# vue

Vue.js

Vue (读音 /vjuː/，类似于 **view**) 是一套用于构建用户界面的**渐进式框架**。与其它大型框架不同的是，Vue 被设计为可以自底向上逐层应用。Vue 的核心库只关注视图层，不仅易于上手，还便于与第三方库或既有项目整合。

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="utf-8">  <title></title>  <script src="vue.js" type="text/javascript" charset="utf-8"></script>  </head>  <body>  <div id="app">  {{msg}}  </div>  <script type="text/javascript">  var app = new Vue({  el: '#app',  data: {  msg: 'hello world'  }  })  </script>  </body>  </html> |

绑定元素 attribute

|  |
| --- |
| <div id="app">  <span v-bind:title="msg">鼠标悬停试试</span>  </div>  <script type="text/javascript">  var app = new Vue({  el: '#app',  data: {  msg: new Date().toLocaleString()  }  })  </script> |
| v-bind attribute 被称为**指令**。指令带有前缀 v-，以表示它们是 Vue 提供的特殊 attribute。它们会在渲染的 DOM 上应用特殊的响应式行为。在这里，该指令的意思是：“将这个元素节点的 title attribute 和 Vue 实例的 message 属性保持一致”。 |

条件v-if/v-else-if/v-else,v-show

|  |
| --- |
| <body>  <div id="app">  <div v-if="type==='A'">A</div>  <div v-else-if="type==='B'">B</div>  <div v-else-if="type==='C'">C</div>  <div v-else>not A/B/C</div>  <h1 v-show="ok">hello</h1>  </div>  <script type="text/javascript">  var app = new Vue({  el: '#app',  data: {  type: "B",  ok: false  }  })  </script>  </body> |
| <h1 style="display: none;">hello</h1>  v-show 初始化成本高，切换成本低 |

循环v-for

|  |
| --- |
| <div id="app">  <ol>  <li v-for="todo in todos">  {{todo.text}}  </li>  </ol>  </div>  <script type="text/javascript">  var app = new Vue({  el: '#app',  data: {  todos: [{  text: '学习'  }, {  text: '运动'  }, {  text: '游戏'  }]  }  })  </script> |
| 控制台输入 app.todos.push({ text: '新项目' }) |

处理用户输入v-on

|  |
| --- |
| v-on 指令添加一个事件监听器 |
| <div id="app">  <p>{{msg}}</p>  <button type="button" v-on:click="reverseMsg">反转</button>  </div>  <script type="text/javascript">  var app = new Vue({  el: '#app',  data: {  msg:"hello vue.js"  },  methods: {  reverseMsg: function() {  this.msg = this.msg.split('').reverse().join('')  }  }  })  </script> |

v-model

|  |
| --- |
| v-model 指令实现表单输入和应用状态之间的双向绑定 |
| <div id="app">  <h1>{{msg}}</h1>  <input type="text" v-model="msg" />  <input type="text" :value="msg" />  </div>  <script>  var vm=new Vue({  el:'#app',  data:{  msg:'abc'  }  })  </script> |

组件化

组件系统是 Vue 的另一个重要概念，因为它是一种抽象，允许我们使用小型、独立和通常可复用的组件构建大型应用。

在 Vue 里，一个组件本质上是一个拥有预定义选项的一个 Vue 实例。

组件的使用

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>      <title></title>      <meta charset="UTF-8">      <link href="css/style.css" rel="stylesheet">      <script src="./vue.js"></script>  </head>  <body>      <div id="app">          <!-- 使用组件 -->          <cpn></cpn>          <cpn1></cpn1>          <cpn2></cpn2>      </div>      <script>          //创建组件          const cpn = Vue.extend({              template: `               <div>                  <h1>test Component</h1>               </div>`          })          //注册组件          Vue.component("cpn", cpn)          //语法糖形式          Vue.component("cpn1", {              template: `               <div>                  <h1>test Component1</h1>               </div>`          });          let vm = new Vue({              el: "#app",              //局部组件              components: {                  cpn2: {                      template: `               <div>                  <h1>test Component2</h1>               </div>`                  }              }          })      </script>  </body>  </html> |

模板抽离

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>      <title></title>      <meta charset="UTF-8">      <link href="css/style.css" rel="stylesheet">      <script src="./vue.js"></script>  </head>  <body>      <!-- 模板抽离 -->      <!-- 方法一 -->      <script type="text/x-template" id="cpn">          <h1>标题一</h1>      </script>      <!-- 方法二 -->      <template id="cpn1">          <h1>标题二</h1>      </template>      <div id="app">          <!-- 使用组件 -->          <cpn></cpn>          <cpn1></cpn1>      </div>      <script>          Vue.component("cpn", { template: "#cpn" });          let vm = new Vue({              el: "#app",              components: {                  cpn1: {                      template: "#cpn1"                  }              }          })      </script>  </body>  </html> |

父子组件传值

父传子props

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">    <head>      <script src="./vue.js"></script>    </head>    <body>      <template id="child">        <ul>          <li v-for="item in childdata">{{item}}</li>        </ul>      </template>      <div id="app">         <child :childdata="parentdata"></child>      </div>      <script>        let vm = new Vue({          el: "#app",          data: {            msg: "xxx",            parentdata: ["xxx", "qqq", "zzz"],          },          components: {            child: {              template: "#child",              props: {                childdata: {                  type: Array,                  //default: ["...", "\*\*\*"],                  //required: true,                },              },            },          },        });      </script>    </body>  </html> |
|  |

子传父$emit

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <script src="./vue.js"></script>  </head>  <body>    <template id="child">      <div>        <button v-for="u in users" @click="childclk(u)">{{u.name}}</button>      </div>    </template>    <div id="app">      <!-- 监听子组件的childclk事件，触发parentMethod方法调用 -->      <child @childclk="parentMethod"></child>    </div>    <script>      const child = {        template: "#child",        data() {          return { users: [{ id: '1001', name: 'zs' }, { id: '1002', name: 'ls' }] }        },        methods: {          childclk(u) {            //向父组件发射childclk事件            this.$emit("childclk", u)          }        }      };      let vm = new Vue({        el: "#app",        methods: {          parentMethod(u) {            console.log(u);          }        },        components: {          child,        },      });    </script>  </body>  </html> |

router

|  |
| --- |
| <script src="https://unpkg.com/vue/dist/vue.js"></script>  <script src="https://unpkg.com/vue-router/dist/vue-router.js"></script>  <div id="app">  <h1>Hello App!</h1>  <p>  <!-- 使用 router-link 组件来导航. -->  <!-- 通过传入 `to` 属性指定链接. -->  <!-- <router-link> 默认会被渲染成一个 `<a>` 标签 -->  <router-link to="/foo">Go to Foo</router-link>  <router-link to="/bar">Go to Bar</router-link>  </p>  <!-- 路由出口 -->  <!-- 路由匹配到的组件将渲染在这里 -->  <router-view></router-view>  </div>  <script>  // 0. 如果使用模块化机制编程，导入Vue和VueRouter，要调用 Vue.use(VueRouter)  // 1. 定义 (路由) 组件。  // 可以从其他文件 import 进来  const Foo = { template: '<div>foo</div>' }  const Bar = { template: '<div>bar</div>' }  // 2. 定义路由  // 每个路由应该映射一个组件。 其中"component" 可以是  // 通过 Vue.extend() 创建的组件构造器，  // 或者，只是一个组件配置对象。  const routes = [  { path: '/foo', component: Foo },  { path: '/bar', component: Bar }  ]  // 3. 创建 router 实例，然后传 `routes` 配置  const router = new VueRouter({  routes // (缩写) 相当于 routes: routes  })  // 4. 创建和挂载根实例。  // 记得要通过 router 配置参数注入路由， 从而让整个应用都有路由功能  const app = new Vue({  router  }).$mount('#app')  // 现在，应用已经启动了！  </script> |

axios

Axios 是一个基于 promise 的 HTTP 库，可以用在浏览器和 node.js 中。

|  |
| --- |
| <script>  {  var fromSeverToken='xxxxx.xxx.ss';  window.sessionStorage.setItem("accessToken",fromSeverToken)  }  {  axios.defaults.baseURL = "http://localhost:8088/test/";  Vue.prototype.$http = axios;  }  {  axios.interceptors.request.use(function(config){  //在请求发出之前进行一些操作  let token=window.sessionStorage.getItem("accessToken")  if(token)  {  config.headers.accessToken=token  }  return config;  },function(err){  return Promise.reject(err);  });  }  var vm=new Vue({  el:'#app',  data:{  msg:'...'  },  methods:{  getUserList()  {  // this.$http.get('getUsers').then(  // rs=>console.log(rs)  // )  this.getUs()  },  async getUs()  {  const {data:rs}= await this.$http.get('getUsers')  console.log(rs)  }  }  })  </script> |
|  |
|  |
|  |

脚手架

|  |
| --- |
| 安装node.js  安装 cnpm npm（node package manager）是nodejs的包管理器  cnpm：安装npm镜像，以后再用到npm的地方直接用cnpm来代替  npm install -g cnpm --registry=https://registry.npm.taobao.org  cnpm -v  安装vue-cli脚手架构建工具  cnpm install vue  npm uninstall vue-cli -g  cnpm install -g @vue/cli  vue --version  创建项目  vue create demo  cd demo  npm run serve  node官网下载最新版msi  where node 得到上次安装的路径  打开msi把nodejs安装到上次的路径  npm install -g npm@latest |
|  |

vueUi

|  |
| --- |
| vueUi |
| win+r cmd 注：可以切换目录至源码存放位置  vue ui  点击创建 |
|  |
|  |
| 功能中选：babel、router、linter/fromatter、**使用配置文件** |
|  |
|  |
|  |
| 安装element-ui插件 |
| 点击头像选中 |
|  |
| 安装axios---运行依赖 |
| 安装less 样式定义---开发依赖 |
|  |
|  |
|  |
|  |
| 清理 |
|  |
|  |
|  |
| 关闭一些默认校验 |
| 模拟登录跳转 |
| 首页布局 |
| 导航 |
| 表格数据加载 |
| **package** com.yc.base.testVue;  **import** org.springframework.context.annotation.Bean;  **import** org.springframework.context.annotation.Configuration;  **import** org.springframework.web.servlet.config.annotation.CorsRegistry;  **import** org.springframework.web.servlet.config.annotation.WebMvcConfigurer;  @Configuration  **public** **class** GlobalCorsConfig {  @Bean  **public** WebMvcConfigurer corsConfigurer() {  **return** **new** WebMvcConfigurer() {  // 重写父类提供的跨域请求处理的接口  @Override  **public** **void** addCorsMappings(CorsRegistry registry) {  registry.addMapping("/\*\*").allowedOriginPatterns("\*")  .allowedMethods("GET", "HEAD", "POST", "PUT", "DELETE", "OPTIONS").allowCredentials(**true**)  .maxAge(3600).allowedHeaders("\*");  }  };  }  } |
|  |
| Mock  生成数据，拦截AJAX请求  --文档<http://mockjs.com/>  cnpm install --save mockjs |
|  |
|  |

vue脚手架项目里引用JQuery包

第一步

在项目根下package.json里的dependencies加入一行

"jquery" : "^3.2.1"

引用jQuery包

vue模板

文件-->首选项-->用户代码片段-->点击新建代码片段--取名vue.json 确定

|  |
| --- |
| {"vue模板":  {  "prefix":"vue",  "body":[  "<template>",  "<div></div>",  "</template>",  "",  "<script>",  "export default {",  "//import 引入的组件需要注入到对象中才能使用",  "components:{},",  "props:{},",  "data(){",  "//这里存数据",  "return{};",  "},",  "//计算属性",  "computed: {",  "",  "},",  "//监控data中数据变化",  "watch: {",  "",  "},",  "//方法",  "methods: {",  "",  "},",  "//声明周期 - 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() {",  "",  "},",  "beforeCreate() {},//生命周期 - 创建之前",  "beforeMount() {},//生命周期 - 挂载之前",  "beforeUpdate() {},//声明周期 - 更新之前",  "updated() {},//生命周期 - 更新之后",  "beforeDestroy() {},//生命周期 - 销毁之前",  "destroyed() {},//生命周期 - 销毁之后",  "activated() {},//缓存keep-alive",  "};",  "</script>",  "",  "<style scoped>",  "</style>",  ],  "description":"生成vue模板"  }  } |
| "http-get请求": {  "prefix": "httpget",  "body": [  "this.\\$http({",  "url: this.\\$http. adornUrl(''),",  "method: 'get',",  "params: this.\\$http.adornParams({})",  "}).then(({data})=>{",  "})"  ],  "description": "httpGet请求"  },  "http-post请求": {  "prefix": "httppost",  "body": [  "this.\\$http({",  "url: this.\\$http.adornUrl(''),",  "method: 'post',",  "data: this.\\$http.adornData(data,false)",  "}).then(({data})=>{});"  ],  "description": "httpPOST请求"  } |

Vue ui