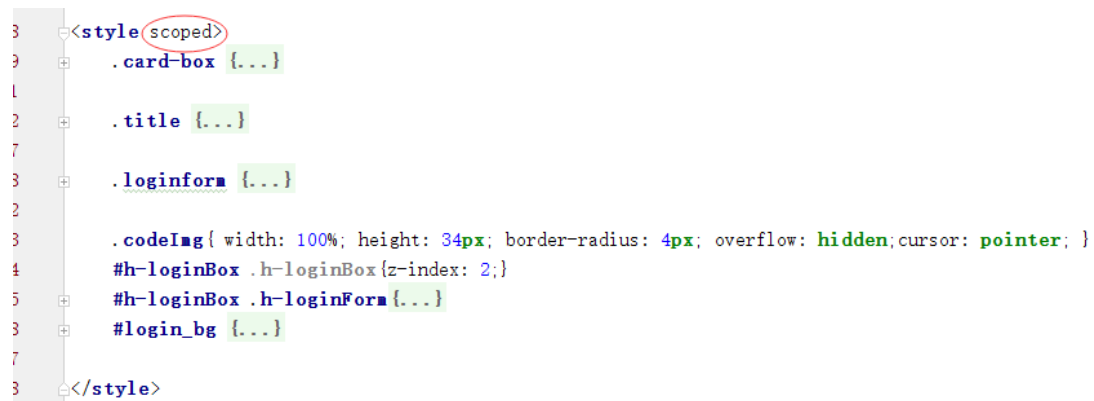
官网 API: <http://element.eleme.io/1.3/#/zh-CN/component/installation>

8.1 升级注意事项

见：<https://my.oschina.net/hgwn/blog/896780>

8.2 组件内样式

通常，组件中标签的样式是全局的，在使用第三方 UI 库（如 element），全局样式很可能影响 UI 库的样式。可通过添加 `scoped` 属性来是 `style` 中的样式只作用于当前组件，如下图：



```
3 <style scoped>
9   .card-box {...}
1
2   .title {...}
7
3   .loginform {...}
2
3   .codeImg{ width: 100%; height: 34px; border-radius: 4px; overflow: hidden; cursor: pointer; }
4   #h-loginBox .h-loginBox{z-index: 2;}
5   #h-loginBox .h-loginForm {...}
3   #login_bg {...}
7
3 </style>
```

注意：在有 `scope` 属性的 `style` 标签内导入其他样式（`@import 'xxxx.css'`），同样会受限作用于域，变为组件内样式。//复用程度较高的样式不建议这样使用

另外：在组件内样式中应避免使用元素选择器，性能大大降低。

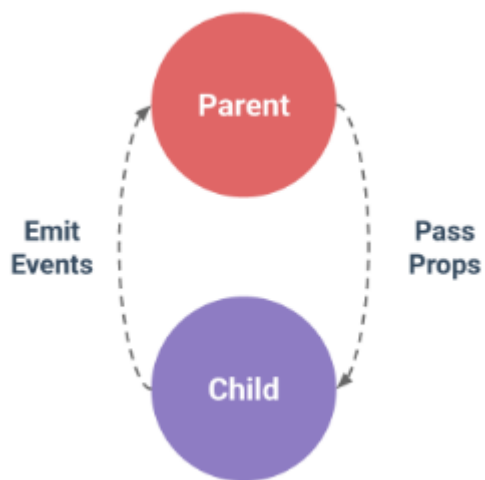
反过来：若是想覆盖掉 element 默认样式，则在组件去掉 `scoped` 属性 + `important`，当然建议在全局样式吧。

9. Vue.js 项目开发心得和遇到坑

9.1 如何解决组件之间通信问题

9.1.1 父组件和子组件之间通信

父子组件的关系可以总结为 props down, events up。父组件通过 props 向下传递数据给子组件，子组件通过 events 给父组件发送消息。



1) 使用 Prop 传递数据，见官网 demo：

组件实例的作用域是**孤立的**。这意味着不能 (也不应该) 在子组件的模板内直接引用父组件的数据。要让子组件使用父组件的数据，我们需要通过子组件的 `props` 选项。

子组件要显式地用 `props` 选项声明它期待获得的数据：

```
JS
Vue.component('child', {
  // 声明 props
  props: ['message'],
  // 就像 data 一样，prop 可以在模板内
  // 同样也可以在 vm 实例中像"this.message"这样使用
  template: '<span>{{ message }}</span>'
})
```

然后我们可以这样向它传入一个普通字符串：

```
HTML
<child message="hello!"></child>
```

结果：

hello!

2) 父组件调用子组件方法：通过\$refs

实例，见 demo 效果：



parent.vue 源码：

```
router\index.js x parent.vue x child1.vue x parent2.vue x child2.vue x
template div
1 <template>
2 <div>
3 <button v-on:click='clickParent'>点击</button>
4 <child1 ref='child1'></child1>
5 </div>
6 </template>
7
8 <script>
9   import Child1 from './child1';
10  export default {
11    name: 'parent',
12    components: {
13      child1: Child1
14    },
15    methods: {
16      clickParent() {
17        this.$refs.child1.handleParentClick("父组件调用子组件方法：通过$refs");
18      }
19    }
20  }
21 </script>
22
```

child1.vue 组件源码：



```
outer\index.js x  parent.vue x  child1.vue x  pare
script
<template>
  <div>
    <h2>child1—我是子组件</h2>
    <p>父组件调用子组件方法：通过$refs</p>
  </div>
</template>

<script>
  export default {
    name: 'child1',
    props: 'msg',
    methods: {
      handleParentClick(e) {
        console.info(e)
      }
    }
  }
</script>
```

3) 子组件向父组件通信

使用 `$on(eventName)` 监听事件

使用 `$emit(eventName)` 触发事件

官网 <https://cn.vuejs.org/v2/guide/components.html#自定义事件>

实例 demo 源码：

parent.vue 父组件

```
uter\index.js x parent.vue x child1.vue x parent2.vue x child2.vue x
script
<template>
  <div id='app'>
    <child msg='父组件传参数22' ref='hello' v-on:sengYzz='showMsg'>/child>
    <br>
    <input type='text' v-model='val'>
    <el-input v-model='val' name='name' style='width: 180px;'>/el-input>
    <button @click='father'>测试</button>
  </div>
</template>
<script>
  import child from './child2'
  export default {
    name: 'app',
    components: {
      child
    },
    data() { ... },
    methods: {
      father() { ... },
      showMsg(data) {
        console.log(data)
        //this.val = data;
      }
    }
  }
</script>
<style>
</style>
```

child2.vue 子组件

```
ue >
router\index.js x parent.vue x child1.vue x parent2.vue x child2.vue x
script
1 <template>
2   <div class="hello">
3     <h1>{{ msg }}</h1>
4     <!--<button @click="sengMsg" type="button">给父组件传值</button-->
5     <el-button type="primary" @click="sengMsg">给父组件传值</el-button>
6   </div>
7 </template>
8 <script>
9   export default {
10     name: 'hello',
11     props: ['msg'],
12     methods: {
13       test() {
14         console.log('我是通过ref调用的')
15       },
16       sengMsg() {
17         this.$emit('sengYzz', 'hello yzzting!')
18       }
19     }
20   }
21 </script>
22 <!-- Add "scoped" attribute to limit CSS to this component only -->
23 <style scoped>
24 </style>
```

9.1.2 非父子关系组件又该如何通信呢？

方法一、事件总线模式

可以理解为：订阅发布模式

场景：数据列表 list.vue 组件和编辑弹窗 model.vue 组件，单击列表某条数据，

然后在弹窗显示编辑。

看代码吧：

- 1)在 main.js 中，Vue 对象添加 prototype 原型属性\$bus


```
File watcher 'Babel' is available for this file. Description: 'Transpiles ECMAScript 6 code to ECMAScript 5'
7  import Vuex from 'vuex'
8  import router from './config/router'
9  import store from './vuex/store'
10
11
12  import './common/style.css' //自定义公共样式
13  import ajaxError from './api/ajaxError' //配置ajax请求错误插件
14  Vue.use(ajaxError)
15  import utils from './common/utils' //配置基础工具类
16  Vue.use(utils)
17
18  Vue.prototype.$bus = new Vue({}); //两个组件传递参数全局属性
19
20  Vue.use(ElementUI)
21  Vue.use(VueRouter)
22  Vue.use(Vuex)
23
```

2)被使用组件，在 created()钩子使用\$on 添加自定义事件回调函数，在
methods:{

//接受参数，自行处理函数

}, 见项目截图：

```
19  },
20  created() {
21    this.$bus.$off('ruleDialogShow');
22    this.$bus.$on('ruleDialogShow', (title, data) => {
23      this.ruleDialogShow(title, data);
24    });
25    this.HShow = this.RuleSetting;
26  },
27  methods: {
28    hideDialog() {...},
37    //显示dialog模态框
38    ruleDialogShow(title, data) {
39      this.editFormVisible = true;
40      this.editFormTtile = title;
41      if(data.id) {
42        this.reload(data);
43      }
44    },

```

3) 使用组件，用\$emit('自定义函数名称',参数)，见截图：

Html:

```

</span>
<span v-else>
  <el-button type='primary' size='small' @click='handleEdit(row)'>编辑</el-button>
  <el-button type='danger' size='small' @click='handleDel(row)'>删除</el-button>
</span>

```

JS :

```

    },
    methods: {
      handleEdit(row) {
        var _data = this.$Utils.dataClone(row);
        this.$bus.$emit('ruleDialogShow', '编辑规则', _data);
        //this.$router.push({path: '/productList/ruleSet/edit', query: {id
      },
    },

```

方法二、使用 vuex

应用场景：规则配置自定义是否显示或隐藏---在数字字典里面进行配置。

简单使用，不废话，看代码——talk is cheap,show me the code!

1) 在新建 vuex 目录下，创建 store.js，引入 vue 和 vuex，定义应用初始状态/所

需的 mutations，最后 export store 的实例，见截图：



```
1  /**
2   * Created by deelon on 2017/5/12.
3   */
4   import Vue from 'vue'
5   import Vuex from 'vuex'
6
7   Vue.use(Vuex)
8
9   // 应用初始状态
10  const state = {
11    ProjectName: 'DeeLon风控系统',
12    RuleSetting: false, //设置是否显示规则配置
13  }
14
15  // 定义所需的 mutations
16  const mutations = {
17
18    // 更新项目名称
19    UPDATE_PROJECTNAME(state, name) {
20      state.ProjectName = name;
21    },
22    // 更新规则配置
23    UPDATE_RULESETTING(state, val) {
24      state.RuleSetting = val;
25      //console.log('store...:' + state.RuleSetting);
26    }
27  }
28
29  // 创建 store 实例
30  export default new Vuex.Store({
31    state,
32    mutations
33  })
```

2) 在 main.js 引入上面创建的 store.js 组件，见截图：

```
router.js ×  main.js ×  store.js ×
File watcher 'Babel' is available for this file. Description: 'Transpiles ECMAScript 6 code to ECMAScript 5'

1  import babelpolyfill from 'babel-polyfill'
2  import Vue from 'vue'
3  import App from './App'
4  import ElementUI from 'element-ui'
5  import 'element-ui/lib/theme-default/index.css'
6  import VueRouter from 'vue-router'
7  import Vuex from 'vuex'
8  import router from './config/router'
9  import store from './vuex/store'
10
11
12  import './common/style.css' //自定义公共样式
13  import ajaxError from './api/ajaxError' //配置ajax请求错误插件
14  Vue.use(ajaxError)
15  import utils from './common/utils' //配置基础工具类
16  Vue.use(utils)
17
18  Vue.prototype.$bus = new Vue({}); //两个组件传递参数全局属性
19
20  Vue.use(ElementUI)
21  Vue.use(VueRouter)
22  Vue.use(Vuex)
23
24
25  new Vue({
26    el: '#app',
27    template: '<App/>',
28    router,
29    store,
30    components: { App }
31    //render: h => h(Login)
32  }).$mount('#app')
33
34  //router.replace('/login')
35  //
```

3) 更新 state 状态

如在 Home.vue 公共组件，通过 ajax 调用 API 接口获取返回的用户自定义的信息，并更新到 store 实例的状态。

Home.vue 组件，部分源码：

- 1、`import { mapState } from 'vuex'`
- 2、`this.$store.commit('UPDATE_RULESETTING', _val);`

代码截图：

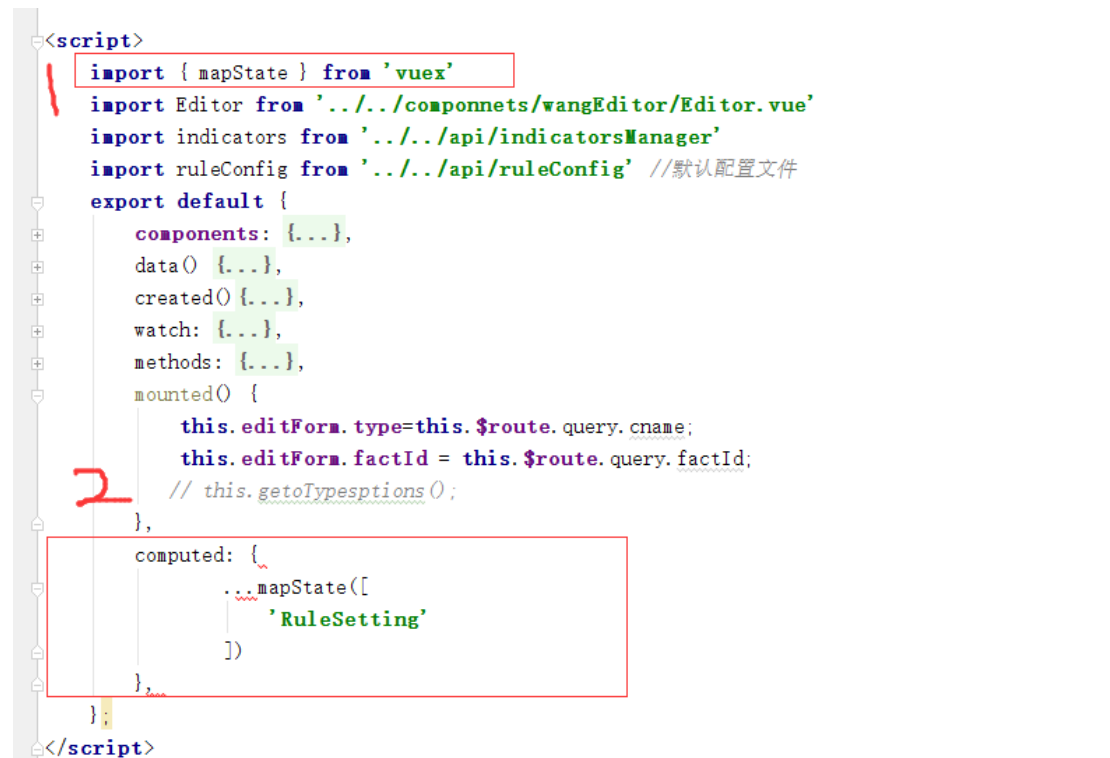


4)使用状态，见部分源码：

Html 部分：

```
<el-row v-show="RuleSetting">
  <el-col :span="24">
    <el-form-item label="指标逻辑" prop="expression">
      <v-editor :input-content="inputContent" :upload-url="uploadUrl" v-
model="editForm.expression" name="expression"></v-editor>
    </el-form-item>
  </el-col>
</el-row>
```

JS: 见下图



```
<script>
import { mapState } from 'vuex'
import Editor from '../componnets/wangEditor/Editor.vue'
import indicators from '../api/indicatorsManager'
import ruleConfig from '../api/ruleConfig' //默认配置文件
export default {
  components: {...},
  data() {...},
  created() {...},
  watch: {...},
  methods: {...},
  mounted() {
    this.editForm.type=this.$route.query.cname;
    this.editForm.factId = this.$route.query.factId;
    // this.getTypesptions();
  },
  computed: {
    ...mapState([
      'RuleSetting'
    ])
  },
};
</script>
```

关于 vuex 具体问题见 9.2 讨论

方法三、VUE2.4.0 版本提供了一种新的解决方案

大概思路：首先组件支持 inheritAttrs 的选项，其次需要用到实例属性 \$attrs。

9.3 vue 开发调试

一、github 下载安装 vue-devtools

1.github 下载地址：<https://github.com/vuejs/vue-devtools>

有 Git 的同学直接 `git clone https://github.com/vuejs/vue-devtools`

2.下载完成之后，打开 cmd 进入 vue-devtools 文件夹，把依赖装好 `npm install` 之后再进行 `npm run build`

3.然后打开 `shells>chrome>src>manifest.json`

把里面的"persistent": false 改为 true

二、配置 chrome 浏览器

打开 chrome

1.打开里面的设置 > 点击扩展程序 > 点击开发者模式

2.再点击加载已解压的扩展程序, 然后把 shells>chrome 这个文件夹放入就 ok 了

