vue 知识点

搭建 vue 环境

（1）安装 nodejs　　https://nodejs.org

注：下载 nodejs，安装的时候，点击同意，一路 next 就可以，会自带一个 npm（npm 是个包管理器，有啥用？是个仓库，需要用到啥 直接 npm install packageName 就可以了）

（2）安装 cnpm

方法一：npm install -g cnpm --registry=https://registry.npm.taobao.org

方法二：npm install -g cnpm

（3）安装 vue-cli（安装脚手架：（注：只安装一次））

方法：npm install vue-cli -g

搭建 vue 项目

（1）基于 webpack 生成模板项目: vue init webpack 项目名字

注：安装过程中的解释

（1）project name （项目名称）可以选择 y 或者自己起一个

（2）Project descrption （项目描述） 可以直接选择回车或者添加描述

（3）Author （作者）可以直接选择回车或者添加作者

（4）运行施加编译的安装包（回车）

（5）Install vue-router （是否安装路由） 回车

（6）Use ESLint to your code （是否进行代码检查或规范）

最后两步是否安装测试环境（直接选择 no）

（2）安装 package.json 里的依赖:npm install

（3）运行:npm run dev

目录结构介绍

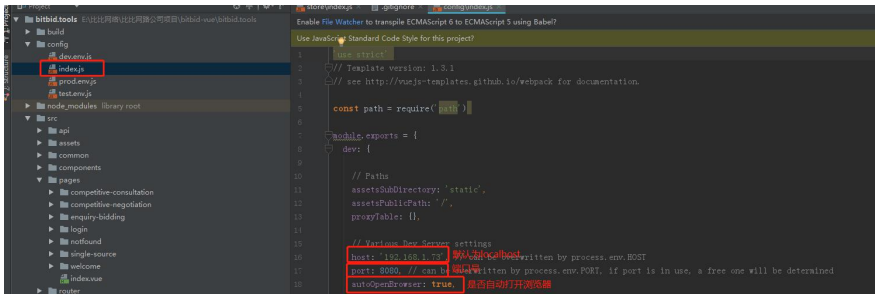
（1）index.html: 首页入口文件，你可以添加一些 meta 信息或统计代码。

（2）package.json:项目配置文件。

（3）.gitignore　忽略的文件

（4）static:静态资源目录，如图片、字体等。

（5）config:配置目录，包括端口号等。我们初学可以使用默认的。



（6）build:最终发布的代码存放位置。

（7）node_modules:npm 加载的项目依赖模块

（8）src:这里是我们要开发的目录，基本上要做的事情都在这个目录里。里面包含了几个目录及文件：assets: 放置一些图片，如 logo 等。

components: 目录里面放了一个组件文件，可以不用。

App.vue: 项目入口文件，我们也可以直接将组件写这里，而不使用 components 目录。

main.js: 项目的核心文件。

为什么选 Vue 进行项目开发

Vue 是一个以数据驱动为核心的渐进式 MVVM 模式框架。采用了虚拟 DOM，在页面的渲染上性能要好一些。同时他还是一个高性能，轻巧，可组件化的库，并且拥有非常易用上手的 API 库。

vue 的两个核心是什么？

数据驱动和组件化

什么是 vue 生命周期

Vue 实例从创建到销毁的过程，从开始创建，初始化数据，编译模板，挂载 Dom——>渲染，更新->渲染，销毁等一系列过程，称为 Vue 的生命周期

vue 生命周期的作用是什么？

它的生命周期中有多个事件钩子，让我们在控制整个 Vue 实例的过程时更容易形成好的逻辑。

vue 生命周期总共有几个阶段？

它可以总共分为 8 个阶段：创建前/后，载入前/后,更新前/后,销毁前/销毁后

第一次页面加载会触发哪几个钩子？

第一次页面加载时会触发 beforeCreate, created, beforeMount, mounted 这几个钩子

DOM 渲染在 哪个周期中就已经完成？

DOM 渲染在 mounted 中就已经完成了。

简单描述每个周期具体适合哪些场景？

beforecreate : （创建前） 在数据观测和初始化事件还未开始 （可以在这加个 loading 事件，在加载实例时触发）

created :（创建后） 完成数据观测，属性和方法的运算（ 初始化完成时的事件写在这里，如在这结束 loading 事件，异步请求也适宜在这里调用）

beforeMount : （载入前） 在挂载开始之前被调用，相关的 render 函数首次被调用。

mounted : （载入后）挂载元素，获取到 DOM 节点

beforeUpdate（更新前） 在数据更新之前调用，发生在虚拟 DOM 重新渲染和打补丁之前。

updated : （更新后） 在由于数据更改导致的虚拟 DOM 重新渲染和打补丁之后调用。调用时，组件 DOM 已经更新，所以可以执行依赖于 DOM 的操作。（如果对数据统一处理，在这里写上相应函数）

beforeDestroy（销毁前） 在实例销毁之前调用。实例仍然完全可用。（可以做一个确认停止事件的确认框 ）

destroyed（销毁后） 在实例销毁之后调用。调用后，所有的事件监听器会被移除，所有的子实例也会被销毁。该钩子在服务器端渲染期间不被调用。

nextTick : 更新数据后立即操作 dom

created 和 mounted 的区别

created:在模板渲染成 html 前调用，即通常初始化某些属性值，然后再渲染成视图。mounted:在模板

渲染成 html 后调用，通常是初始化页面完成后，再对 html 的 dom 节点进行一些需要的操作

vue 获取数据在哪个周期函数

一般在 created(或 beforeRouter) 里面就可以，如果涉及到需要页面加载完成之后的话就用 mounted

注：在 created 的时候，视图中的 html 并没有渲染出来，所以此时如果直接去操作 html 的 dom 节点，一定找不到相关的元素而在 mounted 中，由于此时 html 已经渲染出来了，所以可以直接操作 dom 节点

mvvm 框架是什么

MVVM 是 Model-View-ViewModel 的简写。即模型-视图-视图模型。【模型】指的是后端传递的数据。

【视图】指的是所看到的页面。【视图模型】mvvm 模式的核心，它是连接 view 和 model 的桥梁

注：MVVM 模式的三款框架：Vue，Angular，Avalon

1：Vue 是尤雨溪老师开发的，不支持 IE8，是一个新生儿，2014 年才发布没有 Angular 那么成熟，影响程度不是很大。但是简单易学官方文档很清晰，强大，对模块很友好，能够快速的更新 DOM

2:Angular：是一个比较完善的前端 MVW 框架，包含模板，数据双向绑定，路由，模块化，服务，依赖注入等所有功能，模板功能强大丰富，并且是声明式的，自带了丰富的 Angular 指令。但是太大又很全面学习起来就比较困难。同样不支持 IE8

3：Avalon 使用简单，没有任何依赖，兼容 IE6。自带 AMD 模块加载器，省得与其他加载器进行整合。

vue-router 是什么?它有哪些组件

Vue Router 是 Vue.js 官方的路由管理器

组件有：`<router-view></router-view>`

```
<router-link to="/xxx"></router-link> 等
```

active-class 是哪个组件的属性

vue-router 模块的 router-link 组件中的属性，用来做选中样式的切换

注：1、直接在路由 js 文件中配置 linkActiveClass

2、在 router-link 中写入 active-class

分别简述 computed 和 watch 的区别

1、功能上：computed 是计算属性，watch 是监听一个值的变化，然后执行对应的回调。

2、是否调用缓存：computed 中的函数所依赖的属性没有发生变化，那么调用当前的函数的时候会从缓存中读取，而 watch 在每次监听的值发生变化的时候都会执行回调。

3、是否调用 return：computed 中的函数必须要用 return 返回，watch 中的函数不是必须要用 return。

4、使用场景：computed----当一个属性受多个属性影响的时候，使用 computed-------购物车商品结算。watch----当一条数据影响多条数据的时候，使用 watch-------搜索框。

怎么定义 vue-router 的动态路由？怎么获取传过来的动态参数？

在 router 下的 index.js（或者自己配的路由里边），对 path 属性加上/:id，获取方式是 this.$route.params.id

组件之间的传值？

父组件与子组件传值：　1、父组件通过标签上面定义传值
2、子组件通过 props 方法接受数据

子组件向父组件传递数据：1、子组件通过$emit 方法传递参数

例：子组件内：this.$emit("自定义名字"，传递的内容)

父组件：　在组件上定义@upLoadFile="upLoadFile"，然后在 methods 中调用

注：数据类型传递限制：在子组件中 props 中对象定义传递的数据类型

vue 中 v-if 和 v-show 的区别

相同点：v-show 和 v-if 都能控制元素的显示和隐藏

不同点：V-if 当不满足条件时，该元素消失，代码也会消失，相当于将代码删除了，当满足条件时，该元素显示，页面会重新渲染该元素

v-if 指令是直接销毁和重建 DOM 达到让元素显示和隐藏的效果

V-show 是控制的隐藏显示，只是将 css 属性设为 display: none 或 block

v-show 指令是通过修改元素的 display 的 CSS 属性让其显示或者隐藏

v-if 有更高的切换开销，而 v-show 有更高的初始渲染开销。因此，如果需要非常频繁地切换，则使用 v-show 较好；如果在运行时条件很少改变，则使用 v-if 较好。

V-html 和 v-text 相同和不同点

相同点：都是渲染文本

不同点：v-html：html 标签在渲染的时候被解析

v-text： html 标签在渲染的时候被源码输出

v-model 是什么？

v-model 是 Vue 用于表单元素上创建双向数据绑定，监听用户输入事件并更新数据的功能

如何让 CSS 只在当前组件中起作用

将当前组件的<style>修改为<style scoped>

this.set()方法的作用

实现实时更新视图数据 用法：this.$set(this.data,”key”,value’）

注：this.data 要操作的数组或者对象

”key” 设置的属性

value 设置的值

为什么避免 v-if 和 v-for 用在一起

当 Vue 处理指令时，v-for 比 v-if 具有更高的优先级，通过 v-if 移动到容器元素，不会再重复遍历列表中的每个值。取而代之的是，我们只检查它一次，且不会在 v-if 为否的时候运算 v-for。

说出至少 4 种 vue 当中的指令和它的用法？

v-if：判断是否隐藏；v-for：数据循环；v-bind:class：绑定一个属性；v-model：实现双向绑定

为什么使用 key？

key 的作用就是为了快速的找到新节点对应的旧节点。key 是给每一个 vnode 唯一的 id,可以依靠 key,更准确，更快的拿到 oldVnode 中对应的 vnode 节点

Vue 组件中包括三个部分：

template：视图　　Script：逻辑　　Style：样式

vue 常用的修饰符

a、.delete

.delete（捕获"删除"和"退格"键）　　用法上和事件修饰符一样，挂载在 v-on:后面，语法：

```
v-on:keyup.xxx=’yyy’　　<inputclass = ’aaa’ v-model="inputValue" @keyup.delete="onKey"/>
```

b、.trim

如果要自动过滤用户输入的首尾空白字符，可以给 v-model 添加 trim 修饰符

C、.stop

用来阻止单击事件的冒泡

按键修饰符

d、.entrer

```
@keyup.enter.native="logonFormbtn('logonForm')"
```

vue 等单页面应用及其优缺点

优点：1、具有桌面应用的即时性、网站的可移植性和可访问性。

2、用户体验好、快，内容的改变不需要重新加载整个页面。

3、基于上面一点，SPA 相对对服务器压力小。

4、良好的前后端分离。SPA 和 RESTful 架构一起使用，后端不再负责模板渲染、输出页面工作，web 前端和各种移动终端地位对等，后端 API 通用化。

5、同一套后端程序代码，不用修改就可以用于 Web 界面、手机、平板等多种客户端；

缺点：1、不利于 SEO。（如果你看中 SEO，那就不应该在页面上使用 JavaScript，你应该使用网站而不是 Web 应用）

2、初次加载耗时相对增多。

vuex 是什么？怎么使用？哪种功能场景使用它？

vuex 是：vue 框架中状态管理。

使用方法：在 main.js 引入 store，注入。新建一个目录 store，….. export 。

场景有：单页应用中，组件之间的状态。音乐播放、登录状态、加入购物车

不用 Vuex 会带来什么问题？

可维护性会下降，想修改数据要维护三个地方；

可读性会下降，因为一个组件里的数据，根本就看不出来是从哪来的；

增加耦合，大量的上传派发，会让耦合性大大增加，本来 Vue 用 Component 就是为了减少耦合，现在这么用，和组件化的初衷相背。

vue 脚手架（vue-cli）的作用

vue-cli 作为 vue 的脚手架，可以帮助我们在实际开发中自动生成 vue.js 的模板工程。

vuex 有哪几种属性？

有五种，分别是 State、 Getter、Mutation 、Action

一：vuex 的 State 特性

A、Vuex 就是一个仓库，仓库里面放了很多对象。其中 state 就是数据源存放地，对应于一般 Vue 对象里面的 data

B、state 里面存放的数据是响应式的，Vue 组件从 store 中读取数据，若是 store 中的数据发生改变，依赖这个数据的组件也会发生更新

C、它通过 mapState 把全局的 state 和 getters 映射到当前组件的 computed 计算属性中

二、vuex 的 Getter 特性

A、getters 可以对 State 进行计算操作，它就是 Store 的计算属性

B、 虽然在组件内也可以做计算属性，但是 getters 可以在多组件之间复用

C、 如果一个状态只在一个组件内使用，是可以不用 getters

三、vuex 的 Mutation 和 Action 特性

Action 类似于 mutation，不同在于 :Action 提交的是 mutation，而不是直接变更状态；Action 可以包含任意异步操作。

<keep-alive></keep-alive>的作用是什么？

<keep-alive></keep-alive> 包裹动态组件时，会缓存不活动的组件实例，主要用于保留组件状态或避免重新渲染（不理解）

例子：https://www.cnblogs.com/sysuhanyf/p/7454530.html

vue-router 有哪几种导航钩子？

有三种方式可以植入路由导航过程中： 全局的 单个路由独享的 组件级的

第一种：全局导航钩子（全局导航钩子主要有两种钩子：前置守卫、后置钩子，）

router.beforeEach(to,from,next)，作用：跳转前进行判断拦截。

例：//定义一个路由

```
const router = new VueRouter({ ... })
// 点击导航前调用 （前置守卫）
router.beforeEach((to, from, next) => { // ... })
// 点击导航后调用 （后置钩子）
router.afterEach(route => { // ... })
```

当一个导航触发时，全局的 before 钩子按照创建顺序调用。钩子是异步解析执行，此导航在所有钩子 resolve 完之前一直处于 等待中。

每个钩子方法接收三个参数：

1.to: Route: 即将要进入的目标 路由对象

2.from: Route: 当前导航正要离开的路由

3.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。

第二种：单独路由独享组件

例：const router = new VueRouter({

```
routes: [
 {
    path: '/foo',
    component: Foo,
    beforeEnter: (to, from, next) => {
     // ... }
    }
 ]
});
```

第三种：组件内的钩子

```
const Foo = {
    template: `<div>This is file</div>`,
     beforeRouteEnter (to, from, next) {
    // 在渲染该组件的对应路由被 confirm 前调用
    // 不能！获取组件实例 `this`
    // 因为当钩子执行前，组件实例还没被创建
    },
    beforeRouteUpdate (to, from, next) {
    // 在当前路由改变，但是该组件被复用时调用
    // 举例来说，对于一个带有动态参数的路径 /foo/:id，在 /foo/1 和 /foo/2
    间跳转的时候，
    // 由于会渲染同样的 Foo 组件，因此组件实例会被复用。而这个钩子就会在这个情
    况下被调用。
    // 可以访问组件实例 `this`
    },
     beforeRouteLeave (to, from, next) {
    // 导航离开该组件的对应路由时调用
    // 可以访问组件实例 `this`
    }
}
```

完整的导航解析流程：

1.导航被触发

2.在失活的组件里调用离开守卫

3.调用全局的 beforeEach 守卫

4.在重用的组件里调用 beforeRouteUpdate 守卫

5.在路由配置里调用 beforEnter

6.解析异步路由组件

7.在被激活的组件里调用 beforeRouteEnter

8.调用全局的 beforeResolve 守卫

9.导航被确认

10.调用全局的 afterEach 钩子

11.触发 DOM 更新

12.在创建好的实例调用 beforeRouteEnter 守卫中传给 next 的回调函数

Vue 工作中遇到的问题

vue 单页应用中如何使用 jquery 的方法示例

1.首选通过 npm 安装 jquery （npm install jquery --save）

2.在 build/webpack.base.conf 文件当中引入 jquery

```
module.exports = {
  resolve: {
    extensions: ['.js', '.vue', '.json'],
    alias: {
      'vue$': 'vue/dist/vue.esm.js',
      '@': resolve('src'),
      'jquery': path.resolve(__dirname, '../node_modules/jquery/src/jquery'),
      // 静态资源目录
      'static': resolve('static')
    }
  },
}
```

3. 在需要的地方 （import $ from 'jquery'）

element el-table 表格列 v-if 显示 隐藏 乱序问题

解决方法：在 table-column 中加入:key="Math.random()"

如何利用 Vue 实现页面的局部刷新

例子：https://github.com/15234477664/Vue-reload

1. 首先需要修改 App.vue

```
<template>
    <div id="app">
        <router-view v-if="isRouterAlive"></router-view>
    </div>
</template>
<script>
export default {
  name: "app",
  provide() {
    return {
      reload: this.reload
    };
  },
  data() {
    return {
      isRouterAlive: true
    };
  },
  methods: {
    reload() {
      this.isRouterAlive = false;
      this.$nextTick(function() {
        this.isRouterAlive = true;
      })
    }
  }
};
</script>
```

2. 到需要刷新的页面进行引用，使用 inject 导入引用 reload，然后直接调用即可

```
.then(response => {
  response = response.data;
  if (response.code == 200) {
    this.addVisible = false;
    this.reload();
    this.$message.success(`添加成功`);
  } else {
```



```
export default {
  inject: ["reload"],
  data() {
    return {
```


vue 中表格验证

链接
https://github.com/15234477664/From-Table/blob/master/README.md

```
<template>
 <div class="app-container">
   <div class="filter-container">
     <el-button type="primary" @click="init()" icon="el-icon-circle-plus">add</el-button>
   </div>
   <el-dialog title="表单Table" :visible.sync="dialogFormVisible">
     <el-form :model="fromData" ref="from">
       <el-table :data="fromData.domains">
         <el-table-column label="姓名">
           <template slot-scope="scope">
             <el-form-item :prop="'domains.'+scope.$index+'.name'" :rules="fromaDataRules.name">
               <el-input v-model="scope.row.name"></el-input>
             </el-form-item>
           </template>
         </el-table-column>
         <el-table-column label="地址">
           <template slot-scope="scope">
             <el-form-item :prop="'domains.'+scope.$index+'.desc'" :rules="fromaDataRules.desc">
               <el-input v-model="scope.row.desc"></el-input>
             </el-form-item>
           </template>
         </el-table-column>
       </el-table>
     </el-form>
     <el-button type="warning" @click="submit('from')">submit</el-button>
   </el-dialog>
 </div>
</template>
```

复制

```
<script>
  export default {
    data() {
      return {
        dialogFormVisible:false,
        fromData:{
          domains:undefined
        },
        fromaDataRules:{
          name:[{ required: true, message: '请输入活动名称', trigger: 'blur' }],
          desc:[ { required: true, message: '请填写活动形式', trigger: 'blur' }]
        },
        domains:[],
      }
    },
```

复制

vue　放大镜

链接地址：https://github.com/15234477664/vue-piczoom

链接地址：https://github.com/lemontree2000/vue-magnify　（兼容 ie9 ,用起来效果不太好）

vue 中 for 循环出的加验证

链接地址：https://www.jb51.net/article/142750.htm（或 github 上）

ie 9 文件上传

链接地址：https://github.com/15234477664/vue-upload-web

数组去重（整合）

方法一：(普通方法)

```
let indexArray = []
this.heavyArray.map((its) => {
  this.accountTableData.map((item, index) => {
    if (item.id === its.id) {
      if (indexArray.indexOf(index) === -1) {
        indexArray.push(index)
      }
    }
  })
})
let index = 0
indexArray.map((item) => {
  item -= index
  this.accountTableData.splice(item, 1)
  index++
})
this.accountTableData.concat(this.heavyArray)
```

方法二：

```
some 方法
this.multipleSelection.map((item) => {
  if (!this.associatedData.some((value) => {
    return value.id === item.id
```

```
  })) {
    tempArry.push(item)
  }
})
this.associatedData = this.associatedData.concat(tempArry)
```

方法三：

```
every 方法

this.multipleSelection.map((item) => {
  if (this.associatedData.every((value) => {
    return value.id !== item.id
  })) {
    tempArry.push(item)
  }
})
this.associatedData = this.associatedData.concat(tempArry)
let tempArry = []
this.multipleSelection.map((item) => {
  if (this.associatedData.indexOf(item) === -1) {
    tempArry.push(item)
  }
})
this.associatedData = this.associatedData.concat(tempArry)
```

方法四：

```
set 实现数组去重
let array = Array.from(new Set([1, 1, 1, 2, 3, 2, 4]));
console.log(array);

ES5 实现数组去重
var array = [1, '1', 1, 2, 3, 2, 4];
var tmpObj = {};
var result = [];
array.forEach(function(a) {
  var key = (typeof a) + a;
  if (!tmpObj[key]) {
    tmpObj[key] = true;
    result.push(a);
  }
});
```