主要写一些 官方手册 上没有写，但是实际开发中会遇到的问题，需要一定知识基础。

**涉及技术栈**

* CLI: Vue-CLI
* UI: Element
* HTML: Pug(Jade)
* CSS: Less
* JavaScript: ES6

正文：

**polyfill 与 transform-runtime**

首先，vue-cli 为我们自动添加了 babel-plugin-transform-runtime 这个插件，该插件多数情况下都运作正常，可以转换大部分 ES6 语法。

但是，存在如下两个问题：

1. 异步加载组件时，会产生 polyfill 代码冗余
2. 不支持对全局函数与实例方法的 polyfill

两个问题的原因均归因于 babel-plugin-transform-runtime 采用了沙箱机制来编译我们的代码（即：不修改宿主环境的内置对象）。

由于异步组件最终会被编译为一个单独的文件，所以即使多个组件中使用了同一个新特性（例如：Object.keys()），那么在每个编译后的文件中都会有一份该新特性的 polyfill 拷贝。如果项目较小可以考虑不使用异步加载，但是首屏的压力会比较大。

不支持全局函数（如：Promise、Set、Map），Set 跟 Map 这两种数据结构应该大家用的也不多，影响较小。但是 Promise 影响可能就比较大了。

不支持实例方法（如：'abc'.include('b')、['1', '2', '3'].find((n) => n 等等），这个限制几乎废掉了大部分字符串和一半左右数组的新特性。

一般情况下 babel-plugin-transform-runtime 能满足大部分的需求，当不满足需求时，推荐使用完整的 babel-polyfill。

**替换 babel-polyfill**

首先，从项目中移除 babel-plugin-transform-runtime

卸载该依赖：

npm un babel-plugin-transform-runtime -D

修改 babel 配置文件

// .babelrc

{

  //...

  "plugins": [

    // - "transform-runtime"

  ]

  //...

}

然后，安装 babel-polyfill 依赖：

npm i babel-polyfill -D

最后，在入口文件中导入

// src/main.js

import 'babel-polyfill'

**ES6 import 引用问题**

在 ES6 中，模块系统的导入与导出采用的是引用导出与导入（非简单数据类型），也就是说，如果在一个模块中定义了一个对象并导出，在其他模块中导入使用时，导入的其实是一个变量引用（指针），如果修改了对象中的属性，会影响到其他模块的使用。

通常情况下，系统体量不大时，我们可以使用 JSON.parse(JSON.stringify(str)) 简单粗暴地来生成一个全新的深度拷贝的 数据对象。不过当组件较多、数据对象复用程度较高时，很明显会产生性能问题，这时我们可以考虑使用 Immutable.js。

鉴于这个原因，进行复杂数据类型的导出时，需要注意多个组件导入同一个数据对象时修改数据后可能产生的问题。

此外，模块定义变量或函数时即便使用 let 而不是 const，在导入使用时都会变成只读，不能重新赋值，效果等同于用 const 声明。

**在 Vue 中使用 Pug 与 Less**

**安装依赖**

Vue 中使用 vue-loader 根据 lang 属性自动判断所需要的 loader，所以不用额外配置 Loader，但是需要手动安装相关依赖：

npm i pug -D

npm i less-loader -D

还是相当方便的，不用手动修改 webpack 的配置文件添加 loader 就可以使用了

使用 pug 还是 pug-loader？sass 两种语法的 loader 如何设置？

— 请参考 预处理器 · vue-loader（https://vue-loader.vuejs.org/zh-cn/configurations/pre-processors.html）

**使用**

.action(v-if='hasRight') ul li 编辑 li 删除

**定义全局函数或变量**

许多时候我们需要定义一些全局函数或变量，来处理一些频繁的操作（这里拿 AJAX 的异常处理举例说明）。但是在 Vue 中，每一个单文件组件都有一个独立的上下文（this）。通常在异常处理中，需要在视图上有所体现，这个时候我们就需要访问 this 对象，但是全局函数的上下文通常是 window，这时候就需要一些特殊处理了。

简单粗暴型

最简单的方法就是直接在 window 对象上定义一个全局方法，在组件内使用的时候用 bind、call 或 apply 来改变上下文。

定义一个全局异常处理方法：

// errHandler.js

window.errHandler = function () { // 不能使用箭头函数

  if (err.code && err.code !== 200) {

    this.$store.commit('err', true)

  } else {

    // ...

  }

}

在入口文件中导入：

// src/main.js

import 'errHandler.js'

在组件中使用：

// xxx.vue

export default {

  created () {

    this.errHandler = window.errHandler.bind(this)

  },

  method: {

    getXXX () {

      this.$http.get('xxx/xx').then(({ body: result }) => {

        if (result.code === 200) {

          // ...

        } else {

          this.errHandler(result)

        }

      }).catch(this.errHandler)

    }

  }

}

优雅安全型

在大型多人协作的项目中，污染 window 对象还是不太妥当的。特别是一些比较有个人特色的全局方法（可能在你写的组件中几乎处处用到，但是对于其他人来说可能并不需要）。这时候推荐写一个模块，更优雅安全，也比较自然，唯一不足之处就是每个需要使用该函数或方法的组件都需要进行导入。

使用方法与前一种大同小异，就不多作介绍了。￣ω￣=

**Moment.JS 与 Webpack**

在使用 Moment.js 遇到一些问题，发现最终打包的文件中将 Moment.js 的全部语言包都打包了，导致最终文件徒然增加 100+kB。查了一下，发现可能是 webpack 打包或是 Moment.js 资源引用问题（?），目前该问题还未被妥善处理，需要通过一些 trick 来解决这个问题。

在 webpack 的生产配置文件中的 plugins 字段中添加一个插件，使用内置的方法类 ContextReplacementPlugin 过滤掉 Moment.js 中那些用不到的语言包：

// build/webpack.prod.conf.js

new webpack.ContextReplacementPlugin(/moment[\\/]locale$/, /^\.\/(zh-cn)$/)

解决方案采自 oleg-nogin@webpack/webpack#3128。

问题讨论详见 GitHub Issue: moment/moment#2373、webpack/webpack#3128。

**自定义路径别名**

可能有些人注意到了，在 vue-cli 生成的模板中在导入组件时使用了这样的语法：

import Index from '@/components/Index'

这个 @ 是什么东西？后来改配置文件的时候发现这个是 webpack 的配置选项之一：路径别名。

我们也可以在基础配置文件中添加自己的路径别名，比如下面这个就把 ~ 设置为路径 src/components 的别名：

// build/webpack.base.js

{

  resolve: {

    extensions: ['.js', '.vue', '.json'],

    alias: {

      'vue$': 'vue/dist/vue.esm.js',

      '@': resolve('src'),

      '~': resolve('src/components')

    }

  }

}

然后我们导入组件的时候就可以这样写：

// import YourComponent from 'YourComponent'

// import YourComponent from './YourComponent'

// import YourComponent from '../YourComponent'

// import YourComponent from '/src/components/YourComponent'

import YourComponent from '~/YourComponent'

既解决了路径过长的麻烦，又解决了相对路径的烦恼，方便很多吧！ヾ(ﾟ∀ﾟゞ)

**CSS 作用域与模块**

**组件内样式**

通常，组件中 标签里的样式是全局的，在使用第三方 UI 库（如：Element）时，全局样式很可能影响 UI 库的样式。我们可以通过添加 scoped 属性来使 style 中的样式只作用于当前组件：

<style lang="less" scoped>

*@import 'other.less';*

  .title {

    font-size: 1.2rem;

  }

</style>

在有 scoped 属性的 style 标签内导入其他样式，同样会受限于作用域，变为组件内样式。复用程度较高的样式不建议这样使用。

另，在组件内样式中应避免使用元素选择器，原因在于元素选择器与属性选择器组合时，性能会大大降低。

— 两种组合选择器的测试：classes selector，elements selector

**导入样式**

相对于 style 使用 scoped 属性时的组件内样式，有时候我们也需要添加一些全局样式。当然我们可以用没有 scoped 属性的 style 来写全局样式。但是相比较，更推荐下面这种写法：

/\* 单独的全局样式文件 \*/

/\* style-global.less \*/

body {

  font-size: 10px;

}

.title {

  font-size: 1.4rem;

  font-weight: bolder;

}

然后在入口文件中导入全局样式：

// src/main.js

import 'style-global.less'/pre>

**获取表单控件值**

通常我们可以直接使用 v-model 将表单控件与数据进行绑定，但是有时候我们也会需要在用户输入的时候获取当前值（比如：实时验证当前输入控件内容的有效性）。

这时我们可以使用@input或 @change事件绑定我们自己的处理函数，并传入 $event象以获取当前控件的输入值：

<input type="text"/>

change (e) {

  let curVal = e.target.value

  if (/^\d+$/.test(curVal)) {

    this.num = +curVal

  } else {

    console.error('%s is not a number!', curVal)

  }

}

当然，如果 UI 框架采用 Element 会更简单，它的事件回调会直接传入当前值。

**v-for 的使用 tips**

v-for 指令很强大，它不仅可以用来遍历数组、对象，甚至可以遍历一个数字或字符串。

基本语法就不讲了，这里讲个小 tips：

索引值

在使用 v-for 根据对象或数组生成 DOM 时，有时候需要知道当前的索引。我们可以这样：

* {{ key }} – {{ item }}

但是，在遍历数字的时候需要注意，数字的 value 是从 1 开始，而 key 是从 0 开始：

* {{ k }}-{{ v }}

2.2.0+ 的版本里，当在组件中使用 v-for 时，key 现在是必须的。

模板的唯一根节点

与 JSX 相同，组件中的模板只能有一个根节点，即下面这种写法是 错误 的：

**项目路径配置**

由于 vue-cli 配置的项目提供了一个内置的静态服务器，在开发阶段基本不会有什么问题。但是，当我们把代码放到服务器上时，经常会遇到静态资源引用错误，导致界面一片空白的问题。

这是由于 vue-cli 默认配置的 webpack 是以站点根目录引用的文件，然而有时候我们可能需要把项目部署到子目录中。

我们可以通过 config/index.js 来修改文件引用的相对路径：

build.assetsSubDirectory: 'static'

  build.assetsPublicPath: '/'

  dev.assetsSubDirectory: 'static'

  dev.assetsPublicPath: '/'

我们可以看到导出对象中 build 与 dev 均有 assetsSubDirectory、assetsPublicPath 这两个属性。

其中 assetsSubDirectory 指静态资源文件夹，也就是打包后的　js、css、图片等文件所放置的文件夹，这个默认一般不会有问题。

assetsPublicPath 指静态资源的引用路径，默认配置为 /，即网站根目录，与 assetsSubDirectory 组合起来就是完整的静态资源引用路径 /static。

写到这里解决方法已经很明显了，只要把根目录改为相对目录就好了：

 build.assetsSubDirectory: 'static'

  build.assetsPublicPath: './'

没错！就是一个 . 的问题。ㄟ( ▔, ▔ )ㄏ