初识VUE

- **什么是Vue**：*使用前端开发的一种前端框架,简单小巧大小仅为17KB（压缩后）*
  - **渐进式框架：简单来说如果你想用Vue的一些功能就直接使用，如果不想使用其他功能就不用，vue不强求你使用所有功能，未要求你把所有功能都完善了**
  - **MVVM框架：类似于后端MVC框架，vue采用了M-V-VM模式对前端项目管理,当视图层发生变化时，视图模型就会自动更新所发生的变化,反之亦然,View-ViewModel之间通过双向绑定，可看下图**

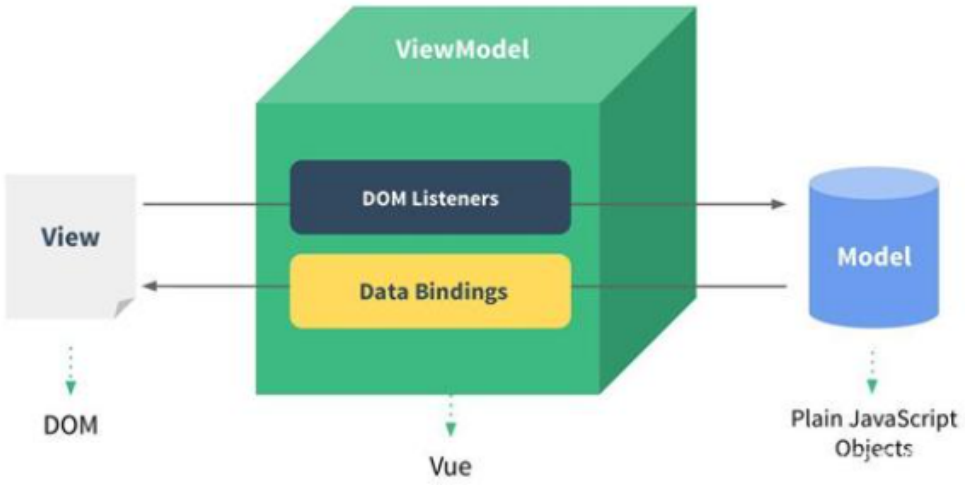


图1-1 MMVM1

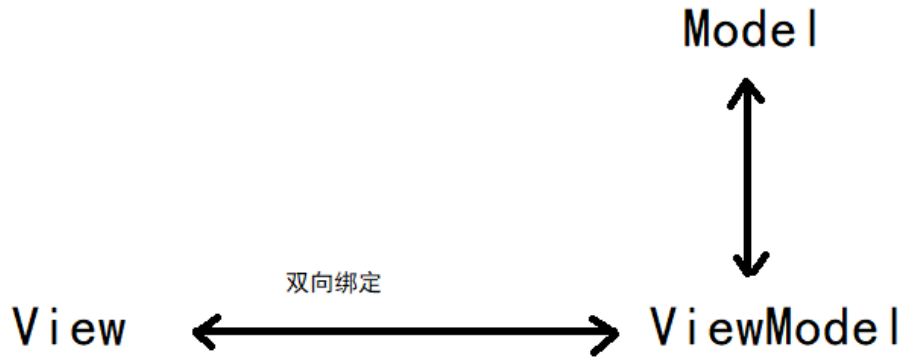


图1-2 MVVM2

- **Vue能做什么**
  - ***解耦视图与数据加载***
  - ***可复用组件***
  - ***前端路由***
  - ***状态管理***
  - ***虚拟DOM***
- **vue怎么用**
- **简单使用：只需要引入vue.js即可使用（基础）**
  - **前端工程化使用：webpack（进阶）**

Vue注意事项

1.不应该使用箭头函数来定义一个生命周期方法
2.不应该使用箭头函数来定义 method 函数
3.不应该使用箭头函数来定义计算属性函数
4.不应该对 data 属性使用箭头函数
5.不应该使用箭头函数来定义 watcher 函数

数据绑定与第一个Vue应用

- **Hello Word**
  - 查看vue/Helloword.html案例
- **实例与数据**
  - 查看vue/databind.html案例
- **生命周期**
  - 查看vue/life.html案例　ps：不能直接使用当前的app.a
- **插值表达式**
  - 查看vue/express-value.html案例
  - 简单三元,简单的运算：查看vue/compute.html案例
  - 一些vue白名单的函数及对象可在插值表达式中使用
- **过滤器**
- 查看vue/filter.html案例　　（可进行串联）　=> {{|filterA|filterB|.....}}
- **指令与事件**
  - ***指令：*** 查看vue/filter.html案例，vue/command.html案例
    - v-html（非纯为本）:直接写入html（可能导致XSS攻击）
    - v-pre：可跳过这个元素和子元素的编译过程
    - v-if,v-else-if,v-else:vue真正的条件渲染，他会根据表达式内容适当的销毁,重建,绑定事件与子组件
    - v-bind: 用于绑定标签属性　　语法糖为　:属性名
    - v-model:用于表单内容的一些标签,select,textarea等
    - v-for:用于循环生成标签值　(book,index) in books
    - v-cloak:不需要编写内容，当页面网速过慢插值表达式未编译完我们需要进行插值表达式隐藏则使用V-cloak,一般配合css使用如：　　[v-cloak]{display:none;}
    - v-show：只是简单CSS属性切换,无论条件真与否,都会被编译,v-if使用在条件不经常改变的场景，v-if开销比v-show大
    - v-once：数据只渲染一次,之后改变也不会变化
    - v-on：用于绑定事件　　语法糖为　　@click,@change
  - ***事件：***
    - **load,unload,abort,error,select,resize,scroll事件**
    - **焦点事件**
    - **鼠标与滚轮事件**
    - **键盘和文本事件**

Vue计算属性

- **可以对数据进行一些简单的清洗计算**
- **基于双方绑定当插值表达式的值改变的时候，计算属性计算结果也会跟着变化**
- **可以是代码看着不那么臃肿,利于维护**

V-bind及class与style绑定

- **可用于绑定class属性类**
- **可用于绑定style样式**

  参考案例:vue/bind.html ps:属性或样式使用{}以键值对　进行书写

Vue内置指令

- **指令演示：v-cloak,v-if,v-else-if,v-once,v-for,v-show**
- **参考案例：vue/innerOrder.html** **
- **数组更新：基于vue的双向绑定的特点,当发现数据变化时会立即更新当前的视图，因此数组变化时vue提供一些方法用于观测数组变异方法如下**
  - push,pop，shift，unshift，splice，sort，reverse（是因为这些方法会改变原数组数据就变化了）
  - filter，concat，slice（不会改变原数组）
  - 参考案例:vue/arrayUpdate.html
- **vue事件修饰符**

  用法@事件.修饰符
  - @事件.stop
  - @事件.prevent
  - @事件.capture
  - @事件.self
  - @事件.once
  - @事件.enter
  - @事件.tab
  - @事件.delete
  - @事件.esc
  - @事件.space
  - @事件.up,@事件.down
  - @事件.left,@事件.right
  - @事件.ctrl,@事件.alt,@事件.shift,@事件.meta(window是窗口键,mac是Command键)

Vue表单与V-model(最好在你使用表单使用)

- **v-model修饰符**
  - v-model.lazy,v-model.number，v-model.trim
- **绑定值：v-model只能绑定静态字符串,业务中需要绑定动态数据使用v-bind**
- **选择列表当v-model='selected'时，首先会取value属性值,若没有value属性则取option的text**
  - 参考案例：vue/model.html

Vue组件

为什么使用组件?

- 组件可以让代码具有很高的复用率
- 简化和提高我们的开发效率，是组件能够通信
- 组件渲染方式划分为每个小模块，方便维护和管理

组件使用

- Vue.component('组件名',{组件编写})
- 使用组件必须在实例注册前,HTML标签中一些搭配使用的标签不能直接自定义组件如table,ul,ol,select标签，必须使用is='自定义组件'才可挂载
- **局部注册与全局注册 参考案例：vue/partAndGlobal.html**
- **props传递数据 （支持驼峰命令） 参考案例：vue/props.html，不可以自定义组件中含有自定义组件**
- **单项数据流 Vue.component('组件名',{props:[]|{},template:'标签',data:function(){},computed:{}}),data就是单项数据流，我们只需要关注data中的数据即可,内部不在操作父级数据**
- **数据验证 使用props对象进行实现{ propA:Number，propB：[Number,String]，PropC：{type：类型，require：boolean,defalut：如果类型是boolean则为true,如果为对象或数组,返回必须是函数}，validator：function（value）{return xxx}}**
- **组件通信 :组件的通信不只是父-子,归纳可为下图所示**

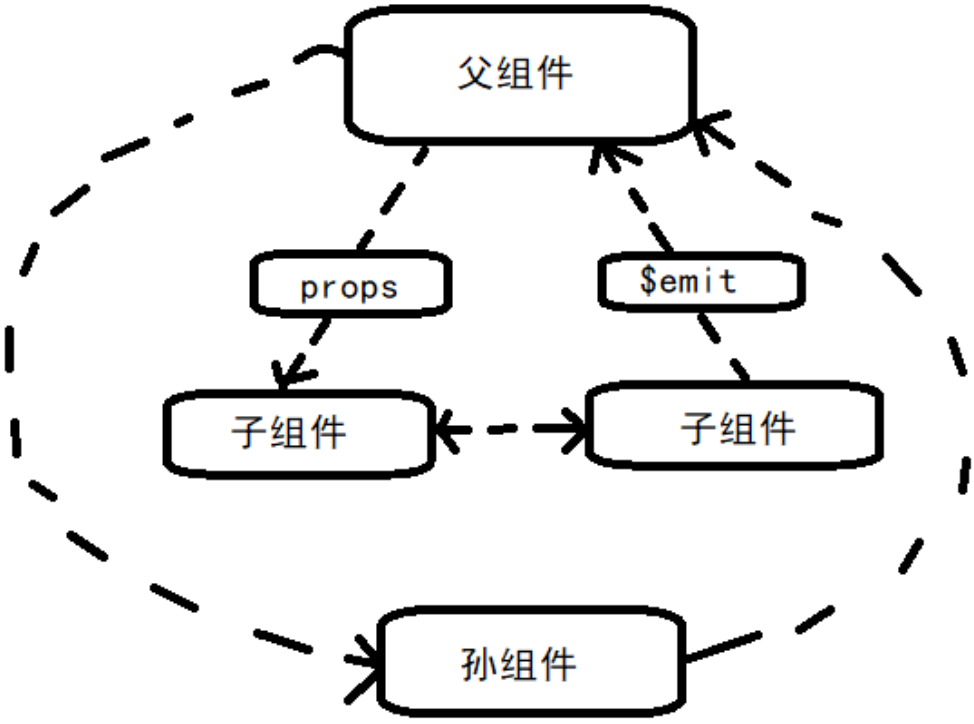


图1-3 组件通信

  - 自定义事件通信（兄弟,祖孙,子-父通信）：子组件dispatchEvent（$emit）,父组件addEventListener($on)这是2.x的版本,vue1.X为$dispatch,$broadcast;
  - 父链通信：$parent可以直接修改父链信息 参考案例:vue/communication1.html
  - 组件相互嵌套：定义全局注册最简单,写完直接用即可，局部注册在当前vue实例components中在嵌套一个components即可 参考案例:vue/communication1.html
  - 子索引组件:参考案例:vue/communication1.html
- slot内容分发
  - 单个slot分发内容
  - 具名slot：给slot指定name可与单slot共存
  - 作用域卡槽scope：也可以拥有具名 ,子组件中

组件高级用法

- **递归组件：只需要在组件中指定name属性即可**
- **内联模板：组件标签加入inline-template，这样组件就把他的内容当做模板而不是当做内容分发**

- **动态组件：vue特殊的元素用于挂载不同组件,,is也可以是对象**
- **异步组件：定义组件之后传入一个function（resolve,reject）{} resolve为回调函数,reject指示加载失败**
- **$nextTick:vue有一个重要的概念，vue不会直接改变Dom,vue会创建一个异步更新队列,当在同一事件循环中发生的所有数据改变,在缓冲时会清除重复数据,避免不必要的计算和dom操作。$nextTick可以知道什么时候dom更新完毕。 案例：vue/nextTick.html**
- X-Template：为了避免没有使用webpack，gulp等工具所带来的编写组件困扰,我们这里提供 ;最后在组件注册中template属性绑定即可

  例： id="component-a" => template:'#component-a'即可使用
- 手动挂载实例：Vue.extend 和$mount手动挂载
- 实战：用已经学知识实现一个输入框功能,需求如下：数字输入框只能输入数字,有两个按钮用以控制+ -,并且可以设置初始值,最大值,最小值 案例：vue/case1.html

Vue自定义指令

- **全局注册与局部注册** Vue.directive('指令名称',{指令选项}) directives:{ 指令名称：{指令选项} }
- **指令选项（自定义指令选项）：bind，unbind，inserted，update，componentUpdated**
  - **bind:每当指令绑定到元素上时，会立即执行这个bind函数，只执行一次**
  - **inserted:表示元素插入到DOM中时，会执行inserted函数，只触发一次，el表示被绑定的那个原生js对象**
  - **update:当VNode更新时会执行updated，可能触发多次**
  - 钩子函数参数（el,binding,vnode,Oldnode）

Vue之虚拟DOM：Render函数

学习须知：学习render函数必须熟练前面的知识点,尤其是组件部分

- **createElement用法：**
  - **第一个参数：可以是html标签|组件选项|一个函数 {String|Object|Function}**
  - **第二个参数：一个Object，用于指定该标签属性 {attrs,on,class,style,domProps,props,nativeOn,directives:[],scopeSlots:{default:function(){}}}**
  - **第三个参数:可为String|Array ,如果是String则是内部标签内容**
- **javascript代替模板功能：**
  - **因为Virtual Dom本身就是轻量级的JavaScript,所以在Render中直接使用javascript语法就行**
- **函数化组件：**
  - **组件模板中提供一个functional字段,当它为True的时候,则表明是一个函数化组件,意思是没有data,和this上下文,但是提供Render第二个参数为context，里面包括了所有需要的操作但是是函数式的操作,如以前this.$slots.default函数化组件之后是context.children**
  - **组件需要的data,props,slots,children,parent都是通过这个context上下文传递的**
- **Render函数之JSX插件：需要前端工程化webpack**

拓展：钩子函数

您可以通过为生命周期挂钩添加前缀“ on”来访问组件的生命周期挂钩。

下表包含如何在setup （） 内部调用生命周期挂钩：

选件API	钩在里面 `setup`
`beforeCreate`	并不需要*
`created`	并不需要*
`beforeMount`	`onBeforeMount`
`mounted`	`onMounted`
`beforeUpdate`	`onBeforeUpdate`
`updated`	`onUpdated`
`beforeUnmount`	`onBeforeUnmount`
`unmounted`	`onUnmounted`
`errorCaptured`	`onErrorCaptured`
`renderTracked`	`onRenderTracked`
`renderTriggered`	`onRenderTriggered`

Vue之Webpack

什么是webpack?

- webpack是一个进行模块化开发的工具，在ES6之前我们想进行模块化开发必须借助其他工具。webpack优势在于让开发者进行模块化开发，webpack会处理模块化中的各种关系,不仅仅是js，CSS、图片,json等等都被当做模块化处理。当我们想使用的使用import即可，使我们更加专注于开发业务，以及多人协同开发。

前排提醒：我们如果想使用webpack必须掌握Npm，node.js,ES6的相关语法和用法

相关文档使用可以查看：https://webpack.docschina.org/concepts/

webpack.config.js配置项

- **入口（entry）**：重点加粗部分
  - **入口起点(entry point)**指示 webpack 应该使用哪个模块，来作为构建其内部 依赖图(dependency graph) 的开始。进入入口起点后，webpack 会找出有哪些模块和库是入口起点（直接和间接）依赖的。

    默认值是 `./src/index.js`，但你可以通过在 webpack configuration 中配置 `entry` 属性，来指定一个（或多个）不同的入口起点。例如：

    webpack.config.js

```
module.exports = {
  entry: './path/to/my/entry/file.js',
};
```

- **输出（output）**
  - output 属性告诉 webpack 在哪里输出它所创建的 *bundle*，以及如何命名这些文件。主要输出文件的默认值是 `./dist/main.js`，其他生成文件默认放置在 `./dist` 文件夹中。

    你可以通过在配置中指定一个 `output` 字段，来配置这些处理过程：

    webpack.config.js

```
const path = require('path');  //导入 node.js核心模块，用于操作处理文件和目录路径的实用程序

module.exports = {
  entry: './path/to/my/entry/file.js',
  output: {
    path: path.resolve(__dirname, 'dist'),   //使用了path模块用以解析为绝对路径
    filename: 'my-first-webpack.bundle.js',
  },
};
```

    在上面的示例中，我们通过 `output.filename` 和 `output.path` 属性，来告诉 webpack bundle 的名称，以及我们想要 bundle 生成(emit)到哪里。可能你想要了解在代码最上面导入的 path 模块是什么，它是一个 Node.js 核心模块，用于操作文件路径。

- **loader**
  - **webpack 只能理解 JavaScript 和 JSON 文件**，这是 webpack **开箱可用的自带能力。loader** 让 webpack 能够去处理其他类型的文件，并将它们转换为有效 模块，以供应用程序使用，以及被添加到依赖图中。
  - 在更高层面，在 webpack 的配置中，**loader** 有两个属性：
    1. `test` 属性，识别出哪些文件会被转换。
    2. `use` 属性，定义出在进行转换时，应该使用哪个 loader。
  - **webpack.config.js**

```
const path = require('path');

module.exports = {
  output: {
    filename: 'my-first-webpack.bundle.js',
  },
  module: {
    rules: [{ test: /\.txt$/, use: 'raw-loader' }],
  },
};
```

以上配置中，对一个单独的 module 对象定义了 `rules` 属性，里面包含两个必须属性：`test` 和 `use`。这告诉 webpack 编译器(compiler) 如下信息：

> “嘿，webpack 编译器，当你碰到「在 `require()` / `import` 语句中被解析为 '.txt' 的路径」时，在你对它打包之前，先 **use(使用)** `raw-loader` 转换一下。

- **插件（plugins）**
  - loader 用于转换某些类型的模块，而插件则可以用于执行范围更广的任务。包括：打包优化，资源管理，注入环境变量。
  - 想要使用一个插件，你只需要 `require()` 它，然后把它添加到 `plugins` 数组中。多数插件可以通过选项(option)自定义。你也可以在一个配置文件中因为不同目的而多次使用同一个插件，这时需要通过使用 `new` 操作符来创建一个插件实例。
  - **webpack.config.js**

    ```
    const HtmlWebpackPlugin = require('html-webpack-plugin'); // 通过 npm 安装
    const webpack = require('webpack'); // 用于访问内置插件

    module.exports = {
      module: {
        rules: [{ test: /\.txt$/, use: 'raw-loader' }],
      },
      plugins: [new HtmlWebpackPlugin({ template: './src/index.html' })],
    };
    ```

    在上面的示例中，`html-webpack-plugin` 为应用程序生成一个 HTML 文件，并自动注入所有生成的 bundle。
- **模式（mode）**
  - 通过选择 `development`, `production` 或 `none` 之中的一个，来设置 `mode` 参数，你可以启用 webpack 内置在相应环境下的优化。其默认值为 `production`。

    ```
    module.exports = {
      mode: 'production',
    };
    ```
- **浏览器兼容性（browser compatibility）**
  - webpack 支持所有符合 ES5 标准 的浏览器（不支持 IE8 及以下版本）。webpack 的 `import()` 和 `require.ensure()` 需要 `Promise`。如果你想要支持旧版本浏览器，在使用这些表达式之前，还需要 提前加载 polyfill。
- **环境（environment）**
  - **webpack 5 运行于 Node.js v10.13.0+ 的版本。**

webpack使用(有些包可自行根据需求选择性安装)

- npm init
- npm i webpack@4.46.0 --save-dev
- npm i webpack-dev-server@3.11.1 --save-dev
- npm i webpack-cli@3.3.12 --save-dev
- npm i jquery --save-dev
- npm i html-webpack-plugin@4.5.2 --save-dev
- cnpm css-loader style-loader --save-dev
- npm i extract-text-webpack-plugin --save-dev（这种方式不支持webpack4及以上，需要使用npm install --save-dev mini-css-extract-plugin即可
- npm i url-loader file-loader --save-dev

以上是不进行vue开发的前端工程化的一些包依赖,相关的使用请参考npmjs

vue开发包依赖

- cnpm i babel-loader @babel/core @babel/preset-env -D
- cnpm i @babel/runtime @babel/plugin-transform-runtime -D
- cnpm i @babel/plugin-proposal-class-properties -D
- cnpm i vue --save
- cnpm i vue-loader --save-dev
- cnpm i vue-style-loader --save-dev
- cnpm i vue-template-compiler --save-dev
- cnpm i vue-hot-reload-api --save-dev
- cnpm i extract-text-webpack-plugin@next -d

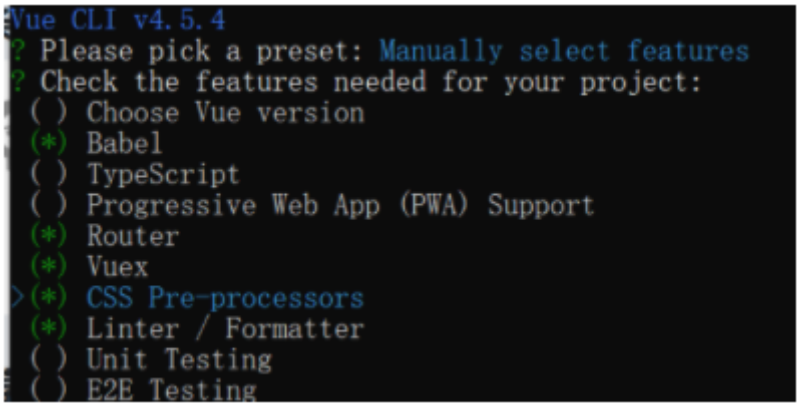
Vue Cli脚手架

- cnpm i @vue/cli -g
- **创建项目:**
  - 3.创建项目

```
vue create 项目名称(貌似不能含有大写字母)
```

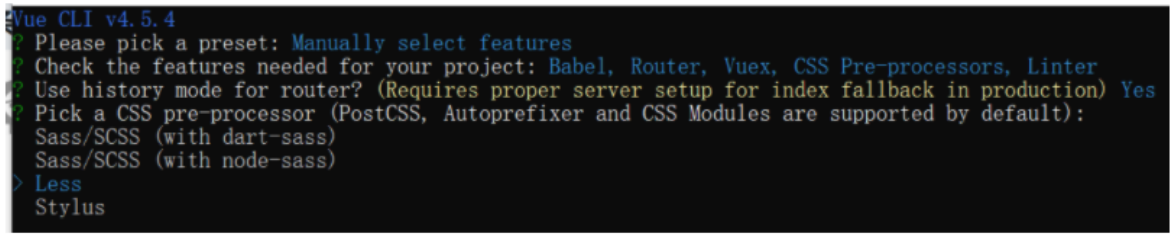
选择配置方式： default: 默认配置 Manually select features 手动配置

选中Manually select features后，再选择功能:

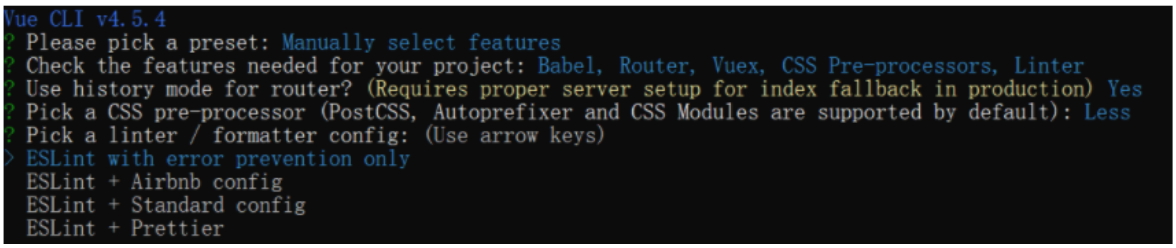


history mode，页面路由含不含有#;选择Y

选择CSS预编译器，这里我们选择Less

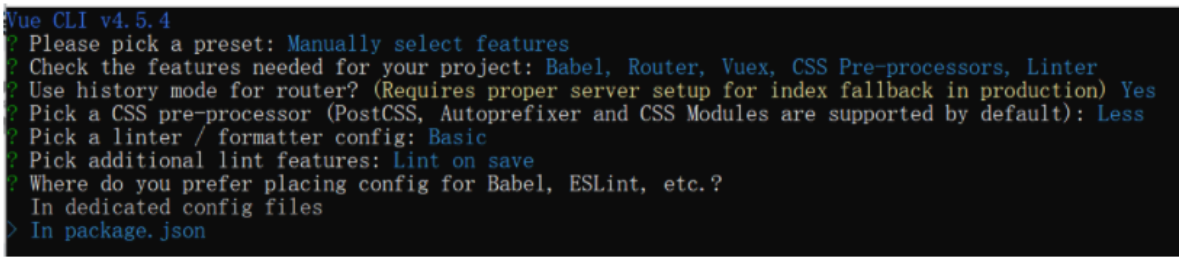


ESLint 代码校验规则



代码校验，Lint on save 保存就检查

配置保存到哪个文件中，选择存放到 package.json



插件

1.前端路由与Vue-Router

说明：简单来说前端路由就是页面的url,深入点说就是进行Restful请求时后端有一个正则匹配的路由列表，匹配到一条路径后发给相应的controller进行处理,处理完成之后把数据返回给前端。

为什么使用前端路由：

- 对比没有使用前端路由之前,浏览器要访问其他页面时点击之后需要进行加载各种资源请求,前端路由推荐缘由之一我们可以直接在后端服务器把所有数据渲染返回给浏览器,这样前端页面就不需要加载各种资源加载完之后在渲染页面了,加快了页面的获取响应的速度,增加用户体验。

（1）Vue-Router基本用法

- import阶段：导入vue,vue-router
- 使用Vue.use（vue-router）
-

```
const routes = [
  {
    path: "/",
    name: "Home",
    component: Home
  },
  {
    path: "/about",
    name: "About",
    // route level code-splitting
    // this generates a separate chunk (about.[hash].js) for this route
    // which is lazy-loaded when the route is visited.
    //()=>import("../views/About.vue")
(resolve)=>require(['../views/About.vue'],resolve) 都是懒加载-异步加载类似于
Vue.component（组件,(resolve,reject){内容}）
    component: require(['../views/About.vue'])

  }
```

```
];

const router = new vueRouter({
  routes:routes
});

export default router;
```

- **在main.js中导入,挂载到响应的实例中即可**

(2) 跳转

- router-link

```
<router-link to="路由"></router-link>
```

- button

```
<button @click="handlerRouter">About</button>
export default {
    methods:{
        handlerRouter(){this.$router.push('/About')} //replace
    }
}
```

(3) 高级用法

- 相关用法开发时请查看官方文档：vue-router

2.状态管理与VueX

(1) 状态管理与使用场景

Vue状态管理顾名思义即是维护组件状态,使我们在开发应用的时候可以更加灵活的操作数据。

使用场景：通常应用于大型单页应用,和多人协同开发。若项目比较小但是想要状态管理短期有效，那么你就有必要考虑是否需要使用Vue的状态管理

(2) Vuex基本用法

Vuex属性store,Mutations　　使用文档Vuex

(3) 高级用法

Getters，Actions,Modules　　高级使用文档

3.拓展

(1) 中央事件总线插件vue-bus

Iview经典组件

Iview文档